Write-, Read-Optimized Layouts

CSE 4/562: Database Systems | Lecture 10

DB. Sys.: T.C.B.: Ch. 8.1-8.2

The Log Structured Merge Tree

Decomposable Searching Problems: I. Static to Dynamic Transformation

An Introduction to Bε-trees and Write-Optimization

The Case for Learned Indexes

Specialized Indexes

- Read and Write Amplification
- Optimizing for Writes (at the cost of reads)
  - LSM Trees / Bentley-Saxe Structures
  - $\beta - \varepsilon$ Trees
- Optimizing for Reads (at the cost of writes)
  - CDF-based Indexing

Existing File

- Page 0
- ...
- Page 7510
  - Slot 0: “Avery”
  - Slot 1: “Brea”
  - Slot 2: “Chen”
  - Slot 3: “Div”
  - Slot 4: **EMPTY**
  - Slot 5: “Fiona”
  - ...

Write Amplification

How many times do we need to write the record “Div”?

Write Amplification

The write amplification of a record is
the number of times it will be written to disk

Write Amplification

With B records per page, how many times (Expected and Worst Case) will a record be written to disk after N insertions in a...

- Heap File?

Write Amplification

With B records per page, how many times (Expected and Worst Case) will a record be written to disk after N insertions in a...

- Heap File?
- B+ Tree?

Write Amplification

With B records per page, how many times (Expected and Worst Case) will a record be written to disk after N insertions in a...

- Heap File?
- B+ Tree?
- Hash Table?

Write Amplification

With B records per page, how many times (Expected and Worst Case) will a record be written to disk after N insertions in a...

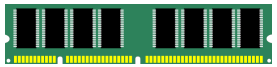
- Heap File?
- B+ Tree?
- Hash Table?
- ISAM Tree?

Write Amplification

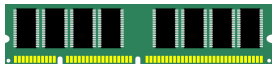
With B records per page, how many times (Expected and Worst Case) will a record be written to disk after N insertions in a...

- Heap File?
- B+ Tree?
- Hash Table?
- ISAM Tree?
- Sorted Array?

Buffering

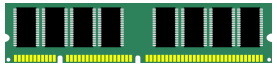


Buffering



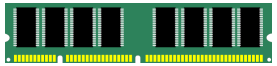
R_1

Buffering



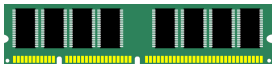
R_1, R_2

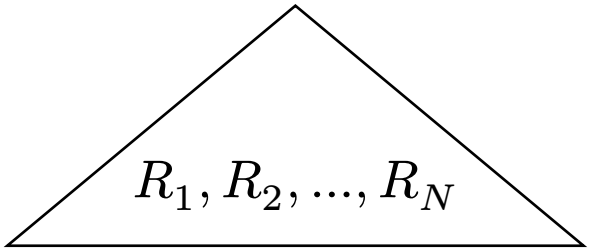
Buffering



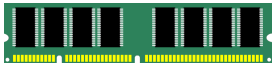
$$R_1, R_2, ..., R_N$$

Buffering





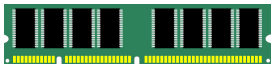
Buffering

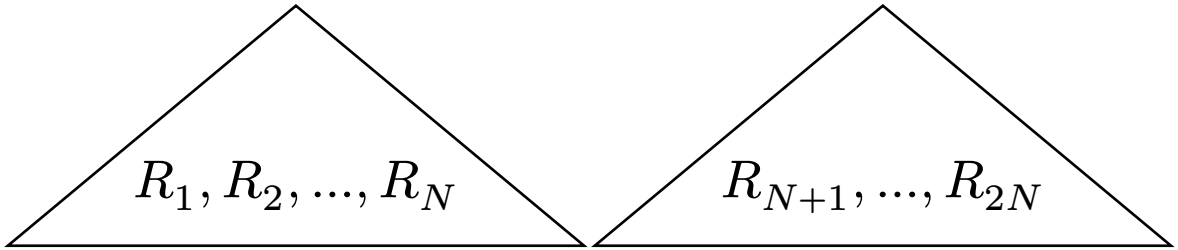


$R_{N+1}, ..., R_{2N}$

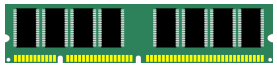
$R_1, R_2, ..., R_N$

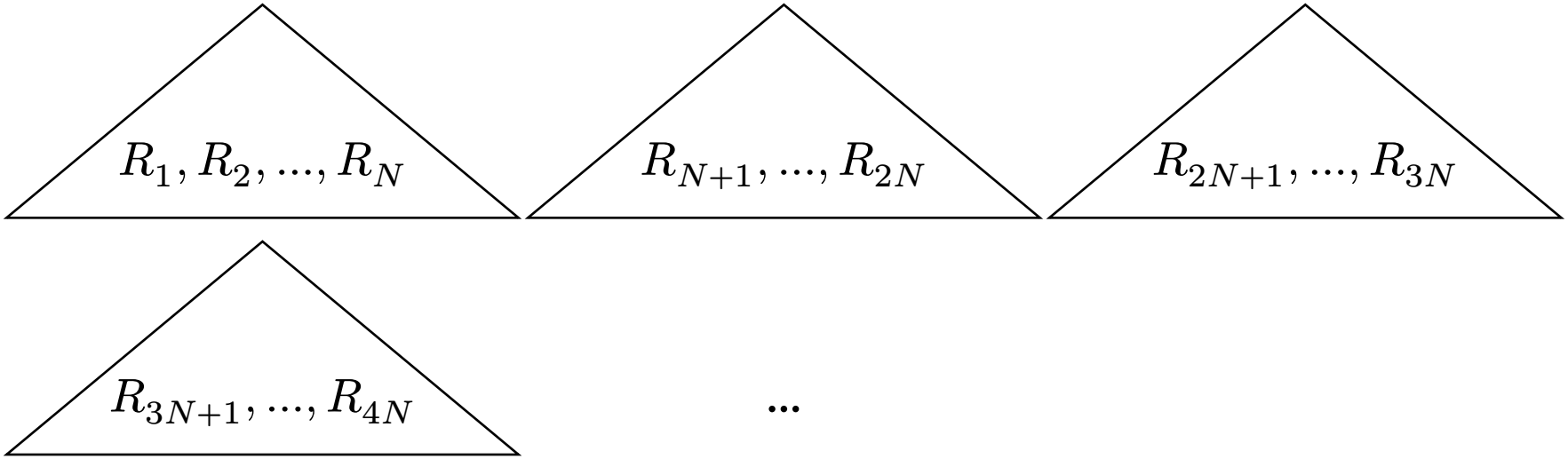
Buffering





Buffering





What is the write amplification of this data structure?

