CSE 373
Data Structures and Algorithms

Lecture 25: B-Trees

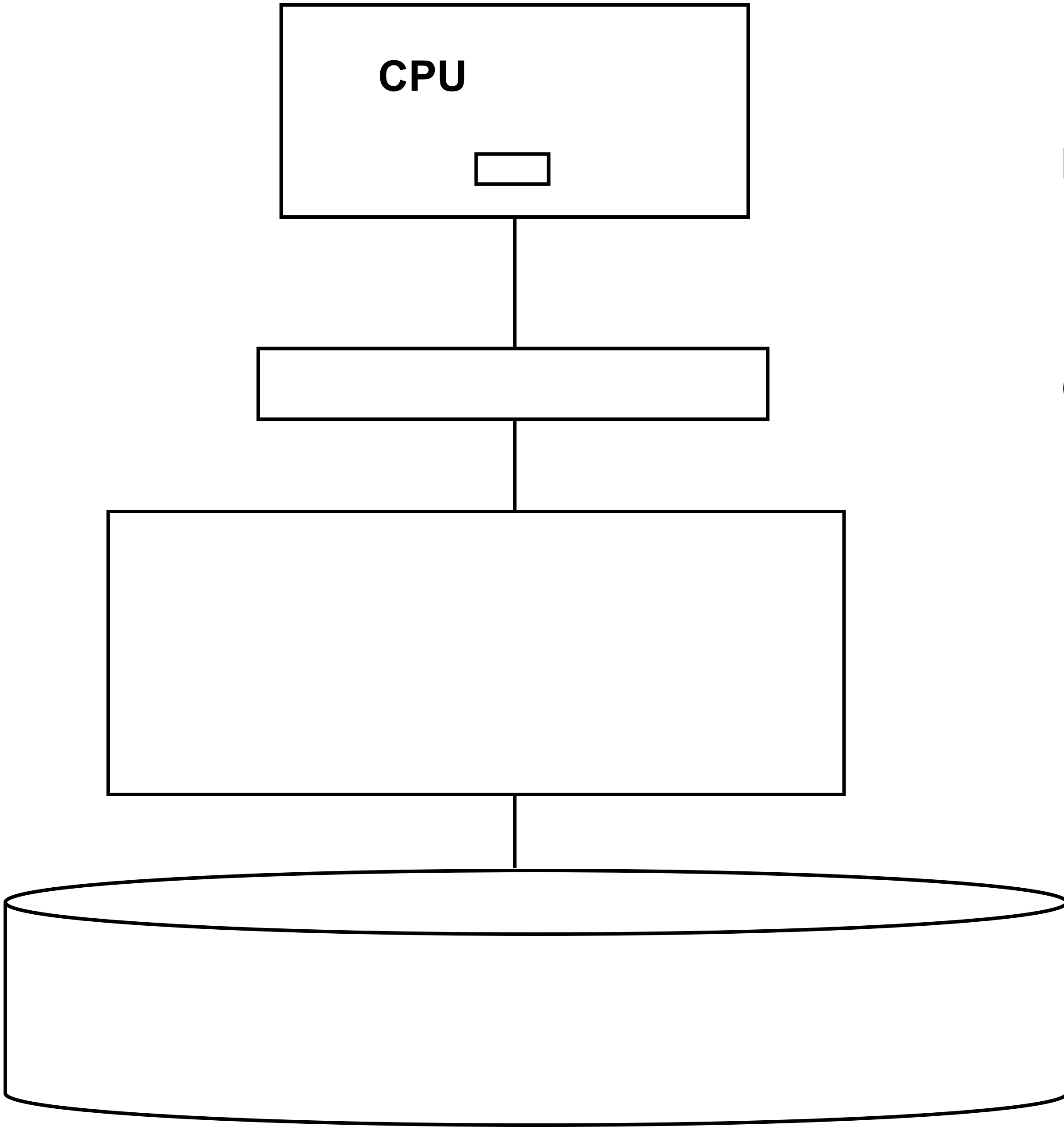
Assumptions of Algorithm Analysis

- Big-Oh assumes all operations take the same amount of time
  - Is this really true?

Disk Based Data Structures

- All data structures we have examined are limited to main memory
  - Have been assuming data fits into main memory

- Counter-example: Transaction data of a bank > 1 GB per day
  - Uses secondary storage (e.g., hard disks)
  - Operations: insert, delete, searches



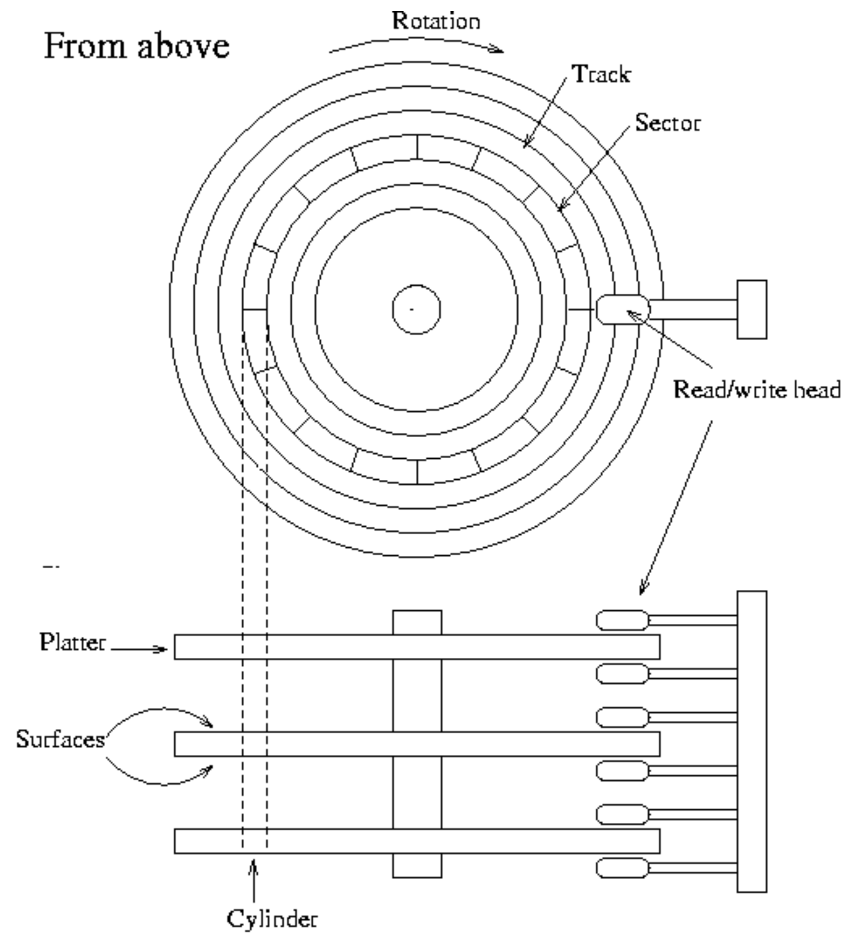
Cycles* to access:

Registers	**1**
Cache	**tens**
Main memory	**hundreds**
Disk	**millions**

***cycle**: time it takes to execute an instruction

Hard Disks

- Large amount of storage but slow access
- Identifying a particular block takes a long time
  - Read or write data in chunks ("a page") of 0.5 – 8 KB in size
- (Newer technology) Solid-state drives are 50 – 100 times faster
  - Still "slow"



Algorithm Analysis

- Running time of disk-based data structures measured in terms of
  - Computing time (CPU)
  - Number of disk accesses

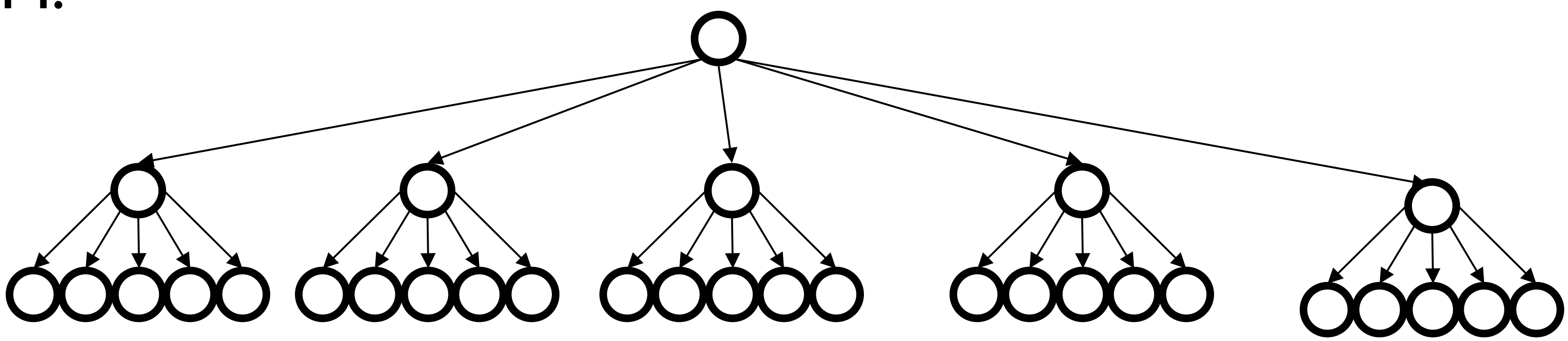
- Regular main-memory algorithms that work one data element at a time can not be "ported" to secondary storage in a straight forward way

Principles

- Almost all of our data structure is on disk.
- Every time we access a node in the tree it amounts to a random disk access.
- How can we address this problem?

