CS528
Task Scheduling
(Part II)

A Sahu
Dept of CSE, IIT Guwahati

Parallel Machines

Ti	P1	P2	P3	P4
T1	10	10	10	10
T2	12	12	12	12
T3	16	16	16	16
T4	20	20	20	20

P: Identical

Ti	P1	P2	P3	P4
T1	10	15	20	25
T2	12	18	24	30
T3	16	24	32	40
T4	20	30	40	50

Q: Uniform : with speed difference (S_1=1, S_2=2/3, S_3=1/2, S_4=2/5

Ti	P1	P2	P3	P4
T1	10	8	12	2
T2	12	28	25	13
T3	16	4	32	14
T4	20	38	42	22

R: Unrelated : heterogeneous

Classification of Scheduling Problems

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

$$\alpha \quad | \quad \beta \quad | \quad \gamma$$

where

- α specifies the **machine environment**,
- β specifies the **job characteristics**, and
- γ describes the **objective function(s).**

$P_m || C_{max}$

- n tasks, m processors
- ET: t_1, t_2, t_3,…., t_n
- **m-Subset Sum problem**
- **INDEP(m) Problem: NPC in strong sense**
- Divide the tasks in m sets such that
  - Difference of Sum of ETs of all the set is minimized**: does not exceed a value K**
  - Min (Max(Sum(Set_1), Sum(Set_2), …Sum(Set_m)))

$P_m | pmtn | C_{max}$

- n tasks, m processors, infinite pre-emption allowed
- ET: $t_1, t_2, t_3, \ldots, t_n$
- Divide all the work among all the cores equally
  - Avg=$(\Sigma\ t_i)/m$ work to each cores
  - If $max(t_i)>Avg$, $C_{max}=max(t_i)$, Task need to execute serially
- Handle the boundary cases

$Q_m | pmtn | C_{max}$

- n tasks, m uniform processors, infinite pre-emption allowed
- Longer task executed on high speed processor till it execution is long enough as compared to others
  - Sort the tasks based on LPT
  - Allocate long task to higher speed processors one by one
  - When execution time of longer task is no longer long as compared to other then co-execute

$Q_m \mid pmtn \mid C_{max}$

Algorithm level

```
t := 0;
WHILE there exist jobs with positive level {
        Assign(t);
    t1 := min{s > t | a job completes at time s};
    t2 := min{s > t | there are jobs i, j with pi(t) > pj(t)
                 and pi(s) = pj(s) at time s};
    t := min{t1, t2}
}
```

$t := 0;$

WHILE there exist jobs with positive level {

Assign(t);

$t_1 := \min\{s > t \mid \text{a job completes at time } s\};$

$t_2 := \min\{s > t \mid \text{there are jobs } i, j \text{ with } p_i(t) > p_j(t) \text{ and } p_i(s) = p_j(s) \text{ at time } s\};$

$t := \min\{t_1, t_2\}$

}

$Q_m \mid pmtn \mid C_{max}$

Assign (t)

$J := \{i \mid pi(t) > 0\}$;

$M := \{M1, . . .,Mm\}$;

WHILE $J = \emptyset$ and $M = \emptyset$ {

Find the set $I \subseteq J$ of jobs with highest level;

