Indexing, Order-Based Indexes

CSE 4/562: Database Systems | Lecture 8

DB. Sys.: T.C.B.: Ch. 8.3-8.4, 14.1-14.2, 14.4

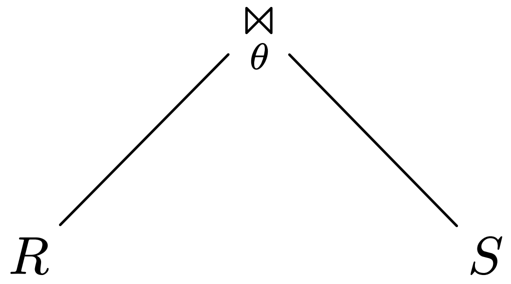
Quiz

Supporting $\sigma_\theta(R)$ and $\left(\ldots \underset{\theta}{\bowtie} R\right)$

Joins

$$\begin{tikzcd}
 & \sigma_\theta \arrow[d, dash] & \\
 & \times \arrow[dl, dash] \arrow[dr, dash] & \\
R & & S
\end{tikzcd}$$

Joins



Finding records in a table really fast

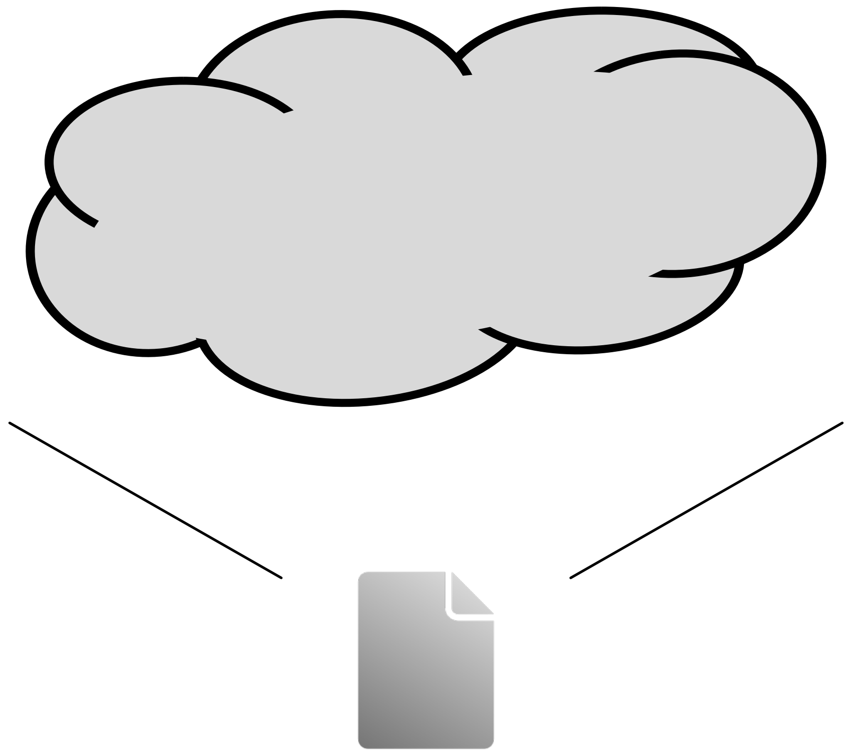
$$\begin{tikzcd}
\pi_A \arrow[d, no head] \\
\sigma_\theta \arrow[d, no head] \\
R
\end{tikzcd}$$

Finding records in a table really fast

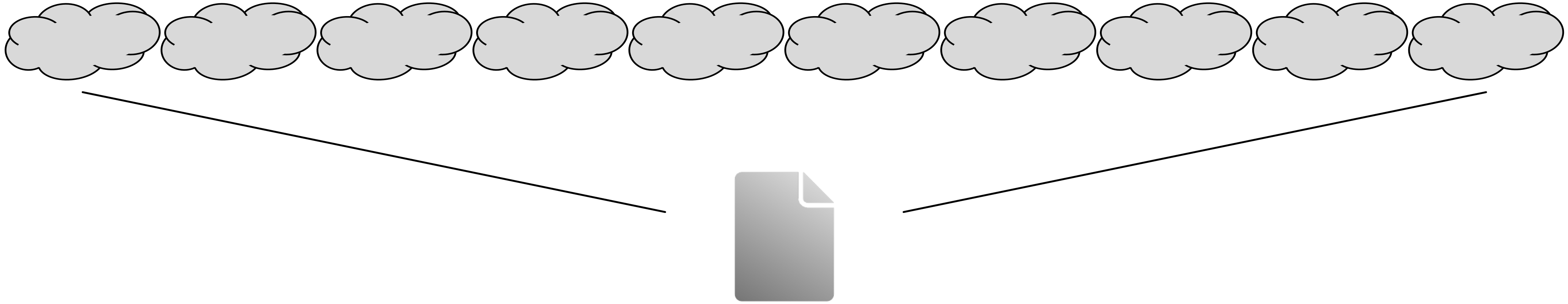
$$\begin{tikzcd}
\pi_A \arrow[d, dash] \\
\texttt{IndexScan(}R\texttt{, }\theta\texttt{)}
\end{tikzcd}$$

