Building an Efficient Key-Value Store in a Flexible Address Space

Chen Chen[1], Wenshao Zhong[1], Xingbo Wu[2]

[1] University of Illinois at Chicago

[2] Microsoft Research Cambridge

KV Stores

- The backbone of widely used Internet-based services

KV Stores

- The backbone of widely used Internet-based services
- Simple interface
  - `db.put("foo", "abcd")`
  - `db.put("bar", 0xbee)` Write
  - `db.put("cat", 2022)`
  - `db.get("cat") => 2022` Point Query
  - `db.scan("c", 2) => {"cat": 2022, "foo": "abcd"}` Range Query

KV Stores

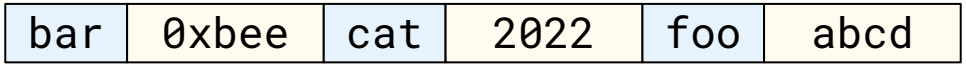
- The backbone of widely used Internet-based services
- Simple interface
  - `db.put("foo", "abcd")`
  - `db.put("bar", 0xbee)` Write
  - `db.put("cat", 2022)`
  - `db.get("cat") => 2022` Point Query
  - `db.scan("c", 2) => {"cat": 2022, "foo": "abcd"}` Range Query

- Major KV stores manage **sorted data** for range queries

Managing Sorted Data

- Storing sorted data in structured files

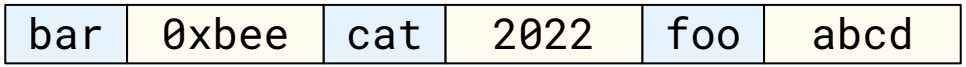
(Simplified) File



Managing Sorted Data

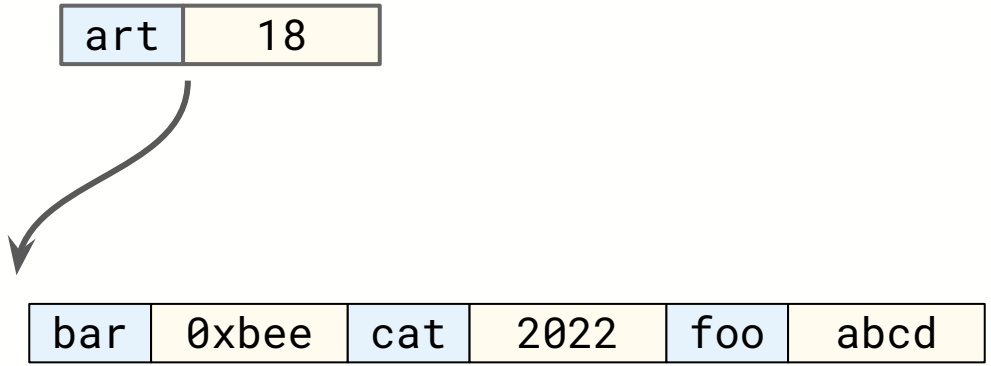
- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`

(Simplified) File



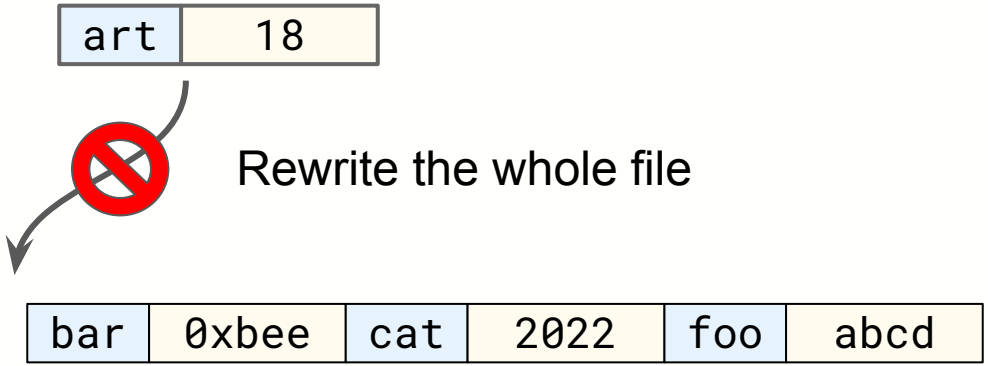
Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`



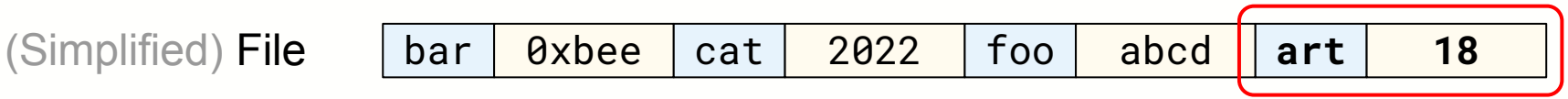
Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`



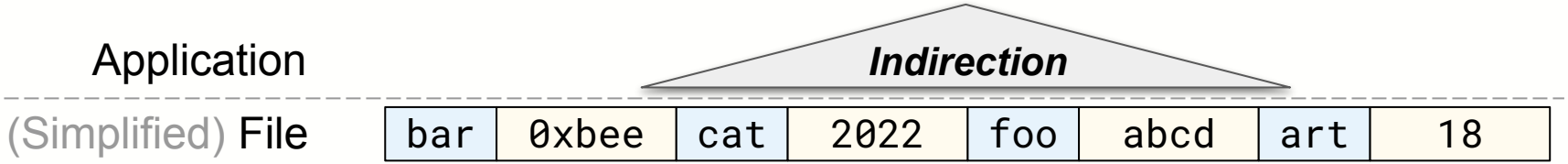
Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - Common approach: append writes



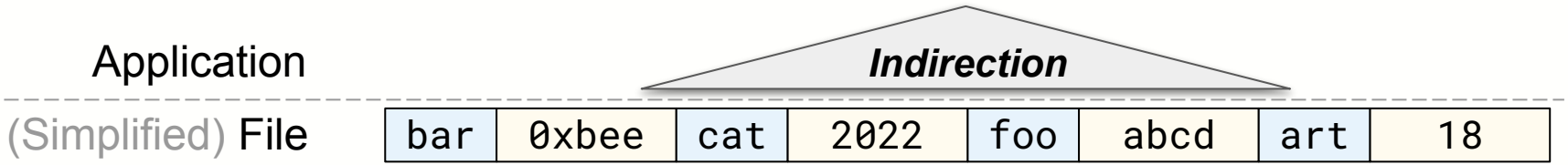
Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - Common approach: append writes + indirections



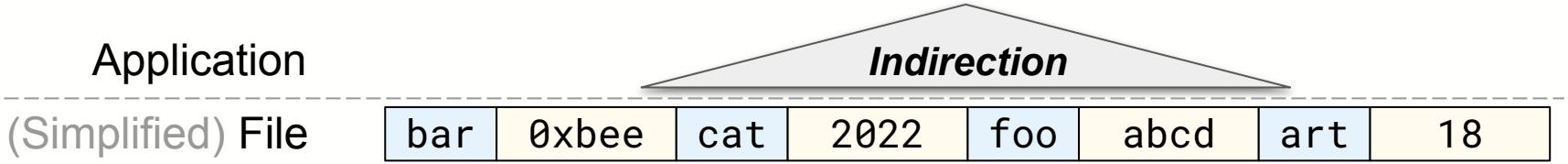
Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - Common approach: append writes + indirections
    - Unordered file but sorted index for *all* keys



Managing Sorted Data

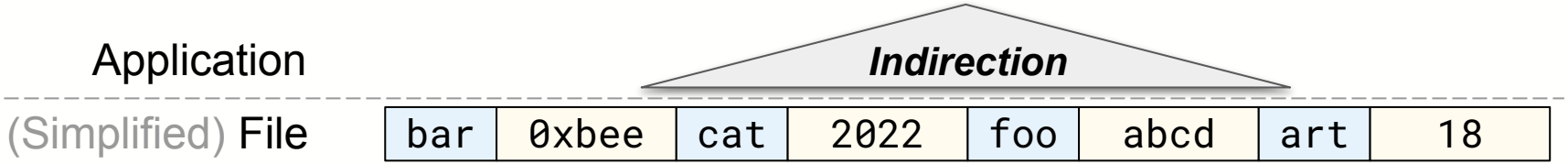
- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - Common approach: append writes + indirections
    - Unordered file but sorted index for *all* keys ☹ Maintaining *large* index



Managing Sorted Data

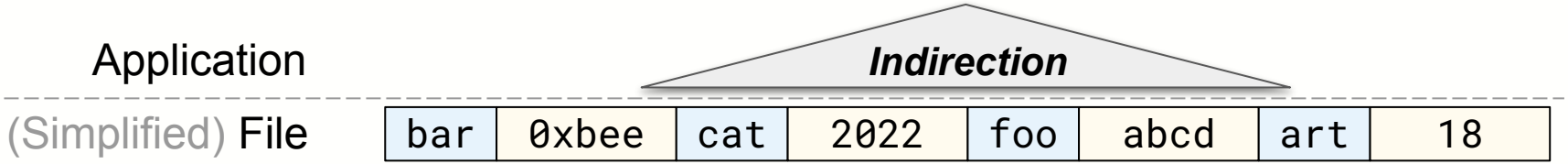
- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - Common approach: append writes + indirections
    - Unordered file but sorted index for *all* keys ☹ Maintaining *large* index
    - Indexing sorted leaf nodes (e.g., B^+-Tree)

      Log-structured sort-merge approach (e.g., LSM)



Managing Sorted Data

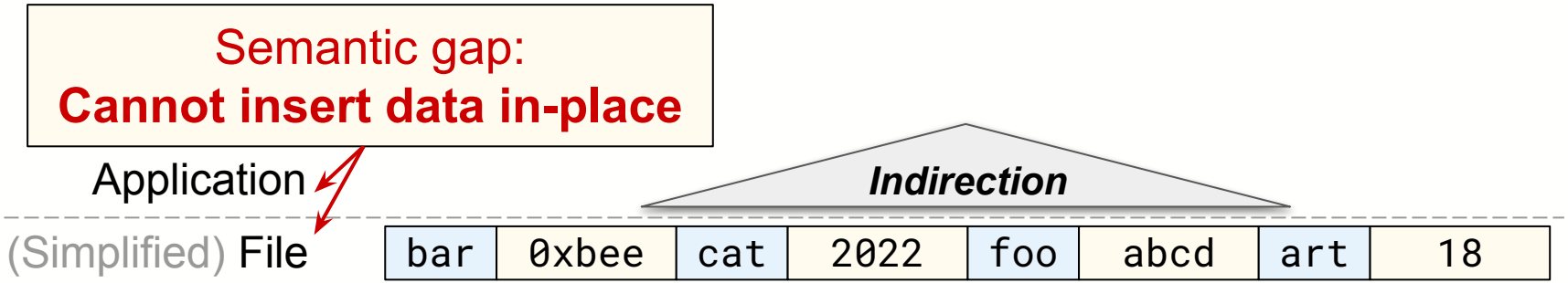
- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - Common approach: append writes + indirections
    - Unordered file but sorted index for *all* keys ☹ Maintaining *large* index
    - Indexing sorted leaf nodes (e.g., B^+-Tree)
      Log-structured sort-merge approach (e.g., LSM) ☹ Repeated *rewrites*



