Scheduling

VU VRIJE UNIVERSITEIT AMSTERDAM
LOOKING FURTHER

Theory for this week:

All theory is on the slides and in the exercises of this week.

For more theory/examples, see the book by Pinedo:

Scheduling:Theory, Algorithms, and Systems (Link on Canvas)

(Only as an extra. Not required for the exam.)

All these slides are exam material.

You should be able to

- understand + know the different scheduling problems,

- understand + know the algorithms,

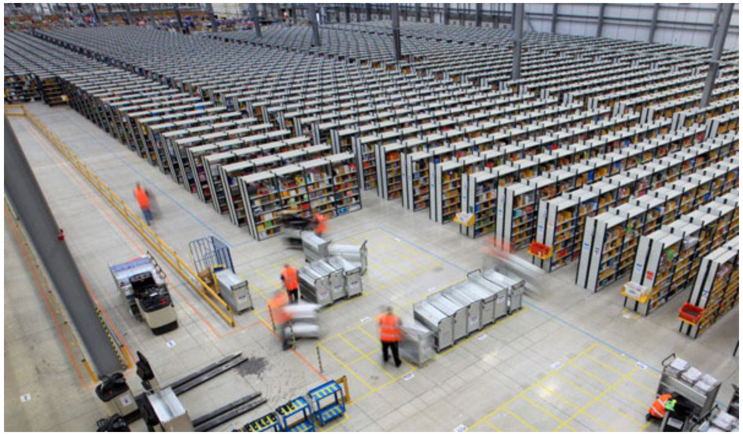
- understand + know the proofs

- apply the theory to other scheduling problems.

(If some of this theory will be removed from the exam material, then this will be clearly communicated via Canvas.)

Scheduling applications

– Logistics

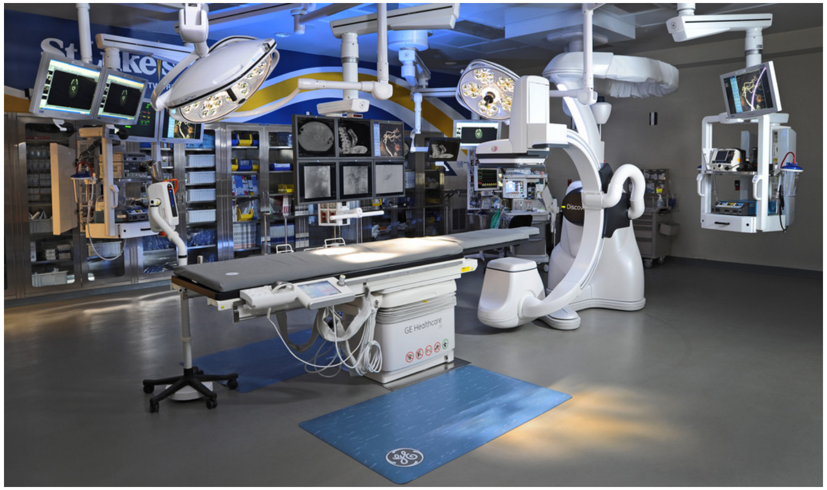


Scheduling applications

– Personnel planning

Scheduling applications

– Healthcare



Standard scheduling notation

What is scheduling?

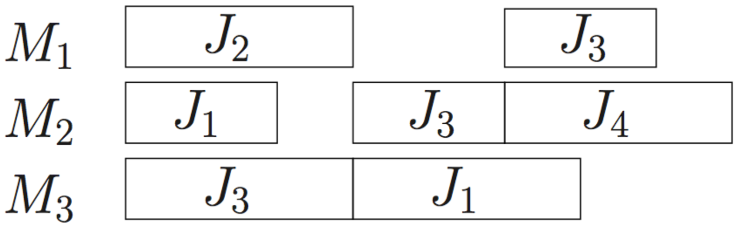
Scheduling concerns optimal allocation or assignment of resources, over time, to a set of tasks/activities/jobs.

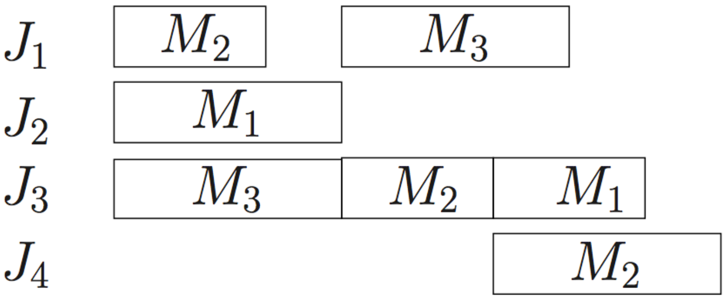
Resourses (M_i) : machines, people, space

Tasks (J_j) : production, jobs, classes, flights

Henry Gantt (VS)
1861–-1919

Schedules may be represented by Gantt charts





Standard scheduling notation

Machines:

m : machines $i=1,\ldots,m$

n : jobs $j=1,\ldots,n$

Jobs:

p_j : processing time of job j

r_j : release date of job j (earliest starting time)

d_j : due date (deadline) (committed completion time)

w_j : weight of job j (importance)

Schedule

C_j: completion time of a job

Classification of Scheduling Problems

Many scheduling problems can be described by a three field notation $\alpha|\beta|\gamma$, where

α describes the machine environment
β describes the job characteristics, and
γ describes the objective criterion to be minimized (or max.)

Remark: A field may contain more than one entry but may also be empty

Example $\mathbf{1 \mid r_j \mid \Sigma_j C_j}$

- Single machine.
- Jobs have release times.
- Objective is minimizing the sum of the completion times.

Machine environment (α)

Single machine ($\alpha = 1$)

Identical parallel machines (α = P or Pm)

m identical machines running in parrallel

p_j is the process time of job j

Uniform parallel machines (α = Q or Qm)

m identical machines running at different speed

s_i is speed of machine i

$p_{ij} = p_j/s_i$ is the process time of job j if scheduled on machine i

Unrelated parallel machines (α = R or Rm)

m different machines in parallel

p_{ij} is the process time of job j if scheduled on machine i

Job characteristics (β)

Release dates (r_j)

- job j may not start before its release time r_j

Deadlines (d_j)

- job j should finish before its deadline d_j

Preemption (pmtn)

- processing of a job on a machine may be interrupted and resumed at any machine.

Unit processing times ($p_j = 1$)

- each job (operation) has unit processing times

Precedence constraints (prec)

- job cannot start before some other jobs are finished
- presented by an acyclic graph

Objective function (γ)

Makespan C_{max}

- Minimizing the last completion time: $C_{max} = max_j\ C_j$

Maximum lateness L_{max}

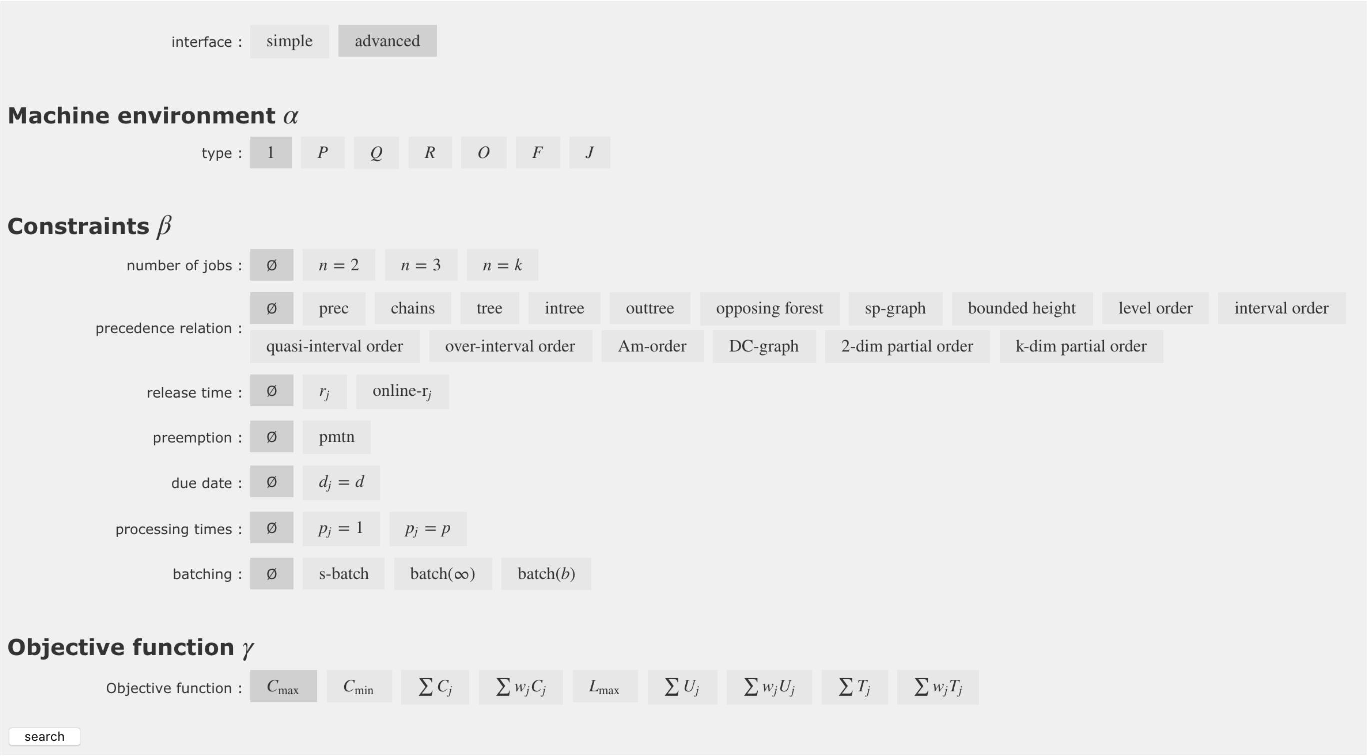
- Lateness of job j: $L_j = C_j - d_j$
- $L_{max} = max_j\ L_j$

Total completion time $\Sigma_j\ C_j$

Total weighted completion time $\Sigma_j\ w_jC_j$

Many more models in literarture !

Scheduling zoo



Notation itself creates a world of scheduling problems.

many, of course, are not that relevant.

- http://schedulingzoo.lip6.fr/
- http://www2.informatik.uni-osnabrueck.de/knust/class/

Let's look at some easy scheduling problems .....

Single machine problems $1|\beta|\gamma$

1. $1 \,||\, \Sigma C_j$
2. $1 \,||\, \Sigma w_j C_j$
3. $1 \,|\, r_j, \mathit{pmtn} \,|\, \Sigma C_j$
4. $1 \,||\, L_{max}$
5. $1 \,|\, r_j \,|\, \Sigma C_j$
6. $1 \,|\, r_j, prec \,|\, \Sigma C_j$

Parallel machine problems

7. $P \mid pmtn \mid C_{max}$

8. $P \mid\mid C_{max}$

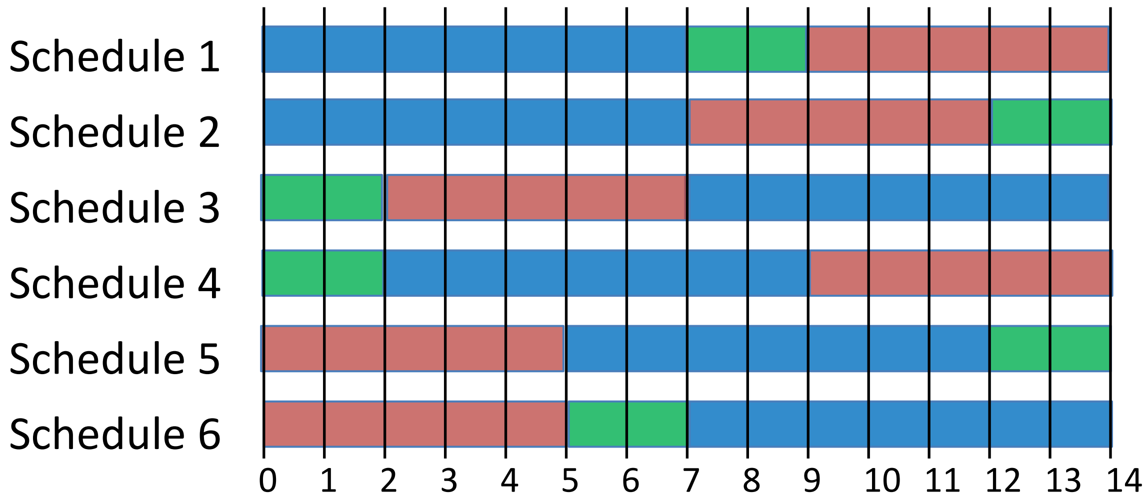
9. $R \mid\mid \Sigma C_j$

10. $R \mid\mid C_{max}$

$1 \mid . \mid \Sigma_j C_j$

- single machine
- minimizing total completion time $\Sigma_j C_j$

jobs	1	2	3
length p_j	7	2	5



$\Sigma_j C_j$

7+9+14 = 30

7+12+14 = 33

2+7+14 = 23

2+9+14 = 25

5+12+14 = 31

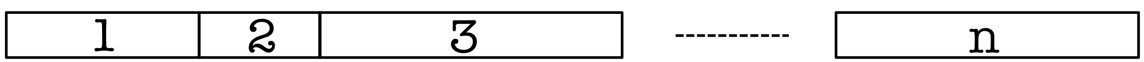
5+7+14 = 26

$1 \mid . \mid \Sigma_j C_j$

- single machine
- minimizing total completion time

Theorem Shortest Processing Time (SPT) rule is optimal.

Proof (sketch)



$$C_1 = p_1$$
$$C_2 = p_1 + p_2$$
$$C_3 = p_1 + p_2 + p_3$$
$$\ldots$$
$$C_n = p_1 + p_2 + p_3 + \ldots + p_n$$

$$\boldsymbol{\sum_j C_j = n p_1 + (n-1) p_2 + (n-2) p_3 + \ldots + 1 p_n}$$

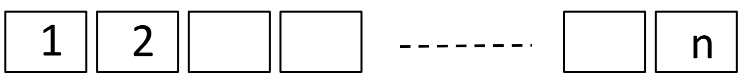
$1 \mid p_j = 1 \mid \Sigma_j w_j C_j$

- single machine
- unit length jobs
- minimizing total weighted completion time

Theorem Decreasing order of weights is optimal.

Proof (sketch)

Label jobs in schedule $1, 2, 3, \ldots$.



Then, $\sum_j w_j C_j = 1w_1 + 2w_2 + \quad \ldots \quad + nw_n$

$1 \mid . \mid \Sigma_j w_j C_j$

- single machine
- minimizing total weighted completion time

jobs	1	2	3
weight w_j	10	5	2
length p_j	7	2	6

5/2 > 10/7 > 2/6

Optimal ordering : 2, 1, 3

How to order? By weight? By length?

Smith's ratio rule :

Schedule jobs in decreasing order of w_j / p_j

Theorem

Smith's ratio rule is optimal.

$1 \mid . \mid \Sigma_j w_j C_j$

Theorem

Smith's ratio rule is optimal.

Proof

Assume not in Smith's order: $w_1/p_1 < w_2/p_2$

Swap the jobs:

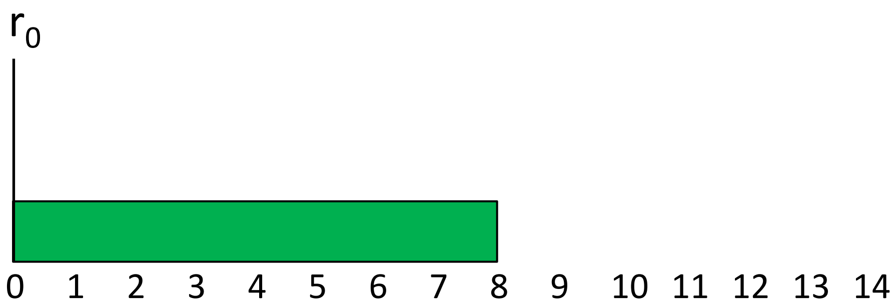
Then, the increase in total weighted completion time is:

$$w_1 p_2 - w_2 p_1 < 0$$

→ A schedule is optimal if and only if jobs are in Smith's order.

