How to Plan Ahead

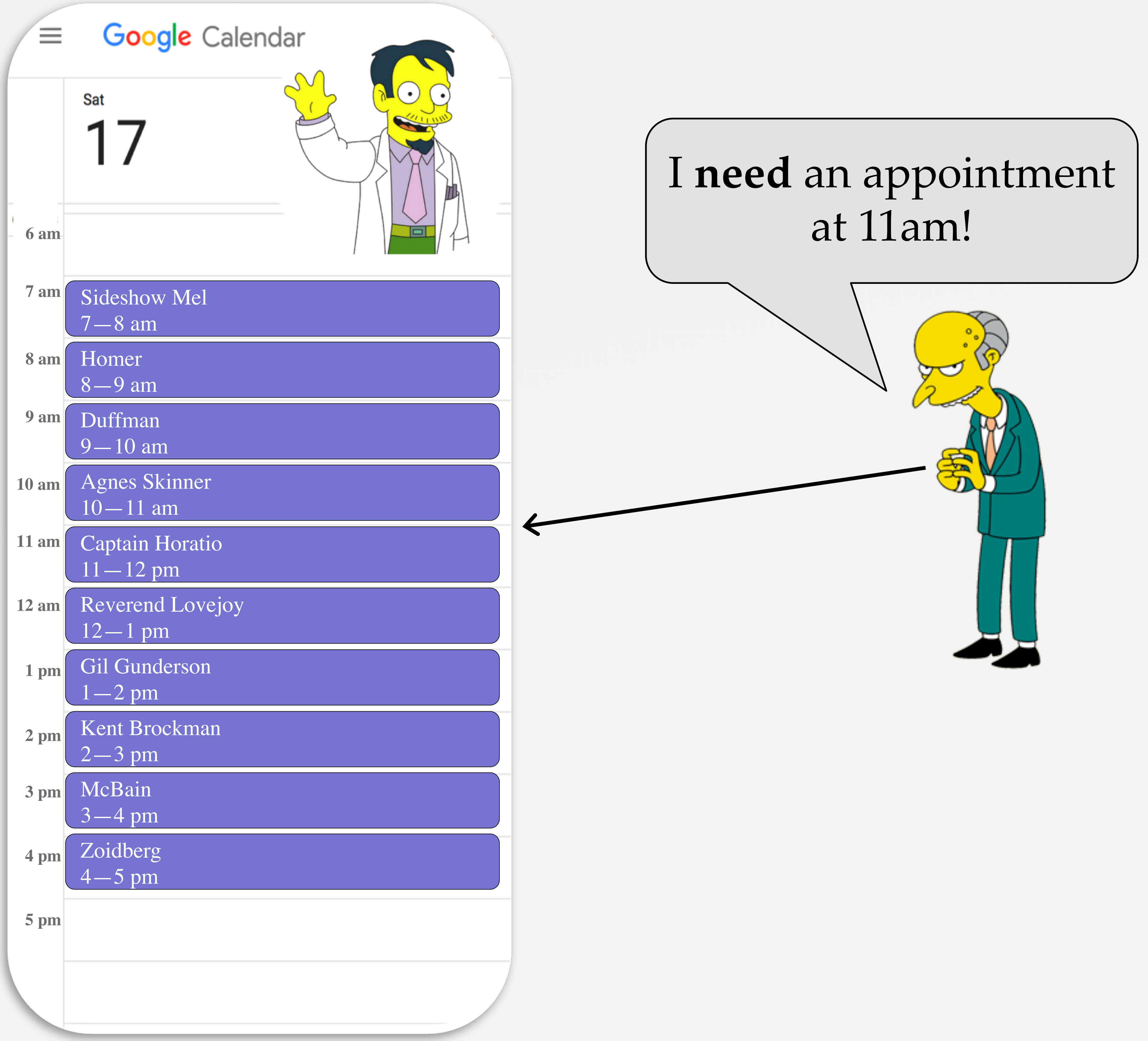
Seth Gilbert

Joint work with:

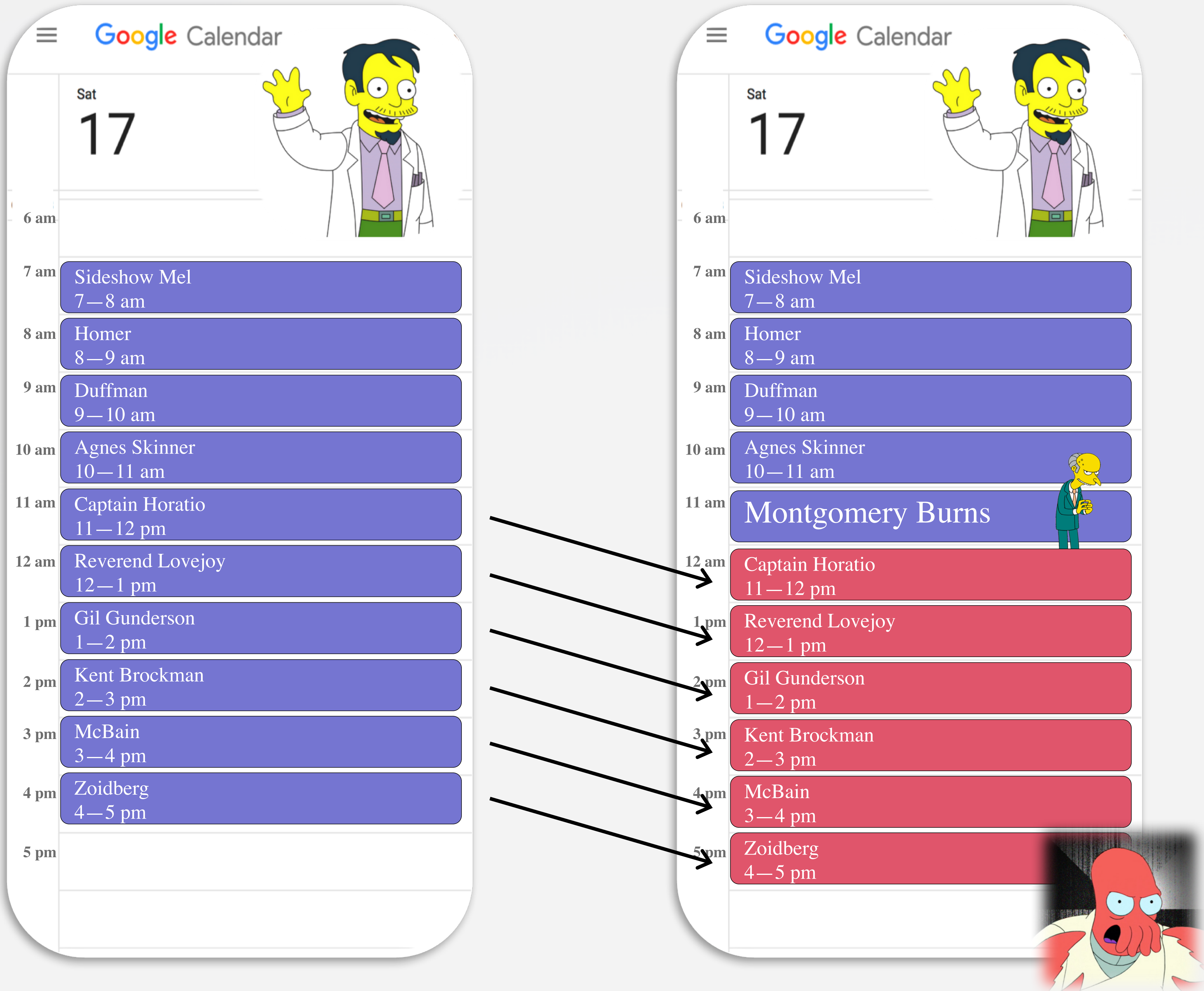
Michael Bender, Martin Farach-Colton, Sandor Fekete, Jeremy Fineman, Shunhao Oh


Plans change.
Be prepared.

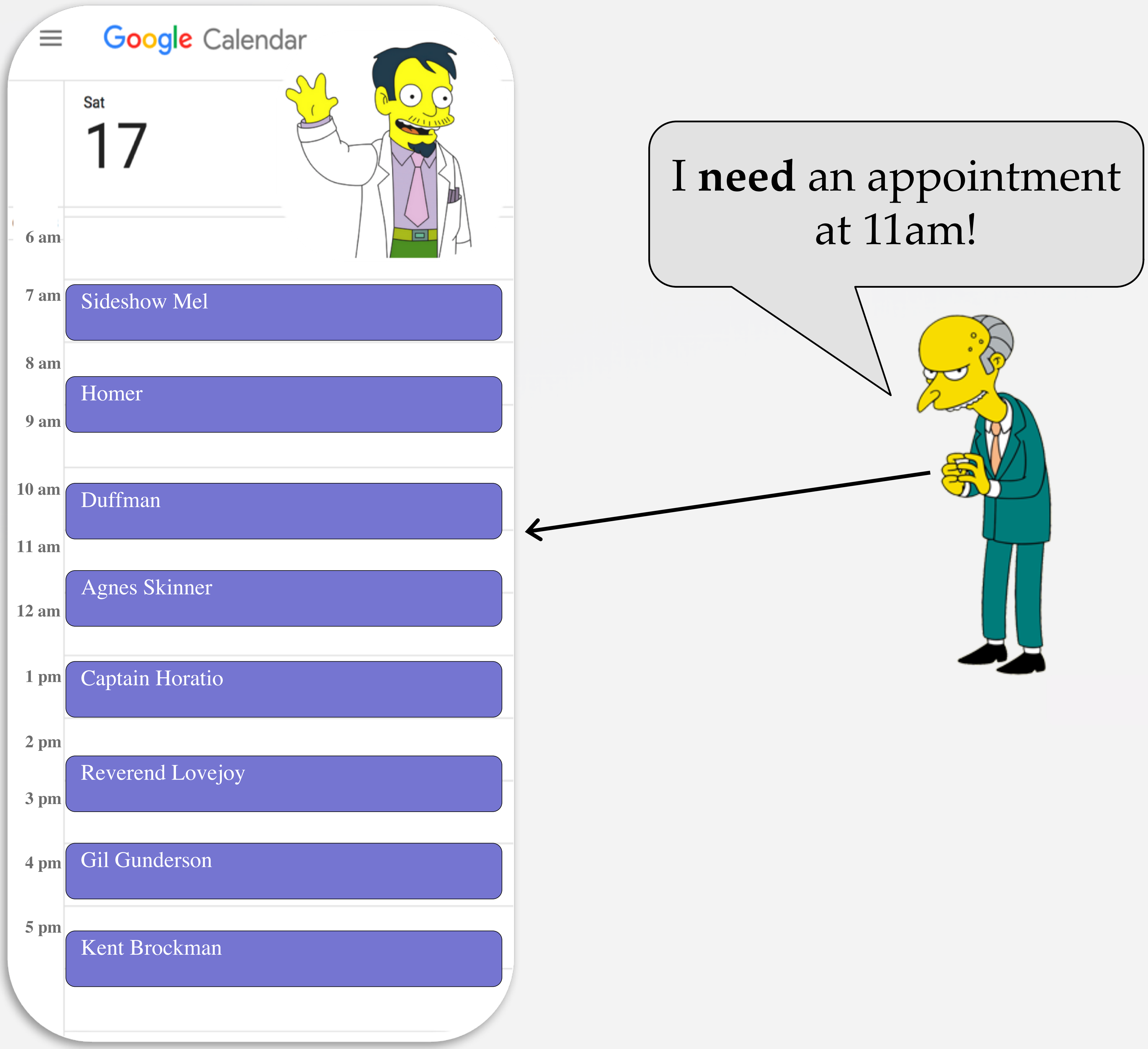
Scheduling Appointments



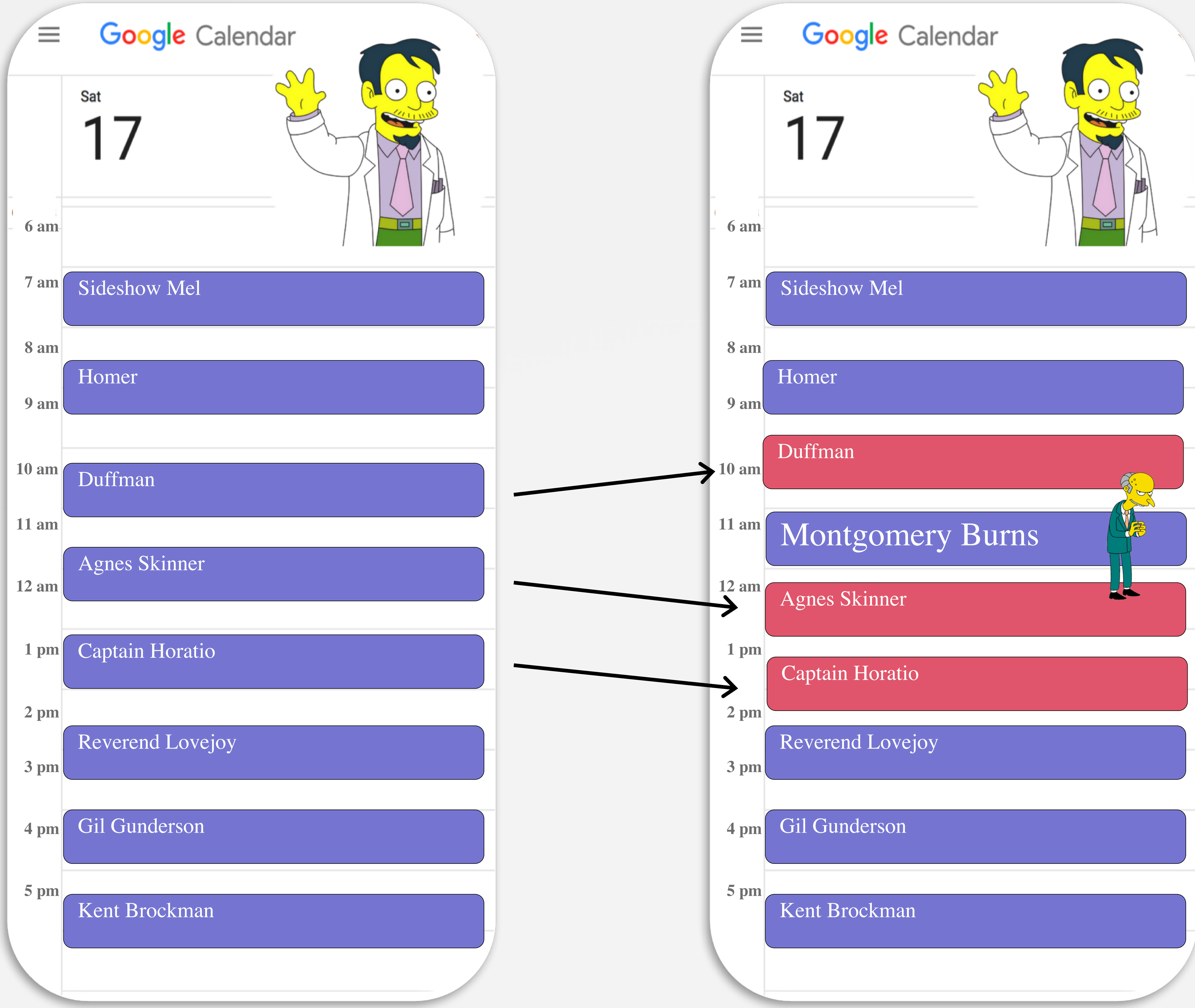
Scheduling Appointments



Scheduling Appointments



Scheduling Appointments



Which schedule is better?

¯_(ツ)_/¯

More efficient
(More patients scheduled)

More flexible
(Less disruption on changes)

Big Picture Goal:

How do you maintain a *near optimal* schedule with *minimal reallocation cost* when:

- Jobs are <u>added</u> and <u>removed</u>.
- Existing jobs can be reallocated <u>at some cost</u>.

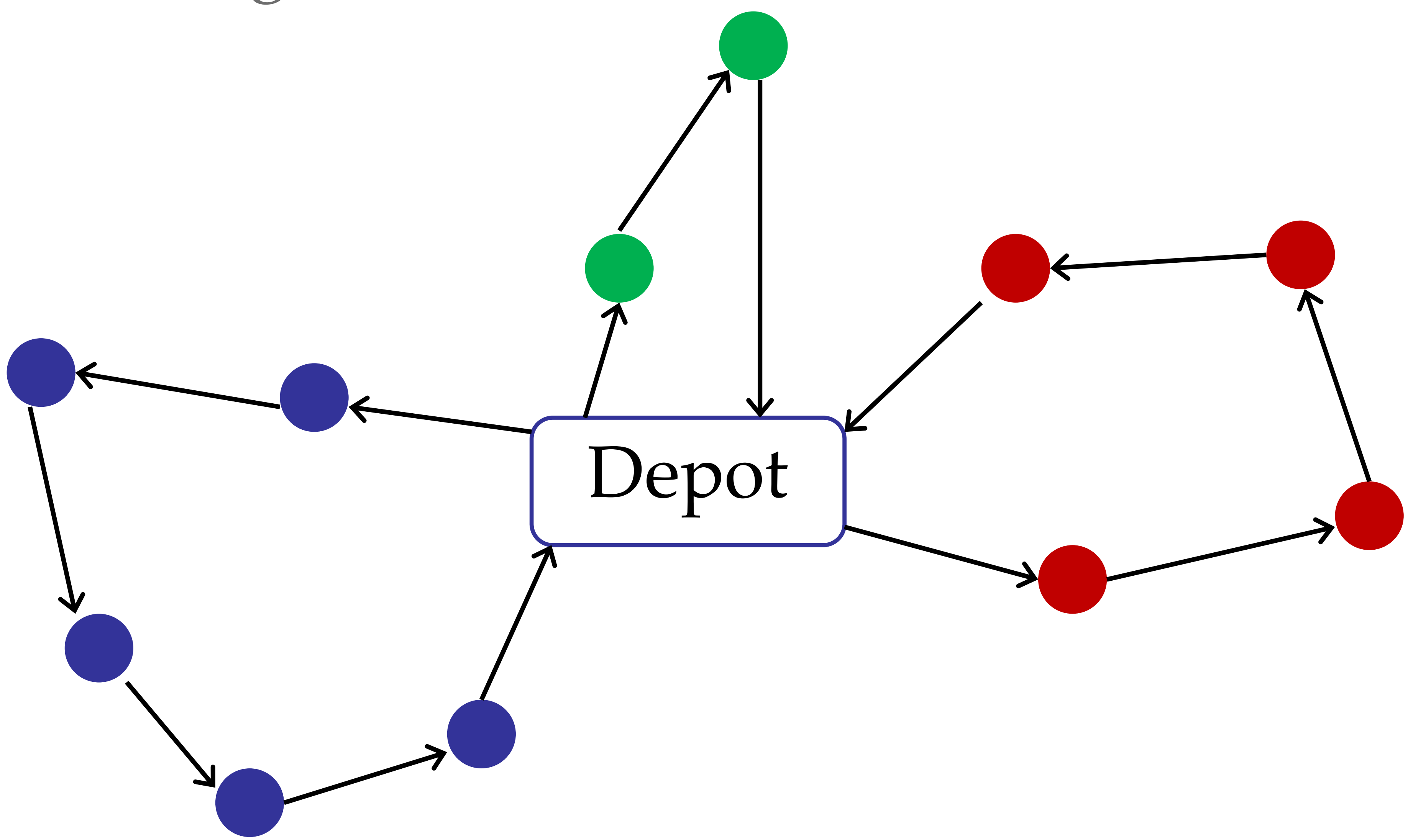
Secondary Story:

How do you design a *data structure* to efficiently maintain a set of elements:

- Elements are added and removed.
- Elements are *spread out* in the structure.

