Busy-Time Scheduling: Offline and Online Algorithms

FREDERIC KOEHLER (MIT) AND SAMIR KHULLER (UMD)

WADS 2017

How are cloud services priced?

A: It’s complicated

Amazon EC2 Pricing

Amazon EC2 is free to try. There are four ways to pay for Amazon EC2 instances: On-Demand, Reserved Instances, and Spot Instances. You can also pay for Dedicated Hosts which provide you with EC2 instance capacity on physical servers dedicated for your use.

Launch an Amazon EC2 Instance for Free

Try Amazon EC2 for Free

AWS Free Tier includes 750 hours of Linux and Windows t2.micro instances each month for one year. To stay within the Free Tier, use only EC2 Micro instances.

View AWS Free Tier Details »

On-Demand

With On-Demand instances, you pay for compute capacity by the hour with no long-term commitments or upfront payments. You can increase or decrease your compute capacity depending on the demands of your application and only pay the specified hourly rate for the instances you use.

On-Demand instances are recommended for:

- Users that prefer the low cost and flexibility of Amazon EC2 without any up-front payment or long-term commitment
- Applications with short-term, spiky, or unpredictable workloads that cannot be interrupted
- Applications being developed or tested on Amazon EC2 for the first time

See On-Demand Pricing

Spot Instances

Amazon EC2 Spot instances allow you to bid on spare Amazon EC2 computing capacity for up to 90% off the On-Demand price. Learn More.

Spot instances are recommended for:

- Applications that have flexible start and end times
- Applications that are only feasible at very low compute prices
- Users with urgent computing needs for large amounts of additional capacity

See Spot Pricing

Reserved Instances

Reserved Instances provide you with a significant discount (up to 75%) compared to On-Demand instance pricing. In addition, when Reserved Instances are assigned to a specific Availability Zone, they provide a capacity reservation, giving you additional confidence in your ability to launch instances when you need them.

For applications that have steady state or predictable usage, Reserved Instances can provide significant savings compared to using On-Demand instances. See How to Purchase Reserved Instances for more information.

Reserved Instances are recommended for:

- Applications with steady state usage
- Applications that may require reserved capacity
- Customers that can commit to using EC2 over a 1 or 3 year term to reduce their total computing costs

See Reserved Pricing

Dedicated Hosts

A Dedicated Host is a physical EC2 server dedicated for your use. Dedicated Hosts can help you reduce costs by allowing you to use your existing server-bound software licenses, including Windows Server, SQL Server, and SUSE Linux Enterprise Server (subject to your license terms), and can also help you meet compliance requirements. Learn more.

- Can be purchased On-Demand (hourly).
- Can be purchased as a Reservation for up to 70% off the On-Demand price.

See Dedicated Pricing

Dedicated Hosts Configuration Table

A Dedicated Host is configured to support one instance type at a time. For example, if you allocate a c3.xlarge Dedicated Host, you can use a Dedicated Host with two sockets and 20 physical cores configured to support up to 8 c3.xlarge instances. Refer to the table below for Dedicated Host instance configurations. For more information on instances, visit EC2 Instance Types.

Dedicated Host Attributes			Instance Capacity Per Host by Instance Size								
Instance Type	**Sockets**	**Physical Cores**	**medium**	**large**	**xlarge**	**2xlarge**	**4xlarge**	**8xlarge**	**10xlarge**	**16xlarge**	**32xlarge**
c3	2	20	-	16	8	4	2	1	-	-	-
c4	2	20	-	16	8	4	2	1	-	-	-
p2	2	36	-	-	16	-	-	2	-	1	-
g3	2	36	-	-	-	-	4	2	-	1	-
m3	2	20	32	16	8	4	-	-	-	-	-
d2	2	24	-	-	8	4	2	1	-	-	-

Pricing

On-Demand Pricing

When you pay On-Demand for Dedicated Hosts, you pay for each hour that the Dedicated Host is active in your account (or allocated). You can terminate billing for any particular On-Demand Dedicated Host by releasing it. On-Demand gives you the flexibility to scale up or down without long-term commitments. To learn more about how to allocate or release a Dedicated Host, visit Dedicated Hosts Getting Started.

