

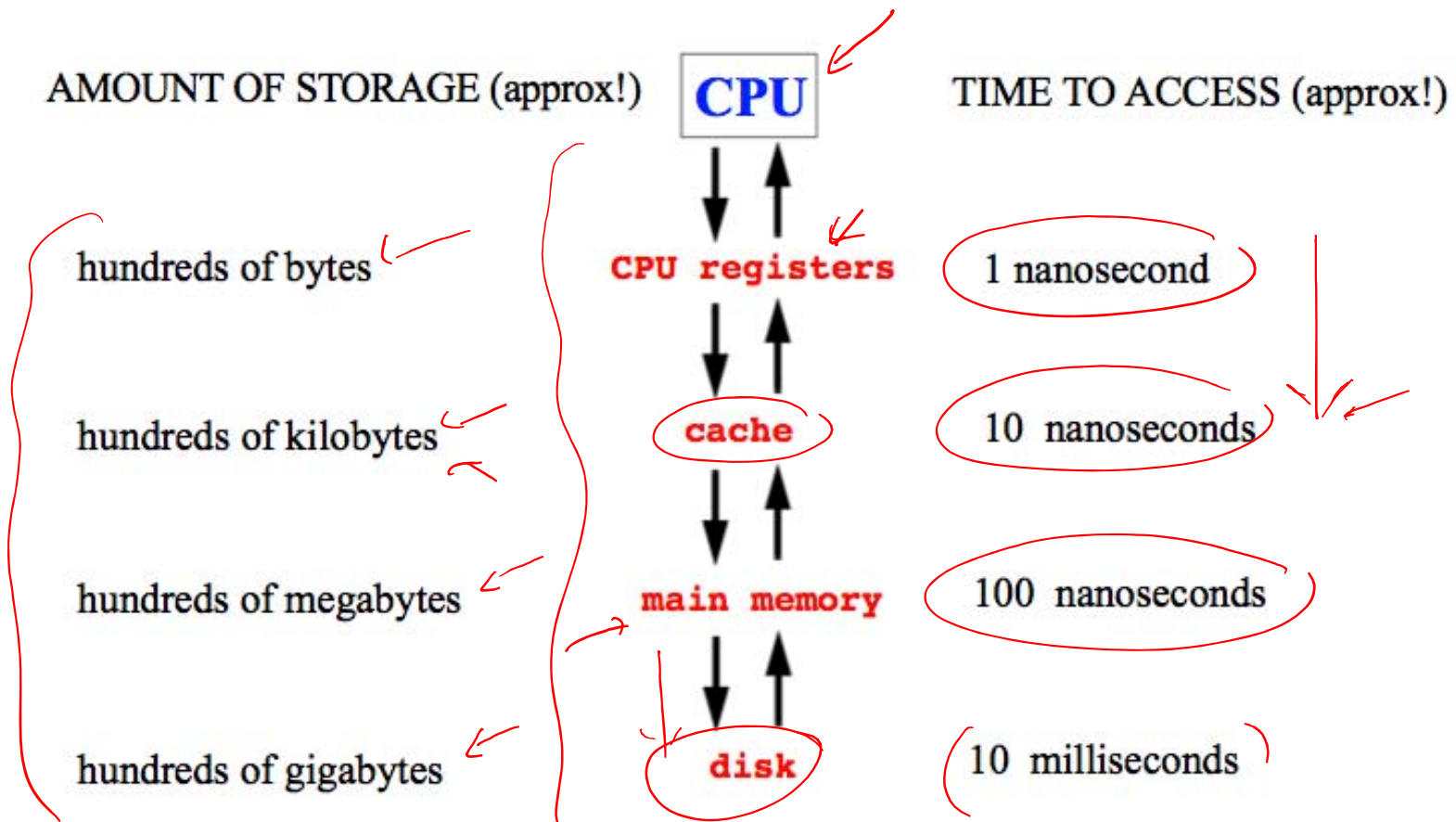


CSE 100: B TREES

B- Trees

- Data structures for efficient search on secondary storage

Typical memory hierarchy: a picture



1sec = 1,000 millisec = 1,000,000 microsec =
1,000,000,000 nanosec

Consequences of the memory hierarchy

- Accessing a variable can be fast or slow, depending on various factors
- If a variable is in slow memory, accessing it will be slow
- However, when it is accessed, the operating system will typically move that variable to faster memory (“cache” or “buffer” it), along with some nearby variables
 - The idea is: if a variable is accessed once in a program, it (and nearby variables) is likely to be accessed again
- So it is possible for one access of a variable to be slow, and the next access to be faster; possibly orders of magnitude faster

```
x = z[i]; // if z[i] is on disk this takes a long time
z[i] = 3; // now z[i] is in cache, so this is very fast!
z[i+1] = 9; // nearby variables also moved, so this is fast
```

