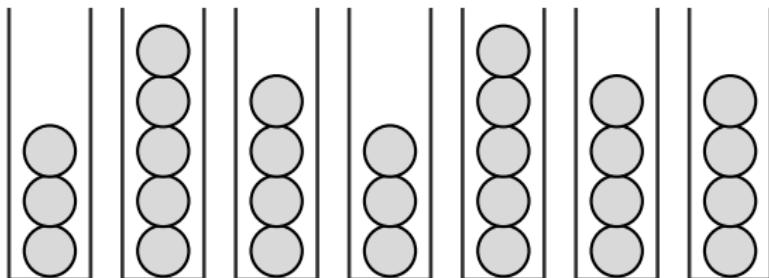


History-Independent Load Balancing



Michael A. Bender

Stony Brook University

William Kuszmaul

CMU

Elaine Shi

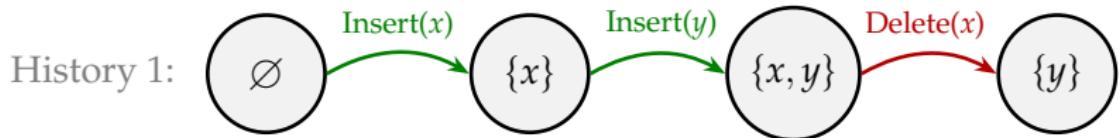
CMU

Rose Silver

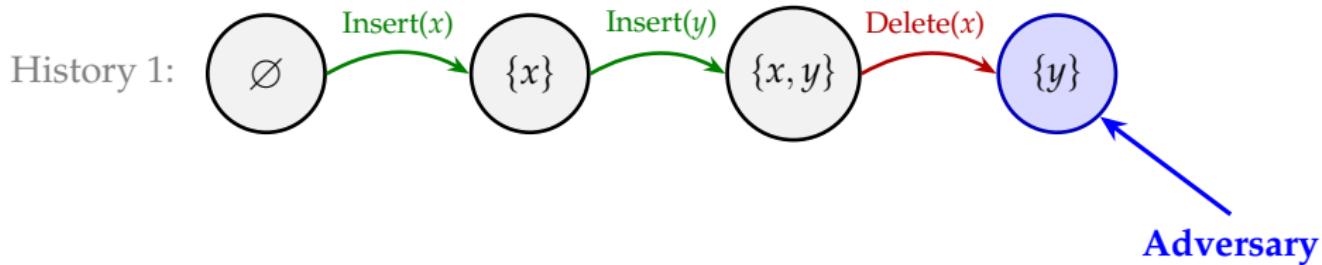
CMU

HISTORY-INDEPENDENT DATA STRUCTURES

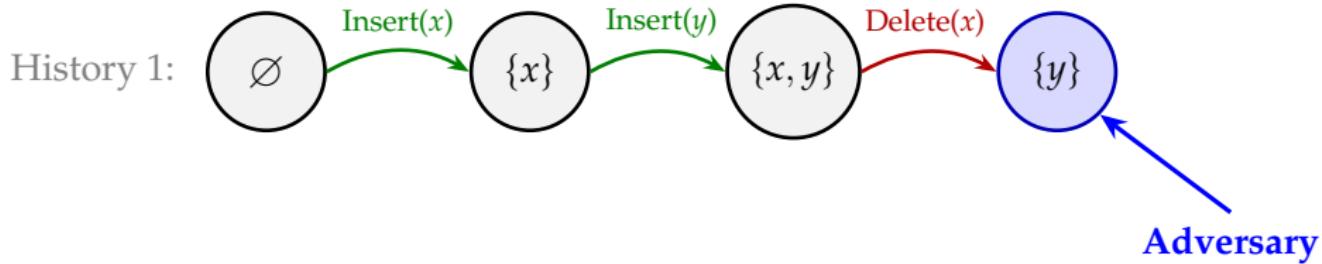
HISTORY-INDEPENDENT DATA STRUCTURES



HISTORY-INDEPENDENT DATA STRUCTURES



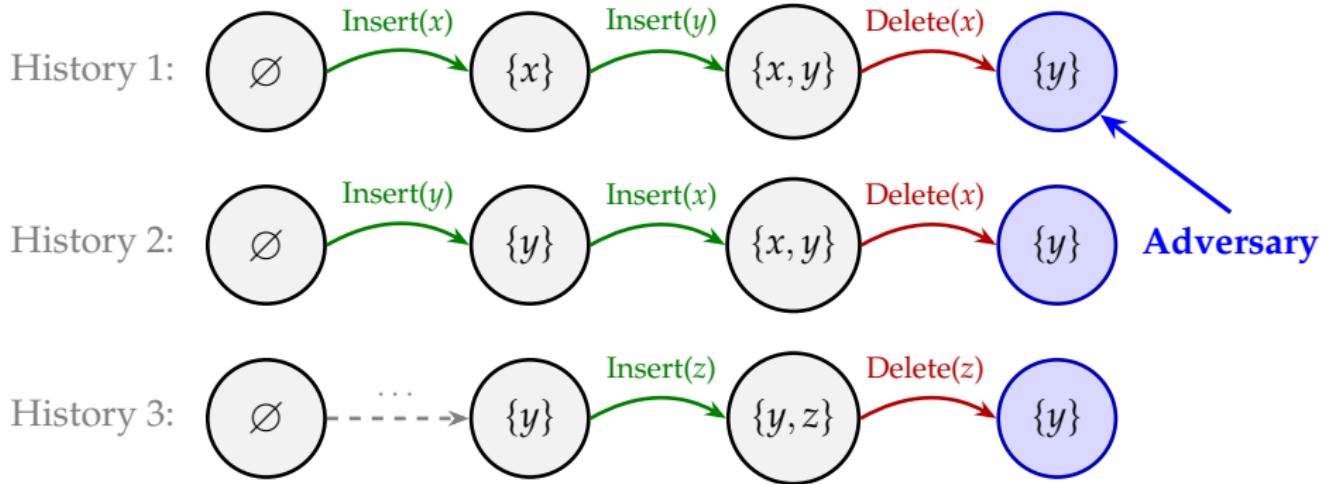
HISTORY-INDEPENDENT DATA STRUCTURES



History Independence (Micciancio '97, Naor & Teague '01)

- ▶ The state reveals only the current elements—**not the history of operations**.

HISTORY-INDEPENDENT DATA STRUCTURES



History Independence (Micciancio '97, Naor & Teague '01)

- ▶ The state reveals only the current elements—not the history of operations.

HISTORY INDEPENDENT DATA STRUCTURES

A History of Applications

Hash tables, trees, memory allocation, PMAs, graph algorithms, cache-oblivious data structures, and more.

Micciancio '97, Naor & Teague '01, Buchbinder & Petrank '03, Molnar et al. '06, Blelloch & Golovin '07,
Moran et al. '07, Naor et al. '08, Golovin '08–'10, Bajaj & Sion '13, Roche et al. '15, Bender et al. '16

HISTORY INDEPENDENT DATA STRUCTURES

A History of Applications

Hash tables, trees, memory allocation, PMAs, graph algorithms, cache-oblivious data structures, and more.

Micciancio '97, Naor & Teague '01, Buchbinder & Petrank '03, Molnar et al. '06, Blelloch & Golovin '07,
Moran et al. '07, Naor et al. '08, Golovin '08–'10, Bajaj & Sion '13, Roche et al. '15, Bender et al. '16

Yet some basic questions remain open.

HISTORY INDEPENDENT DATA STRUCTURES

A History of Applications

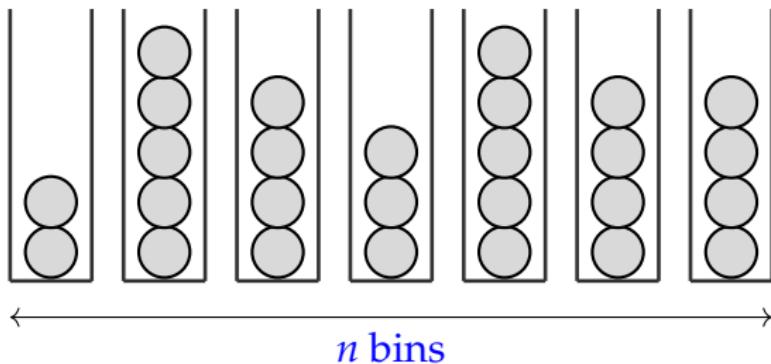
Hash tables, trees, memory allocation, PMAs, graph algorithms, cache-oblivious data structures, and more.

Micciancio '97, Naor & Teague '01, Buchbinder & Petrank '03, Molnar et al. '06, Blelloch & Golovin '07,
Moran et al. '07, Naor et al. '08, Golovin '08–'10, Bajaj & Sion '13, Roche et al. '15, Bender et al. '16

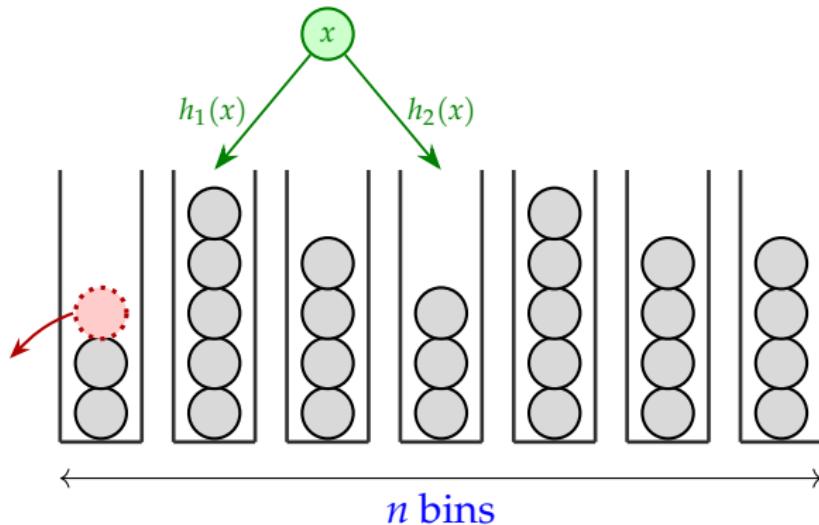
Yet some basic questions remain open.

This work: History-Independent Load Balancing

TWO-CHOICE LOAD BALANCING

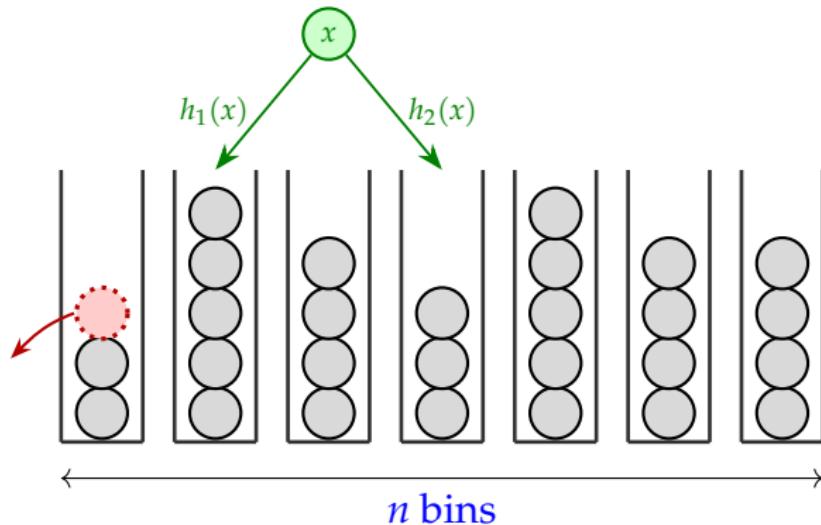


TWO-CHOICE LOAD BALANCING



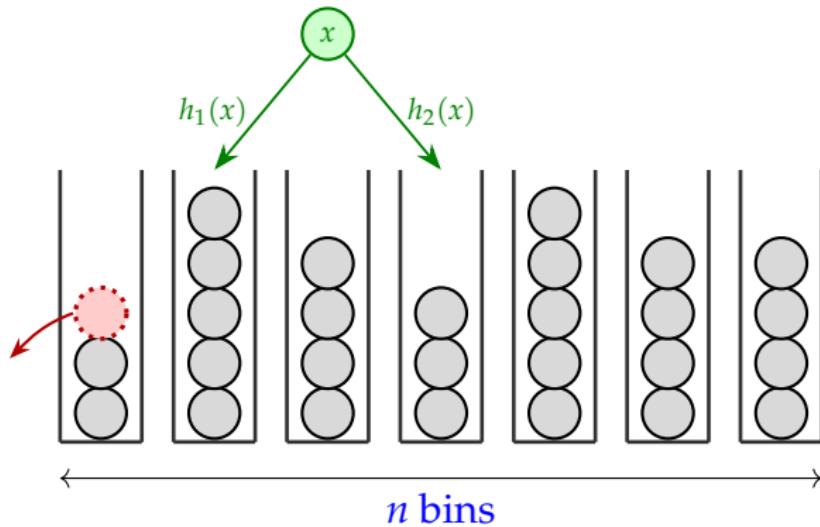
- ▶ Balls are **inserted**/**deleted**, with up to m present at a time.

TWO-CHOICE LOAD BALANCING



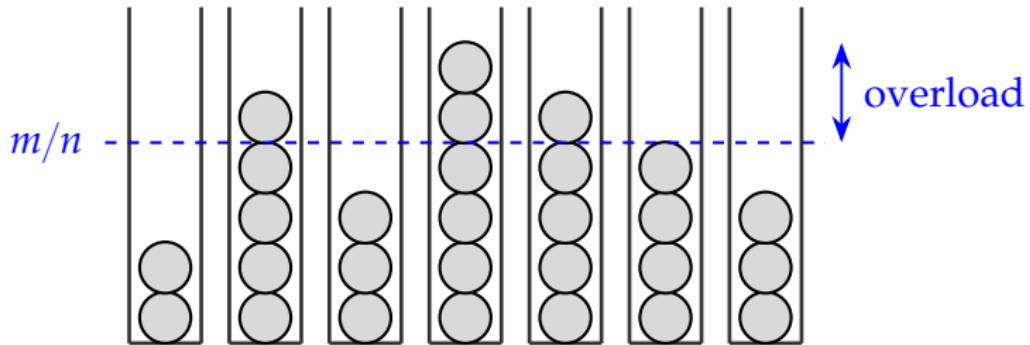
- ▶ Balls are **inserted**/**deleted**, with up to m present at a time.
- ▶ Each ball has two random bins where it can go.

