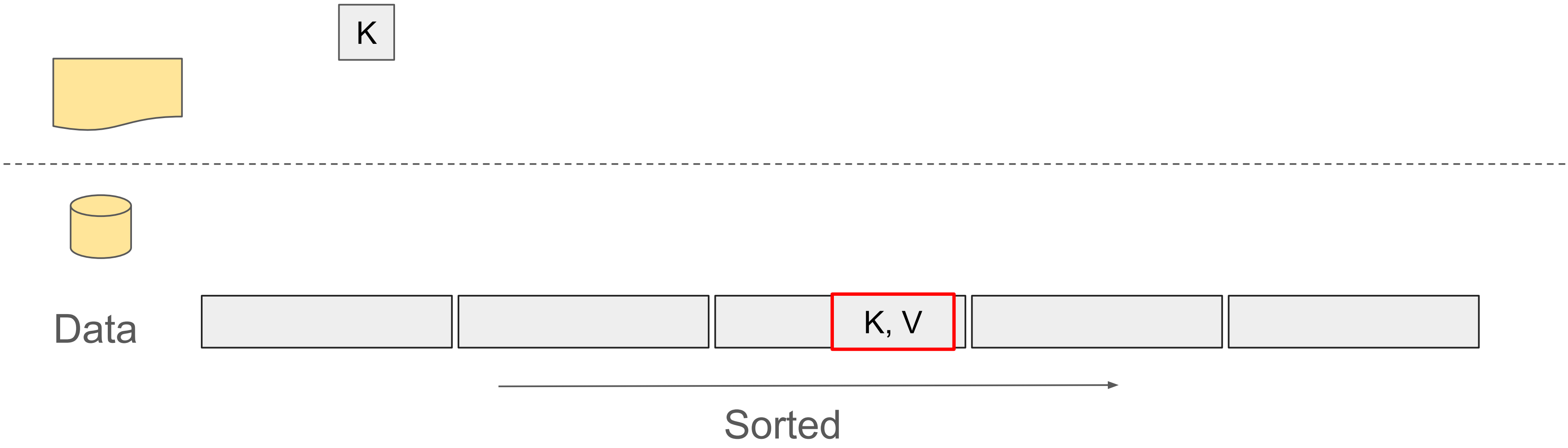
Learned Indexes

CS 6530, Fall 2024
Yuvaraj Chesetti

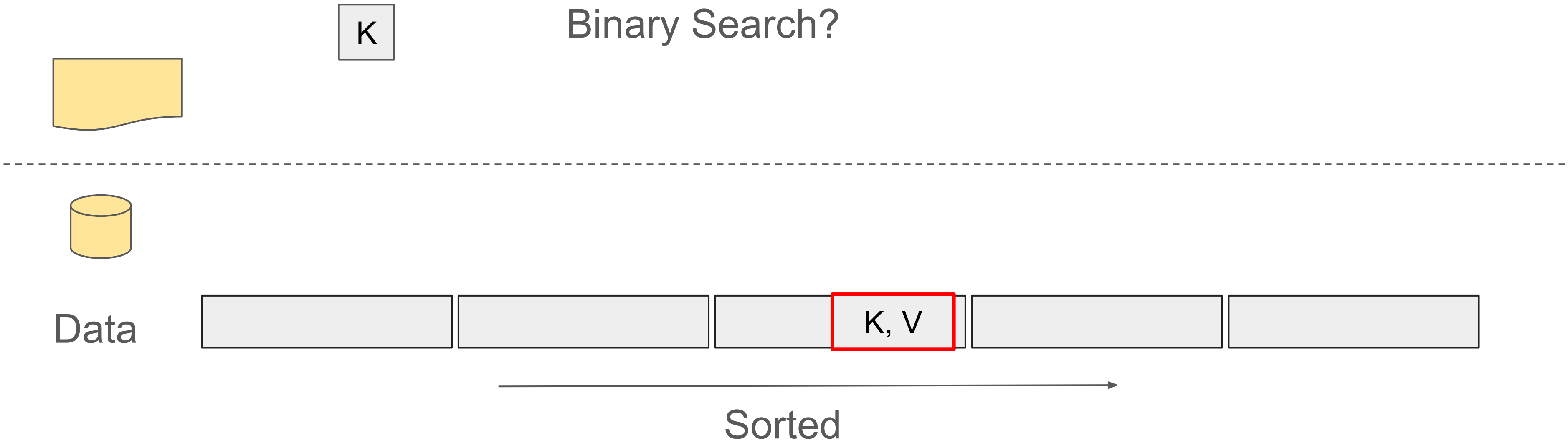
What is an Index?

Task: Retrieve V, given K



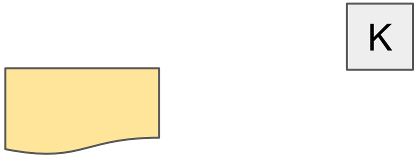
What is an Index?

