Scheduling Algorithms for Heterogeneous Platforms

Yves Robert

Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon, France

Yves.Robert@ens-lyon.fr

http://graal.ens-lyon.fr/~yrobert

HiPC Tutorial – December 21, 2005

Scheduling Algorithms for Heterogeneous Platforms

Yves Robert

Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon, France

Yves.Robert@ens-lyon.fr

http://graal.ens-lyon.fr/~yrobert

HiPC Tutorial – December 21, 2005

Outline

Outline

Evolution of parallel machines

An ever-increasing demand of computing power

Evolution of parallel machines

An ever-increasing demand of computing power
Parallelism is an attempt to answer

Unity is strength

Evolution of parallel machines

An ever-increasing demand of computing power
Parallelism is an attempt to answer

Unity is strength

Evolution of parallel machines

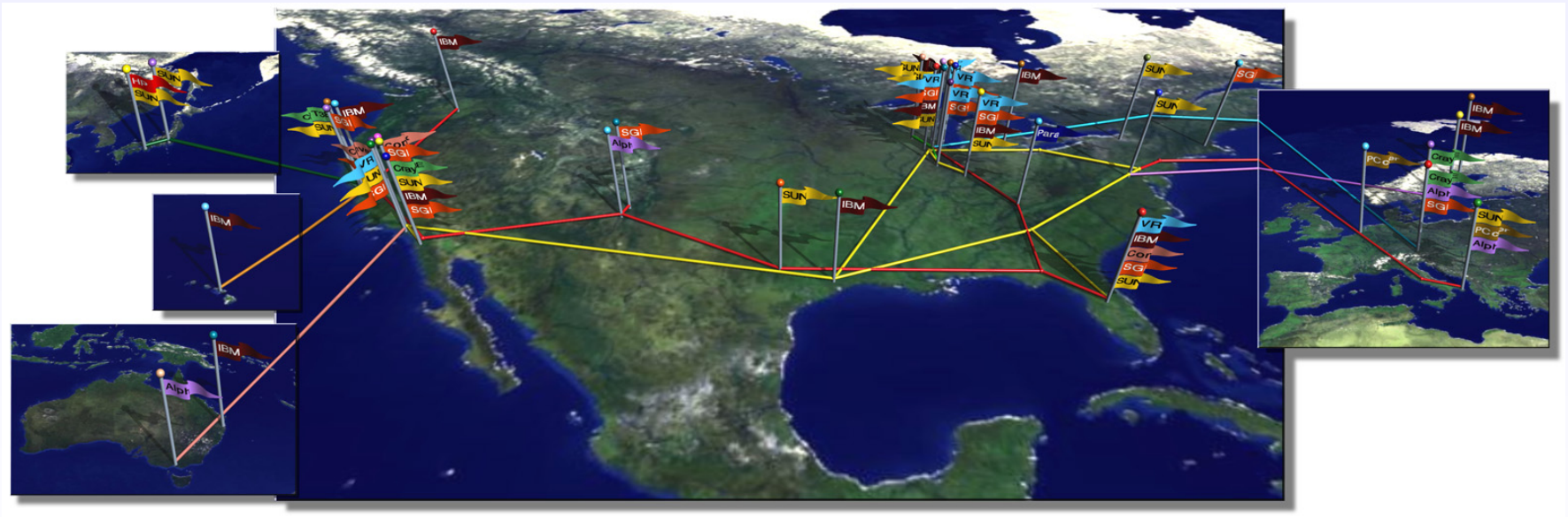
An ever-increasing demand of computing power
Parallelism is an attempt to answer

Unity is strength

Evolution of parallel machines

An ever-increasing demand of computing power
Parallelism is an attempt to answer

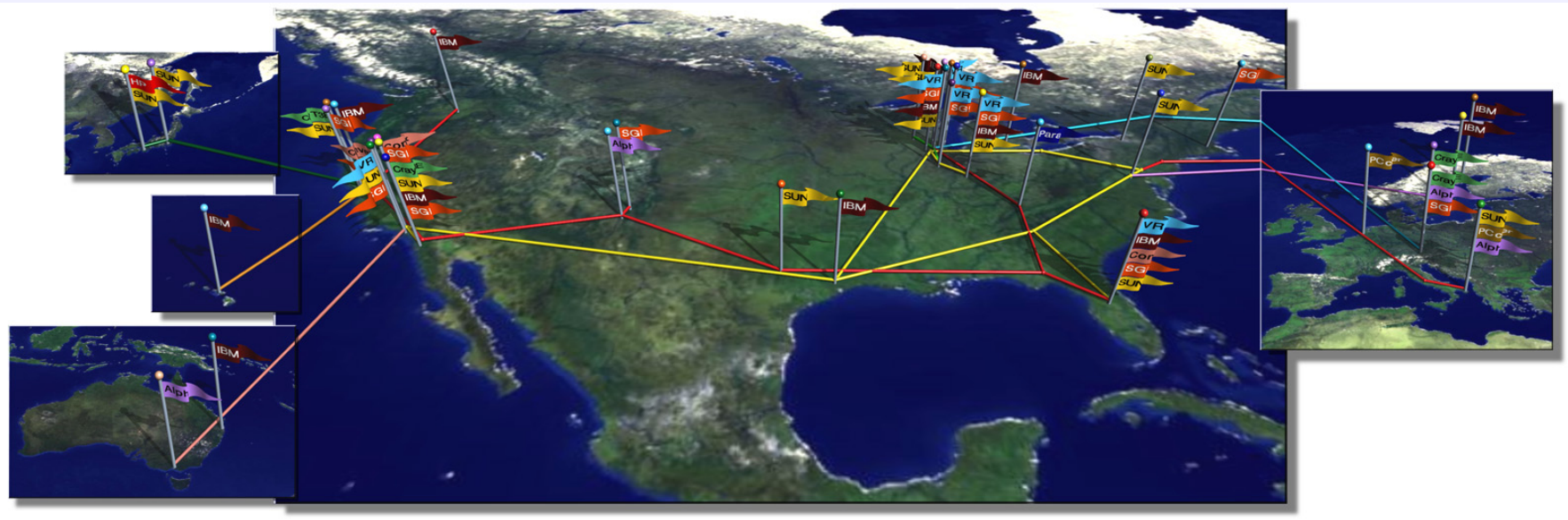
Unity is strength



Evolution of parallel machines

An ever-increasing demand of computing power
Parallelism is an attempt to answer

Unity is strength

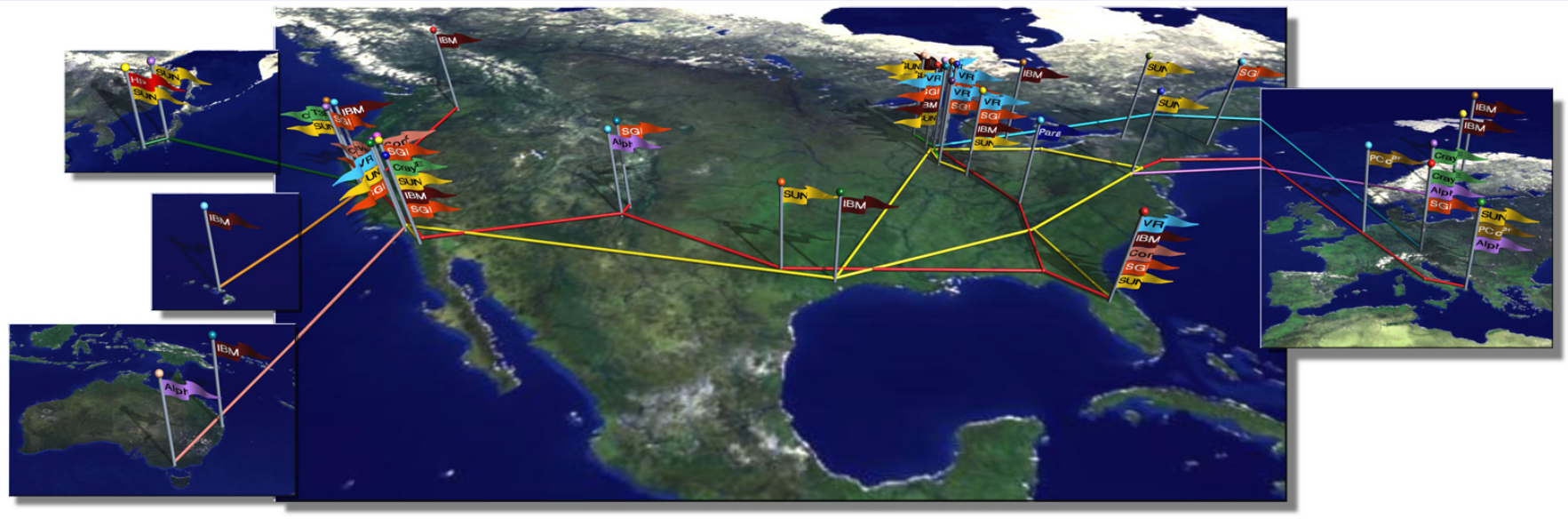


Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines

Evolution of parallel machines

An ever-increasing demand of computing power
Parallelism is an attempt to answer

Unity is strength



Parallel algorithm design and scheduling were already difficult tasks with homogeneous machines

On heterogeneous platforms, it gets worse

New platforms, new problems, new solutions

Target platforms: Large-scale heterogenous platforms
(networks of workstations, clusters, collections of clusters, grids, ...)

New problems

- Heterogeneity of processors (CPU power, memory)
- Heterogeneity of communication links
- Irregularity of interconnection network
- Non-dedicated platforms

Need to adapt algorithms and scheduling strategies: new objective functions, new models

New platforms, new problems, new solutions

Target platforms: Large-scale heterogenous platforms
(networks of workstations, clusters, collections of clusters, grids, ...)

New problems

- Heterogeneity of processors (CPU power, memory)
- Heterogeneity of communication links
- Irregularity of interconnection network
- Non-dedicated platforms

Need to adapt algorithms and scheduling strategies: new objective functions, new models

New approaches

1. Divisible load model Assume work can be arbitrarily divided into sub-tasks

2. Steady-state throughput Rather than optimizing execution from the very beginning to the very end, optimize execution core instead

Both approaches are relaxation methods: simplifying model allows to tackle more complex problems

Outline

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - $\text{alloc}(T)$ = processor assigned to it

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - $\text{alloc}(T)$ = processor assigned to it

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - $\text{alloc}(T)$ = processor assigned to it

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - alloc(T) = processor assigned to it

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - alloc(T) = processor assigned to it

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - $\text{alloc}(T)$ = processor assigned to it

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - $\text{alloc}(T)$ = processor assigned to it

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - alloc(T) = processor assigned to it

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - alloc(T) = processor assigned to it

Traditional scheduling – Framework

- Application = DAG $G = (\mathcal{T}, E, w)$
  - $\mathcal{T}$ = set of tasks
  - E = dependence constraints
  - $w(T)$ = computational cost of task T (execution time)
  - $c(T, T')$ = communication cost (data sent from T to T')
- Platform
  - Set of p identical processors
- Schedule
  - $\sigma(T)$ = date to begin execution of task T
  - alloc(T) = processor assigned to it

Traditional scheduling – Constraints

- Data dependences If $(T, T') \in E$ then
  - if $\text{alloc}(T) = \text{alloc}(T')$ then

$$\sigma(T) + w(T) \leqslant \sigma(T')$$

  - if $\text{alloc}(T) \neq \text{alloc}(T')$ then

$$\sigma(T) + w(T) + c(T, T') \leqslant \sigma(T')$$

- Resource constraints

$$\text{alloc}(T) = \text{alloc}(T') \Rightarrow$$
$$[\sigma(T), \sigma(T) + w(T)[\bigcap [\sigma(T'), \sigma(T') + w(T')[= \emptyset$$

Traditional scheduling – Constraints

- Data dependences If $(T, T') \in E$ then
  - if $\text{alloc}(T) = \text{alloc}(T')$ then

$$\sigma(T) + w(T) \leqslant \sigma(T')$$

  - if $\text{alloc}(T) \neq \text{alloc}(T')$ then

$$\sigma(T) + w(T) + c(T, T') \leqslant \sigma(T')$$

- Resource constraints

$$\text{alloc}(T) = \text{alloc}(T') \Rightarrow$$
$$[\sigma(T), \sigma(T) + w(T)[\bigcap [\sigma(T'), \sigma(T') + w(T')[= \emptyset$$

Traditional scheduling – Constraints

- Data dependences If $(T, T') \in E$ then
  - if $\text{alloc}(T) = \text{alloc}(T')$ then

$$\sigma(T) + w(T) \leqslant \sigma(T')$$

  - if $\text{alloc}(T) \neq \text{alloc}(T')$ then

$$\sigma(T) + w(T) + c(T, T') \leqslant \sigma(T')$$

- Resource constraints

$$\text{alloc}(T) = \text{alloc}(T') \Rightarrow$$
$$[\sigma(T), \sigma(T) + w(T)[\bigcap [\sigma(T'), \sigma(T') + w(T')[= \emptyset$$

Traditional scheduling – Constraints

- Data dependences If $(T, T') \in E$ then
  - if $\text{alloc}(T) = \text{alloc}(T')$ then

$$\sigma(T) + w(T) \leqslant \sigma(T')$$

  - if $\text{alloc}(T) \neq \text{alloc}(T')$ then

$$\sigma(T) + w(T) + c(T, T') \leqslant \sigma(T')$$

- Resource constraints

$$\text{alloc}(T) = \text{alloc}(T') \Rightarrow$$
$$[\sigma(T), \sigma(T) + w(T)[\bigcap [\sigma(T'), \sigma(T') + w(T')[= \emptyset$$

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} (\sigma(T) + w(T))$$

- Other classical objectives:
  - Sum of completion times
  - With arrival times: maximum flow (response time), or sum flow
  - More fairness with maximum stretch, or sum stretch

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} (\sigma(T) + w(T))$$

- Other classical objectives:
  - Sum of completion times
  - With arrival times: maximum flow (response time), or sum flow
  - More fairness with maximum stretch, or sum stretch

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} (\sigma(T) + w(T))$$

- Other classical objectives:
  - Sum of completion times
  - With arrival times: maximum flow (response time), or sum flow
  - More fairness with maximum stretch, or sum stretch

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} (\sigma(T) + w(T))$$

- Other classical objectives:
  - Sum of completion times
  - With arrival times: maximum flow (response time), or sum flow
  - More fairness with maximum stretch, or sum stretch

Traditional scheduling – Objective functions

- Makespan or total execution time

$$MS(\sigma) = \max_{T \in \mathcal{T}} (\sigma(T) + w(T))$$

- Other classical objectives:
  - Sum of completion times
  - With arrival times: maximum flow (response time), or sum flow
  - More fairness with maximum stretch, or sum stretch

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

  In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

  In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

  In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

  In fact, model assumes infinite network capacity

Traditional scheduling – About the model

- Simple but OK for computational resources
  - No CPU sharing, even in models with preemption
  - At most one task running per processor at any time-step
- Very crude for network resources
  - Unbounded bandwidth on any link
  - Unlimited number of simultaneous sends/receives per processor
  - Fully connected interconnection graph (clique)

  In fact, model assumes infinite network capacity

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)
- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

Makespan minimization

- NP-hardness
  - $Pb(p)$ NP-complete for independent tasks and no communications ($E = \emptyset$, $p = 2$ and $c = \overline{0}$)
  - $Pb(p)$ NP-complete for UET-UCT graphs ($w = c = \overline{1}$)

- Approximation algorithms
  - Without communications, list scheduling is a $(2 - \frac{1}{p})$-approximation
  - With communications, result extends to coarse-grain graphs
  - With communications, no λ-approximation in general

List scheduling – Without communications (1/2)

- *Initialization:*
  1. Compute priority level of all tasks
  2. Priority queue = list of free tasks (tasks without predecessors) sorted by priority
  3. t is the current time step: $t = 0$.
- *While there remain tasks to execute*:
  1. Add new free tasks, if any, to the queue. If the execution of a task terminates at time step t, suppress this task from the predecessor list of all its successors. Add those tasks whose predecessor list has become empty.
  2. If there are q available processors and r tasks in the queue, remove first $\min(q, r)$ tasks from the queue and execute them; if T is one of these tasks, let $\sigma(T) = t$.
  3. Increment t.

List scheduling – Without communications (2/2)

- Priority level
  - Use critical path: longest path from the task to an exit node
  - Computed recursively by a bottom-up traversal of the graph
- Implementation details
  - Cannot iterate from $t = 0$ to $t = MS(\sigma)$ (exponential in problem size)
  - Use a heap for free tasks valued by priority level
  - Use a heap for processors valued by termination time
  - Complexity $O(|V| \log |V| + |E|)$

List scheduling – With communications (1/2)

- Priority level
  - Use *pessimistic* critical path: include all edge costs in the weight
  - Computed recursively by a bottom-up traversal of the graph
- MCP *Modified Critical Path*
  - Assign free task with highest priority to *best* processor
  - Best processor = finishes execution first, given already taken scheduling decisions
  - Free tasks may not be ready for execution (communication delays)
  - May explore inserting the task in empty slots of schedule
  - Complexity $O(|V| \log |V| + (|E| + |V|)p)$

List scheduling – With communications (2/2)

- ETF *Earliest Finish Time*
  - Dynamically recompute priorities of free tasks
  - Select free task that finishes execution first (on best processor), given already taken scheduling decisions
  - Higher complexity $O(|V|^3 p)$
  - May miss "urgent" tasks on critical path
- Other approaches
  - Two-step: clustering + load balancing
    - DSC Dominant Sequence Clustering $O((|V| + |E|) \log |V|)$
    - LLB List-based Load Balancing $O(C \log C + |V|)$ (C number of clusters generated by DSC)
  - Low-cost: FCP Fast Critical Path
    - Maintain constant-size sorted list of free tasks:
    - Best processor = first idle or the one sending last message
    - Low complexity $O(|V| \log p + |E|)$

Extending the model to heterogeneous clusters

- Task graph with n tasks $T_1, \ldots, T_n$.
- Platform with p heterogeneous processors $P_1, \ldots, P_p$.
- Computation costs:
  - w_{iq} = execution time of T_i on P_q
  - $\overline{w_i} = \frac{\sum_{q=1}^{p} w_{iq}}{p}$ average execution time of T_i
  - particular case: consistent tasks $w_{iq} = w_i \times \gamma_q$
- Communication costs:
  - data(i, j): data volume for edge $e_{ij} : T_i \rightarrow T_j$
  - v_{qr}: communication time for unit-size message from P_q to P_r (zero if $q = r$)
  - com(i, j, q, r) = data$(i, j) \times v_{qr}$ communication time from T_i executed on P_q to P_j executed on P_r
  - $\overline{\text{com}_{ij}} = \text{data}(i, j) \times \frac{\sum_{1 \leqslant q, r \leqslant p, q \neq r} v_{qr}}{p(p-1)}$ average communication cost for edge $e_{ij} : T_i \rightarrow T_j$

Rewriting constraints

Dependences For $e_{ij} : T_i \rightarrow T_j$, $q = \text{alloc}(T_i)$ and $r = \text{alloc}(T_j)$:

$$\sigma(T_i) + w_{iq} + \text{com}(i, j, q, r) \leqslant \sigma(T_j)$$

Resources If $q = \text{alloc}(T_i) = \text{alloc}(T_j)$, then

$$(\sigma(T_i) + w_{iq} \leqslant \sigma(T_j)) \text{ or } (\sigma(T_j) + w_{jq} \leqslant \sigma(T_i))$$

Makespan

$$\max_{1 \leqslant i \leqslant n} \left(\sigma(T_i) + w_{i,\text{alloc}(T_i)}\right)$$

HEFT: Heterogeneous Earliest Finishing Time

1. Priority level:
   - $\text{rank}(T_i) = \overline{w_i} + \max_{T_j \in \text{Succ}(T_i)}(\overline{\text{com}_{ij}} + \text{rank}(T_j))$, where $\text{Succ}(T)$ is the set of successors of T
   - Recursive computation by bottom-up traversal of the graph
2. Allocation
   - For current task T_i, determine best processor P_q: minimize $\sigma(T_i) + w_{iq}$
   - Enforce constraints related to communication costs
   - Insertion scheduling: look for $t = \sigma(T_i)$ s.t. P_q is available during interval $[t, t + w_{iq}[$
3. Complexity: same as MCP without/with insertion

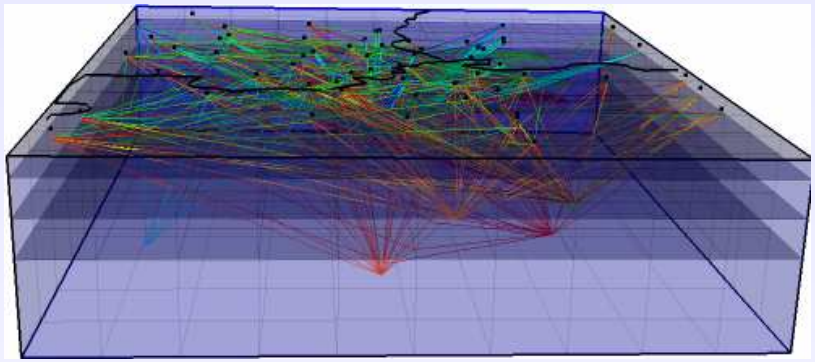
Bibliography

- Introductory book:
  Distributed and parallel computing, H. El-Rewini and T. G. Lewis, Manning 1997
- FCP:
  On the complexity of list scheduling algorithms for distributed-memory systems, A. Radulescu and A.J.C. van Gemund, 13th ACM Int Conf. Supercomputing (1999), 68-75
- HEFT:
  Performance-effective and low-complexity task scheduling for heterogeneous computing, H. Topcuoglu and S. Hariri and M.-Y. Wu, IEEE TPDS 13, 3 (2002), 260-274

Outline

Earth seismic tomography

- Modeling the internal structure of earth



- Validation by comparing expected propagation time of a wave in a model against collected experimental data
- Set of seismic events during 1999: 817101 events
- Original code written for a parallel machine:

```
if (rank = ROOT)
   raydata ← read n lines from data file;
MPI_Scatter(raydata, n/P, ..., rbuff, ...,
                ROOT, MPI_COMM_WORLD);
compute_work(rbuff);
```

Divisible load applications

- Divisible load applications can be divided into any number of independent pieces
- Perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of workers
- Model = good approximation for applications that consist of very (very) large numbers of identical, low-granularity computations
- Large spectrum of scientific problems

Divisible load applications

- Divisible load applications can be divided into any number of independent pieces
- Perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of workers
- Model = good approximation for applications that consist of very (very) large numbers of identical, low-granularity computations
- Large spectrum of scientific problems

Divisible load applications

- Divisible load applications can be divided into any number of independent pieces
- Perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of workers
- Model = good approximation for applications that consist of very (very) large numbers of identical, low-granularity computations
- Large spectrum of scientific problems

Divisible load applications

- Divisible load applications can be divided into any number of independent pieces
- Perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of workers
- Model = good approximation for applications that consist of very (very) large numbers of identical, low-granularity computations
- Large spectrum of scientific problems

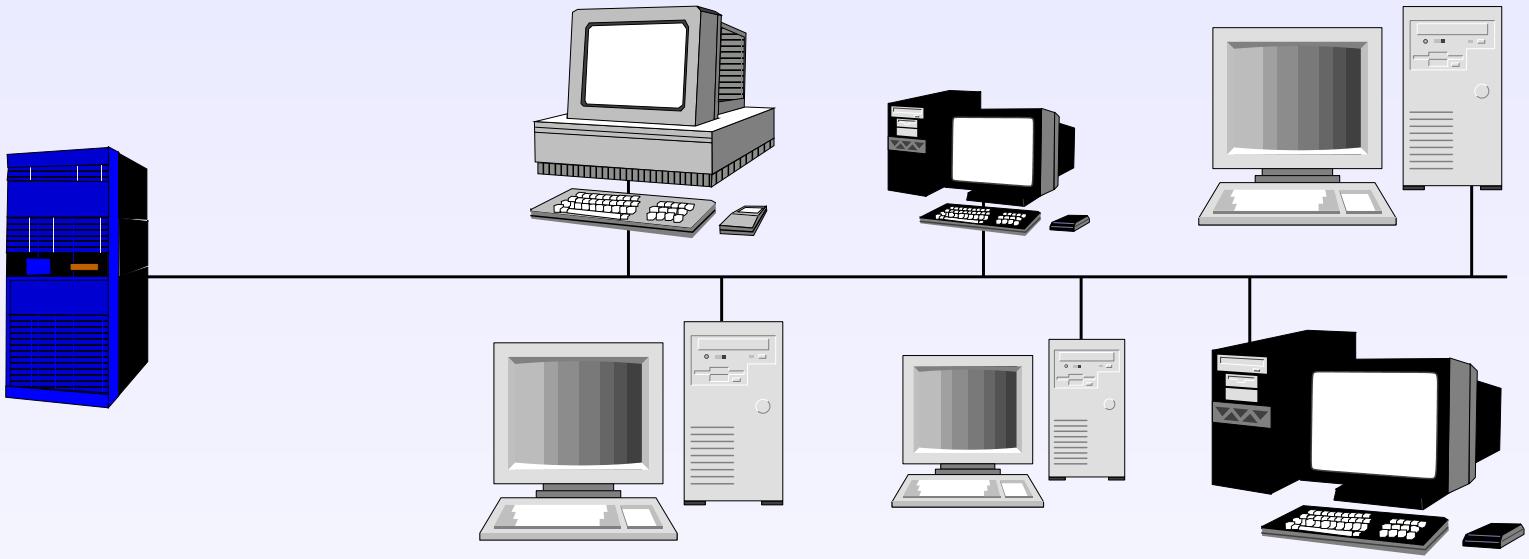
Divisible Load Scheduling (DLS)

Applications composed of a very (very) large number of low-granularity computations
Computation time proportional to data volume processed: linear cost model

Independent computations: neither synchronizations nor communications

Outline

Bus network



- Identical links between master and slaves
- Slaves have *different* computing power

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $n_i \in \mathbb{N}$ load units, where $\sum_i n_i = W_{\text{total}}$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $n_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $n_i \in \mathbb{N}$ load units, where $\sum_i n_i = W_{\text{total}}$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $n_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $n_i \in \mathbb{N}$ load units, where $\sum_i n_i = W_{\text{total}}$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $n_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $n_i \in \mathbb{N}$ load units, where $\sum_i n_i = W_{\text{total}}$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $n_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

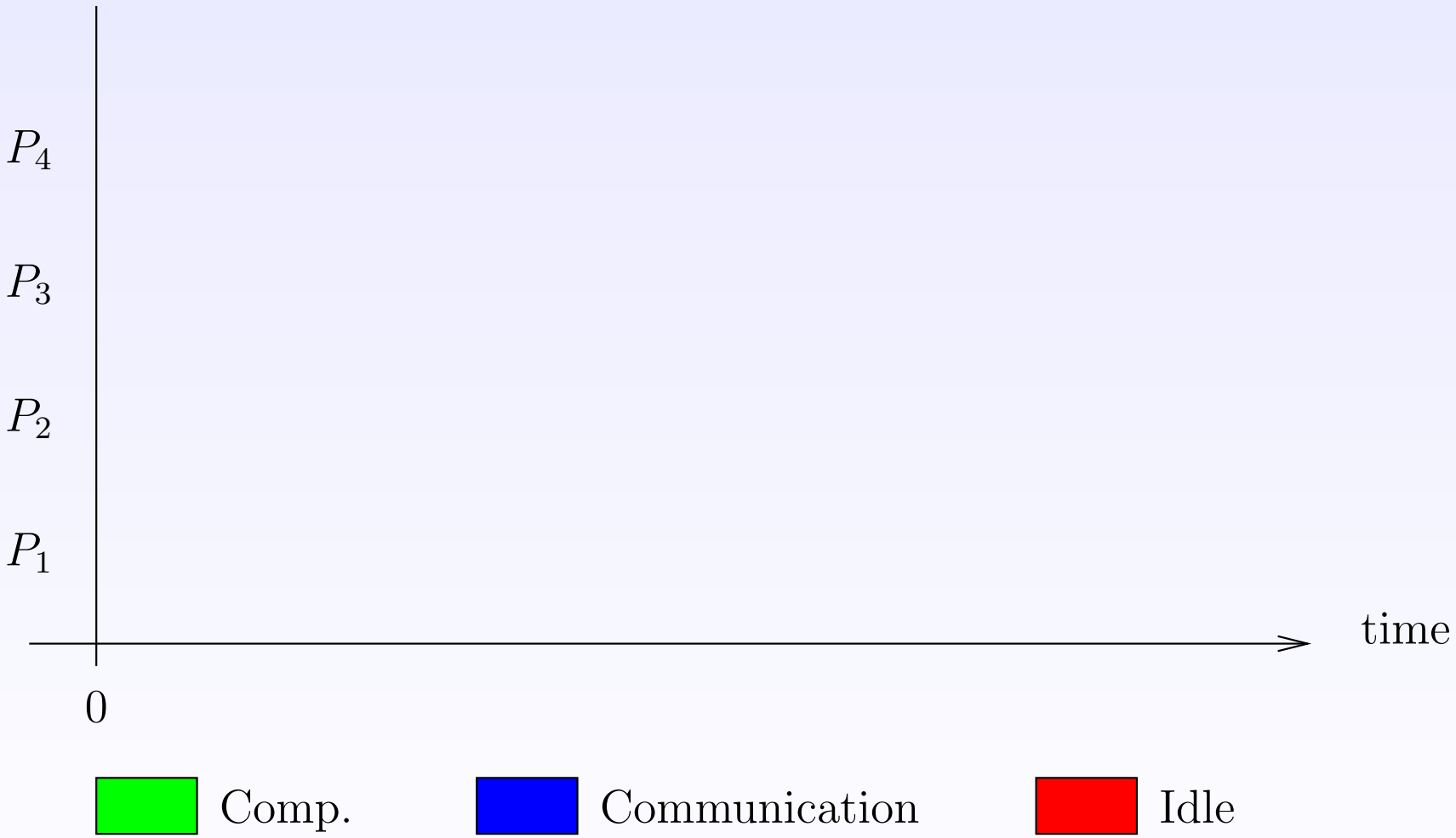
Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $n_i \in \mathbb{N}$ load units, where $\sum_i n_i = W_{\text{total}}$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $n_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

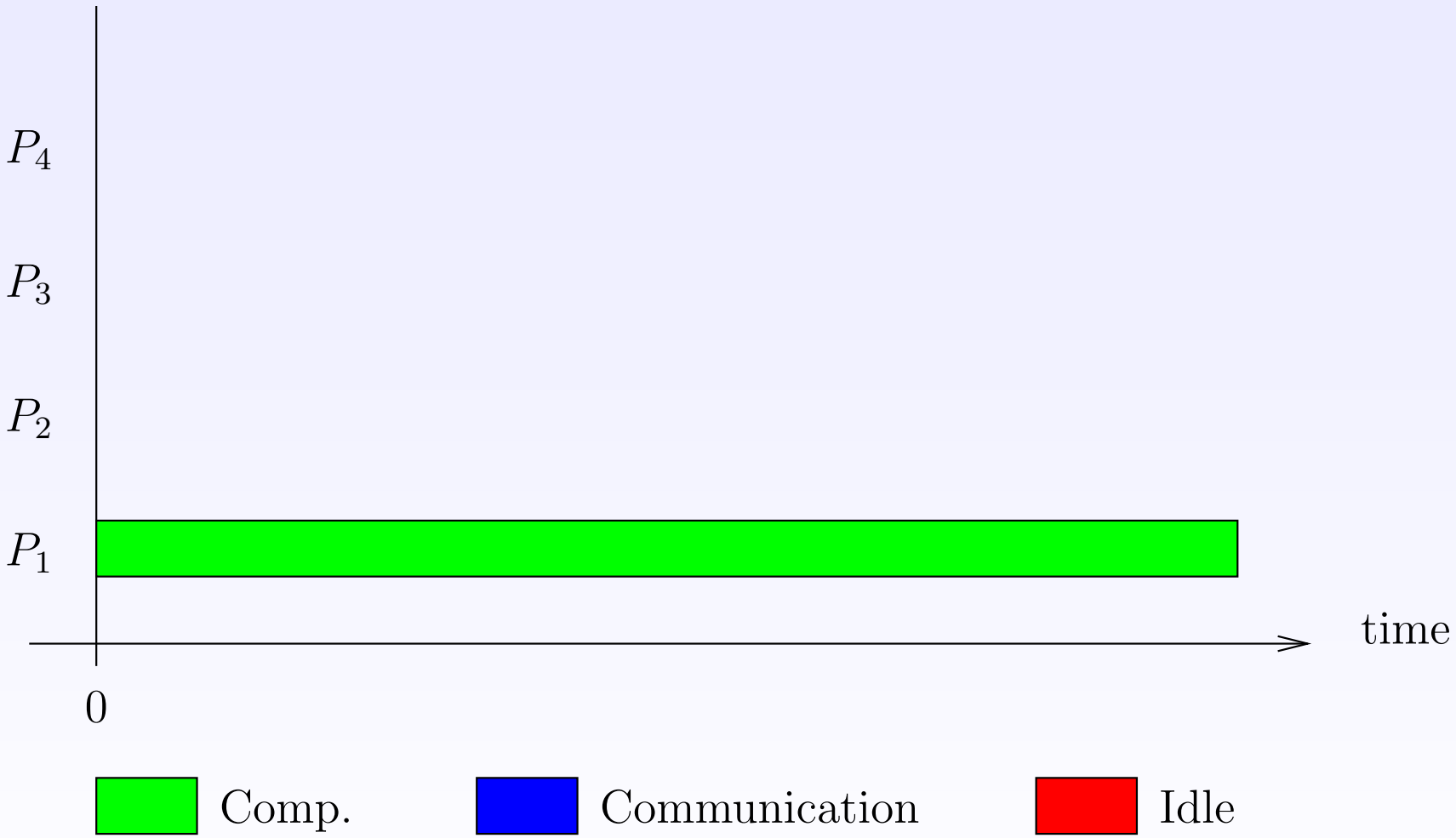
Assessing the model

- One-port hypothesis is realistic ☺
- Linear cost model is simple ☹ but acceptable ☺ for
  - large problems
  - one-round scenarios

