Don’t need to read from disk
can read from book instead

# CSE 250
# Lecture 36

## ISAM Indexes

# Binary Search: Complexity

- IO Complexity:
  - **Stage 1**:
    - Each step does one load: O(log(n) - log(C)) = O(log(n))
  - **Stage 2**:
    - Exactly one load for the entire step: O(1)
  - Total IO is the sum of the IOs of the component steps

**IO Complexity scales as $\log_2(n)$**

# How do we improve Binary Search?

- **Trivial Solution:**
    - Preload the entire array into memory upfront
        - Load once, re-use for all subsequent searches
        - **Problem**: Works at 64MB, maybe not at 2TB
- **Question**: Do we need to preload the entire array?

# How do we improve Binary Search?

- **Observation 1**:
  - 64 MB of $2^{20}$ x sizeof(key + data)

    vs
  - $2^{20}$ x 8B = 8 MB of keys
- **Observation 2**:
  - We don’t need to know which array index the record is at
    - ... only the page it’s on
    - ... and each page stores a contiguous range of keys

# Fence Pointers

- **Idea**: In-memory data structure with enough information to identify which page a record is on.
    - Precompute the (ideally smaller) data structure
    - Re-use the in-memory data structure for all searches

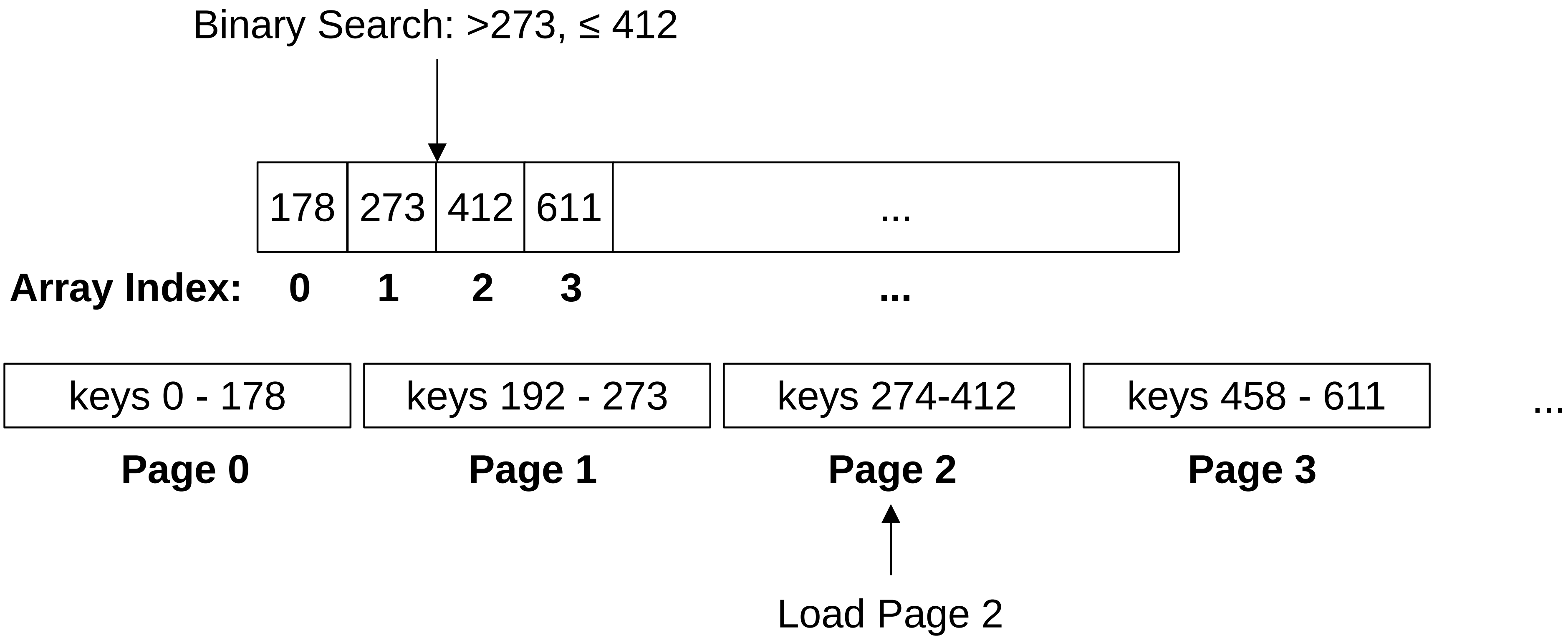
# Fence Pointers

- Precompute the greatest key in each page in memory
  - n records; 64 records/page; $^{n}/_{64}$ keys
  - e.g., $n=2^{20}$ records; Needs $2^{14}$ keys
    - $2^{20}$ 64 byte records = 64 MB
    - $2^{14}$ 8 byte records = $2^{19}$ bytes = 512 ***K***B
  - Call this a “Fence Pointer Table”

**RAM:** $2^{14}$ = 16,384 keys (Fence Pointer Table)

**Disk:** 16,384 pages (Actual Data)

# Example



Binary Search: >273, ≤ 412

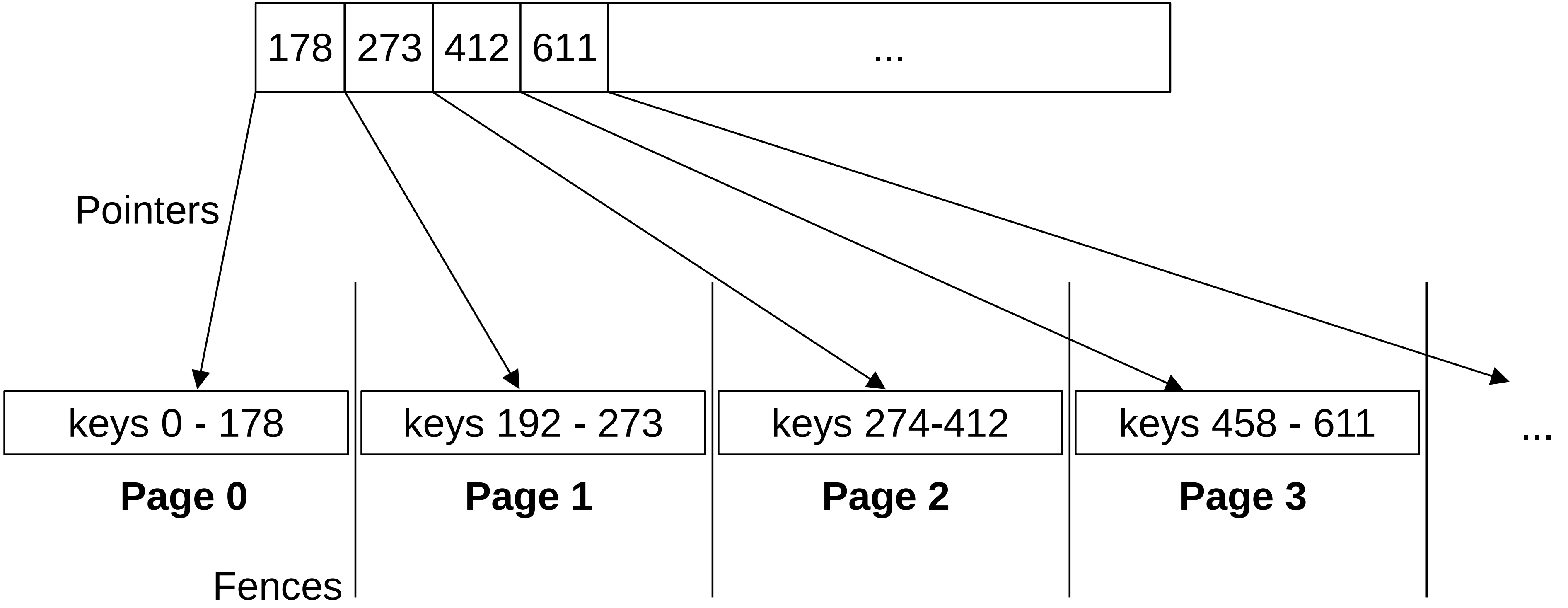
| 178 | 273 | 412 | 611 | ... |
|---|---|---|---|---|

