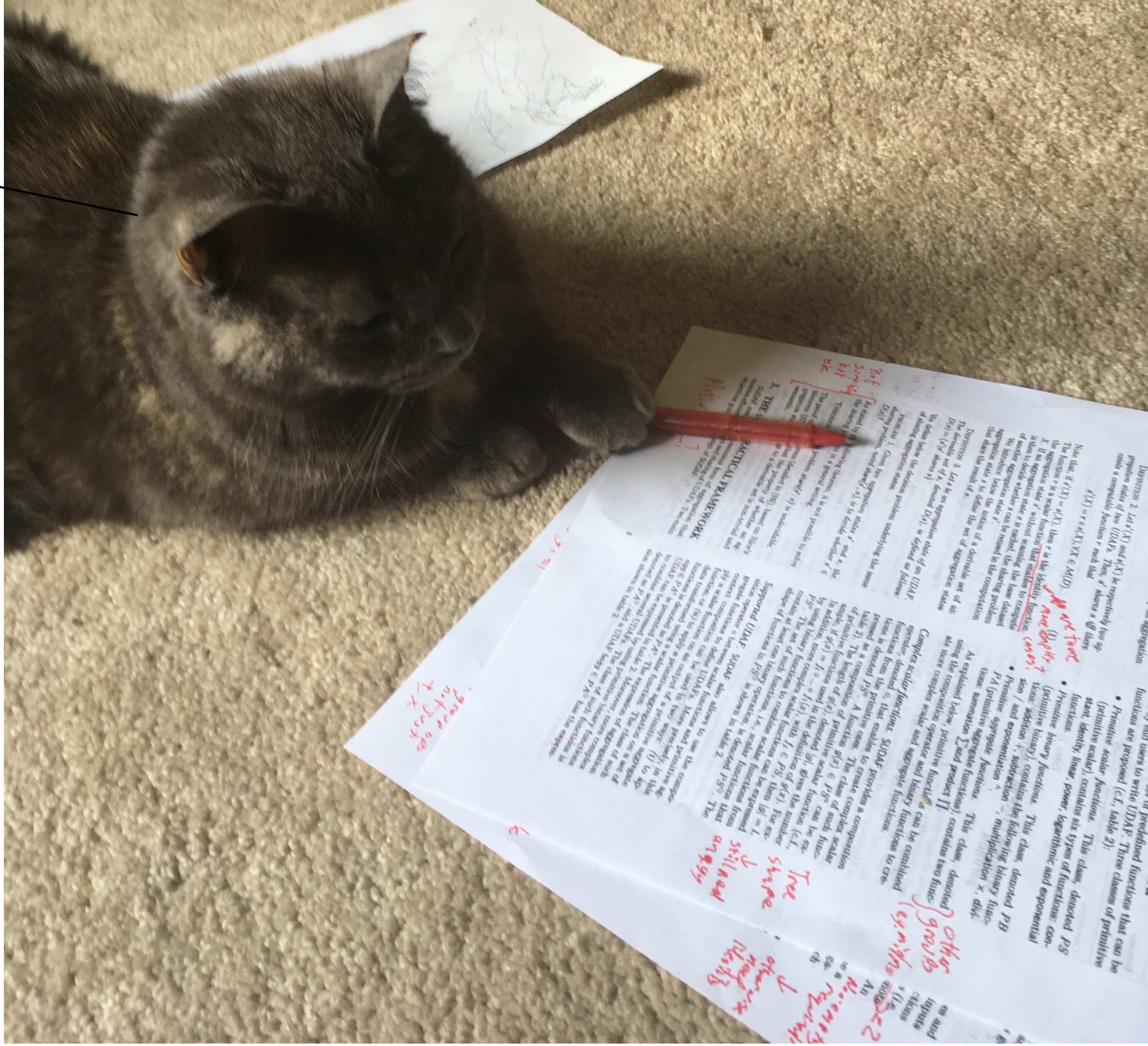
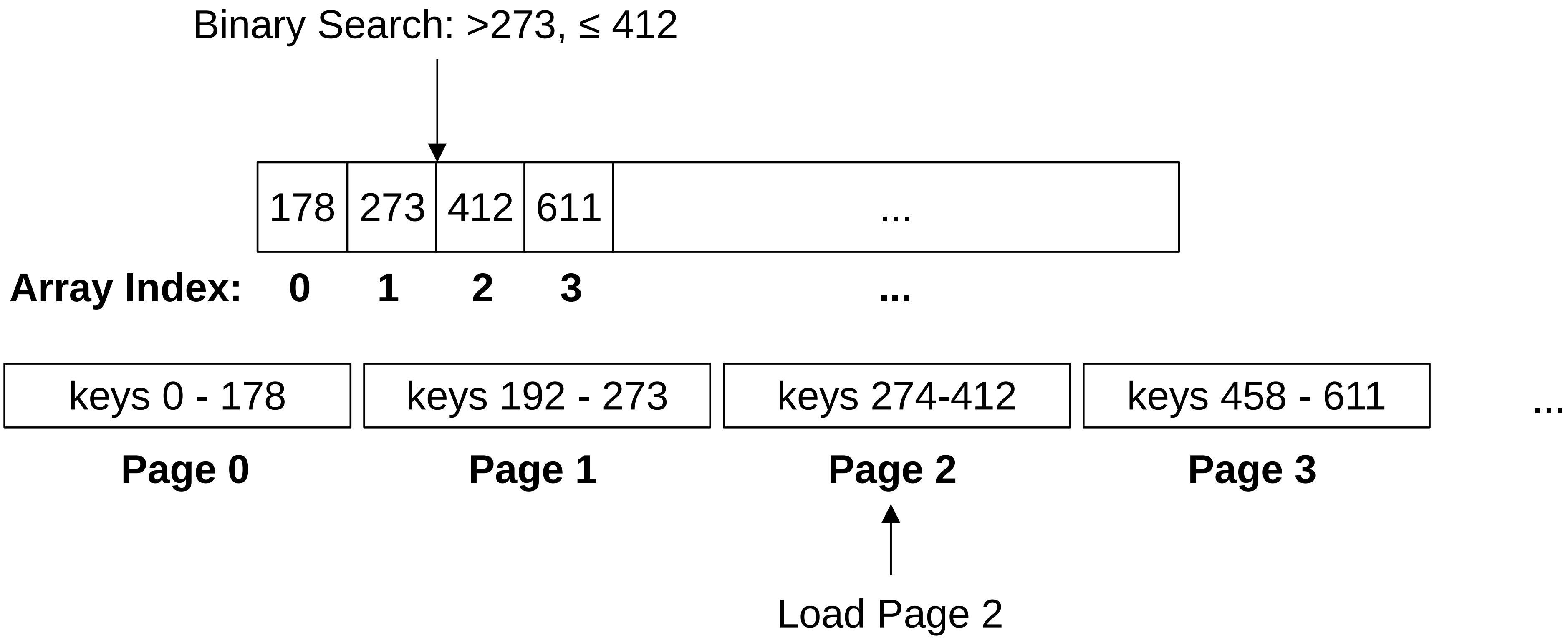
submit exams with kibble plzkthx



