History-Independent Load Balancing

Michael A. Bender[1] Bill Kuszmaul[2] Elaine Shi[2] **Rose Silver[2]**

[1]Stony Brook University

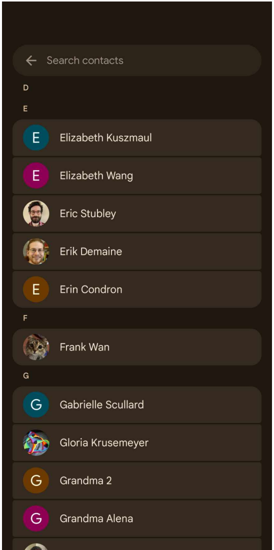
[2]Carnegie Mellon University

History Independent Data Structures

History Independence: "If an adversary were to see the state of the data structure, they would learn only the current set of elements, and nothing else about the history of past operations."

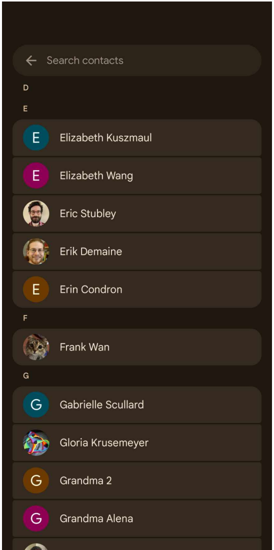
[Micciancio '97], [Naor, Teague '01]

History vs Content



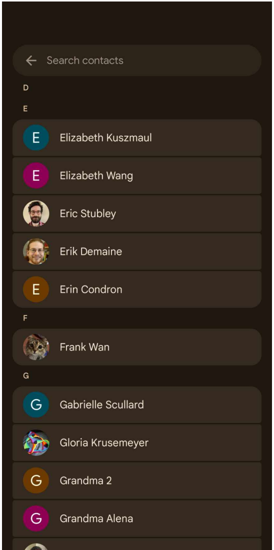
- If someone hacks my phone, they can learn my contacts list.

History vs Content



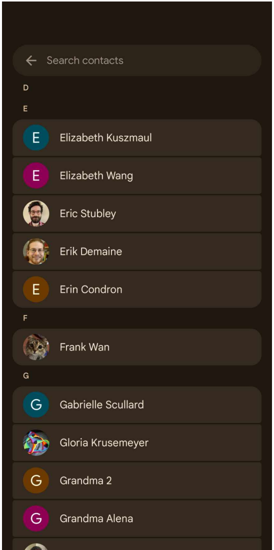
- If someone hacks my phone, they can learn my contacts list.
- But can they learn who my contacts were in the past?

History vs Content



- If someone hacks my phone, they can learn my contacts list.
- But can they learn who my contacts were in the past?
- What about the order in which contacts were added?

History vs Content



- If someone hacks my phone, they can learn my contacts list.
- But can they learn who my contacts were in the past?
- What about the order in which contacts were added?
- A history independent data structure protects this kind of information.

History Independent is a Security Guarantee

History Independence: "If an adversary were to see the state of the data structure, they would learn only the current set of elements, and nothing else about the history of past operations."

[Micciancio '97], [Naor, Teague '01]

History Independent is a Security Guarantee

History Independence: "If an adversary were to see the state of the data structure, they would learn only the current set of elements, and nothing else about the history of past operations."

[Micciancio '97], [Naor, Teague '01]

Lost of successes: Hash tables, trees, memory allocation, PMAs, graph algorithms, B-trees, cache-oblivious data structures... [Micciancio '97], [Naor, Teague '01], [Buchbinder, Petrank '03], [Molnar, Kohno, Sastry, Wagner '06], [Blelloch, Golovin '07], [Moran, Naor, Segev '07] [Naor, Segev, Wieder '08], [Golovin '08 '09 '10], [Tzouramanis '12], [Bajaj, Sion '13] [Bajaj, Chakrabati, Sion '15], [Roche, Aviv, Choi '15], [Bender, Berry, Johnson, Kroeger, McCauley, Phillips, Simon, Singh, Zage '16], ...

History Independent is a Security Guarantee

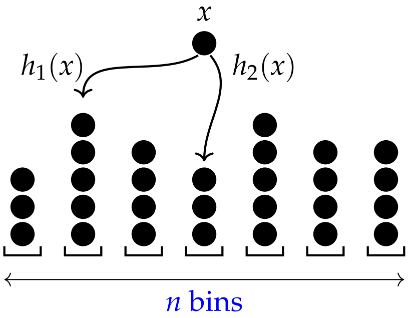
History Independence: "If an adversary were to see the state of the data structure, they would learn only the current set of elements, and nothing else about the history of past operations."

[Micciancio '97], [Naor, Teague '01]

Lost of successes: Hash tables, trees, memory allocation, PMAs, graph algorithms, B-trees, cache-oblivious data structures... [Micciancio '97], [Naor, Teague '01], [Buchbinder, Petrank '03], [Molnar, Kohno, Sastry, Wagner '06], [Blelloch, Golovin '07], [Moran, Naor, Segev '07] [Naor, Segev, Wieder '08], [Golovin '08 '09 '10], [Tzouramanis '12], [Bajaj, Sion '13] [Bajaj, Chakrabati, Sion '15], [Roche, Aviv, Choi '15], [Bender, Berry, Johnson, Kroeger, McCauley, Phillips, Simon, Singh, Zage '16], ...

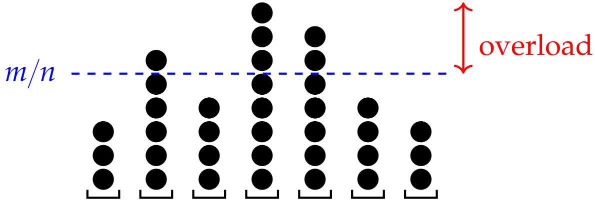
But... some very basic questions also remain open.

