Randomized Data Structures:
Hash Tables

Admin

- Assignment 8 is tonight
- Assignment 9 will be released today
    - Last assignment to be turned in
- Health day: no Lecture on Friday 🎉🎉🎉
- Final will be open book, take home 24-hour exam
    - Can be taken between May 20 until May 28 (by 8.30 pm)
    - Logistics will be similar to the midterm

Hash table

- Array of size m that can store up to n items
    - Often have $m = 2n$ or $m = 1.5n$

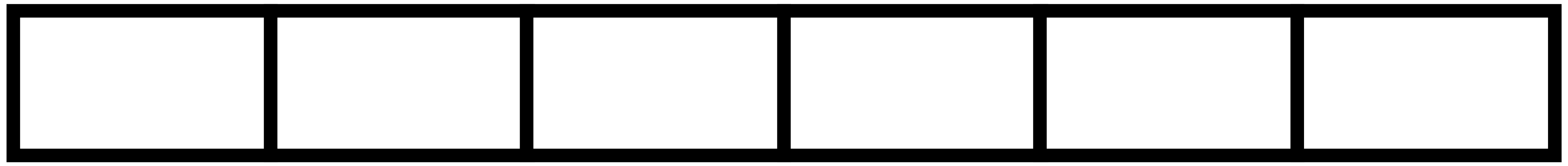
- Want $O(1)$ expected operations:
    - Insert a new item
    - Look up an item
    - Delete an item (we won't discuss)

- Key: hash *function* that maps each item to a slot

Hash table

- Hash function h, array A
- Item i is stored in $A[h(i)]$
- Let's assume that there is only one item that hashes to each slot.
  Then, we're done: $O(1)$ time insert, lookup, delete

Amir

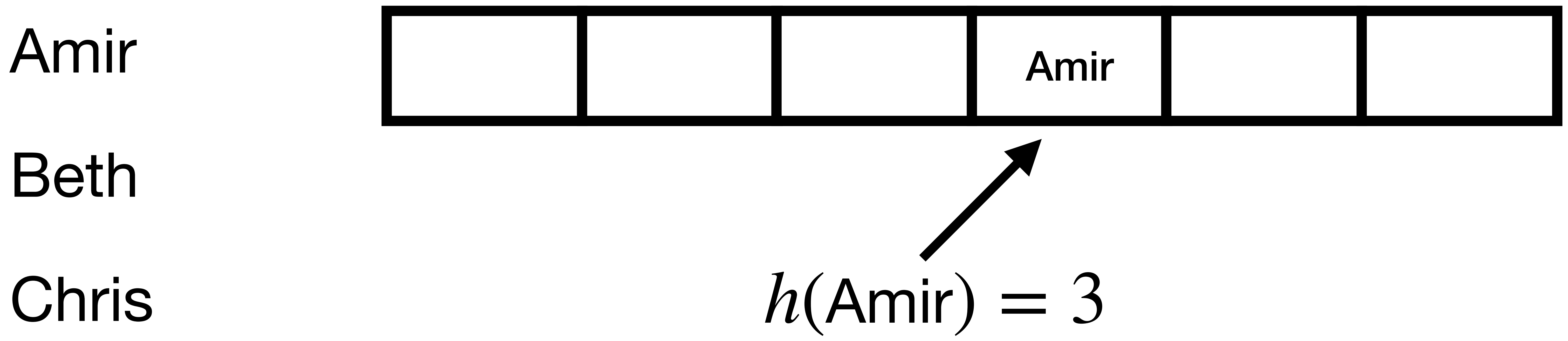


Beth

Chris

Hash table

- Hash function h, array A
- Item i is stored in $A[h(i)]$
- Let's assume that there is only one item that hashes to each slot. Then, we're done: $O(1)$ time insert, lookup, delete



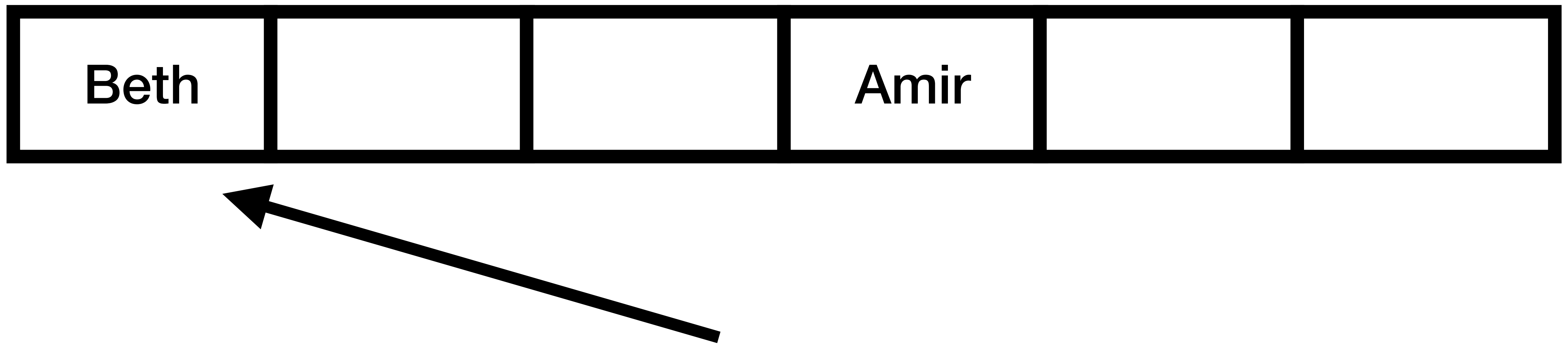
Hash table

- Hash function h, array A
- Item i is stored in $A[h(i)]$
- Let's assume that there is only one item that hashes to each slot. Then, we're done: $O(1)$ time insert, lookup, delete

Amir

Beth

Chris



$h(\text{Beth}) = 0$

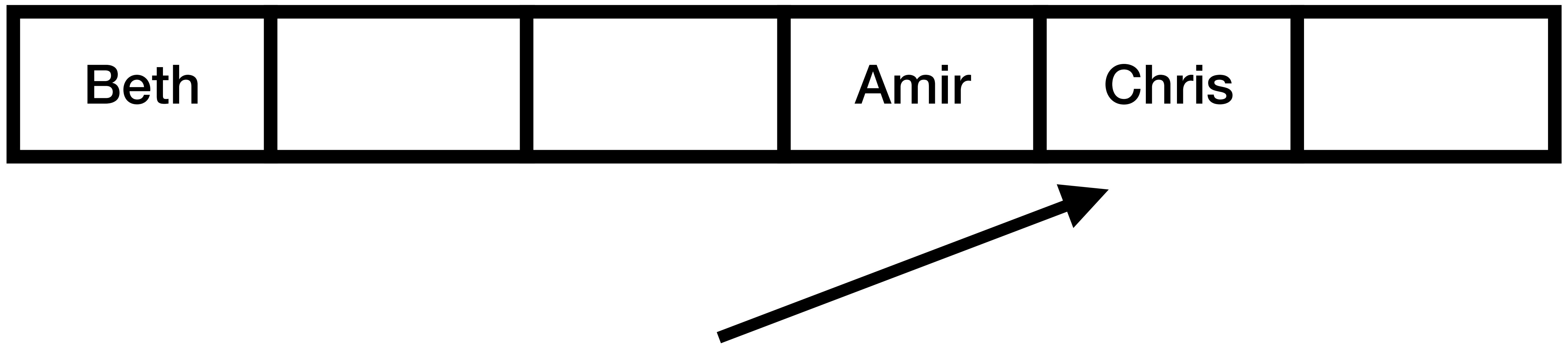
Hash table

- Hash function h, array A
- Item i is stored in $A[h(i)]$
- Let's assume that there is only one item that hashes to each slot. Then, we're done: $O(1)$ time insert, lookup, delete

