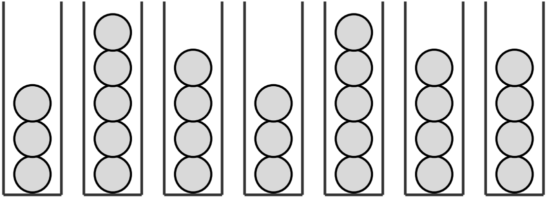
History-Independent Load Balancing



Michael A. Bender — Stony Brook University

William Kuszmaul — CMU

Elaine Shi — CMU

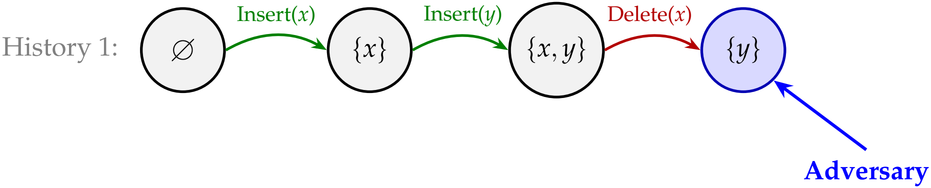
Rose Silver — CMU

History-Independent Data Structures

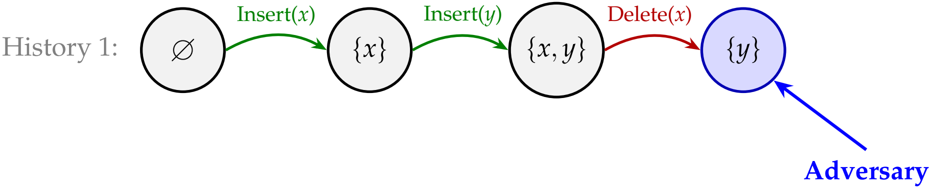
History-Independent Data Structures

History 1: $\varnothing \xrightarrow{\text{Insert}(x)} \{x\} \xrightarrow{\text{Insert}(y)} \{x, y\} \xrightarrow{\text{Delete}(x)} \{y\}$

History-Independent Data Structures



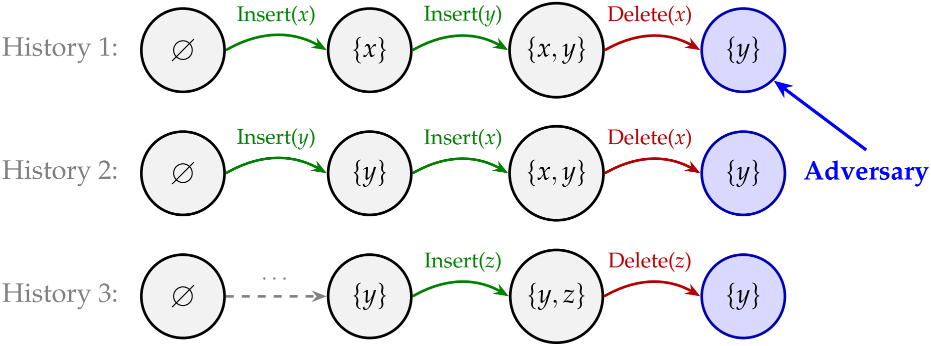
History-Independent Data Structures



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

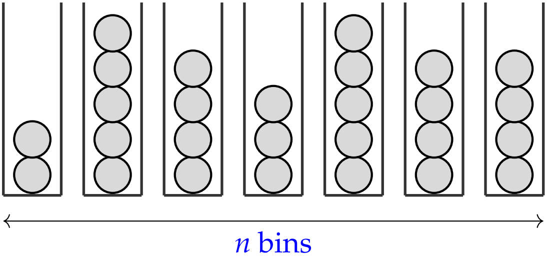
Hash tables, trees, memory allocation, PMAs, graph algorithms, cache-oblivious data structures, and more.

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

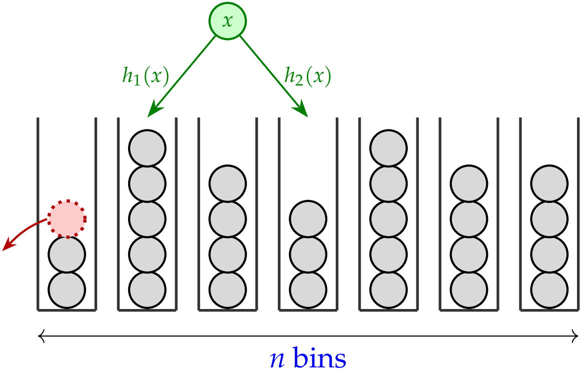
Yet some basic questions remain open.

This work: History-Independent Load Balancing

Two-Choice Load Balancing

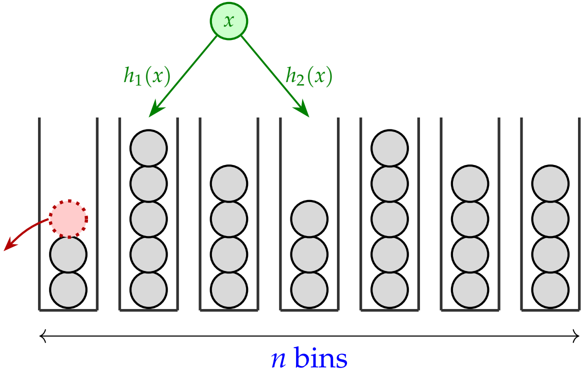


Two-Choice Load Balancing



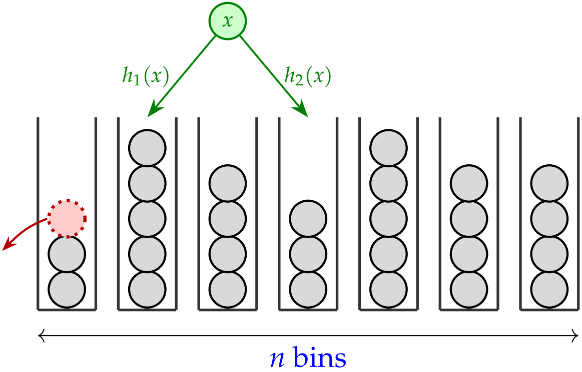
- Balls are inserted/deleted, with up to m present at a time.

Two-Choice Load Balancing



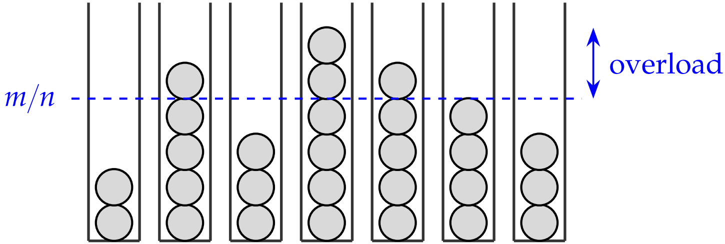
- Balls are inserted/deleted, with up to m present at a time.
- Each ball has two random bins where it can go.

