Online Vector Scheduling

Debmalya Panigrahi

Duke University

slides

Work done with:

Sungjin Im
(UC Merced)

Nat Kell
(Duke)

Janardhan Kulkarni
(MSR → UMN)

Maryam Shadloo
(UC Merced)

Online Load Balancing

[Graham ’66]

Job: 1 2 3 4 5 6 7 8

Load: 1.1 1.3 1.2 0.8 2 0.9 1.2 1.2 . . .

(processing time)

Machine 1

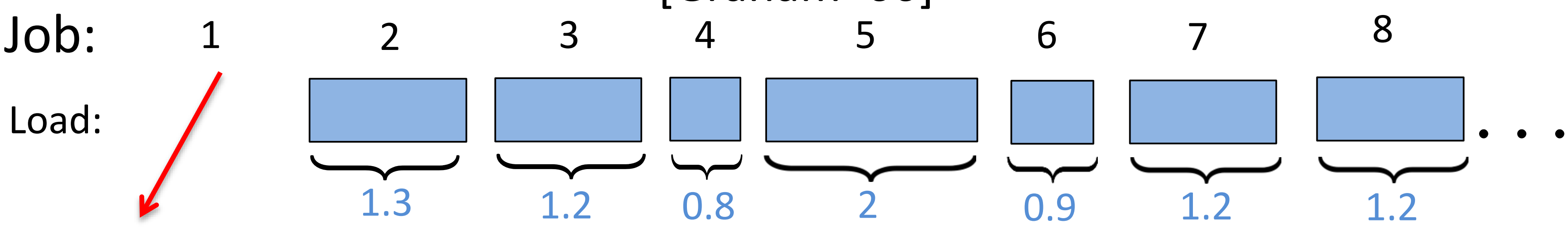
Machine 2

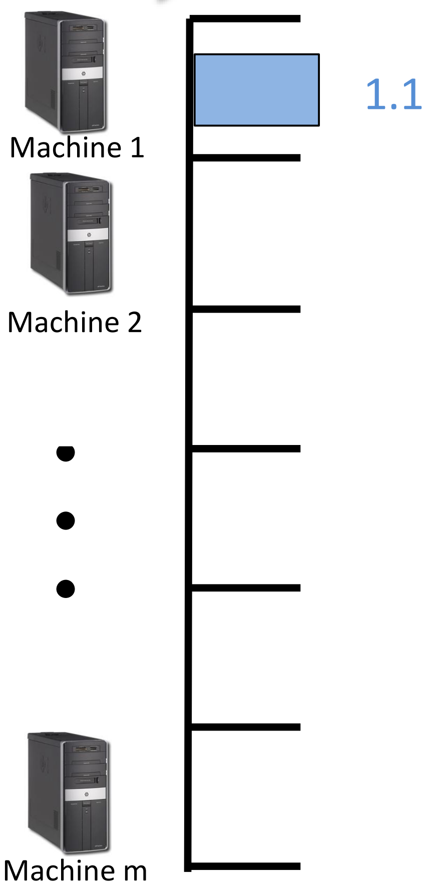
•
•
•

Machine m

Online Load Balancing

[Graham ’66]

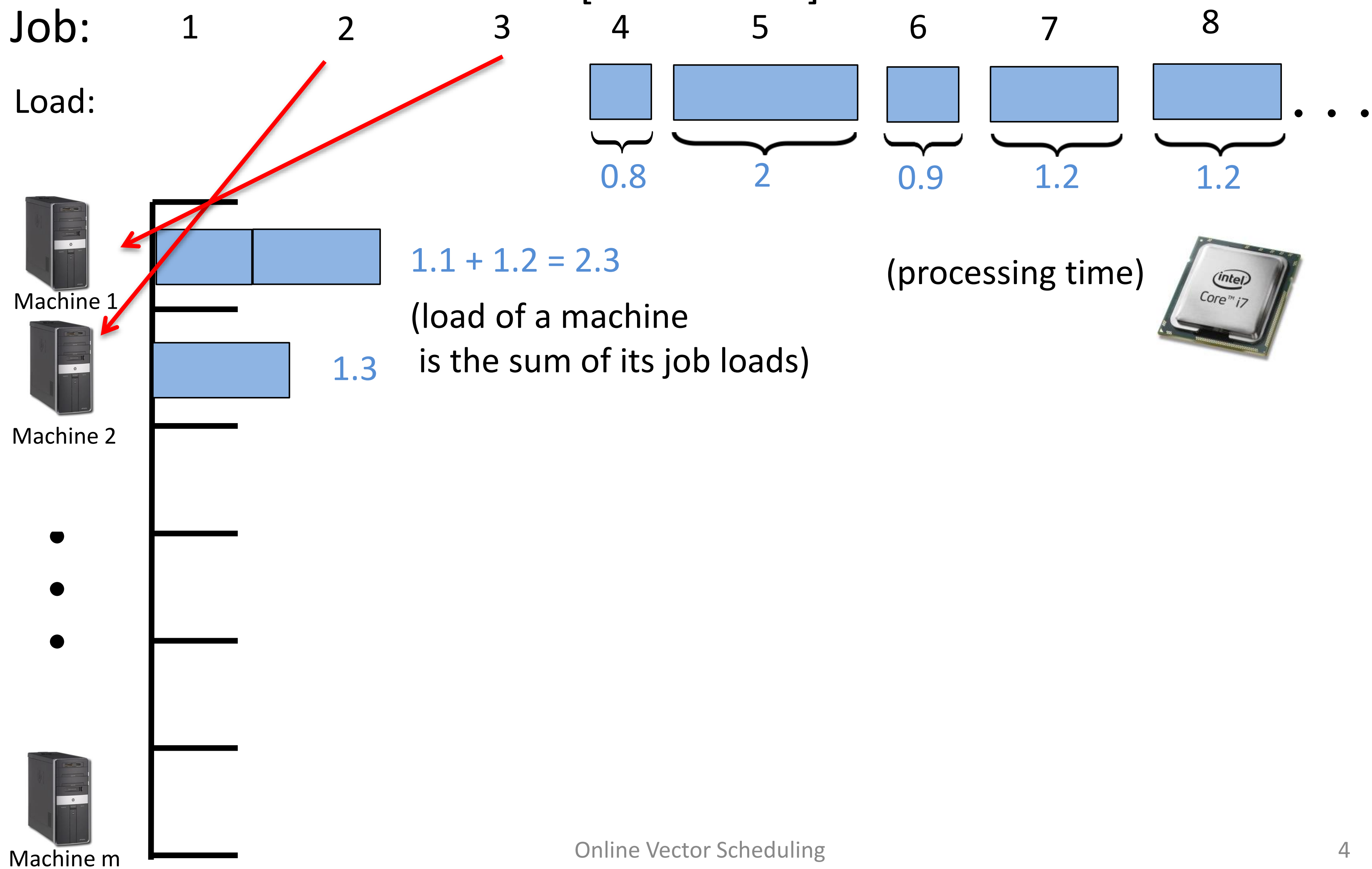




(processing time)

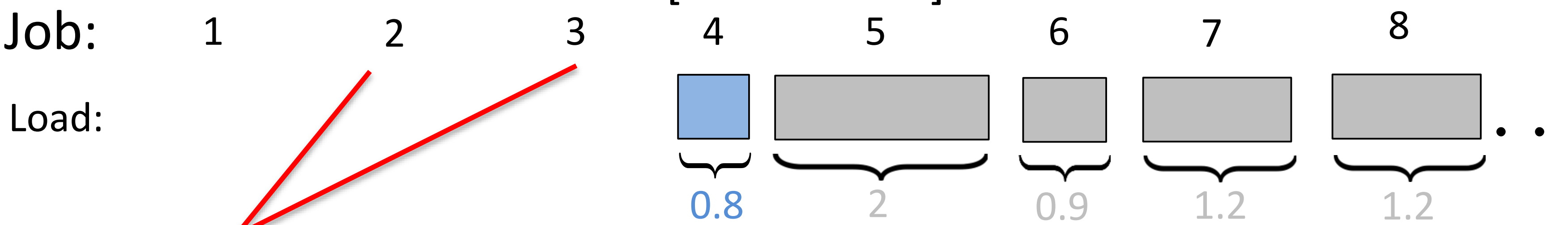
Online Load Balancing

[Graham '66]



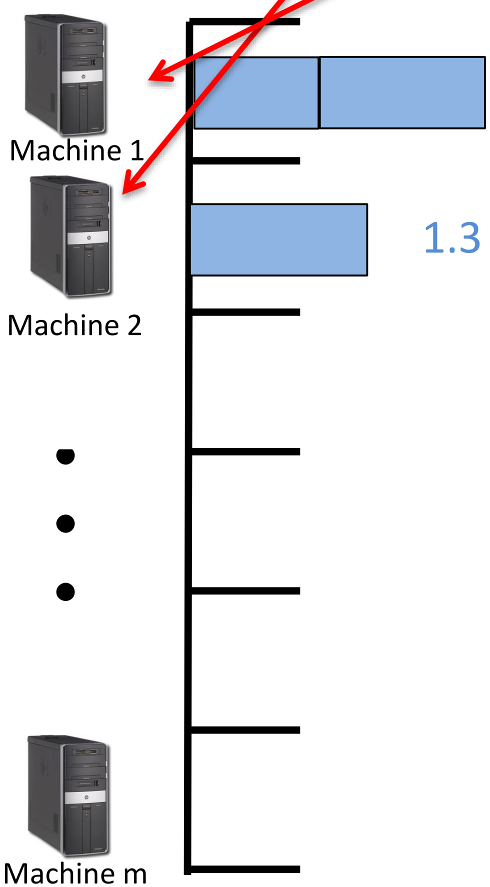
Online Load Balancing

[Graham ’66]



Online problem: cannot see future jobs.

(processing time)



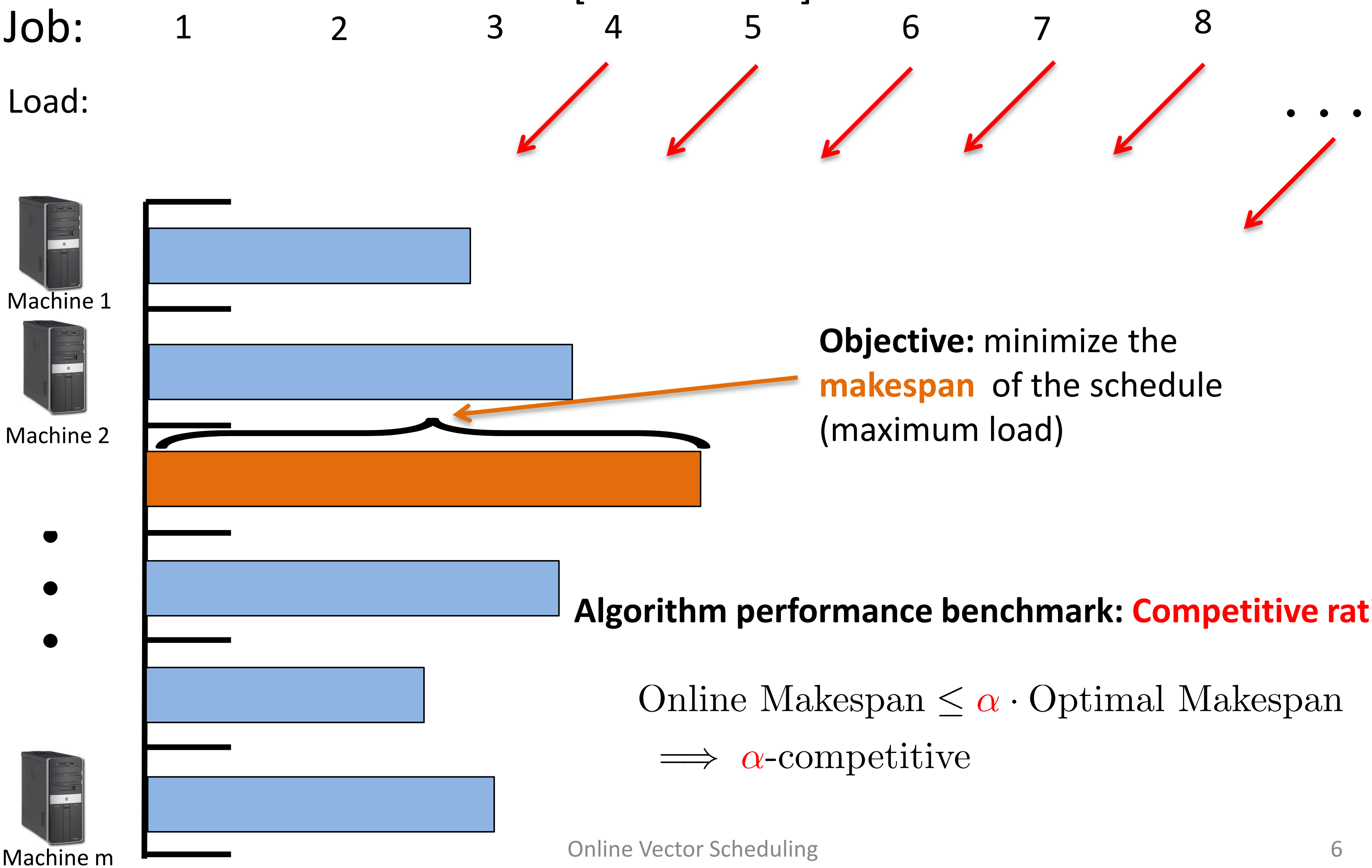
1.1 + 1.2 = 2.3

(load of a machine
is the sum of its job loads)

1.3

Online Load Balancing

[Graham '66]



Objective: minimize the **makespan** of the schedule (maximum load)

Algorithm performance benchmark: Competitive ratio

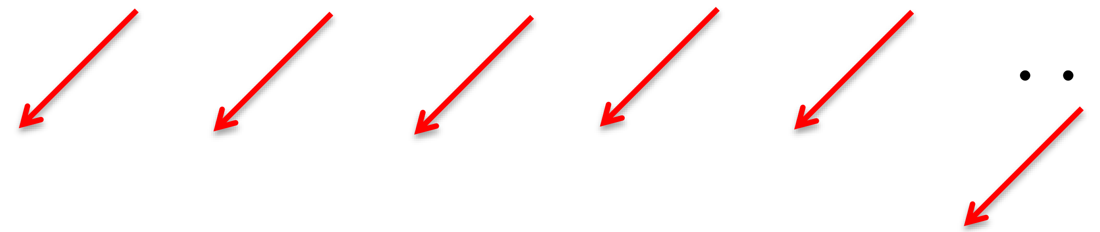
$$\text{Online Makespan} \leq \alpha \cdot \text{Optimal Makespan}$$
$$\implies \alpha\text{-competitive}$$

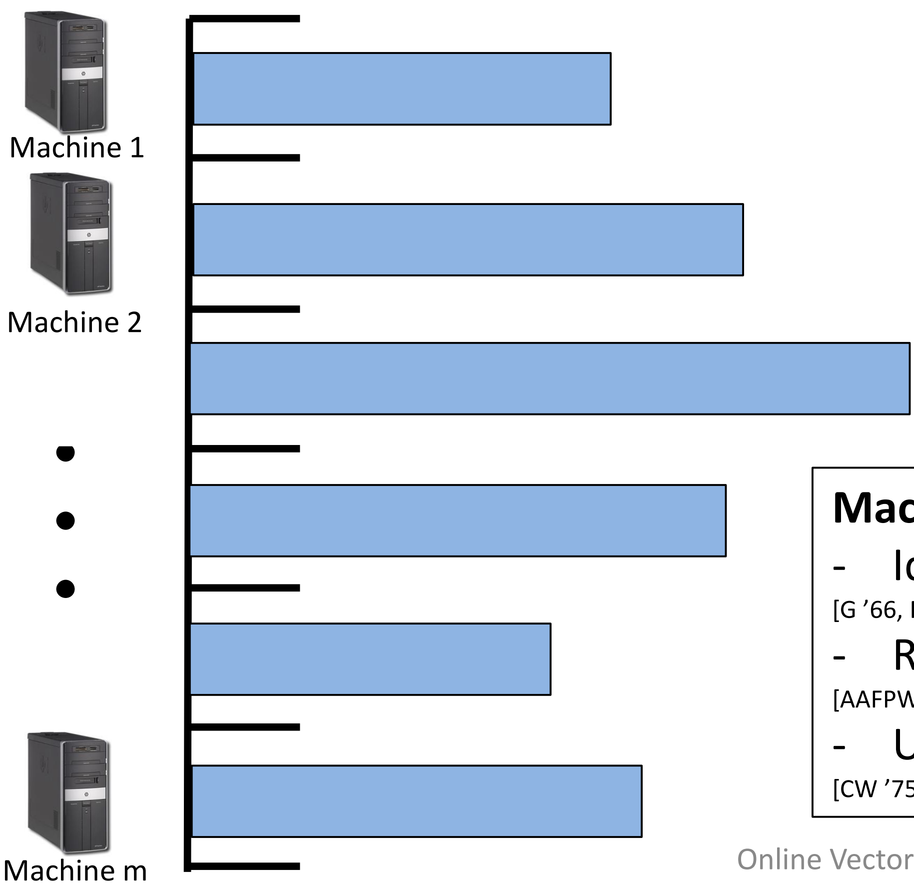
Online Load Balancing

[Variants]

Job: 1 2 3 4 5 6 7 8

Load:





Objectives: minimize the **p-norm** of the machine loads (makespan is the ∞-norm)

[CW ’75, CC ’76, AAGKKV ’95, AAS ’01, C ’08, CFKKM ’11]

Machine models:

- Identical machines (load = p_j)

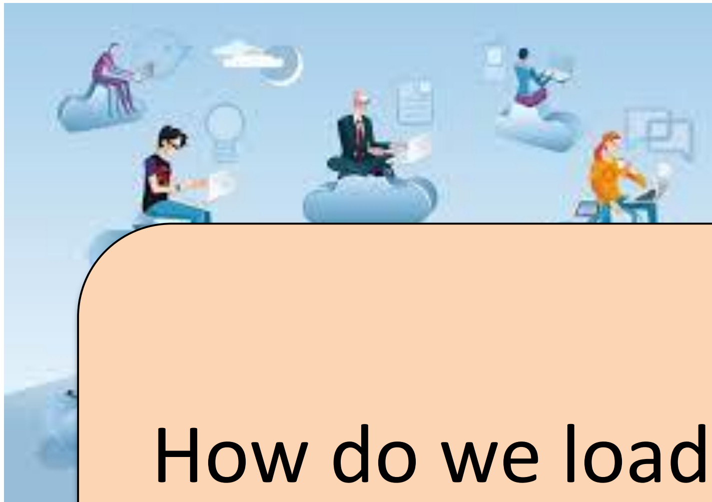
[G ’66, FKT ’89, BKR ’94, BFKV ’95, KPT ’96, A ’99, FW ’00, GRTW ’00, R ’01, AAS ’01]

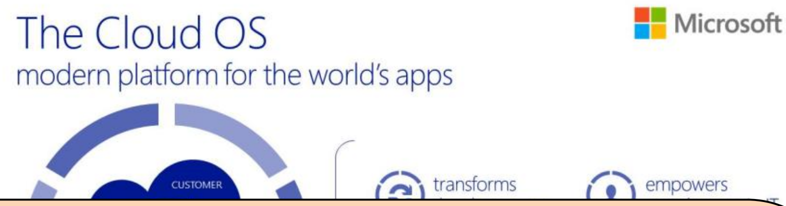
- Related machines (load = p_j / s_i)

[AAFPW ’97, BCK ’00]

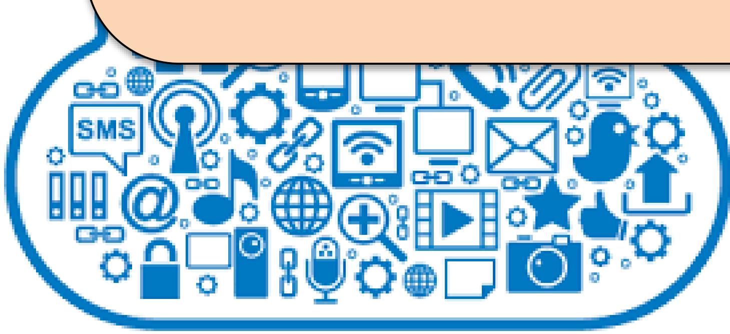
- Unrelated machines (load = p_{ij})

[CW ’75, CC ’76, AAGKKV ’95, AAFPW ’97, C ’08, ANR ’95, CFKKM ’11]

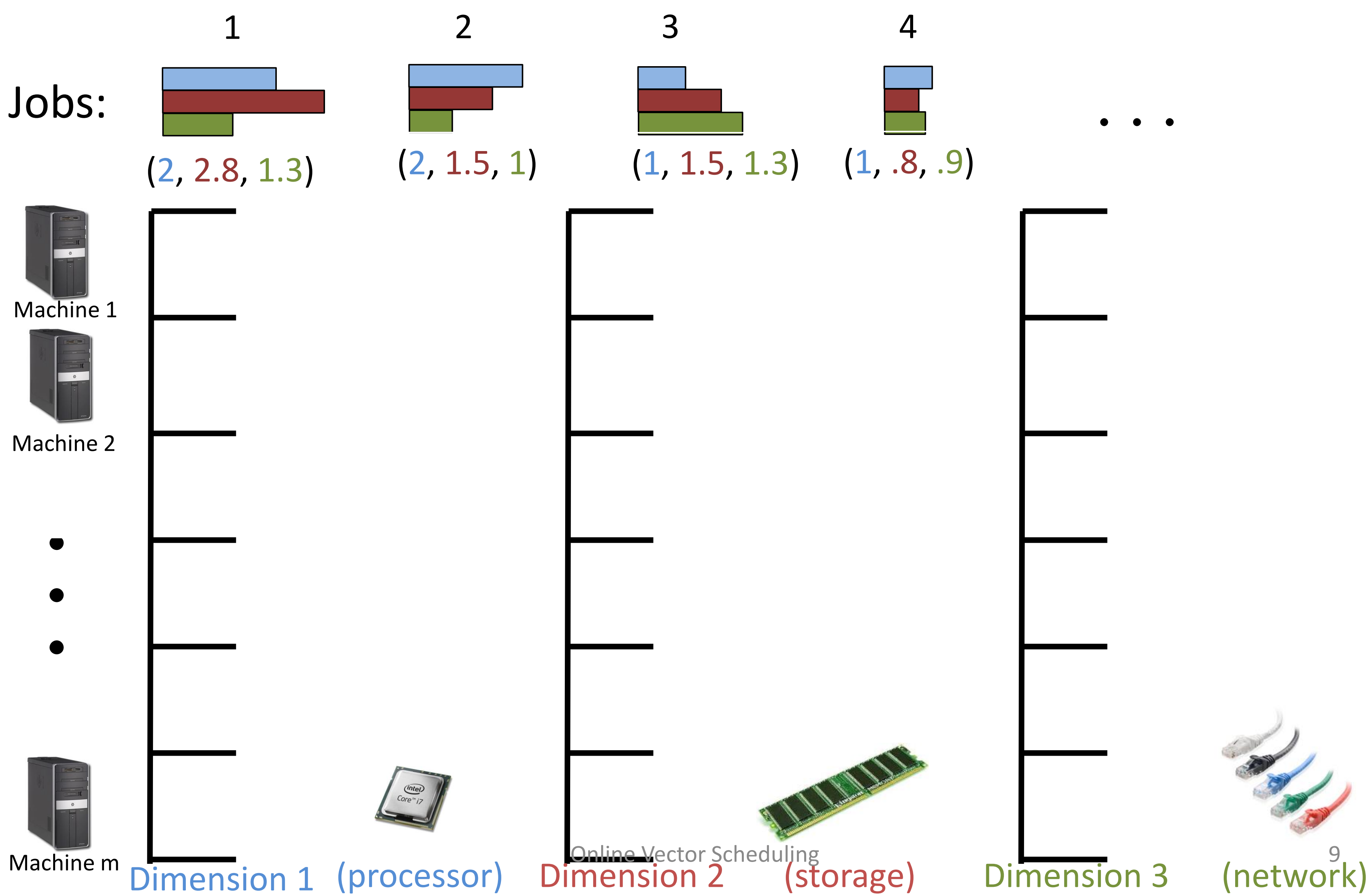




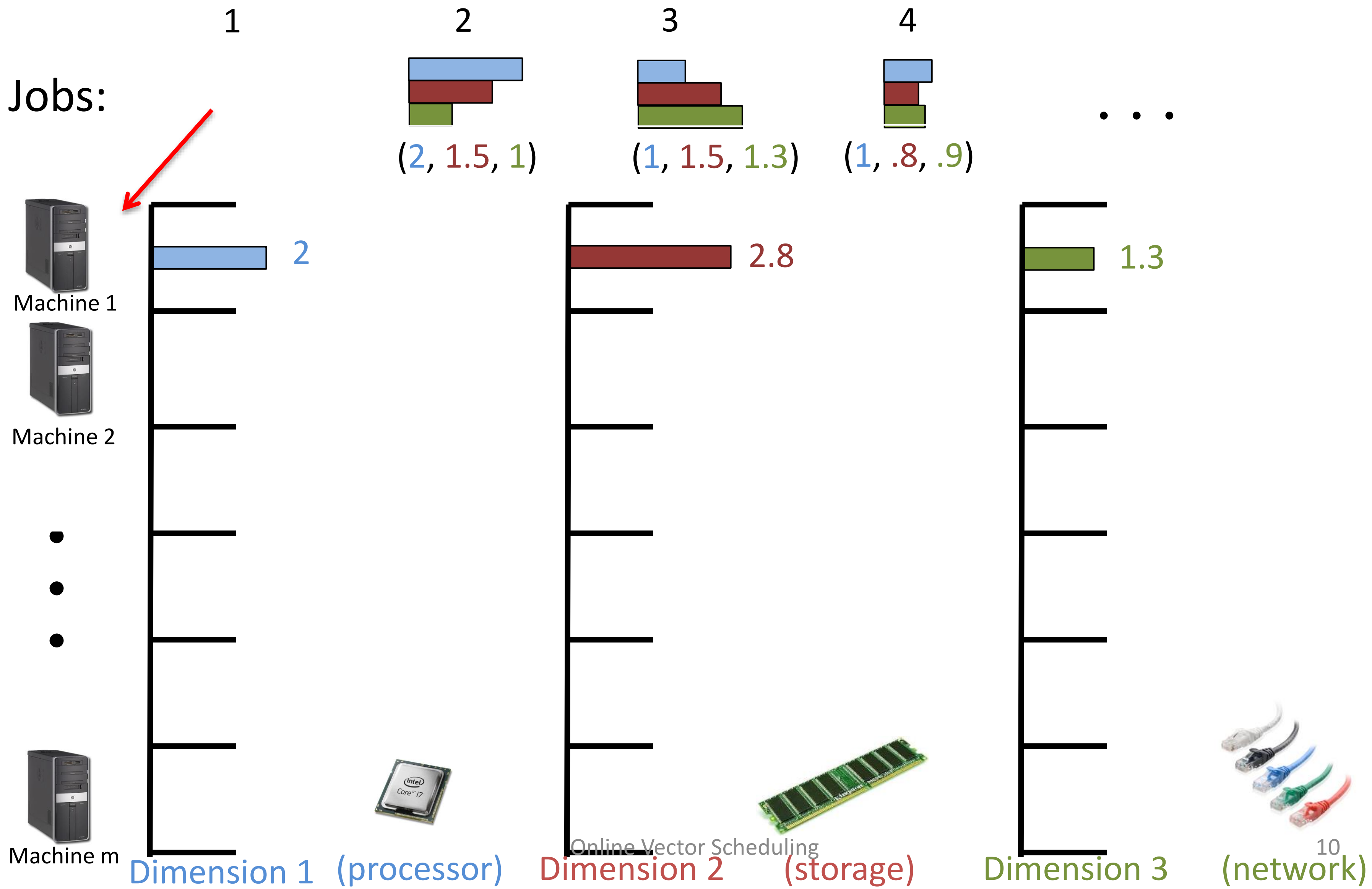
How do we load balance simultaneously on multiple resources (e.g., in data centers)?