Amir

Beth

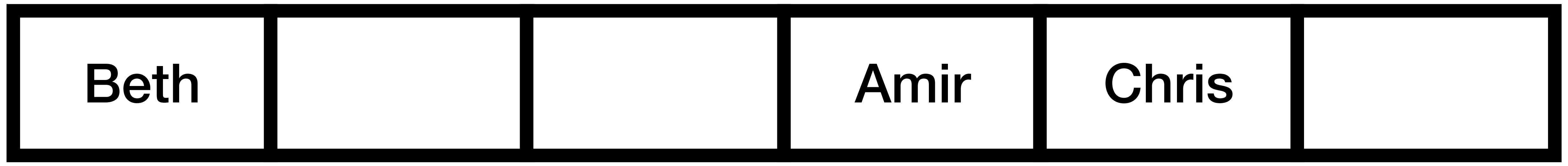
Chris



$h(\text{Chris}) = 4$

Hash function

- Goal: for any set of items, the hash function maps the items to different slots as much as possible (spreads them out)
- How can we achieve this?
- Idea: use randomness



Hash function: theory versus practice

- Select a hash function from a random family
- Classic example:
- $h(i) = (ai + b) \mod p \mod m$
- a and b are chosen at random; selecting them determines the exact hash function
- p is a large prime
- For any items i_1, i_2: $\Pr_{a,b}\left[h(i_1) = h(i_2)\right] = 1/m$
- By choosing a *random* hash function, we can guarantee that *any* two items probably don't collide

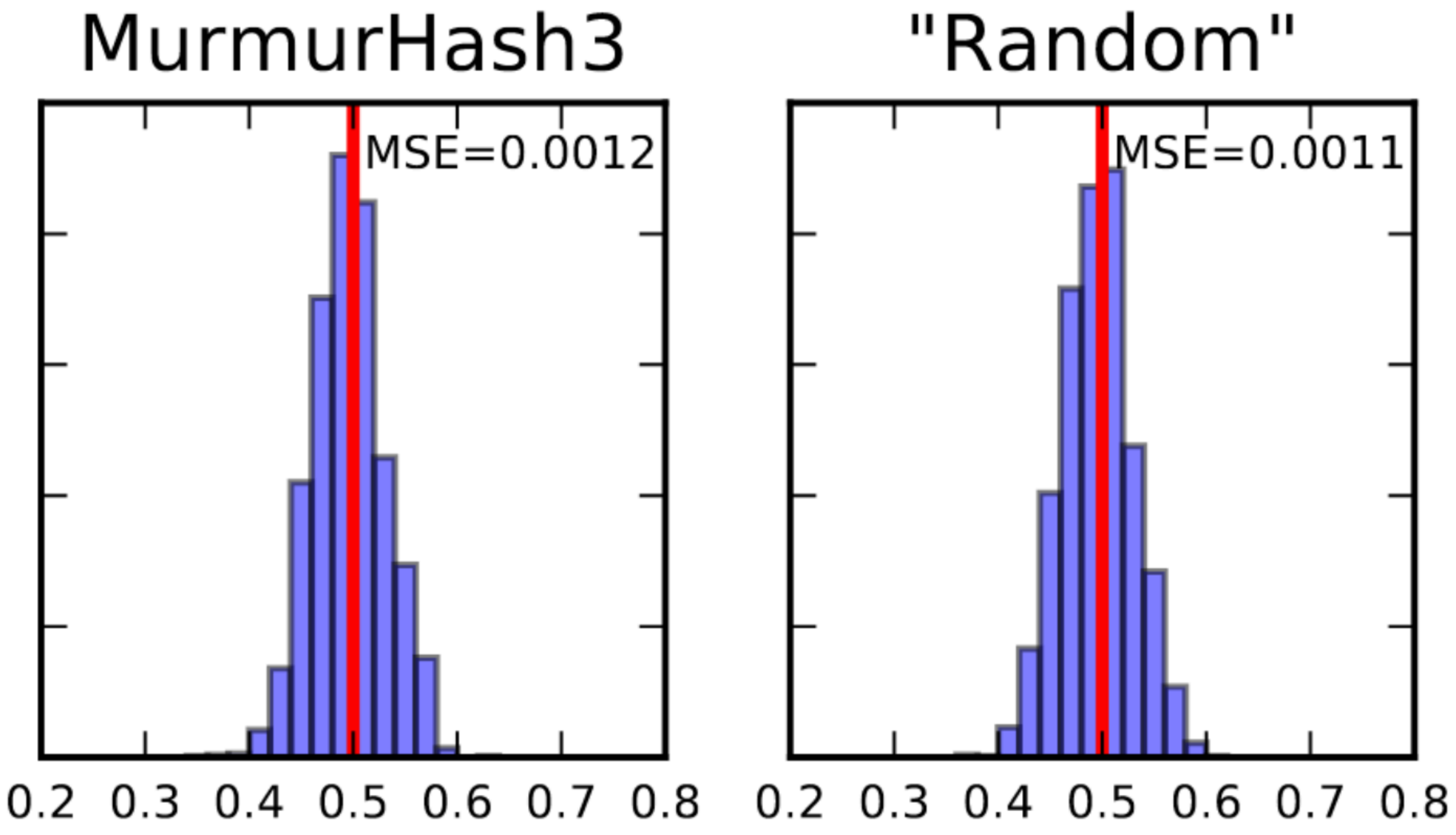
Beth			Amir	Chris	

Hash function: theory versus practice

- We will assume hash function is *ideal* :
  - For all i, k, $\Pr(h(i) = k) = 1/m$
  - The hashes of all items are independent:
    $\Pr(h(i) = k \,|\, h(i_2) = k_2, h(i_3) = k_3, \ldots) = 1/m$

