Learned Secondary Index

Chen Yang
Junru He

What is included

1. Brief review of traditional indexing
2. Introduction to learned index
3. Learned secondary index

    3.1 Permutation vector

    3.2 Fingerprint vector

    3.3 How to build LSI

1. Lookup procedure
2. Evaluation

What is included

1. **Brief review of traditional indexing**
2. Introduction to learned index
3. Learned secondary index

    3.1 Permutation vector

    3.2 Fingerprint vector

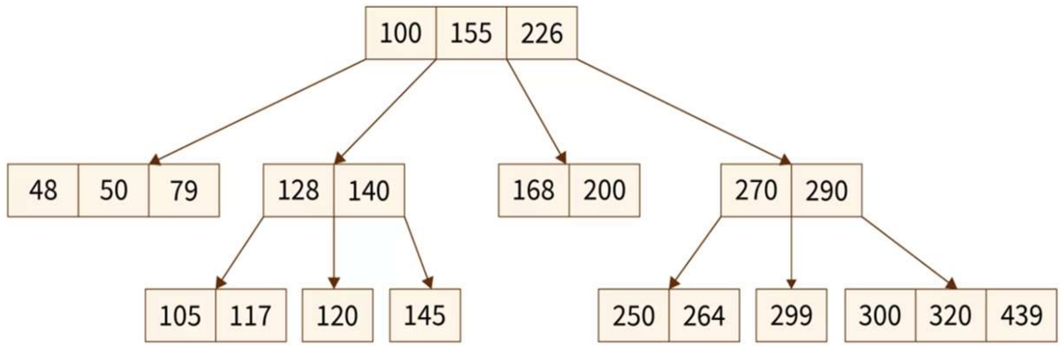
    3.3 How to build LSI

1. Lookup procedure
2. Evaluation

Traditional Index

Indexing is a process of narrowing the search range to make it easier to find the lookup key

B-Tree & B+ Tree:
Narrowing the range level by level



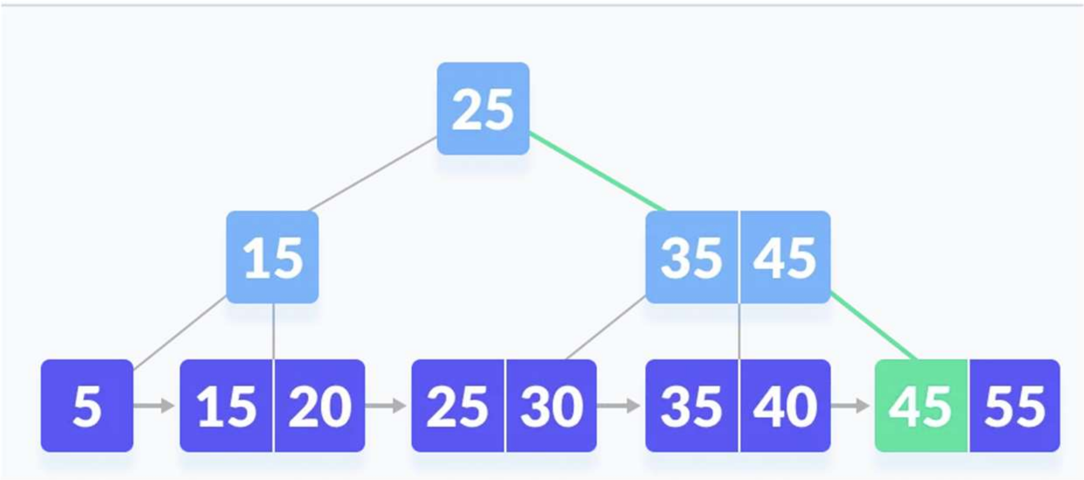
E.g B-Tree

Data must be sorted

Traditional Index

Indexing is a process of narrowing the search range to make it easier to find the lookup key

B-Tree & B+ Tree:
Narrowing the range level by level



E.g B+ Tree

Data must be sorted

Traditional Index

1. Storing key-pointer
2. Narrowing the search range by comparing key level by level
3. The underlying data must be sorted

Traditional Index

1. Storing key-pointer
2. **Narrowing the search range by comparing key level by level**
3. The underlying data must be sorted

Q1: Is it possible to narrow the search range faster or directly?

Traditional Index

1. Storing key-pointer
2. Narrowing the search range by comparing key level by level
3. **The underlying data must be sorted**

Q1: Is it possible to narrow the search range faster or directly?

Q2: How could we know the lookup key position on unsorted underlying data?

