



COMP 482 / ELEC 420

Skip Lists

Reading

- Skim [CLRS] 10, 12, 13 – background & related.
- Read “Skip Lists: A Probabilistic Alternative to Balanced Trees” by Pugh, 1990. [Skip list paper](#)

Review

Should be familiar with many kinds & uses of trees, e.g.,

- Binary search trees
- Balanced trees
- Heaps

- Syntax trees
- Spanning trees

This course:

- More search trees
- More heaps
- Tries

Balanced Trees Overview

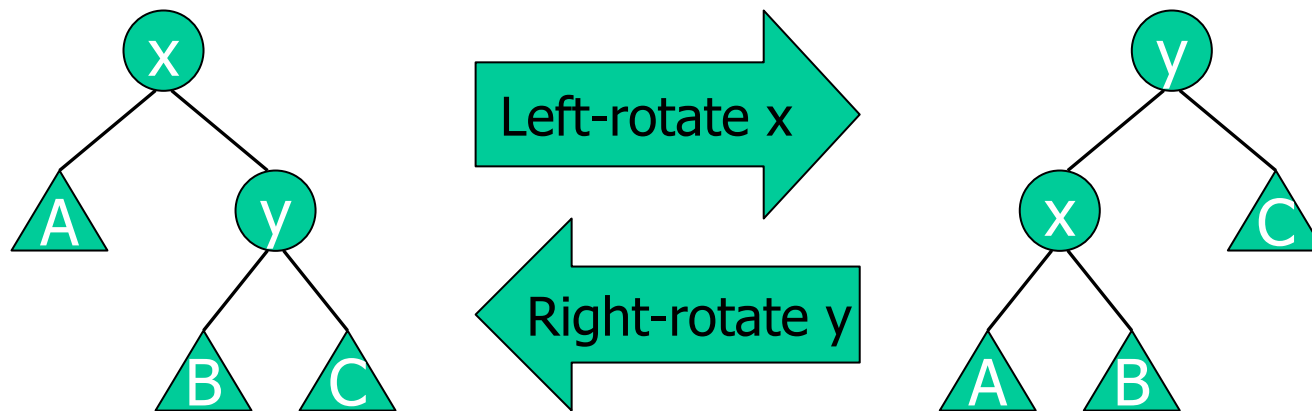
Many variations, but common implementation themes.

Most interested in "dictionaries":

- Create, insert, delete, search
- Thus, frequently are variations on binary search trees.
- Want $O(\log n)$ per operation.

Implementation often uses "rotations":

- Maintains the search-tree ordering: $A < x < B < y < C$
- Can change tree depth, based on depths on A, B, C



Background: Some Balanced Search Tree Variations

- Randomly-built BST
 - Reasonably balanced in expected case
 - Only feasible if all inserts happen first, & can randomize their relative order.
- AVL tree (1962)
 - BST
 - Guarantees depths of all leaves differ by at most 1
 - Balancing requires $O(\log n)$ rotations.
- Red-Black tree (1972)
 - BST
 - Guarantees depth of all leaves within a factor of 2
 - Balancing requires at most 2 rotations.
- AA-tree (1996)
 - Easier-to-code variant of Red-Black tree

Background: Some Balanced Search Tree Variations

- 2-3 tree (1970)
 - Search tree, where nodes have either 2 or 3 children and 1 or 2 keys, respectively
 - Guarantees depth of all leaves within a constant factor
 - Isomorphic to AA-tree
- 2-3-4 tree
 - Generalization of 2-3 trees
 - Guarantees depth of all leaves equal
 - Lower constant costs than 2-3 trees, because only makes one pass on tree, instead of two, on each operation
 - Isomorphic to Red-Black tree
- B-tree (1972)
 - 2-3-...-n tree
 - Bushy → increase spatial locality
 - Shallow → minimize the number of node accesses
 - Most suitable for huge data sets, as in databases

Background: Some Search Tree Variations

- Treap (1980/1989)
 - BST also maintaining heap property
 - Conceptually simple
 - Low constant costs, but logarithmic bounds only expected
- Skip list (1990)
 - Singly-linked sorted list, with extra pointers to later in list
 - Conceptually fairly simple
 - High space overhead for lots of pointers, and logarithmic bounds only expected
- Splay tree (1983)
 - Self-adjusting, i.e., changes tree even on non-insert/delete operations, so as to improve later accesses
 - Logarithmic bounds only when amortized
- Trie (1960)
 - Specialized to strings
 - Linear time in length of input
 - Many variations

Some Applets

AVL

<http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html>

Red-Black

<http://www.ececs.uc.edu/~franco/C321/html/RedBlack/redblack.html>

2-3-4

<http://www.cs.unm.edu/~rlpm/499/ttft.html>

Treap

http://babbage.clarku.edu/~achou/cs160fall03/examples/bst_animation/Treap-Example.html

BST, AVL, B-Tree, Red-Black, AA, Skip list, Heap, Treap, Scapegoat, Splay

<http://people.ksp.sk/~kuko/bak/>

BST, AVL, Red-Black, AA, Splay, Tries, Patricia Tries

<http://www.cis.ksu.edu/~howell/viewer/viewer.html>

Skip Lists: Overview

Intuitive idea, but trickier details.