Region: US East (Ohio)

	Price Per Hour
General Purpose - Current Generation	
m4	$2.42
Compute Optimized - Current Generation	
c4	$1.75
GPU Instances - Current Generation	
p2	$15.84
g3	$5.016
Memory Optimized - Current Generation	
x1	$14.672

Summary

Pay for every hour a machine is on!

- Scheduling idle time = Saving money
- I.e. want to minimize *busy time = total time m/c's are on*

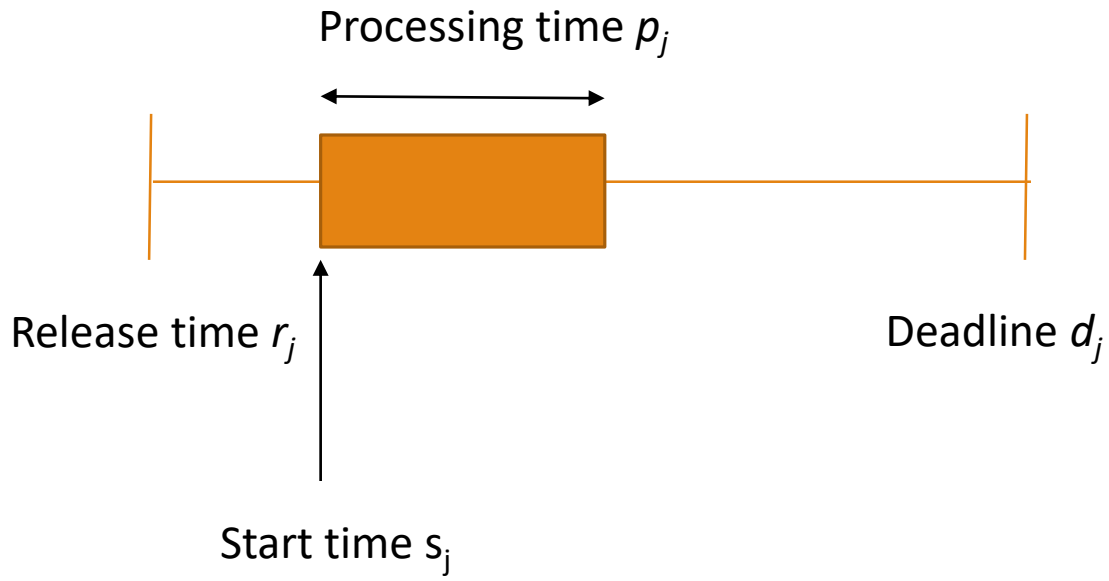
Each machine has *multiple processors*

- Want to make use of all of them!

Busy-Time Scheduling Problem

We suppose we have access to *infinitely many identical machines,* each with *g* processors (cores)

