CPTS 223 Advanced Data Structure C/C++

Tree: B-tree and B+ Tree

Overview

- Tree data structure
- Binary search trees
    - Support O(lg(n)) operations
    - Balanced trees
- STL set and map classes
- B-trees for accessing secondary storage
- Applications of Tree

Overview

- Tree data structure
- Binary search trees
  - Support O(lg(n)) operations
  - Balanced trees
- STL set and map classes
- B-trees for accessing secondary storage
- Applications of Tree

Accessing a hard disk location takes about 5ms = 5,000,000ns

Accessing a solid state drive location takes about 25us = 25,000ns

Data from https://www.pcmag.com/encyclopedia/term/access-time#:~:text=(1)%20Memory%20access%20time%20is,character%20after%20initiating%20a%20request.

Overview

- Tree data structure
- Binary search trees
  - Support O(lg(n)) operations
  - Balanced trees
- STL set and map classes
- B-trees for accessing secondary storage
- Applications of Tree

Primary storage: accessing a SDRAM location takes about 10ns

Accessing a hard disk location takes about 5ms = 5,000,000ns

Accessing a solid state drive location takes about 25us = 25,000ns

Data from https://www.pcmag.com/encyclopedia/term/access-time#:~:text=(1)%20Memory%20access%20time%20is,character%20after%20initiating%20a%20request.

Large Databases

Organization	Database Size
WDCC	6000TB
NERSC	2800TB
AT&T	323TB
Google	33 trillion rows (91 million insertions per day)
Sprint	3 trillion rows (100 million insertions per day)
ChoicePoint	250TB
Yahoo!	100TB
YouTube	45TB
Amazon	42TB
Library of Congress	20TB

Count the bytes

- Kilo: x 10^3
- Mega: x 10^6
- Giga: x 10^9
- Tera: x 10^{12}
- Peta: x 10^{15}
- Exa: x 10^{18}
- Zetta: x 10^{21}

WSU

Count the bytes

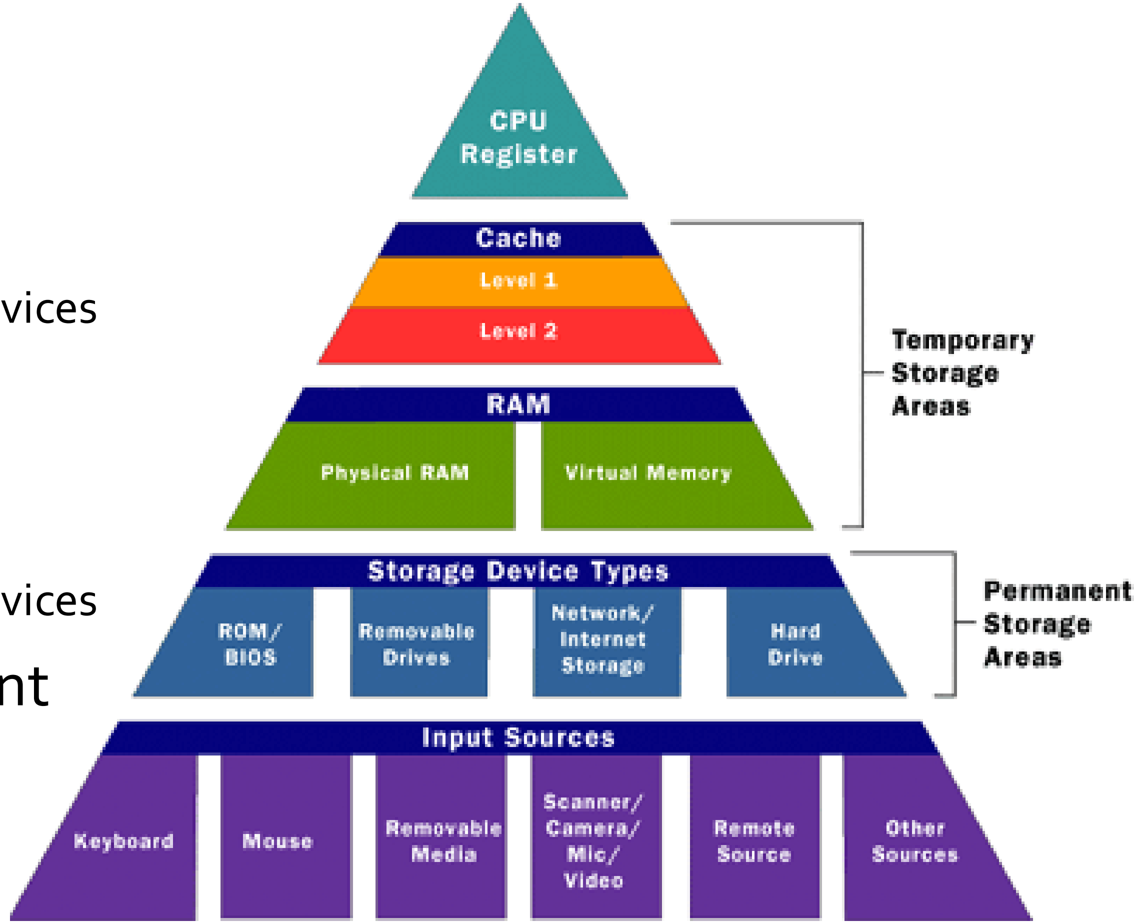
- Kilo: x 10^3
- Mega: x 10^6
- Giga: x 10^9
- Tera: x 10^{12}
- Peta: x 10^{15}
- Exa: x 10^{18}
- Zetta: x 10^{21}

Current limit for single node storage

Needs more complicated disk/IO machine

Storage in computer architecture

- CPU: temporary
  - Register (0, 1)
  - Cache (L1, L2): B – KB, probably L3, MB
  - Good for random access, pure electronic devices
- RAM (MB – 100 GB): temporary
  - Physical RAM
  - Virtual memory (disk)
  - Good for random access, pure electronic devices
- Storage devices (100 GB - TB): permanent
  - ROM/BIOS
  - Removable Drives: USB
  - Hard drive
    - Good for sequential access, with a motor and head arm
  - SSD



WSU

Count the bytes

	Primary storage	Secondary storage
Hardware	RAM, cache	Disk (i.e., IO)
Storage capacity	100MB to ~100GB	GB (10^9) to TB (10^{12})
Data persistence	Temporary (erased after process terminates)	Persistent (permanently stored)
Data access speed	a few clock cycles (ie., x 10^{-9} seconds, ns)	milliseconds (x 10^{-3} seconds, ms) Data seek + read

Use a balanced BST?

