Throughput Maximization on Identical Machines

Coauthors: Benjamin Moseley, Kirk Pruhs, Clifford Stein

IPCO 2022

Summary

- A deterministic $O(1)$-competitive alg. for Online Throughput Maximization on $m > 1$ machines.
- Concludes 20-year line of research since $m = 1$ case settled

Summary

- A deterministic $O(1)$-competitive alg. for Online Throughput Maximization on $m > 1$ machines.
- Concludes 20-year line of research since $m = 1$ case settled

Bala Kalyanasundaram, Kirk Pruhs:
Maximizing job completions online. J. Algorithms 2003.

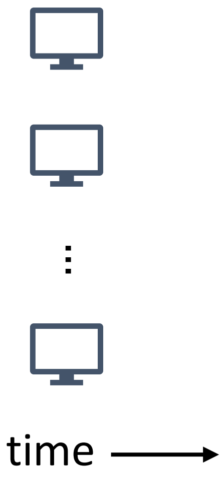
Summary

- A deterministic $O(1)$-competitive alg. for Online Throughput Maximization on $m > 1$ machines.
- Concludes 20-year line of research since $m = 1$ case settled
- **Algorithm:** Run 3 algs. on $m/3$ machines each

Bala Kalyanasundaram, Kirk Pruhs:
Maximizing job completions online. J. Algorithms 2003.

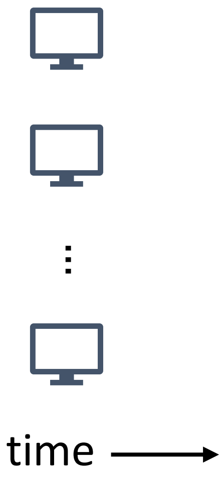
Online Throughput Maximization

- m identical machines



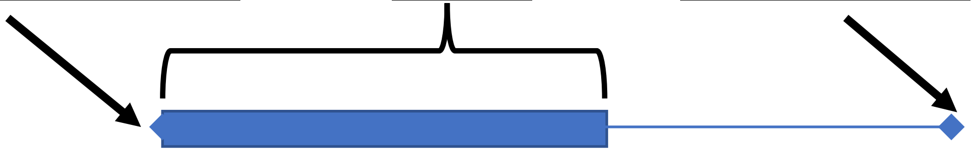
Online Throughput Maximization

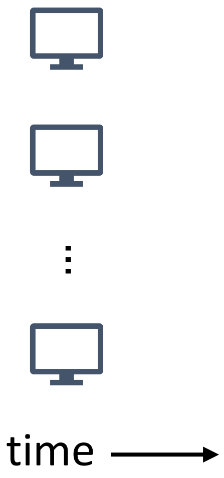
- m identical machines
- Jobs arrive online at their release times with sizes and deadlines



Online Throughput Maximization

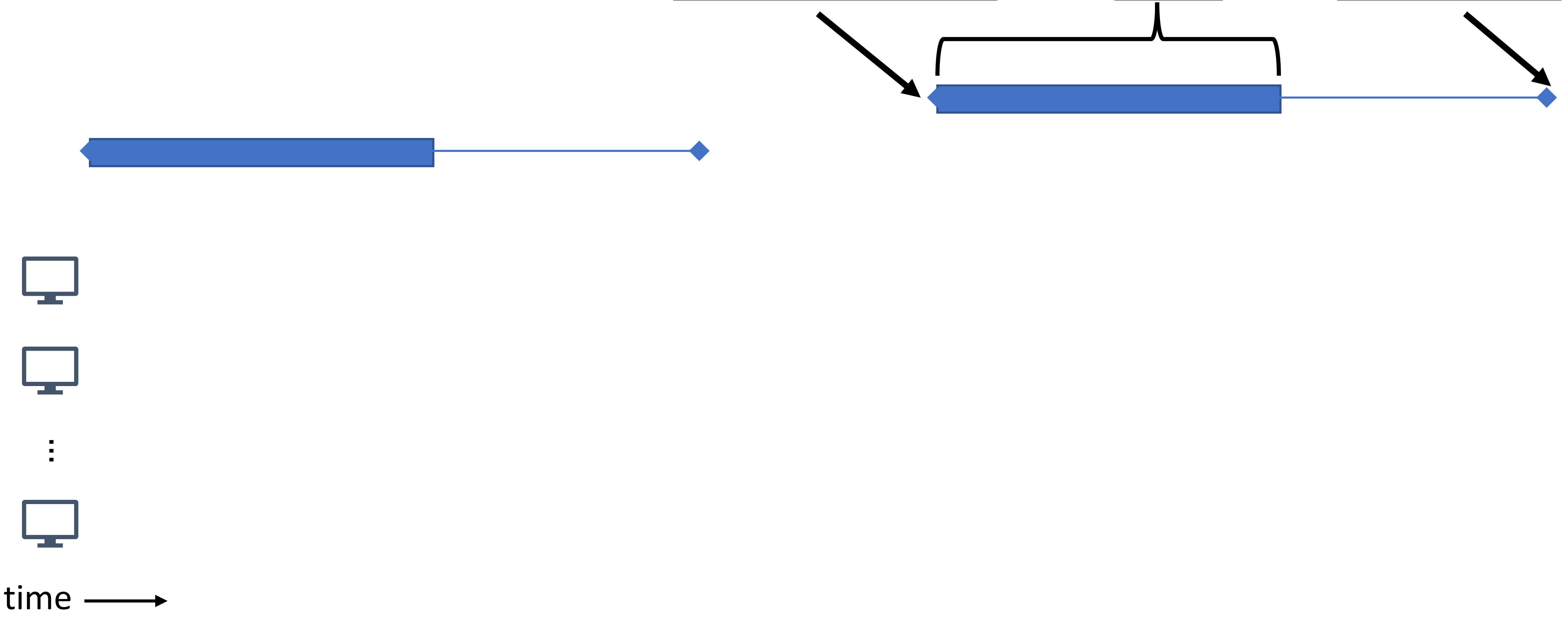
- m identical machines
- Jobs arrive online at their release times with sizes and deadlines





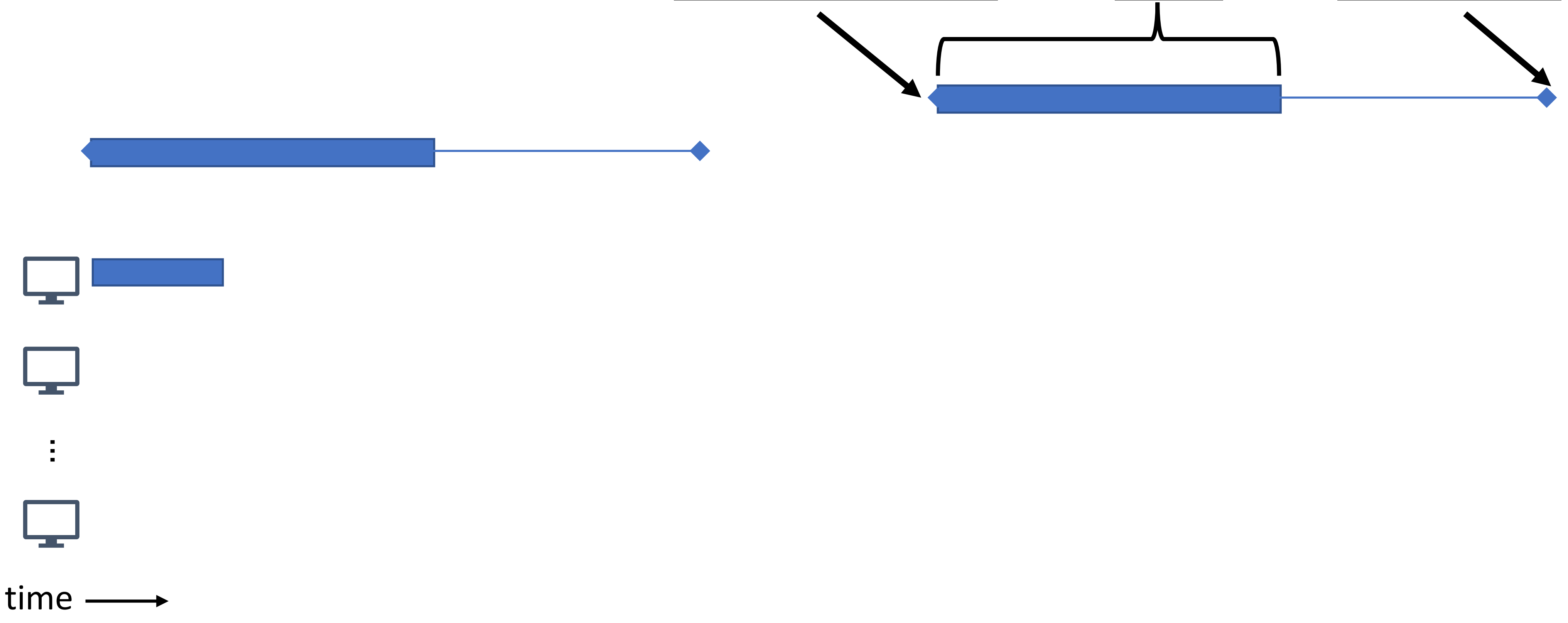
Online Throughput Maximization

- m identical machines
- Jobs arrive online at their release times with sizes and deadlines



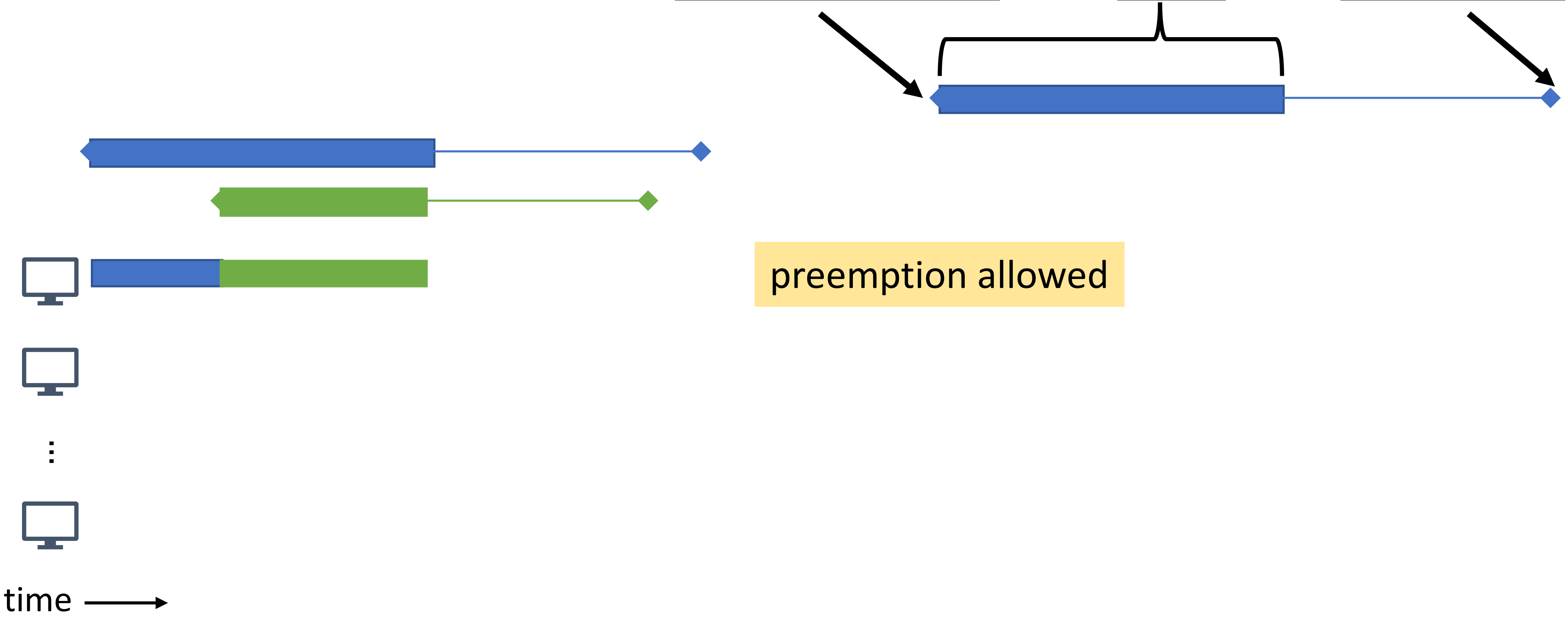
Online Throughput Maximization

- m identical machines
- Jobs arrive online at their release times with sizes and deadlines