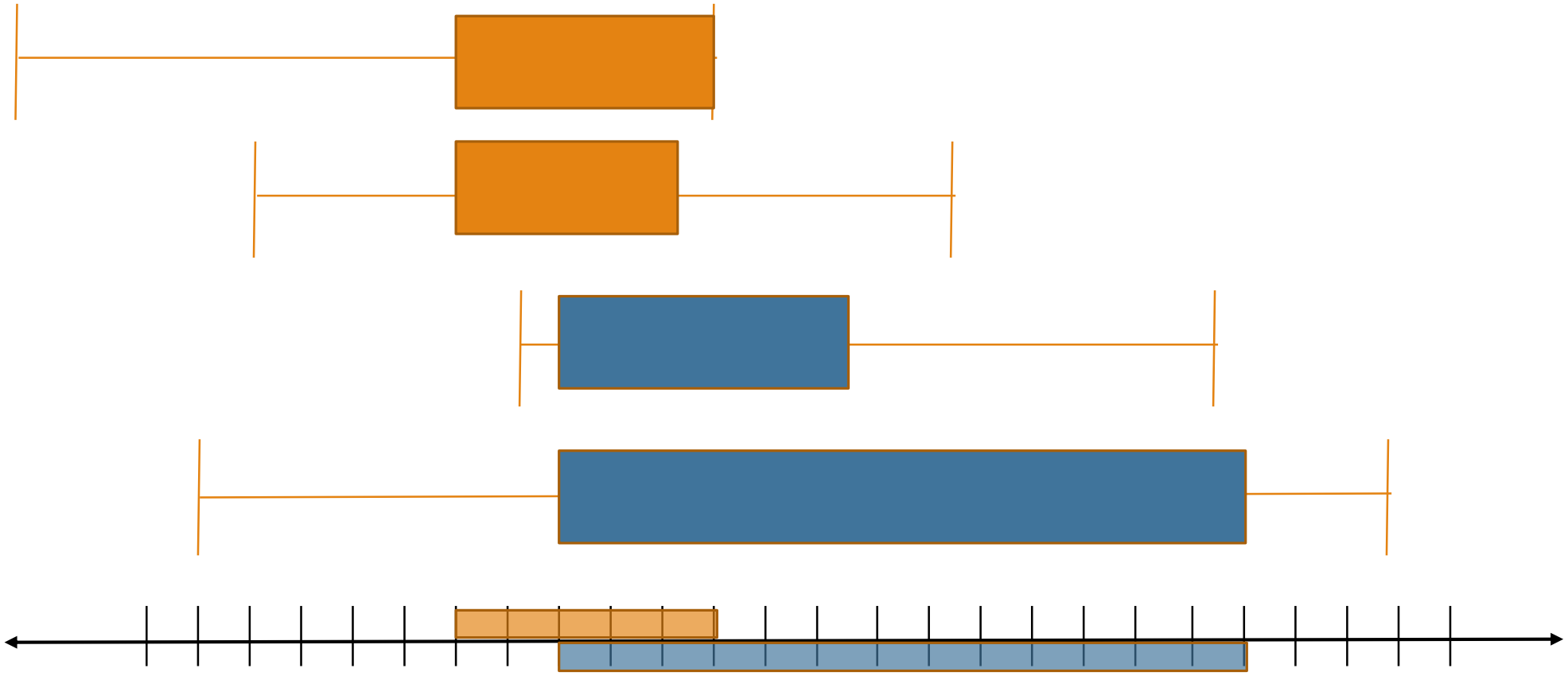
- We will cover finite #machines case later



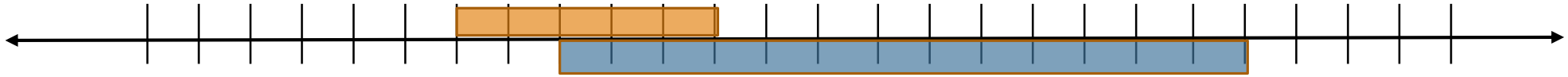
Input: Jobs with Availability Constraints (r_j, d_j, p_j)

Output: Start times s_j, machine assignments m_j

Example (g = 2, 2 machines are used)



Busy time Objective



Busy-time = len ( ) + len( )

= 5 + 13 = 18

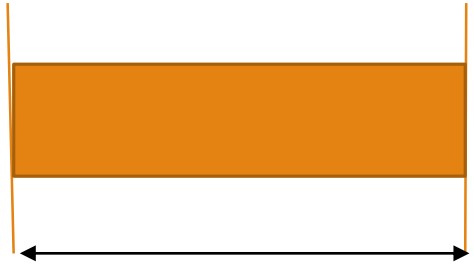
In general: Busy-time = $\sum_{j=1}^{\infty}(total\ time\ machine\ j\ is\ on)$

Algorithms?

Trivially NP-Hard: Use Knapsack

Interval Job Case: ($r_j + p_j = d_j$, *i.e. start times are fixed)*

- Studied in context of fiber optic network design (OADM's)
- Still NP-Hard (Winkler and Zhang '03)
- 2-Approximation (Alicherry and Bhatia '03, Kumar and Rudra '05)
- 4-Approximation (Flammini et al '09)
  - Authors were unaware of previous work
  - Worse approximation but has simple analysis!