Two-Choice Load Balancing



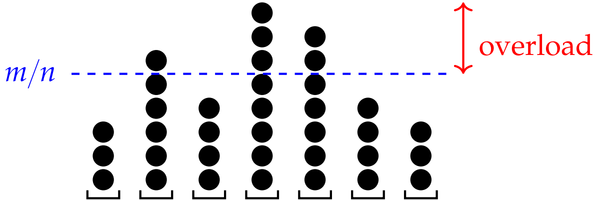
- Balls are inserted/deleted, with up to m present at a time.
- Each ball has two random bins where it can go.
- We must maintain a valid assignment of balls to bins.

TWO GOALS



Minimize Overload:
The amount by which the fullest bin exceeds m/n is small.

Two Goals



Minimize Overload:
The amount by which the fullest bin exceeds m/n is small.

Minimize Recourse:
On any given insertion/deletion, the number of balls moved around is small.

This Paper

Question: Does there exist a history-independent solution with small recourse and overload?

This Paper

Question: Does there exist a history-independent solution with small recourse and overload?

Theorem: There exists a history-independent solution with:

- Overload $O(1)$, with high probability.
- Expected recourse $O(\log\log(m/n))$.

What About Non-History-Independent Solutions?

Lots of work on the insertion-only case.
[Azar, Broder, Karlin and Upfal ’94] [Berenbrink, Czumaj, Steger, and Vöcking ’00][Dietzfelbinger and Weidling ’07] [Frieze and Petti ’18] . . .

What About Non-History-Independent Solutions?

Lots of work on the insertion-only case.
[Azar, Broder, Karlin and Upfal '94] [Berenbrink, Czumaj, Steger, and Vöcking '00][Dietzfelbinger and Weidling '07] [Frieze and Petti '18] . . .

But the fully dynamic case has remained largely open.
[Vöcking '99] [Dietzfelbinger and Weidling '07] [Bender, Conway, Farach-Colton, Kuszmaul, Tagliavini '21] [Bansal, Kuszmaul '22] . . .

What About Non-History-Independent Solutions?

Lots of work on the insertion-only case.
[Azar, Broder, Karlin and Upfal '94] [Berenbrink, Czumaj, Steger, and Vöcking '00][Dietzfelbinger and Weidling '07] [Frieze and Petti '18] ...

But the fully dynamic case has remained largely open.
[Vöcking '99] [Dietzfelbinger and Weidling '07] [Bender, Conway, Farach-Colton, Kuszmaul, Tagliavini '21] [Bansal, Kuszmaul '22] ...

Open Question:
Is there a fully dynamic solution with recourse $o(m/n)$ and overload $O(1)$?

What About Non-History-Independent Solutions?

Lots of work on the insertion-only case.
[Azar, Broder, Karlin and Upfal '94] [Berenbrink, Czumaj, Steger, and Vöcking '00][Dietzfelbinger and Weidling '07] [Frieze and Petti '18] ...

But the fully dynamic case has remained largely open.
[Vöcking '99] [Dietzfelbinger and Weidling '07] [Bender, Conway, Farach-Colton, Kuszmaul, Tagliavini '21] [Bansal, Kuszmaul '22] ...

Open Question:
Is there a fully dynamic solution with recourse $o(m/n)$ and overload $O(1)$?

Answer:
Yes! We get recourse $O(\log\log(m/n))$ and overload $O(1)$!

This Paper

Question: Does there exist a history-independent solution with small recourse and overload?

Theorem: There exists a history-independent solution with:

- Overload $O(1)$, with high probability.
- Expected recourse $O(\log\log(m/n))$.

This Paper

Question: Does there exist a history-independent solution with small recourse and overload?

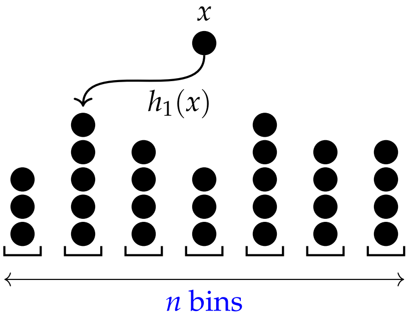
Theorem: There exists a history-independent solution with:

- Overload $O(1)$, with high probability.
- Expected recourse $O(\log \log (m/n))$.

Rest of Talk: A simple history-independent algorithm with overload $O(\log \log n)$ and expected recourse $O(m/n)$.

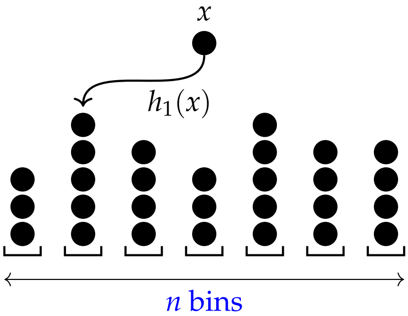
Warmup 1: The Single-Choice Strategy

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



WARMUP 1: THE SINGLE-CHOICE STRATEGY

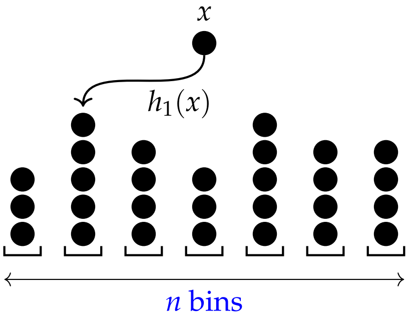
To insert a ball x, just put it in bin $h_1(x)$:



- This is history-independent ✓

Warmup 1: The Single-Choice Strategy

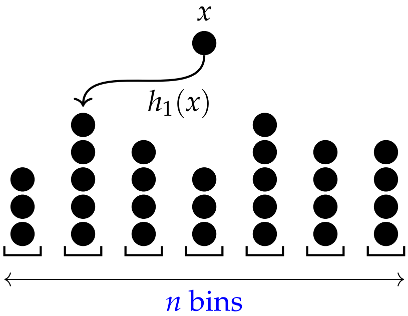
To insert a ball x, just put it in bin $h_1(x)$:



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

WARMUP 1: THE SINGLE-CHOICE STRATEGY

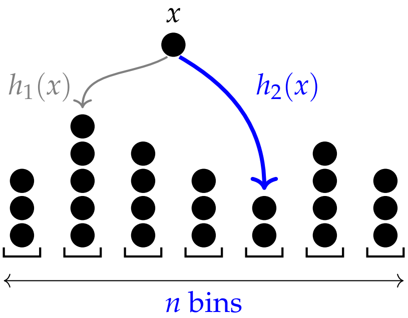
To insert a ball x, just put it in bin $h_1(x)$:



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

Warmup 2: Greedy Insertions

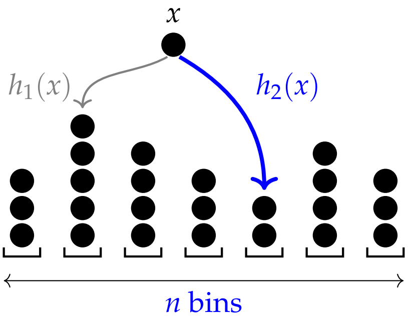
To insert a ball x, put it in the emptier of its choices:



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

Warmup 2: Greedy Insertions

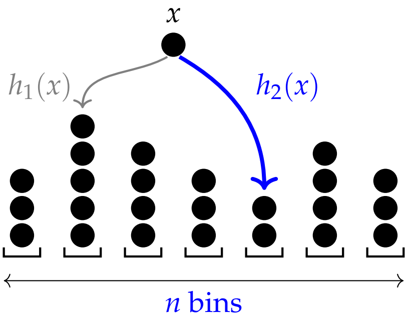
To insert a ball x, put it in the emptier of its choices:



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

Warmup 2: Greedy Insertions

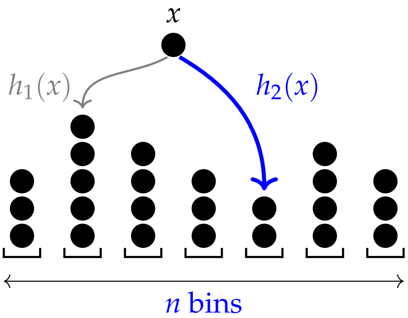
To insert a ball x, put it in the emptier of its choices:



- 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]

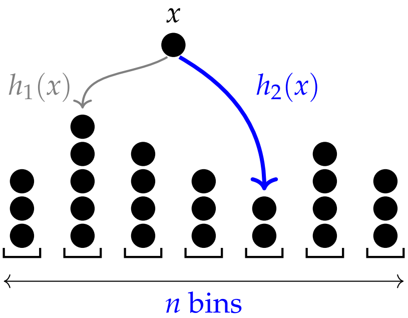
A Simple History-Independent Algorithm



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

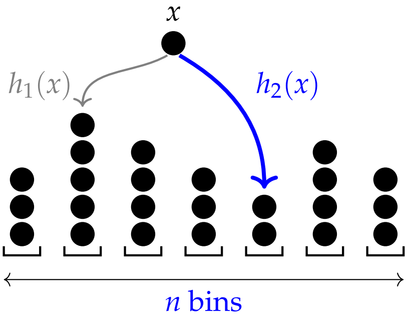


Given a set S of balls, define $\texttt{Greedy}(S)$ as:

- 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.

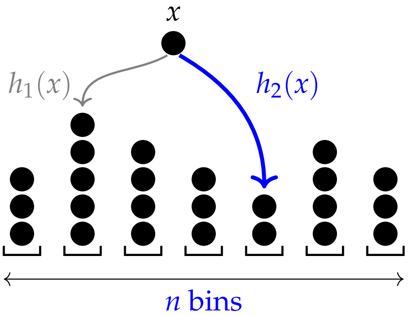
A History-Independent Algorithm: At any given moment, if S is the current set of balls, use allocation $\texttt{Greedy}(S)$.

Analyzing History-Independent Greedy



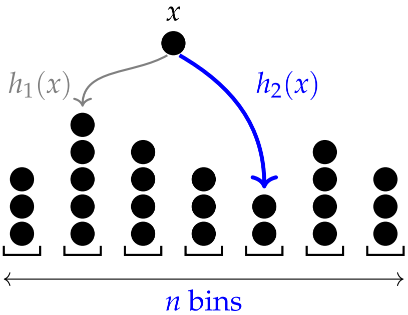
- The algorithm is history independent ✓

Analyzing History-Independent Greedy



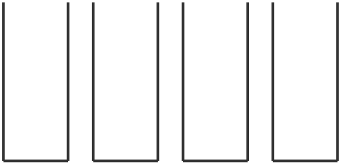
- The algorithm is history independent ✓
- The overload is $O(\log \log n)$ ✓

Analyzing History-Independent Greedy

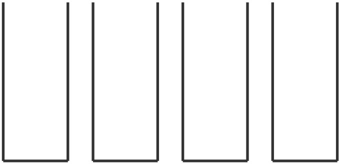


- The algorithm is history independent ✓
- The overload is $O(\log \log n)$ ✓
- But what about the recourse?

