# CS343: Operating System

# Scheduling Algorithms

## Lect14 : 28th Aug 2023


**Dr. A. Sahu**

**Dept of Comp. Sc. & Engg.**

**Indian Institute of Technology Guwahati**

# Introduction to Scheduling Algorithms

# A bit of Theoretical View

# Classification of Scheduling Problems

Classes of scheduling problems can be specified in terms of the three-field classification

$$\alpha \mid \beta \mid \gamma$$

where

- $\alpha$ specifies the **machine environment**,
- $\beta$ specifies the **job characteristics**, and
- $\gamma$ describes the **objective function(s).**

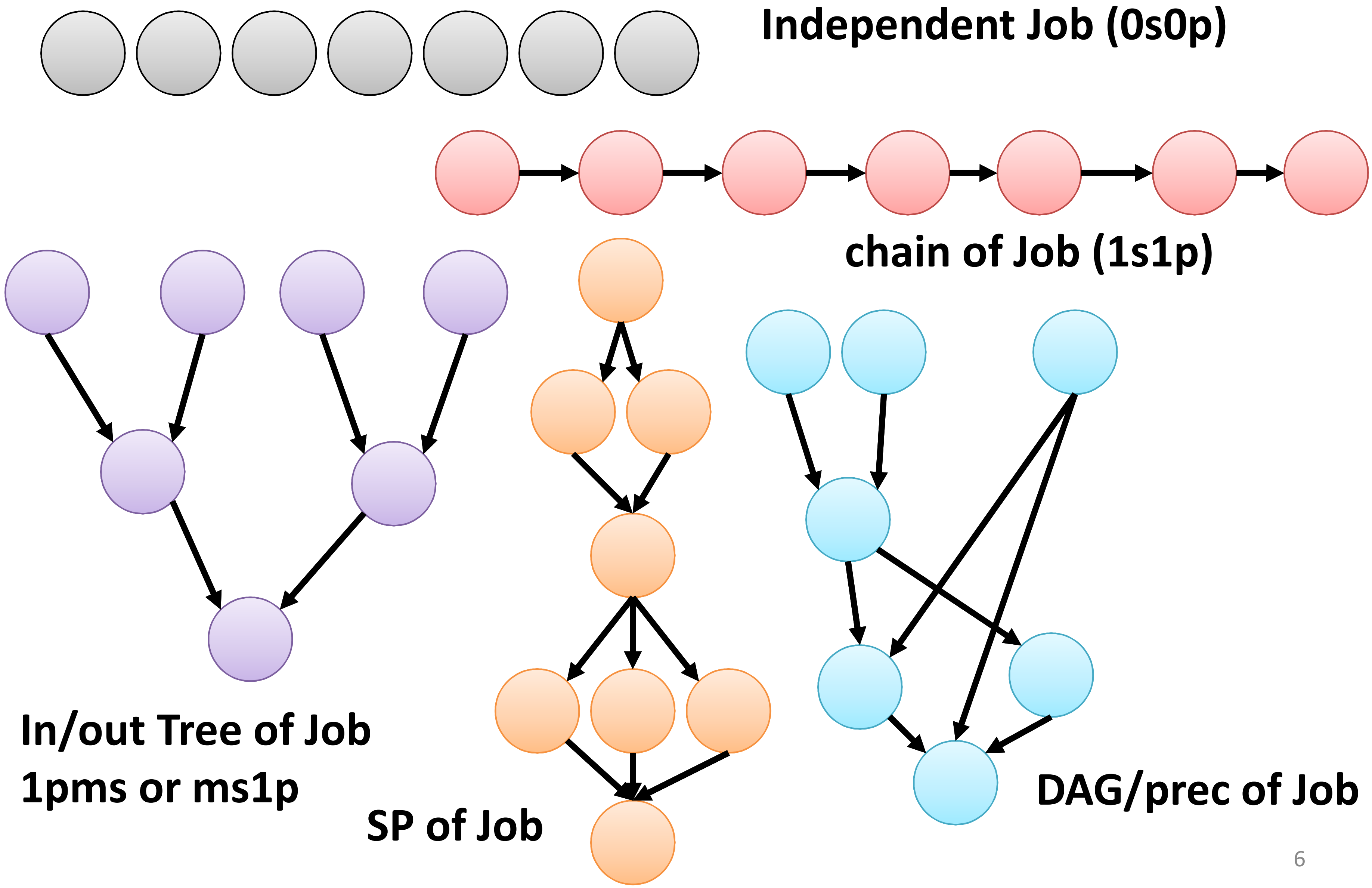
# Parallel Machine Problems

- P: For **identical machines $M_1, \dots, M_m$**
  - The processing time for ***j*** is the same on each machine.
- Q: For **uniform machine**
  - if $p_{jk} = p_j/r_k$.
- R: For **unrelated machines**
  - The processing time $p_{jk}$ depends on the machine $M_k$ on which j is processed.

# Job Characteristics

- **pmtn** preemption
- $r_j$ release times /arrival time
- $d_j$ deadlines
- $p_j = 1$ **or** $p_j = p$ **or** $p_j \in \{1,2\}$ restricted processing times

# Job Precedence Examples



# Objective Functions

Two types of objective functions are most common:

- **bottleneck objective functions** $\max \{f_j(C_j) \mid j = 1, \ldots, n\}$, and
- **sum objective functions** $\sum f_j(C_j) = f_1(C_1) + f_2(C_2) + \ldots \ldots + f_n(C_n)$ .

# Objective Functions

- $\mathbf{C_{max}}$ and $\mathbf{L_{max}}$ symbolize the bottleneck objective functions with
  - $f_j(C_j) = C_j$ (makespan)
  - $f_j(C_j) = C_j - d_j$ (maximum lateness)
- Common sum objective functions are:
  - $\Sigma\, C_j$ (mean flow-time)
  - $\Sigma\, \omega_j\, C_j$ (weighted flow-time)

# Objective Functions

- **Number of Late Job**
  - $\sum U_j$ (number of late jobs) and $\sum \omega_j U_j$ (weighted number of late jobs) where $U_j = 1$ if $C_j > d_j$ and $U_j = 0$ otherwise.
- **Tardiness**
  - $\sum T_j$ (sum of tardiness) and $\sum \omega_j T_j$ (weighted sum of tardiness)
  - Tardiness of job $j$ is given by

$$T_j = \max\{0, C_j - d_j\}.$$

# Examples

- $1 \mid prec; p_j = 1 \mid \Sigma\, \omega_j C_j$
- $P2 \mid \mid C_{max}$
- $P \mid p_j = 1; r_j \mid \Sigma\, \omega_j U_j$
- $R2 \mid chains; pmtn \mid C_{max}$
- $P3 \mid n = 3 \mid C_{max}$
- $Pm \mid p_{ij} = 1; outtree; r_j \mid \Sigma\, C_j$

# Example: $1|C_{max}$

- N independent job without pre-emption
- 1 processor
- Minimize $C_{max}$
- Sol: Schedule in any orders

# Example: $1|\sum C_i$

- N independent job without pre-emption
- 1 processor
- Minimize $\sum C_i$
- Sol: Schedule shortest processing time first
  - SJF is optimal

# Today Class

# Example: $1|\sum w_iC_i$

- N independent job without pre-emption
- 1 processor
- Minimize $\sum w_iC_i$
- Sol:
  - Calculate processing time to weight ratio
  - Rank jobs in increasing order of $p_i/w_i$ and schedule accordingly
  - The Weighted Shortest Processing Time First rule is Optimal for $\mathbf{1|\sum w_iC_i}$

# Example: $1|chain|\sum w_iC_i$

- N independent jobs with chain precedence without pre-emption
- 1 processor, multiple chain
- Minimize $\sum w_iC_i$
- If chain have weight, problem is easy to solve by sol of **$1||\sum w_iC_i$**
- Sol:
  - Calculate processing time to weight ratio (ρ) of chains (by including a number of tasks from a chains)
  - Process the tasks from chain till the ρ of the chain is higher than others chain

# Example: $1|prec|\sum w_iC_i$

- For general precedence the problem is Hard
- NP-Complete problem

# P3 | ptmn | $C_{max}$

- 3 Identical machine, Independent Jobs, release time ri=0, $C_{max}$
- Solvable in Polynomial time
- Suppose N tasks with execution time $t_i$, 3 processor
- $C_{max} = (\sum t_i)/3$
- Distribute $C_{max}$ unit amount task to each processor in any order

# $P3 | ptmn | C_{max}$

- M1, M2, M3
- 10 tasks: 5,6,10,4,3,8,6,3,7,12
- $C_{max}$= (5+6+10+4+3+8+6+3+7+12)/3=64/3=21+1/3
- Assign 20+1/3 unit time of tasks in any ways