Task: Retrieve V, given K

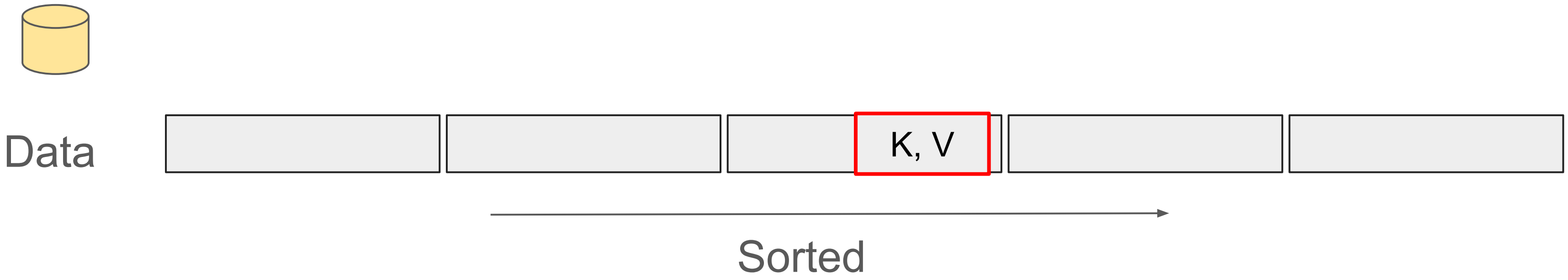


Indexes

Task: Retrieve V, given K

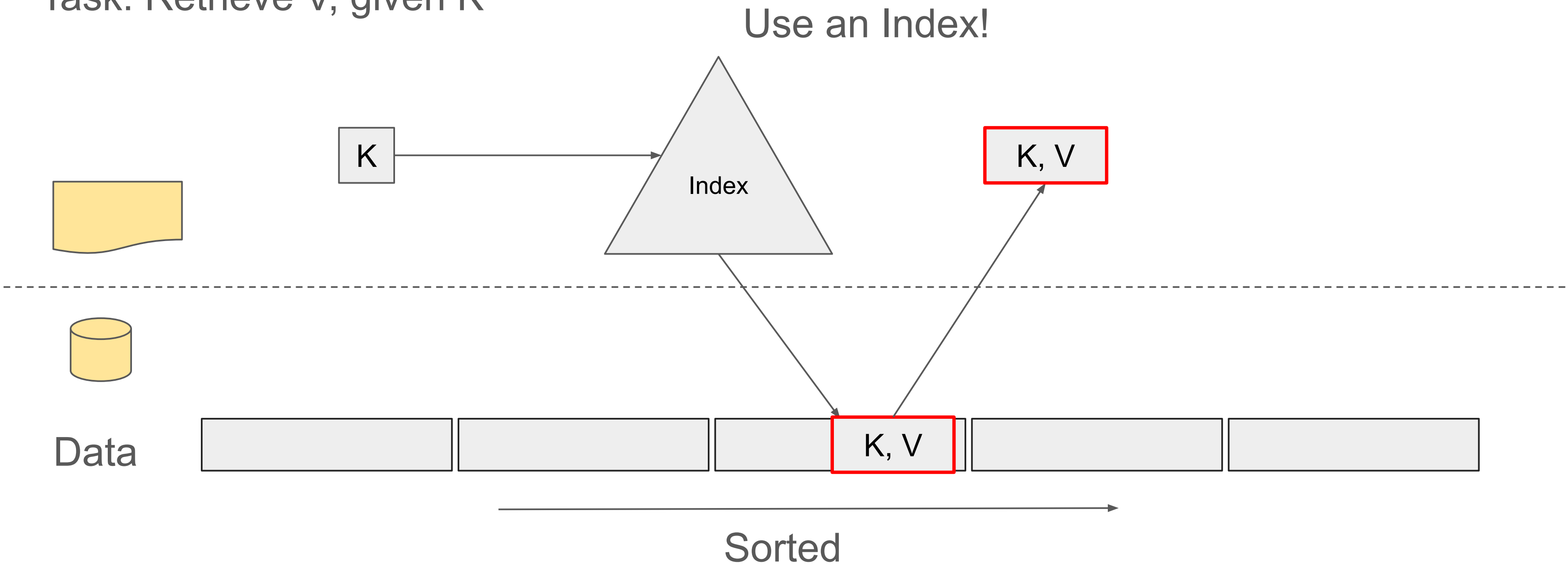


Binary Search?
Too Slow - $\log_2(N)$ I/Os



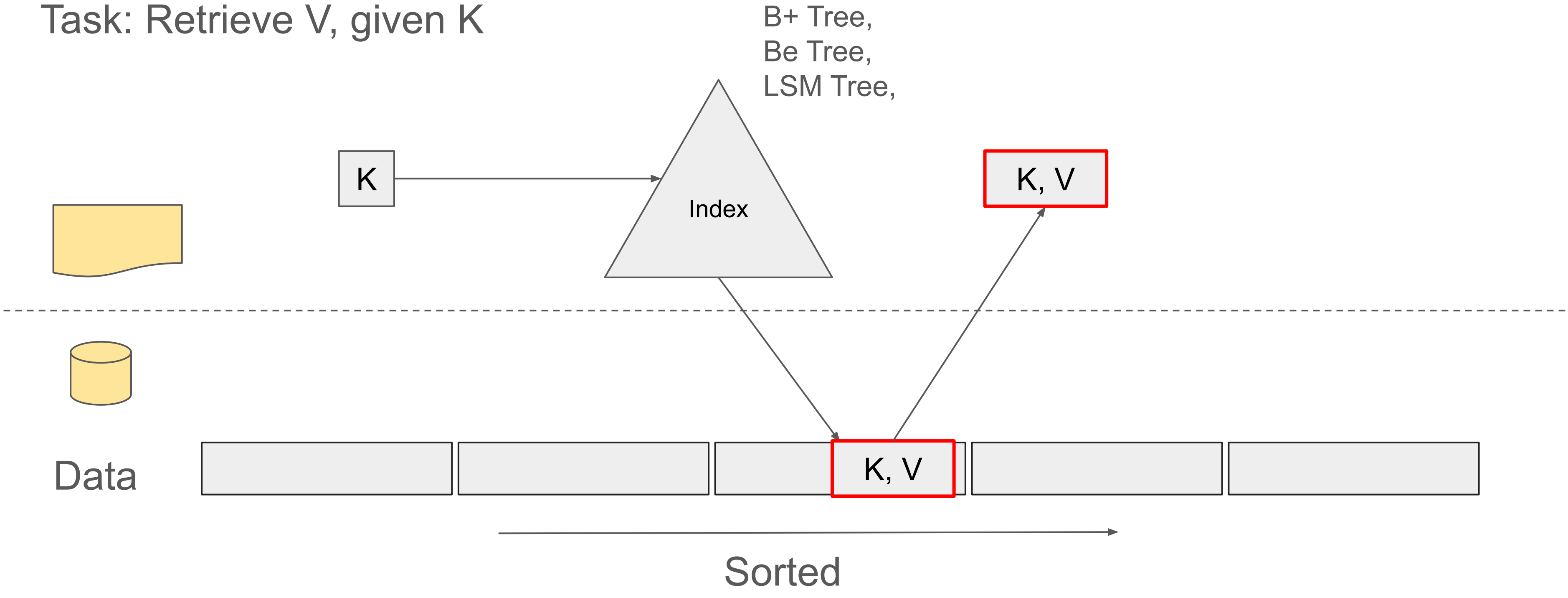
Indexes

Task: Retrieve V, given K



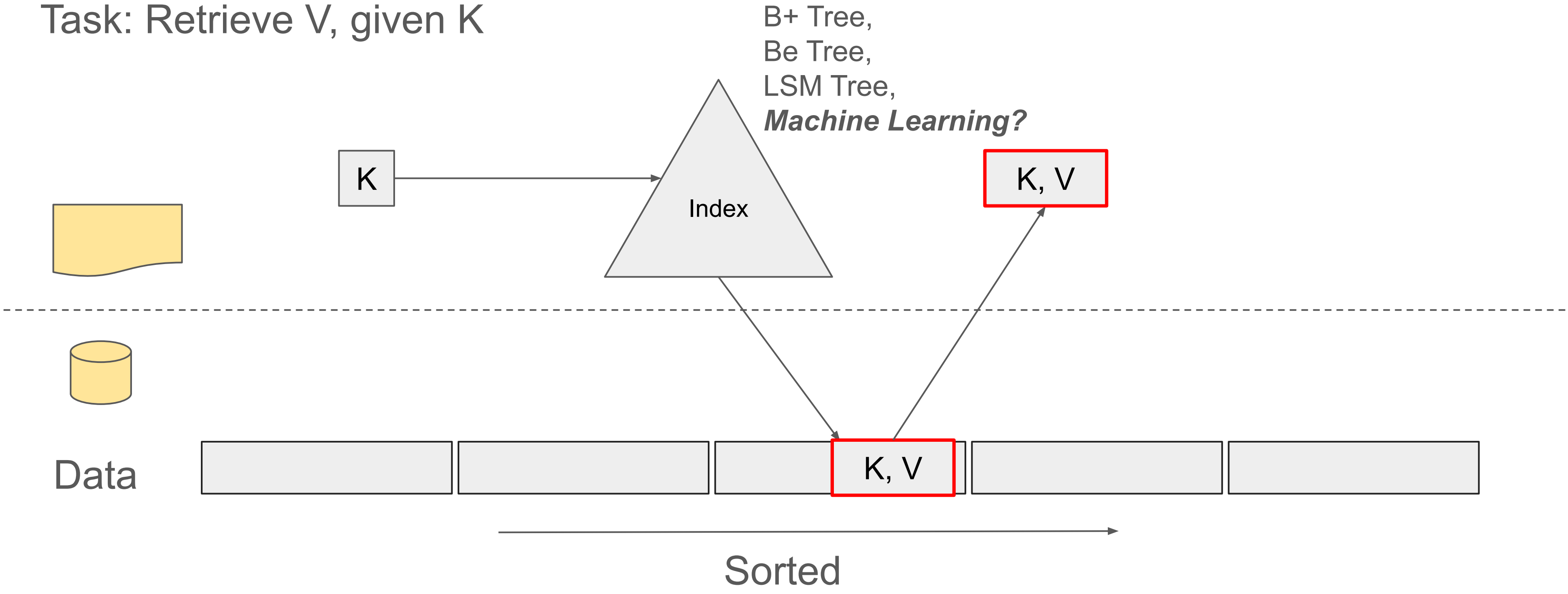
Indexes

Task: Retrieve V, given K



Indexes

Task: Retrieve V, given K



Learn Data Distribution

Task: Retrieve V, given K

Q: If the data followed a pattern, do you need a index?

Learn Data Distribution

Task: Retrieve V, given K

Q: If the data followed a pattern, do you need a index?

Data					
	0, 2, 4, 6	8, 10, 12, 14	16, 18, 20, 22	24, 26, 28, 30	32, 34, 36, 38

Sorted

Learn Data Distribution

Task: Retrieve V, given K

Q: If the data followed a pattern, do you need a index?

