The benefit of preemption
(for scheduling problems)

Leah Epstein

Department of Mathematics
University of Haifa

Scheduling seminar, Sept 29, 2021
schedulingseminar.com

Plan

We would like to compare two types of schedules for identical inputs

1. **Reminder/definition of the models**
2. **Definition of benefit/power of preemption**
3. **Examples and simple proofs for several variants**

Goals:

1. **To get an intuition of why one model (the preemptive one) is easier in a sense, compared to the other one (non-preemptive)**
2. **To see why such problems are attractive**

Scheduling paradigms - summary

A scheduling problem consists of

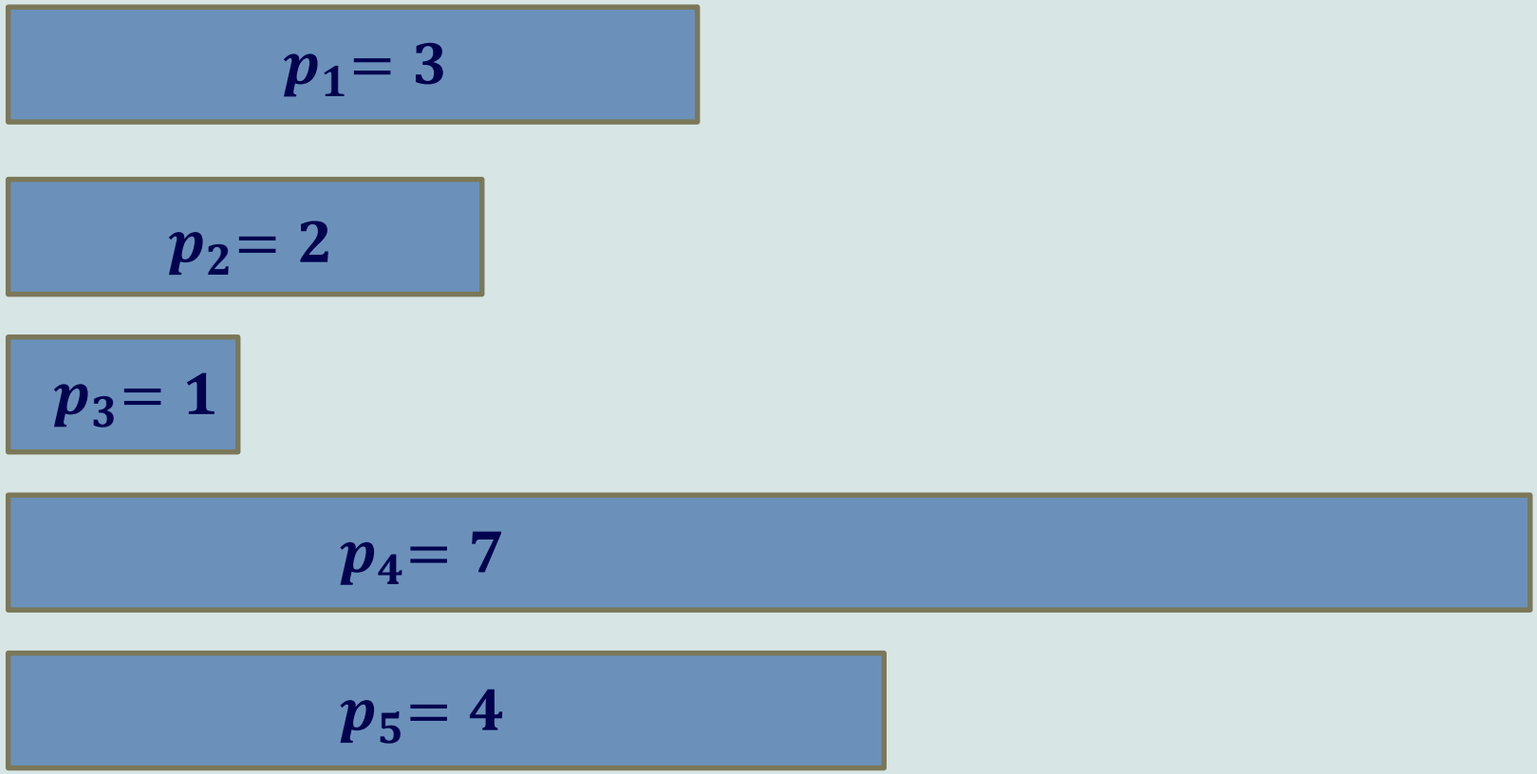
- **Jobs and their attributes**
  - **Size/processing time**
  - **Release date (the job cannot be started before this time)**
  - …
- **The number of machines and their types**
  - **One machine**
  - **Parallel identical machines**
  - **Uniformly related machines (with speeds)**
  - …
- **An objective/goal function**
  - **Makespan**
  - **Total completion time, total weighted completion time**
  - **ℓ_p norm**
  - …
- **Feasible schedules**
  - **preemptive or non-preemptive**
  - …

Jobs - notation

There are n jobs

Jobs have indexes 1,2,3,...,n

The size of job j is a positive integer p_j



Assigning/scheduling a job

A non-preemptive schedule

Every job must be executed completely

Every job receives a **continuous time slot**

on some machine

A preemptive schedule

Every job must be executed completely but..

a job may be **split into parts**

Parts can be assigned to run separately

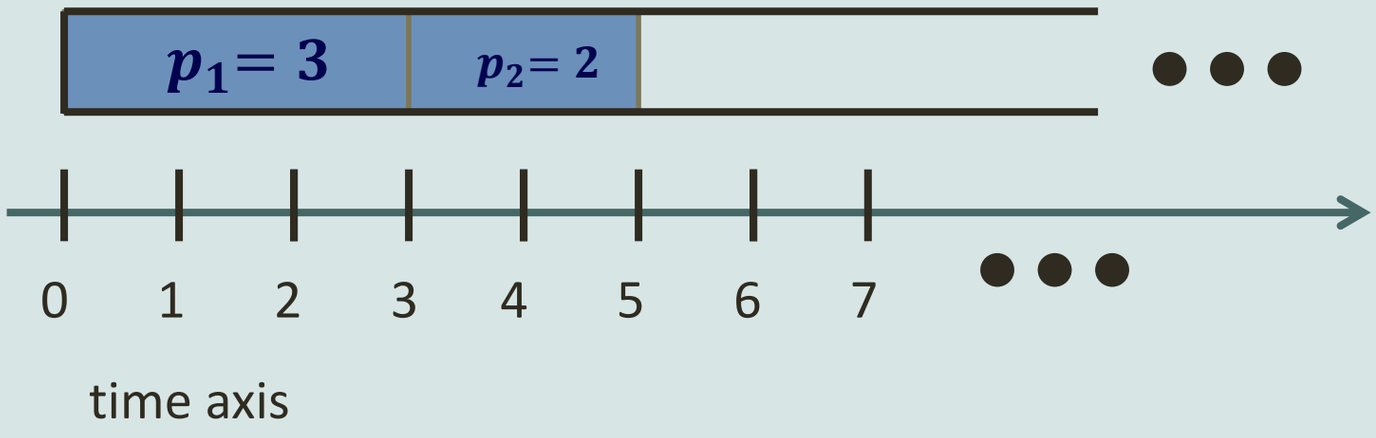
On the same machine or different machines

never at the same time

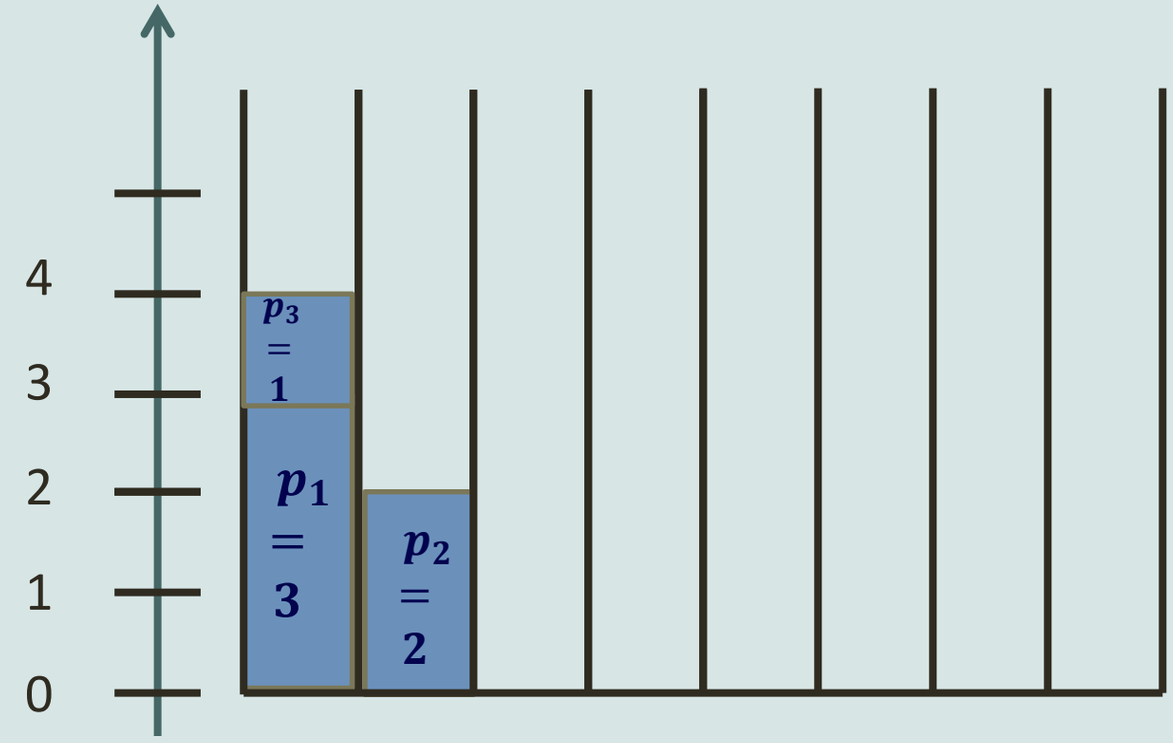
Machines

- Each machine can be used for running jobs
  - Available continuously starting from time zero
    - We do not consider other models here
    - We exhibit concepts using the most standard models
- A machine can process at most one job at each time
  - Any job can be processed on one machine at each time
  - No resource sharing and no parallel processing

