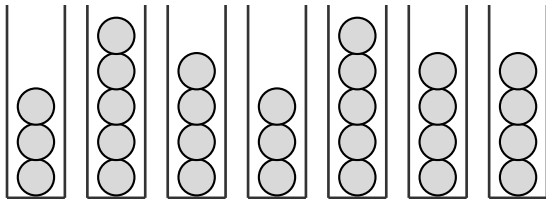


# History-Independent Load Balancing



Michael A. Bender

Stony Brook University

Bill Kuszmaul

CMU

Elaine Shi

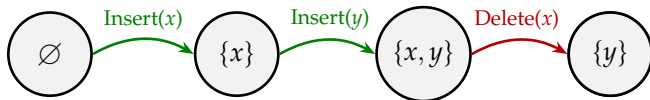
CMU

**Rose Silver**

CMU

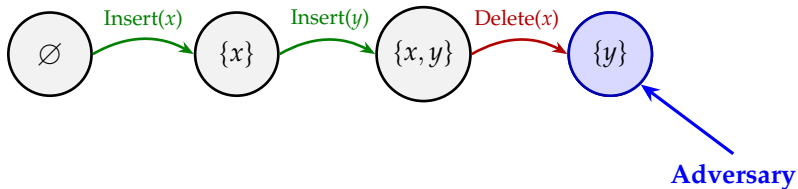
# HISTORY-INDEPENDENT DATA STRUCTURES

History 1:

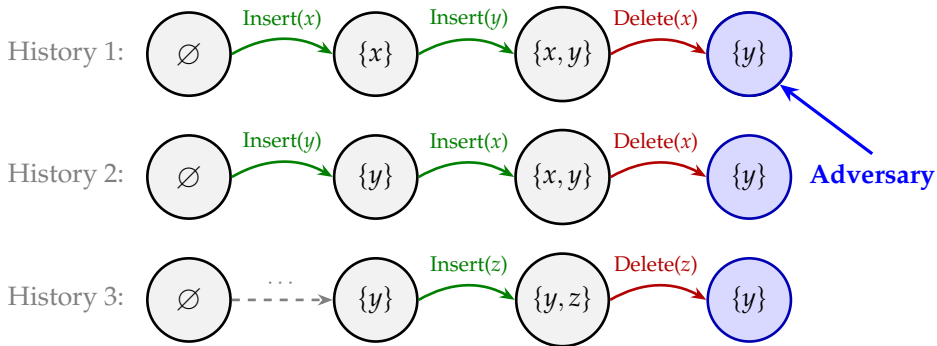


# HISTORY-INDEPENDENT DATA STRUCTURES

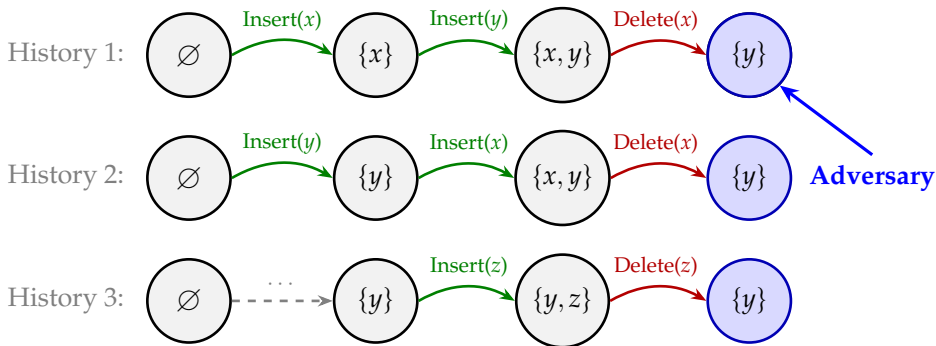
History 1:



# 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

## 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, Belloch & 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 fundamental 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.

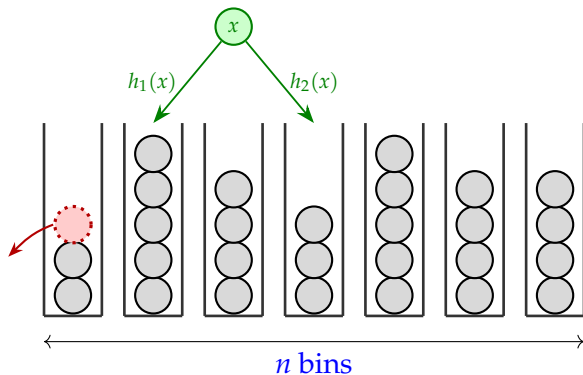
Micciancio '97, Naor & Teague '01, Buchbinder & Petrank '03, Molnar et al. '06, Belloch & Golovin '07, Moran et al. '07, Naor et al. '08, Golovin '08–'10, Bajaj & Sion '13, Roche et al. '15, Bender et al. '16

**Yet fundamental questions remain open.**

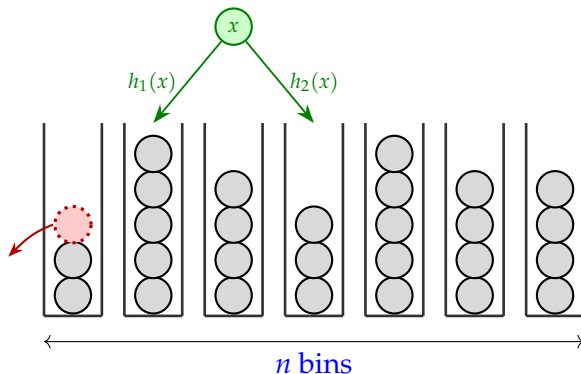
**This work:** History-Independent Load Balancing



