Online Scheduling of Moldable Task Graphs under Common Speedup Models

Lucas Perotin (speaker)[1] Thomas Verrecchia Padma Raghavan[2]

[1]LIG, Grenoble INP, France

[2]Vanderbilt University, USA

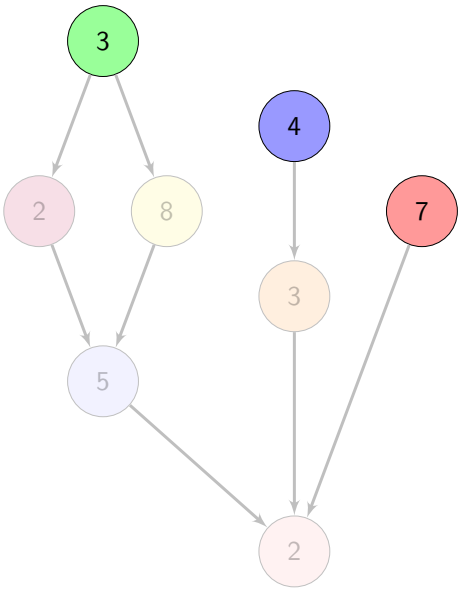
March 18, 2026

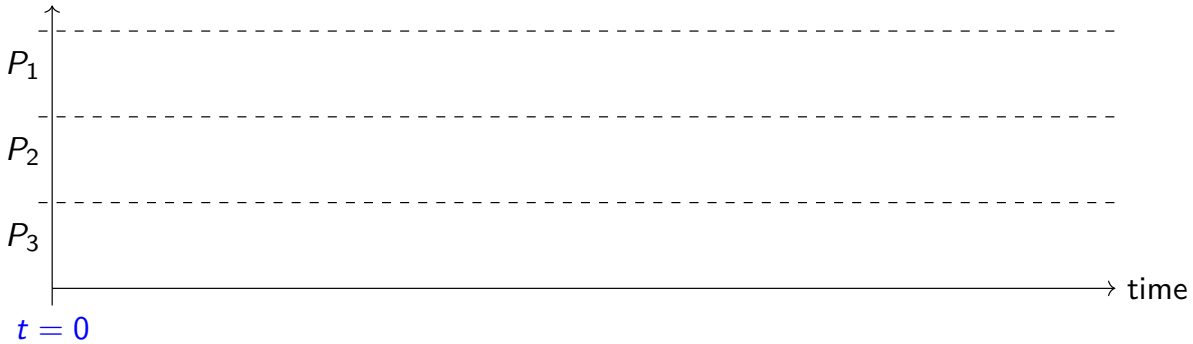
Introduction

- **Offline Scheduling vs. Online Scheduling.**
  - Offline: All tasks are known in advance,
    Goal: Find approximation ratios on polynomial algorithms.
  - Online: Tasks released on the fly,
    Goal: Derive competitive ratios against an optimal offline scheduler.
- **Tasks type**
  - Rigid jobs: Processor allocation is fixed.
  - Moldable jobs: Processor allocation is decided by the system but cannot be changed.
  - Malleable jobs: Processor allocation can be dynamically changed.
- **Independent Tasks vs. Task Graphs.**
  - Independent tasks: Tasks released and discovered on the fly,
  - Task graphs: Graph released at the start of execution,
    Tasks discovered when all predecessors are completed.

$\Rightarrow$ In this work, we focus on *online moldable task graphs.*

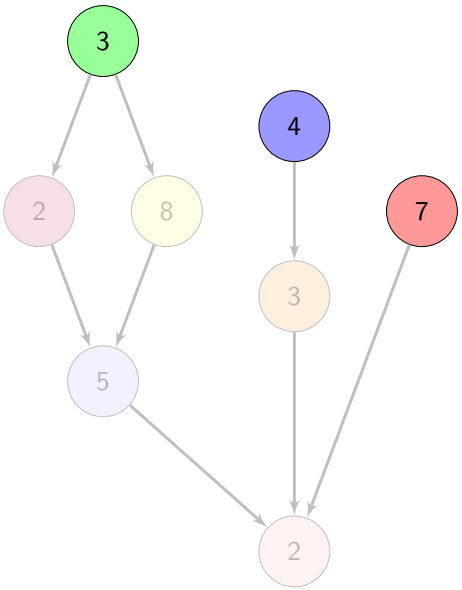
Scheduling Online Moldable Task Graphs

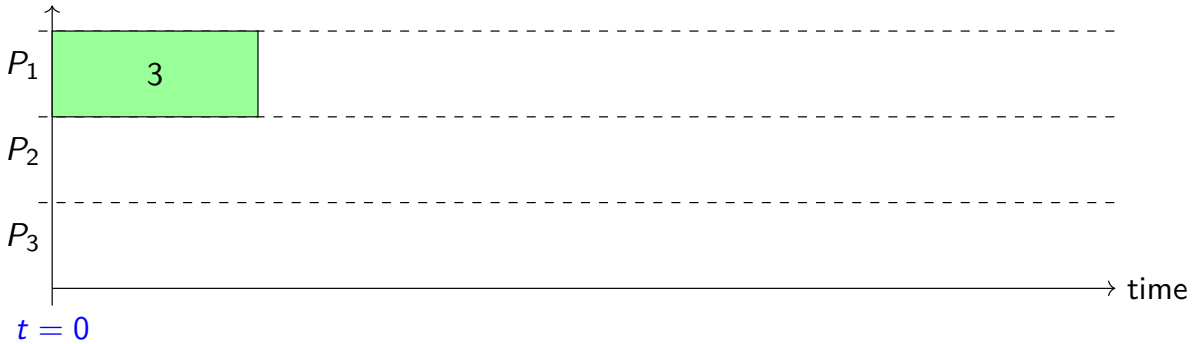




- Graph of n moldable tasks with precedence constraints. Each task is released when all predecessors are completed,
- P processors to process the tasks,
- Each task j's execution time $t_j(p_j)$ depends on the number of processors allocated to it and is considered known when the task is released,
- Area is $a_j(p_j) = p_j \times t_j(p_j)$.

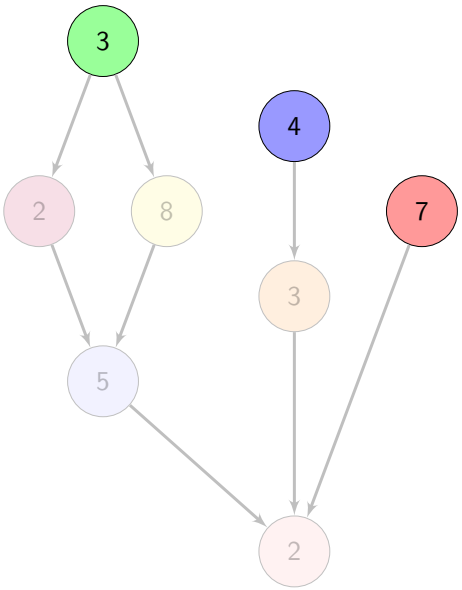
Scheduling Online Moldable Task Graphs

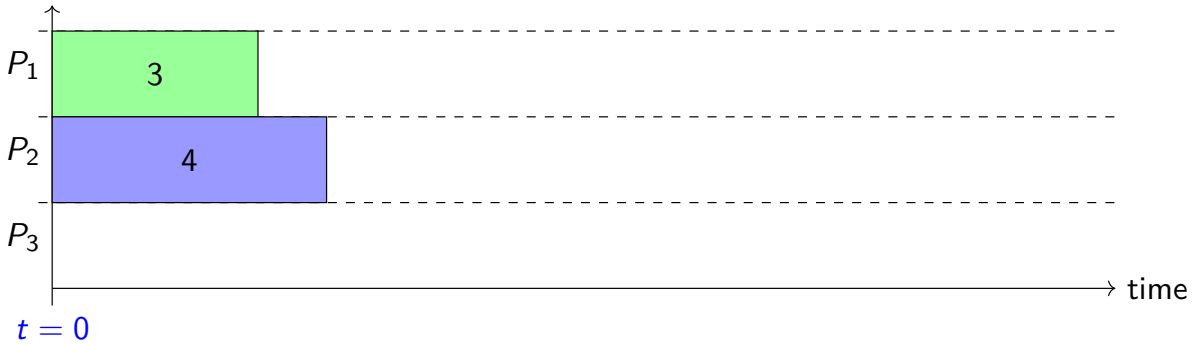




- Graph of n moldable tasks with precedence constraints. Each task is released when all predecessors are completed,
- P processors to process the tasks,
- Each task j's execution time $t_j(p_j)$ depends on the number of processors allocated to it and is considered known when the task is released,
- Area is $a_j(p_j) = p_j \times t_j(p_j)$.