Is this a good index?

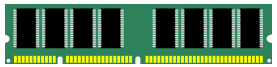
Read Amplification

The read amplification of a record is
the number of places on disk where it could reside

Amplification

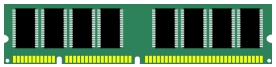
Strategy	Read Amplification	(Expected) Write Amplification
Constantly Update (normal indexes)	1	$\frac{N}{B}$
Append Only	$\frac{N}{B}$	1

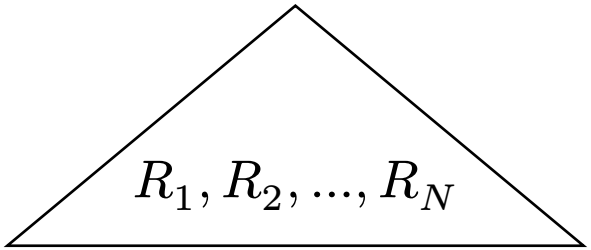
Buffering



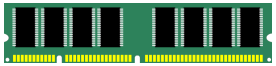
$$R_1, R_2, ..., R_N)$$

Buffering





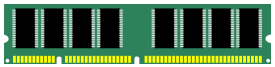
Buffering

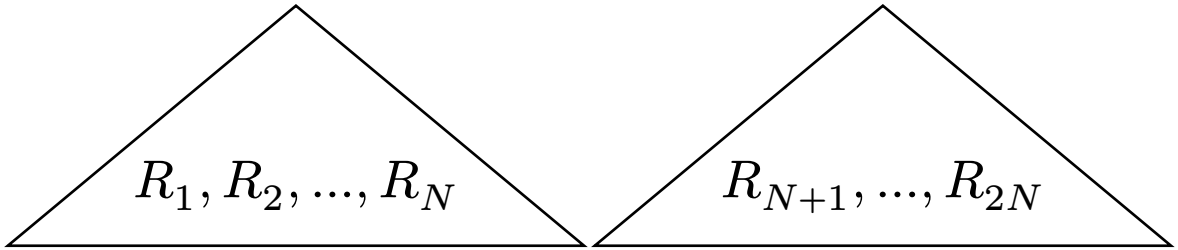


$R_{N+1}, ..., R_{2N}$

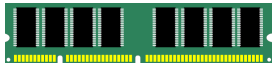
$R_1, R_2, ..., R_N$

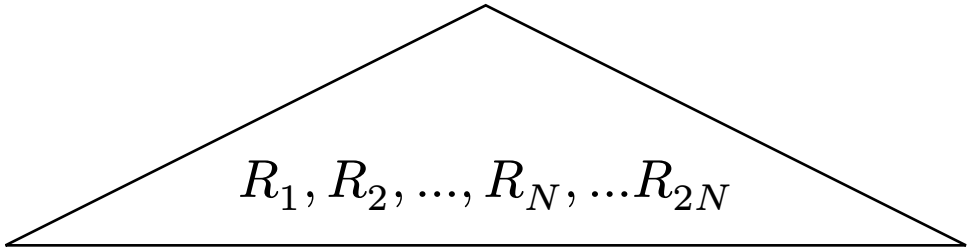
Buffering



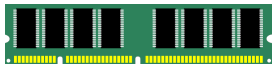


Buffering

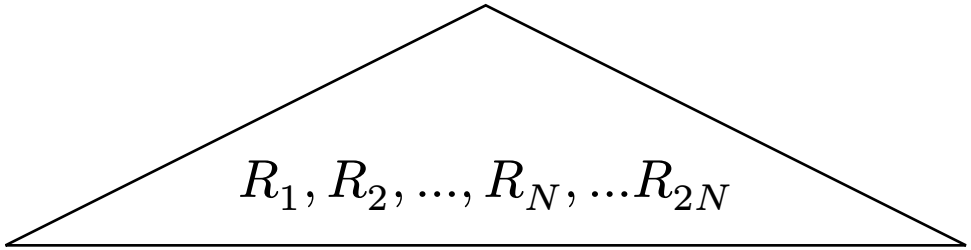




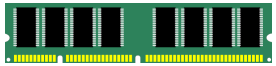
Buffering

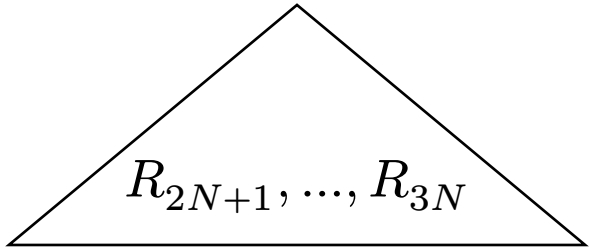


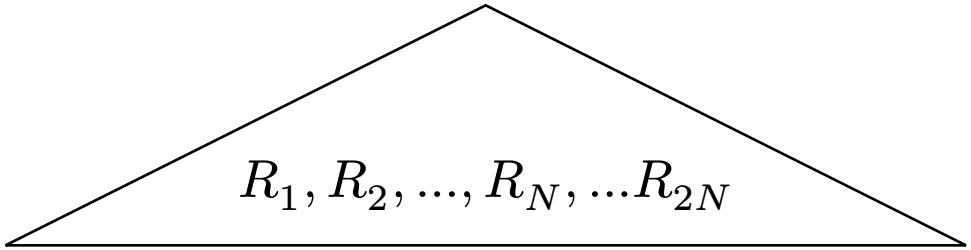
$R_{2N+1}, ..., R_{3N}$



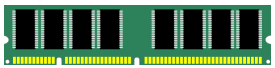
Buffering