Data Layout

Data Layout



Data Layout



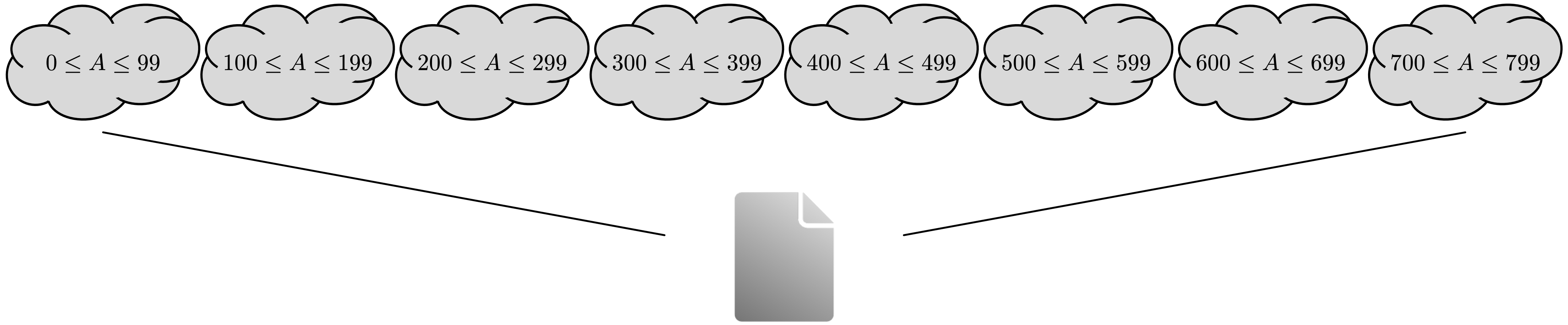
Data Layout Goals

- Point Lookups: $A = 3$ or $A = 3 \wedge B = \text{Apple}$
- Range Lookups: $A > 5$ or $\text{Jan } 23,\ 2026 < A < \text{Feb } 11,\ 2026$
- Multidimensional Range Lookups: $\text{Jan } 23,\ 2026 < A < \text{Feb } 11,\ 2026 \wedge B < \$50$
- Nearest Neighbor Lookups: “The K closest restaurants to (lat, lon)”

Data Layout Tricks

- Partition the data...
  - ... by ranges
  - ... by hash value
- Introduce signposts

Sorted Files



How do we know that page 3 holds records where $300 \leq A \leq 399$?

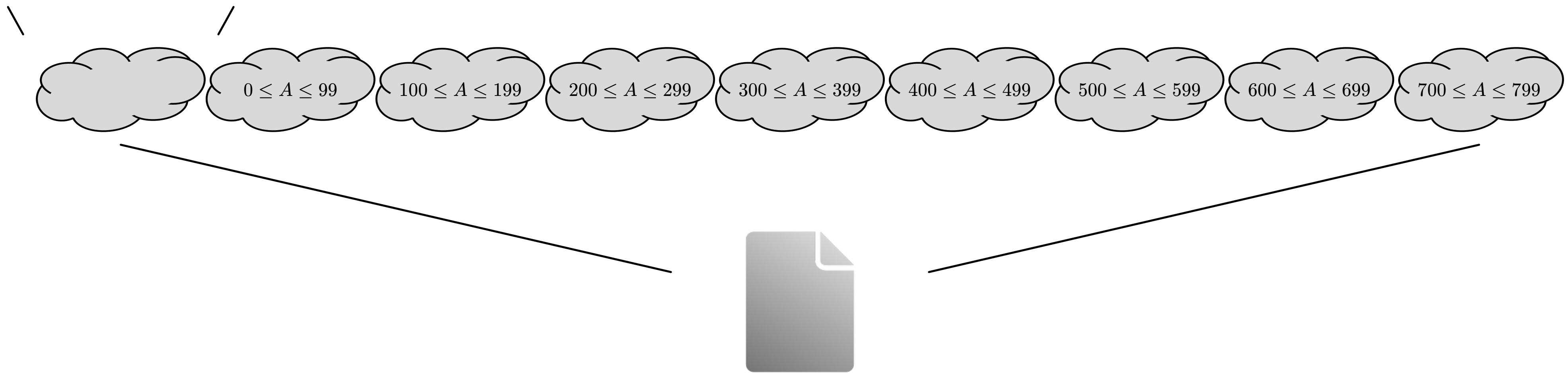
Fence Pointer Tables

Idea: Store a “separator” for each partition boundary.

(This is called a “Fence Pointer Table”)

Fence Pointer Tables

$-\infty$	p1
≥ 100	p2
≥ 200	p3
≥ 300	p4
≥ 400	p5
≥ 500	p6
≥ 600	p7
≥ 700	p8



Fence Pointer Tables

Idea: Store a “separator” for each partition boundary, with a signpost to where in the file the partition lives.

(This is called a “Fence Pointer Table”)

The good

- Easy to implement
- Efficient point reads ($O(1)$ IO)
- Efficient range scans ($O(1 + |\text{Result}|)$ IO)

The not-so-good

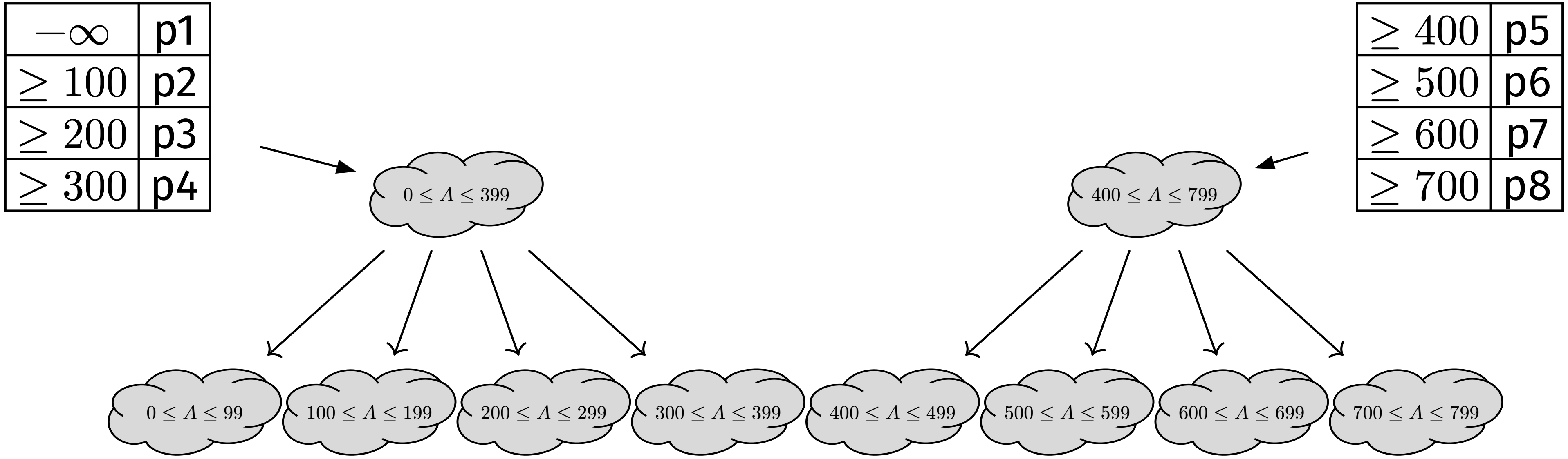
- Requires $O(N)$ memory to get the benefit
- Hard to update

ISAM Index

Idea: Store multiple layers of separators and signposts (each layer points to the next).