One machine

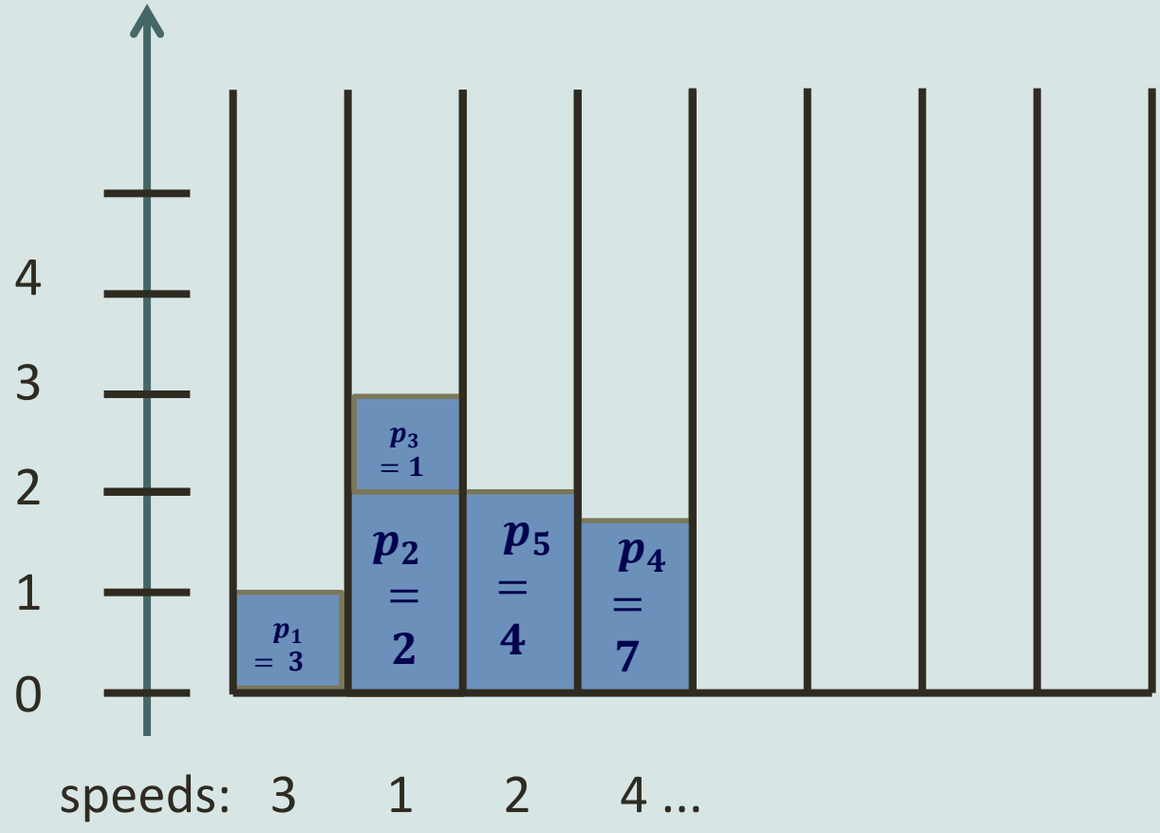


Parallel identical machines - illustrations



Uniformly related machines

$s_i \geq 0$ **is the speed of machine i**



Processing time of job j on machine i

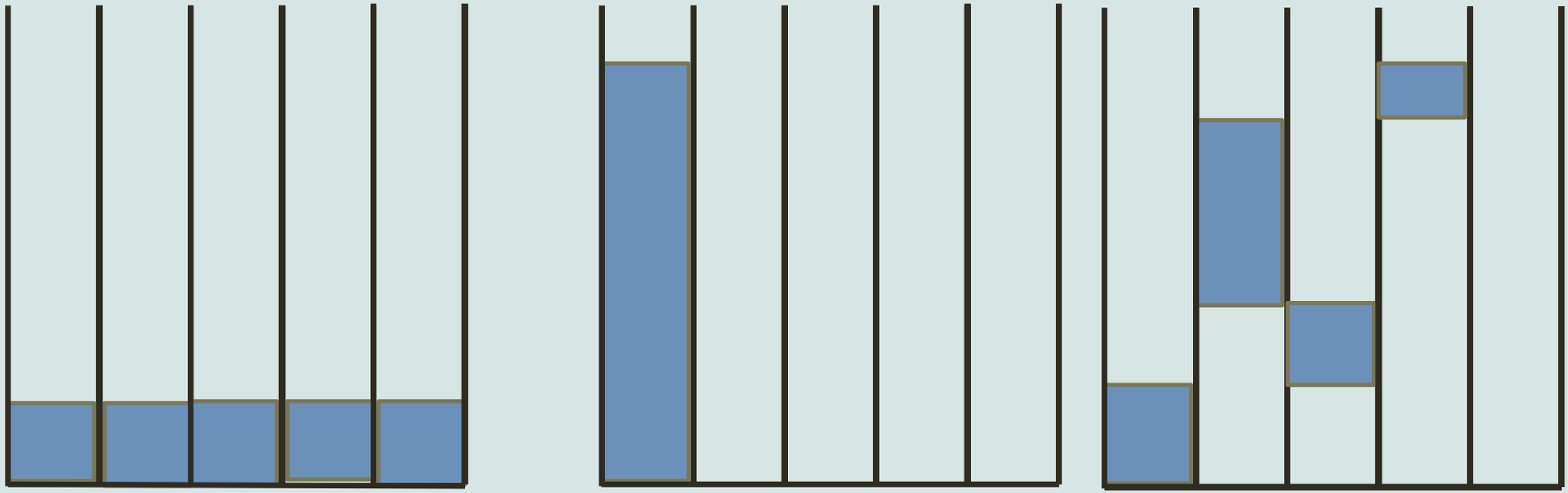
$$\frac{p_j}{s_i}$$

Preemptive schedules versus Non-preemptive schedules

- In a **non-preemptive** schedule, every job is assigned to one machine
  - It is also assigned to a time **slot to run completely**
  - It is assigned to its time slot, to **run continuously** on its machine
- In a **preemptive** schedule **every job is also assigned to run completely**
  - But it can be split into (a finite number of) parts
  - Every part is assigned separately
  - **Parts cannot run in parallel**
- **Fractional scheduling**
  - A job is split into parts
  - Every part is assigned independently
  - **Parts can run in parallel**

Differences: example

- Identical machines
- There is just one job of size m - the number of machines



Fractional:
the job is split
equally

non-
preemptive

Preemptive:
splitting will not help

Makespan: last completion time
equal to 1 for the fractional schedule, m for the integral one

Preemptive versus non-preemptive

Three jobs, each of size 2

Preemptive – splitting helps

Non-preemptive

Makespan: last completion time

3 for the preemptive schedule, 4 for the non-preemptive one

Benefit of preemption
or power of preemption

A measure

of how much **allowing preemption can reduce costs** of solutions

For a given scheduling problem/variant and input I

Let $OPT_p(I)$ denote the cost of an optimal **preemptive** solution

Let $OPT_n(I)$ denote the cost of an optimal **non-preemptive** solution

$$OPT_n(I) \geq OPT_p(I)$$

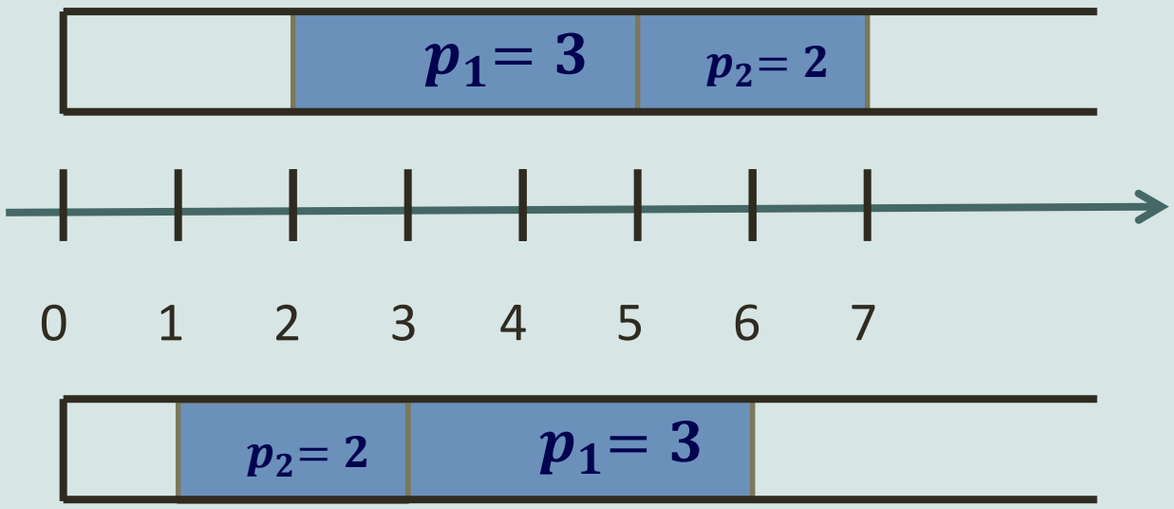
The set of preemptive solutions is a superset of non-preemptive solutions, for the same problem

The benefit of preemption is

$$\sup_I \frac{OPT_n(I)}{OPT_p(I)} \geq 1$$

In general: the benefit of preemption for a certain problem may be infinite or 1 or anything in between

Attributes: weights, release dates and **objectives**



$p_1 = 2$ $p_2 = 1$

$r_1 = 2$ $r_2 = 1$

$w_1 = 11$ $w_2 = 2$

Makespan: 7 for the top schedule, 6 for the bottom one

Total completion time or sum of completion times of all jobs
12 for the top schedule, 9 for the bottom one

Total weighted completion time or sum of weighted completion times of all jobs
69 for the top schedule, 72 for the bottom one

Which schedule is "better"?

