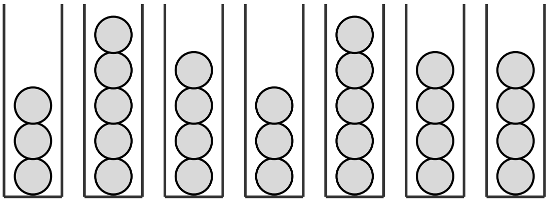
History-Independent Load Balancing



Michael A. Bender — Stony Brook University

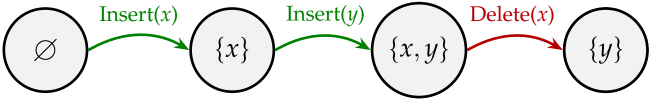
Bill Kuszmaul — CMU

Elaine Shi — CMU

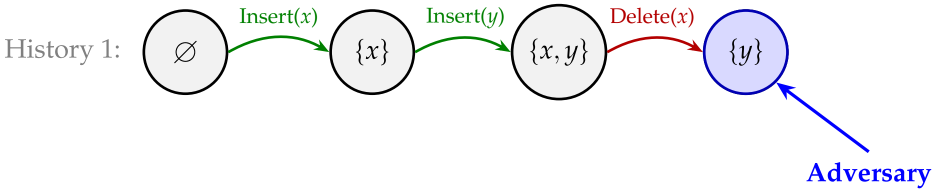
Rose Silver — CMU

History-Independent Data Structures

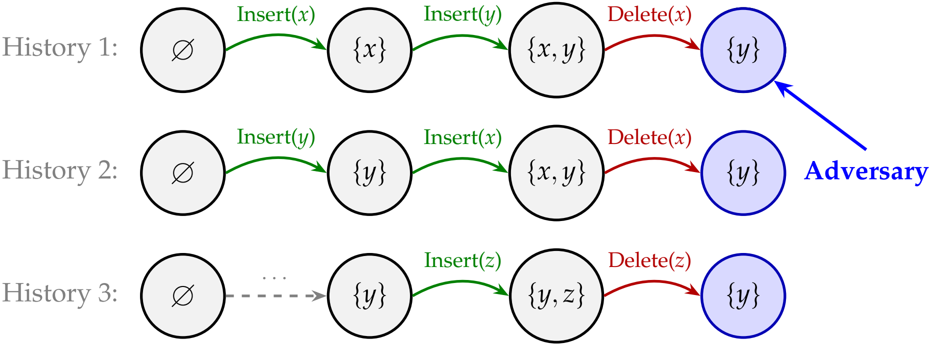
History 1:



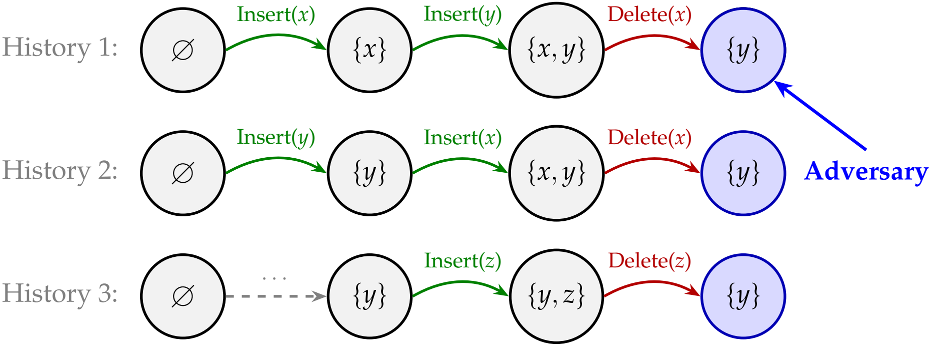
History-Independent Data Structures



History-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, 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 fundamental questions remain open.

History Independent Data Structures

A History of Applications

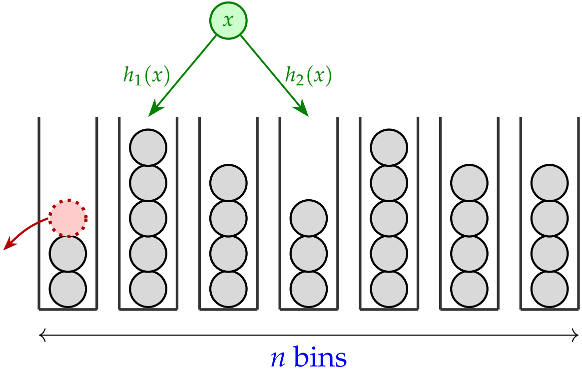
Hash tables, trees, memory allocation, PMAs, graph algorithms, cache-oblivious data structures, and more.

Micciancio '97, Naor & Teague '01, Buchbinder & Petrank '03, Molnar et al. '06, 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

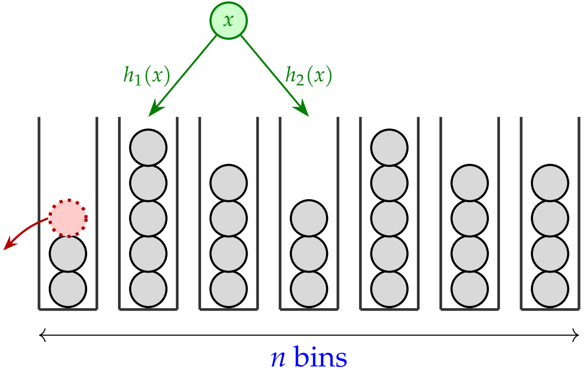
Yet fundamental questions remain open.