  12+6+3 +1/3

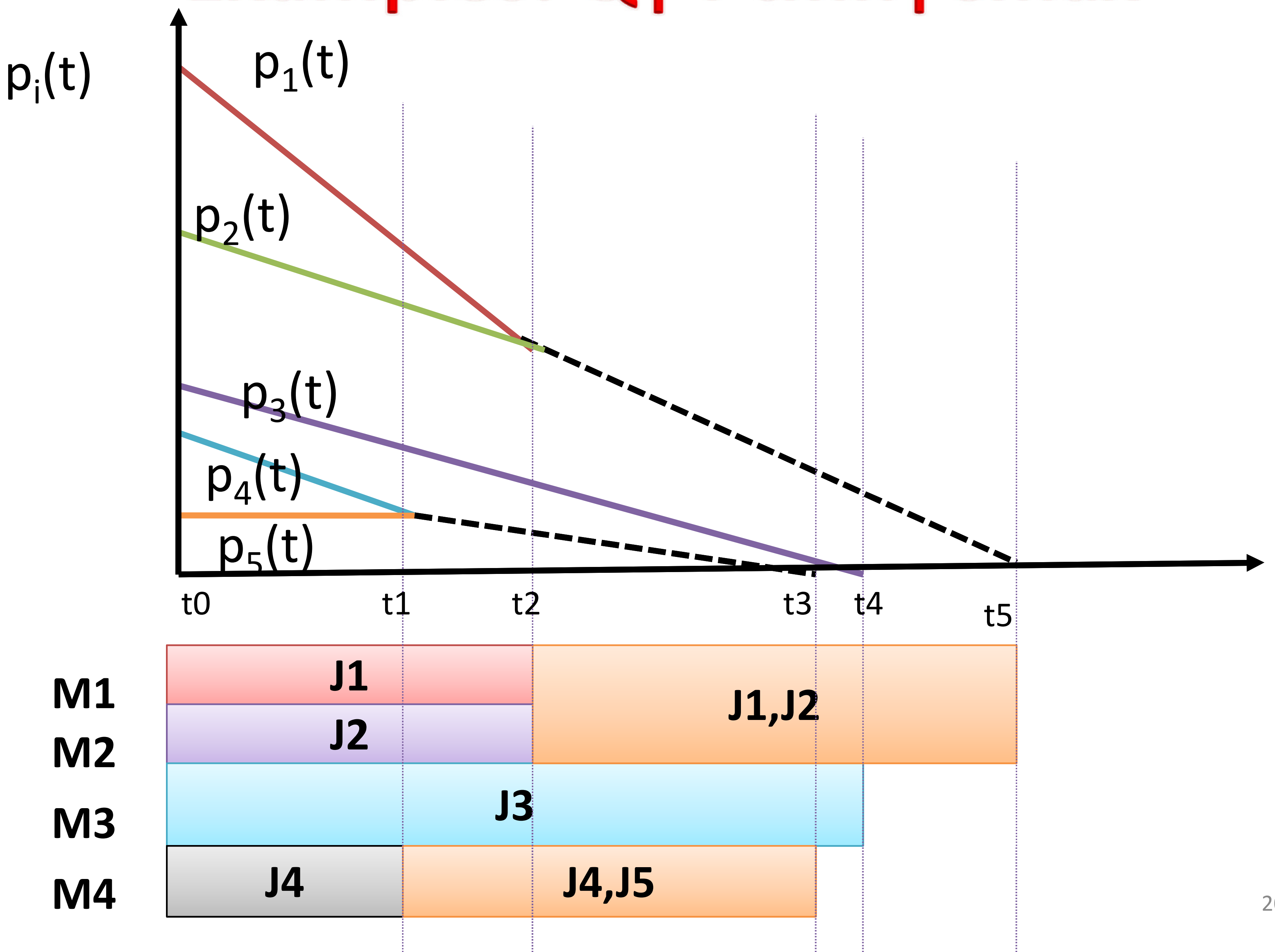
  10+7+4 +1/3

  8+5+6+ 2 +1/3

# $Q|ptmn|C_{max}$

- Suppose 5 Job p1, p2, p3, p4 and p5
  - With P1 is longest and P5 is shortest
  - P1>p2>p3>p4>p5
- 4 machine M1 > M2 > M3 > M4. M1 is fastest and M4 is slowest
- Each job have level : based on how time left before finish (un-finished part of job).
- Fist try to finish the highest level job on the fastest machine.
- When level of two job are same jointly process on the machine

# Examples: Q| Ptmn|Cmax



# $Q|ptmn|C_{max}$

**Algorithm level**

1. *t := 0;*
2. WHILE there exist jobs with positive level DO {
3. Assign(*t);*
4. *t1 := min{s > t | a job completes at time s};*
5. *t2 := min{s > t | there are jobs i, j with pi(t) > pj(t) and pi(s) = pj(s)};*
6. *t := min{t1, t2}*
7. Construct the schedule.

# $Q|ptmn|C_{max}$

**Assign (t) {**

1. *J := {i | pi(t) > 0};*
2. *M := {M1, . . .,Mm};*
3. WHILE *J = Ø and M = Ø DO {*
4. Find the set *I ⊆ J of jobs with highest level;*
5. *r := min{|M|, |I|};*
6. Assign jobs in *I to be processed jointly on the r fastest* machines in *M;*
7. *J := J\I;*
8. Eliminate the *r fastest machines in M from M*

# $P_m || C_{max}$

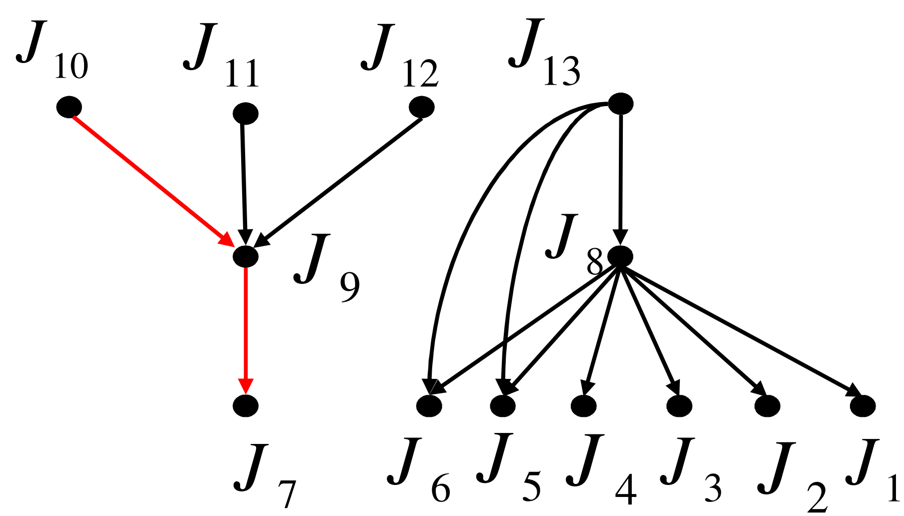
- 2 || Cmax can be easily mapped to 2 set partition problem
  - Pseudo Polynomial Time algorithm
- m≥3, m=3 mapped to 3 partition problem
  - NP Complete Problem
- Approximation Algorithms
  - Grahams  List scheduling
  - Longest Task First

# Precedence constraints (*prec*)

Before certain jobs are allowed to start processing, one or more jobs first have to be completed.

Definition

- Successor
- Predecessor
- Immediate successor
- Immediate predecessor
- Transitive Reduction



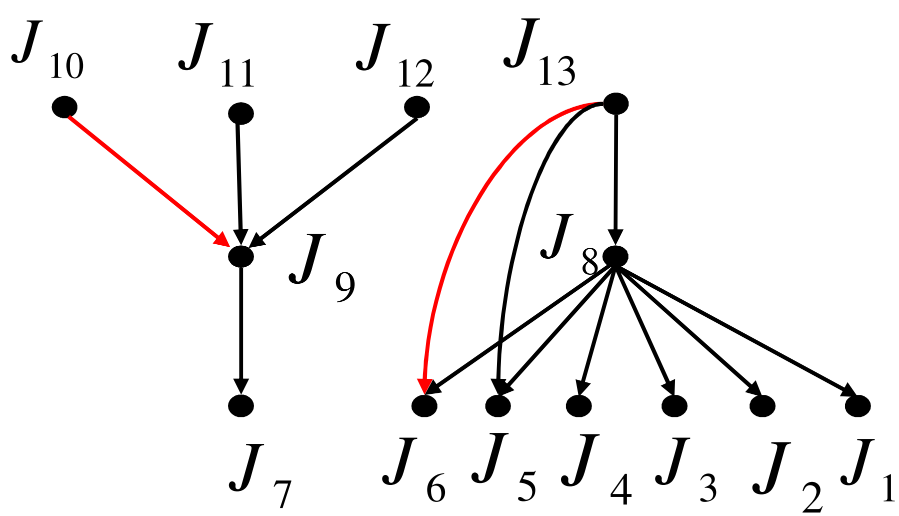

$$p(J_i) = 1$$

# Precedence constraints (*prec*)

One or more job have to be completed before another job is allowed to start processing.

Definition

- Successor
- Predecessor
- Immediate successor
- Immediate predecessor
- Transitive Reduction



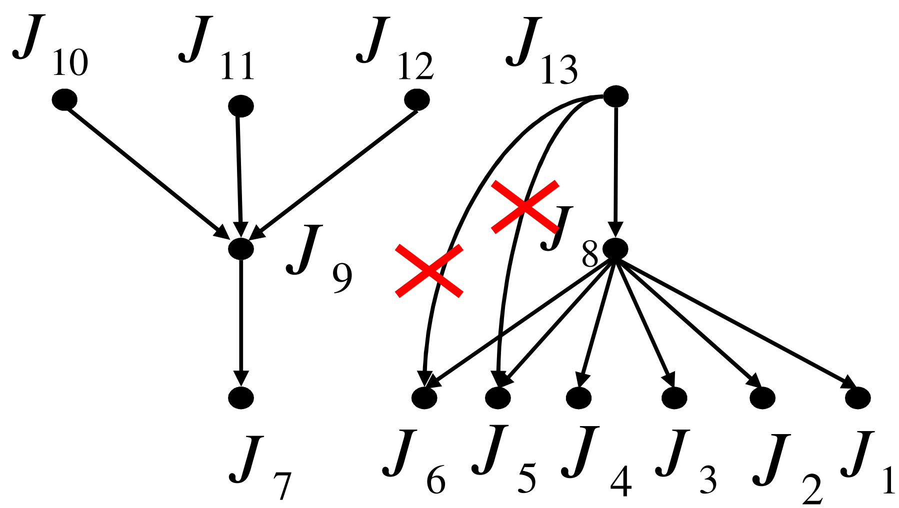

$$p(J_i) = 1$$

# Precedence constraints (*prec*)

One or more job have to be completed before another job is allowed to start processing. *Prec : Arbitrary acyclic graph*

