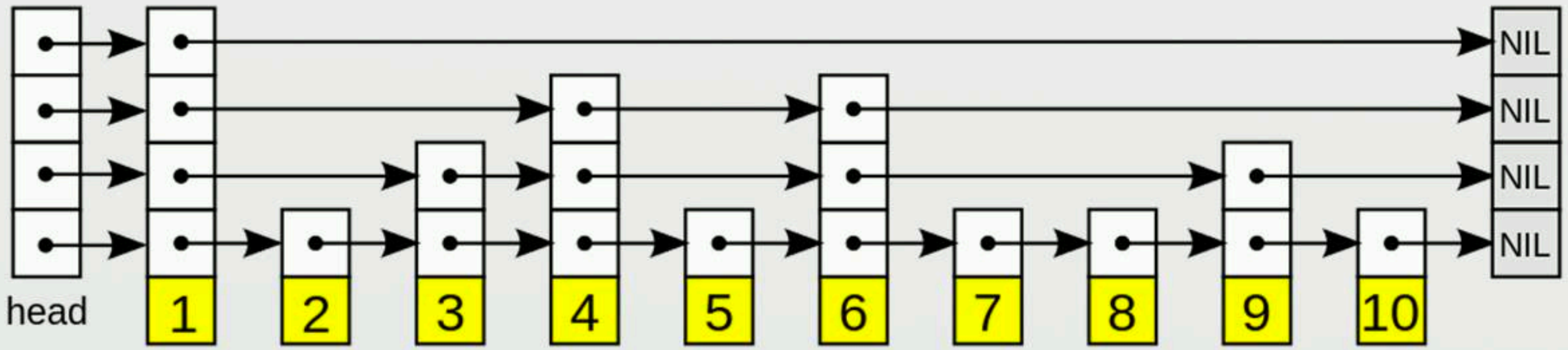
Fun with Randomness:
Skip Lists and Cuckoo Hashing

Admin

- **Course evaluations** in class on Wed (last lecture)
    - Bring your laptops to class!
    - We will end early: ~15-20 minutes left for filling out evals
- Review hours for final:
    - 2-3.30 pm Monday, **7-9 pm Tuesday**, 1.30-3 pm Wednesday
    - Ask questions in general or about practice problems (HW 10)
- **Final**. 24-hour final, will be available on Gradescope from **Thursday May 20 8.30 am** and must be submitted by **May 28, 8.30 pm**
- **Honor Code (final).** Can only refer to course materials (notes/book/GLOW), honor code violation to use external resources, google terms, or discuss exam with others

Skip list



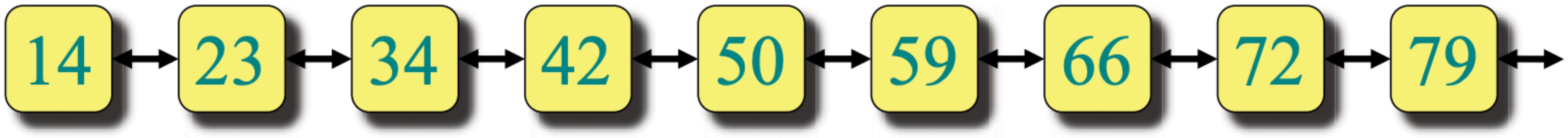
https://en.wikipedia.org/wiki/File:Skip_list.svg

Skip Lists: Randomized Search Trees

- Invented around 1990 by Bill Pugh
- Idea: binary search trees are a pain to implement
- Skip lists balance randomly; no rules to remember, no rebalancing
- Build out of simple structure: sorted linked lists
- Inserts, deletes, search, predecessor, successor are all $O(\log n)$ with high probability
- No rebalancing makes them useful in concurrent programming
    - E.g, lock-free data structures

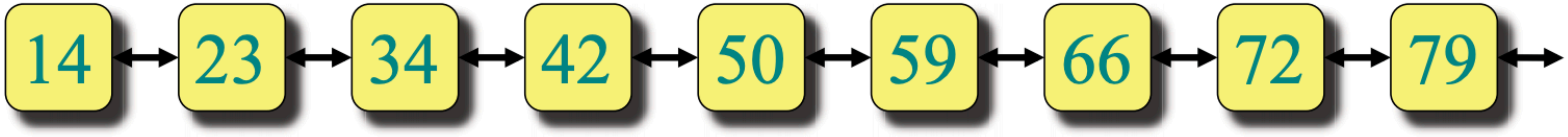
One Linked List

- Start from simplest data structure: (sorted) linked list
- Search cost?
  - $\Theta(n)$
- How can we improve it?



Two Linked Lists

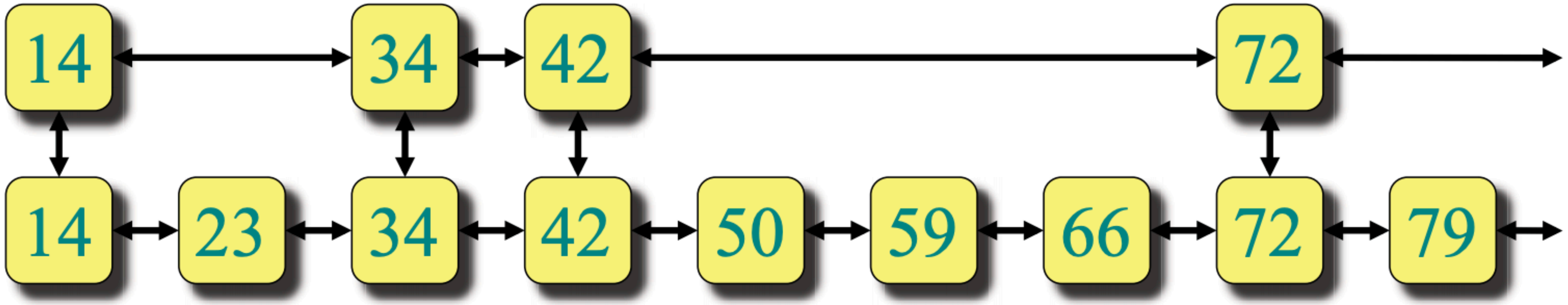
- Suppose you had *two* sorted linked list
  - List can contain subset of elements
- Each element can appear in one or both lists
- **Class exercise.** How can you use two lists to speed up searches?



NYC Subway System

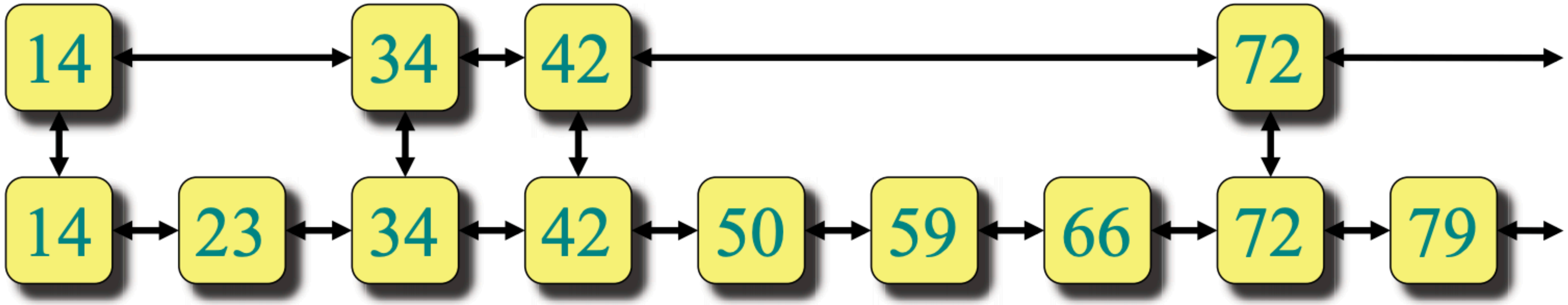
Two Linked Lists

- **Idea**: express and local subways
- Express lines connects a few main stations (and skips a bunch)
- Local line connects all stations but is slow
- Links between local and express line so you can switch