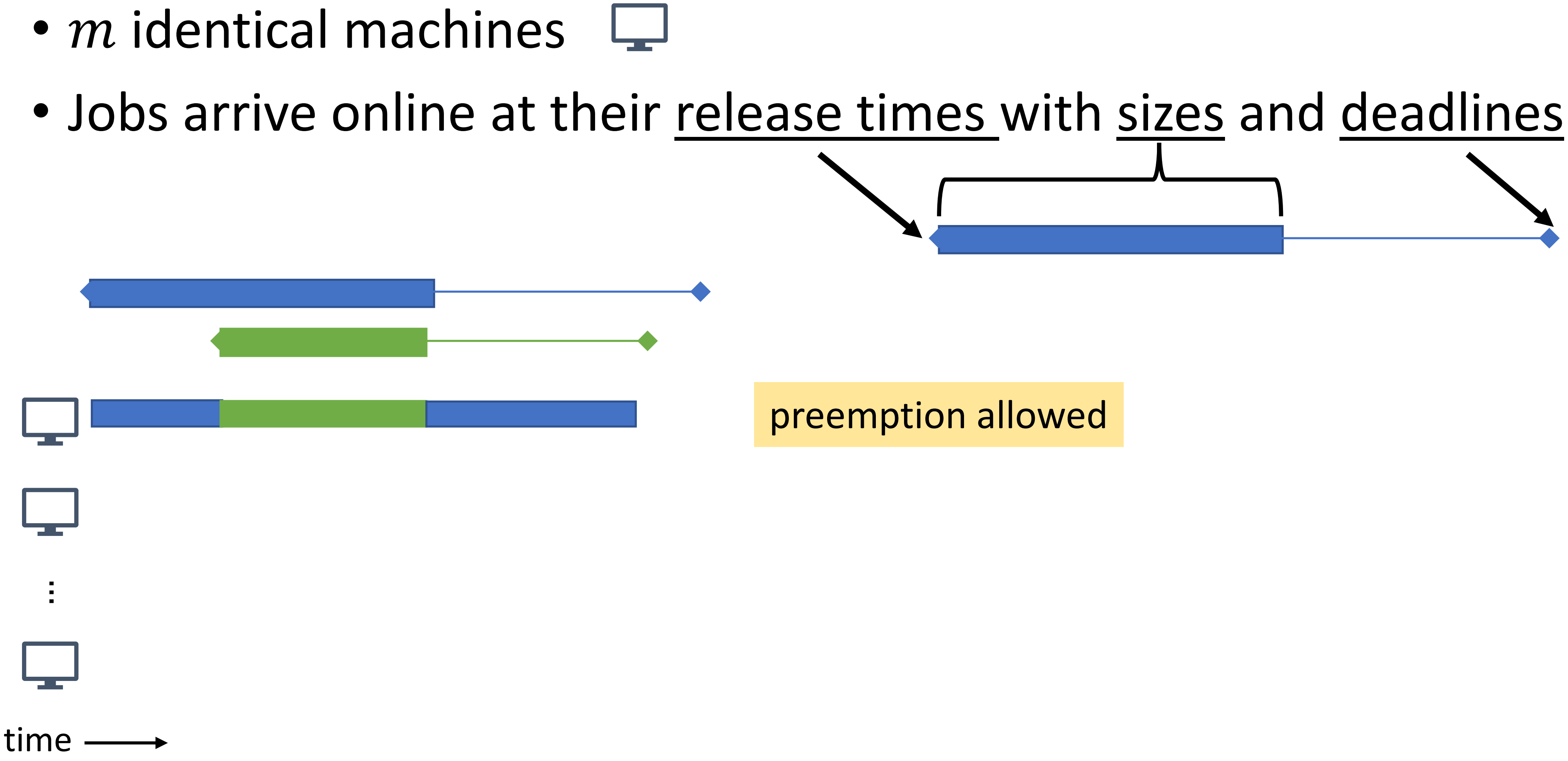
Online Throughput Maximization

- m identical machines
- Jobs arrive online at their <u>release times</u> with <u>sizes</u> and <u>deadlines</u>



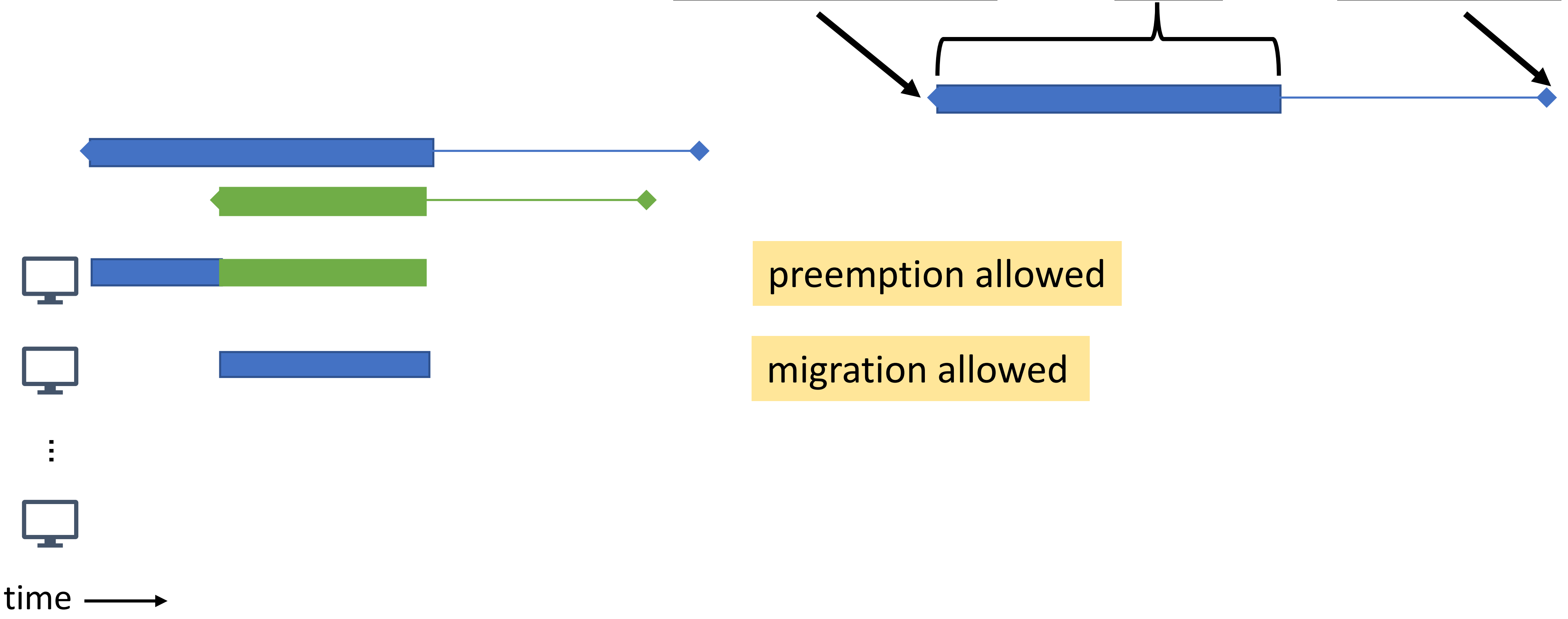
Online Throughput Maximization

- m identical machines
- Jobs arrive online at their release times with sizes and deadlines



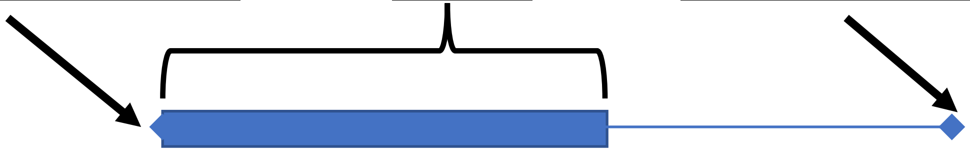
Online Throughput Maximization

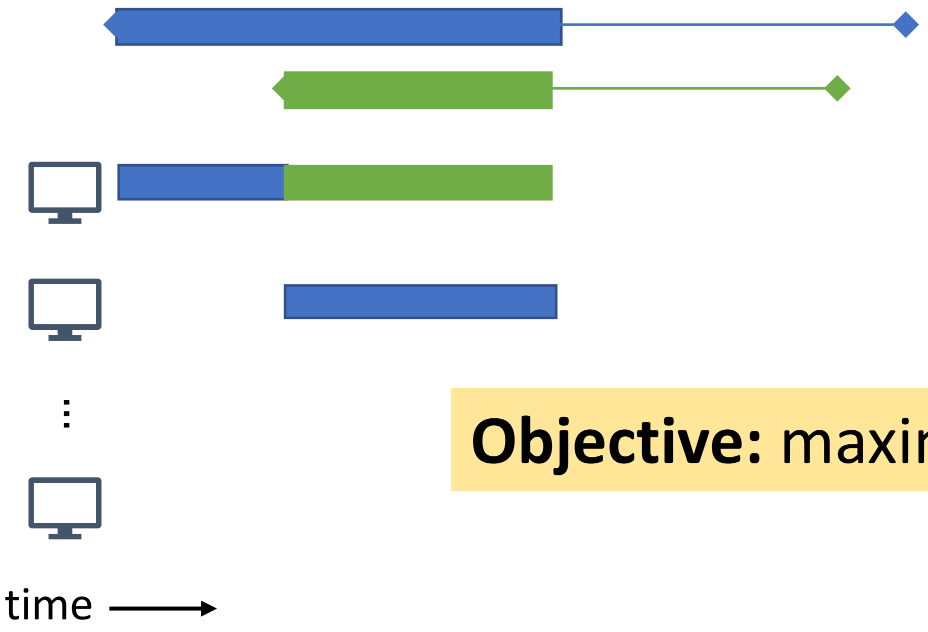
- m identical machines
- Jobs arrive online at their release times with sizes and deadlines



Online Throughput Maximization

- m identical machines
- Jobs arrive online at their release times with sizes and deadlines