M-ary Search Tree

- Suppose we devised a search tree with branching factor M:

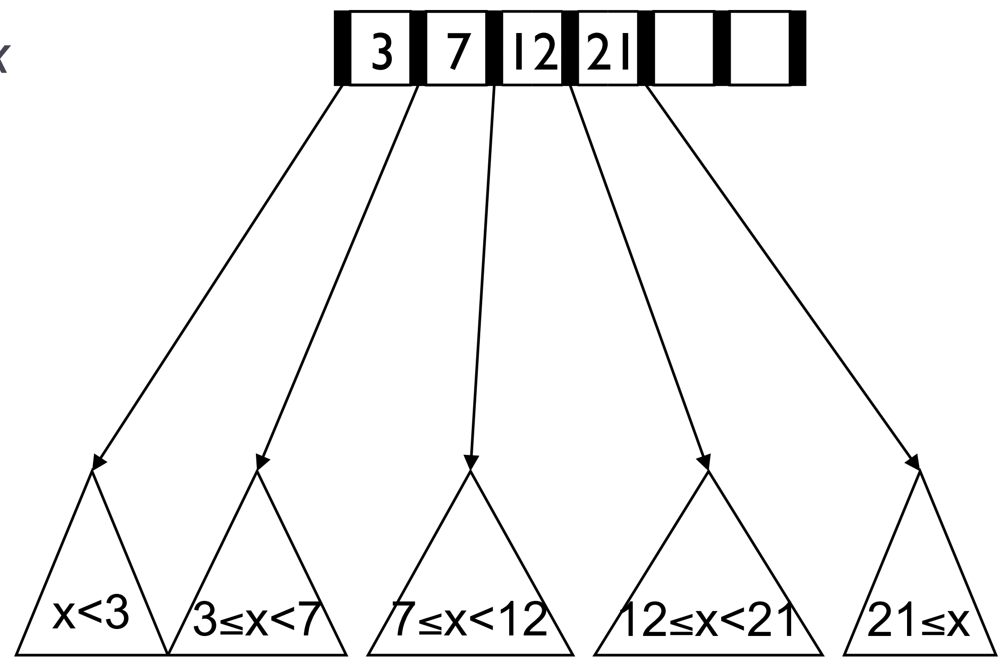


- M – 1 keys needed to decide branch to take
- Complete tree has height: $\Theta(\log_M n)$
- # Nodes accessed for search: $\Theta(\log_M n)$

B-Trees

- Internal nodes store (up to) M - 1 keys
- Order property:
  - Subtree between two keys *x* and *y* contain leaves with *values* *v* such that $x \leq v < y$
  - Note the "≤"
- Leaf nodes contain up to L sorted values/ data items ("records").

M = 7



B-Tree Structure Properties

- Root (special case)
  - Has between 2 and M children (or could be a leaf)

- Internal nodes
  - Stores up to M-1 keys
  - Have between ceiling(M/2) and M children

Nodes are at least ½ full

- Leaf nodes
  - Where data is stored
  - Contains between ceiling(L/2) and L data items

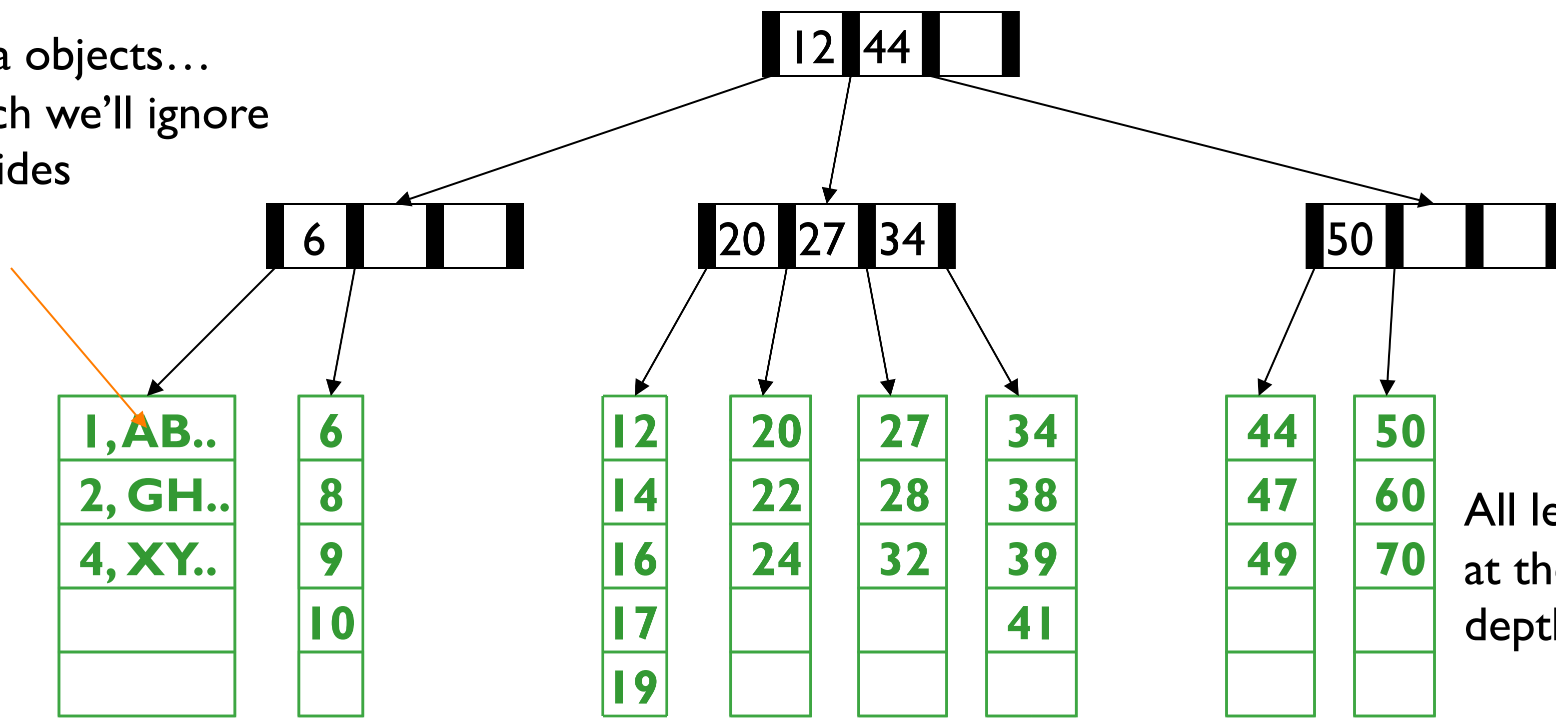
Leaves are at least ½ full

The tree is ***perfectly balanced*** !

B-Tree: Example

- B-Tree with M = 4 (# pointers in internal node)
  and L = 5 (# data items in leaf)

Data objects…
which we'll ignore
in slides



All leaves
at the same
depth

Definition for later: "neighbor" is the next sibling to the left or right.

Disk Friendliness

- Many keys stored in a node
  - Each node is one disk page/block.
  - All brought to memory/cache in one disk access.

- Internal nodes contain only keys; only **leaf nodes** contain actual data

- What is limiting you from increasing the number of keys stored in each node?
  - The page size

Exercise: Computing M and L

- Exercise: If disk block is 4000 bytes, key size is 20 bytes, pointer size is 4 bytes, and data/value size is 200 bytes, what should M (# of branches) and L (# of data items per leaf) be for our B-Tree?

  Solve for M:

  M - 1 keys + M pointers = 20M - 20 + 4M = 24M – 20

  24M - 20 <= 4000

  M = 167

  Solve for L:

  L = 4000 / 200

  L = 20

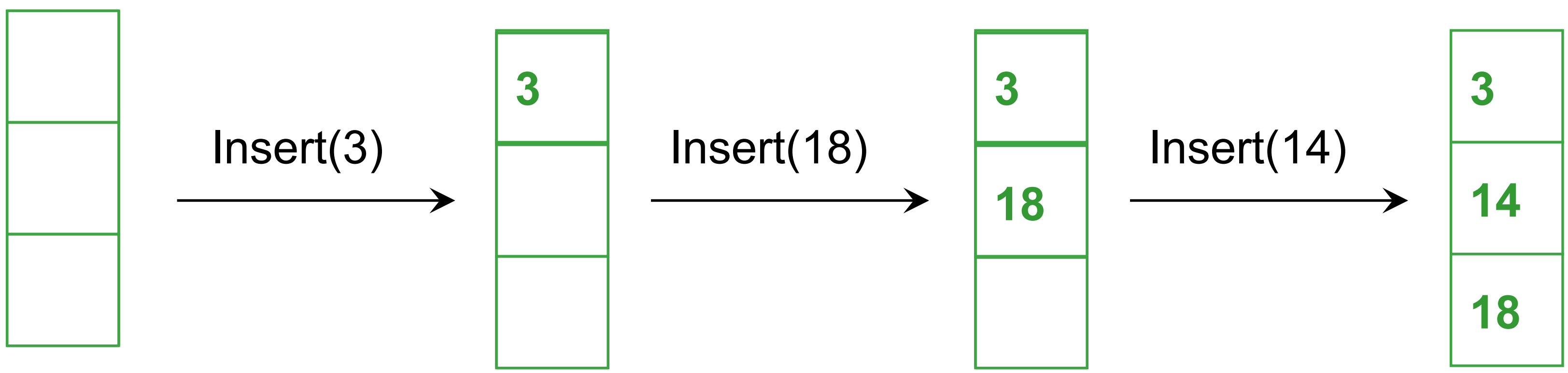
B-trees vs. AVL trees

Suppose we have n = 10^9 (billion) data items:

- Depth of AVL Tree: $\log_2 10^9 = 30$
- Depth of B-Tree with M = 256, L = 256:
  $\sim \log_{M/2} 10^9 = \log_{128} 10^9 = 4.3$
  (M/2 because keys half-full)

