The Case for Learned Index Structures

Authors: Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis

Presenter: Ruijia Mao

Agenda

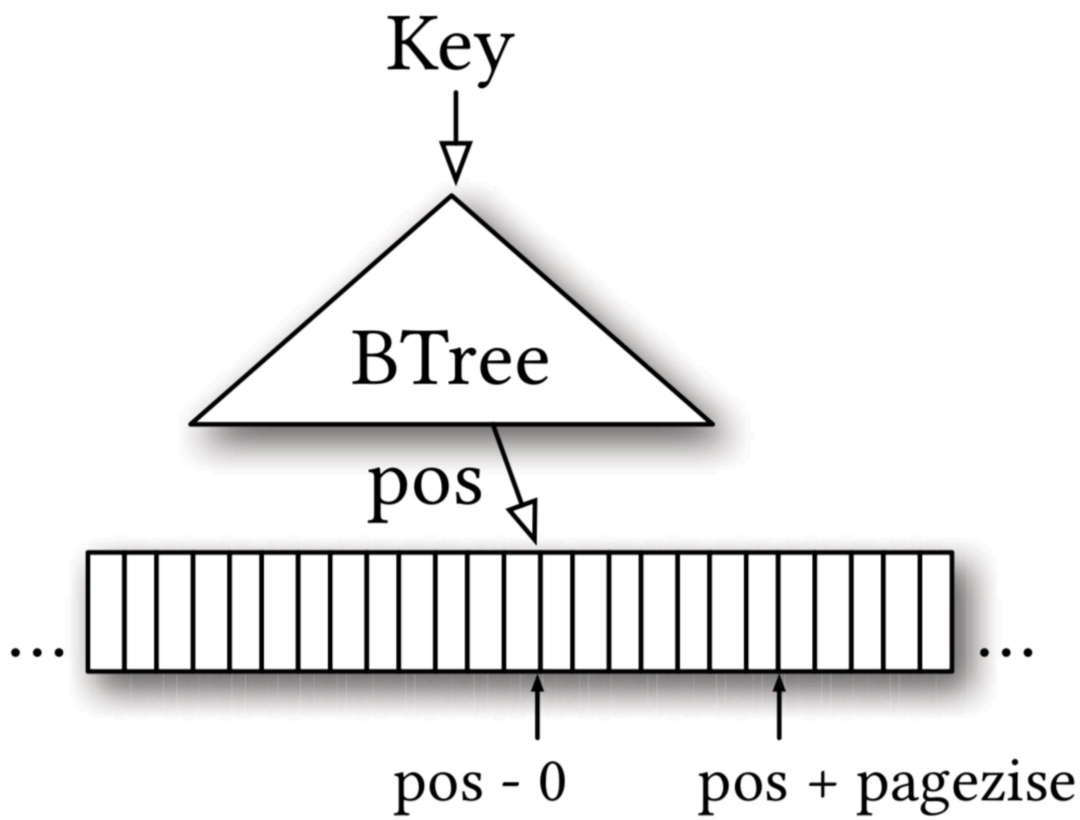
- **Introduction**
- **Range Indexes** - B-Tree Index
- **Point Index** - Hash-Map Index
- **Existence Index** - Bloom Filter Index
- **Conclusion & Future Work**