Managing Sorted Data

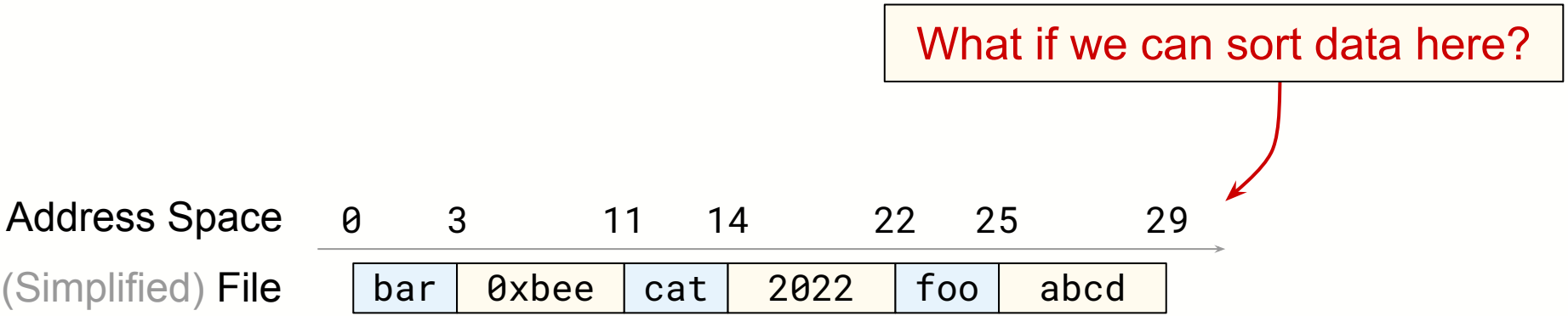
- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - Common approach: append writes + indirections
    - Unordered file but sorted index for *all* keys ☹ Maintaining *large* index
    - Indexing sorted leaf nodes (e.g., B$^+$-Tree)

      Log-structured sort-merge approach (e.g., LSM) ☹ Repeated *rewrites*



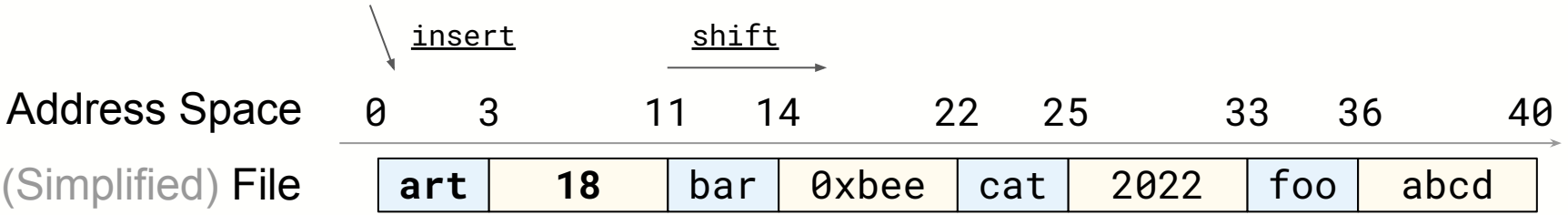
Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`



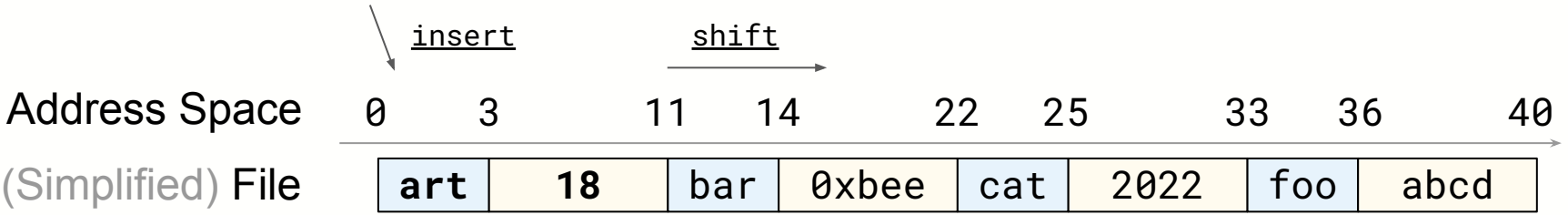
Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - In-place updates by sorting in the address space



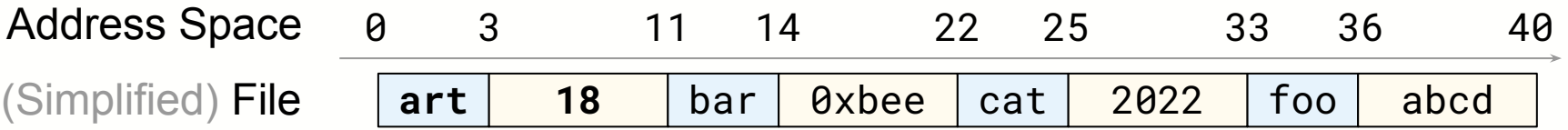
Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - In-place updates by sorting in the address space ☺ Straightforward



Managing Sorted Data

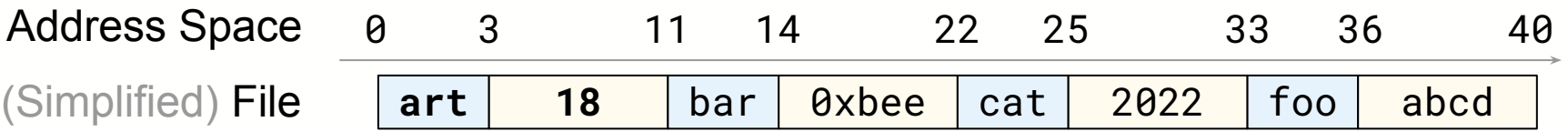
- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - In-place updates by sorting in the address space
  - Existing effort: *insert-range* and *collapse-range* (ext4, XFS, F2FS …)



Managing Sorted Data

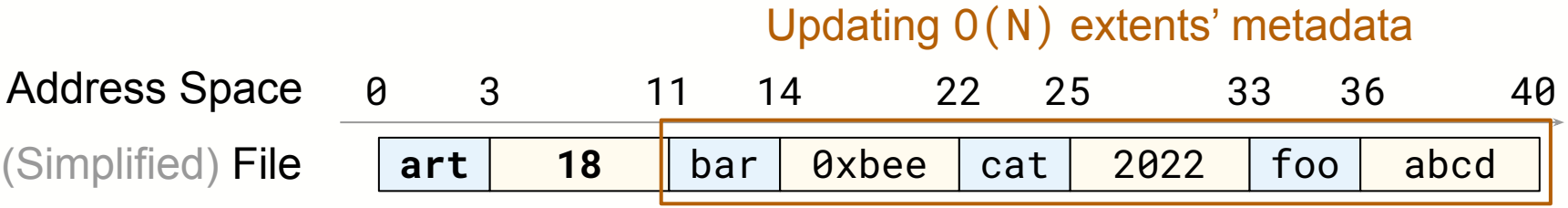
- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - In-place updates by sorting ir
  - Existing effort: *insert-range* a FS …)
    - **Inefficient shifting in extent indexes** ☹

Logical offset -> physical address



Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - In-place updates by sorting in the address space
  - Existing effort: *insert-range* and *collapse-range* (ext4, XFS, F2FS …)
    - **Inefficient shifting in extent indexes** ☹



Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - In-place updates by sorting in the address space
  - Existing effort: *insert-range* and *collapse-range* (ext4, XFS, F2FS …)
    - In

3 orders of magnitude gap

Address Space

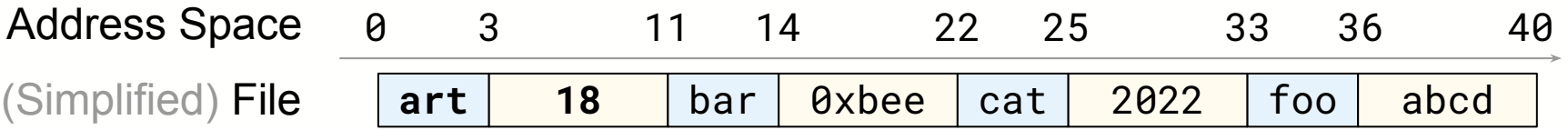
33 36 40

(Simplified) File

art	18	bar	0xbee	cat	2022	foo	abcd

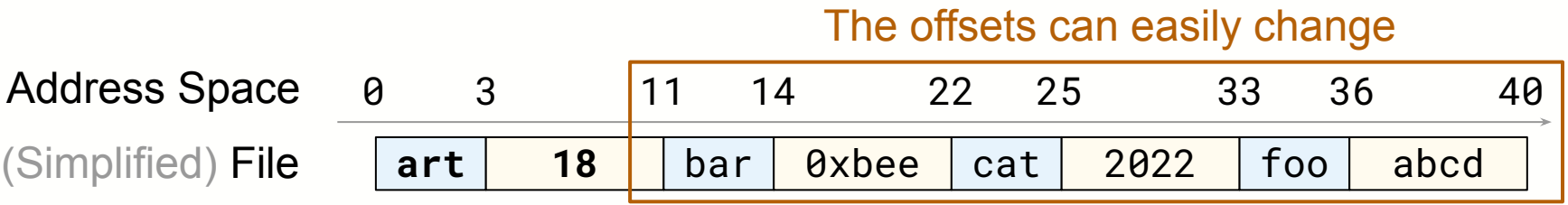
Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - In-place updates by sorting in the address space
  - Existing effort: *insert-range* and *collapse-range* (ext4, XFS, F2FS …)
    - **Inefficient shifting in extent indexes** ☹
    - **Rigid block alignment requirements** ☹



Managing Sorted Data

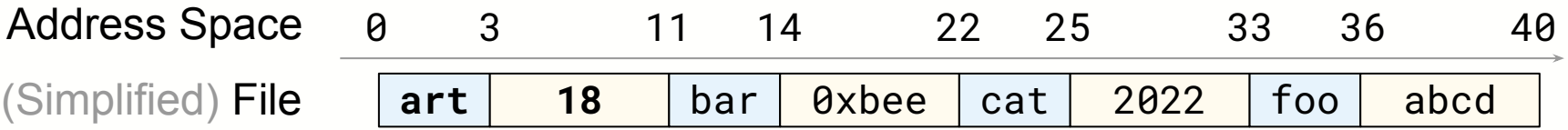
- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - In-place updates by sorting in the address space
  - Existing effort: *insert-range* and *collapse-range* (ext4, XFS, F2FS …)
    - **Inefficient shifting in extent indexes** ☹
    - **Rigid block alignment requirements** ☹
    - **Inability to track shifting data** ☹



Managing Sorted Data

- Storing sorted data in structured files
- How to commit new data? e.g., `db.put("art", 18)`
  - In-place updates by sorting in the address space
  - Existing effort: *insert-range* and *collapse-range* (ext4, XFS, F2FS …)

A more flexible storage abstraction to manage sorted data!



Our Idea: Flexible Address Space

- Lightweight in-place insertions and deletions
  - Sorting data easily in the address space

Our Idea: Flexible Address Space

- Lightweight in-place insertions and deletions
  - Sorting data easily in the address space
- Challenges
  - Inefficient shifting in extent index
  - Rigid block alignment requirements
  - Inability to track shifting data

Our Idea: Flexible Address Space

- Lightweight in-place insertions and deletions
  - Sorting data easily in the address space
- Challenges => The Solution
  - Inefficient shifting in extent index => Index structure with efficient shifting
  - Rigid block alignment requirements => No alignment requirements
  - Inability to track shifting data => Managing shifting data

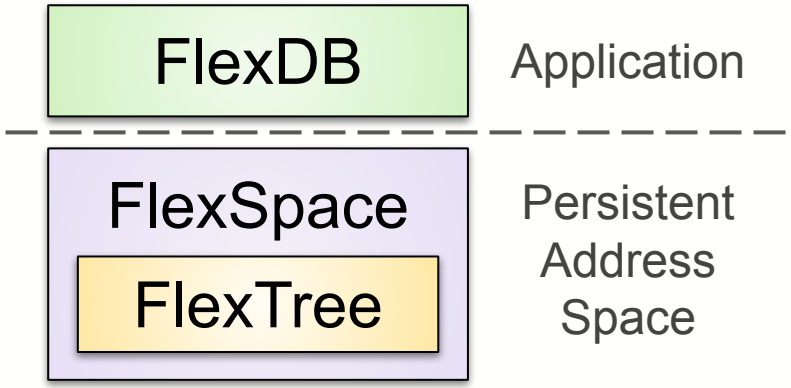
Our Idea: Flexible Address Space

- Lightweight in-place insertions and deletions
  - Sorting data easily in the address space
- The solution:

Managing shifting data

No alignment requirements

Index with efficient shifting



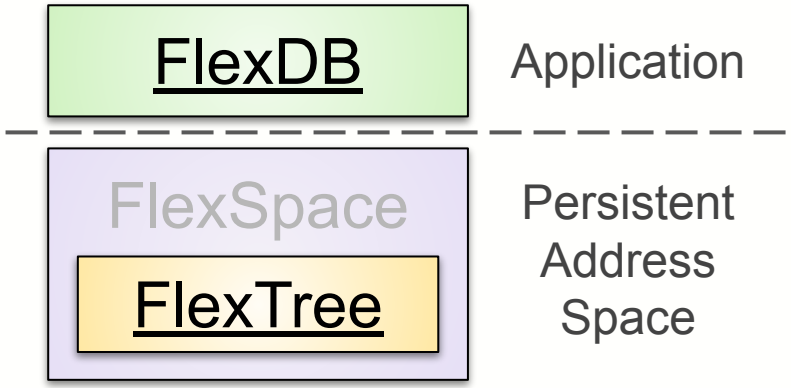
Our Idea: Flexible Address Space

- Lightweight in-place insertions and deletions
  - Sorting data easily in the address space
- The solution:

Managing shifting data

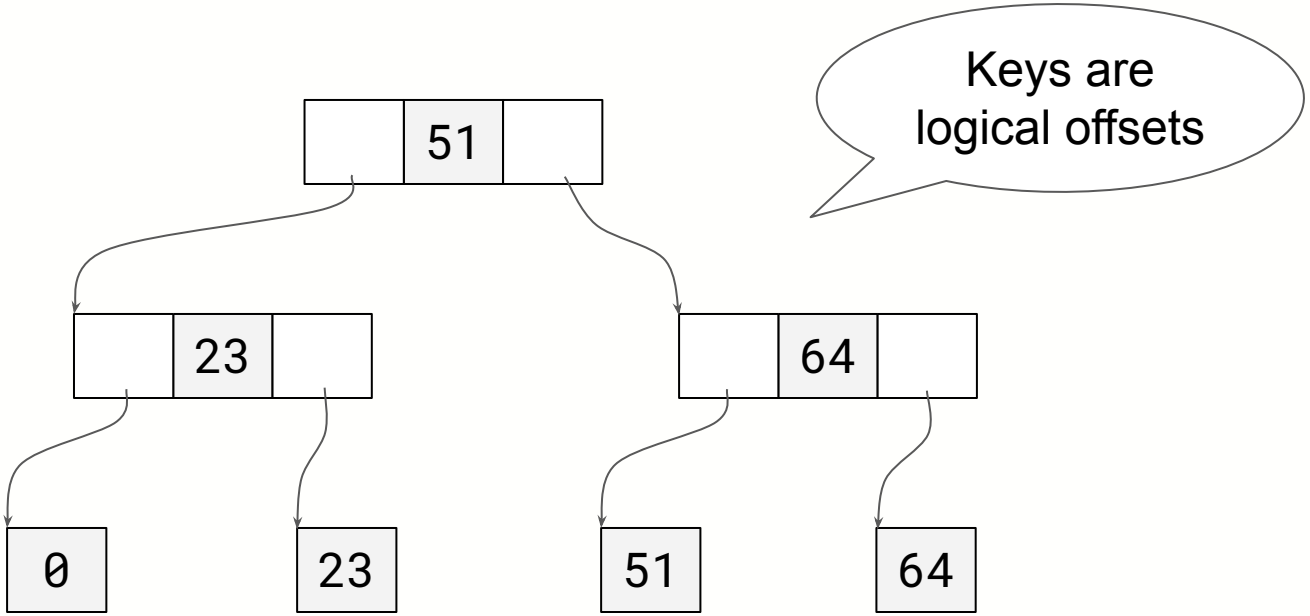
No alignment requirements

Index with efficient shifting



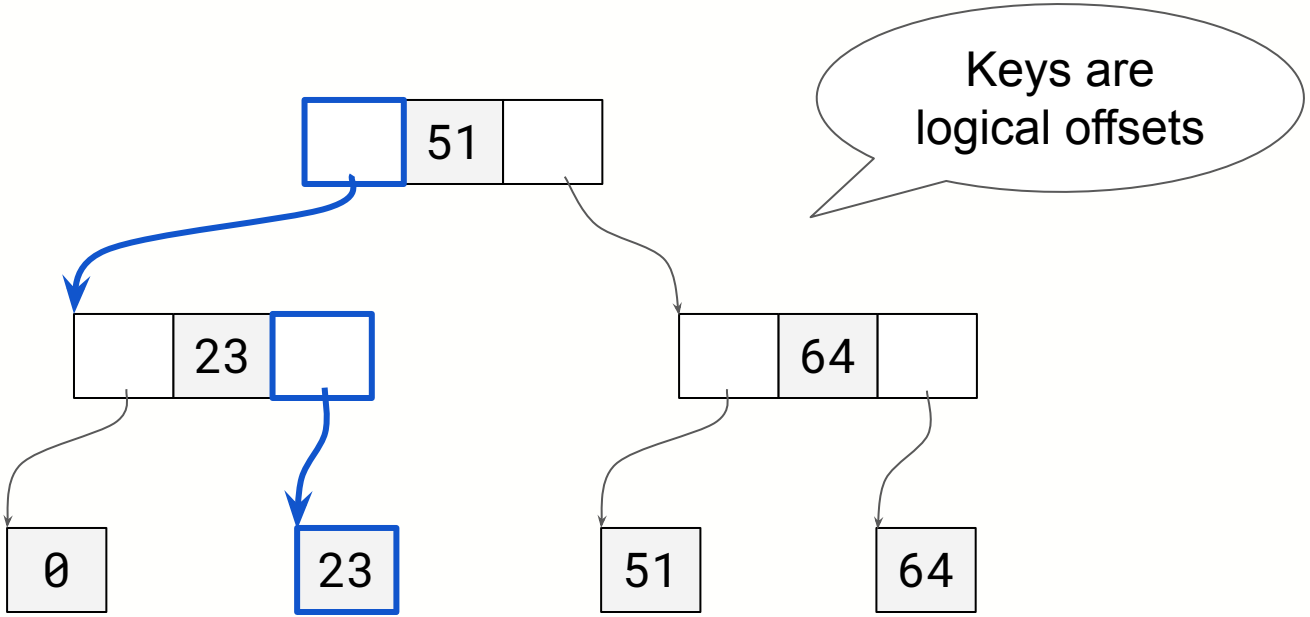
Starting from B$^+$-Tree

- When managing an address space using a B$^+$-Tree



Starting from B$^+$-Tree

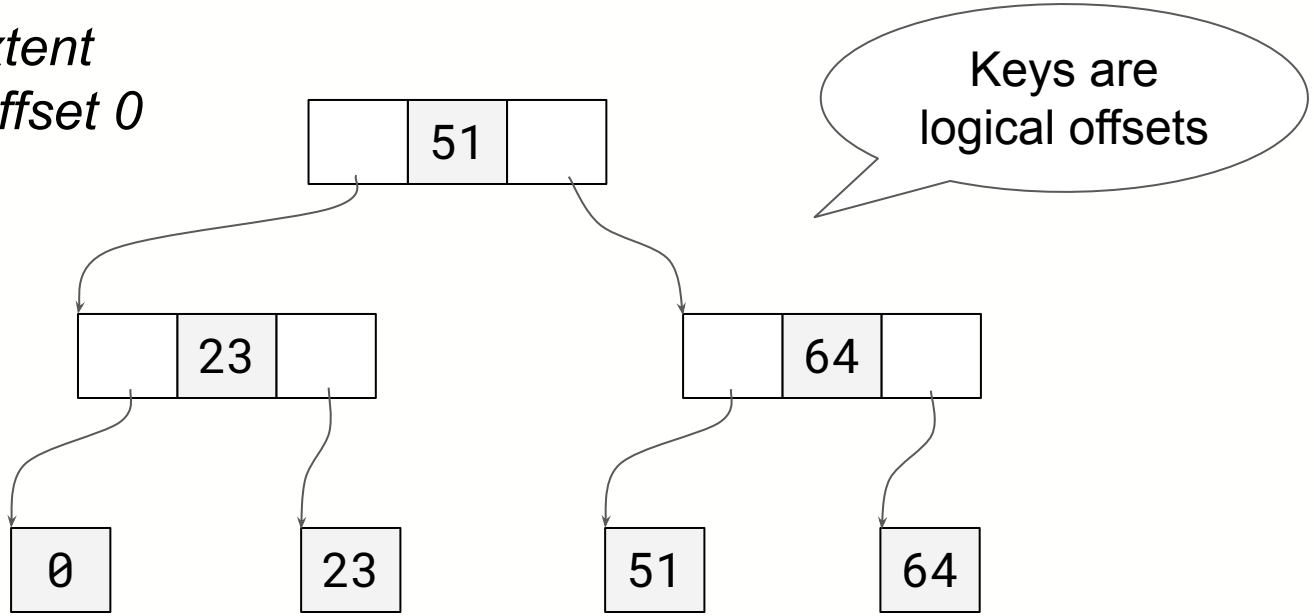
- When managing an address space using a B$^+$-Tree



Starting from B$^+$-Tree

- When managing an address space using a B$^+$-Tree

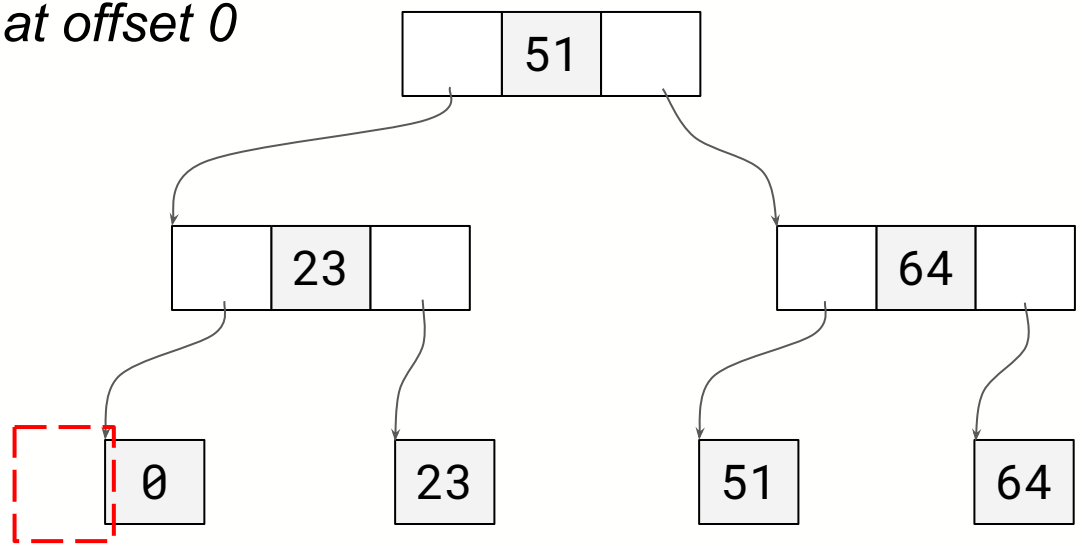
Insert a new extent (length = 10) at offset 0



Starting from B$^+$-Tree

- When managing an address space using a B$^+$-Tree

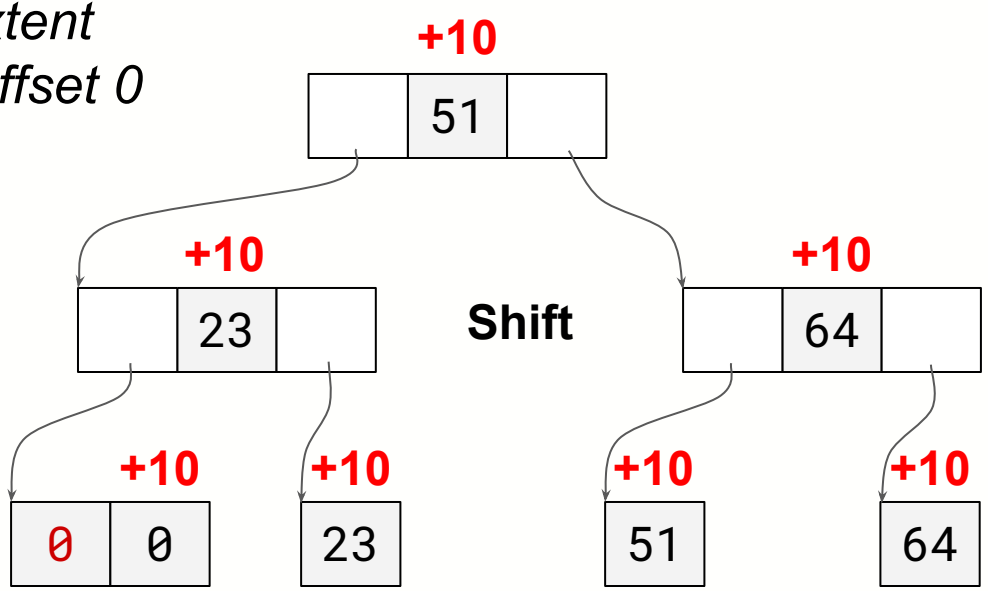
Insert a new extent (length = 10) at offset 0



