CS2040S
Data Structures and Algorithms
(e-learning edition)

Skip Lists!

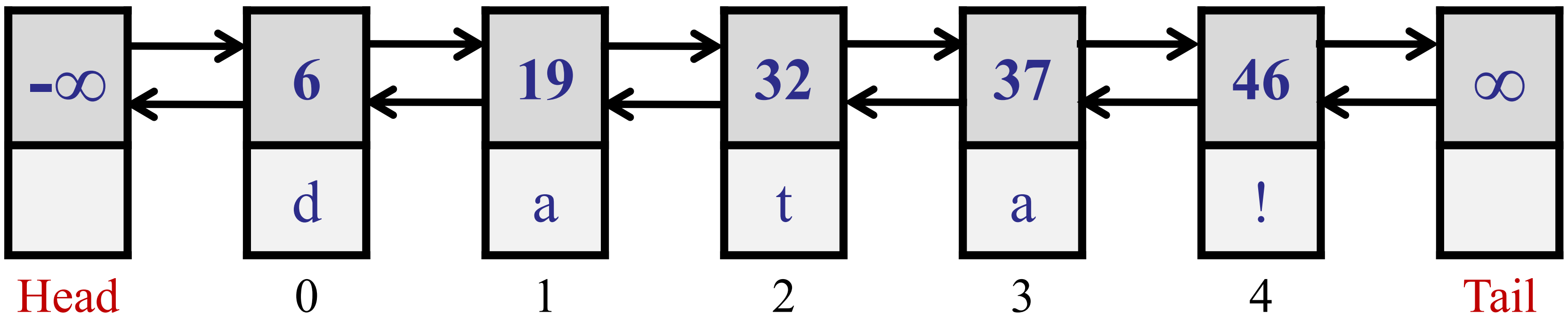
Dictionary Interface

A collection of (key, value) pairs:

interface	**IDictionary**	
void	insert(Key k, Value v)	*insert (k,v) into table*
Value	search(Key k)	*get value paired with k*
Key	successor(Key k)	*find next key > k*
Key	predecessor(Key k)	*find next key < k*
void	delete(Key k)	*remove key k (and value)*
boolean	contains(Key k)	*is there a value for k?*
int	size()	*number of (k,v) pairs*

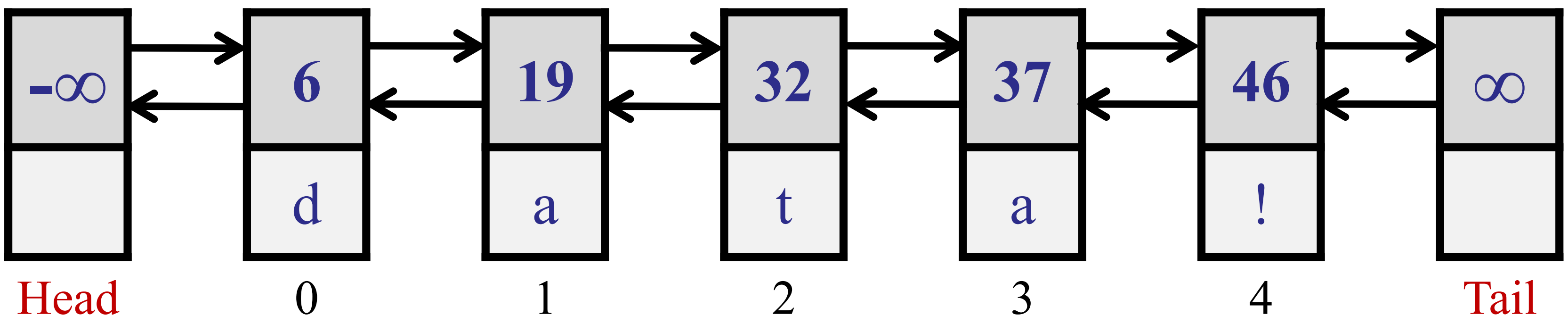
Implementing a dictionary, again…

Store keys in a sorted linked list:



Implementing a dictionary, again…

Store keys in a sorted linked list:



special
head node

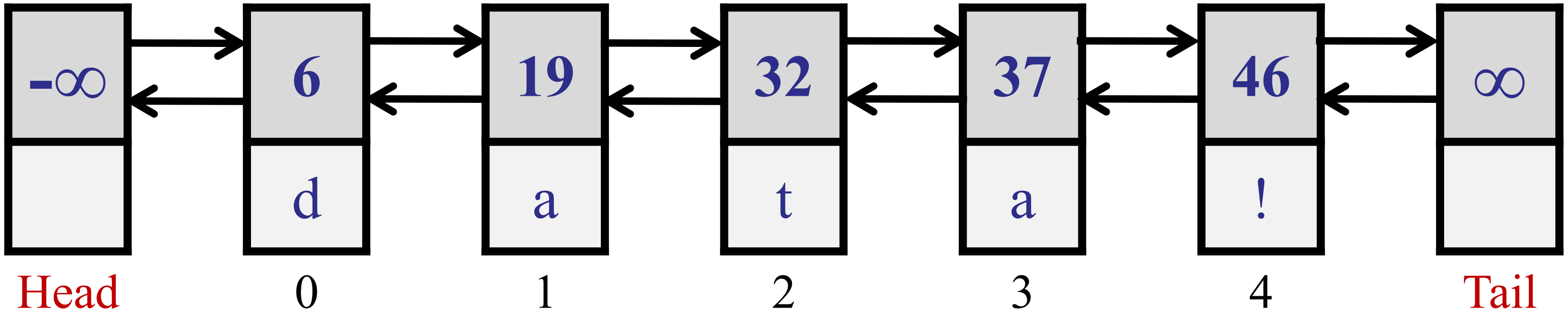
Each node stores:

- key
- value
- links

special
tail node

Implementing a dictionary, again…

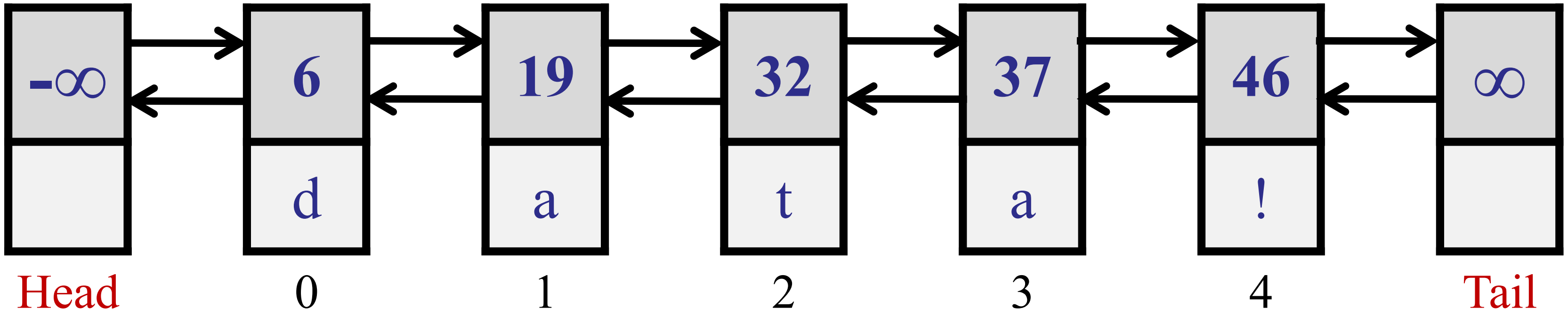
Store keys in a sorted linked list:



<u>Doubly linked:</u>
Each node has a link to its predecessor and successor in the list.

Implementing a dictionary, again…

Store keys in a sorted linked list:

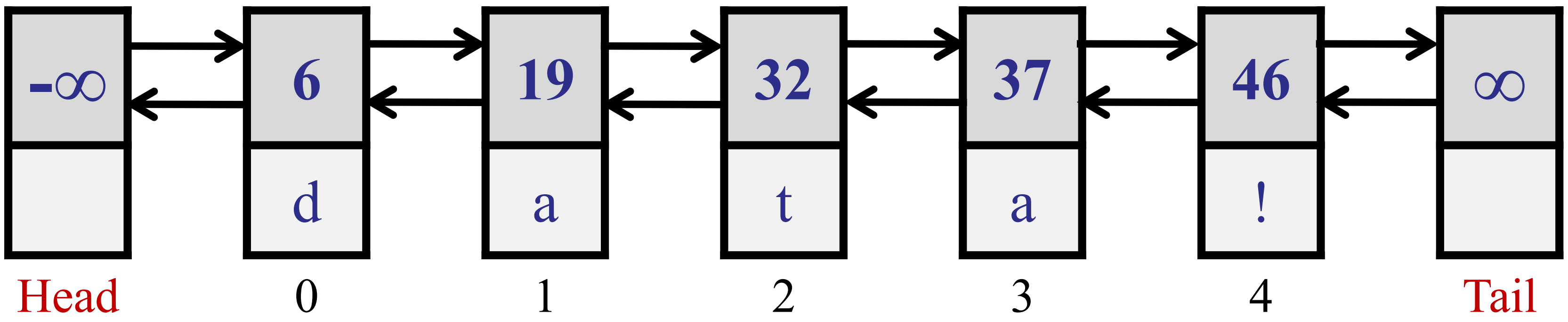


Time:

- Search: O(n)
- Insert: O(n)

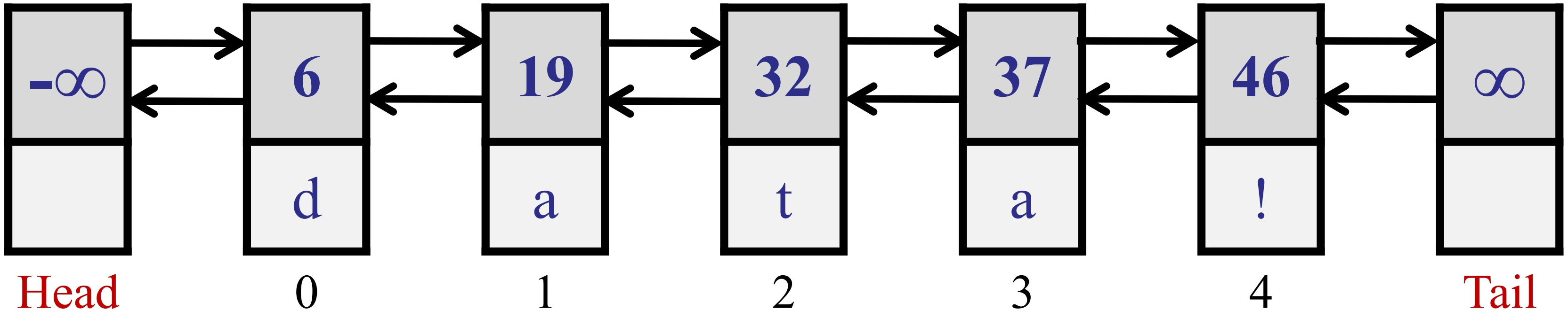
Implementing a dictionary, again…

Why doesn't binary search work??



