# Online Algorithms
# Tutorial 2 — Scheduling

Book chapter:
`https://link.springer.com/chapter/10.1007/BFb0029570`

# Finalizing Bin Packing

Remarks regarding Bin Packing:

- First Fit 1.7-competitive
- Best possible for Any Fit algorithms
- But there are better algorithms!
- Lower bound for any algorithm: 1.5401 (LP technique)
- Idea: Categorize items by size
- Harmonic algorithm uses categories $(\frac{1}{2}, 1], (\frac{1}{3}, \frac{1}{2}], \ldots$
- Next Fit within categories
- With sufficient categories, better than 1.7
- Better algorithms: More categories, more complex packing
- Currently: $1.57829\ldots$ (Advanced Harmonic) (2018)
- Current best lower bound: $1.54278\ldots$ (2020)

# Randomized Online Algorithm Adversaries

**Oblivious adversary**

- Adversary knows $A$
- Adversary generates $\sigma$ and optimal offline solution $\mathsf{OPT}(\sigma)$
- $A$ runs on $\sigma$, generating $A(\sigma)$

$$c = \sup_{\sigma} \frac{\mathbb{E}(A(\sigma))}{\mathsf{OPT}(\sigma)}$$

# Randomized Online Algorithm Adversaries

**Oblivious adversary**

- Adversary knows $A$
- Adversary generates $\sigma$ and optimal offline solution $\mathsf{OPT}(\sigma)$
- $A$ runs on $\sigma$, generating $A(\sigma)$

$$c = \sup_{\sigma} \frac{\mathbb{E}(A(\sigma))}{\mathsf{OPT}(\sigma)}$$

**Adaptive online adversary**

- Adversary knows $A$
- While not done:
  - Adversary generates request $\sigma_i$
  - $A$ is given $\sigma_i$
  - Adversary learns response and state of $A$
  - Adversary responds to $\sigma_i$
  - Next input request or end

# Randomized Online Algorithm Adversaries

Which of these adversaries is stronger?

# Randomized Online Algorithm Adversaries

Which of these adversaries is stronger?

Clearly, adaptive online can simulate oblivious beforehand!

# Randomized Online Algorithm Adversaries

Which of these adversaries is stronger?

Clearly, adaptive online can simulate oblivious beforehand!

Did we already see adaptive online adversaries in the lecture?

**Claim from the lecture:**

Any (deterministic or randomized) online file migration algorithm has a competitive ratio of at least $3$.

**Claim from the lecture:**
Any (deterministic or randomized) online file migration algorithm has a competitive ratio of at least $3$.

How did the proof go? What was the input sequence?

**Claim from the lecture:**
Any (deterministic or randomized) online file migration algorithm has a competitive ratio of at least $3$.

How did the proof go? What was the input sequence?

Always request the file where the algorithm does not have it.

# Randomized File Migration

**Claim from the lecture:**
Any (deterministic or randomized) online file migration algorithm has a competitive ratio of at least $3$.

How did the proof go? What was the input sequence?

Always request the file where the algorithm does not have it.

Can an oblivious adversary do that?

# Randomized File Migration

**Claim from the lecture:**
Any (deterministic or randomized) online file migration algorithm has a competitive ratio of at least $3$.

How did the proof go? What was the input sequence?

Always request the file where the algorithm does not have it.

Can an oblivious adversary do that?

No! Can an adaptive online algorithm do that?

# Randomized File Migration

**Claim from the lecture:**
Any (deterministic or randomized) online file migration algorithm has a competitive ratio of at least $3$.

How did the proof go? What was the input sequence?

Always request the file where the algorithm does not have it.

Can an oblivious adversary do that?

No! Can an adaptive online algorithm do that?

**Claim:**
Against an adaptive online adversary, any randomized online file migration algorithm has a competitive ratio of at least $3$.

# Online Scheduling

Another classic problem: Distribute jobs on machines

- $m$ machines $M_1, \ldots, M_m$, $m$ known
- $n$ jobs $J_1, \ldots, J_n$, $n$ unknown
- Running time $t(J_i) > 0$

# Online Scheduling

Another classic problem: Distribute jobs on machines

- $m$ machines $M_1, \ldots, M_m$, $m$ known
- $n$ jobs $J_1, \ldots, J_n$, $n$ unknown
- Running time $t(J_i) > 0$

Many different versions:

- Precedence constraints
- Release times
- Preemption
- Machine faults
- Unsure job running time
- Different machines (speed, possible jobs)
- Parallel jobs
- Minimum makespan
- Minimum waiting time, equal load, ...

# Online Scheduling

Natural online problem: nearly all variants NP-hard ($m \geq 2$)

Our variant:

- $m$ identical machines, $n$ jobs
- Running times $t(J_i)$
- Minimize makespan
- Assign job $J_i$ to some machine before getting $J_{i+1}$
- Somewhat similar to bin packing

# Online Scheduling

Natural online problem: nearly all variants NP-hard ($m \geq 2$)

Our variant:

- $m$ identical machines, $n$ jobs
- Running times $t(J_i)$
- Minimize makespan
- Assign job $J_i$ to some machine before getting $J_{i+1}$
- Somewhat similar to bin packing

Idea for simple online algorithm?

# Online Scheduling

Natural online problem: nearly all variants NP-hard ($m \geq 2$)

Our variant:

- $m$ identical machines, $n$ jobs
- Running times $t(J_i)$
- Minimize makespan
- Assign job $J_i$ to some machine before getting $J_{i+1}$
- Somewhat similar to bin packing

Idea for simple online algorithm?

List Scheduling

- Always put the next job on the machine with least load

# Online Scheduling

Natural online problem: nearly all variants NP-hard ($m \geq 2$)

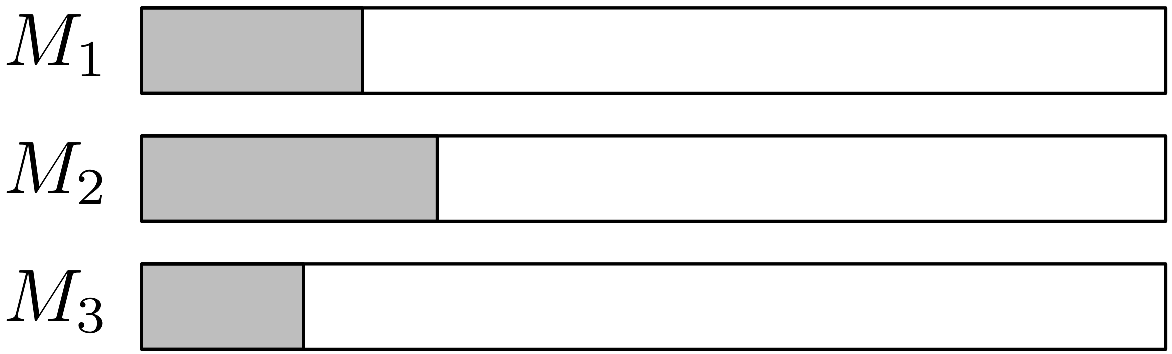
Our variant:

- $m$ identical machines, $n$ jobs
- Running times $t(J_i)$
- Minimize makespan
- Assign job $J_i$ to some machine before getting $J_{i+1}$
- Somewhat similar to bin packing

Idea for simple online algorithm?