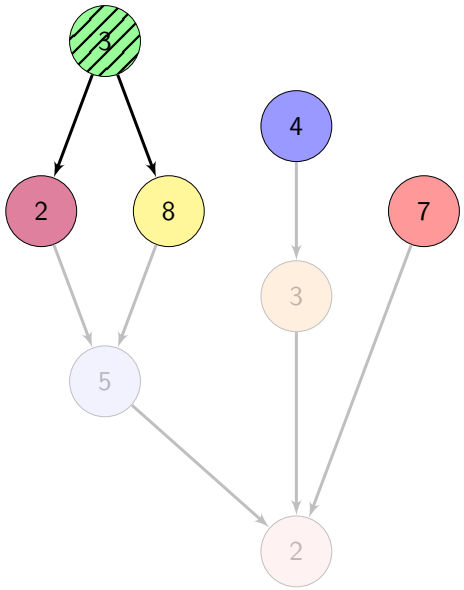
Scheduling Online Moldable Task Graphs

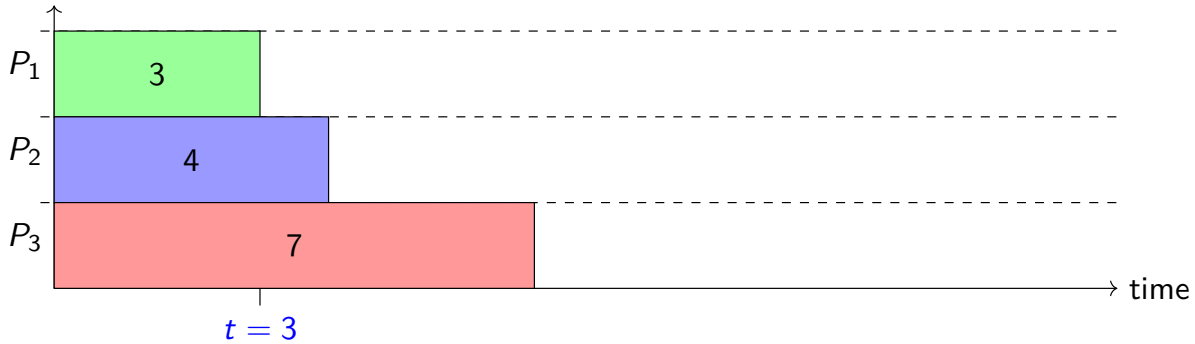




- Graph of n moldable tasks with precedence constraints. Each task is released when all predecessors are completed,
- P processors to process the tasks,
- Each task j's execution time $t_j(p_j)$ depends on the number of processors allocated to it and is considered known when the task is released,
- Area is $a_j(p_j) = p_j \times t_j(p_j)$.

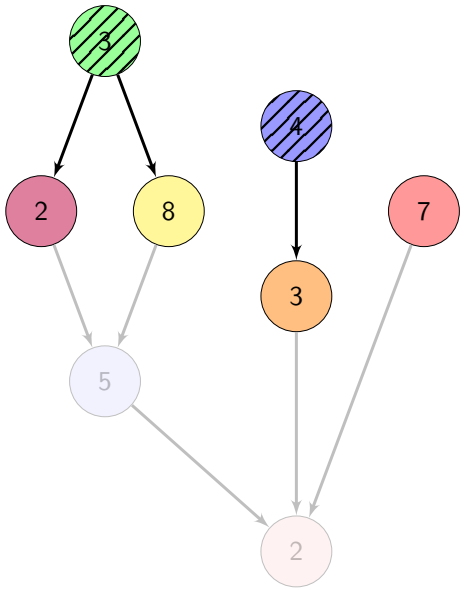
Scheduling Online Moldable Task Graphs

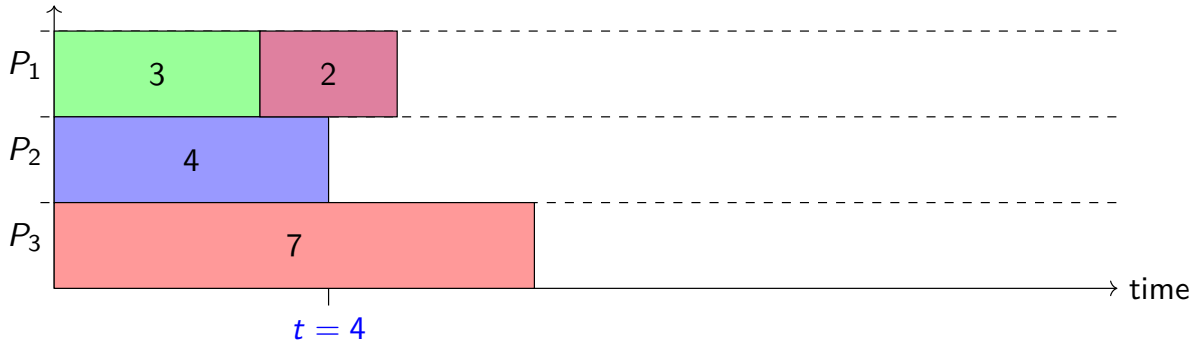




- Graph of n moldable tasks with precedence constraints. Each task is released when all predecessors are completed,
- P processors to process the tasks,
- Each task j's execution time $t_j(p_j)$ depends on the number of processors allocated to it and is considered known when the task is released,
- Area is $a_j(p_j) = p_j \times t_j(p_j)$.

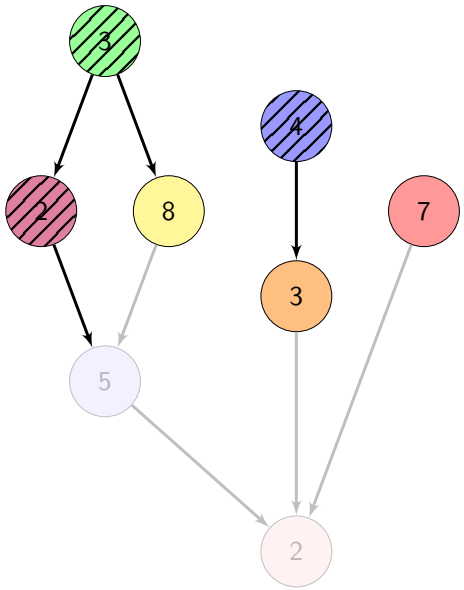
Scheduling Online Moldable Task Graphs

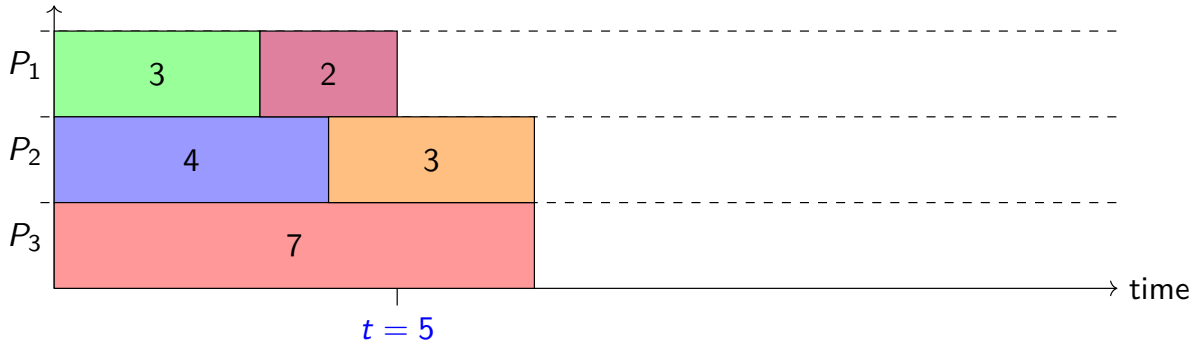




- Graph of n moldable tasks with precedence constraints. Each task is released when all predecessors are completed,
- P processors to process the tasks,
- Each task j's execution time $t_j(p_j)$ depends on the number of processors allocated to it and is considered known when the task is released,
- Area is $a_j(p_j) = p_j \times t_j(p_j)$.