# TWO-CHOICE LOAD BALANCING

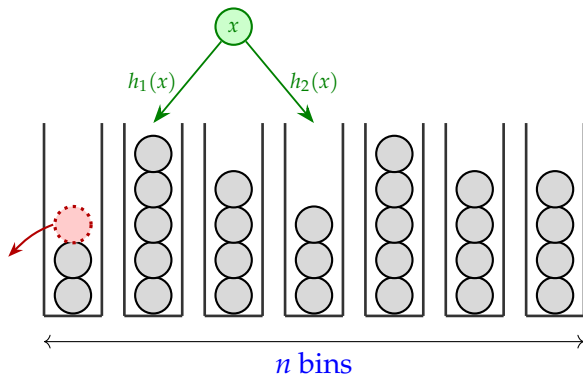


# TWO-CHOICE LOAD BALANCING



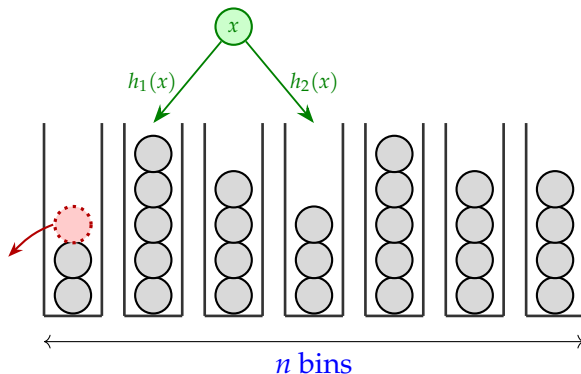
- Balls are **inserted**/**deleted**, with up to  $m$  present at a time.

# TWO-CHOICE LOAD BALANCING



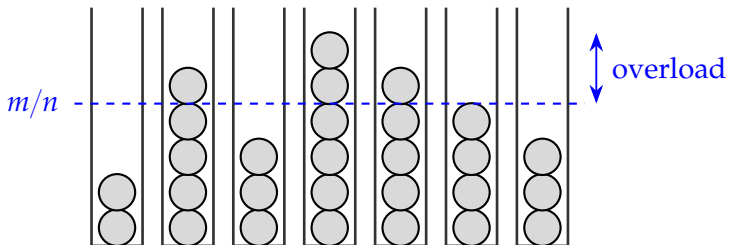
- ▶ Balls are **inserted**/**deleted**, with up to  $m$  present at a time.
- ▶ Each ball has two random bins where it can go.

# 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.

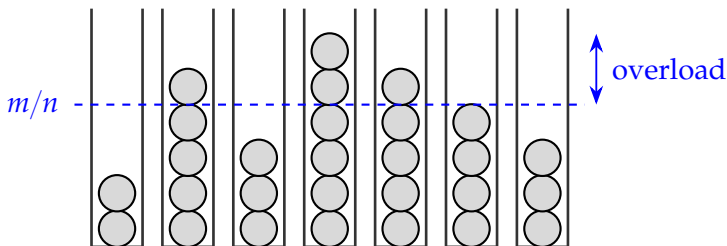
## TWO GOALS



### 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!

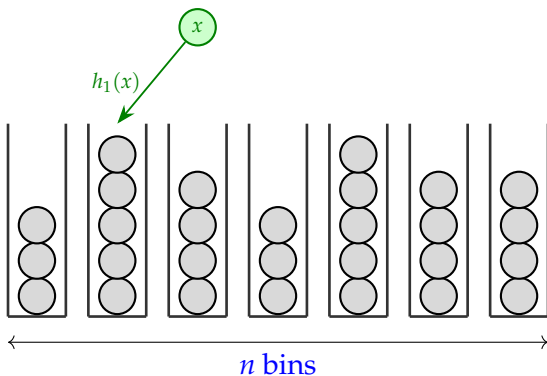
## REST OF TALK: A SIMPLE WARMUP

**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

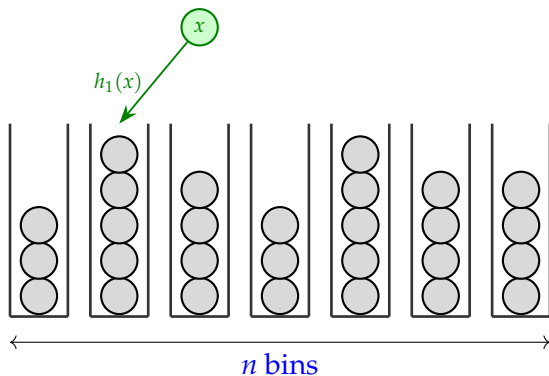
To insert a ball  $x$ , just put it in bin  $h_1(x)$ :





## BASLINE 1: THE SINGLE-CHOICE STRATEGY

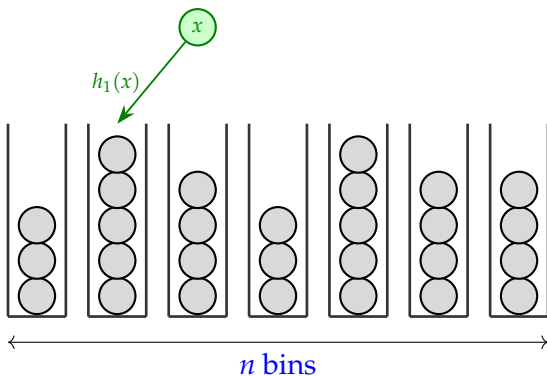
To insert a ball  $x$ , just put it in bin  $h_1(x)$ :



