Index Structure Journey 1st Week
Traditional Index Structure Overview

Presented by Hojin Shin
System Software Lab.

DANKOOK UNIVERSITY

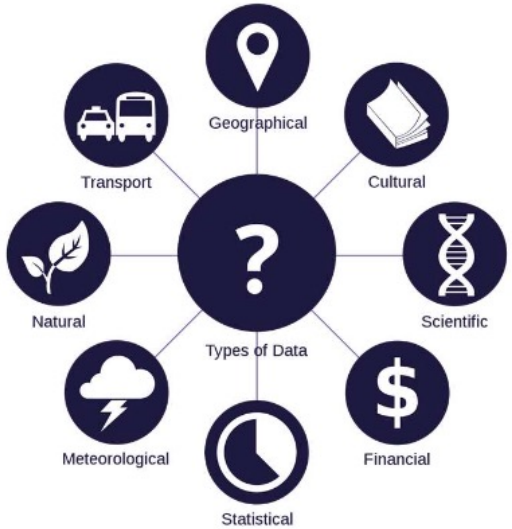
Dankook University
System Software Laboratory

Contents

Introduction:
Data and In-memory Index

Introduction: Data and In-memory Index

- What is data?
  - Units of information, often numeric, that are collected through observation
  - Fact on which a theory is based
  - Data in the form of letters, numbers, sounds, pictures that a computer can process



Introduction: Data and In-memory Index

- What is BigData?
  - A large amount of structured data that exceed existing DB management tools
  - Set of unstructured data that is not in the form of data

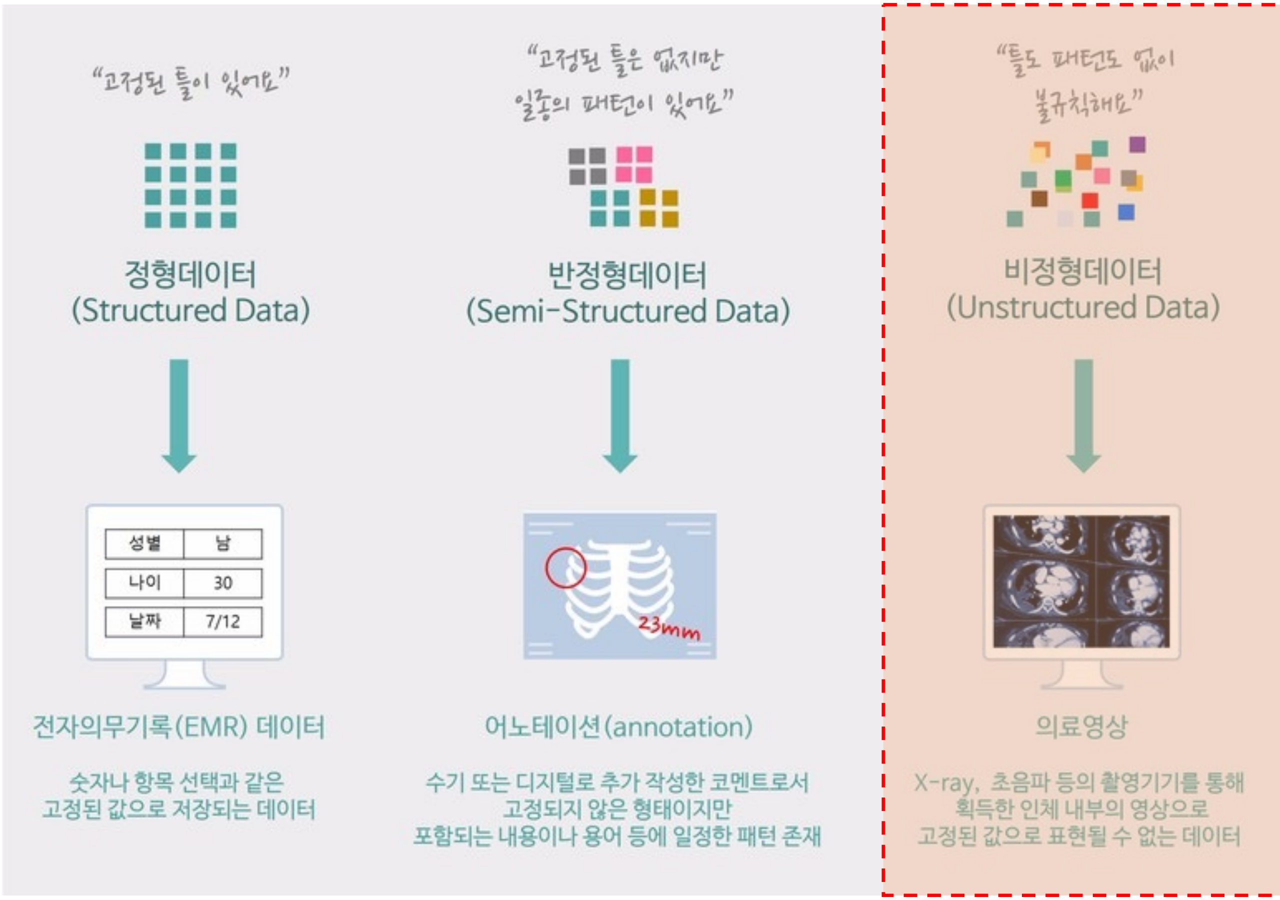
BigData 5V

Volume	Variety	Velocity	Veracity	Value
Data's size	Structured Unstructured	Speed	Accuracy	Meaningful