List Scheduling

- Always put the next job on the machine with least load



# Online Scheduling

Natural online problem: nearly all variants NP-hard ($m \geq 2$)

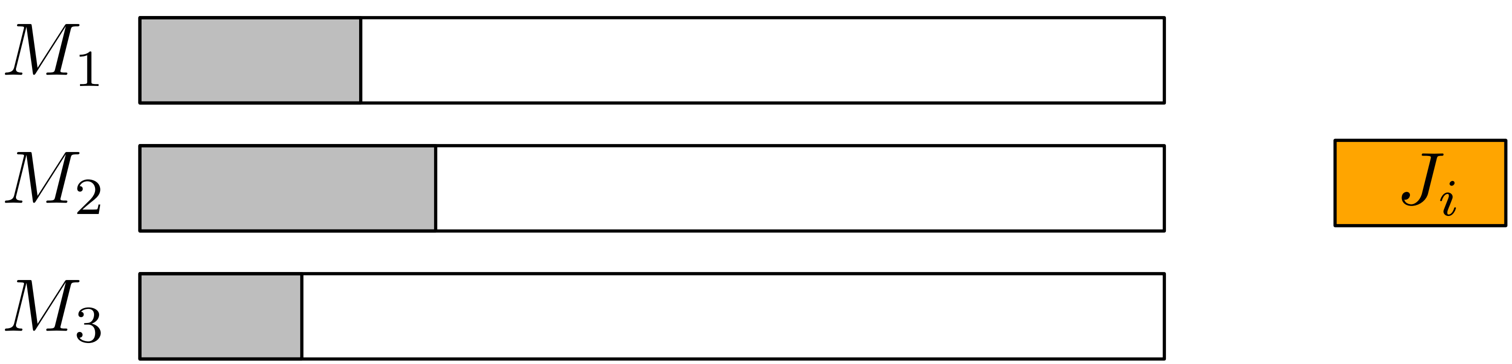
Our variant:

- $m$ identical machines, $n$ jobs
- Running times $t(J_i)$
- Minimize makespan
- Assign job $J_i$ to some machine before getting $J_{i+1}$
- Somewhat similar to bin packing

Idea for simple online algorithm?

List Scheduling

- Always put the next job on the machine with least load



# Online Scheduling

Natural online problem: nearly all variants NP-hard ($m \geq 2$)

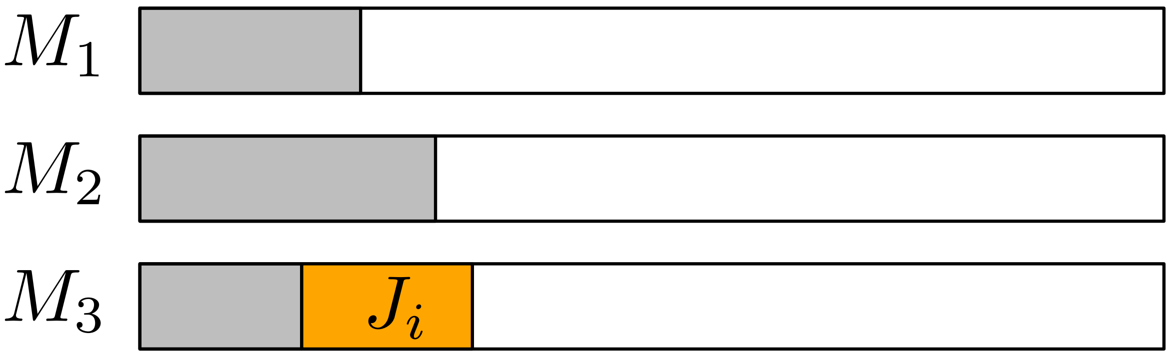
Our variant:

- $m$ identical machines, $n$ jobs
- Running times $t(J_i)$
- Minimize makespan
- Assign job $J_i$ to some machine before getting $J_{i+1}$
- Somewhat similar to bin packing

Idea for simple online algorithm?

List Scheduling

- Always put the next job on the machine with least load



# Online Scheduling

Natural online problem: nearly all variants NP-hard ($m \geq 2$)

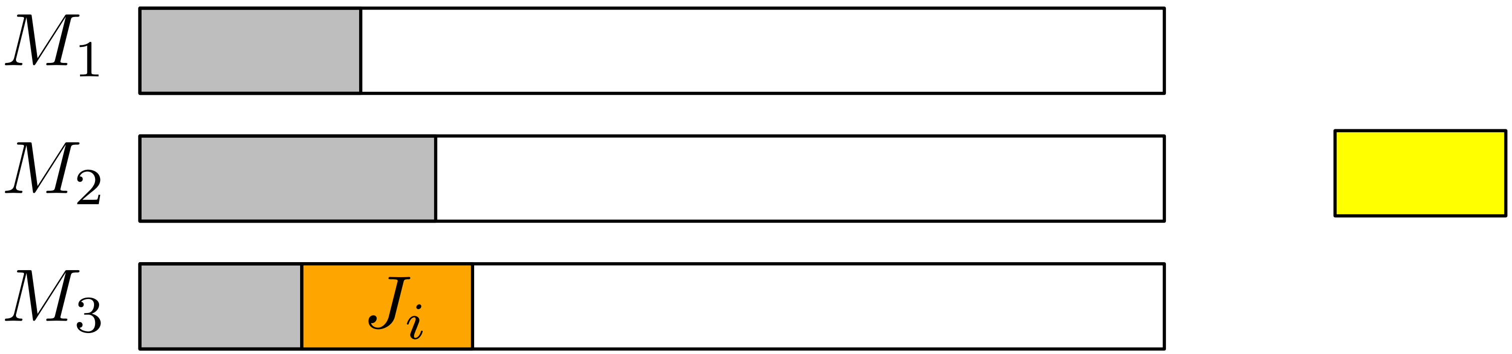
Our variant:

- $m$ identical machines, $n$ jobs
- Running times $t(J_i)$
- Minimize makespan
- Assign job $J_i$ to some machine before getting $J_{i+1}$
- Somewhat similar to bin packing

Idea for simple online algorithm?

LIST SCHEDULING

- Always put the next job on the machine with least load



# Online Scheduling

Natural online problem: nearly all variants NP-hard ($m \geq 2$)

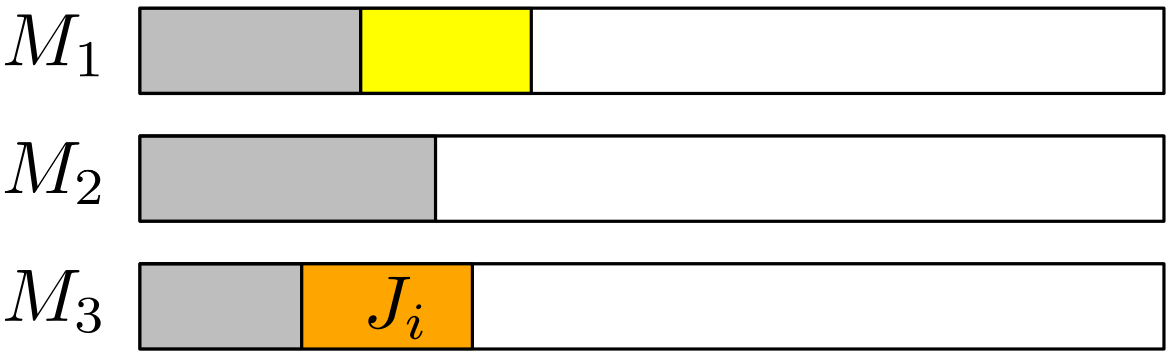
Our variant:

- $m$ identical machines, $n$ jobs
- Running times $t(J_i)$
- Minimize makespan
- Assign job $J_i$ to some machine before getting $J_{i+1}$
- Somewhat similar to bin packing

Idea for simple online algorithm?

