Scheduling with Machine-Dependent Priority Lists

Reichman University

Tami Tamir

Based on joint work with
Vipin Ravindran Vijayalakshmi and
Marc Schroder

Traditional Scheduling Algorithms

- A centralized authority (a scheduler) determines the outcome.
- The centralized authority aims to maximize the system's utilization and the total users' welfare.
- All the users obey it.

Job Scheduling Games

- The jobs are controlled by selfish agents who select the jobs assignment.

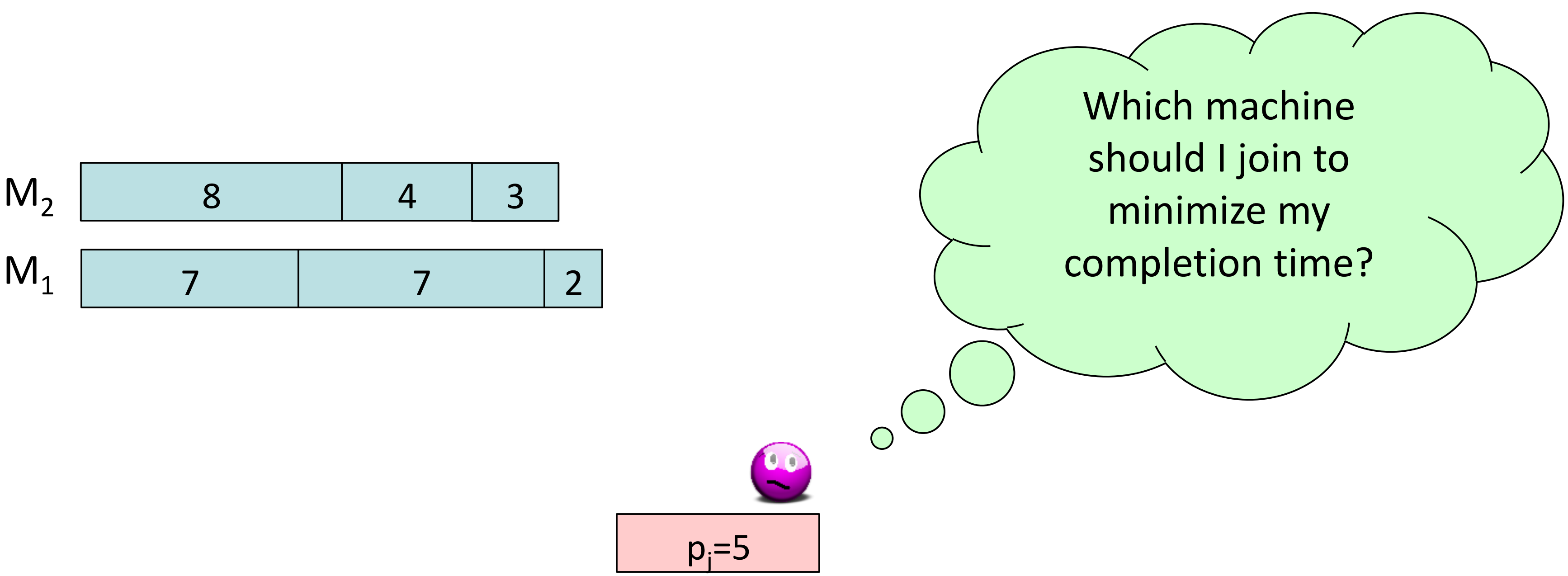
- No centralized authority.

Every job selects its machine, trying to maximize its own utility

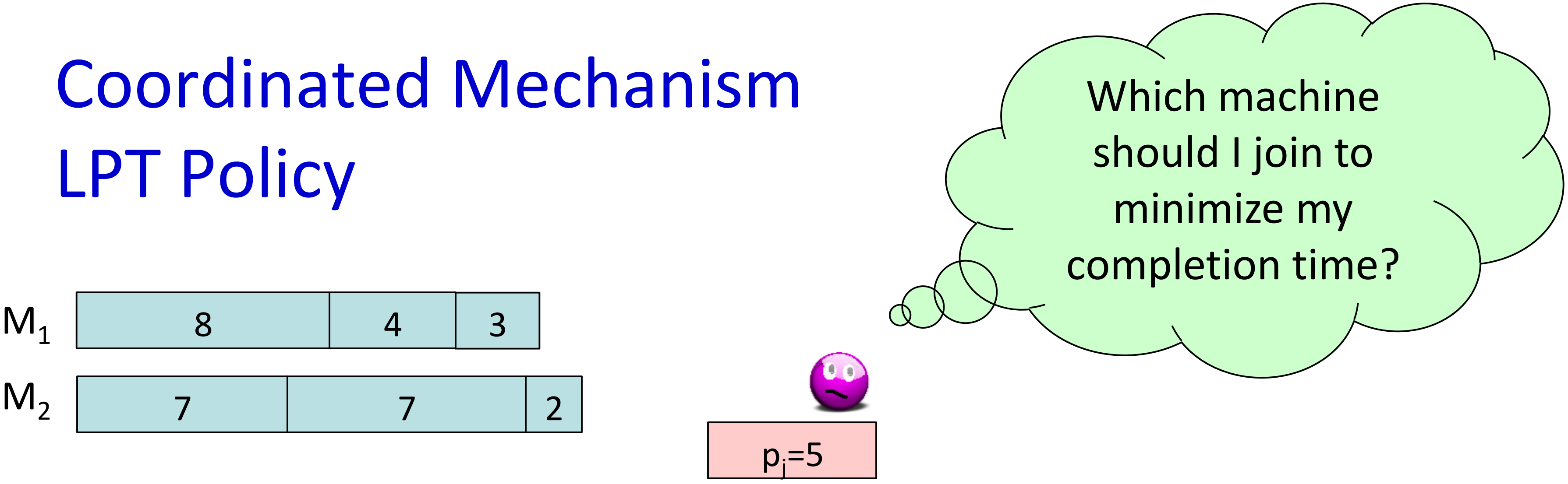
Coordinated Mechanism

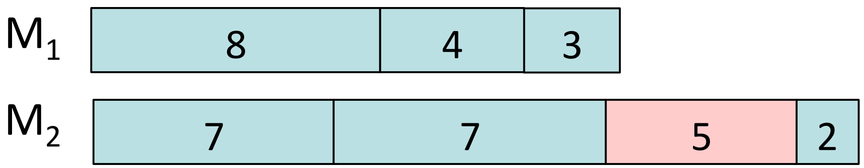
Machines have a local scheduling policy.
The jobs know this policy and select their machine accordingly.

Example: Assume that all the machines schedule the jobs in LPT (Longest first) order.

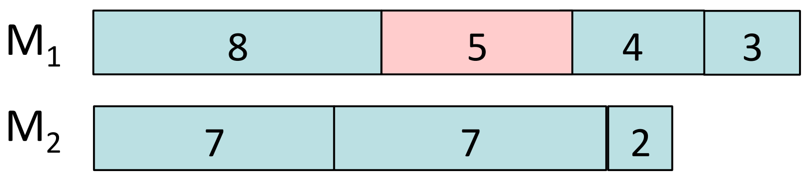


Coordinated Mechanism
LPT Policy





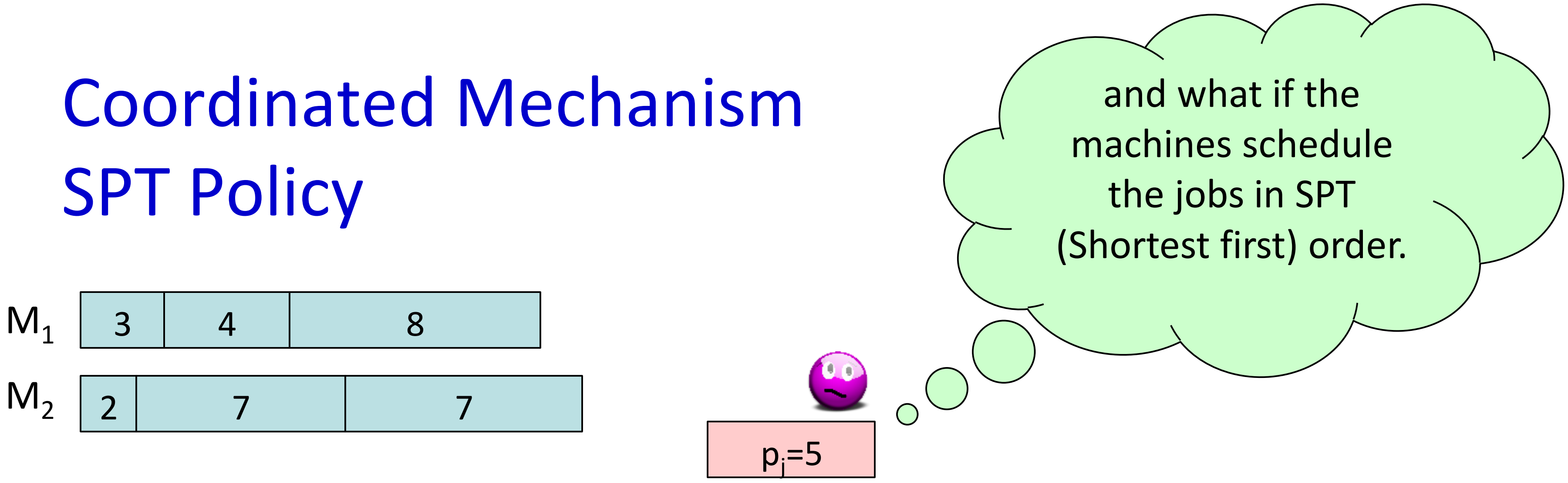
C_j=7+7+5=19 if I join M_2

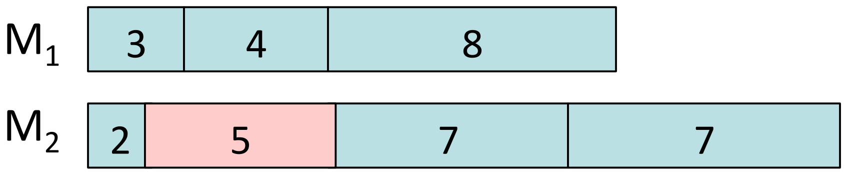


C_j=8+5=13 if I join M_1

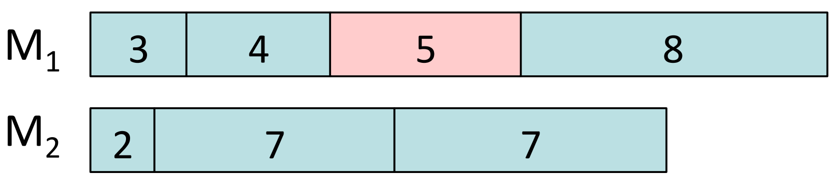
If the local policy is LPT, I'll better join M_1.
This is my best-response.