Introduction: Data and In-memory Index

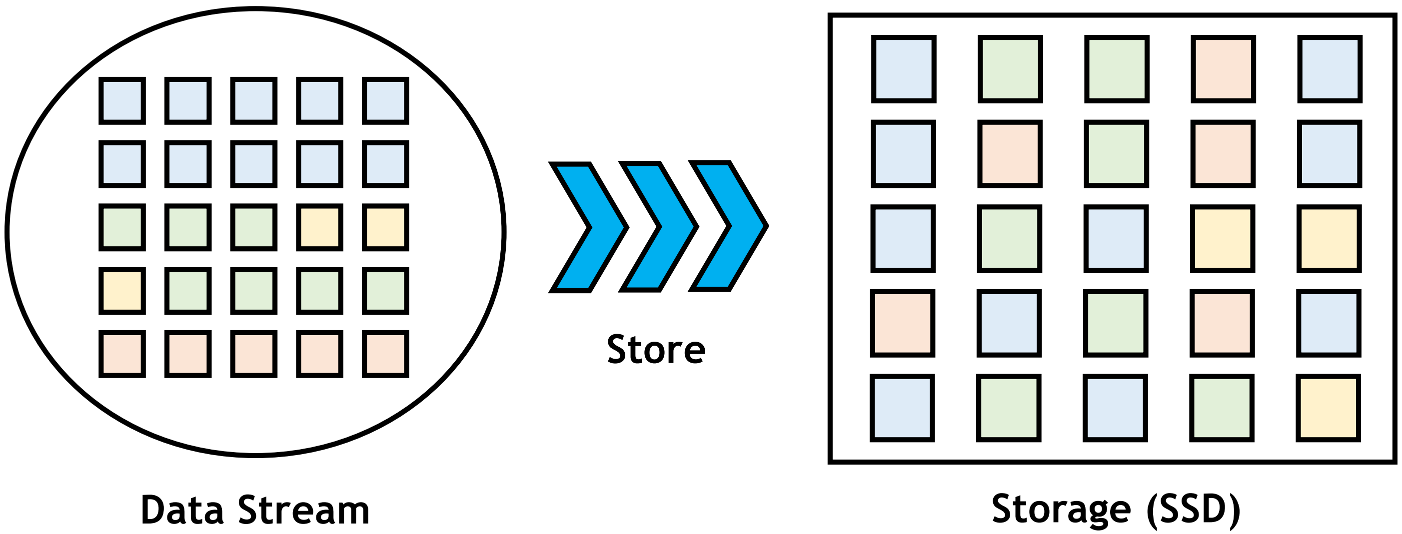
- Kind of Data
  - Structured Data
    - Data organized and processed into a form suitable for immediate statistical analysis
    - Data stored in fixed fields
  - Unstructured Data
    - One piece of data, not a set of data, is objectified as collected data
    - Difficult to understand the meaning of a value because there is no set rule
  - Semi-structured Data
    - File type, metadata (schema of structured data inside data)

Introduction: Data and In-memory Index



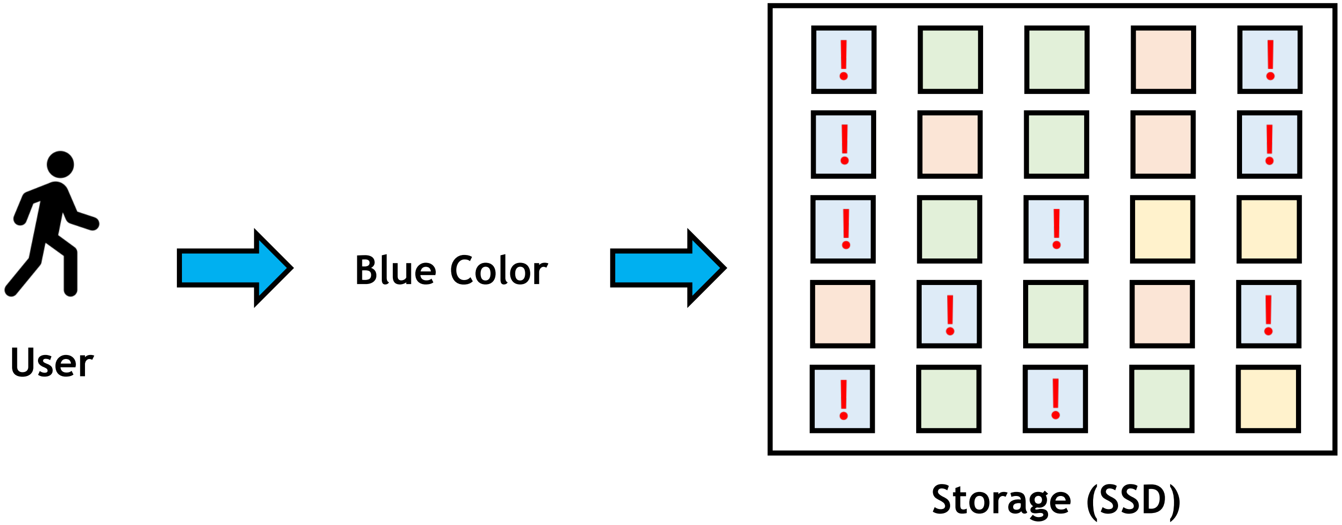
Introduction: Data and In-memory Index

- What is In-memory Index?



