Steady-state scheduling on CELL

Mathias Jacquelin,

joint work with Matthieu Gallet, Loris Marchal and Yves Robert

INRIA GRAAL project-team
LIP (ENS-Lyon, CNRS, INRIA)
École Normale Supérieure de Lyon, France

"Scheduling for large-scale systems" workshop,
Knoxville, May 14, 2009.

Outline

Motivation

- Multicore architectures: new opportunity to test the scheduling strategies designed in the GRAAL team.
- Our trademark: efficient scheduling on heterogeneous platforms

- Most multicore architecture are homogeneous, regular
  - Need for tailored algorithms (linear algebra,...)
- Emerging heterogeneous multicore:
  - Dedicated processing units on GPUs
  - Mixed system: processor + accelerator

- This study: steady-state scheduling on CELL (bounded heterogeneity) to demonstrate the usefulness of complex (static) scheduling techniques
- Ongoing work: only preliminary results

Motivation

- Multicore architectures: new opportunity to test the scheduling strategies designed in the GRAAL team.
- Our trademark: efficient scheduling on heterogeneous platforms

- Most multicore architecture are homogeneous, regular
  - Need for tailored algorithms (linear algebra,...)
- Emerging heterogeneous multicore:
  - Dedicated processing units on GPUs
  - Mixed system: processor + accelerator

- This study: steady-state scheduling on CELL (bounded heterogeneity) to demonstrate the usefulness of complex (static) scheduling techniques
- Ongoing work: only preliminary results

Motivation

- Multicore architectures: new opportunity to test the scheduling strategies designed in the GRAAL team.
- Our trademark: efficient scheduling on heterogeneous platforms

- Most multicore architecture are homogeneous, regular
  - Need for tailored algorithms (linear algebra,...)
- Emerging heterogeneous multicore:
  - Dedicated processing units on GPUs
  - Mixed system: processor + accelerator

- This study: steady-state scheduling on CELL (bounded heterogeneity) to demonstrate the usefulness of complex (static) scheduling techniques
- Ongoing work: only preliminary results

Introduction: Steady-state Scheduling

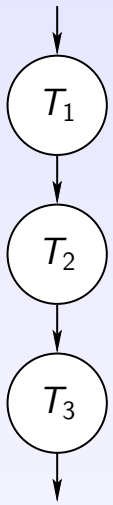
Rationale:

- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per seconds)
- Today: simple case where each task has to be mapped on one single resource

Introduction: Steady-state Scheduling

Rationale:

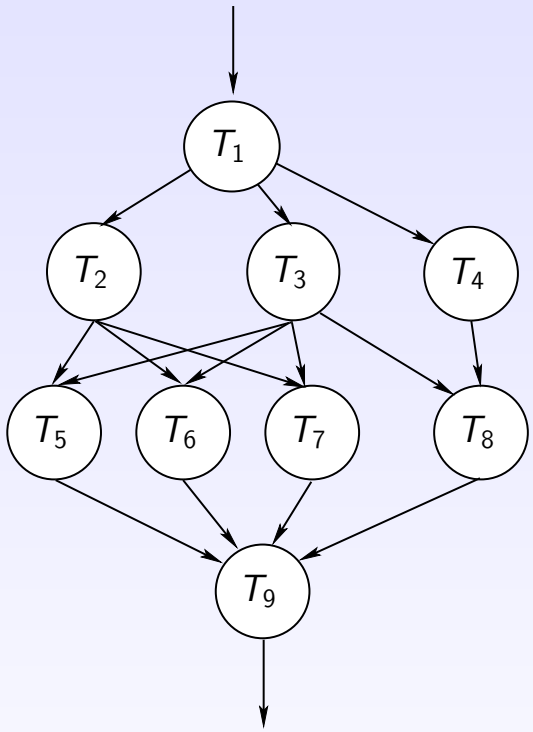
- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per seconds)
- Today: simple case where each task has to be mapped on one single resource



Introduction: Steady-state Scheduling

Rationale:

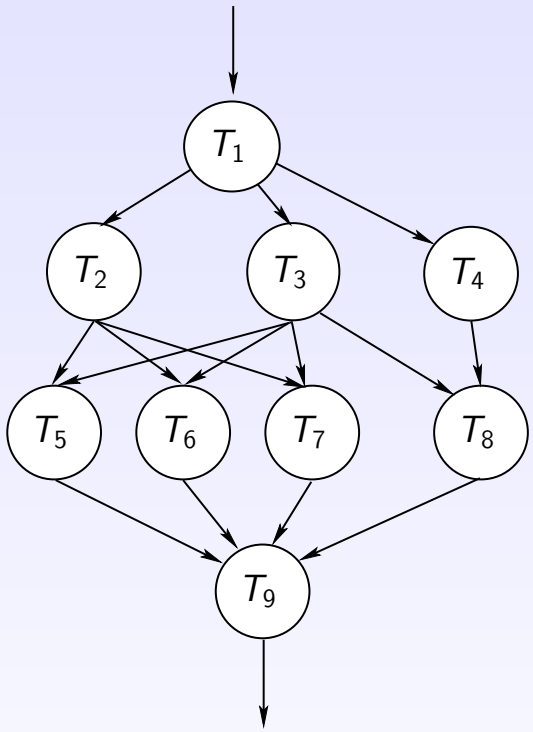
- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per seconds)
- Today: simple case where each task has to be mapped on one single resource



Introduction: Steady-state Scheduling

Rationale:

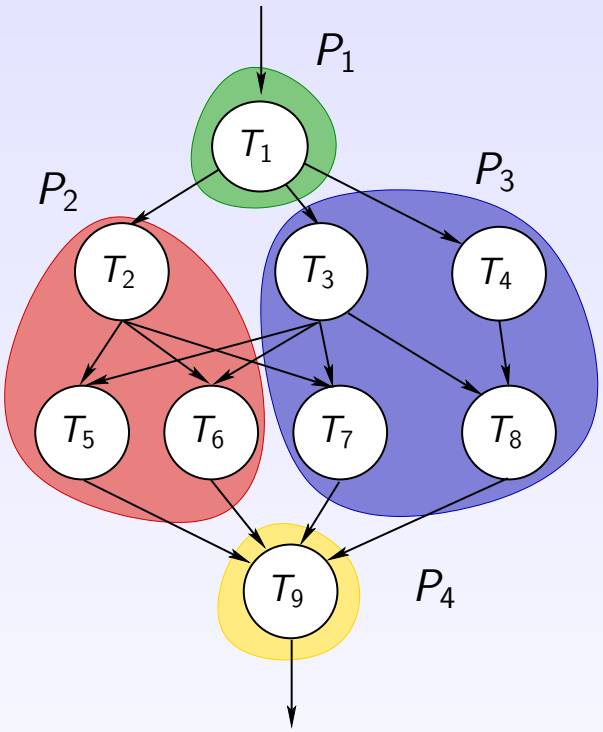
- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per seconds)
- Today: simple case where each task has to be mapped on one single resource



Introduction: Steady-state Scheduling

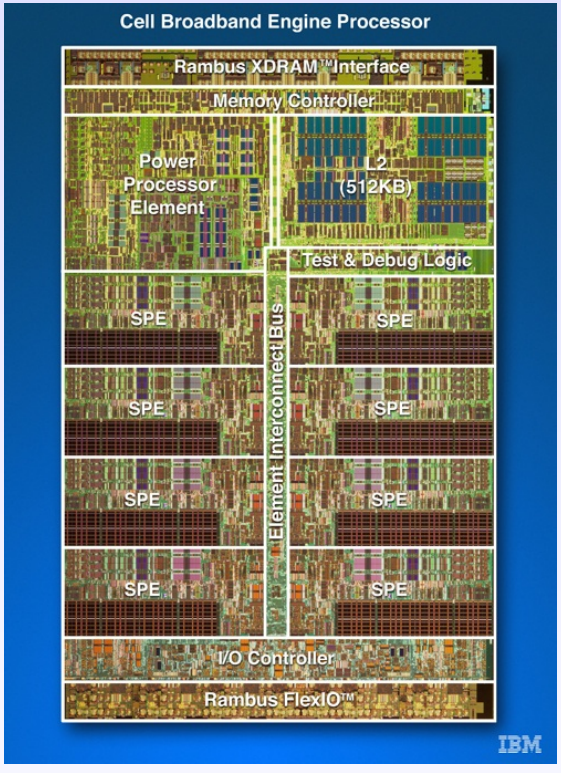
Rationale:

- A pipelined application:
  - Simple chain
  - More complex application (Directed Acyclic Graph)
- Objective: optimize the throughput of the application (number of input files treated per seconds)
- Today: simple case where each task has to be mapped on one single resource



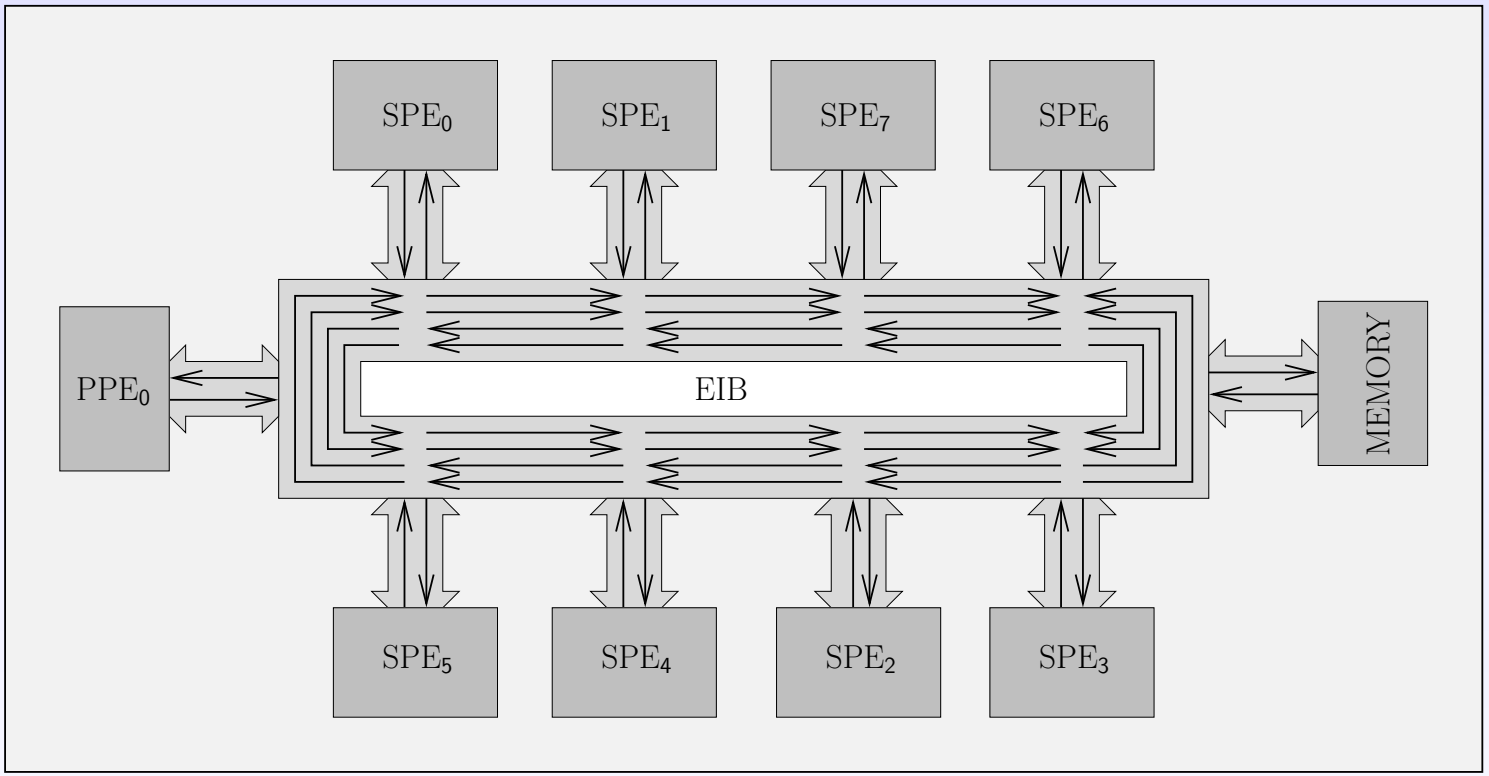
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



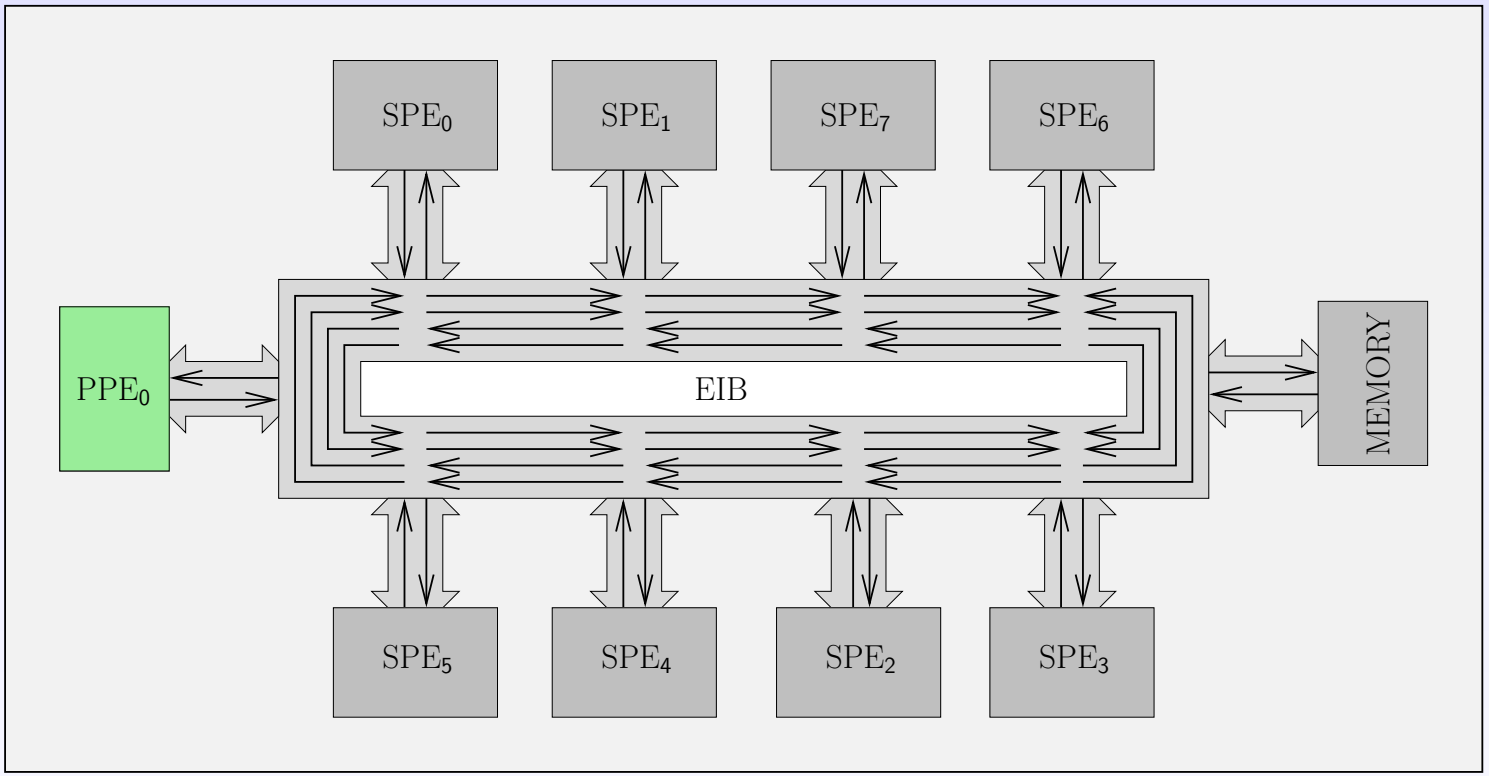
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



CELL brief introduction

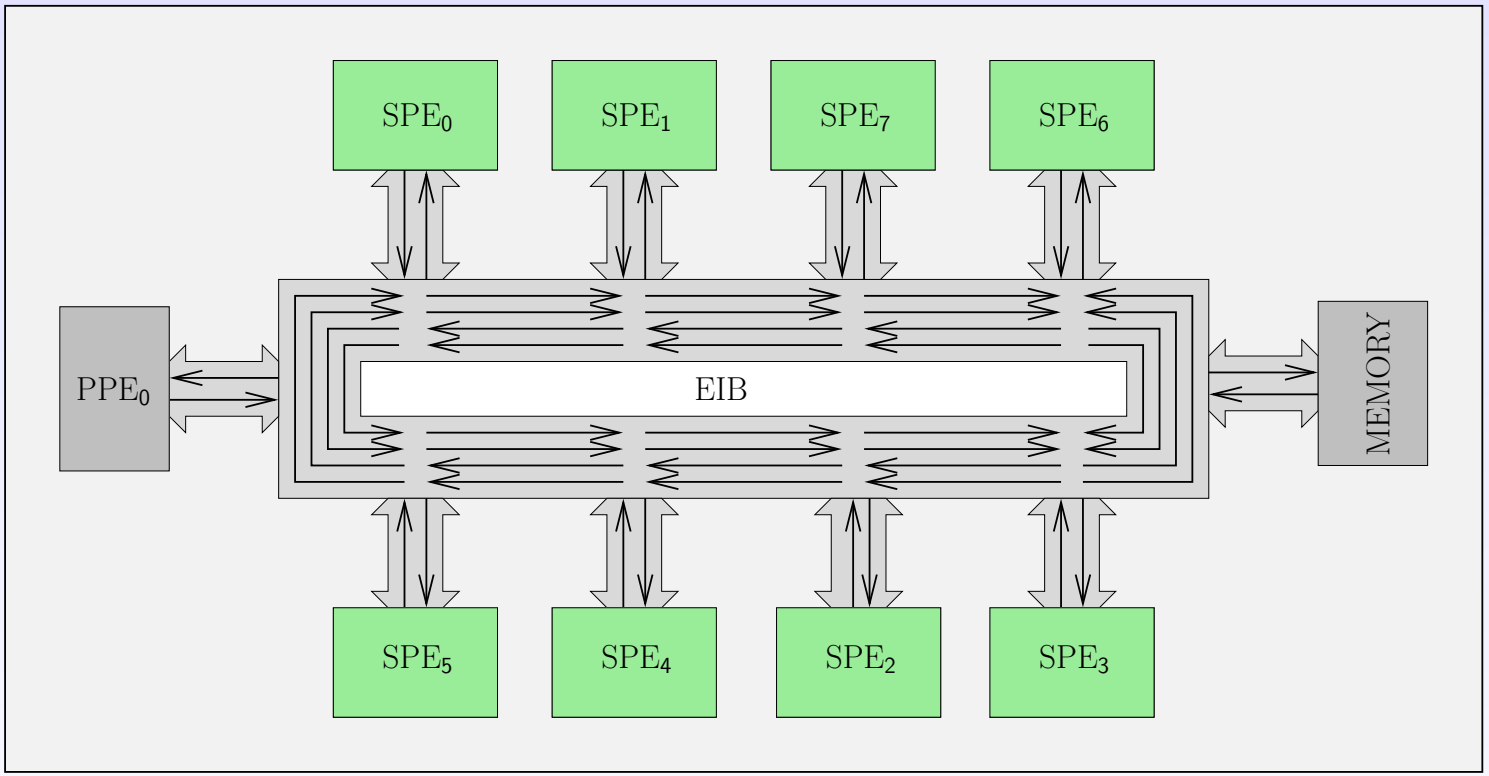
- Multicore heterogeneous processor
- Accelerator extension to Power architecture