Analyzing the Recourse



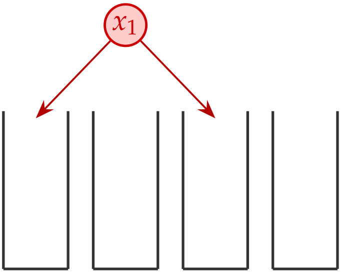
Computing `Greedy`(S)



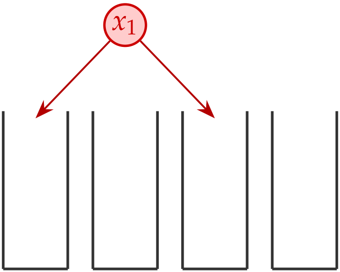
Computing `Greedy`$(S \cup \{x^*\})$

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

Analyzing the Recourse

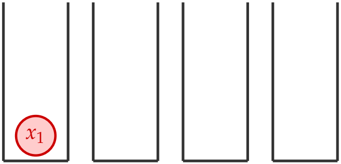


Computing $\texttt{\textbf{Greedy}}(S)$

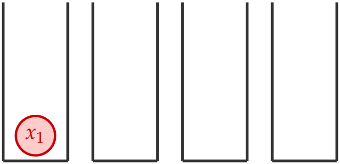


Computing $\texttt{\textbf{Greedy}}(S \cup \{x^*\})$

Analyzing the Recourse

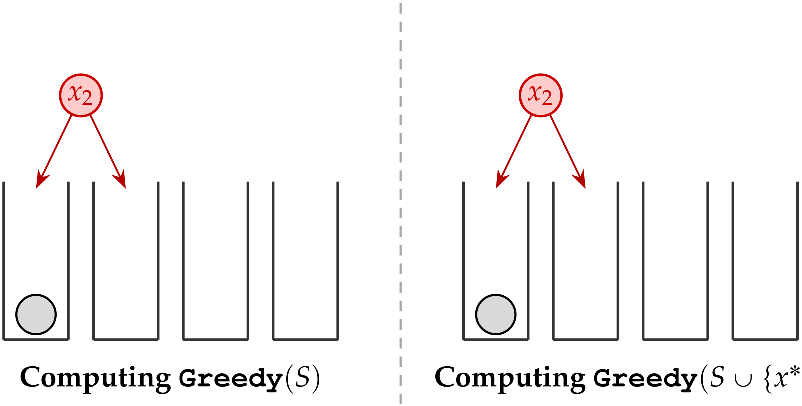


Computing $\texttt{\textbf{Greedy}}(S)$



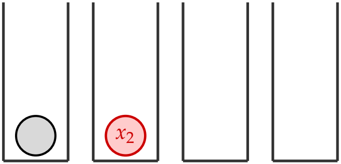
Computing $\texttt{\textbf{Greedy}}(S \cup \{x^*\})$

Analyzing the Recourse

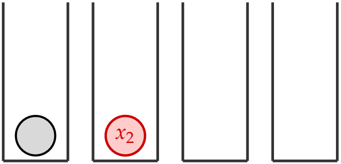


Computing $\texttt{Greedy}(S)$ | **Computing** $\texttt{Greedy}(S \cup \{x^*\})$

Analyzing the Recourse

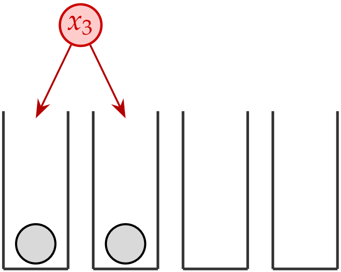


Computing **`Greedy`**(S)

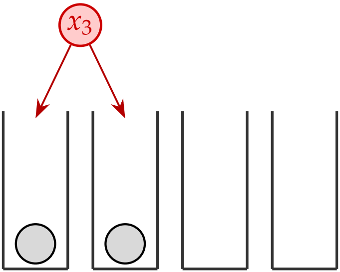


Computing **`Greedy`**$(S \cup \{x^*\})$

Analyzing the Recourse

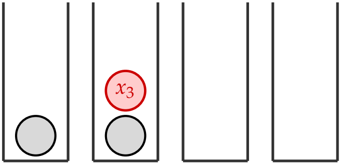


Computing `Greedy`(S)

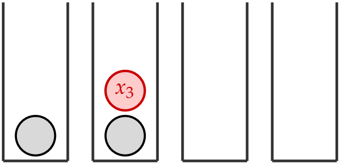


Computing `Greedy`$(S \cup \{x^*\})$

Analyzing the Recourse

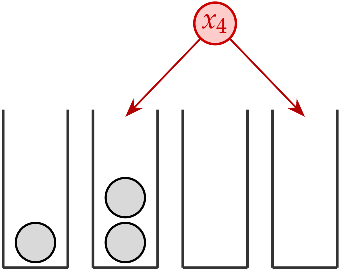


Computing `Greedy`(S)

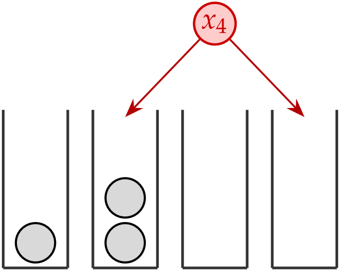


Computing `Greedy`$(S \cup \{x^*\})$

Analyzing the Recourse

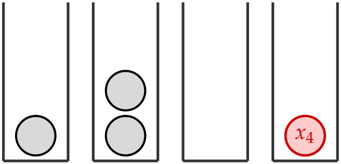


Computing `Greedy`(S)