(This is called an ISAM Index)

ISAM Index

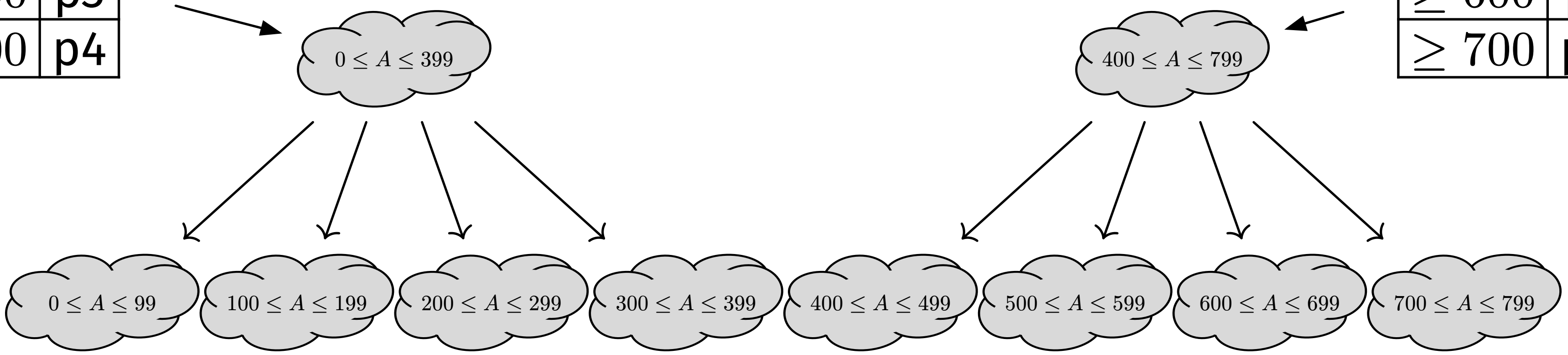


$-\infty$	p1
≥ 100	p2
≥ 200	p3
≥ 300	p4

≥ 400	p5
≥ 500	p6
≥ 600	p7
≥ 700	p8

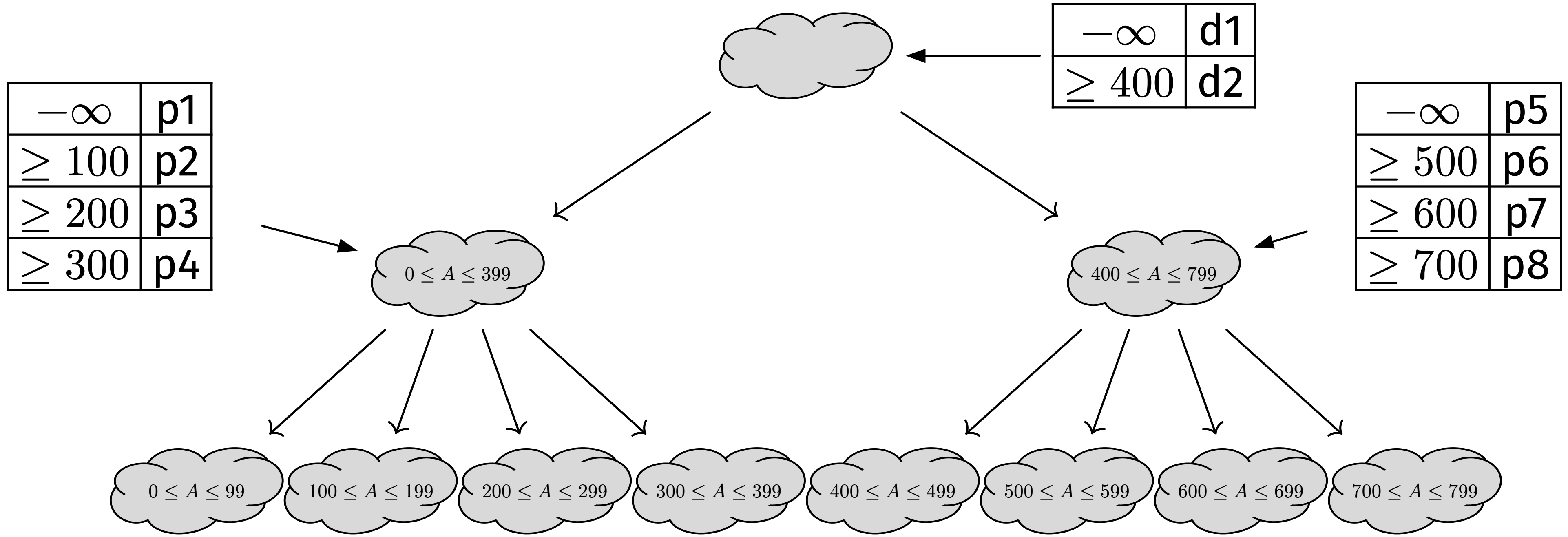
ISAM Index

$-\infty$	p1
≥ 100	p2
≥ 200	p3
≥ 300	p4



$-\infty$	p5
≥ 500	p6
≥ 600	p7
≥ 700	p8

ISAM Index



$-\infty$	d1
≥ 400	d2

$-\infty$	p1
≥ 100	p2
≥ 200	p3
≥ 300	p4

$-\infty$	p5
≥ 500	p6
≥ 600	p7
≥ 700	p8

ISAM Index

Idea: Store multiple layers of separators and signposts (each layer points to the next).