- 1 PPE core
  - VMX unit
  - L1, L2 cache
  - 2 way SMT

CELL brief introduction

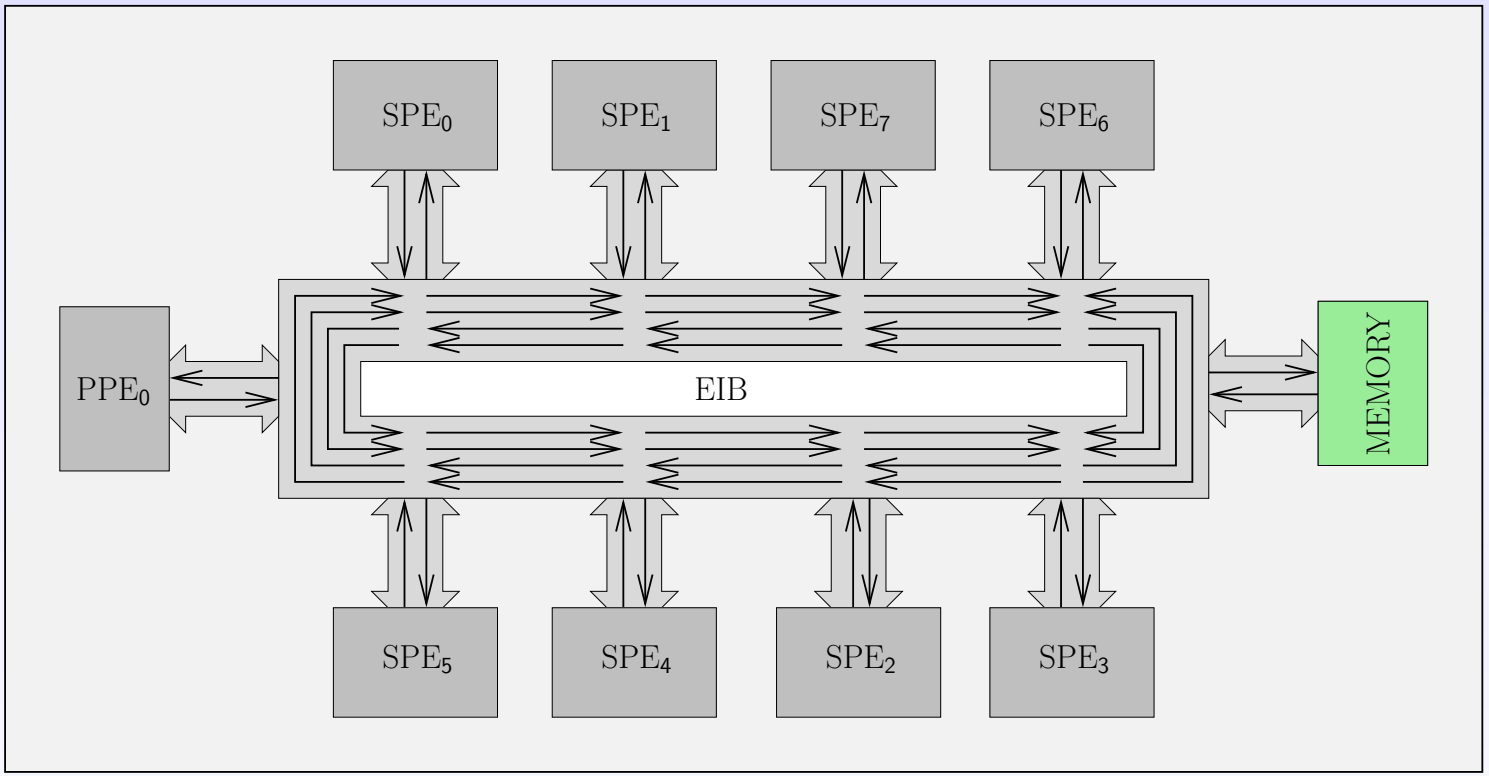
- Multicore heterogeneous processor
- Accelerator extension to Power architecture



- 8 SPEs
  - 128-bit SIMD instruction set
  - Local store 256KB
  - Dedicated Asynchronous DMA engine

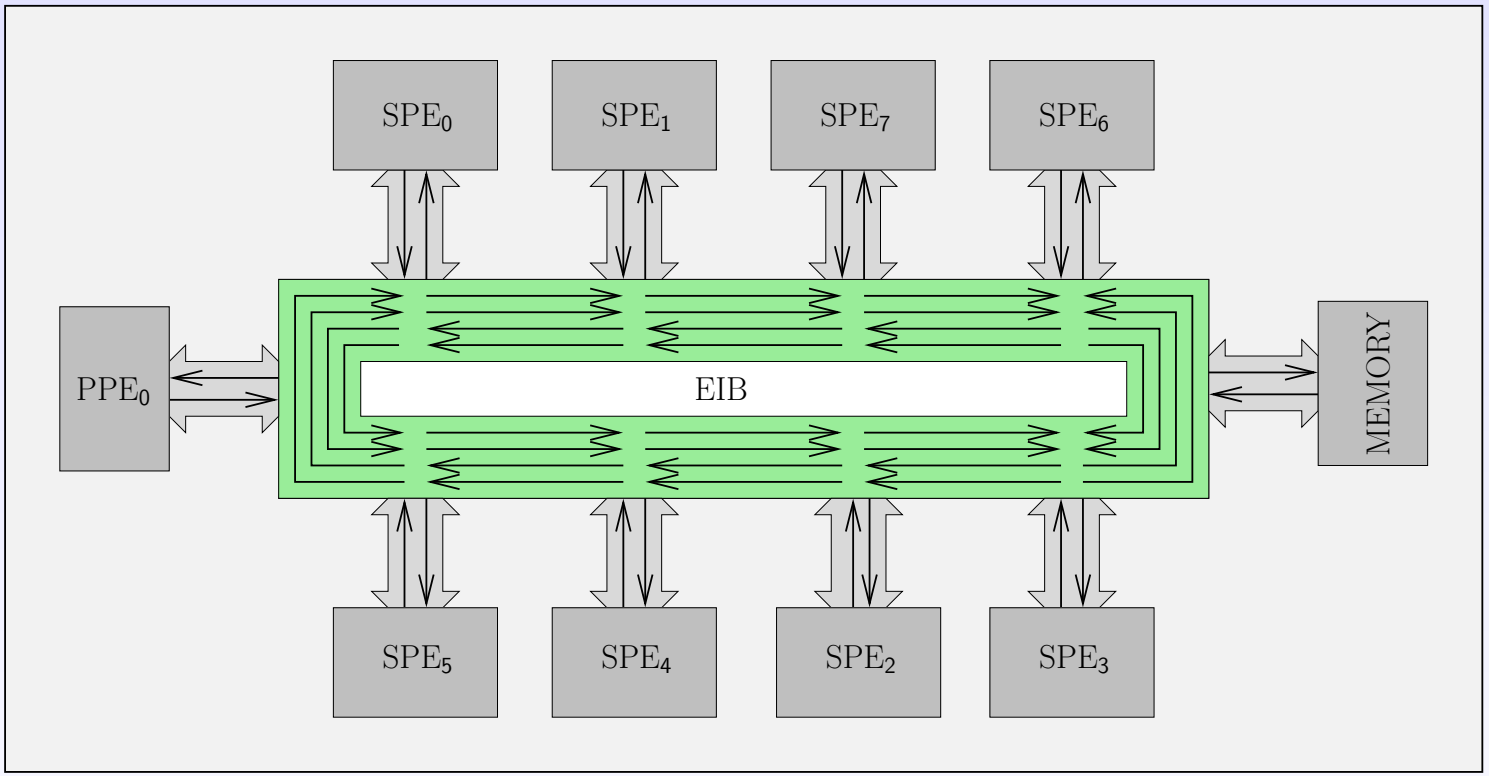
CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



CELL brief introduction

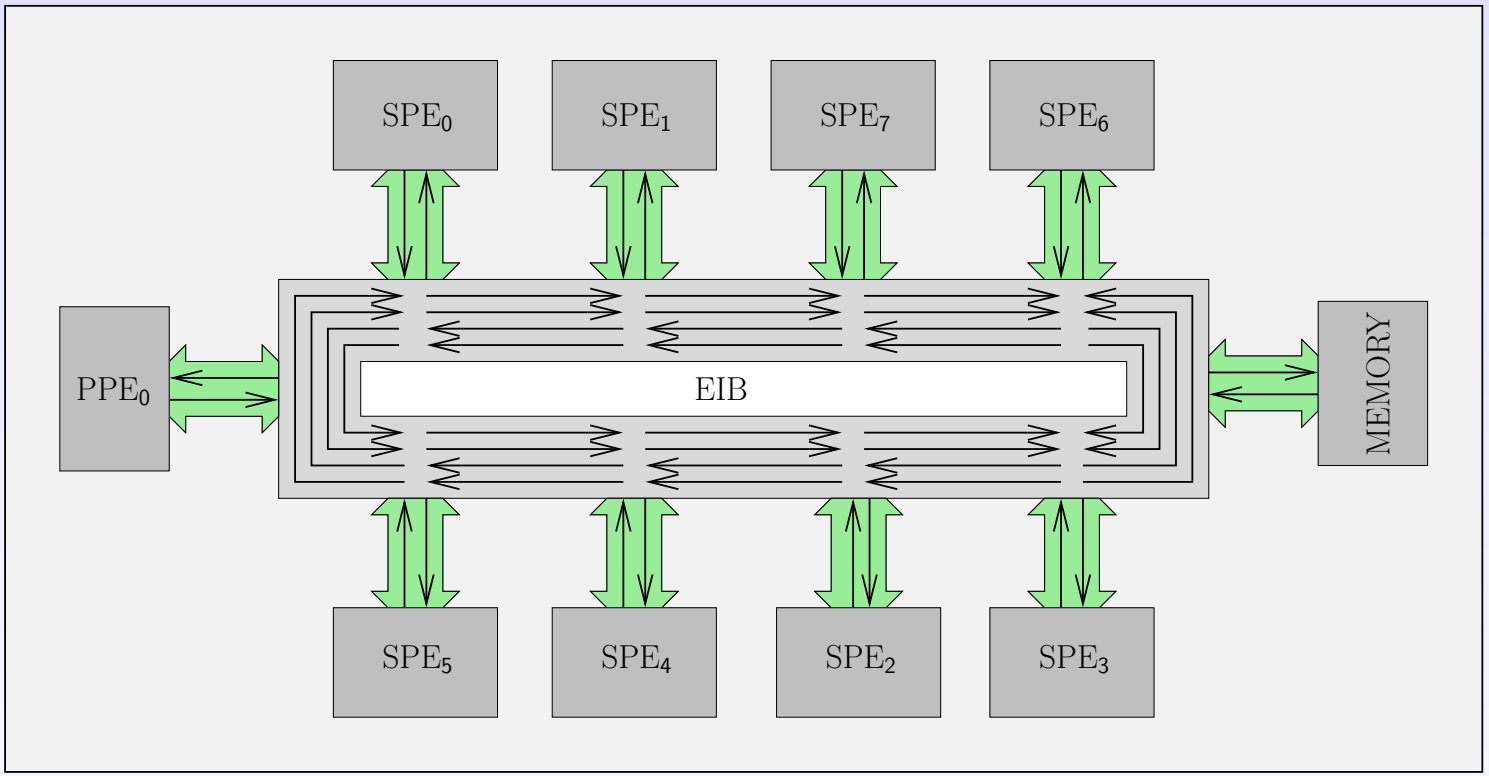
- Multicore heterogeneous processor
- Accelerator extension to Power architecture



- Element Interconnect Bus (EIB)
  - 200 GB/s bandwidth

CELL brief introduction

- Multicore heterogeneous processor
- Accelerator extension to Power architecture



- 25 GB/s bandwidth

Outline

Platform modeling

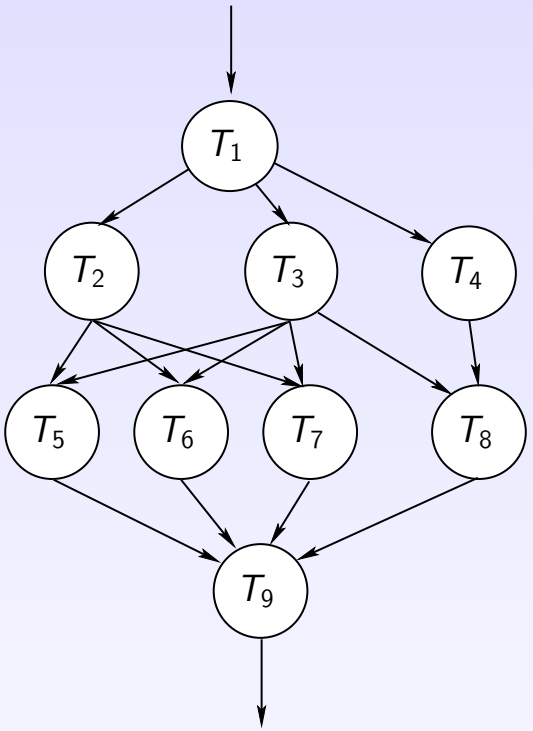
Simple CELL modeling:

- 1 PPE and 8 SPE: 9 processing elements $P_1, \ldots, P_9$, with *unrelated* speed,
- Each processing element access the communication bus with a (bidirectional) bandwidth $b = (25GB/s)$,
- The bus is able to route all concurrent communications without contention (in a first step),
- Due to the limited size of the DMA stack on each SPE:
  - Each SPE can perform at most 16 simultaneous DMA operations,
  - The PPE can perform at most 8 simultaneous DMA operations to/from a given SPE.
- Linear cost communication model:
  a data of size S is sent/received in time S/b

Application modeling

Application is described by a directed acyclic graph:

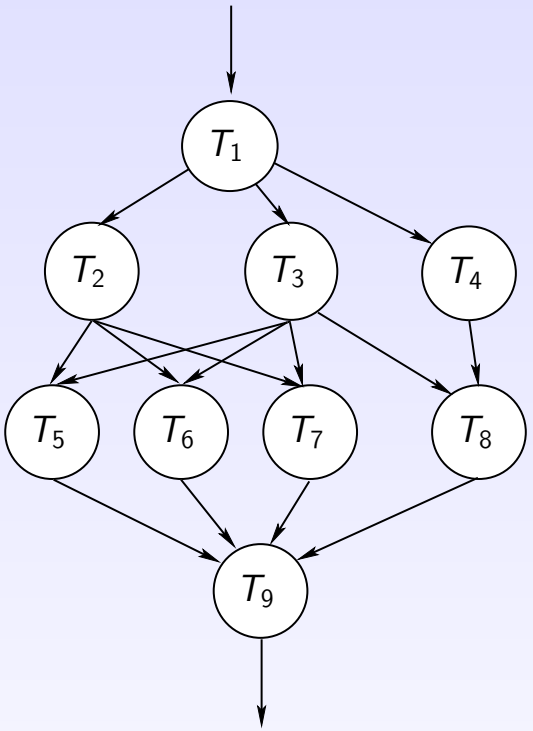
- Tasks $T_1, \ldots, T_n$
- Processing time of task T_k on P_i is $t_i(k)$,
- If there is a dependency $T_k \rightarrow T_l$, $\text{data}_{k,l}$ is the size of the file produced by T_k and needed by T_l,
- If T_k is an input task, it reads read_k bytes from main memory,
- If T_k is an output task, it writes write_k bytes to main memory,



Application modeling

Application is described by a directed acyclic graph:

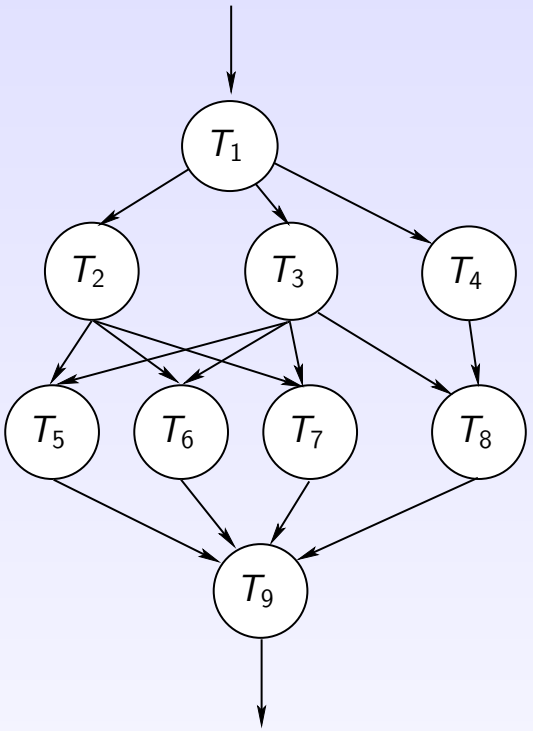
- Tasks $T_1, \ldots, T_n$
- Processing time of task T_k on P_i is $t_i(k)$,
- If there is a dependency $T_k \rightarrow T_l$, $\text{data}_{k,l}$ is the size of the file produced by T_k and needed by T_l,
- If T_k is an input task, it reads read_k bytes from main memory,
- If T_k is an output task, it writes write_k bytes to main memory,



