CSE 250: The Memory Hierarchy (contd.)

Lecture 37

Dec 04, 2023

Reminders

- WA3 due Sun, Dec 3
  - PA3 showed you that even 'anonymized' data can be problematic
  - WA3: Look for other cases of problems
- Course Evals Bonus
  - Get to 90% completion across all 3 sections, we'll release an exam question.
  - Section C: 17/112 as of Friday (15%)

Lies!

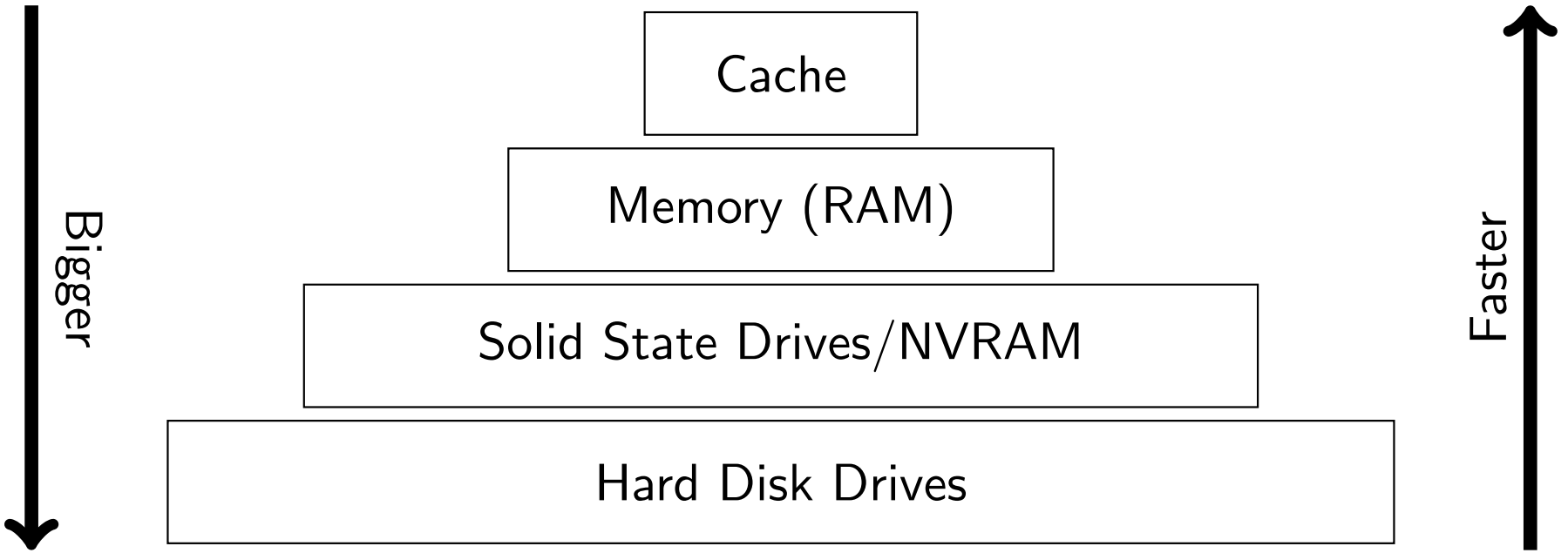
- **Lie 1:** Any array access is $O(1)$
  - This is the RAM model of computation
    - Simple, useful, but not perfect
  - Real-World Hardware isn't this elegant
    - The Memory Hierarchy: L1 Cache, L2 Cache, L3 Cache, RAM, SSD, HDD, Tape...
    - Non-Uniform Memory Access CPUs: AMD Ryzen
- **Lie 2:** The constant factors don't matter

These are useful simplifications at 50k ft, but they don't tell the whole story.

