FINEdex: A Fine-grained Learned Index Scheme for Scalable and Concurrent Memory Systems

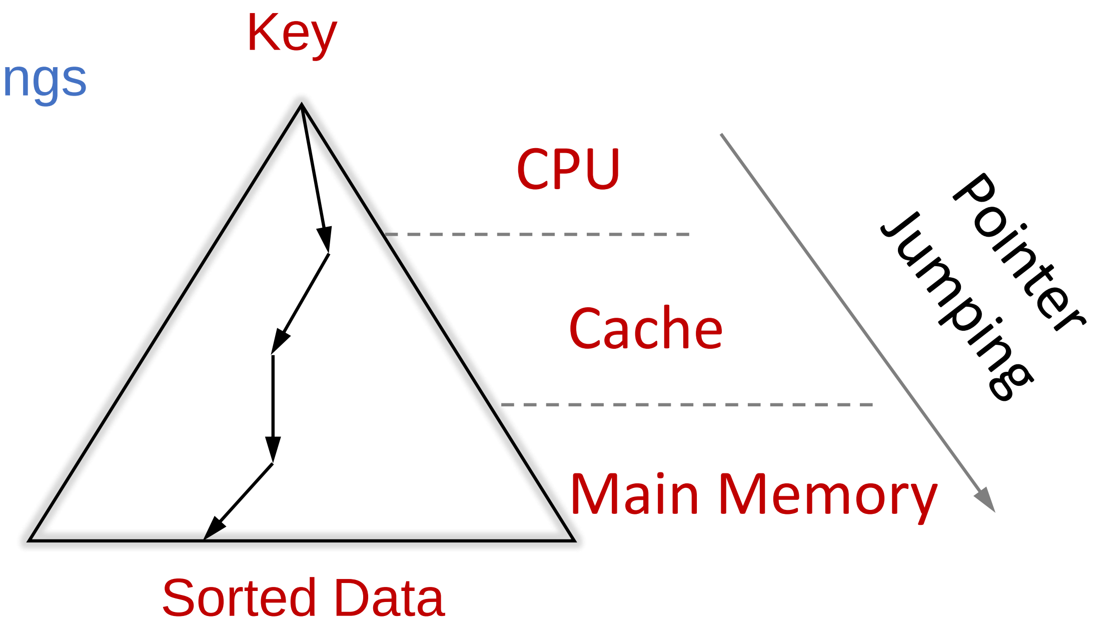
Pengfei Li, Yu Hua, Jingnan Jia, Pengfei Zuo

Huazhong University of Science and Technology

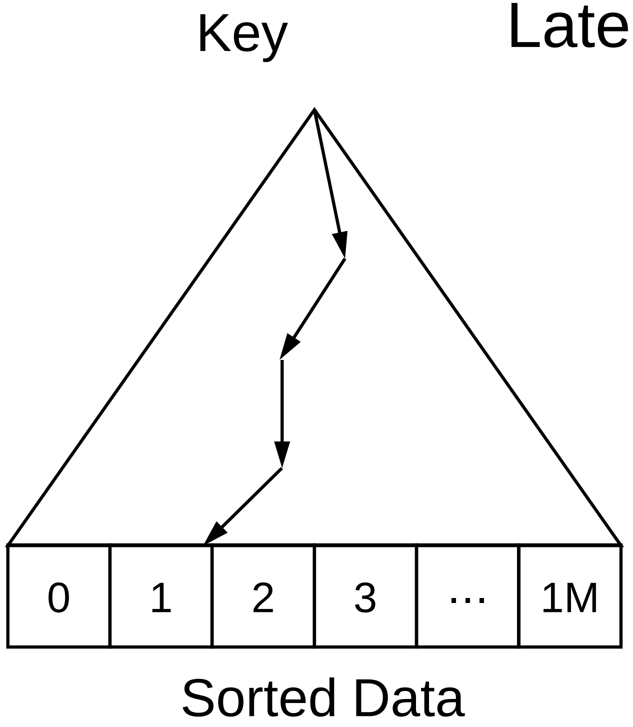
VLDB 2022

Traditional B-Trees overlook data patterns

- ✗ Efficient point/range query
  - High penalty of multiple pointer jumpings
- ✓ Dynamic structure adjustment
  - Dynamic tree balancing
- ✗ Low memory footprint
  - Multiple-level inner nodes
- ✗ Enable high concurrency
  - Heavy dependency among nodes



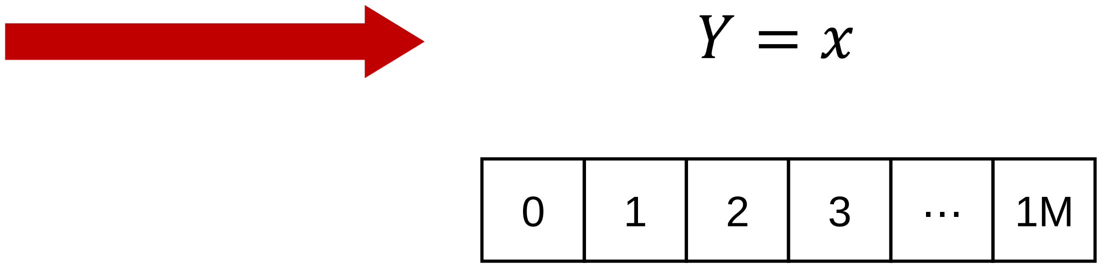
Exact data distribution enables efficiency



Latency & Memory footprint:

- memory jumpings > cost-efficient computations
- multiple-level nodes > small number of parameters