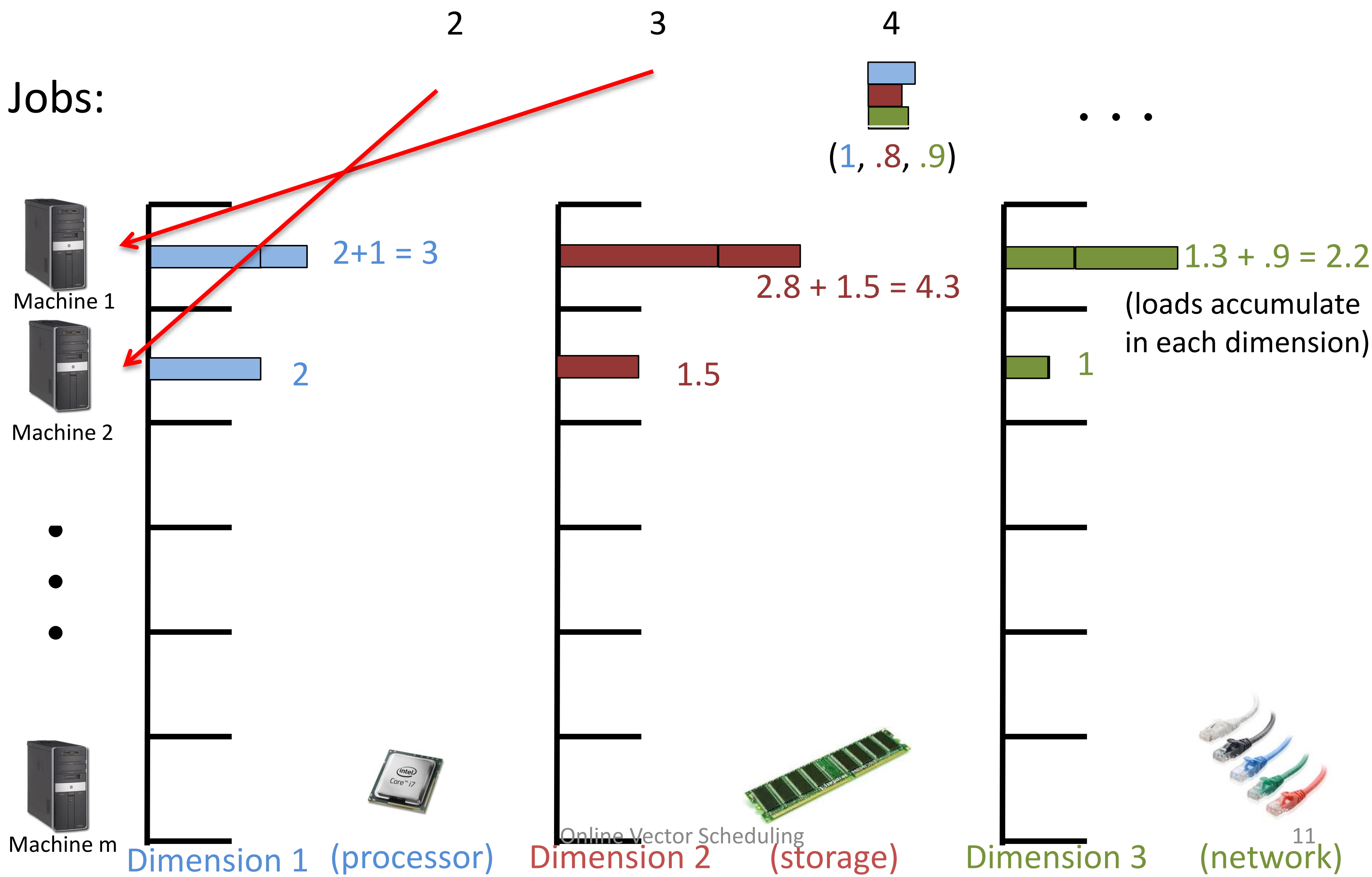
Online Vector Scheduling



Online Vector Scheduling

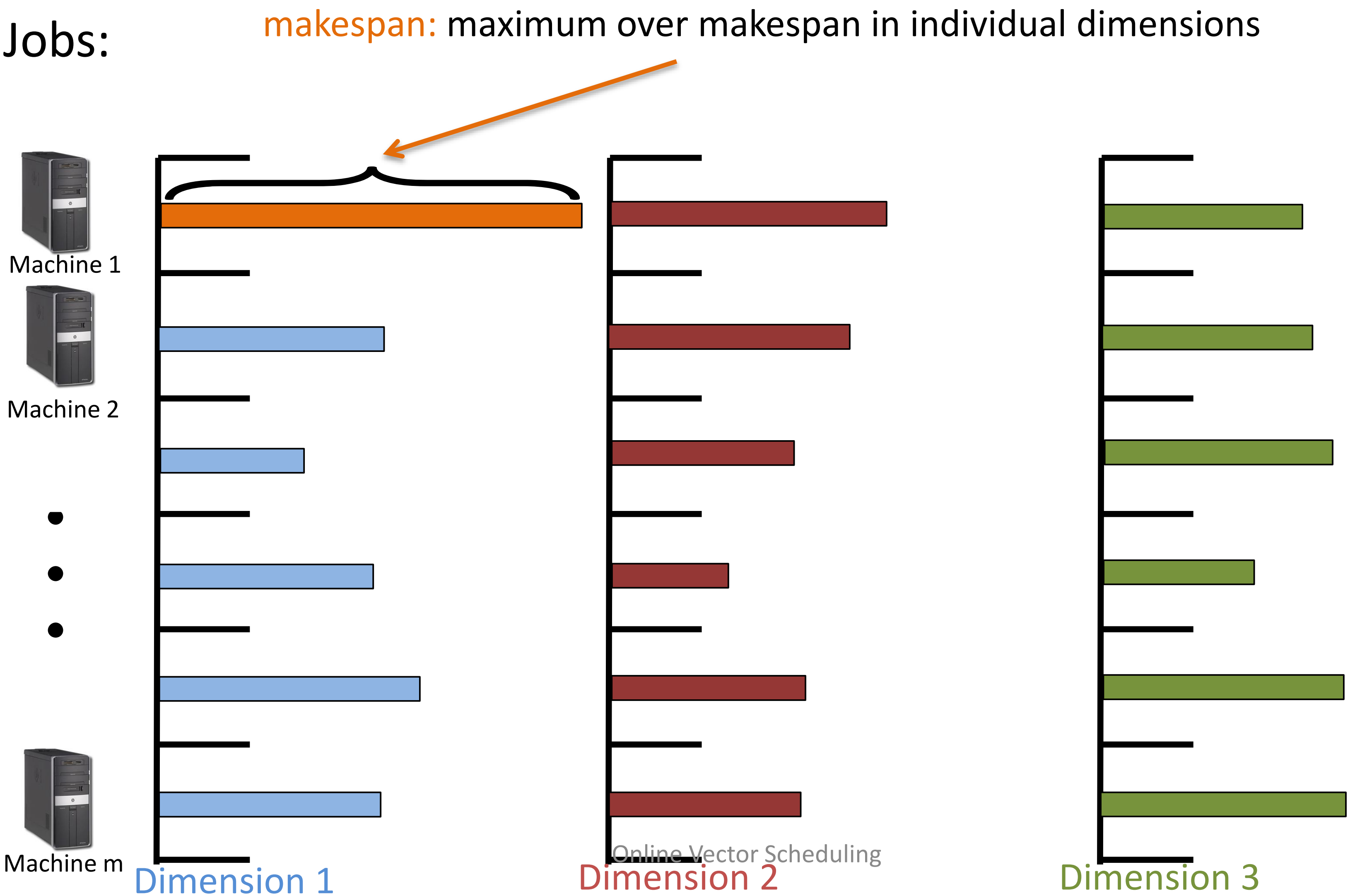


Online Vector Scheduling



Online Vector Scheduling

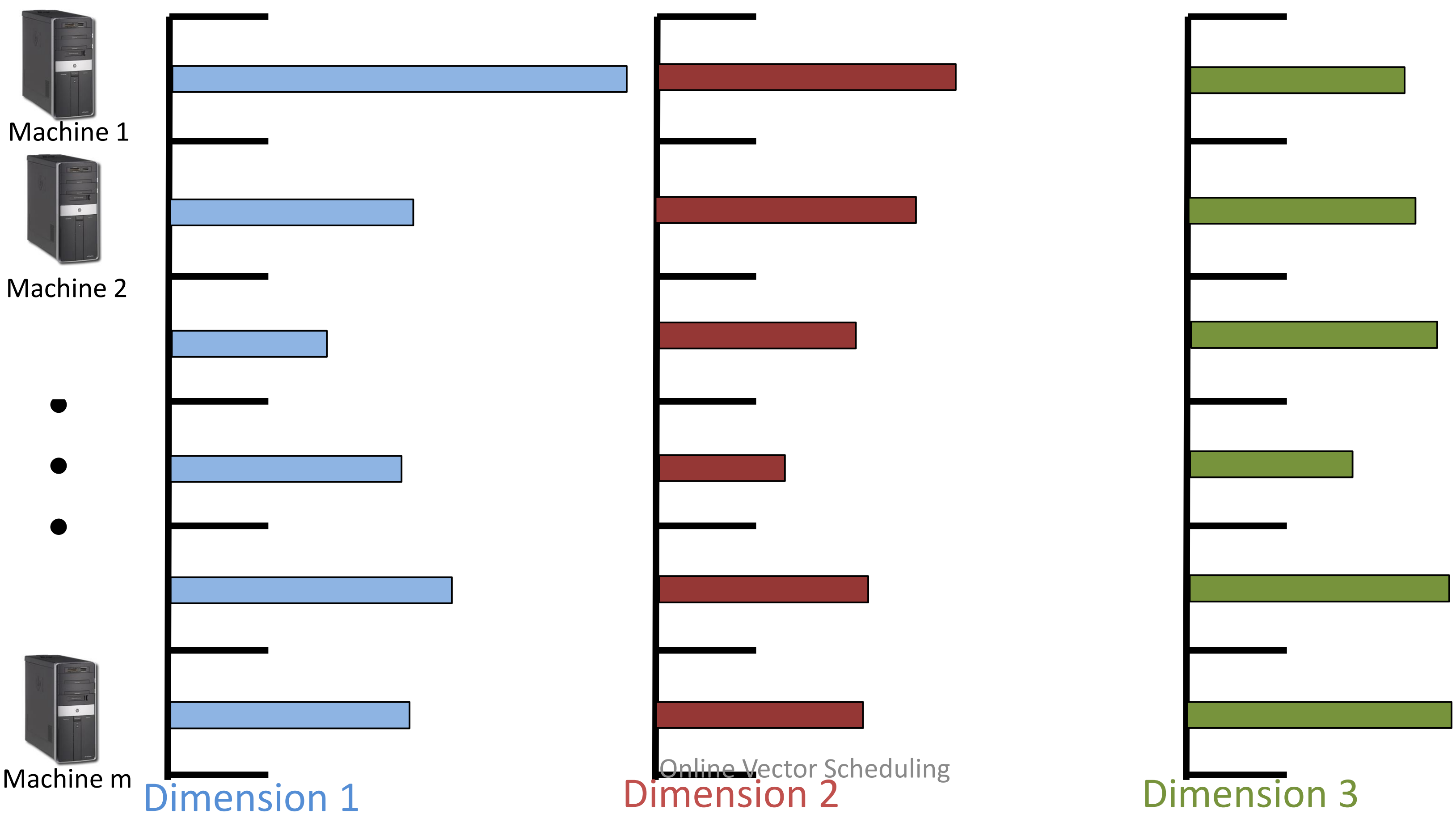
Jobs:



Online Vector Scheduling

Jobs:

p-norms: maximum over p-norms in individual dimensions



Summary of Results

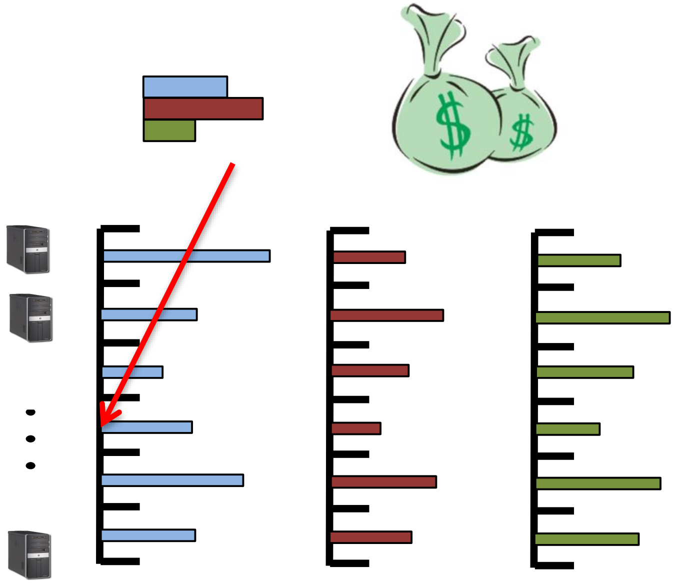
	Makespan minimization	p-norm minimization	
Identical machines	O(log d) [Azar *et al* '13, Meyerson *et al* '14] Our result: **Θ(log d/log log d)**	Our result: **$\Theta((\log d/\log\log d)^{1-1/p})$**	(Im-Kulkarni-Kell-P. FOCS '15)
Unrelated machines (machine dependent loads)	O(log d + log m) [Meyerson *et al* '14] Our result: **Θ(log d + log m)**	Our result: **Θ(log d + p)**	
Related machines (non-uniform machine speeds)	Later…	Later…	(Im-Kell-P.-Shadloo '17)

Summary of Results

	Makespan minimization	p-norm minimization	
Identical machines	O(log d) [Azar *et al* ’13, Meyerson *et al* ’14] Our result: **Θ(log d/log log d)**	Our result: **$\Theta((\log d/\log\log d)^{1-1/p})$**	(Im-Kulkarni-Kell-P. FOCS ’15)
Unrelated machines (machine dependent loads)	O(log d + log m) [Meyerson *et al* ’14] Our result: **Θ(log d + log m)**	Our result: **Θ(log d + p)**	
Related machines (non-uniform machine speeds)	Later…	Later…	(Im-Kell-P.-Shadloo ’17)

Identical machines algorithm: First attempt

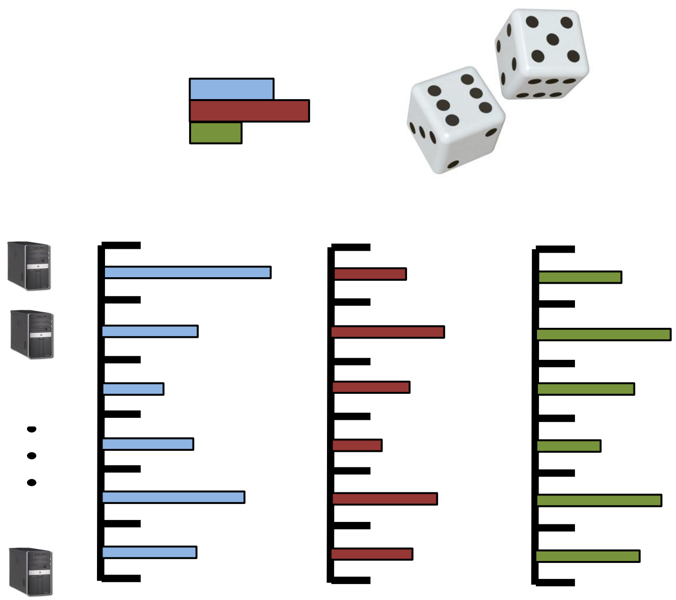
Greedy assignment
(minimize maximum load
across all machines and dimensions)



unbalanced loads on dimensions
…can be as bad as poly(d)-competitive

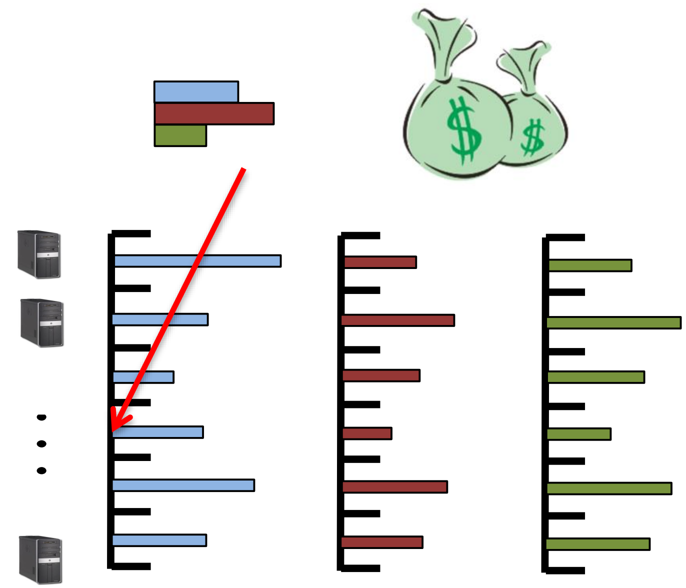
Identical machines algorithm: First attempt

Random Assignment
(assignment uniformly at random)



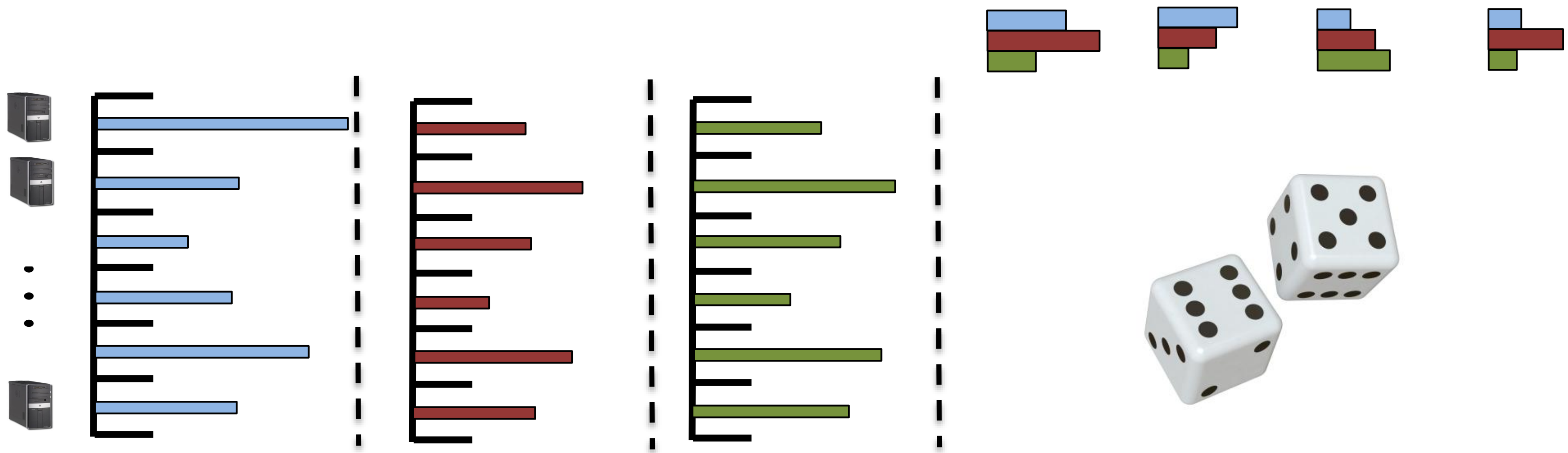
Chernoff bounds:
O(log(dm))-competitive
(optimal for unrelated machines)

Greedy assignment
(minimize maximum load
across all machines and dimensions)



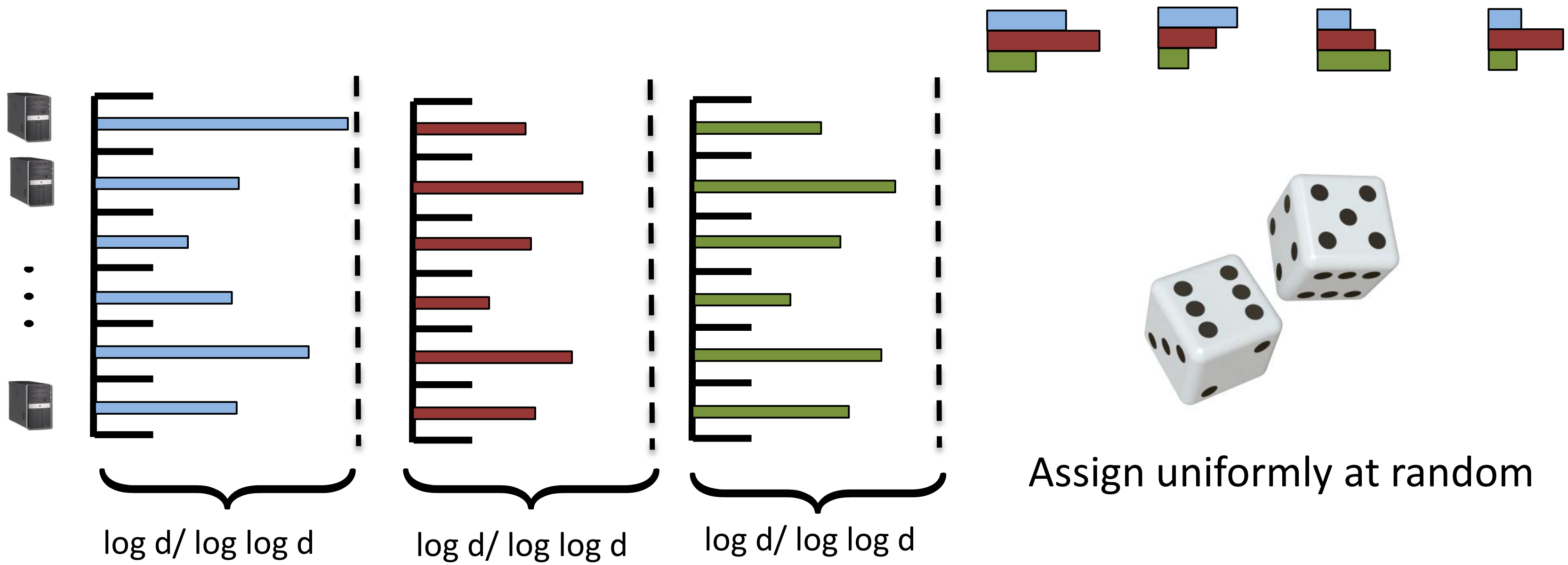
unbalanced loads on dimensions
...can be as bad as poly(d)-competitive