- This is history-independent ✓

## BASLINE 1: THE SINGLE-CHOICE STRATEGY

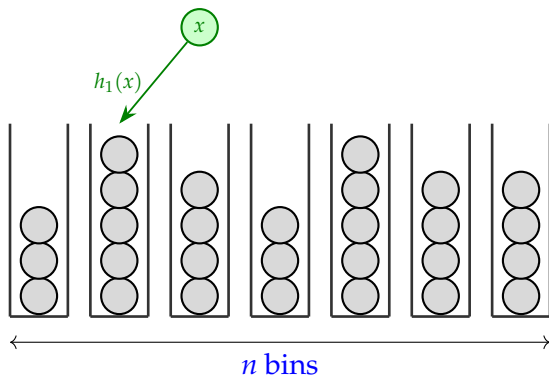
To insert a ball  $x$ , just put it in bin  $h_1(x)$ :



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

# BASLINE 1: THE SINGLE-CHOICE STRATEGY

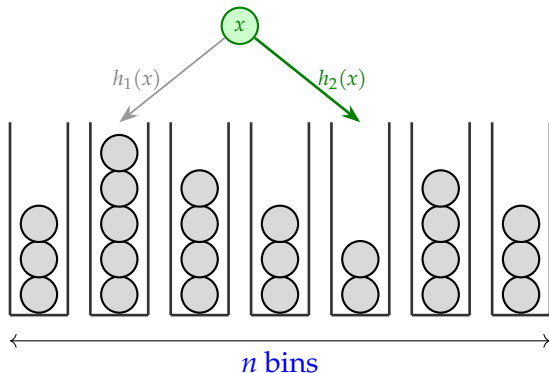
To insert a ball  $x$ , just put it in bin  $h_1(x)$ :



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

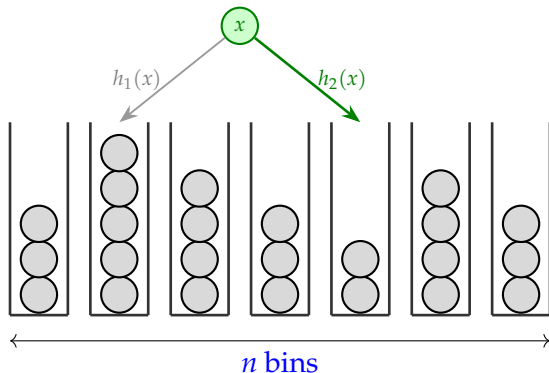
## BASELINE 2: GREEDY INSERTIONS

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



## BASELINE 2: GREEDY INSERTIONS

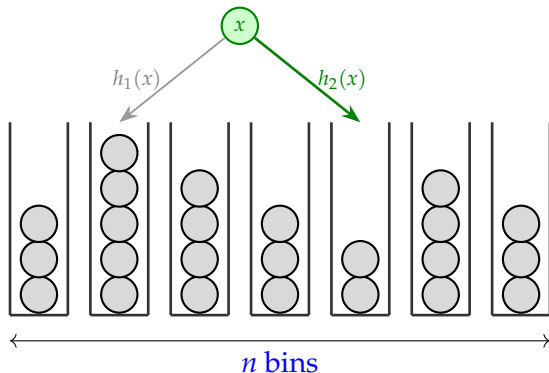
To insert a ball  $x$ , put it in the **emptier** of its choices:



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

## BASELINE 2: GREEDY INSERTIONS

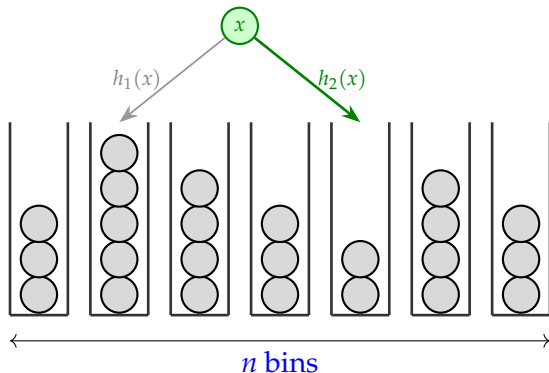
To insert a ball  $x$ , put it in the **emptier** of its choices:



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

## BASLINE 2: GREEDY INSERTIONS

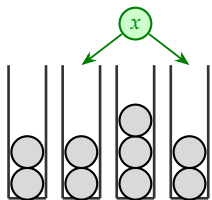
To insert a ball  $x$ , put it in the **emptier** of its choices:



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

[Azar, Broder, Karlin and Upfal '94]

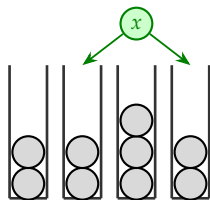
# A SIMPLE HISTORY-INDEPENDENT ALGORITHM



Current state  $A_5$



# A SIMPLE HISTORY-INDEPENDENT ALGORITHM

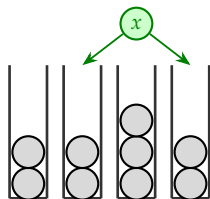


Current state  $A_S$

At each time step:

1. Observe current set  $S$
2. Compute  $A_{S \cup \{x\}}$
3. Update the system to reflect  $A_{S \cup \{x\}}$

# A SIMPLE HISTORY-INDEPENDENT ALGORITHM



Current state  $A_S$

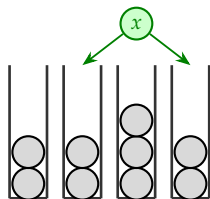
At each time step:

1. Observe current set  $S$
2. Compute  $A_{S \cup \{x\}}$
3. Update the system to reflect  $A_{S \cup \{x\}}$

**Compute  $A_{S \cup \{x\}}$  by simulating greedy:**

- 2a. Start with empty bins
- 2b. Sort balls in  $S$  to get  $x_1, x_2, \dots$
- 2c. Insert balls using greedy

# A SIMPLE HISTORY-INDEPENDENT ALGORITHM



Current state  $A_S$



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

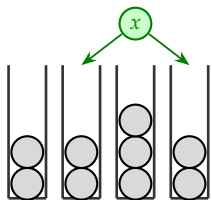
At each time step:

1. Observe current set  $S$
2. Compute  $A_{S \cup \{x\}}$
3. Update the system to reflect  $A_{S \cup \{x\}}$

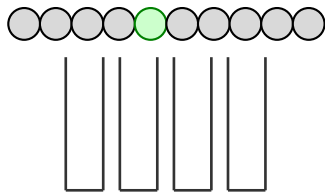
**Compute  $A_{S \cup \{x\}}$  by simulating greedy:**

- 2a. Start with empty bins
- 2b. Sort balls in  $S$  to get  $x_1, x_2, \dots$
- 2c. Insert balls using greedy

# A SIMPLE HISTORY-INDEPENDENT ALGORITHM



Current state  $A_S$



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

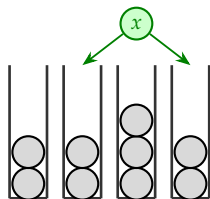
At each time step:

1. Observe current set  $S$
2. Compute  $A_{S \cup \{x\}}$
3. Update the system to reflect  $A_{S \cup \{x\}}$

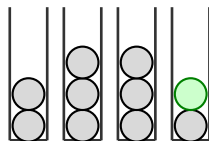
**Compute  $A_{S \cup \{x\}}$  by simulating greedy:**

- 2a. Start with empty bins
- 2b. Sort balls in  $S$  to get  $x_1, x_2, \dots$
- 2c. Insert balls using greedy

# A SIMPLE HISTORY-INDEPENDENT ALGORITHM



Current state  $A_S$



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

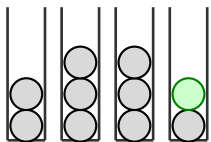
At each time step:

1. Observe current set  $S$
2. Compute  $A_{S \cup \{x\}}$
3. Update the system to reflect  $A_{S \cup \{x\}}$

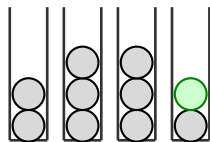
**Compute  $A_{S \cup \{x\}}$  by simulating greedy:**

- 2a. Start with empty bins
- 2b. Sort balls in  $S$  to get  $x_1, x_2, \dots$
- 2c. Insert balls using greedy

# A SIMPLE HISTORY-INDEPENDENT ALGORITHM



Updated state  $A_{S \cup \{x\}}$



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

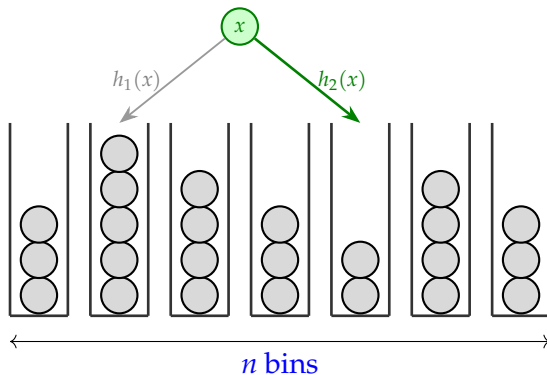
At each time step:

1. Observe current set  $S$
2. Compute  $A_{S \cup \{x\}}$
3. Update the system to reflect  $A_{S \cup \{x\}}$

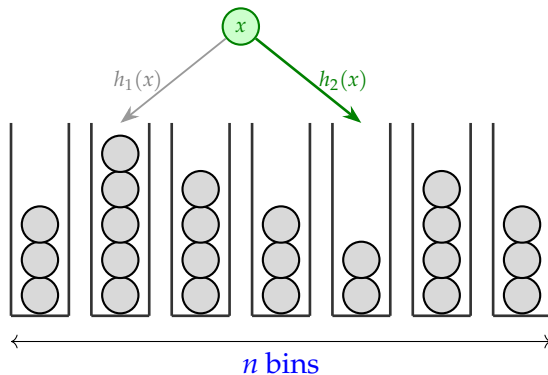
**Compute  $A_{S \cup \{x\}}$  by simulating greedy:**

- 2a. Start with empty bins
- 2b. Sort balls in  $S$  to get  $x_1, x_2, \dots$
- 2c. Insert balls using greedy