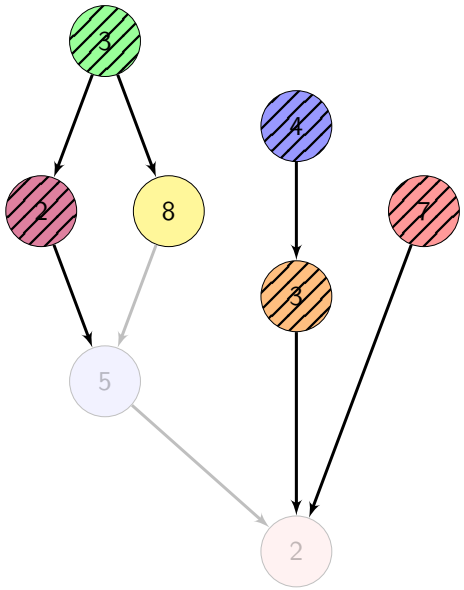
Scheduling Online Moldable Task Graphs

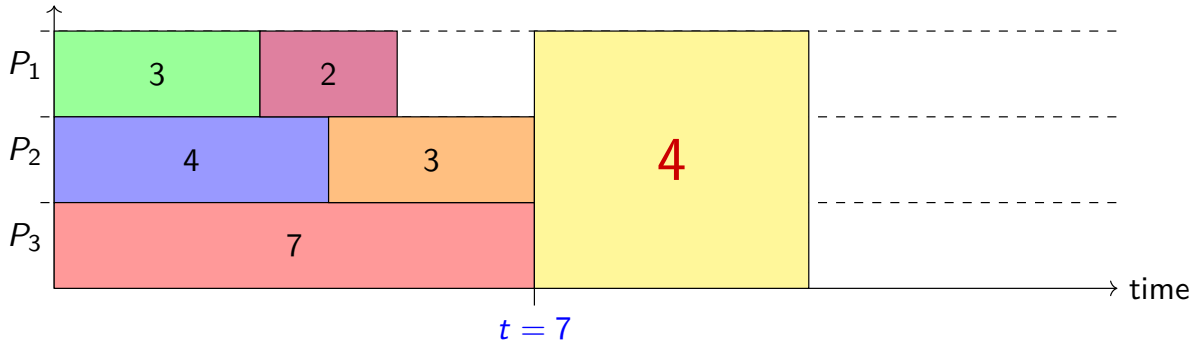




- Graph of n moldable tasks with precedence constraints. Each task is released when all predecessors are completed,
- P processors to process the tasks,
- Each task j's execution time $t_j(p_j)$ depends on the number of processors allocated to it and is considered known when the task is released,
- Area is $a_j(p_j) = p_j \times t_j(p_j)$.

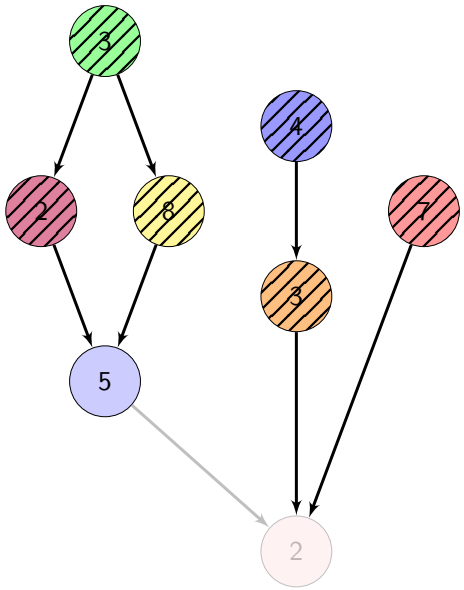
Scheduling Online Moldable Task Graphs

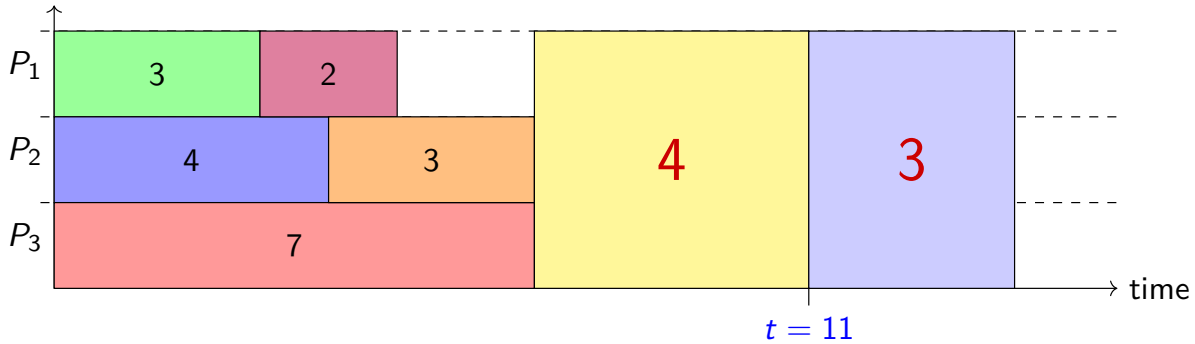




- Graph of n moldable tasks with precedence constraints. Each task is released when all predecessors are completed,
- P processors to process the tasks,
- Each task j's execution time $t_j(p_j)$ depends on the number of processors allocated to it and is considered known when the task is released,
- Area is $a_j(p_j) = p_j \times t_j(p_j)$.

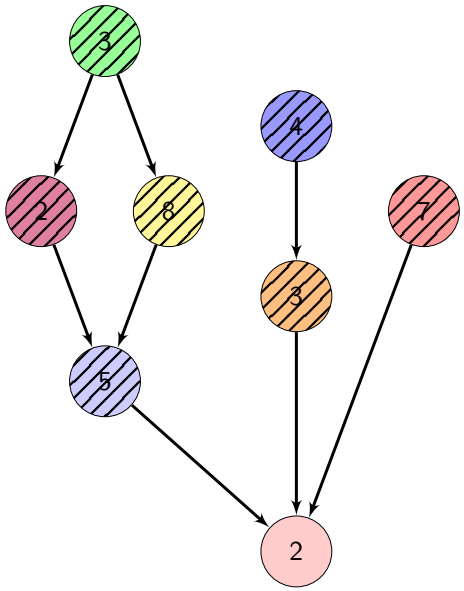
Scheduling Online Moldable Task Graphs

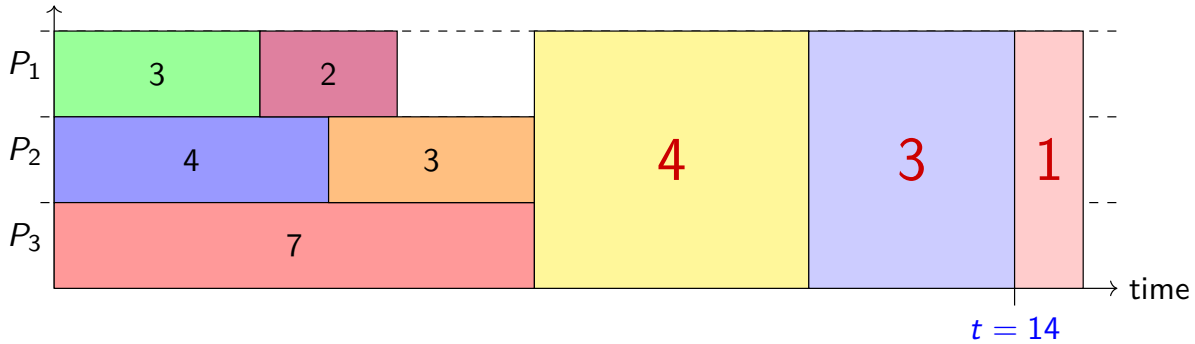




- Graph of n moldable tasks with precedence constraints. Each task is released when all predecessors are completed,
- P processors to process the tasks,
- Each task j's execution time $t_j(p_j)$ depends on the number of processors allocated to it and is considered known when the task is released,
- Area is $a_j(p_j) = p_j \times t_j(p_j)$.

