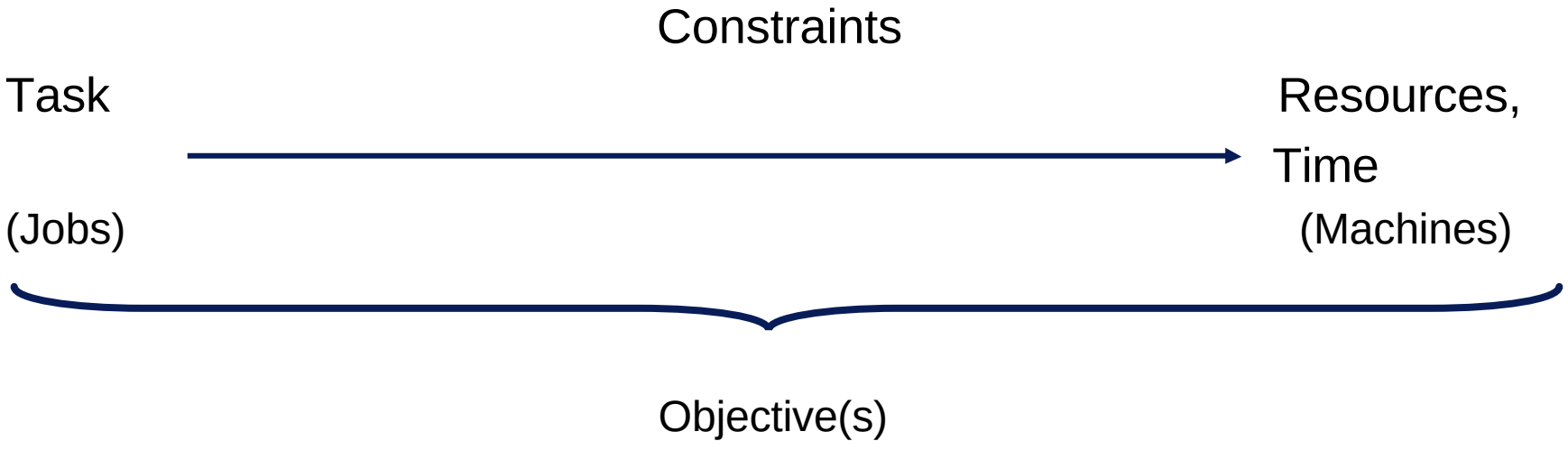
# Scheduling Problems and Solutions

Uwe Schwiegelshohn

CEI University Dortmund
Summer Term 2003

# Scheduling Problem




Areas:

- Manufacturing and production
- Transportations and distribution
- Information - processing

# Example 1 Paper Bag Factory

- Different types of paper bags
- 3 production stages
  - printing of the logo
  - gluing of the side
  - sewing of one or both ends
- several machines for each stage
  - differences in speed and function
  - processing speed and quantity
  - setup time for change of bag type
- due time and late penalty
- minimization of late penalties, setup times

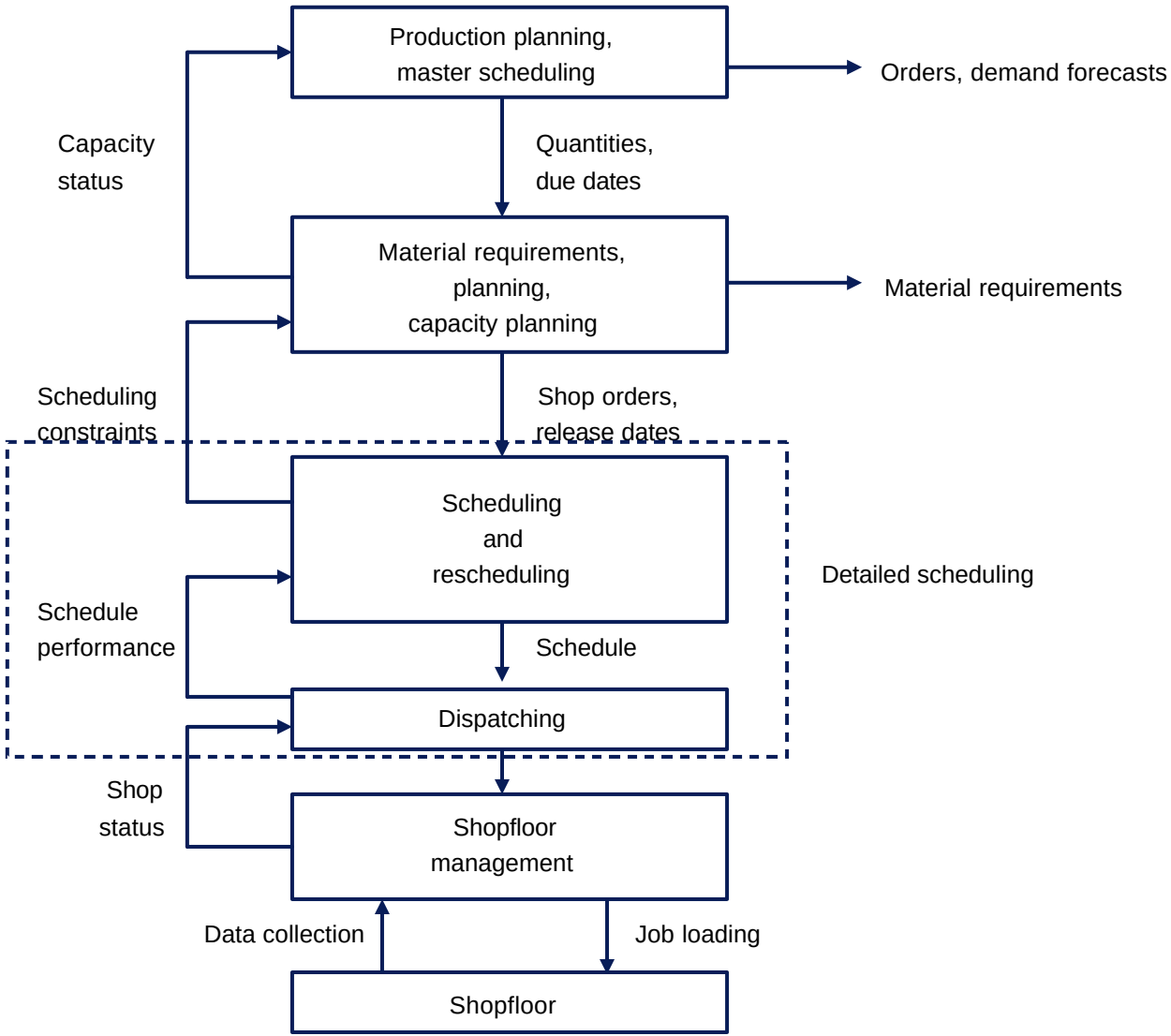
# Example 2
# Gate Assignments at Airport

- Different types of planes (size)
- Different types of gates (size, location)
- Flight schedule
  - randomness (weather, take off policy)
- Service time at gate
  - deplaning of passengers
  - service of airplane
  - Boarding of passengers
- Minimization of work for airline personnel, airplane delay

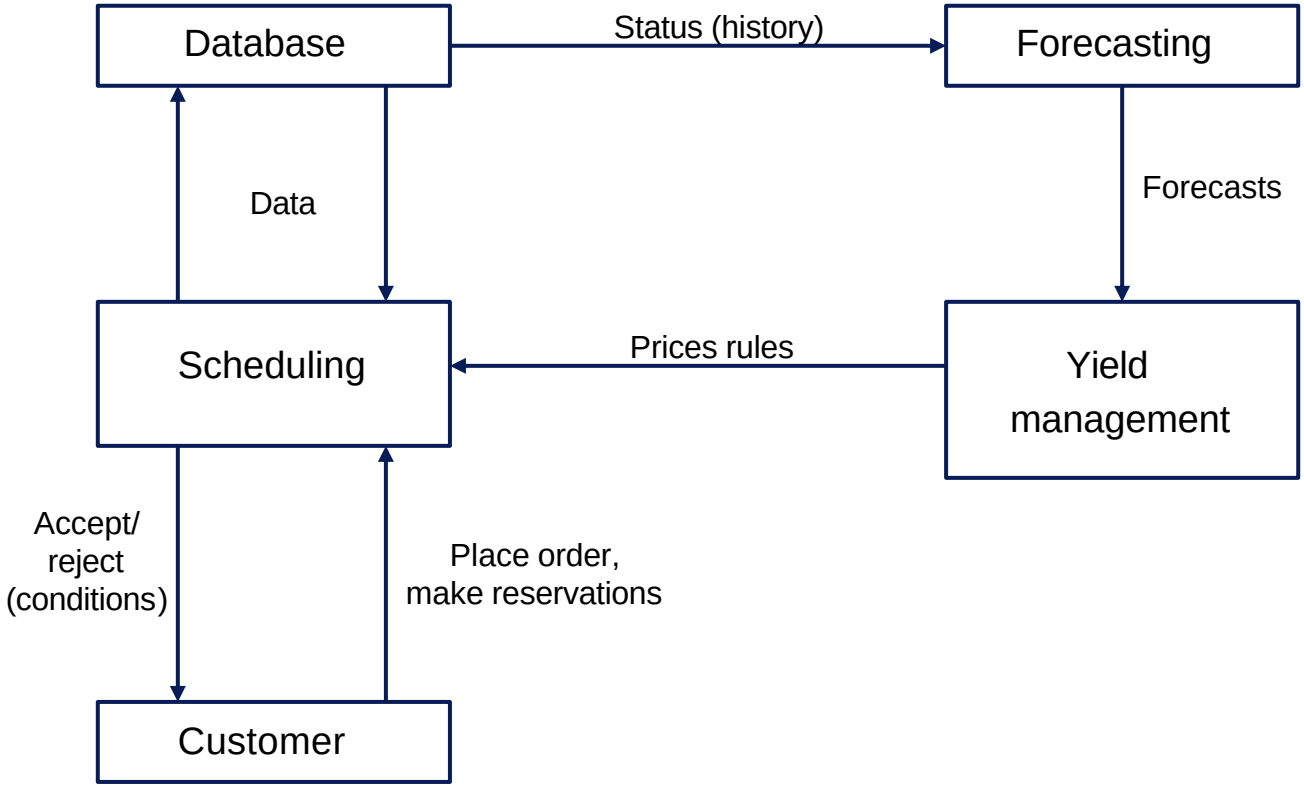
# Example 3 Tasks in a CPU

- Different applications
  - unknown processing time
  - known distributions (average, variance)
  - priority level
- Multitasking environment
  - preemption
- Minimization of sum of expected weighted completion times

# Information Flow Diagram in a Manufacturing System



# Information Flow Diagram in a Service System



# Abstract Models

Job properties:

$p_{ij}$ :processing time of job j on machine i

($p_j$ : identical processing time of job j on all machines)

$r_j$ :release data of job j (earliest starting time)

$d_j$ :due date of job j (completion of job j after $d_j$ => late penalty)

$\overline{d}_j$ :deadline ($\overline{d}_j$ must be met)

# Machine Environment

**1** : single machine

$P_m$ : identical machines in parallel

$Q_m$ : machines in parallel with different speeds

$R_m$ : unrelated machines in parallel

$F_m$ : flow shop with **m** machines in series

- each job must be processed on each machine using the same route
- queues between the machines

  FIFO queues => permutation flow shop

$FF_c$ : flexible flow shop with **c** stages in series and several identical machines at each stage, one job needs processing on only one (arbitrary) machine at each stage

# Machine Environment

$J_m$ : job show with m machines with a separate predetermined route for each job. A machine may be visited more than once by a job => recirculation

$FJ_c$: flexible job shop with c stages and several identical machines at each stage, see $FF_c$

$O_m$ : Open shop with m machines

- each job must be processed on each machine

# Restrictions and Constraints

- release dates → job properties
- sequence dependent setup times
  - $S_{ijk}$ : setup time between job j and job k on machine i
  - ($S_{jk}$ : identical for all machines)
  - ($S_{0j}$ : startup for job j)
  - ($S_{j0}$ : cleanup for job j)
- preemption (prmp)
  - the processing of a job can be interrupted and later resumed (on the same or another machine)
- precedence constraints (prec)
  - one (or more) jobs must be computed before another job can be started
    - representation as a directed acyclic graph (DAG)

# Restrictions and Constraints (2)

- machine breakdowns (brkdwn)
  - ➔ machines are not continuously available e.g. m(t) identical parallel machines are available at time t
- machine eligibility restrictions ($M_j$)
  - ➔ $M_j$ denotes the set of parallel machines that can process job j (for $P_m$ and $Q_m$)
- permutation (prmu) see $F_m$
- blocking (block)
  - ➔ A completed job cannot move from one machine to the next due to limited buffer space in the queue ⟶ it blocks the previous machine ($F_m$, $FF_c$)

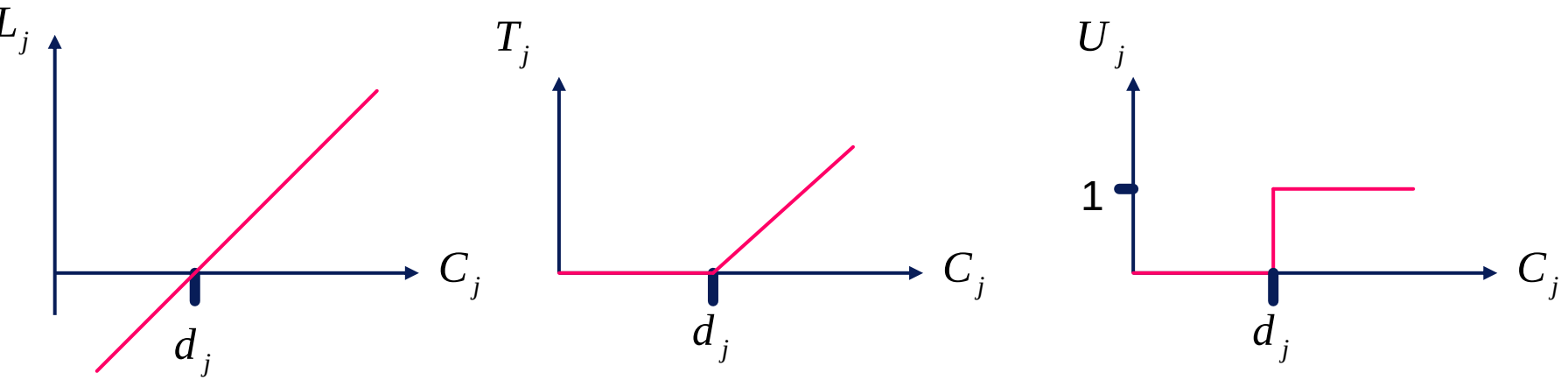
# Restrictions and Constraints (3)

- no – wait (nwt)
  - ➔ A job is not allowed to wait between two successive executions on different machines ($F_m$ , $FF_c$ )
- recirculation (recirc)

# Objective Functions

- $C_j$ : Completion time of job j
- $L_j = C_j - d_j$ : lateness of job j
  - ➔ may be positive or negative
- $T_j = \max(L_j, 0)$ tardiness
- $U_j = \begin{cases} 1 & \text{if } C_j > d_j \\ 0 & \text{otherwise} \end{cases}$

# Objective Functions (2)

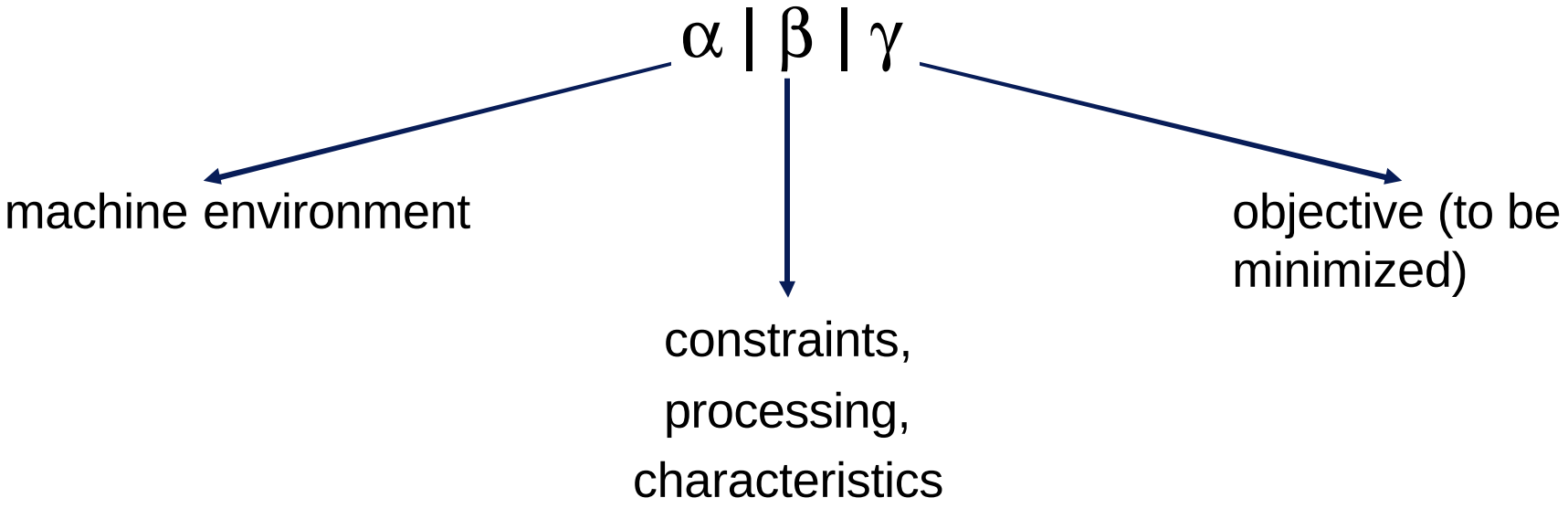



- Makespan $C_{max}$ : max ($C_1$ ,...,$C_n$ )
  - ➔ completion time of the last job in the system
- Maximum Lateness $L_{max}$ : max ($L_1$,..., $L_n$ )

# Objective Functions (3)

- Total weighted completion time $\Sigma w_j C_j$
- Total weighted flow time $(\Sigma w_j ( C_j – r_j )) = \Sigma w_j C_j – \underbrace{\Sigma w_j r_j}_{\text{const.}}$
- Discounted total weighted completion time
  - ➔ $(\Sigma w_j (1 – e^{-rc_j}))$ $0<r<1$
- Total weighted tardiness $(\Sigma w_j T_j)$
- Weighted number of tardy jobs $(\Sigma w_j U_j)$
- Regular objective functions:
  - ➔ non decreasing in $C_1 ,...,C_n$
  - ➔ $E_j = \max (-L_j , 0)$ earliness
    - non increasing in $C_j$
- $\Sigma E_j + \Sigma T_j$ , $\Sigma w_j' E_j + \Sigma w_j'' T_j$ ⟶ not regular obj. functions

# Description of a Scheduling Problem




Examples:

- Paper bag factory — $FF3 \mid r_j \mid \Sigma w_j T_j$
- Gate assignment — $P_m \mid r_j , M_j \mid \Sigma w_j T_j$
- Tasks in a CPU — $1 \mid r_j \text{ prmp} \mid \Sigma w_j C_j$
- Traveling Salesman — $1 \mid s_{jk} \mid C_{max}$

# Classes of Schedules

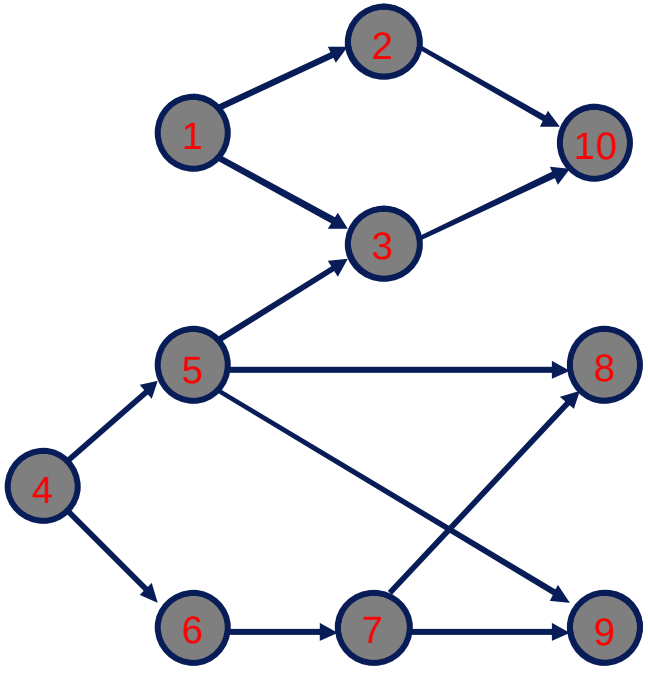
- Nondelay (greedy) schedule
  - ➔ No machine is kept idle while a task is waiting for processing

Nondelay schedule need not be optimal!

Example: P2 | prec | $C_{max}$

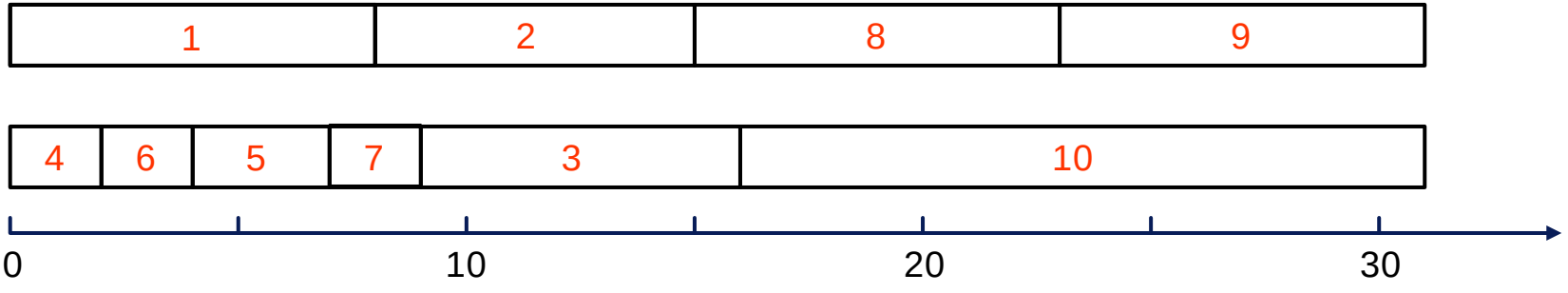
| jobs | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $p_j$ | 8 | 7 | 7 | 2 | 3 | 2 | 2 | 8 | 8 | 15 |

# Precedence Constraints Original Schedule

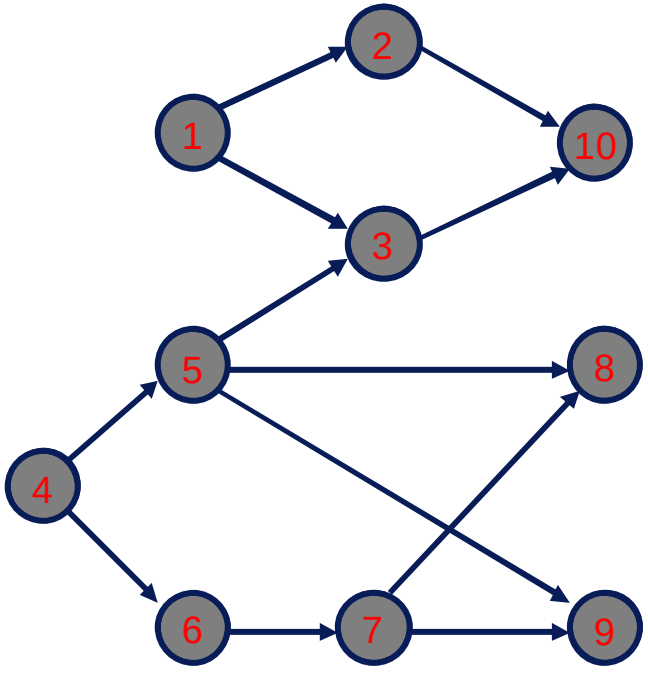



| jobs | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $p_j$ | 8 | 7 | 7 | 2 | 3 | 2 | 2 | 8 | 8 | 15 |

= job completed

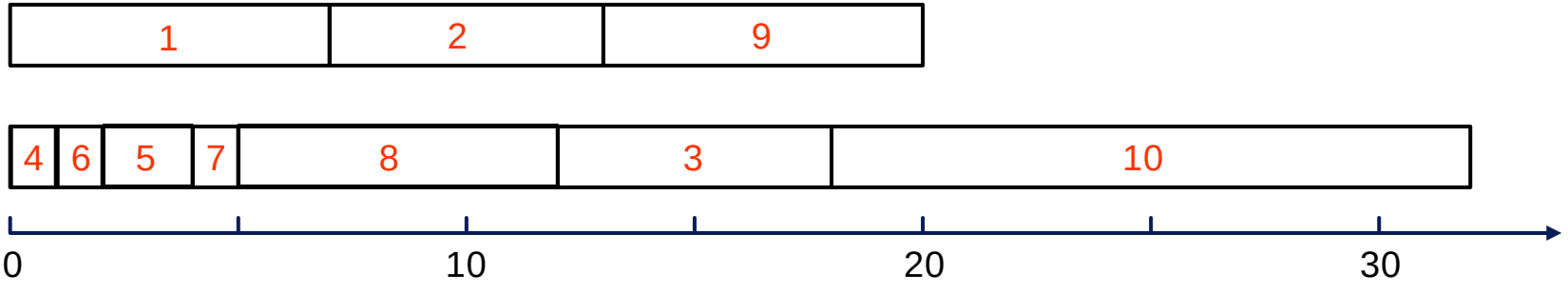


# Precedence Constraints (2)
# Processing Time 1 Unit Less



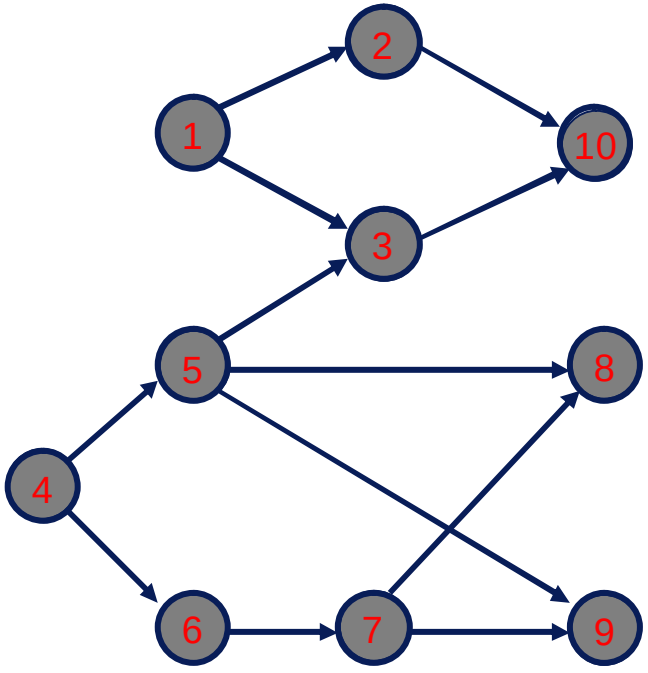

| jobs | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $p_j$ | 7 | 6 | 6 | 1 | 2 | 1 | 1 | 7 | 7 | 14 |

= job completed



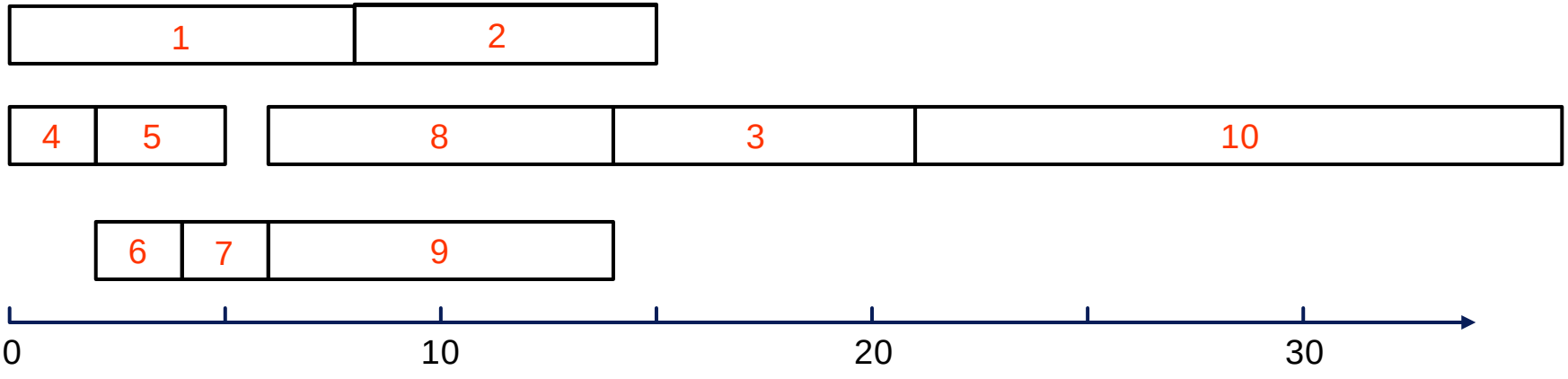
# Precedence Constraints (3)

## Original Processing Times and 3 Machines




| jobs | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $p_j$ | 8 | 7 | 7 | 2 | 3 | 2 | 2 | 8 | 8 | 15 |

= job completed



# Active Schedule

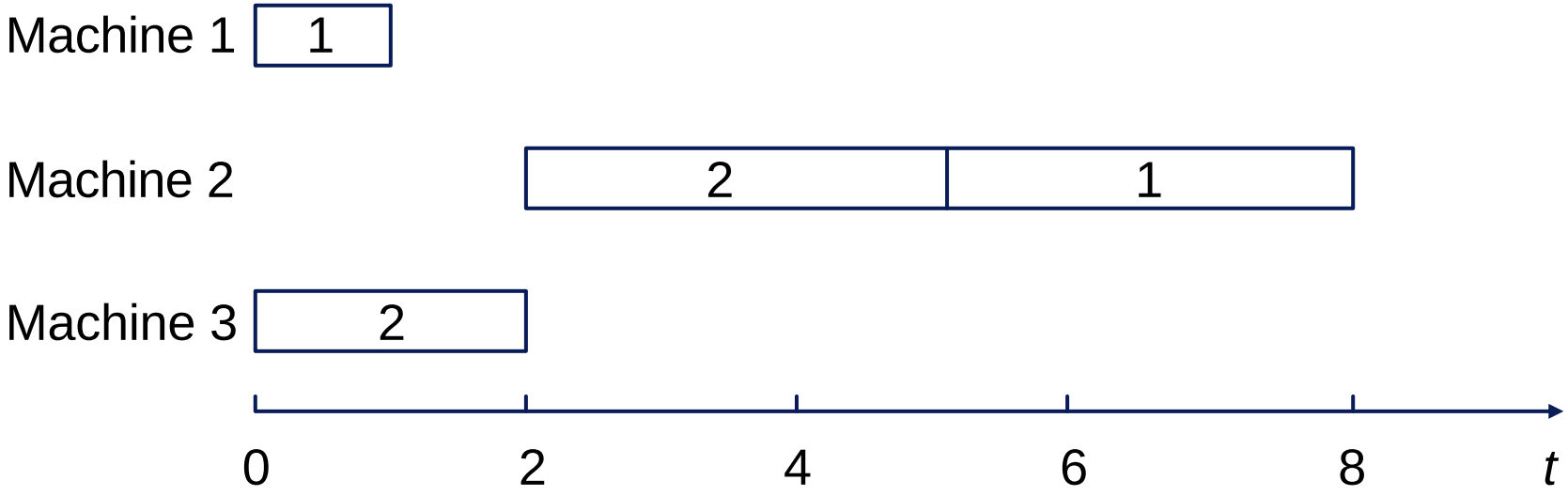
It is not possible to construct another schedule by changing the order of processing on the machines and having at least one task finishing earlier without any task finishing later.