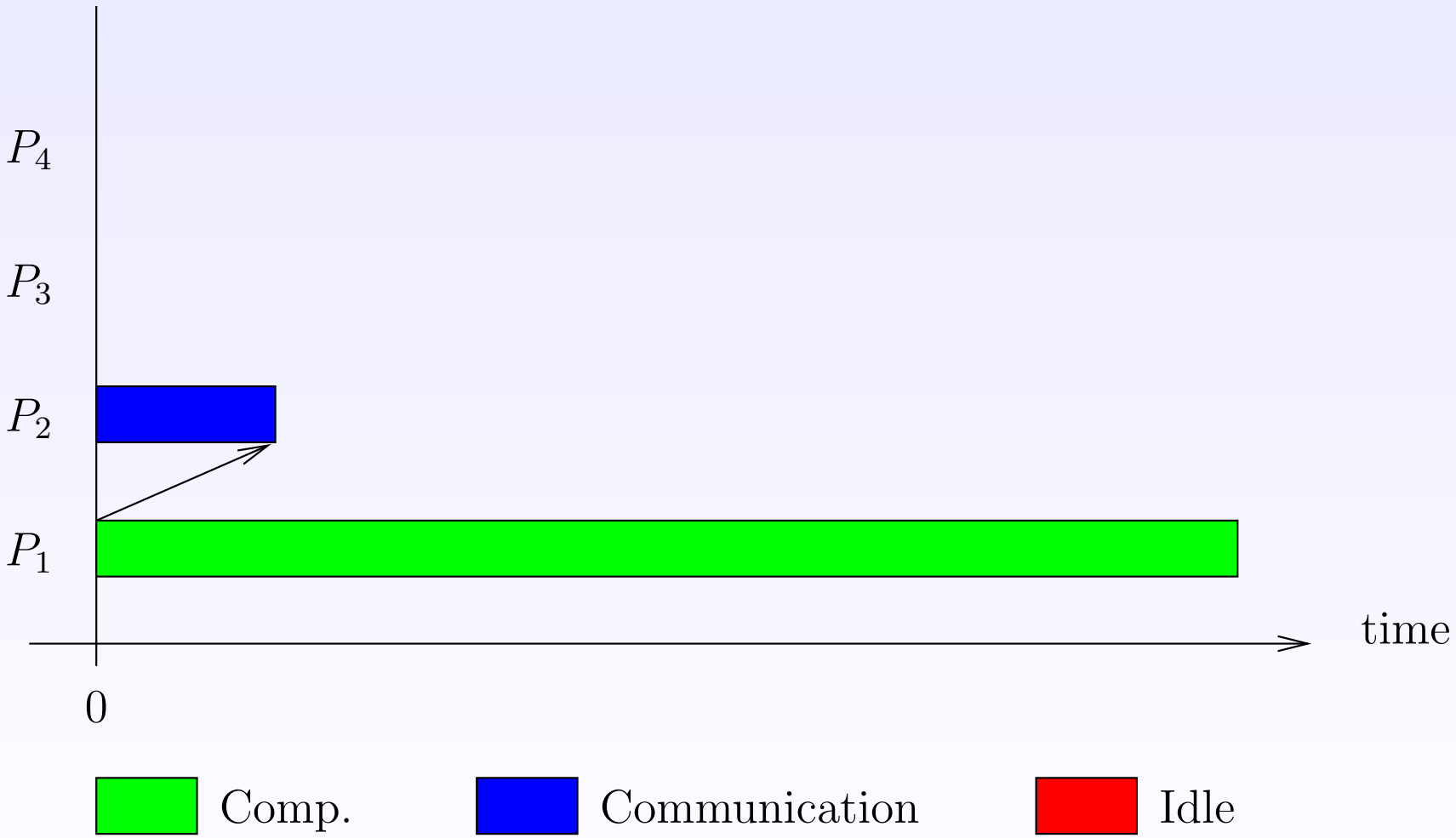
Scheduling example



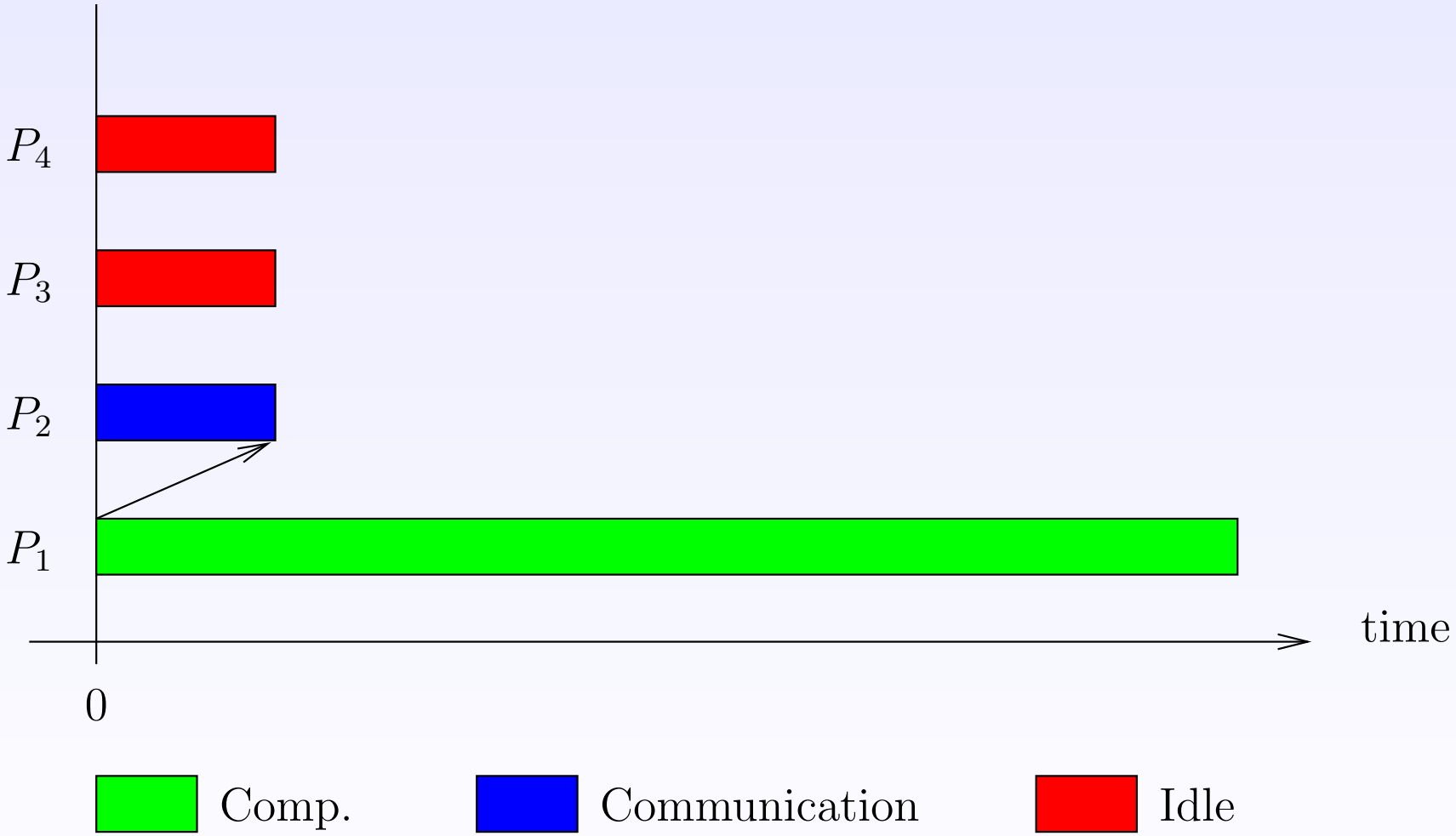
Scheduling example



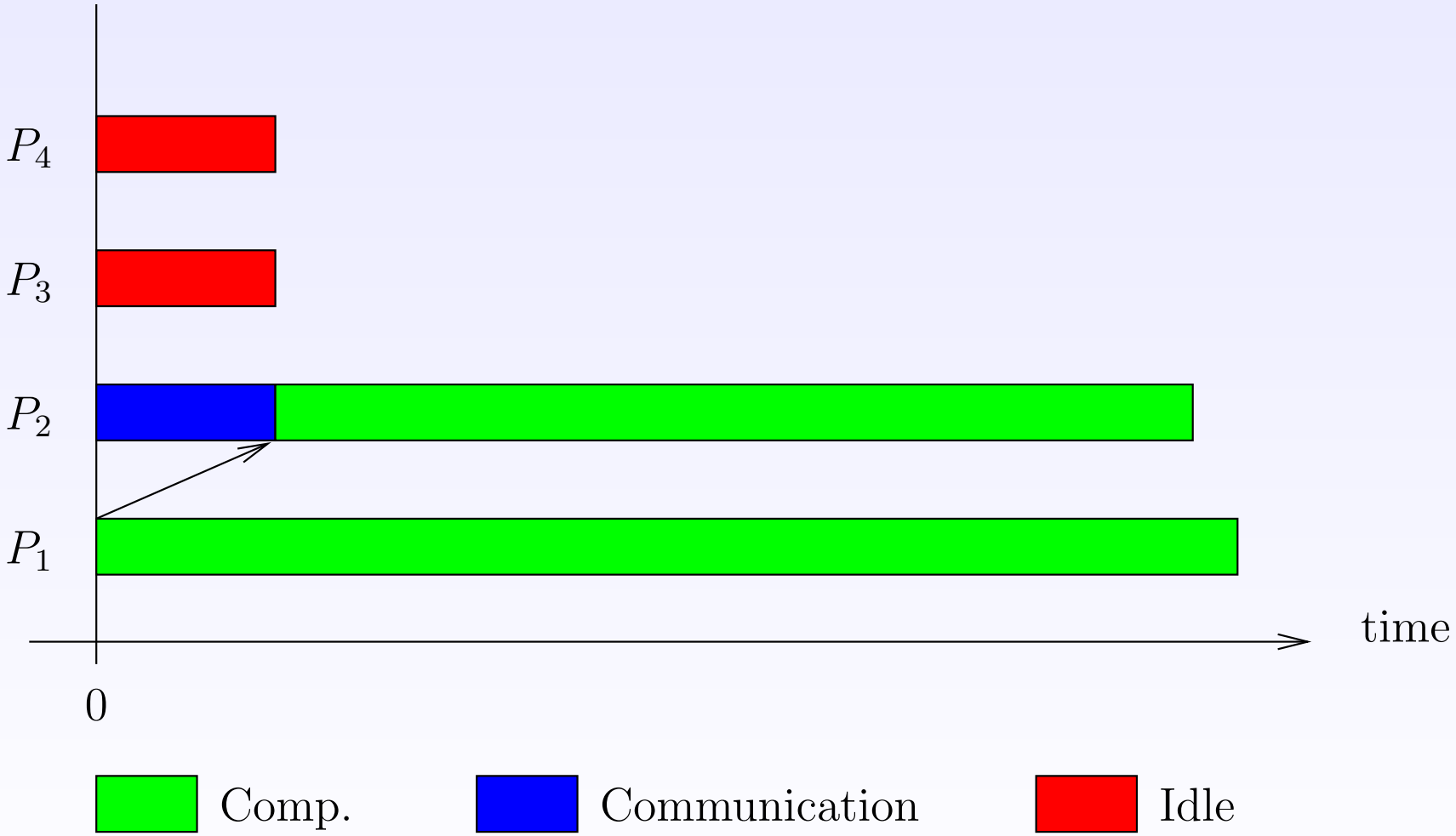
Scheduling example



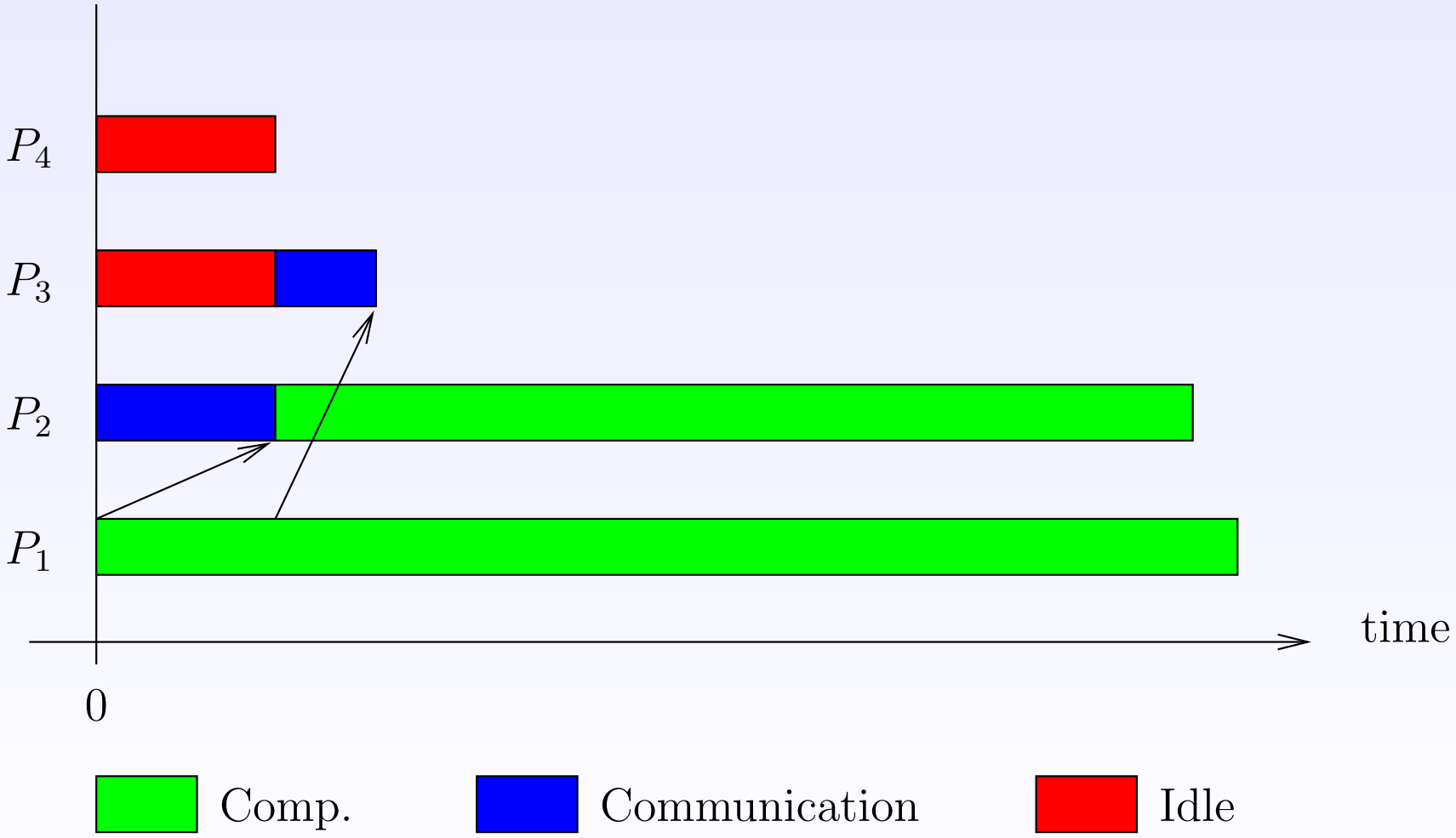
Scheduling example



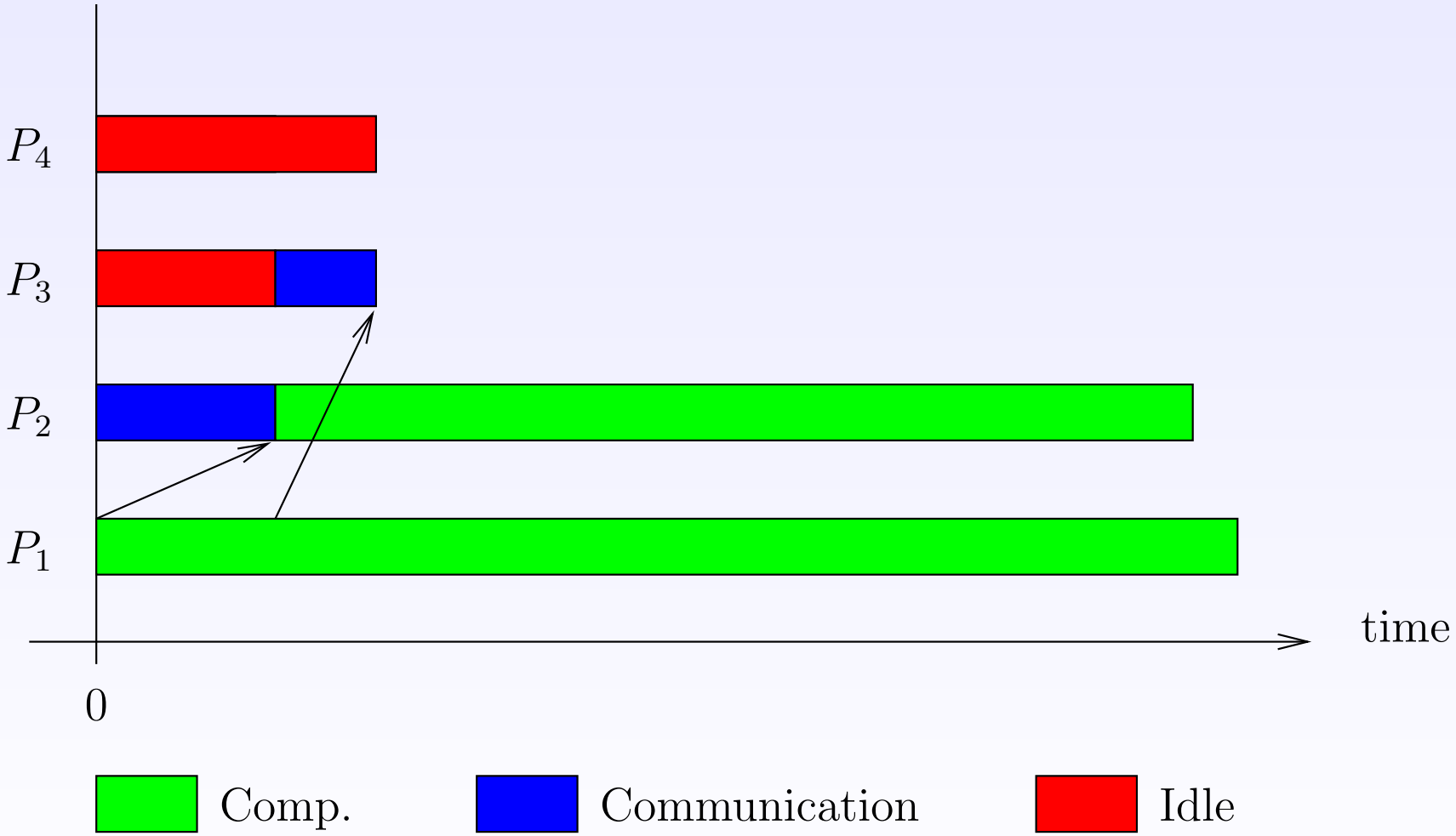
Scheduling example



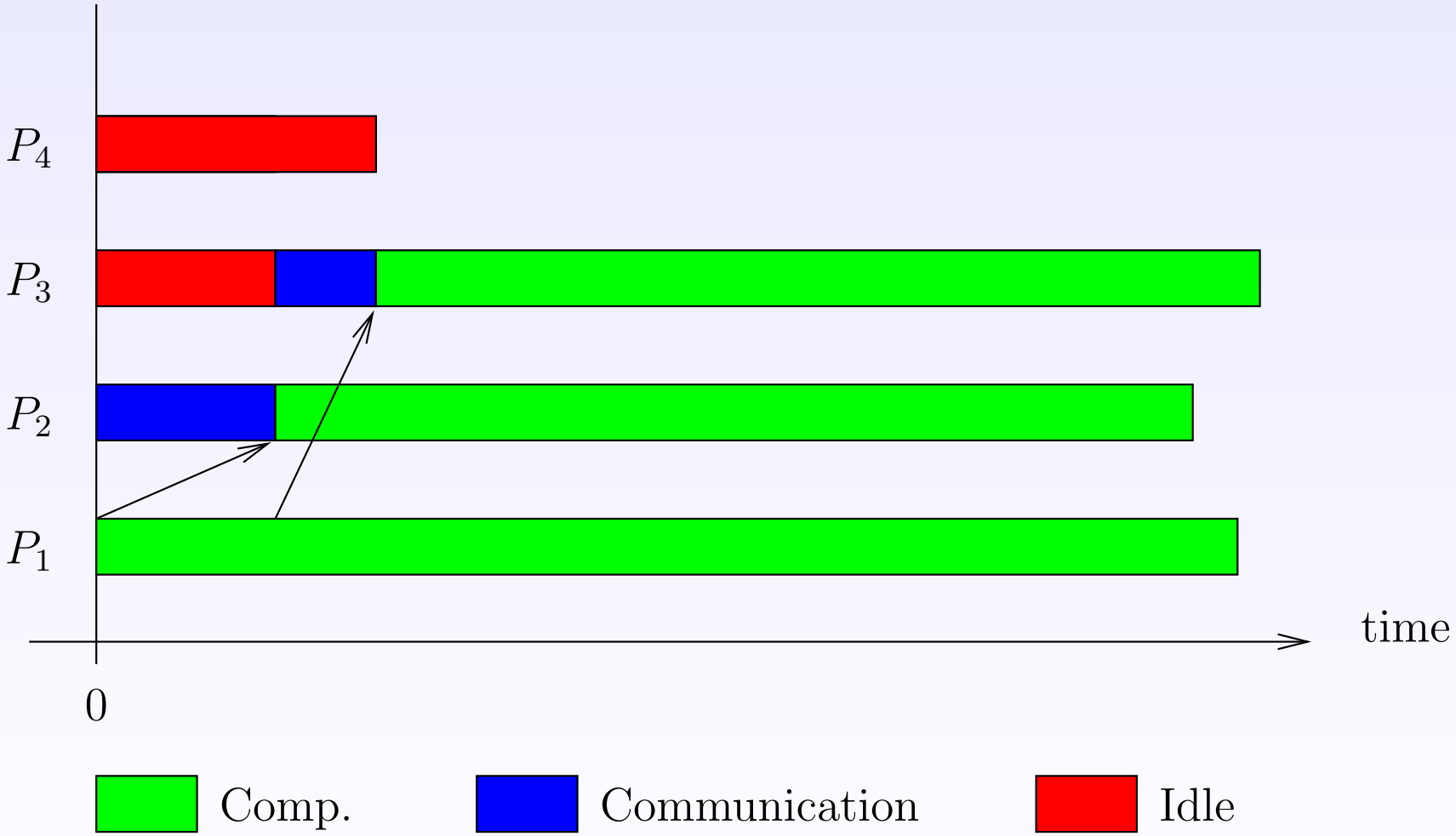
Scheduling example



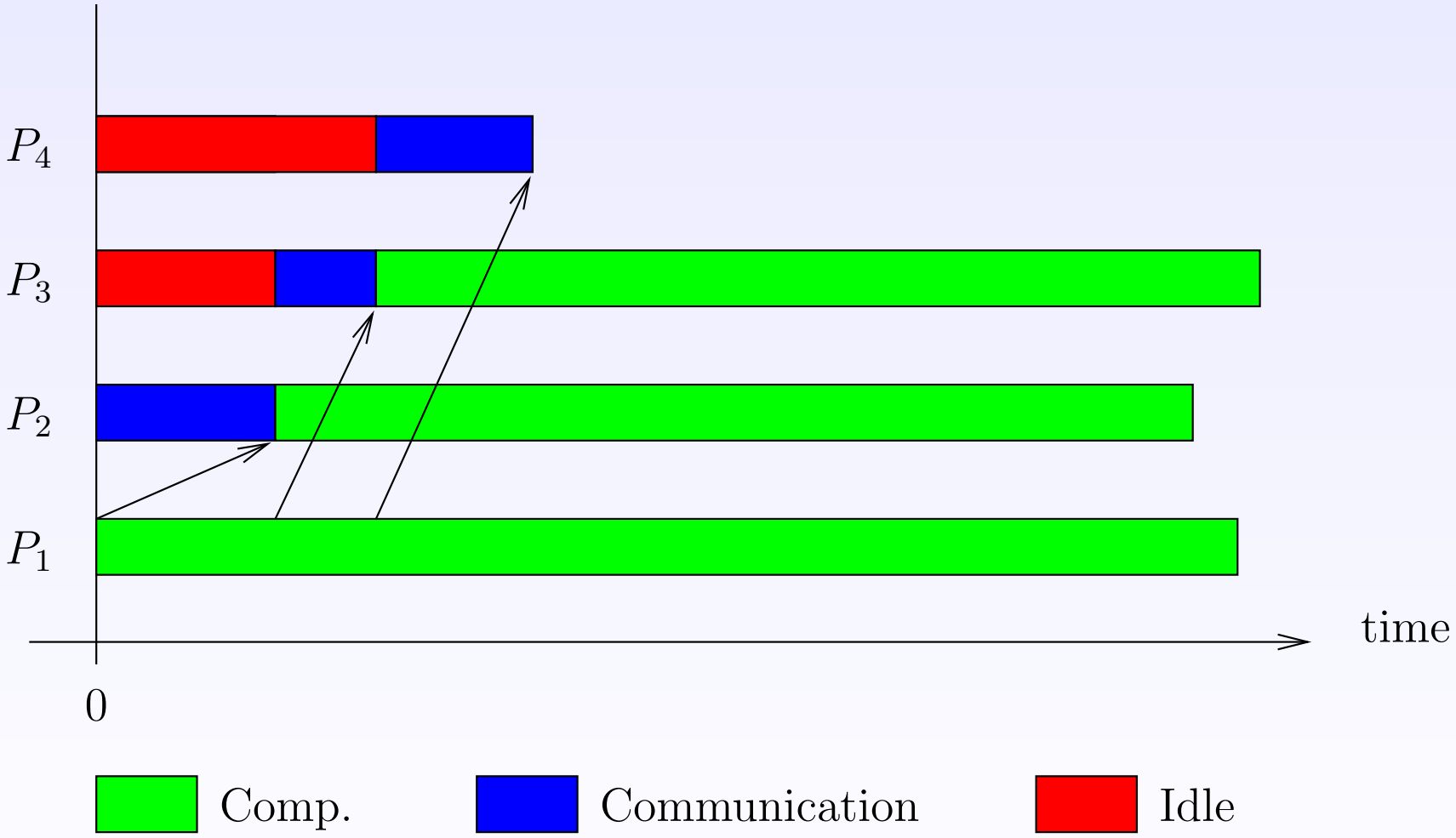
Scheduling example



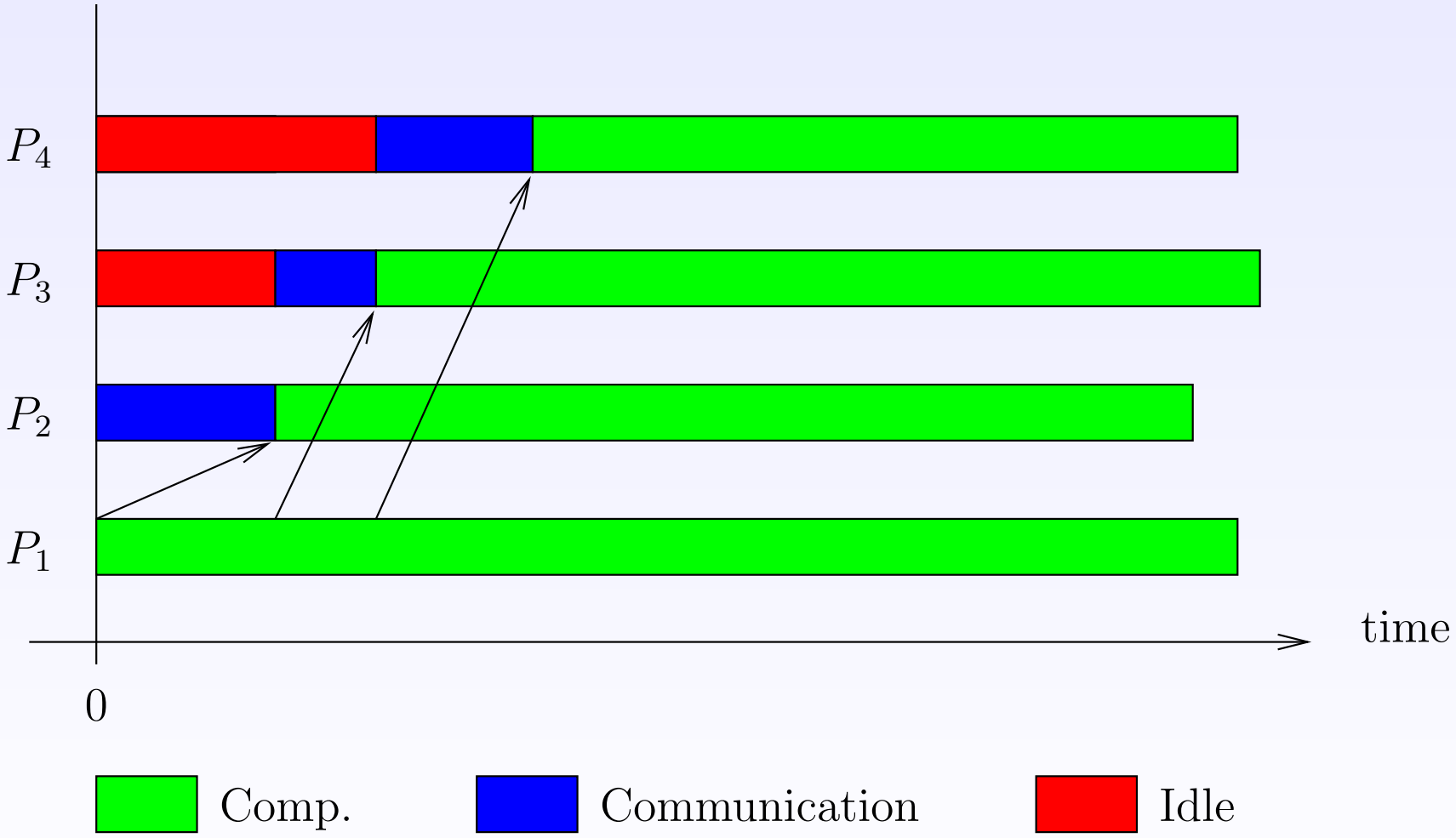
Scheduling example



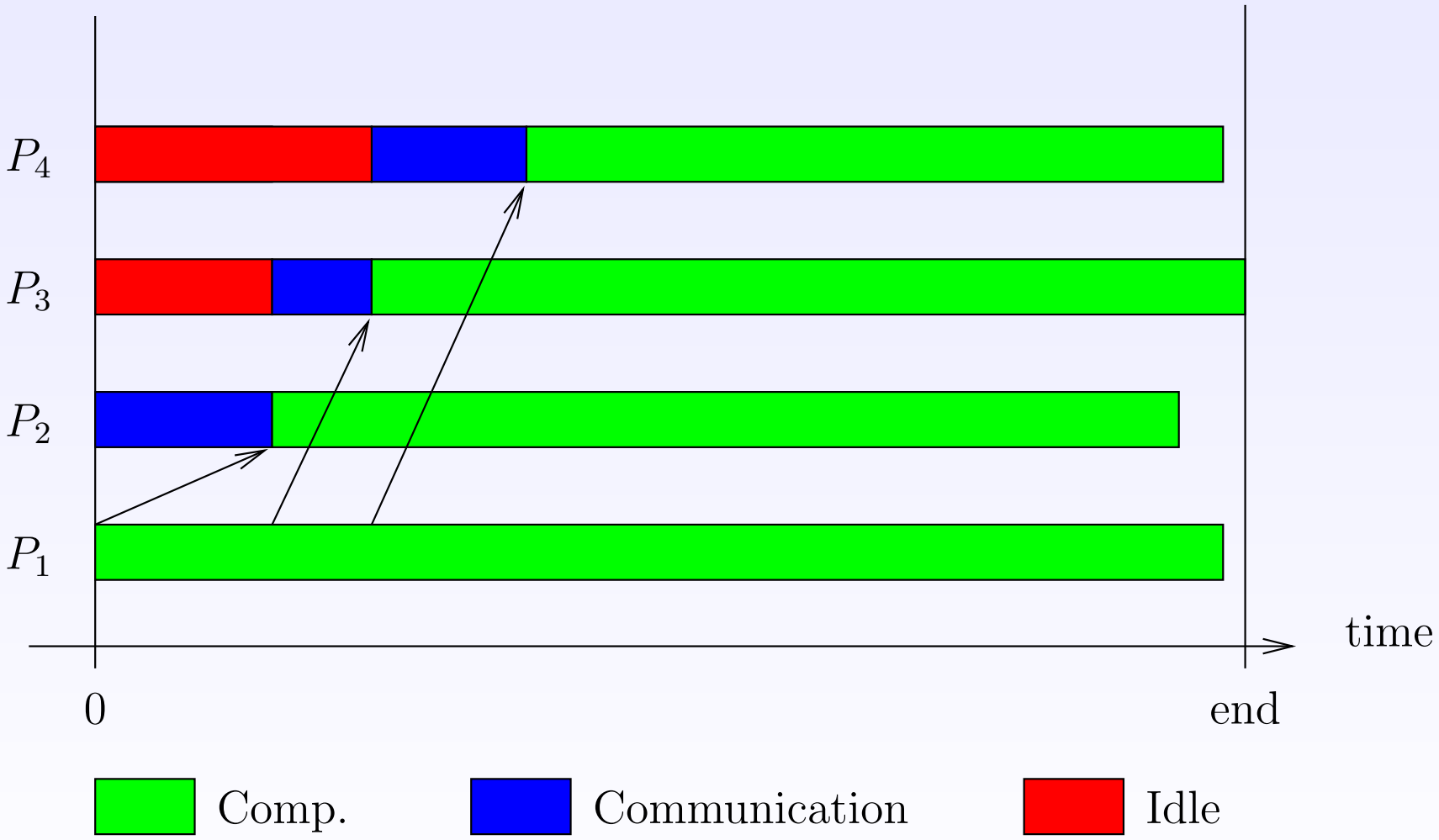
Scheduling example



Scheduling example



Scheduling example



Problem specification

- Master sends n_i data items to P_i within a single message
- Master sequentially sends messages to slaves in the order

$$P_2, P_3, ..., P_p$$

- Simultaneously, master processes its own n_1 data items
- A slave cannot start computing before reception of its message is complete
- Question: compute best allocation and corresponding schedule

Problem specification

- Master sends n_i data items to P_i within a single message
- Master sequentially sends messages to slaves in the order

$$P_2, P_3, ..., P_p$$

- Simultaneously, master processes its own n_1 data items
- A slave cannot start computing before reception of its message is complete
- Question: compute best allocation and corresponding schedule

Problem specification

- Master sends n_i data items to P_i within a single message
- Master sequentially sends messages to slaves in the order

$$P_2, P_3, ..., P_p$$

- Simultaneously, master processes its own n_1 data items
- A slave cannot start computing before reception of its message is complete
- Question: compute best allocation and corresponding schedule

Problem specification

- Master sends n_i data items to P_i within a single message
- Master sequentially sends messages to slaves in the order

$$P_2, P_3, ..., P_p$$

- Simultaneously, master processes its own n_1 data items
- A slave cannot start computing before reception of its message is complete
- Question: compute best allocation and corresponding schedule

Problem specification

- Master sends n_i data items to P_i within a single message
- Master sequentially sends messages to slaves in the order

$$P_2, P_3, ..., P_p$$

- Simultaneously, master processes its own n_1 data items
- A slave cannot start computing before reception of its message is complete
- Question: compute best allocation and corresponding schedule

Equations

- P_1: $T_1 = n_1.w_1$
- P_2: $T_2 = n_2.c + n_2.w_2$
- P_3: $T_3 = (n_2.c + n_3.c) + n_3.w_3$
- P_i: $T_i = \sum_{j=2}^{i} n_j.c + n_i.w_i$ for $i \geqslant 2$
- P_i: $T_i = \sum_{j=1}^{i} n_j.c_j + n_i.w_i$ for $i \geqslant 1$
  where $c_1 = 0$ and $c_j = c$ if $j \neq 1$

Equations

- P_1: $T_1 = n_1.w_1$
- P_2: $T_2 = n_2.c + n_2.w_2$
- P_3: $T_3 = (n_2.c + n_3.c) + n_3.w_3$
- P_i: $T_i = \sum_{j=2}^{i} n_j.c + n_i.w_i$ for $i \geqslant 2$
- P_i: $T_i = \sum_{j=1}^{i} n_j.c_j + n_i.w_i$ for $i \geqslant 1$
  where $c_1 = 0$ and $c_j = c$ if $j \neq 1$

Equations

- P_1: $T_1 = n_1.w_1$
- P_2: $T_2 = n_2.c + n_2.w_2$
- P_3: $T_3 = (n_2.c + n_3.c) + n_3.w_3$
- P_i: $T_i = \sum_{j=2}^{i} n_j.c + n_i.w_i$ for $i \geqslant 2$
- P_i: $T_i = \sum_{j=1}^{i} n_j.c_j + n_i.w_i$ for $i \geqslant 1$
  where $c_1 = 0$ and $c_j = c$ if $j \neq 1$

Equations

- P_1: $T_1 = n_1.w_1$
- P_2: $T_2 = n_2.c + n_2.w_2$
- P_3: $T_3 = (n_2.c + n_3.c) + n_3.w_3$
- P_i: $T_i = \sum_{j=2}^{i} n_j.c + n_i.w_i$ for $i \geqslant 2$
- P_i: $T_i = \sum_{j=1}^{i} n_j.c_j + n_i.w_i$ for $i \geqslant 1$
  where $c_1 = 0$ and $c_j = c$ if $j \neq 1$

Equations

- P_1: $T_1 = n_1.w_1$
- P_2: $T_2 = n_2.c + n_2.w_2$
- P_3: $T_3 = (n_2.c + n_3.c) + n_3.w_3$
- P_i: $T_i = \sum_{j=2}^{i} n_j.c + n_i.w_i$ for $i \geqslant 2$
- P_i: $T_i = \sum_{j=1}^{i} n_j.c_j + n_i.w_i$ for $i \geqslant 1$
  where $c_1 = 0$ and $c_j = c$ if $j \neq 1$

Execution time (1/2)

$$T = \max_{1 \leqslant i \leqslant p} \left(\sum_{j=1}^{i} n_j.c_j + n_i.w_i \right)$$

Goal: find distribution n_1, ..., n_p to minimize T

Execution time (2/2)

$$T = \max\left(n_1.c_1 + n_1.w_1, \max_{2 \leqslant i \leqslant p}\left(\sum_{j=1}^{i} n_j.c_j + n_i.w_i\right)\right)$$

$$T = n_1.c_1 + \max\left(n_1.w_1, \max_{2 \leqslant i \leqslant p}\left(\sum_{j=2}^{i} n_j.c_j + n_i.w_i\right)\right)$$

Optimal distribution of W_{total} data among p processors:
- assign n_1 data items to P_1
- use optimal distribution of $W_{\text{total}} - n_1$ data items among $p-1$ remaining processors

Dynamic Programming Algorithm

1: $\mathit{solution}[0,p] \leftarrow \text{cons}(0, NIL)$; $\mathit{cost}[0,p] \leftarrow 0$
2: **for** $d \leftarrow 1$ **to** W_{total} **do**
3: $\quad \mathit{solution}[d,p] \leftarrow \text{cons}(d, NIL)$
4: $\quad \mathit{cost}[d,p] \leftarrow d \cdot c_p + d \cdot w_p$
5: **for** $i \leftarrow p-1$ **downto** 1 **do**
6: $\quad \mathit{solution}[0,i] \leftarrow \text{cons}(0, \mathit{solution}[0,i+1])$
7: $\quad \mathit{cost}[0,i] \leftarrow 0$
8: $\quad$ **for** $d \leftarrow 1$ **to** W_{total} **do**
9: $\quad\quad (sol, min) \leftarrow (0, \mathit{cost}[d,i+1])$
10: $\quad\quad$ **for** $e \leftarrow 1$ **to** d **do**
11: $\quad\quad\quad m \leftarrow e \cdot c_i + \max(e \cdot w_i, \mathit{cost}[d-e,i+1])$
12: $\quad\quad\quad$ **if** $m < min$ **then**
13: $\quad\quad\quad\quad (sol, min) \leftarrow (e, m)$
14: $\quad\quad \mathit{solution}[d,i] \leftarrow \text{cons}(sol, \mathit{solution}[d-sol,i+1])$
15: $\quad\quad \mathit{cost}[d,i] \leftarrow min$
16: **return** $(\mathit{solution}[W_{\text{total}},1], \mathit{cost}[W_{\text{total}},1])$

Complexity

- **Theoretical**

$$O(W_{\text{total}}^2 \cdot p)$$

- **In practice**
  With $W_{\text{total}} = 817101$ and $p = 16$, using a 933MHz Pentium III: over two days ...
  (Optimized version: 6 minutes)

Do we really need such an exact integer solution?
No! ☺

Complexity

- **Theoretical**

$$O(W_{\text{total}}^2 \cdot p)$$

- **In practice**
  With $W_{\text{total}} = 817101$ and $p = 16$, using a 933MHz Pentium III:
  over two days ...
  (Optimized version: 6 minutes)

Do we really need such an exact integer solution?

No! ☺

Complexity

- **Theoretical**

$$O(W_{\text{total}}^2 \cdot p)$$

- **In practice**
  With $W_{\text{total}} = 817101$ and $p = 16$, using a 933MHz Pentium III: over two days ...
  (Optimized version: 6 minutes)

Do we really need such an exact integer solution?

No! ☺

Outline

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- **Processor P_i receives $\alpha_i W_{\text{total}}$ load units where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$**
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- **Processor P_i receives $\alpha_i W_{\text{total}}$ load units where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$**
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- **Processor P_i receives $\alpha_i W_{\text{total}}$ load units where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$**
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- **Processor P_i receives $\alpha_i W_{\text{total}}$ load units where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$**
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- **Processor P_i receives $\alpha_i W_{\text{total}}$ load units where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$**
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- Communication time of one load-unit from P_1 to P_i: c
  One-port model: P_1 serially sends one message to each slave

Equations

For processor P_i (where $c_1 = 0$ et $c_j = c$ if $j \neq 1$):

$$T_i = \sum_{j=1}^{i} \alpha_j W_{\text{total}}.c_j + \alpha_i W_{\text{total}}.w_i$$

$$T = \max_{1 \leqslant i \leqslant p} \left(\sum_{j=1}^{i} \alpha_j W_{\text{total}}.c_j + \alpha_i W_{\text{total}}.w_i \right)$$

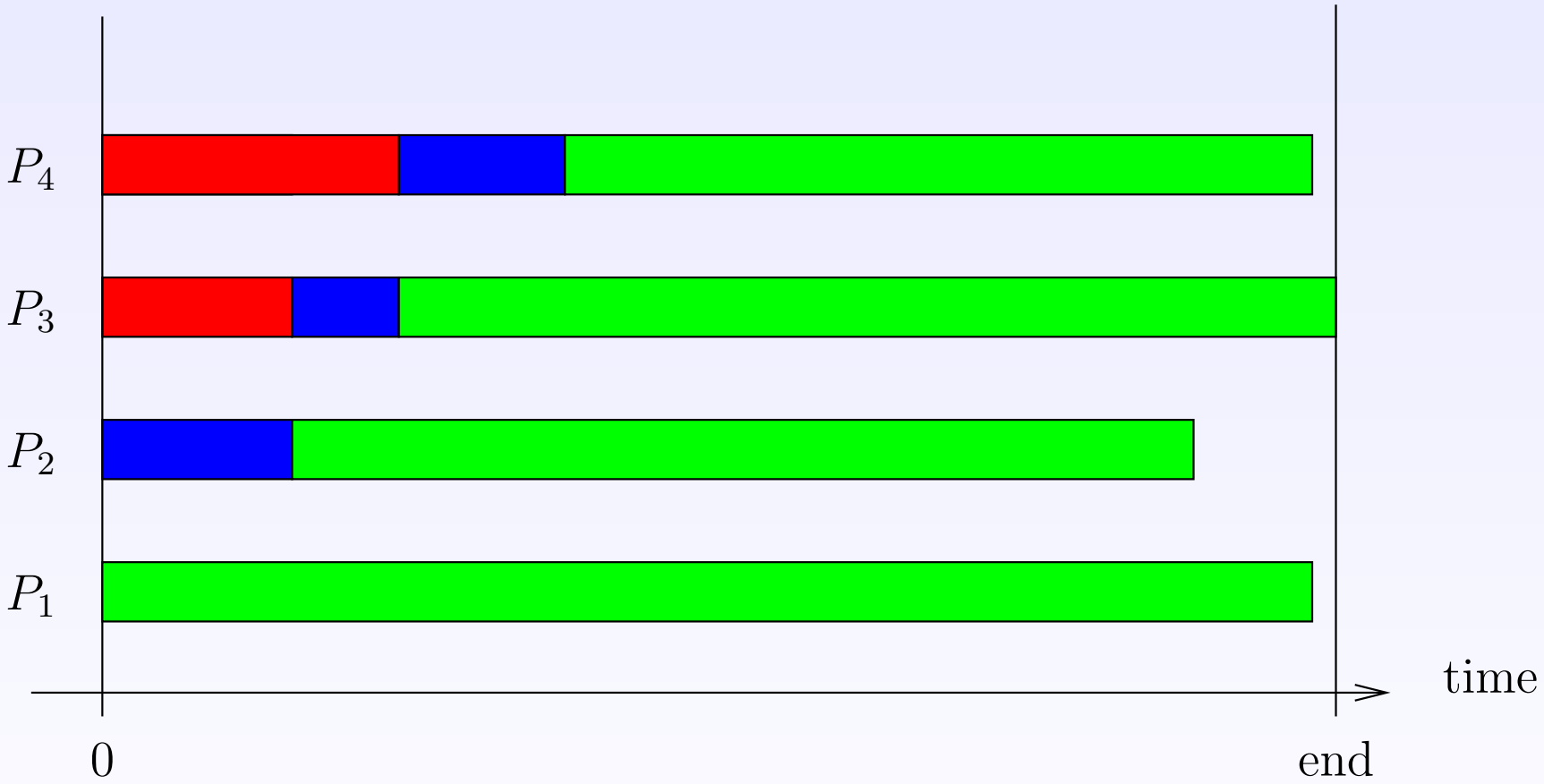
Goal: find distribution α_1, ..., α_p to minimize T

Load balancing property

Lemma *In any optimal solution, all processors terminate execution simultaneously*

Proof of Lemma (1/2)

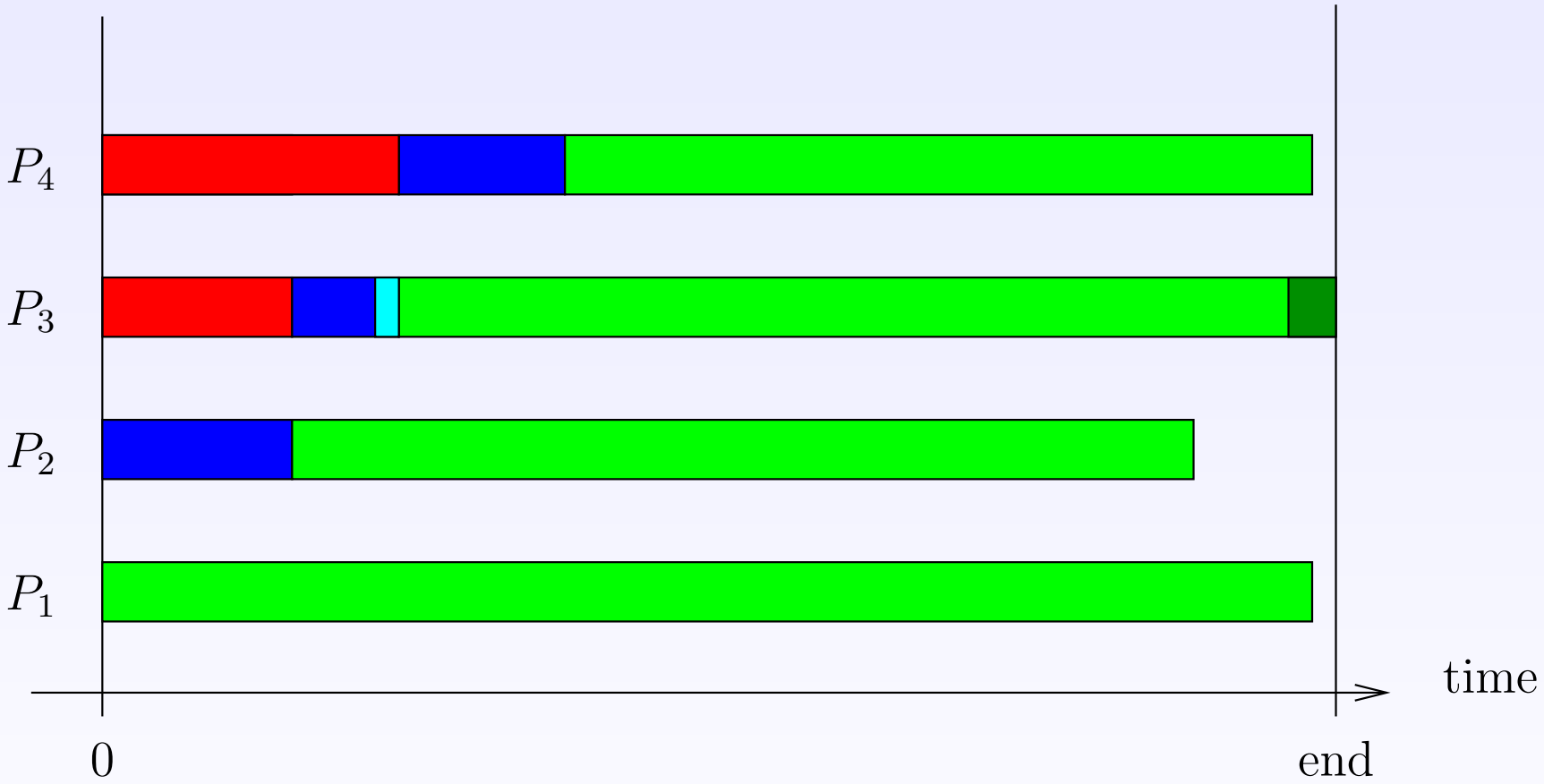
Two slaves i and $i+1$ with $T_i < T_{i+1}$



Decrease α_{i+1} by ε

Proof of Lemma (1/2)

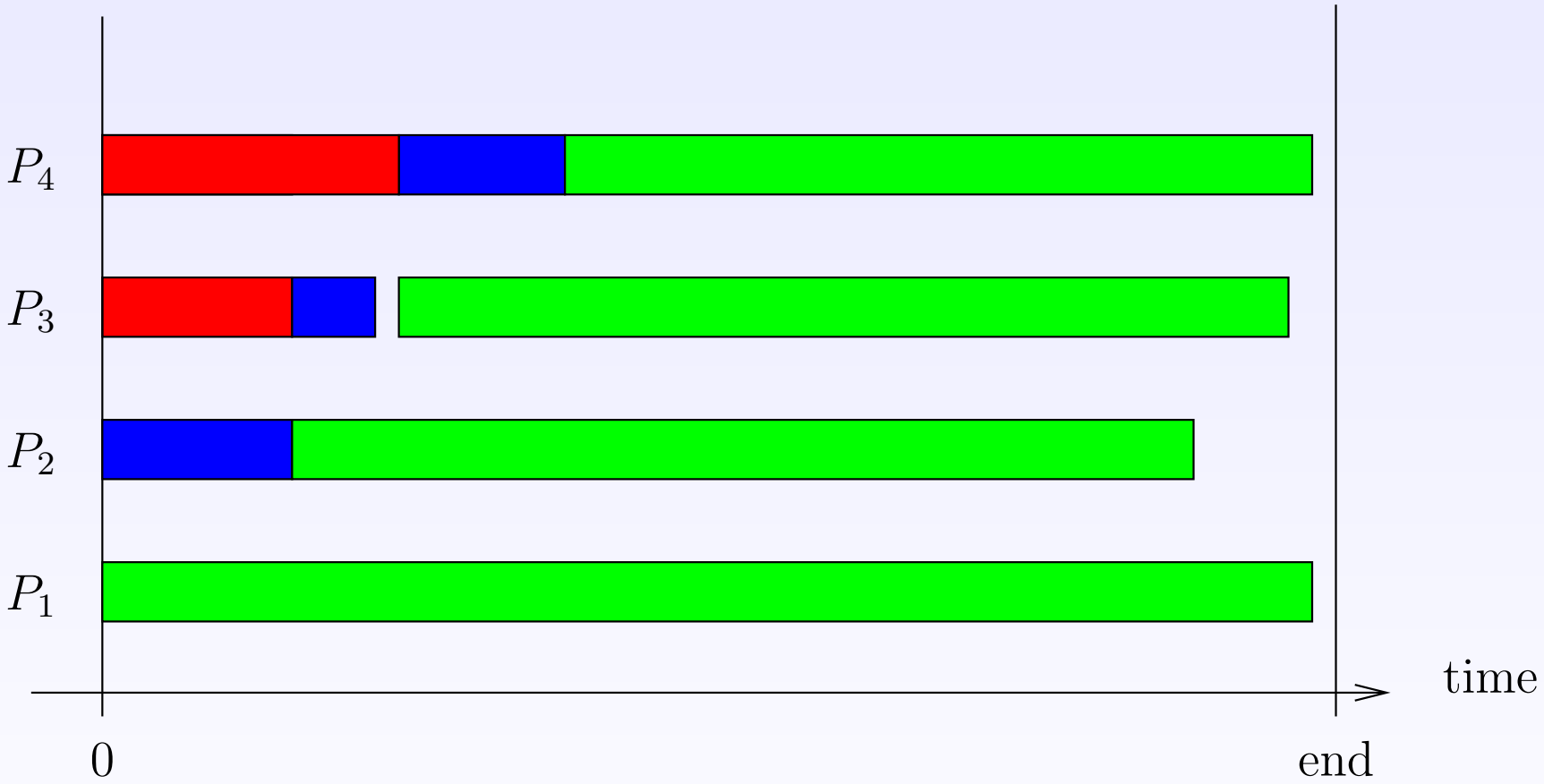
Two slaves i and $i+1$ with $T_i < T_{i+1}$



Decrease α_{i+1} by ε

Proof of Lemma (1/2)

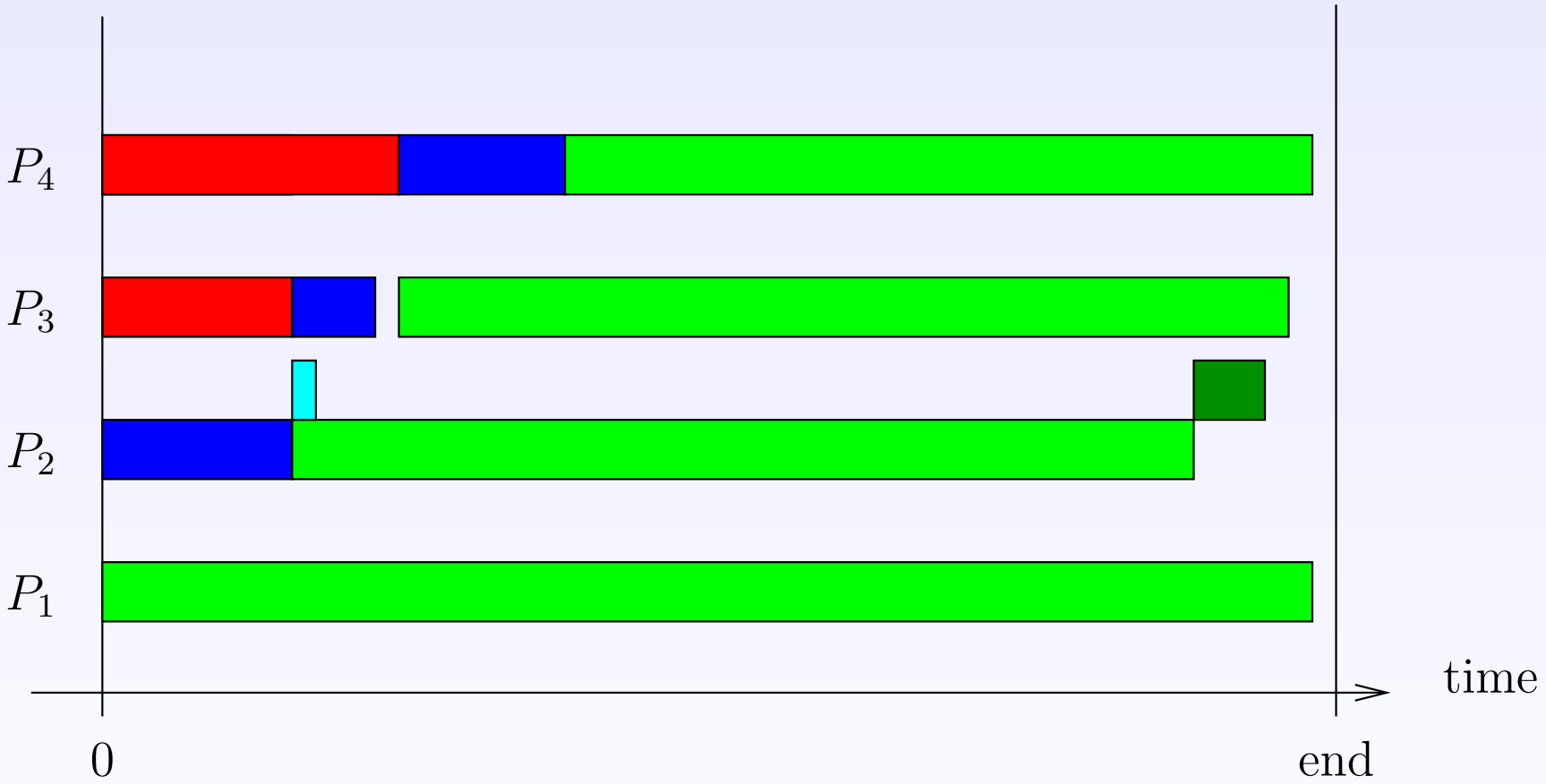
Two slaves i and $i+1$ with $T_i < T_{i+1}$



Decrease α_{i+1} by ε

Proof of Lemma (1/2)

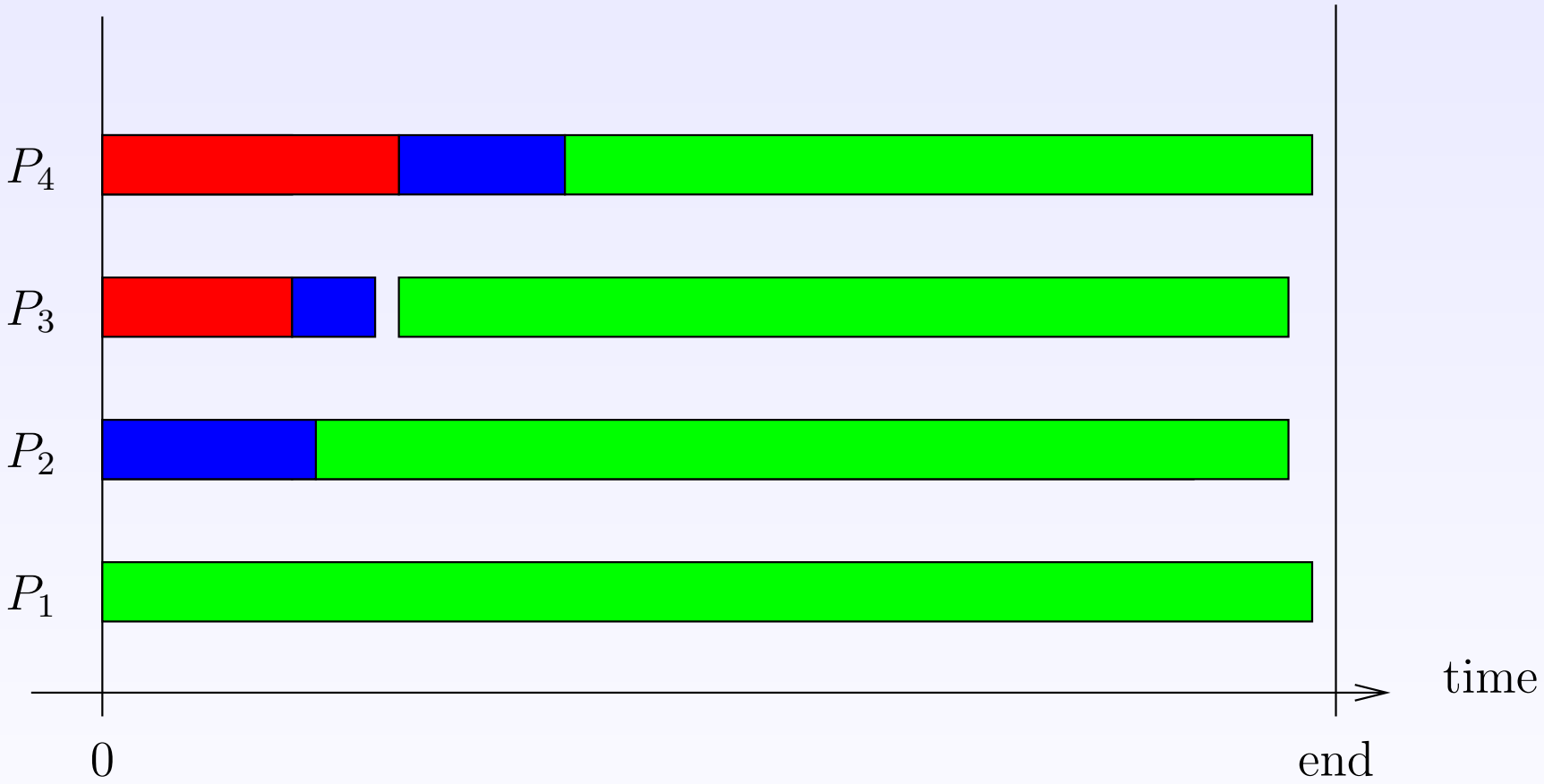
Two slaves i and $i+1$ with $T_i < T_{i+1}$



Increase α_i by ε

Proof of Lemma (1/2)

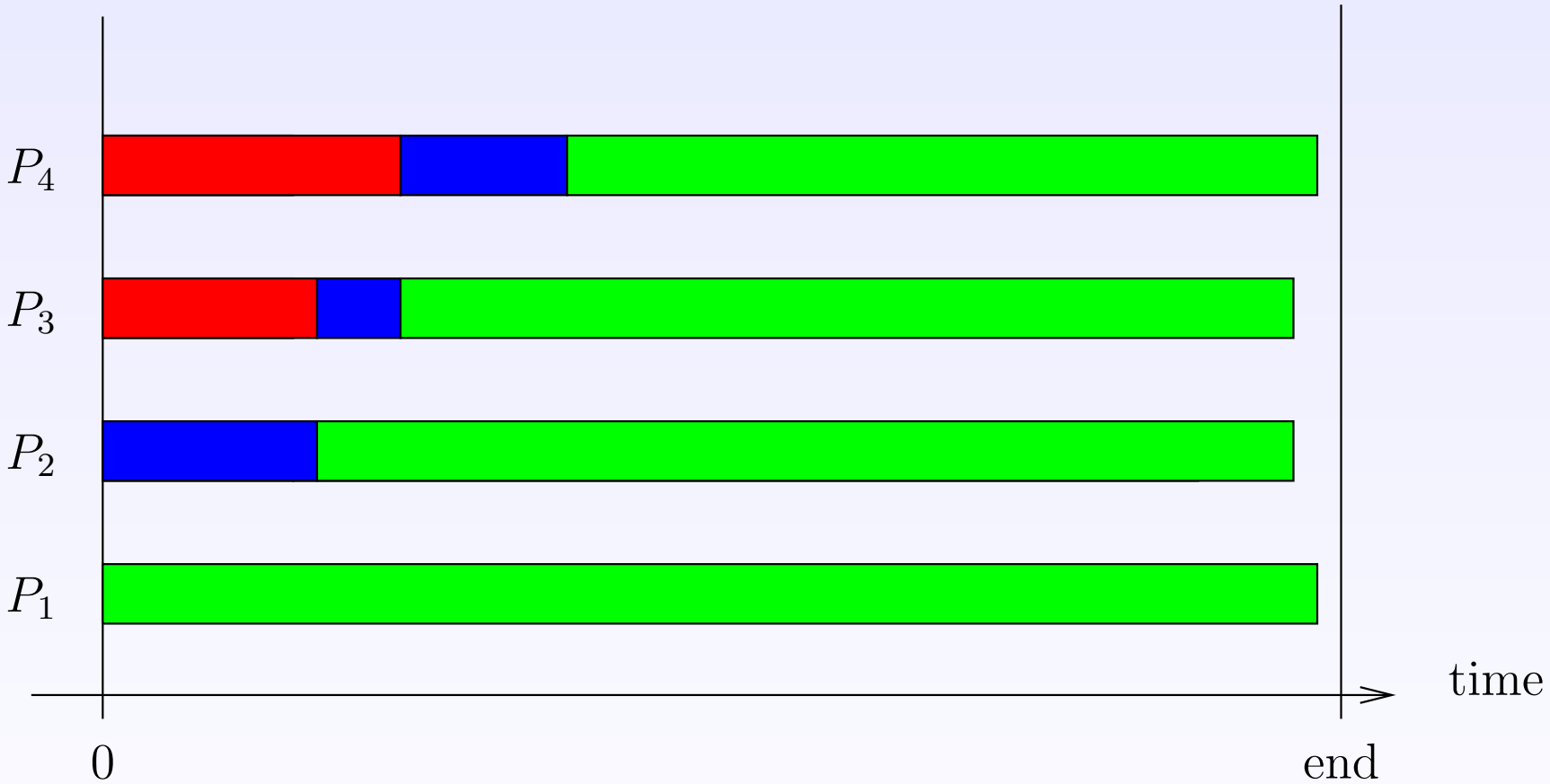
Two slaves i and $i+1$ with $T_i < T_{i+1}$



Increase α_i by ε

Proof of Lemma (1/2)

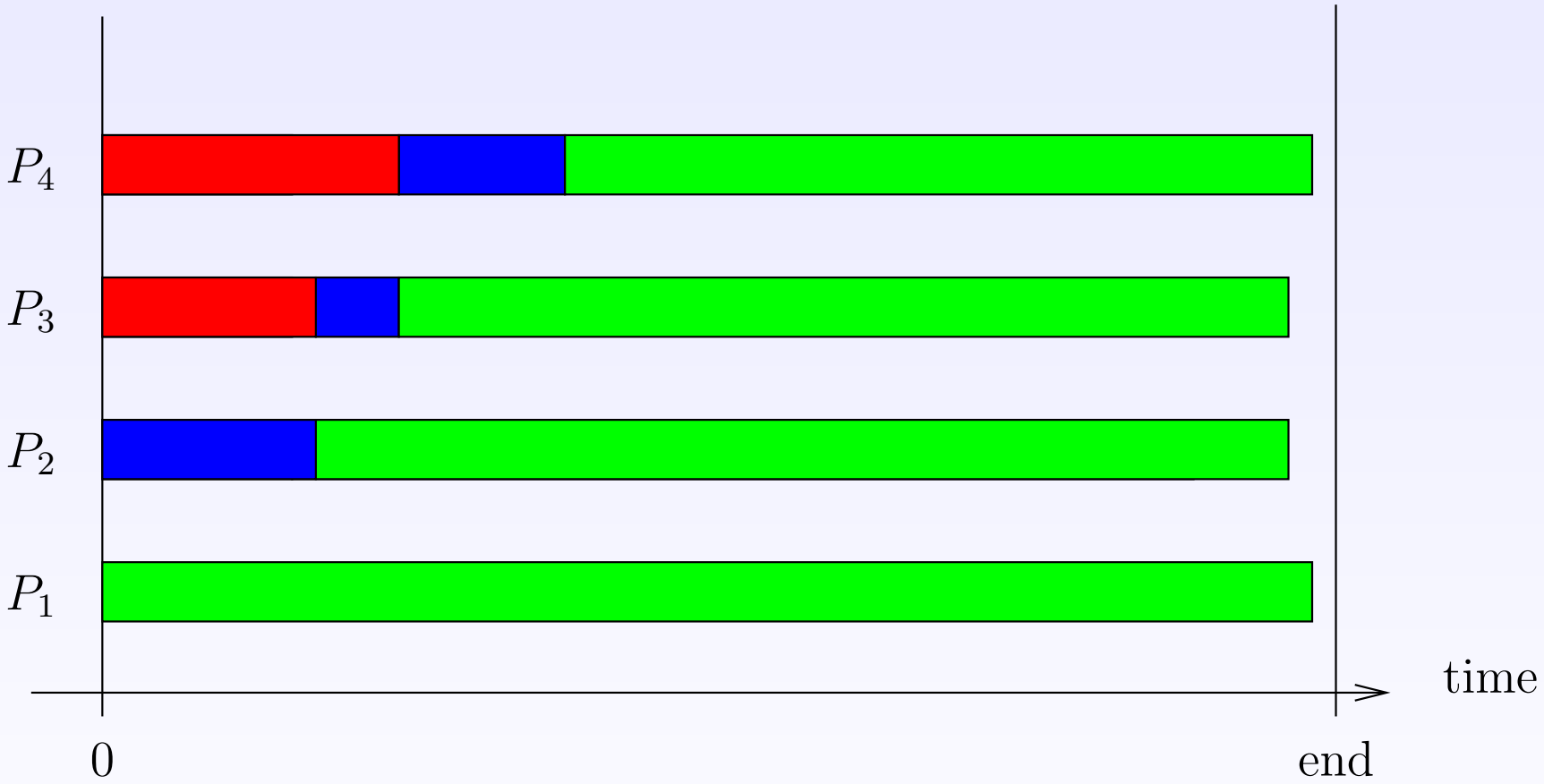
Two slaves i and $i+1$ with $T_i < T_{i+1}$



Communication time for following processors does not change

Proof of Lemma (1/2)

Two slaves i and $i+1$ with $T_i < T_{i+1}$



Get better solution!

Proof of Lemma (2/2)

- Ideally: $T'_i = T'_{i+1}$
  Choose ε such that:
  $(\alpha_i+\varepsilon)W_{\text{total}}(c+w_i) = (\alpha_i+\varepsilon)W_{\text{total}}c+(\alpha_{i+1}-\varepsilon)W_{\text{total}}(c+w_{i+1})$
- If master finishes earlier: absurd
- If master finishes later: similar, decrease its load by ε

Resource selection

Lemma *In any optimal solution, all processors are enrolled*

Proof: simple follow-up of previous Lemma

Resource selection

Lemma *In any optimal solution, all processors are enrolled*

Proof: simple follow-up of previous Lemma

Solution

$$T = \alpha_1 W_{\text{total}} w_1$$

$T = \alpha_2(c + w_2)W_{\text{total}}$. Hence $\alpha_2 = \frac{w_1}{c+w_2}\alpha_1$

$T = (\alpha_2 c + \alpha_3(c + w_3))W_{\text{total}}$. Hence $\alpha_3 = \frac{w_2}{c+w_3}\alpha_2$

$\alpha_i = \frac{w_{i-1}}{c+w_i}\alpha_{i-1}$ for $i \geqslant 2$

$\sum_{i=1}^{n} \alpha_i = 1$

$$\alpha_1 \left(1 + \frac{w_1}{c + w_2} + \ldots + \prod_{k=2}^{j} \frac{w_{k-1}}{c + w_k} + \ldots \right) = 1$$

Solution

$T = \alpha_1 W_{\text{total}} w_1$

$T = \alpha_2 (c + w_2) W_{\text{total}}$. Hence $\alpha_2 = \frac{w_1}{c+w_2} \alpha_1$

$T = (\alpha_2 c + \alpha_3 (c + w_3)) W_{\text{total}}$. Hence $\alpha_3 = \frac{w_2}{c+w_3} \alpha_2$

$\alpha_i = \frac{w_{i-1}}{c+w_i} \alpha_{i-1}$ for $i \geqslant 2$

$\sum_{i=1}^{n} \alpha_i = 1$

$$\alpha_1 \left(1 + \frac{w_1}{c + w_2} + \ldots + \prod_{k=2}^{j} \frac{w_{k-1}}{c + w_k} + \ldots \right) = 1$$

Solution

$T = \alpha_1 W_{\text{total}} w_1$

$T = \alpha_2 (c + w_2) W_{\text{total}}$. Hence $\alpha_2 = \frac{w_1}{c+w_2} \alpha_1$

$T = (\alpha_2 c + \alpha_3 (c + w_3)) W_{\text{total}}$. Hence $\alpha_3 = \frac{w_2}{c+w_3} \alpha_2$

$\alpha_i = \frac{w_{i-1}}{c+w_i} \alpha_{i-1}$ for $i \geqslant 2$

$\sum_{i=1}^{n} \alpha_i = 1$

$$\alpha_1 \left(1 + \frac{w_1}{c + w_2} + \ldots + \prod_{k=2}^{j} \frac{w_{k-1}}{c + w_k} + \ldots \right) = 1$$

Solution

$T = \alpha_1 W_{\text{total}} w_1$

$T = \alpha_2(c + w_2)W_{\text{total}}$. Hence $\alpha_2 = \frac{w_1}{c+w_2}\alpha_1$

$T = (\alpha_2 c + \alpha_3(c + w_3))W_{\text{total}}$. Hence $\alpha_3 = \frac{w_2}{c+w_3}\alpha_2$

$\alpha_i = \frac{w_{i-1}}{c+w_i}\alpha_{i-1}$ for $i \geqslant 2$

$\sum_{i=1}^{n} \alpha_i = 1$

$$\alpha_1 \left(1 + \frac{w_1}{c + w_2} + \ldots + \prod_{k=2}^{j} \frac{w_{k-1}}{c + w_k} + \ldots \right) = 1$$

Solution

$T = \alpha_1 W_{\text{total}} w_1$

$T = \alpha_2 (c + w_2) W_{\text{total}}$. Hence $\alpha_2 = \frac{w_1}{c+w_2} \alpha_1$

$T = (\alpha_2 c + \alpha_3 (c + w_3)) W_{\text{total}}$. Hence $\alpha_3 = \frac{w_2}{c+w_3} \alpha_2$

$\alpha_i = \frac{w_{i-1}}{c+w_i} \alpha_{i-1}$ for $i \geqslant 2$

$\sum_{i=1}^{n} \alpha_i = 1$

$$\alpha_1 \left(1 + \frac{w_1}{c + w_2} + \ldots + \prod_{k=2}^{j} \frac{w_{k-1}}{c + w_k} + \ldots \right) = 1$$

Solution

$T = \alpha_1 W_{\text{total}} w_1$

$T = \alpha_2(c + w_2)W_{\text{total}}$. Hence $\alpha_2 = \frac{w_1}{c+w_2}\alpha_1$

$T = (\alpha_2 c + \alpha_3(c + w_3))W_{\text{total}}$. Hence $\alpha_3 = \frac{w_2}{c+w_3}\alpha_2$

$\alpha_i = \frac{w_{i-1}}{c+w_i}\alpha_{i-1}$ for $i \geqslant 2$

$\sum_{i=1}^{n} \alpha_i = 1$

$$\alpha_1 \left(1 + \frac{w_1}{c + w_2} + \ldots + \prod_{k=2}^{j} \frac{w_{k-1}}{c + w_k} + \ldots \right) = 1$$

Impact of communication ordering

?

No impact!

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c+w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c W_{\text{total}} + \alpha_{i+1}(c+w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$

Processors P_i and P_{i+1}:

$$\alpha_i + \alpha_{i+1} = \frac{c+w_i+w_{i+1}}{(c+w_i)(c+w_{i+1})}$$

No impact!

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c+w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c W_{\text{total}} + \alpha_{i+1}(c+w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$

Processors P_i and P_{i+1}:

$$\alpha_i + \alpha_{i+1} = \frac{c+w_i+w_{i+1}}{(c+w_i)(c+w_{i+1})}$$

No impact!

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c+w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c W_{\text{total}} + \alpha_{i+1}(c+w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$

Processors P_i and P_{i+1}:

$$\alpha_i + \alpha_{i+1} = \frac{c+w_i+w_{i+1}}{(c+w_i)(c+w_{i+1})}$$

No impact!

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c+w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c W_{\text{total}} + \alpha_{i+1}(c+w_{i+1})W_{\text{total}} = T$
Hence $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$

Processors P_i and P_{i+1}:

$$\alpha_i + \alpha_{i+1} = \frac{c + w_i + w_{i+1}}{(c+w_i)(c+w_{i+1})}$$

No impact!