Useful notation and simple properties

- For a fixed input I **with** **m machines , n jobs**

P denotes the total size of all jobs **q denotes the maximum job size**

Properties valid for both preemptive and non-preemptive schedules:

- **For a single machine, the makespan is at least P**
- **For m identical machines, the makespan is at least** $\frac{P}{m}$

the pigeonhole principle – averaging

It is also at least q - every job must be processed and no parallelism is allowed

- For m **uniformly related machines**, the makespan is at least $\frac{P}{\sum_i s_i}$

the pigeonhole principle – weighted averaging based on speeds

It is also at least $\frac{q}{\max s_i}$

- even if it runs on the fastest machine, it needs to be completed

Topics

- Makespan
  - One machine with release dates
  - Identical machines
  - Uniformly related machines

- Total (weighted/unweighted) completion times
  - One machine with release dates
  - Identical machines
  - Uniformly related machines

Makespan, one machine, release dates: easy

This variant has an optimal algorithm. It is optimal both for the preemptive variant and the non-preemptive one **The benefit of preemption is therefore 1**

The algorithm: Assume (by sorting) that $r_1 \le r_2 \le \cdots \le r_n$

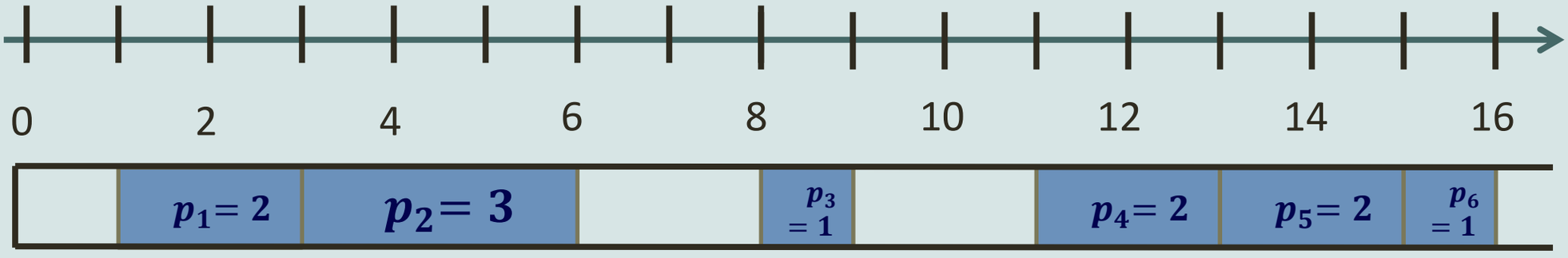
Assign job 1 at time $S_1 = r_1$, into the time slot $[S_1, T_1) = [r_1, r_1 + p_1)$

For i=2,3,...,n in this order:

Assign job i at time $S_i = \max\{r_i, T_{i-1}\}$, into the time slot $[S_i, S_i + p_i)$

j	1	2	3	4	5	6
p_j	2	3	1	2	2	1
r_j	1	2	8	11	12	13

In other words, assign every job as early as possible, given the already assigned jobs and its release date



Optimality

Consider the suffix of jobs scheduled consecutively – "block"

y, y+1, y+2, …, n for some $1 \leq y \leq n$

y=4 in the example

They are scheduled starting time r_y

The makespan is $L = r_y + \sum_{j=y}^{n} p_j$

where we have $r_j \geq r_y$ **for $j \geq y$**

In the example $r_y = 11$ *and* $\sum_{j=y}^{n} p_j = p_4 + p_5 + p_6 = 5$

No valid schedule preemptive or non-preemptive can complete all jobs earlier

Makespan:

Identical machines **and** Uniformly related machines

Woeginger, 2000 no release dates, no weights

The proofs are not complicated

This allows us to see the full analysis

The **tight bound** for uniformly related machines is $\frac{2m-1}{m} = 2 - \frac{1}{m}$

can be improved to $\frac{2m}{m+1} = 2 - \frac{2}{m+1}$ for equal speeds – **identical machines**

For m=2 the bounds are 1.5 and 4/3

Before we try to prove this, a question:

what can be say about optimal algorithms?

It can be useful because we compare optimal solutions

Non-preemptive: the problems are NP-hard

Preemptive: McNaughton designed a simple and intuitive optimal algorithm for identical machine, about 60 years ago

McNaughton's algorithm

Consider a fixed input I and let $T = \max\left\{\frac{P}{m}, q\right\}$

Recall that q and $\frac{P}{m}$ are lower bounds on the makespan

There exists a preemptive schedule with makespan T

The algorithm:

Assign jobs only during the time slot [0,T) on every machine

- Start from the first machine at time zero
- Assign jobs one after the other without idle time
- **Cut a job if time T is reached and continue**
  to schedule jobs starting from time 0 on the next machine
- Do this until all jobs are scheduled

McNaughton's algorithm, example

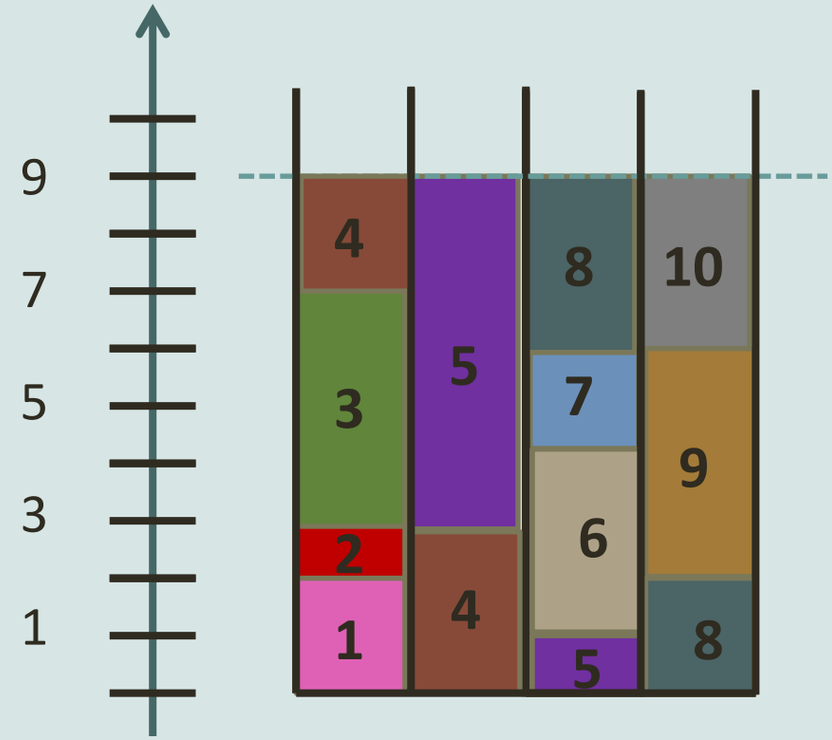
job index	1	2	3	4	5	6	7	8	9	10
size	2	1	4	5	7	3	2	5	4	3

m=4 and we have P=36, q=7, T=9

Properties:

Jobs split into two parts are assigned to their later time slots first

If q>P/m, there will be empty times before time T on the last machines and possibly on other machines too



Correctness: since $T \geq \frac{P}{m}$ there is space for all jobs

since $T \geq q$ there is no overlap between parts of one job

Identical machines, lower bound

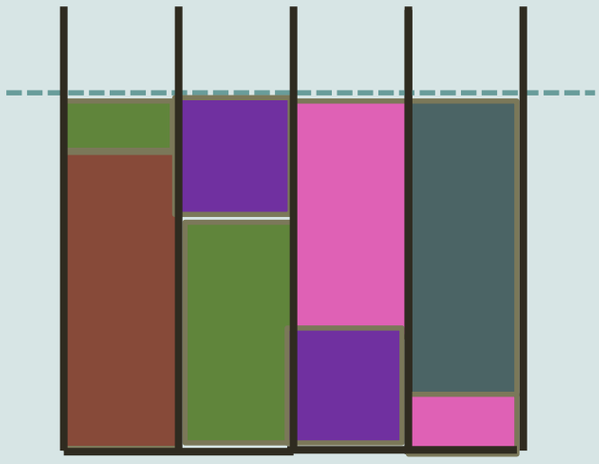
Consider m+1 identical jobs each of size m

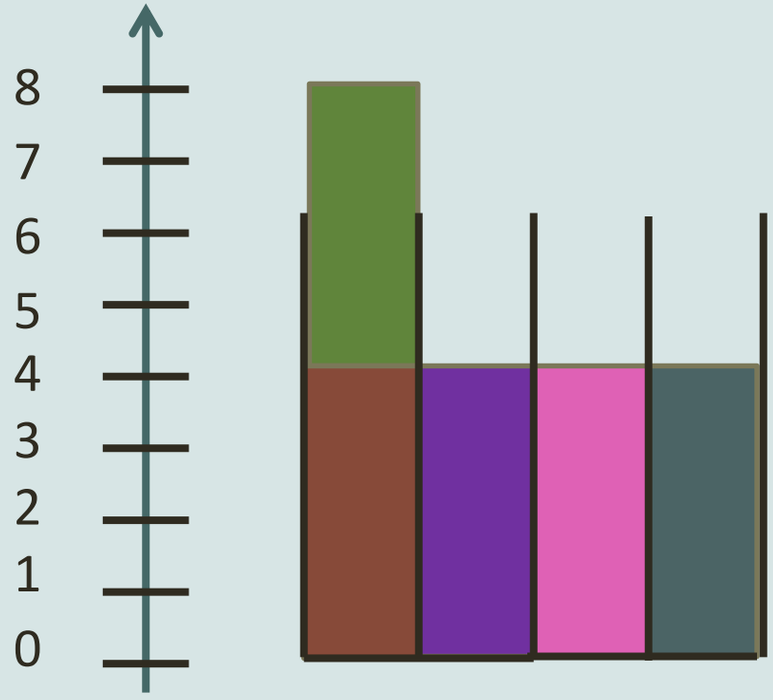
A non-preemptive solution has a machine with at least two jobs (the pigeonhole principle) and its makespan is at least 2m

For a preemptive solution: $T = \min\left\{m, \frac{m(m+1)}{m}\right\} = m + 1$

This gives a ratio of at least $\frac{2m}{m+1}$

An illustration of the schedules for m=4:





Identical machines, upper bound

Algorithm LPT will be used for the analysis

It is **simple** and creates a **valid non-preemptive schedule**

Its makespan **bounds the optimal one from above**

Most likely there is no simple optimal algorithm

- even an exponential one – but we find an upper bound on its cost

LPT (longest processing time)

Sort the jobs by non-increasing size to create a list

and apply LS (List Scheduling, Graham 1966)

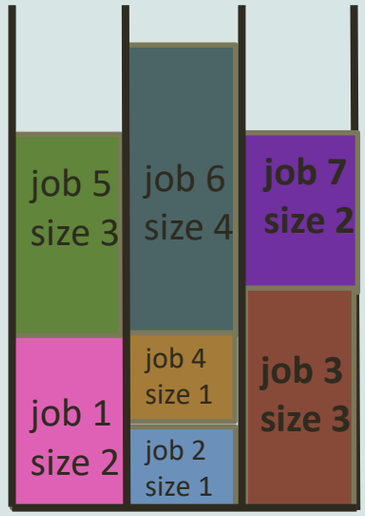
LS Process the input list of jobs, **assigning every job to the machine that will complete it first** - breaking ties arbitrarily

For identical machines it is **equivalent to assigning to the machine of current minimum completion time**

Two examples for m=3

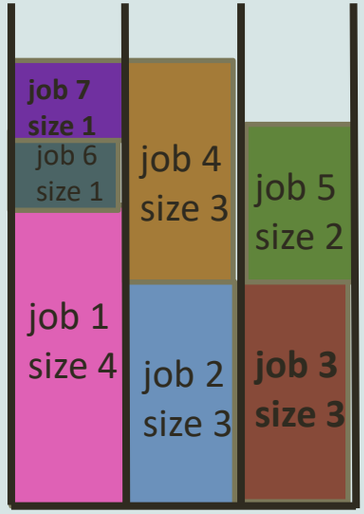
LS

job index	1	2	3	4	5	6	7
size	2	1	3	1	3	4	2



LPT

job index	1	2	3	4	5	6	7
size	4	3	3	3	2	1	1



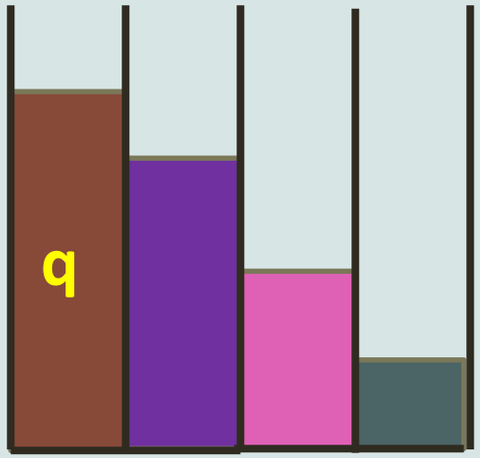
Analysis for the benefit of preemption, for identical machines, makespan, using LPT

If there are at most m jobs

LPT assigns each job to a different machine

the schedule is optimal,

since its makespan is at most q



Otherwise:

Assume that the job of maximum completion time is the last one

If this is not the case, remove all further jobs, the makespan is unchanged for LPT and an optimal preemptive solution cannot have a larger makespan – because the new input is a subset of the old one

Let p be the last job and its size.