Logarithmic bounds only expected.

Analysis is a useful example.

Linear in worst-case.

Intuition

Worst-case search

Start with singly-linked sorted list.

n

Add ptrs between every 2nd element.

$\Theta(n/2)$

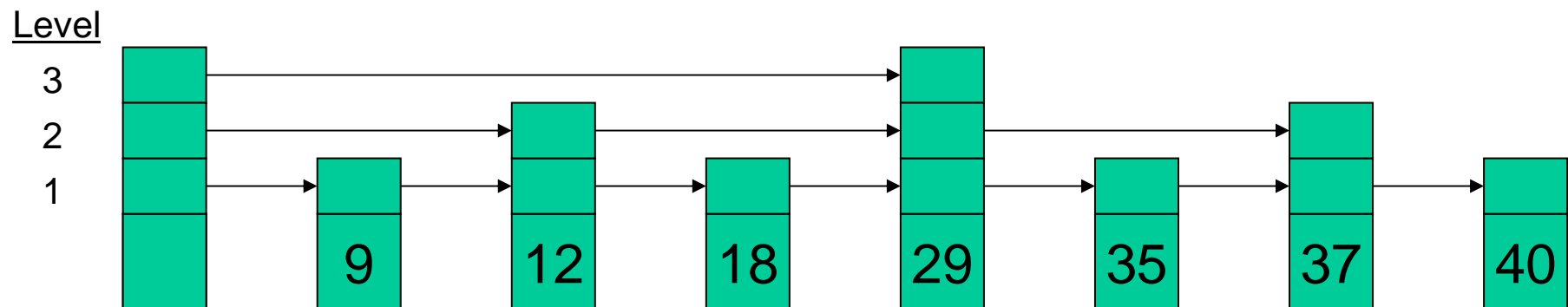
Add ptrs between every 4th element.

$\Theta(n/4)$

Add ptrs between every 2ⁱ-th element, $\forall i$.

$\Theta(\lg n)$

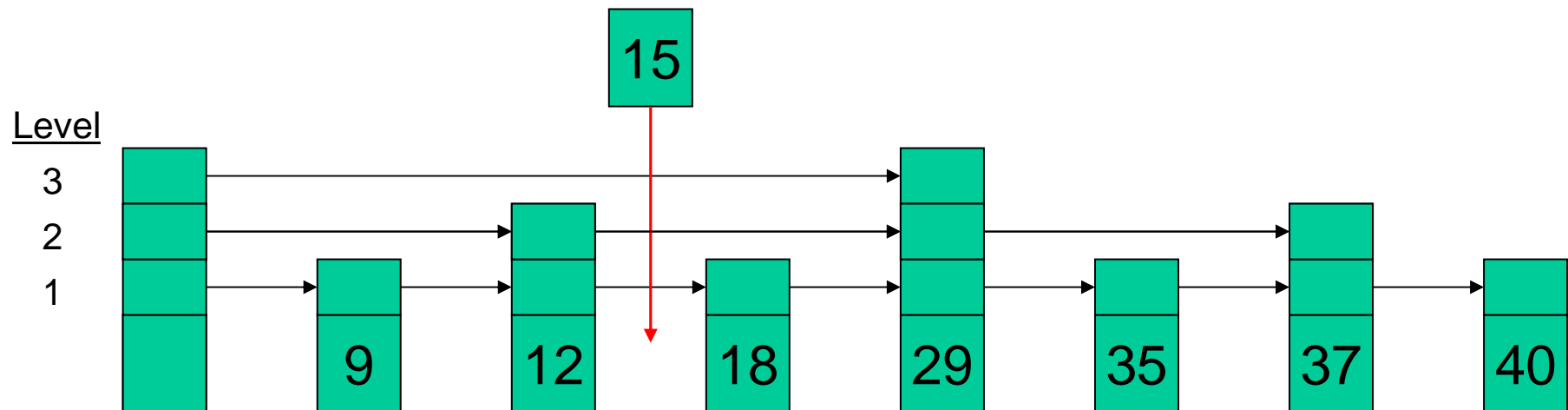
Behaves just like a BST.