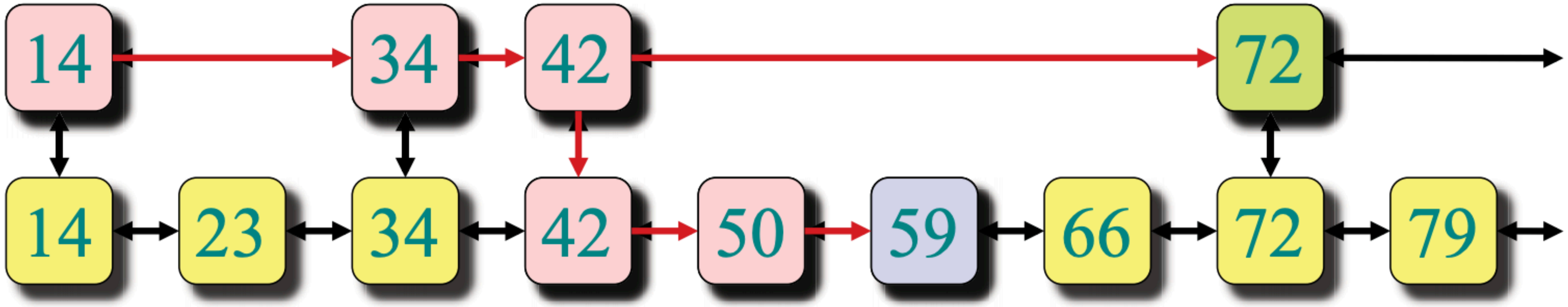
Two Linked Lists

- **Search**(x):
  - Walk right in top linked list L_1 until going right would be too far
  - Walk down to bottom linked list L_2
  - Walk right in L_2 until x is found or reach end (report not found)



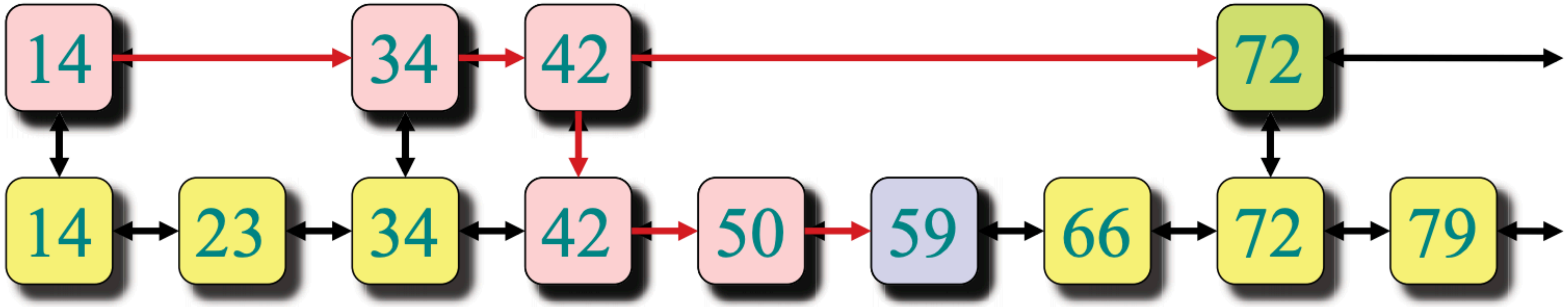
Two Linked Lists

- **Search(59)**:
  - Walk right in top linked list L_1 until going right would be too far
  - Walk down to bottom linked list L_2
  - Walk right in L_2 until x is found or reach end (report not found)



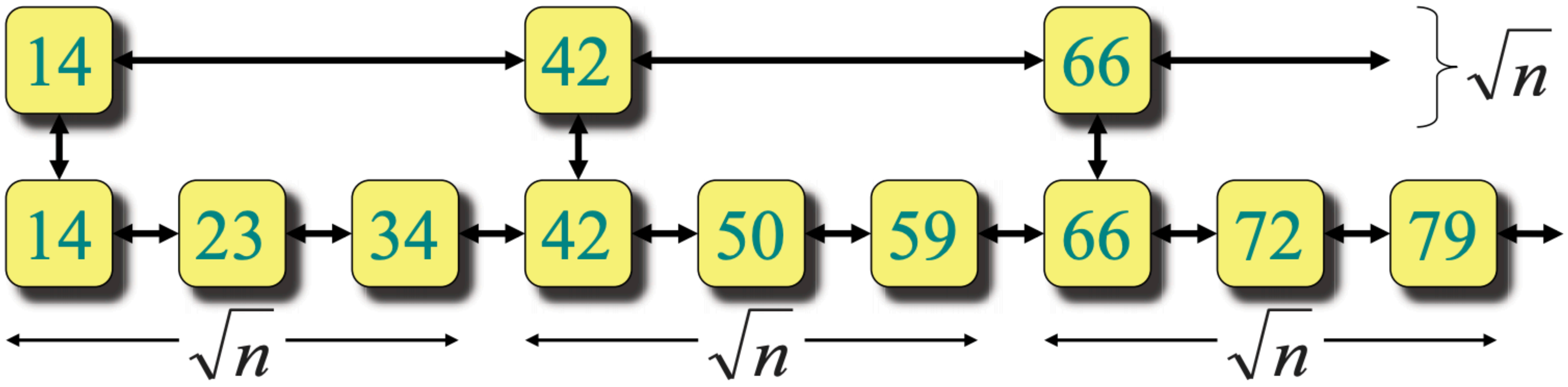
Two Linked Lists

- How should we organize the two lists?
  - Which nodes go in L_2?
  - How much gap to leave between elements?
  - **Best approach**: evenly space and promote elements



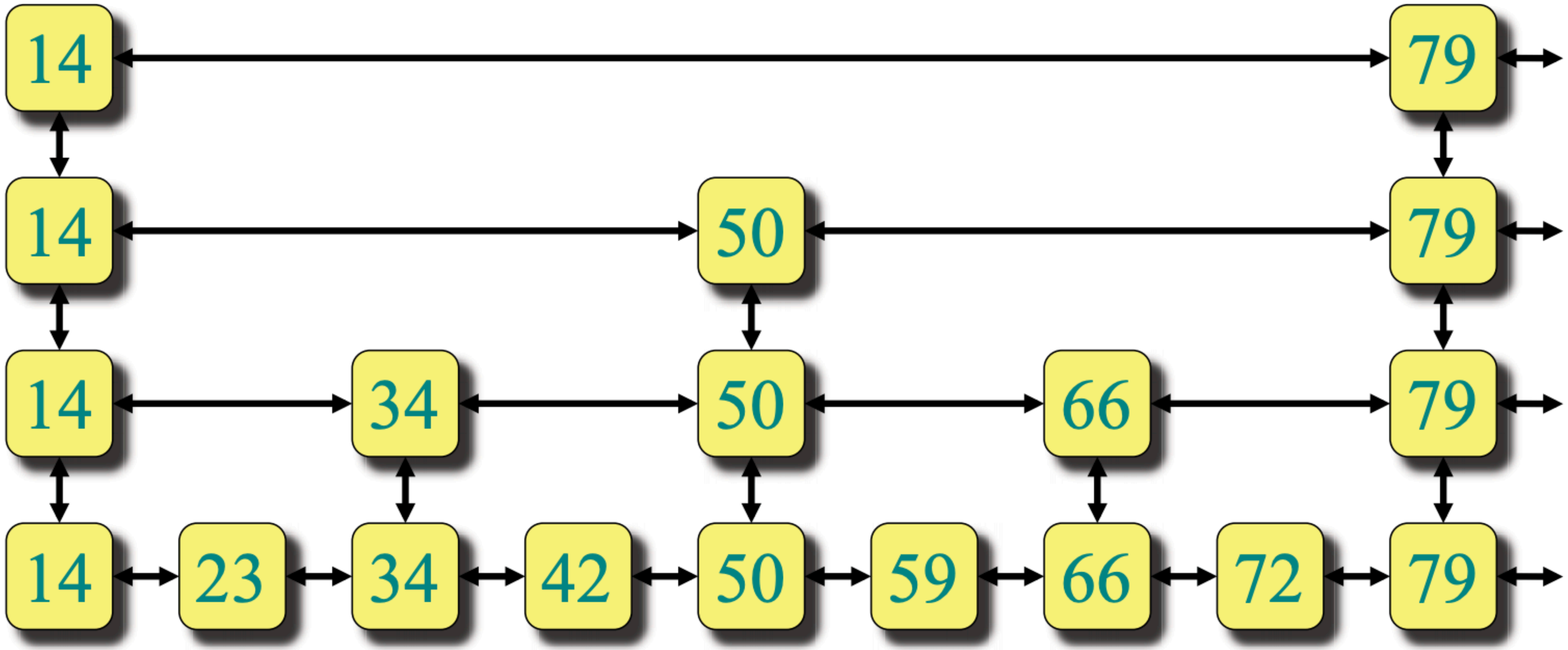
Two Linked Lists

- If gap between elements in top list is g, then the number of elements traversed (search cost) is at most $g + n/g$
- Optimized by setting $g = \sqrt{n}$
- So the search cost is at most $2\sqrt{n}$