**Array Index:** **0** **1** **2** **3** **...**

| keys 0 - 178 | keys 192 - 273 | keys 274-412 | keys 458 - 611 | ... |
|---|---|---|---|---|
| **Page 0** | **Page 1** | **Page 2** | **Page 3** | |

Load Page 2

# Example (Why “fence pointer”?)



# Fence Pointers

- **Step 1**: Binary Search on the Fence Pointer Table
  - All in-memory (IO complexity = 0)
- **Step 2**: Load page
  - One load (IO complexity = 1)
- **Step 3**: Binary search within page
  - All in-memory (IO complexity = 0)
- Total IO Complexity: O(1)

# Fence Pointers

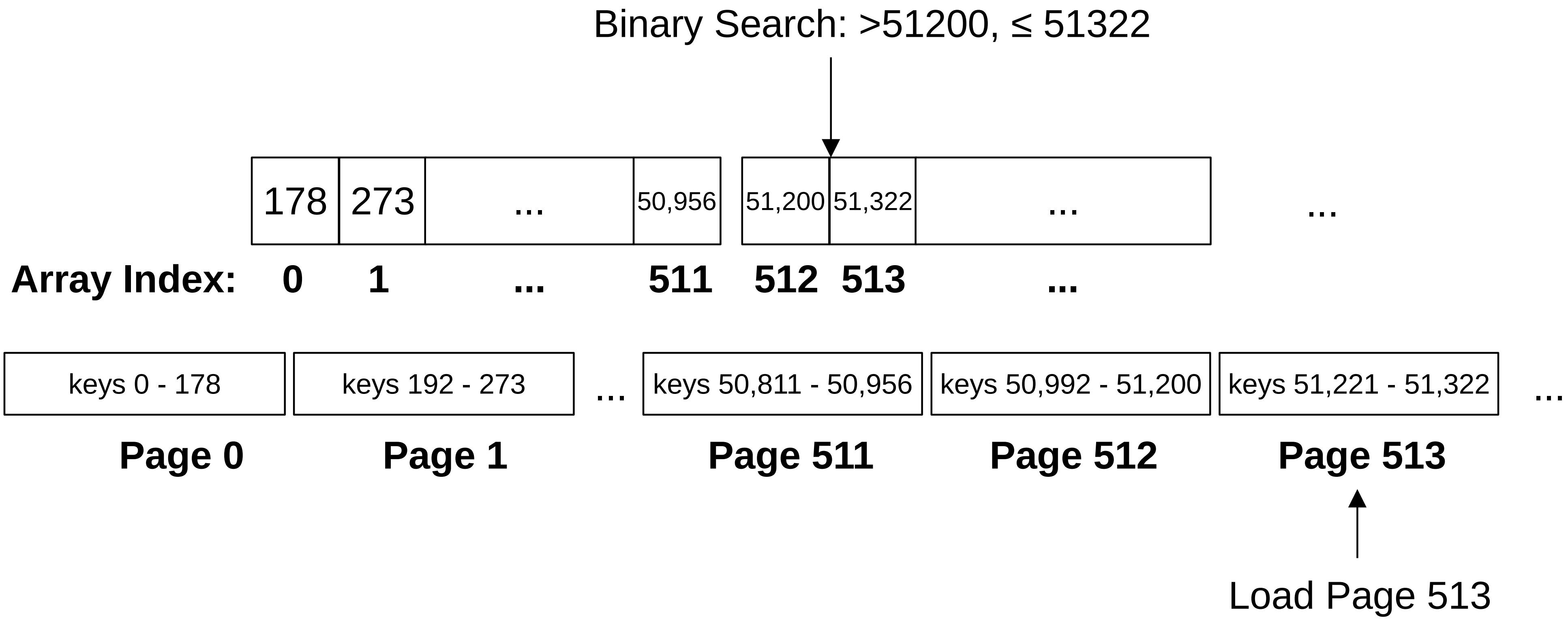
- Memory Complexity:
  - Need the entire fence pointer table in memory **at all times**
    - O(n / C) pages = O(n)
  - Steps 2, 3 load one more page
  - **Total**: O(n+1) = O(n)