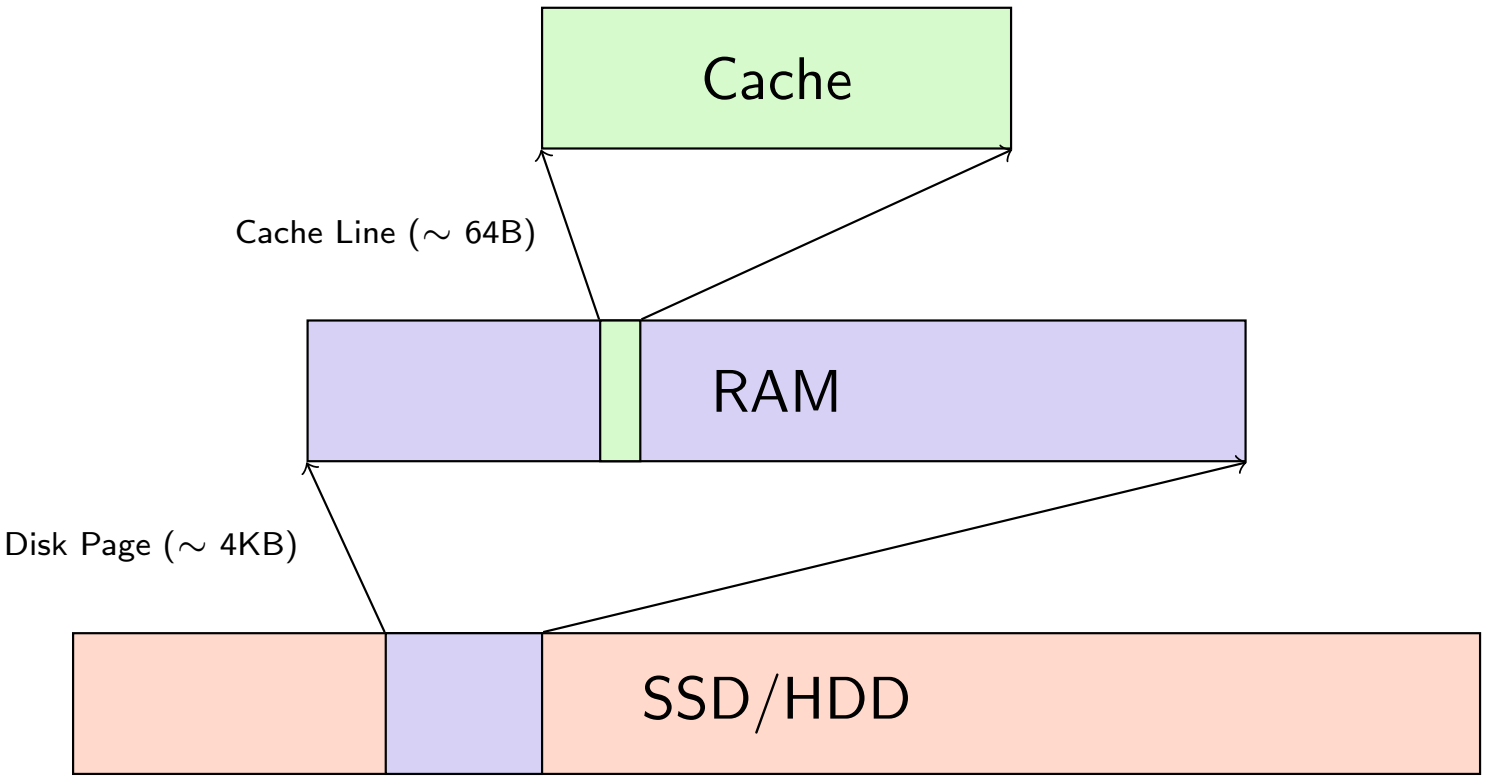
Algorithm Bounds

- Runtime Complexity
  - The algorithm takes $O(\ldots)$ steps/cpu cycles/time
- Memory Complexity
  - The algorithm needs $O(\ldots)$ MB of RAM
- IO Complexity
  - The algorithm performs $O(\ldots)$ accesses to slower memory.
  - Sometimes separately tracks reads and writes.
  - Sometimes considers > 2 memory speeds.

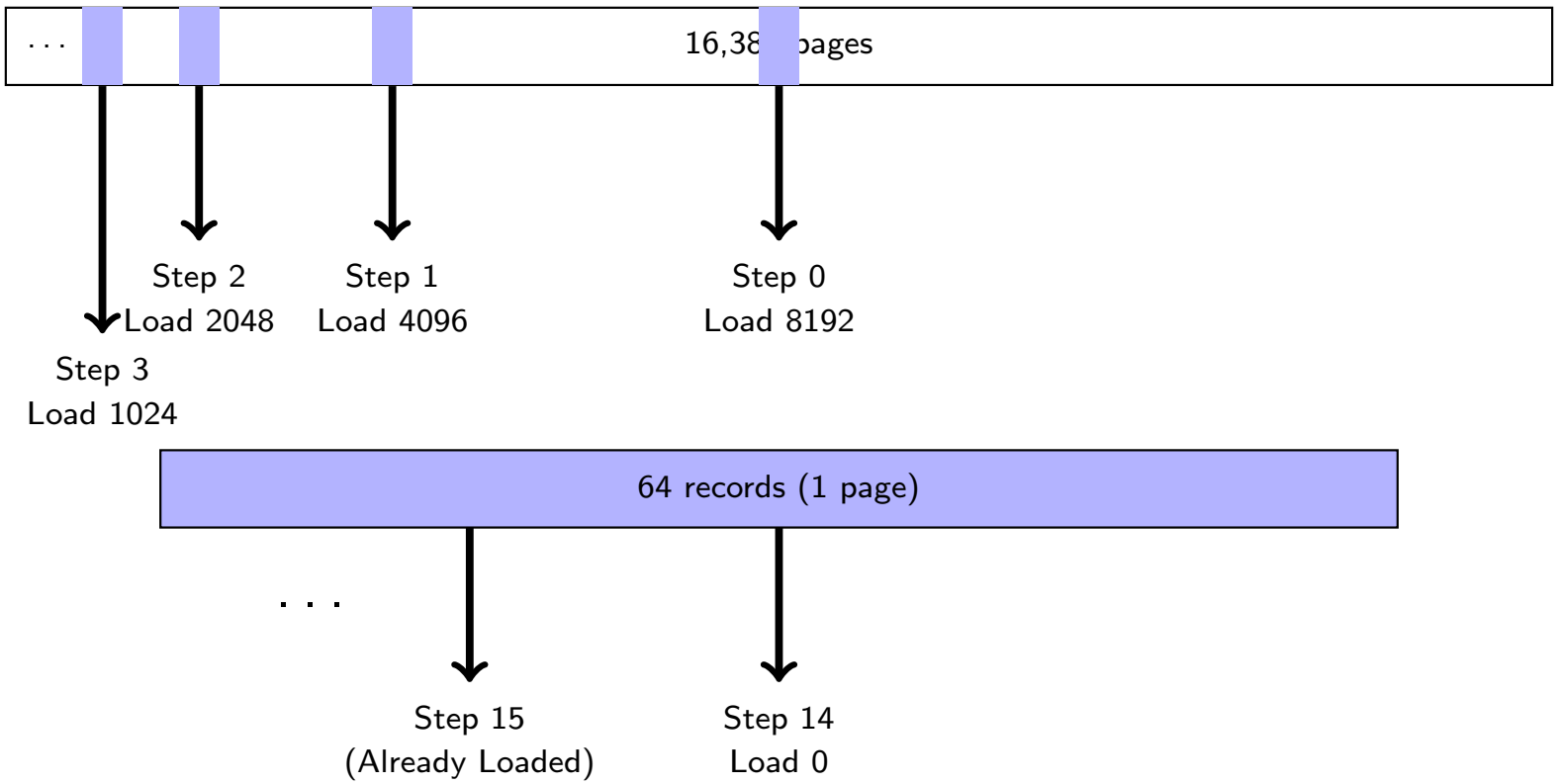
The Memory Hierarchy (simplified)



