# Chapter 6

# Flow Shops
# &
# Flexible Flow Shops

# Presentation Approach

- First Steps

- Flow Shops

- infinite buffer space

- Permutation Flow shops
- Two flow Shops
- $F_2 \mid \mid C_{max}$
- $Fm \mid prmu \mid C_{max}$
- $F_3 \mid \mid C_{max}$
- Prop. Prmu FS

- zero/finite buffer space

- $F_2 \mid block \mid C_{max}$
- Permutation Flow shops
- 2 m-machine Flow Shops
- $F_2 \mid block, p_{ij} = p_j \mid C_{max}$
- $F_m \mid block \mid C_{max}$
- No Wait Flow Shops

- Flexible Flow Shops

- infinite buffer space

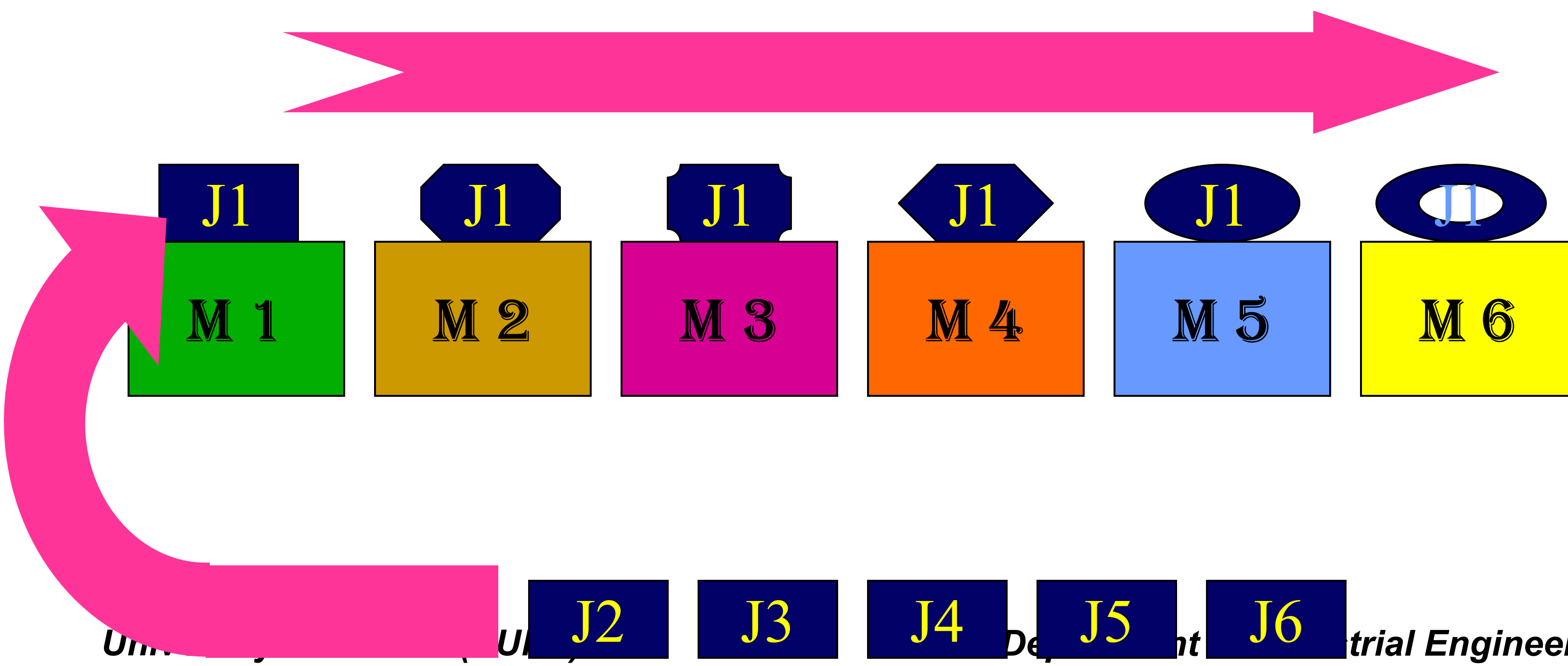
- $FFC \mid p_{ij} = p_j \mid XXX$
- Divergent $FFC \mid p_{ij} = p_j \mid \Sigma C_j$

$C_{max}$ paramount

# First Steps….

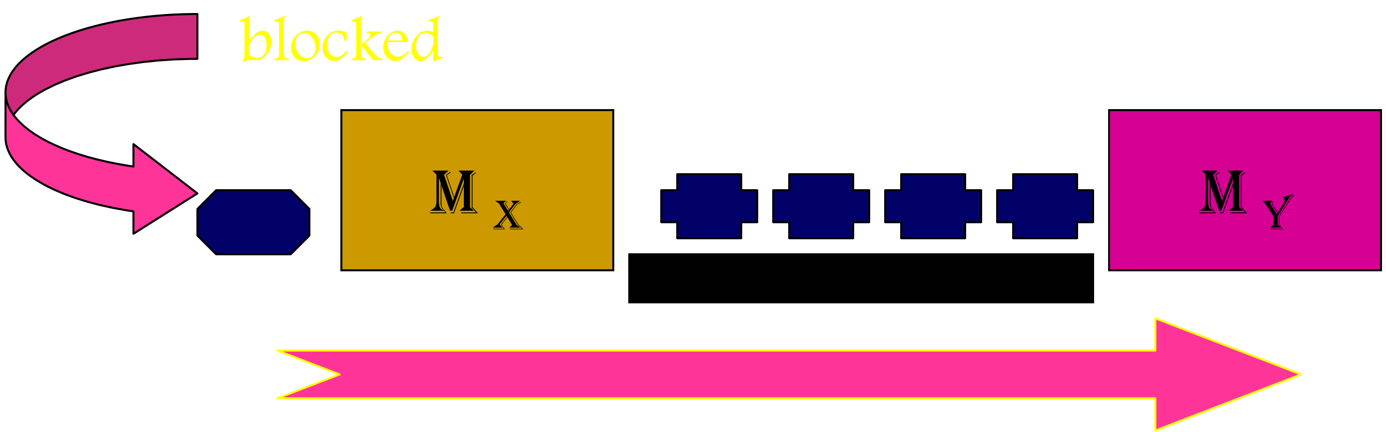
# First Steps…

- All operations on every job (every machine)
- All jobs on the same route (same order)
- Machines in series



# First Steps….

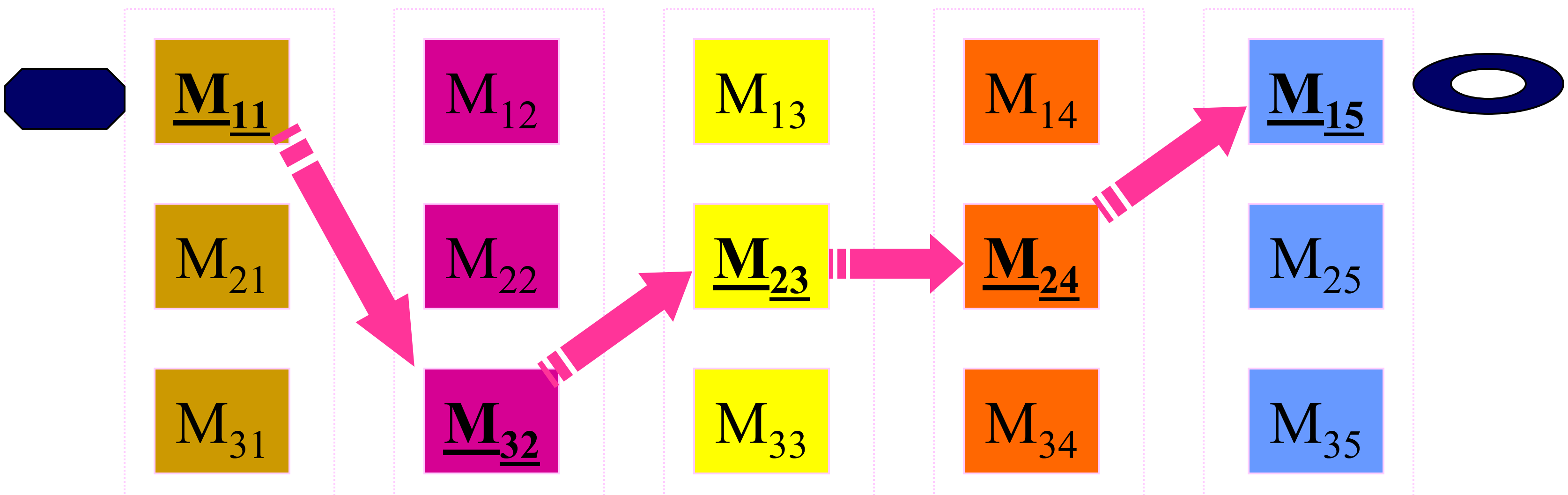
- Machines in series → issue of buffer space in between
  - Small items – no problem; space unlimited
  - Large items – capacity (space) constraints
- Blocking
  - When buffer space is full



# Flexible Flow Shop

**Also called Compound, Hybrid or Multiprocessor flow shop**

- More generic environment
- Number of stages in series
- Machines in parallel at each stage
- A given job processed on one machine at each stage



# Makespan Objective ~ $C_{max}$

- Paramount focus of research
- Practical Interest
  - Utilization = Processing time/Makespan
  - Hence minimize $C_{max}$ → maximize Util.
- $C_{max}$ already hard to optimize
- Other objectives ($\Sigma$ $C_j$, $D_j$ related, etc.) offer harder challenges

# Flow shops ~ differentiation

According to intermediate buffer space capacity (between machines)

| Flow shops | Flexible FFs |
| --- | --- |
| Unlimited capacity<br>Limited capacity | Unlimited capacity<br>Limited capacity |

# Flow Shops with…..

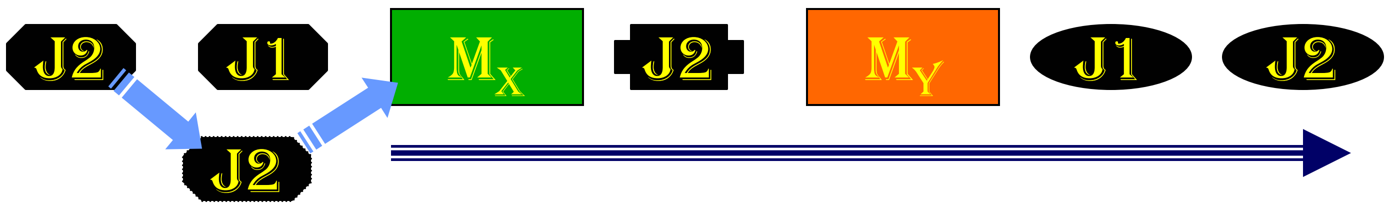
# ….unlimited buffer space

# Flow shops – unlimited buffer space

$Fm \mid \mid C_{max}$ – no constraints and unlimited buffer space

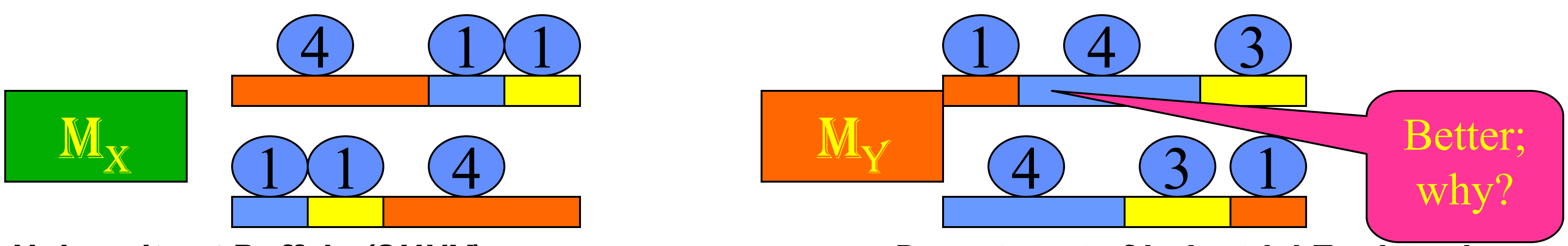
Is one permutation of jobs traversing sufficient ?