Implementing a dictionary, again…

Store keys in a sorted linked list:



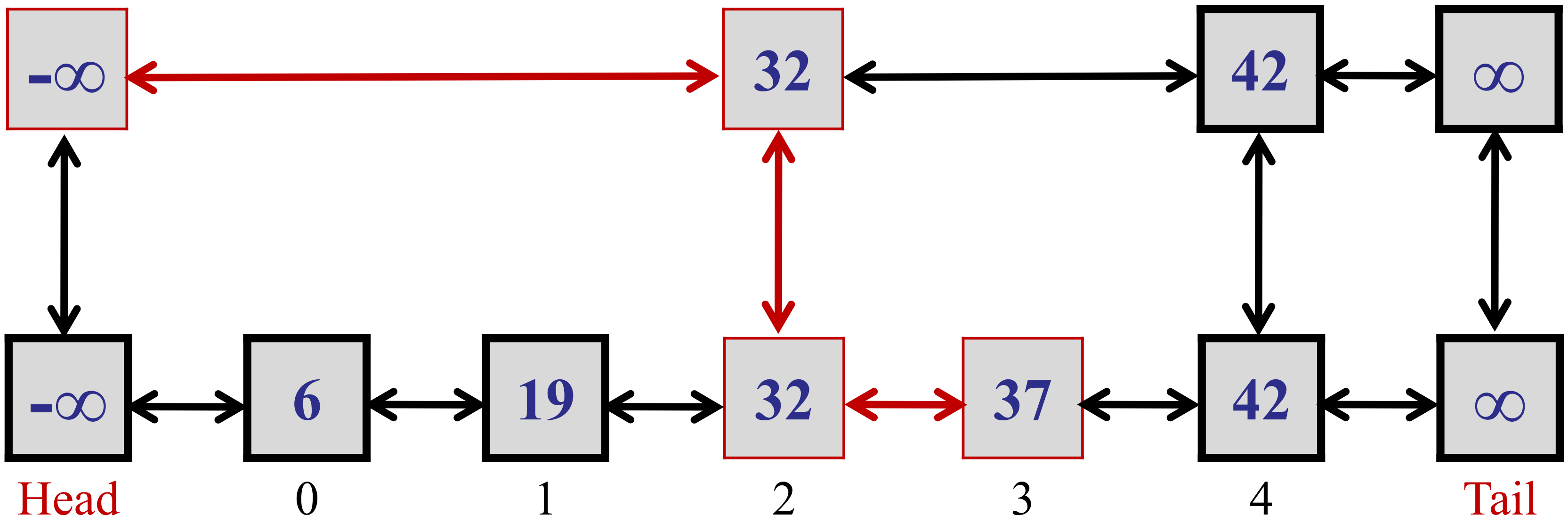
Time:

- Search: O(n)
- Insert: O(n)

What if…

What if we use two lists?

- Express train
- Local train

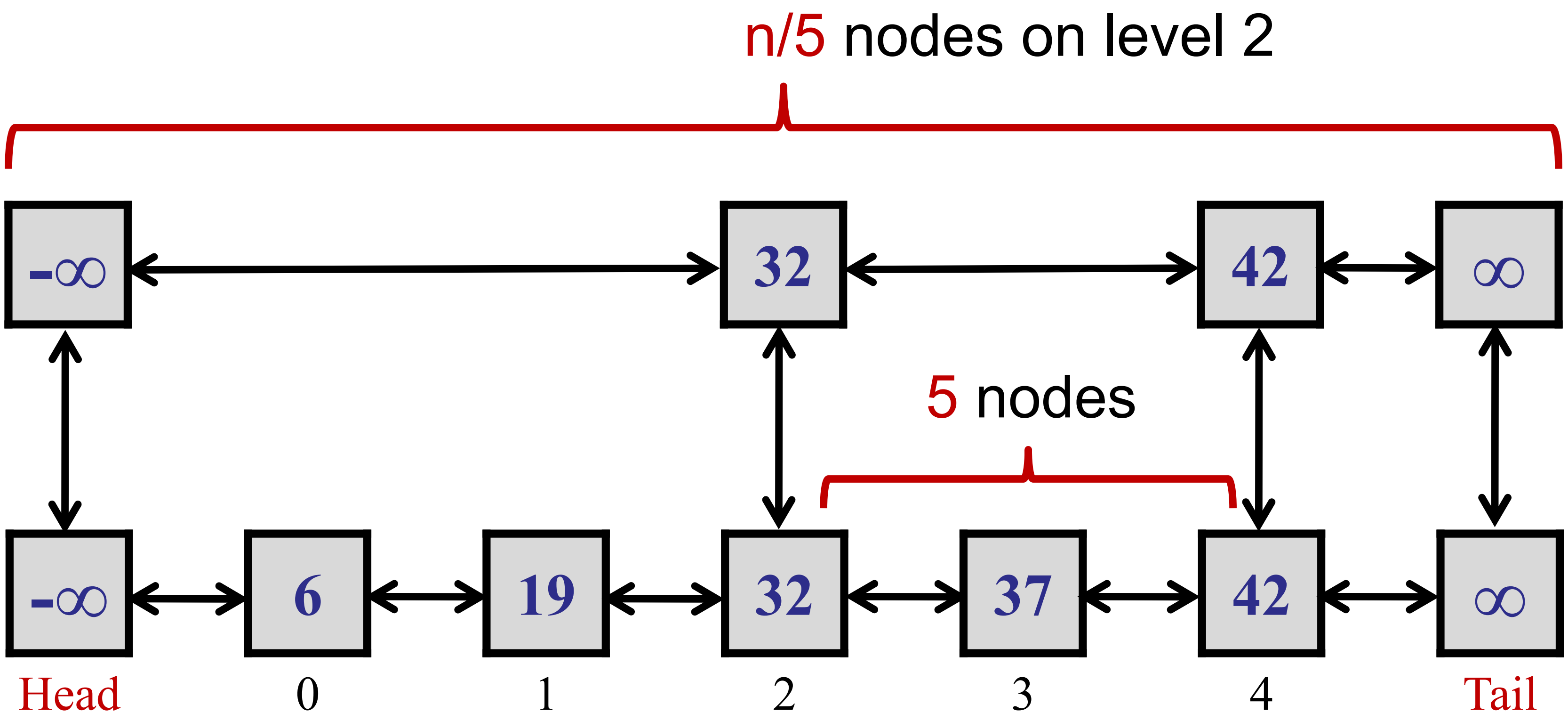


`search(37)` takes only 3 steps!

What if…

Calculation:

If the "express" list skips 5 elements per "stop", then search takes at most: $n/5 + 5$ steps



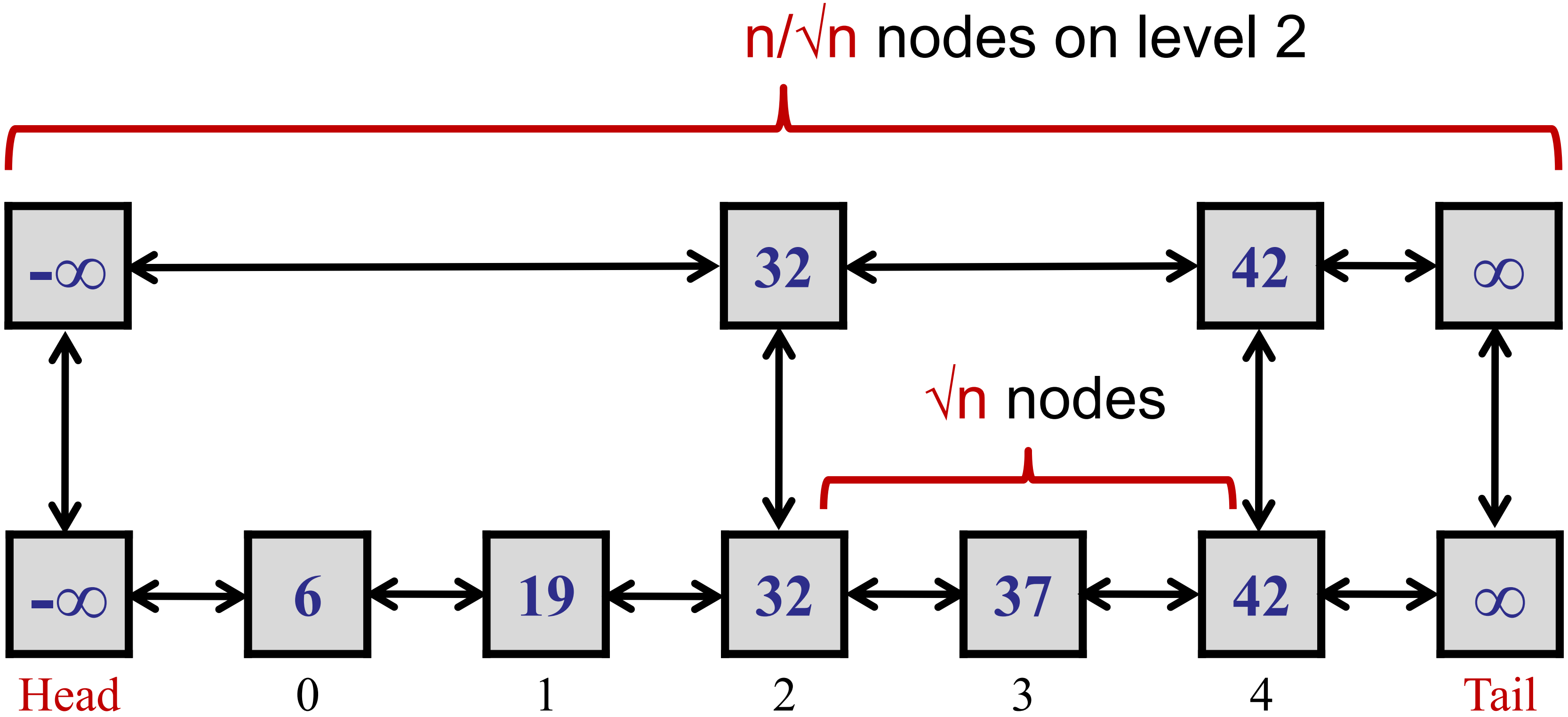
With two lists, how many elements should the express list skip per hop?