The Memory Hierarchy (simplified)



Binary Search



Binary Search

- Steps 0-14 each load 1 page (15 pages loaded)
  - Slooooooooooow...
- Steps 15-19 access the same page as step 14
  - Fast!

- Runtime Complexity: $O(\log N)$
- Memory Complexity: $O(1)$
- IO Complexity: $O(\log N)$

How do we improve Binary Search?

- **Observation 1:** Keys are usually much smaller than full records.
- **Observation 2:** We always need to load the page with the record we want on it, so our main goal should be to reduce the number of IOs needed to find page.

Idea: Track the greatest key from each page.

How do we improve Binary Search?

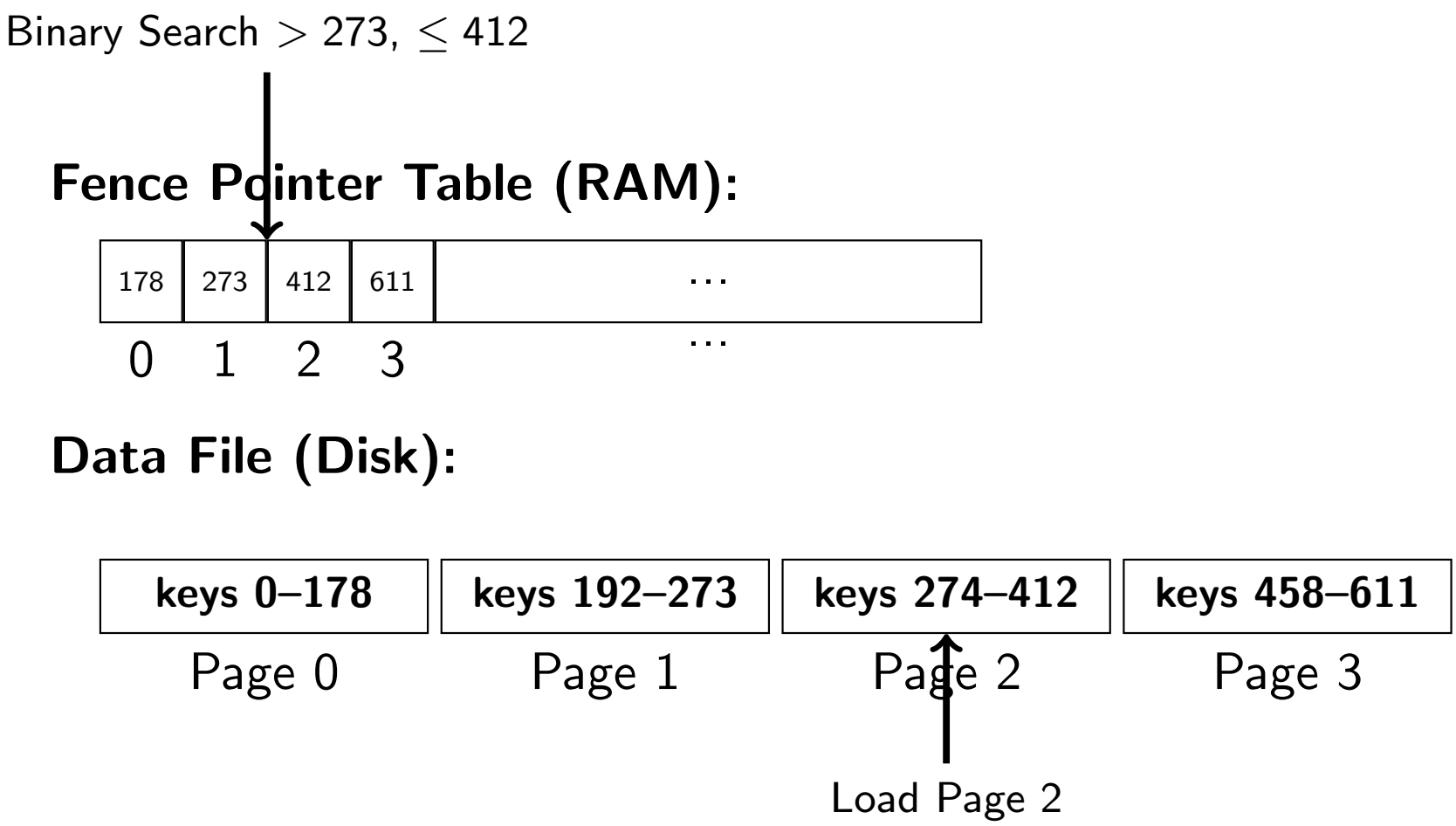
There should be exactly one page where:

- The key we're looking for is lesser than or equal to the greatest key on the page
- The key we're looking for is greater than the greatest key on the page.