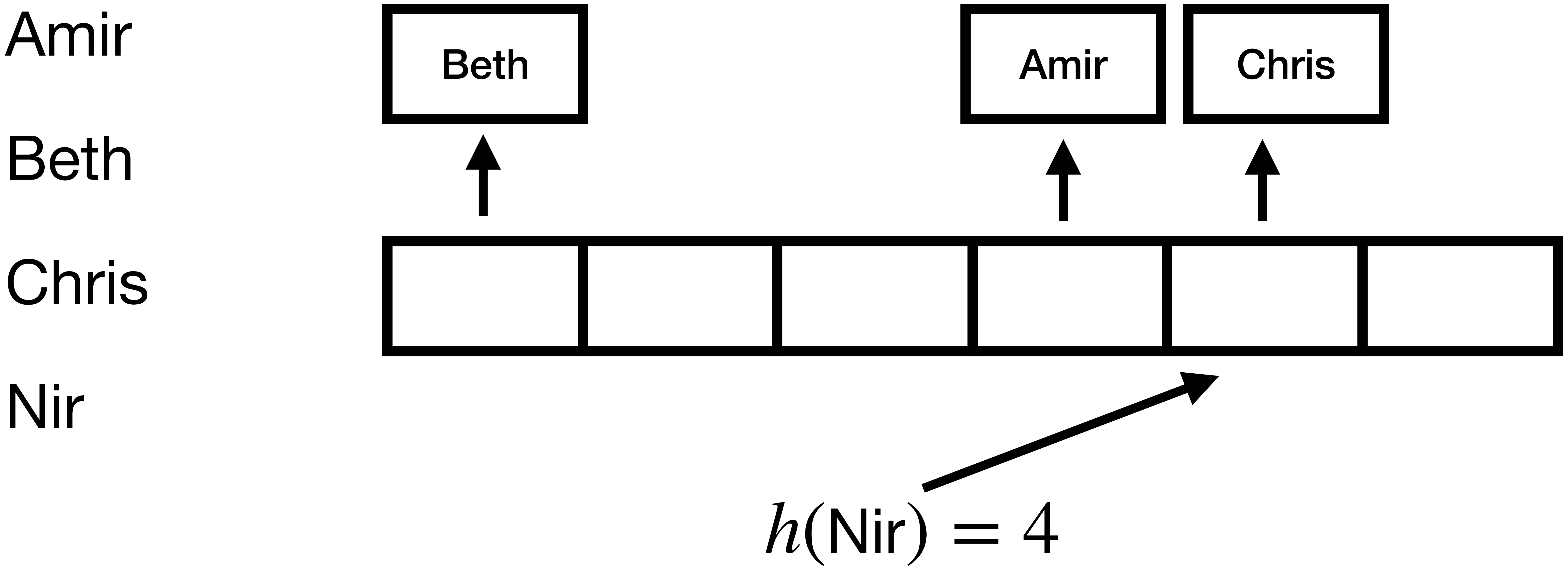
- Called **uniform random hash functions**
- Good hash functions do behave this way in practice
- Lots of theoretical work about weaker assumptions on the hash functions

Dahlgaard et al. 2017



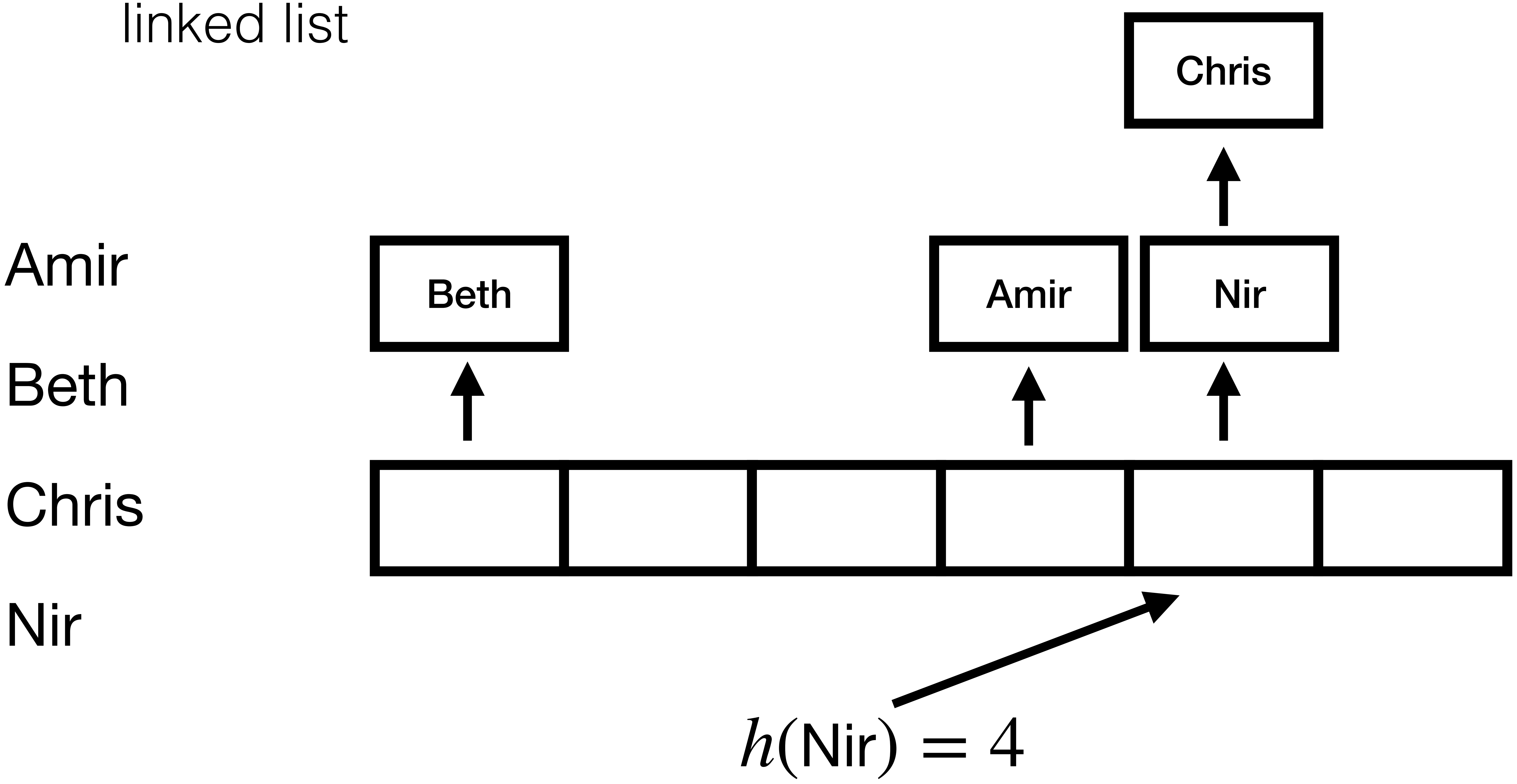
Chaining

- Store a linked list at each array entry
- When an item hashes to a slot, prepend it to the linked list