Algorithm: Random and Greedy



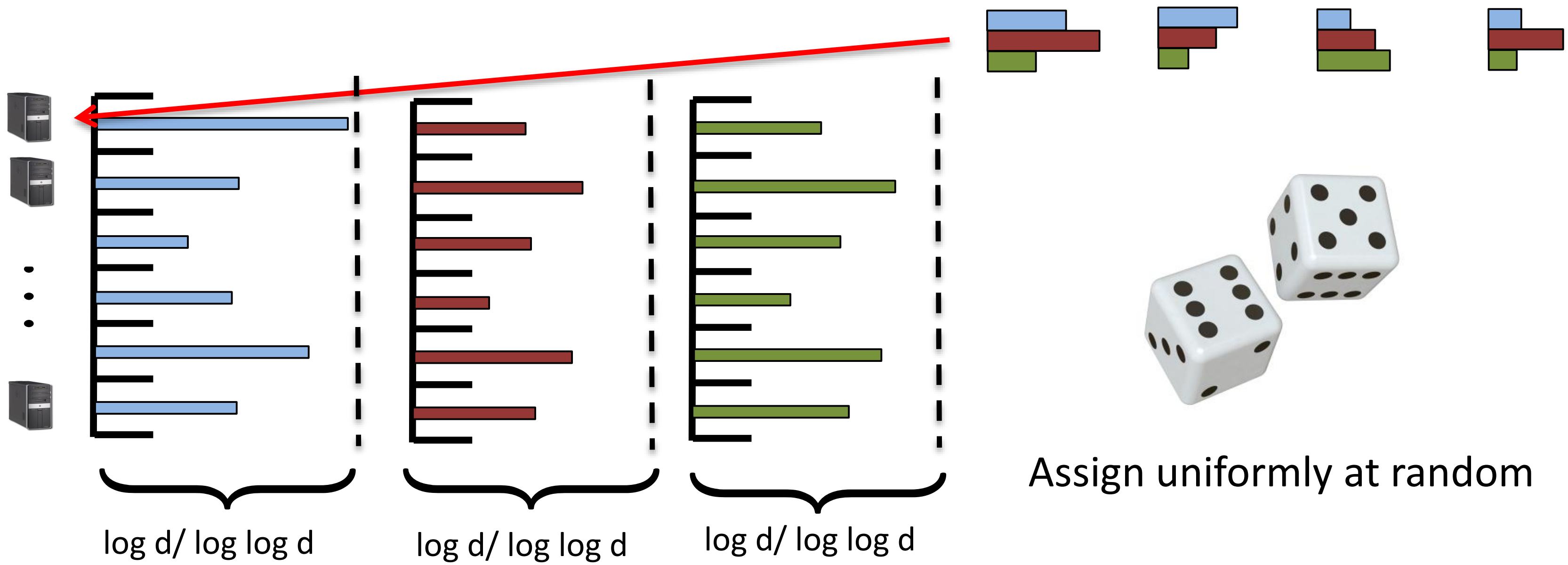
Assign uniformly at random

Algorithm: Random and Greedy

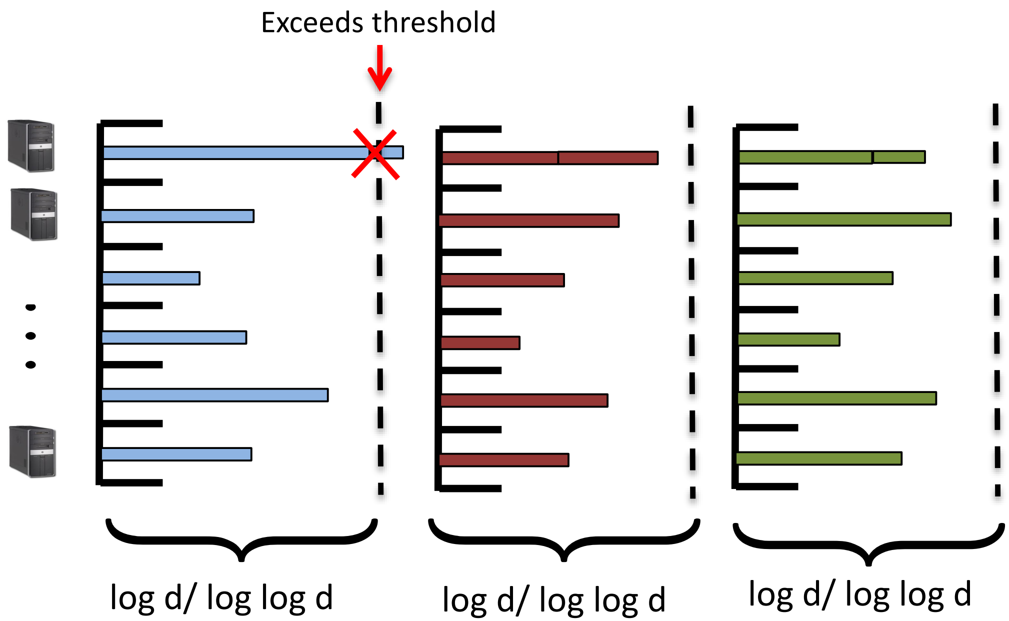


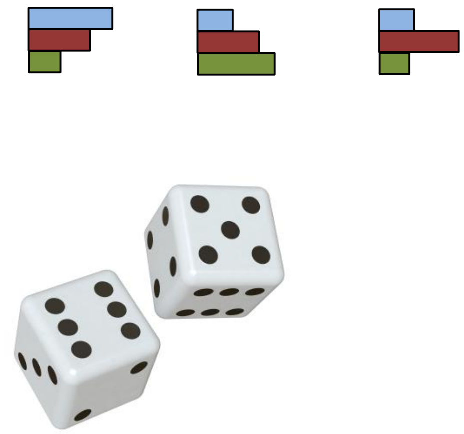
Assign uniformly at random

Algorithm: Random and Greedy



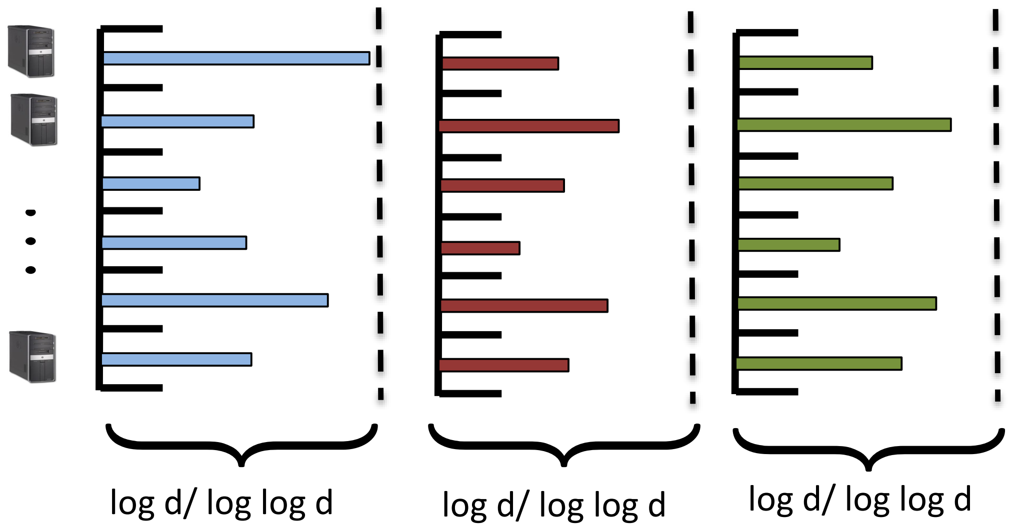
Algorithm: Random and Greedy

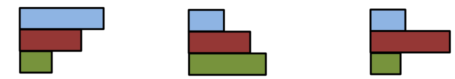




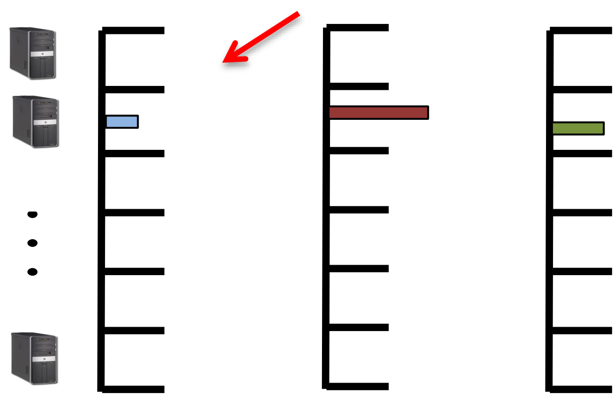
Assign uniformly at random

Algorithm: Random and Greedy



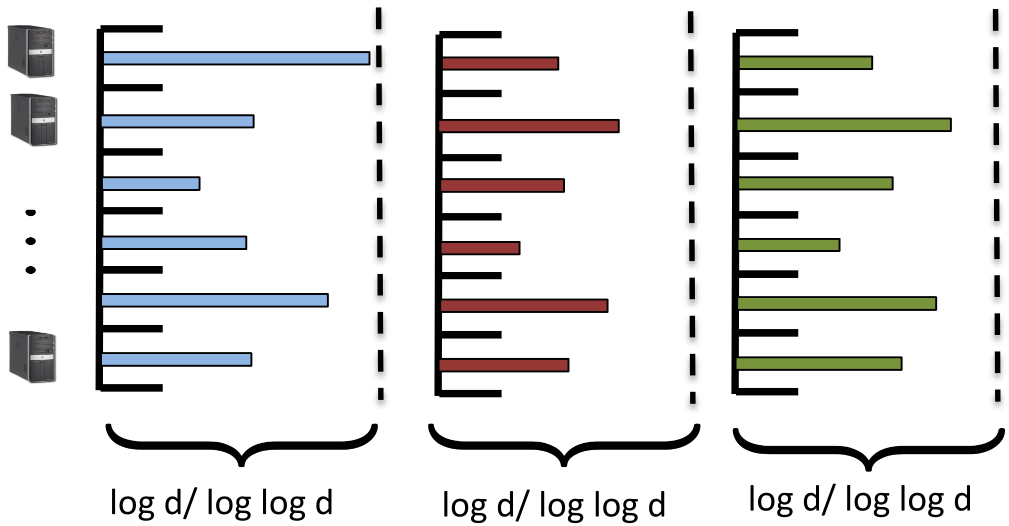


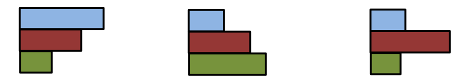
Assign uniformly at random



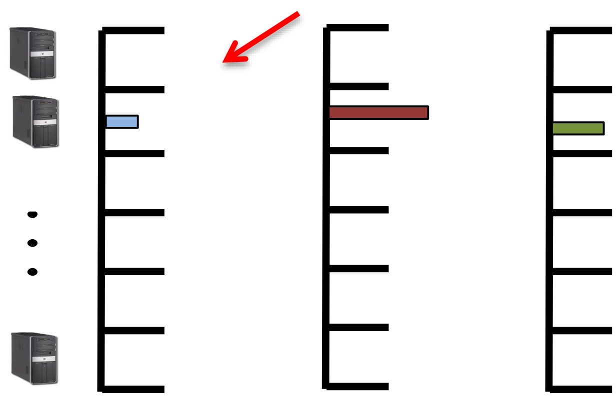
Greedy schedule
(minimize max over
all machines and dimensions)

Algorithm: Random and Greedy





Assign uniformly at random



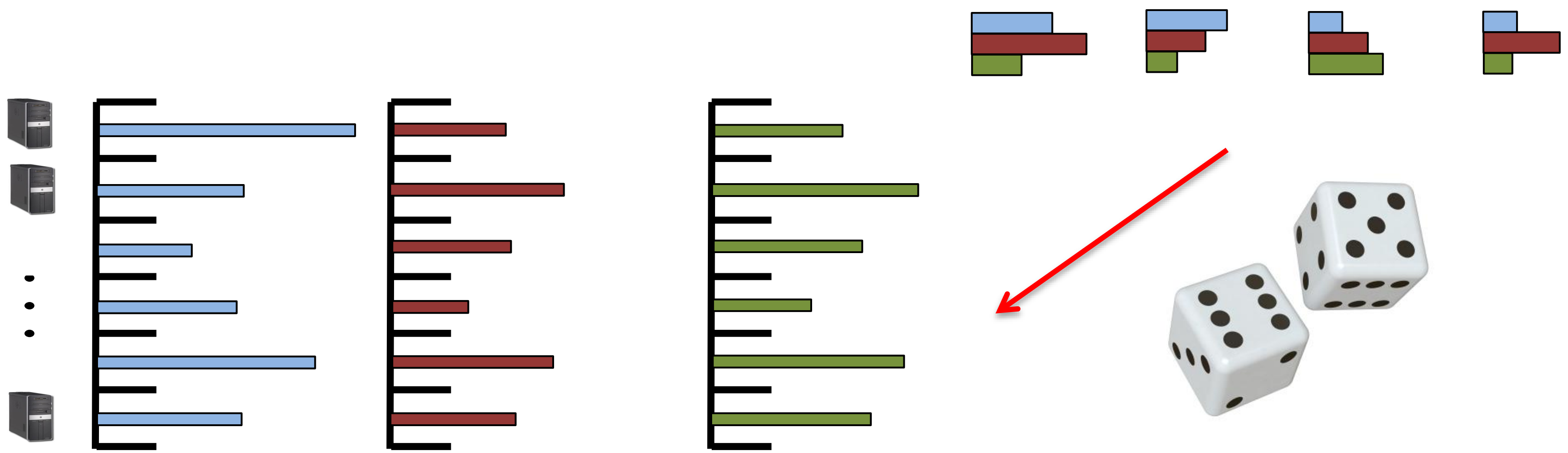
Greedy schedule
(minimize max over
all machines and dimensions)

E[greedy volume] < volume/poly(d)
(Chernoff bounds)

$\Longrightarrow$

E[greedy makespan] = O(1)

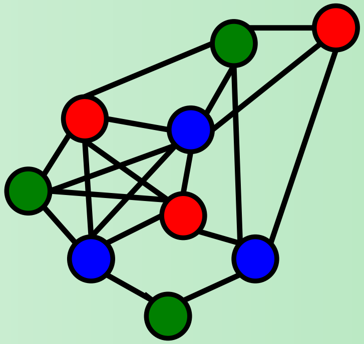
Algorithm: Random and Greedy



Competitive ratio: O(log d/log log d)

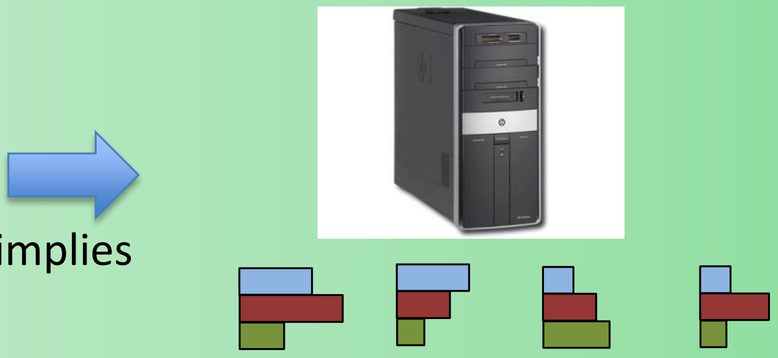
Best we can do?

Turns out yes:



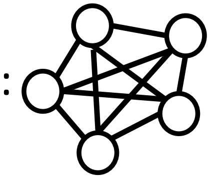
Coloring lower bound

implies



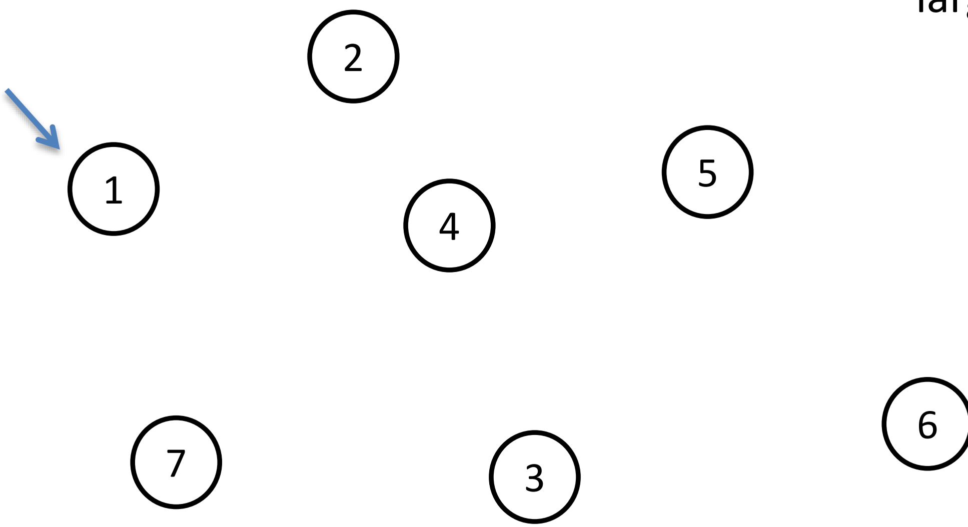
Ω(log d/log log d) lower bound
For vector scheduling

Online Monochromatic Clique

Clique: 

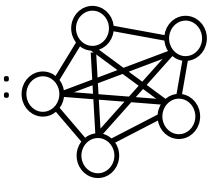
Given *fixed* of t colors: red, blue, and green. (here t = 3)

Objective: minimize the largest monochromatic clique.



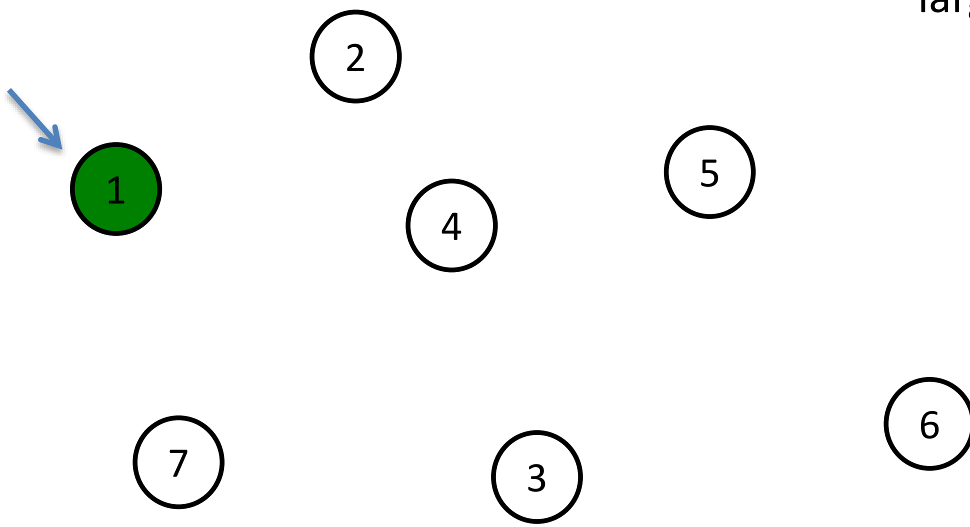
*i*th vertex arrives: online algorithm gets adjacencies with vertices *1, …, i-1*

Online Monochromatic Clique

Clique: 

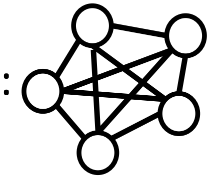
Given *fixed* of t colors: red, blue, and green. (here t = 3)

Objective: minimize the largest monochromatic clique.



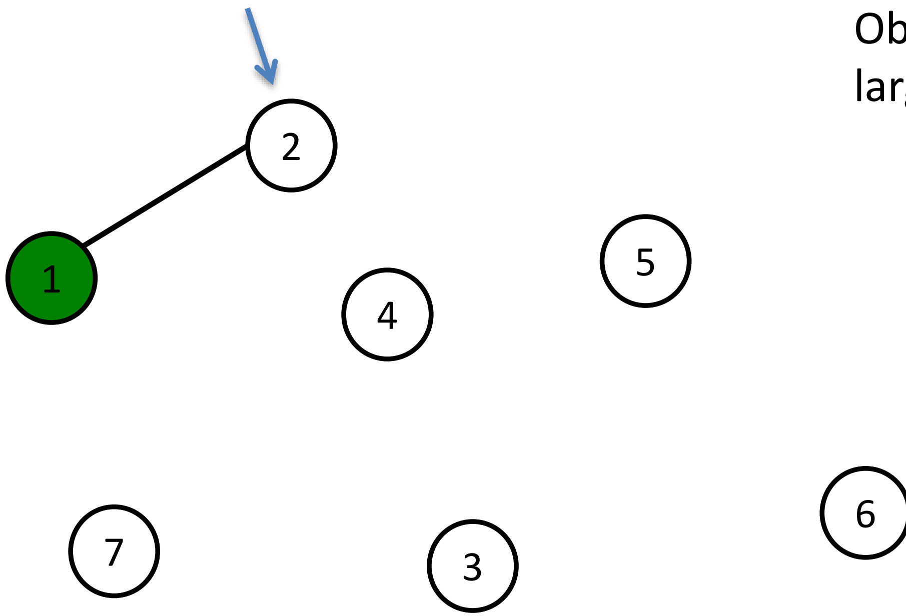
*i*th vertex arrives: online algorithm gets adjacencies with vertices *1, …, i-1*

Online Monochromatic Clique

Clique: 

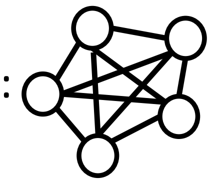
Given *fixed* of t colors: red, blue, and green. (here t = 3)

Objective: minimize the largest monochromatic clique.



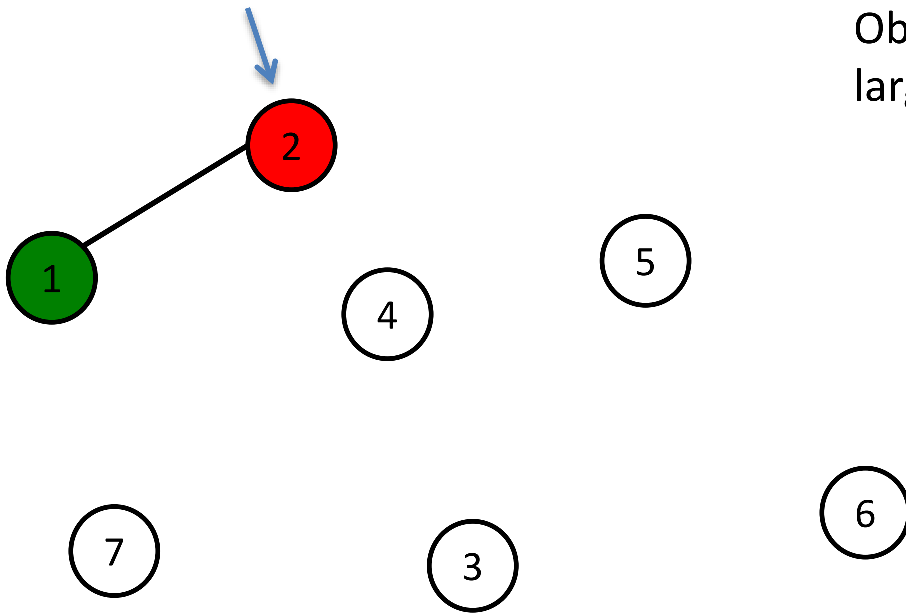
*i*th vertex arrives: online algorithm gets adjacencies with vertices *1, …, i-1*

Online Monochromatic Clique

Clique: 

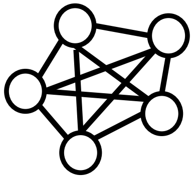
Given *fixed* of t colors: red, blue, and green. (here t = 3)

Objective: minimize the largest monochromatic clique.



