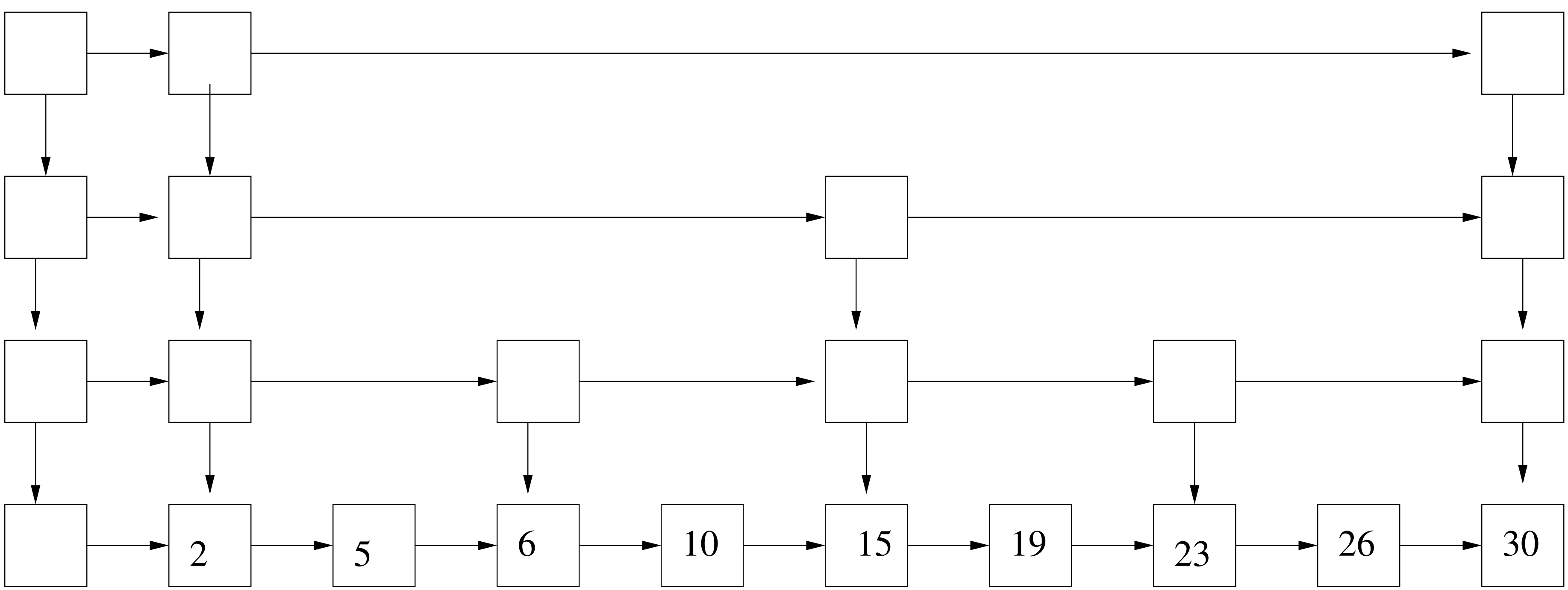
Skip Lists

Simple "balanced trees" using randomization

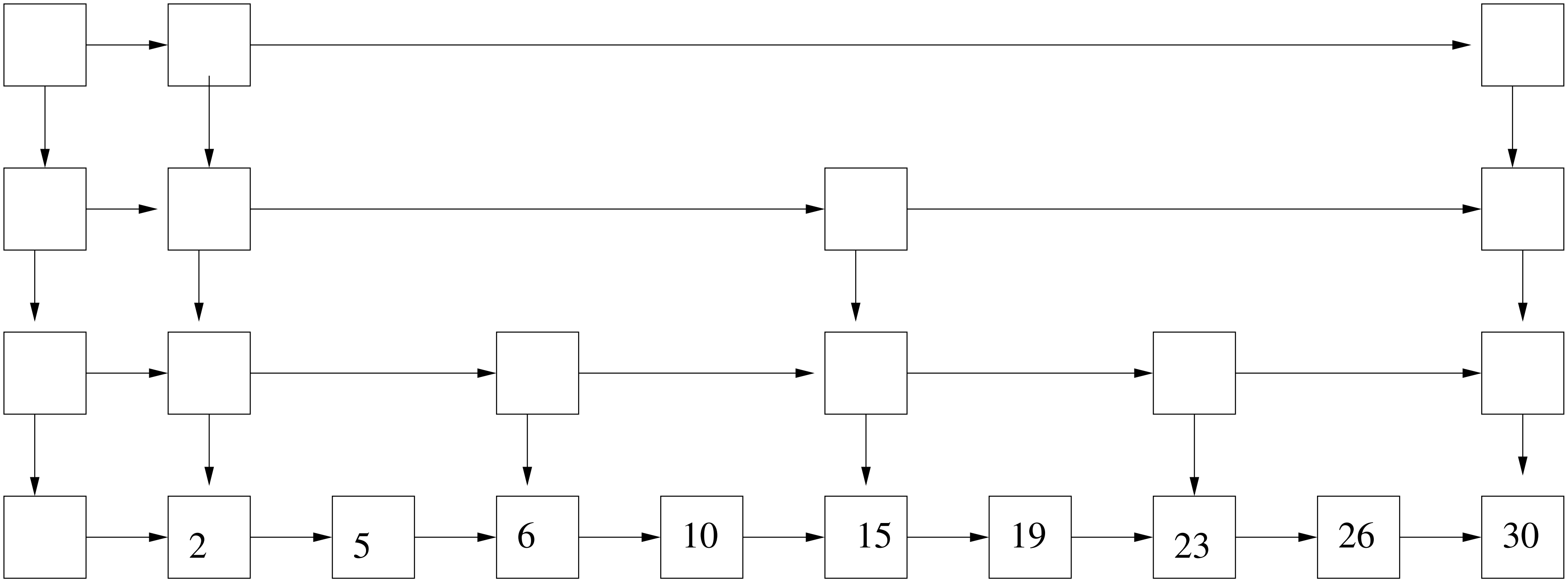
Motivation

- **Ease of coding**
- **speed (debatable)**

Starting Point **Linked Lists (slow, simple)**



Starting Point



- **Lists** $1, \ldots, \log n$
- $n/2^{\ell-1}$ **nodes in list** ℓ
- **Define the level of a node to be the highest list it is in. A node a level** i **is in lists** $1, \ldots, i$**. There are** $n/2^{i-1}$ **nodes at level** i
- **Can search in** $O(\log n)$ **time**
- **What about insert and delete?**

Idea: Maintain approximately and randomly

- **Each node j chooses a level, $v(j)$ and is then on lists $1, \ldots v(j)$.**
- **Approximately $n/2^i$ nodes at level i**
- **Let maximum level be MAXLEVEL**
- **We maintain MAXLEVEL linked lists.**

```
RANDOM-LEVEL()
level = 1
while (RANDOM(0,1) < 1/2)
        do level = level + 1
return level
```

- $\Pr(\mathit{level} = 1) = 1/2$
- $\Pr(\mathit{level} = 2) = 1/4$