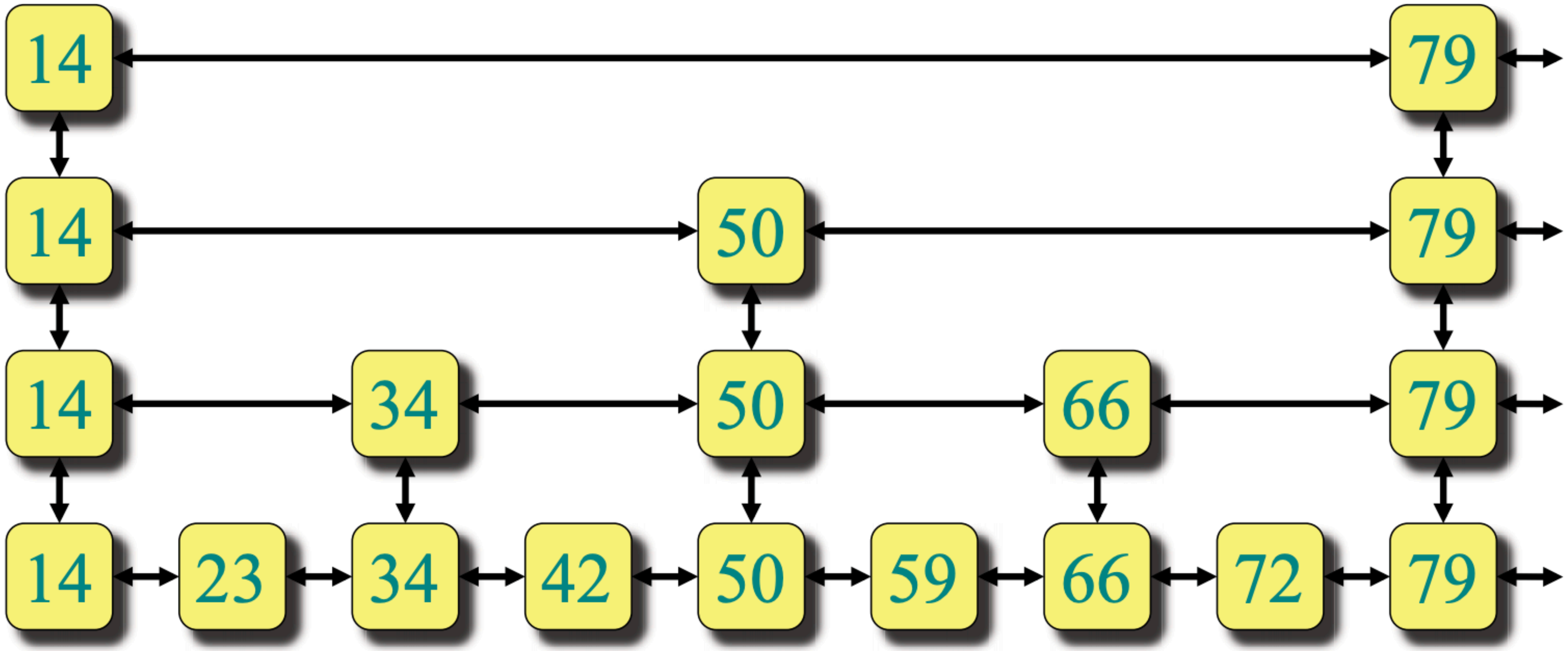
More Linked Lists

- Search cost with two linked list: $2\sqrt{n}$
- Search cost with three linked list: $3n^{1/3}$



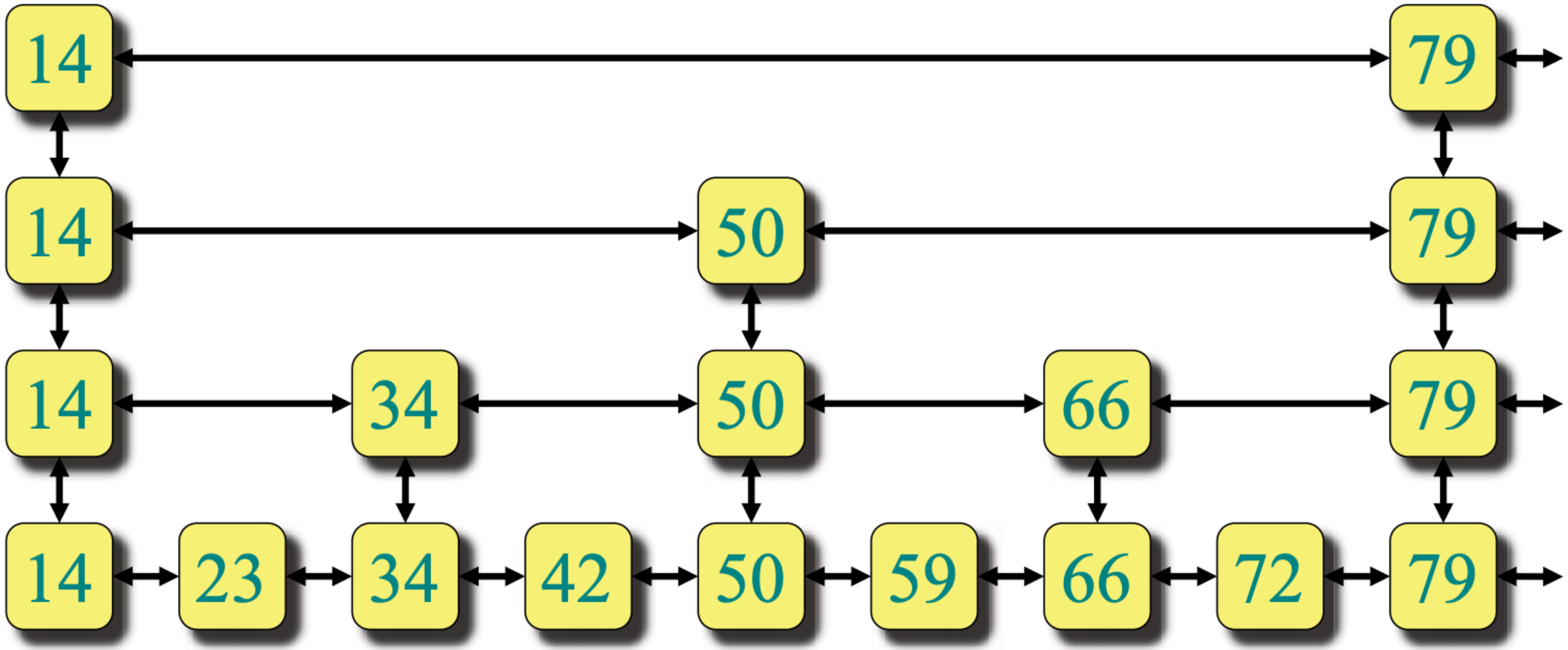
k Linked Lists

- Search cost with k linked lists: $kn^{1/k}$
- Search cost with $\log n$ linked lists: $\log n \cdot n^{1/\log n}$
  - $\log n \cdot n^{1/\log n} = 2 \log n$



Randomize the lists: Skip Lists

- This is good, but how can we insert?
- Every new element disrupts our spacing
- Idea: use randomness!