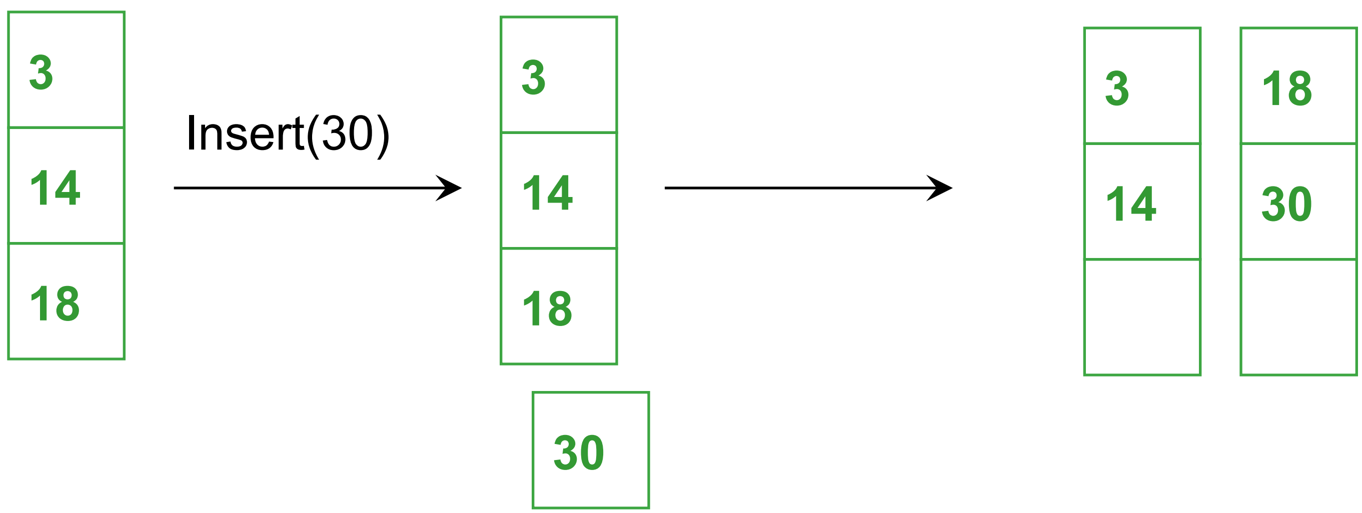
14

Building a B-Tree with Insertions

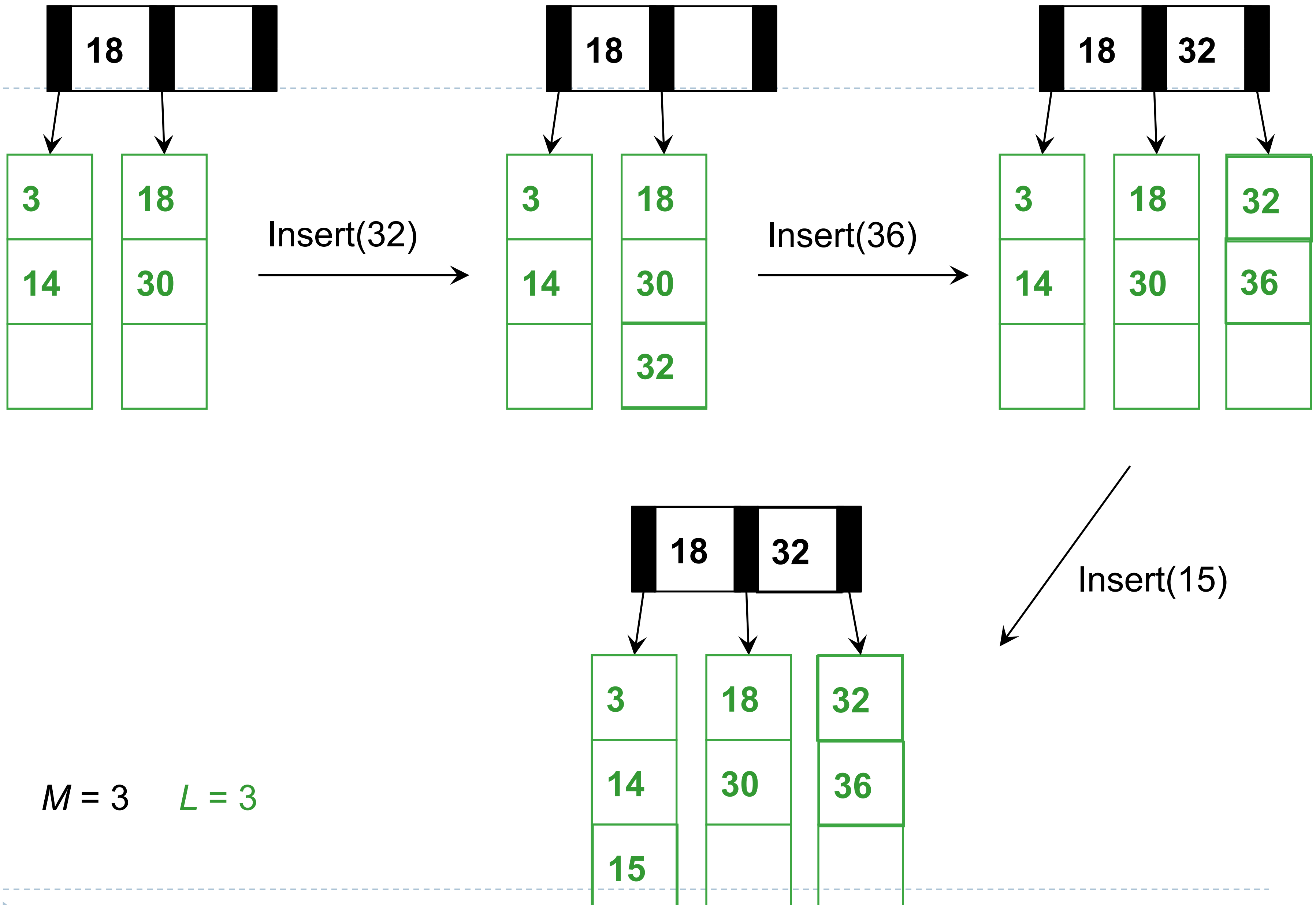


The empty B-Tree

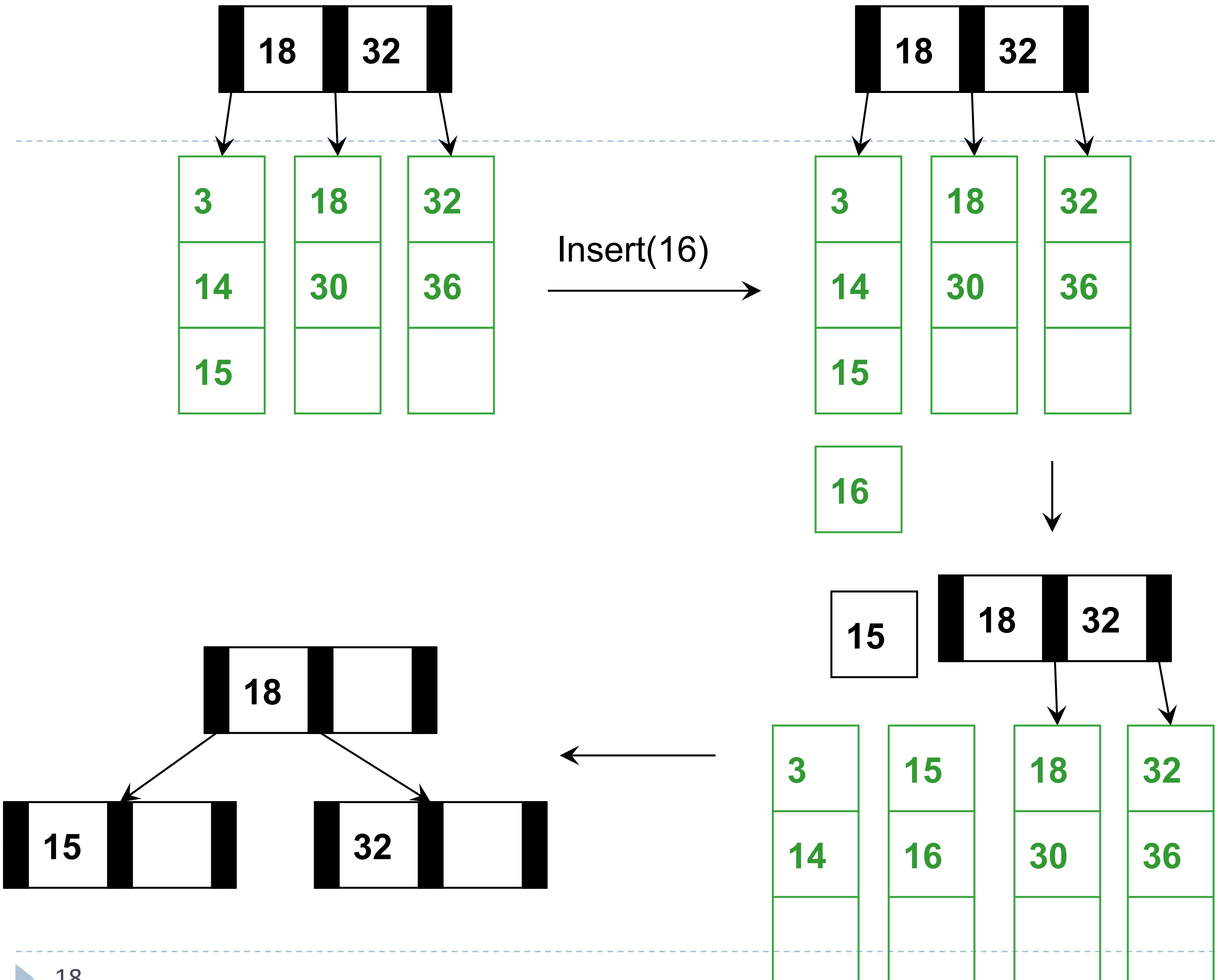
$M = 3$ $L = 3$



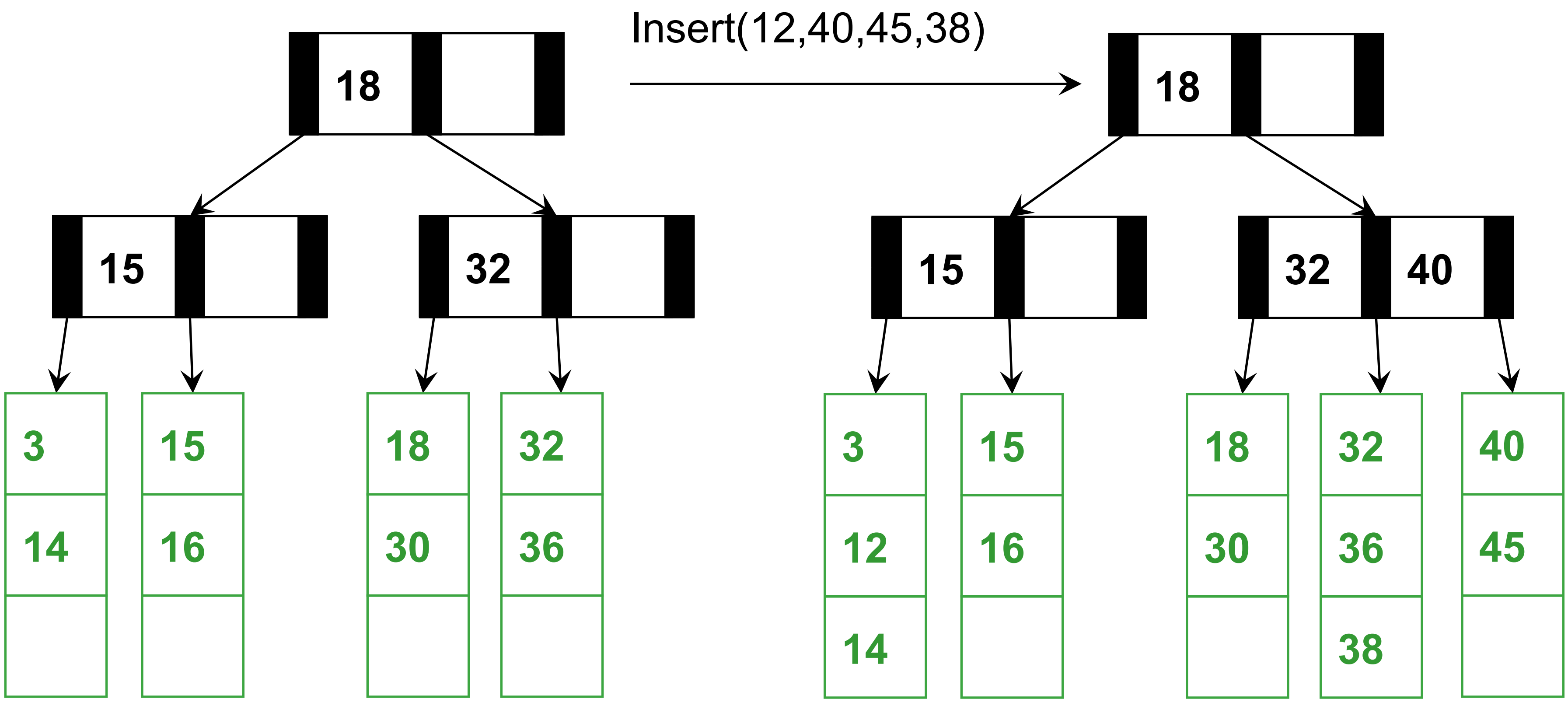
M = 3 *L* = 3


18
3
14
18
30
Insert(32)
18
3
14
18
30
32
Insert(36)
18
32
3
14
18
30
32
36
Insert(15)
18
32
3
14
15
18
30
32
36

M = 3 *L* = 3

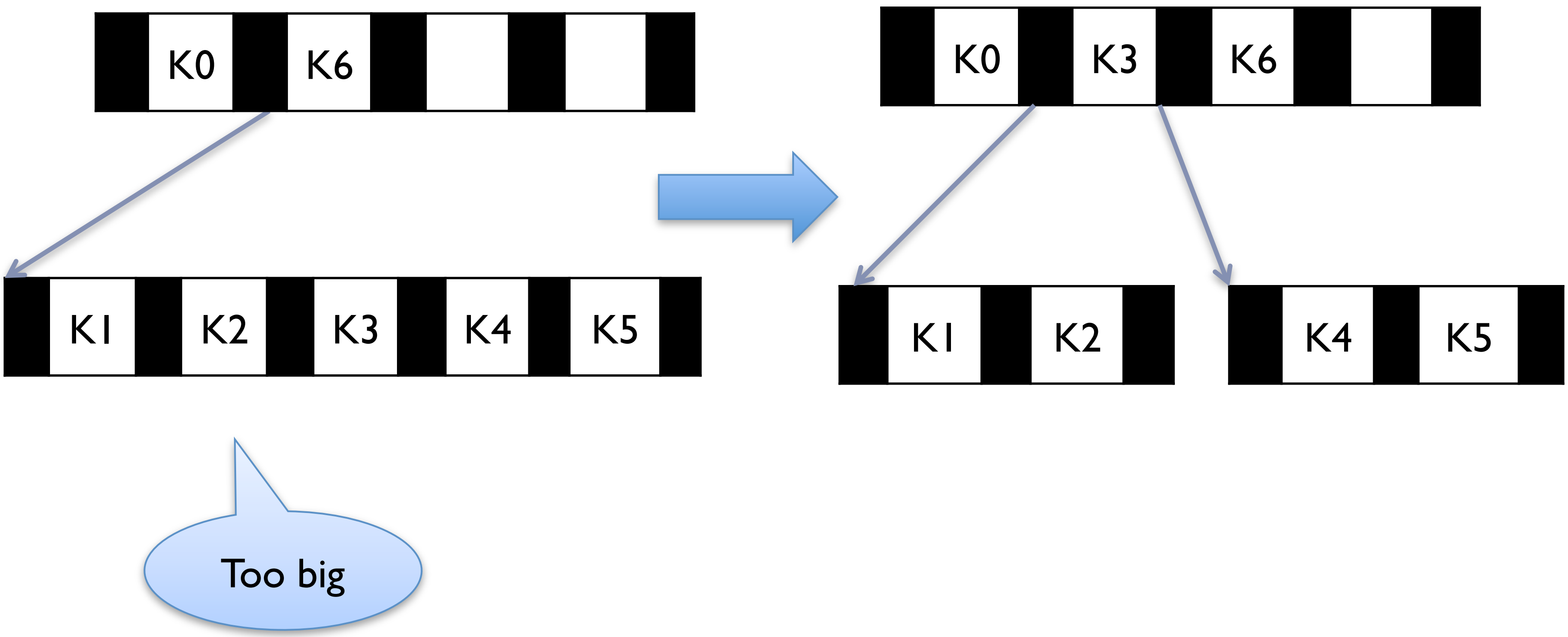

18
32
3
14
15
18
30
32
36
Insert(16)
18
32
3
14
15
18
30
32
36
16
15
18
32
3
14
15
16
18
30
32
36
18
15
32

$M = 3$ $L = 3$



Insert(12,40,45,38)
18
15
32
3
14
15
16
18
30
32
36
18
15
32
40
3
12
14
15
16
18
30
32
36
38