This work: History-Independent Load Balancing

Two-Choice Load Balancing

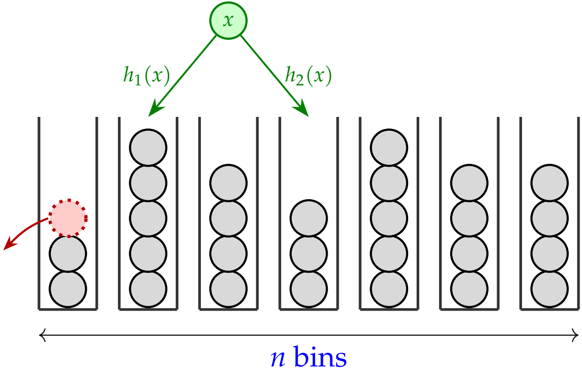


Two-Choice Load Balancing



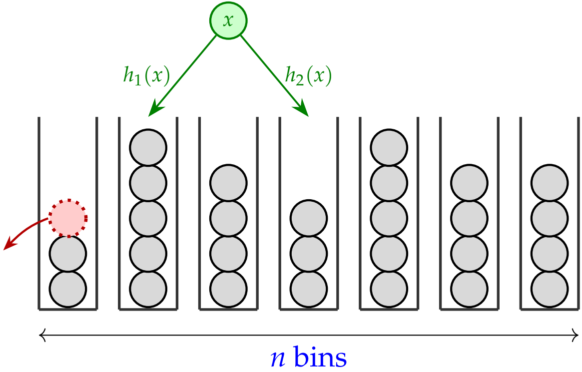
- Balls are inserted/deleted, with up to m present at a time.

Two-Choice Load Balancing



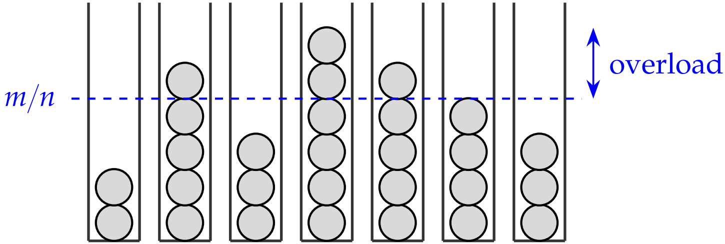
- Balls are inserted/deleted, with up to m present at a time.
- Each ball has two random bins where it can go.

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.

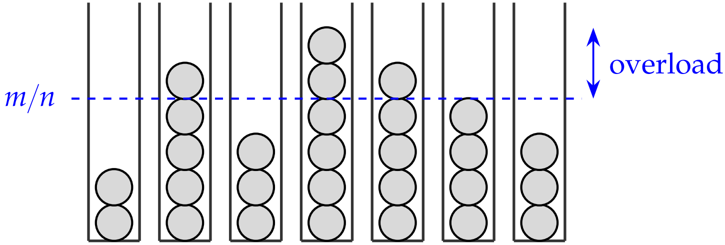
TWO GOALS



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!

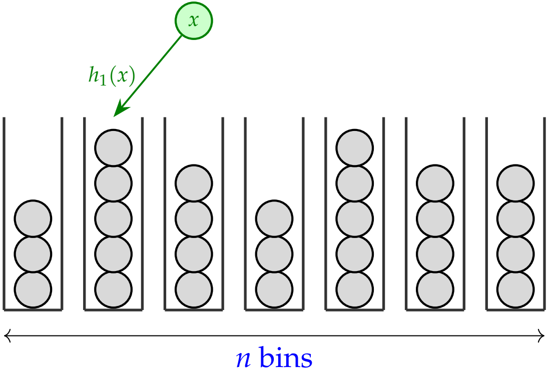
Rest of Talk: A Simple Warmup

Theorem: There exists a history-independent solution with:

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

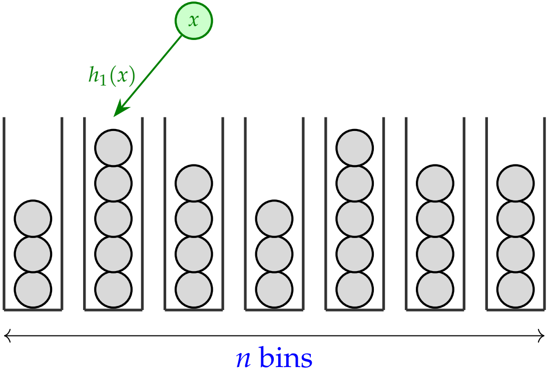
Baseline 1: The Single-Choice Strategy

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



Baseline 1: The Single-Choice Strategy

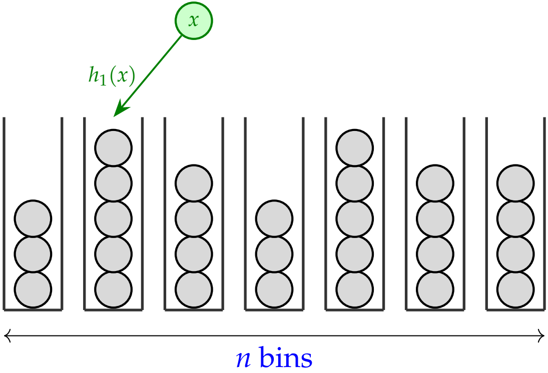
To insert a ball x, just put it in bin $h_1(x)$:



- This is history-independent ✓

Baseline 1: The Single-Choice Strategy

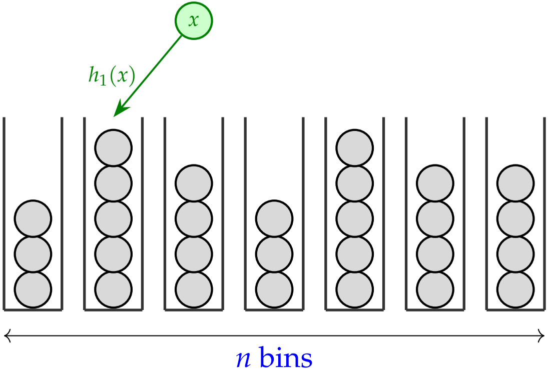
To insert a ball x, just put it in bin $h_1(x)$:



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

Baseline 1: The Single-Choice Strategy

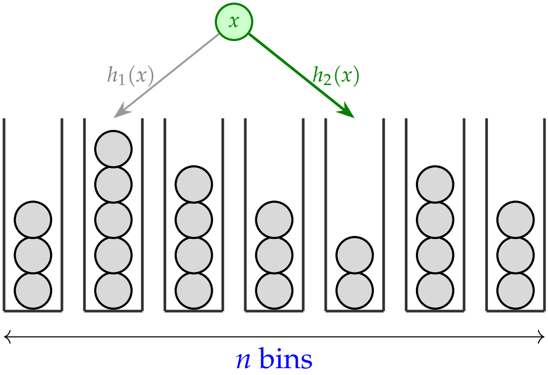
To insert a ball x, just put it in bin $h_1(x)$:



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

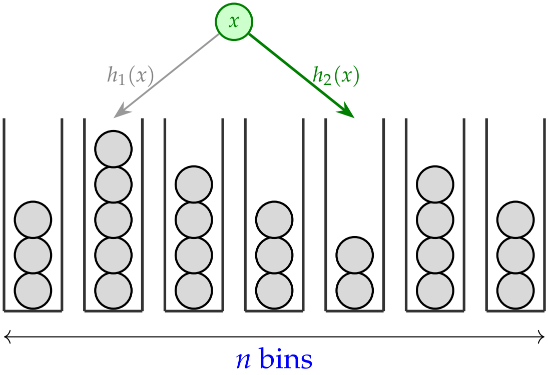
Baseline 2: Greedy Insertions

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



Baseline 2: Greedy Insertions

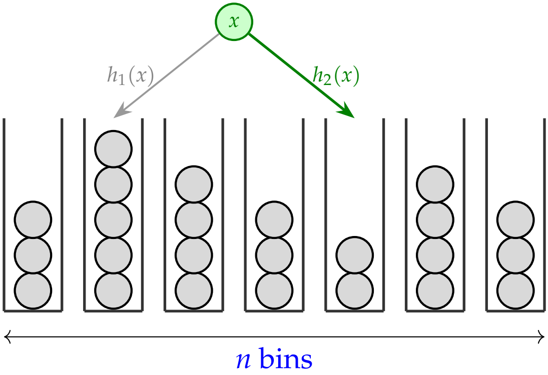
To insert a ball x, put it in the emptier of its choices:



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

Baseline 2: Greedy Insertions

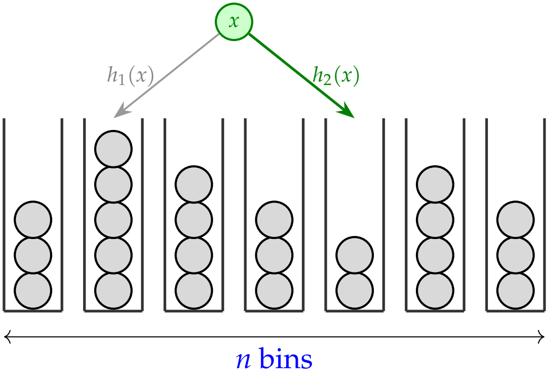
To insert a ball x, put it in the emptier of its choices:



- 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:

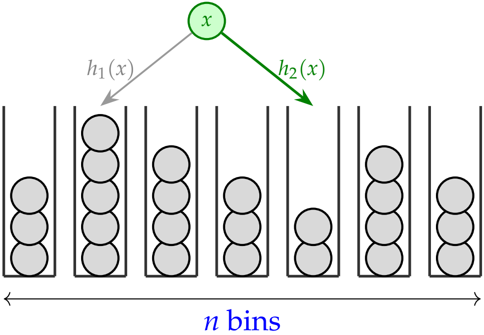


- This is **not** history-independent ✗
- The recourse is 0 ✓
- In the insertion-only case, the overload is $O(\log \log n)$ ✓

[Azar, Broder, Karlin and Upfal '94]

A Simple History-Independent Algorithm

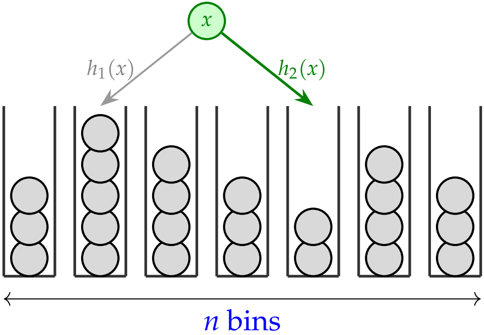
A Simple History-Independent Algorithm



Given a set S of balls, define `Greedy`(S) as:

- Start with empty bins.
- Sort the balls in S to get a sequence $x_1, x_2, \ldots$.
- Insert $x_1, x_2, \ldots$ using the greedy algorithm.

A Simple History-Independent Algorithm

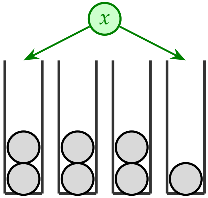


Given a set S of balls, define `Greedy`(S) as:

- Start with empty bins.
- Sort the balls in S to get a sequence $x_1, x_2, \ldots$.
- Insert $x_1, x_2, \ldots$ using the greedy algorithm.

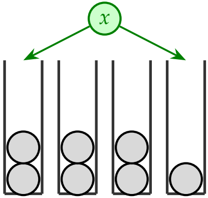
A History-Independent Algorithm: If S is the current set, use `Greedy`(S).

A Simple History-Independent Algorithm



Current state

A Simple History-Independent Algorithm

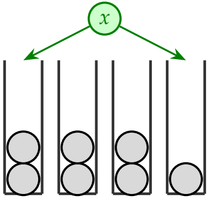


Current state

At each time step:

1. Observe current set S
2. Compute A_S
3. Update the system to reflect A_S

A Simple History-Independent Algorithm



Current state

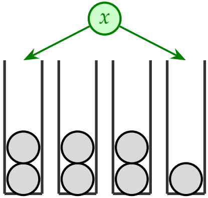
At each time step:

1. Observe current set S
2. Compute A_S
3. Update the system to reflect A_S

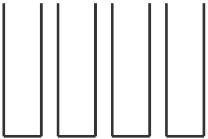
Compute A_S by simulating greedy:

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

A Simple History-Independent Algorithm



Current state



Computing A_S

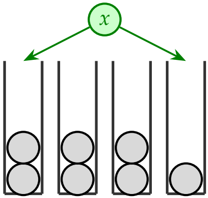
At each time step:

1. Observe current set S
2. Compute A_S
3. Update the system to reflect A_S

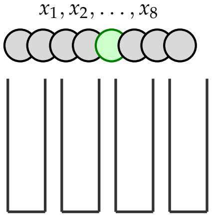
Compute A_S by simulating greedy:

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

A Simple History-Independent Algorithm



Current state



Computing A_S

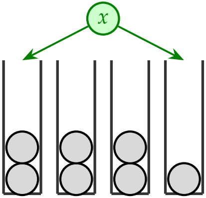
At each time step:

1. Observe current set S
2. Compute A_S
3. Update the system to reflect A_S

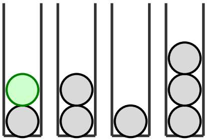
Compute A_S by simulating greedy:

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

A Simple History-Independent Algorithm



Current state



Computing A_S

At each time step:

1. Observe current set S
2. Compute A_S
3. Update the system to reflect A_S

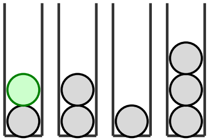
Compute A_S by simulating greedy:

2a. Start with empty bins

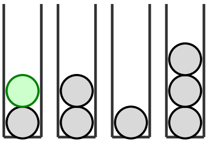
2b. Sort balls in S to get $x_1, x_2, \ldots$

2c. Insert balls using greedy

A Simple History-Independent Algorithm



Updated state



Computing A_S

At each time step:

1. Observe current set S
2. Compute A_S
3. Update the system to reflect A_S

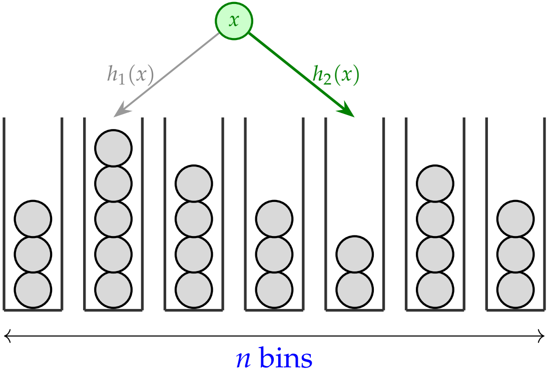
Compute A_S by simulating greedy:

2a. Start with empty bins

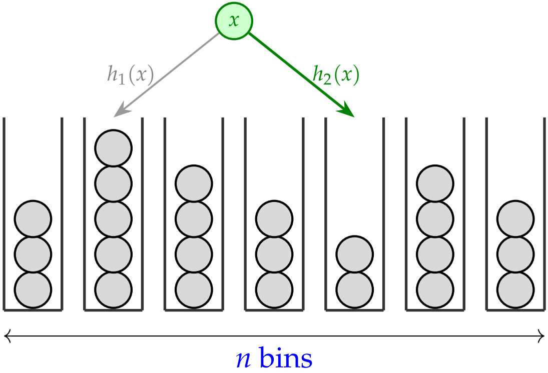
2b. Sort balls in S to get $x_1, x_2, \ldots$

2c. Insert balls using greedy

Analyzing History-Independent Greedy

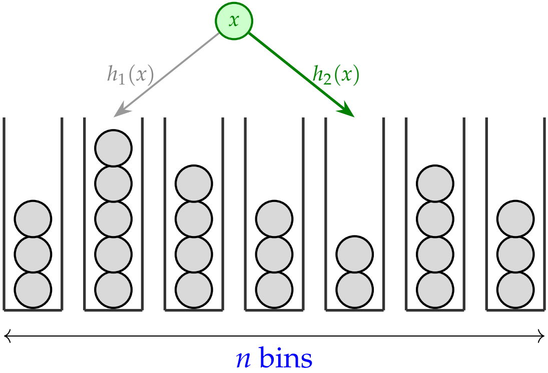


Analyzing History-Independent Greedy



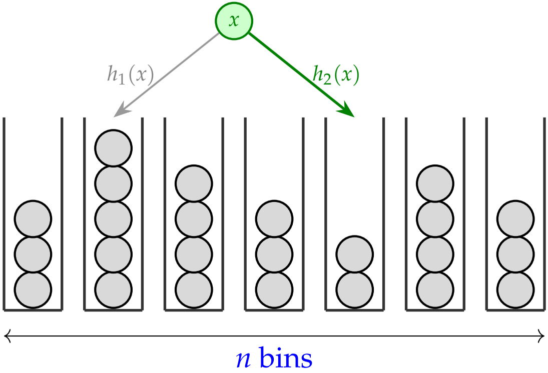
- The algorithm is history independent ✓

Analyzing History-Independent Greedy



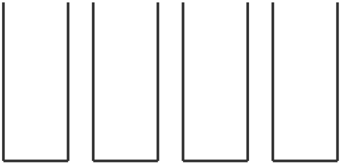
- The algorithm is history independent ✓
- The overload is $O(\log \log n)$ ✓

ANALYZING HISTORY-INDEPENDENT GREEDY

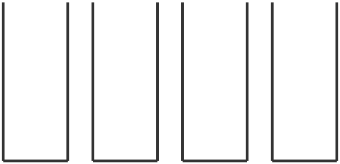


- The algorithm is history independent ✓
- The overload is $O(\log \log n)$ ✓
- What is the recourse?

ANALYZING THE RECOURSE



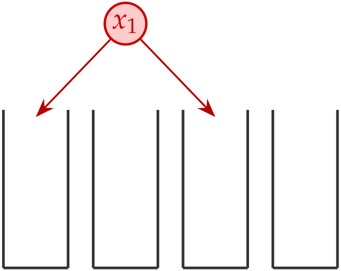
Computing `Greedy`(S)



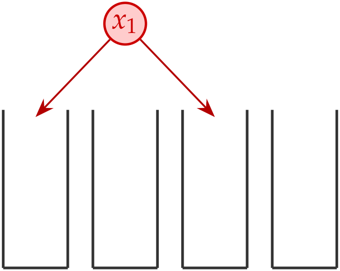
Computing `Greedy`$(S \cup \{x^*\})$

How does `Greedy`(S) change if we add a ball $x^\star$?