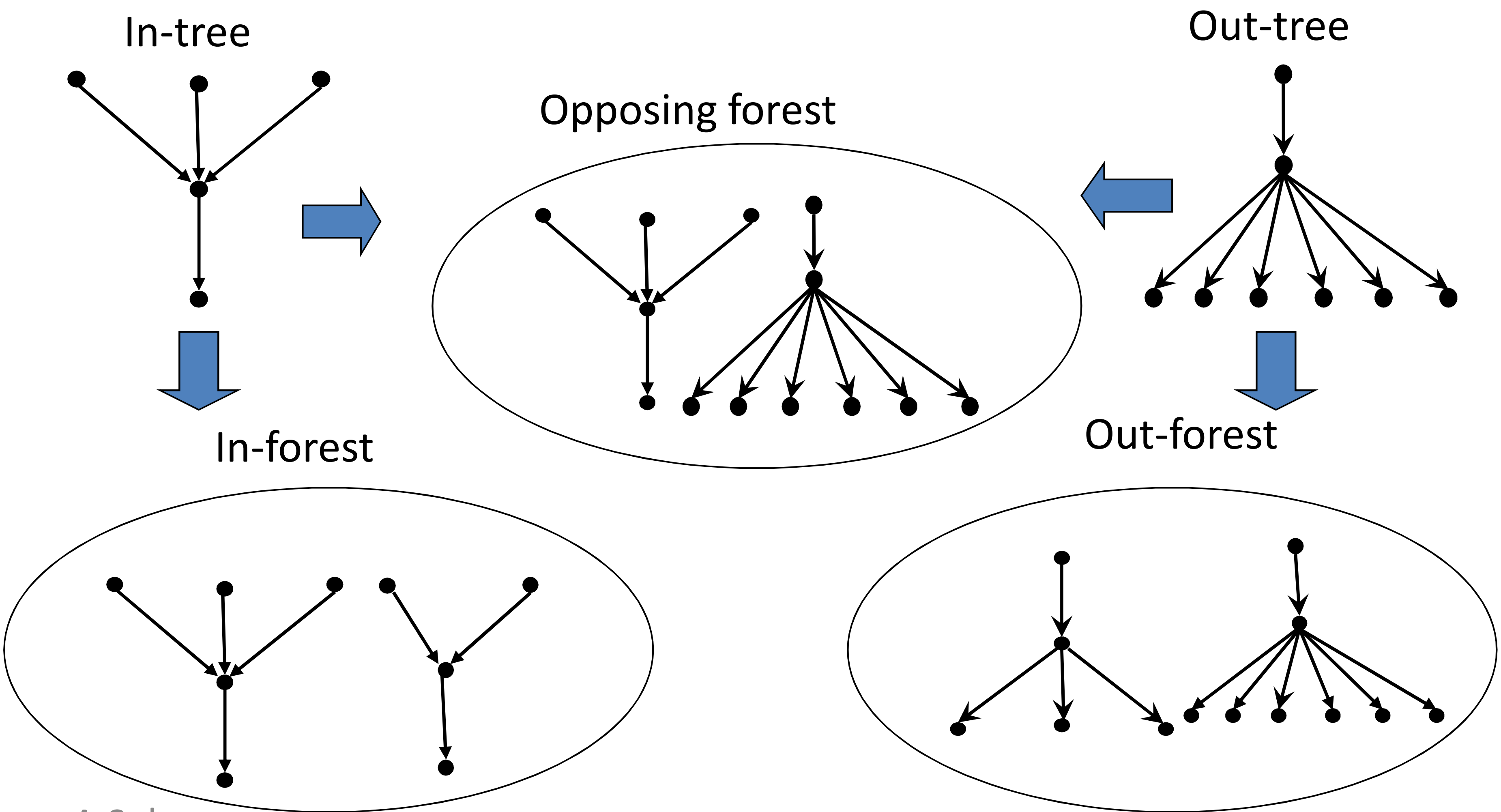
Definition

- Successor
- Predecessor
- Immediate successor
- Immediate predecessor
- Transitive Reduction




$$p(J_i) = 1$$

# Special precedence constraints




A Sahu

# Approximation for: $Pm|prec,pi=1|C_{max}$ and $Pm||C_{max}$

- **CP Algorithms: Introduction to Algorithms, Corman Leisserson Rivest (CLR) , 3rd Ed, Page 779-783**
- ***Algorithm Design, Eva Tardos, Page 600-605,***

# Complexity

| | PMTN, pi=1 | Pi=1, NO PMTN | Pi PMTN | Pi NO-PMTN |
|---|---|---|---|---|
| Tree | **YES** | **YES** | **YES** | NO |
| Series-Parallel | | | | NO |
| Opposing Forest | | | | NO |
| DAG | NO | NO | ?Takehome | NO |

# $Pm \mid prec, p_j = 1 \mid C_{max}$

- Processor Environment
  - m identical processors are in the system.
- Job characteristics
  - Precedence constraints : precedence graph;
  - Preemption is not allowed;
  - The release time of all the jobs is 0.
- Objective function
  - $C_{max}$ : the time the last job finishes execution.
  - If $c_j$ denotes the finishing time of $J_j$ in a schedule S,

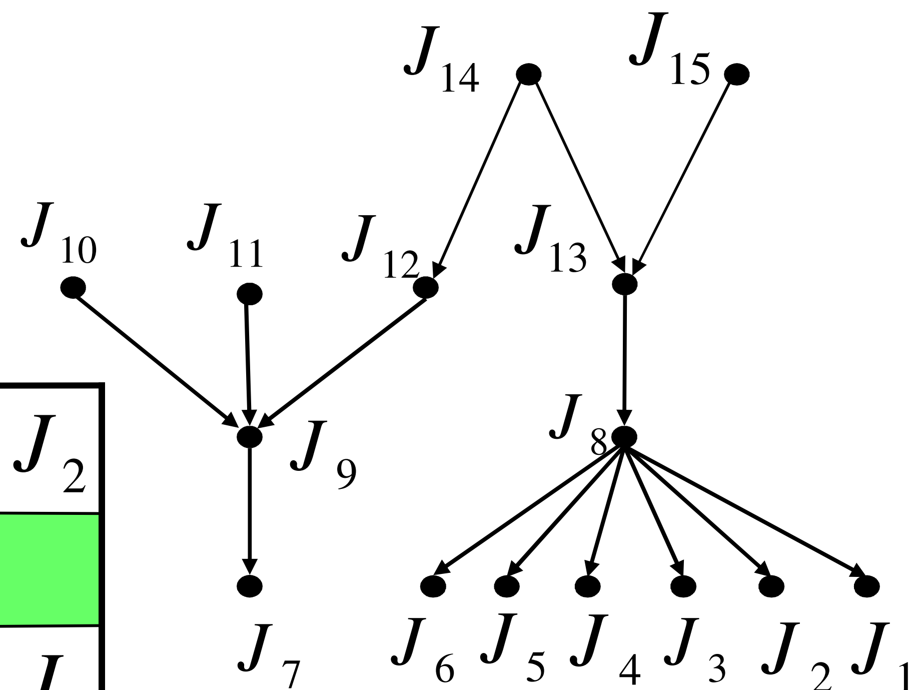
$$C_{max} = max_{1 \leq j \leq n}\, c_j$$

# Example: Gantt Chart

A Gantt chart indicates the time each job spends in execution, as well as the processor on which it executes ~~of some Schedule~~

| | Slot 1 | Slot 2 | | | | | |
|---|---|---|---|---|---|---|---|
| P1 | | | $J_{15}$ | $J_{13}$ | $J_9$ | $J_5$ | $J_2$ |
| P2 | | $J_{14}$ | $J_{12}$ | $J_8$ | $J_7$ | $J_4$ | |
| P3 | $J_{10}$ | $J_{11}$ | | | $J_6$ | $J_3$ | $J_1$ |

Time axis




$$p(J_i) = 1$$

# $Pm \mid prec, p_j = 1 \mid C_{max}$

**Theorem 1**

$Pm \mid prec, p_j = 1 \mid C_{max}$ is NP-complete.

1. Ullman (1976)

   $3SAT \leq Pm \mid prec, p_j = 1 \mid C_{max}$

2. Lenstra and Rinooy Kan (1978)

   $k\text{-clique} \leq Pm \mid prec, p_j = 1 \mid C_{max}$

**Corollary 1.1**

**The problem of determining the existence of a schedule with $C_{max} \leq 3$ for the problem $Pm \mid prec, pj = 1 \mid C_{max}$ is NP-complete.**

***Proof: out of Syllabus***

# $Pm \mid prec, p_j = 1 \mid C_{max}$

Mayr (1985)

- **Theorem 2**

$Pm \mid p_j = 1, SP \mid C_{max}$ is NP-complete.

SP: Series - parallel

- **Theorem 3**

$Pm \mid p_j = 1, OF \mid C_{max}$ is NP-complete.

OF: Opposing - forest

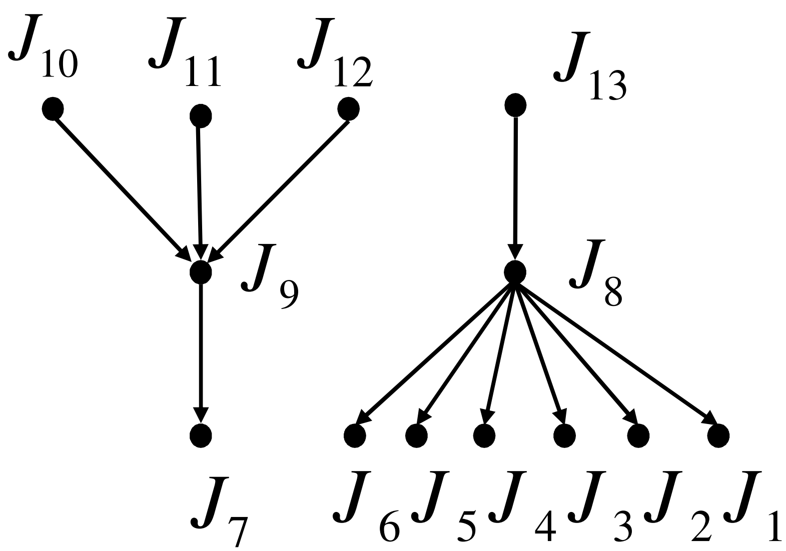
***Proof: out of Syllabus***

# PTAS : Pm | prec, $p_j = 1$ | $C_{max}$

- PTAS : Polynomial Time Approximation Scheme
- Approximation List scheduling policies
  - Graham's list algorithm
  - HLF algorithm
  - MSF algorithm

# List scheduling policies

- Set up a priority list L of jobs.
- When a processor is idle, assign the first ready job to the processor and remove it from the list L.




| | | | | |
|---|---|---|---|---|
| $J_{11}$ | $J_9$ | $J_8$ | $J_6$ | $J_3$ |
| $J_{10}$ | $J_{13}$ | $J_7$ | $J_5$ | $J_2$ |
| $J_{12}$ | | | $J_4$ | $J_1$ |

$$L=(J_9,J_8,J_7,J_6,J_5,J_{11},J_{10},J_{12},J_{13},J_4,J_3,J_2,J_1)$$

# Graham's list algorithm

- Graham first analyzed the performance of the simplest list scheduling algorithm.
- List scheduling algorithm with an arbitrary job list is called Graham's list algorithm.
- Approximation ratio for