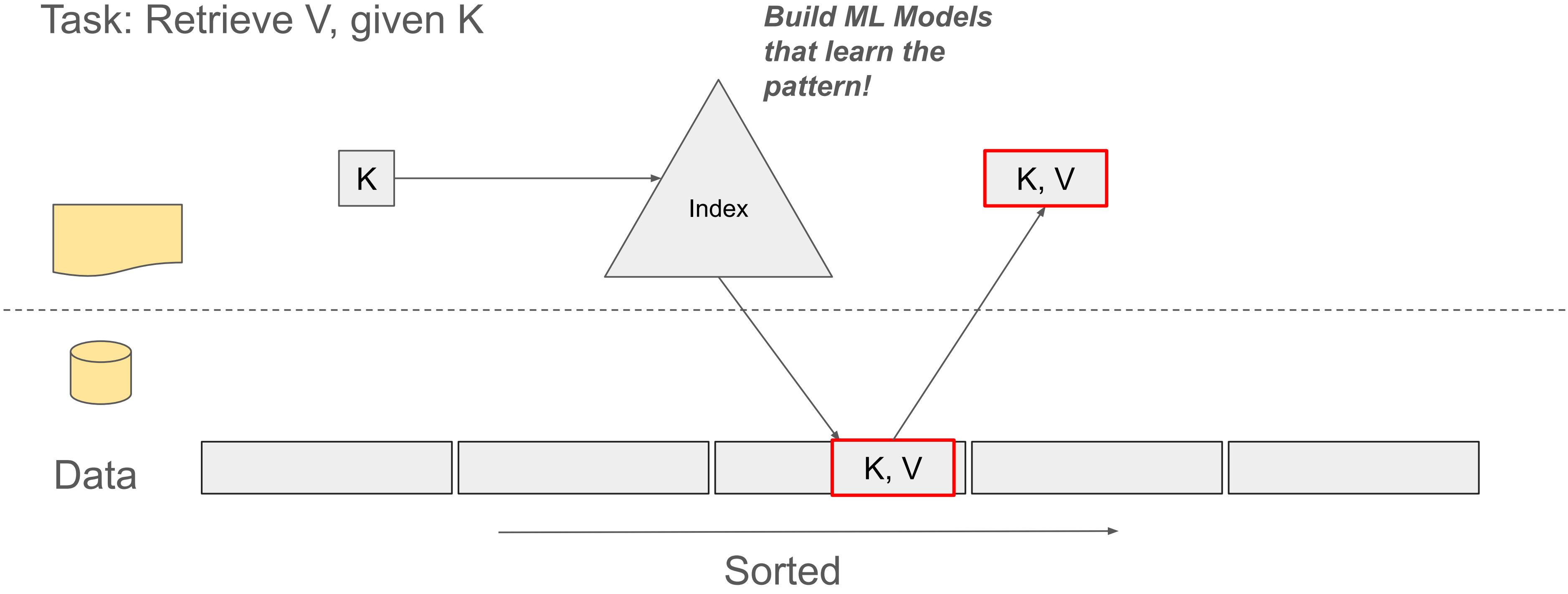
No, O(1) Disk Access

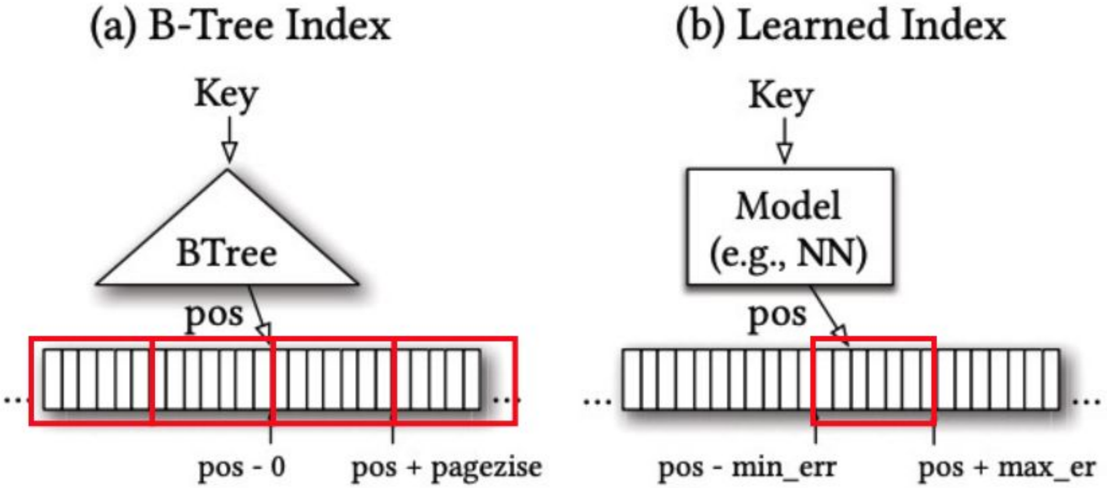
Data					
	0, 2, 4, 6	8, 10, 12, 14	16, 18, 20, 22	24, 26, 28, 30	32, 34, 36, 38

Sorted

Indexes

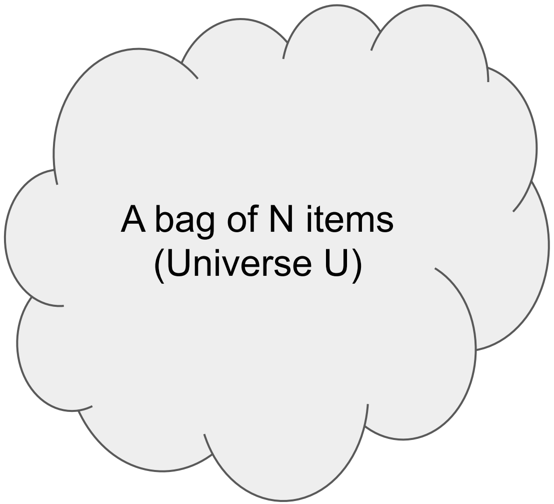
Task: Retrieve V, given K



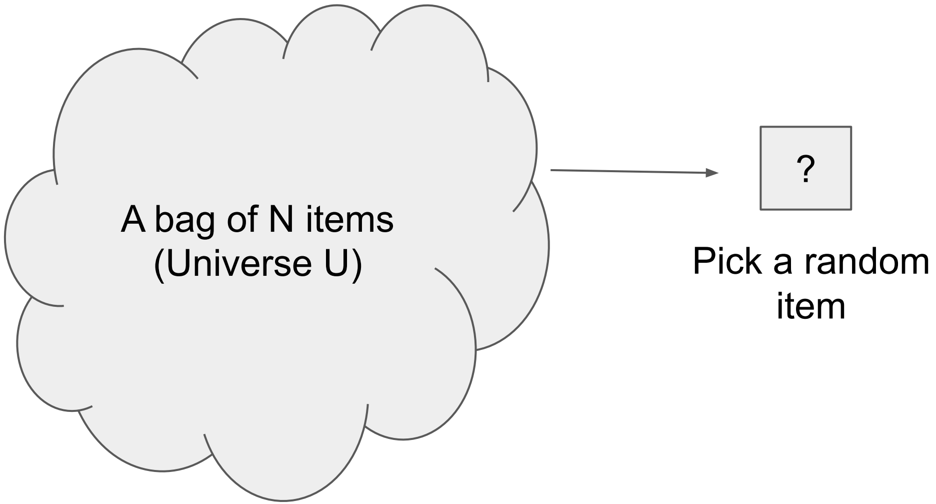

(a) B-Tree Index
Key
BTree
pos
pos - 0
pos + pagezise
(b) Learned Index
Key
Model
(e.g., NN)
pos
pos - min_err
pos + max_er

What to Learn?

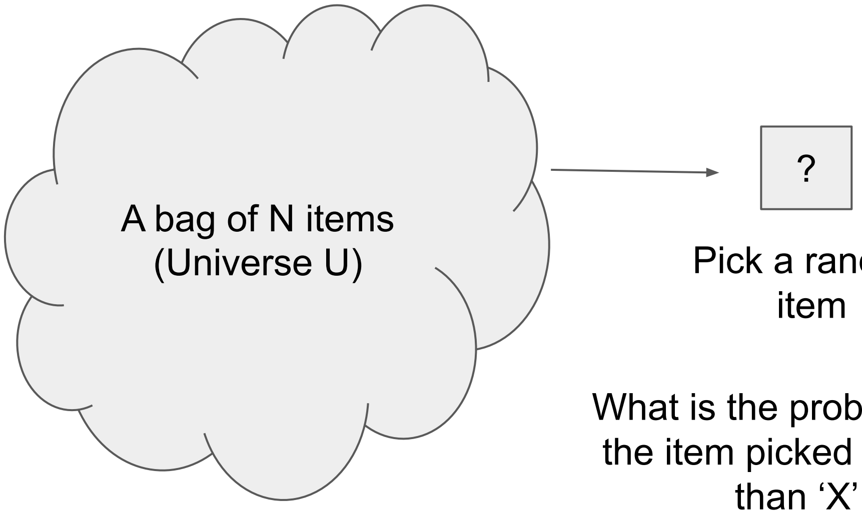
Cumulative Distributive Function



Cumulative Distributive Function



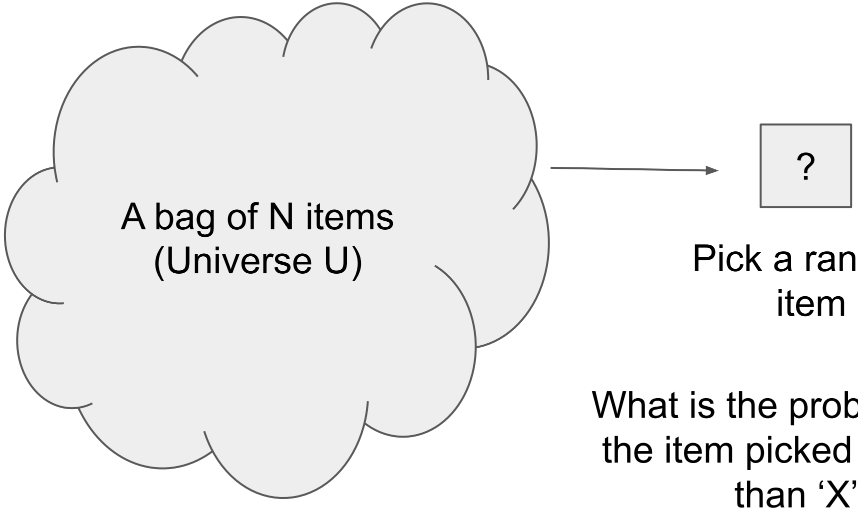
Cumulative Distributive Function



Pick a random item

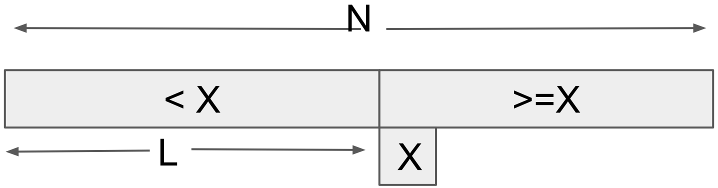
What is the probability that the item picked is smaller than ‘X’ ?

Cumulative Distributive Function



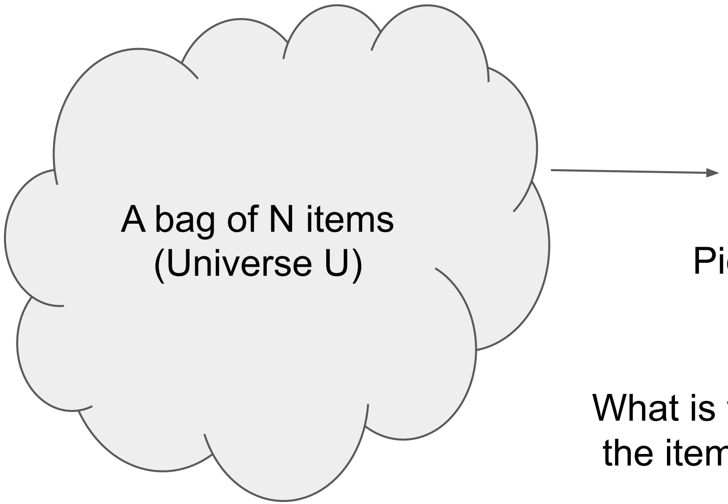
Pick a random item

What is the probability that the item picked is smaller than ‘X’ ?