Any other job has size p or more **because of the sorted order**

P is the total size of all brown jobs and the pink job p

Consider job p, **there are at least m+1 jobs including p**

Let i be the machine of p in the LPT schedule

Let X_k be the completion time of machine k

just before the last job is assigned

The makespan of LPT is $L = X_i + p$

Since i is selected such that p is completed as early as possible

for every k we get $X_k \geq X_i$

If we add a job of size p to every machine,

all machine "completion times" will be at least L

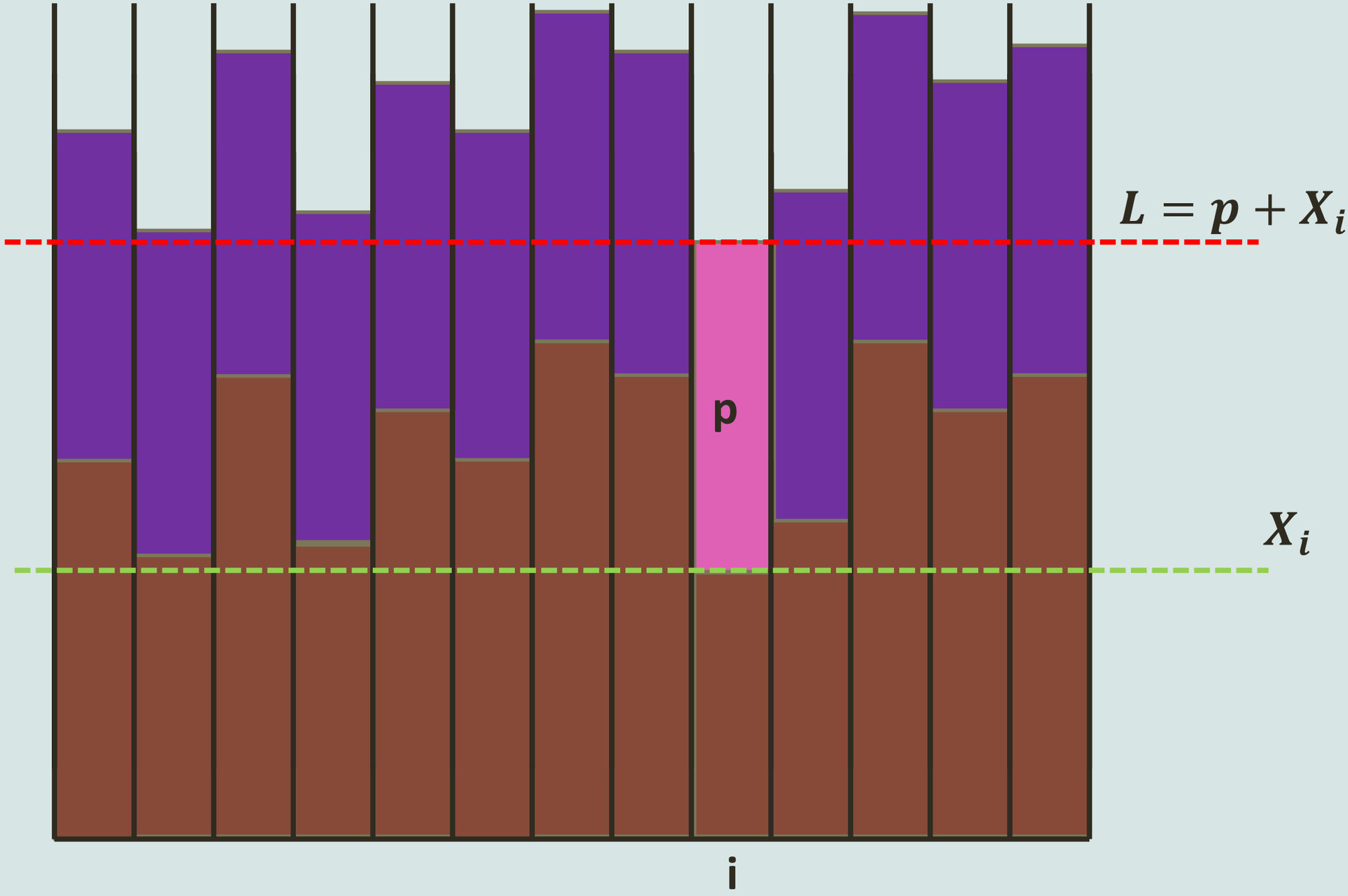
so $P' \geq m \cdot L$ **and thus** $L \leq \frac{P'}{m}$

Where P' is the total size of old jobs and added jobs

P'=P+(m-1)p

m-1 and not m because there was one such job of size p in the input

and when we added such jobs only m-1 are new



P is the total size of all brown jobs together with the pink job p.

P' is the total size of all jobs

Benefit of preemption

The preemptive optimal cost is at least P/m

We use $L \leq \frac{P'}{m}$ **and** **P'=P+(m-1)p**

Also **P $\geq$(m+1)p** **because there are at least m+1 jobs**

Due to the sorted order, each has size at least p **so** $p \leq \frac{P}{m+1}$

The non-preemptive optimal cost is at most

$$L \leq \frac{P'}{m} = \frac{P + (m-1)p}{m} \leq \frac{P + \frac{P(m-1)}{m+1}}{m} = \frac{2m}{m(m+1)} \cdot P$$

Comparing to P/m, the ratio between the costs is at most $\frac{2m}{m+1}$

Uniformly related machines

For this paradigm, **there is also an optimal preemptive algorithm**

In fact there are many of those

Horvath, Lam, Sethi 1977, Gonzales, Sahni 1978, Shachnai, Tamir, Woeginger, 2002, Ebenlendr, Sgall 2004, E. Tassa, 2006

But none of them is as simple as the one of McNaughton for identical machines. So we will only write an expression fo the resulting makespan

Assume for simplicity that jobs are sorted by size

$$p_1 \geq p_2 \geq \cdots \geq p_n$$

We assume $n \geq m$, as otherwise we can assume that the slowest m-n machines are not used by optimal solutions, and these machines are removed from the instance

Why? at most n machines can be active simultaneously, even in a preemptive schedule

Let the total size of the largest k jobs be $P_k = \sum_{\ell=1}^{k} p_\ell$

Assume for simplicity that machines are sorted by speed

$$s_1 \geq s_2 \geq \cdots \geq s_m$$

Let the total speed of the fastest k machines be

$$S_k = \sum_{\ell=1}^{k} s_\ell$$

It is known that

the makespan of an optimal preemptive solution is

$$\max\left\{\frac{P_1}{S_1}, \frac{P_2}{S_2}, \ldots, \frac{P_{m-1}}{S_{m-1}}, \frac{P_n}{S_m}\right\}$$

this covers in particular the case that there are k very large jobs

Uniformly related machines: lower bound

m jobs, each of size 2m-1

m-1 machines of speed 2, one slower machine of speed 1

What is the makespan of a non-preemptive schedule?

If a fast machine has two jobs or more, its completion time is at least $\frac{2(2m-1)}{2} = 2m-1$

Otherwise, the slow machine has at least one job by pigeonhole principle, **and its completion time is 2m-1**

So the makespan is at least 2m-1

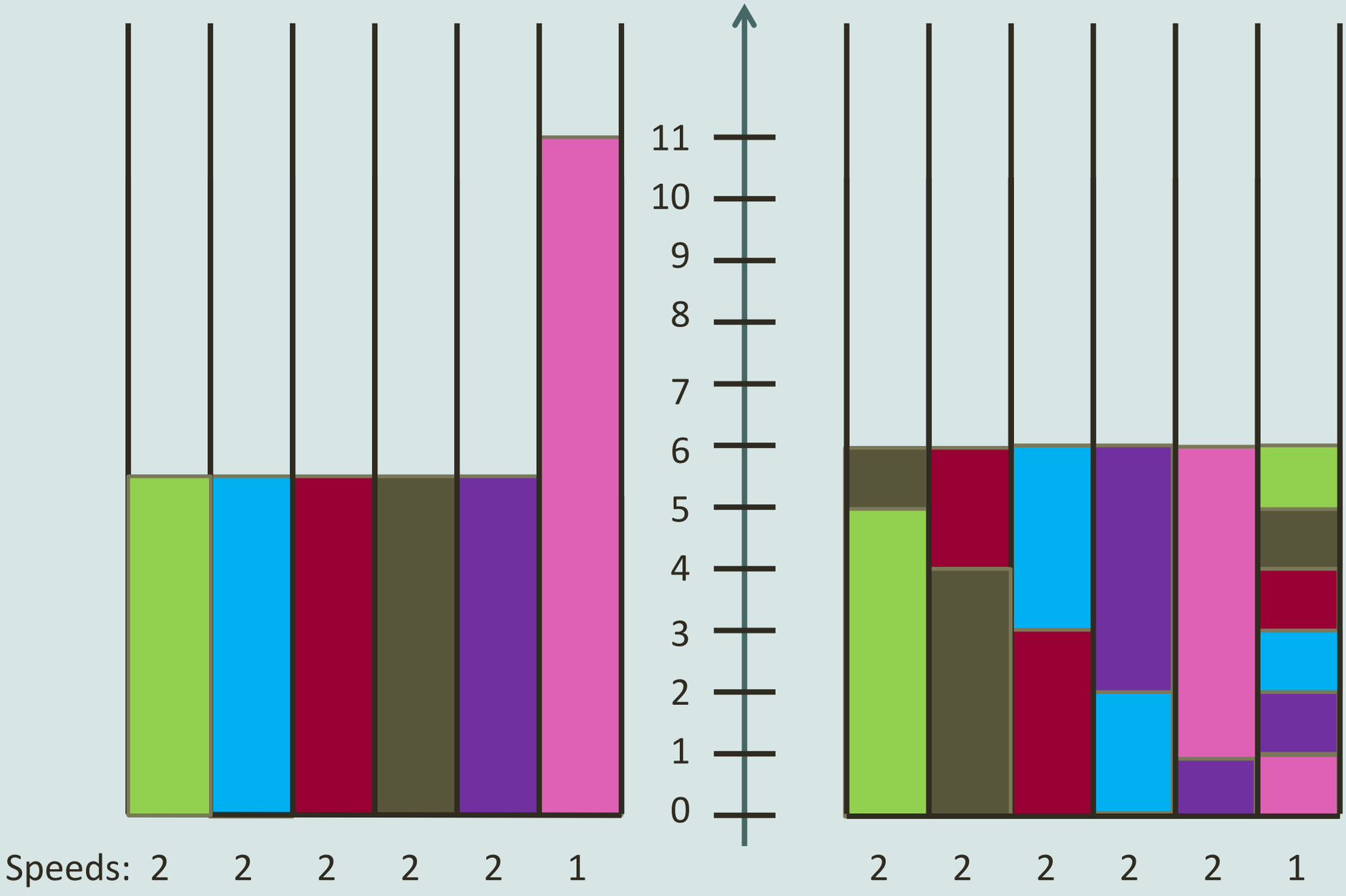
As for the preemptive schedule, we have:

$P_k = k(2m-1)$ $S_k = 2k$ **for** $k < m$ and $S_m = 2m-1$

Plugging into the formula we have $\max\left\{\frac{2m-1}{2}, \frac{m(2m-1)}{2m-1}\right\} = m$

So the benefit of preemption is at least $\frac{2m-1}{m}$

What do the schedules look like?