$$P_m \mid prec, p_j = 1 \mid C_{max}$$

App Ratio $\delta = 2 - 1/m$

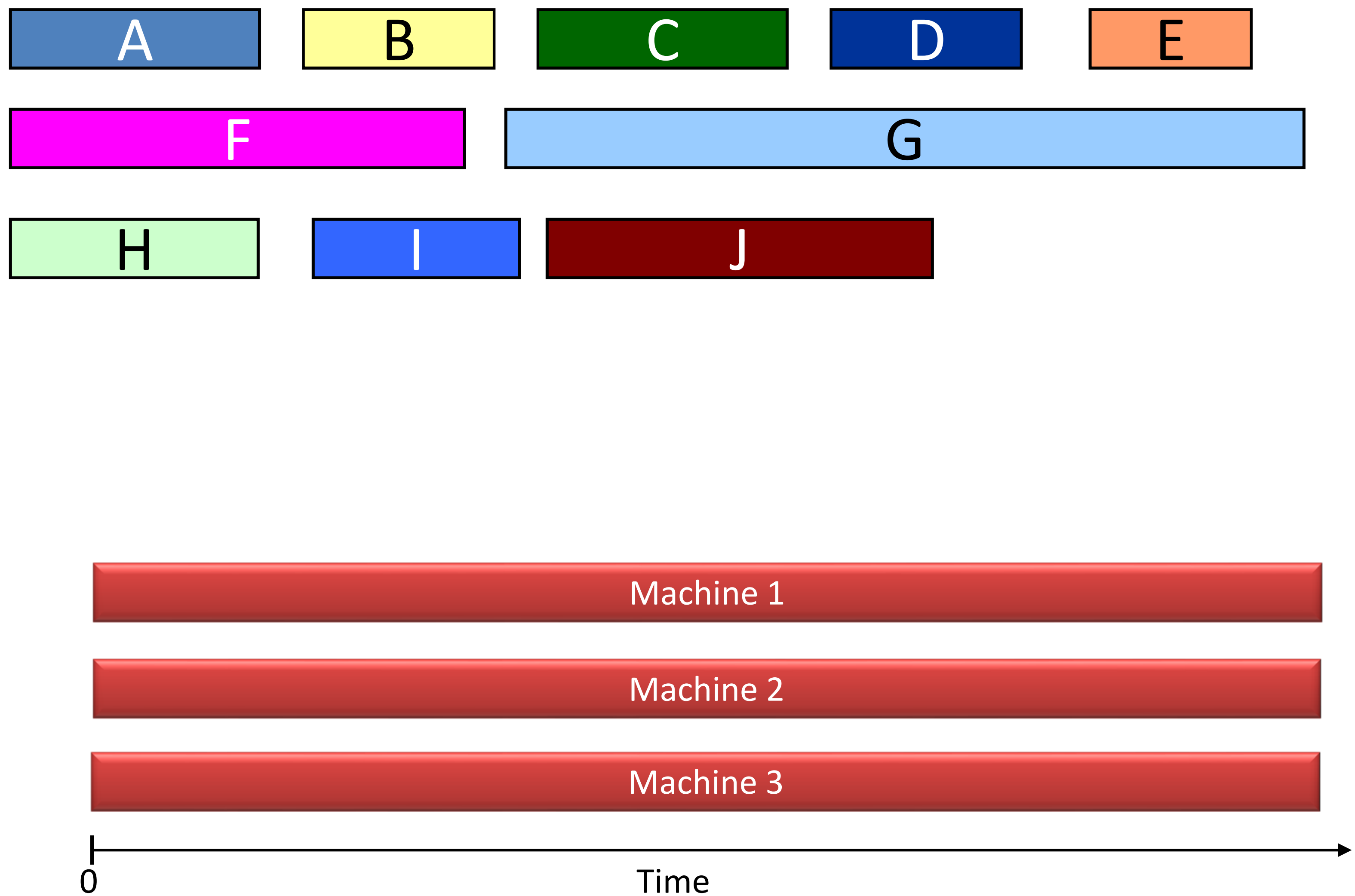
# Pm| | Cmax
# Minimum makespan scheduling

- Given
  - Processing times for n jobs $p_1, p_2, \ldots, p_n$
  - And an integer m (number of processor )
- Find an assignment of the jobs to m identical machines
- So that the completion time, also called the makespan, is minimized.

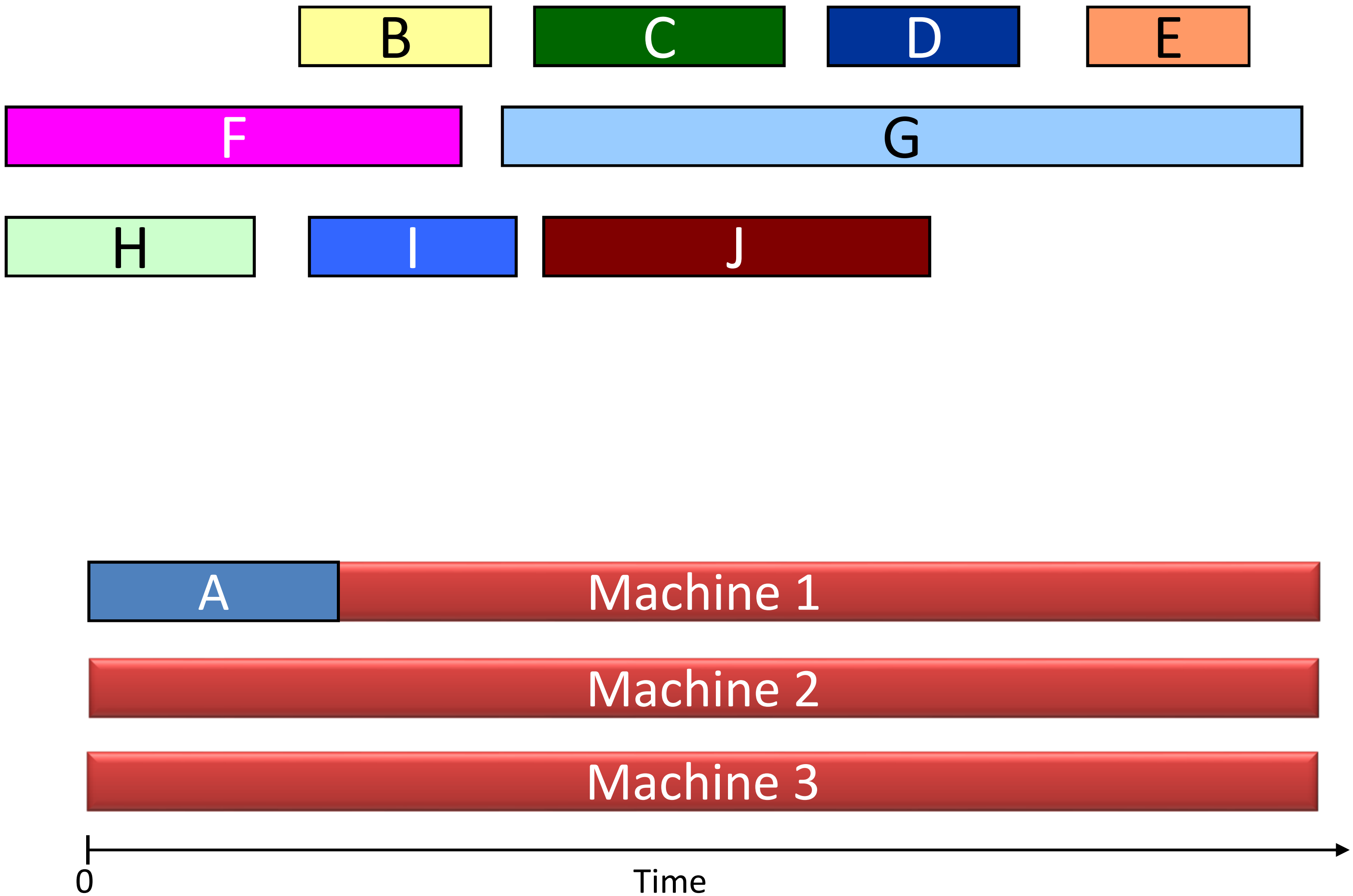
# Minimum Makespan scheduling: Arbitrary List

- Algorithm
  - 1. Order the jobs arbitrarily.
  - 2. Schedule jobs on machines in this order, scheduling the next job on the machine that has been assigned the least amount of work so far.
- Achieves an approximation guarantee of 2