(This is called an ISAM Index)

The good

- Efficient point reads ($O(\log(N))$ IO)
  - Most of these will be cached
- Efficient range scans ($O(\log(N) + |\mathrm{Result}|)$ IO)
- $O(1)$ memory

The not-so-good

- Still not easy to update

B+ Tree

Problem: Contiguous data pages make it hard to insert new records in between.

B+ Tree

Problem: Contiguous data pages make it hard to insert new records in between.

Idea: Drop the contiguous page requirement.

B+ Tree

Problem: Contiguous data pages make it hard to insert new records in between.

Idea: Drop the contiguous page requirement.

Problem: … but contiguous pages make range scans efficient.

B+ Tree

Problem: Contiguous data pages make it hard to insert new records in between.

Idea: Drop the contiguous page requirement.

Problem: ... but contiguous pages make range scans efficient.

Idea: Organize leaf pages in a linked list.

B+ Tree

Problem: Contiguous data pages make it hard to insert new records in between.

Idea: Drop the contiguous page requirement.

Problem: ... but contiguous pages make range scans efficient.

Idea: Organize leaf pages in a linked list.

Problem: Can’t easily insert/remove records if every leaf page has to be full.

B+ Tree

Problem: Contiguous data pages make it hard to insert new records in between.

Idea: Drop the contiguous page requirement.

Problem: … but contiguous pages make range scans efficient.

Idea: Organize leaf pages in a linked list.

Problem: Can’t easily insert/remove records if every leaf page has to be full.

Idea: Don’t require leaf pages to be full?

B+ Tree

Problem: Contiguous data pages make it hard to insert new records in between.

Idea: Drop the contiguous page requirement.

Problem: ... but contiguous pages make range scans efficient.

Idea: Organize leaf pages in a linked list.

Problem: Can’t easily insert/remove records if every leaf page has to be full.

Idea: Don’t require leaf pages to be full?

Problem: Do we still have guaranteed log(N) IO?

B+ Tree

Problem: Contiguous data pages make it hard to insert new records in between.

Idea: Drop the contiguous page requirement.

Problem: ... but contiguous pages make range scans efficient.

Idea: Organize leaf pages in a linked list.

Problem: Can’t easily insert/remove records if every leaf page has to be full.

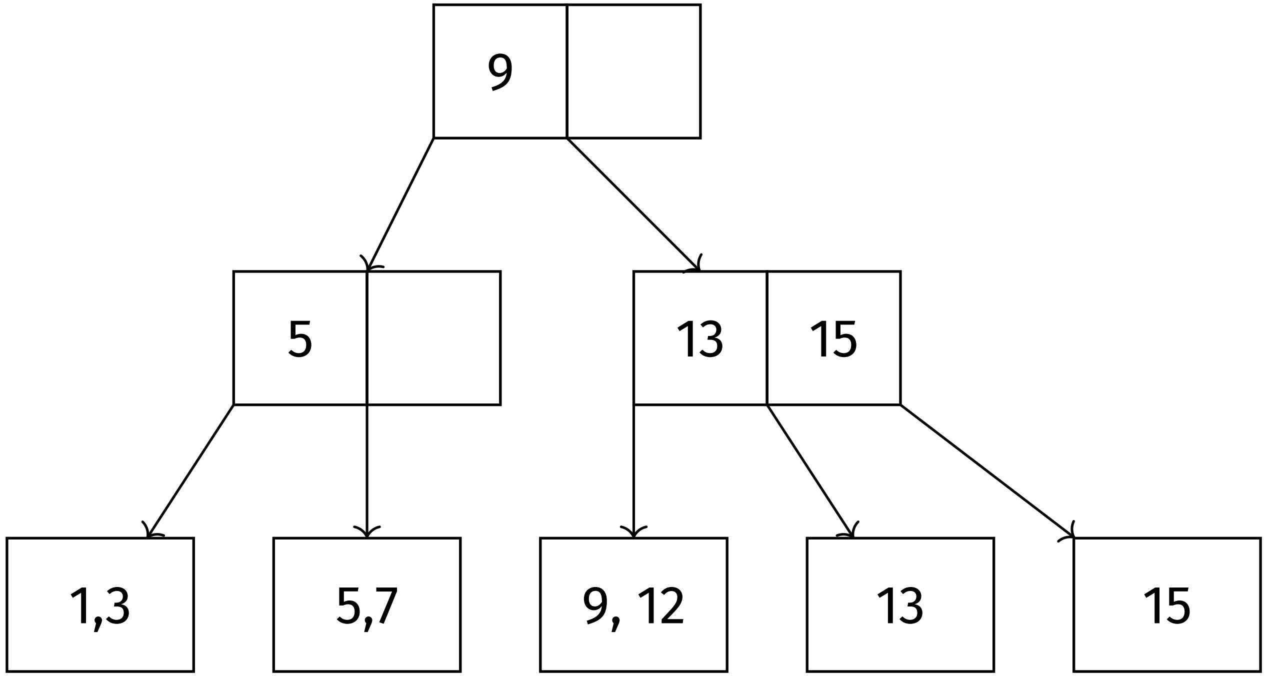
Idea: Don’t require leaf pages to be full?

Problem: Do we still have guaranteed log(N) IO?