Uniformly related machines
Upper bound

Let $S = S_m = \sum_i s_i$

<u>One difference</u> in the analysis **compared to that of identical machines** is that while the makespan of any solution

preemptive or non-preemptive **is at least P/S**

there are **additional bounds** for optimal solutions in the formula

- **<u>Another difference</u> is that LPT now acts on machines with speeds**
  - The first m jobs **are not necessarily assigned to m different machines**
  - We will nevertheless assume that **there are at least m jobs**
    - **Otherwise at least one machine is redundant**

    We can use the result for a smaller value of m, which is smaller

    2-1/m is monotonically increasing in m
- The main reason for the different bound is that we can **only** assume that there are at least **m jobs and not m+1**
  - Indeed the lower bound example has m jobs, so here it is a harder case

- **We still have the property that adding p to every machine brings the completion time to L or more, due to the choice of machine, where the job will be completed first.**

L is the cost of LPT

  - But now we have $L = X_i + \frac{p}{s_i}$ **for machine i that has p**
  - For any machine k we have $X_k + \frac{p}{s_k} \geq L$,

    since p is assigned to i
  - Due to the speeds we have $\sum_k X_k \cdot s_k = P - p$
  - By $X_k + \frac{p}{s_k} \geq L$ we have $X_k \cdot s_k + p \geq L \cdot s_k$

    and equivalently $X_k \cdot s_k \geq L \cdot s_k - p$
  - By taking the sum over all machines

$$P - p = \sum_k X_k \cdot s_k \geq L \cdot (\sum_k s_k) - m \cdot p = L \cdot S - m \cdot p$$

By $P - p = L \cdot S - m \cdot p$

we get $L \leq \frac{(m-1)p+P}{S}$

L is the cost of LPT

By $P \geq m \cdot p$ we get

$$L \leq \frac{(m-1)p+P}{S} \leq \frac{\frac{(m-1)P}{m}+P}{S} = \frac{(2m-1)P}{mS}$$

Since the makespan of any preemptive schedule is at least P/S the benefit of preemption is at most

$$\frac{2m-1}{m}$$

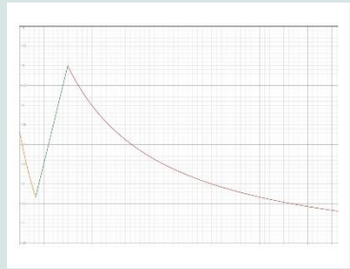
More on makespan

The special case of two uniformly related machines was studied:

Jiang, Weng, Hu, 2014, Soper and Strusevich, 2014

We found that the tight bound is 1.5 for this case

In these papers, the benefit of preemption was studied as a function of the speed ratio $s \geq 1$ between the two machines



It is 1.5 only for the case of s=2

We saw that is is 4/3 for s=1

They found the following tight result:

$$\begin{cases} \frac{2s+2}{3s} & \text{for} \quad 1 \leq s \leq \frac{4}{3} \\ \frac{s+1}{2} & \text{for} \quad \frac{4}{3} \leq s \leq 2 \\ \frac{s+1}{s} & \text{for} \quad s \geq 2 \end{cases}$$

The ratio tends to 1 for large s, because then one machine is so much slower than the other that it is almost not used

Total (weighted) completion time

This variant is much more interesting that the previous one for a single machine, and it is also of interest for multiple machines

Proofs of upper bounds are harder

we will mainly consider lower bound examples, in order to see the flavor of this variant

For identical machines, the benefit of preemption is 1 for total weighted completion time

Rothkopf 1966 - quite surprising (both the result, and the early stage that it was proved).

We will show bounds for the benefit of preemption based on

- One machine, release dates **E., Levin 2016**
  - This work contains lower bounds
  - The overall tight upper bound for total weighted completion time follows from previous work
    - Chekuri, Motwani, Natarajan, Stein 2001, where its value is $\frac{e}{e-1} \approx$ **1.581**
- Uniformly related machines **E., Levin, Soper, Strusevich 2017**
  - In this work there is an analysis of a tight upper bound of value $\approx$ **1.39795**
    - For the unweighted case
    - For two machines the tight value is **1.2**

One machine, release dates

We saw that for preemption does not help for makespan minimization

Why does it help for total completion time?

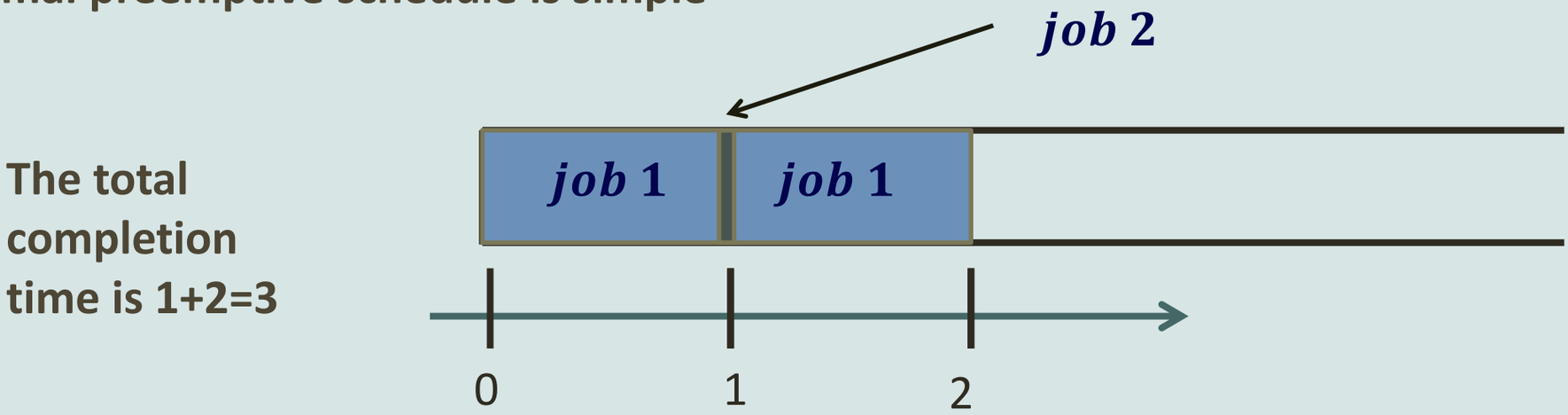
Consider an instance with two jobs

A large job $p_1 = 2,\ r_j = 0$

and a small job $p_2 = 0,\ r_2 = 1$

We allow jobs of size zero for simplicity, formally we can use very big jobs and jobs of size 1

An optimal preemptive schedule is simple



The total completion time is 1+2=3

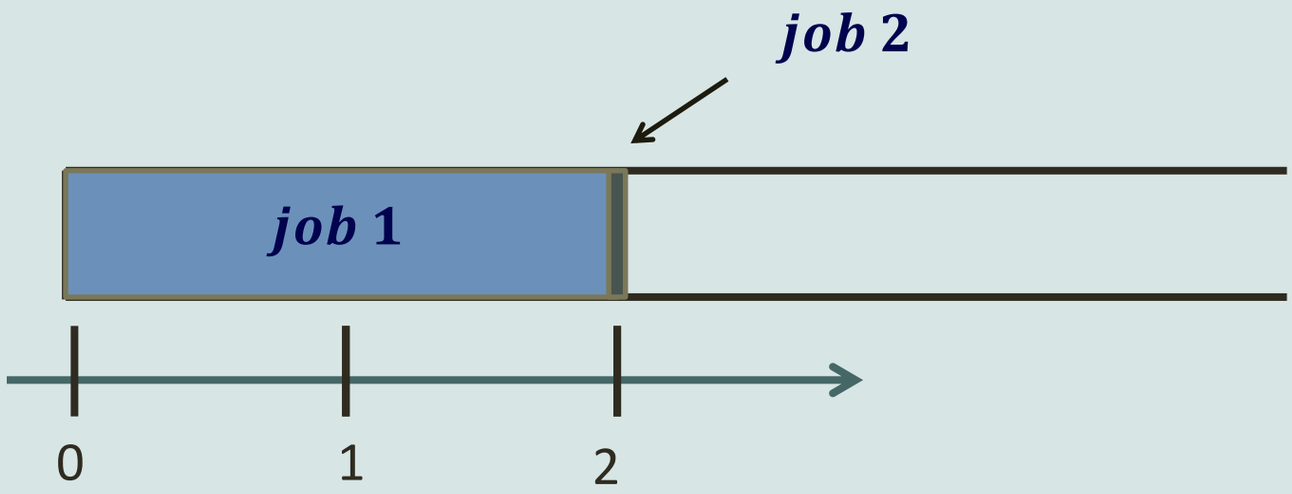
The schedule is optimal, as every job j is completed at the time $r_j + p_j$