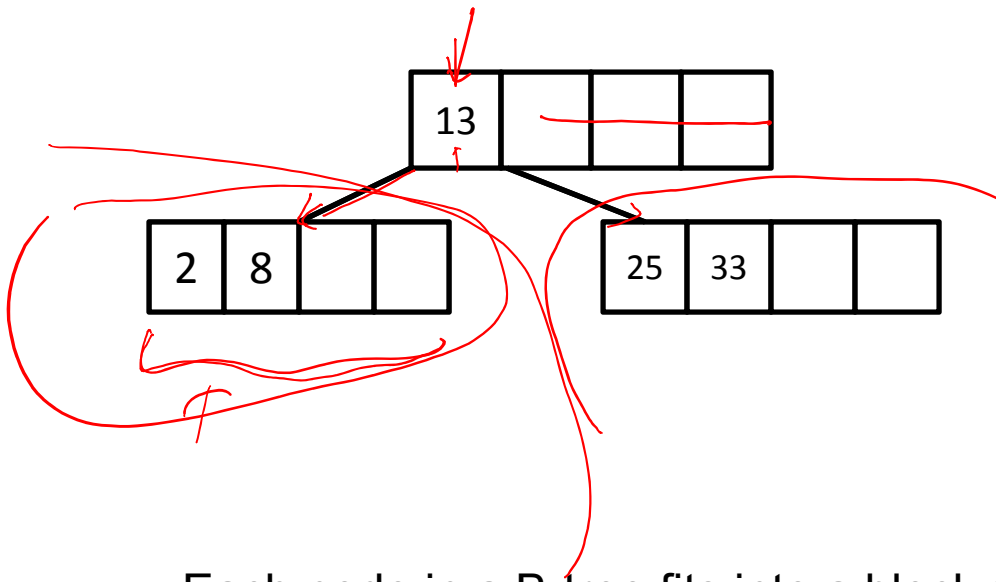
- The biggest speed difference is between disk access and semiconductor memory access, so that's what we will pay most attention to

Accesssing data on disk

- Because disk accesses are many (thousands!) of times slower than semiconductor memory accesses, if a datastructure is going to reside on disk, it is important that it can be used with very few disk accesses
- The most commonly used data structure for large disk databases is a B-tree, which can be designed to use disk accesses very efficiently
- Operations that we are interested in:
 - Insert ✓
 - Delete ✓
 - Find ✓

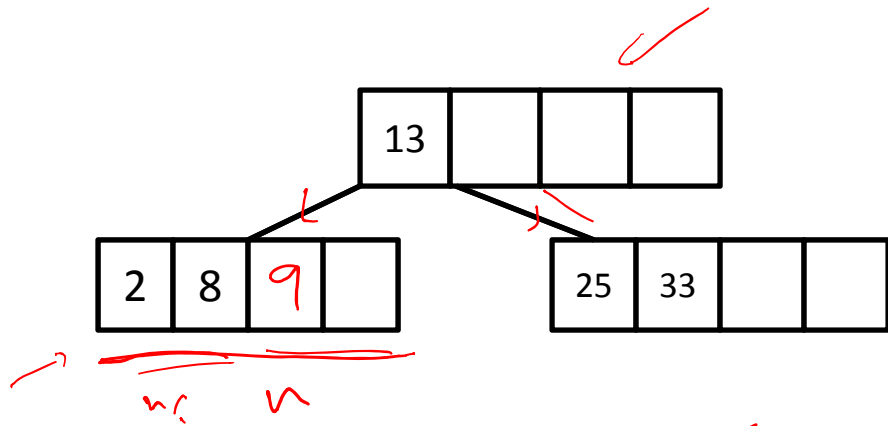
All of them should be done with fewest disk accesses as possible

B-trees and storage



- Each node in a B-tree fits into a block (i.e., if you get part of the node, you get it all)
- Search tree property ↙
- Keys in each node are sorted