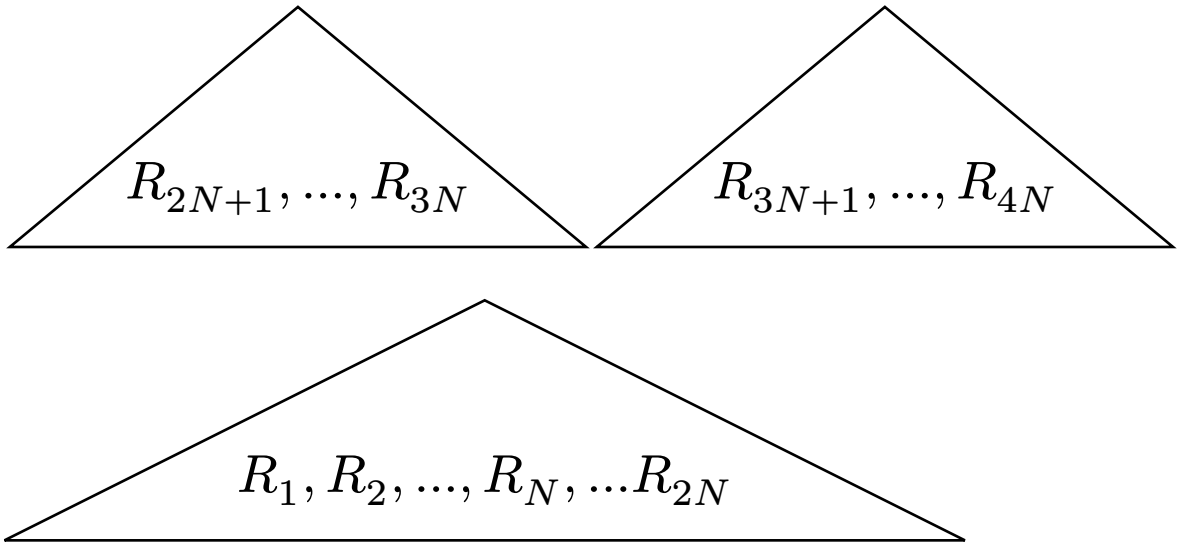




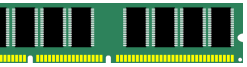


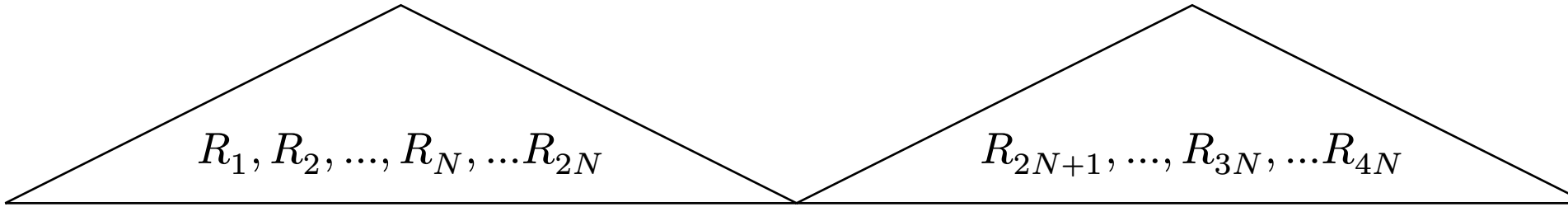
Buffering



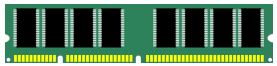


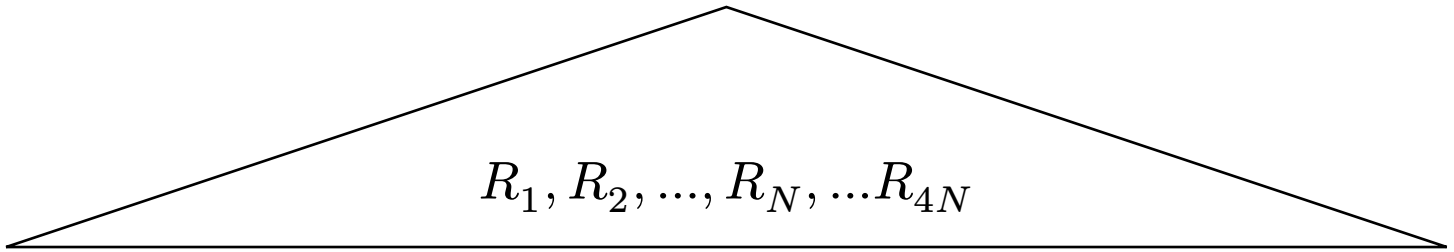
Buffering





Buffering





- Level 0: RAM. B records
- Level 1: Disk. B records
- Level 2: Disk. $2B$ records
- Level 3: Disk. $4B$ records
- Level 4: Disk. $8B$ records
- Level i: Disk. $2^{i-1}B$ records

Questions

- How many levels do we need for N records?
- What is the write amplification of this structure?
- What is the read amplification of this structure?

- 1996: O'Neil, Cheng, Gawlick, O'Neil "The Log Structured Merge Tree"
  - Each layer is an ISAM index
- 1980: Bentley, Saxe "Decomposable Searching Problems: I. Static to Dynamic Transformation"
  - Each layer is any immutable index

Requirements of Bentley-Saxe

There exists an efficiently computable way to...

- ... merge answers for queries over indices constructed on disjoint sets of answers.
- ... merge indices constructed on disjoint sets of answers.