1. O(1)
2. log(n)
3. √n
4. n/√n
5. n/log(n)
6. Something else.

What if…

If the “express” list skips √n elements per “stop”, then search takes at most:

$$\frac{n}{\sqrt{n}} + \sqrt{n} = 2\sqrt{n} = O(\sqrt{n})$$



Why stop at two?

Add more lists:

- Two lists: $2\sqrt{n}$
- Three lists: $3\sqrt[3]{n}$

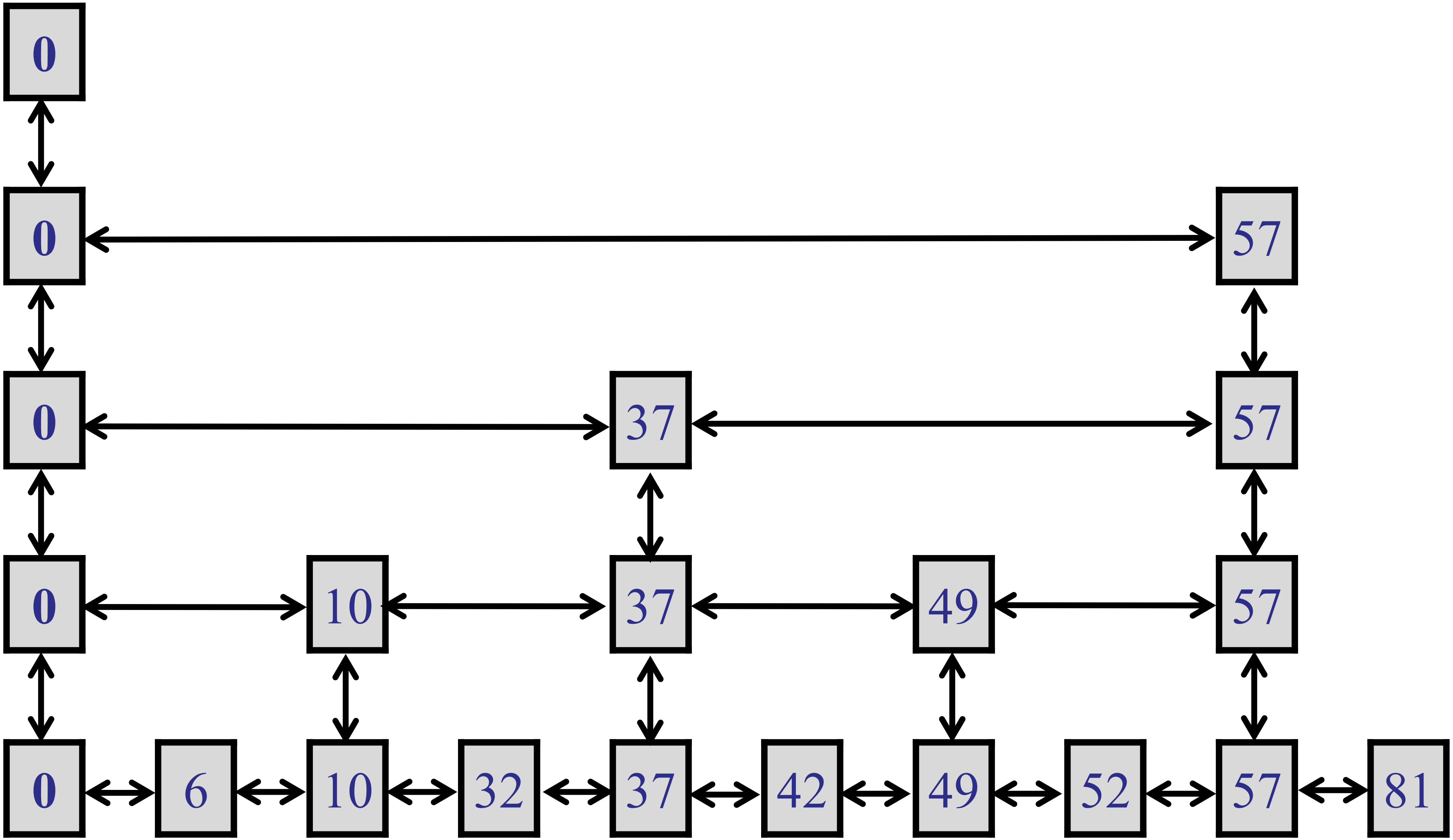
…

- *k* lists: $k\sqrt[k]{n}$

- log(*n*) lists: $\log(n)\ \sqrt[\log(n)]{n} = \log(n) n^{1/\log(n)}$

$$= 2\log(n)$$

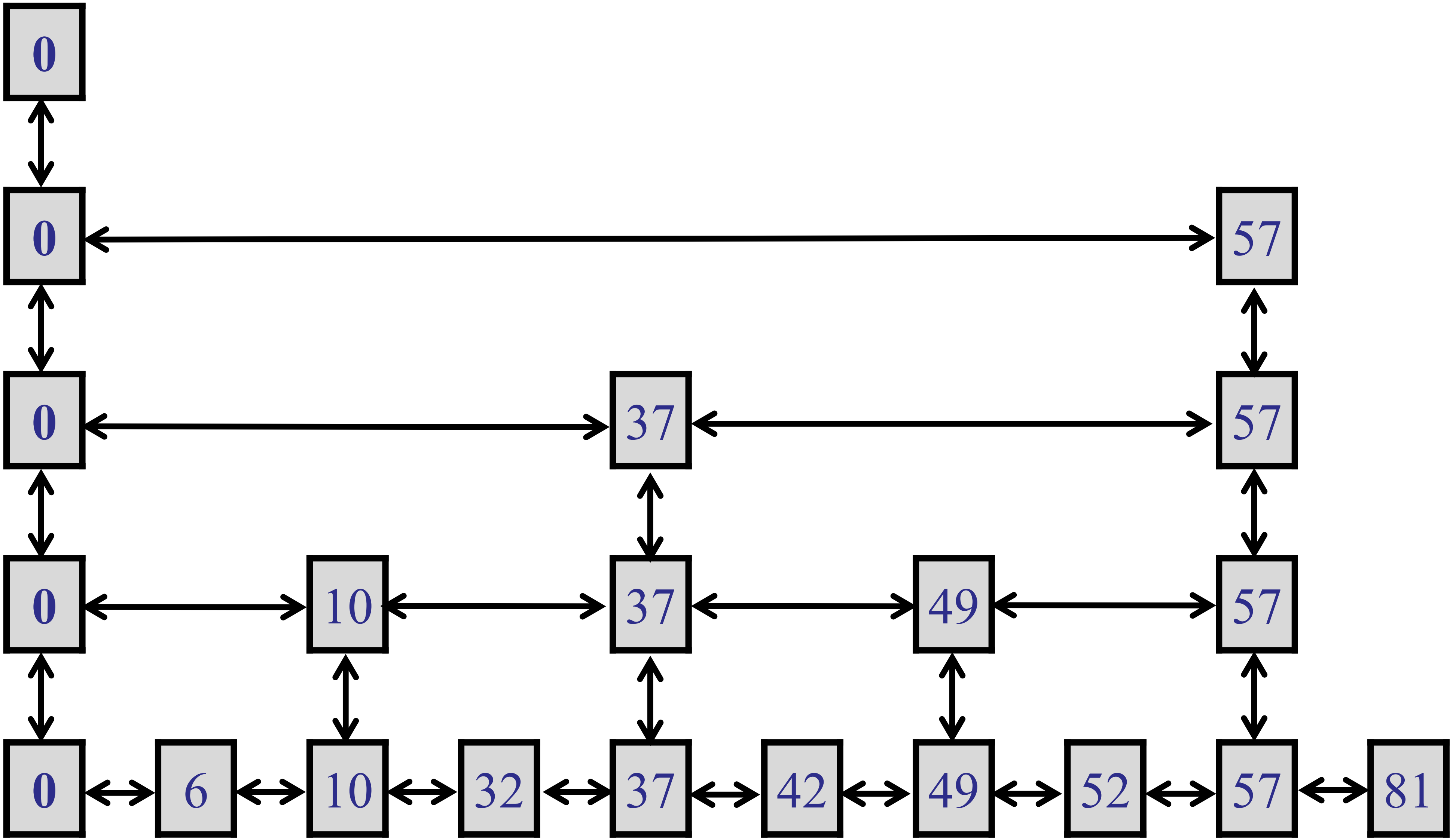
Another way to think about it…



How many levels?

1. $O(1)$
2. $\log(n)$
3. $2\log(n)$
4. $\log^2(n)$
5. $\sqrt{n}$
6. None of the above.

Another way to think about it...