**O(n) is... not ideal**

# Improving on Fence Pointers

- Store the Fence Pointers on Disk
    - 512 x 8 byte keys per 4KB page
- **Idea**: Binary Search the Fence Pointers on Disk First

# Example



Binary Search: >51200, ≤ 51322

178 | 273 | ... | 50,956 | 51,200 | 51,322 | ... | ...

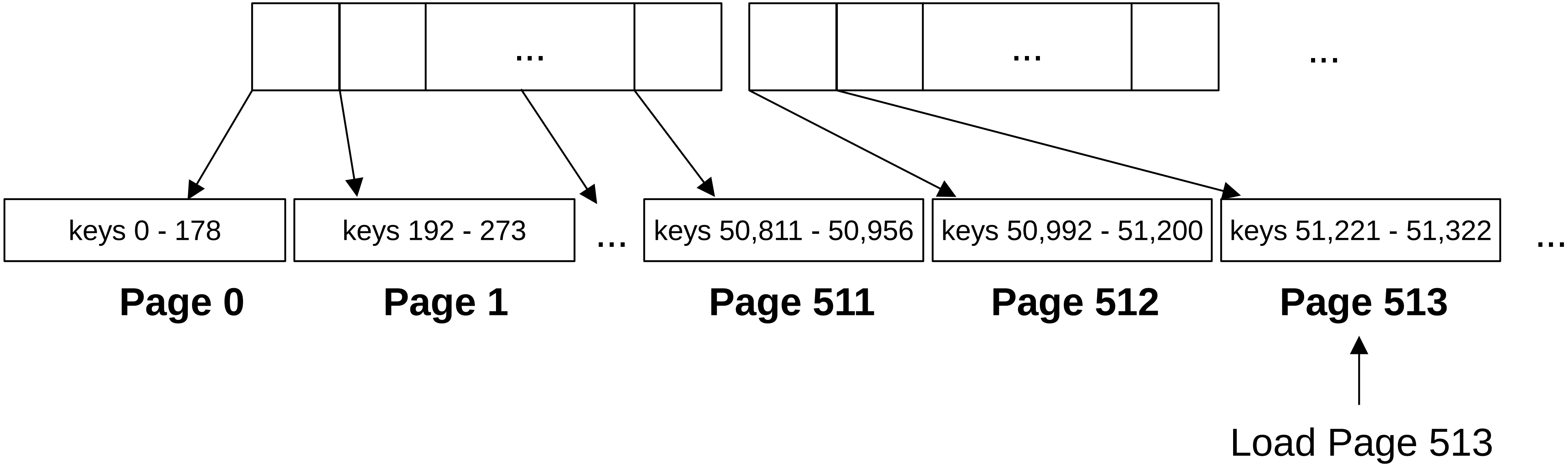
**Array Index:** **0** **1** **...** **511** **512** **513** **...**

keys 0 - 178 | keys 192 - 273 | ... | keys 50,811 - 50,956 | keys 50,992 - 51,200 | keys 51,221 - 51,322 | ...