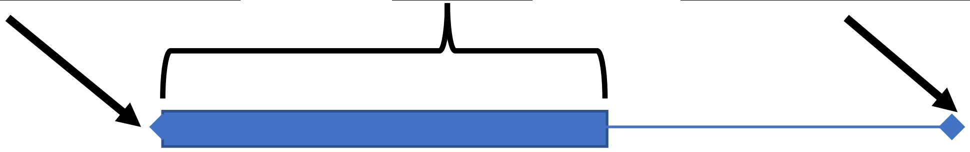
preemption allowed

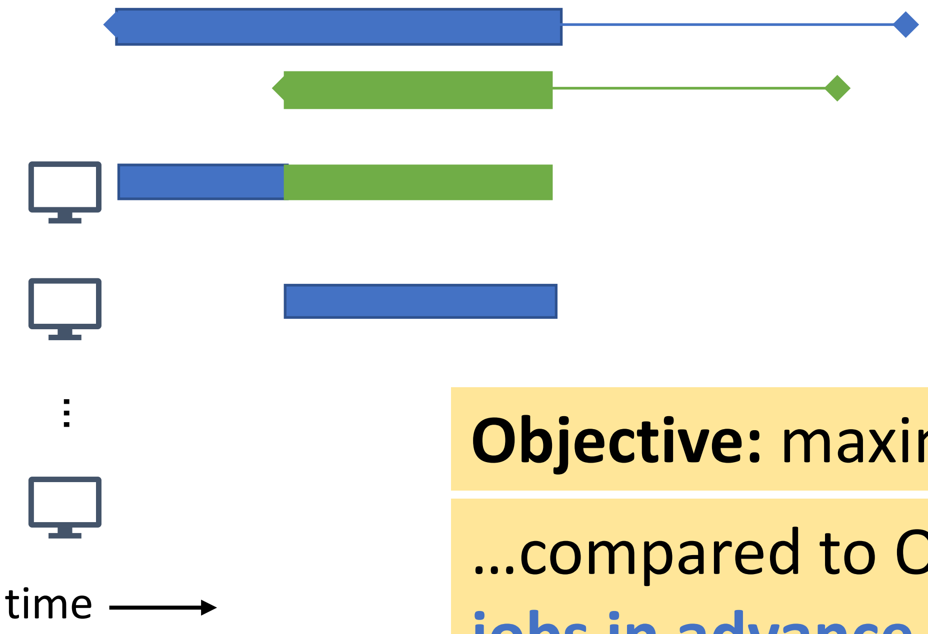
migration allowed

Objective: maximize # jobs completed by deadline

Online Throughput Maximization

- m identical machines
- Jobs arrive online at their release times with sizes and deadlines





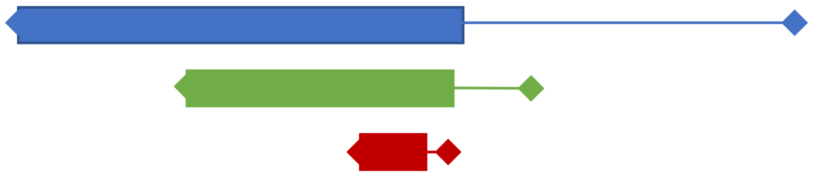
preemption allowed

migration allowed

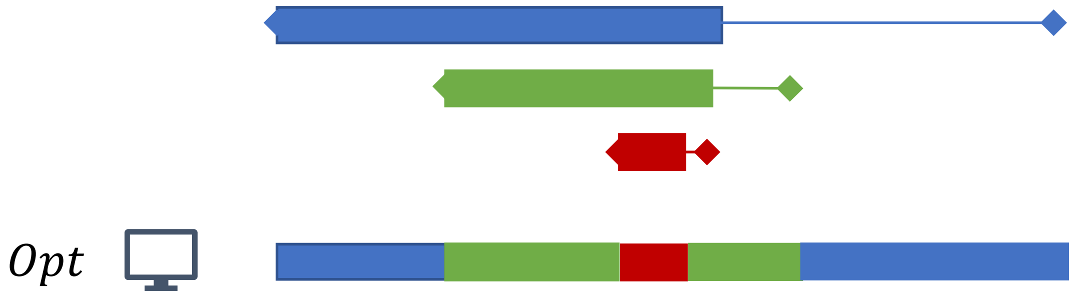
Objective: maximize # jobs completed by deadline

…compared to Opt. offline sched. that **knows all jobs in advance**

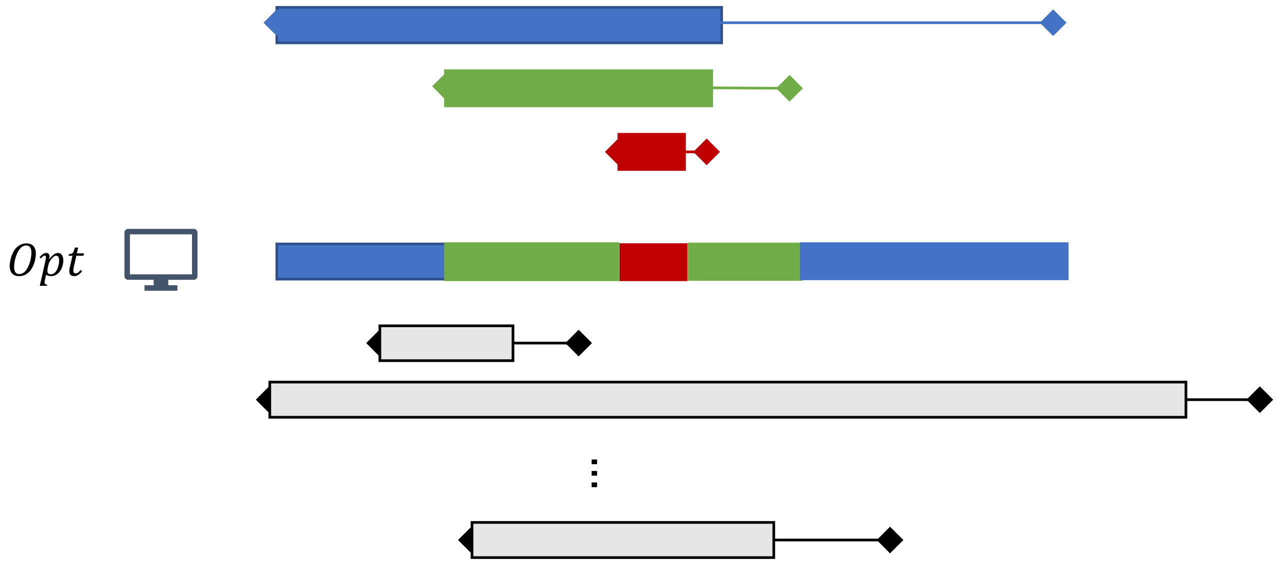
Hard Example



Hard Example

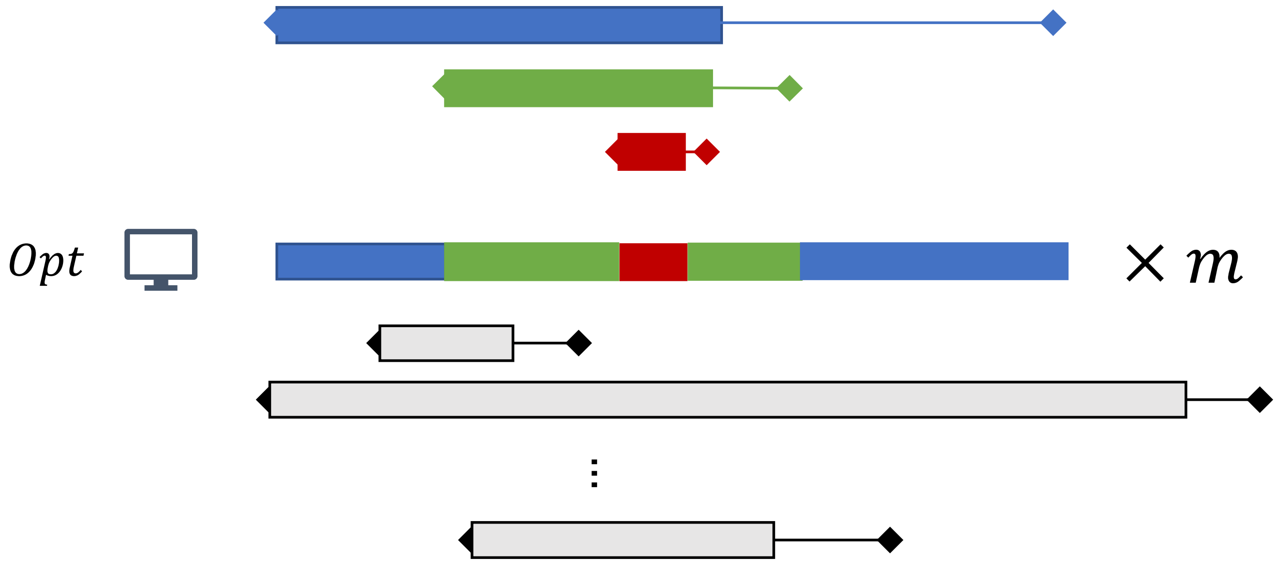


Hard Example



Q: How can Alg. know what is the 'right' job to preempt for?

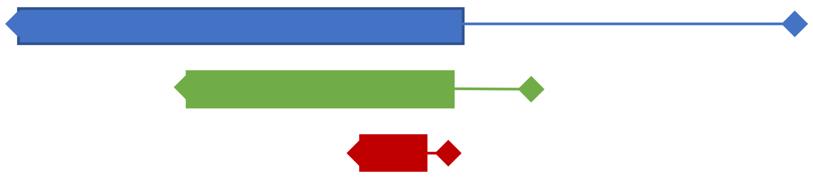
Hard Example



Q: How can Alg. know what is the ‘right’ job to preempt for?

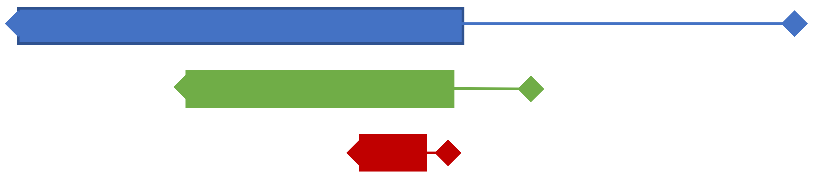
Q: How can Alg. know what is the ‘right’ machine?