CSE 250 Lecture 37

B+Trees

Fence Pointers Example



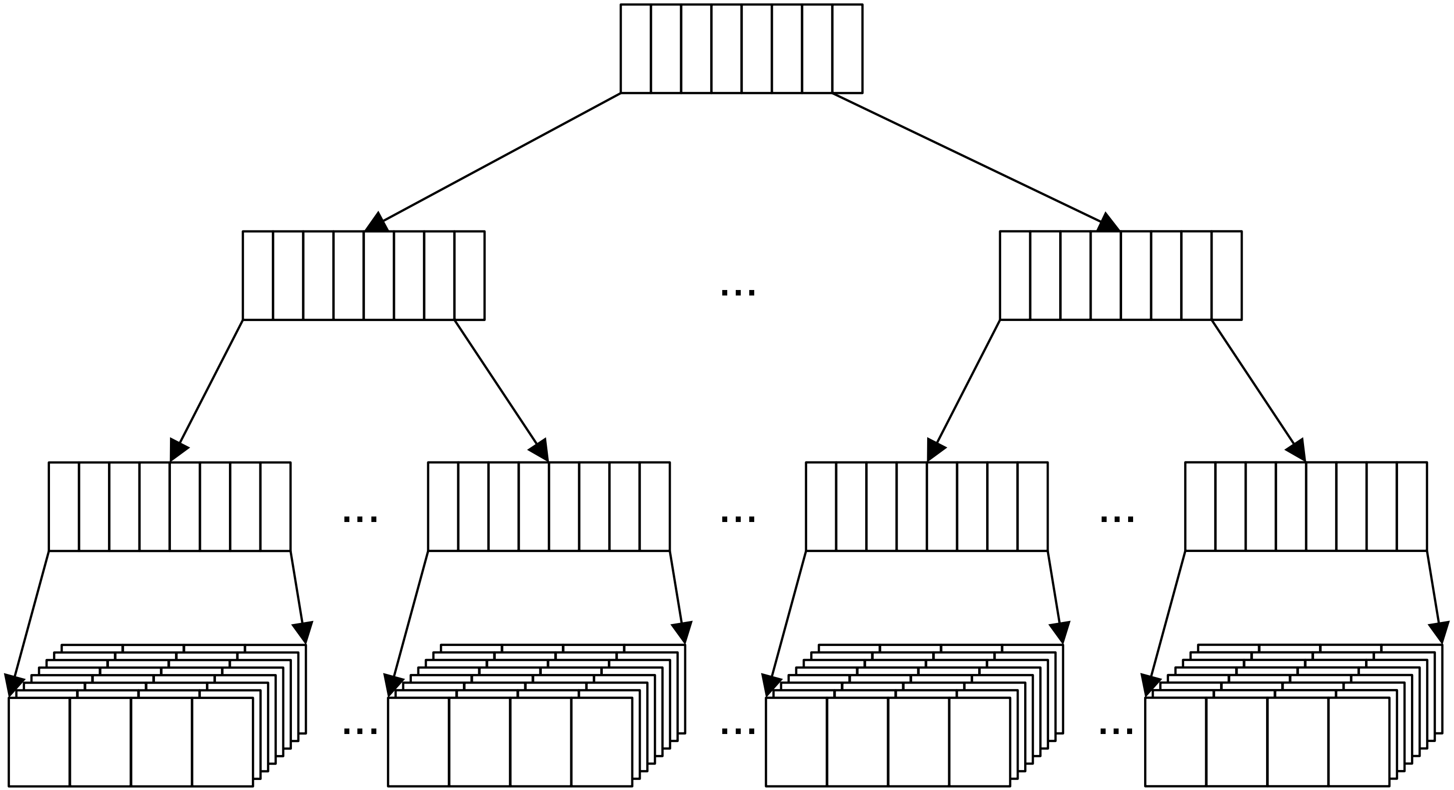
ISAM Index

Binary Search @ Level 0
to find a Level 1 page

Binary Search @ Level 1
to find a Level 2 page

Binary Search @ Level 2
to find a Data page

Binary Search @ Data
to find the record



What does this look like?

ISAM Index

$$n = C_{data} C_{key}^{max+1}$$

$$\frac{n}{C_{data}} = C_{key}^{max+1}$$

$$\log_{C_{key}}\left(\frac{n}{C_{data}}\right) = max + 1$$

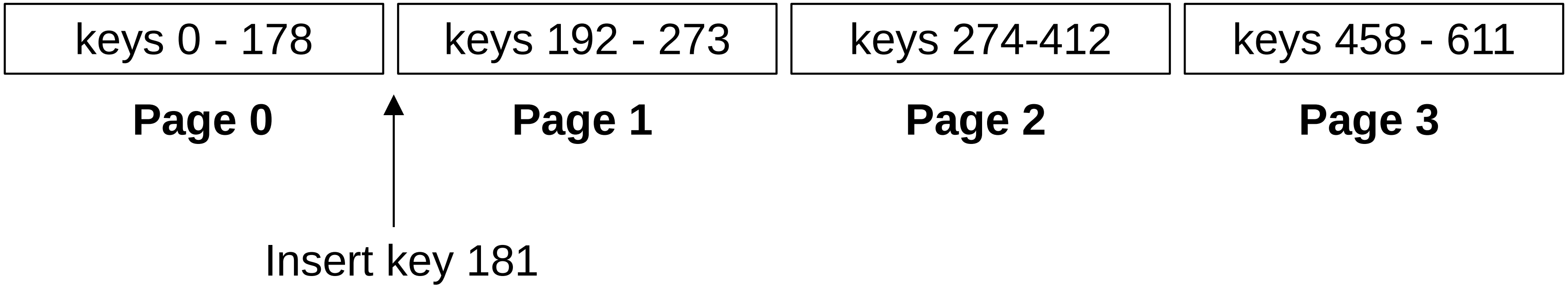
$$\log_{C_{key}}(n) - \log_{C_{key}}(C_{data}) = max + 1$$

Number of Levels: $O\left(\log_{C_{key}}(n)\right)$ **= IO Complexity**

ISAM Index

What if the data changes?

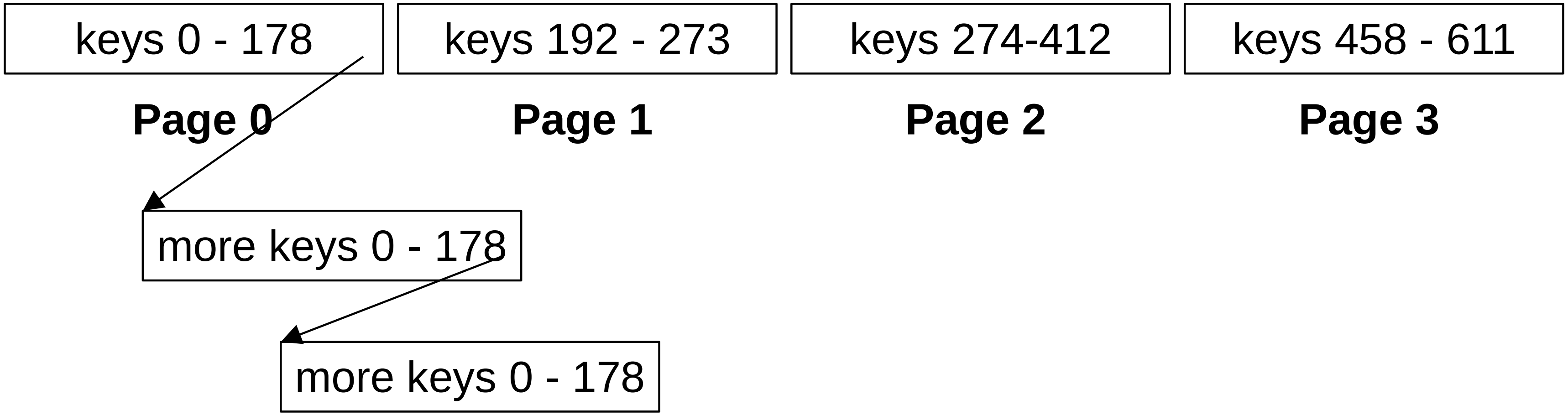
Putting Data on Pages



Putting Data on Pages

- **Idea:** Can keep “free” space on each page
  - ... what happens when the space fills up?

Putting Data on Pages



Putting Data on Pages

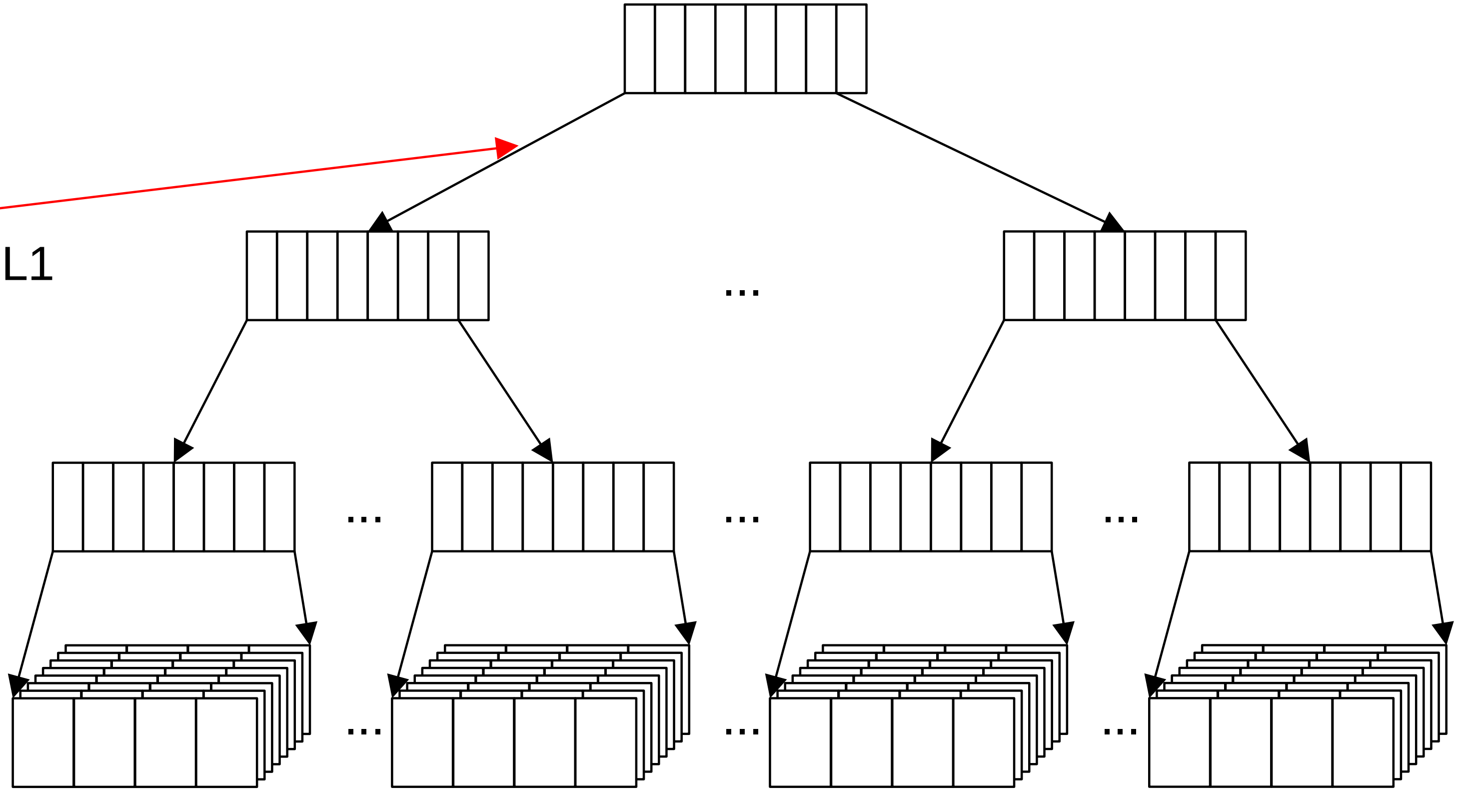
- **Idea:** Can keep “free” space on each page
  - ... what happens when the space fills up?
- **Idea:** Can use linked lists to store overflow
  - ... but that can makes the IO complexity O(n)
- **Idea:** Rearrange the tree