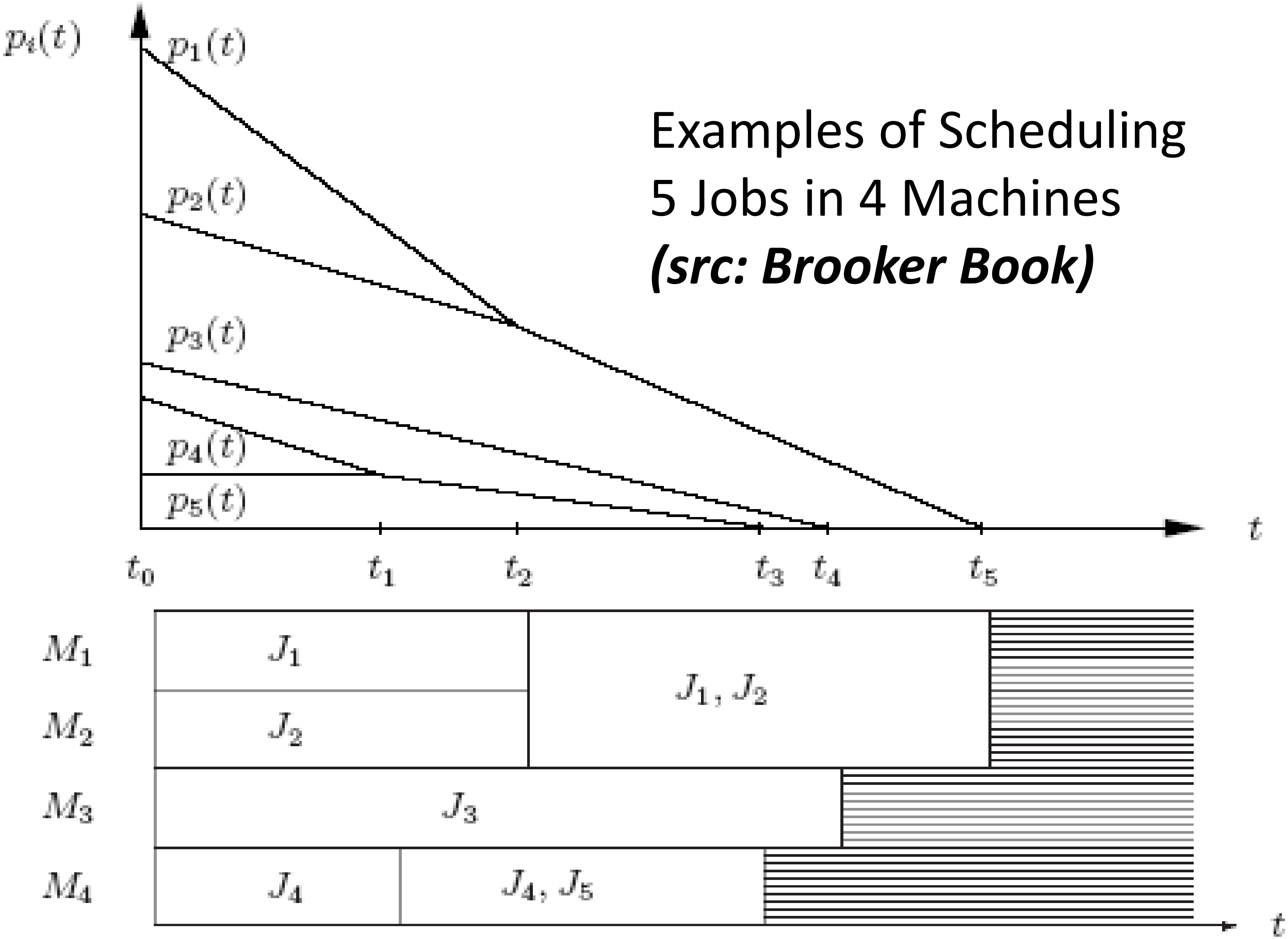
$r := \min\{|M|, |I|\}$;

Assign jobs in I to be processed jointly on the r fastest machines in M;

$J := J \setminus I$;

Eliminate the r fastest machines in M from M

$Q_m \mid pmtn \mid C_{max}$



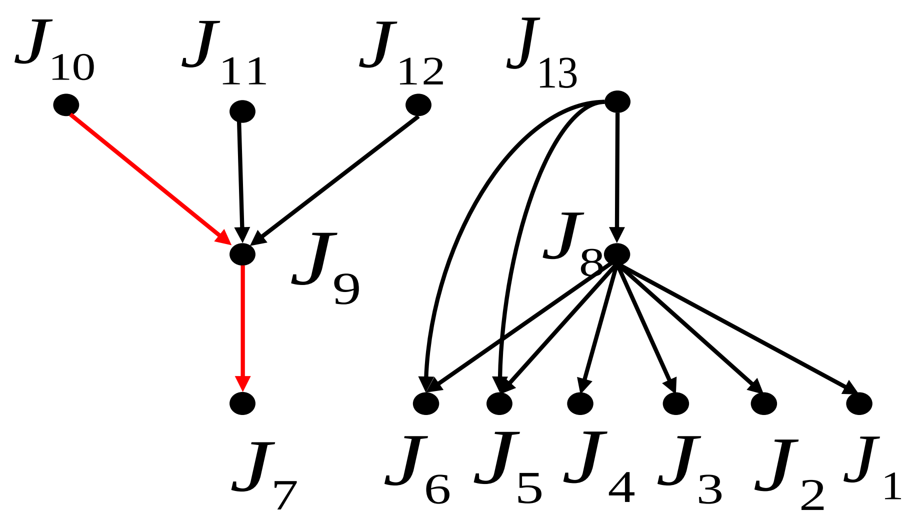
Examples of Scheduling
5 Jobs in 4 Machines
(src: Brooker Book)