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.

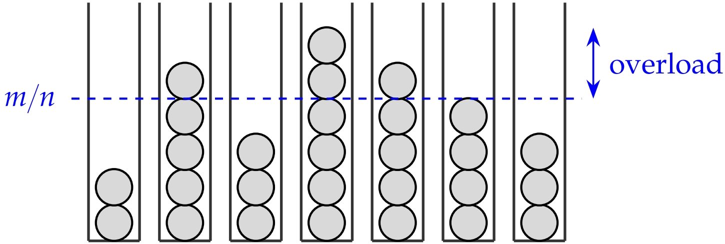
Two Goals



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!

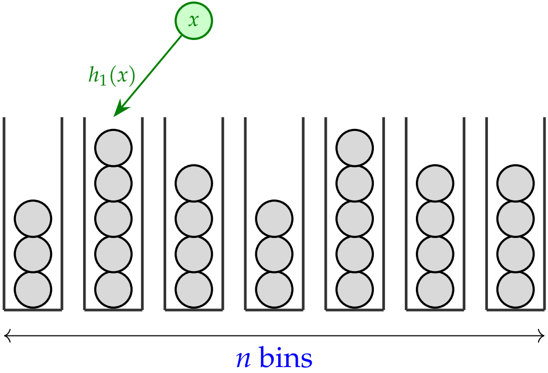
Rest of Talk: A Simple Warmup

Theorem: There exists a history-independent solution with:

- High-probability overload $\cancel{O(1)}$ $O(\log\log n)$.
- Expected recourse $\cancel{O(\log\log(m/n))}$ $O(m/n)$.

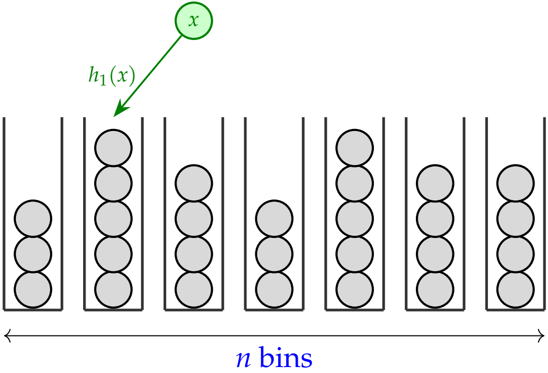
Baseline 1: The Single-Choice Strategy

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



Baseline 1: The Single-Choice Strategy

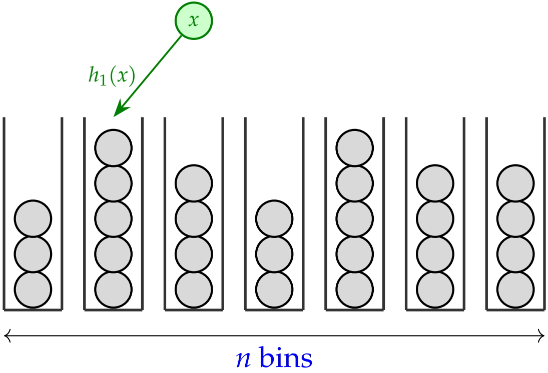
To insert a ball x, just put it in bin $h_1(x)$:



- This is history-independent ✓

Baseline 1: The Single-Choice Strategy

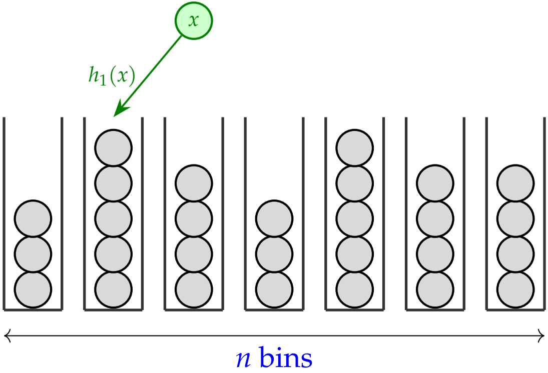
To insert a ball x, just put it in bin $h_1(x)$:



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

Baseline 1: The Single-Choice Strategy

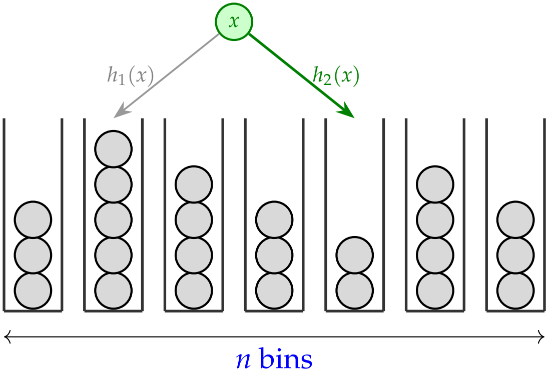
To insert a ball x, just put it in bin $h_1(x)$:



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

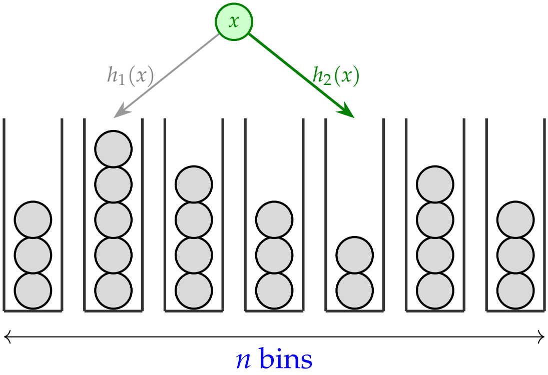
Baseline 2: Greedy Insertions

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



Baseline 2: Greedy Insertions

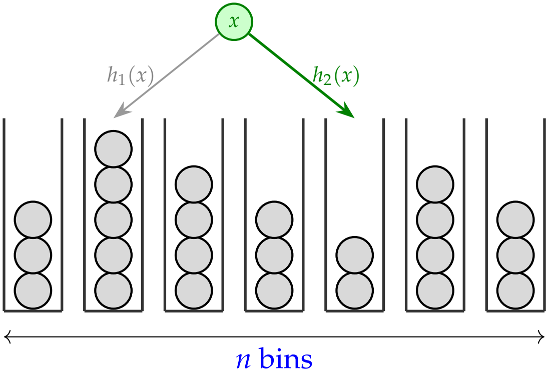
To insert a ball x, put it in the emptier of its choices:



- This is **not** history-independent ✗

Baseline 2: Greedy Insertions

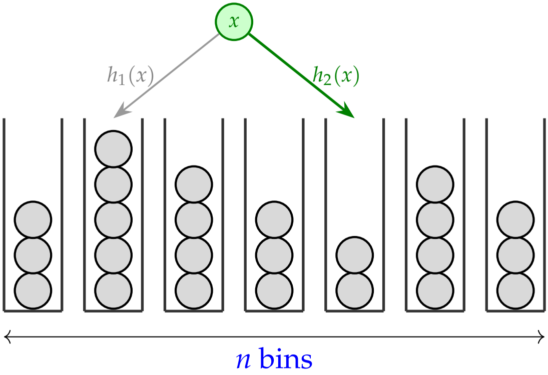
To insert a ball x, put it in the emptier of its choices:



- 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:

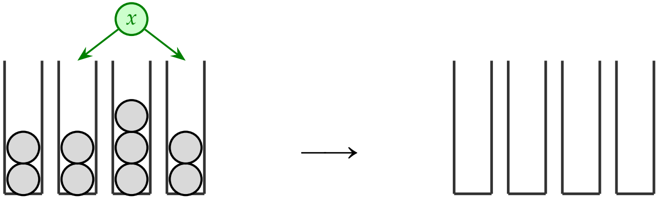


- This is **not** history-independent ✗
- The recourse is 0 ✓
- In the insertion-only case, the overload is $O(\log \log n)$ ✓

[Berenbrink, Czumaj, Steger, and Vöcking '00]

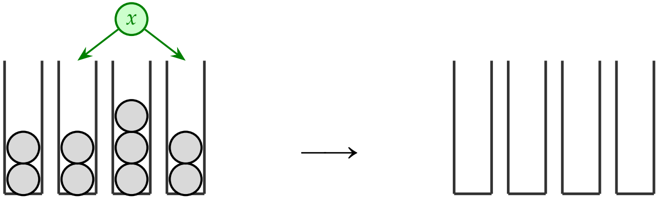
Warmup: History-Independent Greedy

Warmup: History-Independent Greedy



Current state A_S $\longrightarrow$ Computing $A_{S \cup \{x\}}$

Warmup: History-Independent Greedy

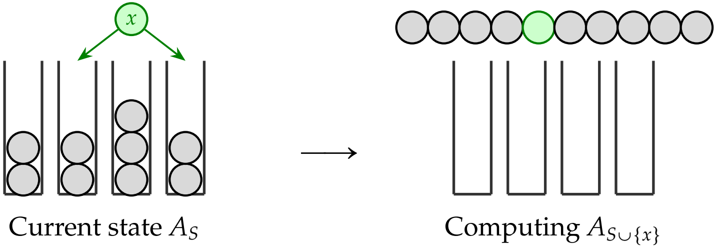


Current state A_S $\longrightarrow$ Computing $A_{S \cup \{x\}}$

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

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

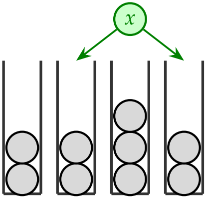
Warmup: History-Independent Greedy



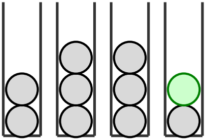
To compute $A_{S \cup \{x\}}$:

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

WARMUP: HISTORY-INDEPENDENT GREEDY



Current state A_S

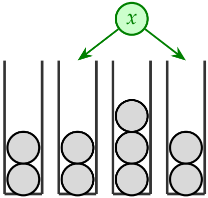


Computing $A_{S \cup \{x\}}$

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

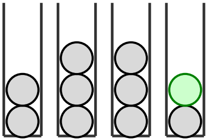
1. Empty out the bins.
2. Sort the balls in $S \cup \{x\}$ to get $x_1, x_2, \ldots$
3. Insert the balls in sorted order using greedy.

Analyzing History-Independent Greedy



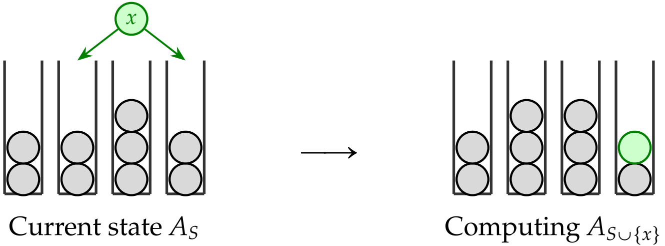
Current state A_S

$\longrightarrow$



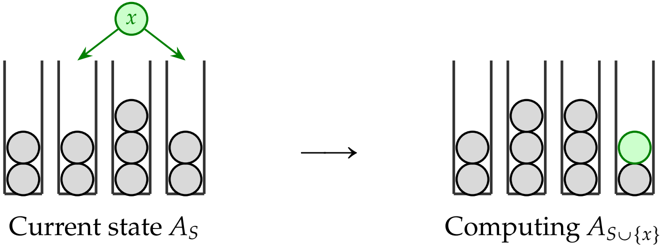
Computing $A_{S \cup \{x\}}$

ANALYZING HISTORY-INDEPENDENT GREEDY



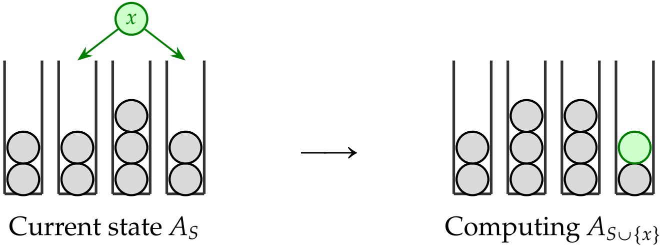
- The algorithm is history independent ✓

Analyzing History-Independent Greedy



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

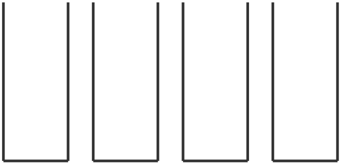
ANALYZING HISTORY-INDEPENDENT GREEDY



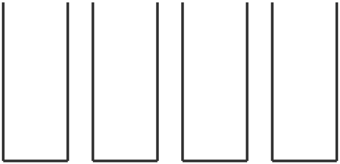
- The algorithm is history independent ✓
- The overload is $O(\log \log n)$ ✓
- What is the recourse?

Analyzing the Recourse

Recourse $= 0$



Computing A_S

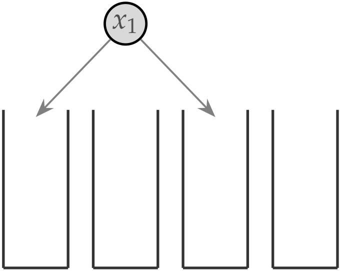


Computing $A_{S \cup \{x\}}$

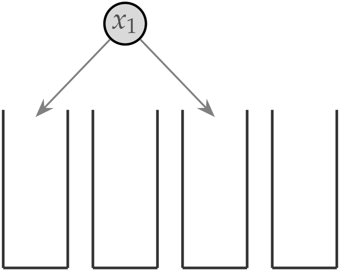
How many balls change assignments between A_S and $A_{S \cup \{x\}}$?

