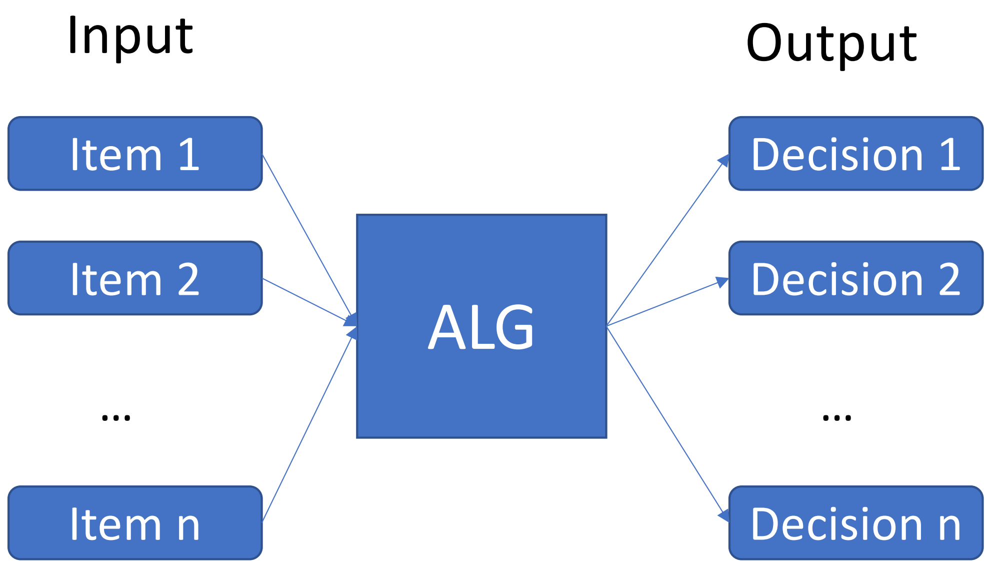
COMP 6651

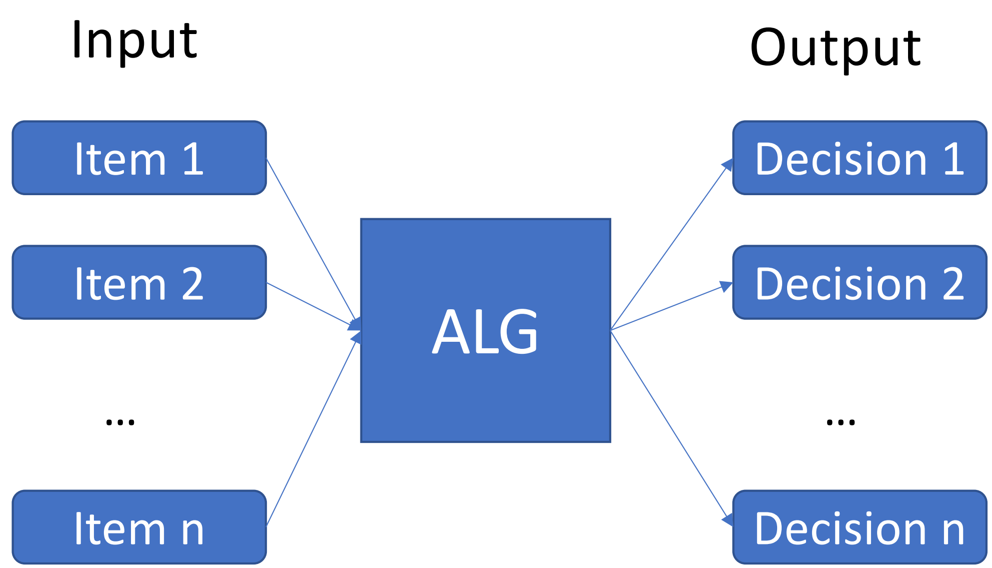
Lecture on Online Algorithms and Competitive Analysis, Part 2

Denis Pankratov

Offline



Online



Online algorithm formally

Online algo: produces decisions based on past, but not future, inputs

$$d_k = d_k(i_1, i_2, \dots, i_k)$$

Offline optimum: produces decisions based on entire input

$$\widehat{d_k} = \widehat{d_k}(i_1, i_2, \dots, i_n)$$

Competitive ratio for minimization problem, informally

Accumulated cost of our online algorithm

Competitive ratio

$$\rho = \max_{\{n, i_1, i_2, \ldots, i_n\}} \frac{ALG(i_1, i_2, \ldots, i_n)}{OPT(i_1, i_2, \ldots, i_n)}$$

"How close can we get to hindsight?"

Accumulated cost of an optimal offline algorithm

Last Time

Ski Rental

- Deterministic Algorithm: Break Even, 2-competitive
- Lower Bound Argument: Adversarial Argument
- Randomized Algorithm

Line Search

- Deterministic Algorithm: Doubling Strategy, 9-competitive
- Lower Bound Argument: Bounding Cones

Paging

Another example: Paging

Cache of size k

Page requests come online

Cache Hit: requested page is already in the cache

Cache Miss: requested page is not in the cache

If cache is full and cache miss occurs, need

An eviction policy: which page(s) to remove from the cache to bring in the newly requested page in.

Paging

Pages are represented as numbers from universe $[n]$

Cache contents:

Example:

Request sequence:

5	3	4	3	1	5	3	4

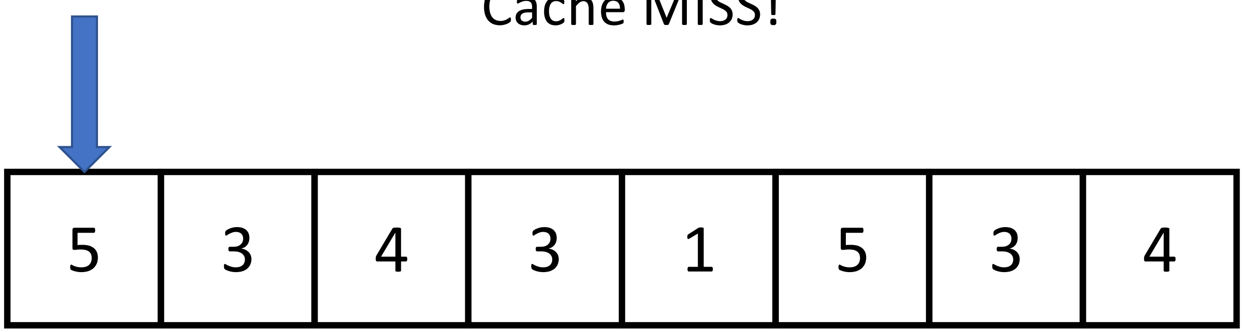
Paging Example

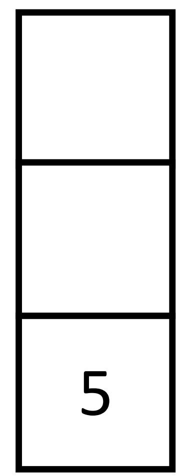
Cache size $k = 3$, universe size = $n = 5$

Time: 1

Cache contents:

Cache MISS!





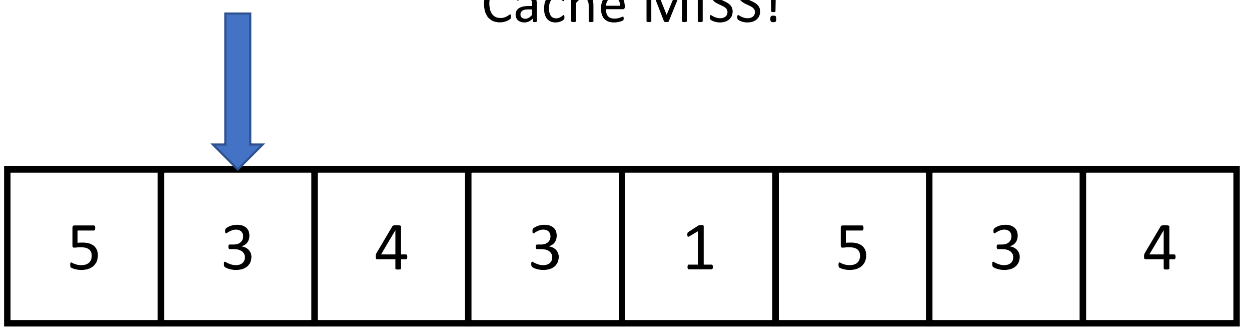
Paging Example

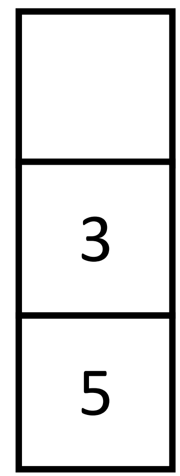
Cache size $k = 3$, universe size = $n = 5$

Time: 2

Cache contents:

Cache MISS!





Paging Example

Cache size $k = 3$, universe size = $n = 5$

Time: 3

Cache contents:

Cache MISS!

5	3	4	3	1	5	3	4

4
3
5

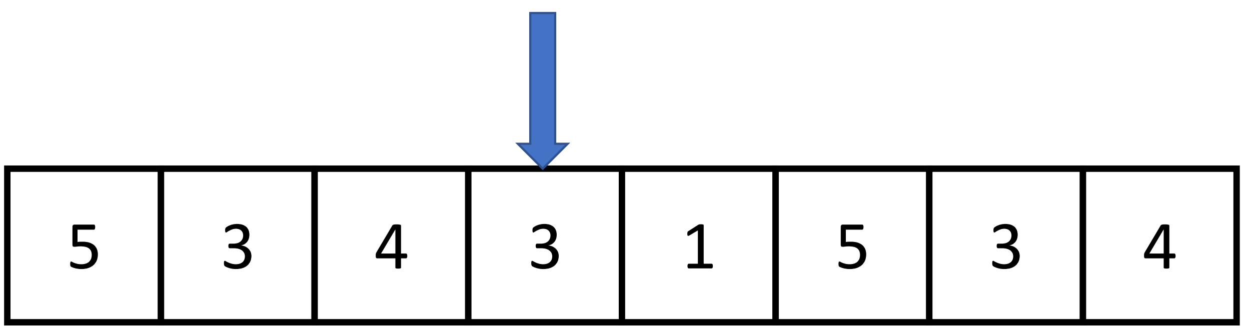
Paging Example

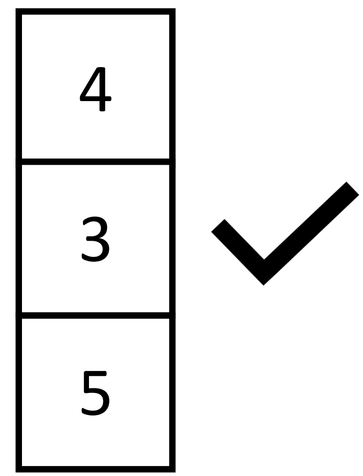
Cache size $k = 3$, universe size = $n = 5$

Time: 3

Cache contents:

Cache HIT!





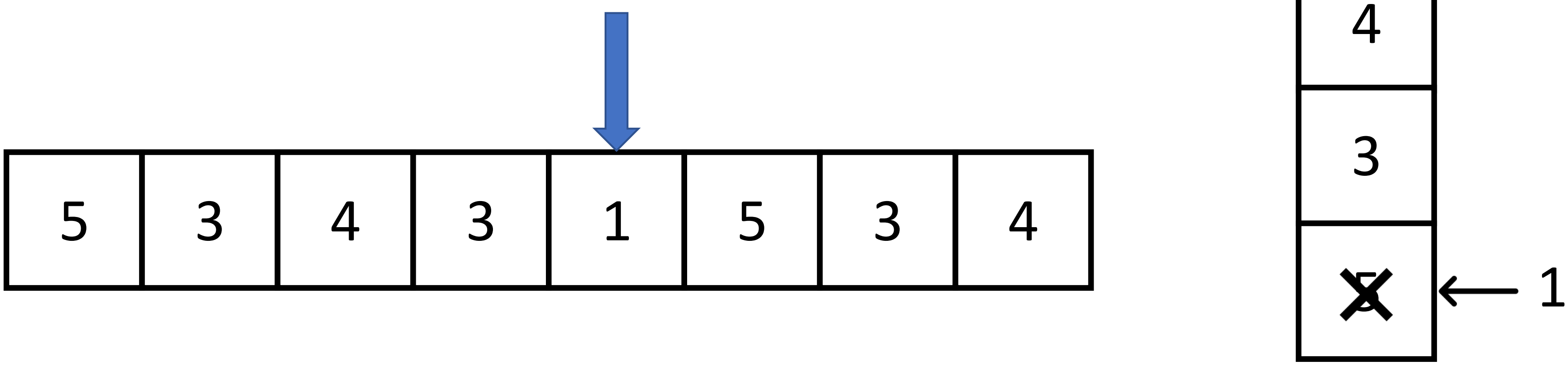
Paging Example

Cache size $k = 3$, universe size = $n = 5$

Time: 3

Cache contents:

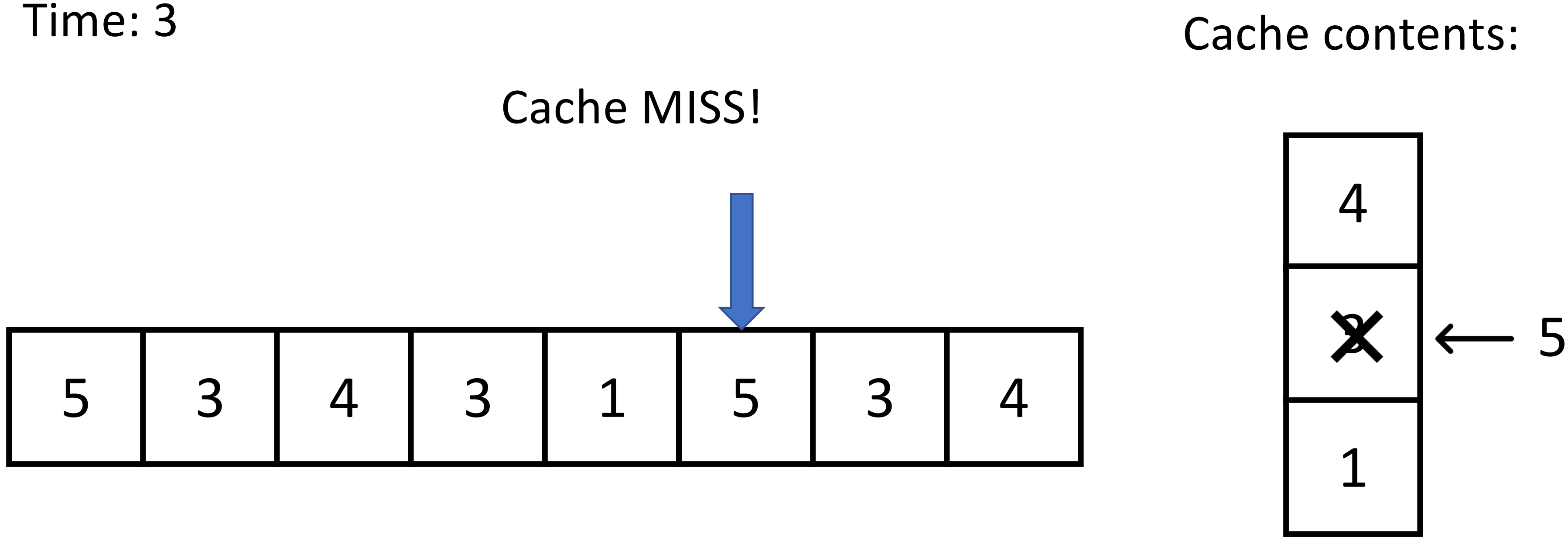
Cache MISS!



Paging Example

Cache size $k = 3$, universe size = $n = 5$

Time: 3



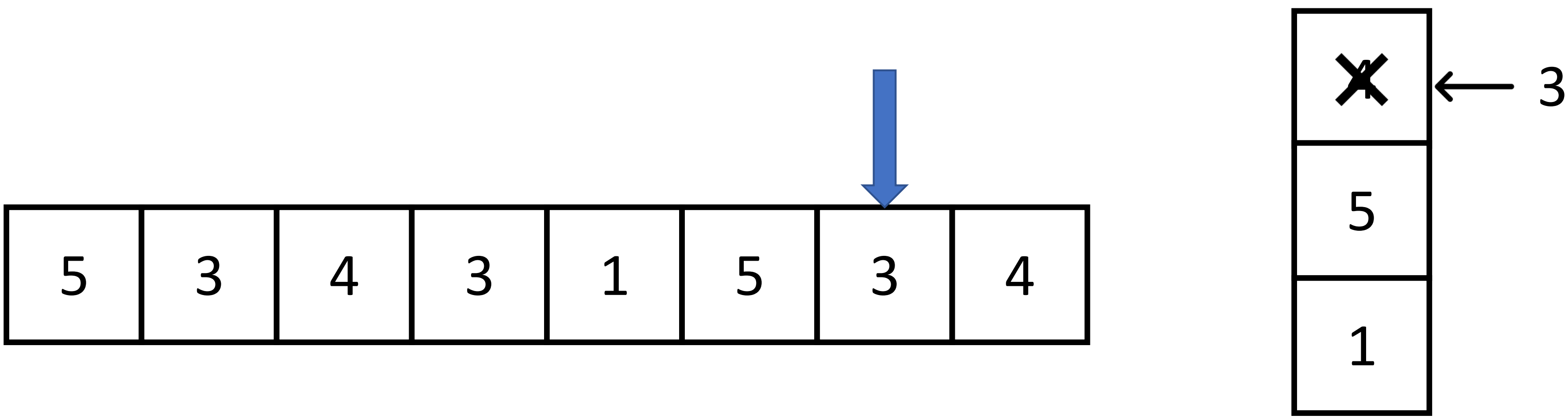
Paging Example

Cache size $k = 3$, universe size = $n = 5$

Time: 3

Cache contents:

Cache MISS!