Coordinated Mechanism
SPT Policy





C_j=2+5=7 if I join M_2



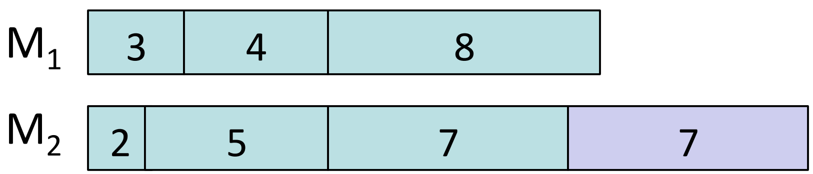
C_j=3+4+5=12 if I join M_1

If the local policy is SPT, my best-response is to join M_2.

Coordinated Mechanism
SPT Policy

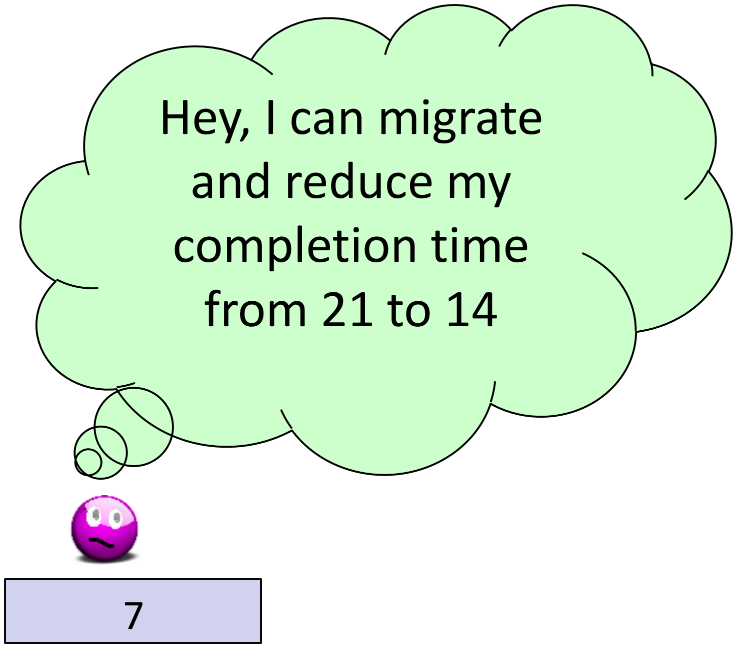
Assume that 

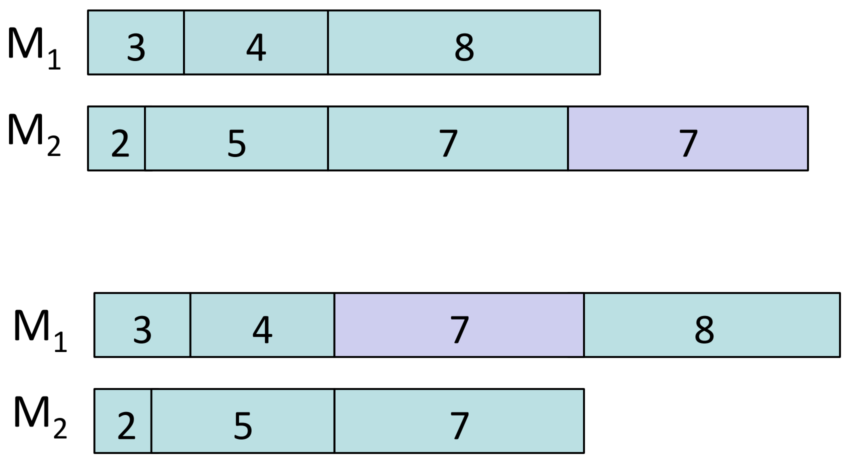
joins M_2



Consider the 2[nd] job of length 7 in the resulting schedule

Coordinated Mechanism
SPT Policy





Now, other jobs may have a beneficial migration...

Best Response Dynamics (BRD)

- A local search method.
- Players proceed in turns, each performing a selfish improving step.
- An important question: Does BRD converge to a pure Nash equilibrium.

A stable profile in which no player has an improving step.

Our Work

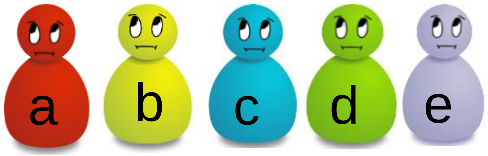
We study coordinated mechanisms in which different machines may have different local policies.

For the associated game, we analyzed:

- Nash equilibrium existence and calculation
- BRD convergence
- Equilibrium inefficiency

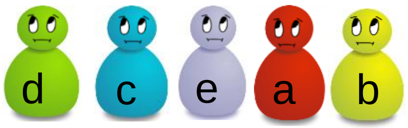
Not less important: We studied the centralized version of this setting.

The Setting



- A set J of n jobs
  - Every job $j \in J$ has processing time p_j
- A set M of m parallel machines
  - Every machine $i \in M$ has speed s_i and a priority list $\pi_i: J \rightarrow \{1,\ldots,n\}$, defining its scheduling policy.

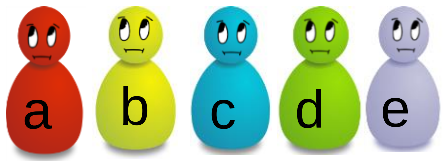
$\pi_1 =$ (



)

1 2 3 4 5

The Setting

Example: $J=\{$ 

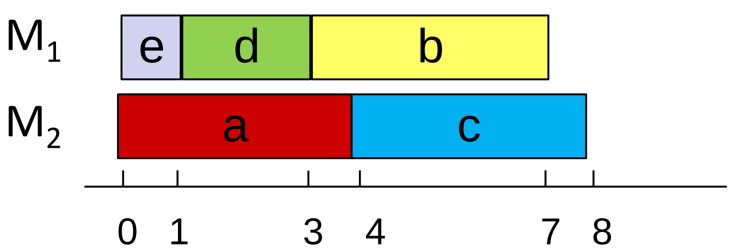
$\}$,

$p_j=$ 2 4 2 2 1 processing times

$m=2$,

$s_1=1$ $\pi_1 = (e,d,c,b,a)$

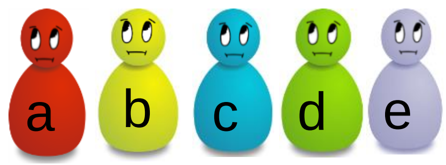
$s_2=0.5$ $\pi_2 = (a,b,c,d,e)$



$C_j=(4,7,8,3,1)$

A profile of the game:
A schedule $\sigma: J \rightarrow M$.
$C_j(\sigma)$ = the completion time of job j in profile σ

The Setting

Example: J={ 

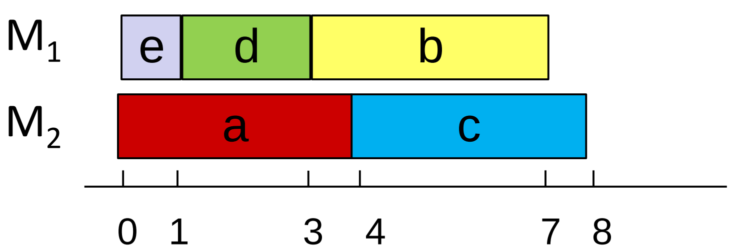
},

p_j= 2 4 2 2 1 processing times

$m=2$,

$s_1=1$ $\pi_1 = (e,d,c,b,a)$

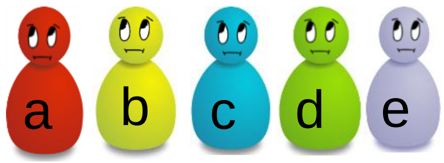
$s_2=0.5$ $\pi_2 = (a,b,c,d,e)$



$C_j=(4,7,8,3,1)$

Does anyone have a beneficial migration?

The Setting

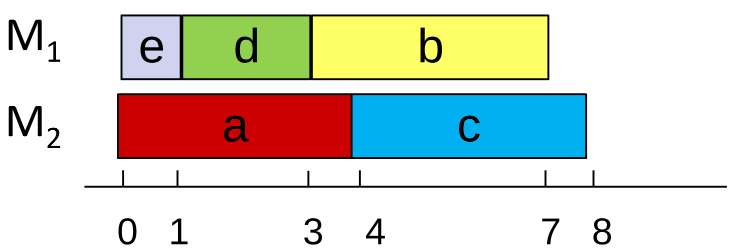
Example: J={  },

p_j= 2 4 2 2 1 processing times

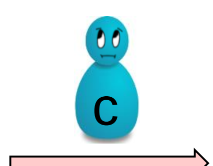
m=2,

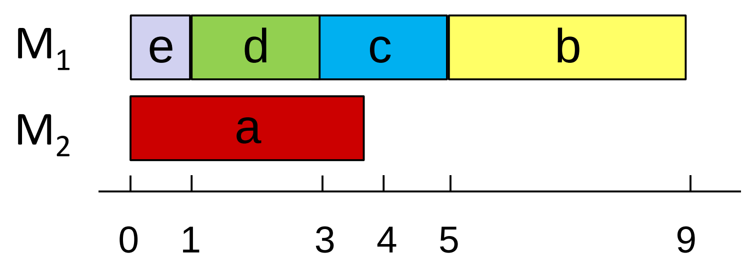
s_1=1 π_1 = (e,d,c,b,a)

s_2=0.5 π_2 = (a,b,c,d,e)



C_c=8



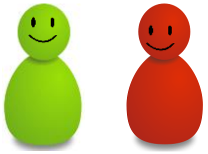


C_c=5

Game Theory Definitions

A profile is a pure Nash equilibrium (NE) if no job can reduce its completion time by changing its strategy (migrating to a different machine)

A social optimum (SO) of a game is a profile that attains some optimality criteria.
For example:
social optimum w.r.t total flow time (=sum of C_j)
social optimum w.r.t makespan (=maximal C_j).

