Hashing

Admin

- Welcome back
- Assignment 9 is out; intended to be done between Monday and Thursday
- Assignment 10 will be ungraded midterm review
- I think Zoom works now (can raise your physical hand, or your Zoom hand, to ask questions)
- Any questions?

Today

- What a hash function/hash table is from an algorithmic point of view
- A little bit about good hash functions

- Three kinds of hash table:
    - Chaining
    - Linear probing
    - Cuckoo Hashing

Hash table

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

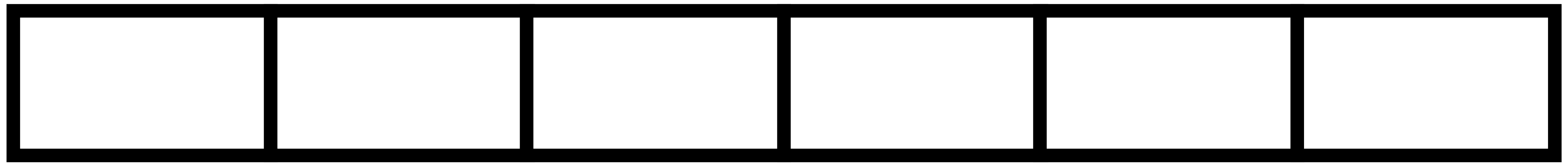
- $O(1)$ expected operations:
    - Insert a new item
    - Look up an item
    - Delete an item (we won't discuss)

- 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

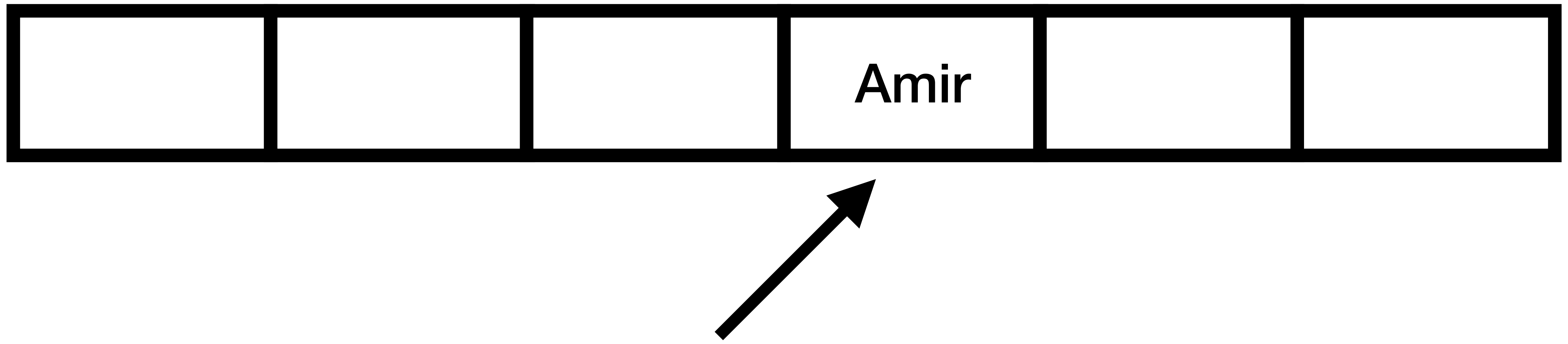
Hash table

- 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

Amir

Beth

Chris



$h(\text{Amir}) = 3$

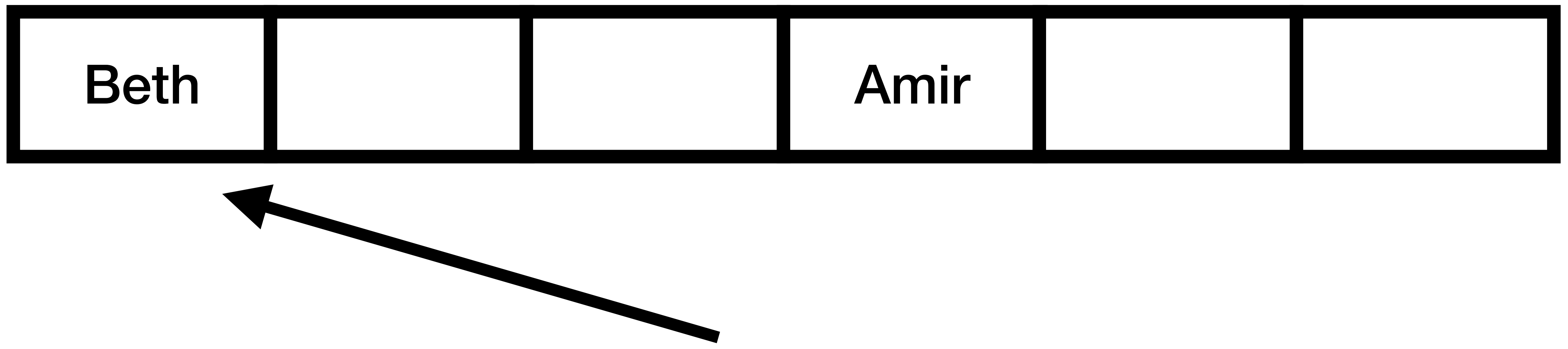
Hash table

- 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



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

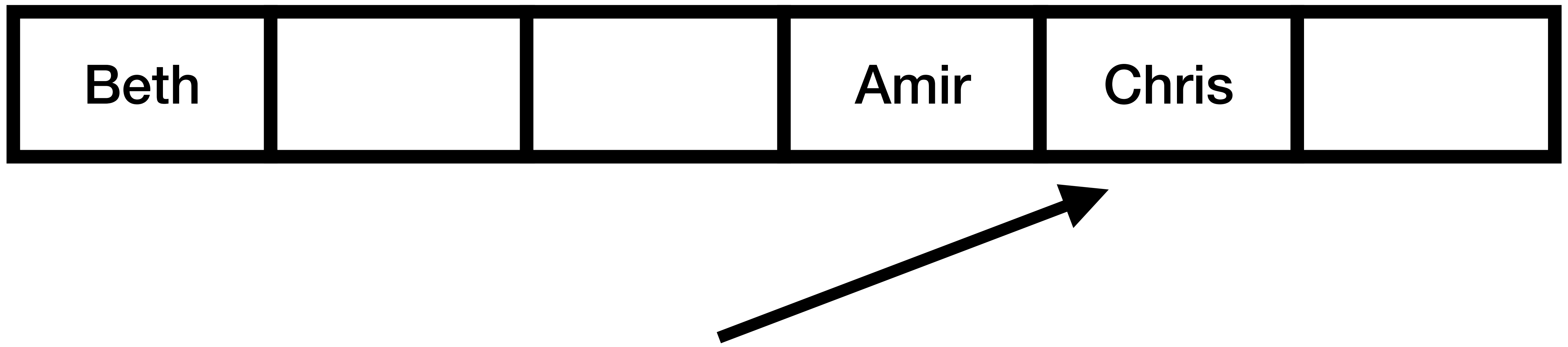
Hash table

- 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

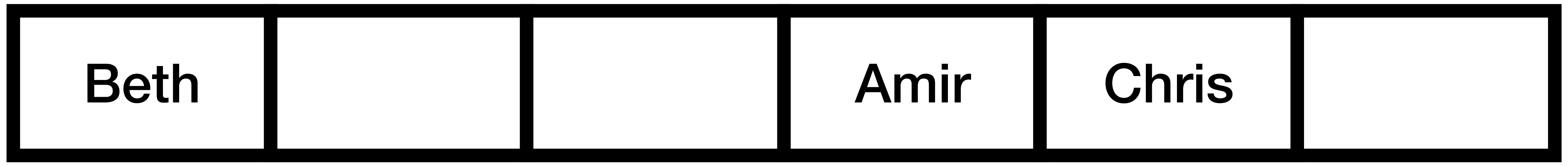
Chris



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

Hash function

- Goal: for any set of items, the hash function maps the items to different slots
- How can we guarantee 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

Beth			Amir	Chris	

- 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

Hash function: theory versus practice

- Some hash functions use a *seed*; same idea
- Our hash table performance guarantees were in expectation
- Our expectation is over the random choice of hash function

- Hashing: data is worst-case, hash function is random!