SkipList Background

Simple randomized, dynamic search structure

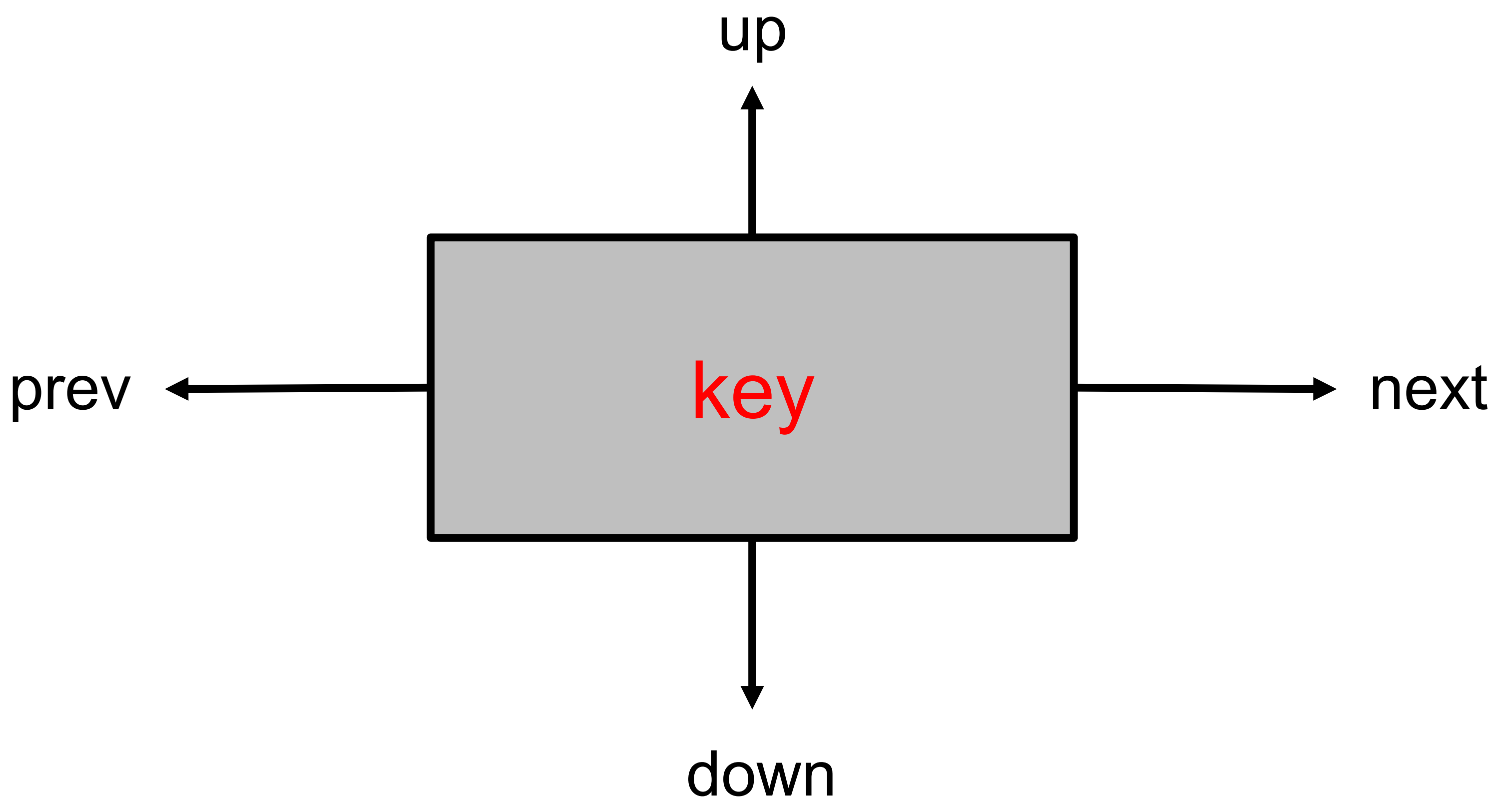
- Invented by William Pugh in 1989
- Easy to implement

Maintains a set of n elements:

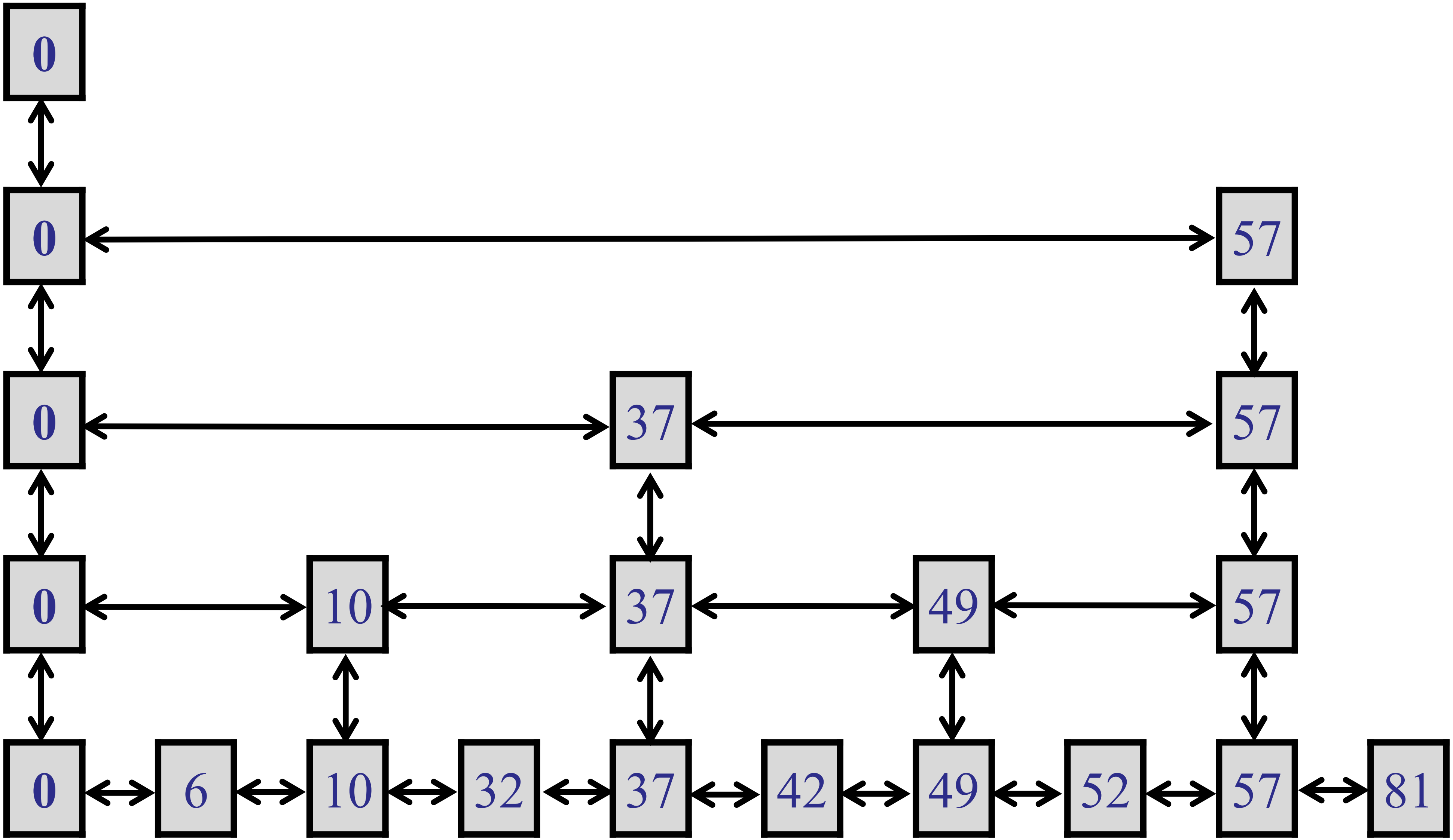
- search: O(log n) time
- insert/delete: O(log n) time