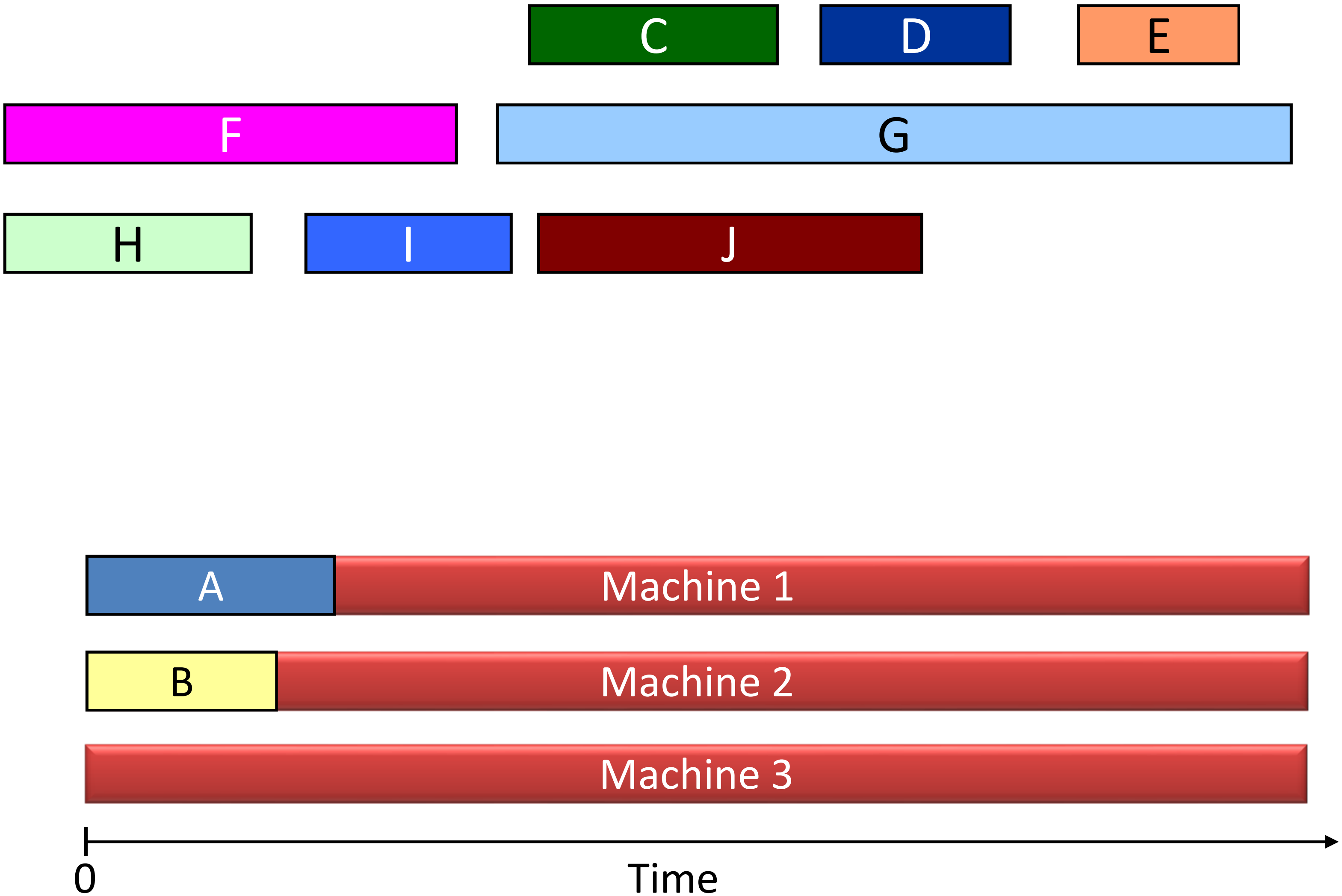
# Load Balancing: List Scheduling



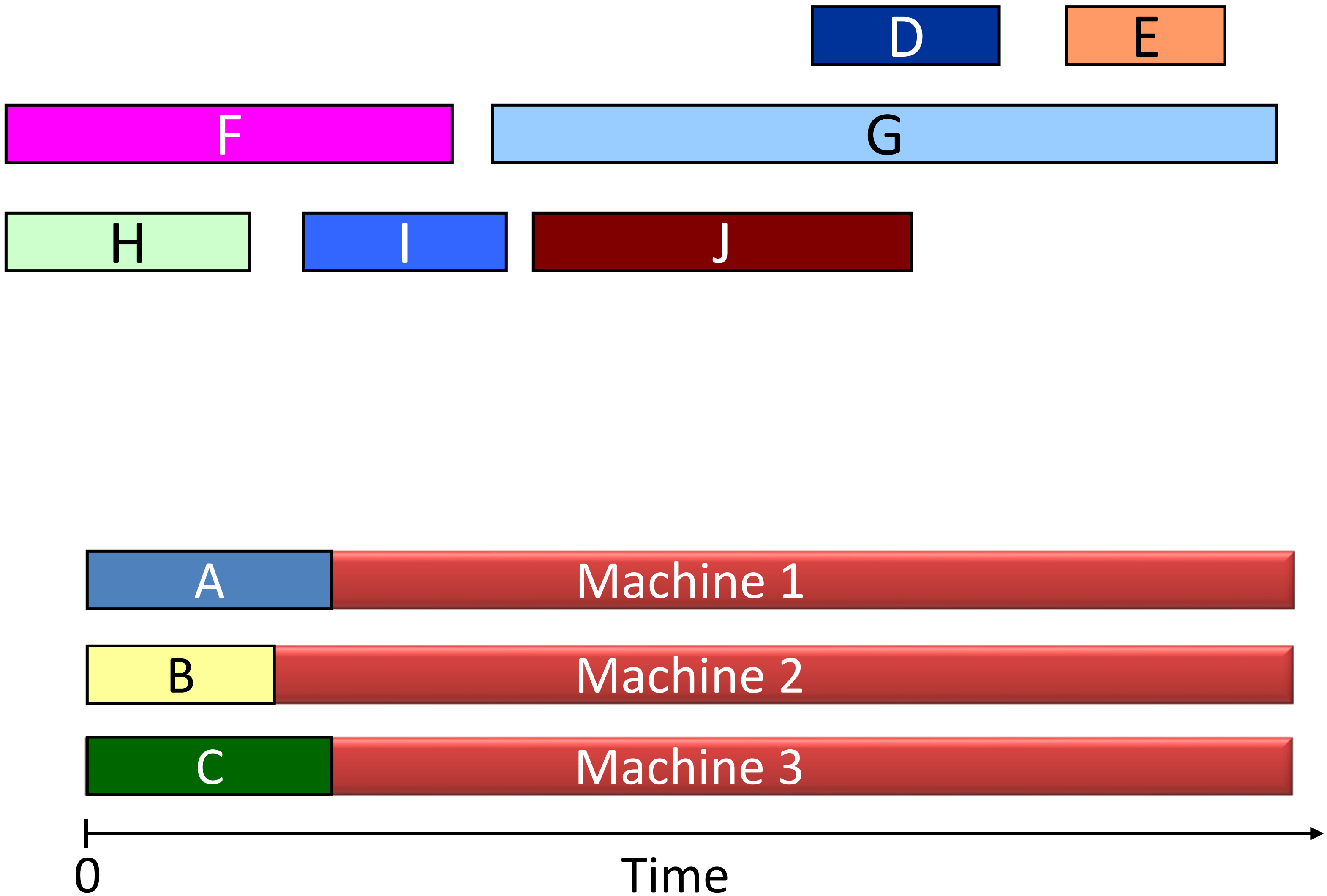
# Load Balancing: List Scheduling



# Load Balancing: List Scheduling

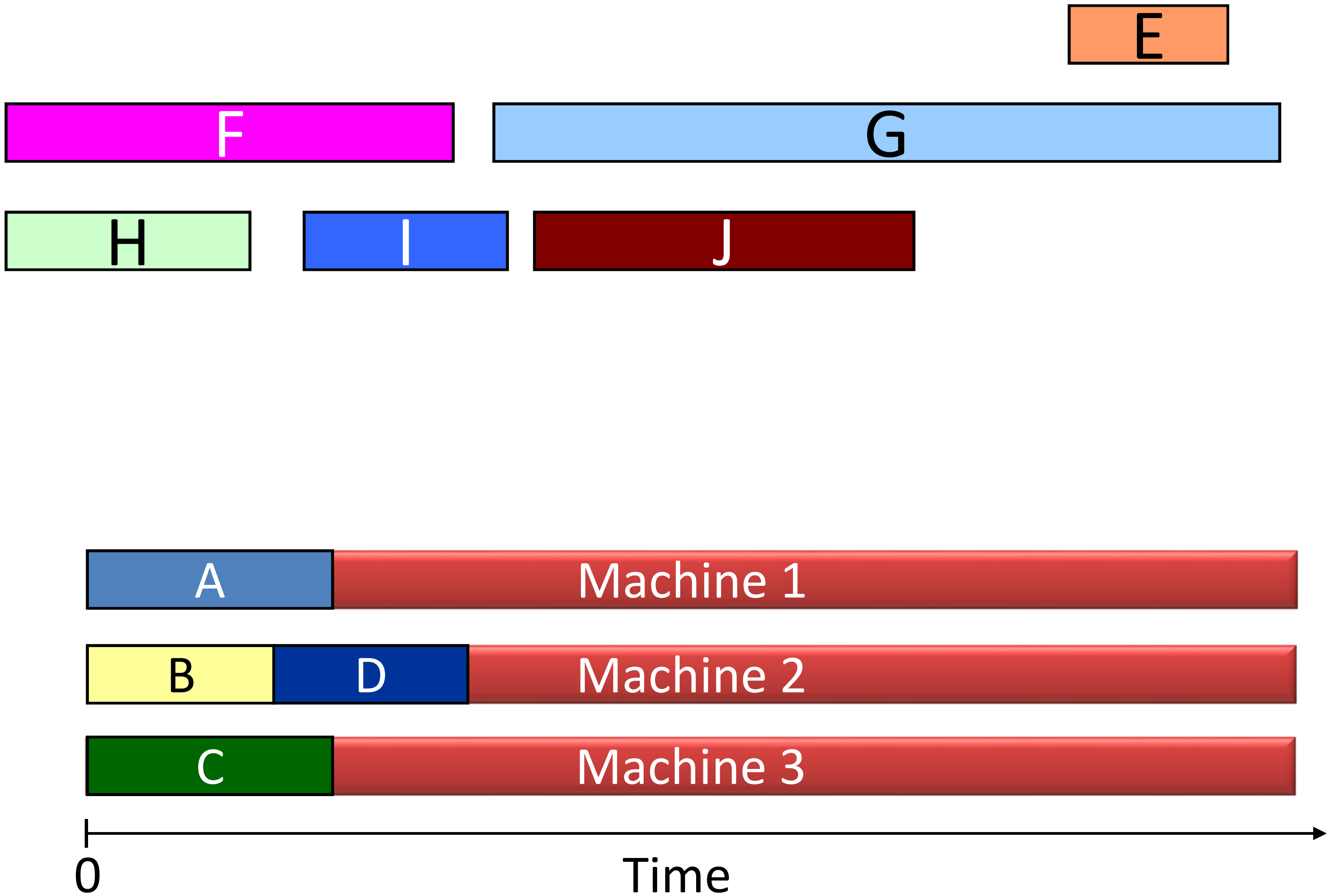


# Load Balancing:  List Scheduling

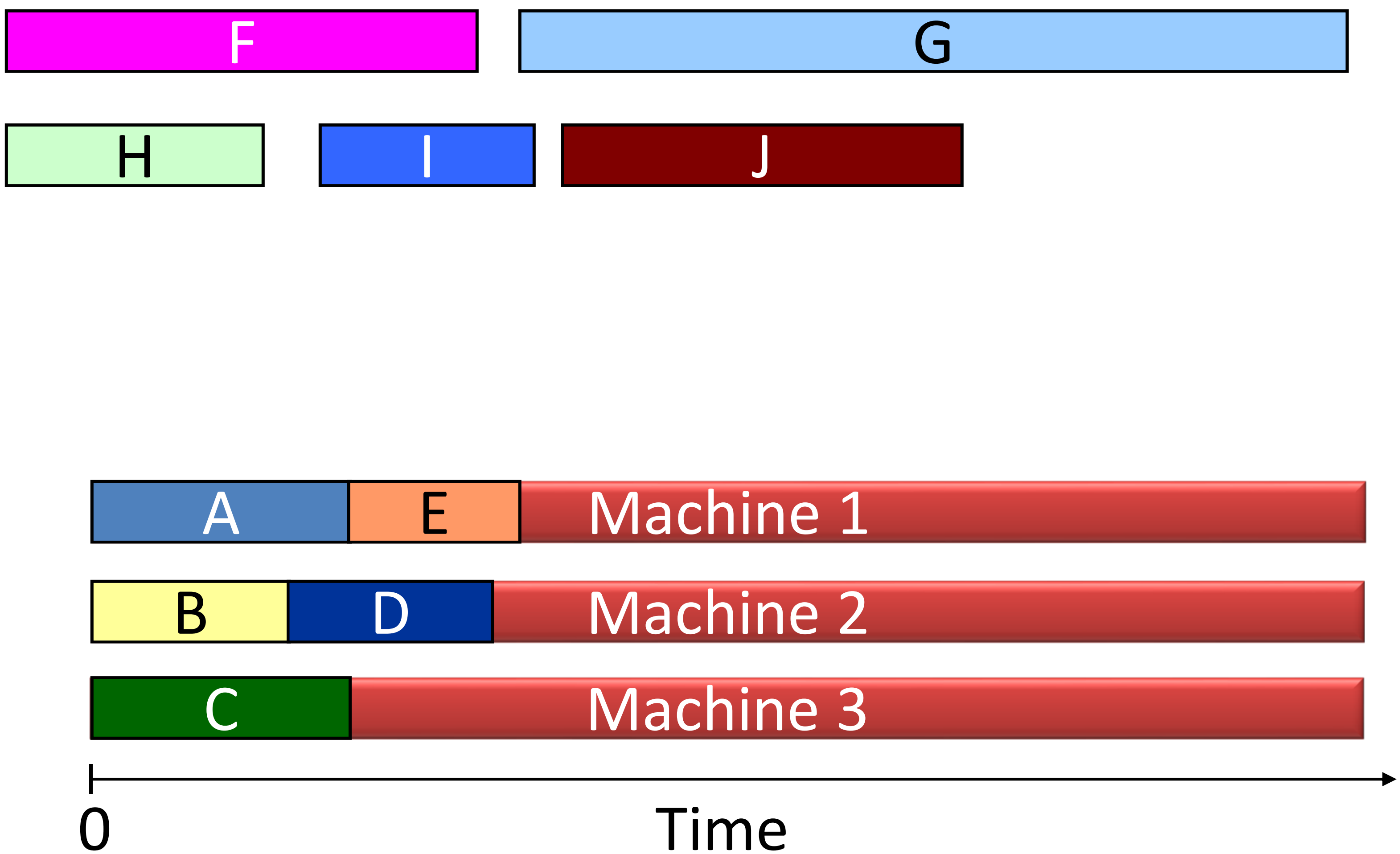



A Sahu

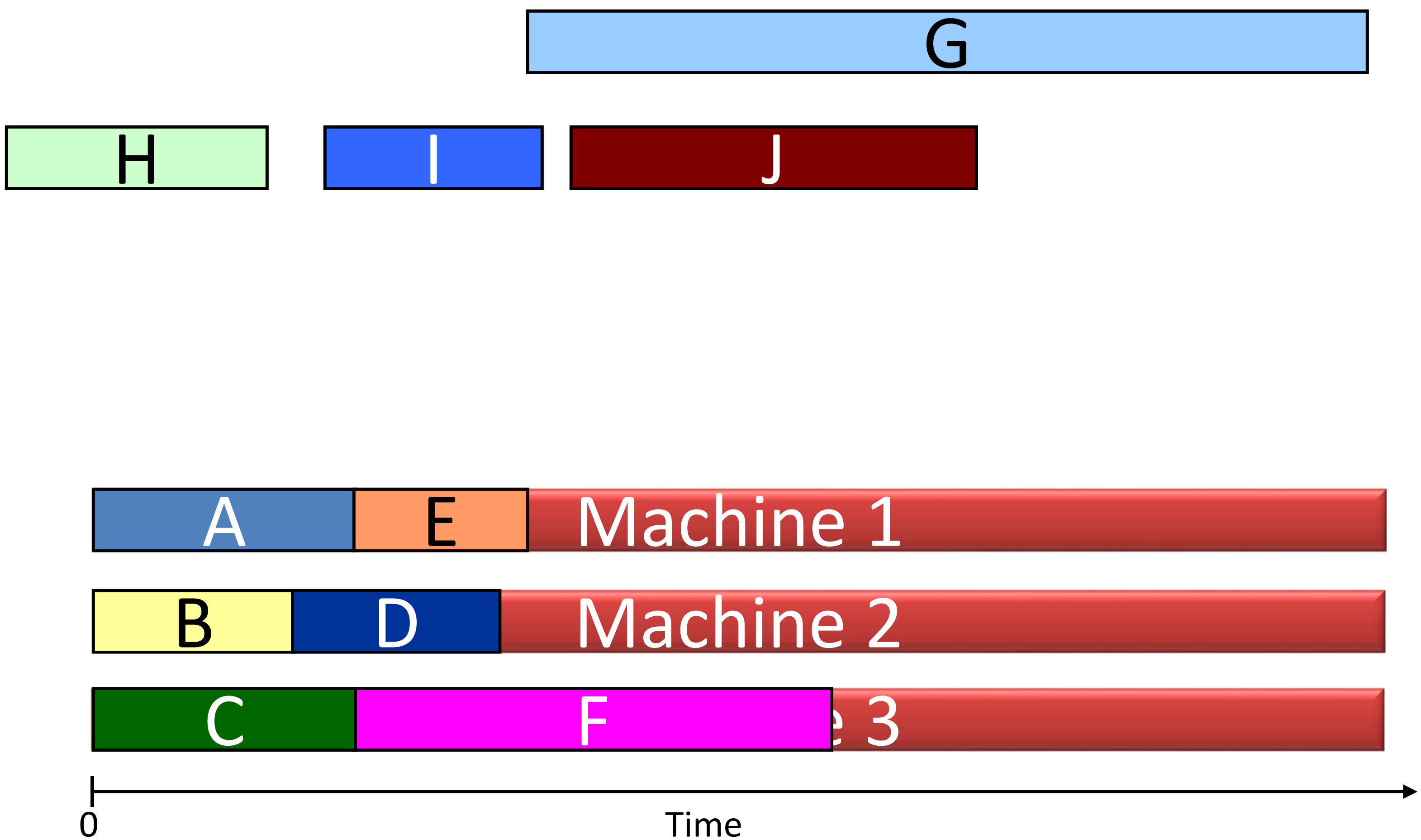
# Load Balancing:  List Scheduling



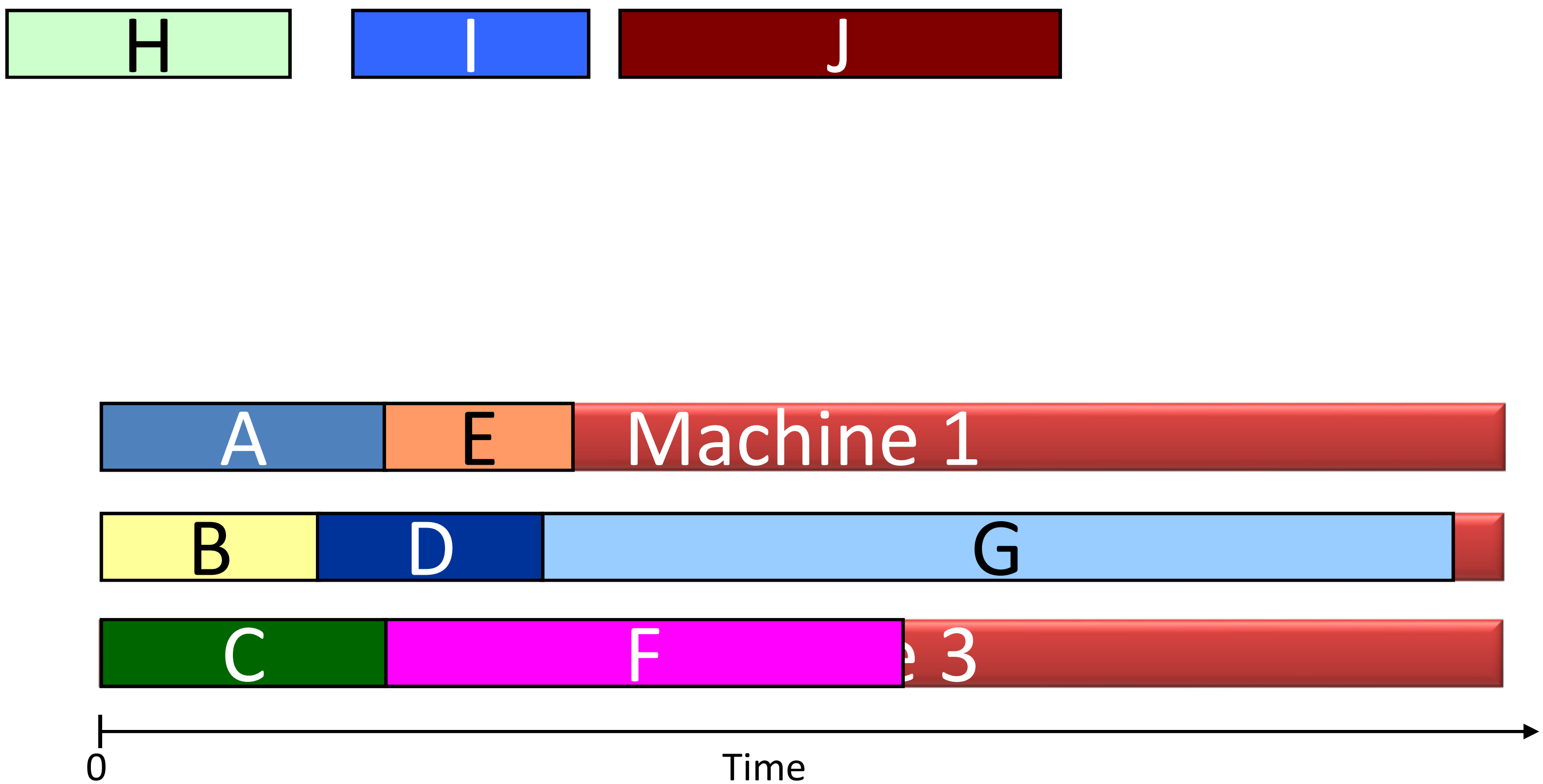
# Load Balancing: List Scheduling



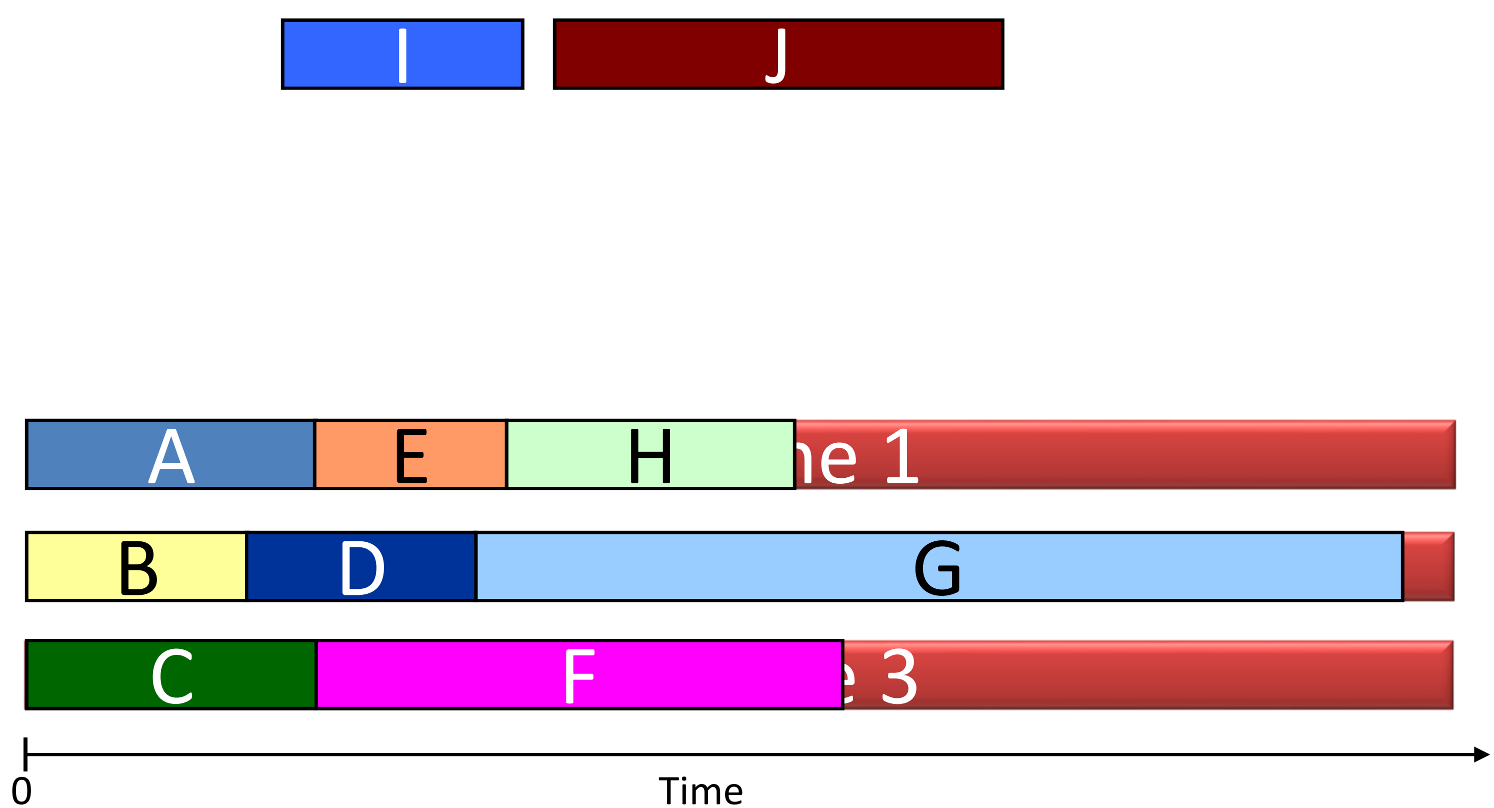
# Load Balancing: List Scheduling



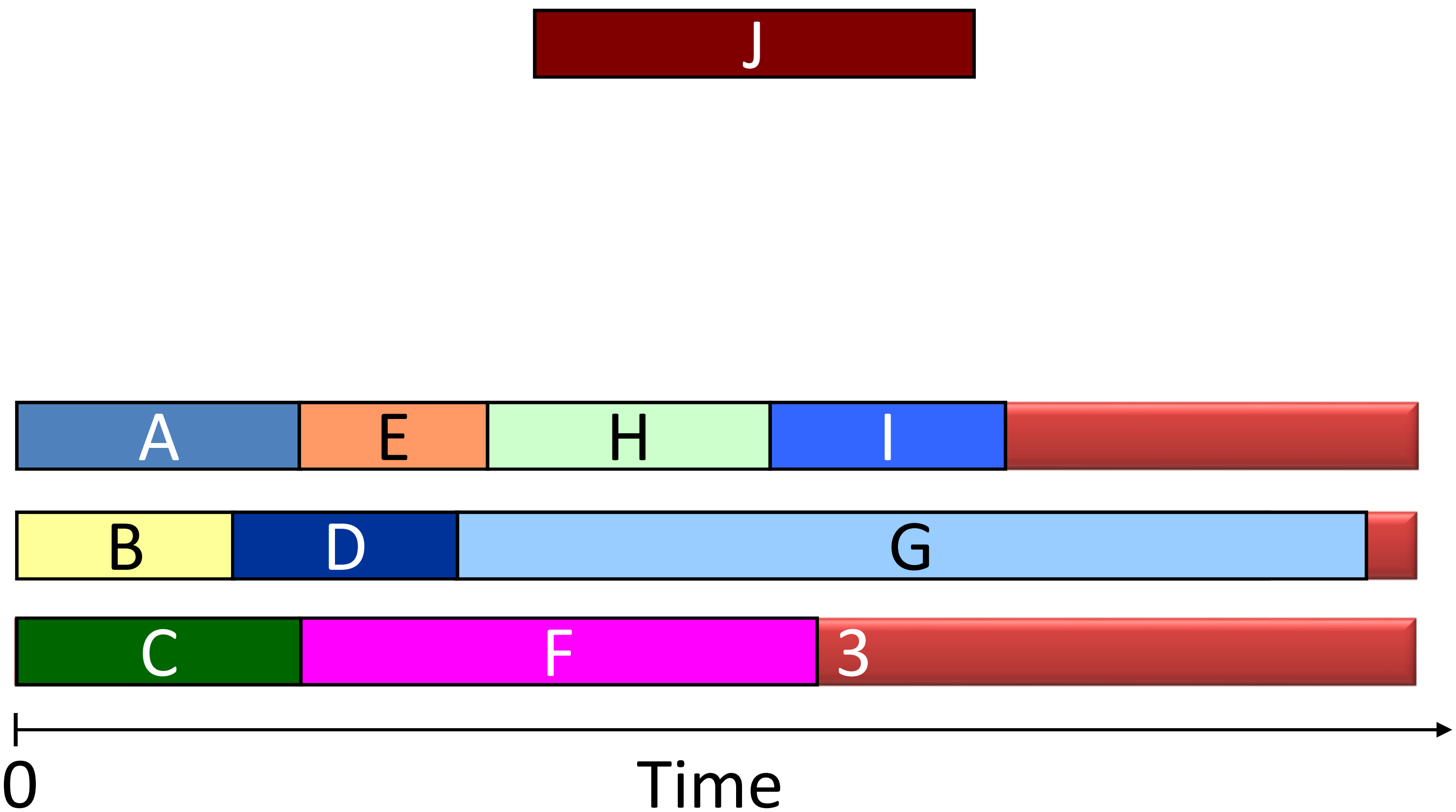
# Load Balancing:  List Scheduling



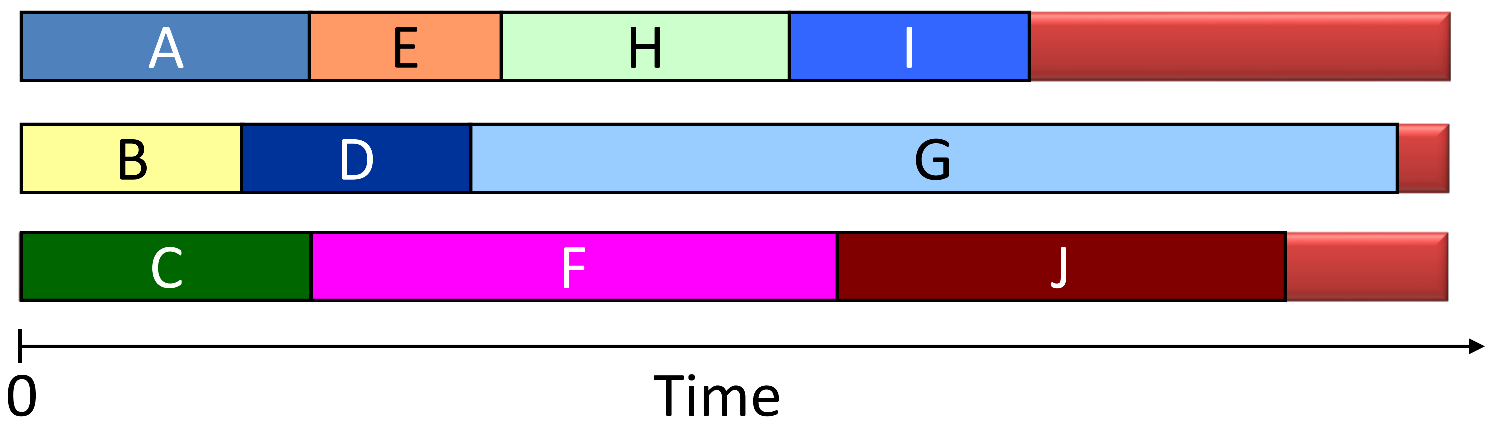
# Load Balancing: List Scheduling



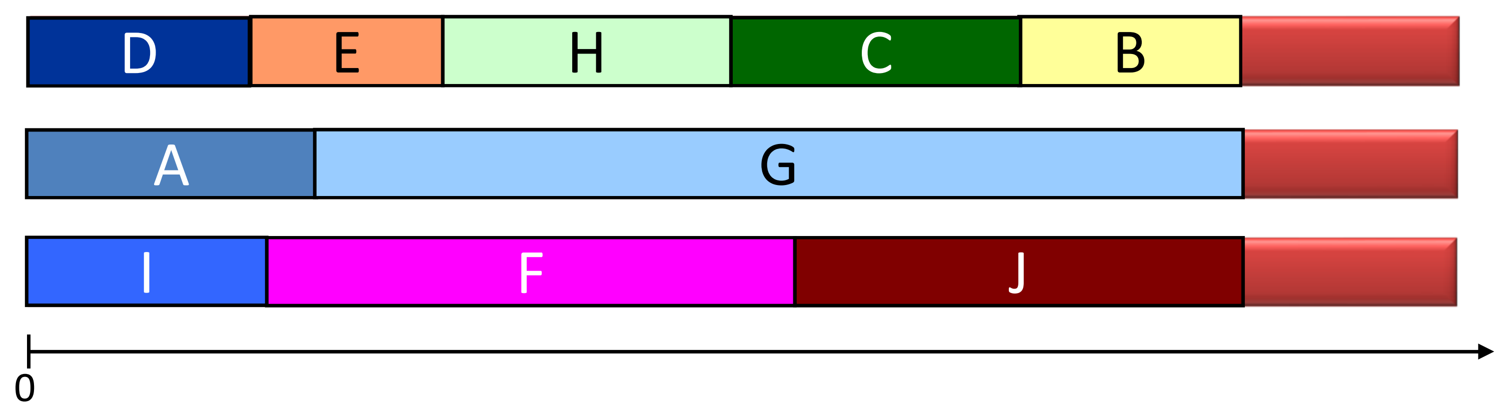
# Load Balancing: List Scheduling



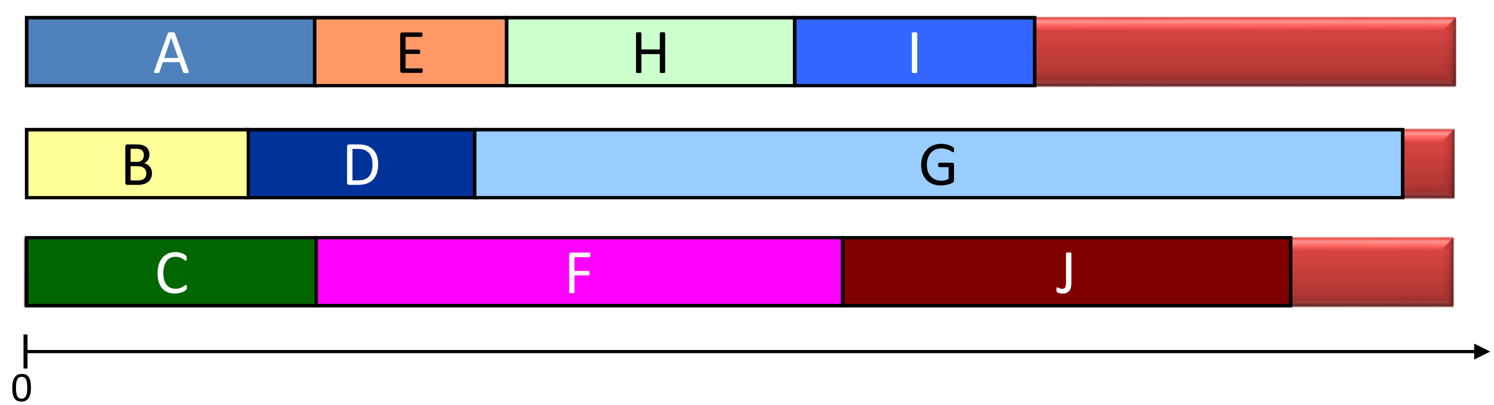
# Load Balancing:  List Scheduling



# Load Balancing: List Scheduling




Optimal Schedule