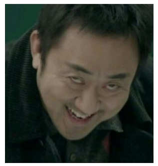
Introduction: Data and In-memory Index

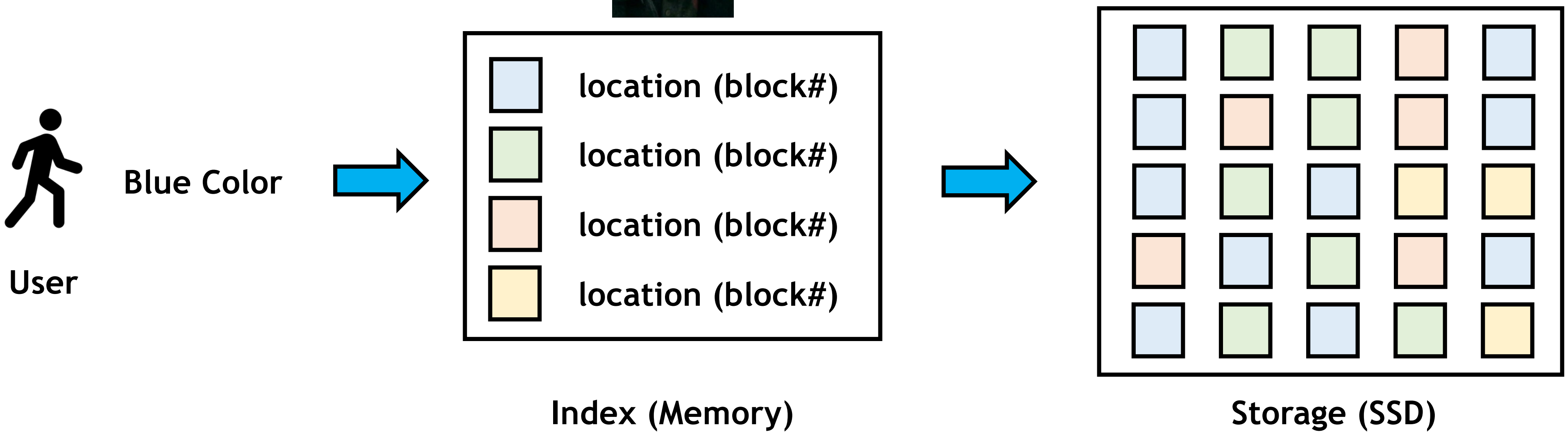
- What is In-memory Index?



Introduction: Data and In-memory Index

- What is In-memory Index?





Introduction: Data and In-memory Index

- The reason to use In-memory indexing
  - Memory has a low latency for processing requests
    - If we process each request to disk, it goes slow down
    - DRAM: 100ns < SATA SSD: ~70us
  - To effectively index the input data
  - To reduce disk-based I/O

Introduction: Data and In-memory Index

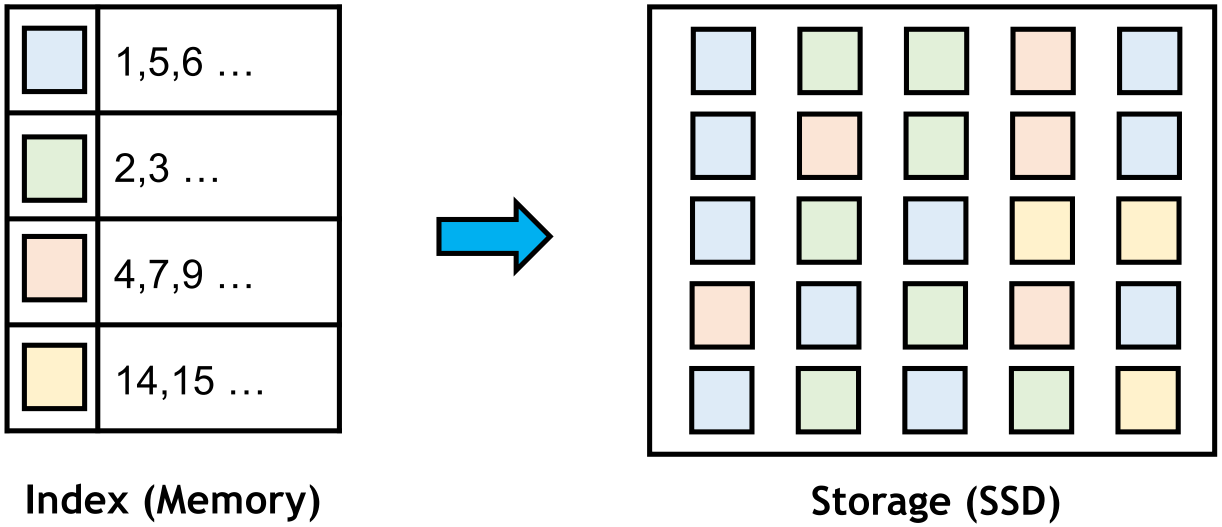
- The reason to use In-memory indexing
  - Memory has a low latency for processing requests
    - If we process each request to disk, it goes slow down
    - DRAM: 100ns < SATA SSD: ~70us
  - To effectively index the input data
  - To reduce disk-based I/O

```
        0.5 ns - CPU L1 dCACHE reference
          1  ns - speed-of-light (a photon) travel a 1 ft (30.5cm) distance
          5  ns - CPU L1 iCACHE Branch mispredict
          7  ns - CPU L2  CACHE reference
         71  ns - CPU cross-QPI/NUMA best  case on XEON E5-46*
        100  ns - MUTEX lock/unlock
        100  ns - own DDR MEMORY reference
        135  ns - CPU cross-QPI/NUMA best  case on XEON E7-*
        202  ns - CPU cross-QPI/NUMA worst case on XEON E7-*
        325  ns - CPU cross-QPI/NUMA worst case on XEON E5-46*
     10,000  ns - Compress 1K bytes with Zippy PROCESS
     20,000  ns - Send 2K bytes over 1 Gbps NETWORK
    250,000  ns - Read 1 MB sequentially from MEMORY
    500,000  ns - Round trip within a same DataCenter
 10,000,000  ns - DISK seek
 10,000,000  ns - Read 1 MB sequentially from NETWORK
 30,000,000  ns - Read 1 MB sequentially from DISK
150,000,000  ns - Send a NETWORK packet CA -> Netherlands
|   |   |   |
|   |   | ns|
|   | us|
| ms|
```