- Google: 33 trillion items
- Indexed by ?
  - IP, HTML page content
- Estimated access time (if we use a simple balanced BST):
  - h = O(log2(33x10^12)) → 44.9 disk accesses
  - Assume disk access speed: 120 disk accesses per second
  - → Each search takes 0.37 seconds
  - 0.37*1^6 / 3600=102.7778 hours

If we do one million (1^6) searches?

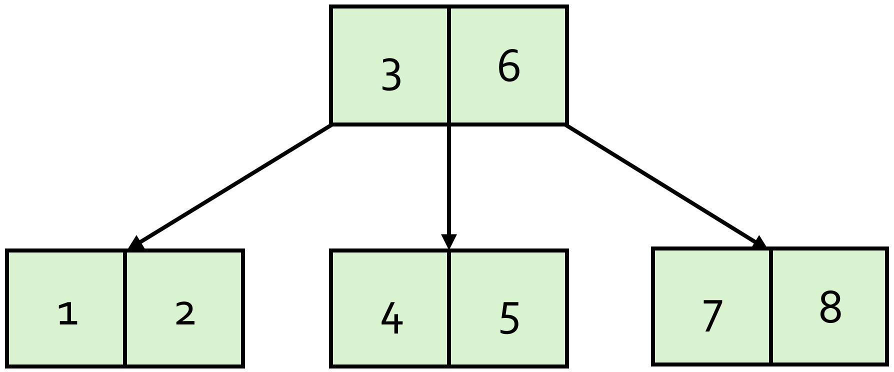
WSU

Height reduction

- Balanced BST trees at best have heights O(lg (n))
  - N=10^6 → log2(10^6) is roughly 20
  - 20 disk seeks for each worse-case search would be too much
- → Reduce the height
- How?
  - Increase the log base beyond 2
  - e.g., log5(10^6) is < 9 ← halve the height
  - Instead of binary (2-ary) trees, use m-ary search trees s.t. m>2

WSU

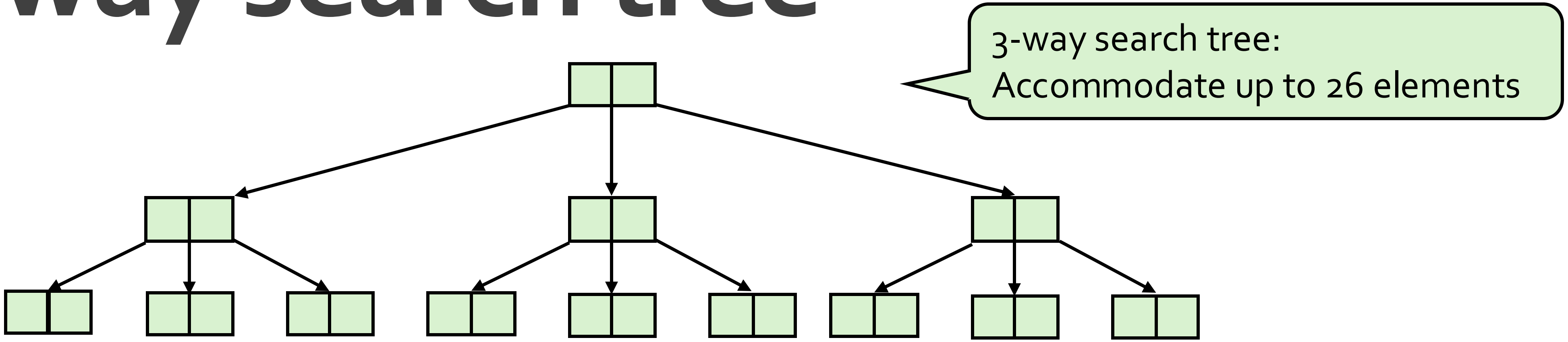
M-way search tree

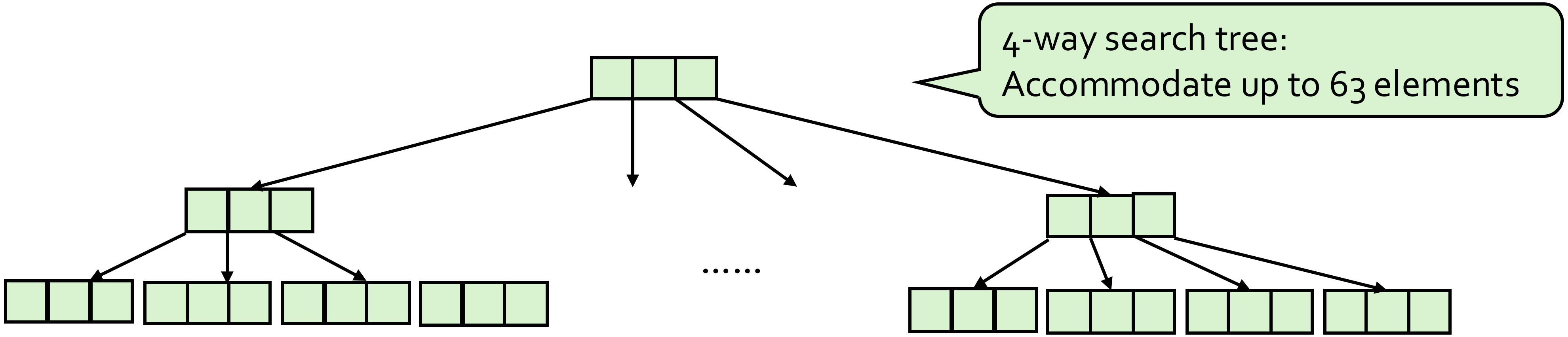


- Example: 3-way search tree
- Each node stores:
  - ≤ 2 keys
  - ≤ 3 children
- Height of a balanced 3-way search tree? (a function of N → h(N))