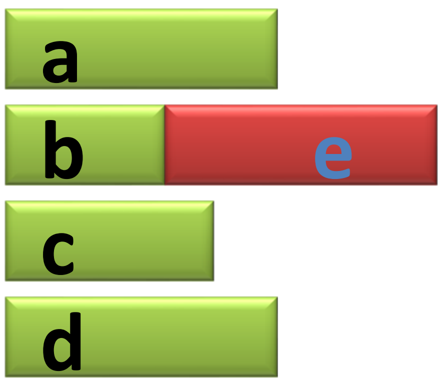

List schedule

# List Scheduling is “2-approximation” (Graham, 1966)

**Algorithm: List scheduling**

**Basic idea: In a list of jobs,**

**schedule the next one as soon as a machine is free**




**machine 1**

**machine 2**

**machine 3**

**machine 4**

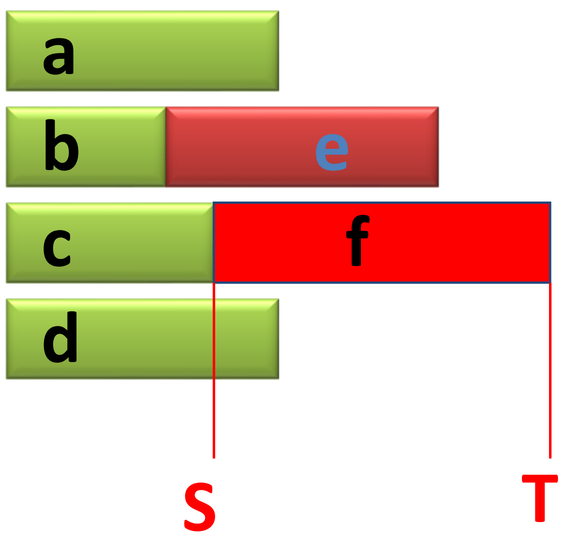
**Good or bad ?**

# List Scheduling is "2-approximation" (Graham, 1966)

**Algorithm: List scheduling**

**Basic idea: In a list of jobs,**

**schedule the next one as soon as a machine is free**




**machine 1**

**machine 2**

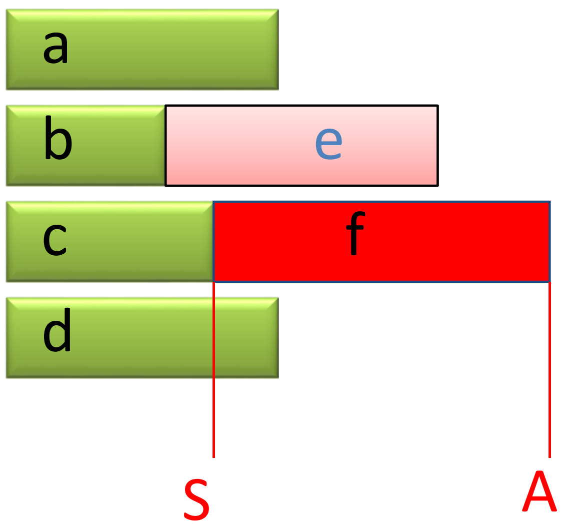
**machine 3**

**machine 4**

**job f finishes last, at time T**

**compare to time OPT of best schedule: how ?**

# List Scheduling is "2-approximation"




machine 1

machine 2

machine 3

machine 4

job f finishes last, at time A