Dynamic Page Allocation

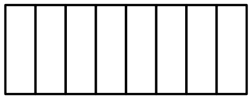
- Treat the disk as an ADT:
  - **allocate(): PageID**
    - Allocates a page in the data file and returns its position
  - **load[T](page: PageID): T**
    - Reads in a 4k chunk of data (e.g., T = Array[Byte])
  - **write[T](page: PageID, data: T)**
    - Writes a 4k chunk of data to the page

ISAM Index/Dynamic Page Allocation

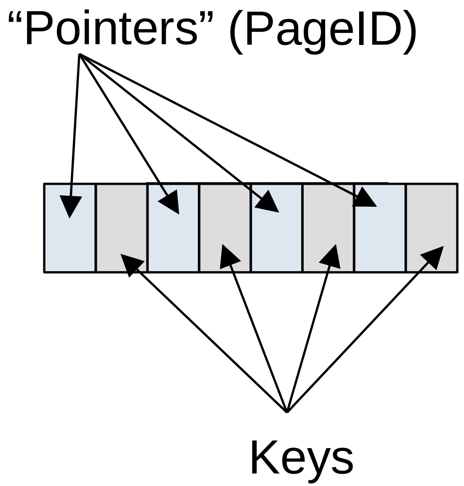
Problem:
Keys at L0 no longer "line up" with pages at L1



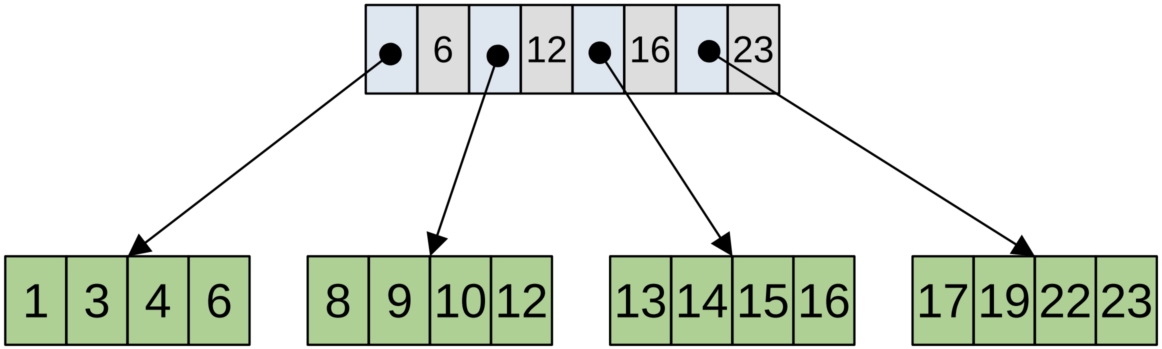
Pointers to Pages



Pointers to Pages



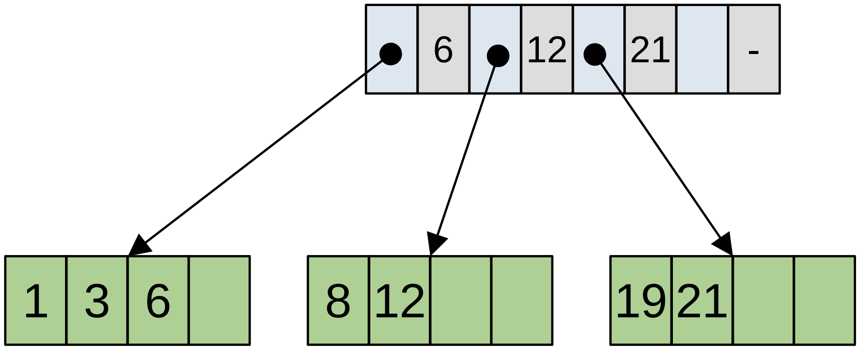
Pointers to Pages



Pointers to Pages

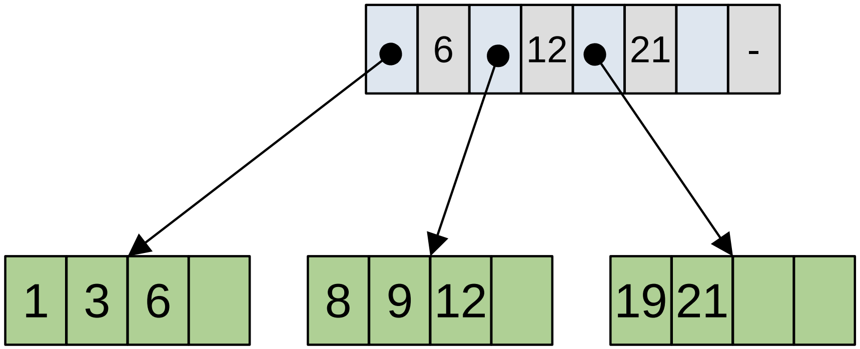
```
class DirectoryPage[K](size:Int)
{
  val entries = new Array[ (PageID, K) ](size)
}
```