Application modeling

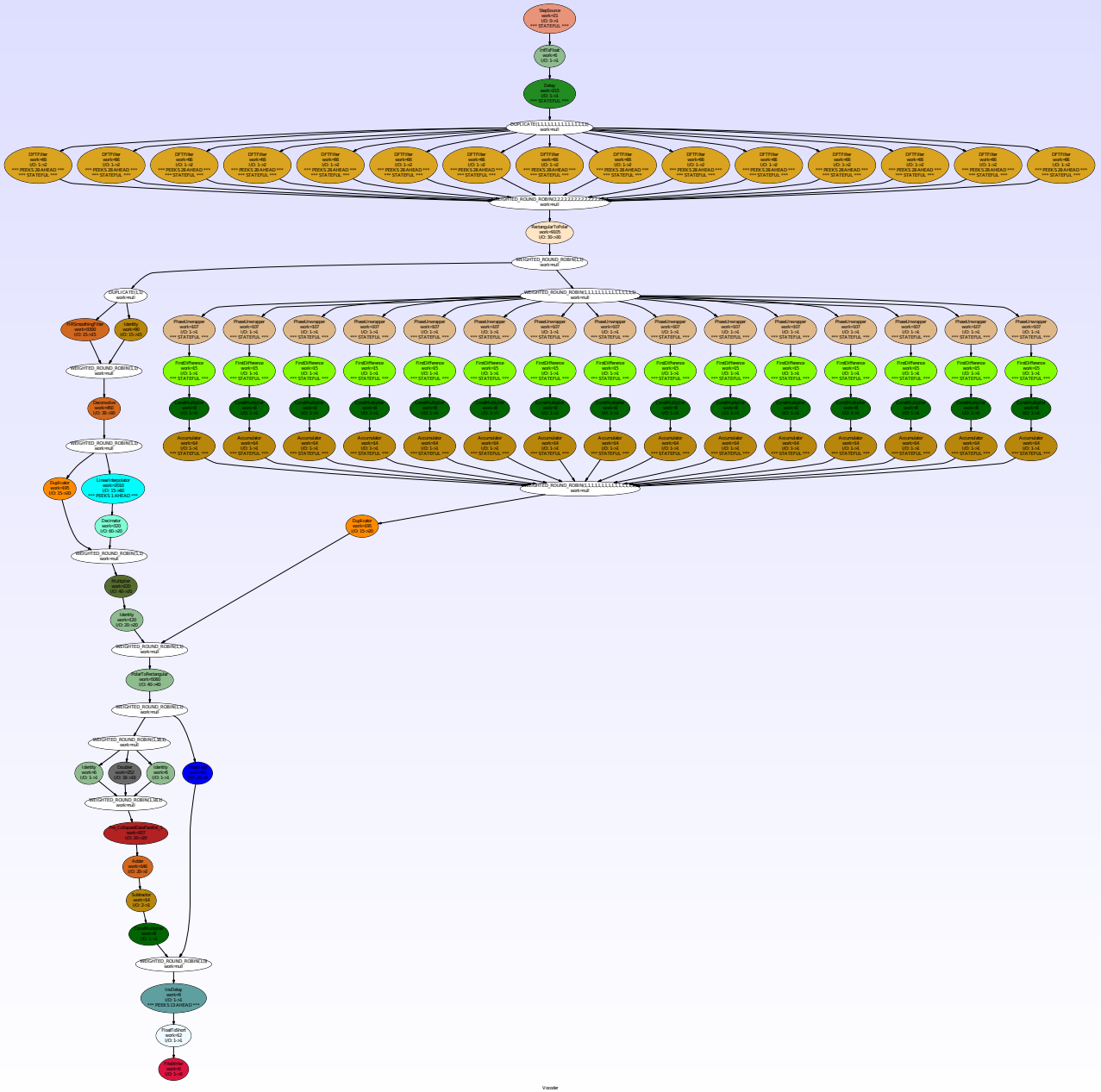
Application is described by a directed acyclic graph:

- Tasks $T_1, \ldots, T_n$
- Processing time of task T_k on P_i is $t_i(k)$,
- If there is a dependency $T_k \rightarrow T_l$, $\text{data}_{k,l}$ is the size of the file produced by T_k and needed by T_l,



- If T_k is an input task, it reads read_k bytes from main memory,
- If T_k is an output task, it writes write_k bytes to main memory,

Target application: vocoder



Outline

How to compute an optimal mapping

- Ojective: maximize throughput ρ
- Method: write a linear program gathering constraints on the mapping
- Binary variables: $\alpha_i^k = \begin{cases} 1 & \text{if } T_k \text{ is mapped on } P_i \\ 0 & \text{otherwise} \end{cases}$
- Other useful binary variables: $\beta_{i,j}^{k,l} = 1$ iff file $T_k \longrightarrow T_l$ is transfered from P_i to P_j

Constraints 1/2

On the application structure:

- Each task is mapped on a processor:

$$\forall T_k \quad \sum_i \alpha_i^k = 1$$

- Given a dependency $T_k \rightarrow T_l$, the processor computing T_l must receive the corresponding file:

$$\forall (k,l) \in E, \forall P_j, \quad \sum_i \beta_{i,j}^{k,l} \geq \alpha_j^l$$

- Given a dependency $T_k \rightarrow T_l$, only the processor computing T_k can send the corresponding file:

$$\forall (k,l) \in E, \forall P_i, \quad \sum_j \beta_{i,j}^{k,l} \leq \alpha_i^k$$

Constraints 2/2

On the achievable throughput $\rho = 1/T$:

- On a given processor, all tasks must be completed within T:

$$\forall P_i, \quad \sum_k \alpha_i^k \times t_i(k) \leq T$$

- All incoming communications must be completed within T:

$$\forall P_j, \quad \frac{1}{b}\Big(\sum_k \alpha_j^k \times \text{read}_k + \sum_{k,l}\sum_i \beta_{i,j}^{k,l} \times \text{data}_{k,l}\Big) \leq T$$

- All outgoing communications must be completed within T:

$$\forall P_i, \quad \frac{1}{b}\Big(\sum_k \alpha_i^k \times \text{write}_k + \sum_{k,l}\sum_i \beta_{i,j}^{k,l} \times \text{data}_{k,l}\Big) \leq T$$

+ constraints on the number of incoming/outgoing communications to respect the DMA requirements
+ constraints on the available memory on SPE

Optimal mapping computation

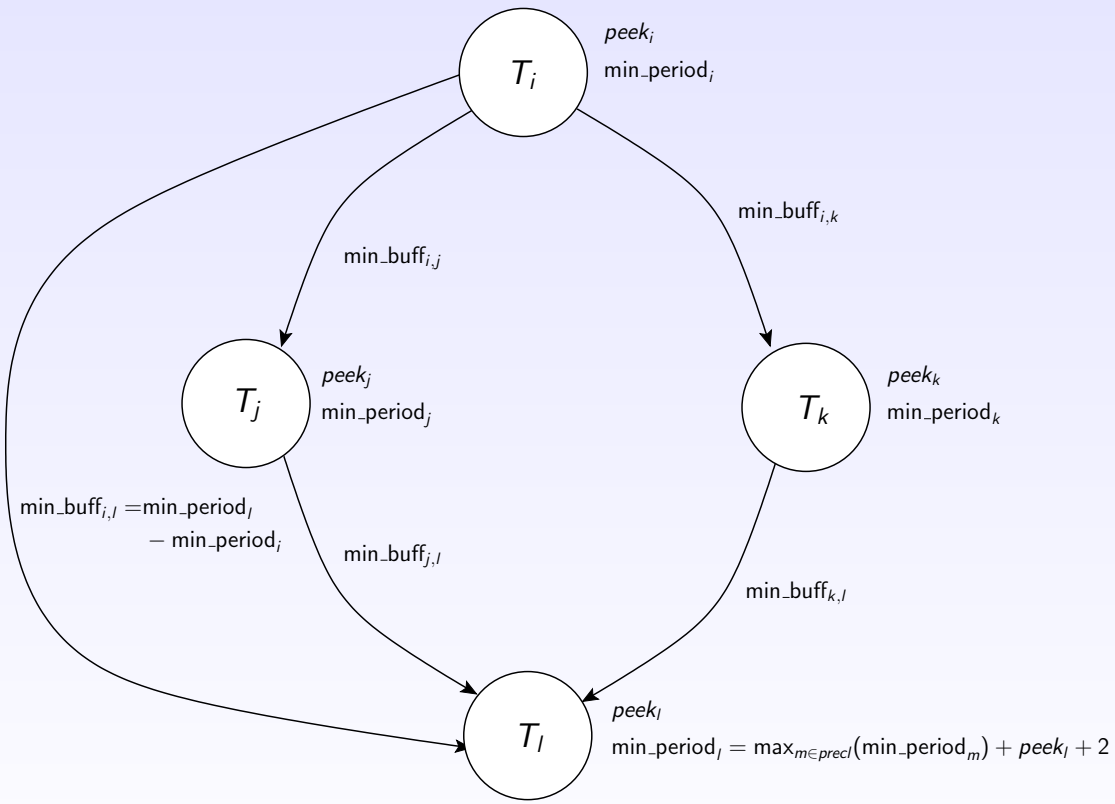
- Linear program with the objective of minimizing T
- Integer (binary) variables: Mixed Integer Programming
- NP-complete problem
- Efficient solvers exist with short running time
  - for small-size problems
  - or when an approximate solution is searched
- We use CPLEX, and look for an approximate solution (5% of the optimal throughput is good enough)

Outline

Preprocessing of the schedule

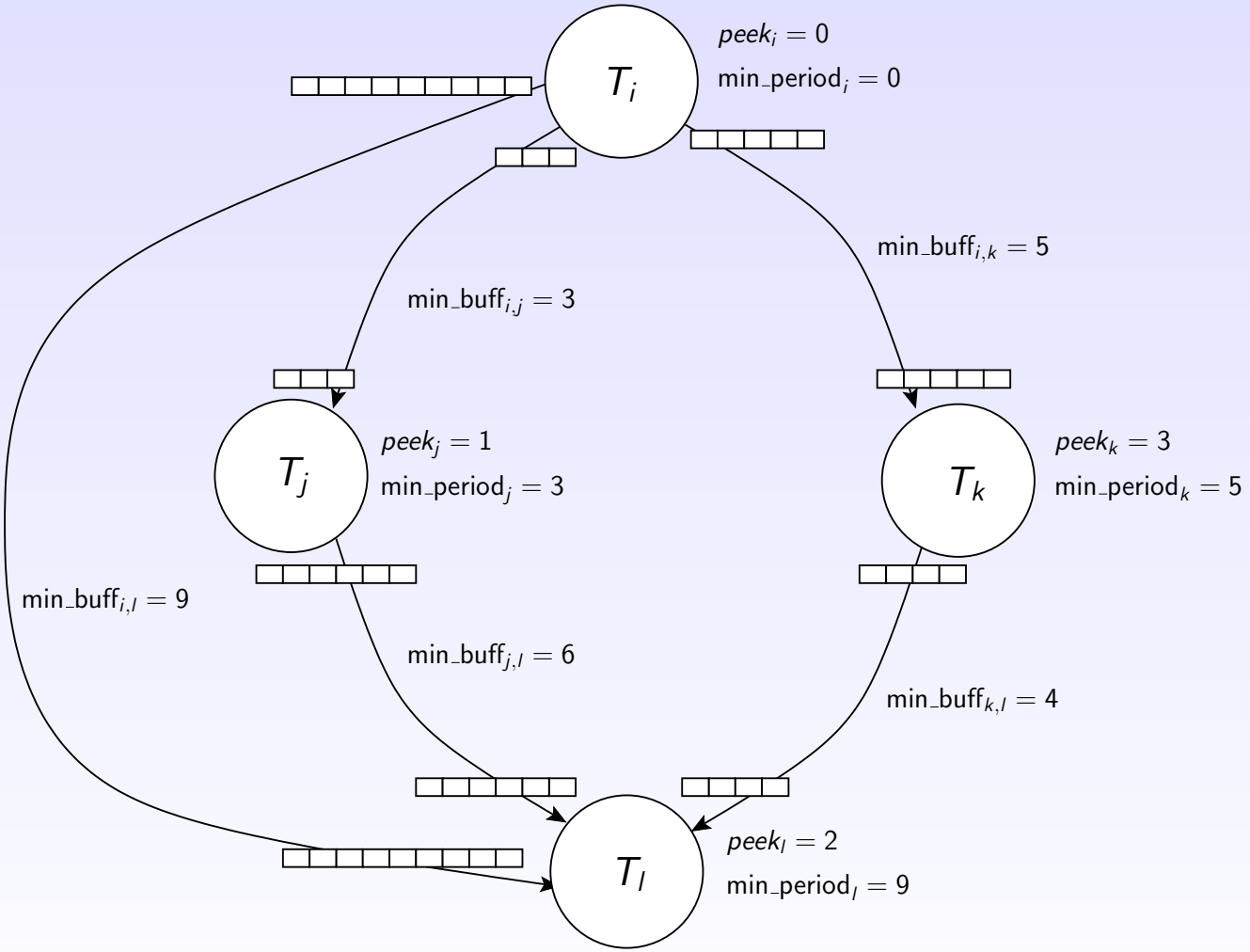
Main Objective: Compute minimal starting period and buffer sizes.

- $\text{min\_period}_l = \max_{m \in precl}(\text{min\_period}_m) + peek_l + 2$
- $\text{min\_buff}_{i,l} = \text{min\_period}_l - \text{min\_period}_i$



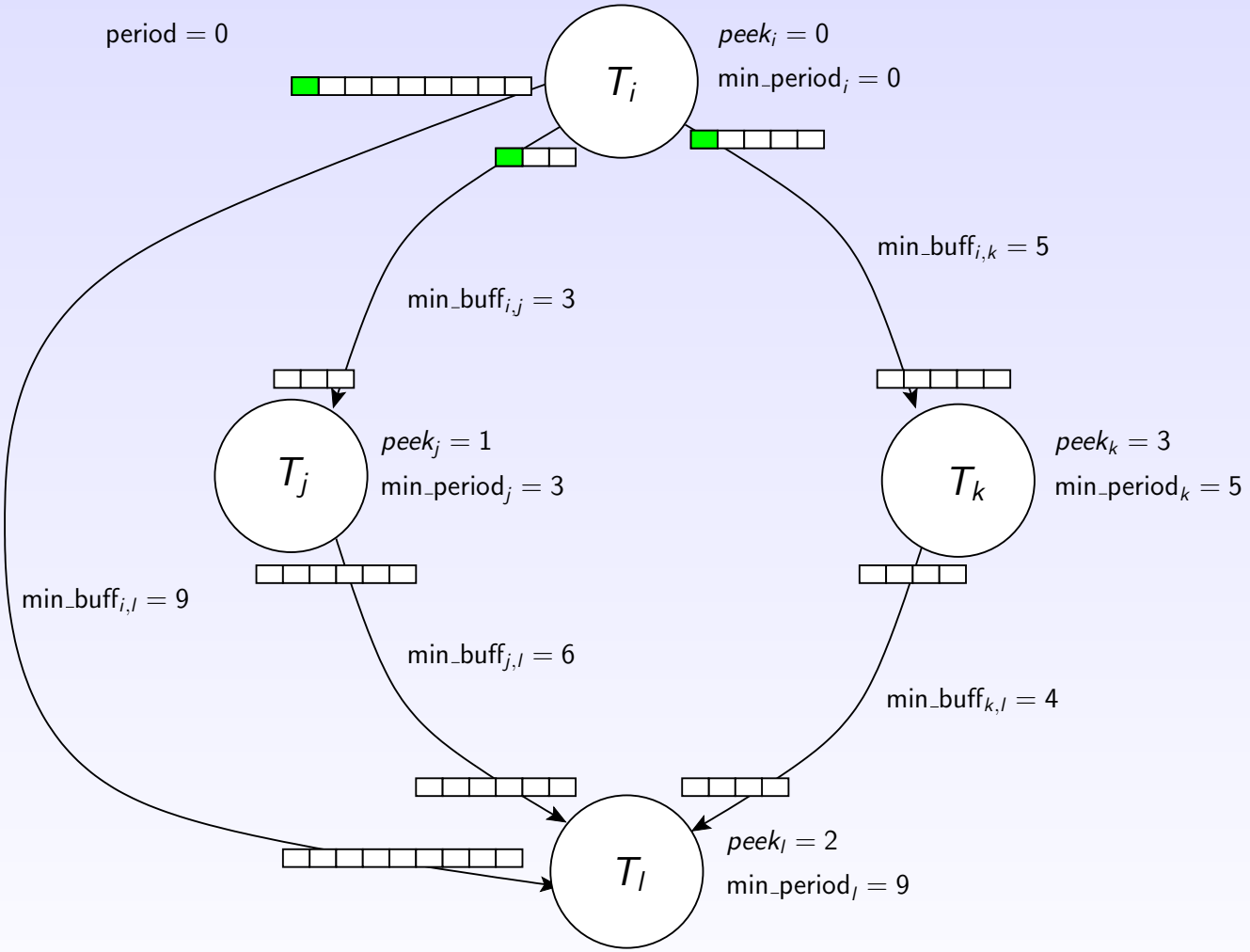
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