The large job is stopped in order to process the small job

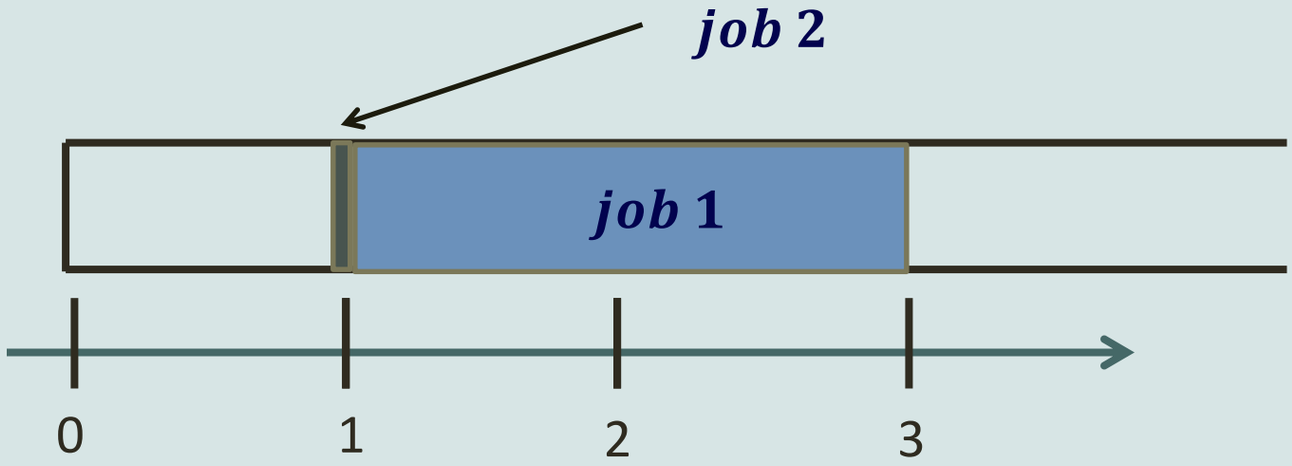
But this cannot be done in a non-preemptive schedule

There are two reasonable non-preemptive schedules

The total completion time is 2+2=4



The total completion time is 1+3=4



This gives a lower bound of 4/3 on the benefit of preemption for the objective of (unweighted) total completion time

It can be slightly improved (to 1.39) with additional jobs of size zero and another large job, but we will focus on the weighted case anyway

Total weighted completion time – lower bound

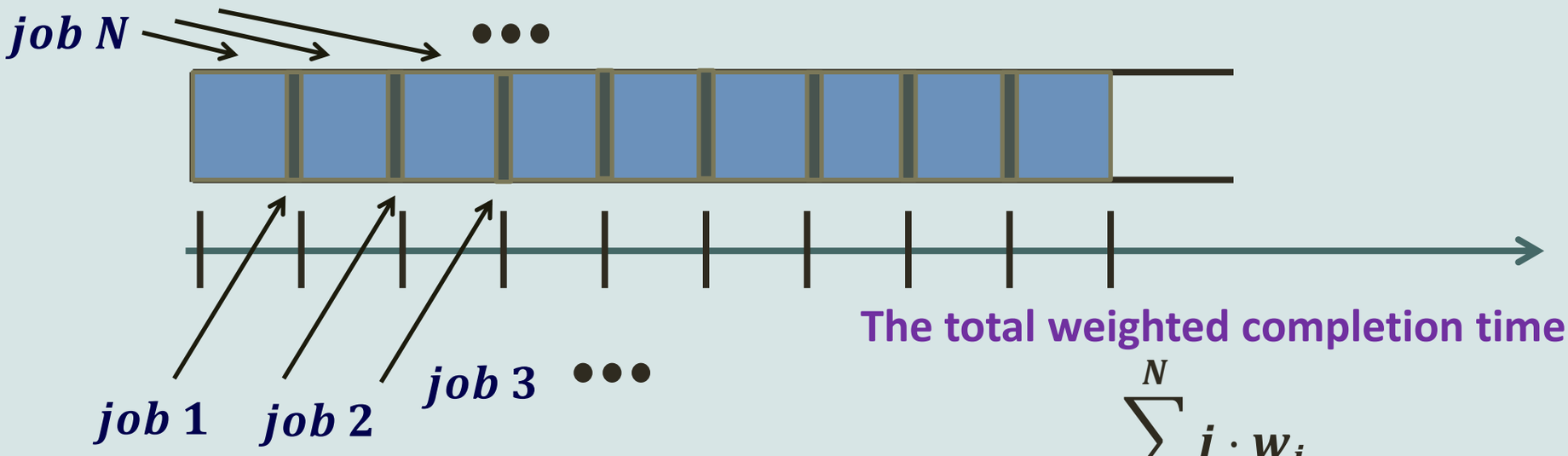
Now the large job has size N (and release date 0)

There are N-1 small jobs, each of size 0, and release dates:

1,2,3,…,N-1 **weights are specified later**

The calculations are easier if the large job is the job of index N

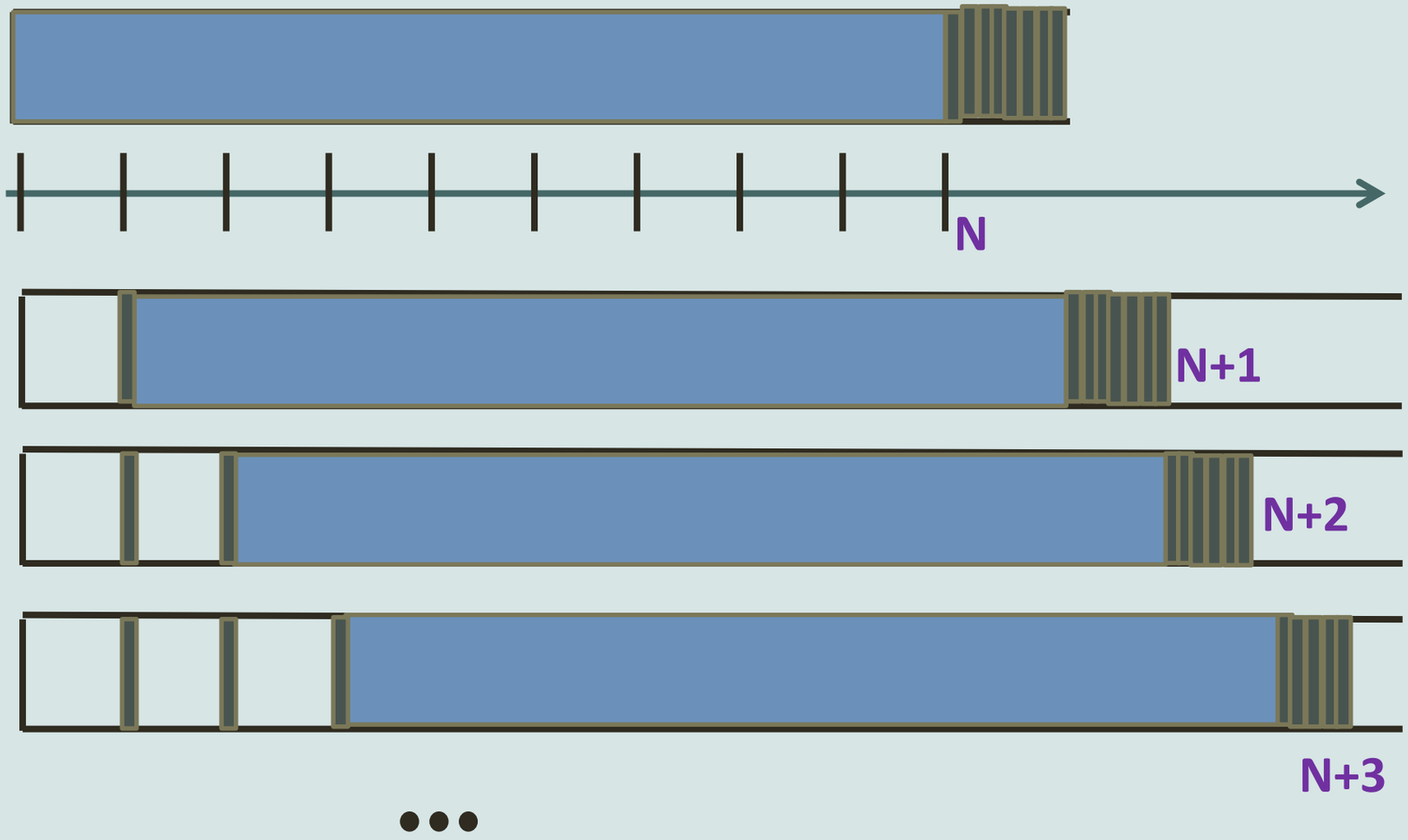
Similarly to the small example, an optimal preemptive schedule completes job j at time $r_j + p_j = j$



The total weighted completion time is

$$\sum_{j=1}^{N} j \cdot w_j$$

Reasonable non-preemptive schedules: N schedules in total



Possible starting times for the large job are 0,1,2,...,N-1

The cost for the first schedule is

$$N(w_1 + w_2 + \cdots + w_N)$$

The cost for the second schedule is

$$w_1 + (N+1)(w_2 + w_3 + \cdots + w_N)$$

The cost for the third schedule is

$$w_1 + 2w_2 + (N+2)(w_3 + w_4 + \cdots + w_N)$$

•••

The cost for the i-*th* schedule is

$$\sum_{j=1}^{i-1} j \cdot w_j + (N + i - 1)(w_i + w_{i+1} + \cdots + w_N)$$

•••

The cost for the N-*th* schedule is

$$\sum_{j=1}^{N-1} j \cdot w_j + (2N - 1) \cdot w_N$$

We would like to assign weights such that all N costs are equal

The resulting costs are:

$$\boldsymbol{w_i = a^i} \text{ for } i = 1,2,\dots,N-1 \text{ and } \boldsymbol{w_N = N \cdot a^N} \text{ for } \boldsymbol{a = 1 - \frac{1}{N}}$$

This gives a ratio that tends to $\frac{e}{e-1} \approx 1.581$

for N growing to infinity

The origin of the e is due to the term $\left(1 - \frac{1}{N}\right)^N$

The upper bound follows from an algorithm that tries to imitate an optimal preemptive algorithm
It gives priority to jobs for which an optimal preemptive algorithm completed a certain fraction

But both the preemptive problem and the non-preemptive problem are NP-hard

Without weights, the non-preemptive problem is polynomial:

Algorithm SRPT (shortest/smallest remaining processing time)

is optimal – Scharge 1968

At each time it runs the job with smallest remaining residue – it may swap the job if another job is released (or if a job is completed).

Uniformly related machines total completion time

For a single machine, without release dates preemption does not help. We considered the variant with release dates.

For a single machine we can define **SPT (shortest processing time):**

Sort the jobs by **non-decreasing processing times** and schedule them in this order **Lawler, 1973 proved a generalized version**

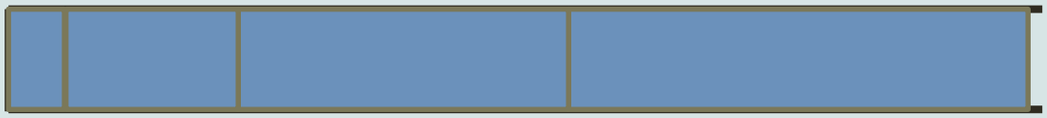
For multiple uniformly related machines, release dates would complicate the problem, and it is non-trivial even without release dates (and it has a clean structure).