Jobs can pass one another while waiting in queues

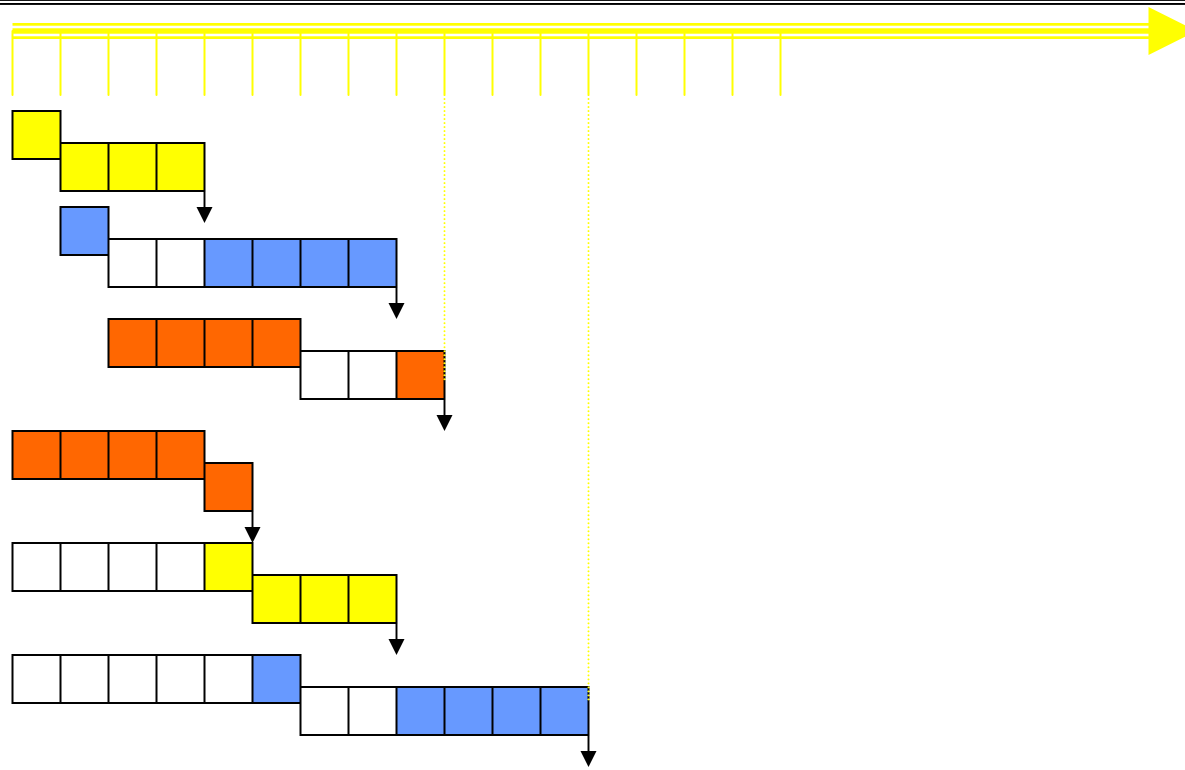



Sequence of jobs will change from machine to machine

Changing sequences of jobs between machines may result in lower $C_{max}$

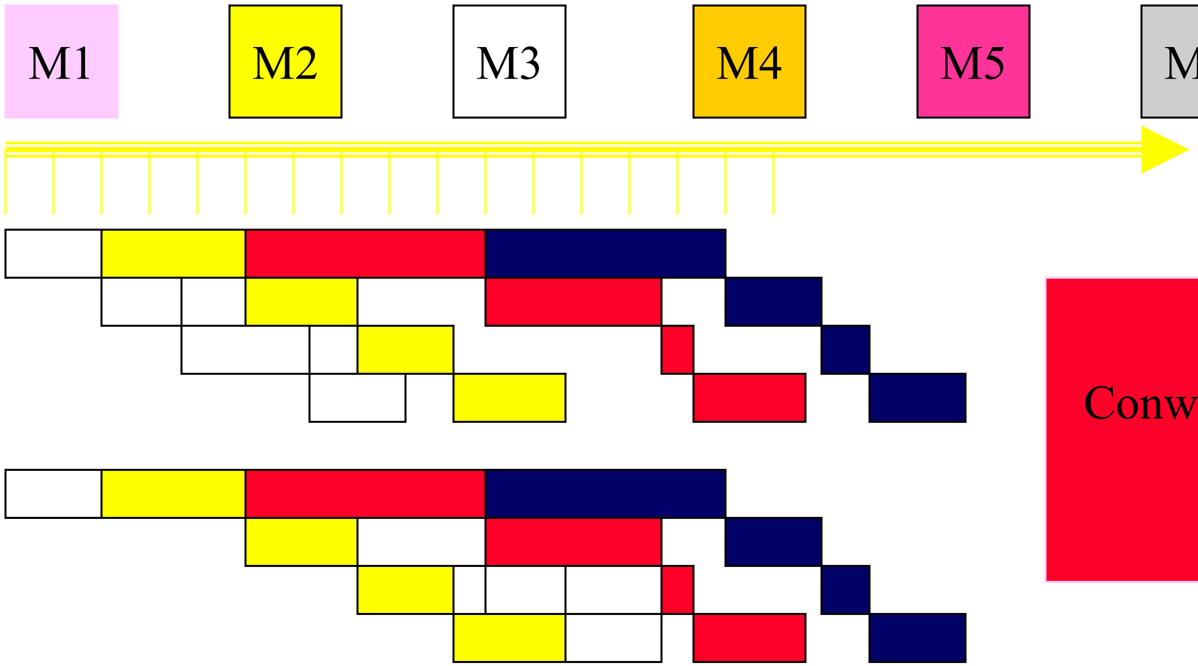


| | $M_x$ | $M_y$ |
|---|---|---|
| Y | 1 | 3 |
| B | 1 | 4 |
| R | 4 | 1 |



For an m-machine Flowshop, there exists an optimal schedule that does not need jobs to be re-sequenced between the **first 2** and **last 2** machines

# Re-Sequencing



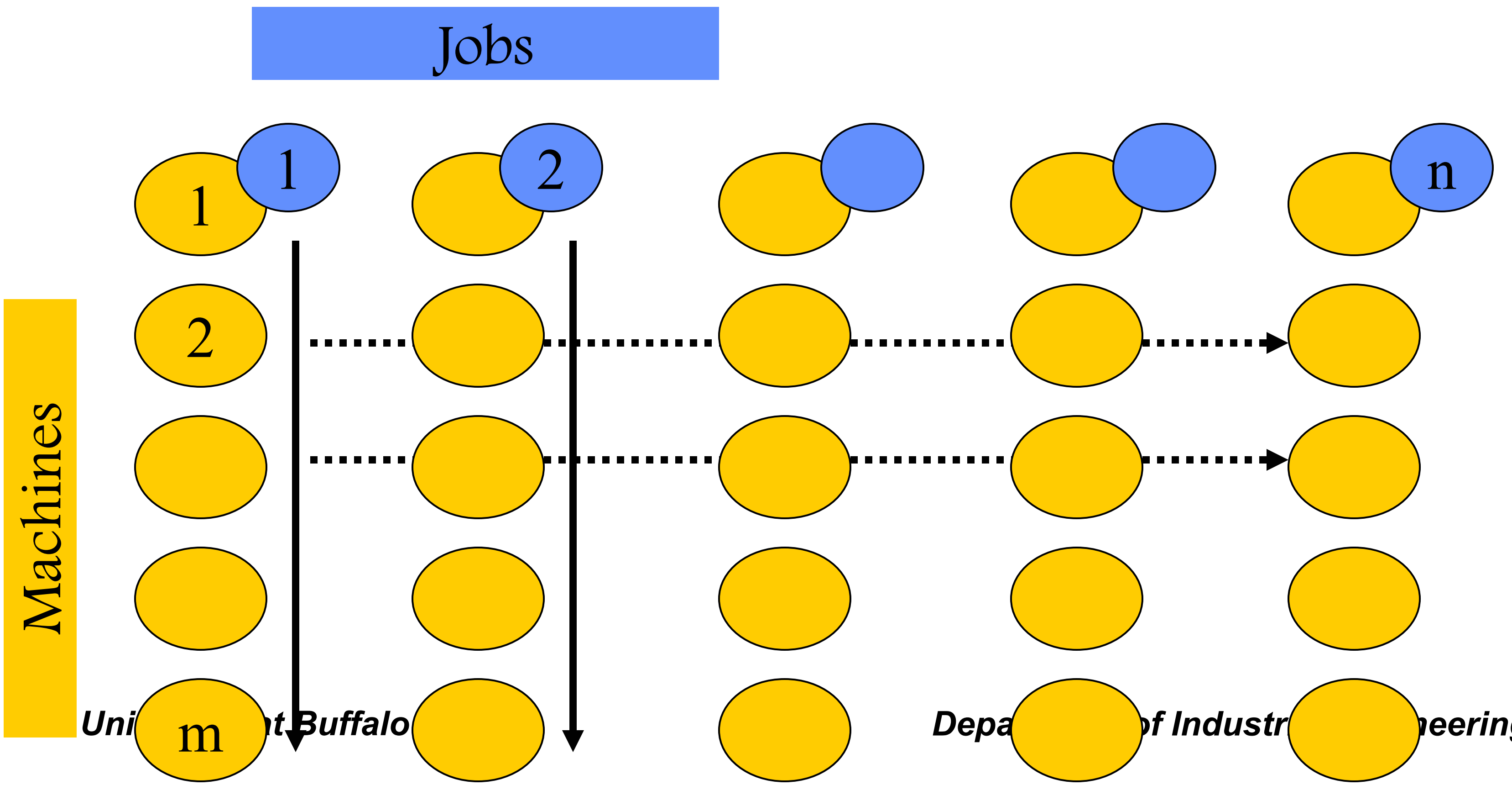

Conway, Maxwell and Miller

For 2 machines in series, there will always be an optimal schedule without job sequence changes

$F_2 \mid \mid C_{max}$ ?  $F_3 \mid \mid C_{max}$ ?  $F_{>3} \mid \mid C_{max}$ ?