40
45

Insertion Algorithm: The Overflow Step



$M = 5$

Insertion Algorithm

1. Insert the key in its leaf in sorted order

2. If the leaf ends up with **L+1** items, **overflow**!
   - Split the leaf into two nodes with each half of the data.
   - Add the new leaf to the parent.
   - If the parent ends up with **M+1** children, **overflow**!

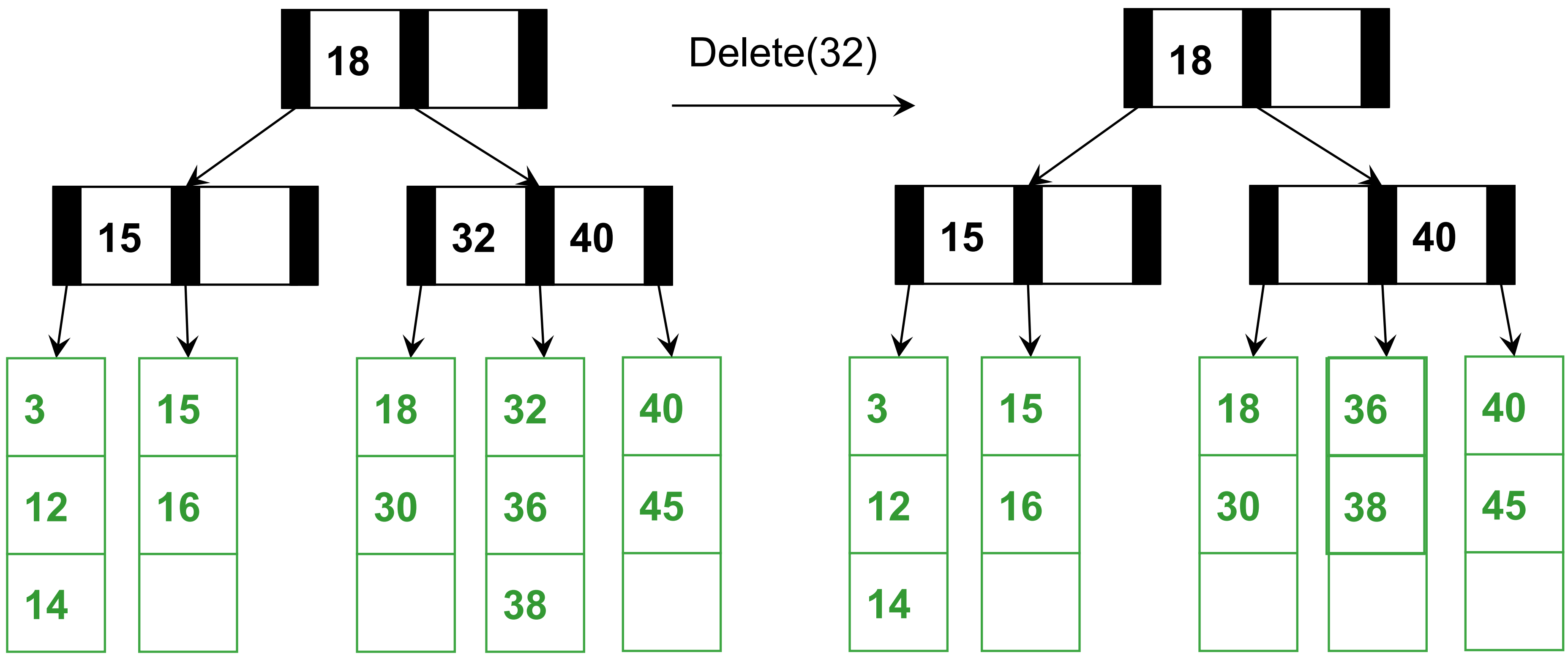
Insertion Algorithm

3. If an internal node ends up with ***M+1*** children, **overflow**!
   - Split the internal node into two nodes each with half the keys.
   - Add the new child to the parent
   - If the parent ends up with ***M*+1** items, **overflow**!

