

The background of the slide features a wide-angle photograph of a mountain range. The peaks are covered in snow, and the lower slopes are densely forested with dark green trees. The sky is a mix of blue and warm orange/pink hues, suggesting either a sunrise or sunset. A full moon is positioned in the upper right corner of the image.

The Case for Learned Index Structures

Rui Yang

4/10/2023

Outline

- Background on Traditional Index Structures in DBMS
 - Why do we even need index structures at all?
 - B-Tree & CDF Model
- Learned Index Structures (LIS)
 - Naïve Approach (A single NNR)
 - Reclusive Model Index (RMI)
- Conclusion

Fundamental Building Blocks of Database Systems

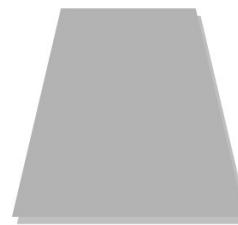


Index Structures

B-Tree



HashMap



Blooming Filter



- Purpose of having these structures in DBMS?
- Hit: Tradeoff between speed and storage
- Scarifies storage reduce the # of blocks to read

Accessing Data Without Index Structures

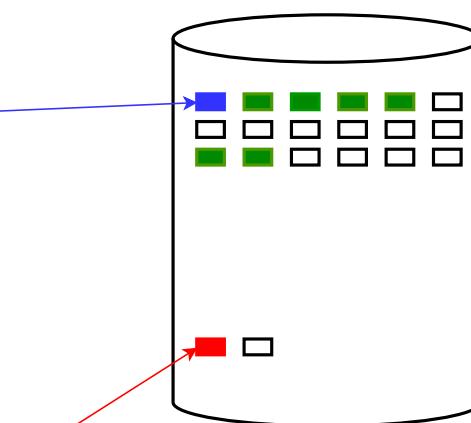
UVA DS School Info Table

SID	s_name	s_major	GPA
1	student_1	CS	4.0
2	student_2	DS	4.0

:

SID	s_name	s_major	GPA
98	student_98	DA	4.0
99	student_99	CS	4.0

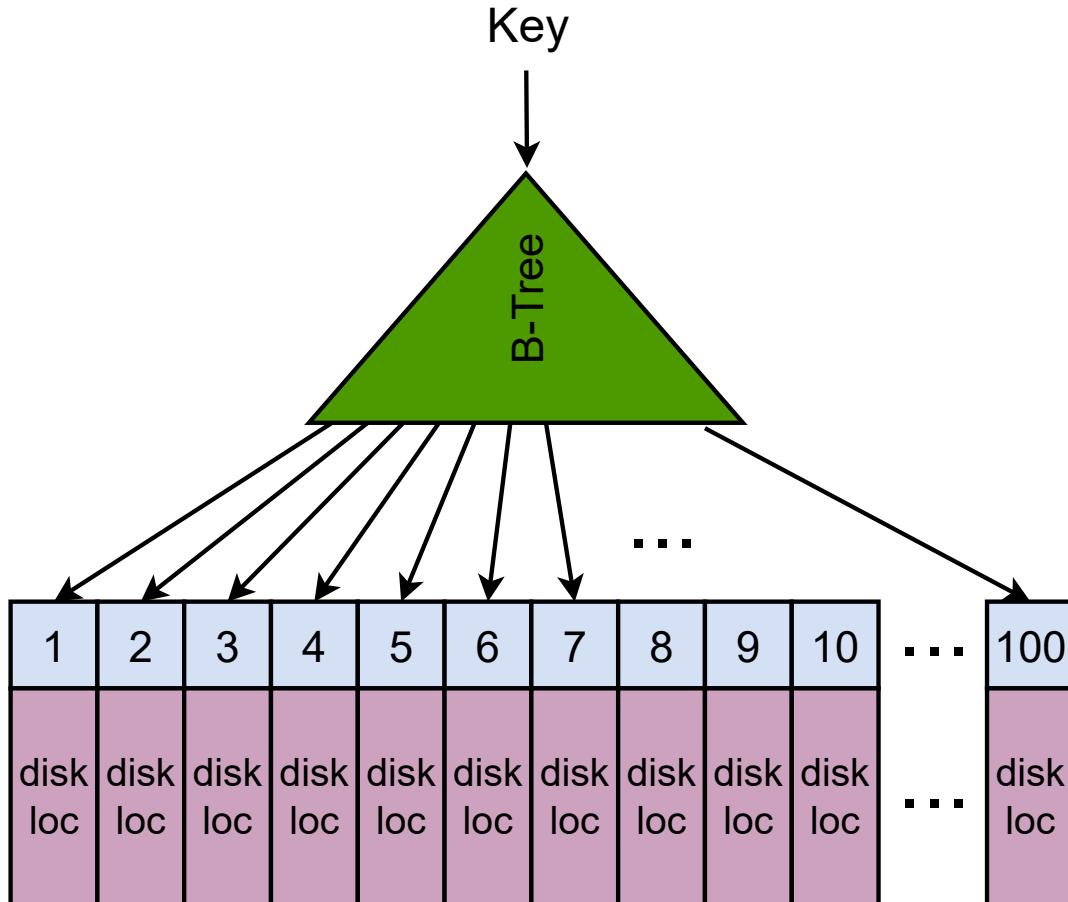
100 student_100 DS 4.0



100 Entries
2 Entries per Block
50 Blocks

- To access $SID==1$ entry we need read at most **50 blocks** from disk. **Too slow!**
- Can we reduce the number of blocks to read?

Indexing SID using B-Tree



Prepare a B-Tree index structure for SID from 1 to 100

1	2	3	4	5	6	7	8	9	10	...	100
---	---	---	---	---	---	---	---	---	----	-----	-----

Time: $O(\log n)$
Space: $O(n)$

B-Tree Operations

- Operations: *INSERT()*, *LOOKUP()*, *DELETE()*, *UPDATE()*
- *LOOKUP()* walk through example
 - Visualization : <https://people.ksp.sk/~kuko/gnarley-trees/Btree.html#>

- No assumption about data distribution
- Knowing data distribution may increase performance significantly from both speed and storage.

Indexing all integers from 1 to 100

1	2	3	4	5	6	7	8	9	10	...	100
---	---	---	---	---	---	---	---	---	----	-----	-----

array[lookup - 1]

Indexing all even integers from 2 to 100

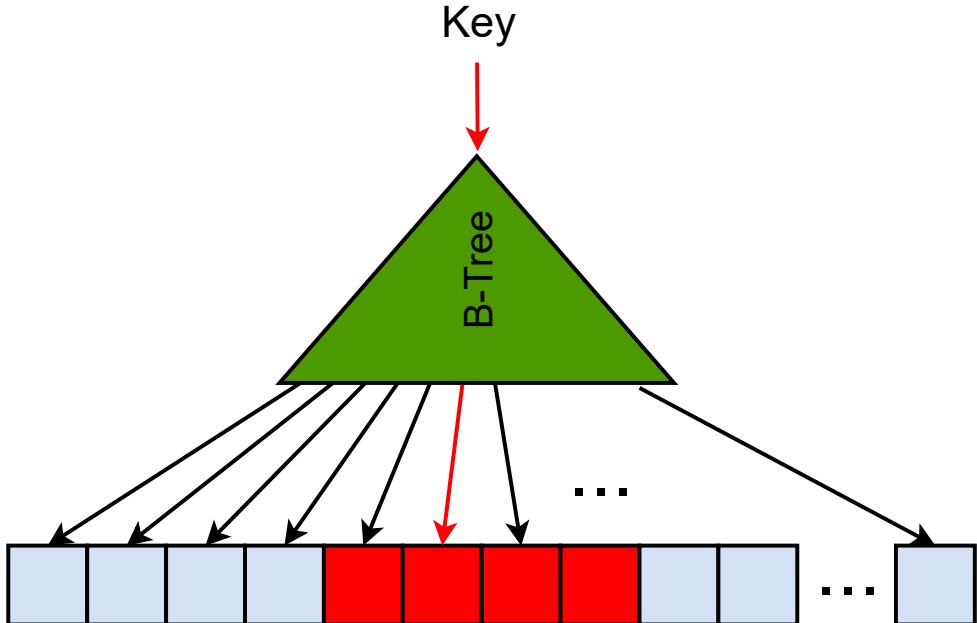
2	4	6	8	10	12	14	16	18	20	...	100
---	---	---	---	----	----	----	----	----	----	-----	-----

array[(lookup - 2) / 2]