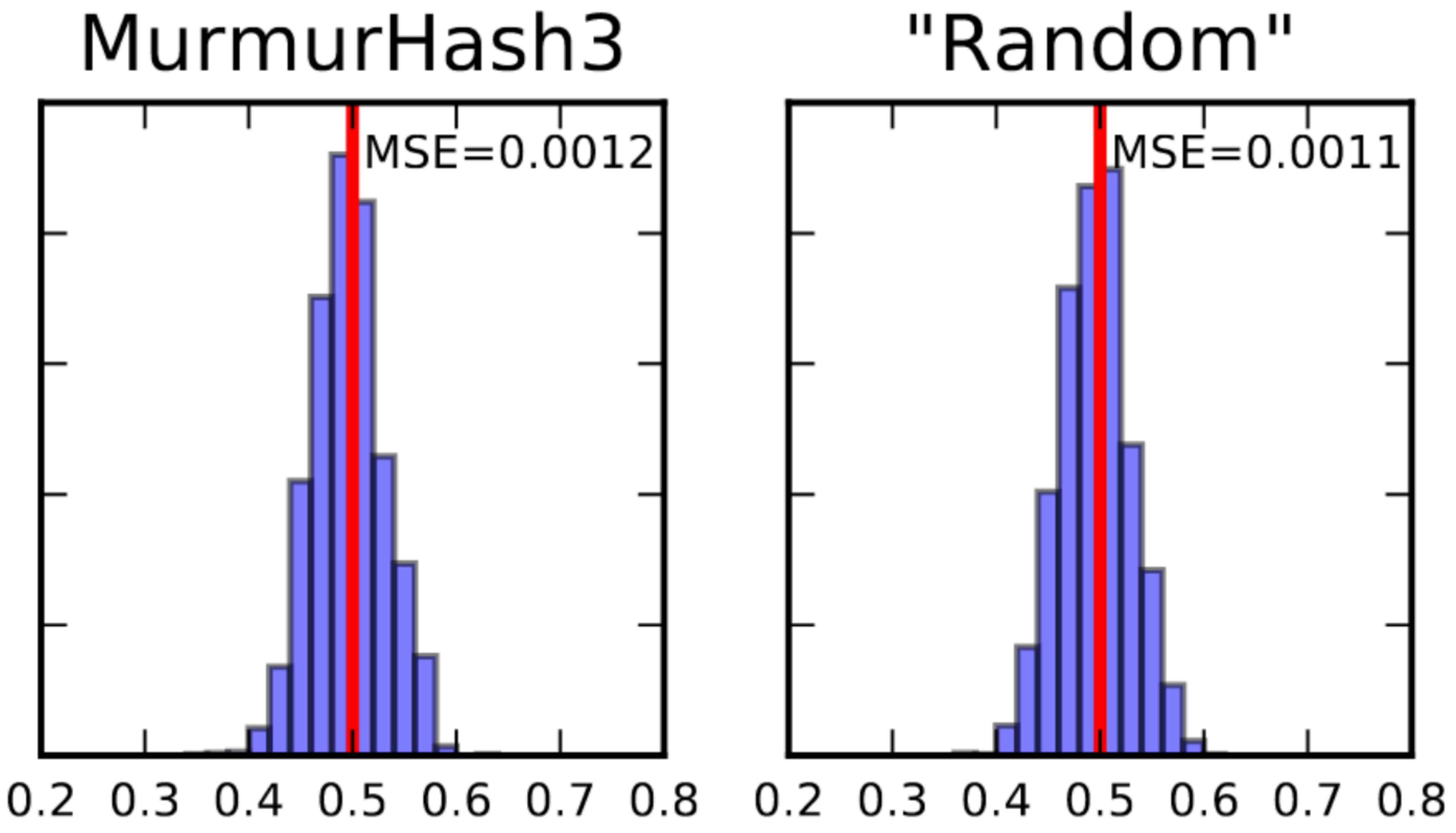
Hash function: theory versus practice

- Sometimes people use hashes that aren't random (Java and python hashes aren't random)
- That only works if your data is "spread out" — there are many datasets on which Java hashing does poorly
- In fact, for integers of ≤ 32 bits, Java uses $h(i) = i$

Hash function: theory versus practice

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

Dahlgaard et al. 2017



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

Hash Tables and Performance

Goal

- The only problem is what to do when multiple items happen to share the same hash
- What can we do about that?
- Assuming our hash functions are ideal, what is the resulting performance?

Chaining

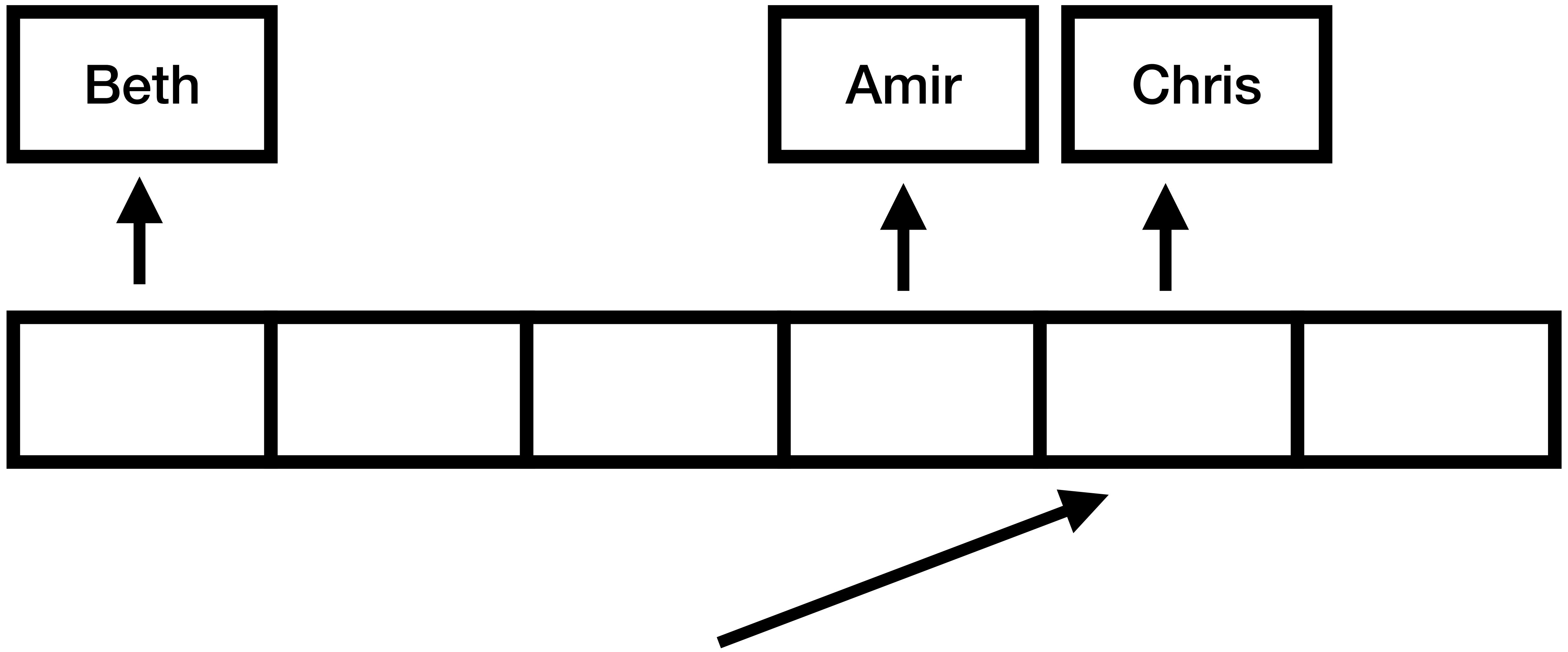
- Store a linked list at each array entry
- When an item hashes to a slot, prepend it to the linked list