Analyzing the Recourse

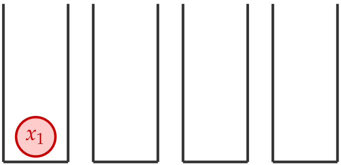


Computing `Greedy`(S)

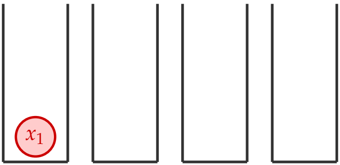


Computing `Greedy`$(S \cup \{x^*\})$

Analyzing the Recourse

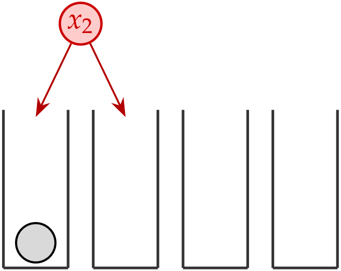


Computing `Greedy`(S)

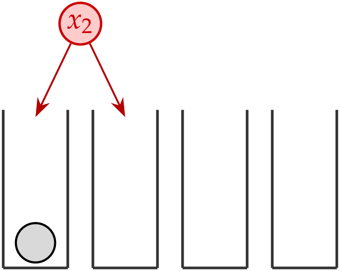


Computing `Greedy`$(S \cup \{x^*\})$

Analyzing the Recourse

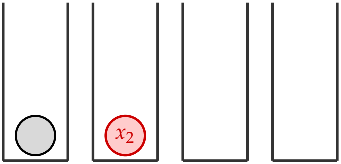


Computing $\texttt{Greedy}(S)$

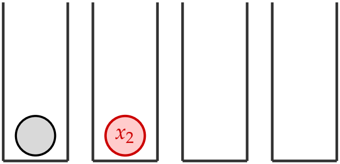


Computing $\texttt{Greedy}(S \cup \{x^*\})$

Analyzing the Recourse

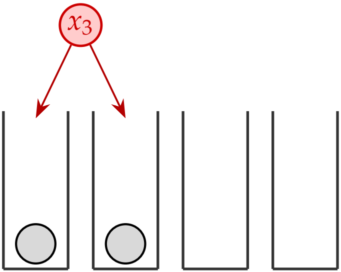


Computing **`Greedy`**(S)

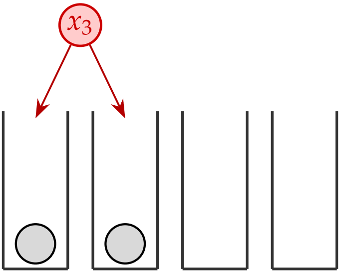


Computing **`Greedy`**$(S \cup \{x^*\})$

Analyzing the Recourse

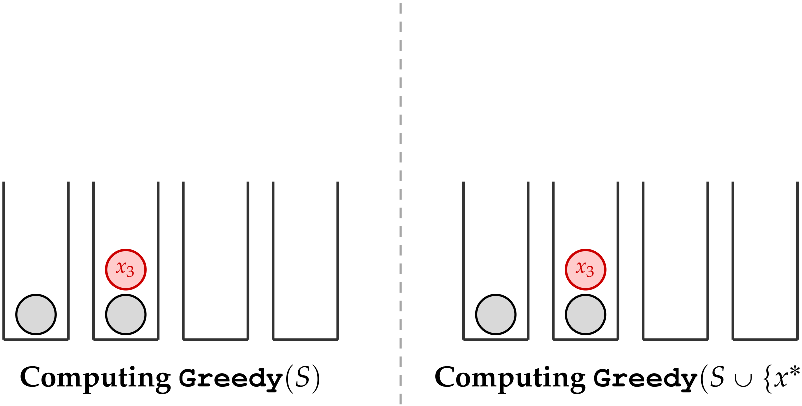


Computing `Greedy`(S)



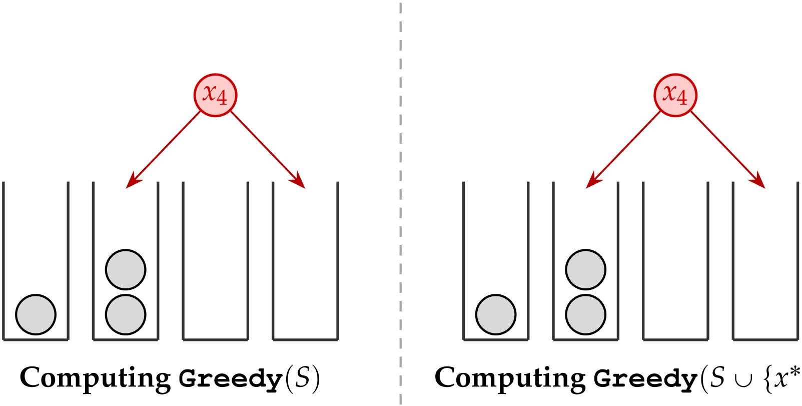
Computing `Greedy`$(S \cup \{x^*\})$

Analyzing the Recourse



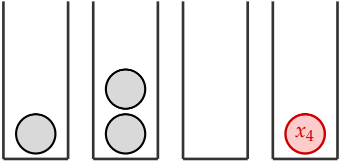
Computing $\texttt{\textbf{Greedy}}(S)$ **Computing** $\texttt{\textbf{Greedy}}(S \cup \{x^*\})$

Analyzing the Recourse

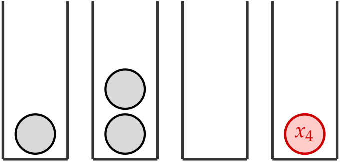


Computing `Greedy`(S) **Computing `Greedy`**$(S \cup \{x^*\})$

Analyzing the Recourse

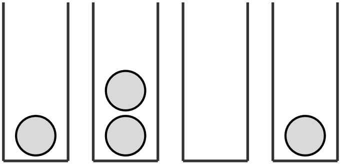


Computing $\texttt{Greedy}(S)$

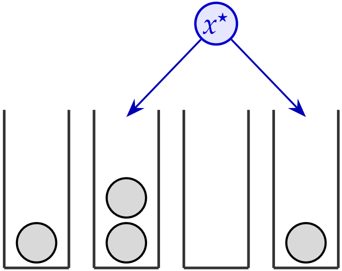


Computing $\texttt{Greedy}(S \cup \{x^*\})$

Analyzing the Recourse

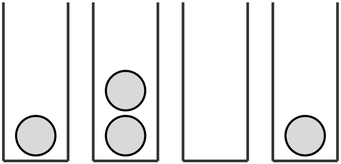


Computing `Greedy`(S)

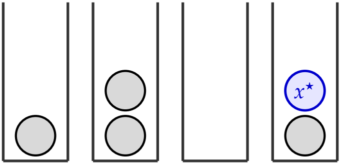


Computing `Greedy`$(S \cup \{x^*\})$

Analyzing the Recourse

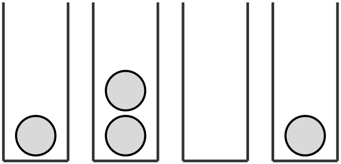


Computing `Greedy`(S)