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

jobs	1	2	3	4
release time r_j	0	3	4	7
length p_j	8	4	1	2

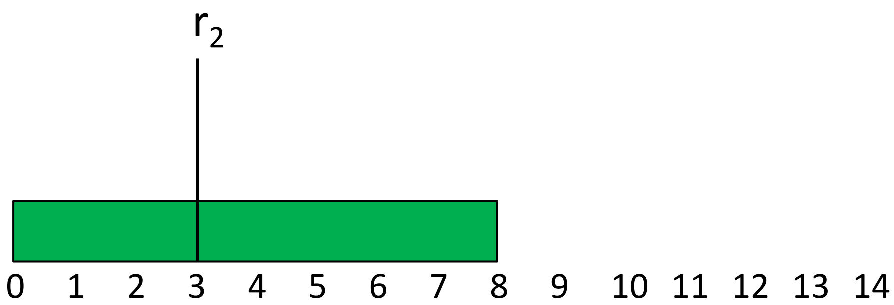


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

jobs	1	2	3	4
release time r_j	**0**	**3**	**4**	**7**
length p_j	**8**	**4**	**1**	**2**

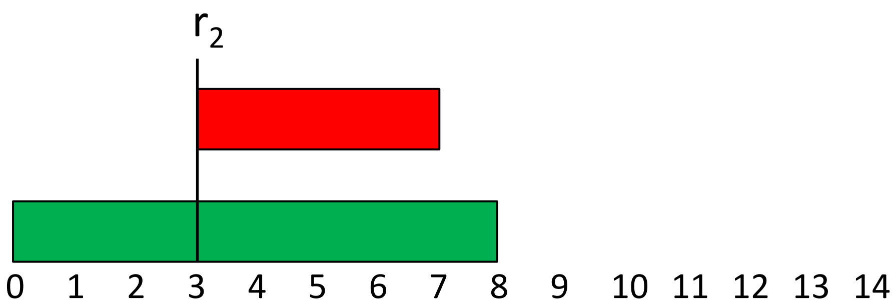


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

jobs	1	2	3	4
release time r_j	**0**	**3**	**4**	**7**
length p_j	**8**	**4**	**1**	**2**

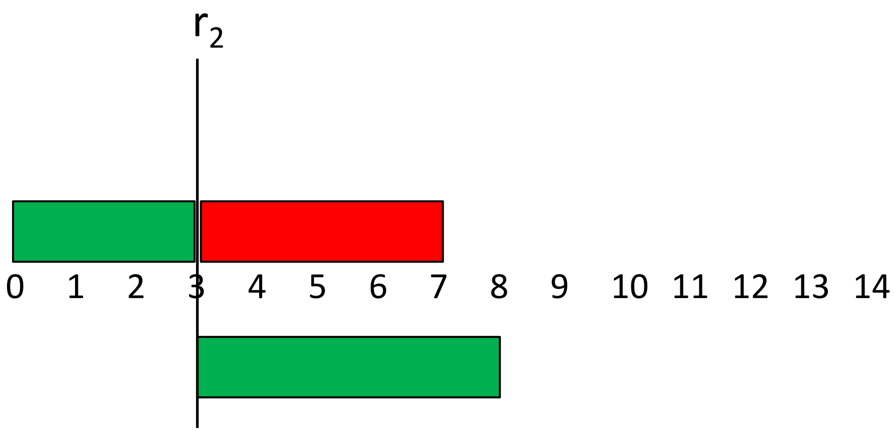


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

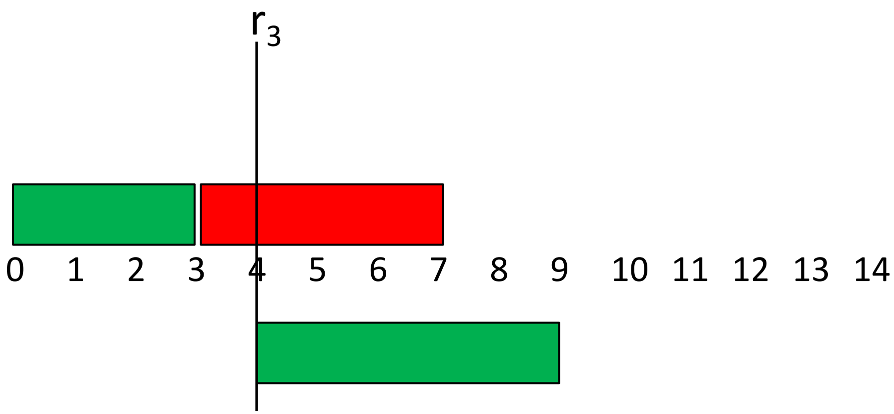
jobs	1	2	3	4
release time r_j	**0**	**3**	**4**	**7**
length p_j	**8**	**4**	**1**	**2**



SRPT :
At any moment, process the job with the Smallest Remaining Processing Time

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

jobs	1	2	3	4
release time r_j	**0**	**3**	**4**	**7**
length p_j	**8**	**4**	**1**	**2**

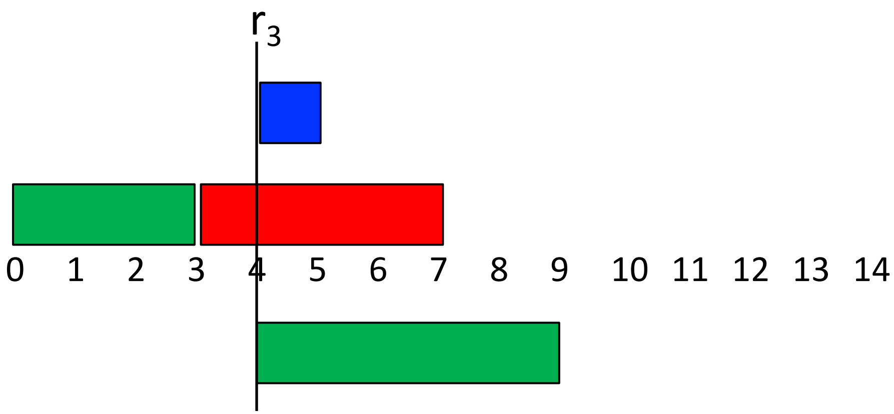


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

jobs	1	2	3	4
release time r_j	0	3	4	7
length p_j	8	4	1	2

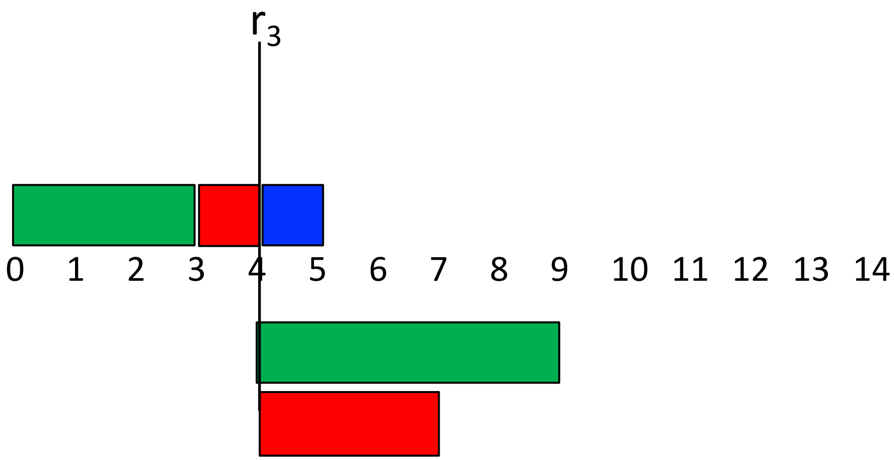


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1\ |r_j,pmtn|\ \Sigma_j\ C_j$

jobs	1	2	3	4
release time r_j	0	3	4	7
length p_j	8	4	1	2

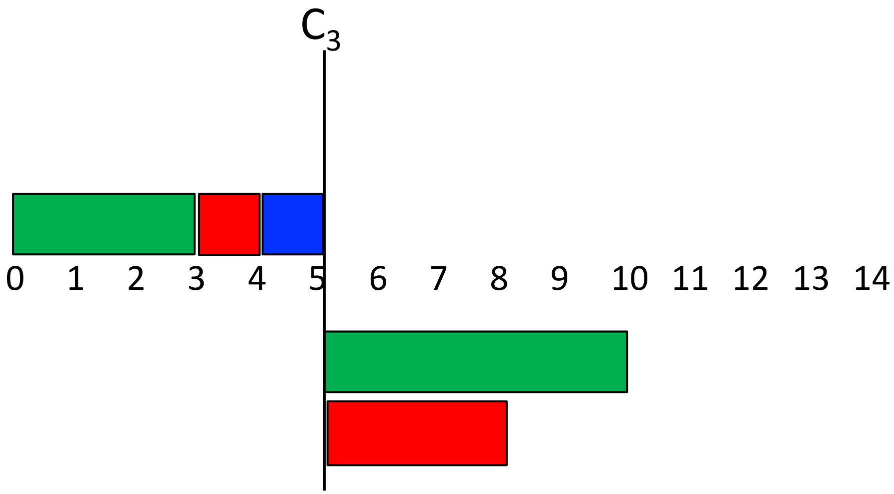


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1 \ | r_j, pmtn | \ \Sigma_j C_j$

jobs	1	2	3	4
release time r_j	0	3	4	7
length p_j	8	4	1	2

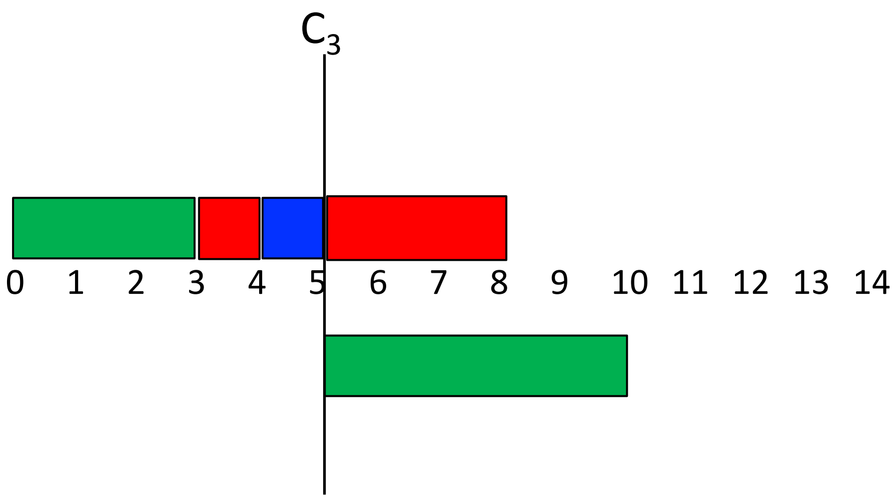


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

jobs	1	2	3	4
release time r_j	0	3	4	7
length p_j	8	4	1	2

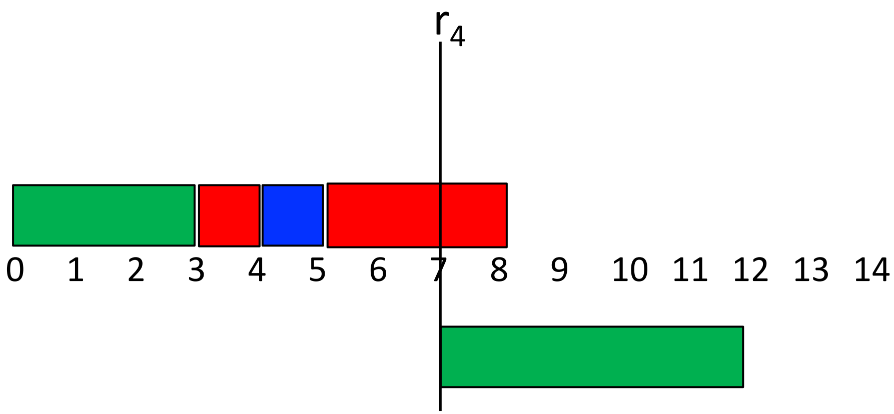


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

1 |r_j,pmtn| $\Sigma_j C_j$

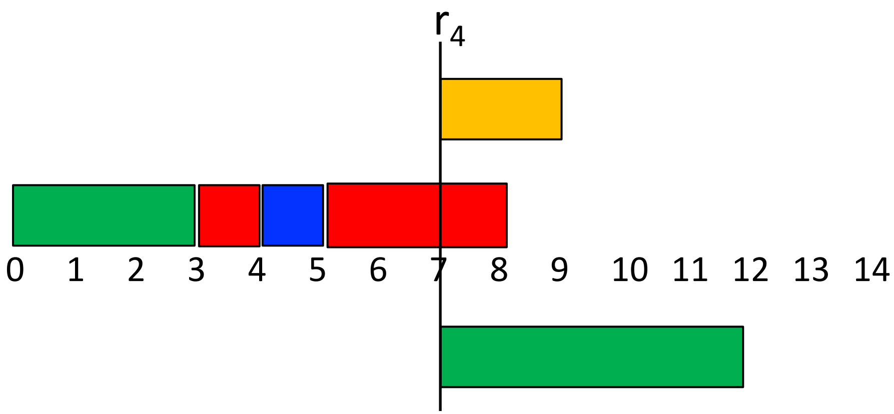
jobs	1	2	3	4
release time r_j	**0**	**3**	**4**	**7**
length p_j	**8**	**4**	**1**	**2**



SRPT :
At any moment, process the job with the Smallest Remaining Processing Time

1 | r_j,pmtn | $\Sigma_j C_j$

jobs	1	2	3	4
release time r_j	0	3	4	7
length p_j	8	4	1	2

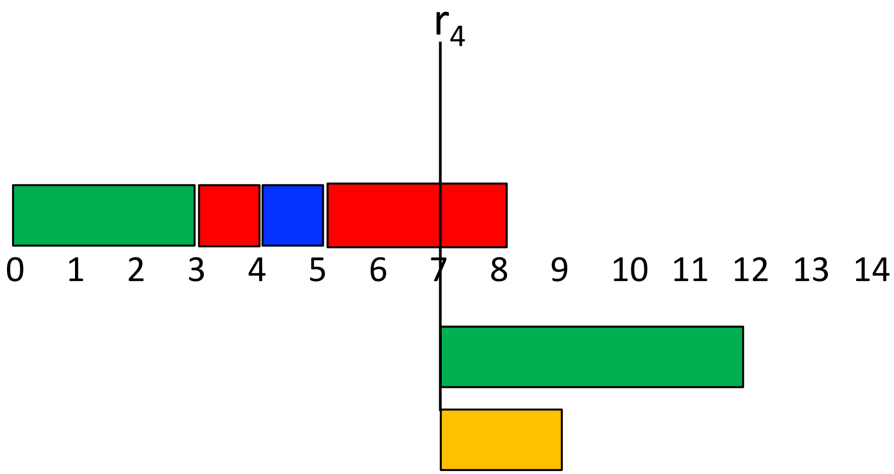


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

jobs	1	2	3	4
release time r_j	**0**	**3**	**4**	**7**
length p_j	**8**	**4**	**1**	**2**

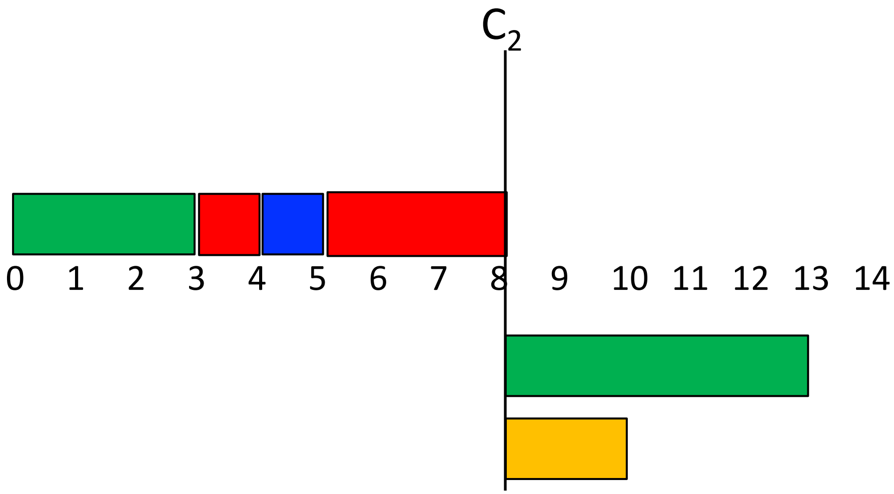


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

jobs	1	2	3	4
release time r_j	**0**	**3**	**4**	**7**
length p_j	**8**	**4**	**1**	**2**

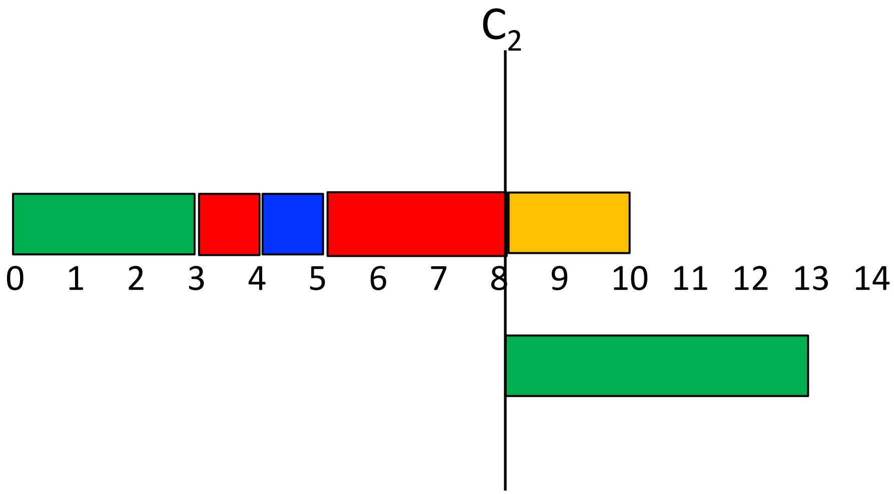


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

jobs	1	2	3	4
release time r_j	0	3	4	7
length p_j	8	4	1	2

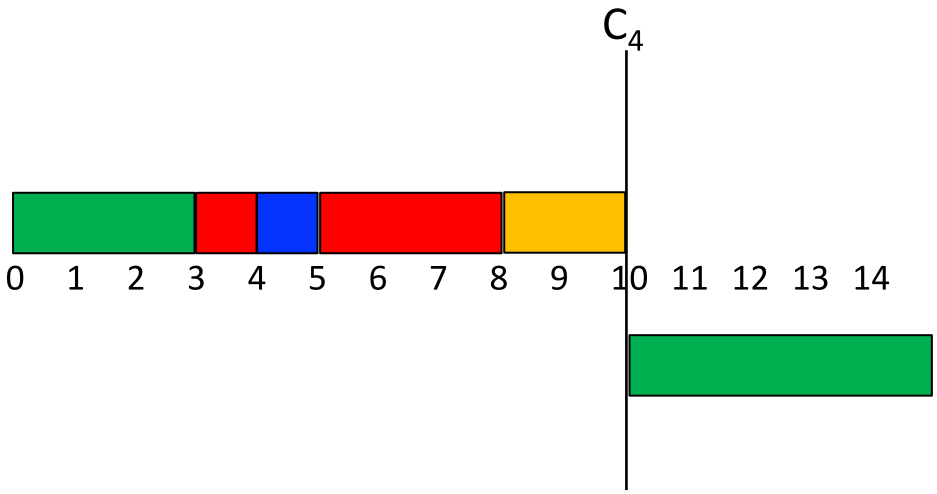


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1\ |r_j, pmtn|\ \Sigma_j\ C_j$

jobs	1	2	3	4
release time r_j	**0**	**3**	**4**	**7**
length p_j	**8**	**4**	**1**	**2**