The advantage here: **Both variants compared here are polynomial**

The algorithms are attributed to multiple articles

see the books by Pinedo and by Brucker

The **non-preemptive** version: "**The multipliers algorithm**"



Optimality of SPT for a set of jobs on one machine:

-- Obviously idle time increases the objective

so every reasonable schedule is just a **permutation** scheduled **consecutively starting from time zero**

-- If the schedule is not sorted (an exchange argument)

apply “sorting”: swap pairs of consecutive jobs and as a result:

one completion time does not change - the first job gets the completion time of the second one - and the other one decreases – because the job is smaller

Thus, in a non-preemptive schedule, once the set of jobs of a machine is selected, it is scheduled by SPT on that machine

The multipliers algorithm

Every machine is scheduled using the SPT rule

But how do we partition jobs among machines?

For jobs of sizes $a_1, a_2, \dots, a_k$ assigned to a machine of speed s

to run in this order

If the completion time of the *i*-th job is T_i

The cost for this machine **is** $(T_1 + T_2 + \cdots + T_k)/s$

Since $T_i = a_1 + a_2 + \cdots + a_i$ the *i*-th job waits for all jobs that run before it to be completed, and we get a cost of:

$$\sum_{i=1}^{k} \frac{(k-i+1)a_i}{s}$$

The last job to run on this machine

We call $\frac{k-i+1}{s}$ a multiplier

For this machine the multipliers are (in reverse order) 1/s, 2/s, 3/s,...

Out of the possible multipliers, the smallest n are chosen greedily and assigned in reverse order to the sorted list of jobs

The largest job gets the smallest multiplier as in SPT

in which the multipliers in reverse order are 1,2,...,n

n is the multiplier of the smallest job – that will run first

assigned to run starting from time 0 **but assigned from top to bottom**

Example: for machines with speeds 7,5,3,2

The four lists of multipliers are

1/7,2/7,3/7,4/7,5/7,6/7,7/7,8/7,....

1/5,2/5,3/5,4/5,5/5, 6/5,....

1/3,2/3,3/3,4/3,....

1/2,2/2,3/2,....

Breaking ties arbitrarily or using a fixed rule

smallest/largest machine index for machines of sorted speeds

The sorted list of multipliers is:

1/7~**0.142857**, 1/5=**0.2**, 2/7~**0.2857**, 1/3~**0.33333**, 2/5=**0.4**, 3/7~**0.42857**, ½=**0.5**,
4/7~**0.57**, 3/5=**0.6**, 2/3~**0.6667**, 5/7~**0.71**, 4/5=**0.8**, 6/7~**0.857**, 2/2=**1**, 3/3=**1**,...

The **preemptive** version: "The staircase algorithm"

The idea:

At each time, run the smallest m jobs on the machines

if there are less jobs, use the fastest machines

Sorted such that the smallest job runs on the fastest machine etc.

Why? we would like to complete small jobs as fast as we can

This means that we start the smallest job on the fastest machine, and the second smallest on the machine of next speed

When the smallest job is completed, the job of next priority is moved there (or started there)

Isn't this the same? example for both algorithms

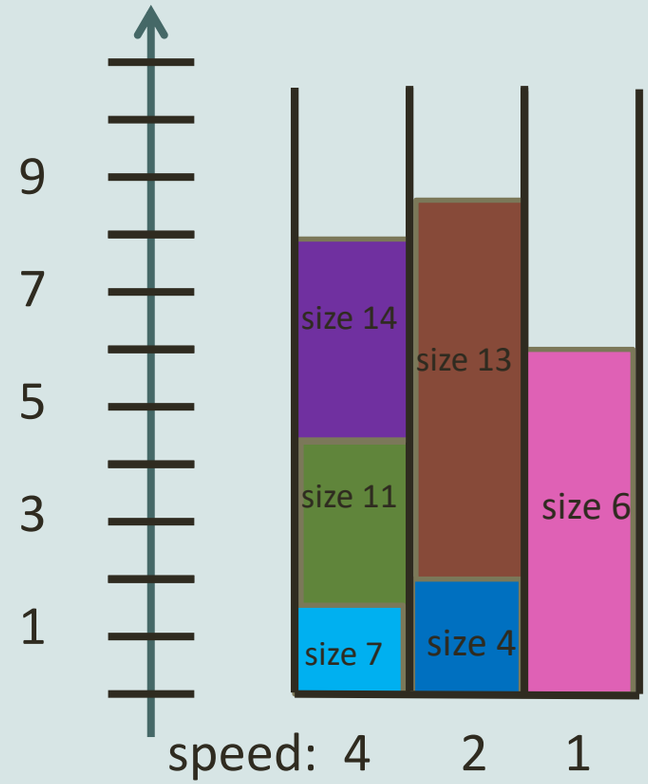
m=3 machines of speeds 4,2,1

Jobs of sizes: **4,6,7,11,13,14**

Non-preemptive: use the multipliers $\frac{1}{4}, \frac{1}{2}, \frac{2}{4}, \frac{3}{4}, \frac{1}{1}, \frac{2}{2}$

¼ for 14, ½ for 13,...

The schedule:

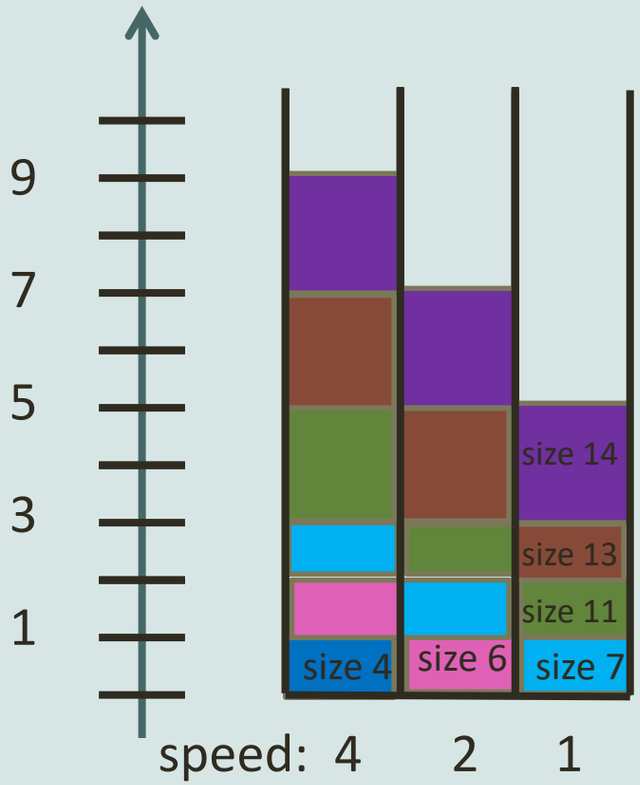


Total completion time: 30.75

Preemptive: The staircase algorithm

m=3 machines of speeds 4,2,1 Jobs of sizes: 4,6,7,11,13,14

The schedule:



Total completion time: 27

The cost of a non-preemptive schedule that assigns all jobs to the fast machine and schedules them in SPT order is 38.75

We can see that the difference between the two optimal schedules is not very large but there is some difference

Similar motivation – everyone waits for the first job, so it should be the smallest one

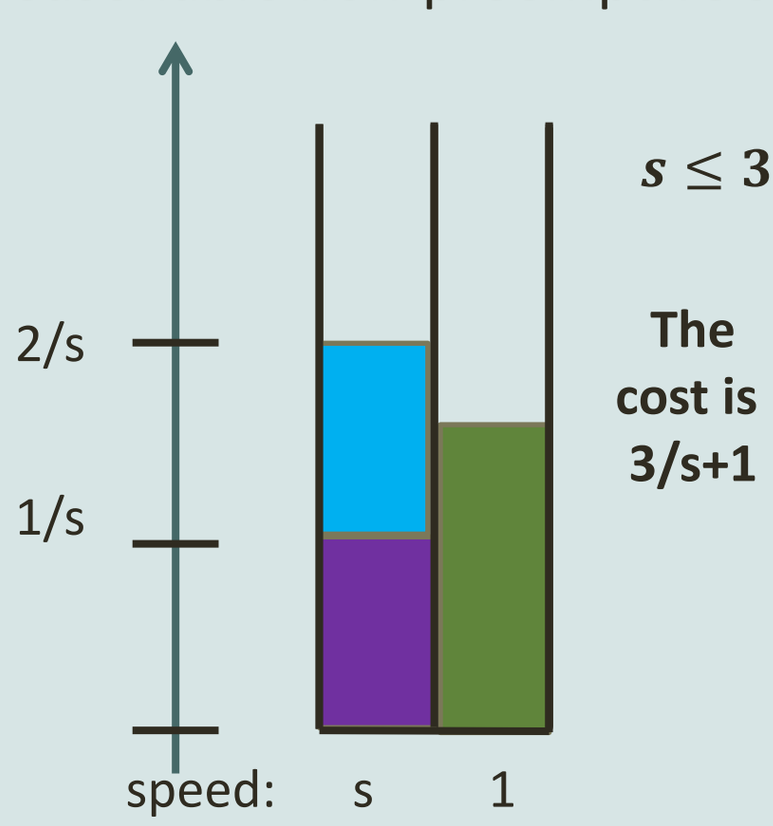
Uniformly related machines, total completion time, lower bounds

The previous example implies a lower bound of 1.1388..