Example :

Consider a job shop with three machines and two jobs.

- Job 1 needs one time unit on machine 1 three time units on machine 2.
- Job 2 needs two time units on machine 3 and three time units on machine 2.
- Both jobs have to be processed last on machine 2.

# Active Schedule (Example)




An active schedule that is not nondelay.

It is clear that this schedule is active; reversing the sequence of the two jobs on machine 2 postpones the processing of job 2. However, the schedule is *not* nondelay. Machine 2 remains idle until time 2 while there is already a jab available for processing at time 1.

It can be shown that there exists for $J_m \mid\mid \gamma$ an optimal schedule that is active provided the objective function $\gamma$ is regular.

# Semi – active Schedule

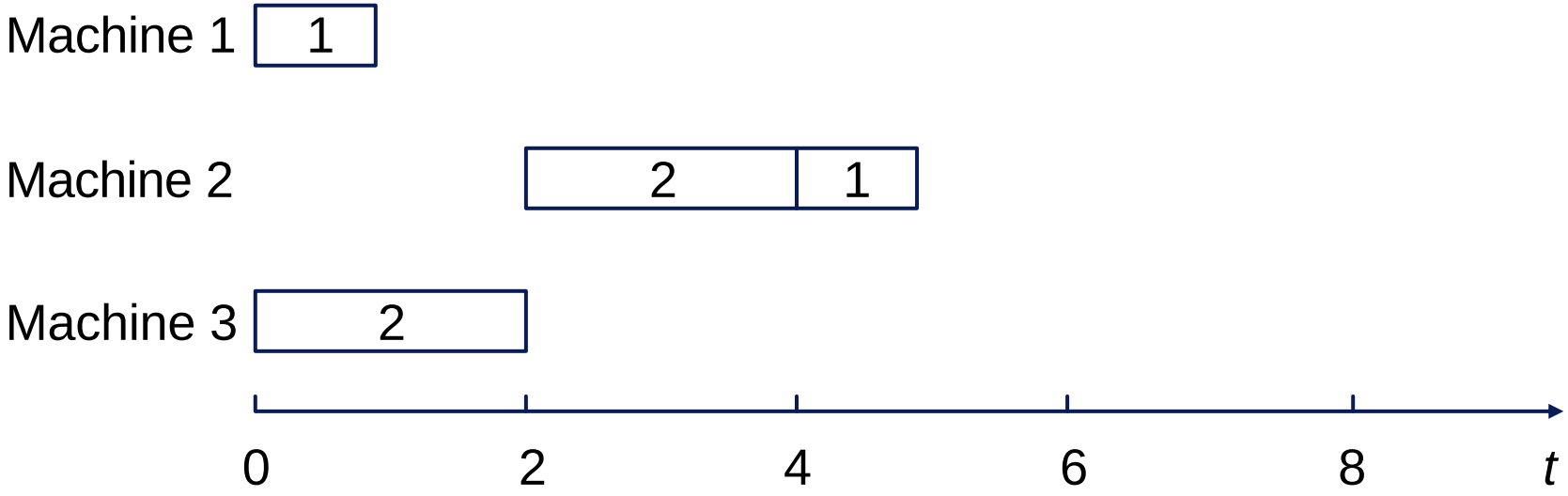
No operation can be completed earlier without changing the order of processing on any one of the machines.

Example:

Consider again a schedule with three machines and two jobs. The routing of the two jobs is the same as in the previous example.

- The processing times of job 1 on machines 1 and 2 are both equal to 1.
- The processing times of job 2 on machines 2 and 3 are both equal to 2.

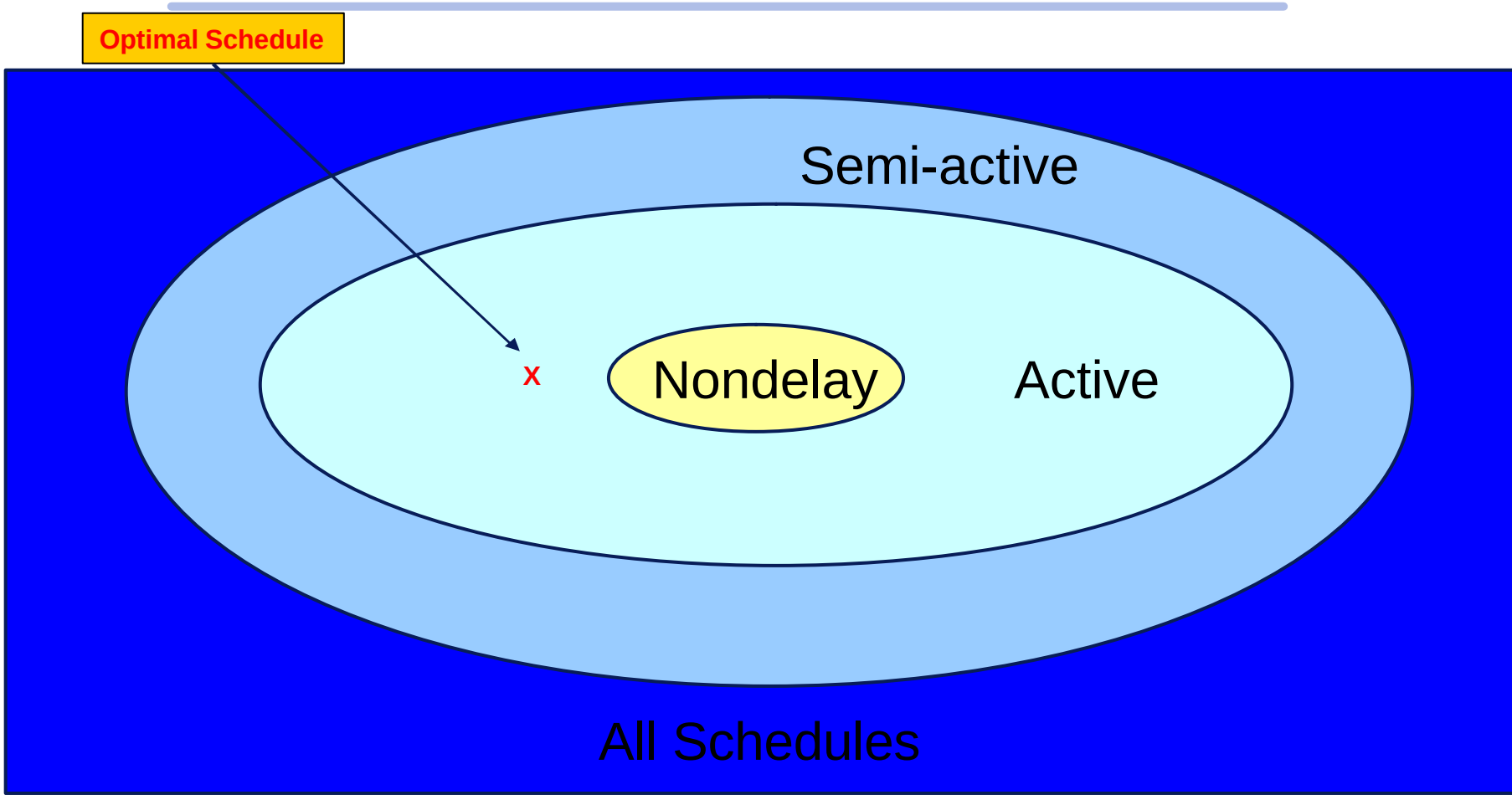
# Semi – active Schedule (Example)




A semi - active schedule that is not active.

Consider the schedule under which job 2 is processed on machine 2 before job 1. This implies that job 2 starts its processing on machine 2 at time 2 and job 1 starts its processing on machine 2 at time 4. This schedule is semi-active. However, it is not active, as job 1 can be processed on machine 2 without delaying the processing of job 2 on machine 2.

# Venn Diagram of Classes of Schedules for Job Shops




A Venn diagramm of the three classes of non preemptive schedules;
the non preemptive, nondelay schedules, the active schedules, and the semi-active schedules

# Complexity Hierarchy

Some problems are special cases of other problems

$\alpha_1 \mid \beta_1 \mid \gamma_1 \;\; \mu$ (reduces to) $\;\; \alpha_2 \mid \beta_2 \mid \gamma_2$

$1 \mid\mid \Sigma C_j \;\; \mu \;\; 1 \mid\mid \Sigma w_j C_j \;\; \mu \;\; P_m \mid\mid \Sigma w_j C_j \;\; \mu \;\; Q_m \mid \text{prec} \mid \Sigma w_j C_j$

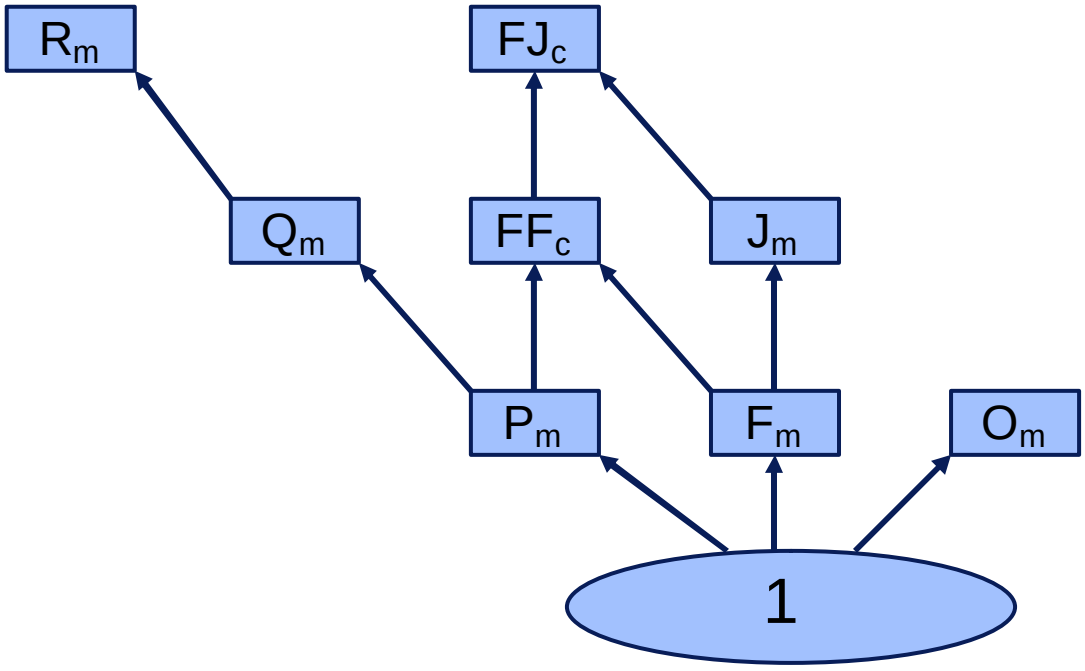
Complex cases

$\alpha \mid \beta \mid L_{max} \;\; \mu \;\; \alpha \mid \beta \mid \Sigma U_j$

$\alpha \mid \beta \mid L_{max} \;\; \mu \;\; \alpha \mid \beta \mid \Sigma T_j$

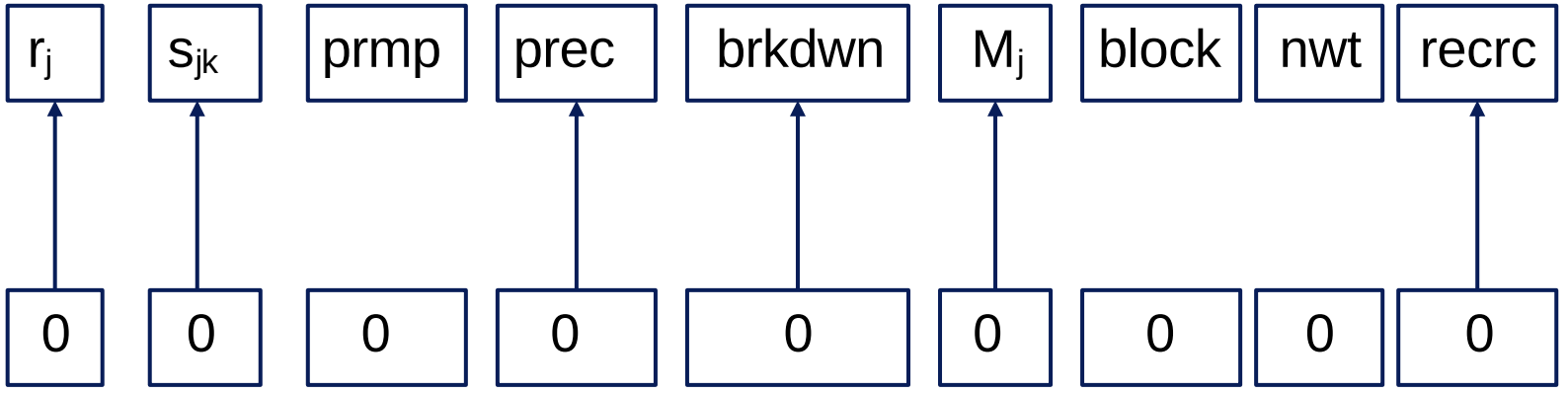
variation of $d_j$ + logarithmic search

# Complexity Hierarchies of Deterministic Scheduling Problems



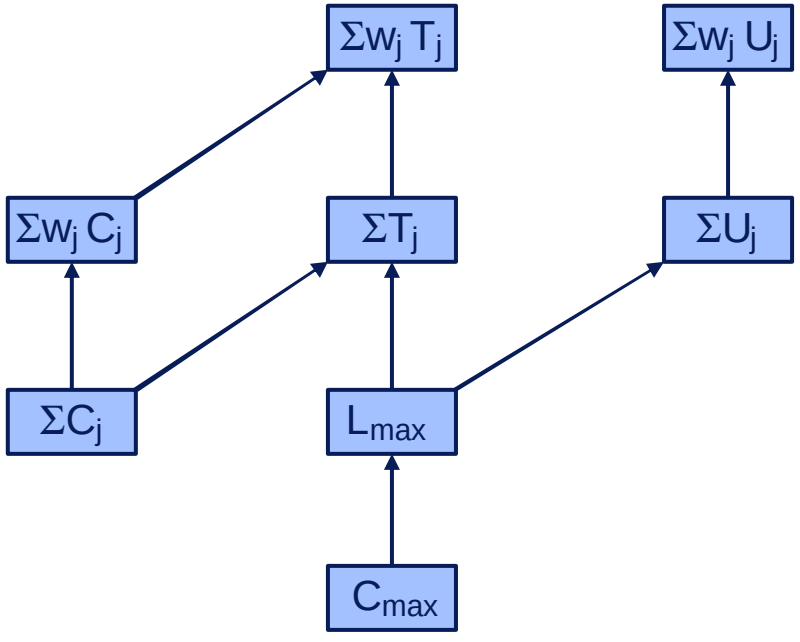

- machine environment

# Complexity Hierarchies of Deterministic Scheduling Problems



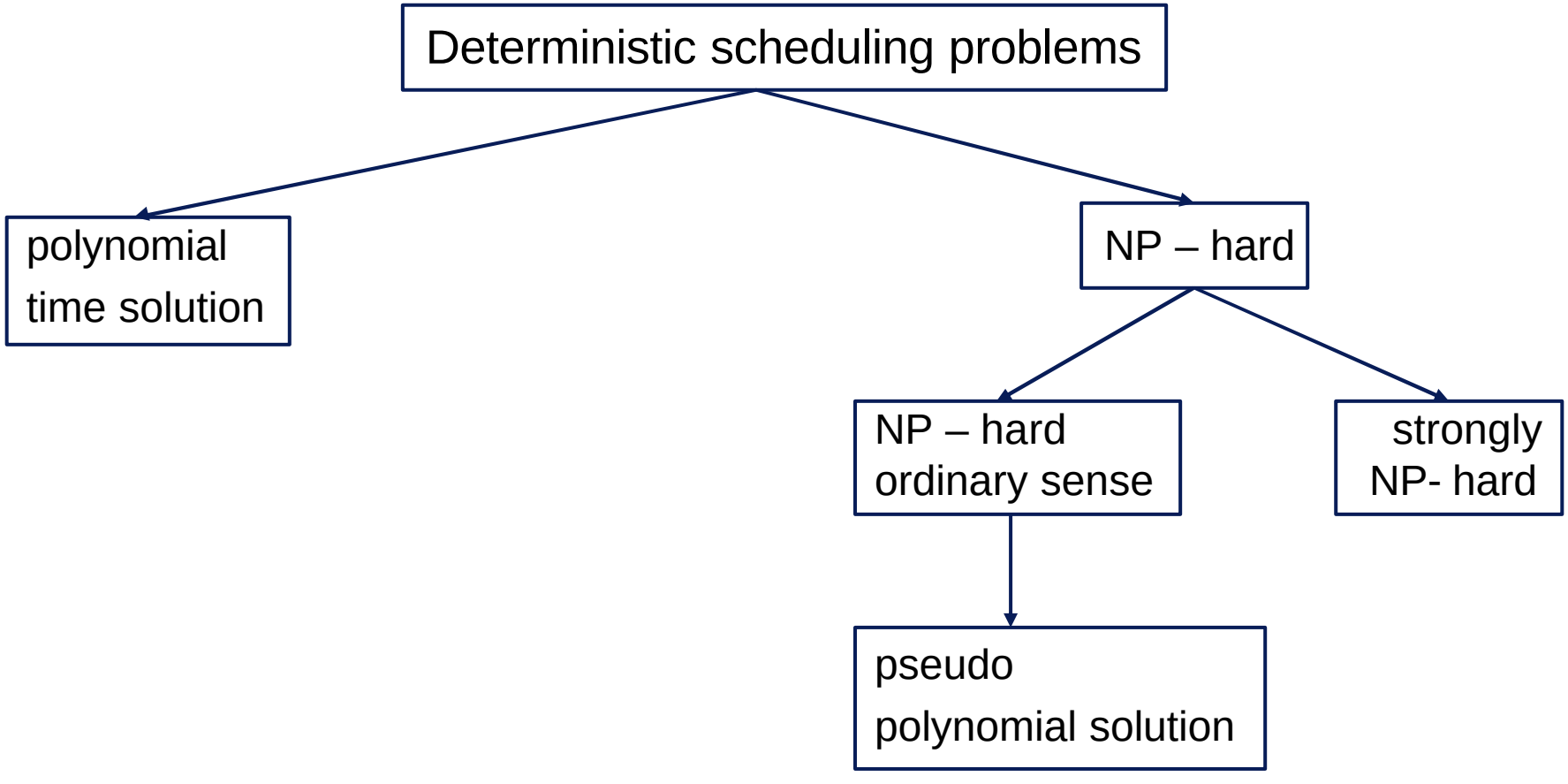

- processing restrictions and constraints

# Complexity Hierarchies of Deterministic Scheduling Problems



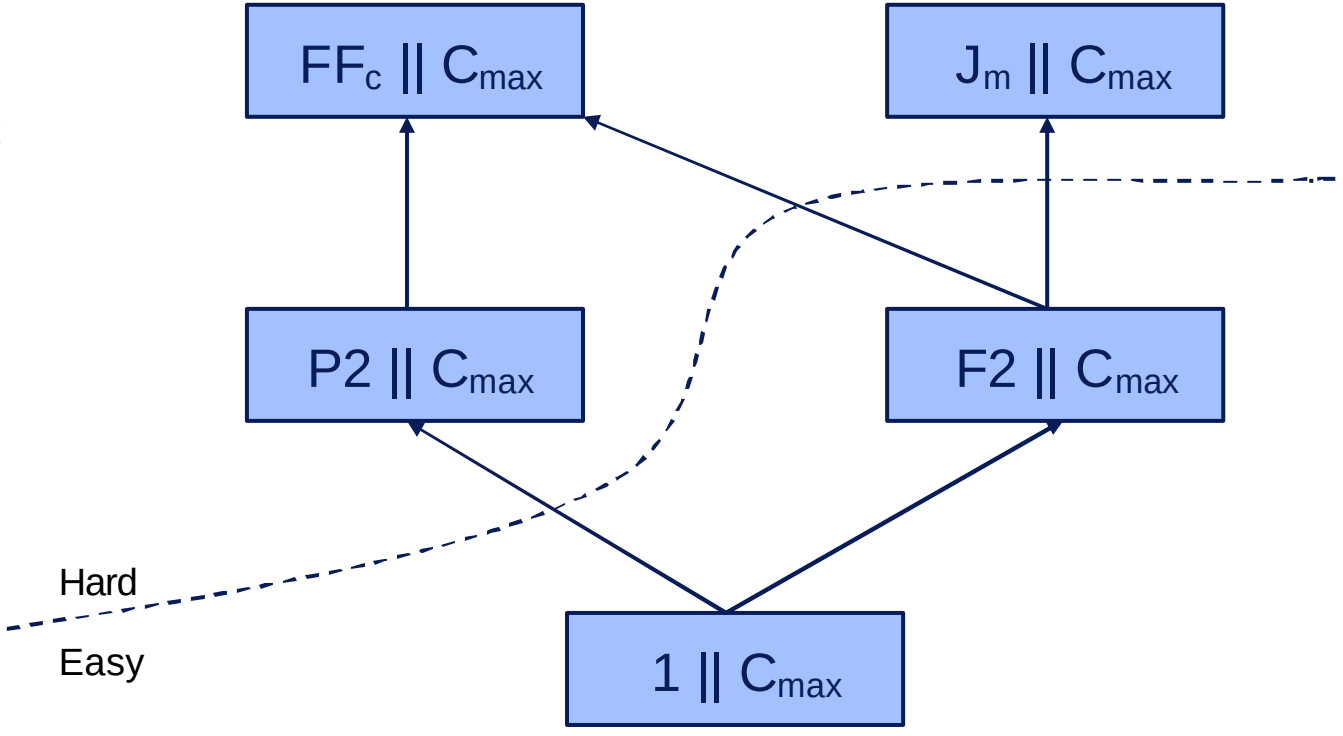

- objective functions

# Deterministic Scheduling Problems



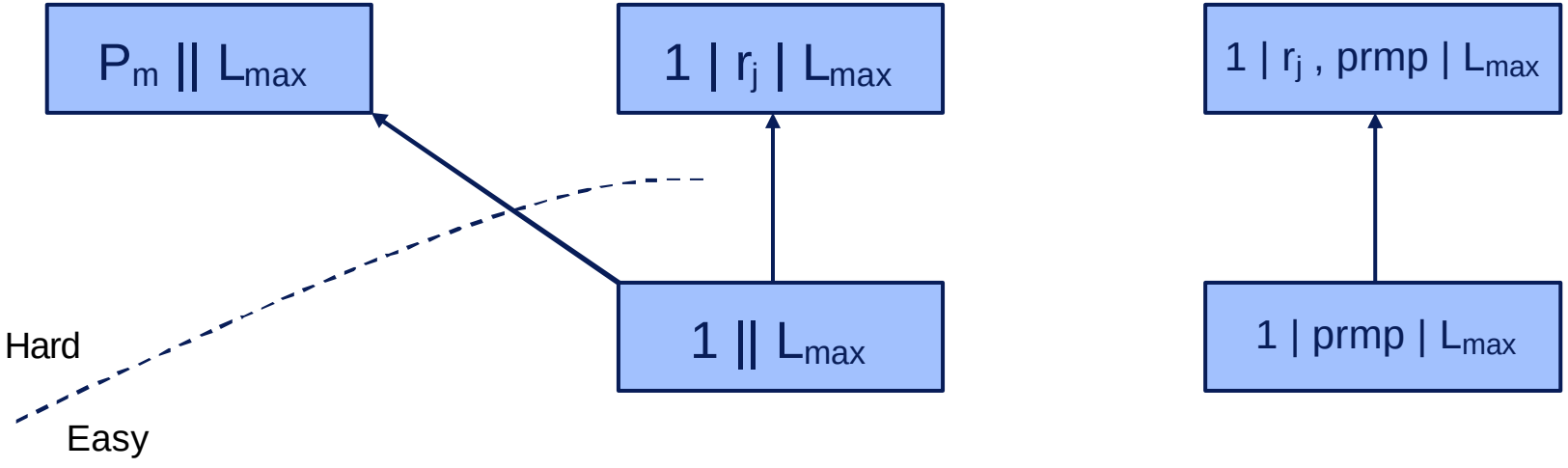
# Complexity of Makespan Problems

1. $1 \mid\mid C_{max}$
2. $P2 \mid\mid C_{max}$
3. $F2 \mid\mid C_{max}$
4. $J_m \mid\mid C_{max}$
5. $FF_c \mid\mid C_{max}$



# Complexity of Maximum Lateness Problems

1. $1 \mid\mid L_{max}$
2. $1 \mid prmp \mid L_{max}$
3. $1 \mid r_j \mid L_{max}$
4. $1 \mid r_j , prmp \mid L_{max}$
5. $P_m \mid\mid L_{max}$



# Total Weighted Completion Time

$1 \,||\, \Sigma w_j C_j$

Weighted Shortest Processing Time first (WSPT)

$$\frac{w_j}{p_j}$$ (Smith ratio)

WSPT rule is optimal for $1 \,||\, \Sigma w_j C_j$

Proof by contradiction:

WSPT rule is violated ⟶ It is violated by a pair of neighboring task h and k


$S_1$: $\Sigma w_j C_j = \ldots + w_h(t+p_h) + w_k(t + p_h + p_k)$

# Total Weighted Completion Time


$S_2$: $\Sigma\, w_j C_j = \ldots + w_k(t+p_k) + w_h(t + p_k + p_h)$

Difference between both schedules $S_1$ und $S_2$

$S_1 - S_2$: $w_k\, p_h - w_h\, p_k > 0$ (improvement by exchange)

$$\Leftrightarrow \quad \frac{w_k}{p_k} > \frac{w_h}{p_h}$$

Complexity dominated by sorting $\Rightarrow$ O (n log(n))

# Total Weighted Completion Time

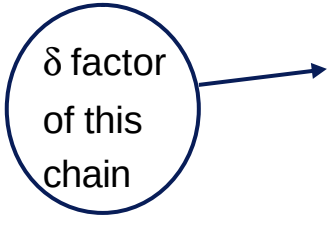
Use of precedence constraints

$$1 \mid prec \mid \Sigma w_j C_j$$

Independent chains!