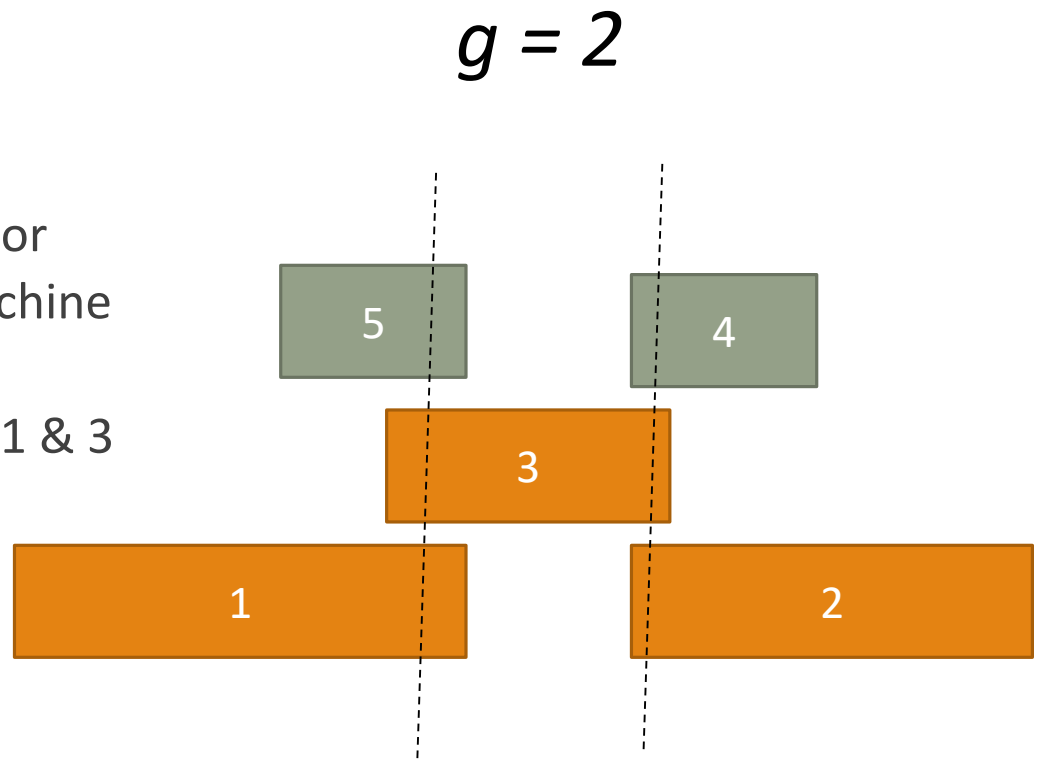


Algorithm of Flammini et al

Simple greedy algorithm:

- Sort jobs longest to shortest
- Start with one machine open
- For each job:
  - Place this job onto the first available processor
  - If no processors available => open a new machine

NOTE: Job 4 can “blame” jobs 2 & 3, 5 blames 1 & 3
For being forced on a new machine
ALSO: Jobs blaming 3 lie in a window
of size 3 p_3



Analysis of Flammini et al

Trivial *load bound:* (sum of p_j)/g ≤ *(*opt busytime)

Every job on machine k has at least g longer jobs to "blame" for it not fitting onto machine k – 1.

So busy-time(machine k) ≤ 3 (sum of p_j on machine (k – 1))/g

Summing,

(total busy-time) ≤ (machine 1 busy-time) + 3(sum of p_j)/g

≤ 4 (opt. busy-time)

General Case (non-Interval jobs)

4-apx by Khandekar et al, ’10. (3-apx by Chang et al ‘14)

If $g = \infty$, can solve optimally by dynamic programming

If $g < \infty$:

- Choose start times as if $g = \infty$, as above
- With start times fixed (!), use algorithm of Flammini et al

(total busy-time) ≤ (machine 1 busy-time) + 3(sum of p_j)/g

≤ 4 (opt. busy-time)