Leveling vs Tiering

- **Leveling**: As described above.
- **Tiering**: **K** runs as each “tier”.
  - Lower write amplification ($O\left(\log_{K(N)}\right)$ vs $O(\log_2(N))$)
  - Higher read amplification ($O\left(K \log_{K(N)}\right)$ vs $O(\log_2(N))$)

Storage

- Sorted runs
- Fence pointer table (no ISAM)
- Bloom filters

Normal sets

- `Insert(A)`
- `Insert(B)`
- `Test(A)` $\rightarrow$ `true`
- `Test(C)` $\rightarrow$ `false`

A regular set needs to explicitly store every element inserted into it

Lossy sets

- `Insert(A)`
- `Insert(B)`
- `Test(A)` → `true`
- `Test(C)` → `false`
- `Test(D)` → `true` ✅
- `Test(B)` → `false` ❌

A lossy set can provide false positives, but never false negatives!

Every item is assigned to a random $\frac{K}{S}$ bits:

- A: 00001001
- B: 00010010
- C: 00010001
- D: 10001000

The bloom filter is the bitwise OR of the bits assigned to its elements

$$\text{Bloom}(A, B) = 00001001 \mid 00010010 = 00011011$$

We test by checking to see if every bit in the checked element is set

- $\text{Test}(A, 00011011) = (00001001 \& 00011011 == 00001001)$

Every item is assigned to a random $\frac{K}{S}$ bits:

- A: 00001001
- B: 00010010
- C: 00010001
- D: 10001000

The bloom filter is the bitwise OR of the bits assigned to its elements

$$\text{Bloom}(A, B) = 00001001 \mid 00010010 = 00011011$$

We test by checking to see if every bit in the checked element is set

- $\text{Test}(A, 00011011) = (00001001 \& 00011011 == 00001001)$
- $\text{Test}(B, 00011011) = (00010010 \& 00011011 == 00010010)$

Every item is assigned to a random $\frac{K}{S}$ bits:

- A: 00001001
- B: 00010010
- C: 00010001
- D: 10001000

The bloom filter is the bitwise OR of the bits assigned to its elements

$$\text{Bloom}(A, B) = 00001001 \mid 00010010 = 00011011$$

We test by checking to see if every bit in the checked element is set

- $\text{Test}(A, 00011011) = (00001001 \& 00011011 == 00001001)$
- $\text{Test}(B, 00011011) = (00010010 \& 00011011 == 00010010)$
- $\text{Test}(C, 00011011) = (00010001 \& 00011011 == 00010001)$

Every item is assigned to a random $\frac{K}{S}$ bits:

- A: 00001001
- B: 00010010
- C: 00010001
- D: 10001000

The bloom filter is the bitwise OR of the bits assigned to its elements

$$\text{Bloom}(A, B) = 00001001 \mid 00010010 = 00011011$$

We test by checking to see if every bit in the checked element is set

- $\text{Test}(A, 00011011) = (00001001 \& 00011011 == 00001001)$
- $\text{Test}(B, 00011011) = (00010010 \& 00011011 == 00010010)$
- $\text{Test}(C, 00011011) = (00010001 \& 00011011 == 00010001)$
- $\text{Test}(D, 00011011) = (10001000 \& 00011011 == 10001000)$

