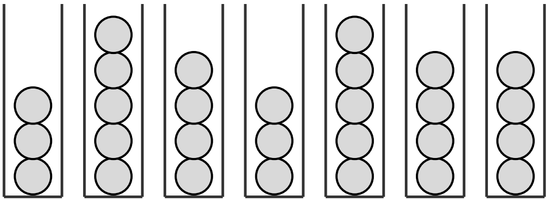
# History-Independent Load Balancing




Michael A. Bender — Stony Brook University

William Kuszmaul — CMU

Elaine Shi — CMU

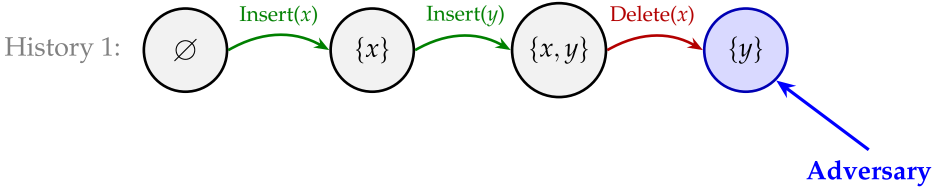
**Rose Silver** — CMU

# History-Independent Data Structures

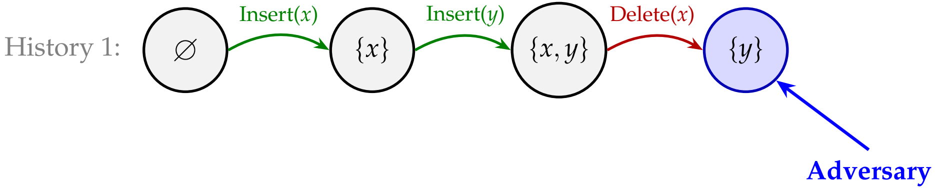
# History-Independent Data Structures

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

# History-Independent Data Structures



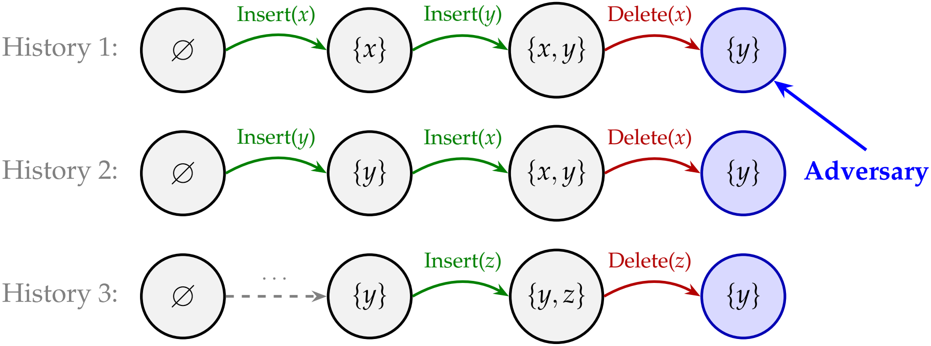
# History-Independent Data Structures




**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

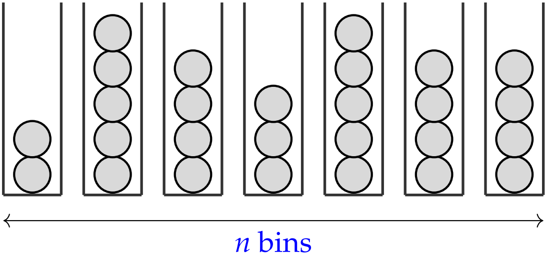
Hash tables, trees, memory allocation, PMAs, graph algorithms, cache-oblivious data structures, and more.

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

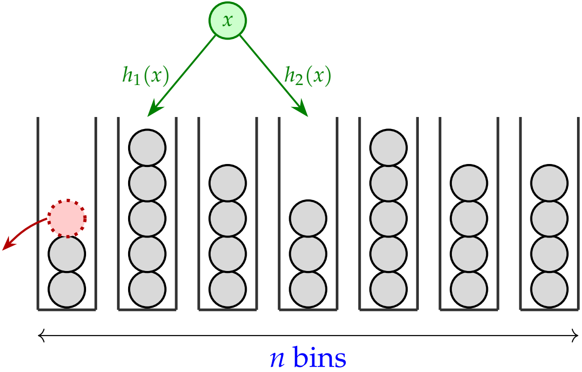
**Yet some basic questions remain open.**

**This work:** History-Independent Load Balancing

# Two-Choice Load Balancing

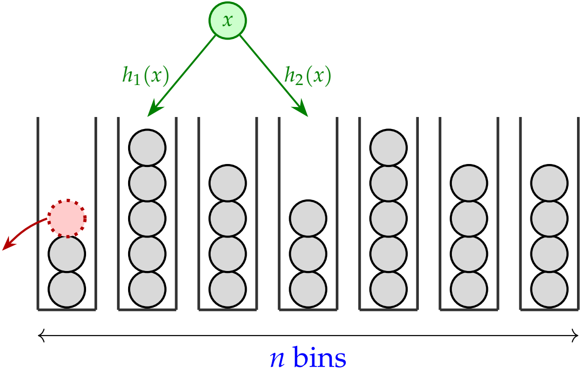


# Two-Choice Load Balancing



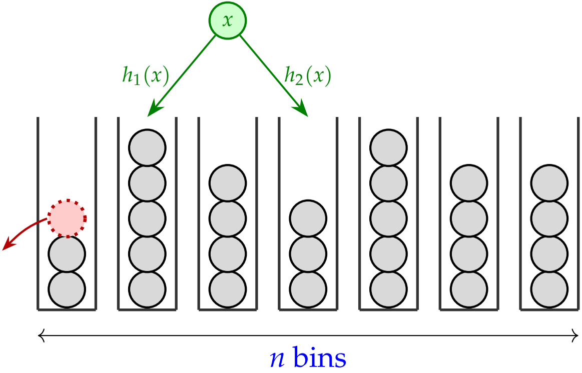

- Balls are inserted/deleted, with up to $m$ present at a time.

# Two-Choice Load Balancing



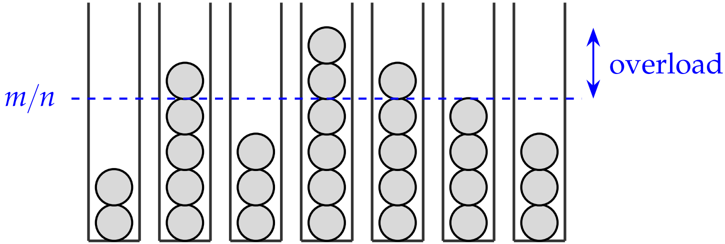

- Balls are inserted/deleted, with up to $m$ present at a time.
- Each ball has two random bins where it can go.

# 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.

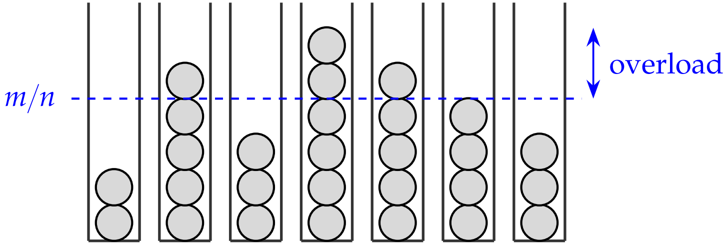
# Two Goals




**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

1. A Simple Warmup
2. The Full Algorithm

# Part 1: A Simple Warmup

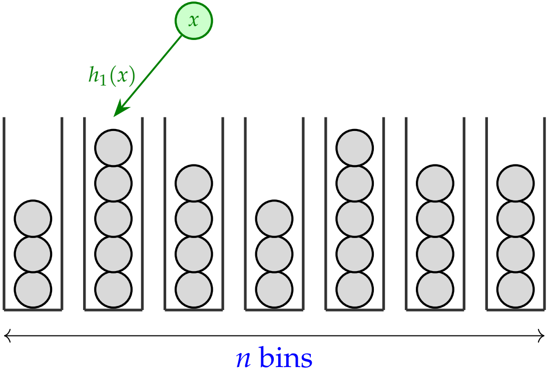
# A Simple Warmup

**Theorem:** There exists a history-independent solution with:

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

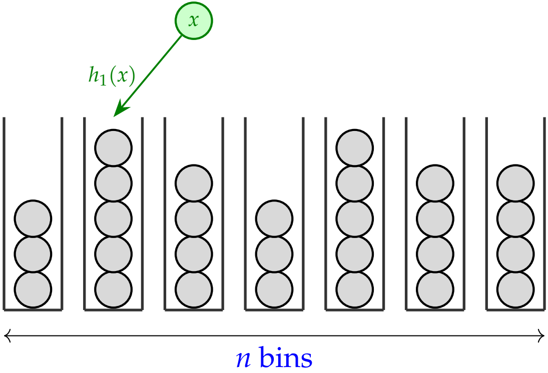
# Baseline 1: The Single-Choice Strategy