Laxity



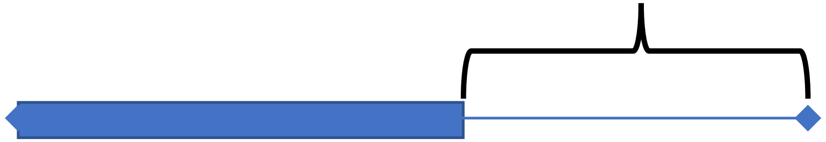
- Hard examples = jobs have **low laxity**

Laxity

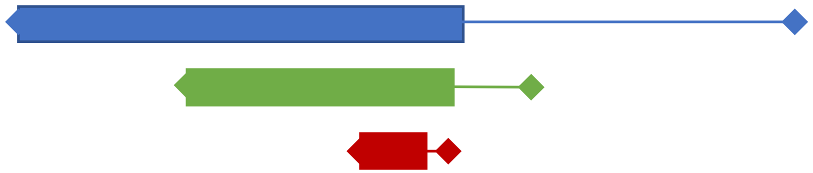


- Hard examples = jobs have **low laxity**

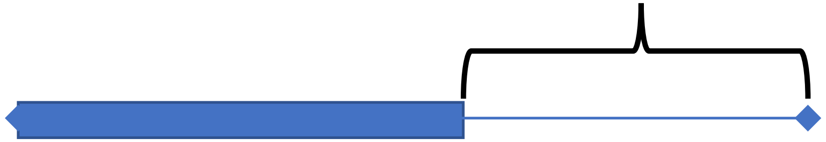
$laxity = lifetime - size$



Laxity

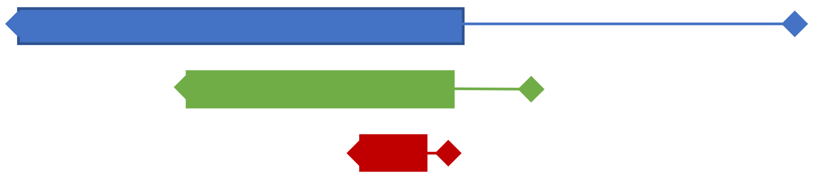


$$laxity = lifetime - size$$

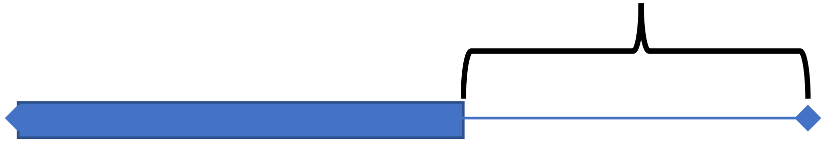


- Hard examples = jobs have **low laxity**
- All jobs have $laxity = \Omega(size) \Rightarrow O(1)$ −competitive alg. known

Laxity



$laxity = lifetime - size$

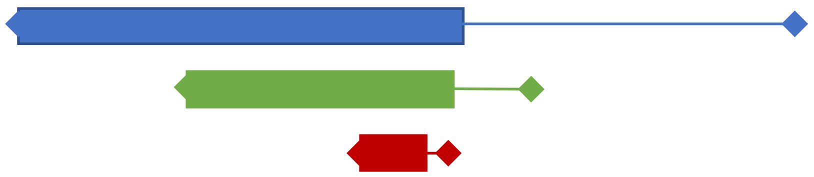


- Hard examples = jobs have **low laxity**
- All jobs have $laxity = \Omega(size) \Rightarrow O(1)$ −competitive alg. known

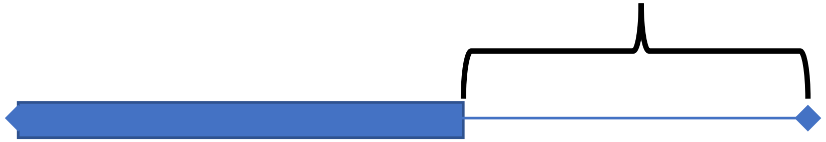
Brendan Lucier, Ishai Menache, Joseph Naor, Jonathan Yaniv:
Efficient online scheduling for deadline-sensitive jobs. SPAA 2013.

Franziska Eberle, Nicole Megow, Kevin Schewior:
Optimally handing commitment issues in online throughput maximization. ESA 2020.

Laxity



$$laxity = lifetime - size$$



- Hard examples = jobs have **low laxity**
- All jobs have $laxity = \Omega(size) \Rightarrow O(1)$ −competitive alg. known

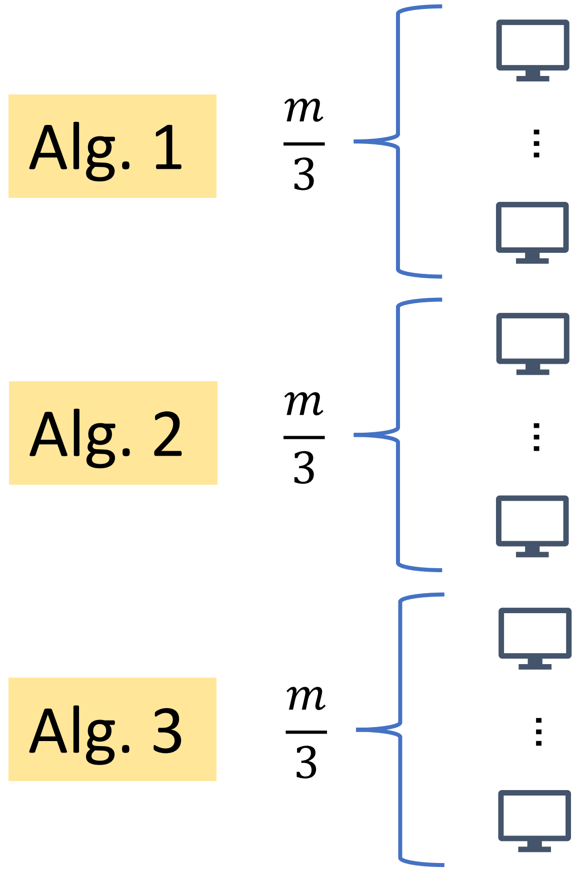
Main Challenge: How to handle jobs where $\boldsymbol{laxity \ll size}$?

Brendan Lucier, Ishai Menache, Joseph Naor, Jonathan Yaniv:
Efficient online scheduling for deadline-sensitive jobs. SPAA 2013.

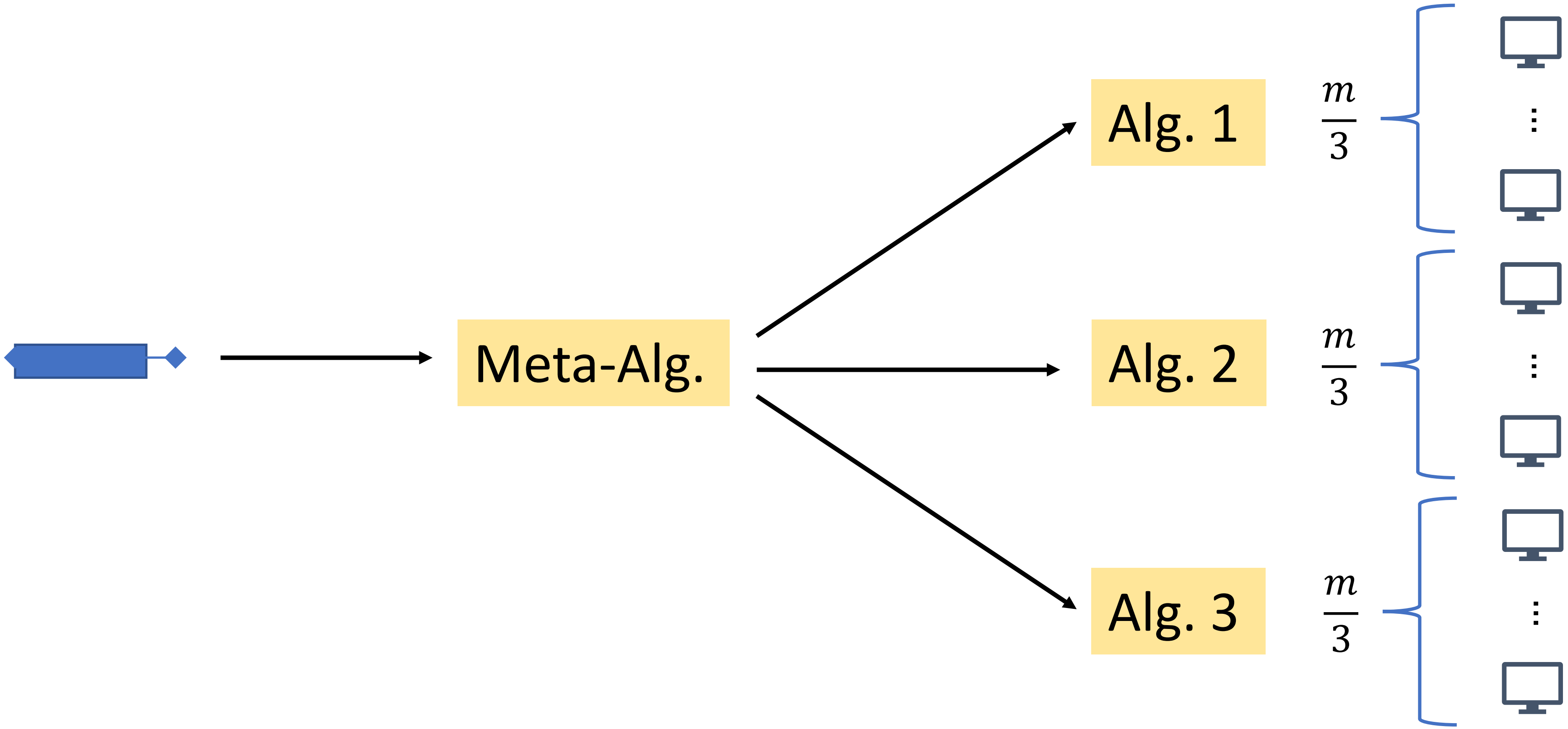
Franziska Eberle, Nicole Megow, Kevin Schewior:
Optimally handing commitment issues in online throughput maximization. ESA 2020.