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c+w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c W_{\text{total}} + \alpha_{i+1}(c+w_{i+1})W_{\text{total}} = T$
Hence $\alpha_{i+1} = \frac{1}{c+w_{i+1}}(\frac{T}{W_{\text{total}}} - \alpha_i c) = \frac{w_i}{(c+w_i)(c+w_{i+1})}\frac{T}{W_{\text{total}}}$

Processors P_i and P_{i+1}:

$$\alpha_i + \alpha_{i+1} = \frac{c + w_i + w_{i+1}}{(c+w_i)(c+w_{i+1})}$$

Choosing master processor

Compare processors P_1 and P_2

Processor P_1**:** $\alpha_1 w_1 W_{\text{total}} = T$. Hence $\alpha_1 = \frac{1}{w_1} \frac{T}{W_{\text{total}}}$

Processor P_2**:** $\alpha_2 (c + w_2) W_{\text{total}} = T$. Hence $\alpha_2 = \frac{1}{c+w_2} \frac{T}{W_{\text{total}}}$

Total load processed:

$$\alpha_1 + \alpha_2 = \frac{c + w_1 + w_2}{w_1(c + w_2)} = \frac{c + w_1 + w_2}{cw_1 + w_1 w_2}$$

Load maximal when $w_1 < w_2$
Master = fastest processor

Choosing master processor

Compare processors P_1 and P_2

Processor P_1: $\alpha_1 w_1 W_{\text{total}} = T$. Hence $\alpha_1 = \frac{1}{w_1} \frac{T}{W_{\text{total}}}$

Processor P_2: $\alpha_2 (c + w_2) W_{\text{total}} = T$. Hence $\alpha_2 = \frac{1}{c+w_2} \frac{T}{W_{\text{total}}}$

Total load processed:

$$\alpha_1 + \alpha_2 = \frac{c + w_1 + w_2}{w_1(c + w_2)} = \frac{c + w_1 + w_2}{cw_1 + w_1 w_2}$$

Load maximal when $w_1 < w_2$
Master = fastest processor

Choosing master processor

Compare processors P_1 and P_2

Processor P_1: $\alpha_1 w_1 W_{\text{total}} = T$. Hence $\alpha_1 = \frac{1}{w_1} \frac{T}{W_{\text{total}}}$

Processor P_2: $\alpha_2 (c + w_2) W_{\text{total}} = T$. Hence $\alpha_2 = \frac{1}{c+w_2} \frac{T}{W_{\text{total}}}$

Total load processed:

$$\alpha_1 + \alpha_2 = \frac{c + w_1 + w_2}{w_1(c + w_2)} = \frac{c + w_1 + w_2}{cw_1 + w_1 w_2}$$

Load maximal when $w_1 < w_2$
Master = fastest processor

Choosing master processor

Compare processors P_1 and P_2

Processor P_1: $\alpha_1 w_1 W_{\text{total}} = T$. Hence $\alpha_1 = \frac{1}{w_1} \frac{T}{W_{\text{total}}}$

Processor P_2: $\alpha_2 (c + w_2) W_{\text{total}} = T$. Hence $\alpha_2 = \frac{1}{c+w_2} \frac{T}{W_{\text{total}}}$

Total load processed:

$$\alpha_1 + \alpha_2 = \frac{c + w_1 + w_2}{w_1(c + w_2)} = \frac{c + w_1 + w_2}{cw_1 + w_1 w_2}$$

Load maximal when $w_1 < w_2$
Master = fastest processor

Choosing master processor

Compare processors P_1 and P_2

Processor P_1**:** $\alpha_1 w_1 W_{\text{total}} = T$. Hence $\alpha_1 = \frac{1}{w_1} \frac{T}{W_{\text{total}}}$

Processor P_2**:** $\alpha_2 (c + w_2) W_{\text{total}} = T$. Hence $\alpha_2 = \frac{1}{c+w_2} \frac{T}{W_{\text{total}}}$

Total load processed:

$$\alpha_1 + \alpha_2 = \frac{c + w_1 + w_2}{w_1(c + w_2)} = \frac{c + w_1 + w_2}{cw_1 + w_1 w_2}$$

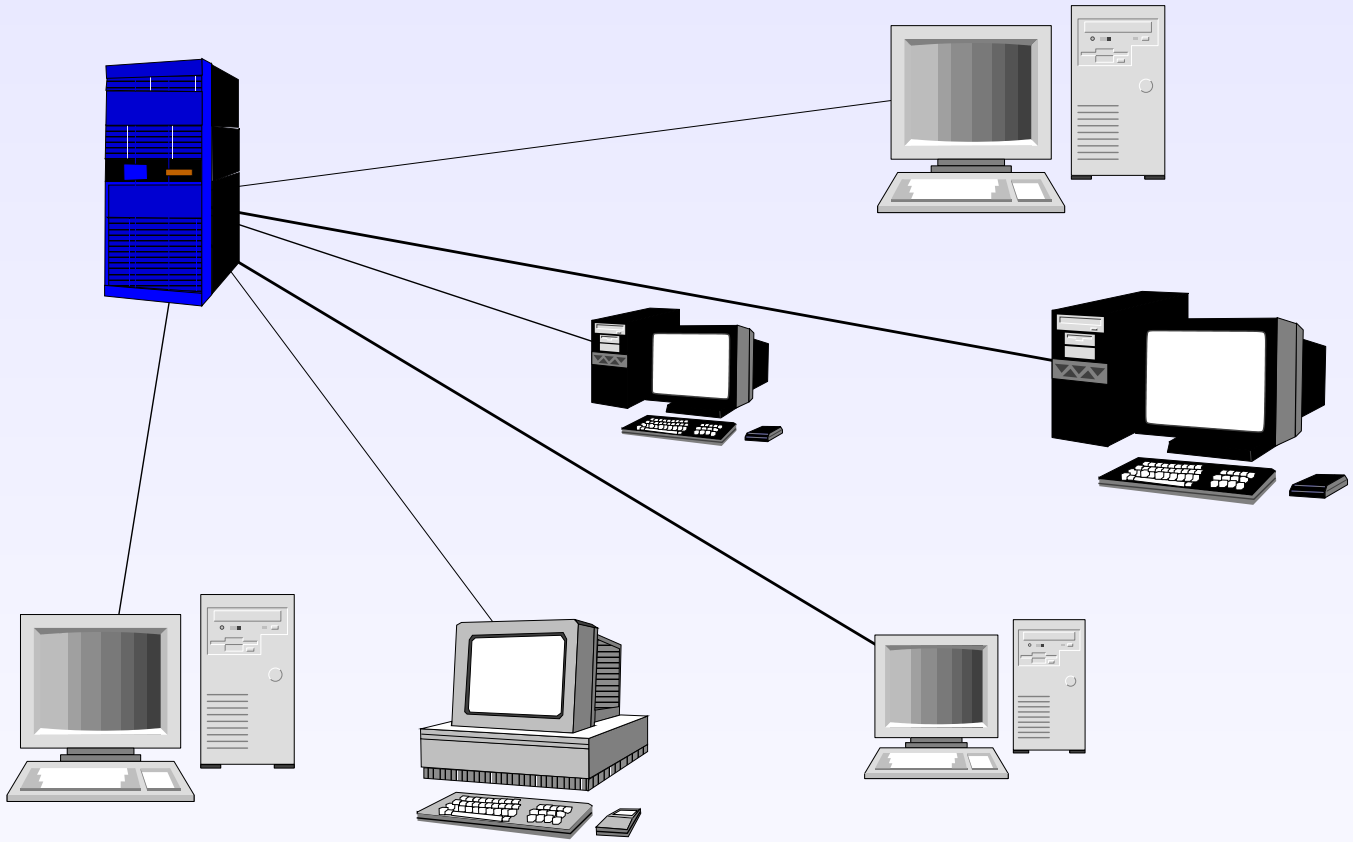
Load maximal when $w_1 < w_2$
Master = fastest processor

Conclusion

- Closed-form formula for execution time and load distribution
- Choice of master
- Ordering of slaves not important
- All processors are enrolled
- *Powerful approach!* ☺

Outline

Star network



- Communication links between master and slaves have *different* bandwidths
- Slaves have *different* computing power

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $\alpha_i W_{\text{total}}$ load units, where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- **Communication time of one load-unit from P_1 to P_i: c_i**
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $\alpha_i W_{\text{total}}$ load units, where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- **Communication time of one load-unit from P_1 to P_i: c_i**
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $\alpha_i W_{\text{total}}$ load units, where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- **Communication time of one load-unit from P_1 to P_i: c_i**
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $\alpha_i W_{\text{total}}$ load units, where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- **Communication time of one load-unit from P_1 to P_i: c_i**
  One-port model: P_1 serially sends one message to each slave

Notations

- Set P_1, ..., P_p of processors
- P_1 is the master, initially holds all data
- Total amount of work: W_{total}
- Processor P_i receives $\alpha_i W_{\text{total}}$ load units,
  where $\alpha_i W_{\text{total}} \in \mathbb{Q}$ and $\sum_i \alpha_i = 1$
  Time for one load unit on P_i: w_i
  Execution time on P_i: $\alpha_i w_i$
- **Communication time of one load-unit from P_1 to P_i:** c_i
  One-port model: P_1 serially sends one message to each slave

Impact of communication ordering

?

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c_i + w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c_i+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1}c_{i+1} = \frac{c_i c_{i+1}+c_{i+1}w_i+c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Serve processors with higher bandwidth first

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c_i + w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c_i+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1}c_{i+1} = \frac{c_i c_{i+1}+c_{i+1}w_i+c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Serve processors with higher bandwidth first

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c_i + w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c_i+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$

Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1}+c_{i+1}w_i+c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Serve processors with higher bandwidth first

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c_i + w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c_i+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$
Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1}+c_{i+1}w_i+c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Serve processors with higher bandwidth first

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c_i + w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c_i+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$
Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1}+c_{i+1}w_i+c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Serve processors with higher bandwidth first

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c_i + w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c_i+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$
Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1}+c_{i+1}w_i+c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Serve processors with higher bandwidth first

Impact of communication ordering

Load processed by P_i and P_{i+1} within time T

Processor P_i: $\alpha_i(c_i + w_i)W_{\text{total}} = T$. Hence $\alpha_i = \frac{1}{c_i+w_i}\frac{T}{W_{\text{total}}}$

Processor P_{i+1}: $\alpha_i c_i W_{\text{total}} + \alpha_{i+1}(c_{i+1} + w_{i+1})W_{\text{total}} = T$
Hence $\alpha_{i+1} = \frac{1}{c_{i+1}+w_{i+1}}(1 - \frac{c_i}{c_i+w_i})\frac{T}{W_{\text{total}}} = \frac{w_i}{(c_i+w_i)(c_{i+1}+w_{i+1})}\frac{T}{W_{\text{total}}}$

Total load processed: $\alpha_i + \alpha_{i+1} = \frac{c_{i+1}+w_i+w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Communication time: $\alpha_i c_i + \alpha_{i+1} c_{i+1} = \frac{c_i c_{i+1}+c_{i+1}w_i+c_i w_{i+1}}{(c_i+w_i)(c_{i+1}+w_{i+1})}$

Serve processors with higher bandwidth first

Resource selection

Lemma *In any optimal solution, all processors are enrolled*

Proof of Lemma

Consider an optimal solution
Let P_k be a non-participating processor
Enroll P_k in the end and assign load α_k s.t.
$\alpha_k(c_k + w_k)W_{\text{total}} =$ execution time of last activated processor

Why place P_k in last position?

Proof of Lemma

Consider an optimal solution
Let P_k be a non-participating processor
Enroll P_k in the end and assign load α_k s.t.
$\alpha_k(c_k + w_k)W_{\text{total}} =$ execution time of last activated processor

Why place P_k in last position?

Load balancing property

Lemma *In any optimal solution, all processors terminate execution simultaneously*

Proof

- Most existing proofs are incorrect
- Either resort to very technical arguments (properties of solutions of linear programs) or to tedious case-by-case analysis

Conclusion

- Closed-form formula for execution time and load distribution
- Serve faster-communicating processors first
- All processors terminate execution simultaneously
- All processors are enrolled
- *Powerful approach!* ☺

Various extensions

Good news One-round, linear model extends to other topologies (e.g. trees, linear chains)

Still open Adding return messages, although very natural, renders problem quite combinatorial

Bad news

- Becomes NP-hard when adding communication/computation latencies
- Unfortunately, adding latencies absolutely needed to deal with multi-round algorithms

Bibliography

- Pionneering book:
  Scheduling divisible loads in parallel and distributed systems, V. Bharadwaj, D. Ghose, V. Mani and T.G. Robertazzi, IEEE Computer Society Press 1996
- Recent survey:
  Ten reasons to use Divisible Load Theory, T.G. Robertazzi, Computer 36, 5 (2003), 63-68
- Bags of tasks:
  Optimal sharing of bags of tasks in heterogeneous clusters, M. Adler, Y. Gong and A.L. Rosenberg, 15th ACM SPAA (2003), 1-10
- Archive of DLS literature:
  http://www.ece.sunysb.edu/~tom/dlt.html

Outline

Outline

Problem

Routing a set of messages from sources to destinations

Given a network, several flows of packets must be routed concurrently
Each flow must be sent from a source to a destination, along a fixed path

Goal: find a schedule which minimizes execution time

Notations

- (V, A) oriented graph, modeling a communication network
- There are n_c flows to be routed
- The k-th flow is denoted as (s_k, t_k, P_k, n_k), where
  - s_k is the source of the packets
  - t_k is their destination
  - P_k is the (fixed) path to follow
  - n_k is the number of packets to be transferred
  - $a_{k,i}$ is the i-th arc of path P_k.

Notations

- (V, A) oriented graph, modeling a communication network
- There are n_c flows to be routed
- The k-th flow is denoted as (s_k, t_k, P_k, n_k), where
  - s_k is the source of the packets
  - t_k is their destination
  - P_k is the (fixed) path to follow
  - n_k is the number of packets to be transferred
  - $a_{k,i}$ is the i-th arc of path P_k.

Notations

- (V, A) oriented graph, modeling a communication network
- There are n_c flows to be routed
- The k-th flow is denoted as (s_k, t_k, P_k, n_k), where
  - s_k is the source of the packets
  - t_k is their destination
  - P_k is the (fixed) path to follow
  - n_k is the number of packets to be transferred
  - $a_{k,i}$ is the i-th arc of path P_k.

Notations

- (V, A) oriented graph, modeling a communication network
- There are n_c flows to be routed
- The k-th flow is denoted as (s_k, t_k, P_k, n_k), where
  - s_k is the source of the packets
  - t_k is their destination
  - P_k is the (fixed) path to follow
  - n_k is the number of packets to be transferred
  - $a_{k,i}$ is the i-th arc of path P_k.

Notations

- (V, A) oriented graph, modeling a communication network
- There are n_c flows to be routed
- The k-th flow is denoted as (s_k, t_k, P_k, n_k), where
  - s_k is the source of the packets
  - t_k is their destination
  - P_k is the (fixed) path to follow
  - n_k is the number of packets to be transferred
  - $a_{k,i}$ is the i-th arc of path P_k.

Notations

- (V, A) oriented graph, modeling a communication network
- There are n_c flows to be routed
- The k-th flow is denoted as (s_k, t_k, P_k, n_k), where
  - s_k is the source of the packets
  - t_k is their destination
  - P_k is the (fixed) path to follow
  - n_k is the number of packets to be transferred
  - $a_{k,i}$ is the i-th arc of path P_k.

Notations

- (V, A) oriented graph, modeling a communication network
- There are n_c flows to be routed
- The k-th flow is denoted as (s_k, t_k, P_k, n_k), where
  - s_k is the source of the packets
  - t_k is their destination
  - P_k is the (fixed) path to follow
  - n_k is the number of packets to be transferred
  - $a_{k,i}$ is the i-th arc of path P_k.

Notations

- (V, A) oriented graph, modeling a communication network
- There are n_c flows to be routed
- The k-th flow is denoted as (s_k, t_k, P_k, n_k), where
  - s_k is the source of the packets
  - t_k is their destination
  - P_k is the (fixed) path to follow
  - n_k is the number of packets to be transferred
  - $a_{k,i}$ is the i-th arc of path P_k.

Hypotheses

- A packet crosses an edge within one time-step
- At any time-step, at most one packet crosses an edge
- Scheduling: for each time-step, decide which packet crosses any given edge

Hypotheses

- A packet crosses an edge within one time-step
- At any time-step, at most one packet crosses an edge
- Scheduling: for each time-step, decide which packet crosses any given edge

Hypotheses

- A packet crosses an edge within one time-step
- At any time-step, at most one packet crosses an edge
- Scheduling: for each time-step, decide which packet crosses any given edge

Lower bound

Congestion C_a of edge $a \in A$ = total number of packets that cross a

$$C_a = \sum_{k \mid a \in P_k} n_k \qquad C_{\max} = \max_a C_a$$

$C_{\max}$ lower bound on schedule makespan

$$C^* \geqslant C_{\max}$$

$\Rightarrow$ "Fluidified" solution in $C_{\max}$?

Lower bound

Congestion C_a of edge $a \in A$ = total number of packets that cross a

$$C_a = \sum_{k \mid a \in P_k} n_k \qquad C_{\max} = \max_a C_a$$

$C_{\max}$ lower bound on schedule makespan

$$C^* \geqslant C_{\max}$$

$\Rightarrow$ "Fluidified" solution in $C_{\max}$?

Lower bound

Congestion C_a of edge $a \in A$ = total number of packets that cross a

$$C_a = \sum_{k \mid a \in P_k} n_k \qquad C_{\max} = \max_a C_a$$

$C_{\max}$ lower bound on schedule makespan

$$C^* \geqslant C_{\max}$$

$\Rightarrow$ "Fluidified" solution in $C_{\max}$?

Notations

Main idea

- Look for rational, not integer, solution

- $n_{k,i}(t)$ (rational) number of packets waiting to cross edge $a_{k,i}$ (i-th arc of P_k) at time t

- $T_{k,i}(t)$ total time spent by $a_{k,i}$ to route packets of k-th flow during time-interval $[0; t]$

Notations

Main idea

- Look for rational, not integer, solution
- $n_{k,i}(t)$ (rational) number of packets waiting to cross edge $a_{k,i}$ (i-th arc of P_k) at time t
- $T_{k,i}(t)$ total time spent by $a_{k,i}$ to route packets of k-th flow during time-interval $[0; t]$

Notations

Main idea

- Look for rational, not integer, solution

- $n_{k,i}(t)$ (rational) number of packets waiting to cross edge $a_{k,i}$ (i-th arc of P_k) at time t

- $T_{k,i}(t)$ total time spent by $a_{k,i}$ to route packets of k-th flow during time-interval $[0; t]$

Equations

1. Initialization

$$n_{k,1}(t) = n_k - T_{k,1}(t), \qquad \text{for all } k$$

2. Conservation law

$$n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad \text{for all } k$$

3. Resource constraints

$$\sum_{(k,i) \mid a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) \leqslant t_2 - t_1, \forall a \in A, \forall t_2 \geqslant t_1 \geqslant 0$$

4. Objective function

$$\text{MINIMIZE } C_{\text{frac}} = \int_0^{\infty} \mathbb{1} \left(\sum_{k,i} n_{k,i}(t) \right) dt$$

Equations

1. Initialization

$$n_{k,1}(t) = n_k - T_{k,1}(t), \qquad \text{for all } k$$

2. Conservation law

$$n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad \text{for all } k$$

3. Resource constraints

$$\sum_{(k,i) \mid a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) \leqslant t_2 - t_1, \forall a \in A, \forall t_2 \geqslant t_1 \geqslant 0$$

4. Objective function

$$\text{MINIMIZE } C_{\text{frac}} = \int_0^{\infty} \mathbb{1} \left(\sum_{k,i} n_{k,i}(t) \right) dt$$

Equations

1. Initialization

$$n_{k,1}(t) = n_k - T_{k,1}(t), \qquad \text{for all } k$$

2. Conservation law

$$n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad \text{for all } k$$

3. Resource constraints

$$\sum_{(k,i) \mid a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) \leqslant t_2 - t_1, \forall a \in A, \forall t_2 \geqslant t_1 \geqslant 0$$

4. Objective function

$$\text{MINIMIZE } C_{\text{frac}} = \int_0^{\infty} \mathbb{1} \left(\sum_{k,i} n_{k,i}(t) \right) dt$$

Equations

1. Initialization

$$n_{k,1}(t) = n_k - T_{k,1}(t), \qquad \text{for all } k$$

2. Conservation law

$$n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t), \qquad \text{for all } k$$

3. Resource constraints

$$\sum_{(k,i) \mid a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) \leqslant t_2 - t_1, \forall a \in A, \forall t_2 \geqslant t_1 \geqslant 0$$

4. Objective function

$$\text{MINIMIZE } C_{\text{frac}} = \int_0^\infty \mathbb{1} \left(\sum_{k,i} n_{k,i}(t) \right) dt$$

Lower bound

- $n_{k,1}(t) = n_k - T_{k,1}(t)$, for all k
- $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t)$, for all k
- At any time-step t, $\sum_{j=1}^{i} n_{k,j}(t) = n_k - T_{k,i}(t)$
- For each edge a:

$$\sum_{(k,i)|a_{k,i}=a} \sum_{j=1}^{i} n_{k,j}(t) = \sum_{(k,i)|a_{k,i}=a} n_k - \sum_{(k,i)|a_{k,i}=a} T_{k,i}(t) \geqslant C_a - t$$

While $t < C_a$, there remains packets inside the network

Hence $C_{\text{frac}} \geqslant \max_a C_a = C_{\max}$

Lower bound

- $n_{k,1}(t) = n_k - T_{k,1}(t)$, for all k
- $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t)$, for all k
- At any time-step t, $\sum_{j=1}^{i} n_{k,j}(t) = n_k - T_{k,i}(t)$
- For each edge a:

$$\sum_{(k,i)|a_{k,i}=a} \sum_{j=1}^{i} n_{k,j}(t) = \sum_{(k,i)|a_{k,i}=a} n_k - \sum_{(k,i)|a_{k,i}=a} T_{k,i}(t) \geqslant C_a - t$$

While $t < C_a$, there remains packets inside the network

Hence $C_{\text{frac}} \geqslant \max_a C_a = C_{\max}$

Lower bound

- $n_{k,1}(t) = n_k - T_{k,1}(t)$, for all k
- $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t)$, for all k
- At any time-step t, $\sum_{j=1}^{i} n_{k,j}(t) = n_k - T_{k,i}(t)$
- For each edge a:

$$\sum_{(k,i)|a_{k,i}=a} \sum_{j=1}^{i} n_{k,j}(t) = \sum_{(k,i)|a_{k,i}=a} n_k - \sum_{(k,i)|a_{k,i}=a} T_{k,i}(t) \geqslant C_a - t$$

While $t < C_a$, there remains packets inside the network

Hence $C_{\text{frac}} \geqslant \max_a C_a = C_{\max}$

Lower bound

- $n_{k,1}(t) = n_k - T_{k,1}(t)$, for all k
- $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t)$, for all k
- At any time-step t, $\sum_{j=1}^{i} n_{k,j}(t) = n_k - T_{k,i}(t)$
- For each edge a:

$$\sum_{(k,i)|a_{k,i}=a} \sum_{j=1}^{i} n_{k,j}(t) = \sum_{(k,i)|a_{k,i}=a} n_k - \sum_{(k,i)|a_{k,i}=a} T_{k,i}(t) \geqslant C_a - t$$

While $t < C_a$, there remains packets inside the network

Hence $C_{\text{frac}} \geqslant \max_a C_a = C_{\max}$

Lower bound

- $n_{k,1}(t) = n_k - T_{k,1}(t)$, for all k
- $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t)$, for all k
- At any time-step t, $\sum_{j=1}^{i} n_{k,j}(t) = n_k - T_{k,i}(t)$
- For each edge a:

$$\sum_{(k,i)|a_{k,i}=a} \sum_{j=1}^{i} n_{k,j}(t) = \sum_{(k,i)|a_{k,i}=a} n_k - \sum_{(k,i)|a_{k,i}=a} T_{k,i}(t) \geqslant C_a - t$$

While $t < C_a$, there remains packets inside the network

Hence $C_{\text{frac}} \geqslant \max_a C_a = C_{\max}$

Lower bound

- $n_{k,1}(t) = n_k - T_{k,1}(t)$, for all k
- $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t)$, for all k
- At any time-step t, $\sum_{j=1}^{i} n_{k,j}(t) = n_k - T_{k,i}(t)$
- For each edge a:

$$\sum_{(k,i)|a_{k,i}=a} \sum_{j=1}^{i} n_{k,j}(t) = \sum_{(k,i)|a_{k,i}=a} n_k - \sum_{(k,i)|a_{k,i}=a} T_{k,i}(t) \geqslant C_a - t$$

While $t < C_a$, there remains packets inside the network

Hence $C_{\text{frac}} \geqslant \max_a C_a = C_{\max}$

A candidate solution

For $t \leqslant C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for all k and i
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \qquad \forall k$
- $n_{k,i}(t) = 0$, for all k et $i \geqslant 2$.

For $t \geqslant C_{\max}$

- $T_{k,i}(t) = n_k$
- $n_{k,i}(t) = 0$

This solution is a schedule of makespan $C_{\max}$. Is it feasible?

A candidate solution

For $t \leqslant C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for all k and i
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \qquad \forall k$
- $n_{k,i}(t) = 0$, for all k et $i \geqslant 2$.

For $t \geqslant C_{\max}$

- $T_{k,i}(t) = n_k$
- $n_{k,i}(t) = 0$

This solution is a schedule of makespan $C_{\max}$. Is it feasible?

A candidate solution

For $t \leqslant C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for all k and i
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \qquad \forall k$
- $n_{k,i}(t) = 0$, for all k et $i \geqslant 2$.

For $t \geqslant C_{\max}$

- $T_{k,i}(t) = n_k$
- $n_{k,i}(t) = 0$

This solution is a schedule of makespan $C_{\max}$. Is it feasible?

A candidate solution

For $t \leqslant C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for all k and i
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \qquad \forall k$
- $n_{k,i}(t) = 0$, for all k et $i \geqslant 2$.

For $t \geqslant C_{\max}$

- $T_{k,i}(t) = n_k$
- $n_{k,i}(t) = 0$

This solution is a schedule of makespan $C_{\max}$. Is it feasible?

A candidate solution

For $t \leqslant C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for all k and i
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \qquad \forall k$
- $n_{k,i}(t) = 0$, for all k et $i \geqslant 2$.

For $t \geqslant C_{\max}$

- $T_{k,i}(t) = n_k$
- $n_{k,i}(t) = 0$

This solution is a schedule of makespan $C_{\max}$. Is it feasible?

A candidate solution

For $t \leqslant C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for all k and i
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \qquad \forall k$
- $n_{k,i}(t) = 0$, for all k et $i \geqslant 2$.

For $t \geqslant C_{\max}$

- $T_{k,i}(t) = n_k$
- $n_{k,i}(t) = 0$

This solution is a schedule of makespan $C_{\max}$. Is it feasible?

A candidate solution

For $t \leqslant C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for all k and i
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \qquad \forall k$
- $n_{k,i}(t) = 0$, for all k et $i \geqslant 2$.

For $t \geqslant C_{\max}$

- $T_{k,i}(t) = n_k$
- $n_{k,i}(t) = 0$

This solution is a schedule of makespan $C_{\max}$. Is it feasible?

A candidate solution

For $t \leqslant C_{\max}$

- $T_{k,i}(t) = \dfrac{n_k}{C_{\max}} t$, for all k and i
- $n_{k,1}(t) = n_k - T_{k,1}(t) = n_k - \dfrac{n_k}{C_{\max}} t = n_k \left(1 - \dfrac{t}{C_{\max}}\right), \qquad \forall k$
- $n_{k,i}(t) = 0$, for all k et $i \geqslant 2$.

For $t \geqslant C_{\max}$

- $T_{k,i}(t) = n_k$
- $n_{k,i}(t) = 0$

This solution is a schedule of makespan $C_{\max}$. Is it feasible?

Verifying solution (for $t \leqslant C_{\max}$)

1. $n_{k,1}(t) = n_k - T_{k,1}(t)$, for all k
   OK by definition.

2. $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t)$, for all k
   $T_{k,i}(t) - T_{k,i+1}(t) = \frac{n_k}{C_{\max}}t - \frac{n_k}{C_{\max}}t = 0 = n_{k,i+1}(t)$

3. $\sum_{(k,i) \mid a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) \leqslant t_2 - t_1, \forall a \in A, \forall t_2 \geqslant t_1 \geqslant 0$
   $\sum_{(k,i) \mid a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) = \sum_{(k,i) \mid a_{k,i}=a} \frac{n_k}{C_{\max}}(t_2 - t_1) =$
   $\frac{C_a}{C_{\max}}(t_2 - t_1) \leqslant t_2 - t_1$

Verifying solution (for $t \leqslant C_{\max}$)

1. $n_{k,1}(t) = n_k - T_{k,1}(t)$, for all k
   OK by definition.

2. $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t)$, for all k
   $T_{k,i}(t) - T_{k,i+1}(t) = \frac{n_k}{C_{\max}}t - \frac{n_k}{C_{\max}}t = 0 = n_{k,i+1}(t)$

3. $\sum_{(k,i) \mid a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) \leqslant t_2 - t_1, \forall a \in A, \forall t_2 \geqslant t_1 \geqslant 0$
   $\sum_{(k,i) \mid a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) = \sum_{(k,i) \mid a_{k,i}=a} \frac{n_k}{C_{\max}}(t_2 - t_1) =$
   $\frac{C_a}{C_{\max}}(t_2 - t_1) \leqslant t_2 - t_1$

Verifying solution (for $t \leqslant C_{\max}$)

1. $n_{k,1}(t) = n_k - T_{k,1}(t)$, for all k
   OK by definition.

2. $n_{k,i+1}(t) = T_{k,i}(t) - T_{k,i+1}(t)$, for all k
   $T_{k,i}(t) - T_{k,i+1}(t) = \frac{n_k}{C_{\max}}t - \frac{n_k}{C_{\max}}t = 0 = n_{k,i+1}(t)$

3. $$\sum_{(k,i) \mid a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) \leqslant t_2 - t_1, \forall a \in A, \forall t_2 \geqslant t_1 \geqslant 0$$
   $$\sum_{(k,i) \mid a_{k,i}=a} T_{k,i}(t_2) - T_{k,i}(t_1) = \sum_{(k,i) \mid a_{k,i}=a} \frac{n_k}{C_{\max}}(t_2 - t_1) =$$
   $$\frac{C_a}{C_{\max}}(t_2 - t_1) \leqslant t_2 - t_1$$

Rounds for periodic schedule

- $\Omega \approx$ length of a round (to be defined later)
- m_k: number of packets of k-th flow distributed within one round

$$m_k = \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil .$$

- $D_a = \sum_{(k,i)|a_{k,i}=a} 1 = |\{k|a \in P_k\}|$

$$D_{\max} = \max_a D_a \leqslant n_c$$

- Schedule period: $\Omega + D_{\max}$.

Rounds for periodic schedule

- $\Omega \approx$ length of a round (to be defined later)
- m_k: number of packets of k-th flow distributed within one round

$$m_k = \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil .$$

- $D_a = \sum_{(k,i)|a_{k,i}=a} 1 = |\{k|a \in P_k\}|$

$$D_{\max} = \max_a D_a \leqslant n_c$$

- Schedule period: $\Omega + D_{\max}$.

Rounds for periodic schedule

- $\Omega \approx$ length of a round (to be defined later)
- m_k: number of packets of k-th flow distributed within one round

$$m_k = \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil .$$

- $D_a = \sum_{(k,i)|a_{k,i}=a} 1 = |\{k|a \in P_k\}|$

$$D_{\max} = \max_a D_a \leqslant n_c$$

- Schedule period: $\Omega + D_{\max}$.

Rounds for periodic schedule

- $\Omega \approx$ length of a round (to be defined later)
- m_k: number of packets of k-th flow distributed within one round

$$m_k = \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil .$$

- $D_a = \sum_{(k,i)|a_{k,i}=a} 1 = |\{k|a \in P_k\}|$

$$D_{\max} = \max_a D_a \leqslant n_c$$

- Schedule period: $\Omega + D_{\max}$.

Periodic schedule

During time-interval $[j(\Omega + D_{\max}); (j+1)(\Omega + D_{\max})]$:

Edge a transfers m_k packets of k-th flow if there exists i such that $a_{k,i} = a$

Edge a remains idle during

$$\Omega + D_{\max} - \sum_{(k,i)|a_{k,i}=a} m_k$$

(If less than m_k packets are waiting at edge a at time-step $j(\Omega + D_{\max})$, a transfers what is available and remains idle for a longer time)

Periodic schedule

During time-interval $[j(\Omega + D_{\max}); (j+1)(\Omega + D_{\max})]$:

Edge a transfers m_k packets of k-th flow if there exists i such that $a_{k,i} = a$

Edge a remains idle during

$$\Omega + D_{\max} - \sum_{(k,i)|a_{k,i}=a} m_k$$

(If less than m_k packets are waiting at edge a at time-step $j(\Omega + D_{\max})$, a transfers what is available and remains idle for a longer time)

Periodic schedule

During time-interval $[j(\Omega + D_{\max}); (j+1)(\Omega + D_{\max})]$:

Edge a transfers m_k packets of k-th flow if there exists i such that $a_{k,i} = a$

Edge a remains idle during

$$\Omega + D_{\max} - \sum_{(k,i)|a_{k,i}=a} m_k$$

(If less than m_k packets are waiting at edge a at time-step $j(\Omega + D_{\max})$, a transfers what is available and remains idle for a longer time)

Periodic schedule

During time-interval $[j(\Omega + D_{\max}); (j+1)(\Omega + D_{\max})]$:

Edge a transfers m_k packets of k-th flow if there exists i such that $a_{k,i} = a$

Edge a remains idle during

$$\Omega + D_{\max} - \sum_{(k,i)|a_{k,i}=a} m_k$$

(If less than m_k packets are waiting at edge a at time-step $j(\Omega + D_{\max})$, a transfers what is available and remains idle for a longer time)

Feasibility

$$\sum_{(k,i)|a_{k,i}=a} m_k = \sum_{(k,i)|a_{k,i}=a} \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil$$

$$\leqslant \sum_{(k,i)|a_{k,i}=a} \left(\frac{n_k \Omega}{C_{\max}} + 1 \right)$$

$$\leqslant \frac{C_a}{C_{\max}} \Omega + D_a$$

$$\leqslant \Omega + D_{\max}$$

Feasibility

$$\sum_{(k,i)|a_{k,i}=a} m_k = \sum_{(k,i)|a_{k,i}=a} \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil$$

$$\leqslant \sum_{(k,i)|a_{k,i}=a} \left(\frac{n_k \Omega}{C_{\max}} + 1 \right)$$

$$\leqslant \frac{C_a}{C_{\max}} \Omega + D_a$$

$$\leqslant \Omega + D_{\max}$$

Feasibility

$$\sum_{(k,i)|a_{k,i}=a} m_k = \sum_{(k,i)|a_{k,i}=a} \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil$$

$$\leqslant \sum_{(k,i)|a_{k,i}=a} \left(\frac{n_k \Omega}{C_{\max}} + 1 \right)$$

$$\leqslant \frac{C_a}{C_{\max}} \Omega + D_a$$

$$\leqslant \Omega + D_{\max}$$

Feasibility

$$\begin{aligned}
\sum_{(k,i)|a_{k,i}=a} m_k &= \sum_{(k,i)|a_{k,i}=a} \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil \\
&\leqslant \sum_{(k,i)|a_{k,i}=a} \left(\frac{n_k \Omega}{C_{\max}} + 1 \right) \\
&\leqslant \frac{C_a}{C_{\max}} \Omega + D_a \\
&\leqslant \Omega + D_{\max}
\end{aligned}$$

Feasibility

$$\begin{aligned}
\sum_{(k,i)|a_{k,i}=a} m_k &= \sum_{(k,i)|a_{k,i}=a} \left\lceil \frac{n_k \Omega}{C_{\max}} \right\rceil \\
&\leqslant \sum_{(k,i)|a_{k,i}=a} \left(\frac{n_k \Omega}{C_{\max}} + 1 \right) \\
&\leqslant \frac{C_a}{C_{\max}} \Omega + D_a \\
&\leqslant \Omega + D_{\max}
\end{aligned}$$

Sources

- $N_{k,i}(t)$: number of packets waiting to cross edge $a_{k,i}$ at time t
- $a_{k,1}$ sends m_k packets during $[0, \Omega + D_{\max}]$.
  $N_{k,1}(\Omega + D_{\max}) = n_k - m_k$
- $a_{k,1}$ sends m_k packets during $[\Omega + D_{\max}, 2(\Omega + D_{\max})]$.
  $N_{k,1}(2(\Omega + D_{\max})) = n_k - 2m_k$
- Let $T = \left\lceil \frac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max})$

$$N_{k,1}(T) \leqslant n_k - \frac{T}{\Omega + D_{\max}} m_k \leqslant n_k - \frac{n_k \Omega}{C_{\max}} \frac{C_{\max}}{\Omega} = 0$$

Sources

- $N_{k,i}(t)$: number of packets waiting to cross edge $a_{k,i}$ at time t
- $a_{k,1}$ sends m_k packets during $[0, \Omega + D_{\max}]$.
  $N_{k,1}(\Omega + D_{\max}) = n_k - m_k$
- $a_{k,1}$ sends m_k packets during $[\Omega + D_{\max}, 2(\Omega + D_{\max})]$.
  $N_{k,1}(2(\Omega + D_{\max})) = n_k - 2m_k$
- Let $T = \left\lceil \frac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max})$

$$N_{k,1}(T) \leqslant n_k - \frac{T}{\Omega + D_{\max}} m_k \leqslant n_k - \frac{n_k \Omega}{C_{\max}} \frac{C_{\max}}{\Omega} = 0$$

Sources

- $N_{k,i}(t)$: number of packets waiting to cross edge $a_{k,i}$ at time t
- $a_{k,1}$ sends m_k packets during $[0, \Omega + D_{\max}]$.
  $N_{k,1}(\Omega + D_{\max}) = n_k - m_k$
- $a_{k,1}$ sends m_k packets during $[\Omega + D_{\max}, 2(\Omega + D_{\max})]$.
  $N_{k,1}(2(\Omega + D_{\max})) = n_k - 2m_k$
- Let $T = \left\lceil \frac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max})$

$$N_{k,1}(T) \leqslant n_k - \frac{T}{\Omega + D_{\max}} m_k \leqslant n_k - \frac{n_k \Omega}{C_{\max}} \frac{C_{\max}}{\Omega} = 0$$

Sources

- $N_{k,i}(t)$: number of packets waiting to cross edge $a_{k,i}$ at time t
- $a_{k,1}$ sends m_k packets during $[0, \Omega + D_{\max}]$.
  $N_{k,1}(\Omega + D_{\max}) = n_k - m_k$
- $a_{k,1}$ sends m_k packets during $[\Omega + D_{\max}, 2(\Omega + D_{\max})]$.
  $N_{k,1}(2(\Omega + D_{\max})) = n_k - 2m_k$
- Let $T = \left\lceil \frac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max})$

$$N_{k,1}(T) \leqslant n_k - \frac{T}{\Omega + D_{\max}} m_k \leqslant n_k - \frac{n_k \Omega}{C_{\max}} \frac{C_{\max}}{\Omega} = 0$$

Propagation delay

- $a_{k,1}$ sends m_k packets during $[0, \Omega + D_{\max}]$.
  $N_{k,1}(\Omega + D_{\max}) = n_k - m_k$ $\quad$ $N_{k,2}(\Omega + D_{\max}) = m_k$
  $N_{k,i\geqslant 3}(\Omega + D_{\max}) = 0$

- $a_{k,1}$ sends m_k packets during $[\Omega + D_{\max}, 2(\Omega + D_{\max})]$.
  $N_{k,1}(2(\Omega + D_{\max})) = n_k - 2m_k$ $\quad$ $N_{k,2}(2(\Omega + D_{\max})) = m_k$
  $N_{k,3}(2(\Omega + D_{\max})) = m_k$ $\quad$ $N_{k,i\geqslant 4}(2(\Omega + D_{\max})) = 0$

- Delay between time when a packet crosses first edge of P_k and time when it crosses last edge is at most
  $$(|P_k| - 1)(\Omega + D_{\max})$$
  Let $L = \max_k |P_k|$

Makespan

$$C_{\text{total}} \leqslant T + (L-1)(\Omega + D_{\max})$$

$$= \left\lceil \frac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max})$$

$$\leqslant \left(\frac{C_{\max}}{\Omega} + 1 \right) (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max})$$

$$= C_{\max} + LD_{\max} + \frac{D_{\max} C_{\max}}{\Omega} + L\Omega$$

Upper bound minimum if $\Omega = \sqrt{\frac{D_{\max} C_{\max}}{L}}$

$$C_{\text{total}} \leqslant C_{\max} + 2\sqrt{C_{\max} D_{\max} L} + D_{\max} L$$

Makespan

$$C_{\text{total}} \leqslant T + (L-1)(\Omega + D_{\max})$$
$$= \left\lceil \frac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max})$$
$$\leqslant \left(\frac{C_{\max}}{\Omega} + 1 \right) (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max})$$
$$= C_{\max} + LD_{\max} + \frac{D_{\max} C_{\max}}{\Omega} + L\Omega$$

Upper bound minimum if $\Omega = \sqrt{\frac{D_{\max} C_{\max}}{L}}$

$$C_{\text{total}} \leqslant C_{\max} + 2\sqrt{C_{\max} D_{\max} L} + D_{\max} L$$

Makespan

$$
\begin{aligned}
C_{\text{total}} &\leqslant T + (L-1)(\Omega + D_{\max}) \\
&= \left\lceil \frac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max}) \\
&\leqslant \left(\frac{C_{\max}}{\Omega} + 1 \right) (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max}) \\
&= C_{\max} + L D_{\max} + \frac{D_{\max} C_{\max}}{\Omega} + L\Omega
\end{aligned}
$$

Upper bound minimum if $\Omega = \sqrt{\dfrac{D_{\max} C_{\max}}{L}}$

$$C_{\text{total}} \leqslant C_{\max} + 2\sqrt{C_{\max} D_{\max} L} + D_{\max} L$$

Makespan

$$
\begin{aligned}
C_{\text{total}} &\leqslant T + (L-1)(\Omega + D_{\max}) \\
&= \left\lceil \frac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max}) \\
&\leqslant \left(\frac{C_{\max}}{\Omega} + 1 \right) (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max}) \\
&= C_{\max} + LD_{\max} + \frac{D_{\max} C_{\max}}{\Omega} + L\Omega
\end{aligned}
$$

Upper bound minimum if $\Omega = \sqrt{\frac{D_{\max} C_{\max}}{L}}$

$$C_{\text{total}} \leqslant C_{\max} + 2\sqrt{C_{\max} D_{\max} L} + D_{\max} L$$

Makespan

$$
\begin{aligned}
C_{\text{total}} &\leqslant T + (L-1)(\Omega + D_{\max}) \\
&= \left\lceil \frac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max}) \\
&\leqslant \left(\frac{C_{\max}}{\Omega} + 1 \right) (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max}) \\
&= C_{\max} + LD_{\max} + \frac{D_{\max} C_{\max}}{\Omega} + L\Omega
\end{aligned}
$$

Upper bound minimum if $\Omega = \sqrt{\dfrac{D_{\max} C_{\max}}{L}}$

$$C_{\text{total}} \leqslant C_{\max} + 2\sqrt{C_{\max} D_{\max} L} + D_{\max} L$$

Makespan

$$
\begin{aligned}
C_{\text{total}} &\leqslant T + (L-1)(\Omega + D_{\max}) \\
&= \left\lceil \frac{C_{\max}}{\Omega} \right\rceil (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max}) \\
&\leqslant \left(\frac{C_{\max}}{\Omega} + 1 \right) (\Omega + D_{\max}) + (L-1)(\Omega + D_{\max}) \\
&= C_{\max} + LD_{\max} + \frac{D_{\max} C_{\max}}{\Omega} + L\Omega
\end{aligned}
$$

Upper bound minimum if $\Omega = \sqrt{\frac{D_{\max} C_{\max}}{L}}$

$$C_{\text{total}} \leqslant C_{\max} + 2\sqrt{C_{\max} D_{\max} L} + D_{\max} L$$

Asymptotic optimality

$$C_{\max} \leqslant C^* \leqslant C_{\text{total}} \leqslant C_{\max} + 2\sqrt{C_{\max} D_{\max} L} + D_{\max} L$$

$$1 \leqslant \frac{C_{\text{total}}}{C_{\max}} \leqslant 1 + 2\sqrt{\frac{D_{\max} L}{C_{\max}}} + \frac{D_{\max} L}{C_{\max}}$$

$$\text{with } \Omega = \sqrt{\frac{D_{\max} C_{\max}}{L}}$$

Resources

$$\sum_{(k,i)|a_{k,i}=a,k\geqslant 2} m_k \leqslant \sum_{(k,i)|a_{k,i}=a,k\geqslant 2} \left(\frac{n_k}{C_{\max}} \sqrt{\frac{D_{\max} C_{\max}}{L}} + 1 \right)$$

$$\leqslant \sqrt{\frac{D_{\max} C_{\max}}{L}} + D_{\max}$$

Conclusion

- Neglect initialization and clean-up phases
- Rational solution of steady-state equations
- Rounds of length square root of solution
  - Each round "loses" a constant number of time-steps
  - Sum of "lost" time-steps grows more slowly than schedule rate
  - Buffers of size square root of solution

Outline

Problem

- Same problem as before, but paths are not fixed a priori
- There are n_c collections of packets to be routed
- Each flow is routed via a *set* of flows (packets of a same collection may follow different paths)
- $n^{k,l}$: total number of packets to be routed from k to l
- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)

Congestion: $C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$ $\quad C_{\max} = \max_{i,j} C_{i,j}$

Problem

- Same problem as before, but paths are not fixed a priori
- There are n_c collections of packets to be routed
- Each flow is routed via a *set* of flows (packets of a same collection may follow different paths)
- $n^{k,l}$: total number of packets to be routed from k to l
- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i, j)

Congestion: $C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$ $\quad C_{\max} = \max_{i,j} C_{i,j}$

Problem

- Same problem as before, but paths are not fixed a priori
- There are n_c collections of packets to be routed
- Each flow is routed via a *set* of flows (packets of a same collection may follow different paths)
- $n^{k,l}$: total number of packets to be routed from k to l
- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)

Congestion: $C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$ $\quad C_{\max} = \max_{i,j} C_{i,j}$

Problem

- Same problem as before, but paths are not fixed a priori
- There are n_c collections of packets to be routed
- Each flow is routed via a *set* of flows (packets of a same collection may follow different paths)
- $n^{k,l}$: total number of packets to be routed from k to l
- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)

Congestion: $C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$ $\quad C_{\max} = \max_{i,j} C_{i,j}$

Problem

- Same problem as before, but paths are not fixed a priori
- There are n_c collections of packets to be routed
- Each flow is routed via a *set* of flows (packets of a same collection may follow different paths)
- $n^{k,l}$: total number of packets to be routed from k to l
- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i, j)

Congestion: $C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$ $\qquad C_{\max} = \max_{i,j} C_{i,j}$

Problem

- Same problem as before, but paths are not fixed a priori
- There are n_c collections of packets to be routed
- Each flow is routed via a *set* of flows (packets of a same collection may follow different paths)
- $n^{k,l}$: total number of packets to be routed from k to l
- $n_{i,j}^{k,l}$: total number of packets routed from k to l and crossing edge (i,j)

  Congestion: $C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$ $\qquad C_{\max} = \max_{i,j} C_{i,j}$

Equations (1/2)

1. Initialization

$$\sum_{j|(k,j)\in A} n_{k,j}^{k,l} = n^{k,l}$$

2. Reception

$$\sum_{i|(i,l)\in A} n_{i,l}^{k,l} = n^{k,l}$$

3. Conservation law

$$\sum_{i|(i,j)\in A} n_{i,j}^{k,l} = \sum_{i|(j,i)\in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$

Equations (1/2)

1. Initialization

$$\sum_{j|(k,j)\in A} n_{k,j}^{k,l} = n^{k,l}$$

2. Reception

$$\sum_{i|(i,l)\in A} n_{i,l}^{k,l} = n^{k,l}$$

3. Conservation law

$$\sum_{i|(i,j)\in A} n_{i,j}^{k,l} = \sum_{i|(j,i)\in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$

Equations (1/2)

1. Initialization

$$\sum_{j|(k,j)\in A} n_{k,j}^{k,l} = n^{k,l}$$

2. Reception

$$\sum_{i|(i,l)\in A} n_{i,l}^{k,l} = n^{k,l}$$

3. Conservation law

$$\sum_{i|(i,j)\in A} n_{i,j}^{k,l} = \sum_{i|(j,i)\in A} n_{j,i}^{k,l} \quad \forall (k,l), j \neq k, j \neq l$$

Equations (2/2)

4. Congestion

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$$

5. Objective function

$$C_{\max} \geqslant C_{i,j}, \qquad \forall i, j$$

Minimize $C_{\max}$

Linear program in rational numbers: polynomial-time solution. In practice use Maple or Mupad

Equations (2/2)

4. Congestion

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$$

5. Objective function

$$C_{\max} \geqslant C_{i,j}, \qquad \forall i, j$$

Minimize $C_{\max}$

Linear program in rational numbers: polynomial-time solution. In practice use Maple or Mupad

Equations (2/2)

4. Congestion

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$$

5. Objective function

$$C_{\max} \geqslant C_{i,j}, \qquad \forall i, j$$

Minimize $C_{\max}$

Linear program in rational numbers: polynomial-time solution. In practice use Maple or Mupad

Equations (2/2)

4. Congestion

$$C_{i,j} = \sum_{(k,l)|n^{k,l}>0} n_{i,j}^{k,l}$$

5. Objective function

$$C_{\max} \geqslant C_{i,j}, \qquad \forall i, j$$

$$\text{Minimize } C_{\max}$$

Linear program in rational numbers: polynomial-time solution. In practice use Maple or Mupad

Routing algorithm

1. Computing optimal solution $C_{\max}$ of previous linear program
2. Let Ω to be defined later
   During time-interval $[p\Omega, (p+1)\Omega]$, edge (i,j) forwards:

$$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor$$

   packets that go from k to l.
3. After time-step

$$T \equiv \left\lceil \frac{C_{\max}}{\Omega} \right\rceil \Omega \leqslant C_{\max} + \Omega$$

   sequentially process M residual packets in no longer than ML time-steps, where L is the maximum length of a simple path in the network

Routing algorithm

1. Computing optimal solution $C_{\max}$ of previous linear program
2. Let Ω to be defined later
   During time-interval $[p\Omega, (p+1)\Omega]$, edge (i,j) forwards:

$$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor$$

   packets that go from k to l.
3. After time-step

$$T \equiv \left\lceil \frac{C_{\max}}{\Omega} \right\rceil \Omega \leqslant C_{\max} + \Omega$$

   sequentially process M residual packets in no longer than ML time-steps, where L is the maximum length of a simple path in the network

Routing algorithm

1. Computing optimal solution $C_{\max}$ of previous linear program
2. Let Ω to be defined later
   During time-interval $[p\Omega, (p+1)\Omega]$, edge (i,j) forwards:

$$m_{i,j}^{k,l} = \left\lfloor \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} \right\rfloor$$

   packets that go from k to l.
3. After time-step

$$T \equiv \left\lceil \frac{C_{\max}}{\Omega} \right\rceil \Omega \leqslant C_{\max} + \Omega$$

   sequentially process M residual packets in no longer than ML time-steps, where L is the maximum length of a simple path in the network

Feasibility

$$\sum_{(k,l)} m_{i,j}^{k,l} \leqslant \sum_{(k,l)} \frac{n_{i,j}^{k,l}\Omega}{C_{\max}} = \frac{C_{i,j}\Omega}{C_{\max}} \leqslant \Omega$$

Makespan

- Define Ω as $\Omega = \sqrt{C_{\max} n_c}$.
- Total number of packets still inside network at time-step T is at most
$$2|A|\sqrt{C_{\max} n_c} + |A| n_c$$
- Makespan:
$$C_{\max} \leqslant C^* \leqslant C_{\max} + \sqrt{C_{\max} n_c} + 2|A|\sqrt{C_{\max} n_c}|V| + |A| n_c |V|$$

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed

Search

- which (rational) fraction of time is spent computing for which application?
- which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

$\Rightarrow$ react on the fly to resource availability variations

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- which (rational) fraction of time is spent computing for which application?
- which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

$\Rightarrow$ react on the fly to resource availability variations

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

$\Rightarrow$ react on the fly to resource availability variations

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

$\Rightarrow$ react on the fly to resource availability variations

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

$\Rightarrow$ react on the fly to resource availability variations

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

$\Rightarrow$ react on the fly to resource availability variations

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

$\Rightarrow$ react on the fly to resource availability variations

Steady-state scheduling

Background Approach pioneered by Bertsimas and Gamarnik

Rationale Maximize throughput (total load executed per period)

Simplicity Relaxation of makespan minimization problem

- Ignore initialization and clean-up phases
- Precise ordering/allocation of tasks/messages not needed
- Characterize resource activity during each time-unit:
  - which (rational) fraction of time is spent computing for which application?
  - which (rational) fraction of time is spent receiving or sending to which neighbor?