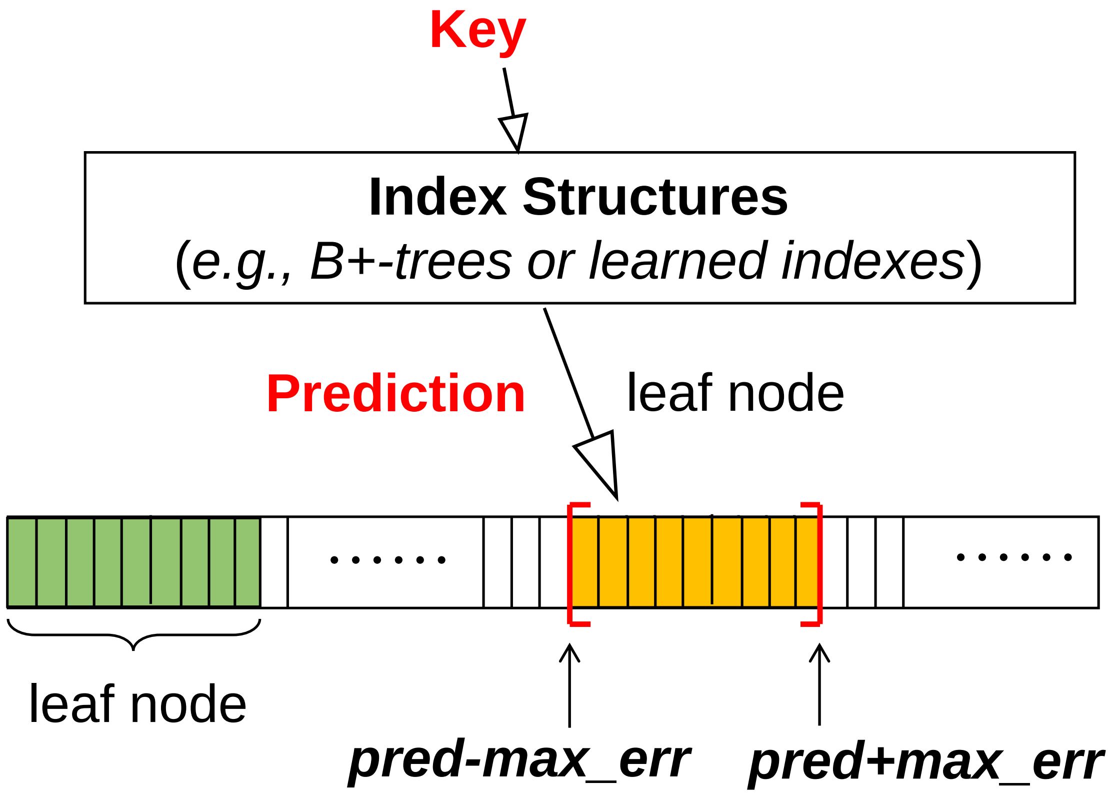
$$Y = x$$



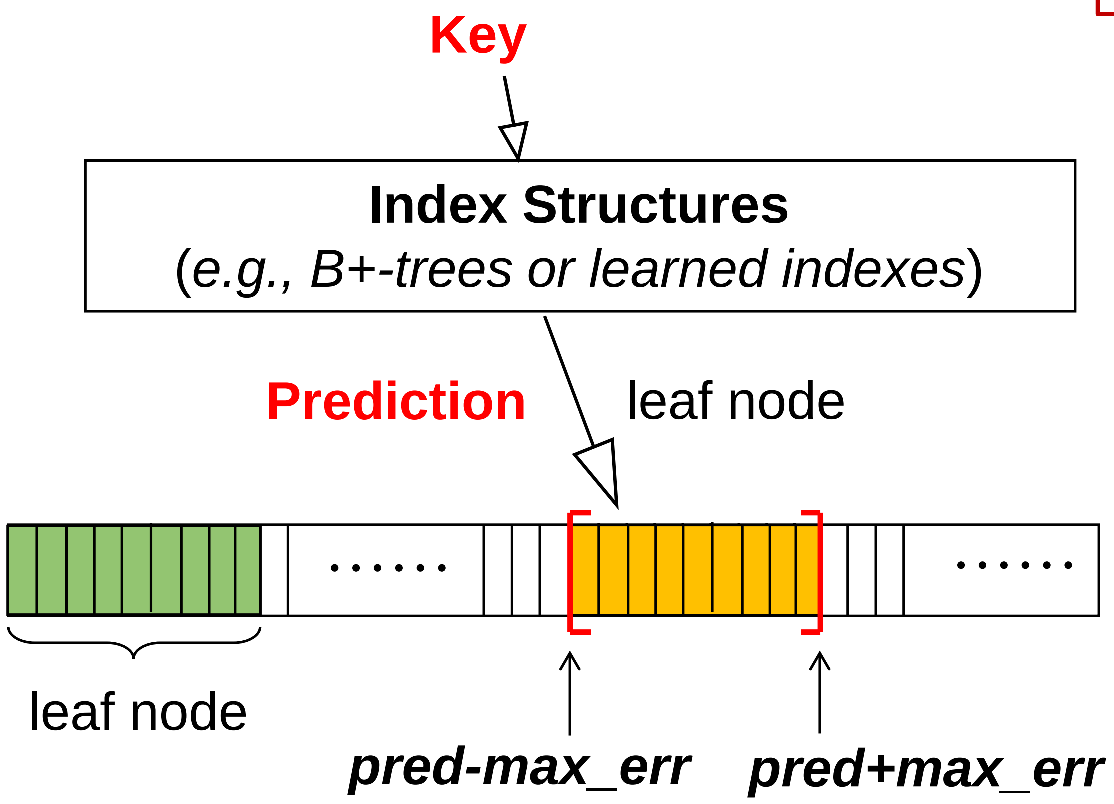
Consider Indexes as ML models

Learned indexes

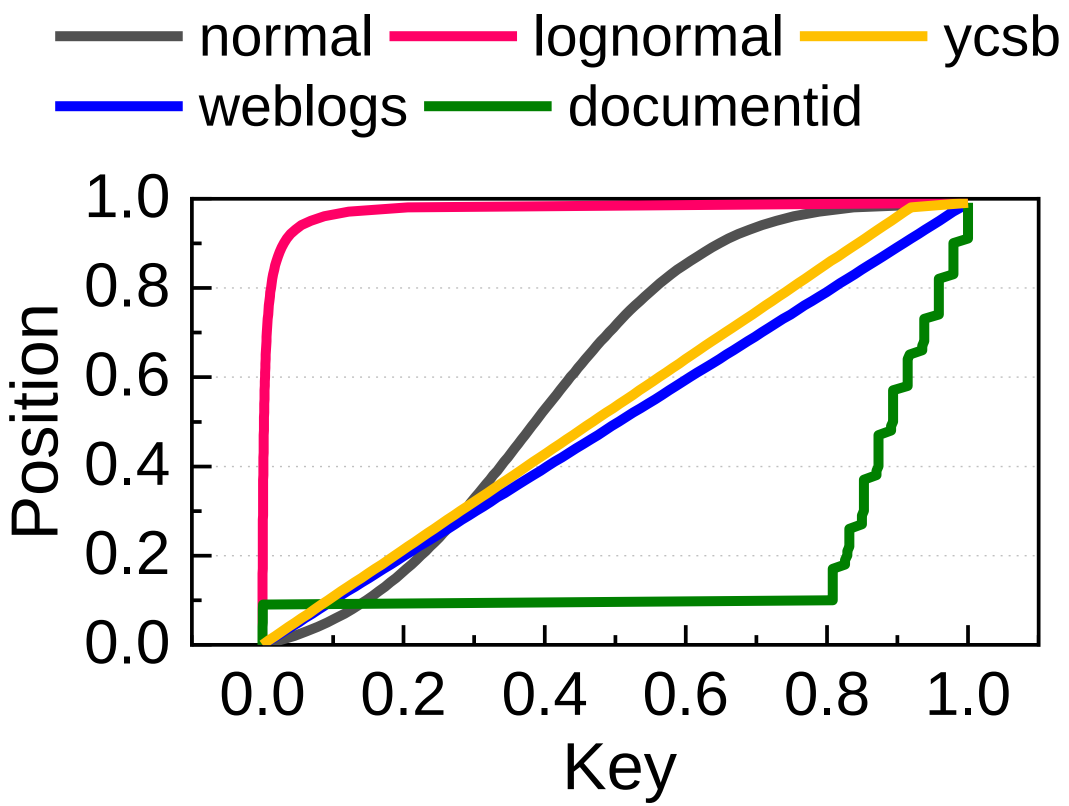
Indexes are regression models



Learned indexes



Learn CDF for high accuracy



Learned indexes could be better

- ✔ Efficient point/range query
  - Cost-efficient computations for searching
- ✔ Low memory footprint
  - Small number of parameters
- ✘ Dynamic structure adjustment
  - High-overhead retraining
- ✘ Enable high concurrency
  - Heavy data dependency

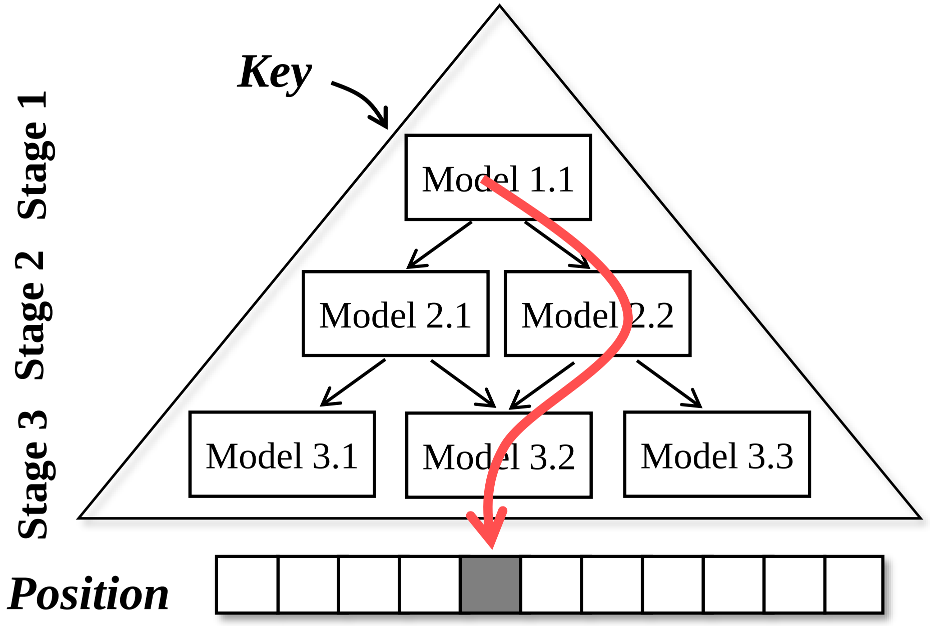
Challenge 1: Limited Scalability

Schemes	Insertion without data loss	Keep all data sorted	Concurrency Write	Concurrency retrain
Learned indexes SIGMOD' 18	✗	✓	✗	✗
FITing-tree SIGMOD' 19	✓	✗	✗	✗
Xindex PPoPP' 20	✓	✗	✓	✓
ALEX SIGMOD' 20	✓	✓	✗	✗
PGM-index VLDB' 20	✓	✓	✗	✗
FINEdex	✓	✓	✓	✓