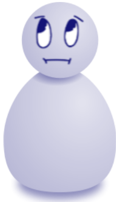


SO = Optimal solution for the centralized problem $P|\pi|C_{max}$ or $P|\pi|\sum_j C_j$

Interesting Questions

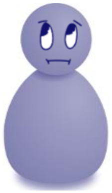
- Calculating a NE for a given game instance
- What is the equilibrium inefficiency?

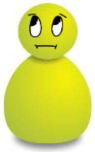
Price of Anarchy $\stackrel{\text{def}}{=}$ worst NE / SO



Price of Stability $\stackrel{\text{def}}{=}$ best NE / SO

- Convergence of Best-Response Dynamics

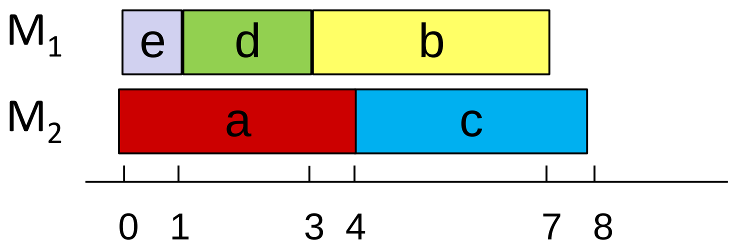




Back to our example

$s_1=1 \quad \pi_1 = (e,d,c,b,a)$

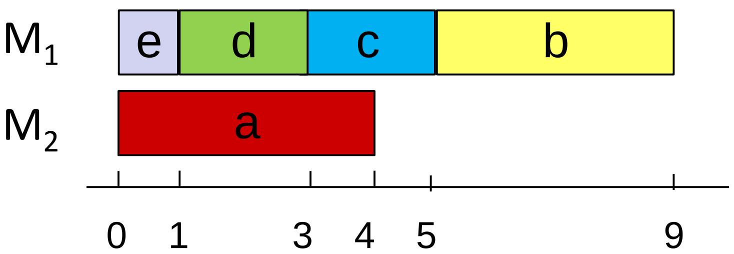
$s_2=0.5 \quad \pi_2 = (a,b,c,d,e)$



$C_j=(4,7,8,3,1)$

$C_{max}=8$

A social optimum w.r.t Makespan, but not a NE.



$C_j=(4,9,5,3,1)$

A possible NE profile.

$C_{max}=9$

Price of anarchy $\geq 9/8$

Related Work

- Koutsoupias and Papadimitriou (1999)
- Czumaj and Vocking (2003)
- Christodoulou, Koutsoupias and Nanavati (2004)
- Cole, Correa, Gkatzelis, Mirrokni and Olver (2015)
- Immorlica, Li, Mirrokni and Schulz (2005)
- Farzad, Olver and Vetta (2008)
- Correa and Queyranne (2012)
- Cole, Correa, Gkatzelis, Mirrokni and Olver (2015)
- Hoeksma and Uetz (2019)
- Bosman, Frascaria, Olver, Sitters, Stougie (2019)

Selfish Scheduling / Coordinated mechanism /Priority-based model of routing/ The centralized problem.

NE Calculation

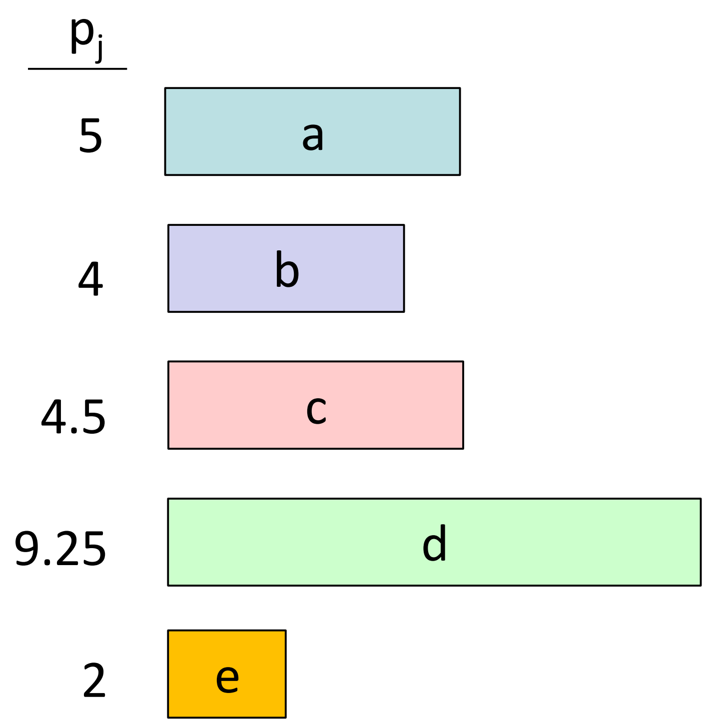
Given $\langle J, M, \{s_i\}, \{\pi_i\} \rangle$, calculate a NE profile

Wait, who says that
NE exists?

NE existence

$n=5$

$J=\{a,b,c,d,e\}$

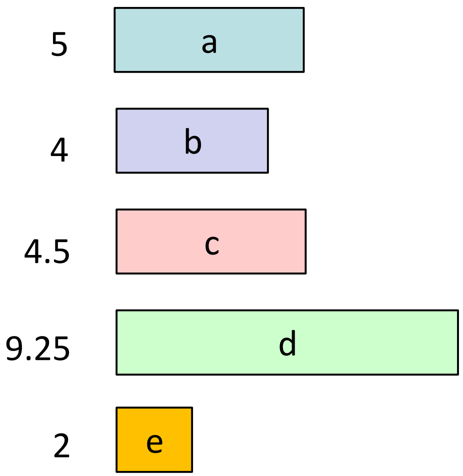


$m=3$, $M=\{M_1,M_2,M_3\}$

$s_1=1$ $\pi_1 = (a,b,c,d,e)$ One fast machine.

$s_2=s_2=0.5$ $\pi_2 = \pi_3 = (e,d,b,c,a)$ Two slow machines

NE existence

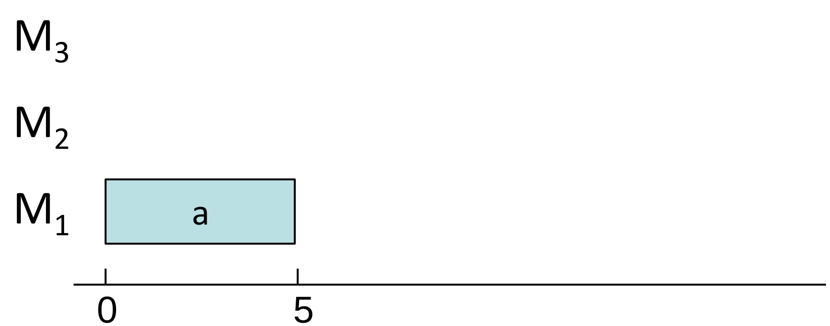


Since $\boxed{a}$ is the first on π_1, it is first on M_1 in any NE schedule.

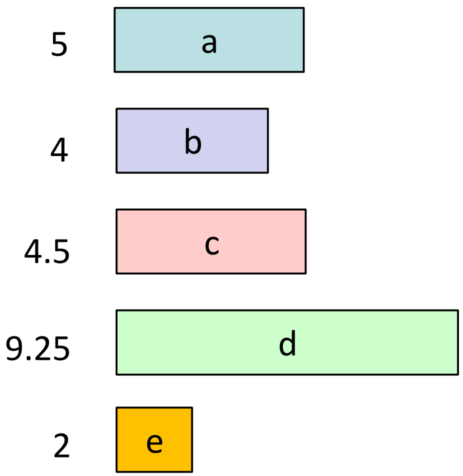
$s_3=0.5$, $\pi_3 =(e,d,b,c,a)$

$s_2=0.5$, $\pi_2 =(e,d,b,c,a)$

$s_1=1$, $\pi_1 =(a,b,c,d,e)$



NE existence

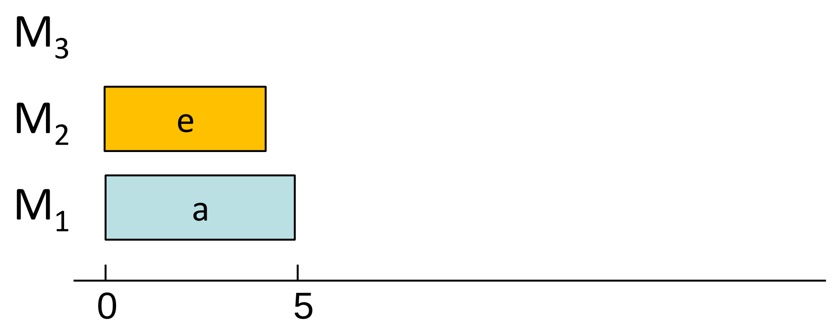


Given that a is on M_1, Since e is the first on π_2, it is first on M_2 (w.l.o.g) in any NE schedule

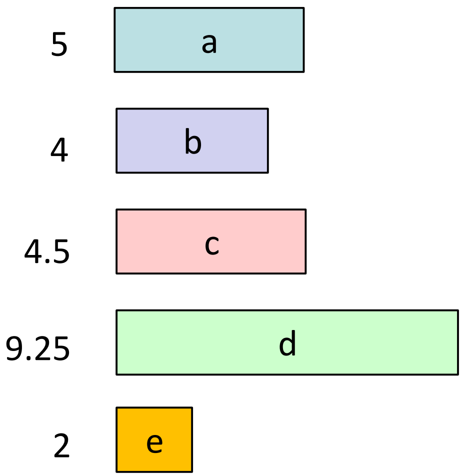
s_3=0.5, π_3 =(e,d,b,c,a)

s_2=0.5, π_2 =(e,d,b,c,a)

s_1=1, π_1 =(a,b,c,d,e)