List Scheduling

- Always put the next job on the machine with least load



# Online Scheduling

Natural online problem: nearly all variants NP-hard ($m \geq 2$)

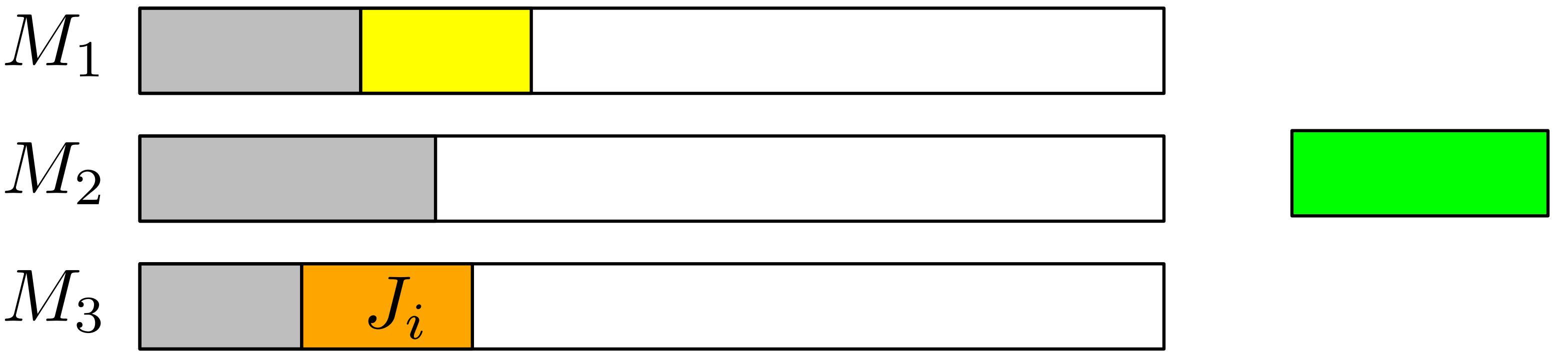
Our variant:

- $m$ identical machines, $n$ jobs
- Running times $t(J_i)$
- Minimize makespan
- Assign job $J_i$ to some machine before getting $J_{i+1}$
- Somewhat similar to bin packing

Idea for simple online algorithm?

List Scheduling

- Always put the next job on the machine with least load



# Online Scheduling

Natural online problem: nearly all variants NP-hard ($m \geq 2$)

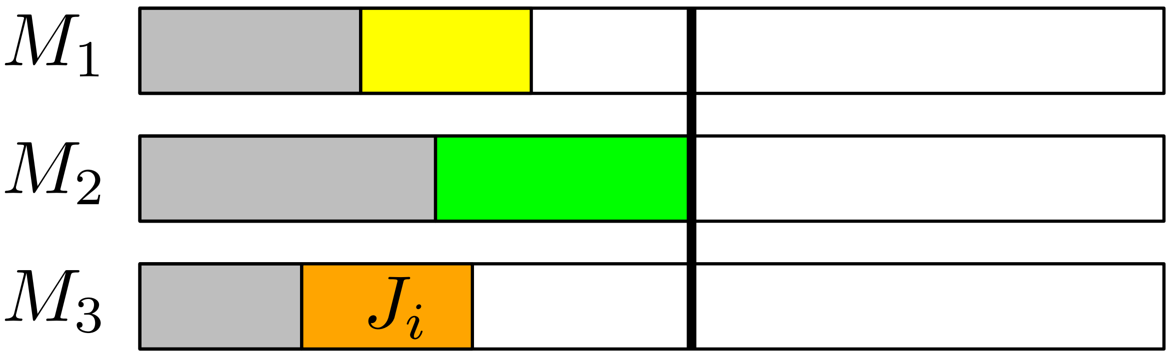
Our variant:

- $m$ identical machines, $n$ jobs
- Running times $t(J_i)$
- Minimize makespan
- Assign job $J_i$ to some machine before getting $J_{i+1}$
- Somewhat similar to bin packing

Idea for simple online algorithm?

LIST SCHEDULING

- Always put the next job on the machine with least load

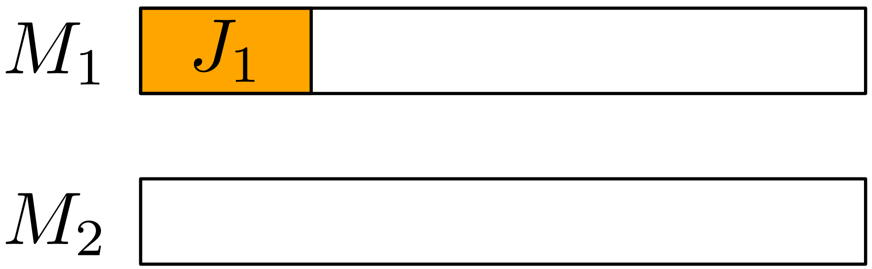


# List Scheduling: Competitive Ratio?

Competitive ratio for $m = 2$?

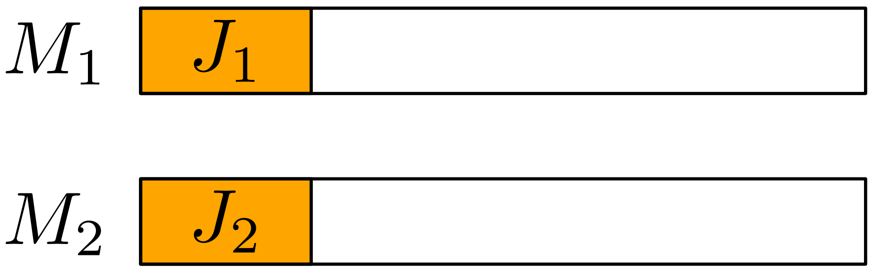
# List Scheduling: Competitive Ratio?

Competitive ratio for $m = 2$?



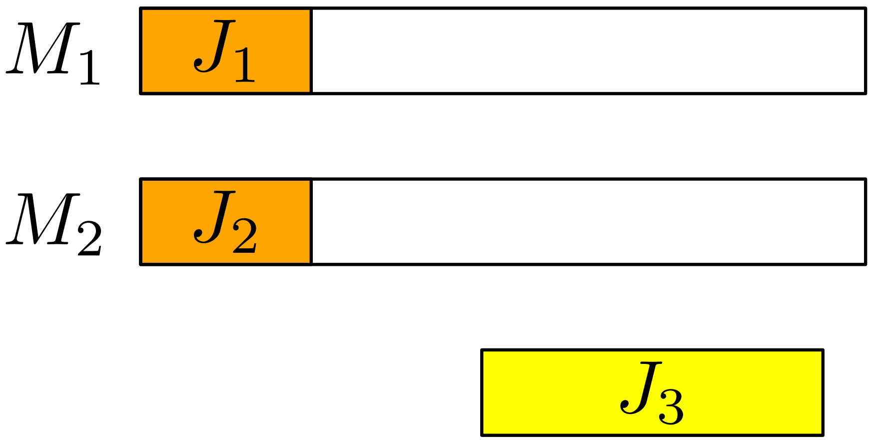
# List Scheduling: Competitive Ratio?

Competitive ratio for $m = 2$?



# List Scheduling: Competitive Ratio?

Competitive ratio for $m = 2$?