**Page 0** **Page 1** **Page 511** **Page 512** **Page 513**

Load Page 513

# Example


...
...
...
keys 0 - 178
keys 192 - 273
...
keys 50,811 - 50,956
keys 50,992 - 51,200
keys 51,221 - 51,322
...
Page 0
Page 1
Page 511
Page 512
Page 513
Load Page 513

# Improving on Fence Pointers

- Store the Fence Pointers on Disk
  - 512 x 8 byte keys per 4KB page
- **Idea**: Binary Search the Fence Pointers on Disk First
  - $2^{20}$ records / 64 records per page = $2^{14}$ pages of records
  - $2^{14}$ fence pointer keys = $2^{5}$ pages of fence pointers
  - 512 = $2^{9}$ keys per page
- Total pages searched: 5

  = log(n) - log(records per page) - log(keys per page)

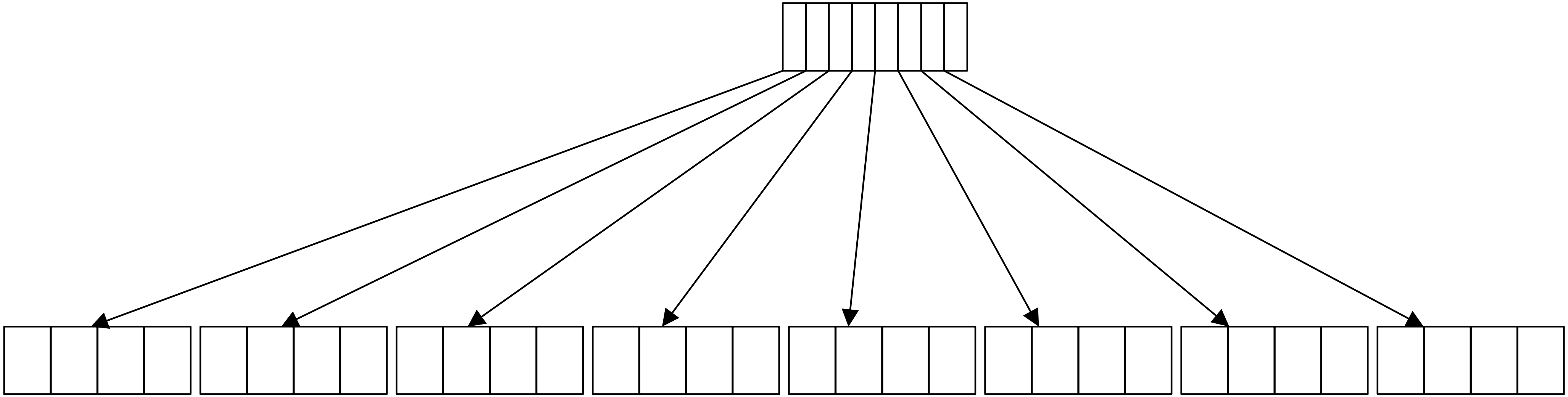
# Improving on Fence Pointers

- Example IO Requirements
  - 5 reads for binary search on the Fence Pointer File
  - 1 read on the data Array
- IO Complexity
  - $C_{data}$ = Records per page (e.g., 64)
  - $C_{key}$ = Keys per page (e.g., 512)
  - Total complexity: $\log(n) - \log(C_{data}) - \log(C_{key})$