Algorithm Idea

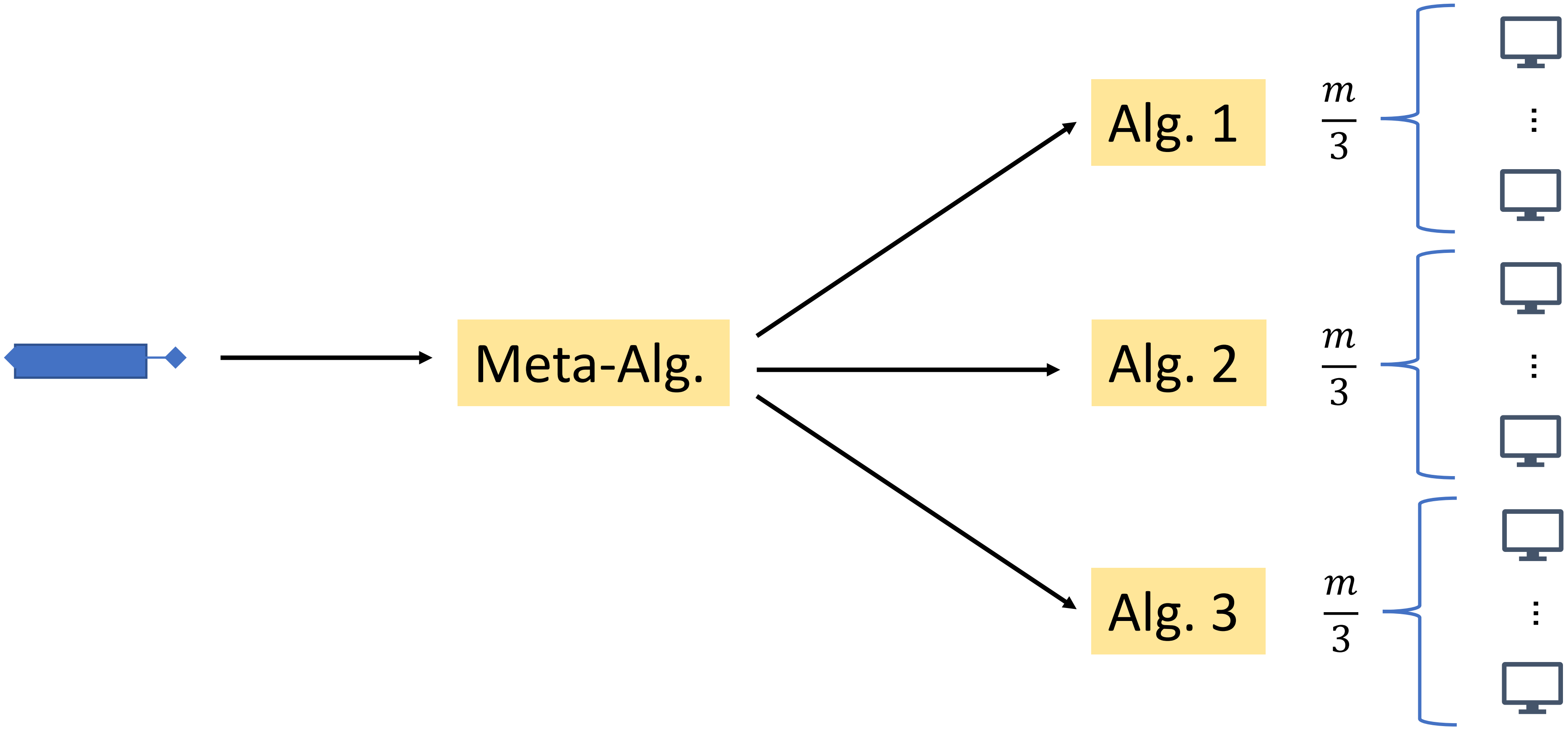
Algorithm Idea



Algorithm Idea

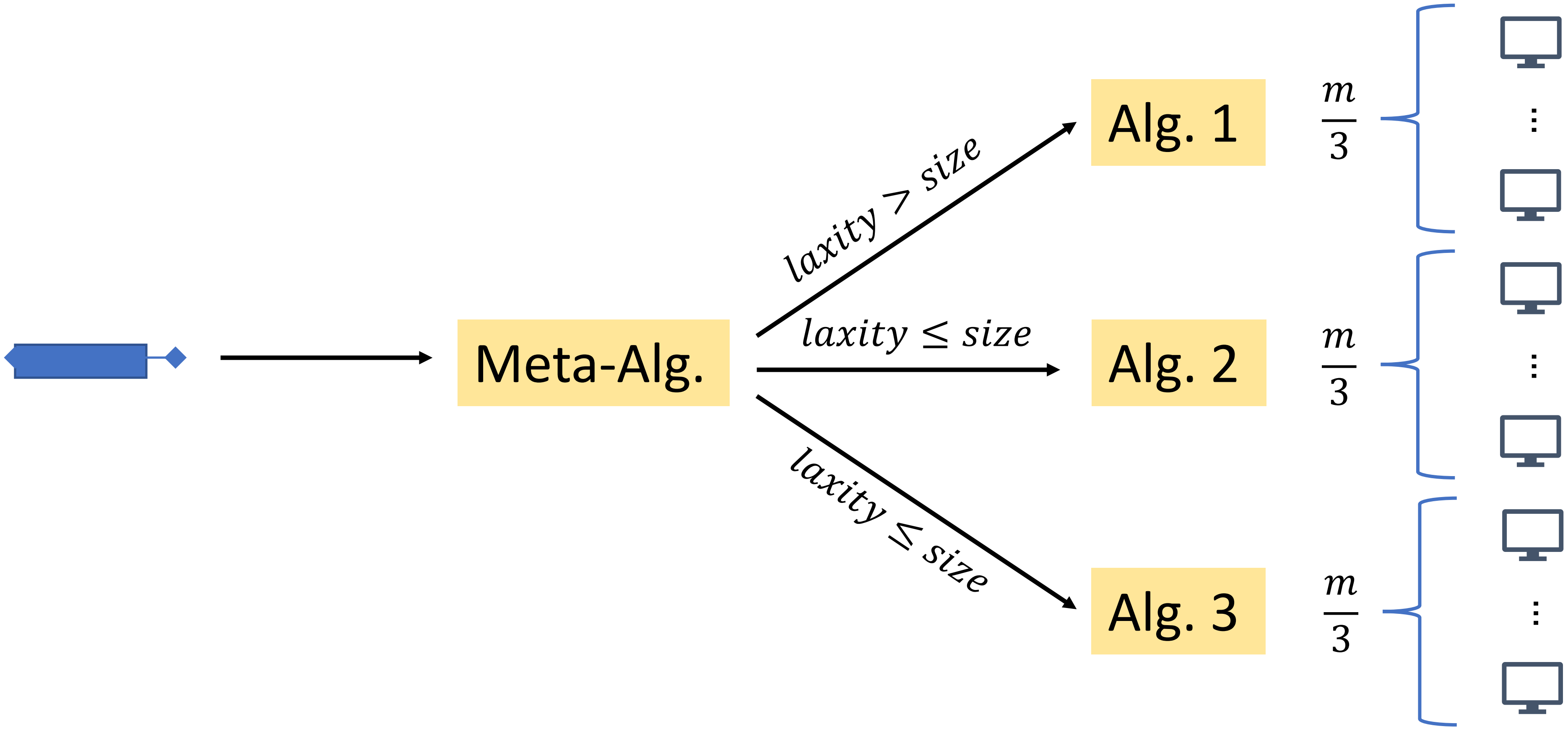


Algorithm Idea



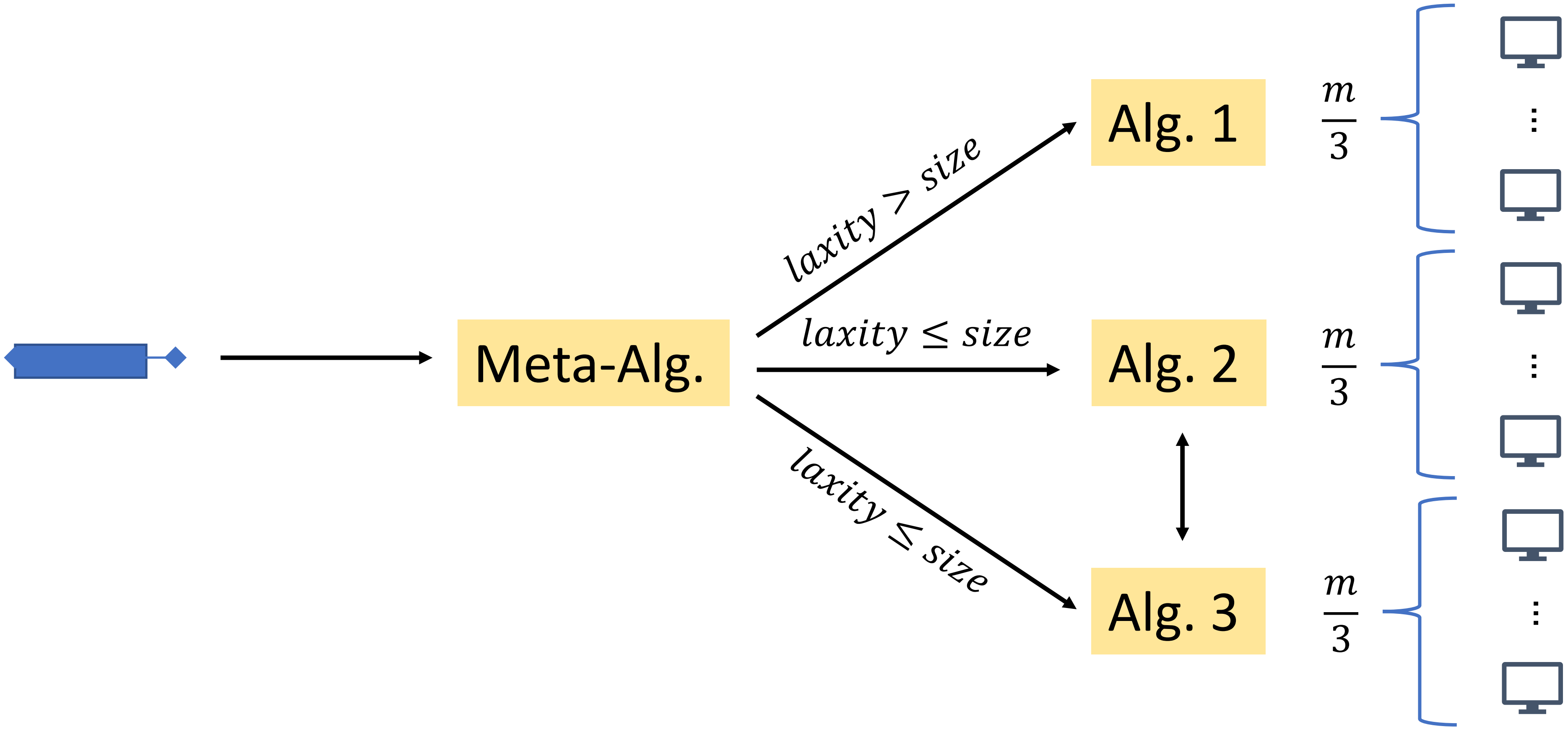
Idea: Meta-Alg. distributes jobs to 3 Algs.
,each ‘responsible’ for different part of Opt

Algorithm Idea



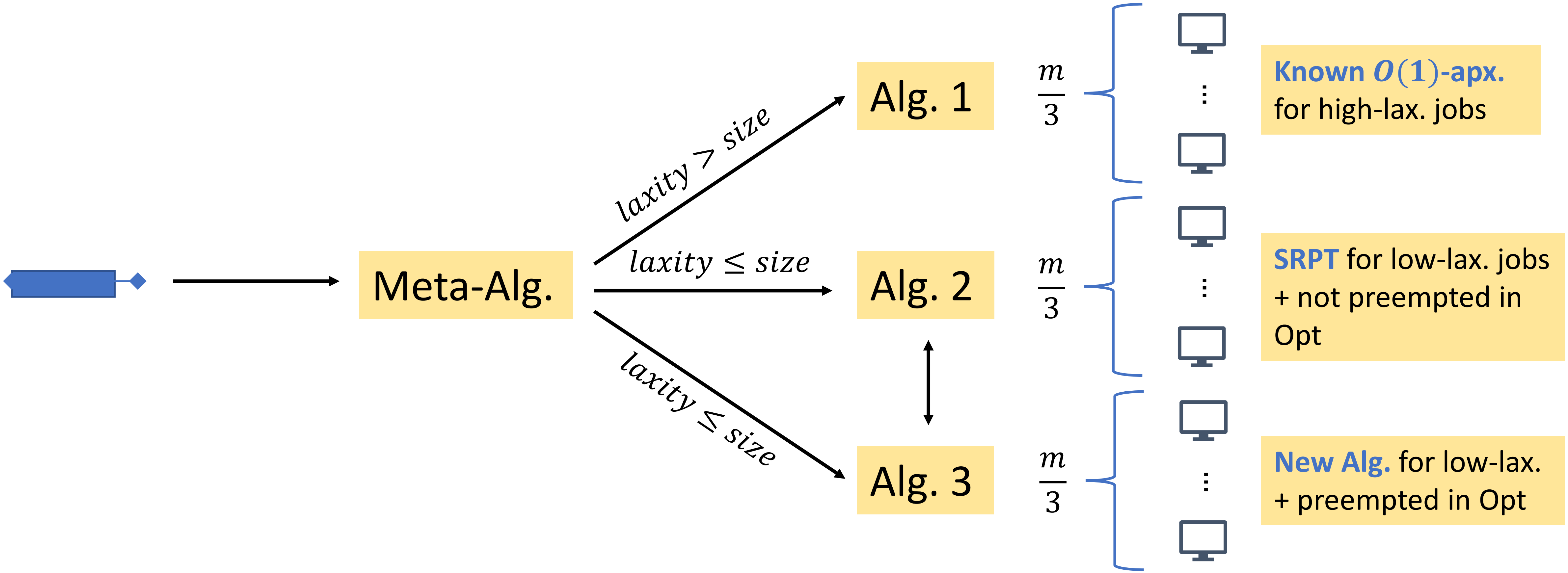
Idea: Meta-Alg. distributes jobs to 3 Algs.
,each 'responsible' for different part of Opt