Many other problems:

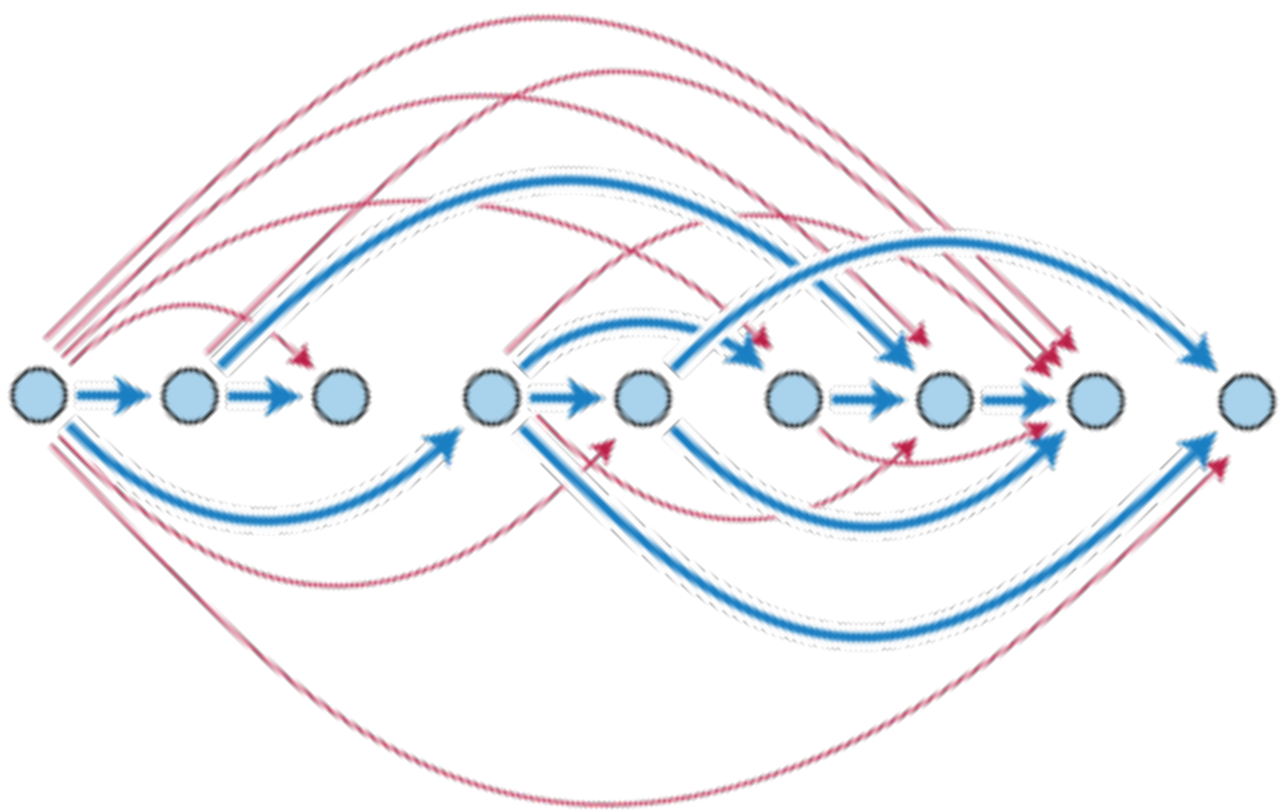
Scheduling Ikea deliveries:



Minimize disruption caused by new deliveries / cancellations.

Many other problems:

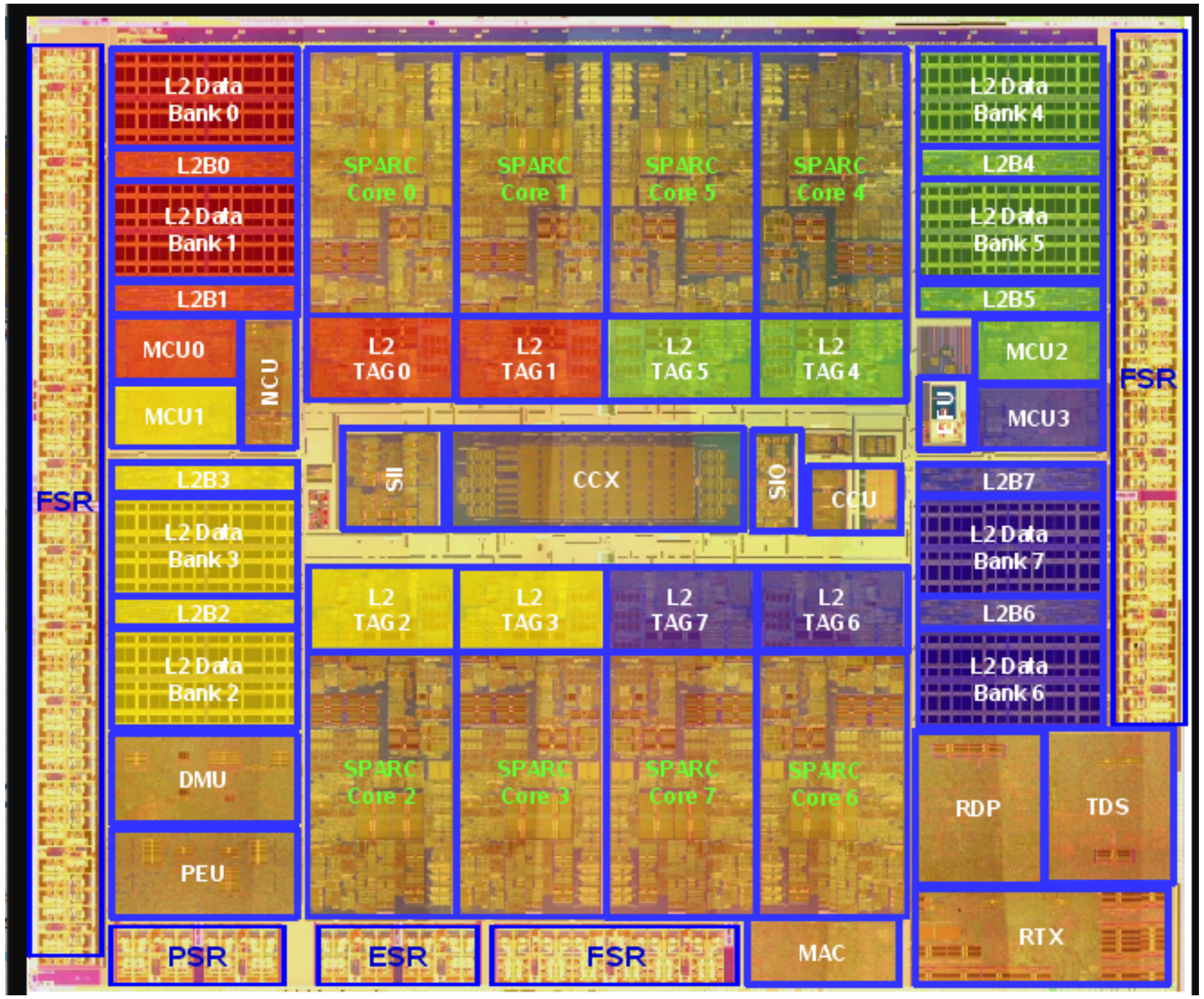
Scheduling an assembly line:



Minimize disruption caused by future changes.

Many other problems:

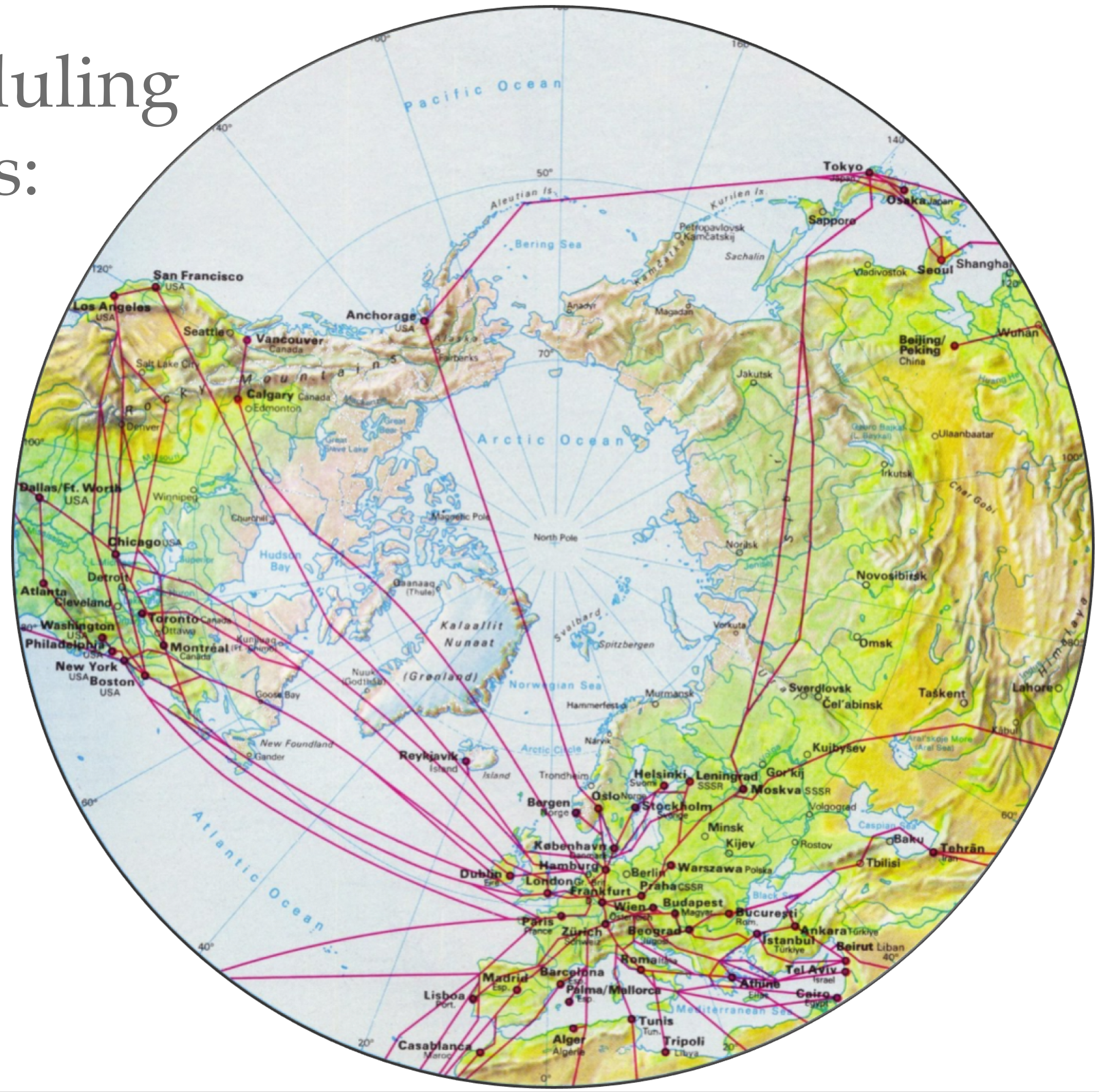
Scheduling blocks on an FPGA:



Minimize disruption caused by changing functional units.

Many other problems:

Scheduling flights:



Which schedule is better?

¯_(ツ)_/¯

More efficient
(More patients scheduled)

More flexible
(Less disruption on changes)

How to Plan Ahead
A Play in Three Acts

Act I : A Few Reservations

Wherein we schedule simple (unit-length) tasks with arrival times and deadlines.

Act II : One Algorithm to Rule Them All

Wherein we discover a (cost-oblivious) champion able to defeat any (subadditive) reallocation cost. Our champion knows how to minimize the makespan, but no more.

Act III : Data Structures to the Rescue!

Wherein our champion seeks aid from the faraway Land of Data Structures in order to minimize the sum-of-completion-times dragon.

Related Work

Many related approaches:

Load balancing / Server scheduling:

- **M. Andrews, M. X. Goemans, and L. Zhang**. Improved bounds for on-line load balancing. Algorithmica, 23(4):278–301, 1999.
- **P. Sanders, N. Sivadasan, and M. Skutella**. Online scheduling with bounded migration. Math. Oper. Res., 34(2):481–498, 2009.
- **M. Skutella and J. Verschae**. A robust PTAS for machine covering and packing. In Proc. ESA, pages 36–47, 2010.
- **J. C. Verschae**. The Power of Recourse in Online Optimization: Robust Solutions for Scheduling, Matroid and MST Problems. PhD thesis, TU Berlin, June 2012.
- **J. Westbrook**. Load balancing for response time. J. of Alg., 35(1):1 – 16, 2000.

Migration: reallocate a fraction of tasks to other machines in order to maintain a good schedule.

Many related approaches:

Reoptimization:

- **G. Baram and T. Tamir**. Reoptimization of the minimum total flow-time scheduling problem. In Proc. MedAlg, volume 7659 of LLNCS, pages 52–66, 2012.
- **C. Archetti, L. Bertazzi, and M. G. Speranza**. Reoptimizing the Traveling Salesman Problem. Networks, 42(3):154–159, 2003.
- **G. Ausiello, B. Escoffier, J. Monnet, V. Paschos**. Reoptimization of minimum and maximum traveling salesman's tours. Journal of Discrete Algorithms 7:4, 2009.
- **H. Shachnai, G. Tamir, and T. Tamir.** A theory and algorithms for combinatorial reoptimization. In Proc. LATIN, pages 618–630, 2012.

Given an optimal solution for an input, compute a near-optimal solution for a closely related input

Many related approaches:

Other reallocation / rescheduling:

- **Gupta, Kumar, and Stein.** Maintaining assignments online: matching, scheduling, flows. In SODA 2014.
- **S. Davis, J. Edmonds, and R. Impagliazzo.** Online algorithms to minimize resource reallocations and network communication. In Proc. APPROX-RANDOM, pages 104–115, 2006.
- **L. Epstein and A. Levin.** A robust APTAS for the classical bin packing problem. In Proc. ICALP, pages 214–225, 2006.
- **N. G. Hall and C. N. Potts.** Rescheduling for new orders. Op. Res., 52(3), 2004.
- **A. T. Unal, R. Uzsoy, and A. S. Kiran.** Rescheduling on a single machine with part-type dependent setup times and deadlines. Ann. Op. Res., 70, 1997.

Many related approaches:

Transportation:

- **A. Caprara, L. Galli, L. Kroon, G. Maroti, and P. Toth**. Robust train routing and online re-scheduling. In Proc. ATMOS, pages 24–33, 2010.
- **V. Chiraphadhanakul and C. Barnhart.** Robust flight schedules through slack re-allocation. EURO Journal on Transportation and Logistics, 2(4):277–306, 2013.
- **H. Jiang and C. Barnhart.** Dynamic airline scheduling. Transp. Sc., 43(3):336–354, 2009.

How to Plan Ahead
A Play in Three Acts

Act I : A Few Reservations

Wherein we schedule simple (unit-length) tasks with arrival times and deadlines.

Act II : One Algorithm to Rule Them All

Wherein we discover a (cost-oblivious) champion able to defeat any (subadditive) reallocation cost. Our champion knows how to minimize the makespan, but no more.

Act III : Data Structures to the Rescue!

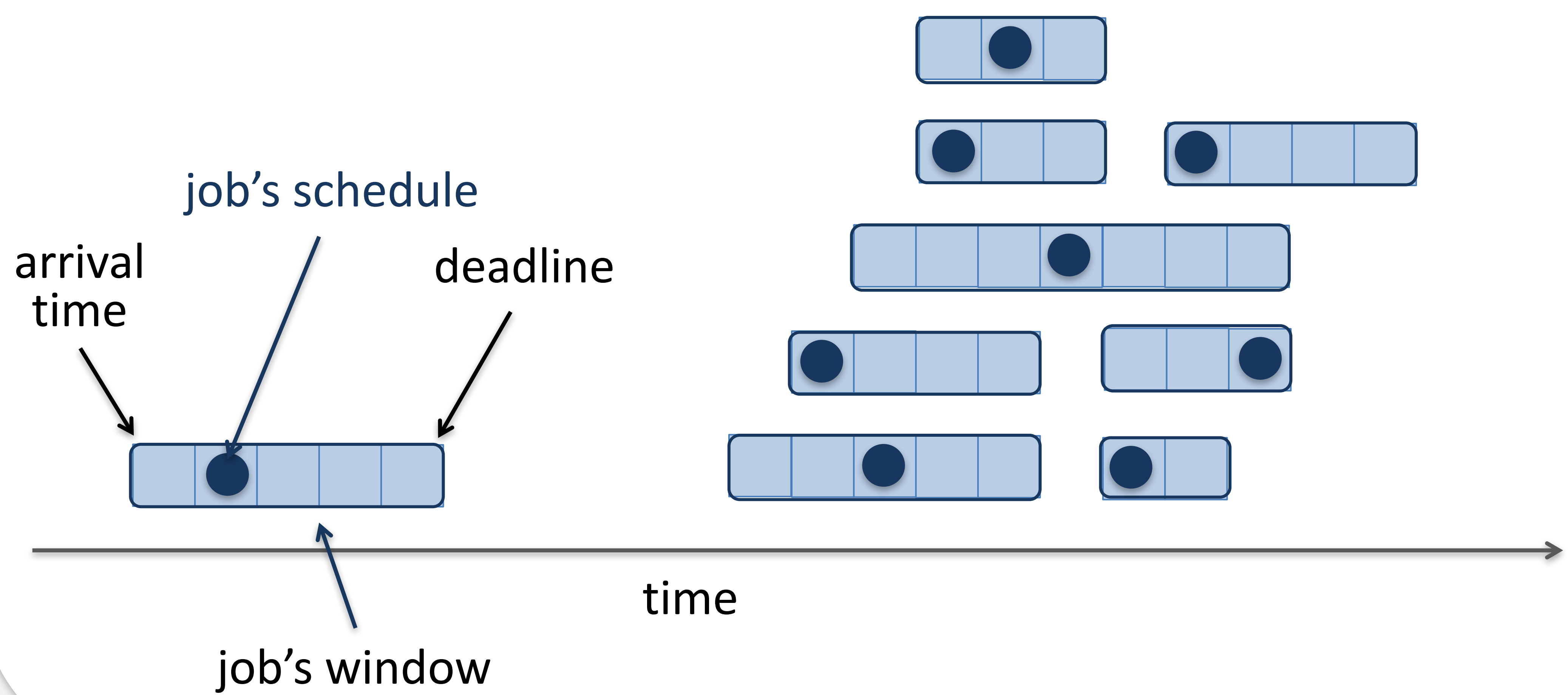
Wherein our champion seeks aid from the faraway Land of Data Structures in order to minimize the sum-of-completion-times dragon.

“It does not do to leave a live dragon out of your calculations, if you live near him.” (Tolkien)

Scheduling Simple Tasks

Today: 1 server

Unit-length tasks:



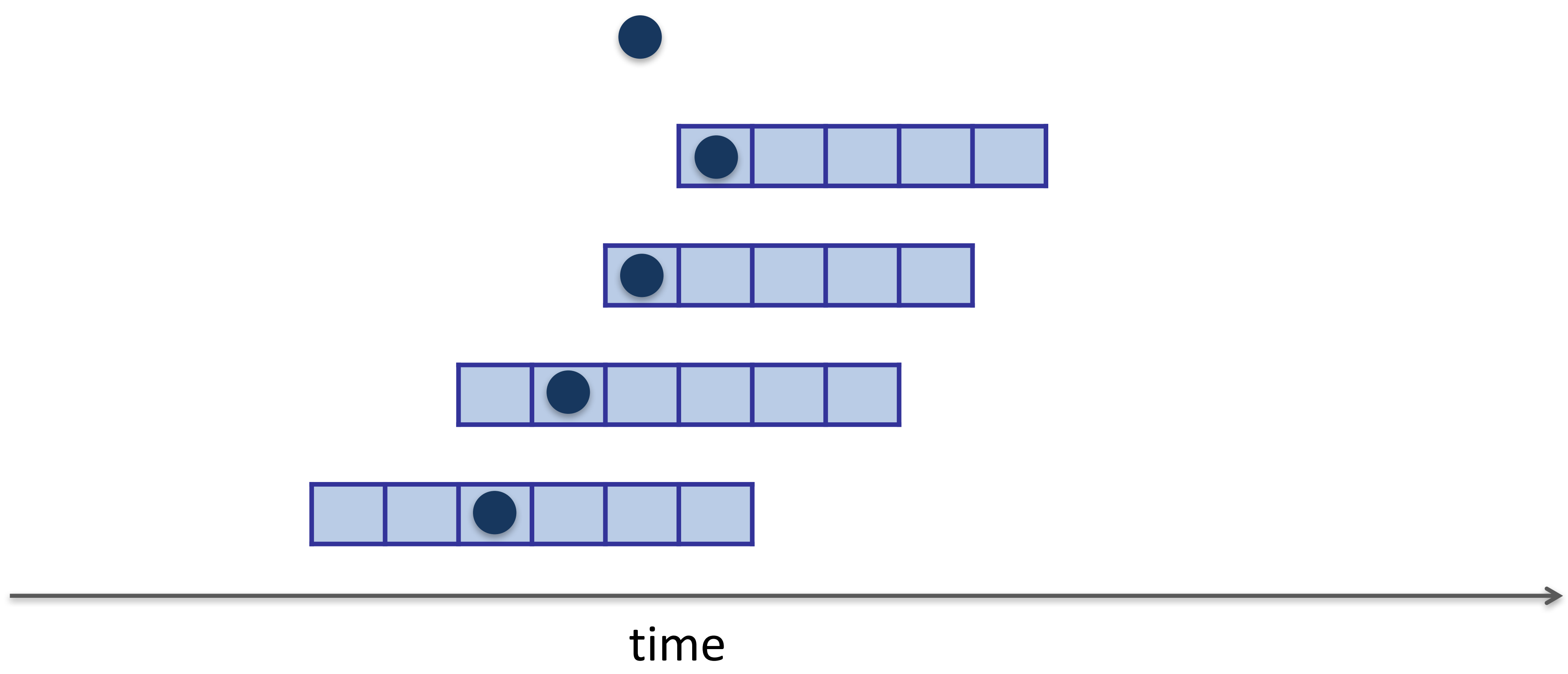
Easy offline solution: Earliest deadline first.