Chain of jobs 1, ... , k

l* satisfies




$$\frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j} = \max_{1 \le l \le k} \left( \frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j} \right)$$

l* determines the δ-factor of the chain 1, ... , k

Computer Engineering Institute

# Algorithm: Total Weighted Completion Time with Chains

Whenever the machine is available, select among the remaining chains the one with the highest δ-factor. Schedule all jobs from this chain (without interruption) until the job that determines the δ-factor.

Proof concept

There is an optimal schedule that processes all jobs 1, ... , l* in succession + Pair wise interchange of chains

# Example: Total weighted Completion Time with Chains

- Consider the following two chains:

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$$

and

$$5 \rightarrow 6 \rightarrow 7$$

The weights and processing times of the jobs are given in the following table

| jobs | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $w_j$ | 6 | 18 | 12 | 8 | 8 | 17 | 18 |
| $p_j$ | 3 | 6 | 6 | 5 | 4 | 8 | 10 |

# Example: Total Weighted Completion Time with Chains (2)

- δ-factor of first chain $(6+18)/(3+6) = \frac{24}{9}$ ⟶ Job 2
- δ-factor of second chain $(8+17)/(4+8) = \frac{25}{12} < \frac{24}{9}$ ⟶ Job 6
  - ➔ Job 1 and 2 are processed first
- δ-factor of remaining part of first chain $\frac{12}{6} < \frac{24}{9}$ ⟶ Job 3
  - ➔ Job 5 and 6 are scheduled next
- $\frac{w_7}{p_7} = \frac{18}{10} < \frac{12}{6}$ ⟶ Job 3 is scheduled next
- $\frac{w_4}{p_4} = \frac{8}{5} < \frac{18}{10}$ ⟶ Job 7 and finally Job 4

# Algorithm: Total Weighted Completion Time with Chains (2)

- $1 \mid prec \mid \Sigma w_j C_j$ strongly NP hard for arbitrary precedence constraints
- $1 \mid r_j, prmp \mid \Sigma w_j C_j$ strongly NP hard
  - ➔ WSPT (remaining processing time) is not optimal

Example: Select another job that can be completed before the next release

$1 \mid r_j, prmp \mid \Sigma C_j$ is easy

$1 \mid r_j \mid \Sigma C_j$ is strongly NP hard

$1 \mid\mid \Sigma w_j (1 - e^{-rC_j})$ can be solved optimally with Weighted Discounted Shortest Processing Time first (WDSPT) rule:

$$\frac{w_j \bullet e^{-rp_j}}{1 - e^{-rp_j}}$$

# Maximum Lateness

## $1 \mid prec \mid h_{max}$

- $h_j(t)$: non decreasing cost function
- $h_{max} = \max(h_1(C_1), \ldots, h_n(C_n))$
- Backward dynamic programming algorithm
- $C_{max} = \Sigma\, p_j$ : makespan
- J: set of all jobs already scheduled (backwards)
  - ➔ in $[C_{max} - \sum_{j \in J} p_j, C_{max}]$
  - ➔ $J^c = \{1, \ldots, n\} \setminus J$: jobs still to be scheduled
  - ➔ $J' \subseteq J^c$ : jobs that can be scheduled (precedence constraints)

# Algorithm: Minimizing Maximum Cost

- Step 1 — Set $J = \emptyset$, $J^c = \{1, \ldots, n\}$ and $J'$ the set of all jobs with no successors.
- Step 2 — Let $j^* \in J'$ be such that

$$h_{j^*}\left(\sum_{j \in J^c} p_j\right) = \min_{j \in J'}\left(h_j\left(\sum_{k \in J^c} p_k\right)\right)$$

  Add $j^*$ to $J$

  Delete $j^*$ from $J^c$

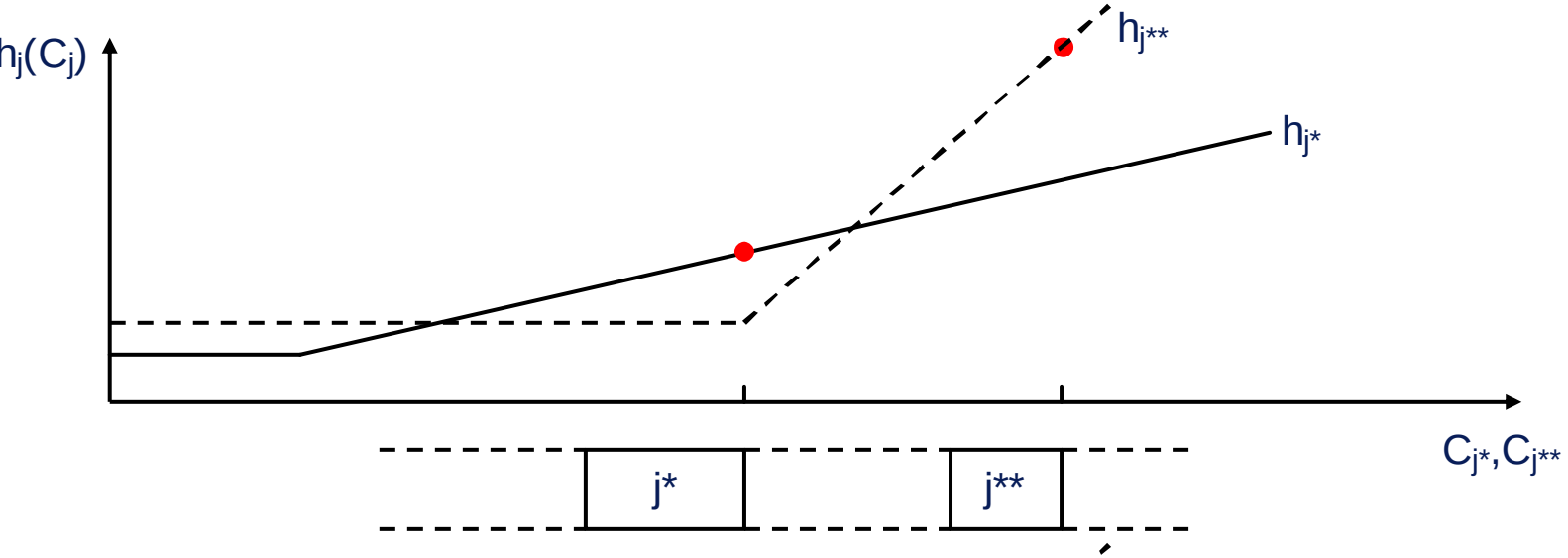
  Modify $J'$ to represent the new set of schedulable jobs.
- Step 3 — If $J^c = \emptyset$ STOP, otherwise go to Step 2.

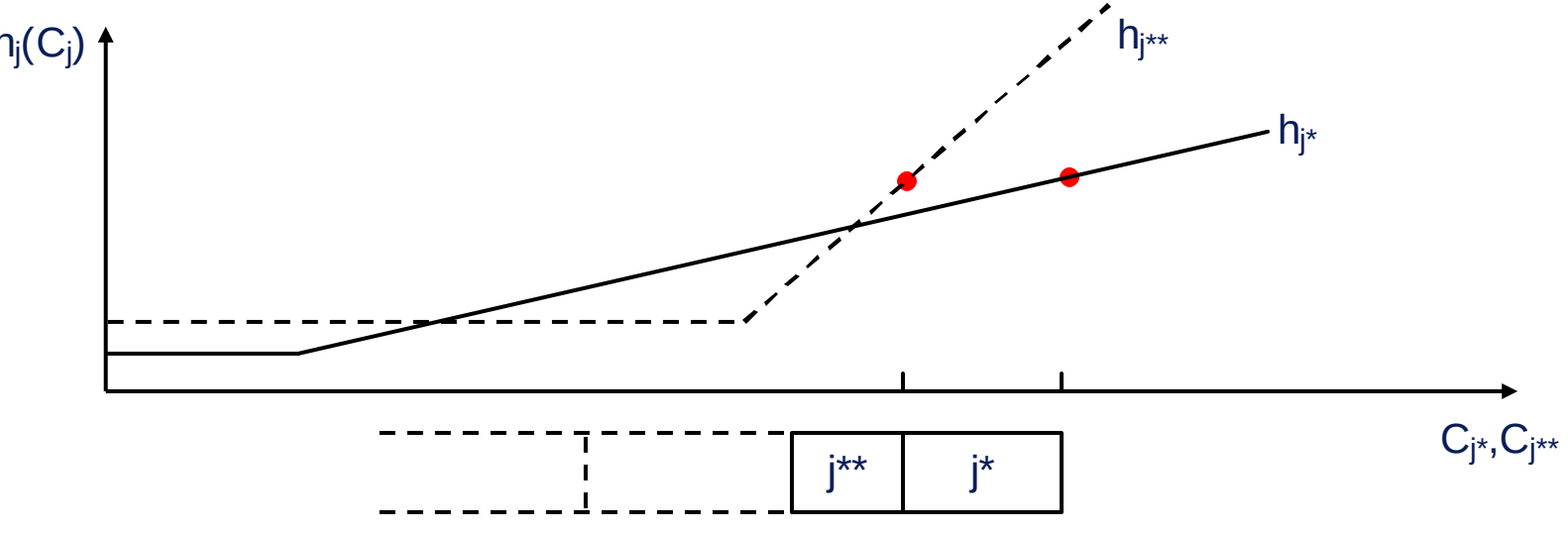
This algorithm yields an optimal schedule for $1 \mid prec \mid h_{max}$

# Algorithm: Minimizing Maximum Cost (Proof of Optimality)

- Assumption: The optimal schedule and the schedule of the previous algorithm are identical at positions k+1, ... , n
- At position k (time t)
  - ➔ Optimal schedule job j**
  - ➔ Schedule of algorithm job j* with $h_{j**}(t) > h_{j*}(t)$
- Job j* is scheduled at position k‘ < k in the optimal schedule
- j* is removed and put at position k
  - ➔ $h_j(C_j)$ does not increase for all jobs {1, ... , n} \ {j*}
  - ➔ $h_{j*}(t) < h_{j**}(t)$ (Algorithm j**)
- The exchange cannot increase $h_{max}$
  - ➔ An optimal schedule and schedule of the previous algorithm are identical at positions k, k+1, ..., n

# Algorithm: Minimizing Maximum Cost (Proof of Optimality)(2)



# Example: Minimizing Maximum Cost

| jobs | 1 | 2 | 3 |
|---|---|---|---|
| $p_j$ | 2 | 3 | 5 |
| $h_j(C_j)$ | $1 + C_j$ | $1.2\ C_j$ | 10 |

- $C_{max} = 2+3+5 = 10$
- $h_3(10) = 10 < h_1(10) = 11 < h_2(10) = 12$
  - ➔ Job 3 is scheduled last
- $h_2(10 - p_3) = h_2(5) = 6 = h_1(5)$
  - ➔ **Optimal schedules 1,2,3 and 2,1,3**

# $1 \mid\mid L_{max}$ and $1 \mid prec \mid h_{max}$

- $1 \mid\mid L_{max}$ special case of $1 \mid prec \mid h_{max}$
- $h_j = C_j - d_j \longrightarrow$ Earliest Due Date first, nondelay schedule
- $1 \mid r_j \mid L_{max} \longrightarrow$ strongly NP complete

Proof:

- reduction of 3-Partition to $1 \mid r_j \mid L_{max}$ integers $a_1, \ldots, a_{3t}, b$

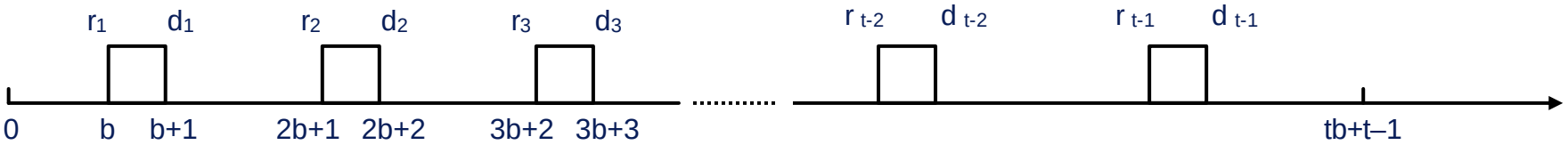
$$\frac{b}{4} < a_j < \frac{b}{2} \qquad \sum_{j=1}^{3t} a_j = t * b$$

➔ $n = 4t - 1$ jobs

| | | | |
|---|---|---|---|
| $r_j = jb + (j-1)$, | $p_j = 1$, | $d_j = jb + j$, | $j = 1, \ldots, t-1$ |
| $r_j = 0$, | $p_j = a_{j-t+1}$, | $d_j = tb + (t-1)$, | $j = t, \ldots, 4t-1$ |

# 1 || $L_{max}$ Complexity Proof

- $L_{max} \leq 0$ if every job j (1,..., t – 1) can be processed from $r_j$ to $r_j + p_j = d_j$ and all other jobs can be partitioned over t intervals of length b $\Rightarrow$ 3 – Partitions has a solution




$1 \mid r_j \mid L_{max}$ is strongly NP – hard

# Optimal Solution for 1 | $r_j$ | $L_{max}$

Optimal solution for 1 | $r_j$ | $L_{max}$ with branch and bound

➔ Tree with n+1 levels

- Level 0: 1 node
- Level 1: n nodes ( a specific job scheduled at the first position)
- Level 2: n*(n-1) nodes (from each node of Level 1 n – 1 edges to nodes of Level 2)

  (a second specific job scheduled at the second position)

➔ Nodes at Level k specify the first k positions

Assumption: $r_{j_k} < \min_{l \in J} (\max(t, r_l) + p_l)$

J: jobs that are not scheduled at the father node of Level k – 1

t: makespan at the father node of Level k – 1

➔ $j_k$ need not to be considered at a node of Level k with this specific father at Level k – 1

# Optimal Solution for $1 \mid r_j \mid L_{max}$ (2)

- Finding bounds:

  If there is a better schedule than the one generated by a branch $\Rightarrow$ the branch can be ignored

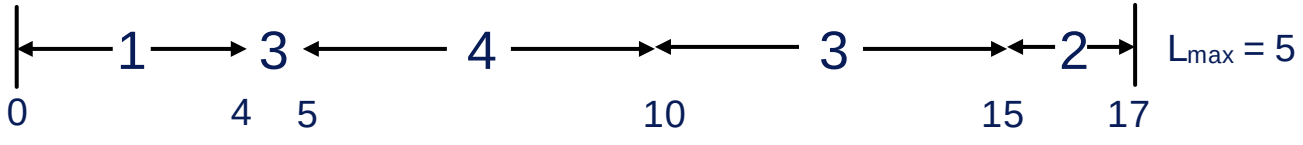
  $1 \mid r_j$ , prmp $\mid L_{max}$ can be solved by the preemptive EDD rule (non delay schedule)

  If this rule creates a non preemptive Schedule $\Rightarrow$ Optimality

# Branch and Bound Applied to Minimizing Maximum Lateness

| jobs | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $p_j$ | 4 | 2 | 6 | 5 |
| $r_j$ | 0 | 1 | 3 | 5 |
| $d_j$ | 8 | 12 | 11 | 10 |

- Level 1 (1, ?, ?, ?) (2, ?, ?, ?) (3, ?, ?, ?) (4, ?, ?, ?)
  - ➔ disregard (3, ?, ?, ?) and (4, ?, ?, ?)
    (Job 2 can be completed not after $r_3$ and $r_4$)
- Lower bound for node (1, ?, ?, ?)




- Lower bound for node (2, ?, ?, ?)

# Branch and Bound Applied to Minimizing Maximum Lateness (2)

- Lower bound for node (1, 2, ?, ?)

  1, 2, 3, 4 (non preemptive, $L_{max} = 6$)

  ➔ Disregard (2, ?, ?, ?)
- Lower bound for node (1, 3, ?, ?)

  1, 3, 4, 2 (non preemptive, $L_{max} = 5$) ← optimal

  ➔ Disregard (1, 2, ?, ?)
- Lower bound for node (1, 4, ?, ?)

  1, 4, 3, 2 (non preemptive, $L_{max} = 6$)

$1 \mid r_j$ , prec $\mid L_{max}$ similar approach

➔ more constraints (precedence) $\Rightarrow$ less nodes

# Tardy Jobs $1 \mid\mid \Sigma U_j$

- set A: all jobs that meet their due dates
  - ➔ Jobs are scheduled according to EDD
- set B: all jobs that do not meet their due dates
  - ➔ These jobs are not scheduled!

Solution with forward algorithm

J: jobs that are already scheduled (set A)

$J^d$: jobs that have been considered and are assigned to set B

$J^c$: jobs that are not yet considered

# Algorithm: Minimizing Number of Tardy Jobs

- Step 1 — Set $J = \varnothing$, $J^d = \varnothing$, $J^c = \{1, \ldots, n\}$
- Step 2 — Let j* denote the job which satisfies $d_{j^*} = \min_{j \in J^c} (d_j)$

  Add j* to J

  Delete j* from $J^c$

  Go to Step 3.
- Step 3 — If

$$\sum_{j \in J} p_j \leq d_{j^*}$$

  Go to Step 4,

  otherwise

  let k* denote the job which satisfies

$$p_{k^*} = \max_{j \in J} (p_j)$$

  Delete k* from J

  Add k* to $J^d$
- Step 4 — If $J^c = \varnothing$ STOP, otherwise go to Step 2.

# Algorithm: Minimizing Number of Tardy Jobs (Proof of Optimality)

- The worst case computation time is that of simple sorting $O(n \bullet \log(n))$
- Optimality proof

$d_1 \leq d_2 \leq \ldots \leq d_n$ (appropriate ordering)

$J_k$ subset of jobs [1, ... , k]

(I) maximum number of Jobs $|J_k| = N_k$ among [1, ... ,k] completed by their due dates

(II) Of all subsets of [1, ... ,k] with $N_k$ jobs completed by their due dates the subset with the smallest total processing time

➔ $J_n$ corresponds to optimal schedule

# Algorithm: Minimizing Number of Tardy Jobs (Proof of Optimality) (2)

- Proof concept : induction
  - ➔ correct for k=1 $\Rightarrow$ assumption correct for k

1. Job k+1 is added to set $J_k$ and it is completed by its due date
   - ➔ $J_{k+1} = J_k \cup \{k+1\}$ ?

$$|J_{k+1}| \leq N_k + 1 \quad \text{and} \quad \{k+1\} \in J_{k+1}$$

$\surd$ minimum total processing time

2. Job k+1 is added to set $J_k$ and is not completed in time $\Rightarrow$ one job (longest processing time) is deleted
   - ➔ $N_{k+1} = N_k$
   - ➔ total processing time of $J_k$ is not increased
   - ➔ no other subset of [1, ... ,k+1] can have $N_k$ on-time completions and a smaller processing time

Computer Engineering Institute

# Example: Minimizing Number of Tardy Jobs

| jobs | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $p_j$ | 7 | 8 | 4 | 6 | 6 |
| $d_j$ | 9 | 17 | 18 | 19 | 21 |

- Job 1fits $J_1 = \{1\}$
- Job 2fits $J_2 = \{1, 2\}$
- Job 3 does not fit $J_3 = \{1, 3\}$
- Job 4 fits $J_4 = \{1, 3, 4\}$
- Job 5 does not fit $J_5 = \{3, 4, 5\}$
  - **schedule order** **3, 4, 5, (1, 2)** $\sum$ **$U_j = 2$**

$1 \,||\, \sum w_j U_j$ is NP hard

- all due dates being the same
- knapsack problem
- size of the knapsack: $d = d_j$
- size of item j: $p_j$
- benefit of item j: $w_j$

# Example: Minimizing Number of Tardy Jobs (2)

- Heuristic WSPT rule ($w_j$ / $p_j$ ordering)
  - ➔ ratio $\frac{\sum w_j U_j(\text{WSPT})}{\sum w_j U_j(\text{OPT})}$ may be very large
- Example: WSPT: 1, 2, 3 $\quad \Sigma w_j U_j = 89$
  2, 3, 1 $\quad \Sigma w_j U_j(\text{OPT}) = 12$

| jobs | 1 | 2 | 3 |
|---|---|---|---|
| $p_j$ | 11 | 9 | 90 |
| $w_j$ | 12 | 9 | 89 |
| $d_j$ | 100 | 100 | 100 |

# Total Tardiness

- $1 \mid\mid \Sigma T_j$ : NP hard in the ordinary sense
  - ➔ pseudo polynomial time algorithm
    - polynomial in the size of the valves
- Elimination criterion (Dominance result)
  - ➔ A large number of sequences can be disregarded ⇒ new precedence constraints ⇒ easier problem

1. If $p_j \leq p_k$ and $d_j \leq d_k$ then there exists an optimal sequence in which job j is scheduled before job k

   <u>Proof by simple exchange</u>

   2 problem instances $p_1, ..., p_n$

   First instance $d_1, ..., d_n$

   $C'_k$: latest possible completion sequence (S')

   Second instance

   $d_1, ..., d_{k-1}, \max(d_k, C'_k), d_{k+1}, ..., d_n$

   S'' optimal sequence $C_j$'' completion time of job j in sequence S''

# Total Tardiness (2)

2. Any sequence that is optimal for the second instance is optimal for the first instance as well

   Assumption: $d_1 \leq \ldots \leq d_n$

   $p_k = \max(p_1, \ldots, p_n)$

   ➔ $k^{th}$ smallest due date has largest processing time

3. There is an integer $\delta$, $0 \leq \delta \leq n - k$ such that there is an optimal sequence S in which job k is preceded by all other jobs j with $j \leq k+\delta$ and followed by all jobs j with $j > k+\delta$.

   ➔ An optimal sequence consists of

   1. jobs 1, ..., k-1, k+1, ..., k+δ in some order
   2. job k
   3. jobs k+ δ+1, ... , n in some order

$$C_k(\delta) = \sum_{j \leq k+\delta} p_j$$

# Algorithm: Minimizing Total Tardiness

- ➔ Use of a special true subset of {1, ..., n}

- J(j, l, k): all jobs in the set {j, ..., l} with a processing time ≤ $p_k$ but job k is not in J(j, l, k)
- V(J(j, l, k), t) is the total tardiness of this subset in an optimal sequence that starts at time t.
- **Algorithm: Minimizing Total Tardiness**

Initial conditions:

$$V(\varnothing, t) = 0$$

$$V(\{j\}, t) = \max(0, t + p_j - d_j)$$

Recursive relation:

$$V(J(j,l,k),t) = \min_{\delta}(V(J(j,k'+\delta,k'),t) + \max(0, C_{k'}(\delta) - d_{k'}) + V(J(k'+\delta+1,l,k'), C_{k'}(\delta)))$$

where k‘ is such that

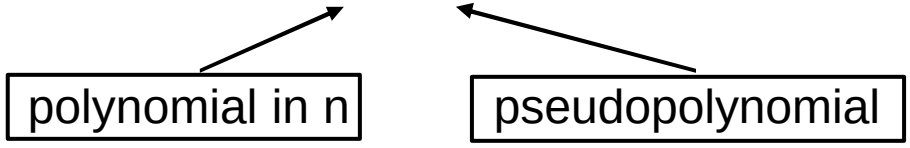
$$p_{k'} = \max(p_{j'} \mid j' \in J(j,l,k))$$

Optimal value function

$$V(\{1, ..., n\},0)$$

# Algorithm: Minimizing Total Tardiness (2)

- $O(n^3)$ subsets $J(j, l, k)$ $\Sigma p_j$ points in t
  - ➔ $O(n^3 \Sigma p_j)$ recursive equations
- Each recursion takes O(n) time
  - ➔ Running time $O(n^4 \Sigma p_j)$



# Example: Minimizing Total Tardiness

| jobs | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $p_j$ | 121 | 79 | 147 | 83 | 130 |
| $d_j$ | 260 | 266 | 266 | 336 | 337 |

- k=3 (largest processing time) $\Rightarrow 0 \leq \delta \leq 2 = 5 - 3$
- $V(\{1, 2, ..., 5\}, 0) = \min \begin{cases} V(J(1, 3, 3, ), 0) + 81 + V(J(4, 5, 3), 347) \\ V(J(1, 4, 3), 0) + 164 + V(J(5, 5, 3), 430) \\ V(J(1, 5, 3), 0) + 294 + V(\varnothing, 560) \end{cases}$
- V(J(1, 3, 3), 0) = 0  Sequences 1, 2 and 2, 1
- V(J(4, 5, 3), 347) = 347 +83 – 336 +347 + 83 +130 – 337 = 317
  Sequence 4, 5
- V(J(1, 4, 3), 0) = 0  Sequences 1, 2, 4 and 2, 1, 4
- V(J(5, 5, 3), 430) = 430 + 120 – 337 =223

# Example: Minimizing Total Tardiness (2)

- V(J(1, 5, 3), 0) = 76  Sequences 1, 2, 4, 5 and 2, 1, 5, 4

➔ $V(\{1, ..., 5\}, 0) = \min \left\{ \begin{array}{l} 0 + 81 + 317 \\ 0 + 164 + 223 \\ 76 + 294 + 0 \end{array} \right\} = 370$

Optimal sequences 1, 2, 4, 5, 3 and 2, 1, 4, 5, 3

# Total Weighted Tardiness

- $1 \mid\mid \Sigma w_j T_j$ is NP complete in the strong sense
- Proof by reduction of 3 – Partition
- Dominance result

  If there are two jobs j and k with $d_j \leq d_k$ , $p_j \leq p_k$ and $w_j \geq w_k$ then there is an optimal sequence in which job j appears before job k.