with high probability

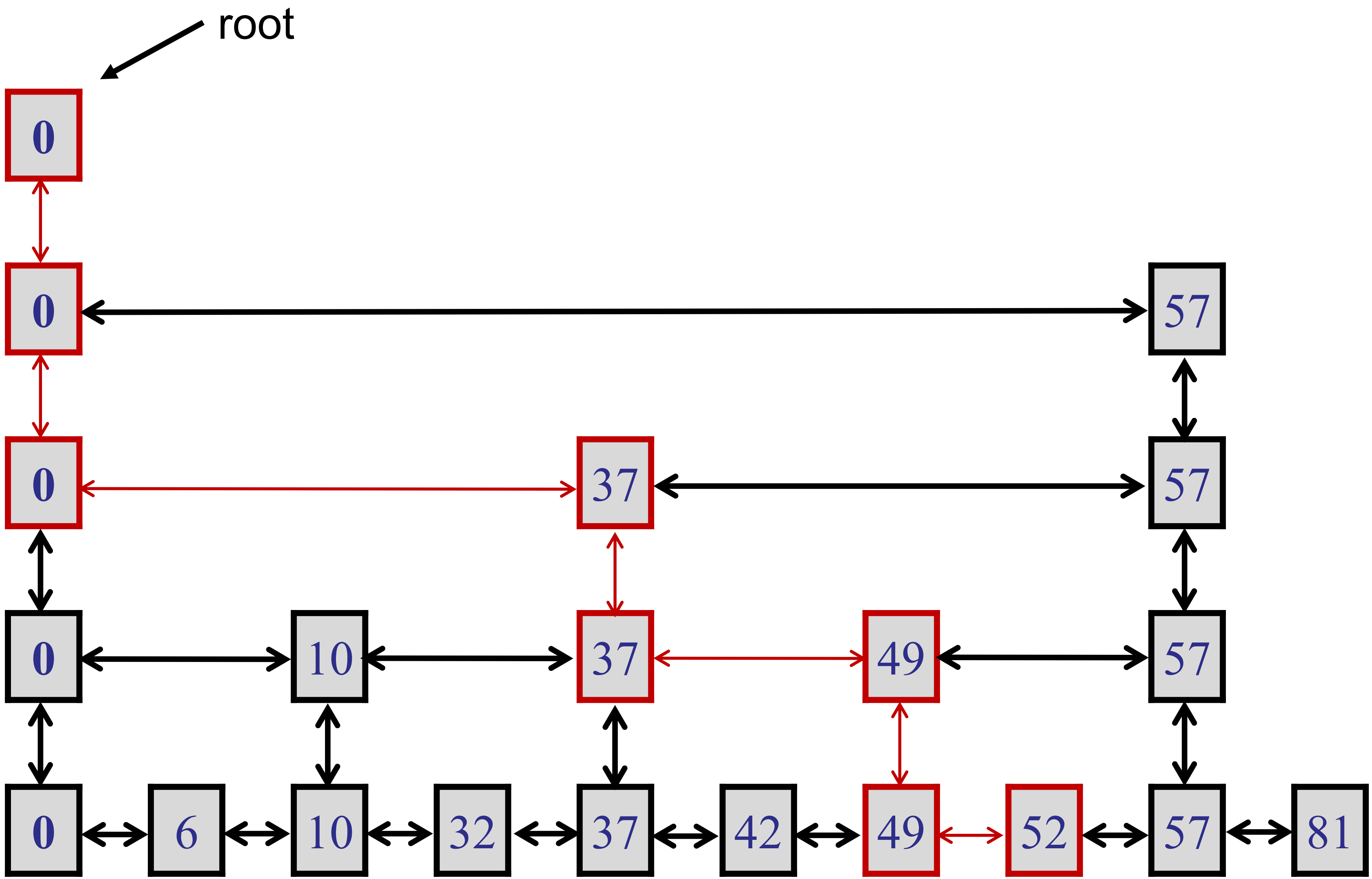
SkipList node



Example: `search(52)`

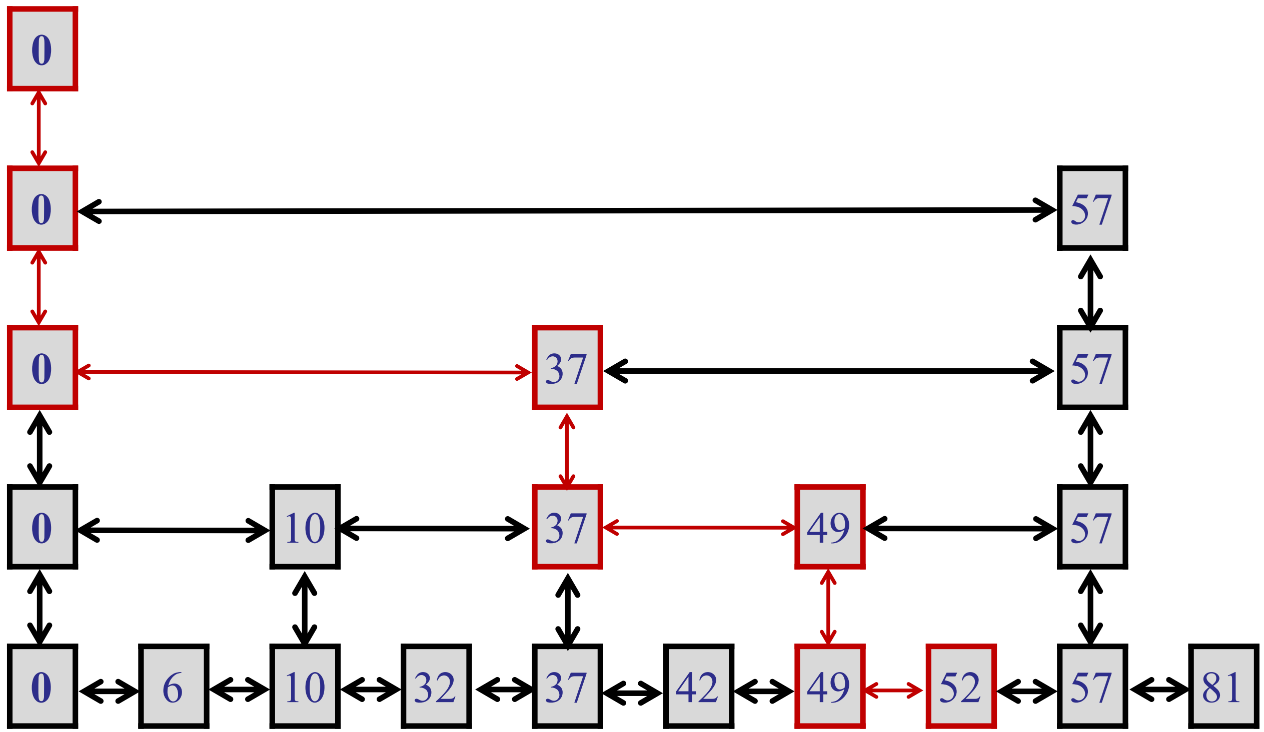


Example: `search(52)`



search(key)

```
node = root;
while (node.key < key) and (node.level > 1):
        while (node.next.key < key):
                node = node.next
        node = node.down
return node
```



Insertions

To insert a new element:

1. Add element to bottom list.

   (Invariant: bottom list contains every element.)

2. Add element to some other lists to maintain balance.

   Goal: about half of elements at level `j` get promoted to level `j+1`.

Insertions

Key idea: flip a coin

```
k = 0;
while (!done) {
        Insert element into level k list.
        Flip a fair coin:
                with probability ½: done = true;
                with probability ½: k = k+1;
}
```

Insertions

To insert a new element:

1. Add element to bottom list.

   (Invariant: bottom list contains every element.)

2. Flip coins to decide how many levels to promote.

– On average: **Level 0:** n

Level 1: $n/2$

Level 2: $n/4$

…

Level $\log(n)$**:** $O(1)$

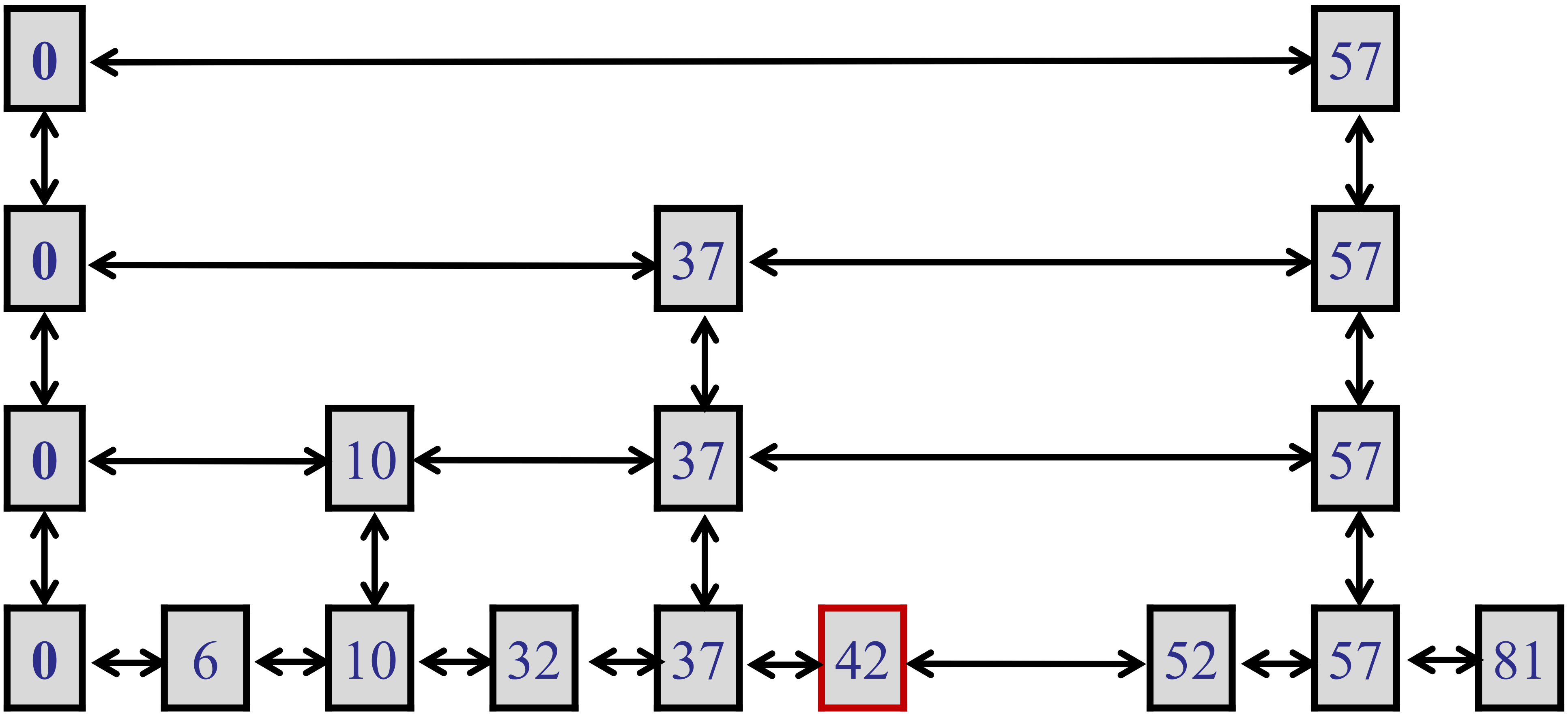
SkipList

Randomized process:

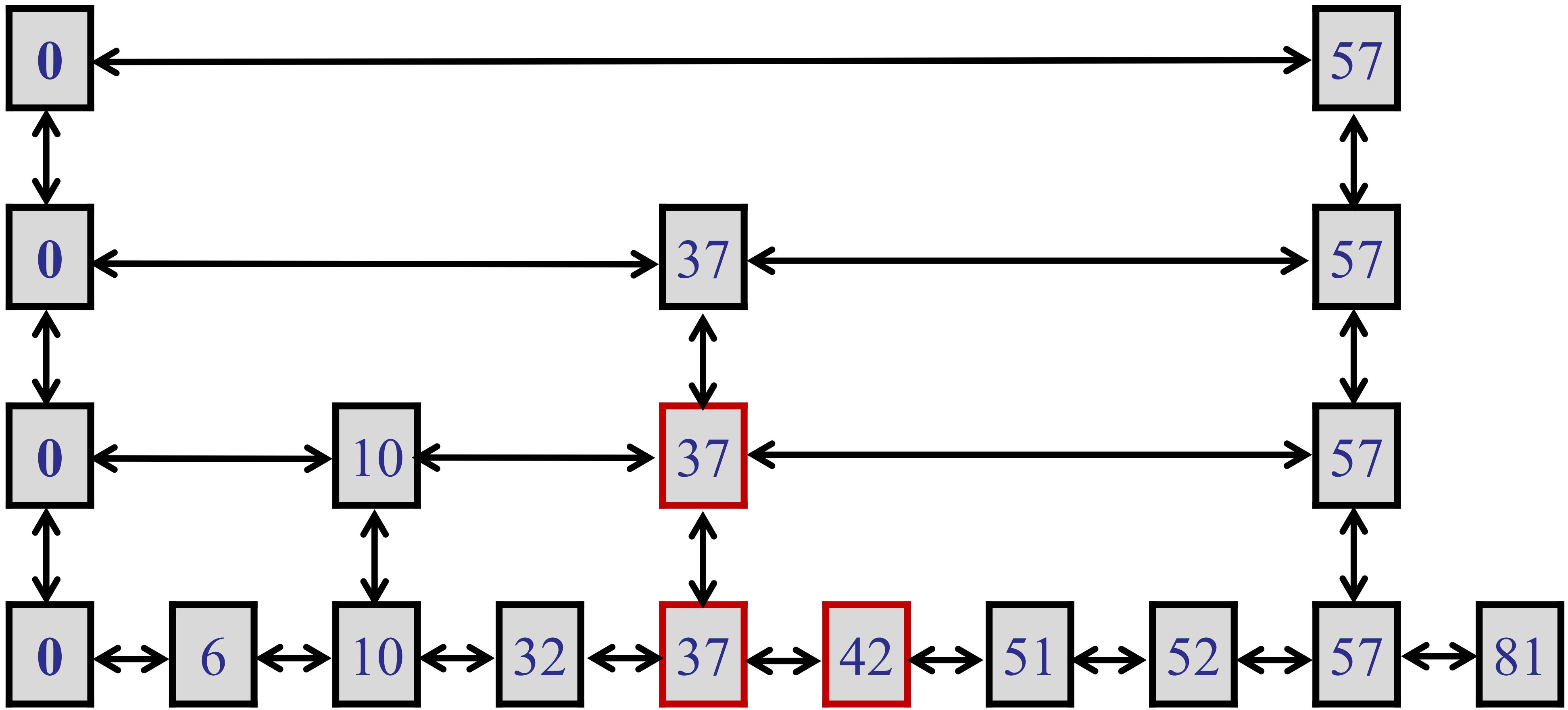
- Not a perfect distribution.
- Good, on average.
- Really good, *almost always*.

- As usual with randomized algorithms, easy to implement, harder to analyze.