Efficiency Periodic schedule, described in compact form

Adaptability Dynamically record observed performance during current period, and inject this information to compute optimal schedule for next period

$\Rightarrow$ react on the fly to resource availability variations

Outline

Master-slave platform

Master-slave tasking Simple yet efficient

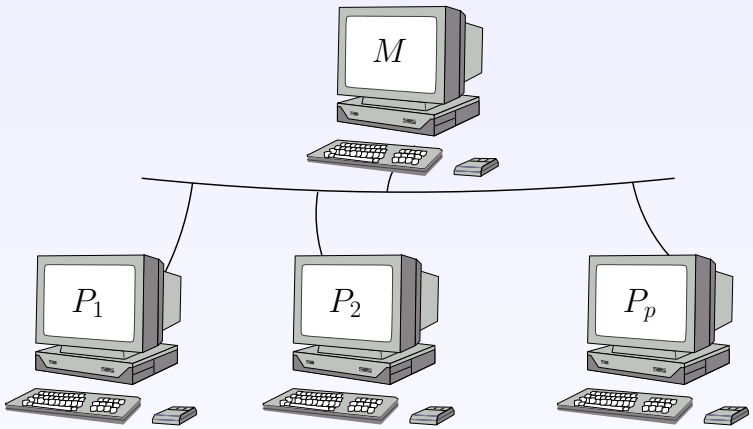
Standard implementation Independent tasks are executed by identical processors (the slaves) under the supervision of a special processor (the master)

Heterogeneous version Computing times and communication times are different from slave to slave

Master-slave platform

Master-slave tasking Simple yet efficient

Standard implementation Independent tasks are executed by identical processors (the slaves) under the supervision of a special processor (the master)

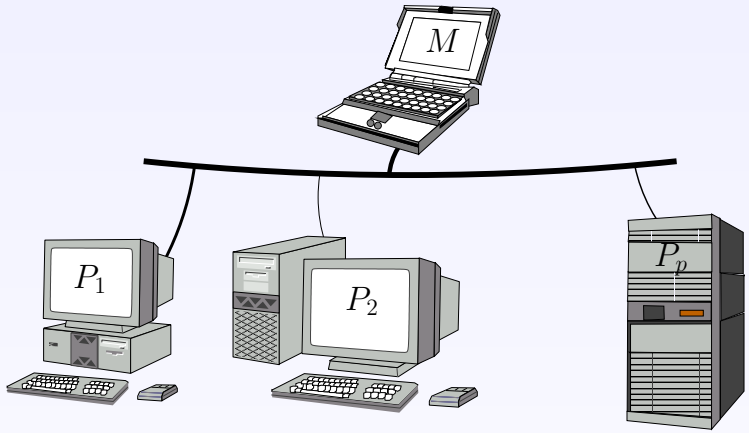


Heterogeneous version Computing times and communication times are different from slave to slave

Master-slave platform

Master-slave tasking Simple yet efficient

Standard implementation Independent tasks are executed by identical processors (the slaves) under the supervision of a special processor (the master)



Heterogeneous version Computing times and communication times are different from slave to slave

Model

- Set of independent tasks to be executed by p slaves
- All tasks are *identical*: each represents the same amount of computations
- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the *one-port* model: M can only send one task at a given time-step
- *Overlap* computations and communications

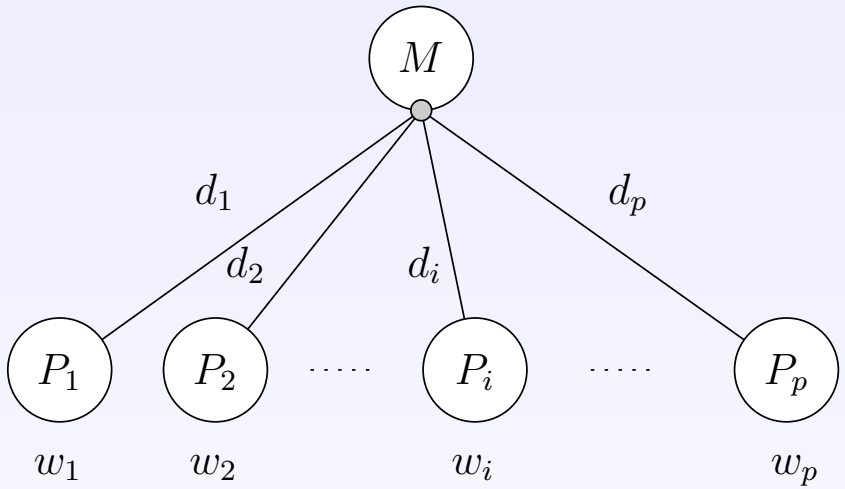
Model

- Set of independent tasks to be executed by p slaves
- All tasks are *identical*: each represents the same amount of computations

- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the *one-port* model: M can only send one task at a given time-step
- *Overlap* computations and communications

Model

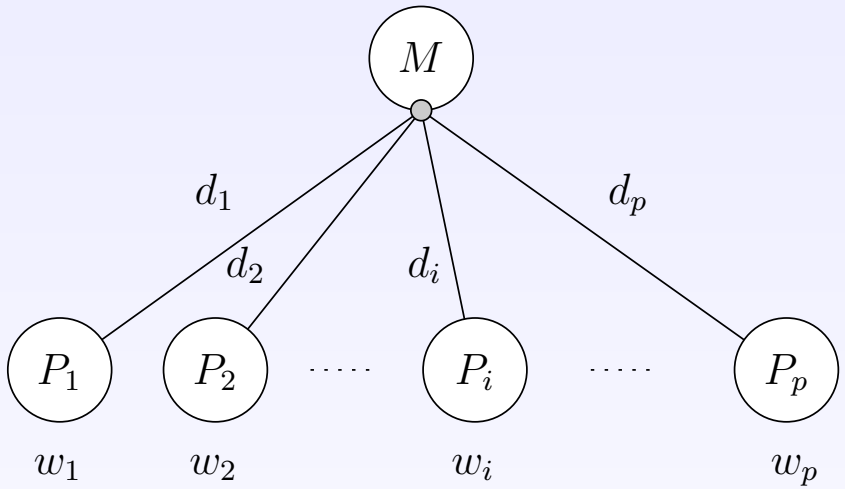
- Set of independent tasks to be executed by p slaves
- All tasks are *identical*: each represents the same amount of computations



- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the *one-port* model: M can only send one task at a given time-step
- *Overlap* computations and communications

Model

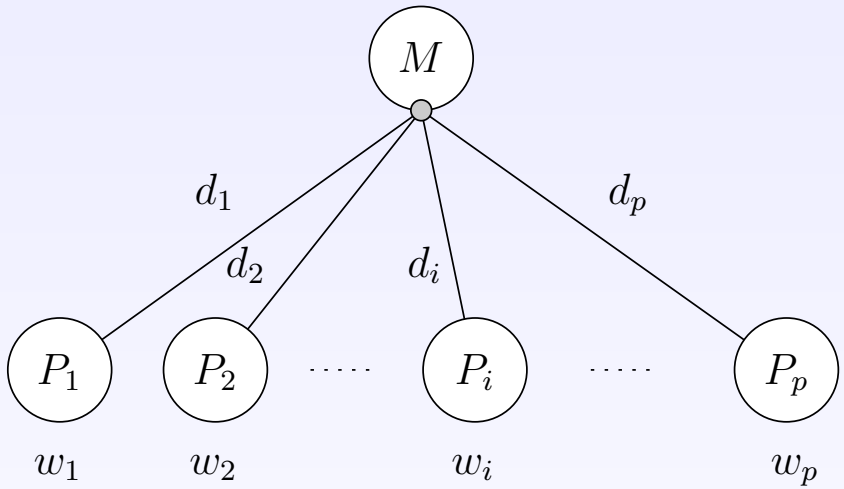
- Set of independent tasks to be executed by p slaves
- All tasks are *identical*: each represents the same amount of computations



- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the *one-port* model: M can only send one task at a given time-step
- *Overlap* computations and communications

Model

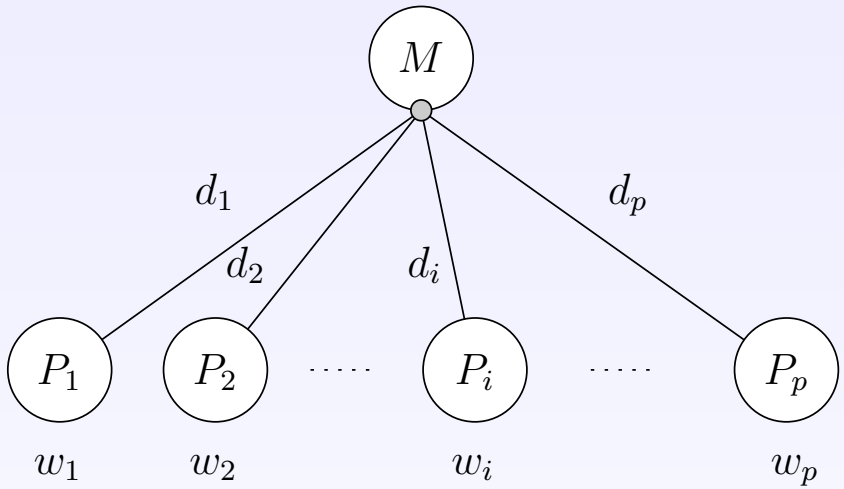
- Set of independent tasks to be executed by p slaves
- All tasks are *identical*: each represents the same amount of computations



- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the *one-port* model: M can only send one task at a given time-step
- *Overlap* computations and communications

Model

- Set of independent tasks to be executed by p slaves
- All tasks are *identical*: each represents the same amount of computations



- Need d_i time-units to transfer a task from M to P_i, and w_i time-units to execute it on P_i
- Communications obey the *one-port* model: M can only send one task at a given time-step
- *Overlap* computations and communications

Complexity results

Definition MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$):
Given a master-slave platform with parameters $(d_1, w_1), \ldots, (d_p, w_p)$, what it the minimum time to process n tasks?

Complexity results

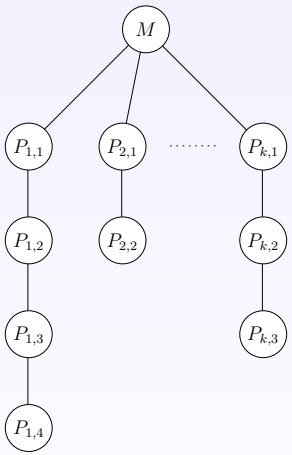
Definition MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$): Given a master-slave platform with parameters $(d_1, w_1), \ldots, (d_p, w_p)$, what it the minimum time to process n tasks?

MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$) can be solved at cost $O(n^2p^2)$ by a complicated greedy algorithm

Complexity results

Definition MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$): Given a master-slave platform with parameters $(d_1, w_1), \ldots, (d_p, w_p)$, what it the minimum time to process n tasks?

MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$) can be solved at cost $O(n^2p^2)$ by a complicated greedy algorithm

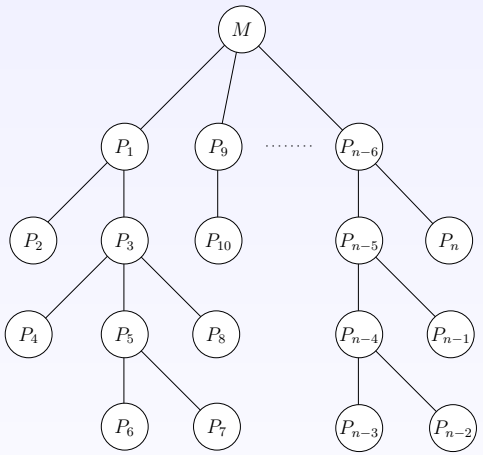


If the interconnection de plate-forme is a linear chain or a harpoon, problem still polynomial

Complexity results

Definition MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$): Given a master-slave platform with parameters $(d_1, w_1), \ldots, (d_p, w_p)$, what it the minimum time to process n tasks?

MasterSlave($P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T^{(1)}, \ldots, T^{(n)}$) can be solved at cost $O(n^2p^2)$ by a complicated greedy algorithm



If the interconnection de plate-forme is a linear chain or a harpoon, problem still polynomial

However, for tree-shaped platforms, problem becomes NP-complete

A model not well-suited...

- Hardness comes from the metric: *makespan* minimization
- Not suited to large-scale distributed platforms
  - Modeling a collection of clusters, and acquiring all various parameters: long, tedious and error-prone
  - Given difficulty and time needed to deploy applications on such platforms, number of tasks expected to be very large
- Concentrate on *steady-state*, and target complex platforms (with cycles and multiple paths) while designing efficient (asymptotically optimal) schedulings

A model not well-suited...

- Hardness comes from the metric: *makespan* minimization
- Not suited to large-scale distributed platforms
  - Modeling a collection of clusters, and acquiring all various parameters: long, tedious and error-prone
  - Given difficulty and time needed to deploy applications on such platforms, number of tasks expected to be very large
- Concentrate on *steady-state*, and target complex platforms (with cycles and multiple paths) while designing efficient (asymptotically optimal) schedulings

A model not well-suited...

- Hardness comes from the metric: *makespan* minimization
- Not suited to large-scale distributed platforms
  - Modeling a collection of clusters, and acquiring all various parameters: long, tedious and error-prone
  - Given difficulty and time needed to deploy applications on such platforms, number of tasks expected to be very large
- Concentrate on *steady-state*, and target complex platforms (with cycles and multiple paths) while designing efficient (asymptotically optimal) schedulings

A model not well-suited...

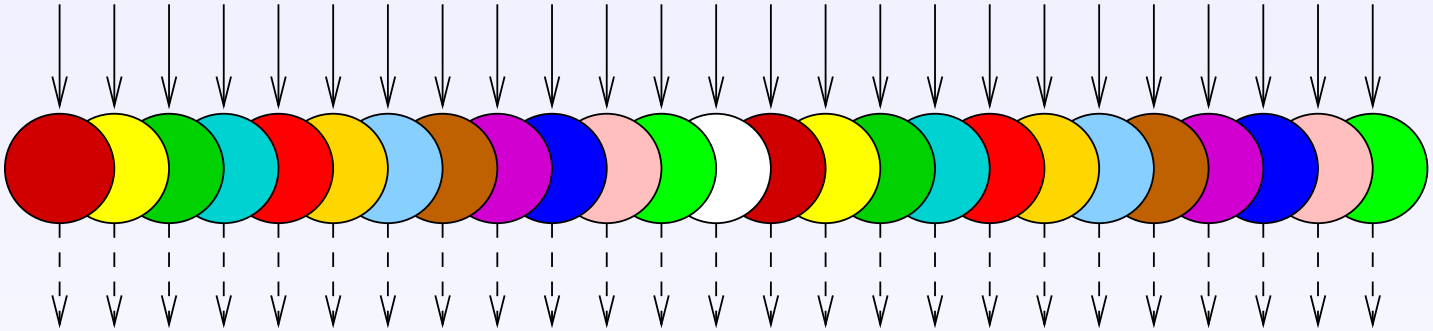
- Hardness comes from the metric: *makespan* minimization
- Not suited to large-scale distributed platforms
  - Modeling a collection of clusters, and acquiring all various parameters: long, tedious and error-prone
  - Given difficulty and time needed to deploy applications on such platforms, number of tasks expected to be very large
- Concentrate on *steady-state*, and target complex platforms (with cycles and multiple paths) while designing efficient (asymptotically optimal) schedulings

A model not well-suited...

- Hardness comes from the metric: *makespan* minimization
- Not suited to large-scale distributed platforms
  - Modeling a collection of clusters, and acquiring all various parameters: long, tedious and error-prone
  - Given difficulty and time needed to deploy applications on such platforms, number of tasks expected to be very large
- Concentrate on *steady-state*, and target complex platforms (with cycles and multiple paths) while designing efficient (asymptotically optimal) schedulings

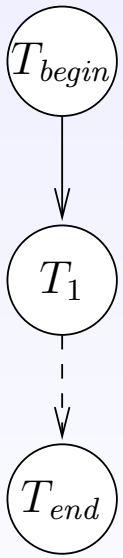
Application graph

n problem instances $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(n)}$, where n is large



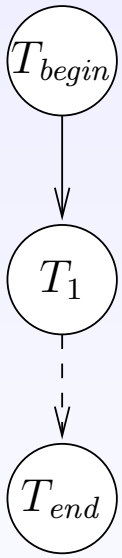
Application graph

n problem instances $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(n)}$, where n is large
Each problem corresponds to a copy of the same task graph
$G_A = (V_A, E_A)$, the *application graph*



Application graph

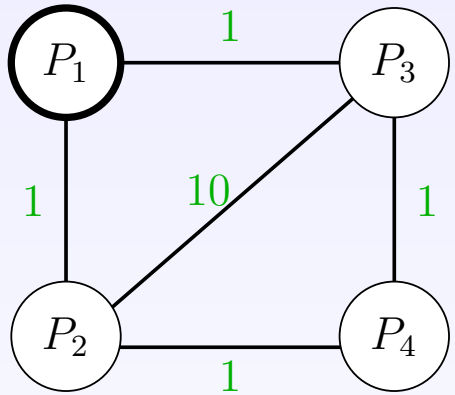
n problem instances $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(n)}$, where n is large
Each problem corresponds to a copy of the same task graph
$G_A = (V_A, E_A)$, the *application graph*



T_{begin} et T_{end} are fictitious tasks, used to model the scattering of input files and the gathering of output files

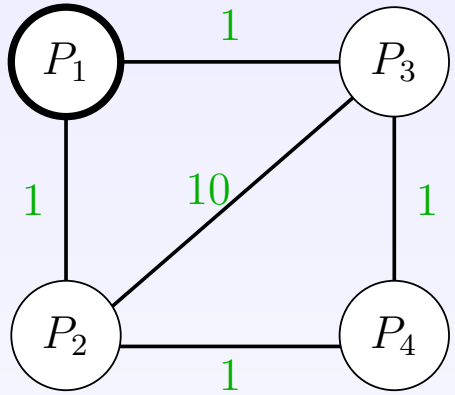
Platform graph

Target platform represented by *platform graph* $G_P = (V_P, E_P)$



Platform graph

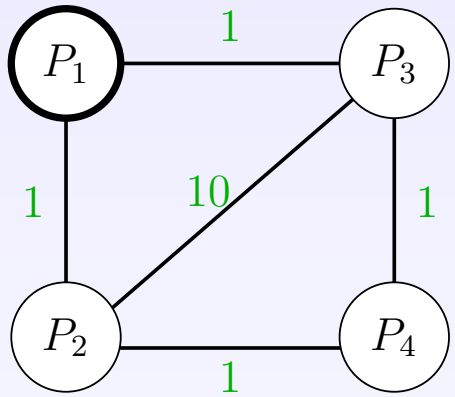
Target platform represented by *platform graph* $G_P = (V_P, E_P)$



Edge $P_i \rightarrow P_j$ is labeled with $c_{i,j}$: time needed to send a unit-length message from P_i to P_j

Platform graph

Target platform represented by *platform graph* $G_P = (V_P, E_P)$



Edge $P_i \rightarrow P_j$ is labeled with $c_{i,j}$: time needed to send a unit-length message from P_i to P_j

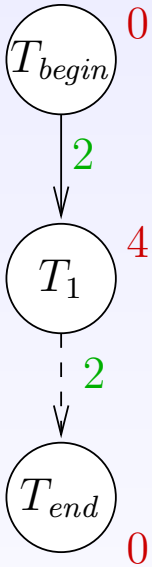
Communication model: full overlap, one-port for incoming *and* outgoing messages

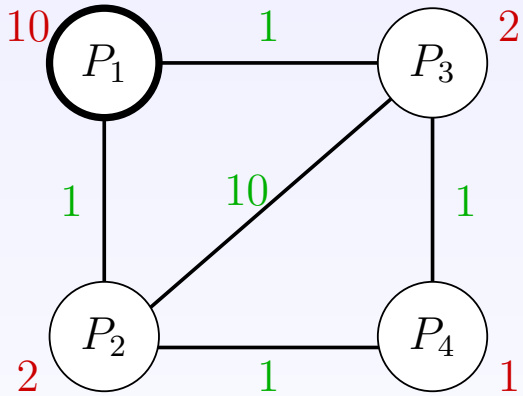
Computations and communications

P_i requires $w_{i,k}$ time-units to process task T_k ($k \in \{begin, 1, end\}$).

Computations and communications

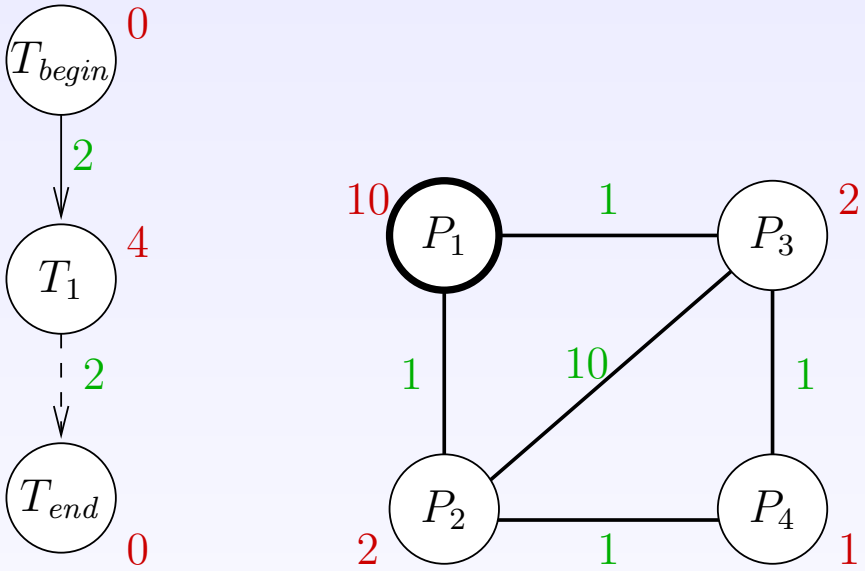
P_i requires $w_{i,k}$ time-units to process task T_k ($k \in \{begin, 1, end\}$).





Computations and communications

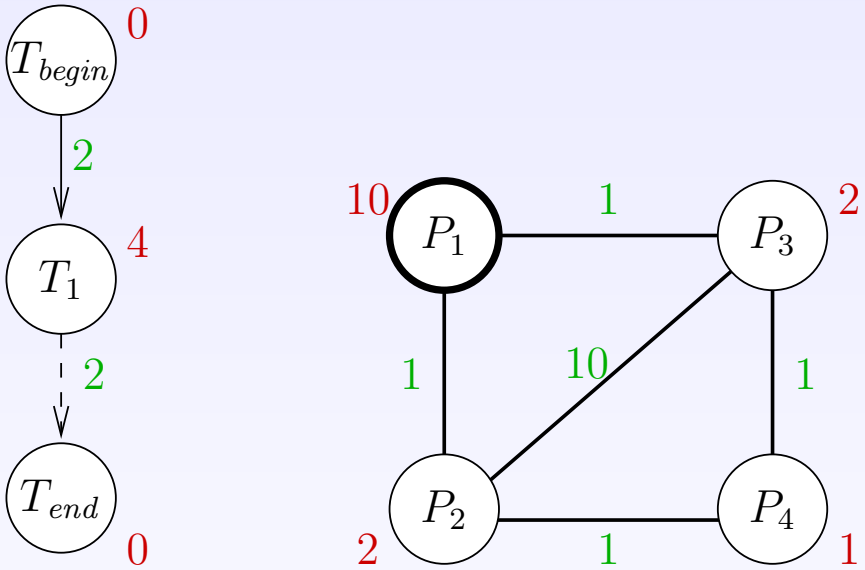
P_i requires $w_{i,k}$ time-units to process task T_k ($k \in \{begin, 1, end\}$).



Edge $e_{k,l} : T_k \rightarrow T_l$ in G_A is labeled with $data_{k,l}$: data volume generated by T_k and used by T_l

Computations and communications

P_i requires $w_{i,k}$ time-units to process task T_k ($k \in \{begin, 1, end\}$).



Edge $e_{k,l} : T_k \longrightarrow T_l$ in G_A is labeled with $data_{k,l}$: data volume generated by T_k and used by T_l

Transfer time of a file $e_{k,l}$ from P_i to P_j: $data_{k,l} \times c_{i,j}$

Definitions

Allocation An allocation is a pair of mappings: $\pi : V_A \mapsto V_P$ and $\sigma : E_A \mapsto \{\text{paths in } G_P\}$

Schedule A schedule associated to an allocation (π, σ) is a pair of mappings: $t_\pi : V_A \mapsto \mathbb{R}$ and application $t_\sigma : E_A \times E_P \mapsto \mathbb{R}$, satisfying to:

- precedence constraints
- resource constraints on processors
- resource constraints on network links
- one-port constraints

Definitions

Allocation An allocation is a pair of mappings: $\pi : V_A \mapsto V_P$ and $\sigma : E_A \mapsto \{\text{paths in } G_P\}$

Schedule A schedule associated to an allocation (π, σ) is a pair of mappings: $t_\pi : V_A \mapsto \mathbb{R}$ and application $t_\sigma : E_A \times E_P \mapsto \mathbb{R}$, satisfying to:

- precedence constraints
- resource constraints on processors
- resource constraints on network links
- one-port constraints

Definitions

Allocation An allocation is a pair of mappings: $\pi : V_A \mapsto V_P$ and $\sigma : E_A \mapsto \{\text{paths in } G_P\}$

Schedule A schedule associated to an allocation (π, σ) is a pair of mappings: $t_\pi : V_A \mapsto \mathbb{R}$ and application $t_\sigma : E_A \times E_P \mapsto \mathbb{R}$, satisfying to:

- precedence constraints
- resource constraints on processors
- resource constraints on network links
- one-port constraints

Definitions

Allocation An allocation is a pair of mappings: $\pi : V_A \mapsto V_P$ and $\sigma : E_A \mapsto \{\text{paths in } G_P\}$

Schedule A schedule associated to an allocation (π, σ) is a pair of mappings: $t_\pi : V_A \mapsto \mathbb{R}$ and application $t_\sigma : E_A \times E_P \mapsto \mathbb{R}$, satisfying to:

- precedence constraints
- resource constraints on processors
- resource constraints on network links
- one-port constraints

Definitions

Allocation An allocation is a pair of mappings: $\pi : V_A \mapsto V_P$ and $\sigma : E_A \mapsto \{\text{paths in } G_P\}$

Schedule A schedule associated to an allocation (π, σ) is a pair of mappings: $t_\pi : V_A \mapsto \mathbb{R}$ and application $t_\sigma : E_A \times E_P \mapsto \mathbb{R}$, satisfying to:

- precedence constraints
- resource constraints on processors
- resource constraints on network links
- one-port constraints

Definitions

Allocation An allocation is a pair of mappings: $\pi : V_A \mapsto V_P$ and $\sigma : E_A \mapsto \{\text{paths in } G_P\}$

Schedule A schedule associated to an allocation (π, σ) is a pair of mappings: $t_\pi : V_A \mapsto \mathbb{R}$ and application $t_\sigma : E_A \times E_P \mapsto \mathbb{R}$, satisfying to:

- precedence constraints
- resource constraints on processors
- resource constraints on network links
- one-port constraints

Activity variables

$cons(P_i, T_k)$: average number of tasks of type T_k processed by P_i every time-unit

$$\forall P_i, \forall T_k \in V_A,\ 0 \leqslant cons(P_i, T_k) \times w_{i,k} \leqslant 1$$

$sent(P_i \rightarrow P_j, e_{k,l})$: average number of files of type $e_{k,l}$ sent from P_i to P_j every time-unit

$$\forall P_i, P_j,\ 0 \leqslant sent(P_i \rightarrow P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) \leqslant 1$$

Activity variables

$cons(P_i, T_k)$: average number of tasks of type T_k processed by P_i every time-unit

$$\forall P_i, \forall T_k \in V_A,\ 0 \leqslant cons(P_i, T_k) \times w_{i,k} \leqslant 1$$

$sent(P_i \rightarrow P_j, e_{k,l})$: average number of files of type $e_{k,l}$ sent from P_i to P_j every time-unit

$$\forall P_i, P_j,\ 0 \leqslant sent(P_i \rightarrow P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) \leqslant 1$$

Steady-state equations

One-port for outgoing communications P_i sends messages to its neighbors sequentially

$$\forall P_i, \sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} \left(sent(P_i \to P_j, e_{k,l}) \times data_{k,l} \times c_{i,j} \right) \leqslant 1$$

One-port for ingoing communications P_i receives messages sequentially

$$\forall P_i, \sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} \left(sent(P_j \to P_i, e_{k,l}) \times data_{k,l} \times c_{j,i} \right) \leqslant 1$$

Overlap Computations and communications take place simultaneously

$$\forall P_i, \sum_{T_k \in V_A} cons(P_i, T_k) \times w_{i,k} \leqslant 1$$

Steady-state equations

One-port for outgoing communications P_i sends messages to its neighbors sequentially

$$\forall P_i, \sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} \left(sent(P_i \to P_j, e_{k,l}) \times data_{k,l} \times c_{i,j} \right) \leqslant 1$$

One-port for ingoing communications P_i receives messages sequentially

$$\forall P_i, \sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} \left(sent(P_j \to P_i, e_{k,l}) \times data_{k,l} \times c_{j,i} \right) \leqslant 1$$

Overlap Computations and communications take place simultaneously

$$\forall P_i, \sum_{T_k \in V_A} cons(P_i, T_k) \times w_{i,k} \leqslant 1$$

Steady-state equations

One-port for outgoing communications P_i sends messages to its neighbors sequentially

$$\forall P_i, \sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} \left(sent(P_i \to P_j, e_{k,l}) \times data_{k,l} \times c_{i,j} \right) \leqslant 1$$

One-port for ingoing communications P_i receives messages sequentially

$$\forall P_i, \sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} \left(sent(P_j \to P_i, e_{k,l}) \times data_{k,l} \times c_{j,i} \right) \leqslant 1$$

Overlap Computations and communications take place simultaneously

$$\forall P_i, \sum_{T_k \in V_A} cons(P_i, T_k) \times w_{i,k} \leqslant 1$$

Conservation law

Consider a processor P_i and an edge $e_{k,l}$ of the application graph:

Files of type $e_{k,l}$ received: $\sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l})$

Files of type $e_{k,l}$ generated: $cons(P_i, T_k)$

Files of type $e_{k,l}$ consumed: $cons(P_i, T_l)$

Files of type $e_{k,l}$ sent: $\sum_{P_i \to P_j} sent(P_i \to P_j, e_{k,l})$

In steady state:

$$\forall P_i, \forall e_{k,l} : T_k \to T_l \in E_A,$$

$$\sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l}) + cons(P_i, T_k) =$$

$$\sum_{P_i \to P_j} sent(P_i \to P_j, e_{k,l}) + cons(P_i, T_l)$$

Upper bound for the throughput

MAXIMIZE $\rho = \sum_{i=1}^{p} cons(P_i, T_{end})$,

UNDER THE CONSTRAINTS

$$
\begin{cases}
\text{(1a)} & \forall P_i, \forall T_k \in V_A,\ 0 \leqslant cons(P_i, T_k) \times w_{i,k} \leqslant 1 \\
\text{(1b)} & \forall P_i, P_j,\ 0 \leqslant sent(P_i \to P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) \leqslant 1 \\
\text{(1c)} & \forall P_i,\ \displaystyle\sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} \left(sent(P_i \to P_j, e_{k,l}) \times data_{k,l} \times c_{i,j}\right) \leqslant 1 \\
\text{(1d)} & \forall P_i,\ \displaystyle\sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} \left(sent(P_j \to P_i, e_{k,l}) \times data_{k,l} \times c_{j,i}\right) \leqslant 1 \\
\text{(1e)} & \forall P_i,\ \displaystyle\sum_{T_k \in V_A} cons(P_i, T_k) \times w_{i,k} \leqslant 1 \\
\text{(1f)} & \forall P_i, \forall e_{k,l} \in E_A : T_k \to T_l, \\
& \qquad \displaystyle\sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l}) + cons(P_i, T_k) = \\
& \qquad\qquad \displaystyle\sum_{P_i \to P_j} sent(P_i \to P_j, e_{k,l}) + cons(P_i, T_l)
\end{cases}
$$

How to design a schedule achieving this throughput?

Back to the example

Computations

	$cons(P_i, T_1)$
P_1	0.025
P_2	0.125
P_3	0.125
P_4	0.250
Total	21 tasks / 40 seconds

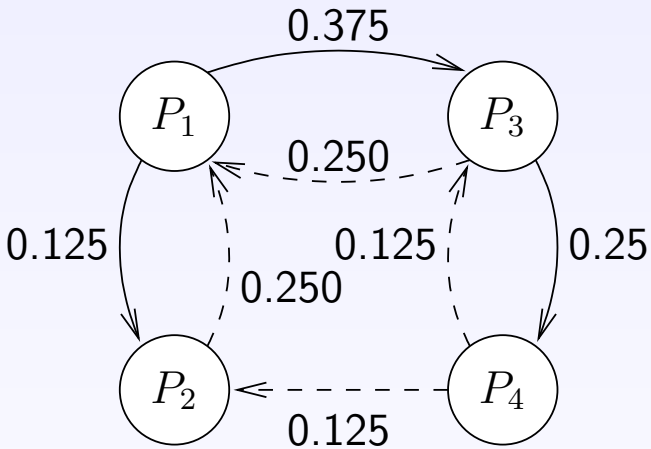
Communications

Back to the example

Computations

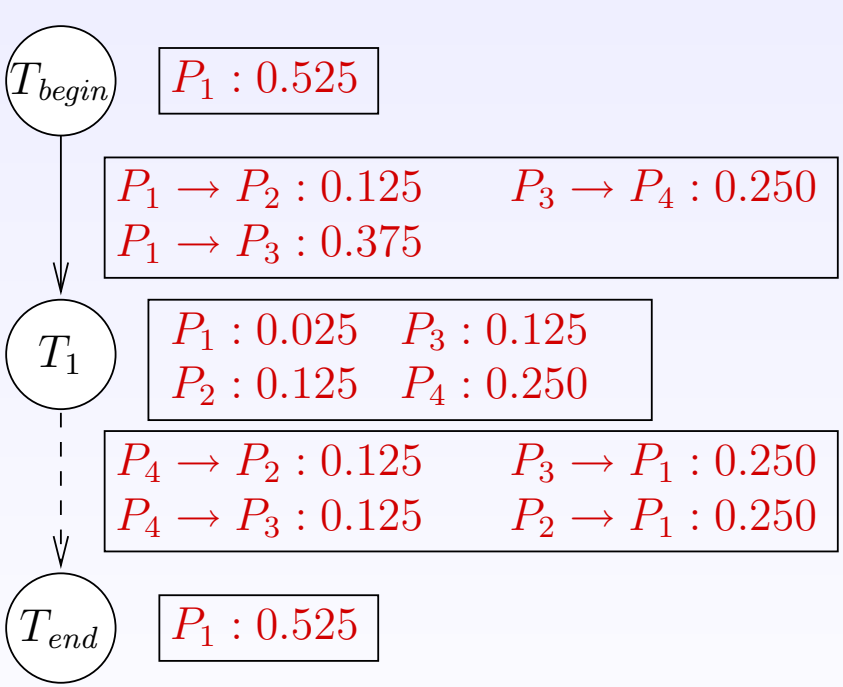
	$cons(P_i, T_1)$
P_1	0.025
P_2	0.125
P_3	0.125
P_4	0.250
Total	21 tasks / 40 seconds

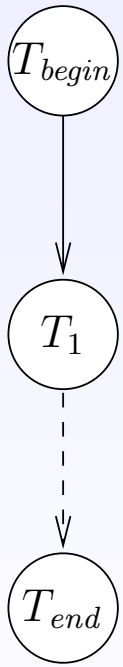
Communications



Decomposition into a set of allocations (1/2)

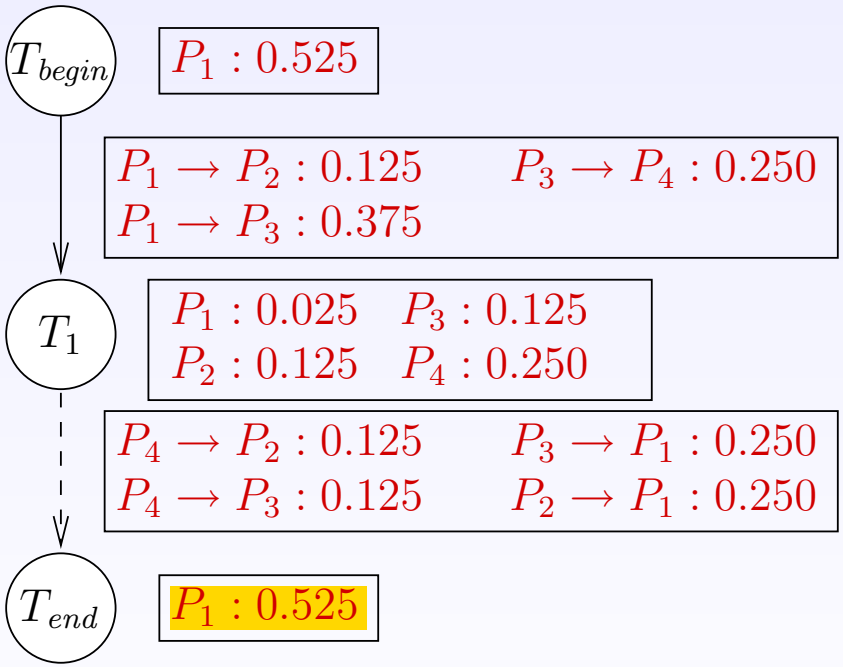
Steady state = superposition of several allocations

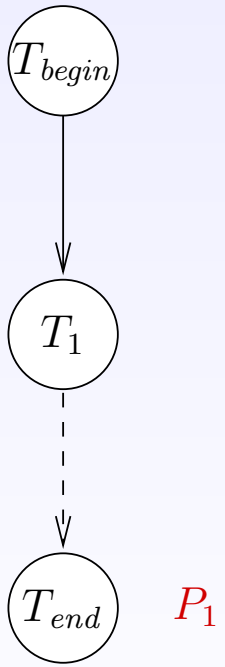




Decomposition into a set of allocations (1/2)

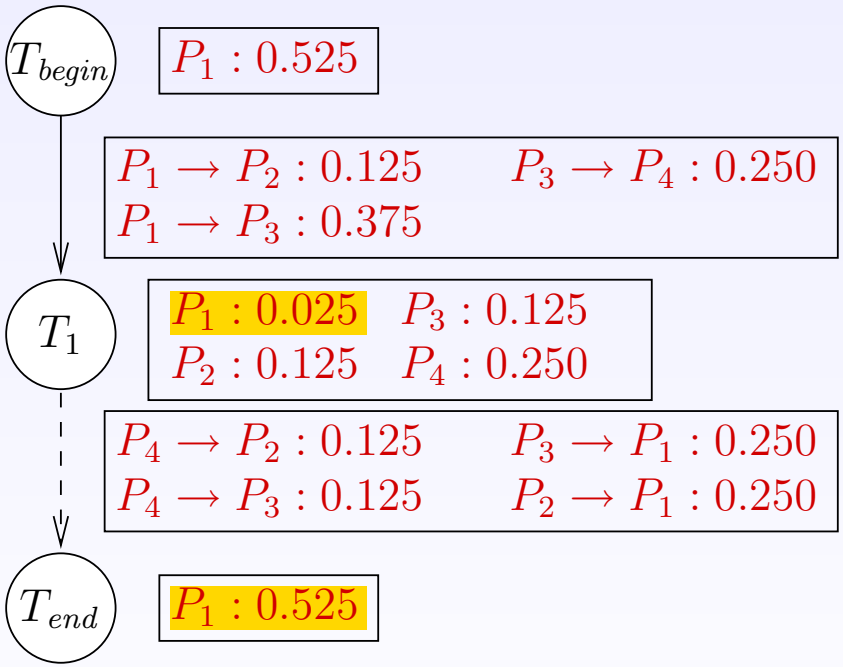
Steady state = superposition of several allocations

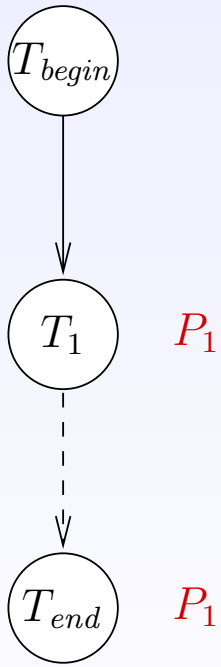




Decomposition into a set of allocations (1/2)

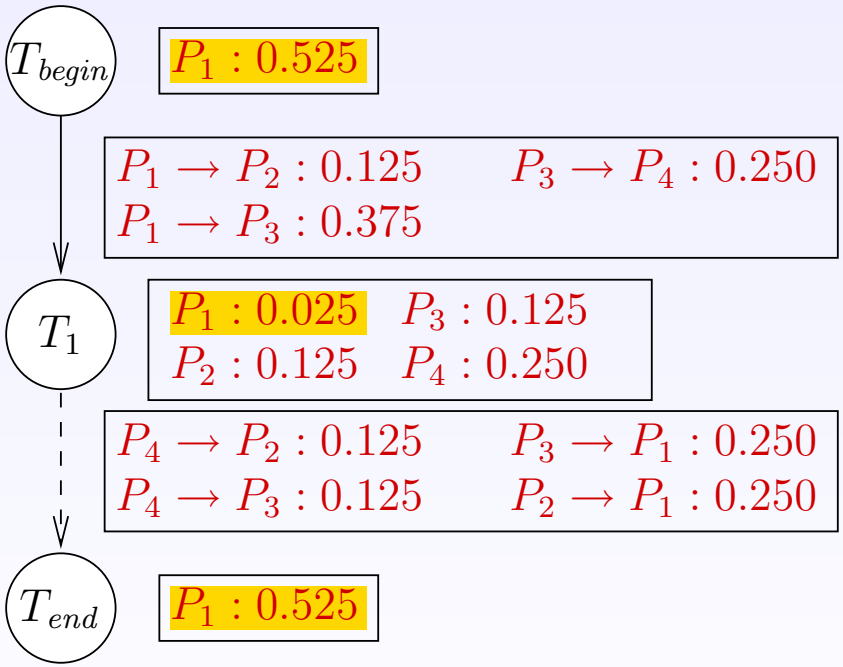
Steady state = superposition of several allocations

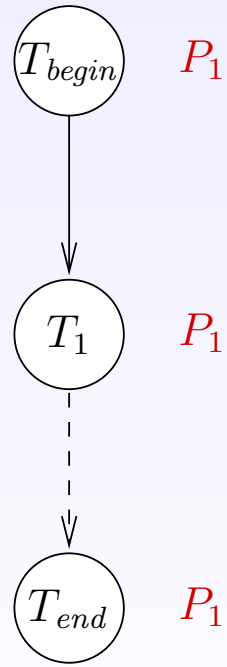




Decomposition into a set of allocations (1/2)

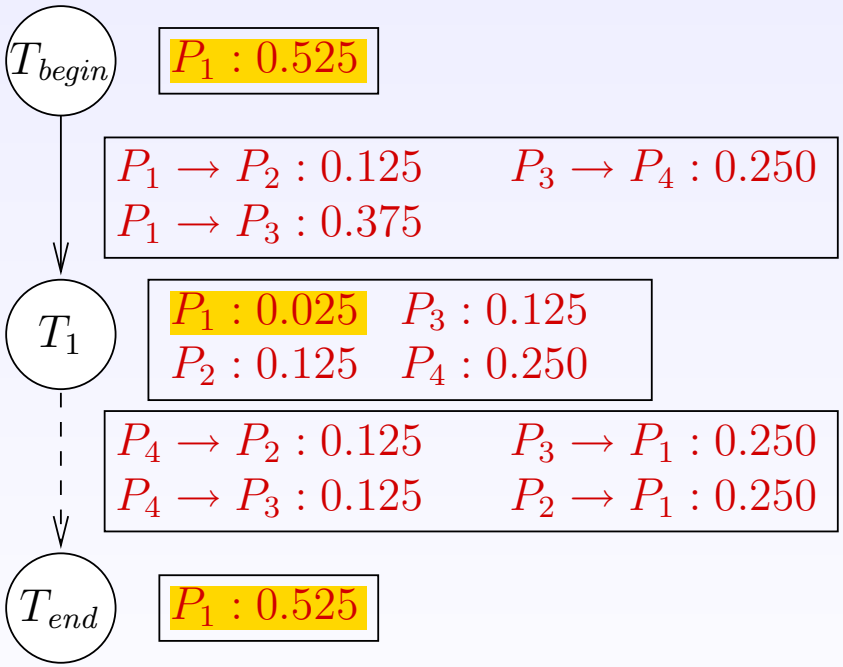
Steady state = superposition of several allocations

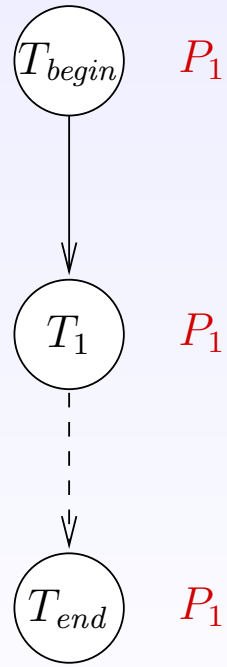




Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations

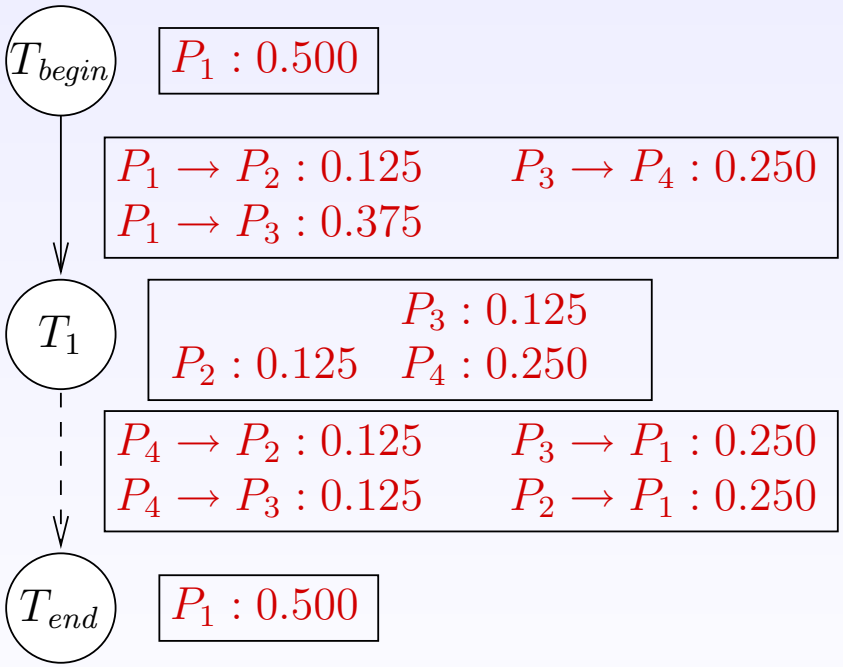


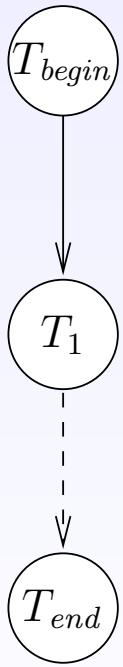


$\mathcal{A}_1 : 0.025$

Decomposition into a set of allocations (1/2)

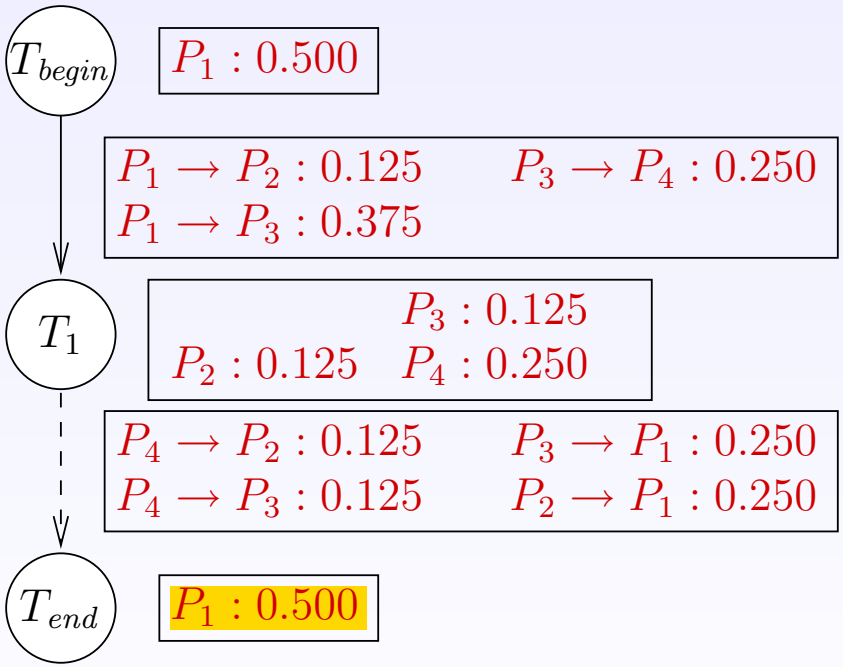
Steady state = superposition of several allocations

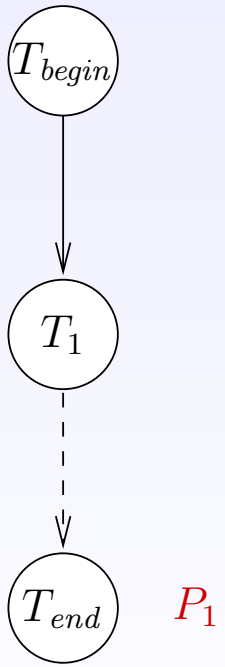




Decomposition into a set of allocations (1/2)

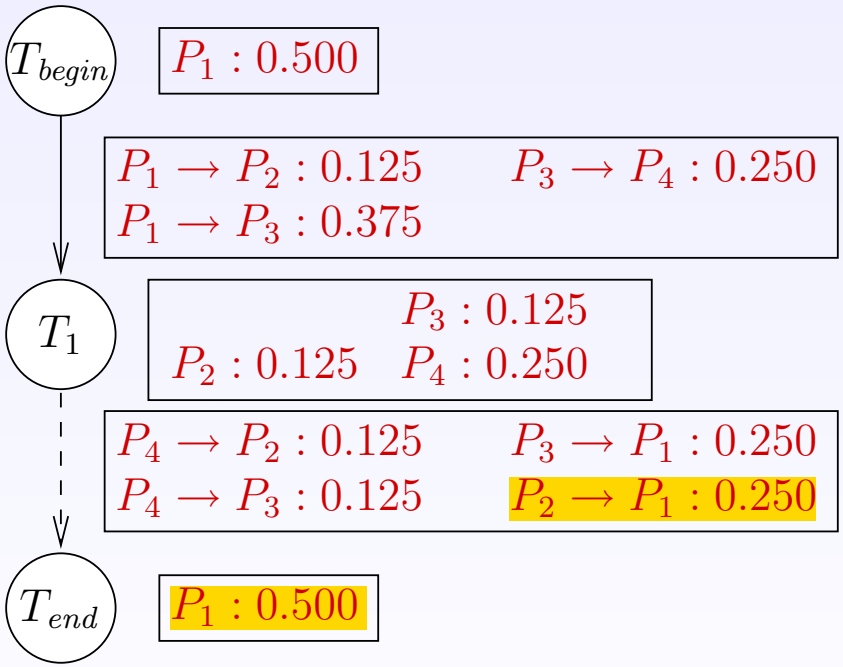
Steady state = superposition of several allocations

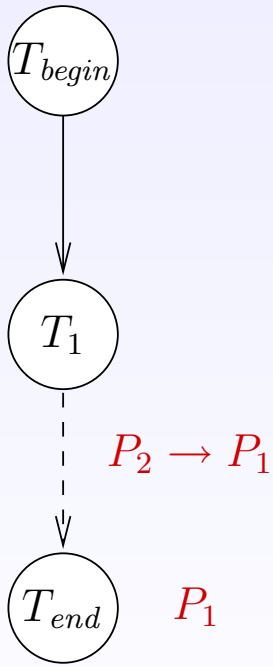




Decomposition into a set of allocations (1/2)

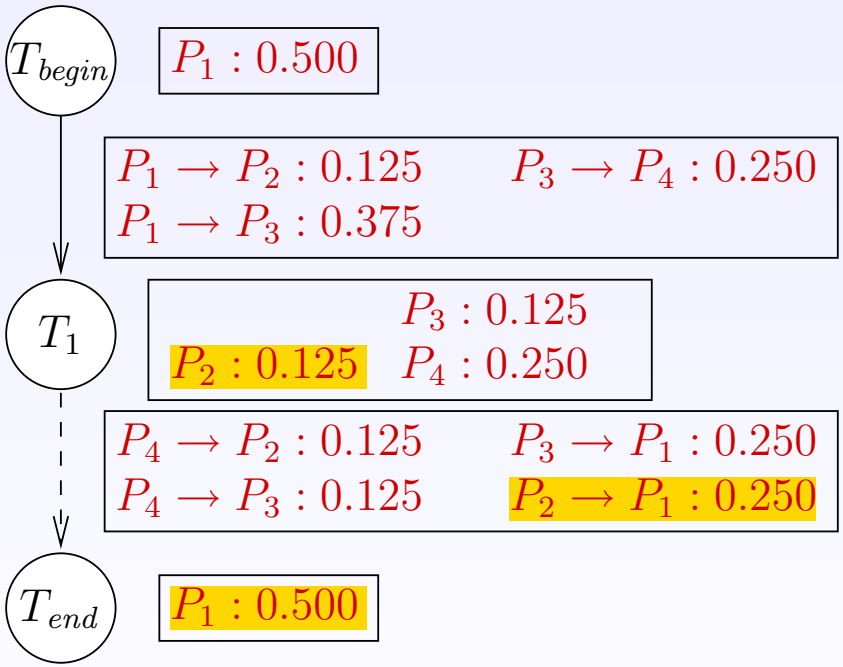
Steady state = superposition of several allocations

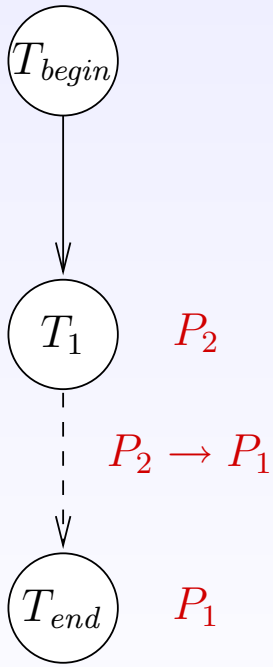




Decomposition into a set of allocations (1/2)

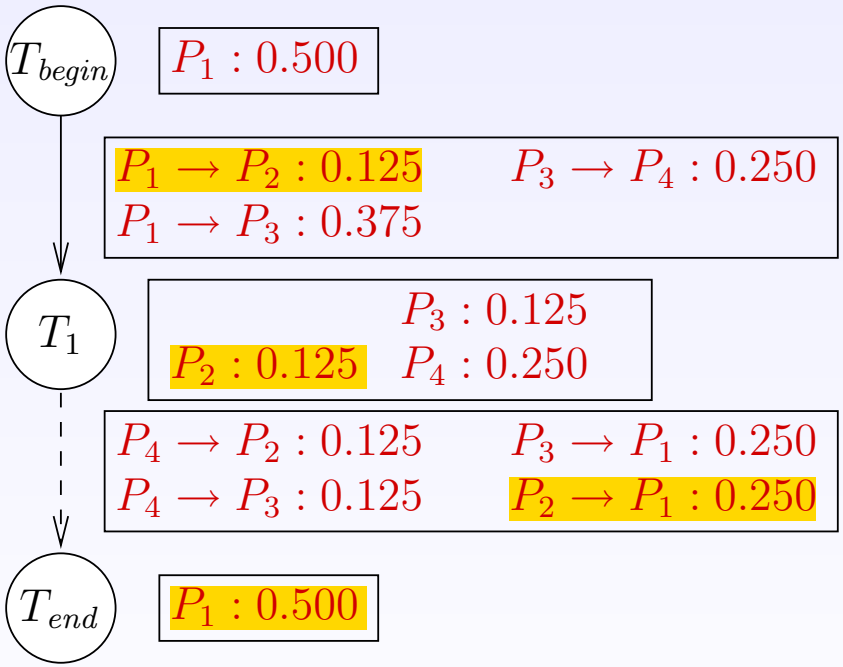
Steady state = superposition of several allocations

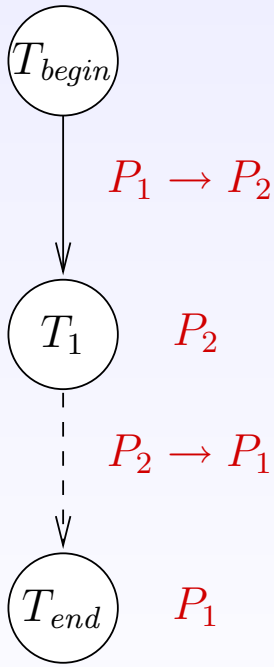




Decomposition into a set of allocations (1/2)

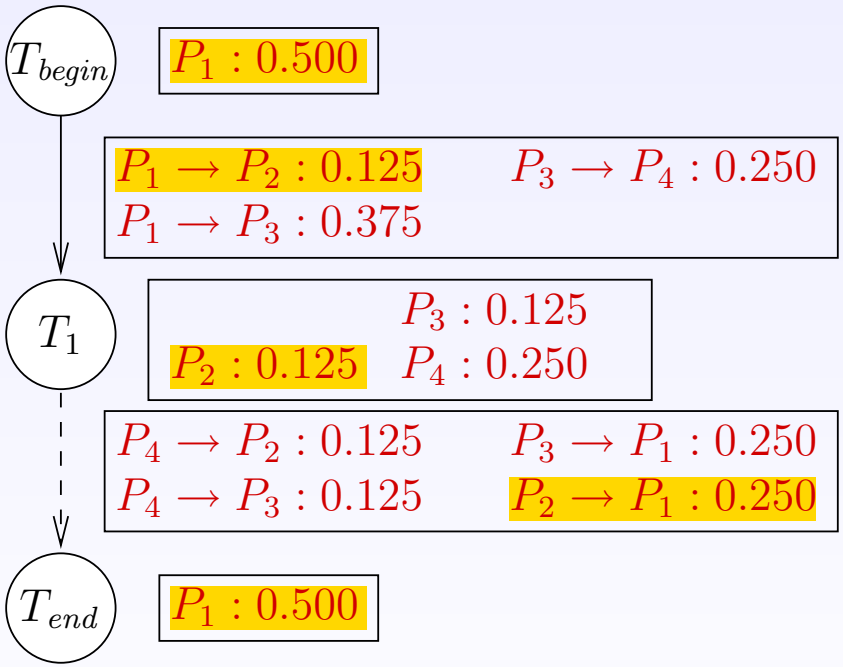
Steady state = superposition of several allocations