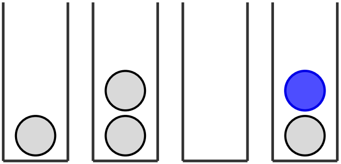


Computing `Greedy`$(S \cup \{x^*\})$

Analyzing the Recourse



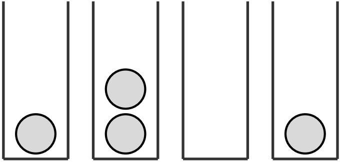
Computing $\texttt{\textbf{Greedy}}(S)$



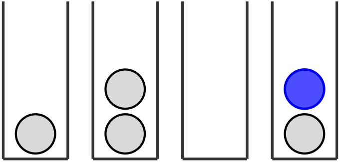
Computing $\texttt{\textbf{Greedy}}(S \cup \{x^*\})$

Subsequent balls will experience either:

Analyzing the Recourse



Computing `Greedy`(S)

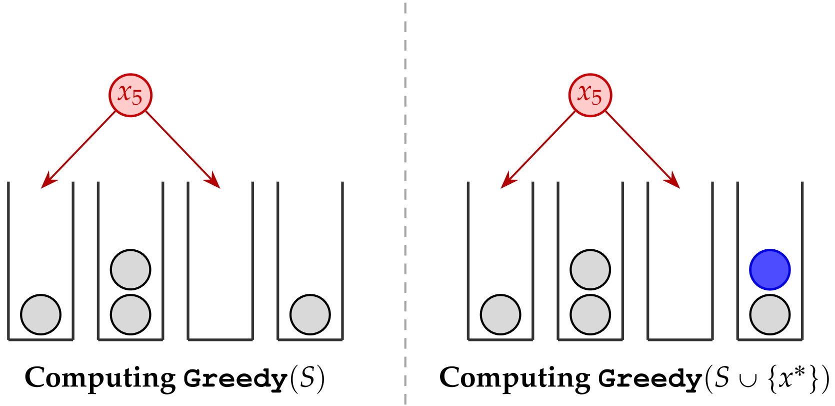


Computing `Greedy`$(S \cup \{x^*\})$

Subsequent balls will experience either:

1. No recourse

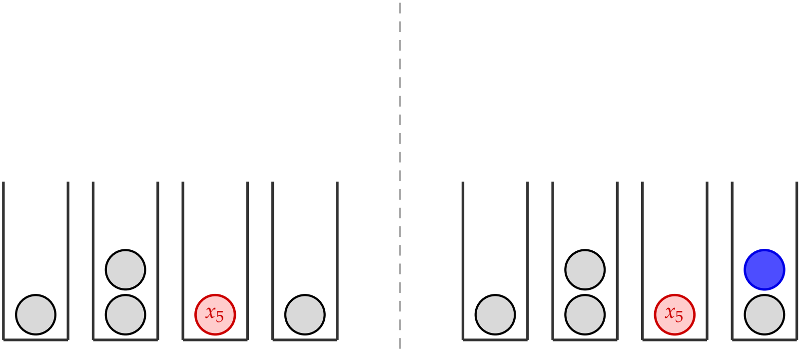
Analyzing the Recourse



Future insertions will experience either:

1. No recourse

Analyzing the Recourse

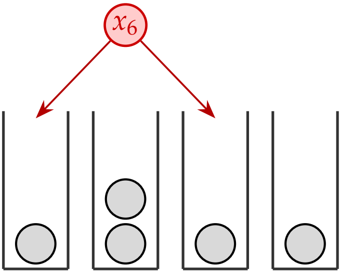


Computing **`Greedy`**(S) | **Computing** **`Greedy`**$(S \cup \{x^*\})$

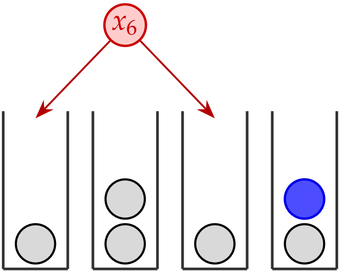
Subsequent balls will experience either:

1. No recourse

Analyzing the Recourse



Computing $\texttt{Greedy}(S)$

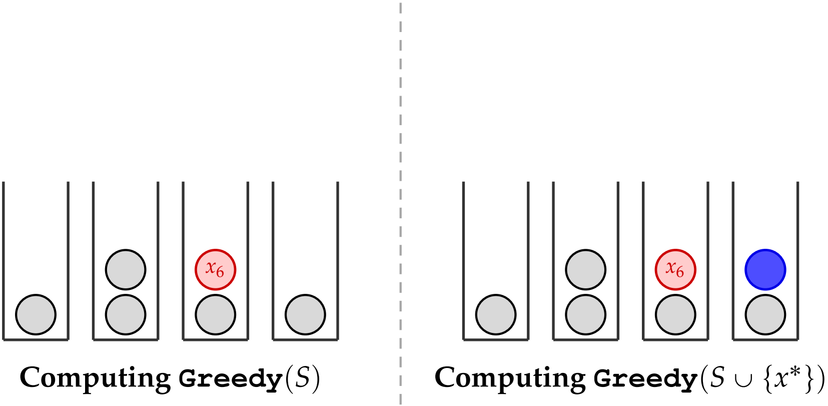


Computing $\texttt{Greedy}(S \cup \{x^*\})$

Subsequent balls will experience either:

1. No recourse

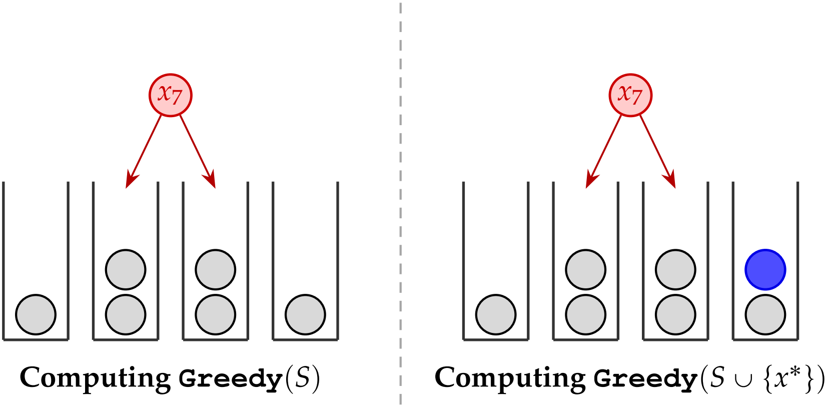
Analyzing the Recourse



Subsequent balls will experience either:

1. No recourse

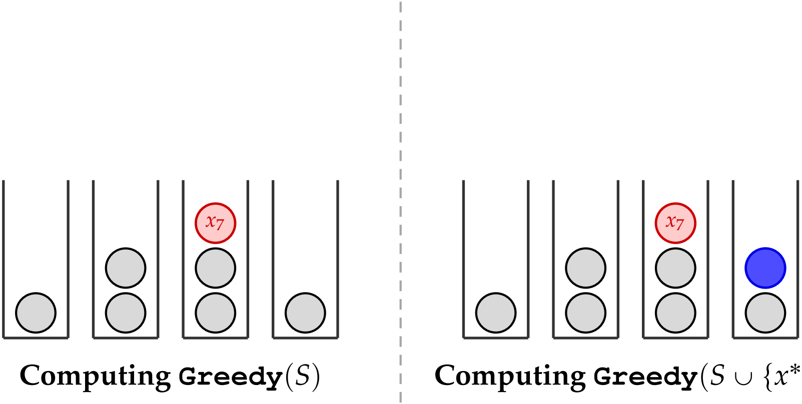
Analyzing the Recourse



Subsequent balls will experience either:

1. No recourse

Analyzing the Recourse

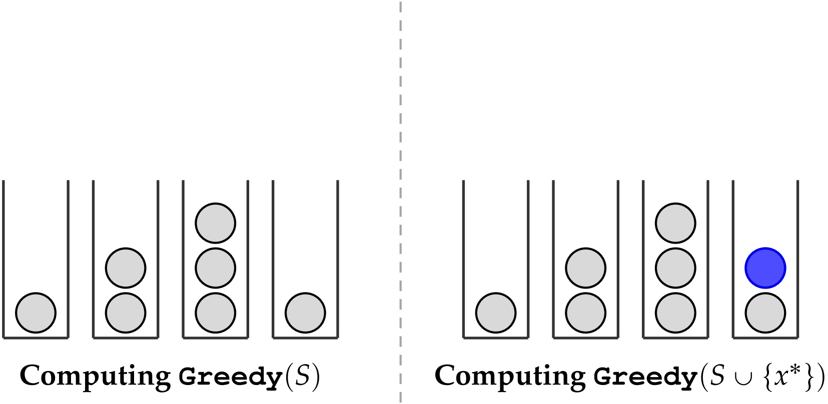


Computing $\texttt{\textbf{Greedy}}(S)$ | **Computing** $\texttt{\textbf{Greedy}}(S \cup \{x^*\})$

Subsequent balls will experience either:

1. No recourse

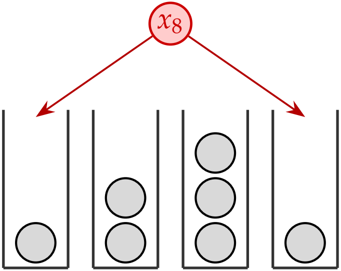
ANALYZING THE RECOURSE



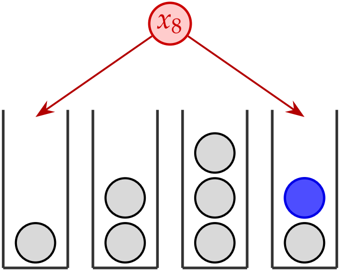
Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse



Computing `Greedy`(S)

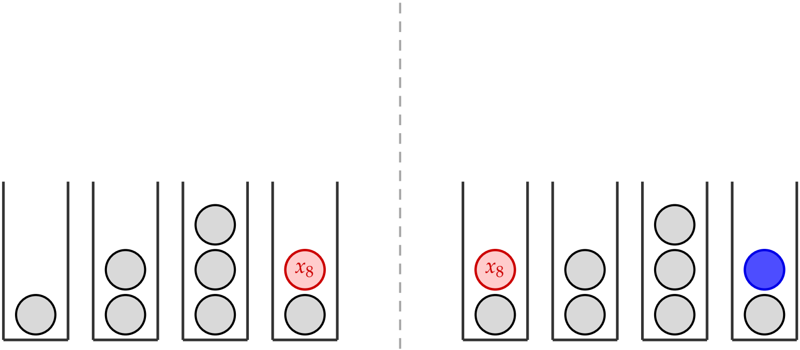


Computing `Greedy`$(S \cup \{x^*\})$

Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse

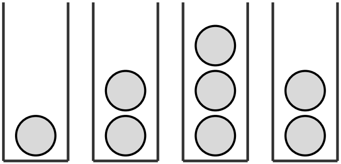


Computing $\texttt{Greedy}(S)$ **Computing** $\texttt{Greedy}(S \cup \{x^*\})$

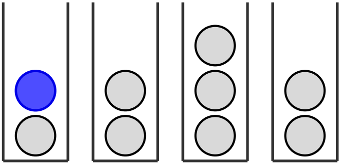
Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse



Computing $\texttt{Greedy}(S)$

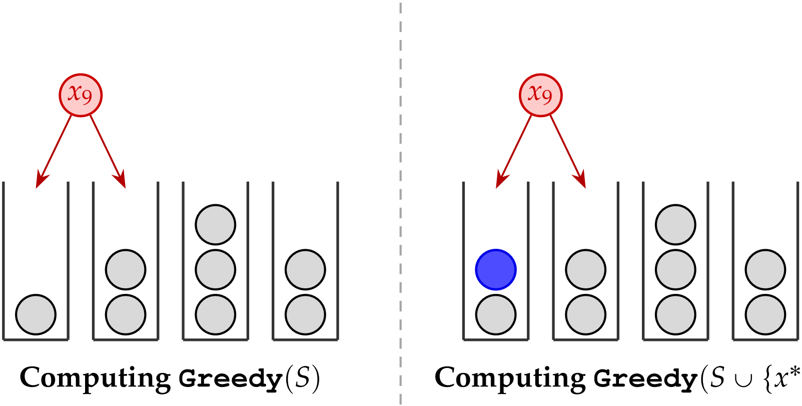


Computing $\texttt{Greedy}(S \cup \{x^*\})$

Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse



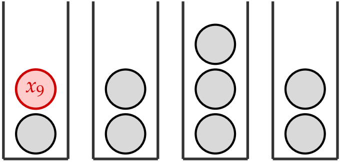
Computing $\texttt{\textbf{Greedy}}(S)$

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

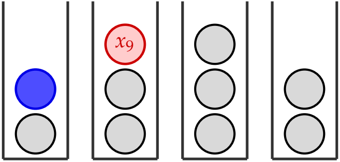
Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse



Computing `Greedy`(S)

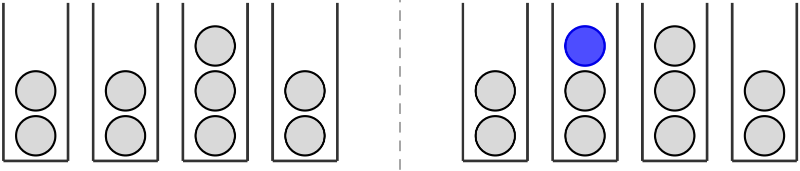


Computing `Greedy`$(S \cup \{x^*\})$

Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse

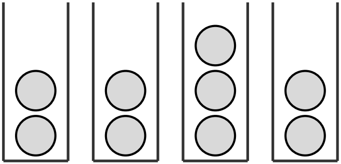


Computing `Greedy`(S) | **Computing `Greedy`**$(S \cup \{x^*\})$

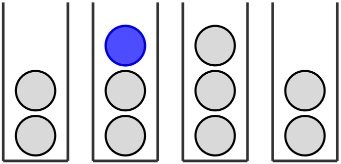
Subsequent balls will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse



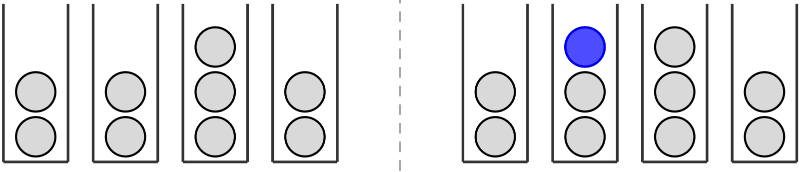
Computing `Greedy`(S)



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

Two key observations:

Analyzing the Recourse

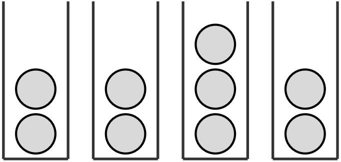


Computing `Greedy`(S) **Computing `Greedy`**$(S \cup \{x^*\})$

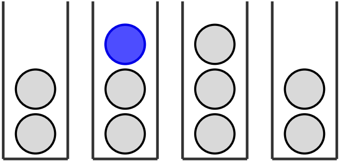
Two key observations:

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

ANALYZING THE RECOURSE



Computing **`Greedy`**(S)

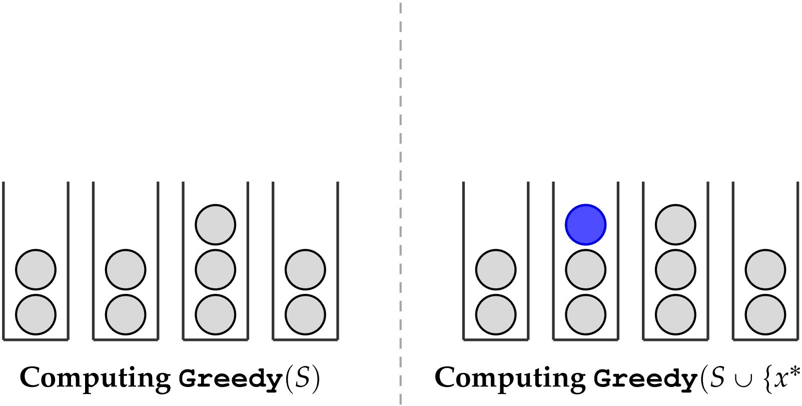


Computing **`Greedy`**$(S \cup \{x^*\})$

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

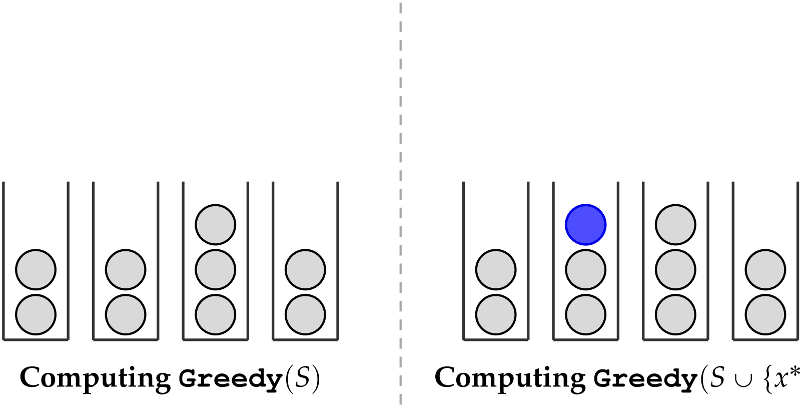


Computing `Greedy`(S)

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

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

Analyzing the Recourse



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

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

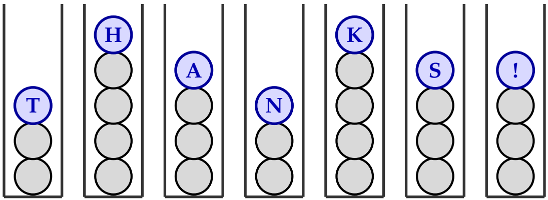
$$\implies \mathbb{E}[\text{total recourse}] = \sum_i \Pr[\text{ball } x_i \text{ incurs recourse}] = O(m/n)$$

A Simple Warmup

Theorem: There exists a history-independent solution with:

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

History-Independent Load Balancing



Michael A. Bender
Stony Brook University

Bill Kuszmaul
CMU

Elaine Shi
CMU

Rose Silver
CMU