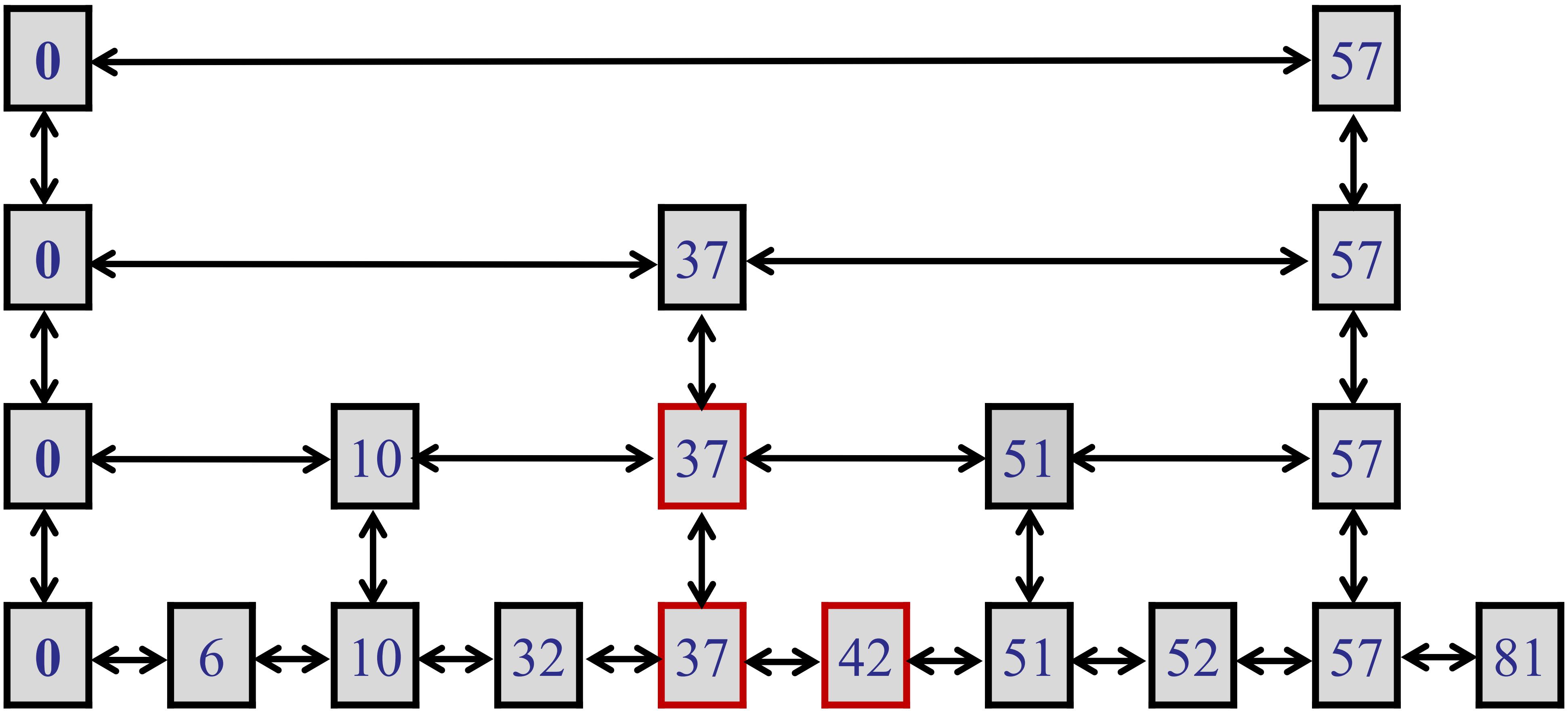
Example: `insert(51)`



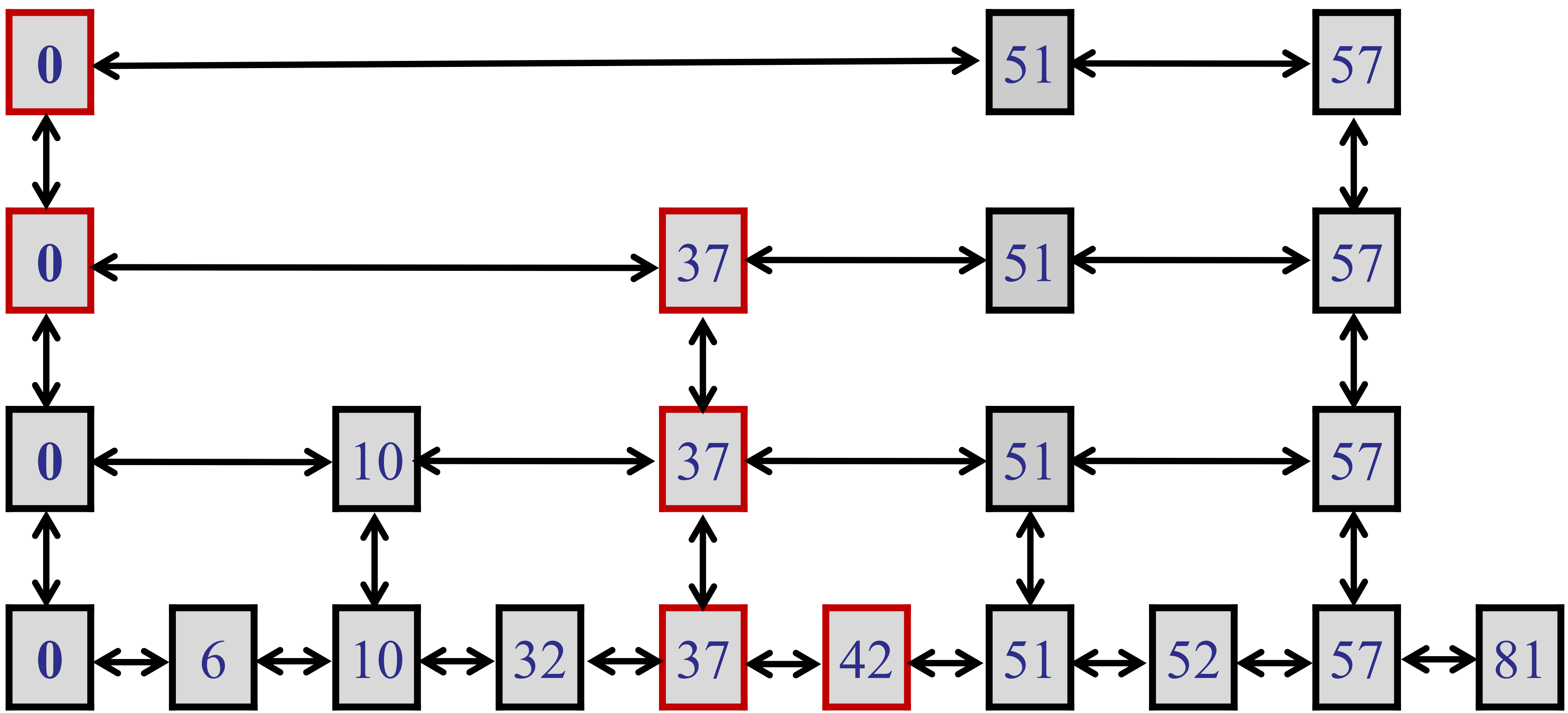
Example: `insert(51)`



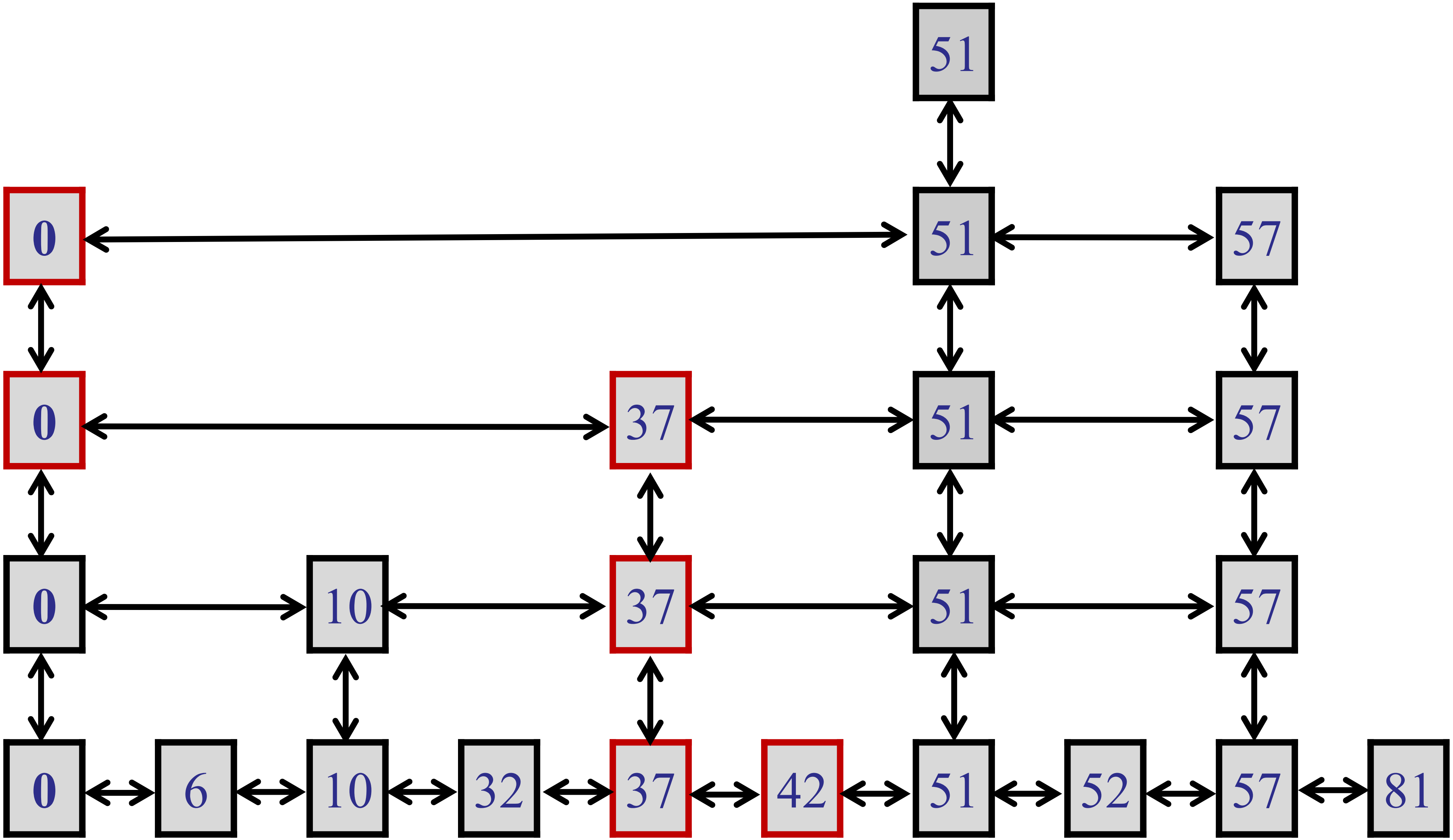
Example: `insert(51)`



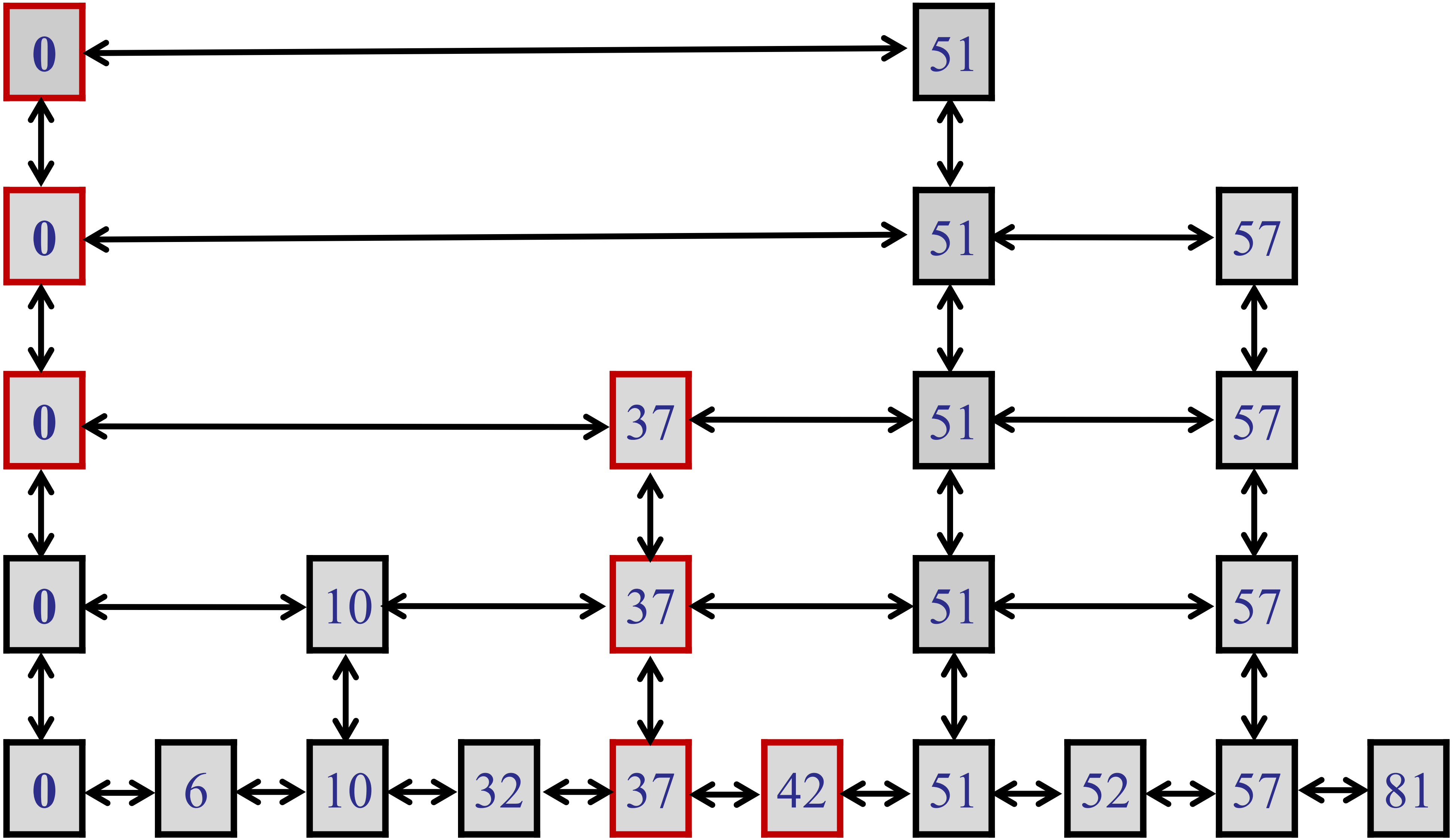
Example: `insert(51)`



Example: `insert(51)`



Example: `insert(51)`



SkipList Analysis

SkipList Analysis

Claim: Every search and insert operation completes in $O(\log n)$ time *with high probability* (i.e., at least $1 - 1/n$).

Key steps:

- Analyze number of levels in a SkipList.
- Look at distribution of promotions:

  SkipList is efficient when each jump skips about the same number of elements.

SkipList Analysis

Claim: With high probability, a SkipList with n elements has $O(\log n)$ levels.

SkipList Analysis

Claim: With high probability, a SkipList with n elements has O(log n) levels.

Proof:

Fix an element x.

Pr[x is higher than c log(n)] $\leq \frac{1}{2^{c \log n}} \leq \frac{1}{n^c}$

Probability of flipping more than $c\log(n)$ heads in a row!

SkipList Analysis

Proof: Fix an element x.

Pr[x is higher than $c \log(n)$] $\leq \dfrac{1}{2^{c \log n}} \leq \dfrac{1}{n^c}$

Define:

- e_1 = probability first element is <u>too high</u> $< 1/n^c$
- e_2 = probability second element is <u>too high</u> $< 1/n^c$

…

- e_n = probability n^th element is <u>too high</u> $< 1/n^c$

Pr(***any*** element is too high) $\leq \dfrac{n}{n^c} \leq \dfrac{1}{n^{c-1}}$

SkipList Analysis

Claim: With high probability, a SkipList with n elements has $O(\log n)$ levels.

SkipList Analysis

Done!

Claim: Every search and (insert) operation completes in O(log n) time *with high probability*.

Done!

Key steps:

- Analyze number of levels in a SkipList.
- Look at distribution of promotions:

  SkipList is efficient when each jump skips about the same number of elements.