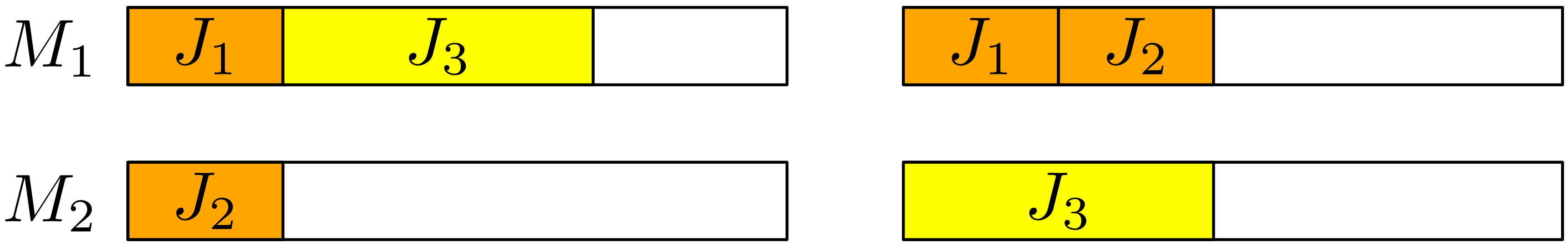
# List Scheduling: Competitive Ratio?

Competitive ratio for $m = 2$?


$$t(J_3) = 2t(J_1) = 2t(J_2)$$

# List Scheduling: Competitive Ratio?

Competitive ratio for $m = 2$?

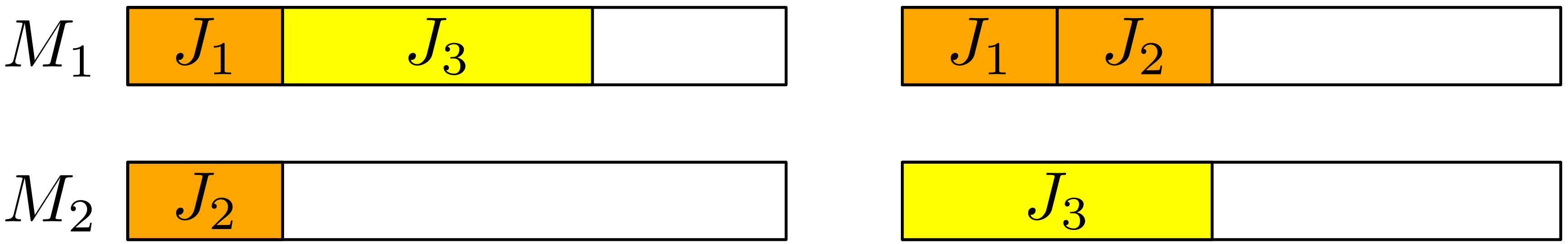



$t(J_3) = 2t(J_1) = 2t(J_2)$

OPT

# List Scheduling: Competitive Ratio?

Competitive ratio for $m = 2$?



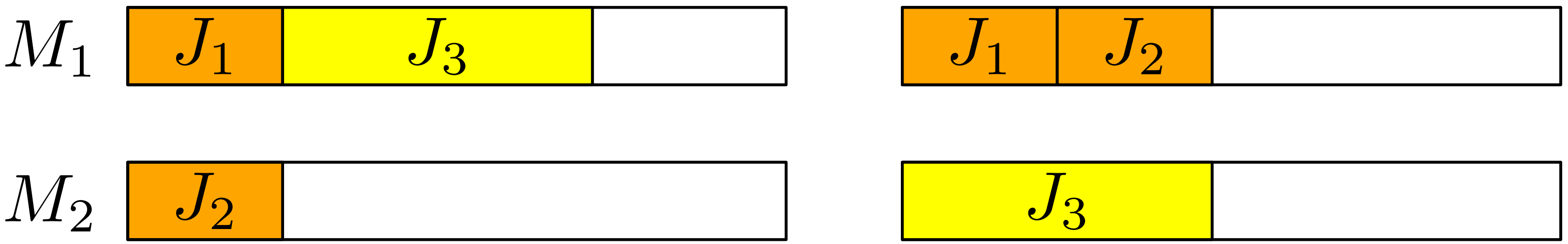

$t(J_3) = 2t(J_1) = 2t(J_2)$

OPT

Competitive ratio $^3/_2$

# List Scheduling: Competitive Ratio?

Competitive ratio for $m = 2$?




$t(J_3) = 2t(J_1) = 2t(J_2)$

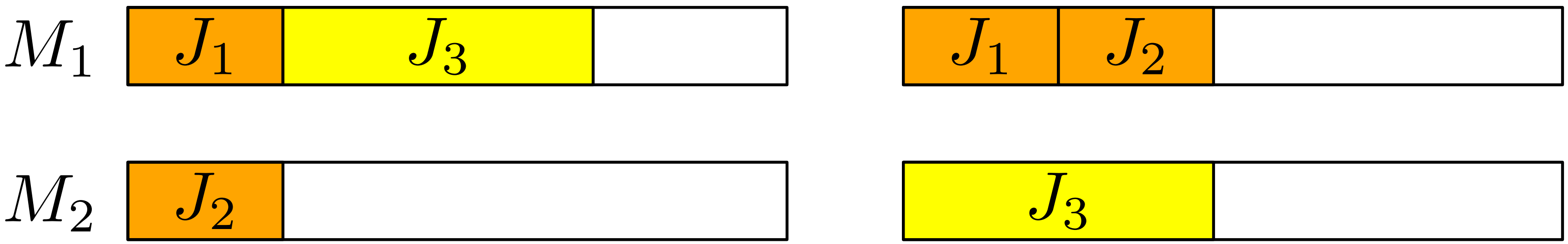
OPT

Competitive ratio $^3/_2$

Idea: We fill machines evenly, but should reserve one.

# List Scheduling: Competitive Ratio?

Competitive ratio for $m = 2$?




$t(J_3) = 2t(J_1) = 2t(J_2)$ OPT

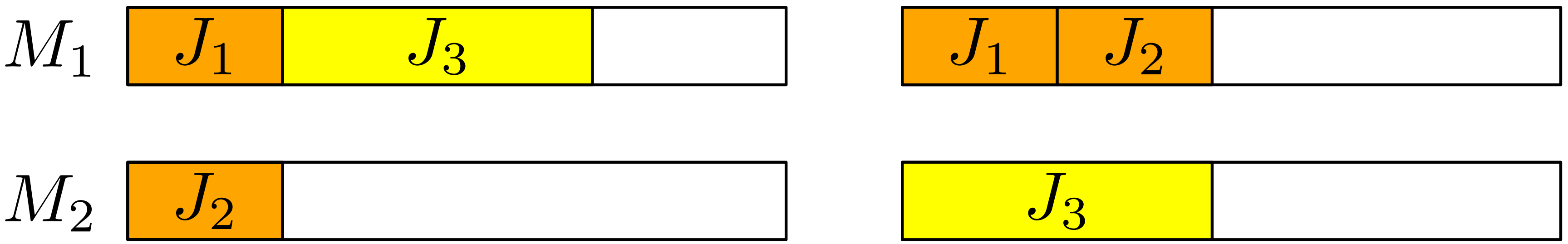
Competitive ratio $^3/_2$

Idea: We fill machines evenly, but should reserve one.

Arbitrary $m$: $m(m-1)$ jobs with time 1, 1 job with time $m$

# List Scheduling: Competitive Ratio?

Competitive ratio for $m = 2$?




$t(J_3) = 2t(J_1) = 2t(J_2)$ OPT

Competitive ratio $^3/_2$

Idea: We fill machines evenly, but should reserve one.