Free Space Revisited



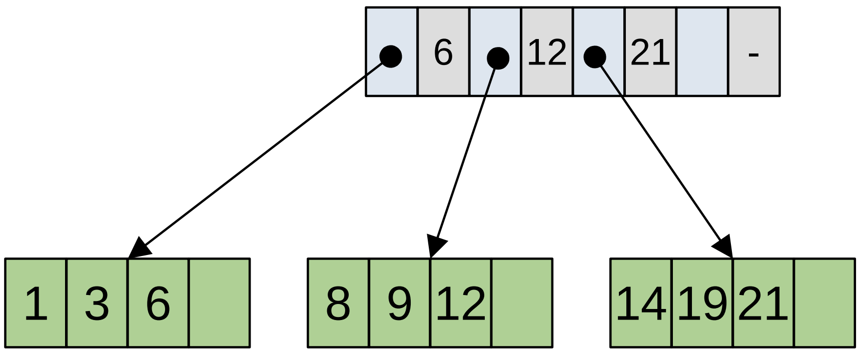
Add 9

Free Space Revisited



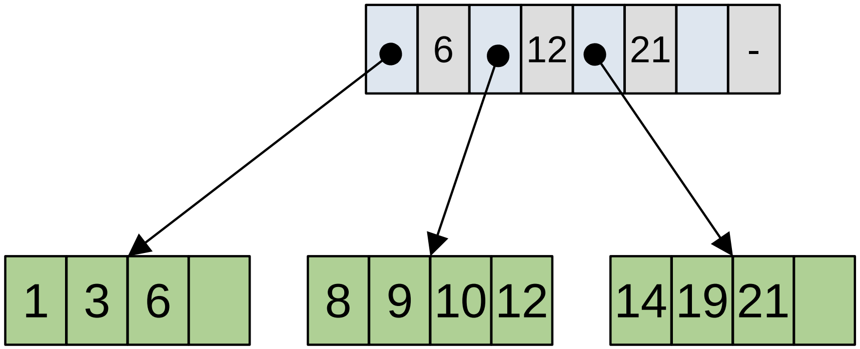
Add 14

Free Space Revisited



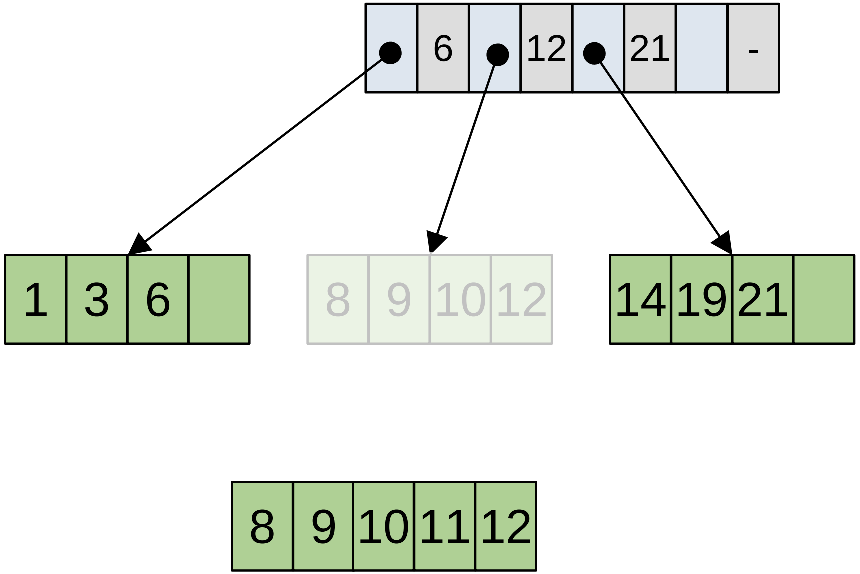
Add 10

Free Space Revisited

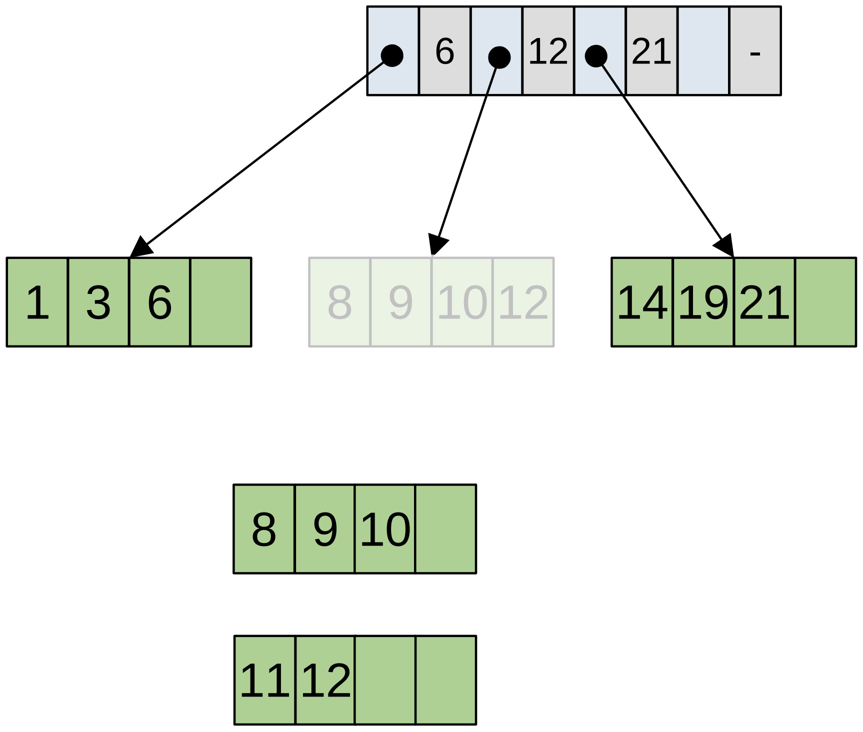


Add 11

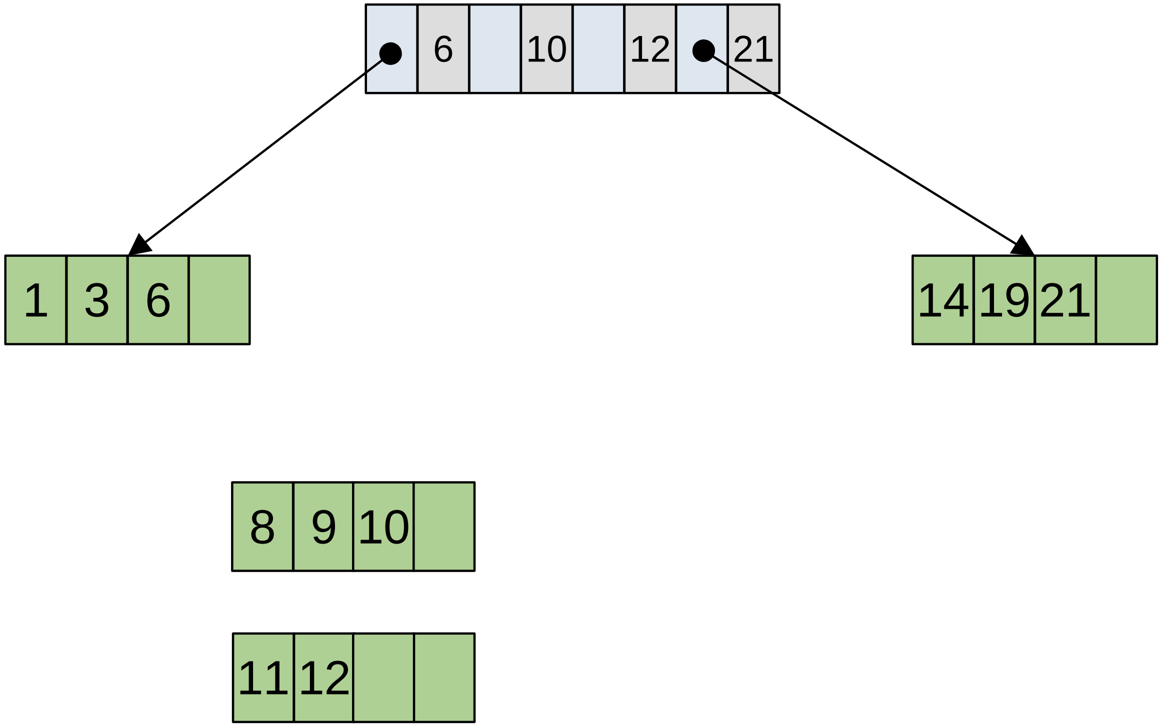
Free Space Revisited



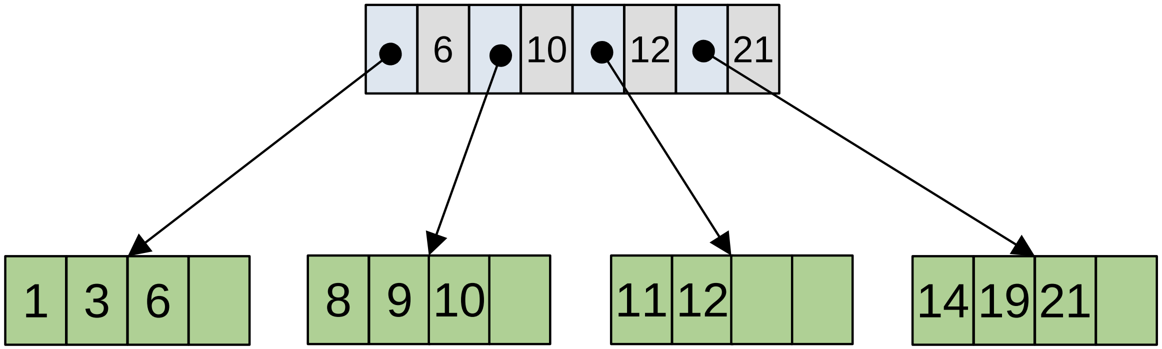
Free Space Revisited



Free Space Revisited

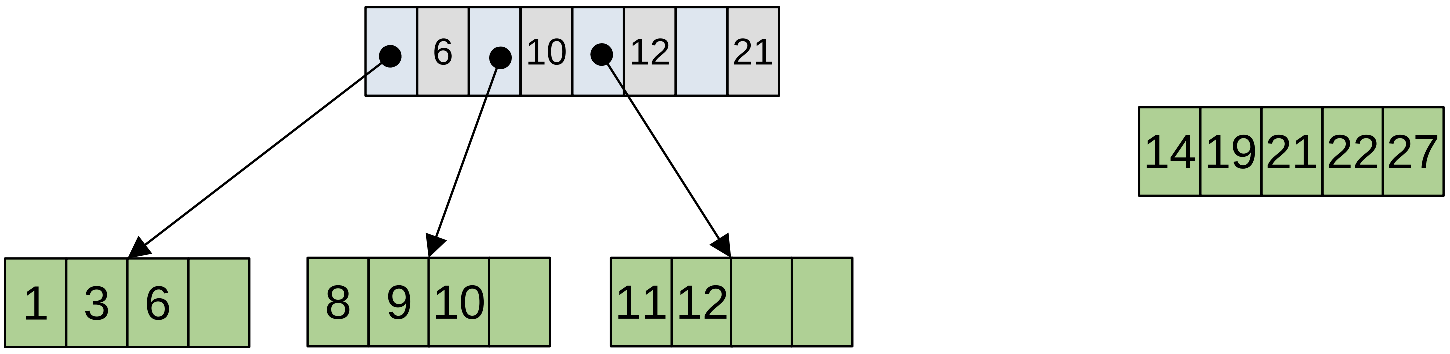


Free Space Revisited

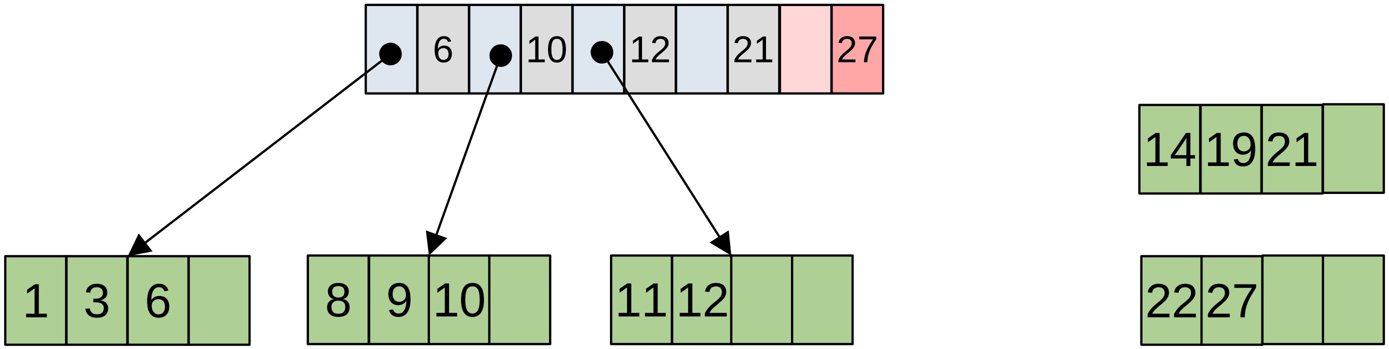