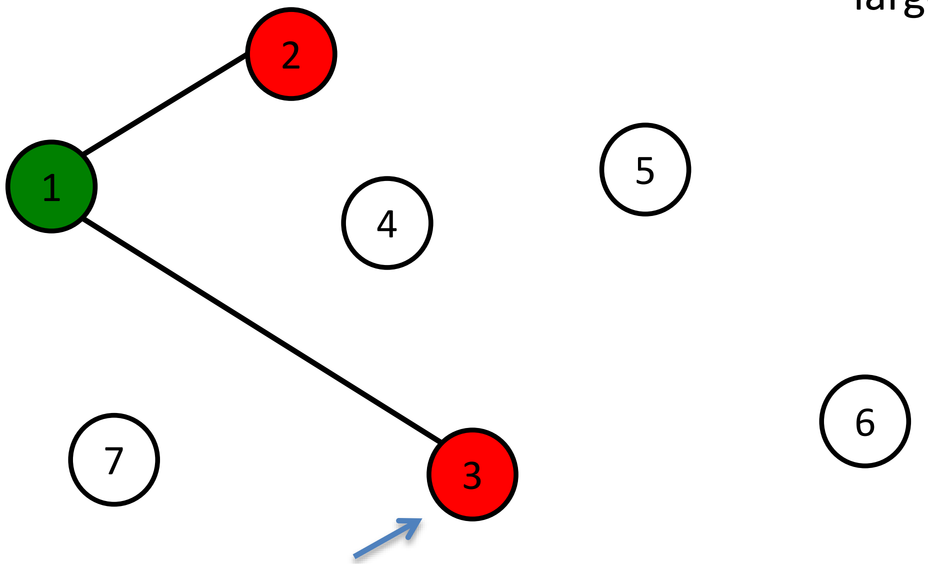
*i*th vertex arrives: online algorithm gets adjacencies with vertices *1, …, i-1*

Online Monochromatic Clique

Clique: 

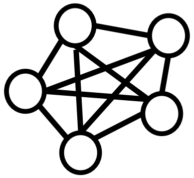
Given *fixed* of t colors: red, blue, and green. (here t = 3)

Objective: minimize the largest monochromatic clique.



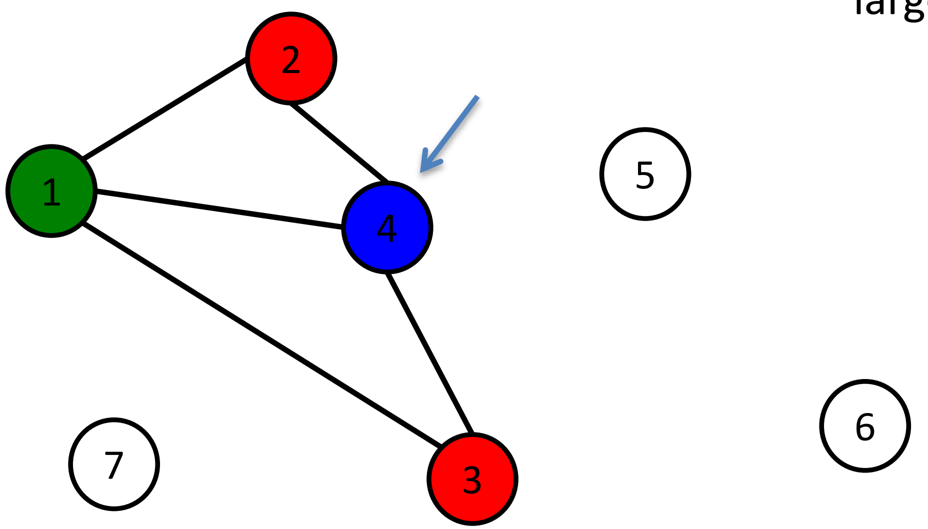
*i*th vertex arrives: online algorithm gets adjacencies with vertices *1, …, i-1*

Online Monochromatic Clique

Clique: 

Given *fixed* of t colors: red, blue, and green. (here t = 3)

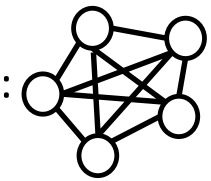
Objective: minimize the largest monochromatic clique.



*i*th vertex arrives: online algorithm gets adjacencies with vertices *1, …, i-1*

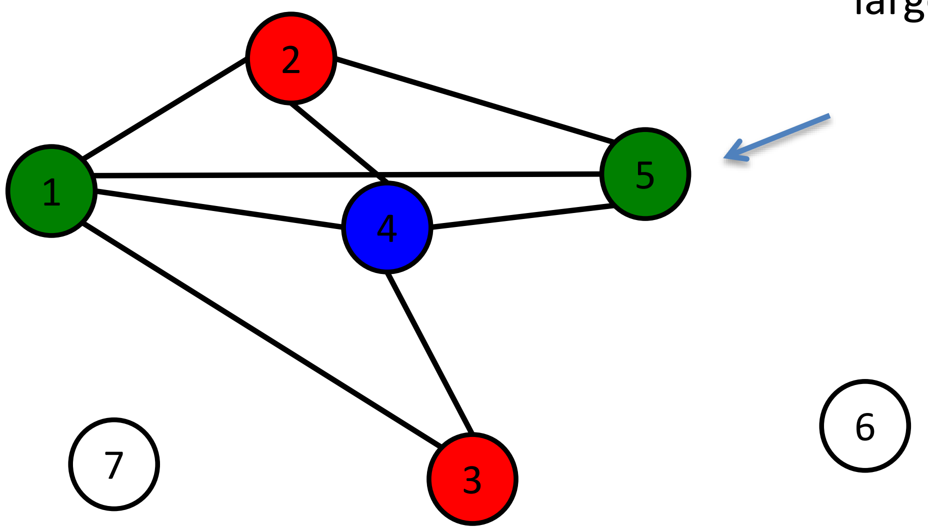
Online Monochromatic Clique

Clique:



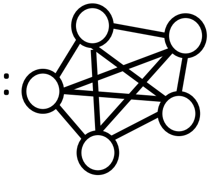
Given *fixed* of t colors: red, blue, and green. (here t = 3)

Objective: minimize the largest monochromatic clique.



*i*th vertex arrives: online algorithm gets adjacencies with vertices *1, …, i-1*

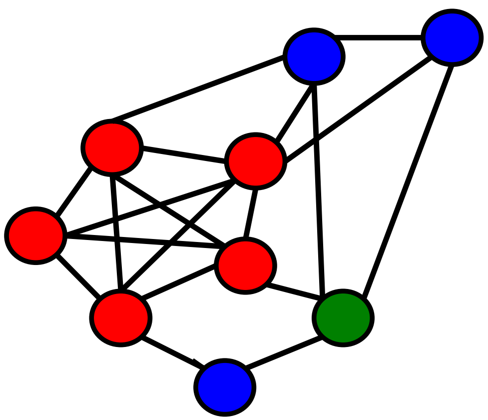
Online Monochromatic Clique

Clique: 

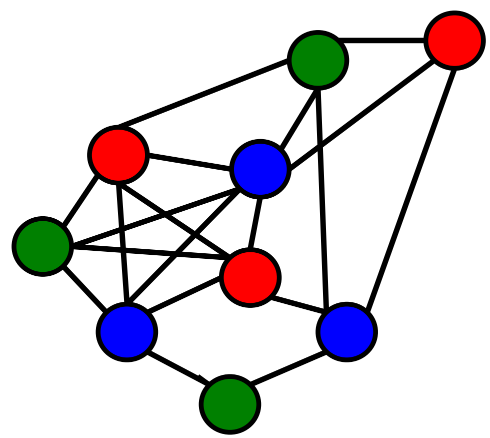
Given *fixed* of t colors: red, blue, and green. (here t = 3)

Objective: minimize the largest monochromatic clique.

Bad



Good



*i*th vertex arrives: online algorithm gets adjacencies with vertices *1, …, i-1*

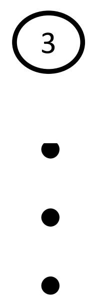
The Game: Bins versus Colors

(…or robots versus blue devils)

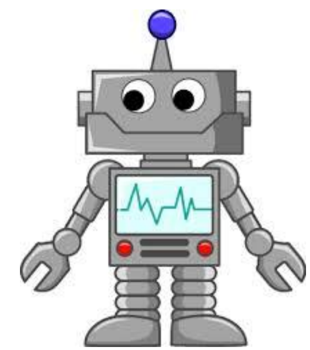
Number of colors: t = 4

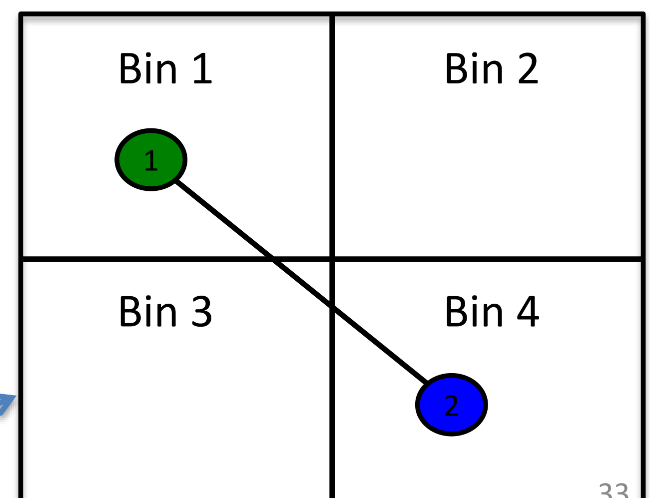
Adversary (us)

My turn!



Online Algorithm





1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO's color).
3. Adversary colors the vertex (OPT's decision)

Bins = algorithm's coloring

The Game: Bins versus Colors

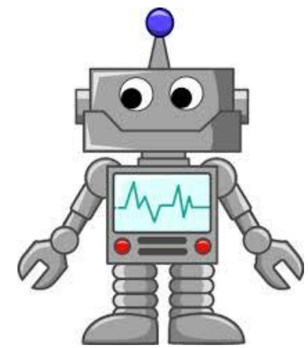
(…or robots versus blue devils)

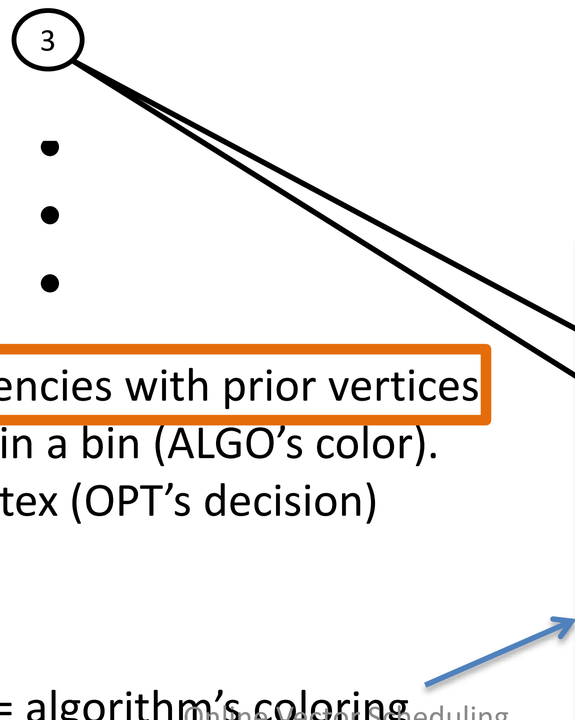
Number of colors: t = 4

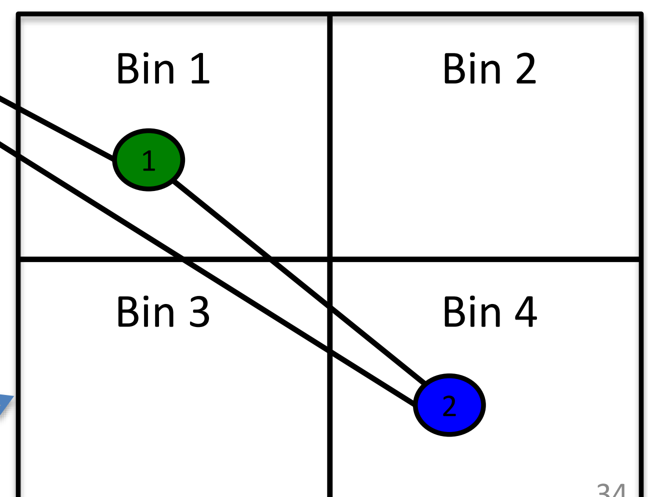
Adversary (us)

My turn!

Online Algorithm







1. Adversary defines adjacencies with prior vertices
2. Algorithm places vertex in a bin (ALGO's color).
3. Adversary colors the vertex (OPT's decision)

Bins = algorithm's coloring

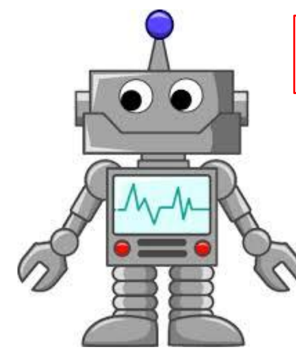
The Game: Bins versus Colors

(…or robots versus blue devils)

Number of colors: t = 4

Adversary (us)

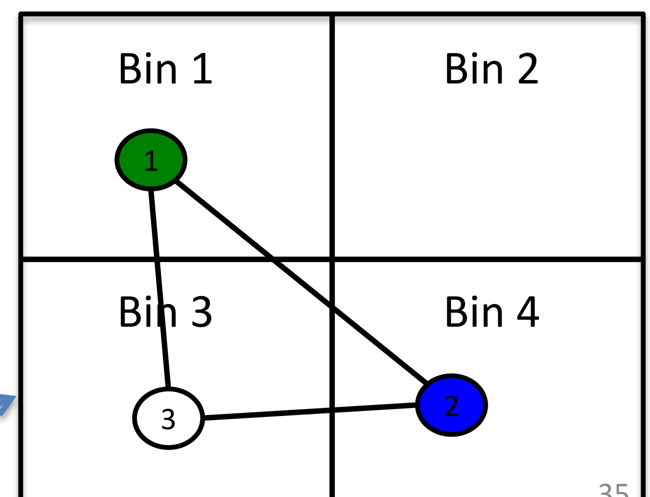
Online Algorithm



My turn!

⋮

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO's color).
3. Adversary colors the vertex (OPT's decision)



Bins = algorithm's coloring

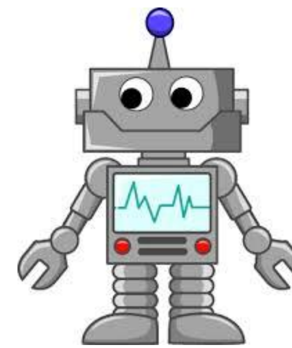
The Game: Bins versus Colors

(…or robots versus blue devils)

Number of colors: t = 4

Adversary (us)

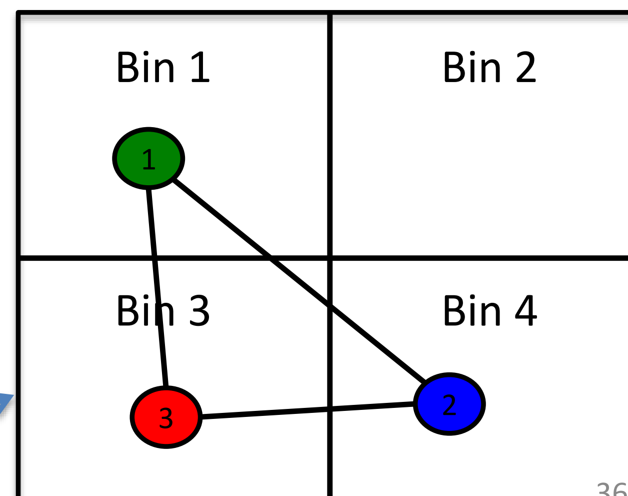
Online Algorithm



My turn!

⋮

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO's color).
3. Adversary colors the vertex (OPT's decision)



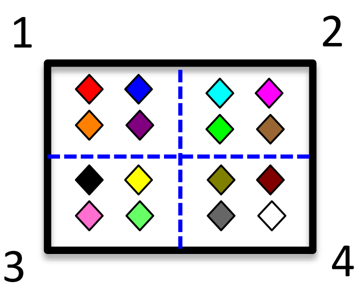
Bins = algorithm's coloring

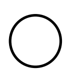
The Adversary Strategy

- Split every bin into $\sqrt{t}$ slots: each slot is associated with a distinct set of $\sqrt{t}$ colors

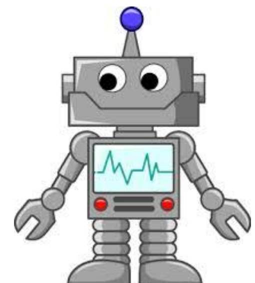
The Construction

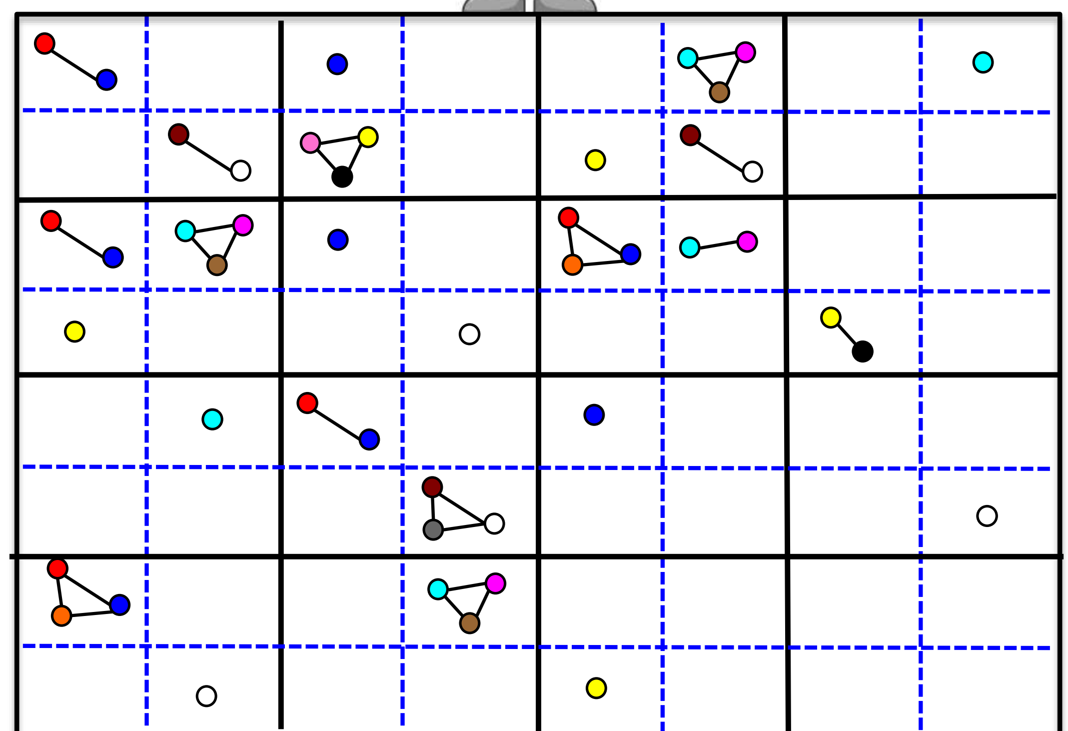
1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO's color).
3. Adversary colors the vertex (OPT's decision).





vertex i



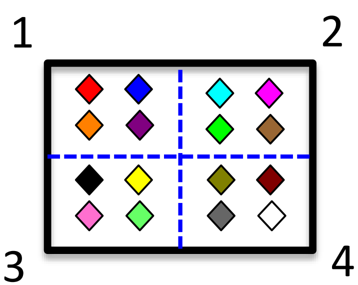


The Adversary Strategy

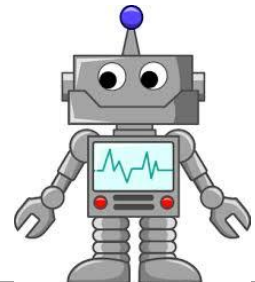
- Split every bin into $\sqrt{t}$ slots: each slot is associated with a distinct set of $\sqrt{t}$ colors
- Generate a "code": a sequence of strings of length t from a $\sqrt{t}$ alphabet
- For the i^{th} vertex, define adjacencies as follows (say $t = 16$):
  - Suppose the i^{th} string in the code is 1312121121413134
  - Then, add edges to all vertices in slot 1 of bin 1, slot 3 of bin 2, slot 1 of bin 3, etc

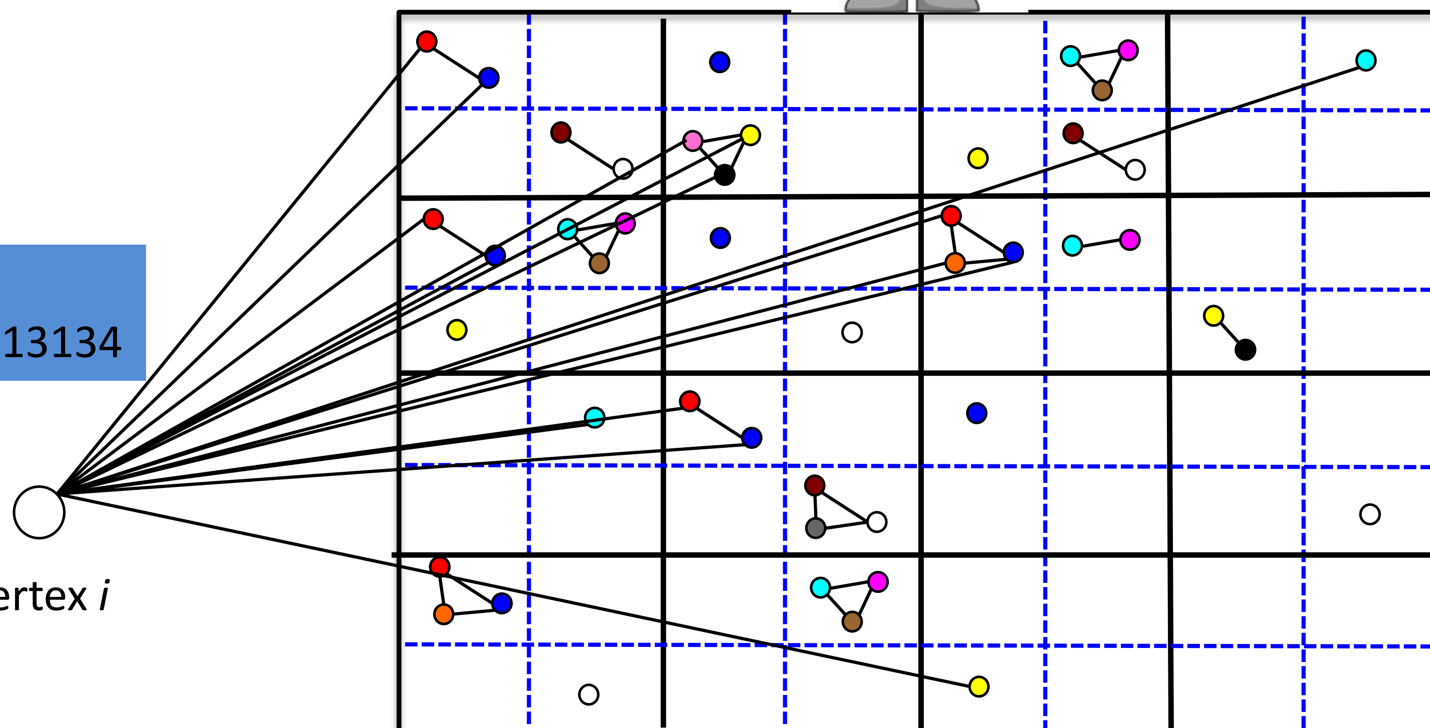
The Construction

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO's color).
3. Adversary colors the vertex (OPT's decision).