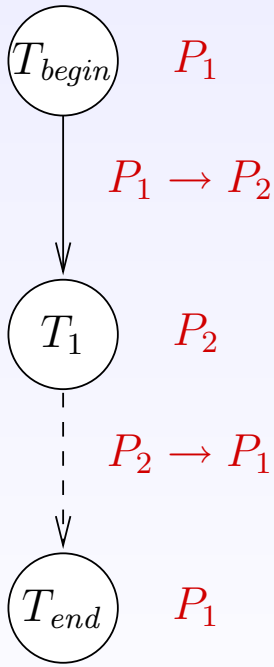




Decomposition into a set of allocations (1/2)

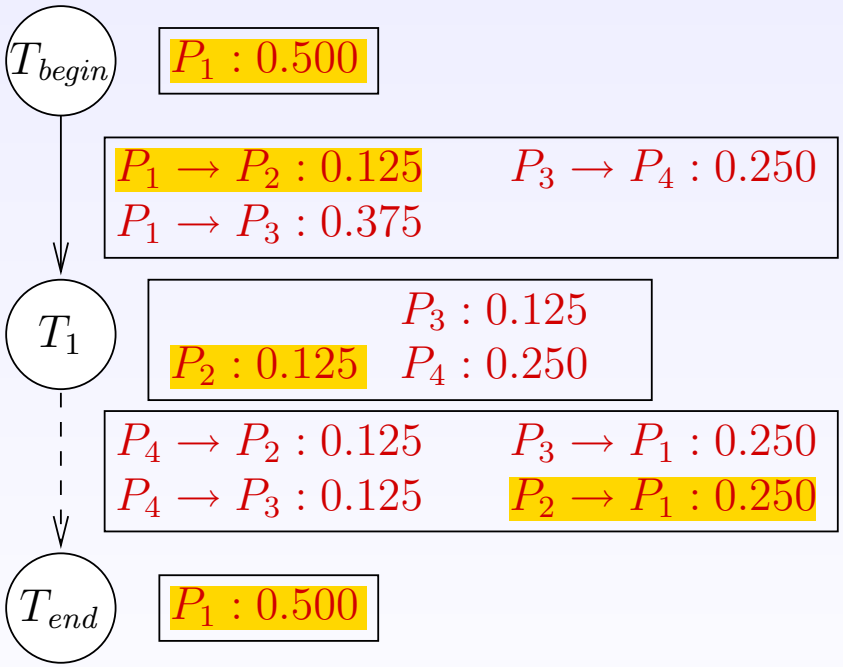
Steady state = superposition of several allocations

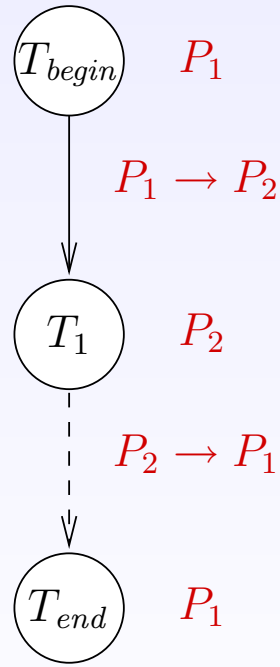




Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations

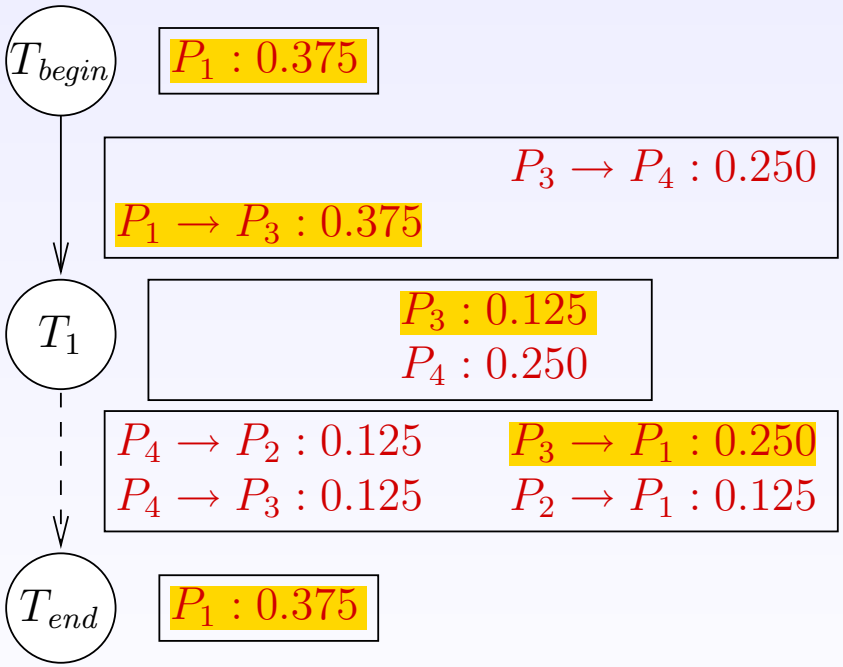


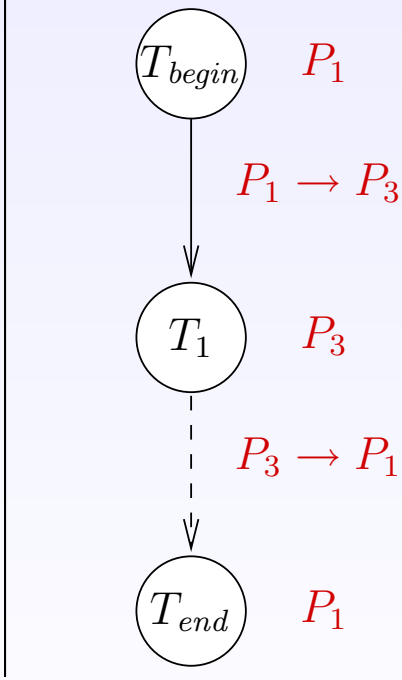


$\mathcal{A}_2 : 0.125$

Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations

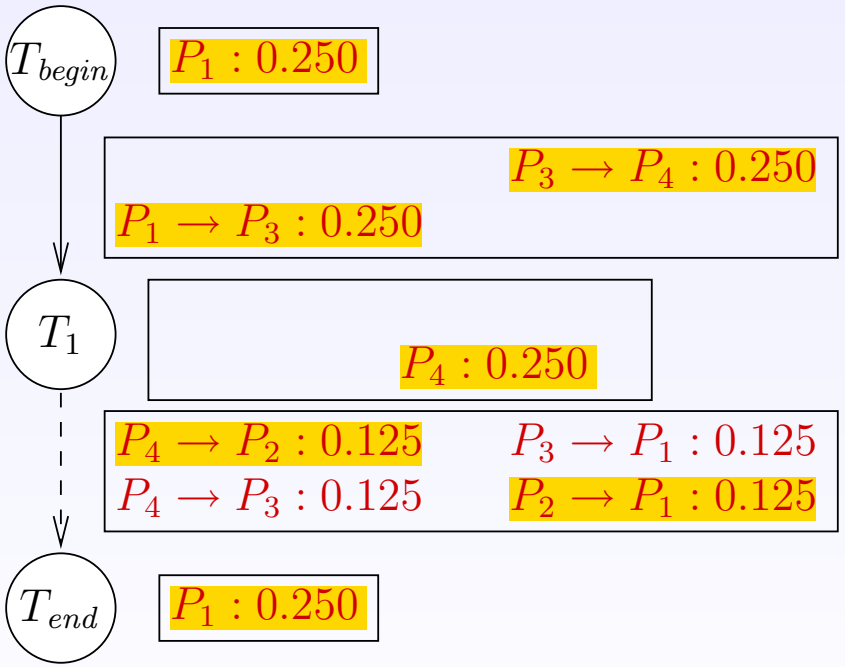


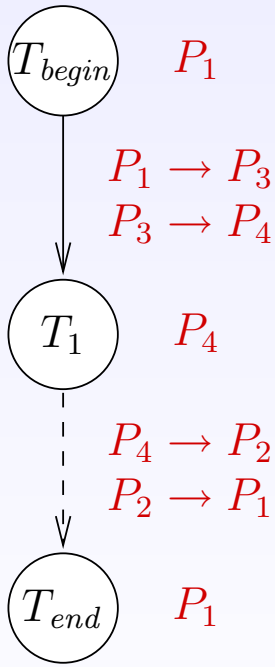


$\mathcal{A}_3 : 0.125$

Decomposition into a set of allocations (1/2)

Steady state = superposition of several allocations

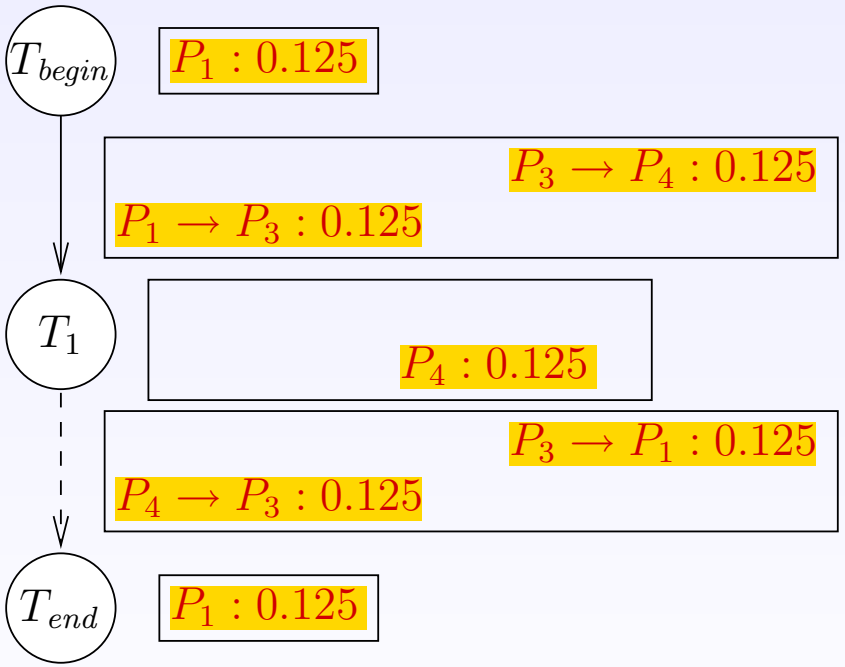


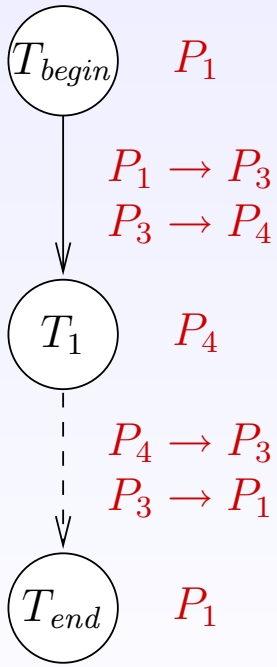


$\mathcal{A}_4 : 0.125$

Decomposition into a set of allocations (1/2)

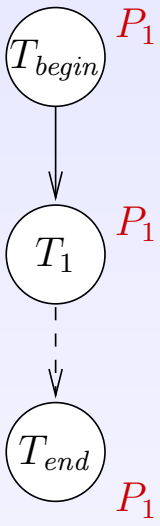
Steady state = superposition of several allocations





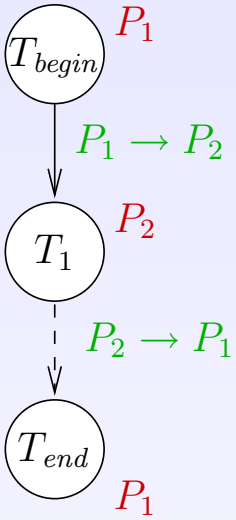
$\mathcal{A}_5 : 0.125$

Decomposition into a set of allocations (2/2)



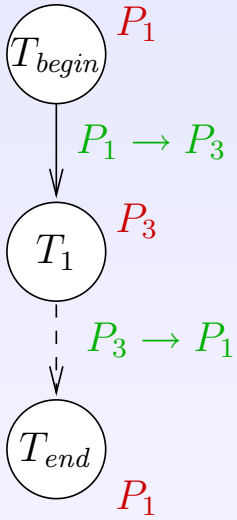
$\mathcal{A}_1$

0,025



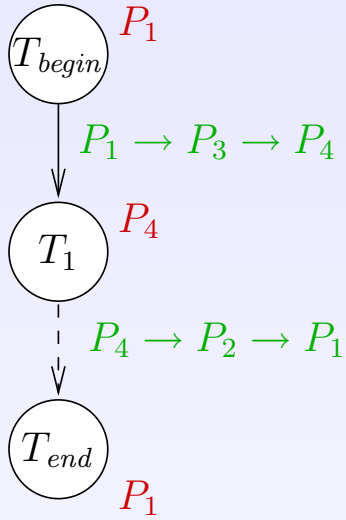
$\mathcal{A}_2$

0,125



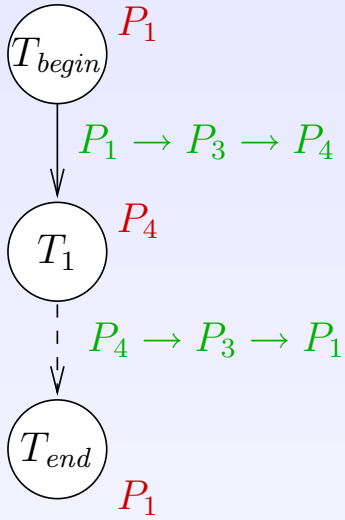
$\mathcal{A}_3$

0,125



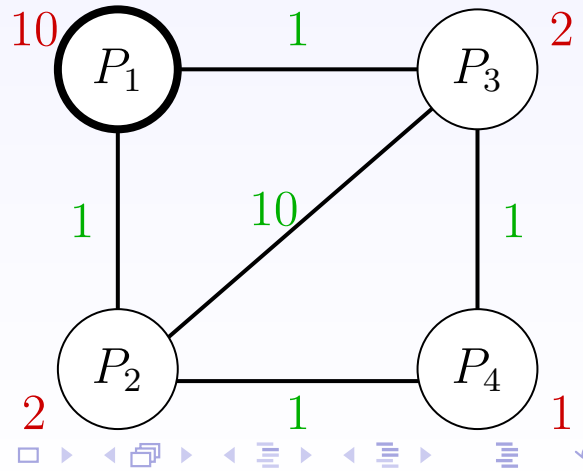
$\mathcal{A}_4$

0,125

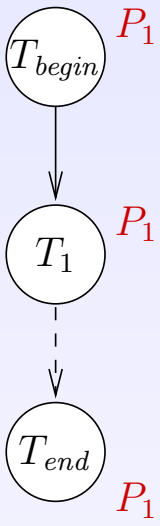


$\mathcal{A}_5$

0,125

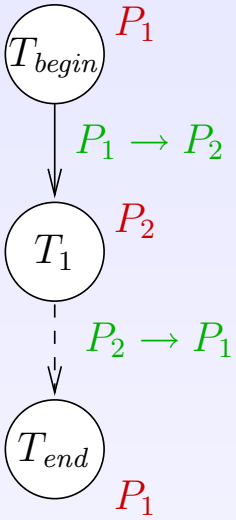


Decomposition into a set of allocations (2/2)



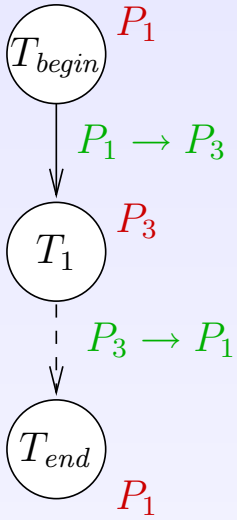
$\mathcal{A}_1$

0,025



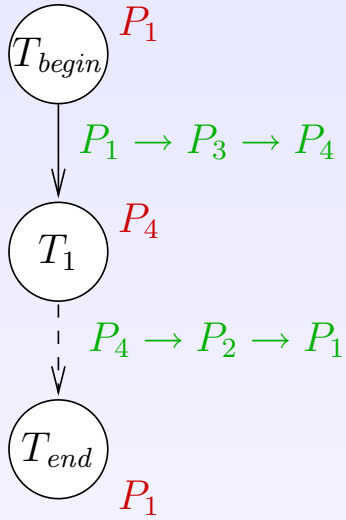
$\mathcal{A}_2$

0,125



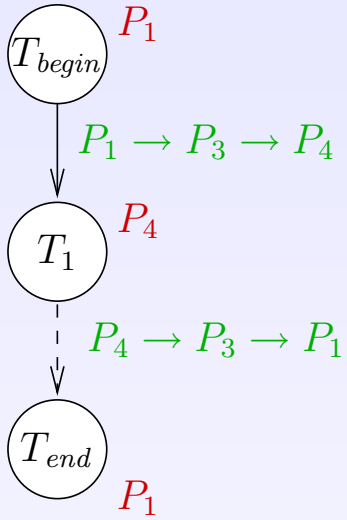
$\mathcal{A}_3$

0,125



$\mathcal{A}_4$

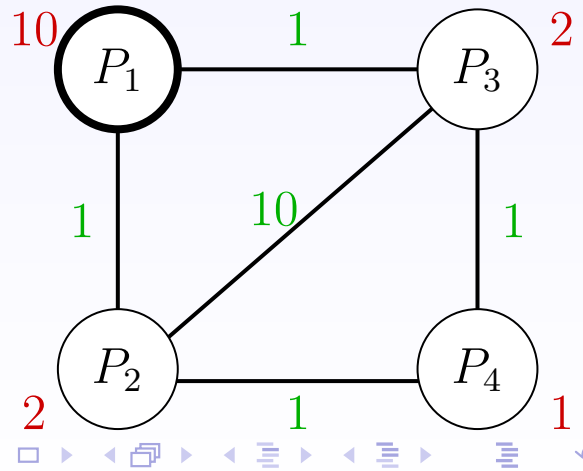
0,125



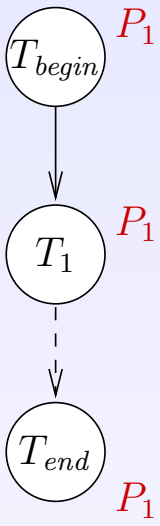
$\mathcal{A}_5$

0,125

This decomposition is always possible

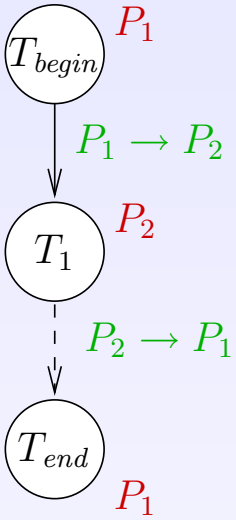


Decomposition into a set of allocations (2/2)



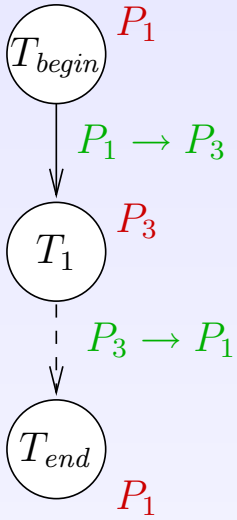
$\mathcal{A}_1$

0,025



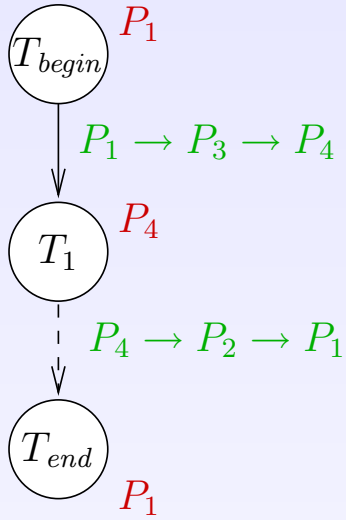
$\mathcal{A}_2$

0,125



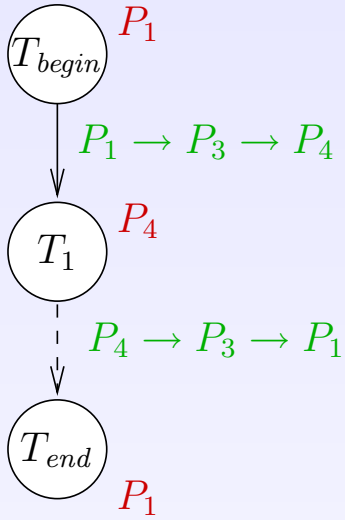
$\mathcal{A}_3$

0,125



$\mathcal{A}_4$

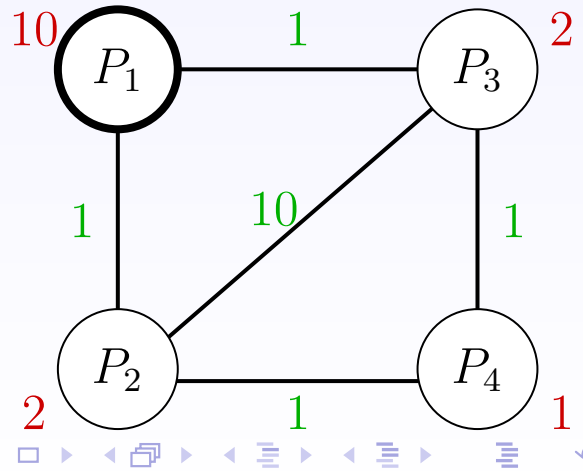
0,125



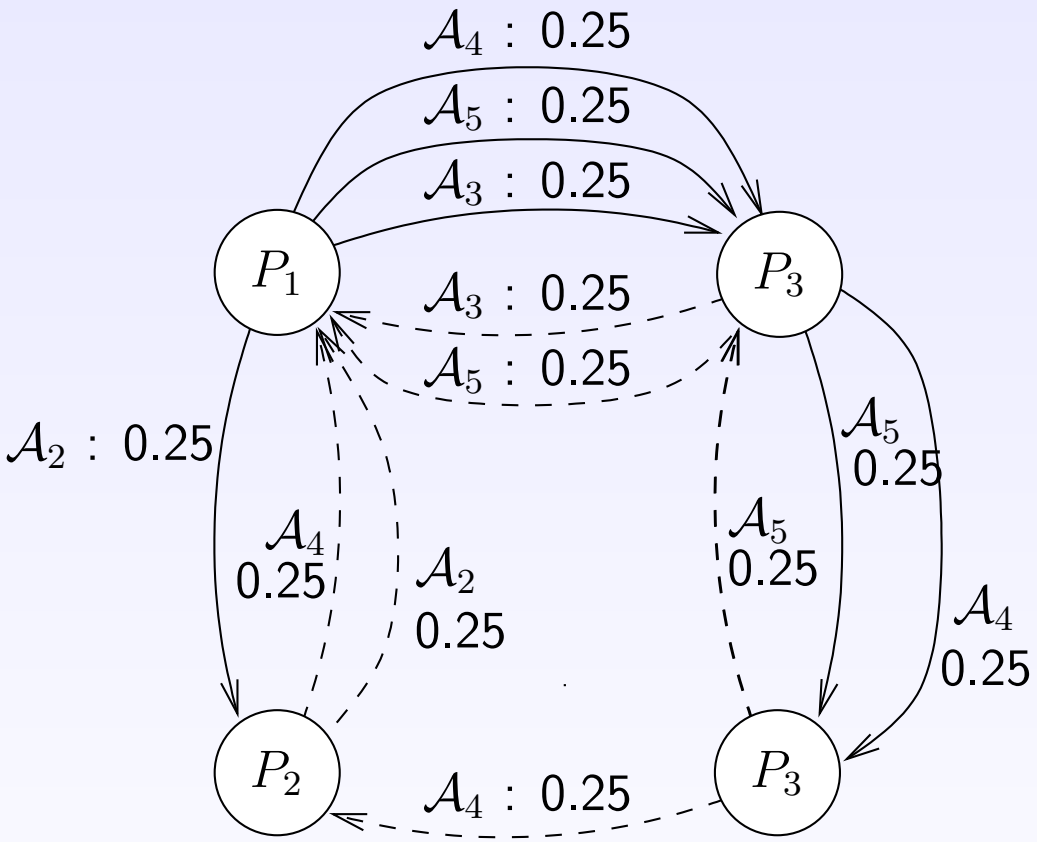
$\mathcal{A}_5$

0,125

How to orchestrate these allocations?

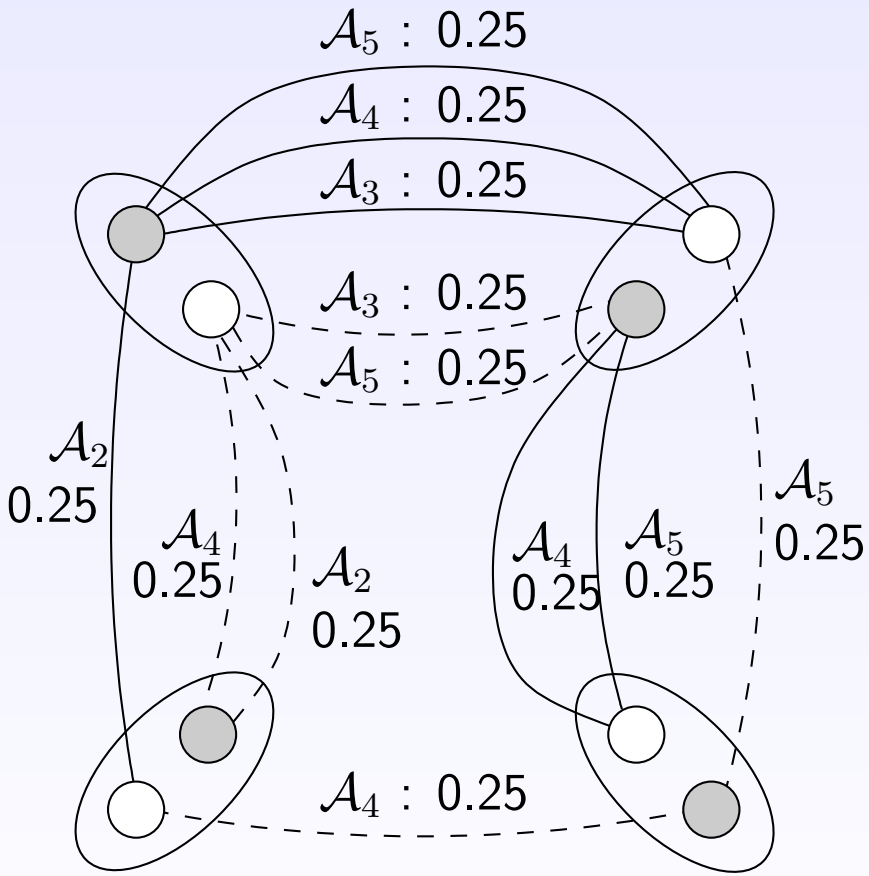


Communication graph

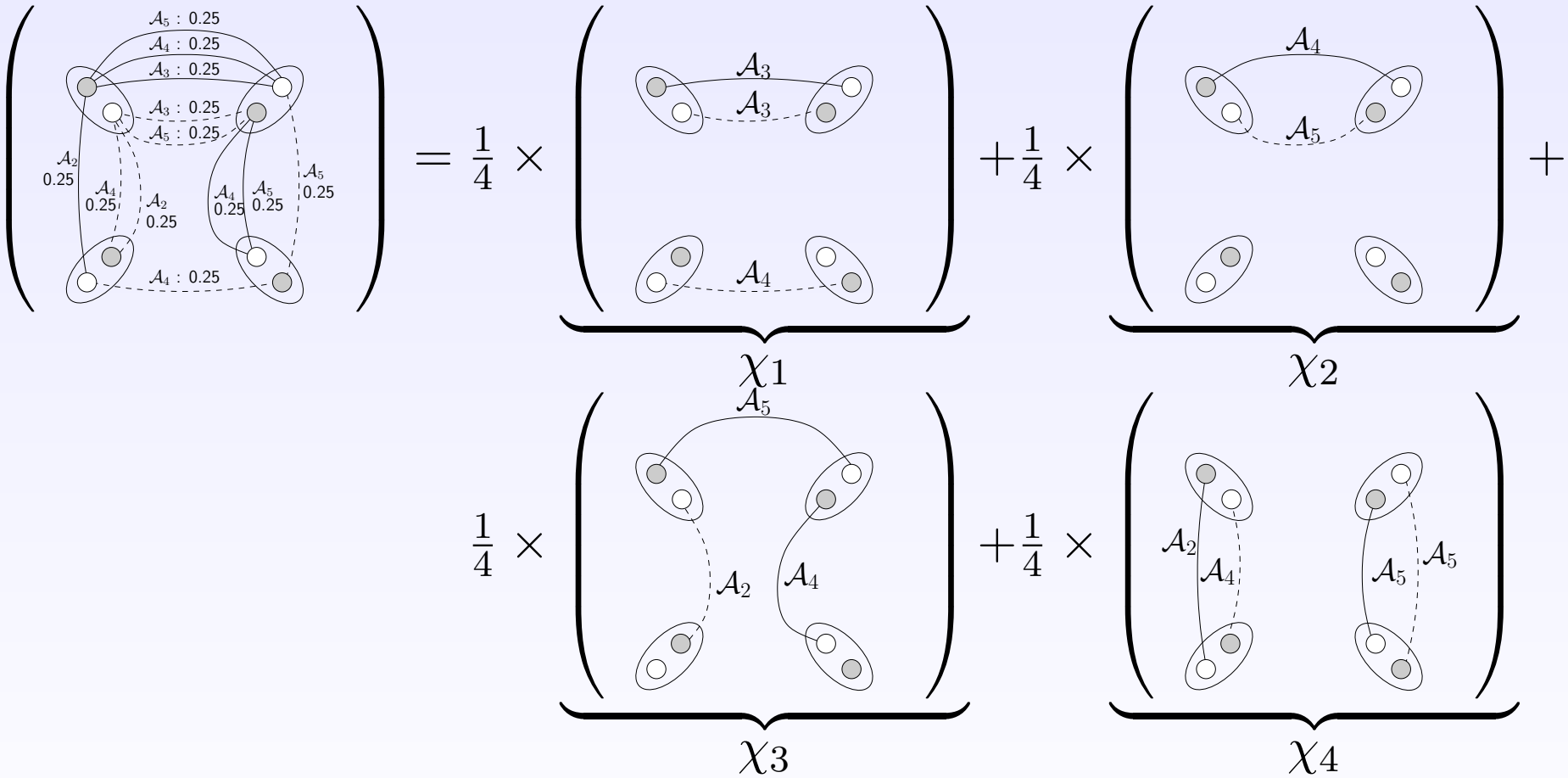


Fraction of time spent transferring some $e_{k,l}$ file from P_i to P_j for a given allocation

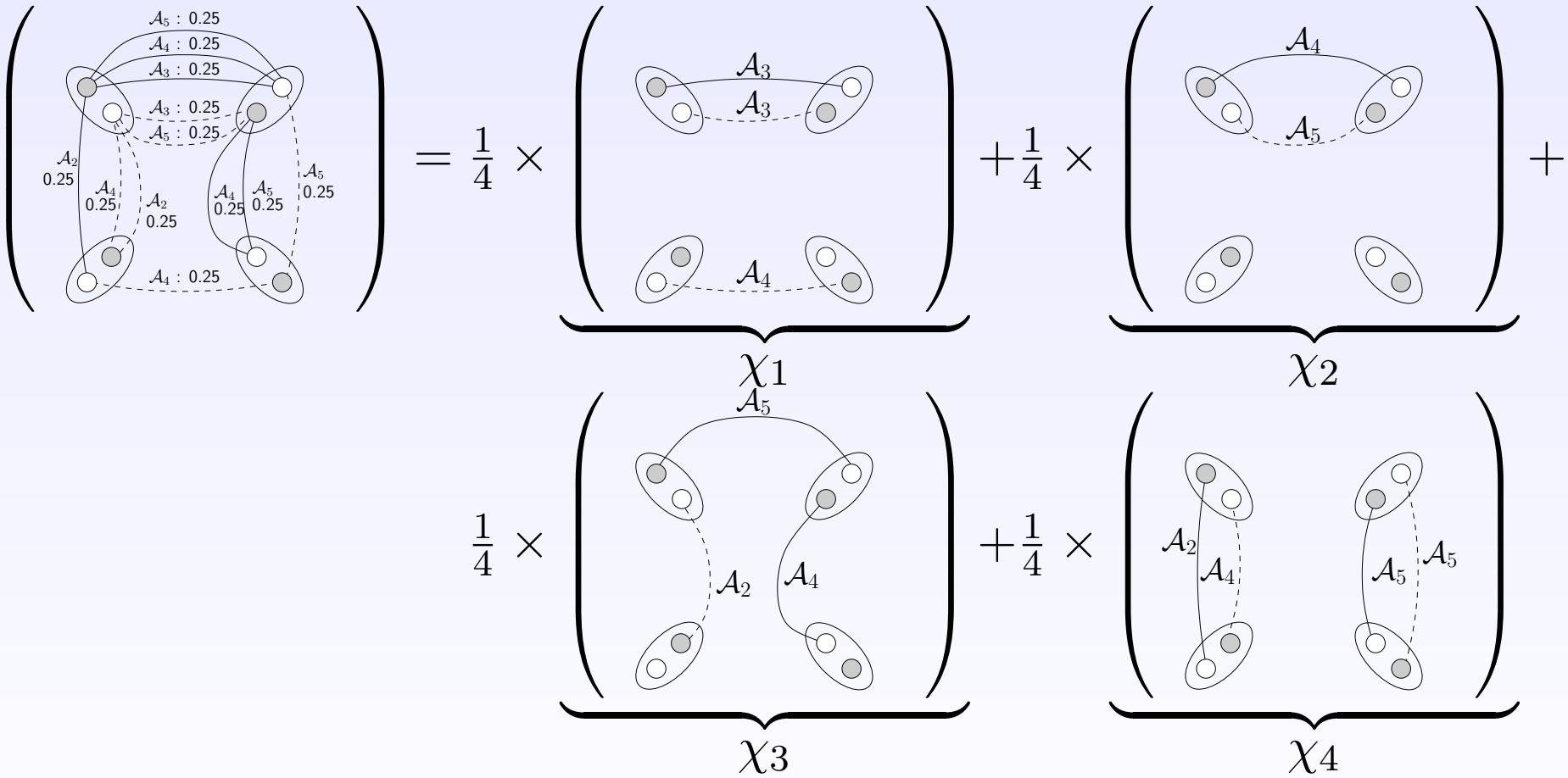
One-port constraints = matching



Edge coloring (decomposition into matchings)

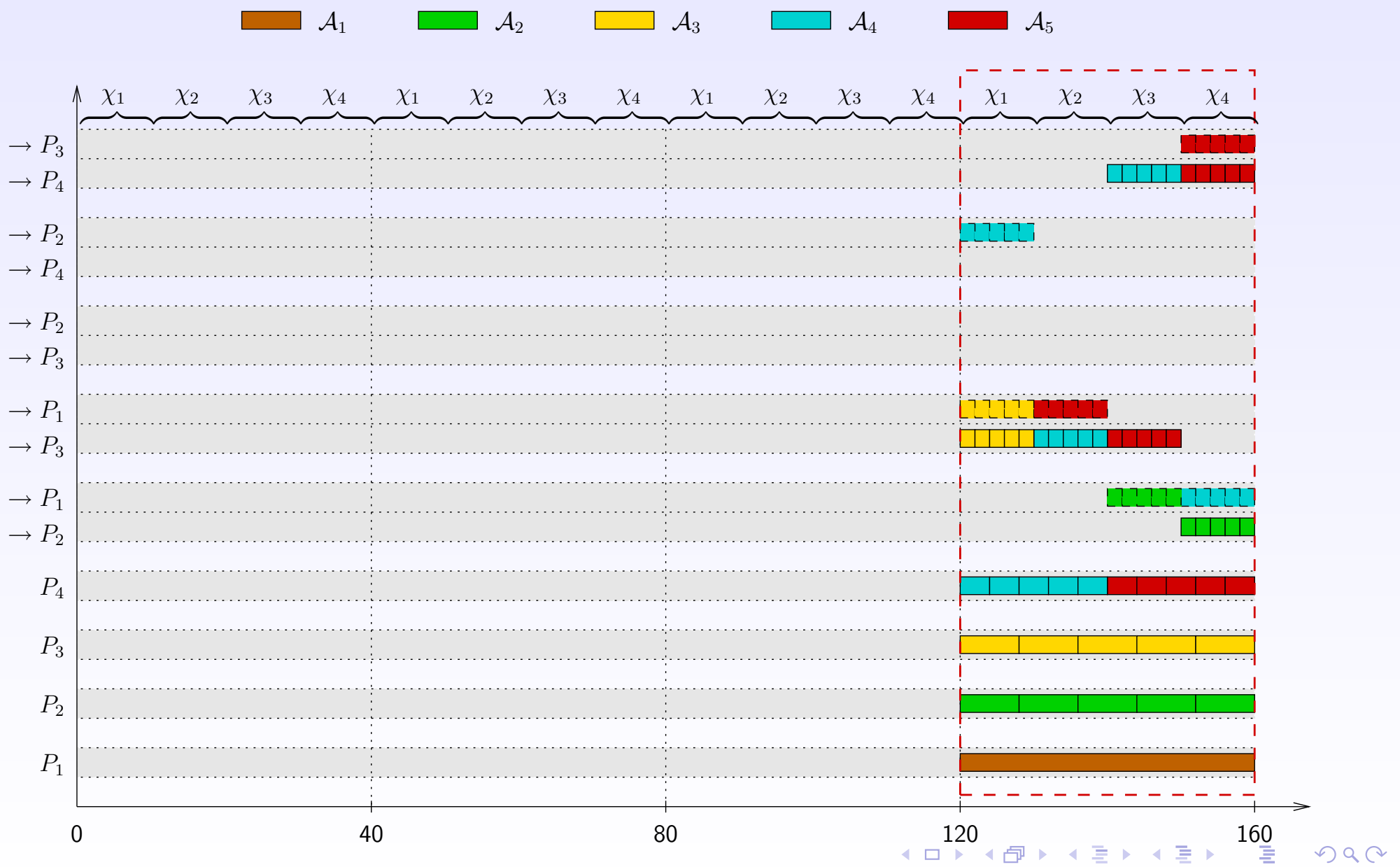


Edge coloring (decomposition into matchings)

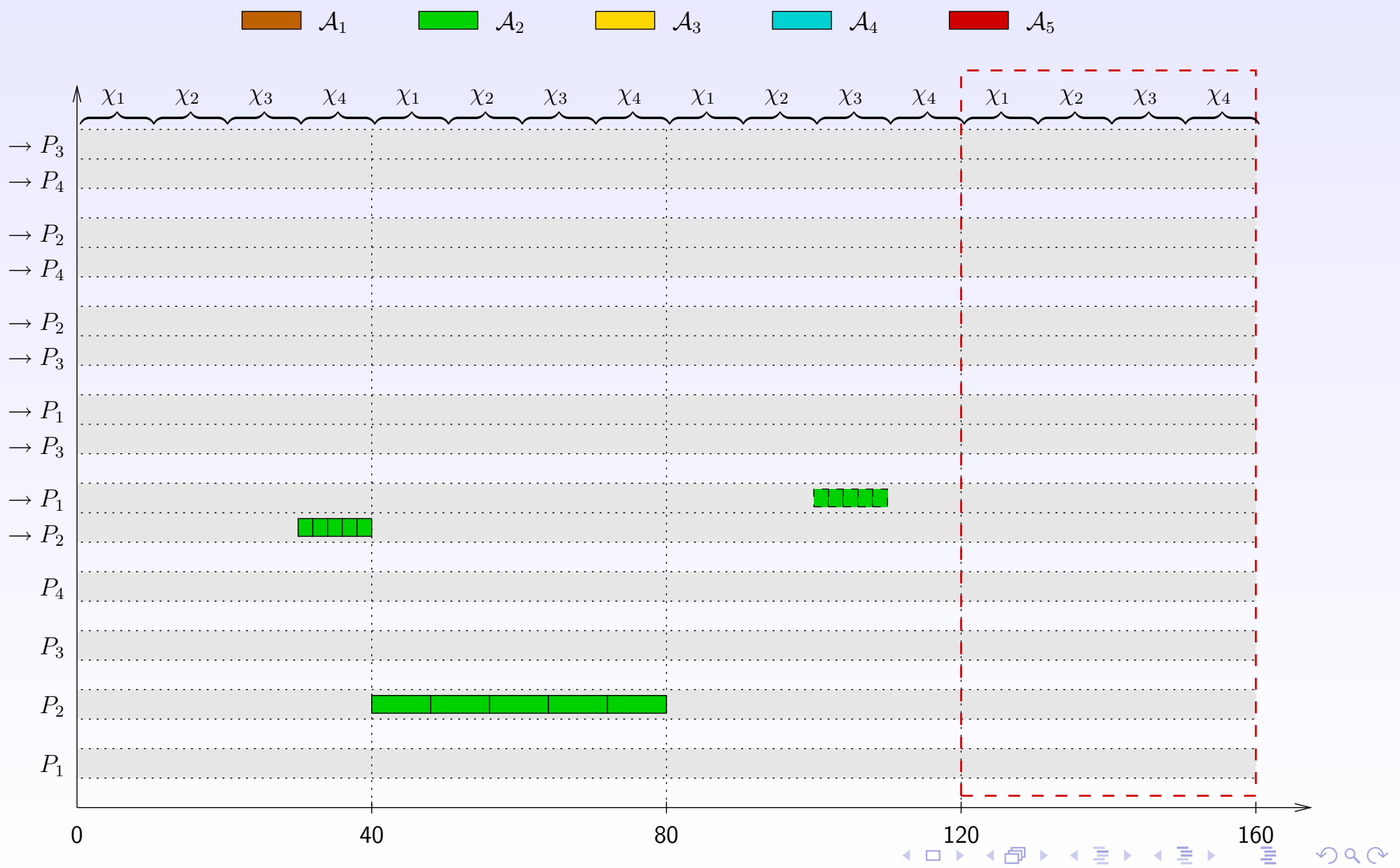


This decomposition is always possible

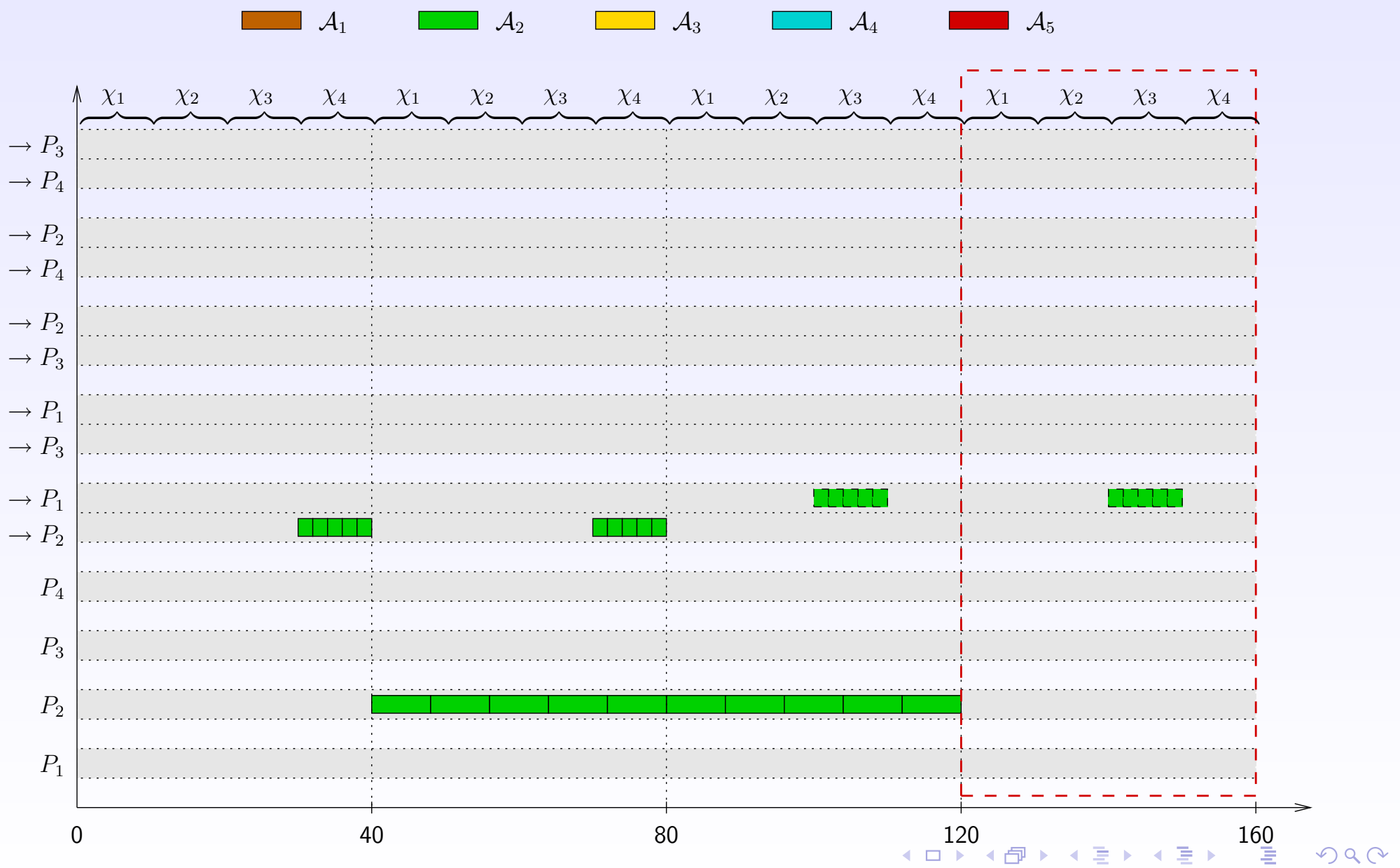
Cyclic scheduling achieving optimal throughput



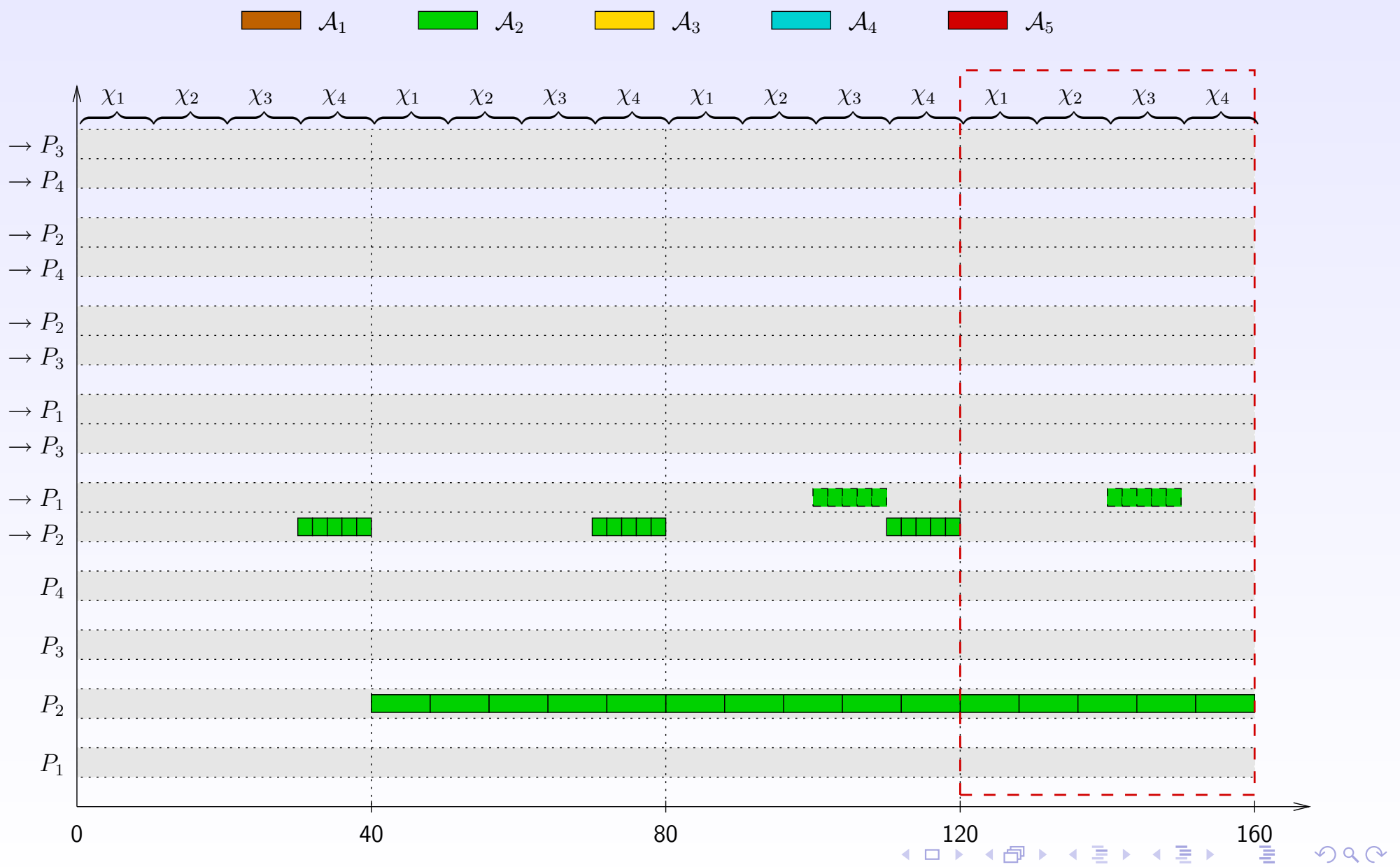
Cyclic scheduling achieving optimal throughput



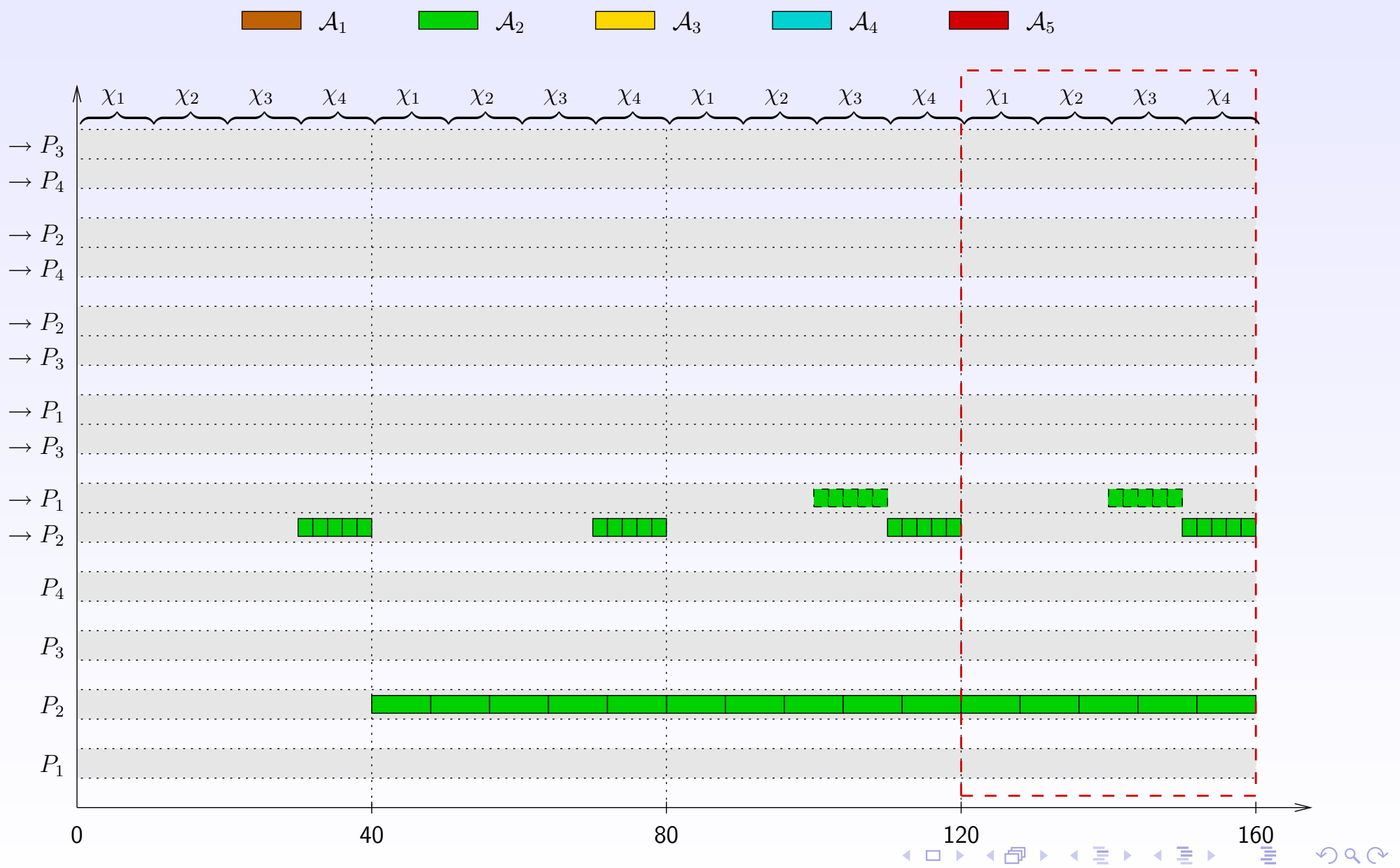
Cyclic scheduling achieving optimal throughput



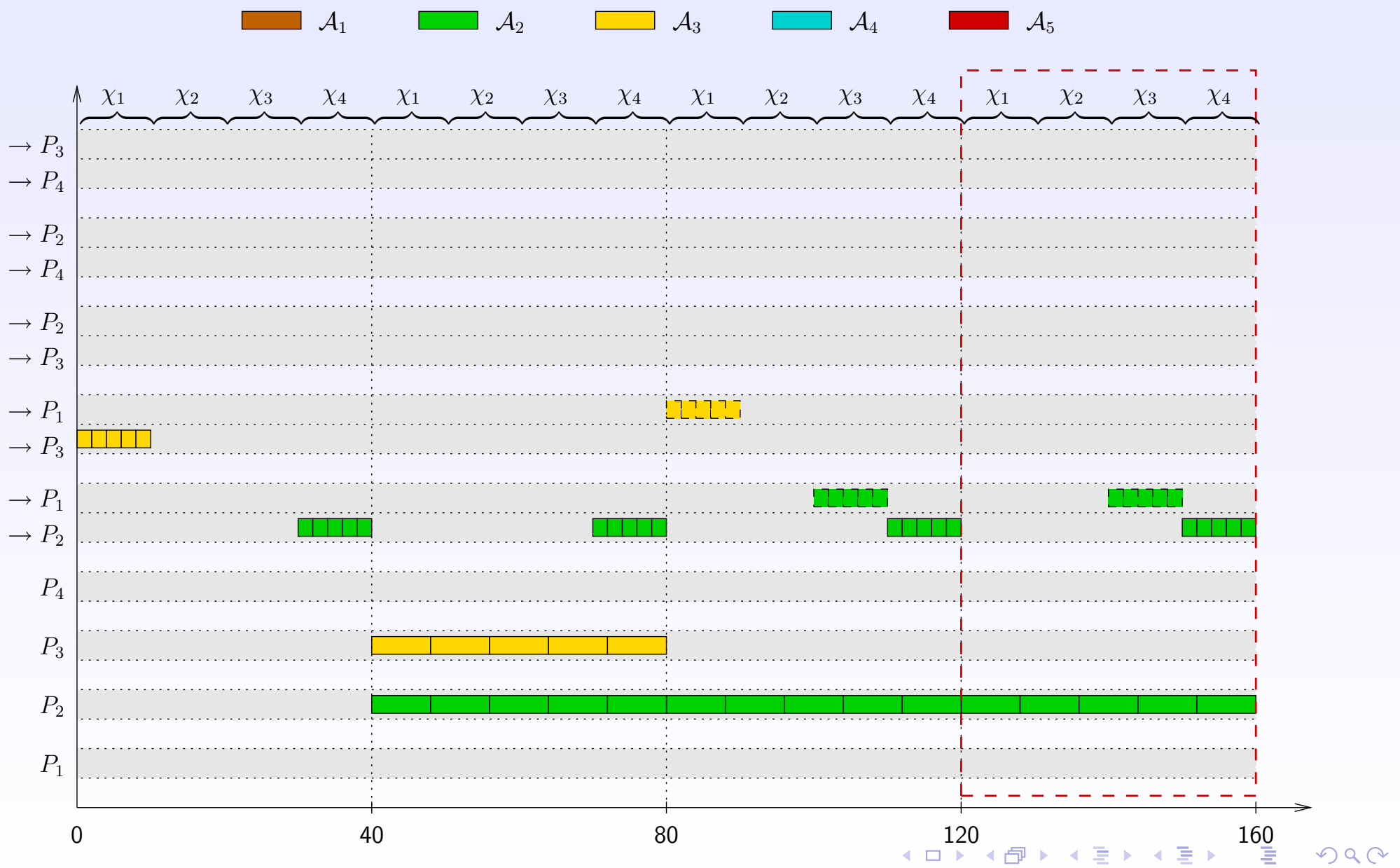
Cyclic scheduling achieving optimal throughput