Computer Engineering Institute

# Total Tardiness: An Approximation Scheme

- NP – hard problems $\Rightarrow$ Finding in polynomial time a (approximate) solution that is close to optimal.
- Fully Polynomial Time Approximation Scheme A for $1 \,||\, \Sigma T_j$ :

$$\sum T_j(A) \le (1+\varepsilon) \underbrace{\sum T_j(OPT)}_{\text{optimal schedule}},$$

- Running time is bounded by a polynomial (fixed degree) in n and $1/\varepsilon$

# Total Tardiness: An Approximation Scheme (2)

a) n jobs can be scheduled with 0 total tardiness iff the EDD schedule has 0 total tardiness

➔ $T_{max}(EDD) \le \sum T_j(OPT) \le \sum T_j(EDD) \le n \bullet T_{max}(EDD)$

**maximum tardiness of any job in the EDD schedule**

b) V(J,t): Minimum total tardiness of job subset J assuming processing starts at $t \ge 0$

➔ There is a time $t^* \ge 0$ such that

$V(J, t)=0$ for $t \le t^*$ and

$V(J, t)>0$ for $t > t^*$

$\Rightarrow V(J, t^* + \delta) \ge \delta$ for $\delta \ge 0$

➔ Using the pseudopolynomial algorithm to compute V(J, t) for

$$t^* < \underset{(\ge 0)}{t} < n \bullet T_{max}(EDD)$$

➔ Running time bound $O(n^5 \bullet T_{max}(EDD))$

# Total Tardiness: An Approximation Scheme (3)

c) Rescaling $p'_j = \lfloor p_j / K \rfloor$

some factor K

$$d'_j = \frac{d_j}{K}$$

S: optimal sequence for rescaled problem

$\sum T_j^*(S)$:total tardiness of sequence S for processing times $K \bullet p'_j \leq p_j$ and due dates $d_j$

$\sum T_j(S)$: total tardiness of sequence S for $p_j < K \bullet (p'_j + 1)$ and $d_j$

$$\rightarrow \sum T_j^*(S) \leq \sum T_j(OPT) \leq \sum T_j(S) < \sum T_j^*(S) + \frac{n(n+1)}{2} \bullet K$$

$$\sum T_j(S) - \sum T_j^*(S) < K \bullet \frac{n(n+1)}{2}$$

$$\sum T_j(S) - \sum T_j(OPT) < K \bullet \frac{n(n+1)}{2}$$

Select $$K = \frac{2\varepsilon}{n(n+1)} \bullet T_{max}(EDD)$$

# Algorithm: PTAS for Minimizing Total Tardiness

- ➔ $\sum T_j(S) - \sum T_j(OPT) \leq \varepsilon \bullet T_{max}(EDD)$
- ➔ Running time bound $O(n^5 \bullet T_{max}(EDD)/K) = O(n^7/\varepsilon)$

- **Algorithm: PTAS for Minimizing Total Tardiness**
- Step 1 Apply EDD and determine $T_{max}$.
  If $T_{max} = 0$, then $\sum T_j = 0$ and EDD is optimal; STOP.
  Otherwise set
  $$K = \left(\frac{2\varepsilon}{n(n+1)}\right) T_{max}(EDD)$$
- Step 2 Rescale processing times and due dates as follows:
  $$p'_j = \lfloor p_j / K \rfloor \qquad d'_j = \frac{d_j}{K}$$
- Step 3 Apply Algorithm „Minimizing Total Tardiness “ (slides 60/61) to the rescaled data.

# Example: PTAS for Minimizing Total Tardiness

| jobs | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $p_j$ | 1210 | 790 | 1470 | 830 | 1300 |
| $d_j$ | 1996 | 2000 | 2660 | 3360 | 3370 |

- Optimal total tardiness 3700
- $T_{max}(EDD)=2230$

$$\varepsilon \overset{!}{=} 0.02 \rightarrow K = 2.973$$

➔ optimal sequences for rescaled problem

1, 2, 4, 5, 3 and 2, 1, 4, 5, 3

optimal sequence for the original problem

total tardiness (original data):3704

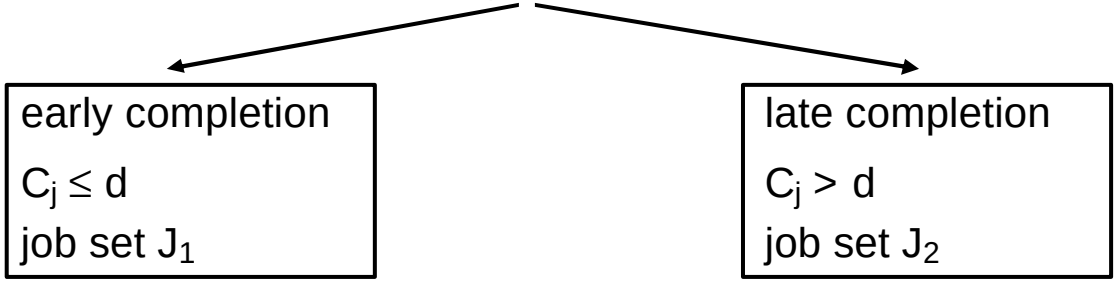
$3704 \leq 1.02 \bullet 3700$

# Total Earliness and Tardiness

- Objective $\Sigma E_j + \Sigma T_j$
  - ➔ more difficult than total tardiness
    Special case $d_j = d$ for all jobs j

**Properties**

- No idleness between any two jobs in the optimal schedule (the first job need not start at time 0)
- Schedule S $\Rightarrow$ 2 disjoint set




- Optimal Schedule:
  Early jobs ($J_1$) use Longest Processing Time first (LPT)
  Late jobs ($J_2$) use Shortest Processing Time first (SPT)

# Total Earliness and Tardiness (2)

- There is an optimal schedule such that one job completes exactly at time **d**

  **Proof:** j* starts before and completes after **d**

  $|J_1| < |J_2|$ shift schedule to the left until j* completes at **d**

  $|J_1| > |J_2|$ shift schedule to the right until j* completes at **d**

  $|J_1| = |J_2|$ j* can be shifted either way

  Assume that the first jobs can start its processing after t = 0 and $p_1 \geq p_2 \geq \ldots \geq p_n$

# Algorithm: Minimizing Total Earliness and Tardiness with a Loose Due Date

- Step 1 — Assign job 1 to set $J_1$
  Set k = 2.
- Step 2 — Assign job k to set $J_1$ and job k + 1 to set $J_2$ or vice versa.
- Step 3 — If $k+2 \leq n-1$, set k = k+2 and go to Step 2
  If k+2 = n, assign job n to either set $J_1$ or set $J_2$ and STOP
  If k+2 = n+1, all jobs have been assigned; STOP.

If job processing must start at 0 $\Rightarrow$ problem is NP – hard

Heuristic algorithm (effective in practice)

$p_1 \geq p_2 \geq \ldots \geq p_n$

# Algorithm: Minimizing Total Earliness and Tardiness with a Tight Due Date

- Step 1 — Set $\tau_1 = d$ and $\tau_2 = \Sigma p_j - d$.
  Set $k = 1$.
- Step 2 — If $\tau_1 > \tau_2$, assign job k to the first unfilled position in the sequence and set $\tau_1 = \tau_1 - p_k$.
  If $\tau_1 < \tau_2$, assign job k to the last unfilled position in the sequence and set $\tau_2 = \tau_2 - p_k$.
- Step 3 — If $k < n$, set $k = k + 1$ and go to Step 2.
  If $k = n$, STOP

Computer Engineering Institute

# Example: Minimizing Total Earliness and Tardiness with a Tight Due Date

- 6 jobs with d = 180

| jobs | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $p_j$ | 106 | 100 | 96 | 22 | 20 | 2 |

- Applying the heuristic yields the following results.

| $t_1$ | $t_2$ | **Assignment** | **Sequence** |
|---|---|---|---|
| 180 | 166 | Job 1 Placed First | 1xxxxx |
| 74 | 166 | Job 2 Placed Last | 1xxxx2 |
| 74 | 66 | Job 3 Placed First | 13xxx2 |
| -22 | 66 | Job 4 Placed Last | 13xx42 |
| -22 | 44 | Job 5 Placed Last | 13x542 |
| -22 | 12 | Job 6 Placed Last | 136542 |

# Example: Minimizing Total Earliness and Tardiness with a Tight Due Date (2)

- Objective $\Sigma\, w'E_j + \Sigma\, w''T_j$, $d_j = d$
  - ➔ Generalization of $\Sigma\, E_j + \Sigma\, T_j$ possible using $w' - w''$
- Objective $\Sigma\, w_j'E_j + \Sigma\, w_j''T_j$, $d_j = d$
  - ➔ use of $(w_j / p_j)$ and WLPT and WSPT instead of LPT and SPT
- Objective $\Sigma\, w'E_j + \Sigma\, w''T_j$, different due dates
  - ➔ NP – hard
  - a) Sequence of the jobs
  - b) Idle times between the jobs
    - ➔ dependent optimization problems
- Objective $\Sigma\, w_j'E_j + \Sigma\, w_j''T_j$, different due dates
  - ➔ NP – hard in the strong sense

    (more difficult than total weighted tardiness)

Predetermined sequence $\Rightarrow$ timing can be determined in polynomial time

# Primary and Secondary Objectives

A scheduling problem is solved with respect to the primary objective. If there are several optimal solutions, the best of those solutions is selected according to the secondary objective

$$\alpha \mid \beta \mid \gamma_1 \text{ (opt)}, \gamma_2$$

primary objectives

secondary objectives

$$1 \mid\mid \Sigma C_j \text{ (opt)}, L_{max}$$

➔ schedule all jobs according to SPT

If several jobs have the same processing time use EDD to order these jobs

➔ SPT/EDD rule

# Reversal of Priorities

- Reversal of priorities $1 \mid\mid L_{max}$ (opt), $\Sigma C_j$
  - ➔ Lmax is determined with EDD
  - ➔ $Z := L_{max}$

Transformation of this problem:

$$\overline{d}_j = d_j + z$$

new deadline — old due dates

# The Optimal Schedule

- Both problems are equivalent
  - The optimal schedule minimizes $\Sigma C_j$ and guarantees that each job completes by its deadline
    - In such a schedule job k is scheduled last if $\overline{d_k} \geq \sum_{j=1}^{n} p_j$

      $p_k \geq p_L$ for all L such that $\overline{d_L} \geq \sum_{j=1}^{n} p_j$

- **Proof:** If the first condition is not met, the schedule will miss a deadline
  - Pair wise exchange of job l and job k (not necessarily adjacent)
  - decreases $\Sigma C_j$ if the second condition is not valid for l and k

# Algorithm: Minimizing Total Completion Time with Deadlines

- Step 1 — Set $k = n$, $\tau = \sum_{j=1}^{n} p_j$ , $J^c = \{1, \ldots , n\}$
- Step 2 — Find $k^*$ in $J^c$ such that
  $\overline{d}_{k^*} \geq \tau$ and
  $p_{k^*} \geq p_l$ , for all jobs $l$ in $J^c$ such that $\overline{d}_l \geq \tau$ .
- Step 3 — Decrease $k$ by 1.
  Decrease $\tau$ by $p_{k^*}$
  Delete job $k^*$ from $J^c$ .
- Step 4 — If $k \geq 1$ go to Step 2, otherwise STOP.

# Example: Minimizing Total Completion Time with Deadlines

| jobs | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $p_j$ | 4 | 6 | 2 | 4 | 2 |
| $\bar{d}_j$ | 10 | 12 | 14 | 18 | 18 |

- $\tau = 18 \Rightarrow d_4 = d_5 = 18 \geq \tau$
  $p_4 = 4 > 2 = p_5$
  - ➔ last job : 4
  - ➔ $\tau = 18 - p_4 = 14 \Rightarrow d_3 = 14 \geq 14$ $\quad d_5 = 18 \geq 14$
    $p_5 = 2 = p_3$
  - ➔ either job can go in the now last position : 3
  - ➔ $\tau = 14 - p_3 = 12 \Rightarrow d_5 = 18 \geq 12$ $\quad d_2 = 12 \geq 12$
    $p_2 = 6 > 2 = p_5$
  - ➔ next last job 2
  - ➔ $\tau = 12 - p_2 = 6 \Rightarrow d_5 = 18 \geq 6$ $\quad d_1 = 10 \geq 12$
    $p_1 = 4 > 2 = p_5$
  - ➔ **sequence** **5 1 2 3 4 (3 1 2 5 4)**

# Multiple Objectives

- The optimal Schedule is always non preemptive even if preemptions are allowed

Multiple Objectives:

$$1 \mid \beta \mid \Theta_1 \gamma_1 + \Theta_2 \gamma_2$$

weighted sum of two (or more) objectives

$\gamma_1, \gamma_2$ objectives

normalization $\Theta_1 + \Theta_2 = 1$

# Pareto-Optimal Schedule

- A schedule is called pareto-optimal if it is not possible to decrease the value of one objective without increasing the value of the other

$\Theta_1 \rightarrow 0$ and $\Theta_2 \rightarrow 1$

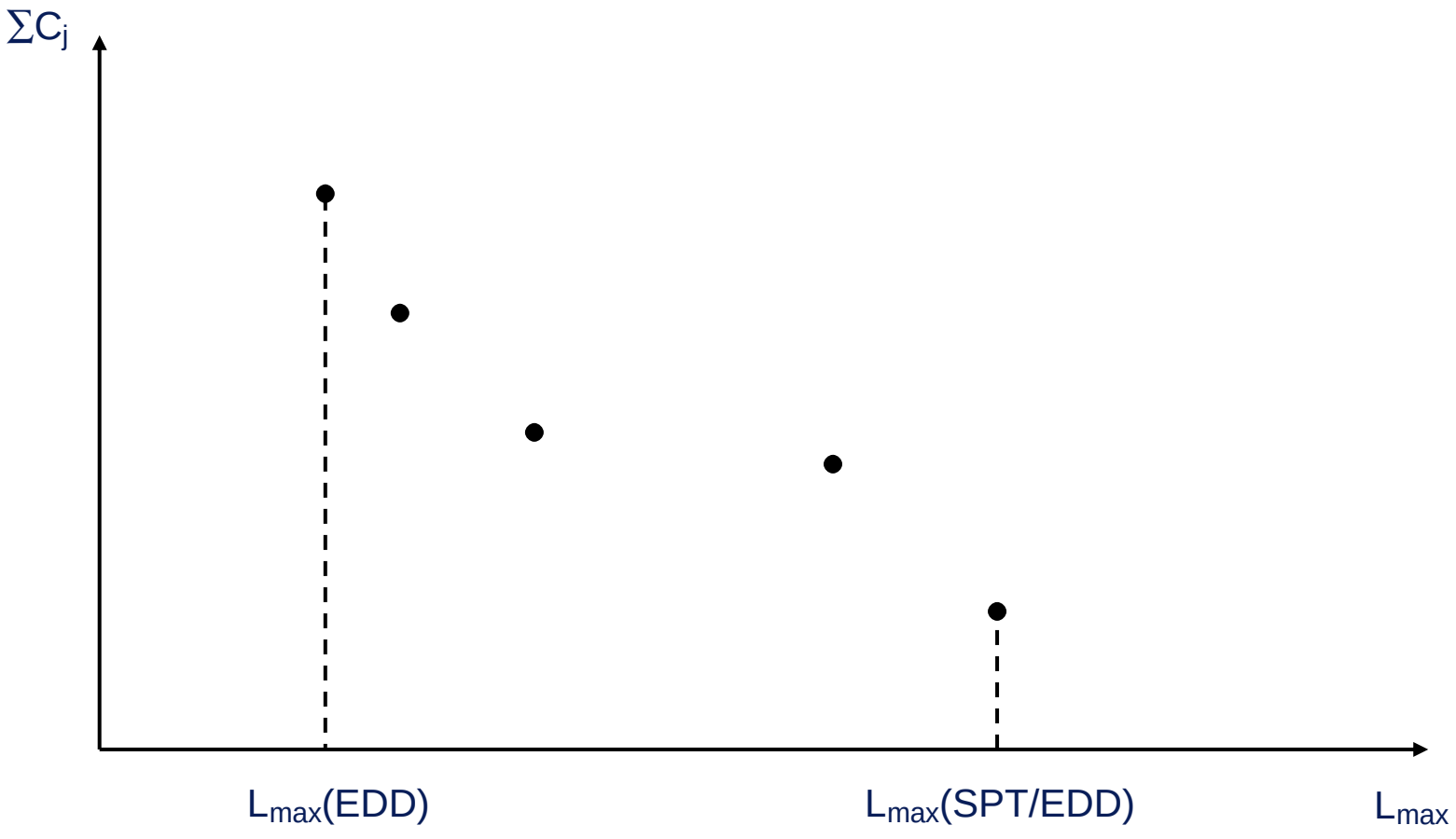
➔ $1|\beta|\Theta_1\gamma_1 + \Theta_2\gamma_2 \rightarrow 1$ $|\beta|\gamma_2(\text{opt}),\gamma_1$

$\Theta_1 \rightarrow 1$ and $\Theta_2 \rightarrow 0$

➔ $1|\beta|\Theta_1\gamma_1 + \Theta_2\gamma_2 \rightarrow 1$ $|\beta|\gamma_1(\text{opt}),\gamma_2$

$\gamma_1 : \sum C_j$ $\gamma_2 = L_{max}$

# Trade-offs between total completion time and maximum lateness



# Pareto-Optimal Solutions

- Generation of a pareto-optimal solutions

Find a new pareto-optimal solution:

Determine the optimal schedule, $L_{max}$

minimum increment of $L_{max}$

decrease the minimum $\Sigma C_j$

Similar to the minimization of the
total weighted completion time with deadlines

Start with the EDD schedule

end with SPT/EDD schedule

# Algorithm: Determining Trade-Offs between Total Completion Time and Maximum Lateness

- Step 1 — Set r = 1

  Set $L_{max} = L_{max}(EDD)$ and $\bar{d}_j = d_j + L_{max}$.

- Step 2 — Set k = n and $J^c = \{1, \ldots, n\}$.

  Set $\tau = \sum_{j=1}^{n} p_j$ and $\delta = \tau$.

- Step 3 — Find j* in $J^c$ such that

  $\bar{d}_{j^*} \geq \tau$ ,and

  $p_{j^*} \geq p_l$ for all jobs in $J^c$ such that $\bar{d}_l \geq \tau$.

  Put job j* in position k of the sequence.

- Step 4 — If there is no job l such that $\bar{d}_l < \tau$ and $p_l > p_{j^*}$ ,go to Step 5.

  Otherwise find j** such that

  $$\tau - \bar{d}_{j^{**}} = \min_l (\tau - \bar{d}_l)$$

  for all l such that $\bar{d}_l < \tau$ and $p_l > p_{j^*}$.

  Set $\delta^{**} = \tau - d_{j^{**}}$.

  If $\delta^{**} < \delta$ ,then $\delta = \delta^{**}$.

# Algorithm: Determining Trade-Offs between Total Completion Time and Maximum Lateness (2)

- Step 5 — Decrease k by 1.
  Decrease $\tau$ by $p_{j^*}$.
  Delete job $j^*$ from $J^c$.
  If $k \geq 1$ go to Step 3,
  otherwise go to Step 6.
- Step 6 — Set $L_{max} = L_{max} + \delta$.
  If $L_{max} > L_{max}$(SPT/EDD), then STOP.
  Otherwise set $r = r + 1$, $\bar{d}_j = \bar{d}_j + \delta$, and go to Step 2.

Maximum number of pareto – optimal points

➔ $n(n-1)/2 = O(n^2)$

Maximum number of pareto – optimal schedule $O(n \bullet \log(n))$

➔ Total complexity $O(n^3 \bullet \log(n))$

# Example: Determining Trade-Offs between Total Completion Time and Maximum Lateness

| jobs | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $p_j$ | 1 | 3 | 6 | 7 | 9 |
| $d_j$ | 30 | 27 | 20 | 15 | 12 |

- EDD sequence  5,4,3,2,1 $\Rightarrow L_{max}$ (EDD) = 2
  $c_3 = 22$  $d_3=20$
- SPT/EDD sequence  1,2,3,4,5 $\Rightarrow L_{max}$ (SPT/EDD) = 14
  $c_5 = 26$  $d_5 = 12$

# Example: Determing Trade-Offs between Total Completion Time and Maximum Lateness (2)

| Iteration r | $(\sum C_j, L_{max})$ | Pareto – optimal schedule | current $\tau + \delta$ | $\delta$ |
|---|---|---|---|---|
| 1 | 98, 2 | 5,4,3,1,2 | 32 29 22 17 14 | 1 |
| 2 | 77, 3 | 1,5,4,3,2 | 33 30 23 18 15 | 2 |
| 3 | 75, 5 | 1,4,5,3,2 | 35 32 25 20 17 | 1 |
| 4 | 64, 6 | 1,2,5,4,3 | 36 33 26 21 18 | 2 |
| 5 | 62, 8 | 1,2,4,5,3 | 38 35 28 23 20 | 3 |
| 6 | 60, 11 | 1,2,3,5,4 | 41 38 31 26 23 | 3 |
| 7 | 58, 14 | 1,2,3,4,5 | 44 41 34 29 26 | Stop |

- $1 \,||\, \Theta_1 \sum w_j C_j + \Theta_2 L_{max}$

  Extreme points can be determined in polynomial time (WSPT/EDD and EDD)
  - ➔ But the problem with arbitrary weights $\Theta_1$ and $\Theta_2$ is NP – hard.

# Parallel Machine Models

- 2 Step process
  - ➔Allocation of jobs to machines
  - ➔Sequence of the jobs on a machine
- Assumption: $p_1 \geq p_2 \geq \ldots \geq p_n$
- Simple problem: Pm || $C_{max}$
- Special case: P2 || $C_{max}$
  - ➔Equivalent to Partition (NP-hard in the ordinary sense)

# Heuristic algorithm

- Assign the remaining job with the longest processing time to the next free machine (LPT)
- Approximation factor: $\frac{C_{max}(LPT)}{C_{max}(OPT)} \leq \frac{4}{3} - \frac{1}{3m}$
- Contradiction: Counter example with smallest n
  - ➔ Property: The shortest job n is the
    - last job to start processing (LPT).
    - last job to finish processing.
  - ➔ If n is not the last job to finish processing then
    - deletion of n does not change $C_{max}$ (LPT)
    - but it may reduce $C_{max}$ (OPT)
  - ➔ A counter example with n – 1 jobs

# Heuristic algorithm (2)

- All machines are busy in time interval $[0, C_{max}(LPT) - p_n]$

starting time of job n

- $$C_{max}(LPT) - p_n \le \frac{\sum_{j=1}^{n-1} p_j}{m}$$

$$C_{max}(LPT) \le p_n + \frac{\sum_{j=1}^{n-1} p_j}{m} = p_n\left(1 - \frac{1}{m}\right) + \frac{\sum_{j=1}^{n} p_j}{m}$$

$$\frac{\sum_{j=1}^{n} p_j}{m} \le C_{max}(OPT)$$

# Heuristic algorithm (3)

$$\frac{4}{3}-\frac{1}{3m}<\frac{C_{max}(LPT)}{C_{max}(OPT)}\leq\frac{p_n(1-\frac{1}{m})+\sum_{j=1}^{n}p_j/m}{C_{max}(OPT)}$$

$$=\frac{p_n(1-\frac{1}{m})}{C_{max}(OPT)}+\frac{\sum_{j=1}^{n}p_j/m}{C_{max}(OPT)}\leq\frac{p_n(1-\frac{1}{m})}{C_{max}(OPT)}+1$$

➔ $$\frac{4}{3}-\frac{1}{3m}<\frac{p_n(1-\frac{1}{m})}{C_{max}(OPT)}+1$$

$C_{max}(OPT) < 3p_n$

- ➔ On each machine at most 2 jobs
- ➔ LPT is optimal for this case

# Example: A Worst Case Example for LPT

| jobs | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $p_j$ | 7 | 7 | 6 | 6 | 5 | 5 | 4 | 4 | 4 |

- 4 parallel machines
- $C_{max}(OPT) = 12$ 7+5; 7+5; 6+6; 4+4+4;
- $C_{max}(LPT) = 15$

| | | |
|---|---|---|
| 7 | 4 | 4 |
| 7 | 4 | |
| 6 | 5 | |
| 6 | 5 | |

$$\frac{C_{max}(LPT)}{C_{max}(OPT)} = \frac{15}{12} = \frac{5}{4} = \frac{4}{3} - \frac{1}{12} = \frac{16-1}{12}$$

➔ Tight factor Time complexity O(n•log(n))

# Example: A Worst Case Example for LPT (2)

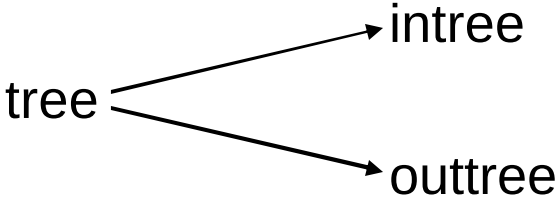
- Arbitrary ordering of jobs (any nondelay schedule)

$$\frac{C_{max}(LIST)}{C_{max}(OPT)} \leq 2 - \frac{1}{m}$$

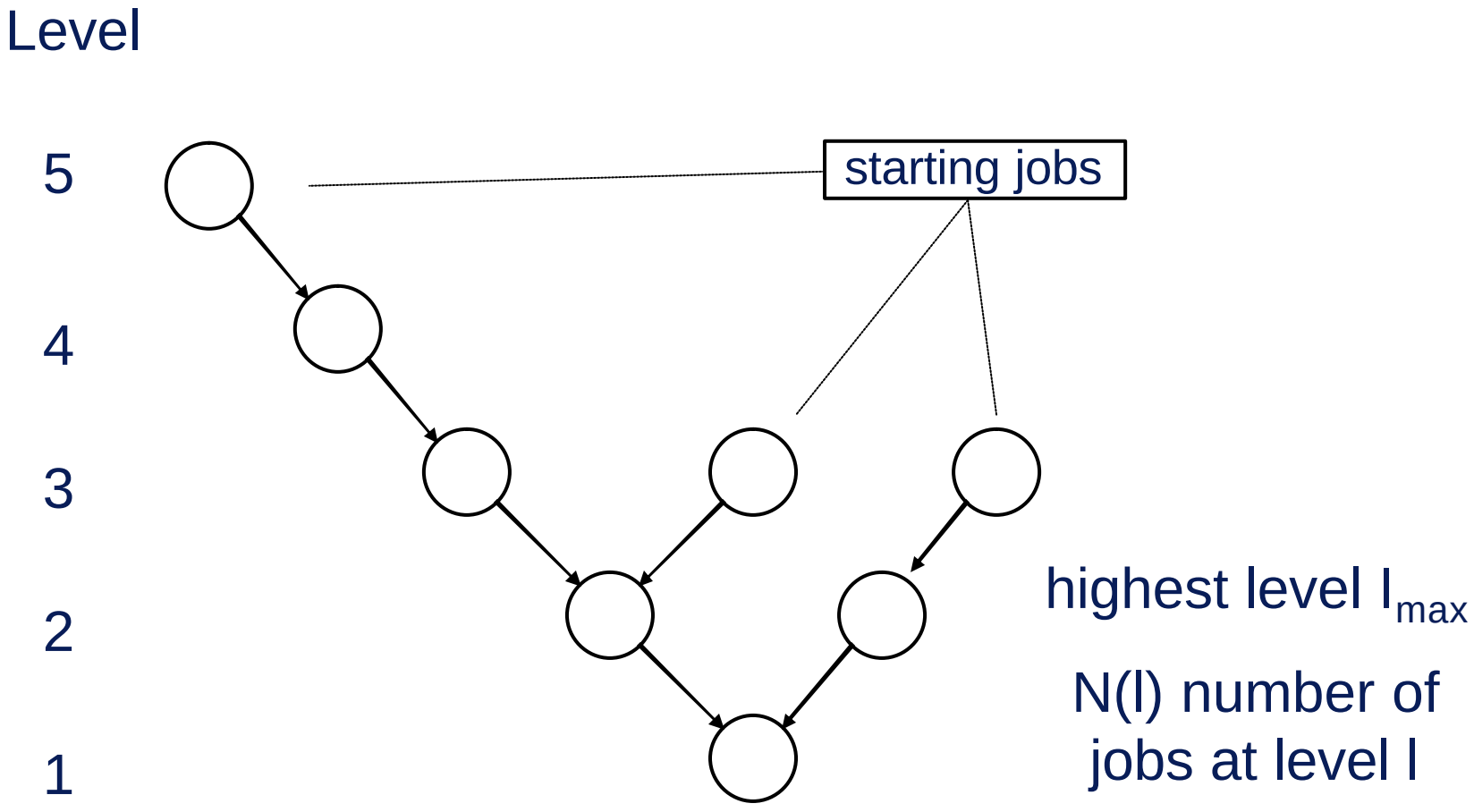
- There are better bounds for other algorithms.
- $P_m \mid prec \mid C_{max}$ $\Rightarrow$ is at least as hard as $P_m \mid \mid C_{max}$