Arbitrary $m$: $m(m-1)$ jobs with time 1, 1 job with time $m$

Competitive ratio?

Our makespan: $(m-1)+m$, OPT: $m$, $c \geq 2 - {}^1/_m$

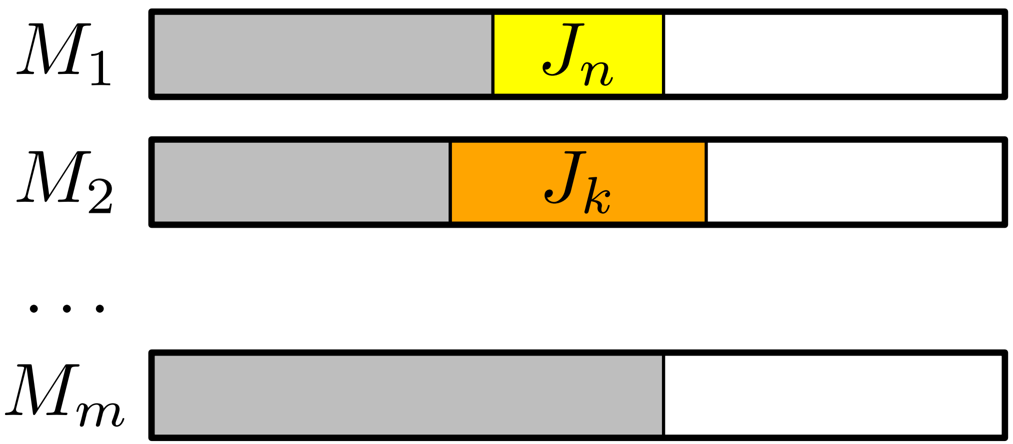
# List Scheduling — Competitive Ratio

Can it get worse?

# List Scheduling — Competitive Ratio

Can it get worse?

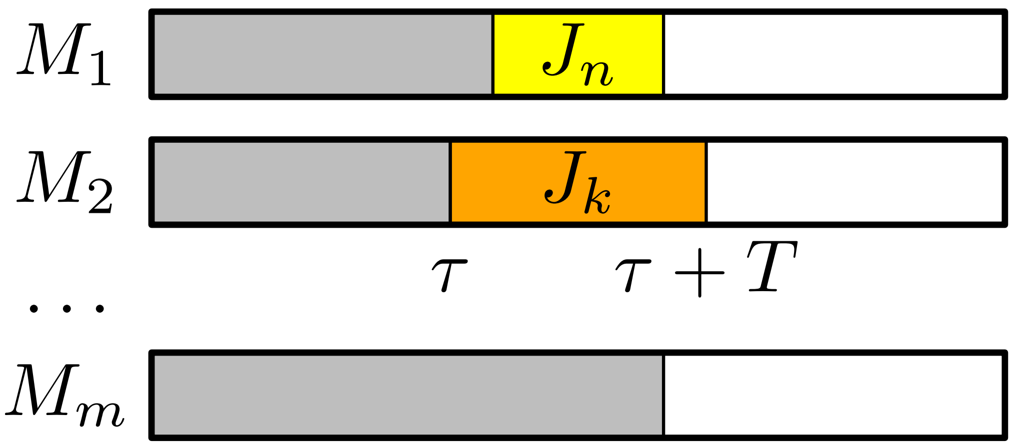
No! Proof: Consider $J_k$, the job that ends last



# List Scheduling — Competitive Ratio

Can it get worse?

No! Proof: Consider $J_k$, the job that ends last

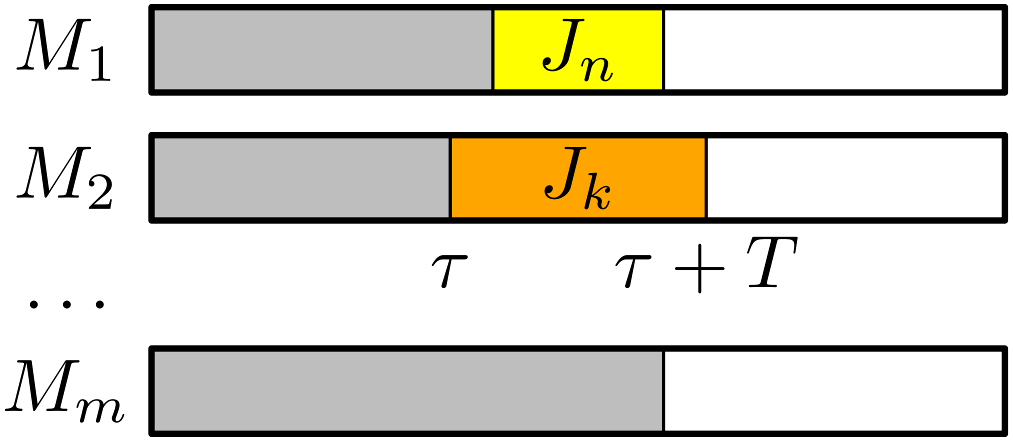



Time $\tau$: Starting time of $J_k$, $T = t(J_k)$

# List Scheduling — Competitive Ratio

Can it get worse?

No! Proof: Consider $J_k$, the job that ends last



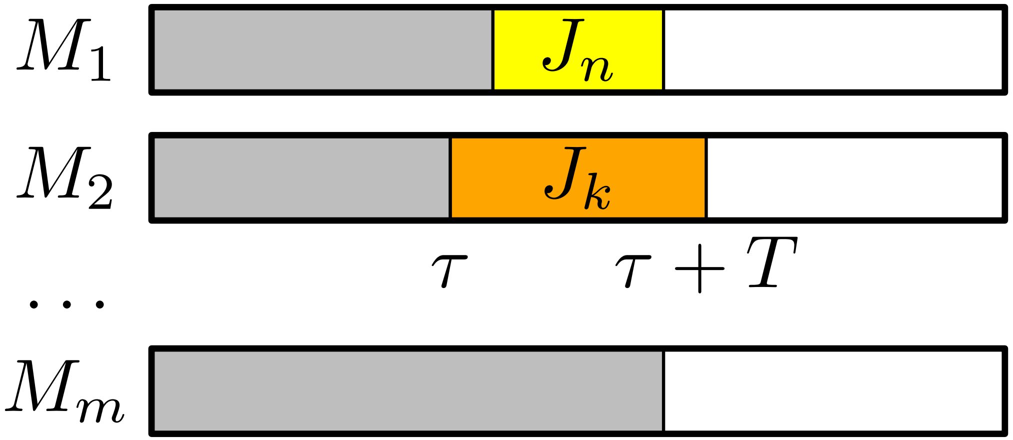

Time $\tau$: Starting time of $J_k$, $T = t(J_k)$

Up to $\tau$: All machines busy! Why?

# List Scheduling — Competitive Ratio

Can it get worse?