Various In-memory Index

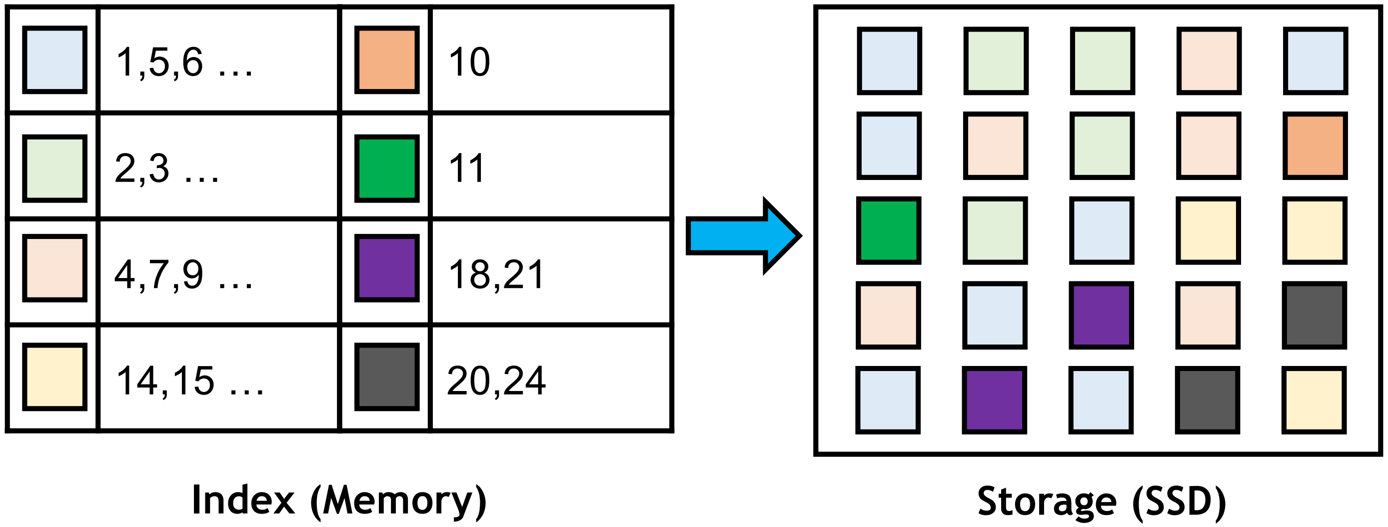
What I've seen: Observation

- Why do various in-memory indexes exist?
  - Aren't arrays enough?
    - Problems arise when there is a lot of data!



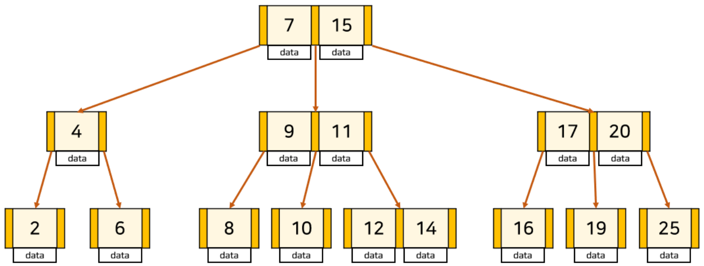
What I’ve seen: Observation

- Why do various in-memory indexes exist?
  - Aren’t arrays enough?
    - Problems arise when there is a lot of data!