NE existence

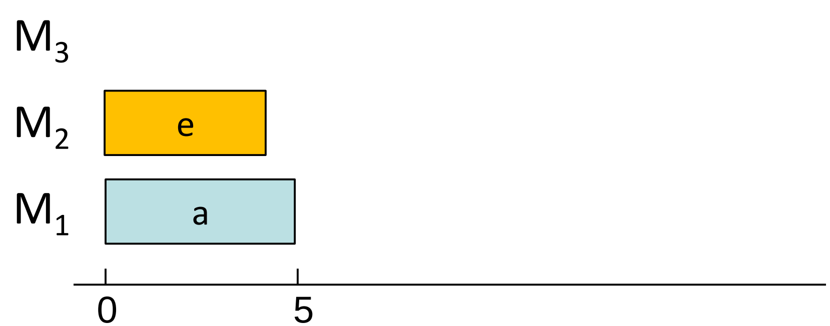


So we know that if a NE exists, then a and e are first on M_1 and M_2, respectively.
Let's consider the possible locations of job d

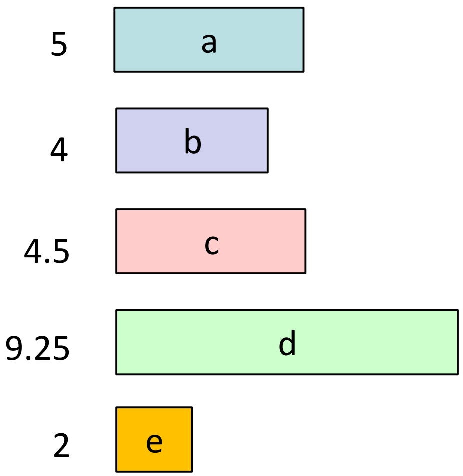
$s_3=0.5$, $\pi_3 =(e,d,b,c,a)$

$s_2=0.5$, $\pi_2 =(e,d,b,c,a)$

$s_1=1$, $\pi_1 =(a,b,c,d,e)$



NE existence



Case 1: Job d on M_1

Job b would go to M_3

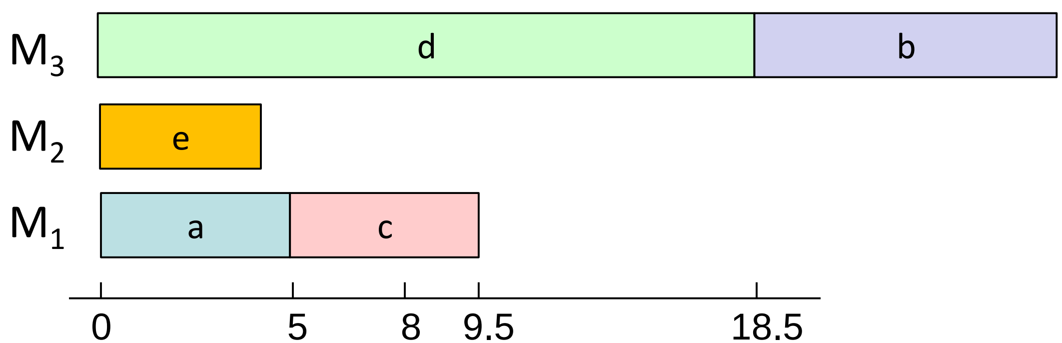
Job c would go to M_1

Job d would go to M_3

s_3=0.5, π_3 =(e,d,b,c,a)

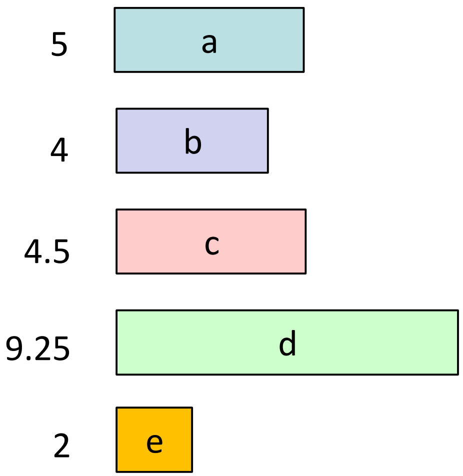
s_2=0.5, π_2 =(e,d,b,c,a)

s_1=1, π_1 =(a,b,c,d,e)



Therefore, there is no NE in which d is on M_1

NE existence

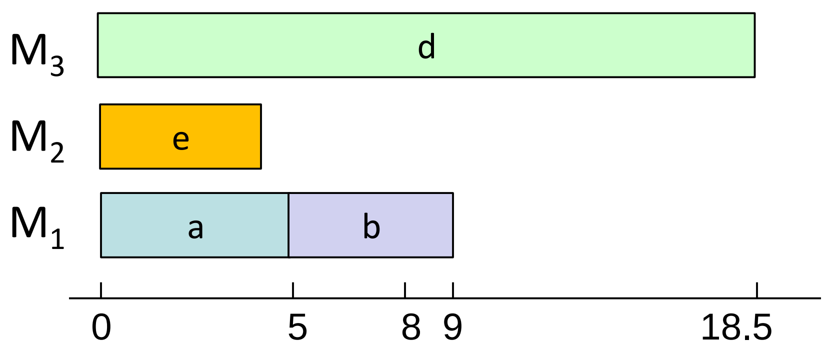


Case 2: Job d on M_2.
It would move to M_3.

$s_3=0.5$, $\pi_3 = (e,d,b,c,a)$

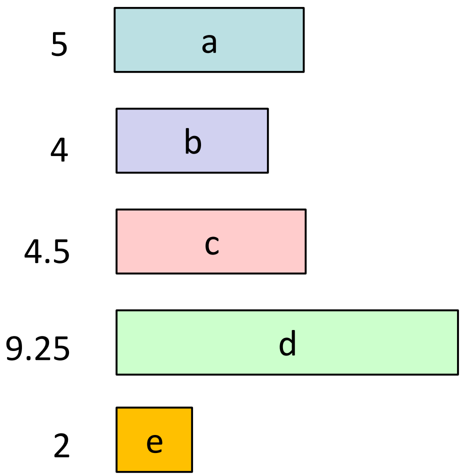
$s_2=0.5$, $\pi_2 = (e,d,b,c,a)$

$s_1=1$, $\pi_1 = (a,b,c,d,e)$



Therefore, there is no NE in which d is on M_2

NE existence



Case 3: Job d on M_3

Job b would go to M_1

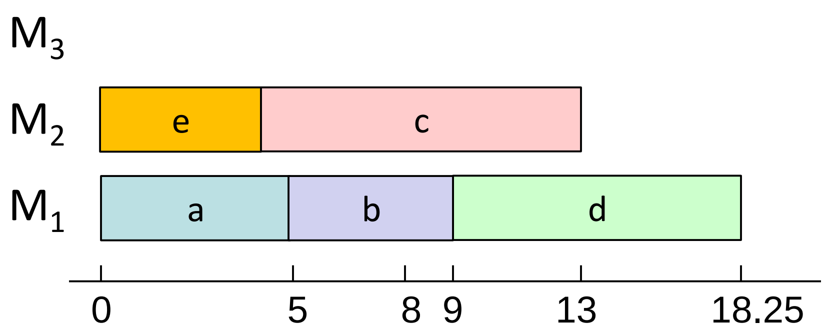
Job c would go to M_2

Job d would go to M_1

s_3=0.5, π_3 =(e,d,b,c,a)

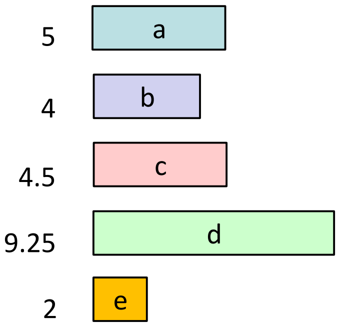
s_2=0.5, π_2 =(e,d,b,c,a)

s_1=1, π_1 =(a,b,c,d,e)

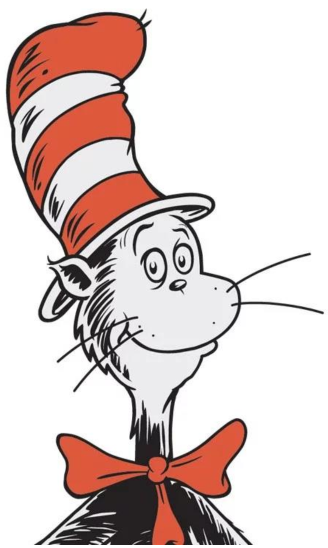


Therefore, there is no NE in which d is on M_3

NE existence



We conclude that there are games in which a NE does not exist



NE existence

Can we characterize games that
have a NE?

Unfortunately, No.

Theorem: Given an instance of a scheduling game, it is NP-complete to decide whether the game has a NE.

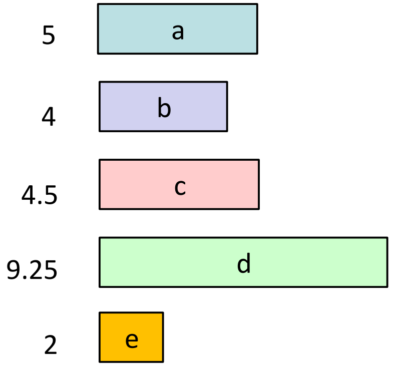
Proof: Reduction from 3-bounded 3-dimensional matching

On the other hand:

We identified four classes of games for which a NE is guaranteed to exist.

$\mathcal{G}_1$: Unit Jobs
$\mathcal{G}_2$: Two machines
$\mathcal{G}_3$: Identical machines
$\mathcal{G}_4$: Global priority list

Note: This characterization is tight. In our No-NE example, there are three machines, two of them are identical (same speed and same priority list)



For each of the four classes, we present:

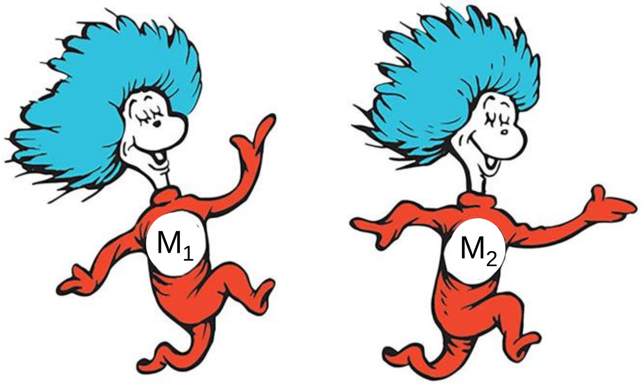
- A polynomial time algorithm for computing a NE
- A proof that BRD converges to a NE
- Tight analysis of the equilibrium inefficiency:

Objective Instance class	Makespan PoA and PoS	Sum of Completion Time PoA and PoS
$\mathcal{G}_1$: Unit Jobs	1	1
$\mathcal{G}_2$: Two machines	$\frac{\sqrt{5}+1}{2}$	$\Theta(n)$
$\mathcal{G}_3$: Identical machines	$2-\frac{1}{m}$	$\Theta\left(\frac{n}{m}\right)$
$\mathcal{G}_4$: Global priority list	$\Theta(m)$	$\Theta(n)$

Two machines

Theorem: If $m = 2$, then a NE exists and can be calculated efficiently.