TWO-CHOICE LOAD BALANCING



- ▶ Balls are **inserted**/**deleted**, with up to m present at a time.
- ▶ Each ball has two random bins where it can go.
- ▶ We must maintain a valid assignment of balls to bins.

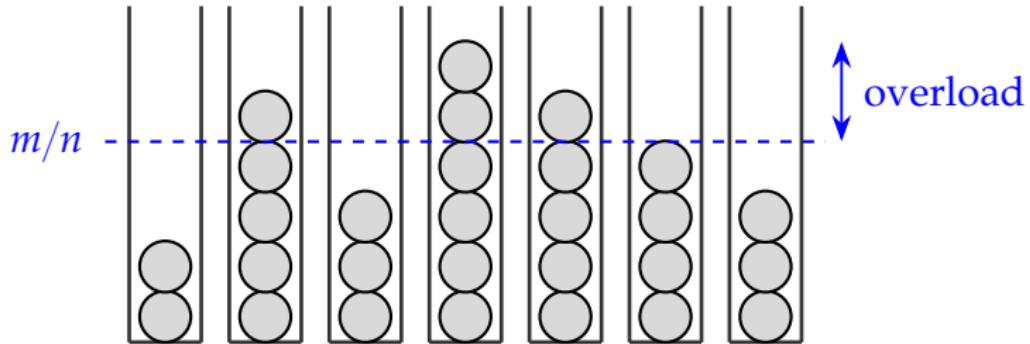
TWO GOALS



Minimize Overload:

- ▶ i.e., the amount by which the fullest bin exceeds m/n .

TWO GOALS



Minimize Overload:

- ▶ i.e., the amount by which the fullest bin exceeds m/n .

Minimize Recourse:

- ▶ i.e., the number of balls moved around on any given insertion/deletion.

PUTTING IT ALL TOGETHER

PUTTING IT ALL TOGETHER

History-Independent Load Balancing:

PUTTING IT ALL TOGETHER

History-Independent Load Balancing:

- ▶ For all sets S of balls: If the current set is S , then the assignment is always A_S .

PUTTING IT ALL TOGETHER

History-Independent Load Balancing:

- ▶ For all sets S of balls: If the current set is S , then the assignment is always A_S .

Question: Does there exist a **history-independent** solution with small **recourse** and small **overload**?

PUTTING IT ALL TOGETHER

History-Independent Load Balancing:

- ▶ For all sets S of balls: If the current set is S , then the assignment is always A_S .

Question: Does there exist a history-independent solution with small recourse and small overload?

Our Main Result: There exists a history-independent solution with:

- ▶ High probability overload $O(1)$
- ▶ Expected recourse $O(\log \log(m/n))$

PAST WORK (NOT HISTORY INDEPENDENT)

Overload	Recourse	Reference	Caveats
$O(\log \log n)$	0	[ABKU '94] [BCSV '00]	insertion-only
$O(1)$	$O(\log(m/n))$	[Dietzfelbinger, Weidling '07]	insertion-only
$\tilde{O}(\sqrt{m/n})$	$O(1)$	[Frieze, Petti '18]	insertion-only
$O(\log(m/n))$	0	[Bansal, Kuszmaul '22]	no reinsertions
$O(1)$	$O(m/n)$	[Dietzfelbinger, Weidling '07]	

PAST WORK (NOT HISTORY INDEPENDENT)

Overload	Recourse	Reference	Caveats
$O(\log \log n)$	0	[ABKU '94] [BCSV '00]	insertion-only
$O(1)$	$O(\log(m/n))$	[Dietzfelbinger, Weidling '07]	insertion-only
$\tilde{O}(\sqrt{m/n})$	$O(1)$	[Frieze, Petti '18]	insertion-only
$O(\log(m/n))$	0	[Bansal, Kuszmaul '22]	no reinsertions
$O(1)$	$O(m/n)$	[Dietzfelbinger, Weidling '07]	
$O(1)$	$O(\log \log(m/n))$	[This Paper]	

PAST WORK (NOT HISTORY INDEPENDENT)

Overload	Recourse	Reference	Caveats
$O(\log \log n)$	0	[ABKU '94] [BCSV '00]	insertion-only
$O(1)$	$O(\log(m/n))$	[Dietzfelbinger, Weidling '07]	insertion-only
$\tilde{O}(\sqrt{m/n})$	$O(1)$	[Frieze, Petti '18]	insertion-only
$O(\log(m/n))$	0	[Bansal, Kuszmaul '22]	no reinsertions
$O(1)$	$O(m/n)$	[Dietzfelbinger, Weidling '07]	
$O(1)$	$O(\log \log(m/n))$	[This Paper]	

If we want overload $O(1)$, our result is a new state of the art!

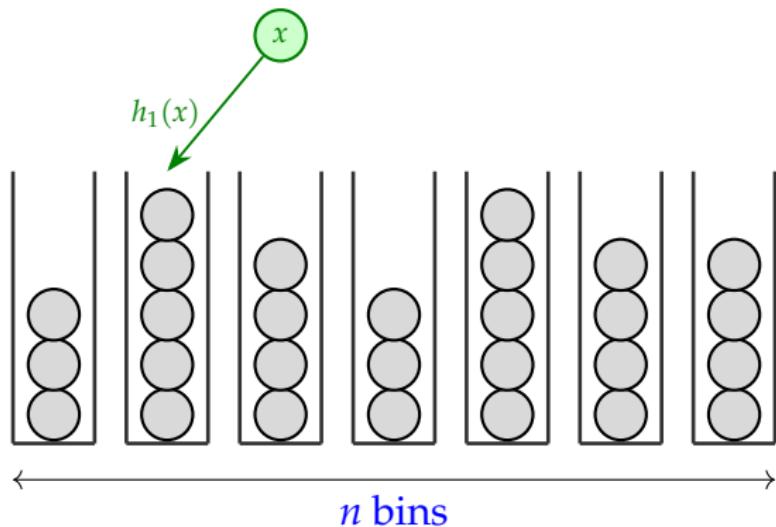
REST OF TALK: A SIMPLE WARMUP

Theorem: There exists a history-independent solution with:

- ▶ High-probability overload $\Theta(1)$ $O(\log \log n)$.
- ▶ Expected recourse $\Theta(\log \log(m/n))$ $O(m/n)$.

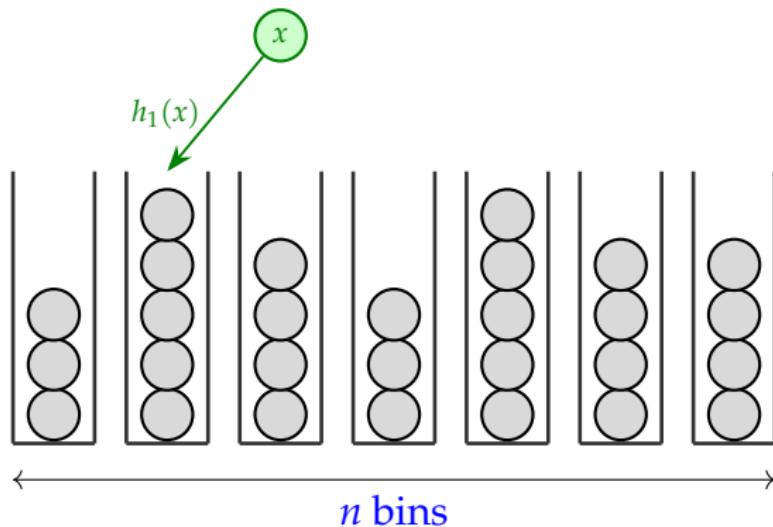
BASELINE 1: THE SINGLE-CHOICE STRATEGY

To insert a ball x , just put it in bin $h_1(x)$:



BASELINE 1: THE SINGLE-CHOICE STRATEGY

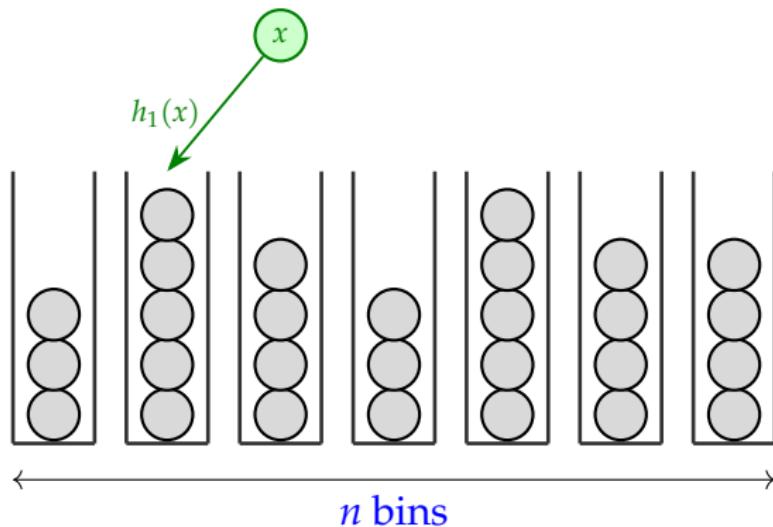
To insert a ball x , just put it in bin $h_1(x)$:



- ▶ This is history-independent ✓

BASELINE 1: THE SINGLE-CHOICE STRATEGY

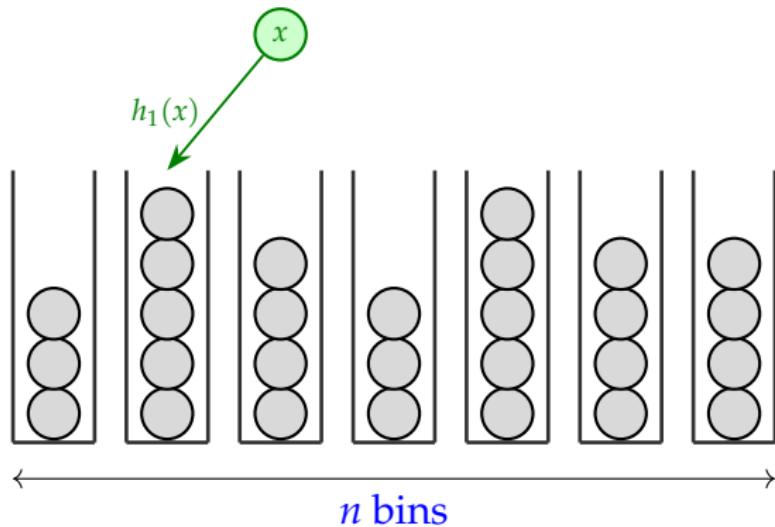
To insert a ball x , just put it in bin $h_1(x)$:



- ▶ This is history-independent ✓
- ▶ The recourse is 0 ✓

BASELINE 1: THE SINGLE-CHOICE STRATEGY

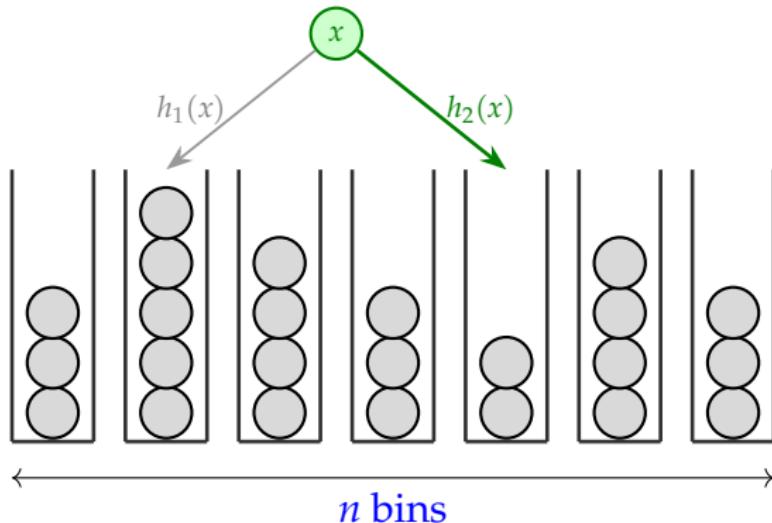
To insert a ball x , just put it in bin $h_1(x)$:



- ▶ This is history-independent ✓
- ▶ The recourse is 0 ✓
- ▶ But... the overload is huge, roughly $\sqrt{m/n}$ ✗

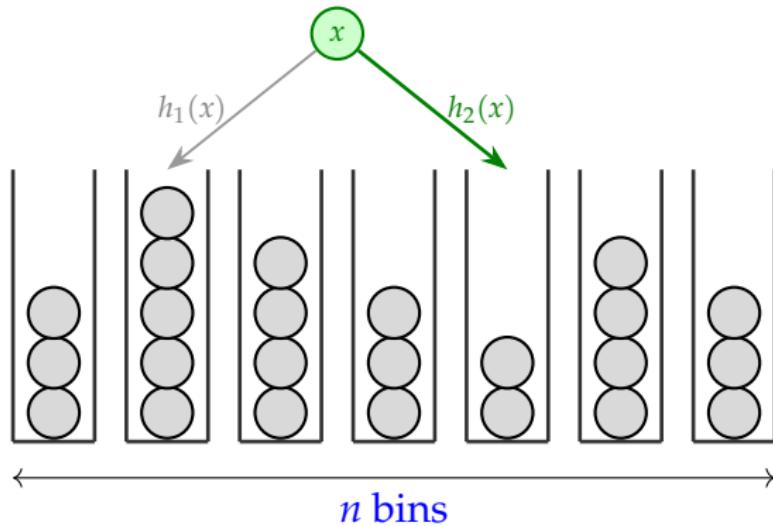
BASELINE 2: GREEDY INSERTIONS

To insert a ball x , put it in the **emptier** of its choices:



BASELINE 2: GREEDY INSERTIONS

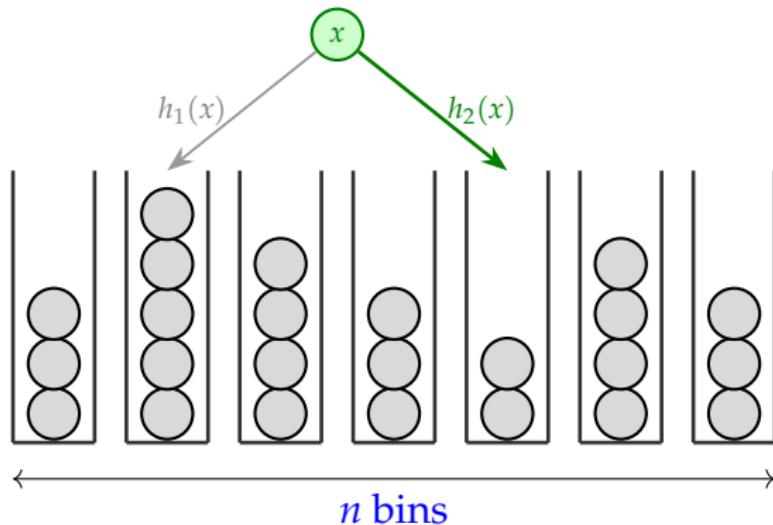
To insert a ball x , put it in the **emptier** of its choices:



- ▶ This is **not** history-independent \times

BASELINE 2: GREEDY INSERTIONS

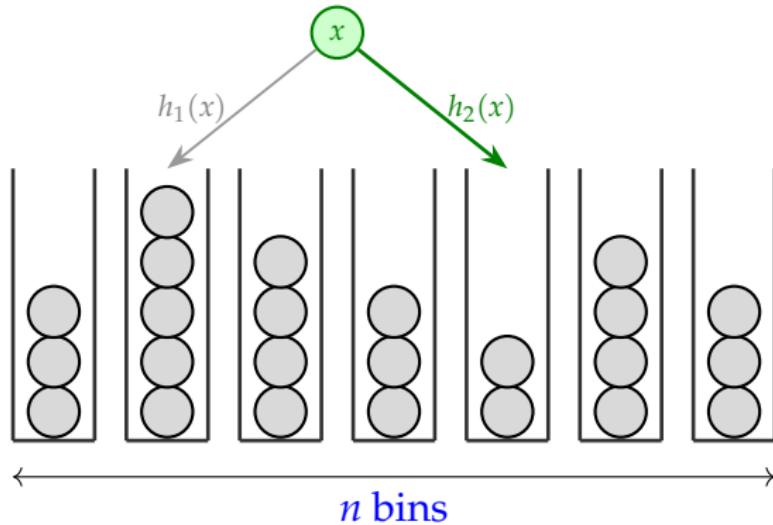
To insert a ball x , put it in the **emptier** of its choices:



- ▶ This is **not** history-independent ✗
- ▶ The recourse is 0 ✓

BASELINE 2: GREEDY INSERTIONS

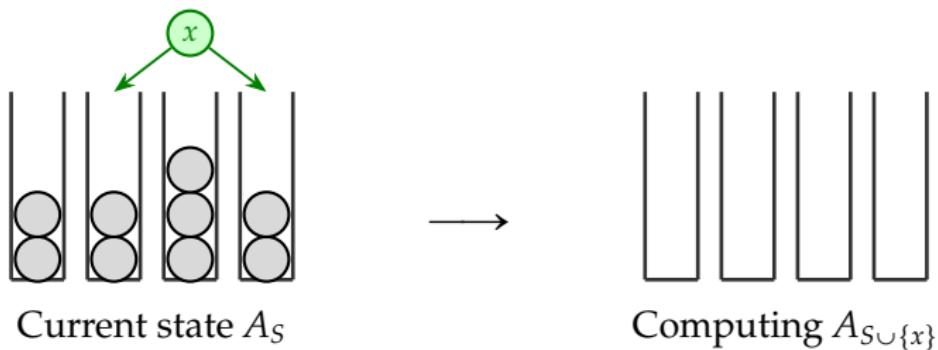
To insert a ball x , put it in the **emptier** of its choices:



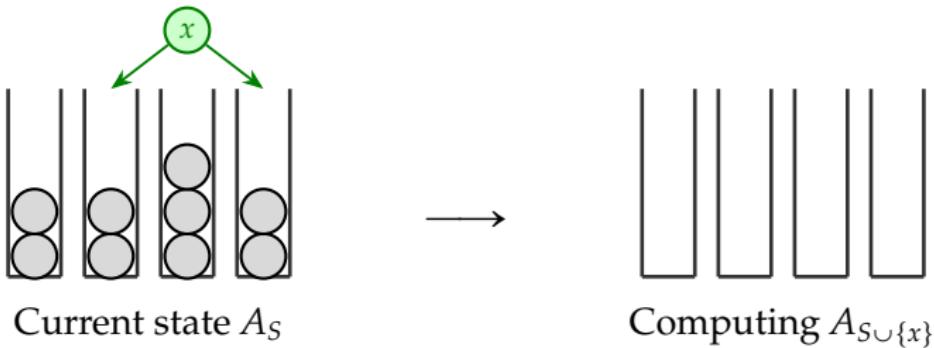
- ▶ This is **not** history-independent ✗
- ▶ The recourse is 0 ✓
- ▶ In the insertion-only case, the overload is $O(\log \log n)$ ✓
[Berenbrink, Czumaj, Steger, and Vöcking '00]

WARMUP: HISTORY-INDEPENDENT GREEDY

WARMUP: HISTORY-INDEPENDENT GREEDY



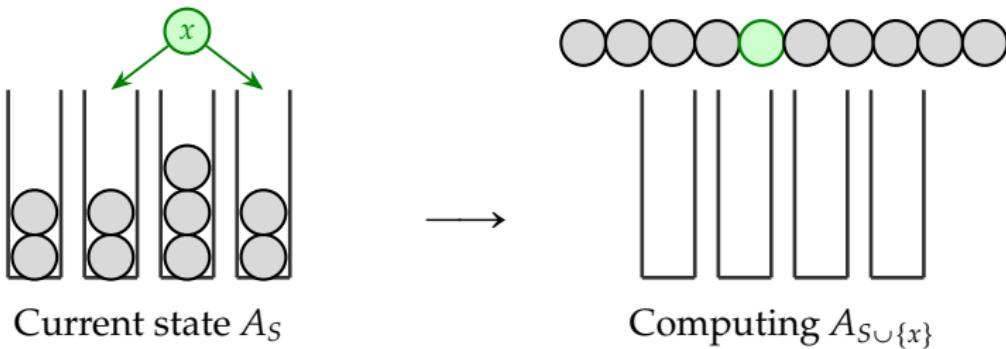
WARMUP: HISTORY-INDEPENDENT GREEDY



To compute $A_{S \cup \{x\}}$:

1. Empty out the bins.
2. Sort the balls in $S \cup \{x\}$ to get x_1, x_2, \dots
3. Insert the balls in sorted order using greedy.

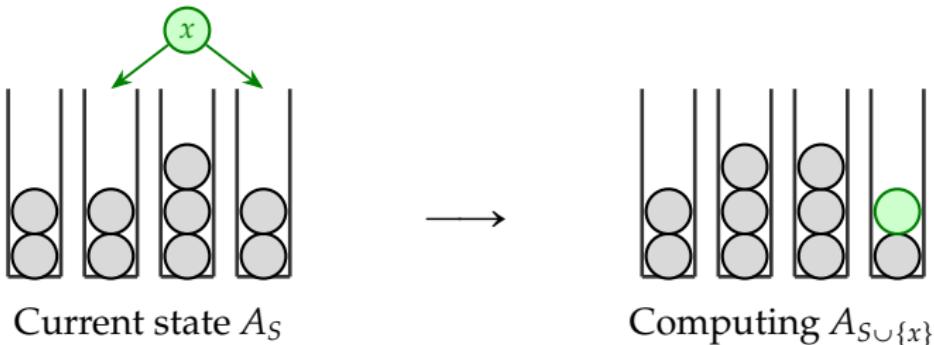
WARMUP: HISTORY-INDEPENDENT GREEDY



To compute $A_{S \cup \{x\}}$:

1. Empty out the bins.
2. Sort the balls in $S \cup \{x\}$ to get x_1, x_2, \dots
3. Insert the balls in sorted order using greedy.

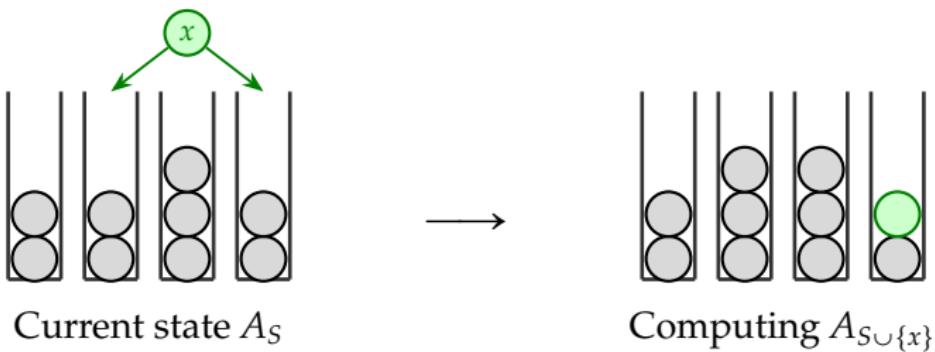
WARMUP: HISTORY-INDEPENDENT GREEDY



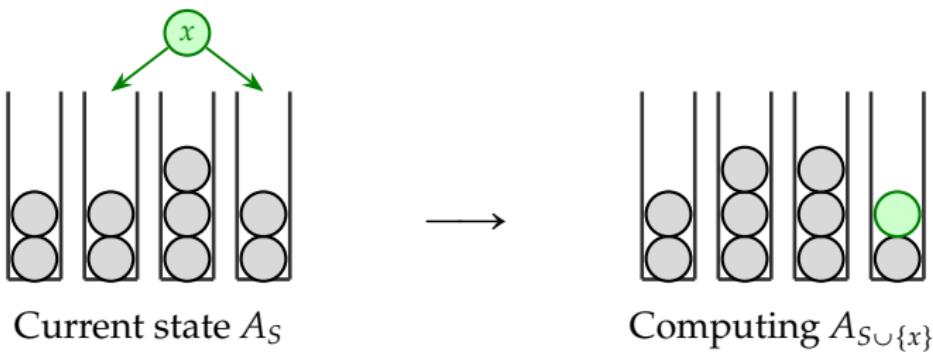
To compute $A_{S \cup \{x\}}$:

1. Empty out the bins.
2. Sort the balls in $S \cup \{x\}$ to get x_1, x_2, \dots
3. Insert the balls in sorted order using greedy.

ANALYZING HISTORY-INDEPENDENT GREEDY

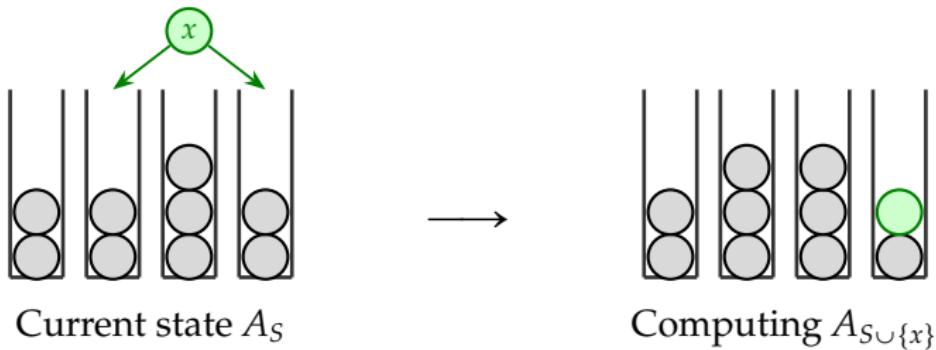


ANALYZING HISTORY-INDEPENDENT GREEDY



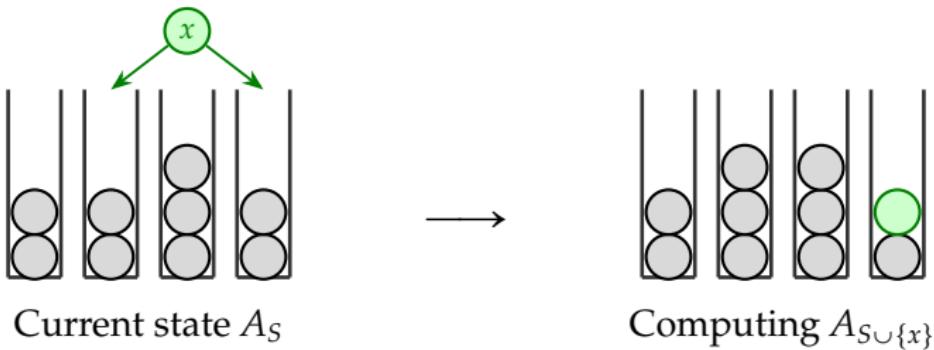
- ▶ The algorithm is history independent ✓