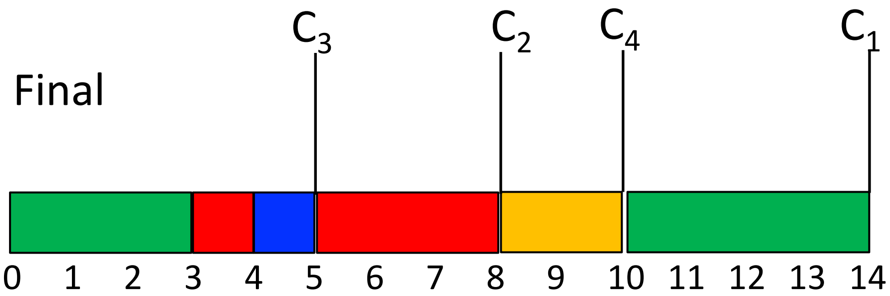


SRPT :

At any moment, process the job with the Smallest Remaining Processing Time

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

jobs	1	2	3	4
release time r_j	**0**	**3**	**4**	**7**
length p_j	**8**	**4**	**1**	**2**



SRPT :

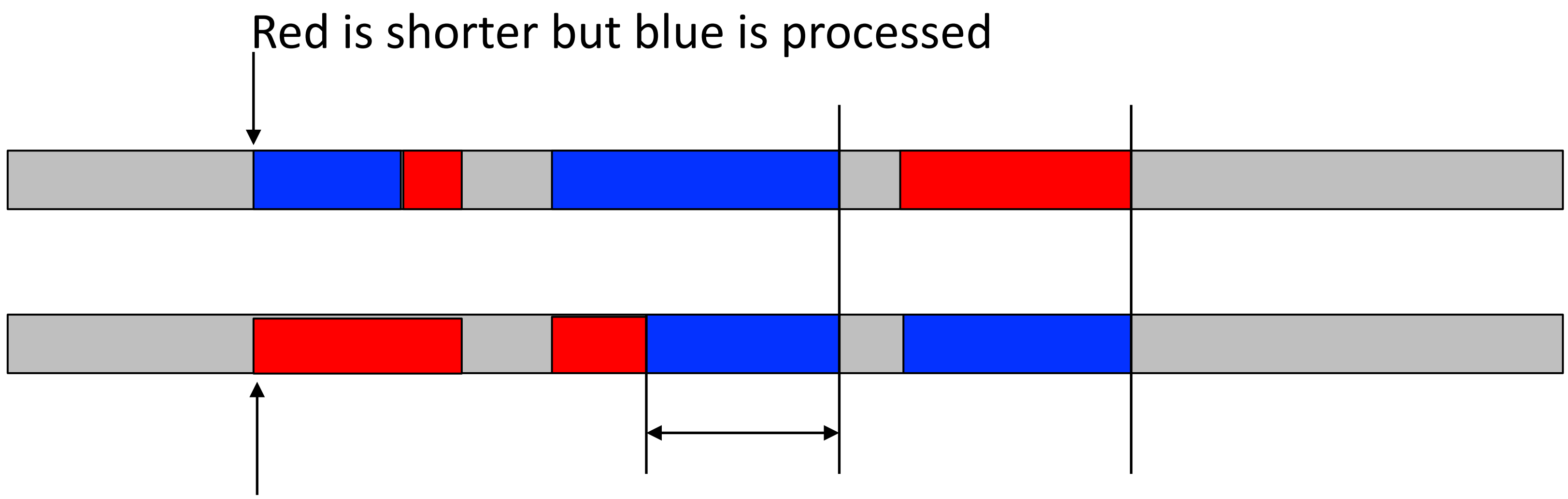
At any moment, process the job with the Smallest Remaining Processing Time

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

Theorem

SRPT is optimal

Proof Consider a schedule for which the SRPT rule does not hold.



Now reorder: First process red, then blue.

→ Total completion time decreases.

→ A schedule is optimal iff it is in SRPT order.

$1 \mid r_j, pmtn \mid \Sigma_j C_j$

So we know that SRPT is optimal for $1|r_j,\mathit{pmtn}|\sum C_j$

Exercise (for tutorial)

Show that SRPT is not optimal on parallel machines.

SPRT on m parallel machine:

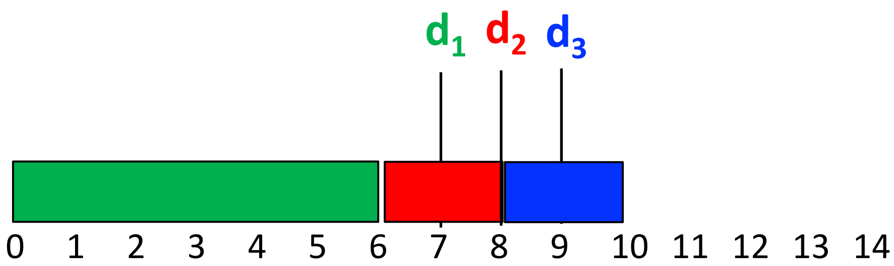
At any moment in time, process the m jobs with smallest remaining processing time (or all jobs if there are less than m jobs available at that time).

$1 \,|\, d_j \,|\, L_{max}$

jobs	1	2	3
due date d_j	7	8	9
length p_j	6	2	2

Lateness $L_j = C_j - d_j$

$L_{max} = \max_j L_j$



$L_{max} = \mathrm{Max}\{-1, 0, 1\} = 1$

Earliest Due Date (EDD) :

Schedule jobs in increasing order of due dates.

$1 \mid d_j \mid L_{max}$

Theorem

EDD is optimal

Proof

Assume not in EDD order: $\mathbf{d_1} > \mathbf{d_2}$

σ 

Swap the jobs:

σ' 

Then, $L_2' = C_2' - d_2 < C_2 - d_2 = L_2$

$L_1' = C_1' - d_1 = C_2 - d_1 < C_2 - d_2 = L_2$

$\rightarrow \max\{L_1', L_2'\} \leq \max\{L_1, L_2\} \rightarrow L_{max}' \leq L_{max}$.

Hence, EDD is optimal.

We have seen some easy problems

$1 \mid\mid \Sigma_j C_j$ ordering by length is optimal

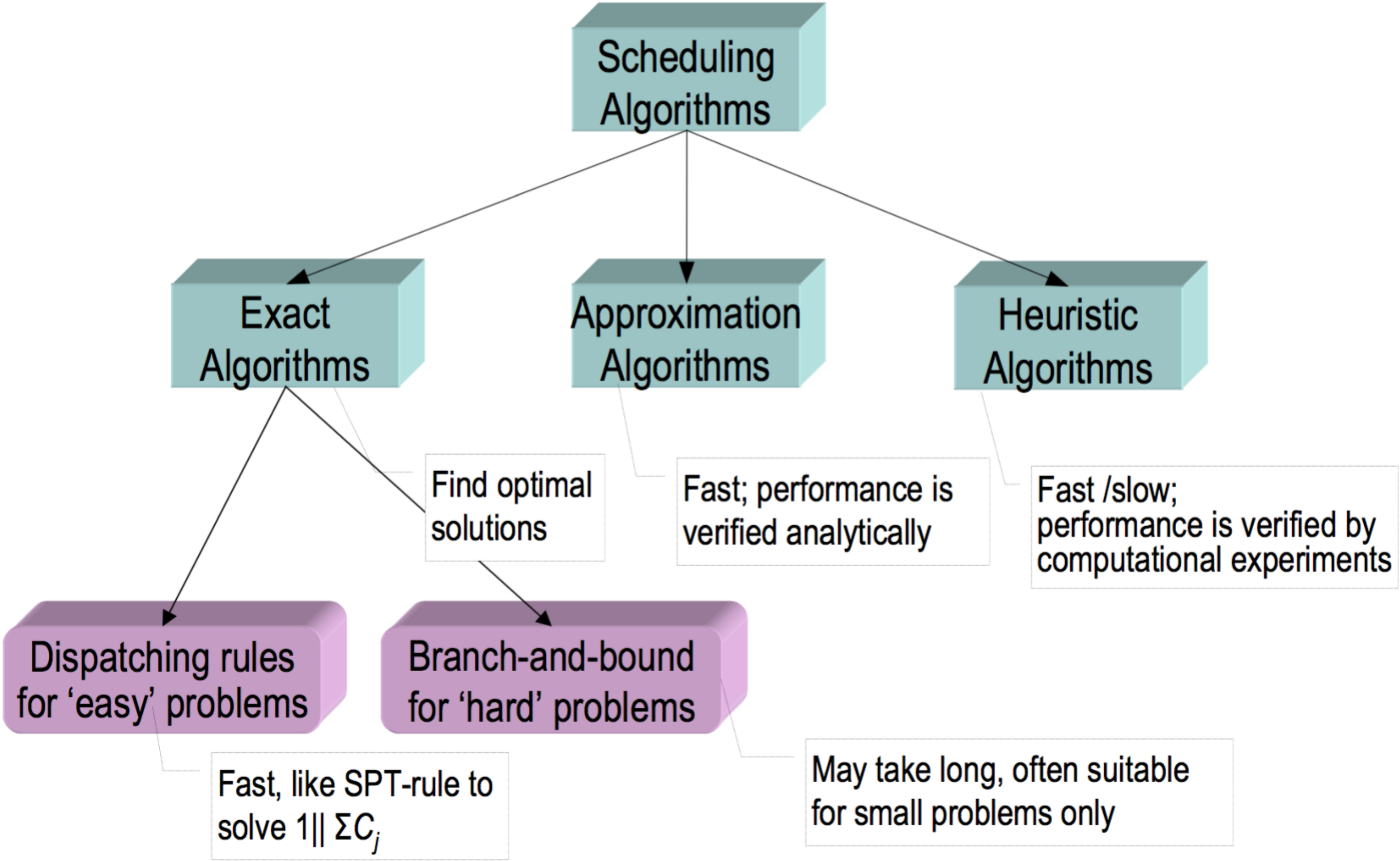
$1 \mid p_j=1 \mid \Sigma_j w_j C_j$ ordering by weight is optimal

$1 \mid . \mid \Sigma_j w_j C_j$ ordering by w_j/p_j is optimal

$1 \mid r_j, pmtn \mid \Sigma_j C_j$ SRPT is optimal

$1 \mid d_j \mid L_{max}$ EDD is optimal

Scheduling algorithms



Approximation algorithms

An α−approximation algorithm:

① The algorithm runs in polynomial time.

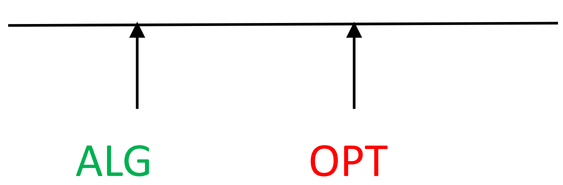
② The algorithm always produces a feasible solution.

③ The value is within a factor α of the optimal value

Maximization problem

For all instances:

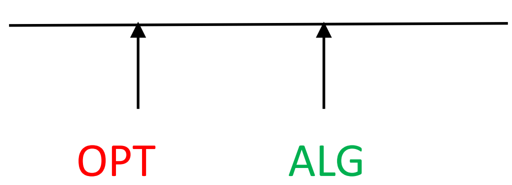
$$\text{ALG} \geq \alpha\text{OPT} \quad (\alpha \leq 1)$$



Minimization problem

For all instances:

$$\text{ALG} \leq \alpha\text{OPT} \quad (\alpha \geq 1)$$

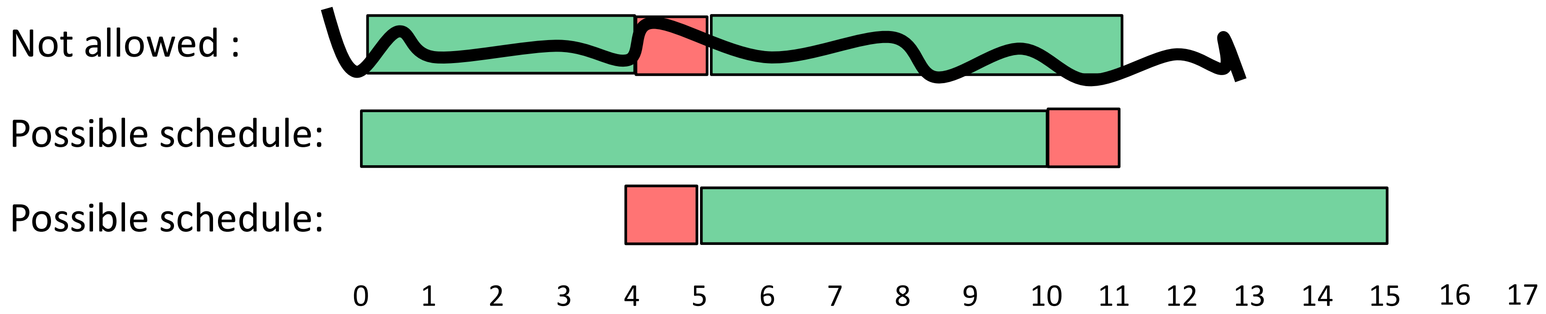


$1|r_j|\ \Sigma_j C_j$

- release times
- preemption is not allowed

Example

jobs	1	2
release time r_j	0	4
length p_j	10	1



Theorem

Problem $1|r_j|\ \Sigma_j C_j$ is NP–hard.

$1 \mid r_j \mid \Sigma_j C_j$

A 2−approximation algorithm

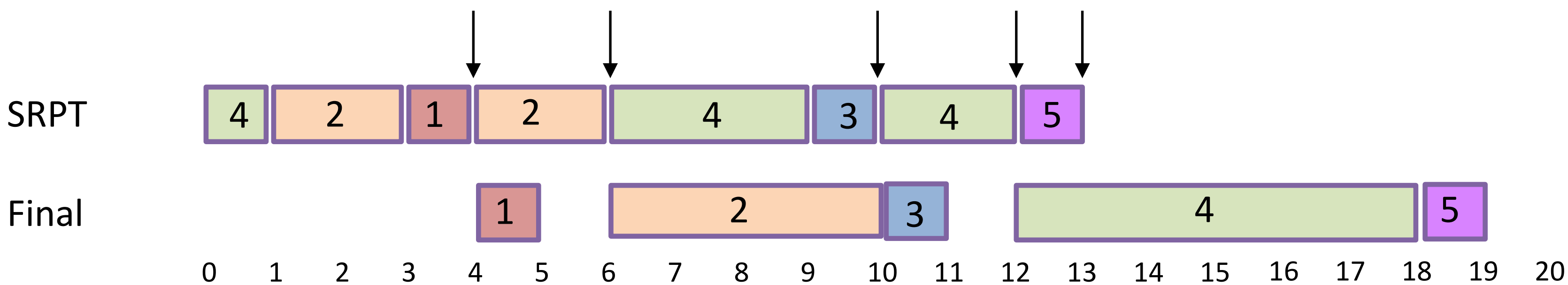
Step 1: Apply Shortest Remaining Processing Time (SRPT).

Step 2: Label jobs by completion time in SRPT schedule: $C_1 < \ldots < C_n$.

For $j=1,2, \ldots ,n$: Schedule job j as early as possible after time C_j

Example

jobs	1	2	3	4	5
release time r_j	3	1	9	0	12
length p_j	1	4	1	6	1



$1|r_j|\Sigma_j C_j$

Proof of approximation ratio 2

Denote

C_j is the completion time of job j in SRPT schedule

C'_j is the completion time of job j in final schedule

Observations:

1. $p_1 + \ldots + p_j \leq C_j$
2. In the final schedule, between time C_j and C'_j there is no idle time.
3. In the final schedule, between time C_j and C'_j there are only jobs $k \leq j$.