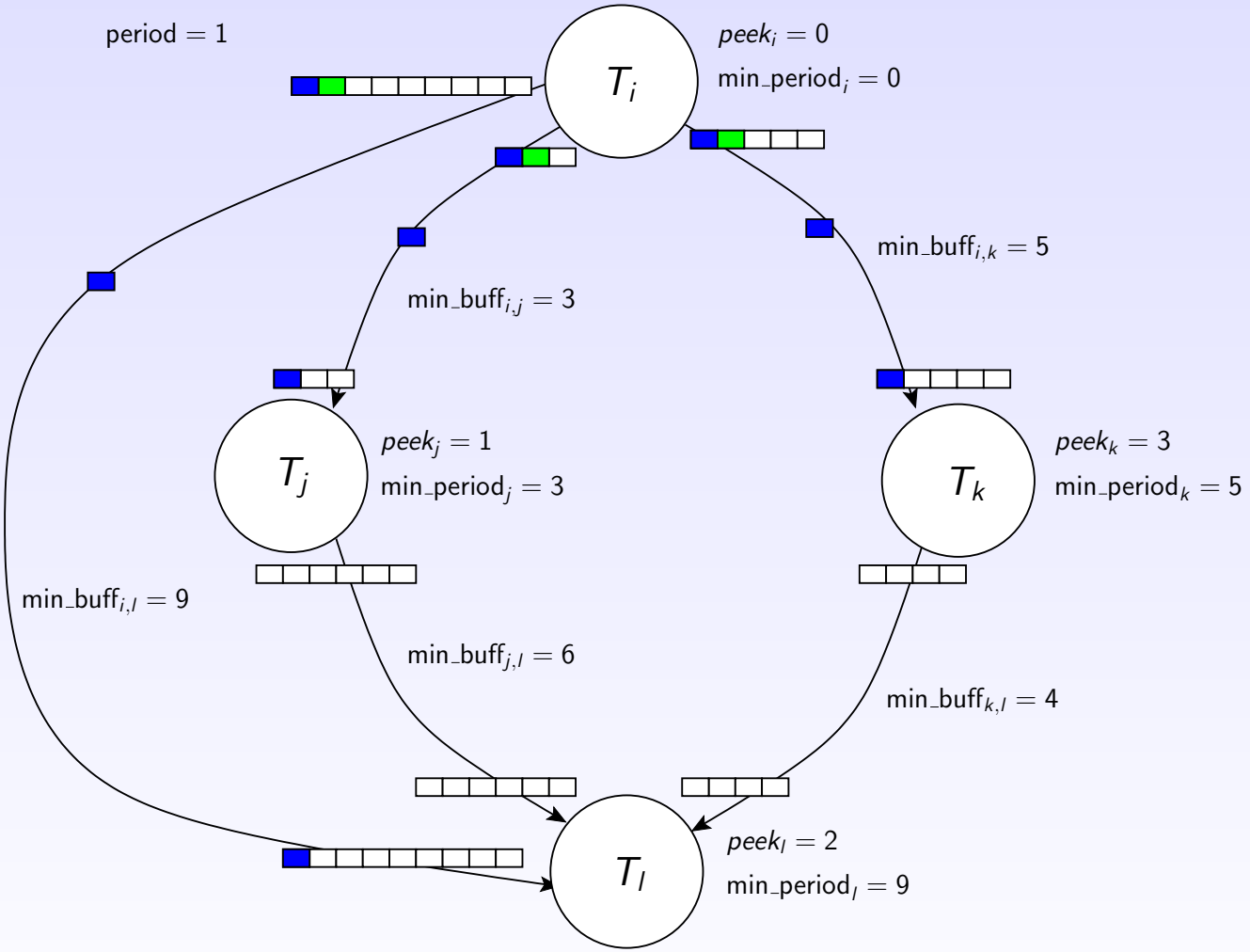
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



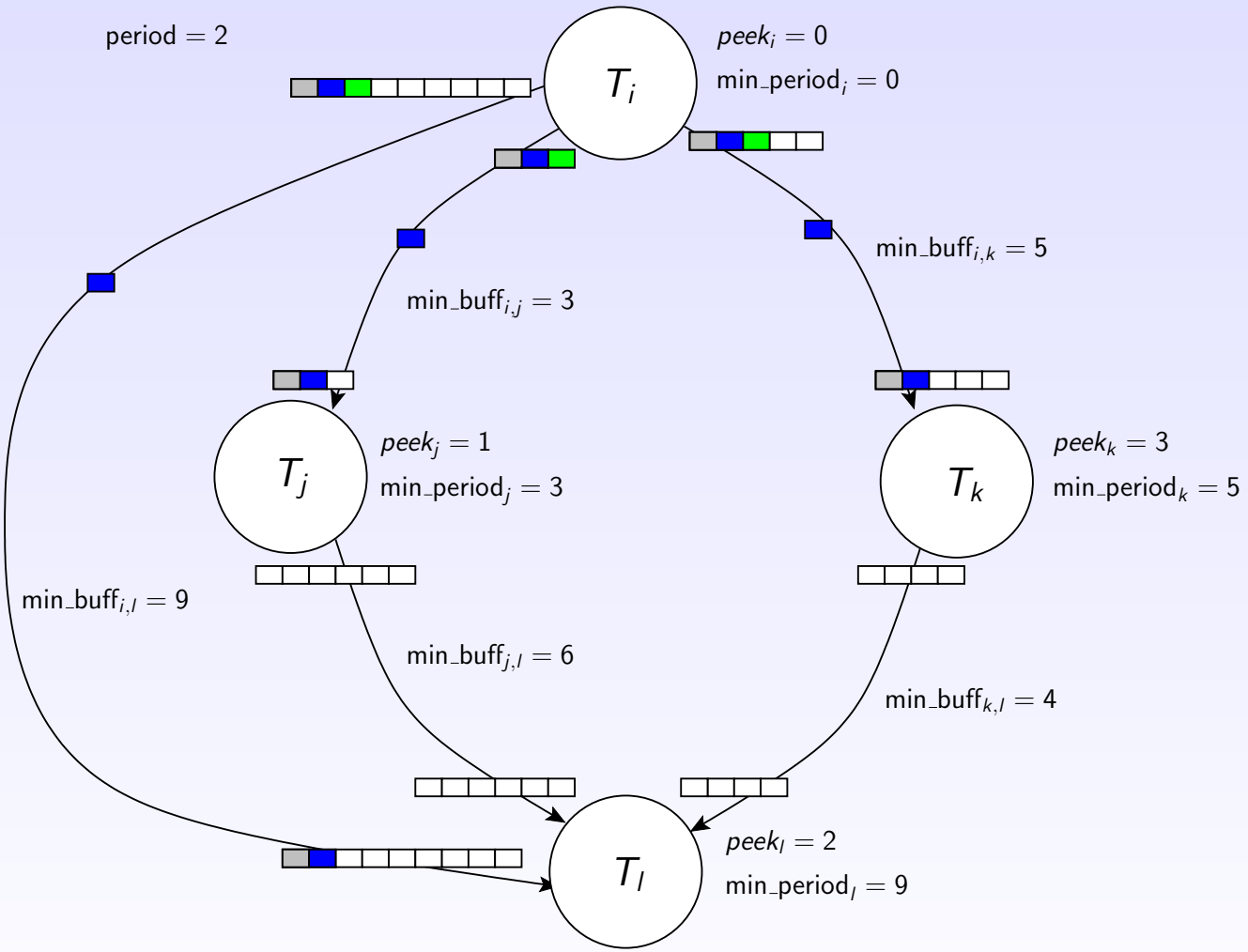
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



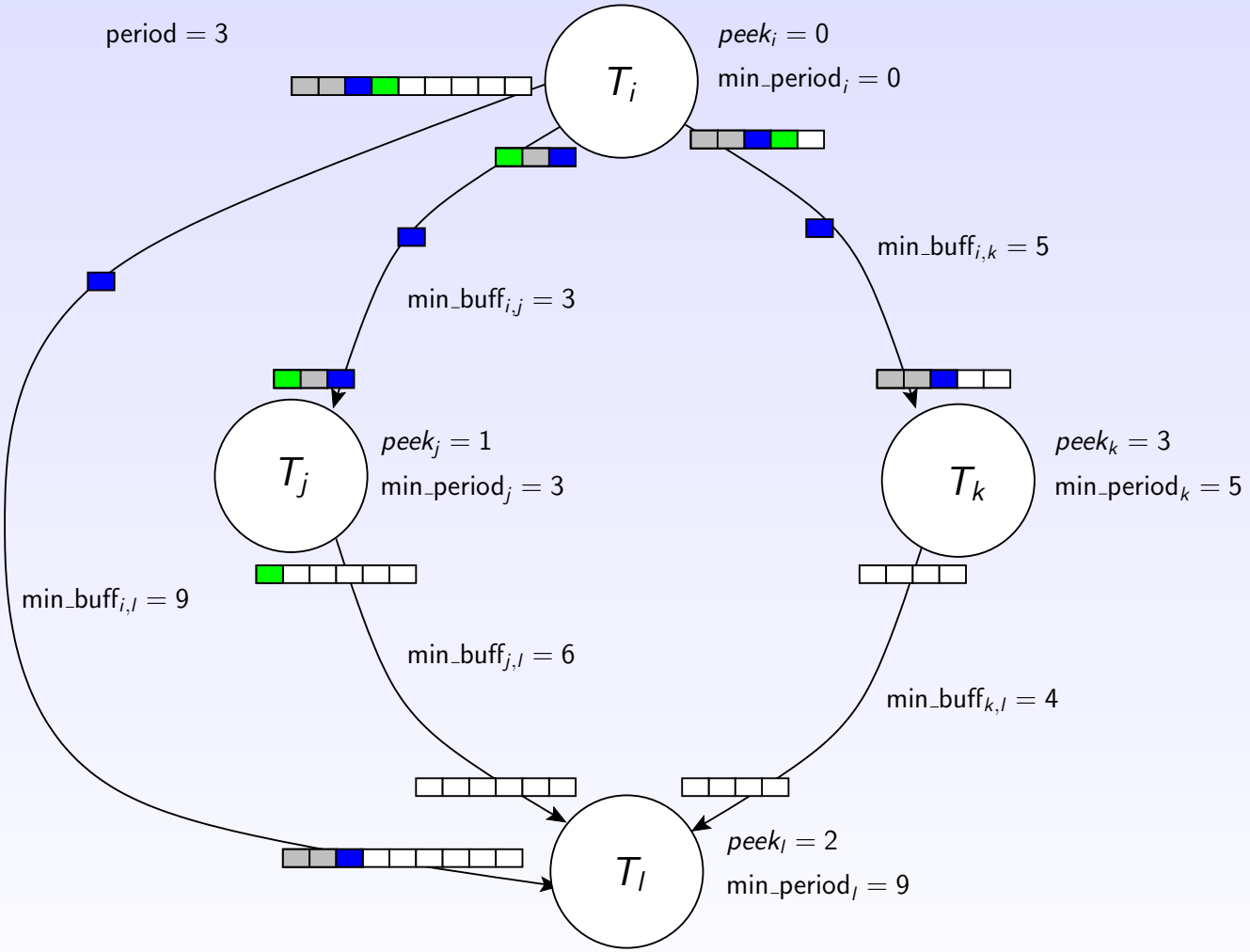
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



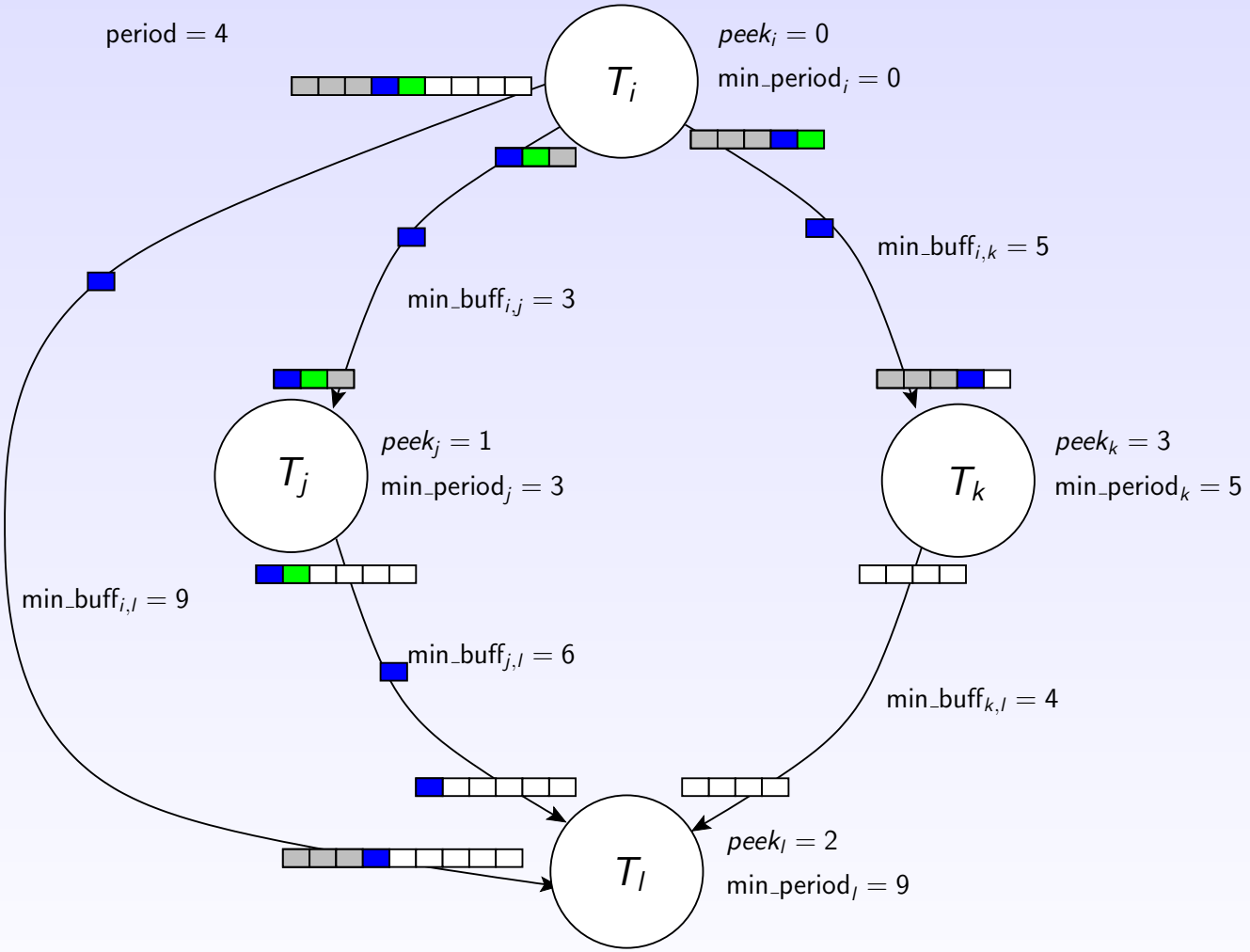
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



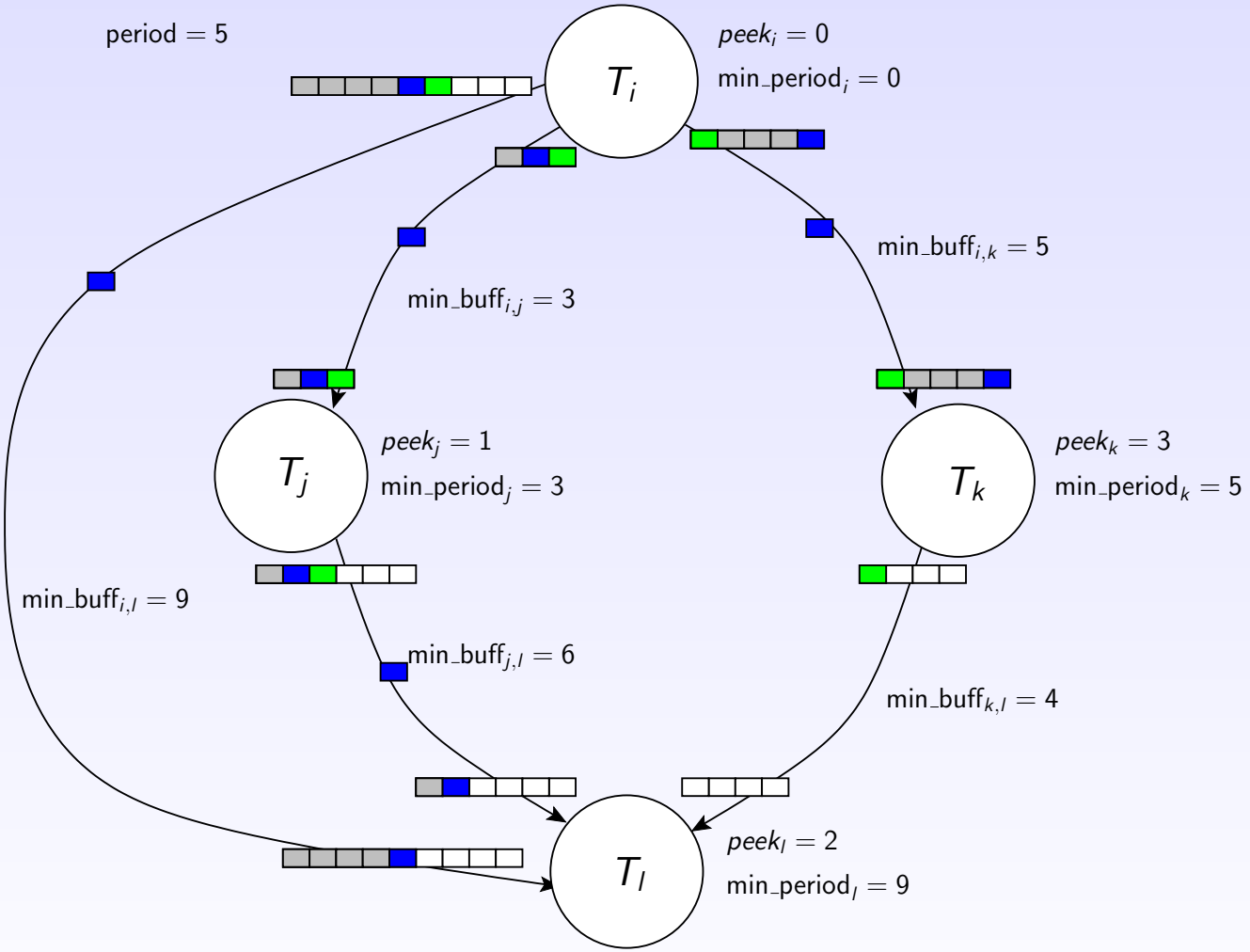
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