A B-Tree is A Model

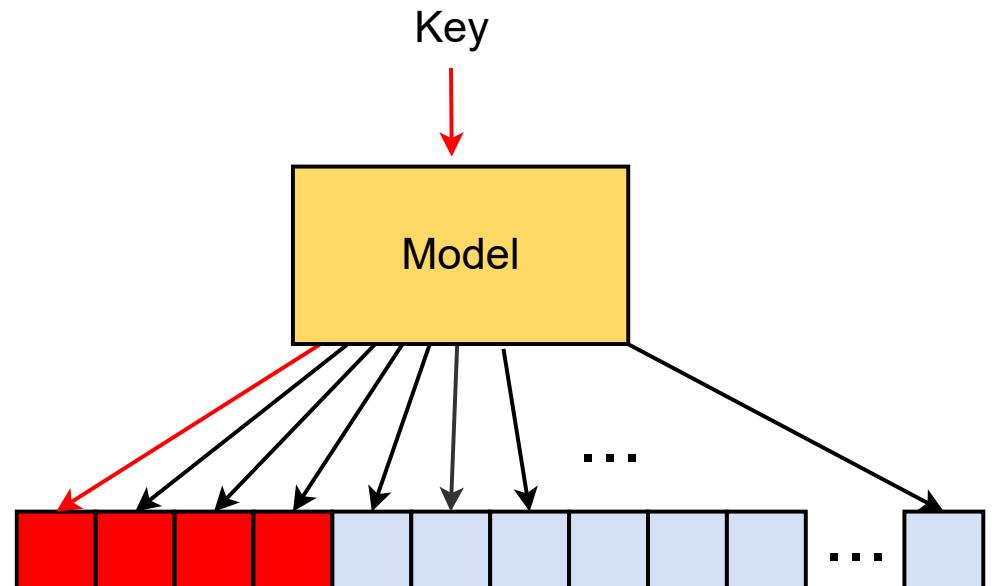
B-Tree

1. Locate the pos of input key
2. Binary search within a page size



Alternative view

1. Locate the pos of input key
2. Binary search within error boundaries



Assuming data are stored in dense array in sorted order

Modeling B-Tree Functionality using CDF

- B-Tree indexes the data in a sorted order
 - $Pos = B\text{-Tree}(key)$
- CDF gives the probability of X that will have a value less than or equal to x
 - $Pos = CDF(Key) * N$
- Data distribution visualization: <https://statdist.com/>

If we can learn the CDF model of a given dataset, we can replace the B-Tree index structure with learned model

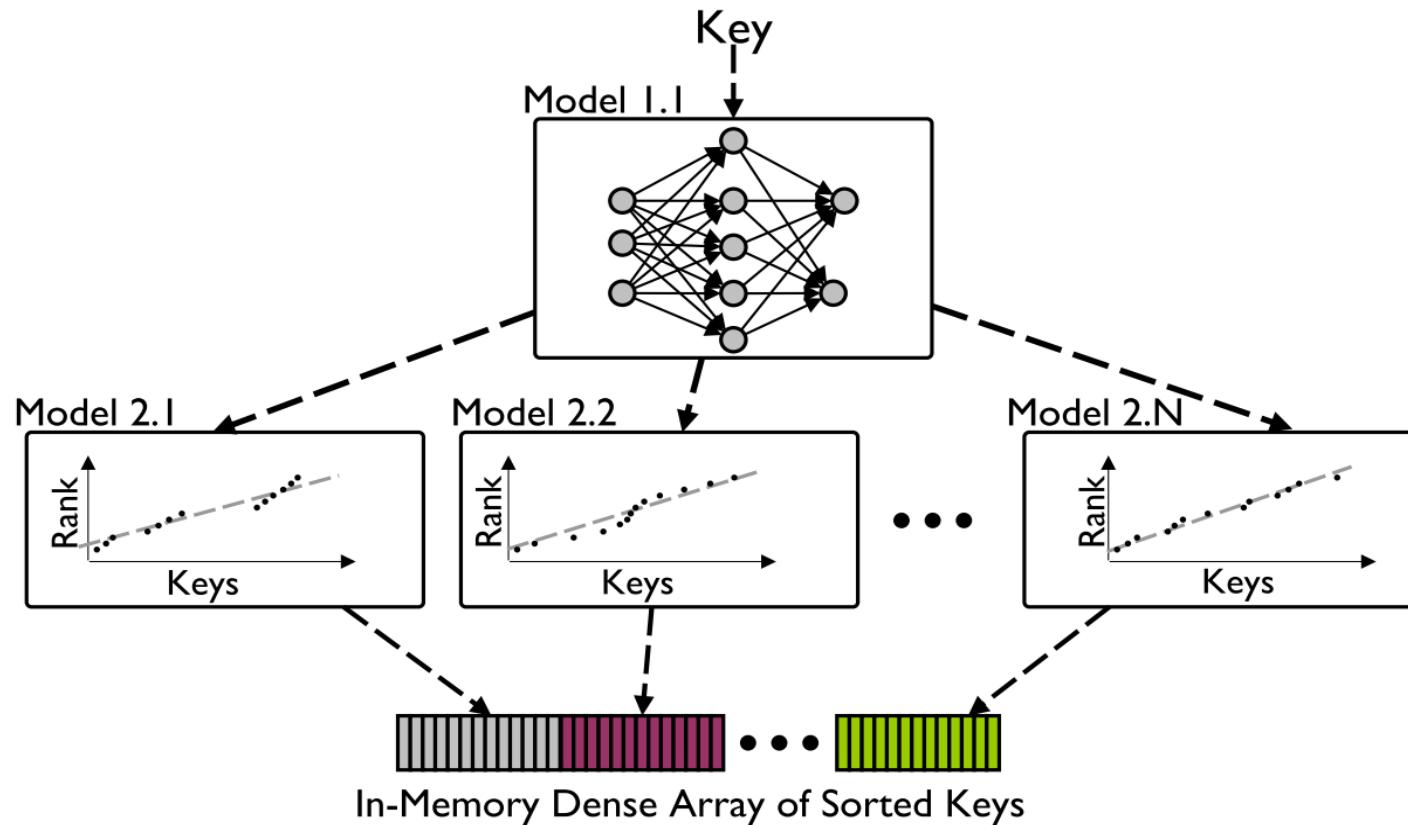
Which ML Model?