ANALYZING HISTORY-INDEPENDENT GREEDY



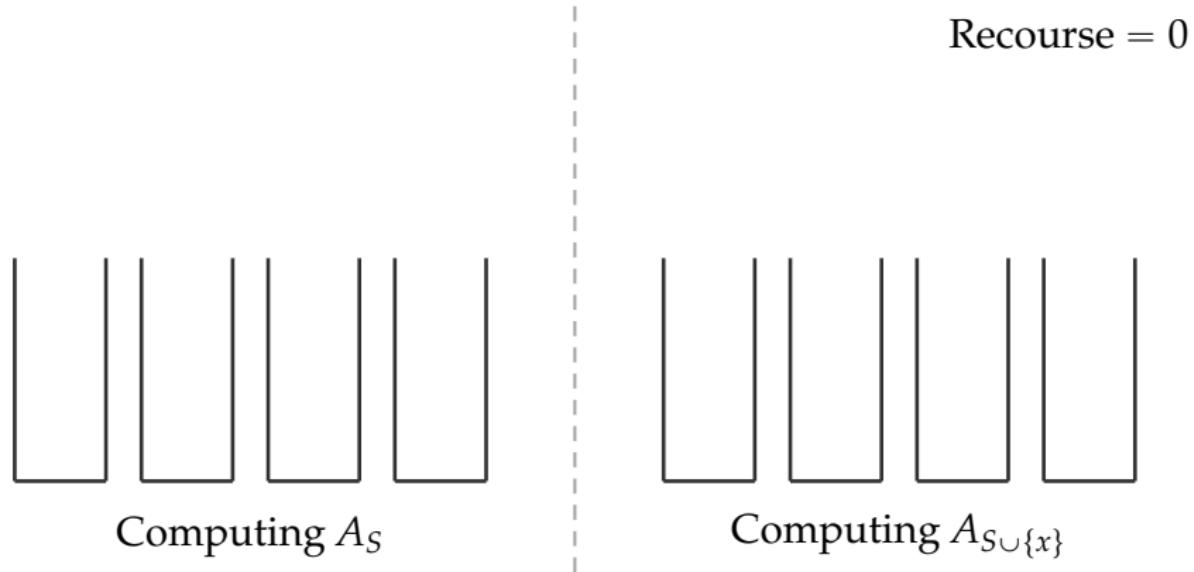
- ▶ The algorithm is history independent ✓
- ▶ The overload is $O(\log \log n)$ ✓

ANALYZING HISTORY-INDEPENDENT GREEDY



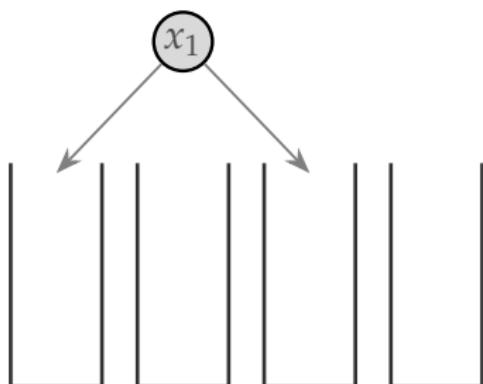
- ▶ The algorithm is history independent ✓
- ▶ The overload is $O(\log \log n)$ ✓
- ▶ What is the recourse?

ANALYZING THE RECOURSE

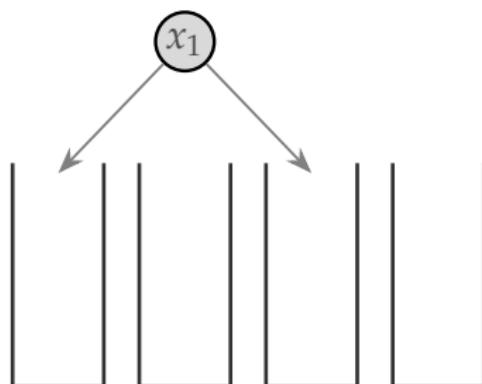


How many balls change assignments between A_S and $A_{S \cup \{x\}}$?

ANALYZING THE RECOURSE



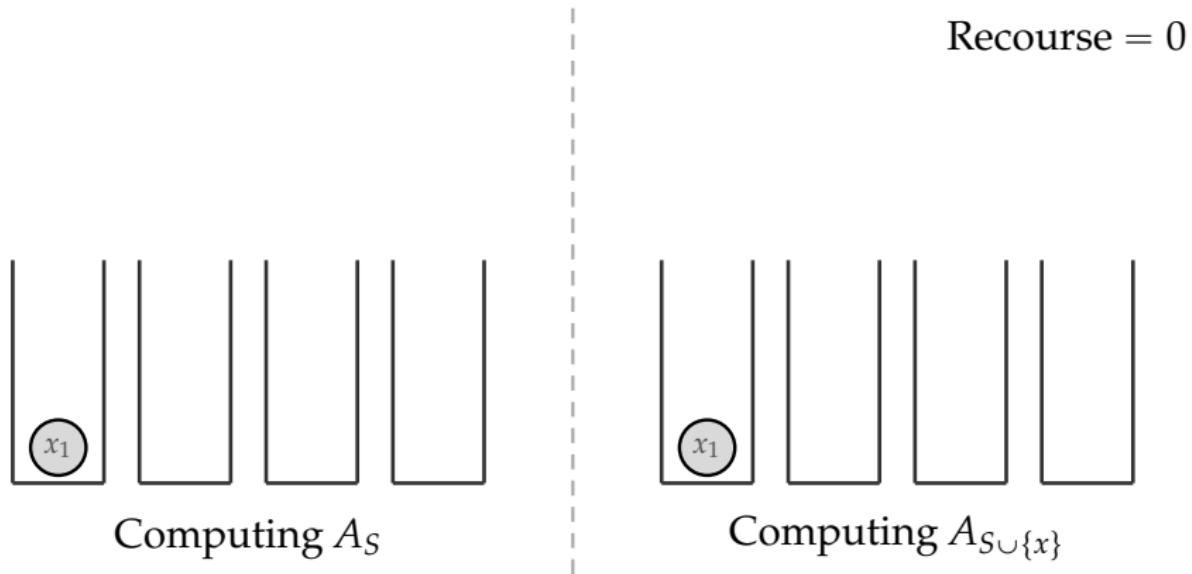
Computing A_S



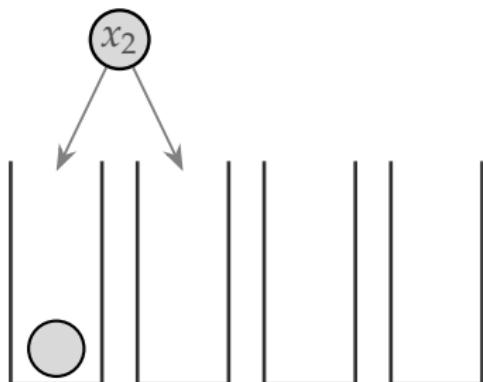
Computing $A_{S \cup \{x\}}$

Recourse = 0

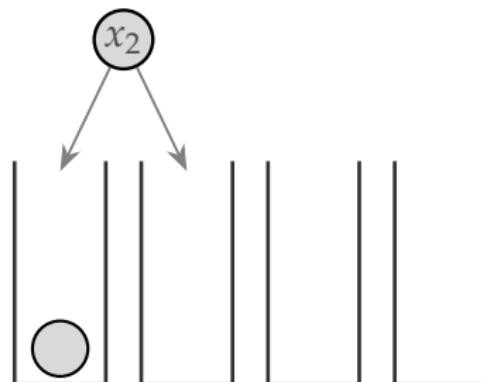
ANALYZING THE RECOURSE



ANALYZING THE RECOURSE



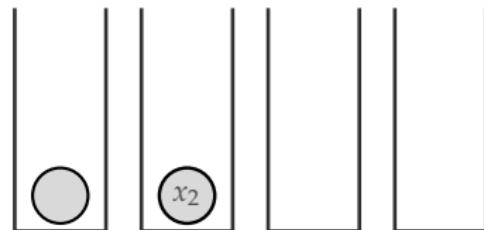
Computing A_S



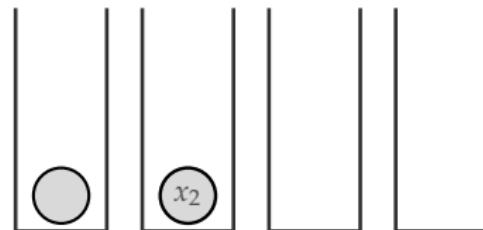
Computing $A_{S \cup \{x\}}$

Recourse = 0

ANALYZING THE RECOURSE



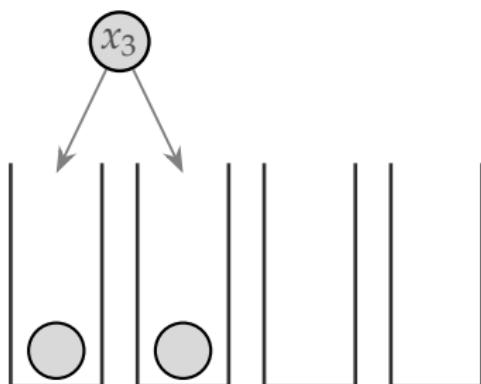
Computing A_S



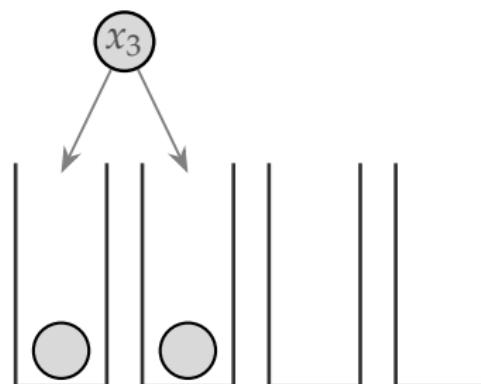
Computing $A_{S \cup \{x\}}$

Recourse = 0

ANALYZING THE RECOURSE



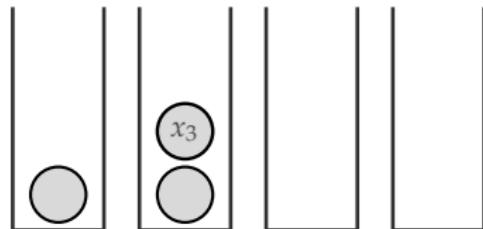
Computing A_S



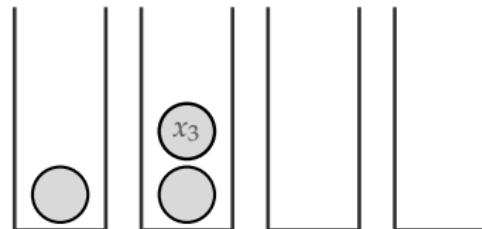
Computing $A_{S \cup \{x\}}$

Recourse = 0

ANALYZING THE RECOURSE



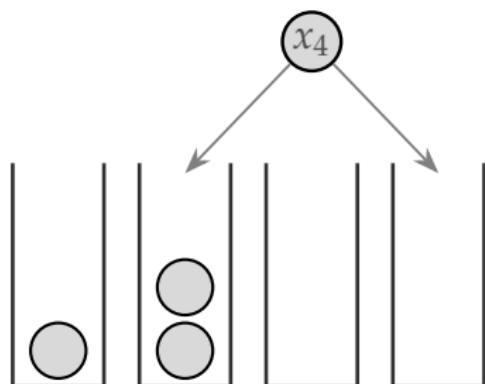
Computing A_S



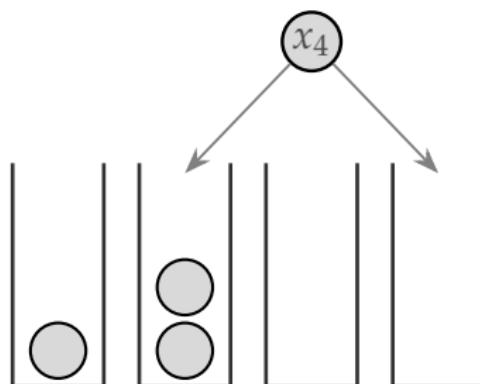
Computing $A_{S \cup \{x\}}$

Recourse = 0

ANALYZING THE RECOURSE



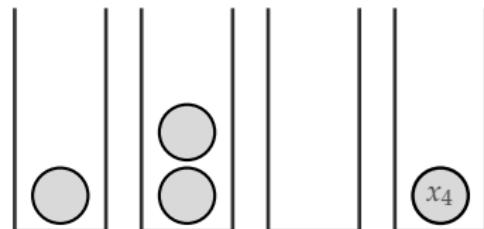
Computing A_S



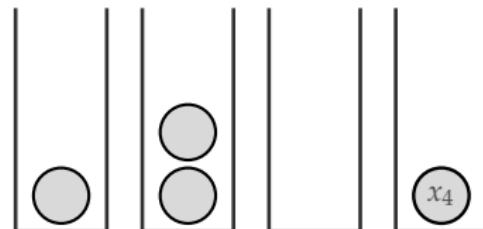
Computing $A_{S \cup \{x\}}$

Recourse = 0

ANALYZING THE RECOURSE



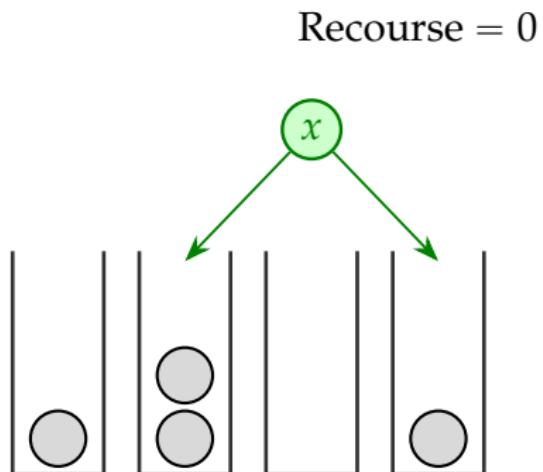
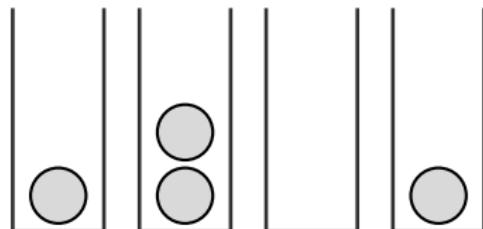
Computing A_S