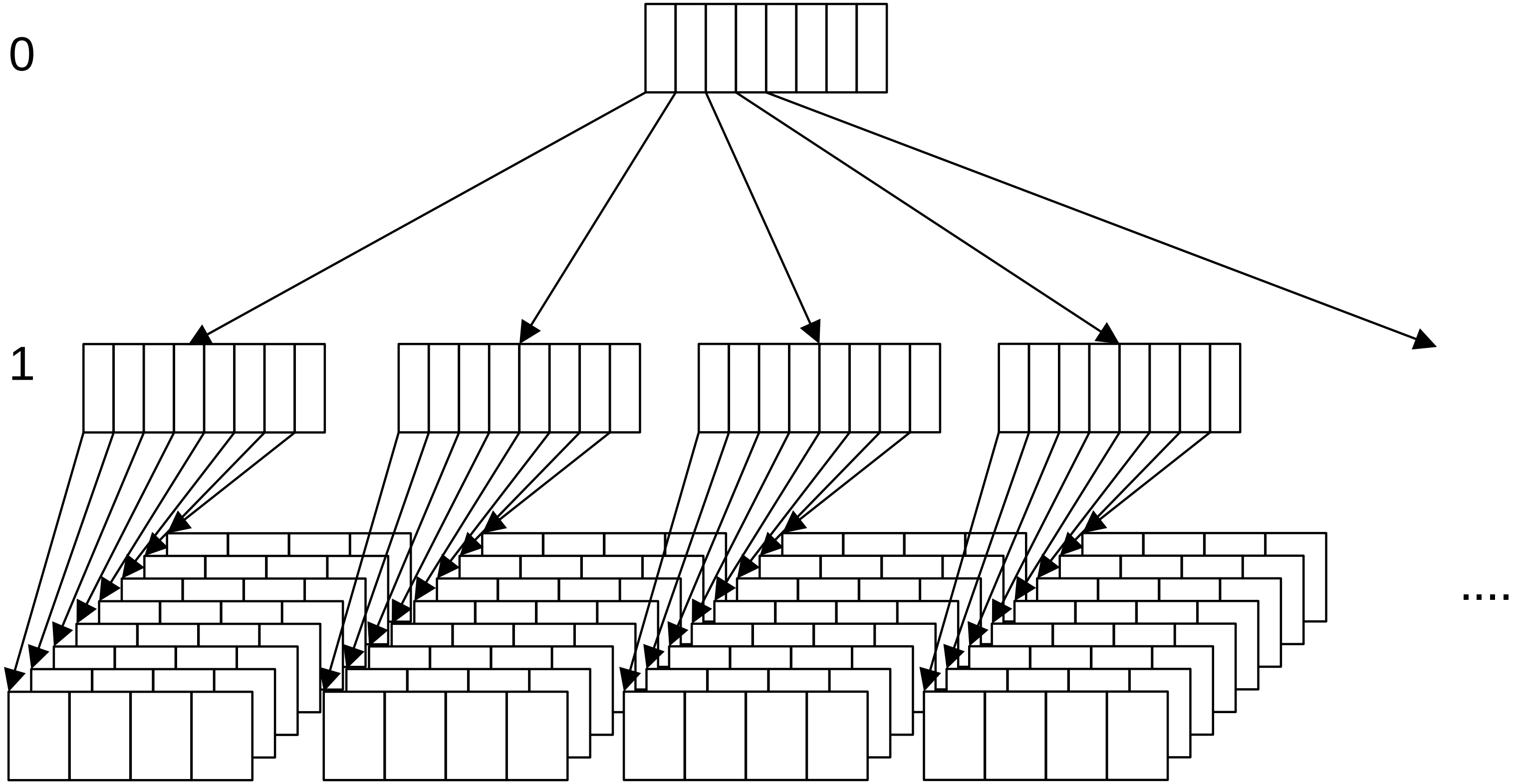
# Improving on Fence Pointers

- **Idea**: Multiple levels of fence pointers
    - Store the greatest key of each fence pointer page.
    - If it fits in memory, done!
    - If not, add another level