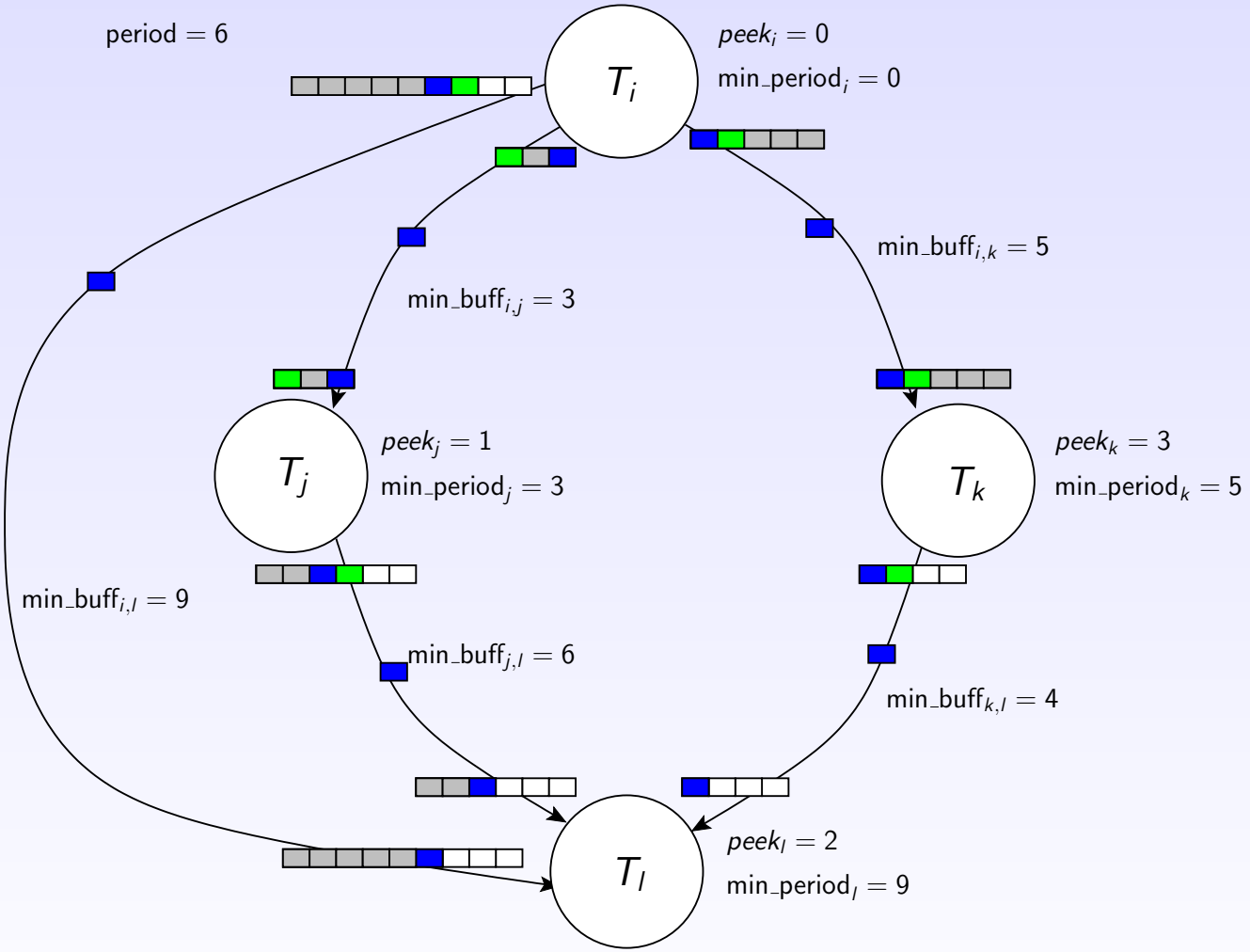
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



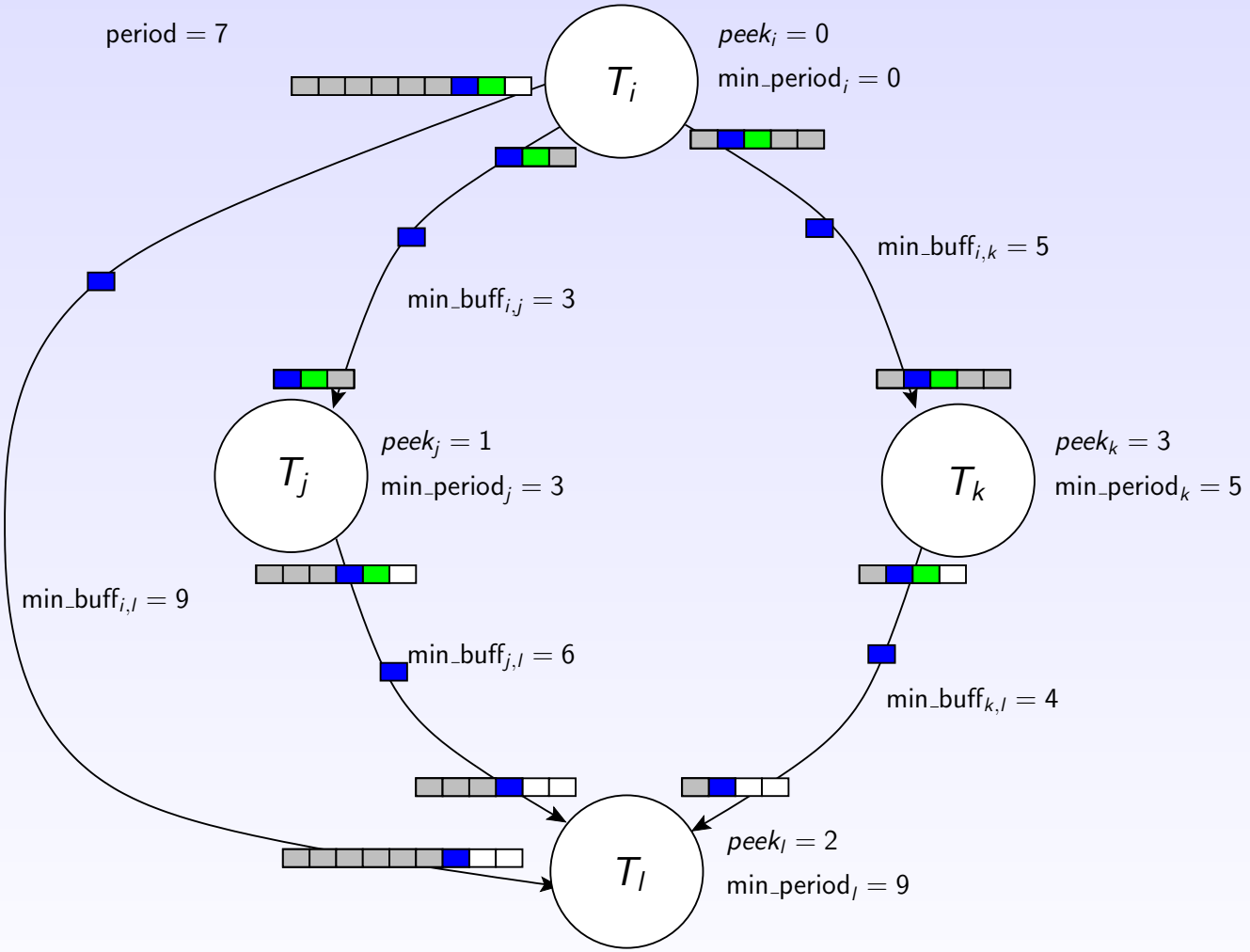
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



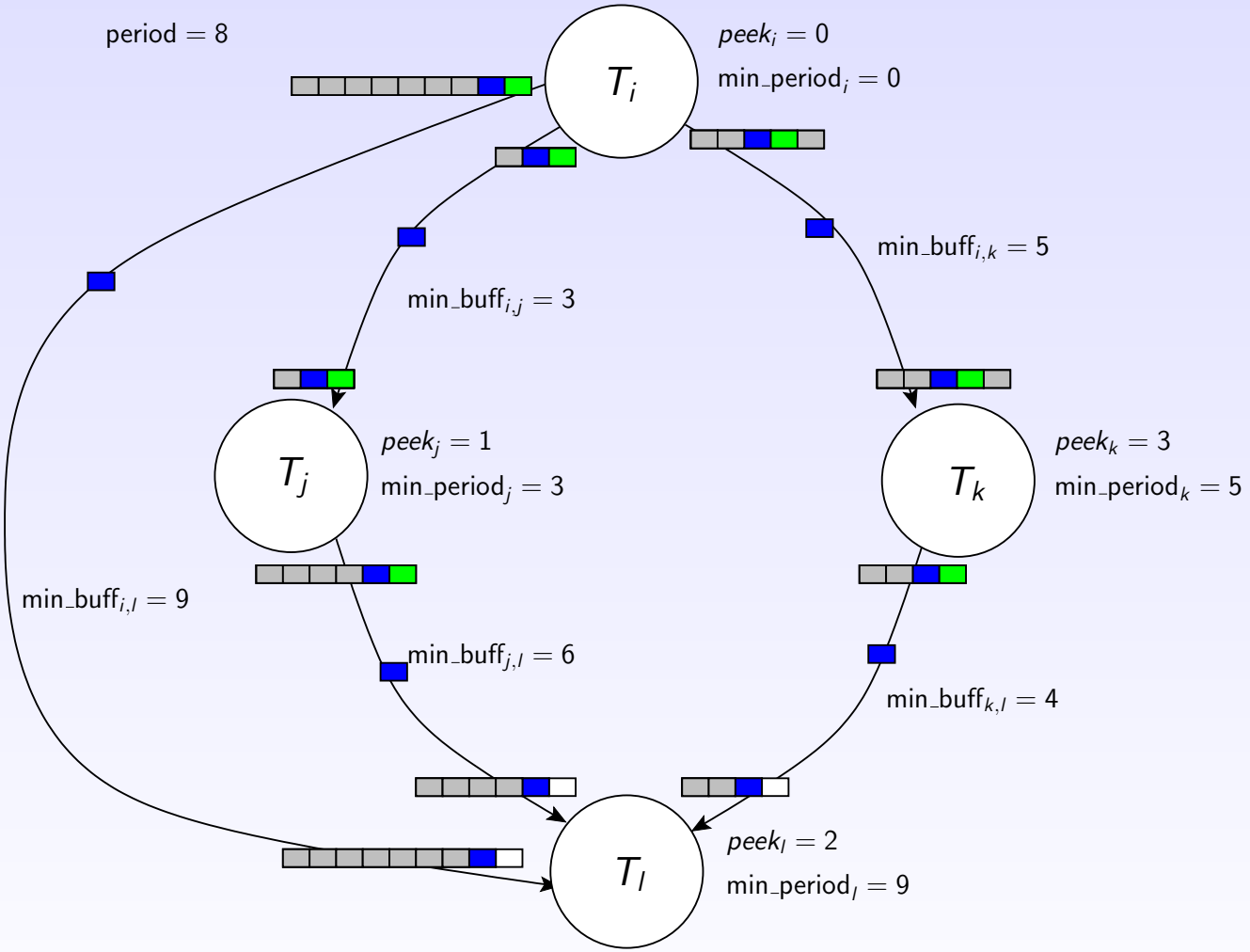
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



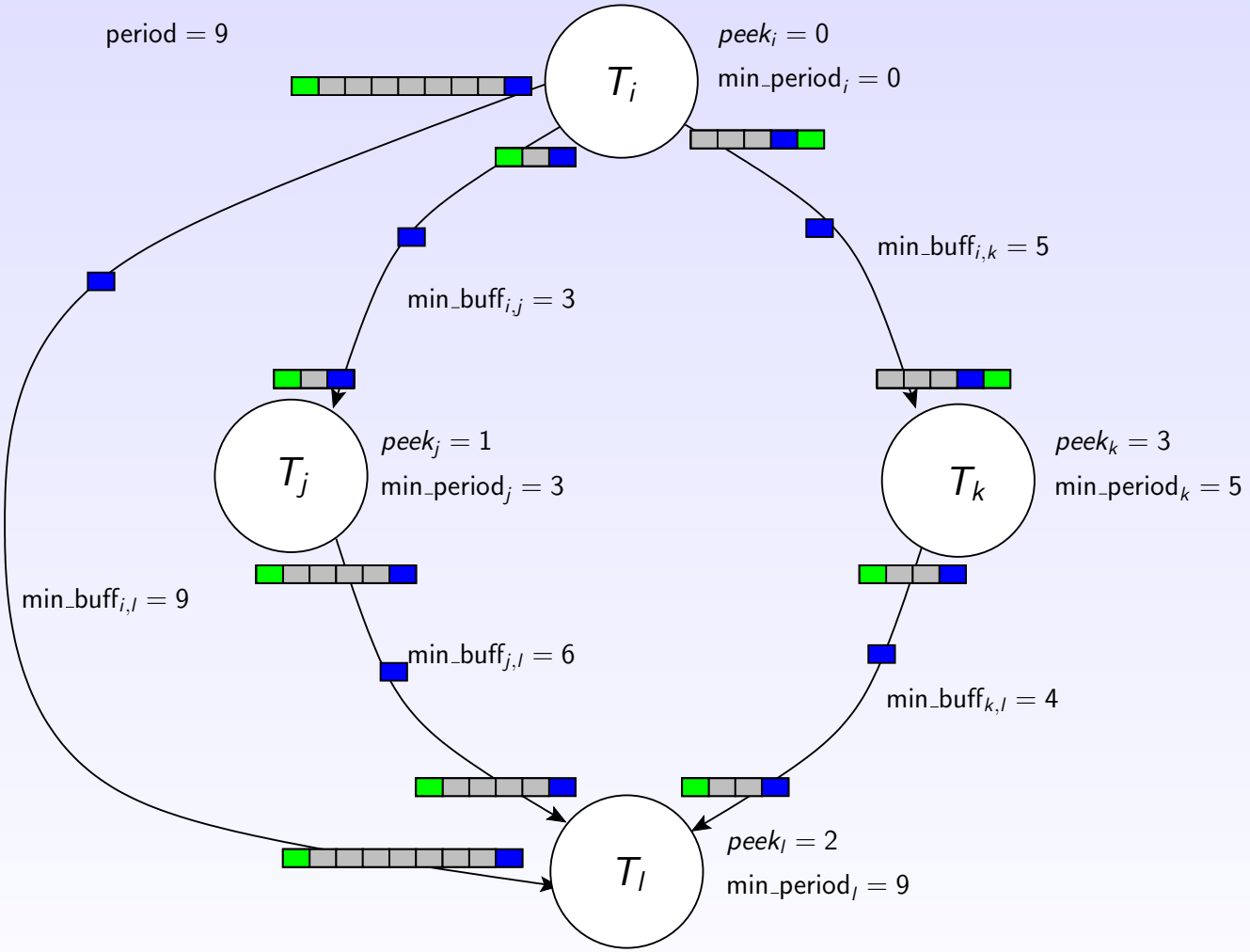
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



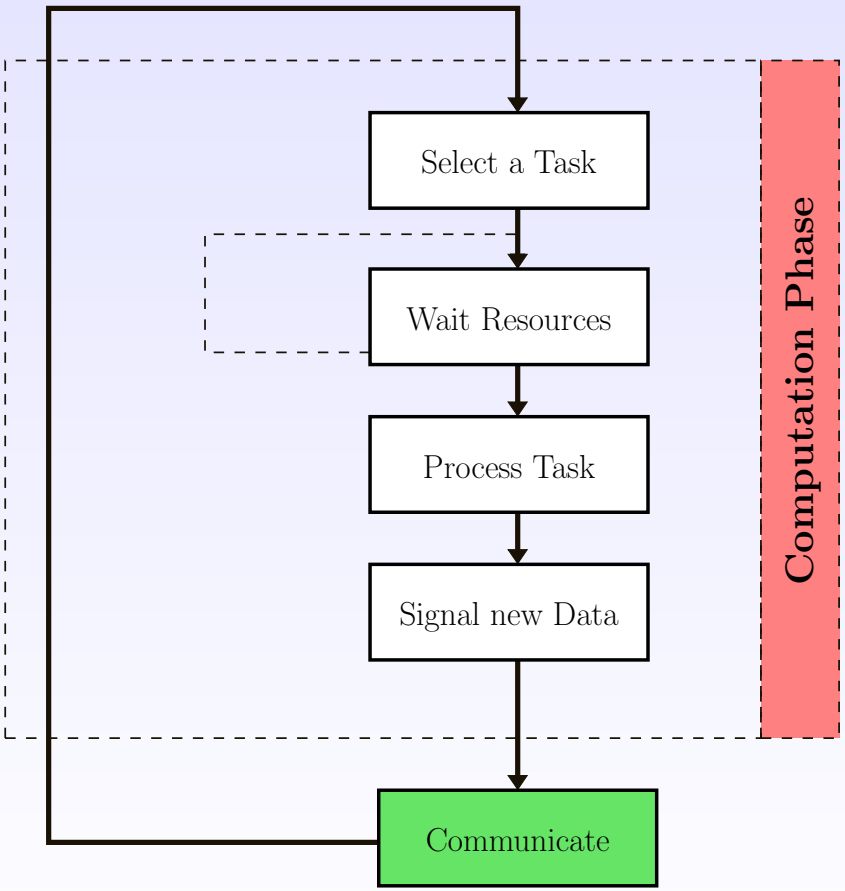
Preprocessing of the schedule

Main Objective: Compute minimal starting period and buffer sizes.