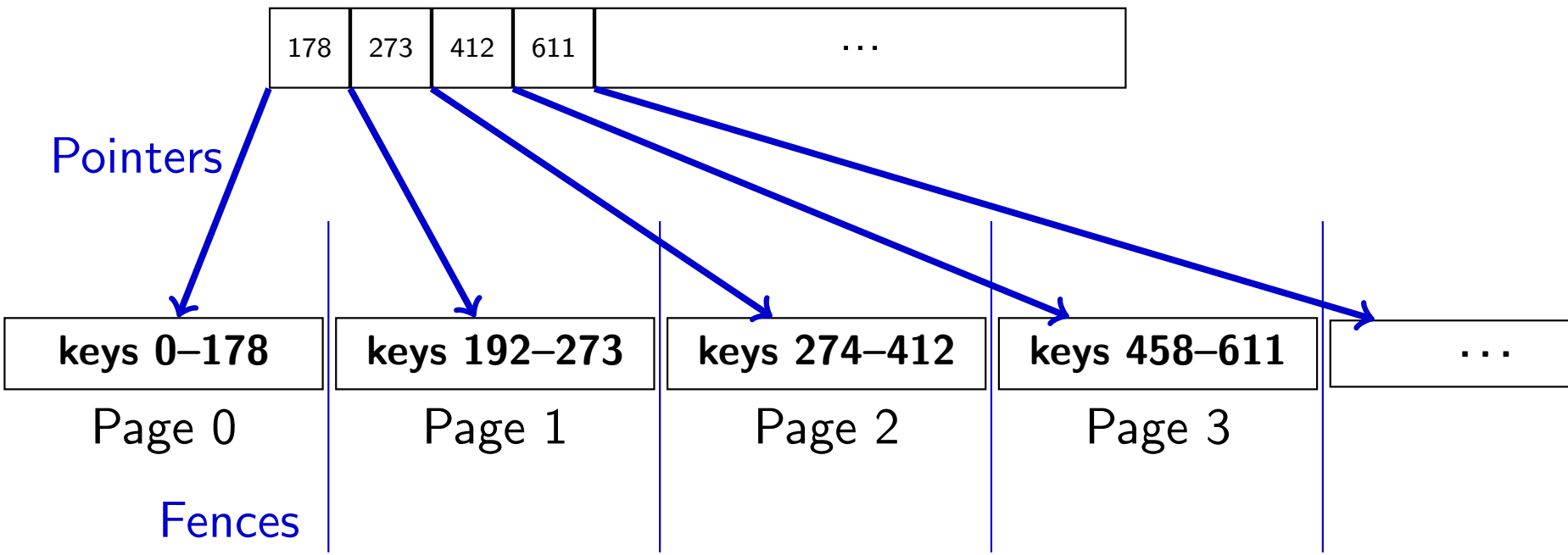
Once we find this page, we only need to do one IO.

Idea: Keep the list of greatest keys in memory always.

Fence Pointer Table



Why "Fence Pointer"?