WSU

M-way search tree





Height reduction

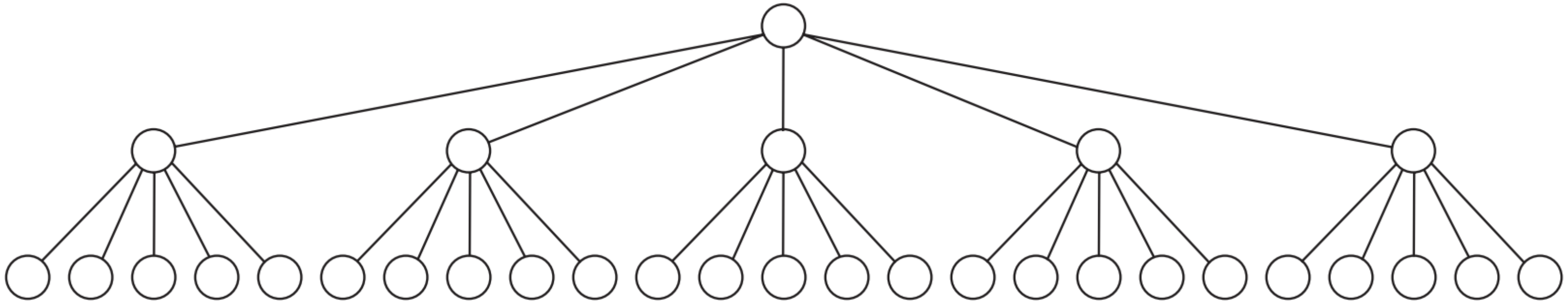
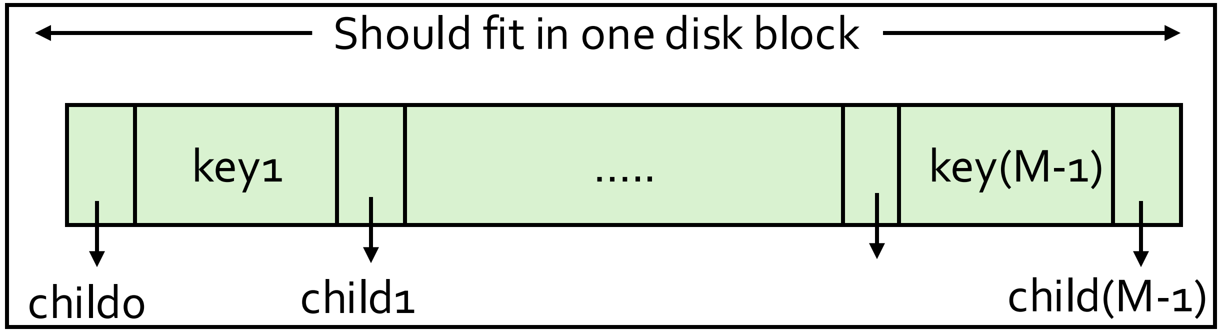


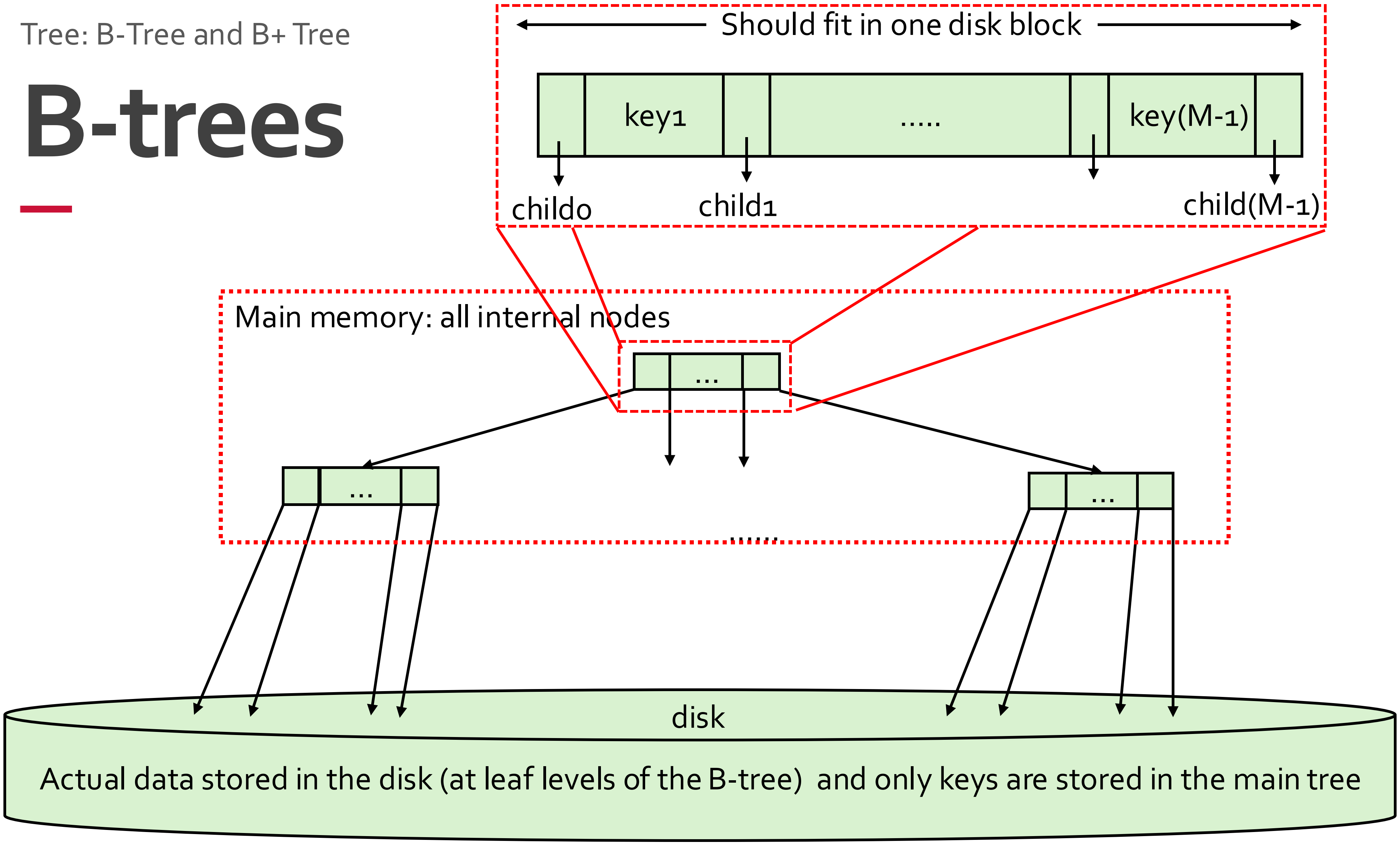
Figure 4.62 5-ary tree of 31 nodes has only three levels