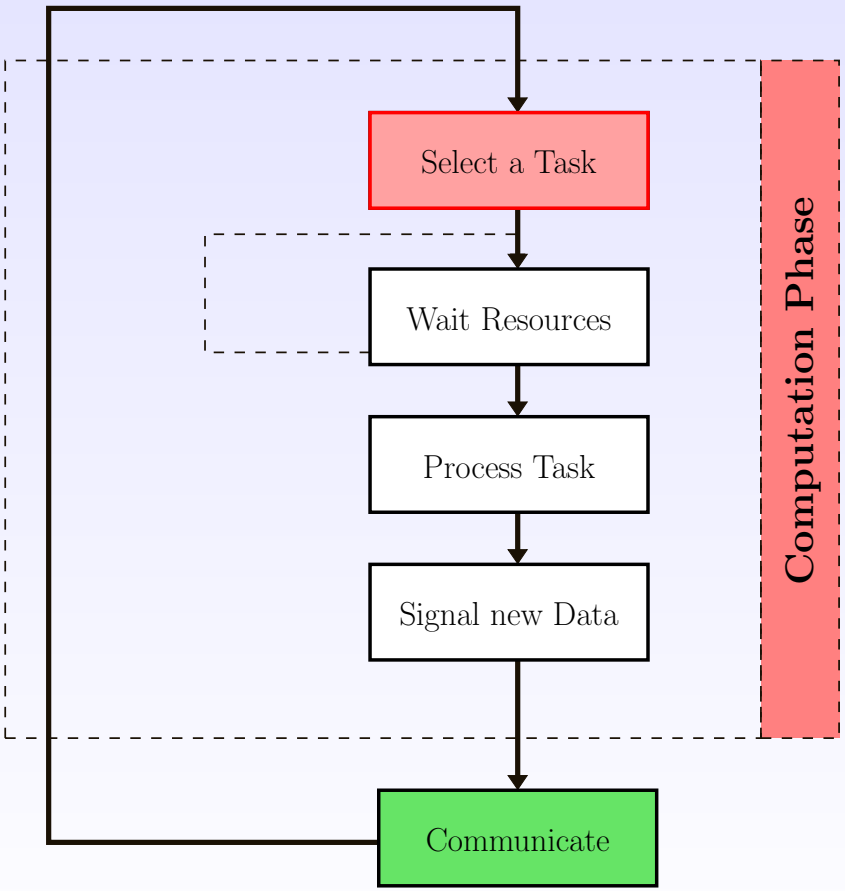
State machine of the application

Two main phases: Computation and Communication



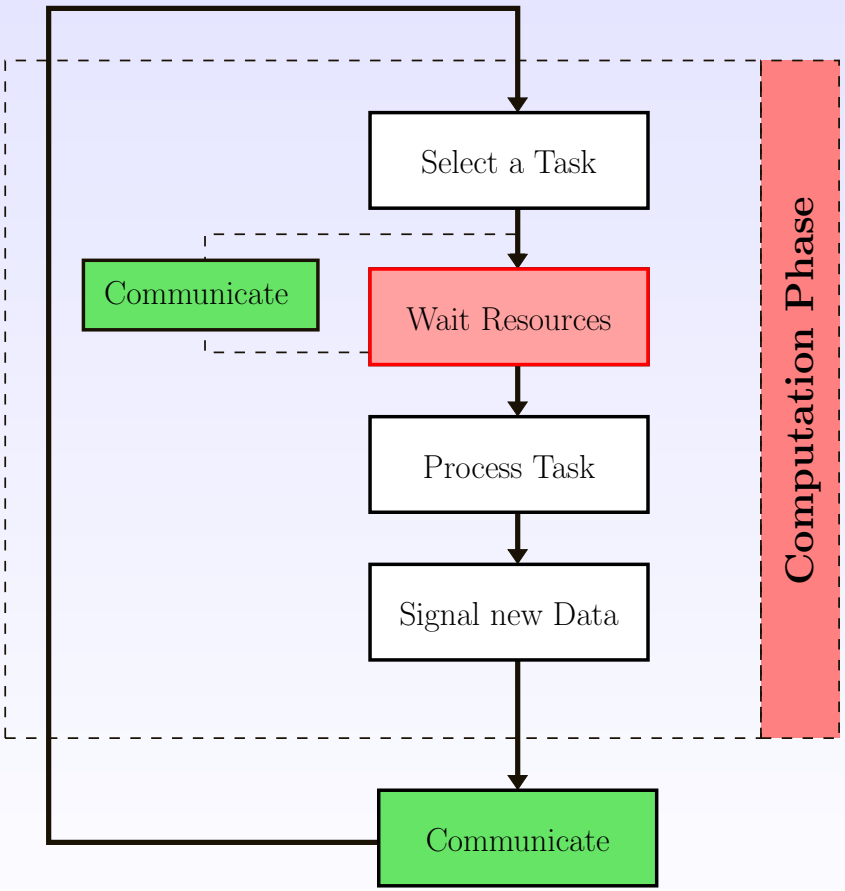
State machine of the application

Two main phases: Computation and Communication



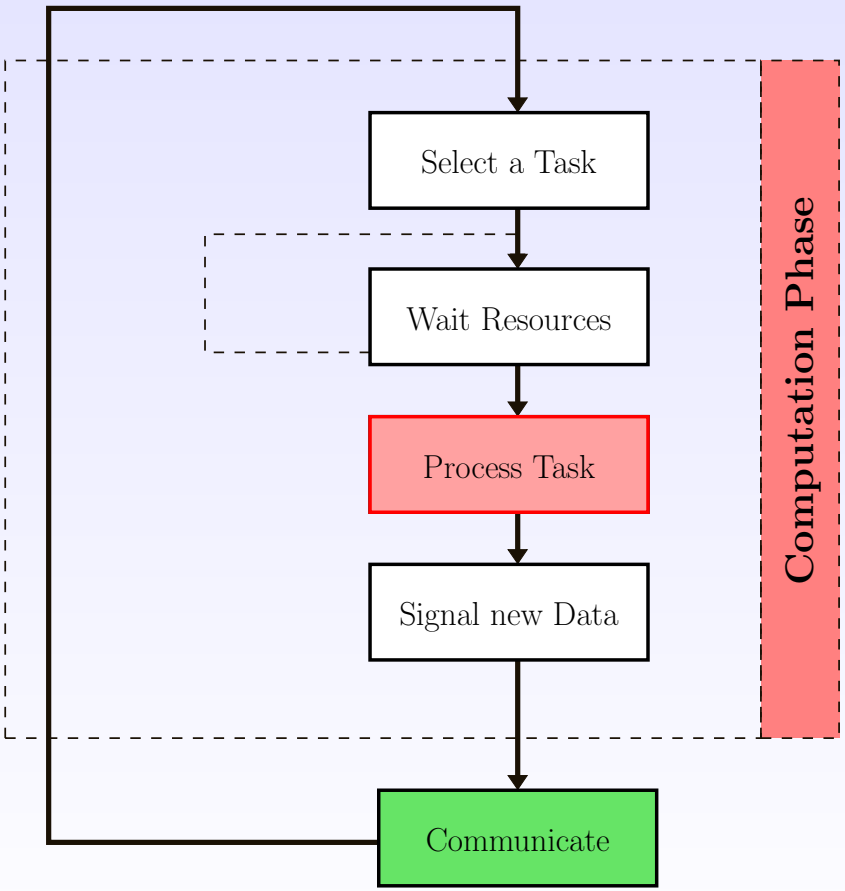
State machine of the application

Two main phases: Computation and Communication



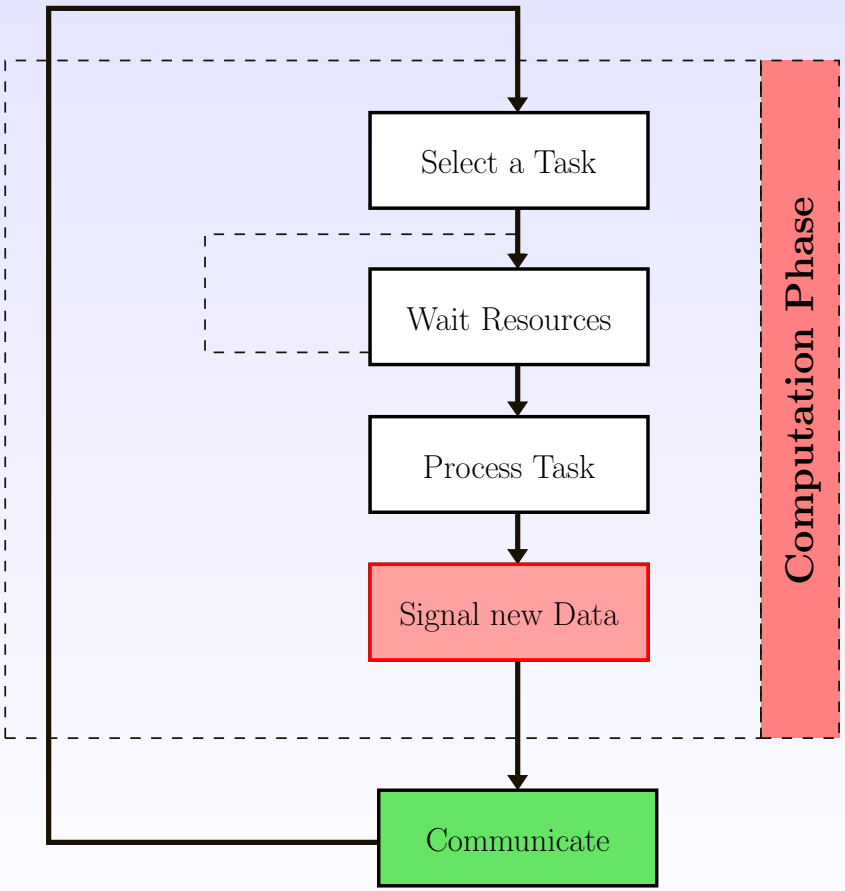
State machine of the application

Two main phases: Computation and Communication



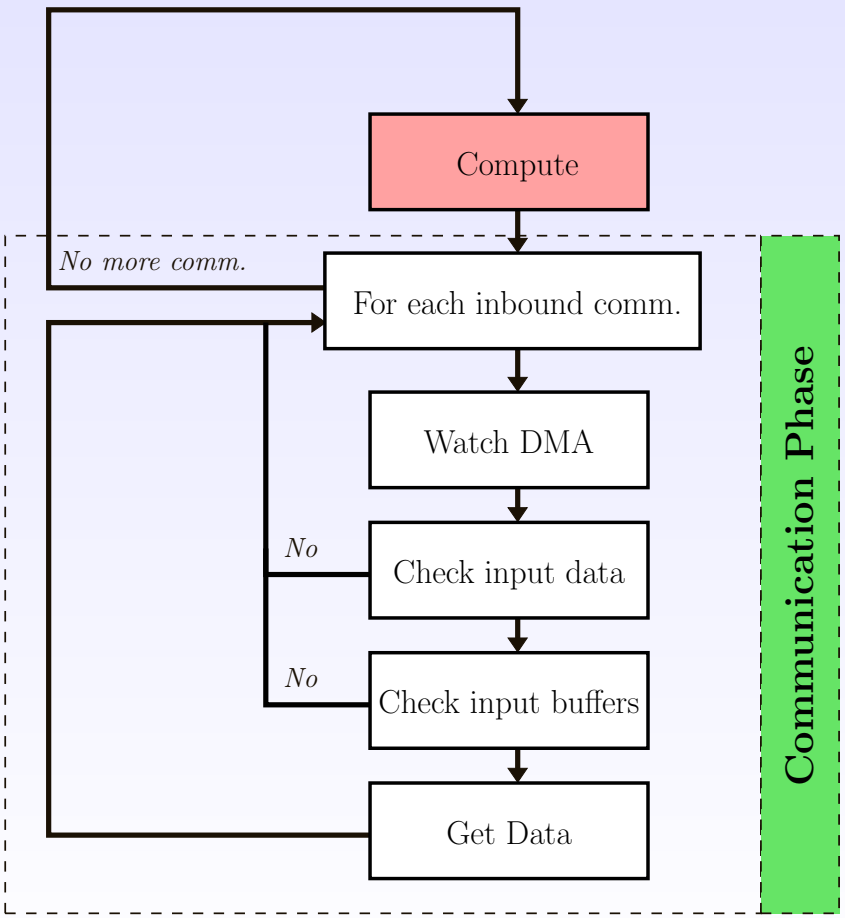
State machine of the application

Two main phases: Computation and Communication



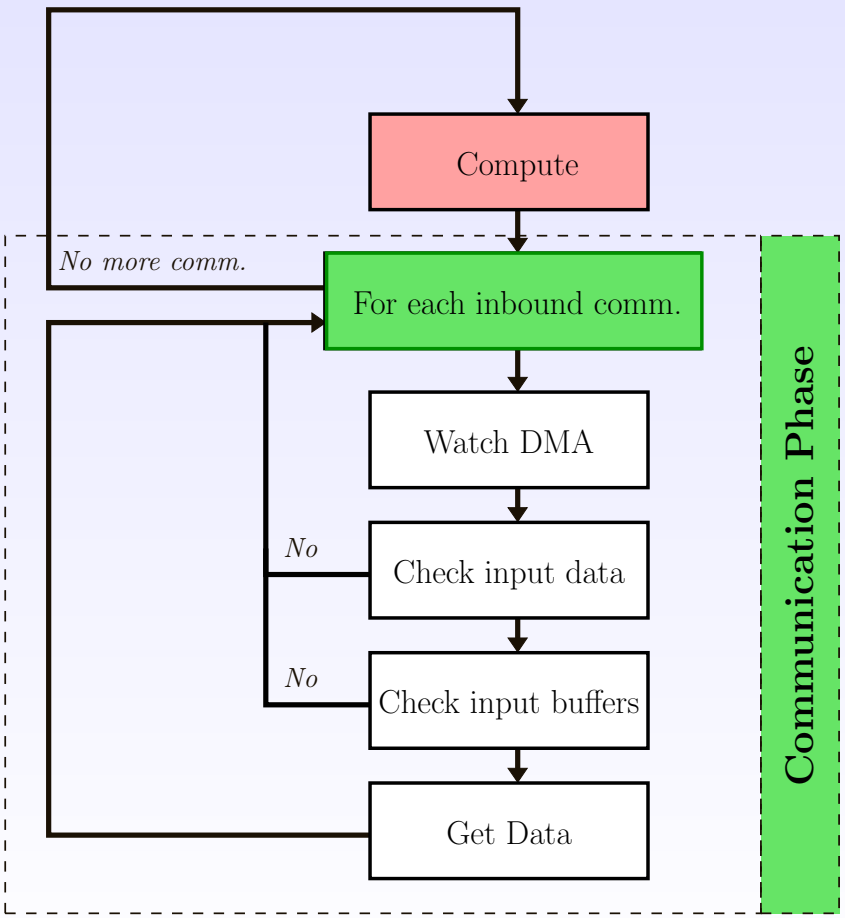
State machine of the application

Two main phases: Computation and Communication



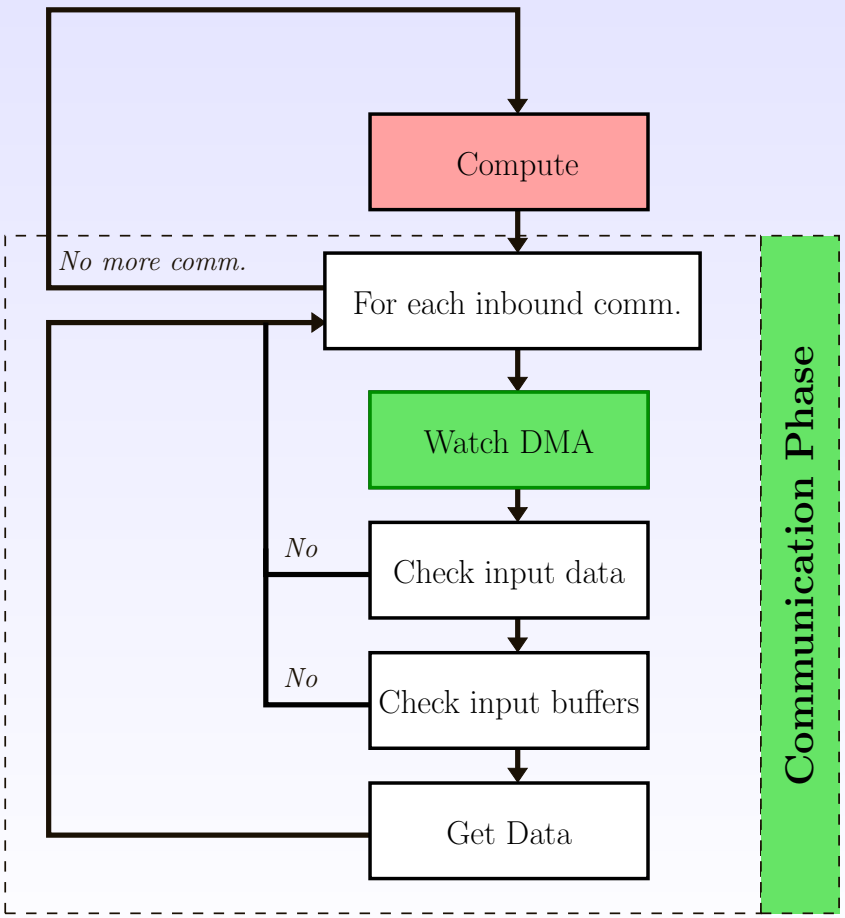
State machine of the application

Two main phases: Computation and Communication



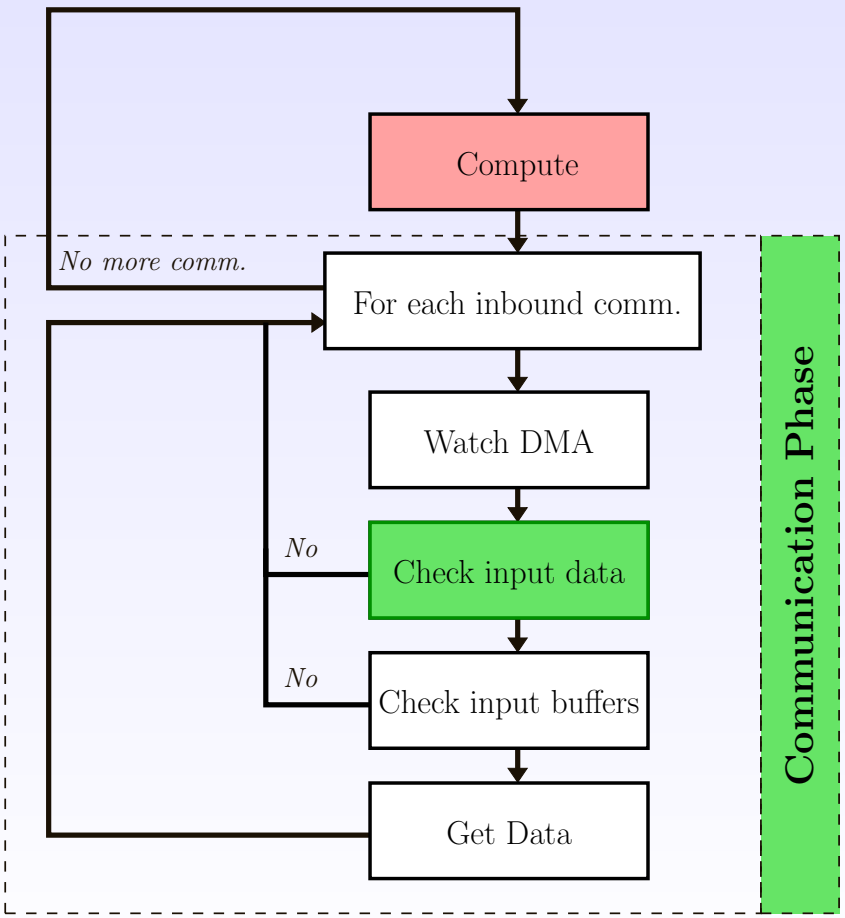
State machine of the application