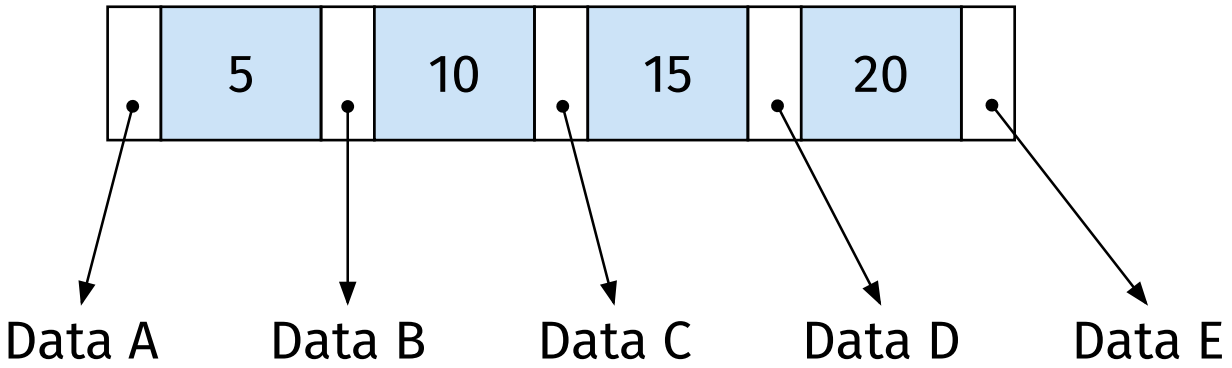
Optimal K depends roughly linearly on S

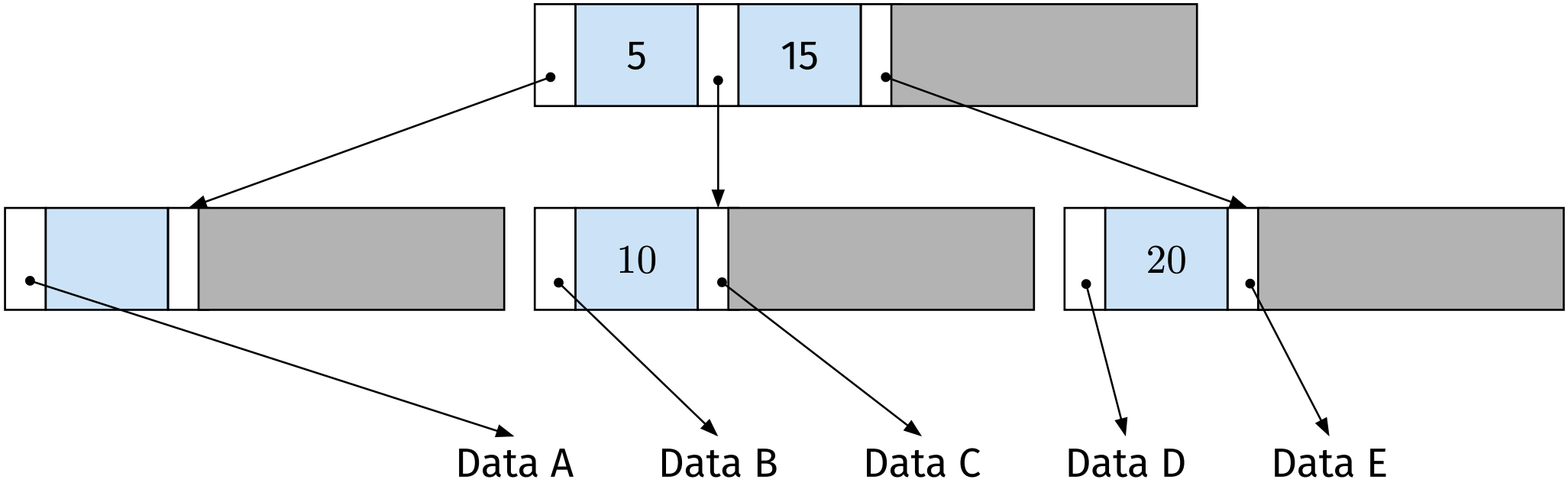
- $S = 5N$ bits: 10% collision chance
- $S = 10N$ bits: 1% collision chance
- vs $32N$ or $64N$ bits for an integer set

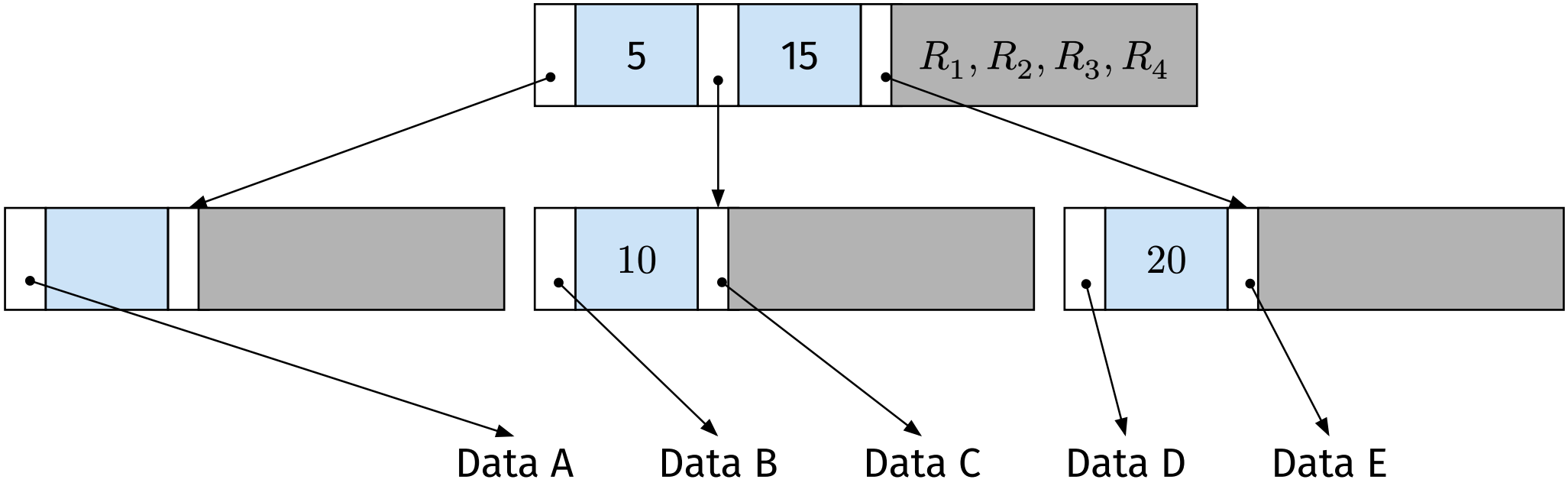
Accessing an LSM Tree

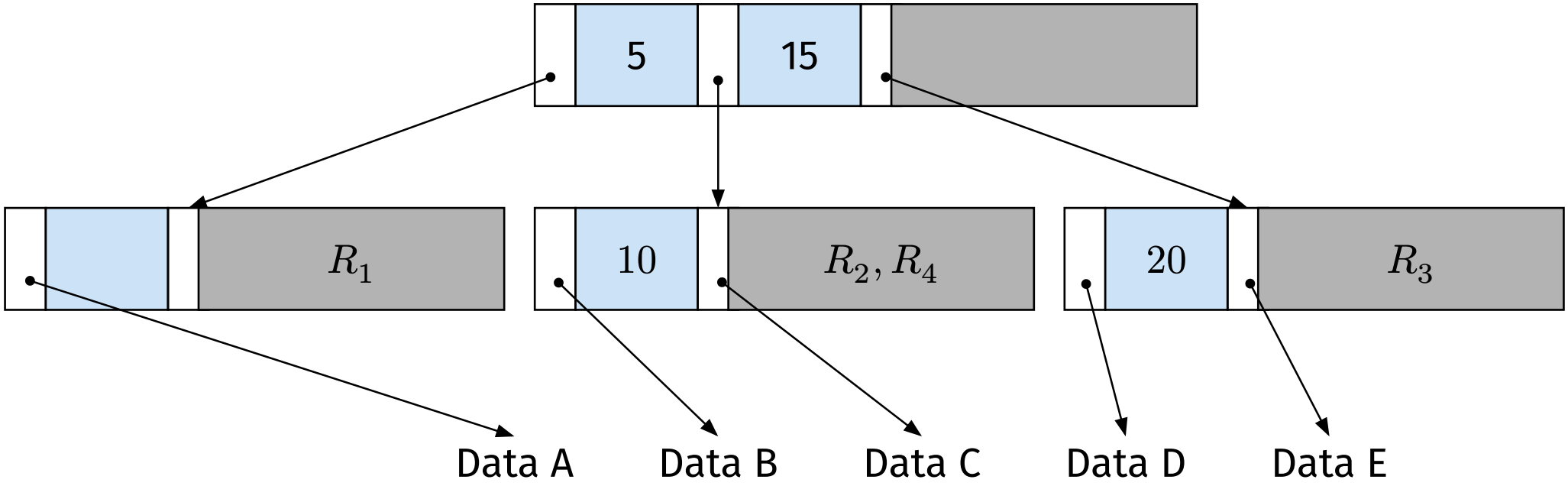

```
def point_lookup(key):
  for layer in all_layers:
    if layer.bloom_filter[key] is False:
      return None

    page_id = layer.fence_pointer[key]

    page = db_read(page_id)
    if record := page.binary_search(key) is not None:
      return record
```