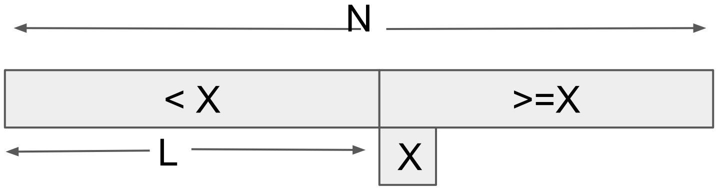
P(‘?’ < X) = L/N

Cumulative Distributive Function



Pick a random item

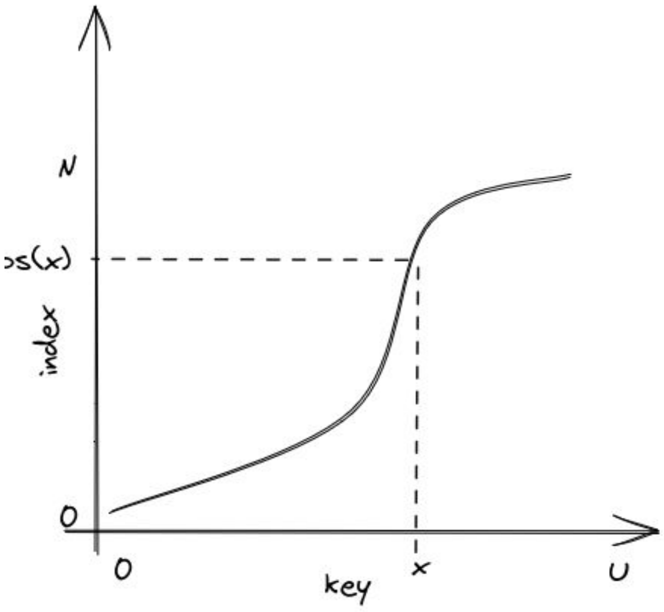
What is the probability that the item picked is smaller than ‘X’ ?



P(‘?’ < X) = L/N

This is the position of ‘X’ if items in the bag were laid out sorted!

Cumulative Distributive Function

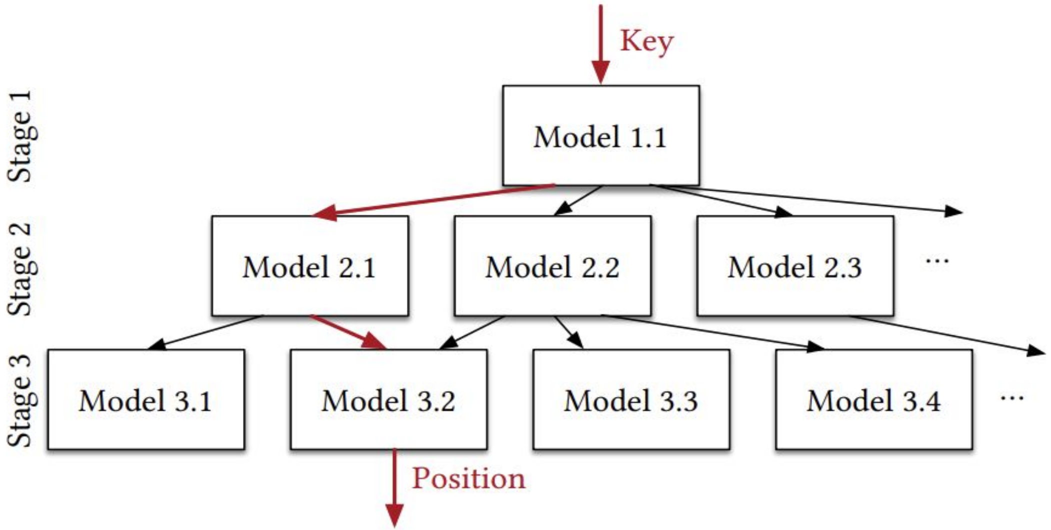


Perfect Index (C.D.F)

- P('?' < X) = L/N is called the Cumulative Distribution Function
- Monotonic increasing function
- Learn the distribution = Learn the C.D.F
- All Indexes learn the C.D.F (Even B+ Trees!)

How to Learn?

Top Down - Recursive Model Index (RMI)

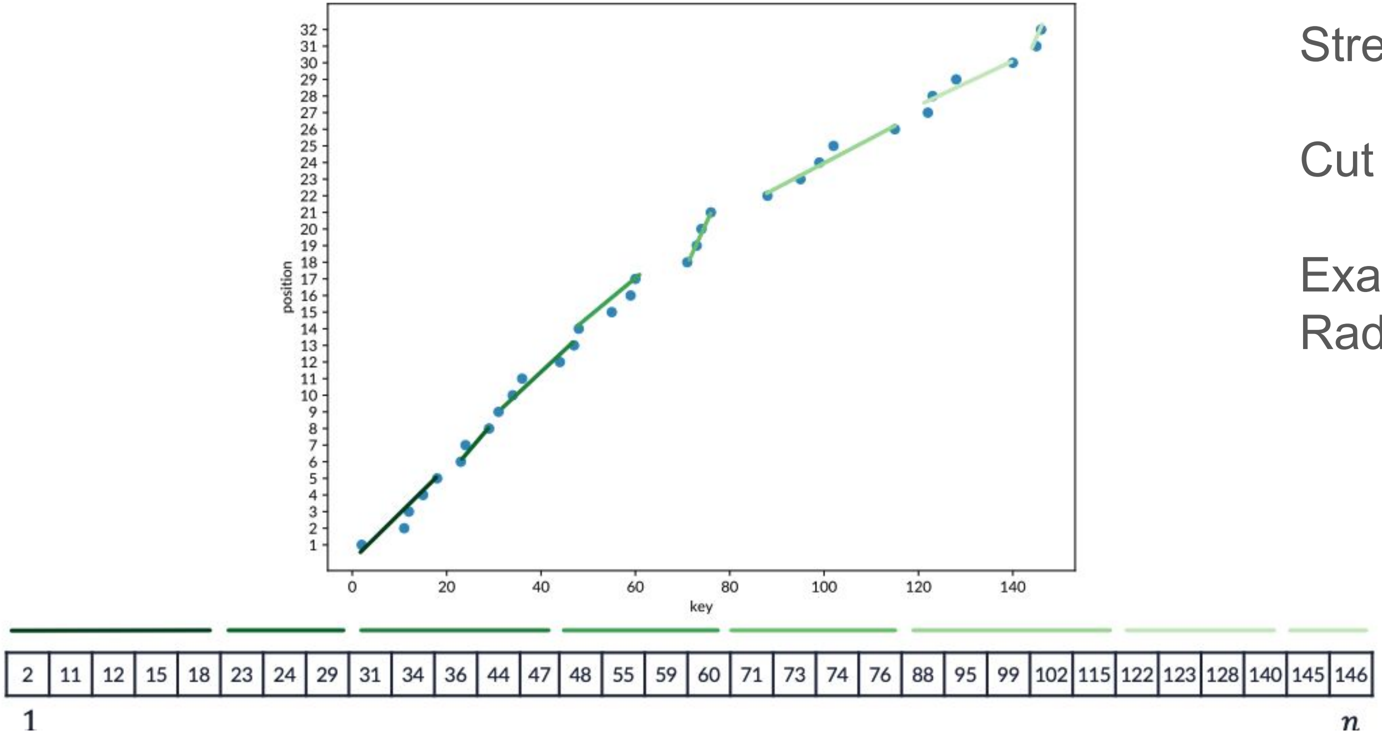


Start with a spec,
Search for a optimal layout

Models:

- Linear f(x)
- Non Linear f(x)
- Neural Networks
- B+ Trees

Bottom Up - Piecewise Linear Regression



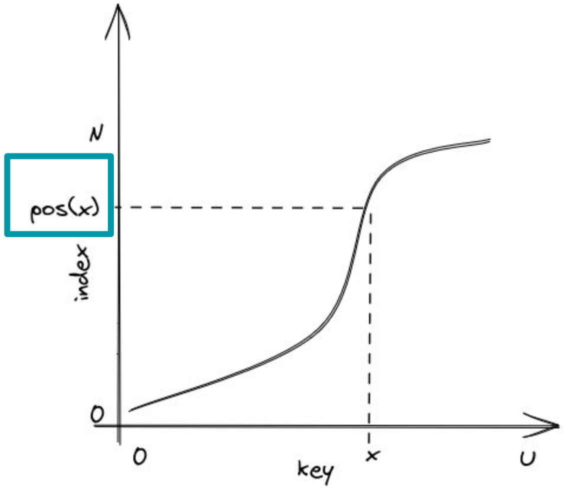
Stream sorted data

Cut Line Segments

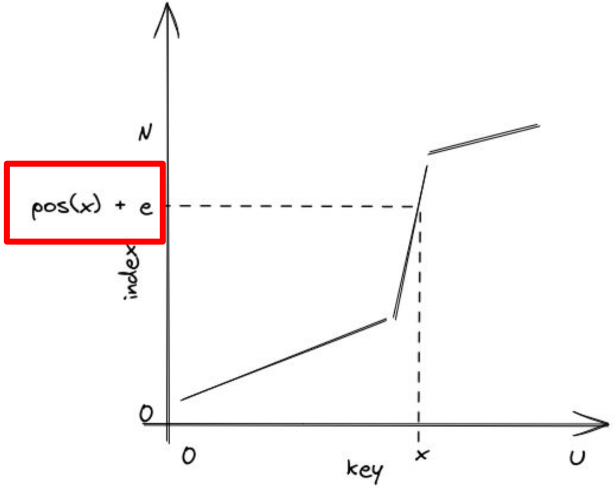
Examples:
RadixSpline, PGM

Fig Source: https://pgm.di.unipi.it/slides-pgm-index-vldb.pdf

What about error?