Starting from B$^+$-Tree

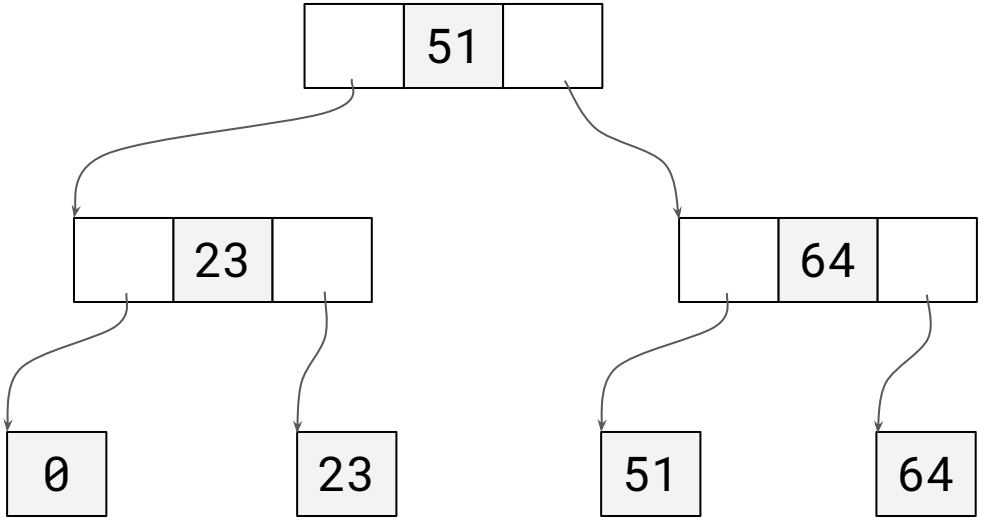
- When managing an address space using a B$^+$-Tree

Insert a new extent (length = 10) at offset 0



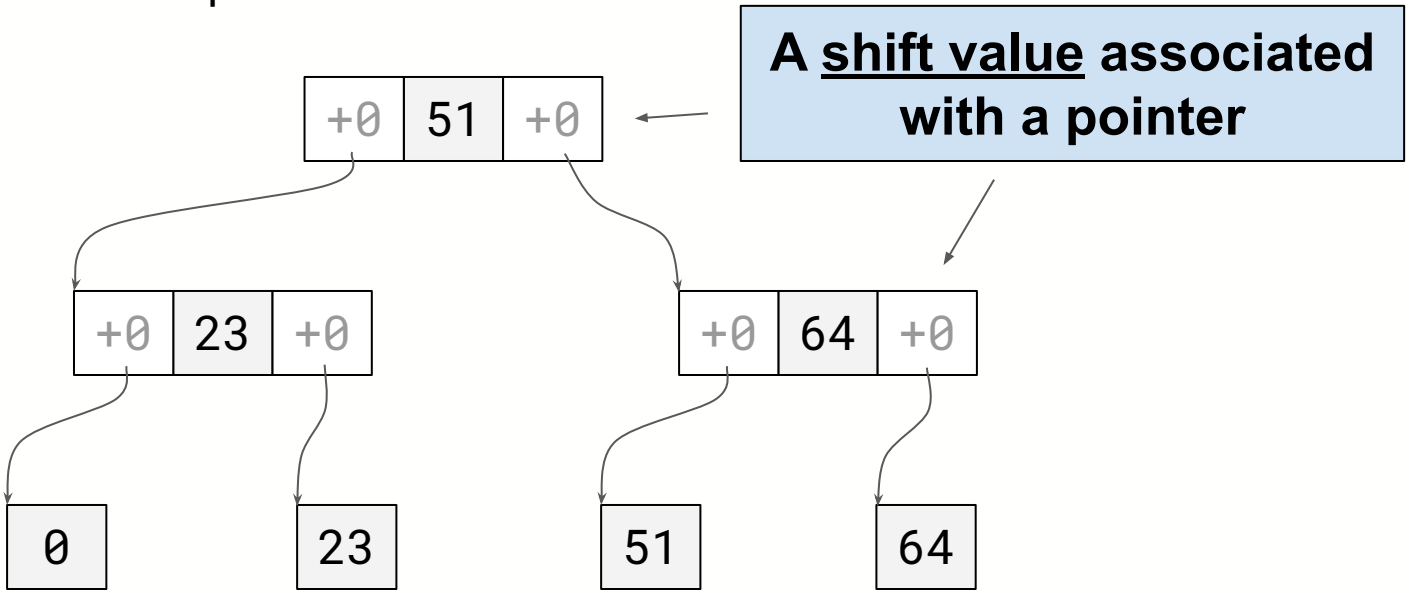
FlexTree: Structure

- An index structure derived from B^+-Tree
  - A new metadata representation scheme



FlexTree: Structure

- An index structure derived from B^+-Tree
  - A new metadata representation scheme

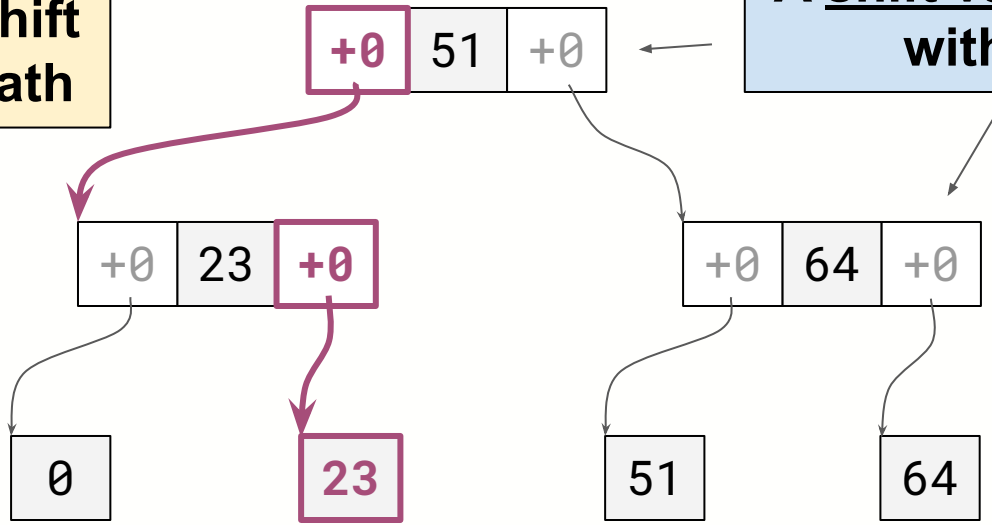


FlexTree: Structure

- An index structure derived from B$^+$-Tree
  - A new metadata representation scheme

Adding all the shift values on the path

A shift value associated with a pointer

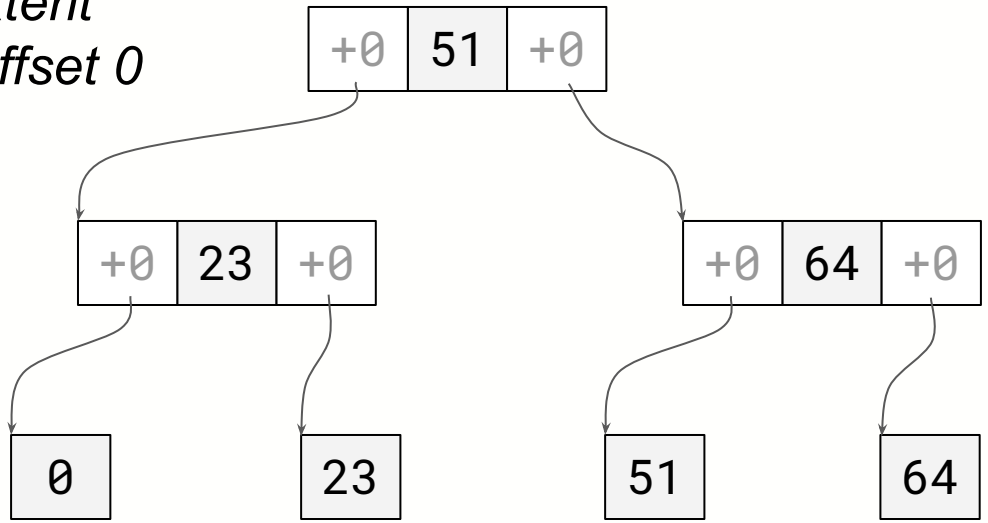


23 + (+0) + (+0) = 23

FlexTree: Operations

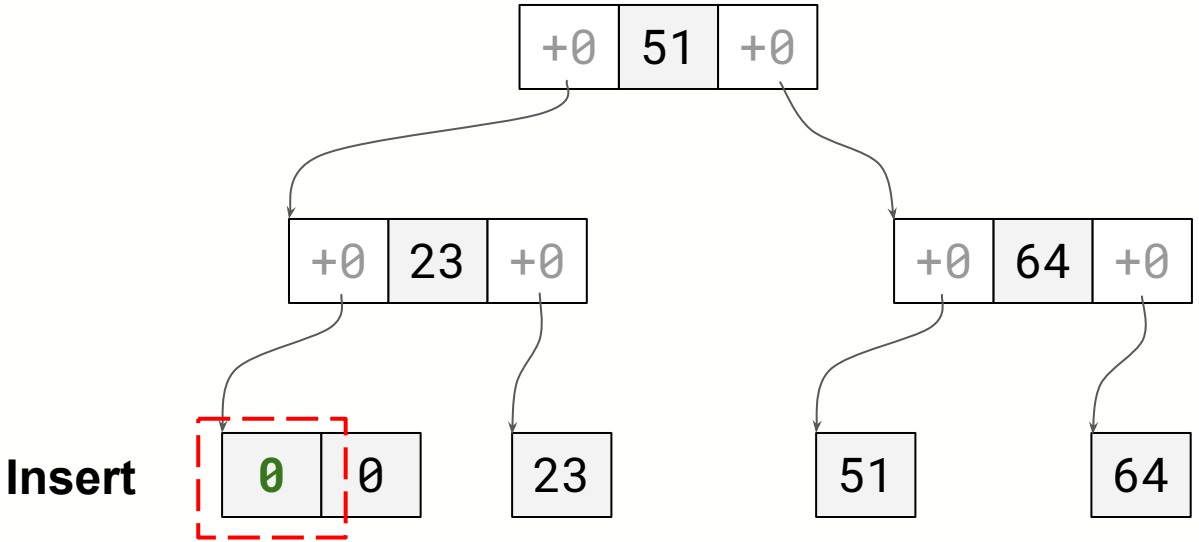
- An index structure derived from B$^+$-Tree
  - A new metadata representation scheme