SkipList Analysis

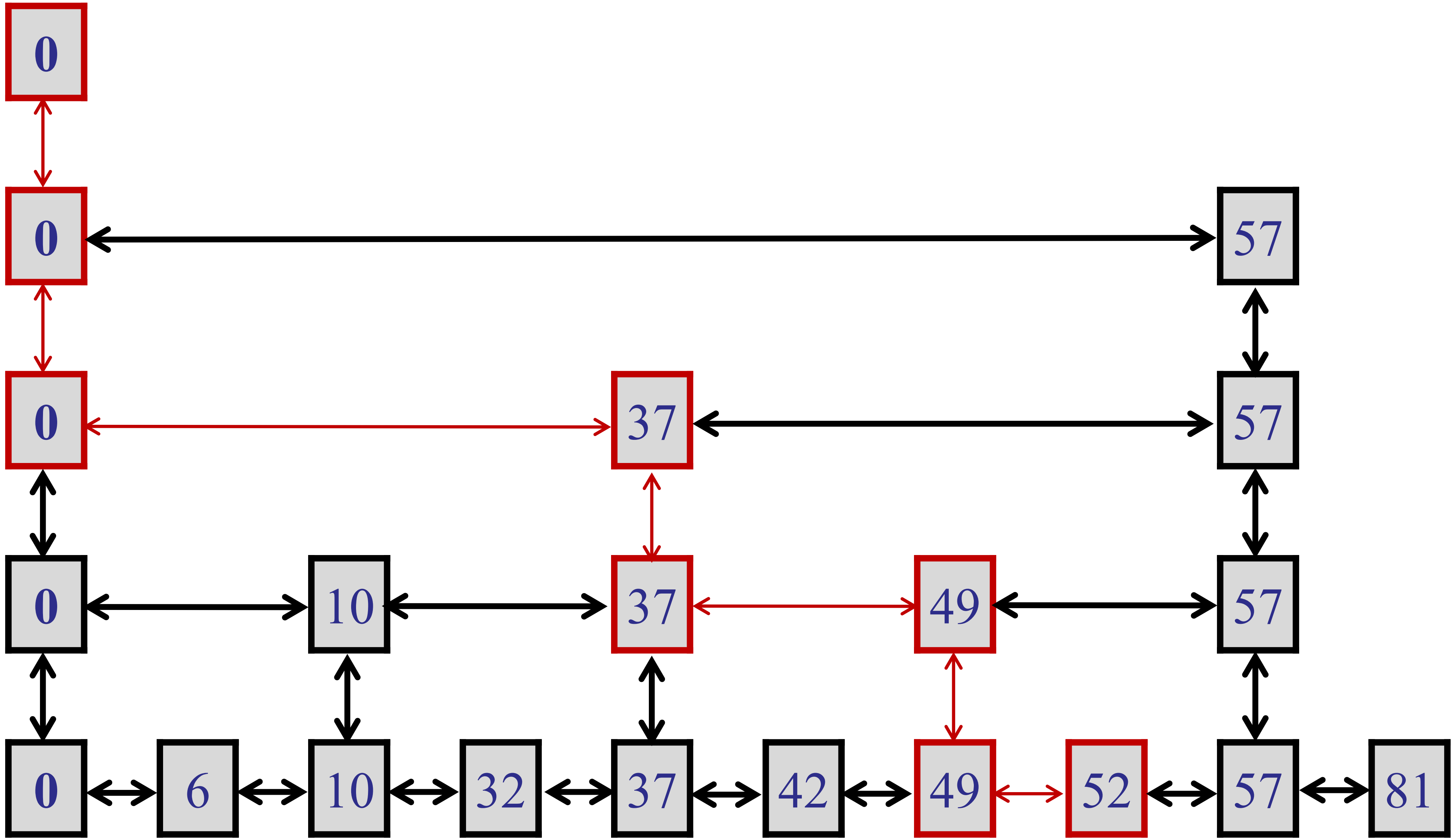
Analyzing a search:

Neat idea: analyze the search backwards.

- Start at leaf.
- For each node visited:
  - If node was not promoted (TAILS), go left.
  - If node was promoted (HEADS), go up.
- Stop at root of tree.

Exact same traversal as search, in reverse!

Example: `search(52)`



SkipList Analysis

Analyzing a search:

Neat idea: analyze the search backwards.

- Start at leaf.
- For each node visited:
  - If node was not promoted (TAILS), go left.
  - If node was promoted (HEADS), go up.
- Stop at root of tree.

Occurs at most O(log n) times!

SkipList Analysis

Analyzing a search:

Neat idea: analyze the search backwards.

- Start at leaf.
- For each node visited:
  - If node was not promoted (TAILS), go left.
  - If node was promoted (HEADS), go up.
- Stop at root of tree.

At most O(log n)

New question: How many times to flip a coin until we get c log(n) heads?

SkipList Analysis

Claim: With high probability, after 10 c log n coin flips, you get c log n heads.

SkipList Analysis

Proof:

- Say we flip 10 c log(n) coins.
- Pr[exactly c log(n) heads] =

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

Number of ways to choose
clog(n) heads out of all the flips:
TTTTHH
TTTHTH
…

SkipList Analysis

Proof:

- Say we flip 10 c log(n) coins.
- Pr[exactly c log(n) heads] =

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

Probability each of the H comes up heads.

Probability each of the T comes up tails.

SkipList Analysis

Proof:

- Say we flip 10 c log(n) coins.
- Pr[exactly c log(n) heads] =

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

bad case!

- Pr[at most c log(*n*) heads] ≤

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

If all 9clog(n) are tails, then not enough heads!

SkipList Analysis

Bounding binomials:

$$\left(\frac{y}{x}\right)^{x} \leq \binom{y}{x} \leq \left(\frac{ey}{x}\right)^{x}$$

SkipList Analysis

Bounding binomials:

$$\left(\frac{y}{x}\right)^{x} \leq \binom{y}{x} \leq \left(\frac{\mathrm{e}y}{x}\right)^{x}$$

$$\binom{10c\log n}{c\log n} \leq \left(\frac{\mathrm{e}10c\log n}{c\log n}\right)^{c\log n} \leq (10\mathrm{e})^{c\log n}$$

$$\leq n^{c\log(10\mathrm{e})}$$

SkipList Analysis

Proof:

- Say we flip 10 c log(n) coins.
- Pr[at most c log(n) heads] ≤

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

$$\leq n^{c\log(10e)}\frac{1}{n^{9c}}$$

α = c(9 – log(10) – log(e)) $\leq \dfrac{1}{n^{\alpha}}$

Generalize for other values of 10…

SkipList Analysis

Claim: With high probability, after O(log n) coin flips, you get c log n heads.

SkipList Analysis

Analyzing a search:

Neat idea: analyze the search backwards.

- Start at leaf.
- For each node visited:
  - If node was not promoted (TAILS), go left.
  - If node was promoted (HEADS), go up.
- Stop at root of tree.

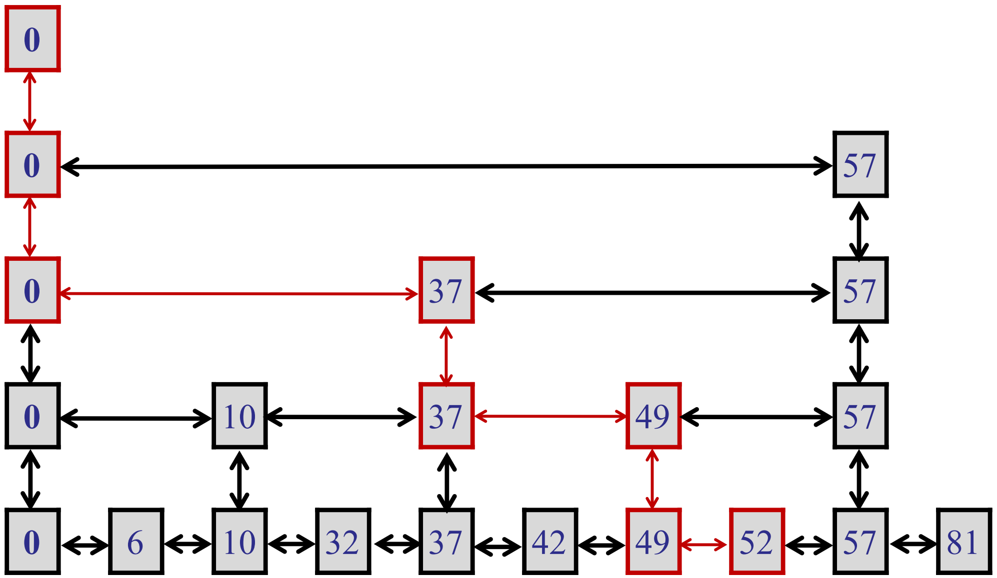
At most O(log n)

After 10 c log(n) coin flips, we will gets c log(n) heads (with high probability).

SkipList Analysis

Claim: With high probability, after O(log n) coin flips, you get c log n heads.

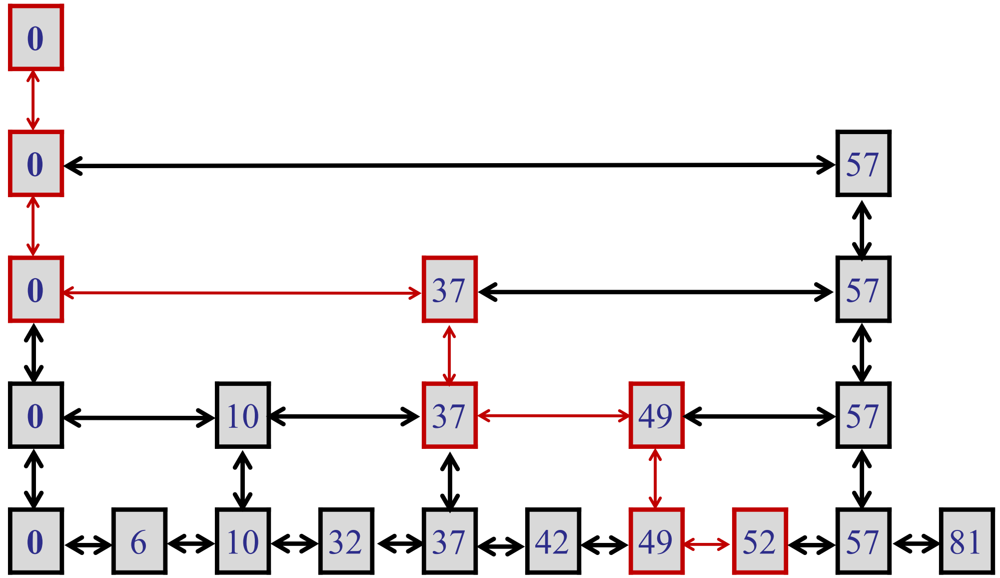
Conclusion: Each search takes O(log n) steps with high probability.



Conclusions

SkipLists

- Simple, efficient, randomized search structure.
- Easy to implement.



Analysis:

- Nice randomized calculations.
- Key idea: analyze backwards!
- Reduce to the problem of flipping coins.