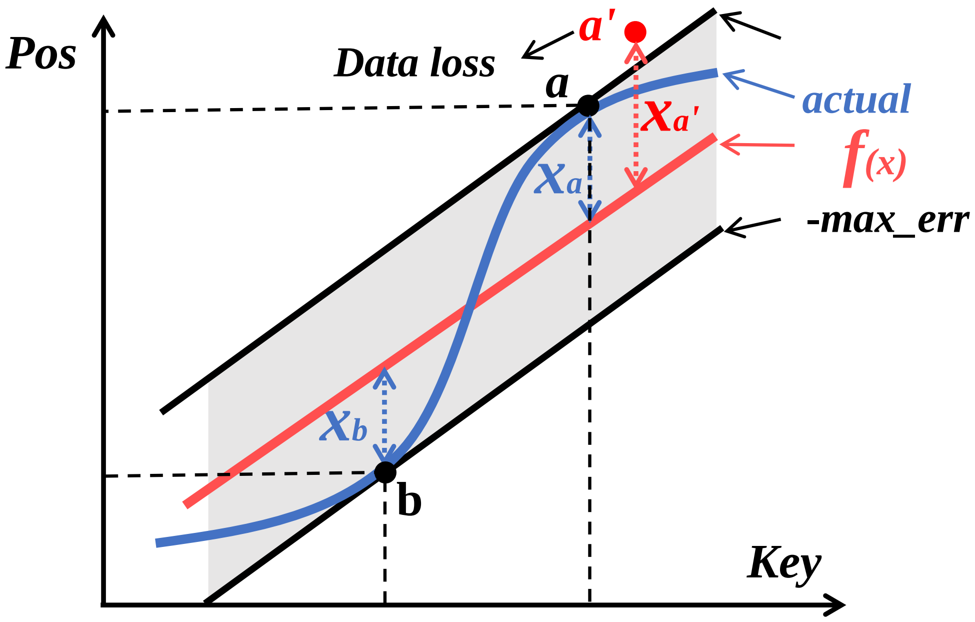
Challenge 1: Limited Scalability

Model & Data dependency hinders scalability

- Inflexible to update models



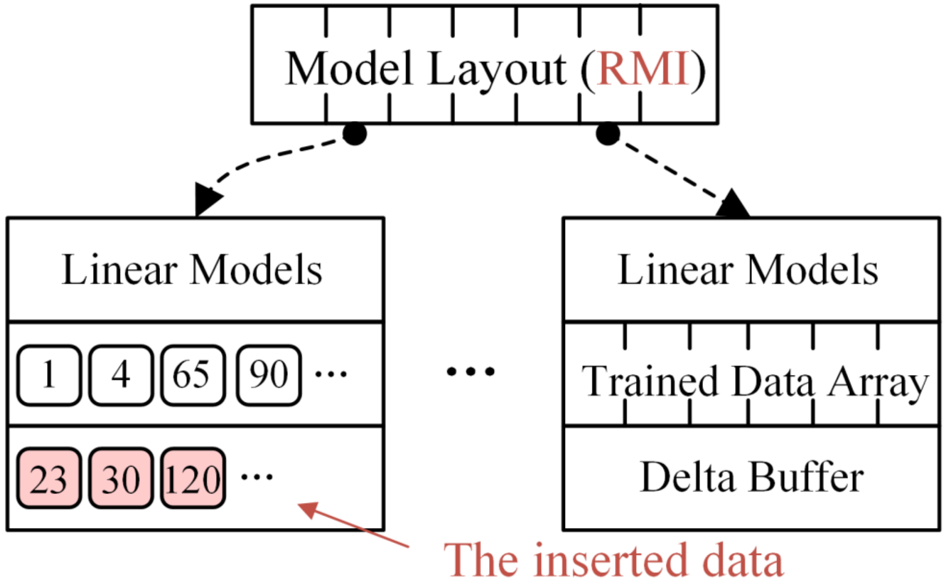
- Fail to process inserts



Challenge 2: High Overheads

[FITing-tree & Xindex] delta buffer

- Construct a delta buffer (e.g., B-tree, Masstree) to process new inserts
- Periodically retrain the retrained data array and the delta buffer



- **Data are not sorted**

  Inefficient range query
- **Large buffer decreases the performance**

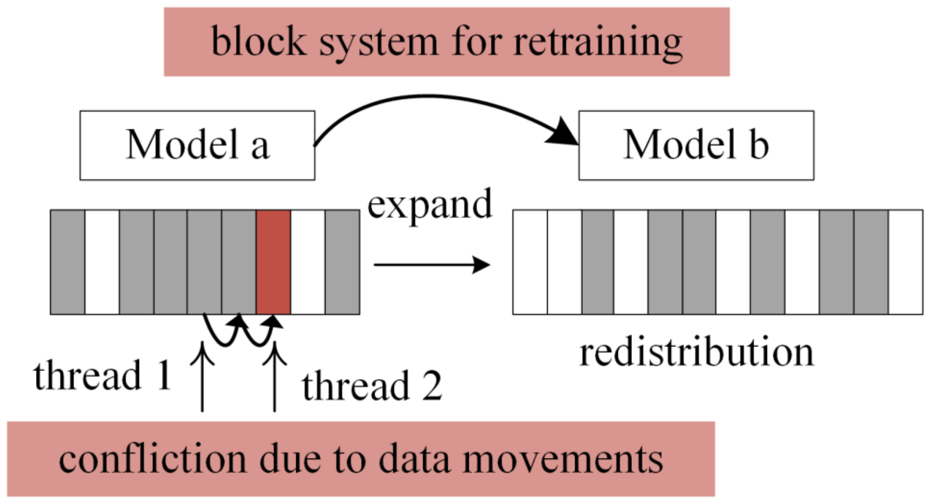
  Long latency to search the buffer
- **Data dependency in the shared buffer**

  Poor concurrent performance

Challenge 2: High Overheads

[ALEX & PGM-index] preserve empty slots

- Preserve empty slots in the trained data array to process inserts
- Expand the trained data array and retrain the models to construct sufficient slots



- **Data dependency → poor concurrency**

  Different threads compete for empty slots
- **Fail to support concurrent retraining**