What is included

1. Brief review of traditional indexing
2. **Introduction to learned index**
3. Learned secondary index

    3.1 Permutation vector

    3.2 Fingerprint vector

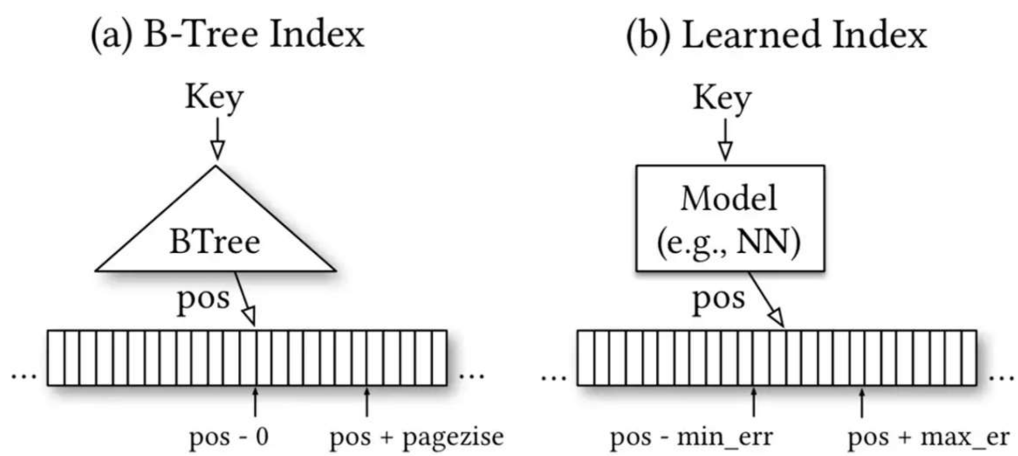
    3.3 How to build LSI

1. Lookup procedure
2. Evaluation

Learned Index

Unlike traditional index structures such as B-trees,
learned indexes build a model over the underlying data to predict the **position** of a lookup key in a sorted array

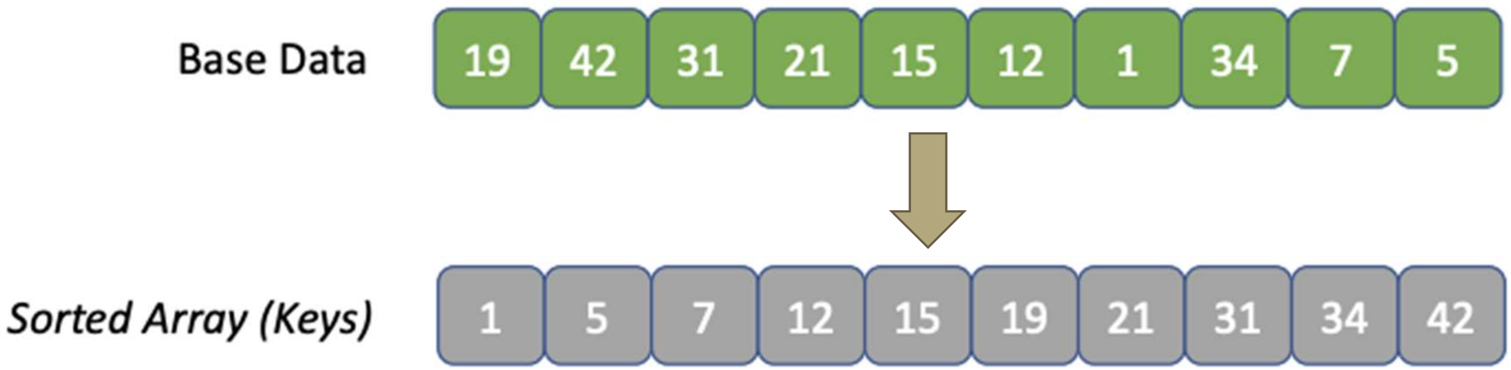
“Position” ⇒ a smaller range



Learned Index

How is this model built?

Step1: Make a sorted copy of the underlying data to train the model



Learned Index

How is this model built?

Step2: Construct a cumulative distribution function (CDF) based on the ordered data.

CDF: CDF indicates the proportion of data items that are less than or equal to a given key.

$$F(x) = P(X \leq x)$$

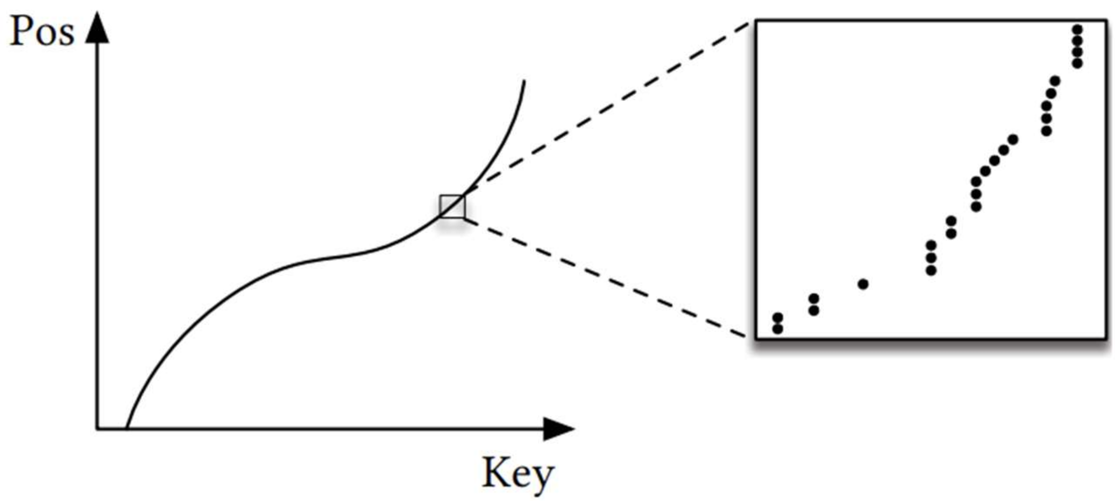
Why CDF? ⇒ It is a approximate of a position predicting model

Learned Index

How is this model built?