Intuition

Problem:

Rigid location of pointers difficult to maintain when inserting/deleting.



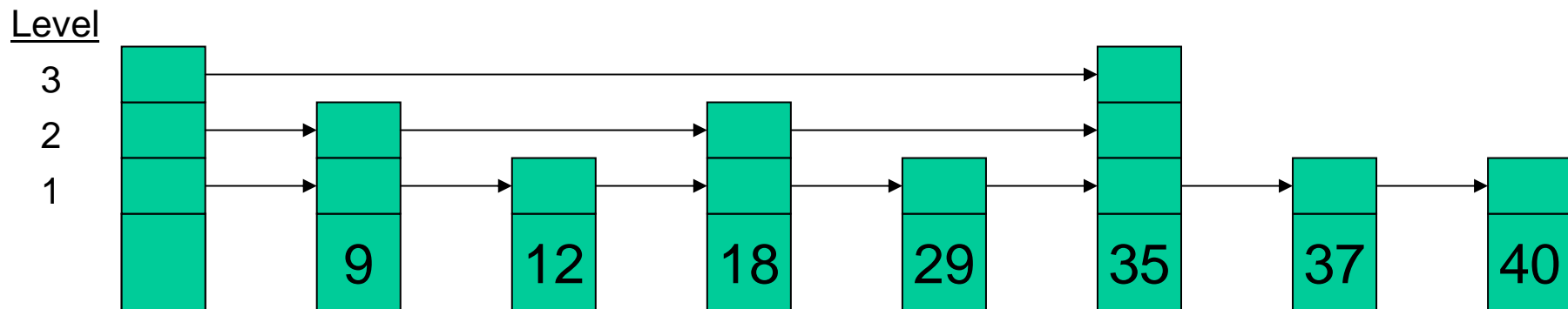
Intuition

Solution:

Only keep the right **proportions** of pointers.

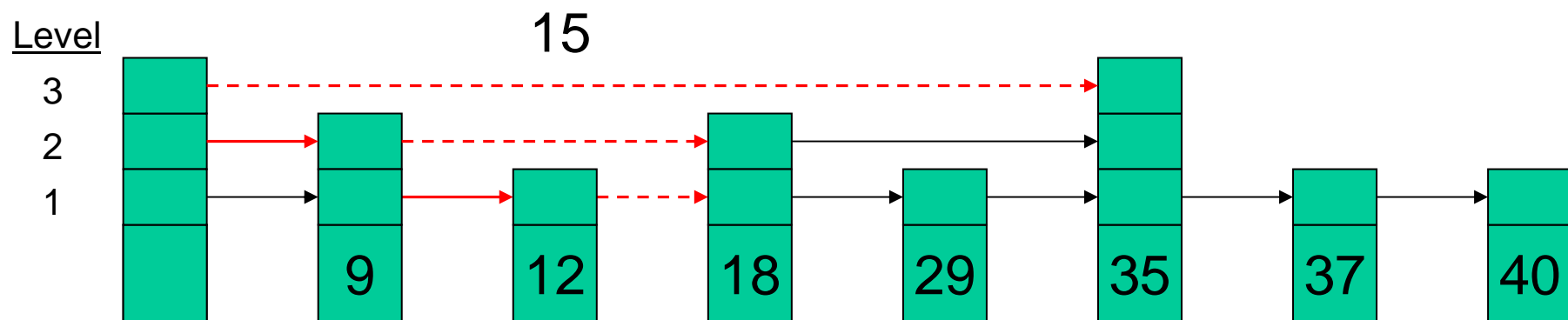
Each node is at a particular level:

100% level 1, 50% level 2, 25% level 3, ..., $1/2^i$ level $i+1$



Search

1. Using top level links, search.
2. For each next level, search – bounded by previous.



Informal Expected-Case Analysis

of next-lower-level nodes
skipped per link = ?

Expected: 2

of levels = ?