The goal of B-Trees

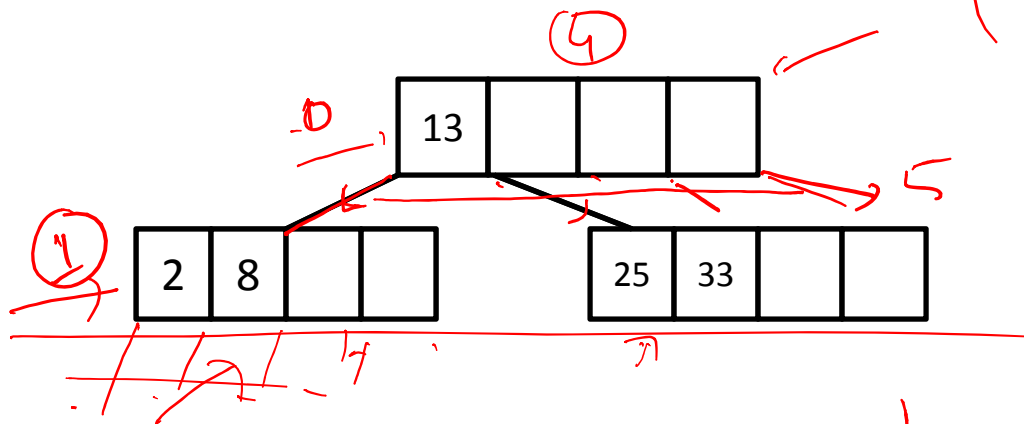


- Always at least half full
- Perfectly Balanced
- Few levels

T

Properties of an m-order B trees

1. The root has at least 2 sub-trees, unless it is a leaf
2. All leaves are at the same level
3. Each node (leaf as well as non leaf) holds $k-1$ keys where $\lceil m/2 \rceil \leq k \leq m$
4. Each non-leaf node additionally holds k pointers to subtrees where $\lceil m/2 \rceil \leq k \leq m$.



($k-1$ keys, k pointer)
 m

$$m=5$$

$$(m-1)$$

(4)

$$km=5$$

$$\lceil \frac{5}{2} \rceil = 3$$

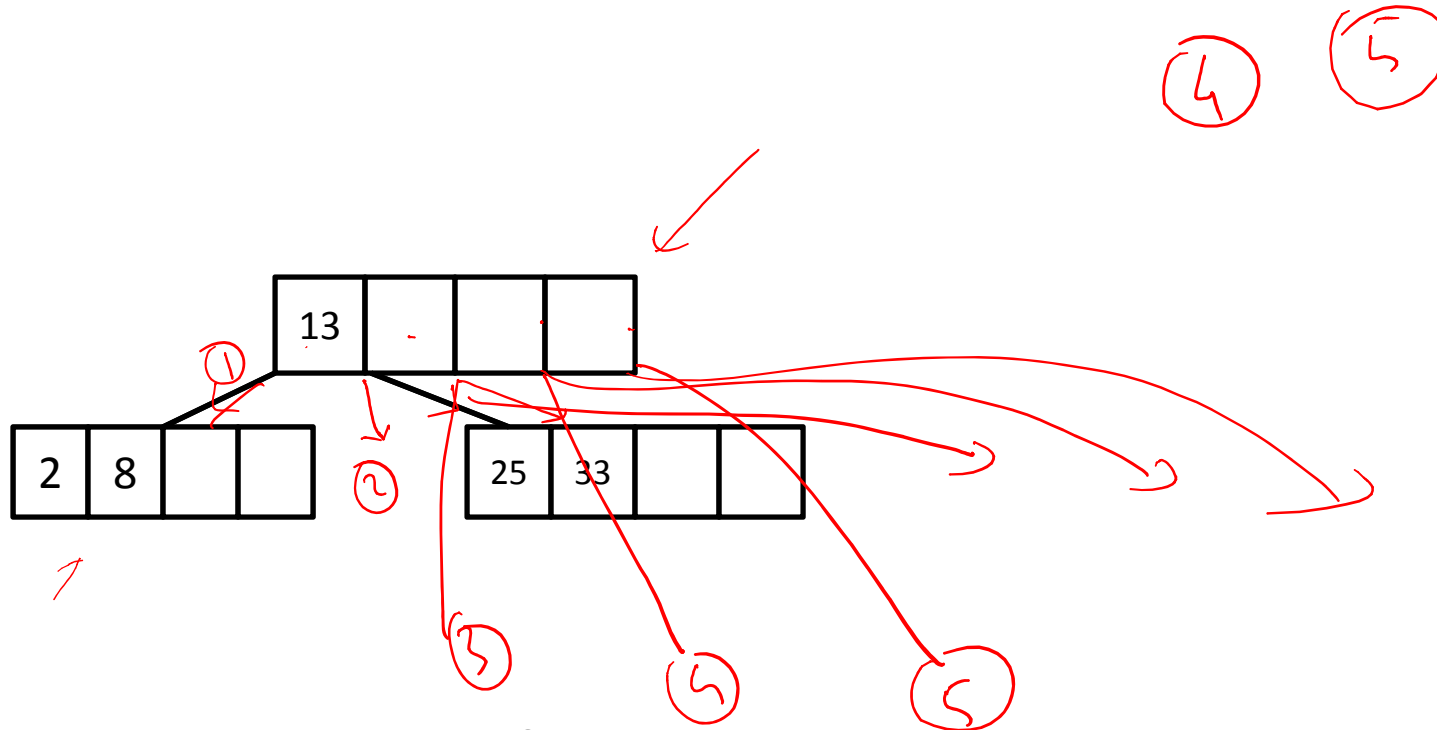
(2)