No! Proof: Consider $J_k$, the job that ends last



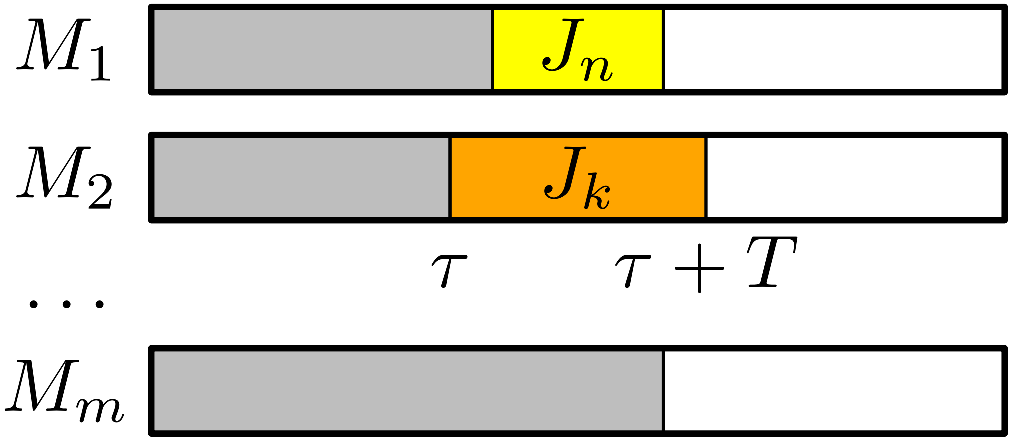

Time $\tau$: Starting time of $J_k$, $T = t(J_k)$

Up to $\tau$: All machines busy! Why? $\Rightarrow$ OPT $\geq \tau + \frac{T}{m}$. Why?

# List Scheduling — Competitive Ratio

Can it get worse?

No! Proof: Consider $J_k$, the job that ends last




Time $\tau$: Starting time of $J_k$, $T = t(J_k)$

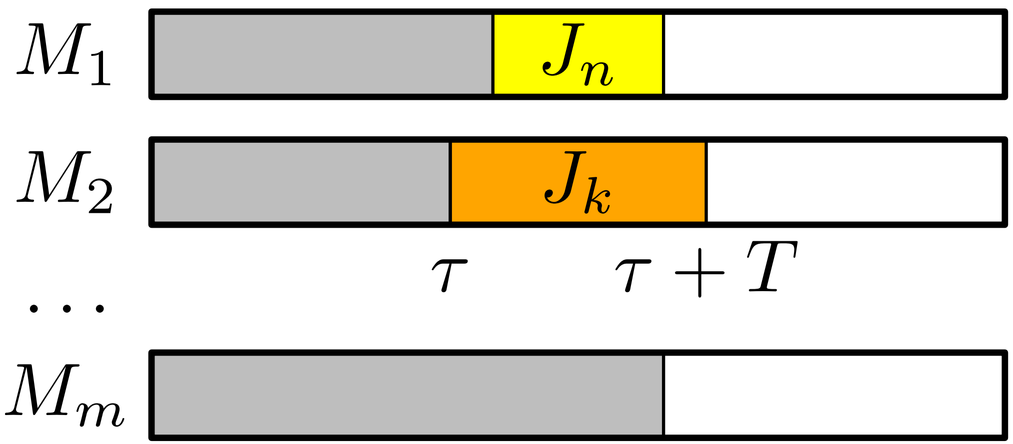
Up to $\tau$: All machines busy! Why? $\Rightarrow$ OPT $\geq \tau + \frac{T}{m}$. Why?

$T \leq \text{OPT} \Rightarrow \tau + T \leq \text{OPT} - \frac{T}{m} + \text{OPT} = \left(2 - \frac{1}{m}\right) \text{OPT}.$

# List Scheduling — Competitive Ratio

Can it get worse?

No! Proof: Consider $J_k$, the job that ends last




Time $\tau$: Starting time of $J_k$, $T = t(J_k)$

Up to $\tau$: All machines busy! Why? $\Rightarrow$ OPT $\geq \tau + \frac{T}{m}$. Why?

$T \leq \text{OPT} \Rightarrow \tau + T \leq \text{OPT} - \frac{T}{m} + \text{OPT} = \left(2 - \frac{1}{m}\right) \text{OPT}.$

Note: Both lower bounds on OPT tight in worst case!

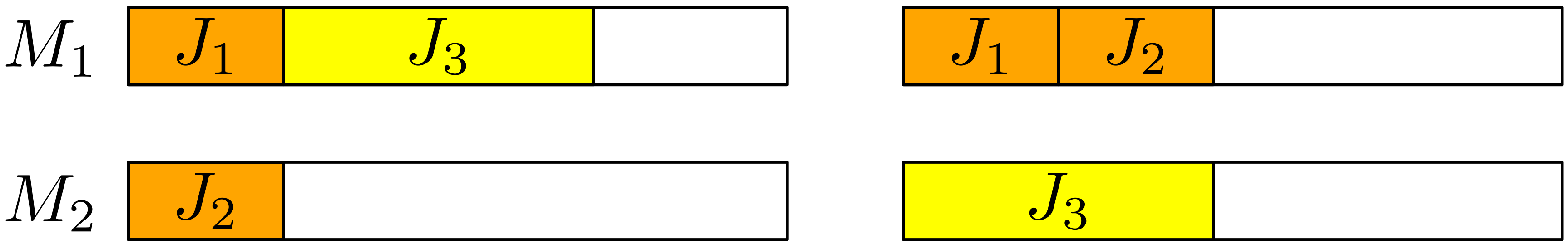
# More Complex Models

We can adapt this analysis to:

- Unknown running times
- Precedence constraints (analysis technical)
- Jobs with release times

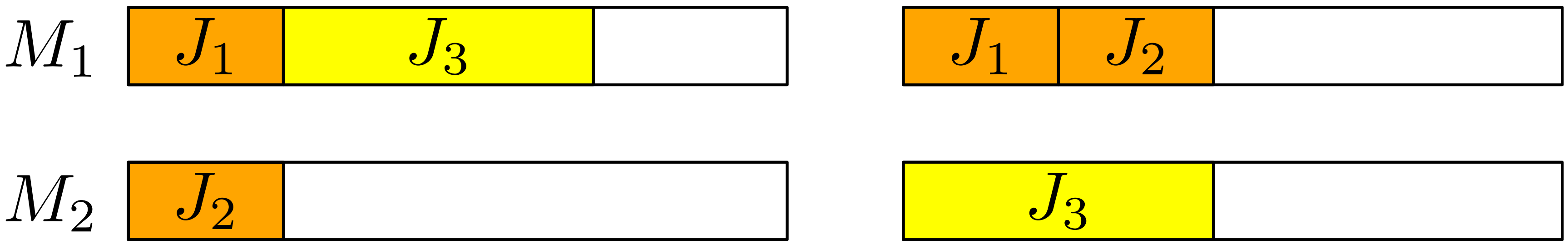
# Can we do better?

$m = 2$: No. Why?



# Can we do better?

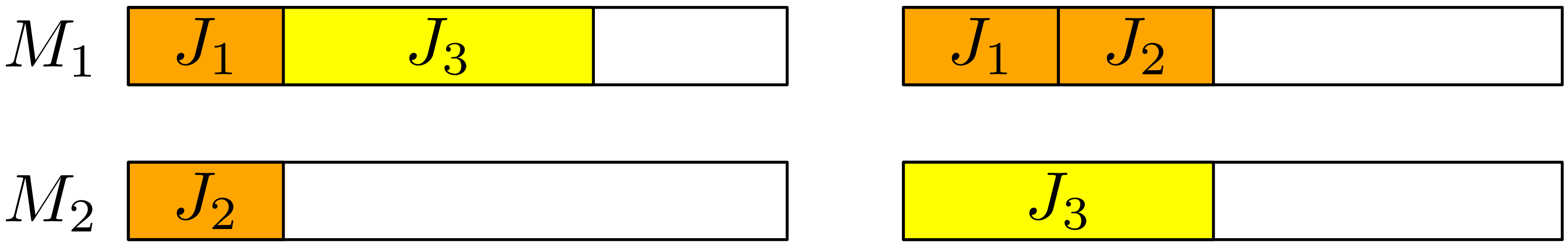
$m = 2$: No. Why?