Two main phases: Computation and Communication



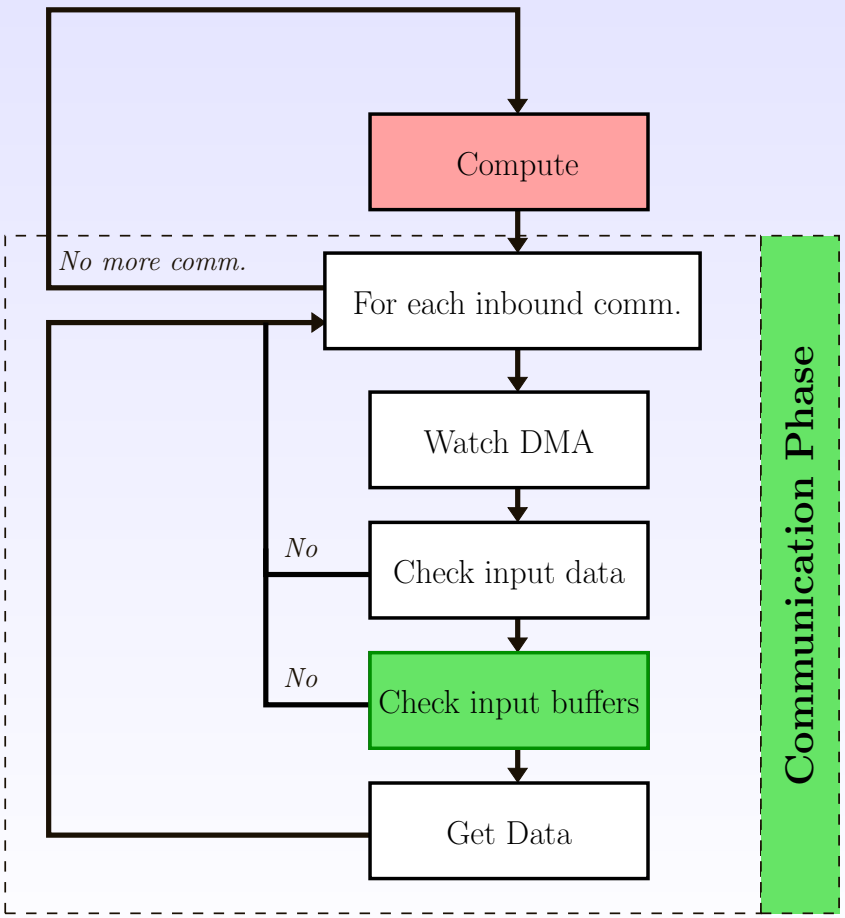
State machine of the application

Two main phases: Computation and Communication



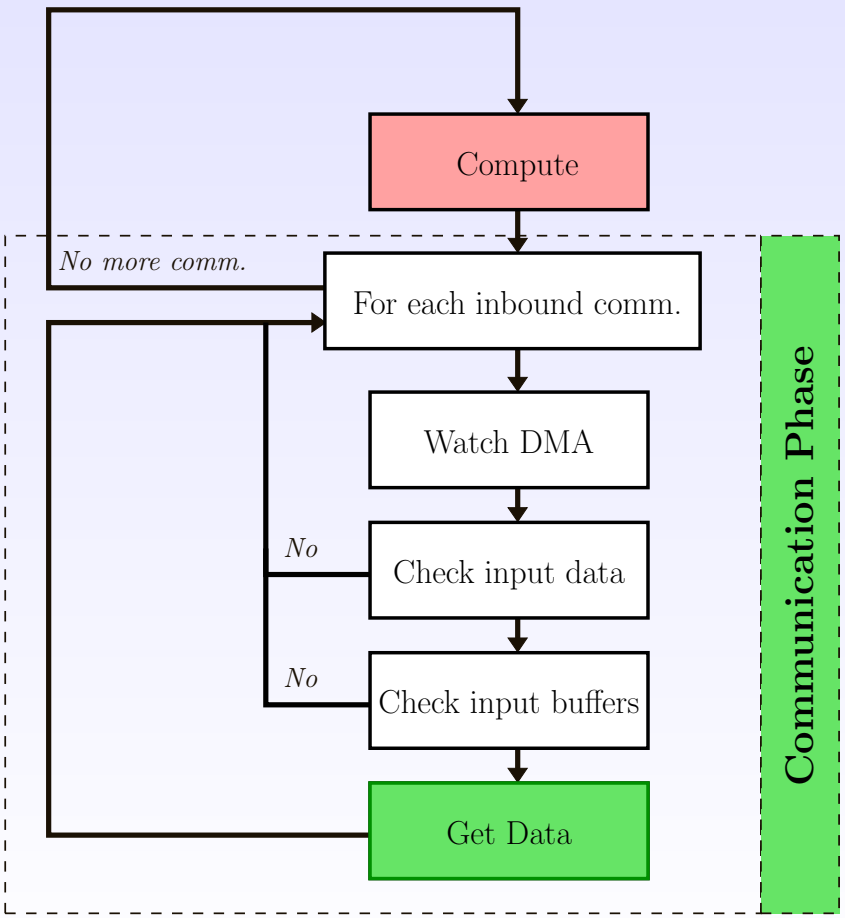
State machine of the application

Two main phases: Computation and Communication

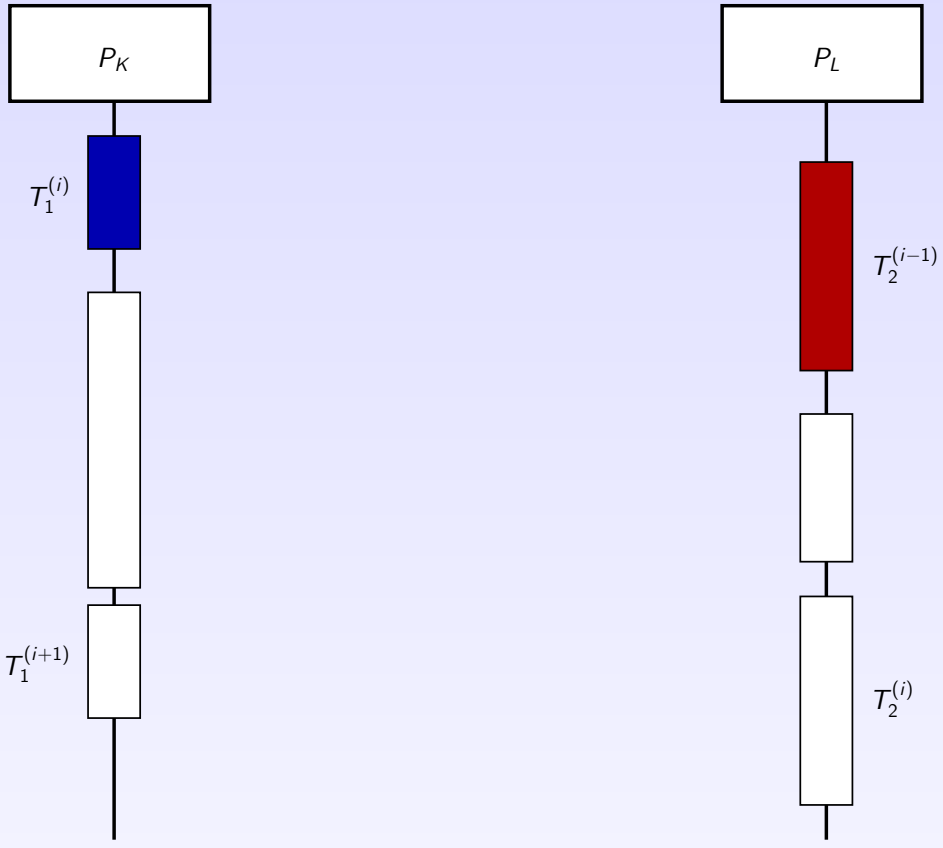


State machine of the application

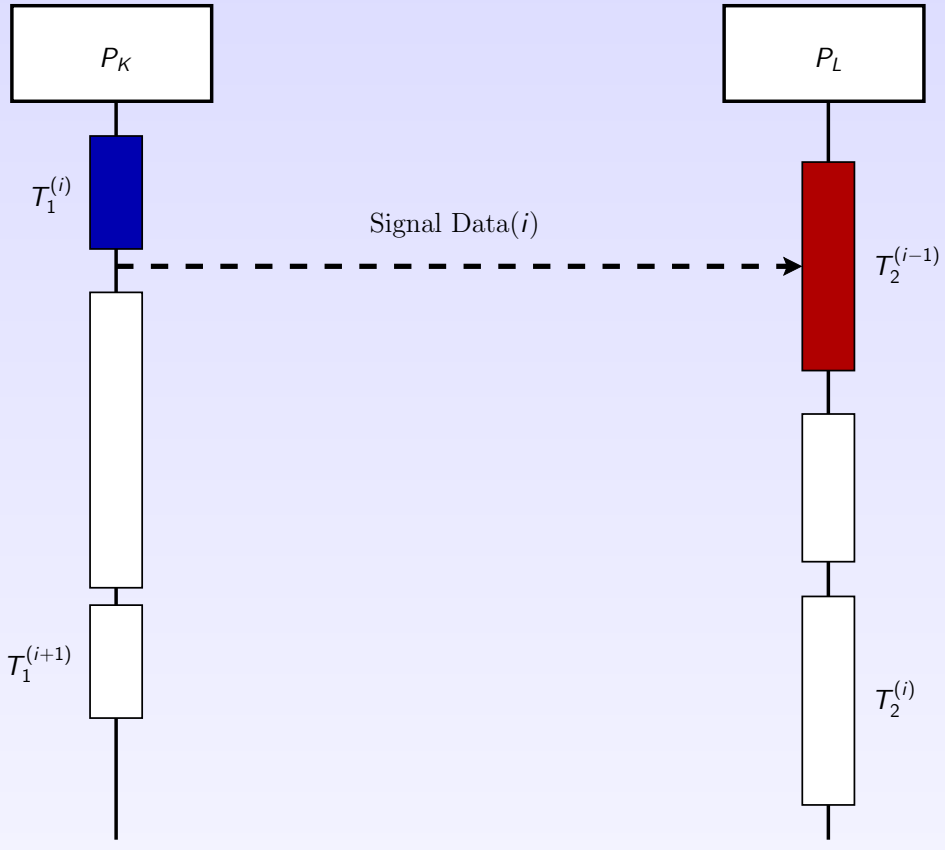
Two main phases: Computation and Communication



Communication between processors



Communication between processors

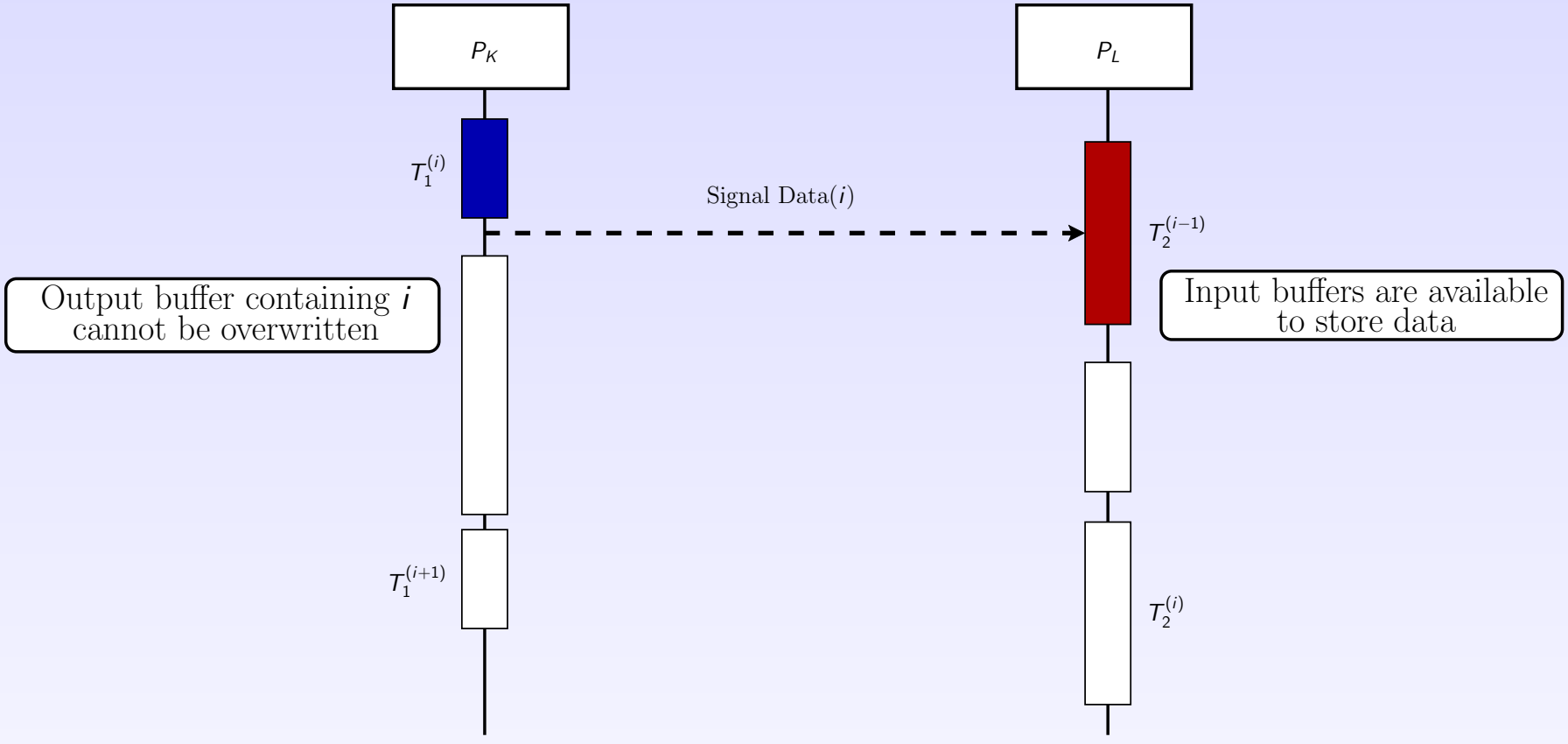


mfc_putb for SPEs' outbound communications.

spe_mfcio_getb for PPEs' outbound communications to SPEs.

memcpy for PPEs' outbound communications to main memory.

Communication between processors

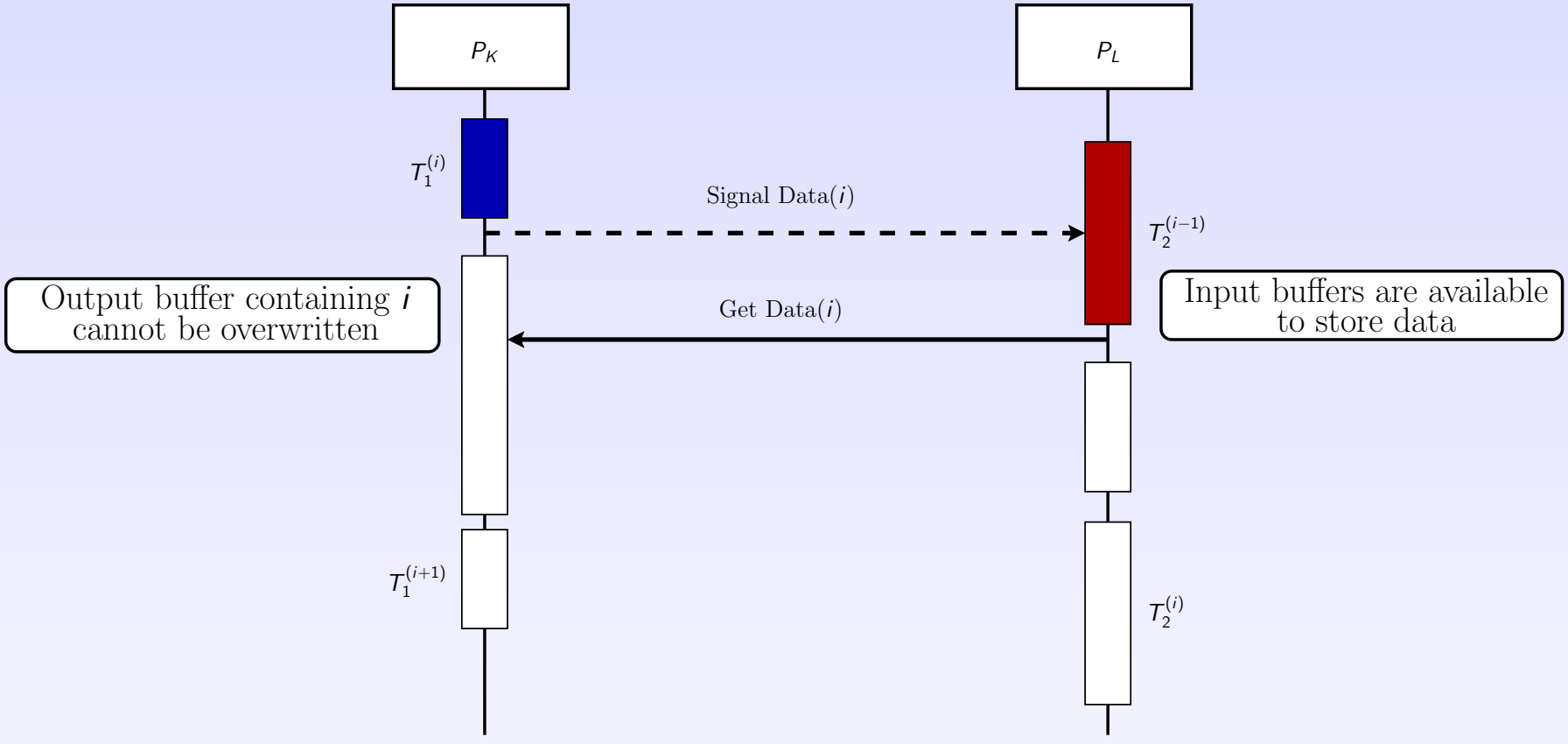


mfc_putb for SPEs' outbound communications.

spe_mfcio_getb for PPEs' outbound communications to SPEs.

memcpy for PPEs' outbound communications to main memory.