Proof: Algorithm

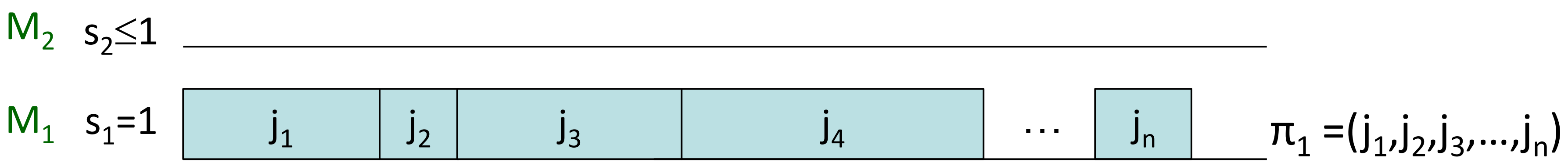


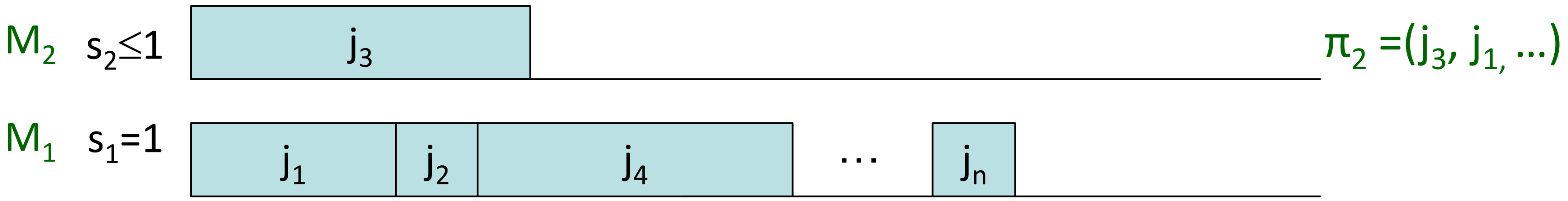
$s_1=1$ $s_2=s\leq 1$

Two machines

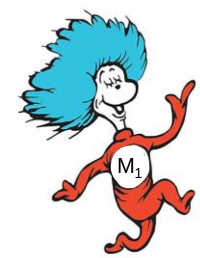
Algorithm:

1. Assign all the jobs on M_1 according to π_1.
2. For $k = 1,\ldots,n$, let job j for which $\pi_2(j) = k$ perform a best-response move.

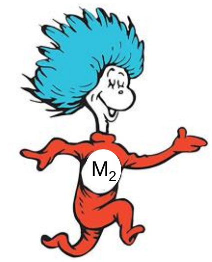




Two machines



$s_1=1$



$s_2=s\leq 1$

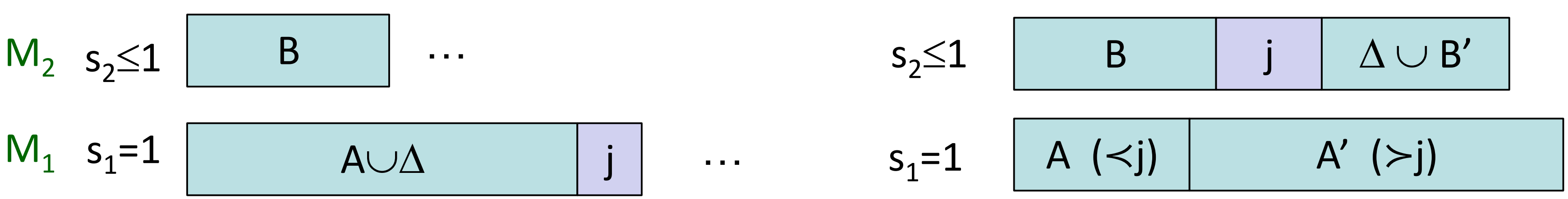
Claim: The algorithm produces a NE.

Proof:

Let σ denote the schedule produced by the algorithm.

1. Jobs on M_1 have no incentive to deviate (easy).
2. Suppose a job j on M_2 has an incentive to deviate.

Let Δ be the set of jobs that have a higher priority on M_1 than j and moved to M_2 after j.



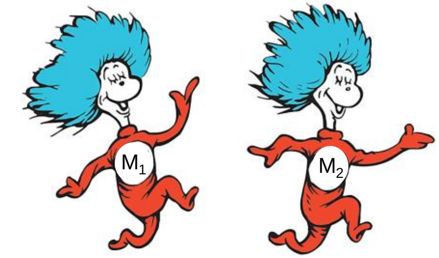
Before j was considered by the algorithm

now

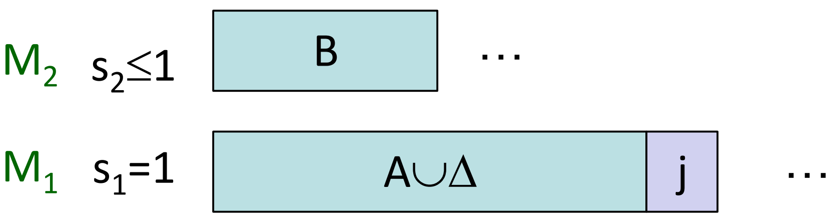
$\pi_1 = (\ldots,\Delta,\ldots,B,\ldots)$

$\pi_2 = (\ldots,B,\ldots,\Delta,\ldots)$

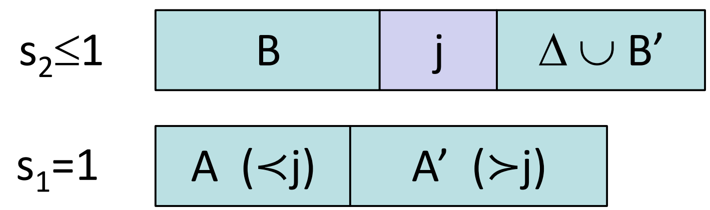
Two machines



s_1=1 s_2=s≤1



Before j was considered by the algorithm



now

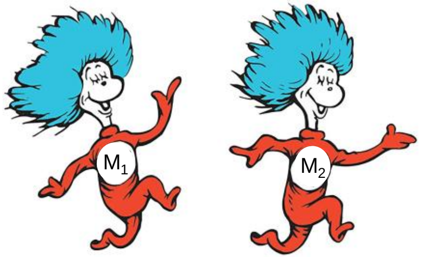
(i) $P_A + p_j < (P_B + p_j)/s_2$

(ii) $(P_B + p_j + P_\Delta)/s_2 < P_A + P_\Delta$

$\Rightarrow$ $p_j + P_\Delta/s_2 < P_\Delta$

A Contradiction (since $p_j \ge 0$ and $s_2 \le 1$)

Two machines



Remark: A possible generalization of our setting considers unrelated machines (p_{ij} is the processing time of job i if processed on machine j).

In this environment, a NE need not exist already with only two unrelated machines.

Equilibrium inefficiency

The makespan of a profile σ, is $C_{max}(\sigma) = max_{j \in J} C_j(\sigma)$

For a game G,

$$PoA(G) = \frac{max_{\sigma \in NE(G)}\, C_{max}(\sigma)}{min_{\sigma^*} C_{max}(\sigma^*)} = \frac{\text{makespan of the worst NE schedule}}{\text{min makespan (social optimum)}}$$

For a class of games $\mathcal{G}$, define $PoA(\mathcal{G}) = sup_{G \in \mathcal{G}} PoA(G)$

Equilibrium inefficiency

The makespan of a profile σ, is $C_{max}(\sigma) = max_{j\in J} C_j(\sigma)$

For a game G,

$$PoA(G) = \frac{max_{\sigma\in NE(G)}\, C_{max}(\sigma)}{min_{\sigma^*} C_{max}(\sigma^*)} = \frac{\text{makespan of the worst NE schedule}}{\text{min makespan (social optimum)}}$$

For a class of games $\mathcal{G}$, define $PoA(\mathcal{G}) = sup_{G\in\mathcal{G}} PoA(G)$

Instance class	Makespan PoA
$\mathcal{G}_1$: Unit Jobs	1
$\mathcal{G}_2$: Two machines	$\frac{\sqrt{5}+1}{2}$
$\mathcal{G}_3$: Identical machines	$2-\frac{1}{m}$
$\mathcal{G}_4$: Global priority list	$\Theta(m)$

Equilibrium inefficiency, two machines.

Theorem: Let G be a game played on two machines, $s_1 = 1$ and $s_2 \leq 1$, then $PoA(G) \leq min\left\{1 + s_2, 1 + \frac{1}{1+s_2}\right\}$

Since $1 + s = 1 + \frac{1}{1+s}$ for $s = \frac{\sqrt{5}-1}{2}$, the theorem implies that $PoA(\mathcal{G}_2) \leq \frac{\sqrt{5}+1}{2}$.

Equilibrium inefficiency, two machines.

Theorem: Let G be a game played on two machines, $s_1 = 1$ and $s_2 \leq 1$, then $PoA(G) \leq min\left\{1 + s_2, 1 + \frac{1}{1+s_2}\right\}$

Proof: Let σ be a NE.

1. $$C_{max}(\sigma) \leq \sum_{j \in J} p_j$$ (if all jobs on fast machine)

2. $$C_{max}(\sigma^*) \geq \frac{\sum_{j \in J} p_j}{1 + s_2}$$ (balanced)

Implying that $C_{max}(\sigma) \leq (1 + s_2) \cdot C_{max}(\sigma^*)$.

Equilibrium inefficiency, two machines.

Theorem: Let G be a game played on two machines, $s_1 = 1$ and $s_2 \leq 1$, then $PoA(G) \leq min\left\{1 + s_2, 1 + \frac{1}{1+s_2}\right\}$

Proof: Let a be the last job to complete in a NE σ.