Analyzing the Recourse

Recourse = 0



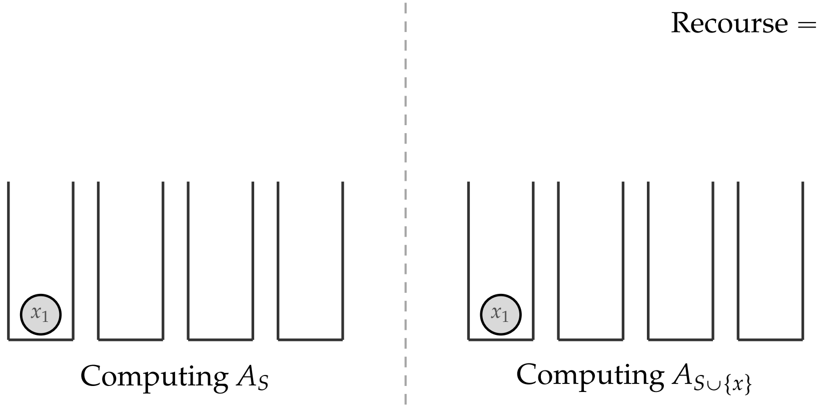
Computing A_S



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

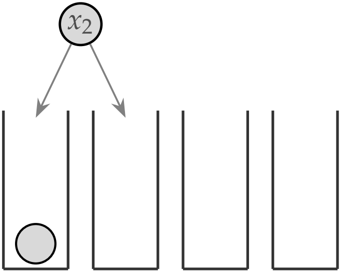
Analyzing the Recourse

Recourse = 0

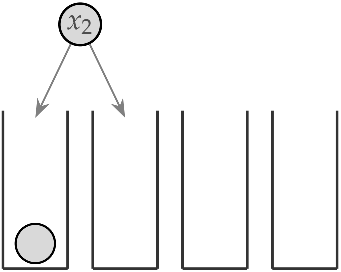


Analyzing the Recourse

Recourse $= 0$



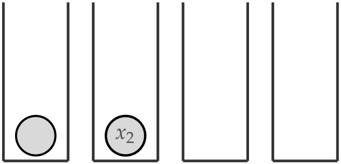
Computing A_S



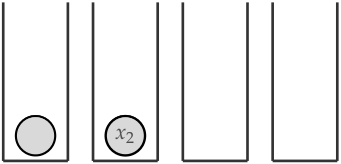
Computing $A_{S \cup \{x\}}$

Analyzing the Recourse

Recourse = 0



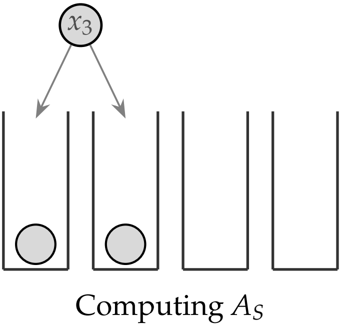
Computing A_S

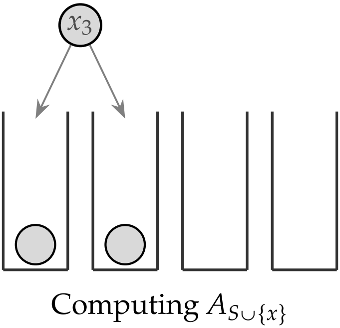


Computing $A_{S \cup \{x\}}$

Analyzing the Recourse

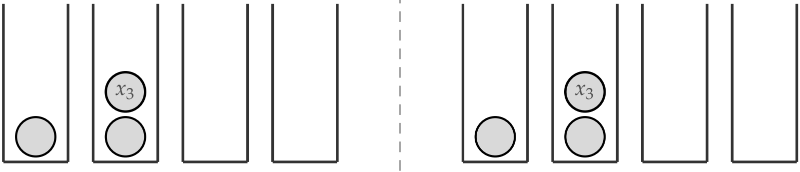
Recourse = 0





Analyzing the Recourse

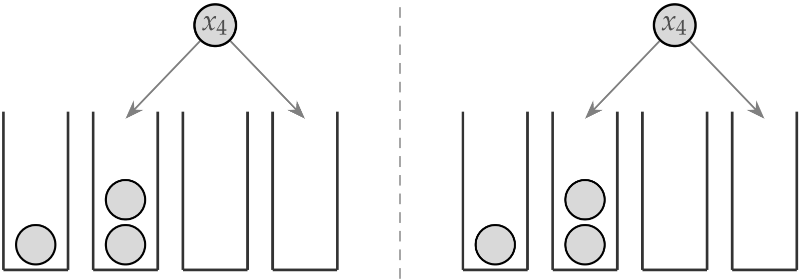
Recourse $= 0$



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

Analyzing the Recourse

Recourse $= 0$

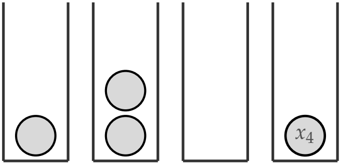


Computing A_S

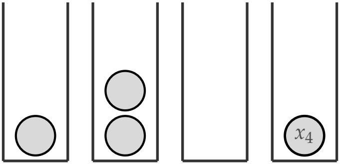
Computing $A_{S \cup \{x\}}$

Analyzing the Recourse

Recourse = 0

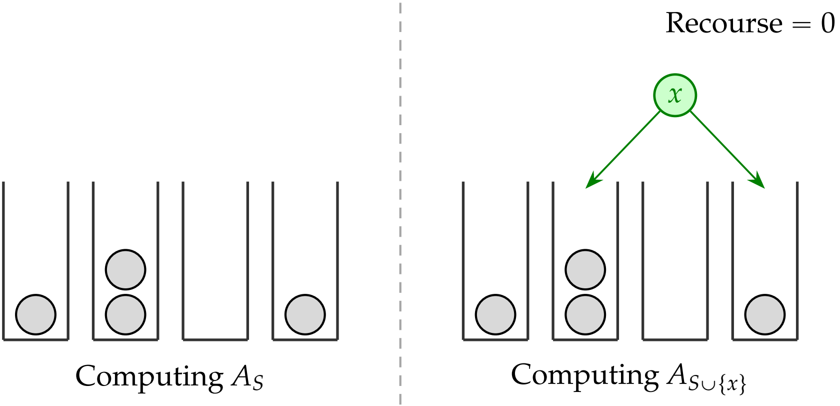