Add 22, 27

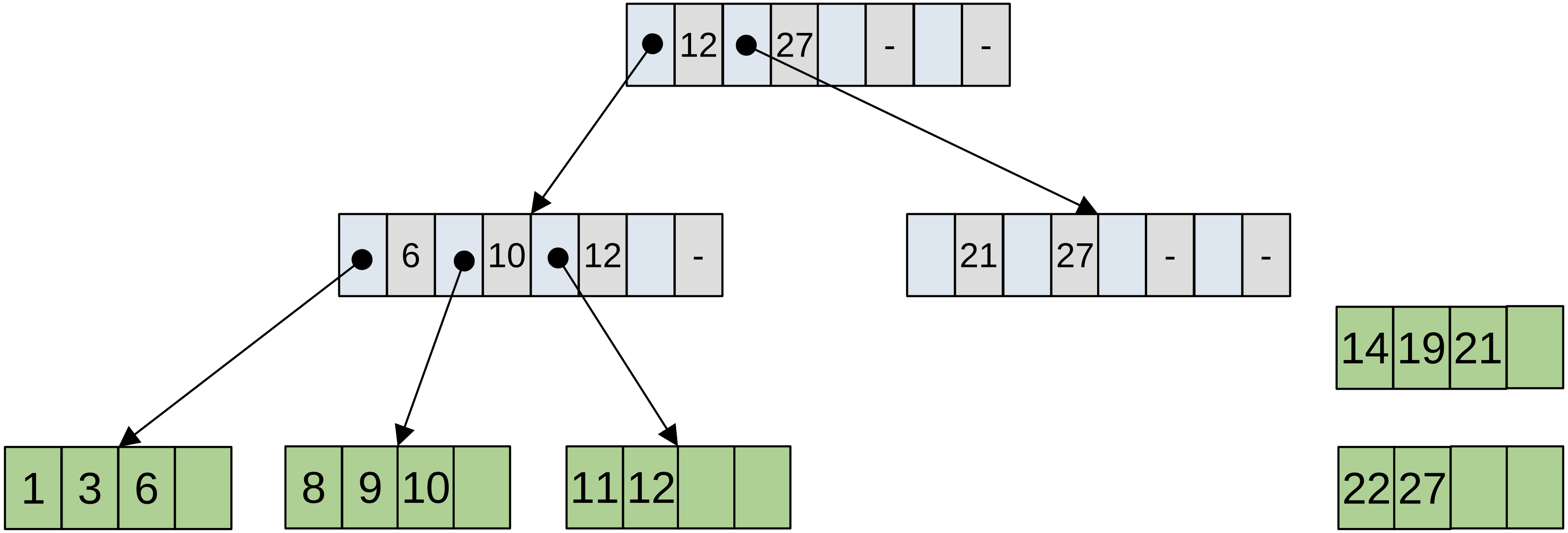
Free Space Revisited



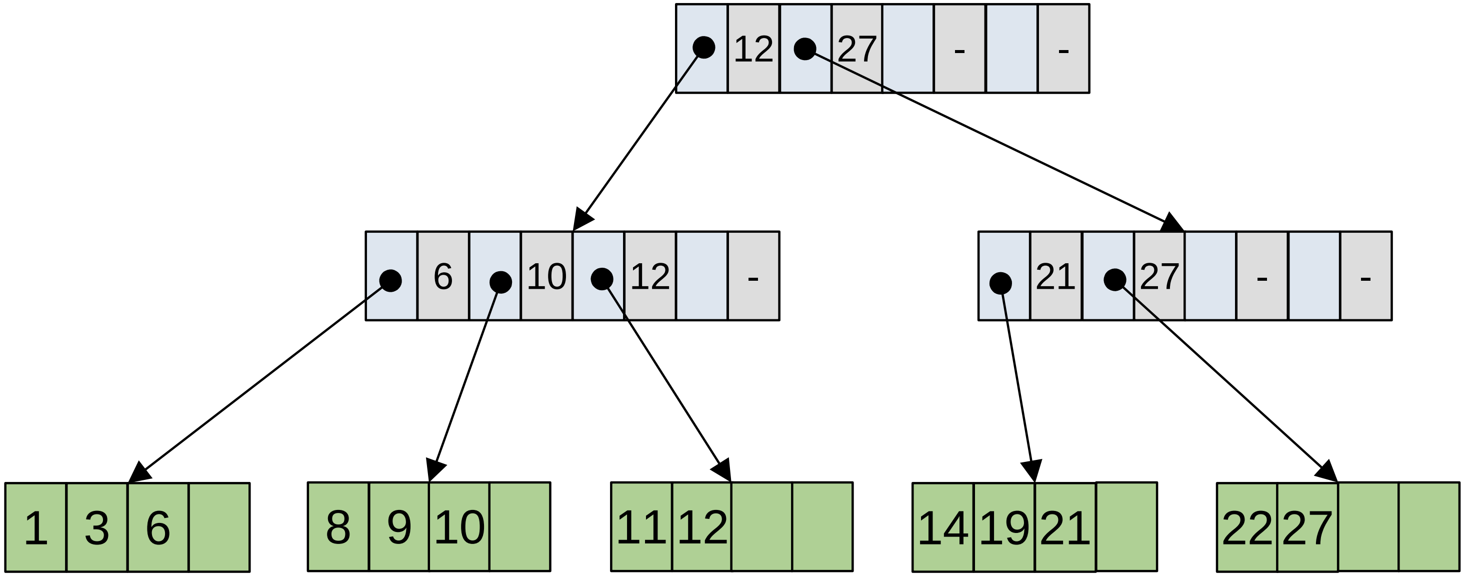
Free Space Revisited



Free Space Revisited



Free Space Revisited



"B+ Tree" (Almost)

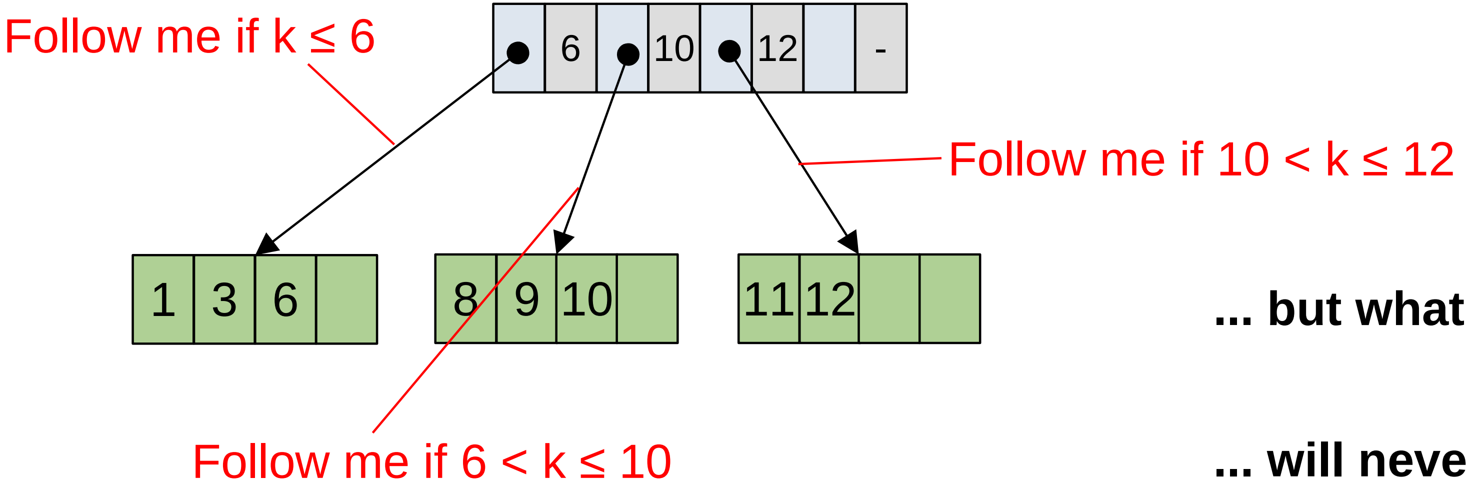
B+ Trees

- **Insert**:
  - Find the page that the record belongs on
  - Insert record there
  - If full, “split” the page
    - Insert additional separator in parent directory page
    - If full, “split” the directory page and repeat with parent
      - If “root split” create a new parent node