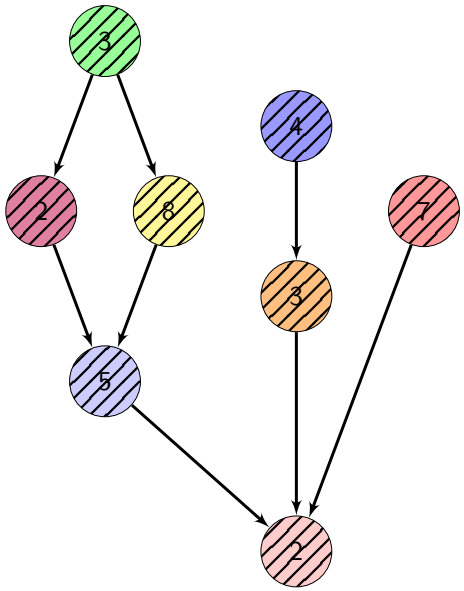
Scheduling Online Moldable Task Graphs

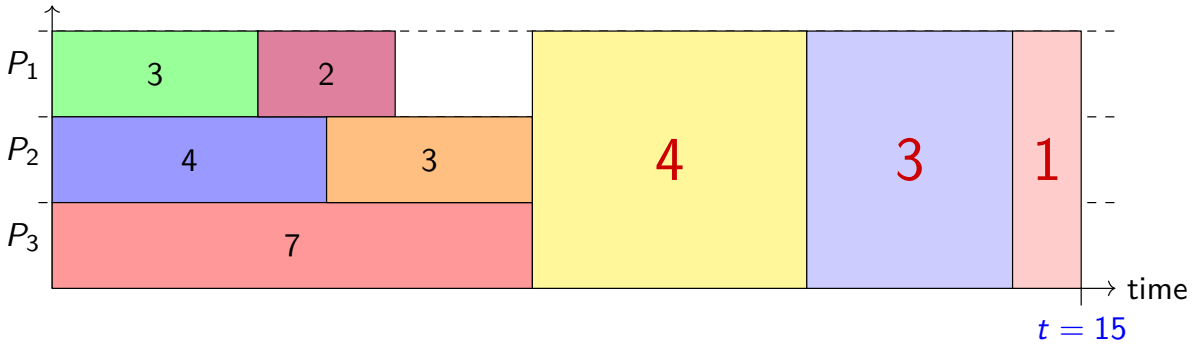




- Graph of n moldable tasks with precedence constraints. Each task is released when all predecessors are completed,
- P processors to process the tasks,
- Each task j's execution time $t_j(p_j)$ depends on the number of processors allocated to it and is considered known when the task is released,
- Area is $a_j(p_j) = p_j \times t_j(p_j)$.

Scheduling Online Moldable Task Graphs





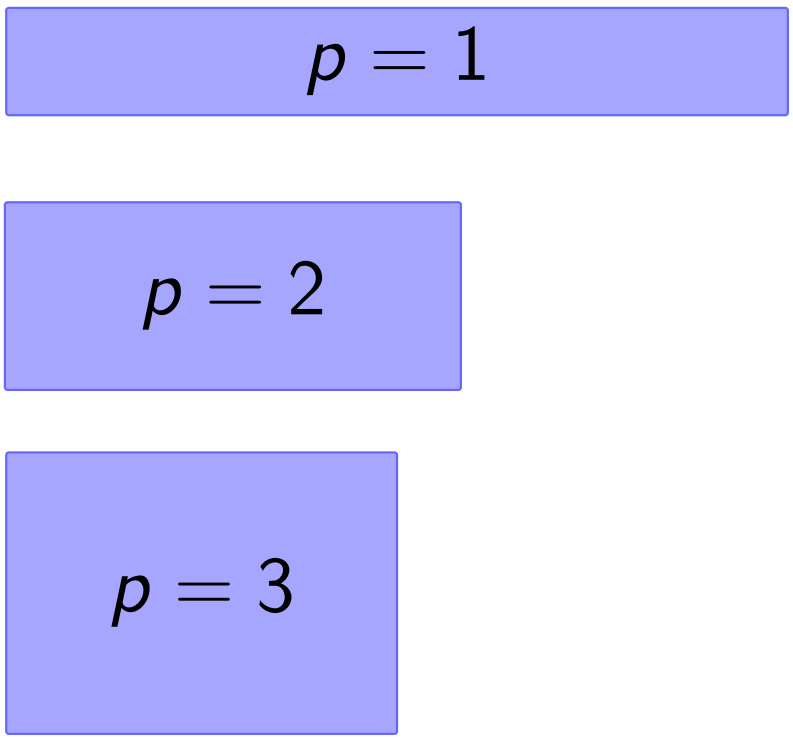
- Graph of n moldable tasks with precedence constraints. Each task is released when all predecessors are completed,
- P processors to process the tasks,
- Each task j's execution time $t_j(p_j)$ depends on the number of processors allocated to it and is considered known when the task is released,
- Area is $a_j(p_j) = p_j \times t_j(p_j)$.

Speedup models

Communication model: fully parallelizable with an extra cost c for communication between processors

	Time	**Area**
$p = 1$	12	12
$p = 2$	$12/2 + 1 = 7$	14
$p = 3$	$12/3 + 2 = 6$	18

$$t(p) = \frac{w}{p} + c(p-1)$$

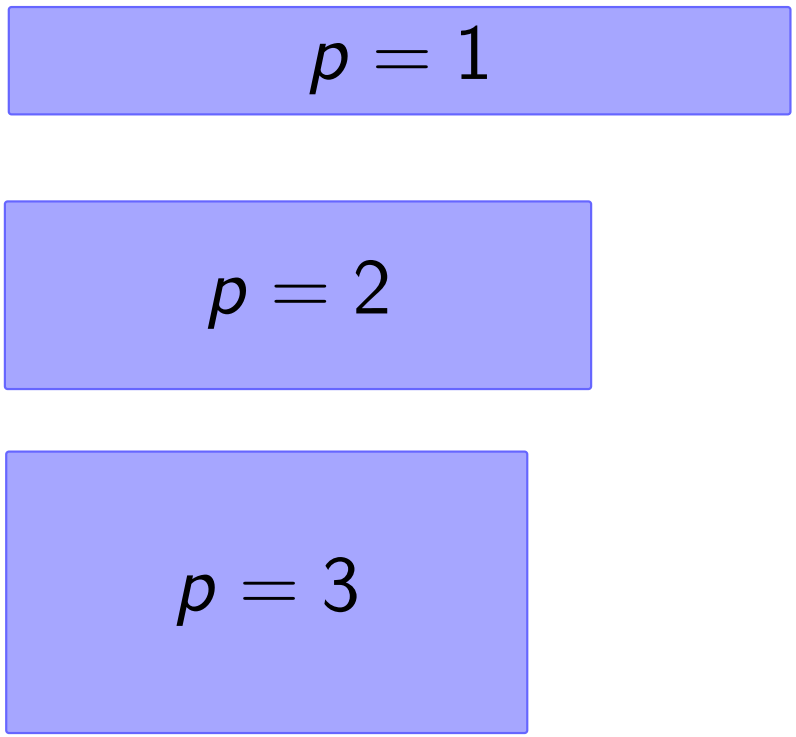


Speedup models

Amdahl model: a fraction γ of the task is sequential

	Time	**Area**
$p = 1$	12	12
$p = 2$	$6 + 6/2 = 9$	18
$p = 3$	$6 + 6/3 = 8$	24

$$t(p) = w\gamma + \frac{w(1-\gamma)}{p}$$



Speedup models

- Roofline model: $t_j(p_j) = \frac{w_j}{\min(p_j, \bar{p}_j)}$, for some $1 \leq \bar{p}_j \leq P$.
- Communication model: $t_j(p_j) = \frac{w_j}{p_j} + (p_j - 1)c_j$, where c_j is the communication overhead.
- Amdahl's model: $t_j(p_j) = w_j(\frac{1-\gamma_j}{p_j} + \gamma_j)$, where γ_j is the inherently sequential fraction.
- General model: $t_j(p_j) = \frac{w_j(1-\gamma_j)}{\min(p_j, \bar{p}_j)} + w_j\gamma_j + (p_j - 1)c_j$, a combination of the three first models.
- Arbitrary model: $t_j(p_j)$ is an arbitrary function of p_j (no superlinear speedup).

Scheduling objective

Scheduling objective:
Find a moldable schedule, i.e., processor allocation p_j and starting time s_j for each task j which

- minimizes makespan: $T = \max_j(s_j + t_j(p_j))$,
- subject to processor constraint: $\sum_{j \text{ active at time } t} p_j \leq P, \forall t$,
- subject to precedence constraint: $j_1 \to j_2 \implies s_{j_2} \geq s_{j_1} + t_{j_1}$.

Competitive ratio:
An online algorithm is said to be *r-competitive* if its makespan T satisfies $\frac{T}{T_{\text{OPT}}} \leq r$ for any task graph, where T_{OPT} denotes the best offline makespan achievable for the instance.

Outline

Offline algorithms

All ($\approx 5 - 10$) offline papers follow:

allocate first $\Longrightarrow$ **schedule greedily**

- Phase 1: choose the number of processors for each task
- Phase 2: apply greedy ASAP list scheduling

Crucial point

Allocations are fixed once and for all: they do not depend on the scheduler state.