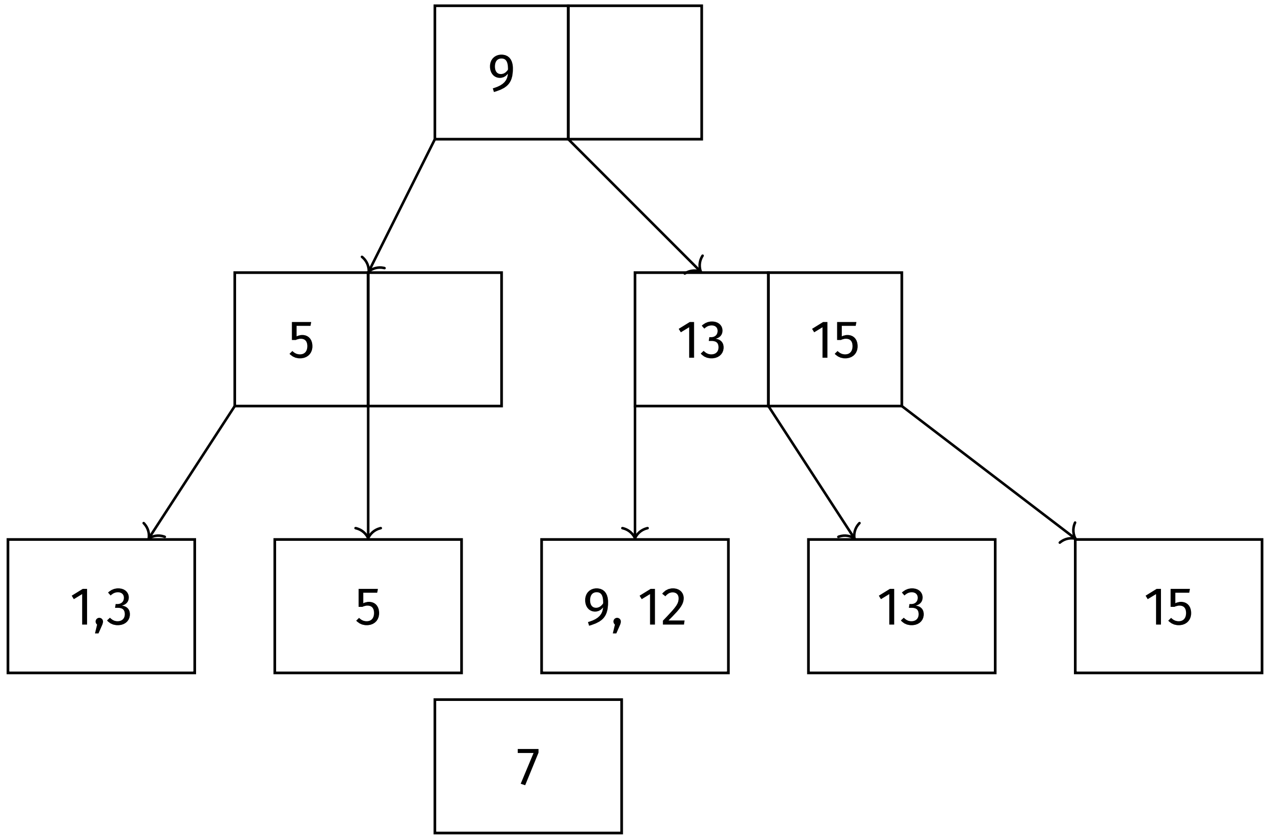
Idea: 50% “Minimum Fill”