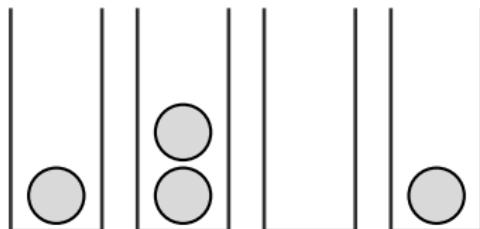
Computing $A_{S \cup \{x\}}$

Recourse = 0

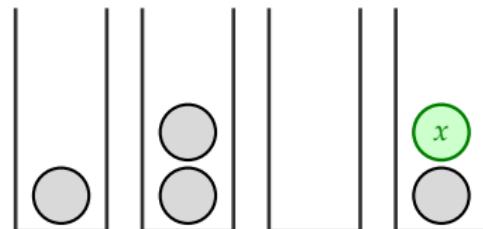
ANALYZING THE RECOURSE



ANALYZING THE RECOURSE



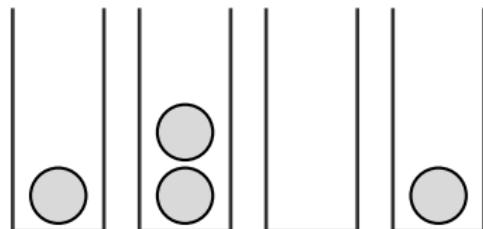
Computing A_S



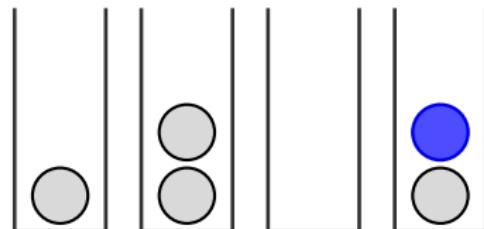
Computing $A_{S \cup \{x\}}$

Recourse = 0

ANALYZING THE RECOURSE



Computing A_S

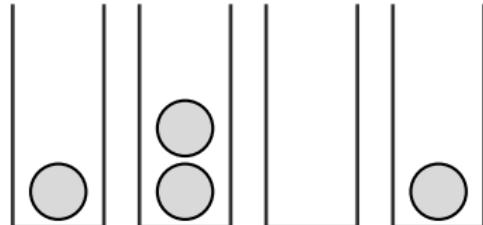


Computing $A_{S \cup \{x\}}$

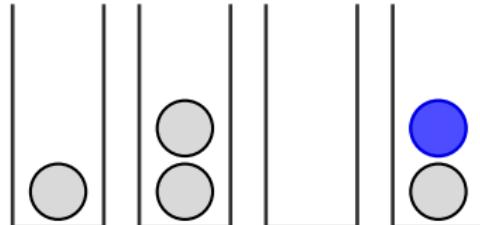
Subsequent balls will experience either:

Recourse = 0

ANALYZING THE RECOURSE



Computing A_S



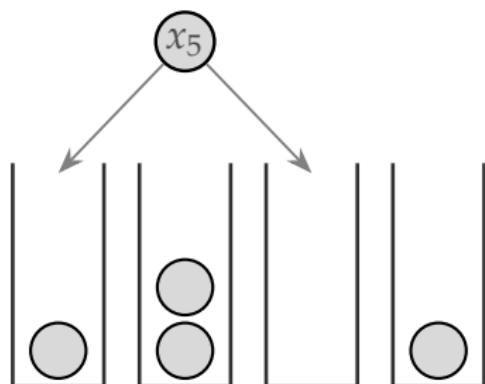
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

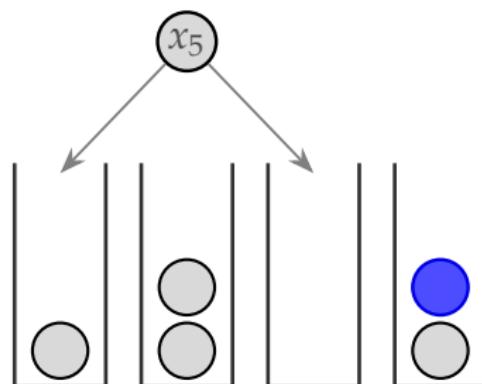
1. No recourse

$$\text{Recourse} = 0$$

ANALYZING THE RECOURSE



Computing A_S



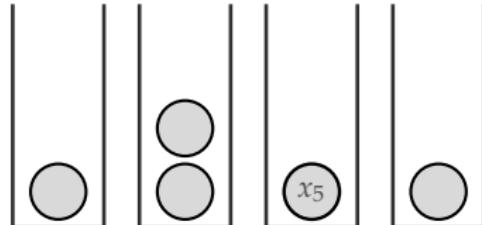
Computing $A_{S \cup \{x\}}$

Future insertions will experience either:

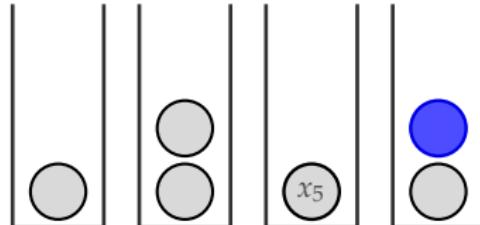
1. No recourse

Recourse = 0

ANALYZING THE RECOURSE



Computing A_S

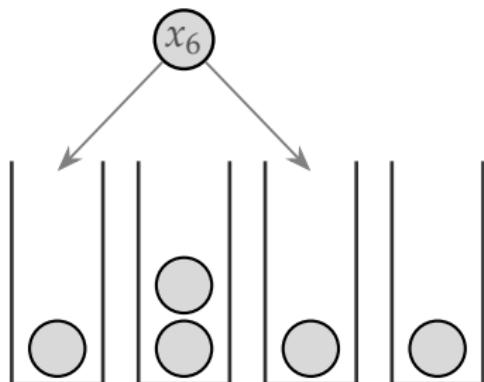


Computing $A_{S \cup \{x\}}$

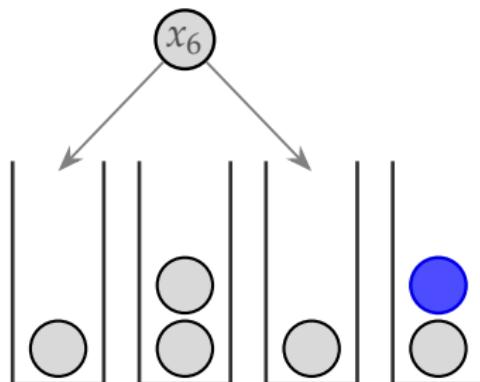
Subsequent balls will experience either:

1. No recourse

ANALYZING THE RECOURSE



Computing A_S



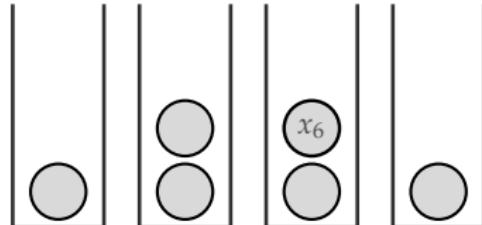
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

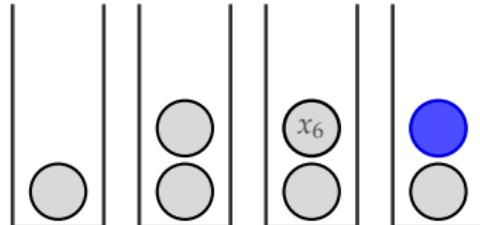
1. No recourse

$$\text{Recourse} = 0$$

ANALYZING THE RECOURSE



Computing A_S

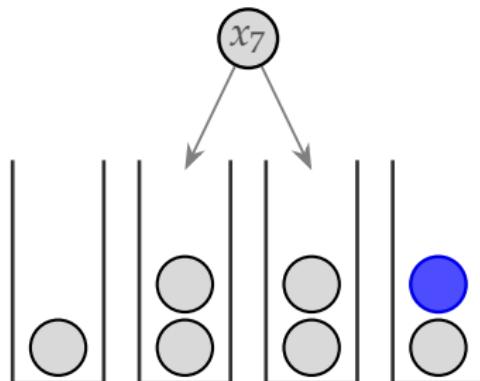
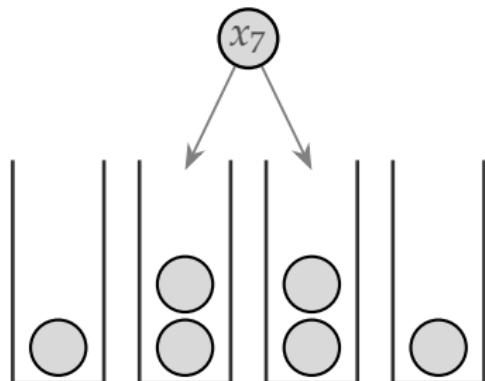


Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse

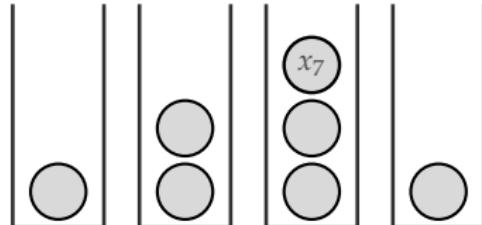
ANALYZING THE RECOURSE



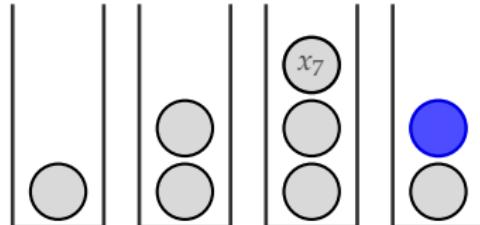
Subsequent balls will experience either:

1. No recourse

ANALYZING THE RECOURSE



Computing A_S



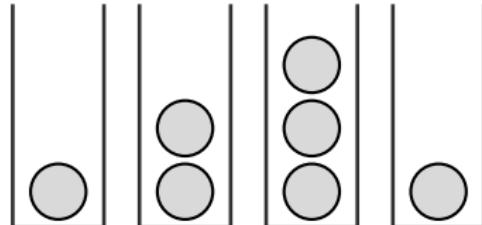
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

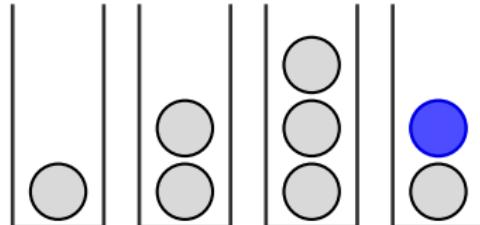
1. No recourse

$$\text{Recourse} = 0$$

ANALYZING THE RECOURSE



Computing A_S



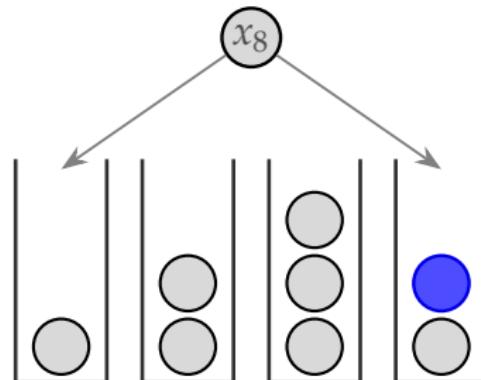
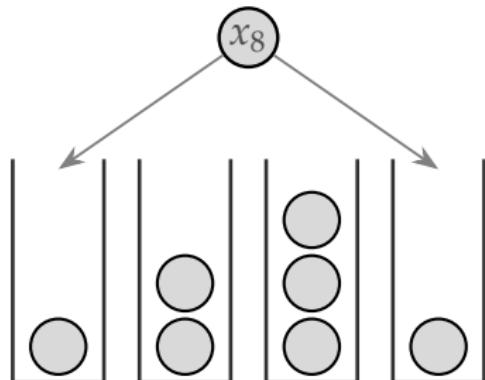
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse
2. Recourse

$$\text{Recourse} = 0$$

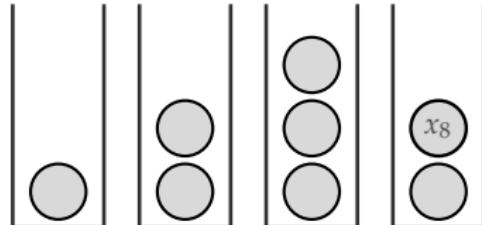
ANALYZING THE RECOURSE



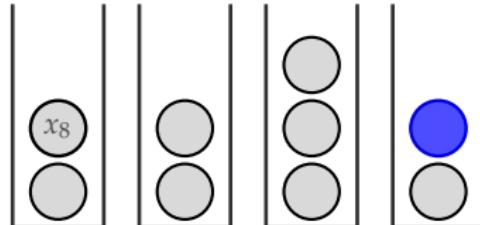
Subsequent balls will experience either:

1. No recourse
2. Recourse

ANALYZING THE RECOURSE



Computing A_S



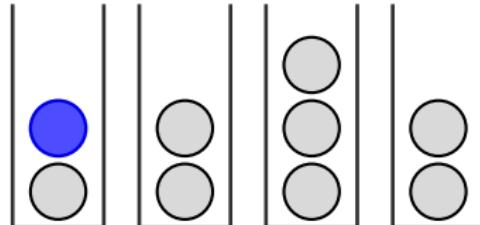
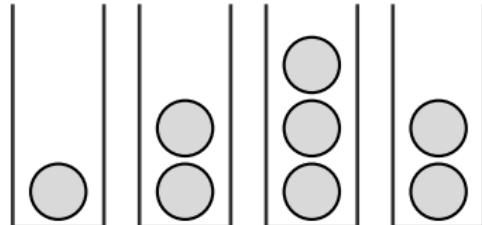
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse
2. Recourse