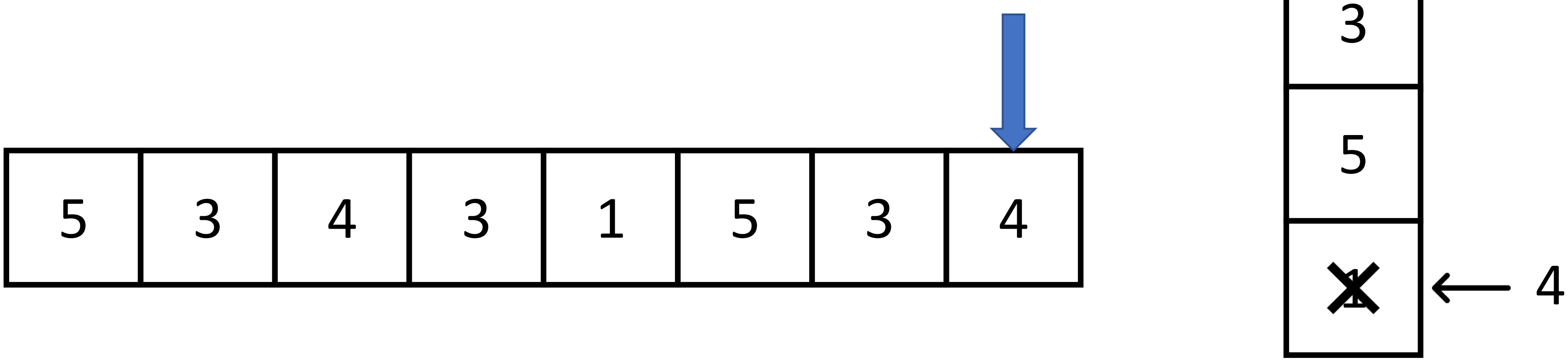
Paging Example

Cache size $k = 3$, universe size = $n = 5$

Time: 3

Cache contents:

Cache MISS!



Paging

Goal: minimize the number of cache misses

Consider the following policies:

FIFO (First In First Out): evict the page that was inserted the earliest

LRU (Least Recently Used): evict the page that was accessed least recently

Flush When Full: evict all pages when cache is full and cache miss occurs

FIFO analysis, positive result

Denote input sequence by X

Subdivide the sequence of pages into blocks

$$B_1, B_2, B_3, \ldots$$

Block B_1 : maximal prefix of the input sequence that contains exactly k distinct pages

Example: $k = 3$

$$\underbrace{\boxed{5}\,\boxed{3}\,\boxed{4}\,\boxed{3}}_{B_1}\,\boxed{1}\,\boxed{5}\,\boxed{3}\,\boxed{4}$$

FIFO analysis, positive result

Input sequence X subdivided into b blocks

$$B_1, B_2, B_3, \dots, B_b$$

B_1: maximal prefix of X with exactly k distinct pages

B_2: maximal prefix of $X - B_1$ with exactly k distinct pages

B_3: maximal prefix of $X - B_1 - B_2$ with exactly k distinct pages

…

FIFO analysis, positive result

Input sequence X subdivided into b blocks

$$B_1, B_2, B_3, \ldots, B_b$$

FIFO incurs at most k cache misses per block (why?)

Therefore, overall number of cache misses of FIFO is at most

$$bk$$

FIFO analysis, positive result

Input sequence X subdivided into b blocks

$$B_1, B_2, B_3, \dots, B_b$$

The entire block B_i and the first page of B_{i+1} have $k + 1$ distinct pages

Therefore, any eviction policy with cache size k incurs at least one cache miss processing each B_i and the first page of B_{i+1} (pigeonhole principle)

Therefore, <u>OPT</u> incurs at least $b - 1$ cache misses

FIFO analysis, positive result

Competitive ratio = FIFO / OPT = $\frac{kb}{b-1} = k + \frac{k}{b-1} \to k$ as $b \to \infty$

FIFO analysis, negative result

Is our analysis of FIFO tight?

We will show more generally that no deterministic policy can achieve competitive ratio better than k

Fix deterministic policy ALG. Take universe size $n = k + 1$

Adversary: initially, request k distinct pages.

each time after that, request a page not in the cache of ALG (such a page exists, because universe size is $k + 1$ and cache size is k)

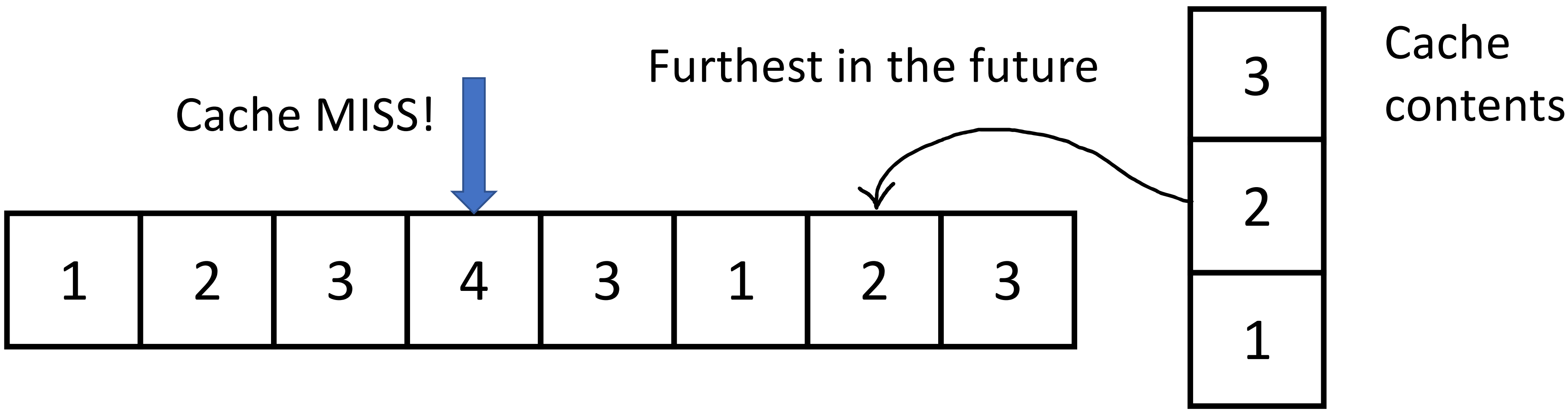
Thus, adversary can always make ALG have at least $|X|$ cache misses

FIFO analysis, negative result

Need: for $n = k + 1$ and any X, OPT $\leq \frac{|X|}{k} + k - 1$ cache misses

First k distinct pages incur k cache misses

After that, when a new page arrives OPT evicts a page from the cache that will be accessed furthest in the future (OPT knows the entire X)



FIFO analysis, negative result

Furthest in the future:

Evicted element has to occur at least $k-1$ page requests later (why?)

Thus, OPT can guarantee at most one page miss per k requests

Overall, we have

$$\text{OPT} \leq \underbrace{k}_{\text{Initial } k \text{ pages}} + \underbrace{\frac{|X|-k}{k}}_{\text{Remaining } |X|-k \text{ pages incur one miss per } k \text{ pages}} = \frac{|X|}{k} + k - 1$$

FIFO analysis, negative result

Putting it together:

Competitive ratio (ALG) = $\frac{ALG}{OPT} = \frac{|X|}{\frac{|X|}{k}+k-1} = \frac{k}{1+\frac{k-1}{|X|}} \rightarrow k$ as $|X| \rightarrow \infty$

LRU vs. FIFO vs. Flush When Full

The same analysis applies to LRU

$$\text{Competitive ratio (FIFO)} = \text{competitive ratio (LRU)} \approx k$$

What about Flush When Full?

Paging: comments

The closer competitive ratio is to 1, the better (closer to OPT)

LRU and FIFO have competitive ratio $\approx k$ = cache size

Therefore, increasing cache size leads to a worse algorithm! (in theory)

Counterintuitive! Does not match practice

Moreover, competitive ratio does not distinguish between LRU vs. FIFO

However, FIFO is terrible in practice, and LRU is excellent!

These shortcomings indicate that competitive ratio does not always match practice

Competitive ratio

To address these criticisms, we could consider other input/algorithmic models

Model “realistic inputs” (parameterized inputs, stochastic inputs, restricted inputs) and allow more power to algorithms (look ahead, advice, etc.)

E.g.: realistic inputs for Paging exhibit locality of reference.

Makespan

Makespan problem setting

You control several *identical* machines

Jobs arrive one by one

You are only concerned with a *processing* time of each job

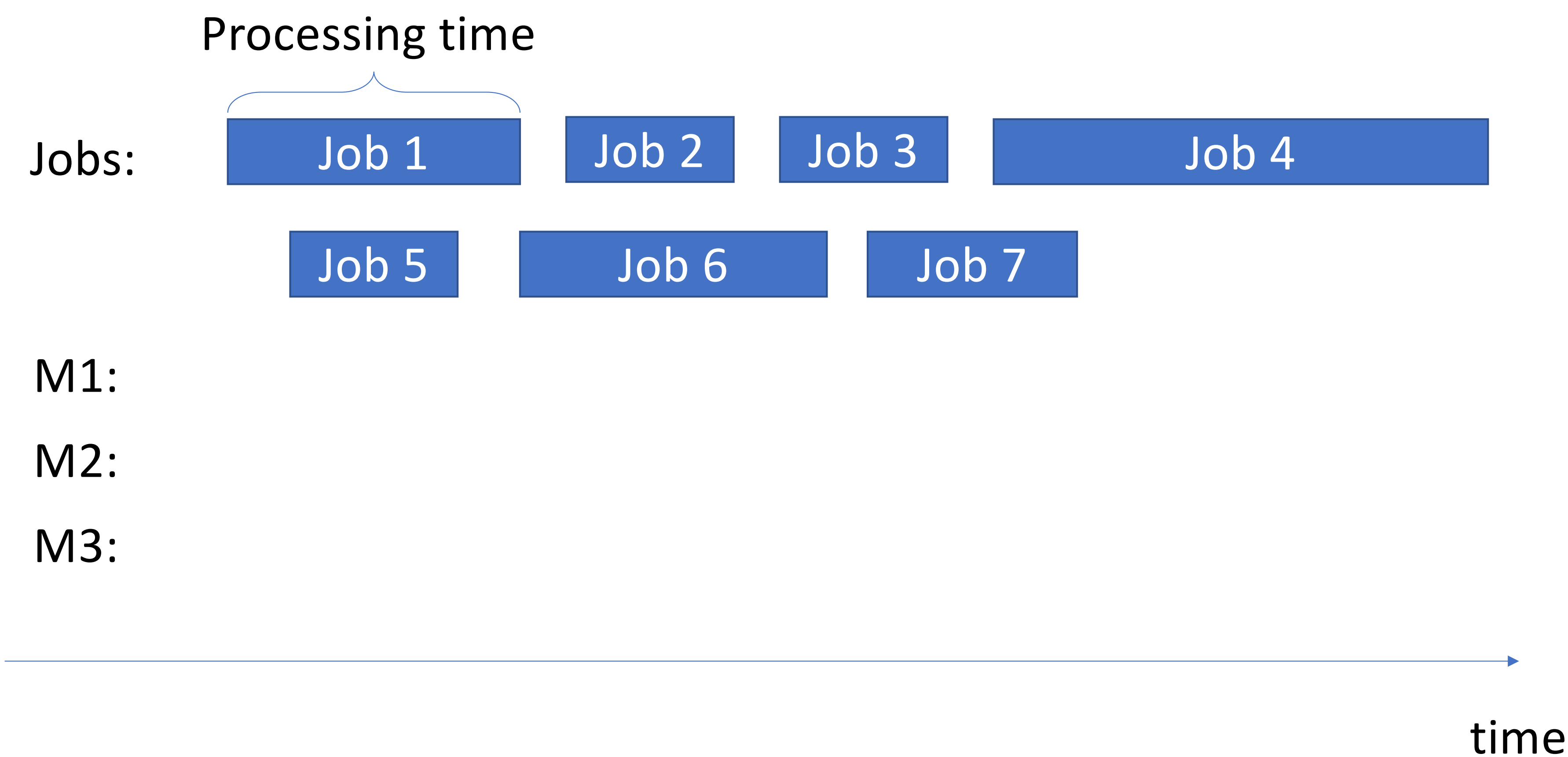
Schedule each job to be executed on one of the machines

Jobs are scheduled back to back on each machine

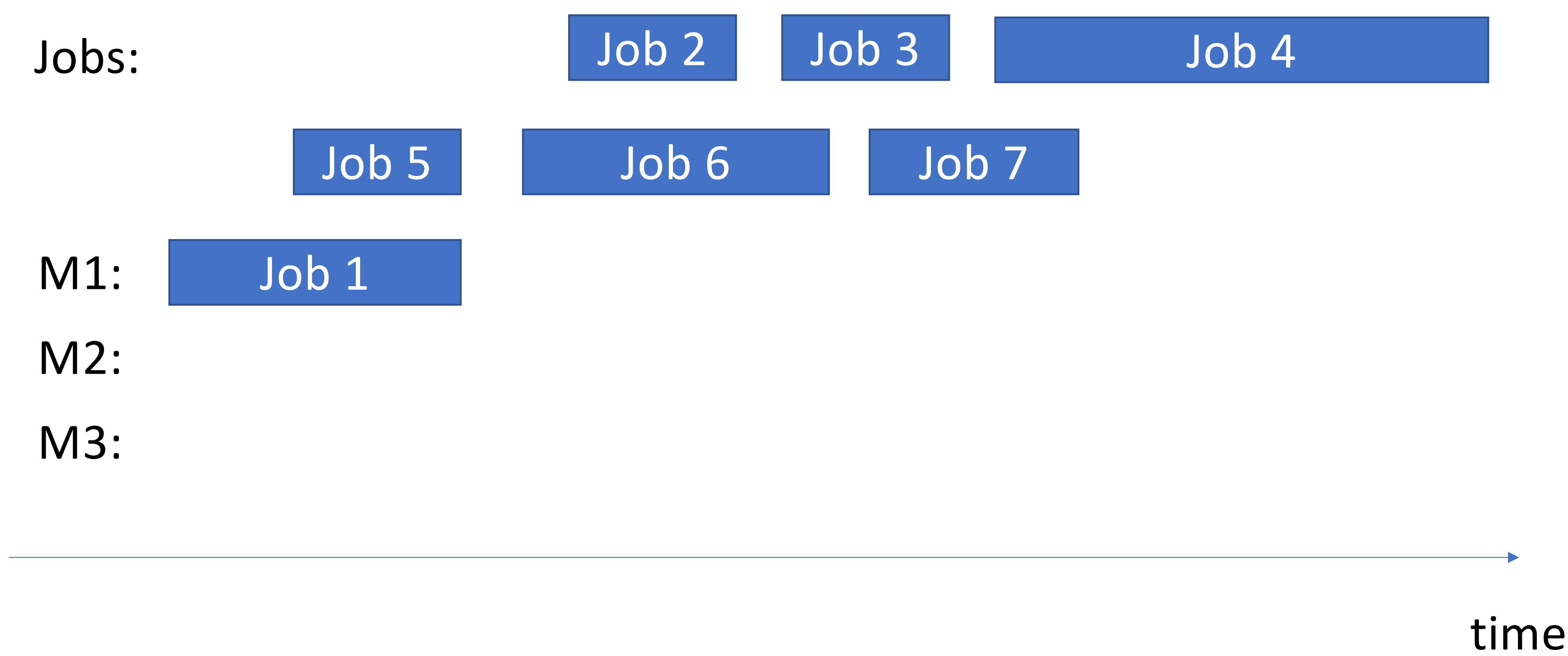
Goal: minimize longest *completion time* of a machine

Real-life examples: scheduling a project consisting of subtasks with multiple workers; running a cloud computing service on several servers; running a supercomputing server; etc.

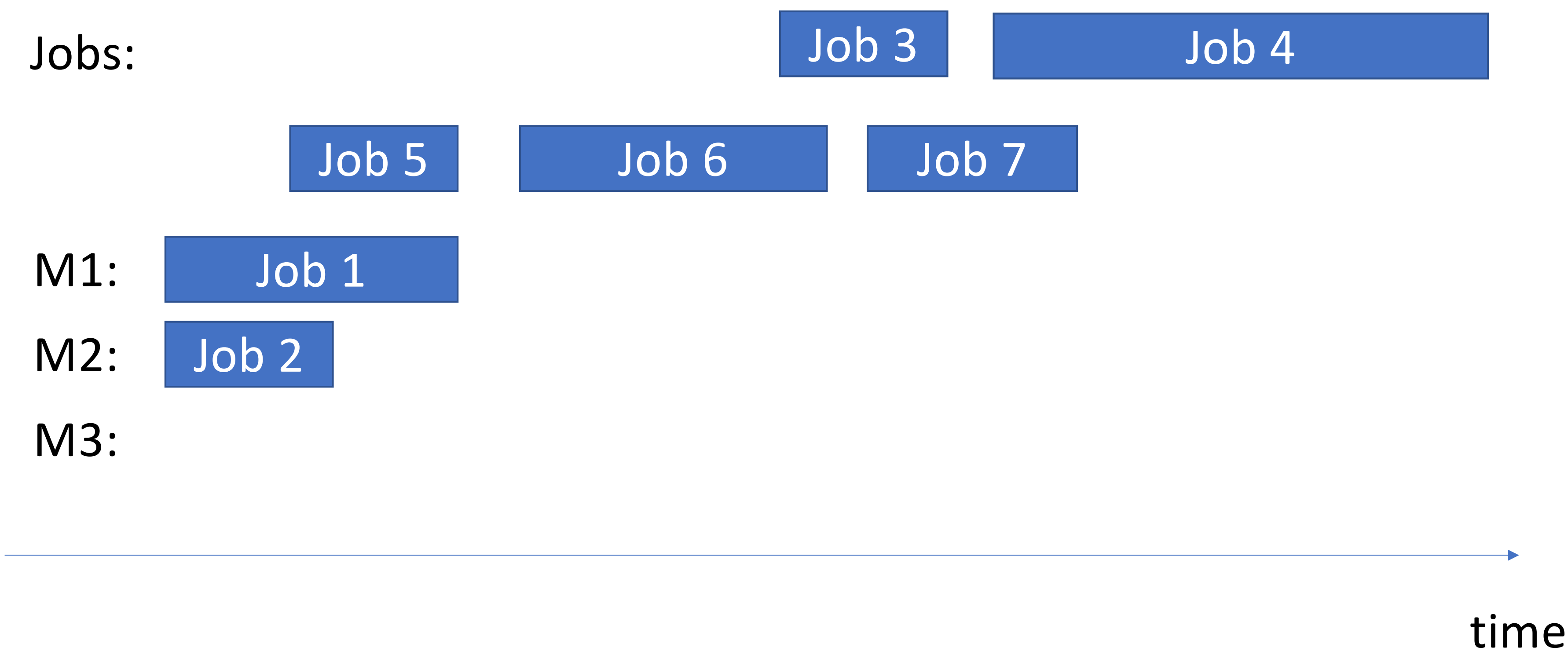
Makespan problem example:



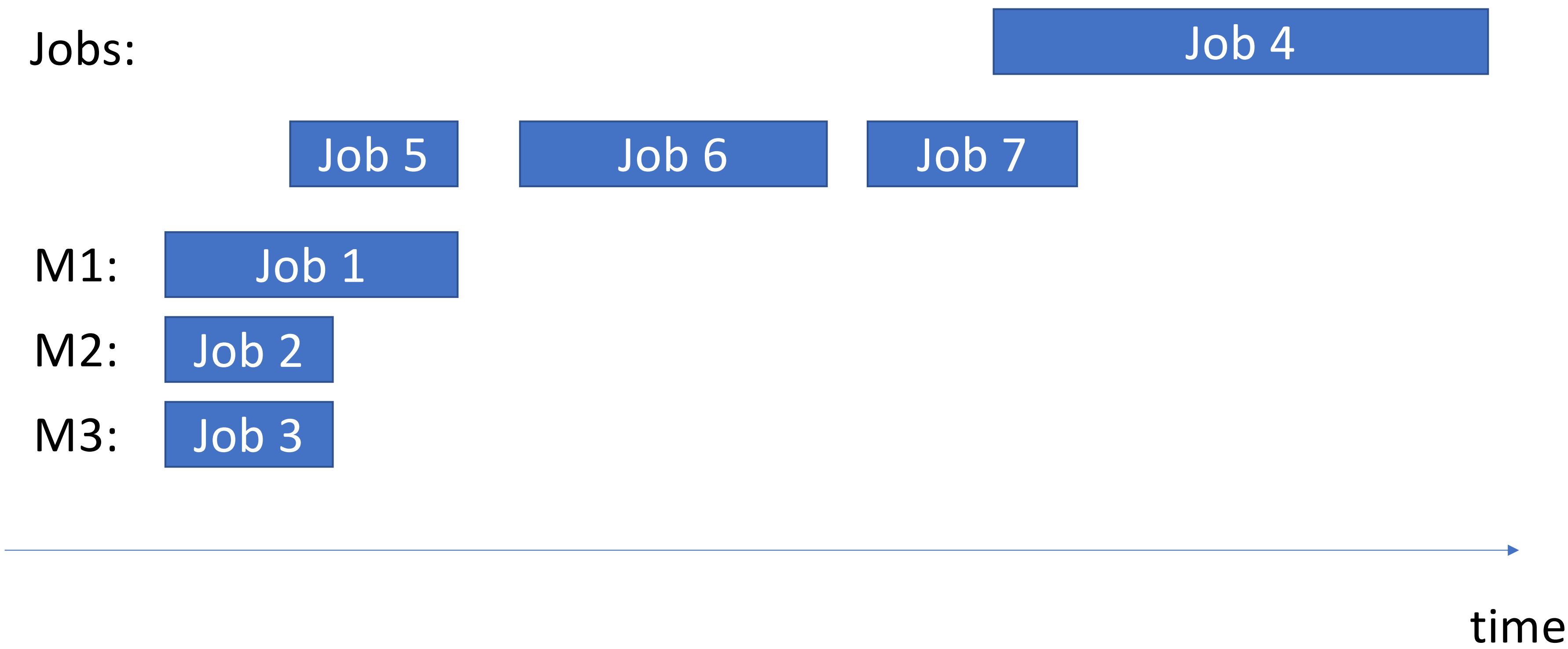
Makespan problem example:



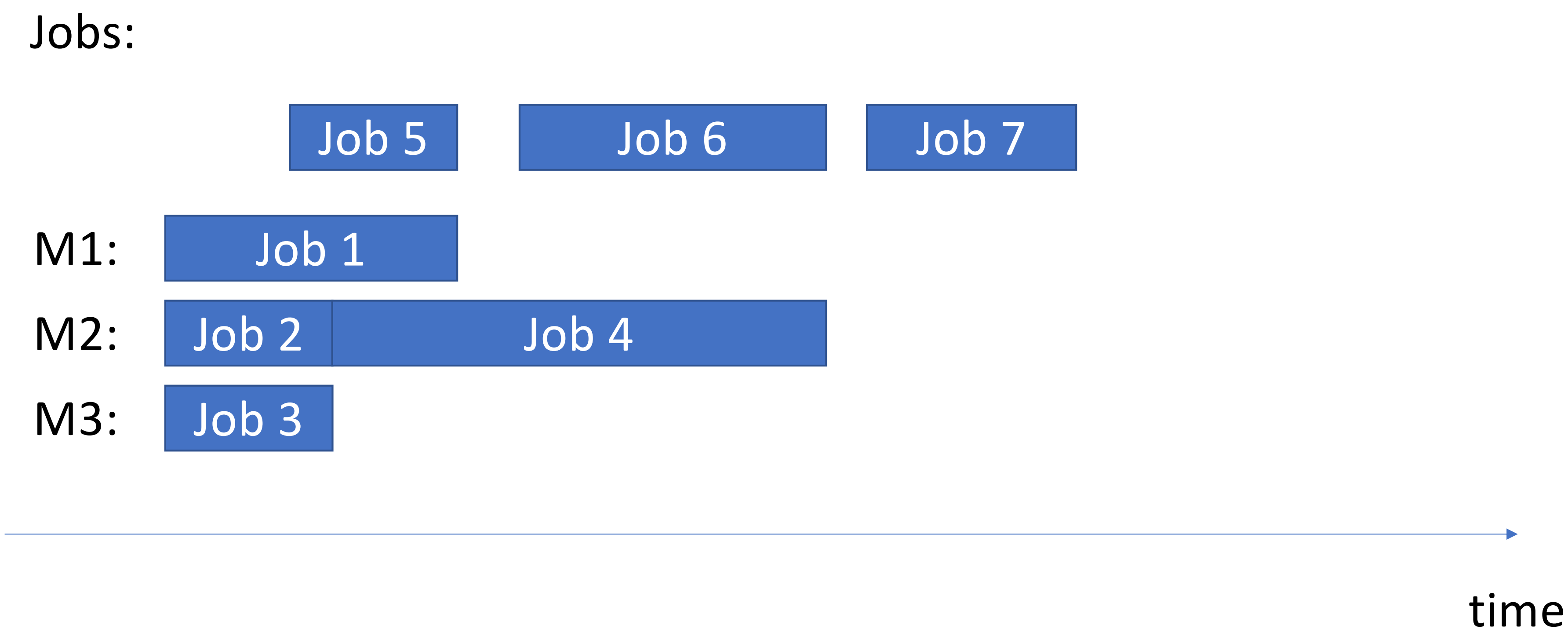
Makespan problem example:



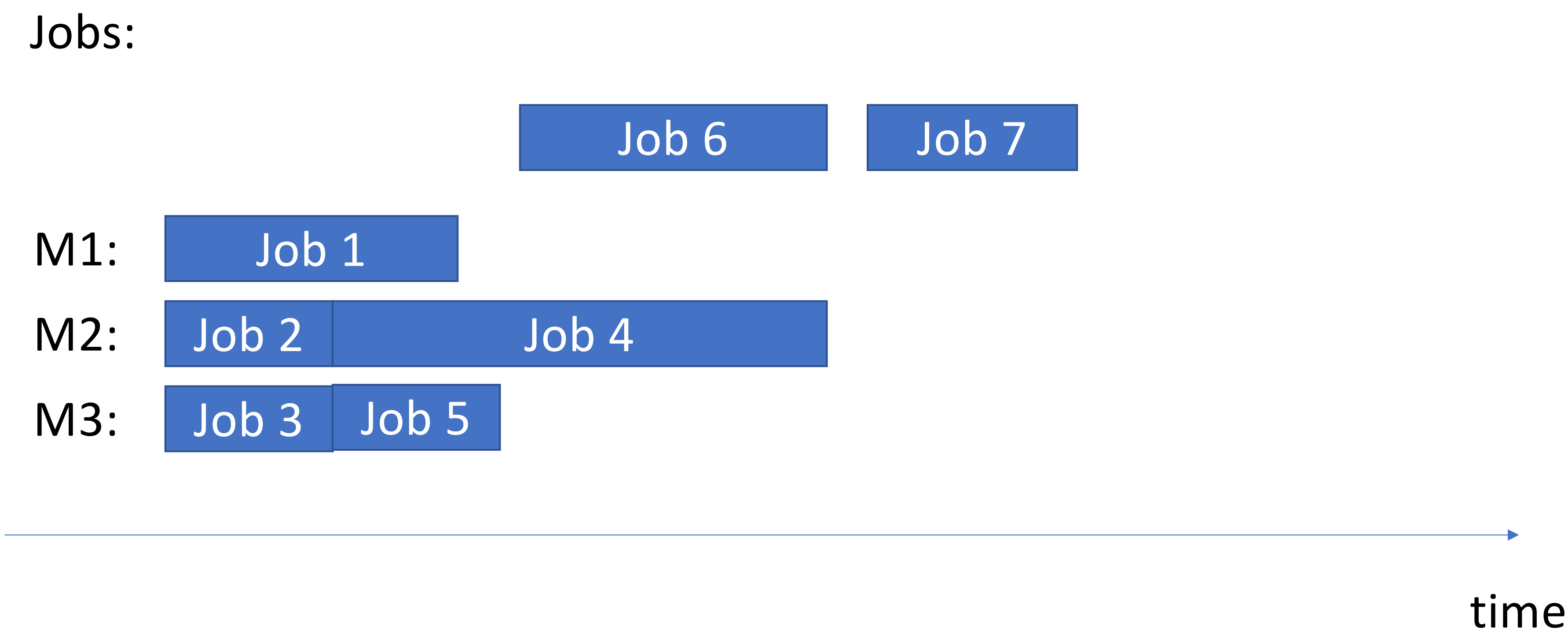
Makespan problem example:



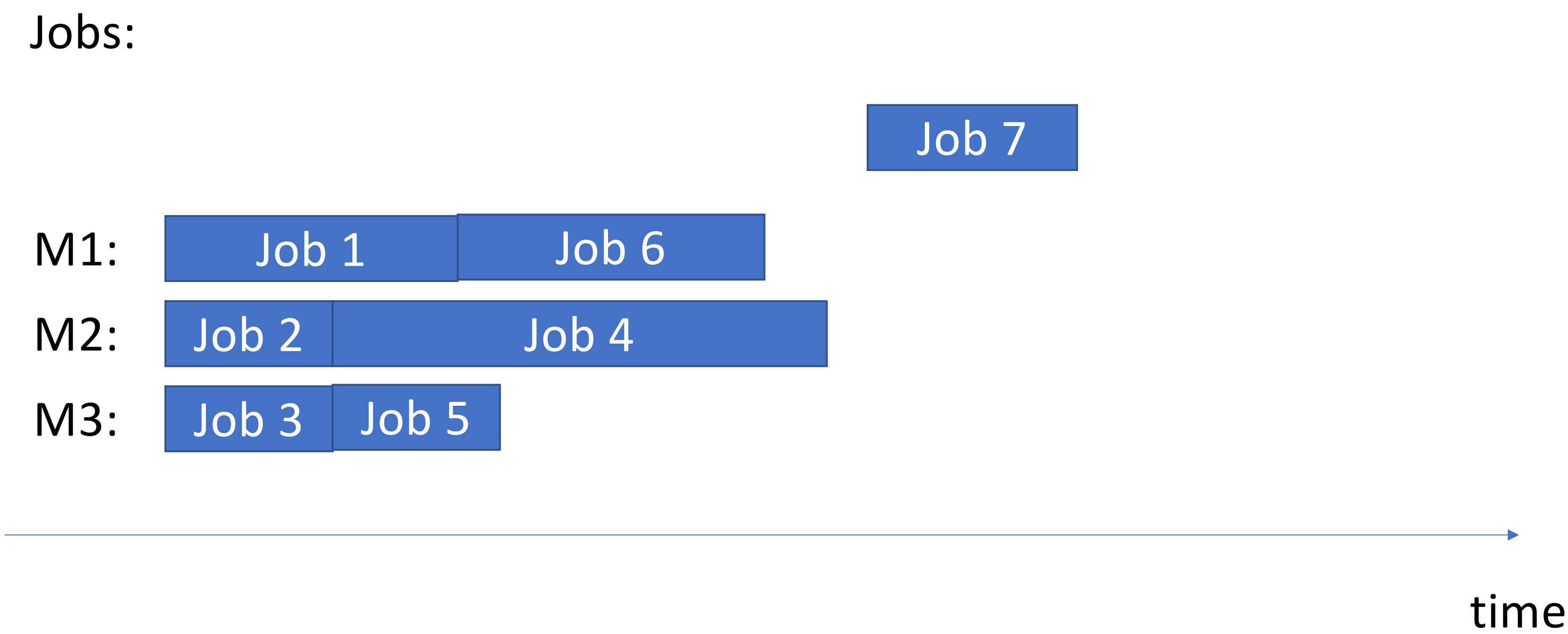
Makespan problem example:



Makespan problem example:

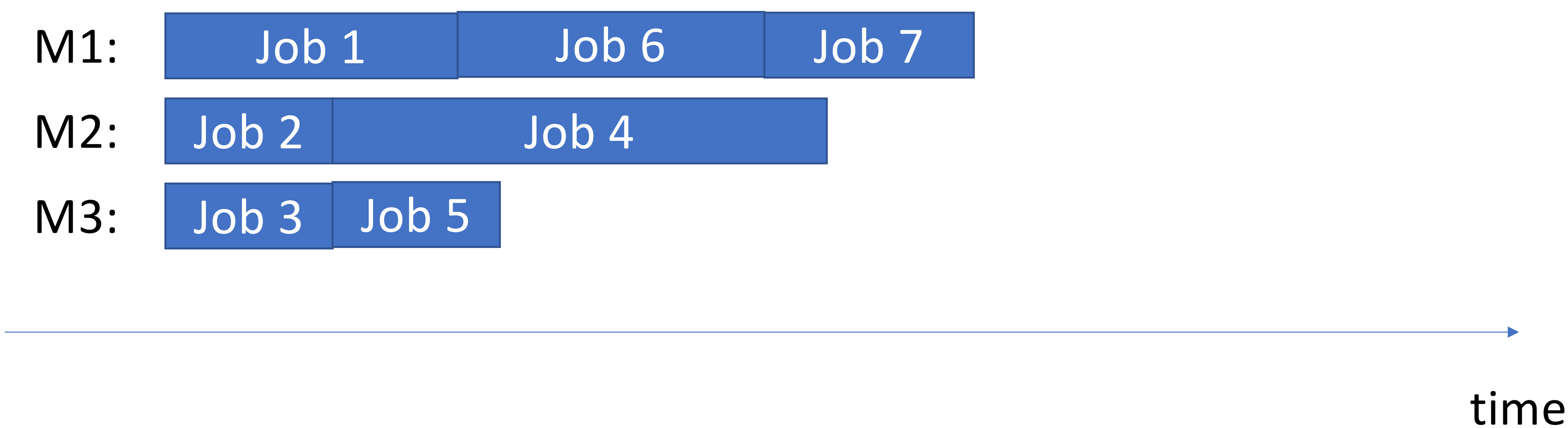


Makespan problem example:

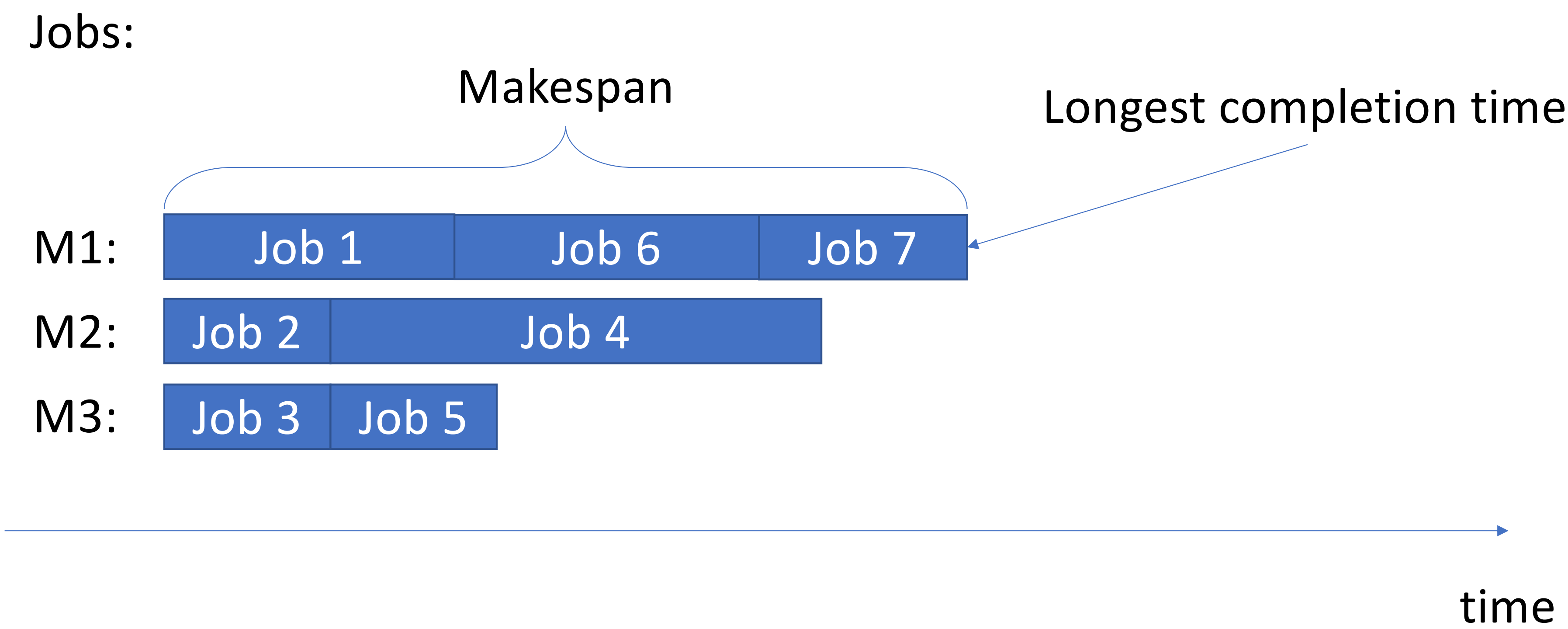


Makespan problem example:

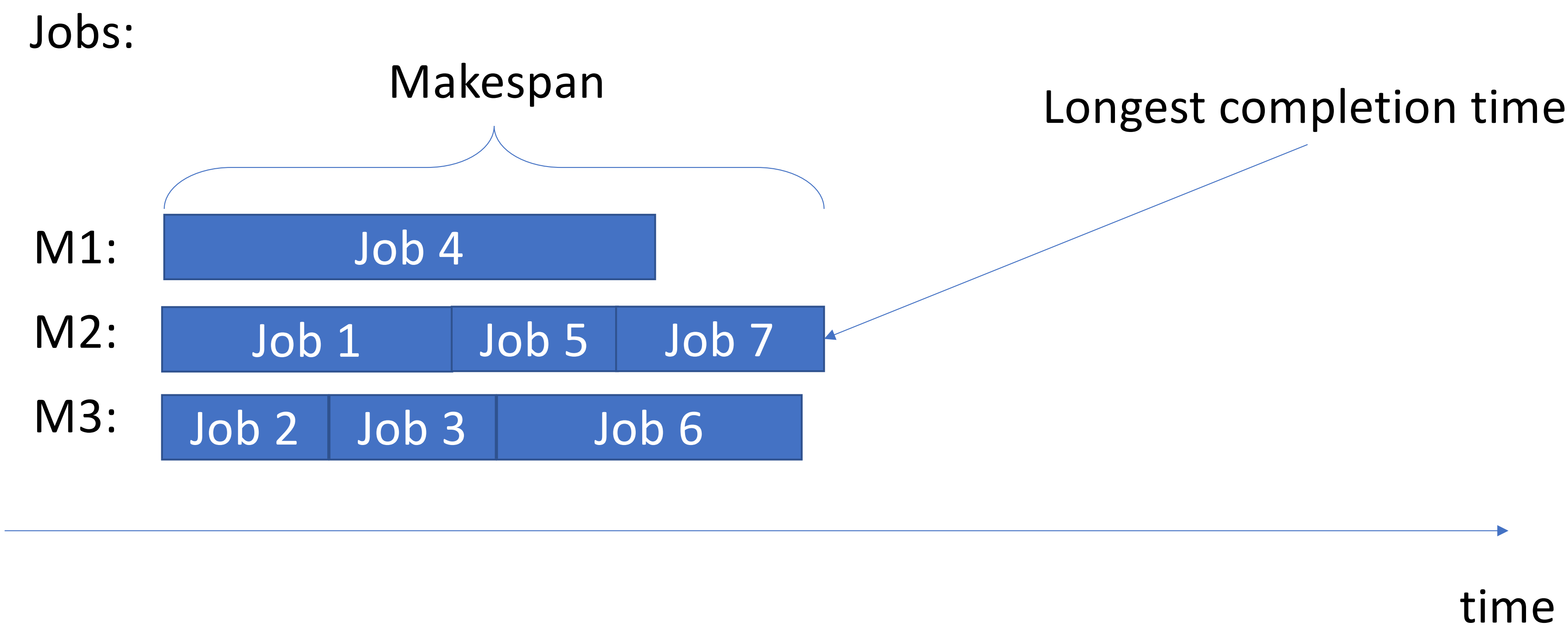
Jobs:



Makespan problem example: **sample schedule**



Makespan problem example: **better schedule**



Makespan formally:

Input: $(p_1, \dots, p_n)$ where p_j is the *processing time* of job j

m – number of *identical machines*

Output: $\sigma : \{1, \dots, n\} \to \{1, \dots, m\}$ – the *schedule*, where

$\sigma(j) = i$ means that job j has been assigned to machine i

Objective: to find σ to *minimize* $\max_i \sum_{j:\sigma(j)=i} p_j$

Simplest online algorithm: greedy

Always schedule the newly arriving job on the ***least loaded machine***

Simplest online algorithm: greedy

Algorithm 1 THE ONLINE GREEDY MAKESPAN

procedure GREEDY MAKESPAN
  initialize $s(i) \leftarrow 0$ for $1 \leq i \leq m$
  $j \leftarrow 1$
  while $j \leq n$ **do**
    $i' \leftarrow \arg\min_i s(i)$
    $\sigma(j) \leftarrow i'$
    $s(i') \leftarrow s(i') + p_j$
    $j \leftarrow j + 1$
  return σ

The current load on machine i

Least loaded machine, can break ties arbitrarily

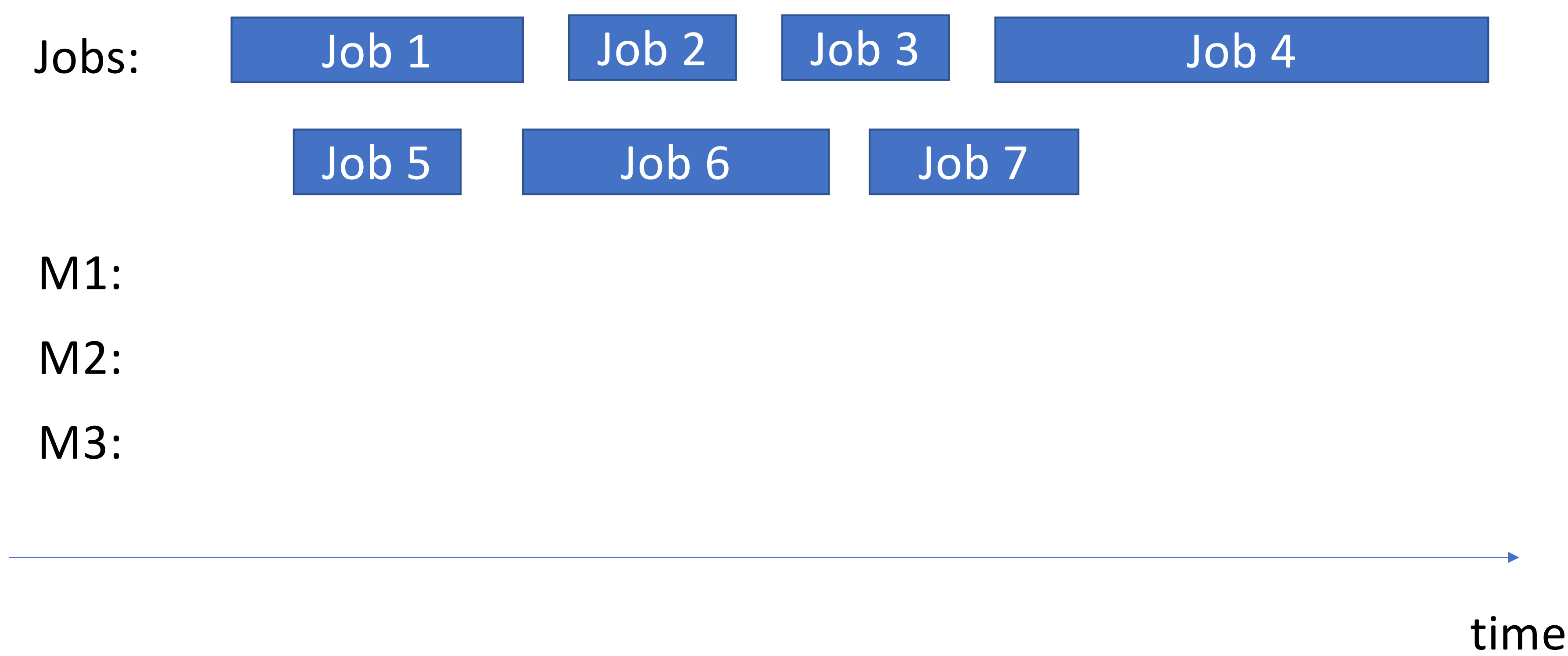
Greedy algorithms

“Greedy algorithm” is a more general term, which means

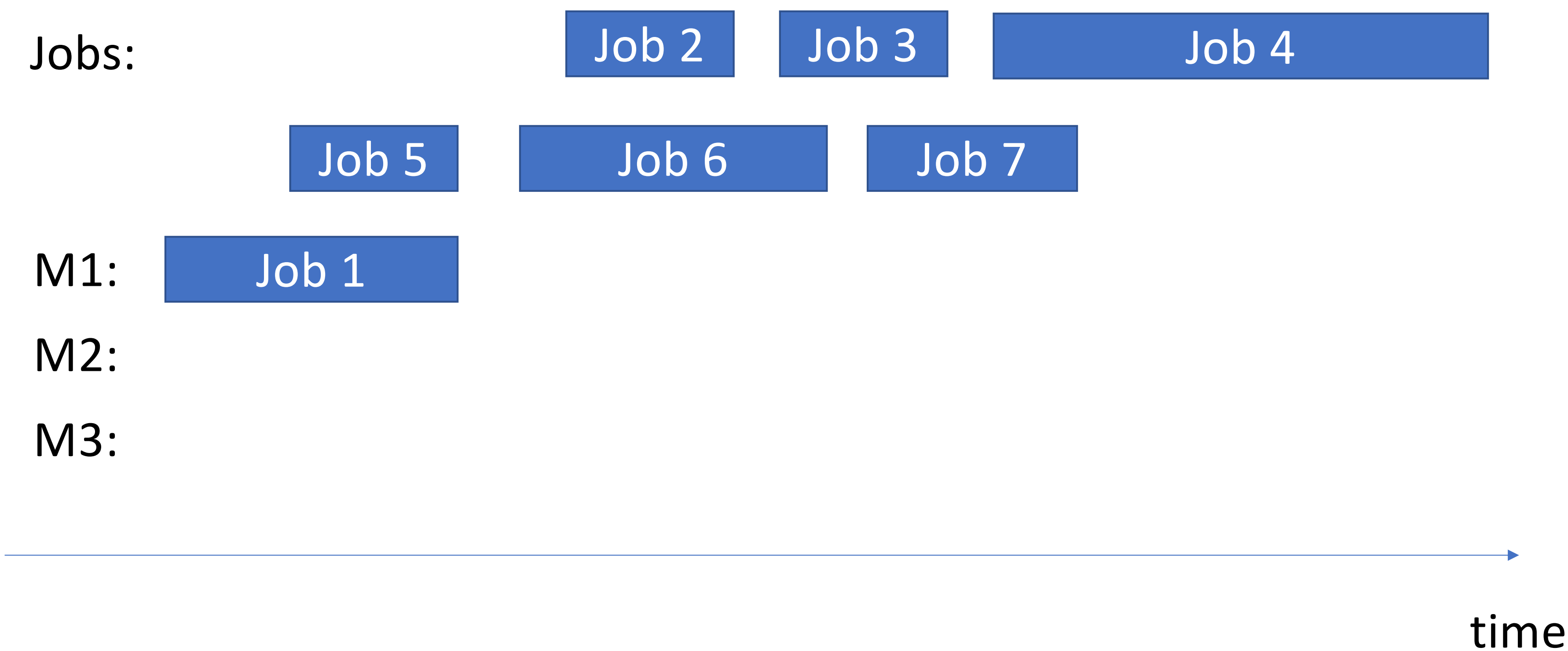
The algorithm tries to optimize the objective function in each step, i.e., it follows the motto “live for today, as if there is no tomorrow.” (YOLO!)

Greedy algorithm treats the latest input item as if it were the last. What is the best move assuming that there are no more input items? This happens even if the algorithm knows that there are more input items to come.

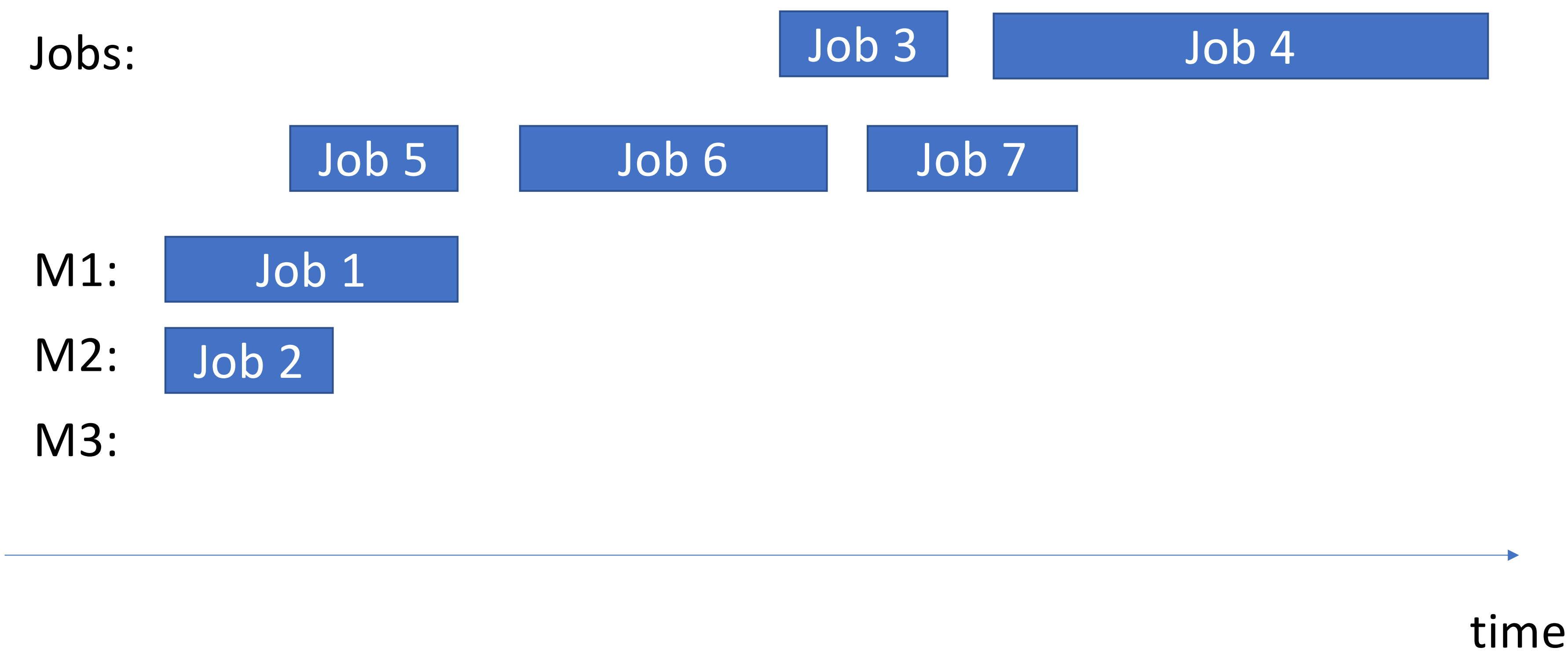
Greedy makespan algorithm example:



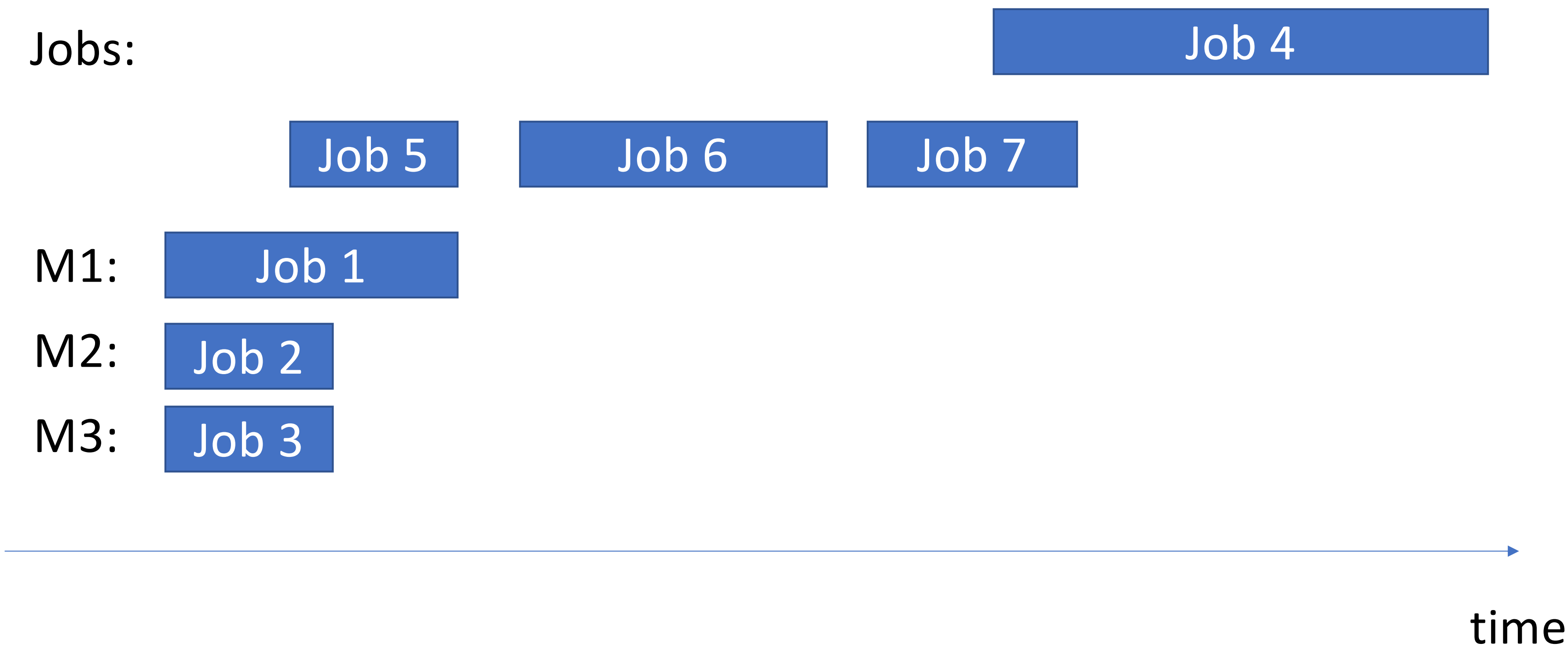
Greedy makespan algorithm example:



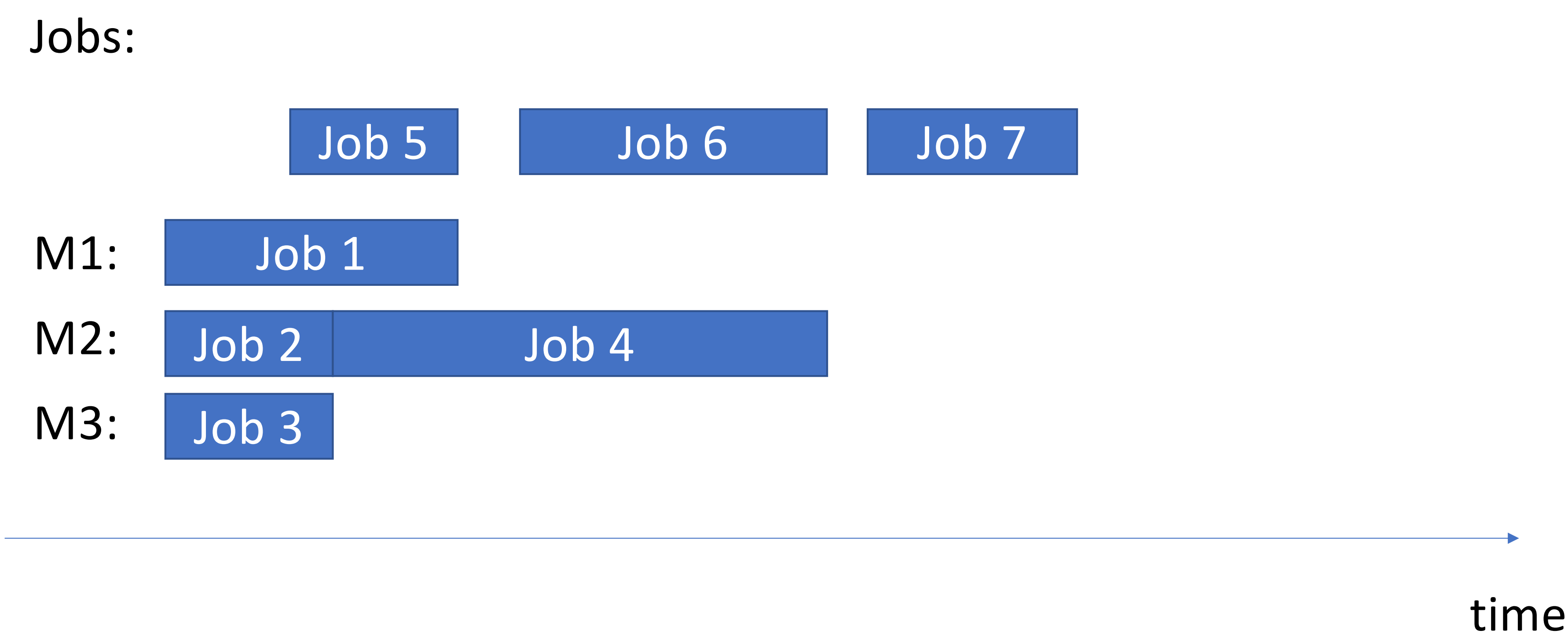
Greedy makespan algorithm example:



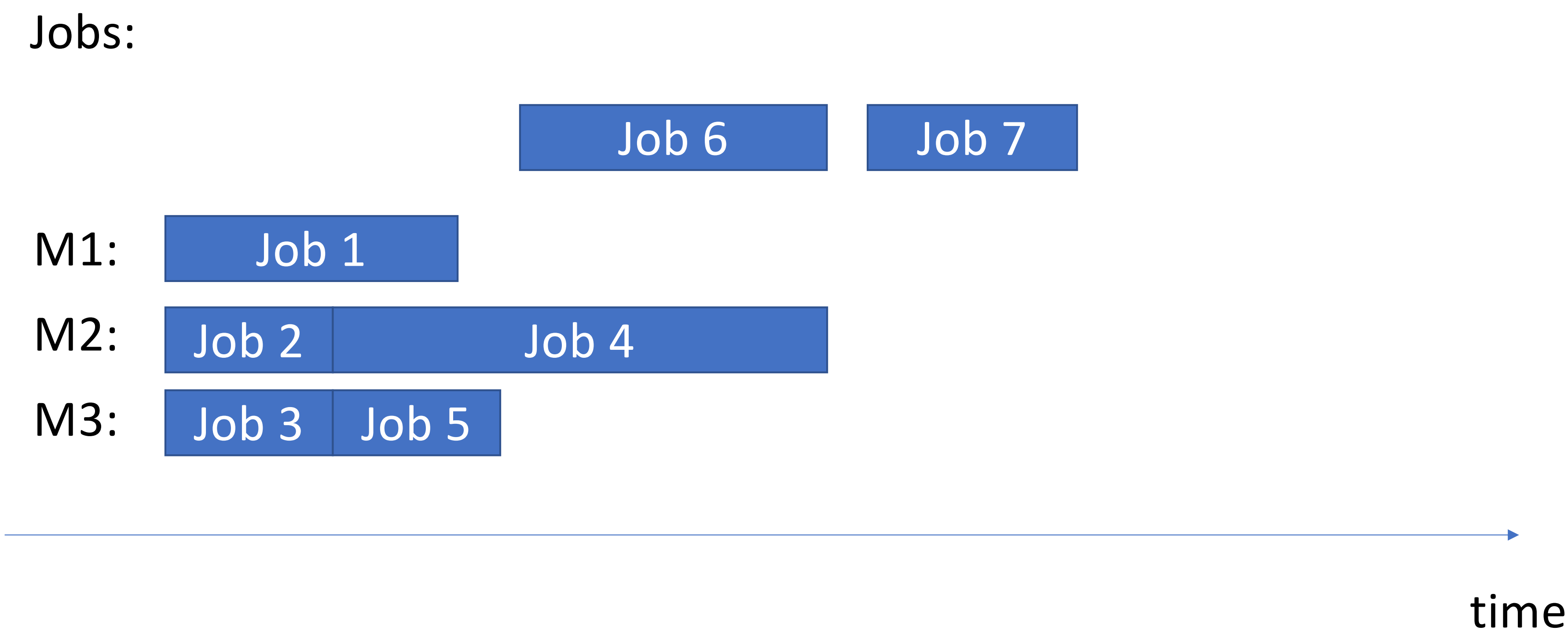
Greedy makespan algorithm example:



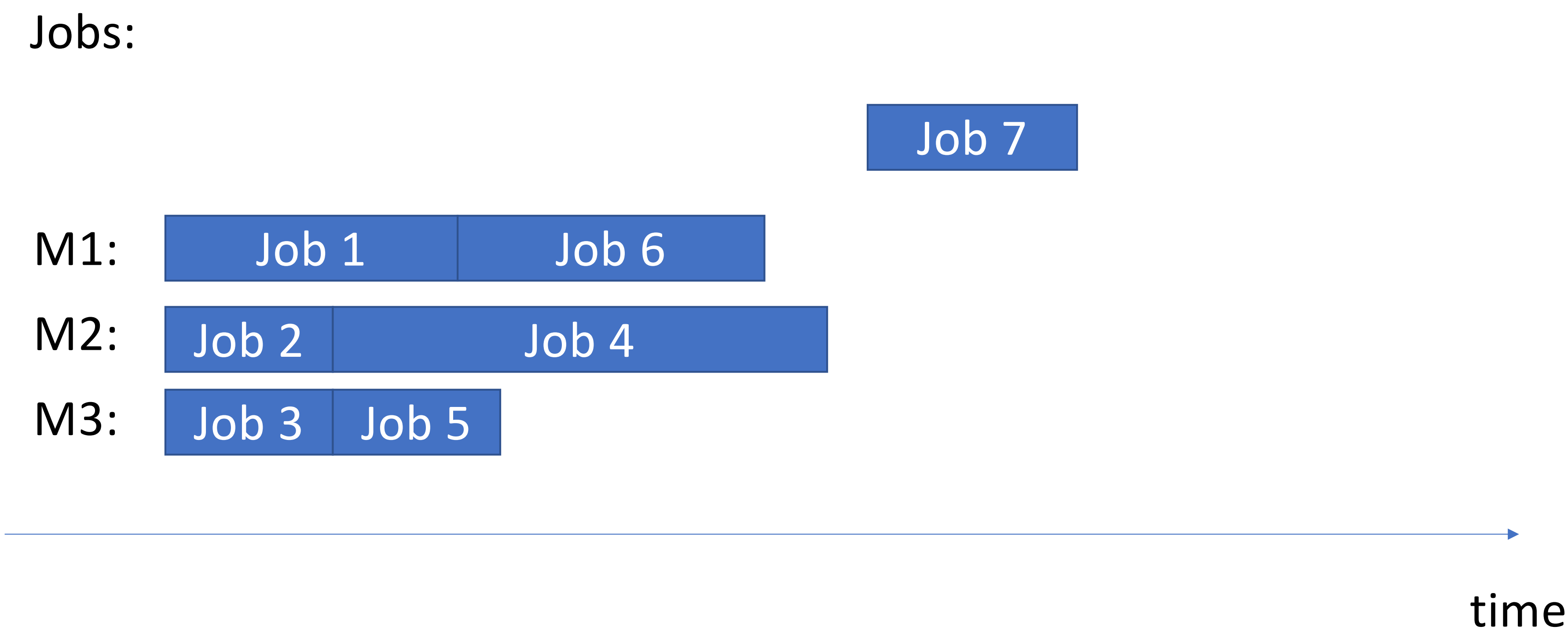
Greedy makespan algorithm example:



Greedy makespan algorithm example:

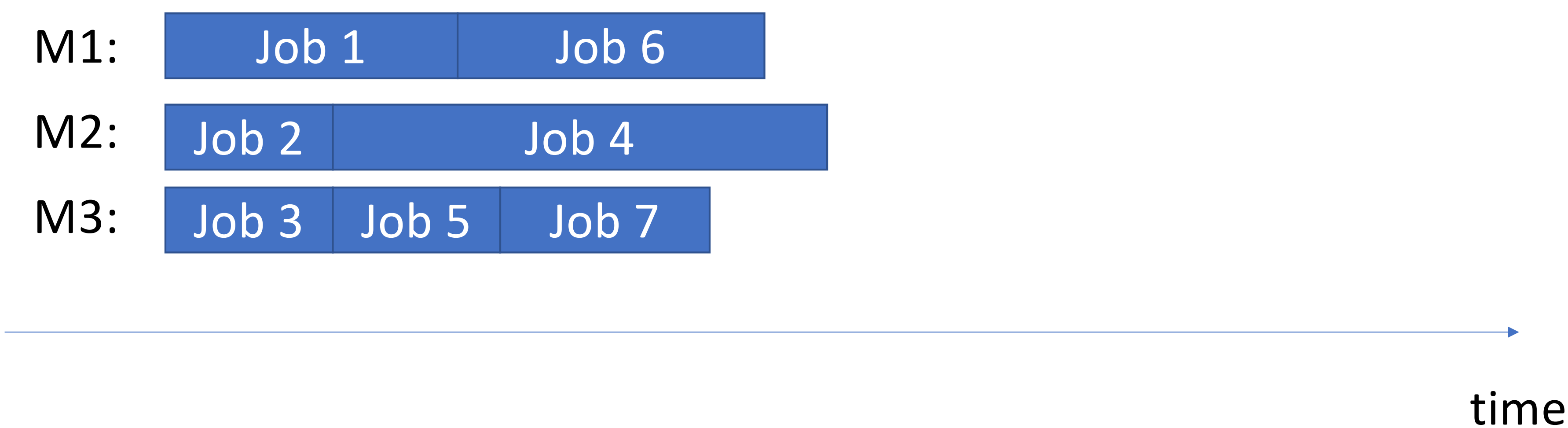


Greedy makespan algorithm example:

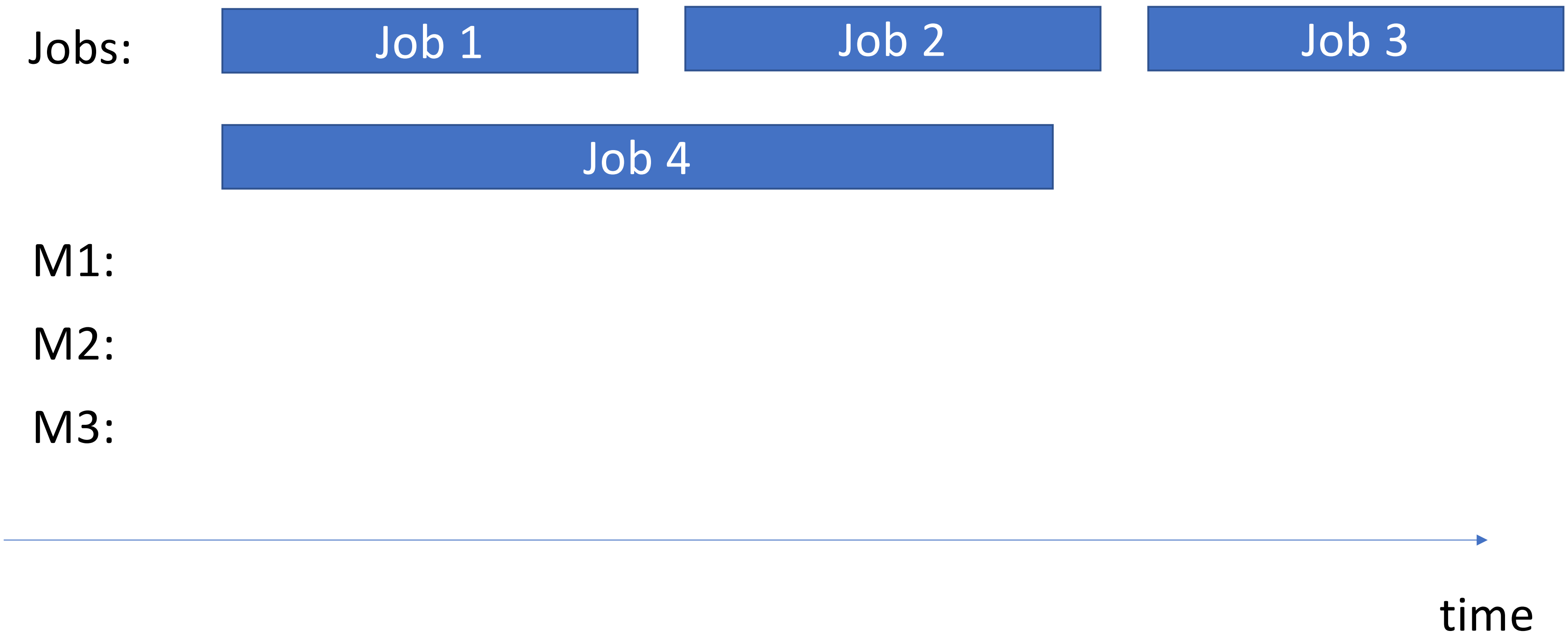


Greedy makespan algorithm example:

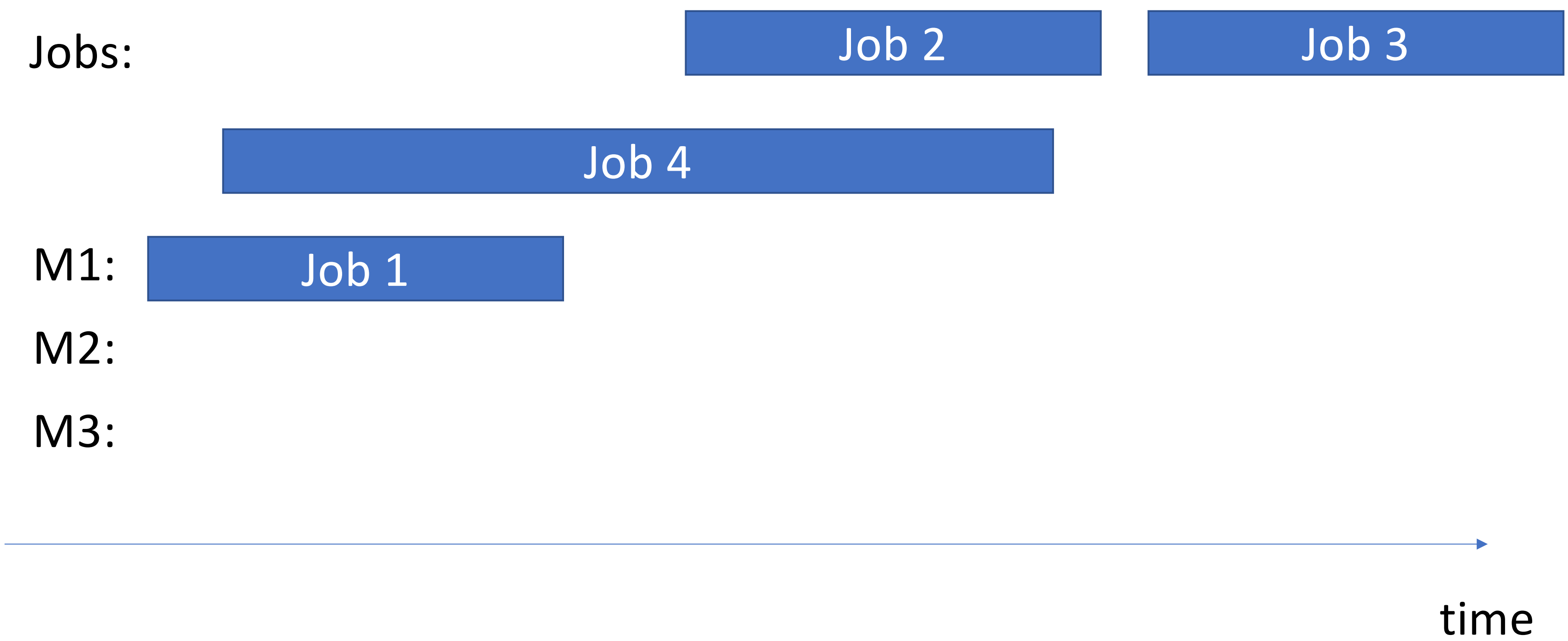
Jobs:



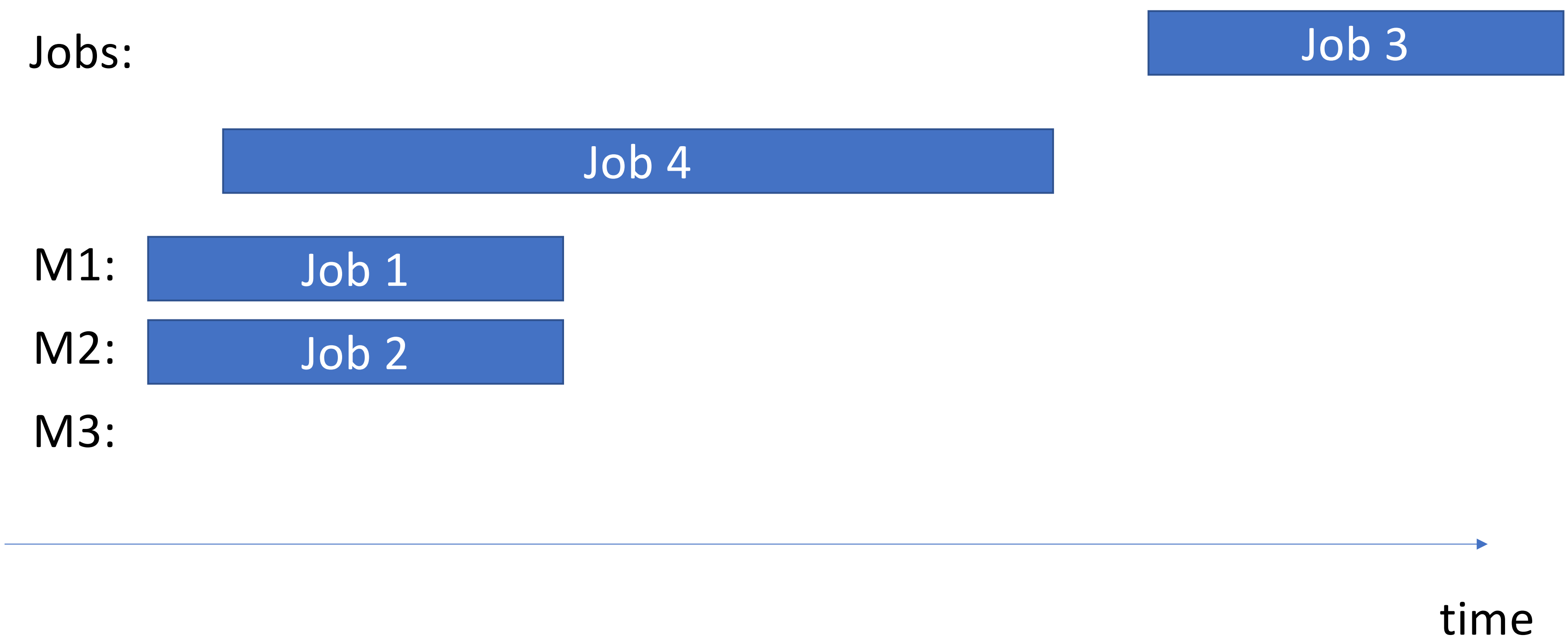
Greedy makespan algorithm **BAD example**:



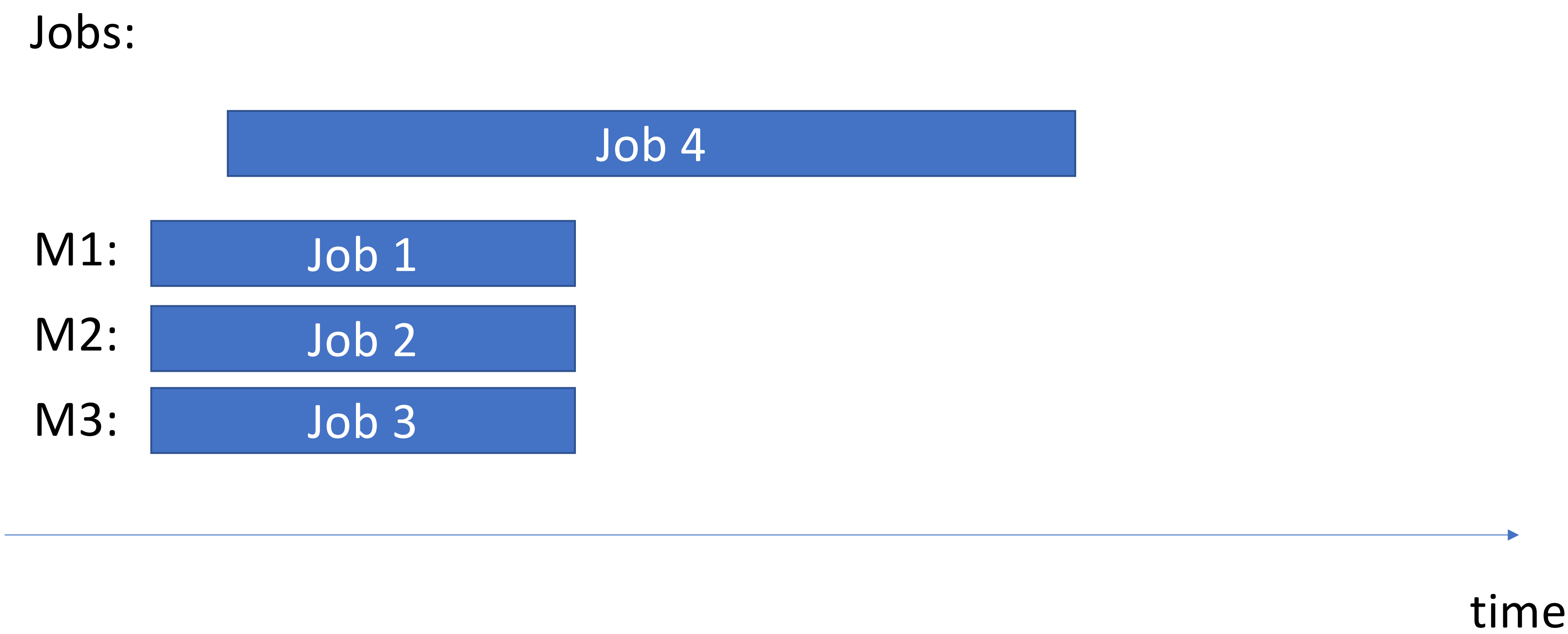
Greedy makespan algorithm **BAD example**:



Greedy makespan algorithm **BAD example**:

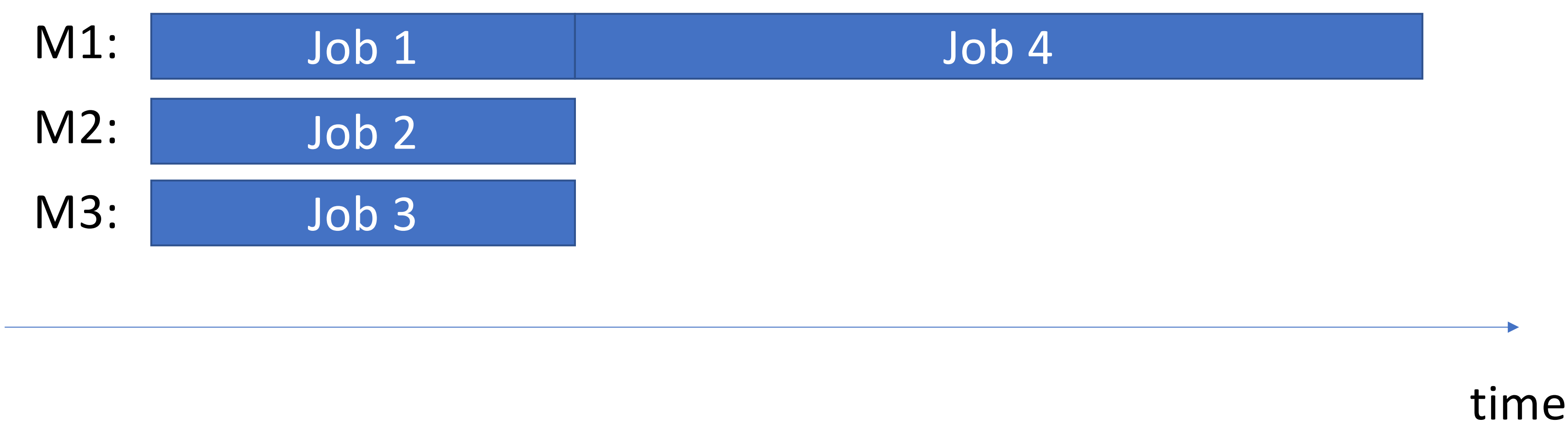


Greedy makespan algorithm **BAD example**:



Greedy makespan algorithm **BAD example**:

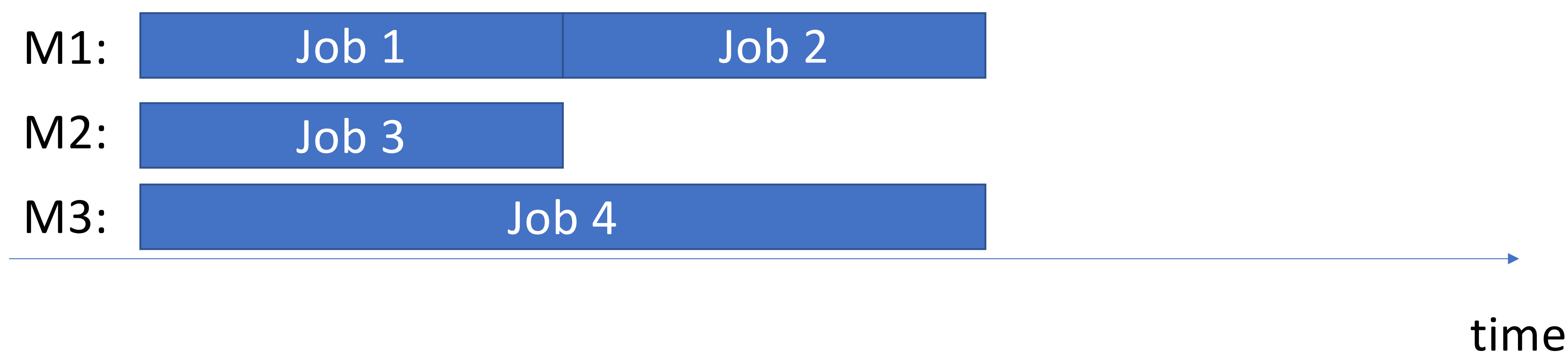
Jobs:



Greedy makespan algorithm **BAD example**:

Greedy solution:

Better solution:



Competitive analysis

How much worse can *greedy* be compared to *offline optimum*?

Theorem

The greedy online algorithm for the Makespan problem with m machines on all inputs I satisfies:

$$\underbrace{ALG(I)}_{\text{Makespan of a solution constructed by the algorithm}} \leq \underbrace{\left(2 - \frac{1}{m}\right)}_{\text{Competitive ratio}} \underbrace{OPT(I)}_{\text{Makespan of an optimal offline solution}}$$

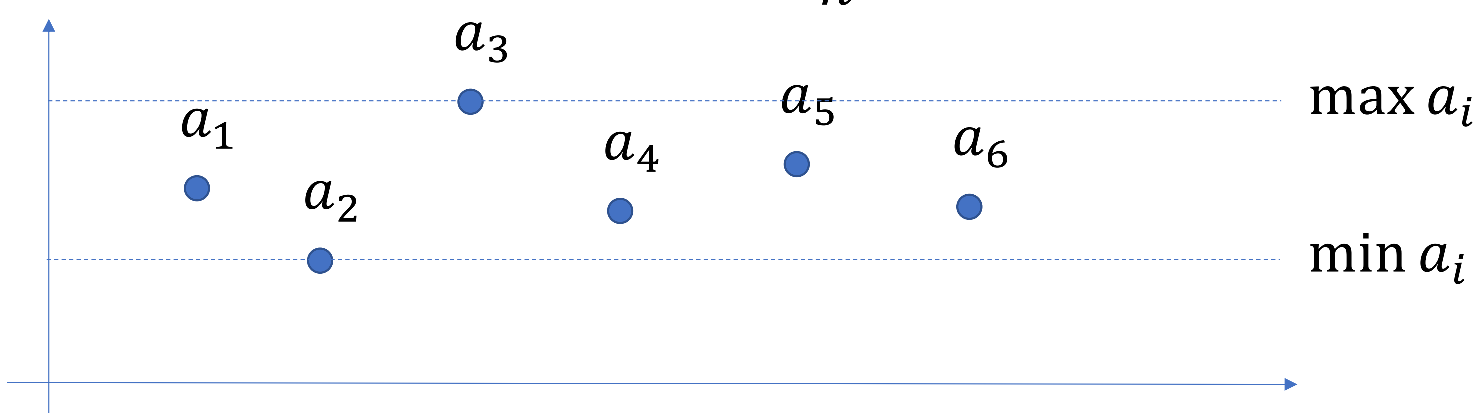
We will use the following simple fact in the proof

Consider a sequence:

$$a_1, a_2, \dots, a_n$$

Then the average is between the smallest and the largest elements:

$$\min a_i \leq \frac{\sum_i a_i}{n} \leq \max a_i$$



Theorem	The greedy online algorithm for the Makespan problem with m machines on all inputs I satisfies: $$ALG(I) \leq \left(2 - \frac{1}{m}\right) OPT(I)$$

Proof:

Let $p_1, p_2, \ldots, p_n$ be an input sequence

Set $p' = \max p_i$

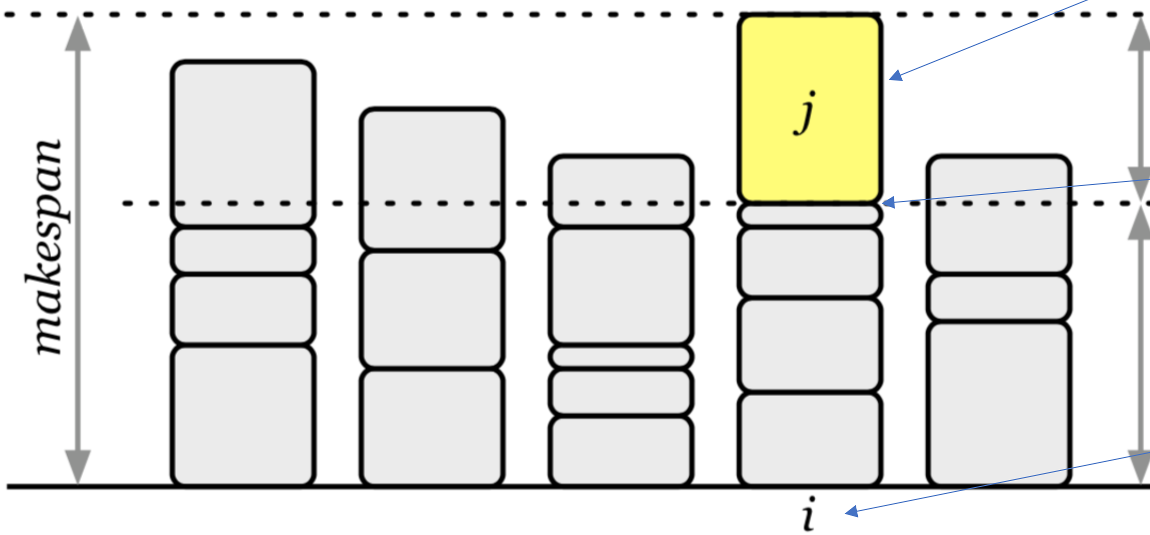
1. $OPT \geq \frac{\sum p_i}{m}$ why?
2. $OPT \geq p'$ why?

Let machine i define the makespan, i.e., have longest completion time

Let p_j be the last job scheduled on machine i

Let q_i be the load on machine i before p_j is scheduled

Proof:



p_j: last job on machine i

q_i: load on machine i before p_j

Machine i has longest completion time

figure from Jeff Erickson's lecture notes

Observation: $q_i \leq \sum_{k \neq j} \frac{p_k}{m}$ (smallest load is at most the average)

Proof:

Putting it together:

$$ALG = q_i + p_j \le \sum_{k \neq j} \frac{p_k}{m} + p_j = \left(\sum_{k=1}^{n} \frac{p_k}{m} \right) - \frac{p_j}{m} + p_j$$

Recalling that $p' = \max p_i$, we have

$$ALG \le \sum_{k=1}^{n} \frac{p_k}{m} + \left(1 - \frac{1}{m}\right) p' \le OPT + \left(1 - \frac{1}{m}\right) OPT \le \left(2 - \frac{1}{m}\right) OPT$$

Since $OPT \ge \frac{\sum p_k}{m}$ and $OPT \ge p'$. QED.