CDF: $$F(x) = P(X \leq x)$$

P = F(Key) * N

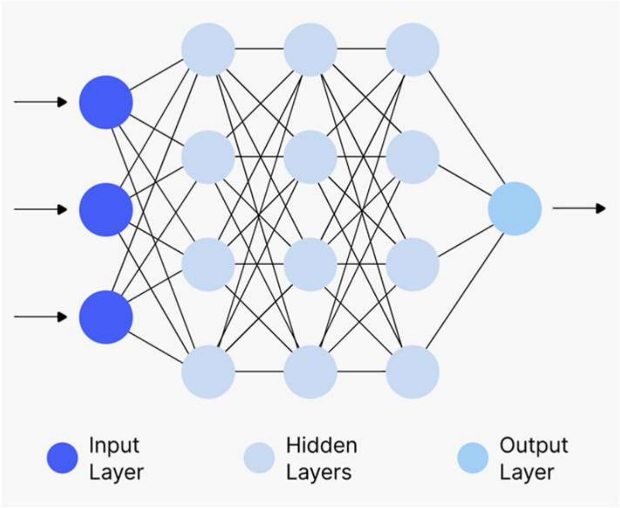


Index can be thought of as an approximation of the cumulative distribution function (CDF) of the data

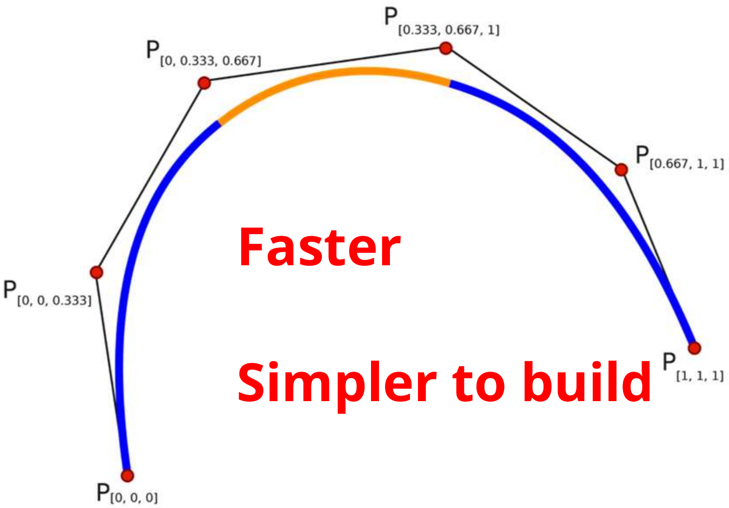
Learned Index

How is this model built?

Step3: Model Selection



VS



Neural network

Simpler models like Spline

Learned Index

The learned index model: **Practical Learned Index (PLEX)**

PLEX

RadixSpline: A linear spline model approximates the CDF of the data

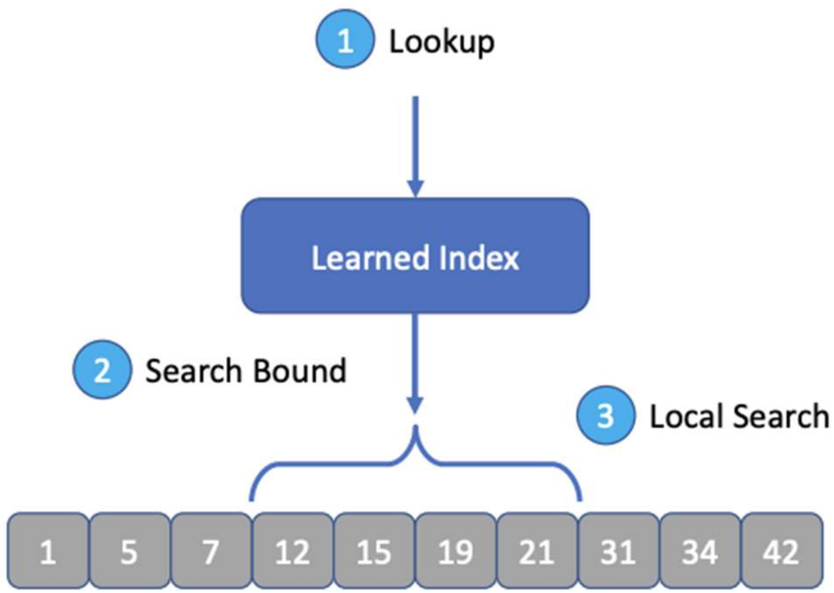
Hist-Tree: approximates the data distributed as a histogram

Reference:
Stoian, M., Kipf, A., Marcus, R., & Kraska, T. (2021). Towards Practical Learned Indexing. Proceedings of the AIDB 2021: 3rd International Workshop on Applied AI for Database Systems and Applications, Copenhagen, Denmark.

Link:
https://www.researchgate.net/publication/353838541_PLEX_Towards_Practical_Learned_Indexing

Learned Index

The learned index model: **Practical Learned Index (PLEX)**



Learned Index

How is this model built?