Amir

Beth

Chris

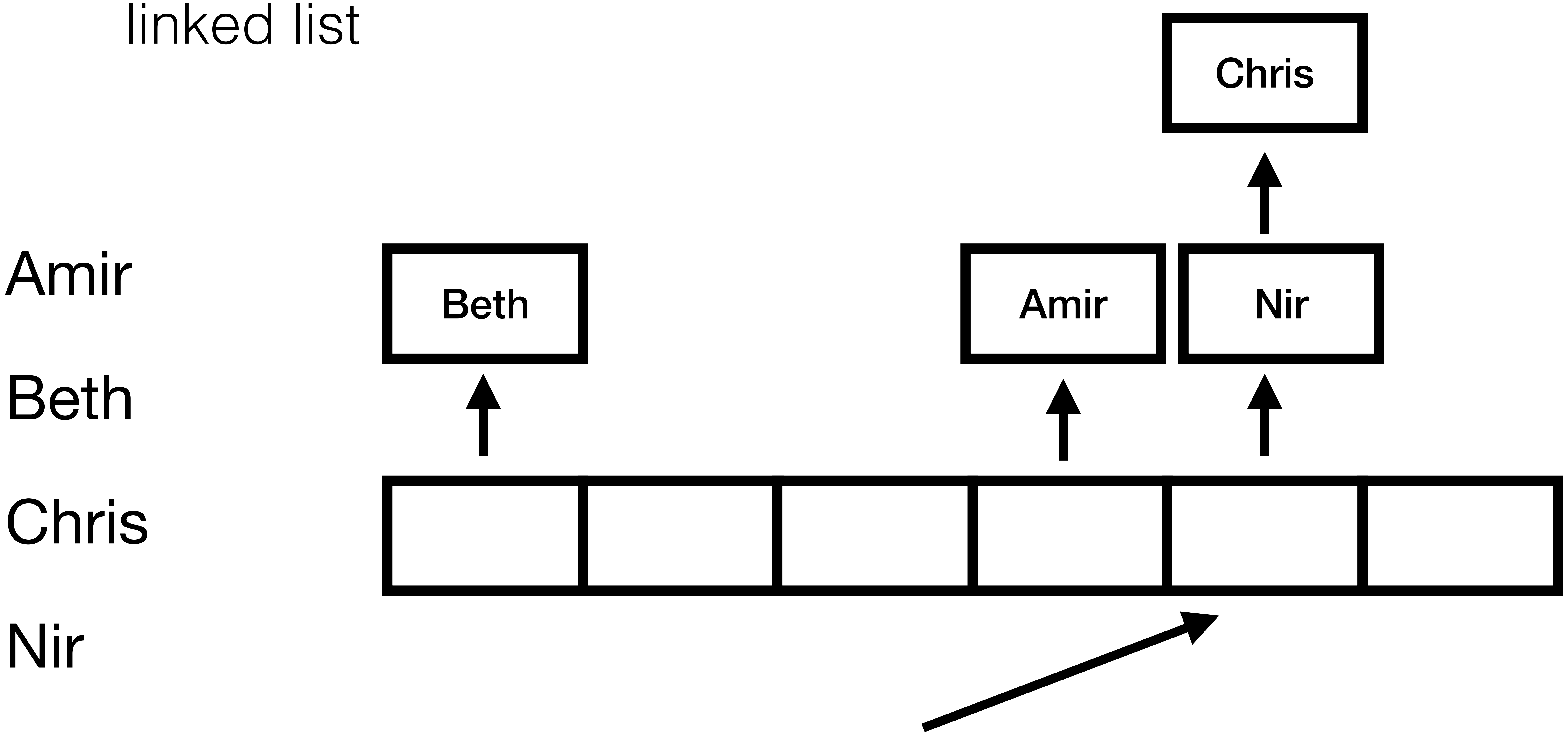
Nir



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

Chaining

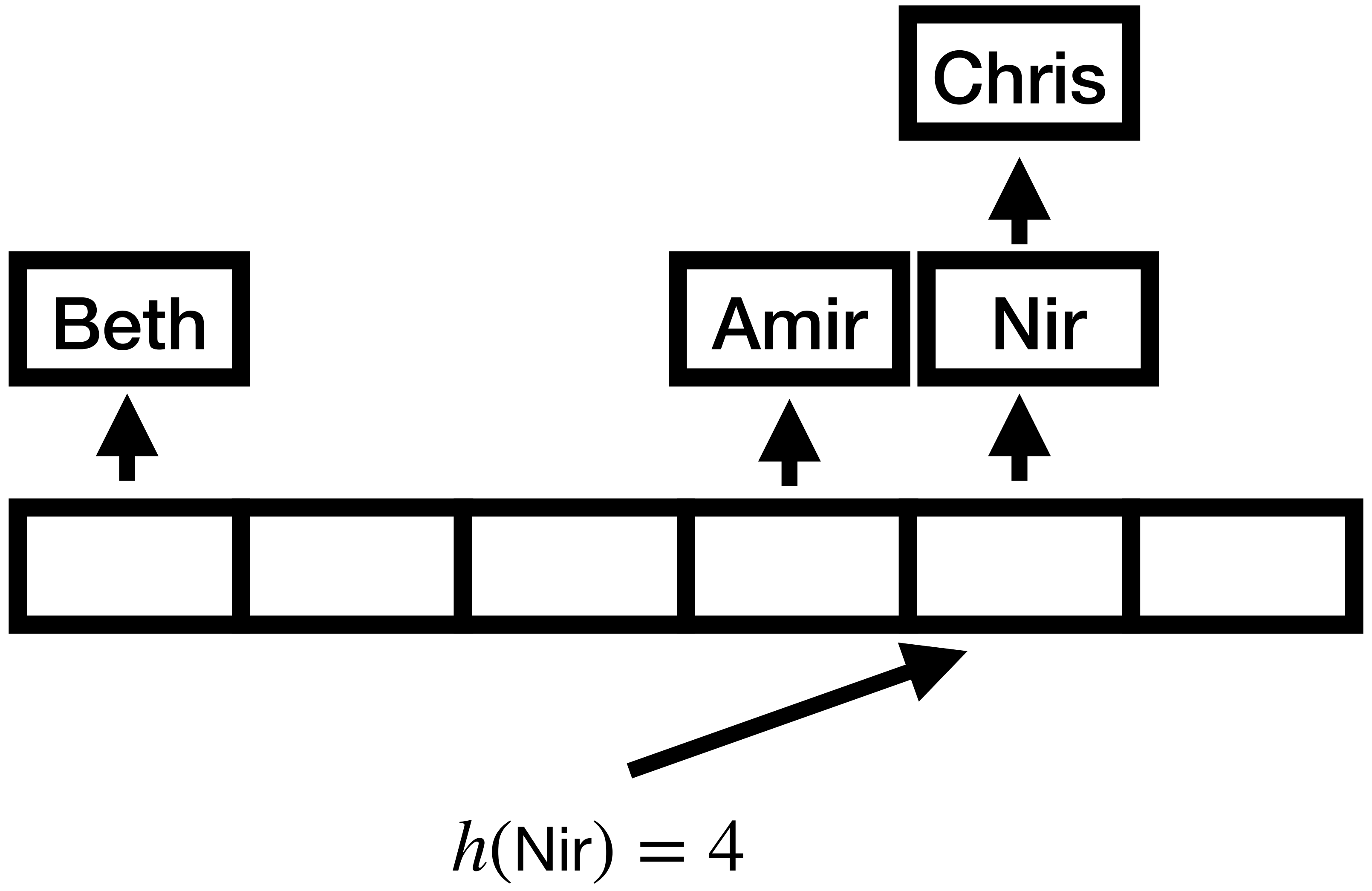
- Store a linked list at each array entry
- When an item hashes to a slot, prepend it to the linked list