→ $C'_j \leq C_j + (p_1 + \ldots + p_j) \leq 2C_j$

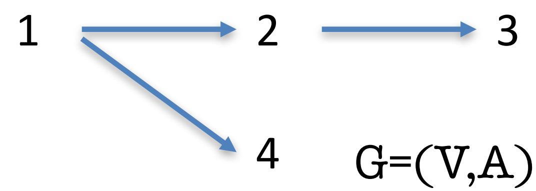
→ $\Sigma_j C'_j \leq 2\Sigma_j C_j \leq 2OPT.$

$1 \,|\, prec \,|\, \Sigma_j C_j$

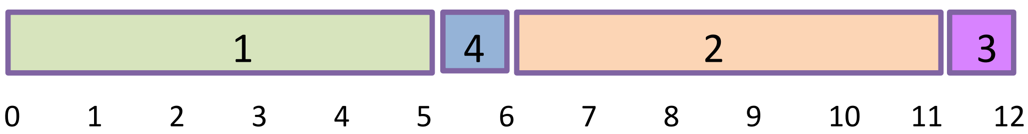
Precedence constraints

Example

jobs	1	2	3	4
length p_j	5	5	1	1



$G=(V,A)$



Theorem $1 \,|\, prec \,|\, \Sigma_j C_j$ is NP–hard

(LP) min $\Sigma_j C_j$

s.t. $C_j \geq p_j$ all jobs j

$C_j \geq C_k + p_j$ all $(k,j) \in A$

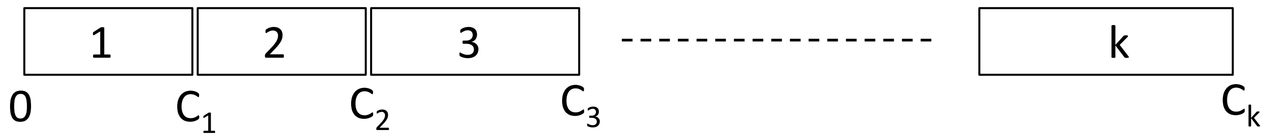
`no overlap of jobs' (not a linear constraint)

$1 \mid prec \mid \Sigma_j C_j$

Denote $p(S) = \Sigma_{j \in S}\, p_j$: total processing time of jobs in subset S.

Lemma $\Sigma_{j \in S}\, p_j C_j \geq \frac{1}{2}\, p(S)^2$ for any subset of jobs S.

proof



Let S={1,2,..,k}. Then,

$$
\begin{aligned}
p_1 C_1 &= p_1 p_1 \\
p_2 C_2 &= p_2(p_1+p_2) \\
p_3 C_3 &= p_3(p_1+p_2+p_3) \\
&\dots \\
+\quad p_k C_k &= p_k(p_1+p_2+\dots+p_k) \\
\hline
\Sigma_j\, p_j C_j &= \tfrac{1}{2}(p_1+p_2+\dots+p_k)^2 + \tfrac{1}{2}(p_1)^2 + \dots + \tfrac{1}{2}(p_k)^2 \\
&\geq \tfrac{1}{2}(p_1+p_2+\dots+p_k)^2 \\
&= \tfrac{1}{2} p(S)^2
\end{aligned}
$$

$1 \mid prec \mid \Sigma_j C_j$

(LP) min $\Sigma_j C_j$

s.t. $C_j \geq p_j$ all jobs j

$C_j \geq C_k + p_j$ all $(k,j) \in A$

$\Sigma_{j \in S} p_j C_j \geq \frac{1}{2} p(S)^2$ all $S \subseteq \{1,2,..,n\}$

2–approximation algorithm

Step 1: Solve LP

Step 2: Schedule jobs in order of increasing LP–values.

Proof

- Feasible? Yes, by LP–constraint $C_j \geq C_k + p_j$ for $(k,j) \in A$
- Polynomial time?
- Ratio?

$1 | prec | \Sigma_j C_j$

(LP) min $\Sigma_j C_j$

s.t. $C_j \geq p_j$ all jobs j

$C_j \geq C_k + p_j$ all $(k,j) \in A$

$\Sigma_{j\in S} p_j C_j \geq \frac{1}{2} p(S)^2$ all $S \subseteq \{1,2,..,n\}$

Lemma (proof omitted)

Let $C_1 \leq C_2 \leq \ .. \leq C_n$ be an LP–solution.

If $\Sigma_{j\in S} p_j C_j \geq \frac{1}{2} p(S)^2$ for all $S = \{1,2,..,k\}$ for $k=1..n$,

then $\Sigma_{j\in S} p_j C_j \geq \frac{1}{2} p(S)^2$ for all $S \subseteq \{1,2,..,n\}$

Corollary

This LP has a separation oracle. (see definition further on)

$1|prec|\Sigma_j C_j$

(LP) min $\Sigma_j C_j$

s.t. $C_j \geq p_j$ all jobs j

$C_j \geq C_k + p_j$ all $(k,j) \in A$

$\Sigma_{j\in S}\, p_j C_j \geq \frac{1}{2}\, p(S)^2$ all $S \subseteq \{1,2,..,n\}$

Proof of ratio

C_j is the completion time of job j in the LP

C'_j is the completion time of job j in final schedule

Final schedule: | 1 | 2 | ----- | k | ----- | n |

C'_k

$S=\{1,2,..,k\}$ for fixed job k

$$\begin{aligned}\tfrac{1}{2}\, p(S)^2 &\leq \Sigma_{j\in S}\, p_j C_j \\ &\leq \Sigma_{j\in S}\, p_j C_k \\ &= p(S) C_k \qquad \rightarrow p(S) \leq 2C_k\end{aligned}$$

Hence, $C'_k = p(S) \leq 2C_k \ \rightarrow\ ALG = \Sigma_k C'_k \leq 2\, \Sigma_k C_k \leq 2OPT.$

The Simplex method

- is very fast in practice but
- may take exponential time in the worst case

The ellipsoid method:

- is not very fast in practice but
- does solve LPs in polynomial time and
- may even solve LPs with an exponential number of contraints

First observation:

Solving an LP can be reduced to finding a feasible solution to a system of linear inequalities: Just find the largest c_0 such that the system has a feasible solution.

$$\begin{aligned} \text{maximize } c^T x \\ Ax \le b \\ x \ge 0 \end{aligned} \quad \Longrightarrow \quad \begin{aligned} c^T x \ge c_0 \\ Ax \le b \\ x \ge 0 \end{aligned}$$

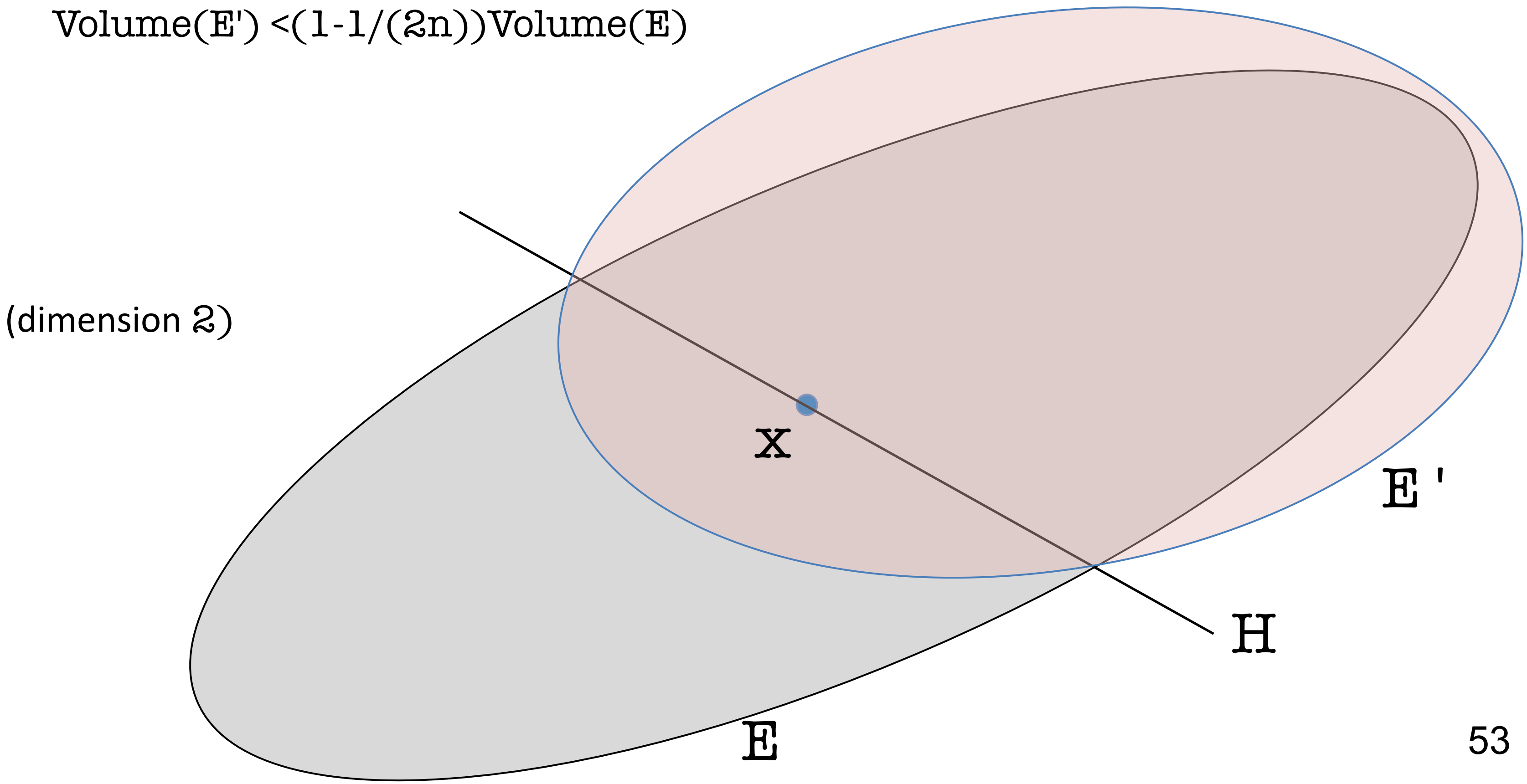
We will sketch how this problem of finding a feasible solution can be solved in polynomial time.

Mathematicians showed the following [1].

Let E be an ellips of dimension n and
let H be an hyperplane containing the center x of E. That means, H splits E exactly in half.

Then, it is possible to compute (in polynomial time) an ellips E' that contains one half of the ellips and

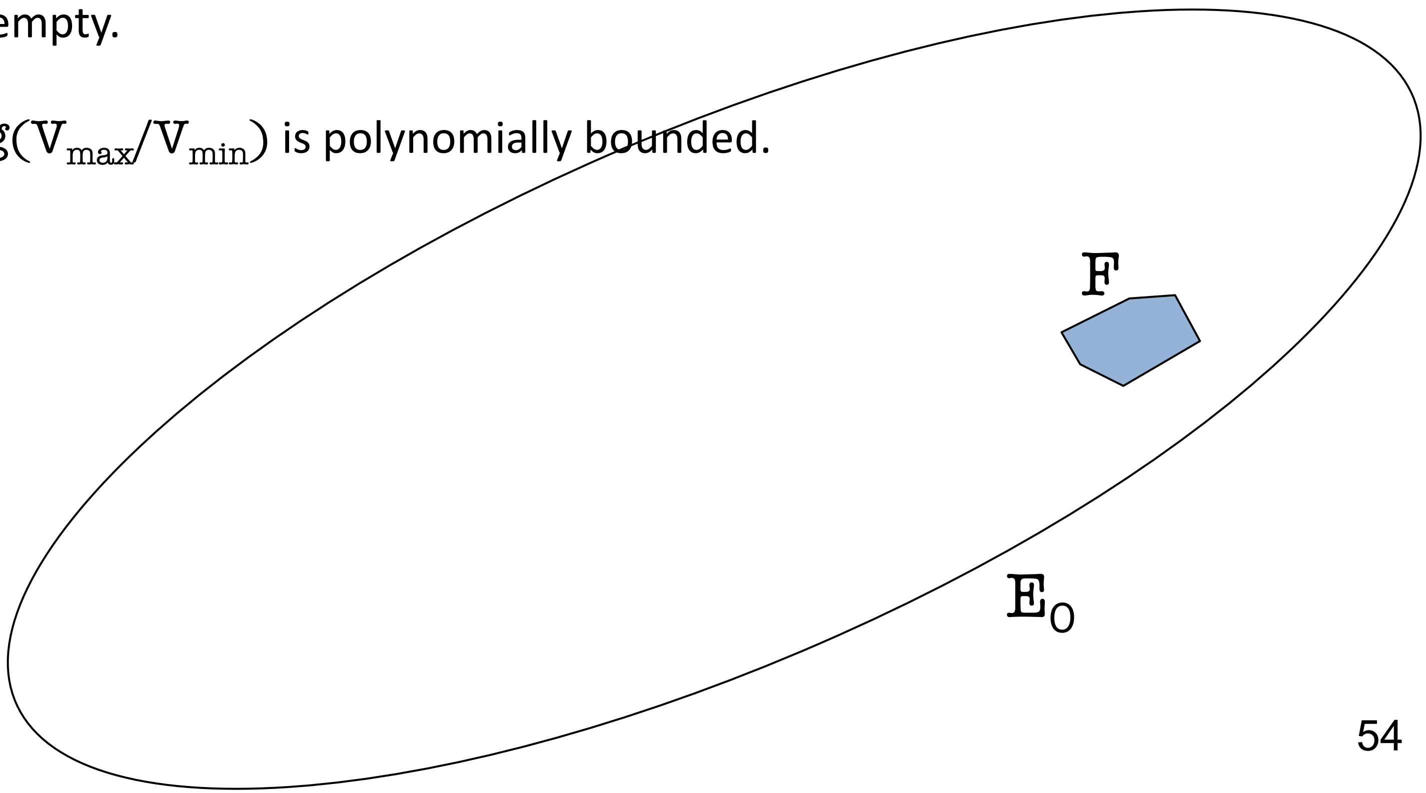
Volume(E') <(1-1/(2n))Volume(E)



Let F be the feasible region of the given system of inequalities. (It might be empty.)

Mathematicians showed the following [2]:

- It is possible to compute an ellipse E_0 such that E_0 contains F, if F it is not empty. Denote the volume of E_0 by V_{max}.
- It is possible to compute a number $V_{min} > 0$ such that if $\text{Volume}(F) < V_{min}$ then F must be empty.
- Further, $\log(V_{max}/V_{min})$ is polynomially bounded.



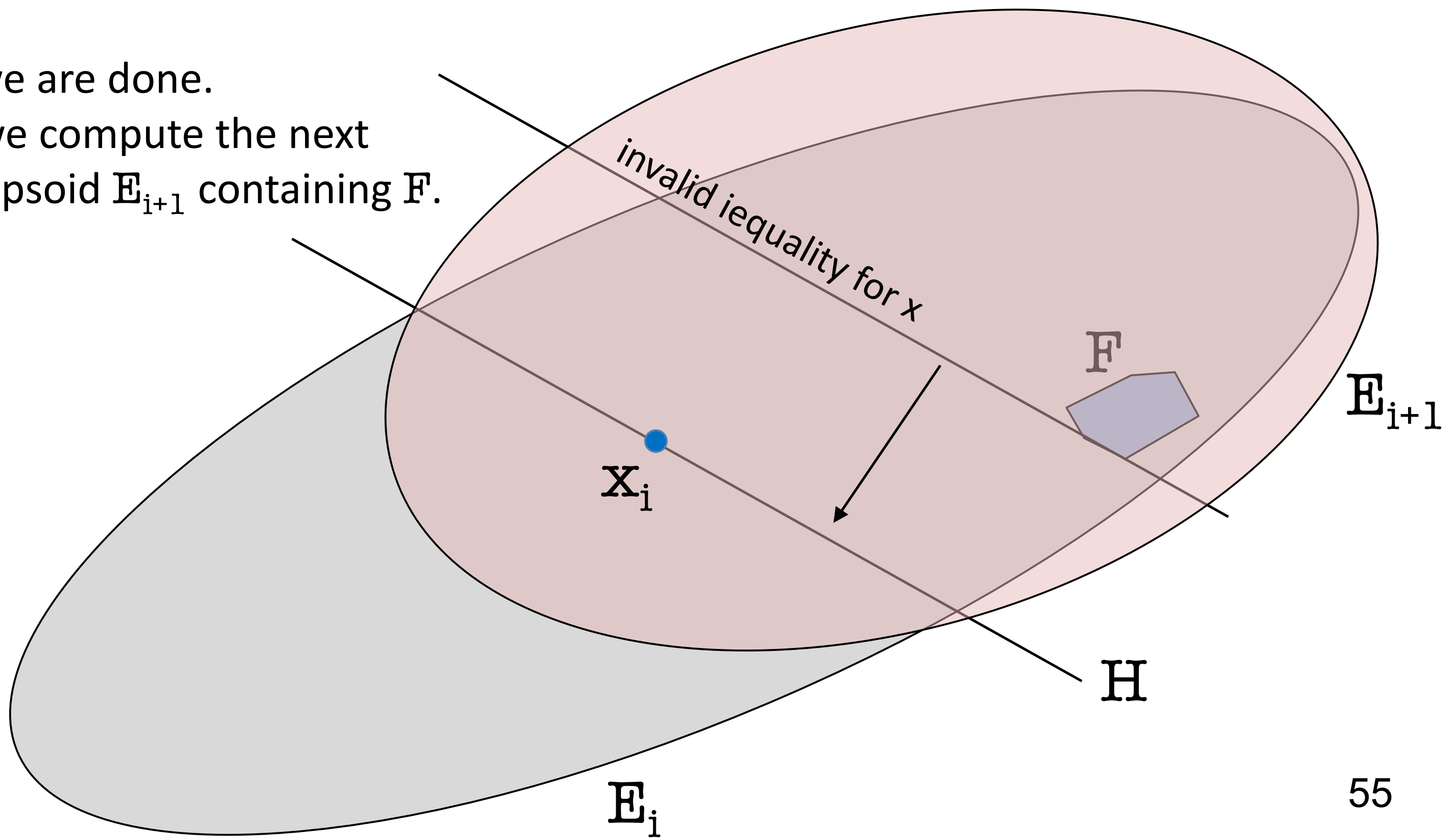
One iteration of the ellipsoid method

Let $\mathtt{x}_i$ be the center of ellipsoid $\mathtt{E}_i$ which contains $\mathtt{F}$.
Then either

a) $\mathtt{x}_i \in \mathtt{F}$ or

b) $\mathtt{x}_i \notin \mathtt{F}$ and then we can find a violated inequality (by checking all inequalities).