B+ Trees

- **Observation**: Don't need the biggest key on each page

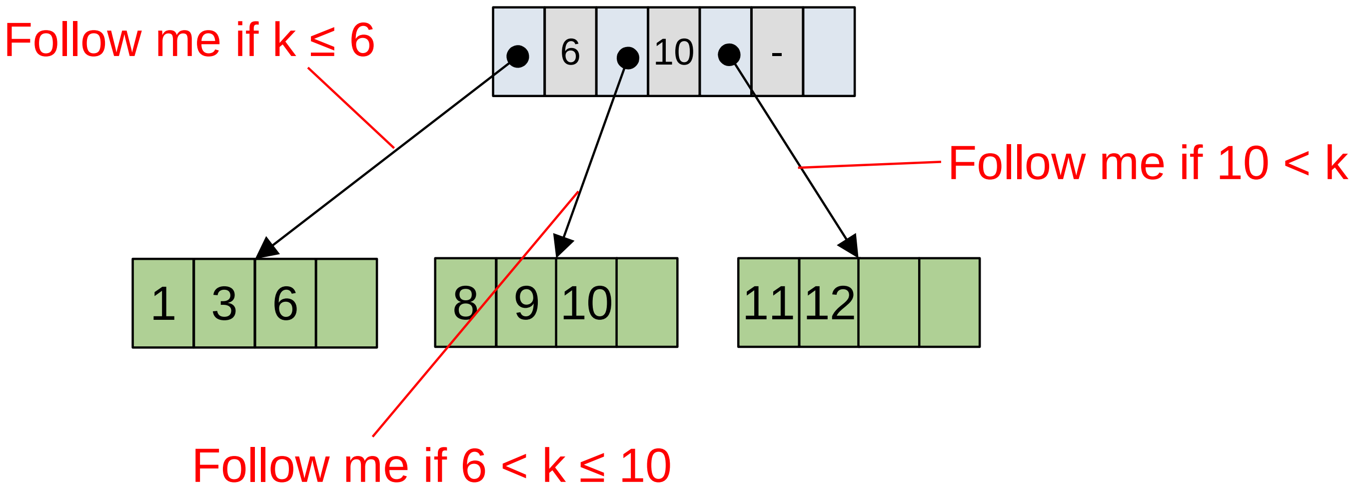
B+ Trees



... but what if k > 12?

... will never happen!

B+ Trees

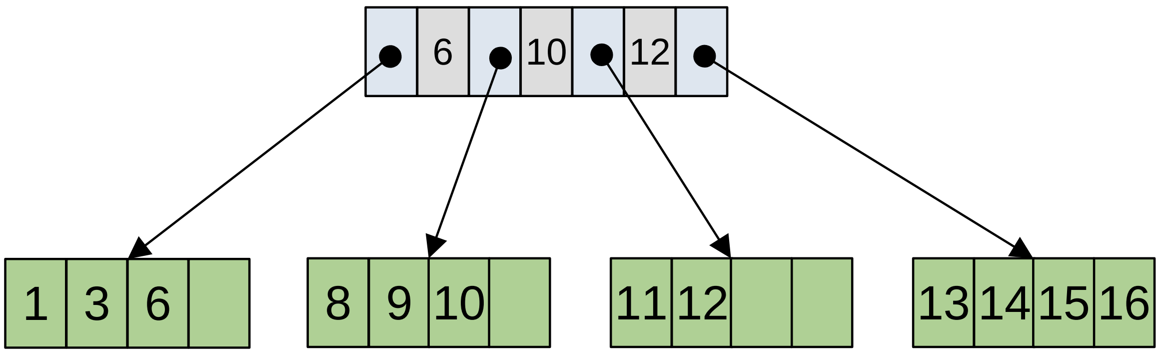


Follow me if k ≤ 6
6
10
-
Follow me if 10 < k
1
3
6
8
9
10
11
12
Follow me if 6 < k ≤ 10

B+ Trees

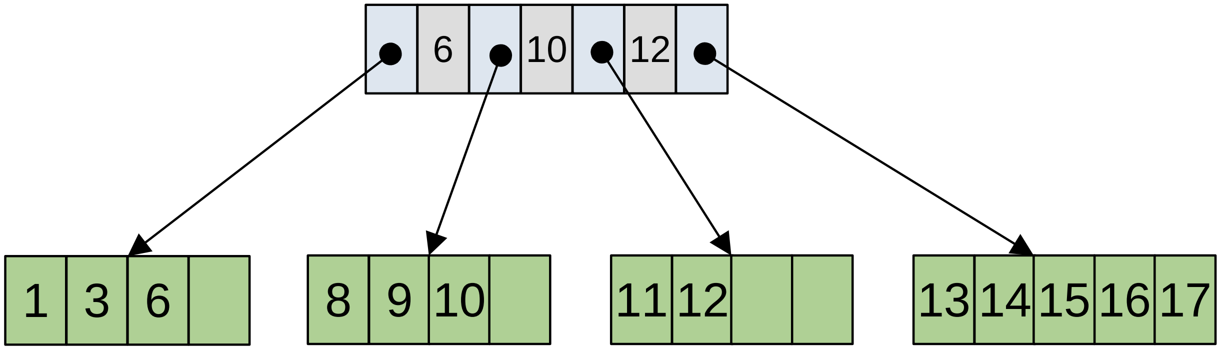
- **Observation**: Don’t need the biggest key
- **Question**: What if the separator value is mispositioned?