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

Chaining

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

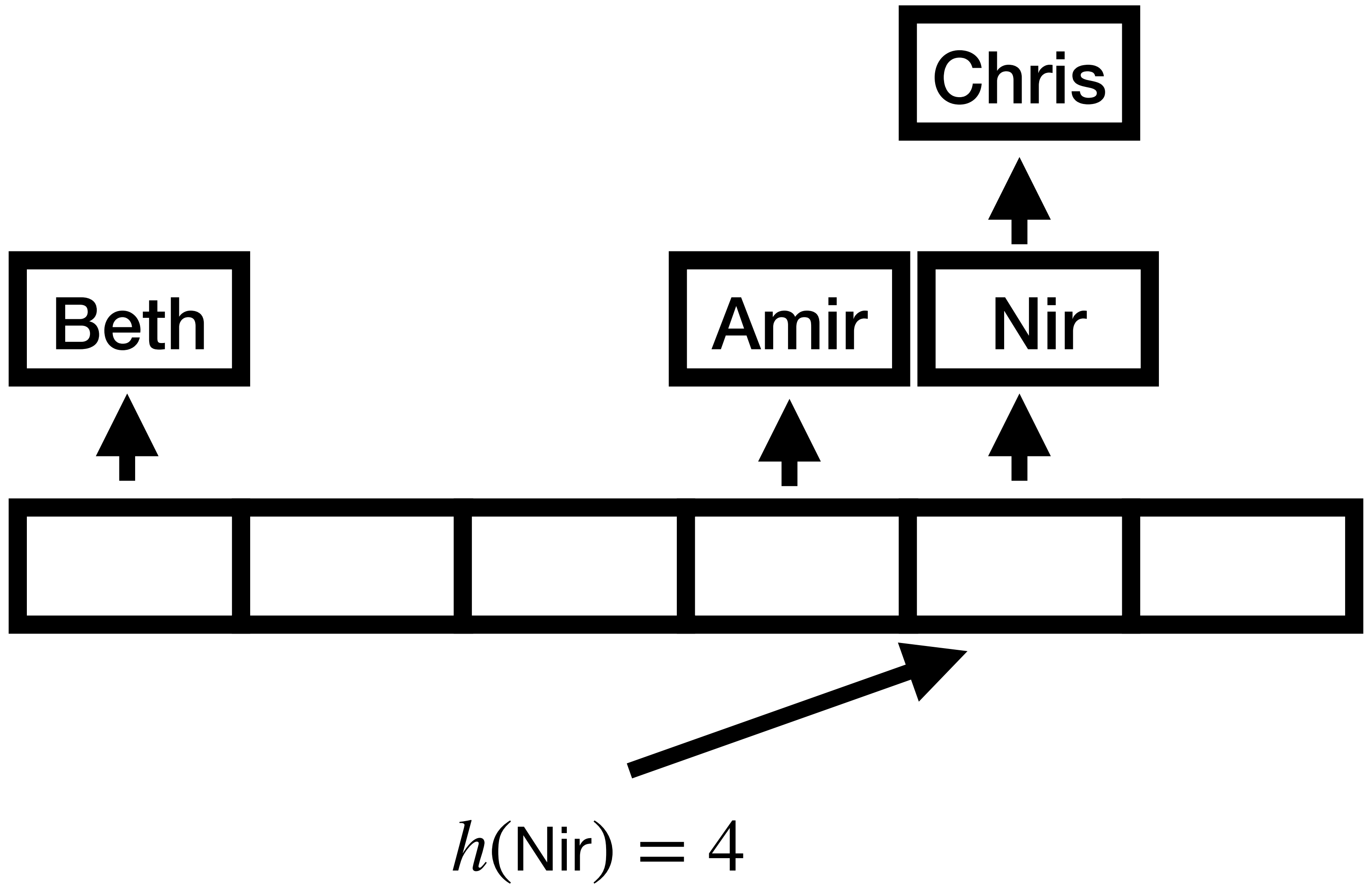


- How can we insert?

- How can we lookup?

- How much time does insert/lookup take?