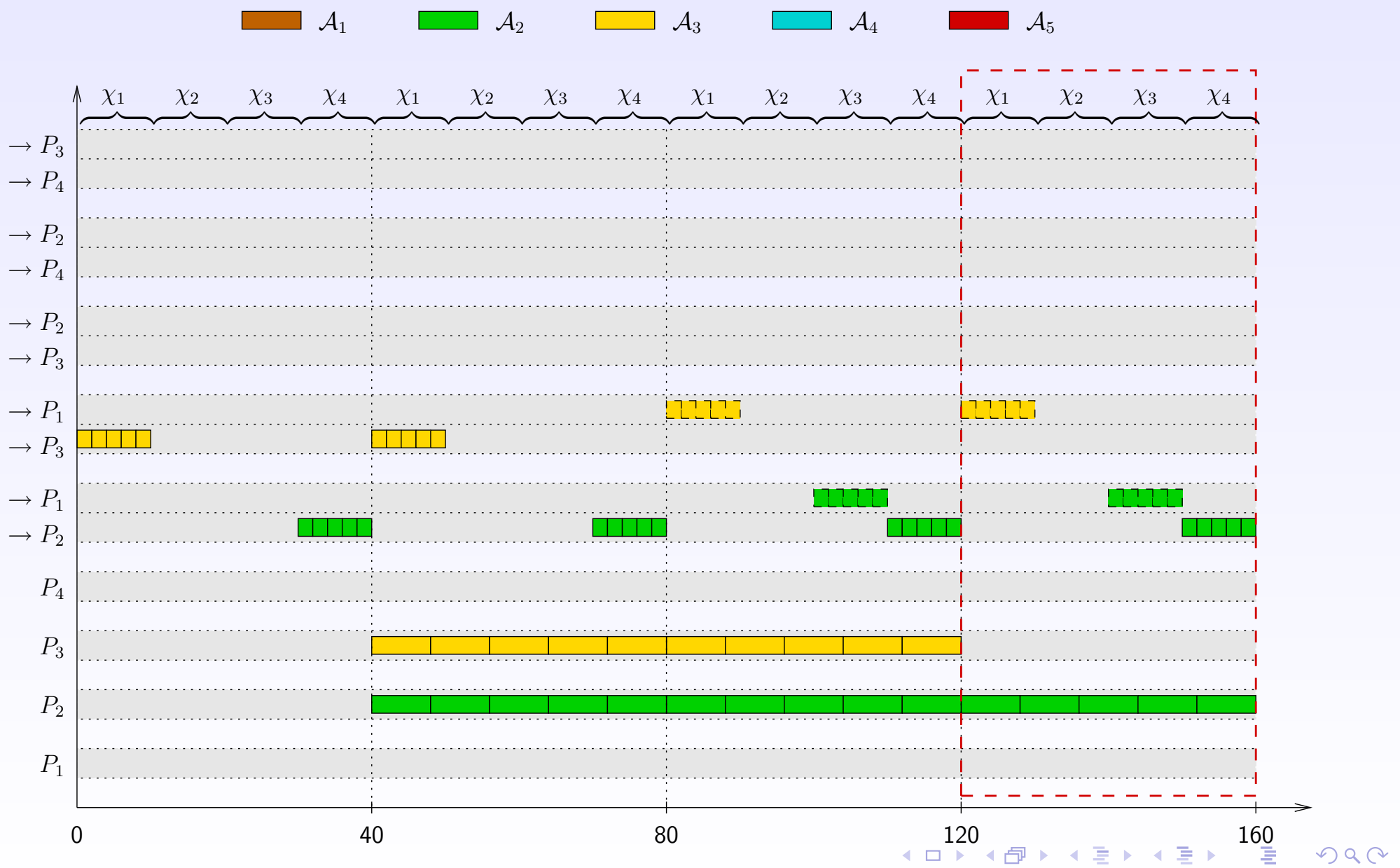
Cyclic scheduling achieving optimal throughput



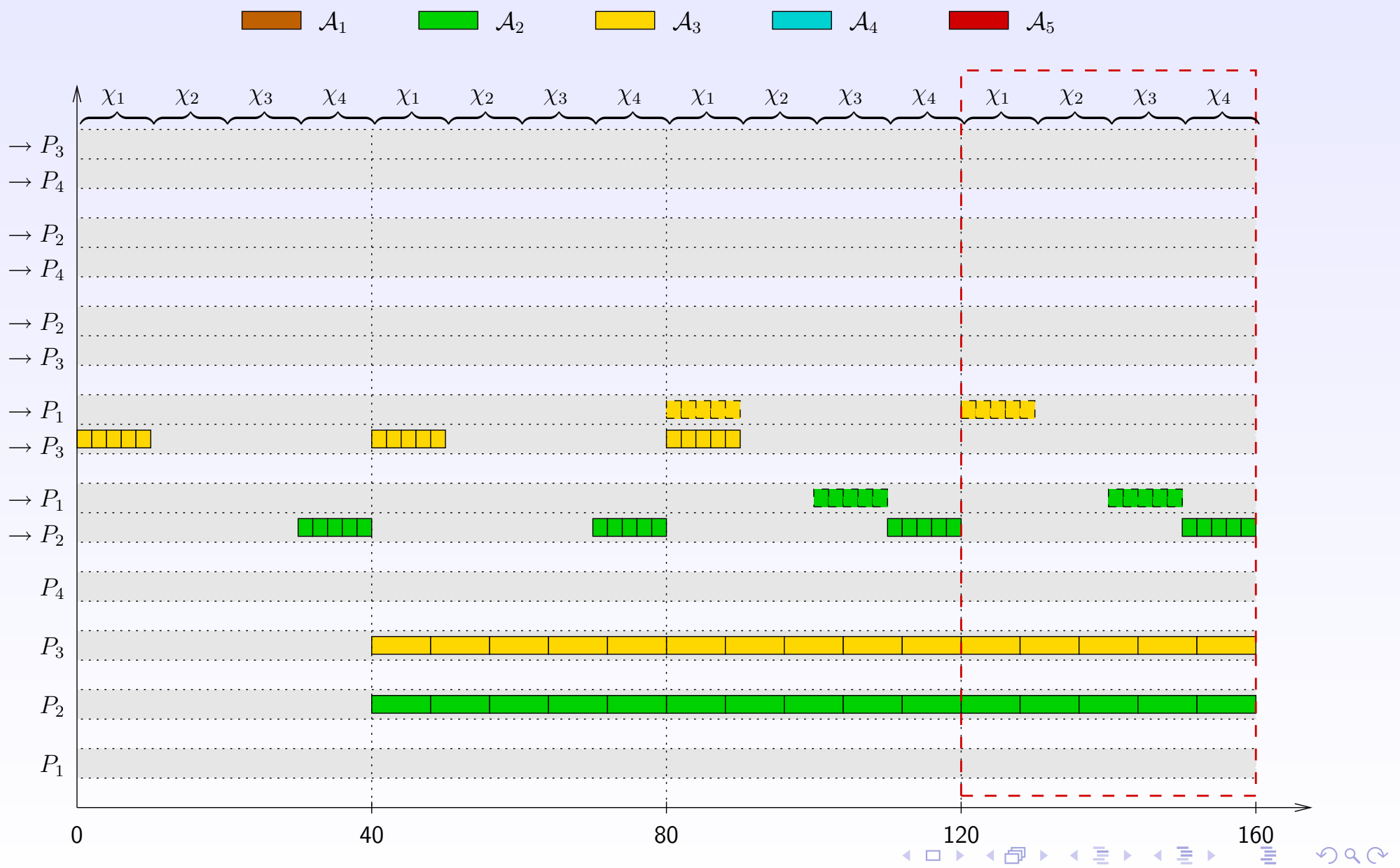
Cyclic scheduling achieving optimal throughput



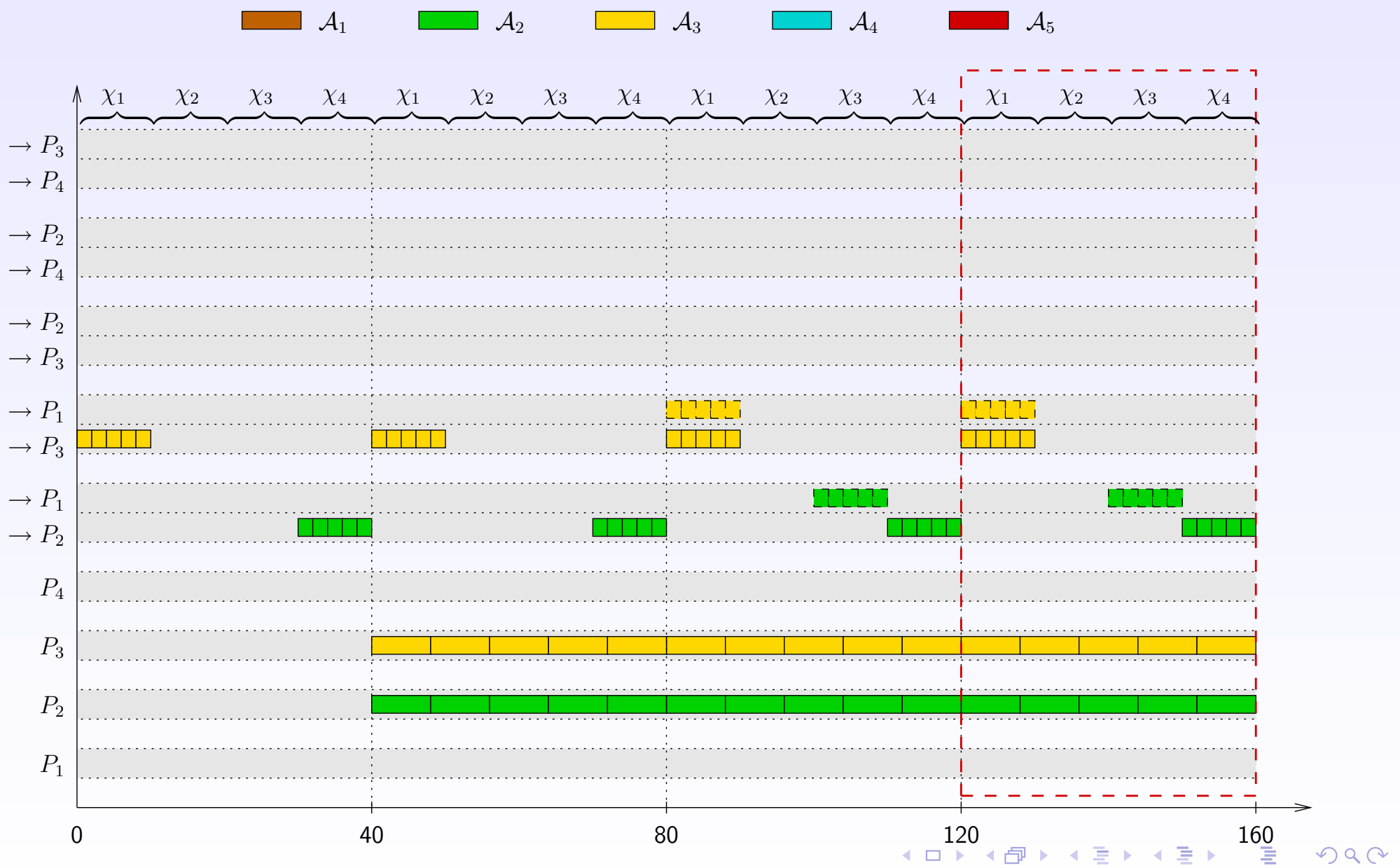
Cyclic scheduling achieving optimal throughput



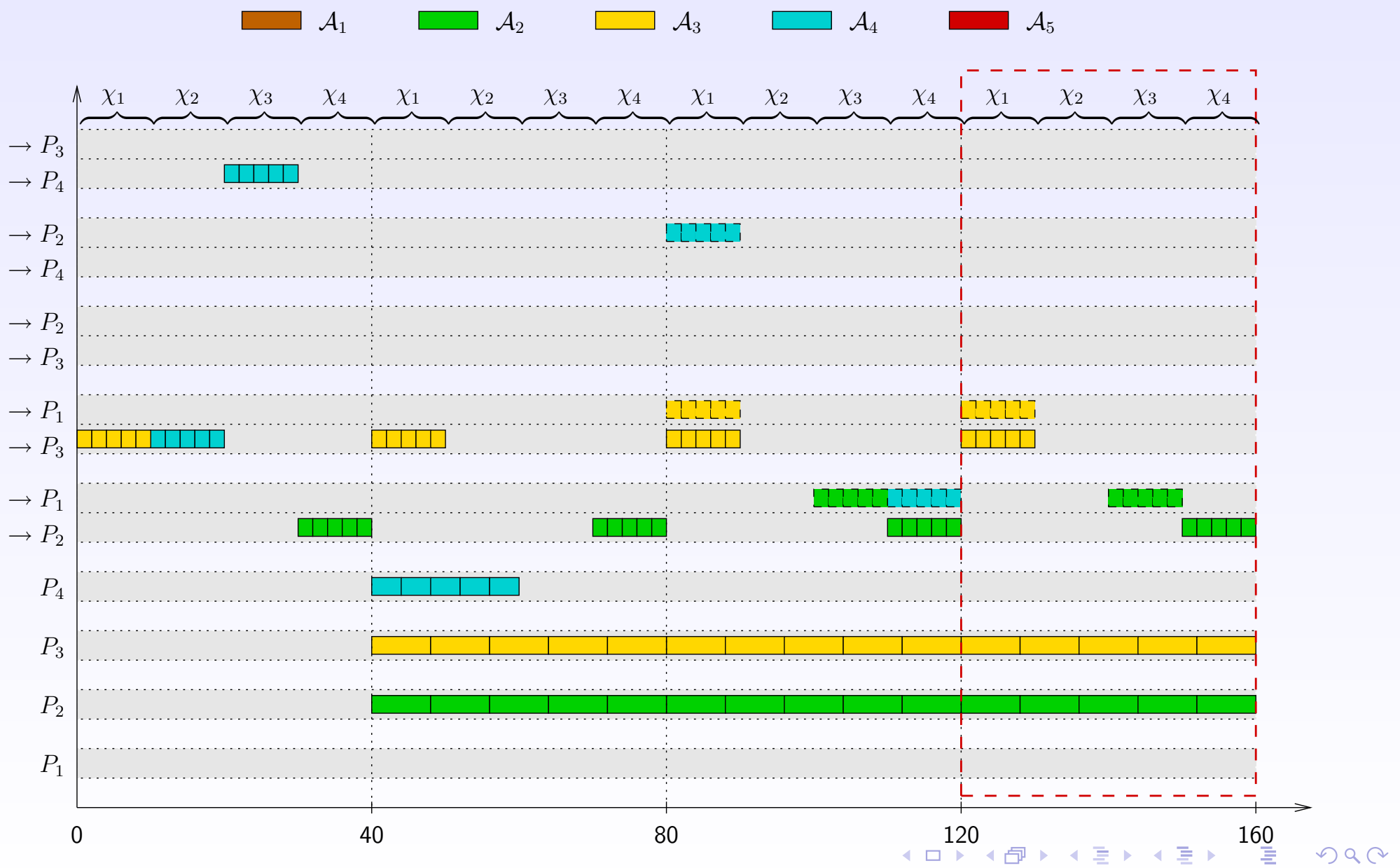
Cyclic scheduling achieving optimal throughput



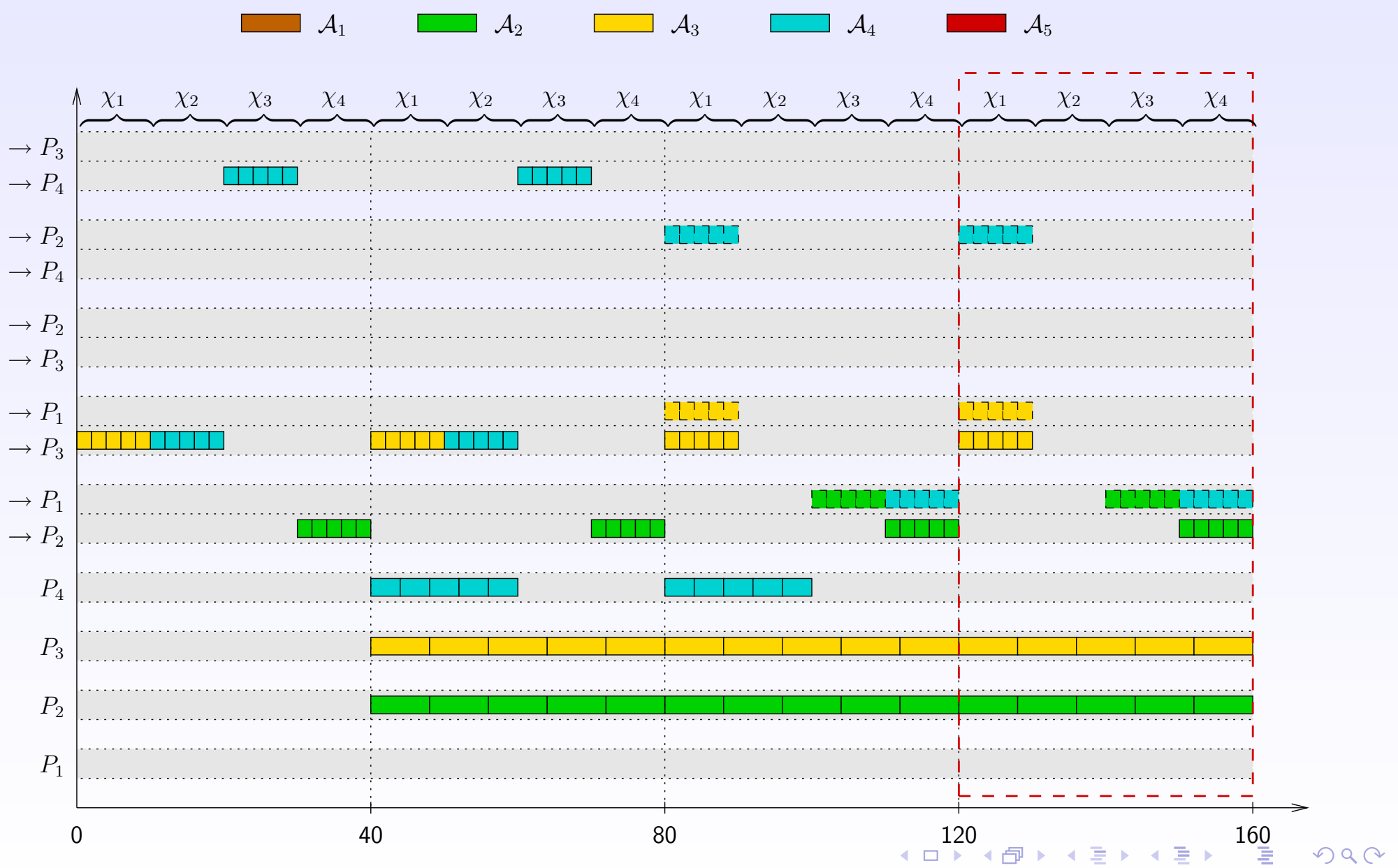
Cyclic scheduling achieving optimal throughput



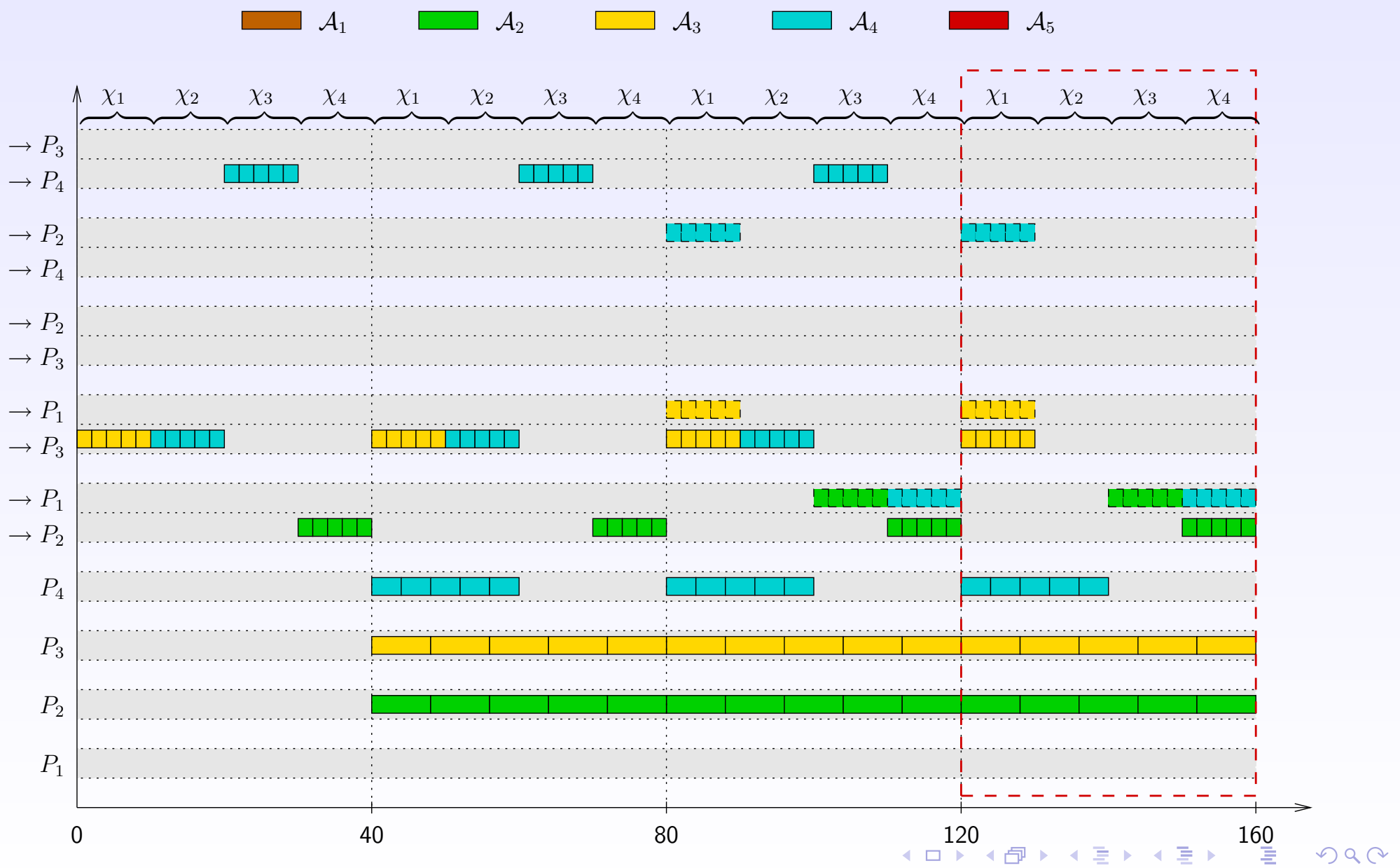
Cyclic scheduling achieving optimal throughput



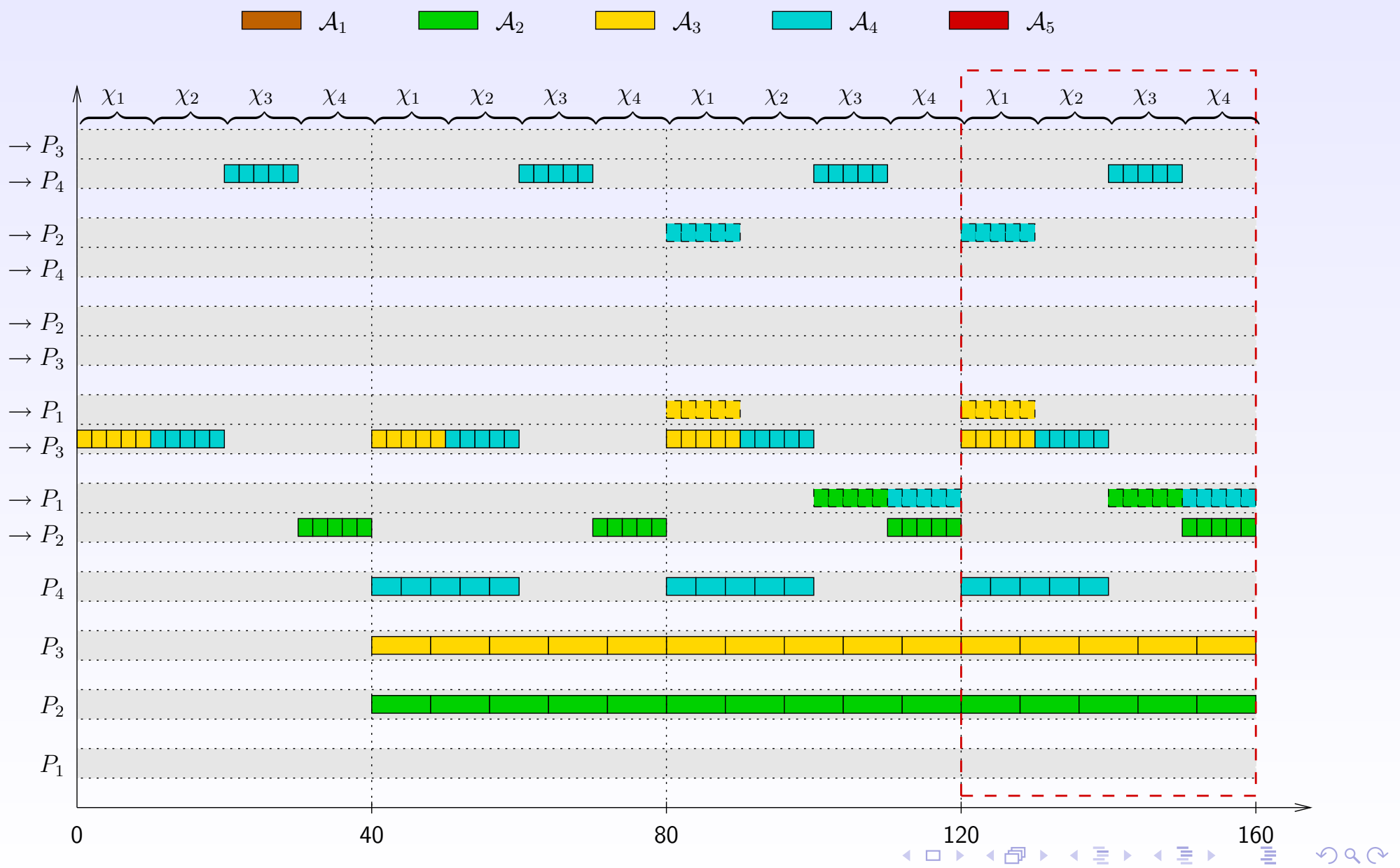
Cyclic scheduling achieving optimal throughput



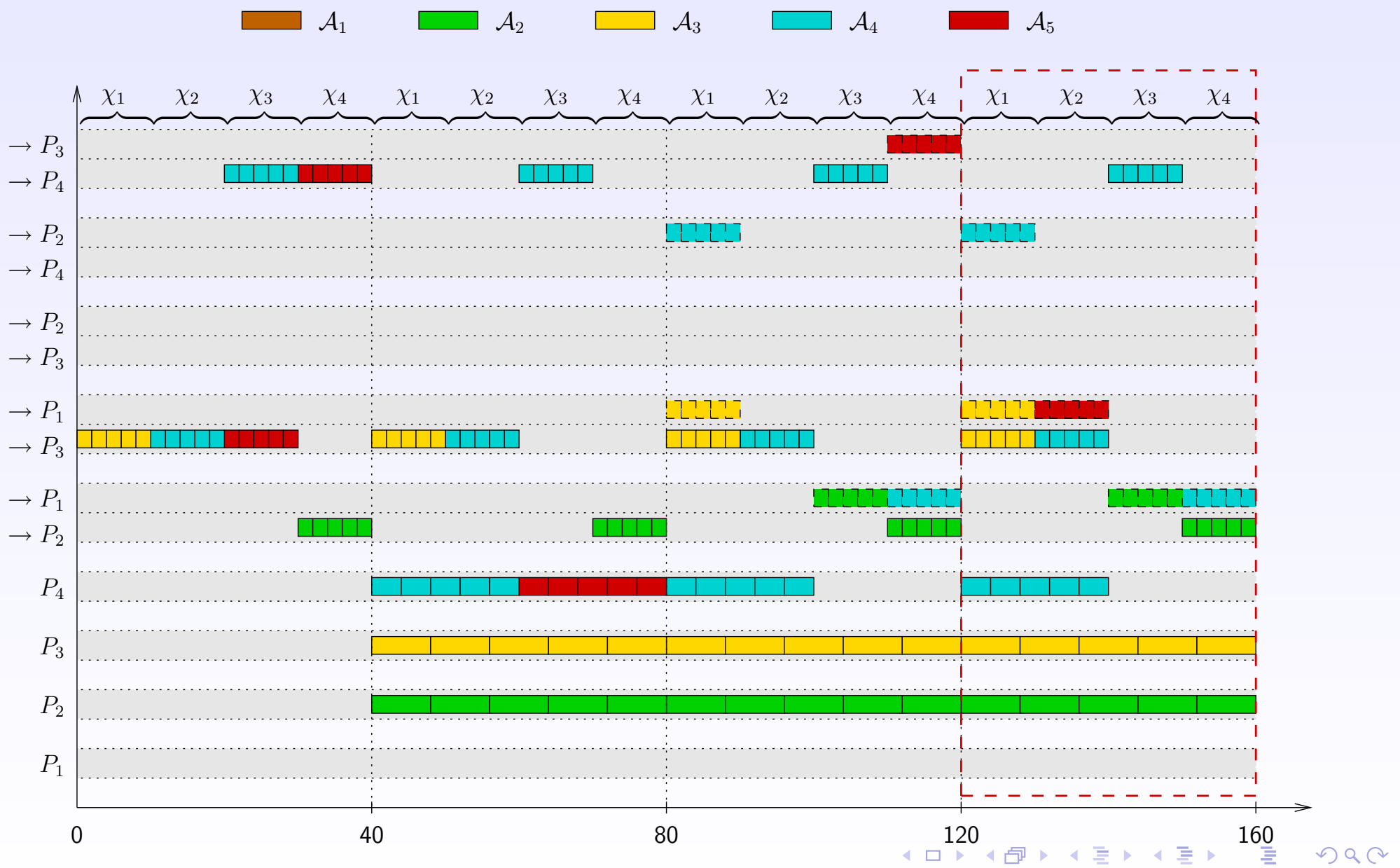
Cyclic scheduling achieving optimal throughput



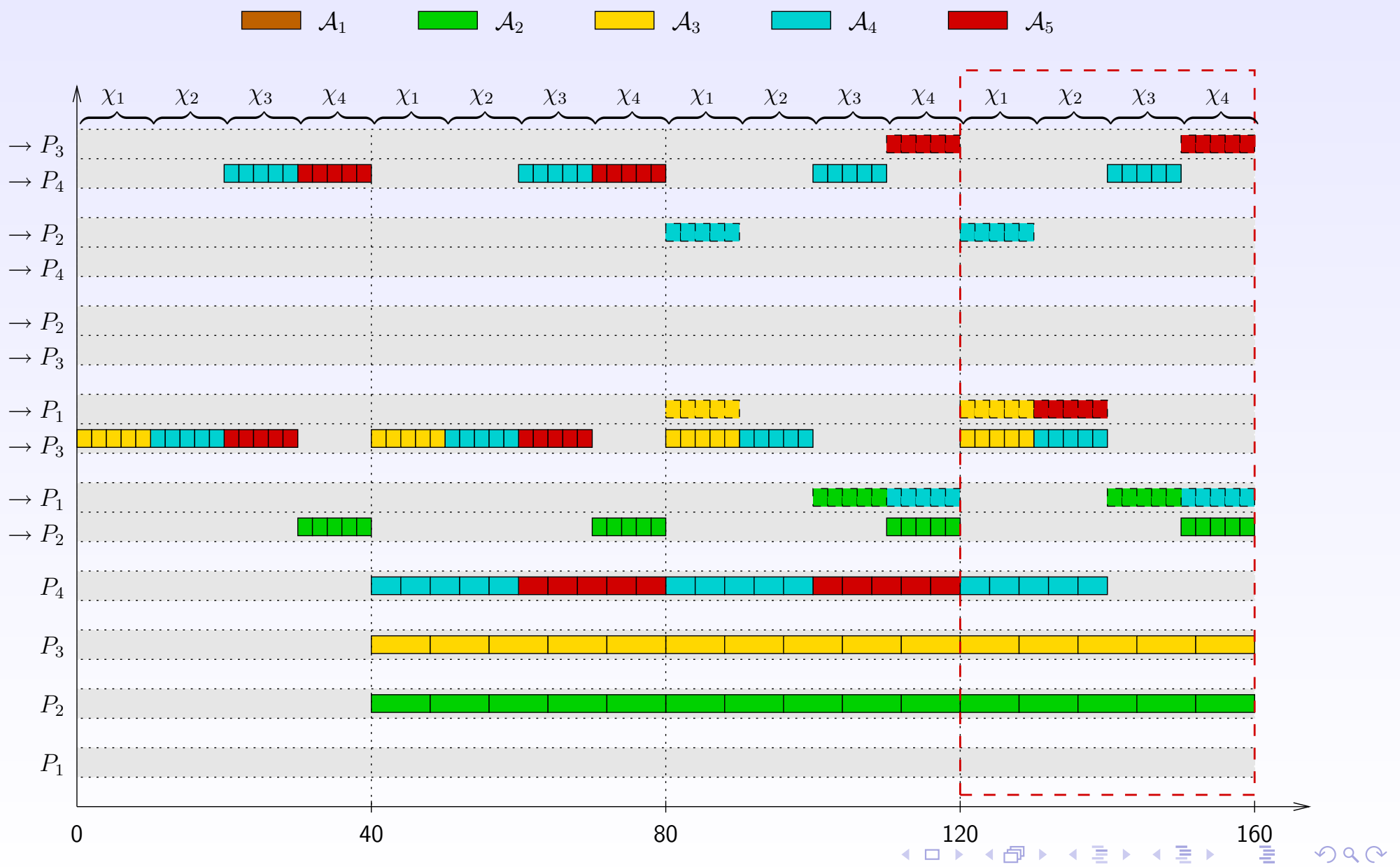
Cyclic scheduling achieving optimal throughput



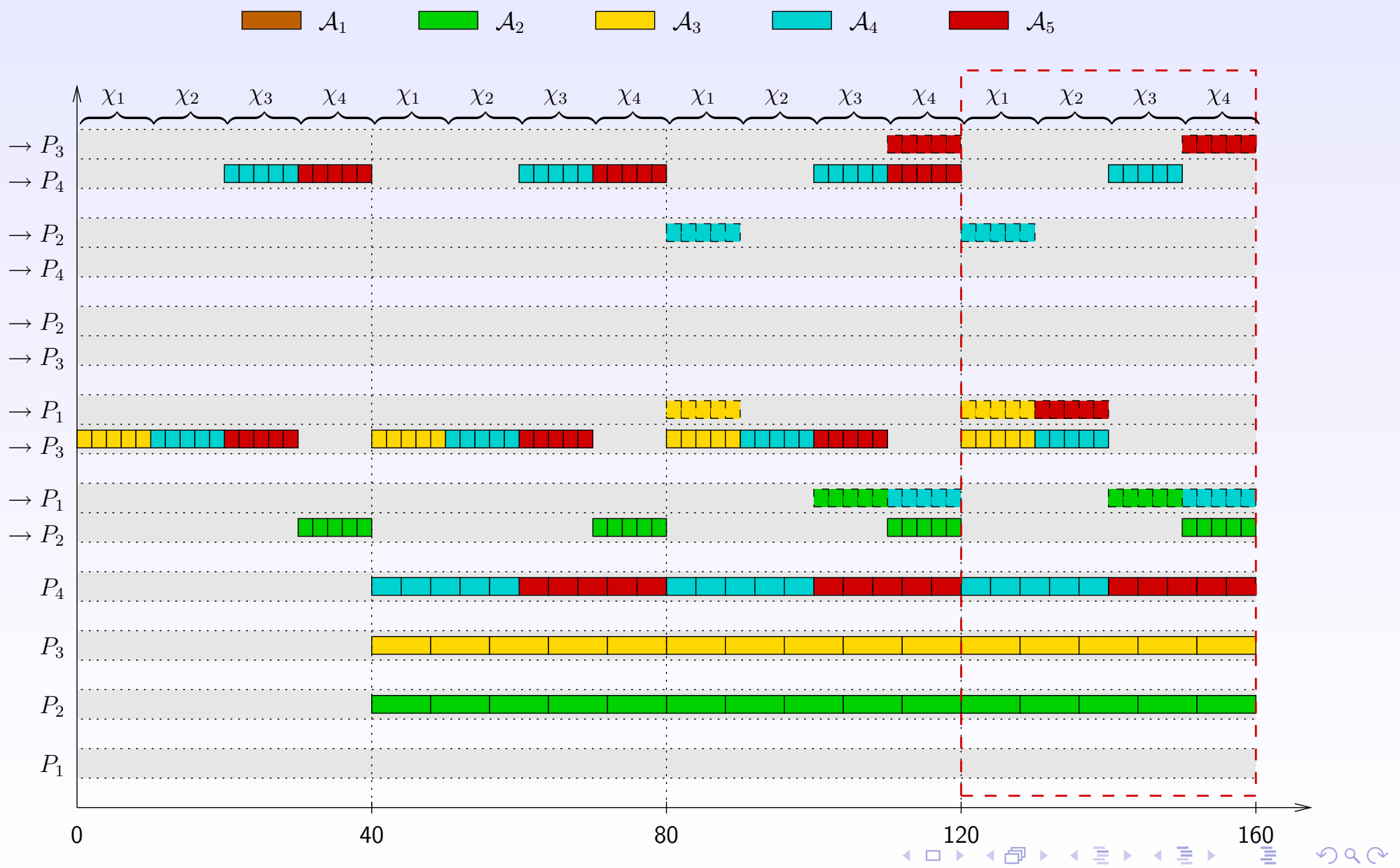
Cyclic scheduling achieving optimal throughput



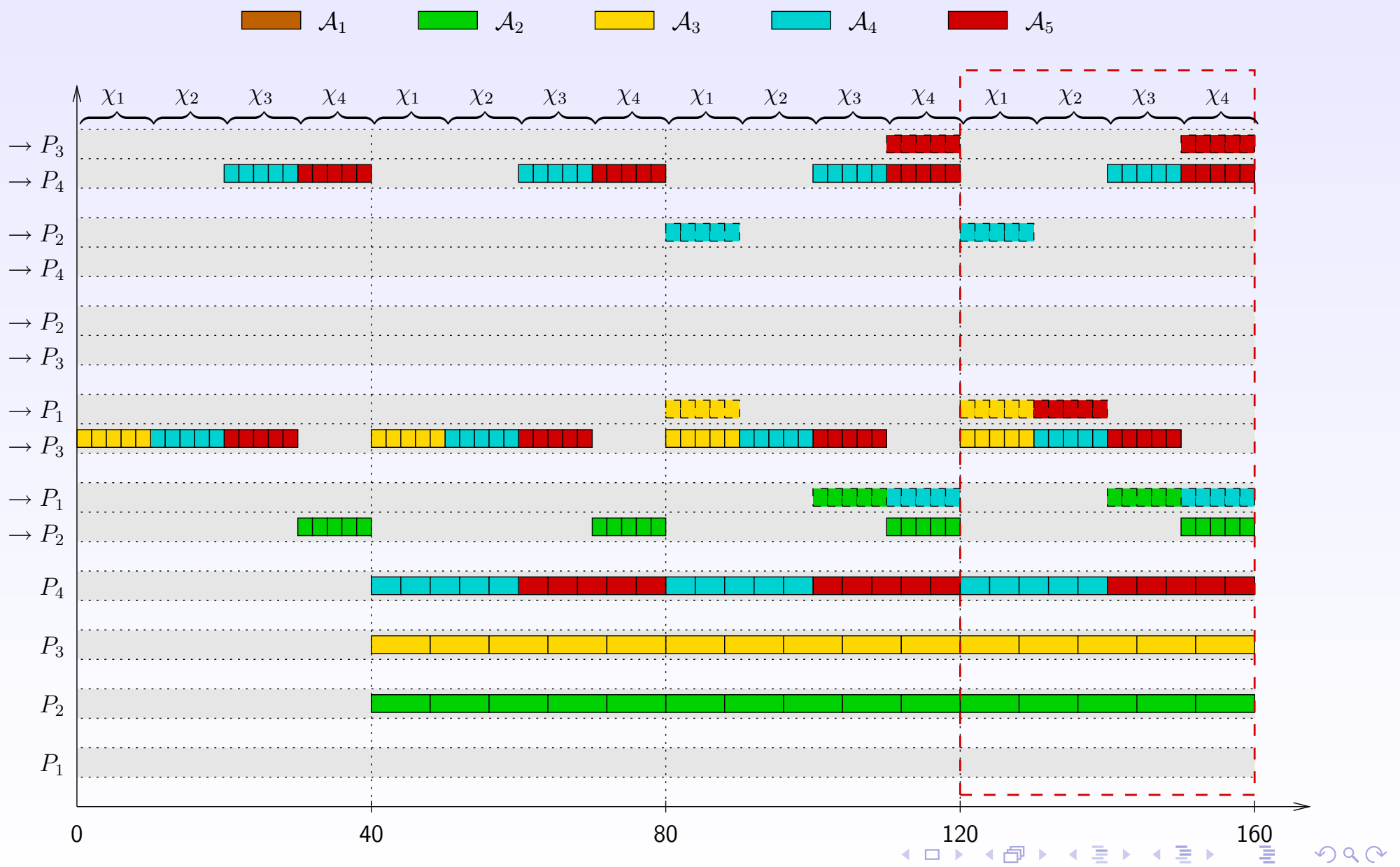
Cyclic scheduling achieving optimal throughput



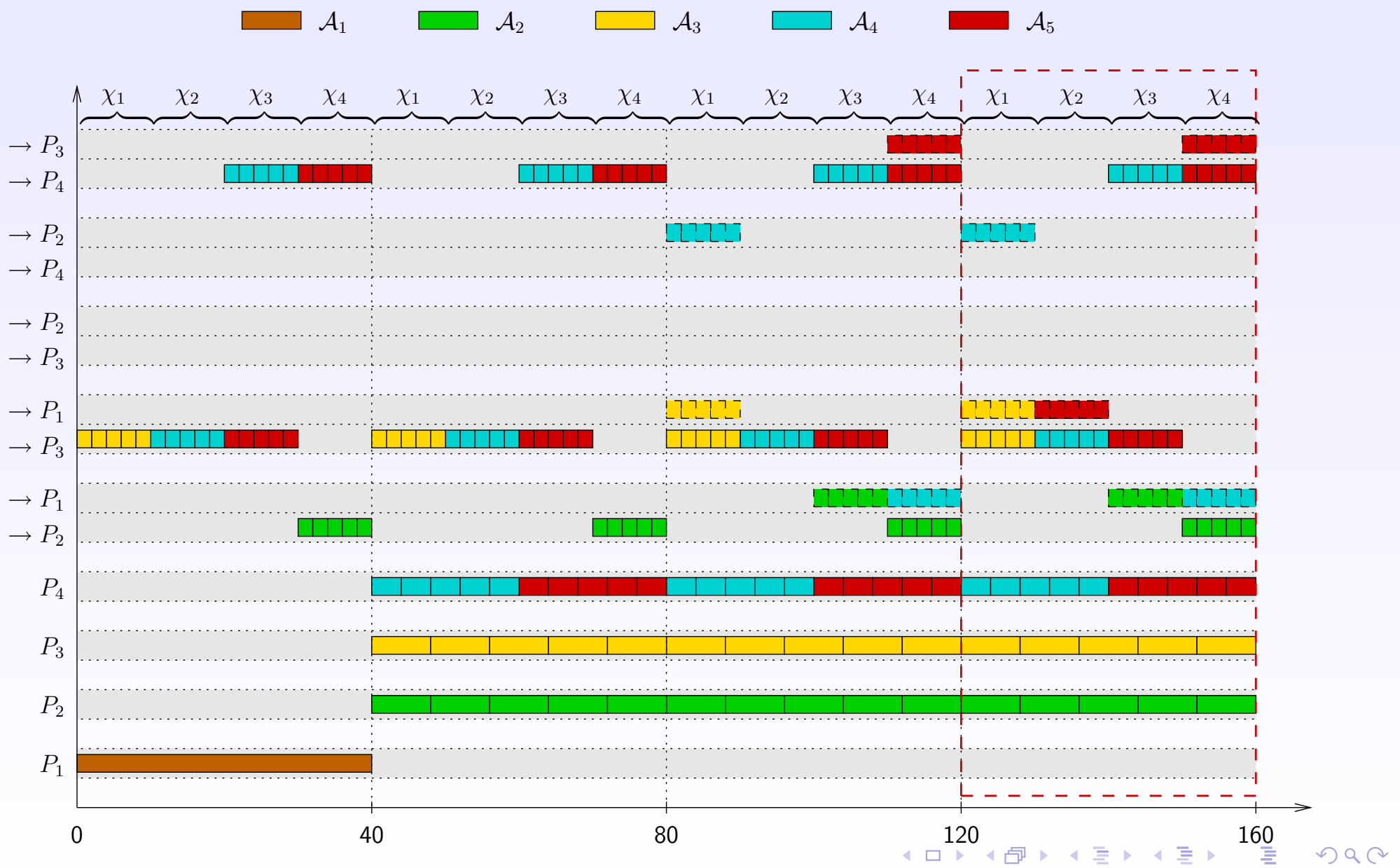
Cyclic scheduling achieving optimal throughput



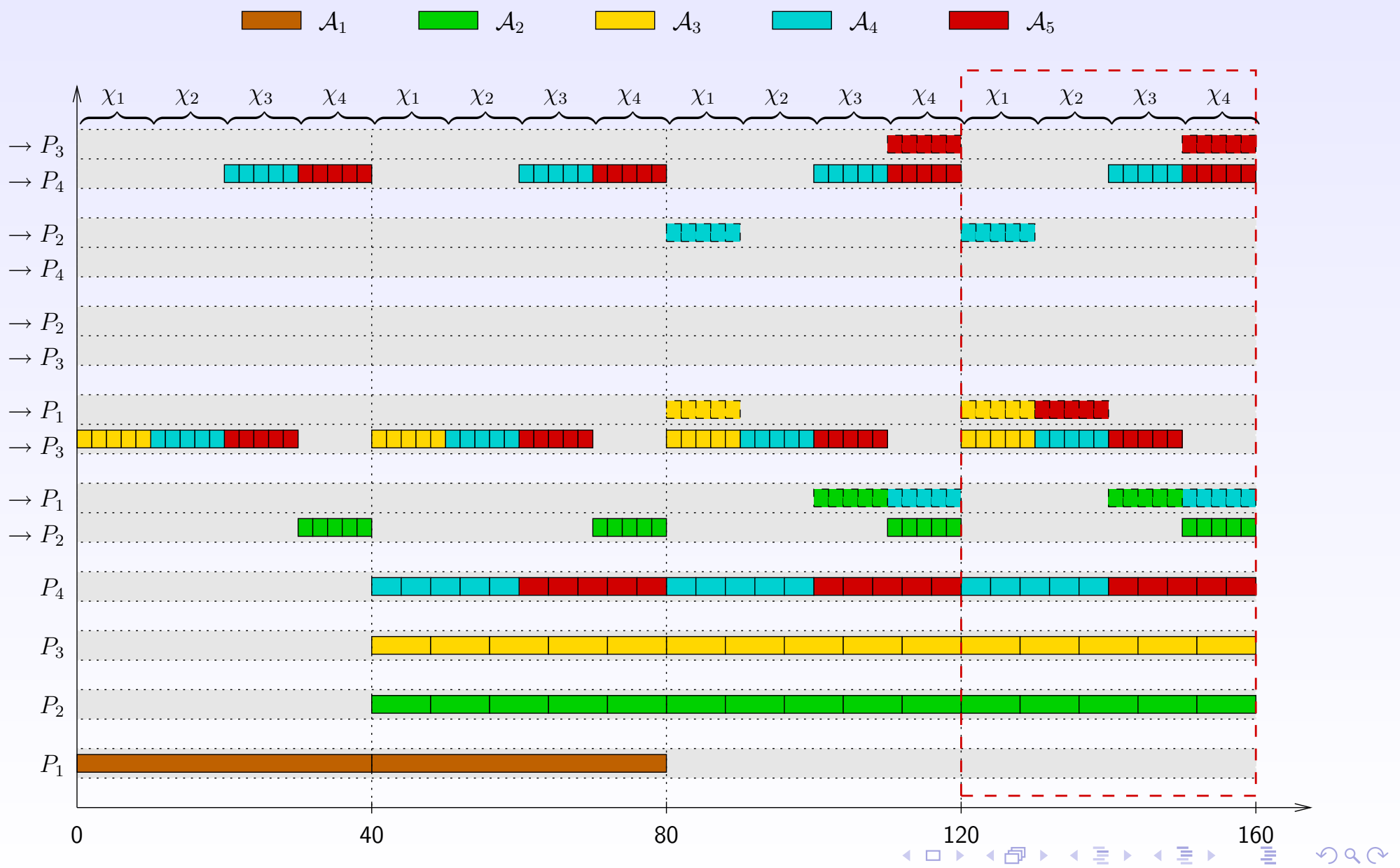
Cyclic scheduling achieving optimal throughput



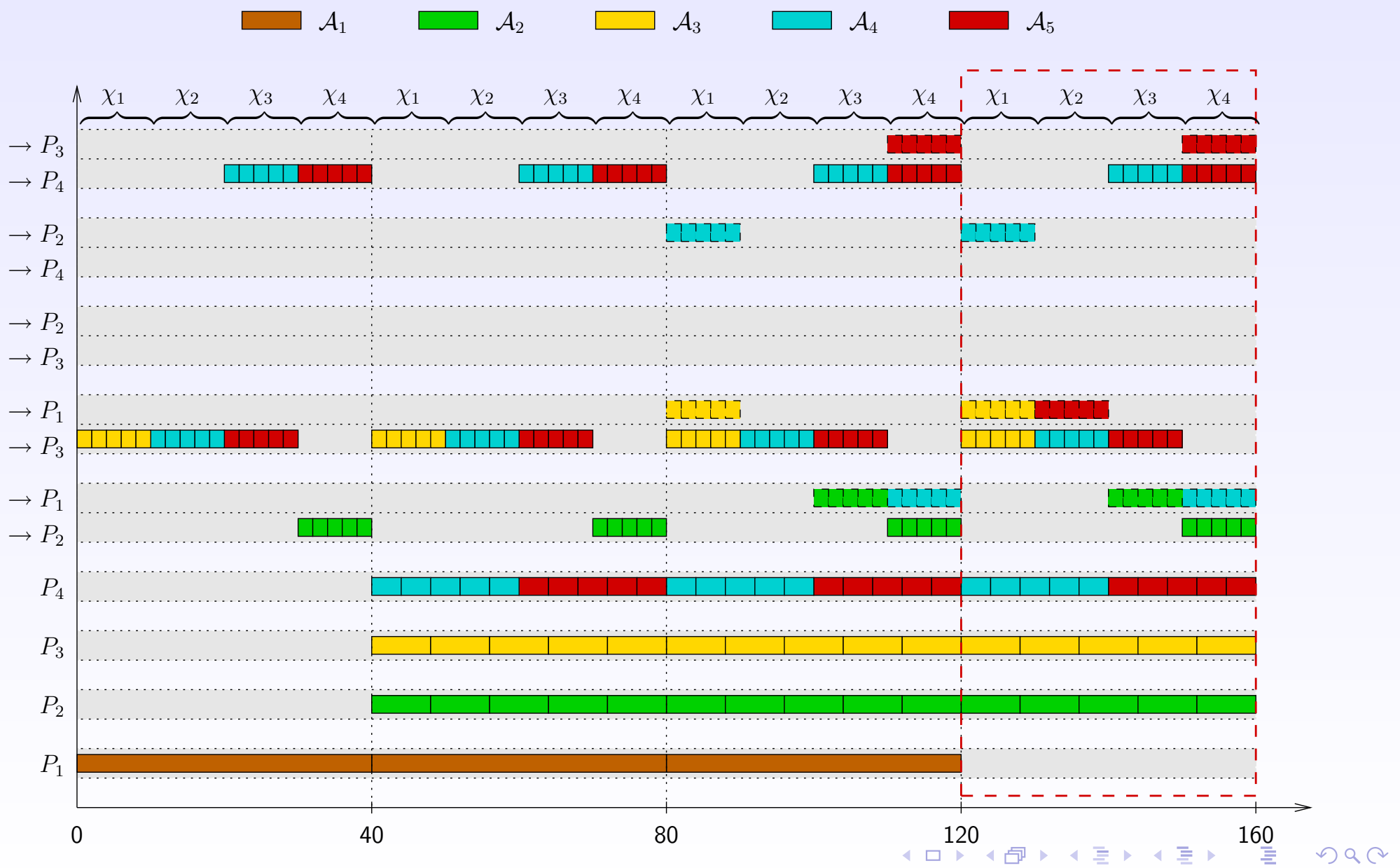
Cyclic scheduling achieving optimal throughput