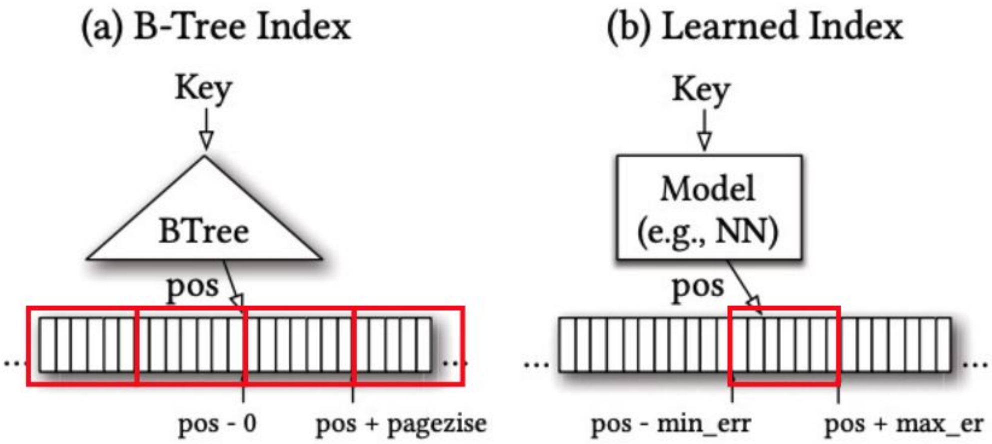
Perfect Index (C.D.F)



Learned Index (P.L.R of C.D.F.)

All Indexes have error (even B+ Trees)

- All indexes have error!
- B-Tree has an error of page-size
- Error can be bounded (at cost of Model size)



Kraska, Tim, et al. "The case for learned index structures." SIGMOD 2018.

The Case for Learned Index Structures

Tim Kraska*
MIT
kraska@mit.edu

Alex Beutel
Google, Inc.
abeutel@google.com

Ed H. Chi
Google, Inc.
edchi@google.com

Jeffrey Dean
Google, Inc.
jeff@google.com

Neoklis Polyzotis
Google, Inc.
npoly@google.com

ABSTRACT

Indexes are models: a B-Tree-Index can be seen as a model to map a key to the position of a record within a sorted array, a Hash-Index as a model to map a key to a position of a record within an unsorted array, and a BitMap-Index as a model to indicate if a data record exists or not. In this exploratory research paper, we start from this premise and posit that all existing index structures can be replaced with other types of models, including deep-learning models, which we term *learned indexes*. We theoretically analyze under which conditions learned indexes outperform traditional index structures and describe the main challenges in designing learned index structures. Our

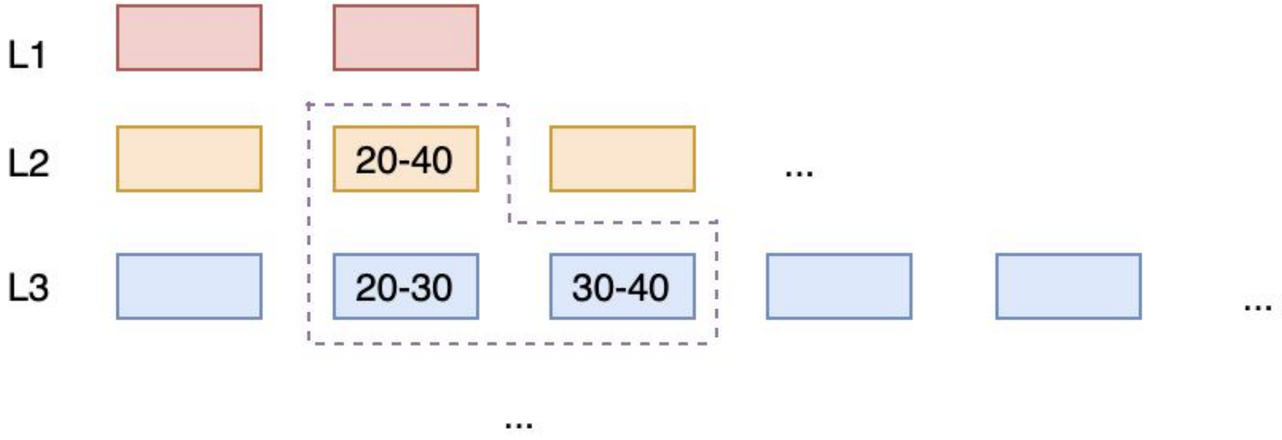
a set of continuous integer keys (e.g., the keys 1 to 100M), one would not use a conventional B-Tree index over the keys since the key itself can be used as an offset, making it an $O(1)$ rather than $O(\log n)$ operation to look-up any key or the beginning of a range of keys. Similarly, the index memory size would be reduced from $O(n)$ to $O(1)$. Maybe surprisingly, similar optimizations are possible for other data patterns. In other words, knowing the exact data distribution enables highly optimizing almost any index structure.

Of course, in most real-world use cases the data do not perfectly follow a known pattern and the engineering effort to build specialized solutions for every use case is usually too

What about updates?

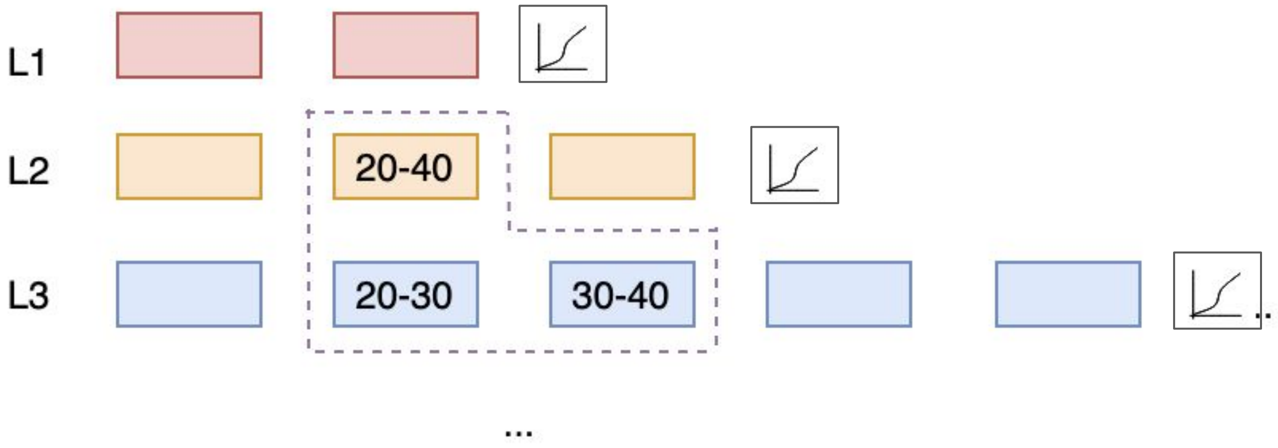
LSM Trees

- Dai, Yifan, et al. "**From {WiscKey} to Bourbon: A Learned Index for {Log-Structured} Merge Trees**." *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 2020.
- Abu-Libdeh, Hussam, et al. "**Learned indexes for a google-scale disk-based database**." *arXiv preprint arXiv:2012.12501* (2020).

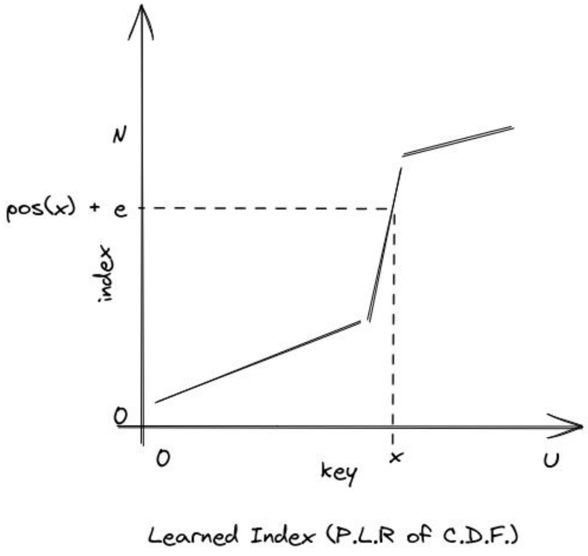


LSM Trees

- Dai, Yifan, et al. "**From {WiscKey} to Bourbon: A Learned Index for {Log-Structured} Merge Trees**." *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 2020.
- Abu-Libdeh, Hussam, et al. "**Learned indexes for a google-scale disk-based database**." *arXiv preprint arXiv:2012.12501* (2020).



Updateability of Learned Indexes

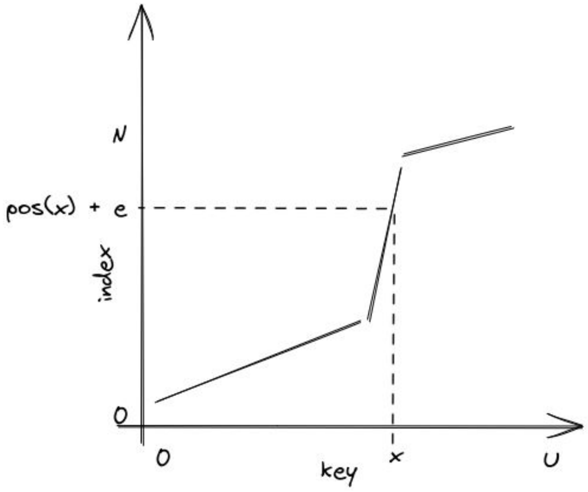


0, 2, 4, 6	8, 10, 12, 14	16, 18, 20, 22	24, 26, 28, 30	

Sorted, Clustered List on Disk

Updateability of Learned Indexes

Where do we insert?
The model has learnt distribution of sorted packed data!



Learned Index (P.L.R of C.D.F.)

7

Need to shift all items across

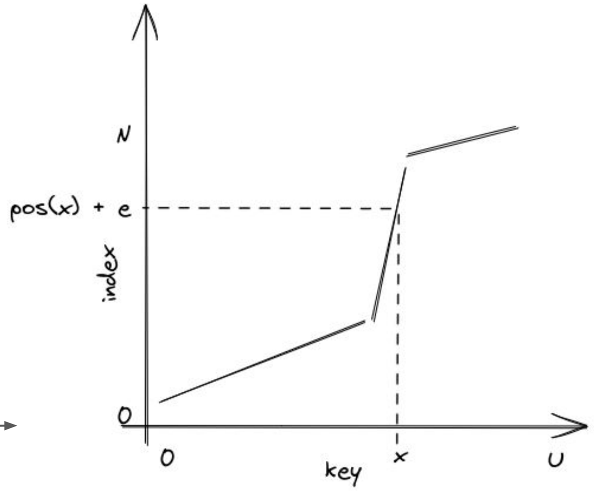
0, 2, 4, 6	8, 10, 12, 14	16, 18, 20, 22	24, 26, 28, 30	

Sorted, Clustered List on Disk

Updateability of Learned Indexes

Should we retrain?
We can tolerate error,
but how much?