Insert a new extent (length = 10) at offset 0



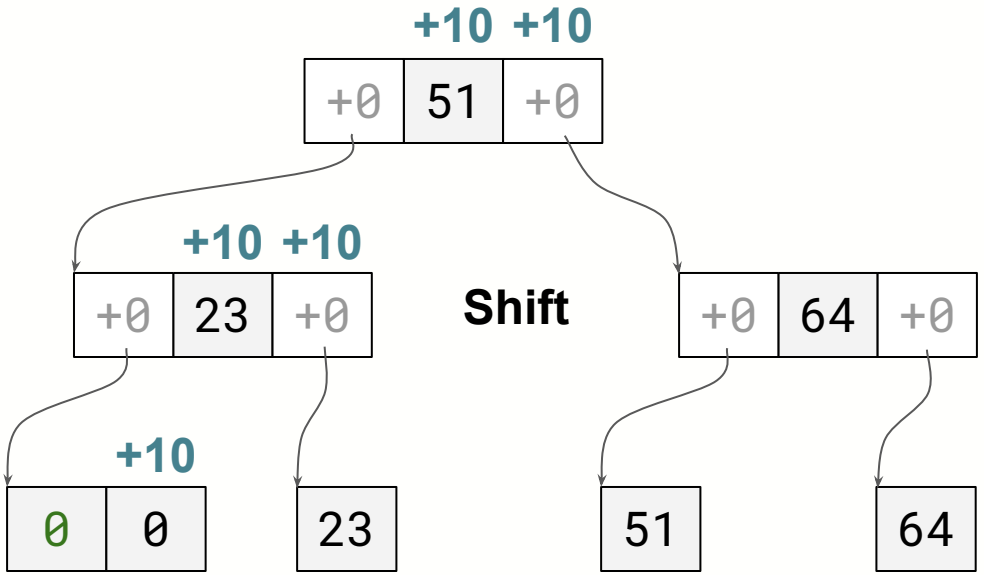
FlexTree: Operations

- An index structure derived from B$^+$-Tree
  - A new metadata representation scheme



FlexTree: Operations

- An index structure derived from B$^+$-Tree
  - A new metadata representation scheme

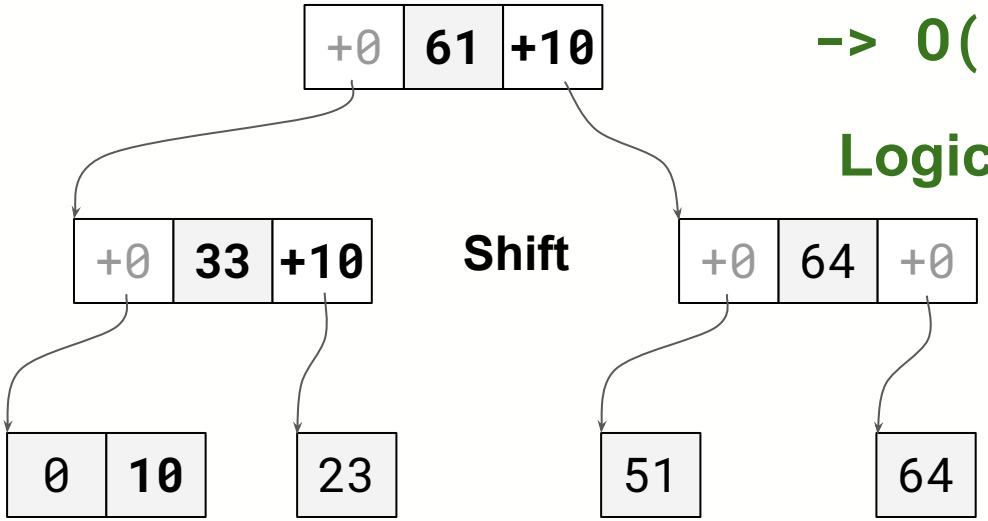


FlexTree: Operations

- An index structure derived from B$^+$-Tree
  - A new metadata representation scheme

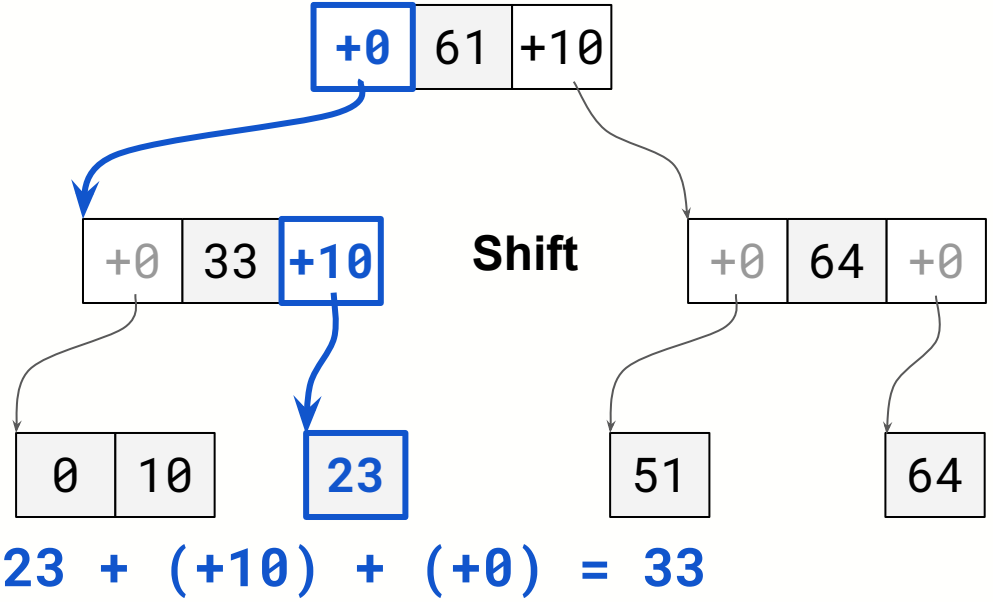
Nodes on the path updated
-> O(logN) cost

Logical logging



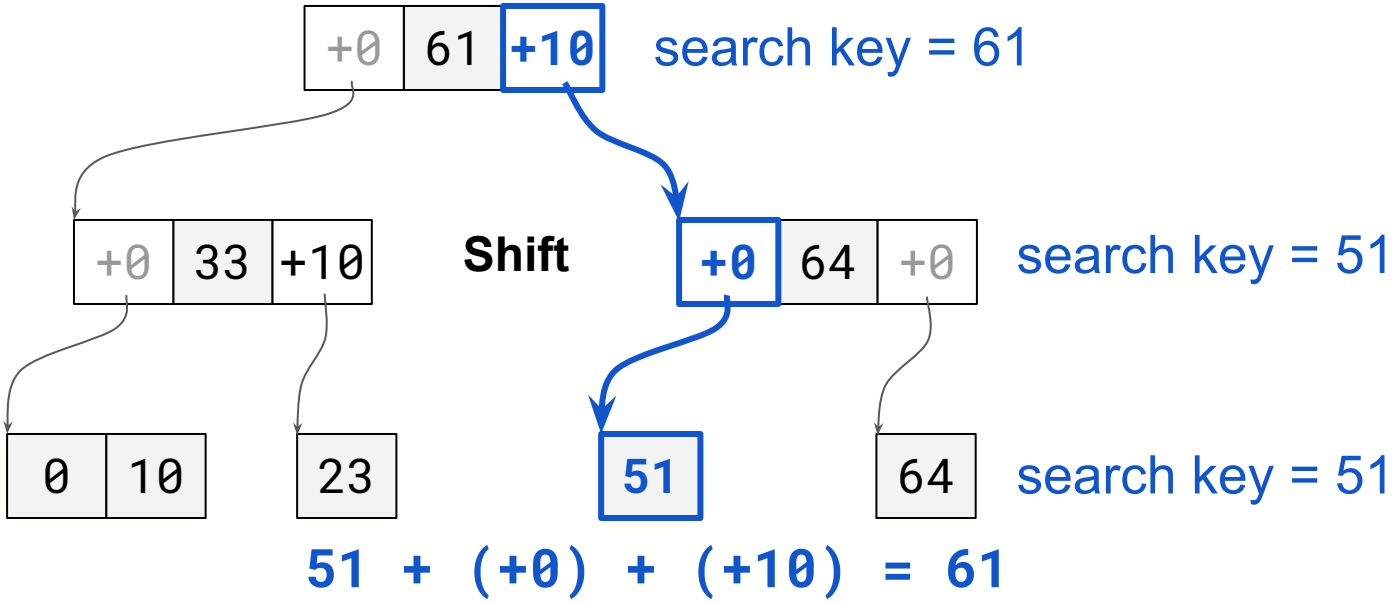
FlexTree: Operations

- An index structure derived from B$^+$-Tree
  - A new metadata representation scheme



FlexTree: Operations

- An index structure derived from B^+-Tree
  - A new metadata representation scheme



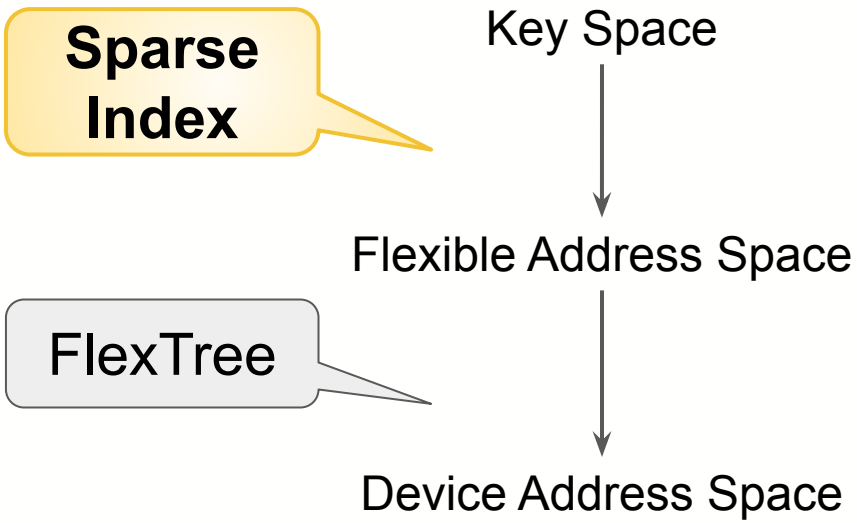
Based on FlexTree

- FlexSpace: Log-structured data storage indexed by FlexTree
  - Supporting read/write/insert-range operations etc.