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



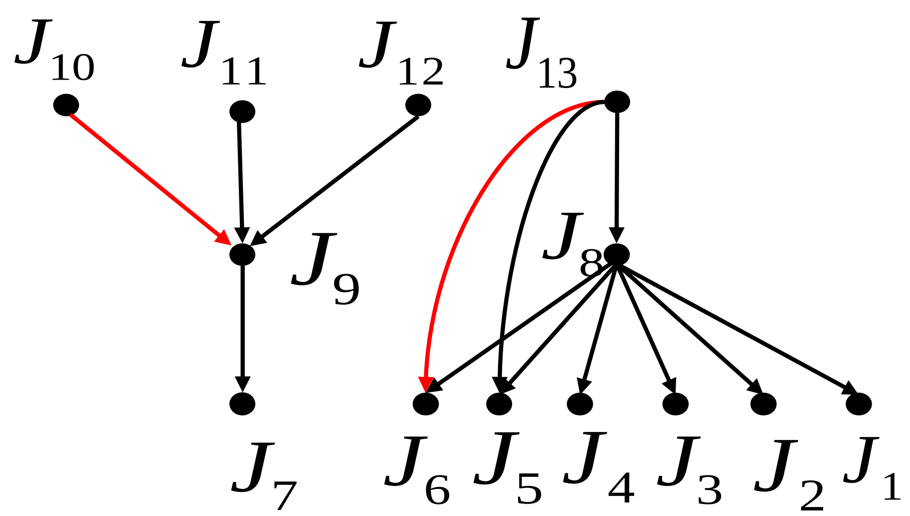
$$p(J_i) = 1$$

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



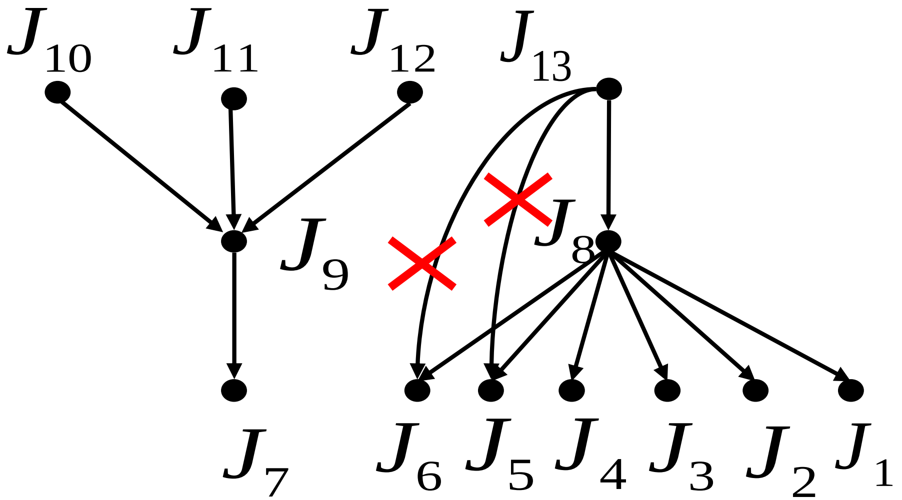
$$p(J_i) = 1$$

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

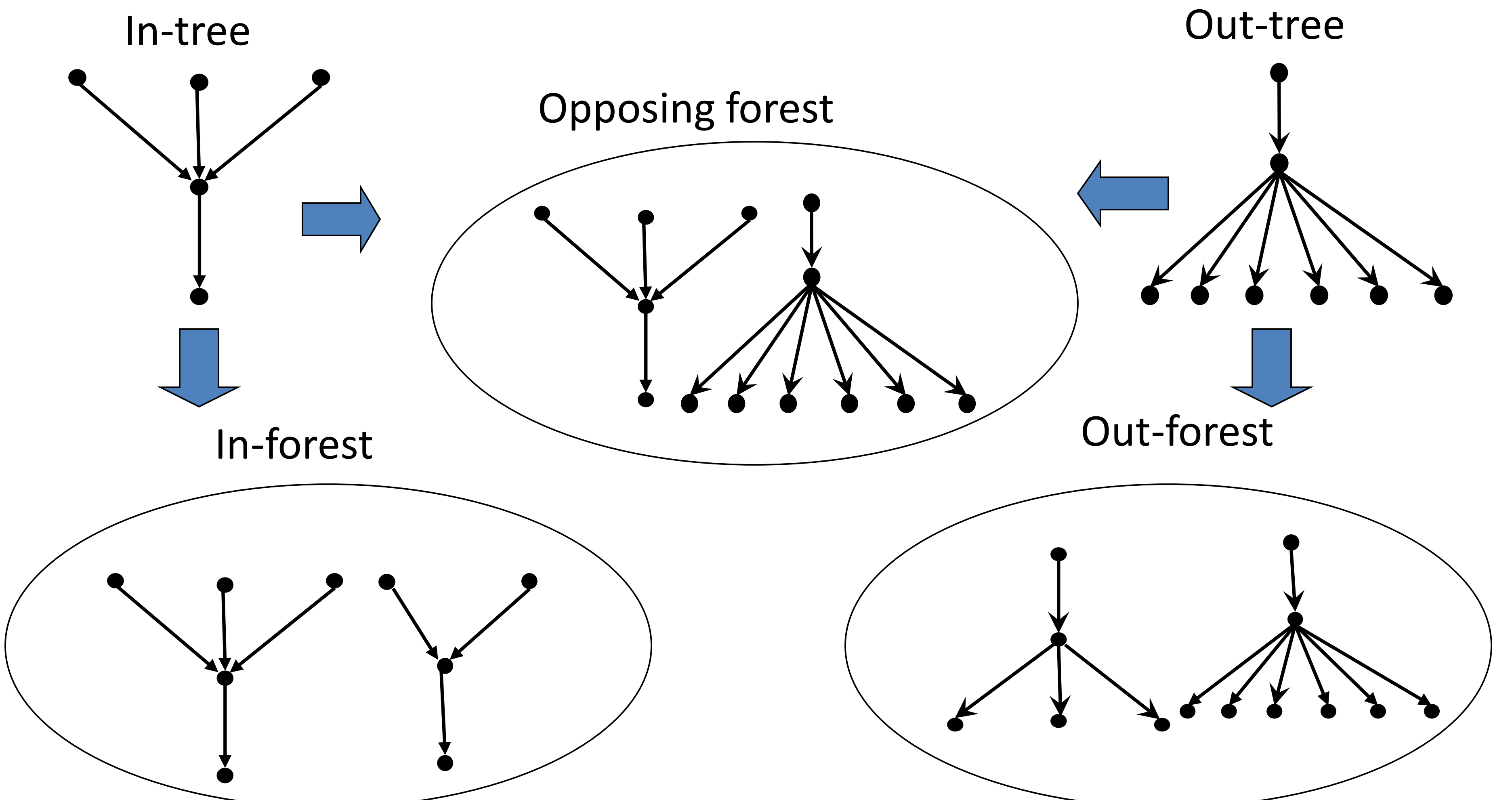


$$p(J_i) = 1$$

Special precedence constraints

- In-tree (Out-tree)
- In-forest (Out-forest)
- Opposing forest
- ***Interval orders***
- ***Series-parallel orders***
- ***Level orders***

Special precedence constraints



A Sahu

P_m | prec, $p_j = 1$ | C_{max} ($m \geq 1$)

- Processor Environment
  - m identical processors are in the system.
- Job characteristics
  - Precedence constraints are given by a 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,