  Block the system to move data and retrain models

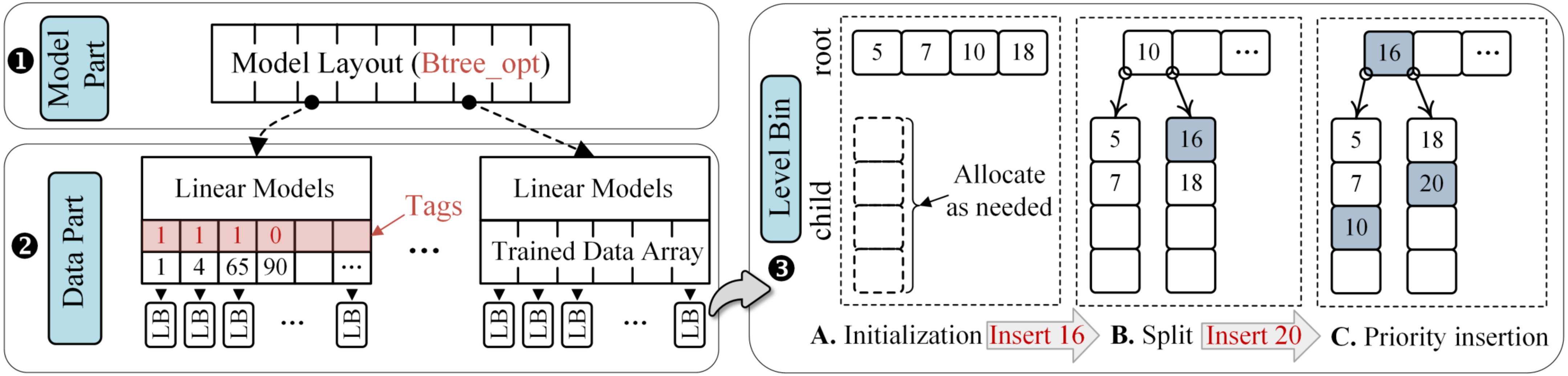
  → Incurs long latency

  → Decrease the overall performance

FINEdex: Fine-grained Learned Index Scheme

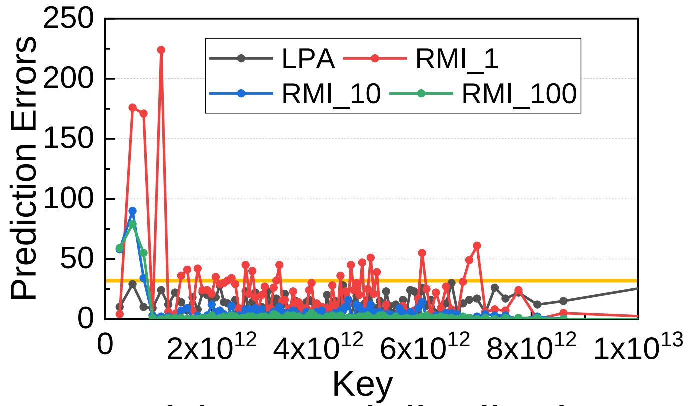
Design overview

- Model part: training independent models
- Data part: flattened data structure with low data dependency
- Two-granularities concurrent retraining

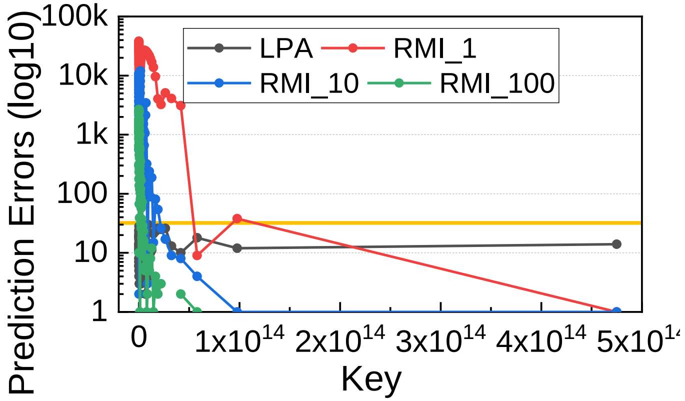


Model part: Model accuracy

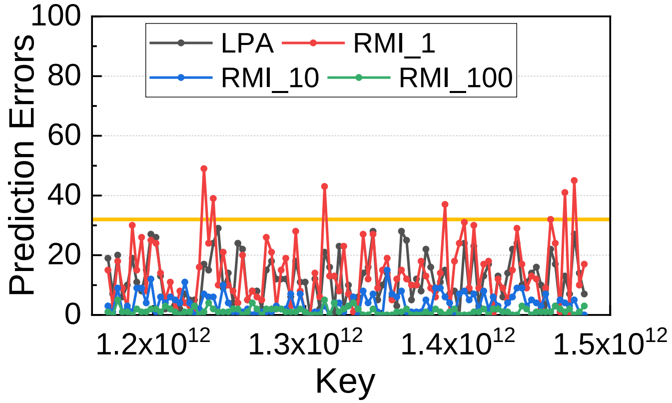
- RMI requires a large number of models for high accuracy
- The model accuracies become diverse in the same data distribution



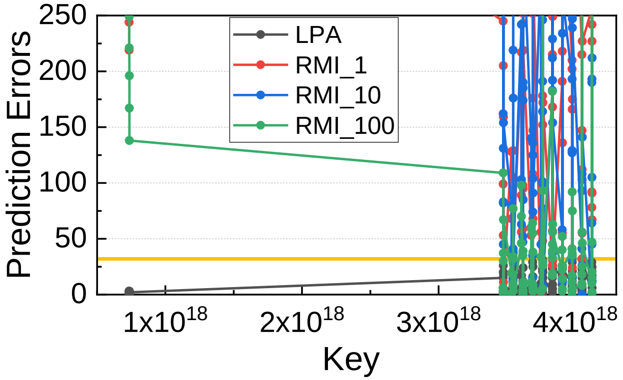
(a) normal distribution



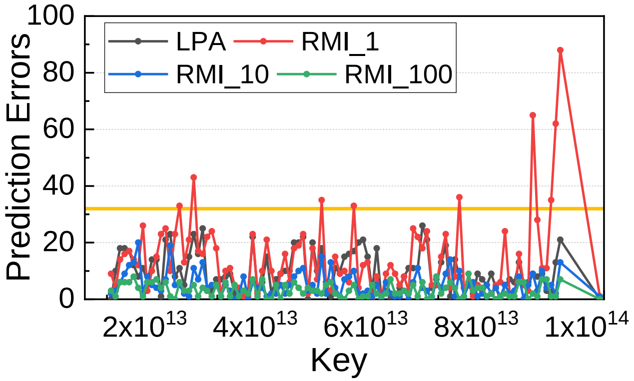
(b) lognormal distribution



(c) weblogs



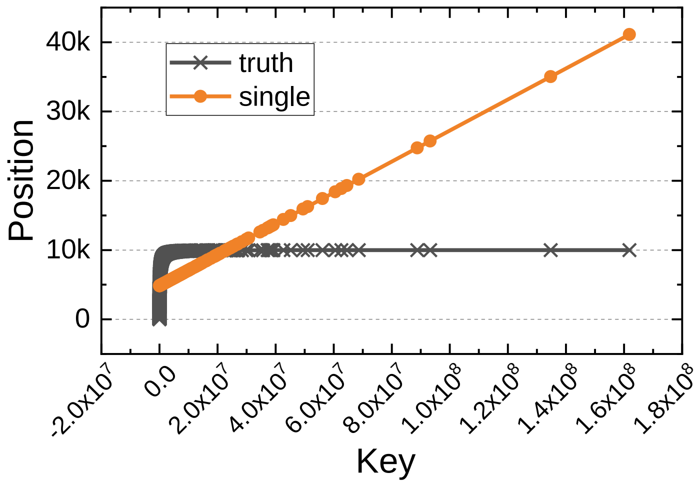
(d) docId



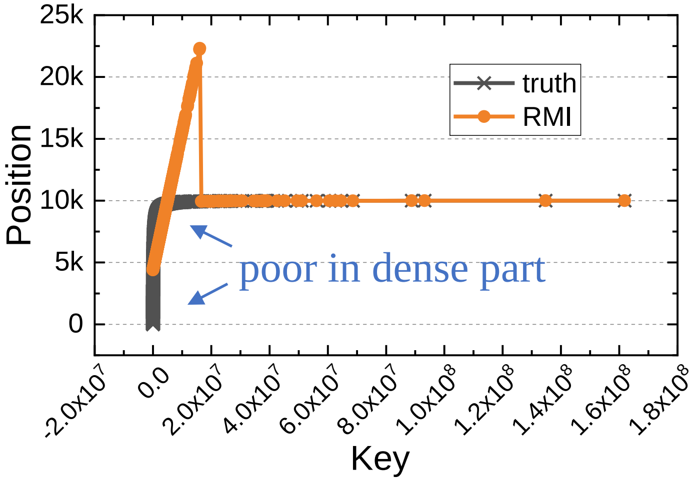
(e) YCSB zipfian

Model part: Model accuracy

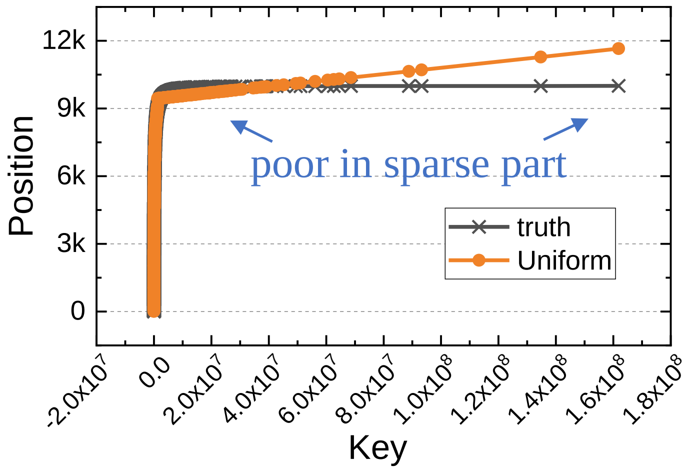
- ➢ Various schemes show different learning effects on the same data distribution
- ➢ Existing schemes fail to learn the data distribution well