In case (a) we are done.
In case (b) we compute the next (smaller) ellipsoid $\mathtt{E}_{i+1}$ containing $\mathtt{F}$.

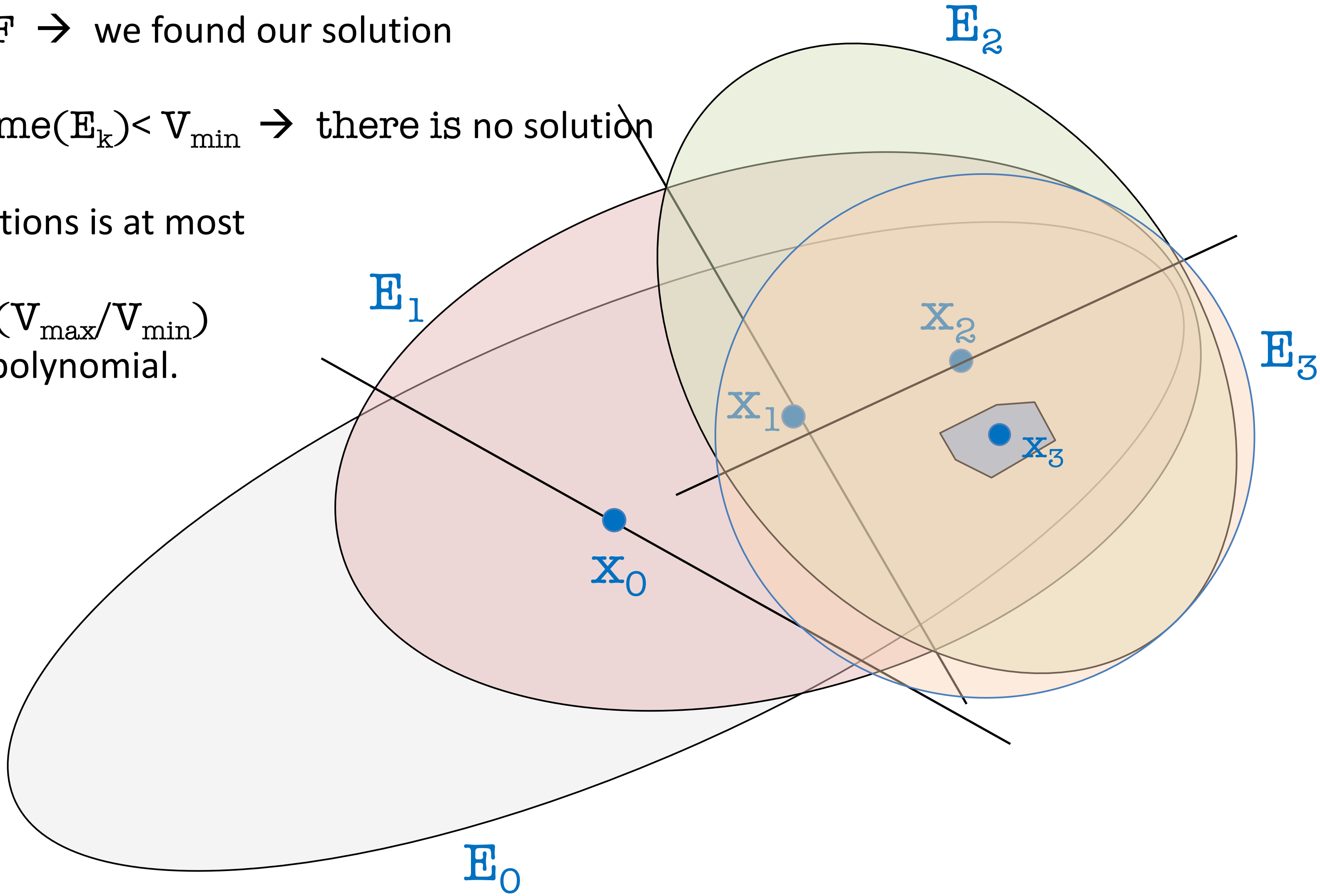


Ellipsoid method ends in iteration $\mathtt{k}$ if:

a) $x_k \in F$ → we found our solution
or
b) $\text{Volume}(E_k) < V_{min}$ → there is no solution

→ # iterations is at most

$O(n)\log(V_{max}/V_{min})$
which is polynomial.



Note that the ellipsoid method runs in polynomial time as long as we have an algorithm that can do the following in polynomial time:

The input is a system of n-dimensional inequalities and $x \in R^n$.
The algorithm either

- tells us that x is in the feasible region F or
- it returns an inequality which is valid for F but not valid for x. (A separating inequality).

Such an algorithm is called a separation oracle.

LPs with a exponential number of constraints may still have a separation oracle. Hence, such an LP can be solved in polynomial time using the Ellipsoid method.

How to solve this scheduling LP with an exponential number of constraints?

(LP) min		$\Sigma_j C_j$	
	s.t.	$C_j \geq p_j$	all jobs j
		$C_j \geq C_k + p_j$	all $(k,j) \in A$
		$\Sigma_{j \in S} p_j C_j \geq \frac{1}{2} p(S)^2$	all $S \subseteq \{1,2,..,n\}$ (*)

In theory, we could use the ellipsoid method, but not very practical (slow).
We take a slightly different (easier) approach.
But we *do use the fact that the LP has a separation oracle*.

Remove constraint (*) from the LP.
Repeat:
Solve the LP
Let $C_1, C_2, \ldots, C_n$ be the solution found.
Let π be a permutation of 1,2,…,n such that $C_{\pi(1)} \leq C_{\pi(2)} \leq \ldots \leq C_{\pi(n)}$.
Let $S_k = \{\pi(1), \pi(2), \ldots, \pi(k)\}$, for k=1,2,..,n.
If constraint (* *) holds all S_k for k=1,2,..,n then:
the current solution is optimal for the complete LP. **Stop**.
Else
add all the violated constraints S_k to the LP.

Results single machine:

1) $1 \mid\mid \Sigma C_j$ SPT is optimal

2) $1 \mid\mid \Sigma w_j C_j$ Smith's ratio rule is optimal: Order by w_j/p_j

3) $1 \mid r_j, pmtn \mid \Sigma C_j$ SRPT is optimal

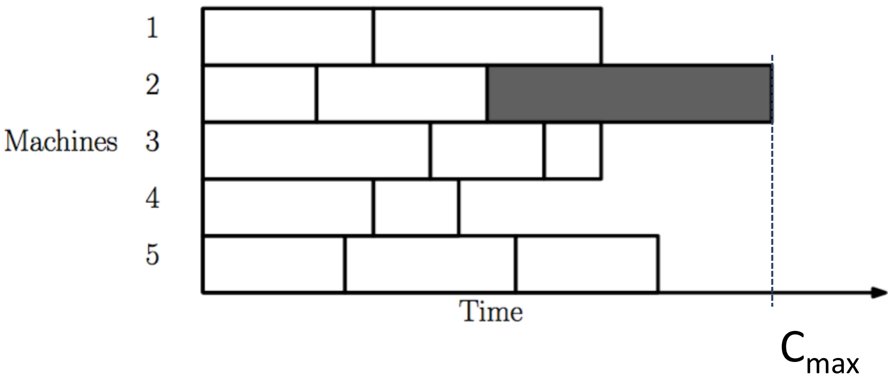
4) $1 \mid\mid L_{max}$ Earliest Due Date (EDD) is optimal

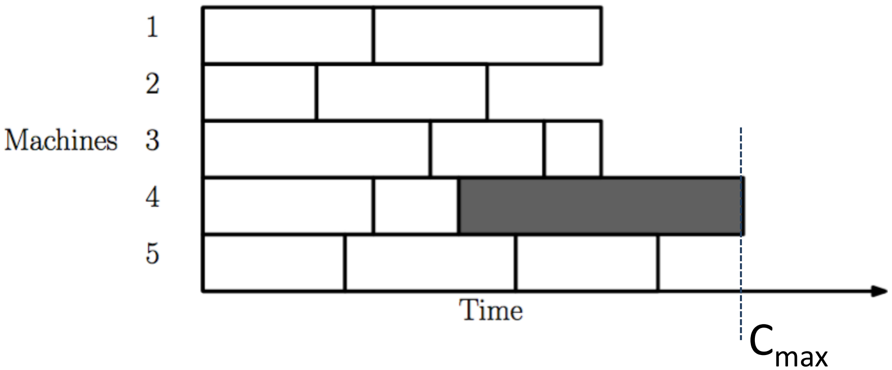
5) $1 \mid r_j \mid \Sigma C_j$ NP-hard. SRPT order gives 2-approximation.

6) $1 \mid prec \mid \Sigma C_j$ NP-hard. LP order gives 2-approximation.

$P|.|C_{max}$

Scheduling jobs on a identical parallel machines:

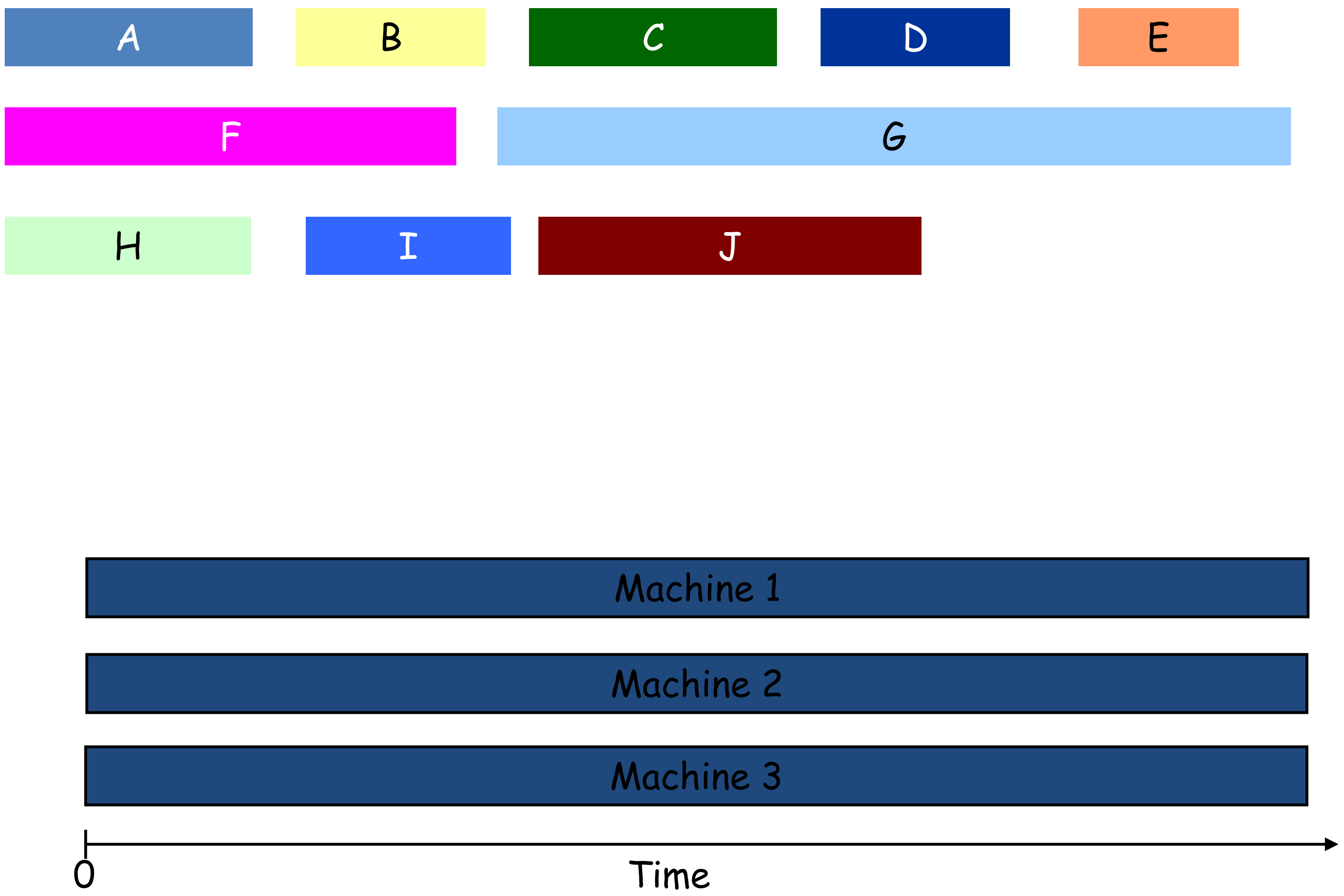




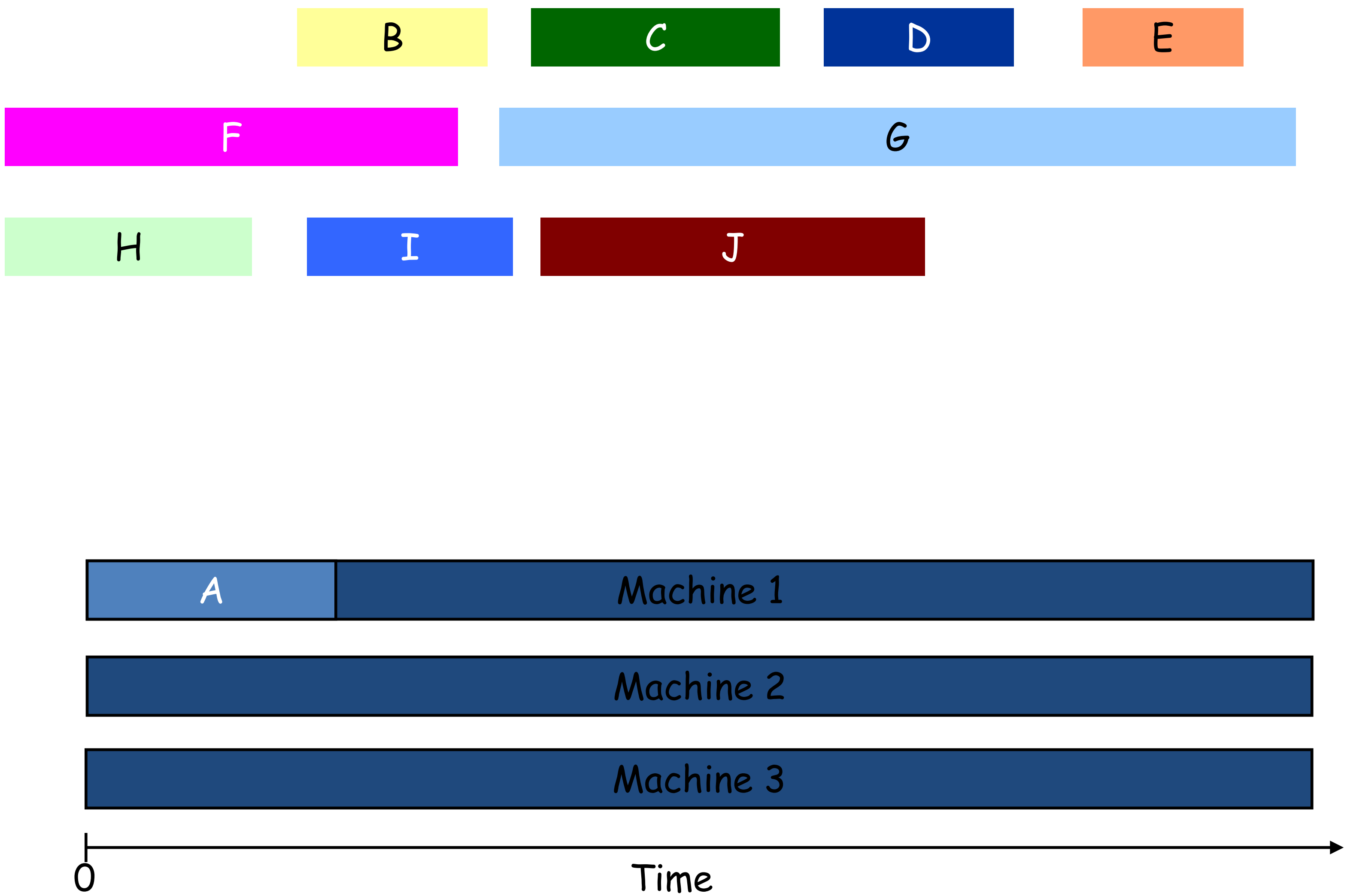
List Scheduling:

Assign the jobs one by one (in arbitrary order) to the machines.
At any step, choose the machine with the smallest load sofar.