M-way search tree

- Each node access brings in (M-1) keys and M child pointers
- Choose M so node size = 1 disk block size
- Height of tree = Θ(logM(N))



B-trees


Should fit in one disk block
key1
.....
key(M-1)
child0
child1
child(M-1)
Main memory: all internal nodes
...
...
...
....
disk
Actual data stored in the disk (at leaf levels of the B-tree) and only keys are stored in the main tree

B-trees: example

- A node should fit in one disk block
- Standard disk block size = 8192 bytes
- Assume keys use 32 bytes, pointers use 4 bytes
  - Keys uniquely identify data elements
- Space per node = 32*(M-1) + 4*M = 8192 bytes
- M = 228
- log228(33x1012) = 5.7 (disk accesses)
- Each search takes 0.047 seconds

Factor 1: capacity of a single block

Factor 2: size of keys and pointers

BST: 0.37 seconds

B-tree (B+ tree) definition

Parameters: M, L

- Leaves store the real data items
- Internal nodes store up to M-1 keys
  - s.t., key i is the smallest key in subtree i+1
- Root can have between 2 to M children
- Each internal node (except root) has between ceil(M/2) to M children
- All leaves are at the same depth
- Each leaf has between ceil(L/2) and L data items, for some L

B-tree of order 5

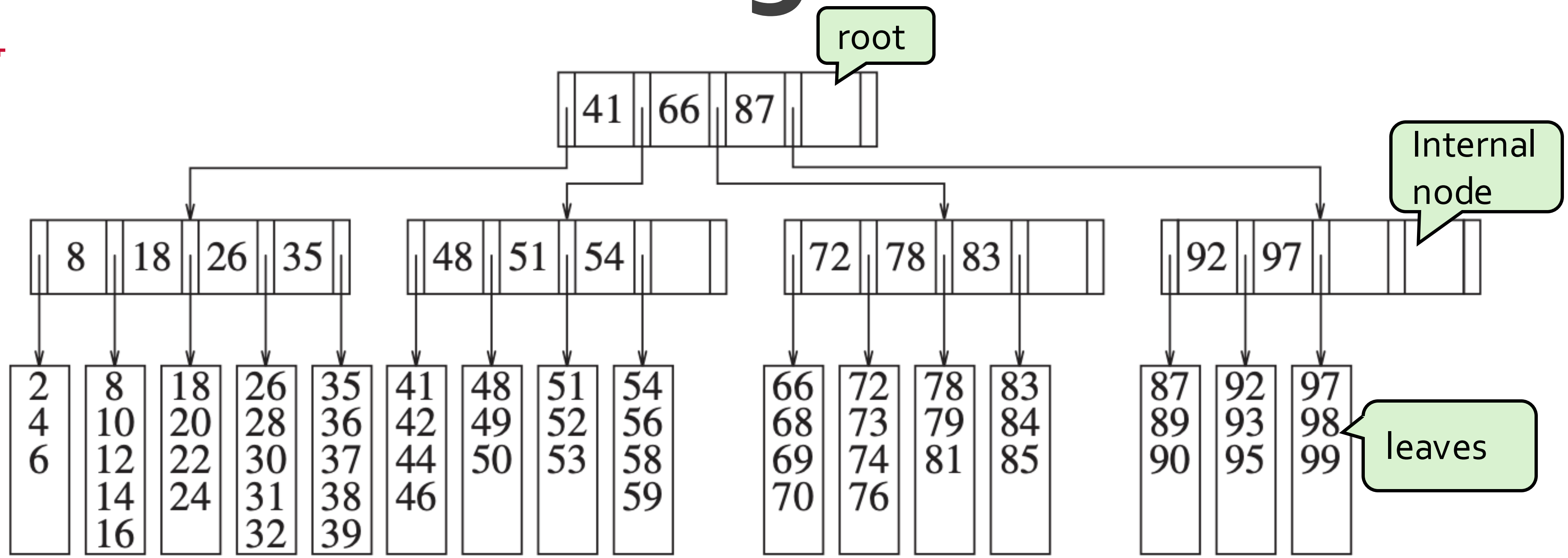


Figure 4.63 B-tree of order 5

M=5 (order of the B+ tree)
L=5 (#data items bound for leaves)

- Each int. node (except root) has to have at least 3 children
- Each leaf has to have at least 3 data items