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



# Baseline 1: The Single-Choice Strategy

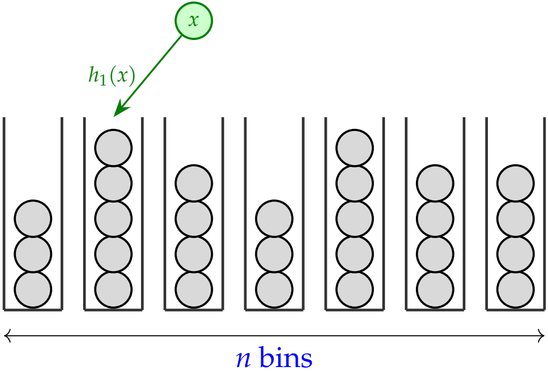
To insert a ball $x$, just put it in bin $h_1(x)$:




- This is history-independent ✓

# Baseline 1: The Single-Choice Strategy

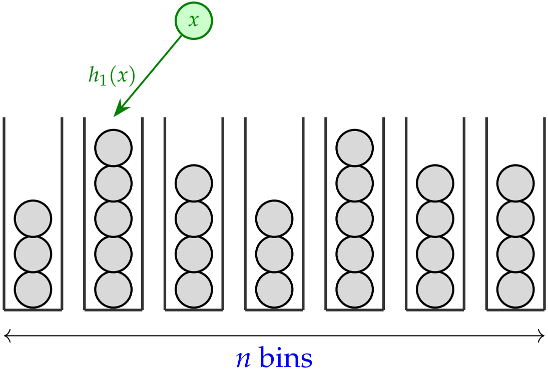
To insert a ball $x$, just put it in bin $h_1(x)$:




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

# Baseline 1: The Single-Choice Strategy

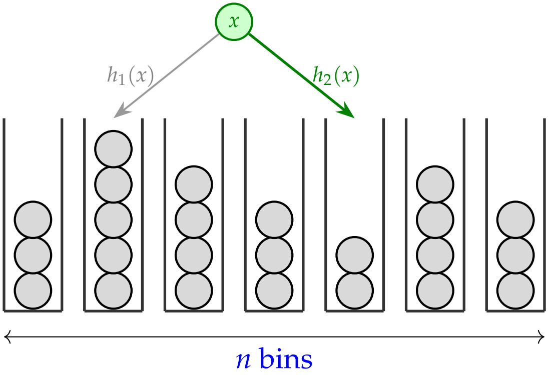
To insert a ball $x$, just put it in bin $h_1(x)$:




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

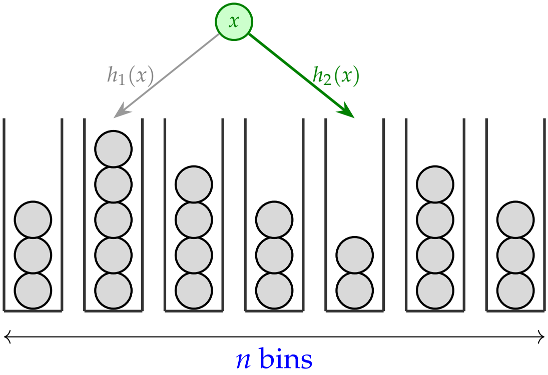
# Baseline 2: Greedy Insertions

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



# Baseline 2: Greedy Insertions

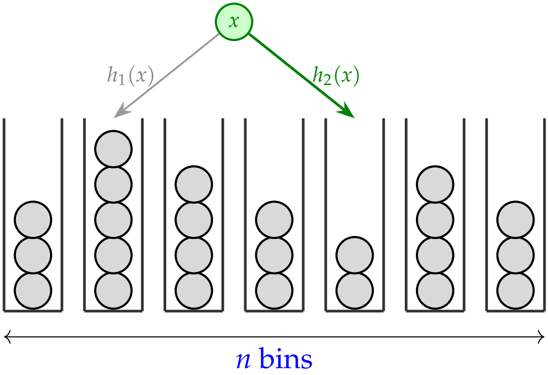
To insert a ball $x$, put it in the emptier of its choices:




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

# Baseline 2: Greedy Insertions

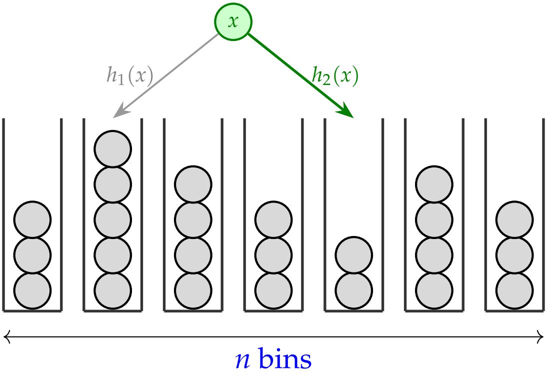
To insert a ball $x$, put it in the emptier of its choices:




- 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:

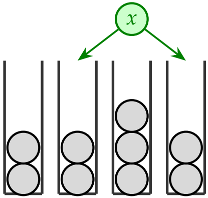



- This is **not** history-independent ✗
- The recourse is 0 ✓
- In the insertion-only case, the overload is $O(\log \log n)$ ✓

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

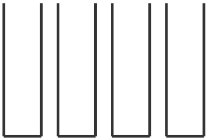
# Warmup: History-Independent Greedy

# Warmup: History-Independent Greedy



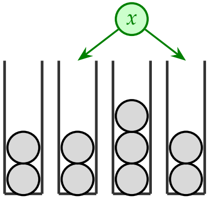

Current state $A_S$

$\longrightarrow$



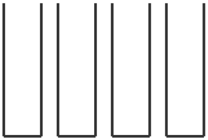
Computing $A_{S \cup \{x\}}$

# WARMUP: HISTORY-INDEPENDENT GREEDY




Current state $A_S$

$\longrightarrow$

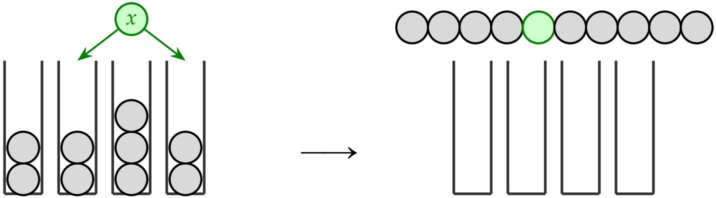


Computing $A_{S\cup\{x\}}$

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

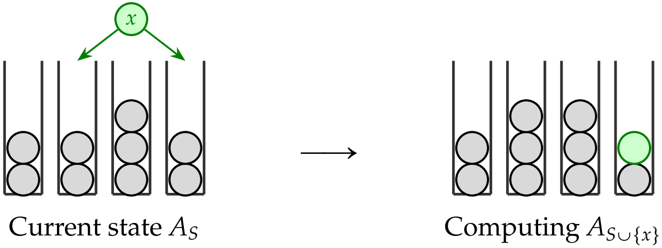



Current state $A_S$     Computing $A_{S \cup \{x\}}$

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.

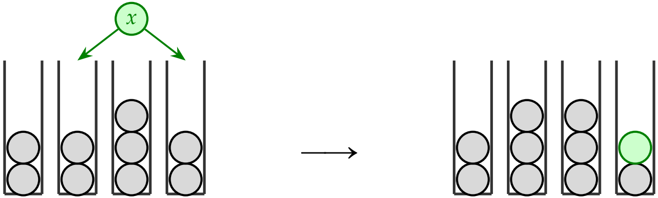
# Warmup: History-Independent Greedy




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

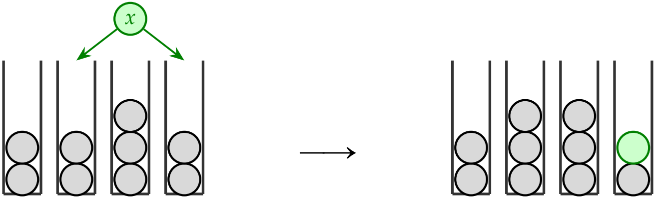
1. Empty out the bins.
2. Sort the balls in $S \cup \{x\}$ to get $x_1, x_2, \ldots$
3. Insert the balls in sorted order using greedy.