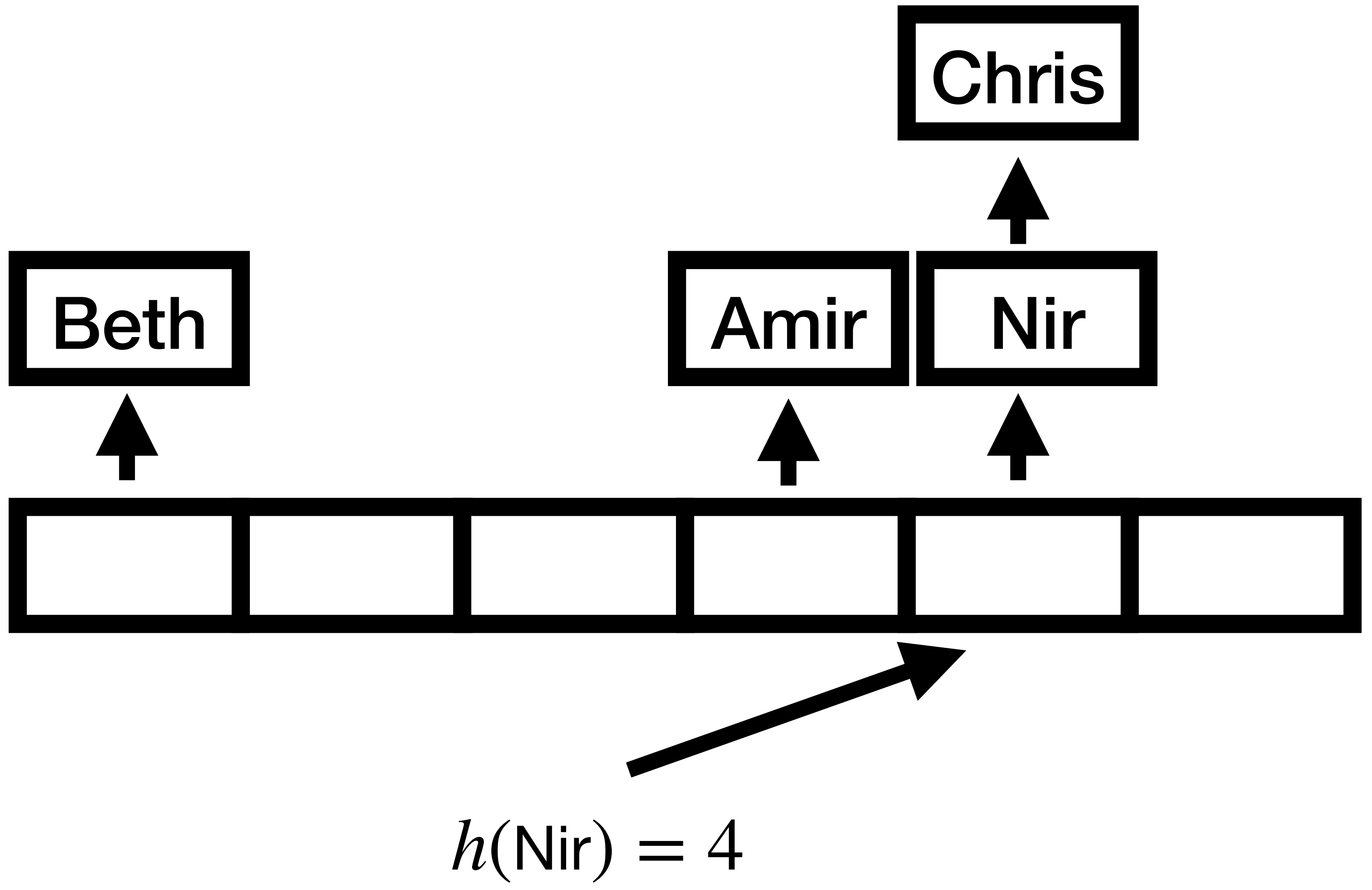
Chaining

- Store a linked list at each array entry
- When an item hashes to a slot, prepend it to the linked list



Chaining

- Store a linked list at each array entry
- When an item hashes to a slot, prepend it to the linked list



- How can we insert?

- How can we lookup?

- How much time does insert/lookup take?

Hash Tables Analysis:
via Balls and Bins

Balls and Bins

- Consider the process of throwing n balls into m bins
- Each ball is thrown into a uniformly random bin, independent of other balls
- What does the distribution of balls in bins looks like?
- This process helps analyze hashing, other processes
    - Distributed load balancing (assigning requests to servers), etc.

Balls and Bins: Hashing

- Expected lookup time in a hash table with chaining
  - Same as expected number of balls in a particular bin
- **Question.** Expected number of balls in a particular bin b?
- Homework 8, Problem 2
- As long as table size m is constant factor of number of elements n, expected cost of hashing with chaining is $O(1)$
- Expectation is just average; but how long can the chains get?
  - How many balls in the **fullest bin**?
  - Turns out, not too big even if $m = n$

Hashing: With High Probability Bounds

- **Lemma.** If n balls are thrown independently and uniformly into n bins, then with high probability, the fullest bin contains $O\left(\frac{\log n}{\log \log n}\right)$

- To prove this we first answer a few helpful questions

- What is the probability that a bin has exactly k balls?

  - $\binom{n}{k}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{n-k}$

- Let X be the number of balls in a bin, then X has a **binomial distribution**

Hashing: With High Probability Bounds

- **Lemma.** If n balls are thrown independently and uniformly into n bins, then with high probability, the fullest bin contains $O\left(\frac{\log n}{\log \log n}\right)$
- To prove this we first answer a few helpful questions
- What is the probability that a bin has exactly k balls?
  - $\binom{n}{k}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{n-k}$
- Let X be the number of balls in a bin, then X has a **binomial distribution**

Hashing: With High Probability Bounds

- **Lemma.** If n balls are thrown independently and uniformly into n bins, then with high probability, the fullest bin contains $O(\frac{\log n}{\log \log n})$
- The probability that a bin has at least k balls? $\leq \binom{n}{k}\left(\frac{1}{n}\right)^k$
- Select k balls, and multiply with probability they land in the bin, the other $n - k$ balls can land anywhere

Hashing: With High Probability Bounds

- **Lemma.** If n balls are thrown independently and uniformly into n bins, then with high probability, the fullest bin contains $O(\frac{\log n}{\log\log n})$

- The probability that a bin has at least k balls? $\leq \binom{n}{k}\left(\frac{1}{n}\right)^k$

$$\leq \left(\frac{en}{k}\right)^k \frac{1}{n^k} = \left(\frac{e}{k}\right)^k$$

- Recall ~~death-bed~~ **formula** $\left(\frac{y}{x}\right)^x \leq \binom{y}{x} \leq \left(\frac{ey}{x}\right)^x$

- Let $k = \dfrac{4\ln n}{\ln\ln n}$

Hashing: With High Probability Bounds

- **Lemma.** If n balls are thrown independently and uniformly into n bins, then with high probability, the fullest bin contains $O(\dfrac{\log n}{\log \log n})$

- Probability that a bin has at least $\dfrac{4 \ln}{\ln \ln n}$ balls

$$\leq \left(\frac{e}{k}\right)^k = \left(\frac{e \ln \ln n}{4 \ln n}\right)^{\frac{4 \ln n}{\ln \ln n}} \leq \left(\frac{e \ln \ln n}{\ln n}\right)^{\frac{4 \ln n}{\ln \ln n}} = e^{\frac{4 \ln n}{\ln \ln n} \cdot \ln \frac{\ln \ln n}{\ln n}}$$

$$= e^{\frac{4 \ln n}{\ln \ln n} \cdot \left(\ln \ln \ln n - \ln \ln n\right)}$$

$$= e^{-4 \ln n + \frac{4 \ln n \ln \ln \ln n}{\ln \ln n}} \leq e^{-3 \ln n} = \frac{1}{n^3}$$ for sufficiently large n

- Thus, Pr[bin b has at least $4 \ln n / (\ln \ln n)$ balls] $\leq 1/n^3$

Useful: Union Bound

- What we have:
  Pr[bin b has at least $4 \ln n/(\ln \ln n)$ balls] $\leq 1/n^3$
- We want to show that the probability that **any bin** has at least k balls is polynomially small in n
- Useful technique for this
- **Union Bound**: For any events $A_1, A_2, \ldots, A_k$, we have

$$\Pr(A_1 \cup A_2 \cup \ldots \cup A_k) \leq \Pr(A_1) + \Pr(A_2) + \ldots + \Pr(A_k)$$

Hashing: With High Probability Bounds

- **Lemma.** If n balls are thrown independently and uniformly into n bins, then with high probability, the fullest bin contains $O(\frac{\log n}{\log \log n})$