Fence Pointers

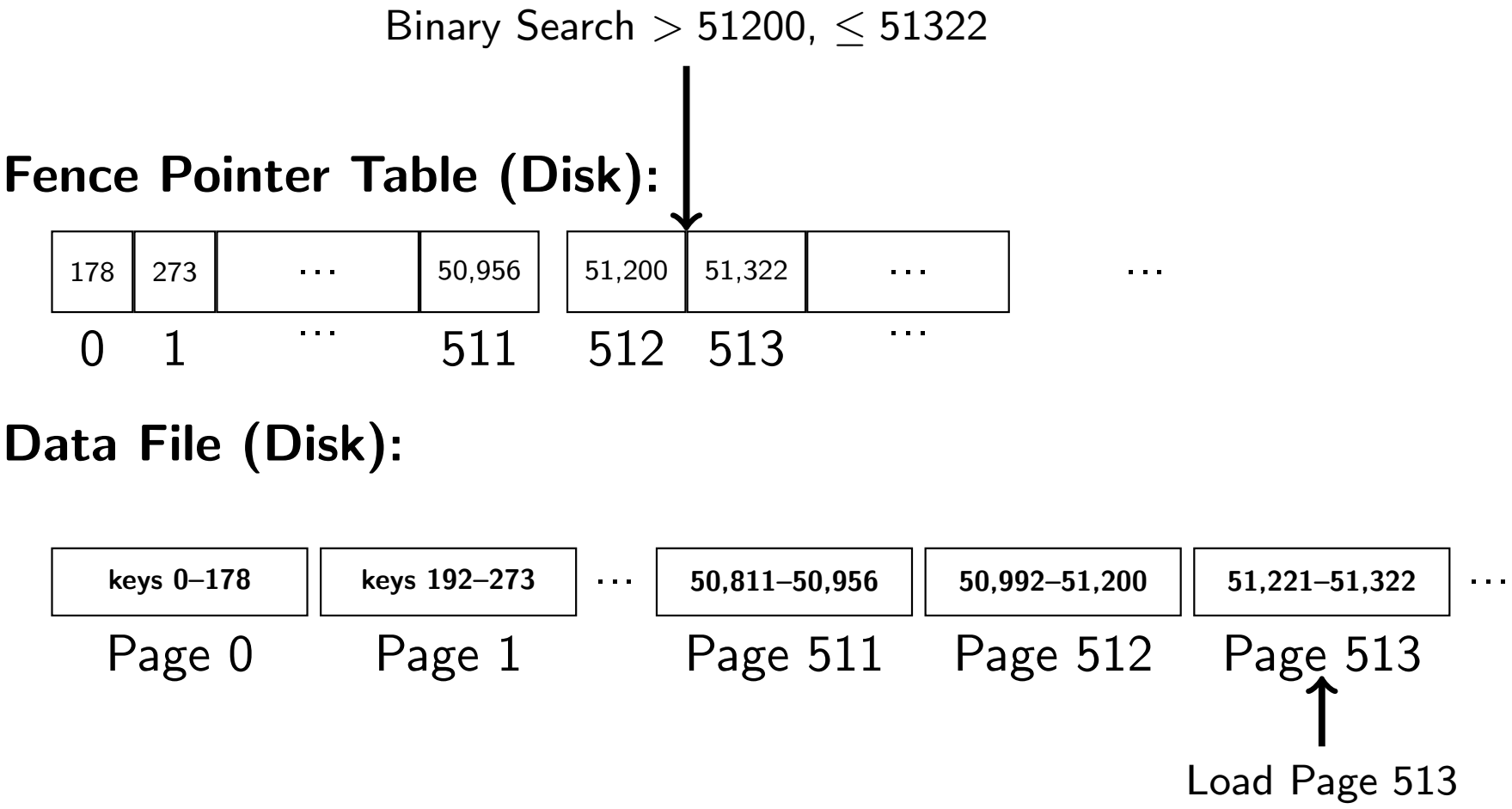
- Runtime Complexity: $O(\log N)$
- Memory Complexity: $O(N)$
- IO Complexity: $1 = O(1)$

Fence Pointers

We can do better...

Idea: Store the fence pointer table on disk.

Fence Pointer Table



Fence Pointers (On Disk)

- 512×8 byte keys per 4KB page (2^9 keys per page)
- 2^{20} records at 64 records per page $= 2^{14}$ pages of records
- 2^{14} fence pointer keys $= 2^5$ pages of fence pointers
- Total pages searched:
  - Fence Pointer Table: $\log_2(2^5) = 5$ pages
  - Data File: 1 page
  - **Total:** 6 pages (vs 15 pages for simple binary search).

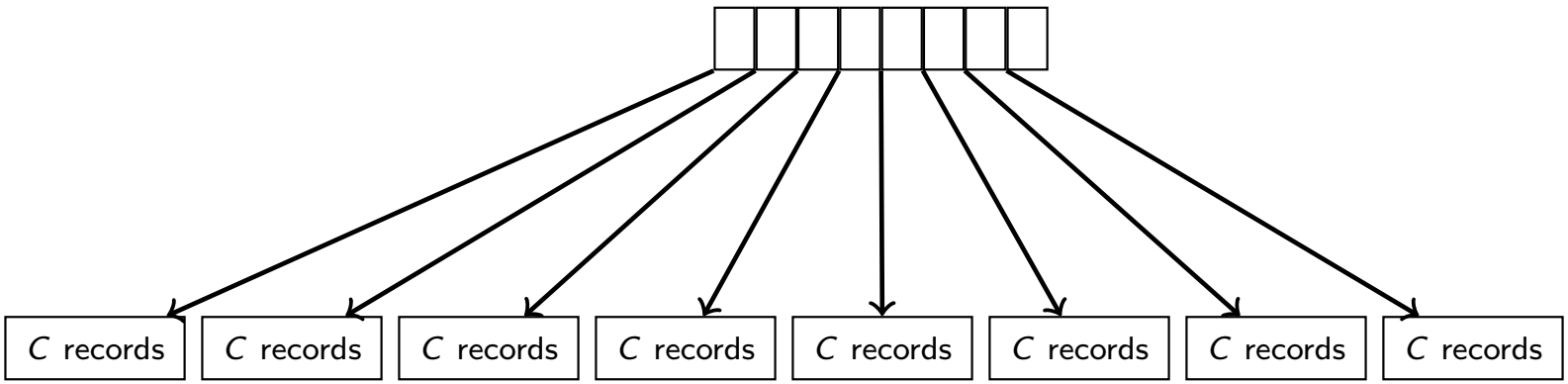
Fence Pointers (On Disk)

- Runtime Complexity: $O(\log N)$
- Memory Complexity: $O(1)$
- IO Complexity: $O(\log N)$