7



Learned Index (P.L.R of C.D.F.)

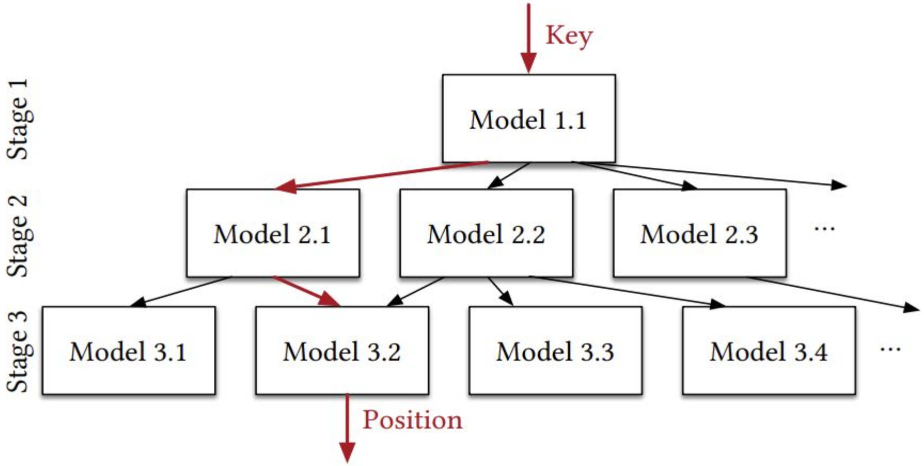
0, 2, 4, 6	8, 10, 12, 14	16, 18, 20, 22	24, 26, 28, 30	

Sorted, Clustered List on Disk

Take ideas from the B+ Tree

- Where do you insert?
  - Leave gaps in nodes
  - Split and merge

- When do you retrain?
  - Split and merge



ALEX: An Updatable Adaptive Learned Index

Jialin Ding[†] Umar Farooq Minhas[‡] Jia Yu[§]
Chi Wang[‡] Jaeyoung Do[‡] Yinan Li[‡] Hantian Zhang[∓] Badrish Chandramouli[‡]
Johannes Gehrke[‡] Donald Kossmann[‡] David Lomet[‡] Tim Kraska[†]
[†]Massachusetts Institute of Technology [‡]Microsoft Research [§]Arizona State University
[∓]Georgia Institute of Technology

ABSTRACT

Recent work on "learned indexes" has changed the way we look at the decades-old field of DBMS indexing. The key idea is that indexes can be thought of as "models" that predict the position of a key in a dataset. Indexes can, thus, be learned. The original work by Kraska et al. shows that a learned index beats a B+Tree by a factor of up to three in search time and by an order of magnitude in memory footprint. However, it is limited to static, read-only workloads.

index for dynamic workloads that effectively combines the core insights from the Learned Index with proven storage & indexing techniques to deliver great performance in both time and space? Our answer is a new in-memory index structure called ALEX, a fully dynamic data structure that simultaneously provides efficient support for point lookups, short range queries, inserts, updates, deletes, and bulk loading. This mix of operations is commonplace in online transaction processing (OLTP) workloads [6, 8, 32] and is also supported by B+Trees [29].

My Takeaway: Combine a B+ Tree structure with learned indexes, result: ALEX

ALEX: An Updatable Adaptive Learned Index

Jialin Ding† Umar Farooq Minhas‡ Jia Yu§
Chi Wang‡ Jaeyoung Do‡ Yinan Li‡ Hantian Zhang∓ Badrish Chandramouli‡
Johannes Gehrke‡ Donald Kossmann‡ David Lomet‡ Tim Kraska†
†Massachusetts Institute of Technology ‡Microsoft Research §Arizona State University
∓Georgia Institute of Technology

21 May 2020

ABSTRACT

Recent work on "learned indexes" has changed the way we look at the decades-old field of DBMS indexing. The key idea is that indexes can be thought of as "models" that predict the position of a key in a dataset. Indexes can, thus, be learned. The original work by Kraska et al. shows that a learned index beats a B+Tree by a factor of up to three in search time and by an order of magnitude in memory footprint. However, it is limited to static, read-only workloads.

index for dynamic workloads that effectively combines the core insights from the Learned Index with proven storage & indexing techniques to deliver great performance in both time and space? Our answer is a new in-memory index structure called ALEX, a fully dynamic data structure that simultaneously provides efficient support for point lookups, short range queries, inserts, updates, deletes, and bulk loading. This mix of operations is commonplace in online transaction processing (OLTP) workloads [6, 8, 32] and is also supported by B+Trees [29].

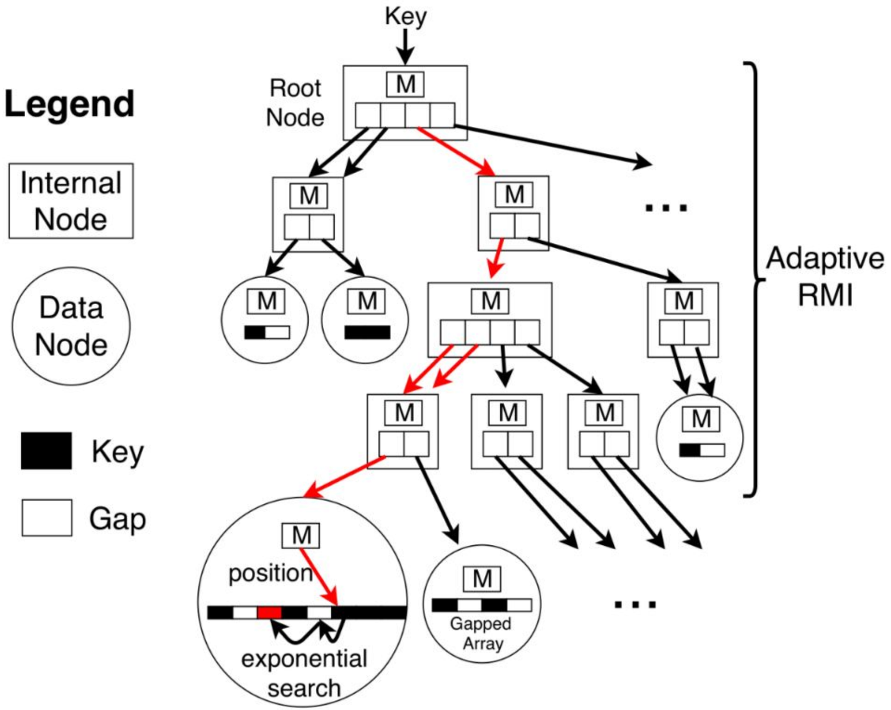
Implementing writes with high performance requires a

ALEX design overview

Structure
- **Dynamic tree** structure
- Each **node** contains a **linear model**
  - **internal** nodes → **models select the child** node
  - **data** nodes → **models predict the position** of a key

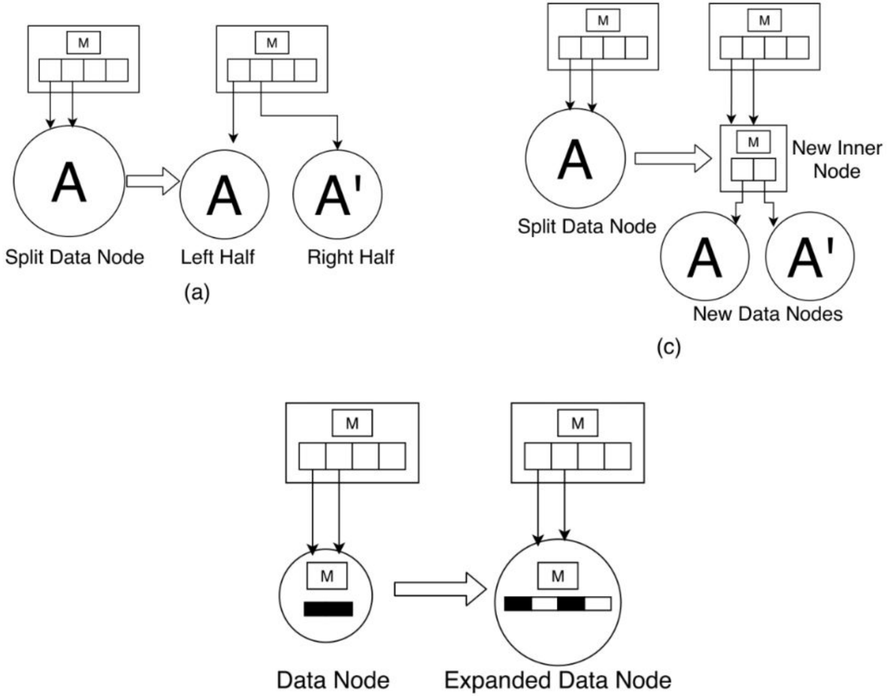
Core operations
- Lookup
  - Use **RMI to predict location of key** in a data node
  - Do **local search to correct** for prediction **error**
- Insert
  - Do a **lookup to find the insert** position
  - **Insert the new key/value** (might require shifting)