# Permutation Flow Shops

- Sequencing of jobs creates scheduling problems
- If sequencing NOT allowed – permutation flow shops – easier to model



# Permutation Flow Shops

- Completion time of job $j_1$ (given) at machine i will depend on earlier processing times of the said job $j_1$
- ➔ $C_{i,j_1}$ = Processing time (of $j_1$) on machine 1 + processing time on machine 2 + ……… + processing time on machine i
    - $C_{i,j_1} = \Sigma P_{s,j_1}$ (summation of s from 1 to i)
    - m equations for the said job at every machine

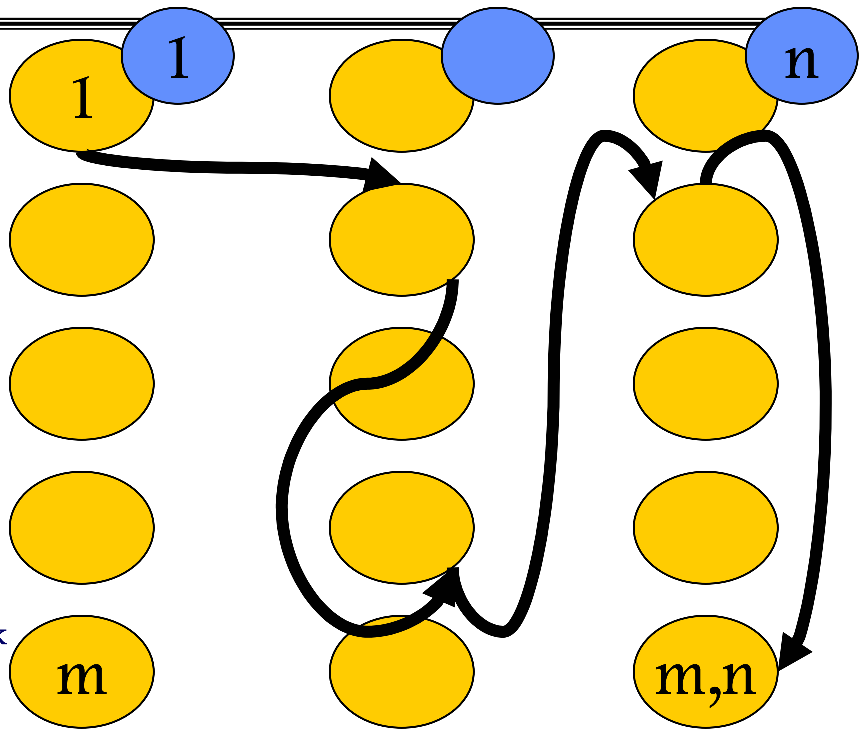
- Completion time of job $j_k$ at machine 1 (given) will depend on the processing times of earlier jobs on the said machine 1
- ➔ $C_{1,j_k}$ = Processing time of $j_1$ on machine 1 + processing time of $j_2$ on machine 1 + ……… + processing time of $j_k$ on machine 1
    - $C_{1,jk} = \Sigma P_{1,j_s}$ (summation of s from 1 to k)
    - n equations for each job at the given machine

# Iterative solution

- The previous two (m+n) equations and
- Completion time of any job $j_k$ at any machine i will depend on
  - Completion time of job $j_{k-1}$ at machine i (earlier job over)
  - Completion time of job $j_k$ at machine i-1 (present job can start)
  - Whichever is later &
  - Processing time of job $j_k$ on machine i
- $C_{i,j_k} = \text{Max}\,(C_{i-1,j_k}, C_{i,j_{k-1}}) + P_{i,j_k}$
  - for m-1 machines from 2 to m and n-1 jobs from 2 to n
- We have
  - initializing equations for machine 1 (for each job)
  - Initializing equations for job $j_1$ (for every machine)
  - SOLVE ITERATIVELY for completion times and makespan

# Alternative solution ~ makespan

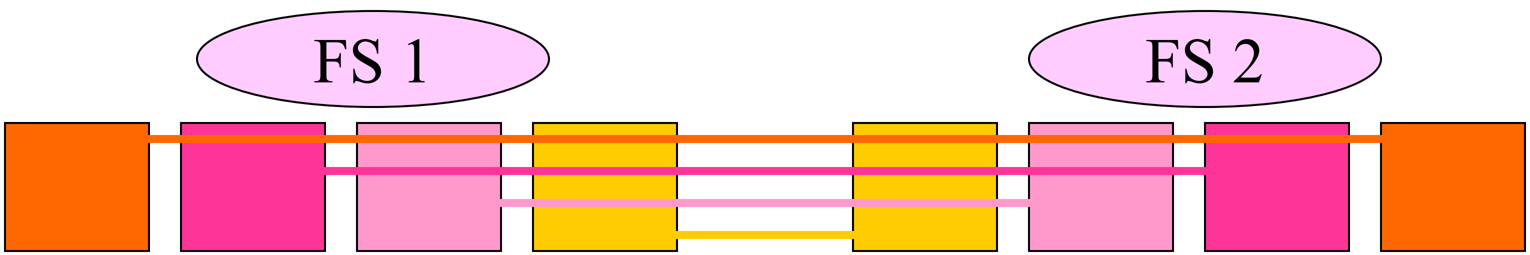
- Using critical path algorithm on a directed graph
  - Each job is processed on each machine i, which means there exists a node $(i, j_k)$ for each operation
  - The weight of each node is the processing time $P_{i,j_k}$
  - Find maximum weighted path $\Sigma P_{i,j_k}$ from node $(1,j_1)$ to node $(m, J_n)$




Both methods for no-changes in sequence situation
Permutation flow shop

# Two Flow Shops

- Both permutation FS with m machines
- Number of jobs n
- Processing time of job j on machine i in first FS = $p_{ij}^1$
- Processing time of job j on machine i in 2$^{nd}$ FS = $p_{ij}^2$
- Assume $p_{ij}^1 = p_{m+1-i,j}^2$




**lemma**

$j_1$ to $j_n$ sequencing in FS 1 =
$j_n$ to $j_1$ sequencing in FS 2

# Reversibility Result

For a Permutation job shop

- Job order reversed
- Jobs traverse machines in reverse order

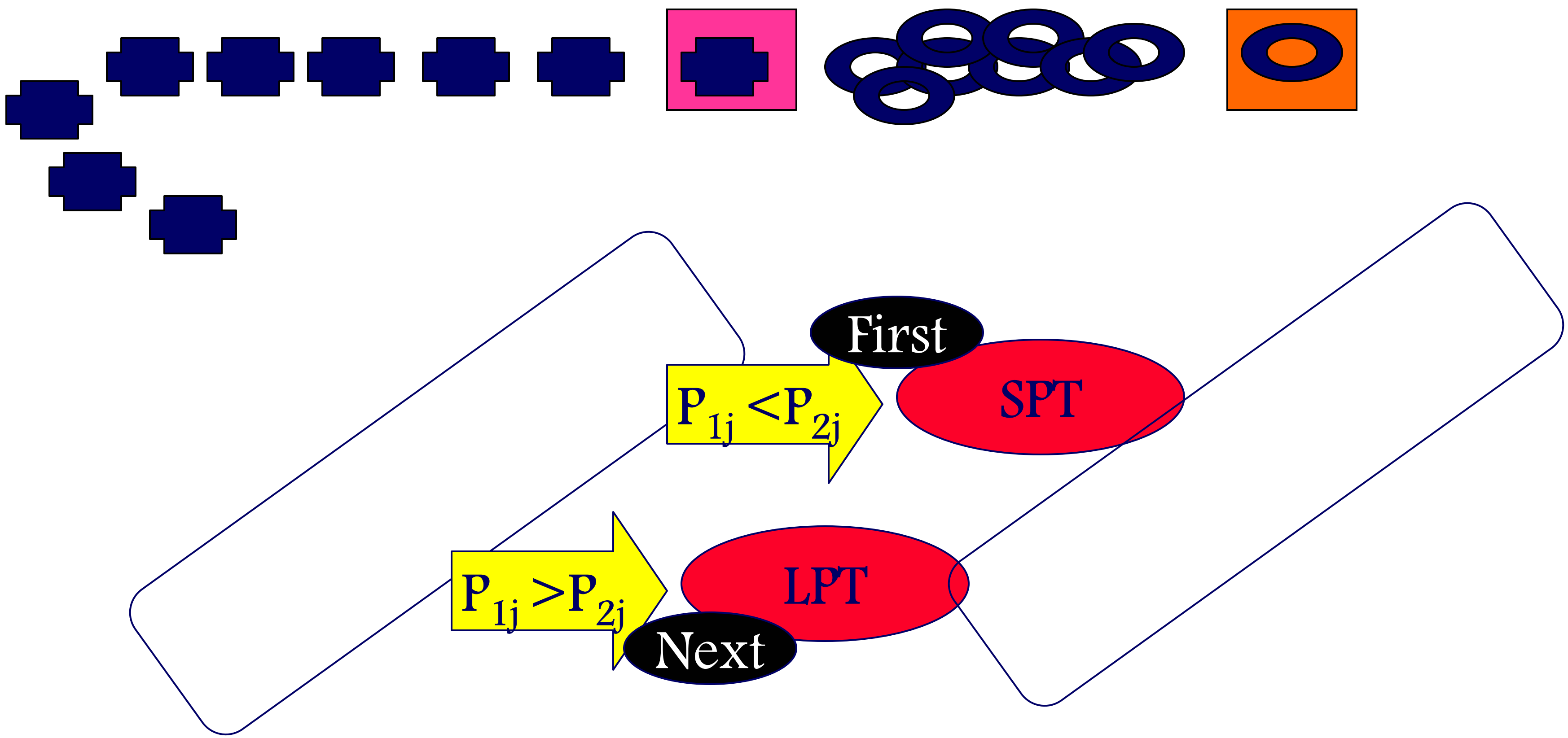
will mean no change in Makespan

Other results with multiple machines are extremely complex

Backtrack to 2 machine problems!!