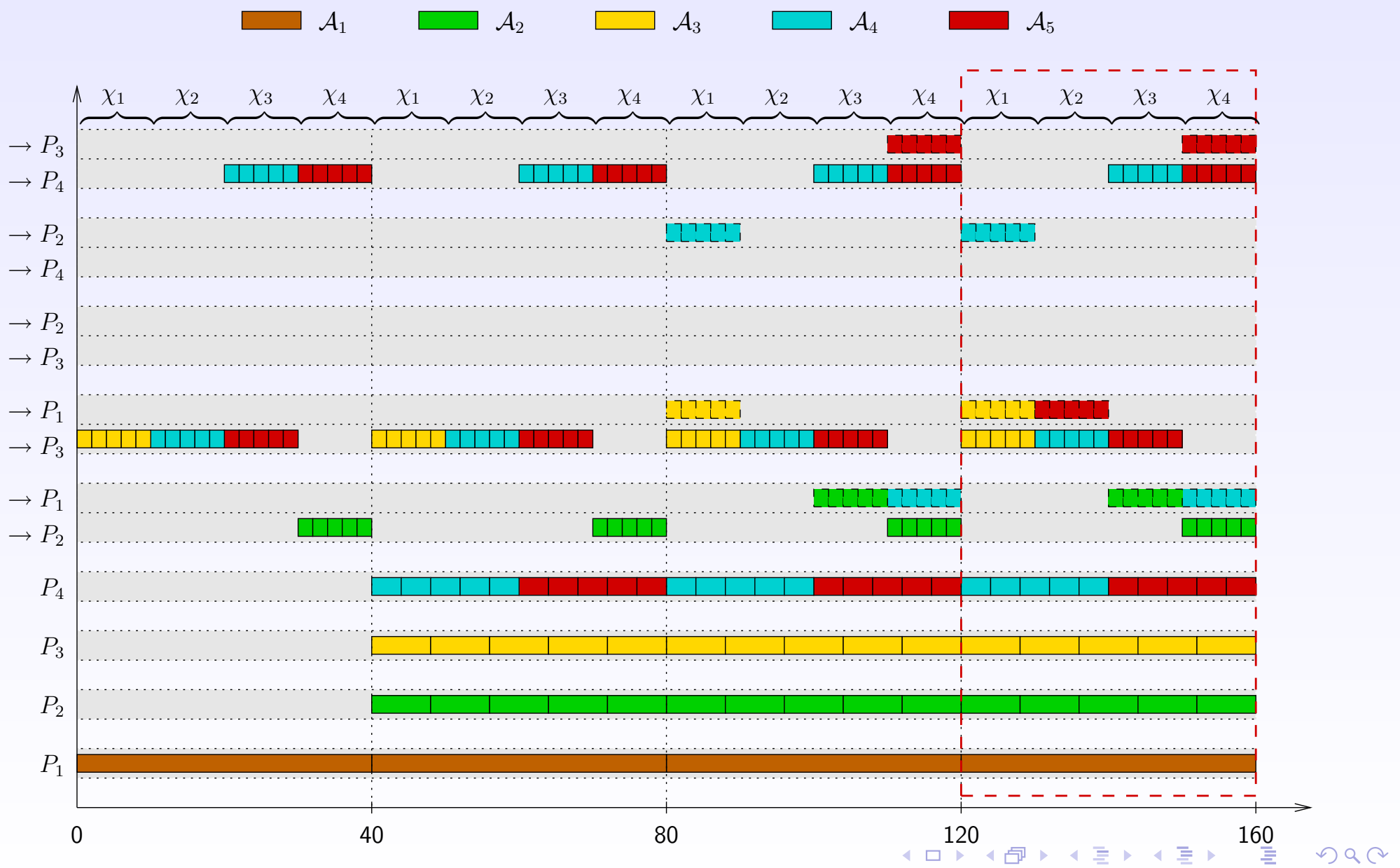
Cyclic scheduling achieving optimal throughput



Cyclic scheduling achieving optimal throughput



Cyclic scheduling achieving optimal throughput



Asymptotically optimal schedule

- The technique used in the example is (i) general and (ii) polynomial
- The resulting schedule is *asymptotically optimal*: within T time-steps, it differs from the optimal schedule by a constant number of tasks (independent of T)

Extensions to collections of general task graphs

- More difficult but possible
- Maximizing throughput NP-hard ☹
- Most application DAGs have polynomial number of joins
  $\Rightarrow$ polynomial solution ☺

Perspectives

- Macro-communications (*scatter*, *gather*, *reduce*, *broadcast*, *multicast*,...)
- Open problems:
  - Period length, approximating cyclic pattern
  - When problem remains difficult after steady-state relaxation?
  - Stability, robustness in front of load variations

Bibliography

- Packet routing:
  Asymptotically optimal algorithms for job shop scheduling and packet routing, D. Bertsimas and D. Gamarnik, Journal of Algorithms 33, 2 (1999), 296-318
- Steady-state scheduling:
  Scheduling strategies for master-slave tasking on heterogeneous processor platforms, C. Banino et al., IEEE TPDS 15, 4 (2004), 319-330
- With bounded multi-port model:
  Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput, B. Hong and V.K. Prasanna, IEEE IPDPS (2004), 52b

Outline

Good news and bad news

- ☺ One-port model: first step towards designing realistic scheduling heuristics
- ☺ Perfectly divisible load simplifies task allocation
- ☺ Steady-state circumvents complexity of scheduling problems . . . while deriving efficient (often asympotically optimal) scheduling algorithms

- ☹ Need to acquire a good knowledge of the platform graph
- ☹ Need to run extensive experiments or simulations

Good news and bad news

- ☺ One-port model: first step towards designing realistic scheduling heuristics
- ☺ Perfectly divisible load simplifies task allocation
- ☺ Steady-state circumvents complexity of scheduling problems . . . while deriving efficient (often asympotically optimal) scheduling algorithms

- ☹ Need to acquire a good knowledge of the platform graph
- ☹ Need to run extensive experiments or simulations

Good news and bad news

- ☺ One-port model: first step towards designing realistic scheduling heuristics
- ☺ Perfectly divisible load simplifies task allocation
- ☺ Steady-state circumvents complexity of scheduling problems . . . while deriving efficient (often asympotically optimal) scheduling algorithms

- ☹ Need to acquire a good knowledge of the platform graph
- ☹ Need to run extensive experiments or simulations

Good news and bad news

- ☺ One-port model: first step towards designing realistic scheduling heuristics
- ☺ Perfectly divisible load simplifies task allocation
- ☺ Steady-state circumvents complexity of scheduling problems . . . while deriving efficient (often asympotically optimal) scheduling algorithms

- ☹ Need to acquire a good knowledge of the platform graph
- ☹ Need to run extensive experiments or simulations

Good news and bad news

- ☺ One-port model: first step towards designing realistic scheduling heuristics
- ☺ Perfectly divisible load simplifies task allocation
- ☺ Steady-state circumvents complexity of scheduling problems . . . while deriving efficient (often asympotically optimal) scheduling algorithms

- ☹ Need to acquire a good knowledge of the platform graph
- ☹ Need to run extensive experiments or simulations

Knowledge of the platform graph

- For regular problems, the *structure* of the task graph (nodes and edges) only depends upon the application, not upon the target platform
- Problems arise from *weights*, i.e. the estimation of execution and communication times
- Classical answer: *"use the past to predict the future"*
- Divide scheduling into phases, during which machine and network parameters are collected (with NWS)
  $\Rightarrow$ This information guides scheduling decisions for next phase
- Moving from heterogeneous clusters to computational grids causes further problems (even discovering the characteristics of the surrounding computing resources may prove a difficult task)

Knowledge of the platform graph

- For regular problems, the *structure* of the task graph (nodes and edges) only depends upon the application, not upon the target platform
- Problems arise from *weights*, i.e. the estimation of execution and communication times
- Classical answer: *"use the past to predict the future"*
- Divide scheduling into phases, during which machine and network parameters are collected (with NWS)
  $\Rightarrow$ This information guides scheduling decisions for next phase
- Moving from heterogeneous clusters to computational grids causes further problems (even discovering the characteristics of the surrounding computing resources may prove a difficult task)

Knowledge of the platform graph

- For regular problems, the *structure* of the task graph (nodes and edges) only depends upon the application, not upon the target platform
- Problems arise from *weights*, i.e. the estimation of execution and communication times
- Classical answer: *"use the past to predict the future"*
- Divide scheduling into phases, during which machine and network parameters are collected (with NWS)
  $\Rightarrow$ This information guides scheduling decisions for next phase
- Moving from heterogeneous clusters to computational grids causes further problems (even discovering the characteristics of the surrounding computing resources may prove a difficult task)

Knowledge of the platform graph

- For regular problems, the *structure* of the task graph (nodes and edges) only depends upon the application, not upon the target platform
- Problems arise from *weights*, i.e. the estimation of execution and communication times
- Classical answer: *"use the past to predict the future"*
- Divide scheduling into phases, during which machine and network parameters are collected (with NWS)
  $\Rightarrow$ This information guides scheduling decisions for next phase

- Moving from heterogeneous clusters to computational grids causes further problems (even discovering the characteristics of the surrounding computing resources may prove a difficult task)

Knowledge of the platform graph

- For regular problems, the *structure* of the task graph (nodes and edges) only depends upon the application, not upon the target platform
- Problems arise from *weights*, i.e. the estimation of execution and communication times
- Classical answer: *"use the past to predict the future"*
- Divide scheduling into phases, during which machine and network parameters are collected (with NWS)
  $\Rightarrow$ This information guides scheduling decisions for next phase

- Moving from heterogeneous clusters to computational grids causes further problems (even discovering the characteristics of the surrounding computing resources may prove a difficult task)

Experiments versus simulations

- Real experiments difficult to drive (genuine instability of non-dedicated platforms)
- Simulations ensure reproducibility of measured data
- Key issue: run simulations against a realistic environment
- *Trace-based scheduling*: record platform parameters today, and simulate the algorithms tomorrow, against recorded data
- Use SimGrid, an event-driven toolkit

Experiments versus simulations

- Real experiments difficult to drive (genuine instability of non-dedicated platforms)
- Simulations ensure reproducibility of measured data
- Key issue: run simulations against a realistic environment
- *Trace-based scheduling*: record platform parameters today, and simulate the algorithms tomorrow, against recorded data
- Use SimGrid, an event-driven toolkit

Experiments versus simulations

- Real experiments difficult to drive (genuine instability of non-dedicated platforms)
- Simulations ensure reproducibility of measured data
- Key issue: run simulations against a realistic environment
- *Trace-based scheduling*: record platform parameters today, and simulate the algorithms tomorrow, against recorded data
- Use SimGrid, an event-driven toolkit

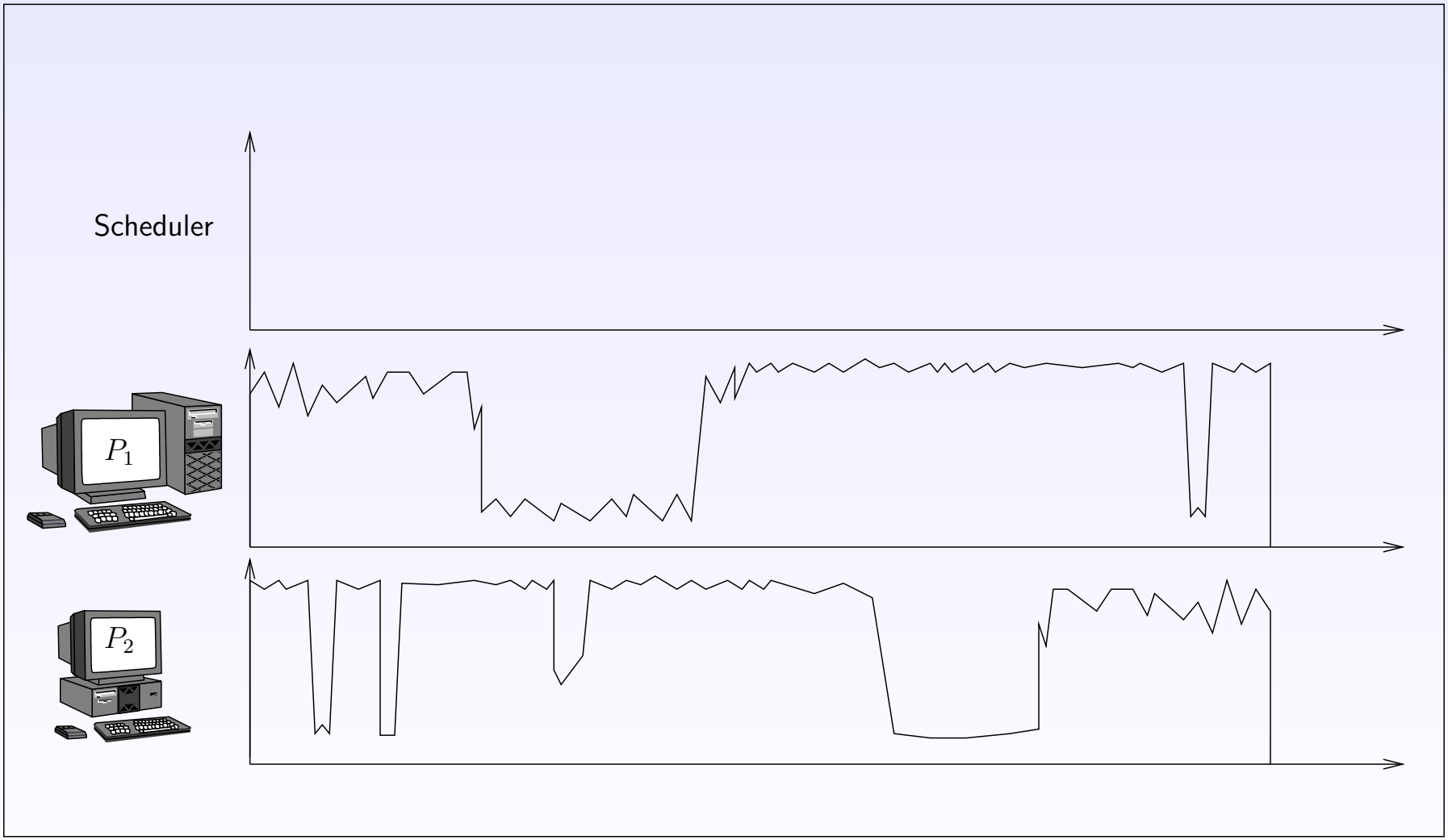
Experiments versus simulations

- Real experiments difficult to drive (genuine instability of non-dedicated platforms)
- Simulations ensure reproducibility of measured data
- Key issue: run simulations against a realistic environment
- *Trace-based scheduling*: record platform parameters today, and simulate the algorithms tomorrow, against recorded data
- Use SimGrid, an event-driven toolkit

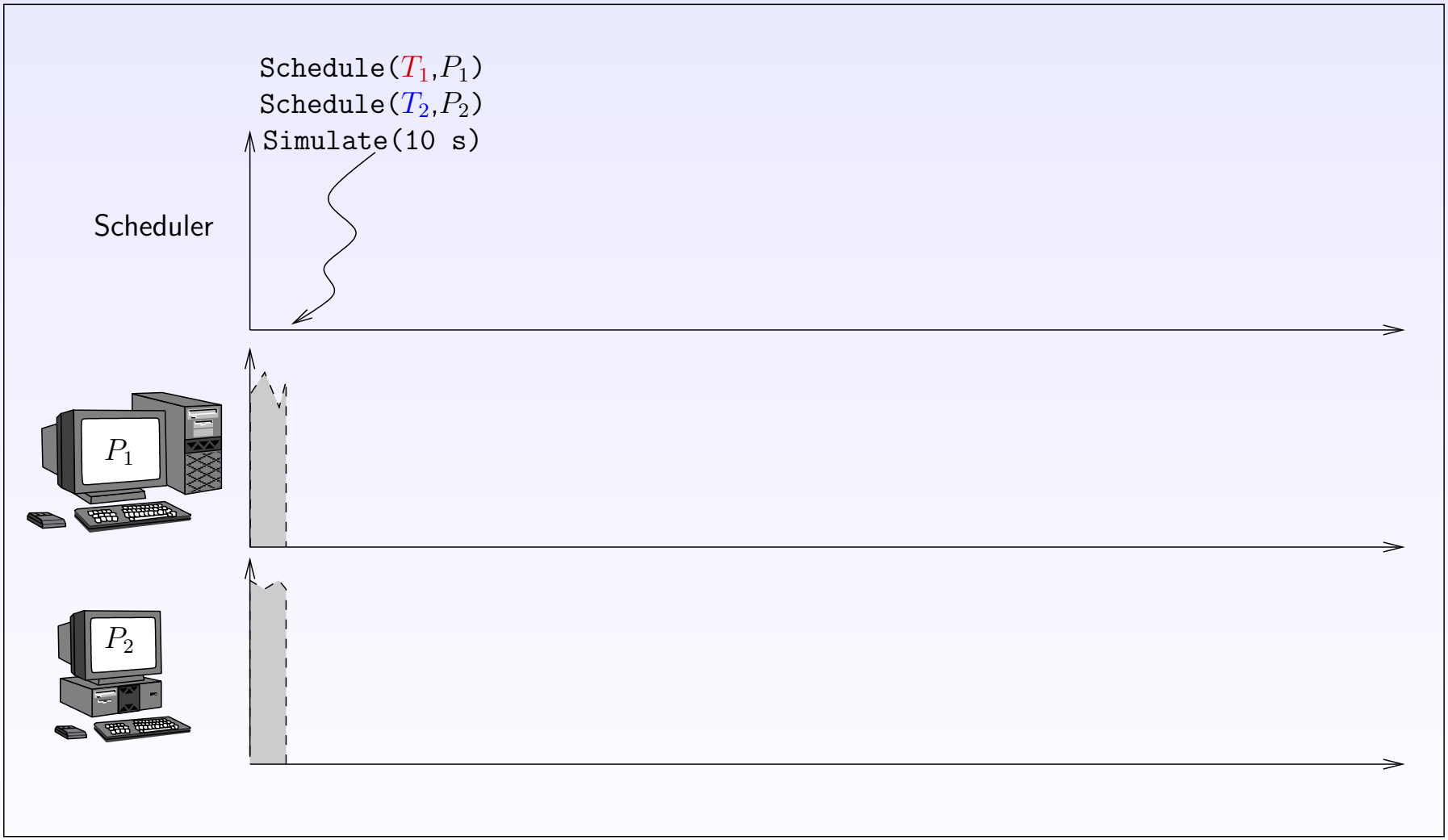
Experiments versus simulations

- Real experiments difficult to drive (genuine instability of non-dedicated platforms)
- Simulations ensure reproducibility of measured data
- Key issue: run simulations against a realistic environment
- *Trace-based scheduling*: record platform parameters today, and simulate the algorithms tomorrow, against recorded data
- Use SimGrid, an event-driven toolkit

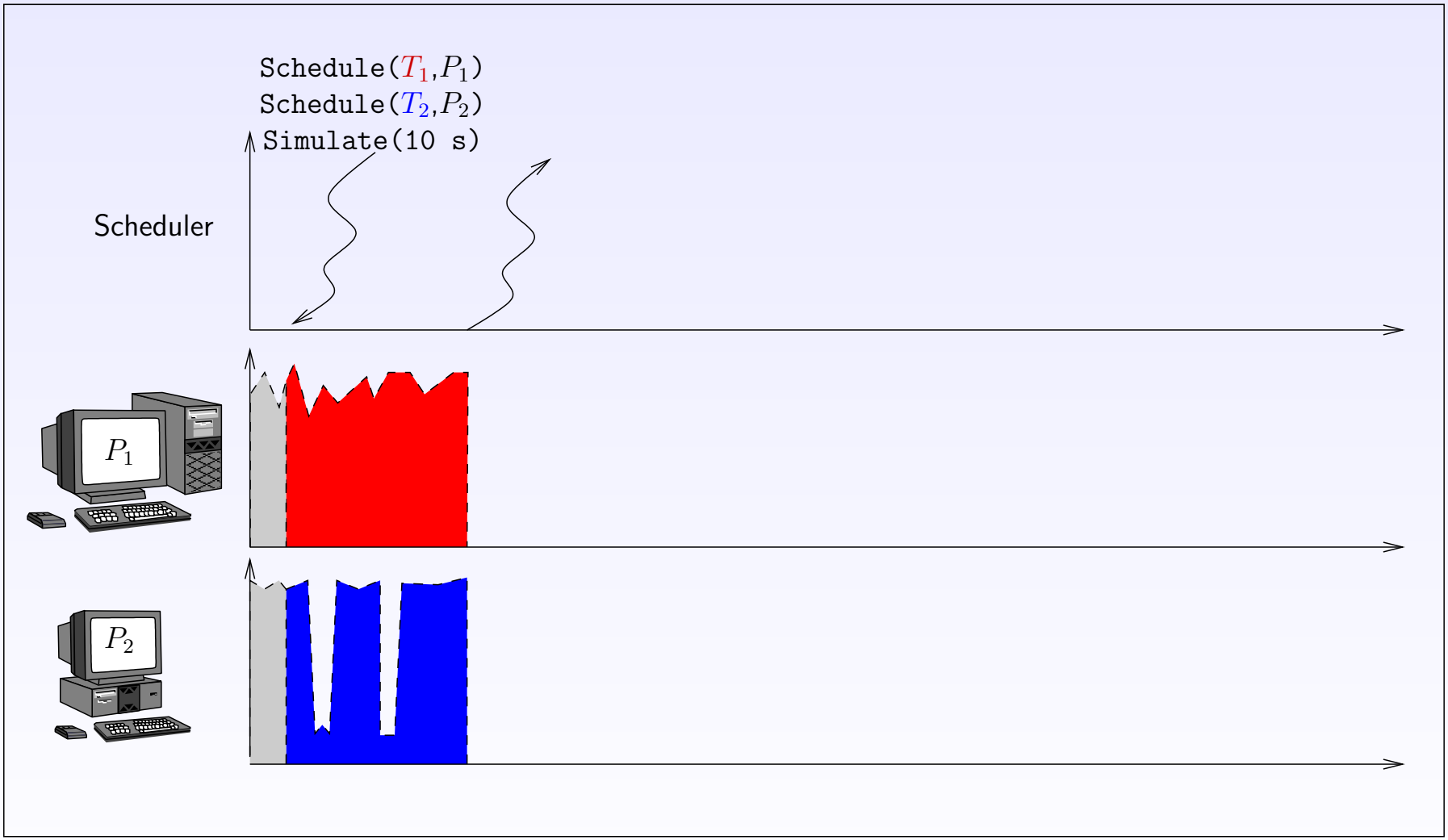
SimGrid traces



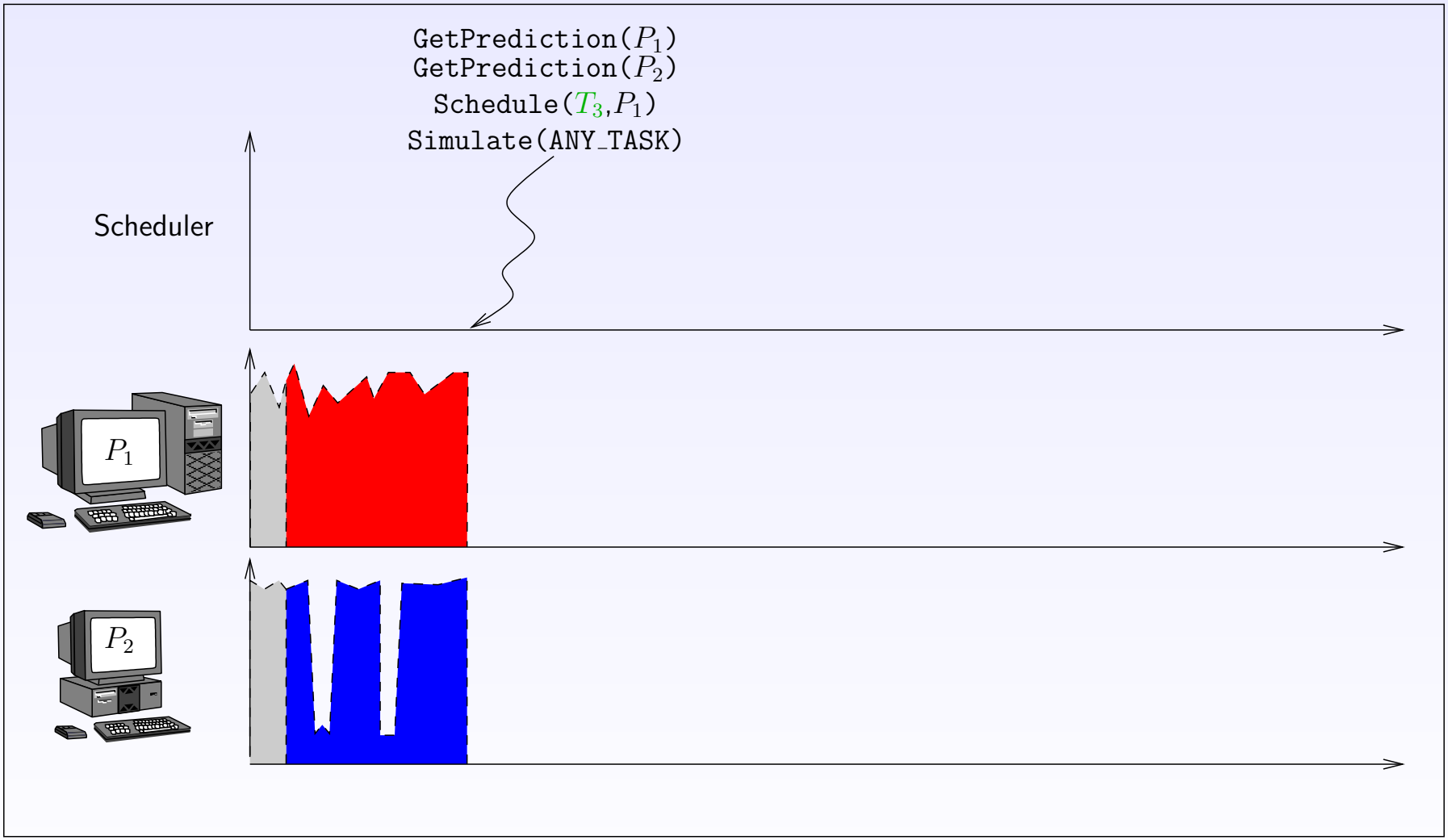
SimGrid traces



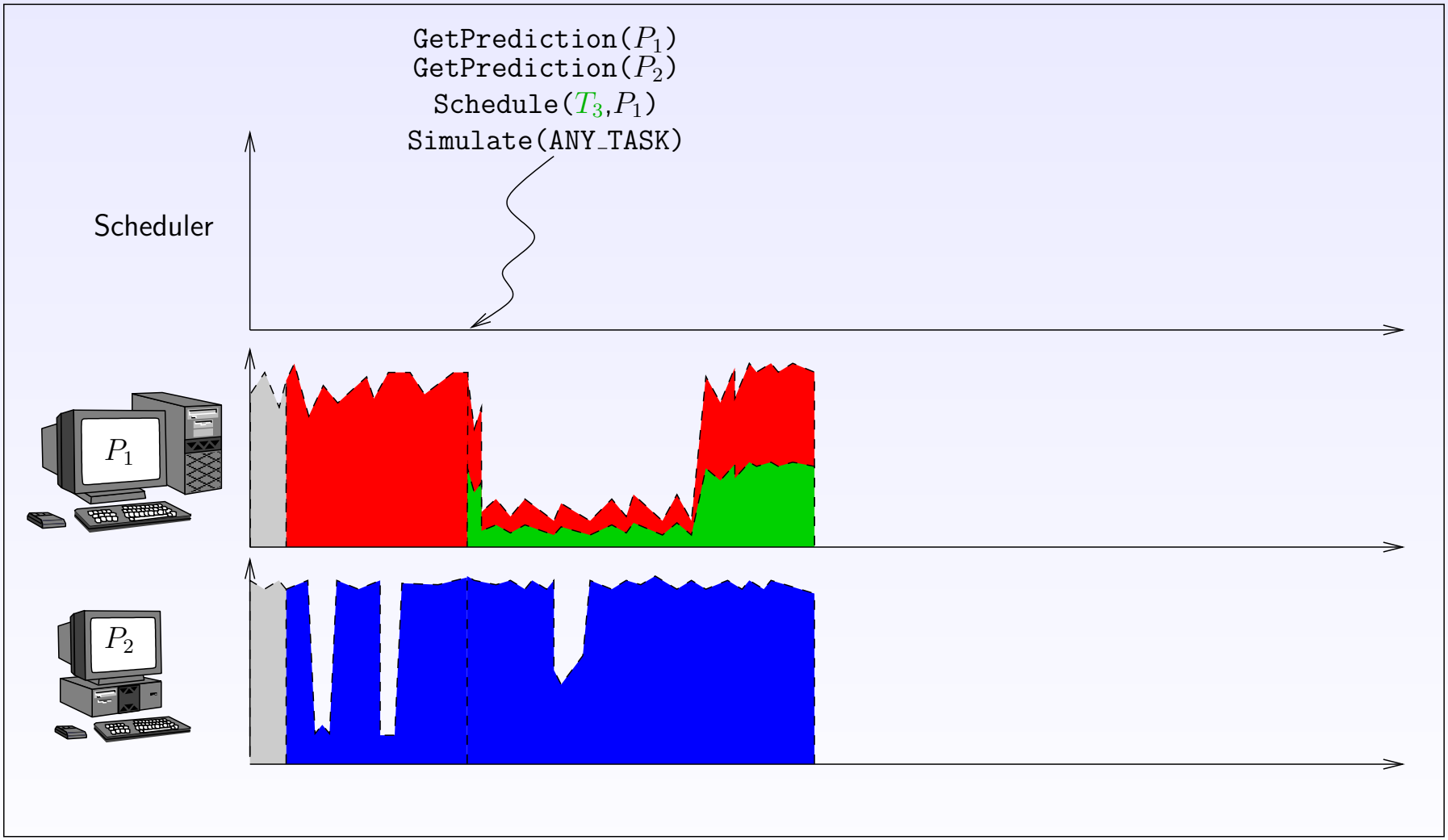
SimGrid traces



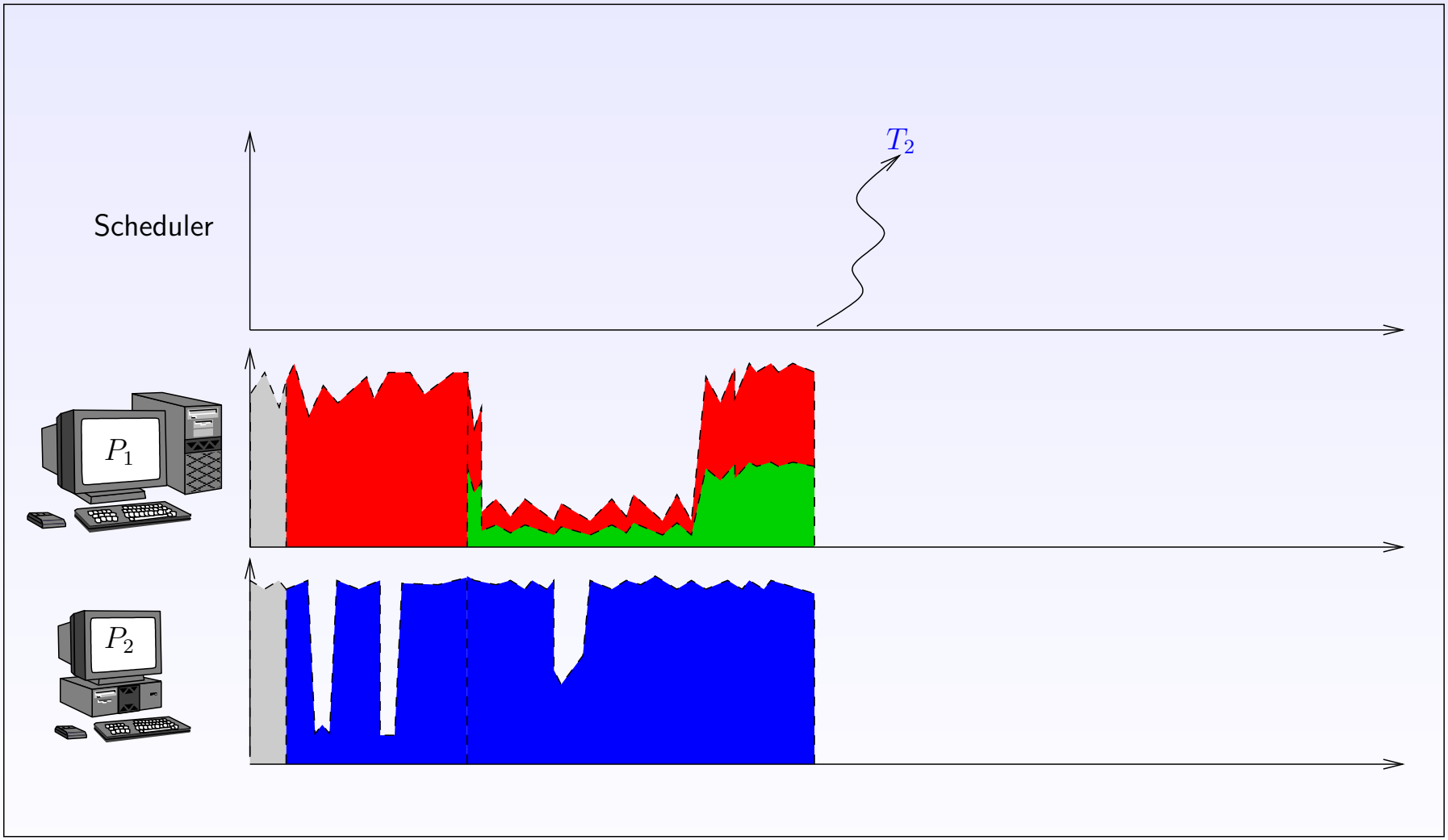
SimGrid traces



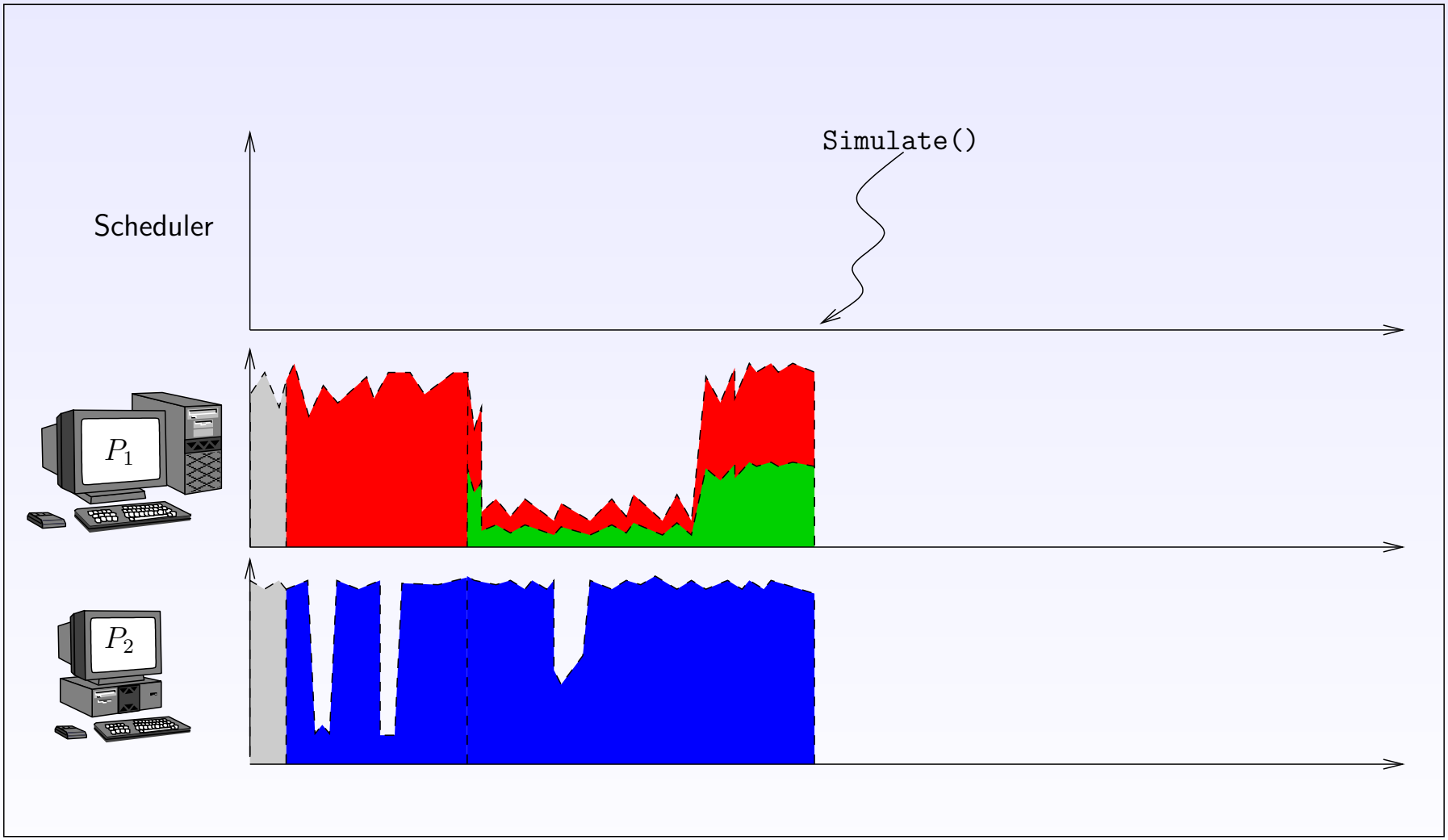
SimGrid traces



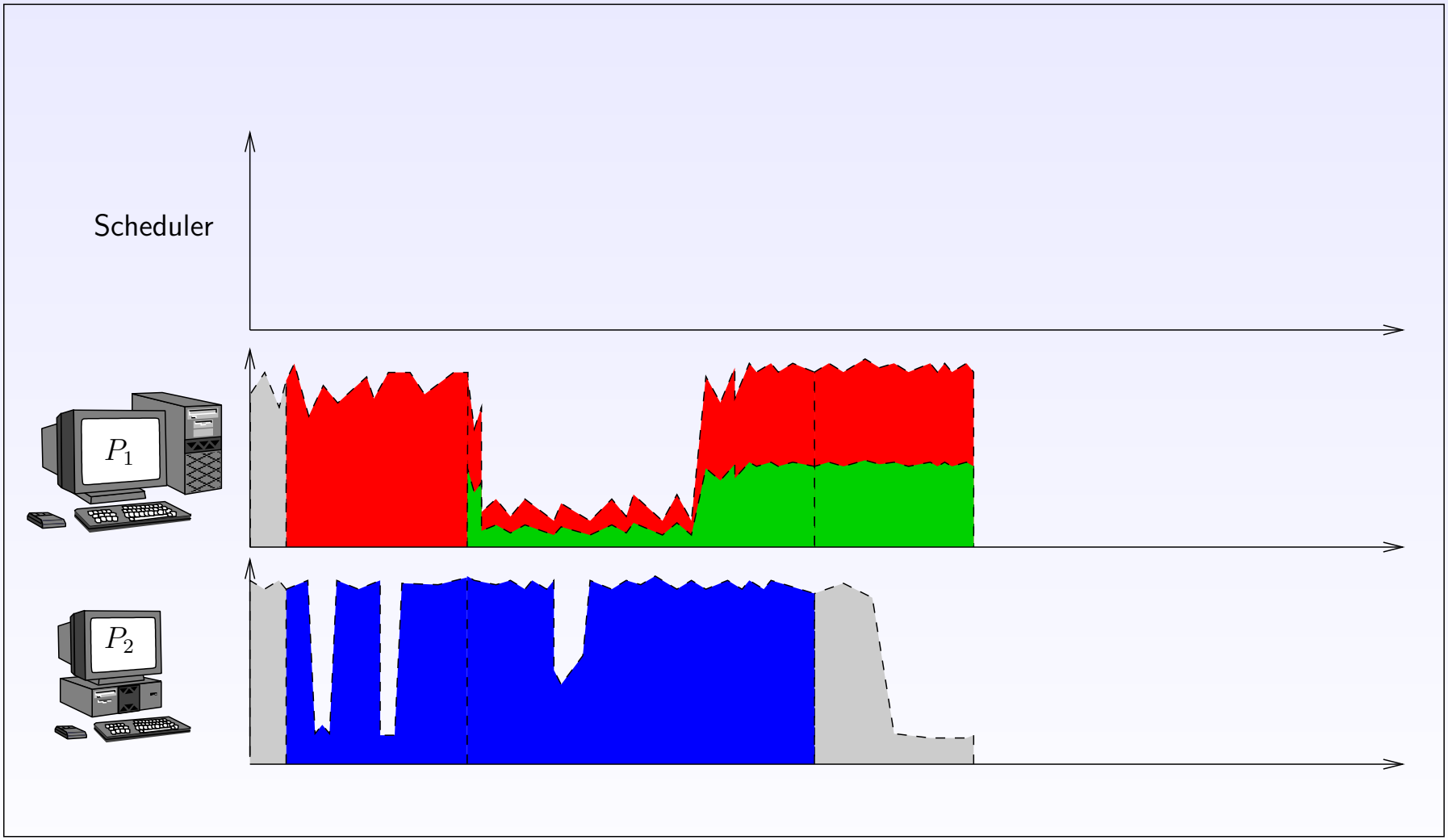
SimGrid traces



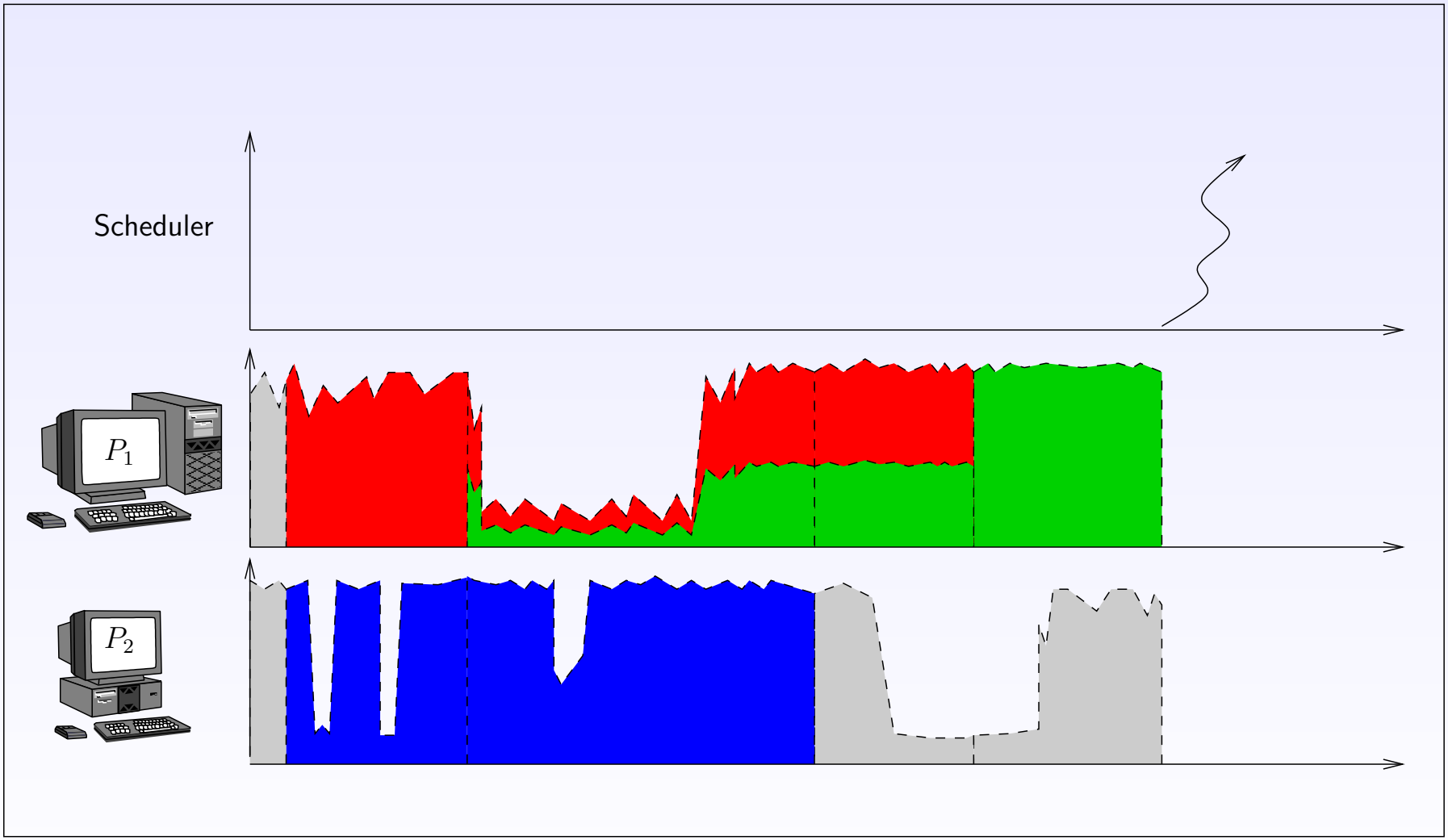
SimGrid traces



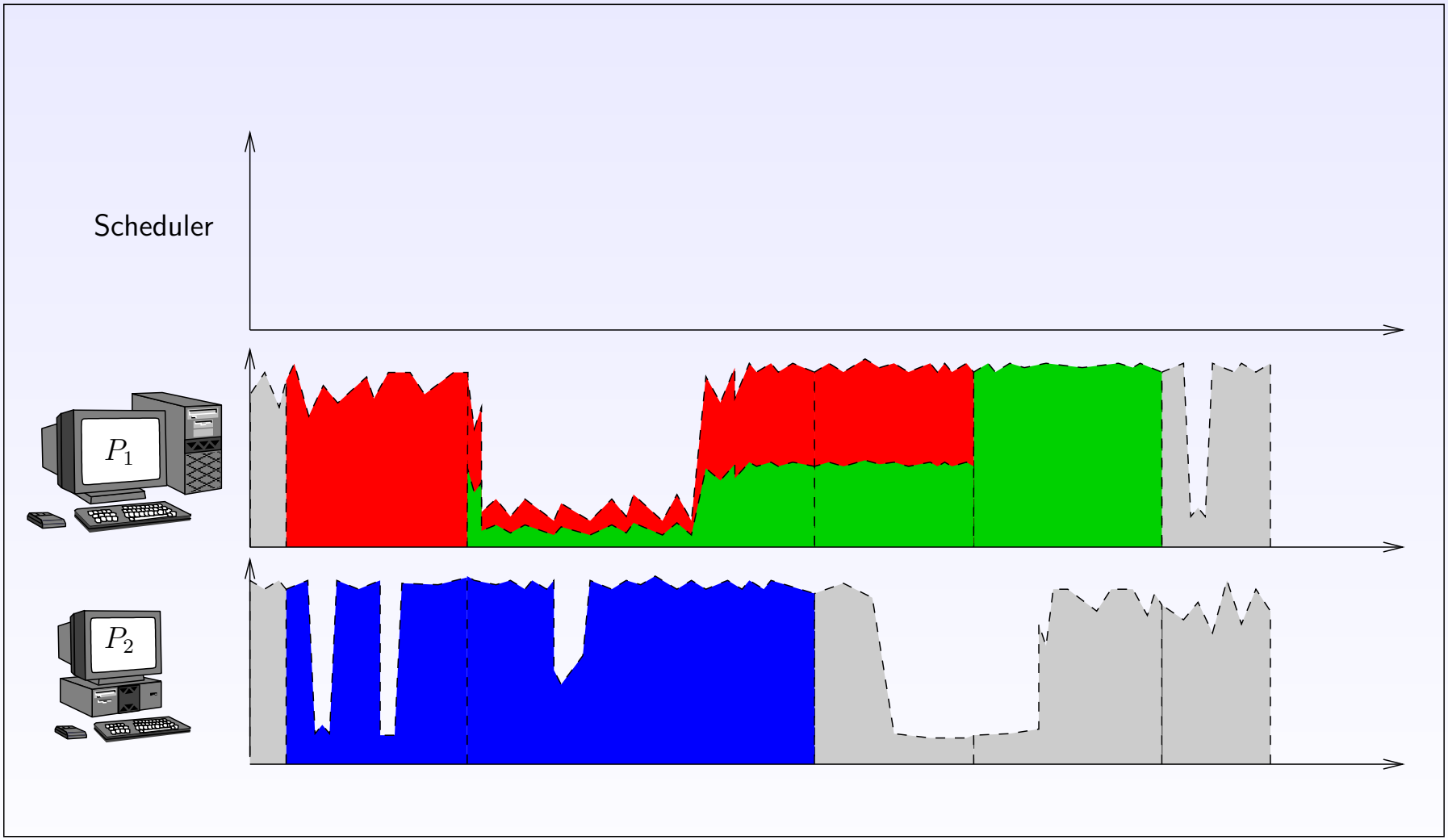
SimGrid traces



SimGrid traces



SimGrid traces

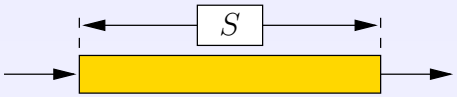


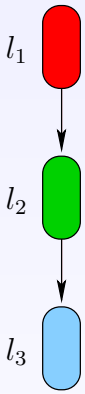
Store & Forward, WormHole, TCP

How to model a file transfer along a path?

Store & Forward, WormHole, TCP

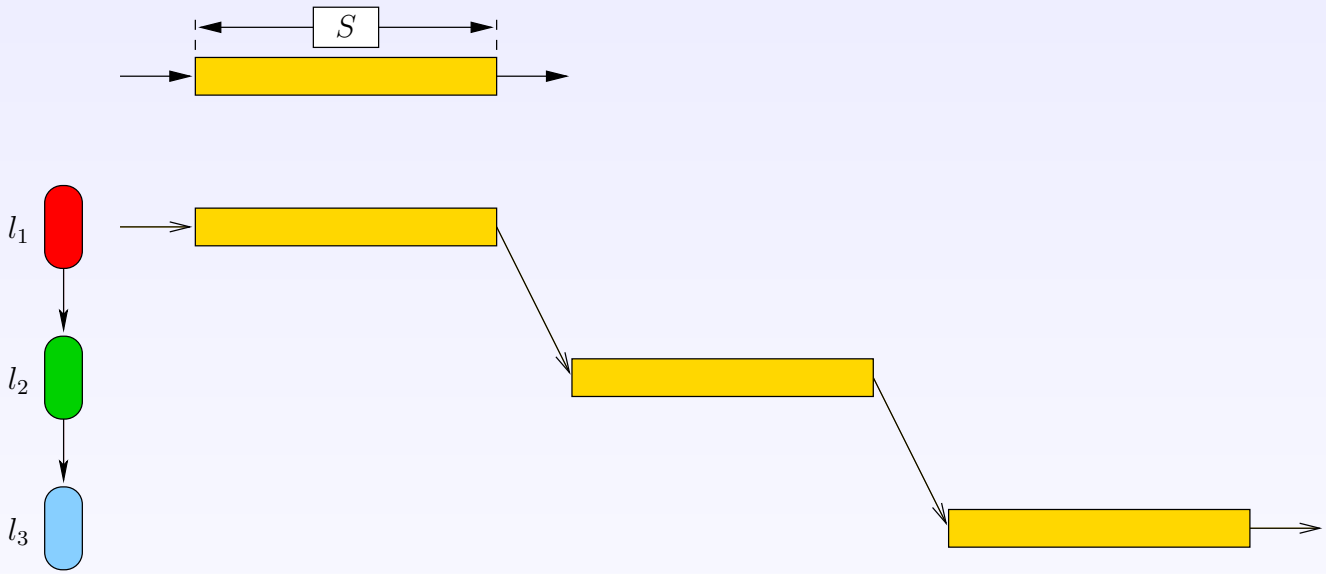
How to model a file transfer along a path?





Store & Forward, WormHole, TCP

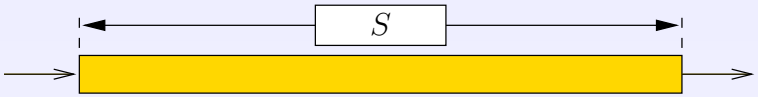
How to model a file transfer along a path?

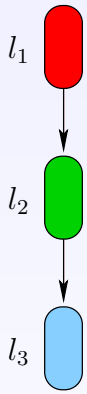


Store & Forward : bad model for contention

Store & Forward, WormHole, TCP

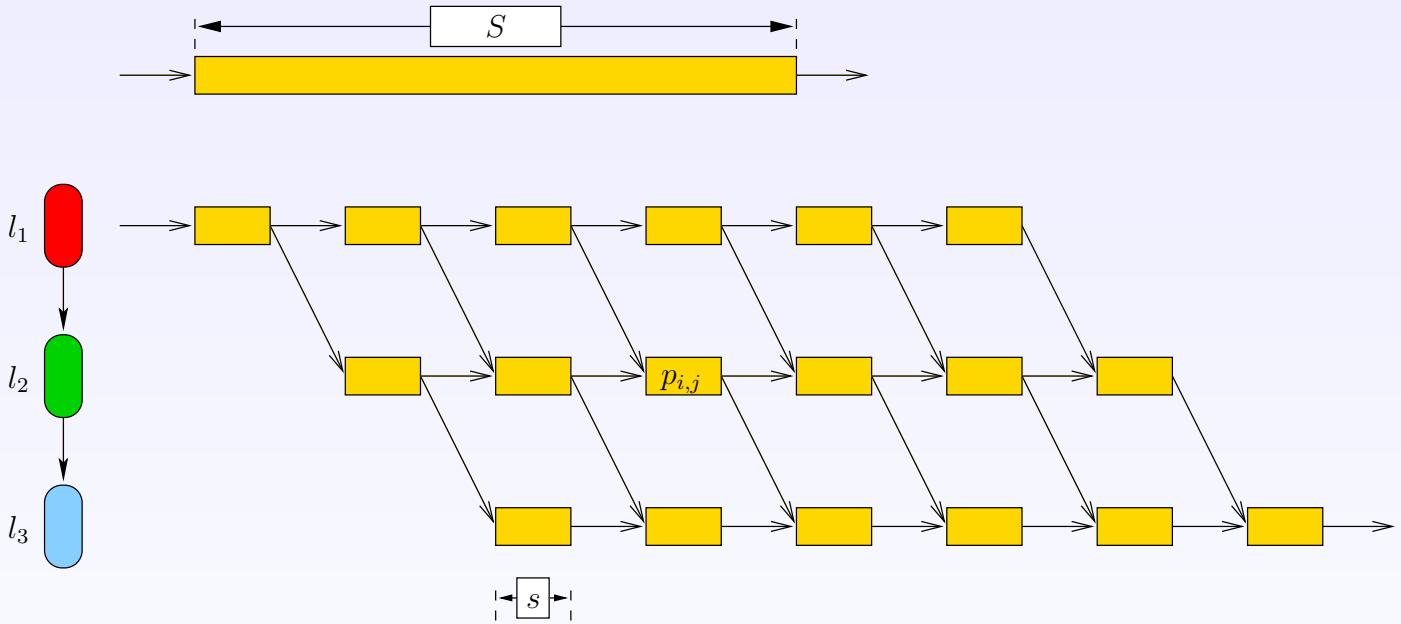
How to model a file transfer along a path?





Store & Forward, WormHole, TCP

How to model a file transfer along a path?



WormHole : computation intensive (packets), not that realistic

Store & Forward, WormHole, TCP

How to model a file transfer along a path?

$$\forall l \in \mathcal{L}, \quad \sum_{r \in \mathcal{R} \text{ s.t. } l \in r} \rho_r \leqslant c_l$$

Analytical model

Store & Forward, WormHole, TCP

How to model a file transfer along a path?

$$\forall l \in \mathcal{L}, \quad \sum_{r \in \mathcal{R} \text{ s.t. } l \in r} \rho_r \leqslant c_l$$

Max-Min Fairness maximize $\min_{r \in \mathcal{R}} \rho_r$

Store & Forward, WormHole, TCP

How to model a file transfer along a path?

$$\forall l \in \mathcal{L}, \quad \sum_{r \in \mathcal{R} \text{ s.t. } l \in r} \rho_r \leqslant c_l$$

Max-Min Fairness maximize $\min_{r \in \mathcal{R}} \rho_r$

Proportional Fairness maximize $\sum_{r \in \mathcal{R}} \rho_r \log(\rho_r)$

Store & Forward, WormHole, TCP

How to model a file transfer along a path?

$$\forall l \in \mathcal{L}, \quad \sum_{r \in \mathcal{R} \text{ s.t. } l \in r} \rho_r \leqslant c_l$$

Max-Min Fairness maximize $\min_{r \in \mathcal{R}} \rho_r$

Proportional Fairness maximize $\sum_{r \in \mathcal{R}} \rho_r \log(\rho_r)$

MCT minimization maximize $\min_{r \in \mathcal{R}} \frac{1}{\rho_r}$

Store & Forward, WormHole, TCP

How to model a file transfer along a path?

$$\forall l \in \mathcal{L}, \quad \sum_{r \in \mathcal{R} \text{ s.t. } l \in r} \rho_r \leqslant c_l$$

Max-Min Fairness maximize $\min_{r \in \mathcal{R}} \rho_r$

Proportional Fairness maximize $\sum_{r \in \mathcal{R}} \rho_r \log(\rho_r)$

MCT minimization maximize $\min_{r \in \mathcal{R}} \frac{1}{\rho_r}$

TCP behavior Close to max-min.