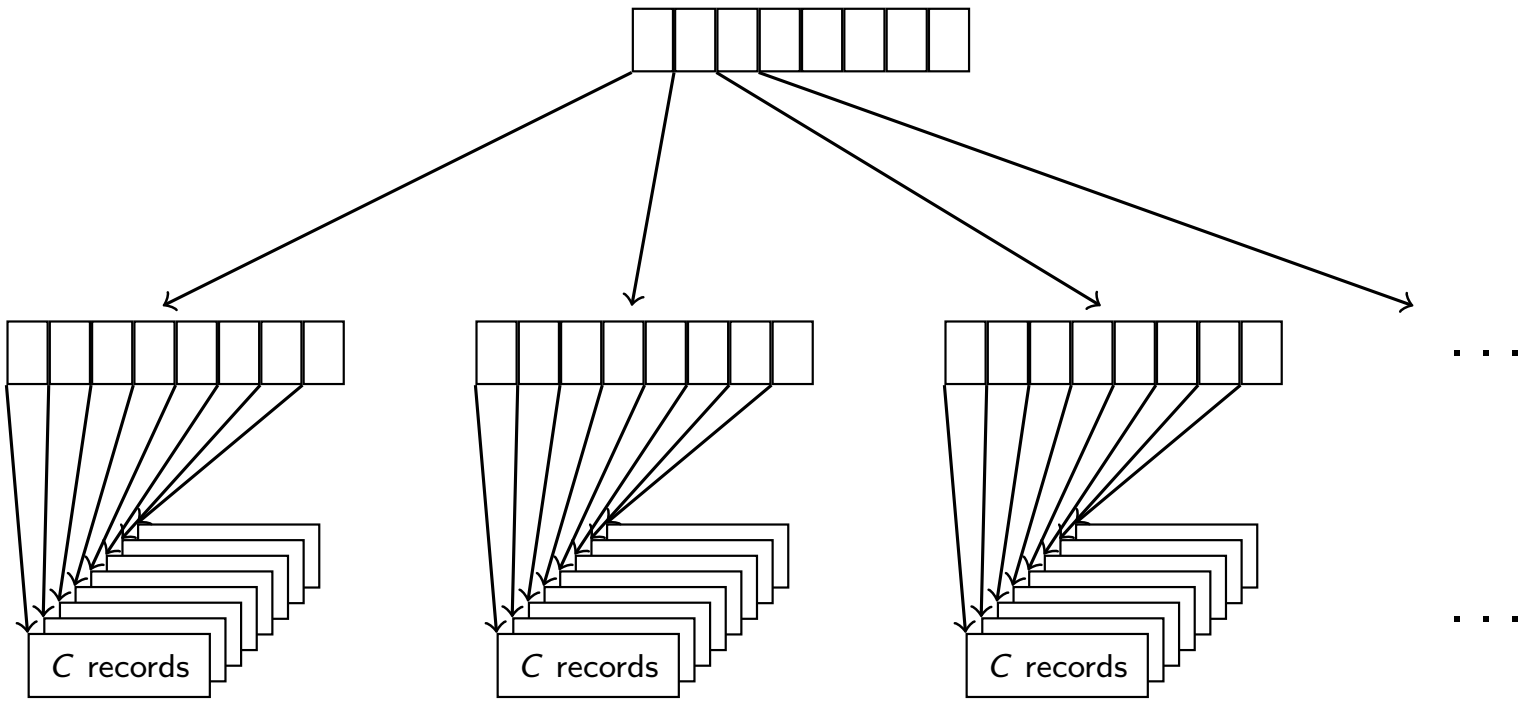
  ... but with a much better constant

Searching the on-disk fence pointer table is better than searching the raw data, but there's still lots of wasted IO.

Improving on Fence Pointers



Improving on Fence Pointers



ISAM Indexes

Repeat layering as needed.

This is called an **ISAM index**.

ISAM Indexes

- Read and Binary Search the level 0 index page to get a level 1 index page.
- Read and Binary Search the level 1 index page to get a level 2 index page.
- ...
- Read and Binary Search the level ℓ index page to get a data page.
- Read and Binary Search the data page to get the record.

If the tree is $\ell + 1$ levels deep, $\ell + 1$ IOs will be required.
At any given time, only one page is required: $O(1)$ memory.

ISAM Index

How many levels are required?

If we can fit K keys into an index page, then ...

- The level 1 index page cuts the search space down to: $\frac{1}{K^1}N$
- The level 2 index page cuts the search space down to: $\frac{1}{K^2}N$
- The level 3 index page cuts the search space down to: $\frac{1}{K^3}N$
- ...
- The data page can't hold more than C records.

If we have ℓ index levels, we can hold $N = K^\ell C$ records:

ISAM Index

If we have ℓ index levels, we can hold $K^\ell C$ records:

$$N = K^\ell C$$

$$\frac{N}{C} = K^\ell$$

$$\log_K \left(\frac{N}{C}\right) = \log_K (K^\ell)$$

$$\log_K \left(\frac{N}{C}\right) = \ell$$

We need $\log K(\frac{N}{C}) = O(\log_K N)$ index levels.