B+ Trees

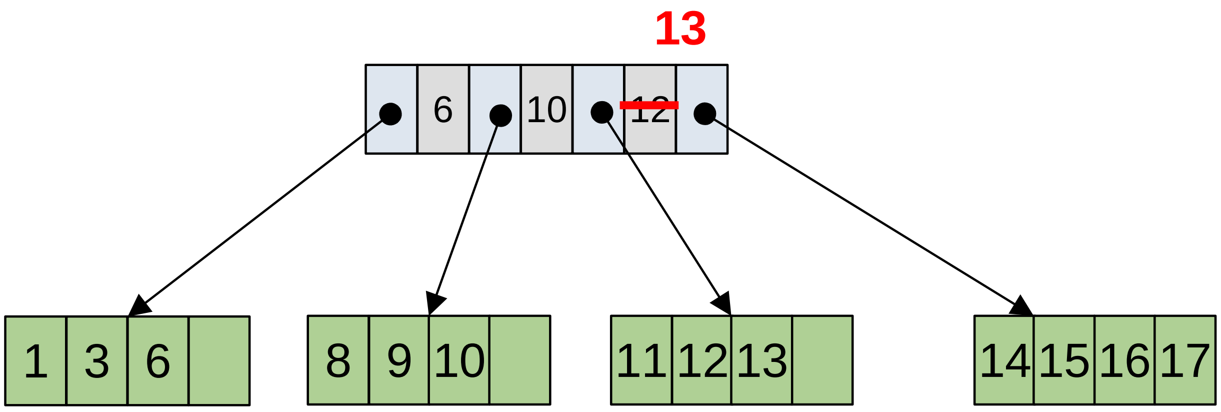


Add 17

B+ Trees



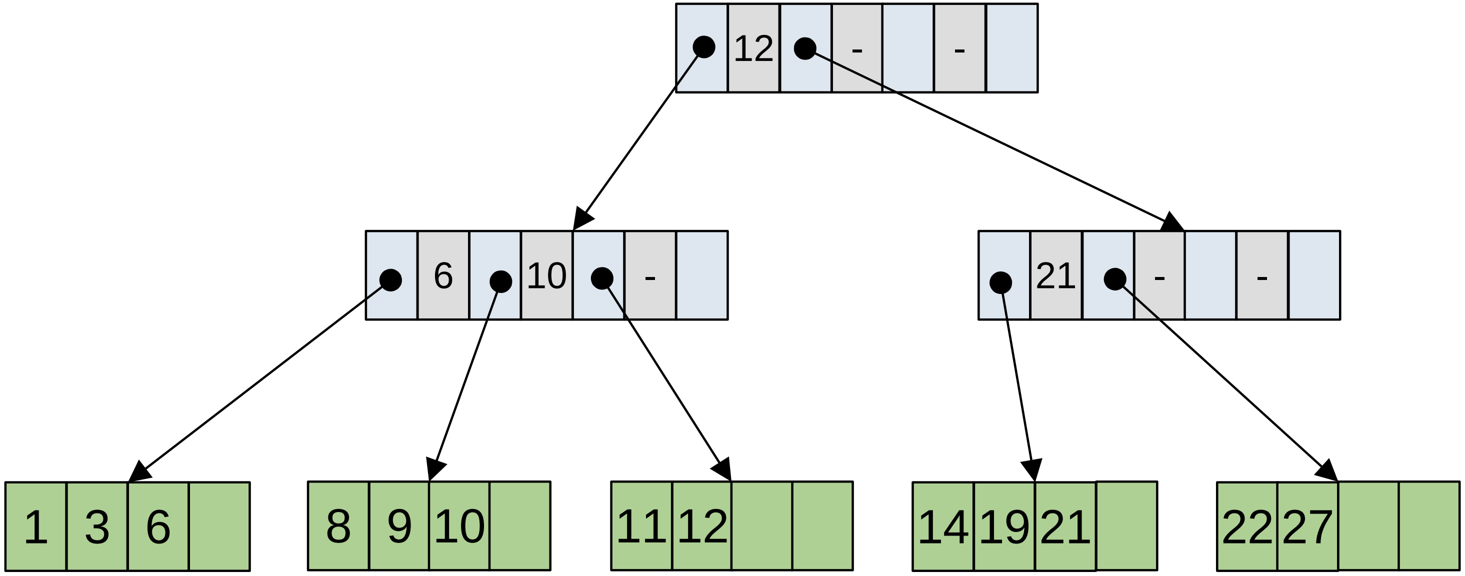
B+ Trees



B+ Trees

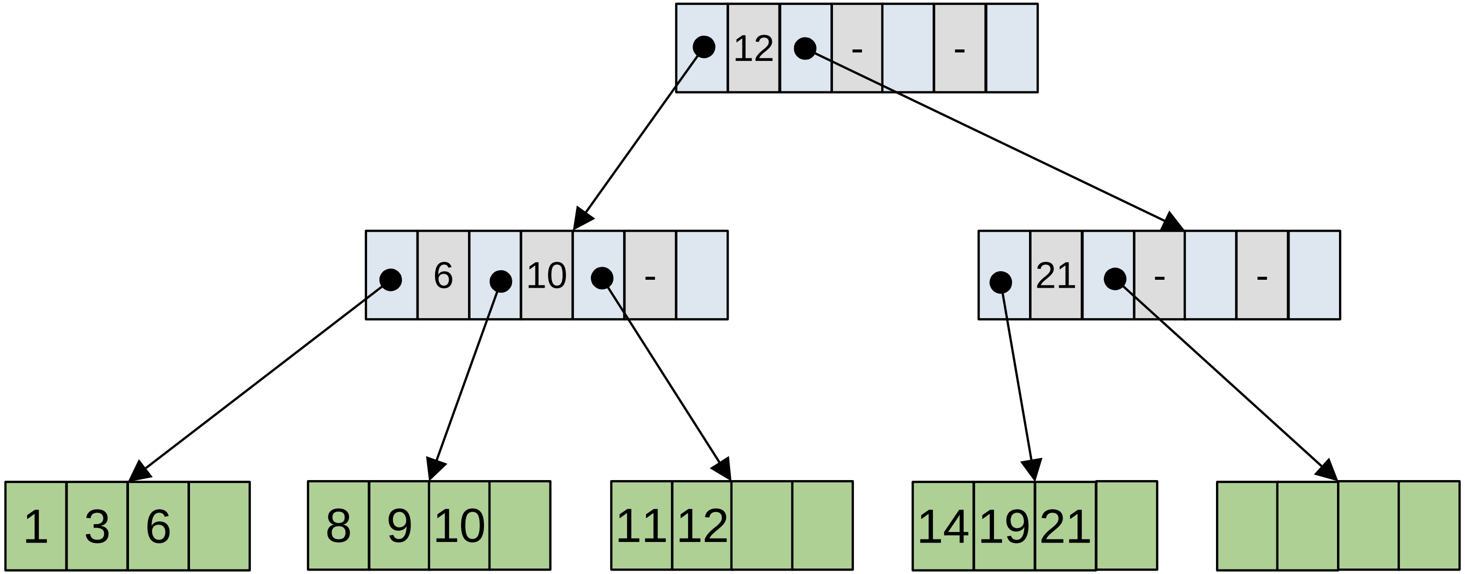
- **Observation**: Don’t need the biggest key
- **Question**: What if the separator value is mispositioned?
  - **Idea:** “Steal” space from adjacent nodes
- **Question**: What happens when we delete records?

B+ Trees

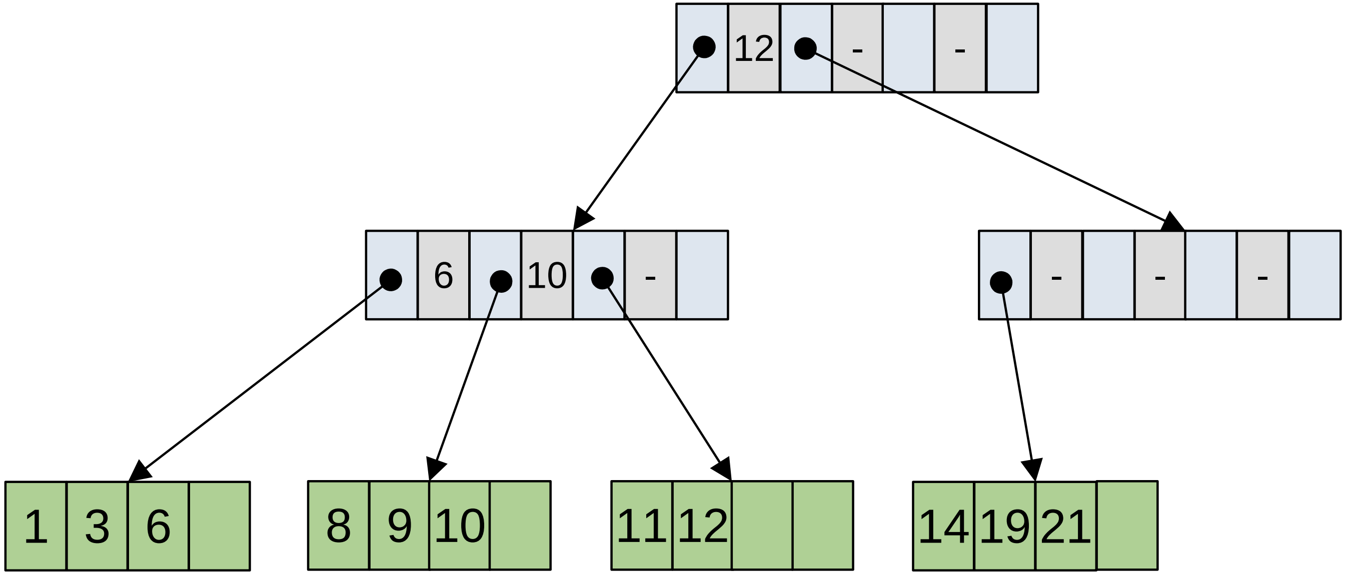


Delete 27, 22

B+ Trees

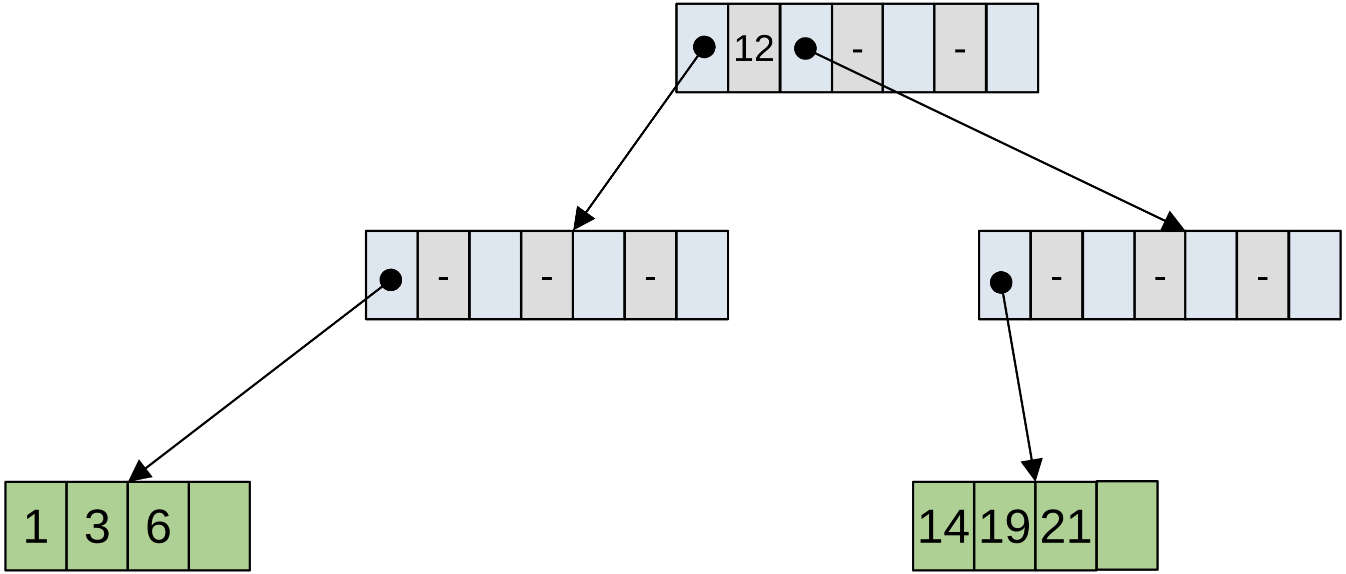


B+ Trees



Delete 8-12

B+ Trees



O(log(n)) reads required per search <u>for the biggest n in the tree's history</u>