# $F_2 \mid \mid C_{max}$

- n jobs 2 machines unlimited storage




More than one schedule can be constructed this way

# Johnson's algorithm

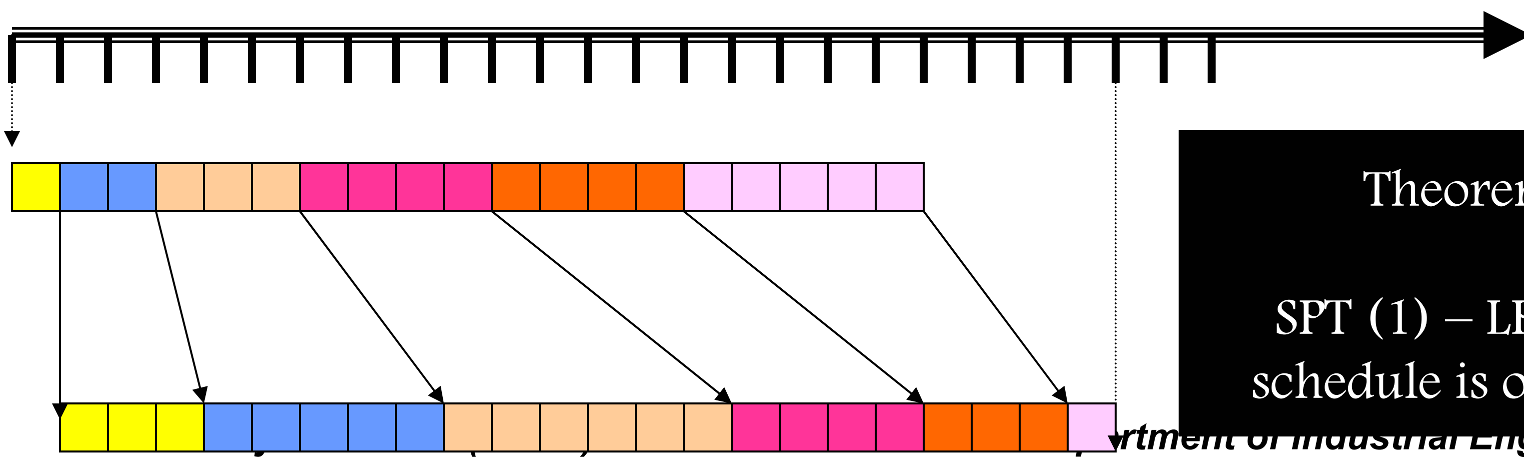
| ~~Jobs~~ | ~~Machine 1~~ | ~~Machine 2~~ |
|---|---|---|
| 1 | 3 | 6 |
| 2 | 5 | 1 |
| 3 | 4 | 3 |
| 4 | 2 | 5 |
| 5 | 1 | 3 |
| 6 | 4 | 4 |

CATEGORY 1

CATEGORY 2

**SPT**
**➔ 5, 4, 1**
**Inc. $P_{1j}$**

**LPT**
**➔ 6, 3, 2**
**Dec. $P_{2j}$**



Theorem

SPT (1) – LPT (2) schedule is optimal

# SPT (1) – LPT (2) Optimality

Proof by contradiction

| | | |
|---|---|---|
| j ε Set 2<br>k ε Set 1 | j and k<br>ε Set 1;<br>$P_{1j} > P_{1k}$ | j and k<br>ε Set 2;<br>$P_{2j} < P_{2k}$ |

To prove: Under any of these conditions, pairwise interchange (j and k) will reduce makespan

Original schedule: let job l < job j < job k < job m

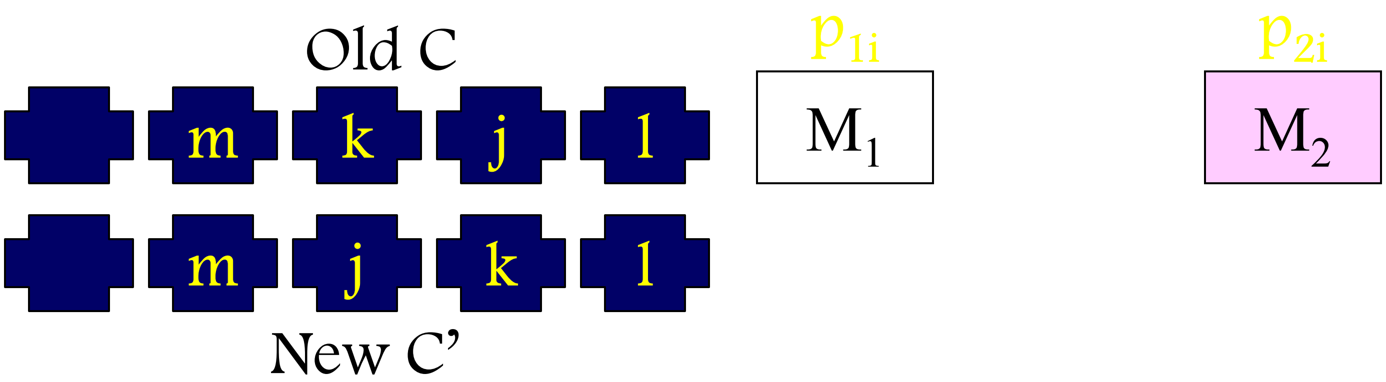
$C_{ij}$

New schedule: let job l < job k < job j < job m

$C_{ij}'$

Look at $C_{ij}$ for Job m

# SPT (1) – LPT (2) Optimality



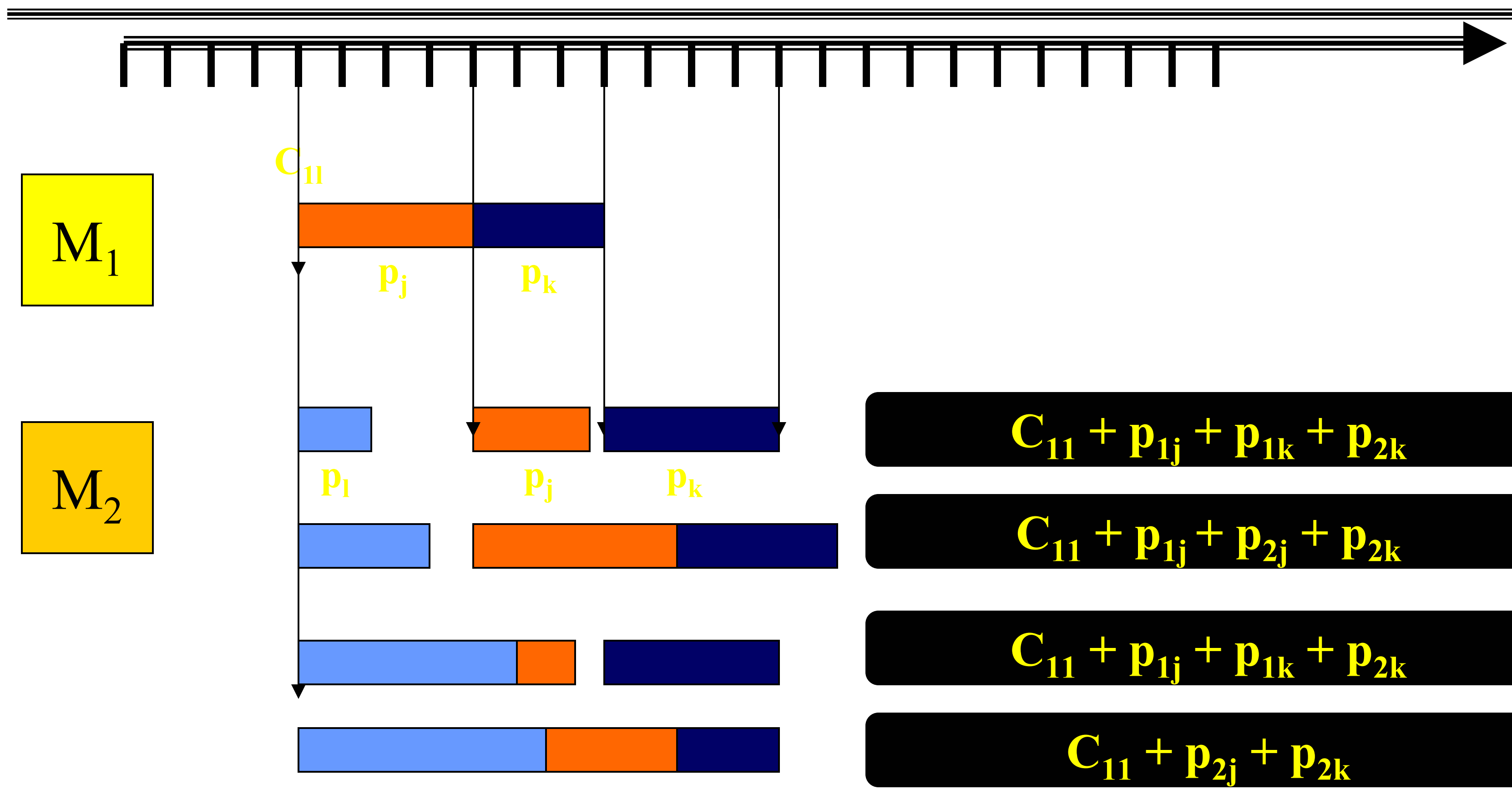

- For job m, $C_{1j}$ (machine 1) will not be different since
  - $C_{1m} = C_{11} + p_{1j} + p_{1k}$
- When does job m reach machine 2?

Old $= C_{2k}$

New $= C_{2j}'$

- Hence, simply show that $C_{2k} > C_{2j}'$

# SPT (1) – LPT (2) Optimality




$C_{11} + p_{1j} + p_{1k} + p_{2k}$

$C_{11} + p_{1j} + p_{2j} + p_{2k}$

$C_{11} + p_{1j} + p_{1k} + p_{2k}$

$C_{11} + p_{2j} + p_{2k}$

**$C_{2k}$ = max of the above**

**Similarly, compute $C'_{2j}$**

# SPT (1) – LPT (2) Optimality

~~$C_{2k} = \text{Max}\,(C_{21} + p_{2j} + p_{2k}, C_{11} + p_{1j} + p_{2j} + p_{2k}, C_{11} + p_{1j} + p_{1k} + p_{2k})$~~

$C'_{2j} = \text{Max}\,(C_{21} + p_{2j} + p_{2k}, C_{11} + p_{1k} + p_{2k} + p_{2j}, C_{11} + p_{1k} + p_{1j} + p_{2j})$

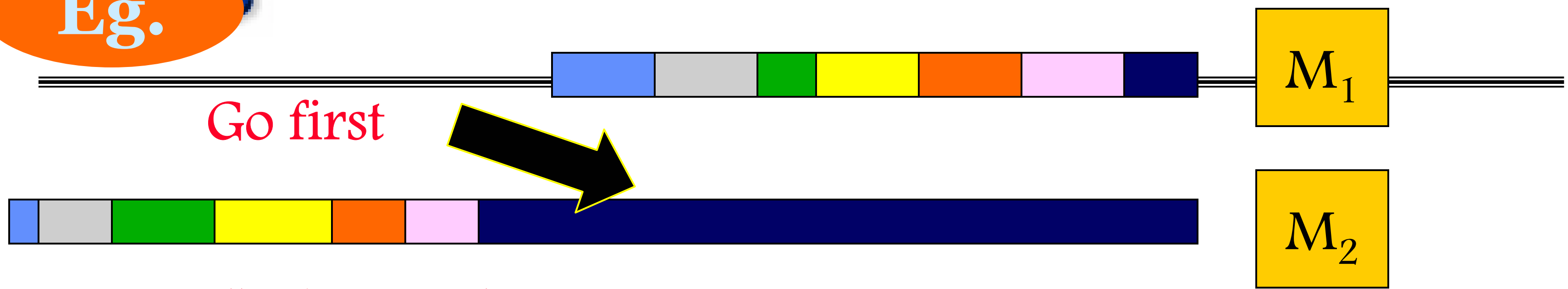
| | |
|---|---|
| Condition 1: j ε Set 2 & k ε Set 1 | $P_{1j} < P_{2j}$ |
| | $P_{1k} > P_{2k}$ |
| j and k ε Set 1; $P_{1j} > P_{1k}$ | $P_{1j} < P_{2j}$ |
| | $P_{1k} < P_{2k}$ |
| j and k ε Set 2 ; $P_{2j} > P_{2k}$ | $P_{1j} > P_{2j}$ |
| | $P_{1k} > P_{2k}$ |

$C_{2j}' < C_{2k}$

These are not the only optimal schedules
Others hard to characterize, data dependent

# Other results

Eg.