If 3 processors are free and a ready task needs 4, the scheduler waits.

Online class studied: LALIST

LALIST = **Local Allocation** + **List Scheduling**

- **Local Allocation:** the allocation rule does not use
  - the task position in the DAG
  - the local scheduler state

  We choose a tradeof between time and area.
- **List Scheduling:** among ready tasks, schedule greedily as soon as possible with backfilling

Motivation

This captures the natural online analogue of the offline template.

Prior results (model-specific scheduling)

Important limitation

All tasks were assumed to follow *one perfectly known speedup model* (and known parameters) in advance.
Otherwise the competitive analysis breaks $\Rightarrow$ **Assumption not realistic.**

Model	Best prior CR	Lower bound[1]
Roofline	2.62	Tight
Communication	3.40	Tight
Amdahl	4.55	Tight
General	4.55[2] (4.58)	Tight
Arbitrary	—	$\Omega(\ln D)$

D = length of the longest chain (diameter / critical path length).

[1] Of any List Scheduling Algorithm with local allocation decision

[2] With ChatGPT improved proof

New results (universal scheduling)

Universal algorithm

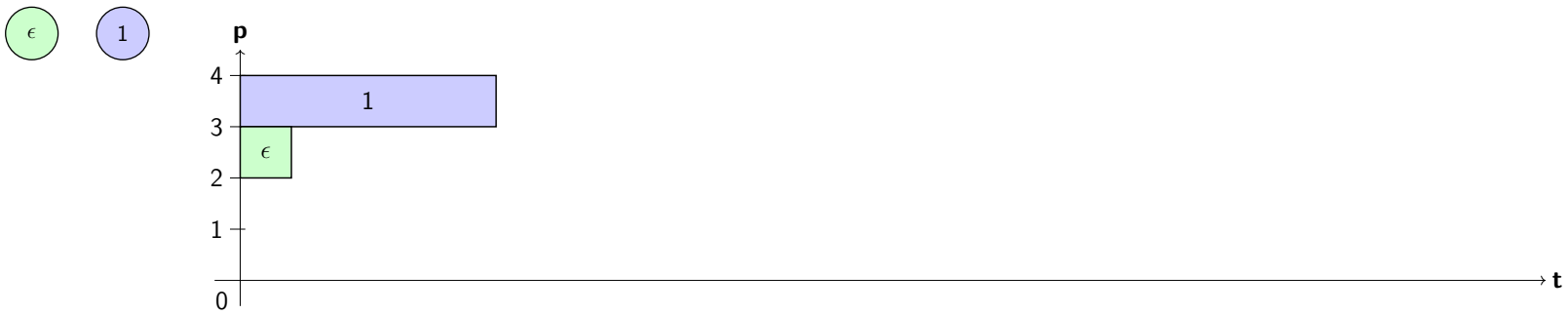
One single online algorithm (Fair) that works *regardless of the true speedup model*. It *learns/adapts* online and still achieves (almost) the best model-specific ratios.

Model	**Best prior CR**	**Fair (universal) CR**
Rooftline (Roo)	2.62	Tight
Communication (Com)	3.40	3.41
Amdahl (Amd)	4.55	4.56
General (Gen)	4.55	4.56[3] (4.62)

- Prior lower bound (arbitrary speedup model): $\Theta(\ln D)$.
- New lower bound: $\frac{\sqrt{P}}{2} - 1$ (and Fair achieves $2\sqrt{P} + 3$).

[3] With ChatGPT improved proof

Idea: Reduce both time and area for each task + ASAP

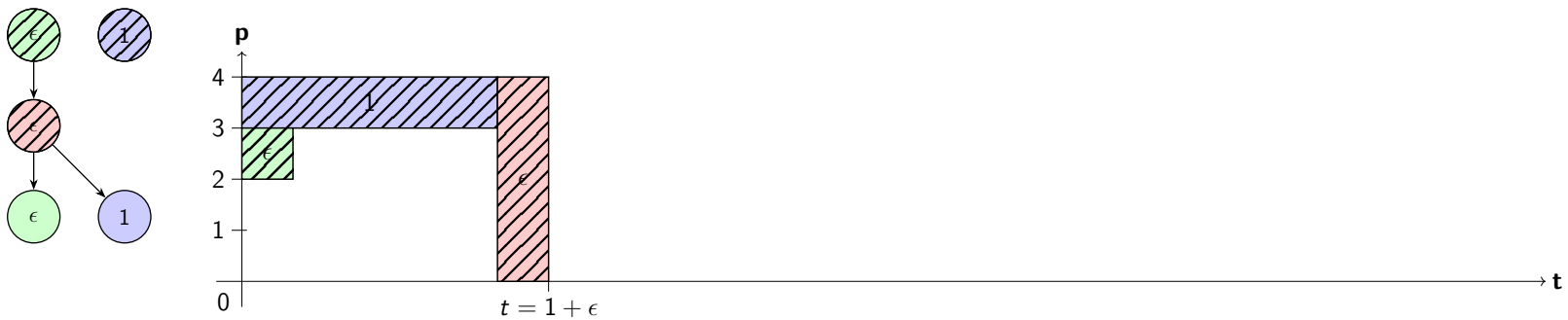


Both tasks are sequential, $p = 1$ minimizes both time and area.

Idea: Reduce both time and area for each task + ASAP

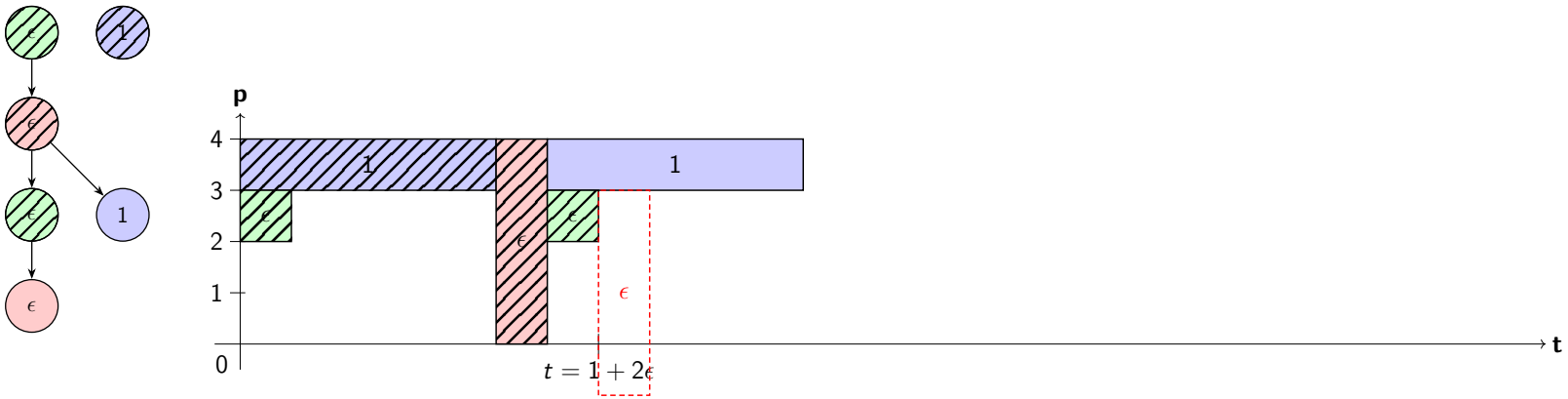
Red task has perfect speedup, $t(p) = \frac{1}{p}$, $p = P$ minimizes both time and area...

Idea: Reduce both time and area for each task + ASAP



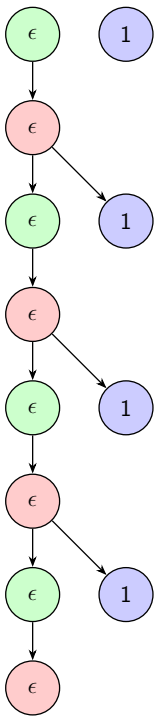
Red task has perfect speedup, $t(p) = \frac{1}{p}$, $p = P$ minimizes both time and area... But it doesn't look great.