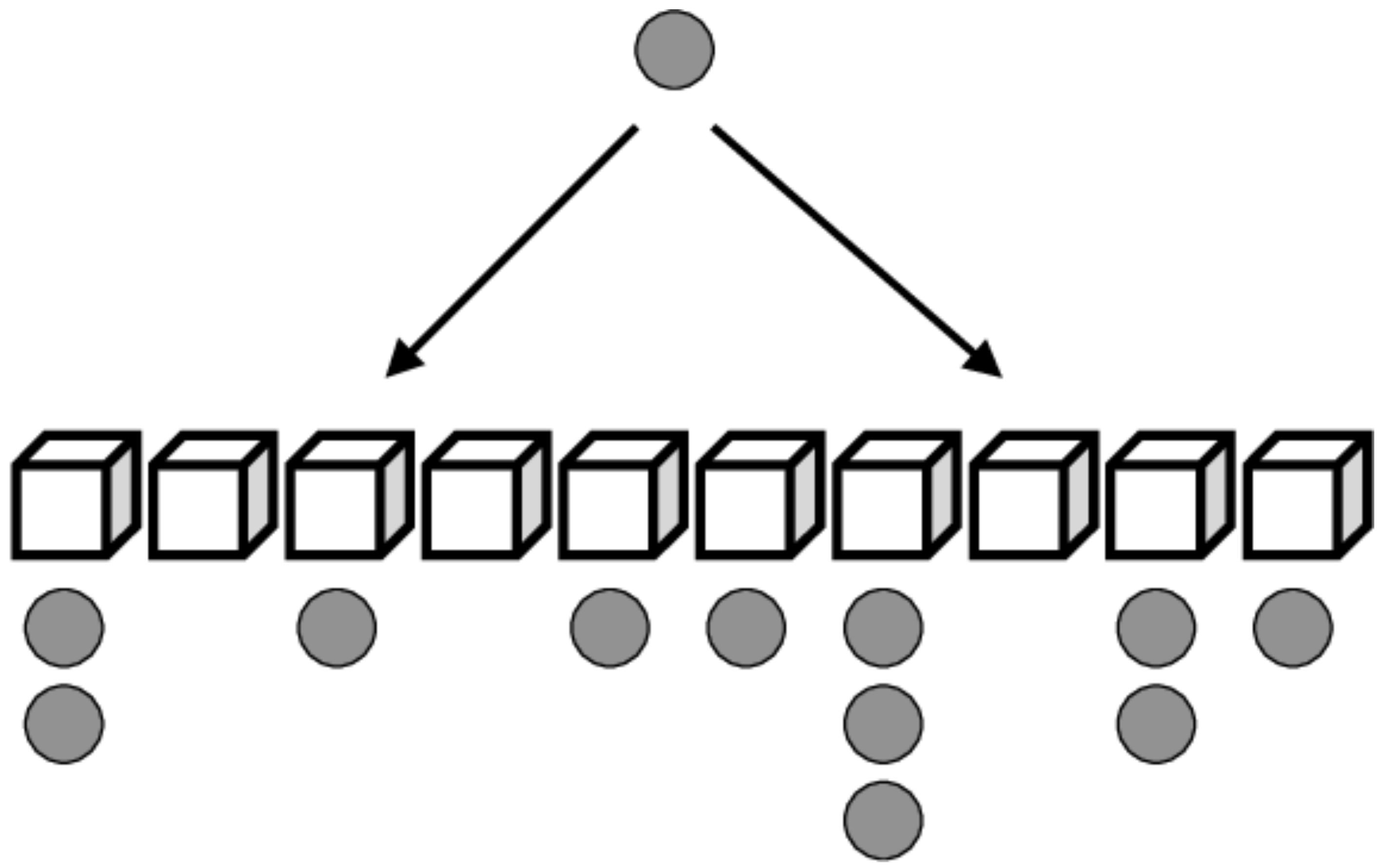
- **By union bound**, the probability that any bin has at least $k = 4\frac{\ln n}{\ln \ln n}$ balls is $\sum_{b=1}^{n}$ Pr[bin b has at least $4 \ln n/(\ln \ln n)$ balls] $\leq \sum_{i=1}^{n} \frac{1}{n^3} = \frac{1}{n^2}$

- Thus, **with high probability** the fullest bin contains $O(\frac{\log n}{\log \log n})$ balls

- **Note.** This bound is tight: fullest bins contains $\Theta(\frac{\log n}{\log \log n})$ balls w.h.p.

Aside: Power of Two Choices

- If we thrown n balls into n bins independently and uniformly at random then the max load is $\Theta\left(\frac{\log n}{\log\log n}\right)$
- **Fun fact.** For each balls, if we instead pick two bins independently and uniformly at random, and put the ball in the less loaded bin, then the max load is at most $\frac{\log\log n}{\log 2} + O(1)$
- Exponential improvement
- Can be generalized to any d choices



Aside: Balls and Bins

- **Question.** How large must n be that it is likely there is a bin with at least two balls?
- Probability that all n balls fall into different bins
- Have we studied this question?
    - Birthday paradox!

Aside: Balls and Bins

- **Question.** How many balls n should be thrown until all bins have at least one ball?

- $\Pr[\text{bin } b \text{ is empty}] = \left(1 - \dfrac{1}{n}\right)^m \approx e^{-m/n}$

$$\left((1 - \frac{1}{x}\right)^x \leq \frac{1}{e}$$

- $\Pr[\text{exists an empty bin}] = \Pr[\cup_{b=1}^{m} \text{bin } b \text{ is empty}]$

$$\leq \sum_{b=1}^{m} \Pr[\text{bin } b \text{ is empty}] = m \cdot e^{-n/m}$$

- E.g. $n = 3m \ln m$, the probability is at most $\dfrac{1}{m^2}$

- That is, if we buy $\Theta(m \log m)$ packs, we get all Pokemons **w.h.p.**

Back to Hashing

Hashing with Chaining

- Let’s say I store the first element of the chain in the table itself
- Then I don’t need a linked list for chains of length 1
- How many chains of length 1 will I have in expectation?
- We'll assume $n = m$
- Let's analyze this using indicator random variables
- Random variable $X_i = 1$ if slot i has exactly one item, 0 otherwise
- By linearity of expectation, we want $\sum_{i=1}^{n} \mathbf{Pr}\left[\text{slot } i \text{ has one item}\right]$

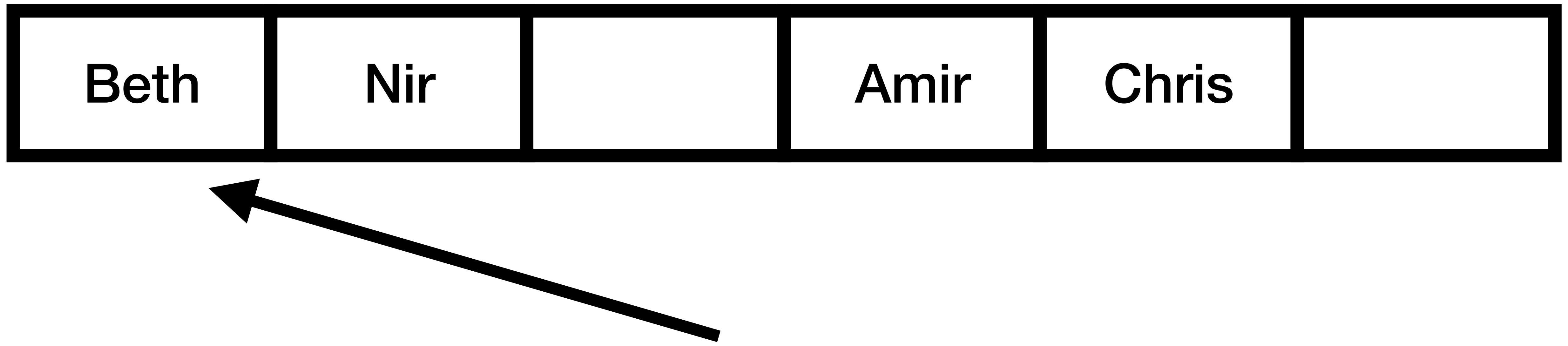
Chaining: Some other questions

- $\Pr[\text{slot } i \text{ has one item}]$
- $= \binom{n}{1}\left(\frac{1}{n}\right)\left(1 - \frac{1}{n}\right)^{n-1}$
- $= \left(1 - \frac{1}{n}\right)^{n-1} \leq \left(1 - \frac{1}{n}\right)^{n} \leq \frac{1}{e}$