Remaining: all orders optimal

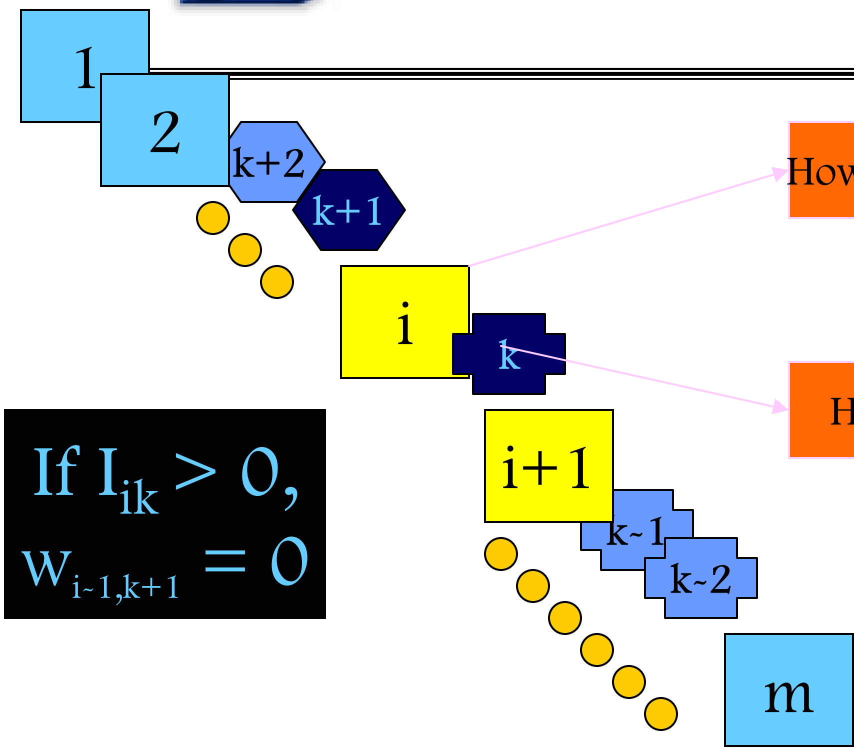
> 2 machines: SPT(1) – LPT(2) schedule not applicable

## Minimizing makespan in $Fm \mid prmu \mid C_{max}$ as an MIP

Define variables

$x_{jk}$ = 1 if j is $k^{th}$ job in sequence, 0 otherwise
$I_{ik}$ = idle time on machine i between processing jobs in $k^{th}$ and $(k+1)^{th}$ position
$W_{ik}$ = waiting time of $k^{th}$ job between machines i and i + 1

# $Fm \mid prmu \mid C_{max}$




How long does the machine i wait? $I_{ik}$

How long does the job k wait? $W_{ik}$

If $I_{ik} > 0$,
$W_{i-1,k+1} = 0$

Minimizing makespan =
Minimizing total idle time on last machine

Total idle time at machine m =

Idle time before (1st) job reaches machine m + Sum of "waits" of all jobs (n – 1) from then on machine m

# $Fm \mid prmu \mid C_{max}$

## MIP Formulation

How long m waits for each job

$$\text{Min } (\Sigma p_{i(1)} + \Sigma I_{mj}) = \text{Min } (\Sigma \Sigma x_{j1} p_{ij} + \Sigma I_{mj})$$

Processing time for all jobs till m

Subject to:

$$\sum_{j} xjk = 1,\ k = 1,\ldots\ldots,n$$

$$\sum_{k} xjk = 1,\ j = 1,\ldots\ldots,n$$

Exactly one job to a given position
Exactly one position for a given job

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

Idle time on machine i after job k over + processing time of (k+1th) job on machine i + Idle time for job k+1 before i+1th machine

Idle time on machine i+1 after job k over + processing time of (kth) job on machine i +1 + Idle time for job k before i+1th machine

In short, (k+1)th job completes on machine i+1LESS kth job completes on machine i must NOT overlap

$$W_{i1} = 0,\ i = 1,\ldots,m-1;\ I_{1k} = 0,\ k = 1,\ldots,n-1$$

NP Hard

# $F_3 \mid\mid C_{max}$

Is strongly NP – hard

Cannot use SPT-NPT algorithms

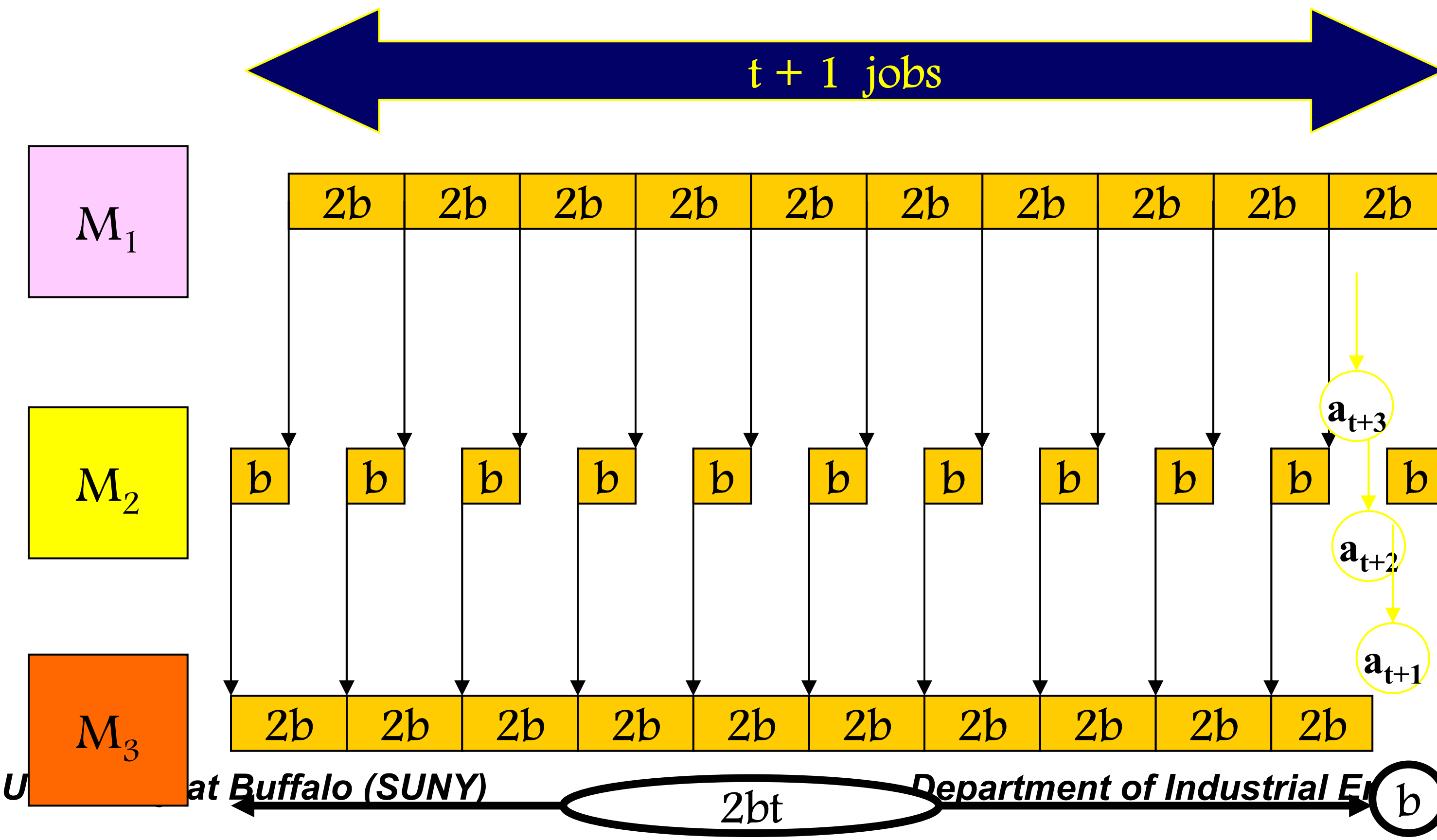
Proof: by reduction from 3-partition (using one unsolvable simple case)

- Consider n = 4t + 1 jobs in all
- Select easy to manipulate processing times (0,b,2b,$a_j$)
- Makespan for t+ 1 jobs= (2t+1)b
- Take first t+1 jobs and schedule them on 3 machines
- You have t gaps in between
- You have 3t jobs left with varying processing times only on machine 2 ($a_j$s)
- Fit 3t jobs thrice over in the t gaps
- Can happen only if all of them fit in

For Permutation as well as Sequence change

# $F_3 \mid\mid C_{max}$

| Job 0 | Jobs 1,…,t | Jobs t+1,…4t |
|---|---|---|
| $M_1=0$, $M_2=b$, $M_3=2b$ | $M_1=2b$, $M_2=b$, $M_3=2b$ | $M_1=0$, $M_2=a_j$, $M_3=0$ |