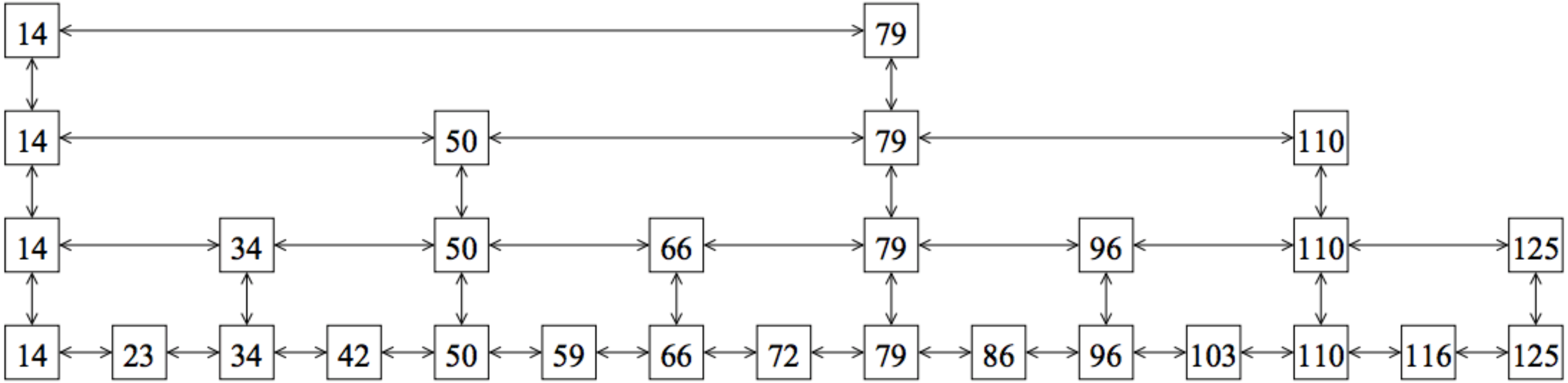
Skip List Details

- Insert(x)
    - Search bottommost list for x's position and insert it there
    - **Invariant**: Bottommost list contains all elements
    - Which other lists should a new item be added to?
    - Insert x at level 1 and flip a coin (idea we want half of the elements to go next level, similar to a balanced binary tree)
    - If heads: element gets promoted to next level, and we repeat
    - If tails element stays put at current level and we are done

Skip List Details

- Thus, on average
  - $1/2$ of the elements go up 1 level
  - $1/4$ of the elements go up 2 levels
  - $1/8$ go up to 3 levels, etc.
- Search(x):
  - Start at top list, go right just before value gets $>$ target
  - Go down and repeat until element is found or hit bottom right

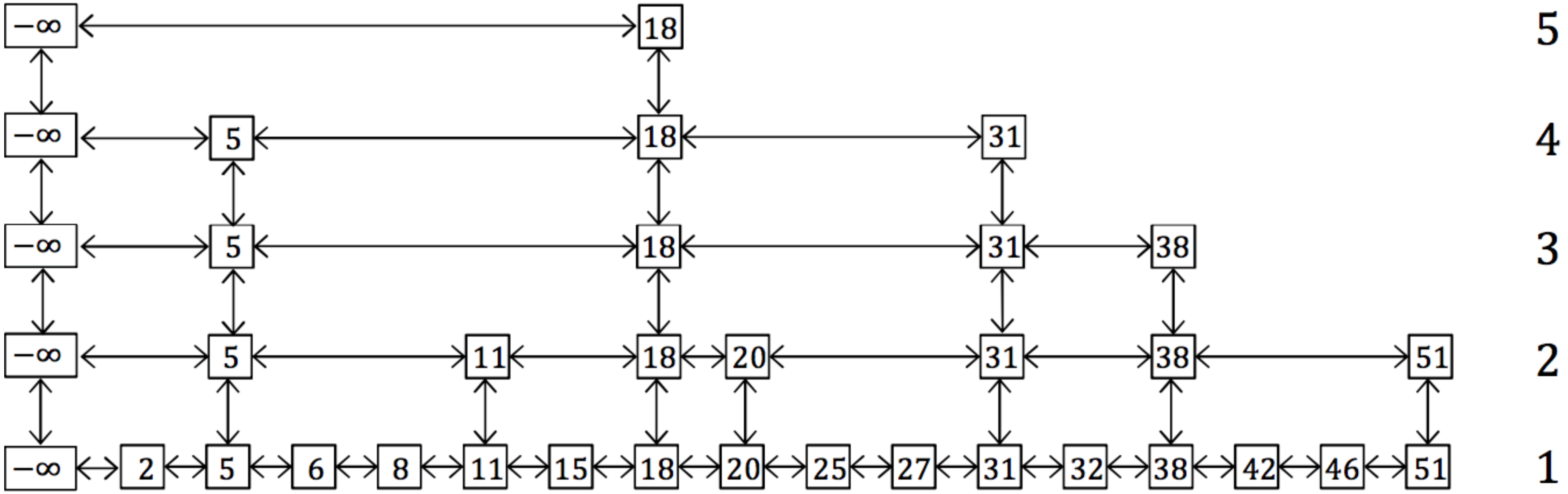
Skip List Search Example



- **Example:** Search for 72
  - Level 1: 14 too small, 79 too big; go down 14
  - Level 2: 14 too small, 50 too small, 79 too big; go down 50
  - Level 3: 50 too small, 66 too small, 79 too big; go down 66
  - Level 4: 66 too small, 72 spot on

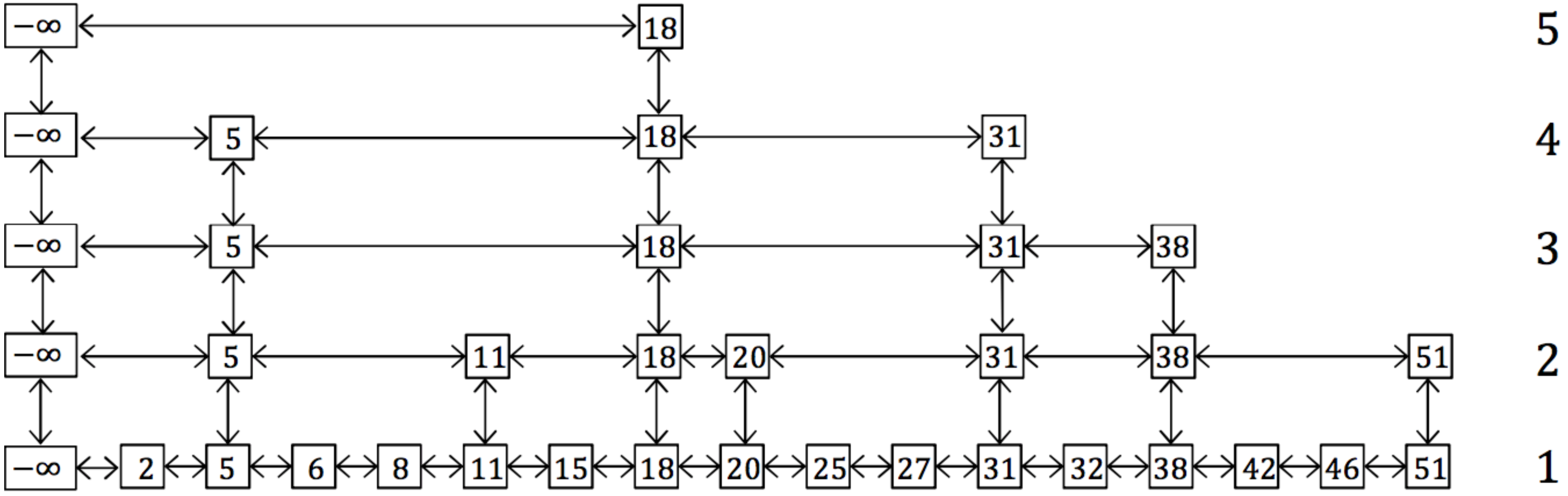
Skip List Analysis: Height

- Let L_k be the set of all items in level $k \geq 1$.
- Height of an element. $\ell(x) = \max\{k \mid x \in L_k\}$
- Height of a skip list. $h(L) = \max\{\ell(x) \mid x \in L_0\}$



Skip List Analysis: Height

- Expected height of a node?
  - Expected number of trials until success (tail): 2
- Worst-case height? $h(L) = \max\{\ell(x) \mid x \in L\}$



Skip List Analysis: Height

- **Claim.** A skip list with n elements has height $O(\log n)$ levels with high probability
- Goal: show that the probability that it has more than $d \log n$ levels is at most $1/n^c$, where the constants c, d usually depend on each other
- **Proof**. For any $x \in L$, $k \geq 1$, the probability that height of x is k
- What is Pr$[\ell(x) = k]$?

$$= \frac{1}{2^k}$$

Skip List Analysis: Height

- **Claim.** A skip list with n elements has height $O(\log n)$ levels with high probability
- Goal: show that the probability that it has more than $d \log n$ levels is at most $1/n^c$, where the constants c, d usually depend on each other
- **Proof**. For any $x \in L$, $k \geq 1$, the probability that height of x is k
- What is $\Pr[\ell(x) = k] = \dfrac{1}{2^k}$
- $\Pr[\ell(x) > k]$ is probability $\ell(x)$ is $k+1$, $k+2$, ... the probability decreases by half each time, thus is at most $\dfrac{1}{2^k}$