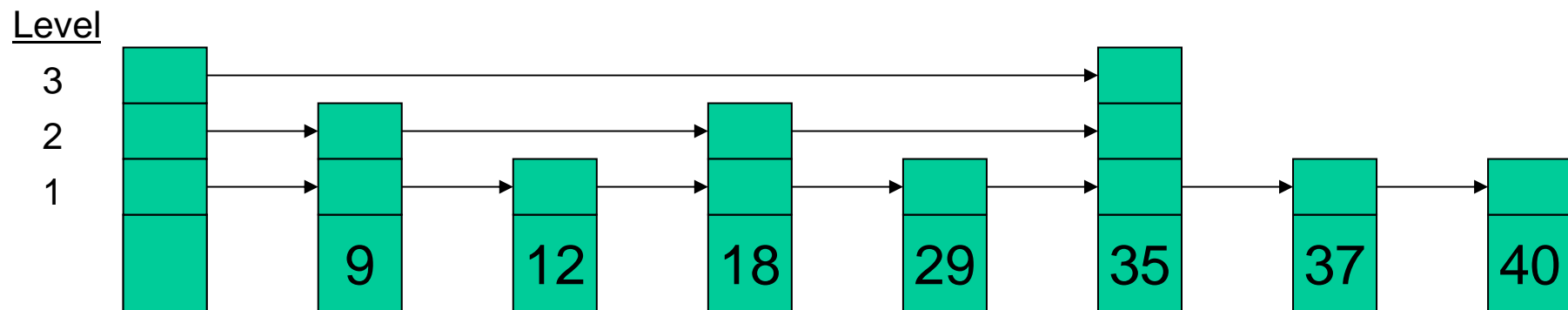
Expected: $\lg n$

Search:

Search of top level – $\Theta(1)$

Bounded search of each next level – $\Theta(1)$ per level

Total = $\Theta(\log n)$



Informal Expected-Case Analysis

Used probability $p=1/2$ of adding link to next level.

What if we generalize?

of next-lower-level nodes
skipped per link = ?

Expected: $1/p$

of levels = ?

Expected: $\log_{1/p} n$

Search:

Search of top level – $\Theta(1)$

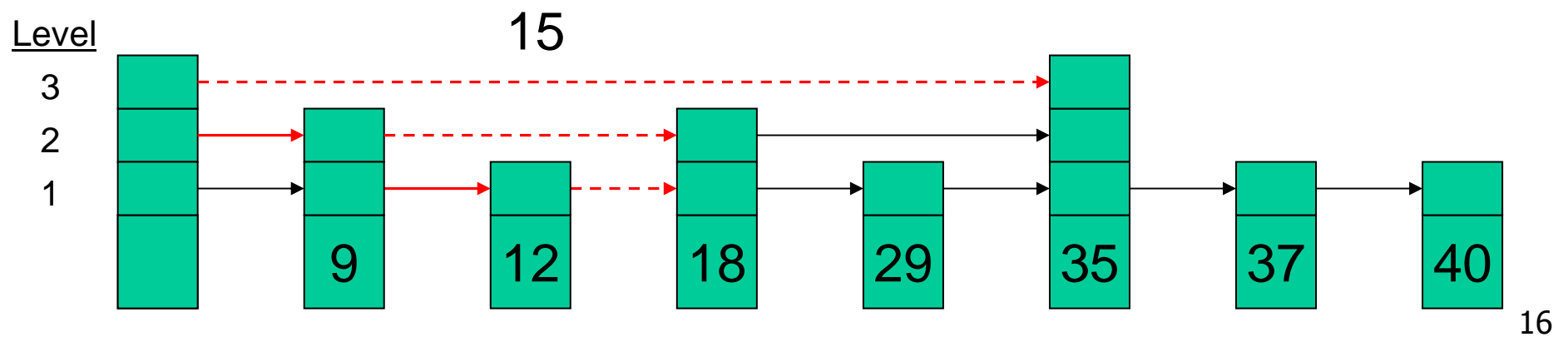
Bounded search of each next level – $\Theta(1)$ per level

Total = $\Theta(\log n)$

Soon: More formal, more precise, & pick optimal p .