- So expected number of slots with a chain of length 1 is n/e

Linear Probing

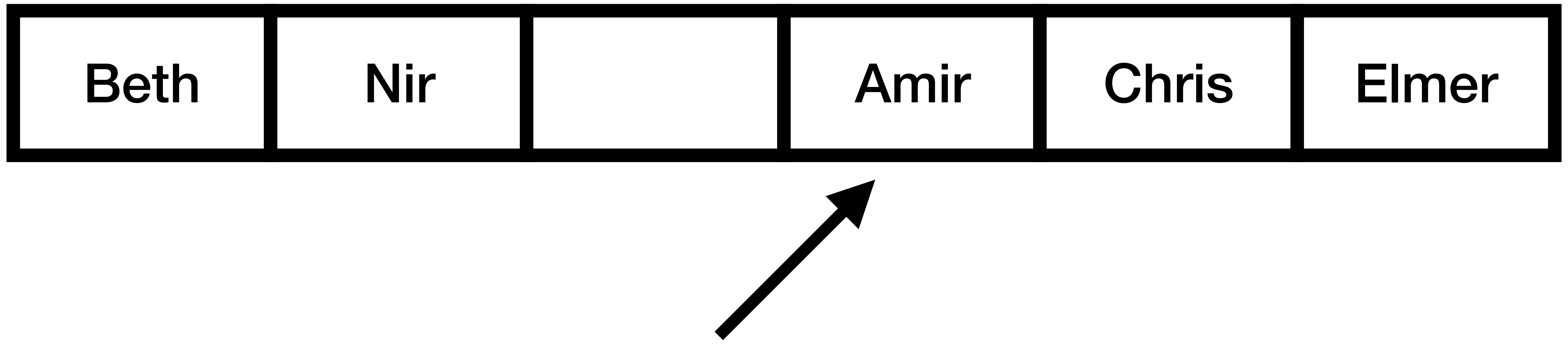
- No linked lists; just the table
- If there is already an item in $A[h(i)]$, check $A[h(i)+1]$, then $A[i+2]$, and so on



$$h(\text{Nir}) = 0$$

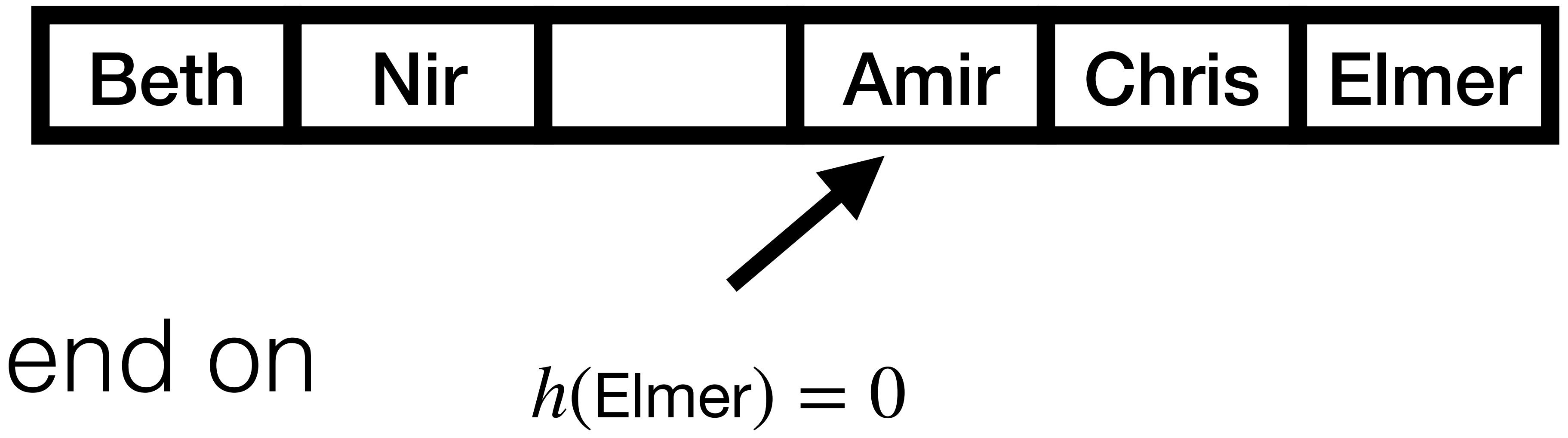
Linear Probing

- No linked lists; just the table
- If there is already an item in $A[h(i)]$, check $A[h(i)+1]$, then $A[i+2]$, and so on
- How can we insert?
- How can we lookup?
- How much time does insert/lookup take?



$$h(\text{Elmer}) = 0$$

Linear Probing



- Calculations are a bit harder because inserts depend on each other
- Larger clusters are more likely to be hashed to, so their size grows
- Expected lookup time if successful [Knuth]:
- $O\left(1 + 1/(1 - n/m)\right)$
- Expected insert/unsuccessful lookup:
- $O\left(1 + 1/(1 - n/m)^2\right)$
- Can show all operations are $O(\log n)$ w.h.p. using similar techniques

Linear Probing vs Chaining?

- What are some advantages of chaining?
    - Constant time inserts
    - Better w.h.p. performance
- What are some advantages of linear probing?
    - Space-efficient
    - Better cache efficiency
- Linear probing is the more common one in practice

Improving Worst Case Lookups

- We need randomness in order to hash
- But can we get worst-case bounds?
- For example, can we get $O(1)$ worst-case lookup, with $O(1)$ expected insert (and $O(\log n)$ insert with high probability)?
- Yes—cuckoo hashing!

Cuckoo Hashing

Cuckoo Hashing

- Uses two hash functions, h_1 and h_2, two hash tables
- Each table size n
- Item i is guaranteed to be in $A[h_1(i)]$ or $A[h_2(i)]$
- So we can lookup in $O(1)$
- How can we insert?