B-tree of order 5

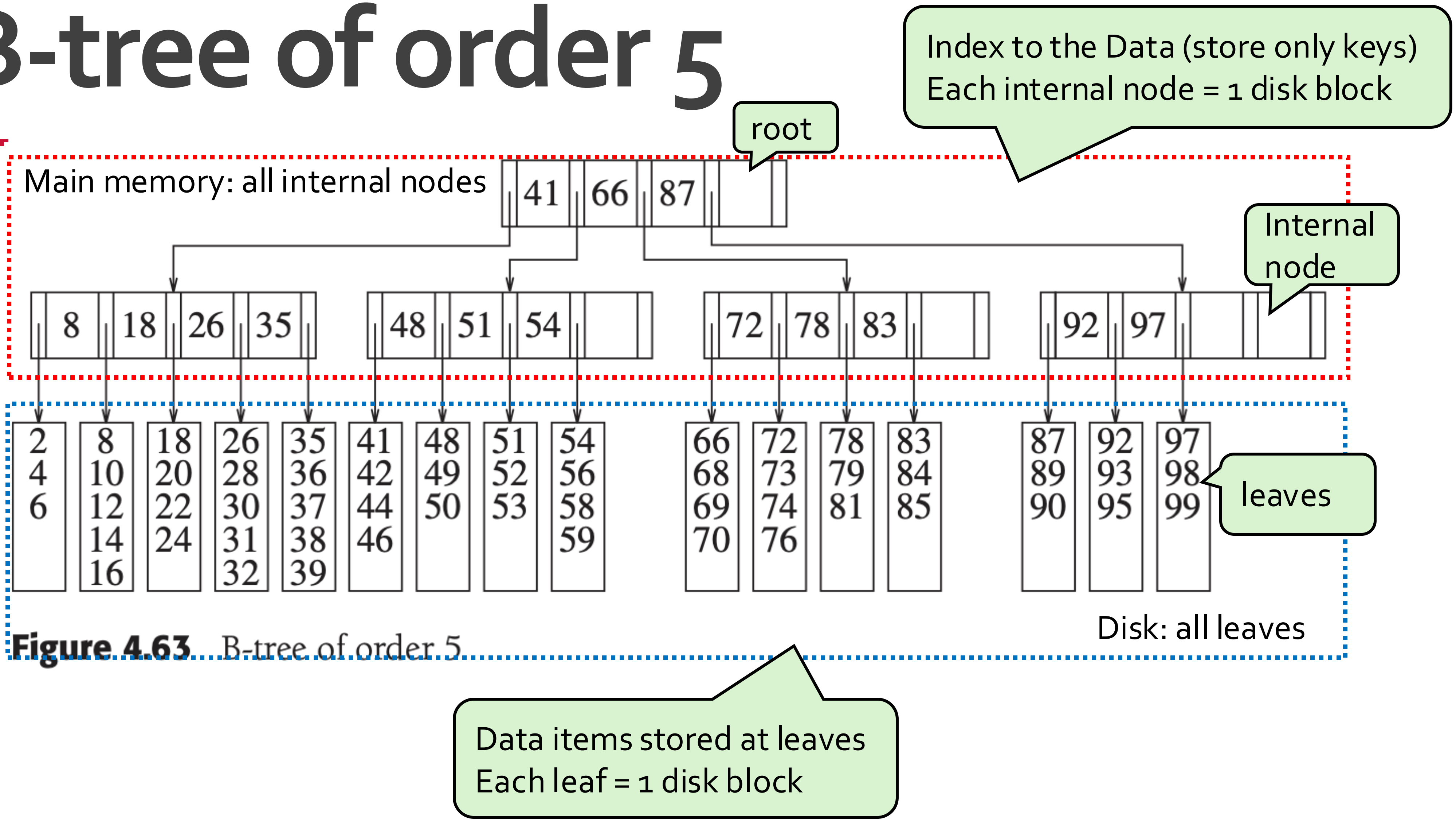


Figure 4.63 B-tree of order 5

B-tree of order 5: insert(57)

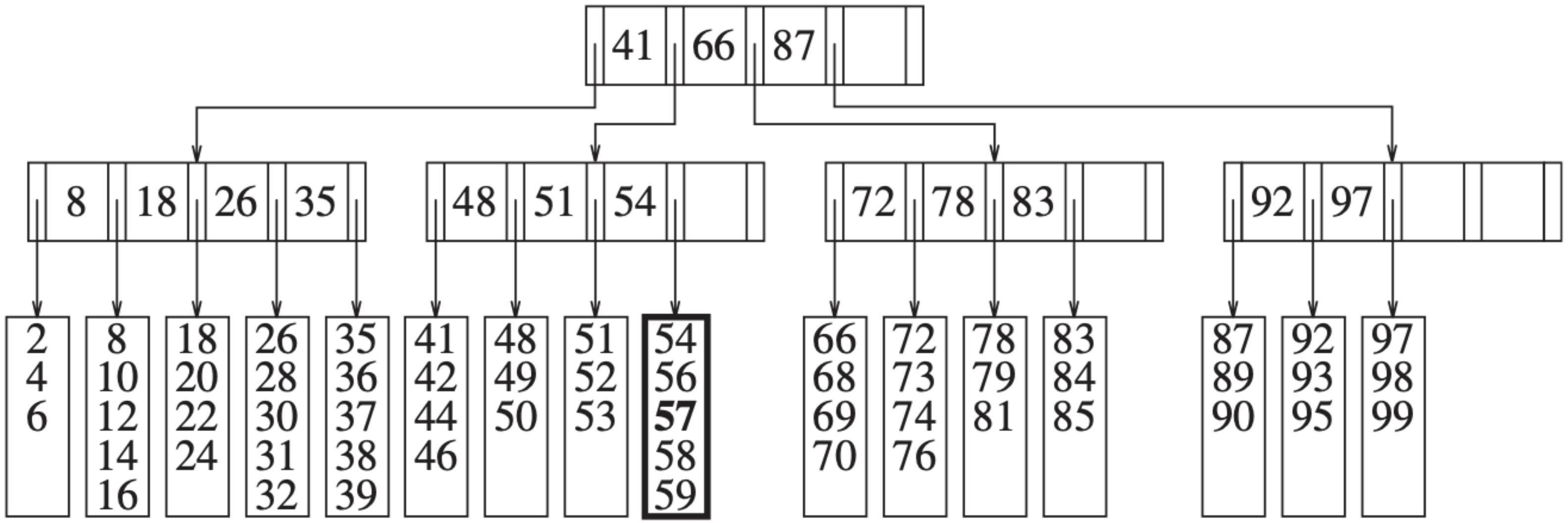


Figure 4.64 B-tree after insertion of 57 into the tree in Figure 4.63

- Searching by keys
- Insert if satisfying B-tree condition

B-tree of order 5: insert(55)