# Fm | prmu | $C_{max}$ – special cases

- Generally NP hard
- Special cases can be solved
- Proportionate permutation FS

If jobs have same processing times on each of the m machines = $p_j$

… Can be solved by SPT-LPT algorithm

Any sequence $j_1, j_2, \ldots\ldots, j_n$ is SPT-LPT solvable only if

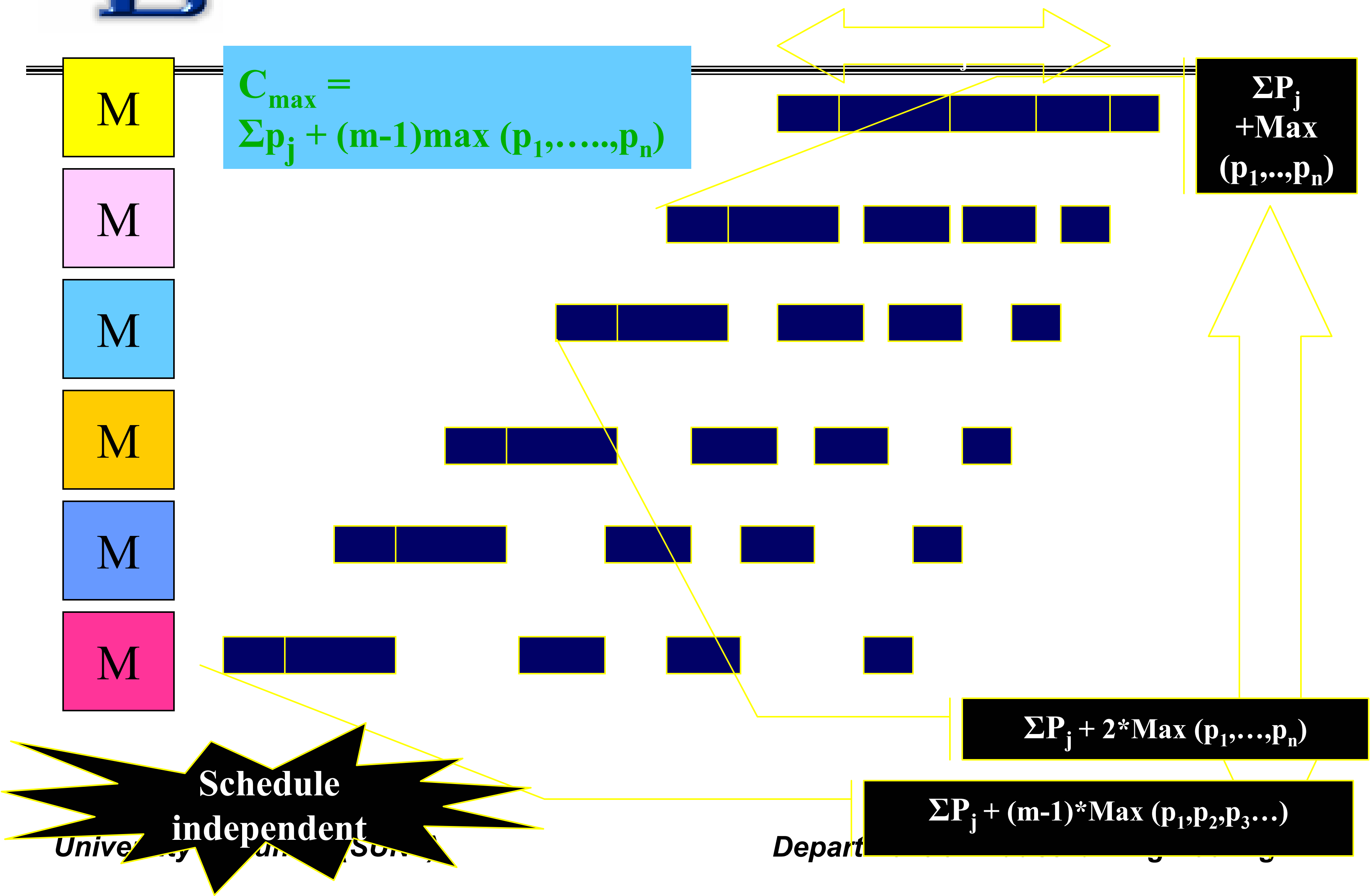
$j_k$ exists such that $p_{j1} \leq p_{j2} \leq \ldots\ldots \leq p_{jk}$ and $p_{jk} \geq p_{jk+1} \geq \ldots\ldots \geq p_{jn}$

SPT-LPT solution is optimal, but so are many others!!!!!

$$C_{max} = \Sigma p_j + (m-1)\max(p_1,\ldots\ldots,p_n)$$

And is INDEPENDENT of the schedule

# Proof : Special Fm | prmu | $C_{max}$



# Independent of Schedule Results

- Take same processing time case (specific to job, not to machine)
- $Fm \mid p_{ij} = p_j \mid C_{max}$
- $Fm \mid prmu \mid C_{max}$ is optimal in above even if jobs can pass one another
- ALSO, owing to independence of schedule, makespan does not depend on sequence

| | | |
|---|---|---|
| | SPT-LPT solvable | SPT-LPT solvable |
| | Same algorithm for optimal schedule | |
| | Same algorithm for optimal schedule | |
| | Same pseudopolynomial programming algorithm | |
| | Same elimination criteria | |

**Many 1 machine algorithms can be applied directly proportionate Fm situations; But counterexamples exist (e.g. total weighted completion time)**

# Prop. Prmu FS – Diff. Speeds

- Makespan becomes schedule dependent
- ~~Speed of machine i = $v_i$ ➔ processing time = $p_j/v_i$~~
- Machine with smallest $v_i$ [i.e. Max $(p_j/v_i)$ for all j] = bottleneck

<u>Theorem</u>: Prop. Prmu FS with different speeds and with first (last) machine as bottleneck ➔ LPT (SPT) minimizes makespan

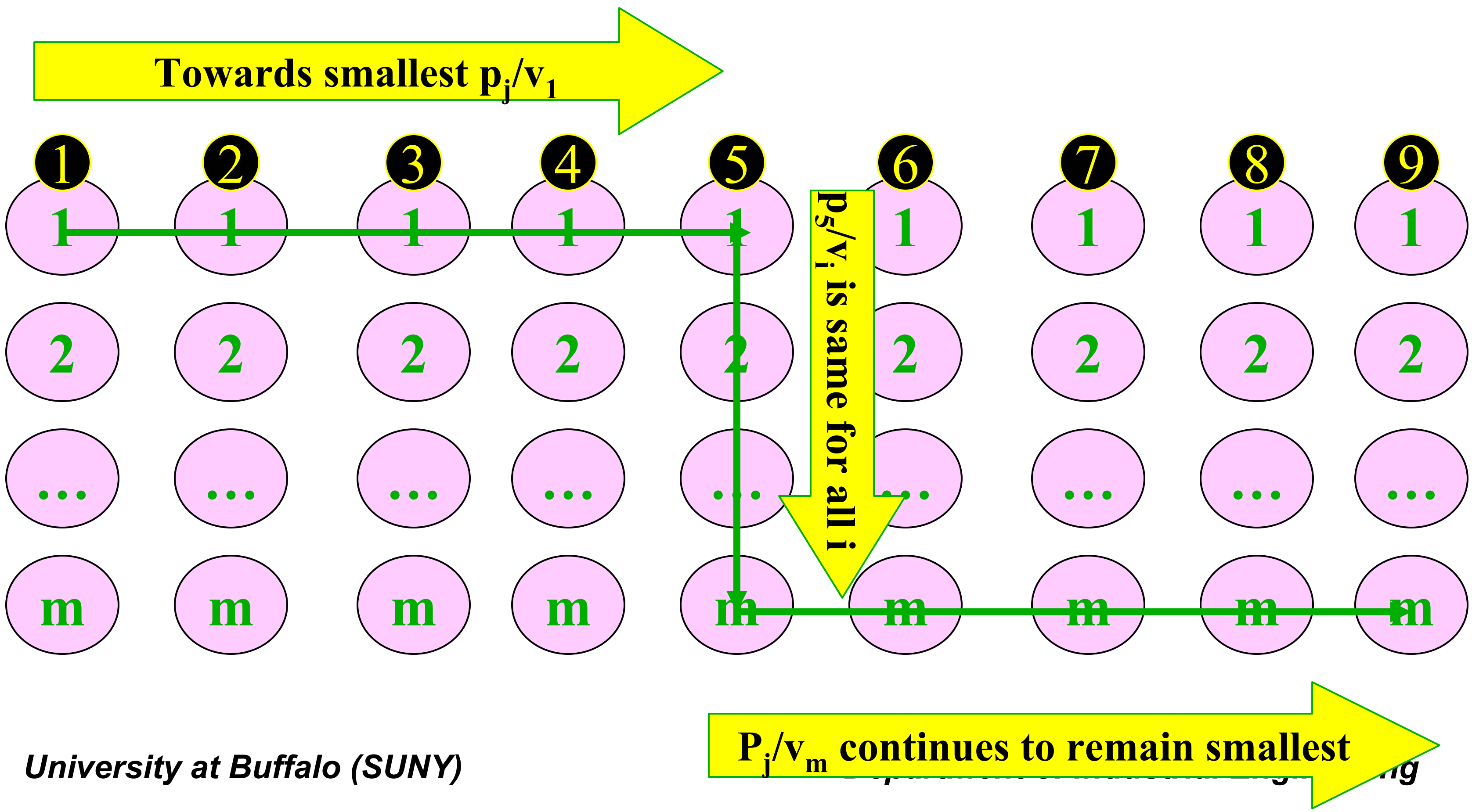
Reversibility theory implies only last machine case need be proved

**Consider special case with $v_m < v_1 < \min(v_1, v_2, \ldots, v_{m-1})$**

Proof: First onward case (for special case above)
Then converse (for special case above)
Then general case

# Prop. Prmu FS – Diff. Speeds

## Onward portion: Special case



# Prop. Prmu FS – Diff. Speeds

- Consider a schedule that is NOT SPT

$j + 1 = k$

$P_{1k} = p_{mk} = p_{m,j+1}$

$\Sigma p_{ij} > \Sigma p_{ik}$

Total = +

$p_{2j}$ +….. + $p_{mj}$ +

$p_{m,j+1}$+

Total = + $p_{ik}$ +

$p_{2k}$ +…. + $p_{mk}$ +

# General case in Fm/prmu/Cmax

Is NP hard and solved through heuristics

Several available

**Slope heuristic** is amongst the first

Reasoning:
from SPT(1)-LPT(2) algorithm theorem (2 machine case)

a) Small PT on 1st m/c & Large PT on 2nd ➔ beginning of schedule
b) Large PT on 1st m/c & Small PT on 2nd ➔ end of schedule

Define a Slope Index for each job ά to a) and 1/ ά to b)

**Large when i large**

$$\text{Slope Index } A_j = -\sum_i (m - (2i - 1))\, p_{ij}$$

# Fm | | Other objective functions

- Are much harder
- $F_2$ | | $\Sigma C_j$ is STRONGLY NP hard (difficult proof)
- Fm | pij = pj| $\Sigma C_j$ is SPT solvable in a proportionate FS

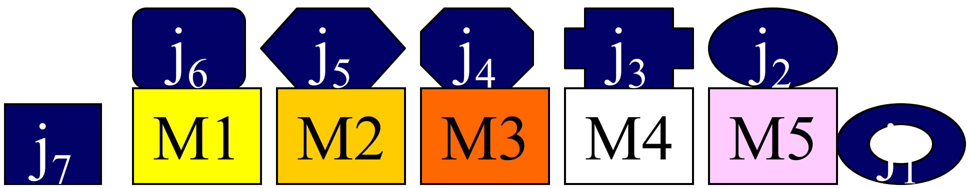
Onward to FS with limited intermediate Storage

# Flow shops with….

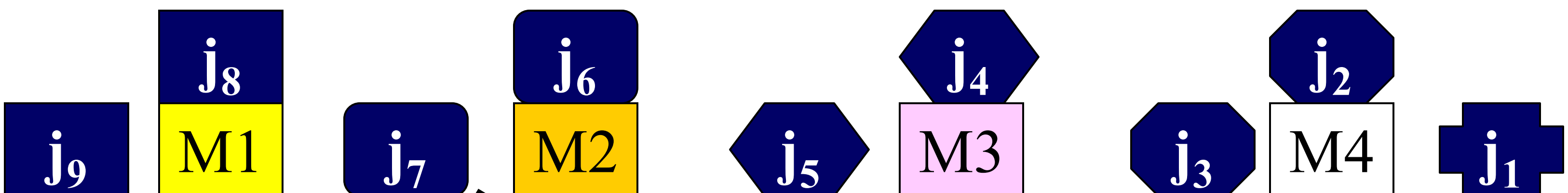
# …….Limited Buffer Space

# Blocking

- Happens when intermediate storage is zero or finite




**Zero space in between for each job**
**job cannot proceed to next machine if**
**That is ON**




**Finite (1 or more) jobs can wait in between machines; the preceding machines can be relieved of one or more jobs when completed**