(a) using a single model



(b) RMI learning



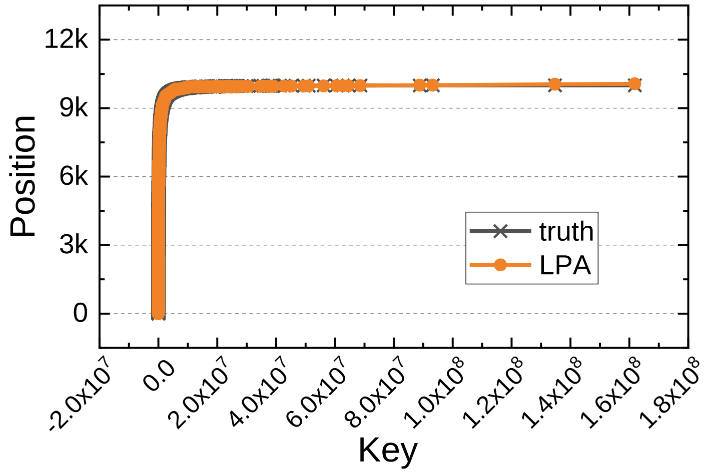
(c) Uniform learning

Fail to train models according to the data distributions

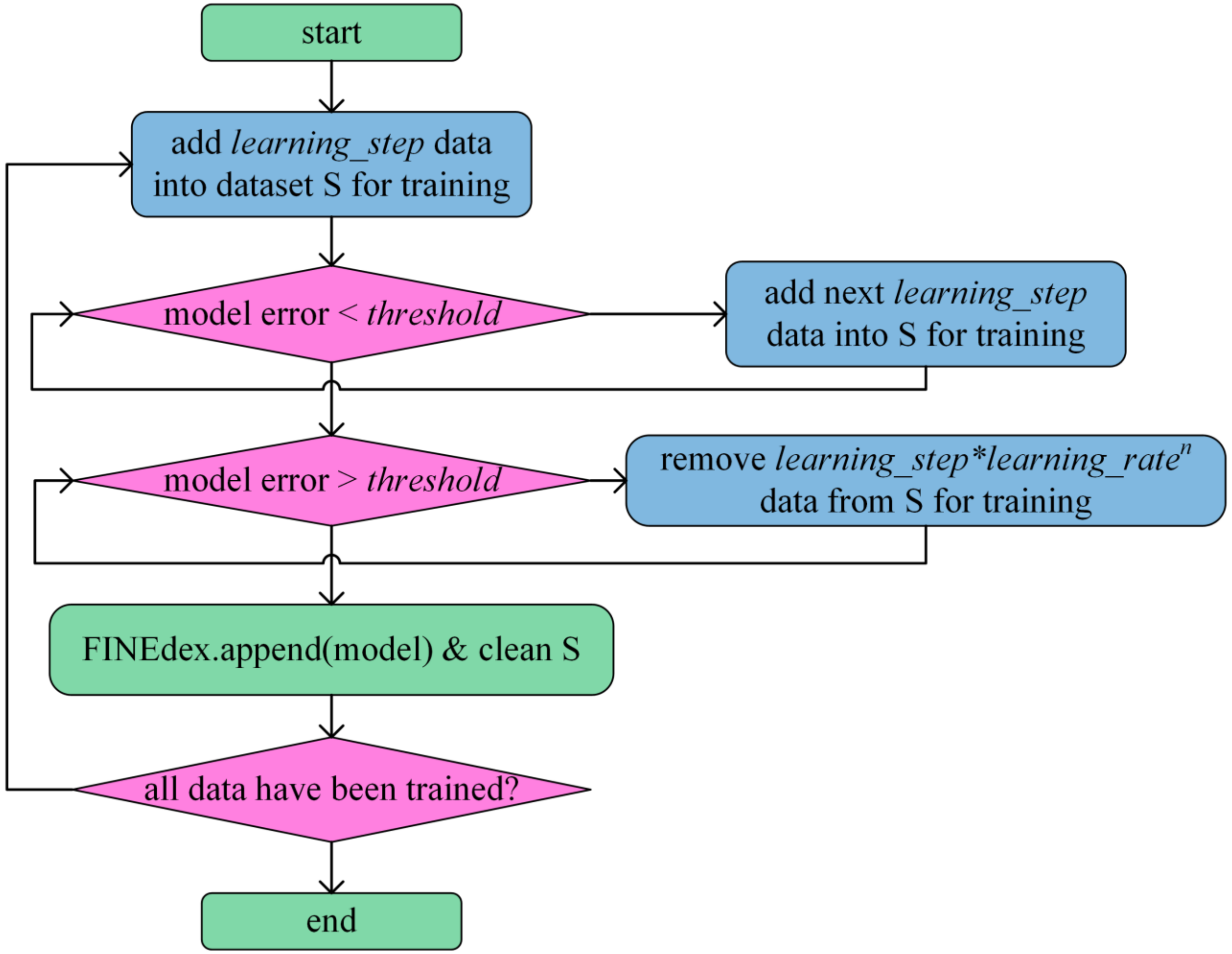
Model part: Learning Probe Algorithm (LPA)

Parameters of LPA:

- *threshold* determine the max error
- *learning_step* & *learning_rate* determine the learning speed