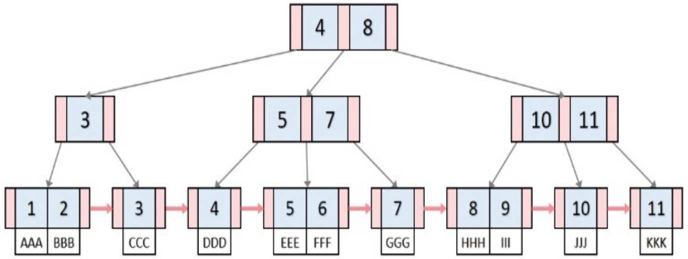


What I've seen: Observation

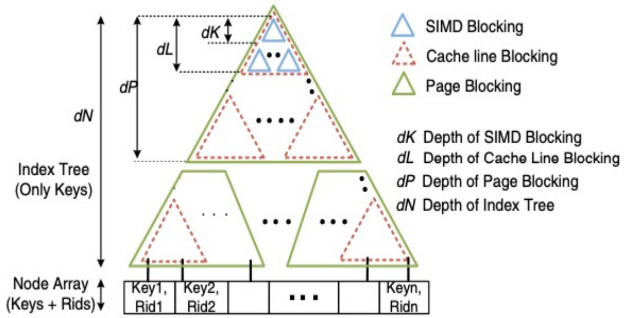
- In-memory structure: Tree-based
  - B-Tree, B+-Tree, FAST (Fast Architecture Sensitive Tree)
  - AVL Tree, RB-Tree



B-Tree[1]



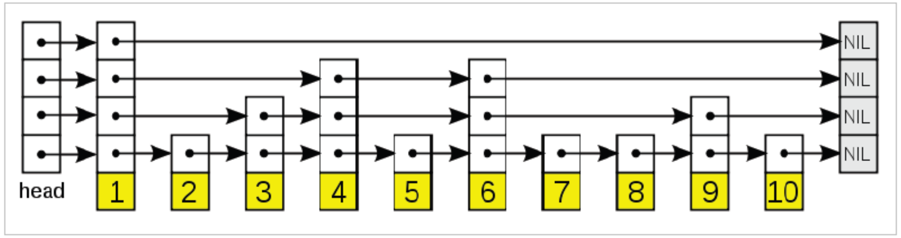
B+-Tree[2]



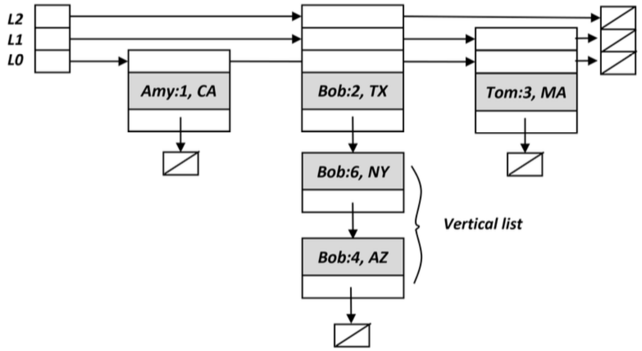
FAST[3]

What I've seen: Observation

- In-memory structure: List-based
  - Skiplist, JellyFish
  - No Hot Spot Skiplist, NUMA Skiplist



Skiplist[4]



JellyFish[5]

What I've seen: Observation

- In-memory structure: List-based
  - Skiplist, JellyFish
  - No Hot Spot Skiplist, NUMA Skiplist

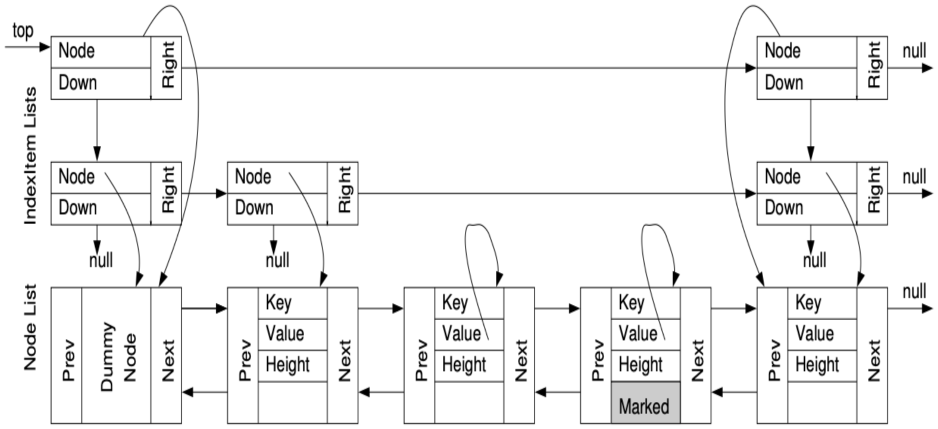
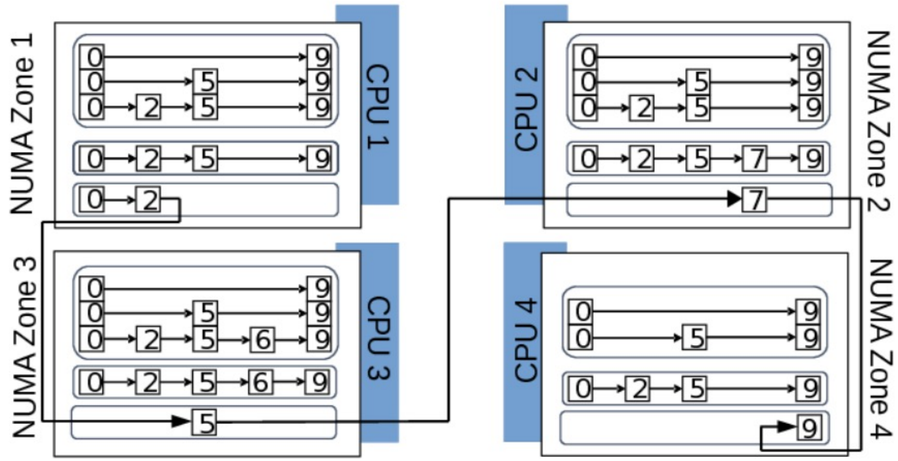