**Each intermediate space with one job**
**Is similar to a machine with zero $p_{ij}$**

Zero storage case is best to consider and analyze

# $F_2 \mid block \mid C_{max}$

- Define $D_{ij}$ = actual time of departure of job j from m/c i
- $D_{ij} > C_{ij}$ which is the completion time
- $D_{0j}$ = time when job j starts on first machine

$D_{i,j1} = \Sigma\, p_{l,j1}$ summation of all processing times on machines 1 to I (for job j1)

$D_{m,jk} = D_{m-1,jk} + p_{m,jk}$ ; Last machine will have infinite space ahead

$D_{i,jk} = \text{Max}\,(D_{i-1,jk} + p_{i,jk}\,,\, D_{i+1,jk-1})$

Time when next machine is done with previous job or

Time when previous machine was done with present job

PLUS

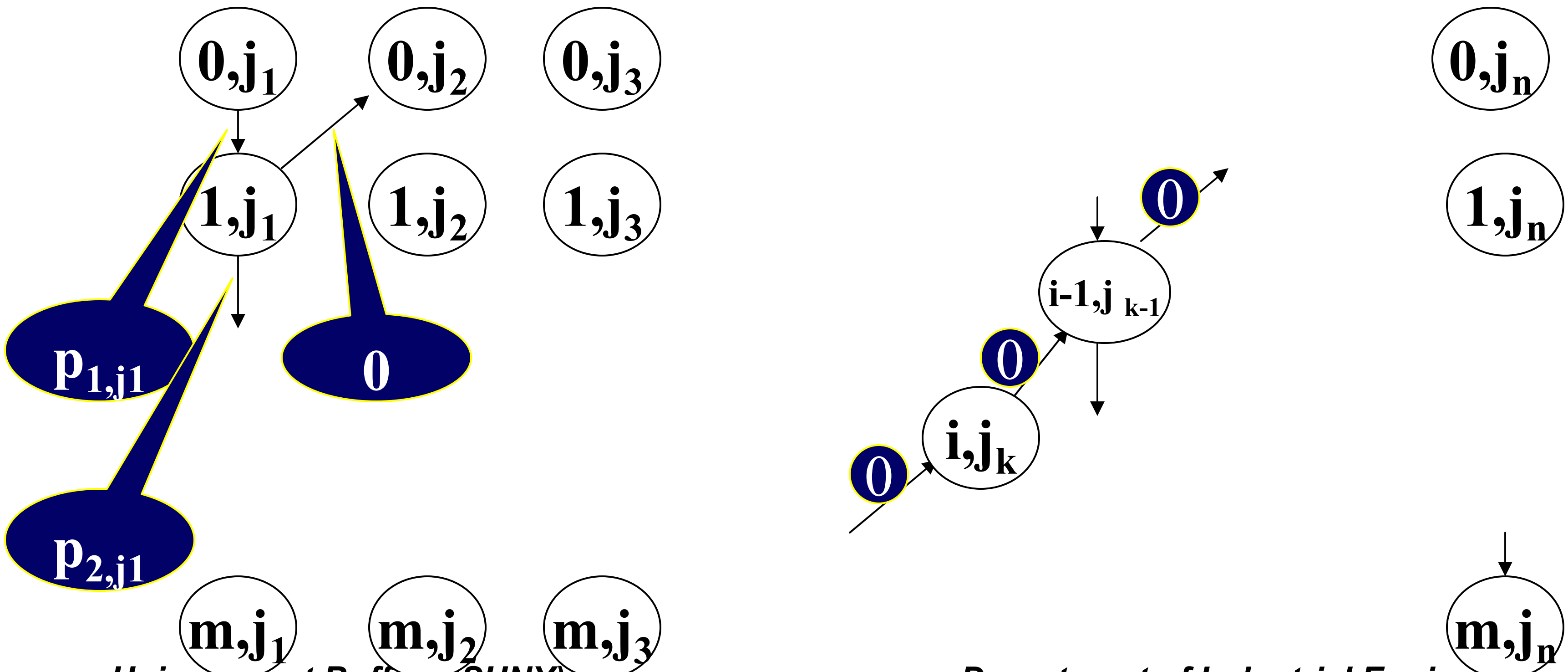
Processing time of present job

| M1 | Present<br>M2 | Previous<br>M3 |
|---|---|---|

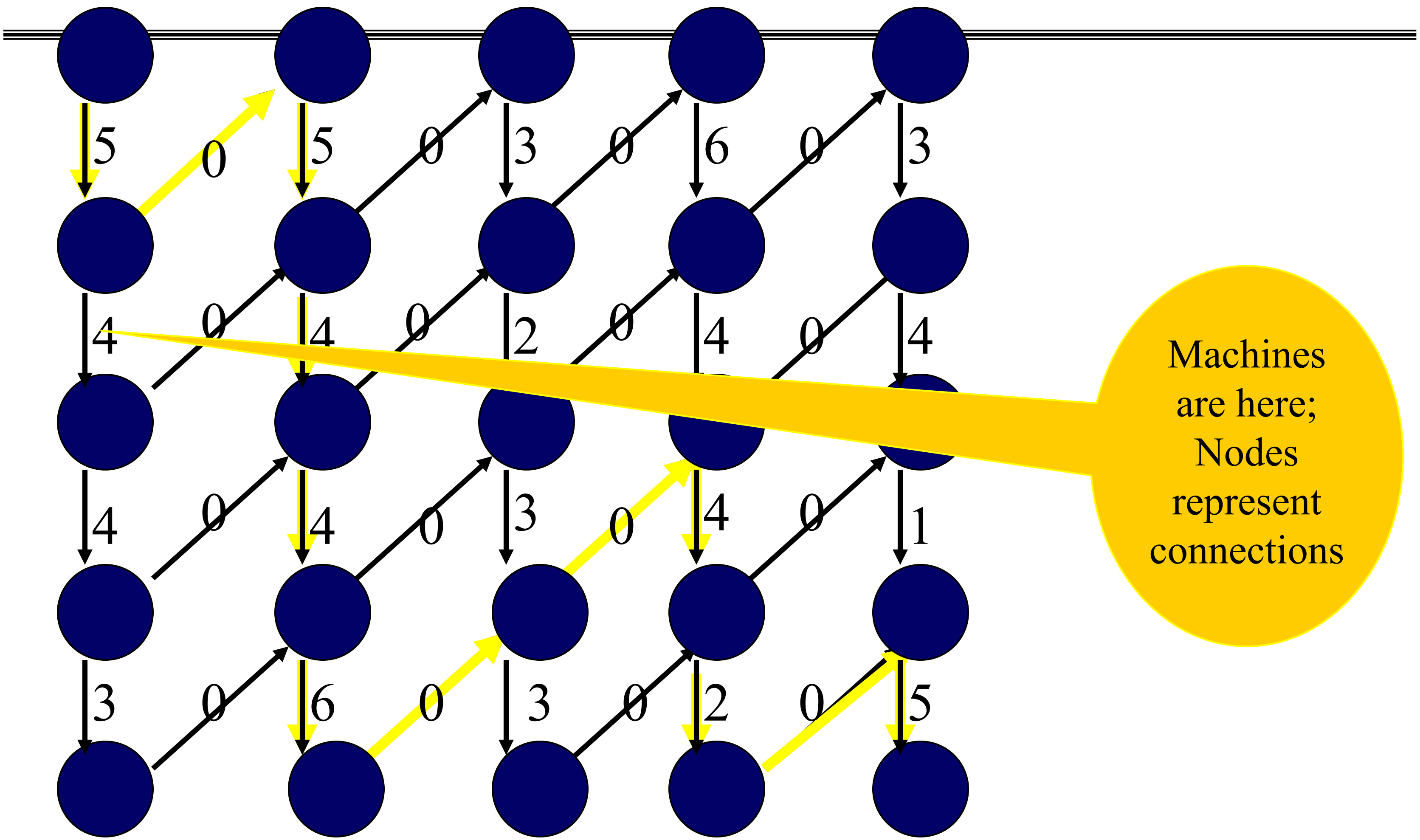
For jobs
$j_1, j_2, \ldots, j_n$

# Prmu schedule model

- Makespan = computed by critical path
- Earlier directed graph (unlimited storage) ➔ nodes had weights
- Now, arcs given weights



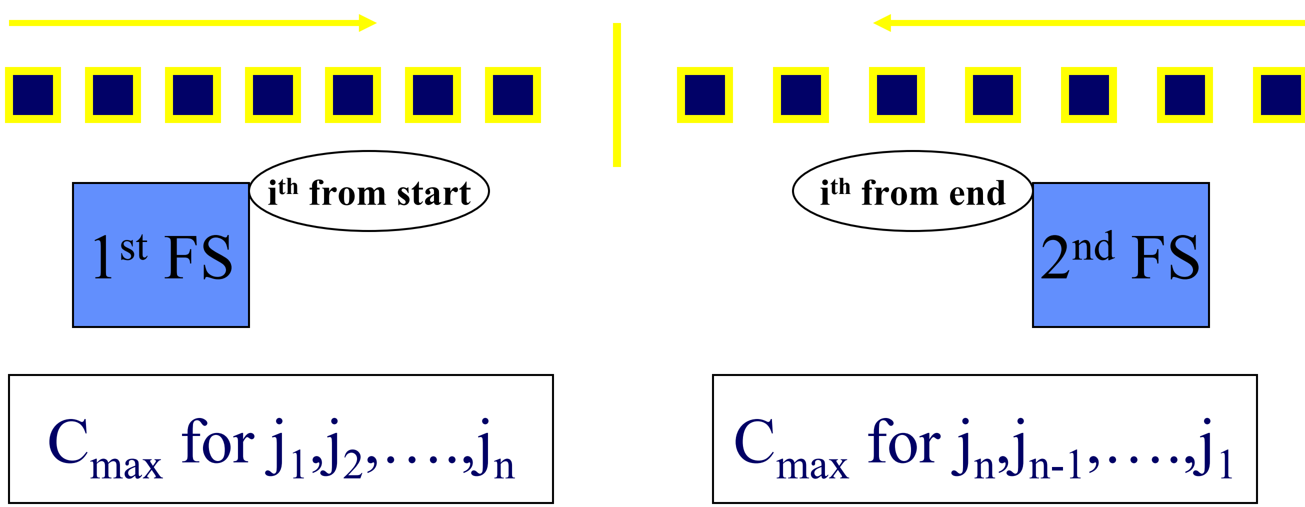
# Prmu schedule example



# 2 m-machine Flow Shops

- Reversibility property true for zero intermediate storage if
- $P_{ij}^{(1)}$ and $P_{ij}^{(2)}$ are the respective processing times and
- $P_{ij}^{(1)} = P_{m+1-i,j}^{(2)}$