# Improving on Fence Pointers



# Improving on Fence Pointers



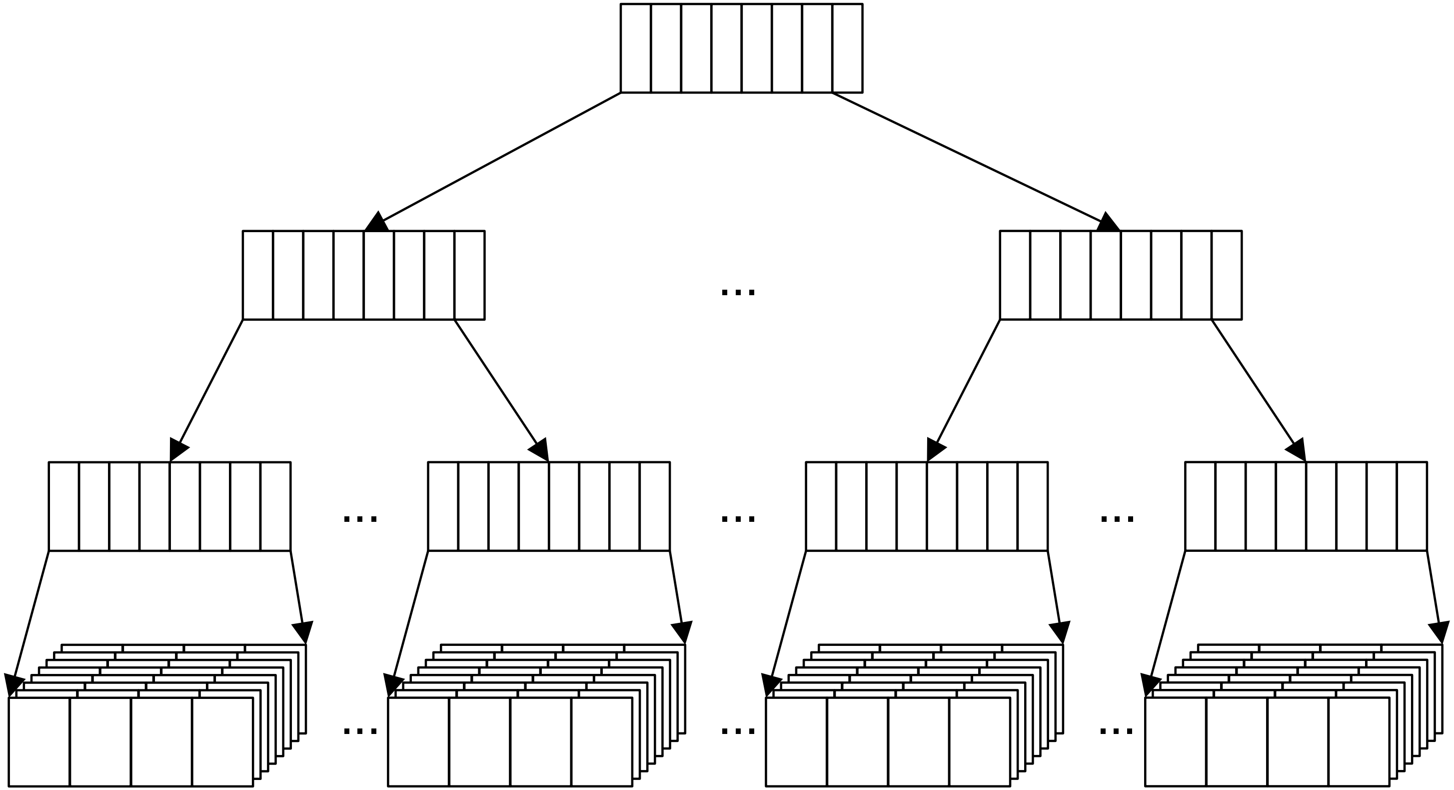
Binary Search @ Level 0
to find a Level 1 page

Binary Search @ Level 1
to find a Data page

Binary Search @ Data
to find the record

# ISAM Index

# ~~Improving on Fence Pointers~~



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

- IO Complexity
  - 1 read at L0 (or assume already in memory)
  - 1 read at L1
  - 1 read at L2
  - …
  - 1 read at $L_{max}$
  - 1 read at Data level

# ISAM Index

- How many levels will there be?
  - Level 0 : 1 page w/ $C_{key}$ keys
  - Level 1 : Up to $C_{key}$ pages w/ $C_{key}^2$ keys
  - Level 2 : Up to $C_{key}^2$ pages w/ $C_{key}^3$ keys
  - Level 3 : Up to $C_{key}^3$ pages w/ $C_{key}^4$ keys
  - Level max : Up to $C_{key}^{max}$ pages w/ $C_{key}^{max+1}$ keys
  - Data level : Up to $C_{key}^{max+1}$ pages w/ $C_{data}\,C_{key}^{max+1}$ records