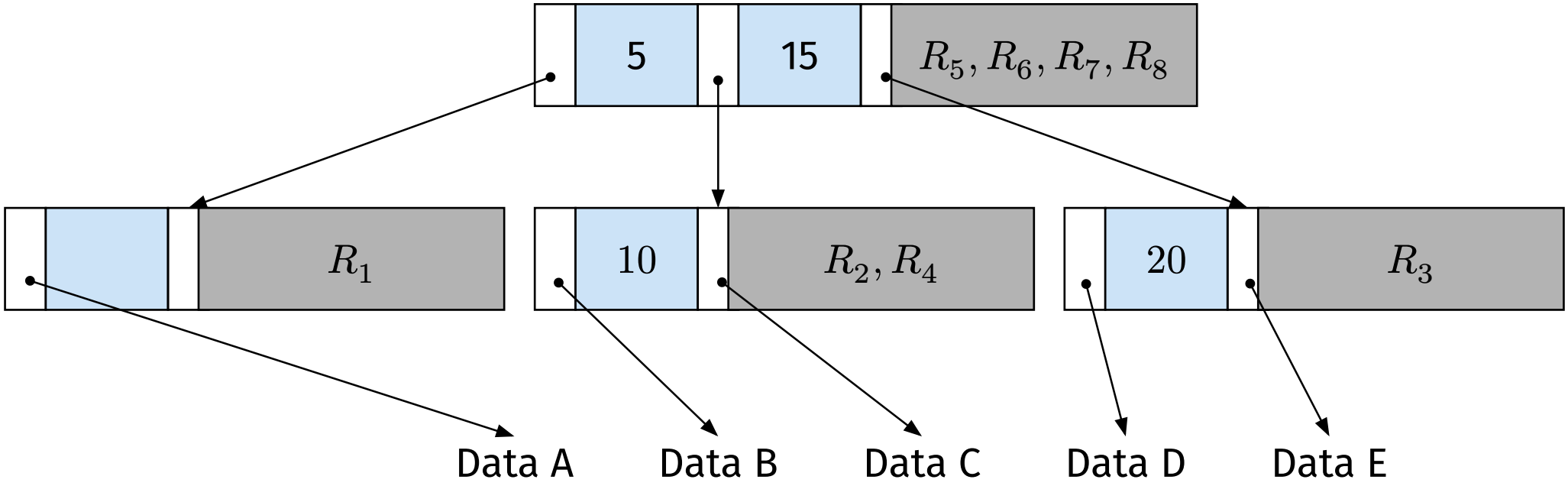
What if $\log(N)$ write amplification is too much?


5
10
15
20
Data A
Data B
Data C
Data D
Data E


5
15
10
20
Data A
Data B
Data C
Data D
Data E


5
15
R_1, R_2, R_3, R_4
10
20
Data A
Data B
Data C
Data D
Data E


5
15
R_1
10
R_2, R_4
20
R_3
Data A
Data B
Data C
Data D
Data E


5
15
R_5, R_6, R_7, R_8
R_1
10
R_2, R_4
20
R_3
Data A
Data B
Data C
Data D
Data E

What if we have infinite prep time?

Fence Pointer Table

Key	Page
$0 \leq k < 1103$	0
$1103 \leq k < 1867$	1
$1867 \leq k < 3093$	2
$3093 \leq k < 3214$	3
$3214 \leq k < 3378$	4
$3378 \leq k < 4353$	5
$4353 \leq k < 5007$	6
$5007 \leq k < 9985$	7