Skip List Analysis: Height

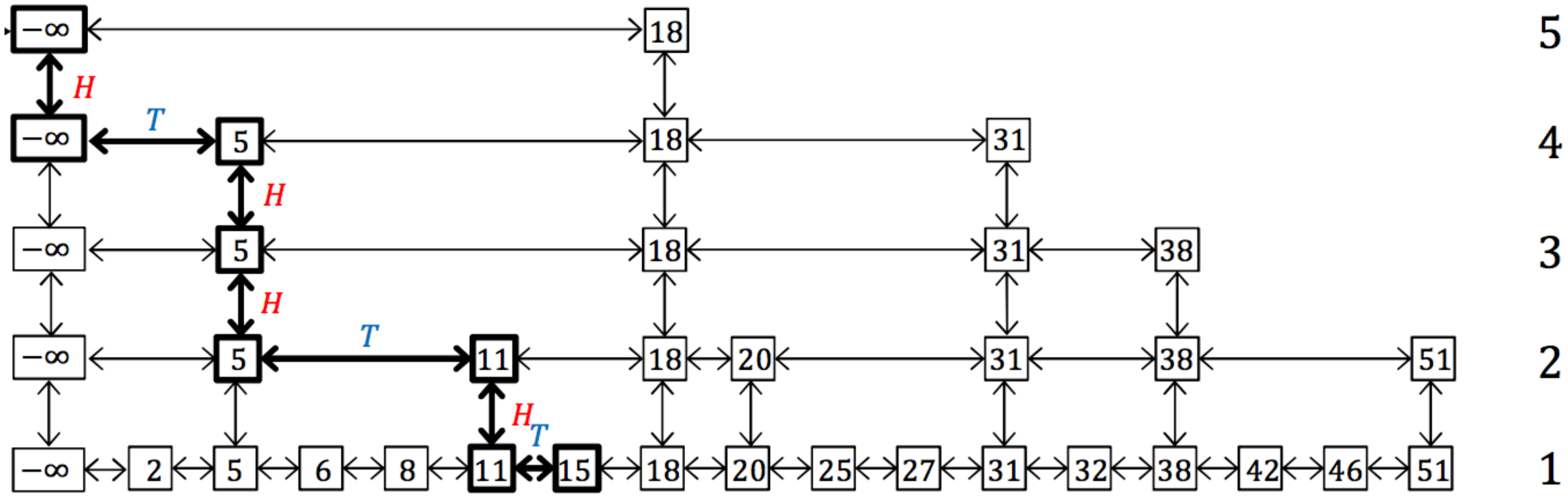
- **Claim**. A skip list with n elements has height $O(\log n)$ levels w.h.p.
- **Proof**. For any $x \in L$, $k \geq 1$, the probability that height of x is k
- $\text{Pr}[\ell(x) > k] = \sum_{k+1}^{\infty} \text{Pr}[\ell(x) = i] = \sum_{i=k+1}^{\infty} \frac{1}{2^i} = \frac{1}{2^k}$
- $\text{Pr}[h(L) > k] = \text{Pr}[\cup_{x \in L} \ell(x) > k] \leq \sum_{x \in L} \Pr[\ell(x) > k] = \frac{n}{2^k}$

Union bound

- $\text{Pr}[h(L) > c \log n] \leq \frac{1}{n^{c-1}}$ [pick any $c > 2$ for w.h.p.]
- Thus, height of skip is $O(\log n)$ with high probability

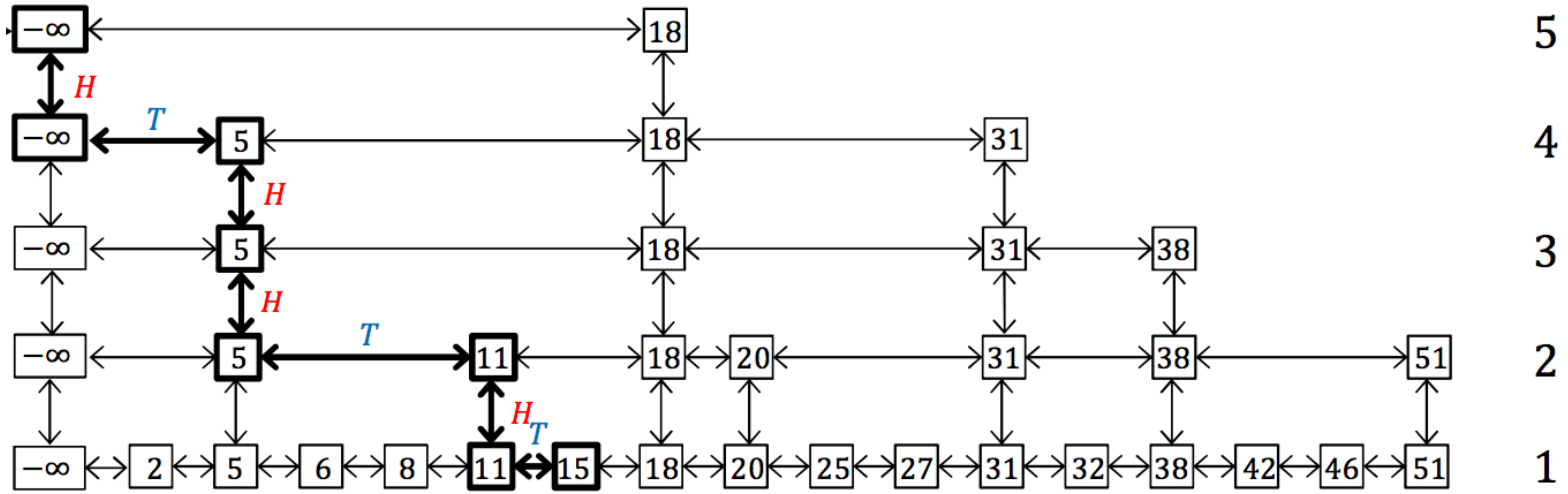
Skip List Search Cost

- **Claim.** Search cost in a skip list is $O(\log n)$ with high probability
- **Proof**. Idea think of the search path "backwards"
- Starting at the target element, going left or up until you reach root or sentinel node $(-\infty)$



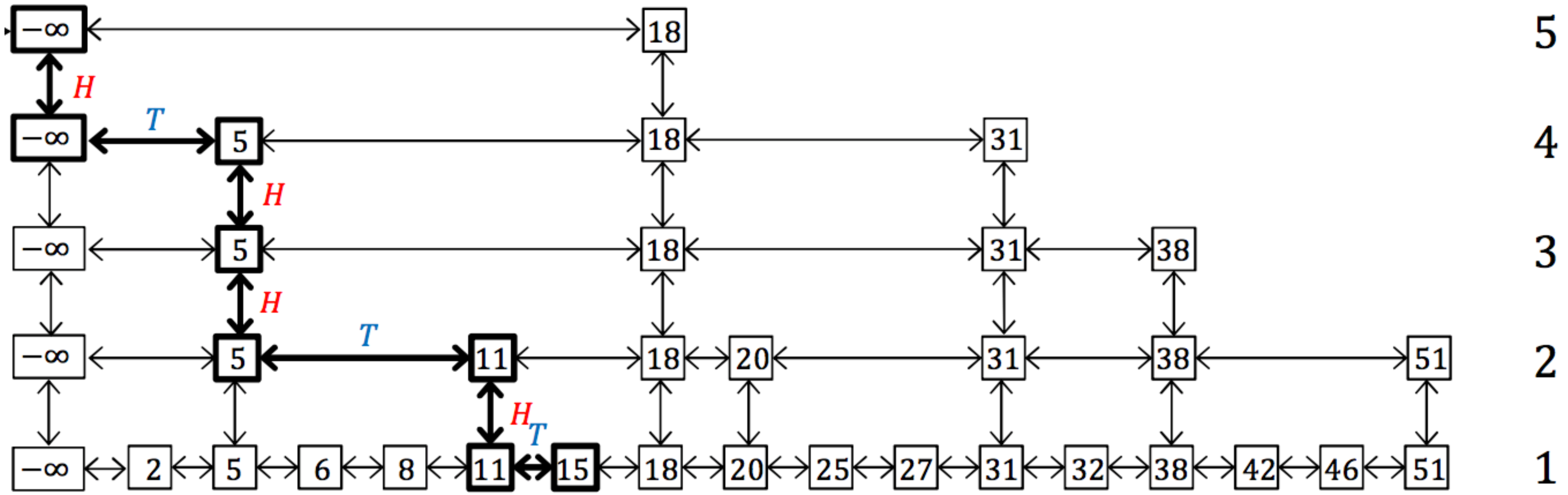
Skip List Search Cost

- Backwards search path, when do go up versus left?
- If node wasn't promoted (got tails here), then we go [came from] left
- If node was promoted (got heads here), then we go [came from] top