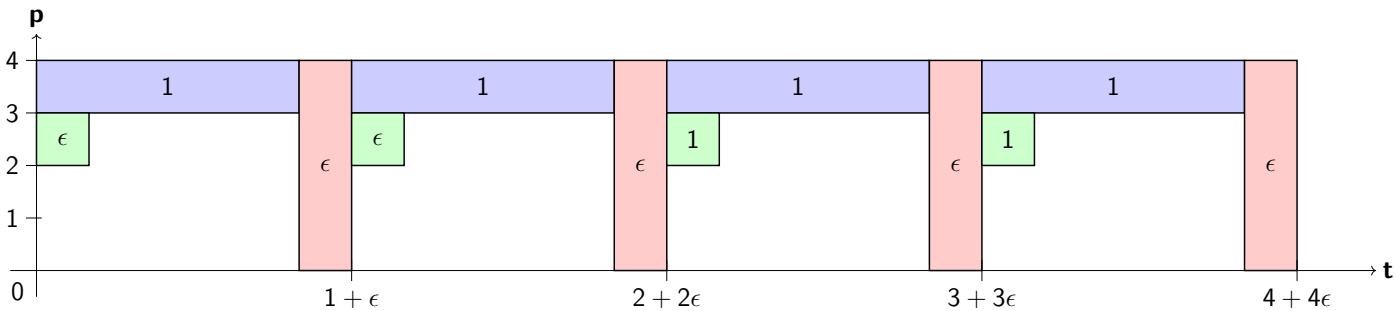
Idea: Reduce both time and area for each task + ASAP



Same situation repeats.

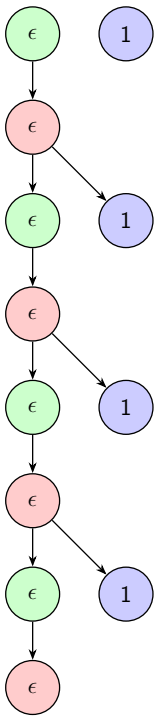
Idea: Reduce both time and area for each task + ASAP

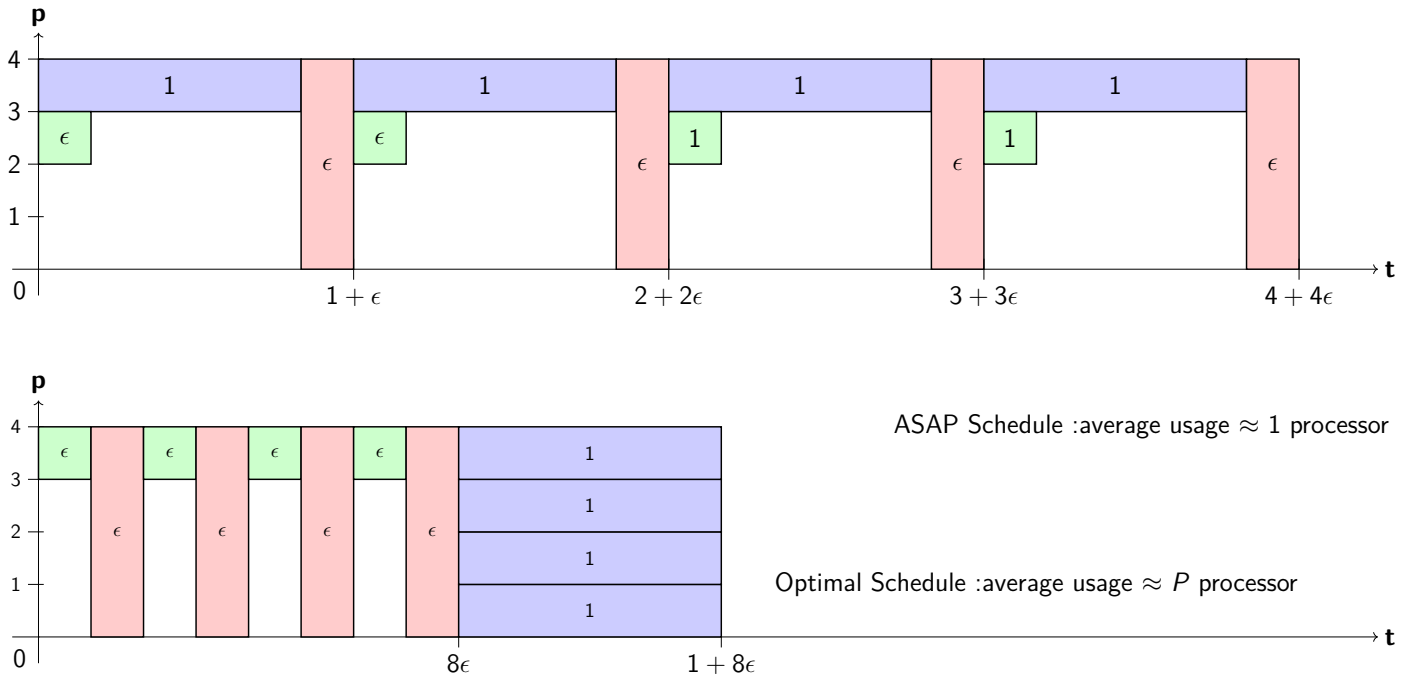




Same situation repeats.

Idea: Reduce both time and area for each task + ASAP



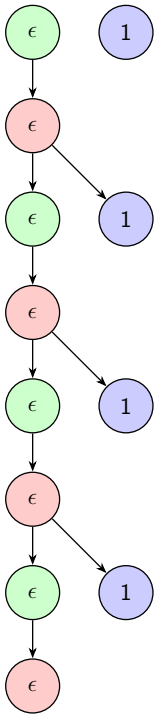


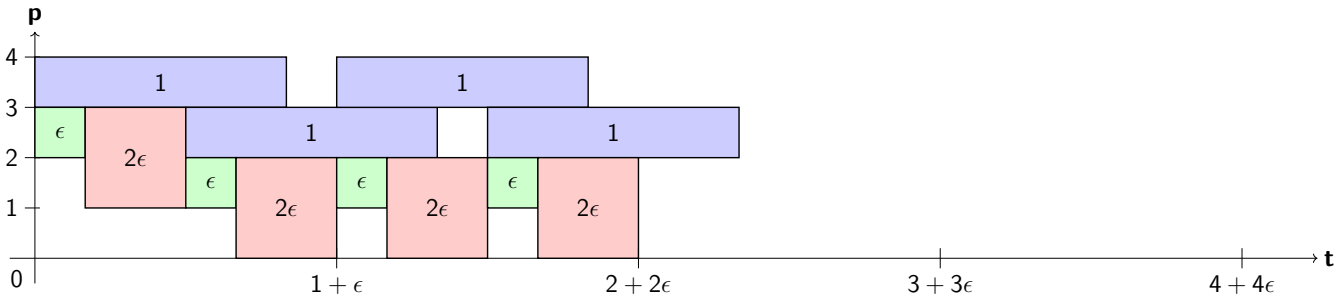
$\frac{T}{T^{opt}} \approx P = \frac{n}{3}$: Unacceptable!

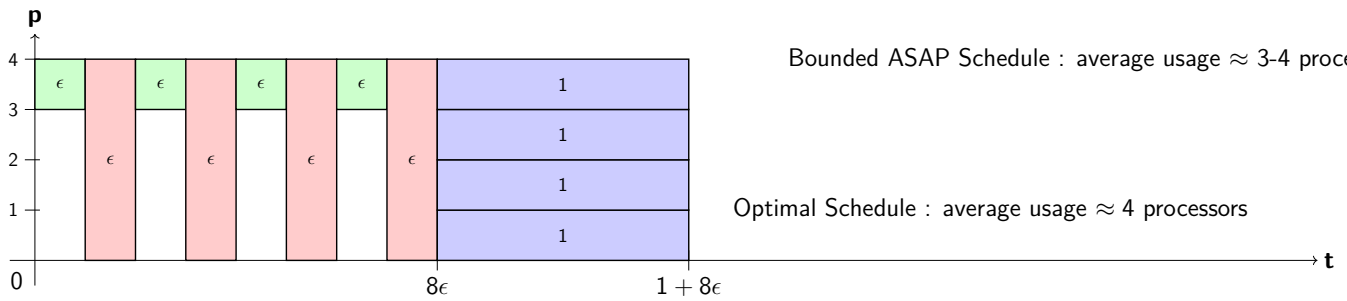
Problem: $P - 1$ processor idle while red tasks are ready.

Solution: Sacrifice optimal execution time to cap the maximum processor usage (for example, to $\frac{P}{2}$).

Online Setting: How bad can ASAP be?







Bounded ASAP Schedule : average usage $\approx$ 3-4 processor

Optimal Schedule : average usage $\approx$ 4 processors

$\frac{T}{T^{opt}} \approx P = \frac{n}{3}$: Unacceptable!

Problem: $P-1$ processor idle while red tasks are ready.

Solution: Sacrifice optimal execution time to cap the maximum processor usage (for example, to $\frac{P}{2}$).