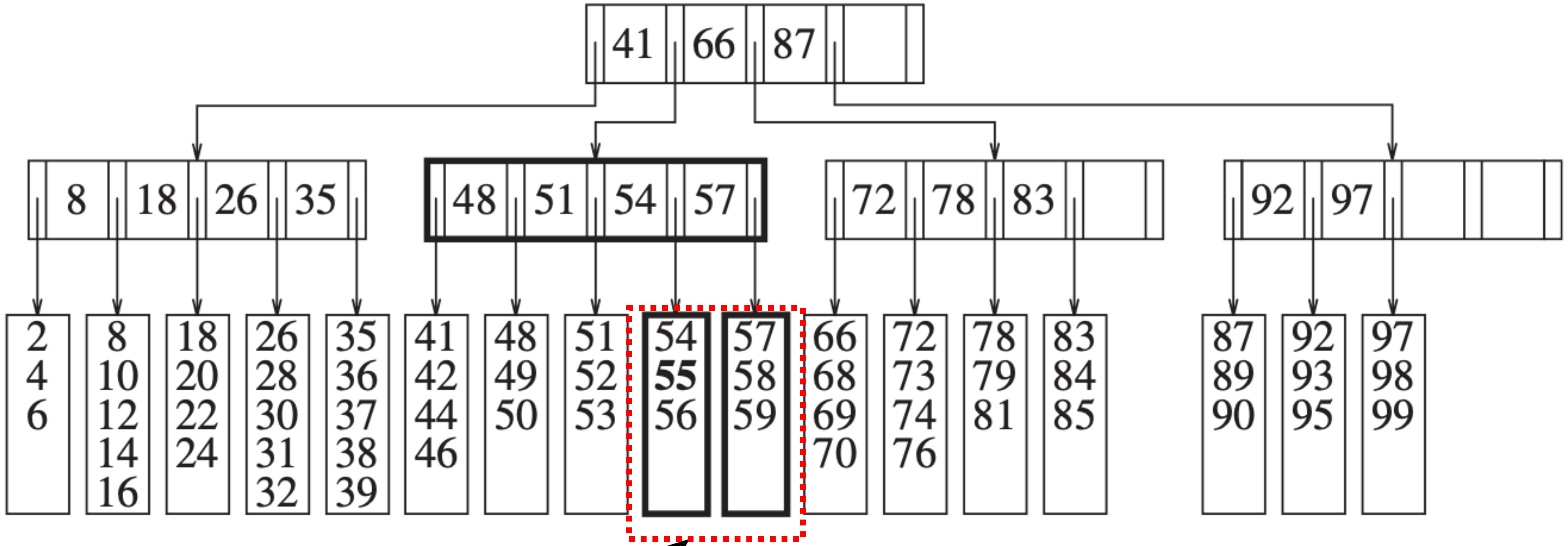


Figure 4.65 Insertion of 55 in B-tree in Figure 4.64 causes a split into two leaves

- Searching by keys
- Insert if satisfying B-tree condition
- If too many: split the leaf into two leaves

- Each leaf has between ceil(L/2) and L data items, for some L
- Implementation: leaf node and are sorted sequentially in the linked list.

B-tree of order 5: insert(40)

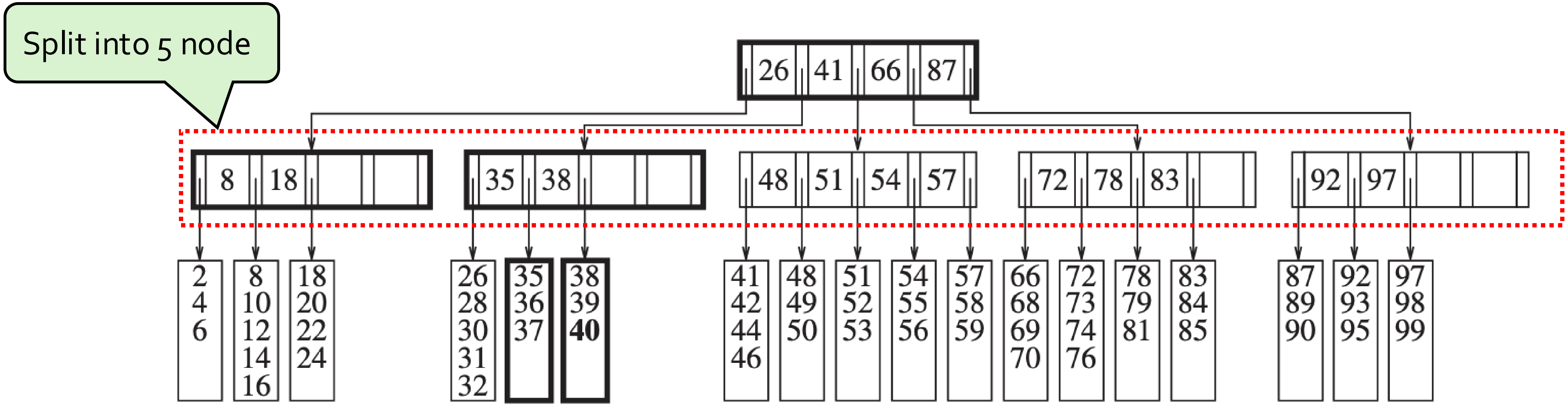


Figure 4.66 Insertion of 40 into the B-tree in Figure 4.65 causes a split into two leaves and then a split of the parent node