1. $C_{max}(\sigma) \leq p_a + \sum_{j \neq a: \sigma_j = 1} p_j$ (a can go to fast machine)
2. $C_{max}(\sigma) \leq (p_a + \sum_{j \neq a: \sigma_j = 2} p_j)/s_2$ (a can go to slow machine)

Implying that ($C_{max}(\sigma) \geq p_a$)

$$C_{max}(\sigma) \leq \frac{p_a + \sum_{j \in J} p_j}{1 + s_2} \leq \left(1 + \frac{1}{1 + s_2}\right) \cdot C_{max}(\sigma^*).$$

■

Equilibrium inefficiency, two machines.

Theorem: For every $s \leq 1$, there exists a game with $s_1 = 1$, $s_2 = s$, and $PoS(G) = min\left\{1 + s, 1 + \frac{1}{1+s}\right\}$.

$$PoS(G) = \frac{\text{makespan of the best NE schedule}}{\text{min makespan (social optimum)}}$$

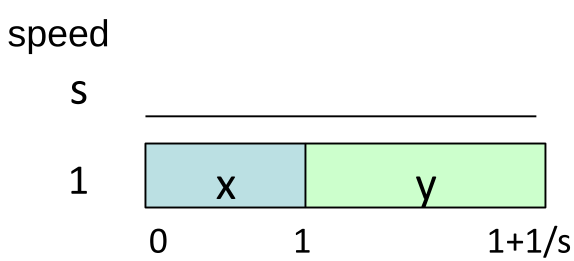
Equilibrium inefficiency, two machines.

Theorem: For every $s \leq 1$, there exists a game with $s_1 = 1$, $s_2 = s$, and $PoS(G) = min\left\{1 + s, 1 + \frac{1}{1+s}\right\}$.

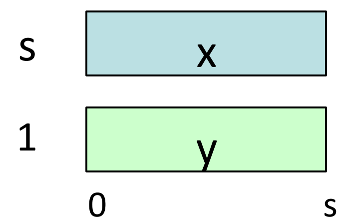
$1 + s = 1 + \frac{1}{1+s}$ for $s = \frac{\sqrt{5}-1}{2}$

Proof: case 1: $s \leq \frac{\sqrt{5}+1}{2}$.

Let J={x,y}, $p_x = 1$, $p_y = \frac{1}{s}$ $\qquad \pi_1 = \pi_2 = (x, y)$.



Unique NE
($1+1/s < 1/s^2$)



Social optimum

(*) if $s = \frac{\sqrt{5}-1}{2}$, take $p_y = \frac{1}{s} - \epsilon$)

PoS = 1+s

Equilibrium inefficiency, two machines.

case 2: $s > \frac{\sqrt{5}-1}{2}$.

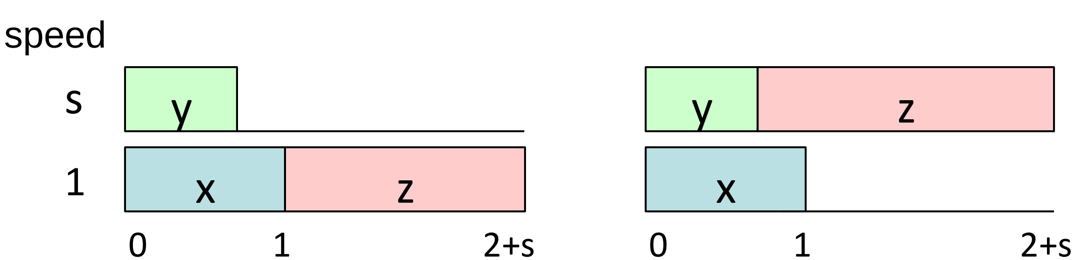
$J=\{x,y,z\}$, $p_x = 1$, $p_y = s^2+s-1$, $p_z = 1+s$.

$\pi_1 = \pi_2 = (x, y, z)$.

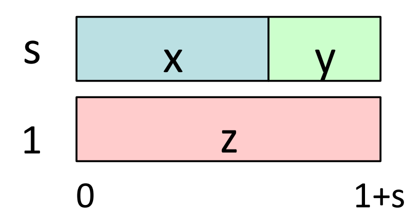
In all NE: (1) x is on the fast machine

(2) y is on the slow machine since $s^2 + s > (s^2+s-1)/s$.

(3) z is indifferent. $p_x + p_z = (p_y + p_z)/s = 2+s$.



NE #1 NE #2



Social optimum

$$\text{PoS} = \frac{2+s}{1+s} = 1 + \frac{1}{1+s}$$

Equilibrium inefficiency, Identical machines

Theorem:

If s_i= 1 for all i$\in$ M, then PoA(G) $\leq 2 - \frac{1}{m}$

Proof:

We show that any NE is a possible outcome of Graham's List-scheduling algorithm

Theorem:

If s_i= 1 for all i$\in$ M, then it is NP-hard to approximate the best NE within a factor of $2 - \frac{1}{m} - \epsilon$ for all $\epsilon > 0$.

Proof:

Reduction from 3D-matching.

Back to centralized setting (not a game)

- A set J of n jobs, and a set M of m parallel machines
  - Every job $j \in J$ has processing time p_j
  - In case of unrelated machines, p_{ij} is the processing time of job j on machine i.
  - Every machine $i \in M$ has a priority list $\pi_i: J \rightarrow \{1,\ldots,n\}$, defining its scheduling policy.

The Goal: Find a schedule that minimizes $\sum_j C_j$

Note: In the centralized setting, priority lists do not \`upgrade' the problem of minimizing the Makespan

The problems $P|\pi|\sum_j C_j$ and $R|\pi|\sum_j C_j$

Without priority lists, both problems are solvable
$P||\sum_j C_j$ - SPT is optimal [Smith 1956]
$R||\sum_j C_j$ - can be represented as a bipartite weighted matching problem [Bruno, Coffman, Sethi 1974]

Theorem: $P|\pi|\sum_j C_j$ is APX-hard ☹

We therefore consider several restricted classes:

- Global priority list
- Fixed number of machines
- Fixed number of priority classes

The problems $P|\pi|\sum_j C_j$ and $R|\pi|\sum_j C_j$

Our results:

	π_i	π_{global}	π_{LPT}	$\pi_{i,c}$	$\pi_{global,c}$
P	APX-hard	QPTAS	P	APX-hard	P
R	APX-hard	APX-hard	APX-hard	APX-hard	APX-hard

$\pi_{i,c}$ and $\pi_{global,c}$: the jobs are partitioned into c job classes $J_1, \ldots, J_c$. For every $1 < k \leq c$, every machine processes jobs from J_k after it processes jobs from $\cup_{j<k} J_j$.
Note: in every optimal schedule, for every $1 \leq i \leq m$ and $1 \leq k \leq c$, machine i processes jobs of J_k in SPT order.

The problems P$|\pi|\sum_j C_j$ and R$|\pi|\sum_j C_j$

Our results:

	$\boldsymbol{\pi_i}$	$\boldsymbol{\pi_{global}}$	$\boldsymbol{\pi_{LPT}}$	$\boldsymbol{\pi_{i,c}}$	$\boldsymbol{\pi_{global,c}}$
P	APX-hard	QPTAS	P	APX-hard	P
R	APX-hard	APX-hard	APX-hard	APX-hard	APX-hard

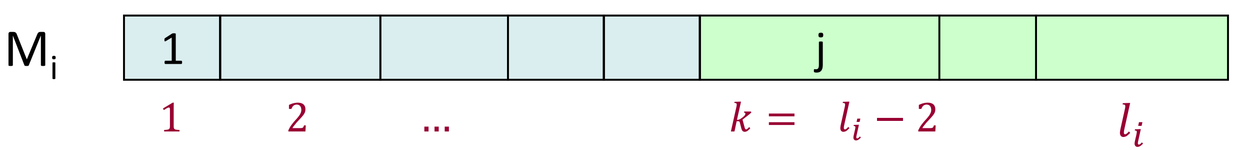
In P if m is a constant

The problems P|π|$\sum_j C_j$ and R|π|$\sum_j C_j$

A Useful Observation: Let l_i denote the number of jobs on machine i.

The job with the k-th highest priority assigned to machine i contributes exactly $l_i + 1 - k$ times its processing time to the sum of completion times.

the delay-coefficient of the job



p_{ij} is counted $l_i + 1 - (l_i - 2) = 3$ times in $\Sigma_j C_j$

π_{LPT} - Longest Processing Time First

π_{LPT} – every machine processes jobs in LPT order.

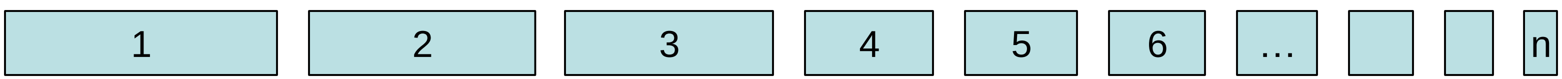
$P|\pi_{LPT}|\sum_j C_j$:
A global priority list $\pi = (1,2,\ldots,n)$, where $p_1 \geq p_2 \geq \ldots \geq p_n$.

$R|\pi_{LPT}|\sum_j C_j$:
For machine i, $\pi_i = (1_i, 2_i, \ldots, n_i)$, where $p_{i,1_i} \geq p_{i,2_i} \geq \ldots \geq p_{i,n_i}$.

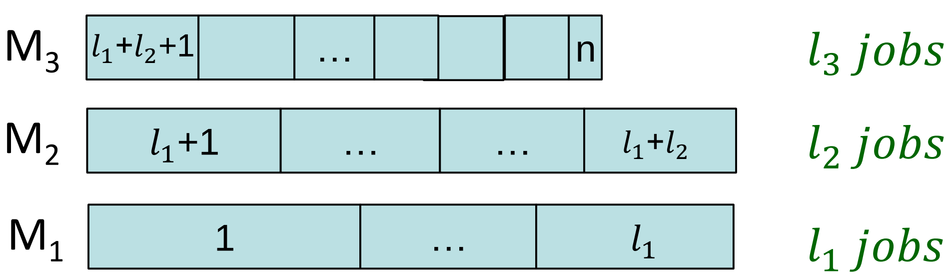
An optimal algorithm for $P|\pi_{LPT}|\sum_j C_j$

Claim: There exists an optimal schedule for $P|\pi_{LPT}|\sum_j C_j$ in which for some $l_1 \le l_2 \le \cdots \le l_m$ such that $\sum_i l_i = n$, it holds that machine i processes the consequent sub-sequence of l_i jobs $1 + \sum_{k<i} l_k, \ldots, \sum_{k\le i} l_k$.