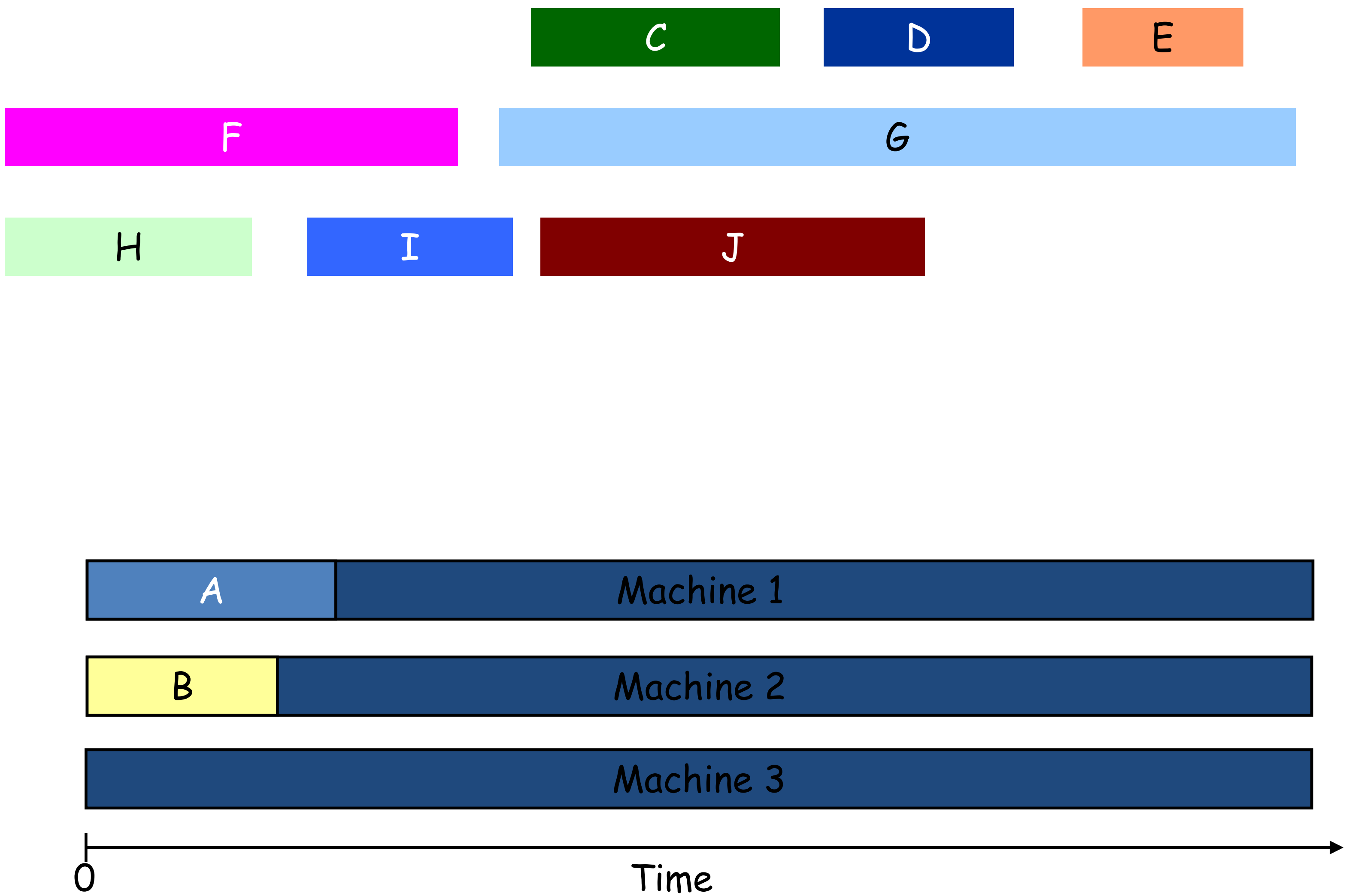
List Scheduling



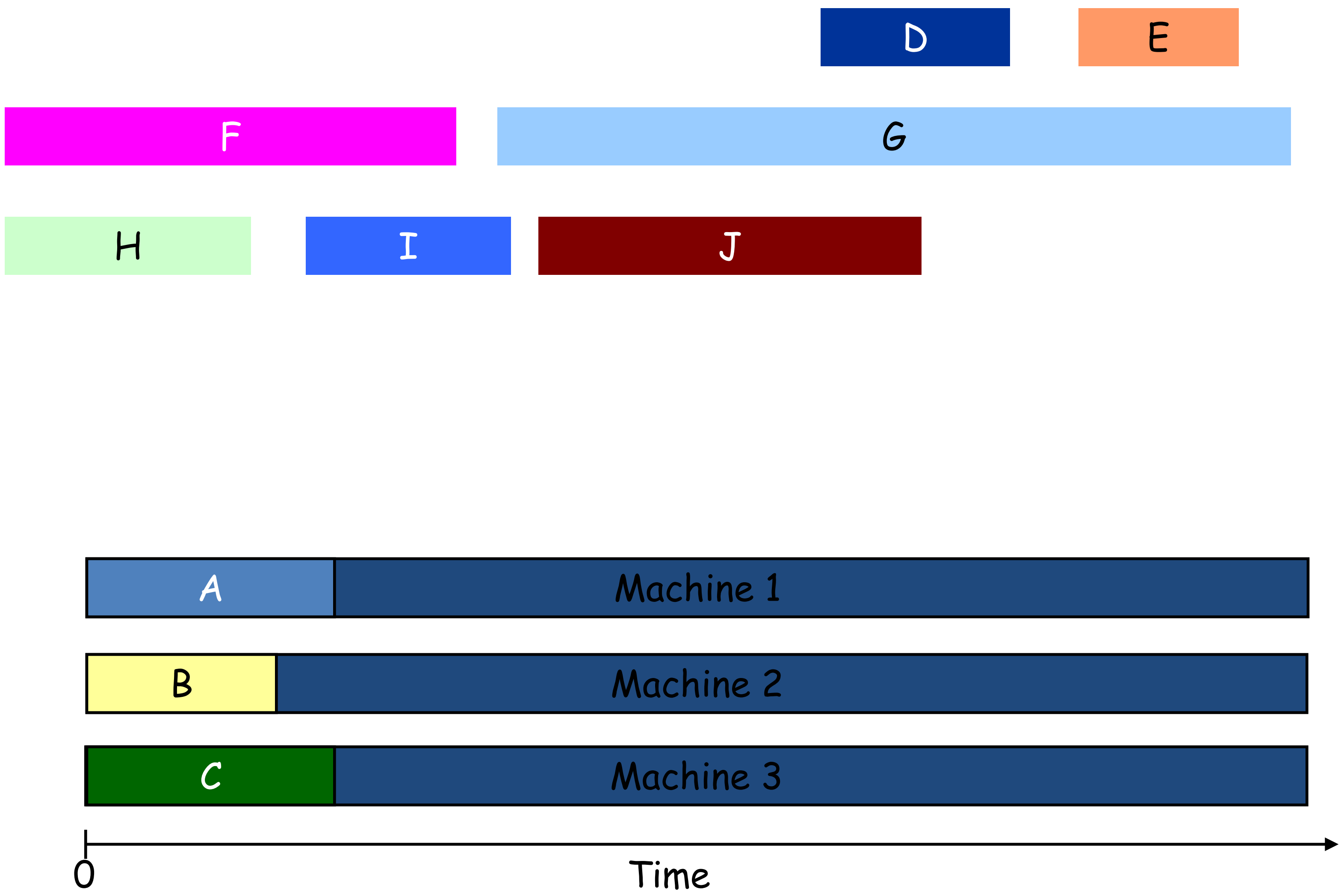
List Scheduling



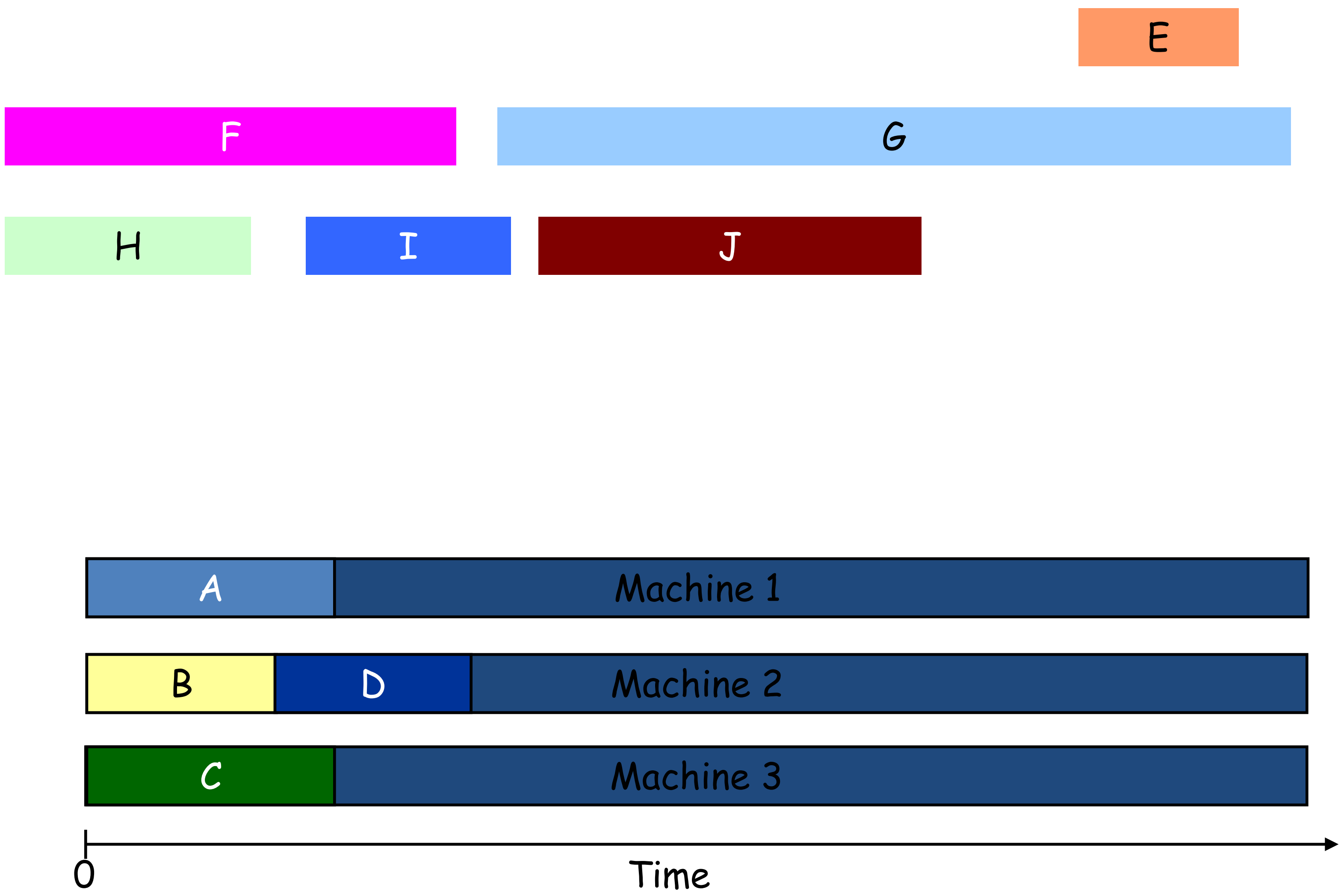
List Scheduling



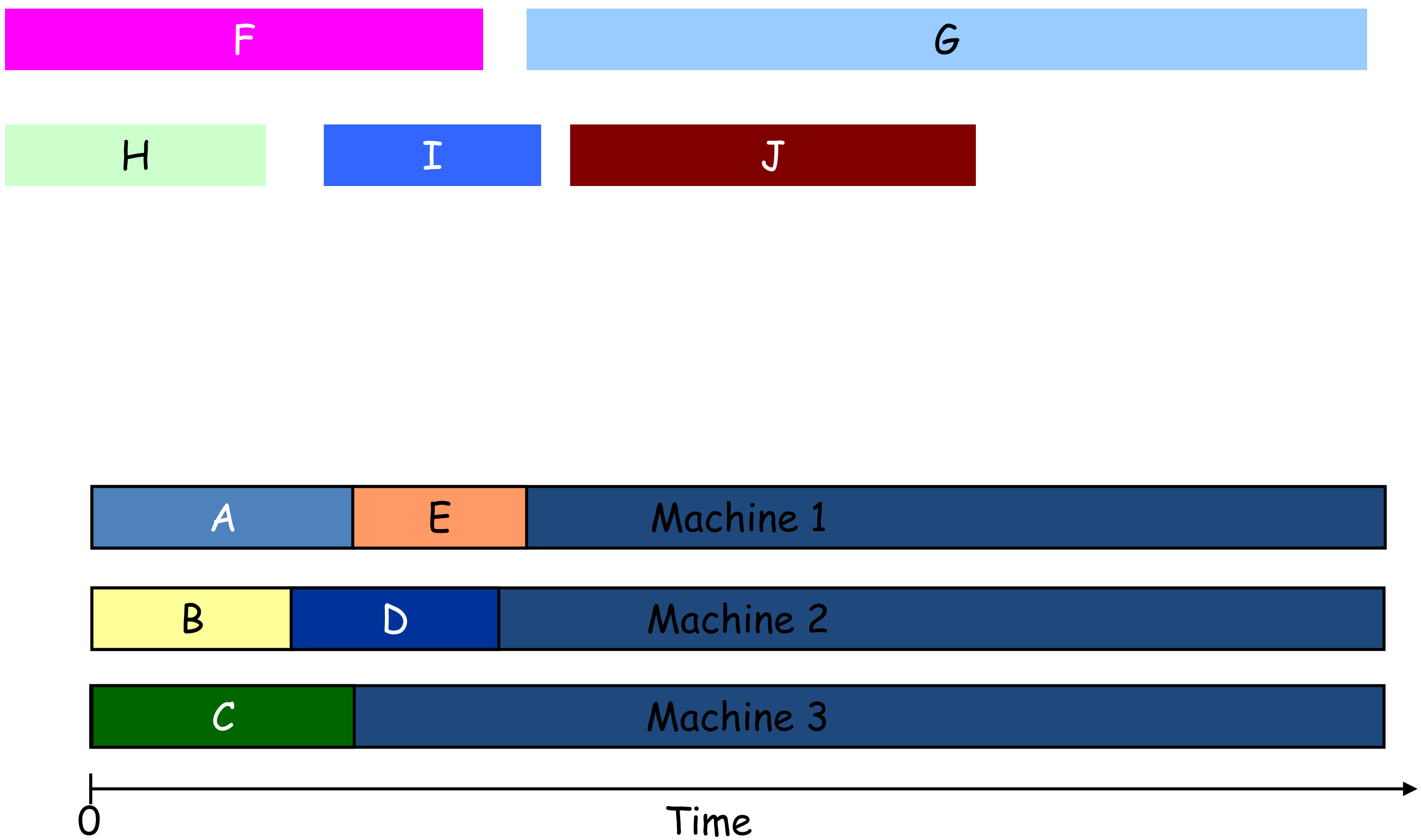
List Scheduling



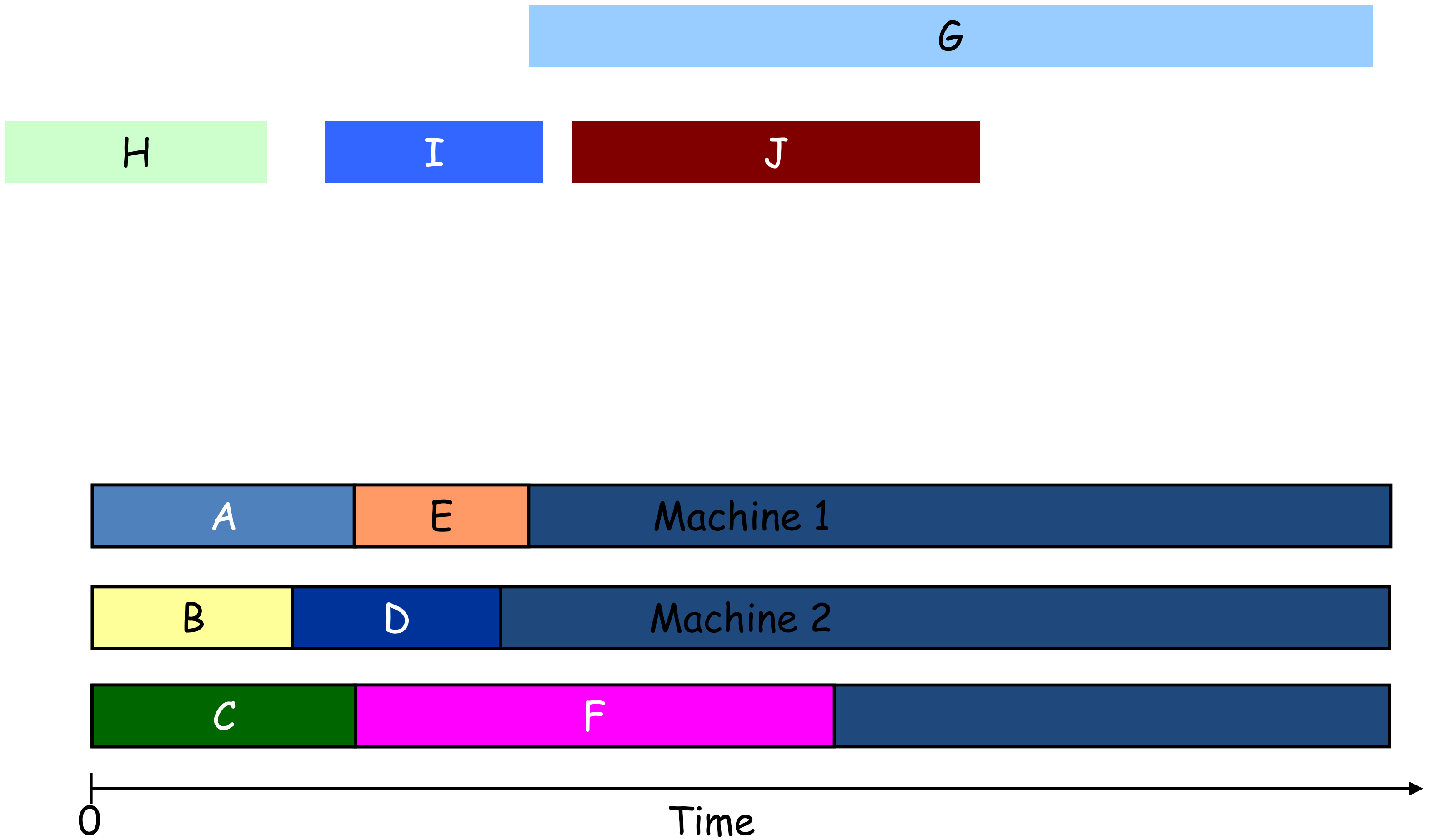
List Scheduling



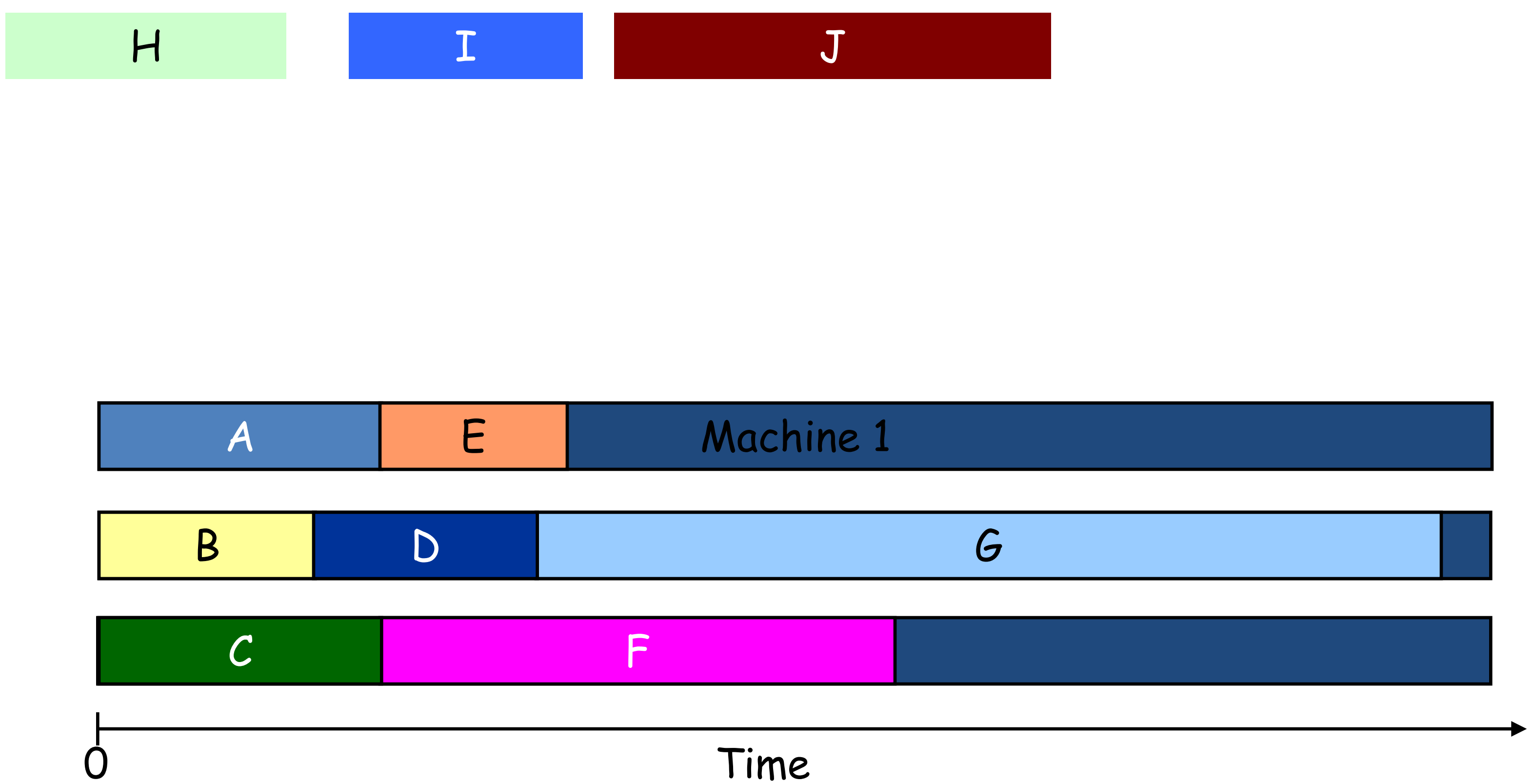
List Scheduling



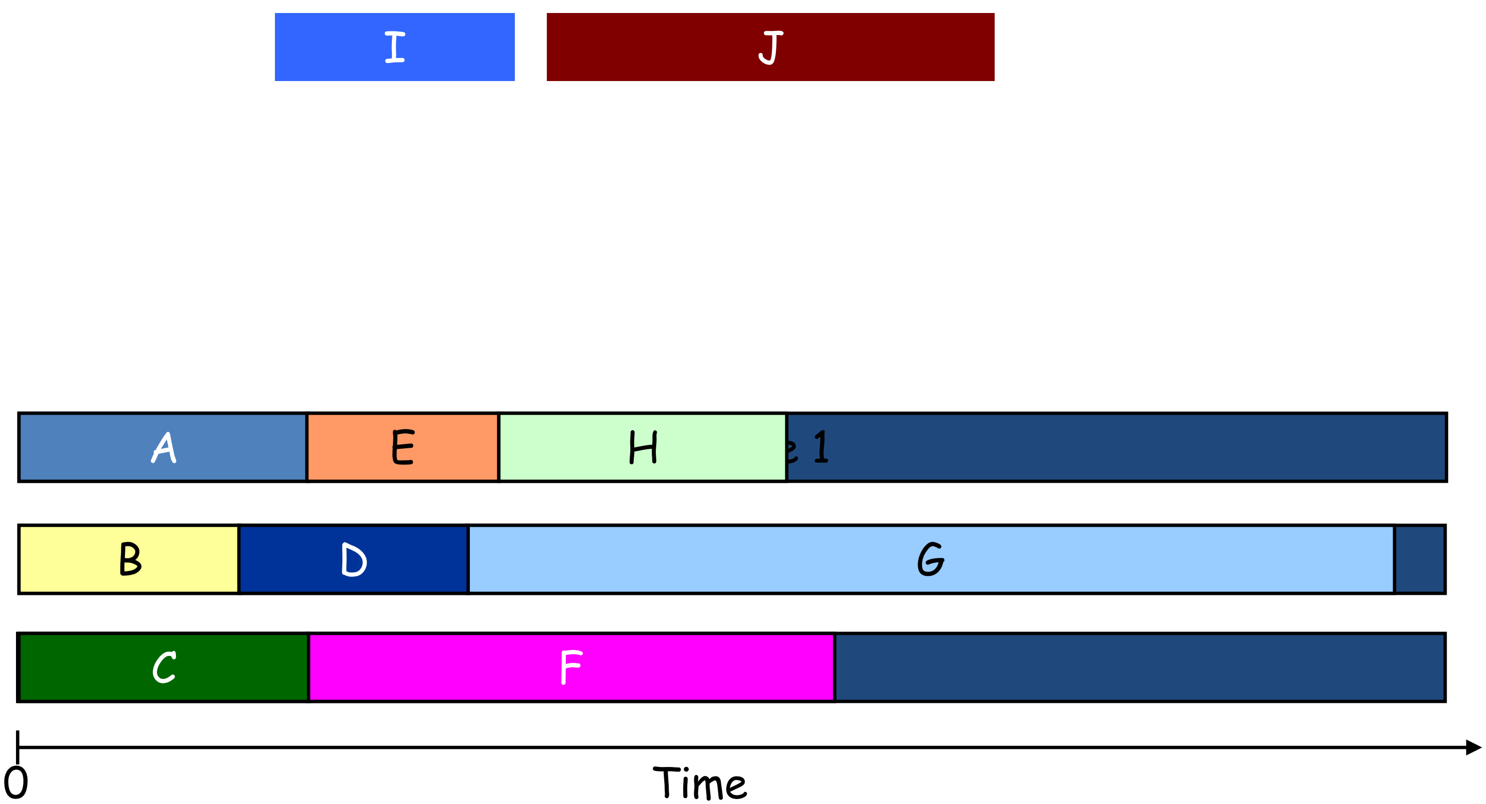