Recourse = 1

ANALYZING THE RECOURSE

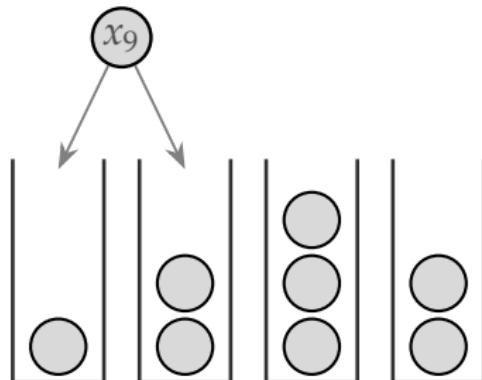


Recourse = 1

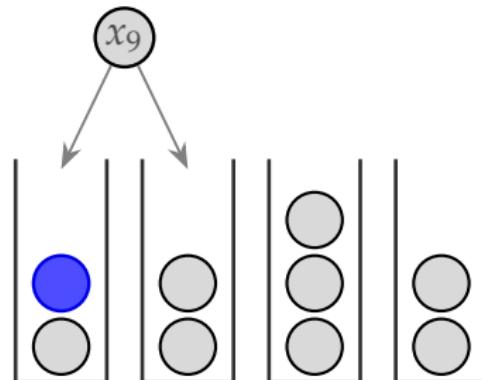
Subsequent balls will experience either:

1. No recourse
2. Recourse

ANALYZING THE RECOURSE



Computing A_S



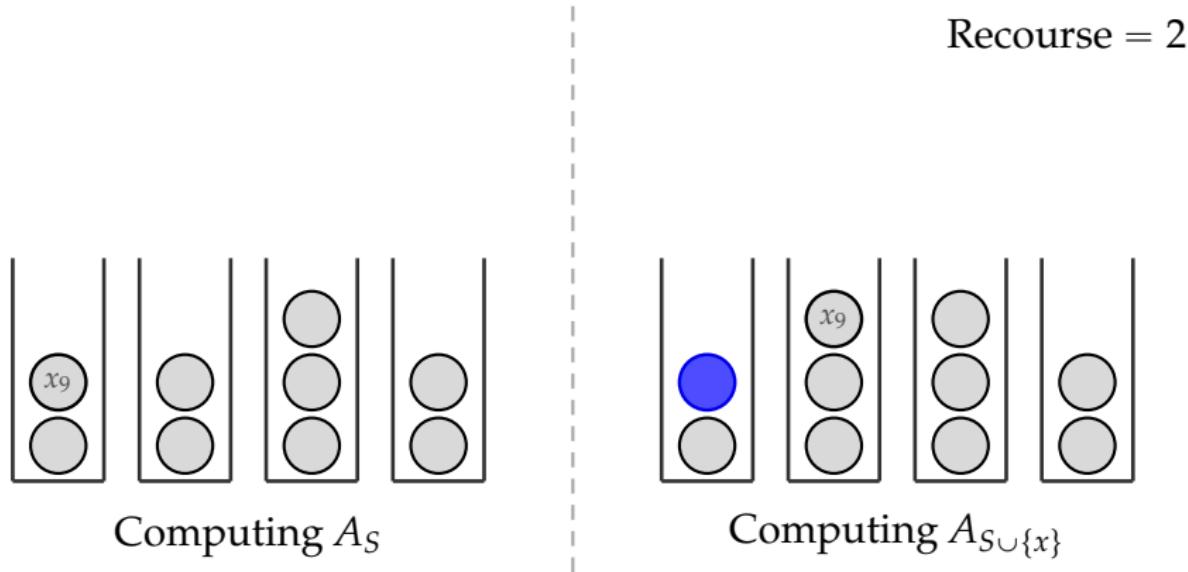
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse
2. Recourse

Recourse = 1

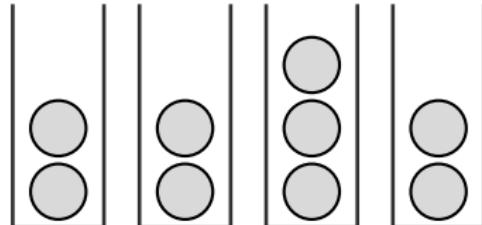
ANALYZING THE RECOURSE



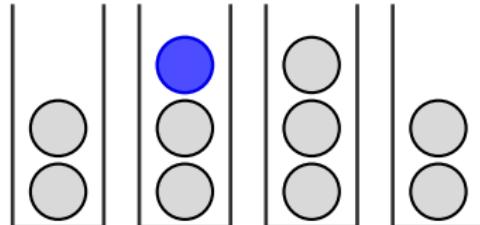
Subsequent balls will experience either:

1. No recourse
2. Recourse

ANALYZING THE RECOURSE



Computing A_S



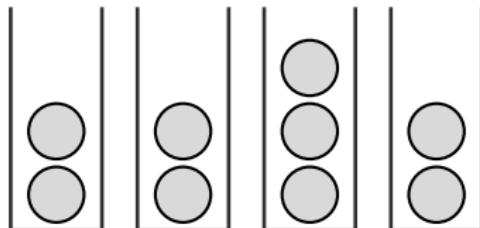
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

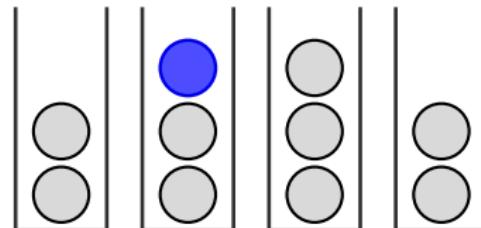
1. No recourse
2. Recourse

Recourse = 2

ANALYZING THE RECOURSE



Computing A_S

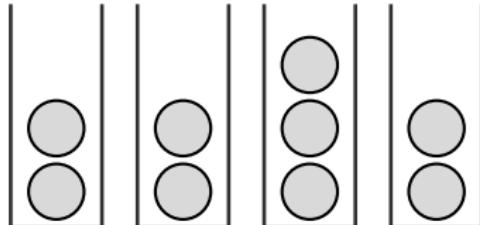


Computing $A_{S \cup \{x\}}$

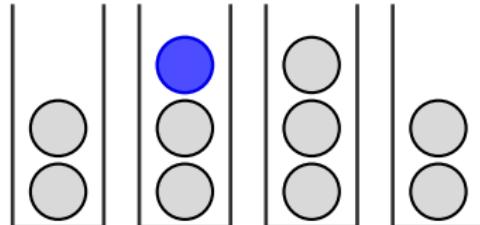
Two key observations:

Recourse = 2

ANALYZING THE RECOURSE



Computing A_S



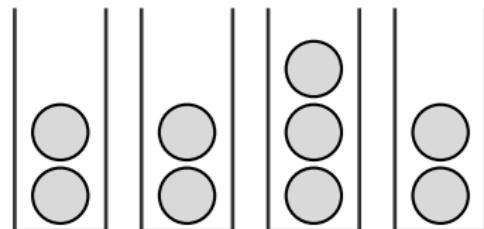
Computing $A_{S \cup \{x\}}$

Two key observations:

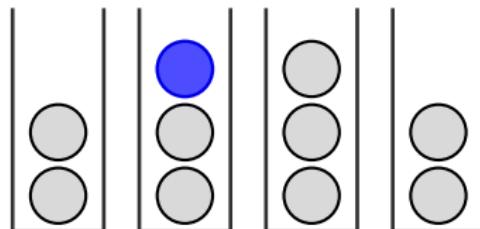
1. There's always one special bin with an extra ball

Recourse = 2

ANALYZING THE RECOURSE



Computing A_S



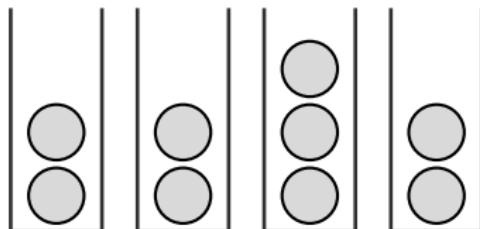
Computing $A_{S \cup \{x\}}$

Two key observations:

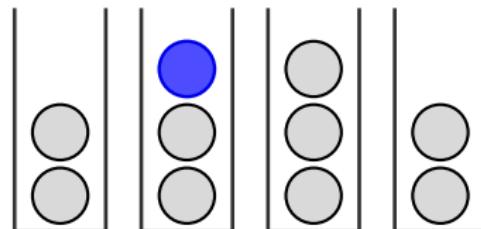
1. There's always one special bin with an extra ball
2. If a ball incurs recourse, one of its choices is the special bin

Recourse = 2

ANALYZING THE RECOURSE



Computing A_S

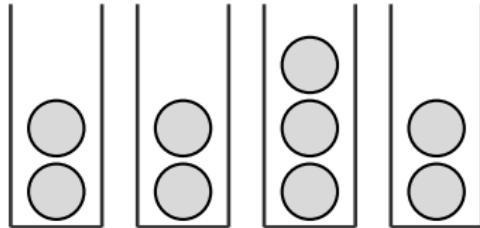


Computing $A_{S \cup \{x\}}$

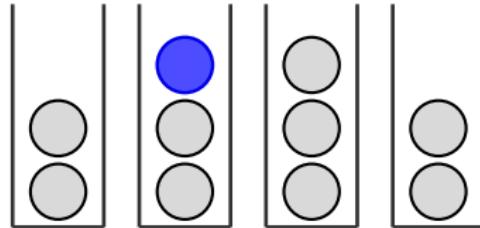
$$\Pr[\text{ball } x_i \text{ incurs recourse}] = O(1/n)$$

Recourse = 2

ANALYZING THE RECOURSE



Computing A_S



Computing $A_{S \cup \{x\}}$

$$\Pr[\text{ball } x_i \text{ incurs recourse}] = O(1/n)$$

$$\implies \mathbb{E}[\text{total recourse}] = \sum_i \Pr[\text{ball } x_i \text{ incurs recourse}] = O(m/n)$$

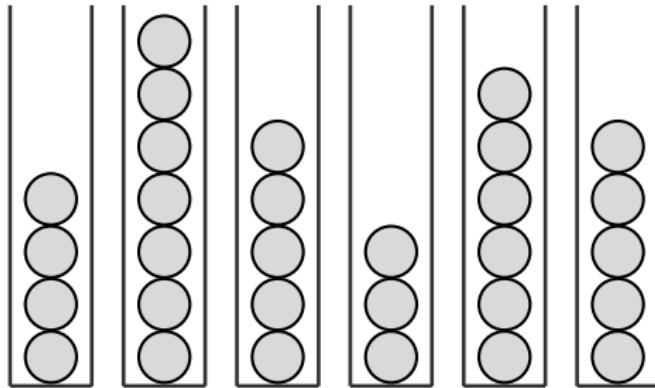
Recourse = 2

A SIMPLE WARMUP

Theorem: There exists a history-independent solution with:

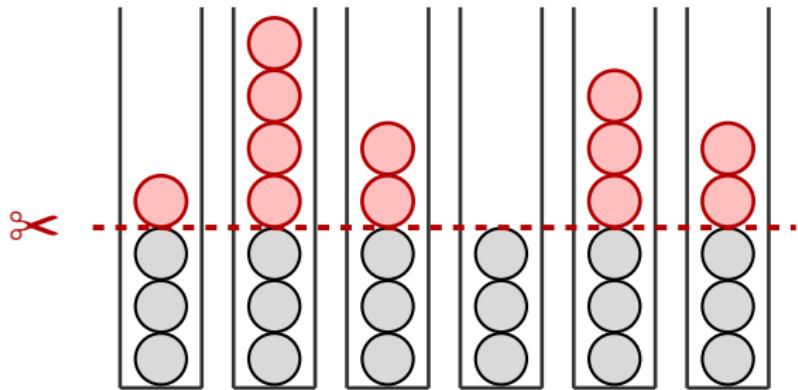
- ▶ High-probability overload $\Theta(1)$ $O(\log \log n)$.
- ▶ Expected recourse $\Theta(\log \log(m/n))$ $O(m/n)$.

SLICE AND SPREAD



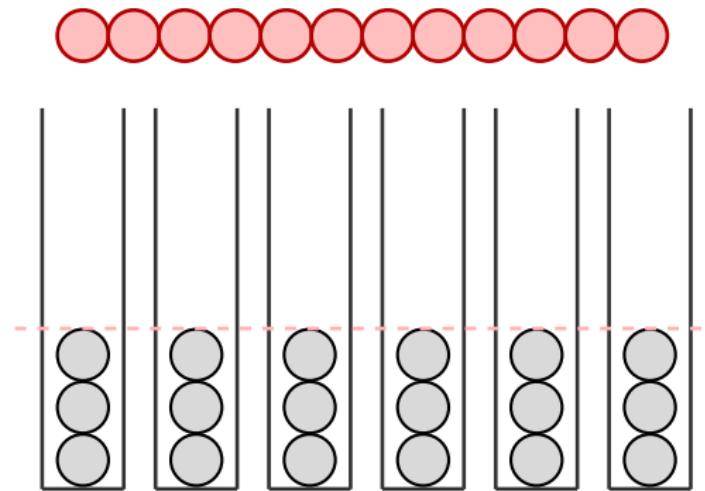
1. **Slice** off the jagged surface

SLICE AND SPREAD



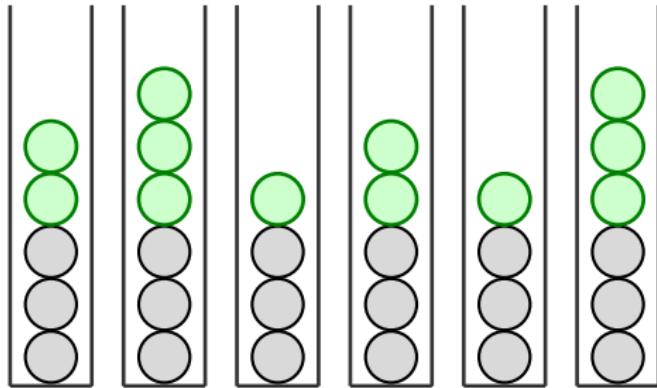
1. **Slice** off the jagged surface

SLICE AND SPREAD



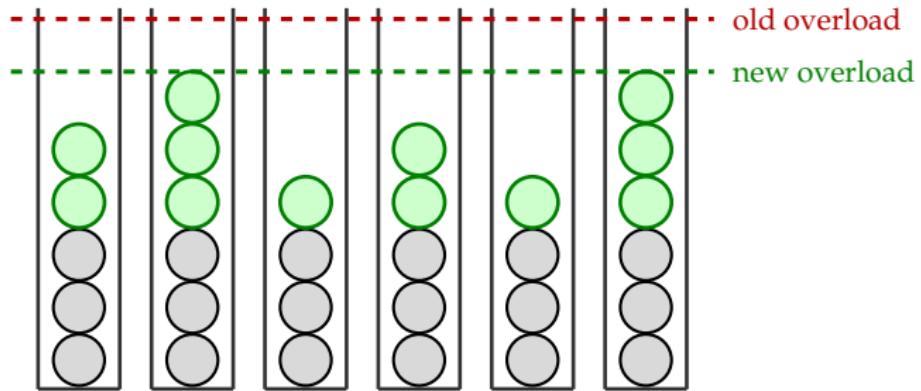
1. **Slice** off the jagged surface
2. **Spread** balls to their second-choice bins