$\Rightarrow$ If less than $\frac{P}{2}$ processors are used, queue is empty.

Universal Algorithm (Fair)

Step (1): Compute a per-task ratio and a reference allocation

For each newly available task j, compute:

- an allocation $p_j \in [1, P]$ that **balances time and area**, and
- a task ratio R_j capturing the **best achievable trade-off** for task j.

$$R_j = \min_{p \in [1,P]} \max \left\{ \frac{t_j(p)}{t_j^{\min}}, \frac{a_j(p)}{a_j^{\min}} \right\}.$$

Let $R^* = \max_j R_j$ over tasks discovered so far ("hardest" task).

Universal Algorithm (Fair)

Step (1): Compute a per-task ratio and a reference allocation

For each newly available task j, compute:

- an allocation $p_j \in [1, P]$ that **balances time and area**, and
- a task ratio R_j capturing the **best achievable trade-off** for task j.

$$R_j = \min_{p \in [1,P]} \max\left\{ \frac{t_j(p)}{t_j^{\min}}, \frac{a_j(p)}{a_j^{\min}} \right\}.$$

Let $R^* = \max_j R_j$ over tasks discovered so far ("hardest" task).

Step (2): Reduce allocations using R^*

When scheduling from the waiting queue, assign

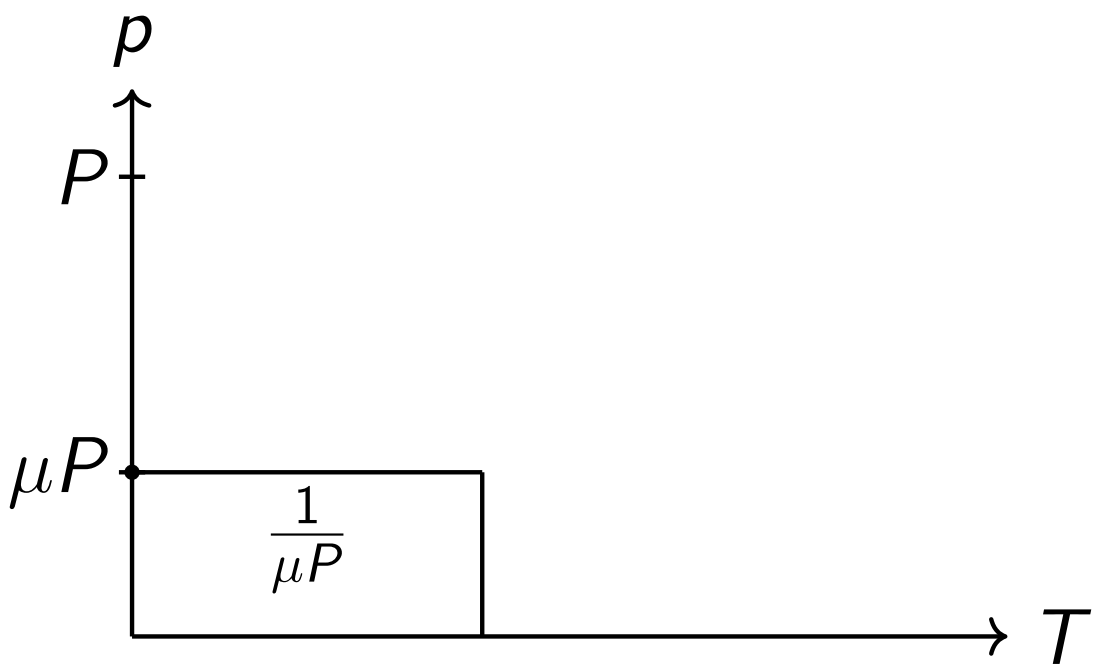
$$p'_j \leftarrow \min\Big(p_j, \ \lfloor \mu(R^*)\, P \rfloor\Big),$$

and start task j as soon as p'_j processors are available.

$\Rightarrow \mu(x) = \frac{2x+1-\sqrt{4x^2+1}}{2x} \approx \frac{1}{2x}$ prevents from using too much processors

Outline

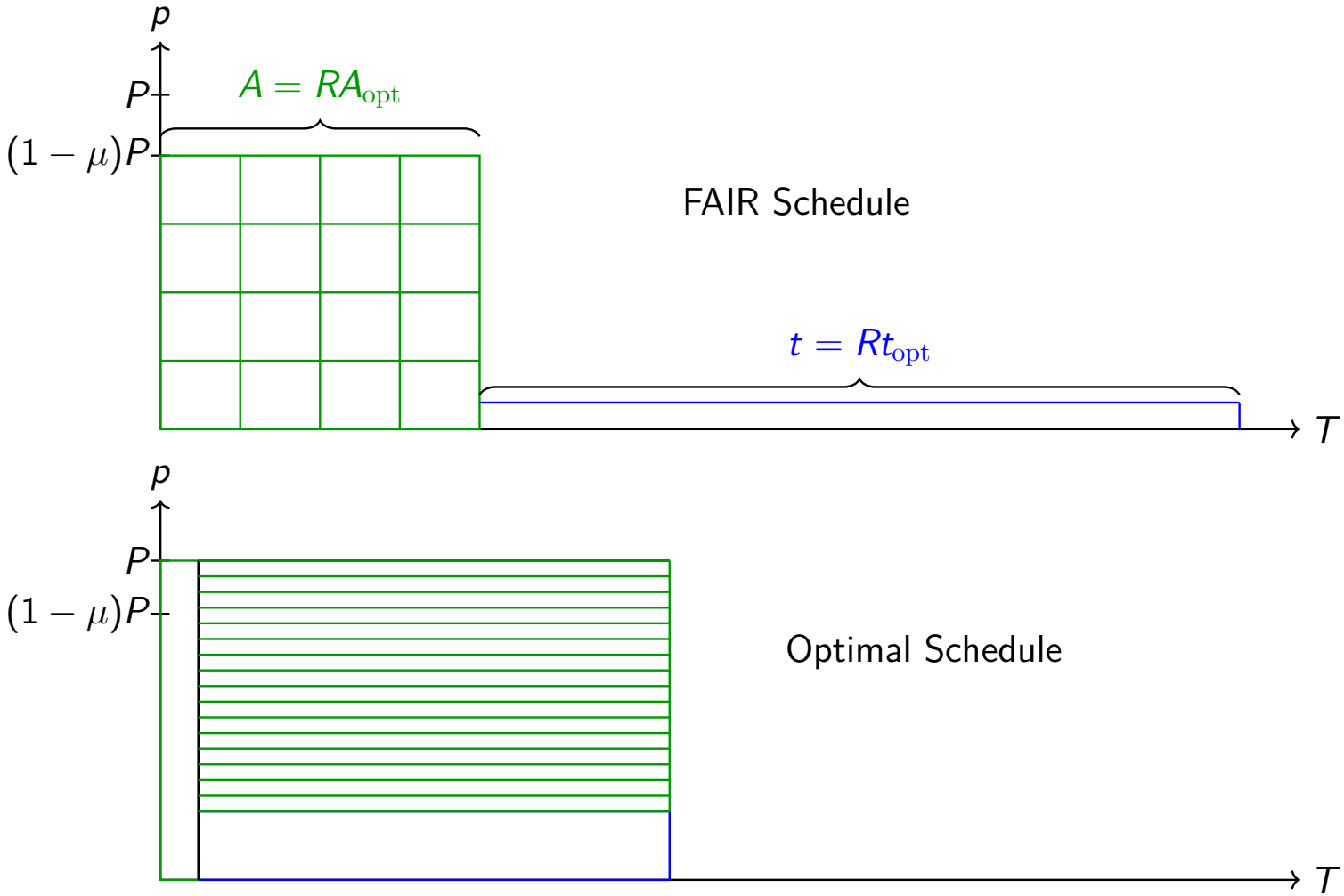
We choose $\mu(R)$ so that $\frac{1}{\mu} = \frac{R}{1-\mu} + R$: 2 worst cases



$$t(p) = \frac{1}{p} \qquad \mathrm{OPT} = \frac{1}{P} \qquad \frac{T}{T_{\mathrm{opt}}} = \frac{\frac{1}{\mu P}}{\frac{1}{P}} = \frac{1}{\mu}$$

We choose $\mu(R)$ so that $\frac{1}{\mu} = \frac{R}{1-\mu} + R$: 2 worst cases

All green tasks are independent, a long blue task after one of them.



$$T = \frac{RA_{opt}}{(1-\mu)P} + Rt_{opt} \qquad T_{opt} \approx \frac{A_{opt}}{P} \approx t_{opt} \qquad \frac{T}{T_{\text{opt}}} \approx \frac{R}{1-\mu} + R$$