List Scheduling



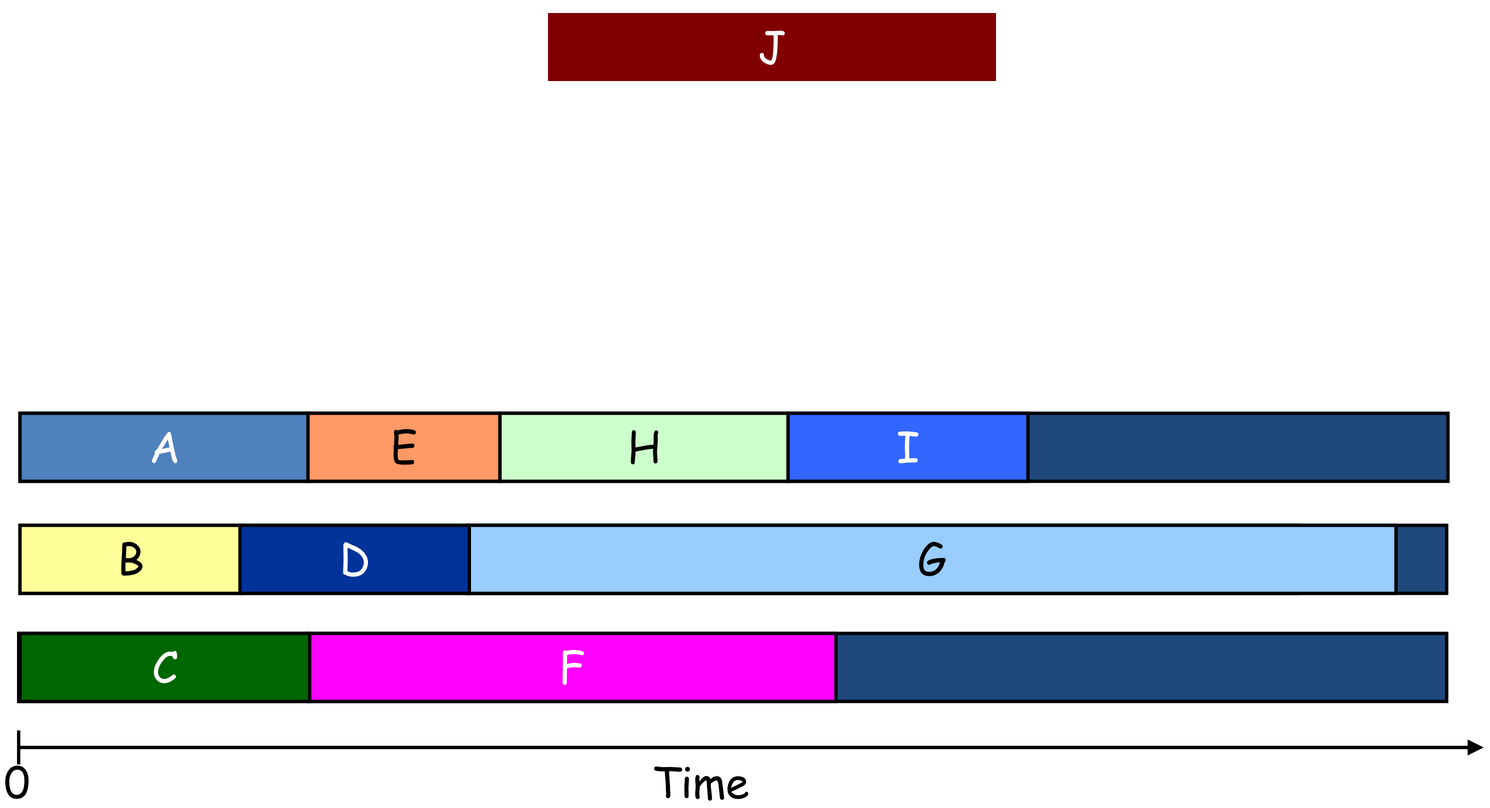
List Scheduling



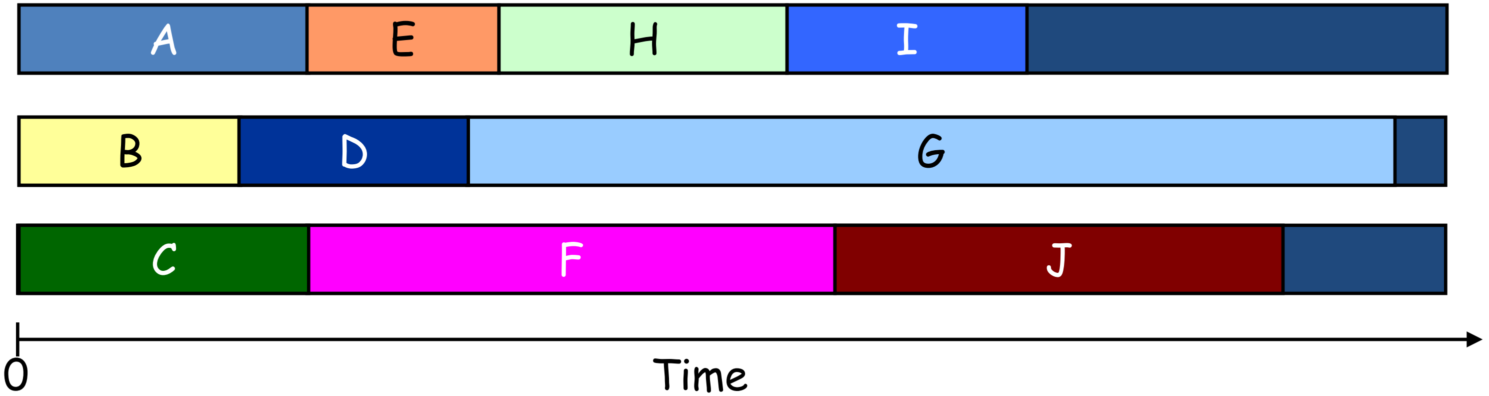
List Scheduling



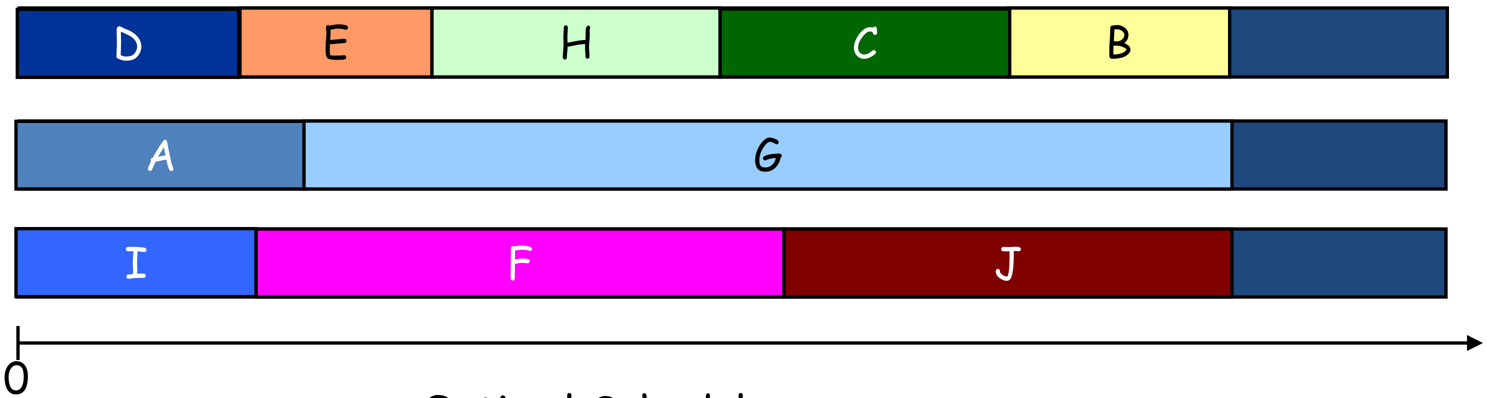
List Scheduling



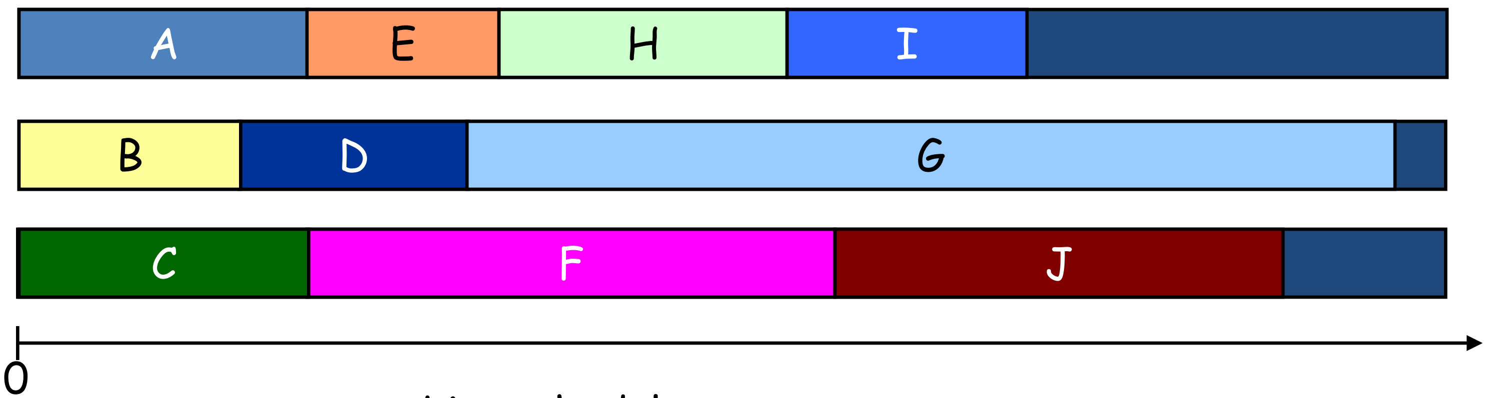
List Scheduling



List Scheduling



Optimal Schedule



List schedule

$P|.|C_{max}$

Theorem

List scheduling is a $(2-1/m)$-approximation algorithm.