- FlexDB: Keeping all KV pairs sorted in a FlexSpace

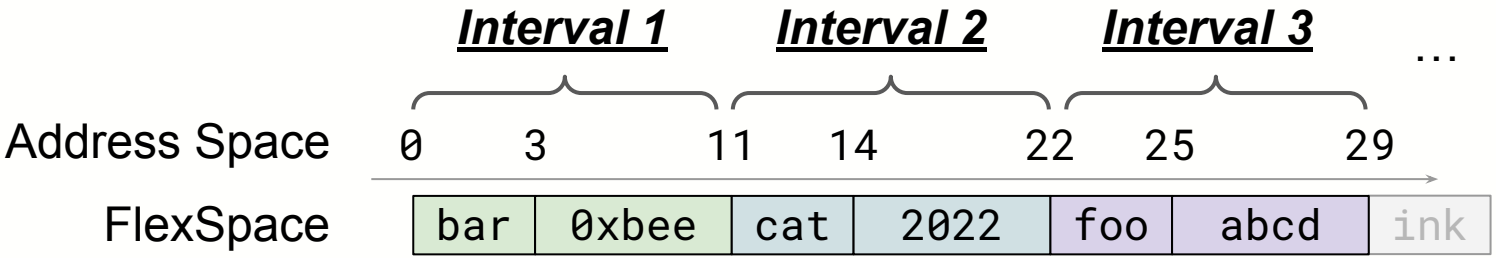
FlexDB

- Keeping all KV pairs sorted in a flexible address space



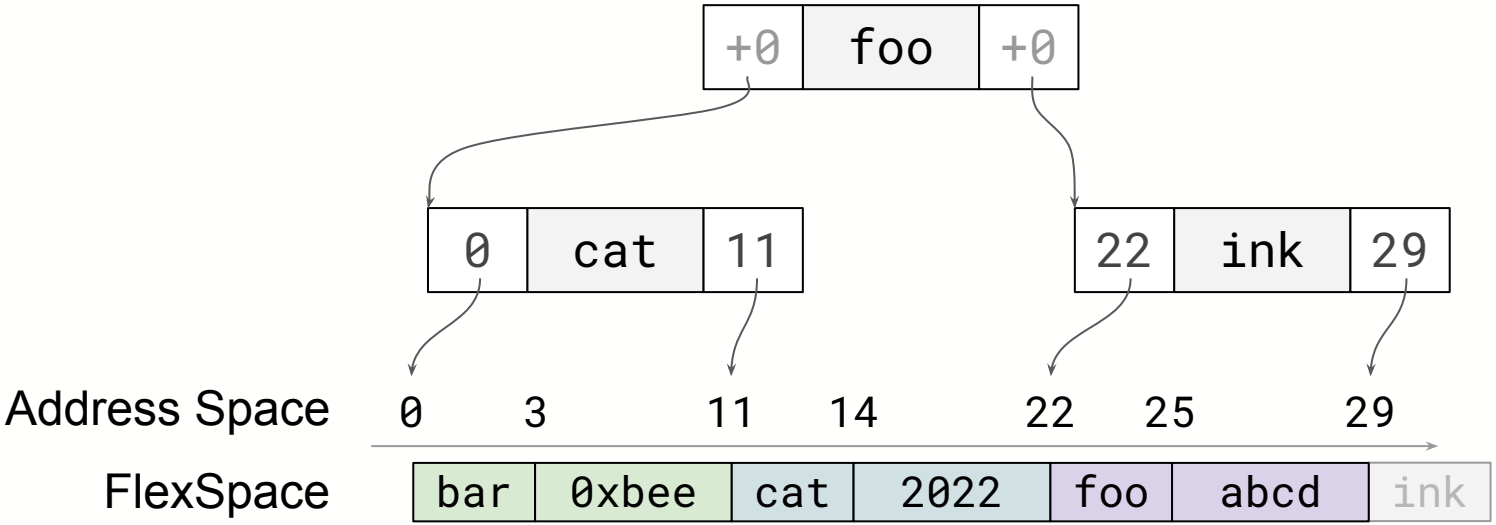
FlexDB: Sparse Index

- Managing sorted keys in *intervals*



FlexDB: Sparse Index

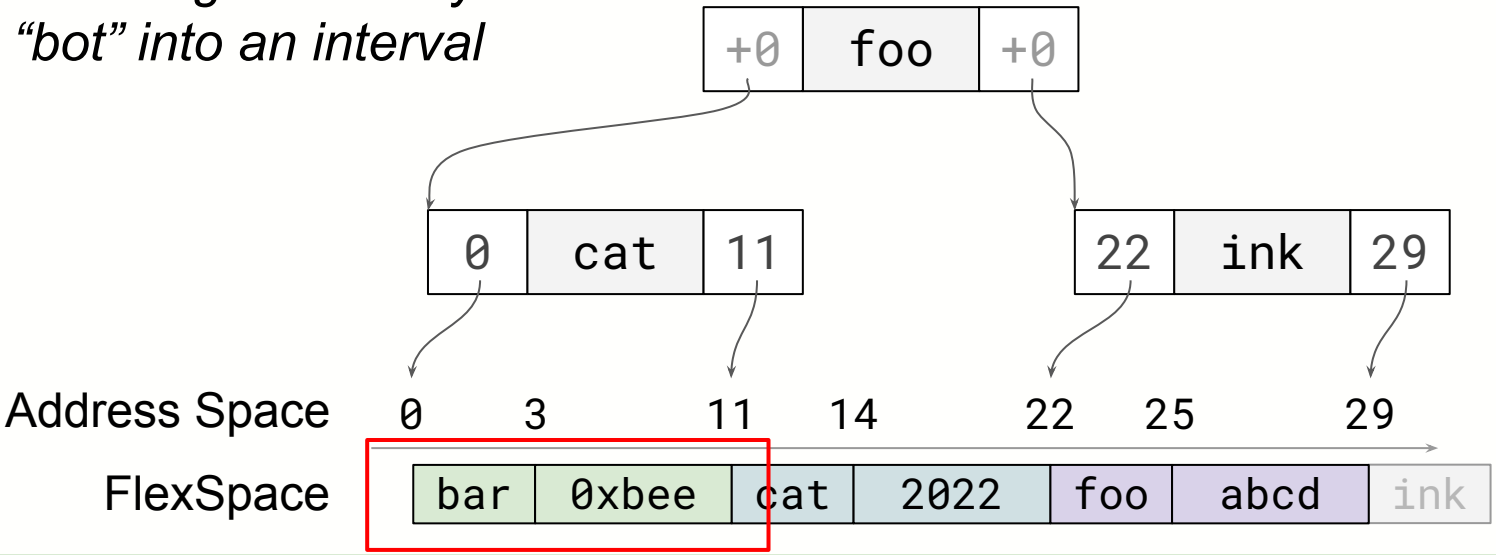
- Managing sorted keys in *intervals*
- Indexing the first key (*anchor*) in each interval



FlexDB: Sparse Index

- Managing sorted keys in *intervals*
- Indexing the first key (*anchor*) in each interval

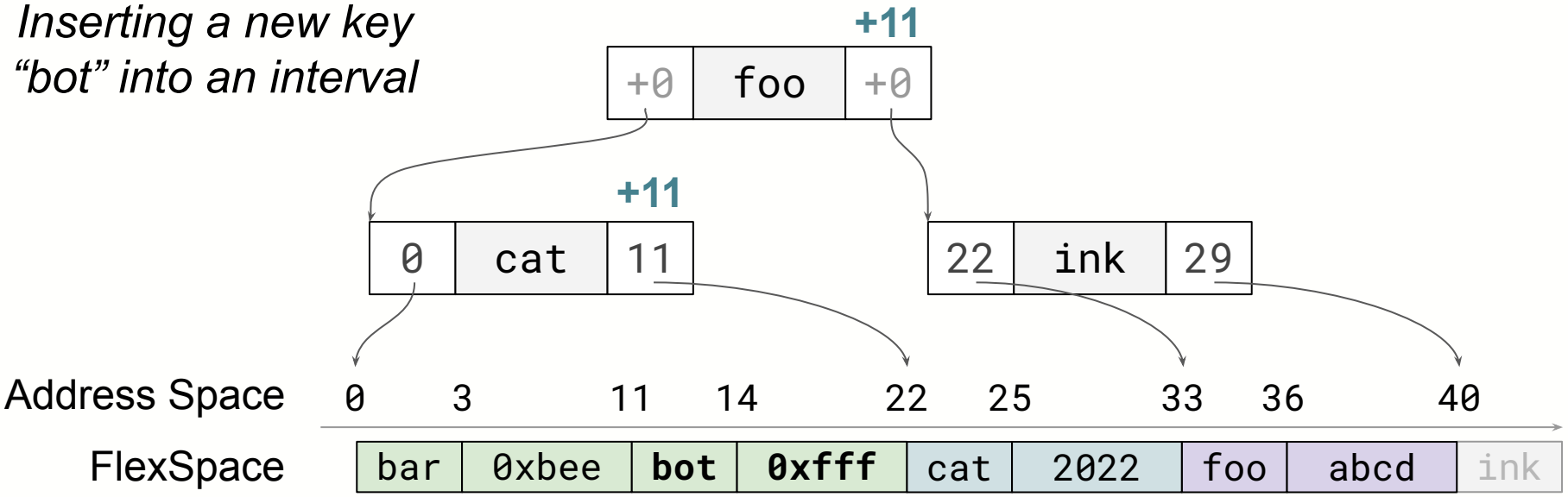
Inserting a new key “bot” into an interval



FlexDB: Sparse Index

- Managing sorted keys in *intervals*
- Indexing the first key (*anchor*) in each interval

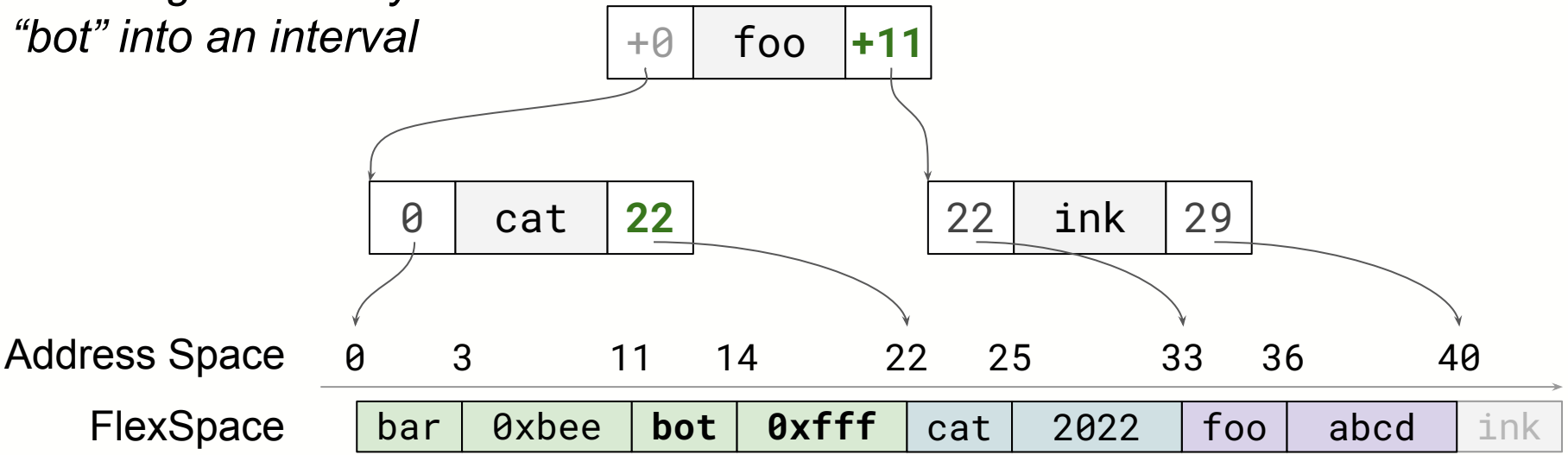
Inserting a new key “bot” into an interval



FlexDB: Sparse Index

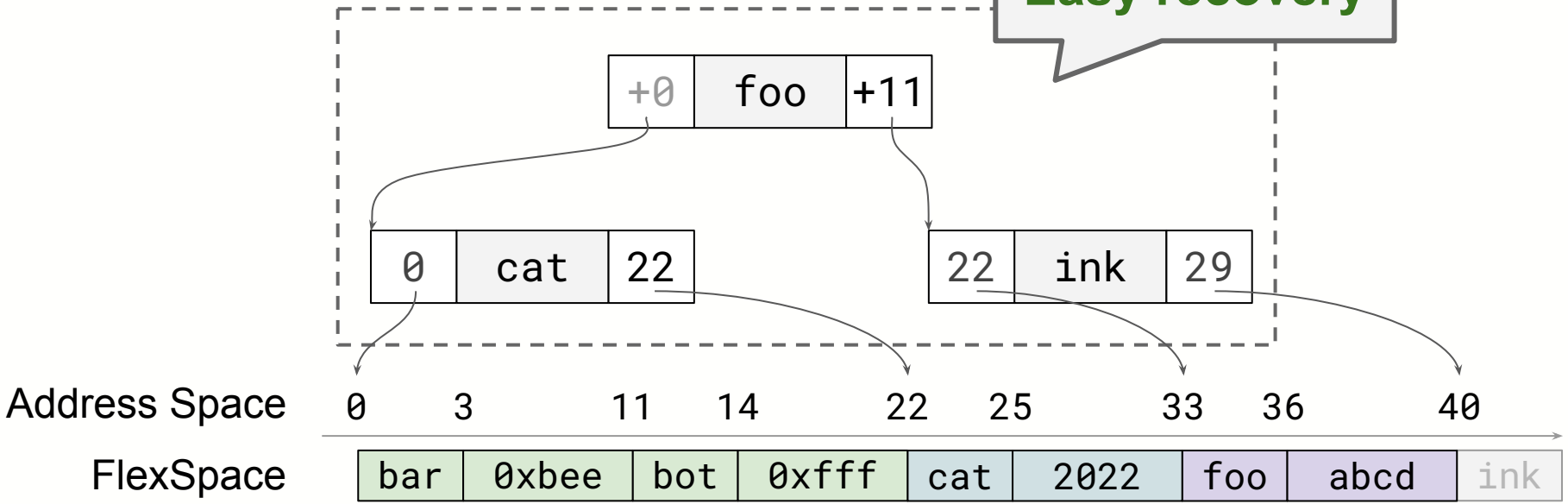
- Managing sorted keys in *intervals*
- Indexing the first key (*anchor*) in each interval