  (strongly NP hard for $2 \leq m < \infty$)

- Special case $m \geq n \Rightarrow P_\infty \mid prec \mid C_{max}$
  - ➔ $P_m \mid p_j = 1, prec \mid C_{max} \Rightarrow$ NP hard
  - ➔ $P_m \mid p_j = 1, tree \mid C_{max} \Rightarrow$ easily solvable with Critical Path Method (CPM)



# Intree


Level
5
4
3
2
1
starting jobs
highest level $l_{max}$
N(l) number of jobs at level l

# Intree (2)

$$H(l_{max} + 1 - r) = \sum_{k=1}^{r} N(l_{max} + 1 - k)$$

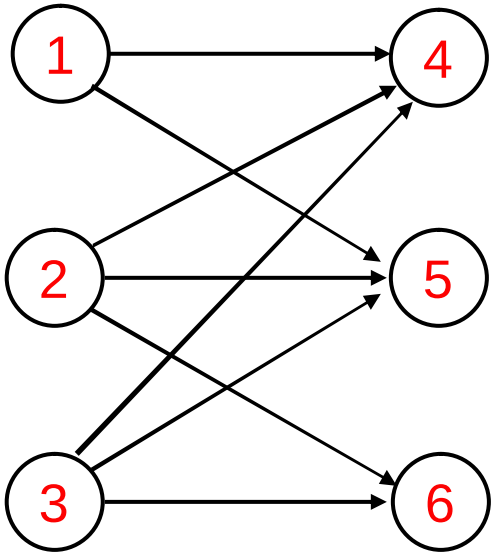
➔ the highest r levels

- Performance of CP algorithm for arbitrary precedence constraints

$$\frac{C_{max}(CP)}{C_{max}(OPT)} \leq \frac{4}{3} \quad \text{for 2 machines}$$

# Example: A Worst Case Example of CP

- 6 jobs, 2 machines




almost fully connected bipartite graph

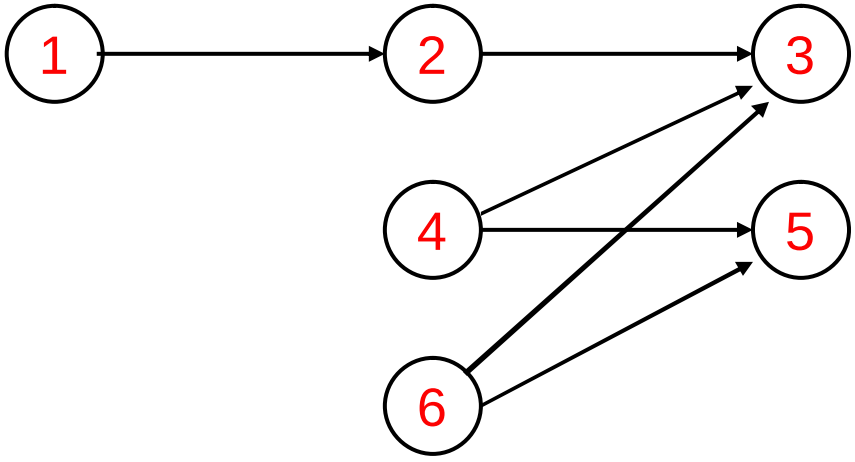
1: 1 5 2 1 5

2: 2 3 4 6 3 6 4

4 3

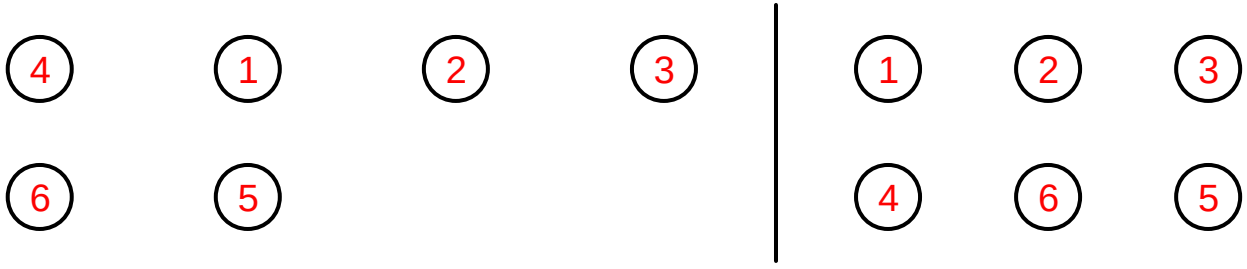
# Example: Application of LNS rule

- Alternative approach:
  - ➔ LNS: Largest Number of Successors first
  - ➔ optimal for in- and outtree




2 machines

1, 4, 6 : 2 successors



# Example: Application of LNS rule (2)

- Generalization to arbitrary processing times
- CP, LNS: largest total amount of processing

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

- $M_j$ are nested: 1 of 4 conditions is valid for jobs j and k.
  - ➔ $M_j = M_k$
  - ➔ $M_j \subset M_k$
  - ➔ $M_k \subset M_j$
  - ➔ $M_k \cap M_j = \varnothing$

# Theorem: The optimal LFJ rule

- Least Flexible Job first (LFJ)
- Machine is free ⟶ Pick the job that can be scheduled on the least number at machines
  - ➔ LFJ is optimal for $P_m \mid p_j = 1 , M_j \mid C_{max}$ if the $M_j$ are nested.
- **Proof by contradiction**

  j is the first job that violates LFJ rule.

  j* could be placed at the position of j

  by use of LFJ rules.
  - ➔ $M_j \cap M_{j^*} = \varnothing$ and $\mid M_{j^*} \mid < \mid M_j \mid$
    - ➔ $M_{j^*} \subset M_j$
  - ➔ Exchange of j* and j still results in an optimal schedule.
  - ➔ LFJ is optimal for $P2 \mid p_j = 1 , M_j \mid C_{max}$

# Example: Application of LFJ rule

- Consider $P4 \mid p_j = 1, M_j \mid C_{max}$ with eight jobs. The eight $M_j$ sets are:
  - $M_1 = \{1,2\}$
  - $M_2 = M_3 = \{1,3,4\}$
  - $M_4 = \{2\}$
  - $M_5 = M_6 = M_7 = M_8 = \{3,4\}$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| LFJ | 1 | 4 | 5 | 6 |
| | 2 | | 7 | 8 |
| | 3 | | | |
| | 2 | 1 | 5 | 7 |
| optimal | 3 | 4 | 6 | 8 |

- LFN (Least Flexible Machine) does not guarantee optimality for this example either.

# Linear Program

- $P_m \mid prmp \mid C_{max}$
- Preemptions $\Rightarrow$ often simpler analysis

$$\sum_{i=1}^{m} x_{ij} = p_j$$

$$\sum_{i=1}^{m} x_{ij} \leq C_{max}$$

processing less than makespan

$$\sum_{j=1}^{n} x_{ij} \leq C_{max}$$

$$x_{ij} > 0$$

processing time

makespan on each machine

# Makespan with Preemptions

- Solution of LP: Processing of each job on each machine $\Rightarrow$ Generation of a schedule

- Lower bound

$$C_{max} \geq \max\left\{p_1, \sum_{j=1}^{n} p_j / m\right\} = C^*_{max}$$

**Algorithm**

- Step 1: Processing all jobs on a single machine
  $\Rightarrow$ makespan $\leq m \bullet C^*_{max}$
- Step 2: Cut into m parts
- Step 3: Execution of each part on a different machine

# Example: Application of LRPT Rule

- Alternative:Longest Remaining Processing Time first (LRPT)
  - ➔ Infinite number of preemptions
- Example: 2 jobs $p_1 = p_2 = 1$ 1 machine

  Time period e: Switching after e
  - ➔ Makespan : 2 Total completion time 4 – e

    (Total completion time optimal case : 3)
- Vector of remaining processing times at time t

  $(p_1(t), p_2(t), .....p_n(t)) = \bar{p}(t)$

  $\bar{p}(t) \geq_m \bar{q}(t)$ $\bar{p}(t)$ majorizes $\bar{q}(t)$ if

  $$\sum_{j=1}^{k} p_{(j)}(t) \geq \sum_{j=1}^{k} q_{(j)}(t) \text{ for all } k = 1, .....n$$

  $p_j(t)$ : j the largest element of $\bar{p}(t)$

  $q_j(t)$ : j the largest element of $\bar{q}(t)$

# Example: Application of LRPT Rule (2)

- Example: (Vector Majorization)

  Consider the two vectors $\bar{p}(t) = (4, 8, 2, 4)$ and $\bar{q}(t) = (3, 0, 6, 6)$. Rearranging the elements within each vector and putting these in decreasing order results in vectors (8, 4, 4, 2) and (6, 6, 3, 0). It can be easily verified that $\bar{p}(t) \geq_m \bar{q}(t)$.

- If $\bar{p}(t) \geq_m \bar{q}(t)$ then LRPT applied to $\bar{p}(t)$ results in a larger or equal makespan than obtained by applying LRPT to $\bar{q}(t)$. (Discrete Times)

# Proof: Application of LRPT Rule

- Proof: Induction on the total amount of remaining processing
- Induction base: 1, 0, 0, ..... 0 and 1, 0, 0, ..... 0

$$\sum_{j=1}^{n} p_j(t+1) \le \sum_{j=1}^{n} p_j(t) - 1$$

$$\sum_{j=1}^{n} q_j(t+1) \le \sum_{j=1}^{n} q_j(t) - 1$$

$$\text{if } \bar{p}(t) \ge_m \bar{q}(t) \Rightarrow \bar{p}(t+1) \ge_m \bar{q}(t+1)$$

# Proof: Application of LRPT Rule (2)

- LPRT yields an optimal schedule for $P_m \mid prmp \mid C_{max}$

- Proof by induction on total amount of remaining processing.
- Induction base: Lemma holds for less than m
  jobs with 1 unit processing time left

➔ LRPT is optimal for $\sum_{j=1}^{n} p_j(t) \leq N-1$

Vector $\bar{p}(t)$ with $\sum_{j=1}^{n} p_j(t) = N$

# Proof: Application of LRPT Rule (3)

- Contraction: Another rule* is optimal at time t
  (but LPRT is valid from time t+1 award)

rule* $\bar{p}(t) \rightarrow \bar{p}'(t+1)$

LRPT $\bar{p}(t) \rightarrow \bar{p}(t+1)$

➔ $\bar{p}'(t+1) \geq_m \bar{p}(t+1)$

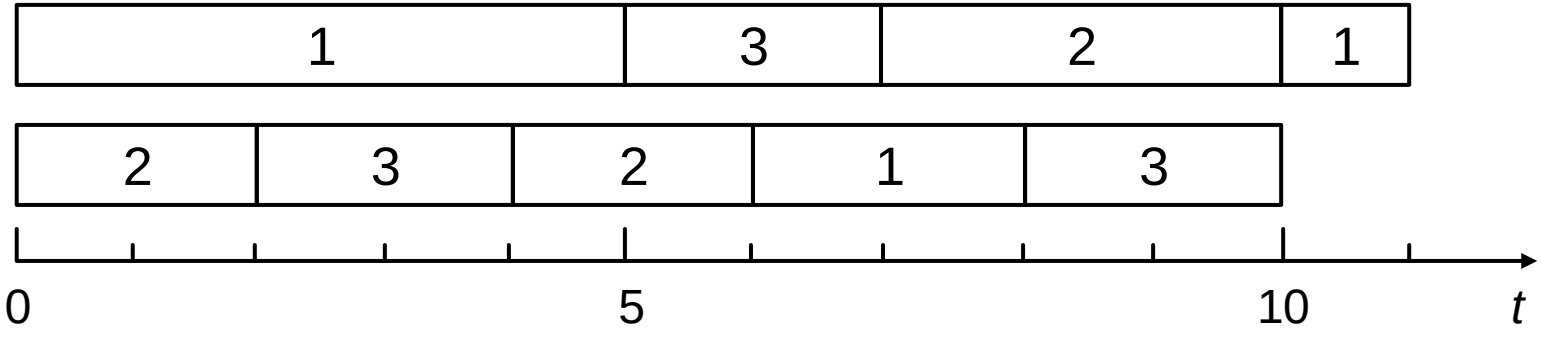
➔ Makespan of rule* is not smaller than makespan of LRPT

# Example: Application of LRPT in Discrete Time

- Consider two machines and three jobs 1, 2 and 3, with processing times 8, 7, and 6.

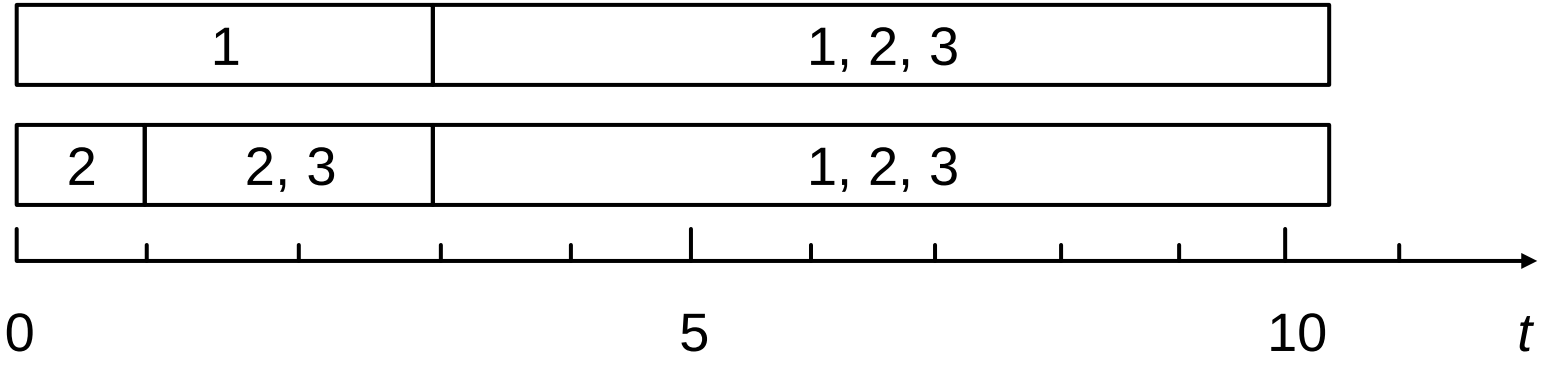
  The makespan is 11.

- $p_1 = 8$ $p_2 = 7$ $p_3 = 6$



# Example: Application of LRPT in Continuous Time

- Consider the same jobs as in the previous example.
  As preemptions may be done at any point in time, processor sharing takes place.
  The makespan is now 10.5.



Computer Engineering Institute

# Example: Application of LPRT in Continuous Time (2)

- $Q_m \mid prmp \mid C_{max}$
- Optimal schedule for $Q_m \mid prmp \mid C_{max}$

$$C_{max} \geq \max\left(\frac{p_1}{v_1}, \frac{p_1 + p_2}{v_1 + v_2}, \frac{\sum_{j=1}^{m-1} p_j}{\sum_{j=1}^{m-1} v_j}, \frac{\sum_{j=1}^{n} p_j}{\sum_{j=1}^{n} v_j}\right)$$

for $v_1 \geq v_2 \geq \ldots\ldots \geq v_m$

# Example: Application of LPRT in Continuous Time (3)

- Longest Remaining Processing Time on the Fastest Machine first (LRPT – FM) yields an optimal schedule for $Q_m \mid prmp \mid C_{max}$

- Discrete framework (speed and time)
  Replace machine i by $v_i$ machines of unit speed.
  A job can be processed on more than one machine in parallel if the machines are derived from the same machine.

# Example: Application of the LRPT-FM Rule

- 2 machines and 3 jobs
- processing times 8, 7, 6
- machine speed $v_1 = 2$, $v_2 = 1$

- Schedule:
  - Job 1 $\longrightarrow$ machine with two speed units
  - Job 2 $\longrightarrow$ machine with three speed unit
  - time 1 all jobs have a remaining processing time equal to 6
  - from time 1 each job occupies one speed unit machine
  - Makespan is equal to 7

- Continuous time: Multiplication of the processing times by a large number K and the speed by a large number V
- LRPT-FM rules yields optimal schedules if applied to $Q_m \mid r_j, prmp \mid C_{max}$

# The Total Completion Time Without Preemptions

- Total completion time without preemptions

  $P_{(j)}$: processing time of job in position j on a single machine

$$\sum C_j = n \bullet p_{(1)} + (n-1) \bullet p_{(2)} + ..... + 2 \bullet p_{(n-1)} + p_{(n)}$$

(nondelay schedule)

➔ $p_{(1)} \leq p_{(2)} \leq p_{(3)} \leq ..... \leq p_{(n-1)} \leq p_{(n)}$ for an optimal schedule

# The Total Completion Time Without Preemptions (2)

- SPT rule is optimal for $P_m \mid\mid \sum C_j$
- $\frac{n}{m}$ is integer (Otherwise add job with processing time 0)
  - ➔ n • m coefficients : m times n
    m times n – 1
    :
    m times 2
    m times 1
- Pm $\mid\mid \sum w_j C_j \Rightarrow$ NP hard

# Example: Application of WSPT Rule

| jobs | 1 | 2 | 3 |
|---|---|---|---|
| $p_j$ | 1 | 1 | 3 |
| $w_j$ | 1 | 1 | 3 |

- 2 machines
- 3 jobs
- Any schedule is WSPT
  - ➔ $w_1 = w_2 = 1 - \varepsilon \Rightarrow$ WSPT is not necessarily optimal
- Approximation factor $\frac{\sum w_j C_j(\text{WSPT})}{\sum w_j C_j(\text{OPT})} < \frac{1}{2}(1+\sqrt{2})$ (tight)
- $P_m \mid \text{prec} \mid \sum C_j$ : strongly NP-hard

# $P_m \mid p_j = 1, \text{outtree} \mid \sum C_j$

- CP rule is optimal for $P_m \mid p_j = 1, \text{outtree} \mid \sum C_j$
- Valid if at most m jobs are schedulable.
- $t_1$: last time instance CP is not applied ( instead CP‘)
- string 1: longest string not assigned at $t_1$
- string 2: shortest string assigned at $t_1$
- $C_1$‘: completion time of last job of string 1 under CP‘
- $C_2$‘: completion time of last job of string 2 under CP‘

- If $C_1' \geq C_2' + 1$ and machines are idle before $C_1' - 1 \Rightarrow$ CP is better
  Otherwise CP is worse than CP‘.
- CP rule is not optimal for intrees!

# LFJ rule is optimal for $P_m \mid p_j = 1, M_j \mid \sum C_j$

- The LFJ rule is optimal for $P_m \mid p_j = 1, M_j \mid \sum C_j$ if the $M_j$ sets are nested.
- Proof: see Makespan proof
- $P_m \mid p_j = 1, M_j \mid \sum C_j \propto R_m \mid\mid \sum C_j$
  $P_m \mid p_j = 1, M_j \mid \sum C_j \not\propto Q_m \mid\mid \sum C_j$
  $R_m \mid\mid \sum C_j$ : special integer program (solvable in polynomial time)

- $x_{ikj} = 1$ if job j is scheduled as the kth to last job on machine i.
- $x_{ikj} = 0$ otherwise

- Minimize $\sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{n} kp_{ij} x_{ikj}$

contribution of job j to $\sum C_j$

# LFJ rule is optimal for $P_m \mid p_j = 1, M_j \mid \sum C_j$

- Constraints: $\sum_{i=1}^{m} \sum_{k=1}^{n} x_{ikj} = 1$ $\quad j = 1, \ldots, n$

  each job is scheduled exactly once.

  $\sum_{j=1}^{n} x_{ikj} \leq 1$ $\quad i = 1, \ldots, m$; $k = 1, \ldots, n$

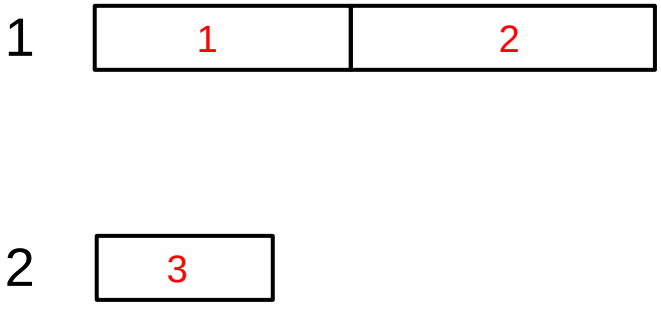
  each position is taken not more than once.

- $x_{ikj} \in \{0, 1\}$ No preemption of jobs
  - ➔ Weighted bipartite matching: n jobs $\Rightarrow$ n • m positions

- The optimal schedule may have unforced idleness.

# Example: Minimizing Total completion Time with Unrelated Machines

| jobs | 1 | 2 | 3 |
|---|---|---|---|
| $p_{1j}$ | 4 | 5 | 3 |
| $p_{2j}$ | 8 | 9 | 3 |




$$\sum C_j = 16$$

# Total Completion Time with Preemptions

- Problem $\mathbf{Q_m \mid prmp \mid \sum C_j}$
  - ➔ There exists an optimal schedule with
    $C_j \leq C_k$ if $p_j \leq p_k$ for all j and k.
- **Proof:** Pair wise exchange
- The SRPT-FM rule is optimal for $Q_m \mid prmp \mid \sum C_j$ .
- Shortest Remaining Processing Time on the Fastest Machine
  $v_1 \geq v_2 \geq \ldots \geq v_n$ $C_n \leq C_{n-1} \leq \ldots \leq C_1$
- There are n machines.
  - ➔ more jobs than machines $\Rightarrow$ add machines with speed 0
  - ➔ more machines than jobs $\Rightarrow$ the slowest machines are not used

# Proof: SRPT-FM rule is optimal for $Q_m \mid prmp \mid \sum C_j$

$$v_1 C_n = p_n$$
$$v_2 C_n + v_1(C_{n-1} - C_n) = p_{n-1}$$
$$v_3 C_n + v_2(C_{n-1} - C_n) + v_1(C_{n-2} - C_{n-1}) = p_{n-2}$$
$$\vdots$$
$$v_n C_n + v_{n-1}(C_{n-1} - C_n) + v_1(C_1 - C_2) = p_1$$

Adding these additions yields

$$v_1 C_n = p_n$$
$$v_2 C_n + v_1 C_{n-1} = p_n + p_{n-1}$$
$$v_3 C_n + v_2 C_{n-1} + v_1 C_{n-2} = p_n + p_{n-1} + p_{n-2}$$
$$\vdots$$
$$v_n C_n + v_{n-1} C_{n-1} + \ldots + v_1 C_1 = p_n + p_{n-1} + \ldots + p_1$$

# Proof: SRPT-FM rule is optimal for $Q_m \mid prmp \mid \sum C_j$ (2)

- S‘ is optimal $\Rightarrow C'_n \leq C'_{n-1} \leq \ldots \leq C'_1$
- $C'_n \geq p_n/v_1 \Rightarrow v_1 C'_n \geq p_n$
- Processing is done on jobs n and n –1
- $\leq (v_1 + v_2)C'_n + v_1(C'_{n-1} - C'_n)$
  - ➔ $v_2 C'_n + v_1 C'_{n-1} \geq p_n + p_{n-1}$
  - ➔ $v_k C'_n + v_{k-1} C'_{n-1} + \ldots + v_1 C'_{n-k+1} \geq p_n + p_{n-1} + \ldots + p_{n-k+1}$
  - ➔

$$v_1 C'_n \geq v_1 C_n$$

$$v_2 C'_n + v_1 C'_{n-1} \geq v_2 C_n + v_1 C_{n-1}$$

$$\vdots$$

$$v_n C'_n + v_{n-1} C'_{n-1} + \ldots + v_1 C'_1 \geq v_n C_n + v_{n-1} C_{n-1} + \ldots + v_1 C_1$$

# Proof: SRPT-FM rule is optimal for $Q_m \mid prmp \mid \sum C_j$ (3)

- Multiply inequality i by $\alpha_i \geq 0$ and obtain $\sum C'_j \geq \sum C_j \Rightarrow$ The proof is complete if those $\alpha_i$ exists.
  - ➔ $\alpha_i$ must satisfy

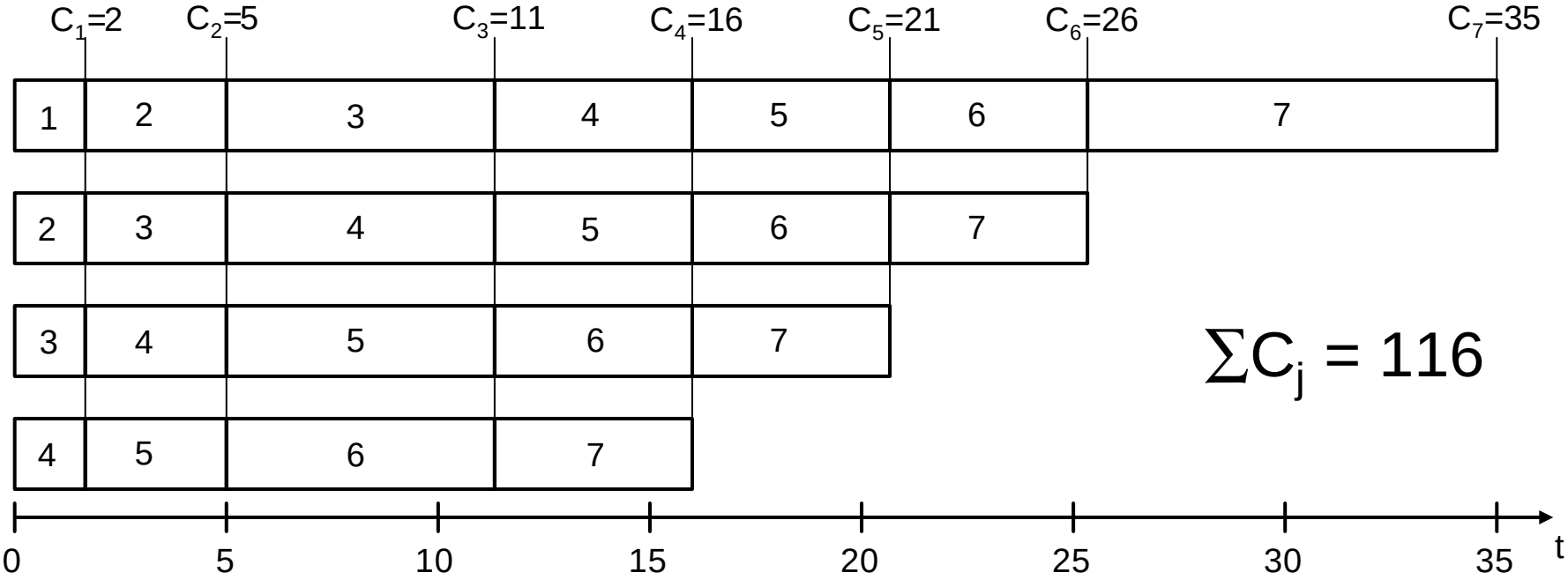
$$v_1\alpha_1 + v_2\alpha_2 + \ \dots \ + v_n\alpha_n = 1$$
$$v_1\alpha_2 + v_2\alpha_3 + \dots + v_{n-1}\alpha_n = 1$$
$$\vdots$$
$$v_1\alpha_n = 1$$

- Those $\alpha_i$ exists as $v_1 \geq v_2 \geq \dots \geq v_n$

# Example: Application of SRPT-FM Rule

| machines | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $v_i$ | 4 | 2 | 2 | 1 |

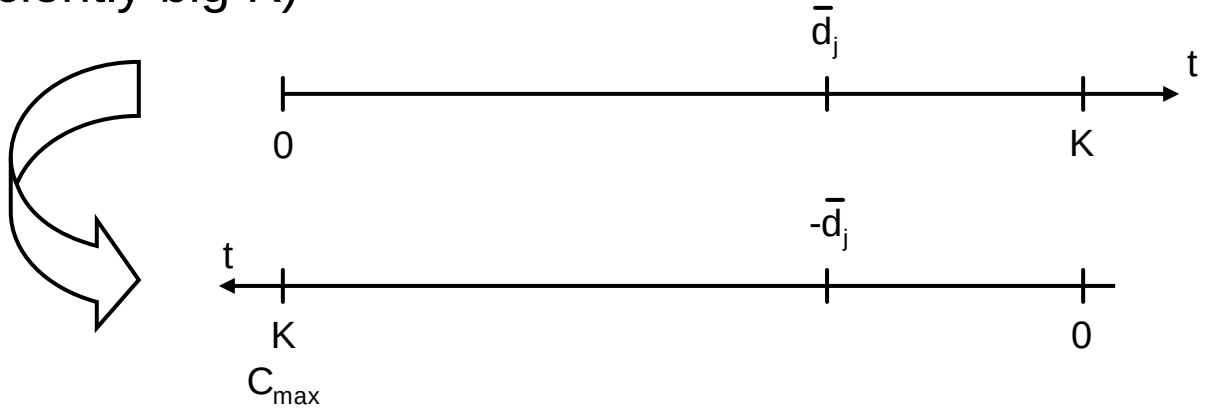
| jobs | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $p_j$ | 8 | 16 | 34 | 40 | 45 | 46 | 61 |




$\sum C_j = 116$

# Due – Date Related Objectives

- $P_m \mid\mid C_{max} \quad \propto \quad P_m \mid\mid L_{max} \Rightarrow$ NP – hard (all due dates 0)
- $Q_m \mid prmp \mid L_{max}$
- Assume $L_{max} = z$
  - ➔ $C_j \leq d_j + z \Rightarrow$ set $\bar{d}_j = d_j + z$ (hard deadline)
- Finding a schedule for this problem is equivalent to solving $Q_m \mid r_j, prmp \mid C_{max}$
- Reverse direction of time. Release each job j at $K - \bar{d}_j$ (for a sufficiently big K)




  - ➔ Solve the problem with LRPT- FM for $L_{max} \leq z$ and do a logarithmic search over z

# Example: Minimizing Maximum Lateness with Preemptions

| jobs | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $d_j$ | 4 | 5 | 8 | 9 |
| $p_j$ | 3 | 3 | 3 | 8 |

- P2 | prmp | $L_{max}$
- 4 jobs
- Is there a feasible schedule with $L_{max} = 0$ ? ($\bar{d}_j = d_j$)

| jobs | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $r_j$ | 5 | 4 | 1 | 0 |
| $P_j$ | 3 | 3 | 3 | 8 |

- Is there a feasible schedule with $C_{max} < 9$?