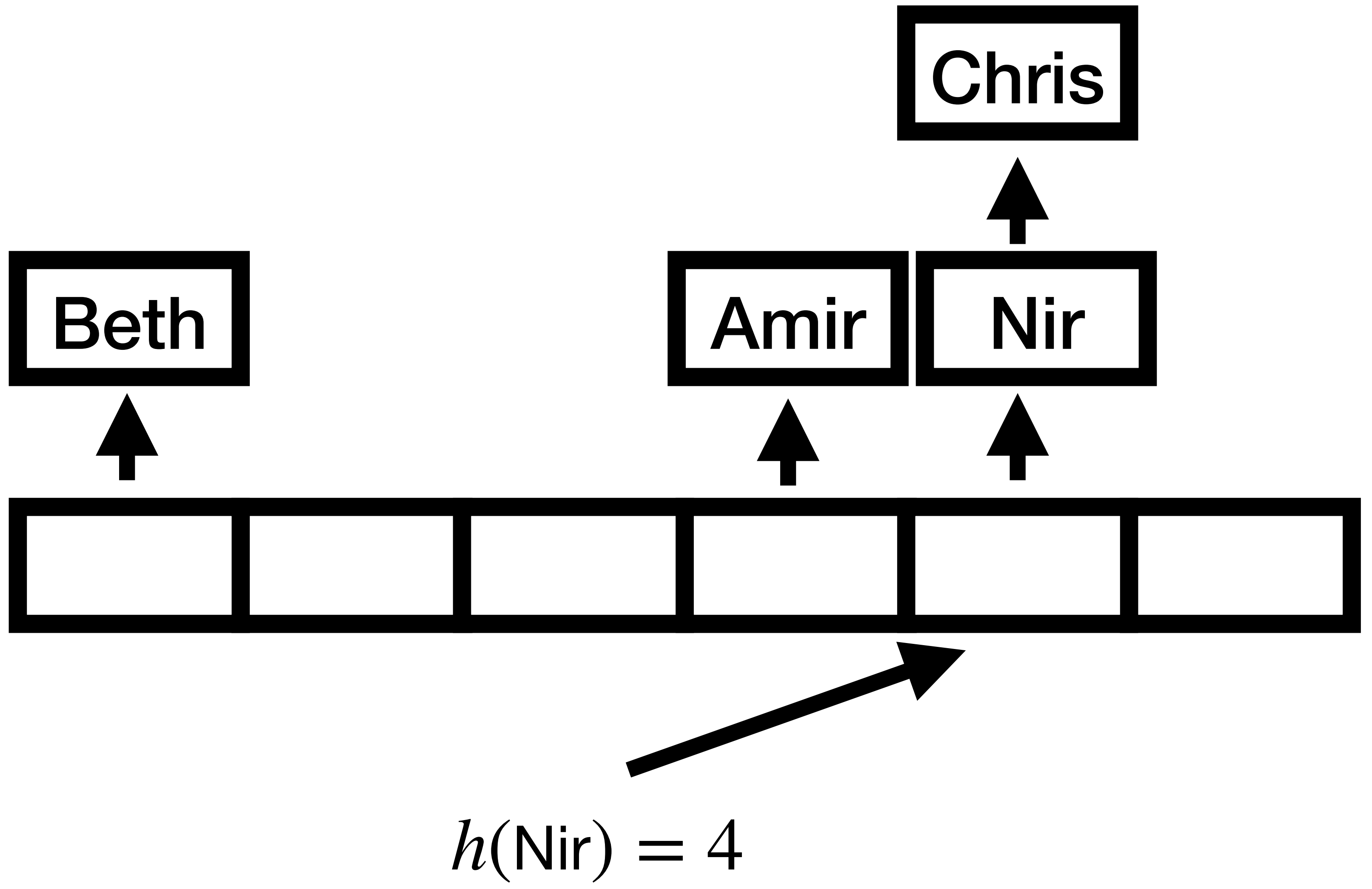
Chaining: Analysis



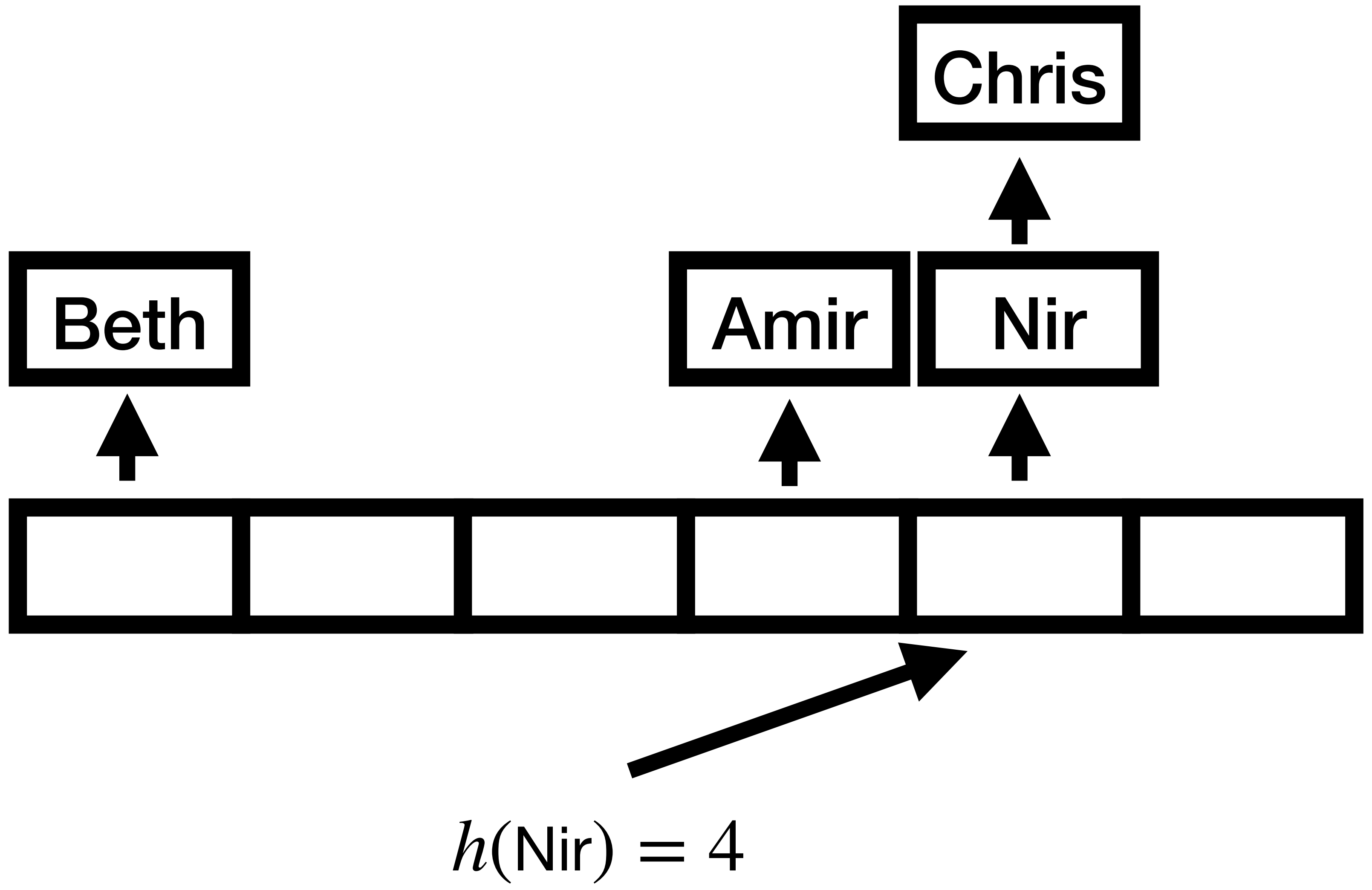
- What is the expected lookup time?
    - You'll do on Assignment 9!
- That's just average. How long can the chains get?
- Let's show: $O(\log n / \log \log n)$ with high probability, even if $m = n$
- (That is to say, with probability $\geq 1 - 1/n^3$)

Chaining: W.h.p. Analysis

- What is the probability that at least k items hash to a given slot?
- Pick k items; each must hash to this slot
- $\binom{n}{k}\left(\frac{1}{n}\right)^k \le \left(\frac{en}{k}\right)^k \left(\frac{1}{n}\right)^k = \left(\frac{e}{k}\right)^k$



Chaining: W.h.p. Analysis



- Substituting $k = 4\ln n/\ln\ln n$,
- Probability that the bin has at least $4\ln n/\ln\ln n$ balls is at most:
- $\left(\frac{e\ln\ln n}{4\ln n}\right)^{4\ln n/\ln\ln n} \leq e^{\frac{4\ln n}{\ln\ln n}\ln\frac{e\ln\ln n}{4\ln n}} \leq e^{\frac{4\ln n}{\ln\ln n}(\ln\ln\ln n - \ln\ln n)} \leq$
- $\leq e^{\ln n - 4\ln n} = e^{-3\ln n} = 1/n^3$
- Can extend to higher powers of $1/n$ by increasing k by a constant factor

Chaining: Some other questions

- 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?

- Random variable $X_i = 1$ if slot i has exactly one item, 0 otherwise
- By linearity of expectation, we want