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

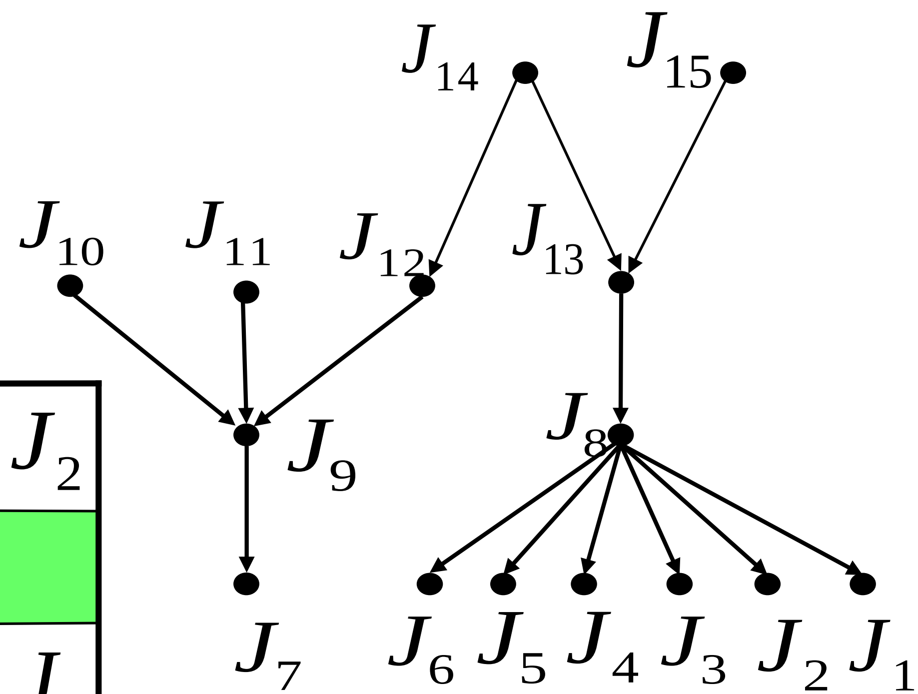
Gantt Chart

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



$$p(J_i) = 1$$

P_m | prec, $p_j = 1$ | C_{max}

Theorem 1

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

1. Ullman (1976)

   3SAT ≤ Pm | prec, $p_j = 1$ | C_{max}

2. Lenstra and Rinooy Kan (1978)

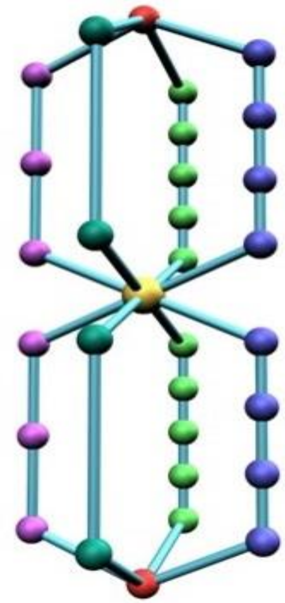
   k-clique ≤ Pm| prec, $p_j = 1$ | C_{max}

P_m| prec, pj = 1 | C_{max} is NP-complete.

Proof: out of Syllabus

P_m | prec, $p_j = 1$ | C_{max}

Mayr (1985)