SLICE AND SPREAD



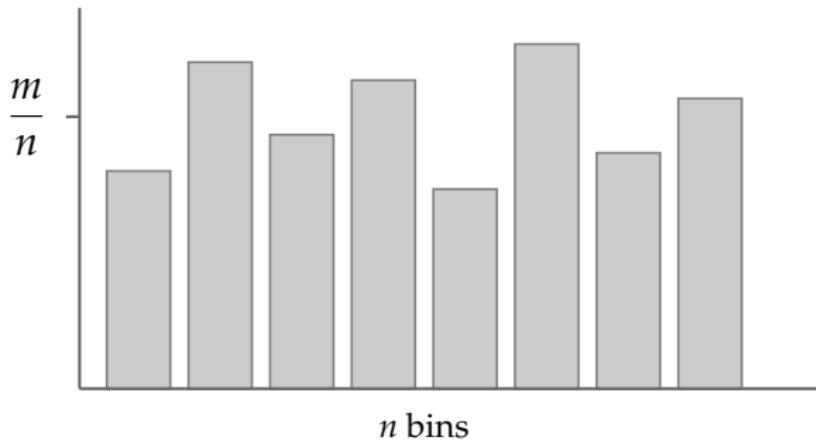
1. **Slice** off the jagged surface
2. **Spread** balls to their second-choice bins

SLICE AND SPREAD

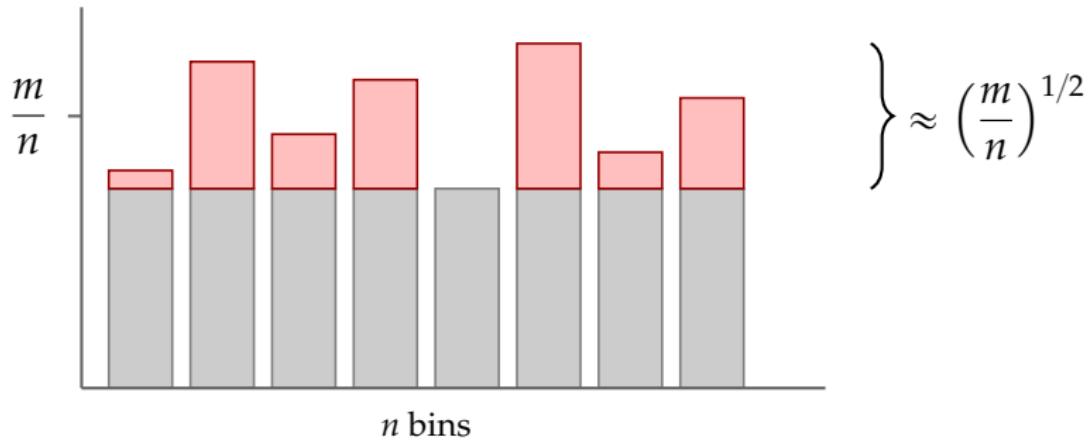


1. **Slice** off the jagged surface
2. **Spread** balls to their second-choice bins

SLICE AND SPREAD REDUCES OVERLOAD



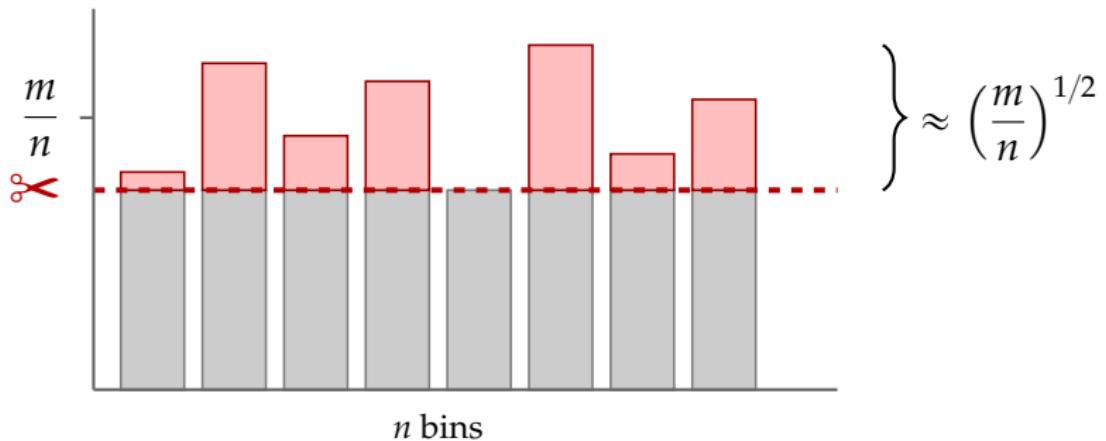
SLICE AND SPREAD REDUCES OVERLOAD



Key Fact

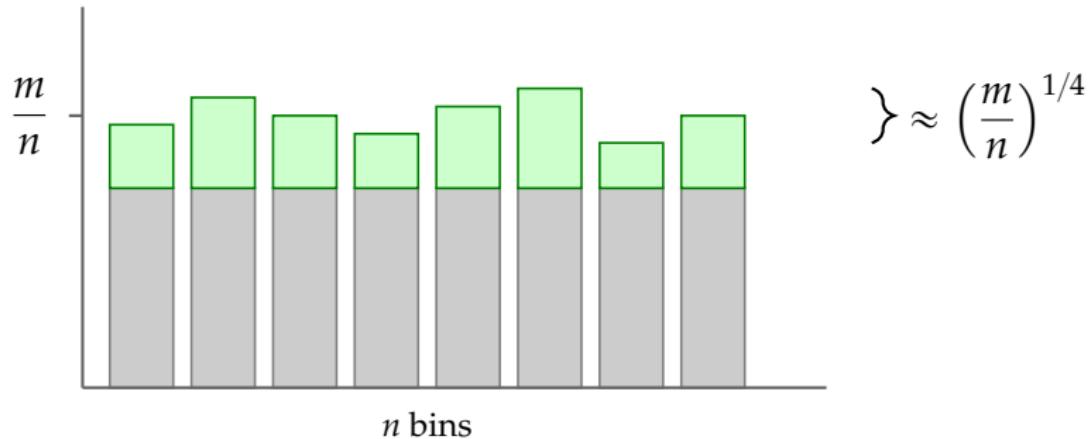
After throwing $m \gg n$ balls uniformly at random into n bins, the bin loads are within roughly $\approx \sqrt{m/n}$ with high probability in n .

SLICE AND SPREAD REDUCES OVERLOAD



- ▶ Balls above dotted line $\approx (mn)^{1/2}$

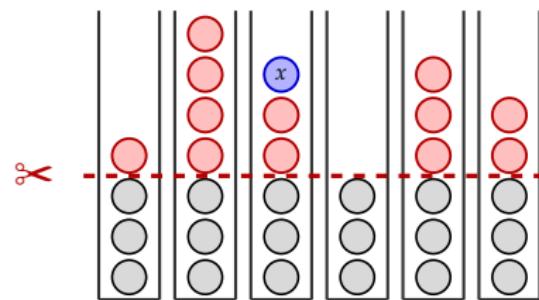
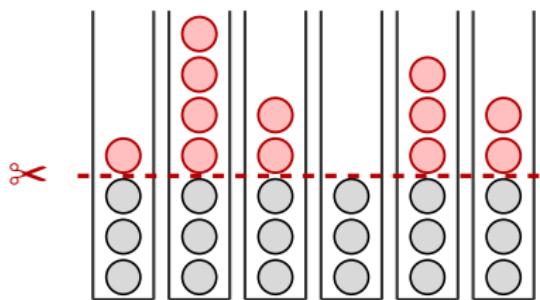
SLICE AND SPREAD REDUCES OVERLOAD



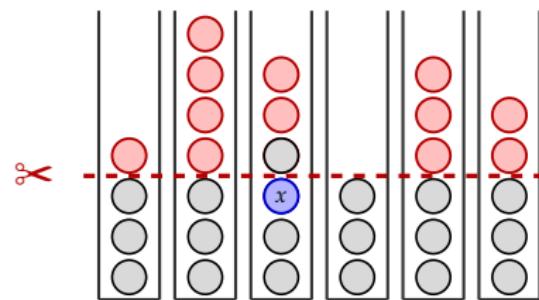
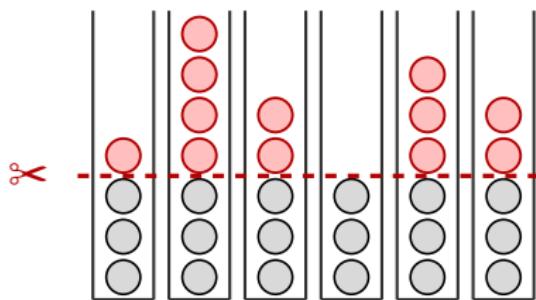
- ▶ Balls above dotted line $\approx (mn)^{1/2}$
- ▶ By Key Fact, bin loads within $\approx (m/n)^{1/4}$

WHAT'S THE RE COURSE?

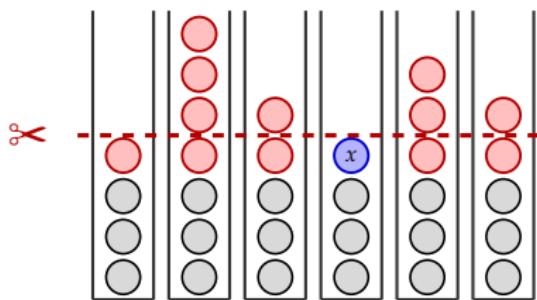
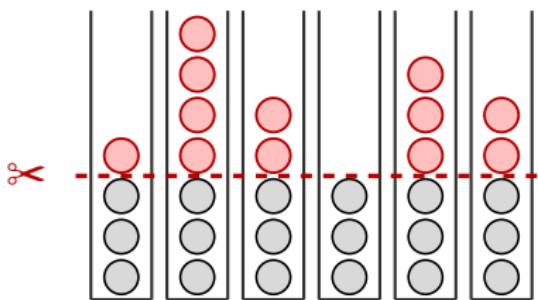
WHAT'S THE RECOURSE?



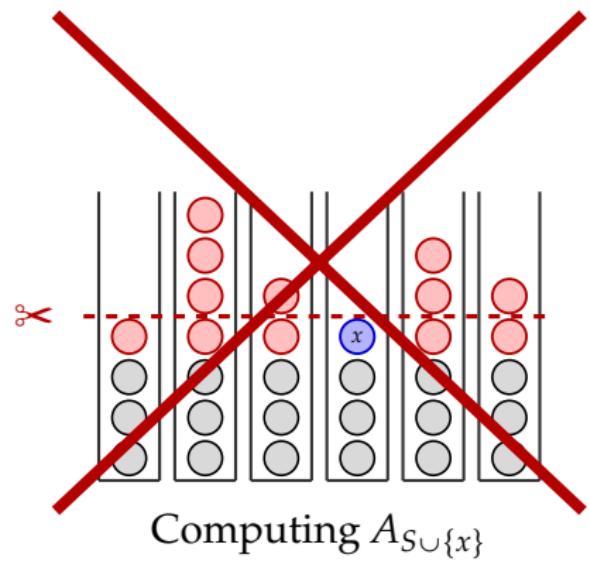
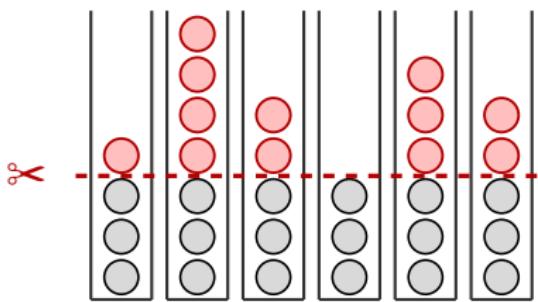
WHAT'S THE RECOURSE?



WHAT'S THE RECOURSE?

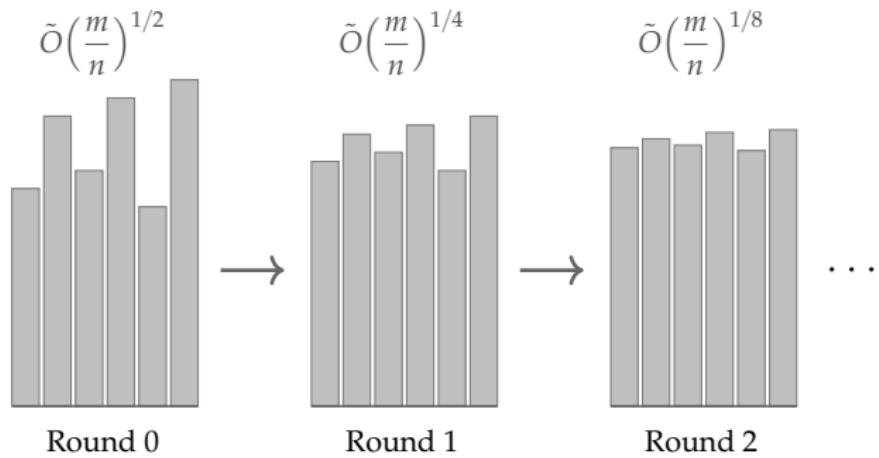


WHAT'S THE RECOURSE?

