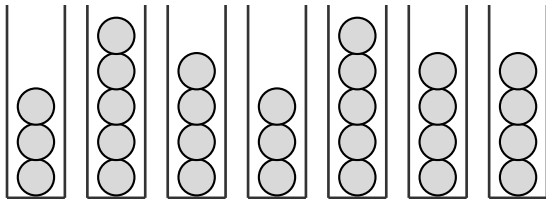


# History-Independent Load Balancing



Michael A. Bender

Stony Brook University

William Kuszmaul

CMU

Elaine Shi

CMU

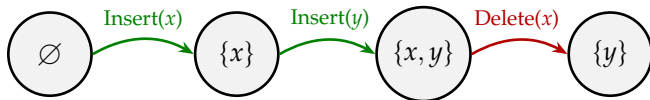
**Rose Silver**

CMU

# HISTORY-INDEPENDENT DATA STRUCTURES

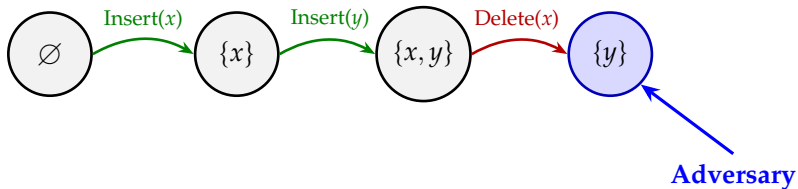
# HISTORY-INDEPENDENT DATA STRUCTURES

History 1:

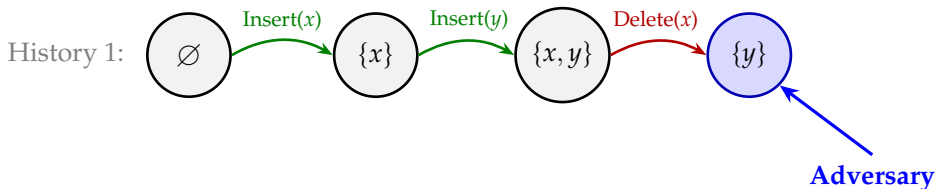


# HISTORY-INDEPENDENT DATA STRUCTURES

History 1:



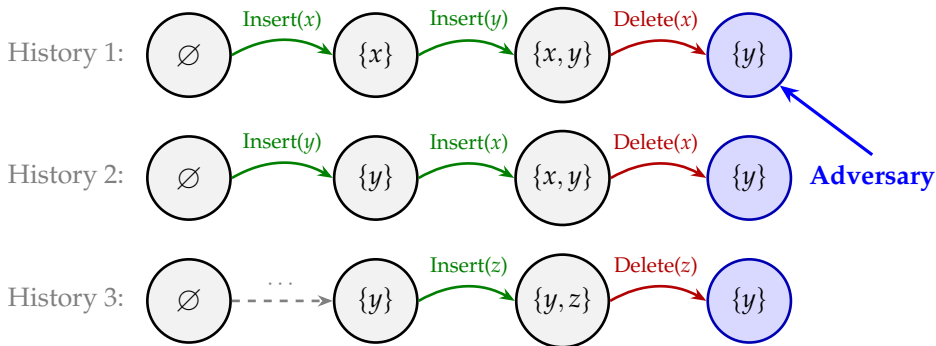
# HISTORY-INDEPENDENT DATA STRUCTURES



**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

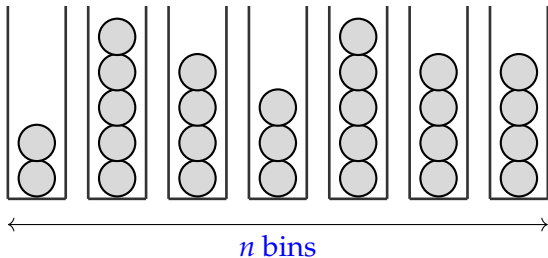
Hash tables, trees, memory allocation, PMAs, graph algorithms, cache-oblivious data structures, and more.

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

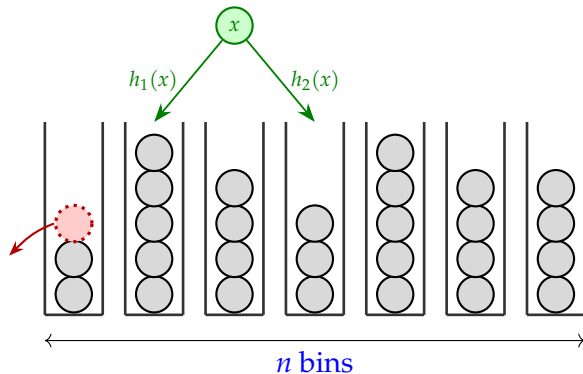
**Yet some basic questions remain open.**