In SimGrid: max-min + bound by $1/RTT$

Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open i TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \dfrac{bw(1)}{1 + (i-1).\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN

Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open i TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow$ $bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \frac{bw(1)}{1 + (i-1).\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN

Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open i TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \dfrac{bw(1)}{1 + (i-1).\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN

Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open i TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow$ $bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \dfrac{bw(1)}{1 + (i-1).\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN

Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open i TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \dfrac{bw(1)}{1 + (i-1).\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN

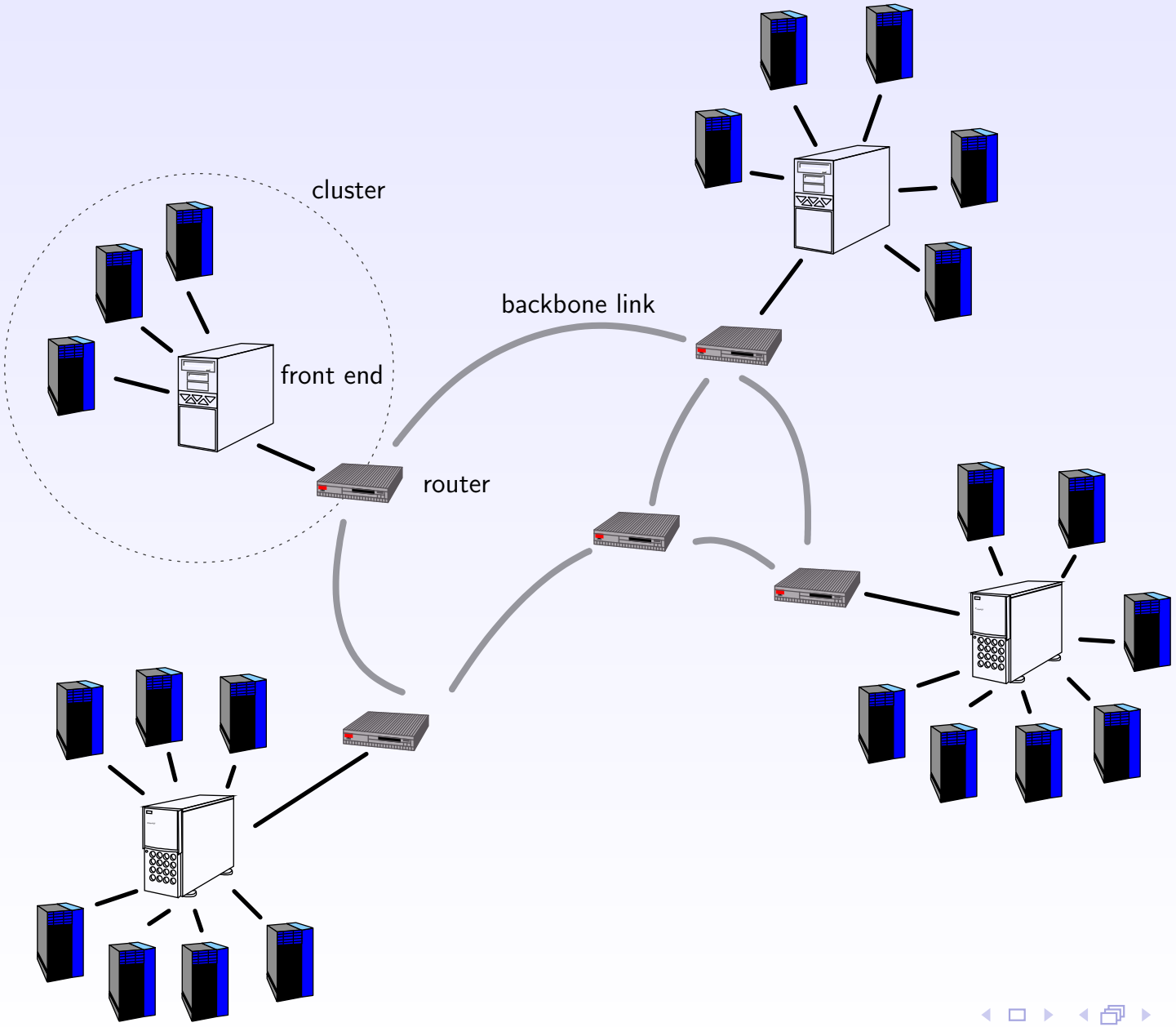
Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open i TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \dfrac{bw(1)}{1 + (i-1).\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN

Bandwidth sharing

- Traditional assumption: Fair Sharing
- Open i TCP connections, receive $bw(i)$ bandwidth per connection
- $bw(i) = bw(1)/i$ on a LAN
- Experimental evidence $\rightarrow$ $bw(i) = bw(1)$ on a WAN
- Backbone links have so many connections that interference among a few selected connections is negligible
- Better model: $bw(i) = \dfrac{bw(1)}{1 + (i-1).\gamma}$
- $\gamma = 1$ for a perfect LAN, $\gamma = 0$ for a perfect WAN

Sample large-scale platform



What topology?

- Generated (GT-ITM, BRITE, etc.) or obtained from monitoring?
  - Very complex (Layer 2 information)
  - Not clear that a scheduling algorithm could exploit/know all that information
- Need a simple model that is
  - More accurate than traditional models (e.g., LAN links, fully-connected)
  - Still amenable to analysis

What topology?

- Generated (GT-ITM, BRITE, etc.) or obtained from monitoring?
  - Very complex (Layer 2 information)
  - Not clear that a scheduling algorithm could exploit/know all that information
- Need a simple model that is
  - More accurate than traditional models (e.g., LAN links, fully-connected)
  - Still amenable to analysis

What topology?

- Generated (GT-ITM, BRITE, etc.) or obtained from monitoring?
  - Very complex (Layer 2 information)
  - Not clear that a scheduling algorithm could exploit/know all that information
- Need a simple model that is
  - More accurate than traditional models (e.g., LAN links, fully-connected)
  - Still amenable to analysis

What topology?

- Generated (GT-ITM, BRITE, etc.) or obtained from monitoring?
  - Very complex (Layer 2 information)
  - Not clear that a scheduling algorithm could exploit/know all that information
- Need a simple model that is
  - More accurate than traditional models (e.g., LAN links, fully-connected)
  - Still amenable to analysis

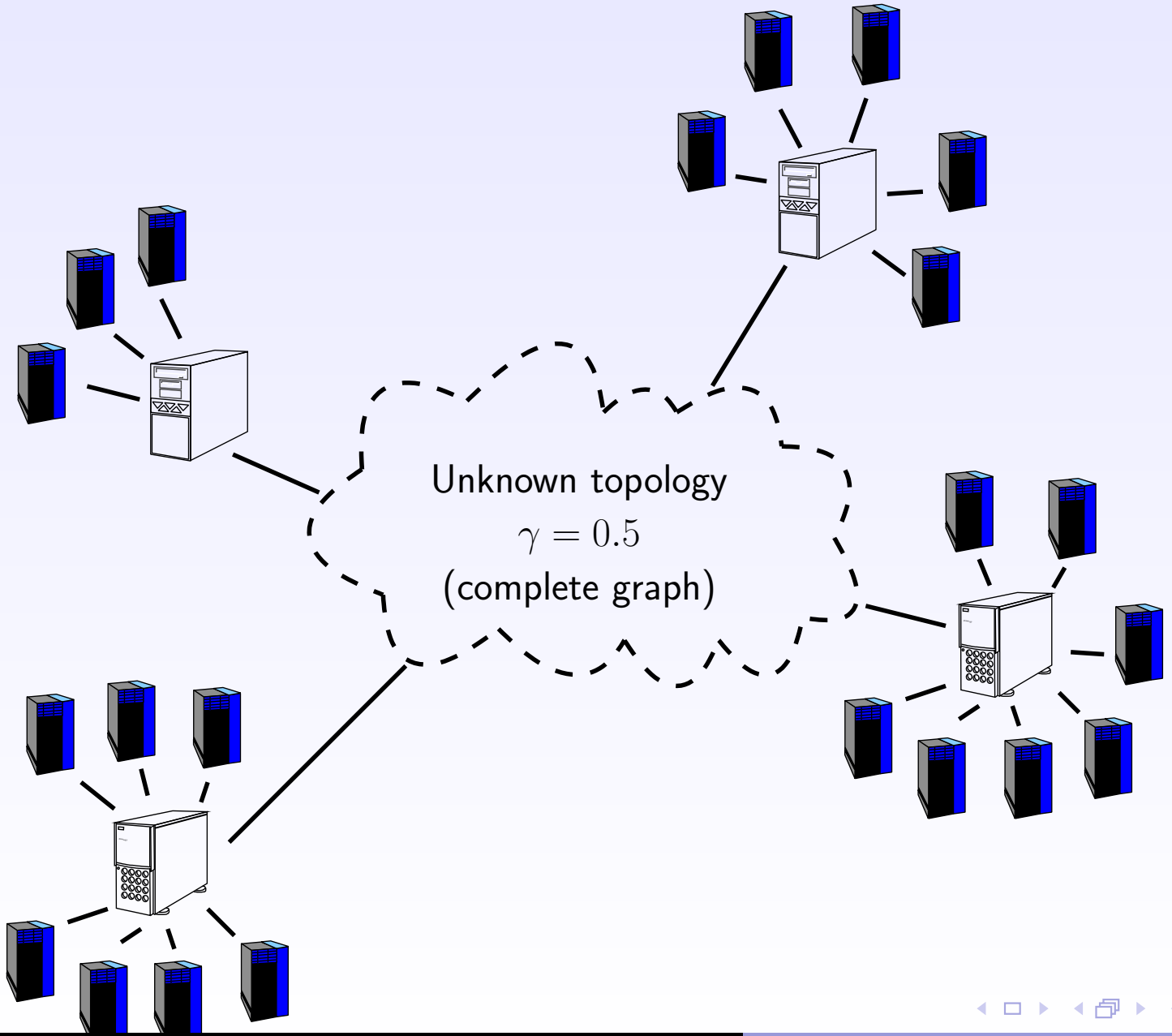
What topology?

- Generated (GT-ITM, BRITE, etc.) or obtained from monitoring?
  - Very complex (Layer 2 information)
  - Not clear that a scheduling algorithm could exploit/know all that information
- Need a simple model that is
  - More accurate than traditional models (e.g., LAN links, fully-connected)
  - Still amenable to analysis

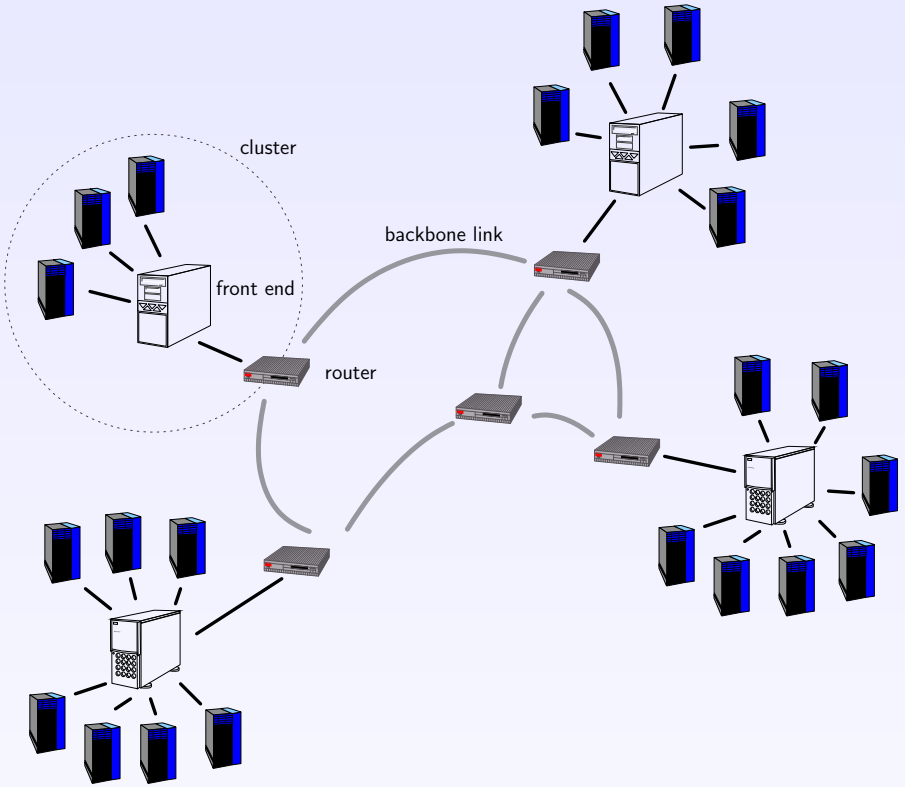
What topology?

- Generated (GT-ITM, BRITE, etc.) or obtained from monitoring?
  - Very complex (Layer 2 information)
  - Not clear that a scheduling algorithm could exploit/know all that information
- Need a simple model that is
  - More accurate than traditional models (e.g., LAN links, fully-connected)
  - Still amenable to analysis

What topology? (cont'd)



What topology? (cont'd)



Hierarchy + BW sharing, but assume knowledge of

- Routing
- Backbone bandwidths
- CPU speeds

Bibliography

- NWS:
  The network weather service: a distributed resource performance forecasting service for metacomputing, R. Wolski, N.T. Spring and J. Hayes, Future Generation Computer Systems 15, 10 (1999), 757-768
- SimGrid:
  Scheduling distributed applications: the SimGrid simulation framework, A. Legrand, L. Marchal, and H. Casanova, 3rd IEEE CCGrid (2003), 138-145
- Bandwidth sharing:
  Bandwidth sharing: objectives and algorithms, L. Massoulié and J. Roberts, IEEE/ACM Trans. Networking 10, 3 (2002), 320-328

Outline

Outline

Motivation

- Multiple divisible load applications with priorities
- Realistic model for large scale (grid) platforms: hierarchy + bandwidth sharing
- Steady-state scheduling

Motivation

- Multiple divisible load applications with priorities
- Realistic model for large scale (grid) platforms: hierarchy + bandwidth sharing
- Steady-state scheduling

Motivation

- Multiple divisible load applications with priorities
- Realistic model for large scale (grid) platforms: hierarchy + bandwidth sharing
- Steady-state scheduling

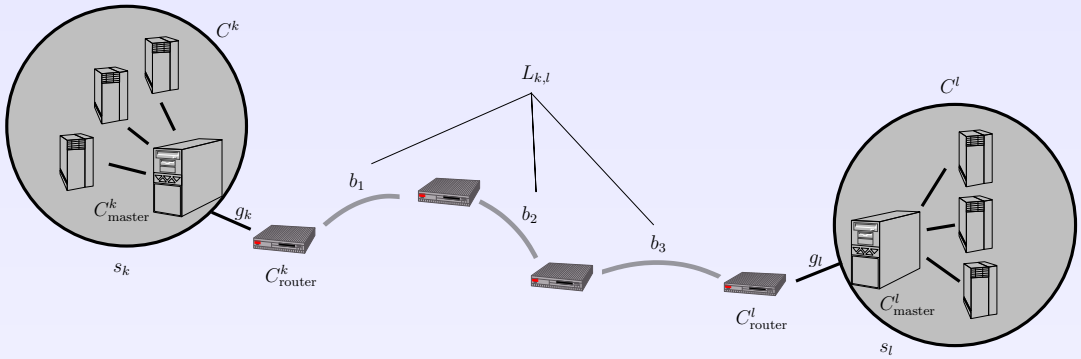
Scheduling multiple divisible load applications

- Large-scale platforms not likely to be exploited in dedicated mode/single application
- Investigate scenarios in which *multiple* divisible loads applications are simultaneously executed on the platform
  $\Rightarrow$ competition for CPU and network resources

Scheduling multiple divisible load applications

- Large-scale platforms not likely to be exploited in dedicated mode/single application
- Investigate scenarios in which *multiple* divisible loads applications are simultaneously executed on the platform
  $\Rightarrow$ competition for CPU and network resources

Notations



- Set $\mathcal{R}$ of routers and $\mathcal{B}$ of backbone links l_i
- bw(l_i) bandwidth available for a new connection
- max-connect(l_i) max. number of connections that can be opened
- K clusters C^k, $1 \leqslant k \leqslant K$:
  - C^k_{master} front-end processor
  - C^k_{router}, one of the routers in $\mathcal{R}$
  - s_k cumulated speed of C^k
  - g_k bandwidth of the LAN link ($\gamma = 1$) from C^k_{master} to C^k_{router}
- Fixed routing: path $L_{k,l}$ of backbones from C^k_{router} to C^l_{router}

Divisible applications

- One divisible load application A_k per cluster C^k:
  - w_k computation size (flops) of elemental chunk
  - δ_k communication size (bytes) of elemental chunk
- $\alpha_{k,l}$: fraction of A_k executed by C^l (per time unit)
  $\alpha_k = \sum_l \alpha_{k,l}$: total work executed for application A_k
- Need $\frac{\alpha_{k,l} \cdot w_k}{s_l}$ time-units to process $\alpha_{k,l}$ chunks of A_k on C^l
- Need $\frac{\alpha_{k,l} \cdot \delta_k}{g_{k,l}}$ time-units to route one chunk of A_k from C^k_{router} to C^l_{router} (along one connection):
  $\rightarrow g_{k,l} = \min_{l_i \in L_{k,l}} \{\text{bw}(l_i)\}$
- $\beta_{k,l}$ number of connections from C^k to C^l

Divisible applications

- One divisible load application A_k per cluster C^k:
  - w_k computation size (flops) of elemental chunk
  - δ_k communication size (bytes) of elemental chunk
- $\alpha_{k,l}$: fraction of A_k executed by C^l (per time unit)
  $\alpha_k = \sum_l \alpha_{k,l}$: total work executed for application A_k
- Need $\frac{\alpha_{k,l} \cdot w_k}{s_l}$ time-units to process $\alpha_{k,l}$ chunks of A_k on C^l
- Need $\frac{\alpha_{k,l} \cdot \delta_k}{g_{k,l}}$ time-units to route one chunk of A_k from C^k_{router} to C^l_{router} (along one connection):
  $\rightarrow g_{k,l} = \min_{l_i \in L_{k,l}} \{\text{bw}(l_i)\}$
- $\beta_{k,l}$ number of connections from C^k to C^l

Divisible applications

- One divisible load application A_k per cluster C^k:
  - w_k computation size (flops) of elemental chunk
  - δ_k communication size (bytes) of elemental chunk
- $\alpha_{k,l}$: fraction of A_k executed by C^l (per time unit)
  $\alpha_k = \sum_l \alpha_{k,l}$: total work executed for application A_k
- Need $\frac{\alpha_{k,l}.w_k}{s_l}$ time-units to process $\alpha_{k,l}$ chunks of A_k on C^l
- Need $\frac{\alpha_{k,l}.\delta_k}{g_{k,l}}$ time-units to route one chunk of A_k from C^k_{router} to C^l_{router} (along one connection):
  $\rightarrow g_{k,l} = \min_{l_i \in L_{k,l}} \{\text{bw}(l_i)\}$
- $\beta_{k,l}$ number of connections from C^k to C^l

Divisible applications

- One divisible load application A_k per cluster C^k:
  - w_k computation size (flops) of elemental chunk
  - δ_k communication size (bytes) of elemental chunk
- $\alpha_{k,l}$: fraction of A_k executed by C^l (per time unit)
  $\alpha_k = \sum_l \alpha_{k,l}$: total work executed for application A_k
- Need $\frac{\alpha_{k,l}.w_k}{s_l}$ time-units to process $\alpha_{k,l}$ chunks of A_k on C^l
- Need $\frac{\alpha_{k,l}.\delta_k}{g_{k,l}}$ time-units to route one chunk of A_k from C^k_{router} to C^l_{router} (along one connection):
  $\rightarrow g_{k,l} = \min_{l_i \in L_{k,l}} \{\text{bw}(l_i)\}$
- $\beta_{k,l}$ number of connections from C^k to C^l

Divisible applications

- One divisible load application A_k per cluster C^k:
  - w_k computation size (flops) of elemental chunk
  - δ_k communication size (bytes) of elemental chunk
- $\alpha_{k,l}$: fraction of A_k executed by C^l (per time unit)
  $\alpha_k = \sum_l \alpha_{k,l}$: total work executed for application A_k
- Need $\frac{\alpha_{k,l}.w_k}{s_l}$ time-units to process $\alpha_{k,l}$ chunks of A_k on C^l
- Need $\frac{\alpha_{k,l}.\delta_k}{g_{k,l}}$ time-units to route one chunk of A_k from C^k_{router} to C^l_{router} (along one connection):
  $\rightarrow g_{k,l} = \min_{l_i \in L_{k,l}} \{\text{bw}(l_i)\}$
- $\beta_{k,l}$ number of connections from C^k to C^l

Steady-state

$$\forall C^k, \quad \sum_l \alpha_{l,k}.w_l \quad \leqslant \quad s_k \tag{1}$$

$$\forall C^k, \underbrace{\sum_{l \neq k} \alpha_{k,l}.\delta_k}_{\text{(outgoing data)}} + \underbrace{\sum_{j \neq k} \alpha_{j,k}.\delta_j}_{\text{(incoming data)}} \quad \leqslant \quad g_k \tag{2}$$

$$\forall l_i, \sum_{\{k,l\},\ l_i \in L_{k,l}} \beta_{k,l} \quad \leqslant \quad \text{max-connect}(l_i) \tag{3}$$

$$\forall (C^k, C^l), \alpha_{k,l}.\delta_k \quad \leqslant \quad \beta_{k,l} \times g_{k,l} \tag{4}$$

$$\text{Maximize} \quad \min_k \left\{ \frac{\alpha_k}{\pi_k} \right\}. \tag{5}$$

Steady-state

$$\forall C^k, \quad \sum_l \alpha_{l,k}.w_l \quad \leqslant \quad s_k \tag{1}$$

$$\forall C^k, \underbrace{\sum_{l \neq k} \alpha_{k,l}.\delta_k}_{\text{(outgoing data)}} + \underbrace{\sum_{j \neq k} \alpha_{j,k}.\delta_j}_{\text{(incoming data)}} \quad \leqslant \quad g_k \tag{2}$$

$$\forall l_i, \sum_{\{k,l\},\ l_i \in L_{k,l}} \beta_{k,l} \quad \leqslant \quad \text{max-connect}(l_i) \tag{3}$$

$$\forall (C^k, C^l), \alpha_{k,l}.\delta_k \quad \leqslant \quad \beta_{k,l} \times g_{k,l} \tag{4}$$

$$\text{Maximize} \quad \min_k \left\{ \frac{\alpha_k}{\pi_k} \right\}. \tag{5}$$

Steady-state

$$\forall C^k, \quad \sum_l \alpha_{l,k}.w_l \quad \leqslant \quad s_k \tag{1}$$

$$\forall C^k, \underbrace{\sum_{l \neq k} \alpha_{k,l}.\delta_k}_{\text{(outgoing data)}} + \underbrace{\sum_{j \neq k} \alpha_{j,k}.\delta_j}_{\text{(incoming data)}} \quad \leqslant \quad g_k \tag{2}$$

$$\forall l_i, \sum_{\{k,l\},\ l_i \in L_{k,l}} \beta_{k,l} \quad \leqslant \quad \text{max-connect}(l_i) \tag{3}$$

$$\forall (C^k, C^l), \alpha_{k,l}.\delta_k \quad \leqslant \quad \beta_{k,l} \times g_{k,l} \tag{4}$$

$$\text{Maximize} \quad \min_k \left\{ \frac{\alpha_k}{\pi_k} \right\}. \tag{5}$$

Steady-state

$$\forall C^k, \quad \sum_l \alpha_{l,k}.w_l \quad \leqslant \quad s_k \tag{1}$$

$$\forall C^k, \underbrace{\sum_{l \neq k} \alpha_{k,l}.\delta_k}_{\text{(outgoing data)}} + \underbrace{\sum_{j \neq k} \alpha_{j,k}.\delta_j}_{\text{(incoming data)}} \quad \leqslant \quad g_k \tag{2}$$

$$\forall l_i, \sum_{\{k,l\},\ l_i \in L_{k,l}} \beta_{k,l} \quad \leqslant \quad \text{max-connect}(l_i) \tag{3}$$

$$\forall (C^k, C^l), \alpha_{k,l}.\delta_k \quad \leqslant \quad \beta_{k,l} \times g_{k,l} \tag{4}$$

$$\text{Maximize} \quad \min_k \left\{ \frac{\alpha_k}{\pi_k} \right\}. \tag{5}$$

Steady-state

$$\forall C^k, \quad \sum_l \alpha_{l,k}.w_l \quad \leqslant \quad s_k \tag{1}$$

$$\forall C^k, \underbrace{\sum_{l \neq k} \alpha_{k,l}.\delta_k}_{\text{(outgoing data)}} + \underbrace{\sum_{j \neq k} \alpha_{j,k}.\delta_j}_{\text{(incoming data)}} \quad \leqslant \quad g_k \tag{2}$$

$$\forall l_i, \sum_{\{k,l\},\ l_i \in L_{k,l}} \beta_{k,l} \quad \leqslant \quad \text{max-connect}(l_i) \tag{3}$$

$$\forall (C^k, C^l), \alpha_{k,l}.\delta_k \quad \leqslant \quad \beta_{k,l} \times g_{k,l} \tag{4}$$

$$\text{MAXIMIZE} \quad \min_k \left\{ \frac{\alpha_k}{\pi_k} \right\}. \tag{5}$$

Linear program

$$\text{MAXIMIZE } \min_k \left\{ \frac{\alpha_k}{\pi_k} \right\},$$

UNDER THE CONSTRAINTS

$$\begin{cases} \text{(6a)} & \forall C^k, \quad \sum_l \alpha_{k,l} = \alpha_k \\ \text{(6b)} & \forall C^k, \quad \sum_l \alpha_{l,k}.w_l \leqslant s_k \\ \text{(6c)} & \forall C^k, \quad \sum_{l \neq k} \alpha_{k,l}.\delta_k + \sum_{j \neq k} \alpha_{j,k}.\delta_j \leqslant g_k \\ \text{(6d)} & \forall l_i, \quad \sum_{l_i \in L_{k,l}} \beta_{k,l} \leqslant \text{max-connect}(l_i) \\ \text{(6e)} & \forall k,l, \quad \alpha_{k,l}.\delta_k \leqslant \beta_{k,l} \times g_{k,l} \\ \text{(6f)} & \forall k,l, \quad \alpha_{k,l} \geqslant 0 \\ \text{(6g)} & \forall k,l, \quad \beta_{k,l} \in \mathbb{N} \end{cases} \tag{6}$$

Technicalities

- ☺ Reconstructing a periodic schedule
- ☹ NP-completeness of optimization problem

Technicalities

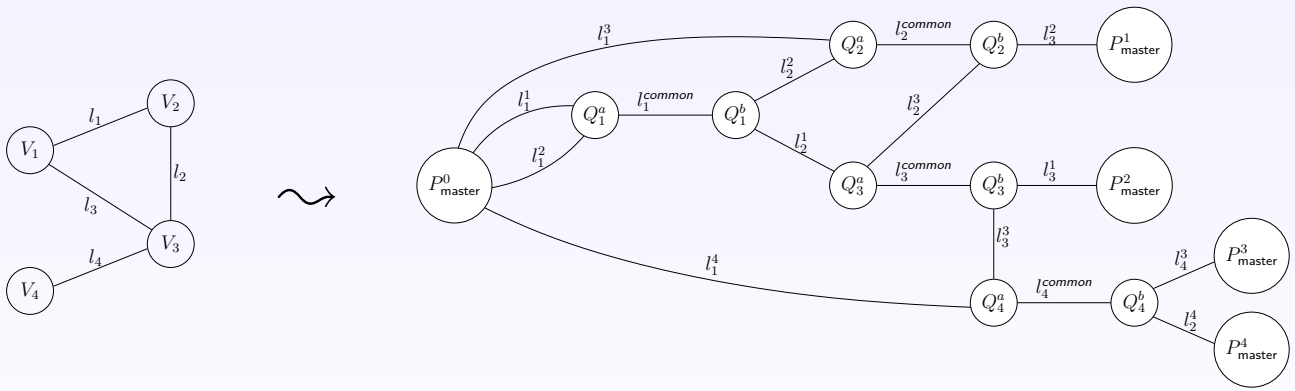
- ☺ Reconstructing a periodic schedule
- ☹ NP-completeness of optimization problem

Reconstructing a periodic schedule

- Write $\alpha_{k,l} = \frac{u_{k,l}}{v_{k,l}}$, with $u_{k,l}$ and $v_{k,l}$ are relatively prime:
  $\Rightarrow$ **Period** of the schedule is $T_p = \text{lcm}_{k,l}(v_{k,l})$
- In steady-state, during each period of length T_p:
  - C^k computes $\alpha_{l,k} \cdot T_p$ load units of application A_l
    - If $k \neq l$, corresponding data has been received during previous period
    - All computations are executed in any order
  - C^k sends $\alpha_{k,l} \cdot \delta_k \cdot T_p$ load units of application A_k, to be processed by C^l during next period
    - Similarly for receptions
    - All communications share serial link, but bandwidth is not exceeded.
- First and last period are different
- Periodic schedule described **in compact form**:
  $\rightarrow$ polynomial number of intervals during which each processor is assigned a given load for a prescribed application.

NP-Completeness

- Mixed linear program ⇒ no polynomial algorithm *a priori*
- Close to *multicommodity flow* problem (NP), but without prescribed location for each work
- Reduction from the MAXIMUM-INDEPENDENT-SET problem



Outline

Greedy Heuristic – Intuition

- At each step:
  (i) select an application A_k
  (ii) determine which cluster C^l executes the work
  (iii) decide how much work to execute for A_k this application.
- Select the application that has received the smallest relative share of the resource so far: α_k/π_k is minimum
- Break ties by giving priority to application with highest priority factor π_k
- Compare payoff of computing on local cluster with payoff of opening a route to the remote clusters. Choose most profitable cluster C^l
- Allocate an amount of work that does not overload C^l so that it will not be usable by other applications

Greedy Heuristic – Intuition

- At each step:
  (i) select an application A_k
  (ii) determine which cluster C^l executes the work
  (iii) decide how much work to execute for A_k this application.
- Select the application that has received the smallest relative share of the resource so far: α_k/π_k is minimum
- Break ties by giving priority to application with highest priority factor π_k
- Compare payoff of computing on local cluster with payoff of opening a route to the remote clusters. Choose most profitable cluster C^l
- Allocate an amount of work that does not overload C^l so that it will not be usable by other applications

Greedy Heuristic – Intuition

- At each step:
  (i) select an application A_k
  (ii) determine which cluster C^l executes the work
  (iii) decide how much work to execute for A_k this application.
- Select the application that has received the smallest relative share of the resource so far: α_k/π_k is minimum
- Break ties by giving priority to application with highest priority factor π_k
- Compare payoff of computing on local cluster with payoff of opening a route to the remote clusters. Choose most profitable cluster C^l
- Allocate an amount of work that does not overload C^l so that it will not be usable by other applications

Greedy Heuristic – Intuition

- At each step:
  (i) select an application A_k
  (ii) determine which cluster C^l executes the work
  (iii) decide how much work to execute for A_k this application.
- Select the application that has received the smallest relative share of the resource so far: α_k/π_k is minimum
- Break ties by giving priority to application with highest priority factor π_k
- Compare payoff of computing on local cluster with payoff of opening a route to the remote clusters. Choose most profitable cluster C^l
- Allocate an amount of work that does not overload C^l so that it will not be usable by other applications

Greedy Heuristic – Intuition

- At each step:
  (i) select an application A_k
  (ii) determine which cluster C^l executes the work
  (iii) decide how much work to execute for A_k this application.
- Select the application that has received the smallest relative share of the resource so far: α_k/π_k is minimum
- Break ties by giving priority to application with highest priority factor π_k
- Compare payoff of computing on local cluster with payoff of opening a route to the remote clusters. Choose most profitable cluster C^l
- Allocate an amount of work that does not overload C^l so that it will not be usable by other applications

General scheme

1. Let $L = \{C^1, \ldots, C^K\}$. Initialize all $\alpha_{k,l}$ and $\beta_{k,l}$ to 0
2. If L is empty, exit
3. **Select application** – Sort L by non-decreasing values of $\left(\frac{\alpha_k}{\pi_k}\right)$. Select A_k as first element of L.
4. **Select cluster** – $\text{benefit}_m = \min\left\{\frac{g_k}{\delta_k}, \frac{g_{k,m}}{\delta_k}, \frac{g_m}{\delta_k}, \frac{s_m}{w_k}\right\}$ $(m \neq k)$, $\text{benefit}_k = \frac{s_k}{w_k}$. Select C^l s.t. benefit_l max.
   If $\text{benefit}_l = 0$, remove C^k from L and go to step 2
5. **Determine amount of work** – If $k \neq l$, allocate $\text{alloc} = \text{benefit}_l$ to cluster C^l. If $k = l$, allocate only
   $\text{alloc} = \max_{m \neq k}\left\{\min\left\{\frac{g_k}{\delta_k}, \frac{g_{k,m}}{\delta_k}, \frac{g_m}{\delta_k}, \frac{s_m}{w_k}\right\}\right\}$ load units
6. **Update variables** – $s_l \leftarrow s_l - \text{alloc}.w_k$, $\alpha_{k,l} \leftarrow \alpha_{k,l} + \text{alloc}$
   If $k \neq l$, $\forall l_i \in L_{k,l}$, $\text{max-connect}(l_i) \leftarrow \text{max-connect}(l_i) - 1$,
   $g_k \leftarrow g_k - \text{alloc}.\delta_k$, $g_l \leftarrow g_l - \text{alloc}.\delta_k$, $\beta_{k,l} \leftarrow \beta_{k,l} + 1$
7. Go to step 2

LP-based heuristics

LPR: Round-off

$$\forall k, l, \quad \widehat{\beta}_{k,l} = \lfloor \widetilde{\beta}_{k,l} \rfloor, \quad \widehat{\alpha}_{k,l} = \min \left\{ \widetilde{\alpha}_{k,l}, \lfloor \widetilde{\beta}_{k,l} \rfloor \min_{l_i \in L_{k,l}} \mathsf{bw}(l_i) \right\}$$

LPRG: Round-off + Greedy

Use LPR, then reclaim residual capacity with Greedy heuristic

LPRR: Randomized Round-off Powerful but ☹ expensive (solve many LPs)

LP-based heuristics

LPR: Round-off

$$\forall k, l, \quad \widehat{\beta}_{k,l} = \lfloor \widetilde{\beta}_{k,l} \rfloor, \quad \widehat{\alpha}_{k,l} = \min \left\{ \widetilde{\alpha}_{k,l}, \lfloor \widetilde{\beta}_{k,l} \rfloor \min_{l_i \in L_{k,l}} \mathsf{bw}(l_i) \right\}$$

LPRG: Round-off + Greedy

Use LPR, then reclaim residual capacity with Greedy heuristic

LPRR: Randomized Round-off

Powerful but ☹ expensive (solve many LPs)

LP-based heuristics

LPR: Round-off

$$\forall k, l, \quad \widehat{\beta}_{k,l} = \lfloor \widetilde{\beta}_{k,l} \rfloor, \quad \widehat{\alpha}_{k,l} = \min \left\{ \widetilde{\alpha}_{k,l}, \lfloor \widetilde{\beta}_{k,l} \rfloor \min_{l_i \in L_{k,l}} \mathsf{bw}(l_i) \right\}$$

LPRG: Round-off + Greedy

Use LPR, then reclaim residual capacity with Greedy heuristic

LPRR: Randomized Round-off Powerful but ☹ expensive (solve many LPs)

Methodology

- Solution to *rational* linear problem as comparator/upper bound
- Use Tiers as topology generator:
  - 100 two-level topologies, each containing 40 WAN nodes, 30 MAN networks each containing 20 MAN nodes (no LAN) $\longrightarrow \approx$ 700 nodes
  - randomly select $K = 5, 7, \ldots, 90$ nodes as participating clusters, compute shortest paths (in hops)
  - pruned topology with computing nodes and routers
- For each pruned Tiers topology, randomly generate 10 configurations $\longrightarrow$ 29,298 platforms

Methodology

- Solution to *rational* linear problem as comparator/upper bound
- Use Tiers as topology generator:
  - 100 two-level topologies, each containing 40 WAN nodes, 30 MAN networks each containing 20 MAN nodes (no LAN) $\rightarrow \approx 700$ nodes
  - randomly select $K = 5, 7, \ldots, 90$ nodes as participating clusters, compute shortest paths (in hops)
  - pruned topology with computing nodes and routers
- For each pruned Tiers topology, randomly generate 10 configurations $\rightarrow$ 29,298 platforms

Methodology

- Solution to *rational* linear problem as comparator/upper bound
- Use Tiers as topology generator:
  - 100 two-level topologies, each containing 40 WAN nodes, 30 MAN networks each containing 20 MAN nodes (no LAN) $\rightarrow \approx 700$ nodes
  - randomly select $K = 5, 7, \ldots, 90$ nodes as participating clusters, compute shortest paths (in hops)
  - pruned topology with computing nodes and routers
- For each pruned Tiers topology, randomly generate 10 configurations $\rightarrow$ 29,298 platforms

Methodology

- Solution to *rational* linear problem as comparator/upper bound
- Use Tiers as topology generator:
  - 100 two-level topologies, each containing 40 WAN nodes, 30 MAN networks each containing 20 MAN nodes (no LAN) $\rightarrow \approx 700$ nodes
  - randomly select $K = 5, 7, \ldots, 90$ nodes as participating clusters, compute shortest paths (in hops)
  - pruned topology with computing nodes and routers
- For each pruned Tiers topology, randomly generate 10 configurations $\rightarrow$ 29,298 platforms

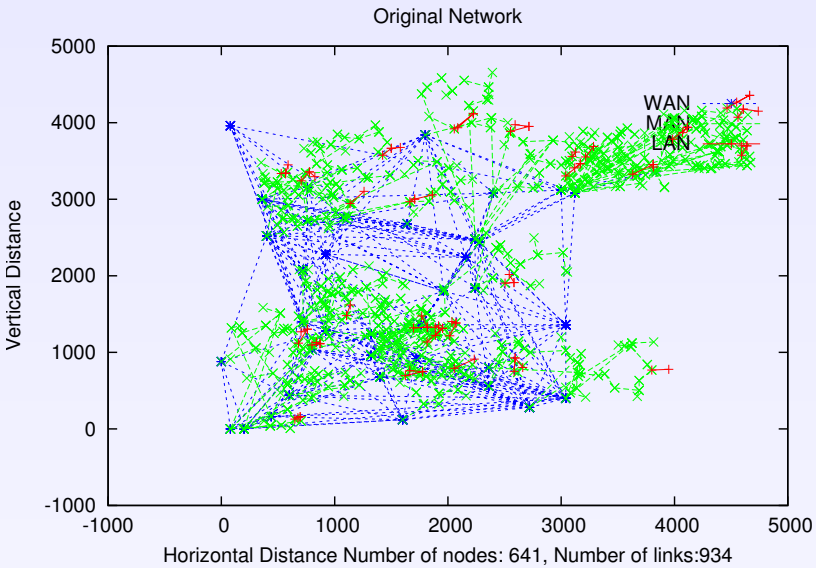
Methodology

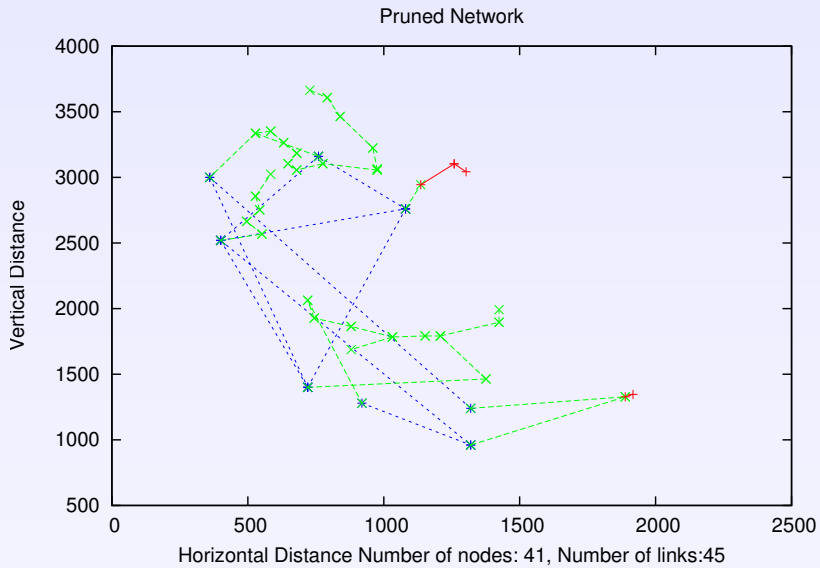
- Solution to *rational* linear problem as comparator/upper bound
- Use Tiers as topology generator:
  - 100 two-level topologies, each containing 40 WAN nodes, 30 MAN networks each containing 20 MAN nodes (no LAN) $\rightarrow \approx 700$ nodes
  - randomly select $K = 5, 7, \ldots, 90$ nodes as participating clusters, compute shortest paths (in hops)
  - pruned topology with computing nodes and routers
- For each pruned Tiers topology, randomly generate 10 configurations $\rightarrow$ 29,298 platforms

Methodology

- Solution to *rational* linear problem as comparator/upper bound
- Use Tiers as topology generator:
  - 100 two-level topologies, each containing 40 WAN nodes, 30 MAN networks each containing 20 MAN nodes (no LAN) $\rightarrow \approx 700$ nodes
  - randomly select $K = 5, 7, \ldots, 90$ nodes as participating clusters, compute shortest paths (in hops)
  - pruned topology with computing nodes and routers
- For each pruned Tiers topology, randomly generate 10 configurations $\rightarrow$ 29,298 platforms

Methodology (cont'd)

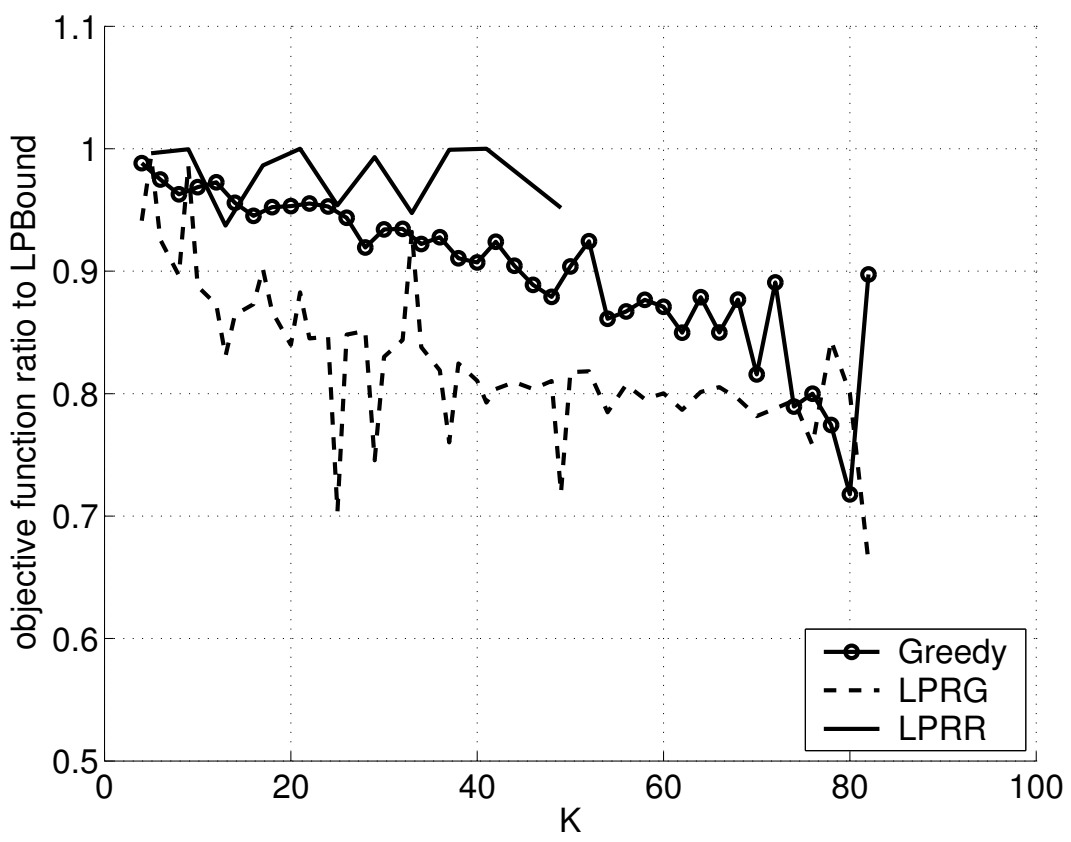




	distribution
K	$5, 7, \ldots, 90$
$\log(bw(l_k))$, $\log(g_k)$	normal ($mean$= $\log(2000)$, std=$\log(10)$)
s_k	uniform, 1000 — 10000
max-connect, δ_k, w_k, π_k	uniform, 1 — 10

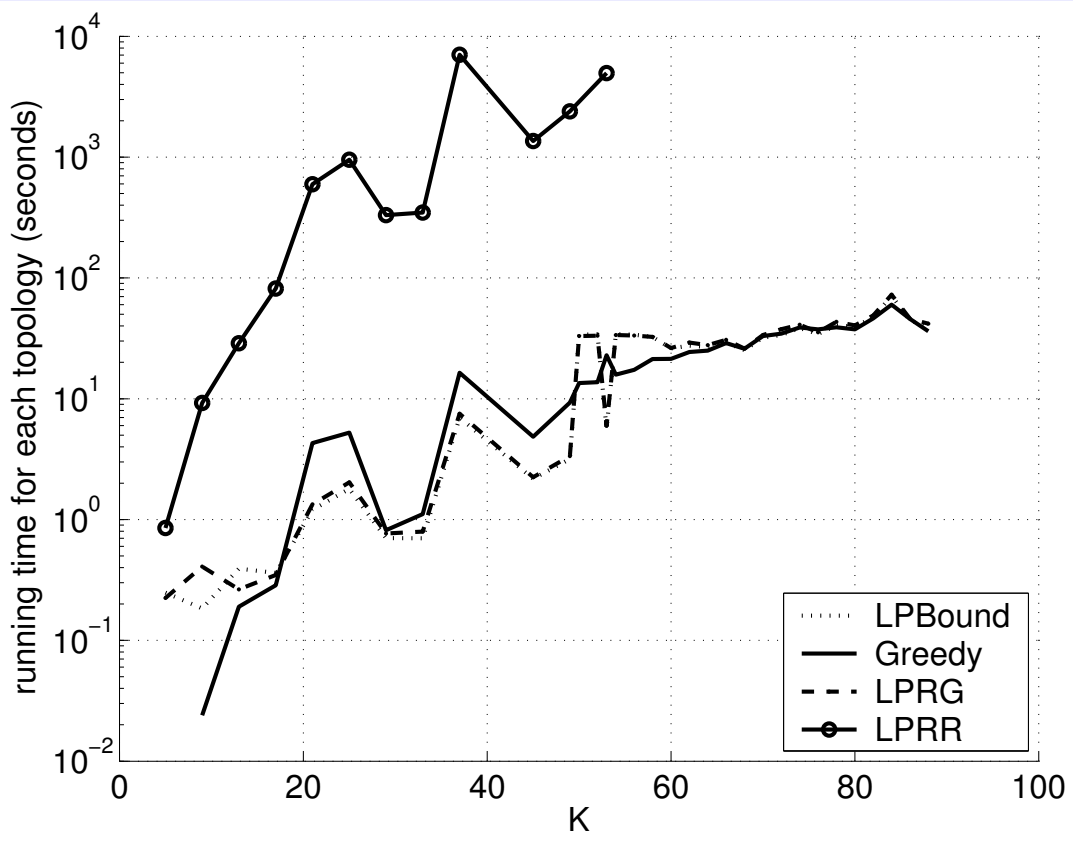
Platform parameters used in simulation

Results



- LPR very bad, sometimes no computation at all
- G better than LPRG
- LPRR very good but time consuming

Running time



- LPRR very good but time consuming (each run $\approx$ one hour at $K = 50$)

Hints for implementation

- Participants sharing resources in a Virtual Organization
- Centralized broker managing applications and resources
- Broker gathers all parameters of LP program
- Priority factors
- Various policies and refinements possible
  $\Rightarrow$ e.g. fixed number of connections per application

Hints for implementation

- Participants sharing resources in a Virtual Organization
- Centralized broker managing applications and resources
- Broker gathers all parameters of LP program
- Priority factors
- Various policies and refinements possible
  $\Rightarrow$ e.g. fixed number of connections per application

Hints for implementation

- Participants sharing resources in a Virtual Organization
- Centralized broker managing applications and resources
- Broker gathers all parameters of LP program
- Priority factors
- Various policies and refinements possible
  $\Rightarrow$ e.g. fixed number of connections per application

Hints for implementation

- Participants sharing resources in a Virtual Organization
- Centralized broker managing applications and resources
- Broker gathers all parameters of LP program
- Priority factors
- Various policies and refinements possible
  $\Rightarrow$ e.g. fixed number of connections per application

Hints for implementation

- Participants sharing resources in a Virtual Organization
- Centralized broker managing applications and resources
- Broker gathers all parameters of LP program
- Priority factors
- Various policies and refinements possible
  $\Rightarrow$ e.g. fixed number of connections per application

Summary

- Key points:
  - Realistic platform model
  - Mixed integer-rational linear program
  - Weighted priorities, fair resource sharing between applications
  - Several heuristics, greedy and LP-based
- Extensions:
  - Simulate platforms and application parameters (by default, values from real-world testbeds and applications suites)
  - Model: include link latencies, TCP bandwidth sharing according to RTT
  - Handle divisible load applications consisting of a set of tasks linked by dependencies
    $\rightarrow$ ☺ mixed task and data parallelism for heterogeneous clusters and grids

Summary

- Key points:
  - Realistic platform model
  - Mixed integer-rational linear program
  - Weighted priorities, fair resource sharing between applications
  - Several heuristics, greedy and LP-based
- Extensions:
  - Simulate platforms and application parameters that are moreinfled from real-world testbeds and applications suites
  - Model: include link latencies, TCP bandwidth sharing according to RTT
  - Handle divisible load applications consisting of a set of tasks linked by dependencies
    $\rightarrow$ ☺ mixed task and data parallelism for heterogeneous clusters and grids

Summary

- Key points:
  - Realistic platform model
  - Mixed integer-rational linear program
  - Weighted priorities, fair resource sharing between applications
  - Several heuristics, greedy and LP-based
- Extensions:
  - Simulate platforms and application parameters (by default, values from real-world testbeds and applications suites)
  - Model: include link latencies, TCP bandwidth sharing according to RTT
  - Handle divisible load applications consisting of a set of tasks linked by dependencies
    → ☺ mixed task and data parallelism for heterogeneous clusters and grids

Summary

- Key points:
  - Realistic platform model
  - Mixed integer-rational linear program
  - Weighted priorities, fair resource sharing between applications
  - Several heuristics, greedy and LP-based
- Extensions:
  - Simulate platforms and application parameters that are measured from real-world testbeds and applications suites
  - Model: include link latencies, TCP bandwidth sharing according to RTT
  - Handle divisible load applications consisting of a set of tasks linked by dependencies
    → ☺ mixed task and data parallelism for heterogeneous clusters and grids

Summary

- Key points:
  - Realistic platform model
  - Mixed integer-rational linear program
  - Weighted priorities, fair resource sharing between applications
  - Several heuristics, greedy and LP-based
- Extensions:
  - Simulate platforms and application parameters that are measured from real-world testbeds and applications suites
  - Model: include link latencies, TCP bandwidth sharing according to RTT
  - Handle divisible load applications consisting of a set of tasks linked by dependencies
    → ☺ mixed task and data parallelism for heterogeneous clusters and grids

Summary

- Key points:
  - Realistic platform model
  - Mixed integer-rational linear program
  - Weighted priorities, fair resource sharing between applications
  - Several heuristics, greedy and LP-based
- Extensions:
  - Simulate platforms and application parameters that are measured from real-world testbeds and applications suites
  - Model: include link latencies, TCP bandwidth sharing according to RTT
  - Handle divisible load applications consisting of a set of tasks linked by dependencies
    $\longrightarrow$ ☺ mixed task and data parallelism for heterogeneous clusters and grids

Summary

- Key points:
  - Realistic platform model
  - Mixed integer-rational linear program
  - Weighted priorities, fair resource sharing between applications
  - Several heuristics, greedy and LP-based
- Extensions:
  - Simulate platforms and application parameters that are measured from real-world testbeds and applications suites
  - Model: include link latencies, TCP bandwidth sharing according to RTT
  - Handle divisible load applications consisting of a set of tasks linked by dependencies
    $\longrightarrow$ ☺ mixed task and data parallelism for heterogeneous clusters and grids

Summary

- Key points:
  - Realistic platform model
  - Mixed integer-rational linear program
  - Weighted priorities, fair resource sharing between applications
  - Several heuristics, greedy and LP-based
- Extensions:
  - Simulate platforms and application parameters that are measured from real-world testbeds and applications suites
  - Model: include link latencies, TCP bandwidth sharing according to RTT
  - Handle divisible load applications consisting of a set of tasks linked by dependencies
    $\longrightarrow$ ☺ mixed task and data parallelism for heterogeneous clusters and grids

Summary

- Key points:
  - Realistic platform model
  - Mixed integer-rational linear program
  - Weighted priorities, fair resource sharing between applications
  - Several heuristics, greedy and LP-based
- Extensions:
  - Simulate platforms and application parameters that are measured from real-world testbeds and applications suites
  - Model: include link latencies, TCP bandwidth sharing according to RTT
  - Handle divisible load applications consisting of a set of tasks linked by dependencies
    $\longrightarrow$ ☺ mixed task and data parallelism for heterogeneous clusters and grids

Bibliography

- Tiers:
  Modeling Internet topology, K. Calvert, M. Doar and E.W. Zegura, IEEE Comm. Magazine 35, 6 (1997), 160-163
- Scheduling multiple applications:
  A realistic network/application model for scheduling divisible loads on large-scale platforms, L. Marchal et al., 19th IEEE IPDPS (2005), 48b

Outline

Key advantages of steady-state scheduling

Simplicity

- From local equations to global behavior
- Resource constraints $\Rightarrow$ dependence constraints

Efficiency

- Periodic schedule, described in compact form
- Asymptotic optimality

Adaptability

- Record observed performance during current period
- Inject information to compute schedule for next period
- React on the fly to resource availability variations

Scheduling for heterogeneous platforms

- If the platform is well identified and relatively stable, try to:
  (i) accurately model the (expected) hierarchical structure of the platform
  (ii) design scheduling algorithms well-suited to this hierarchical structure

- If the platform is not stable enough, or if it evolves too fast, dynamic schedulers are the only option

- Otherwise, grab the opportunity to *inject some static knowledge into dynamic schedulers*:
  ☹ Is this opportunity a niche?
  ☺ Does it encompasses a wide range of applications?