WSU

B-tree of order 5: insert(40)

The first node splits into 2 node

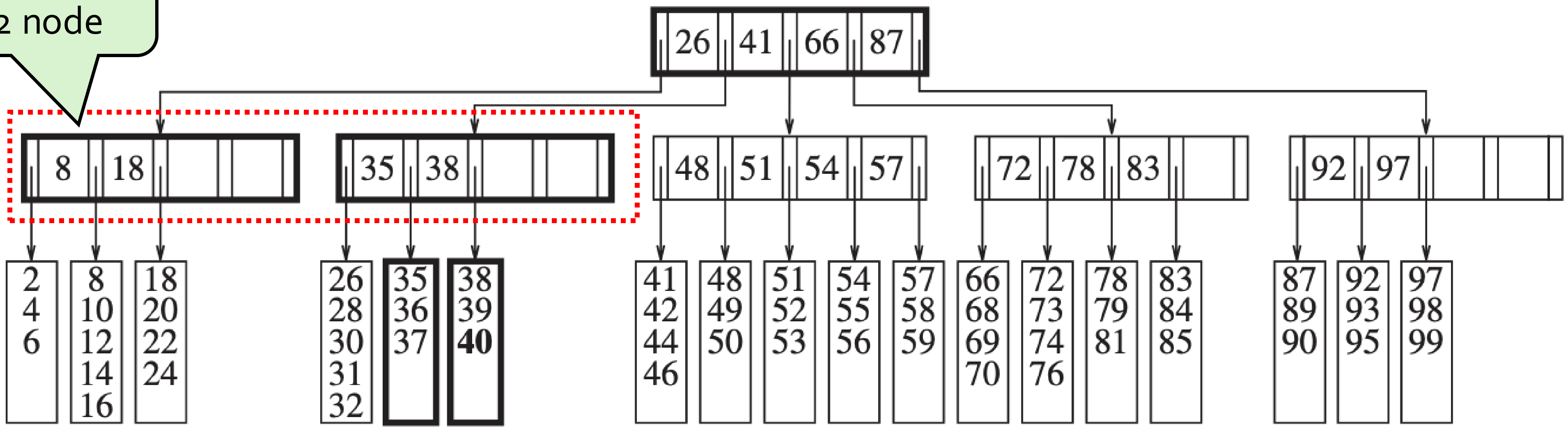


Figure 4.66 Insertion of 40 into the B-tree in Figure 4.65 causes a split into two leaves and then a split of the parent node

WSU

B-tree of order 5: insert(40)

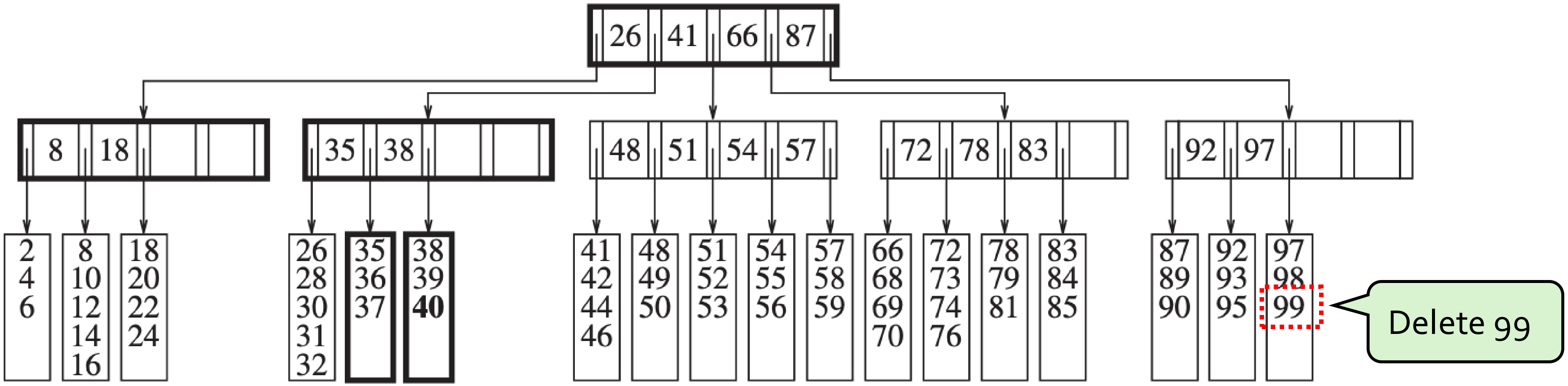


Figure 4.66 Insertion of 40 into the B-tree in Figure 4.65 causes a split into two leaves and then a split of the parent node

WSU

B-tree of order 5: insert(40)