Analysis: high level

General analysis

- Upperbound: Show that the two previous setup are the worst cases
- Lower bound: Show that for any LALIST algorithm, an adversary can build a setup that end up "roughly" in one of the previous cases

Final ratio: $\frac{T}{T_{opt}} = \frac{2R}{2R+1-\sqrt{4R^2+1}} \approx 2R + 0.5$

Model specific analysis

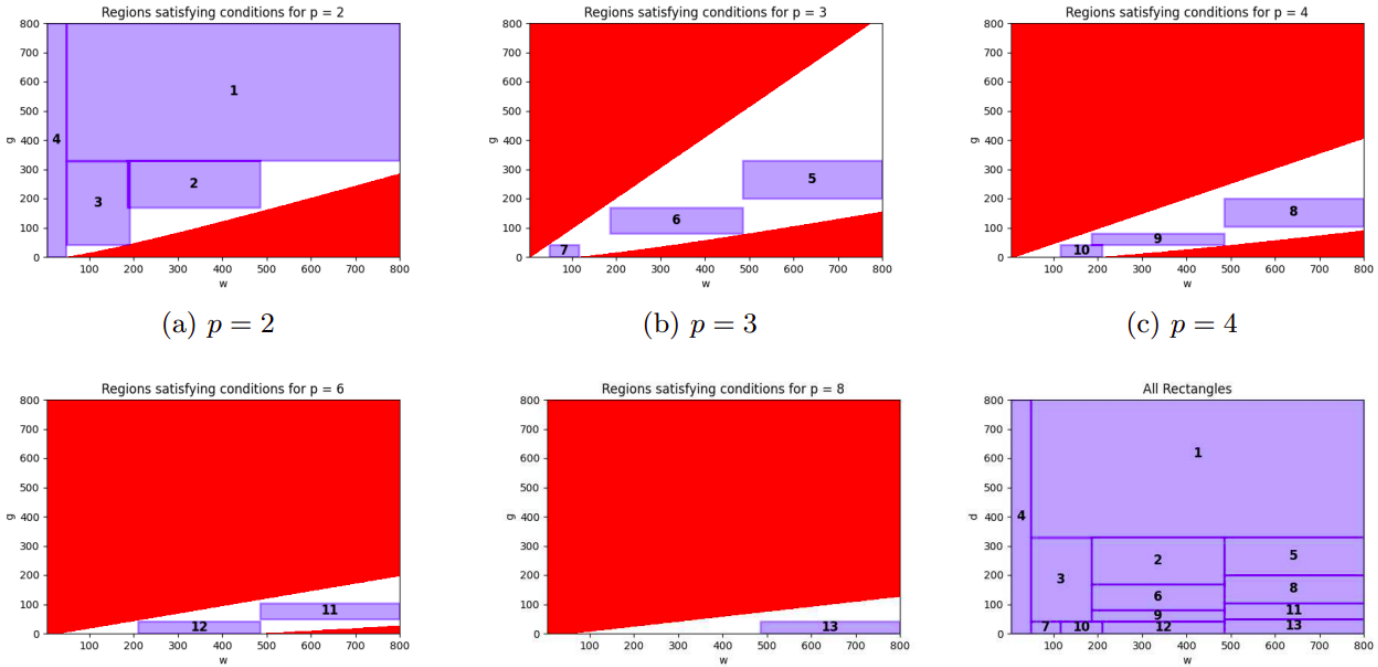
Given a class of function an a number of processor P, what is the worst possible $\bar{R}$?

$\Rightarrow$ Easy to find if we allow p to be non integer, not here.

Model specific analysis

- Roofline model: $t_j(p_j) = \frac{w_j}{\min(p_j, \bar{p}_j)}$, for some $1 \leq \bar{p}_j \leq P$.
  Choose $p_j = \bar{p}_j$ to obtain $R = 1$
- Communication model: $t_j(p_j) = \frac{w_j}{p_j} + (p_j - 1)c_j$,
  Worst $\bar{R}$ happens with $p_j = 1$ or $p_j = 2$, with $\bar{R} = \sqrt{2}$. The proof treats small p_j manually, and for high p_j this is not tight, so rounding a solution in $\mathbb{R}$ works
- Amdahl's model: $t_j(p_j) = w_j(\frac{1-\gamma_j}{p_j} + \gamma_j)$,
  With $p_j = \lceil \frac{w}{d} \rceil$, we obtain $\bar{R} = 2$, tightness is easy
- General model: $t_j(p_j) = \frac{w_j(1-\gamma_j)}{\min(p_j, \bar{p}_j)} + w_j\gamma_j + (p_j - 1)c_j$,
  I could show $2 \leq \bar{R} \leq 2.02$, and this is where AI showed $\bar{R} = 2$

Quest for $\bar{R}$ in General Model: my approach



(a) $p = 2$ (b) $p = 3$ (c) $p = 4$

(d) $p = 6$ (e) $p = 8$ (f) All Rectangles

Fig. 3: Comparative plots for different values of p.

Similarly to communication model, I solve low values of p first, then round continuous solution at cost of an extra 0.02

$$p = \begin{cases} \text{Choose } p \text{ according to figures} & \frac{w}{c} \leq 800 \text{ and } \frac{d}{c} \leq 800, \\ \min\left(\bar{p}, \left\lceil \frac{w+d}{\sqrt{wc}+d} - \frac{1}{2} \right\rceil\right), & \textit{Otherwise.} \end{cases}$$

Quest for $\bar{R}$: ChatGPT Thinking 5.4 solution

Solve separately when c dominates (Communication like) or when d dominates (Amdahl like).

Theorem

Let

$$t(p) = \frac{w}{\min(p, \bar{p})} + d + c(p-1), \qquad 1 \le p \le P,$$

Then there exists $p \in \{1, \dots, P\}$ *such that*

$$t(p) \le 2t_{\min} \qquad \textit{and} \qquad p\,t(p) \le 2t(1) = 2(w+d).$$

One may choose

$$p = \begin{cases} 1, & \frac{w}{c} \le 12, \\ \min\left(\bar{p}, \left\lceil \frac{1}{3}\sqrt{\frac{w}{c}} \right\rceil\right), & \frac{w}{c} > 12 \text{ and } d \le 2\sqrt{wc}, \\ \min\left(\bar{p}, \left\lceil \frac{w}{d+\sqrt{wc}} \right\rceil\right), & \frac{w}{c} > 12 \text{ and } d > 2\sqrt{wc}. \end{cases}$$

Outline

Conclusion

Idea

A simple principle is optimal for LALIST class of algorithm:

control time/area + reduce allocation

- We introduced the class LALIST, capturing the natural online analogue of many offline approaches.
- We designed $\textsc{Fair}$, a **universal** online algorithm for moldable DAG scheduling.
- $\textsc{Fair}$ achieves **near-tight competitive ratios** for several classical speedup models.
- Thus, even without knowing the true speedup model in advance, we can remain close to the best model-specific performance.

We turned an unrealistic algorithm into a more realistic one with almost no increase in the competitive ratio

Extra experiment slides

Param	Default	Test Values	Type	Description
n	2500	$[500, \ldots, 5000]$	Scale	Number of tasks
P	512	$[124, \ldots, 1024]$	System	Available processors
Priority	FIFO	[FIFO, procs . . .]	Policy	Queue-ordering strategy
Density	0.5	$[0, \ldots, 1]$	Graph	Edge probability
Fat	0.5	$[0, \ldots, 1]$	Graph	Width/depth ratio
Regular	0.5	$[0, \ldots, 1]$	Graph	Degree uniformity
Jump	5	$[1, \ldots, 9]$	Graph	Maximum jump distance between layers

Table 5: Experimental parameters and their default values.

- Graph were generated with DAGGEN, speedup functions are random.
- One experiment per model/parameter. We make the parameter vary, all the other are fixed to default value
- 50 instances generated for each parameter, normalized by the Lower Bound

Extra experiment slides

Table 6: Average Values for Each Model and Heuristic

Model	ICPP22	TOPC24	Fair	minTime	minArea
Roofline	1.28	1.28	1.28	1.52	11.28
Communication	1.75	1.38	1.38	2.97	10.22
Amdahl	2.32	1.98	2.01	21.31	7.88
General	2.74	1.50	1.60	4.86	5.37
PowerCom	1.52	1.46	1.35	2.84	12.76
Average	1.92	1.52	1.52	6.70	9.50

Results are almost identical than the best model-specific heuristic