Computing A_S



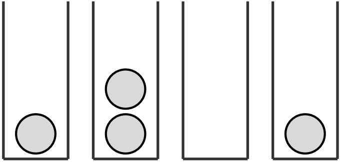
Computing $A_{S \cup \{x\}}$

Analyzing the Recourse

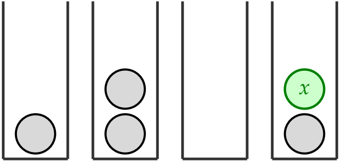


Analyzing the Recourse

Recourse = 0



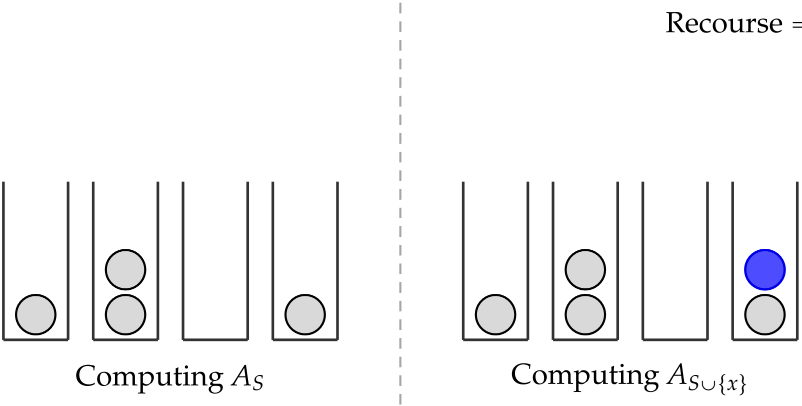
Computing A_S



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

Analyzing the Recourse

Recourse = 0

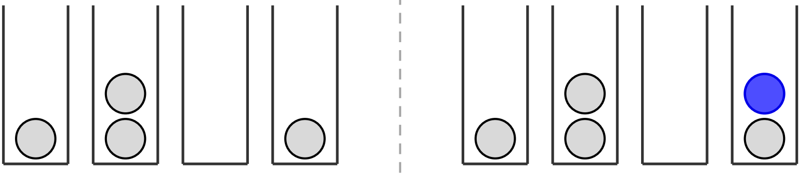


Computing A_S | Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

Analyzing the Recourse

Recourse $= 0$

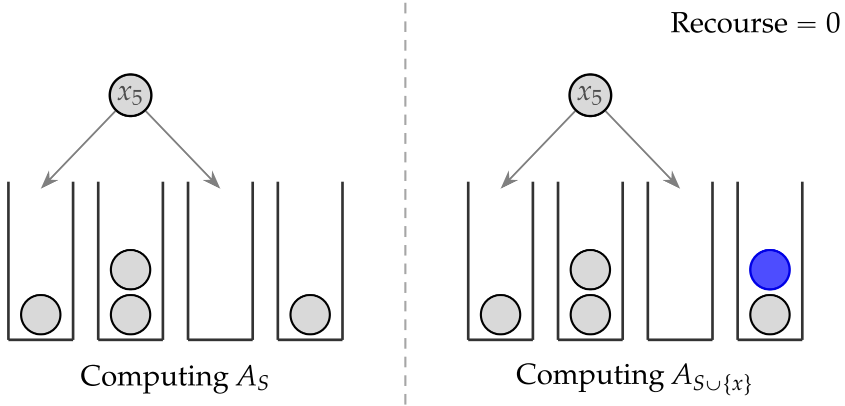


Computing A_S | Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse

Analyzing the Recourse

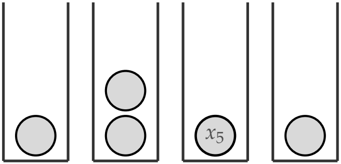


Future insertions will experience either:

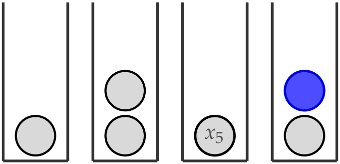
1. No recourse

Analyzing the Recourse

Recourse $= 0$



Computing A_S



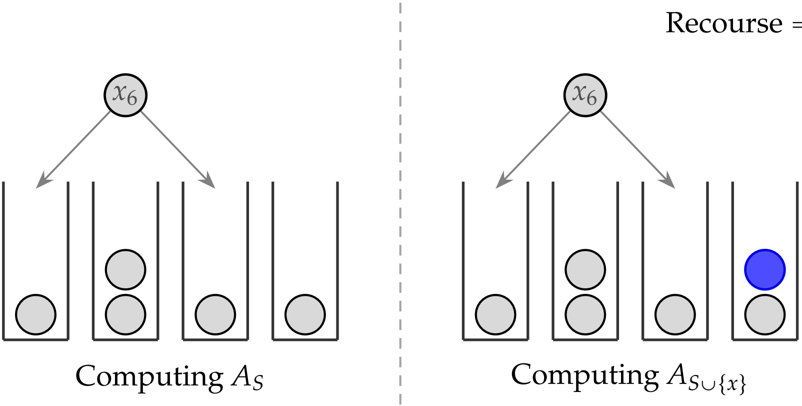
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse

Analyzing the Recourse

Recourse = 0

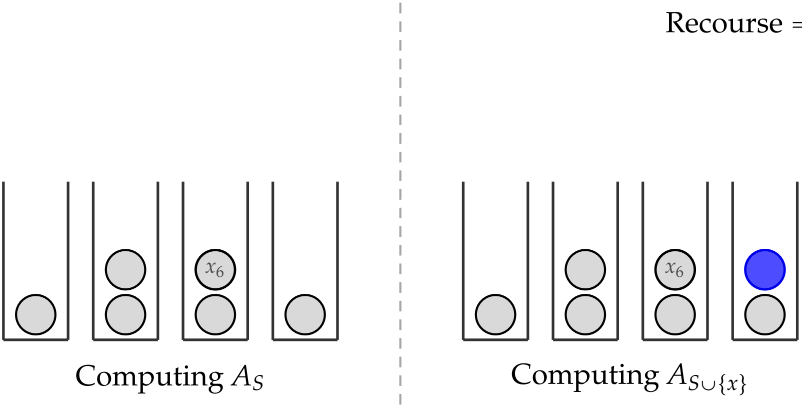


Subsequent balls will experience either:

1. No recourse

Analyzing the Recourse

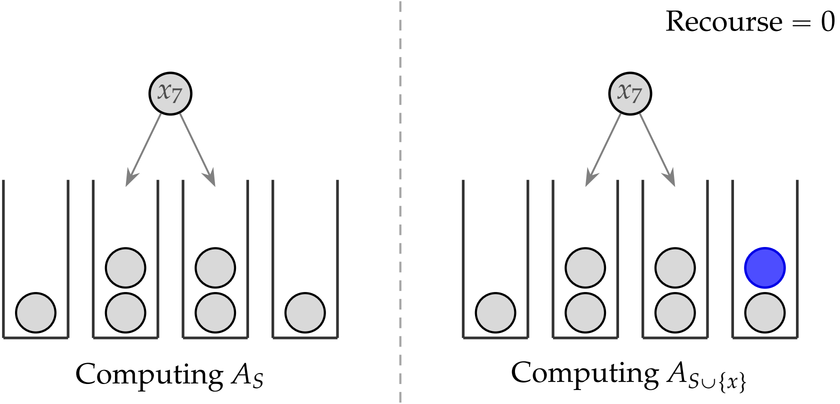
Recourse = 0



Subsequent balls will experience either:

1. No recourse

Analyzing the Recourse

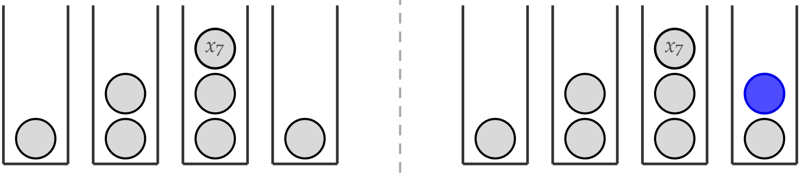


Subsequent balls will experience either:

1. No recourse

Analyzing the Recourse

Recourse = 0

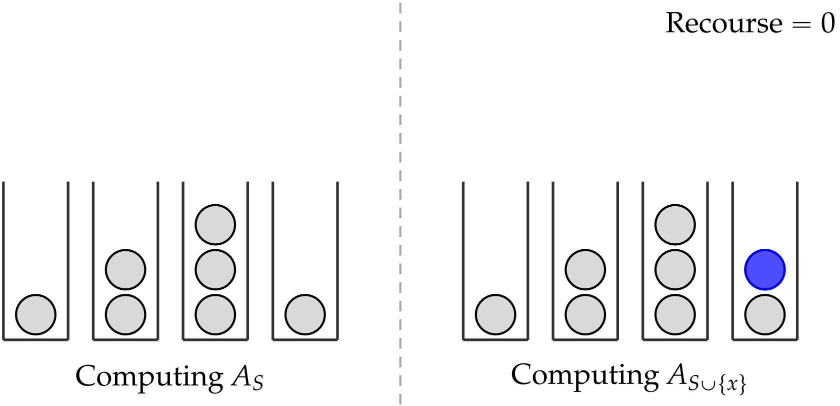


Computing A_S Computing $A_{S\cup\{x\}}$

Subsequent balls will experience either:

1. No recourse

Analyzing the Recourse

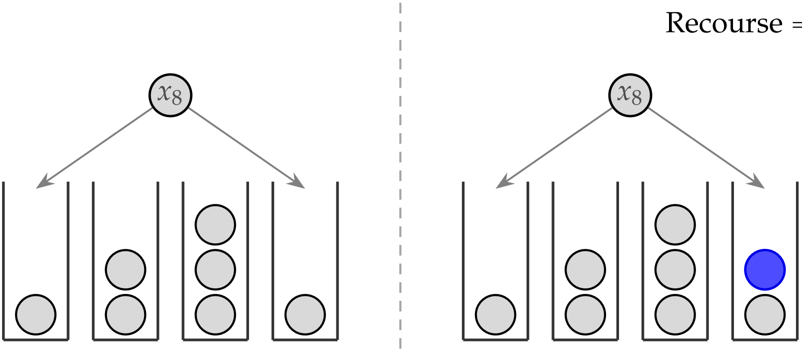


Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse

Recourse = 0



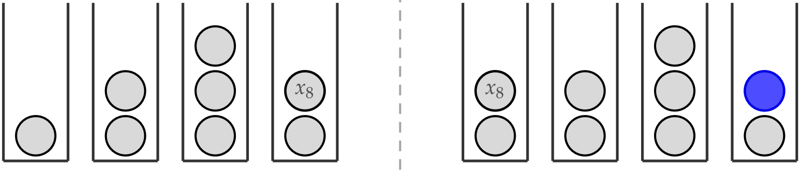
Computing A_S — Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse

Recourse = 1



Computing A_S

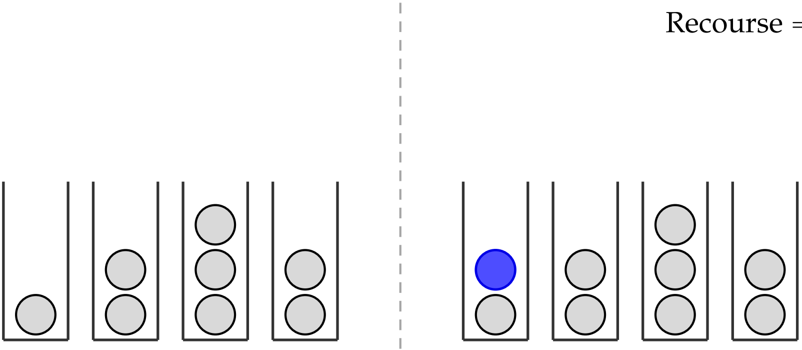
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse

Recourse = 1



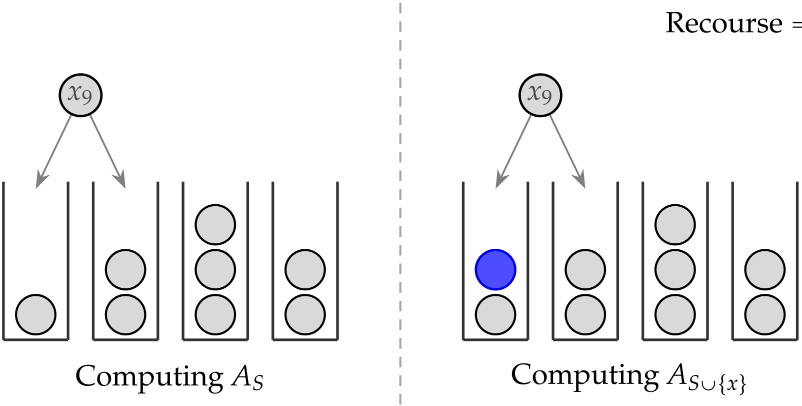
Computing A_S | Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse

Recourse = 1

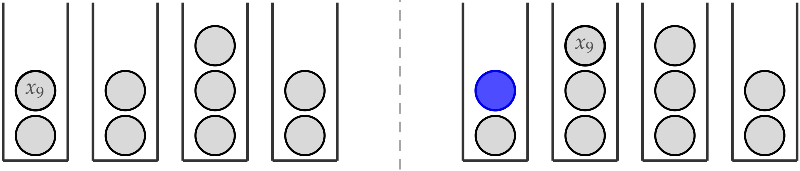


Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse

Recourse = 2



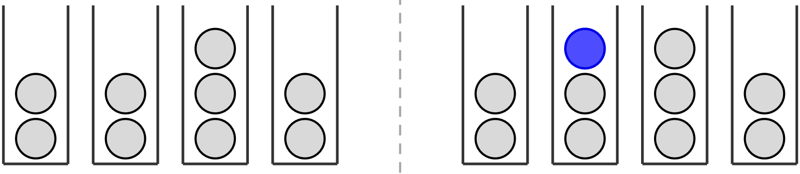
Computing A_S | Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse

Recourse = 2



Computing A_S

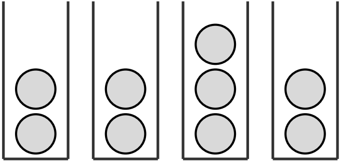
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

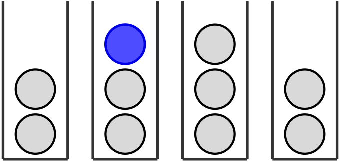
1. No recourse
2. Recourse

Analyzing the Recourse

Recourse $= 2$



Computing A_S

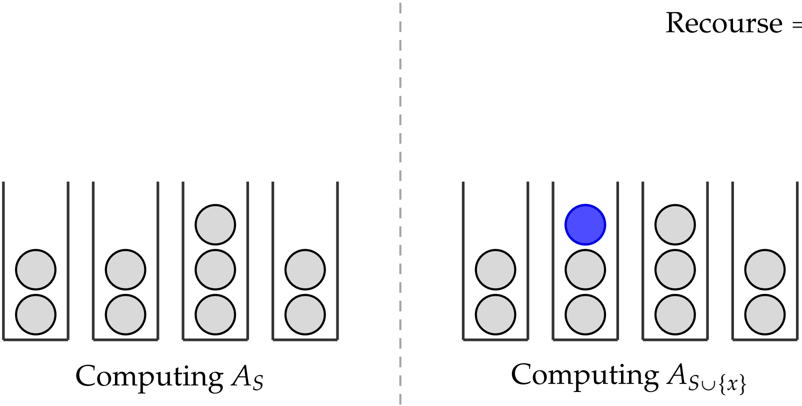


Computing $A_{S \cup \{x\}}$

Two key observations:

Analyzing the Recourse

Recourse = 2



Computing A_S

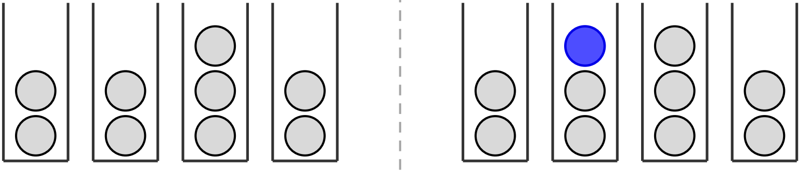
Computing $A_{S \cup \{x\}}$

Two key observations:

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

Analyzing the Recourse

Recourse $= 2$



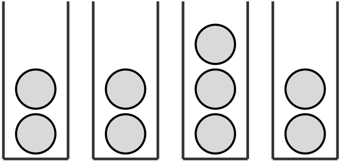
Computing A_S — Computing $A_{S\cup\{x\}}$

Two key observations:

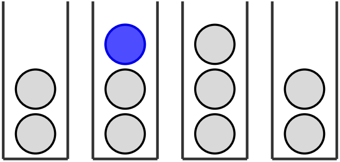
1. There's always one special bin with an extra ball
2. If a ball incurs recourse, one of its choices is the special bin

Analyzing the Recourse

Recourse = 2



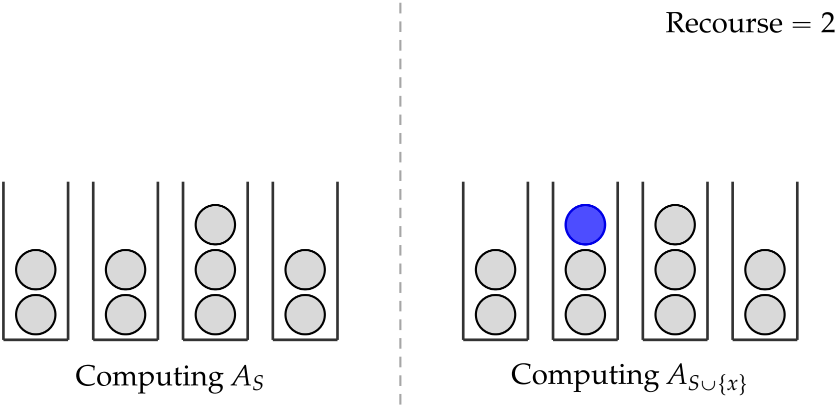
Computing A_S



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

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

Analyzing the Recourse



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

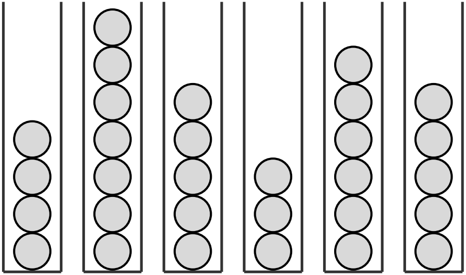
$$\implies \mathbb{E}[\text{total recourse}] = \sum_i \Pr[\text{ball } x_i \text{ incurs recourse}] = O(m/n)$$

A Simple Warmup

Theorem: There exists a history-independent solution with:

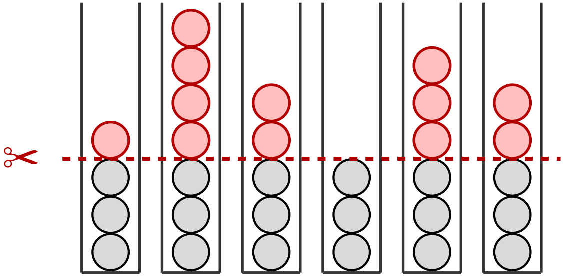
- High-probability overload $\cancel{O(1)}$ $O(\log \log n)$.
- Expected recourse $\cancel{O(\log \log (m/n))}$ $O(m/n)$.