ISAM Index

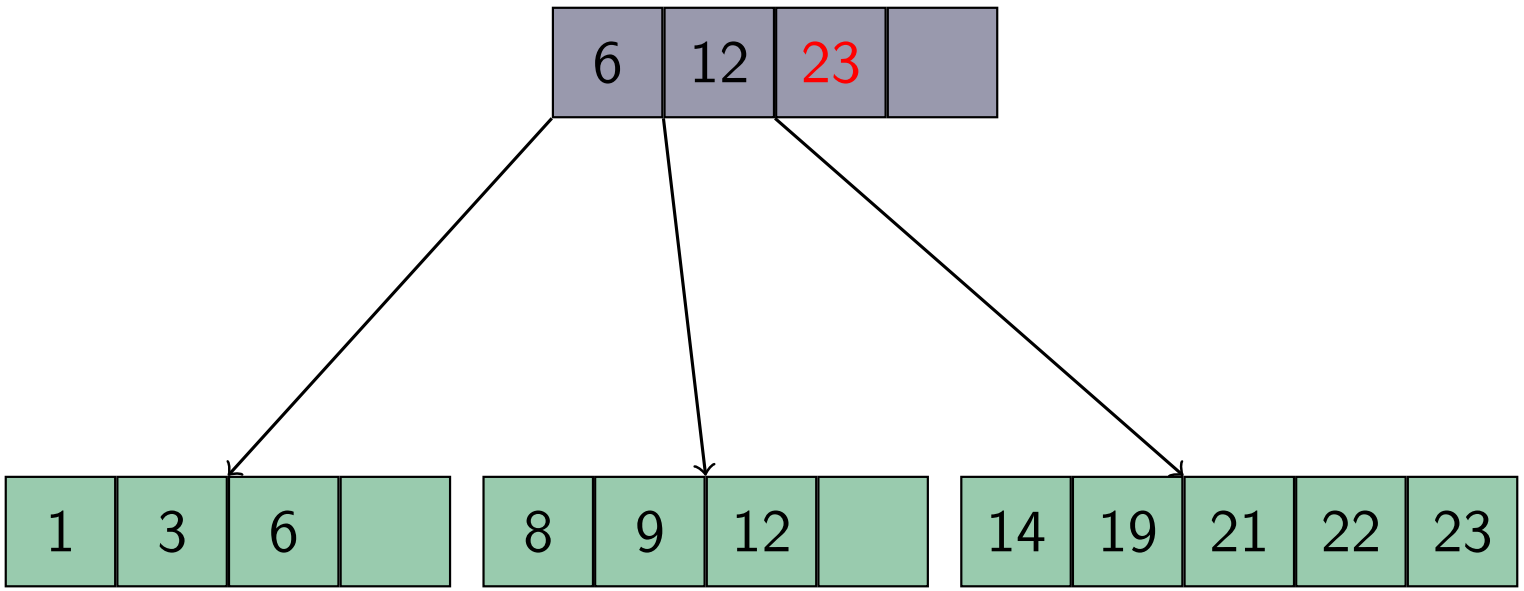
- Runtime Complexity: $O(\log N)$
- Memory Complexity: $O(1)$
- IO Complexity: $O(\log_K N)$
  - Contrast with $O(\log_2 N)$ for naive on-disk binary search.

But...

What if you need to insert a new record?

Idea: Reserve some extra space on each data page.

Extra Space



Add 6

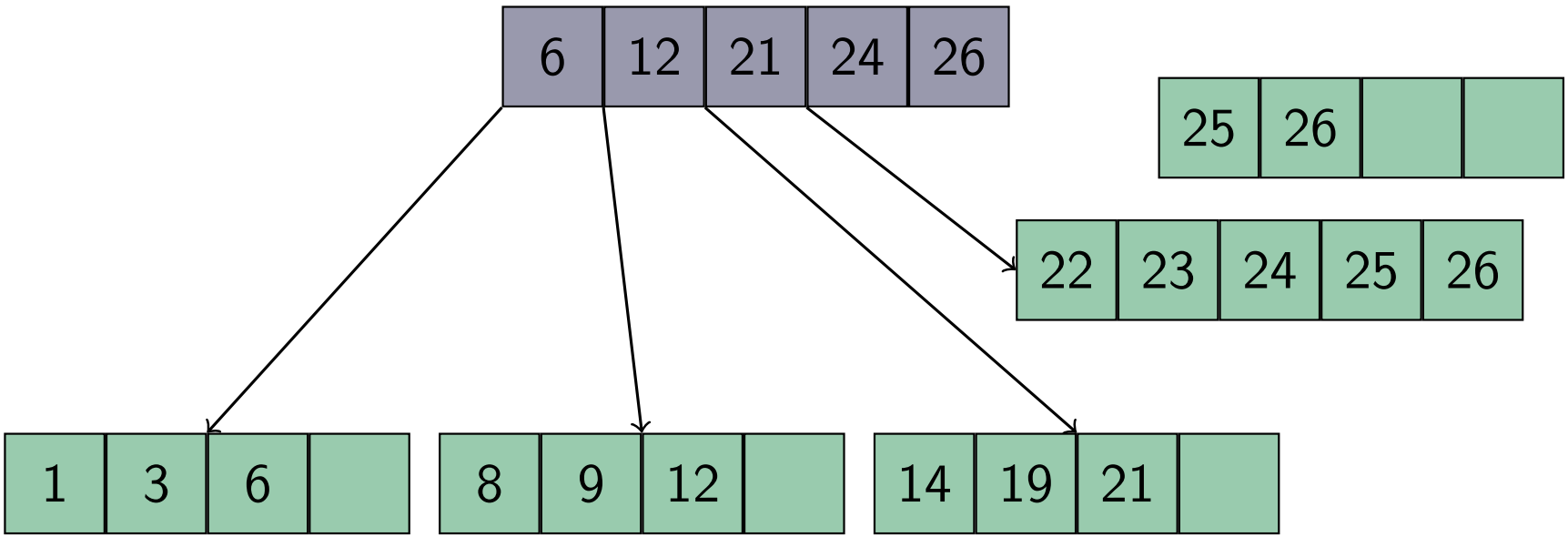
Add 9

Add 14

Add 23

Add 22?