Code string:
131212121413134





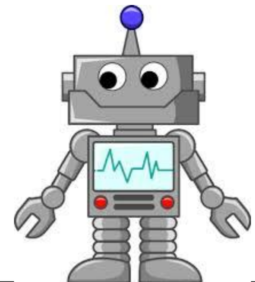
The Adversary Strategy

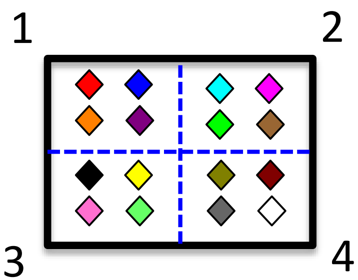
- Split every bin into √t slots: each slot is associated with a distinct set of √t colors
- Generate a "code": a sequence of strings of length t from a √t alphabet
- For the i^{th} vertex, define adjacencies as follows (say t = 16):
  - Suppose the i^{th} string in the code is 1312121121413134
  - Then, add edges to all vertices in slot 1 of bin 1, slot 3 of bin 2, slot 1 of bin 3, etc
  - If the algorithm places the vertex in bin 2, then place it in slot 3 of bin 2

The Construction

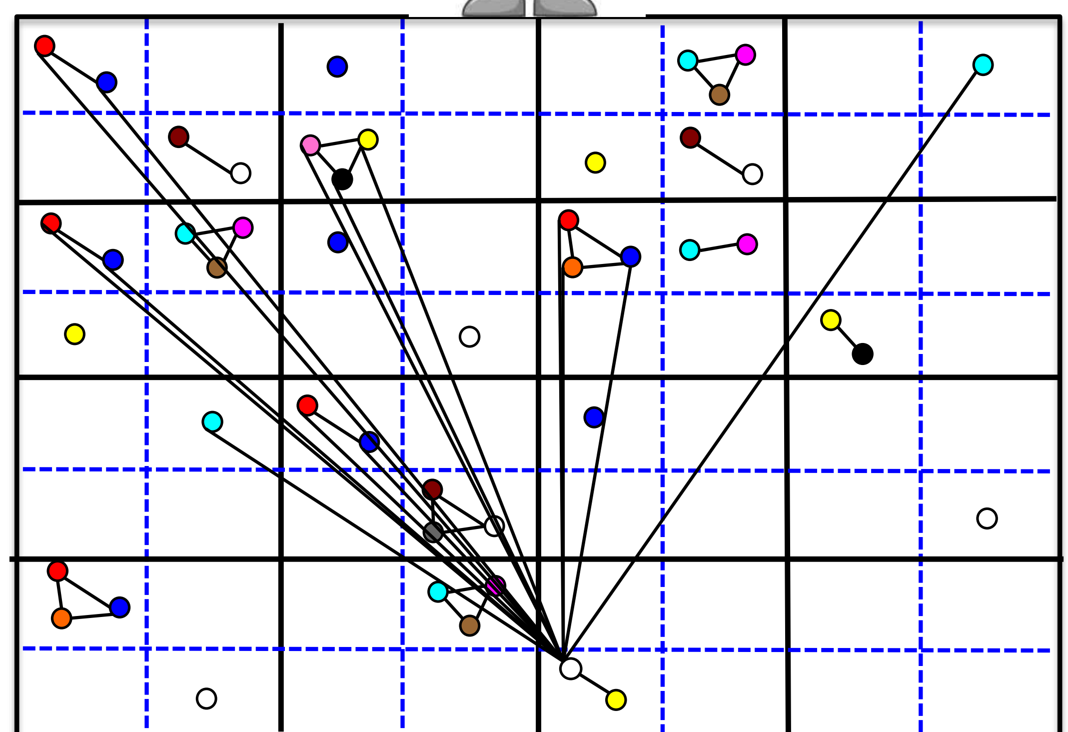
1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO's color).
3. Adversary colors the vertex (OPT's decision).

Bin 15





Code string:
131212121413134

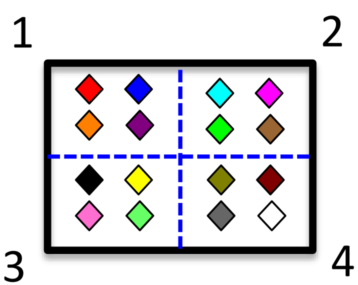


The Adversary Strategy

- Split every bin into $\sqrt{t}$ slots: each slot is associated with a distinct set of $\sqrt{t}$ colors
- Generate a "code": a sequence of strings of length t from a $\sqrt{t}$ alphabet
- For the i^{th} vertex, define adjacencies as follows (say t = 16):
  - Suppose the i^{th} string in the code is 1312121121413134
  - Then, add edges to all vertices in slot 1 of bin 1, slot 3 of bin 2, slot 1 of bin 3, etc
  - If the algorithm places the vertex in bin 2, then place it in slot 3 of bin 2
  - OPT colors the vertex with a color from the $\sqrt{t}$ colors associated with slot 3 that is currently unused in bin 2

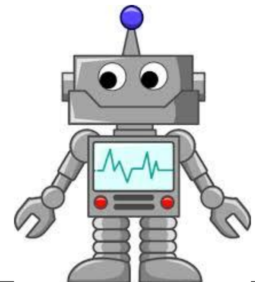
The Construction

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO's color).
3. Adversary colors the vertex (OPT's decision).

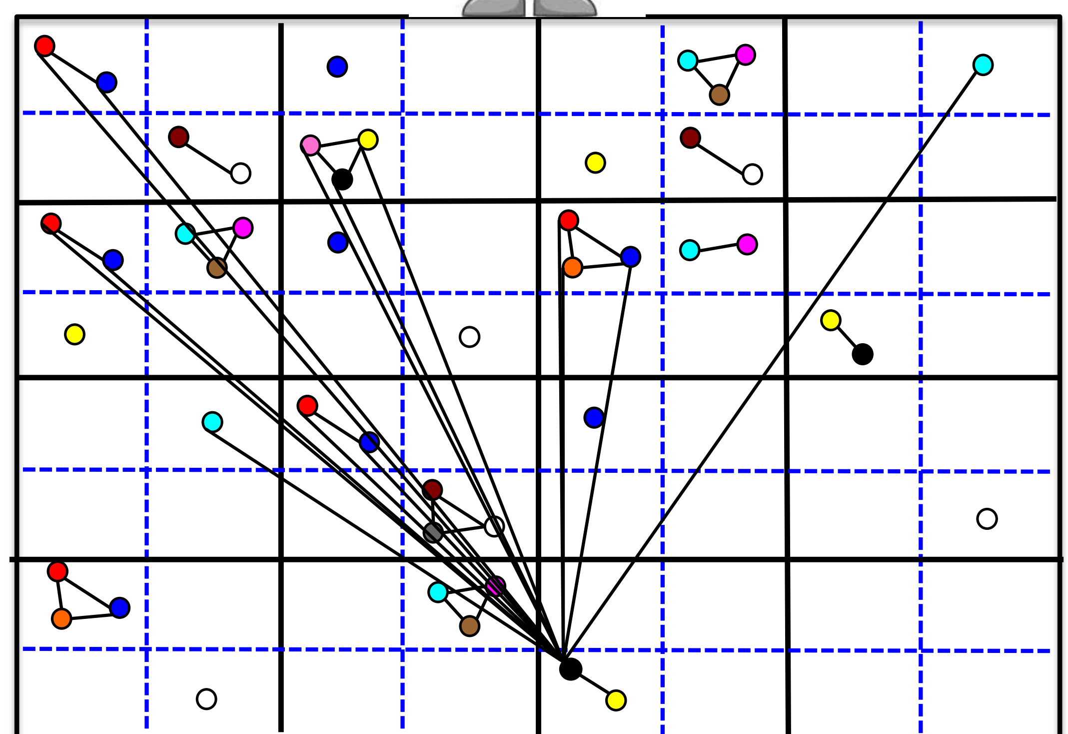


Bin 15

Color black



Code string:
131212121413134

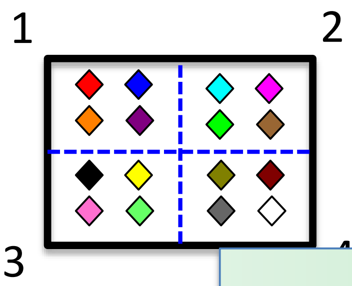


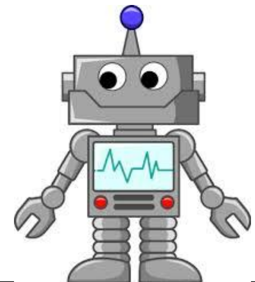
The Adversary Strategy

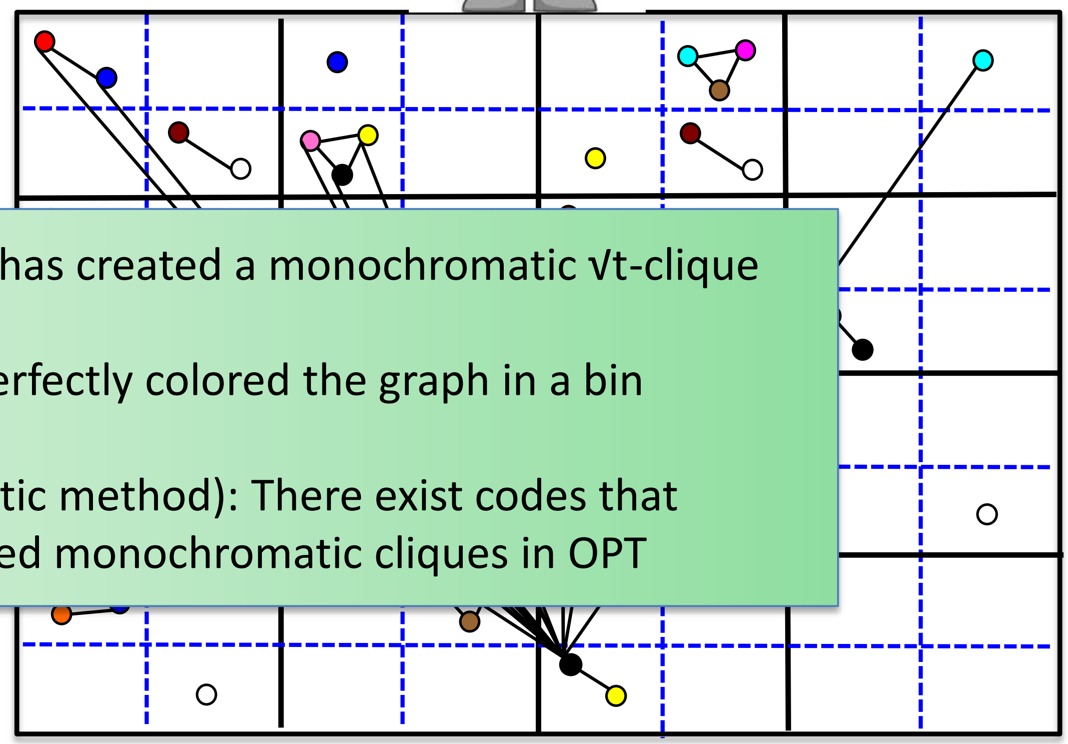
- Split every bin into √t slots: each slot is associated with a distinct set of √t colors
- Generate a "code": a sequence of strings of length t from a √t alphabet
- For the i^{th} vertex, define adjacencies as follows (say t = 16):
  - Suppose the i^{th} string in the code is 1312121121413134
  - Then, add edges to all vertices in slot 1 of bin 1, slot 3 of bin 2, slot 1 of bin 3, etc
  - If the algorithm places the vertex in bin 2, then place it in slot 3 of bin 2
  - OPT colors the vertex with a color from the √t colors associated with slot 3 that is currently unused in bin 2
- Terminate when some slot in some bin has √t vertices

The Construction

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO's color).
3. Adversary colors the vertex (OPT's decision).







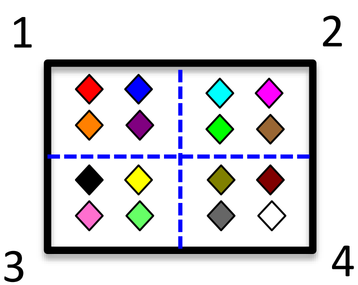
Observation 1: Algorithm has created a monochromatic √t-clique

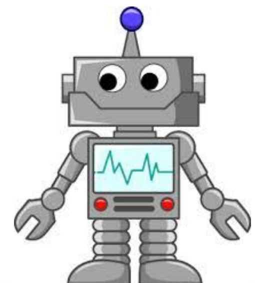
Observation 2: OPT has perfectly colored the graph in a bin

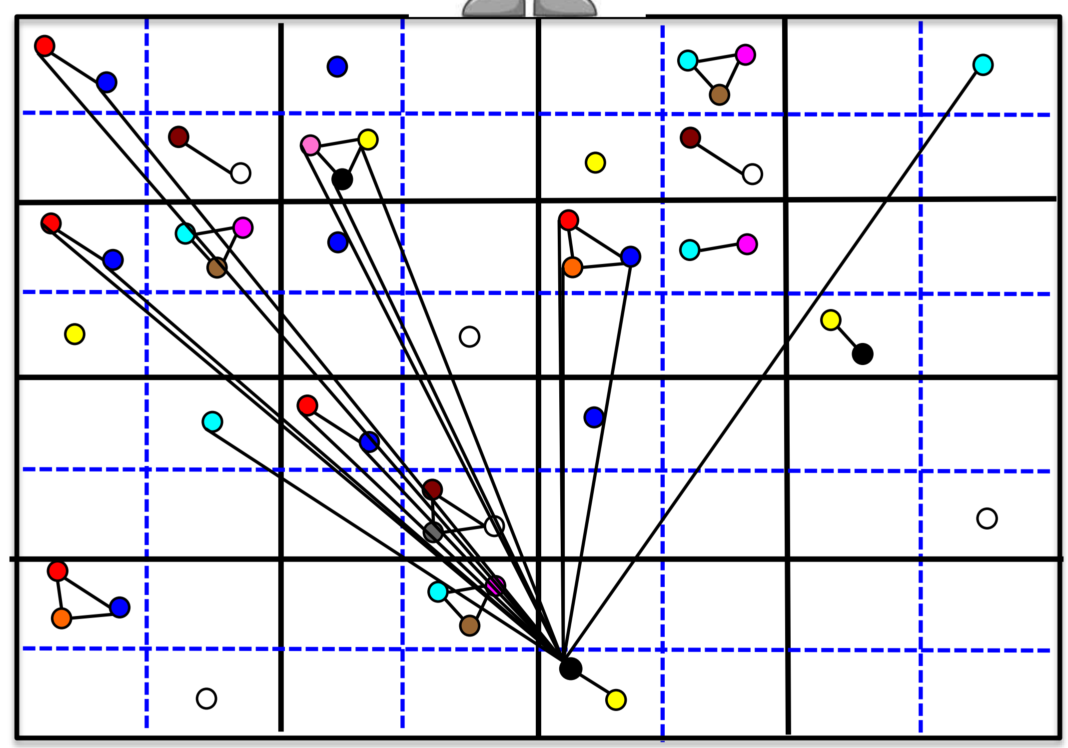
Lemma (via the probabilistic method): There exist codes that produce only constant-sized monochromatic cliques in OPT

The Construction

1. Adversary defines adjacencies with prior vertices.
2. Algorithm places vertex in a bin (ALGO's color).
3. Adversary colors the vertex (OPT's decision).



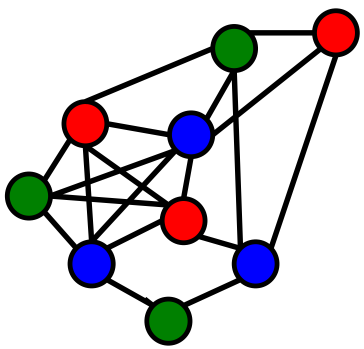




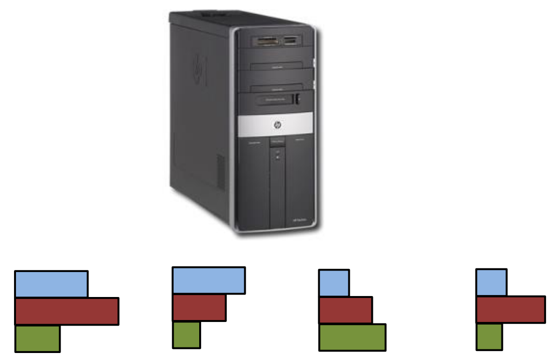
$\Omega(\sqrt{t})$ lower bound

(t = number colors)

Now for the reduction…



implies



Coloring lower bound

$\Omega(\sqrt{t})$ lower bound

Ω(log d/log log d) lower bound
For vector scheduling

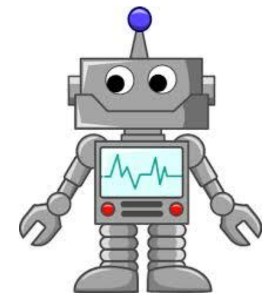
Using MC Lower Bound for Vector Scheduling

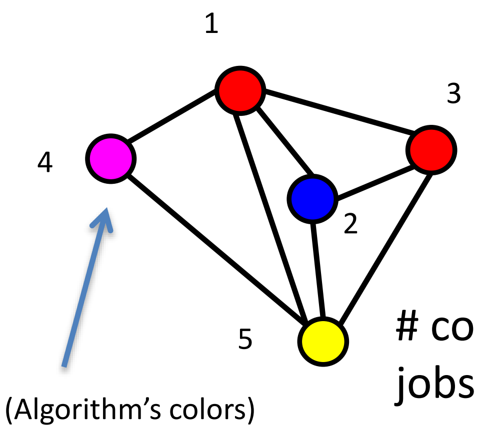
$m = 9$ machines

Issue $m^2 = 81$ jobs

Job dimension $d = \binom{m^2}{\sqrt{m}} = \binom{81}{3}$

$m^{O(m)}$

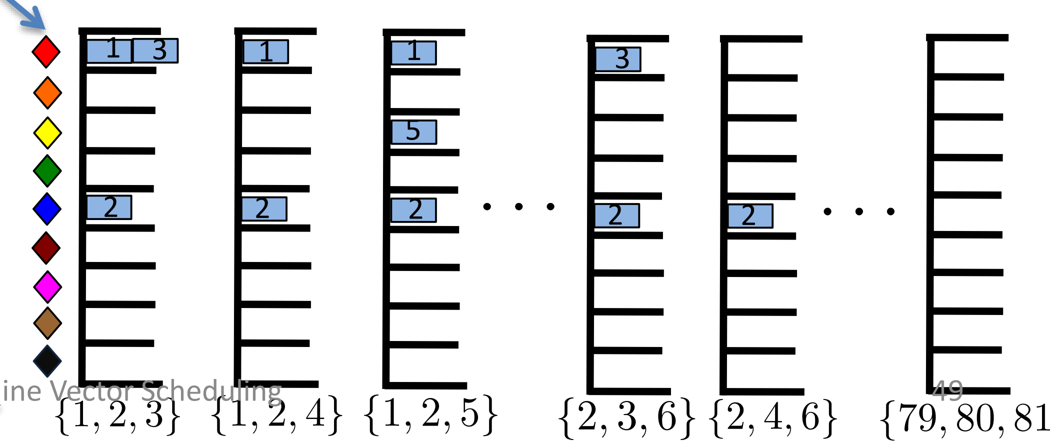




colors t = m

jobs <-> vertices

Colors correspond to machines



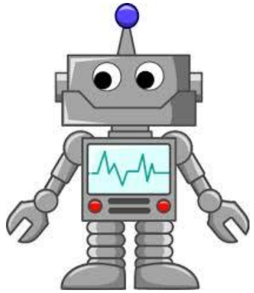
Dimensions correspond to √m size subsets of {1,…, m²}

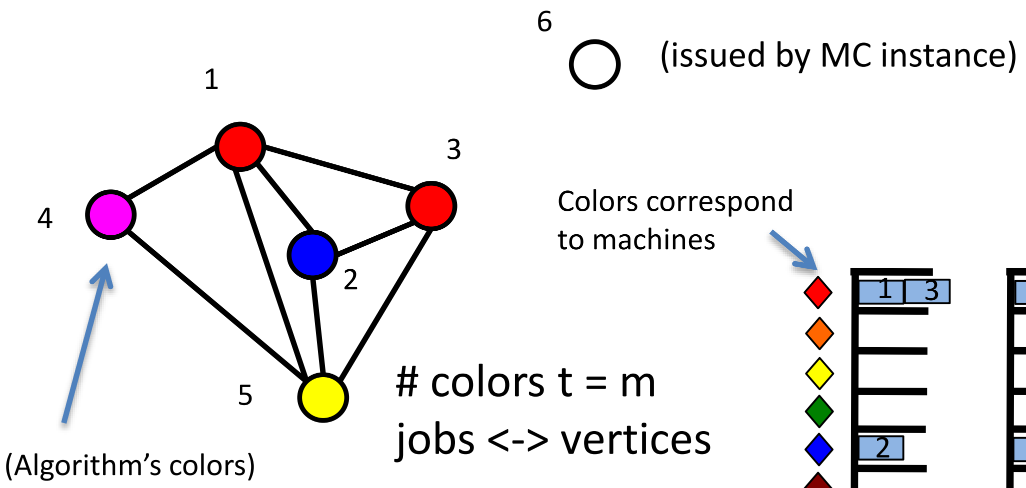
Using MC Lower Bound for Vector Scheduling

$m = 9$ machines

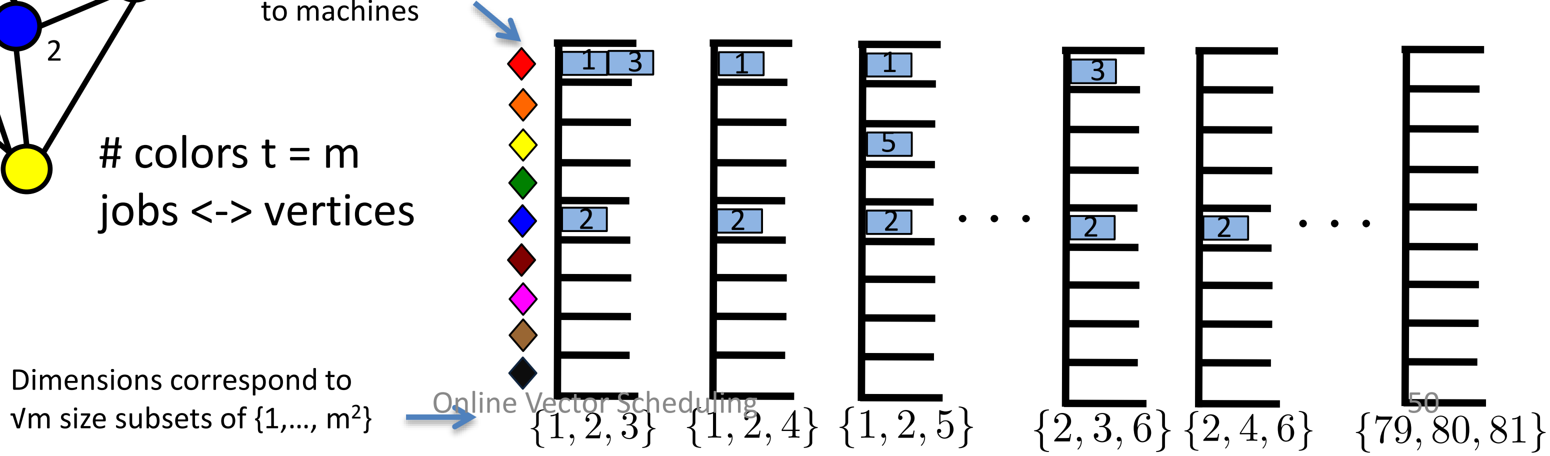
Issue $m^2 = 81$ jobs

Job dimension $d = \binom{m^2}{\sqrt{m}} = \binom{81}{3}$





# colors t = m
jobs <-> vertices

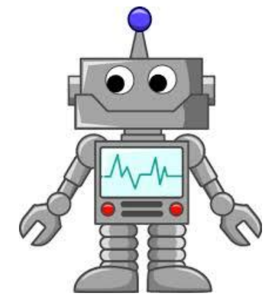


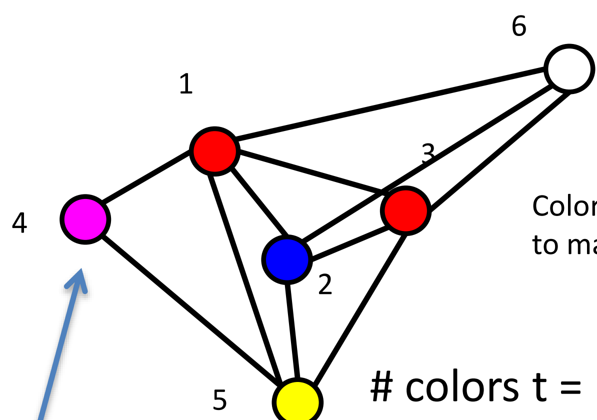
Using MC Lower Bound for Vector Scheduling

$m = 9$ machines

Issue $m^2 = 81$ jobs

Job dimension $d = \binom{m^2}{\sqrt{m}} = \binom{81}{3}$





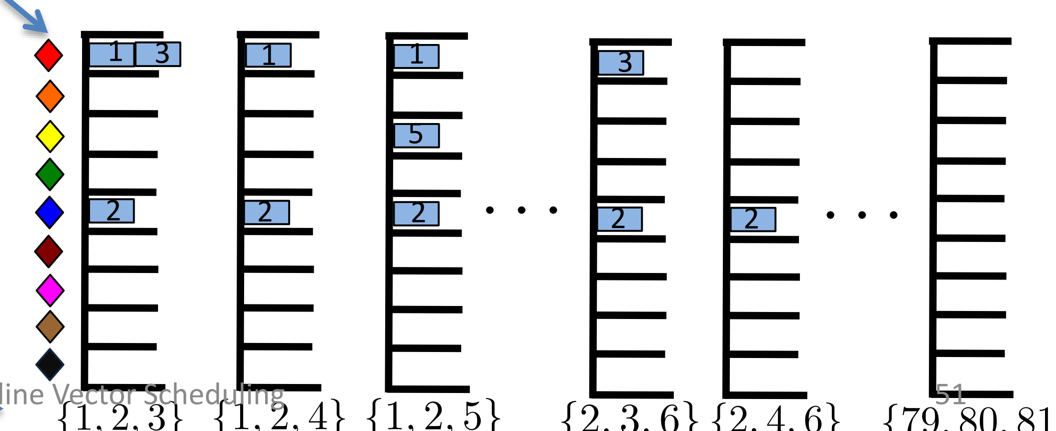
(issued by MC instance)

(Algorithm's colors)