Proof on Board

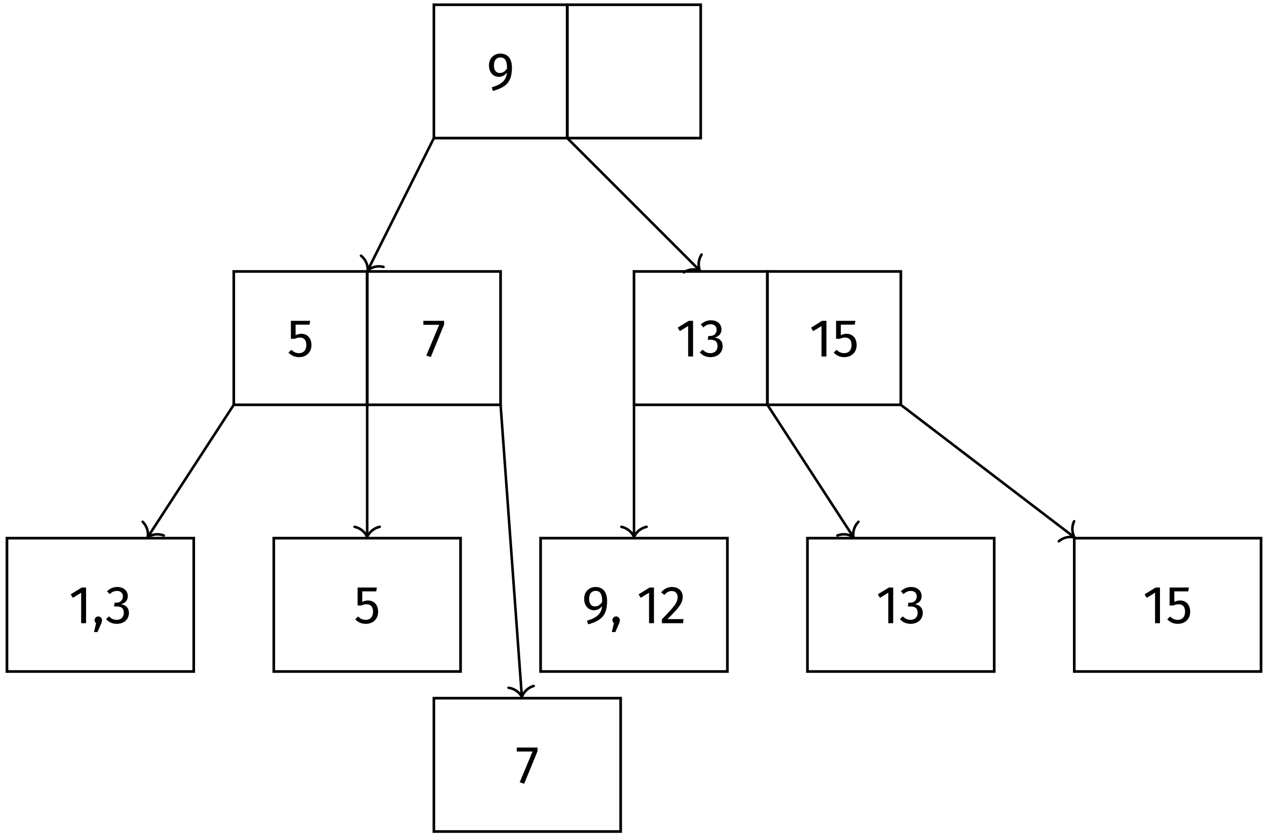
Merge/Split Example



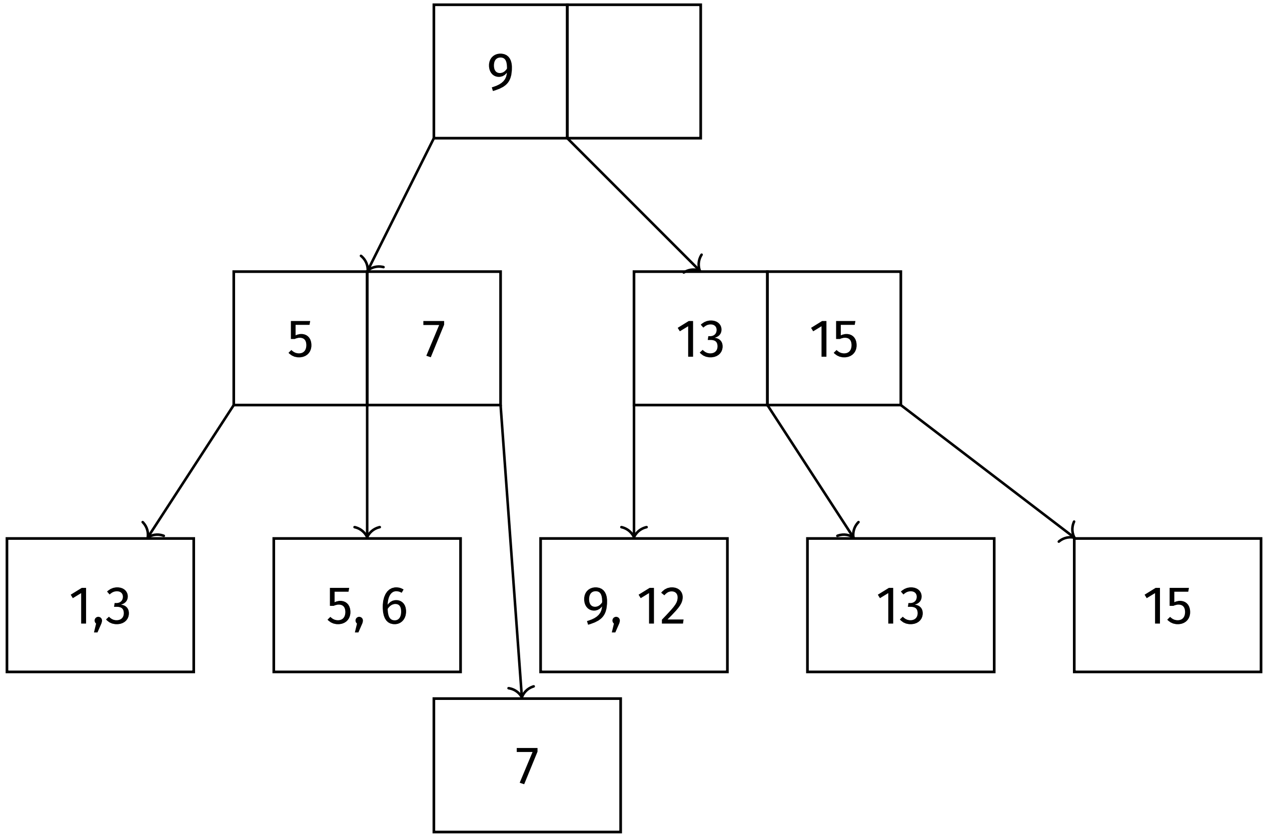
Merge/Split Example



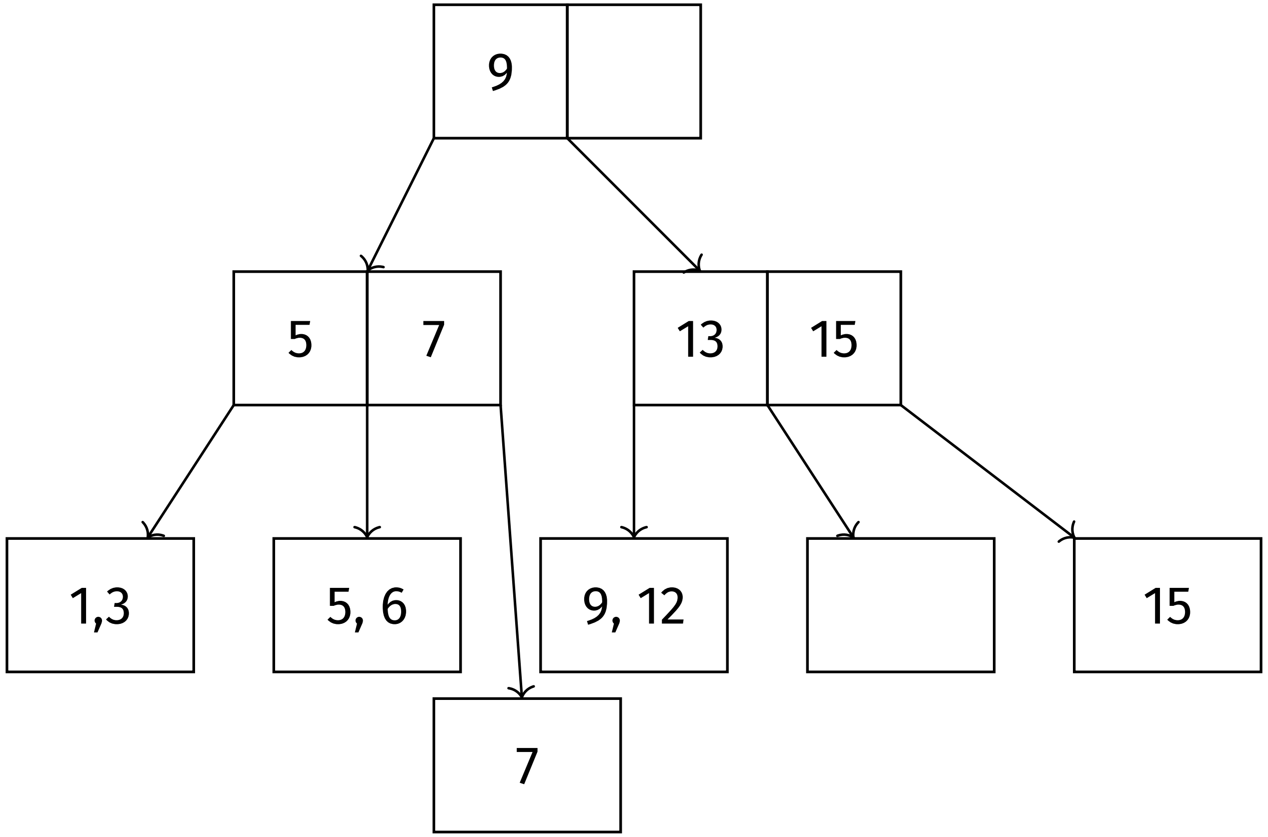
Merge/Split Example



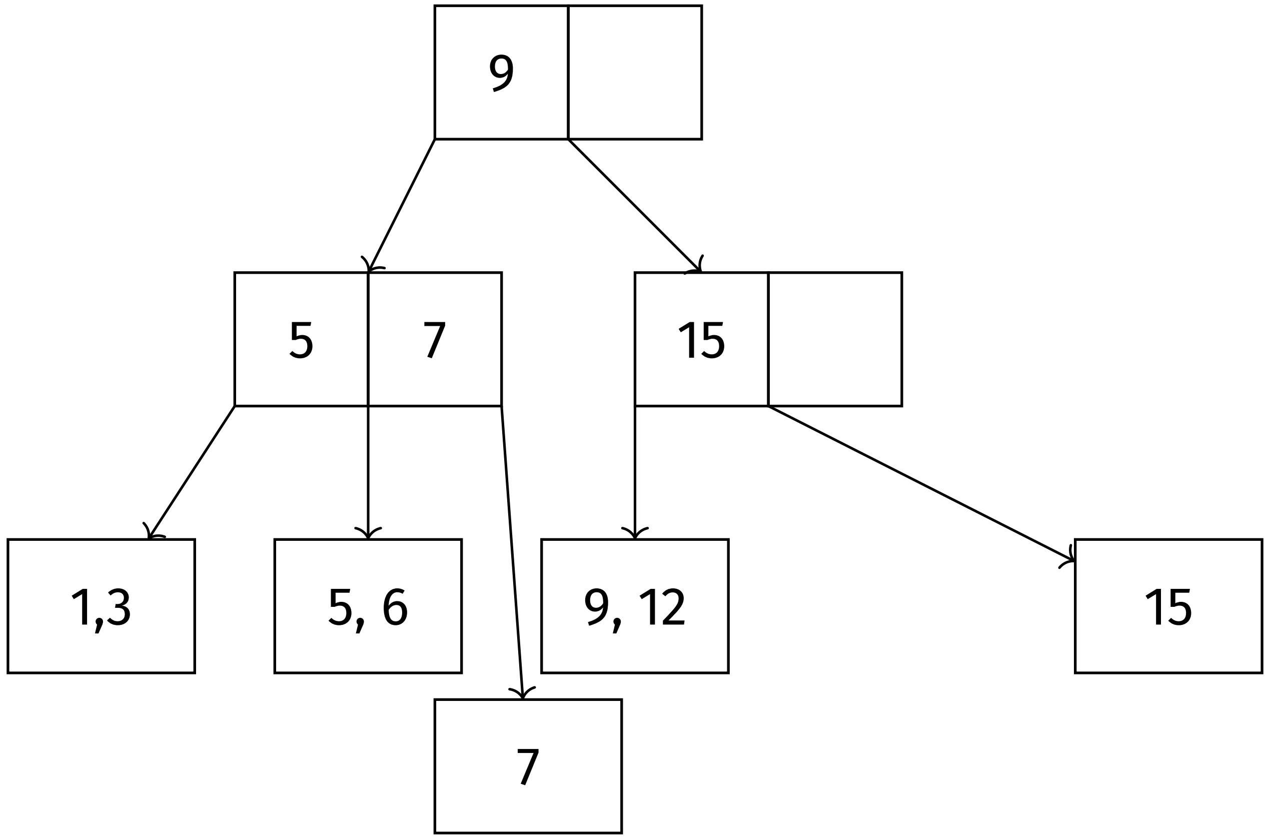
Merge/Split Example



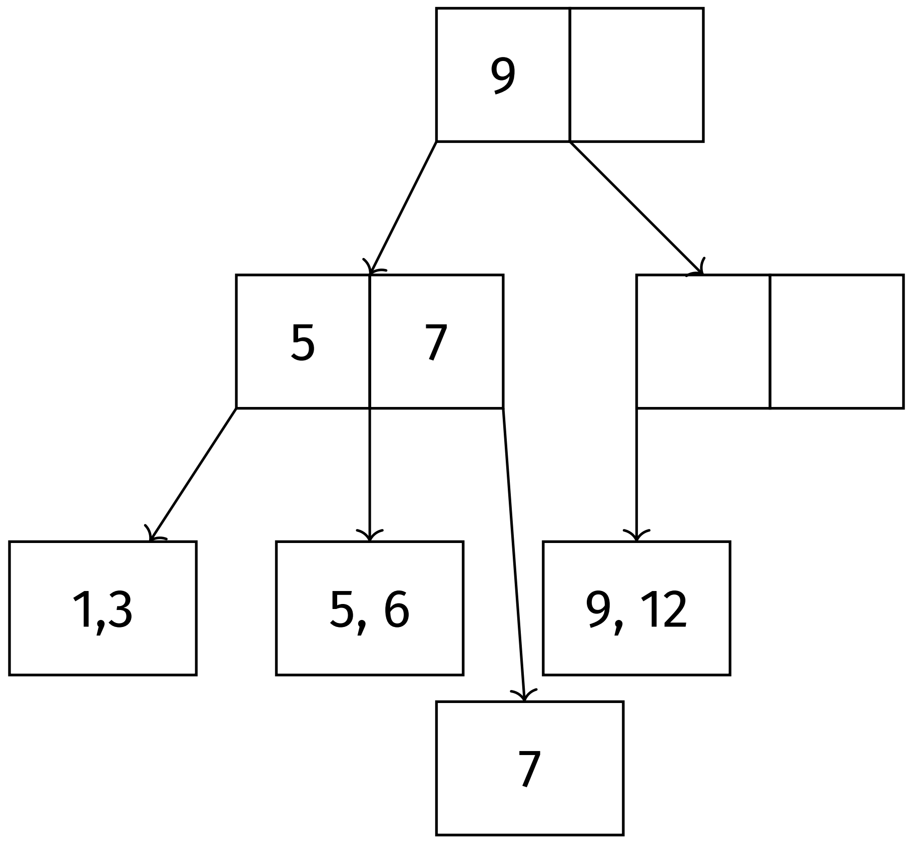
Merge/Split Example



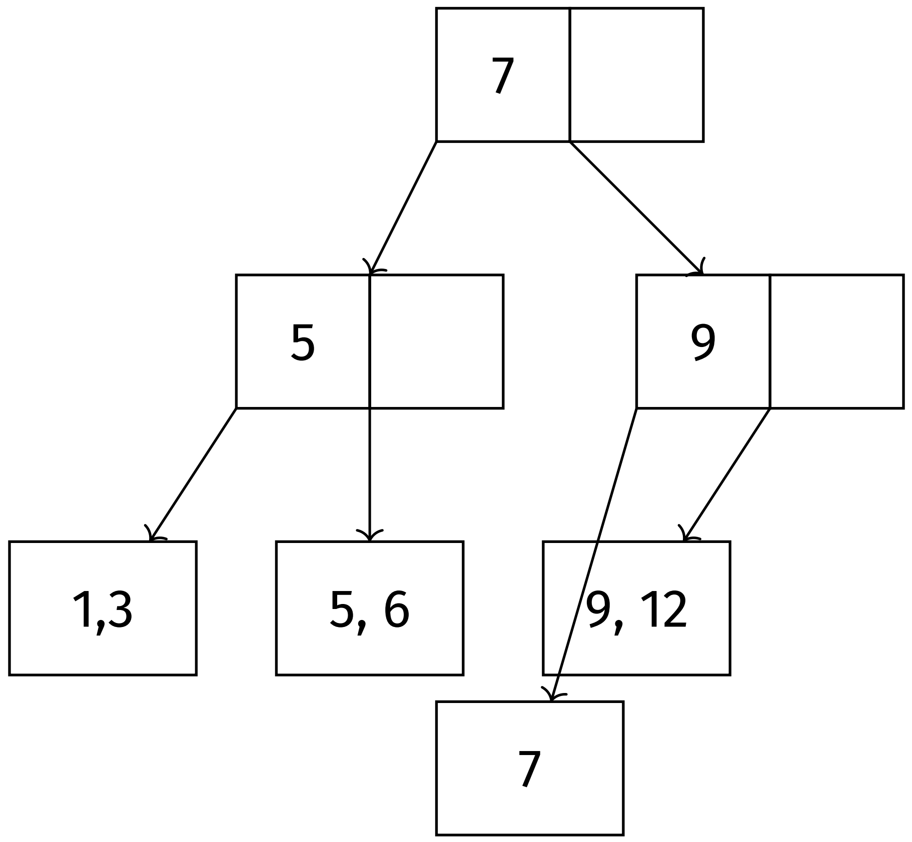
Merge/Split Example



Merge/Split Example



Merge/Split Example



B+ Insert

```
def insert(k, v):
  leaf = find_leaf(k)

  if not leaf.has_space(k, v):
    new_separator, new_leaf = leaf.split()