Algorithm Idea



Idea: Meta-Alg. distributes jobs to 3 Algs. ,each 'responsible' for different part of Opt

Algorithm Idea

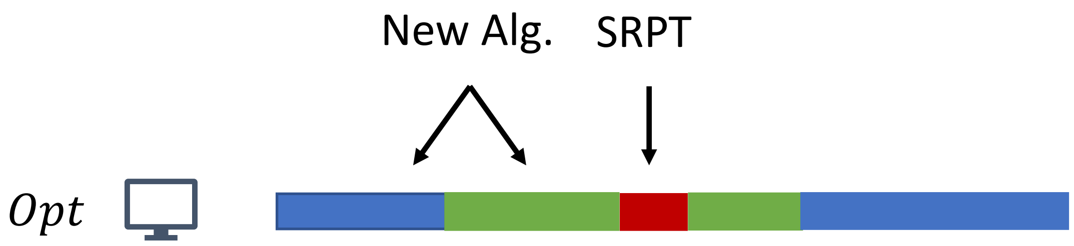


Idea: Meta-Alg. distributes jobs to 3 Algs.
,each ‘responsible’ for different part of Opt

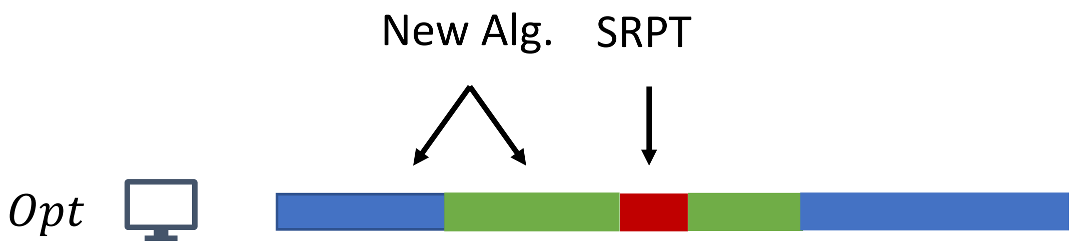
Low-Laxity Jobs

Opt

Low-Laxity Jobs

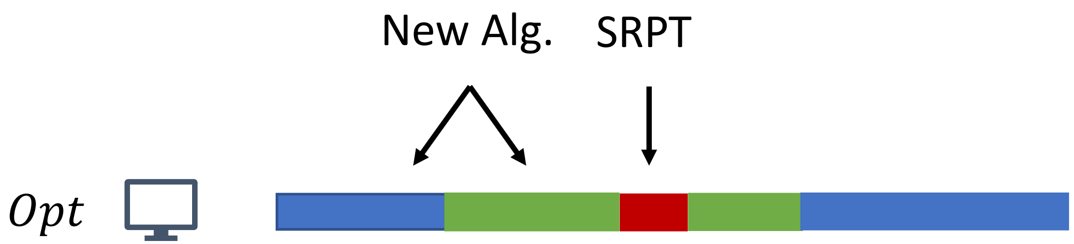


Low-Laxity Jobs



- **SRPT Sketch:**
  - If SRPT also completes red ⇒ ☺
  - Else when Opt runs red, SRPT must be running m jobs with even shorter remaining size than red ⇒ ☺

Low-Laxity Jobs



- **SRPT Sketch:**
  - If SRPT also completes red ⇒ ☺
  - Else when Opt runs red, SRPT must be running m jobs with even shorter remaining size than red ⇒ ☺

Main Challenge: How does **New Alg.** handle jobs where $\boldsymbol{laxity \le size}$ **+ preempted by Opt?**

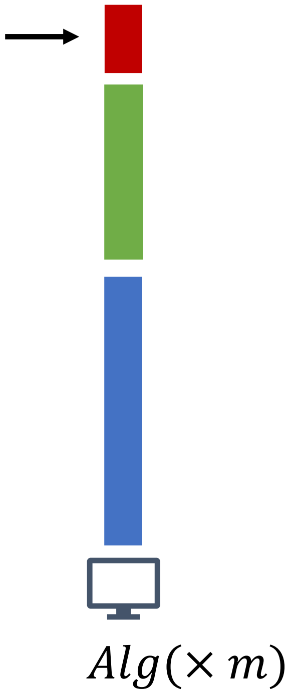
Alg. for Low-Laxity + Preempted

- **Idea:** each mach. maintains a stack of jobs $\sim$ chain of preemptions

$Alg(\times m)$

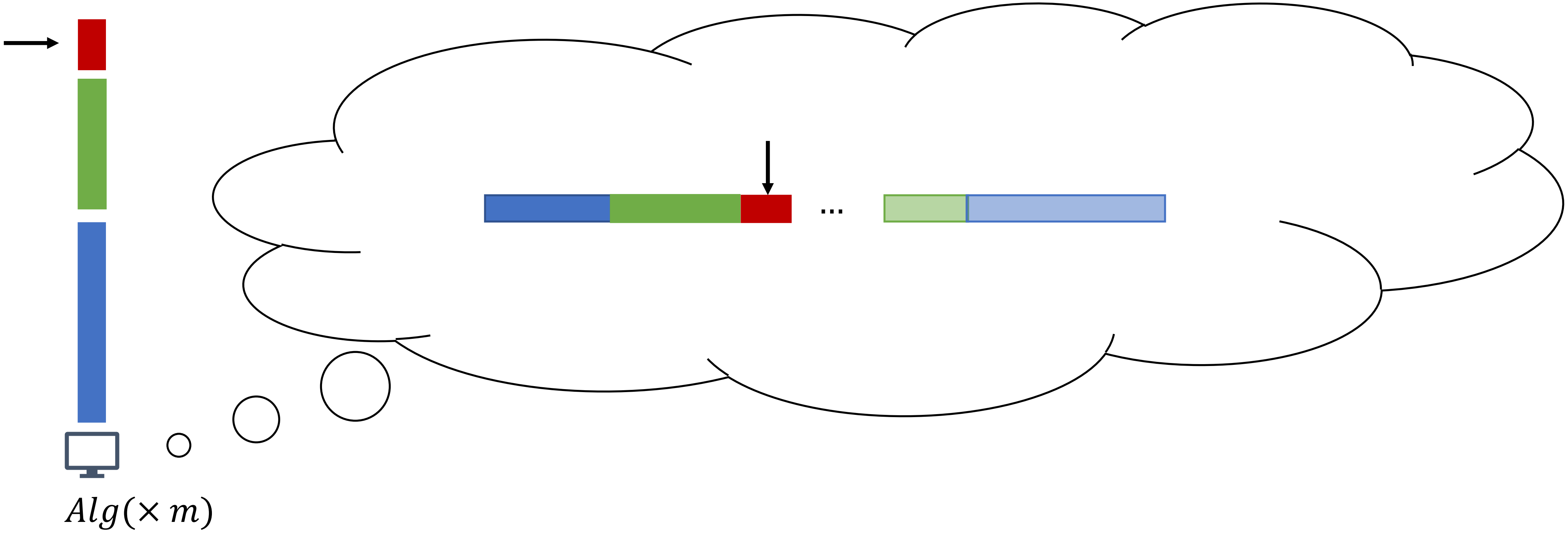
Alg. for Low-Laxity + Preempted

- **Idea:** each mach. maintains a stack of jobs $\sim$ chain of preemptions
- Always run top job of stack



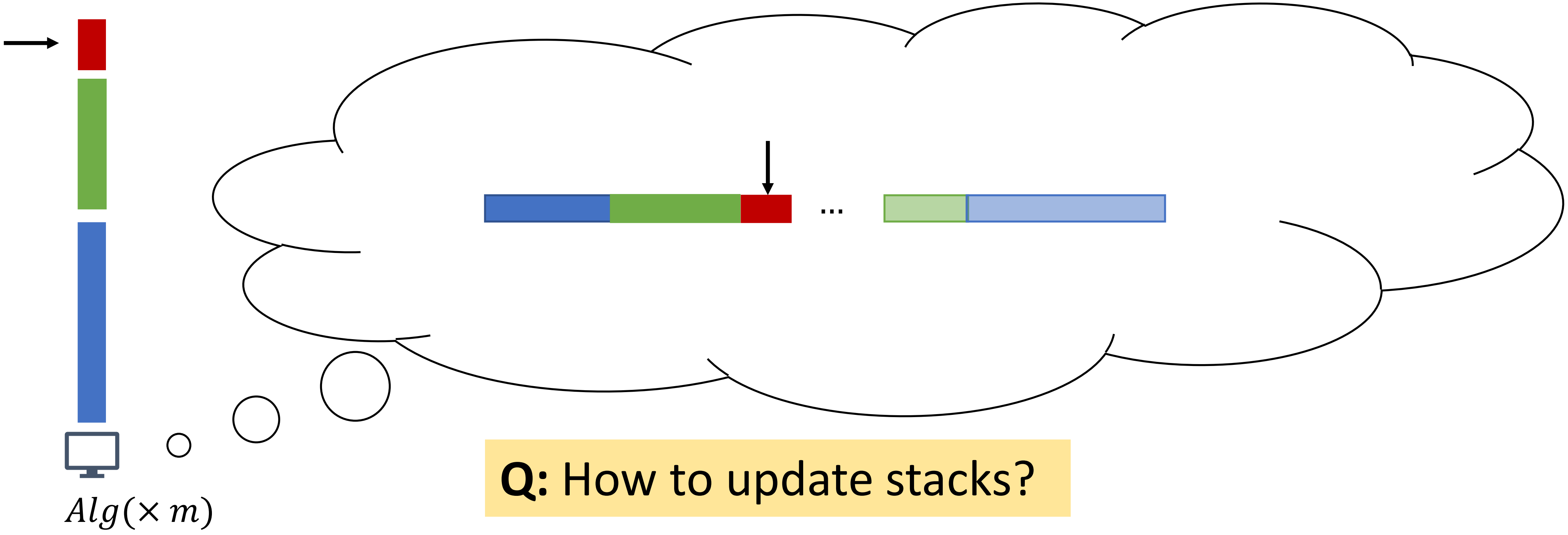
Alg. for Low-Laxity + Preempted

- **Idea:** each mach. maintains a stack of jobs $\sim$ chain of preemptions
- Always run top job of stack



Alg. for Low-Laxity + Preempted

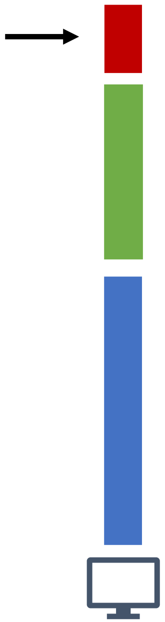
- **Idea:** each mach. maintains a stack of jobs $\sim$ chain of preemptions
- Always run top job of stack



Q: How to update stacks?