# ISAM Index

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

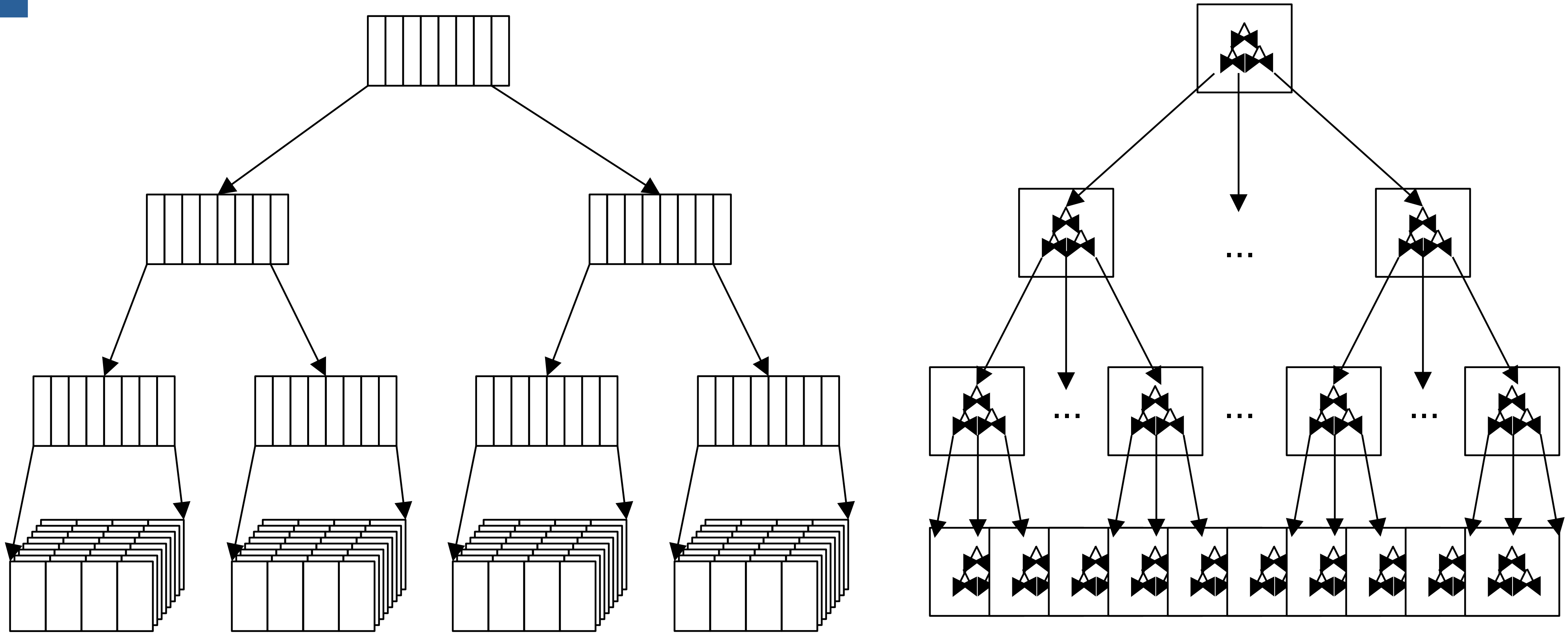
$$\frac{n}{C_{data}} = C_{key}^{max+1}$$

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

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

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

# ISAM Index vs Binary Search...



**Like Binary Search, but "Cache-Friendly"**

# ISAM Index

- As discussed: Disk → Memory
  - Also works for Memory → Cache
    - $C_{key} = {}^{64}/_{8} = 8$
    - $\log_8(n) \ll \log_2(n)$

# ISAM Index

**What if the data changes?**