    leaf.parent.insert(new_separator, new_leaf)

  leaf.insert(k, v)
```

If split reaches root, add another layer.

B+ Delete

```
def delete(k):
  leaf = find_leaf(k)
  leaf.delete(k)

  if leaf.fill() < 50%:
    new_separator = leaf.steal_from_sibling()
    leaf.parent.update_separator(new_separator)

  if leaf.fill() < 50%:
    sibling = leaf.merge_with_sibling()
    leaf.parent.merge(sibling, leaf)
```

If root is empty, drop a layer

B+ Tree Invariants

Insert and delete preserve these two invariants:

1. Every node except the root is always at least half full.
2. Every leaf is at exactly the same depth

These invariants guarantee that the tree depth is $\log(N)$.

B+ Tree

The good

- Efficient point reads ($O(\log(N))$ IO)
  - Most of these will be cached
  - At most 2x the depth of an ISAM index
- Efficient range scans ($O(\log(N) + |\mathrm{Result}|)$ IO)
  - Note that, unlike ISAM, these are **random** reads
- $O(1)$ memory

The not-so-good

- Lots of random IO
- $\log(n)$ is still not small

Reminders

- Checkpoint 2 posted; Autolab up
- Checkpoint 1 solutions posted
- Quiz 1, 2 results posted