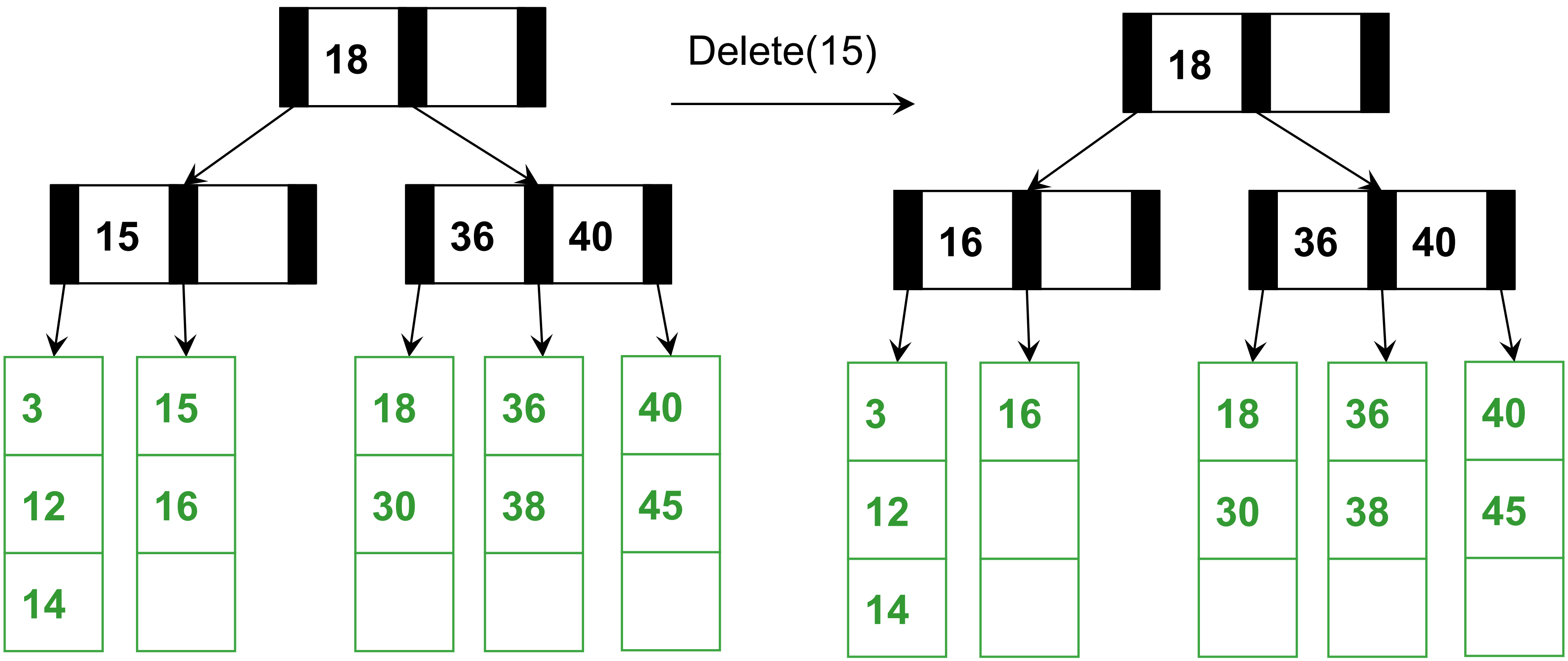
4. If the root ends up with ***M+1*** children, split it in two, and create new root with two children

This makes the tree deeper!

And Now for Deletion…



$M = 3$ $L = 3$

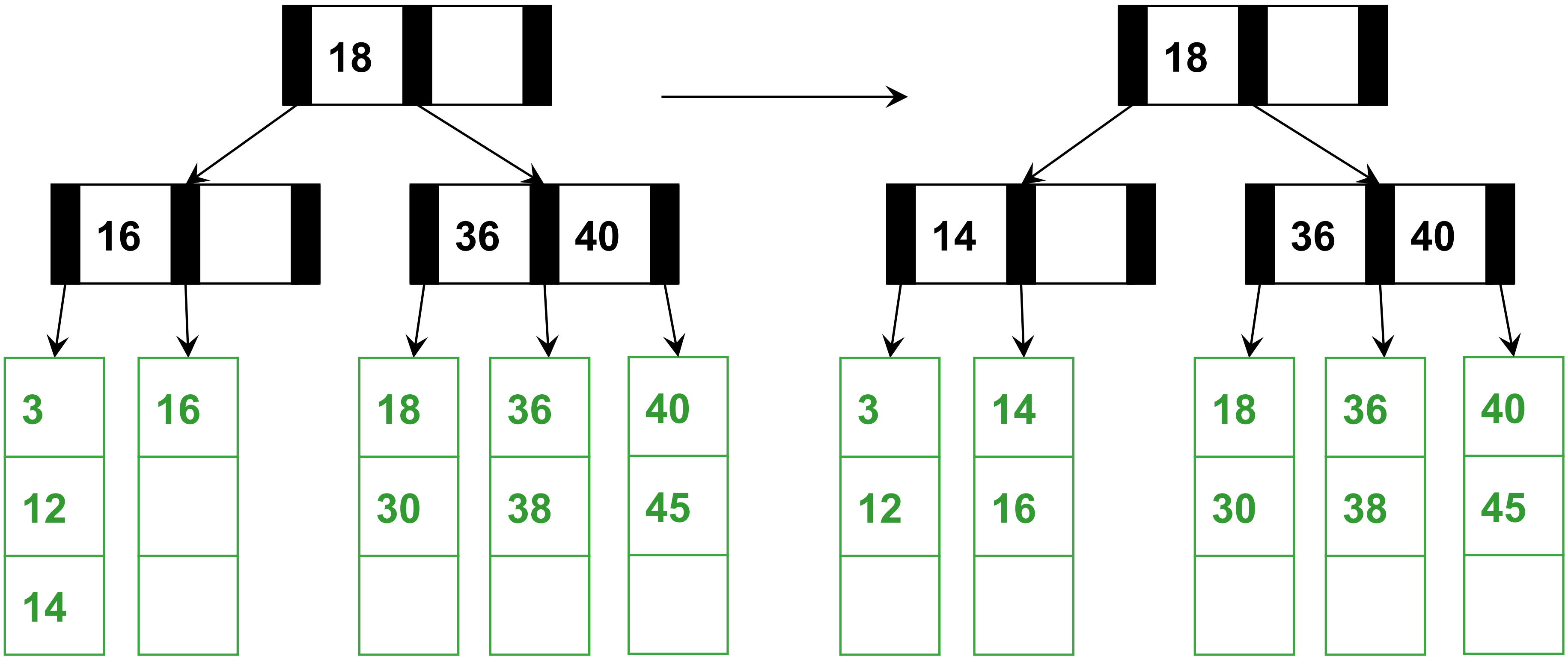


Are we okay?

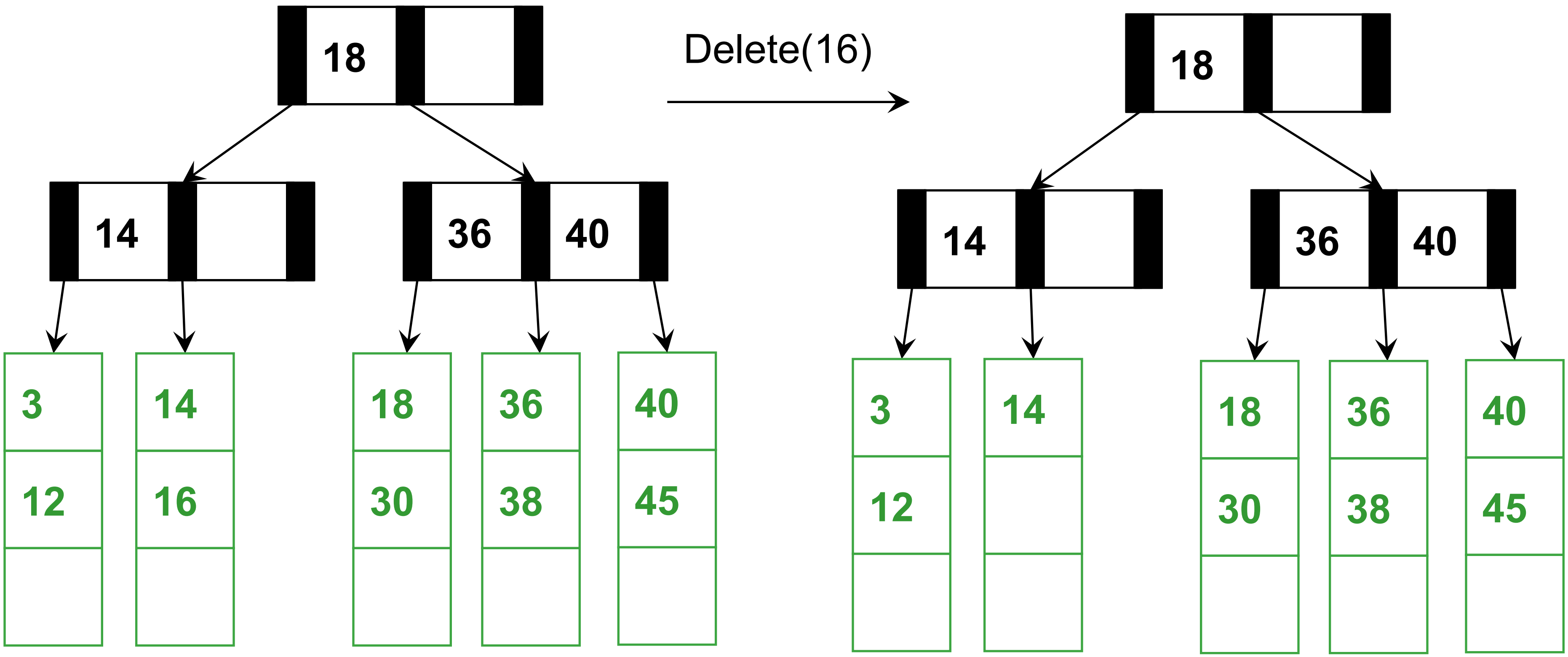
Are you using that 14?
Can I borrow it?

$M = 3$ $L = 3$

Leaf not half full!