# colors t = m
jobs <-> vertices

Colors correspond to machines

Dimensions correspond to √m size subsets of {1,…, m²}

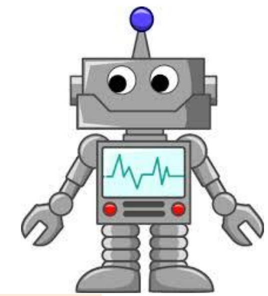


Using MC Lower Bound for Vector Scheduling

$m = 9$ machines

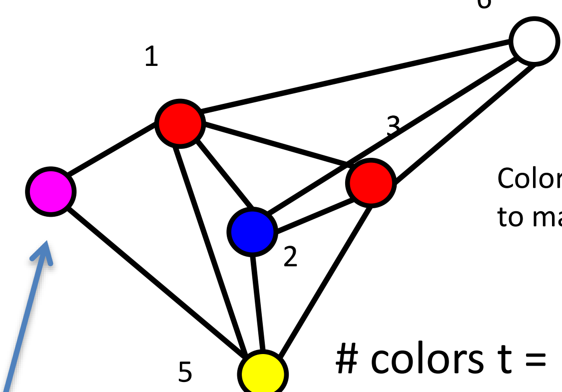
Issue $m^2 = 81$ jobs

Job dimension $d = \binom{m^2}{\sqrt{m}} = \binom{81}{3}$



1 iff vertex forms a clique with previous vertices in the set

(issued by MC instance)



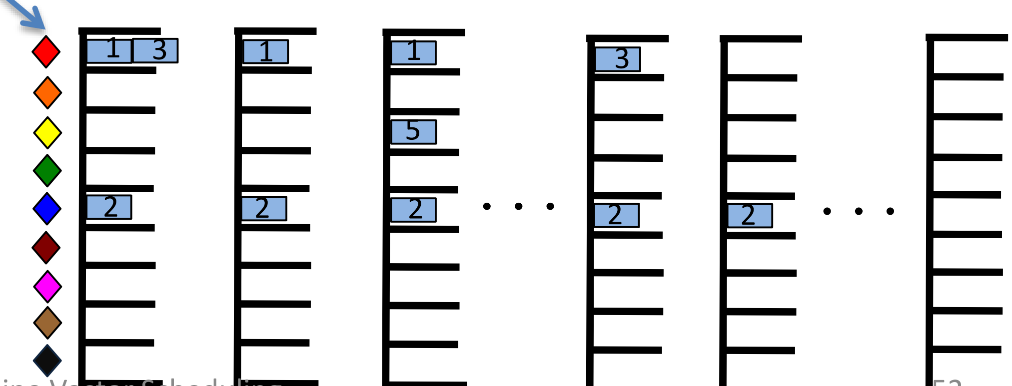
(Algorithm's colors)

colors t = m

jobs <-> vertices

Colors correspond to machines

(0, 0 , 0 , . . . 1, 0, . . . 0)



Dimensions correspond to √m size subsets of {1,..., m²}

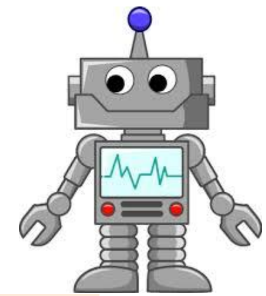
$\{1,2,3\}$ $\{1,2,4\}$ $\{1,2,5\}$ $\{2,3,6\}$ $\{2,4,6\}$ $\{79,80,81\}$

Using MC Lower Bound for Vector Scheduling

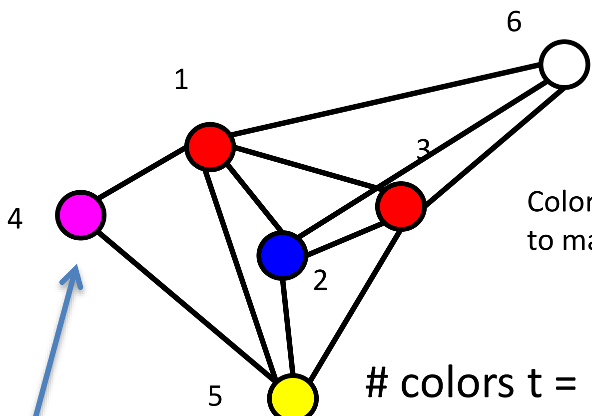
$m = 9$ machines

Issue $m^2 = 81$ jobs

Job dimension $d = \binom{m^2}{\sqrt{m}} = \binom{81}{3}$



1 iff vertex forms a clique with previous vertices in the set



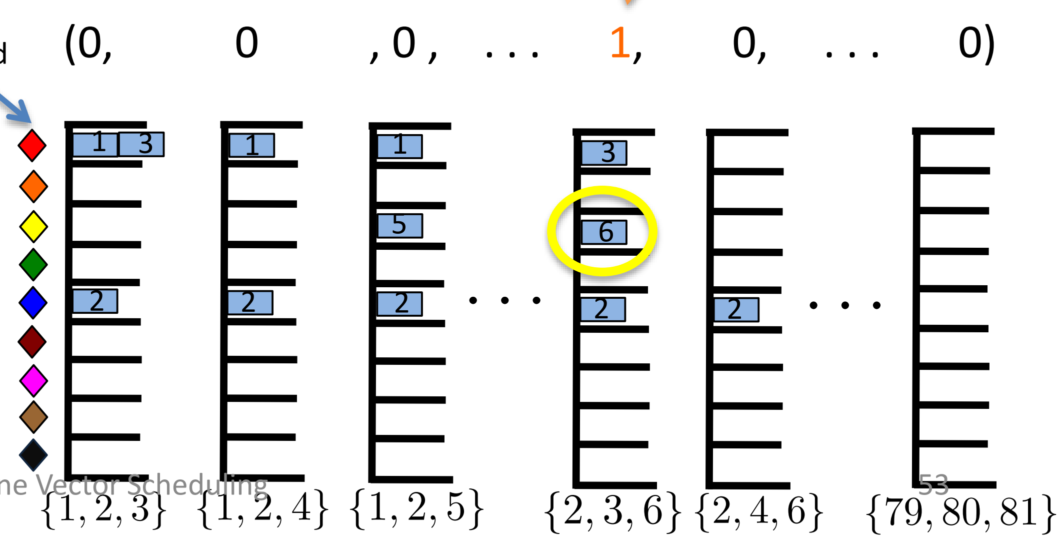
(issued by MC instance)

(Algorithm's colors)

\# colors t = m

jobs <-> vertices

Colors correspond to machines



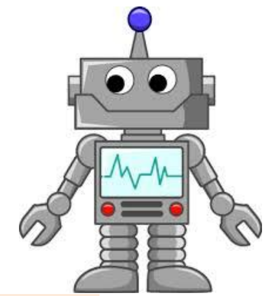
Dimensions correspond to √m size subsets of {1,…, m²}

Using MC Lower Bound for Vector Scheduling

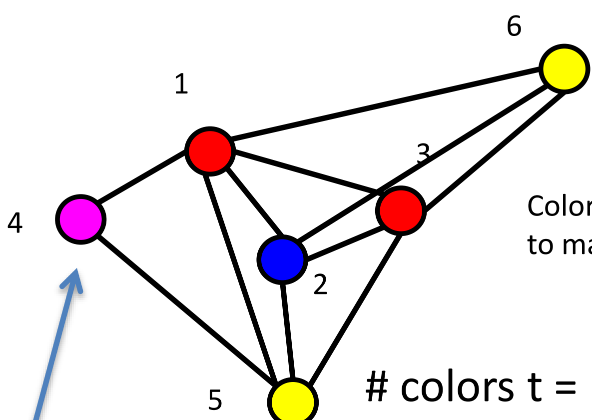
$m = 9$ machines

Issue $m^2 = 81$ jobs

Job dimension $d = \binom{m^2}{\sqrt{m}} = \binom{81}{3}$



1 iff vertex forms a clique with previous vertices in the set



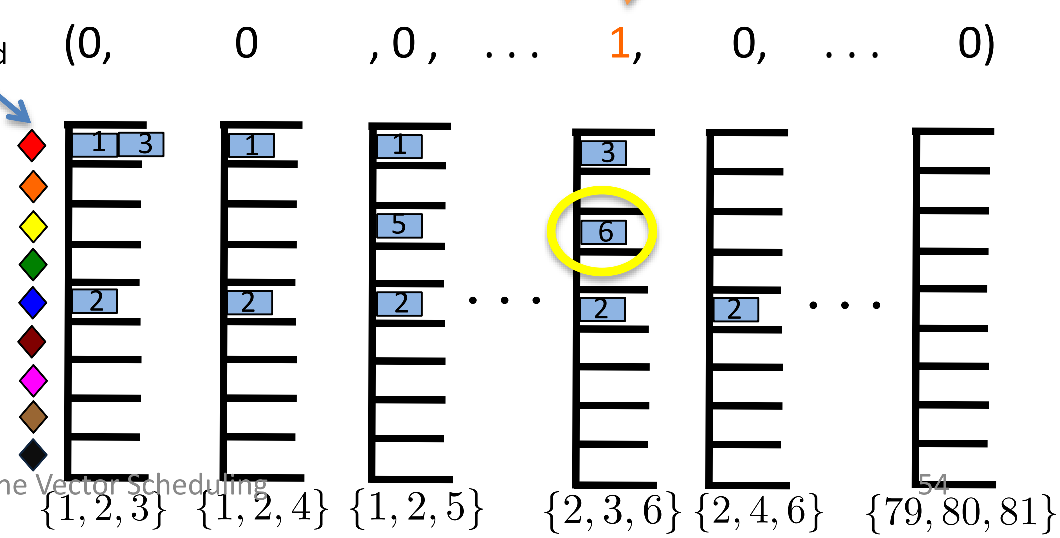
(issued by MC instance)

(Algorithm's colors)

\# colors t = m

jobs <-> vertices

Colors correspond to machines



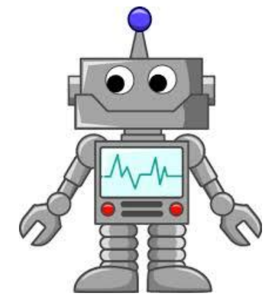
Dimensions correspond to √m size subsets of {1,…, m²}

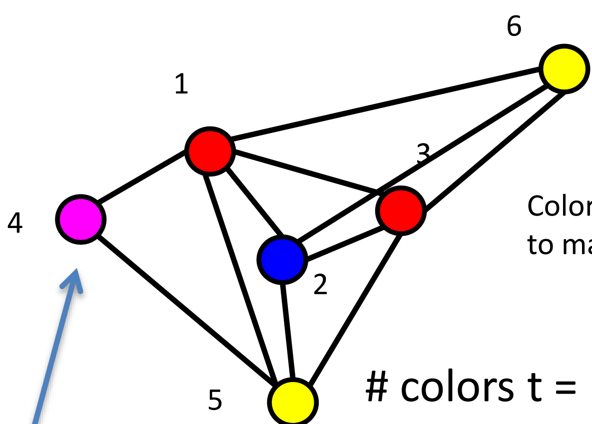
Using MC Lower Bound for Vector Scheduling

$m = 9$ machines

Issue $m^2 = 81$ jobs

Job dimension $d = \binom{m^2}{\sqrt{m}} = \binom{81}{3}$





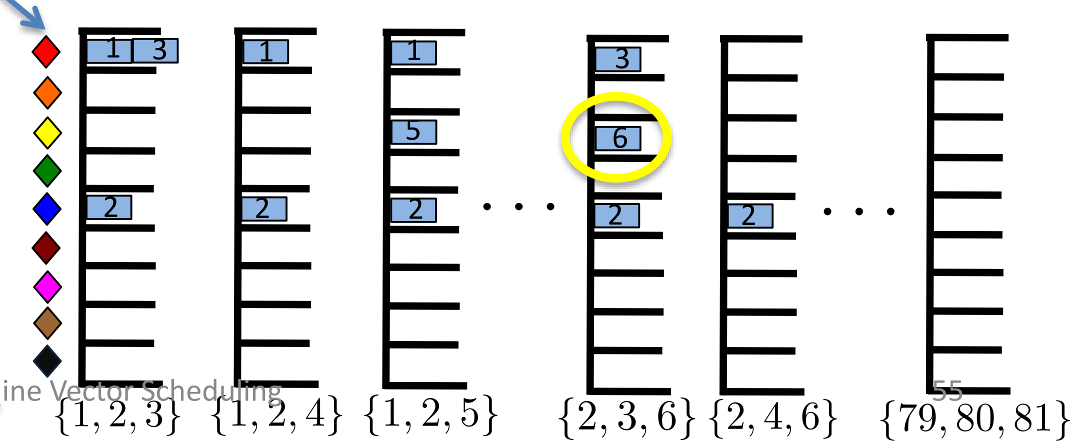
(Algorithm's colors)

\# colors t = m

jobs <-> vertices

1. After m^2 vertices, there will exist a monochromatic clique of size $\sqrt{m}$ on some color c.
2. => dimension corresponding to these vertices will have a load of $\sqrt{m}$ on machine c.
3. Size of largest monochromatic clique in OPT's graph coloring is O(1).
4. ALGO/OPT => $\Omega(\sqrt{m}) = \Omega(\log d / \log\log d)$

Colors correspond to machines



Dimensions correspond to $\sqrt{m}$ size subsets of $\{1,\ldots,m^2\}$

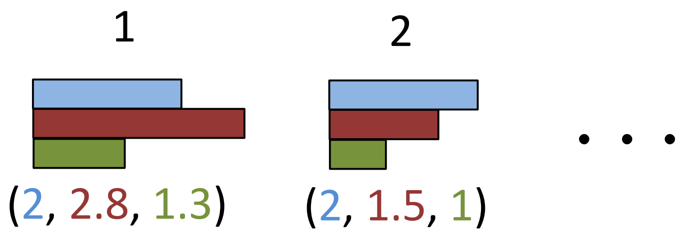
Summary of Results

	Makespan minimization	**p-norm minimization**	
Identical machines	O(log d) [Azar *et al* ’13, Meyerson *et al* ’14] Our result: **Θ(log d/log log d)**	Our result: **$\Theta((\log d/\log\log d)^{1-1/p})$**	(Im-Kulkarni-Kell-P. FOCS ’15)
Unrelated machines (machine dependent loads)	O(log d + log m) [Meyerson *et al* ’14] Our result: **Θ(log d + log m)**	Our result: **Θ(log d + p)**	
Related machines (non-uniform machine speeds)			(Im-Kell-P.-Shadloo ’17)

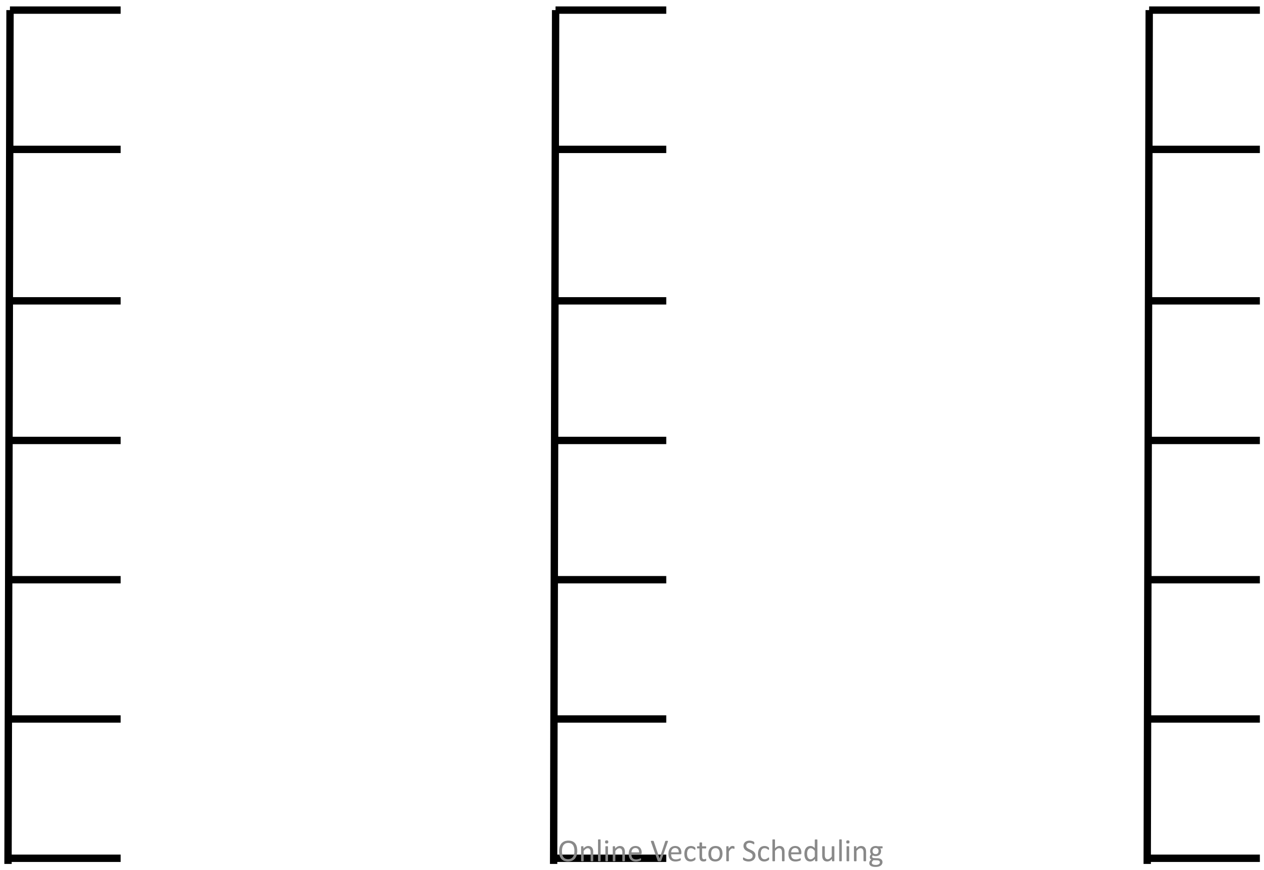
Related Machines (homogenous)

Processing time = load/speed

Jobs:



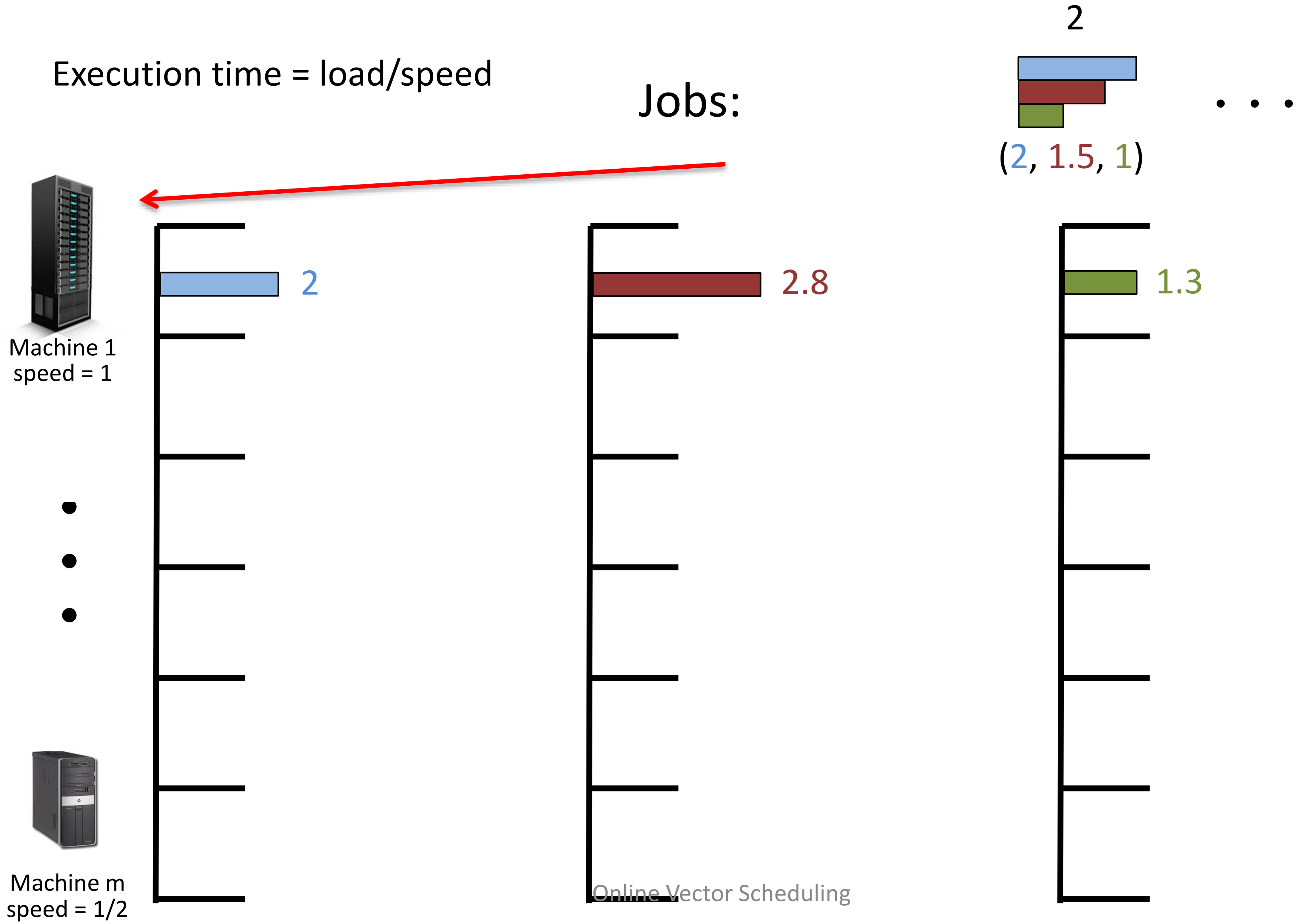
Machine 1
speed = 1



Machine m
speed = 1/2

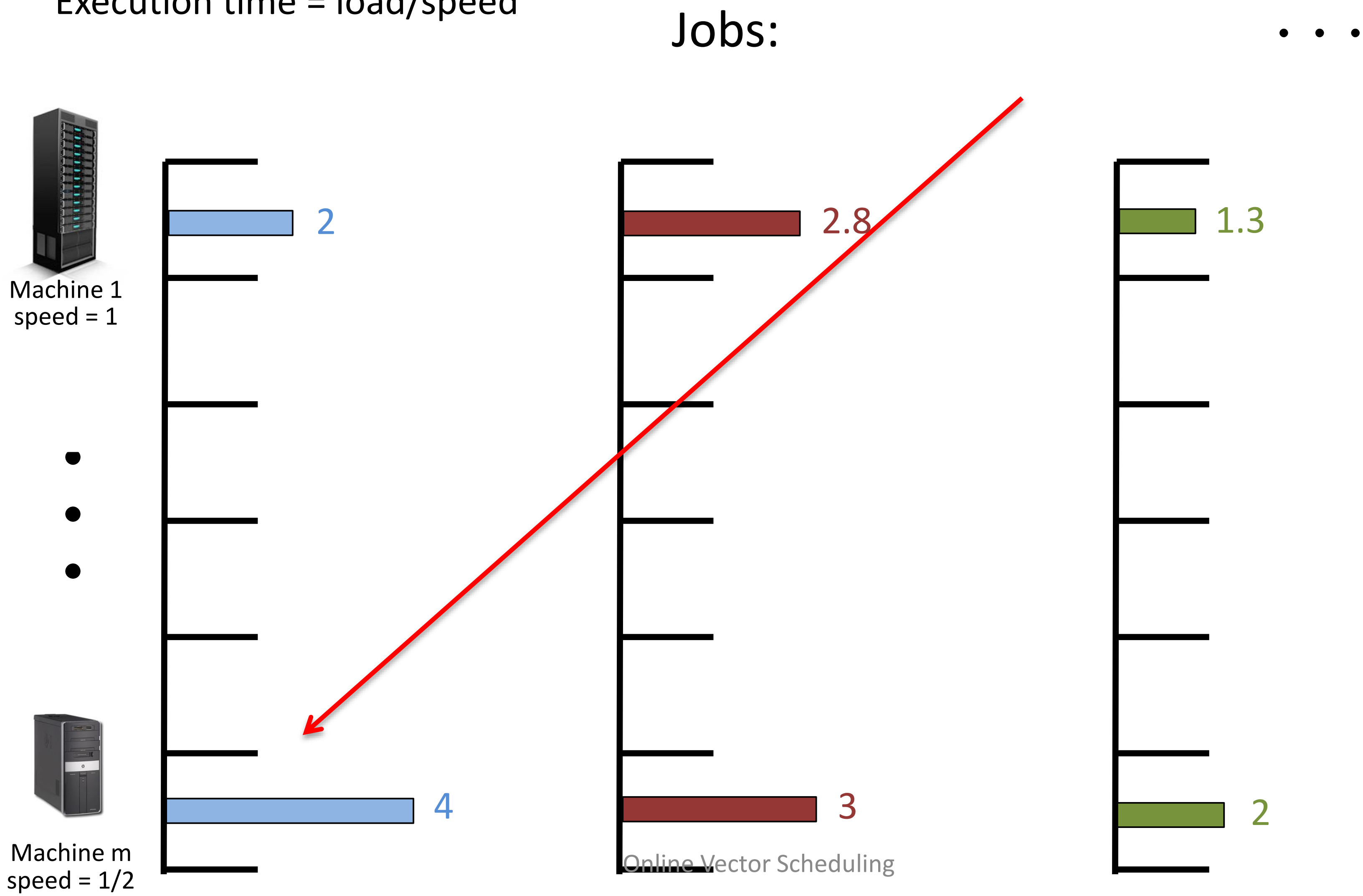
Related Machines (homogenous)