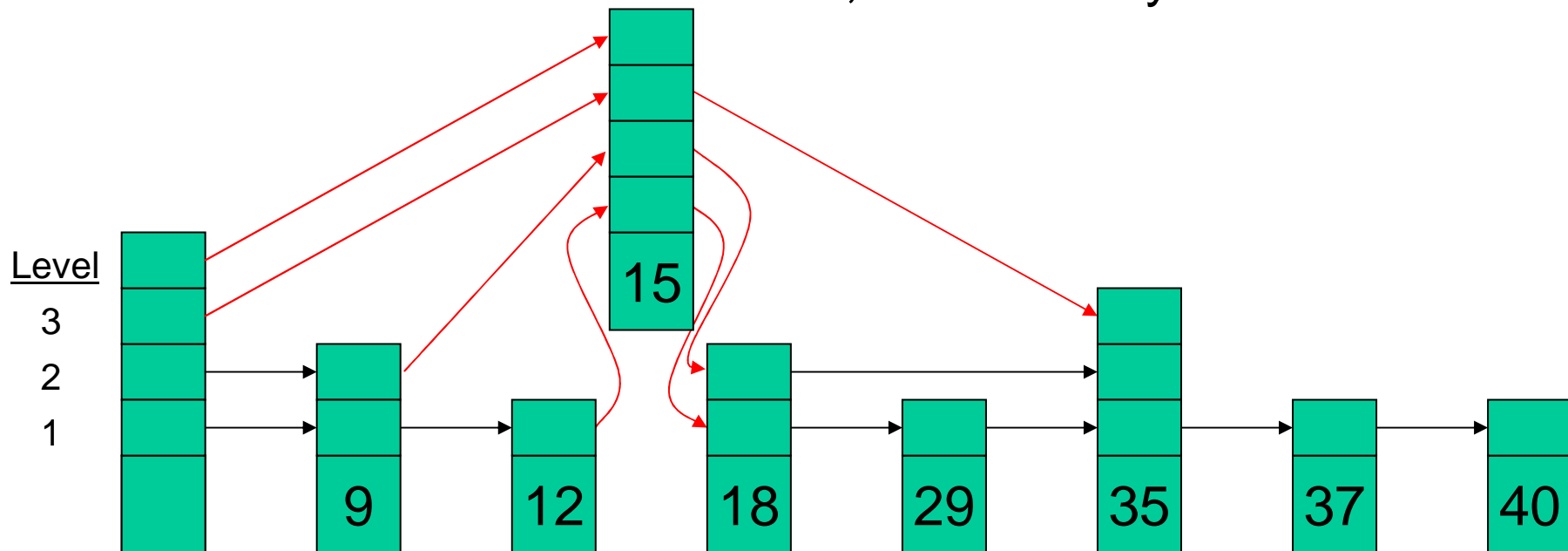
Insert

1. Search for where n belongs.
 - Remember last link at each level.
 - Exit, if value found.



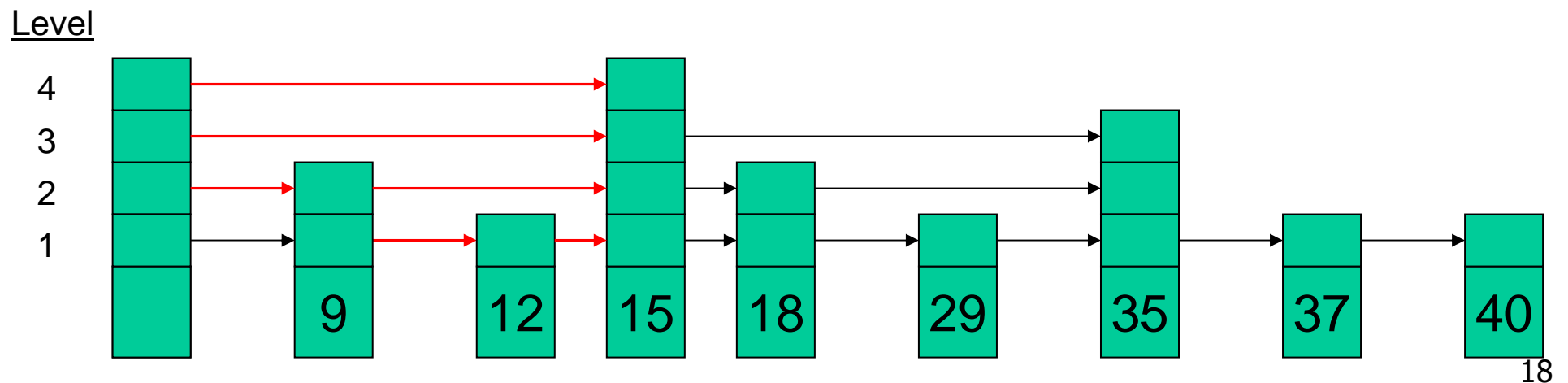
Insert

1. Search for where n belongs.
 - Remember last link at each level.
 - Exit, if value found.
2. Create node. Choose its level k probabilistically.
3. For $i=1, \dots, k$, replace last links.
 - Increase head node's level, if necessary.