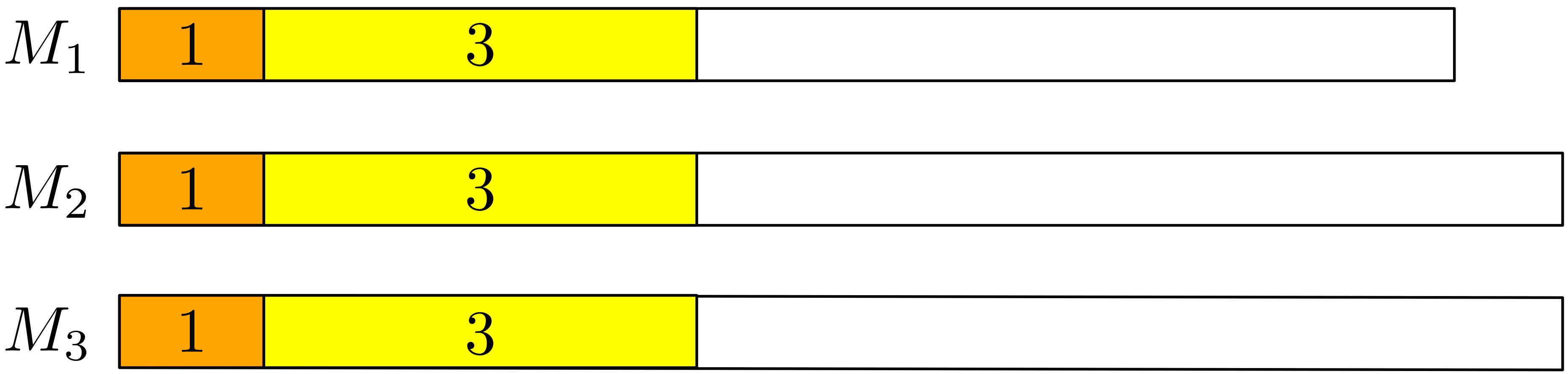

$m = 3$: No.

# Can we do better?

$m = 2$: No. Why?



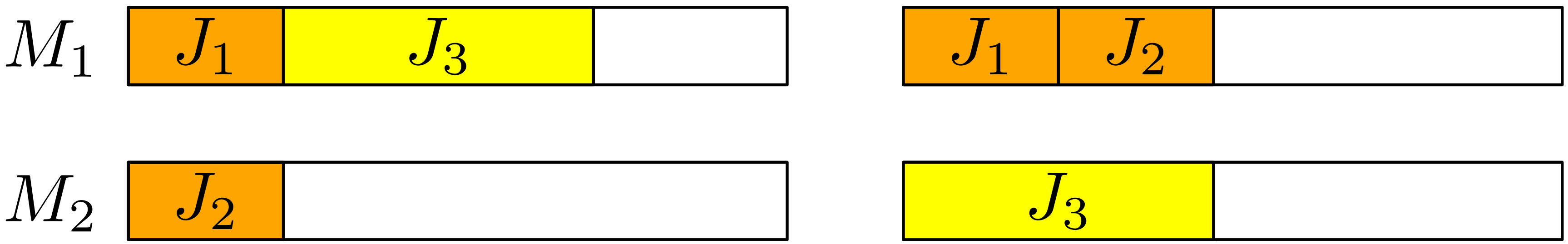

$m = 3$: No.



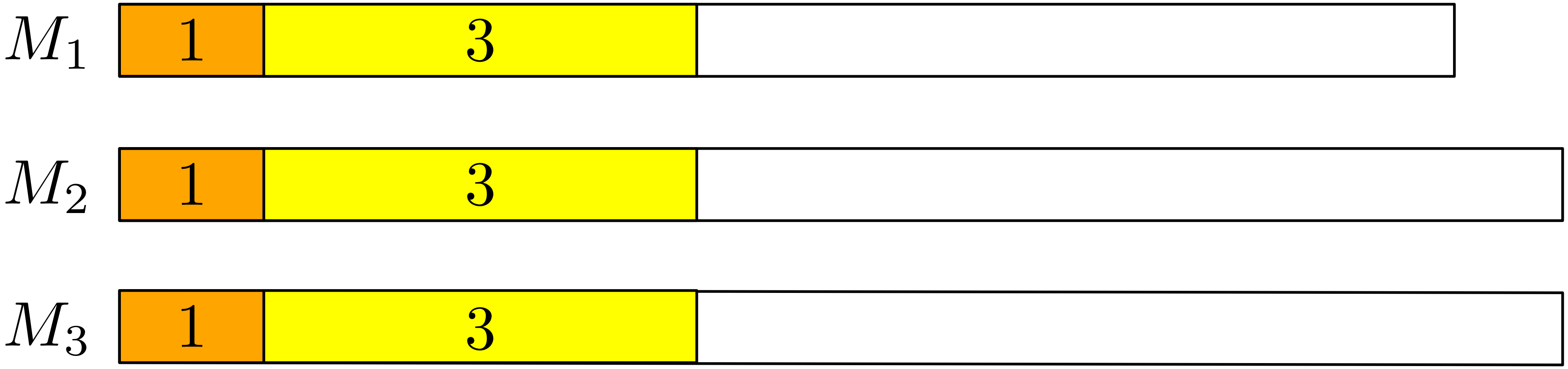

$$\frac{7}{4} > \frac{5}{3}$$

# Can we do better?

$m = 2$: No. Why?



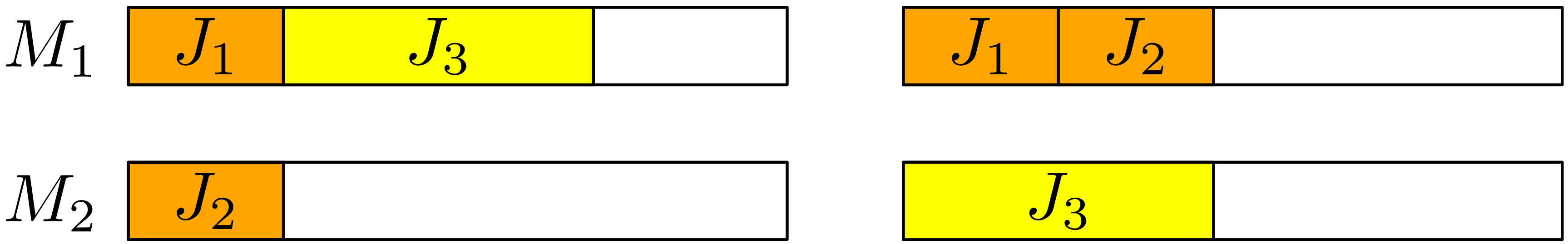

$m = 3$: No.