Step4: Train the Model, Evaluate and Optimize it

Step5: Establish the model's **maximum error bound.**

To make sure the model is reliable

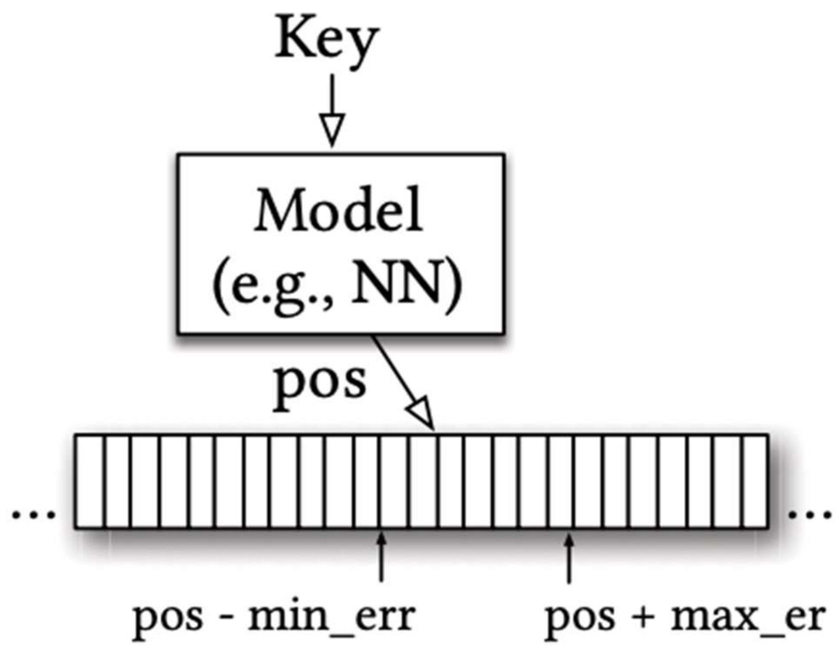
Learned Index

Learned Index can provide:

1. A mapping from keys to predicted range.
2. The maximum error that prediction can incur.

Then we can find the position by using binary search

On this range



Q1: Can we find the position not level by level

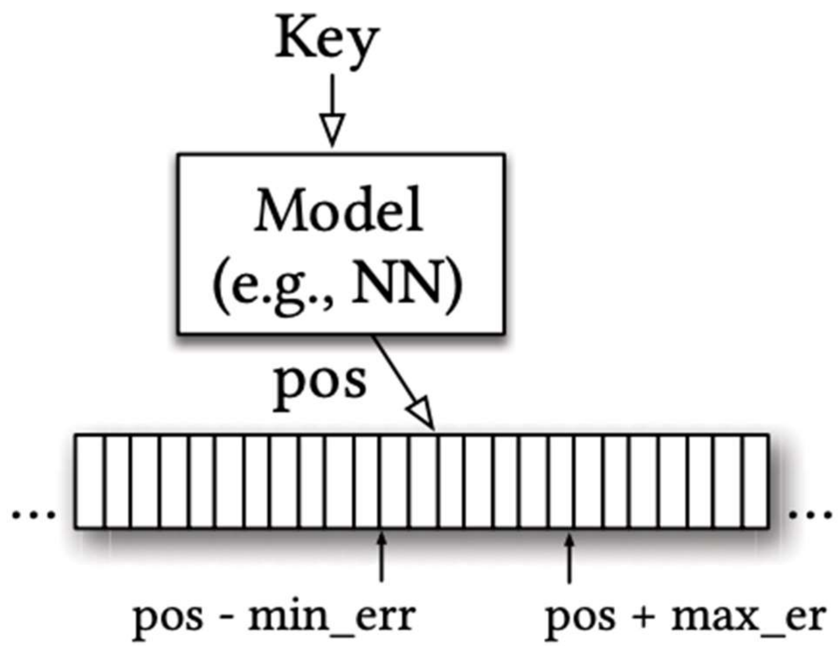
Q2: How could we know the lookup key position on unsorted underlying data?

Learned Index

The problem is:

1. We want to know the position on **unsorted** underlying data

1. We still need to use **binary search**



What is included

1. Brief review of traditional indexing
2. Introduction to learned index
3. **Learned secondary index**

    3.1 Permutation vector

    3.2 Fingerprint vector

    3.3 How to build LSI

1. Lookup procedure
2. Evaluation

What is included

1. Brief review of traditional indexing
2. Introduction to learned index
3. **Learned secondary index**

   3.1 Permutation vector

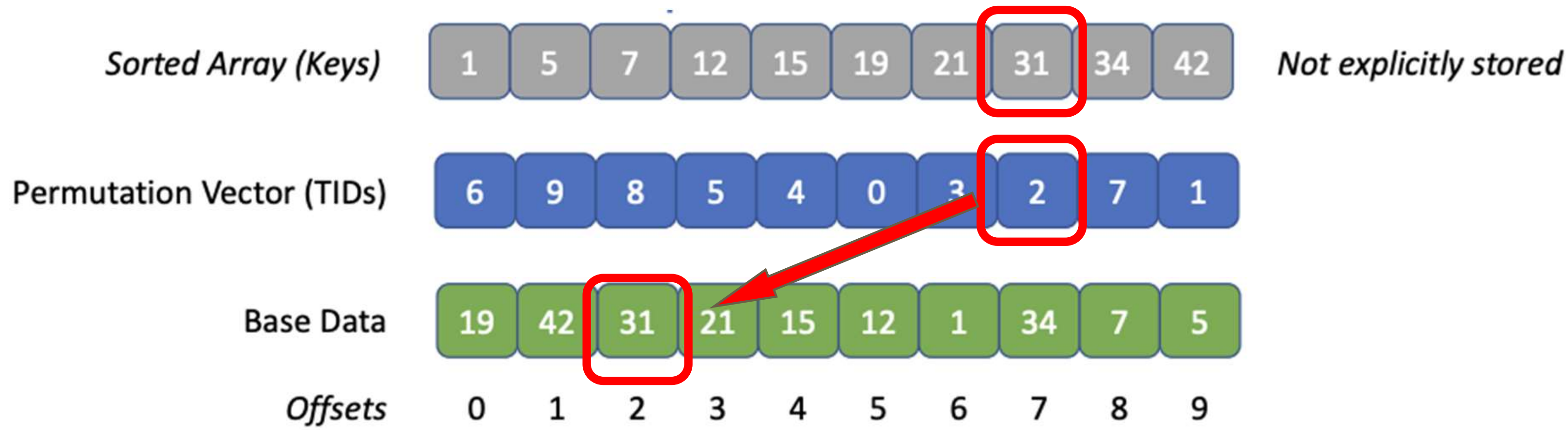
   3.2 Fingerprint vector

   3.3 How to build LSI

1. Lookup procedure
2. Evaluation

Permutation Vector

Permutation vector provides a mapping from unsorted data to a sorted view.



We can use the permutation vector to locate the actual record position.

Use bit-pack to store Permutation Vector

Assume we have: **n** (0 ~ n-1) elements

maximum value to represent : n-1

We need $\log_2(n)$ bits at most → The logarithm base is 2 because data storage is based on binary systems

E.g: n = 1024
Bitpack: take 10 bits at most

Else: (assume we use32 bits integer): 32 - 10 = 22 bits wasted

Traditional Index vs LSI(using PV)

Saves about 6 * spaces

	B+ Tree	LSI
Space	Store Actual Keys	Use PV to map PLEX's predictions into the underlying data.
Time(binary search)	Sorted array (faster)	Unsorted array (slower)

All memory accesses to the unsorted base data are likely out of cache.