Illustration of the claim:



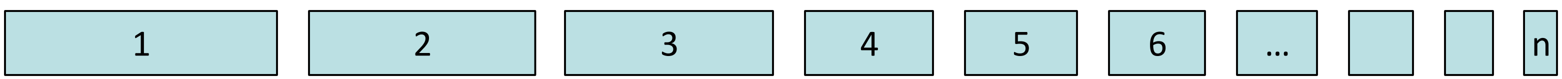
Assume m=3, then some optimal schedule looks like this:



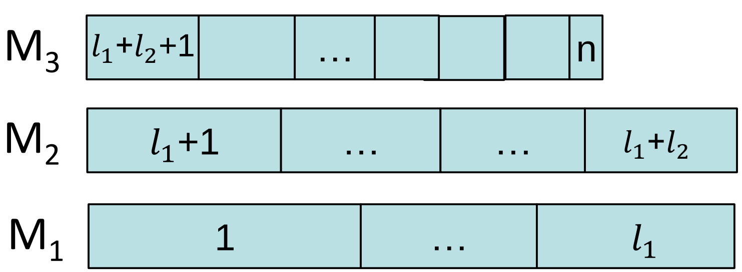
An optimal algorithm for P$|\pi_{\text{LPT}}|\sum_j C_j$

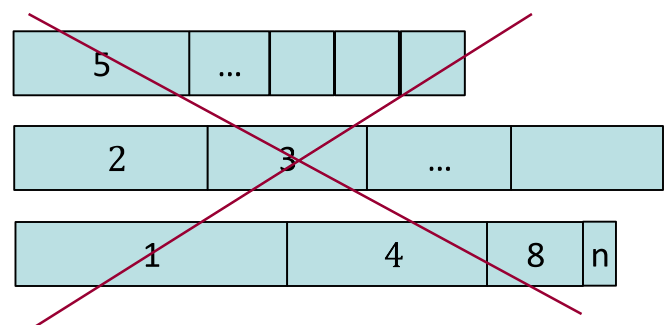
Claim: There exists an optimal schedule for P$|\pi_{\text{LPT}}|\sum_j C_j$ in which for some $l_1 \le l_2 \le \cdots \le l_m$ such that $\sum_i l_i = n$, it holds that machine i processes the consequent sub-sequence of l_i jobs $1 + \sum_{k<i} l_k, \ldots, \sum_{k\le i} l_k$.

Illustration of the claim:



Assume m=3, then some optimal schedule looks like this:

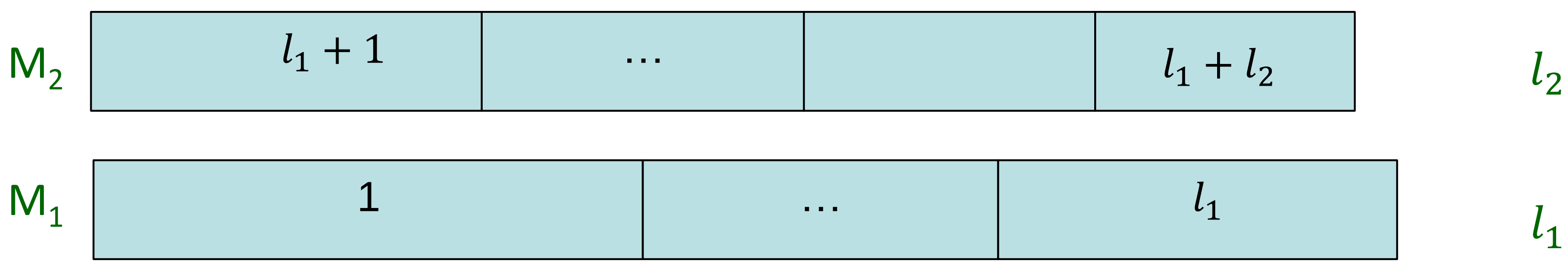




Not structured

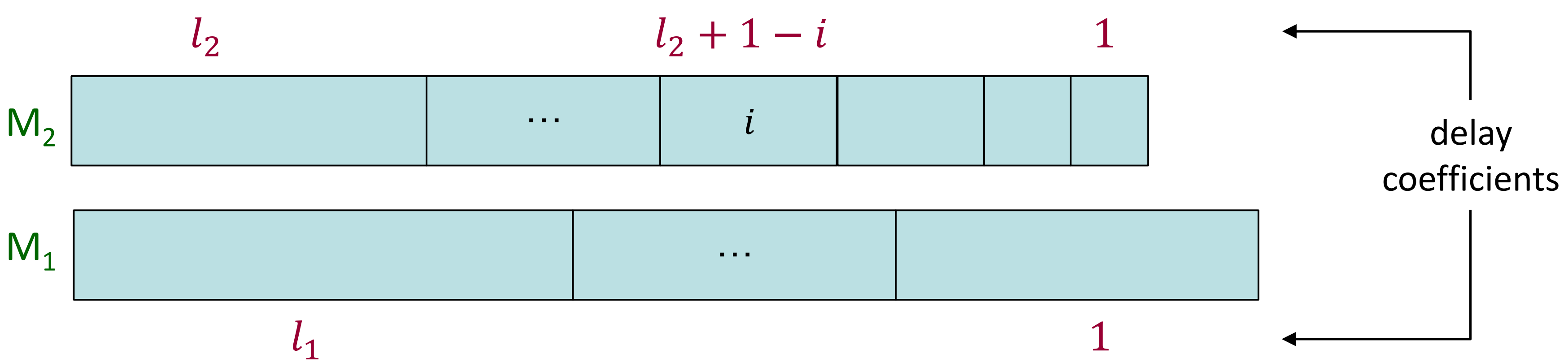
An optimal algorithm for $P|\pi_{\text{LPT}}|\sum_j C_j$

Proof: (for two machines) Assume that we know how many jobs are assigned to each of the machines. W.l.o.g., assume that $l_1 \le l_2$.
We show that in some optimal schedule, M_1 processes the l_1 longest jobs, and M_2 processes the l_2 shortest jobs.



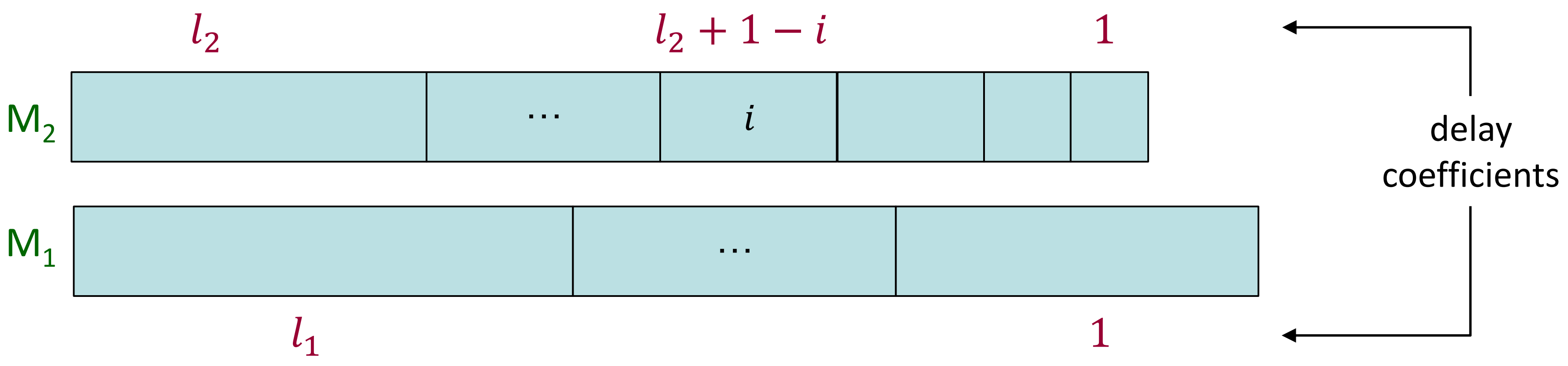
An optimal algorithm for $P|\pi_{LPT}|\sum_j C_j$

Consider the i -th job on machine 2. This job gets a coefficient of $l_2 + 1 - i$.
The shortest possible job that can get this coefficient is job $l_1 + i$.
Consider a job $i \le l_1$. The minimal coefficient job i can get is $l_1 + 1 - i$ (for example, in every schedule, the longest job, has coefficient at least l_1).



An optimal algorithm for $P|\pi_{\text{LPT}}|\sum_j C_j$

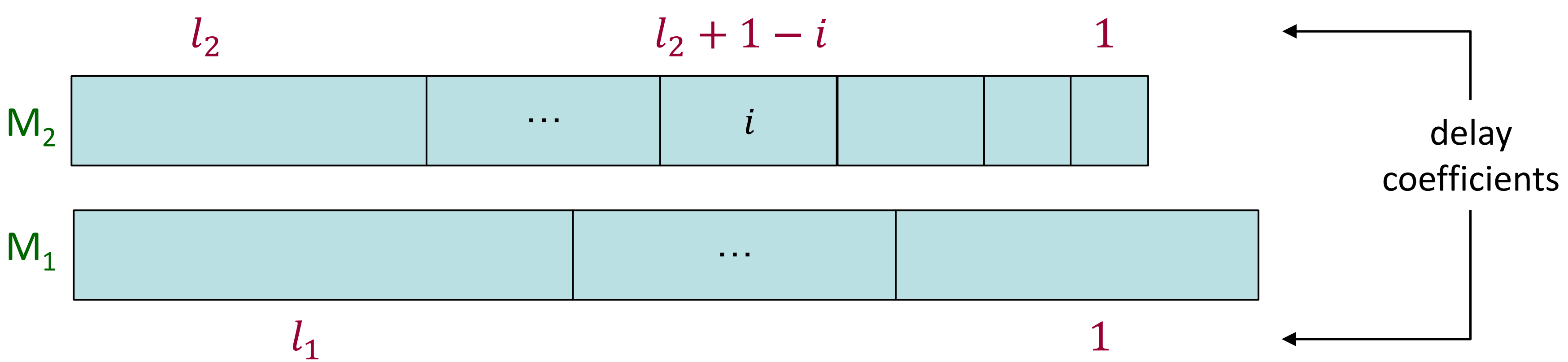
When jobs $j = 1, \ldots, l_1$ are on M_1 and jobs $j = l_1 + 1, \ldots, l_2$ are on M_2, every coefficient (on M_2) is matched with the shortest job that can get this coefficient, and every job (on M_1) is matched with the minimal coefficient it can get. ∎



An optimal algorithm for P$|\pi_{\mathrm{LPT}}|\sum_j C_j$

Theorem: P$|\pi_{\mathrm{LPT}}|\sum_j C_j$ is polynomial time solvable.