- **Theorem 2**

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

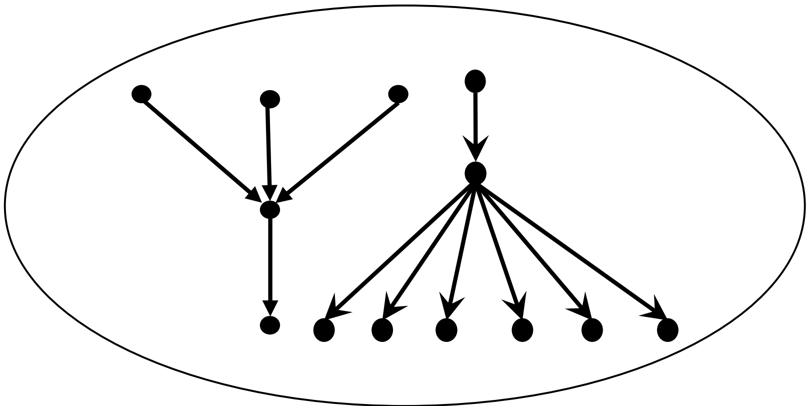
SP: Series - parallel

- **Theorem 3**

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

OF: Opposing - forest

Opposing forest

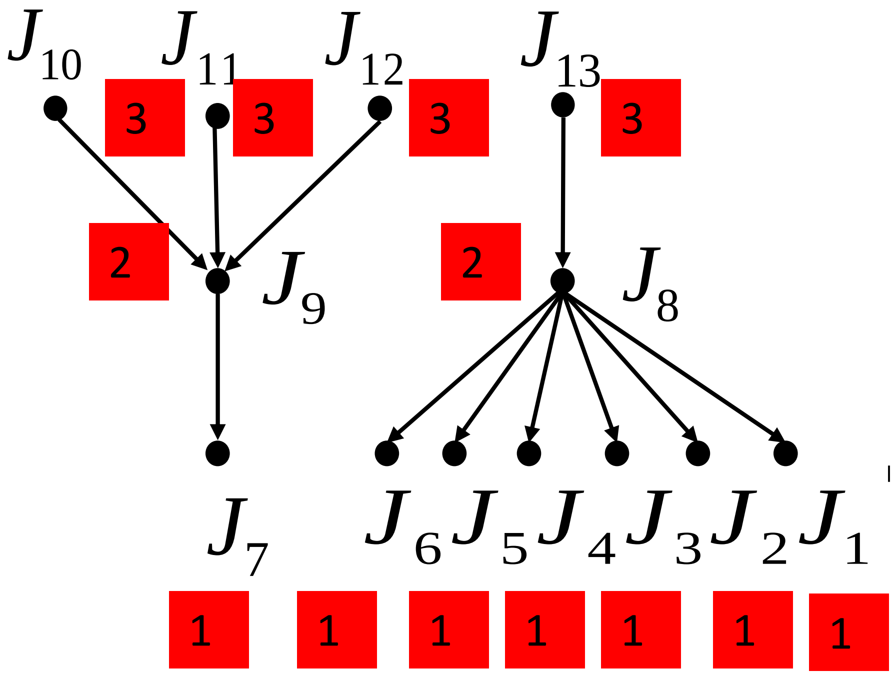


Proof: out of Syllabus

Hu's HLF/CP Algorithm

- T. C. Hu (1961), **Critical Path/Highest Level First**
- Assign a level h to each job.
  - **If job has no successors, h(j) equals 1.**
  - **Otherwise, h(j) equals one plus the maximum level of its immediate successors.**
- Set up a priority list L by nonincreasing order of the jobs' levels.
- Execute the list scheduling policy on this level based priority list L.

HLF/CP algorithm : Example



M2	J_{10}	J_{13}	J_8	J_6	J_3
M2	J_{11}	J_9	J_7	J_5	J_2
M1	J_{12}			J_4	J_1

$$L = (\overbrace{J_{10}, J_{11}, J_{12}, J_{13}}^{\text{Level 3}}, \overbrace{J_9, J_8}^{\text{Level 2}}, \overbrace{J_7, J_6, J_5, J_4, J_3, J_2, J_1}^{\text{Level 1}})$$

HLF/CP algorithm

- **Time complexity**

  O(|V|+|E|)   (|V| is the number of jobs and |E| is the number of edges in the precedence graph)

- **Theorem (Hu, 1961) : HLF/CP for Tree**
  - **The HLF algorithm is optimal for $P_m \mid p_j = 1$, in-tree (out-tree) $\mid C_{max}$.**
  - **The HLF algorithm is optimal for $P_m \mid p_j = 1$, in-forest (out-forest) $\mid C_{max}$.**

HLF/CP algorithm

- N.F. Chen & C.L. Liu (1975)

The approximation ratio of HLF algorithm for the problem with general precedence constraints:

If $m = 2$, $\delta_{HLF} \leq 4/3$.

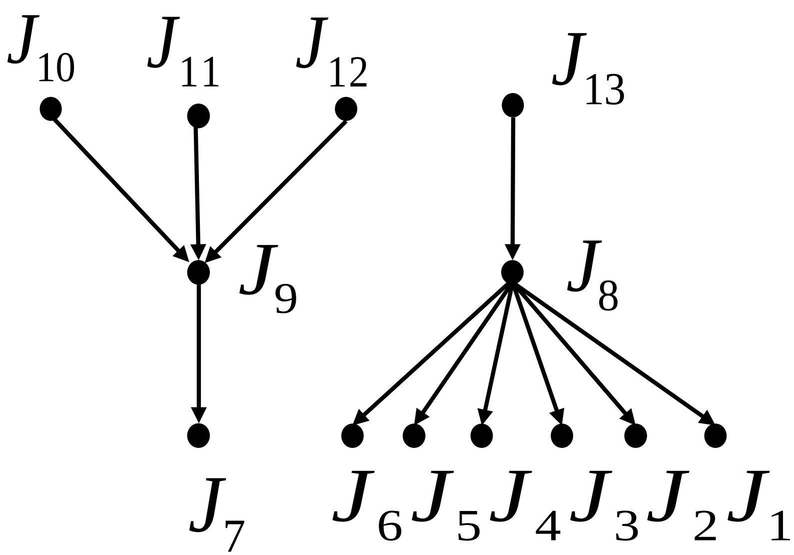
If $m \geq 3$, $\delta_{HLF} \leq 2 - 1/(m-1)$.

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

- PTAS : Polynomial Time Approximation Scheme
- Approximation List scheduling policies
  - Graham's list algorithm/Greedy List
    - **Discussed in Cilk Lectures : $T \leq 2T^*$, Also proved**
    - **CLR Book Chapter 27, Multi-threaded 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