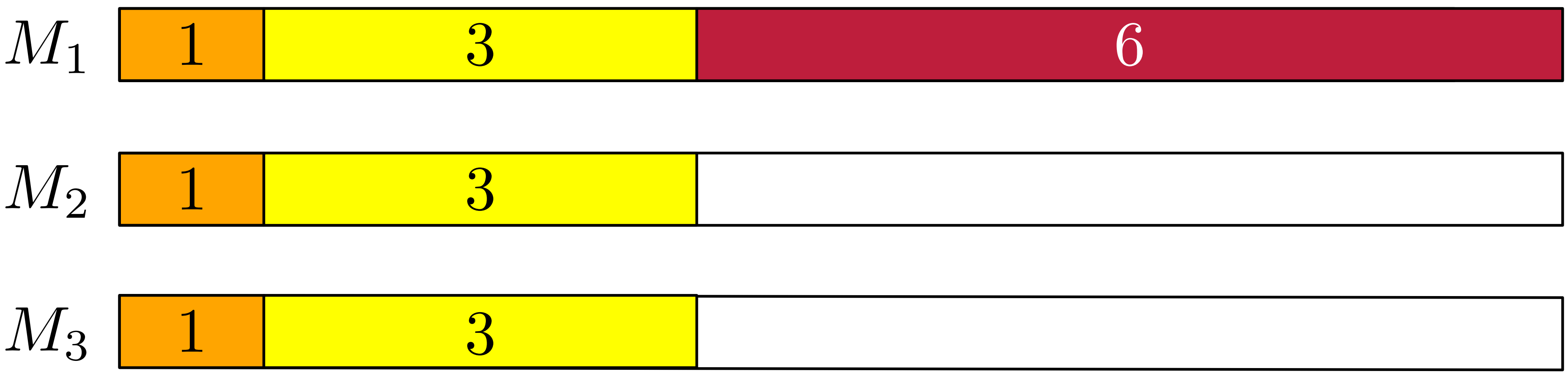

$\frac{7}{4} > \frac{5}{3}$ What item comes next?

# Can we do better?

$m = 2$: No. Why?



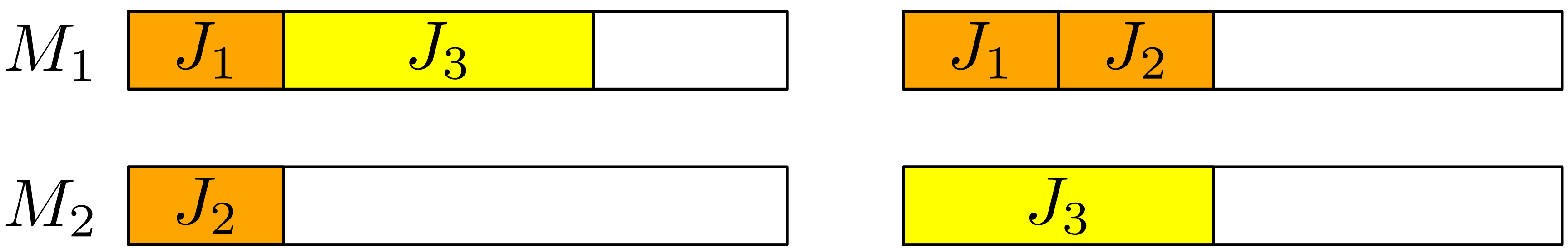

$m = 3$: No.



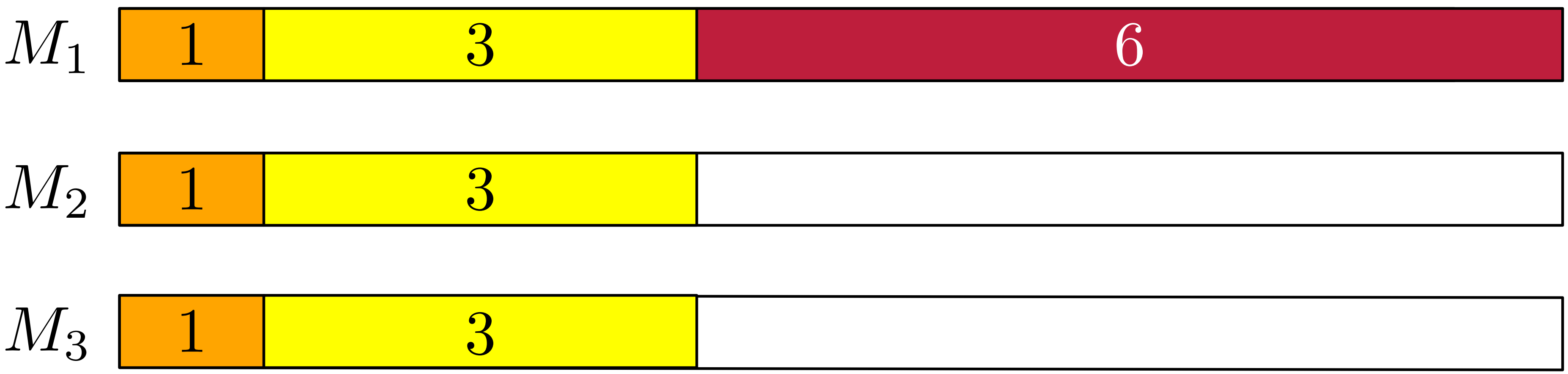

$\frac{7}{4} > \frac{5}{3}$ What item comes next?

# Can we do better?

$m = 2$: No. Why?




$m = 3$: No.




$\frac{7}{4} > \frac{5}{3}$ What item comes next?

$m > 3$: Yes, but not much, and it gets difficult.

# Randomization

Competitive ratio of $A$ against an oblivious adversary:

$$c := \sup_{\sigma} \frac{\mathbb{E}[A(\sigma)]}{OPT(\sigma)}.$$

# Randomization

Competitive ratio of $A$ against an oblivious adversary:

$$c := \sup_{\sigma} \frac{\mathbb{E}[A(\sigma)]}{OPT(\sigma)}.$$

First, a lower bound: For $m \geq 2$, we cannot be better than $4/3$.

- $m$ jobs of length $1$, possibly followed by a single $2$
- $p$ probability of makespan 1 after the $1$s
- After the $1$s: OPT $=$ 1, $\mathbb{E}[A(\sigma)] = p + (1-p) \cdot 2 = 2 - p$
- After the $2$: OPT $=$ 2, $\mathbb{E}[A(\sigma)] = 3p + 2(1-p) = 2 + p$

# Randomization

Competitive ratio of $A$ against an oblivious adversary:

$$c := \sup_{\sigma} \frac{\mathbb{E}[A(\sigma)]}{OPT(\sigma)}.$$

First, a lower bound: For $m \geq 2$, we cannot be better than $4/3$.

- $m$ jobs of length $1$, possibly followed by a single $2$
- $p$ probability of makespan 1 after the 1s
- After the 1s: OPT $= 1$, $\mathbb{E}[A(\sigma)] = p + (1-p) \cdot 2 = 2 - p$
- After the $2$: OPT $= 2$, $\mathbb{E}[A(\sigma)] = 3p + 2(1-p) = 2 + p$

$$c \geq \max\left\{2 - p, \frac{2+p}{2}\right\} \geq \frac{4}{3} \quad (p = 2/3)$$

# Bounds for Scheduling

| | deterministic | | | randomized | |
|---|---|---|---|---|---|
| $m$ | lower bound | upper bound | LS | lower bound | upper bound |
| 2 | 1.5000 | 1.5000 | 1.5000 | 1.3333 | 1.3334 |
| 3 | 1.6666 | 1.6667 | 1.6667 | 1.4210 | 1.5567 |
| 4 | 1.7310 | 1.7333 | 1.7500 | 1.4628 | 1.6589 |
| 5 | 1.7462 | 1.7708 | 1.8000 | 1.4873 | 1.7338 |
| 6 | 1.7730 | 1.8000 | 1.8333 | 1.5035 | 1.7829 |
| 7 | 1.7910 | 1.8229 | 1.8571 | 1.5149 | 1.8169 |
| $\infty$ | 1.8520 | 1.9230 | 2.0000 | 1.5819 | – |