# Analyzing History-Independent Greedy




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

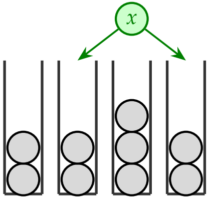
# Analyzing History-Independent Greedy



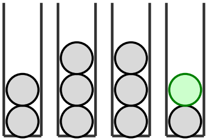

Current state $A_S$ $\longrightarrow$ Computing $A_{S \cup \{x\}}$

- The algorithm is history independent ✓

# ANALYZING HISTORY-INDEPENDENT GREEDY



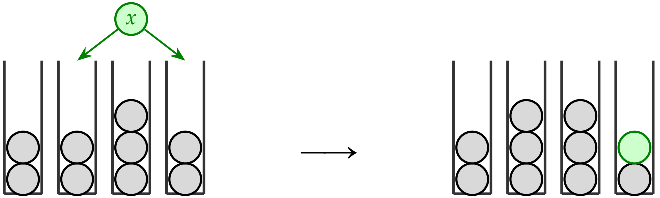

Current state $A_S$



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

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

# Analyzing History-Independent Greedy

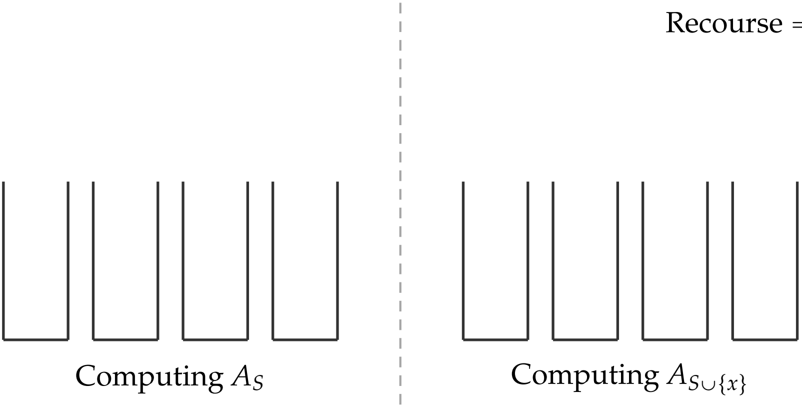



Current state $A_S$ Computing $A_{S \cup \{x\}}$

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

# Analyzing the Recourse

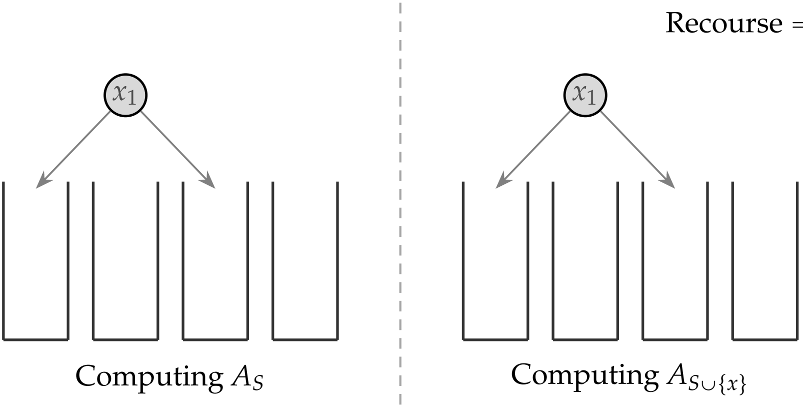
Recourse $= 0$




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

# Analyzing the Recourse

Recourse $= 0$

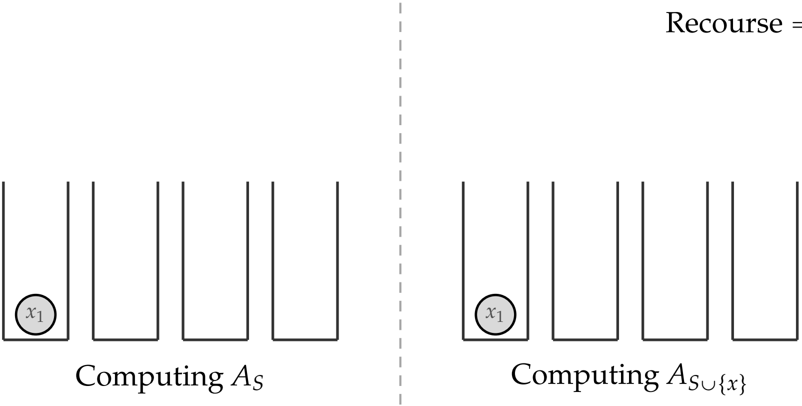



Computing $A_S$

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

# Analyzing the Recourse

Recourse = 0

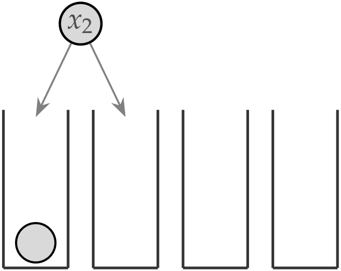



Computing $A_S$

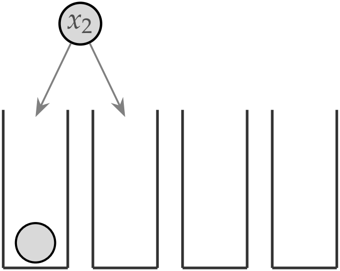
Computing $A_{S \cup \{x\}}$

# Analyzing the Recourse

Recourse $= 0$



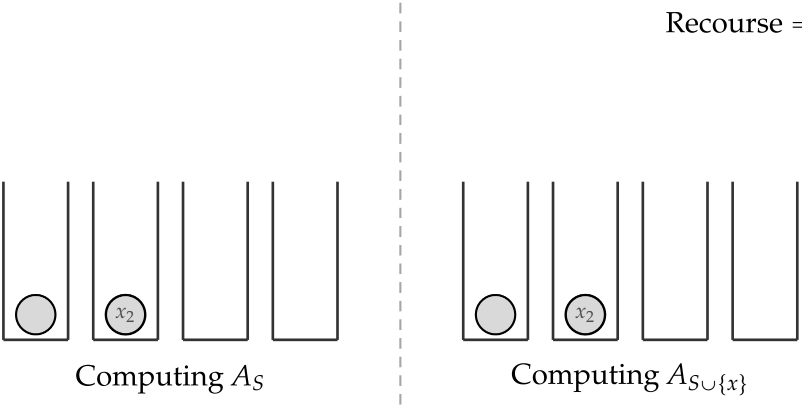

Computing $A_S$




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

# Analyzing the Recourse

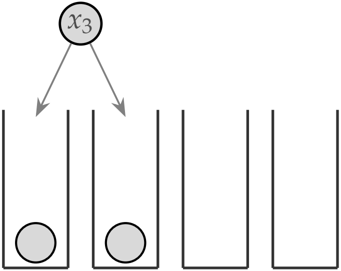
Recourse $= 0$



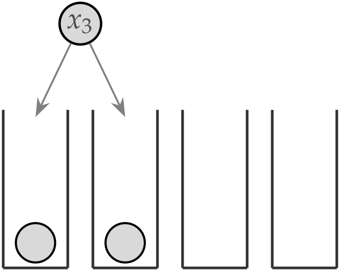

Computing $A_S$ | Computing $A_{S \cup \{x\}}$