Our Contributions: Online Algorithms

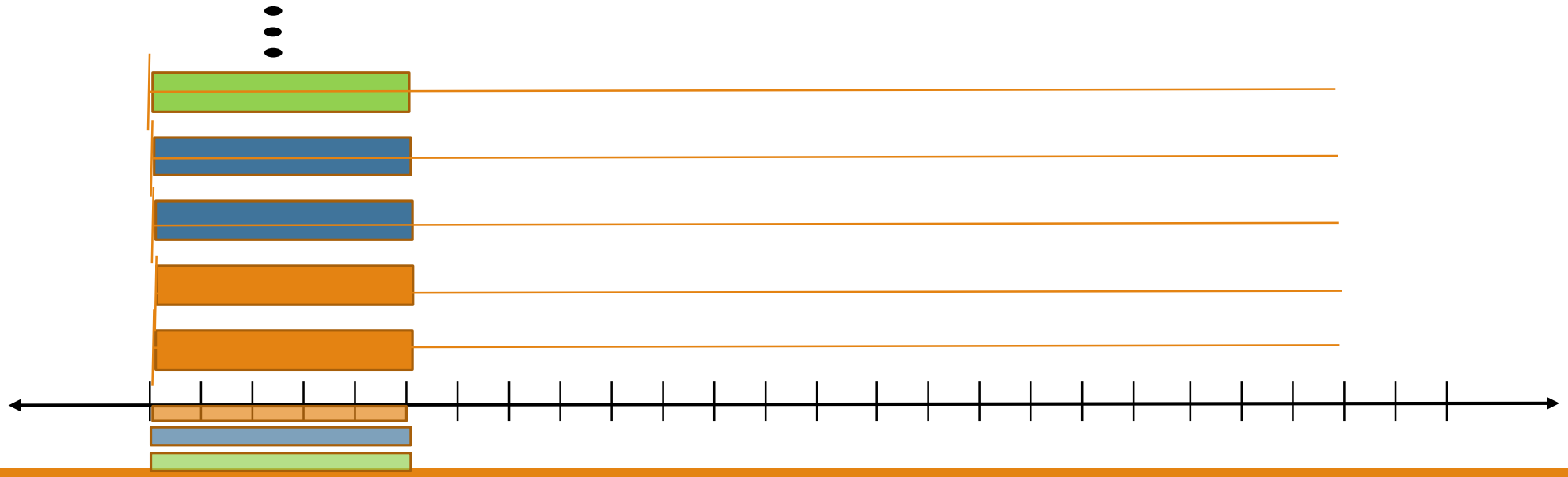
Online Algorithms

- $g = \infty$: 5-competitive online algorithm, 1.618 lower bound
- Ren and Tang (SPAA ‘17) independently gave 6.828-competitive online algorithm
- $g < \infty$: Pick m/c’s via bucketing gives 9 log p_{max}/p_{min} alg
- Much faster than dynamic programming

Our Contributions: Bounded No. of M/C

If allowed to by availability-constraints, previous algorithms schedule all jobs at once!

- This is an issue with “reducing” to g = ∞ problem
- Even Amazon does not allow this --- max 20 instances w/out prior approval
- We give O(1) busy time algorithms with O(M log (max p_j/p_k)) many M/C’s, where M = min # M/C’s needed. Equal p_j => can actually use optimal # of M/C’s!