(3)

(2)

(5)
(4)

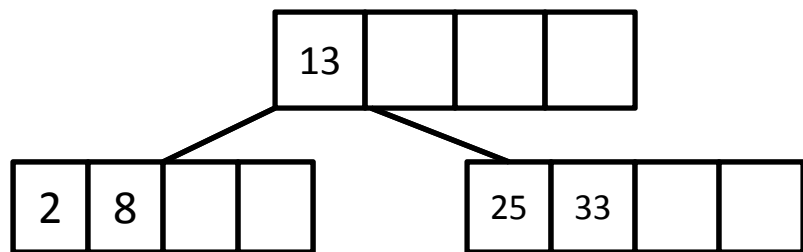
Order of B-trees



What order is this B-tree?

- A. 2
- B. 3
- C. 4
- ☒ D. 5
- E. 6

Insertion and properties of B-trees



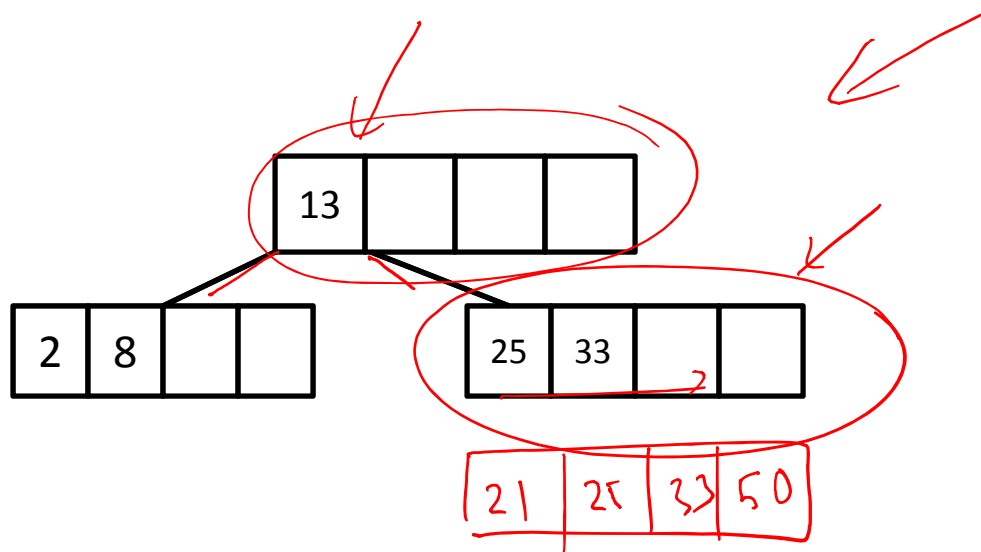
$$\begin{aligned} m &= 5 \\ 3 &\leq k \leq 5 \\ 2 &\leq (k-1) \leq 4 \end{aligned}$$

What is the minimum number of keys each non-root node in this B-tree is allowed to store?