# Flow Shops and Flexible Flow Shops

- Each job must follow the same route ➔ sequence of machines
- Limited buffer space between neighboring machines

➔ Blocking can occur

- Main objective: makespan (related to utilization)
- First-com-first-served principle is in effect ➔ Jobs cannot pass each other: permutation flow shop

# Permutation Schedule

- Permutation Schedule: $j_1, j_2, \ldots, j_n$

$$C_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1} \qquad i = 1,\ldots,m$$

$$C_{1,j_k} = \sum_{l=1}^{k} p_{1,j_l} \qquad k = 1,\ldots,n$$

$$C_{i,j_k} = \max(C_{i-1,j_k}, C_{i,j_{k-1}}) + p_{i,j_k}$$

$$i = 2,\ldots,m \quad k = 2,\ldots,n$$

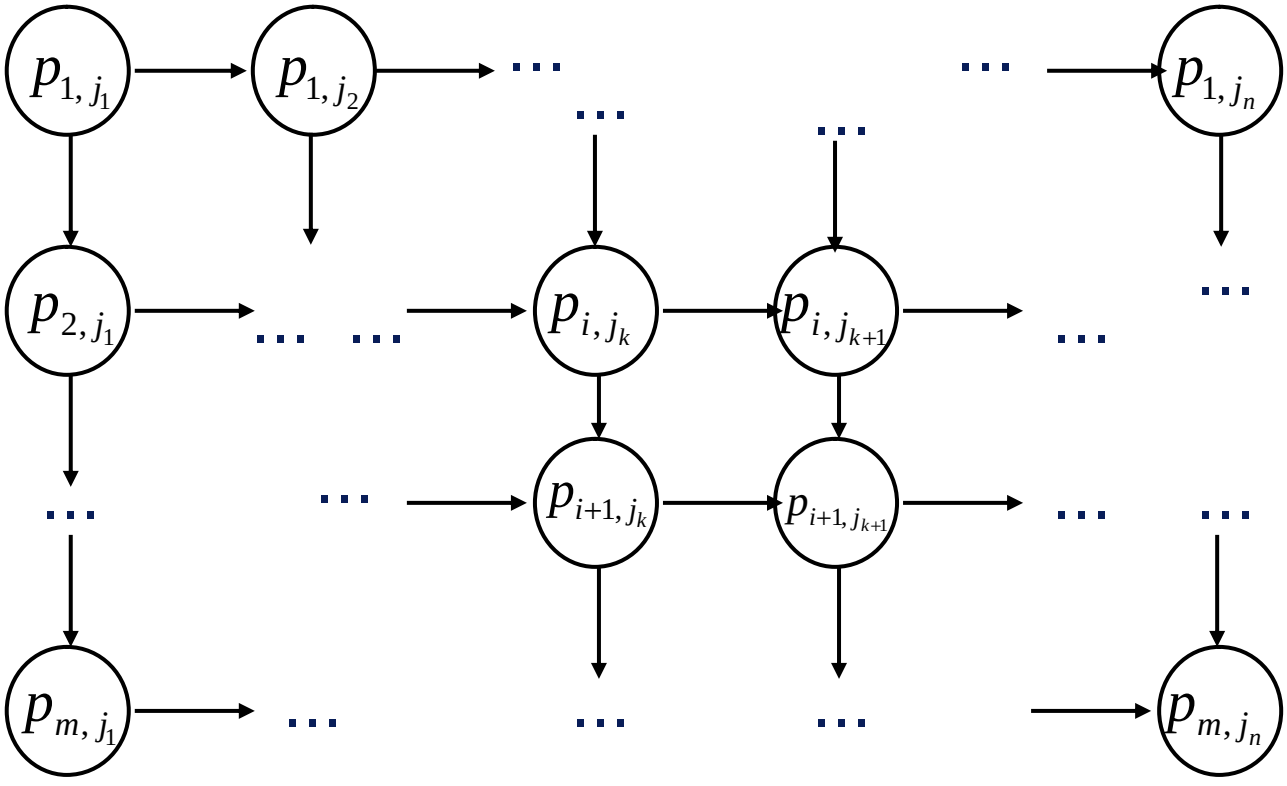
# Direct Graph for the Computation of the Makespan in $F_m|prmu|C_{max}$

- Consider 5 jobs on 4 machines with the following processing times

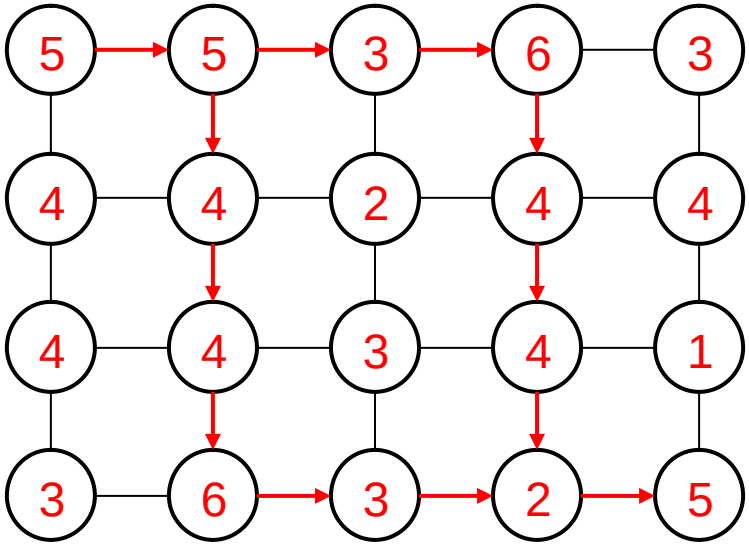
| jobs | $j_1$ | $j_2$ | $j_3$ | $j_4$ | $j_5$ |
|---|---|---|---|---|---|
| $p_{1,j_k}$ | 5 | 5 | 3 | 6 | 3 |
| $p_{2,j_k}$ | 4 | 4 | 2 | 4 | 4 |
| $p_{3,j_k}$ | 4 | 4 | 3 | 4 | 1 |
| $p_{4,j_k}$ | 3 | 6 | 3 | 2 | 5 |

Computer Engineering Institute

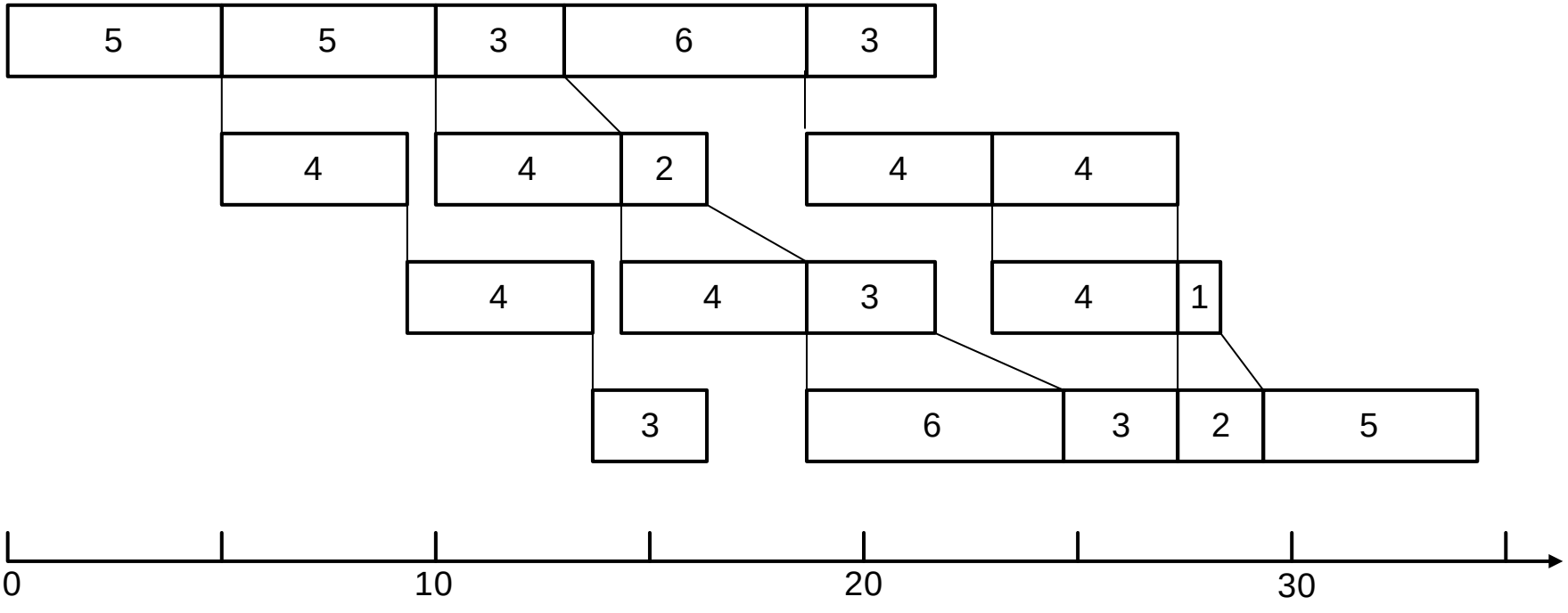
# Direct Graph for the Computation of the Makespan in $F_m|prmu|C_{max}$



# Directed Graph, Critical Paths and Gantt Chart (1)



# Directed Graph, Critical Paths and Gantt Chart (2)



# Critical Path

- The makespan is determined by a Critical path in the directed graph:
- Comparing 2 permutation flow shops with

  $p_{ij}^{(1)}$ $p_{ij}^{(2)}$ and $p_{ij}^{(1)} = p^{(2)}_{m+1-i,j}$

  ➔ Sequencing the jobs according to permutation $j_1, \ldots, j_n$ in the first flow shop produces the same makespan as permutation $j_n, \ldots, j_1$ in the second flow shop (Reversibility)

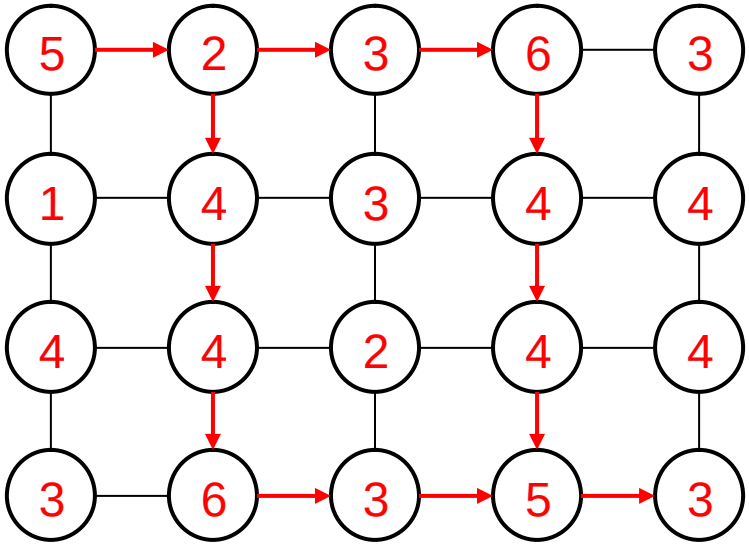
Computer Engineering Institute

# Example: Graph Representation and Reversibility

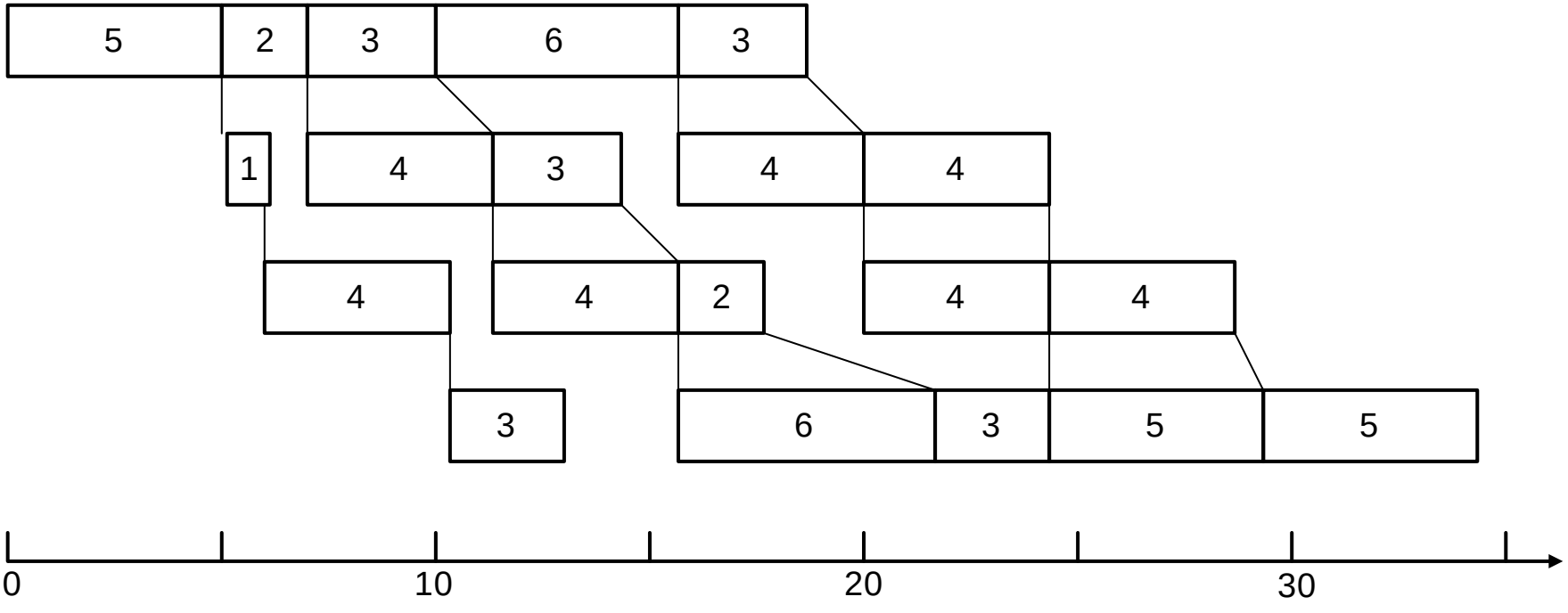
- Consider 5 jobs on 4 machines with the following processing times

| jobs | $j_1$ | $j_2$ | $j_3$ | $j_4$ | $j_5$ |
|---|---|---|---|---|---|
| $p_{1,j_k}$ | 5 | 2 | 3 | 6 | 3 |
| $p_{2,j_k}$ | 1 | 4 | 3 | 4 | 4 |
| $p_{3,j_k}$ | 4 | 4 | 2 | 4 | 4 |
| $p_{4,j_k}$ | 3 | 6 | 3 | 5 | 5 |

# Example: Graph Representation and Reversibility (2)



# Example: Graph Representation and Reversibility (3)



# SPT(1)-LPT(2) Schedule is optimal for $F2||C_{max}$

- Problem $F2||C_{max}$ with unlimited storage (optimal solution is always permutation)
- Johnson's rule produces an optimal sequence

  - Job partitioning into 2 sets

    Set I : all jobs with $p_{1j} \leq p_{2j}$

    Set II : all jobs with $p_{2j} < p_{1j}$

  - Set I : Those jobs are scheduled first in increasing order of $p_{1j}$ (SPT)
  - Set II : Those jobs are scheduled afterwards in decreasing order of $p_{2j}$ (LPT)

  ➔ SPT (1) – LPT(2) schedule

- Proof by pairwise adjacent interchange and contradiction
- Original schedule job l $\Rightarrow$ job j $\Rightarrow$ job k $\Rightarrow$ job h

  $C_{ij}$ : completion time (original schedule)

  $C'_{ij}$ : completion time (new schedule)

# SPT(1)-LPT(2) Schedule is optimal for $F2||C_{max}$ (2)

- Interchange of j and k
  - ➔ Starting time of job h on machine 1 is not affected ($C_{1l} + p_{1j} + p_{1k}$)

    Starting time of job h on machine 2:

    $C_{2k} = \max ( \max ( C_{2l}, C_{1l} + p_{1j}) + p_{2j}, C_{1l} + p_{1j} + p_{1k}) + p_{2k}$

    $= \max ( C_{2l} + p_{2j} + p_{2k},$

    $C_{1l} + p_{1j} + p_{2j} + p_{2k},$

    $C_{1l} + p_{1j} + p_{1k} + p_{2k})$

    $C'_{2j} = \max (C_{2l} + p_{2k} + p_{2j}, C_{1l} + p_{1k} + p_{2k} + p_{2j}, C_{1l} + p_{1k} + p_{1j} + p_{2j})$
- Assume another schedule is optimal
- Case 1 : j $\in$ Set II and k $\in$ Set I
  - ➔ $p_{1j} > p_{2j}$ and $p_{1k} \leq p_{2k}$
  - ➔ $C_{1l} + p_{1k} + p_{2k} + p_{2j} < C_{1l} + p_{1j} + p_{1k} + p_{2k}$

    $C_{1l} + p_{1k} + p_{1j} + p_{2j} \leq C_{1l} + p_{1j} + p_{2j} + p_{2k}$
  - ➔ $C'_{2j} \leq C_{2k}$

# Multiple Schedules That Are Optimal

- Case 2 : j , k $\in$ Set I and $p_{ij} > p_{1k}$
  - ➔ $p_{1j} \leq p_{2j}$ , $p_{1k} \leq p_{2k}$

$$\left.\begin{array}{l} C_{1l} + p_{1k} + p_{2k} + p_{2j} \\ \\ C_{1l} + p_{1k} + p_{1j} + p_{2j} \end{array}\right\} \leq C_{1l} + p_{1j} + p_{2j} + p_{2k}$$

- Case 3 : j , k $\in$ Set II and $p_{2j} < p_{2k}$
  - ➔ similar to Case 2
- There are many other optimal schedules besides SPT(1) – LPT(2) schedules
- $F_m \mid prmu \mid C_{max}$ : Formulation as a Mixed Integer Program (MIP)
- Decision variable $x_{jk} = 1$ if job j is the kth job in the sequence

# Auxiliary variables

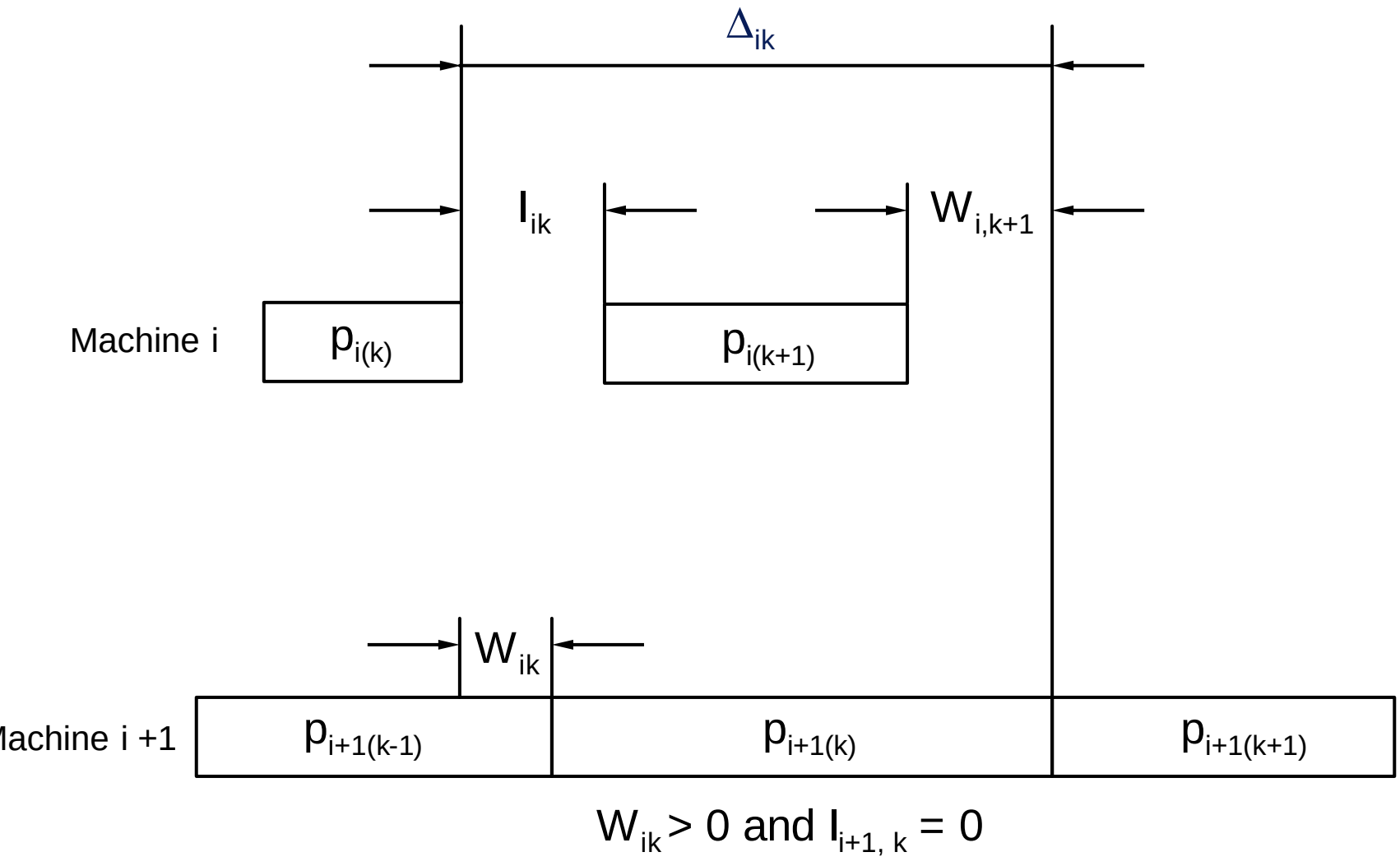
- $I_{ik}$ : amount of idle time on machine i between the processing of jobs in position k and k+1
- $W_{ik}$: amount of waiting time of job in position k between machines i and i+1

Relationship between $I_{ik}$ and $W_{ik}$

- $\Delta_{ik}$: difference between start time of job in position k+1 on machine i+1 and completion time of job in position k on machine i
- $p_{i(k)}$: processing time of job in position k on machine I

➔ $\Delta_{ik} = I_{ik} + p_{i(k+1)} + W_{i,k+1} = W_{ik} + p_{i+1(k)} + I_{i+1,k}$

# Constraints in the Integer Program Formulation




$W_{ik} > 0$ and $I_{i+1,\,k} = 0$

# Multiple Schedules That Are Optimal (2)

- Minimizing the makespan ≡ Minimizing the idle time on machine m

➔ $\sum_{i=1}^{m-1} p_{i(1)} + \sum_{j=1}^{n-1} I_{mj}$

earliest start time of job in position 1 at machine k

intermediate idle time

- Remember : $p_{i(k)} = \sum_{j=1}^{n} x_{jk} p_{ij}$

  ➔ there is only one job at position k!