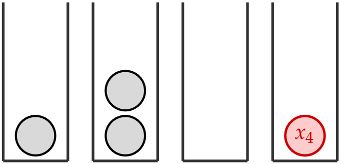


Computing `Greedy`$(S \cup \{x^*\})$

Analyzing the Recourse

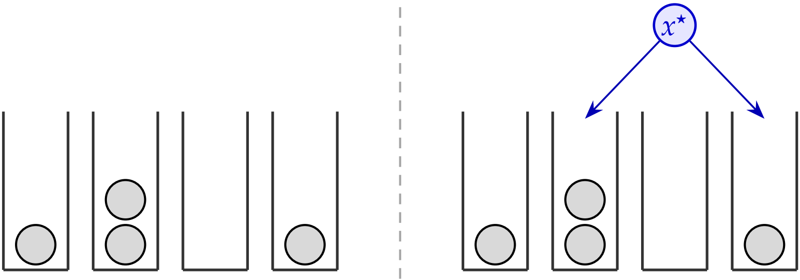


Computing $\texttt{\textbf{Greedy}}(S)$



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

Analyzing the Recourse

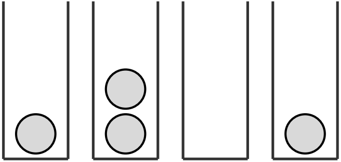


Computing $\texttt{Greedy}(S)$

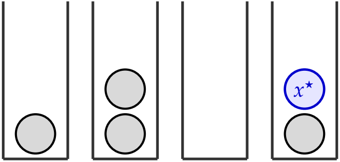
Computing $\texttt{Greedy}(S \cup \{x^*\})$

- $x^\star$ arrives only in World 1

Analyzing the Recourse



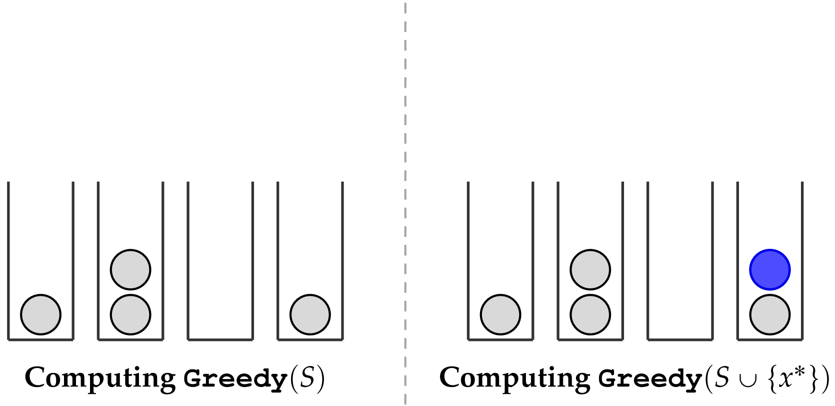
Computing $\texttt{Greedy}(S)$



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

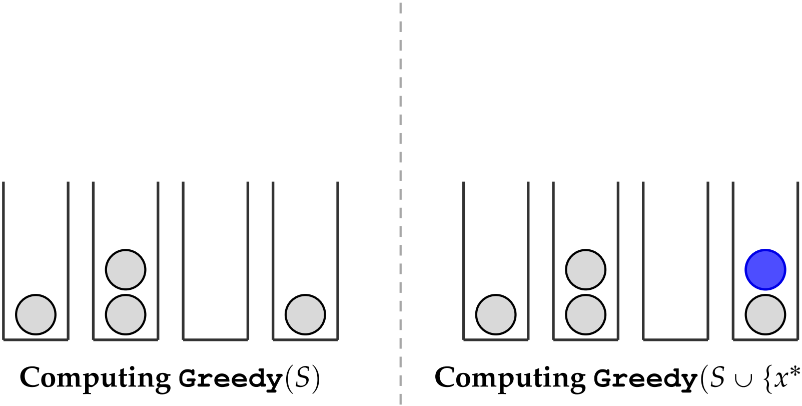
- $x^\star$ arrives only in World 1

Analyzing the Recourse



Question: How do subsequent insertions differ between the two worlds?

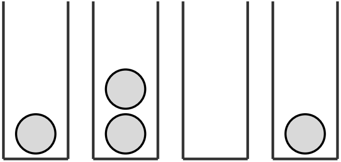
Analyzing the Recourse



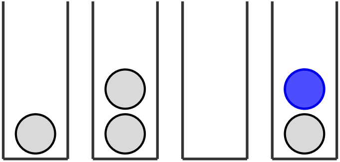
Computing $\texttt{\textbf{Greedy}}(S)$ **Computing** $\texttt{\textbf{Greedy}}(S \cup \{x^*\})$

Future insertions will experience either:

Analyzing the Recourse



Computing `Greedy`(S)

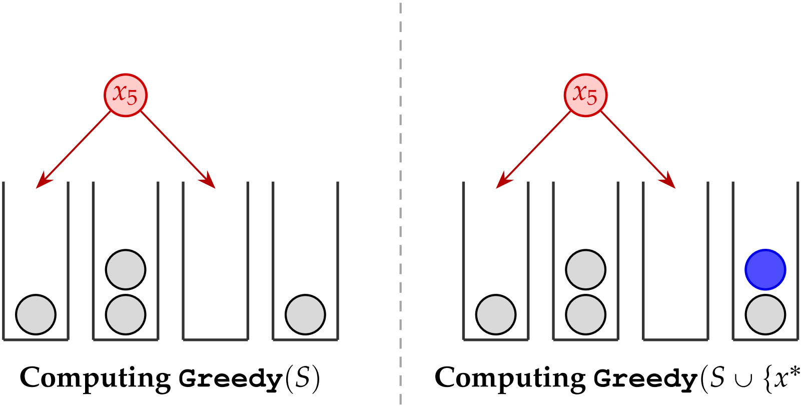


Computing `Greedy`$(S \cup \{x^*\})$

Future insertions will experience either:

1. No recourse

Analyzing the Recourse

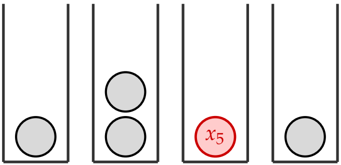


Computing `Greedy`(S) **Computing** `Greedy`$(S \cup \{x^*\})$

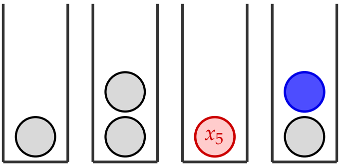
Future insertions will experience either:

1. No recourse

Analyzing the Recourse



Computing `Greedy`(S)

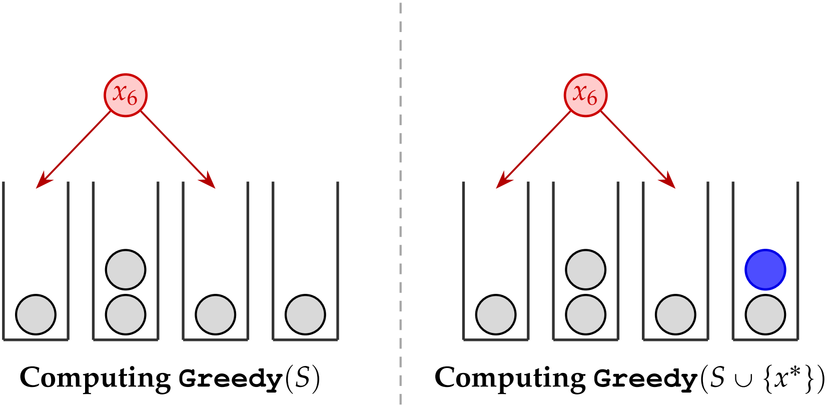


Computing `Greedy`$(S \cup \{x^*\})$

Future insertions will experience either:

1. No recourse

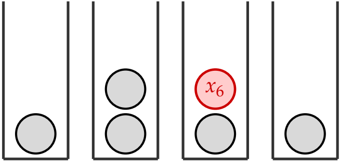
Analyzing the Recourse



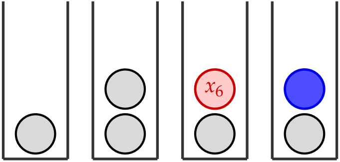
Future insertions will experience either:

1. No recourse

Analyzing the Recourse



Computing `Greedy`(S)

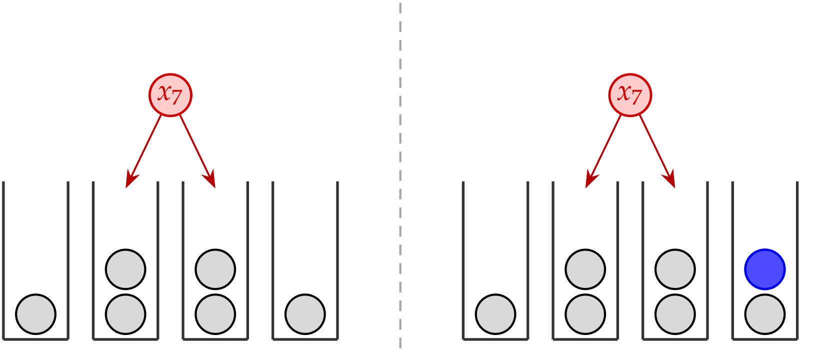


Computing `Greedy`$(S \cup \{x^*\})$

Future insertions will experience either:

1. No recourse

Analyzing the Recourse

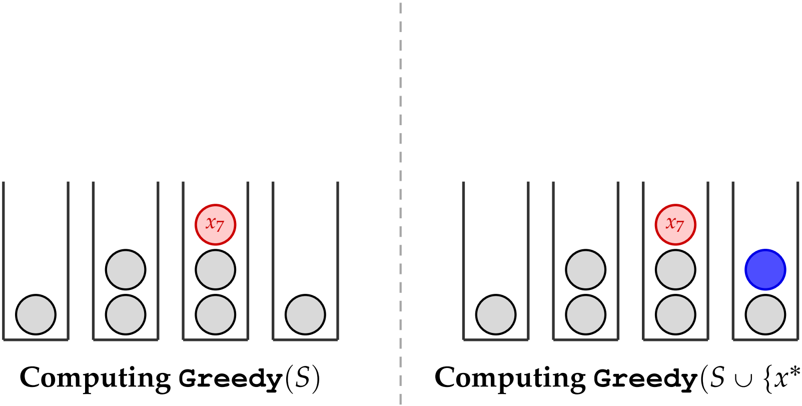


Computing **`Greedy`**(S) **Computing** **`Greedy`**$(S \cup \{x^*\})$

Future insertions will experience either:

1. No recourse

Analyzing the Recourse

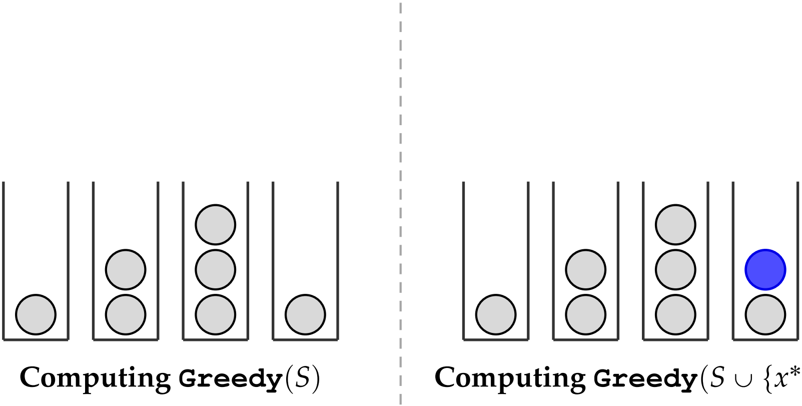


Computing **`Greedy`**(S) **Computing** **`Greedy`**$(S \cup \{x^*\})$

Future insertions will experience either:

1. No recourse

ANALYZING THE RECOURSE



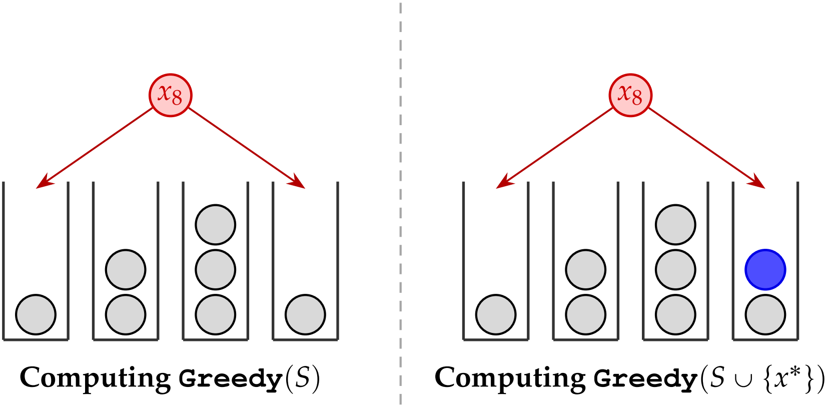
Computing `Greedy`(S)

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

Future insertions will experience either:

1. No recourse
2. Recourse

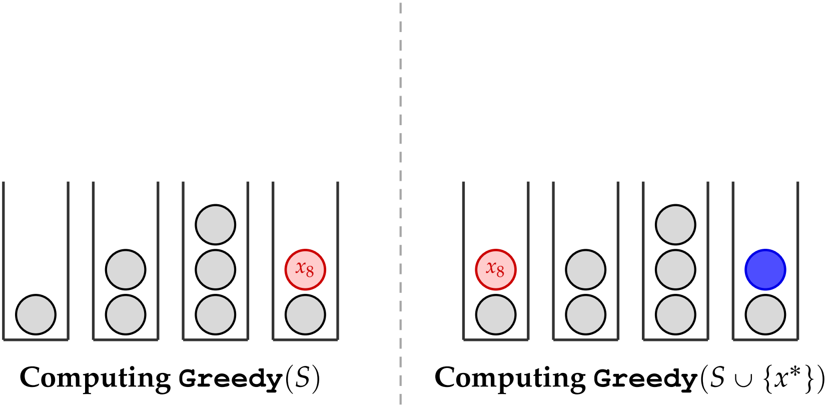
Analyzing the Recourse



Future insertions will experience either:

1. No recourse
2. Recourse

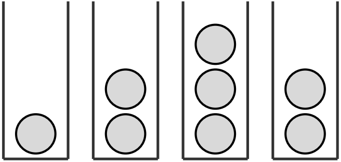
Analyzing the Recourse



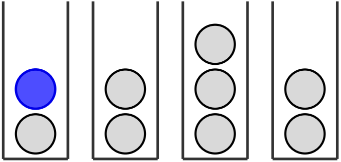
Future insertions will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse



Computing `Greedy`(S)

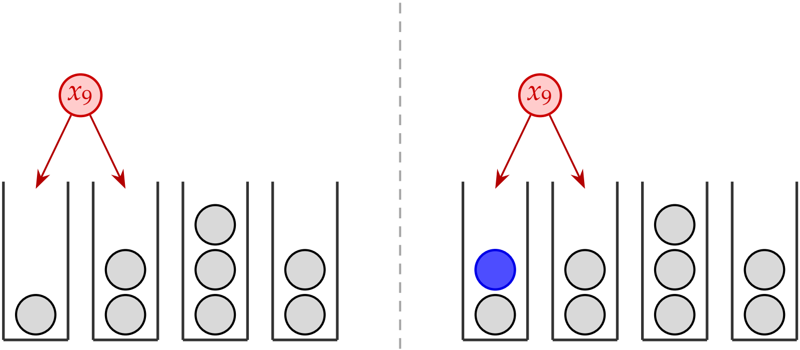


Computing `Greedy`$(S \cup \{x^*\})$

Future insertions will experience either:

1. No recourse
2. Recourse

ANALYZING THE RECOURSE

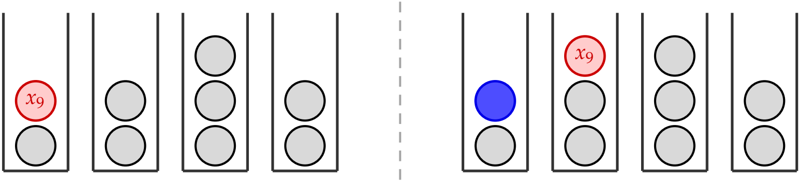


Computing $\texttt{Greedy}(S)$ **Computing** $\texttt{Greedy}(S \cup \{x^*\})$

Future insertions will experience either:

1. No recourse
2. Recourse

ANALYZING THE RECOURSE

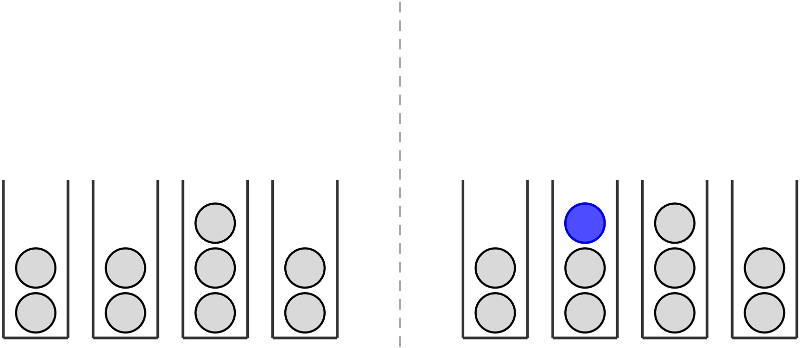


Computing `Greedy`(S) **Computing** `Greedy`$(S \cup \{x^*\})$

Future insertions will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse

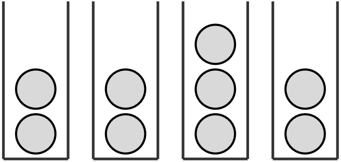


Computing `Greedy`(S) | **Computing `Greedy`**$(S \cup \{x^*\})$

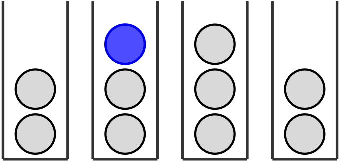
Future insertions will experience either:

1. No recourse
2. Recourse

Analyzing the Recourse



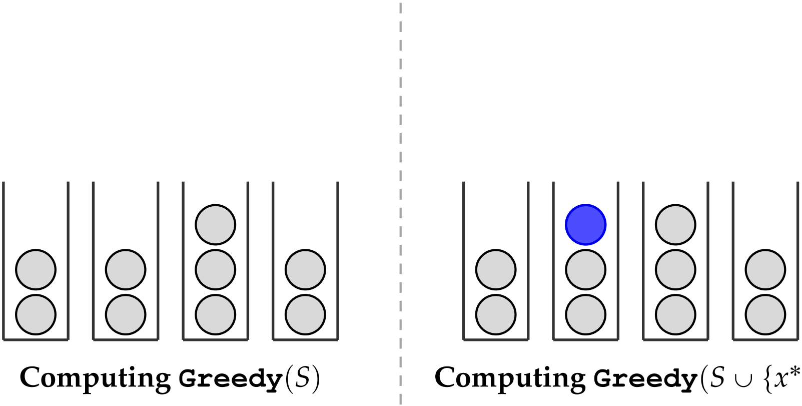
Computing `Greedy`(S)



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

Two key observations:

Analyzing the Recourse



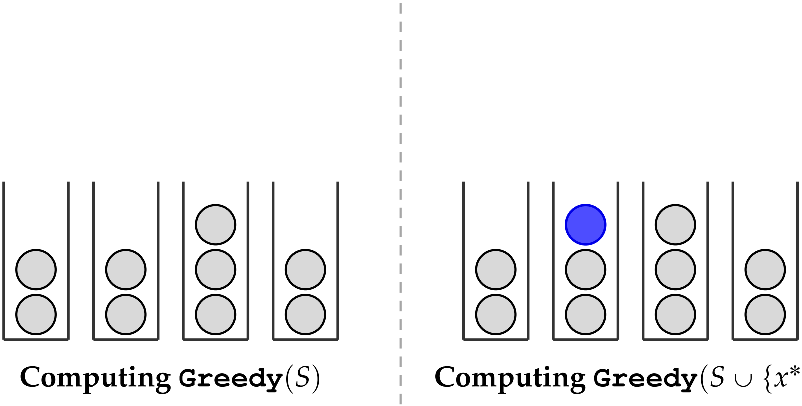
Computing $\texttt{\textbf{Greedy}}(S)$

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

Two key observations:

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

Analyzing the Recourse

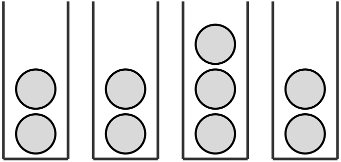


Computing $\texttt{Greedy}(S)$ **Computing** $\texttt{Greedy}(S \cup \{x^*\})$

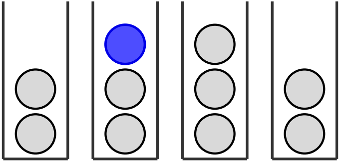
Two key observations:

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



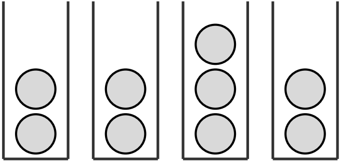
Computing `Greedy`(S)



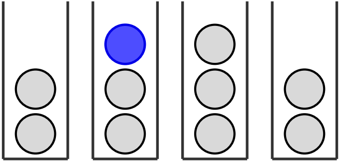
Computing `Greedy`$(S \cup \{x^*\})$

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

Analyzing the Recourse



Computing $\texttt{Greedy}(S)$



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

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

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

This Paper

Question: Does there exist a history-independent solution with small recourse and overload?

Theorem: There exists a history-independent solution with:

- Overload $O(1)$, with high probability.
- Expected recourse $O(\log \log(m/n))$.

This Paper

Question: Does there exist a history-independent solution with small recourse and overload?

Theorem: There exists a history-independent solution with:

- Overload $O(1)$, with high probability.
- Expected recourse $O(\log\log(m/n))$.

Rest of Talk: A simple history-independent algorithm with overload $O(\log\log n)$ and expected recourse $O(m/n)$.