Fig. 2. The contention-friendly non blocking skip list structure

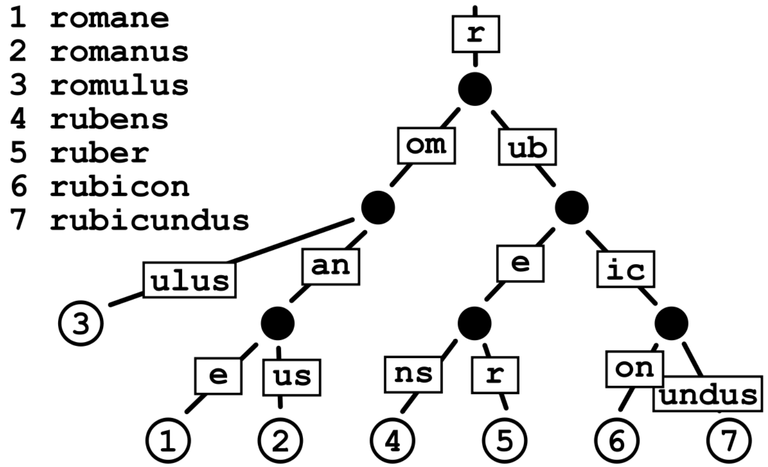
NHS Skiplist[6]



NUMA Skiplist[7]

What I've seen: Observation

- In-memory structure: Trie-based
  - Radix Tree, ART (Adaptive Radix Tree)



Radix Tree

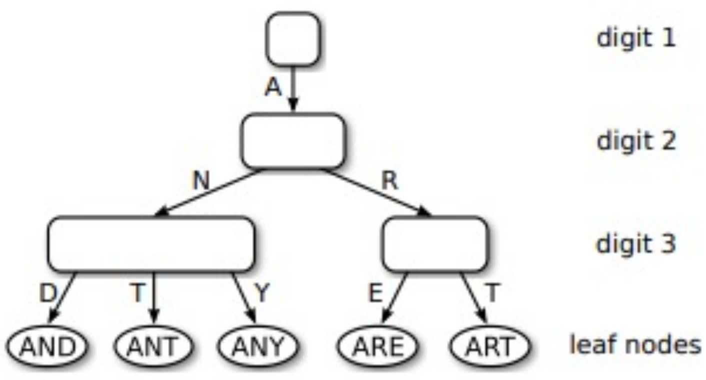
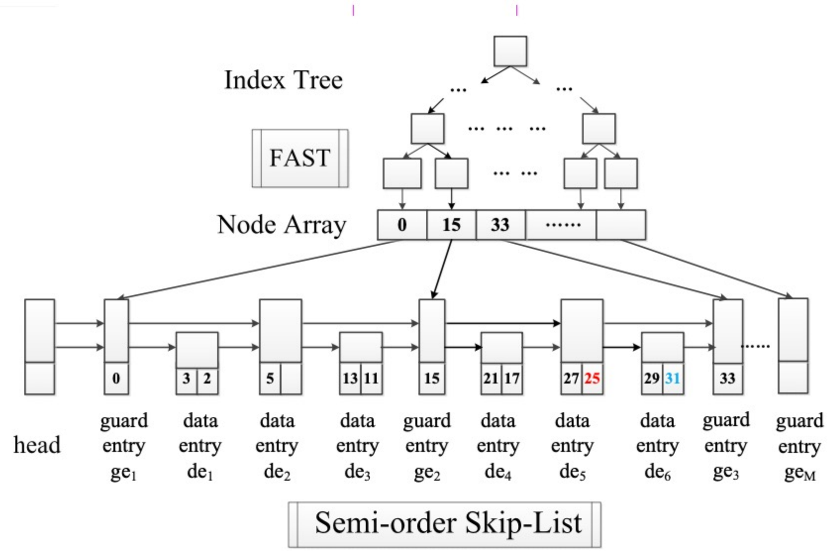


Fig. 1. Adaptively sized nodes in our radix tree.