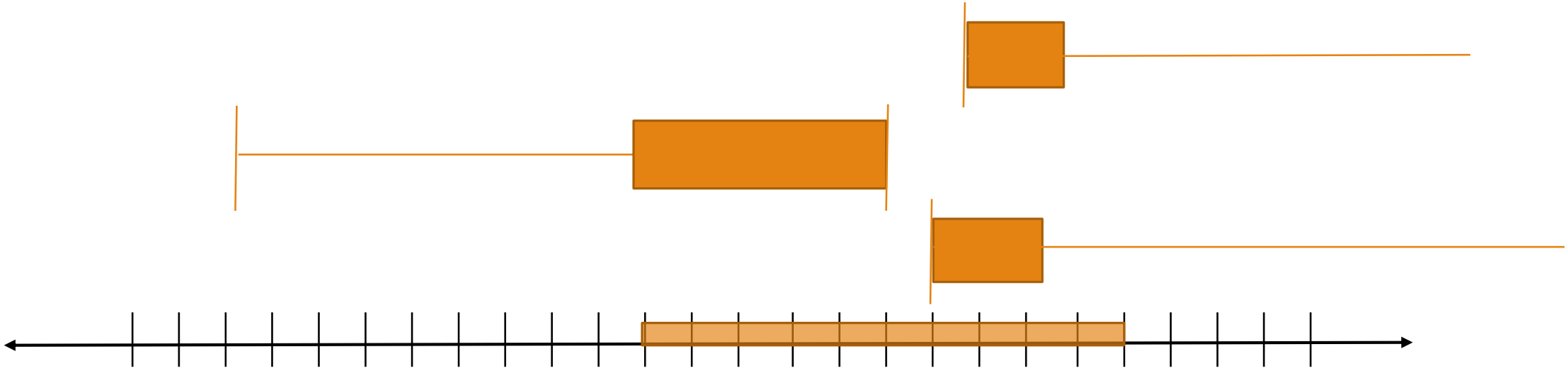


5-Competitive Online Algorithm ($g = \infty$)

Wait until a job *j* hits its *latest start time* $d_j - p_j$

Turn the machine on in the interval $[d_j - p_j, d_j + p_j]$

Run every job that fits while the machine is on



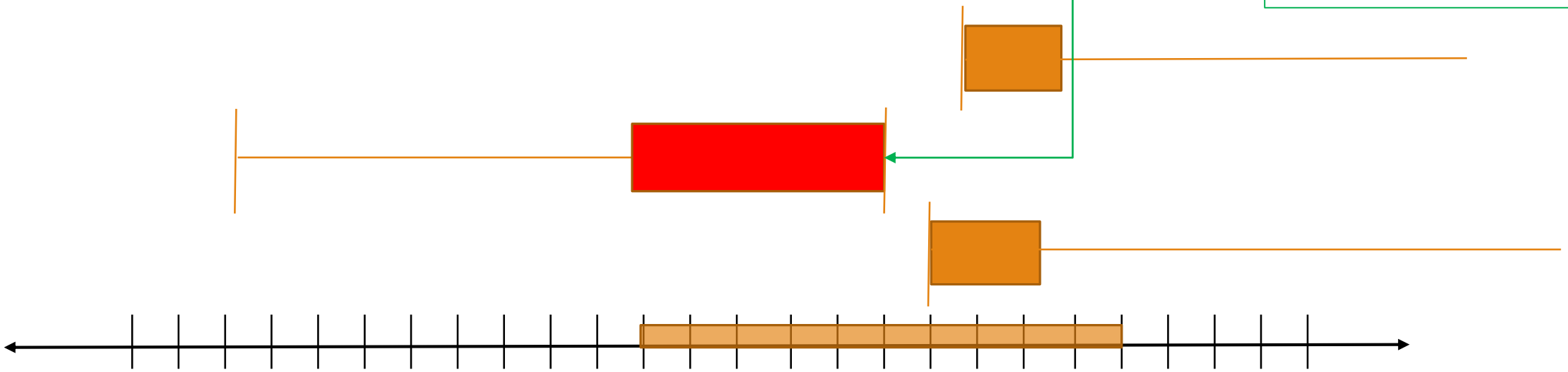
5-Competitive Online Algorithm ($g = \infty$)

Wait until a job *j* hits its *latest start time* $d_j - p_j$

Turn the machine on in the interval $[d_j - p_j, d_j + p_j]$

Run every job that fits while the machine is on

NOTE: We can "charge" the busy-time for the interval to the "primary job" *j* which forced us to run jobs.



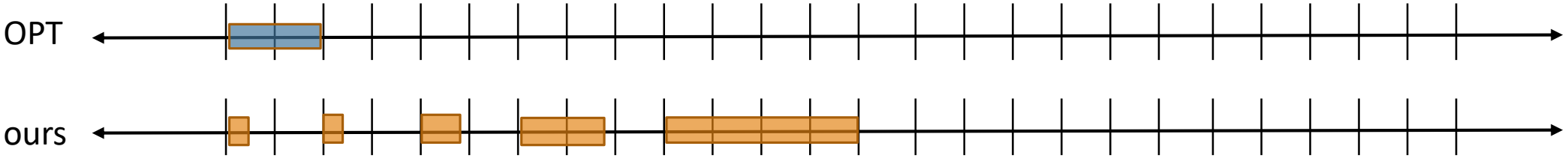
Analysis

Fix an arbitrary busy-time optimal schedule OPT.