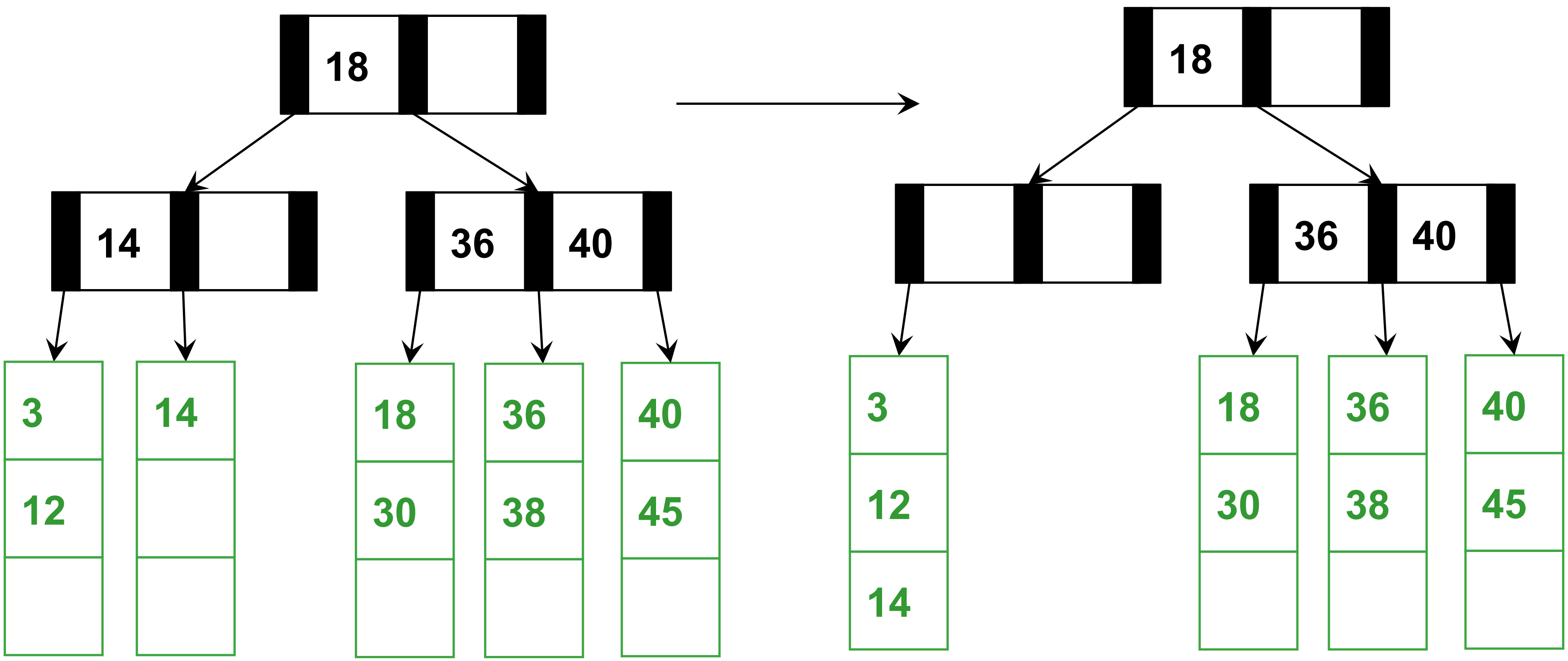

18
16
36
40
3
12
14
16
18
30
36
38
40
45
18
14
36
40
3
12
14
16
18
30
36
38
40
45

$M = 3$ $\quad$ $L = 3$



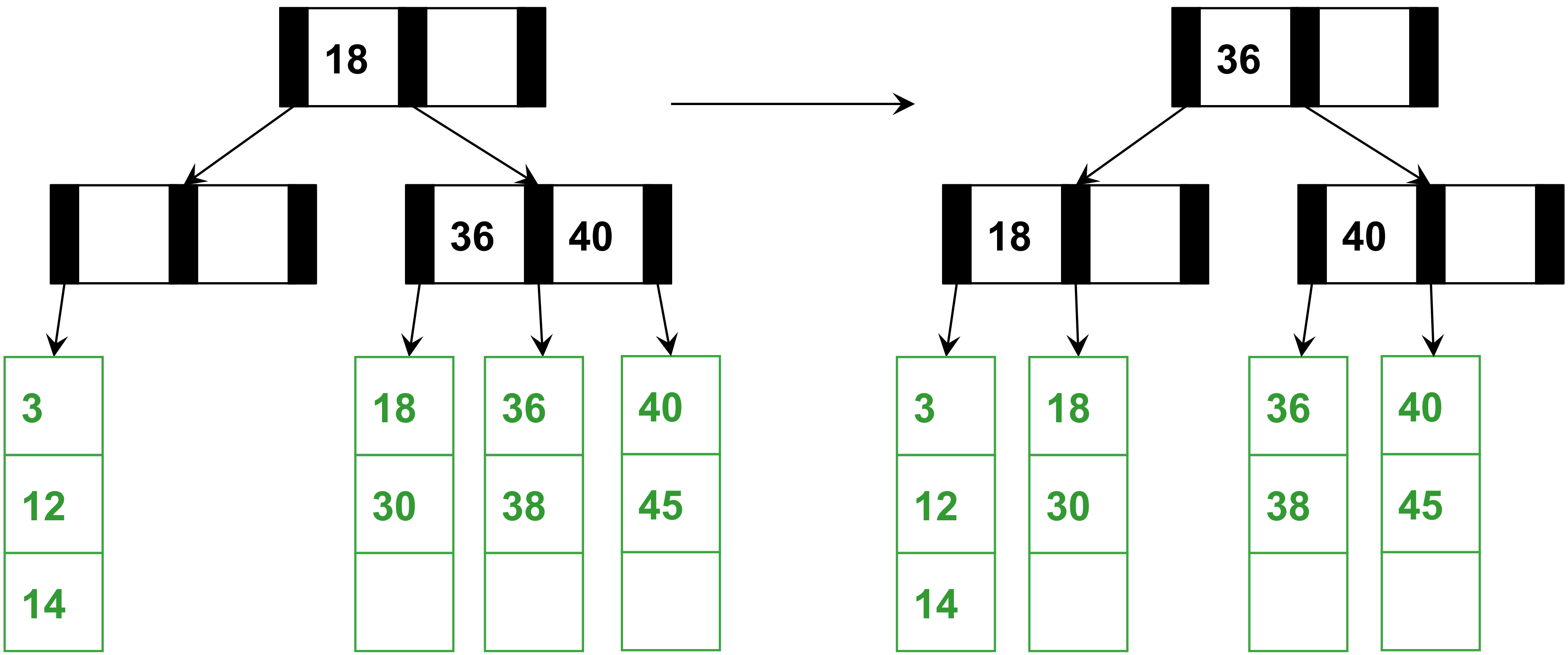
M = 3 *L* = 3

Are you using that 14?