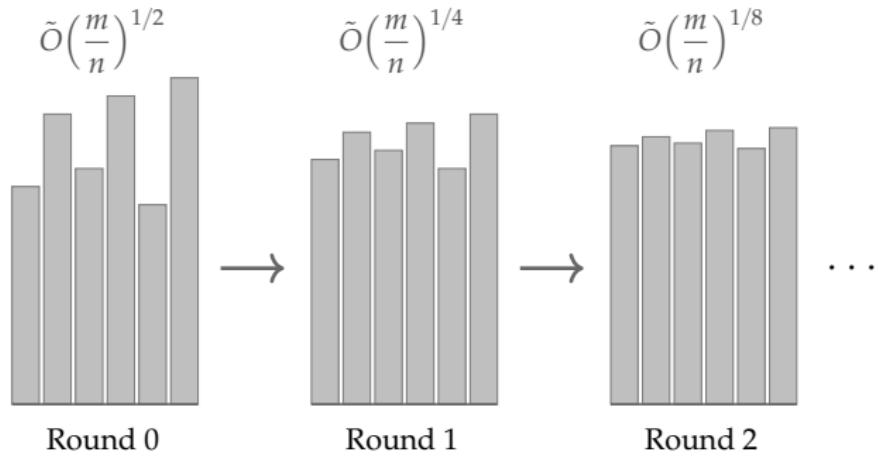


To keep recourse low, fix the slicing line (as a function of the maximum number of balls in the system).

REPEATEDLY SLICING AND SPREADING



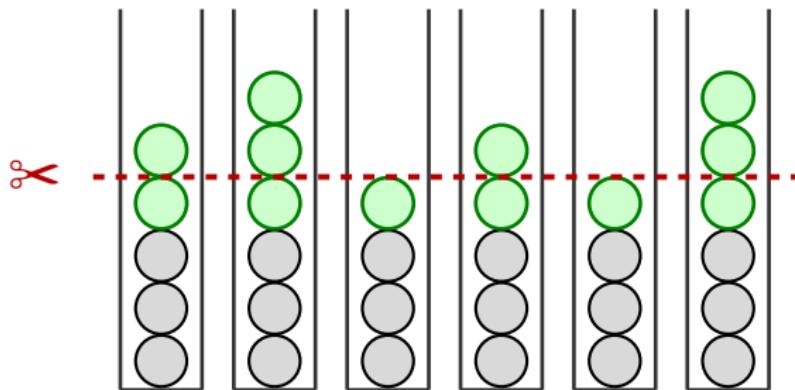
REPEATEDLY SLICING AND SPREADING



Proposition

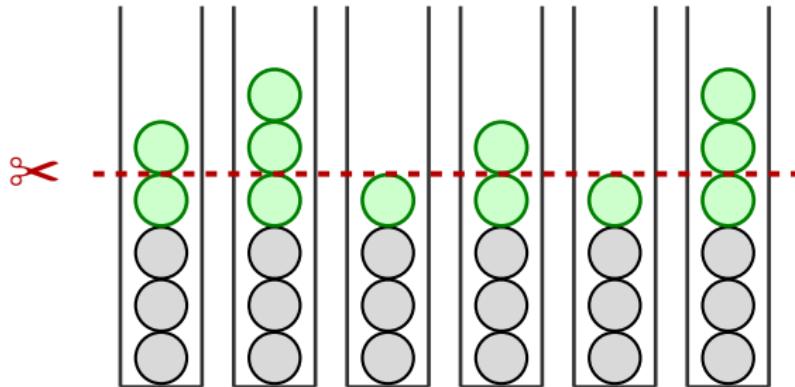
After $O(\log \log(m/n))$ rounds of slice-and-spread, the cumulative overload is $O(n)$ with high probability in n . The expected recourse is $O(\log \log(m/n))$.

ALGORITHMIC QUESTION



Question: Which balls do we slice in each round?

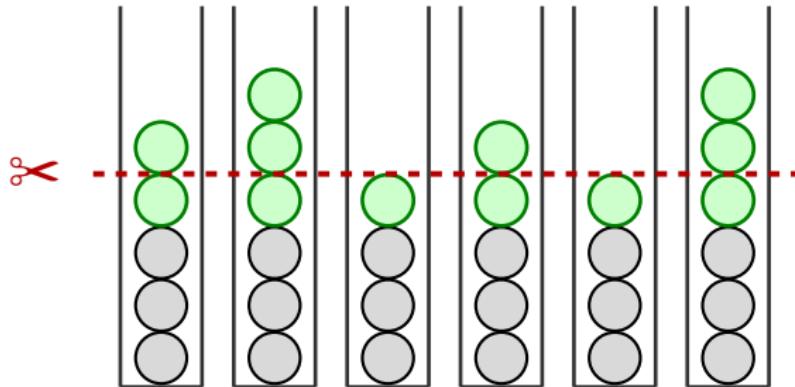
ALGORITHMIC QUESTION



Question: Which balls do we slice in each round?

- ▶ **Option 1:** Scrape off the top ✗ Reuses stale randomness

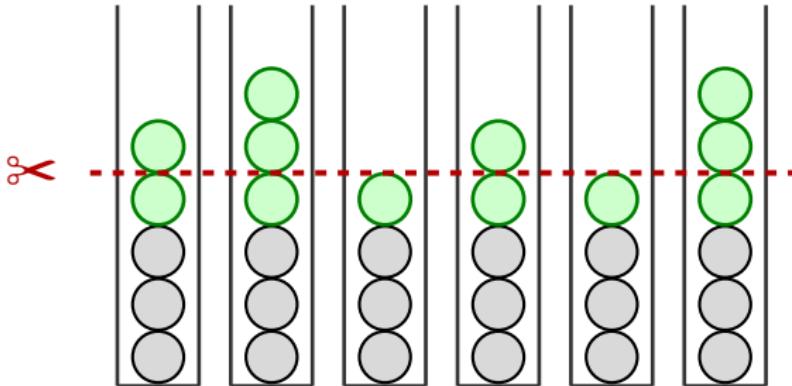
ALGORITHMIC QUESTION



Question: Which balls do we slice in each round?

- ▶ **Option 1:** Scrape off the top ✗ Reuses stale randomness
- ▶ **Option 2:** Priority queue per bin ✗ Exploding recourse

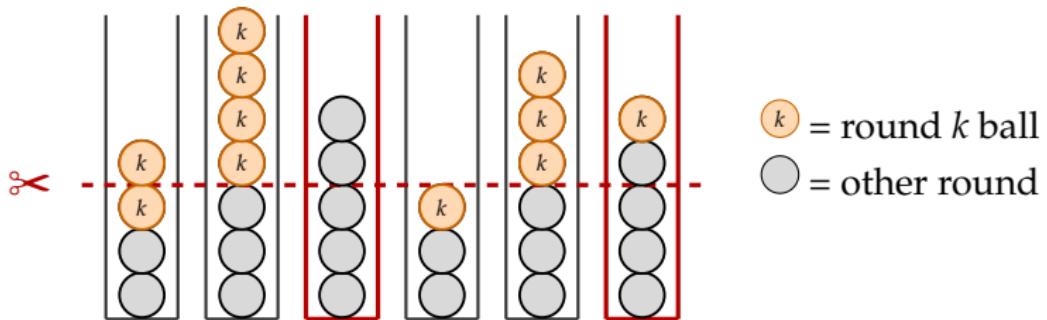
ALGORITHMIC QUESTION



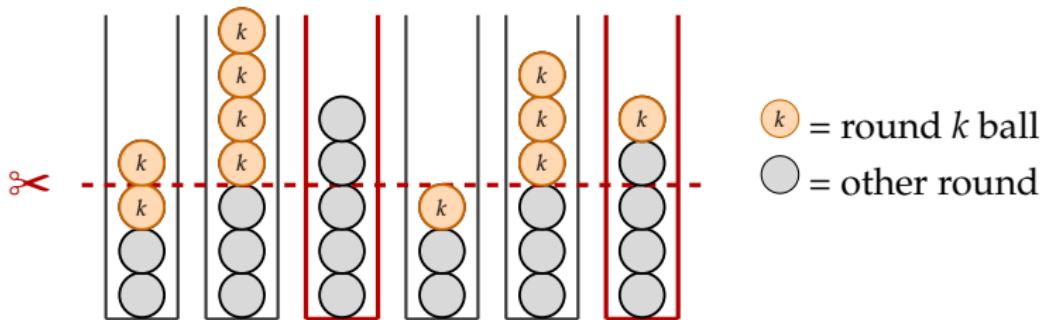
Question: Which balls do we slice in each round?

- ▶ **Option 1:** Scrape off the top ✗ Reuses stale randomness
- ▶ **Option 2:** Priority queue per bin ✗ Exploding recourse
- ▶ **Our approach:** Option 2 + Assign every ball a **round number**

CHALLENGE 1: SLICING FAILURES

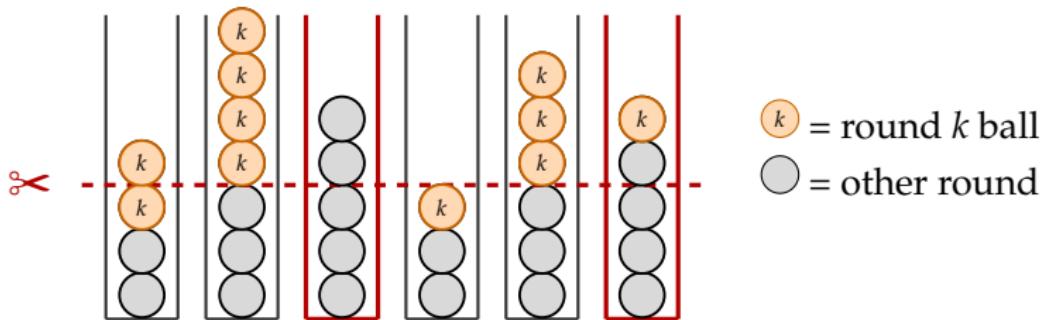


CHALLENGE 1: SLICING FAILURES



Challenge: Some bins may not have enough round- k balls above the line.

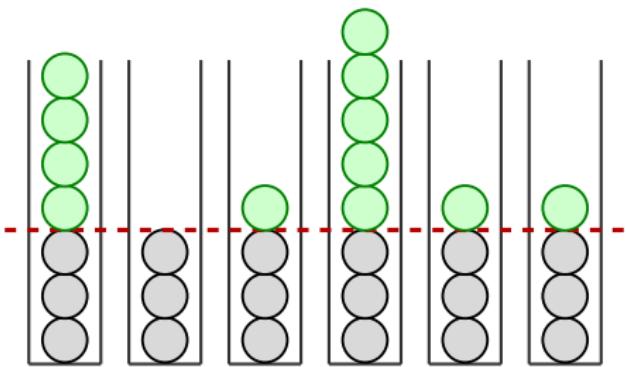
CHALLENGE 1: SLICING FAILURES



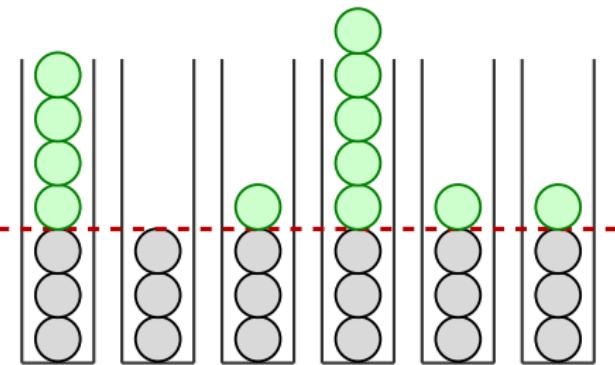
Challenge: Some bins may not have enough round- k balls above the line.

Result: We can't slice evenly — the jaggedness remains in those bins.

CHALLENGE 2: SPREADING FAILURES

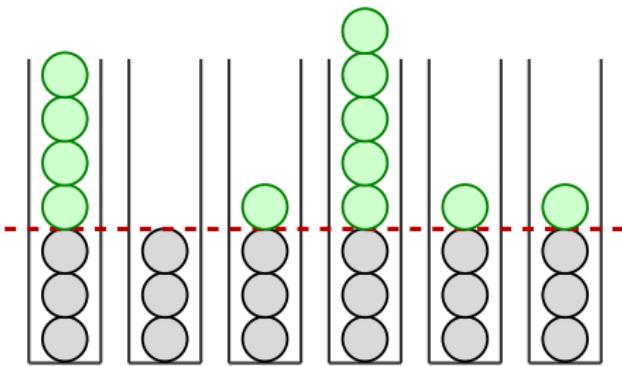


CHALLENGE 2: SPREADING FAILURES



Challenge: The spreading step may distribute balls unevenly — creating new jaggedness.

CHALLENGE 2: SPREADING FAILURES

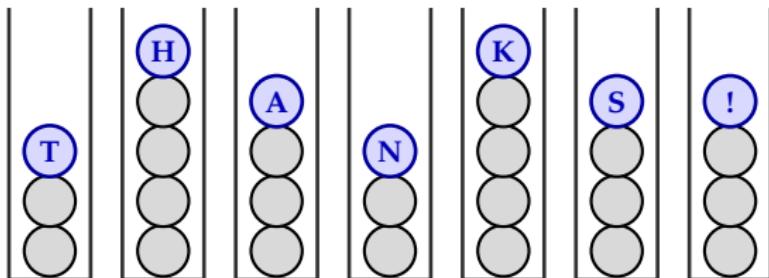


Challenge: The spreading step may distribute balls unevenly — creating new jaggedness.

Dilemma:

- ▶ Slice **more** balls \implies overload may not decrease
- ▶ Slice **fewer** balls \implies jaggedness may not decrease

History-Independent Load Balancing



Michael A. Bender
Stony Brook University

William Kuszmaul
CMU

Elaine Shi
CMU

Rose Silver
CMU