Proof

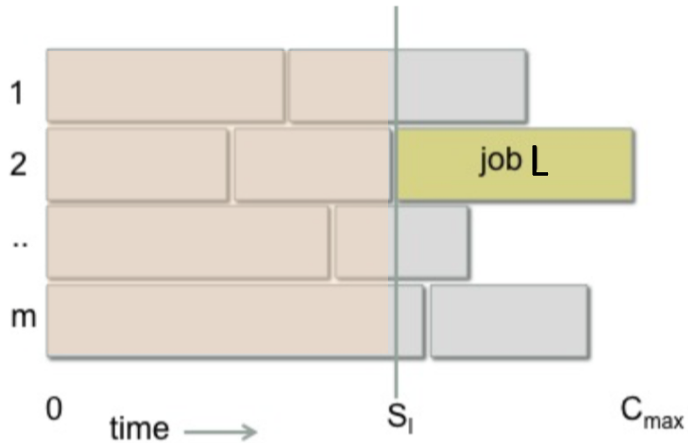
C^*_{max} : Optimal makespan

p_{max} : $\max_j p_j$

Lower Bound 1: $C^*_{max} \geq p_{max}$

Lower Bound 2: $C^*_{max} \geq (p_1+p_2+\ldots+p_n)/m$

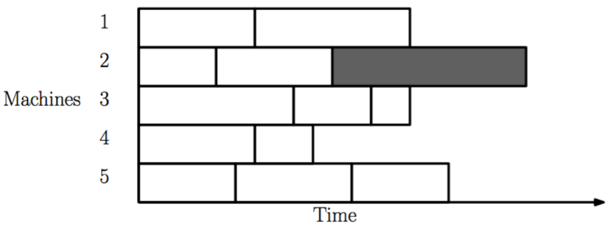
Let job L be last.

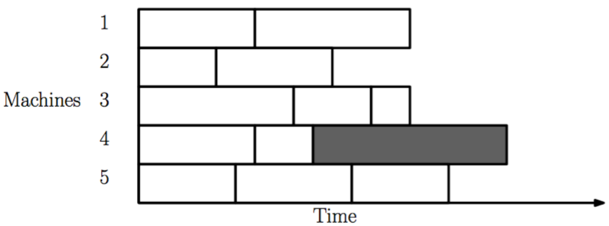


From LS alg: $mS_L \leq (p_1+p_2+\ldots+p_n) - p_L$

$$
\begin{aligned}
C_{max} &= S_L + p_L \\
&\leq (p_1+p_2+\ldots+p_n)/m + (1-1/m)p_L \\
&\leq C^*_{max} + (1-1/m)C^*_{max} \\
&= (2-1/m)C^*_{max}
\end{aligned}
$$

P|.| C_{max}





Local Search:
Start with any schedule.
Repeat as long as possible:
  Move a job to the end of least loaded machine
  ... if that reduces its completion time.

Theorem
Local search is a (2−1/m)−approximation algorithm.
Proof

Ratio? Follows from List Scheduling proof. -> **Exercise**

Polynomial time? Yes. Each job moves at most once. -> **Exercise**

$P|.|C_{max}$

LPT (Longest Processing Time first)

Order jobs : $p_1 \geq p_2 \geq .. \geq p_n$. Apply list scheduling in this order.

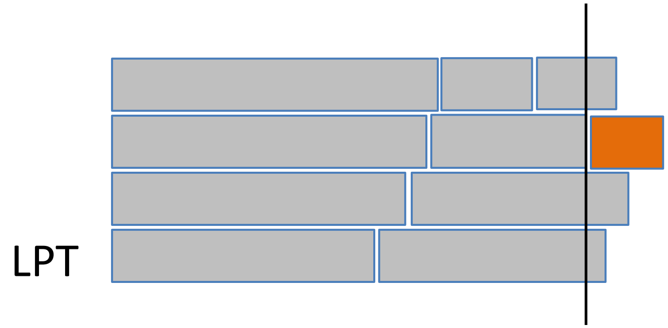
Theorem

LPT is a 4/3-approximation algorithm.

Proof

Case 1: last job has $p_j \leq OPT/3$

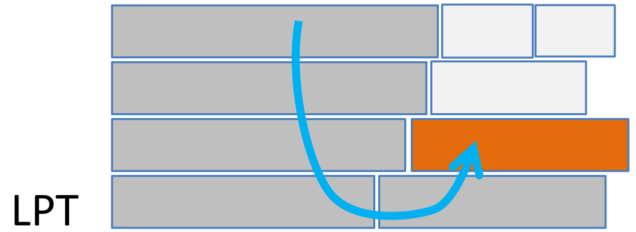
→ $C_{max} \leq OPT + OPT/3$



Case 2: last job has $p_j > OPT/3$

→ OPT has at most 2 jobs per machine

→ LPT is optimal.

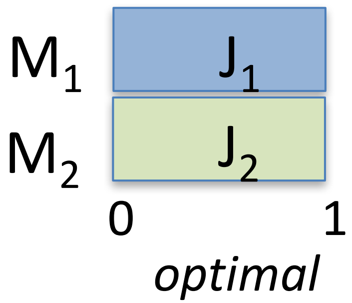


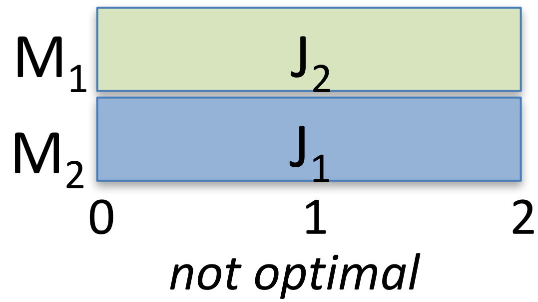
$R|.|\sum_j C_j$

Unrelated machines

p_{ij} : Processing time of job j depends on machine i

p_{ij}	1	2
1	1	2
2	2	1





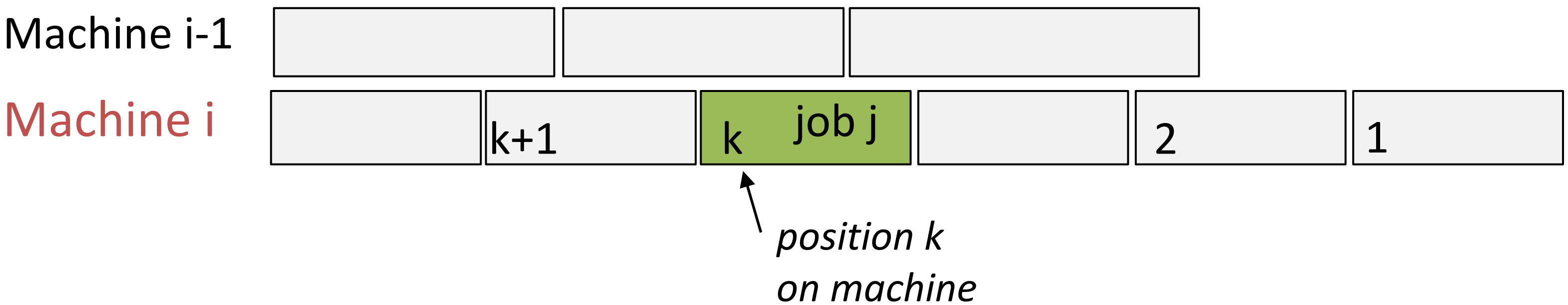
Theorem

The problem $R|.|\sum_j C_j$ can be reduced to the assignment problem

$R \mid . \mid \sum_j C_j$

Observation:

If job j is scheduled on machine i on position k then it contributes exactly $\mathbf{kp_{ij}}$ to the total completion time. **(-> Exercise)**



	1	2		j		n
1,1						
1,2						
...						
1,n						
2,1						
...						
i,k				kp_{ij}		
...						
m,n						

Problem

Find a mincost perfect matching of jobs to positions on machines.
→ assignment problem.

R|.| C_{max}

- Unrelated machines
- Minimize length (makespan)

$$
\begin{aligned}
\text{(LP)}\quad \min \quad & Z \\
s.t. \quad & \sum_{i=1}^{m} x_{ij} = 1 && \text{for all jobs } j \\
& \sum_{j=1}^{n} x_{ij} p_{ij} \leq Z && \text{for all machines } i \\
& x_{ij} \geq 0 && \text{for all } i, j
\end{aligned}
$$

$R \mid . \mid C_{max}$

Algorithm

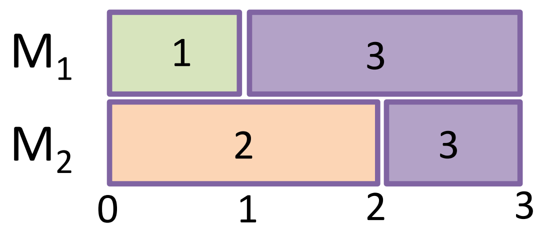
Step 1 Solve LP → x, Z_{LP}

Step 2 Assign j to machine i if $x_{ij}=1$.

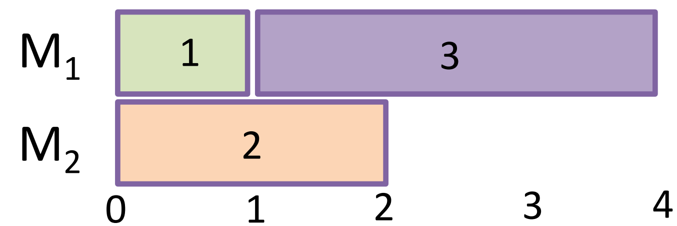
Step 3 Assign the fractional jobs in an optimal way.

Example:

p_{ij}	1	2	3
1	1	9	5
2	9	2	5



Optimal LP–solution



Final schedule

Theorem

Algorithm is a 2-approximation algorithm <u>if m is a constant.</u>

Proof

Ratio:

Length for integer jobs ≤ OPT

Length for fractional jobs ≤ OPT → Total length ≤ 2OPT

Time: ?? (next slide)

$R|.|C_{max}$

Proof Time?

Lemma

Any extreme LP-solution, has at most n+m non-zero variables.

Proof

- nm variables
- $nm + n + m$ constraints.
- In extreme LP-solution, at least nm constraints are tight (=)
  (Known from Lin. Algebra)

→ At least nm – (n+m) variables $x_{ij} = 0$

→ At most (n+m) variables $x_{ij} > 0$.

Corollary

Any extreme LP-solution, has at most m fractional jobs.

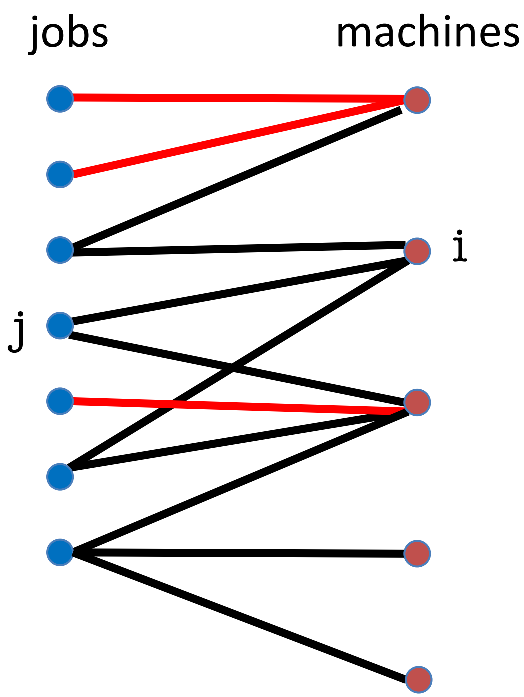
Proof For each fractional job, at least two variables are strictly positive.

$$n+m \geq 2n_f + n_i$$

and $$n = n_f + n_i \quad \rightarrow \quad n_f \leq m.$$

→ Only $O(m^m)$ schedules for fractional jobs.

Can we get the running time polynomial in m?



Support graph : edge if $x_{ij}>0$

From lemma: # edges ≤ # vertices ($n+m$)
This even holds for each component,
since each component is an extreme solution for the induced LP.

→ Each component is a tree or a tree + one edge.

Lemma
For fractional jobs, there is a perfect matching with the machines.

R|.| C_{max} Improving the running time

Algorithm

Step 1 Solve LP → x, Z_{LP}

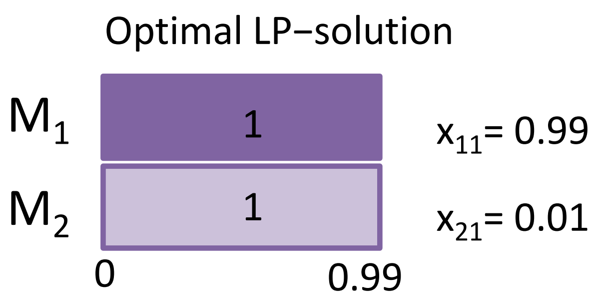
Step 2 Assign j to machine i if $x_{ij}=1$.

Step 3 Assigning fractional jobs ~~in an optimal way~~ by a perfect matching.

Length of schedule is at most OPT + Longest fractional job.

Bad example:

p_{ij}	1
1	1
2	99



$$
\begin{aligned}
\text{(LP)}\quad \min\quad & Z \\
s.t.\quad & \sum_{i=1}^{m} x_{ij} = 1 && \text{for all jobs } j \\
& \sum_{j=1}^{n} x_{ij}p_{ij} \le Z && \text{for all machines } i \\
& x_{ij} \ge 0 && \text{for all } i,j
\end{aligned}
$$

Idea: Guess OPT and let x_{ij}= 0 if p_{ij} > OPT.

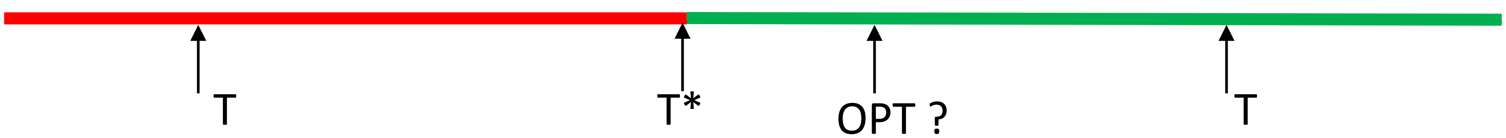
$R|.|C_{max}$ Improving the running time

Guess the optimal makespan *T*. Let $J(T) = \{(i,j) \mid p_{ij} \le T\}$

$$
\begin{array}{ll}
\displaystyle\sum_{i:(i,j)\in J_T} x_{ij}p_{ij} = 1 & \text{for all jobs } j \\
\displaystyle\sum_{j:(i,j)\in J_T} x_{ij}p_{ij} \le T & \text{for all machines } i \\
x_{ij} \ge 0 & \text{for all } (i,j) \in J(T).
\end{array}
$$

Run the algorithm. Then, either

1. it returns a schedule of length at most $2T$
2. or it finds no schedule. But then we know that $T <$ OPT.



By binary search, we find smallest integer T, say T^*, for which the LP has a solution.

→ Length of schedule is at most $2T^* \le 2\text{OPT}$.

Results Parallel machines

7. $P \mid pmtn \mid C_{max}$ - McNaughton's wrap arpund rule is optimal.

8. $P \mid\mid C_{max}$
   - NP-hard.
   - List scheduling is 2-approximation.
   - LPT is 4/3-approximation.

9. $R \mid\mid \Sigma C_j$ - In P since reducible to the min-cost perfect matching.

10. $Rm \mid\mid C_{max}$
    - NP-hard.
    - LP + enumerating schedules gives 2-approx. Running time exponential in m
    - Improvement gives a running time which is polynomial in m.