# ANALYZING HISTORY-INDEPENDENT GREEDY



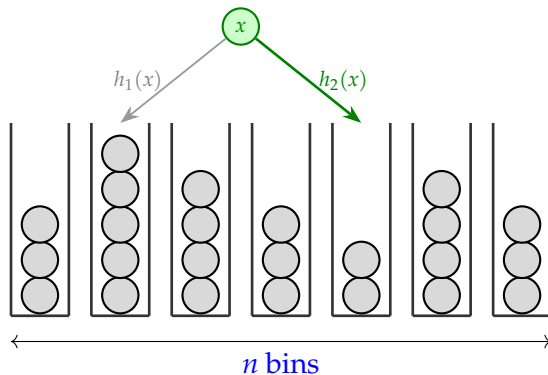
# ANALYZING HISTORY-INDEPENDENT GREEDY



- The algorithm is history independent ✓

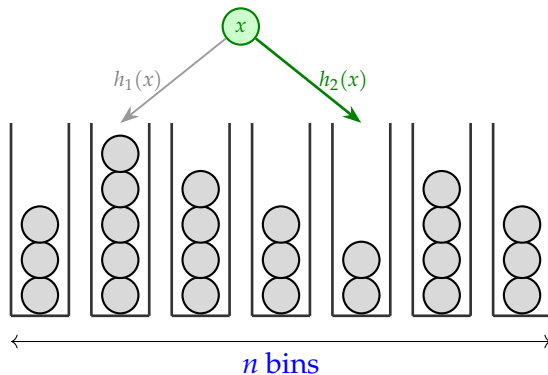


# ANALYZING HISTORY-INDEPENDENT GREEDY



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

# ANALYZING HISTORY-INDEPENDENT GREEDY

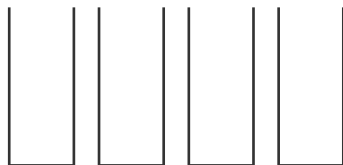


- ▶ The algorithm is history independent ✓
- ▶ The overload is  $O(\log \log n)$  ✓
- ▶ What is the recourse?

# ANALYZING THE RECOURSE



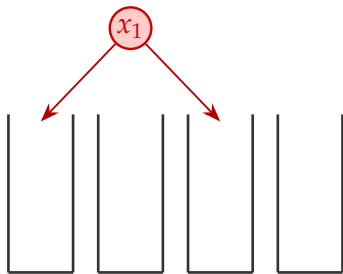
**Computing  $\text{Greedy}(S)$**



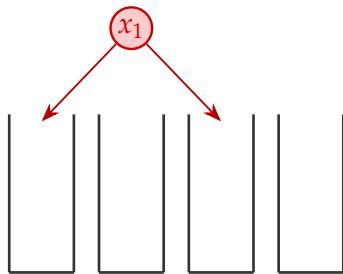
**Computing  $\text{Greedy}(S \cup \{x^*\})$**

How does  $\text{Greedy}(S)$  change if we add a ball  $x^*$ ?

# ANALYZING THE RECOURSE



**Computing Greedy( $S$ )**



**Computing Greedy( $S \cup \{x^*\}$ )**

# ANALYZING THE RECOURSE

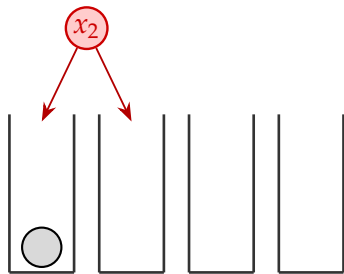


**Computing  $\text{Greedy}(S)$**

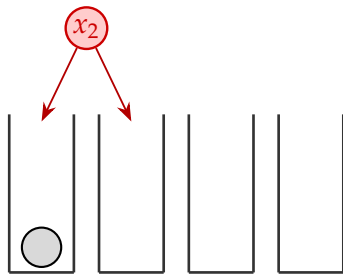


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

# ANALYZING THE RECOURSE

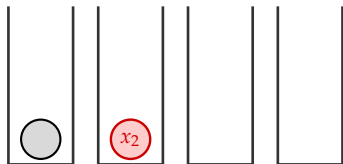


**Computing  $\text{Greedy}(S)$**

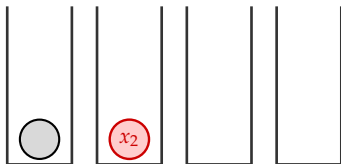


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

# ANALYZING THE RECOURSE

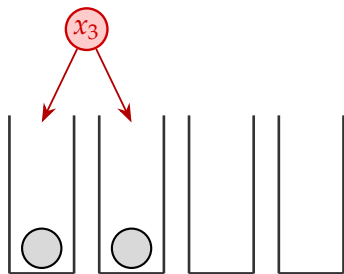


**Computing  $\text{Greedy}(S)$**

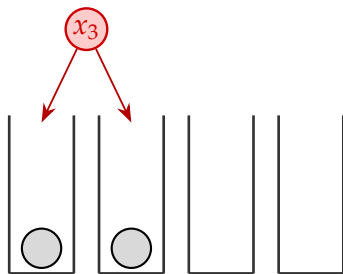


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

# ANALYZING THE RECOURSE



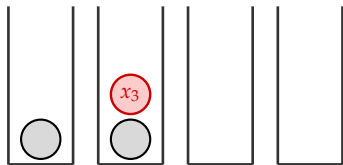
**Computing  $\text{Greedy}(S)$**



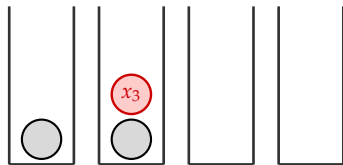
**Computing  $\text{Greedy}(S \cup \{x^*\})$**