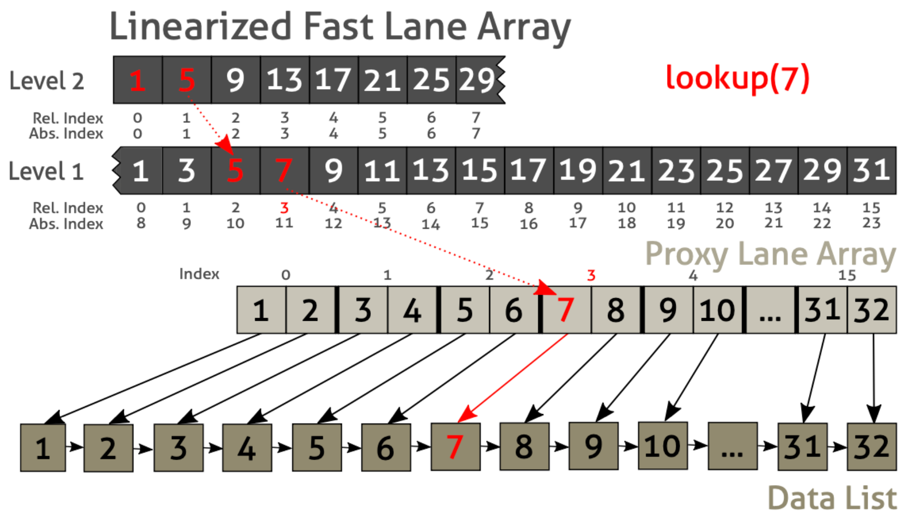
Adaptive Radix Tree[8]

What I’ve seen: Observation

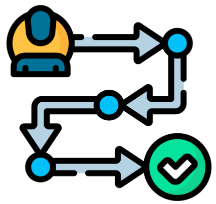
- In-memory structure: Hybrid
  - S3, CSSL (Cache Sensitive Skip List)



S3[9]



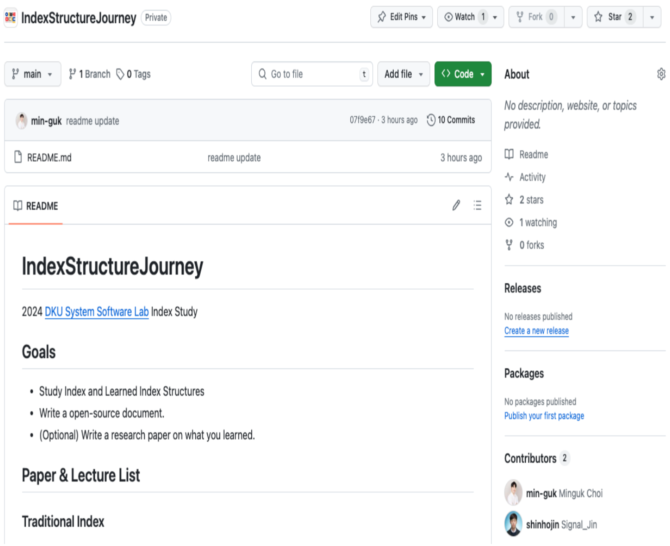
CSSL[10]



Study Github:
Index Structure Journey

Index Structure Journey

- Study Github
  - Link: https://github.com/DKU-StarLab/IndexStructureJourney



Traditional Index

- B-Tree
  - Douglas Comer, "Ubiquitous B-Tree", ACM Computing Surveys, 1979
  - R. Bayer, et al. "Organization and maintenance of large ordered indices", SIGFIDET '70
  - Justin J. Levandoski, et al. "The Bw-Tree: A B-tree for new hardware platform", ICDE 2013
  - J. Rao, et al. "CSB+Tree Making B+-Trees Cache Conscious in Main Memory", SIGMOD 2000
- SkipList
  - William Pugh, "Skip lists: a probabilistic alternative to balanced trees", Communications of the ACM 1990
  - Zhongle Xie, et al. "Parallelizing Skip Lits for In-Memory Multi-Core Database Systems", ICDE 2017
  - Jingtian Zhang, et al. "S3: a scalable in-memory skip-list index for key-value store", VLDB 2019
  - Sprenger, et al. "Cache-Sensitive Skip List: Efficient Range Queries on Modern CPUs", Data Management on New Hardware 2016
  - Jeseong Yeon, et al. "JellyFish: A Fast Skip List with MVCC", Middleware '20
  - Tyler Crain, et al. "No Hot Spot Non-blocking Skip List", ICDCS 2013
  - Henry Daly, et al. "NUMASK: High Performance Scalable Skip List for NUMA", DISC 2018
- ART
  - Viktor Leis, et al. "The adaptive radix tree: ARTful indexing for main-memory databases", ICDE 2013
- MassTree
  - Yandong Mao, et al. "Cache craftiness for fast multicore key-value storage", EuroSys '12
- FAST
  - Changkyu Kim et al. "FAST: fast architecture sensitive tree search on modern CPUs and GPUs", SIGMOD '10
- In-memory Index Survey
  - Z. Xie, et al. "A Comprehensive Performance Evaluation of Modern In-Memory Indices", ICDE 2018
- In-memory Database Survey
  - Abdullah Talha Kabakus, et al, "A performance evaluation of in-memory databases", J. King Saud Univ. Comput. Inf. Sci. 2017