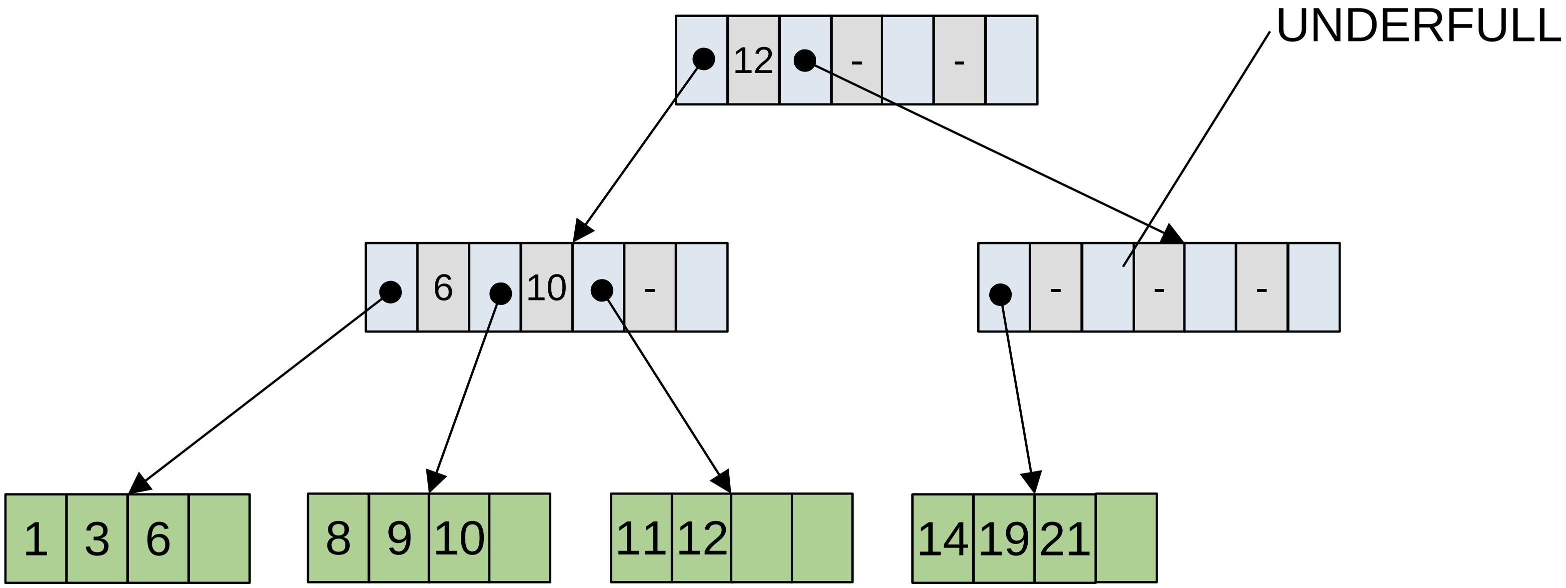
B+ Trees

- **Observation**: Don’t need the biggest key
- **Question**: What if the separator value is mispositioned?
  - **Idea:** “Steal” space from adjacent nodes
- **Question**: What happens when we delete records?
  - **Observation**: The tree becomes unbalanced
    - **Idea**: “Minimum Fill”

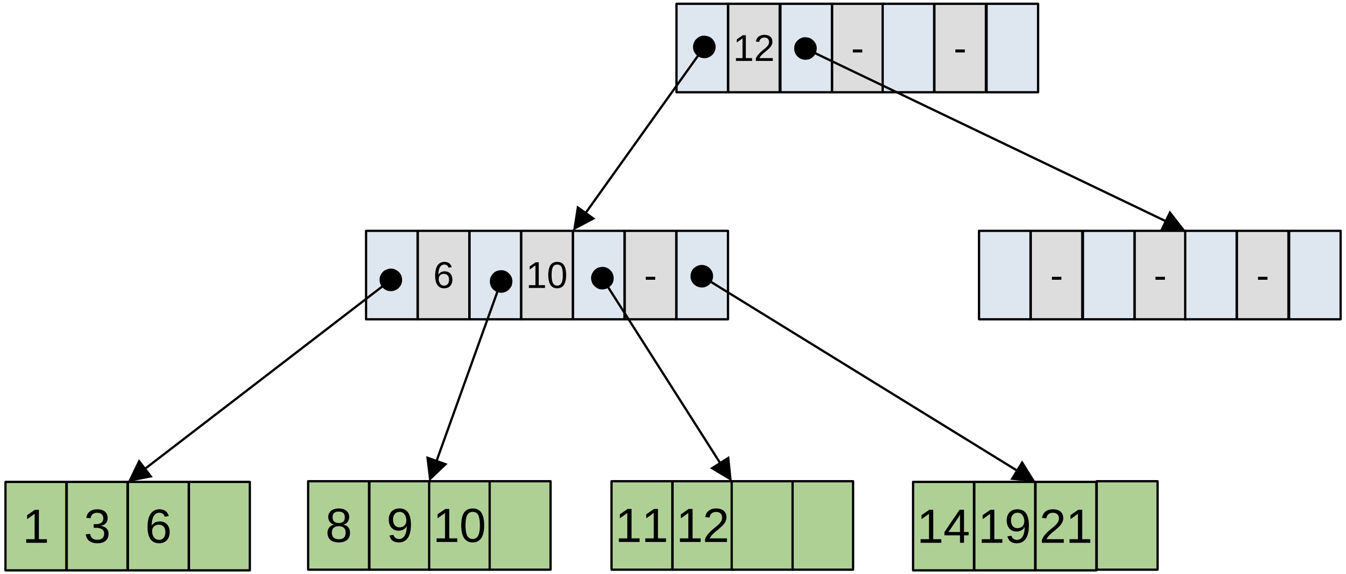
B+ Trees Minimum Fill

- Each directory & data node must have at least $^{C}/_{2}$ records
  - Exception: The root
- 1 page read at level 1: $^{1}/_{C/2}$ of the list left to search
- 1 more page read at level 2: $^{1}/_{C^2/4}$ of the list left to search
- 1 more page read at level 3: $^{1}/_{C^3/8}$ of the list left to search
- Max tree depth:
  - $O(\log_{C/2}(N))$

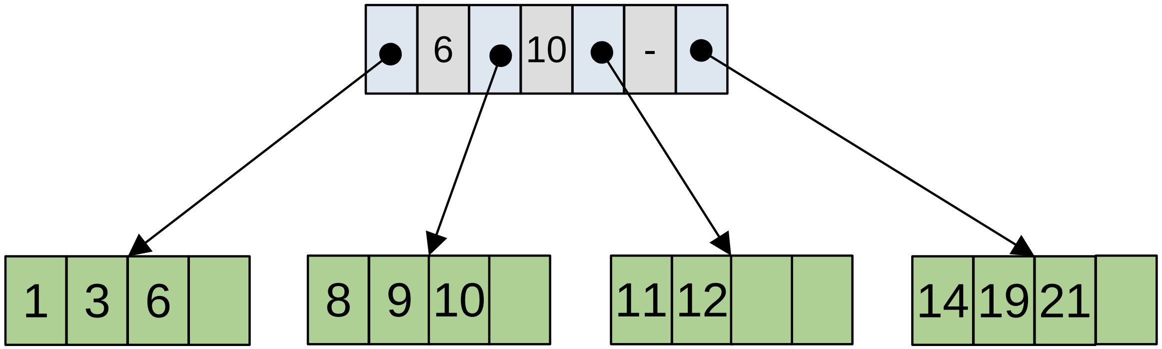
B+ Trees



B+ Trees



B+ Trees



B+ Trees

- **Delete**:
  - Find the page that the record is on
  - Delete record (if present)
  - If underfull, “merge” the page with a neighbor
    - If either neighbor at > $^{c}/_{2}$ entries (can’t merge)
      - “steal” entries from neighbor
    - If parent underfull, “merge” parent with neighbor
      - Repeat as needed
      - If “root merge” drop lowest layer