Proof: A dynamic programming based on the above claim



On the other hand:

With unrelated machines, the problem is hard and hard to approximate:

Theorem: $\mathrm{R}|\pi_{\mathrm{LPT}}|\sum_j C_j$ is APX-hard

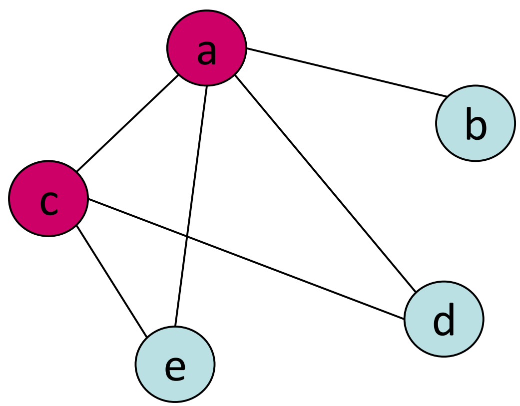
R | π_{LPT} | $\sum_j C_j$ is APX-hard

Theorem: R | π_{LPT} | $\sum_j C_j$ is APX-hard

Proof: (for now, NP-hardness only)

Reduction from vertex-cover

Given a graph G and an integer k, does G have a VC of size k?



VC of size 2.

For every edge, at least one endpoint is in the VC

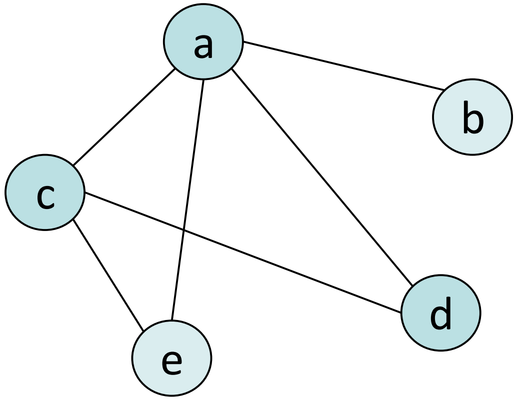
$R|\pi_{LPT}|\sum_j C_j$ is APX-hard

Given G=(V,E) and k, construct an instance for $R|\pi_{LPT}|\sum_j C_j$:
|V| machines, where M_i corresponds to node $i \in V$.
The set of jobs consists of two sets D and A.
D includes |V|-k dummy jobs. $\forall i, d, p_{i,d} = 1$
A includes |E| jobs, each corresponding to an edge $e \in E$.

$$p_{i,(u,v)} = \begin{cases} 0 & i = u \text{ or } i = v \\ 1 & otherwise \end{cases}$$

i is an endpoint of (u,v)

M={a,b,c,d,e}
J= D∪A
D={d_1,d_2,d_3}
A={(ab),(ad),(ae),…}

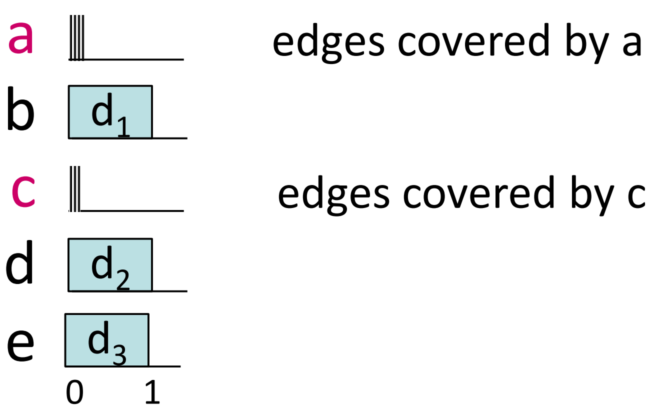


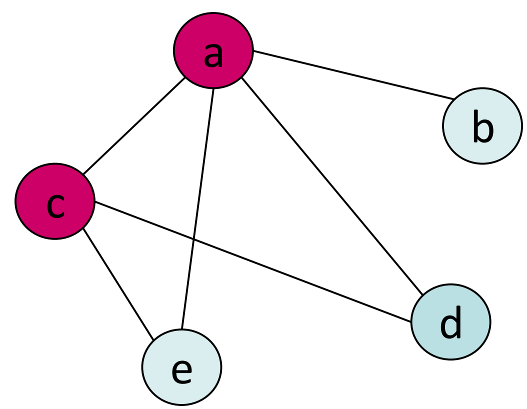
VC of size 2?

$R|\pi_{\text{LPT}}|\sum_j C_j$ is APX-hard

π_{LPT} implies that if a job (edge) is assigned on a machine corresponding to one of its endpoint then it is processed after any dummy job assigned to this machine.

A VC of size k $\Leftrightarrow$ a schedule with $\sum_j C_j = |V| - k$





VC of size 2

Every dummy job goes to a different machine.
All A-jobs have C_j=0.

$R|\pi_{\text{LPT}}|\sum_j C_j$ is APX-hard

Hardness proof for APX-hardness a bit more technical.
The reduction is from Max-k-VC of a bounded degree graph.
Given G, k, where max-degree(G) = Δ, find $U \subseteq V$, $|U|=k$, such that the number of edges adjacent to vertices in U is maximal.

Summary and open problems

- The introduction of machine-dependent priority lists opens a new world of optimization problems.
- Challenging analysis as a game as well as an optimization problem.
- General instances: no guaranteed NE, hard to approx.
- Some important classes behave nicely.

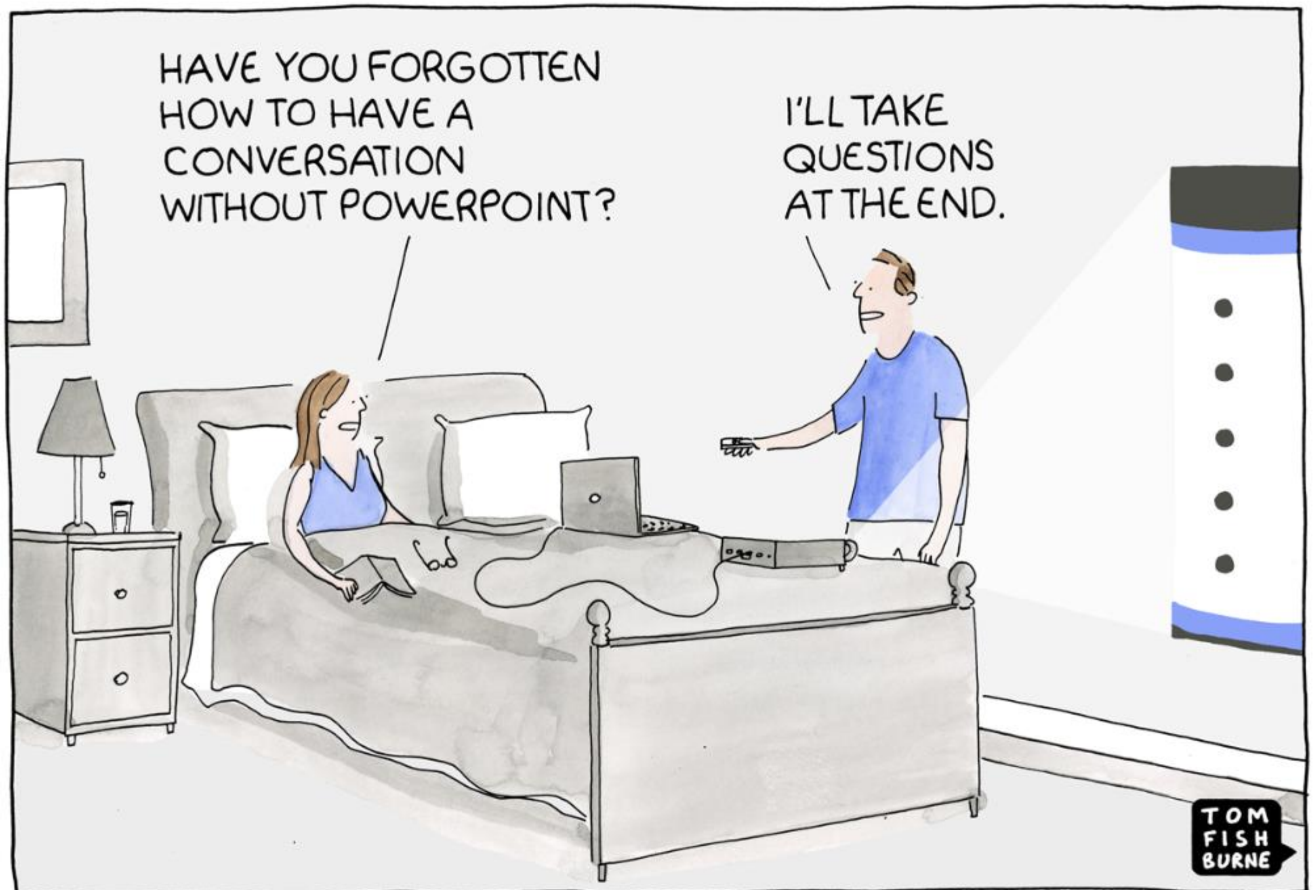
Summary and open problems

To do list:

- Complexity status of $P|\pi|\sum_j C_j$ (QPTAS but no hardness proof)
- Identify additional tractable/stable instances
- Approximation algorithms
- Priority-list can be viewed as a special case of machines-based precedence constraints (precedence constraints given by a chain). Study the general $P|\text{machine-based prec}|\sum_j C_j$
- Analyze instances with due-dates and lateness-related obj.

Assume a global priority list.
What is the minimal number of machines required to complete all jobs on time?

Questions?



© marketoonist.com