Execution time = load/speed



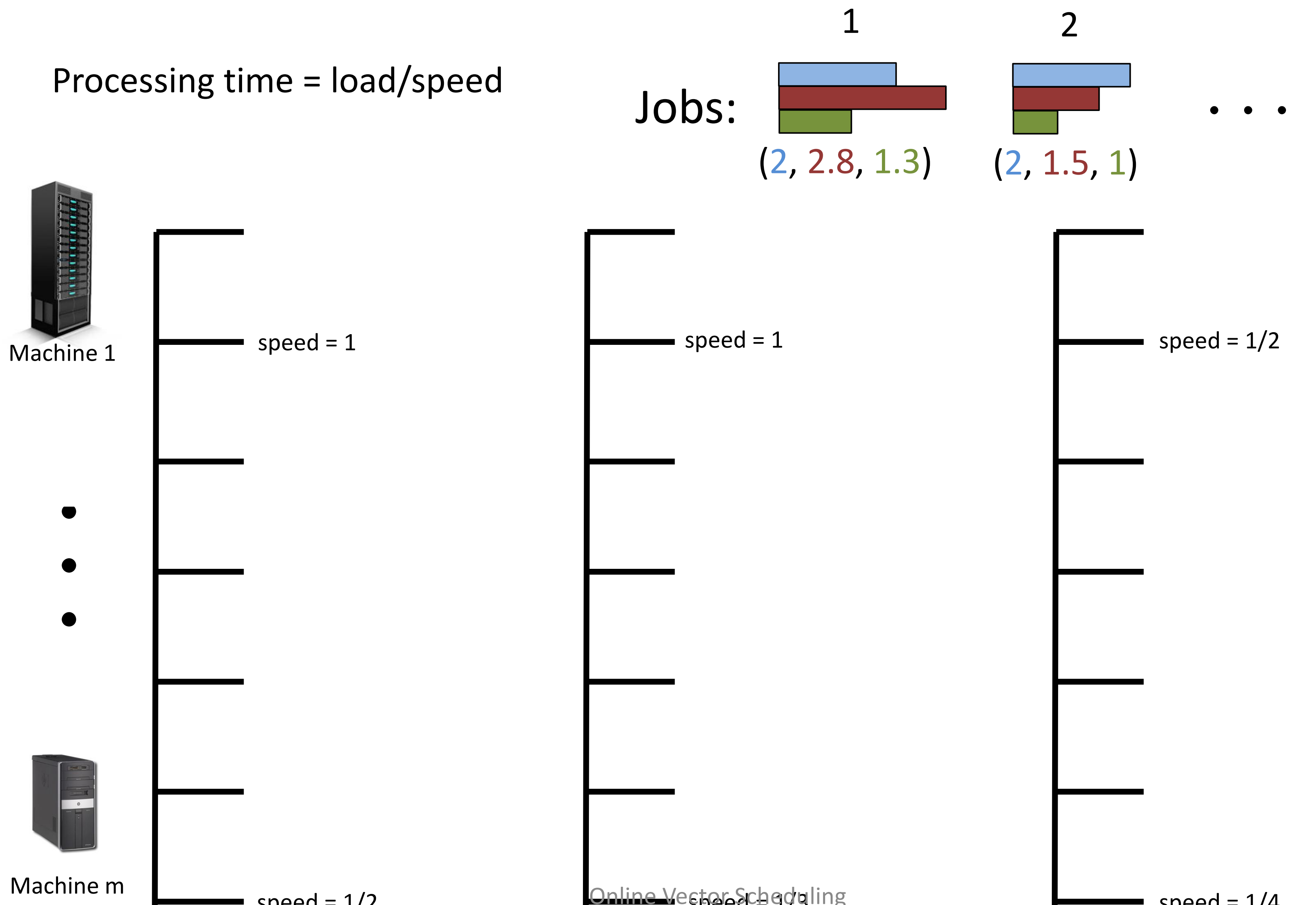
Related Machines (homogenous)

Execution time = load/speed



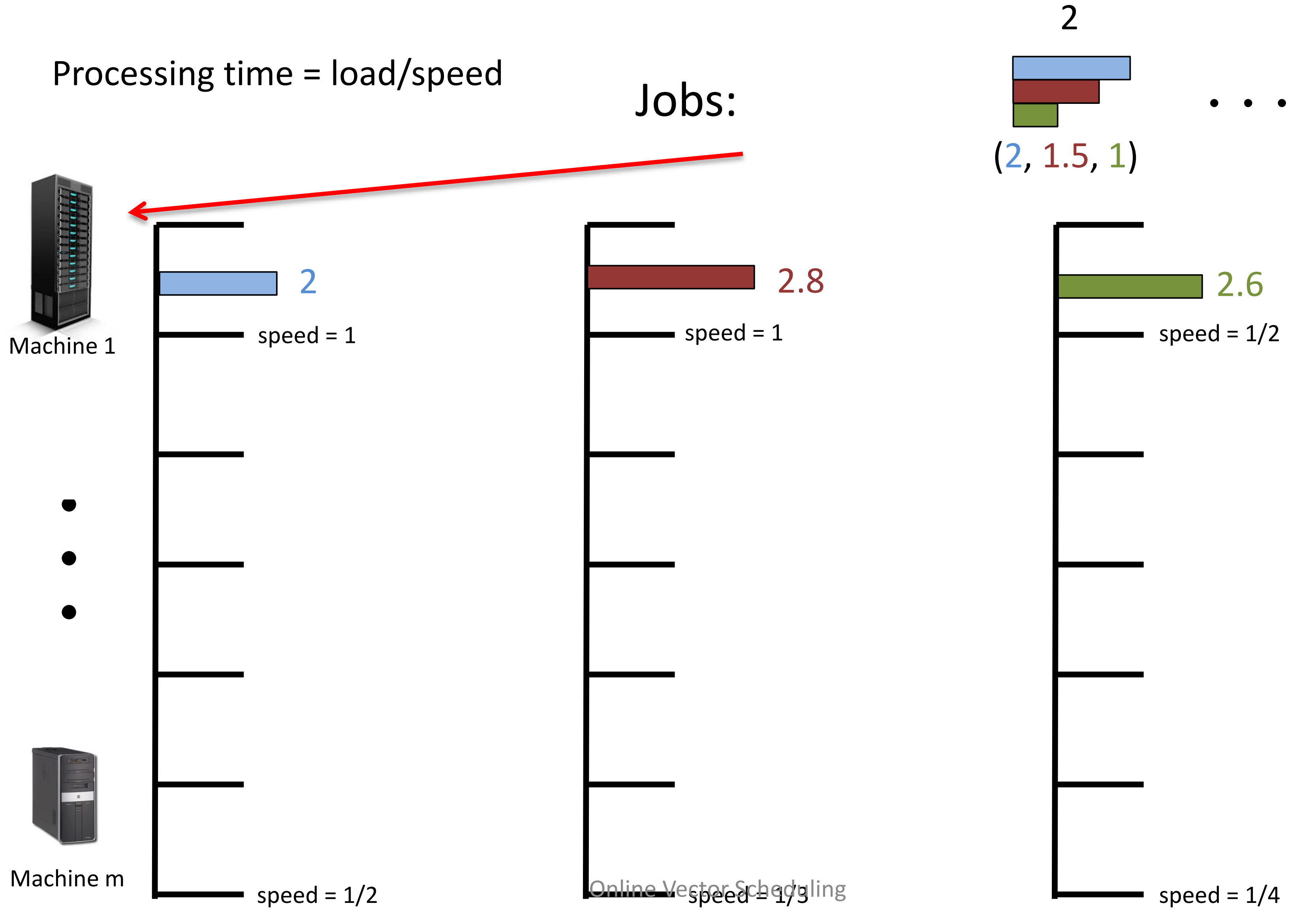
Related Machines (heterogeneous)

Processing time = load/speed



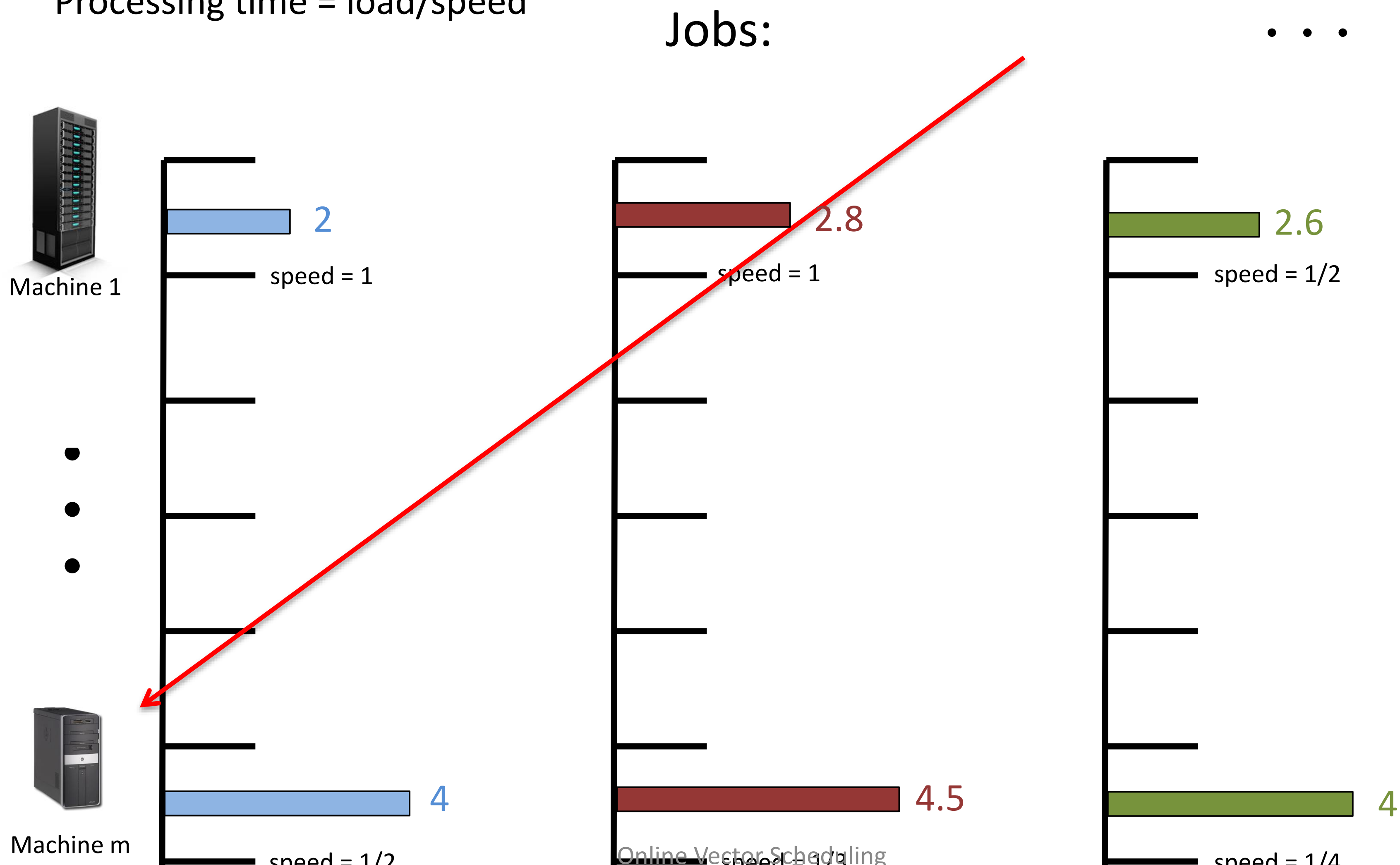
Related Machines (heterogeneous)

Processing time = load/speed



Related Machines (heterogeneous)

Processing time = load/speed



Summary of Results

		Makespan minimization	p-norm minimization	
Identical machines		$O(\log d)$ [Azar *et al* '13, Meyerson *et al* '14] Our result: **$\Theta(\log d/\log\log d)$**	Our result: **$\Theta((\log d/\log\log d)^{1-1/p})$**	(Im-Kulkarni-Kell-P. FOCS '15)
Unrelated machines (machine dependent loads)		$O(\log d + \log m)$ [Meyerson *et al* '14] Our result: **$\Theta(\log d + \log m)$**	Our result: **$\Theta(\log d + p)$**	
Related machines (non-uniform machine speeds)	Homo-geneous	Our result: **$\Theta(\log d/\log\log d)$**	Our result: **$O(\log^3 d)$**	(Im-Kell-P.-Shadloo '17)
	Hetero-geneous	Our result: **$\Theta(\log d + \log m)$**	Our result: **$\Theta(\log d + p)$**	

Summary of Results

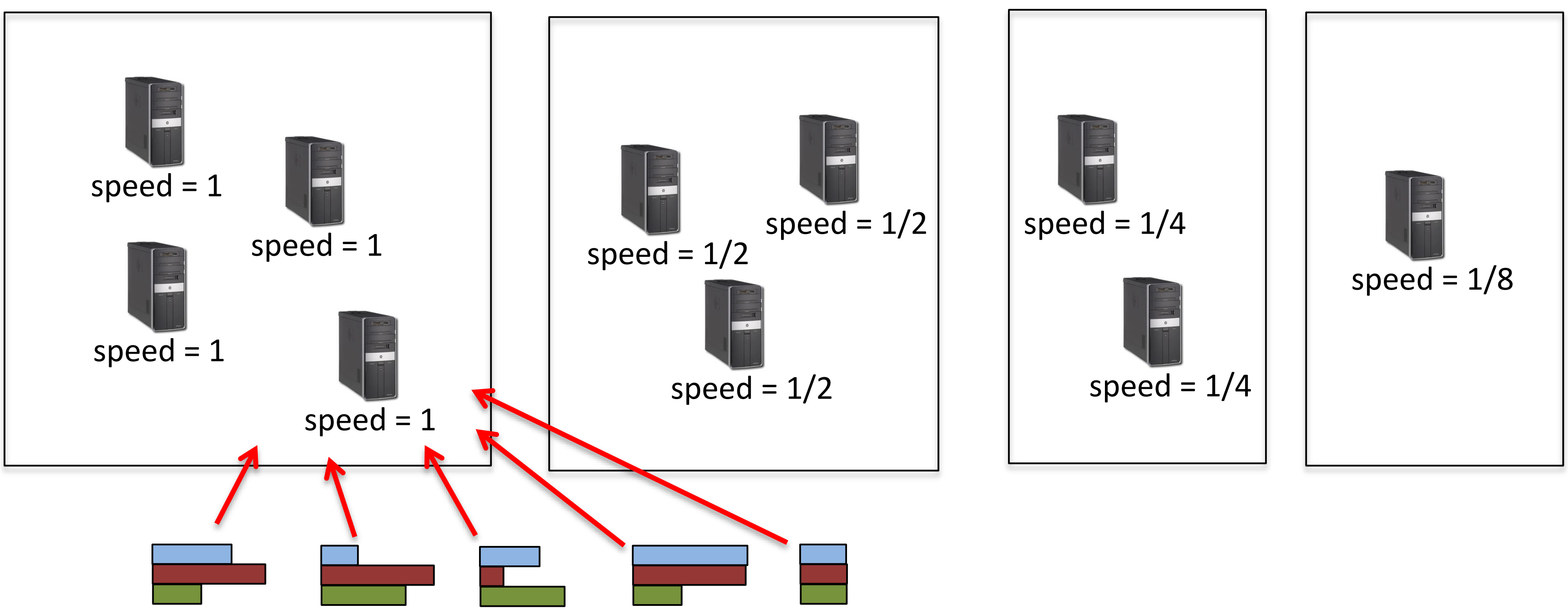
		Makespan minimization	p-norm minimization	
Identical machines		$O(\log d)$ [Azar *et al* ’13, Meyerson *et al* ’14] Our result: $\Theta(\log d/\log\log d)$	Our result: $\Theta((\log d/\log\log d)^{1-1/p})$	(Im-Kulkarni-Kell-P. FOCS ’15)
Unrelated machines (machine dependent loads)		$O(\log d + \log m)$ [Meyerson *et al* ’14] Our result: $\Theta(\log d + \log m)$	Our result: $\Theta(\log d + p)$	
Related machines (non-uniform machine speeds)	Homo-geneous	Our result: $\Theta(\log d/\log\log d)$	Our result: $O(\log^3 d)$	(Im-Kell-P.-Shadloo ’17)
	Hetero-geneous	Our result: $\Theta(\log d + \log m)$	Our result: $\Theta(\log d + p)$	

First $O(1)$ competitive for $d = 1$

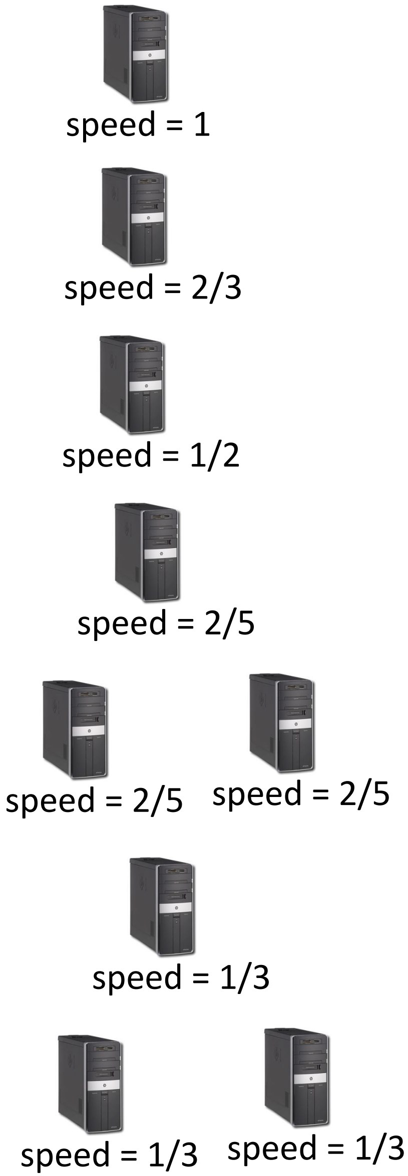
Machine Grouping

Want to reduce problem to identical machines…
Natural to try to groups machines of similar speed.

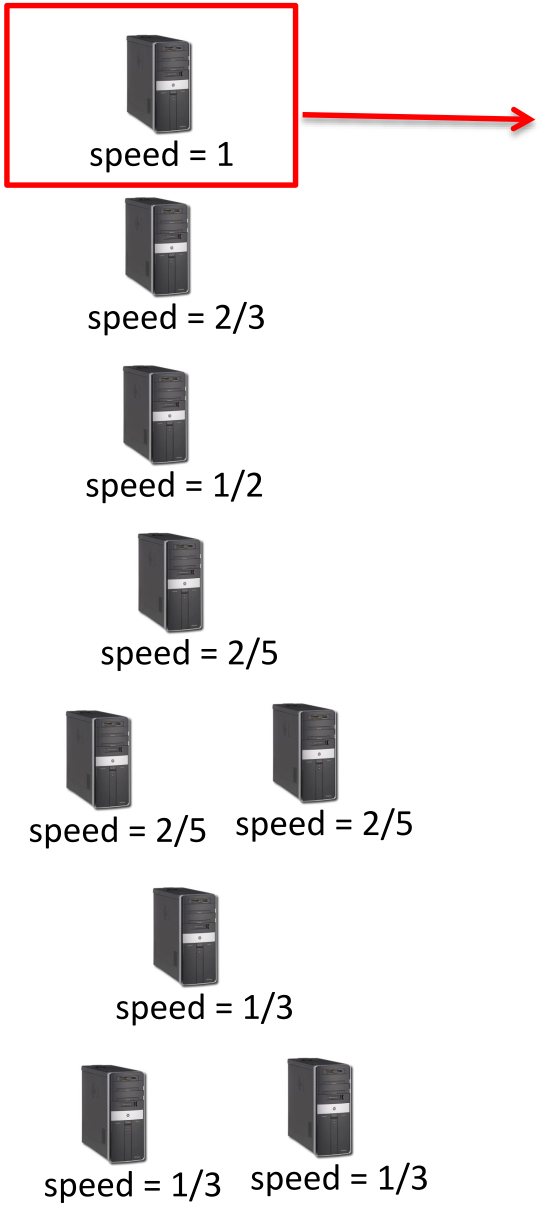
Issue: if total speed (processing power) of faster machines is large, slower machines go unutilized.

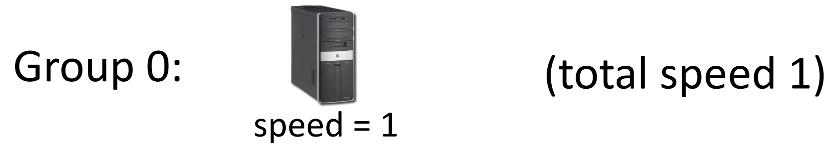


Machine Smoothing



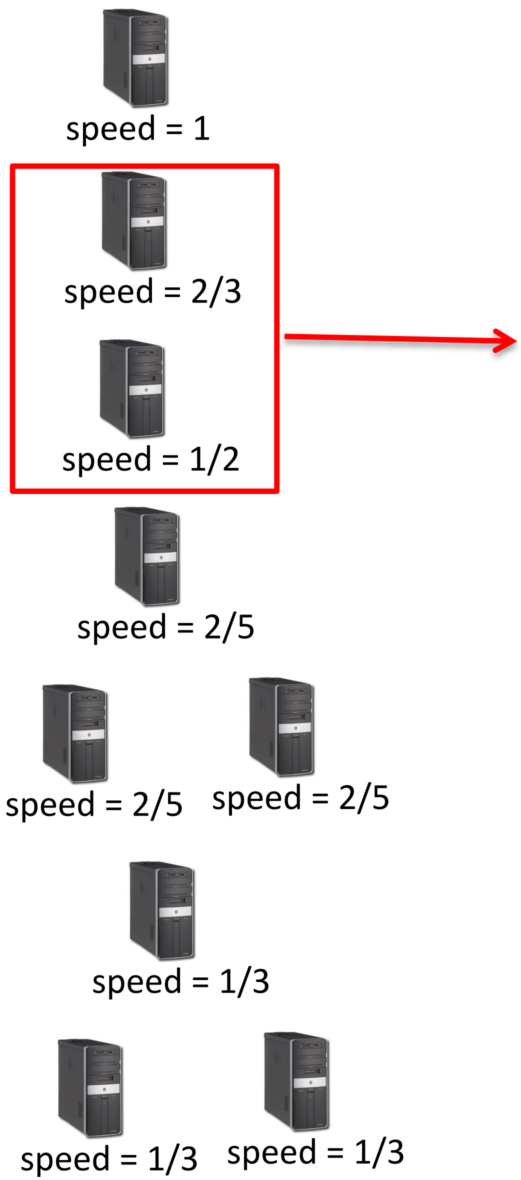
Machine Smoothing

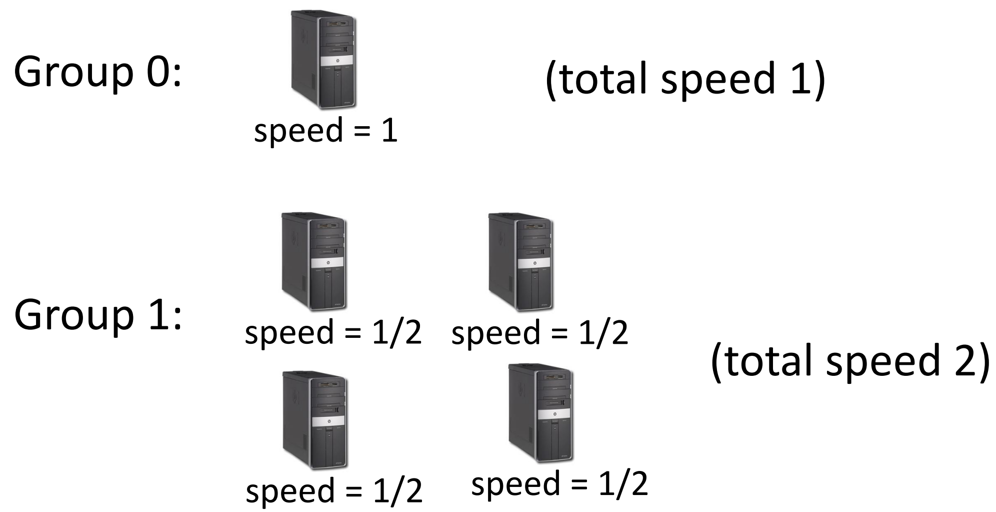




- Group machines so that *total* speed increases exponentially.
- Replace machines with identical machines with (roughly) same total speed.