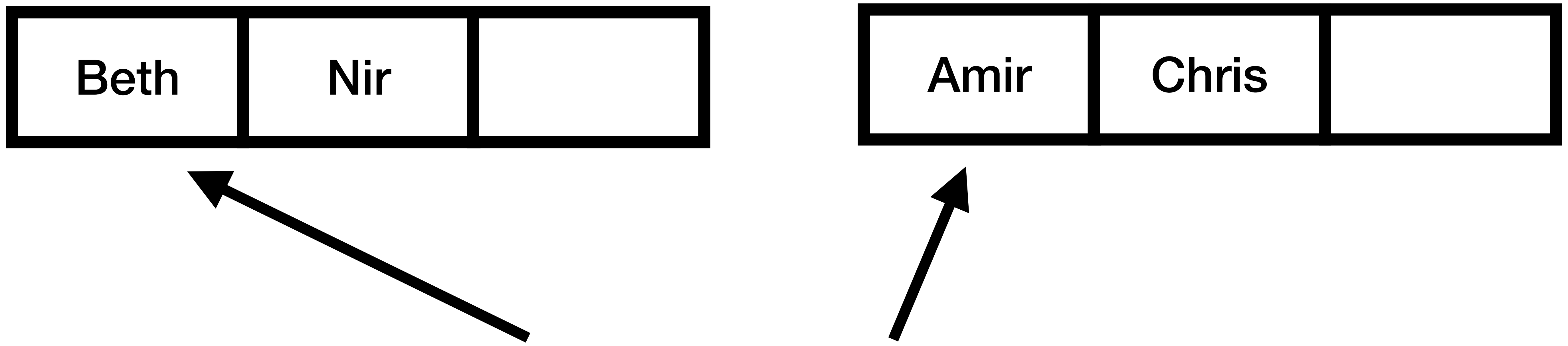
Beth	Nir	

Amir	Chris	

$$h_1(\text{Beth}) = 0, \;\; h_2(\text{Beth}) = 1$$

Cuckoo Hashing: Insert

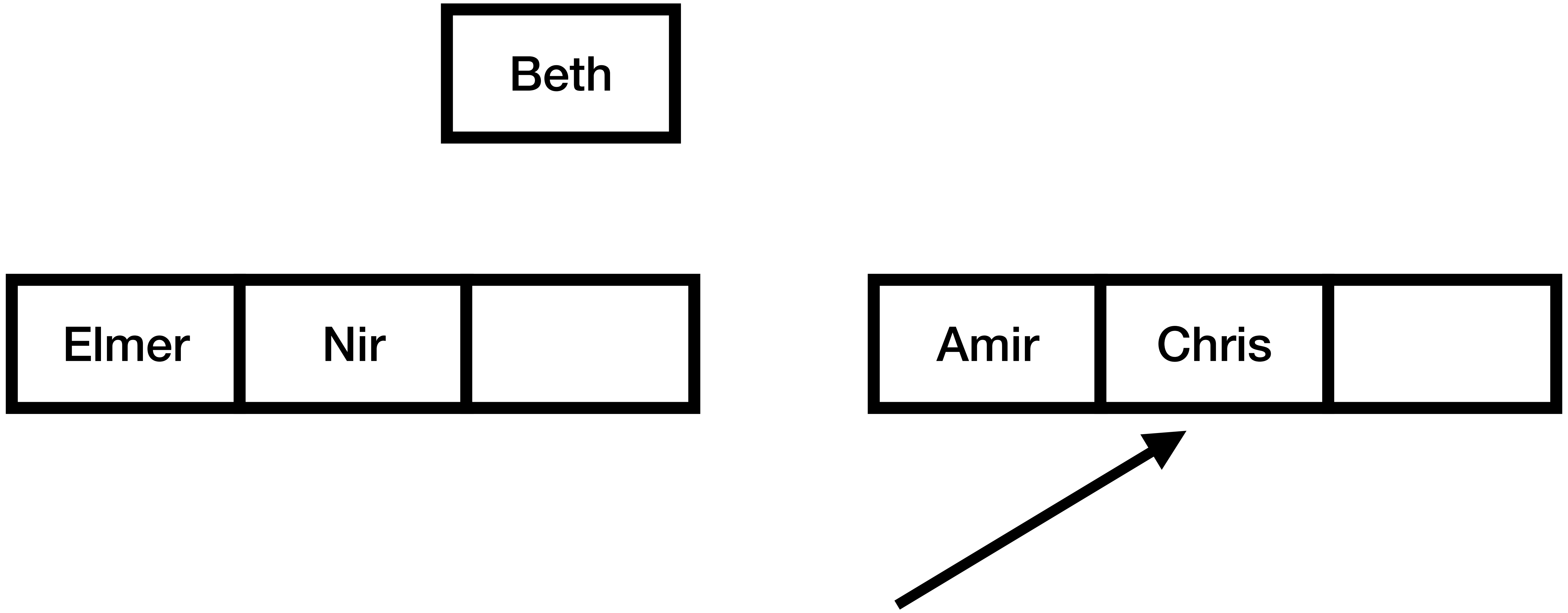
- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash



$$h_1(\text{Elmer}) = 0, \;\; h_2(\text{Elmer}) = 0$$

Cuckoo Hashing: Insert

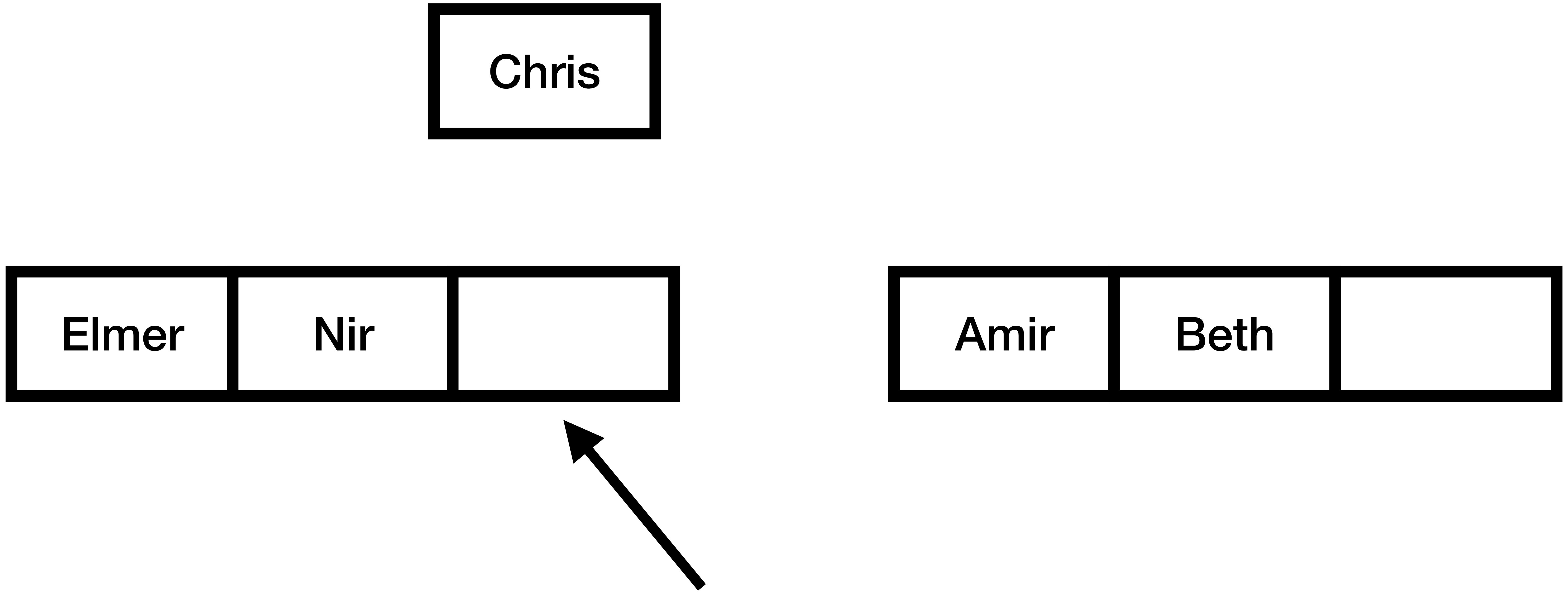
- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash



$$h_1(\text{Beth}) = 0, \;\; h_2(\text{Beth}) = 1$$

Cuckoo Hashing: Insert

- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash



$$h_1(\text{Chris}) = 2, \;\; h_2(\text{Chris}) = 1$$

Cuckoo Hashing: Insert

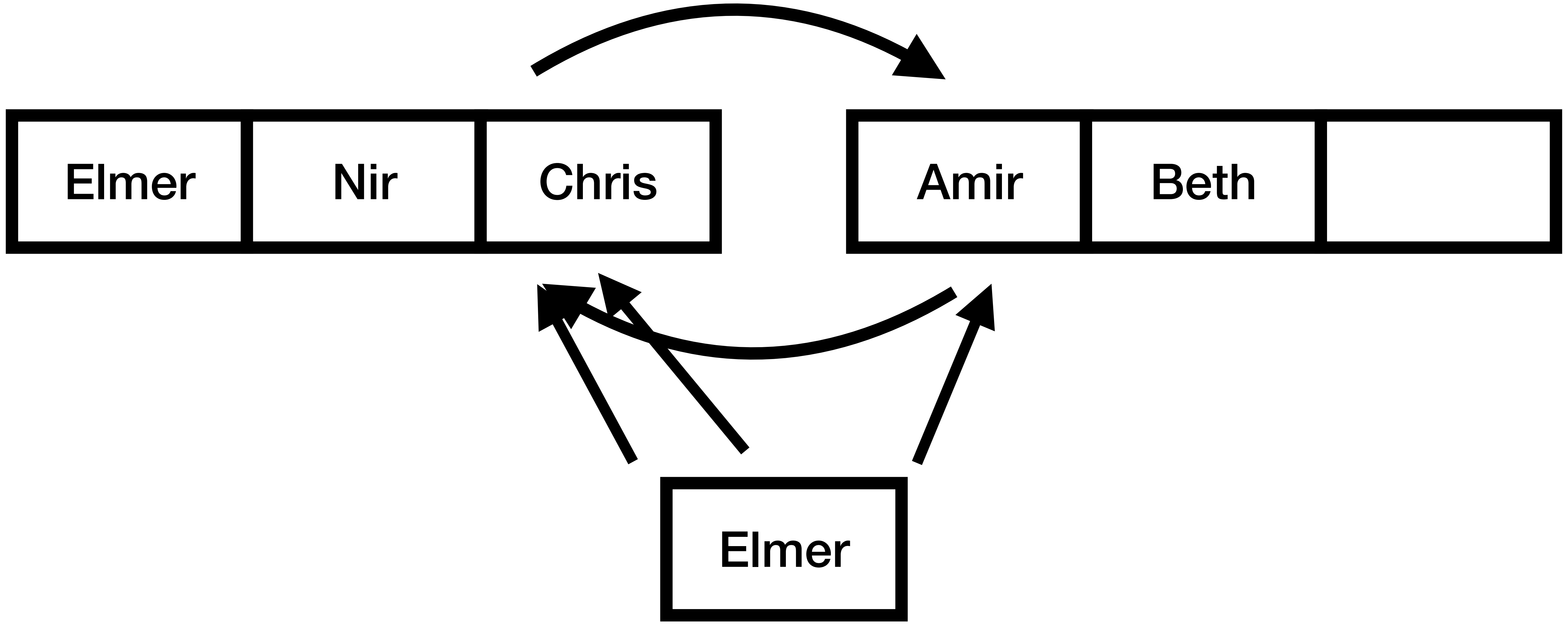
- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash

Elmer	Nir	Chris	Amir	Beth	

$$h_1(\text{Chris}) = 2, \;\; h_2(\text{Chris}) = 4$$

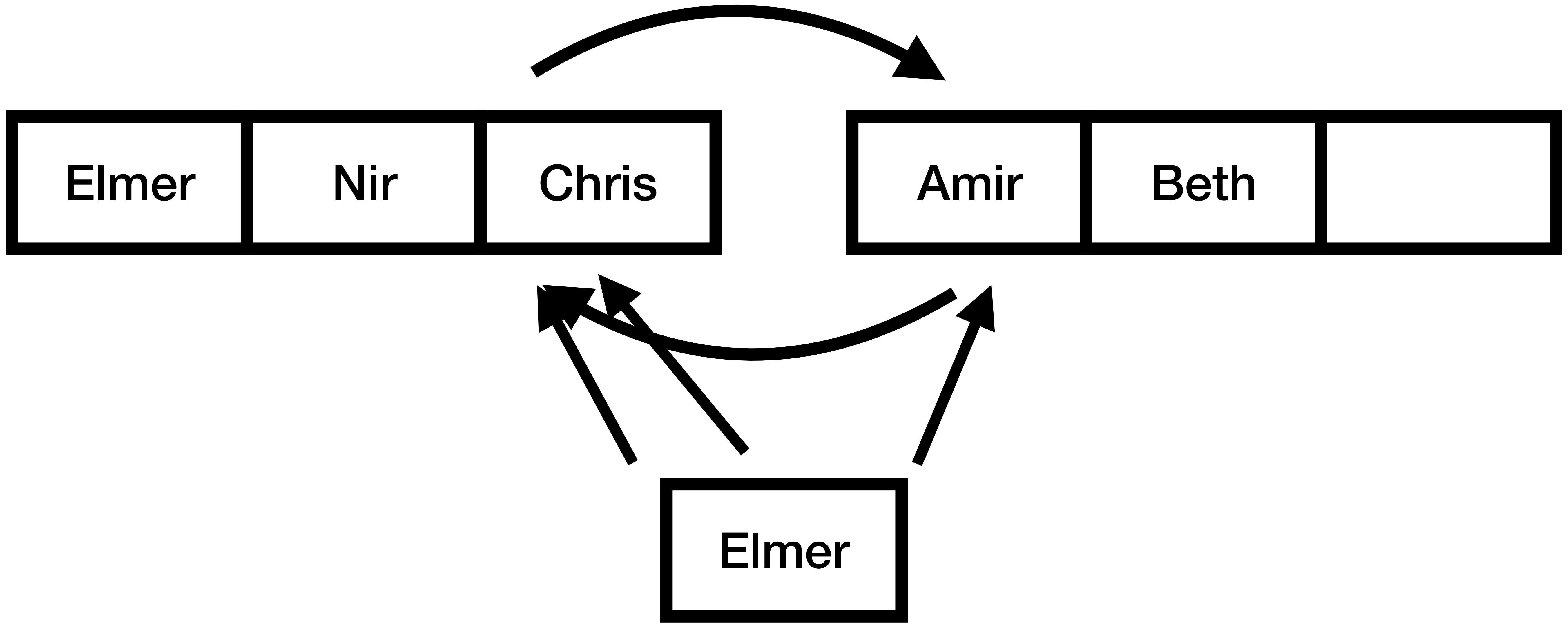
Cuckoo Hashing: Insert

- What can go wrong?
- This process may not end
- Example: 3 items hash to the same two slots
- What is the probability that this happens?
- $\binom{n}{2}\left(\frac{1}{n}\right)^4 = \Theta(1/n^2)$



Cuckoo Hashing: Insert

- More complicated analysis:
- Cuckoo hashing fails with probability $O(1/n^2)$
- What happens when we fail?
- Rebuild the whole hash table
- (Expensive worst-case insert operation)



Cuckoo Hashing: Insert

- How long does an insert take on average?
- One idea: each time we go to the other table, what is the probability the slot is empty?
- $1/2$. (This analysis isn't 100% right due to some subtle dependencies, but it's the right idea)
- So need two moves to find an empty slot in expectation
- At most $O(\log n)$ with high probability

Next class:
Approximation Algorithms

Acknowledgments

- Some of the material in these slides are taken from
    - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
    - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)