- A. 0
- B. 1
- ☒ C. 2
- D. 3
- E. 4

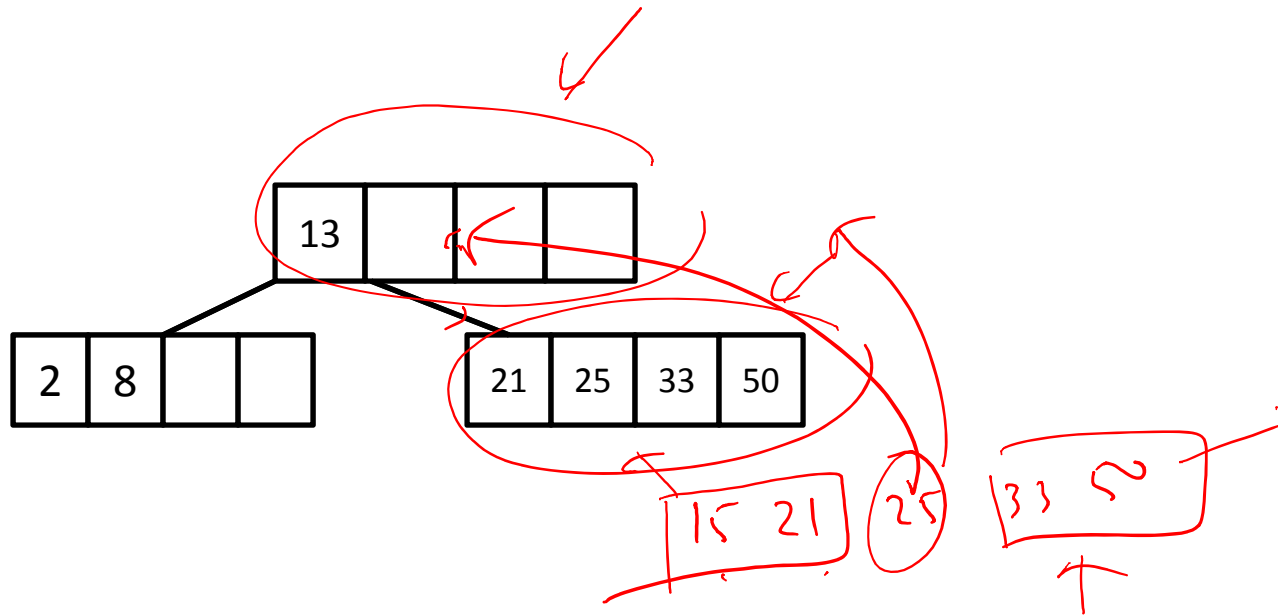
How can we guarantee this?

Insertion and properties of B-trees



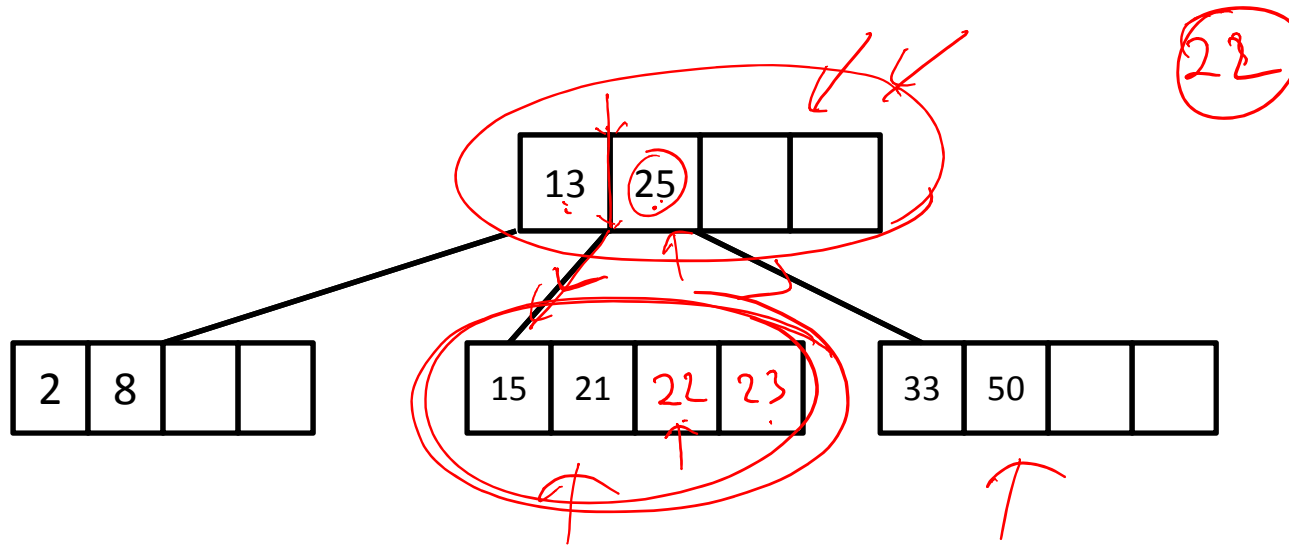
Insert 21 into this B-tree. Then insert 50