Current design constraints
a) In memory
b) Numeric data types
c) Single threaded



4. Adaptive Structure

- Flexible tree structure
  - Split nodes sideways
  - Split nodes downwards
  - Expand nodes
  - Merge nodes, contract nodes
- Key idea: all decisions are made to maximize performance
  - Use cost model of query runtime
  - No hand-tuning
  - Robust to data and workload shifts

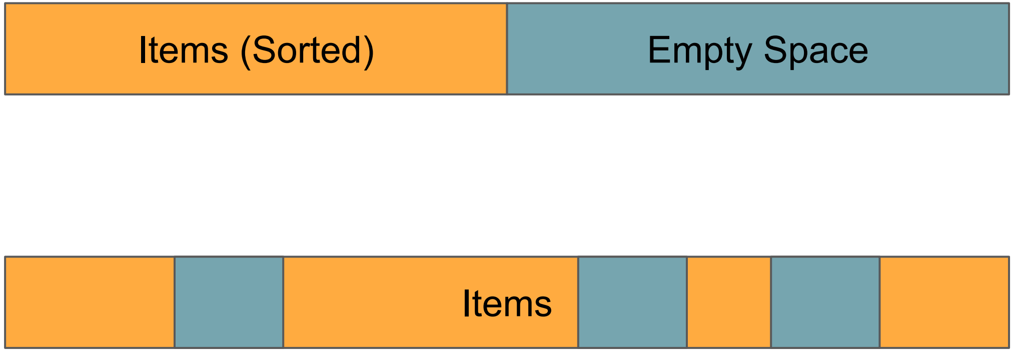


Slide from CS 6530, Fall 2022

ALEX - Optimizations

- Gapped Array
- Model Based Insertion
- Exponential Search

Gapped Array



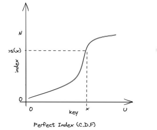
B+ Tree node
Dense, sorted

Gapped Array
Inserts more efficient
(less items to shift)

Model Based Insertion

- Where do you leave the gaps?
- Start with an empty array, insert according to model
- Model has errors, so gaps will naturally occur

X

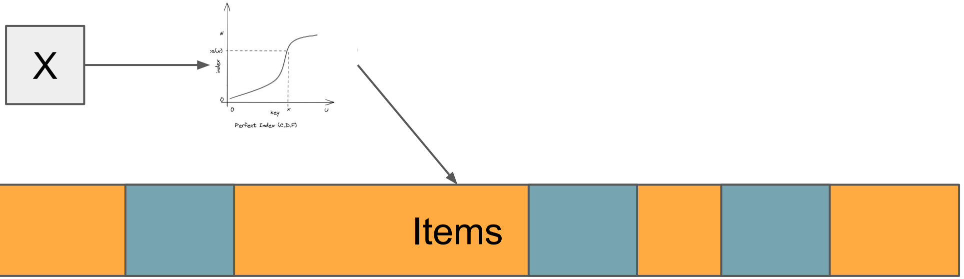


Gapped Array

Inserts more efficient
(less items to shift)

Model Based Insertion

- Where do you leave the gaps?
- Start with an empty array, insert according to model
- Model has errors, so gaps will naturally occur

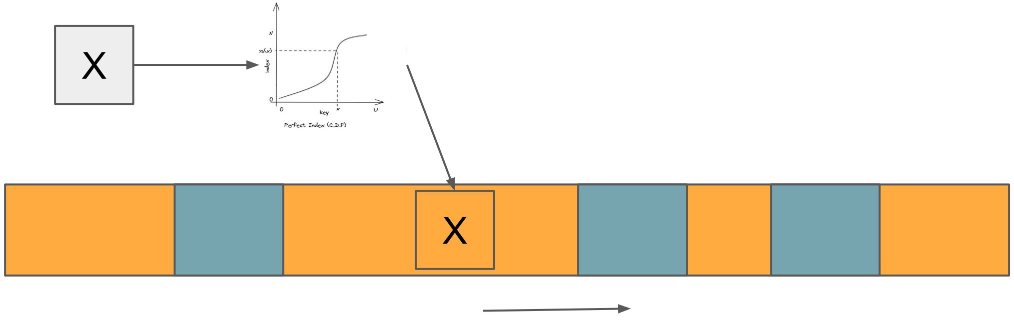


Gapped Array

Inserts more efficient
(less items to shift)

Model Based Insertion

- Where do you leave the gaps?
- Start with an empty array, insert according to model
- Model has errors, so gaps will naturally occur

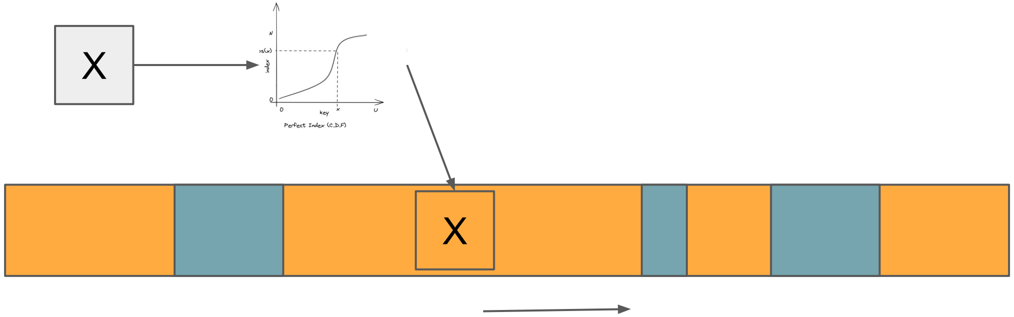


Shift other items to
nearest gap

Gapped Array
Inserts more efficient
(less items to shift)

Model Based Insertion

- Where do you leave the gaps?
- Start with an empty array, insert according to model
- Model has errors, so gaps will naturally occur

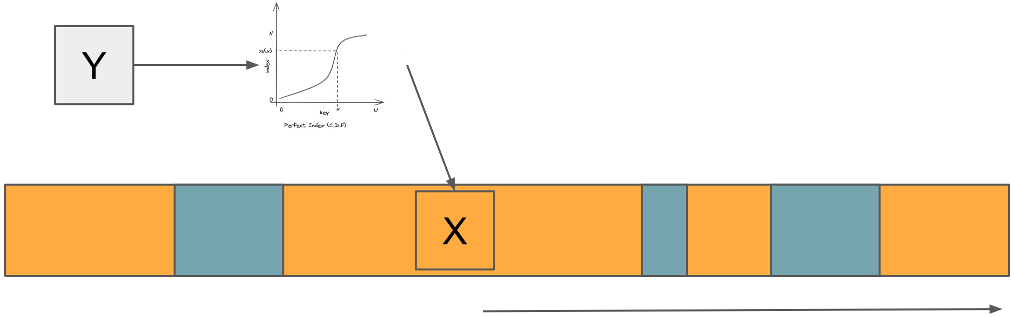


Shift other items to
nearest gap

Gapped Array
Inserts more efficient
(less items to shift)

Lookups in ALEX

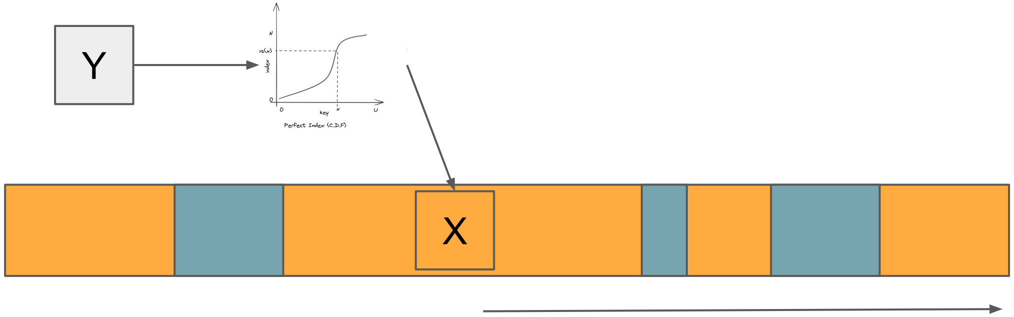
- Use the RMI to reach the correct Gapped Array
- Model based insertion - items will always be at or right of predicted position
- Start search from predicted position



Start search for Y from model
predicted position

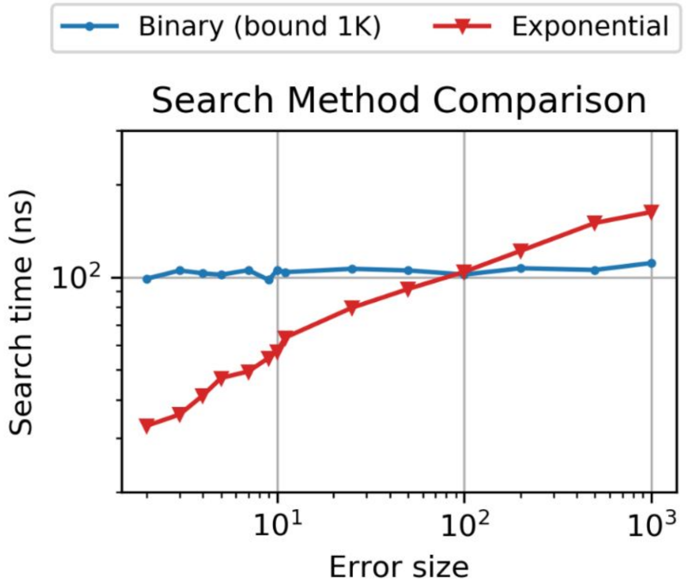
Search Algorithm

- Search - linear, binary or exponential?
- Exponential -
  - Search in windows of 2, 4, 8, 16… 2^x
  - If you overshoot, search in the second half with same window