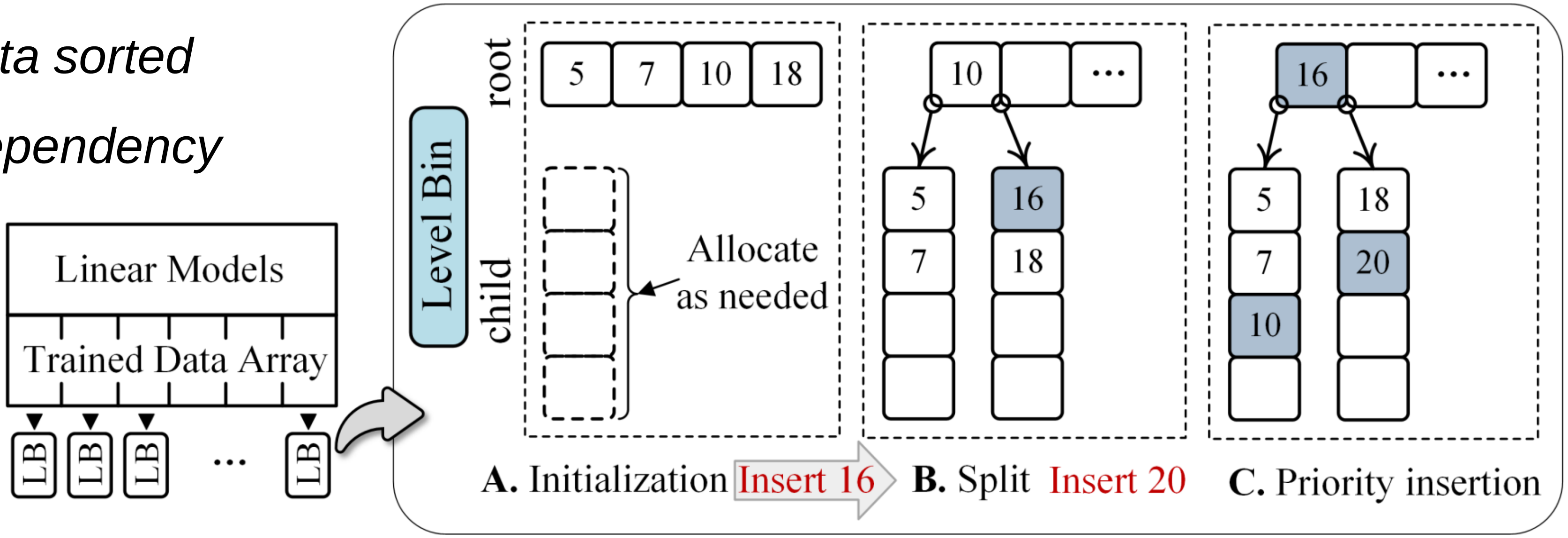


LPA learns the data distribution well



Data part: Level bins

- *No data loss*
- *Keep all data sorted*
- *low data dependency*

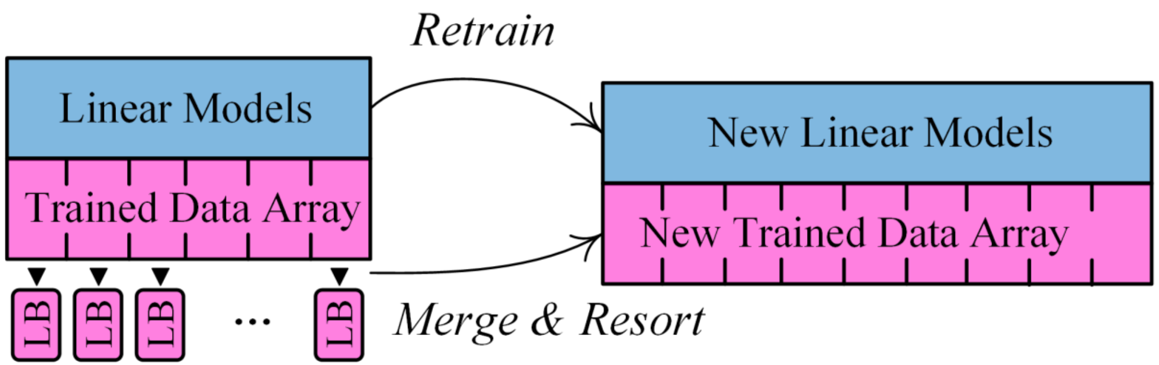


Flattened data structure:

- Append small sized level bins behind each trained data
- Insert data into previous bins for high space utilization