“It does not do to leave a live dragon out of your calculations, if you live near him.” (Tolkien)

The Cost of Reallocation

Online with reallocation:

Cost = # items moves

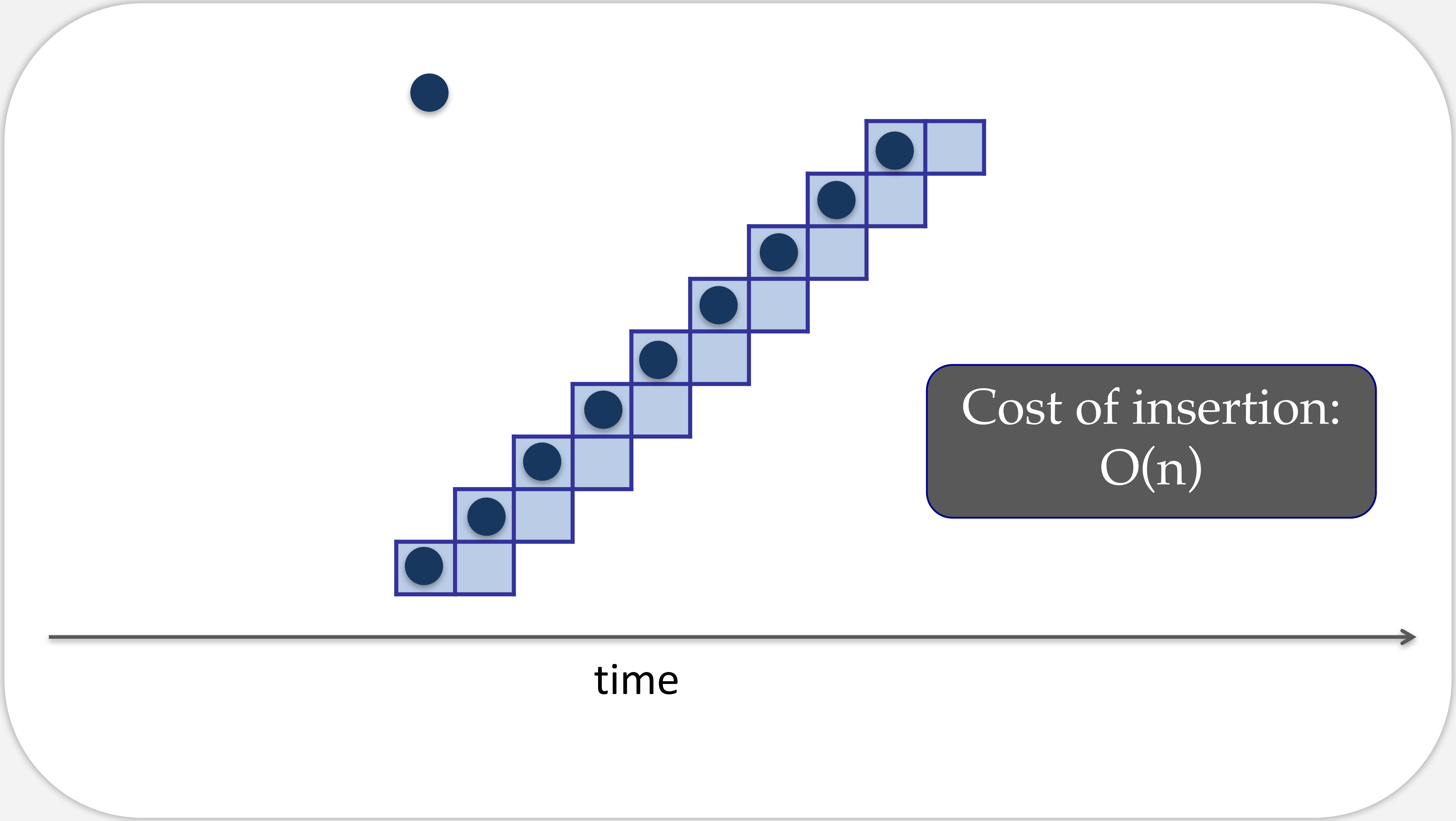


On new insertions: Reallocate jobs to make room

“It does not do to leave a live dragon out of your calculations, if you live near him.” (Tolkien)

The Cost of Reallocation

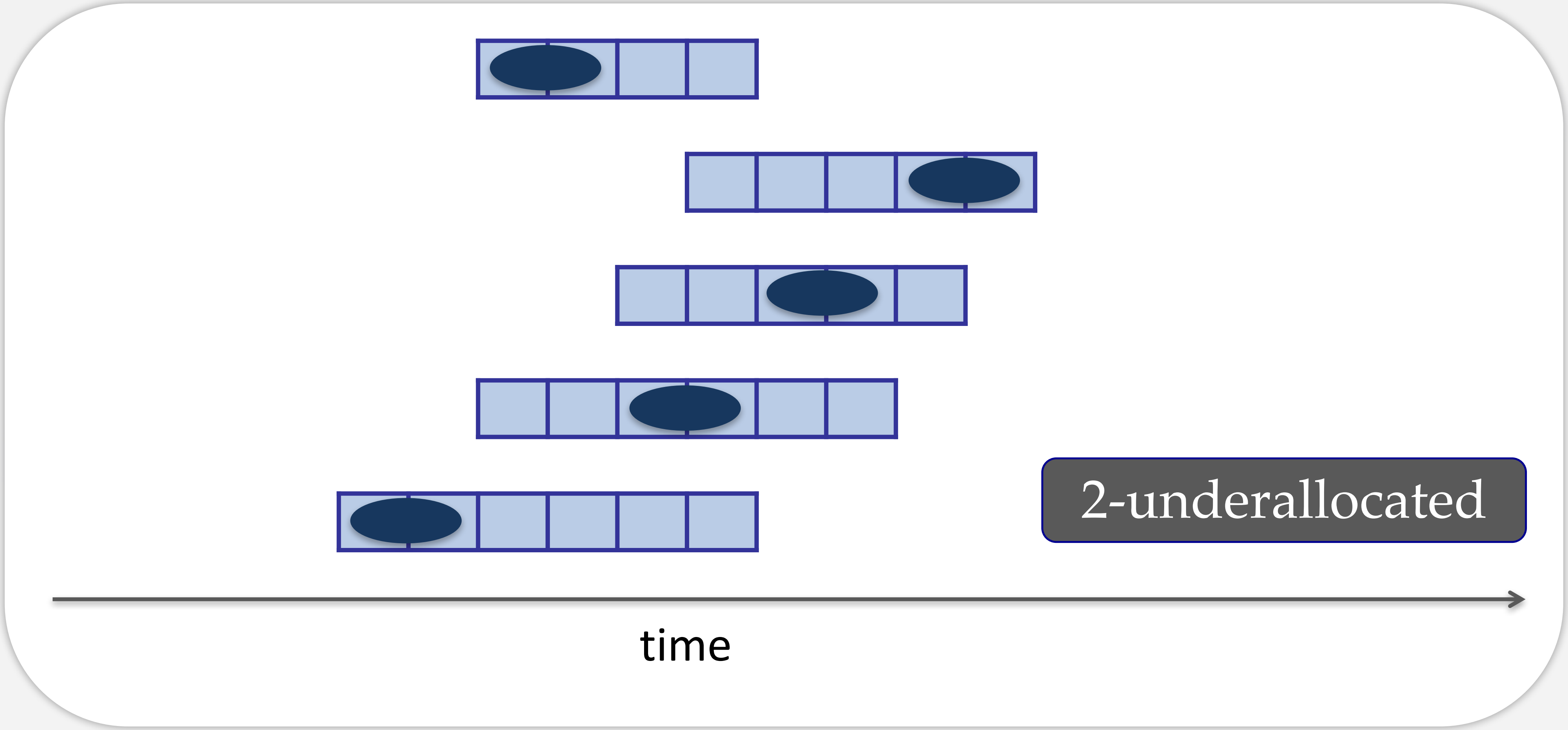
How many items move?



Reallocation can be expensive!

Underallocation

Require some *slack* in the schedule.



Definition:

Instance is γ-underallocated if there exists a feasible schedule where each job takes γ times as long.

"resource augmentation"

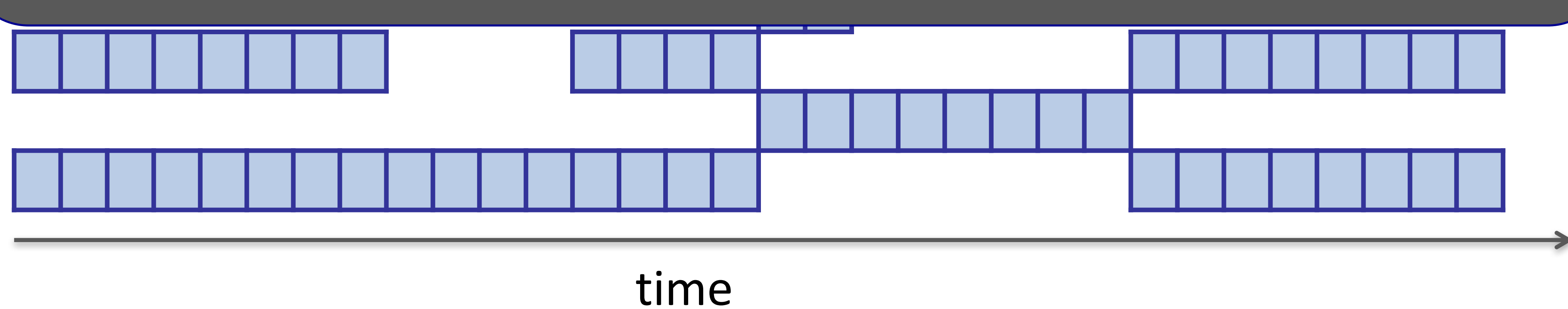
Simplification

Assume: laminar windows

For any two windows W_1 and W_2 either: $W_1 \subseteq W_2$ or $W_2 \subseteq W_1$

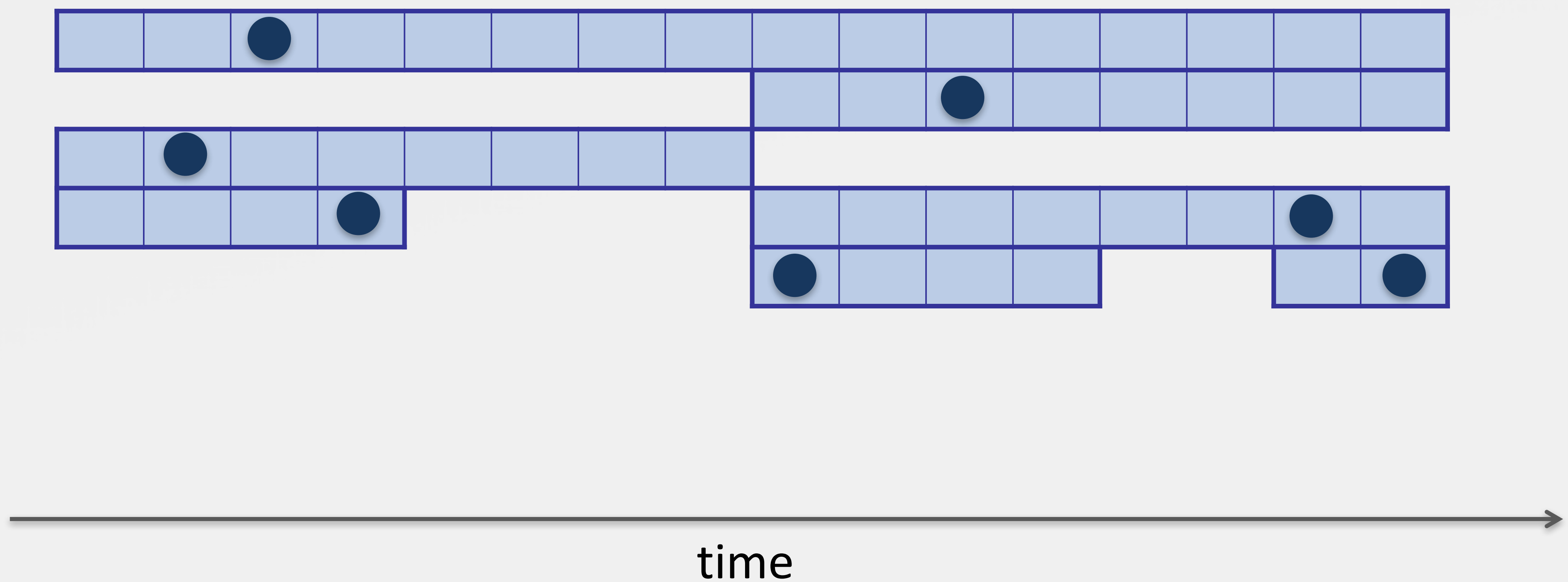
Lemma:

If instance is γ-underallocated, then we can *trim* windows to be laminar and it is still $(\gamma/4)$-underallocated.



Every window: size is a power of 2
start location is a power of 2

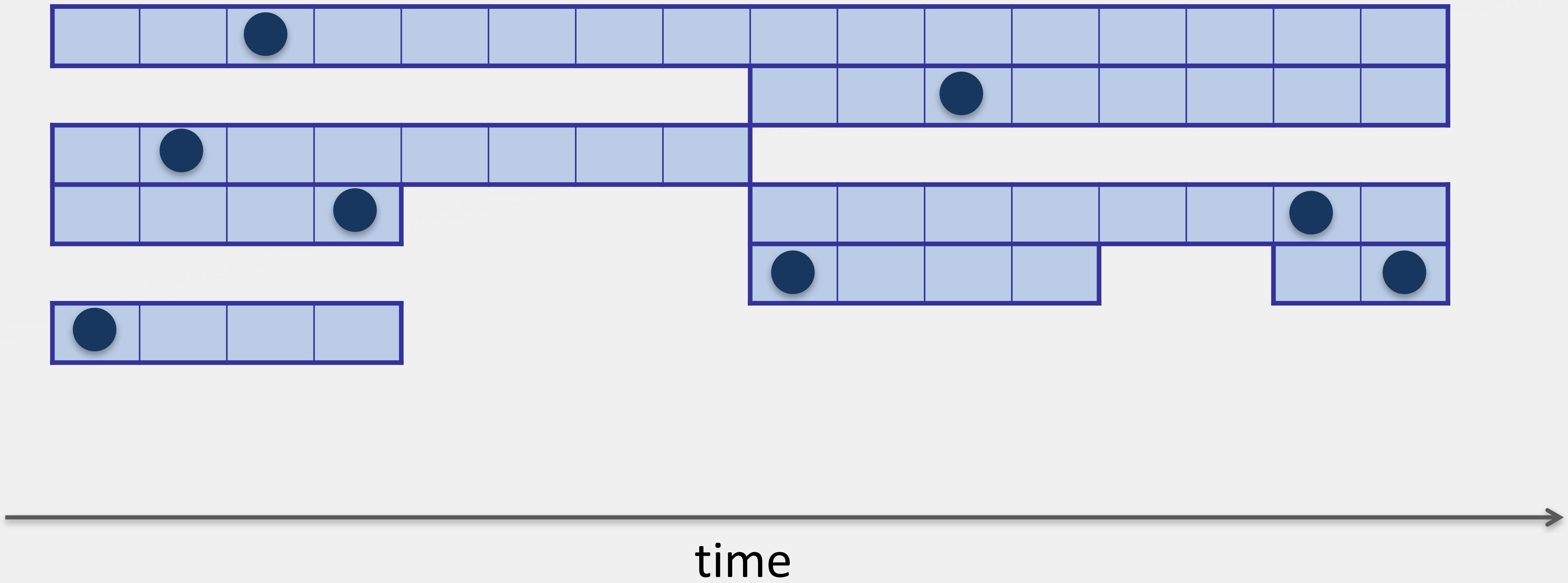
Naïve Pecking Order Scheduling



Rule:

- **Prioritize small windows over big windows.**
- **To allocate a new job, kick out a larger job (if necessary).**

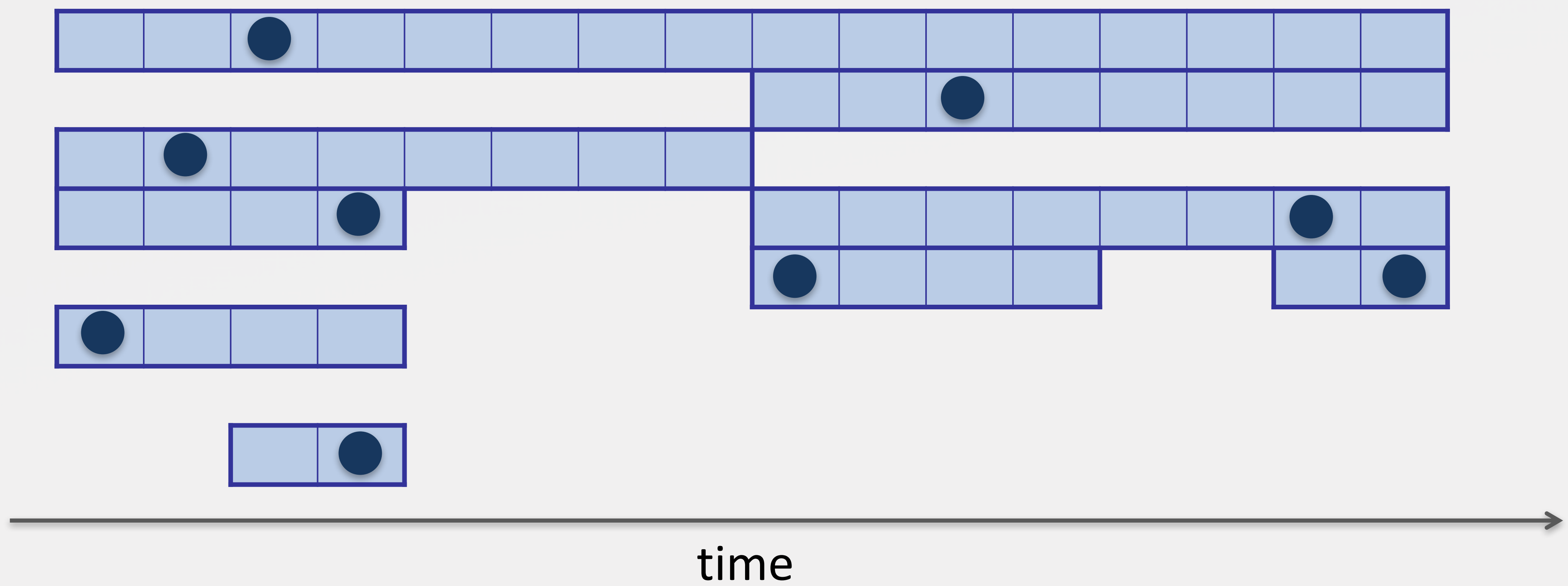
Naïve Pecking Order Scheduling



Algorithm:

- **If there is an empty slot, use it.**

Naïve Pecking Order Scheduling

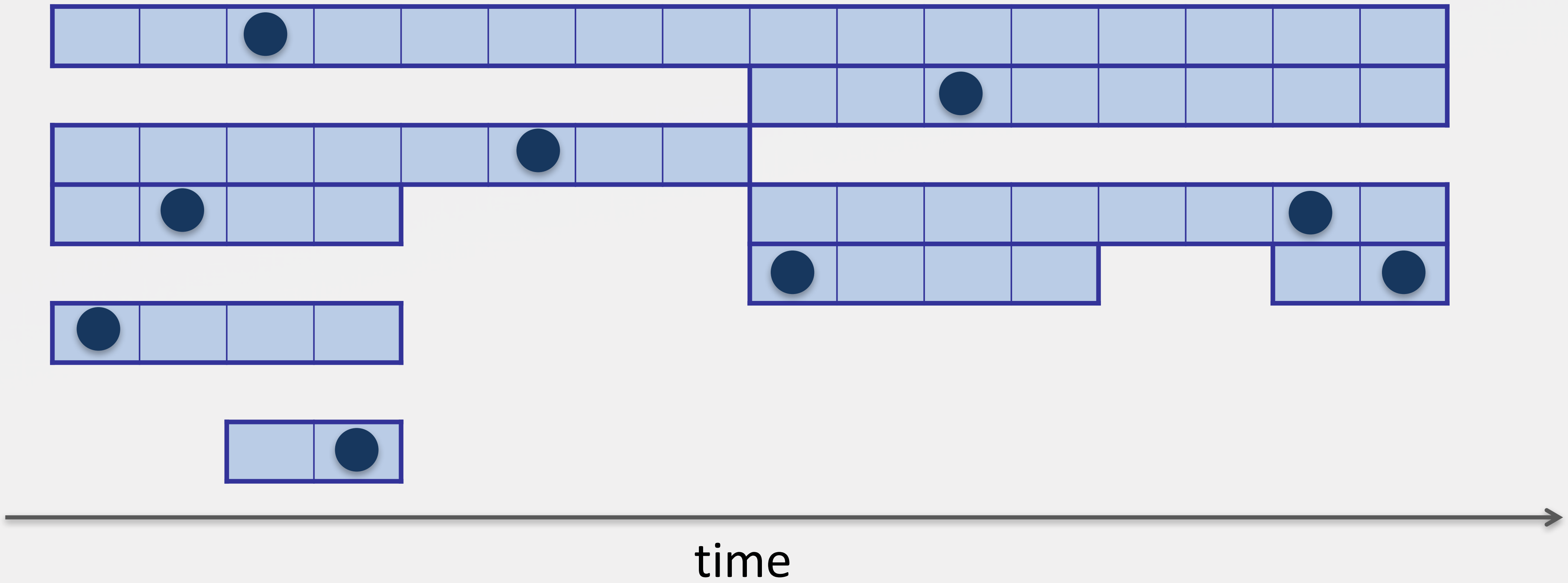


Algorithm:

- **If there is an empty slot, use it.**
- **If there is no empty slot, kick out a job from a larger window and reinsert the kicked job.**

Reallocation Schedulers

Naïve Pecking Order Scheduling

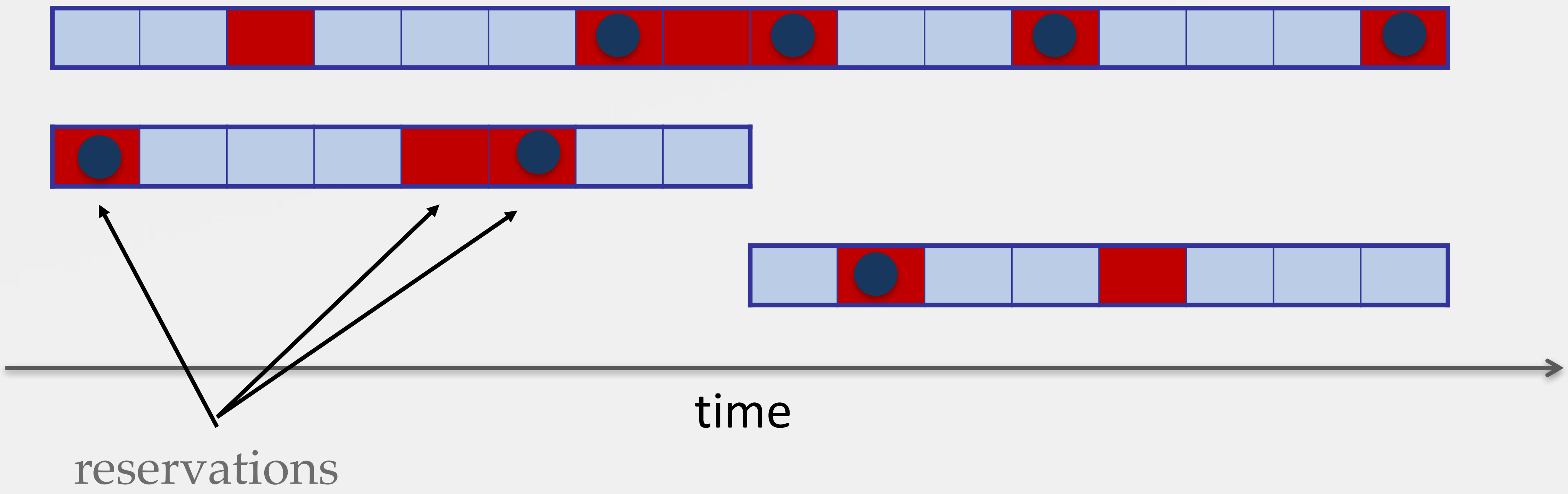


Claim: **Can always find larger job to kick out (if needed).**

Claim: **Worst-case O(log n) reallocations.**

Assume maximum window size is O(n).

Planning Ahead: Reservations

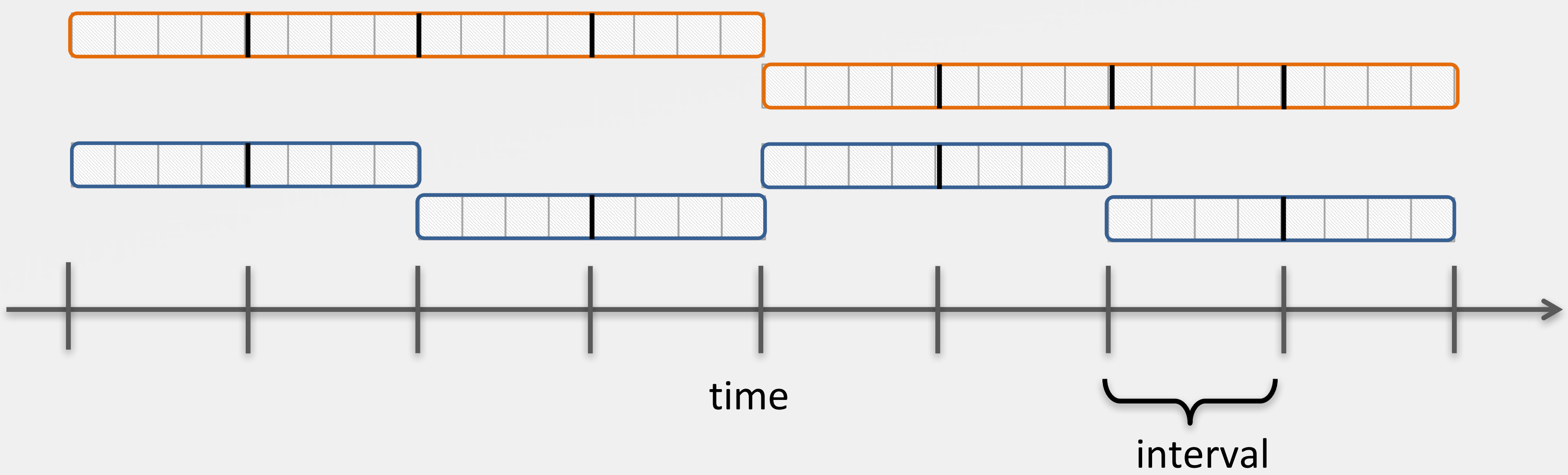


Idea:

- Group jobs by window.
- Each window reserves timeslots.
- Key invariant: each window gets enough reservations.

Reallocation Schedulers

Planning Ahead: Reservations



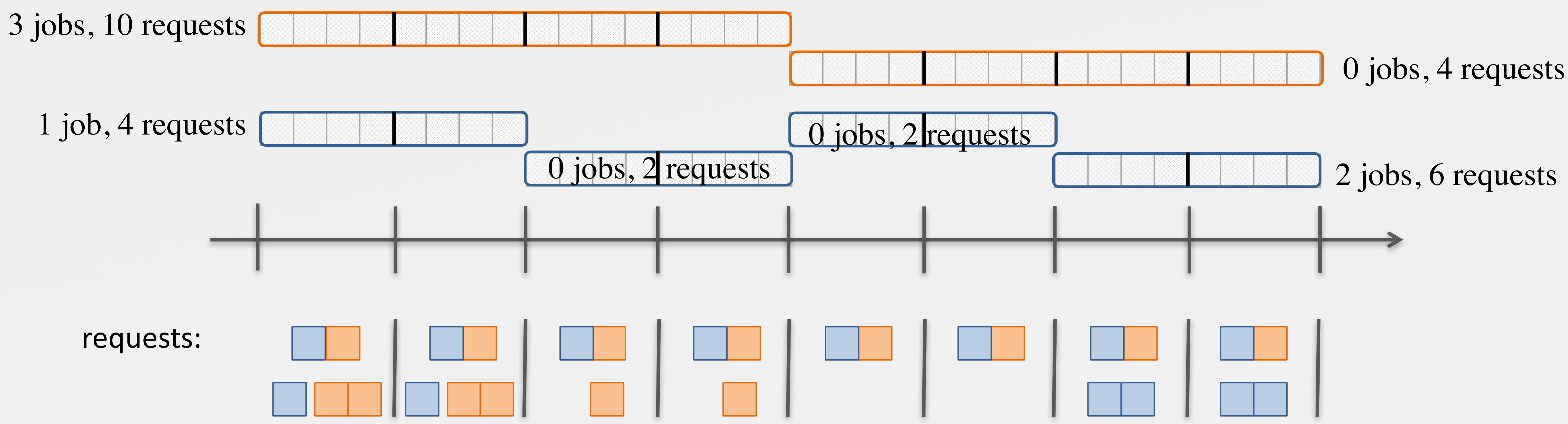
Two-level Scheduling:

- Intervals have size $L = \Theta(\log n)$.
- Schedule any window of size $\leq L$ recursively.
- Focus on scheduling windows of size $> L$.

"No matter what the work you are doing, be always ready to drop it. And plan it so as to be able to leave it. (Tolstoy)

Reallocation Schedulers

Planning Ahead: Reservations



Each window requests:

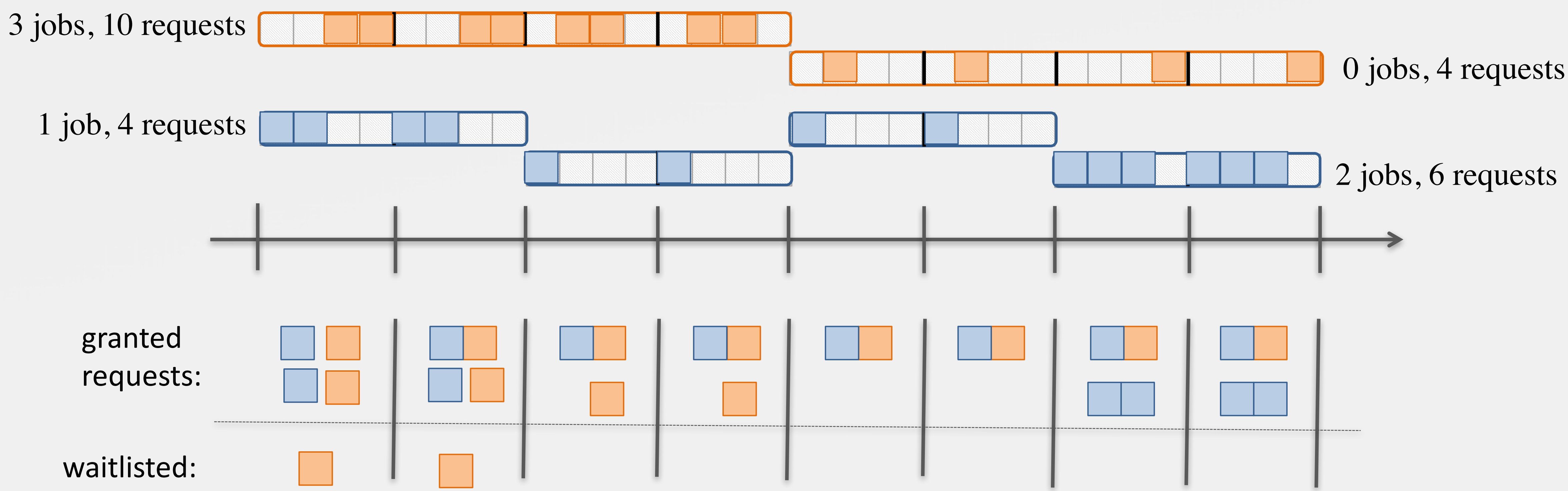
- **1 reservation per contained interval.**
- **2 reservations per job.**

Requests are spread evenly over intervals.

"No matter what the work you are doing, be always ready to drop it. And plan it so as to be able to leave it. (Tolstoy)

Reallocation Schedulers

Planning Ahead: Reservations



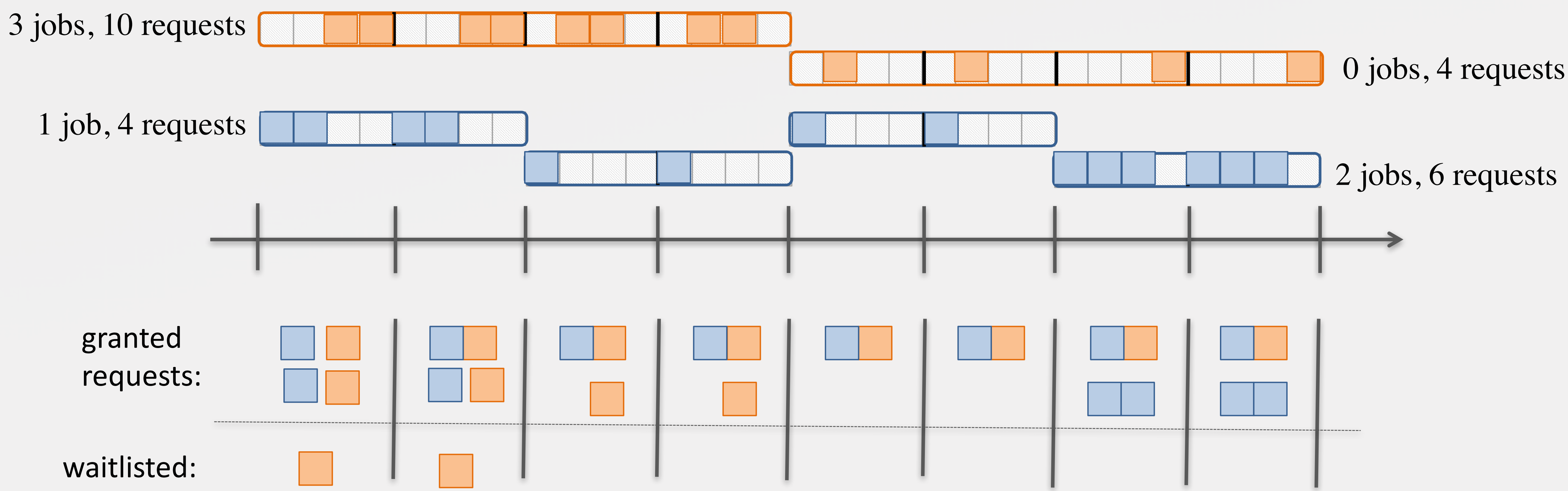
Intervals grant requests:

- **Pecking-order scheduling!**
- **Grant requests for smaller windows first.**

Not all requests are granted.

Reallocation Schedulers

Planning Ahead: Reservations



Key claim:

If instance is 8 underallocated, then each window has *enough* granted reservations.

Planning Ahead: Reservations

Key claim:

If instance is 8 underallocated, then each window has *enough* granted reservations.

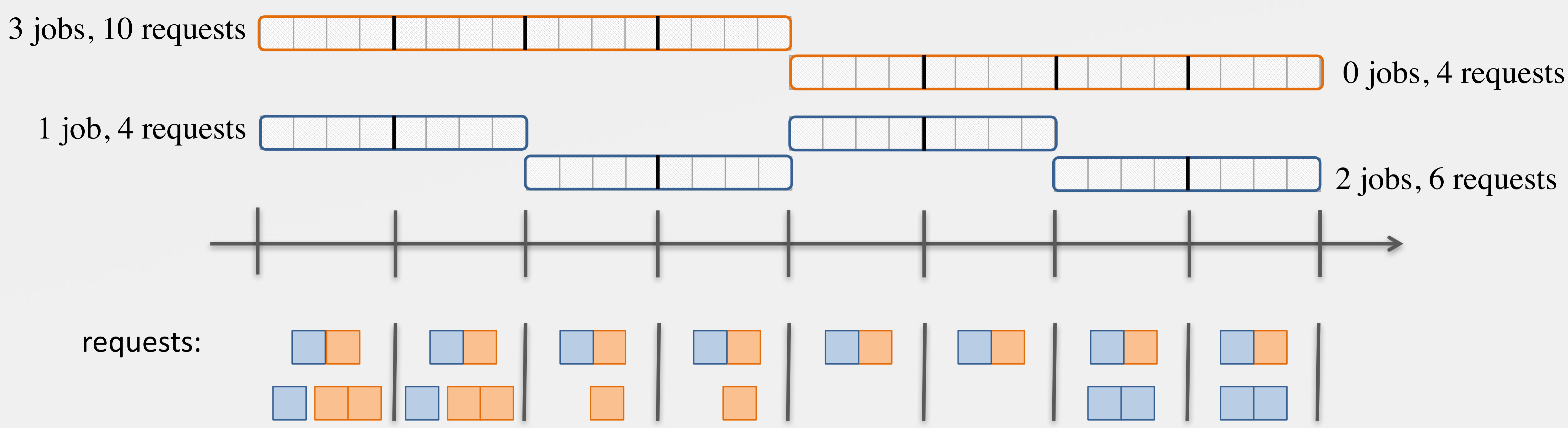
Intuition:

- Assume window W covers I intervals, x jobs, and makes 2x+I requests for reservations.
- Each interval gets ≅ 2x/I reservations.
- If I/2 intervals grant them, then all good.
- If I/2 intervals do not grant them, then they must be completely full, contradicting 8-underallocation.