Communication between processors

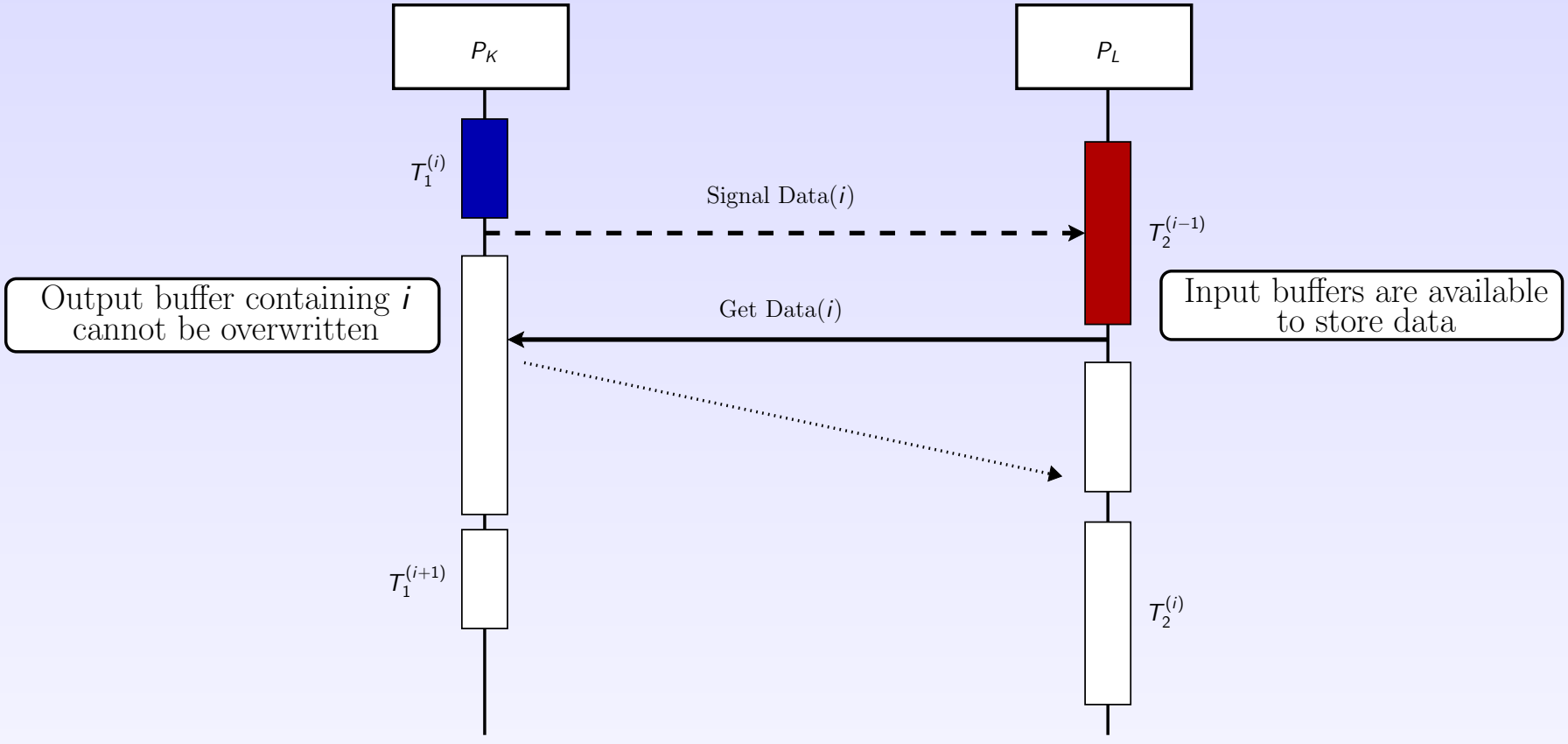


mfc_get for SPEs' inbound communications.

spe_mfcio_put for PPEs' inbound communications from SPEs.

memcpy for PPEs' inbound communications from main memory.

Communication between processors

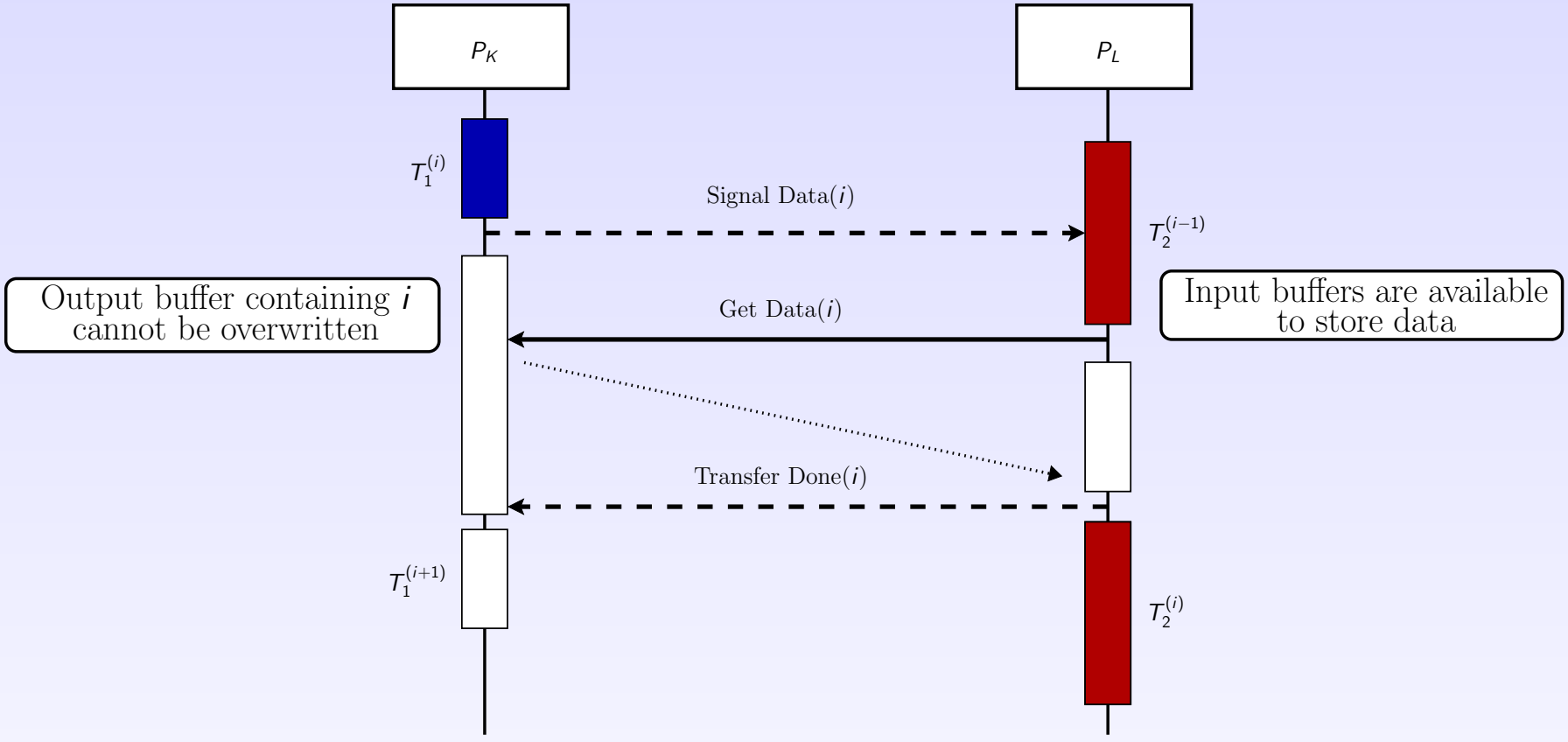


mfc_get for SPEs' inbound communications.

spe_mfcio_put for PPEs' inbound communications from SPEs.

memcpy for PPEs' inbound communications from main memory.

Communication between processors

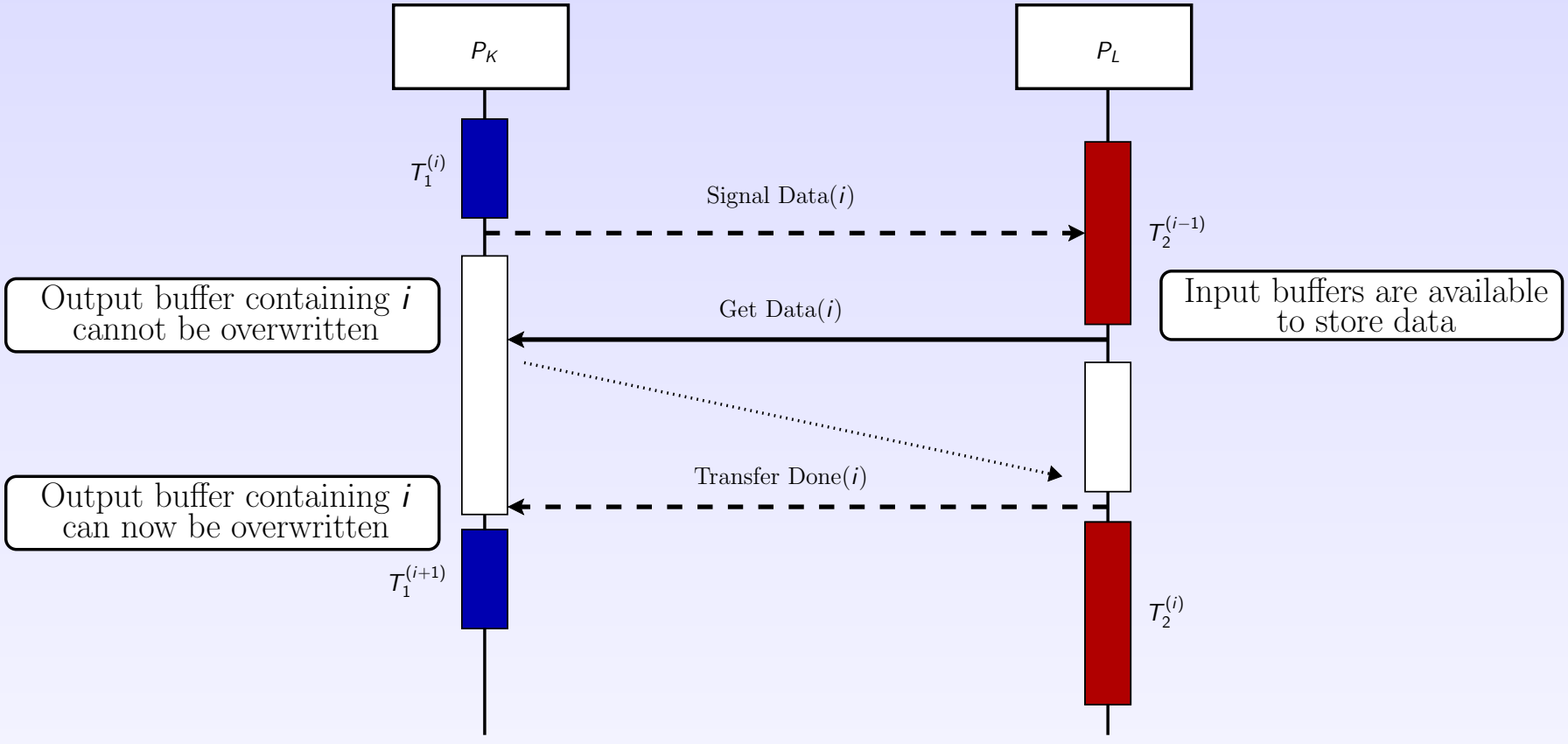


mfc_putb for SPEs' acknowledgements.

spe_mfcio_getb for PPEs' acknowledgements to SPEs.

Self acknowledgement of PPEs' transfers from main memory.

Communication between processors

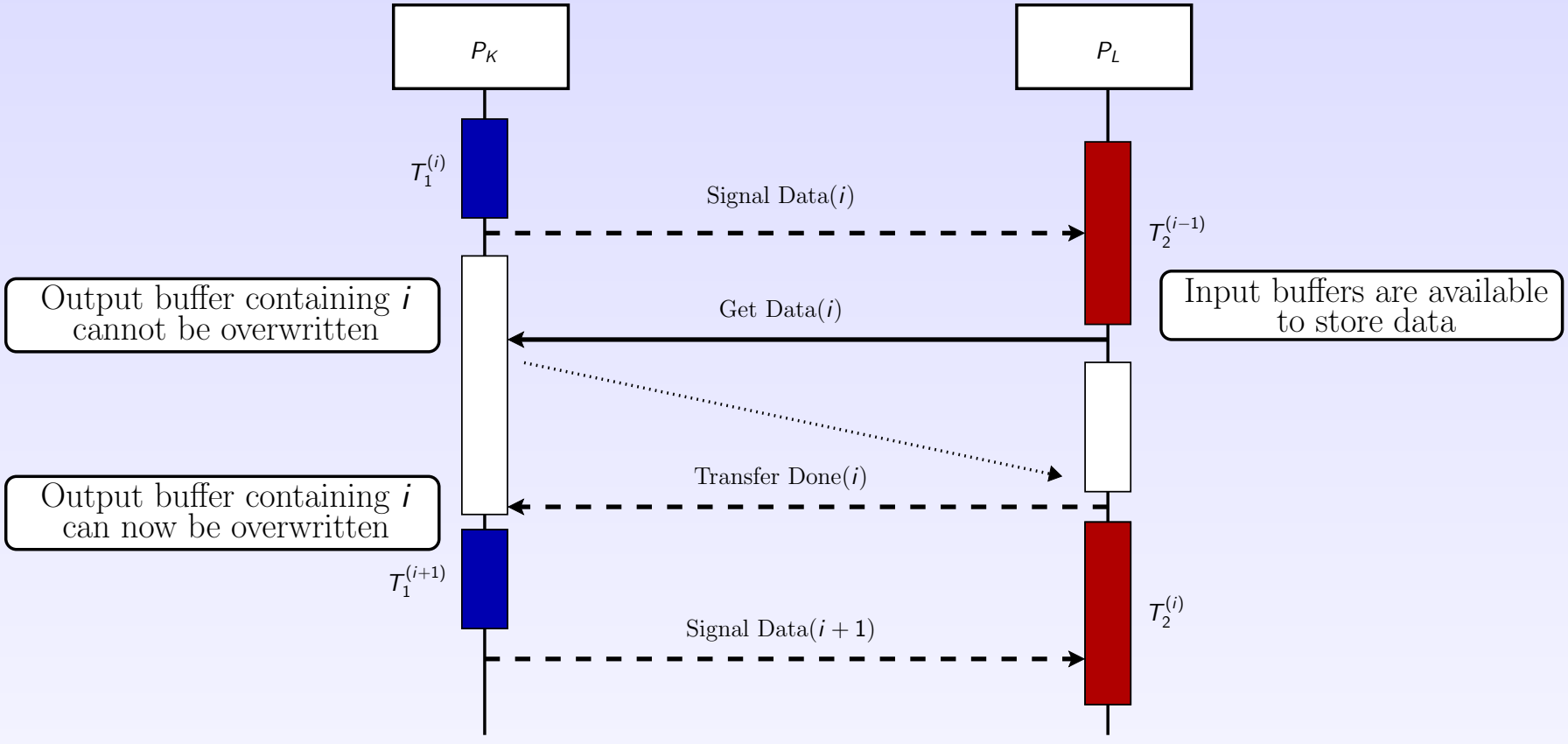


mfc_putb for SPEs' acknowledgements.

spe_mfcio_getb for PPEs' acknowledgements to SPEs.

Self acknowledgement of PPEs' transfers from main memory.

Communication between processors



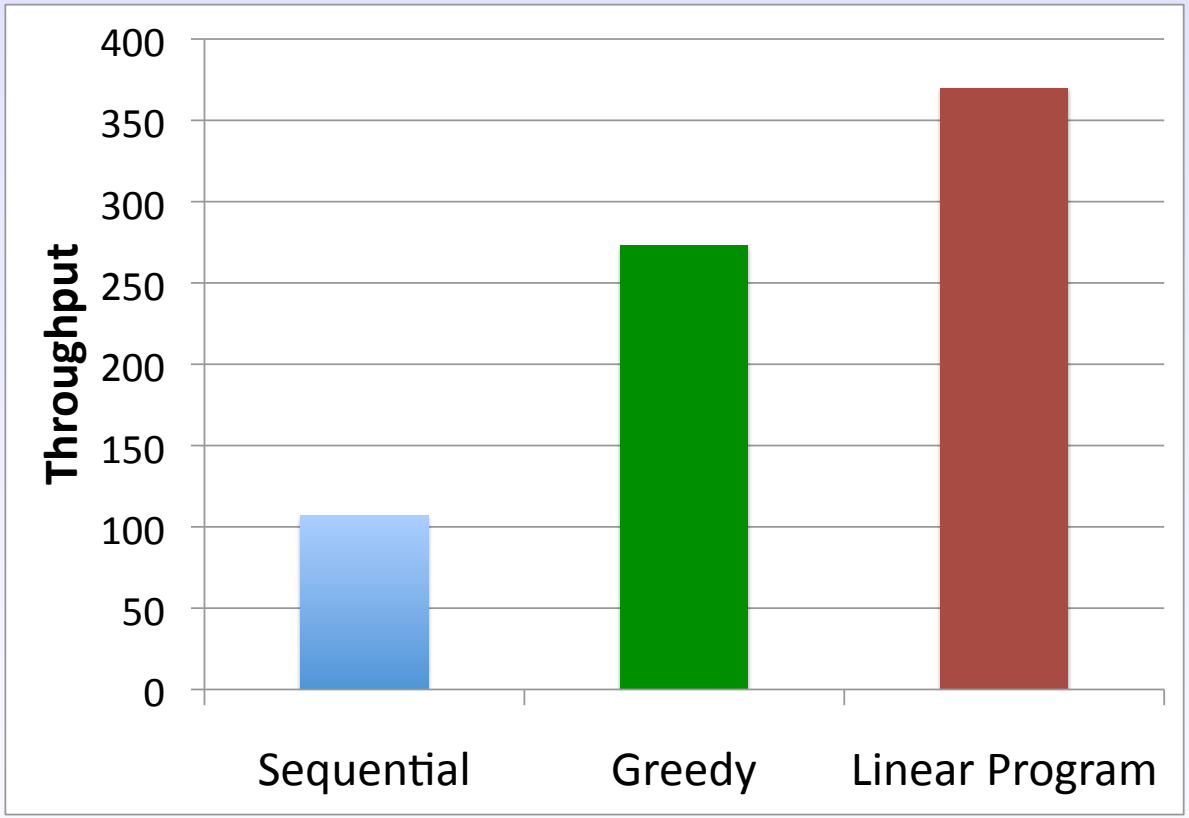
mfc_putb for SPEs' acknowledgements.

spe_mfcio_getb for PPEs' acknowledgements to SPEs.

Self acknowledgement of PPEs' transfers from main memory.

Preliminary results

We outperform both greedy heuristic and sequential version.



Results are obtained over 70000 periods

Outline

Feedback on Cell programming

- Multilevel heterogeneity:
  - 32 bits SPEs vs 64 bits PPE architectures
  - Different communication mechanism and constraints
- Non trivial initialization phase
  - Varying data structure sizes (32/64bits)
  - Runtime memory allocation

On-going and Future work

- Various code optimizations
  - SIMD code for SPEs
  - Reduce control overhead
- Better communication modeling
  - Is linear cost model relevant ?
  - Contention on concurrent DMA operations ?
- Larger platforms
  - Using multiple CELL processors
  - CELL + other type of processing units ?
  - Work on communication modeling
- Design scheduling heuristics
  - MIP is costly