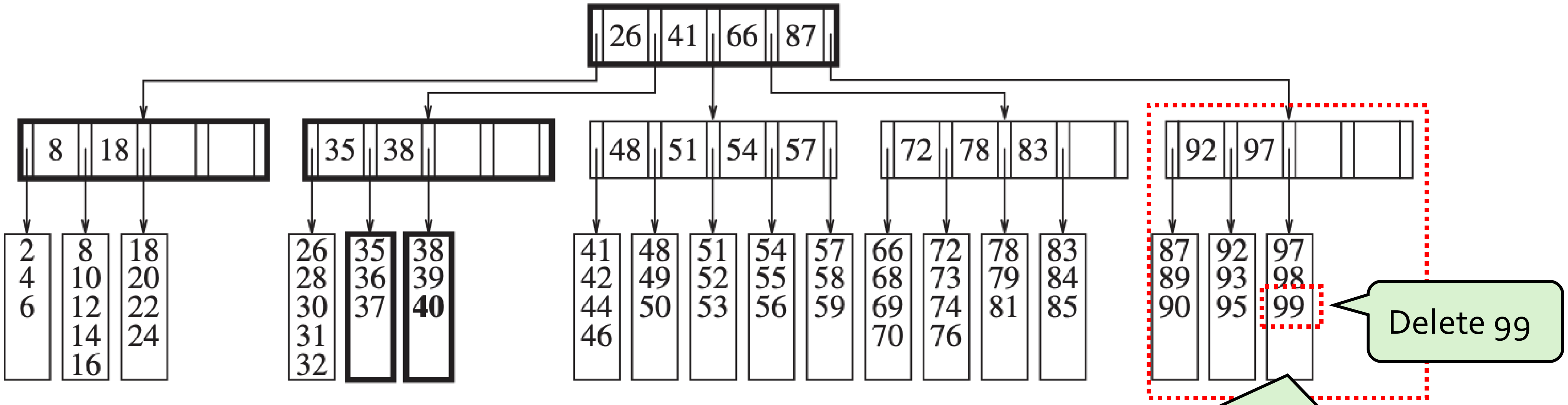


Figure 4.66 Insertion of 40 into the B-tree in Fig
and then a split of the parent node

- Each leaf has between ceil(L/2) and L data items, for some L

B-tree of order 5: insert(40)

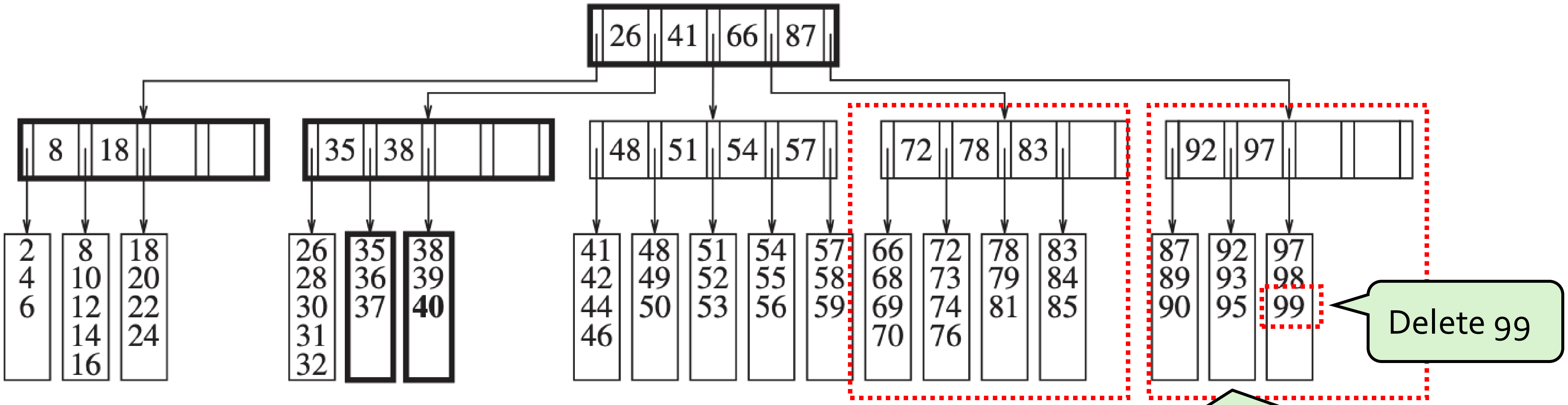


Figure 4.66 Insertion of 40 into the B-tree in Fig... and then a split of the parent node

- Each leaf has between ceil(L/2) and L data items, for some L
- Each internal node (except root) has between ceil(M/2) to M children

B-tree of order 5: delete(99)

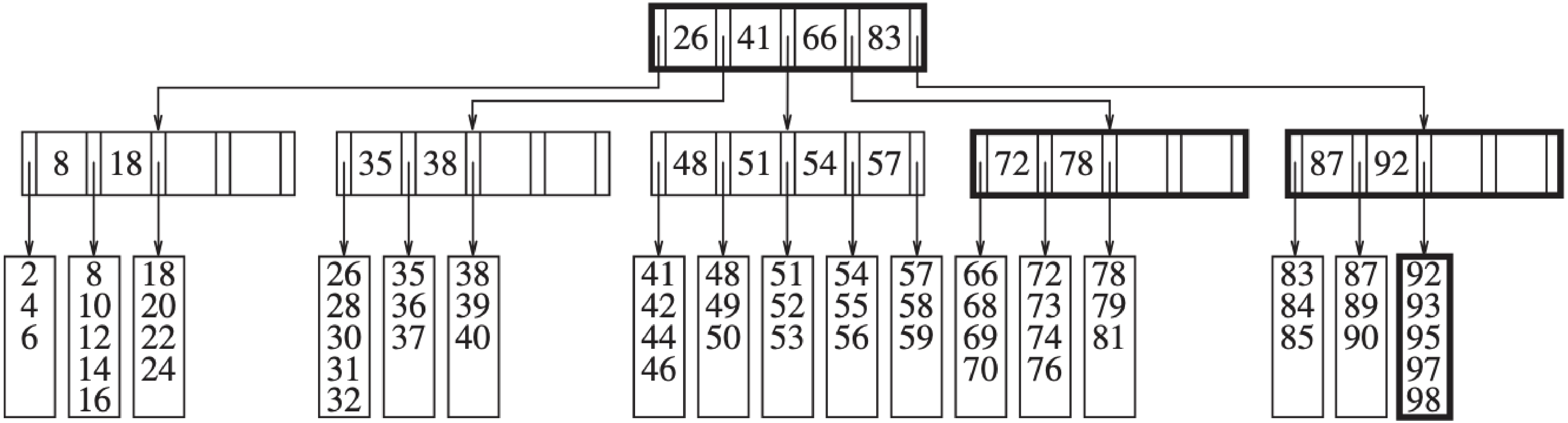


Figure 4.67 B-tree after the deletion of 99 from the B-tree in Figure 4.66