# Analyzing the Recourse

Recourse = 0



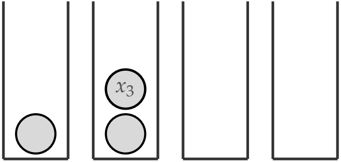

Computing $A_S$



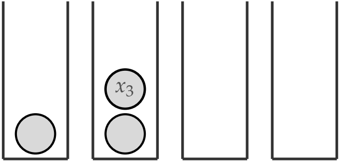

Computing $A_{S \cup \{x\}}$

# Analyzing the Recourse

Recourse $= 0$



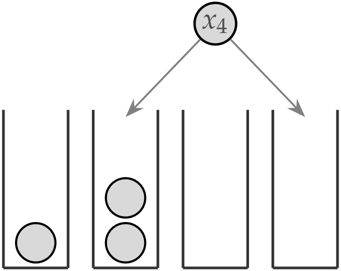

Computing $A_S$



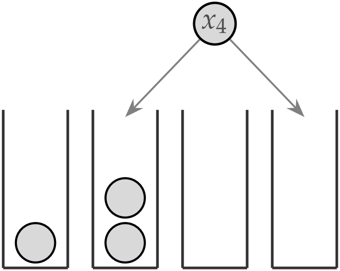

Computing $A_{S \cup \{x\}}$

# ANALYZING THE RECOURSE

Recourse $= 0$



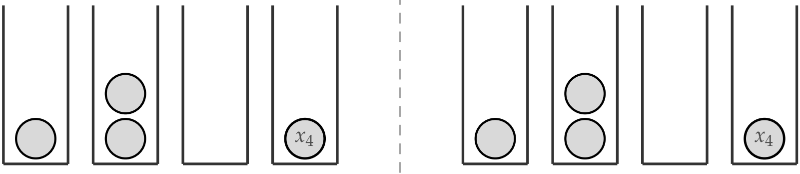

Computing $A_S$




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

# Analyzing the Recourse

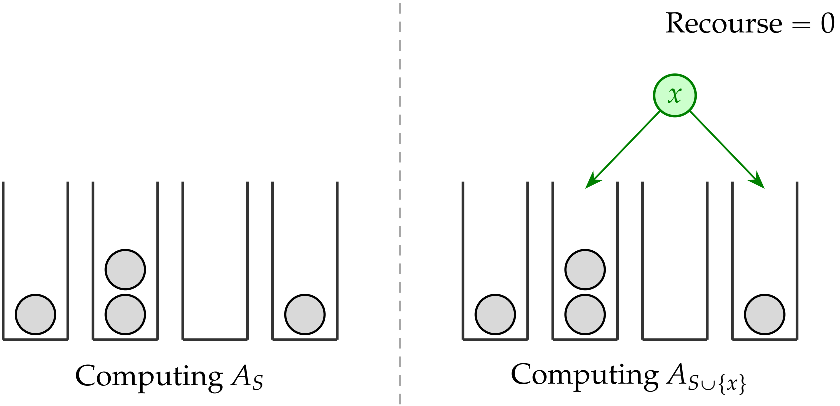
Recourse $= 0$




Computing $A_S$

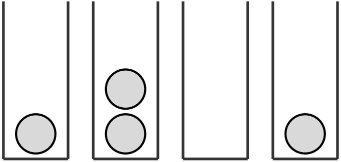
Computing $A_{S \cup \{x\}}$

# Analyzing the Recourse

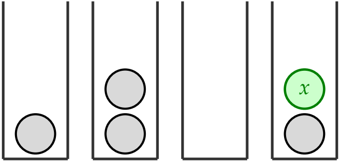


# Analyzing the Recourse

Recourse = 0



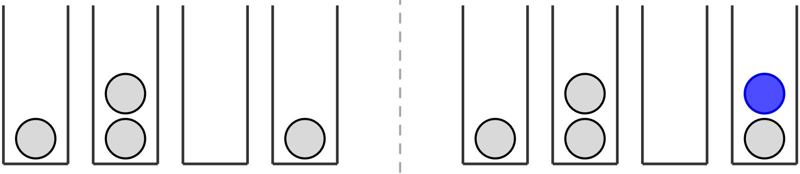
Computing $A_S$




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

# Analyzing the Recourse

Recourse $= 0$



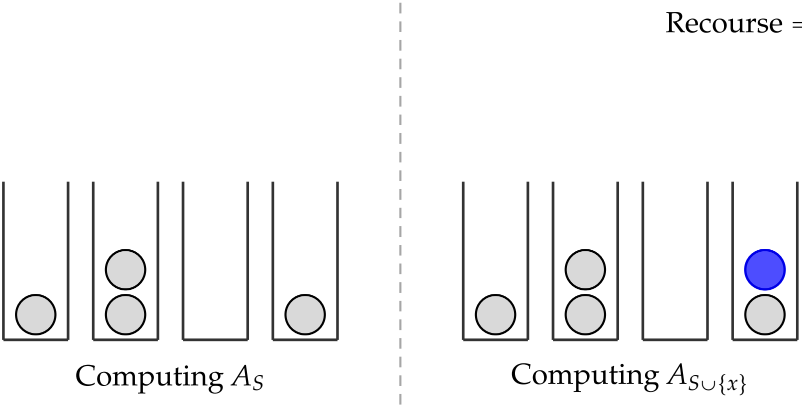
Computing $A_S$

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

Subsequent balls will experience either:

# Analyzing the Recourse

Recourse = 0



Computing $A_S$

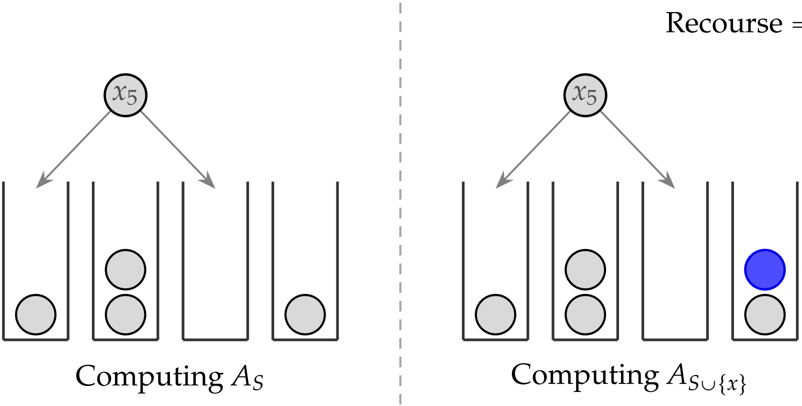
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse

# Analyzing the Recourse

Recourse = 0

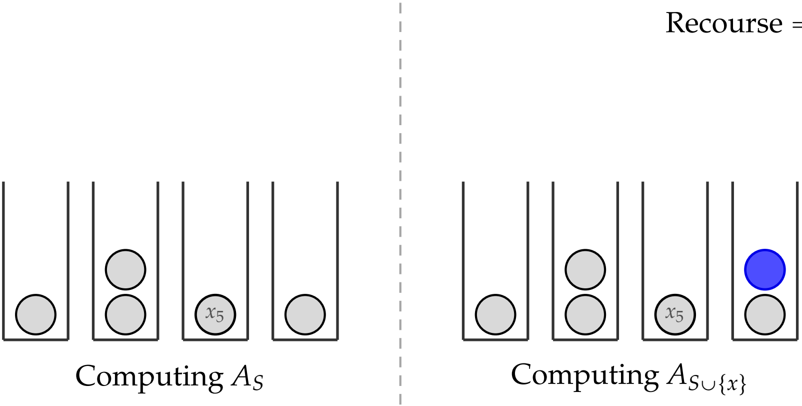



Future insertions will experience either:

1. No recourse

# Analyzing the Recourse

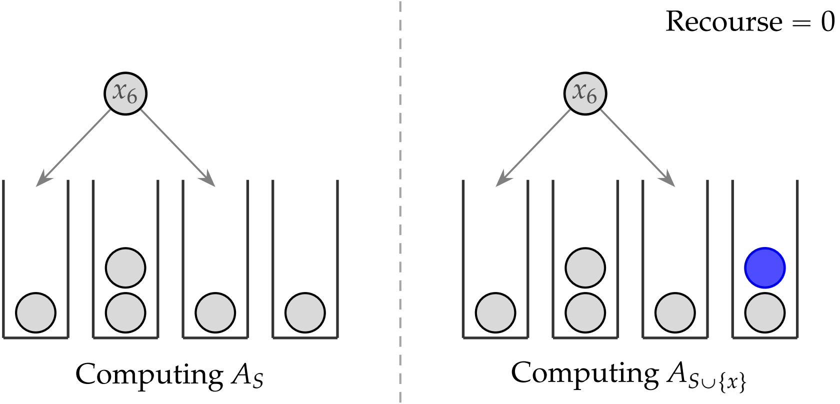
Recourse = 0




Subsequent balls will experience either:

1. No recourse

# Analyzing the Recourse

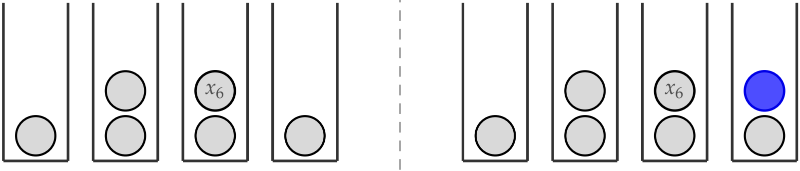



Subsequent balls will experience either:

1. No recourse

# Analyzing the Recourse

Recourse $= 0$

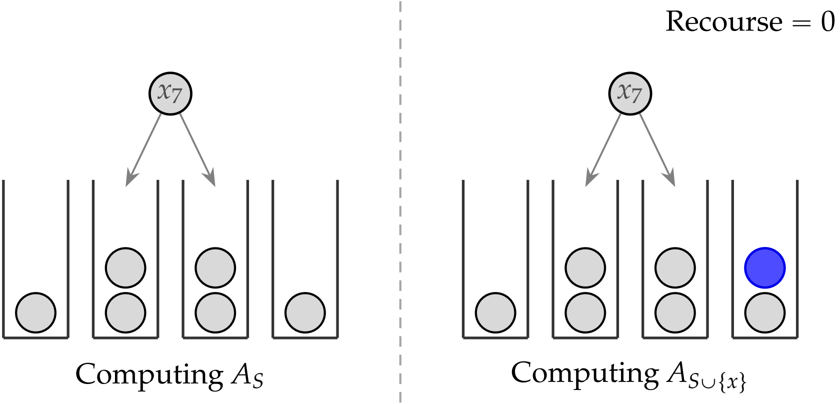



Computing $A_S$ Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse

# Analyzing the Recourse

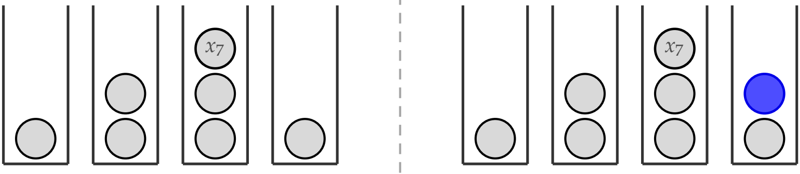



Subsequent balls will experience either:

1. No recourse

# Analyzing the Recourse

Recourse $= 0$



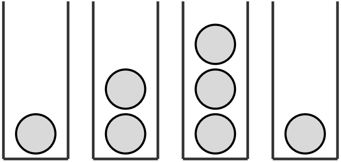

Computing $A_S$ | Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

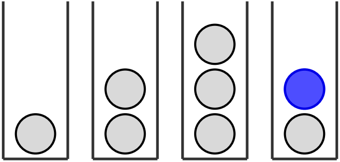
1. No recourse

# Analyzing the Recourse

Recourse $= 0$



Computing $A_S$



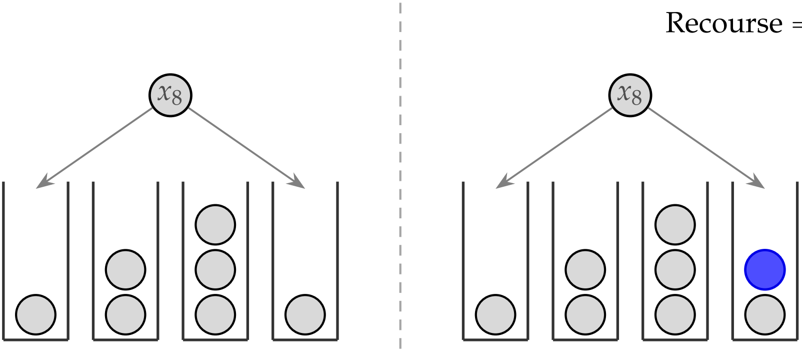
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse
2. Recourse

# ANALYZING THE RECOURSE

Recourse = 0




Computing $A_S$

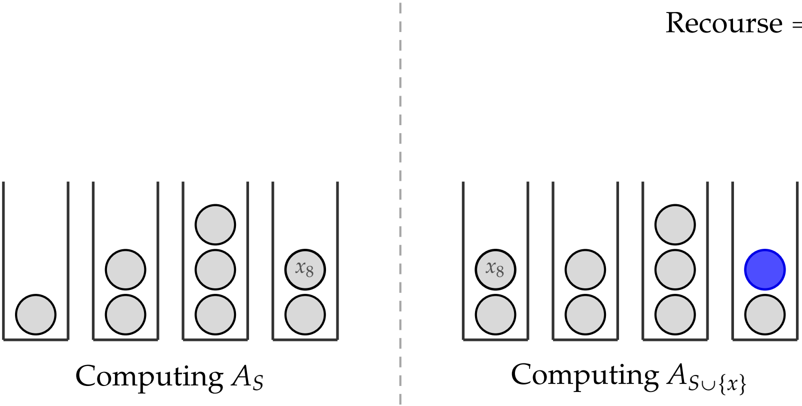
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse
2. Recourse

# ANALYZING THE RECOURSE

Recourse = 1

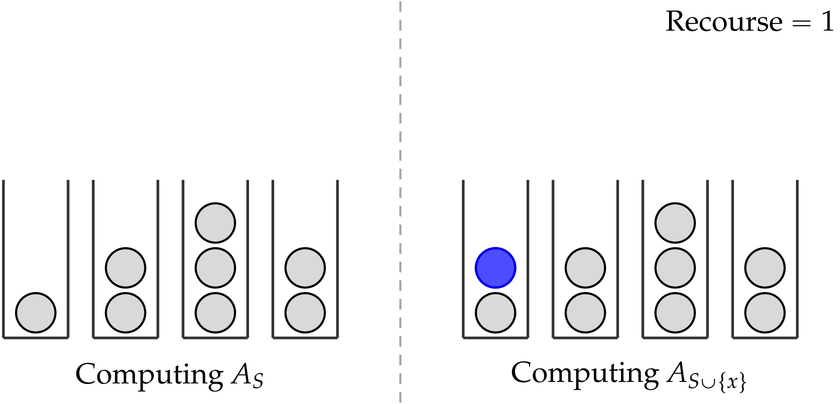



Computing $A_S$ | Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse
2. Recourse

# Analyzing the Recourse

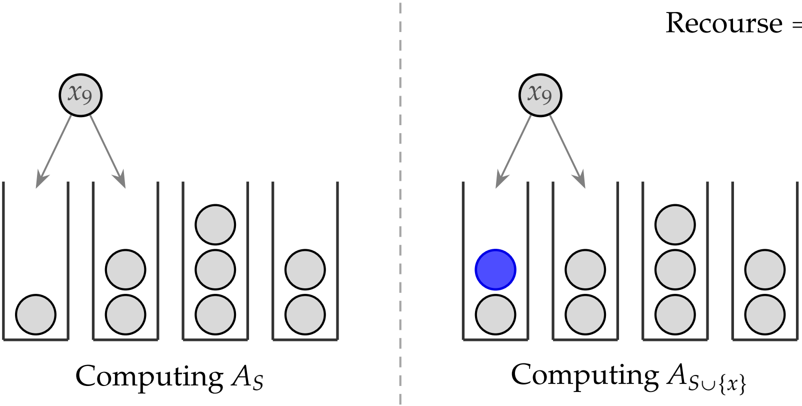



Subsequent balls will experience either:

1. No recourse
2. Recourse

# Analyzing the Recourse

Recourse = 1

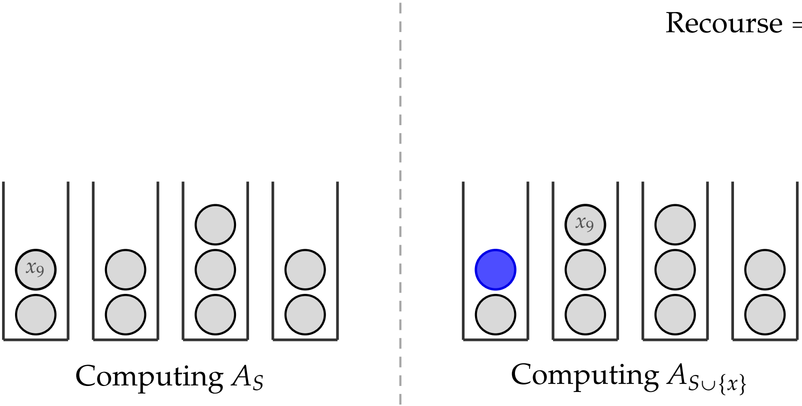



Subsequent balls will experience either:

1. No recourse
2. Recourse

# Analyzing the Recourse

Recourse = 2




Computing $A_S$

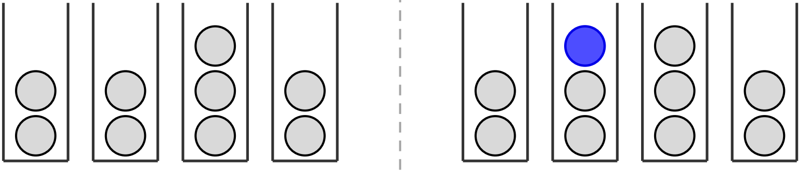
Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

1. No recourse
2. Recourse

# Analyzing the Recourse

Recourse $= 2$



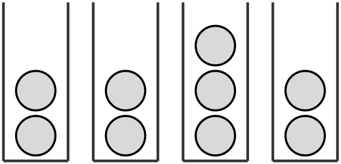
Computing $A_S$ Computing $A_{S \cup \{x\}}$

Subsequent balls will experience either:

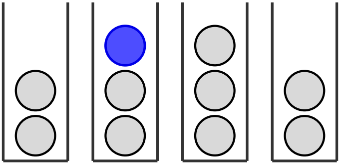
1. No recourse
2. Recourse

# Analyzing the Recourse

Recourse $= 2$



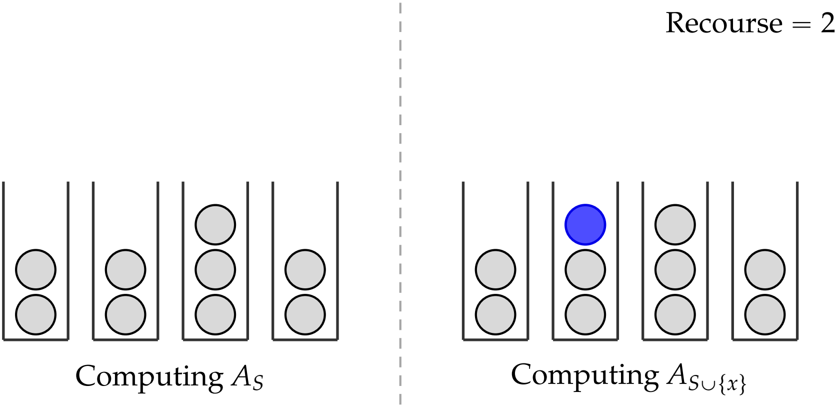
Computing $A_S$



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

Two key observations:

# Analyzing the Recourse

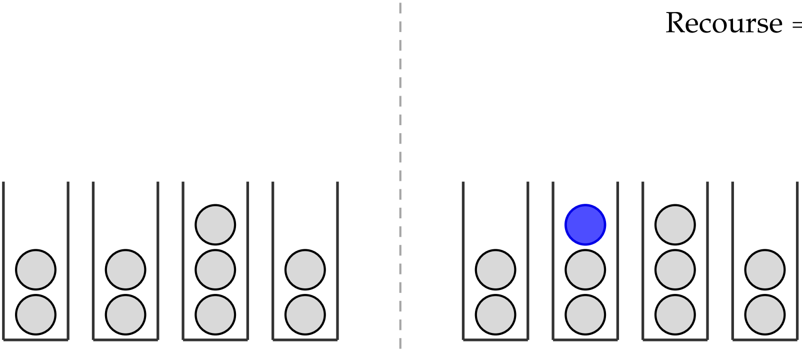



Two key observations:

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

# Analyzing the Recourse

Recourse = 2



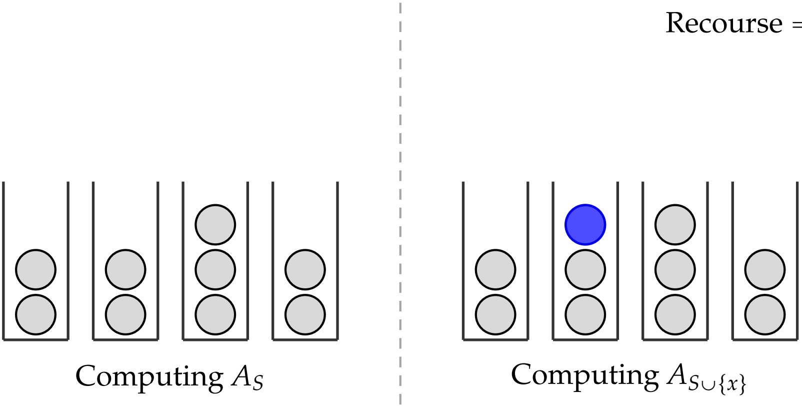
Computing $A_S$ | Computing $A_{S \cup \{x\}}$

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

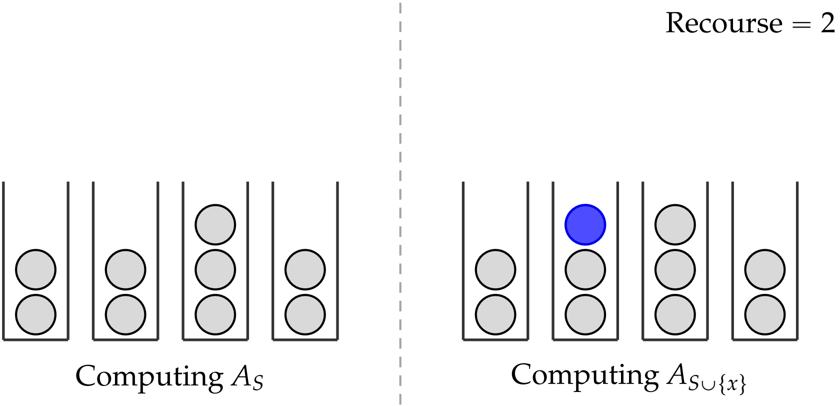


Computing $A_S$

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 $\cancel{O(\log \log (m/n))}$ $O(m/n)$.

# Rest of Talk

1. A Simple Warmup ✓
2. The Full Algorithm

# Part 2: The Full Algorithm

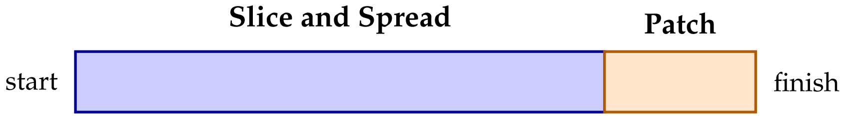
# The Full Algorithm

**Theorem**

There exists a history-independent solution with:

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

**Bird's-eye view:**



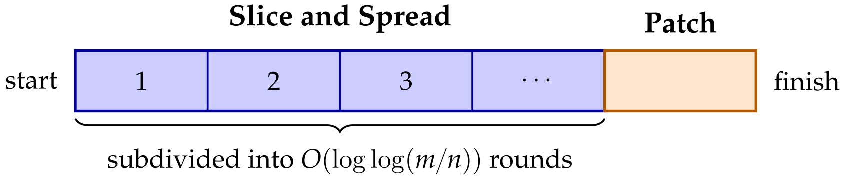
# The Full Algorithm

**Theorem**

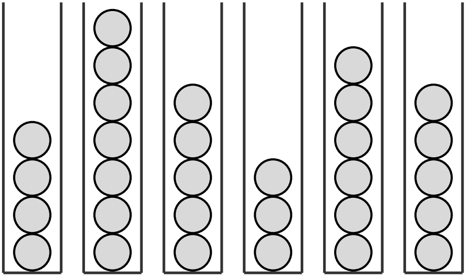
There exists a history-independent solution with:

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

**Bird's-eye view:**

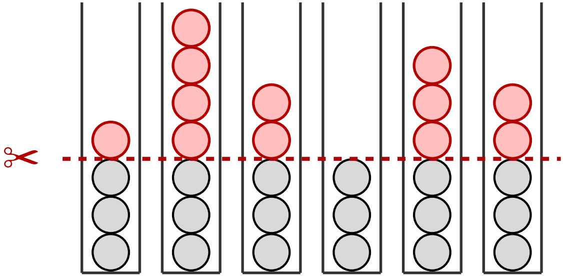


# SLICE AND SPREAD



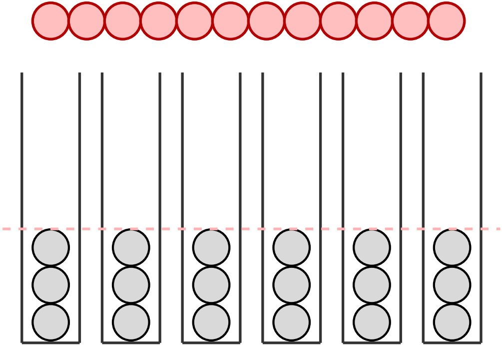
1. **Slice** off the jagged surface

# Slice and Spread



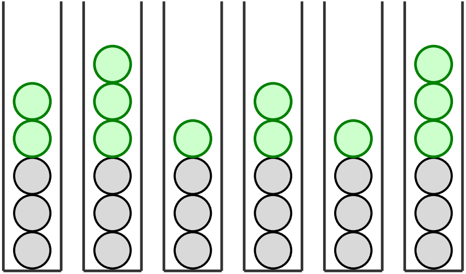
1. **Slice** off the jagged surface

# Slice and Spread



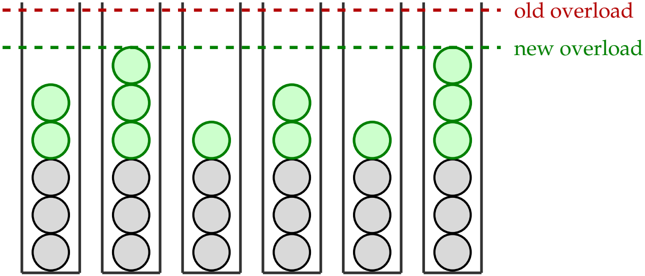
1. **Slice** off the jagged surface
2. **Spread** balls to their second-choice bins

# Slice and Spread



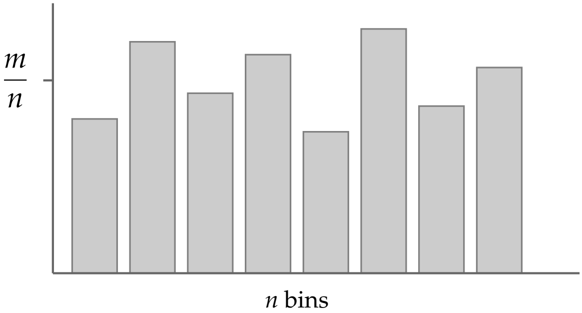
1. **Slice** off the jagged surface
2. **Spread** balls to their second-choice bins

# Slice and Spread

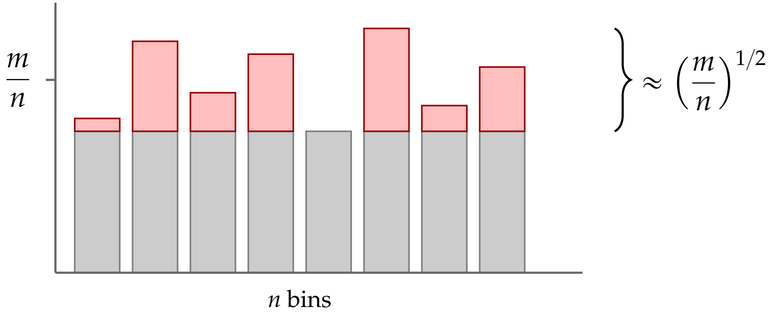



1. **Slice** off the jagged surface
2. **Spread** balls to their second-choice bins

# SLICE AND SPREAD REDUCES OVERLOAD



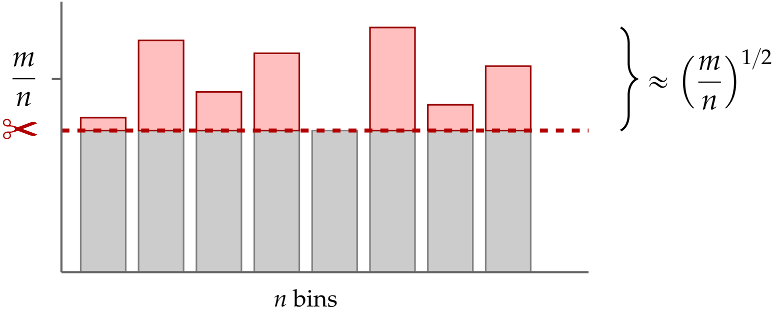
# Slice and Spread Reduces Overload




**Key Fact**

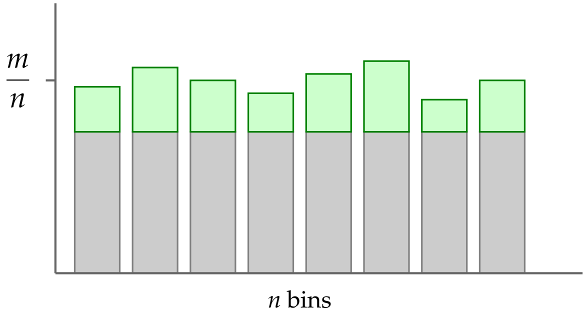
After throwing $m \gg n$ balls uniformly at random into $n$ bins, the bin loads are within roughly $\approx \sqrt{m/n}$ with high probability in $n$.

# Slice and Spread Reduces Overload




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

# Slice and Spread Reduces Overload

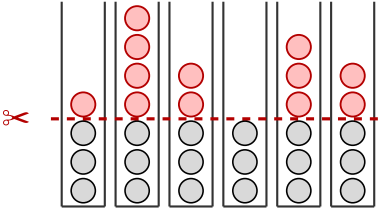



$\} \approx \left(\frac{m}{n}\right)^{1/4}$

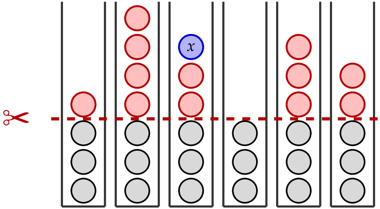
- Balls above dotted line $\approx (mn)^{1/2}$
- By Key Fact, bin loads within $\approx (m/n)^{1/4}$