First Try – Job Release

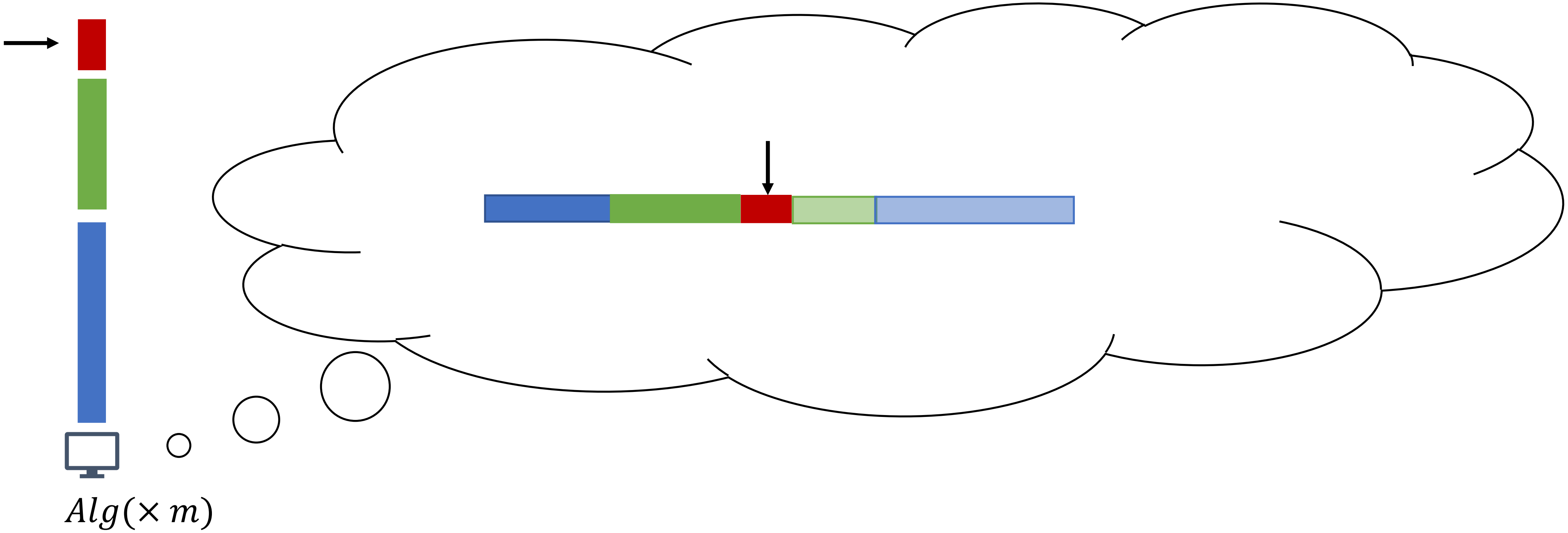
- When job j released:
  - If there exists stack with $size(j) \leq \epsilon \times laxity(top\ job\ of\ stack)$, then push j onto such a stack



$Alg(\times m)$

First Try – Job Release

- When job j released:
  - If there exists stack with $size(j) \leq \epsilon \times laxity(top\ job\ of\ stack)$, then push j onto such a stack



First Try – Alg.

- When job j released*:
  - If there exists stack with $size(j) \leq \epsilon \times laxity(top\ job\ of\ stack)$, then push j onto such a stack
- When job j completed:
  - Pop j off its stack; continue popping that stack until the top job can be feasibly completed

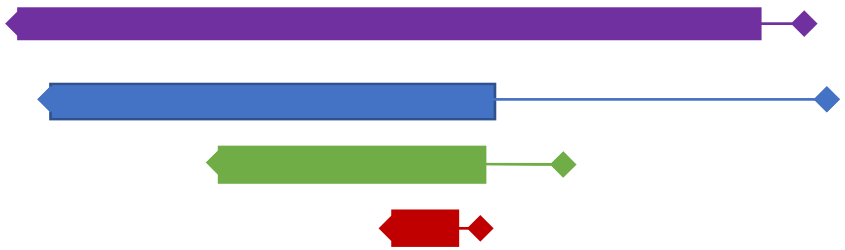
First Try – Alg.

- When job j released*:
  - If there exists stack with $size(j) \leq \epsilon \times laxity(top\ job\ of\ stack)$, then push j onto such a stack
- When job j completed:
  - Pop j off its stack; continue popping that stack until the top job can be feasibly completed

*consider modified release times

First Try – Alg.

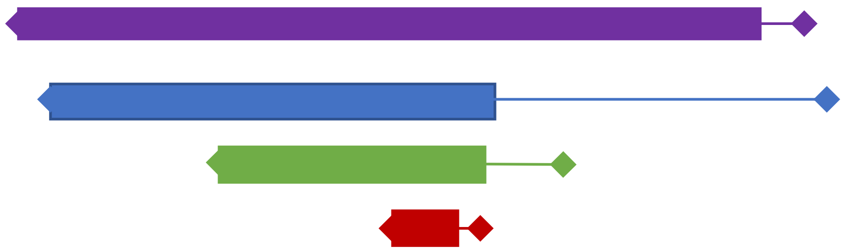
- When job j released*:
  - If there exists stack with $size(j) \leq \epsilon \times laxity(top\ job\ of\ stack)$, then push j onto such a stack
- When job j completed:
  - Pop j off its stack; continue popping that stack until the top job can be feasibly completed



*consider modified release times

First Try – Alg.

- When job j released*:
  - If there exists stack with $size(j) \leq \epsilon \times laxity(top\ job\ of\ stack)$, then push j onto such a stack
- When job j completed:
  - Pop j off its stack; continue popping that stack until the top job can be feasibly completed