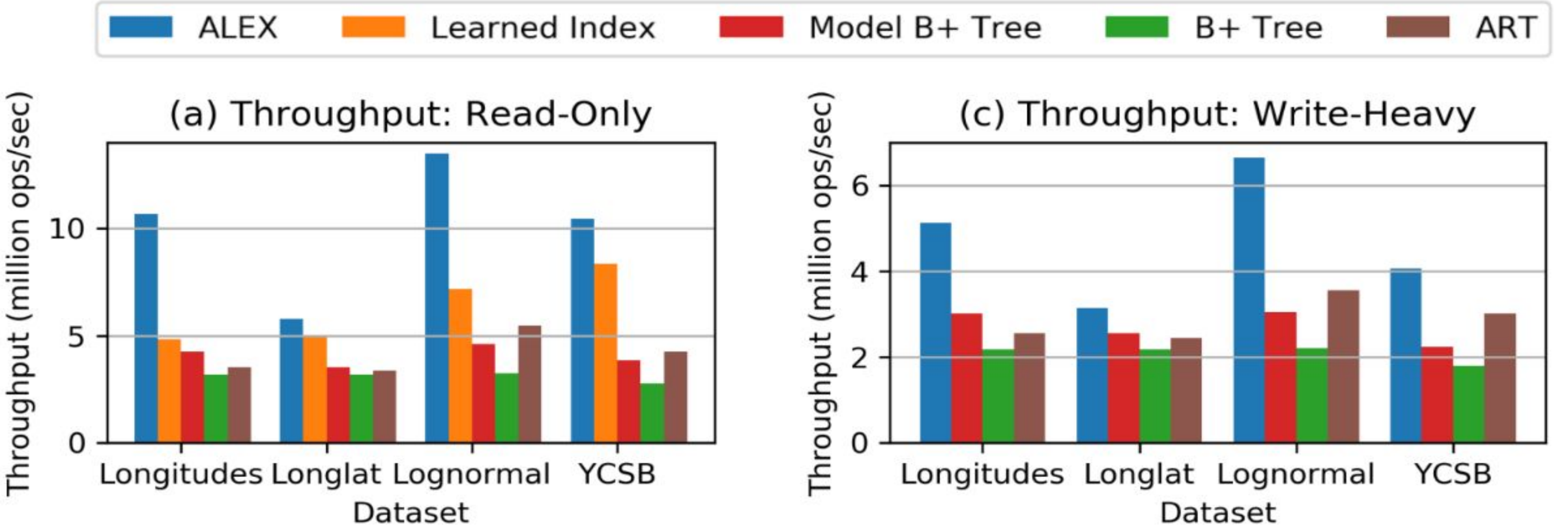
Start search for Y from model predicted position

3. Exponential Search



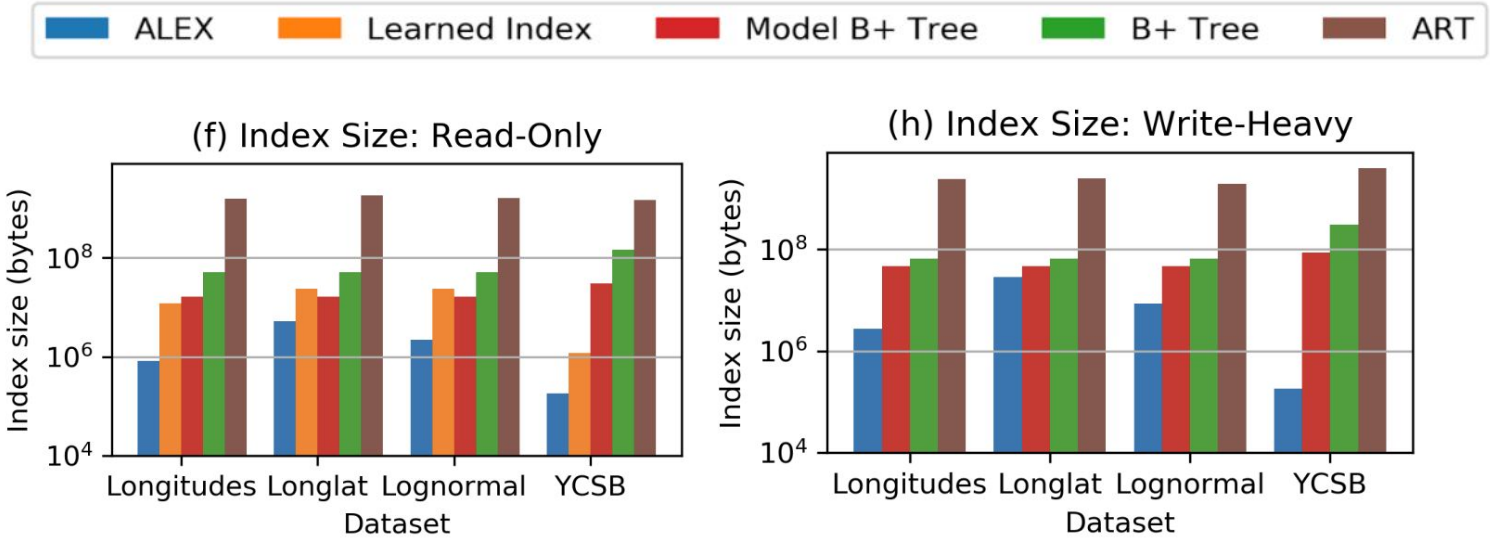
Model errors are low, so exponential search is faster than binary search

Slide from CS 6530, Fall 2022



~4x faster than B+ Tree
~2x faster than Learned Index

~2-3x faster than B+ Tree



~3 orders of magnitude less space for index

Search on Sorted Data Benchmark

Index / Index Size	XS Up to 0.01% of data size	S Up to 0.1% of data size	M Up to 1% of data size ▼	L Up to 10% of data size	XL No limit
RMI	374 ns	225 ns	**172 ns**	**151 ns**	**141 ns**
RS	**169 ns**	**156 ns**	203 ns	193 ns	184 ns
PGM	354 ns	303 ns	247 ns	228 ns	228 ns
ALEX		534 ns	430 ns	355 ns	298 ns
ART		562 ns	441 ns	396 ns	379 ns
BTree	806 ns	601 ns	524 ns	515 ns	514 ns
FAST		633 ns	544 ns	544 ns	544 ns
BinarySearch	680 ns	680 ns	680 ns	680 ns	680 ns

Search on Sorted Data Benchmark

	Index / Index Size	XS Up to 0.01% of data size	S Up to 0.1% of data size	M Up to 1% of data size	L Up to 10% of data size	XL No limit
Plot: Remove	**RMI**	14154004 ns	14254178 ns	9559940 ns	12571378 ns	20855801 ns
Plot: Remove	**RS**	1250036 ns	1336533 ns	1472435 ns	1566636 ns	2463623 ns
Plot: Add	**PGM**	3584017 ns	3584017 ns	5451576 ns	6390041 ns	6390041 ns
Plot: Add	ART		179503 ns	244217 ns	697001 ns	697001 ns
Plot: Remove	BTree	23 ns	174 ns	2138 ns	57139 ns	57139 ns
Plot: Add	FAST		1984 ns	10003 ns	10003 ns	10003 ns
Plot: Remove	**ALEX**		4580 ns	19918 ns	940022 ns	5627242 ns
Plot: Add	BinarySearch	0 ns	0 ns	0 ns	0 ns	0 ns

Open Research Problems

- **Theoretical lower bounds**
- **Large build times**
- Variable length keys
- String keys
- Compression
- Concurrency
- Specialized Hardware

Different Learned Indexes

- RMI
- PGM Index - Ferragina, Paolo, and Giorgio Vinciguerra. "The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds." *Proceedings of the VLDB Endowment* 13.8 (2020): 1162-1175.
- ALEX - Ding, Jialin, et al. "ALEX: an updatable adaptive learned index." *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020
- FITing Tree - Galakatos, Alex, et al. "Fiting-tree: A data-aware index structure." *Proceedings of the 2019 international conference on management of data*. 2019.
- LIPP - Jiacheng Wu, Yong Zhang, Shimin Chen, Yu Chen, Jin Wang, Chunxiao Xing: Updatable Learned Index with Precise Positions. Proc. VLDB Endow. 14(8): 1276-1288 (2021).

Learned Index Performance

- Benchmark
  - SOSD - Marcus, Ryan, et al. "**Benchmarking learned indexes**.", NeurIPS Workshop on Machine Learning for Systems
  - Lan, Hai, et al. "**Updatable Learned Indexes Meet Disk-Resident DBMS-From Evaluations to Design Choices**." *Proceedings of the ACM on Management of Data* 1.2 (2023): 1-22.
- Theoretical
  - Ferragina, Paolo, Fabrizio Lillo, and Giorgio Vinciguerra. "**Why are learned indexes so effective?**." *International Conference on Machine Learning*. PMLR, 2020.
  - Sabek, Ibrahim, et al. "**Can Learned Models Replace Hash Functions?.**" *Proceedings of the VLDB Endowment* 16.3 (2022): 532-545.

Other

- Genomics
    - Ho, Darryl, et al. "**Lisa: Learned indexes for DNA sequence analysis**." *bioRxiv* (2020).
    - Kirsche, Melanie, Arun Das, and Michael C. Schatz. "**Sapling: Accelerating suffix array queries with learned data models**." *Bioinformatics* 37.6 (2021): 744-749.
- Spatial Indexing
    - Varun Pandey, Alexander van Renen, Andreas Kipf, Jialin Ding, Ibrahim Sabek, Alfons Kemper: **The Case for Learned Spatial Indexes.** AIDB@VLDB 2020
- Classical Algorithms
    - Kristo, Ani, et al. "**The case for a learned sorting algorithm**." *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020.
    - Sabek, Ibrahim, and Tim Kraska. "**The Case for Learned In-Memory Joins**." *arXiv preprint arXiv:2111.08824* (2021). (VLDB 2023)

Thanks!

Questions?