“Let our advance worrying become advance thinking and planning.” (Churchill)

Reallocation Schedulers

Planning Ahead: Reservations



On a new job for window W:

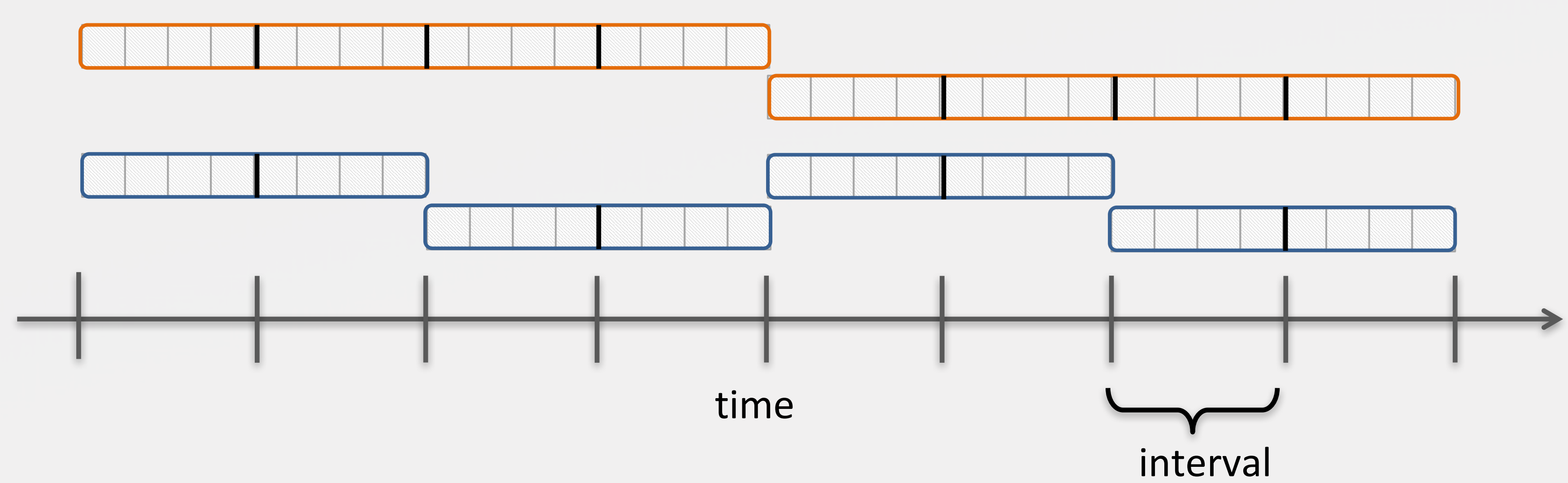
- **Window W makes 2 new reservations requests.**
- **If a new request is granted, it may force *one* reallocation.**
- **Place job in slot for granted reservation.**

O(1) reallocations per insert/delete.

“Let our advance worrying become advance thinking and planning.” (Churchill)

Reallocation Schedulers

Planning Ahead: Reservations



How to schedule jobs of size ≤ L:

1. Naïve pecking-order scheduling ➔ O(log log n) reallocations.
2. Recursively use reservation system ➔ O(log*n) reallocations.

Insert in a smaller interval can cause top level to perform O(1) reallocations.

Reservations to the Rescue

Setting:

- **Unit-length tasks.**
- **Number of servers: p**
- **Arrival times and deadlines.**

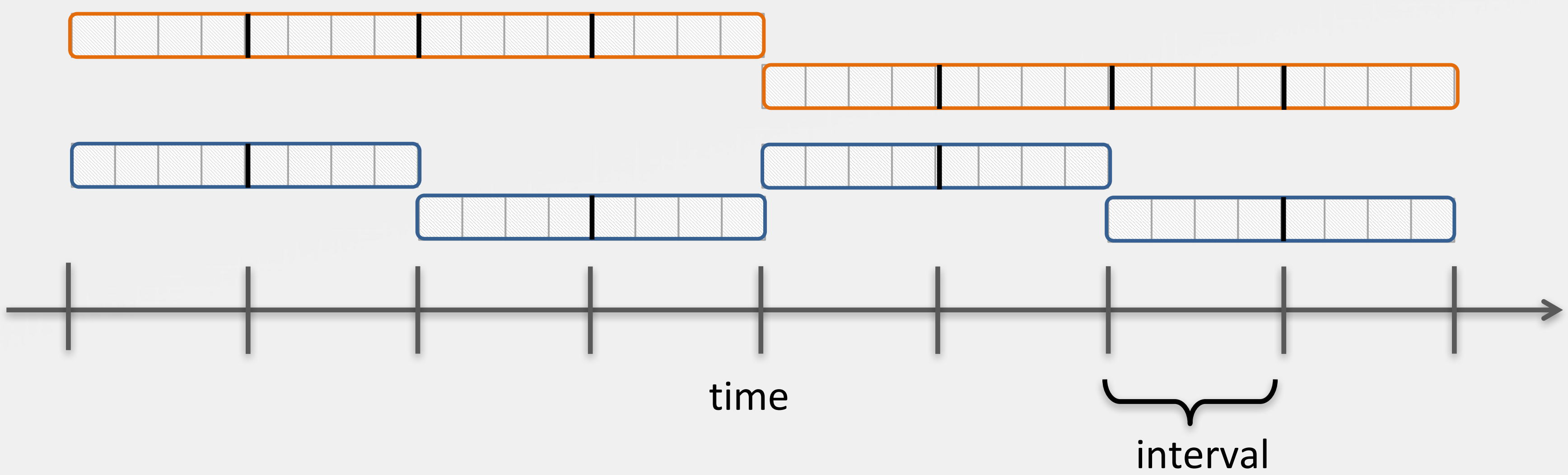
Goal: **Feasibility**

Key take-away:
Make 2 reservations for dinner to ensure you have somewhere to eat!

Key result:

- Assume at all times:
  - At most n tasks
  - At least $O(1)$ slack
- Reallocation cost per insertion/deletion: $O(\log^* n)$
- Number of migrations per insertion/deletion: $O(1)$

Open Questions



Can we get O(1) reallocations?

What about non-unit-length jobs? With preemption?

What about dependencies? Jobs form a DAG?

How to Plan Ahead
A Play in Three Acts

Act I : A Few Reservations

Wherein we schedule simple (unit-length) tasks with arrival times and deadlines.

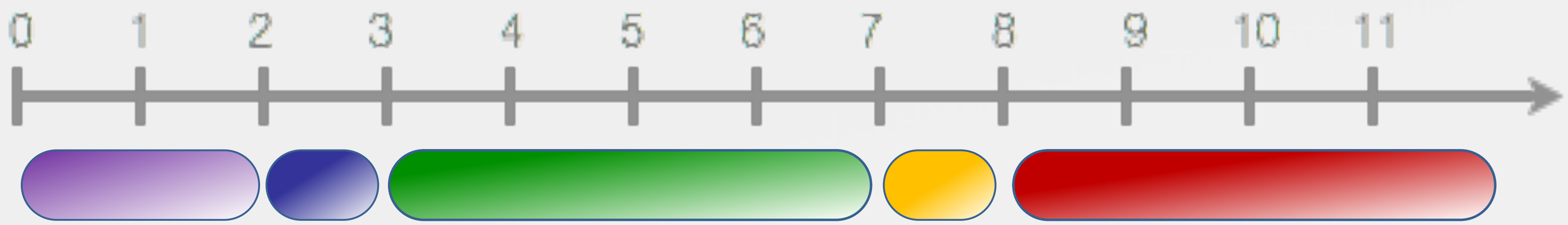
Act II : One Algorithm to Rule Them All

Wherein we discover a (cost-oblivious) champion able to defeat any (subadditive) reallocation cost. Our champion knows how to minimize the makespan, but no more.

Act III : Data Structures to the Rescue!

Wherein our champion seeks aid from the faraway Land of Data Structures in order to minimize the sum-of-completion-times dragon.

Minimize Makespan



insert...
delete(purple)
delete(green)
insert(red, 4)

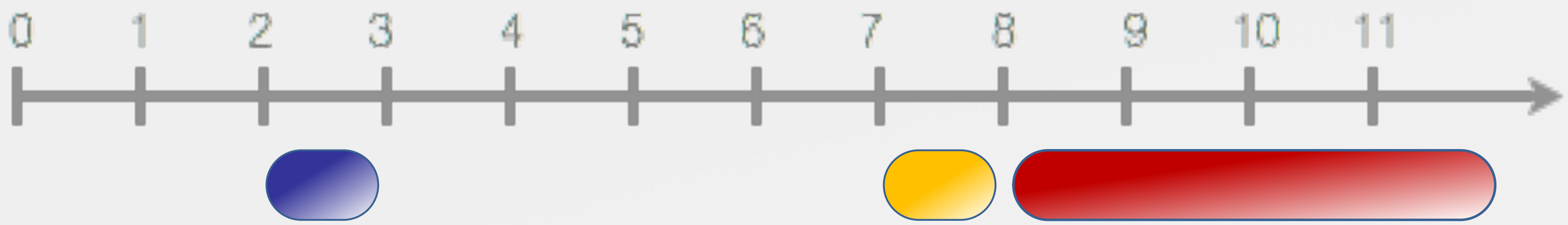
Basic problem:

- Arbitrary length jobs
- Jobs added
- Jobs deleted
- Goal: minimize the makespan

“In preparing for battle, I have always found that plans are useless, but planning is indispensable.” (Eisenhower)

Simple Scheduling

Minimize Memory



malloc…
free(purple)
free(green)
malloc(red, 4)

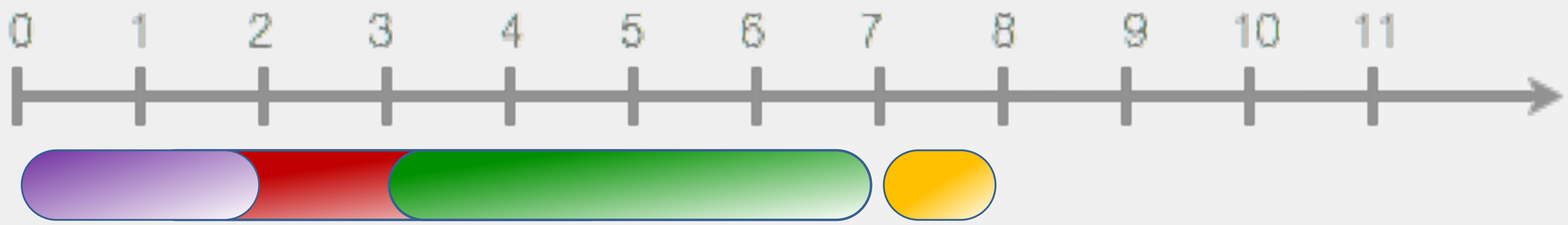
AKA: Memory Allocation

- Allocate and free memory.
- Competitive ratio: **log(n)**

“In preparing for battle, I have always found that plans are useless, but planning is indispensable.” (Eisenhower)

Reallocation

A Better Schedule



insert…
delete(purple)
delete(green)
insert(red, 4)

Reallocate:

- **Move items to make more space.**
- **Results in a better schedule!**

See: garbage collection!

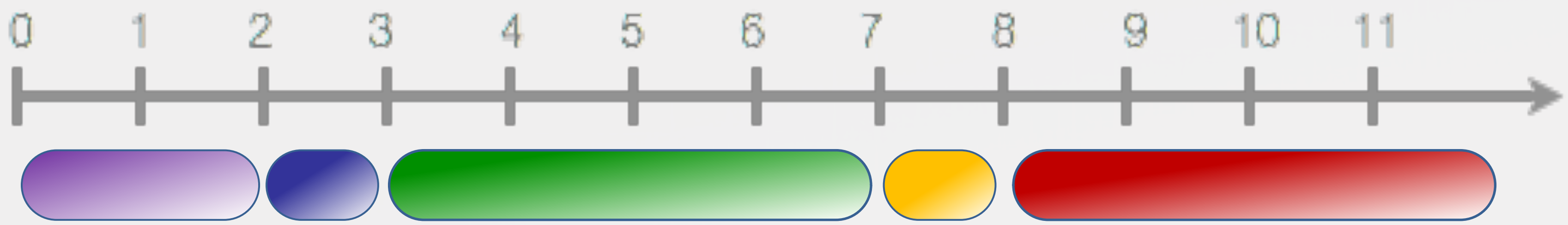
Note: ensuring *optimal* schedule requires reallocating everything.

What is the cost of rescheduling a job?

Options:

- *Unit cost*: 1 per move.
- *Linear cost*: s to move job of size s.
- *Mixed*: migrating to a new server has linear cost while changes on the same server are unit cost.
- *Other*: some function of size that depends on buffering, caches, and other parameters.

What is the cost of reallocation?



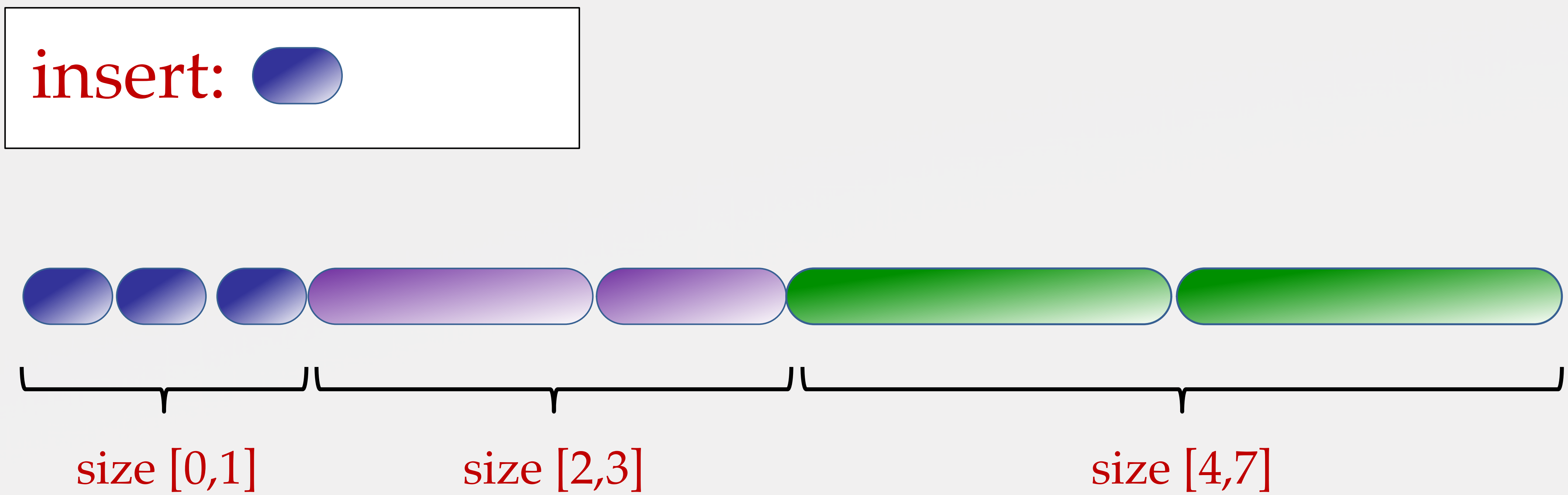
Moving a job of size X has cost 1.

Option 1: Unit cost

- **Maintain schedule O(1) • OPT.**
- **Minimize number of jobs moved.**

Approximately optimal schedule!

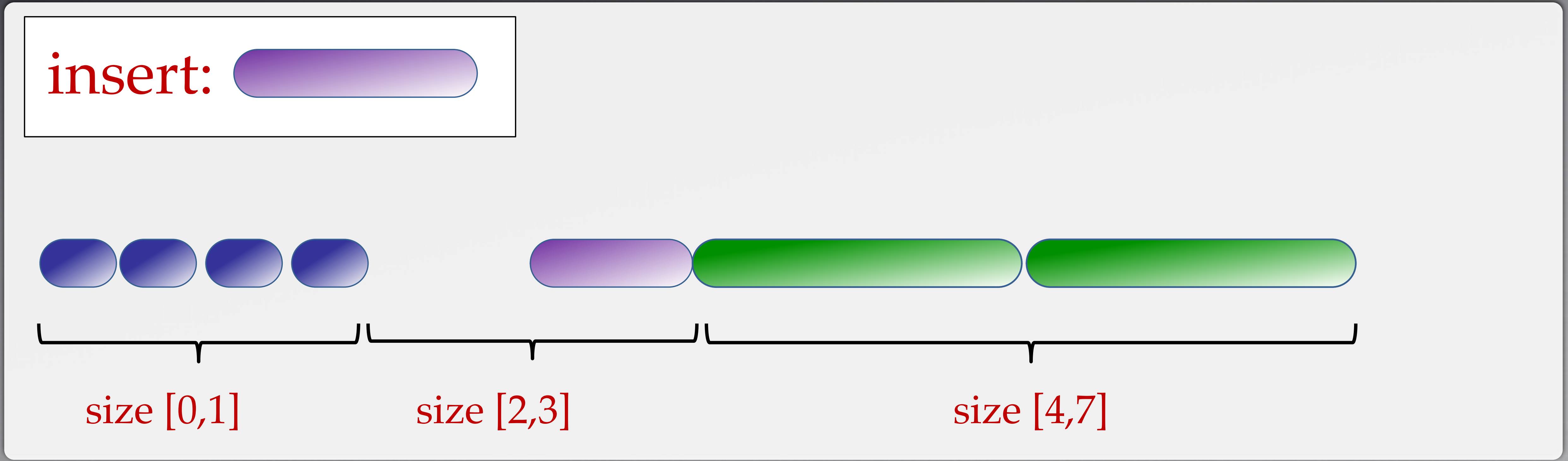
What is the cost of reallocation?



Algorithm:

- **Sort jobs by approximate size: group by powers of 2.**
- **Cascade jobs on insertion / deletion.**

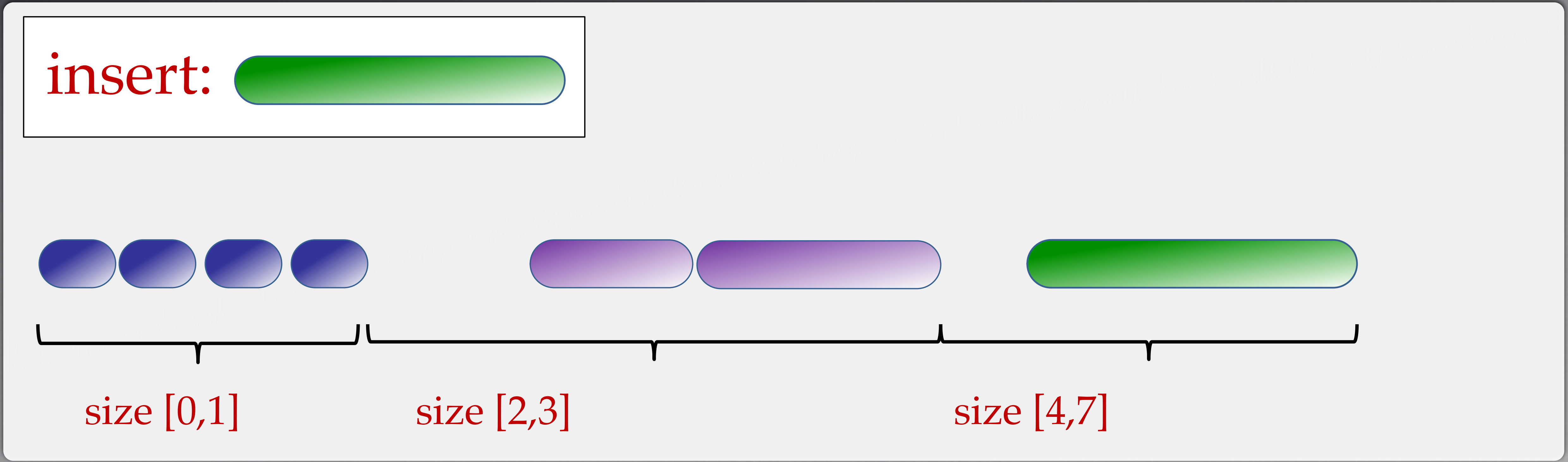
What is the cost of reallocation?



Algorithm:

- Sort jobs by approximate size: group by powers of 2.
- Cascade jobs on insertion / deletion.

What is the cost of reallocation?

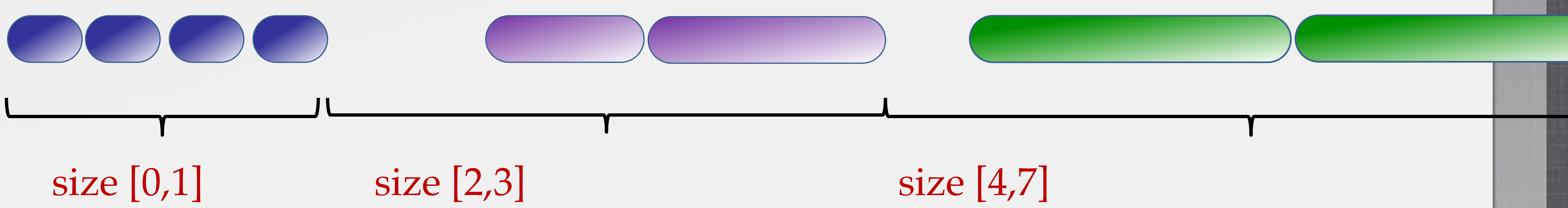


Algorithm:

- Sort jobs by approximate size: group by powers of 2.
- Cascade jobs on insertion / deletion.