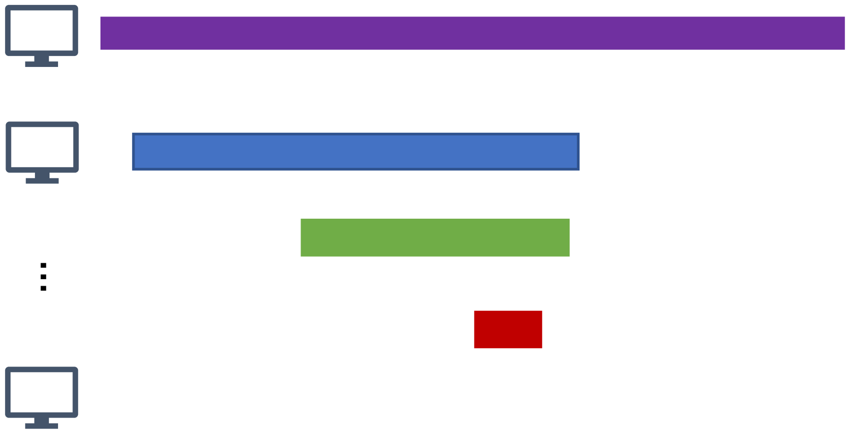


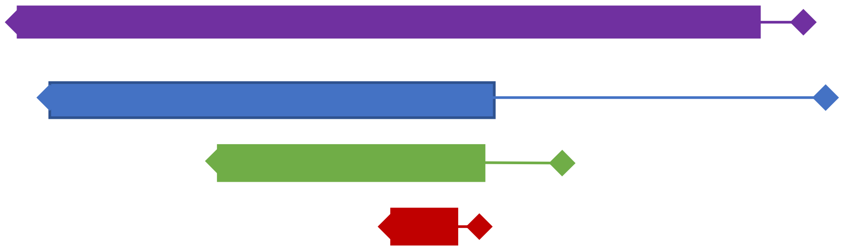
Need to sometimes replace current job

*consider modified release times

Final Alg. – Replacements

- Recall: m machines



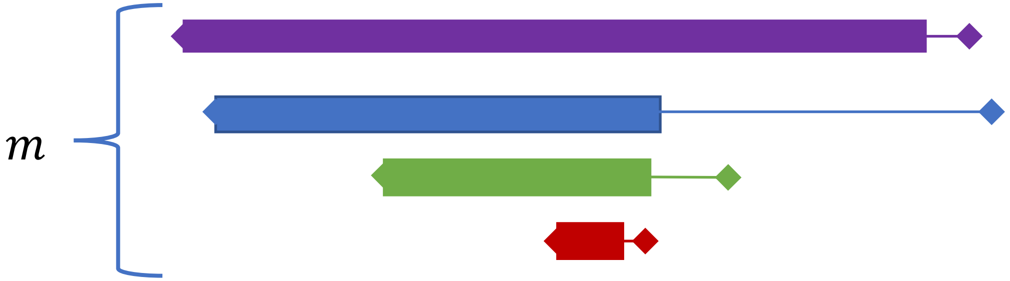


Final Alg. – Replacements

- Recall: m machines

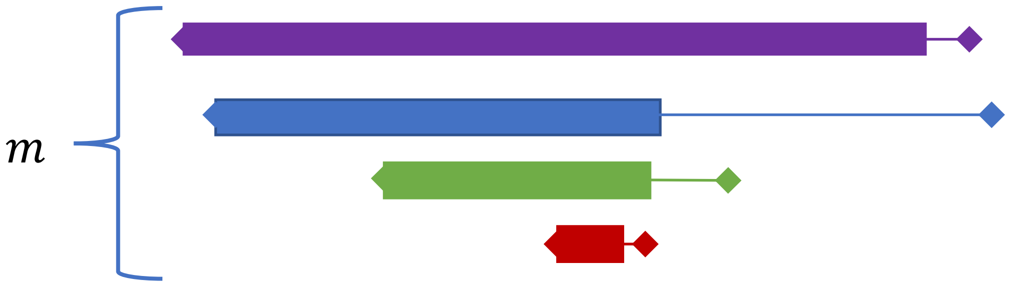
Final Alg. – Replacements

- Recall: m machines



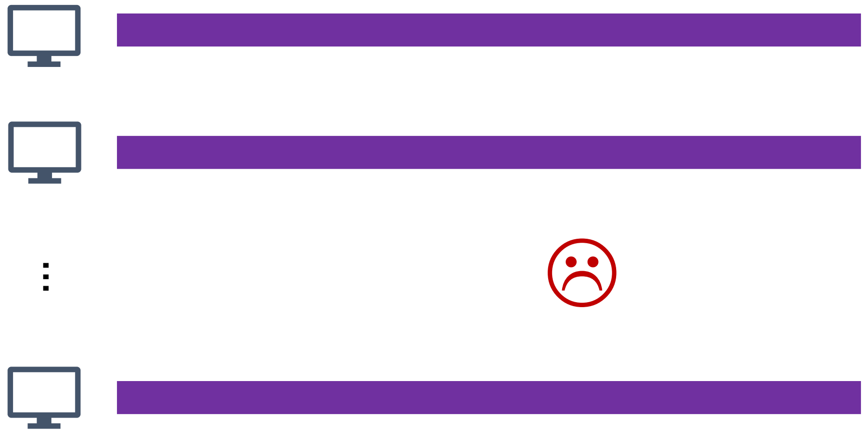
Final Alg. – Replacements

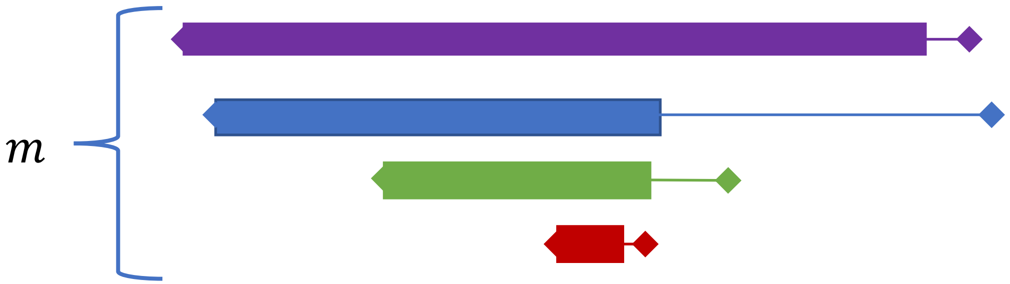
- Recall: m machines



Final Alg. – Replacements

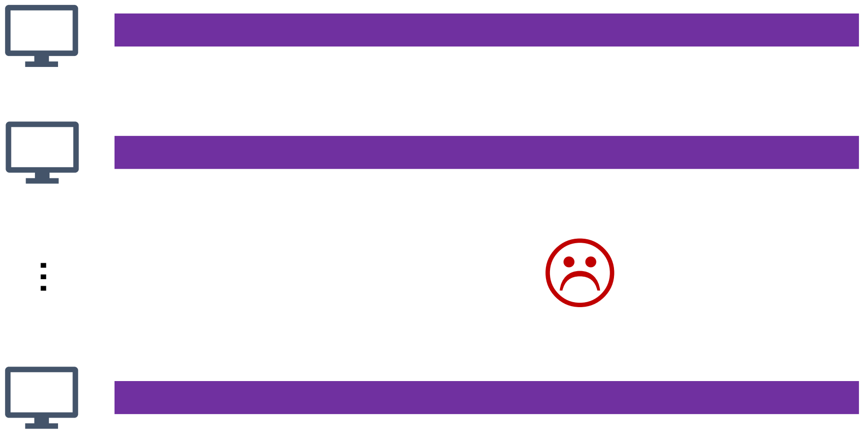
- Recall: m machines

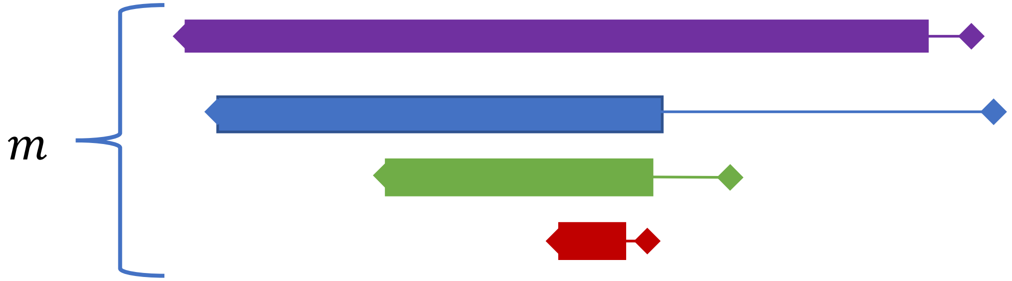




Final Alg. – Replacements

- Recall: m machines





Idea: If new job is much better than majority of the stacks, then replace

Final Alg. – Replacements

- When job j released*:
  - If there exists stack with $size(j) \leq \epsilon \times laxity(top\ job\ of\ stack)$, then push j onto such a stack
  - Else if $\Omega(m)$ stacks satisfy $size(j) \leq \epsilon \times laxity(second\ job\ of\ stack)$ and $laxity(j) > laxity(top\ job\ of\ stack)$ for some such stack, then replace* the top job with j
- When job j completed:
  - Pop j off its stack; continue popping that stack until the top job can be feasibly completed

Final Alg. – Replacements

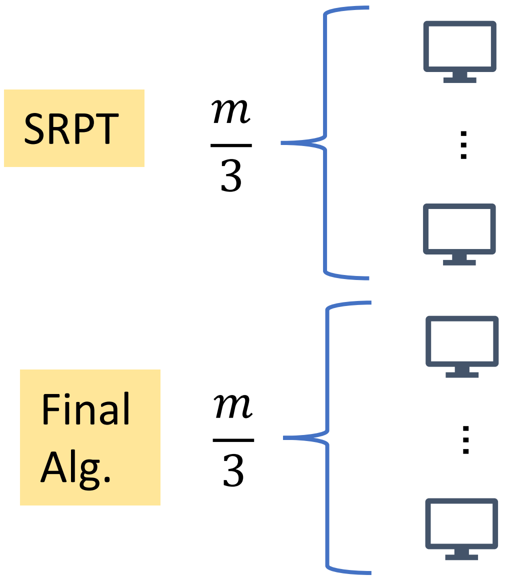
- When job j released*:
  - If there exists stack with $size(j) \leq \epsilon \times laxity(top\ job\ of\ stack)$, then push j onto such a stack
  - Else if $\Omega(m)$ stacks satisfy $size(j) \leq \epsilon \times laxity(second\ job\ of\ stack)$ and $laxity(j) > laxity(top\ job\ of\ stack)$ for some such stack, then replace* the top job with j
- When job j completed:
  - Pop j off its stack; continue popping that stack until the top job can be feasibly completed

*consider modified release times

*some tie-breaking rules apply

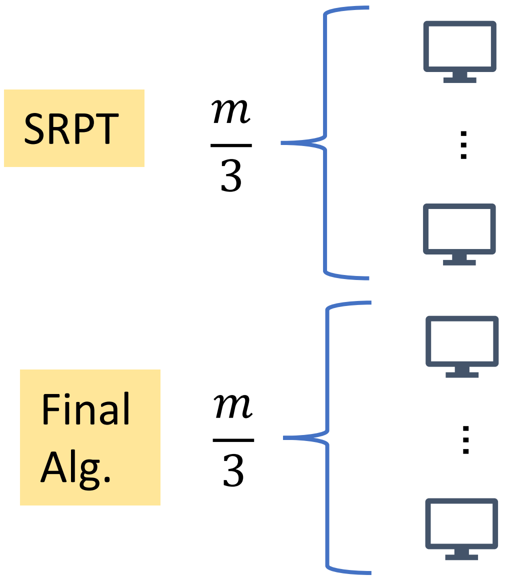
Analysis Sketch

- Show # $(Pushes\ by\ Final\ Alg.) = \Omega(\#(Completions\ by\ Opt))$



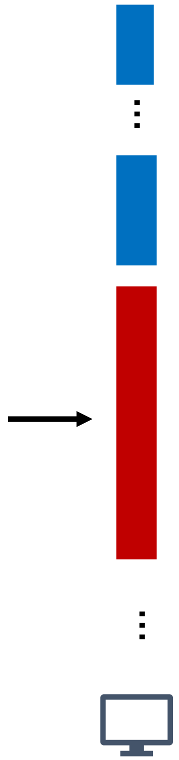
Analysis Sketch

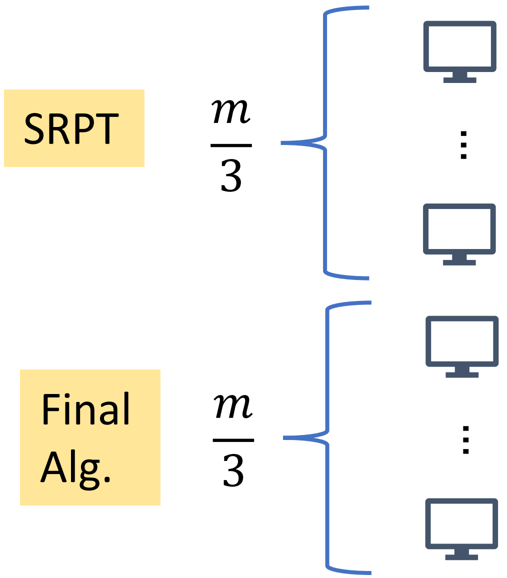
- Show # $(Pushes\ by\ Final\ Alg.) = \Omega(\#(Completions\ by\ Opt))$
- Show do not pop many jobs due to being infeasible



Analysis Sketch

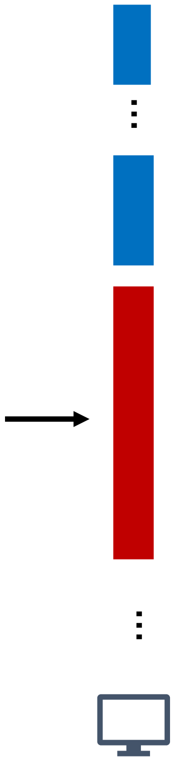
- Show # $(Pushes\ by\ Final\ Alg.) = \Omega(\#(Completions\ by\ Opt))$
- Show do not pop many jobs due to being infeasible
  - If job j is popped due to being infeasible:
    - $\Rightarrow$ many pushes/replacements on j's stack

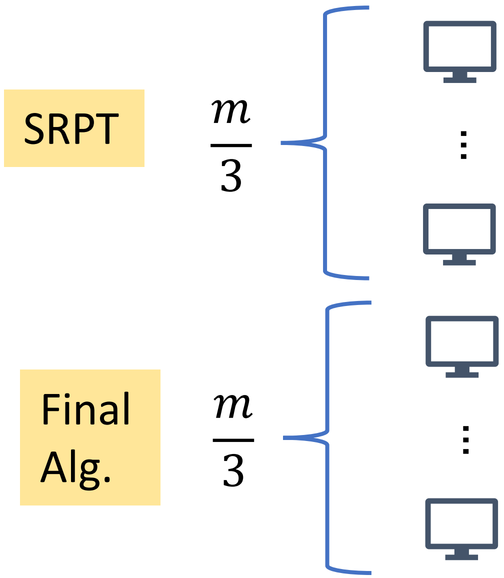




Analysis Sketch

- Show # $(Pushes\ by\ Final\ Alg.) = \Omega(\#(Completions\ by\ Opt))$
- Show do not pop many jobs due to being infeasible
  - If job j is popped due to being infeasible:
    - $\Rightarrow$ many pushes/replacements on j's stack
      - If many pushes, then charge to completion of those pushes
      - If many replacements, then can find long interval where Final Alg. is running jobs much smaller than $j \Rightarrow$ witness that SRPT is running even better jobs





Summary

- A deterministic $O(1)$-competitive alg. for Online Throughput Maximization on $m > 1$ machines.
- Concludes 20-year line of research since $m = 1$ case settled
- **Algorithm:** Run 3 algs. on $m/3$ machines each

Summary

- A deterministic $O(1)$-competitive alg. for Online Throughput Maximization on $m > 1$ machines.
- Concludes 20-year line of research since $m = 1$ case settled
- **Algorithm:** Run 3 algs. on $m/3$ machines each

Open Question: Can make Alg. non-migratory?