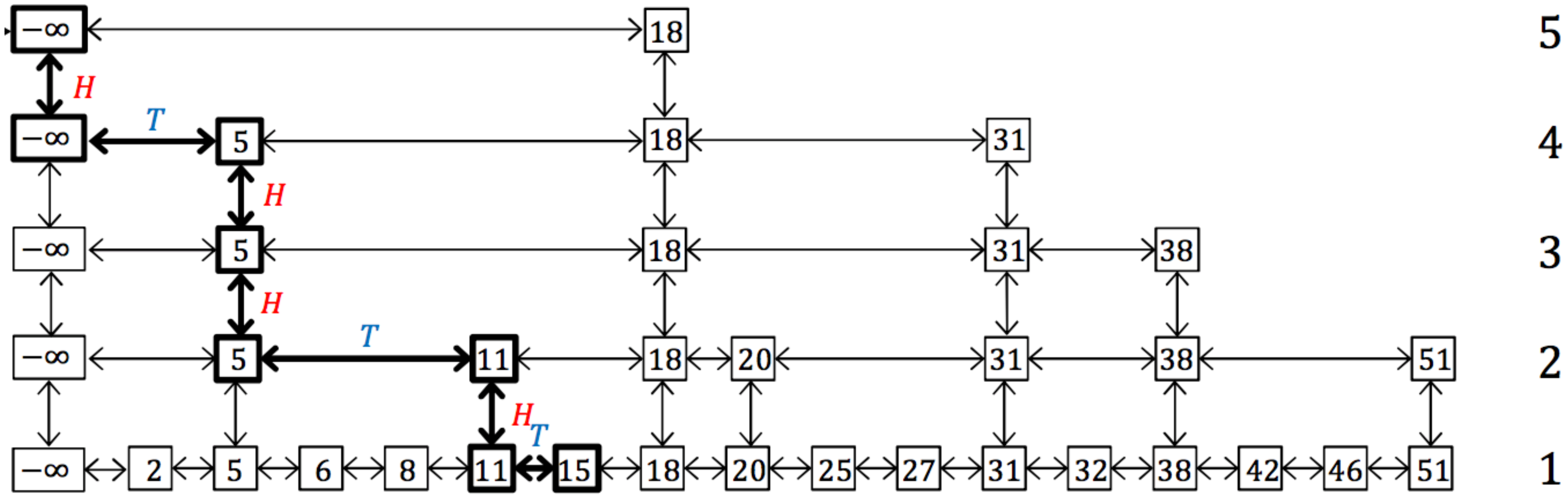
Skip List Search Cost

- How many consecutive tails in a row? (left moves on a level)
  - Same analysis as the height! $O(\log n)$
  - $O(\log^2 n)$ length overall—but I claimed $O(\log n)$ earlier



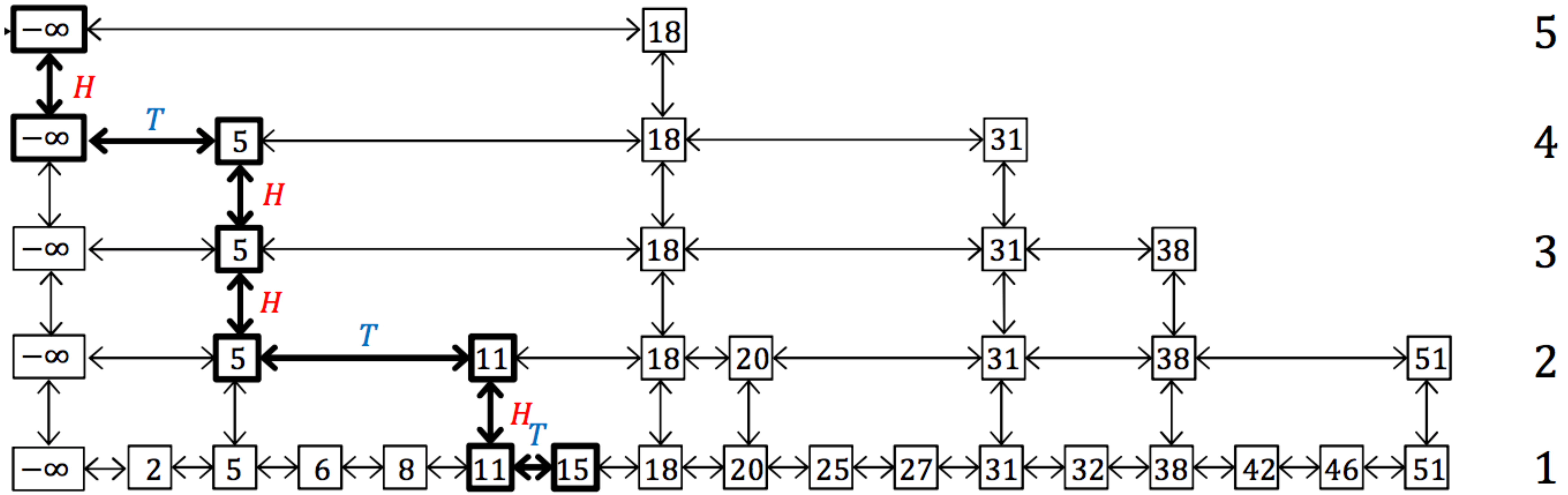
Skip List Search Cost

- Search path is a sequence of $HHHTTTHHTT\dots$
- How many "up" moves (H) before we are done?
  - Height: $c\log n$ with high probability



Skip List Search Cost

- Search ends when we reach top list: have seen at least $c \log n$ heads
- **Search cost**: Can we bound the number of times do we need to flip a coin until we see $c \log n$ heads with high probability?



Coin Flipping

- **Claim.** Number of flips until $c \log n$ heads is $\Theta(\log n)$ with high probability, that is, with probability $1 - 1/n^c$
  - Note. Constant in $\Theta(\log n)$ will depend on c
- **Proof**. Say we flip $10c \log n$ coins
  - Pr[exactly $c \log n$ heads]

$$= \binom{10c \log n}{c \log n} \cdot \left(\frac{1}{2}\right)^{c \log n} \cdot \left(\frac{1}{2}\right)^{9c \log n}$$

  - Pr[at most $c \log n$ heads] $\leq \binom{10c \log n}{c \log n} \cdot \left(\frac{1}{2}\right)^{9c \log n}$

Coin Flipping

- **Claim.** Number of flips until $c \log n$ heads is $\Theta(\log n)$ with high probability, that is, with probability $1 - 1/n^c$
- **Proof**. Pr[at most $c \log n$ heads] $\leq \left(\dfrac{e \cdot 10c \log n}{c \log n}\right)^{c \log n} \cdot \left(\dfrac{1}{2}\right)^{9c \log n}$

$$= (10e)^{c \log n} \cdot \left(\frac{1}{2}\right)^{9c \log n}$$

$$= 2^{\log(10e) \cdot c \log n} \cdot \left(\frac{1}{2}\right)^{9c \log n}$$

$$= 2^{(\log(10e) - 9) \cdot c \log n} = 2^{-d \log n}$$

$$= 1/n^d$$

Coin Flipping

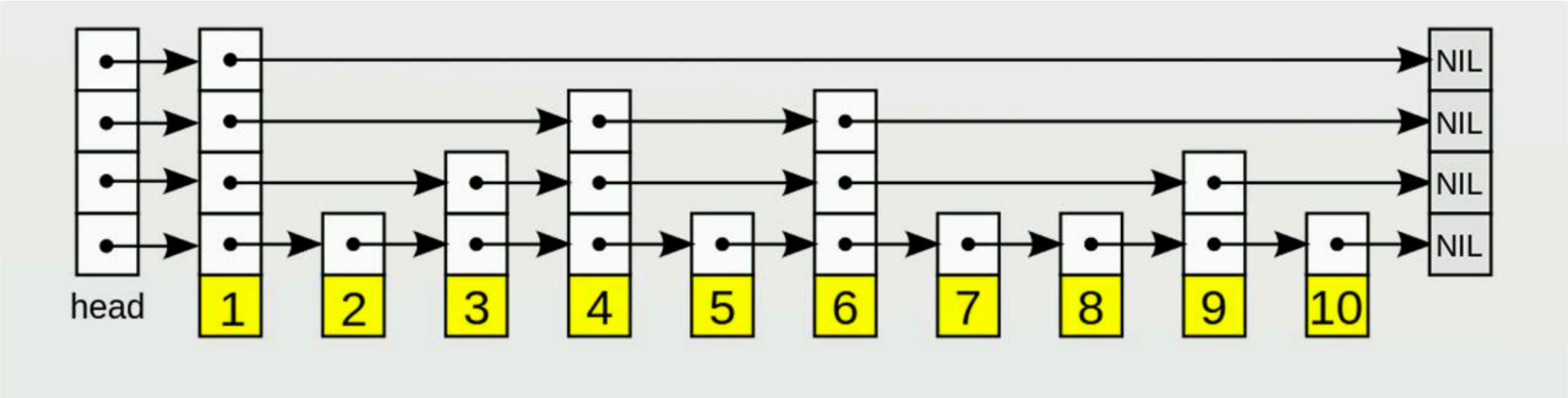
- **Claim.** Number of flips until $c \log n$ heads is $\Theta(\log n)$ with high probability, that is, with probability $1 - 1/n^c$