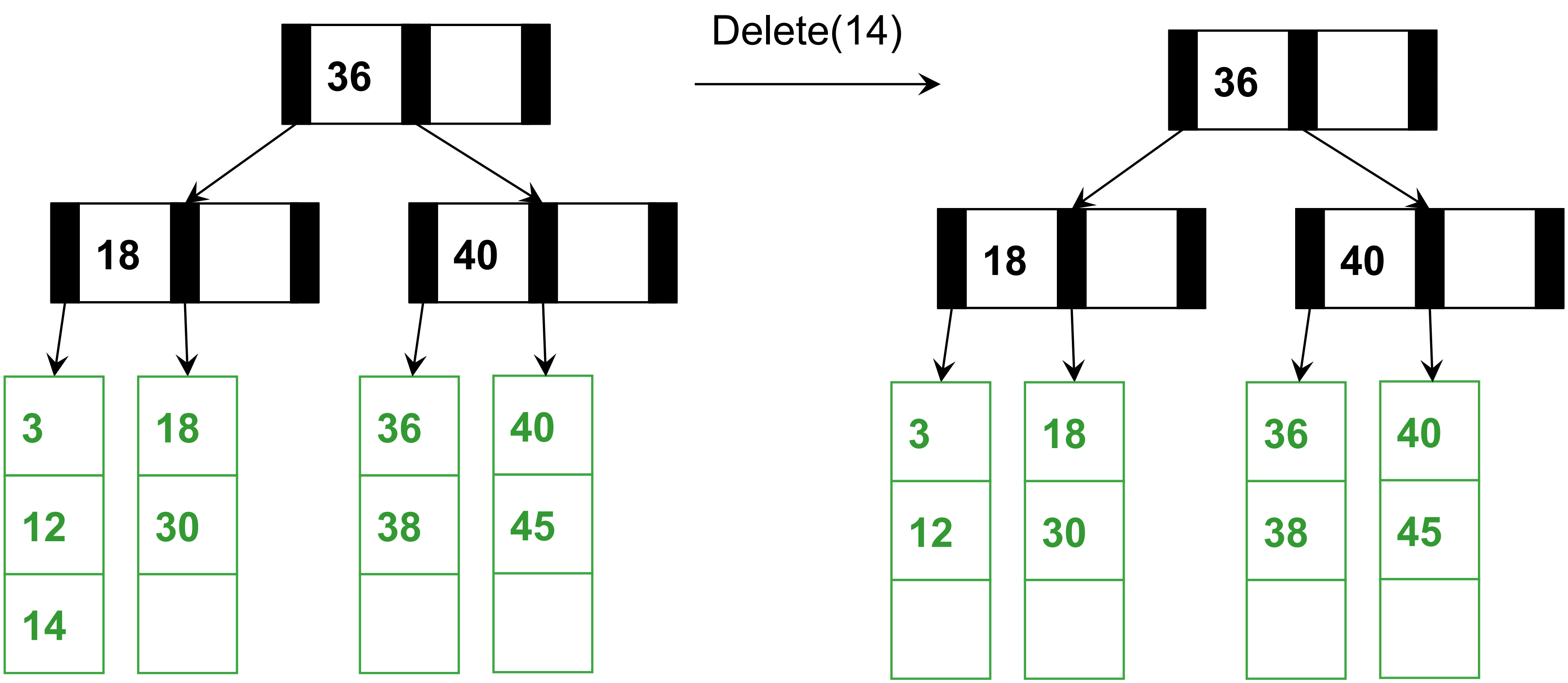


$M = 3$ $\quad L = 3$

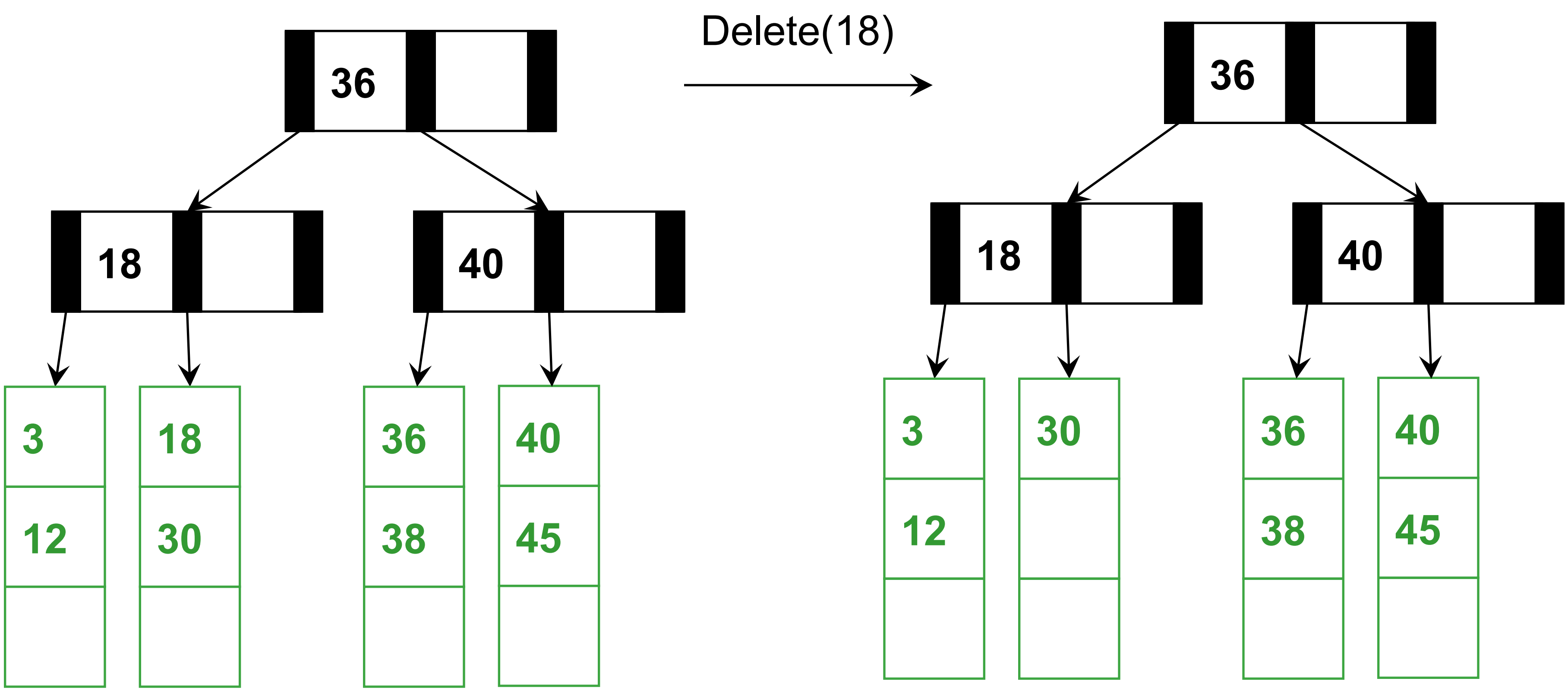
Are you using the node18/30?