Inserting a new key “bot” into an interval



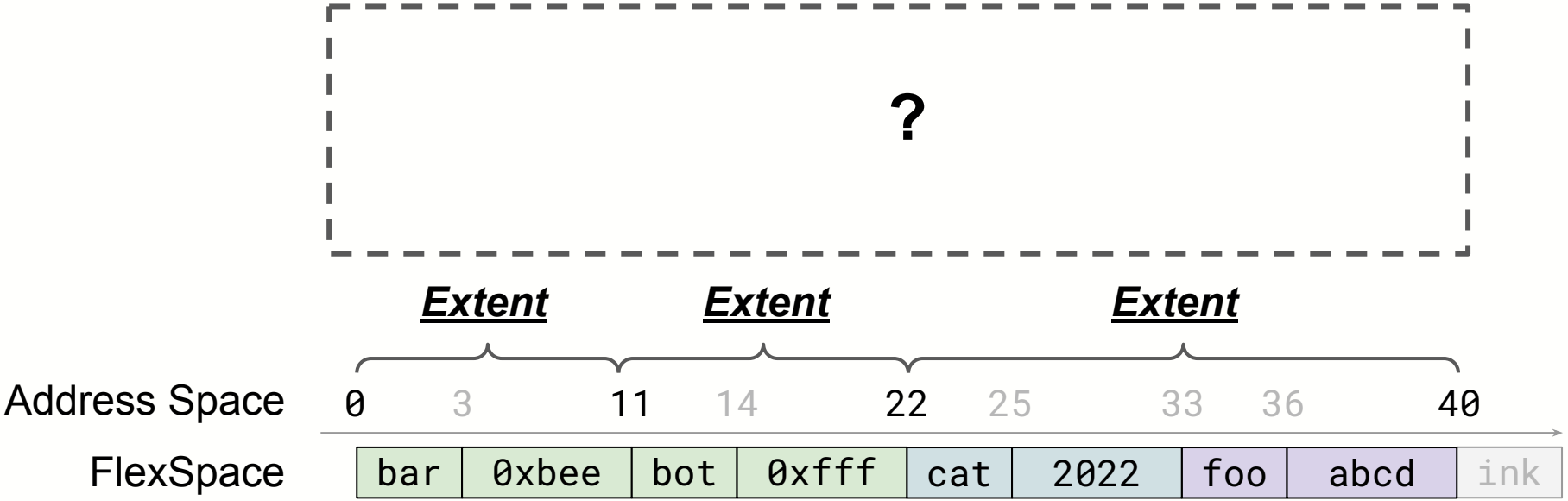
FlexDB: Sparse Index

- Managing sorted keys in *intervals*
- Indexing the first key (*anchor*) in each in



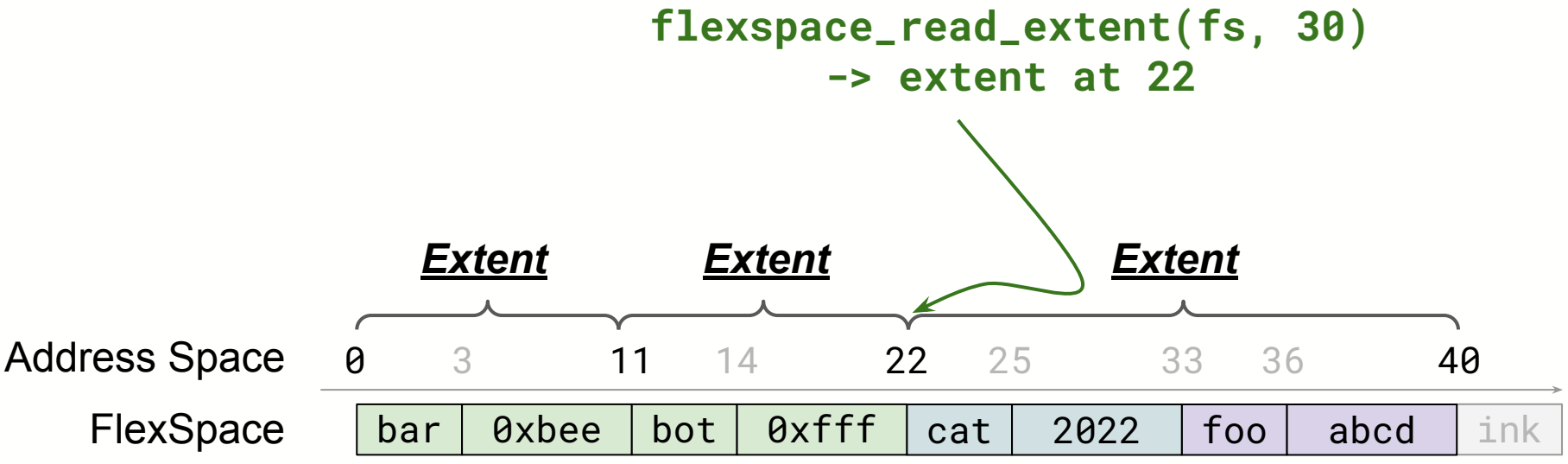
FlexDB: Sparse Index

- Managing sorted keys in *intervals*
- Indexing the first key (*anchor*) in each interval



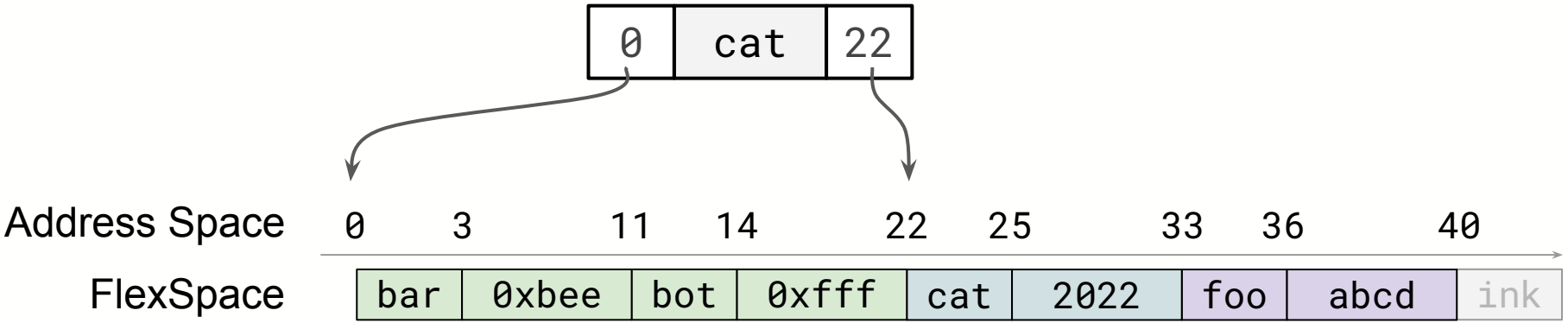
FlexDB: Sparse Index

- Managing sorted keys in *intervals*
- Indexing the first key (*anchor*) in each interval



FlexDB: Sparse Index

- Managing sorted keys in *intervals*
- Indexing the first key (*anchor*) in each interval



Recap

- FlexTree enables lightweight data insertions in a flexible address space.
- FlexDB manages sorted data without using extra persistent indirections.

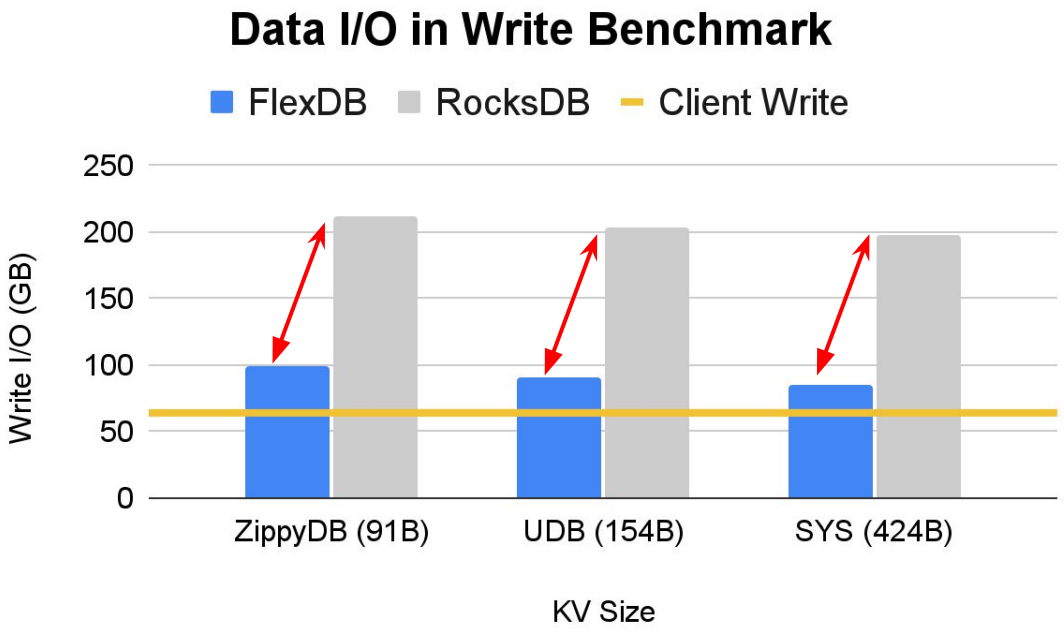
Evaluation: Setup

- Intel Xeon Silver 4210 w/ 10 cores
- 64GB RAM
- Optane 905P SSD
- Key-Value Sizes:
  - ZippyDB: 91 bytes *
  - UDB: 154 bytes *
  - SYS: 424 bytes **
- 4 Client Threads

* Zhichao Cao, Siying Dong, Sagar Vemuri, and David H. C. Du. “Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook”. In: 18th USENIX Conference on File and Storage Technolo gies (FAST’20). 2020, pp. 209–223.
** Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. “Workload Analysis of a Large-Scale Key-Value Store”. In: SIGMETRICS Per form. Eval. Rev. 40.1 (2012), pp. 53–64.