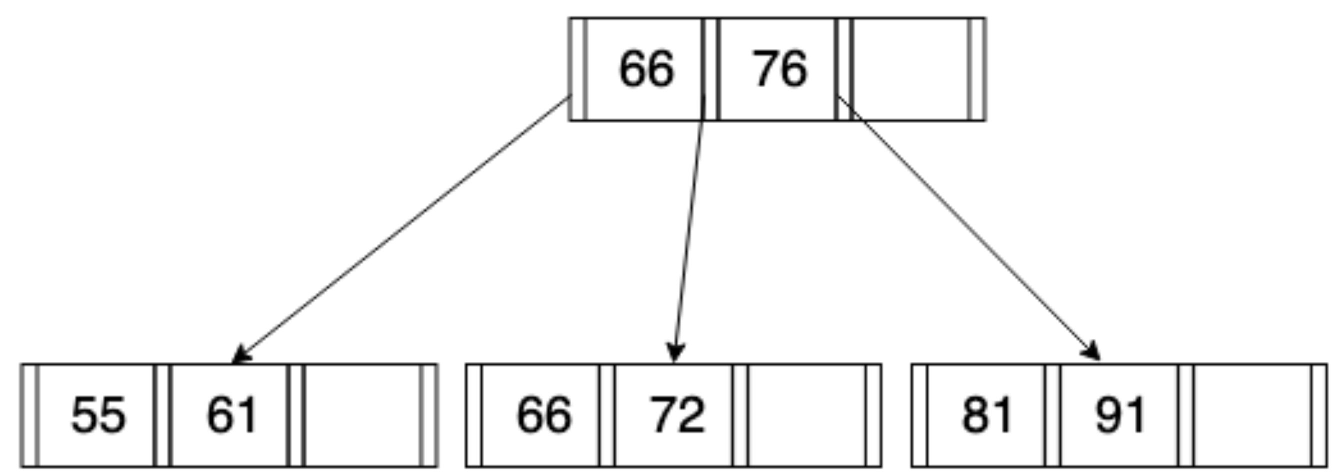
Introduction

Introduction:
Index Examples

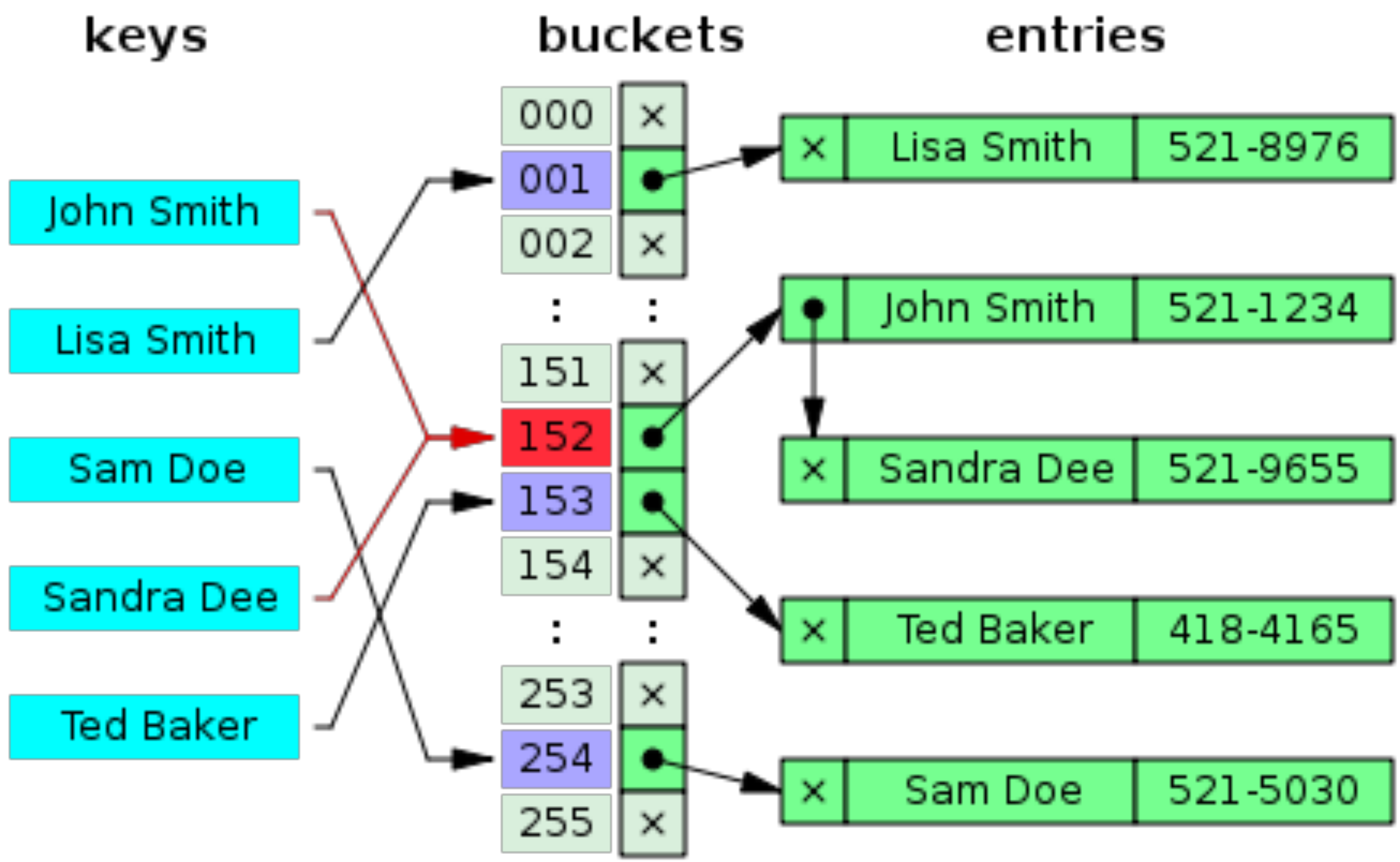
- B-Tree Index
- Hash-Map Index
- Bloom Filter Index