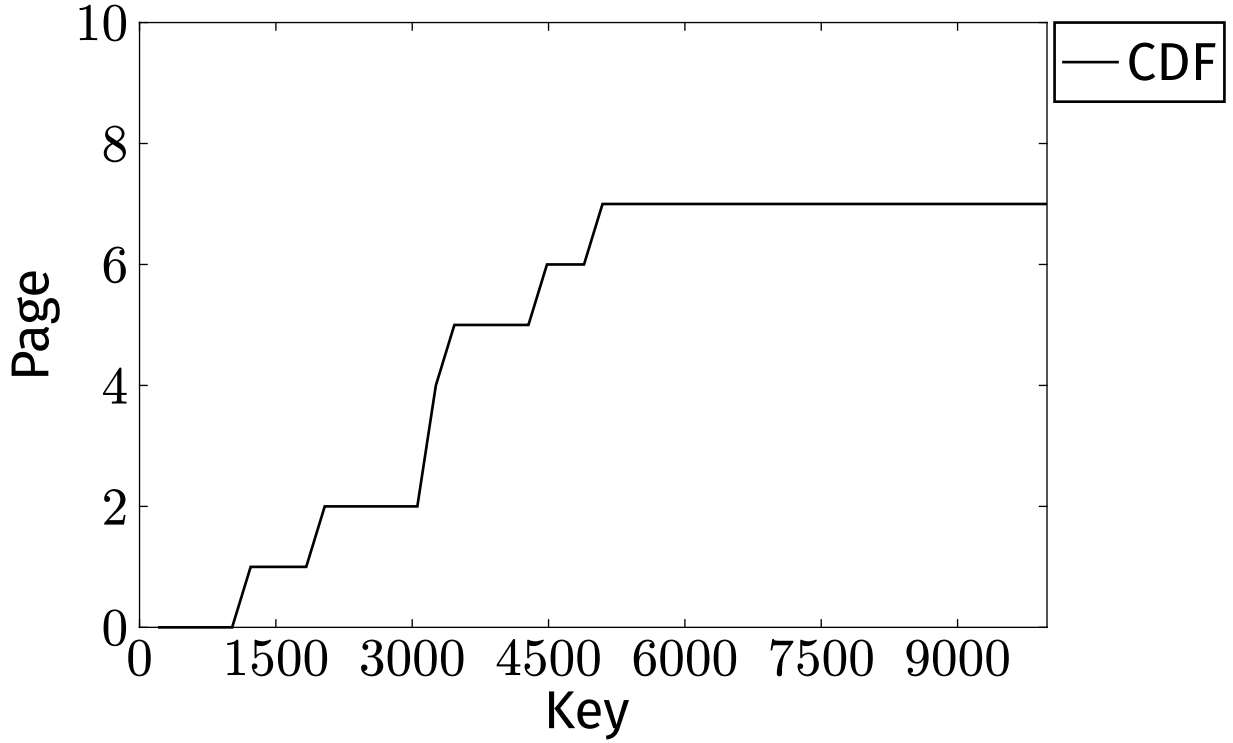
Fence Pointer Table

Key	Page
0	0
1	0
2	0
...	0
1102	0
1103	1
...	1
1866	1
1867	2
...	...

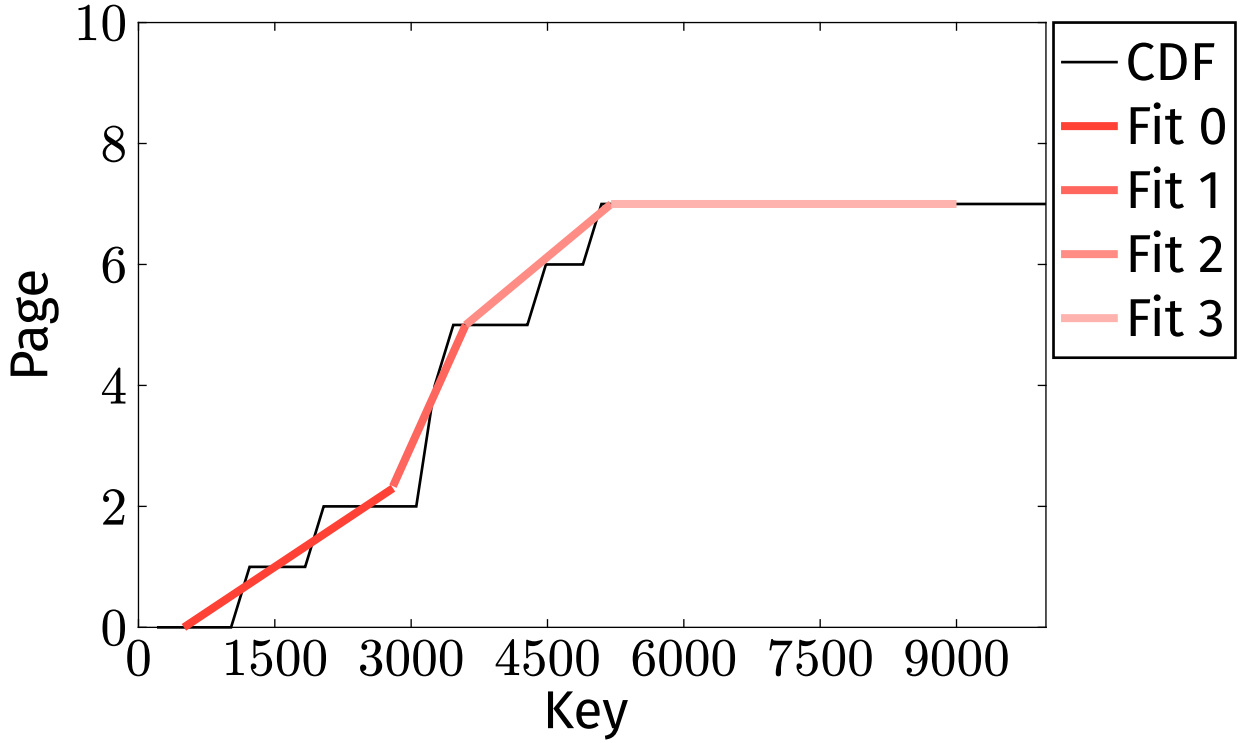
$$f(\text{key}) \rightarrow \text{page}$$

- Fence Pointer Table
- B+ Tree
- Hash Function
- ...?

Fence Pointer Table



Fence Pointer Table



CDF-Based Indexing v1

Idea: Fit m, b in $\text{page} = m \cdot \text{key} + b$

Idea: Fit m, b in $\text{page} = m \cdot \text{key} + b$

Problem

The function will not usually fit precisely

Idea: Fit m, b in $\text{page} = m \cdot \text{key} + b$

Problem

The function will not usually fit precisely

Fixes

- Bounded Error
- More functions
- More interesting functions

With $f'(\text{key}) = m \cdot \text{key} + b$

When Fitting

Compute $\text{max\_err} = \max_{\text{key}} |f(\text{key}) - f'(\text{key})|$

During a Lookup

Binary search in $[f'(\text{key}) \pm \text{max\_err}]$

or

Expanding ring search from $f'(\text{key})$

Binary Search

- Requires an upper **and** lower bound
- Each step cuts the bounded space in half

At most $\log_2(N)$ steps to find any record

... but what if you don't have a fully bounded search to start?