The time the machine is turned on (under OPT) decomposes into a disjoint union of intervals:

Fix a single interval *L*. How much time does our schedule use for the jobs in L?

- More precisely, how much time do the *primary jobs* in our schedule from this interval cost us?



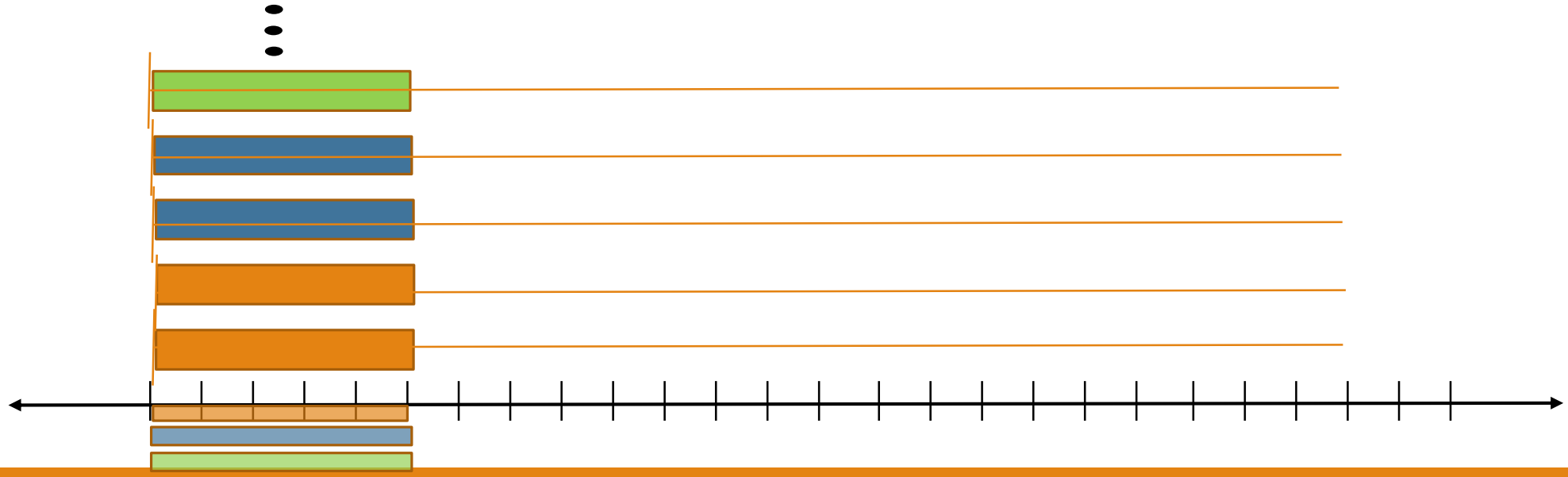
A: At most len(L) + 2 * (1 + ½ + ¼ + …) * len(longest job in L) ≤ 5 len(L)

Summing gives 5-approximation.

Recall: Bounded No. of M/C

If allowed to by availability-constraints, previous algorithms schedule all jobs at once!

- This is an issue with “reducing” to g = ∞ problem
- Even Amazon does not allow this --- max 20 instances w/out prior approval
- We give O(1) busy time algorithms with O(M log (max p_j/p_k)) many M/C’s, where M = min # M/C’s needed. Equal p_j => can actually use optimal # of M/C’s!



(3 + ε)-Approx on Bounded No. of M/C

Observation: Batch similar size jobs together
=> good busytime

Algorithm:

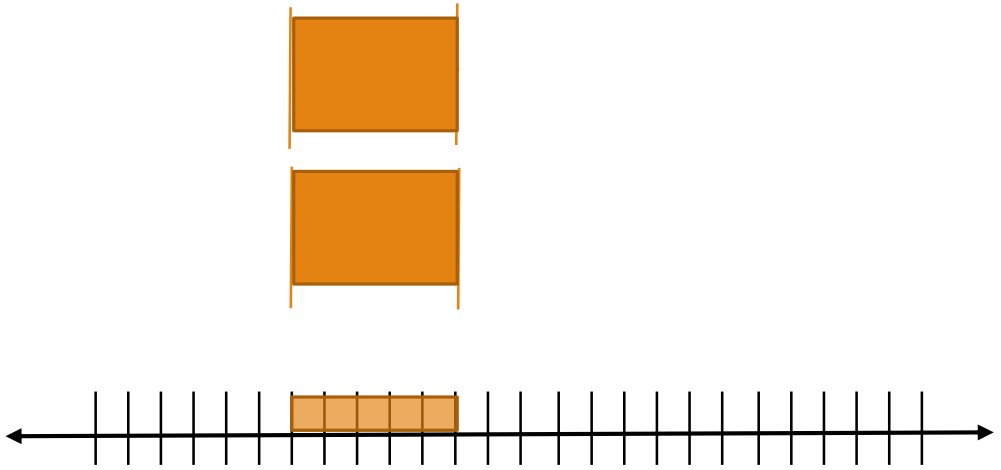
- Bucket jobs according to processing time p_j; round r_j and d_j
  - *I.e.* bucket [1-2], [2-4], [4-8] etc. and round to multiples of 2
- In each bucket:
  - Solve the equal-length scheduling problem to find min # of m/c's
  - Every time a multiple of g many jobs are available, run them together
- Some jobs remain unscheduled!
  - Run them with 3-apx of Chang et al
  - Need to bound # of M/C used in this step. (Nontrivial part!)

Proof Sketch: Busy-time is bounded

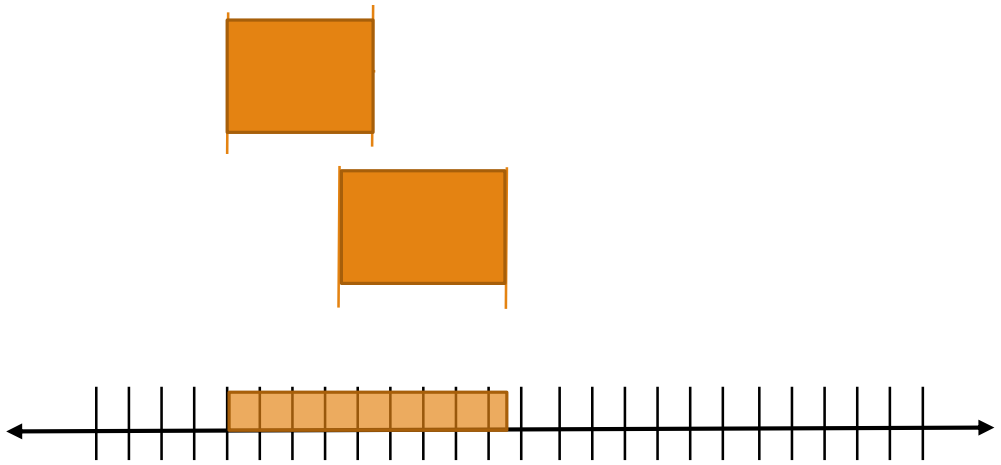
In one bucket, jobs length lie in [p/α, p] for some p

- CLAIM: Busytime is bounded by 2 α * (load bound)
- Sum over buckets, and with bound of Chang et al => (2 α + 1)-approximation

Rounded Time



Real Time (unrounded)



Proof Sketch: Number of M/C Bounded

Number of machines used by each bucket?

- We solve m/c minimization optimally in each bucket with *rounded* r_j, p_j, d_j
- If we are careful in unrounding, get O(α)-approximation

Number of machines used by calling Chang et al?

- Idea: Jobs dropped from a bucket have essentially disjoint availability intervals
- Show that even taking *arbitrary* valid assignment of start times, number of m/c we need is bounded by true optimum

Summary

5-approximation for ($g = \infty$) busy-time scheduling
- Best known; lower bound of $\varphi \approx 1.618$
- What is the true competitive ratio?

$(3 + \varepsilon)$-approximation with only $O(M \max \log p_j/p_k)$ M/C's
- M = minimum number of machines required to run all jobs

In paper: 6-approximation for busy-time on *optimal* # of M/C's
- Only when all processing times are equal.
- *Trade-off lower bounds:* similar result *cannot* hold in general!

Also in paper: online with lookahead by "fusing" algorithms