First job of the list may not be ready

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

Graham's list algorithm

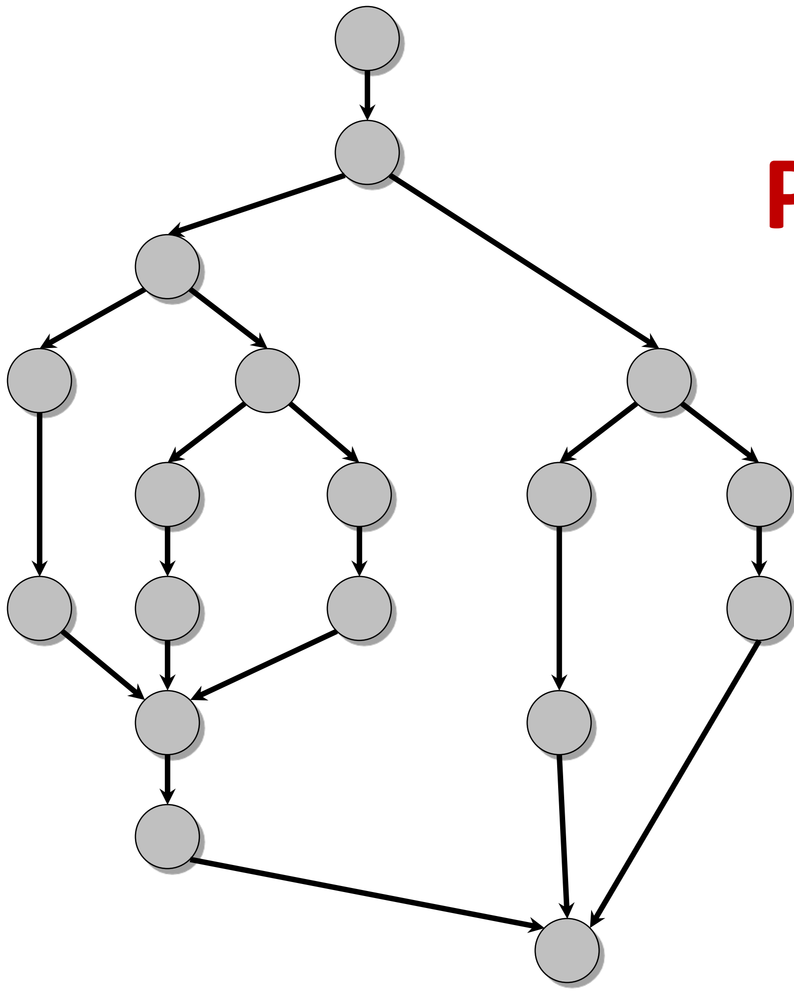
- Graham first analyzed the performance of the simplest list scheduling algorithm.
- List scheduling algorithm with an arbitrary job list is called Graham's list algorithm.
- Approximation ratio for $Pm \mid prec, p_j = 1 \mid C_{max}$

  δ = 2-1/m. (Tight bound!)

  - Approximation ratio is δ if for each input instance, the makespan produced by the algorithm is at most δ times of the optimal makespan.