What is the cost of reallocation?

insert:



Algorithm:

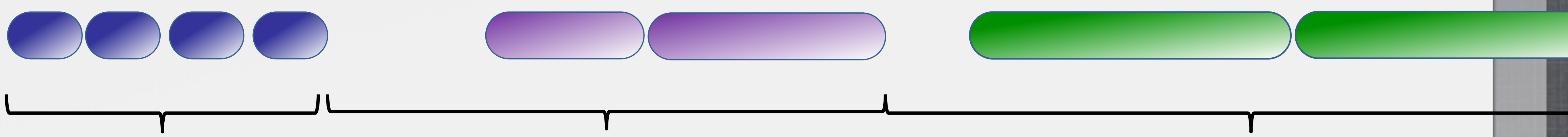
- **Sort jobs by approximate size: group by powers of 2.**
- **Cascade jobs on insertion / deletion.**

O(log Δ) reallocations

O(1)•OPT time

What is the cost of reallocation?

insert:



size [0,1]

size [2,3]

size [4,7]

Algorithm:

Analysis same as binary counter amortized analysis!

- Sort jobs by approximate size: group by powers of 2.
- Cascade jobs on insertion / deletion.

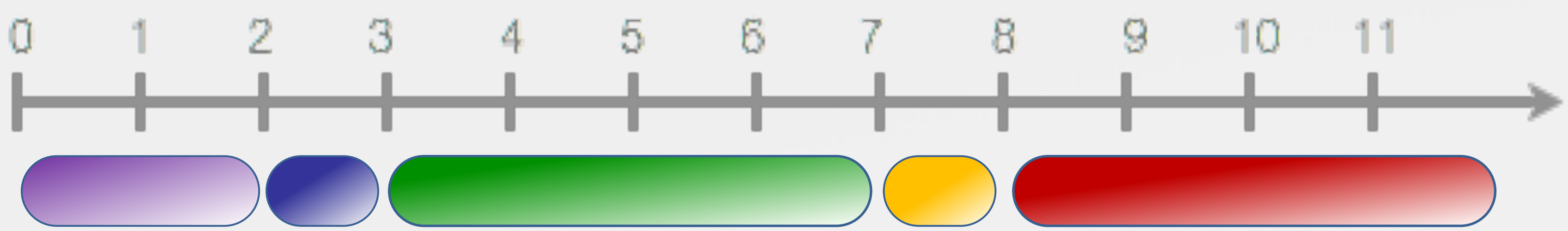
O(1) reallocations (amortized)

O(1)•OPT time

“If you fail to plan, you are planning to fail.” (Benjamin Franklin)

Reallocation

What is the cost of reallocation?



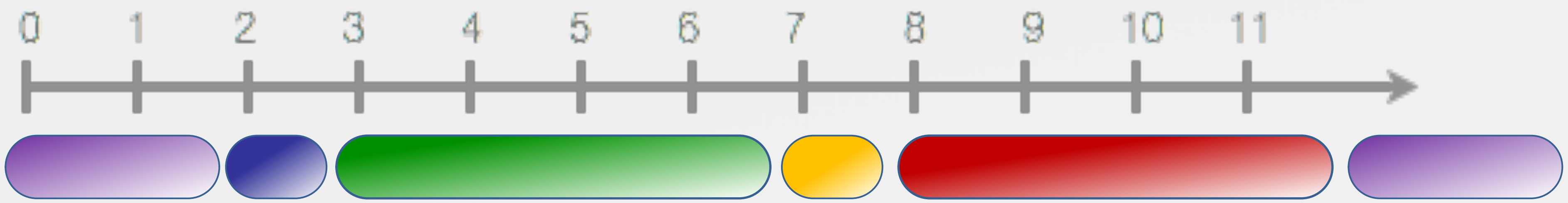
Moving a job of size X has cost X.

Option 2: Linear cost

- **Maintain schedule O(1)•OPT.**
- **Minimize volume of jobs moved.**

"If you fail to plan, you are planning to fail." (Benjamin Franklin)

What is the cost of reallocation?



cost of initial allocation

Algorithm:

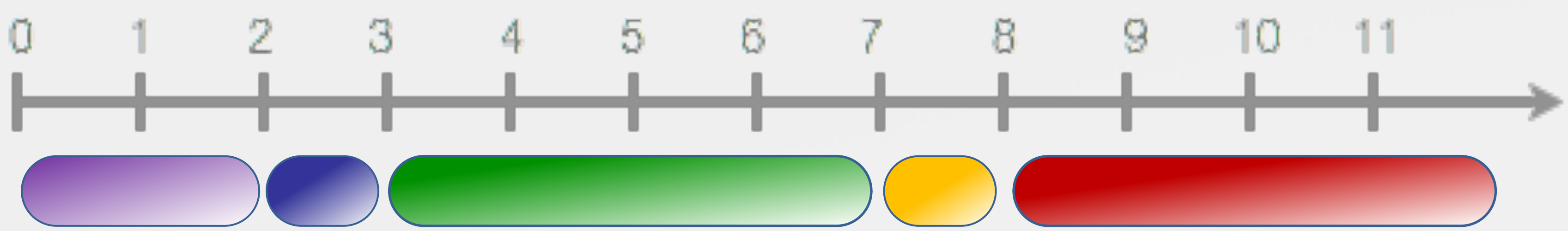
- Do nothing…
- …until half the space is empty.
- Then compress.

O(1)• **VAlloc reallocation cost**

O(1)•OPT time

"There cannot be a crisis next week. My schedule is already full." (Kissinger)

What is the cost of reallocation?



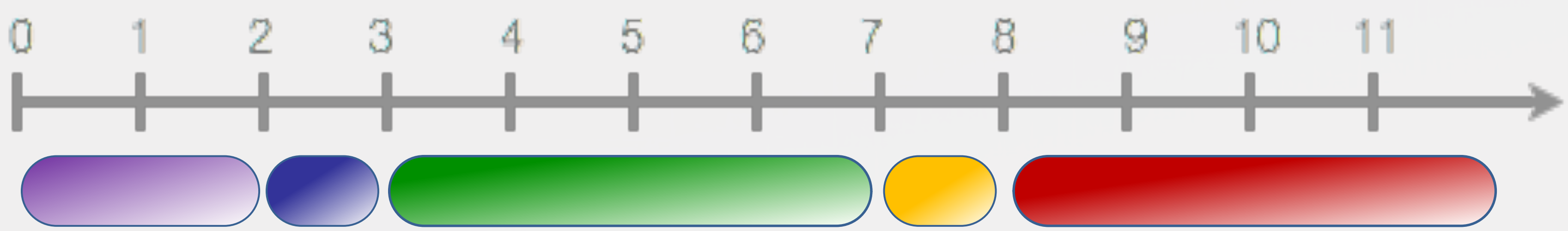
Moving a job of size X has cost f(X).

Option 3: Unknown cost

- Maintain schedule **O(1)•OPT.**
- **Minimize cost of jobs moved.**

“There cannot be a crisis next week. My schedule is already full.” (Kissinger)

What is the cost of reallocation?



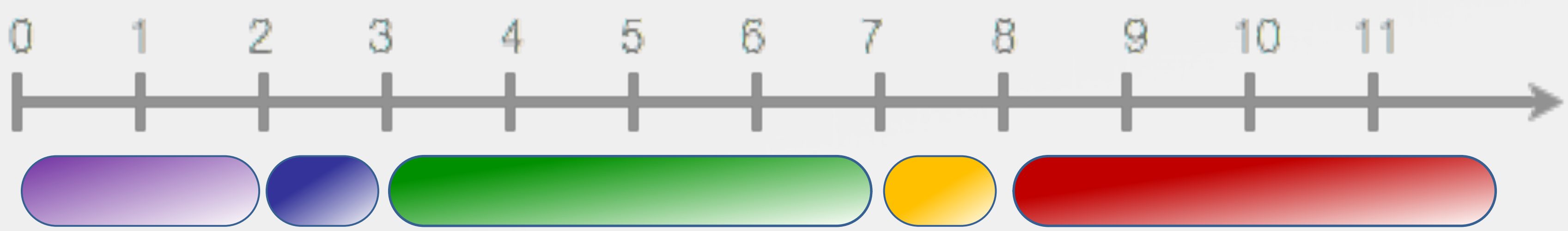
Moving a job of size X has cost f(X).

Example: database storage allocation

- Moving one block has a fixed cost.
- Moving a large number of blocks is linear cost?
- Pre-fetching?
- Caching?

“There cannot be a crisis next week. My schedule is already full.” (Kissinger)

What is the cost of reallocation?



Moving a job of size X has cost f(X).

Option 3: Cost Oblivious

- **Maintain schedule O(1)•OPT.**
- **Minimize volume of jobs moved.**

“subadditive” functions

Assume $f(x + y) \leq f(x) + f(y)$.

Unit Cost ⟷ Linear Cost

Cost-Oblivious Scheduling

Basic Idea

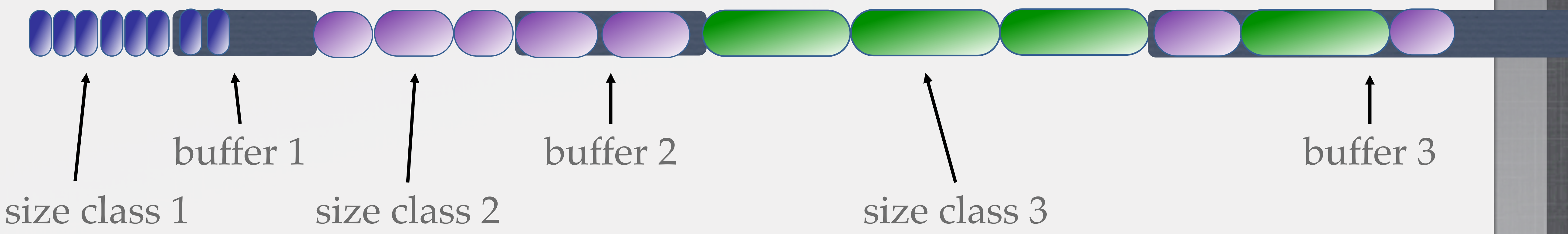
Organization:

- Sort jobs by approximate size: group by powers of 2.
- A job class of volume **V** is followed by a buffer of size **V**.

Cost-Oblivious Scheduling

Basic Idea

Key property: buffer j only holds items from size classes ≤ j



On insertion:

- **Put job from class j in first available buffer ≥ j.**

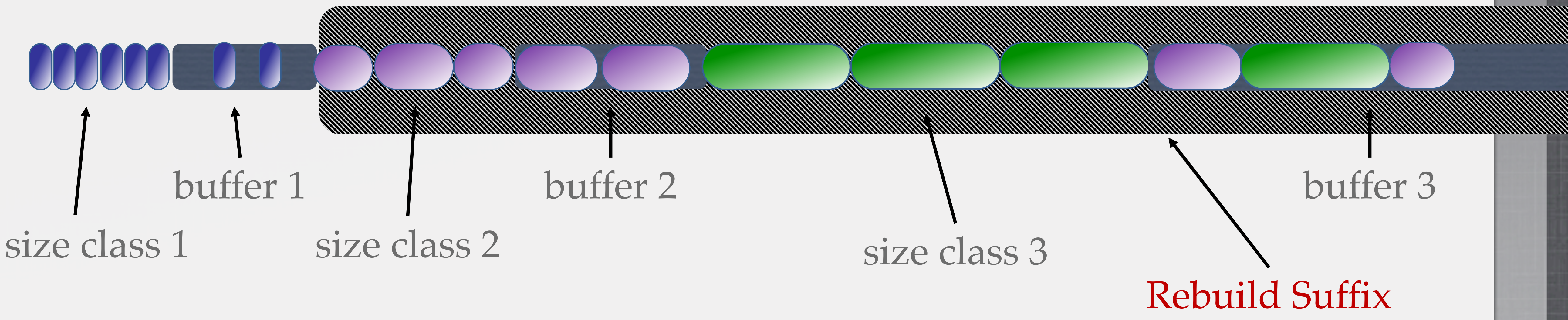
On deletion:

- **Mark deleted (do nothing).**

Cost-Oblivious Scheduling

Basic Idea

Key property: buffer j only holds items from size classes ≤ j



If you cannot insert an item: (buffers are full)

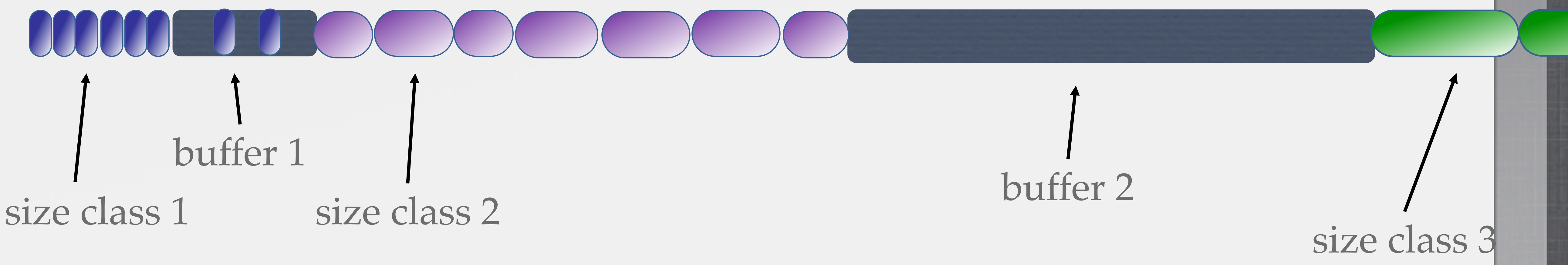
- Find self-contained suffix of array.
- Remove deleted items.
- Rebuild with empty buffer.

“Dreaming, after all, is a form of planning.” (Steinem)

Cost-Oblivious Scheduling

Basic Idea

Key point: full buffers pay for the rebuild.



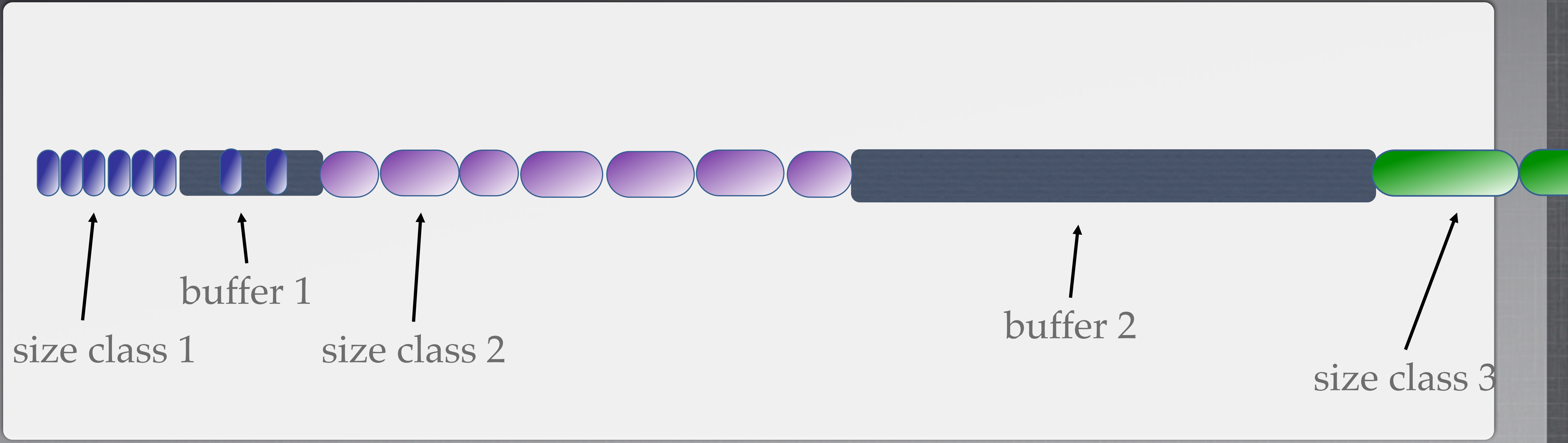
If you cannot insert an item: (buffers full)

- **Find self-contained suffix of array.**
- **Remove deleted items.**
- **Rebuild with empty buffer.**

"Dreaming, after all, is a form of planning." (Steinem)

Cost-Oblivious Scheduling

Quick Analysis



Full buffers pay for size class:

- **Buffers in "rebuild suffix" are half full (or small).**
- **Items in buffer are no larger than items in size class.**
- **Subadditive: many small items have larger budget than equivalent volume big item.**

One Algorithm to Rule Them All

Setting:

- Arbitrary length tasks.
- No constraints on when to schedule.
- Unknown (!?!) allocation/reallocation cost.

Goal: Minimize makespan

Key results:

For any subadditive, non-decreasing cost function:

- ReallocationCost = $O(1) \cdot$ AllocationCost
- Makespan = $O(1+\epsilon) \cdot$ OPT

How to Plan Ahead
A Play in Three Acts

Act I : Reservations to the Rescue

Wherein we schedule simple (unit-length) tasks with arrival times and deadlines.

Act II : One Algorithm to Rule Them All

Wherein we discover a (cost-oblivious) champion able to defeat any (subadditive) reallocation cost. Our champion knows how to minimize the makespan, but no more.

Act III : Data Structures to the Rescue!

Wherein our champion seeks aid from the faraway Land of Data Structures in order to minimize the sum-of-completion-times dragon.

"Give me six hours to chop down a tree and I will spend the first four sharpening the axe." (Lincoln)

Data Structures to the Rescue

Setting:

- Arbitrary length tasks.
- No constraints on when to schedule.
- Number of servers: p
- Unknown (!?!) allocation/reallocation cost.

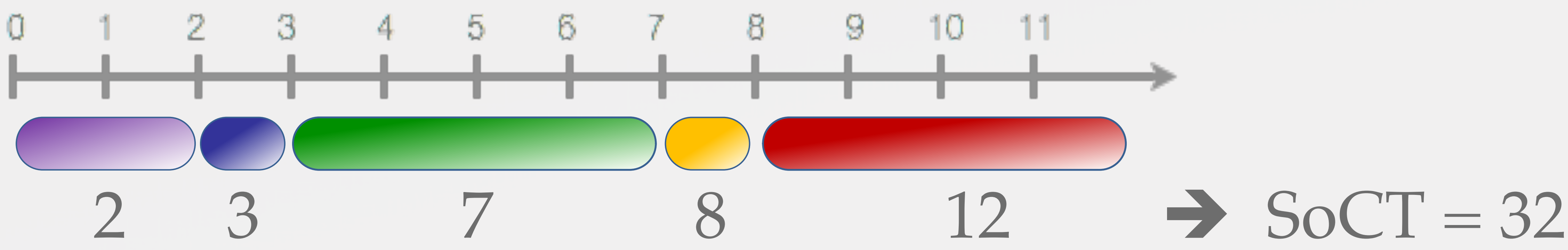
Goal: Minimize sum-of-completion-times

Key results:

For any subadditive, non-decreasing cost function:

- ReallocationCost = $O(\log^3 \log \Delta) \cdot$ AllocationCost
- SoCT= $O(1) \cdot$ OPT

Minimize Sum-of-Completion-Times

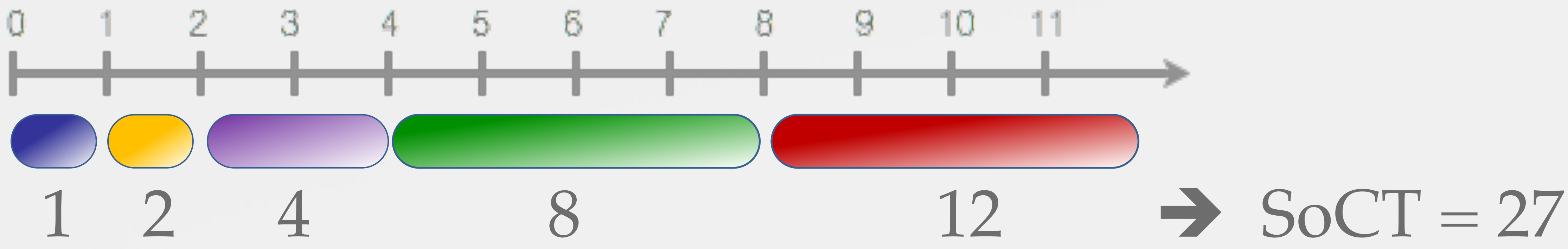


Basic problem:

- **Arbitrary length jobs**
- **Jobs added**
- **Jobs deleted**
- **Goal: minimize the sum-of-completion-times.**

“Give me six hours to chop down a tree and I will spend the first four sharpening the axe.” (Lincoln)

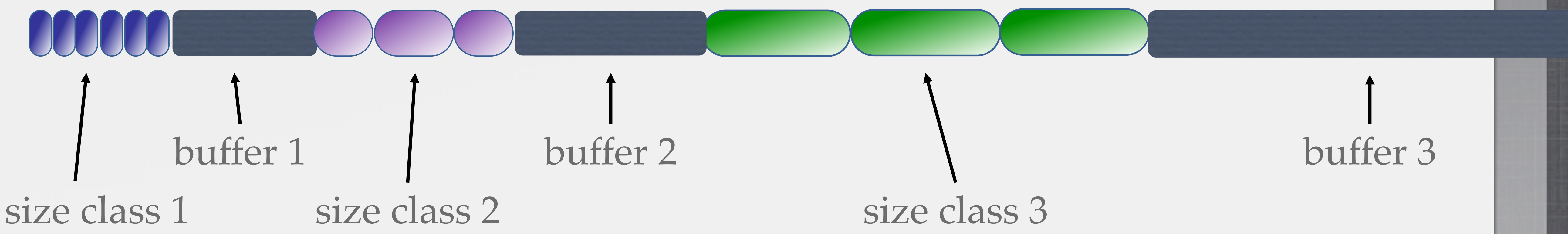
Minimize Sum-of-Completion-Times



Standard solution:

Sort jobs by size from smallest to largest.