$$\sum_{i=1}^{n} \Pr\left[\text{slot } i \text{ has one item}\right]$$

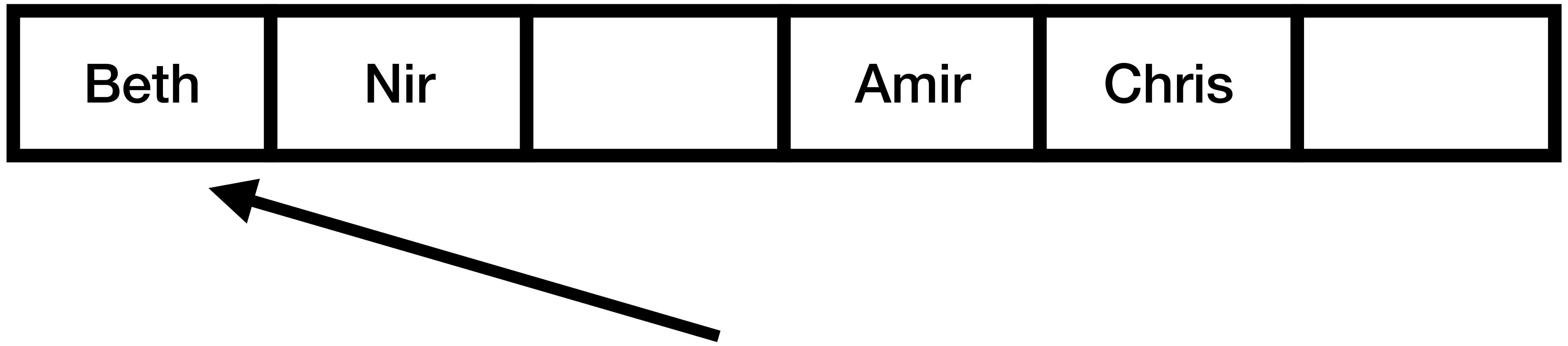
Chaining: Some other questions

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

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

Linear Probing

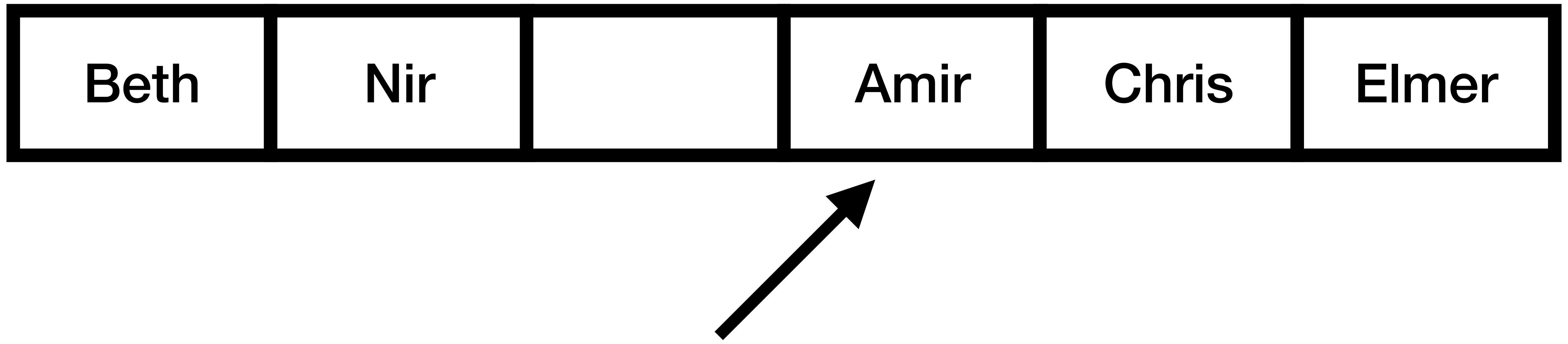
- No linked lists; just the table
- If there is already an item in $A[h(i)]$, check $A[h(i) + 1]$, then $A[i + 2]$, and so on



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

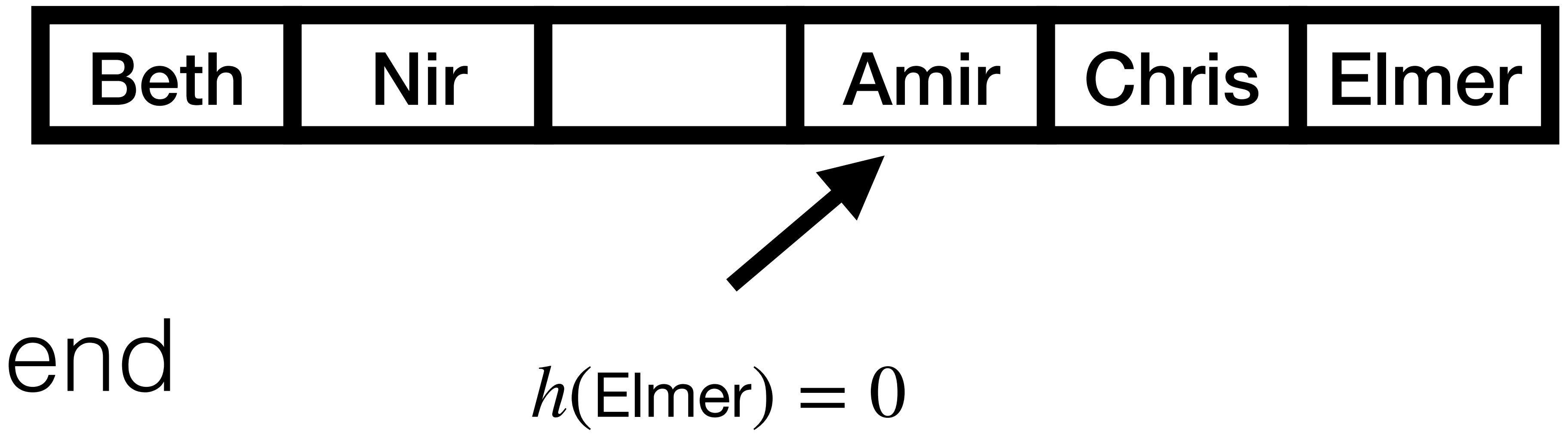
Linear Probing

- 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?



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

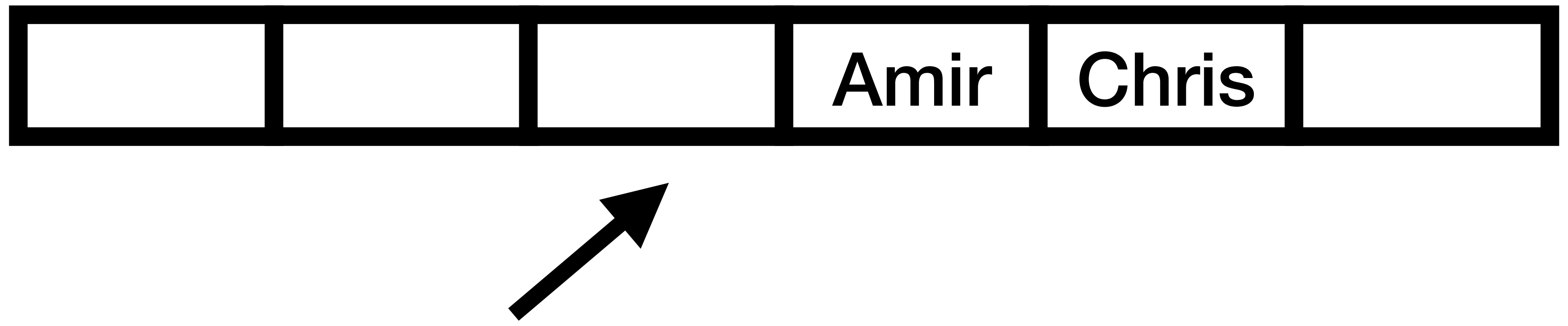
Linear Probing



- 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:
- $O\left(1 + 1/(1 - n/m)^2\right)$

Linear Probing: w.h.p. Analysis



- All operations are $O(\log n)$ w.h.p.

- Here's a sketch of why this is the case:
- What is the probability that, given that this slot is empty, the next $8 \log n$ slots are full?
- Must have exactly $8 \log n$ elements hashing to those $8 \log n$ slots
- Probability:

$$\binom{n}{8 \log n}\left(\frac{8 \log n}{m}\right)^{8 \log n}\left(1-\frac{8 \log n}{m}\right)^{n-8 \log n} \leq \left(\frac{ne}{8 \log n}\right)^{8 \log n}\left(\frac{8 \log n}{m}\right)^{8 \log n}\left(e^{\frac{-8 \log n}{m}}\right)^{n-8 \log n}$$

- $\leq (9/10)^{8 \log n} \leq 1/n^2$ so long as $\dfrac{n}{m} e^{1+(8 \log n)/m-n/m} = \dfrac{e^{.5+(8 \log n)/m}}{2} \leq 9/10$

Linear Probing vs Chaining?