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

# TWO-CHOICE LOAD BALANCING

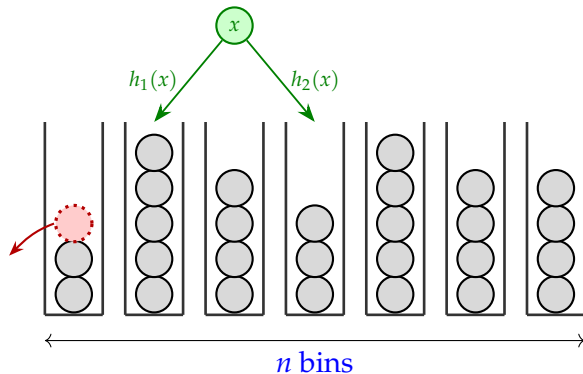


# TWO-CHOICE LOAD BALANCING



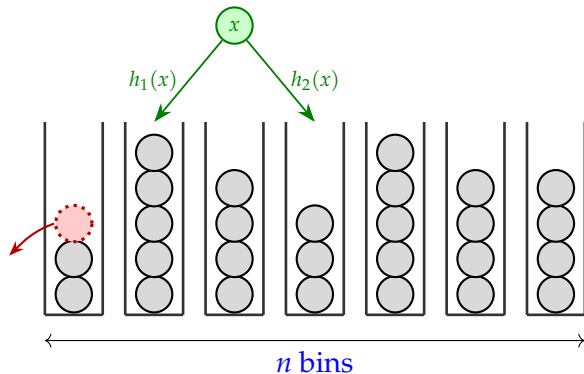
- Balls are **inserted**/**deleted**, with up to  $m$  present at a time.

# TWO-CHOICE LOAD BALANCING



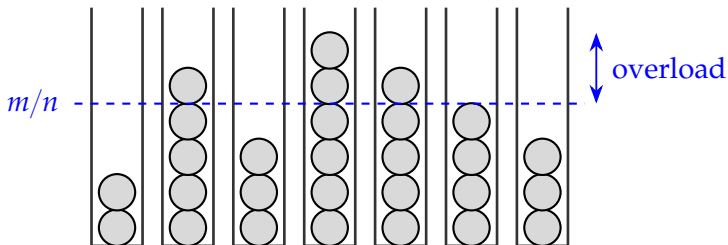
- ▶ Balls are **inserted**/**deleted**, with up to  $m$  present at a time.
- ▶ Each ball has two random bins where it can go.

# 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.

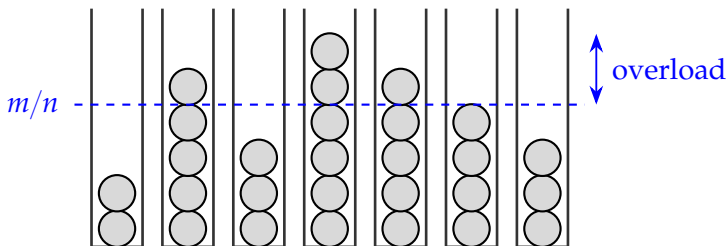
## TWO GOALS



### 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

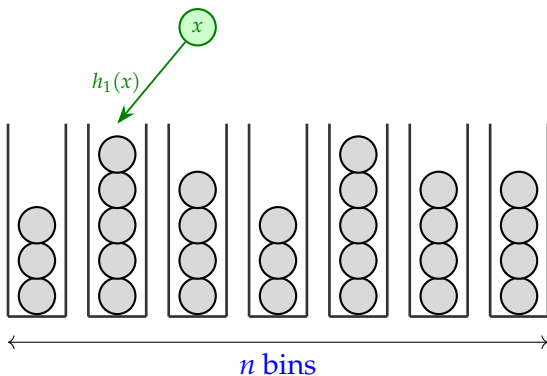
**Theorem:** There exists a history-independent solution with:

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



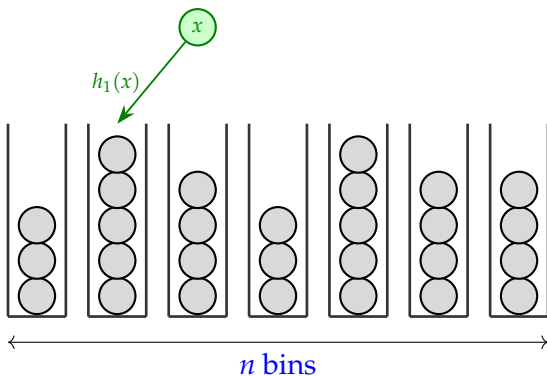
## BASLINE 1: THE SINGLE-CHOICE STRATEGY

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



## BASLINE 1: THE SINGLE-CHOICE STRATEGY

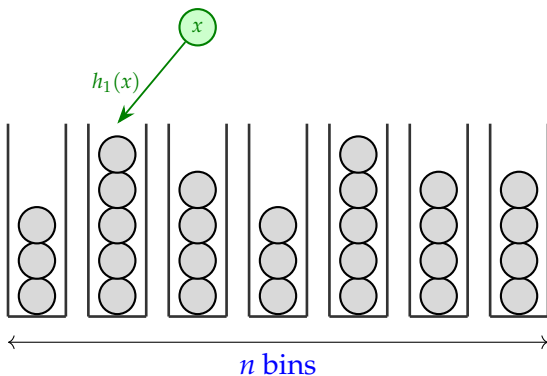
To insert a ball  $x$ , just put it in bin  $h_1(x)$ :



- This is history-independent ✓

## BASLINE 1: THE SINGLE-CHOICE STRATEGY

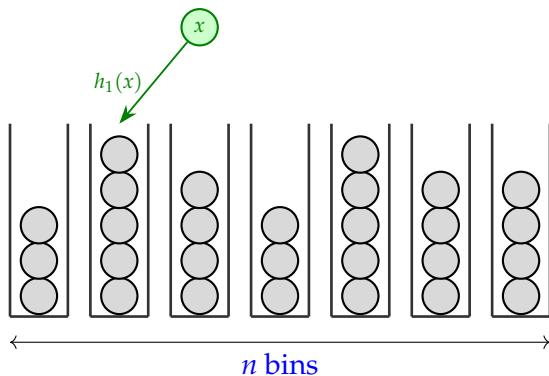
To insert a ball  $x$ , just put it in bin  $h_1(x)$ :