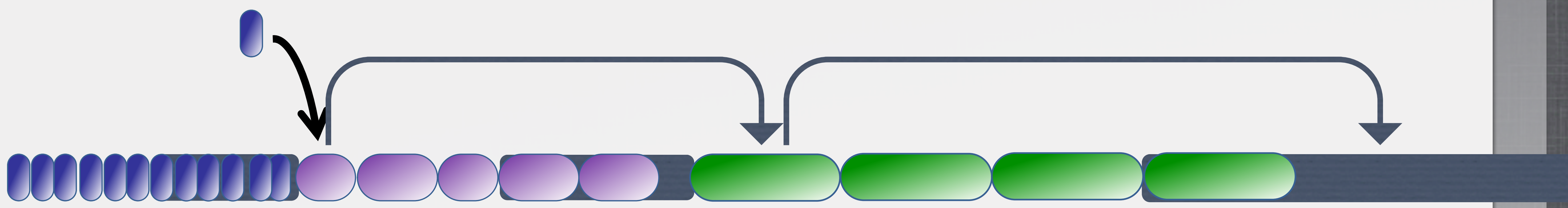
Minimize Sum-of-Completion-Times



Strategy (as before):

- **Sort jobs by approximate size: group by powers of 2.**
- **Job classes are separated by buffers.**

Minimize Sum-of-Completion-Times

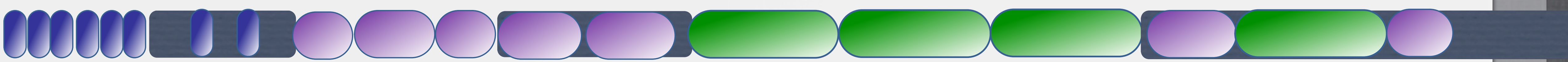


Too expensive: small insert causes large jobs to move.

Things not to do:

- **Do not cascade jobs.**

Minimize Sum-of-Completion-Times

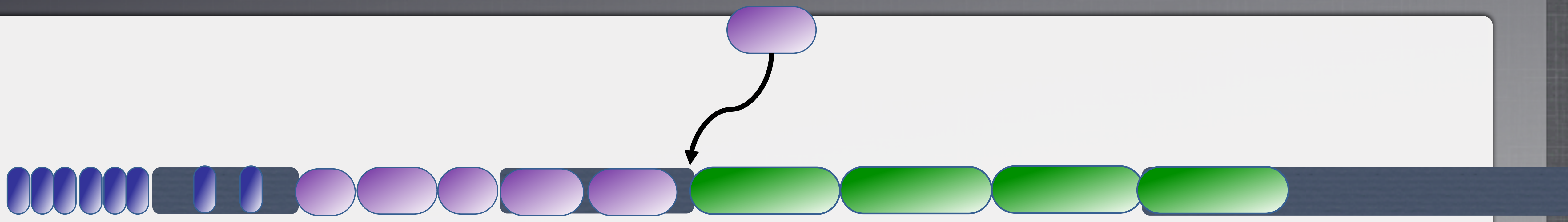


Sum-of-completion-times is too big.

Things not to do:

- **Do not cascade jobs.**
- **Do not use later buffers.**

Minimize Sum-of-Completion-Times

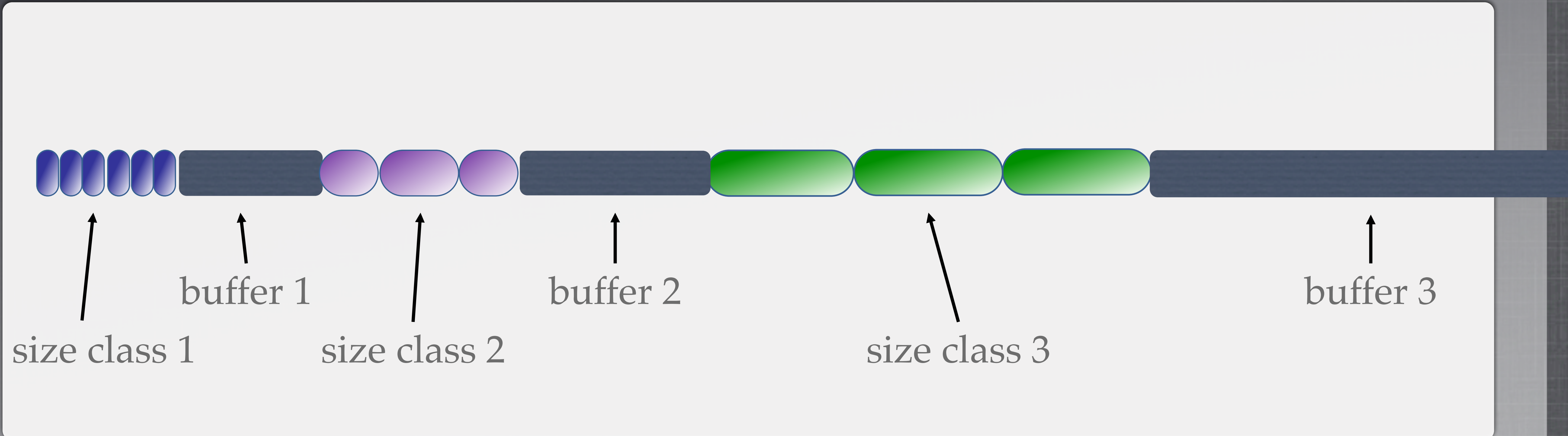


Only move jobs "to the right."

Things not to do:

- **Do not cascade jobs.**
- **Do not use later buffers.**
- **Do not move small jobs to make room for big jobs.**

Minimize Sum-of-Completion-Times



Key requirements:

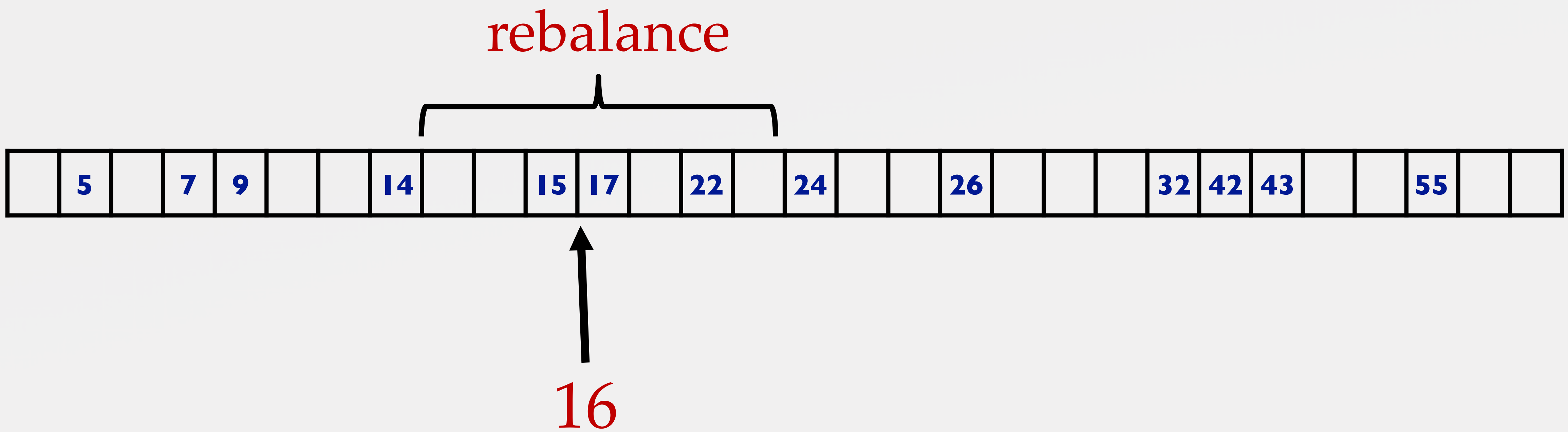
- **Maintain job classes by (approximate) size.**
- **Maintain "prefix density" (i.e., buffers not too big).**
- **On insertion, do not move too many jobs.**
- **Insertions only move larger jobs.**

Data structure problem:

Maintain an array subject to insertion/deletion:

- At most k *cursors*, i.e., points of insertion/deletion.
- *Prefix density*: First x items stored in O(x) space.
- *Movement*: On insertion, items move only to the right. On deletion, items move only to the left.
- *Cost*: Amortized O(log k) items moved per operation.

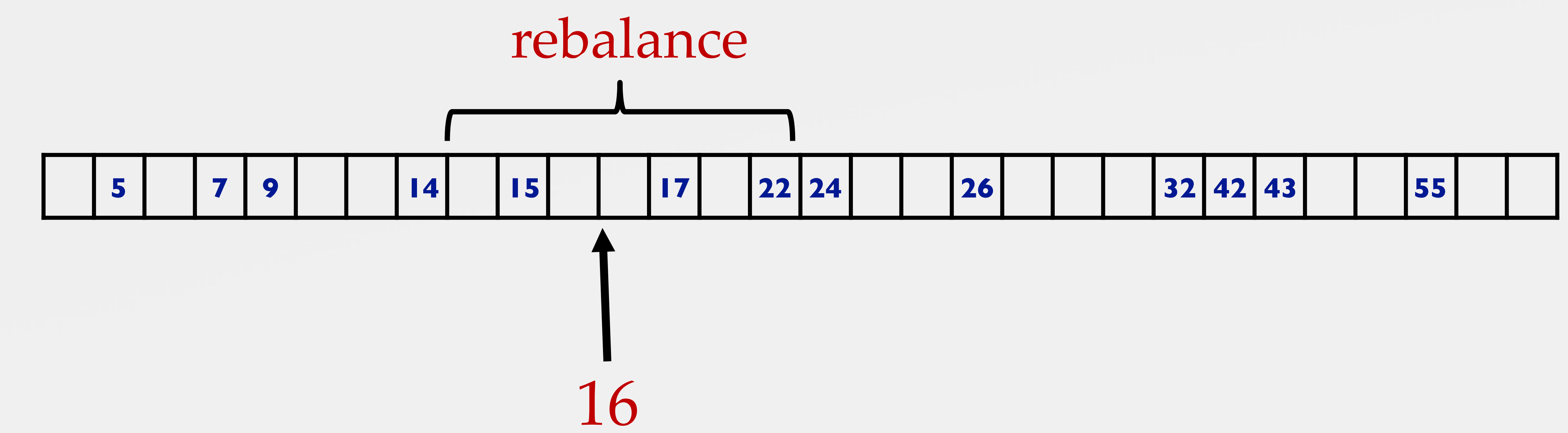
Sparse Table Data Structure



Maintain elements in an array:

- **Stored in order with gaps.**
- **Items rebalanced to make room when necessary.**
- **Support insertions and deletions in $O(\log^2 n)$ time.**

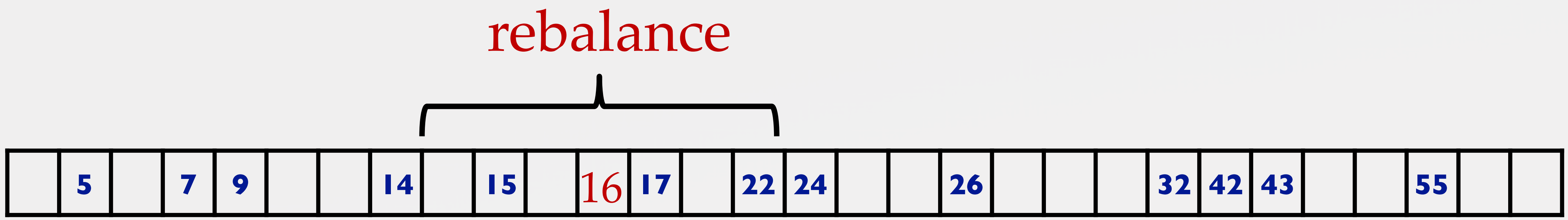
Sparse Table Data Structure



Maintain elements in an array:

- **Stored in order with gaps.**
- **Items rebalanced to make room when necessary.**
- **Support insertions and deletions in $O(\log^2 n)$ time.**

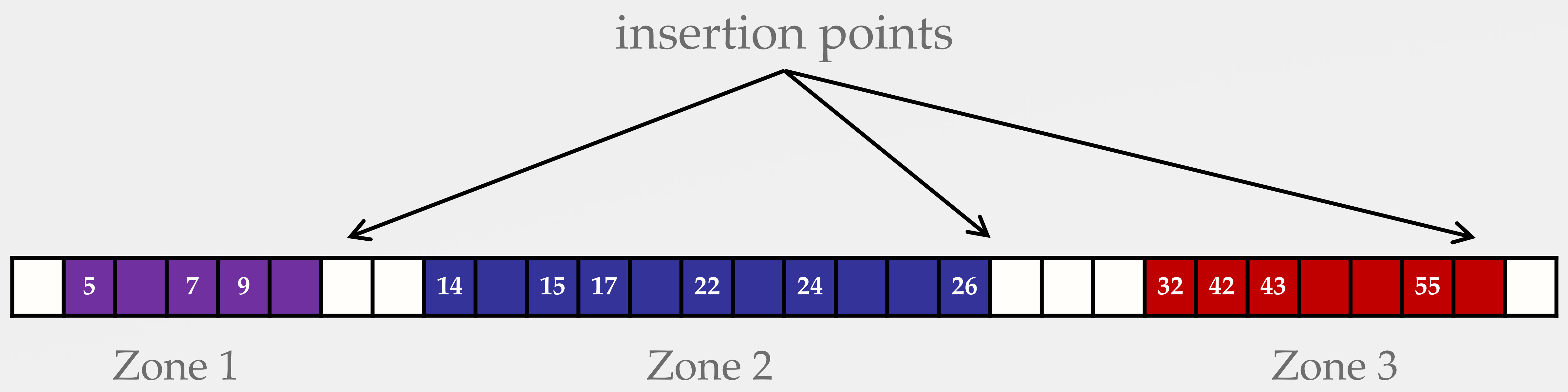
Sparse Table Data Structure



Maintain elements in an array:

- **Stored in order with gaps.**
- **Items rebalanced to make room when necessary.**
- **Support insertions and deletions in $O(\log^2 n)$ time.**

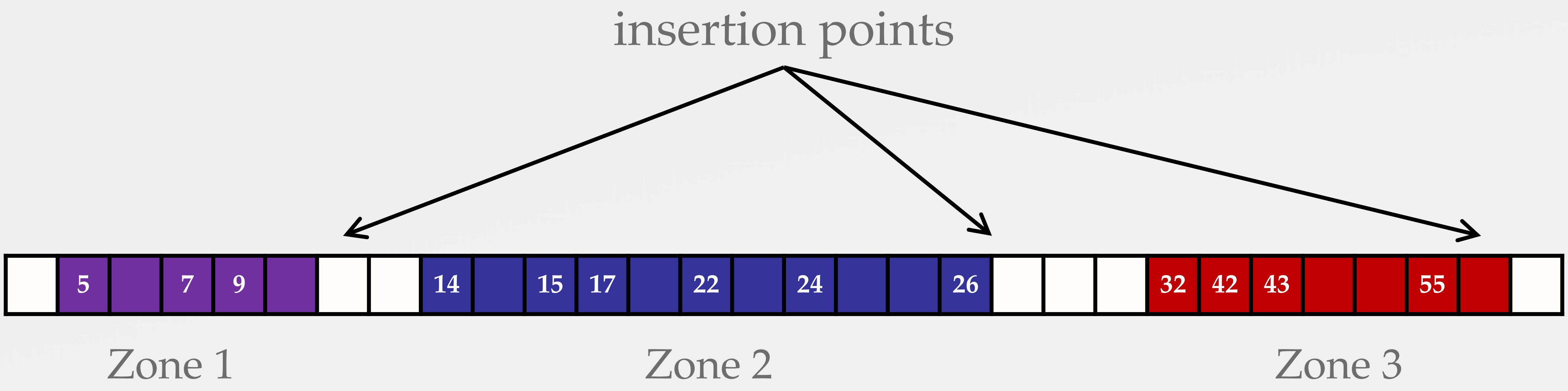
k-Cursor Data Structure



Maintain elements in an array:

- **Items grouped into k zones.**
- **Insert and delete only at end of zone.**
- **Support insertions and deletions in $O(\log^3 k)$ time.**

k-Cursor Data Structure



Maintain size classes:

- **Each zone stores one size class.**
- **Each zone determines boundaries for size class.**
- **Support insertions and deletions in $O(\log^3 \log \Delta)$ time.**

Data Structures to the Rescue

Setting:

- **Arbitrary length tasks.**
- **No constraints on when to schedule.**
- **Number of servers:** p
- **Unknown (!?!) allocation/reallocation cost.**

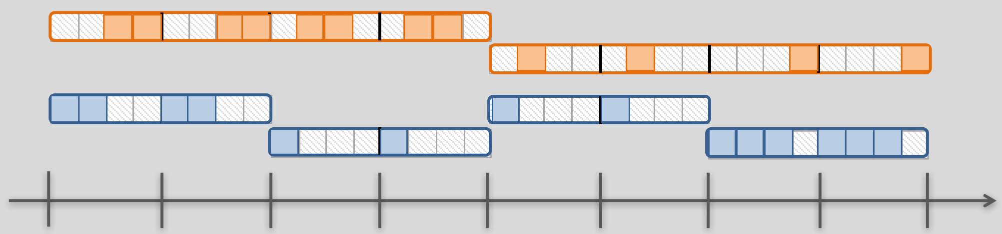
Goal: **Minimize sum-of-completion-times**

Key results:

For any subadditive, non-decreasing cost function:

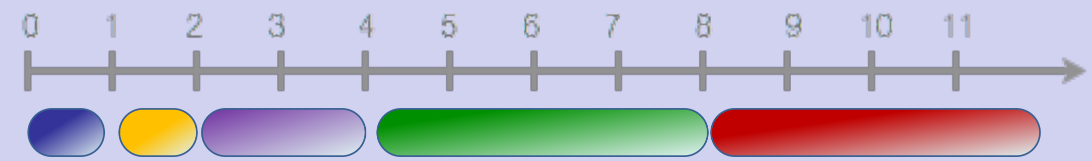
- **ReallocationCost = $O(\log^3 \log \Delta) \cdot$ AllocationCost**
- **SoCT= $O(1) \cdot$ OPT**