- What are some advantages of chaining?
    - Simple (?)
    - 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 the Bounds

- We need randomness in order to hash

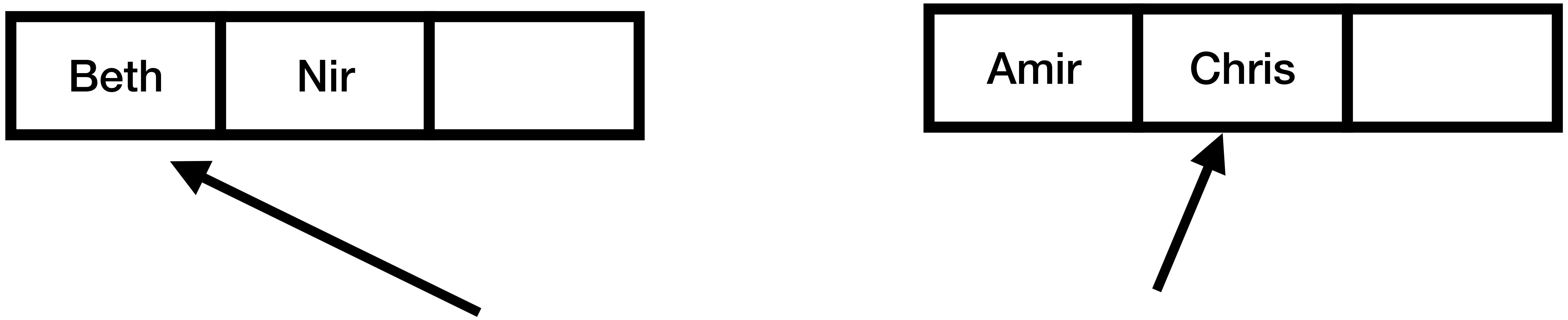
- But can we get worst-case bounds?

- 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

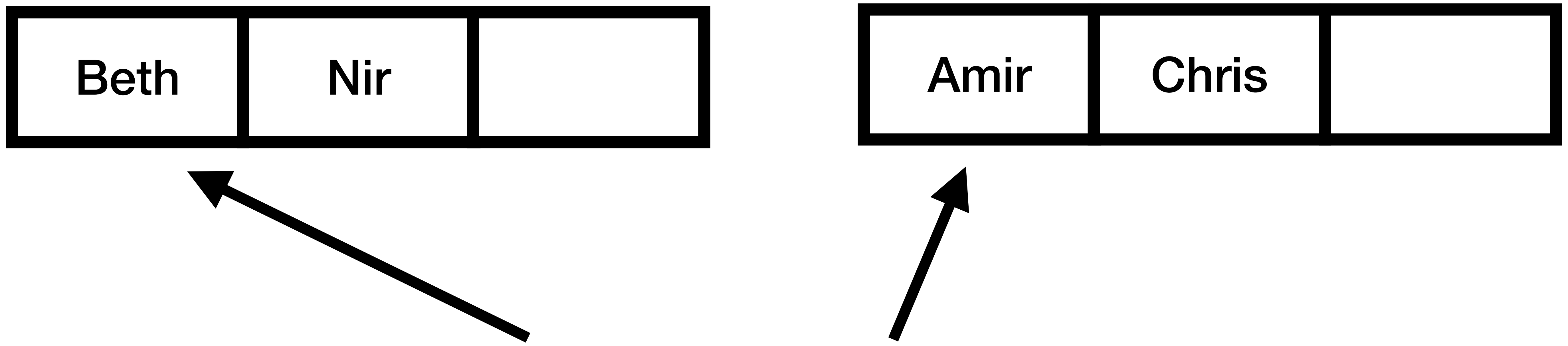
- Uses two hash functions, h_1 and h_2, two hash tables
- Each table size n
- Item i is guaranteed to be in $A[h_1(i)]$ or $A[h_2(i)]$
- So we can lookup in $O(1)$
- How can we insert?



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

Cuckoo Hashing: Insert

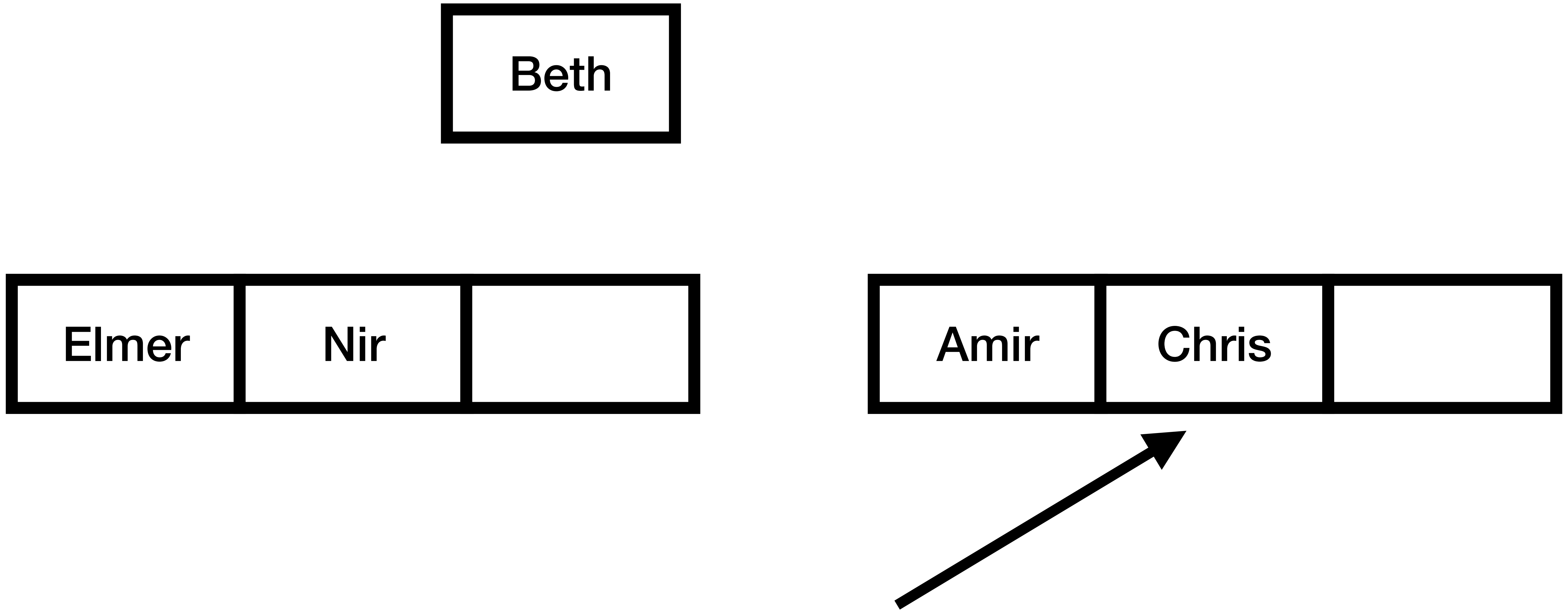
- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash



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

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

Chris

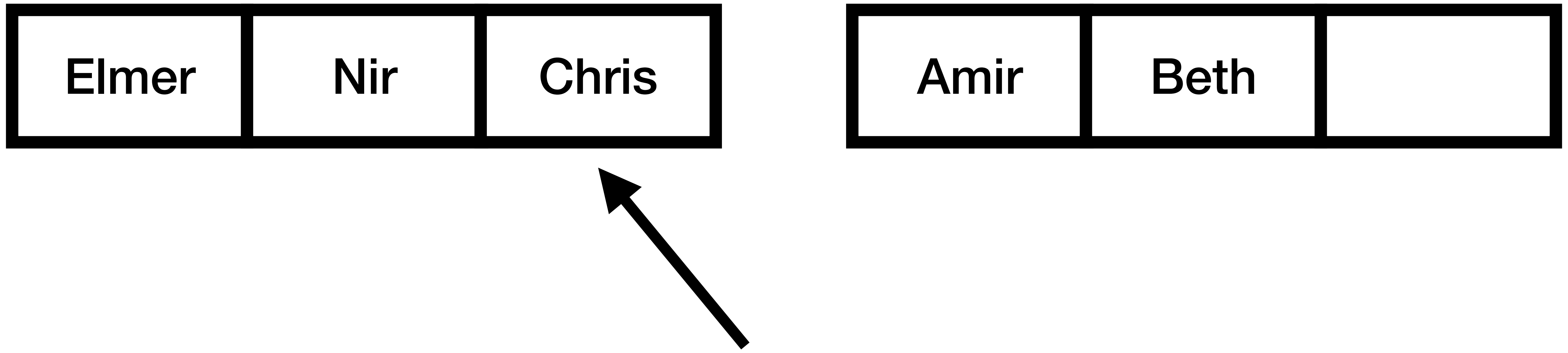
Elmer	Nir	

Amir	Beth	

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

Cuckoo Hashing: Insert

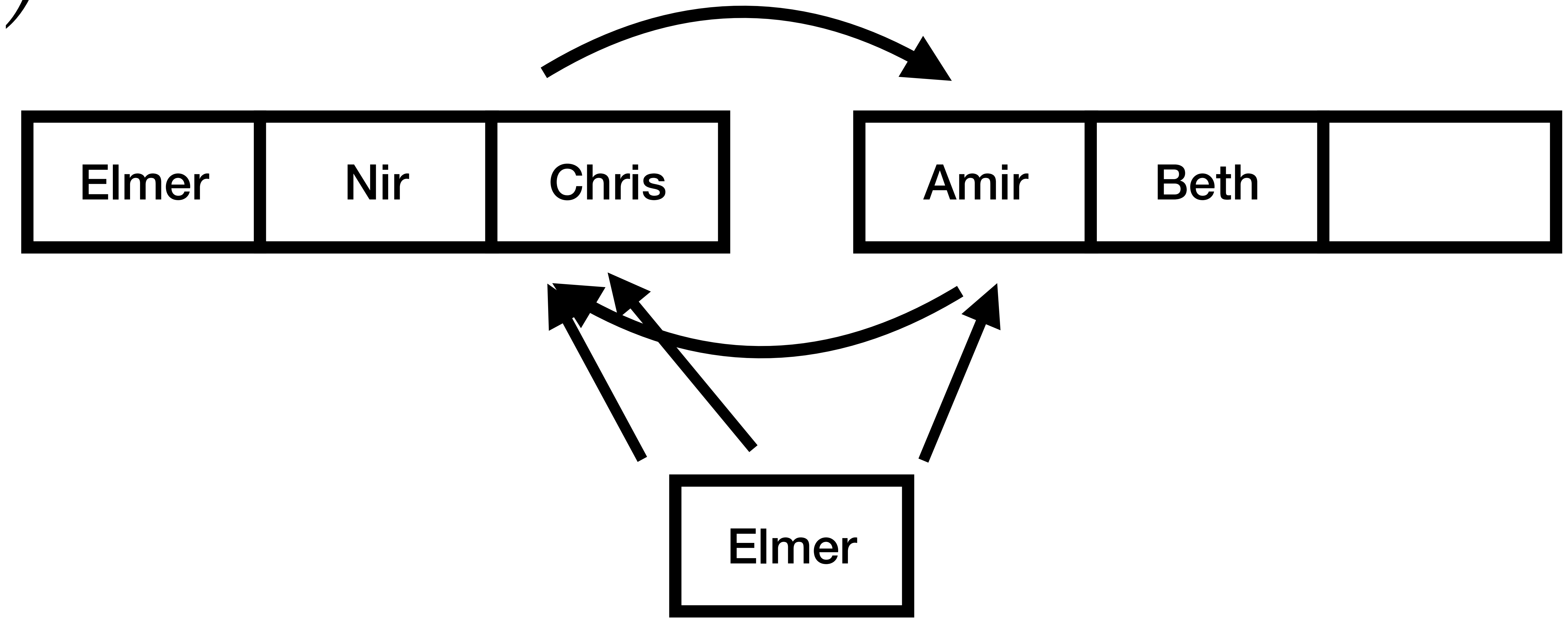
- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash



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

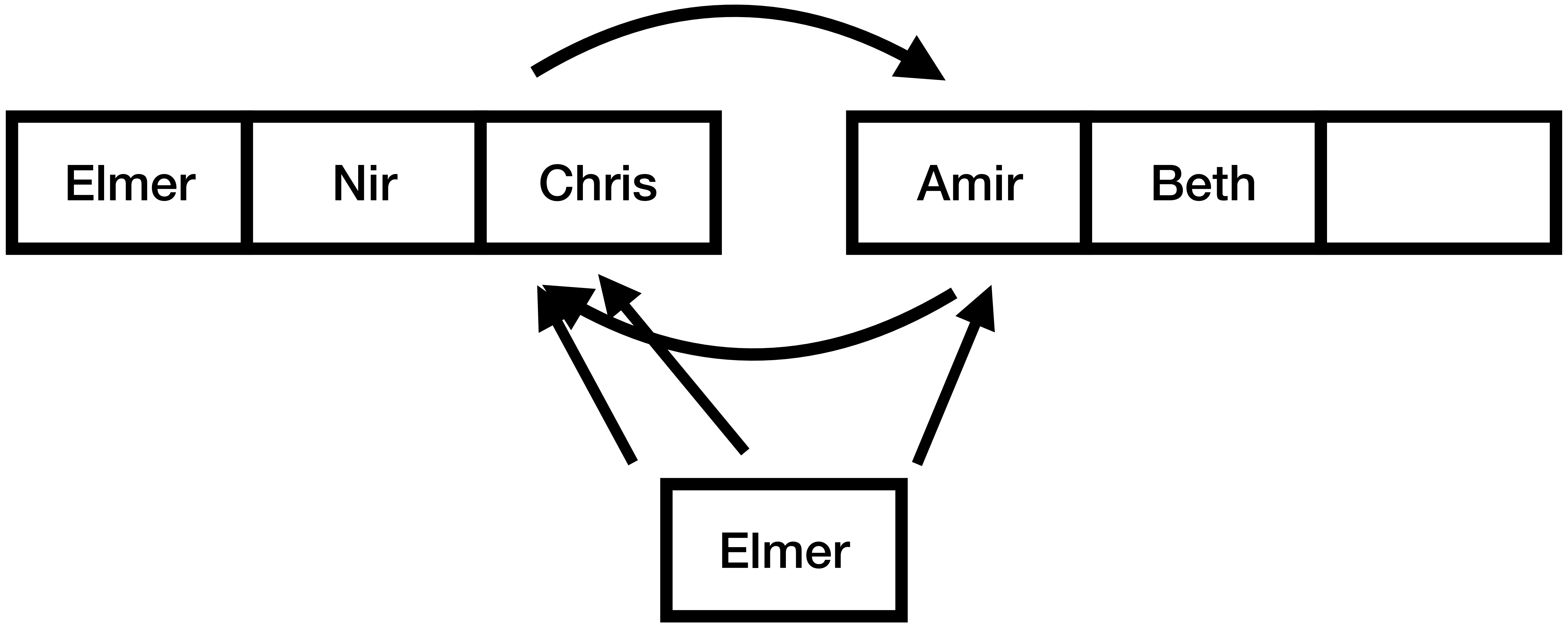
Cuckoo Hashing: Insert

- 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?
- $n\binom{n}{3}\left(\frac{1}{n}\right)^6 = \Theta(1/n^2)$



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)
  - MIT course notes, 6.042/18.062J Mathematics for Computer Science April 26, 2005, Devadas and Lehman