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

# BASELINE 1: THE SINGLE-CHOICE STRATEGY

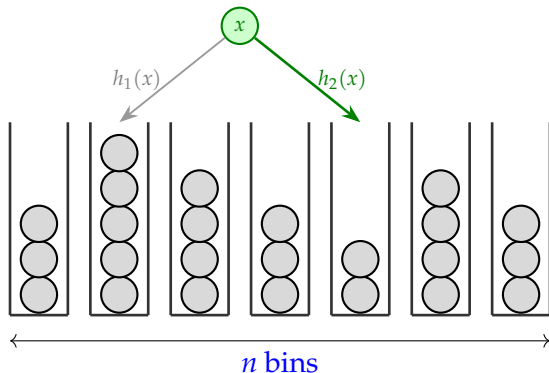
To insert a ball  $x$ , just put it in bin  $h_1(x)$ :



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

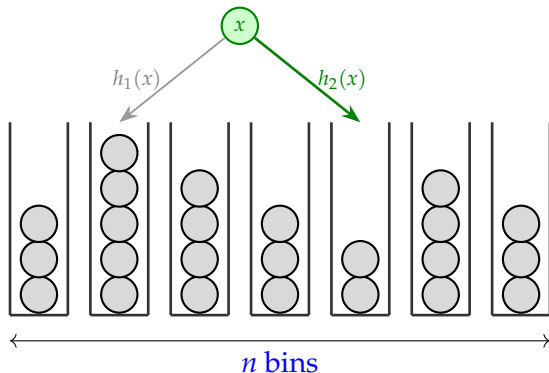
## BASELINE 2: GREEDY INSERTIONS

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



## BASELINE 2: GREEDY INSERTIONS

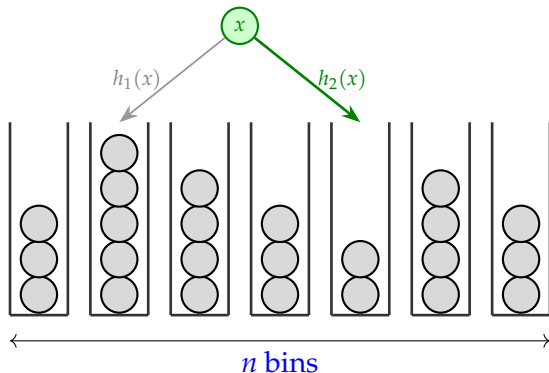
To insert a ball  $x$ , put it in the **emptier** of its choices:



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

## BASELINE 2: GREEDY INSERTIONS

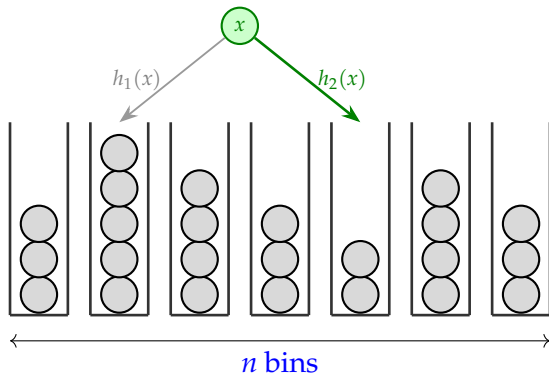
To insert a ball  $x$ , put it in the **emptier** of its choices:



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

## BASLINE 2: GREEDY INSERTIONS

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

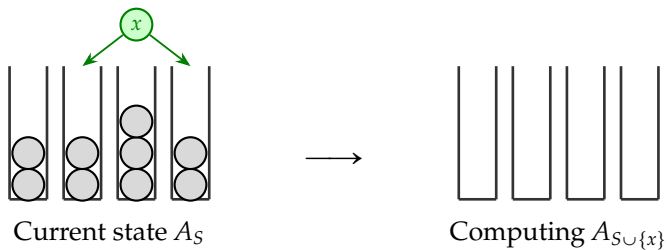


- ▶ This is **not** history-independent ✗
  - ▶ The recourse is 0 ✓
  - ▶ In the insertion-only case, the overload is  $O(\log \log n)$  ✓
- [Berenbrink, Czumaj, Steger, and Vöcking '00]

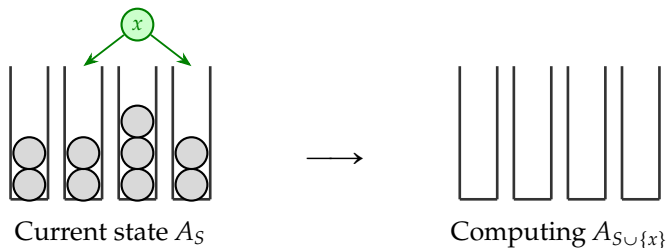


## WARMUP: HISTORY-INDEPENDENT GREEDY

## WARMUP: HISTORY-INDEPENDENT GREEDY



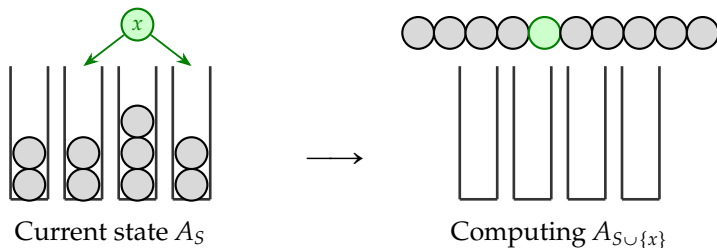
## WARMUP: HISTORY-INDEPENDENT GREEDY



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.