Introduction: B-Tree Index

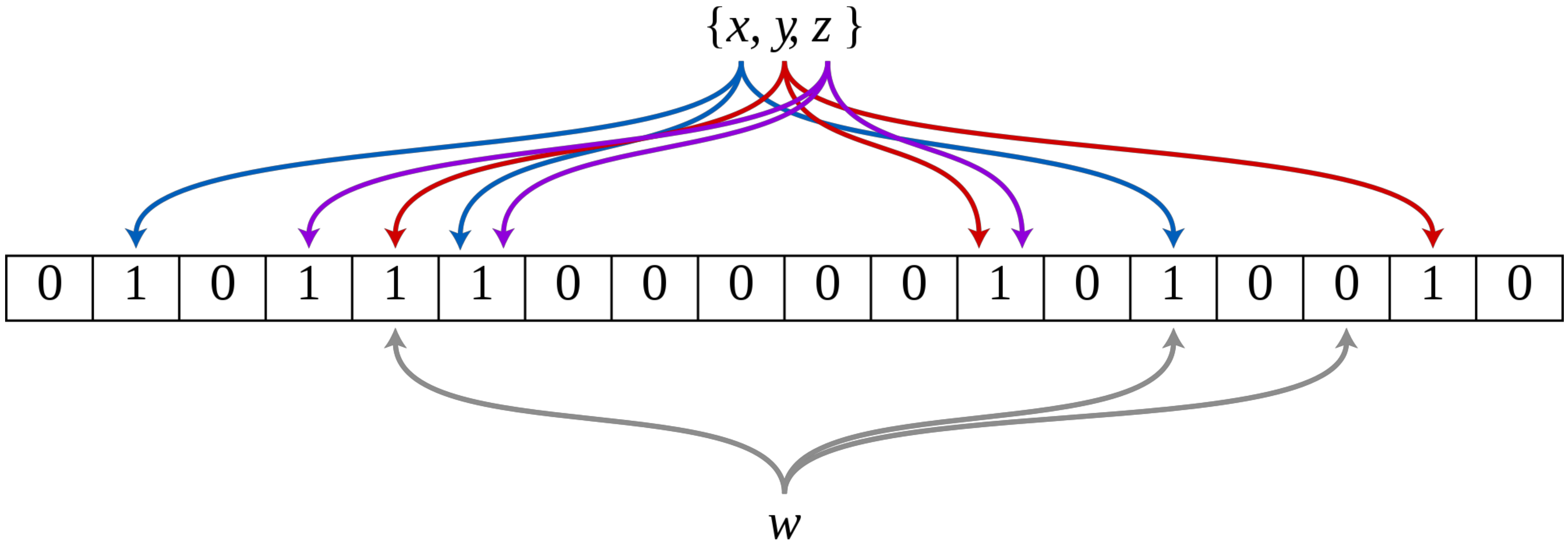




Introduction: Hash-Map Index



Introduction: Bloom Filter



Introduction:
Indexes are models

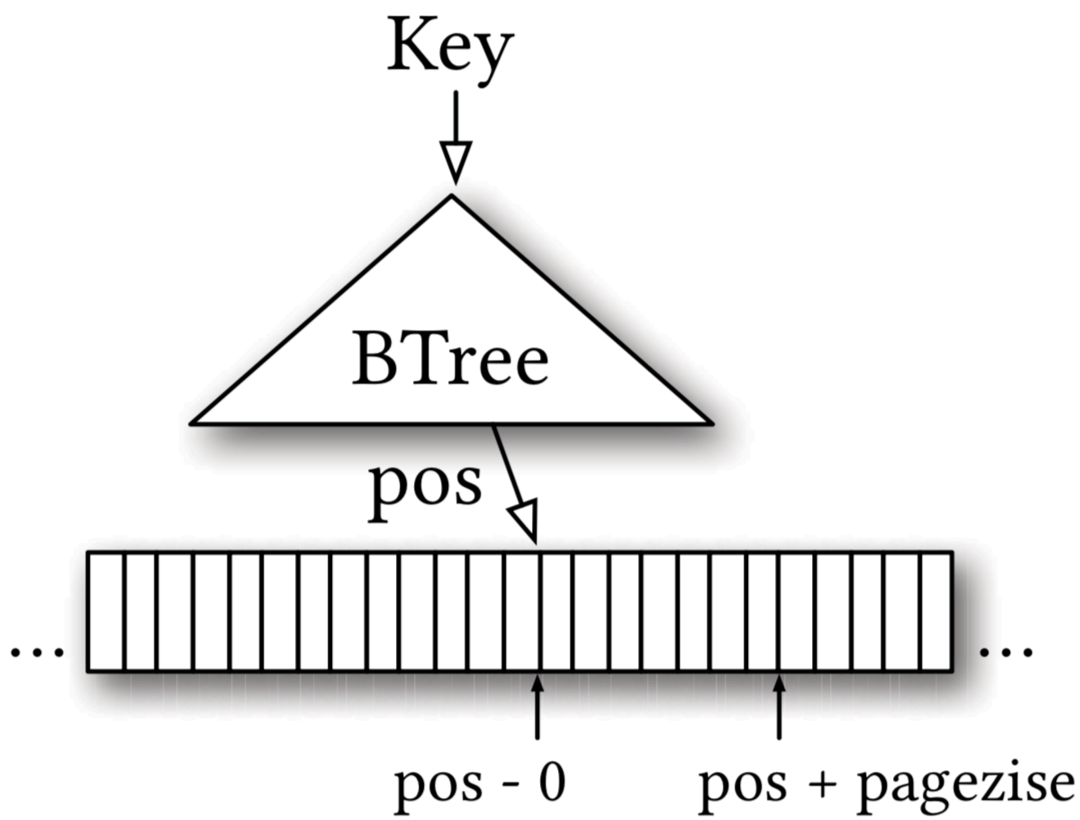
- General purpose index structures assume nothing about data distribution
- **Learned indexes** - learn a model that reflects patterns in the data - automatic synthesis of specialized index structures