- $Pos = CDF(Key) * N$
 - Approximate the position given a key inside a sorted array
 - Learn the relationship between *Key* and *Pos* -> *Regression*
 - Position range: 0 to $N-1$
 - Key range: smallest to largest value
- Which regression model should we pick and why?
 - Linear Regression, Neural Network Regression, and etc.
- Discussion (2 mins)

Naïve LIS

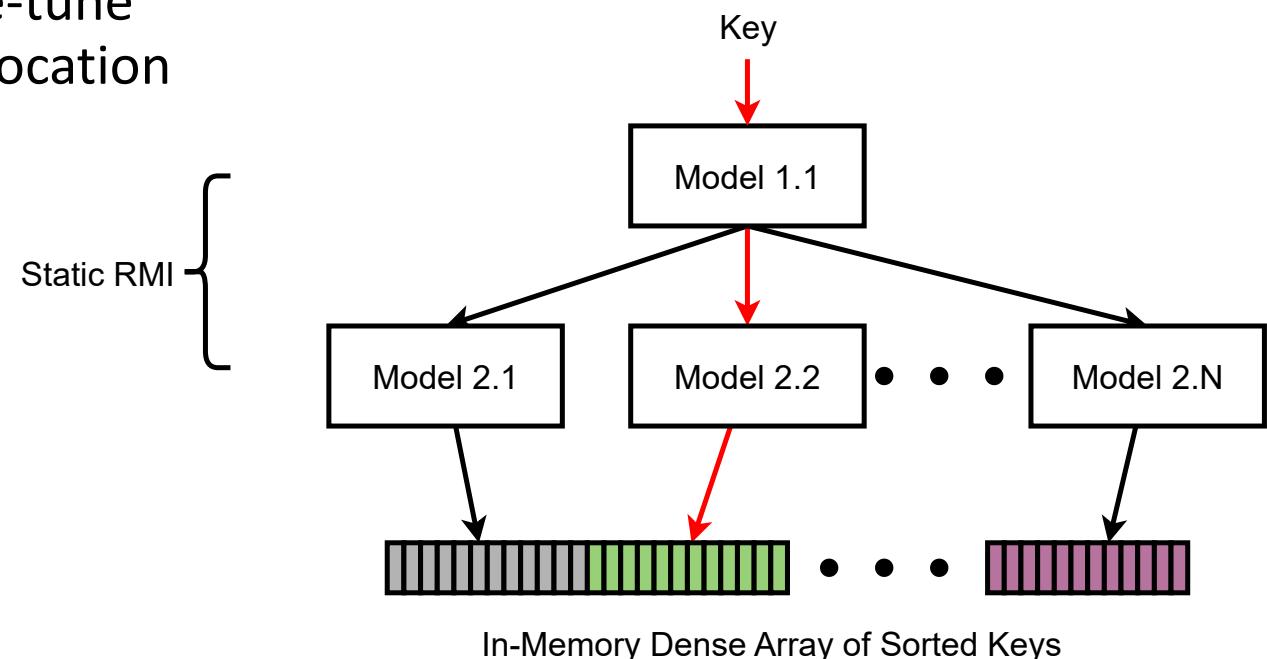
- Use a single neural network regression model
- Good at approximate the general shape of a CDF
- Large error at the last mile of predicting the actual position
- Solution: RMI