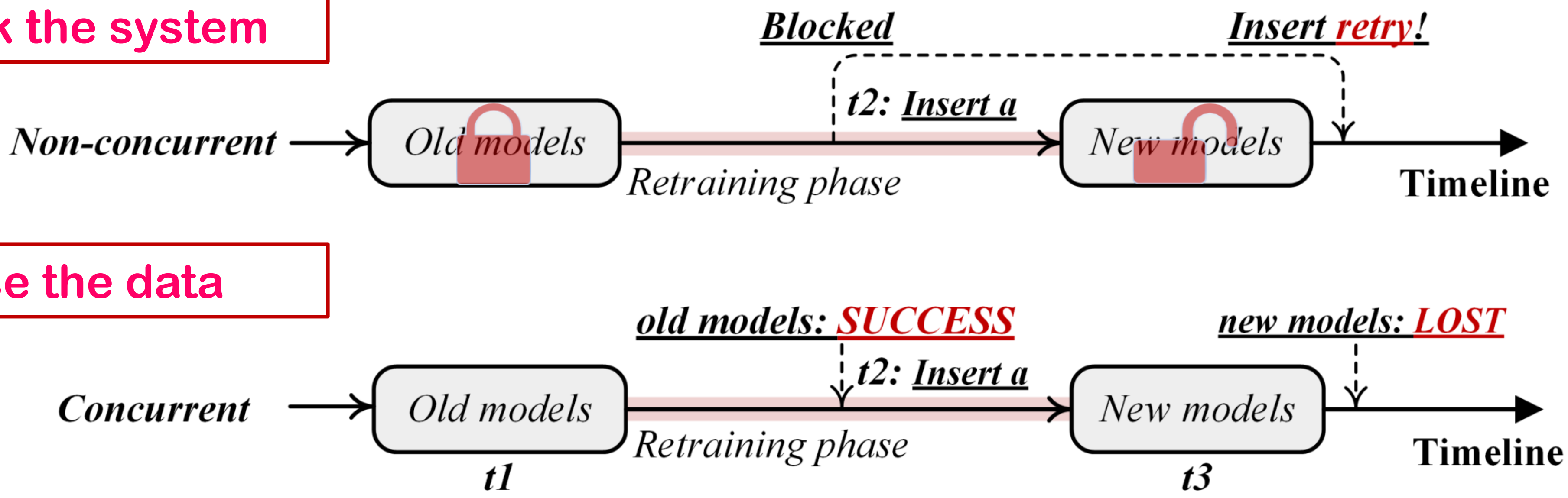
Concurrent Retraining: Challenges

a. Merge data
b. Resort data
c. Retrain new model

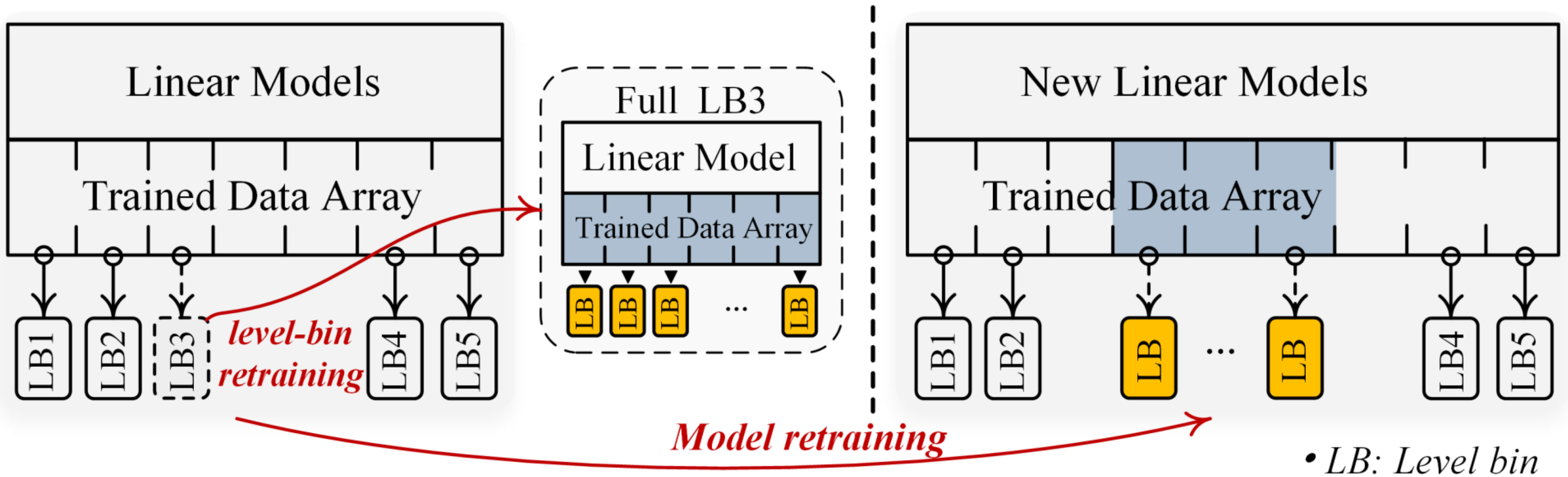


block the system

lose the data



Concurrent Retraining: Retrain in two granularities



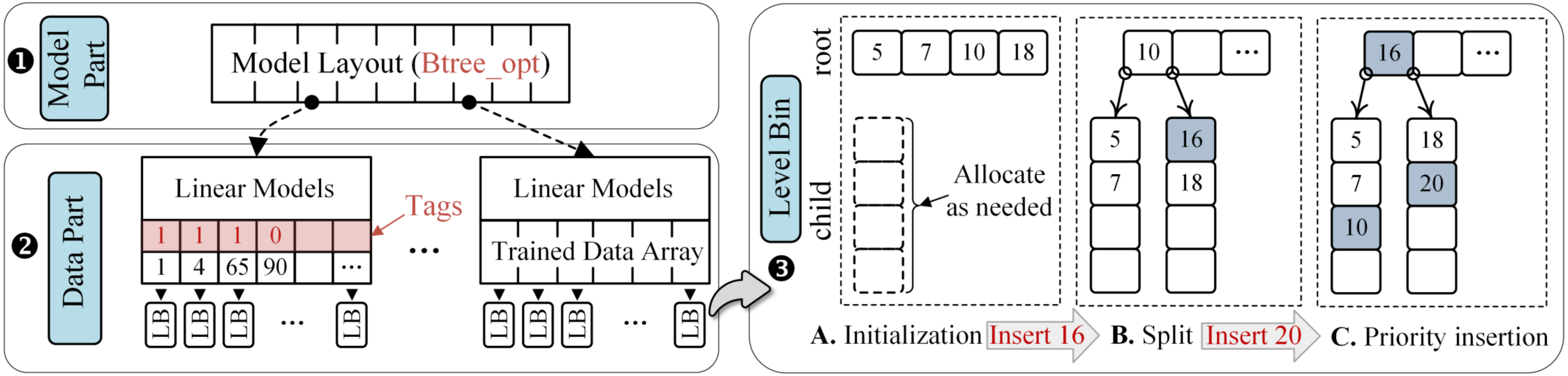
- Level-bin retraining retrains the full level bins

  Other trained data and LBs are not blocked

- Model retraining merges the small models for high performance

  Perform in background

Practical Operations



Search the data:

Stage ❶ Find the model that cover the given key

Stage ❷ Search in the prediction range

Stage ❸ Operate in the level bins

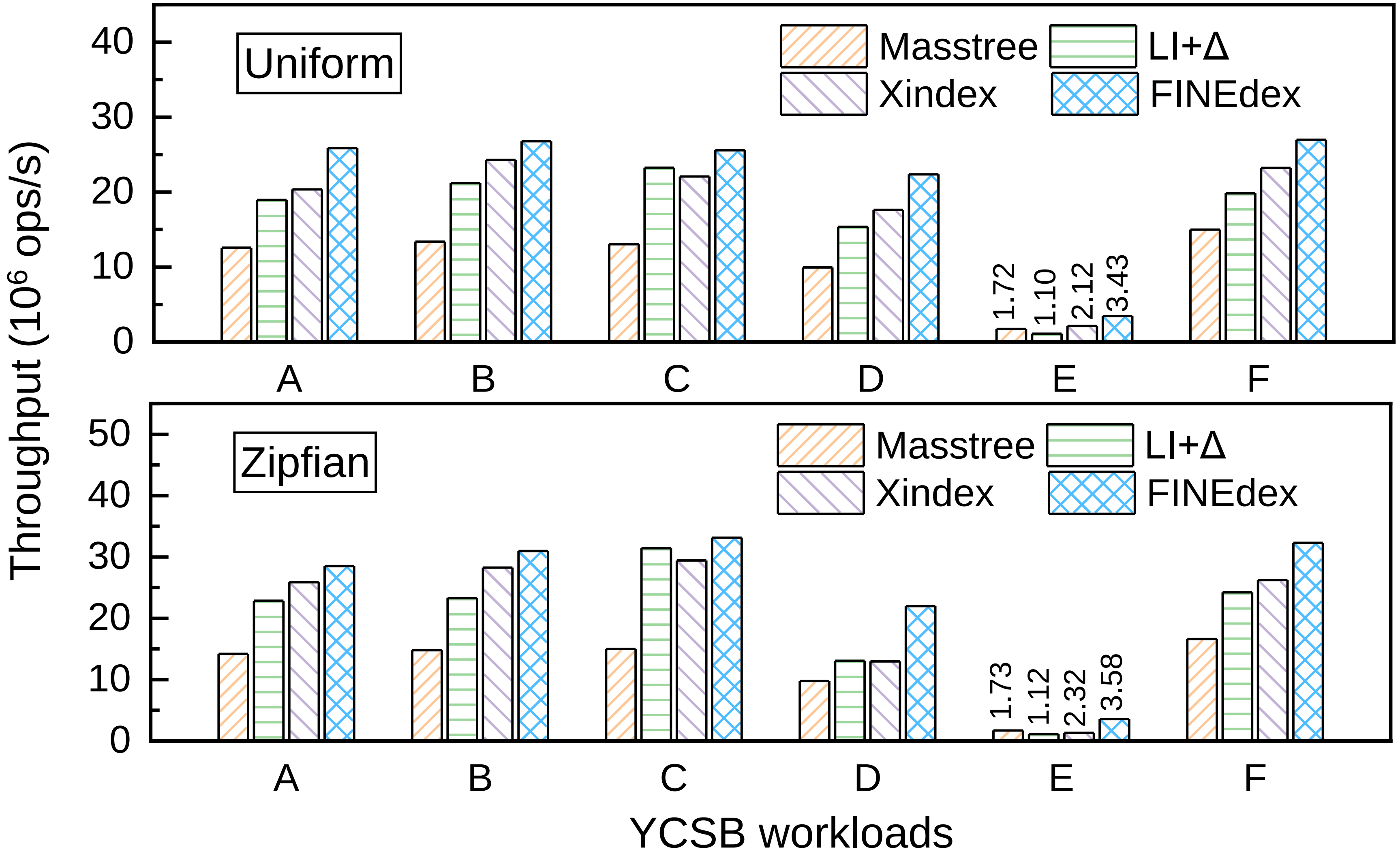
- ✓ *Update* the corresponding value pointer
- ✓ *Insert* into the level bins
- ✓ *Remove* the data from the level bins or *unset* the tags in the trained data array

Experimental Setup

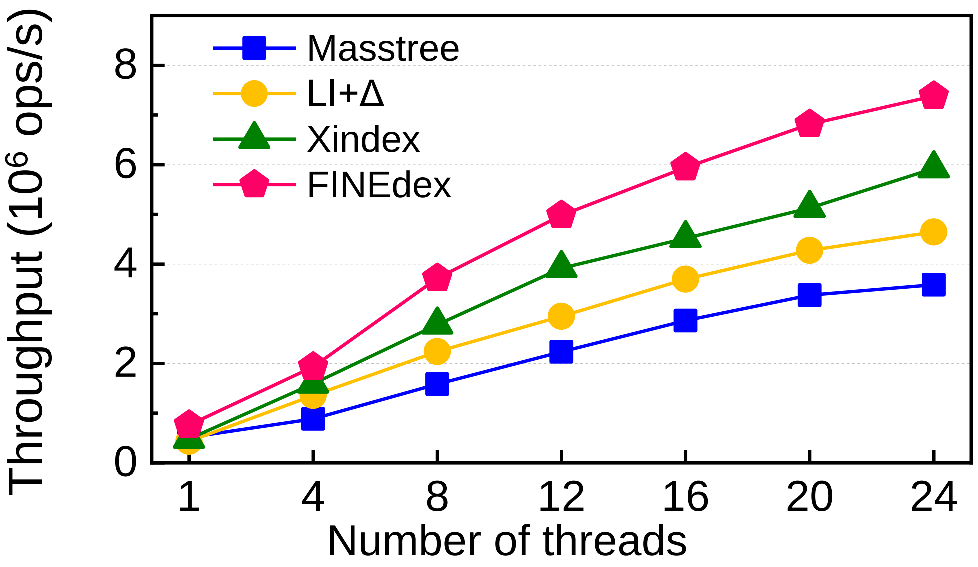
- **Testbed**
  - 12-core Intel(R) Xeon(R) CPU @2.50GHz
  - Run codes with 24 threads
- **Workloads**
  - YCSB with 6 workloads; Weblogs; DocID; Lognormal & Normal distributions
  - 8-byte keys and value-pointers (point to variable-length values)
- **Comparisons**
  - Masstree (a variant of concurrent B+tree) [EuroSys’12]
  - Learned Indexes + delta-buffer (not support concurrent retraining) [SIGMOD’18]
  - XIndex (support concurrent retraining) [PPoPP’20]