Introduction:
Indexes are models

- Indexes are to a large extent learned models
    - B-Tree Index - take a key as an input and predicts the position of a data record in a sorted set
    - Bloom Filter - binary classifier

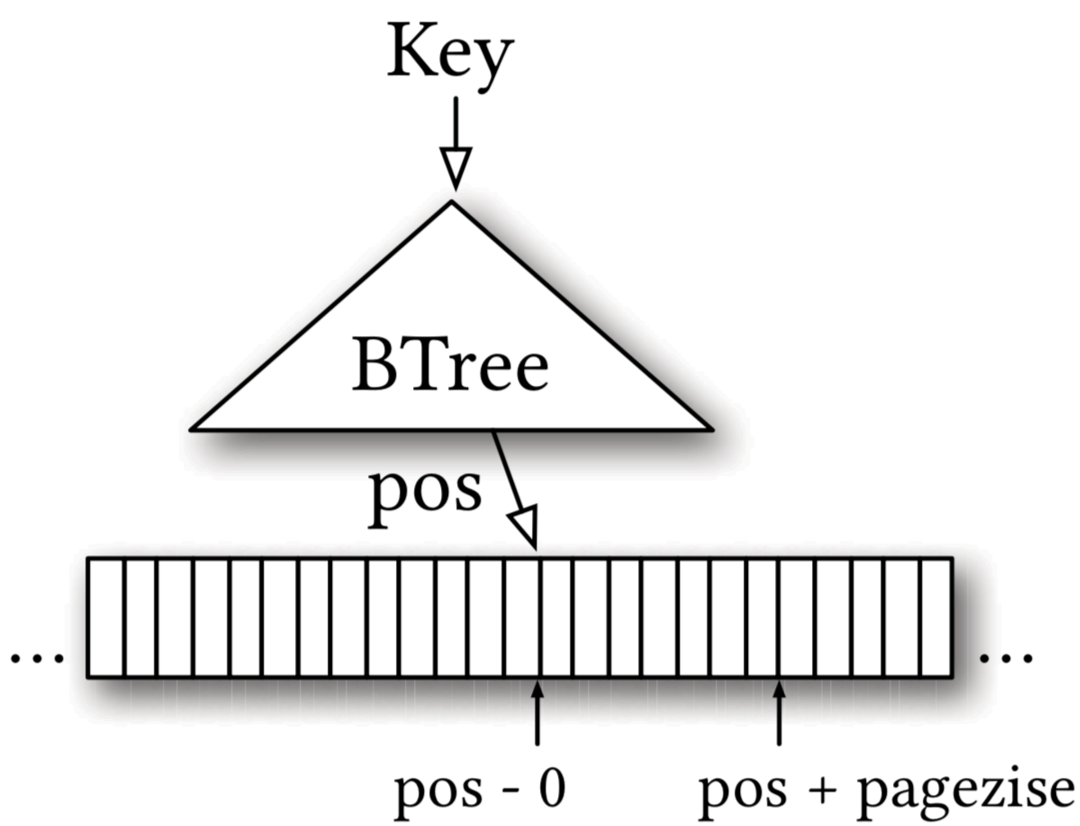
Range Indexes

Range Indexes



- Maps a key to a position
- For efficiency, indexing only the first key of every page