- $\Pr(\mathit{level} = 3) = 1/8$
- $\Pr(\mathit{level} = i) = 1/2^i$

Code

Linked list routines

- LL-SEARCH$(L, start, x)$ - **returns the largest element $< x$ on linked list L starting from start**
- LL-INSERT$(L, start, x)$ - **inserts x into linked list L, starting from start**

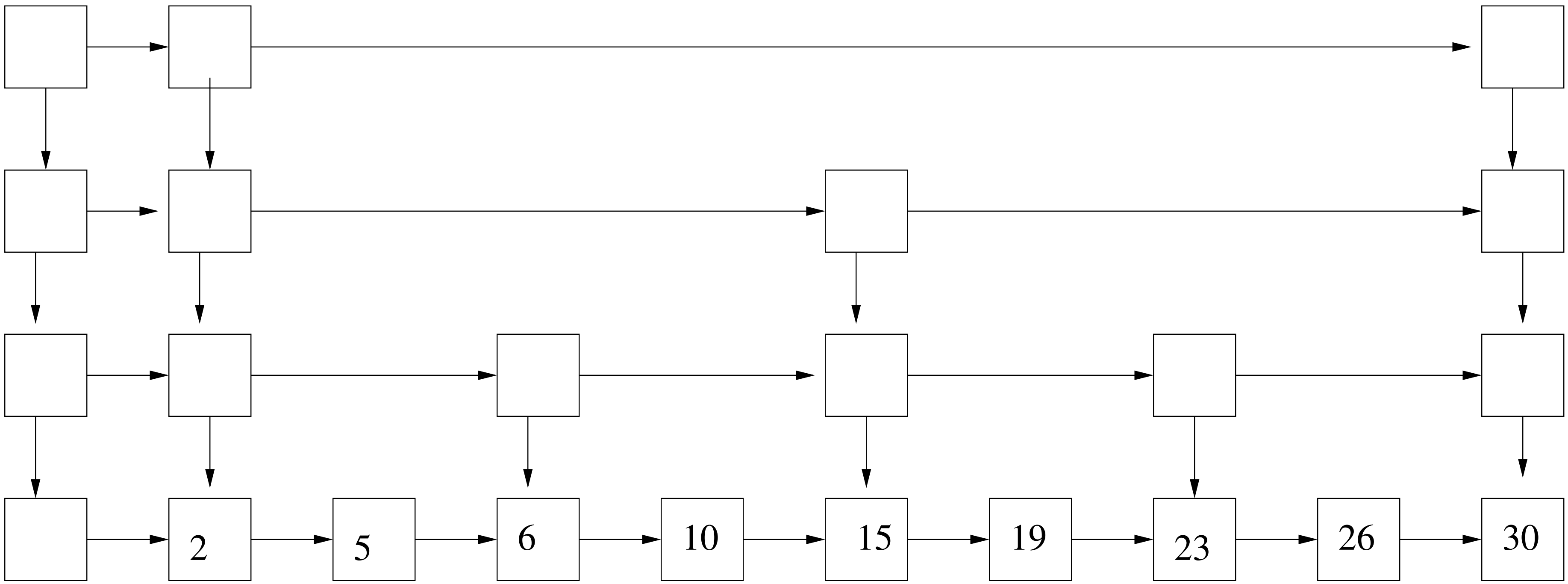
```
SEARCH(x)
p = MAXLEVEL header
for i = MAXLEVEL downto 1
        do p = LL-SEARCH(L[i], p, x)

if p → next → key = x
        do return x
    else
            return "not found"
```

```
INSERT(x)
p = MAXLEVEL header
ℓ = RANDOM-LEVEL()
for i = MAXLEVEL downto 1
        do p = LL-SEARCH(L[i], p, x)
            if (i ≤ ℓ)
                do LL-INSERT(L[i], p, x)
```

Code

Delete Similar to insert

Running Time Big-O of $MAXLEVEL$ + the time to do all the searches. (Total down moves plus right moves).



Analysis

Expected number of moves per list

$$(1/2)1 + (1/4)2 + (1/8)3 + \ldots \leq 2$$

Total time is therefore $O(\log n)$