- **Proof**. Pr[at most $c \log n$ heads] $\leq \left(\dfrac{e \cdot 10c \log n}{c \log n}\right)^{c \log n} \cdot \left(\dfrac{1}{2}\right)^{9c \log n}$

$$= (10e)^{c \log n} \cdot \left(\frac{1}{2}\right)^{9c \log n}$$

Skip Lists

- Using $O(\log n)$ linked lists, achieve same performance as binary search tree
- No stored information about balance, no tricky balancing rules!
- Just flip coins while inserting each new element to decide what lists it goes in



Source: Wikipedia

Cuckoo Hashing:
Brief Overview

Cuckoo Hashing [Pagh, Rodler ’01]

- 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?

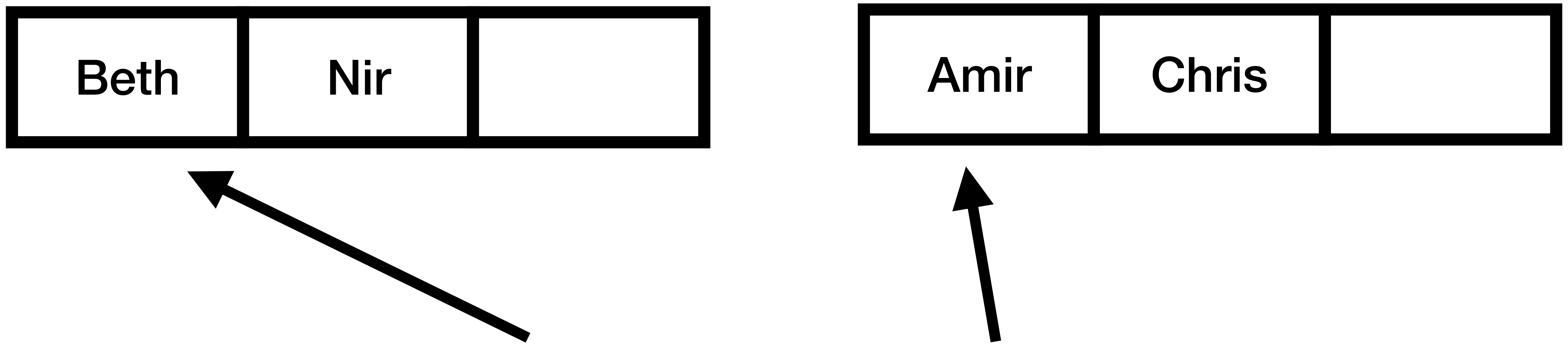
Beth	**Nir**	

Amir	**Chris**	

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

Cuckoo Hashing: Insert

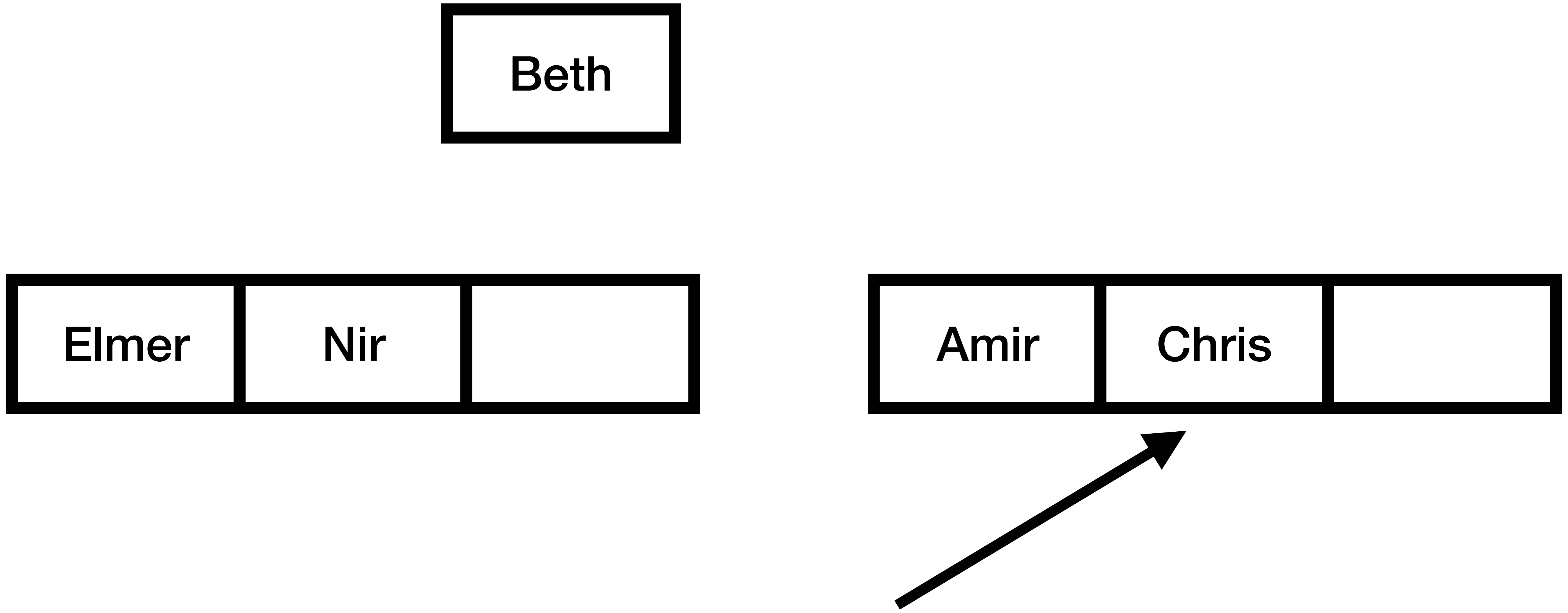
- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash



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

Cuckoo Hashing: Insert

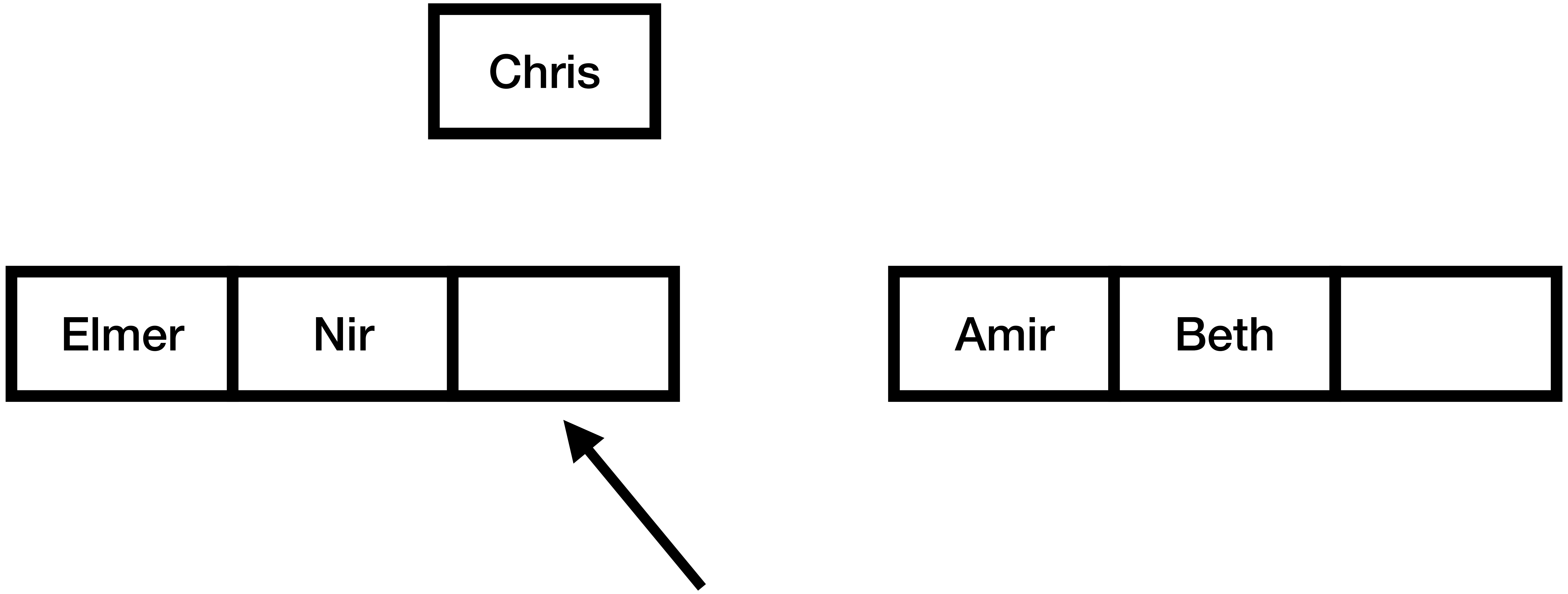
- If $A[h_1(i)]$ or $A[h_2(i)]$ is empty, store i
- Otherwise, kick an item out of one of these locations
- Reinsert that item using its other hash