# MIP

- MIP :

$$\min\left(\sum_{i=1}^{m-1}\sum_{j=1}^{n} x_{j1}p_{ij}\right)+\sum_{j=1}^{n-1} I_{mj}$$

- subject to

$$\sum_{j=1}^{n} x_{jk} = 1 \qquad k = 1, \ldots, n$$

$$\sum_{k=1}^{n} x_{jk} = 1 \qquad j = 1, \ldots, n$$

$$I_{ik} + \sum_{j=1}^{n} x_{j,k+1}p_{ij} + W_{i,k+1} - W_{ik} - \sum_{j=1}^{n} x_{jk}\,p_{i+1,j} - I_{i+1,k} = 0$$

$k = 1, \ldots, n-1;\ i = 1, \ldots, m-1$

$W_{i1} = 0$ $\quad i = 1, \ldots, m-1$ $\quad x_{jk} \in \{0,1\}$ $\quad j=1, \ldots,n$

$I_{1k} = 0$ $\quad k = 1, \ldots, n-1$ $\quad k=1, \ldots,m$

$W_{ik} \geq 0$ $\quad i = 1, \ldots, m-1;\ k = 1, \ldots, n$

$I_{ik} \geq 0$ $\quad i = 1, \ldots, m;\ k = 1, \ldots, n-1$

# $F3||C_{max}$ is strongly NP-hard

- $F3 \,||\, C_{max}$ is strongly NP-hard
- Proof by reduction from 3 – Partition
- An optimal solution for $F3 \,||\, C_{max}$ does not require sequence changes
  $\Rightarrow F_m \,|\, prmu \,|\, C_{max}$ is strongly NP – hard
- $F_m \,|\, prmu,\ p_{ij} = p_j \,|\, C_{max}$ : proportionate permutation flow shop
- The processing of job j is the same on each machine

- $$C_{max} = \sum_{j=1}^{n} p_j + (m-1)\max(p_1, \ldots, p_n) \quad \text{for}$$

  $F_m \,|\, prmu,\ p_{ij} = p_j \,|\, C_{max}$ (independent of the sequence)
  also true for $F_m \,|\, p_{rj} = p_j \,|\, C_{max}$

# Similarities between single machine and proportionate flow shop

- Similarities between single machine and proportionate (permutation) flow shop
  - $1 \| \sum C_j$ $F_m \mid prmu, p_{ij} = p_j \mid \sum C_j$
  - $1 \| \sum U_j$ $F_m \mid prmu, p_{ij} = p_j \mid \sum U_j$
  - $1 \| h_{max}$ $F_m \mid prmu, p_{ij} = p_j \mid h_{max}$
  - $1 \| \sum T_j$ $F_m \mid prmu, p_{ij} = p_j \mid \sum T_j$ (pseudo polynomial algorithm)
  - $1 \| \sum w_j T_j$ $F_m \mid prmu, p_{ij} = p_j \mid \sum w_j T_j$ (elimination criteria)

- Slope index $A_j$ for job j $A_j = -\sum_{i=1}^{m} (m - (2i - 1))p_{ij}$

- Sequencing of jobs in decreasing order of the slope index

# Example: Application of Slope Heuristic

- Consider 5 jobs on 4 machines with the following processing times

| jobs | $j_1$ | $j_2$ | $j_3$ | $j_4$ | $j_5$ |
|---|---|---|---|---|---|
| $p_{1,j_k}$ | 5 | 2 | 3 | 6 | 3 |
| $p_{2,j_k}$ | 1 | 4 | 3 | 4 | 4 |
| $p_{3,j_k}$ | 4 | 4 | 2 | 4 | 4 |
| $p_{4,j_k}$ | 3 | 6 | 3 | 5 | 5 |

$$A_1 = -(3 \times 5) - (1 \times 4) + (1 \times 4) + (3 \times 3) = -6$$

$$A_2 = -(3 \times 5) - (1 \times 4) + (1 \times 4) + (3 \times 6) = +3$$

$$A_3 = -(3 \times 3) - (1 \times 2) + (1 \times 3) + (3 \times 3) = +1$$

$$A_4 = -(3 \times 6) - (1 \times 4) + (1 \times 4) + (3 \times 2) = -12$$

$$A_5 = -(3 \times 3) - (1 \times 4) + (1 \times 1) + (3 \times 5) = +3$$

# Example: Application of Slope Heuristic (2)

- F2 || $\sum C_j$ is strongly NP – hard
  - ➔ Fm | prmu | $\sum C_j$ is strongly NP – hard
    as sequence changes are not required in the optimal schedule for 2 machines

# Flow Shops with Limited Intermediate Storage

- Assumption: No intermediate storage, otherwise one storage place is modeled as machine on which all jobs have 0 processing time
- Fm | block | $C_{max}$
- $D_{ij}$ : time when job j leaves machine i $\qquad D_{ij} = C_{ij}$
- For sequence $j_1, \ldots, j_n$ the following equations hold

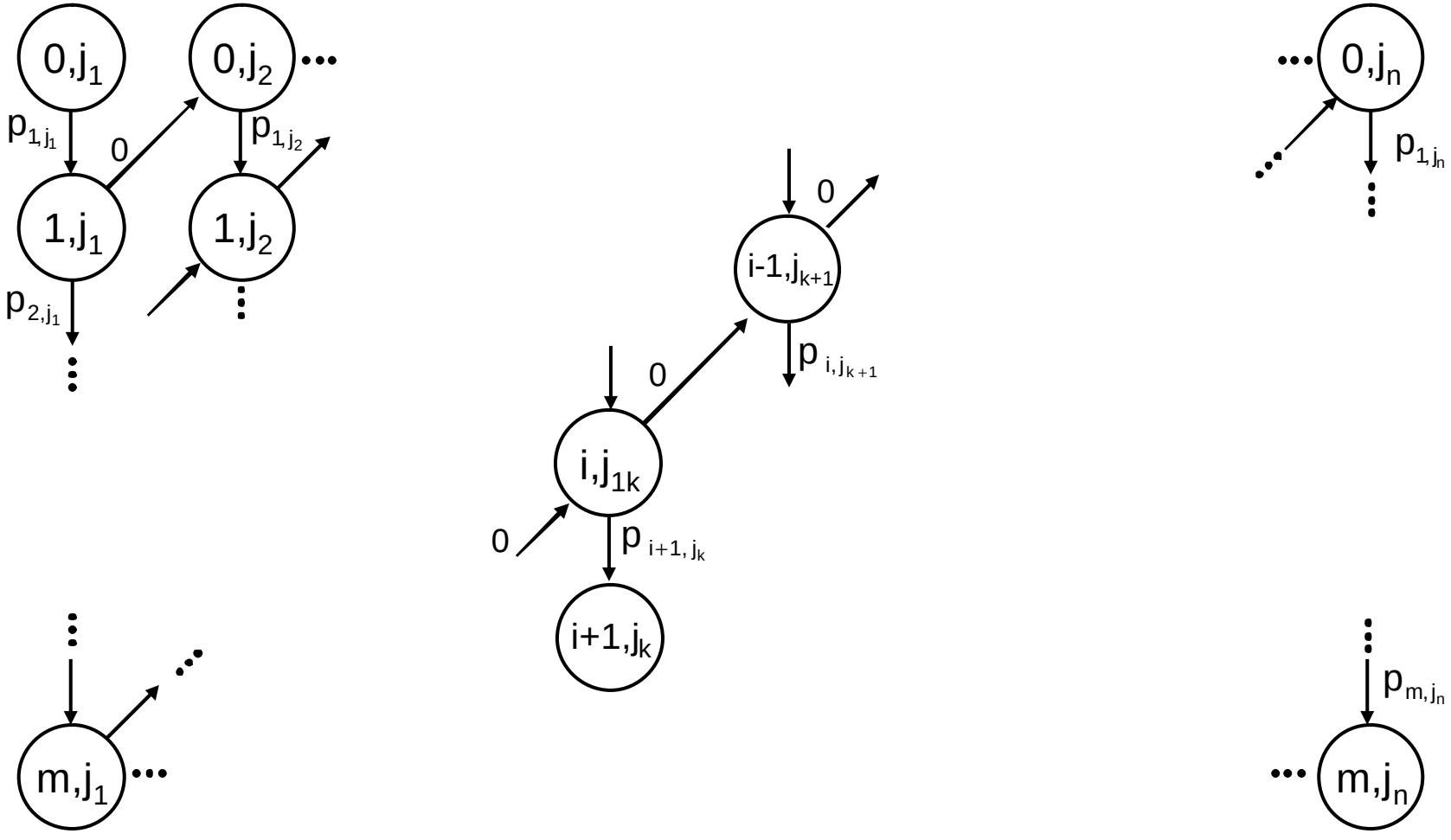
$$D_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1}$$

$$D_{i,j_k} = \max( D_{i-1,j_k} + p_{i,j_k}, D_{i+1,j_{k-1}} )$$

$$D_{m,j_k} = D_{m-1,j_k} + p_{m,j_k}$$

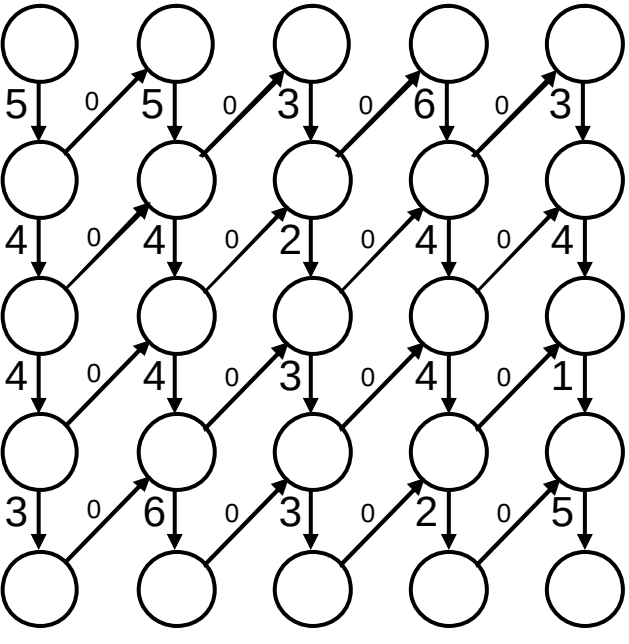
  - ➔ Critical path in a directed graph
    Weight of node $(i, j_k)$ specifies the departure time of job $j_k$ from machine i
    Edges have weights 0 or a processing time

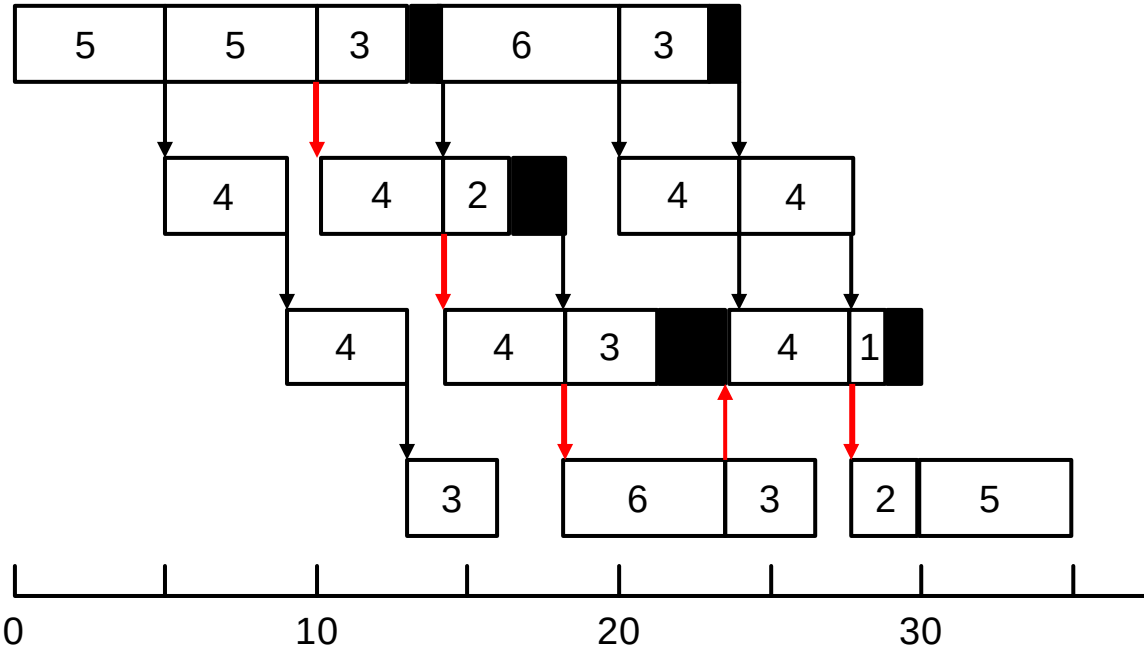
# Directed Graph for the Computation of the makespan



# Graph Representation of a Flow Shop with Blocking

| jobs | $j_1$ | $j_2$ | $j_3$ | $j_4$ | $j_5$ |
|---|---|---|---|---|---|
| $p_{1,j_k}$ | 5 | 5 | 3 | 6 | 3 |
| $p_{2,j_k}$ | 4 | 4 | 2 | 4 | 4 |
| $p_{3,j_k}$ | 4 | 4 | 3 | 4 | 1 |
| $p_{4,j_k}$ | 3 | 6 | 3 | 2 | 5 |



# Flow Shops with Limited Intermediate Storage (2)

- The reversibility result holds as well:
- If $p_{ij}^{(1)} = p^{(2)}_{m+1-l,j}$ then sequence $j_1, \ldots, j_n$ in the first flow shop has the same makespan as sequence $j_n, \ldots, j_1$ in the second flow shop
- F2 | block | $C_{max}$ is equivalent to Traveling Salesman problem with n+1 cities
- When a job starts its processing on machine 1 then the proceeding job starts its processing on machine 2
  - ➔ time for job $j_k$ on machine 1

    $\max( p_{1,j_k}, p_{2,j_{k-1}} )$

Exception: The first job j* in the sequence spends time $p_{1,j^*}$ on machine 1

Distance from city j to city k

$d_{0k} = p_{1k}$

$d_{j0} = p_{2j}$

$d_{jk} = \max (p_{2j}, p_{1k})$

# Example: A Two Machine Flow Shop with Blocking and the TSP

- Consider 4 job instance with processing times

| jobs | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $P_{1,j}$ | 2 | 3 | 3 | 9 |
| $P_{2,j}$ | 8 | 4 | 6 | 2 |

- Translates into a TSP with 5 cities

| cities | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $b_{,j}$ | 0 | 2 | 3 | 3 | 9 |
| $a_{,j}$ | 0 | 8 | 4 | 6 | 2 |

- There are two optimal schedules
- 1 4 2 3 $\Rightarrow$ $0 \rightarrow 1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 0$ and
- 1 4 3 2

# Example: A Two Machine Flow Shop with Blocking and the TSP (2)

- Compansion SPT(1) – LPT(2) schedules for unlimited buffers:

  1, 3, 4, 2;  1, 2, 3, 4  and  1, 3, 2, 4

- F3 | block | $C_{max}$ is strongly NP – hard and cannot be described as a travelling salesman problem

# Special cases of Fm | block | $C_{max}$

- Special cases of Fm | block | $C_{max}$
  - ➔ Proportionate case: Fm | block, $p_{ij} = p_j$ | $C_{max}$
- A schedule is optimal for Fm | block, $p_{ij} = p_j$ | $C_{max}$ if and only if it is an SPT- LPT schedule
- Proof :

$$\underbrace{C_{max} \geq \sum_{j=1}^{n} p_j + (m-1)\max(p_1,\ldots,p_n)}_{\text{optimal makespan with unlimited buffers}}$$

- Proof – concept:
  - Any SPT-LPT schedule matches the lower bound
  - Any other schedule is strictly greater than the lower bound

# A schedule is optimal for Fm | block, $p_{ij} = p_j$ | $C_{max}$ if and only if it is an SPT- LPT schedule

- SPT – part: A job is never blocked
- LPT – part: No machine must ever wait for a job
  - ➔ The makespan of an SPT – LPT schedule is identical to an SPT – LPT schedule for unlimited buffers.
- Second part of the proof by contradiction

  The job $j_k$ with longest processing time spents contributes m time its processing time to the makespan
- If the schedule is no SPT- LPT schedule
  - ➔ a job $j_h$ is positioned between two jobs with a longer processing time
  - ➔ this job is either blocked in the SPT part or the following jobs cannot be processed on machine m without idle time in between

# Heuristic for Fm | block | $C_{max}$ Profile Fitting (PF)

- Local optimization
  - Selection of a first job (e.g. smallest sum of processing time)
  - Pick the first job as next, that wastes the minimal time on all m machines.
  - Using weights to weight the idle times on the machines depending the degree of congestion

# Example: Application of PF Heuristic (unweighted PF)

| jobs | $j_1$ | $j_2$ | $j_3$ | $j_4$ | $j_5$ |
|---|---|---|---|---|---|
| $p_{1,j_k}$ | 5 | 5 | 3 | 6 | 3 |
| $p_{2,j_k}$ | 4 | 4 | 2 | 4 | 4 |
| $p_{3,j_k}$ | 4 | 4 | 3 | 4 | 1 |
| $p_{4,j_k}$ | 3 | 6 | 3 | 2 | 5 |

- First job: job 3 (shortest total processing time)
- Second job : job 1 2 4 5
  idle time 11 11 15 3
  - ➔ job 5
  - ➔ Sequence: 3 5 1 2 4 makespan 32

    makespan for unlimited storage
  - ➔ optimal makespan
- First job: job 2 (largest total processing time)
  - ➔ Sequence: 2 1 3 5 4 makespan 35

but

F2 | block | $C_{max}$ = F2 | nwt | $C_{max}$

Fm | block | $C_{max}$ ? Fm | nwt | $C_{max}$

# Flexible Flow Shop with Unlimited Intermediate Storage

- Proportionate case

  $FF_c \mid p_{ij} = p_j \mid C_{max}$

  non preemptive

  LPT heuristic

  ➔ NP hard

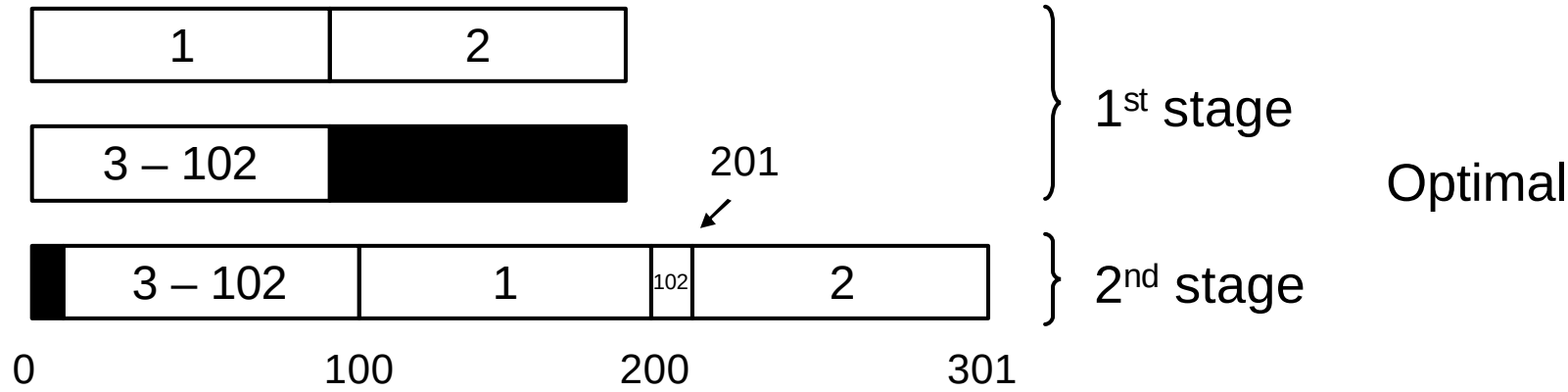
  preemptive

  LRPT heuristic

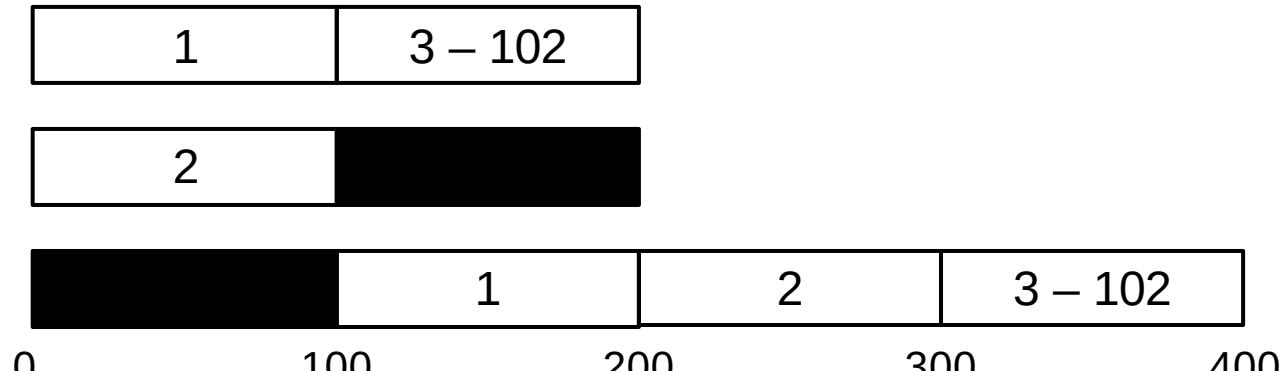
  optimal for a single stage

# Example: Minimizing Makespan with LPT

- $p_1 = p_2 = 100$ $p_3 = p_4 = \ldots = p_{102} = 1$
- 2 stages: 2 machines at first stage
  1 machine at second stage




Optimal schedule




LPT heuristic

# Flexible Flow Shop with Unlimited Intermediate Storage (2)

- $FF_c \mid p_{ij} = p_j \mid \sum C_j$
- SPT is optimal for a single stage and for any numbers of stage with a single machine at each stage
- SPT rule is optimal for $FF_c \mid p_{ij} = p_j \mid \sum C_j$ if each stage has at least as many machines as the preceeding stage
- Proof:

  Single stage SPT minimizes $\sum C_j$ and the sum of the starting times

  $\sum (C_j - p_j)$

  c stages: $C_j$ occurs not earlier than $cp_j$ time units after its starting time at the first stage

  Same number of machines at each stage:

  SPT: each need not wait for processing at the next stage

  ➔ $\sum_{j=1}^{n} C_j$ = sum of the starting times + $\sum_{j=1}^{n} cp_j$

# Job Shops

- **The route of every job is fixed but not all jobs follow the same route**
- J2 || $C_{max}$
- $J_{1,2}$ : set of all jobs that have to be processed first on machine 1
- $J_{2,1}$ : set of all jobs that have to be processed first on machine 2
- Observation:

  If a job from $J_{1,2}$ has completed its processing on machine 1 the postponing of its processing on machine 2 does not matter as long as machine 2 is not idle.
- A similar observation hold for $J_{2,1}$
  - ➔ a job from $J_{1,2}$ has a higher priority on machine 1 than any job form $J_{2,1}$ and vice versa
- Determing the sequence of jobs from $J_{1,2}$
  - ➔ F2 || $C_{max}$ : SPT(1) – LPT(2) sequence
  - ➔ machine 1 will always be busy
- J2 || $C_{max}$ can be reduced to two F2 || $C_{max}$ problems

# Representation as a disjunctive graph G

- Jm || $C_{max}$ is strongly NP hard
- Representation as a disjunctive graph G

Set of nodes N : Each node corresponds to an operation (i, j) of job j on machine i

Set of conjunctive edges A: An edge from (i, j) to (k, j) denotes that job j must be processed on machine k immediately after it is processed on machine i

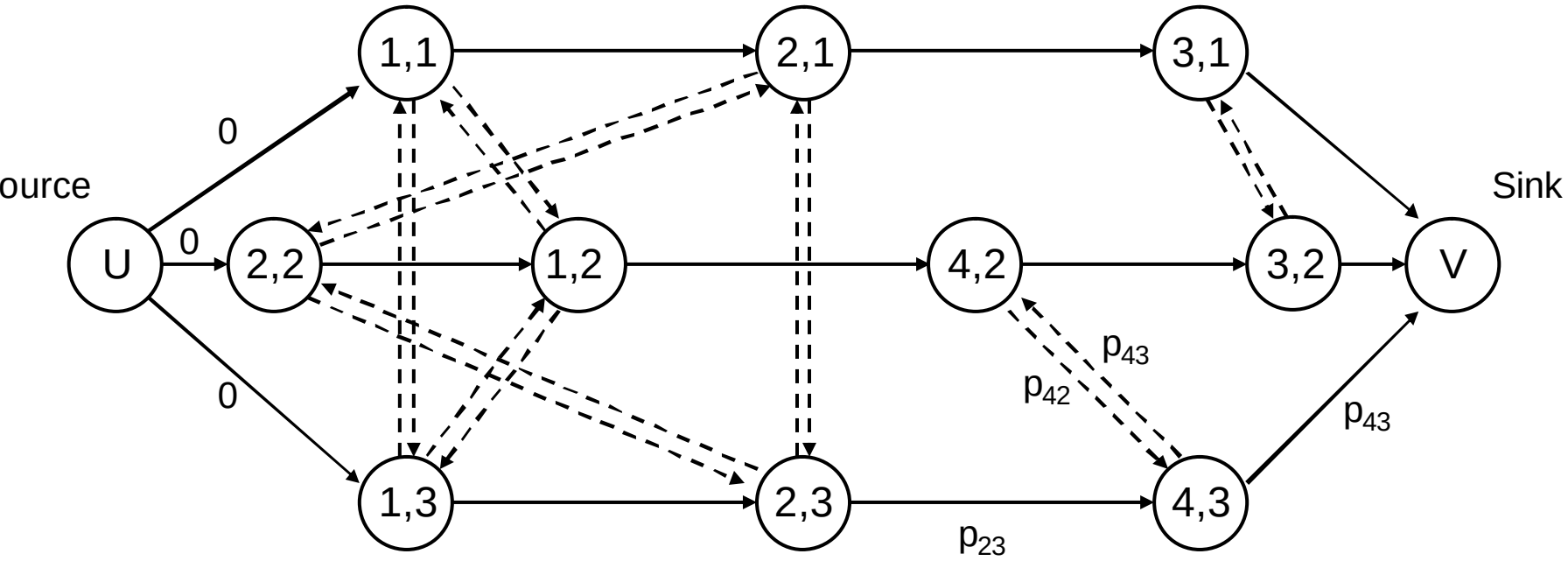
Set of disjunctive edges B: There is a disjunctive edge from any operation (i, j) to any operation (i, h), that is, between any two operations that are executed on the same machine

➔ All disjunctive edges of a machine form a cliques of double arcs

Each edge has the processing time of its origin node as weight

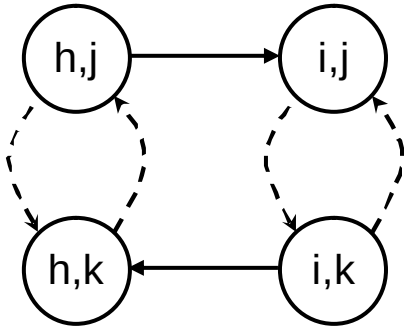
# Directed Graph for Job Shop with Makespan as Objective

- There is a dummy source node U connected to the first operation of each job. The edges leaving U have the weight 0.
- There is a dummy sink node V, that is the target of the last operation of each job.



# Feasible schedule

- Feasible schedule: Selection of one disjunctive edge from each pair of disjunctive edges between two nodes such that the resulting graph is acyclic
- Example




- D: set of selective disjunctive edges
- G(D): Graph including D and all conjunctive edges
- Makespan of a feasible schedule: Longest path from U to V in G(D)
  - ➔ 1. Selection of the disjunctive edges D
  - ➔ 2. Determination of the critical path

# Disjunctive Programming Formulation

- $y_{ij}$: starting time of operation (i,j)
- Minimize $C_{max}$ subject to

$y_{kj} = y_{ij} + p_{ij}$ if $(i,j) \rightarrow (k,j)$ is a conjunctive edge

$C_{max} = y_{ij} + p_{ij}$ for all operations (i,j)

$y_{ij} = y_{il} + p_{il}$ or

$y_{il} = y_{ij} + p_{ij}$ for all (i,l) and (i,j) with i = 1, …, m

$y_{ij} = 0$ for all operations (i,j)

# Example: Disjunctive Programming Formulation

- 4 machines , 3 jobs

| jobs | machine sequence | processing times |
|---|---|---|
| 1 | 1, 2, 3 | $p_{11} = 10, p_{21} = 8, p_{31} = 4$ |
| 2 | 2, 1, 4, 3 | $p_{22} = 8, p_{12} = 3, p_{42} = 5, p_{32} = 6$ |
| 3 | 1, 2, 4 | $p_{13} = 4, p_{23} = 7, p_{43} = 3$ |

- $y_{21} = y_{11} + p_{11} = y_{11} + 10$
- $C_{max} = y_{11} + p_{11} = y_{11} + 10$
- $y_{11} = y_{12} + p_{12} = y_{12} + 3$ or $y_{12} = y_{11} + p_{11} = y_{11} + 10$

# Branch and Bound Method to Determine all Active Schedules

- $\Omega$ :set of all schedulable operations (predecessors of these operations are already scheduled),
- $r_{i,j}$ :earliest possible starting time of operation

  $(i, j) \in \Omega$

- $\Omega' \subseteq \Omega$
- $t(\Omega)$ smallest starting time of a operation

# Generation of all Active Schedules

- Step 1: (Initial Conditions) Let $\Omega$ contain the first operation of each job; Let $r_{ij} = 0$, for all $(i, j) \in \Omega$
- Step 2: (machine selection) compute for the current partial schedule

$$t(\Omega) = \min_{(i,j)\in\Omega}\{r_{ij} + p_{ij}\}$$

  and let i* denote the machine on which the minimum is achieved.
- Step 3: (Branching) Let $\Omega'$ denote the set of all operations (i*,j) on machine i* such that

$$r_{i^*j} \le t(\Omega)$$

For each operation in $\Omega'$, consider an (extended) partial schedule with that operation as the next one on machine i*.

For each such (extended) partial schedule, delete the operation from $\Omega$, include its immediate follower in $\Omega$, and return to Step 2.