**Lemma 6.2.2**




**Proof: one-to-one correspondence between paths of equal weight**

# $F_2 \mid block, p_{ij} = p_j \mid C_{max}$

- Theorem: Only an SPT-LPT schedule is optimal (*for* $F_2 \mid block, p_{ij} = p_j \mid \Sigma C_{max}$ *as well*)
- When unlimited buffer space,

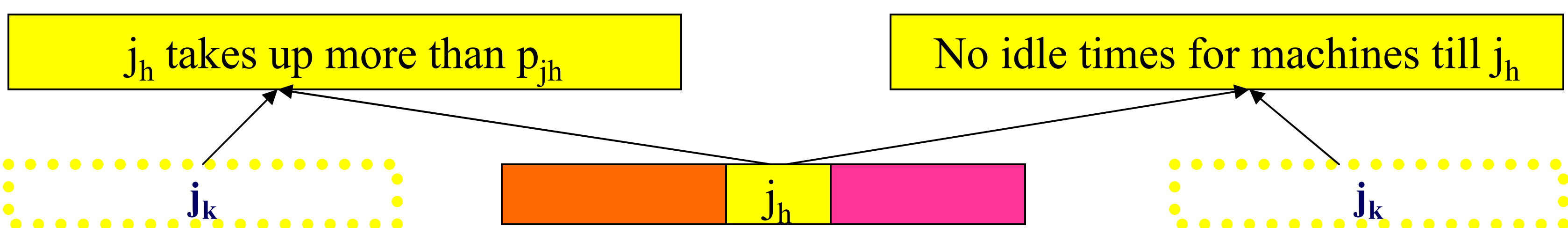
$$C_{max} = \Sigma p_j + (m-1)\max(p_1,\ldots,p_n)$$

- Hence, with limited space, at least as large
- To prove:
  - SPT-LPT will have $C_{max}$ equal to above
  - Any schedule other than SPT LPT will have larger makespan than above

# $F_m \mid block, p_{ij} = p_j \mid C_{max}$

- SPT Portion – jobs *never blocked;* each preceding job is smaller
- Cjk = $\Sigma\, p_{jl} + mp_{jk}$ *(summed over 1 to m-1)*
- LPT Portion – shorter jobs follow longer ones – blocking – but machine *never waits*
- Presence or absence of buffer space NOT important
- Hence, result similar to unlimited buffers case (we know SPT-LPT is optimal)

- That SPT-NPT only is optimal – proved by contradiction
- Consider another schedule (non-SPT-LPT) that's optimal
- Job with longest $p_{jk}$ contributes $mp_{jk}$ in both cases
- Since new schedule is non-SPT-LPT, jh exists such that it is surrounded by 2 jobs with longer processing times

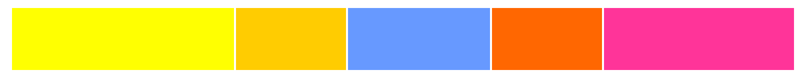


# $F_m \mid block \mid C_{max}$

- Solved by heuristics
- Profile Fitting Heuristic most popular ; Profile → $D_{ij}$ s

Select a job (k) to be sent first, $j_1$

Any policy



**No blocking; $D_{ij1} = \Sigma p_{ik}$**

Next job?

For each job j, $j = 1, \ldots, n$, $j \neq k$

Compute "time wasted" = $D_{ij1} - D_{ij2} - p_{ij2}$

For every machine i

Select smallest

Register profile

All machines treated equally; But bottleneck machines more important w.r.t wasted time

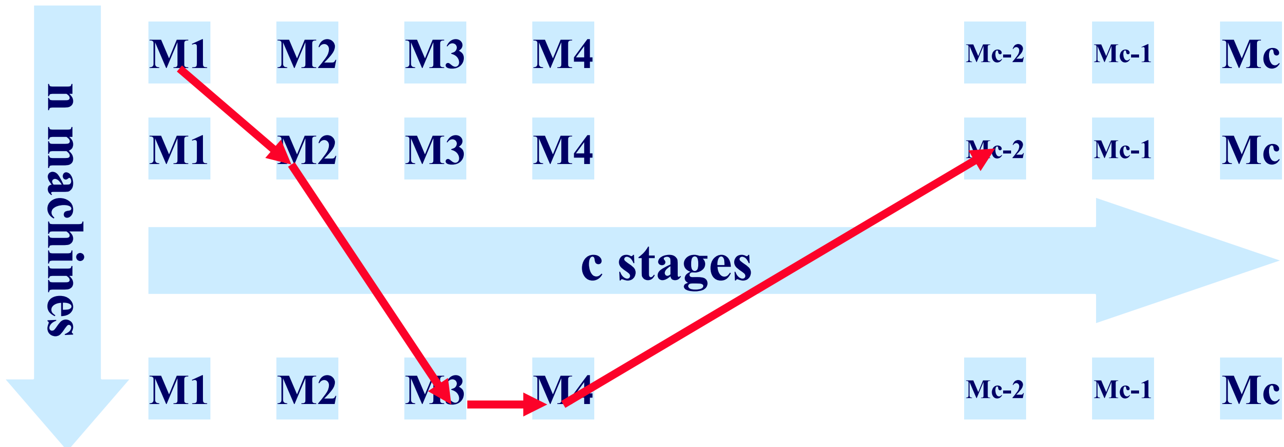
Multiply computed lost time with a weight factor

# No Wait Flow Shops

- No wait as opposed to No block
- ➔ when a machine is done, it turns "idle"
- Jobs progress by "pull down" strategy
- $F_m \mid nwt \mid C_{max}$
- $F_2 \mid nwt \mid C_{max} = F_2 \mid block \mid C_{max}$
- M > 2, "no wait" and "block" are different
- Strongly NP Hard
- TSP (n+1 cities) formulation is different; different intercity distances with complicated calculations

# Flexible Flow Shops

# Flexible Flow Shops –UNLIMITED Buffer space




Any job on any machine within a stage

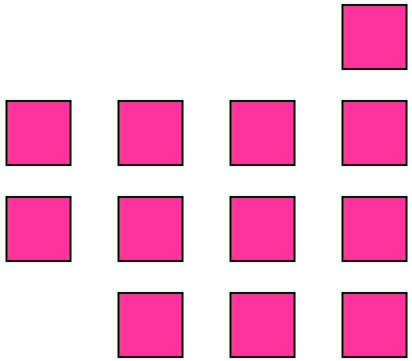
Complex; Parallel single stage case itself hard

Only proportionate FFS considered

# $FFC \mid p_{ij} = p_j \mid XXX$

- LPT heuristic non-preemptive case (worst case worse than single stage)
- LRPT heuristic in preemptive case (not optimal)
  - first stage jobs finish late
  - $2^{nd}$ stage machines inordinately idle
- SPT optimality for $FFC \mid p_{ij} = p_j \mid \Sigma C_j$
  - Exists only when FFS *diverges*

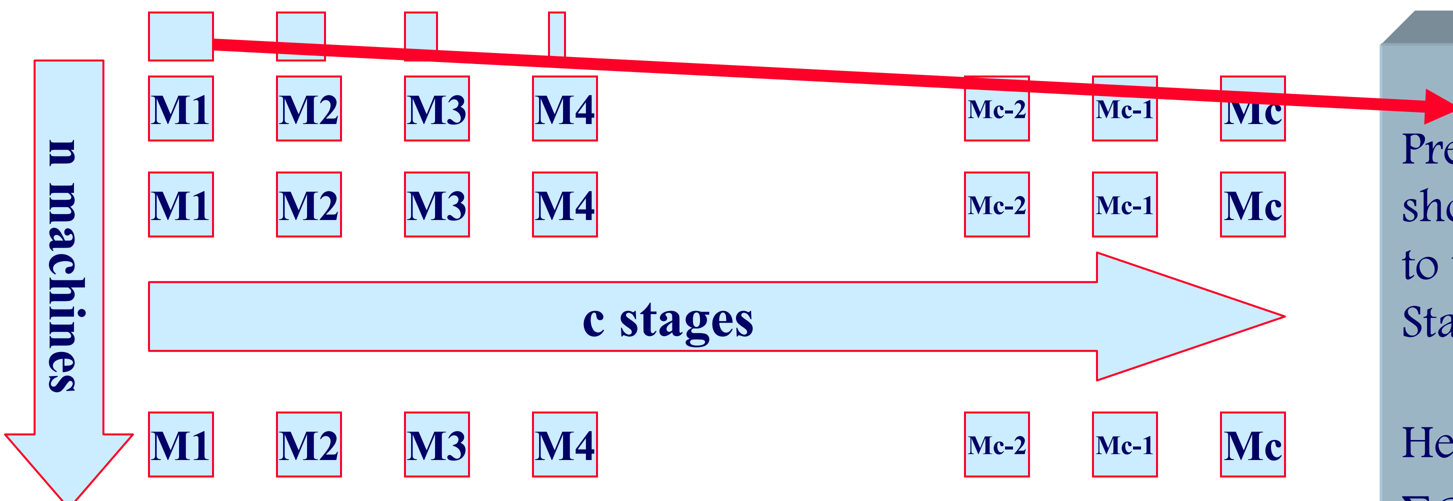
Divergence: At least as many machines as in previous stage



# Divergent FFC | $p_{ij} = p_j$ | $\Sigma C_j$

- Proof of SPT Optimality
- Single stage optimality of Total Completion time clear (Thm 5.3.1) (*sum of starting times also*)

FFS with c stages ➔ $C_j$ of job j will be at least $cp_j$ from starting time of job j

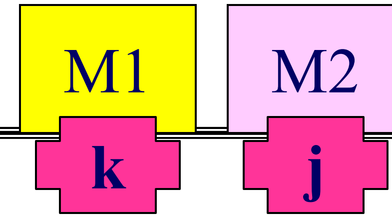



Previous job shorter; no need to wait at any Stage

Hence, $\Sigma C_j = \Sigma C_1 + \Sigma cp_j$

Arbitrary proportionate FFS: result does NOT hold

# TSP Analogy




- $F_2$ | block| $C_{max}$ with zero buffer zone
- When Job j starts of Machine 1, Job j-1 starts on Machine 2
- Job j can be
  - a) processed on Machine 2 immediately after Machine 1 ➔ $p1,j_k$
  - b) blocked because Job $j_{k-1}$ is on Machine 2 ➔ $p_{2,jk-1}$
- Hence processing time for Job $j_k$ = Max ($p_{1,jk}$ , $p_{2,jk-1}$)
- First job $j_1$ processing time = $p_{1,j1}$
- Similar to TSP problem with n+1 cities ➔
- Distance from city j to city k
  - $d_{0k} = p_{1k}$; $d_{j0} = p_{2j}$; $d_{jk} = \max(p_{2j}, p_{1k})$ [ distance analogous to time]

Going from city j to city k = job j precedes job k

To touch city k, TS has to travel max ($d_{0k}$ ,$d_{j0}$)