Two-stage-RMI

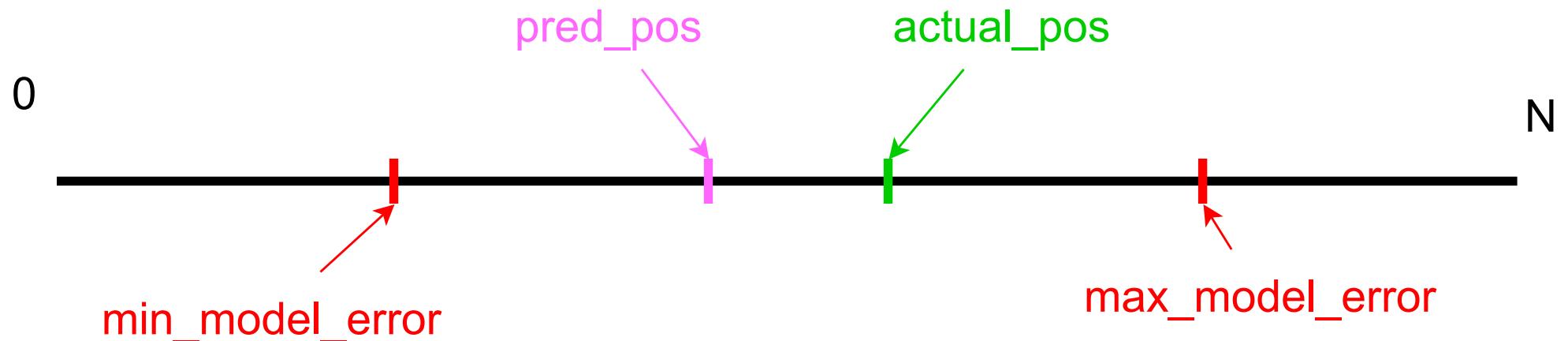


Lookup() on A Two-stage-RMI

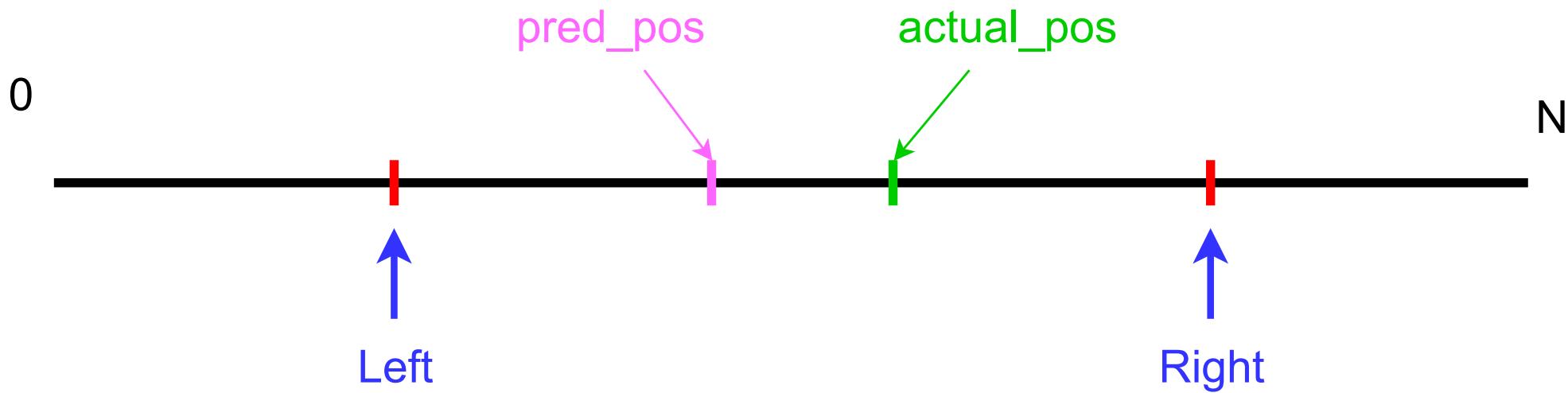
- Two stage RMI
- A higher stage model directs a lookup operation to a lower stage model to fine-tune the precision of the predicted memory location
- `learned_index_lookup(key)`
 - `ret_1 = first_stage_lookup(key)`
 - `ret_2 = second_stage_lookup(ret_1)`
 - `predicted_pos = array[ret_2]`



Local Search



Binary Search



Conclusion

- DBMS index structures
 - B-Tree & CDF model
- LIS
 - Abstracting functionality of B-Tree using regression models
 - Naïve approach: using single regression model
 - RMI: a hierarchical architecture
 - Performing a local search if the predicted pos is off actual pos
 - Using binary search or other search algorithms to find the actual key