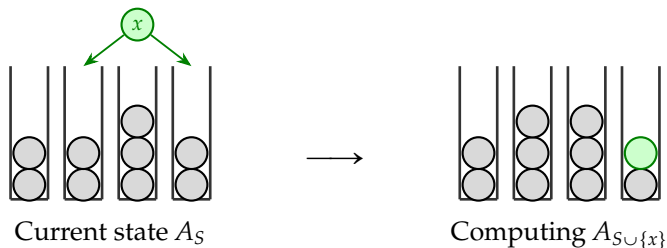
## WARMUP: HISTORY-INDEPENDENT GREEDY



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.

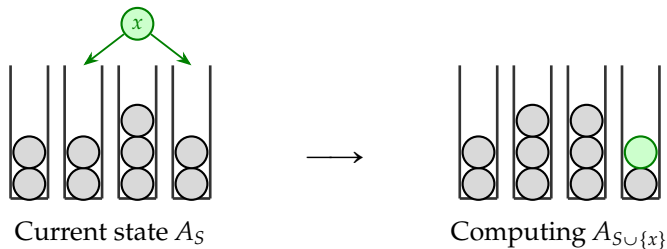
## WARMUP: HISTORY-INDEPENDENT GREEDY



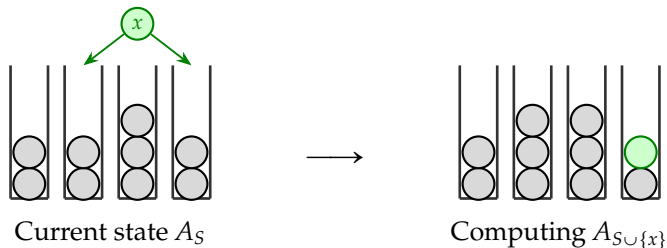
To compute  $A_{S \cup \{x\}}$ :