Greedy Makespan algorithm has competitive ratio $\leq \left(2 - \frac{1}{m}\right)$

Moreover, this ratio is *tight*

That is, there exists an input sequence I such that

$$ALG(I) \geq \left(2 - \frac{1}{m}\right) OPT$$

Example of such a sequence I:

$n = m(m-1) + 1$ – number of jobs

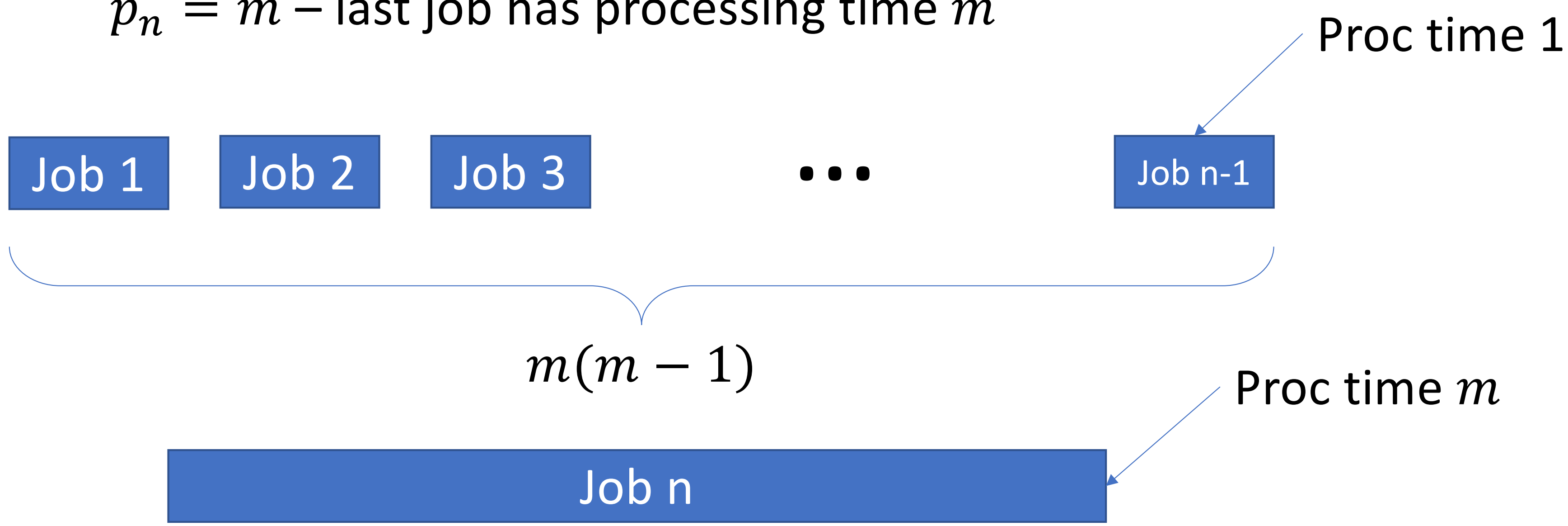
$p_j = 1$ for $j \in \{1, \ldots, m(m-1)\}$ – all but the last job have unit processing times

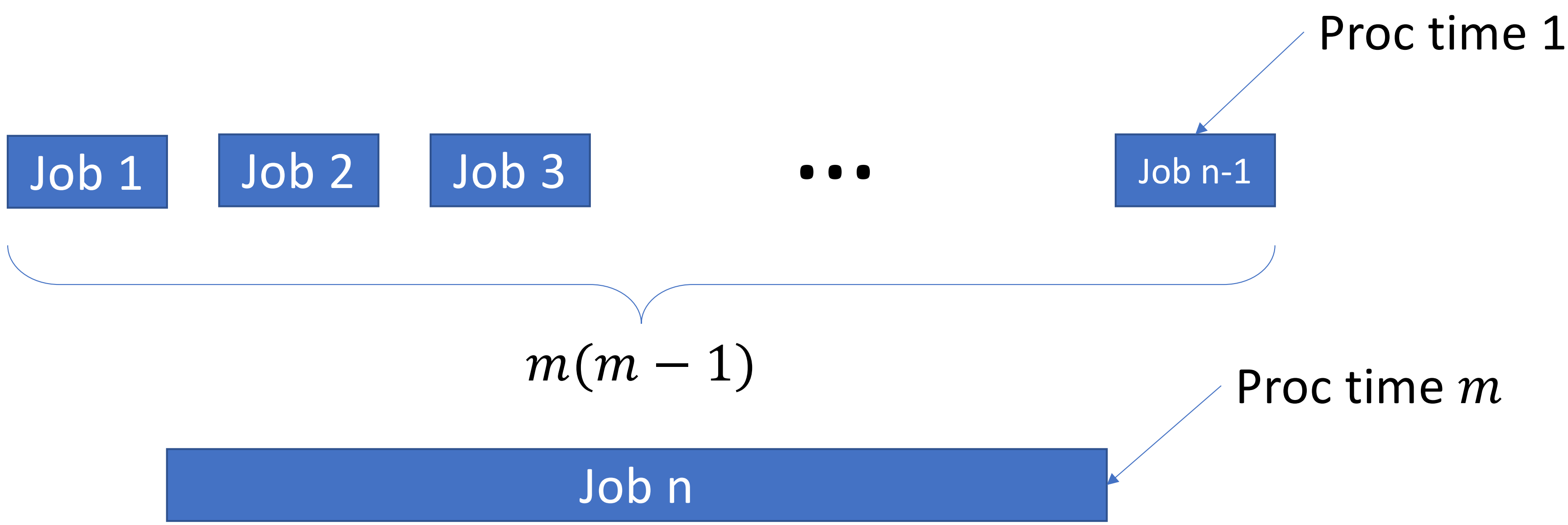
$p_n = m$ – last job has processing time m

$n = m(m-1) + 1$ – number of jobs

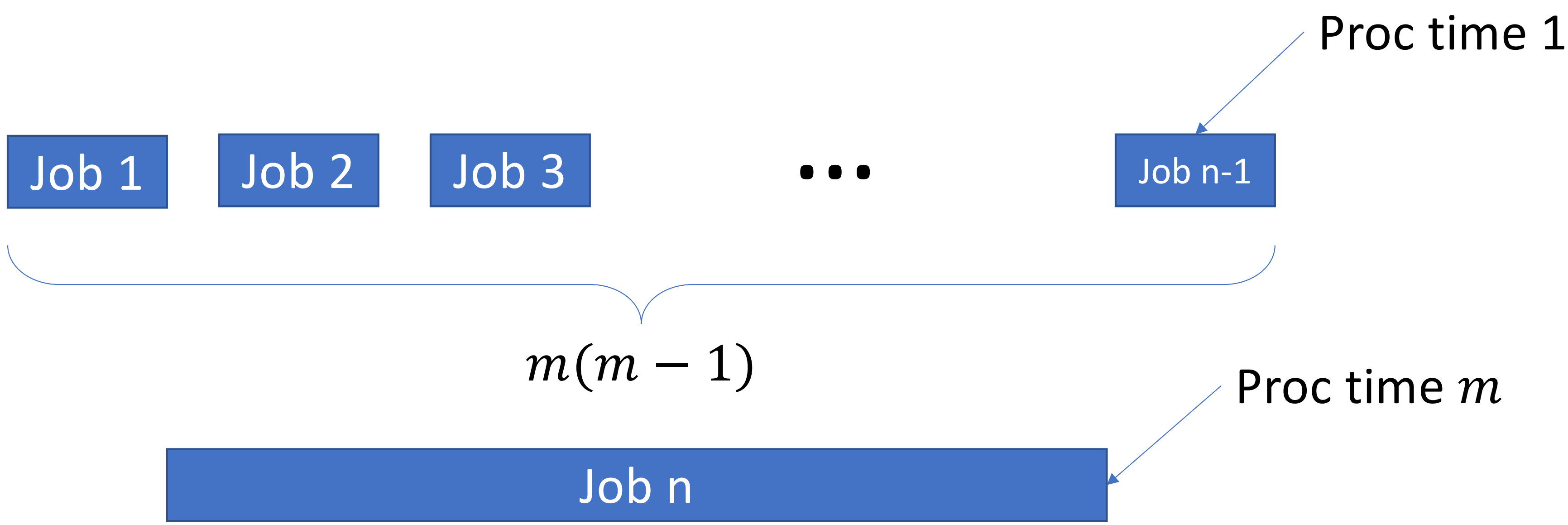
$p_j = 1$ for $j \in \{1, \ldots, m(m-1)\}$ – all but the last job have unit processing times

$p_n = m$ – last job has processing time m





Greedy solution: schedules first $m(m-1)$ small jobs on m machines
results in all machines having makespan $m-1$
has to schedule the last job as well
results in total makespan $m+(m-1)=2m-1$

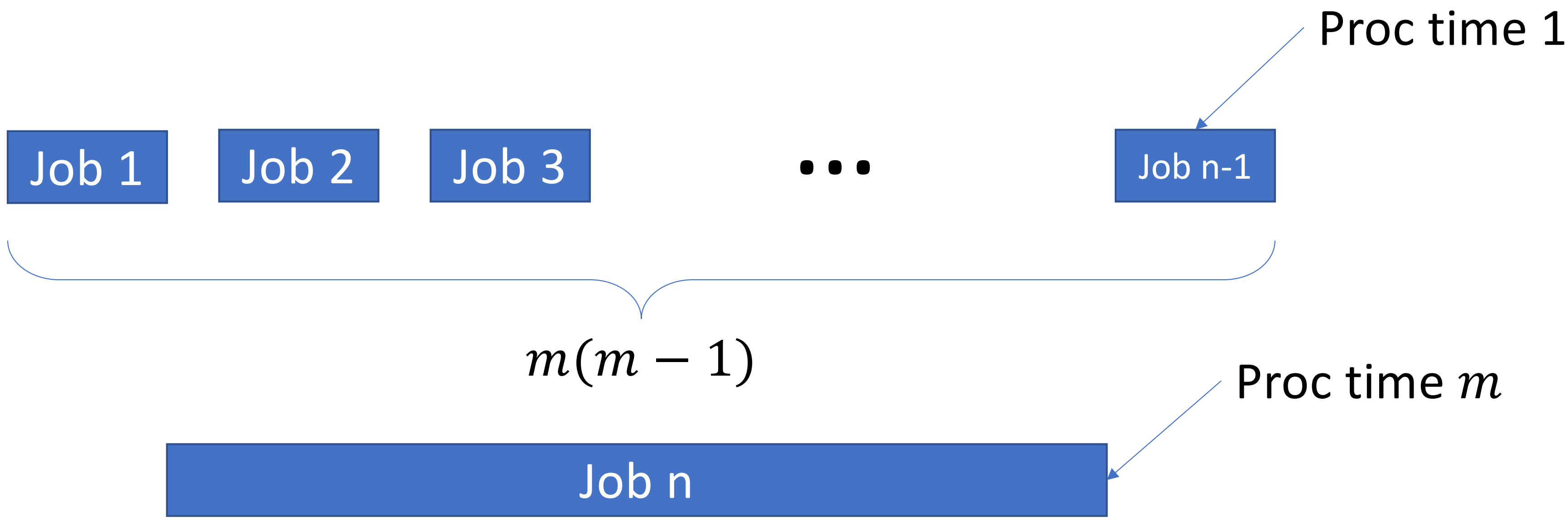


Greedy solution: makespan $m + (m-1) = 2m - 1$

Offline optimum: schedule small jobs on the first $m-1$ machines

schedule last job on the last machine

results in total makespan m



Greedy solution: makespan $m + (m-1) = 2m - 1$

Offline optimum: makespan m

Competitive ratio on this instance: $\frac{2m-1}{m} = 2 - \frac{1}{m}$

The two results together show that

Greedy Makespan algorithm has **tight** competitive ratio $2 - \frac{1}{m}$

But is this competitive ratio best possible?

Does there exist some other (non-greedy) online algorithm with a better competitive ratio?

Currently:

best known positive result (algorithm) is 1.901 (for all m)

best known negative result (adversary) is 1.88 (for large m)

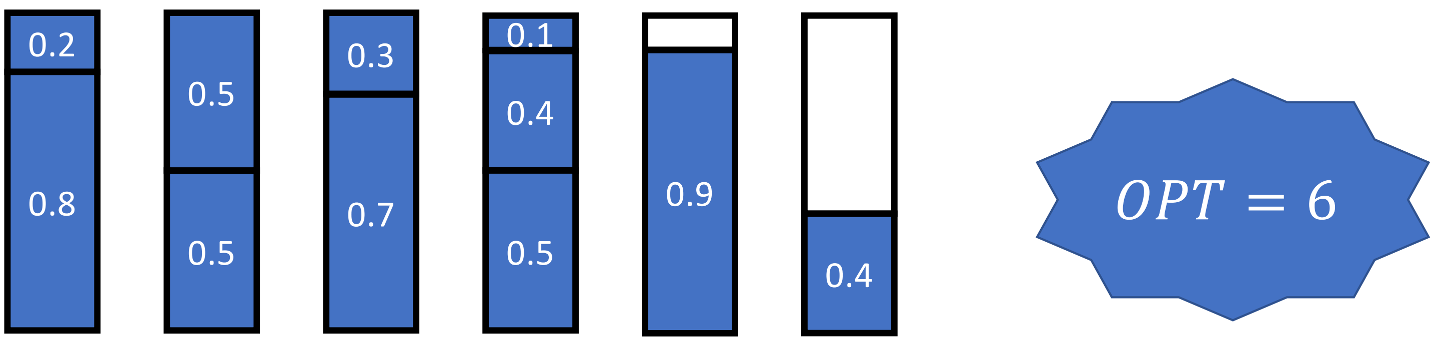
Bin Packing

Bin Packing Problem

The input is a sequence of items, each described by weight $x_i \in [0,1]$

Goal is to pack them into minimum number of bins, each of unit weight capacity

E.g. $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



Bin Packing Problem, formally

Input: $(x_1, \dots, x_n)$, where $x_i \in [0,1]$ is the weight of item i

Output: $\sigma : \{1, \dots, n\} \to \{1, \dots, m\}$ for some integer m

Objective: to find σ as to minimize m subject to

$$\sum_{j:\sigma(j)=i} x_j \leq 1$$

Bin Packing Problem details

Offline version is **NP-hard**, which means

Bin Packing likely does not have *an efficient exact* algorithm

Proved by a reduction from Subset Sum

Offline version can be approximated to within $1 + \epsilon$ for any $\epsilon > 0$ (asymptotically) – PTAS - polynomial time approximation scheme

Many applications in computer memory

Three online algorithms

$NextFit$: if a newly arriving item doesn’t fit in the ***latest*** bin, open a new bin and place the item there.

$FirstFit$: find the ***first*** bin (among already opened ones) that can accommodate a new item and place it there. If the new item doesn’t fit into any existing bins, place it in a new bin.

$BestFit$: find a bin (among already opened ones) that can accommodate a new item ***and leaves the least remaining space***. If the new item doesn’t fit into any existing bins, place it in a new bin.

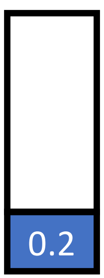
$NextFit$ pseudocode

Algorithm 5 The $NextFit$ algorithm

```
procedure NextFit
    m ← 0                                    ▷ total number of opened bins so far
    R ← 0          ▷ R is the amount of remaining space in the most recently opened bin
    j ← 1
    while j ≤ n do
        if x_j > R then
            m ← m + 1
            R ← 1 − x_j
        else
            R ← R − x_j
        σ(j) ← m
        j ← j + 1
```

$NextFit$ example

Consider $NextFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$

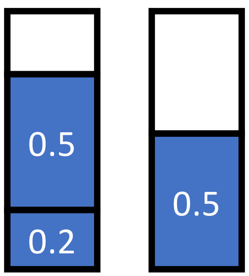


$NextFit$ example

Consider $NextFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$

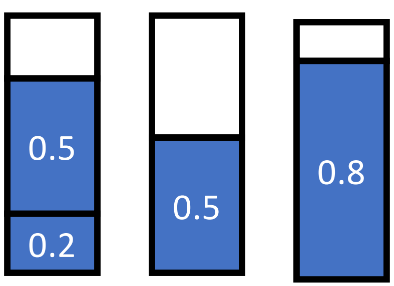
$NextFit$ example

Consider $NextFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



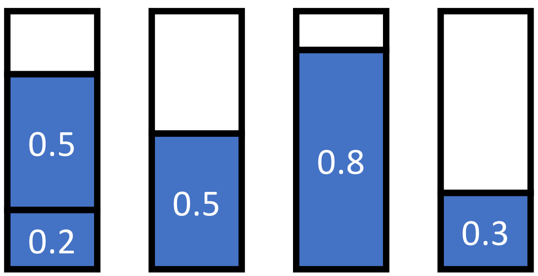
$NextFit$ example

Consider $NextFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



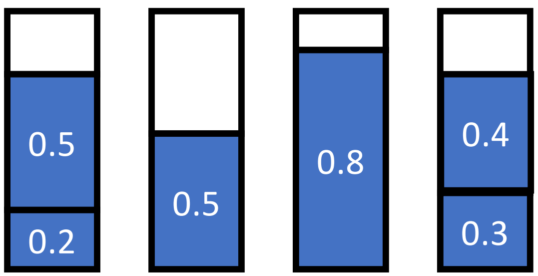
$NextFit$ example

Consider $NextFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



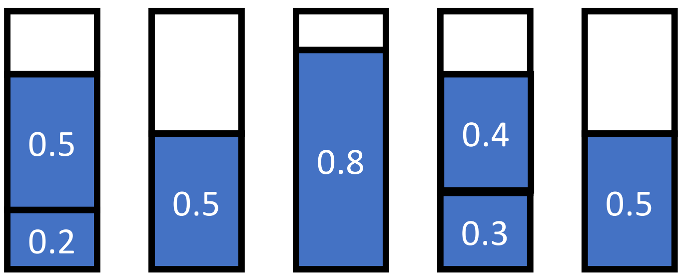
$NextFit$ example

Consider $NextFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



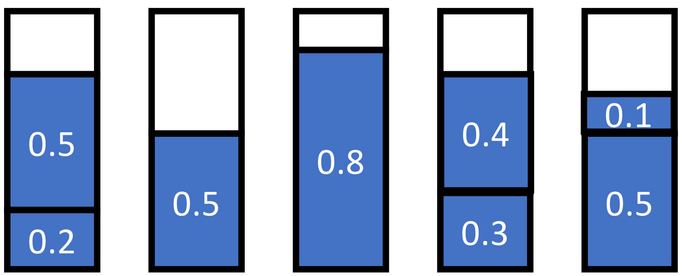
$NextFit$ example

Consider $NextFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



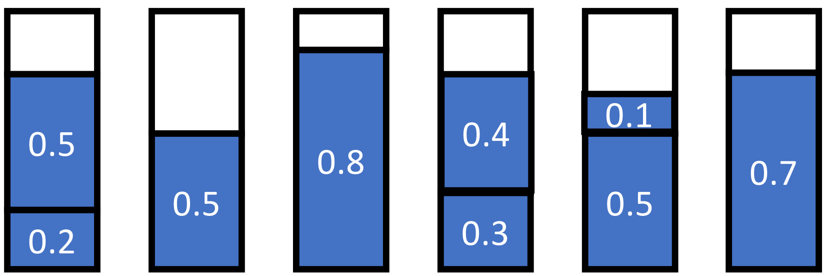
$NextFit$ example

Consider $NextFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



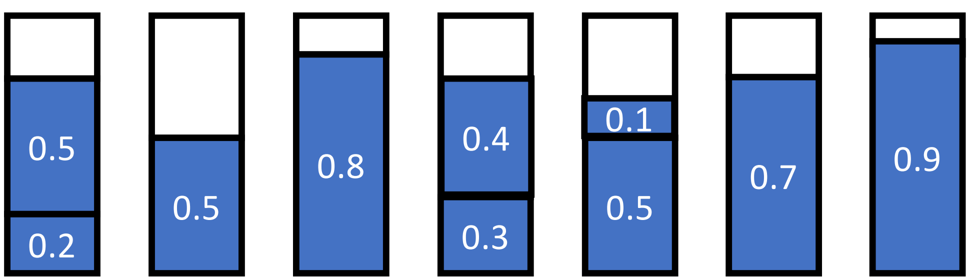
$NextFit$ example

Consider $NextFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



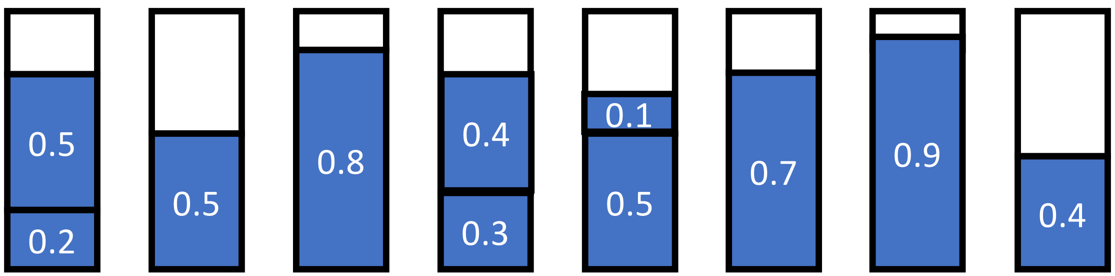
$NextFit$ example

Consider $NextFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



$NextFit$ example

Consider $NextFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



$NextFit = 8$

$NextFit$ negative result

Theorem	$\rho(NextFit) \geq 2$

Proof:

Let $n \in \mathbb{N}$ be arbitrary

Fix $\epsilon = 1/n$

Adversary presents sequence $I = \langle 0.5, \epsilon, 0.5, \epsilon, 0.5, \epsilon, \dots \rangle$ of $2n$ items

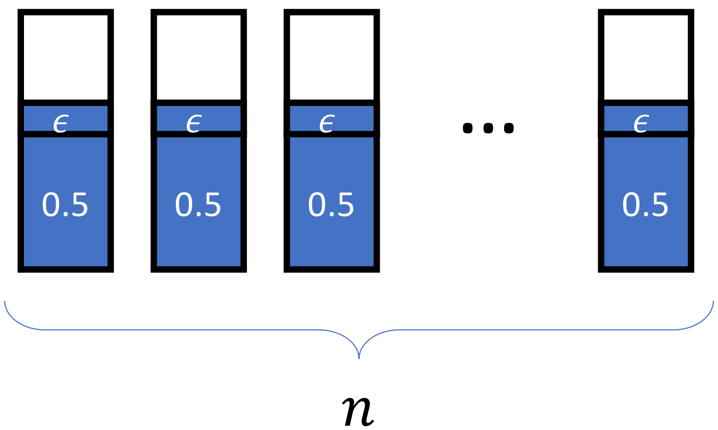
- $NextFit(I) = n$
- $OPT(I) = \frac{n}{2} + 1$

$NextFit$ negative result

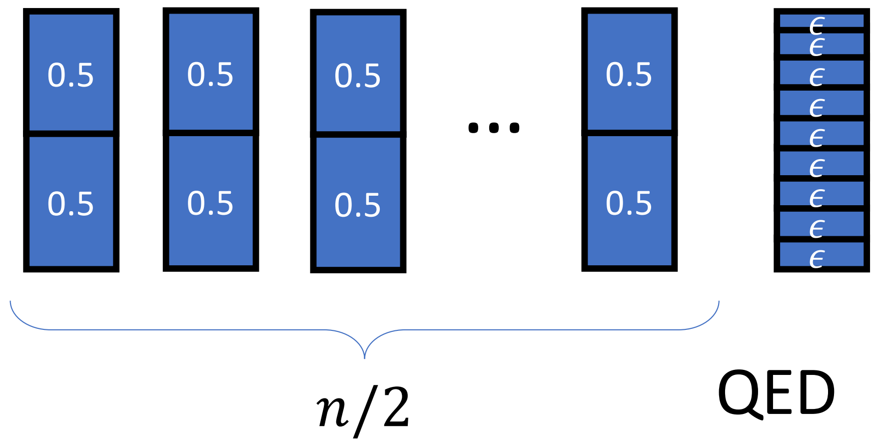
Theorem	$\rho(NextFit) \geq 2$

Proof: $I = \langle 0.5, \epsilon, 0.5, \epsilon, 0.5, \epsilon, \ldots \rangle$ of $2n$ items

$NextFit$



OPT



QED

$NextFit$ positive result

Theorem	$\rho(NextFit) \leq 2$

Proof:

Suppose $NextFit$ uses m bins. Assume m is even for simplicity

Let $B[i]$ be the weight of items in bin i. Then

$$B[1] + B[2] \geq 1$$
$$B[3] + B[4] \geq 1$$
$$\ldots$$
$$B[m-1] + B[m] \geq 1$$

Theorem $\rho(NextFit) \leq 2$

$$B[1] + B[2] \geq 1$$
$$B[3] + B[4] \geq 1$$
$$\dots$$
$$B[m-1] + B[m] \geq 1$$

$$\Rightarrow \sum_{i=1}^{\frac{m}{2}} (B[2i-1] + B[2i]) \geq \frac{m}{2}$$

Thus, we have $\sum_{i=1}^{n} x_i = \sum_{i=1}^{m} B[i] = \sum_{i=1}^{\frac{m}{2}} (B[2i-1] + B[2i]) \geq \frac{m}{2}$

Theorem $\rho(NextFit) \leq 2$

So far, we have $\sum_{i=1}^{n} x_i \geq \frac{m}{2} = \frac{NextFit(I)}{2}$

Lastly, observe $OPT(I) \geq \sum_{i=1}^{n} x_i$

Combining, we have

$$OPT(I) \geq \frac{NextFit(I)}{2}$$

QED

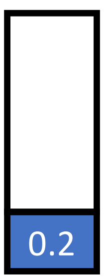
$FirstFit$ pseudocode

Algorithm 6 The $FirstFit$ algorithm

```
procedure FirstFit
    m ← 0                                          ▷ total number of opened bins so far
    R ← a dynamically growing array, initially empty
    ▷ R keeps track of the remaining space in all opened bins
    j ← 1
    while j ≤ n do
        flag ← False
        for i = 1 to m do
            if x_j ≤ R[i] then
                R[i] ← R[i] − x_j
                σ(j) ← i
                flag ← True
                break
        if flag = False then
            m ← m + 1
            Grow the size of R by 1
            R[m] ← 1 − x_j
            σ(j) ← m
        j ← j + 1
```

$FirstFit$ example

Consider $FirstFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$

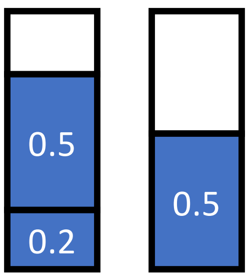


$FirstFit$ example

Consider $FirstFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$

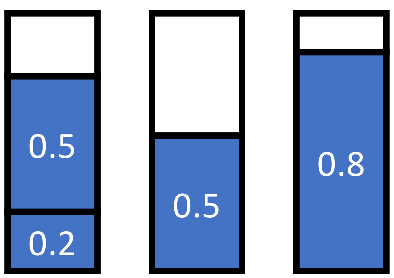
$FirstFit$ example

Consider $FirstFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



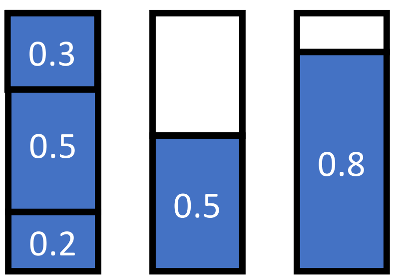
$FirstFit$ example

Consider $FirstFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



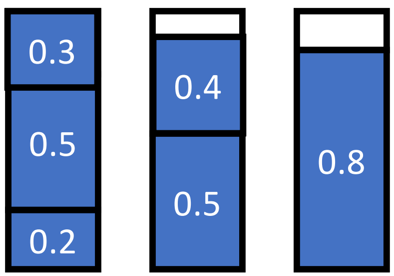
$FirstFit$ example

Consider $FirstFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



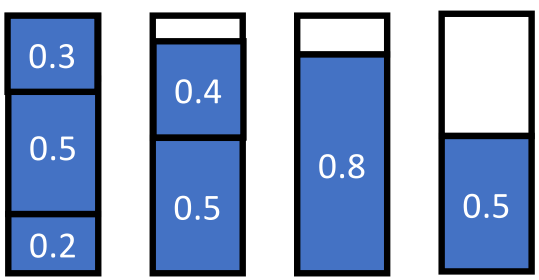
$FirstFit$ example

Consider $FirstFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



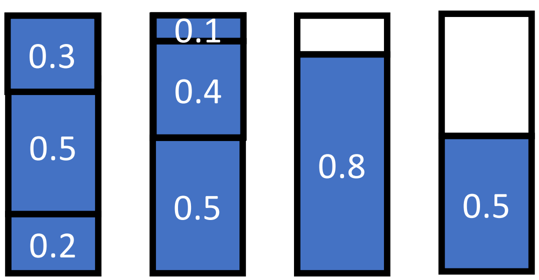
$FirstFit$ example

Consider $FirstFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



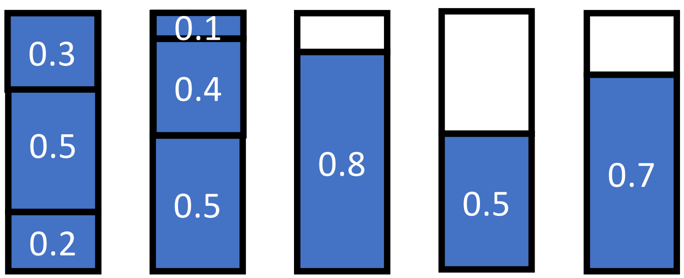
$FirstFit$ example

Consider $FirstFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



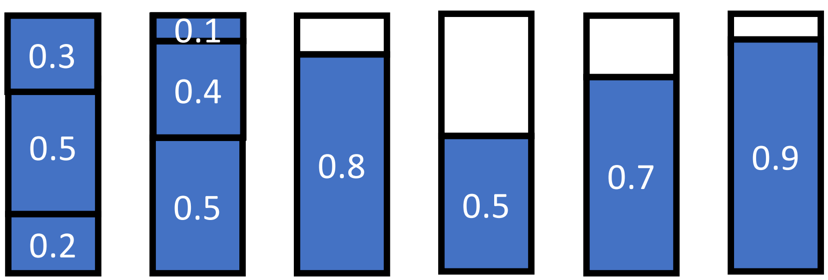
$FirstFit$ example

Consider $FirstFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



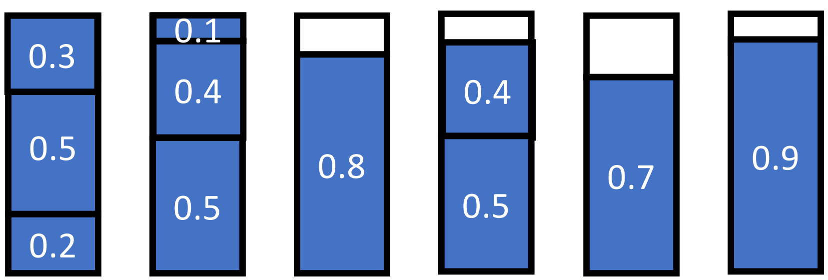
$FirstFit$ example

Consider $FirstFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



$FirstFit$ example

Consider $FirstFit$ on $0.2, 0.5, 0.5, 0.8, 0.3, 0.4, 0.5, 0.1, 0.7, 0.9, 0.4$



$FirstFit$ positive result

Theorem	$\rho(FirstFit) \leq 1.7$

Analysis is moderately complicated – we will actually prove this!

The proof is based on ***the weighting technique***

True weight of an item is x_i

Introduce *weight function* $w : [0,1] \rightarrow \mathbb{R}_{\geq 0}$

The virtual weight of item i is $w(x_i)$

Extend w to indices: for $S \subseteq [n]$ we have $w(S) := \sum_{i \in S} w(x_i)$

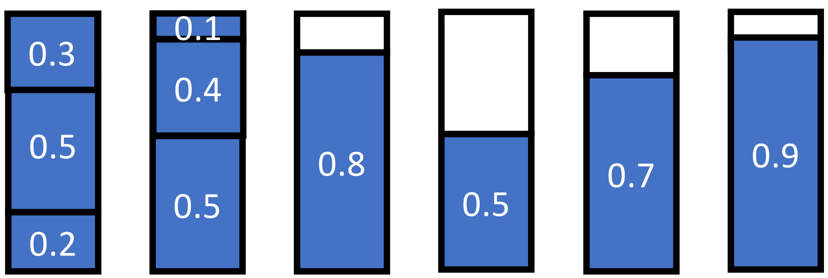
Theorem	$\rho(FirstFit) \le 1.7$

Helpful notation:

Assume $FirstFit$ uses m bins

Let S_i be indices of items in bin i, e.g.,

Index	1	2	3	4	5	6	7	8	9	10	11
Weight	$0.2,$	$0.5,$	$0.5,$	$0.8,$	$0.3,$	$0.4,$	$0.5,$	$0.1,$	$0.7,$	$0.9,$	0.4



$$S_1 = \{1,2,5\}$$
$$S_2 = \{3,6,8\}$$
$$S_3 = \{4\}$$
$$\ldots$$

Note: $w(S_i)$ – virtual weight of all items in bin i

Weighting technique: introduce $w()$. Want 3 key properties:

Property 1: $w(x) \geq x$

Property 2: $FirstFit$ satisfies $w(S_i) \geq 1 - \beta_i$ for some numbers $\beta_i \geq 0$ such that $\sum_i \beta_i$ is bounded by a small constant

"almost all bins created by $FirstFit$ have virtual weight at least close to 1"

Property 3: for any $k \in \mathbb{N}$ and $y_1, \ldots, y_k \in [0,1]$ we have

$$\sum_i y_i \leq 1 \Rightarrow \sum_i w(y_i) \leq \gamma$$

"virtual total weight of a bin never exceeds γ"

Weighting technique explained

If you find virtual weight function with the above property for some γ, then you immediately have competitive ratio $\leq \gamma$

P2: “almost all bins created by $FirstFit$ have virtual weight at least close to 1”

Implies: sum of all virtual weights is at least roughly $m = FirstFit$

P3: *“virtual total weight of a bin never exceeds γ”*

Implies: sum of all virtual weights is at most $\gamma\ OPT$

Weighting technique explained

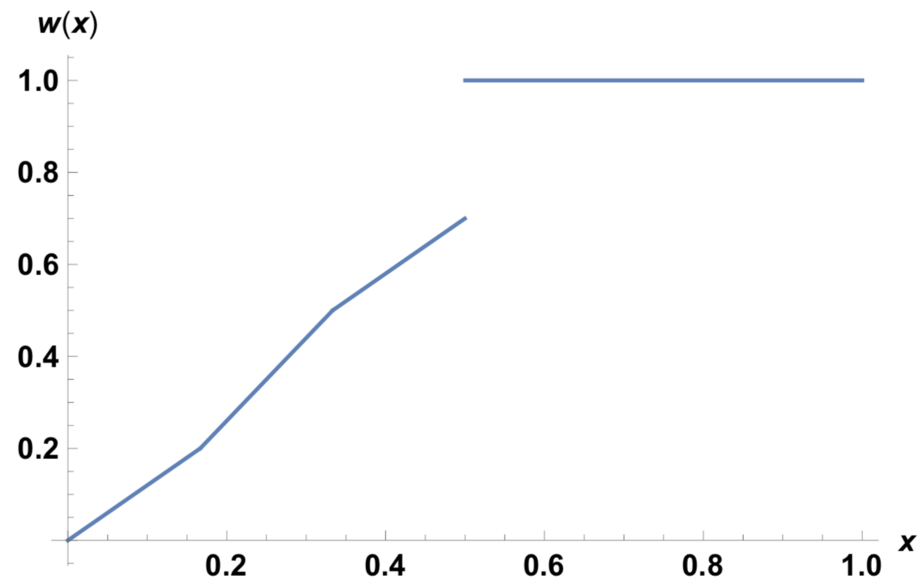
Thus we have

$$FirstFit = m \leq \text{sum of all virtual weights} \leq \gamma\ OPT$$

Thus, we “only” need to find a good w and prove its properties!

The weight function for $FirstFit$

$$w(x) = \begin{cases} \frac{6}{5}x & \text{for } 0 \le x \le \frac{1}{6}, \\ \frac{9}{5}x - \frac{1}{10} & \text{for } \frac{1}{6} < x \le \frac{1}{3}, \\ \frac{6}{5}x + \frac{1}{10} & \text{for } \frac{1}{3} < x \le \frac{1}{2}, \\ 1 & \text{for } \frac{1}{2} < x \le 1. \end{cases}$$



$FirstFit$ positive result

After a lot of calculations (try it!)…

$$\rho(FirstFit) \leq 1.7$$

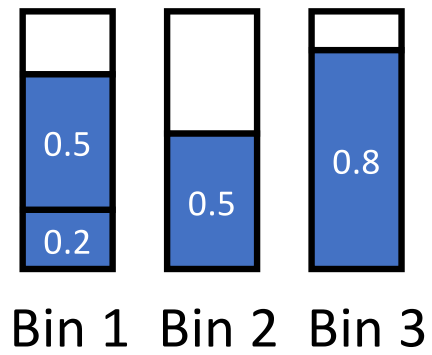
$BestFit$ pseudocode

Algorithm 7 The $BestFit$ algorithm

```
procedure BestFit
    m ← 0                                  ▷ total number of opened bins so far
    R ← 1          ▷ array R keeps track of remaining space in all opened bins
    j ← 1
    while j ≤ n do
        ind ← −1          ▷ ind will be the index of the bin having the best fit
        for i = 1 to m do
            if x_j ≤ R[i] then
                if ind = −1 or R[i] < R[ind] then
                    ind ← i
        if ind = −1 then
            m ← ind ← m + 1
            R[m] ← 1
        σ(j) ← ind
        R[ind] ← R[ind] − x_j
        j ← j + 1
```

$BestFit$, difference from $FirstFit$

Suppose the following is the current configuration:



Furthermore, suppose next item has weight 0.2

It will be assigned to Bin 3 in $BestFit$ and Bin 1 in $FirstFit$

$BestFit$ positive result

Theorem	$\rho(BestFit) \leq 1.7$

Proof:

Exactly the same proof works as for FirstFit (try it!)

QED

$BestFit$, $FirstFit$: outstanding issues

Possible to prove that the positive results for the two algorithms are tight, i.e.

$$\rho(FirstFit) \geq 1.7, \qquad \rho(BestFit) \geq 1.7$$

Easy to show that for any (deterministic or randomized) algorithm ALG we have

$$\rho(ALG) \geq \frac{4}{3}$$

The *best known results* for Bin Packing are

1.5403… (negative result) and 1.57829… (positive result)