Extra Space

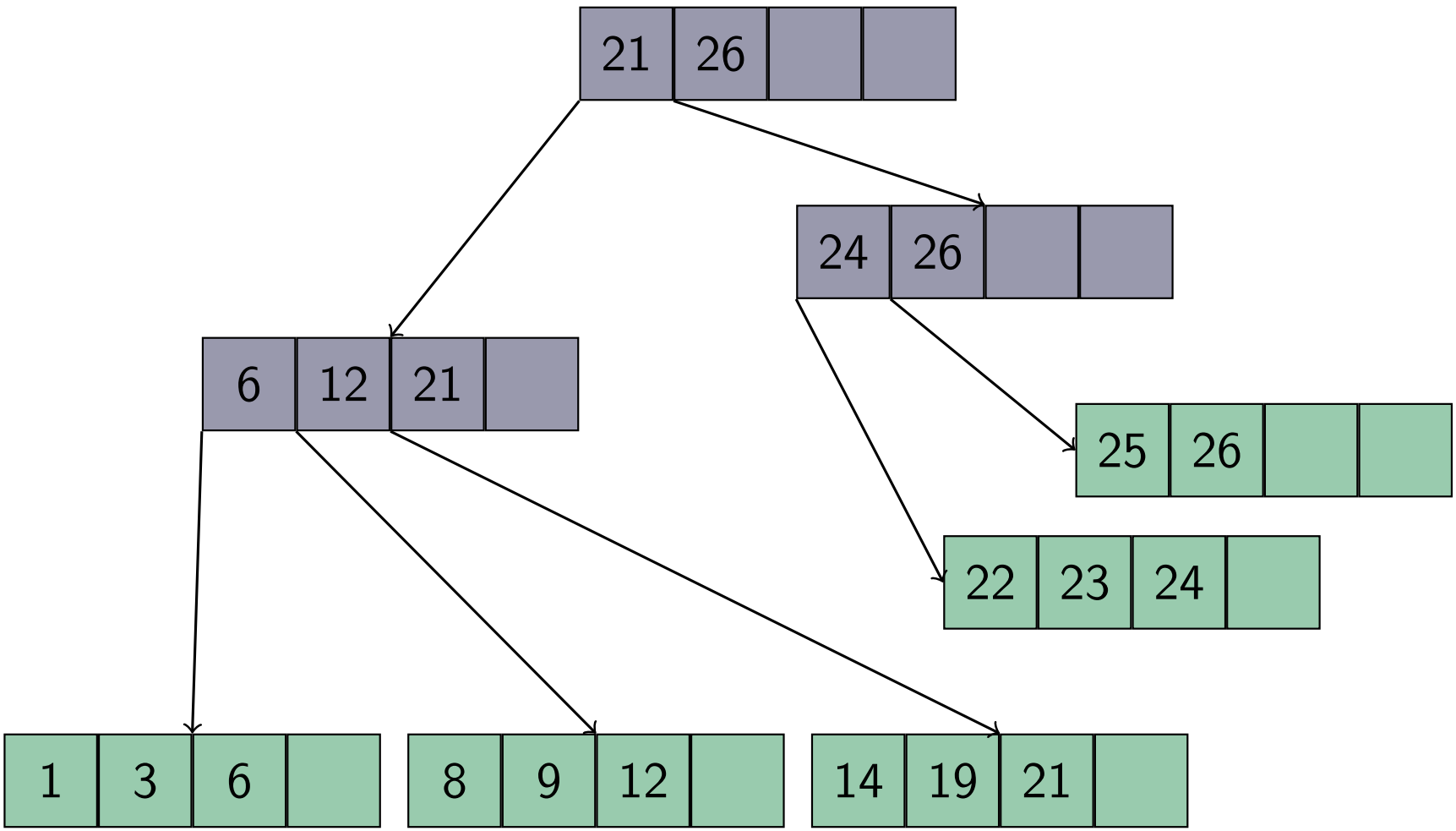


Add 24

Add 25

Add 26

Extra Space



B+ Trees

Resizing a tree like this guarantees balance.

- Only 'root splits' ever change the depth.
- The entire tree has a consistent depth.
- ... but no more than $\log_K N$ 'page splits' per insert.

B+ Tree Deletion

... but what if we allow deletion?

Any data page could end up empty!

Idea: Require every data page be at least 50% full.

B+ Tree Deletion

Idea: Require every data page be at least 50% full.

- If a page drops to $< 50\%$, steal a record from an adjacent page.
- If no adjacent pages to steal from, merge with an adjacent page.
  - ... which deletes an entry from the parent

B+ Tree Invariants

Every data/index node except the root is at least 50% full.

- **Insert**
  - Find the insertion point.
  - Insert the record.
  - 'Split' if needed.
  - Recursively split at ancestors as needed.
- **Delete**
  - Find the record.
  - Delete the record.
  - Steal/merge if needed.
  - Recursively steal/merge at ancestors as needed.