CP Algo: CLR Book Page 779-783

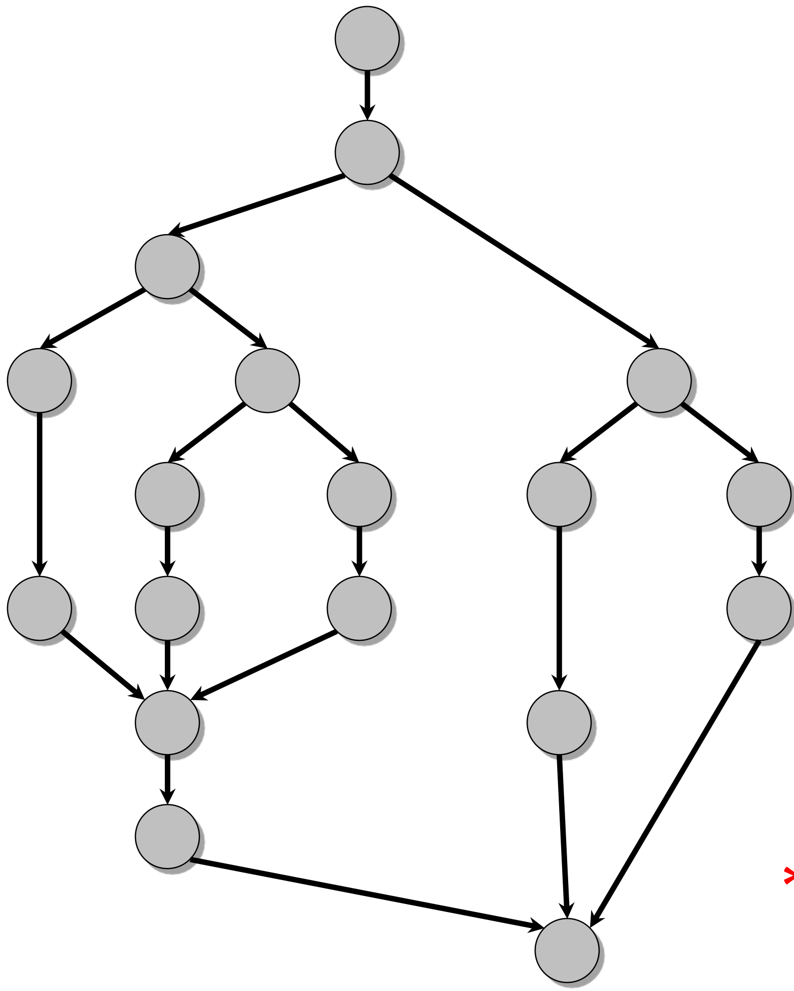
T_P = execution time on P processors



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

CP Algorithms

T_P = execution time on P processors



T_1 = ***work***

T_∞ = ***span****

LOWER BOUNDS

- $T_P \geq T_1/P$
- $T_P \geq T_\infty$

*Also called ***critical-path length*** or ***computational depth***.

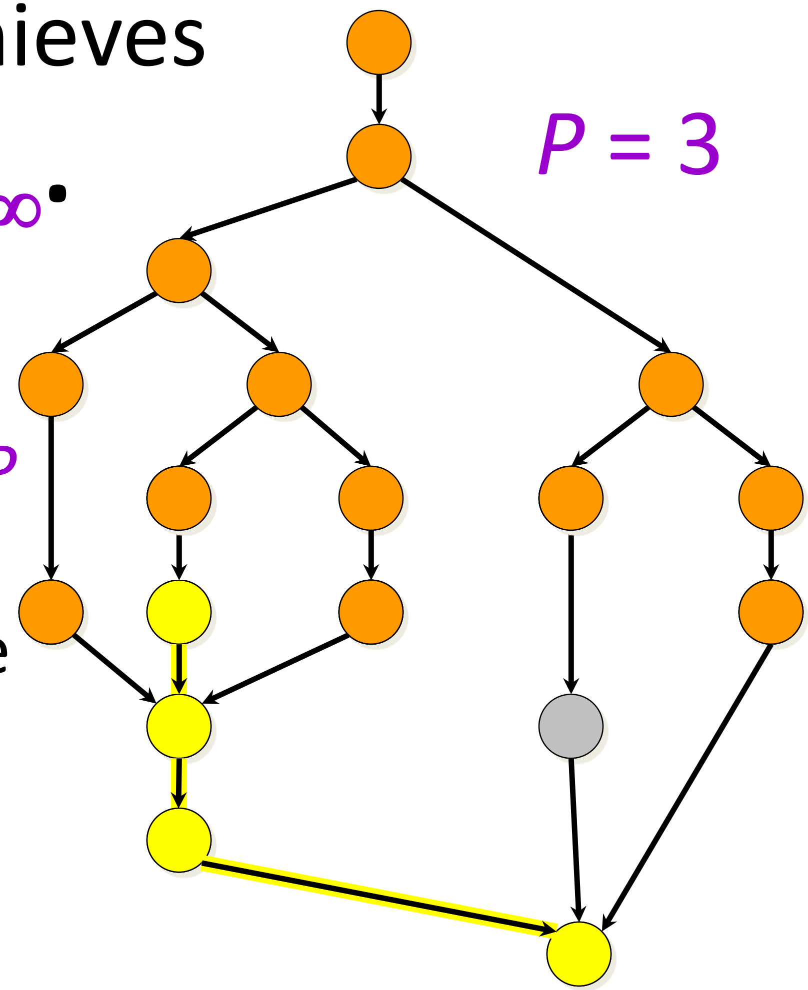
CP: Greedy-Scheduling Theorem

Theorem [Graham '68 & Brent '75].

Any greedy scheduler achieves

$$T_P \leq T_1/P + T_\infty.$$

$P = 3$



Proof.

- # complete steps $\leq T_1/P$, since each complete step performs P work.
- # incomplete steps $\leq T_\infty$, since each incomplete step reduces the span of the unexecuted dag by 1. ■

CP: Optimality of Greedy

Corollary. Any greedy scheduler achieves within a factor of 2 of optimal.

Proof. Let T_P^* be the execution time produced by the optimal scheduler. Since $T_P^* \geq \max\{T_1/P, T_\infty\}$ (lower bounds), we have

$$
\begin{aligned}
T_P &\leq T_1/P + T_\infty \\
&\leq 2 \cdot \max\{T_1/P, T_\infty\} \\
&\leq 2T_P^* . \quad \blacksquare
\end{aligned}
$$