What is included

1. Brief review of traditional indexing
2. Introduction to learned index
3. Learned secondary index

    3.1 Permutation vector

    3.2 Fingerprint vector

    3.3 How to build LSI

1. Lookup procedure
2. Evaluation

Fingerprint Vector

In equality lookups: → Linear Search

↓

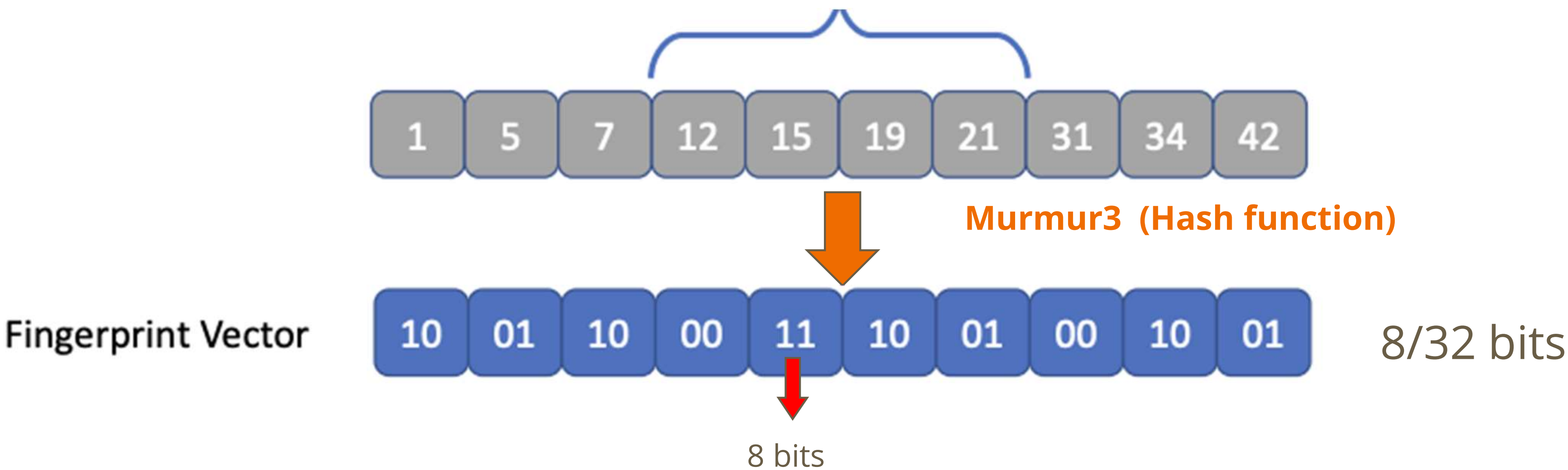
Max error bound = 1

Why not linear search? → Traverse the data

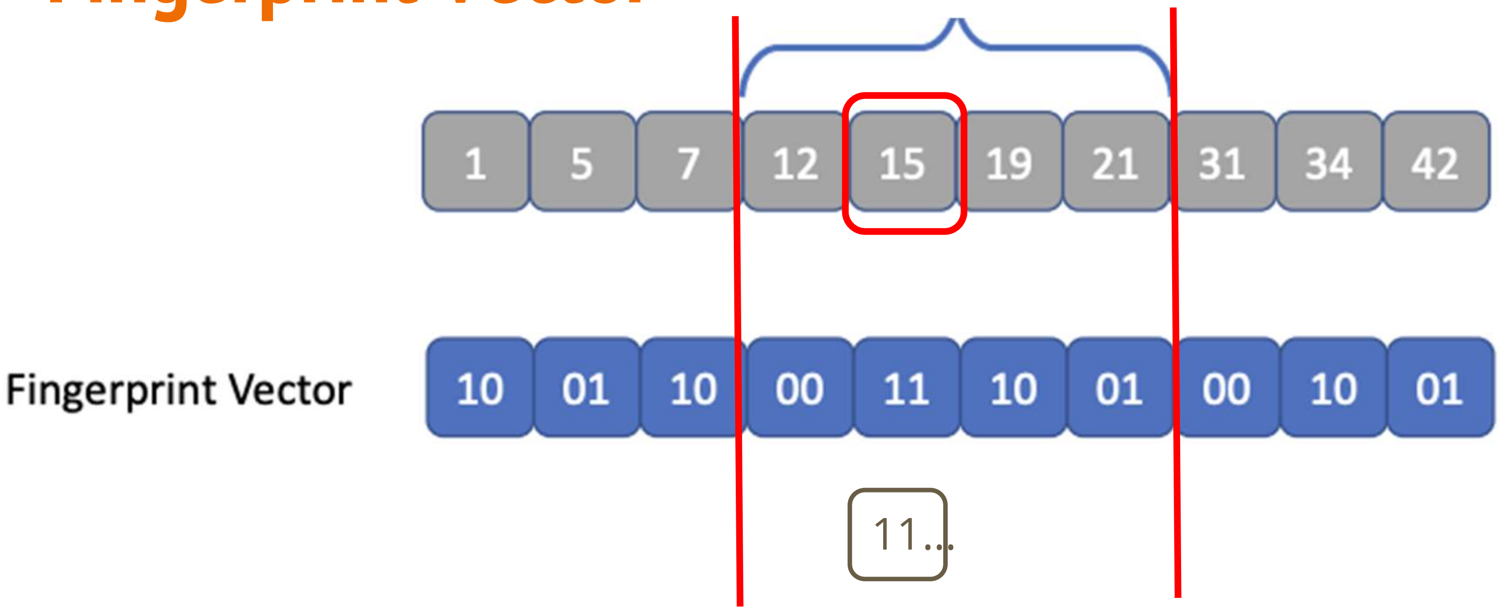
Fingerprint Vector

Fingerprint → Person

Fingerprint Vector → Real Key



Fingerprint Vector



Match the fingerprint then check the key

What is included

1. Brief review of traditional indexing
2. Introduction to learned index
3. Learned secondary index

    3.1 Permutation vector

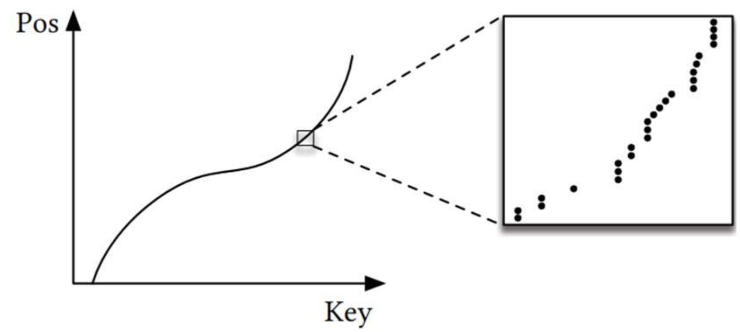
    3.2 Fingerprint vector

    3.3 How to build LSI

1. Lookup procedure
2. Evaluation

How to build LSI

Step1: Create a sorted copy of the base data.



Step2: Build CDF based on the sorted data

Duplicate makes a “steeper” slope in the CDF

Step3: Deduplicate but give the data a rank of weight

Step4: Build PLEX model based on CDF

Step5: Build Permutation Vector & Fingerprint Vector

Step6: Delete the copy

Two important parameter to set

Maximum error: make sure the model is reliable

Fingerprint bits: save space or save accuracy?