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

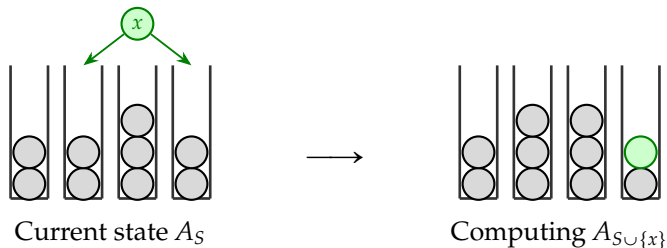


# ANALYZING HISTORY-INDEPENDENT GREEDY



- The algorithm is history independent ✓

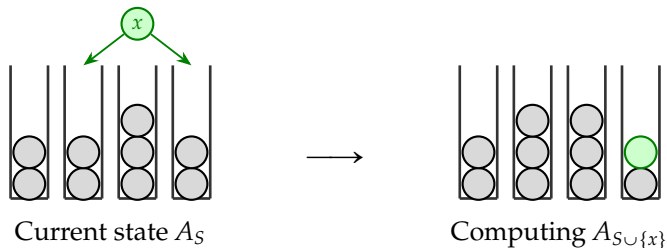
# ANALYZING HISTORY-INDEPENDENT GREEDY



- ▶ 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

Recourse = 0



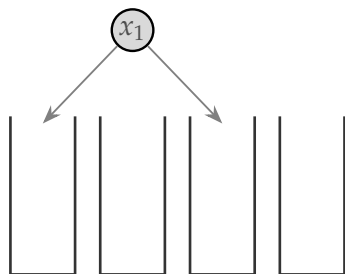
Computing  $A_S$



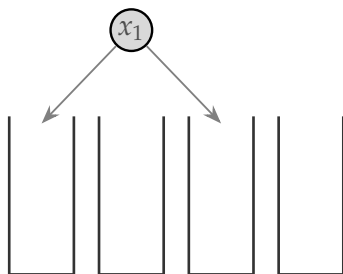
Computing  $A_{S \cup \{x\}}$

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

# ANALYZING THE RECOURSE



Computing  $A_S$



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

Recourse = 0

# ANALYZING THE RECOURSE

Recourse = 0

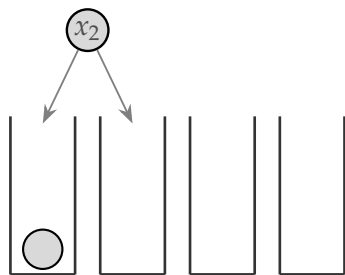


Computing  $A_S$

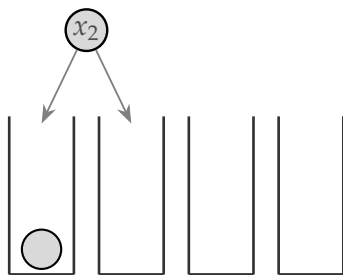


Computing  $A_{S \cup \{x\}}$

# ANALYZING THE RECOURSE



Computing  $A_S$

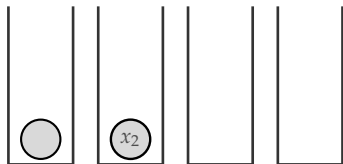


Computing  $A_{S \cup \{x\}}$

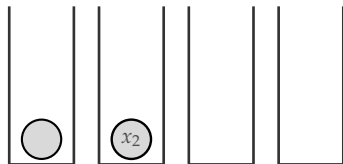
Recourse = 0

# ANALYZING THE RECOURSE

Recourse = 0

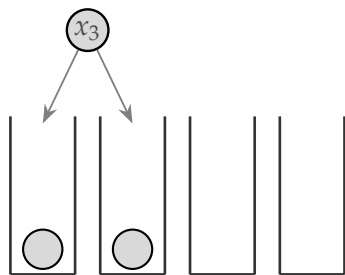


Computing  $A_S$

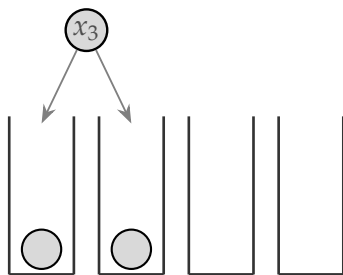


Computing  $A_{S \cup \{x\}}$

# ANALYZING THE RECOURSE



Computing  $A_S$

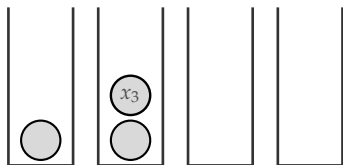


Computing  $A_{S \cup \{x\}}$

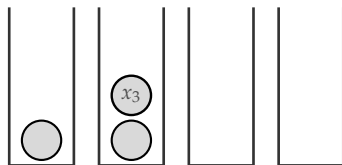
Recourse = 0

# ANALYZING THE RECOURSE

Recourse = 0



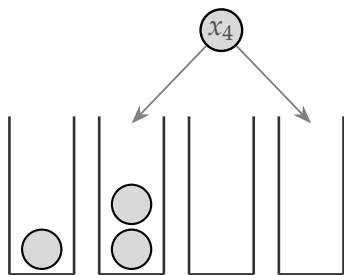
Computing  $A_S$



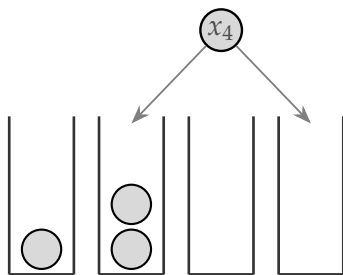
Computing  $A_{S \cup \{x\}}$



# ANALYZING THE RECOURSE



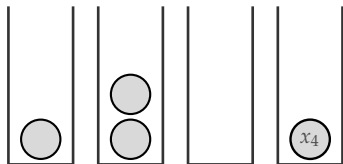
Computing  $A_S$



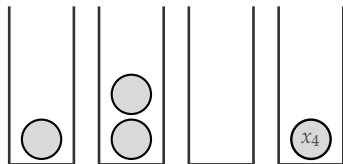
Computing  $A_{S \cup \{x\}}$

# ANALYZING THE RECOURSE

Recourse = 0

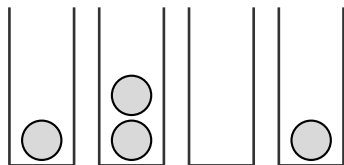


Computing  $A_S$

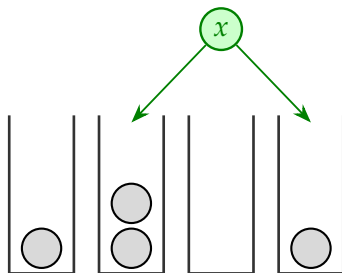


Computing  $A_{S \cup \{x\}}$

# ANALYZING THE RECOURSE



Computing  $A_S$

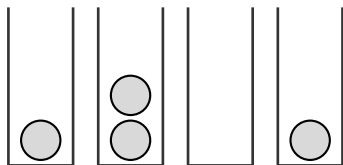


Computing  $A_{S \cup \{x\}}$

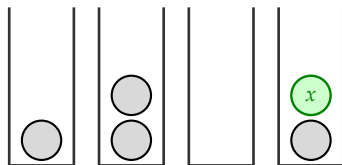
Recourse = 0

# ANALYZING THE RECOURSE

Recourse = 0



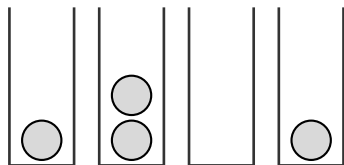
Computing  $A_S$



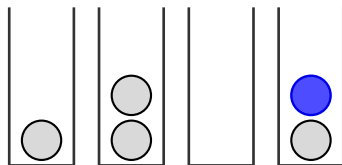
Computing  $A_{S \cup \{x\}}$

# ANALYZING THE RECOURSE

Recourse = 0



Computing  $A_S$

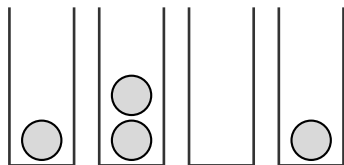


Computing  $A_{S \cup \{x\}}$

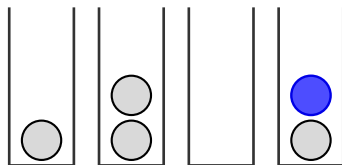
Subsequent balls will experience either:

# ANALYZING THE RECOURSE

Recourse = 0



Computing  $A_S$

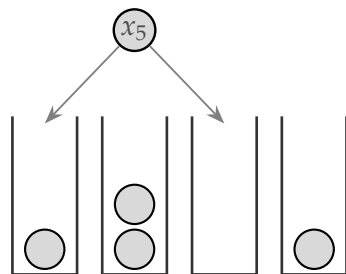


Computing  $A_{S \cup \{x\}}$

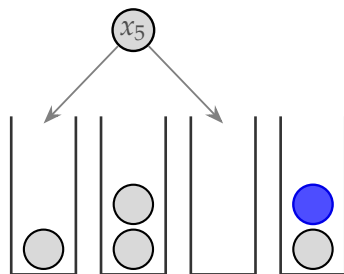
Subsequent balls will experience either:

1. No recourse

# ANALYZING THE RECOURSE



Computing  $A_S$



Recourse = 0

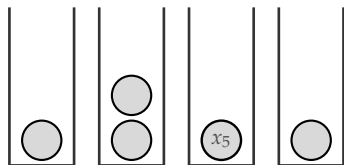
Computing  $A_{S \cup \{x\}}$

Future insertions will experience either:

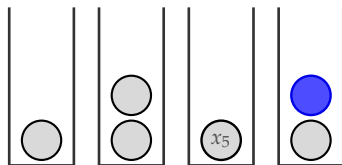
1. No recourse

# ANALYZING THE RECOURSE

Recourse = 0



Computing  $A_S$



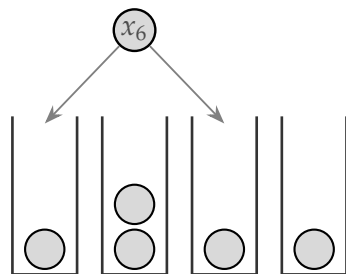
Computing  $A_{S \cup \{x\}}$

Subsequent balls will experience either:

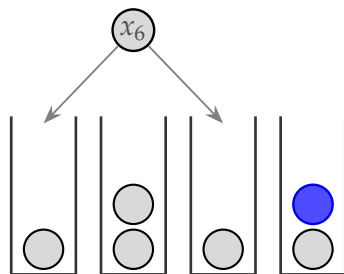
1. No recourse



# ANALYZING THE RECOURSE



Computing  $A_S$



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

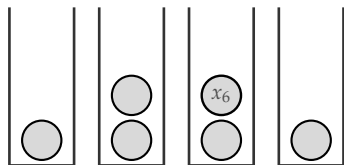
Recourse = 0

Subsequent balls will experience either:

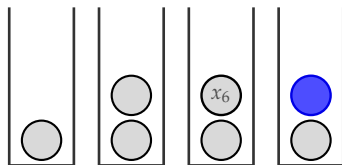
1. No recourse

# ANALYZING THE RECOURSE

Recourse = 0



Computing  $A_S$

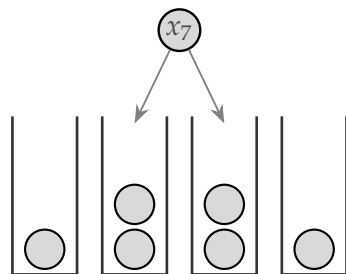


Computing  $A_{S \cup \{x\}}$

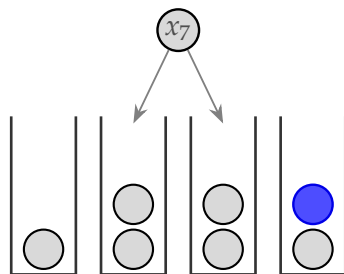
Subsequent balls will experience either:

1. No recourse

# ANALYZING THE RECOURSE



Computing  $A_S$



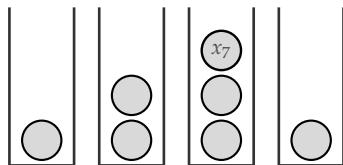
Computing  $A_{S \cup \{x\}}$

Subsequent balls will experience either:

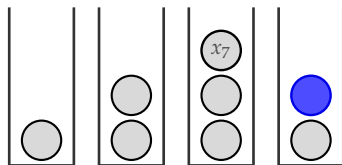
1. No recourse

# ANALYZING THE RECOURSE

Recourse = 0



Computing  $A_S$



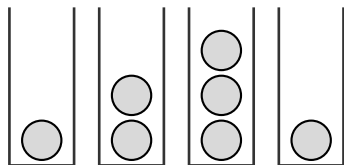
Computing  $A_{S \cup \{x\}}$

Subsequent balls will experience either:

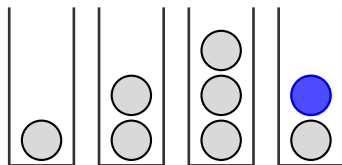
1. No recourse

# ANALYZING THE RECOURSE

Recourse = 0



Computing  $A_S$

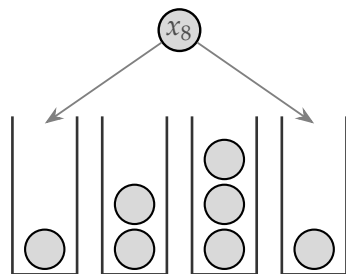


Computing  $A_{S \cup \{x\}}$

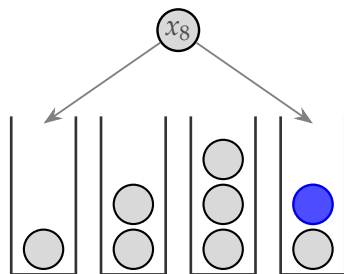
Subsequent balls will experience either:

1. No recourse
2. Recourse

# ANALYZING THE RECOURSE



Computing  $A_S$



Recourse = 0

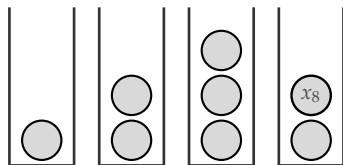
Computing  $A_{S \cup \{x\}}$

Subsequent balls will experience either:

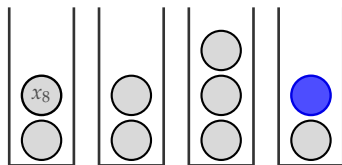
1. No recourse
2. Recourse

# ANALYZING THE RECOURSE

Recourse = 1



Computing  $A_S$



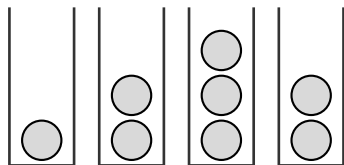
Computing  $A_{S \cup \{x\}}$

Subsequent balls will experience either:

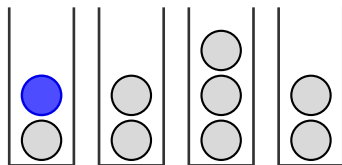
1. No recourse
2. Recourse

# ANALYZING THE RECOURSE

Recourse = 1



Computing  $A_S$



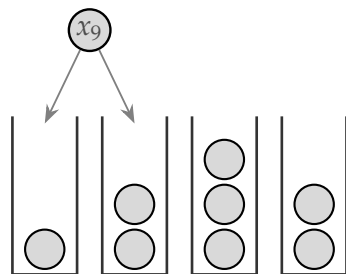
Computing  $A_{S \cup \{x\}}$

Subsequent balls will experience either:

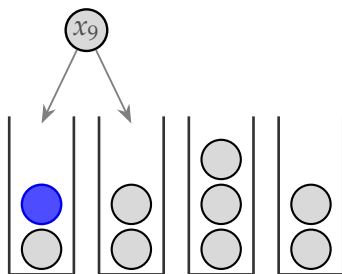
1. No recourse
2. Recourse



# ANALYZING THE RECOURSE



Computing  $A_S$



Recourse = 1

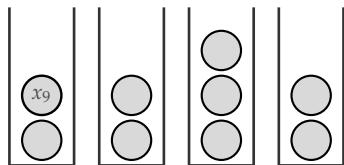
Computing  $A_{S \cup \{x\}}$

Subsequent balls will experience either:

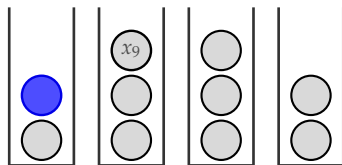
1. No recourse
2. Recourse

# ANALYZING THE RECOURSE

Recourse = 2



Computing  $A_S$



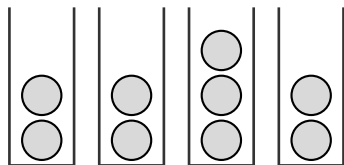
Computing  $A_{S \cup \{x\}}$

Subsequent balls will experience either:

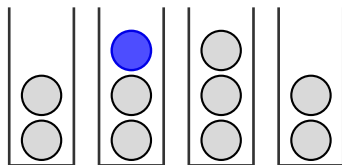
1. No recourse
2. Recourse

# ANALYZING THE RECOURSE

Recourse = 2



Computing  $A_S$



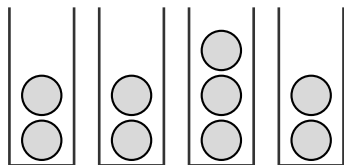
Computing  $A_{S \cup \{x\}}$

Subsequent balls will experience either:

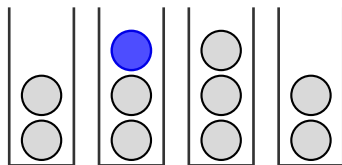
1. No recourse
2. Recourse

# ANALYZING THE RECOURSE

Recourse = 2



Computing  $A_S$

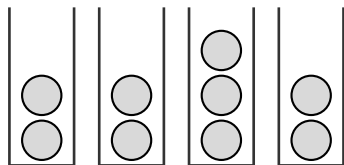


Computing  $A_{S \cup \{x\}}$

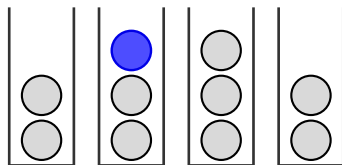
Two key observations:

# ANALYZING THE RECOURSE

Recourse = 2



Computing  $A_S$



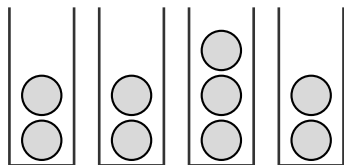
Computing  $A_{S \cup \{x\}}$

Two key observations:

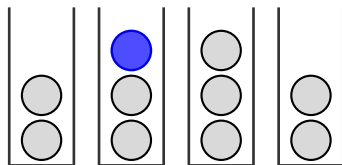
1. There's always one special bin with an extra ball

# ANALYZING THE RECOURSE

Recourse = 2



Computing  $A_S$



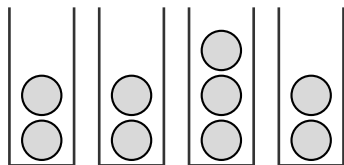
Computing  $A_{S \cup \{x\}}$

Two key observations:

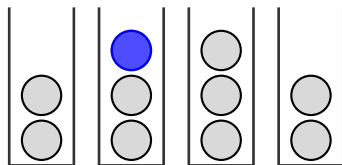
1. There's always one special bin with an extra ball
2. If a ball incurs recourse, one of its choices is the special bin

# ANALYZING THE RECOURSE

Recourse = 2



Computing  $A_S$

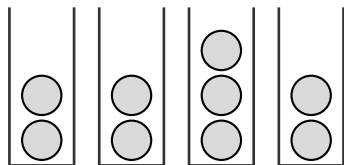


Computing  $A_{S \cup \{x\}}$

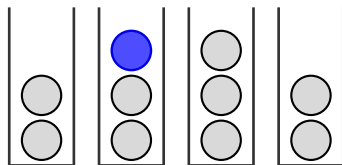
$$\Pr[\text{ball } x_i \text{ incurs recourse}] = O(1/n)$$

# ANALYZING THE RECOURSE

Recourse = 2



Computing  $A_S$



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

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

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



# A SIMPLE WARMUP

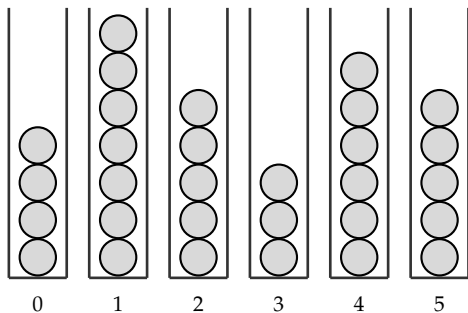
**Theorem:** There exists a history-independent solution with:

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

# SLICE AND SPREAD

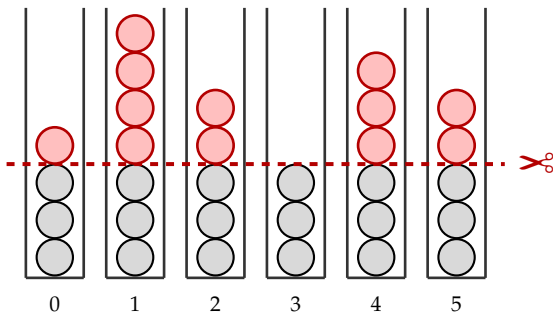
1. **Slice** off the jagged surface
2. **Spread** balls to their other bin choices

## SLICE AND SPREAD (TEST ANIMATION)



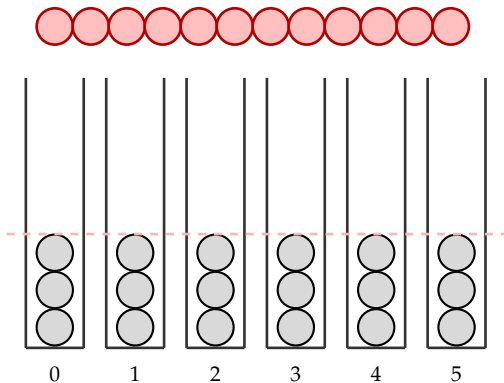
1. **Slice** off the jagged surface

## SLICE AND SPREAD (TEST ANIMATION)



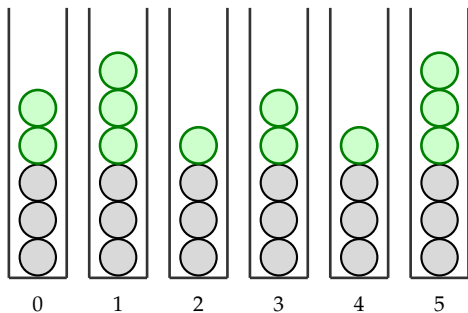
1. **Slice** off the jagged surface

## SLICE AND SPREAD (TEST ANIMATION)



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

## SLICE AND SPREAD (TEST ANIMATION)



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

# SLICE AND SPREAD REDUCES OVERLOAD

**Claim:** Slice and spread reduces overload.

**Proof (Sketch):**

1. **Fact:** After throwing  $m \gg n$  balls into  $n$  bins, every bin has load  $m/n \pm \tilde{O}(\sqrt{m/n})$ .
2. **Before:** Every bin has load  $m/n \pm \tilde{O}(\sqrt{m/n})$ .
3. **After:** Every bin has load  $m/n \pm \tilde{O}(m/n)^{1/4}$ .

## 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.

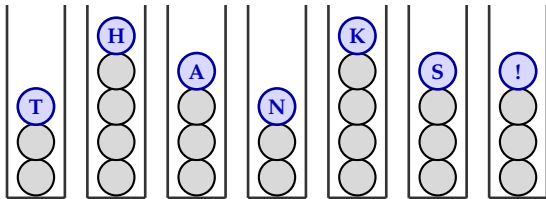
**Result:** We won't be able to slice off enough balls, and so the jaggedness will remain.

## 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