# Generation of all Active Schedules

- Result: Tree with each active schedule being a leaf
- A node v in this tree: partial schedule

➔ Selection of disjunctive edges to describe the order of all operations that are predecessors of $\Omega$

- An outgoing edge of v: Selection of an operation $(i^*, j) \in \Omega'$ as the next job on machine i*

➔ The number of edges leaving node v = number of operations in $\Omega'$

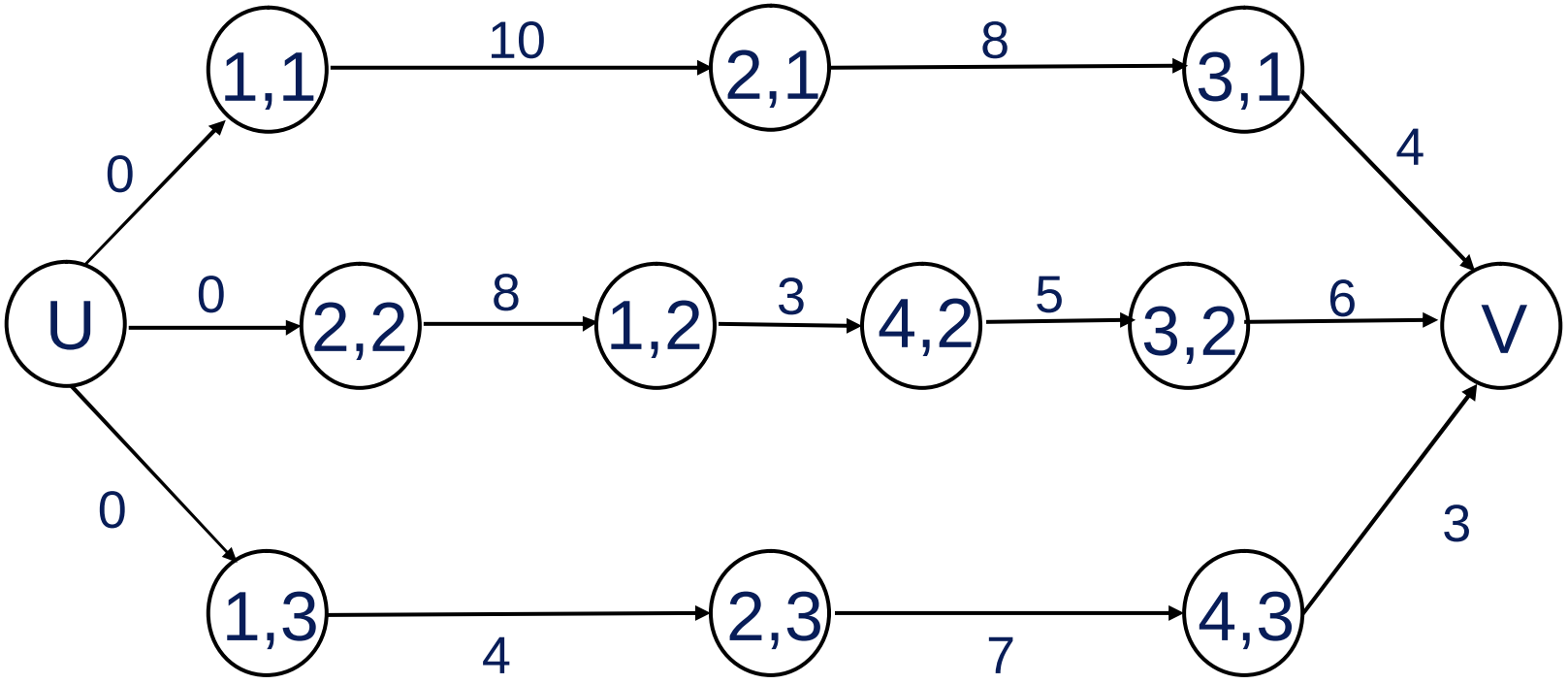
- v': successor of v

➔ Set D' of the selected disjunctive edges at v' ➔ G(D')

# Lower Bound for Makespan at v'

- simple lower bound: critical path in graph G(D')
- complex lower bound:
  - ➔ critical path from the source to any unscheduled operation: release date of this operation
  - ➔ critical path form any unscheduled operation to the sink: due date of this operation
  - ➔ Sequencing of all unscheduled operations on the appropriate machine for each machine separately
    - ➔ $1 \mid r_j \mid L_{max}$ for each machine (strongly NP-hard)
      - ➔ Reasonable performance in practice

# Application of Branch and Bound



# Application of Branch an Bound Level 1

- Initial graph: only conjunctive edges

➔ Makespan: 22

- Level 1:

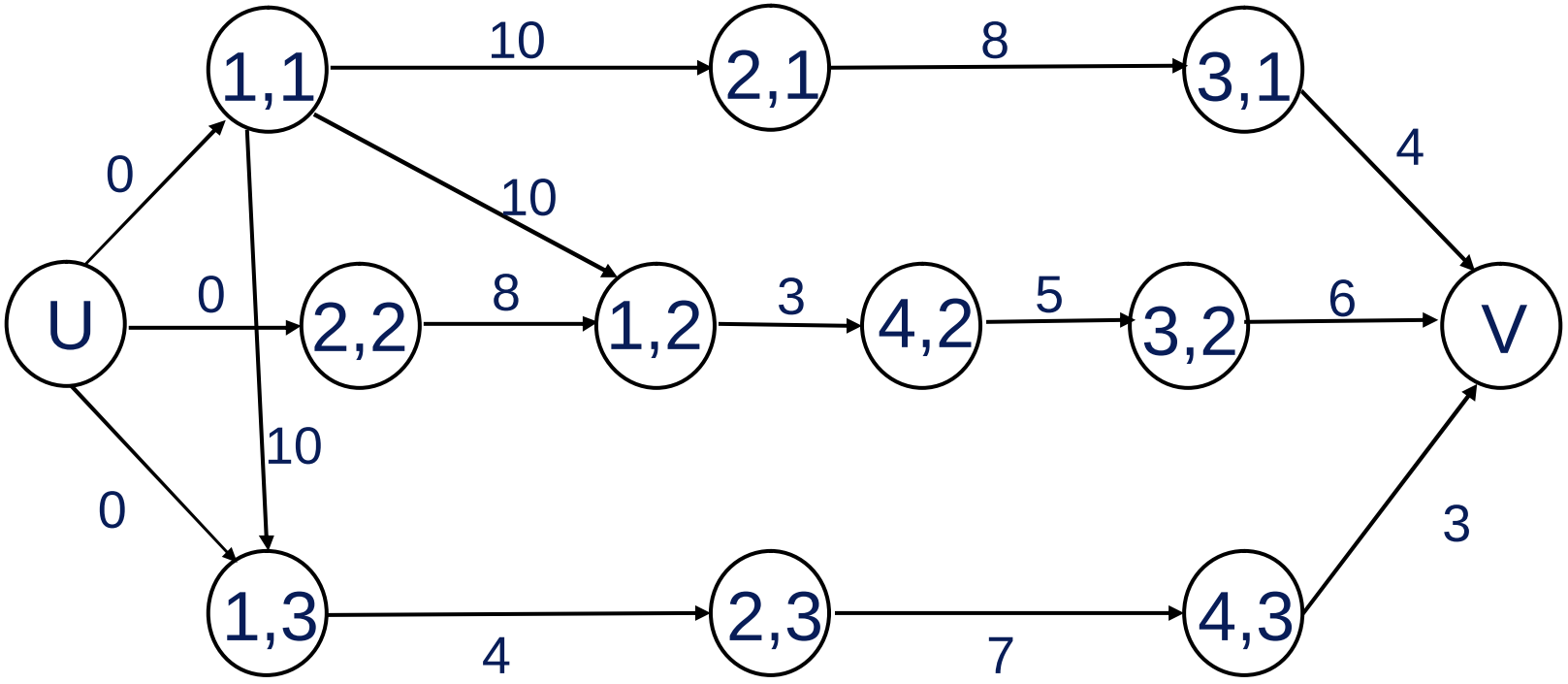
$$\Omega = \{(1,1),(2,2),(1,3)\}$$

$$t(\Omega) = \min(0+10, 0+8, 0+4) = 4$$

$$i^* = 1$$

$$\Omega' = \{(1,1),(1,3)\}$$

# Schedule Operation (1,1) first



# Schedule Operation (1,1) first

- 2disjunctive edges are added
- (1,1) → (1,2)
- (1,1) → (1,3)

→ Makespan: 24

# Schedule Operation (1,1) first

- Improvements of lower bound by generating an instance of $1 \mid r_j \mid L_{max}$ for machine 1

| jobs | 1 | 2 | 3 |
|---|---|---|---|
| $p_{ij}$ | 10 | 3 | 4 |
| $r_{ij}$ | 0 | 10 | 10 |
| $d_{ij}$ | 12 | 13 | 14 |

- $L_{max}$ =3 with sequence 1,2,3
- Makespan: 24+3=27

# Schedule Operation (1,1) first

- Instance of 1 | $r_j$ | $L_{max}$ for machine 2

| jobs | 1 | 2 | 3 |
|---|---|---|---|
| $p_{ij}$ | 8 | 8 | 7 |
| $r_{ij}$ | 10 | 0 | 14 |
| $d_{ij}$ | 20 | 10 | 21 |

- $L_{max}$ = 4 with sequence 2,1,3
- Makespan: 24+4 = 28

# Schedule operation (1,3) first

- 2 disjunctive edges are added ➔ Makespan: 26
- $1 \mid r_j \mid L_{max}$ for machine 1
- $L_{max} = 2$ with sequence 3, 1, 2

➔ Makespan: 26+2=28

# Application of Branch an Bound Level 2

- Level 2: Branch from node (1,1)

$$\Omega = \{(2,2),(2,1),(1,3)\}$$

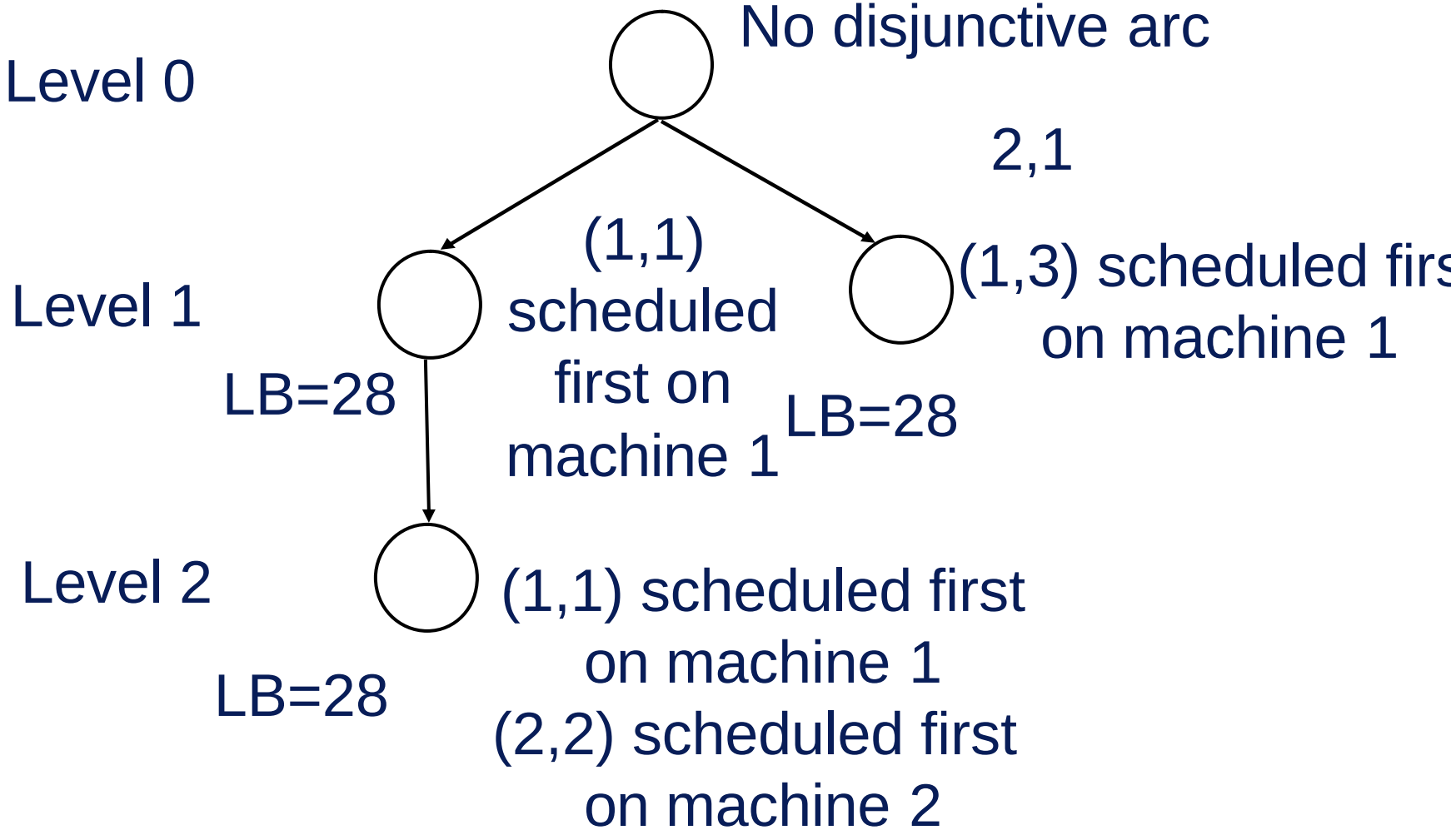
$$t(\Omega) = \min(0+8, 10+8, 10+4) = 8$$

$$i^* = 2$$

$$\Omega' = \{(2,2)\}$$

- There is only one choice
- (2,2) → (2,1); (2,2) → (2,3)
- Two disjunctive edges are added

# Branching Tree

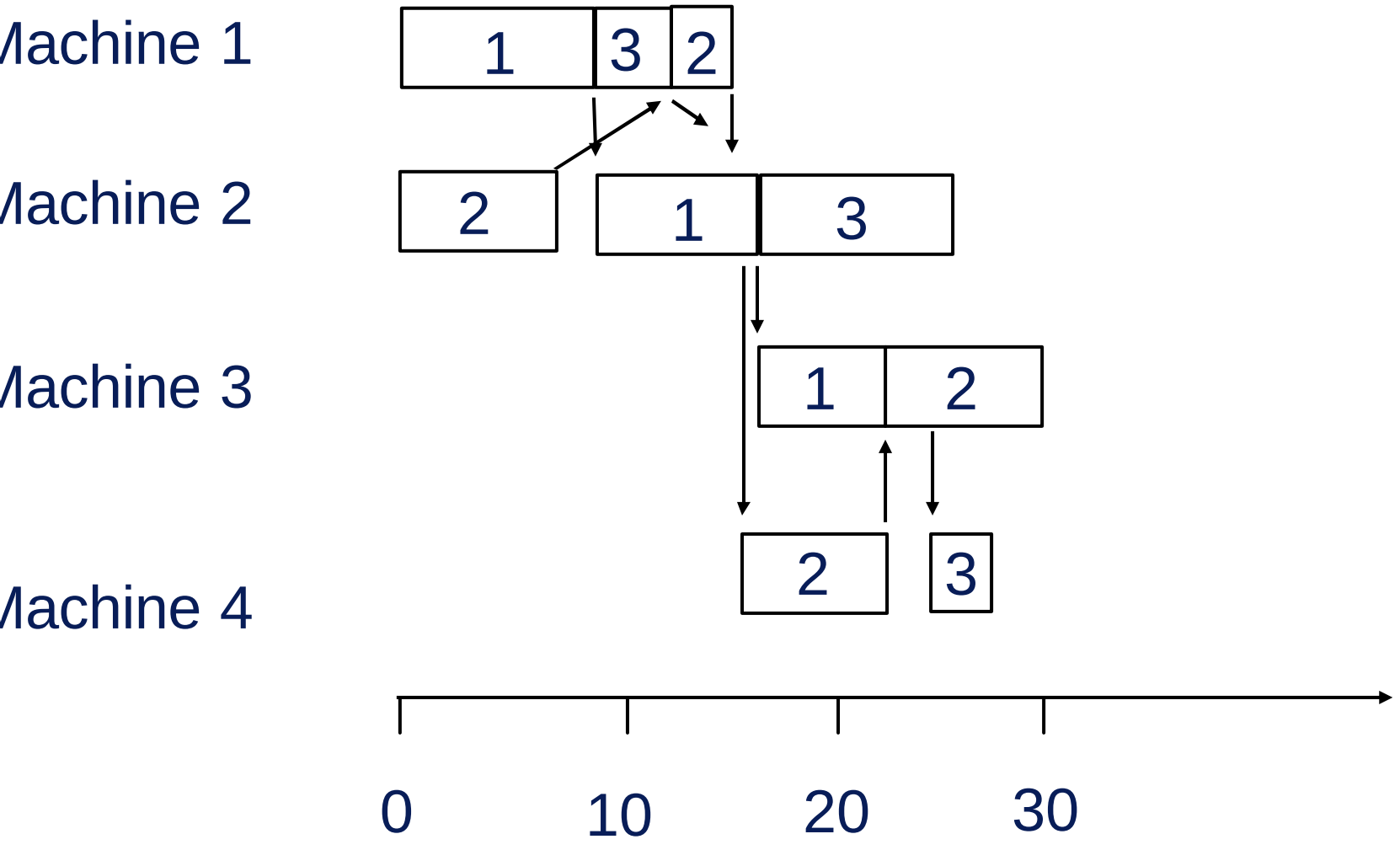


# Continuation of the Procedure yields

| machine | job sequence |
|---|---|
| 1 | 1 3 2 (or 1 2 3) |
| 2 | 2 1 3 |
| 3 | 1 2 |
| 4 | 2 3 |

Makespan: 28

# Gantt chart for J4 || $C_{max}$



# Shifting Bottleneck Heuristic

- A sequence of operations has been determined for a subset $M_0$ of all m machines.
  - ➔ disjunctive edges are fixed for those machines
- Another machine must be selected to be included in $M_0$: Cause of severest disruption (bottleneck)
- All disjunctive edges for machines not in $M_0$ are deleted $\rightarrow$ Graph G'

  Makespan of G' : $C_{max}(M_0)$
  - ➔ for each operation (i, j) with i $\notin$ $M_0$ determine release date and due date
  - ➔ allowed time window
- Each machine not in $M_0$ produces a separate 1 | $r_j$ | $L_{max}$ problem
  - ➔ $L_{max}(i)$: minimum $L_{max}$ of machine i
- Machine k with the largest $L_{max}(i)$ value is the bottleneck
  - ➔ Determination of the optimal sequence for this machine $\rightarrow$ Introduction of disjunctive edges
  - ➔ Makespan increase from $M_0$ to $M_0 \cup \{k\}$ by at least $L_{max}(k)$

# Example: Shifting Bottleneck Heuristic

- Resequencing of the operation of all machines in $M_0$

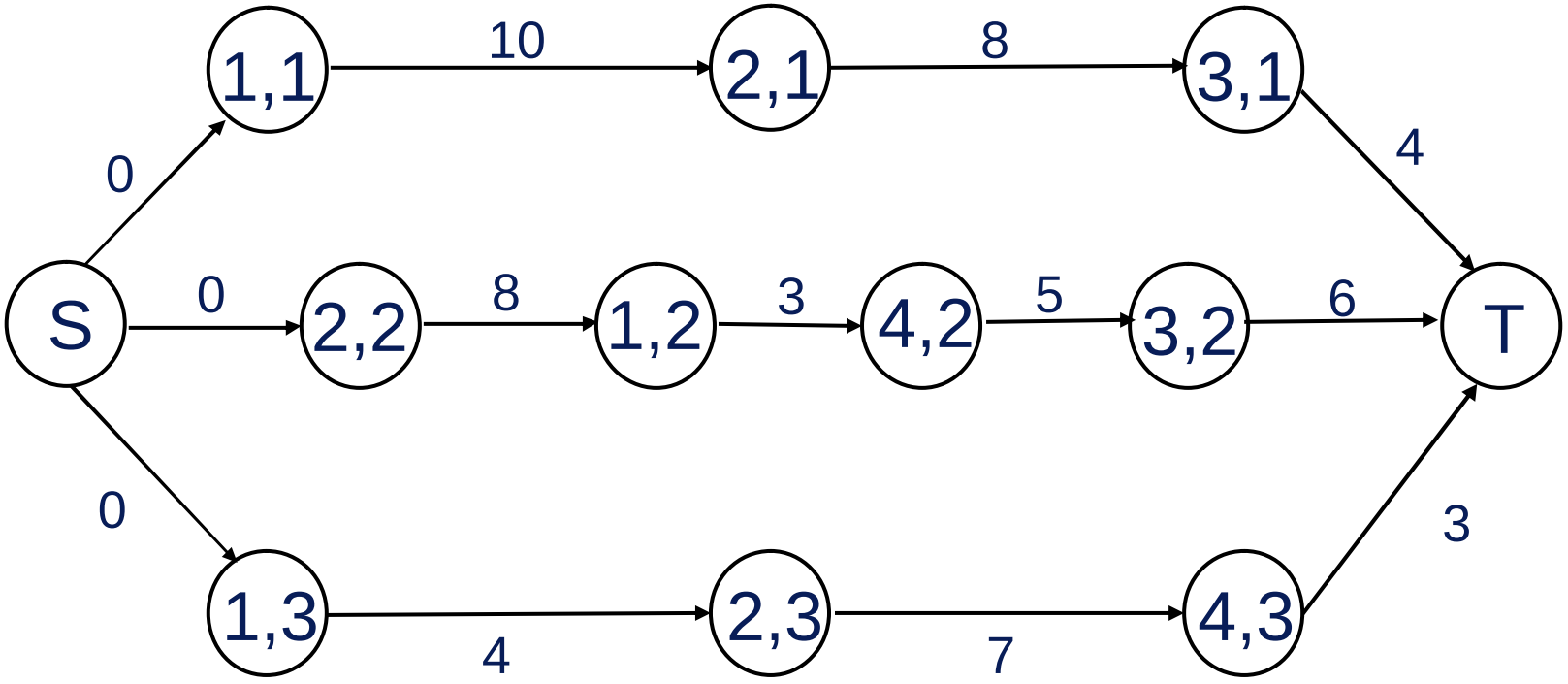
| jobs | machine sequence | processing times |
|---|---|---|
| 1 | 1, 2, 3 | $p_{11} = 10, p_{21} = 8, p_{31} = 4$ |
| 2 | 2, 1, 4, 3 | $p_{22} = 8, p_{12} = 3, p_{42} = 5, p_{32} = 6$ |
| 3 | 1, 2, 4 | $p_{13} = 4, p_{23} = 7, p_{43} = 3$ |

- Iteration 1 : $M_0 = \varnothing$ G' contains only conjunctive edges
  - ➔ Makespan (total processing time for any job ) : 22
- $1 \mid r_j \mid L_{max}$ problem for machine 1:
  optimal sequence 1, 2, 3 $\rightarrow L_{max}(1)=5$
- $1 \mid r_j \mid L_{max}$ problem for machine 2:
  optimal sequence 2, 3, 1 $\rightarrow L_{max}(2)=5$
- Similar $L_{max}(3)=4$, $L_{max}(4)=0$
  - ➔ Machine 1 or machine 2 are the bottleneck

# Example: Shifting Bottleneck Heuristic (2)

➔ Machine 1 is selected → disjunctive edges are added : graph G''

$C_{max}(\{1\}) = C_{max}(\emptyset) + L_{max}(1) = 22 + 5 = 27$



# Example: Shifting Bottleneck Heuristic (3)

- Iteration 2
- $1 \mid r_j \mid L_{max}$ problem for machine 2

  optimal sequence 2, 1, 3 $\rightarrow L_{max}(2)=1$
- $1 \mid r_j \mid L_{max}$ problem for machine 3

  optimal sequences 1, 2 and 2, 1 $\rightarrow L_{max}(3) =1$

  Similar $L_{max}(4) = 0$
- Machine 2 is selected : $M_0 = \{1, 2\}$

  $C_{max}(\{1,2\}) = C_{max}(\{1\}) + L_{max}(2) = 27 + 1 = 28$

  Disjunctive edges are added to include machine 2

  Resequencing for machine 1 does not yield any improvement
- Iteration 3

  No further bottleneck is encountered

  $L_{max}(3)=0, \quad L_{max}(4)=0$

  ➔ Overall makespan 28

| machines | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| sequences | 1, 2, 3 | 2, 1, 3 | 2, 1 | 2, 3 |

# Open Shops

- O2 || $C_{max}$

$$C_{max} \geq \max\left(\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\right)$$

- In which cases is $C_{max}$ strictly greater than the right hand side of the inequality?
- Non delay schedules
  - ➔ Idle period only iff one job remains to be processed and this job is executed on the other machine: at most on one of the two machines
- Longest Alternate Processing Time first (LAPT)

  Whenever a machine is free start processing the job that has the longest processing time on the other machine
- The LAPT rule yields an optimal schedule for O2 || $C_{max}$ with makespan

$$C_{max} = \max\left(\max_{j \in \{1,\ldots,n\}}(p_{1j} + p_{2j}), \sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\right)$$

# Open Shops (2)

- Assumption

  $p_{1j} \leq p_{1k}$ ; $p_{2j} \leq p_{1k}$

  ➔ longest processing time belongs to operation (1, k)

  LAPT: Job k is started on machine 2 at time 0

  ➔ Job k has lowest priority on machine 1

- It is only executed on machine 1 if no other job is available for processing on machine 1

  a) k is the last job to be processed on machine 1

  b) k is the second to last job to be processed in machine 1 and the last job is not available due to processing on machine 2

- Generalization: The 2(n-1) remaining operations can be processed in any order without unforced idleness.
- No idle period in any machine $\rightarrow$ optimal schedule

# Open Shops (3)

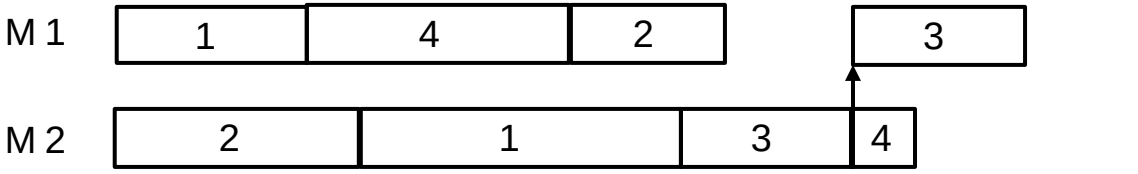
- Case 1: Idle period on machine 2
  - ➔ job2 needs processing on machine 2 (last job on machine 2) and job l is still processed on machine 1
  - ➔ job l starts on machine 2 at the same time when job k starts on machine 1 p1k = p2l → machine 1 determines makespan and there is no idle time on machine 1 → optimal schedule
- Case 2: Idle period on machine 1
  - ➔ all operations are executed on machine 1 except (1, k) and job k is still processed on machine 2
  - ➔ makespan is determined by machine 2 → optimale schedule without idle periods
  - ➔ makespan is determined by machine 1 → makespan $p_{2k} + p_{1k}$, optimal schedule

# General Heuristic Rule

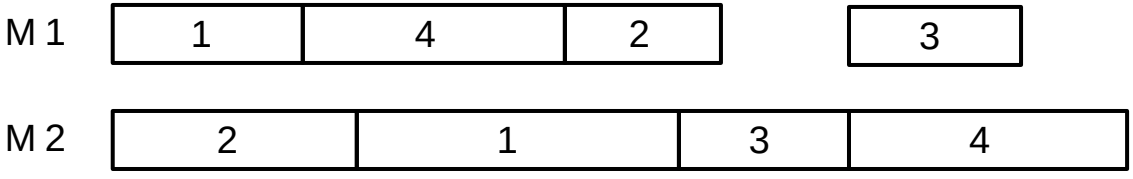
- Longest Total Remaining Processing on Other Machines first rule
  but Om || $C_{max}$ is NP hard for m = 3
  (LAPT is also optimal for O2 | prmp | $C_{max}$
- Lower bound

$$C_{max} \geq \max\left( \max_{j \in \{1,\ldots,n\}} \sum_{i=1}^{m} p_{ij}, \max_{j \in \{1,\ldots,m\}} \sum_{j=1}^{n} p_{ij} \right)$$

  ➔ The optimal schedule matches the lower bound
  The problem O2 || $L_{max}$ is strongly NP hard (Reduction of 3 Partitions)




unnecessary increase in makespan




no unnecessary increase in makespan