Windowed Feasibility



O(log*n) cost

Makespan



O(1) cost

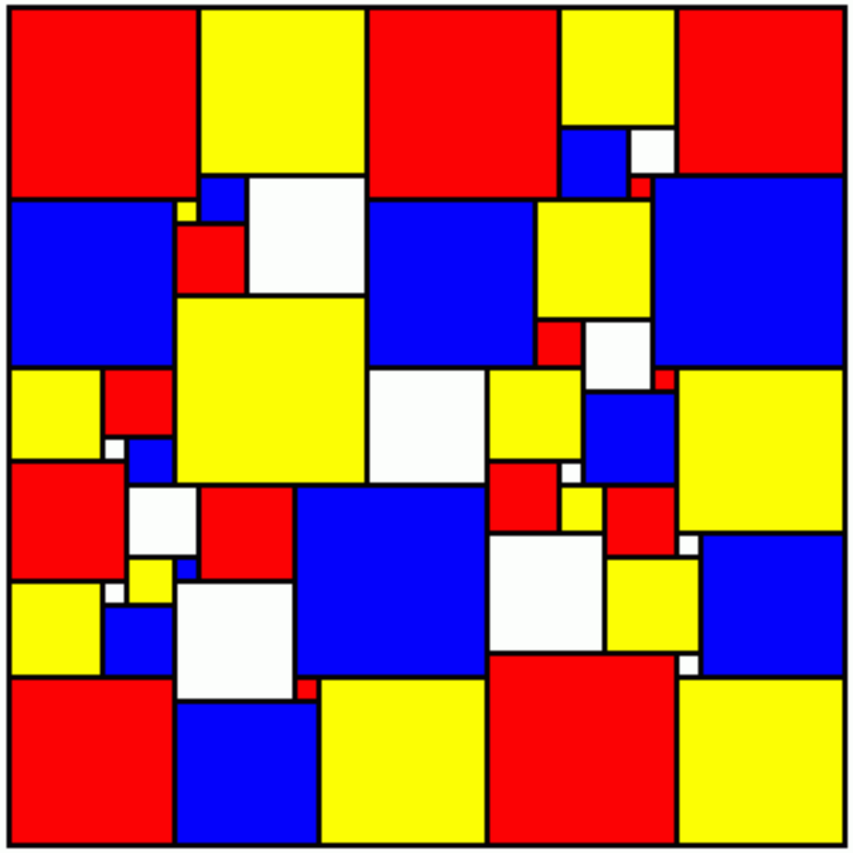
Sum-of-Completion Times

$O(\log^3 \log \Delta)$ cost


100 FEET HIGH
MONSTER
LIZARD!
FANTASTIC
SIGHTS LEAP
AT YOU IN...
2-DIMENSION
TERRORSAUR!
AMAZING! EXCITING! SPECTACULAR!
LAURA MARTIN HENRY DAVIES

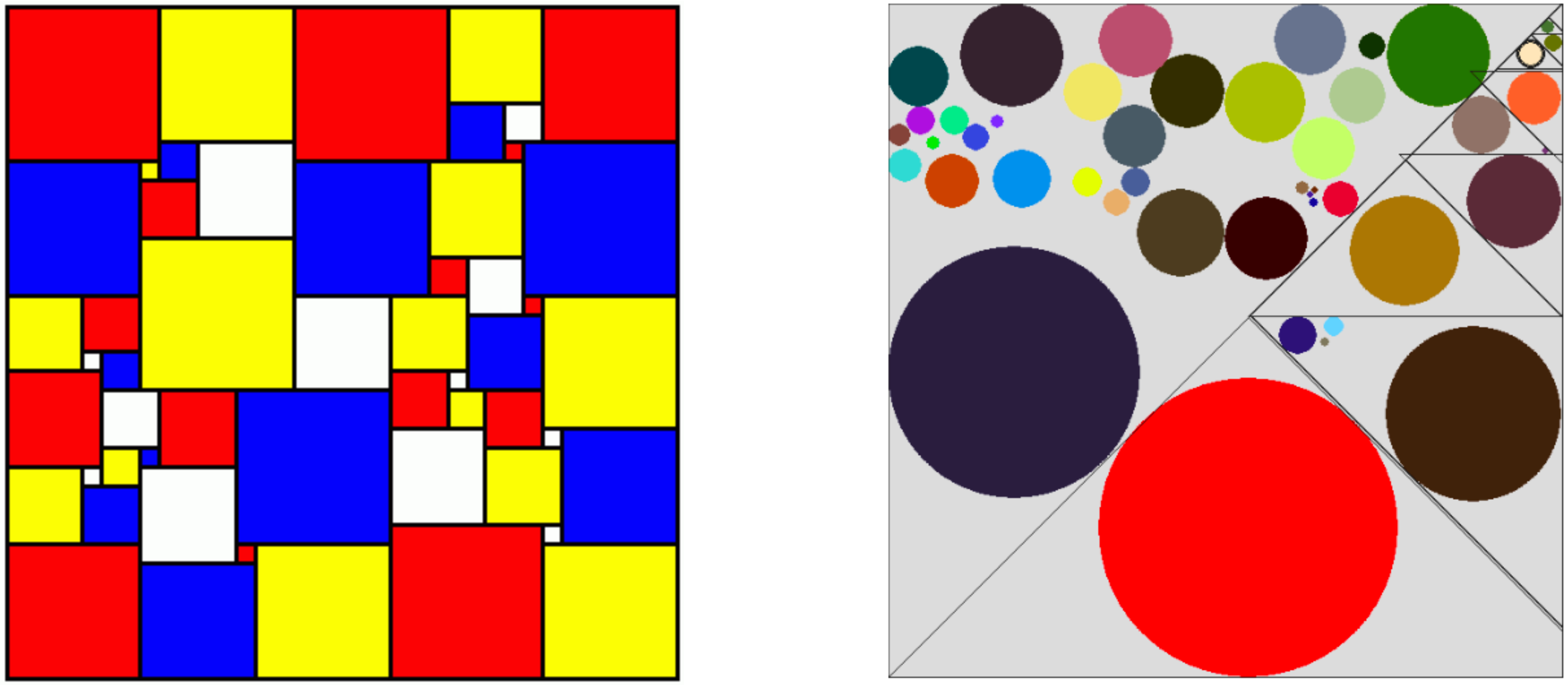
2D-Scheduling?

Online Square Packing



Support: insert/delete with reallocation

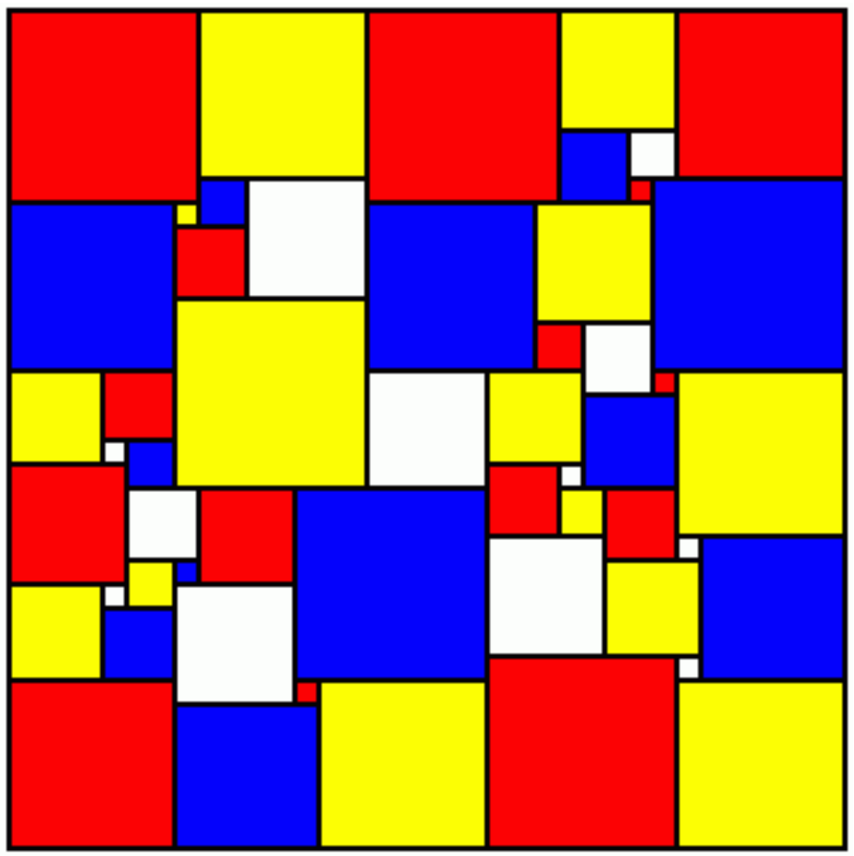
Online Circle / Square Packing

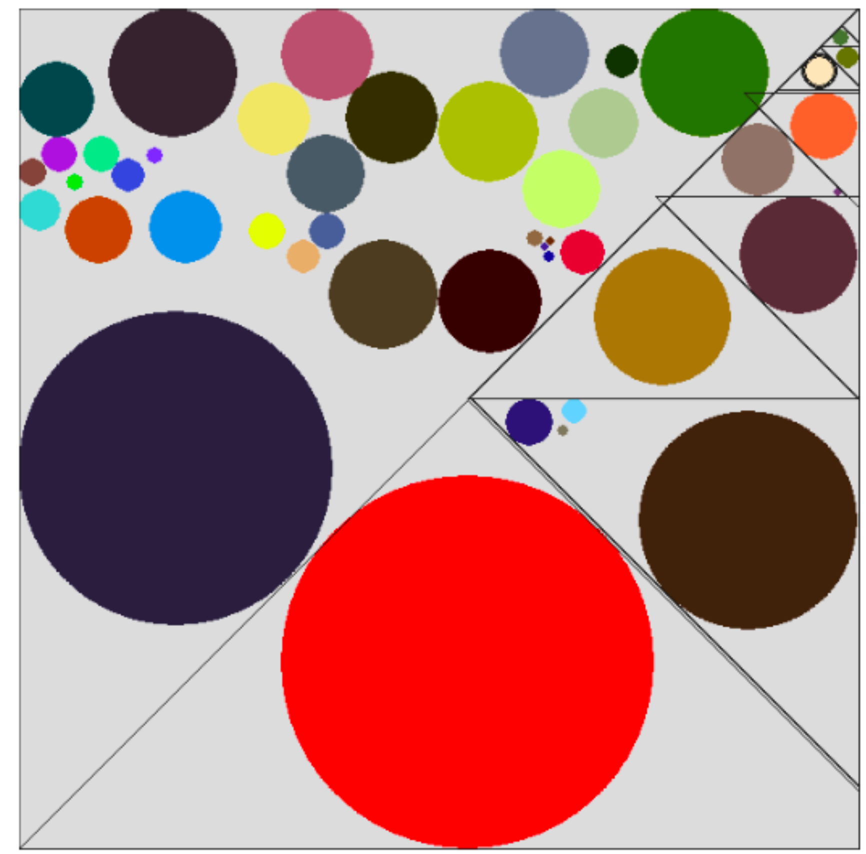


Support: insert/delete with reallocation

2D-Scheduling?

Online Circle / Square Packing





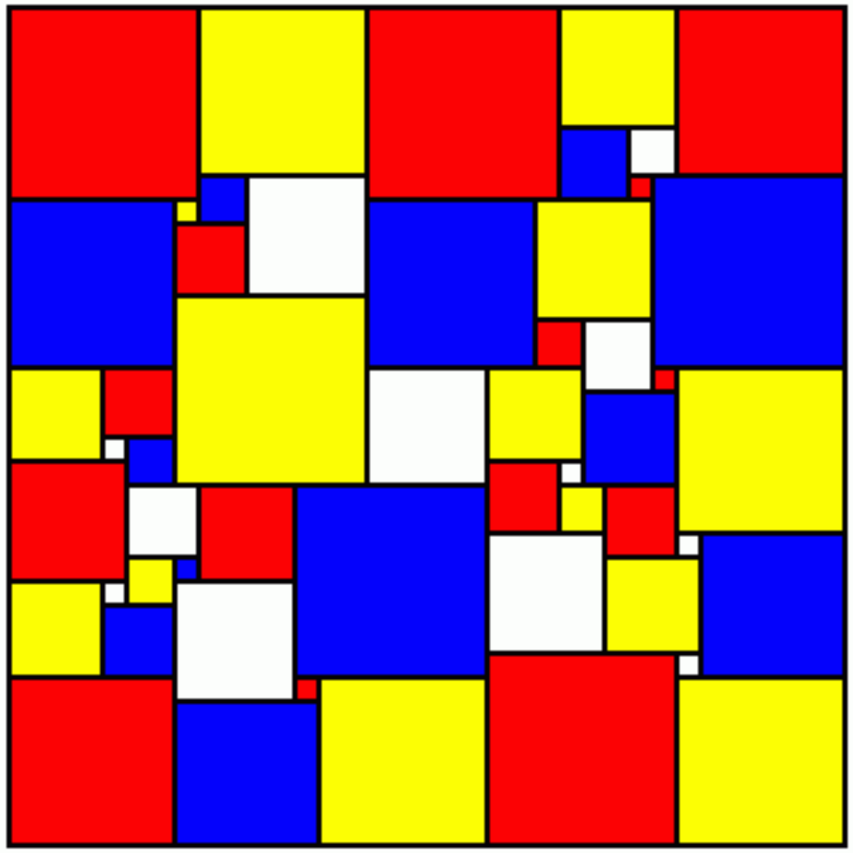
Current results:

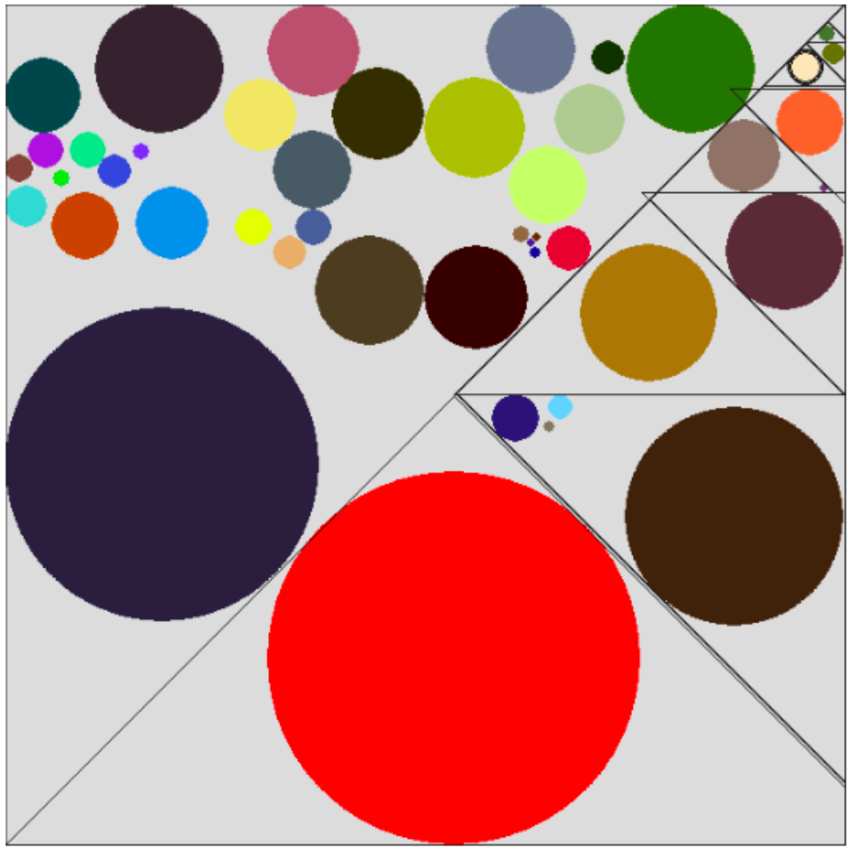
O(1) **efficiency**

O(log Δ)•Volume reallocation cost

2D-Scheduling?

Online Circle / Square Packing





Open questions:

- Unit cost reallocation?
- Cost oblivious?

Open Questions

Other questions:

Open Questions

Other questions:

Scheduling variants:

- Preemption?
- Flowtime?
- Stochastic job sizes?

Open Questions

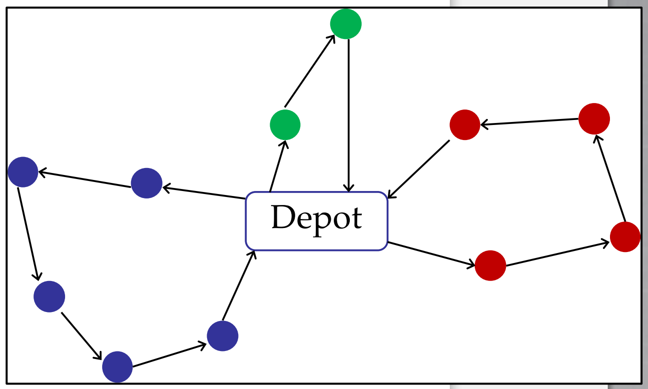
Other questions:

Cost metrics:

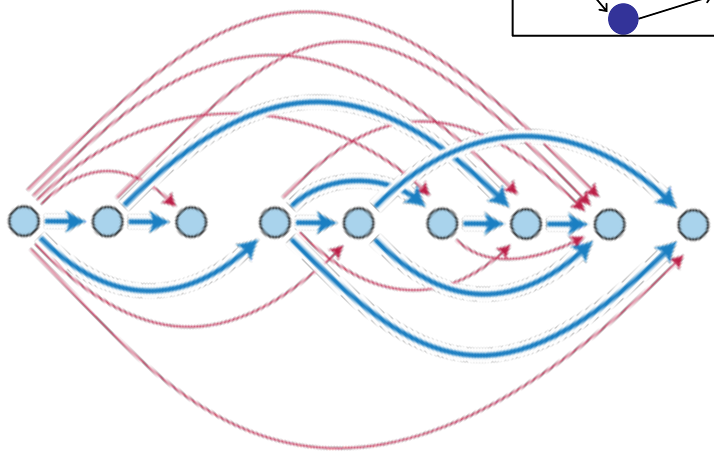
- Mixed costs?
- Migration vs. Local differentiation?
- Non-size-based costs?
- Stochastic costs?

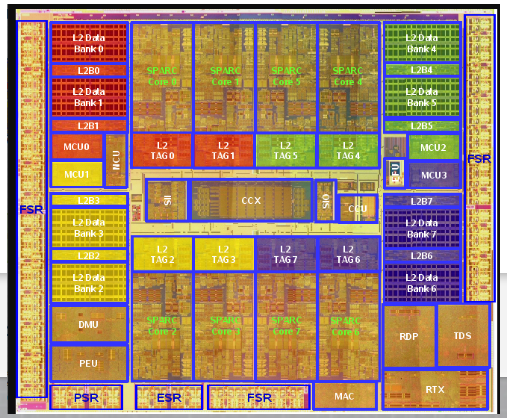
Other questions:

Problem variants:



- Scheduling routes / flows?
- Scheduling delivery routes?
- Scheduling DAGs?
- FPGA reconfiguration?





NUS
National University of Singapore

School of Computing

In search of: PhD students
Postdocs

Plans change.

"It does not do to leave a live dragon out of your calculations, if you live near him." (Tolkien)

"Unless commitment is made, there are only promises and hopes but no plans." (Drucker)

"No matter what the work you are doing, be always ready to drop it. And plan it so as to be able to leave it. (Tolstoy)

"In preparing for battle, I have always found that plans are useless, but planning is indispensable." (Eisenhower)

"There cannot be a crisis next week. My schedule is already full." (Kissinger)

"Give me six hours to chop down a tree and I will spend the first four sharpening the axe." (Lincoln)

"If you don't know where you are going, you'll end up someplace else." (Berra)

"Let our advance worrying become advance thinking and planning." (Churchill)

"A goal without a plan is just a wish." (Saint-Exupery)

"Dreaming, after all, is a form of planning." (Steinem)

"If you fail to plan, you are planning to fail." (Benjamin Franklin)

Be prepared.