# What's the Recourse?

# What's the Recourse?

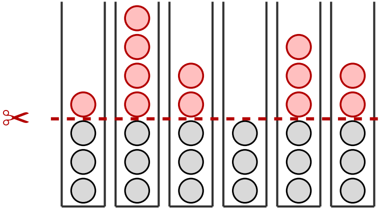


Computing $A_S$

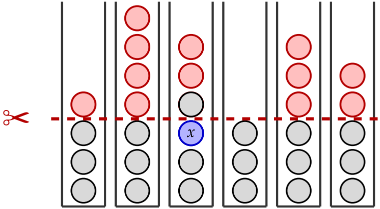



Computing $A_{S \cup \{x\}}$

# What's the Recourse?



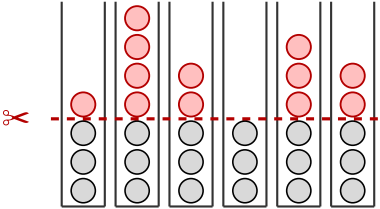
Computing $A_S$



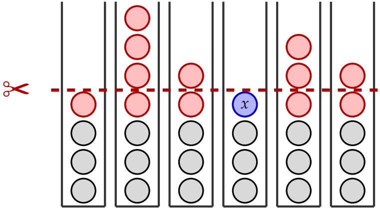


Computing $A_{S \cup \{x\}}$

# WHAT'S THE RECOURSE?

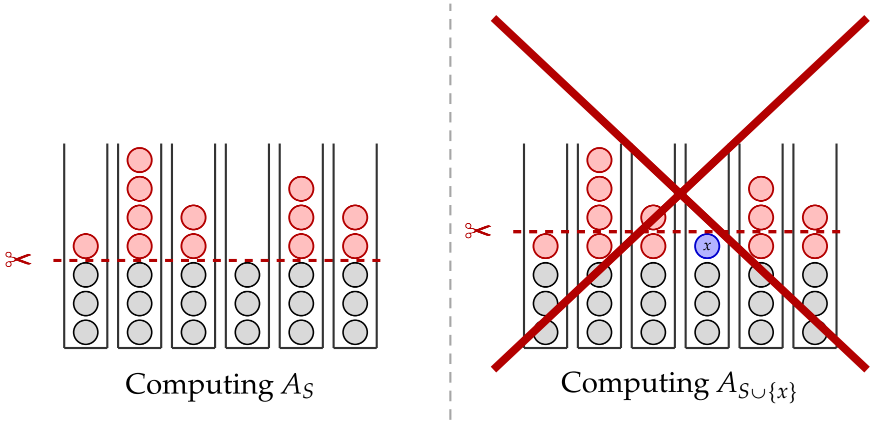


Computing $A_S$




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

# What's the Recourse?

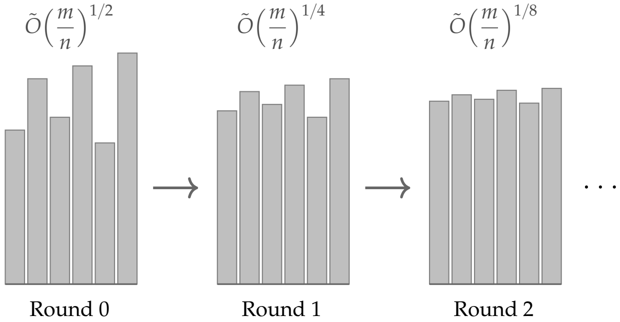



Computing $A_S$

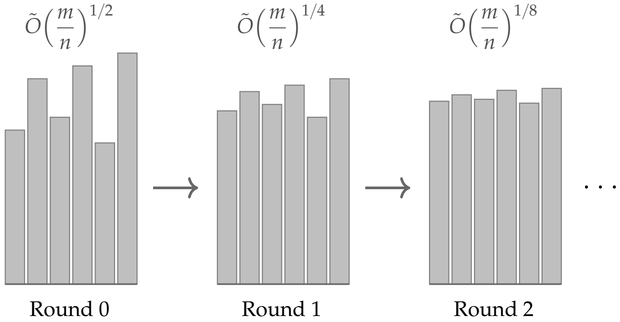
Computing $A_{S \cup \{x\}}$

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

# Repeatedly Slicing and Spreading



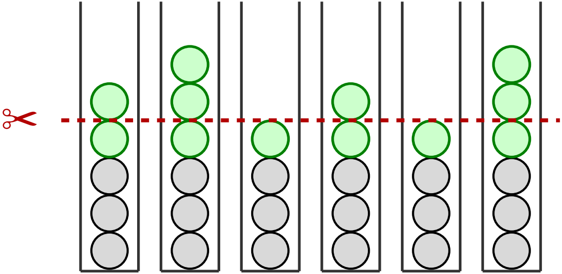
# Repeatedly Slicing and Spreading




## Proposition

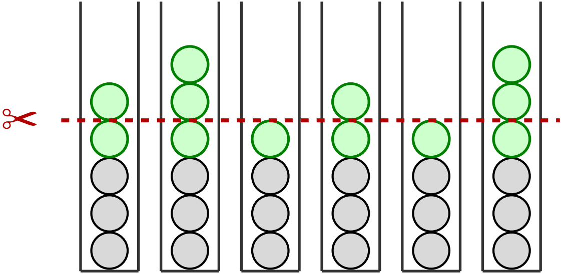
After $O(\log\log(m/n))$ rounds of slice-and-spread, the cumulative overload is $O(n)$ with high probability in $n$. The expected recourse is $O(\log\log(m/n))$.

# Algorithmic Question



**Question:** Which balls do we slice in each round?

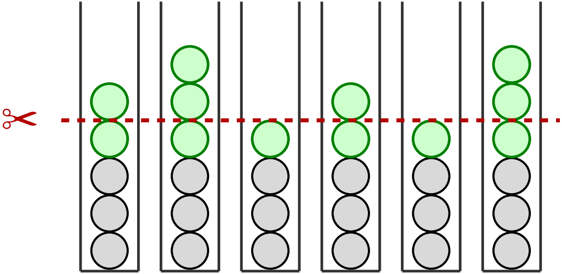
# Algorithmic Question



**Question:** Which balls do we slice in each round?

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

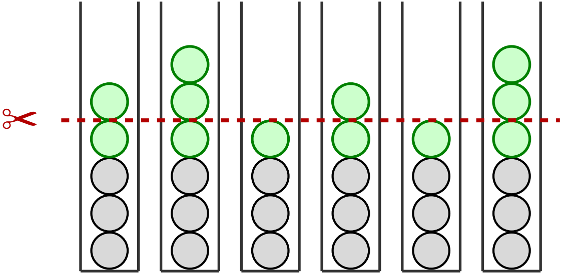
# Algorithmic Question



**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

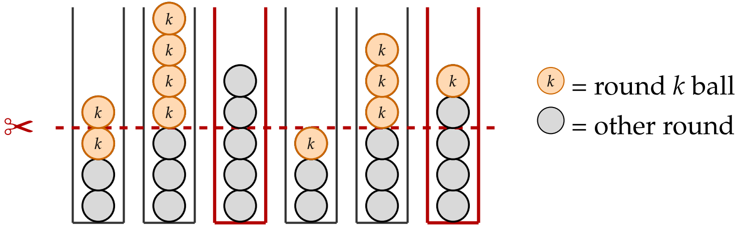
# ALGORITHMIC QUESTION



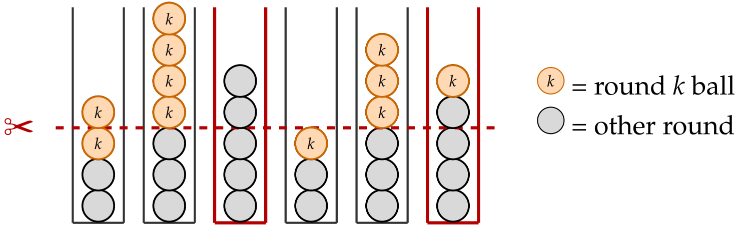
**Question:** Which balls do we slice in each round?

- **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

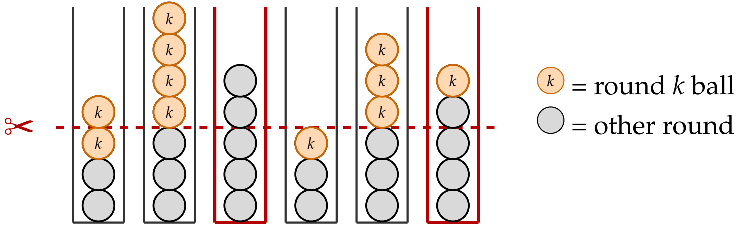


# Challenge 1: Slicing Failures



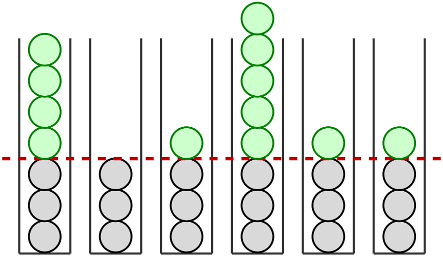

**Challenge:** Some bins may not have enough round-$k$ balls above the line.

# Challenge 1: Slicing Failures

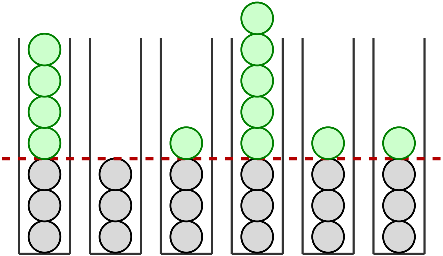



**Challenge:** Some bins may not have enough round-$k$ balls above the line.
**Result:** We can't slice evenly — the jaggedness remains in those bins.

# Challenge 2: Spreading Failures

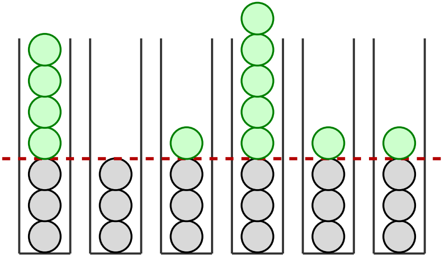


# Challenge 2: Spreading Failures



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

# CHALLENGE 2: SPREADING FAILURES

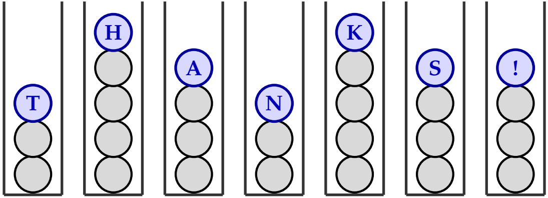


**Challenge:** The spreading step may distribute balls unevenly — creating new jaggedness.

**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