Consider two machines of speeds 1 and s>1

and three identical jobs, each of size 1

The reasonable non-preemptive schedules are as follows:

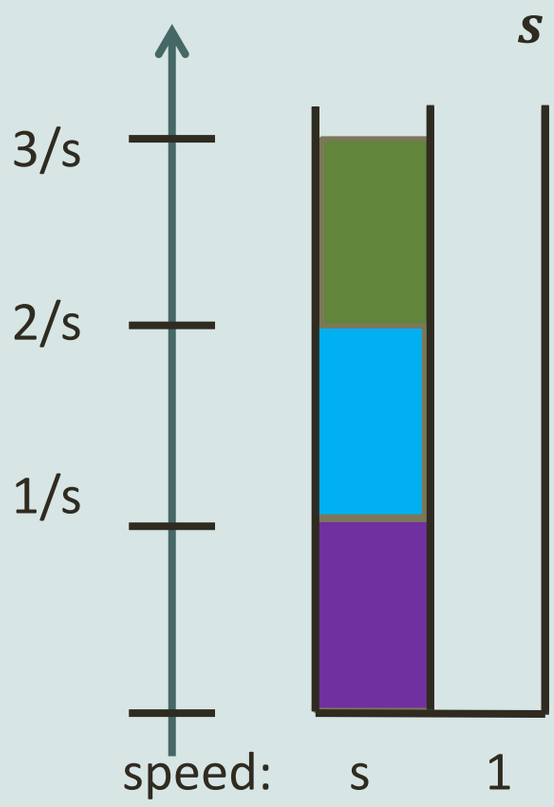


$s \leq 3$

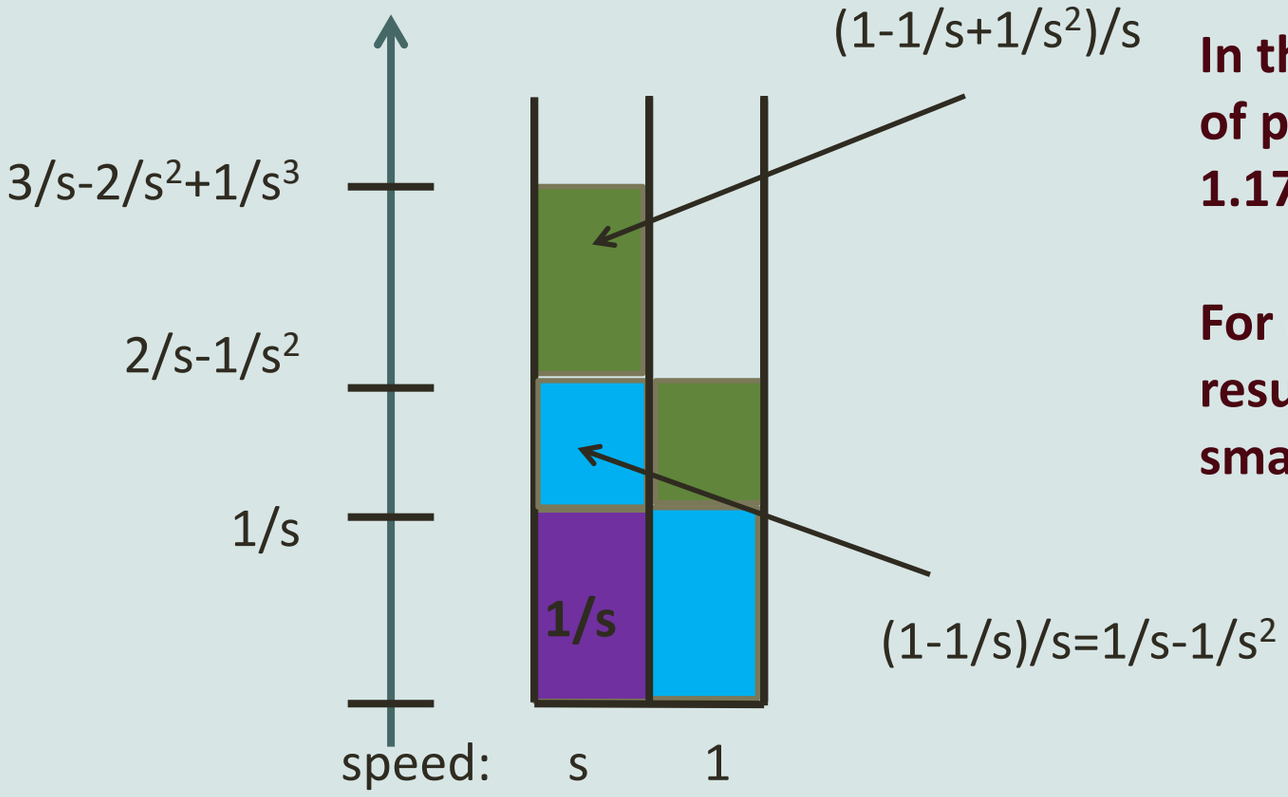
The cost is 3/s+1

The cost is 6/s

$s \geq 3$



The preemptive schedule is as follows:



The cost is $6/s-3/s^2+1/s^3$

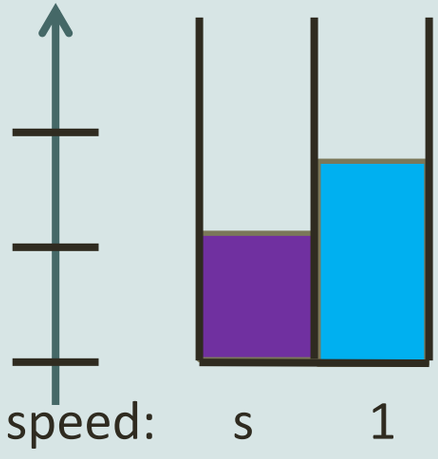
In the case s=3 the benefit of preemption is at least 1.1739

For other values of s the resulting lower bound is smaller

If there are only two jobs the preemptive schedule is as above without the green job and has cost **$3/s-1/s^2$**

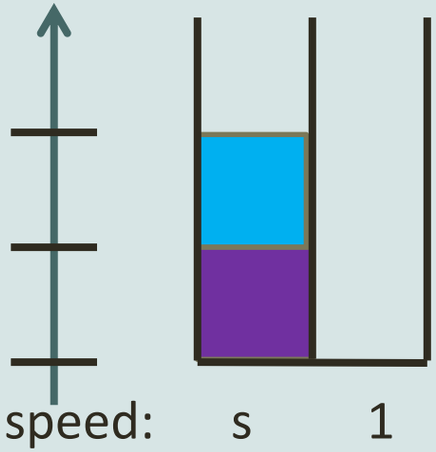
$s \leq 2$

The cost is 1/s+1



Reasonable non-preemptive schedules for two jobs:

The cost is 3/s



$s \geq 2$

For two jobs we find a lower bound of

$$\frac{s^2+s}{3s-1} \quad \text{for} \quad s \leq 2 \quad \text{and} \quad \frac{3s}{3s-1} \quad \text{for } s \geq 2$$

The maximum occurs as s=2 and it is 1.2

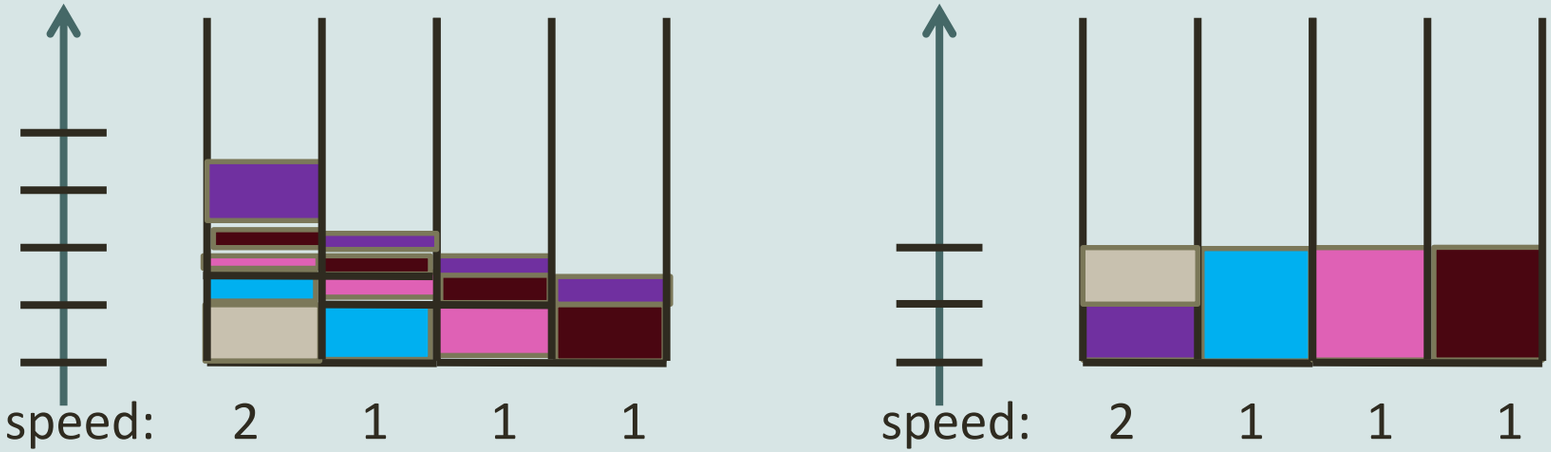
Is this the precise benefit of preemption for two machines?

Yes, we can prove that there is a worst-case input with unit size jobs, for any number of machines and speed combination

We can analyze different numbers of jobs and show that for two machines the worst case is for two jobs

We can also prove that in a worst-case input, there is one fast machine of speed s>1 and all other machines have speed 1

Example: s=2, m=4, n=5 jobs, each of size 1



Goal: find the suitable values of s and n given m
We can assume that m is large

Solving this gives the value of benefit of preemption:
1.39795

There are worst-case instances with n=m and s<m, where s/m~ 0.7959

A generalized model

Unrelated machines

This is the multiprocessor model where every machine i and a job j have a processing time for “j running on i” associated with this pair

A preemptive schedule can be defined, for every machine the size of a job assigned to it defines a fraction, and the sum of fractions has to be 1

Makespan

It is known that the benefit of preemption is exactly 4

Correa, Skutella, Verschae 2009

Total weighted completion time

The benefit of preemption is strictly below 2

An upper bound of 1.81 is known: Sitters, 2008

This is the best result known for uniformly related machines, the weighted case. Open problem: find the tight bound for that!

Other variants

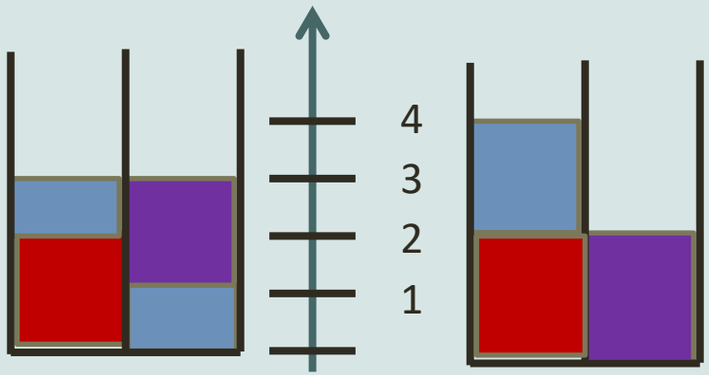
The benefit of preemption is of interest for many scheduling problems - interesting when it is not 1 but still it is a constant

For one machine, the status of total (unweighted) completion time is also unknown, and it is in approximately **[1.39167, 1.5819767]**

There are variants where the benefit of preemption is infinite

If there are deadlines, the tardiness of a job j is defined as

$\max\{0, C_j - d_j\}$ **where** C_j **is its completion time**



In this example with identical machines which we saw, if all deadlines are equal to 3, the cost of the preemptive schedule is 0 and the cost of the non-preemptive one is 4

Similar/related problems

One can compare schedules with different numbers of preemptions

For example

- One preemption versus an unlimited number of preemptions
- Ten preemptions versus 17 preemptions..

One can find how many preemptions are needed to achieve the best possible preemptive schedule (worst-case)

From previous work:

- m-1 for makespan and identical machines
- 2m-2 for makespan and uniformly related machines

Thank you!