Expanding Ring Search

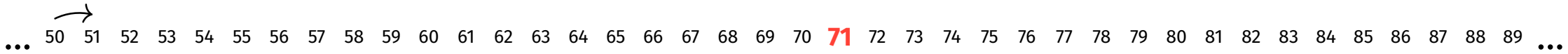
- Requires an upper **or** lower bound

At most $2\log_2(N)$ steps to find any record

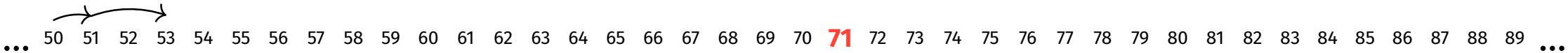
Expanding Ring Search

... 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 **71** 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 ...

Expanding Ring Search

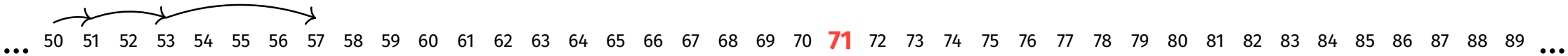


Expanding Ring Search



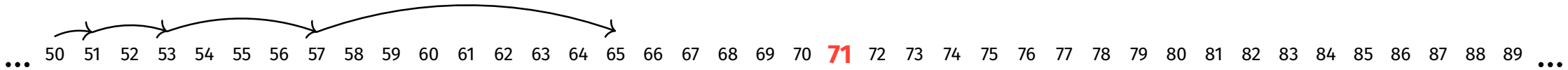
1. Double step size.

Expanding Ring Search



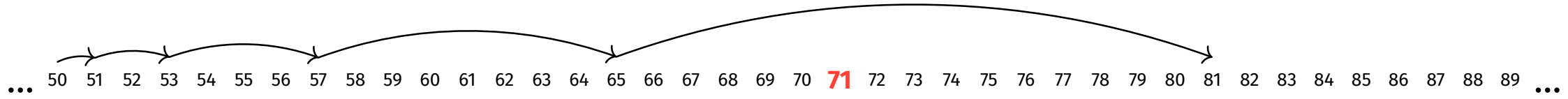
1. Double step size.

Expanding Ring Search



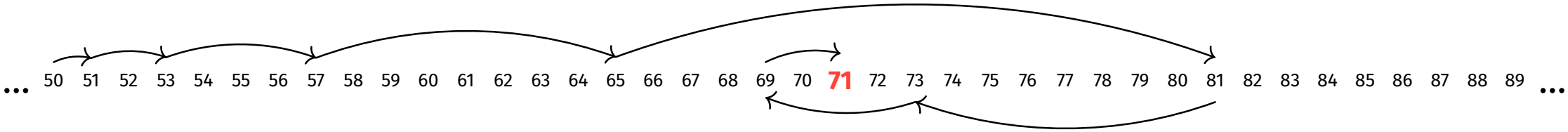
1. Double step size.

Expanding Ring Search



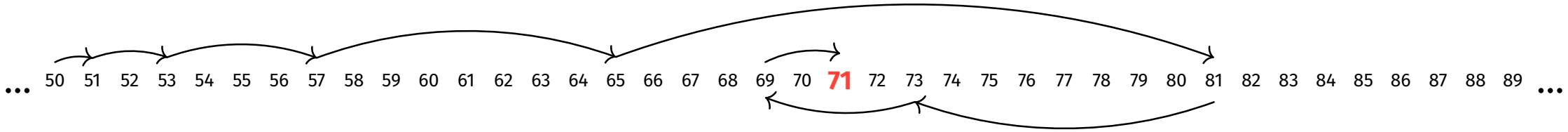
1. Double step size until you pass the target.

Expanding Ring Search



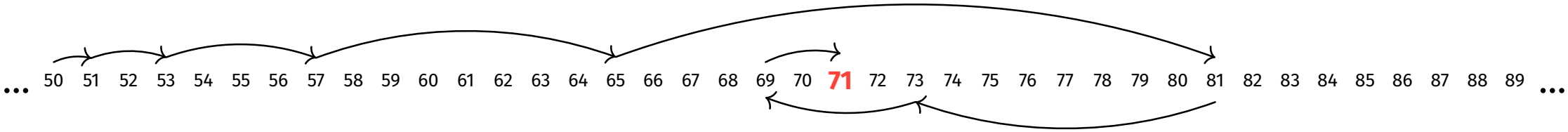
1. Double step size until you pass the target.
2. Use upper/lower bounds to binary search.

Expanding Ring Search



1. Double step size until you pass the target. (at most $\log_2|\text{start} - \text{goal}|$ steps)
2. Use upper/lower bounds to binary search. (at most $\log_2|\text{start} - \text{goal}| - 1$ steps)

Expanding Ring Search



1. Double step size until you pass the target. (at most $\log_2|\text{start} - \text{goal}|$ steps)
2. Use upper/lower bounds to binary search. (at most $\log_2|\text{start} - \text{goal}| - 1$ steps)

Piecewise Linear Functions

B+ Tree, ISAM Tree, Sorted Array, etc... with Linear Regressions at the leaves.

1. Use the index to find the parameters applicable to the key.
2. Plug the key into the regression to find the approximate page.
3. Expanding ring search to find the exact page.

More Interesting Functions...

- Polynomial Regression
- Splines

More Interesting Functions...

- Polynomial Regression
- Splines
- (tiny) Neural Networks

More Interesting Functions...

- Polynomial Regression
- Splines
- (tiny) Neural Networks

> Google did have some success with 2x32 node networks.

More Interesting Functions...

- Polynomial Regression
- Splines
- (tiny) Neural Networks

Google did have some success with 2x32 node networks.

...but the cost of vector/matrix multiplication is high. Piecewise linear regression is faster with low accuracy loss, and B+Trees and FP Tables aren't that much slower.

... still, the name “Learned Indexes” seems to have stuck.

Reminders

- Quiz 3 results posted
- 1 week to Checkpoint 2 due date
  - Don’t forget to sign up for a code review