# ANALYZING THE RECOURSE

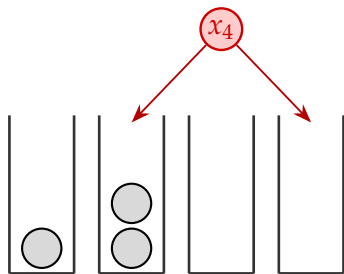


**Computing  $\text{Greedy}(S)$**

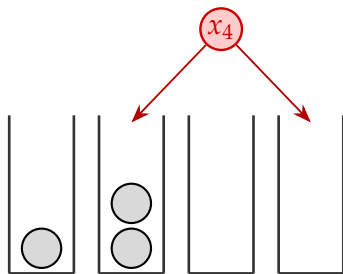


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

# ANALYZING THE RECOURSE

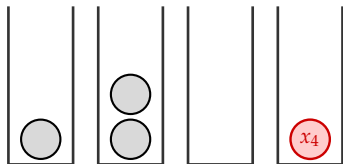


**Computing  $\text{Greedy}(S)$**

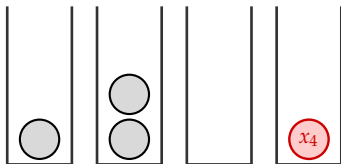


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

# ANALYZING THE RECOURSE

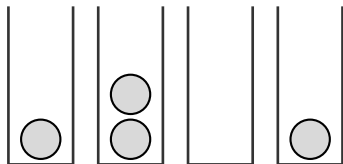


**Computing Greedy( $S$ )**

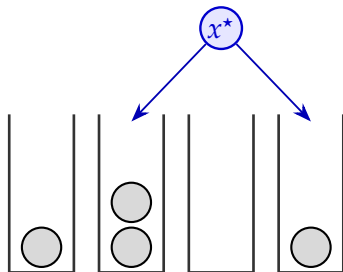


**Computing Greedy( $S \cup \{x^*\}$ )**

# ANALYZING THE RECOURSE

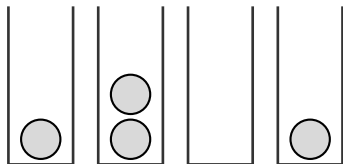


**Computing  $\text{Greedy}(S)$**

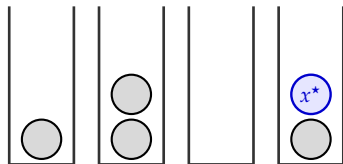


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

# ANALYZING THE RECOURSE

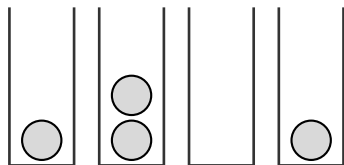


**Computing  $\text{Greedy}(S)$**

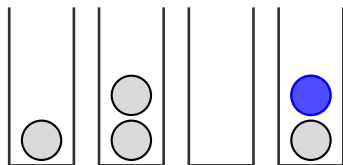


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

# ANALYZING THE RECOURSE



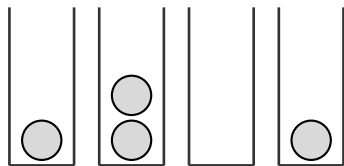
**Computing Greedy( $S$ )**



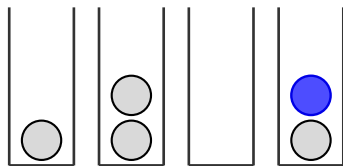
**Computing Greedy( $S \cup \{x^*\}$ )**

Subsequent balls will experience either:

# ANALYZING THE RECOURSE



**Computing  $\text{Greedy}(S)$**

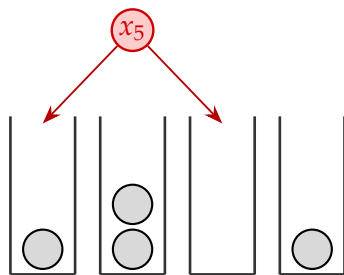


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

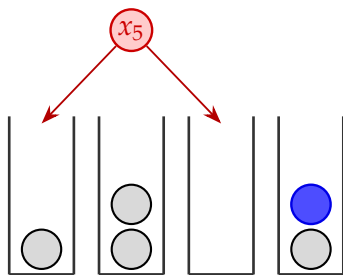
Subsequent balls will experience either:

1. No recourse

# ANALYZING THE RECOURSE



**Computing Greedy( $S$ )**



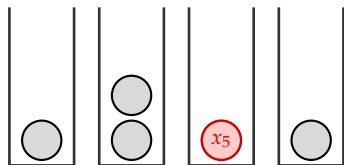
**Computing Greedy( $S \cup \{x^*\}$ )**

Future insertions will experience either:

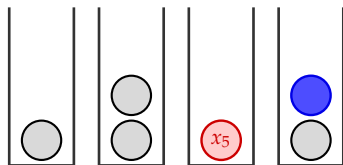
1. No recourse



# ANALYZING THE RECOURSE



**Computing Greedy( $S$ )**

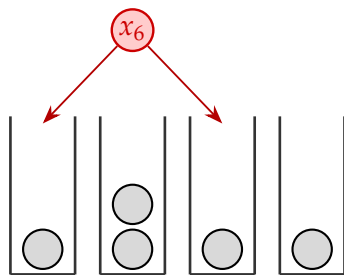


**Computing Greedy( $S \cup \{x^*\}$ )**

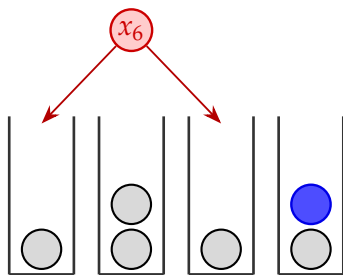
Subsequent balls will experience either:

1. No recourse

# ANALYZING THE RECOURSE



**Computing Greedy( $S$ )**

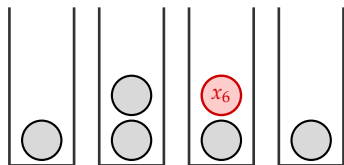


**Computing Greedy( $S \cup \{x^*\}$ )**

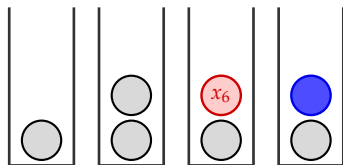
Subsequent balls will experience either:

1. No recourse

# ANALYZING THE RECOURSE



**Computing  $\text{Greedy}(S)$**

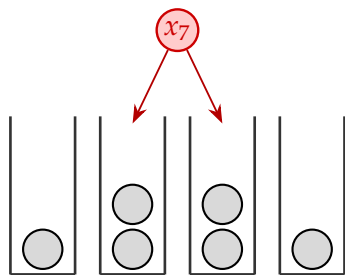


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

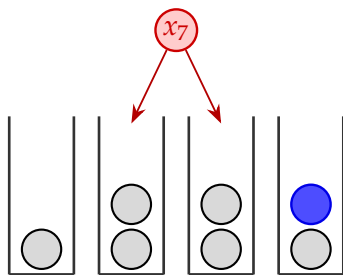
Subsequent balls will experience either:

1. No recourse

# ANALYZING THE RECOURSE



**Computing Greedy( $S$ )**

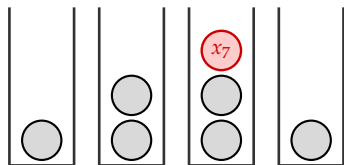


**Computing Greedy( $S \cup \{x^*\}$ )**

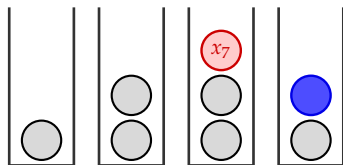
Subsequent balls will experience either:

1. No recourse

# ANALYZING THE RECOURSE



**Computing Greedy( $S$ )**

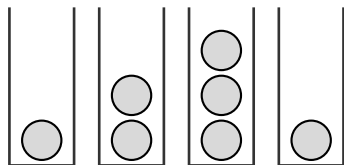


**Computing Greedy( $S \cup \{x^*\}$ )**

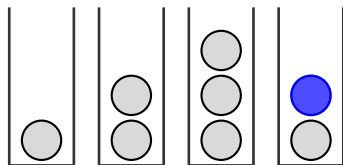
Subsequent balls will experience either:

1. No recourse

# ANALYZING THE RECOURSE



**Computing Greedy( $S$ )**

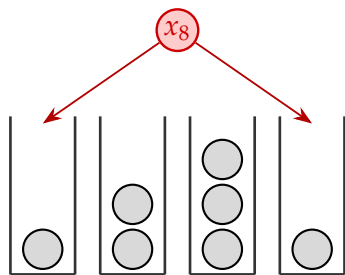


**Computing Greedy( $S \cup \{x^*\}$ )**

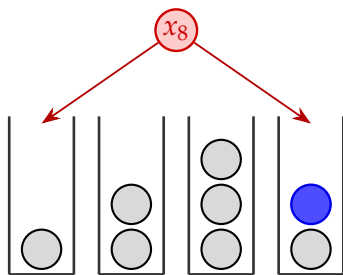
Subsequent balls will experience either:

1. No recourse
2. Recourse

# ANALYZING THE RECOURSE



**Computing  $\text{Greedy}(S)$**

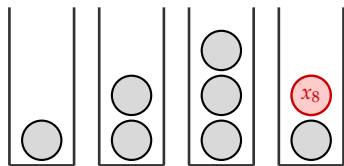


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

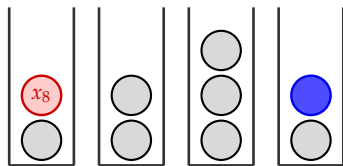
Subsequent balls will experience either:

1. No recourse
2. Recourse

# ANALYZING THE RECOURSE



**Computing  $\text{Greedy}(S)$**



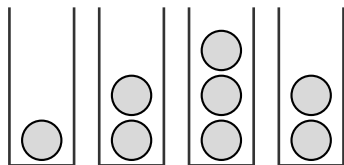
**Computing  $\text{Greedy}(S \cup \{x^*\})$**

Subsequent balls will experience either:

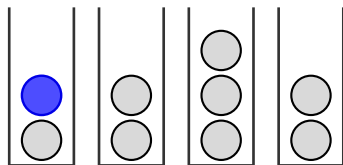
1. No recourse
2. Recourse



# ANALYZING THE RECOURSE



**Computing Greedy( $S$ )**

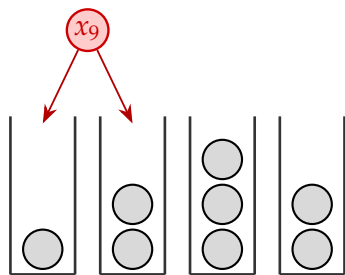


**Computing Greedy( $S \cup \{x^*\}$ )**

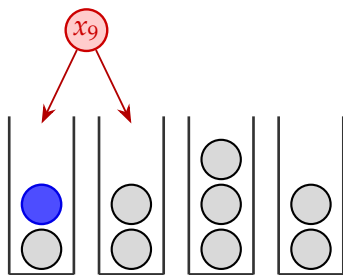
Subsequent balls will experience either:

1. No recourse
2. Recourse

# ANALYZING THE RECOURSE



**Computing Greedy( $S$ )**

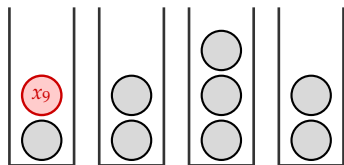


**Computing Greedy( $S \cup \{x^*\}$ )**

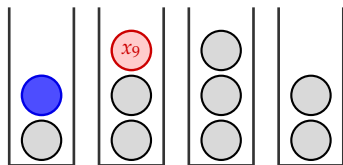
Subsequent balls will experience either:

1. No recourse
2. Recourse

# ANALYZING THE RECOURSE



**Computing  $\text{Greedy}(S)$**

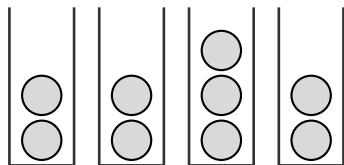


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

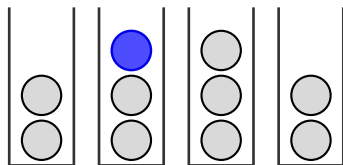
Subsequent balls will experience either:

1. No recourse
2. Recourse

# ANALYZING THE RECOURSE



**Computing Greedy( $S$ )**

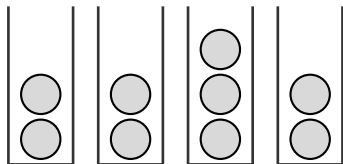


**Computing Greedy( $S \cup \{x^*\}$ )**

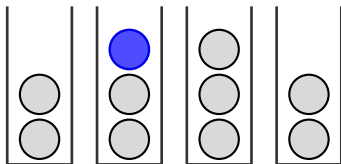
Subsequent balls will experience either:

1. No recourse
2. Recourse

# ANALYZING THE RECOURSE



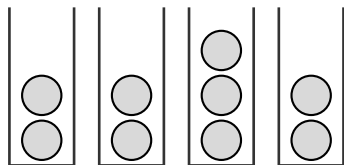
**Computing  $\text{Greedy}(S)$**



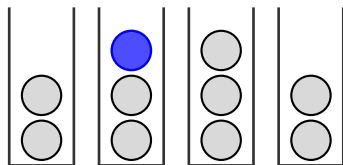
**Computing  $\text{Greedy}(S \cup \{x^*\})$**

Two key observations:

# ANALYZING THE RECOURSE



**Computing Greedy( $S$ )**

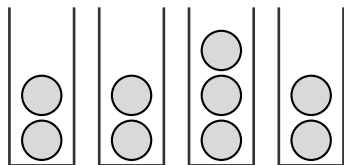


**Computing Greedy( $S \cup \{x^*\}$ )**

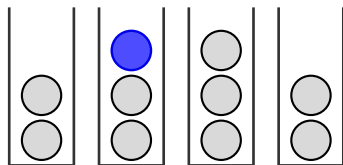
Two key observations:

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

# ANALYZING THE RECOURSE



**Computing  $\text{Greedy}(S)$**

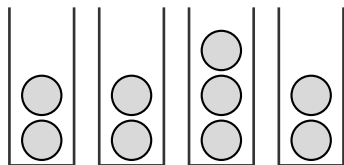


**Computing  $\text{Greedy}(S \cup \{x^*\})$**

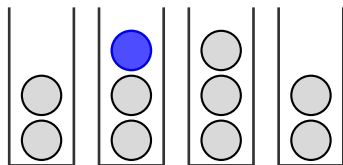
Two key observations:

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



**Computing Greedy( $S$ )**

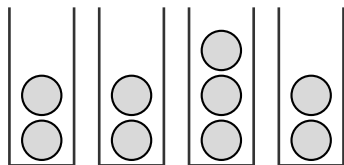


**Computing Greedy( $S \cup \{x^*\}$ )**

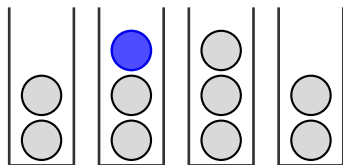
$$\Pr[\text{ball } x_i \text{ incurs recourse}] = O(1/n)$$



# ANALYZING THE RECOURSE



**Computing Greedy**( $S$ )



**Computing Greedy**( $S \cup \{x^*\}$ )

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

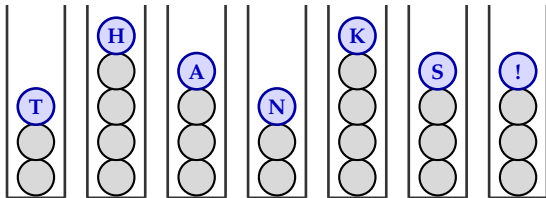
$$\implies \mathbb{E}[\text{total recourse}] = \sum_i \Pr[\text{ball } x_i \text{ incurs recourse}] = O(m/n)$$

# A SIMPLE WARMUP

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

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

# History-Independent Load Balancing



Michael A. Bender

Stony Brook University

Bill Kuszmaul

CMU

Elaine Shi

CMU

**Rose Silver**

CMU