Open-source address: https://github.com/iotlpf/FINEdex

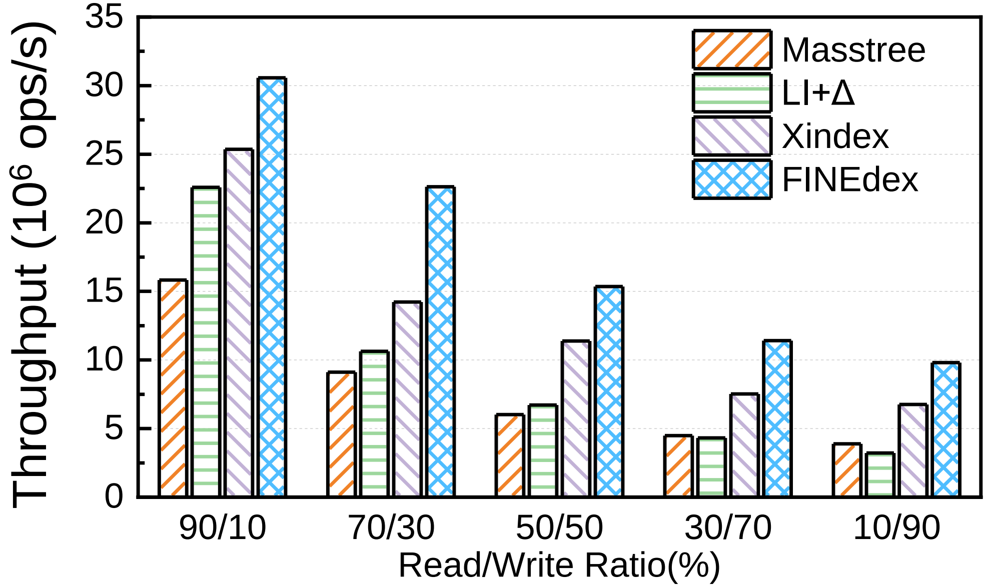
Throughputs on YCSB: Work well on dynamic workloads



Throughputs with heavy writes



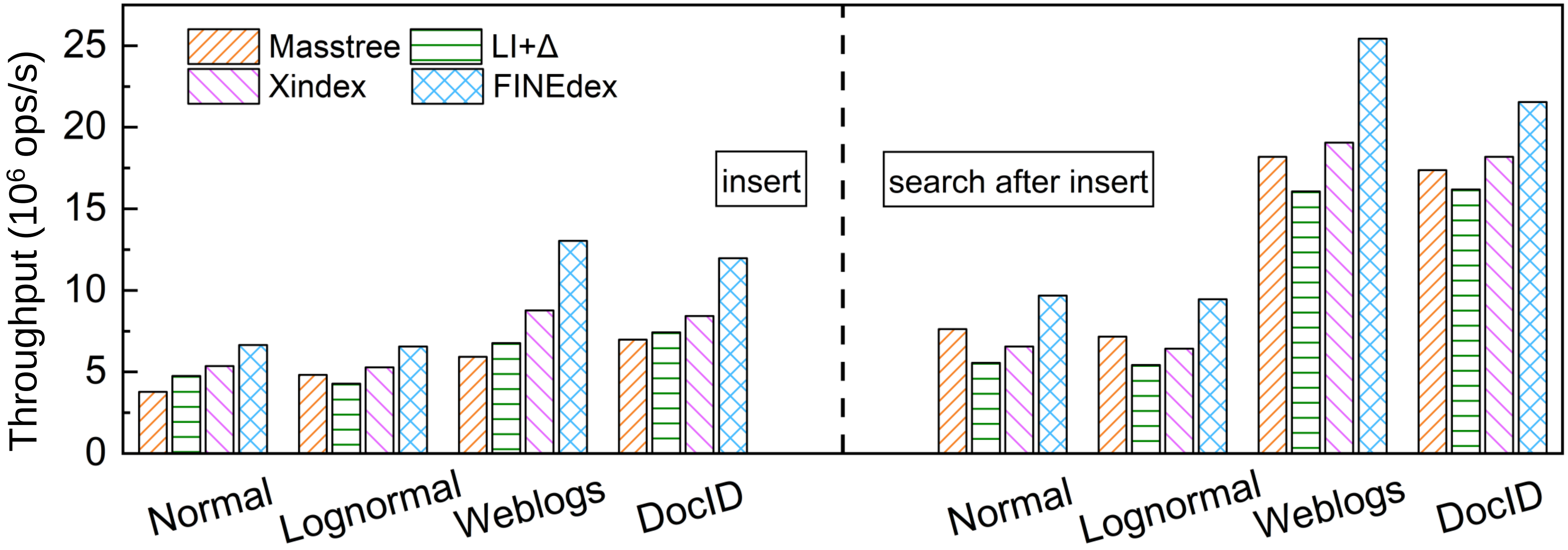
(a) Insert with multiple threads



(b) performance with different R/W ratios

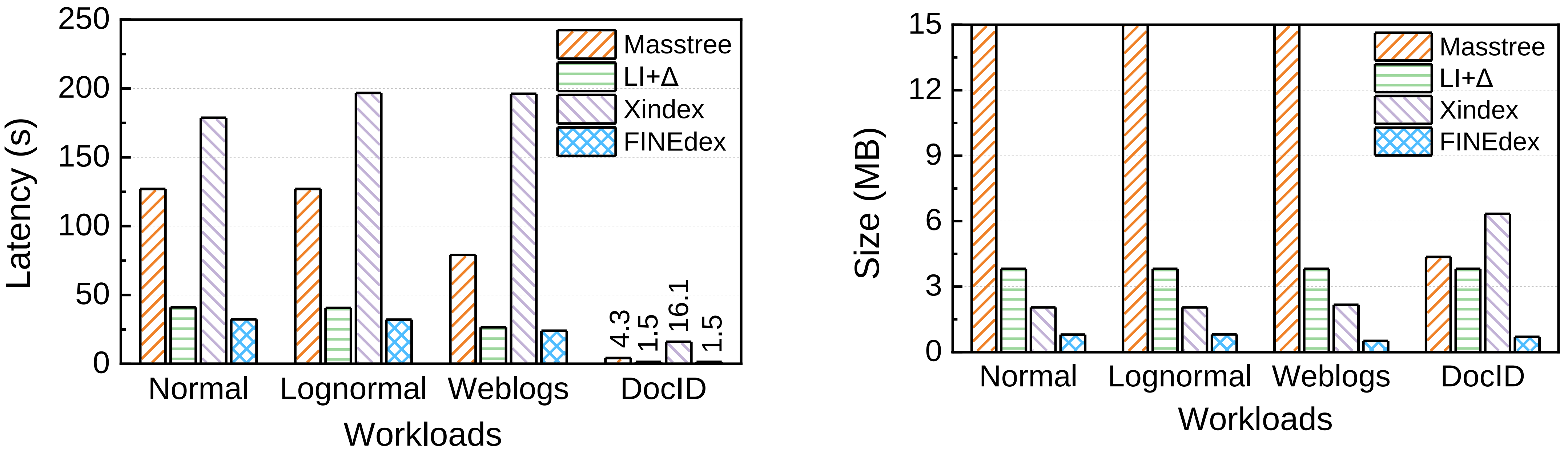
- FINEdex improves the insert performance by 1.3x~2.0x
- FINEdex delivers high performance on write-intensive workloads

Throughputs on different workloads



- FINEdex improves the insertion performance
- FINEdex has high search performance after a large number of inserts

Overheads analysis



(a) Training latency (b) Memory overheads of models/inner nodes

- FINEdex incurs lower latency than other schemes by 1.3x~8.9x
- FINEdex obtains a large amount of memory savings

Conclusion

- **Existing learned index schemes show limited scalability and incurs high overheads to process dynamic workloads**
  - Requirements: No data lose, keep all data sorted, high concurrency
- **We propose FINEdex for scalable and concurrent memory systems**
  - Adaptive training algorithm generates independent models
  - Flattened data structure with low data dependency
  - Cost-efficient concurrent retraining scheme
- **FINEdex outperforms state-of-the-art learned index schemes by up to 2.0x in write-intensive workloads**

Thanks!

Q & A