Slice and Spread



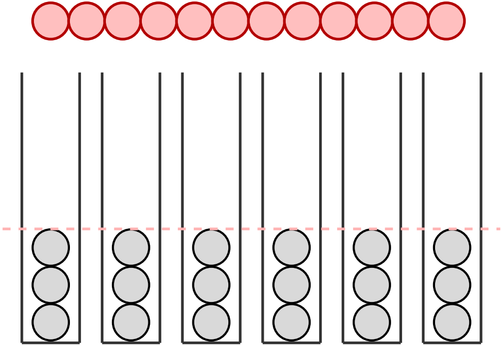
1. **Slice** off the jagged surface

Slice and Spread



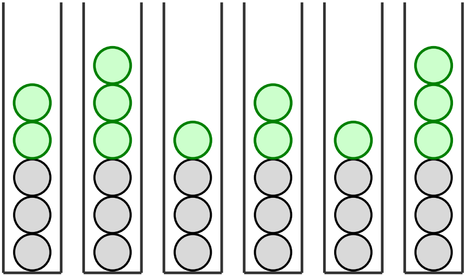
1. **Slice** off the jagged surface

Slice and Spread



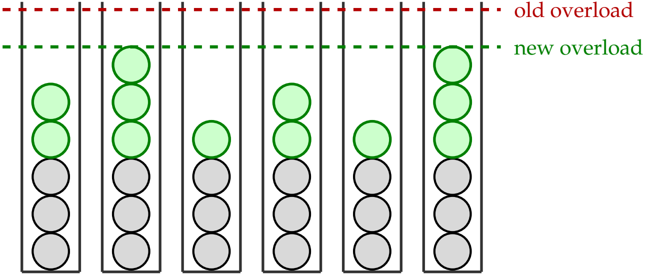
1. **Slice** off the jagged surface
2. **Spread** balls to their second-choice bins

Slice and Spread



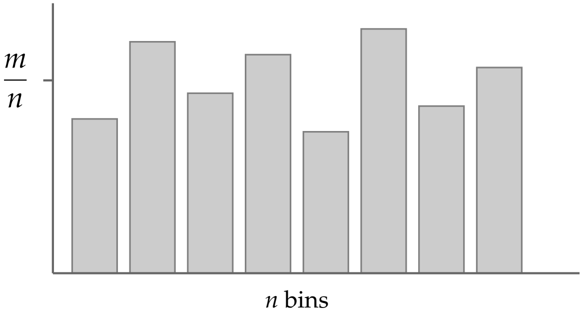
1. **Slice** off the jagged surface
2. **Spread** balls to their second-choice bins

Slice and Spread

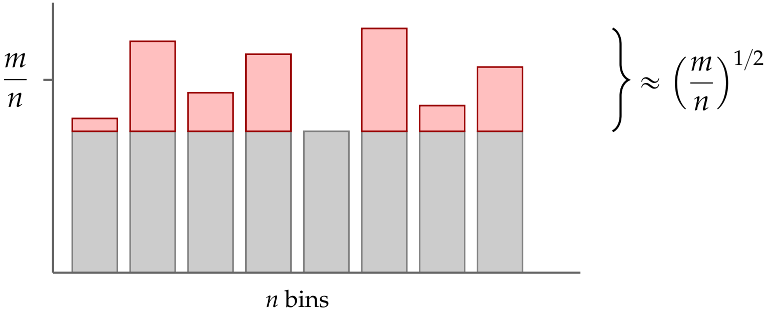


1. **Slice** off the jagged surface
2. **Spread** balls to their second-choice bins

SLICE AND SPREAD REDUCES OVERLOAD



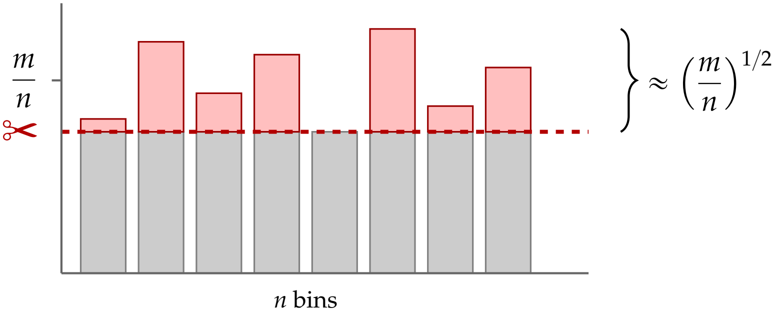
Slice and Spread Reduces Overload



Key Fact

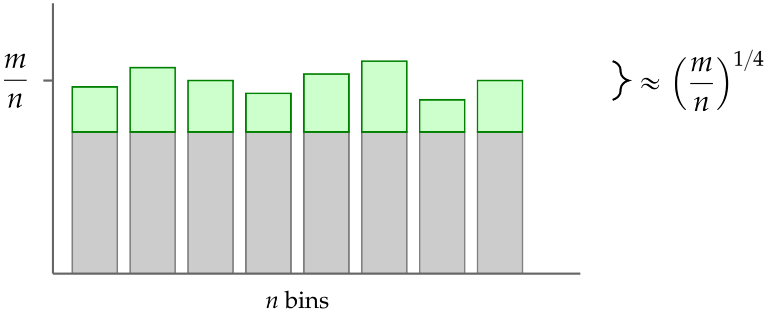
After throwing $m \gg n$ balls uniformly at random into n bins, the bin loads are within roughly $\approx \sqrt{m/n}$ with high probability in n.

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 RECOURSE?

Options:

(A) 0

(B) 1

(C) # balls being sliced off

(D) other?

Repeating Slice and Spread

Good overload, good recourse...Could we keep slicing and spreading to get better and better overload?
Proposition: Test

Algorithmic Question

Question: Which balls will we slice in each round?
Option 1: Scrape off the top **Option 2:** Maintain a priority queue in each bin **What we do:** Assign every ball a round...

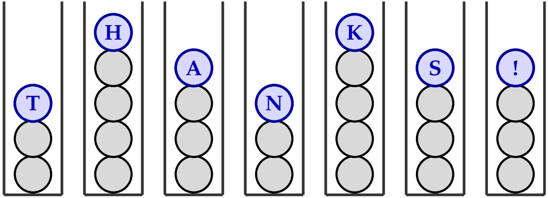
Challenge 1: Slicing Failures

Challenge: There may not be enough fresh randomness in each bin. Not enough balls assigned to that round in that bin.
Result: We won't be able to slice off enough balls, and so the jaggedness will remain.

Challenge 2: Spreading Failures

Challenge: Spreading step got unlucky and didn't spread very evenly. There's more jaggedness than we want
Result: Either we have to slice more balls than we want and may not get the overload down, or we have to slice less balls than we want and may not get the jaggedness down.

History-Independent Load Balancing



Michael A. Bender
Stony Brook University

William Kuszmaul
CMU

Elaine Shi
CMU

Rose Silver
CMU