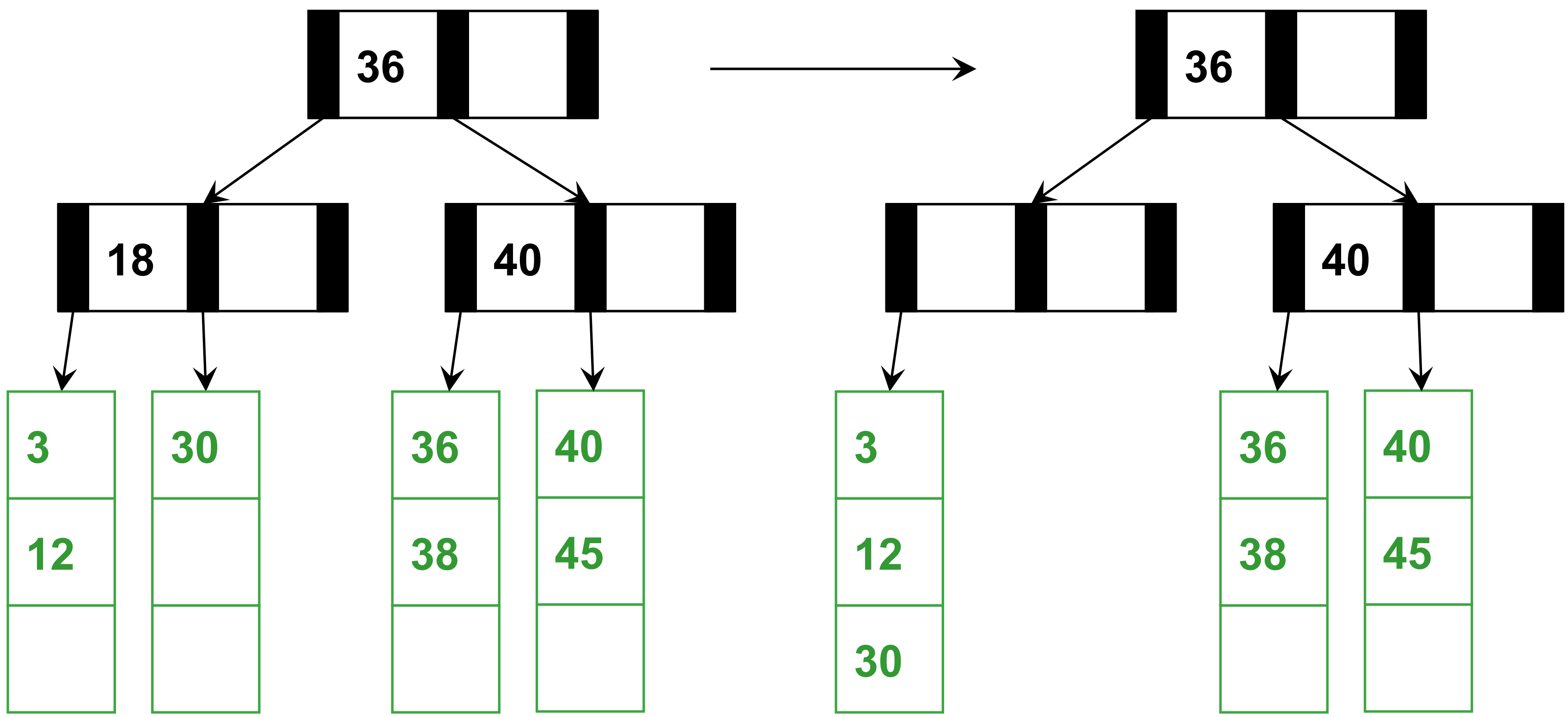
M = 3 *L* = 3


Delete(14)
36
18
40
3
12
14
18
30
36
38
40
45
36
18
40
3
12
18
30
36
38
40
45

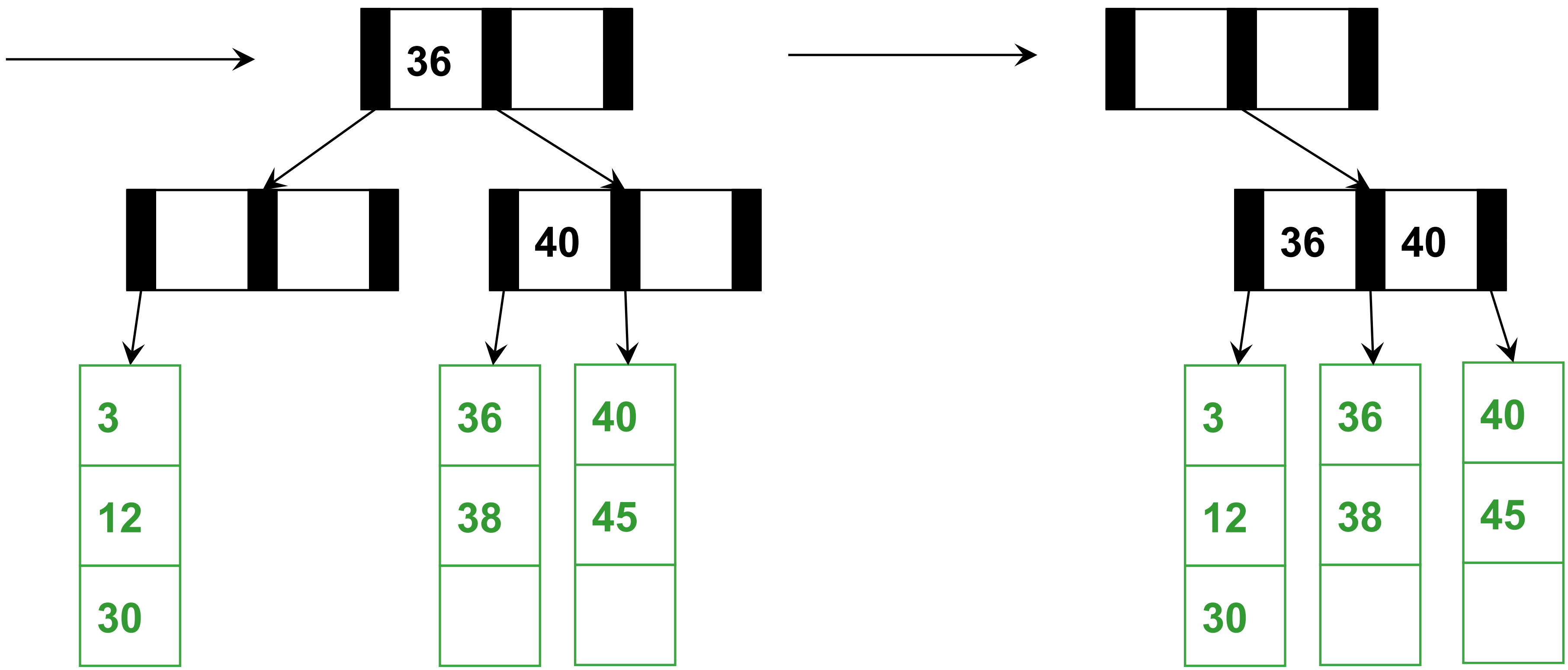
M = 3 *L* = 3



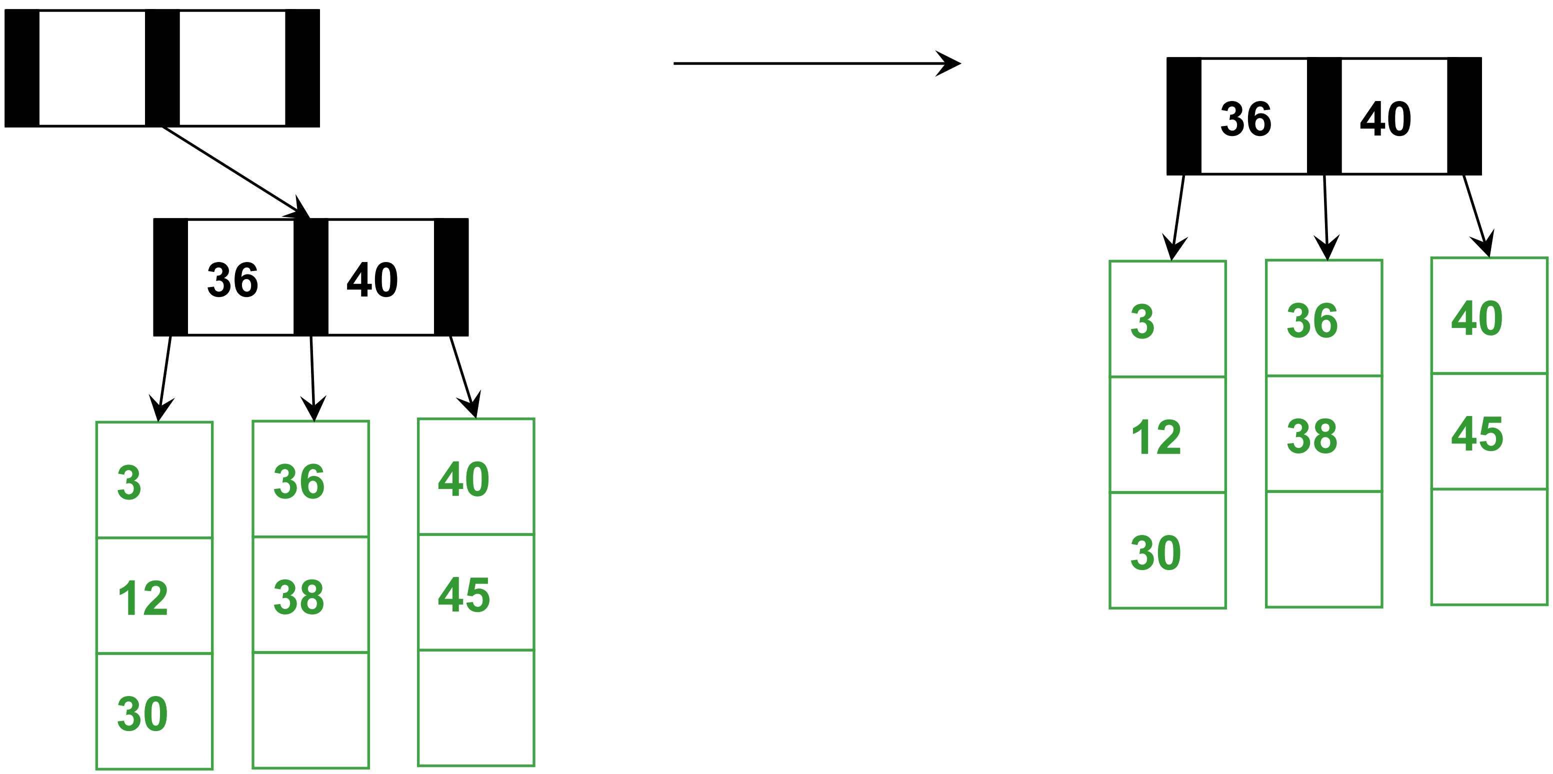
$M = 3$ $L = 3$



$M = 3$ $L = 3$

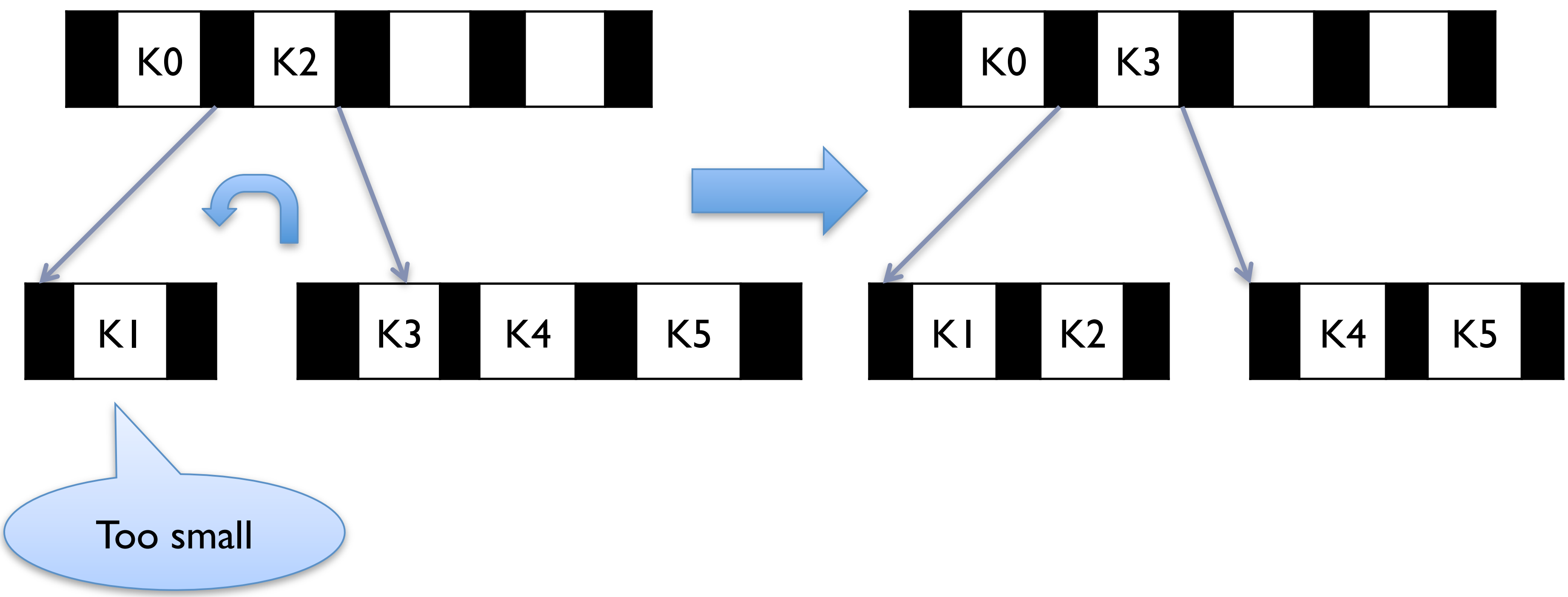


$M = 3$ $L = 3$



$M = 3$ $L = 3$

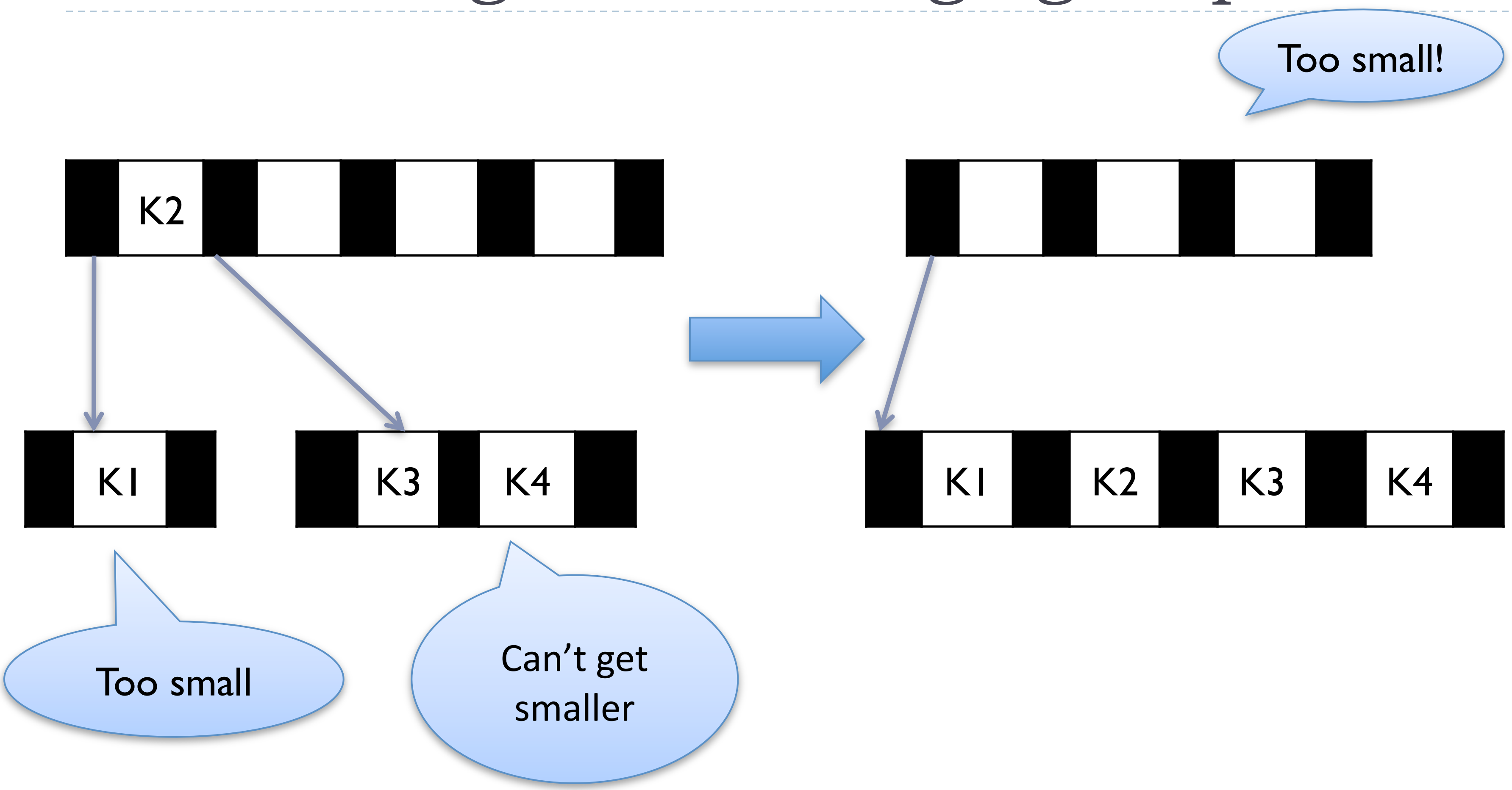
Deletion Algorithm: Rotation Step



M = 5

This is *left* rotation. Similarly, *right* rotation

Deletion Algorithm: Merging Step



$M = 5$

Deletion Algorithm

1. Remove the key from its leaf

2. If the leaf ends up with fewer than $\lceil \boldsymbol{L/2} \rceil$ items, **underflow**!
   - Try a left rotation
   - If not, try a right rotation
   - If not, merge, then check the parent node for underflow

Deletion Slide Two

3. If an internal node ends up with fewer than $\lceil M/2 \rceil$ children, **underflow!**
    - Try a left rotation
    - If not, try a right rotation
    - If not, merge, then check the parent node for underflow

4. If the root ends up with only one child, make the child the new root of the tree

This reduces the height of the tree!