Insertion and properties of B-trees



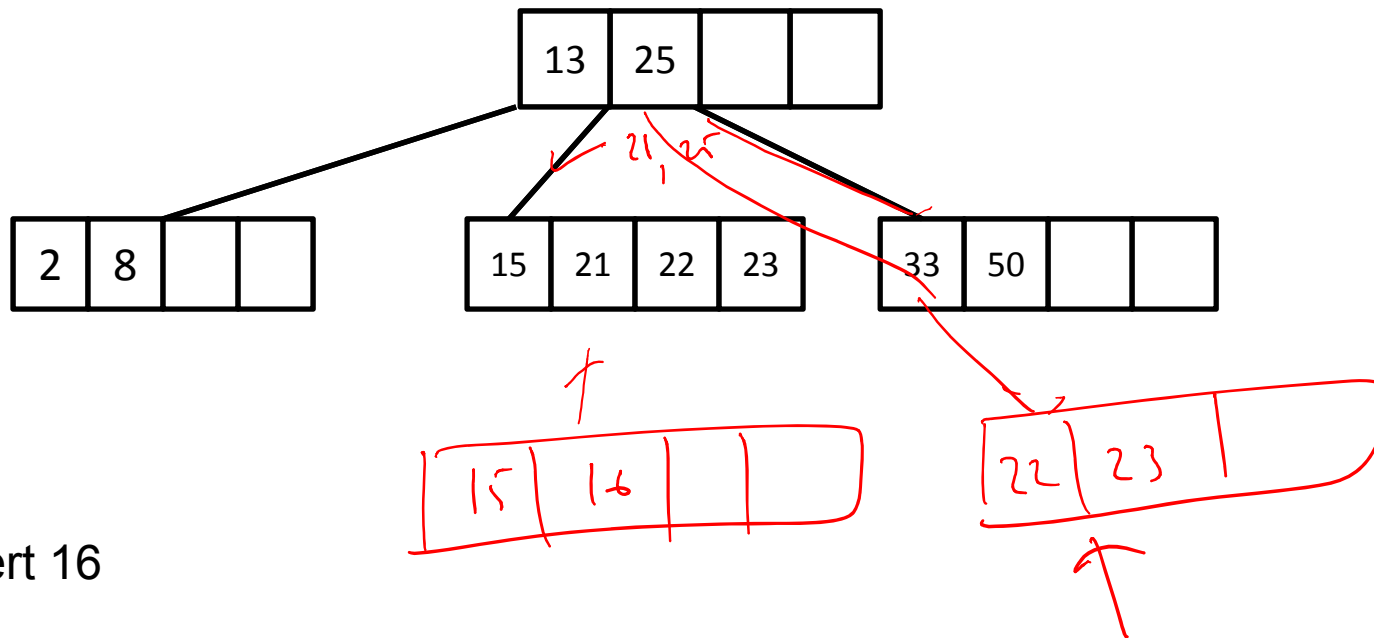
Insert 15 into this B-tree

Insertion and properties of B-trees

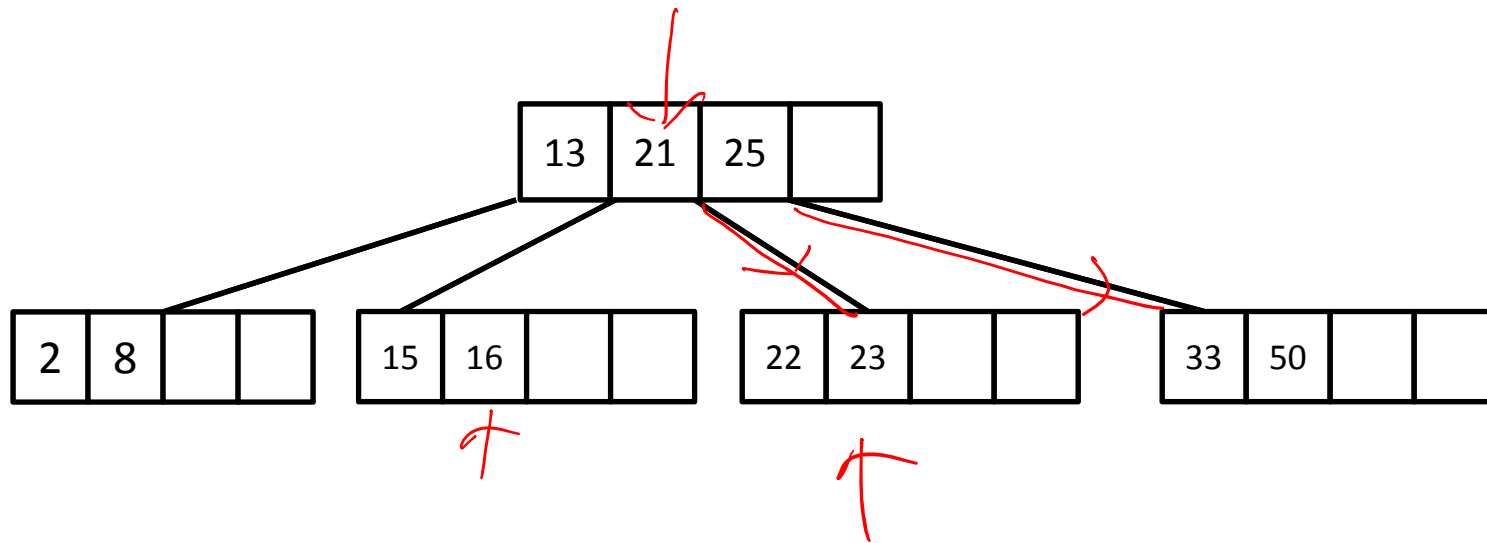


Insert 22 and 23

Insertion and properties of B-trees

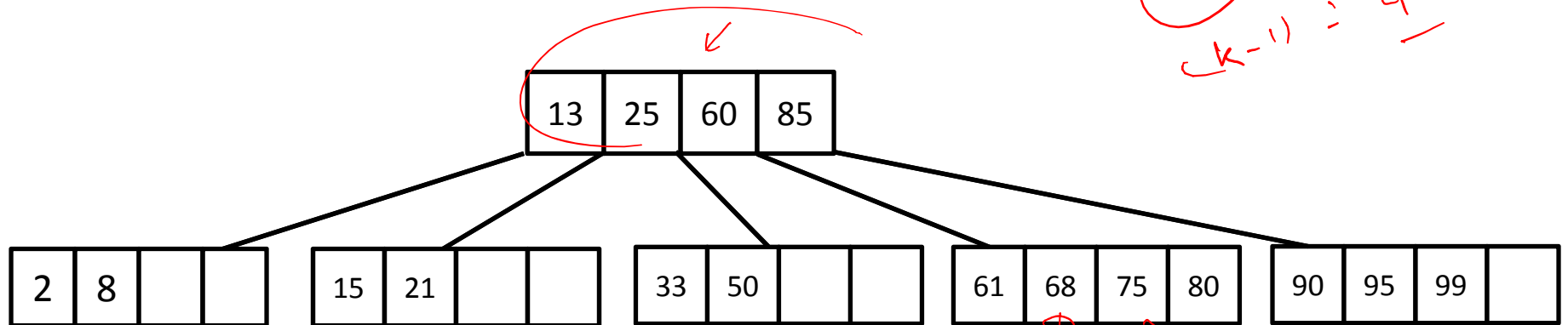


Insertion and properties of B-trees



Insert 16, after

Insertion and properties of B-trees