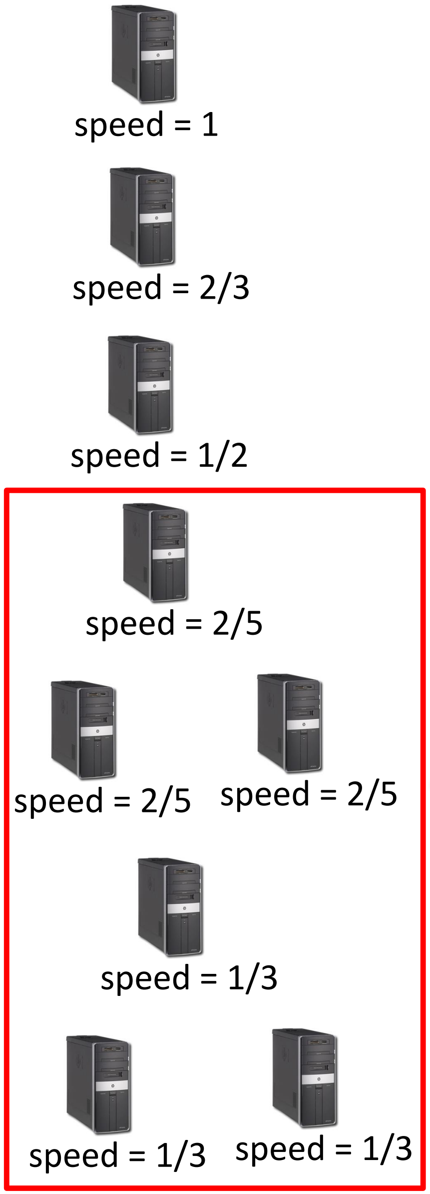
Machine Smoothing

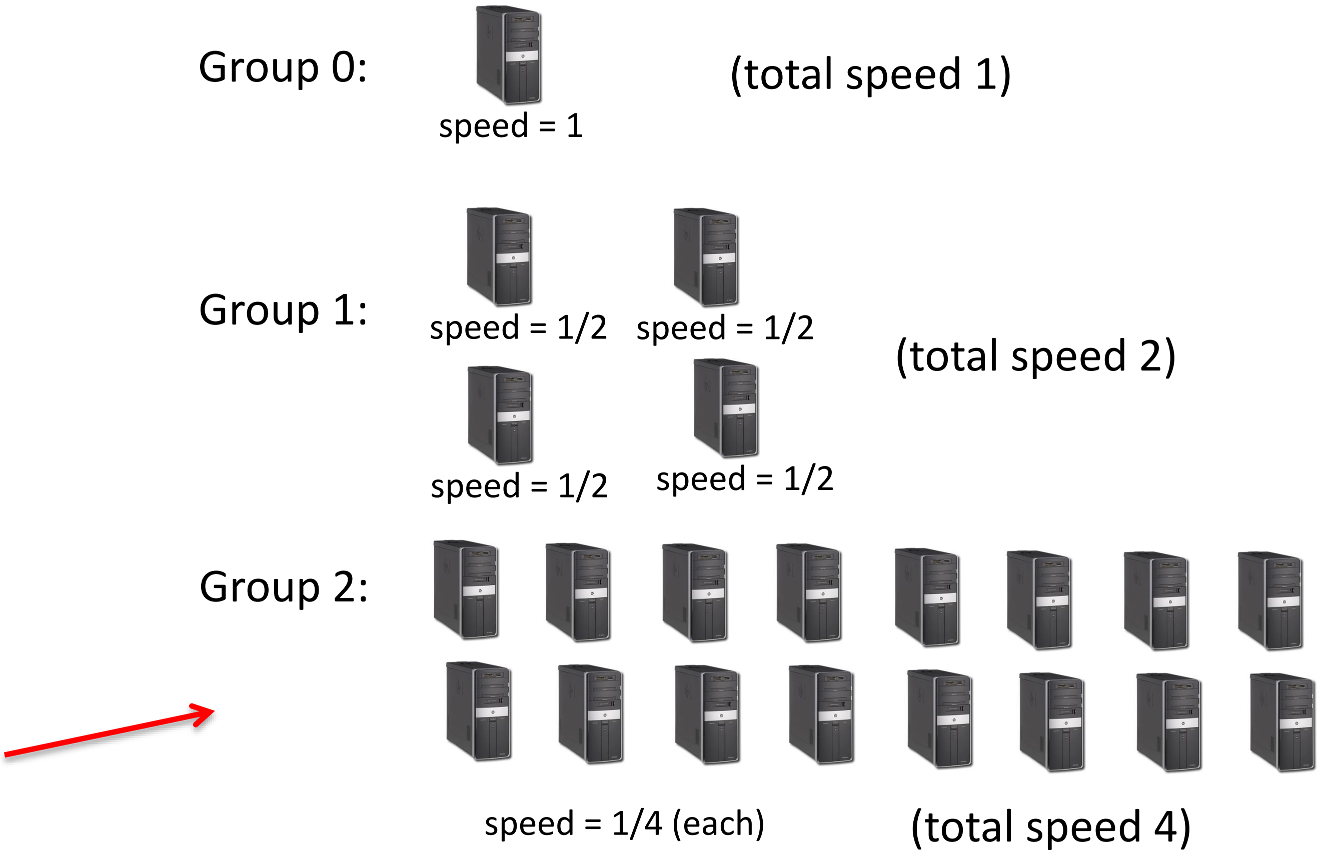




- Group machines so that *total* speed increases exponentially.
- Replace machines with identical machines with (roughly) same total speed.

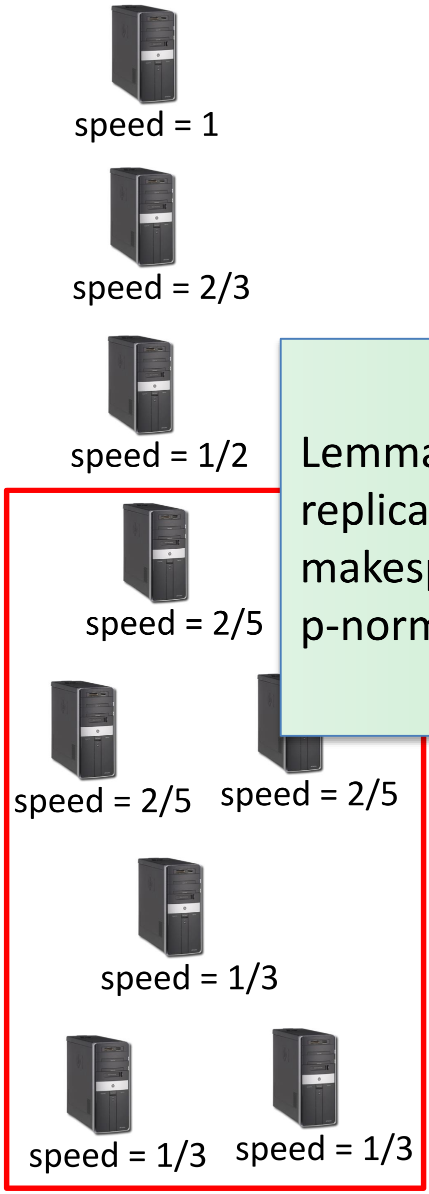
Machine Smoothing

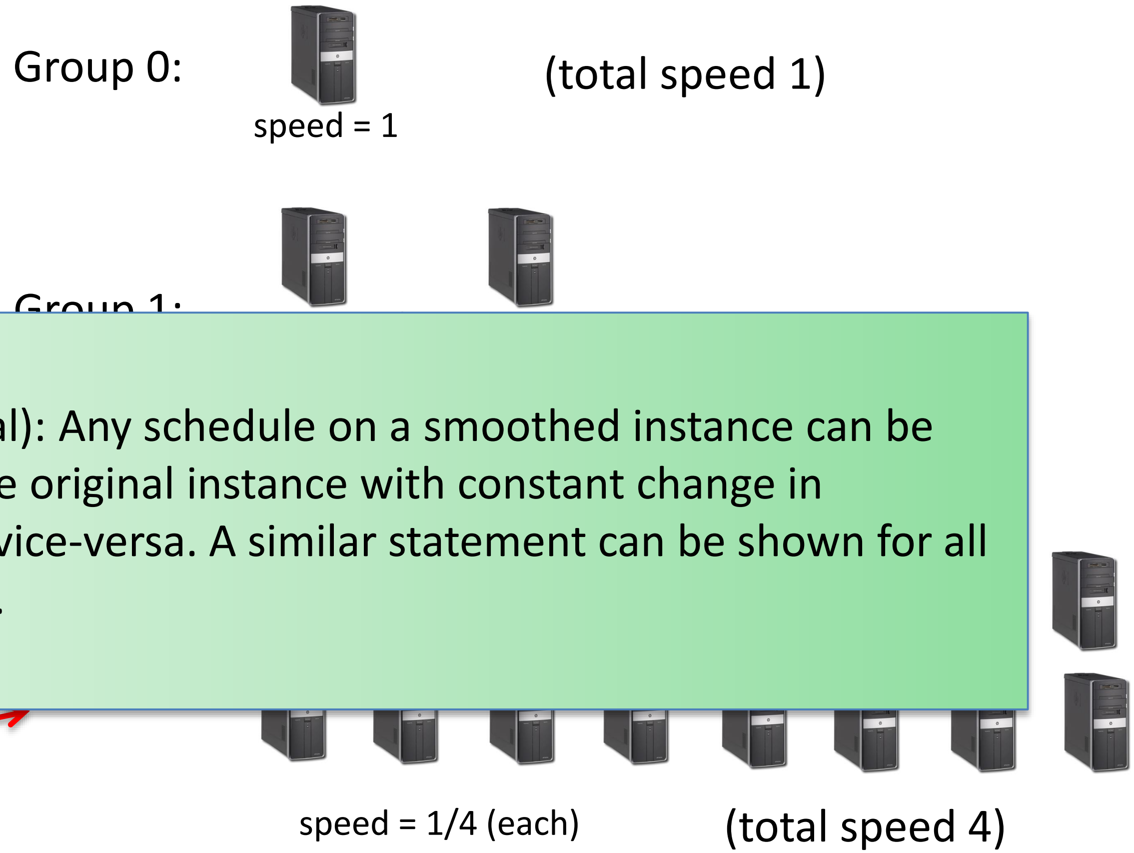




- Group machines so that *total* speed increases exponentially.
- Replace machines with identical machines with (roughly) same total speed.

Machine Smoothing





Lemma (informal): Any schedule on a smoothed instance can be replicated on the original instance with constant change in makespan, and vice-versa. A similar statement can be shown for all p-norms as well.

- Group machines so that *total* speed increases exponentially.
- Replace machines with identical machines with (roughly) same total speed.

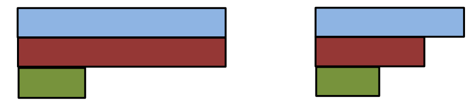
Makespan minimization: Slowest fit on Smoothed Instance

speed = 1

Suppose OPT = 10

Jobs:

1 2



(3, 3, 1) (2, 1.5, 1) . . .

speed = 1/2

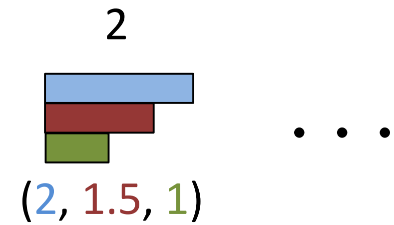
Algorithm: Assign to slowest group such that all execution times are <= c. OPT

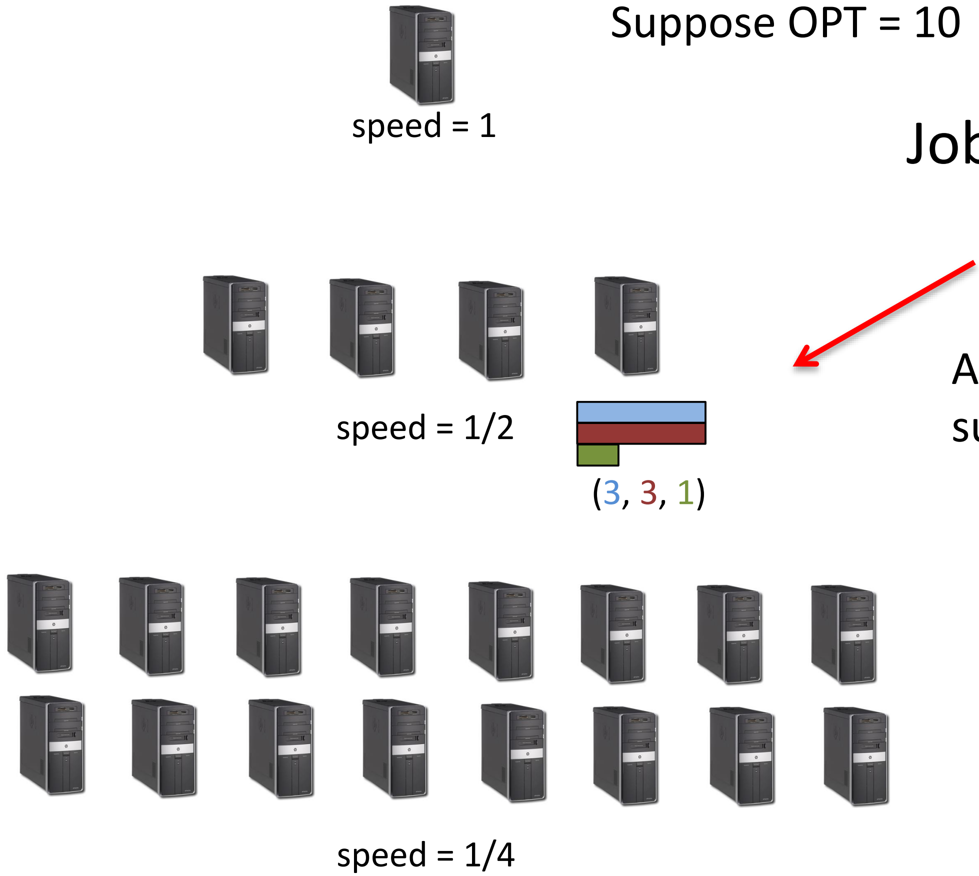
speed = 1/4

Makespan minimization:
Slowest fit on Smoothed Instance

Suppose OPT = 10

Jobs:



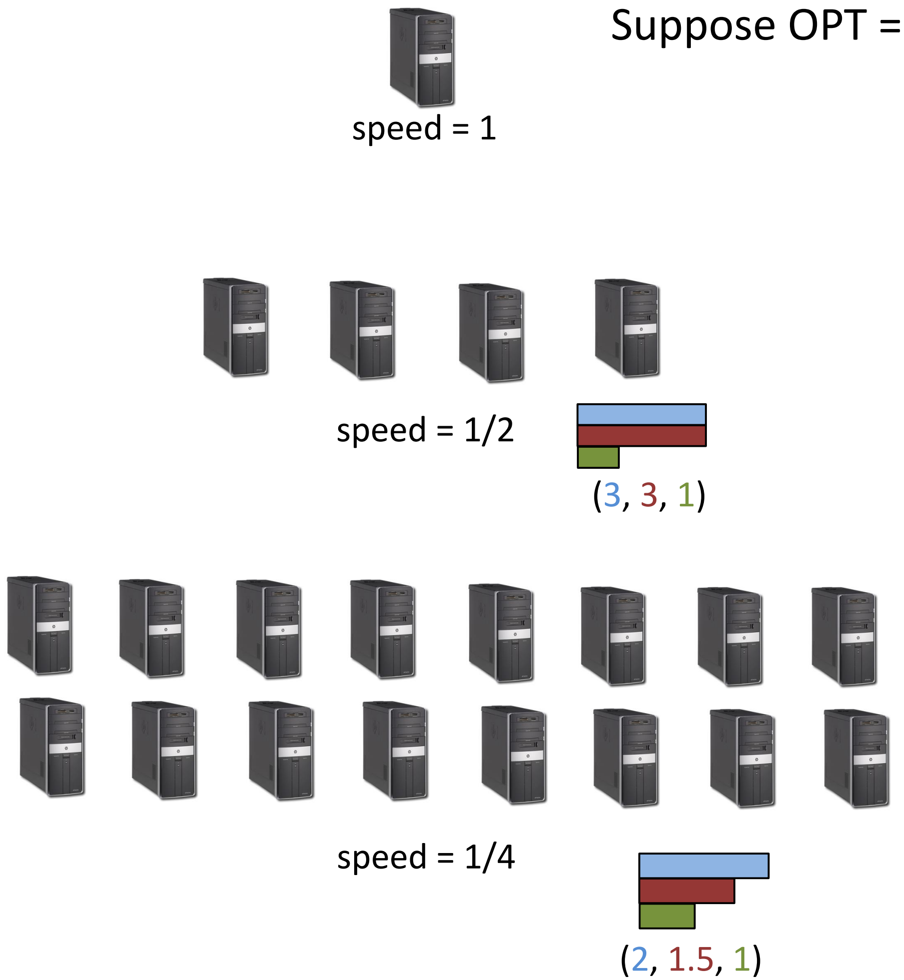


Algorithm: Assign to slowest group such that all execution times are <= c. OPT

Makespan minimization: Slowest fit on Smoothed Instance

Suppose OPT = 10

Jobs: . . .

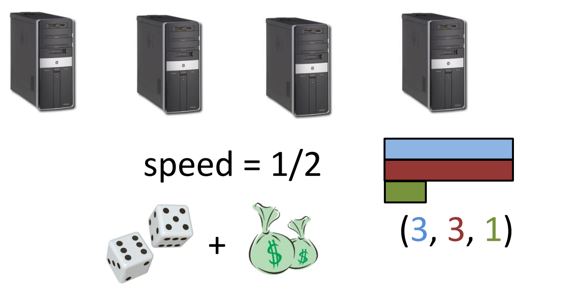


Algorithm: Assign to slowest group such that all execution times are <= c. OPT

Makespan minimization: Slowest fit on Smoothed Instance

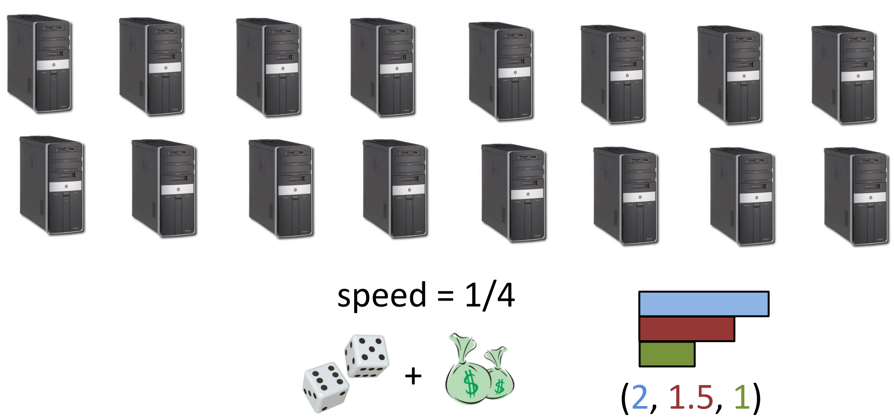
Suppose OPT = 10

Jobs: . . .

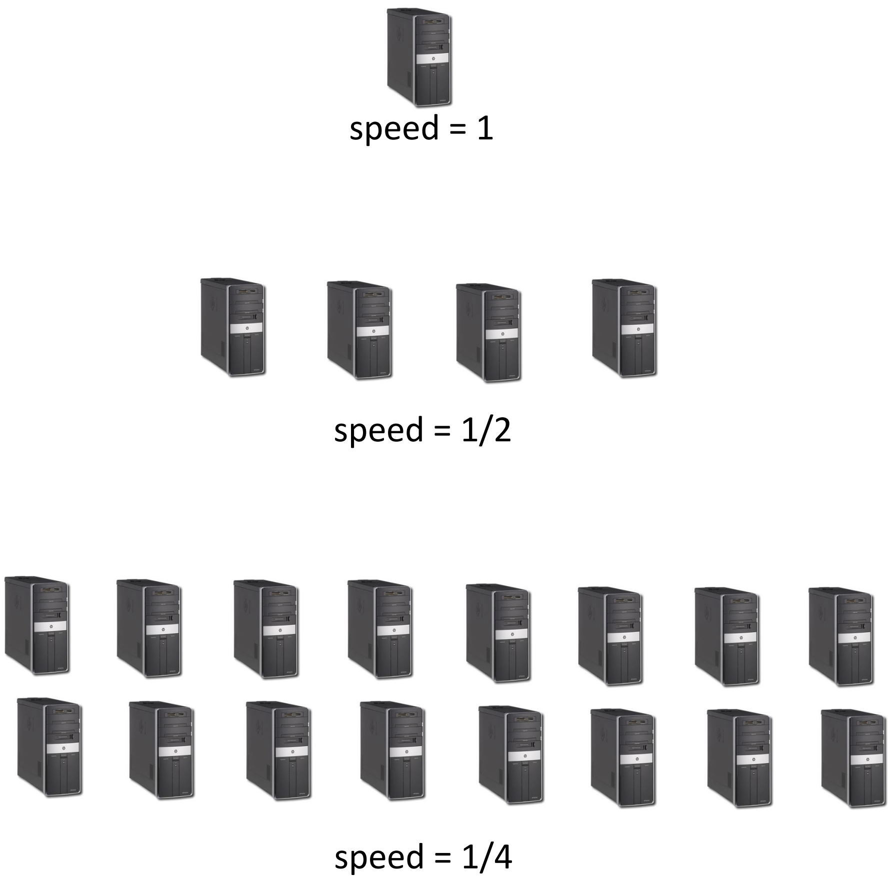


Algorithm: Assign to slowest group such that all processing times are <= c. OPT

…. Then, assign jobs using the identical machines algorithm (within each group).



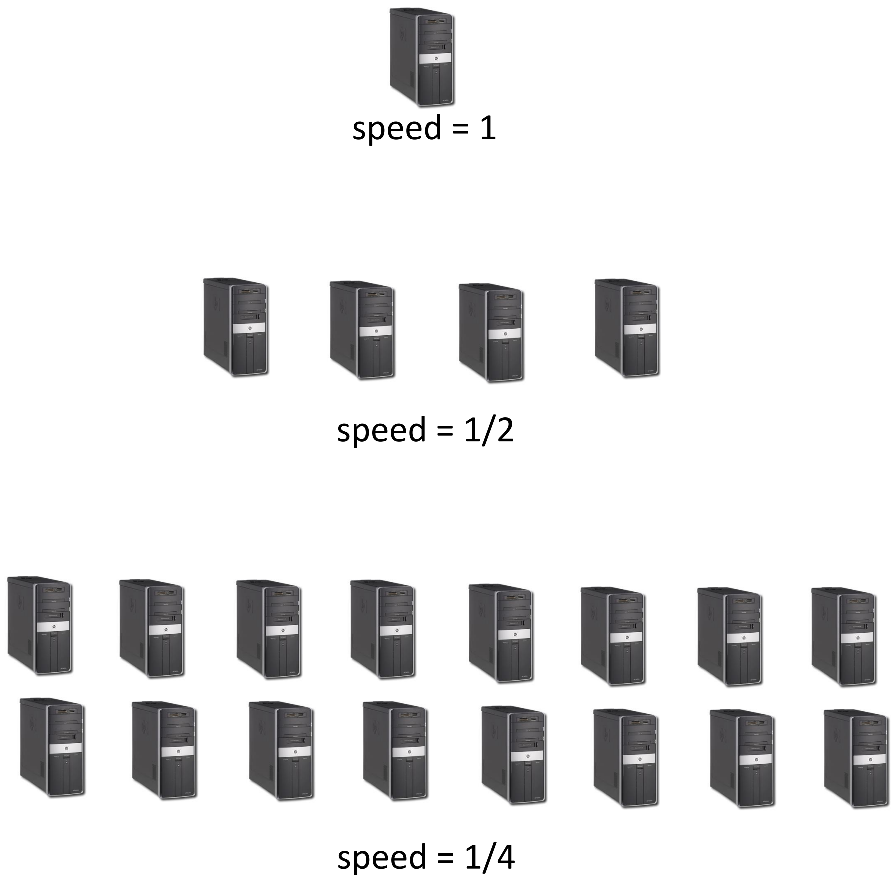
p-norm minimization



Challenge: Even if we are able to guess OPT, how do we divide it among the machine groups?

Indeed, no algorithm previously known even for d = 1

p-norm minimization



Challenge: Even if we are able to guess OPT, how do we divide it among the machine groups?

Indeed, no algorithm previously known even for d = 1

Algorithm has two interleaved stages:

- fractional solution via gradient descent on a potential defined by a suitable fractional relaxation
- online rounding uses a slowest-fit strategy on the fractional solution

Thank You

Questions?