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

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**	

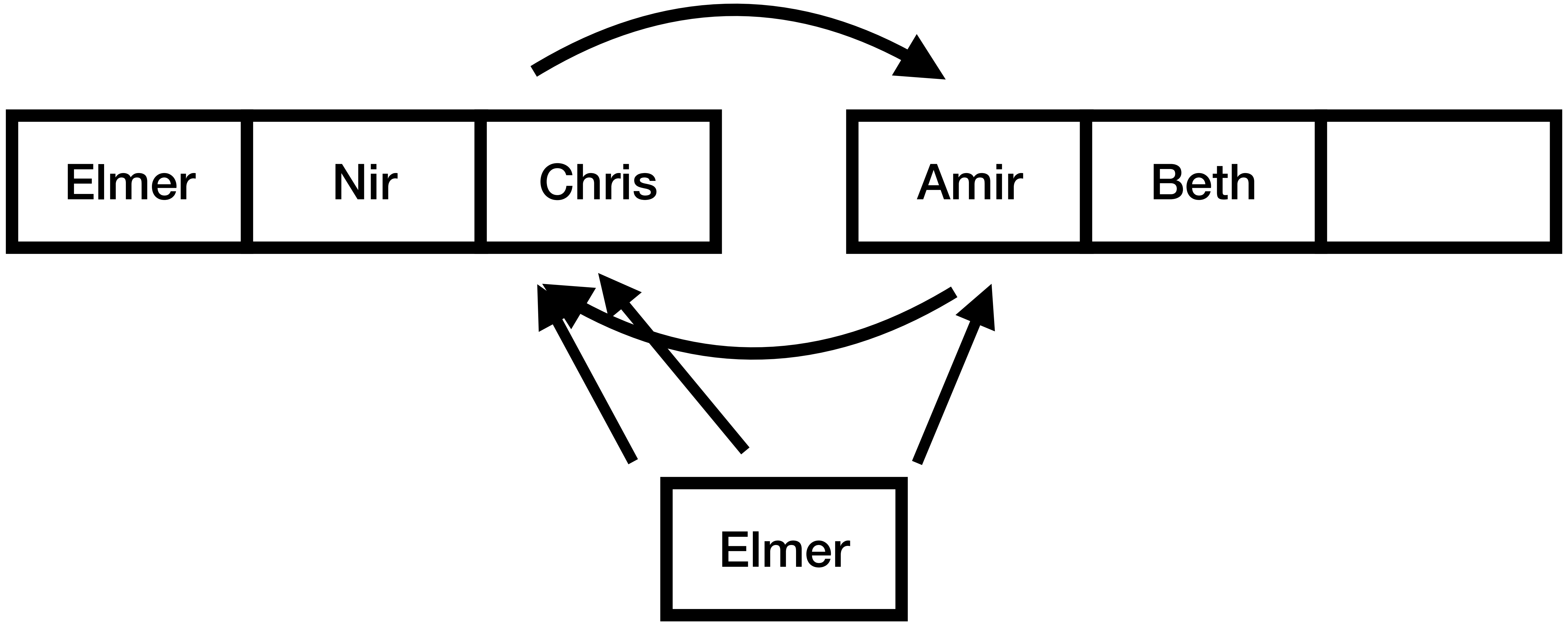
$$h_1(\text{Chris}) = 2, \;\; h_2(\text{Chris}) = 1$$

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 we have an insert to two slots, where each item in those slots only hashes to those two slots?
- $\binom{n}{2}\left(\frac{1}{n}\right)^4 = \Theta(1/n^2)$

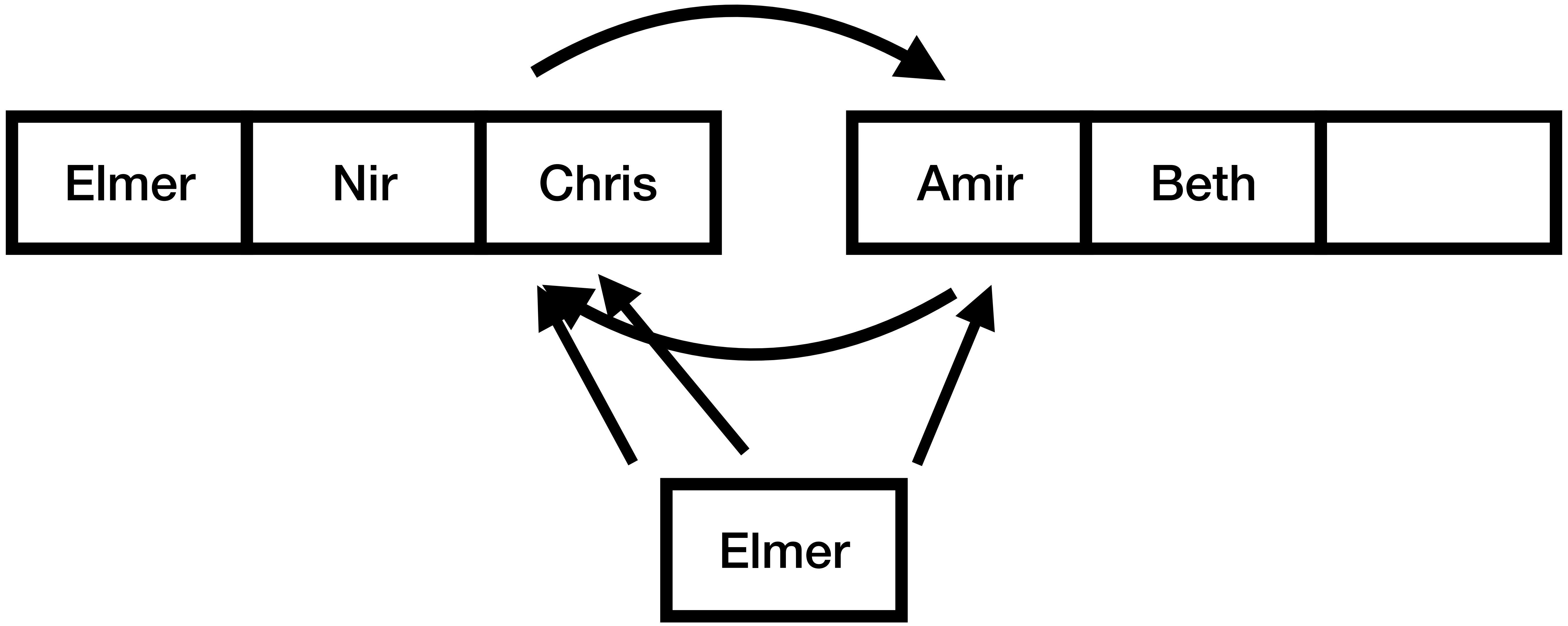
Ways to choose 2 items out of the n inserted

Probability that those two items hash to the given two slots



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?
    - $\approx 1/2$
    - (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

Cuckoo Hashing: In Practice

- Cuckoo hashing ends up being a bit lower than linear probing:
    - Two cache misses per search
    - Can be problematic depending on which memory hierarchy your data structure fits in

Acknowledgments

- Some of the material in these slides are taken from
  - MIT slides: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-12-skip-lists/lec12.pdf
  - Eric Demaine handout: https://courses.csail.mit.edu/6.046/spring04/handouts/skiplists.pdf