What is included

1. Brief review of traditional indexing
2. Introduction to learned index
3. Learned secondary index

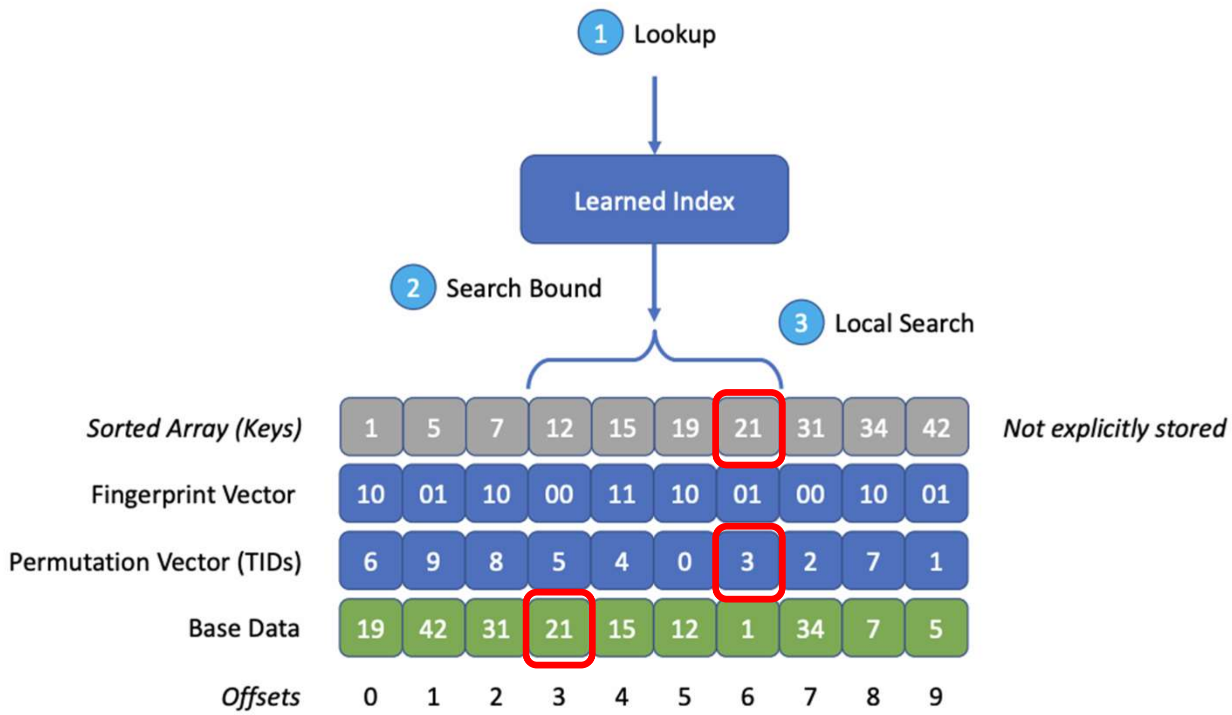
    3.1 Permutation vector

    3.2 Fingerprint vector

    3.3 How to build LSI

1. **Lookup procedure**
2. Evaluation

Lookup procedure



What is included

1. Brief review of traditional indexing
2. Introduction to learned index
3. Learned secondary index

    3.1 Permutation vector

    3.2 Fingerprint vector

    3.3 How to build LSI

1. Lookup procedure
2. **Evaluation**

Evaluation of Learned Secondary Index (LSI)

- Testing Environment
- Datasets
- Baselines for Comparison

Index Build Times Comparison

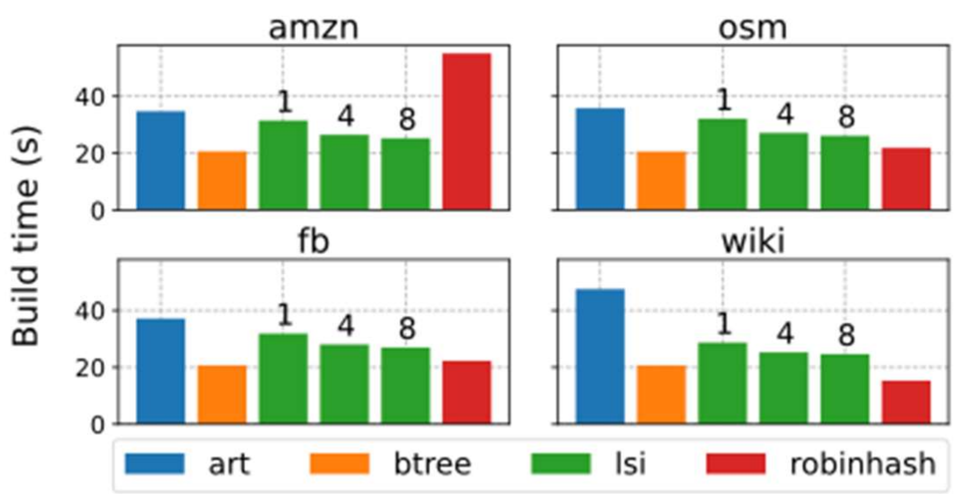


Figure 2: Build time in seconds. The text annotations denote the error bounds.

Lower-Bound Lookups Performance

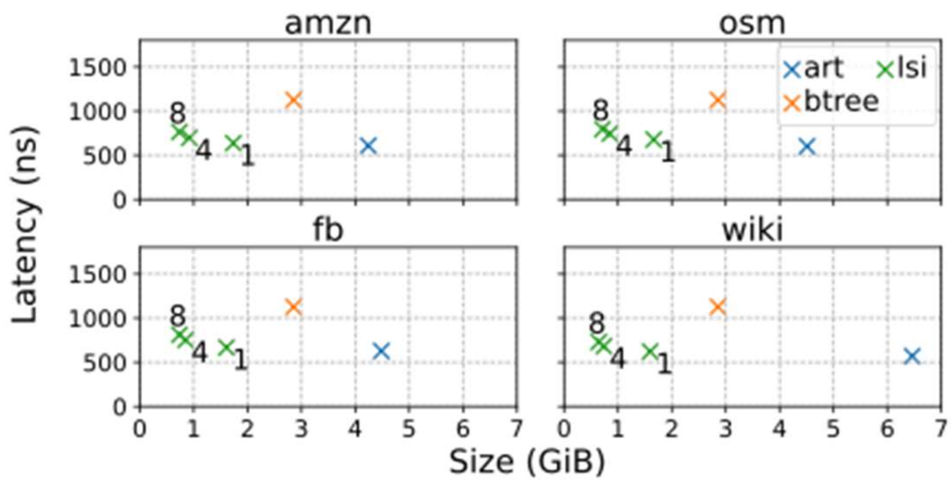


Figure 3: Lower-bound lookups using non-existing keys. The text annotations denote the error bounds.