compare to time OPT of best schedule: how ?

(1) job f must be scheduled in the best schedule at some time:
**f <= OPT.** ➔ **A – S <= OPT.**

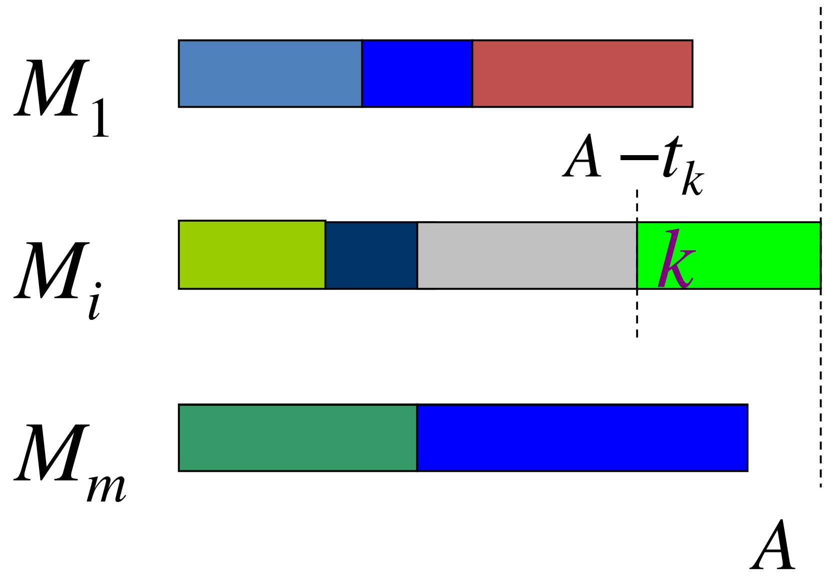
(2) up to time S, all machines were busy all the time, and OPT cannot beat that, and job f was not yet included: **S < OPT.**

(3) both together: **A = A – S + S = (A-S) +S < 2*OPT.**

"2-approximation" (Graham, 1966)

# LS achieves a perf. ratio 2–1/*m*.

So all machines are busy from time 0 through $A\text{-}t_k$
Consequently,




Let $\boldsymbol{T} = \sum \boldsymbol{t_i}$, $i$=1,2…,$n$

$$T\text{-}t_k \geq m(A - t_k) \quad \rightarrow \; T\text{-}t_k \geq \; mA - mt_k$$

$$\rightarrow \quad T\text{-}t_k + mt_k \geq mA \;\; \rightarrow \;\; T + (m\text{-}1)t_k \geq mA$$

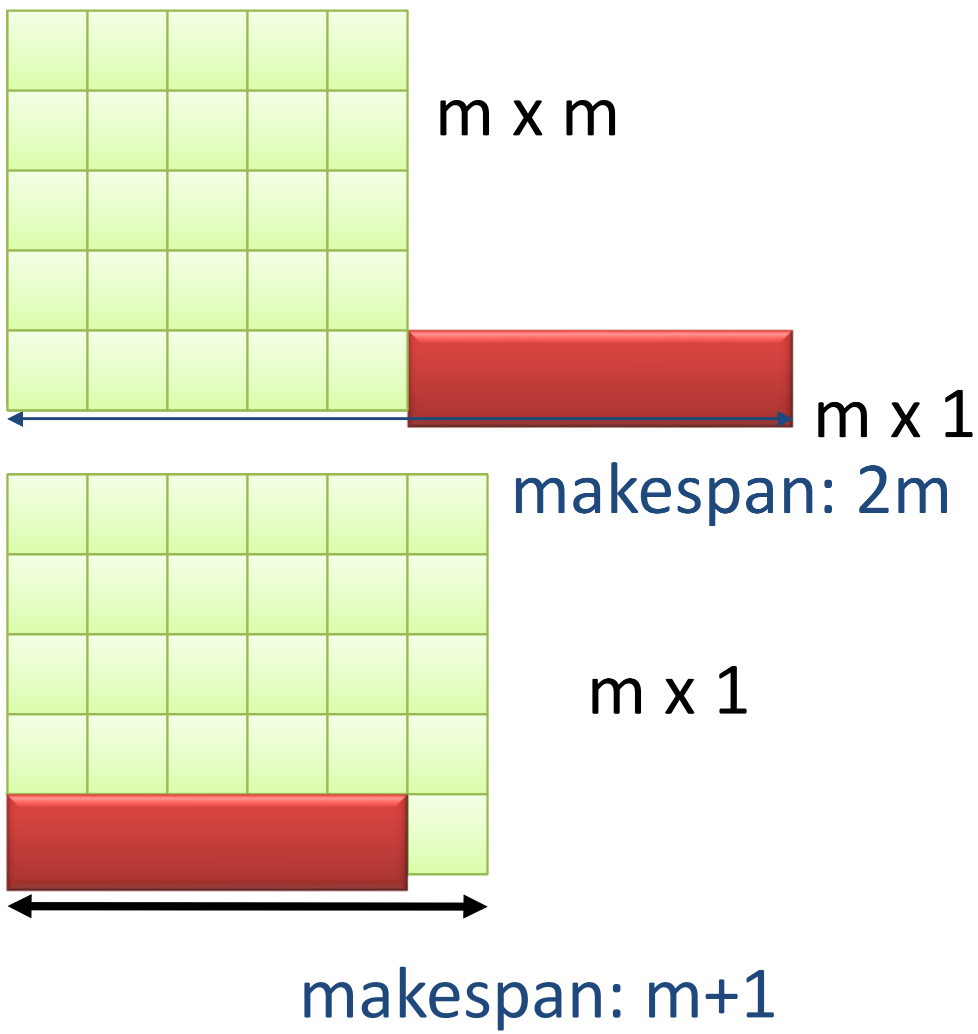
So, $$A \leq T/m + t_k\,(m\text{-}1)/m$$

$$\leq \; T^* + (1\text{-}1/m)\; T^*$$

**$A \leq (2\text{-}1/m)\; T^*$**

**As *m*. T* ≥ T. So,**
**T*≥*T*/*m*.**
**Also T*≥$t_k$ for every *k*.**

# Example: Worst Case



# List with LPT

- List scheduling can do badly if long jobs at the end of the list spoil an even division of processing times.
- We now assume that the jobs are all given ahead of time, i.e. the LPT rule works only in the off-line situation. Consider the "***Largest Processing Time first***" or LPT rule that works as follows.

# List with LPT

LPT($I$)

1 sort the jobs in order of decreasing processing times: $t_1 \geq t_2 \geq . . . \geq t_n$

2 execute list scheduling on the sorted list

3 **return** the schedule so obtained.

**Prove out of Syllabus**

- The LPT rule achieves a performance ratio **3/2**
  - ***Page 605, Algorithms Design Eva Tardos***
- The LPT rule achieves a performance ratio **4/3- 1/(3*m*)**. **Ref: Grahams 1969**