Delete

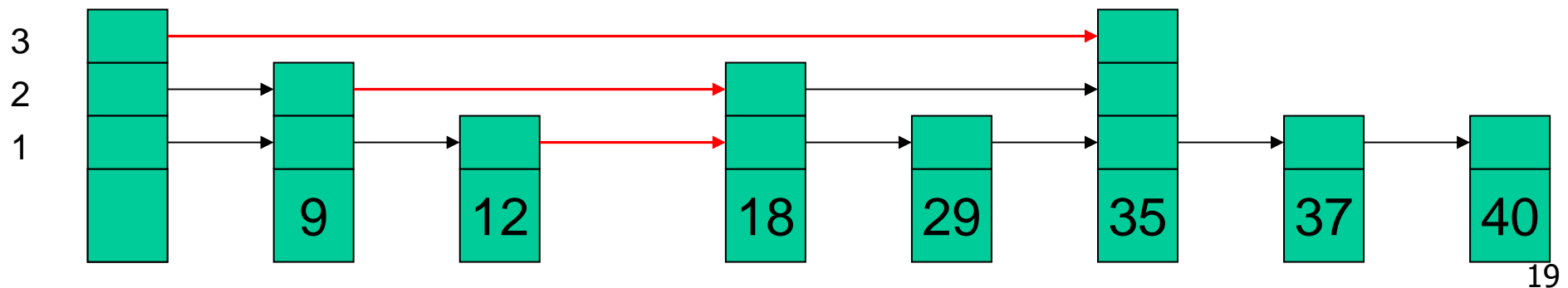
1. Search, forcing search to go on each level.
 - Remember last link on each level.
 - Exit, if value not found.



Delete

1. Search, forcing search to go on each level.
 - Remember last link on each level.
 - Exit, if value not found.
2. Replace last links.
 - Decrease start node's level, if necessary.
3. Delete node.

Level

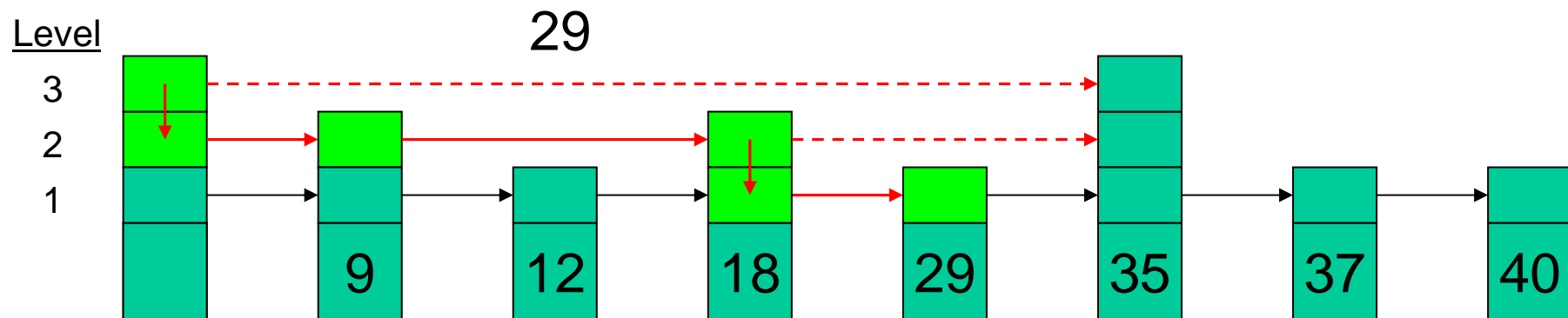


Formal Expected-Case Analysis of Search

We'll analyze a search "backwards".
From the item upwards & leftwards.

$C(k)$ = expected length of search path, when climbing k levels

Equivalent to count either \downarrow or \dashrightarrow .

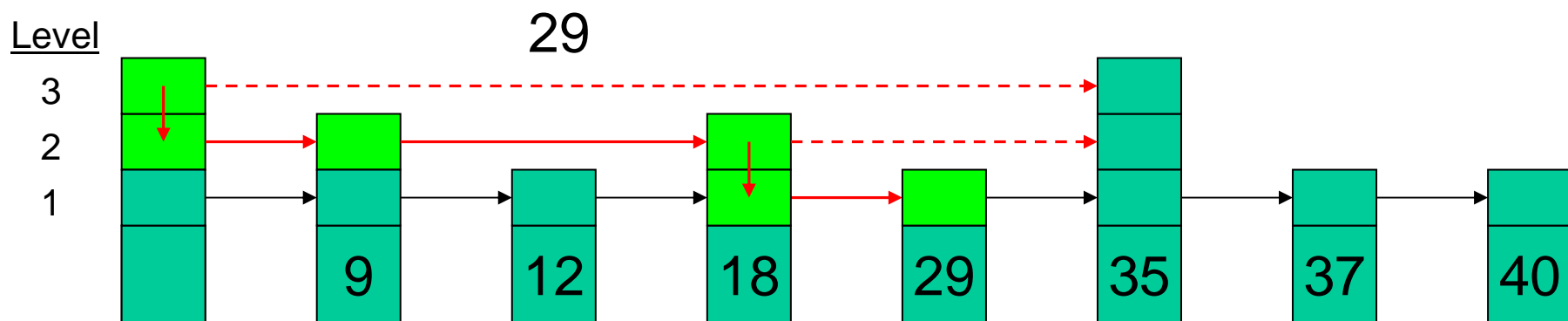


Formal Expected-Case Analysis of Search

$C(k)$ = expected length of search path, when climbing k levels

Base case:

$$C(0) = 0$$

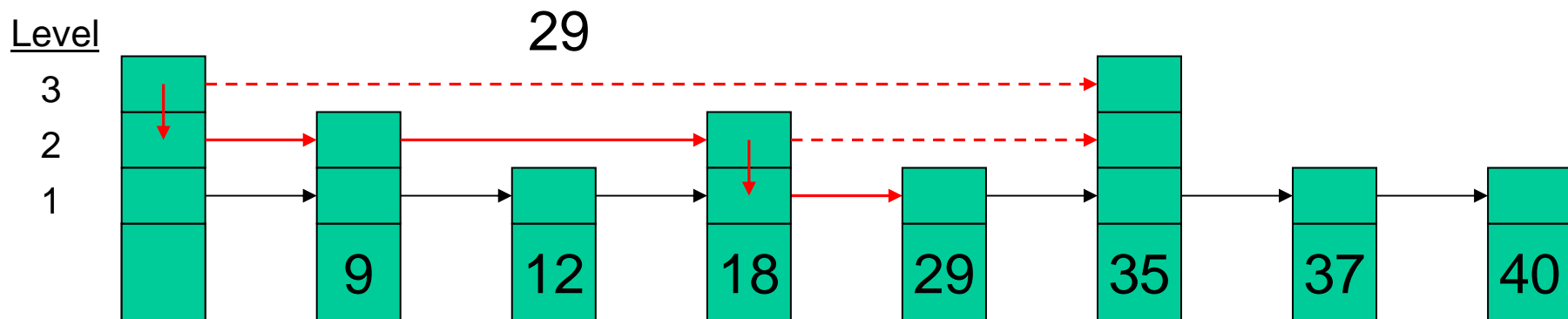


Formal Expected-Case Analysis of Search

$C(k)$ = expected length of search path, when climbing k levels

Inductive case:

$$C(k) = \text{prob?} \times \text{cost?} + \text{prob?} \times \text{cost?}$$

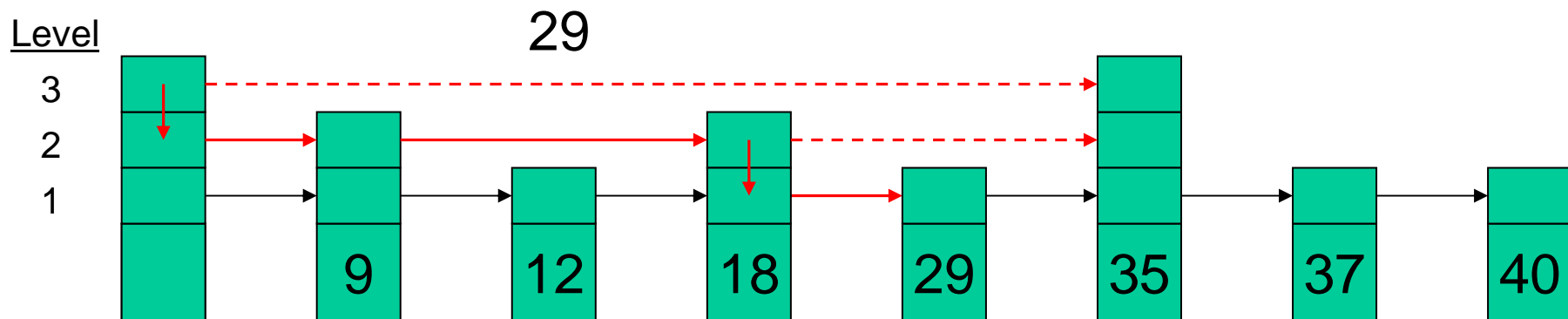


Formal Expected-Case Analysis of Search

$C(k)$ = expected length of search path, when climbing k levels

Inductive case:

$$C(k) = (1-p) \times (1+C(k)) + p \times (1+C(k-1))$$



Formal Expected-Case Analysis of Search

$C(k)$ = expected length of search path, when climbing k levels

Inductive case:

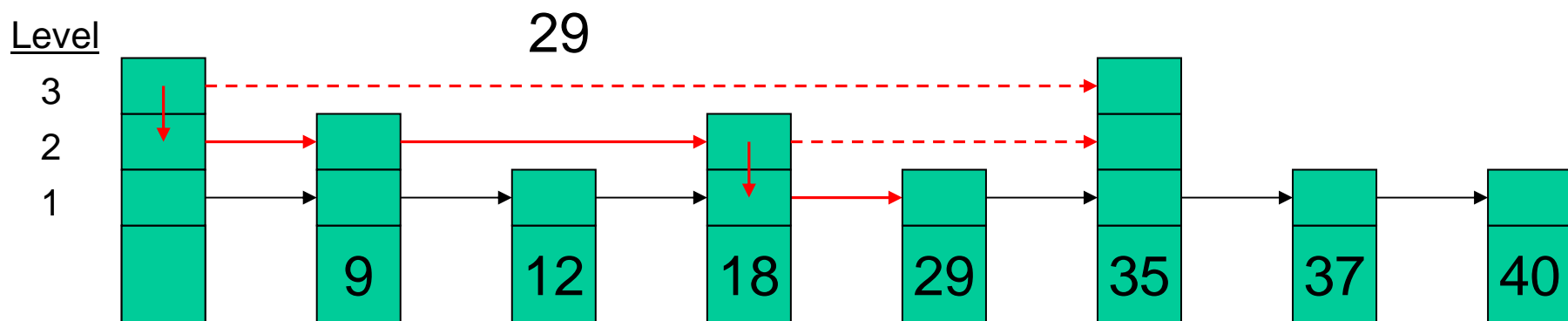
$$C(k) = (1-p) \times (1+C(k)) + p \times (1+C(k-1))$$

$$= 1 + C(k) - p - pC(k) + p + pC(k-1)$$

$$0 = 1 - pC(k) + pC(k-1)$$

$$C(k) = 1/p + C(k-1)$$

$$= k/p$$

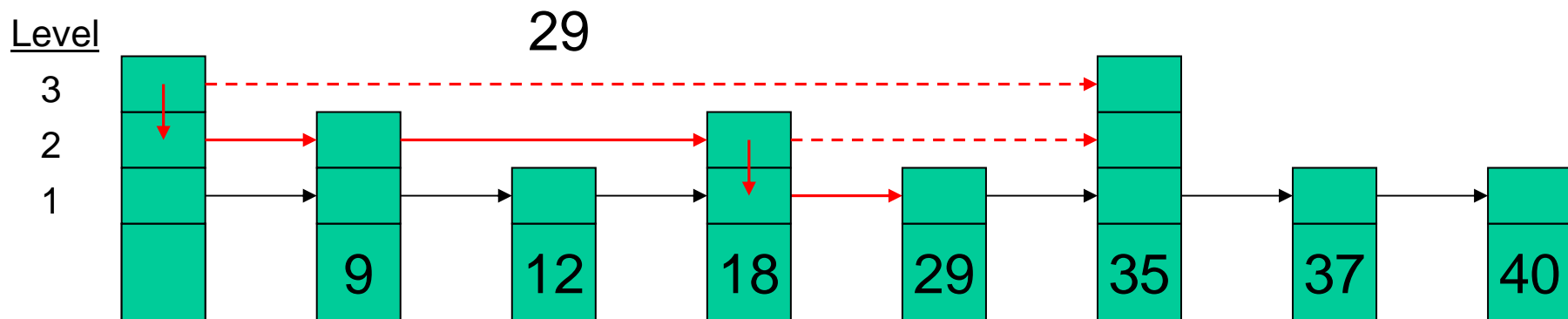


Formal Expected-Case Analysis of Search

How many levels? Intuition: $\log_{1/p} n$

But, could (unluckily) have a very tall node. How likely?

$$\begin{aligned} \Pr\{\text{node } x \text{ is at or above level } i\} &= p^{i-1} \\ \Pr\{\text{node } x \text{ is at or above level } 1+2\log_{1/p} n\} &= p^{2\log_{1/p} n} = 1/n^2 \\ \Pr\{\exists \text{ node at or above level } 1+2\log_{1/p} n\} &= n \times 1/n^2 = 1/n \\ \Pr\{\neg \exists \text{ node at or above level } 1+2\log_{1/p} n\} &= 1 - 1/n \end{aligned}$$



Formal Expected-Case Analysis of Search

Expected length of search path =

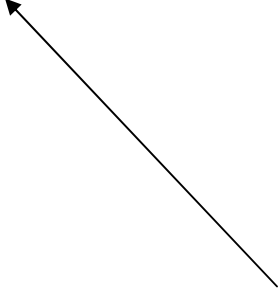
$$\frac{1 + 2\log_{1/p} n}{p}$$

“with high probability”.

Minimized by $p=1/e$.



With probability $1 - 1/n^c$,
for some constant c .



In Practice

- Numerous implementation choices, e.g., how to represent nodes.
- Time – Pretty good, fairly low constants.
- Space – OK. More pointers & larger nodes than many comparable balanced trees.