Equality Lookups - LSI vs. RobinHash

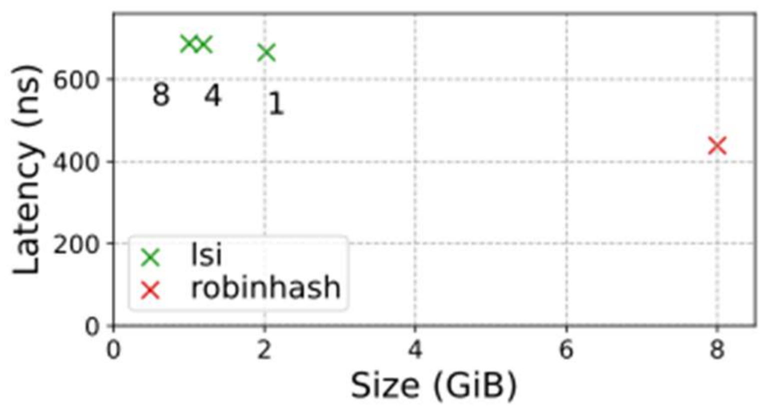


Figure 4: Equality lookups on the amzn dataset comparing LSI to RobinHash.

Binary vs. Linear Search in LSI

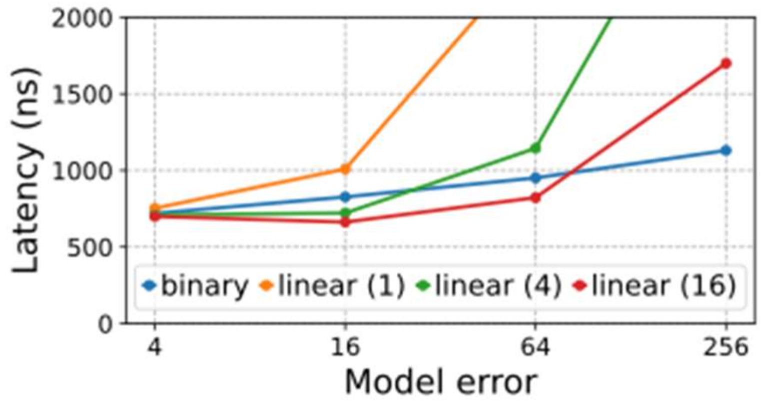


Figure 5: Binary search vs. linear search with varying fingerprint sizes (in brackets).

Space Breakdown of LSI

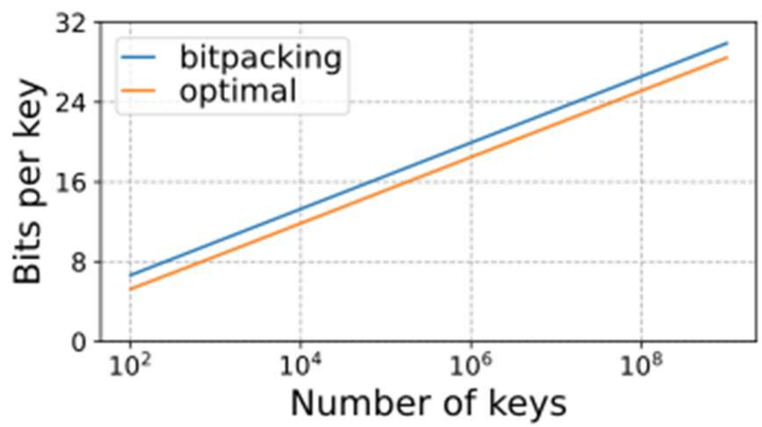


Figure 6: Size of the permutation vector. Information-theoretic lower bound vs. our bit-packed representation.

Impact of Model Choice on LSI Performance

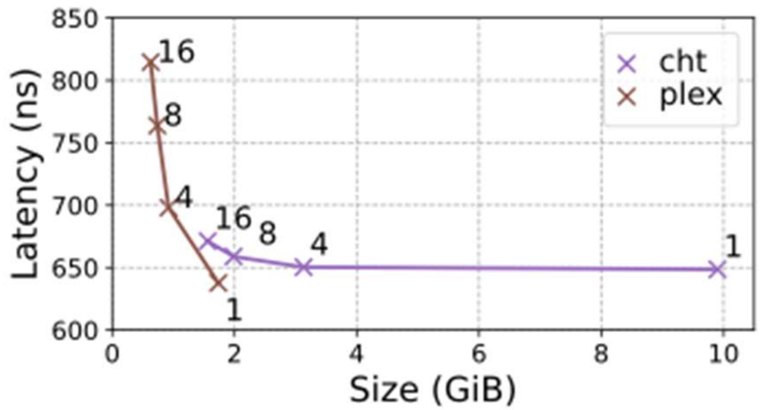


Figure 7: Using PLEX vs. CHT as models in LSI for lower-bound lookups. The text annotations denote the error bounds.