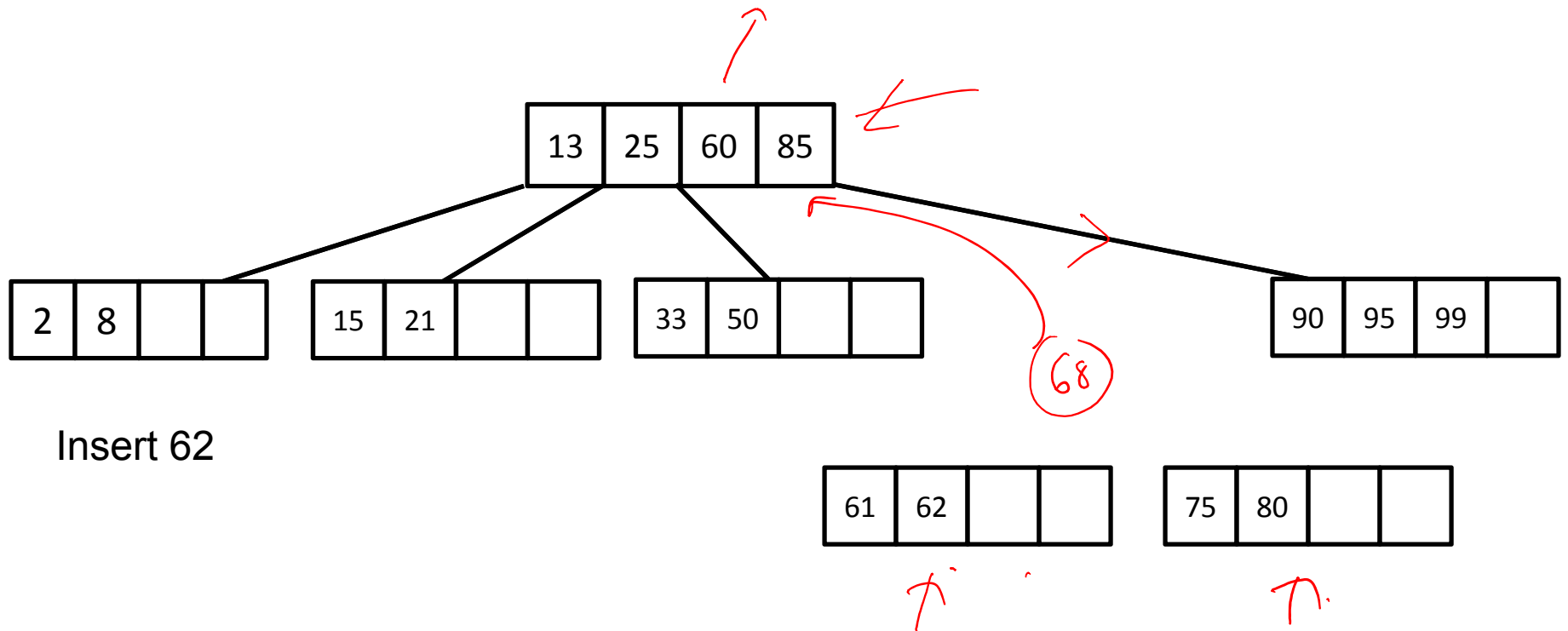


Insert 62

Which key will be promoted up?

- A. 61 B. 62 C. 68 D. 75 E. 80

Insertion and properties of B-trees

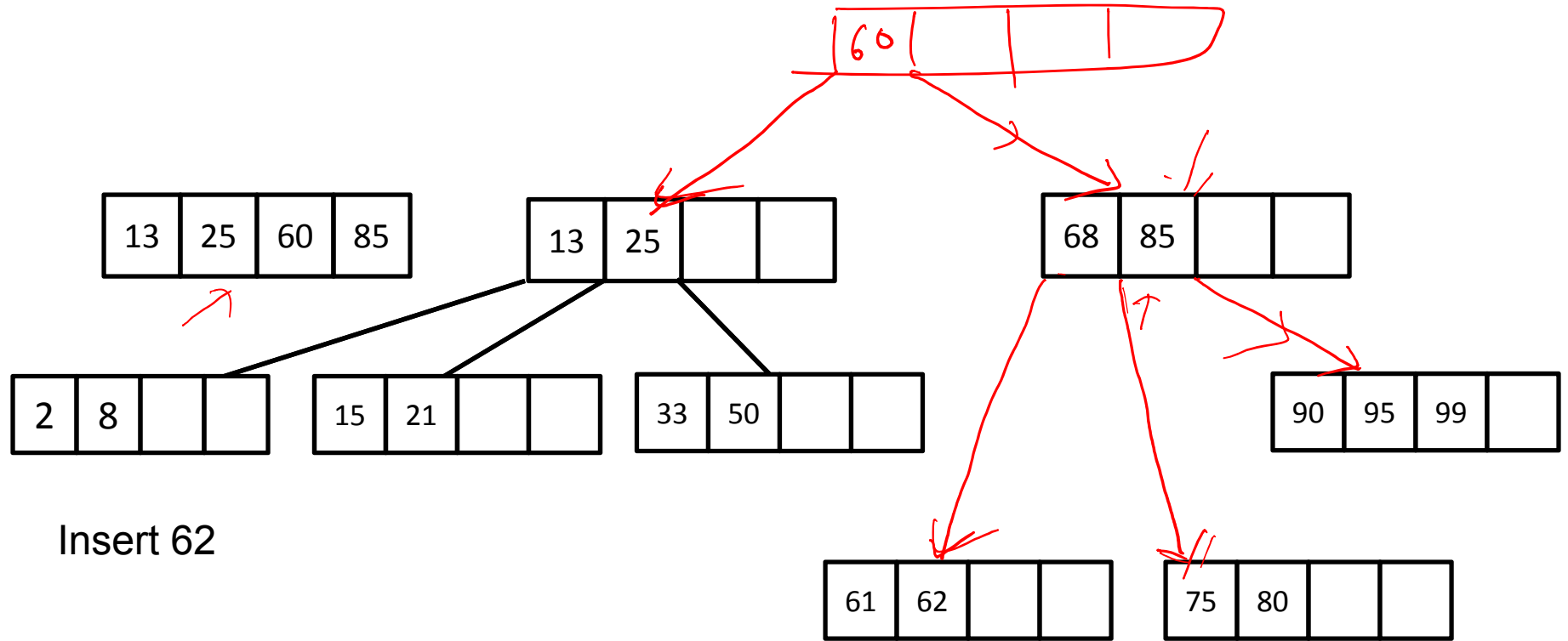


Insert 62

Which key will be promoted up?

- A. 13 B. 25 C. 60 D. 85 E. 68

Insertion and properties of B-trees



$m = 258$



7

B-Tree performance

- The time savings in a B-Tree comes from *efficiently reading lots of data from disk*
- When B-Trees are stored in memory they are typically comparable to other search trees
- When they have to access disk they are a big win

(For details see the slides here: 

<http://cseweb.ucsd.edu/users/kube/cls/100/Lectures/lec17/lec17.pdf> 

You will be responsible for the general ideas behind the tradeoffs of their design, but not the details. Example questions next class.)