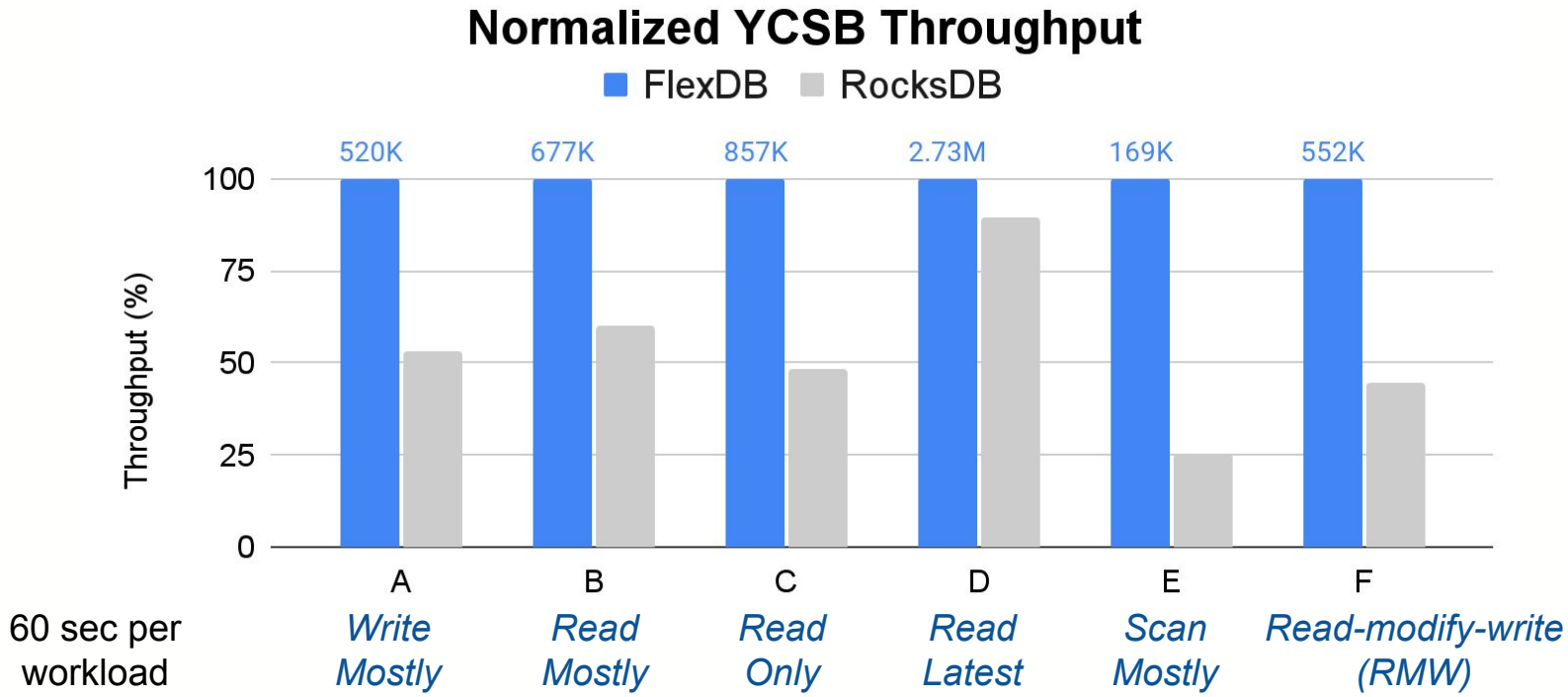
Evaluation: Disk I/O

- Write 64GB into an empty store; Zipfian distribution



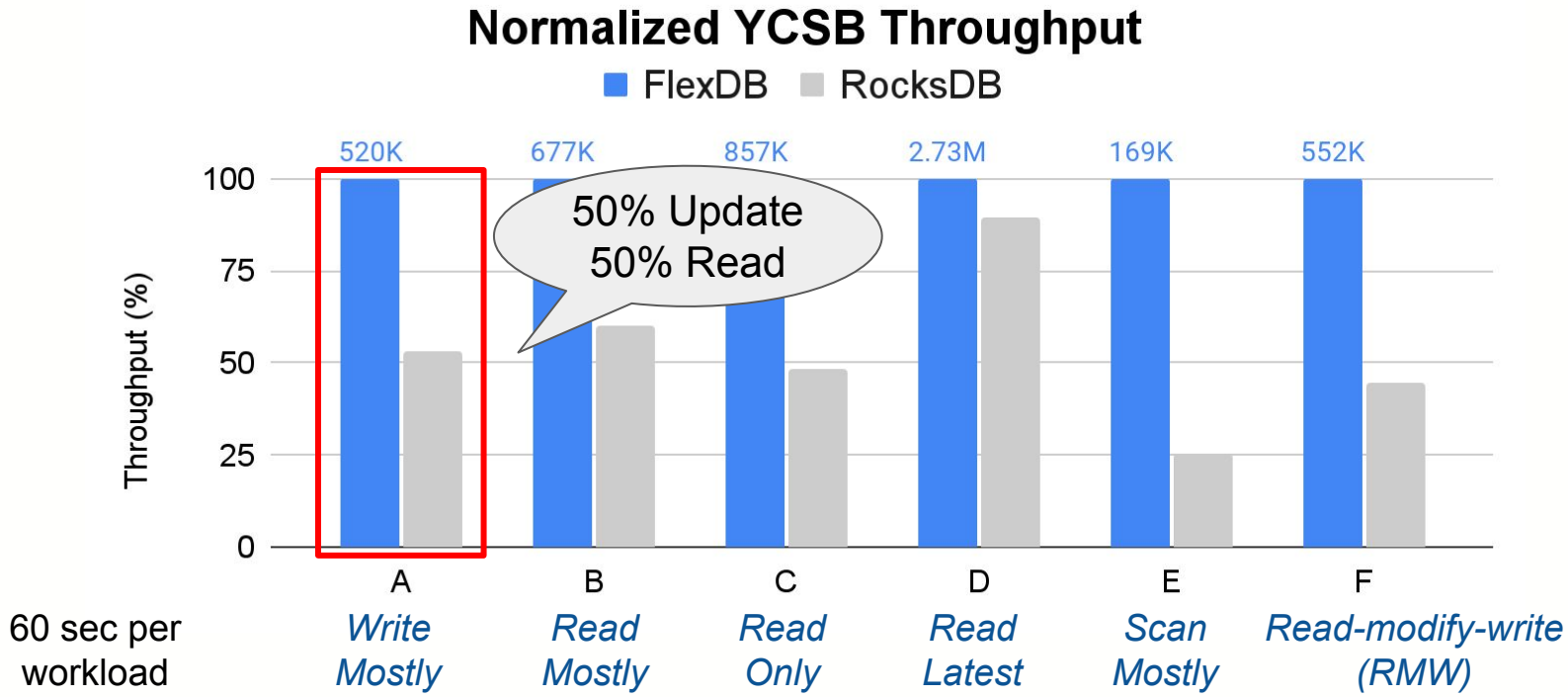
Evaluation: YCSB

- Starting from a 500GB UDB store; Zipfian distribution



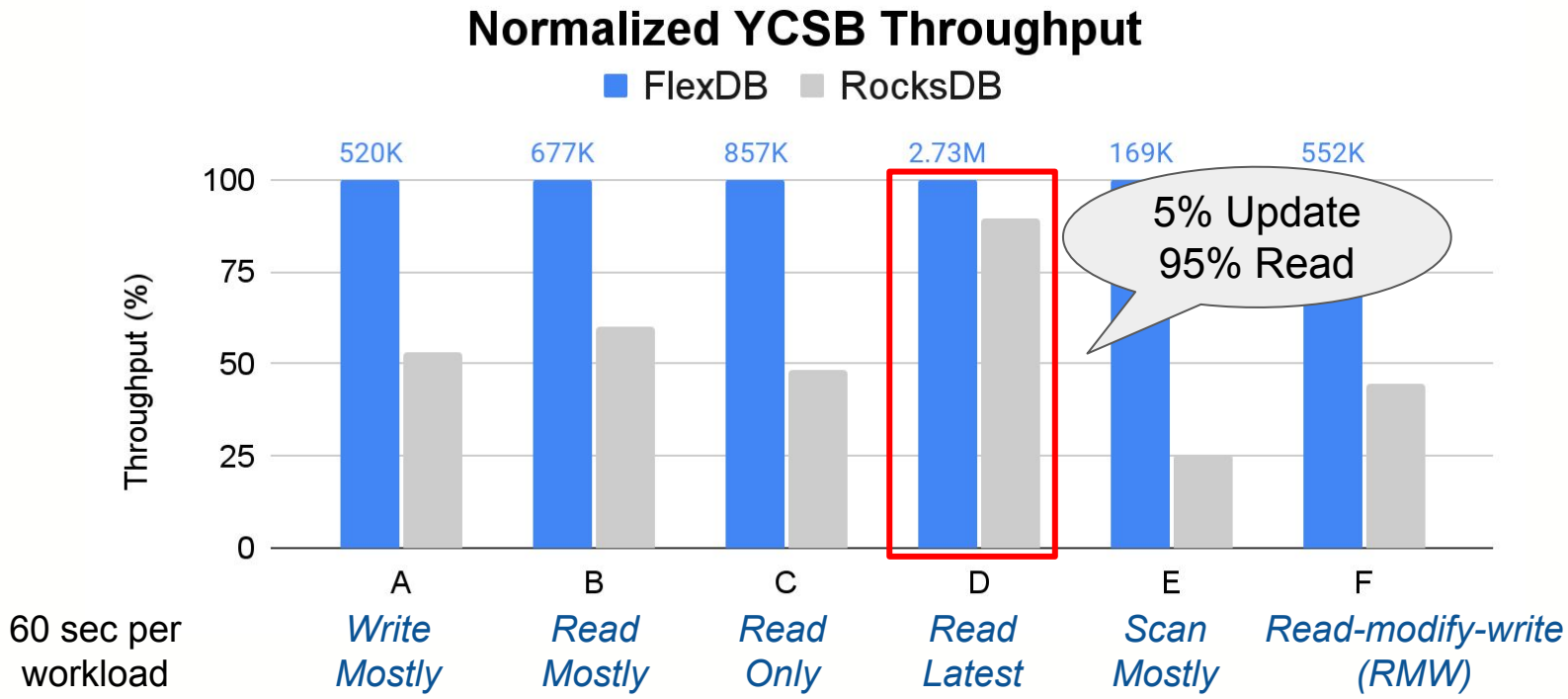
Evaluation: YCSB

- Starting from a 500GB UDB store; Zipfian distribution



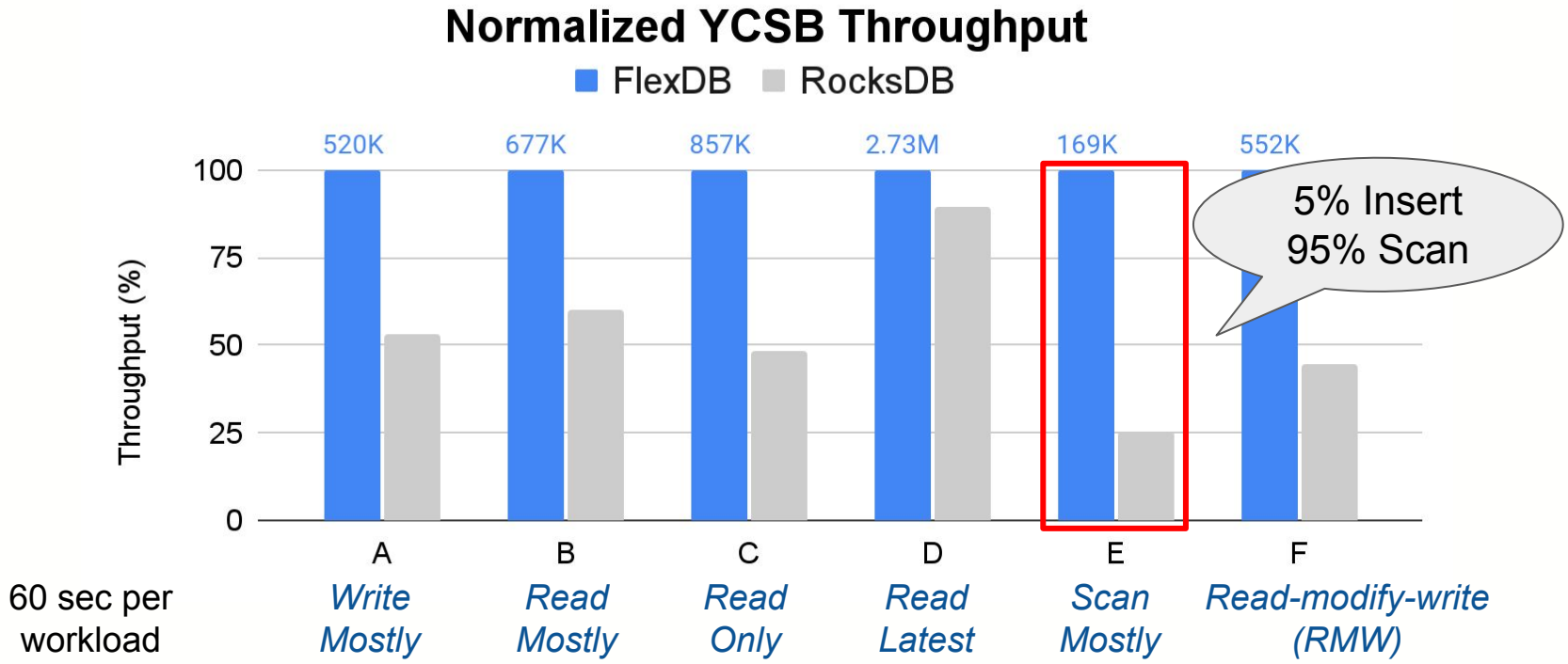
Evaluation: YCSB

- Starting from a 500GB UDB store; Zipfian distribution



Evaluation: YCSB

- Starting from a 500GB UDB store; Zipfian distribution



Summary

- Flexible address space enables lightweight data management.
- FlexDB achieves low WA and high throughput.

- The code of this project is available at:
  https://github.com/flexible-address-space/flexspace