Learned Index

Read-Only Learned Index

- Maltry, Marcel, et al. "A critical analysis of recursive model indexes.", VLDB 22'
- Ferragina, Paolo, et al. "The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds.", VLDB 20'
- Kipf, Andreas, et al. "RadixSpline: a single-pass learned index.", aiDM@SIGMOD 20
- Marcus, Ryan, et al. "Benchmarking learned indexes.", VLDB 20
- Minguk, Choi, et al. "Can Learned Indexes be Build-Efficient? A Deep Dive into Sampling Trade-Offs.", SIGMOD

Updatable Learned Index

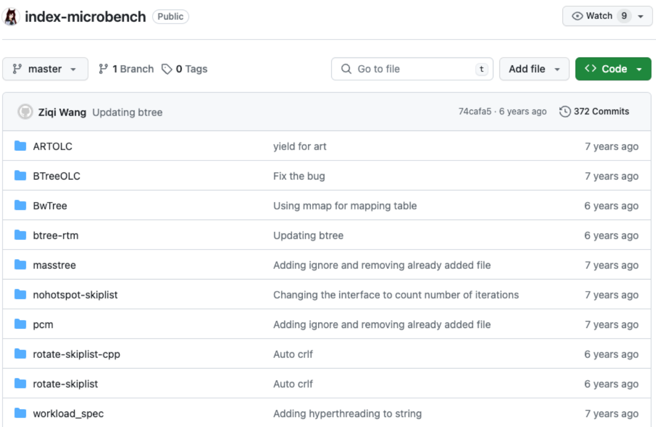
- Ding, Jialin, et al. "ALEX: an updatable adaptive learned index.", SIGMOD 20
- Wu, Jiacheng, et al. "Updatable learned index with precise positions.", VLDB 21
- Ge, Jiake, et al. "SALI: A Scalable Adaptive Learned Index Framework based on Probability Models.", SIGMOD 23
- Wongkham, Chaichon, et al. "Are updatable learned indexes ready?.", VLDB 22'

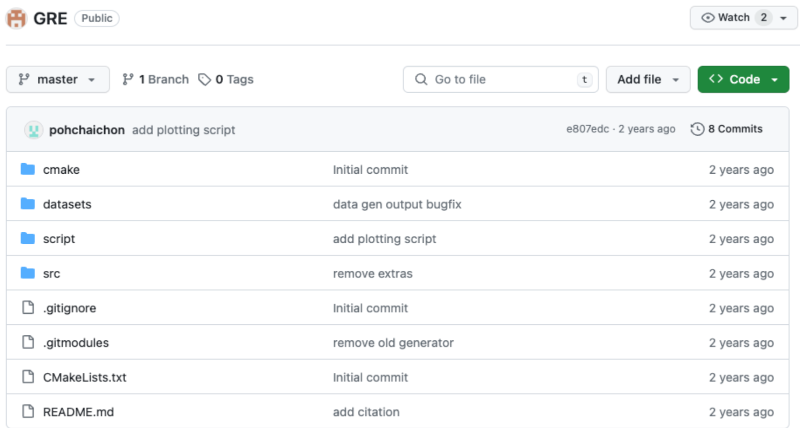
Application

- Error-Bounded PLA Model
  - Xie, Qing, et al. "Maximum error-bounded piecewise linear representation for online stream approximation." VLDB journal 14
- Key-Value Store
  - Dai, Yifan, et al. "From {WiscKey} to Bourbon: A Learned Index for {Log-Structured} Merge Trees.", OSDI 20
  - Yu, Geoffrey X., et al. "Treeline: an update-in-place key-value store for modern storage.", VLDB 22
- NVM Device
  - Lu, Baotong, et al. "APEX: A high-performance learned index on persistent memory.", VLDB 21
- FTL
  - Sun, Jinghan, et al. "Leaftl: A learning-based flash translation layer for solid-state drives.", ASPLOS 23

Index Structure Journey

- Open Source
  - Traditional Index benchmark
    - Index-microbench: https://github.com/wangziqi2016/index-microbench
  - Learned Index benchmark
    - GRE bench: https://github.com/gre4index/GRE





Q & A

Thank you!

Reference

- Reference List

- [1] Douglas Comer, “Ubiquitous B-Tree”, ACM Computing Surveys, 1979
- [2] R. Bayer, et al. “Organization and maintenance of large ordered indices”, SIGFIDET ’70
- [3] Changkyu Kim et al. “FAST: fast architecture sensitive tree search on modern CPUs and GPUs”, SIGMOD ’10
- [4] William Pugh, “Skip lists: a probabilistic alternative to balanced trees”, Communications of the ACM 1990
- [5] Jeseong Yeon, et al. “JellyFish: A Fast Skip List with MVCC”, Middleware ’20
- [6] Tyler Crain, et al. "No Hot Spot Non-blocking Skip List", ICDCS 2013
- [7] Henry Daly, et al. "NUMASK: High Performance Scalable Skip List for NUMA", DISC 2018
- [8] Viktor Leis, et al. "The adaptive radix tree: ARTful indexing for main-memory databases", ICDE 2013
- [9] Jingtian Zhang, et al. "S3: a scalable in-memory skip-list index for key-value store", VLDB 2019
- [10] Sprenger, et al. "Cache-Sensitive Skip List: Efficient Range Queries on Modern CPUs", Data Management on New Hardware 2016