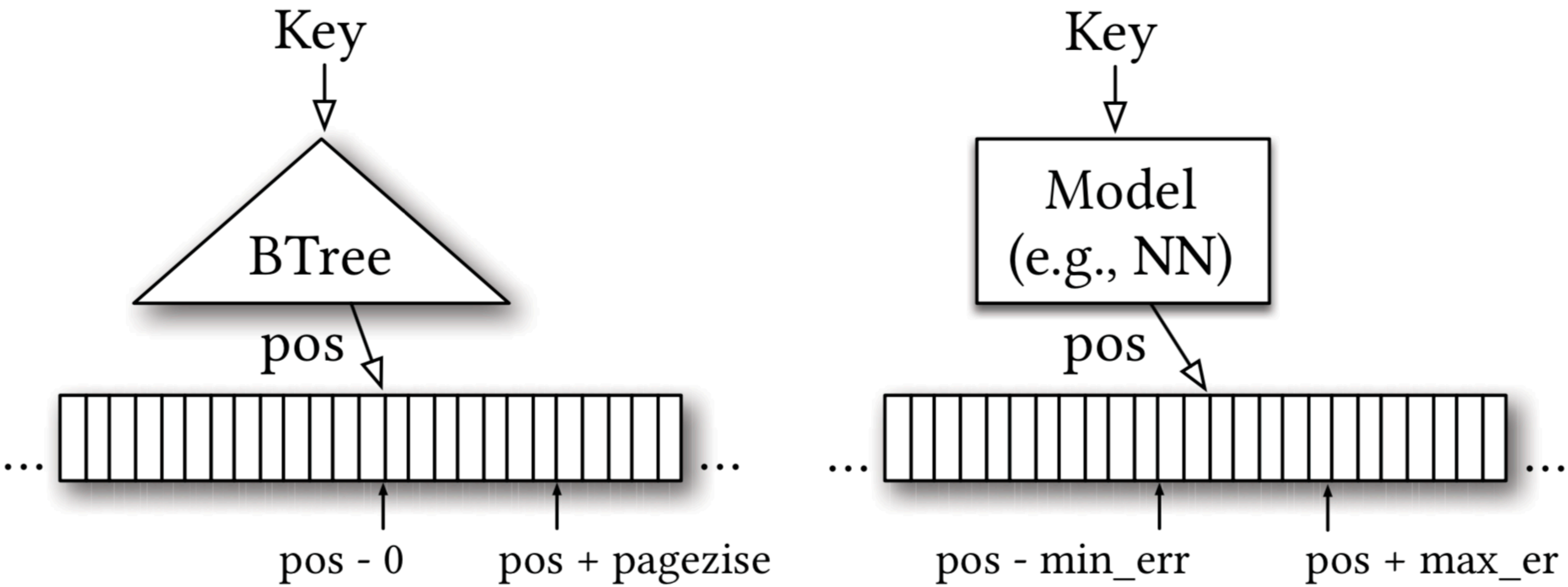
Range Indexes



- The B-Tree is a model, or in ML terminology, a regression tree

- it maps a key to a position with a min- and max-error, with a guarantee that the key can be found in that region if it exists.

Range Indexes



Range Index Models are CDF Models

- A model that predicts the position given a key inside a sorted array effectively approximates the cumulative distribution function (CDF).
- $p = F(Key) * N$
- p is the position estimate
- F(Key) is $P(X \le Key)$

A Frist, Naive Learned Index

- Data: 200M web-server log records
- Goal: building a secondary index over the times- tamps using Tensorflow
- Model: trained a two-layer fully-connected neural network with 32 neurons per layer using ReLU activation functions

A Frist, Naive Learned Index: Results

- Model: ≈ 1250 predictions per second, ≈ 80, 000 nano-seconds (ns) to execute the model with Tensorflow, without the search time
- B-Tree: traversal over the same data ≈ 300ns
- Binary search the entire data: ≈ 900ns

A Frist, Naive Learned Index: Problems

- Tensorflow is designed for larger model
- Last mile: B-Trees are good in overfitting the data with a few operations, while the models are good at approximate the general shape of a CDF
- B-Trees are extremely cache- and operation-efficient

A Frist, Naive Learned Index

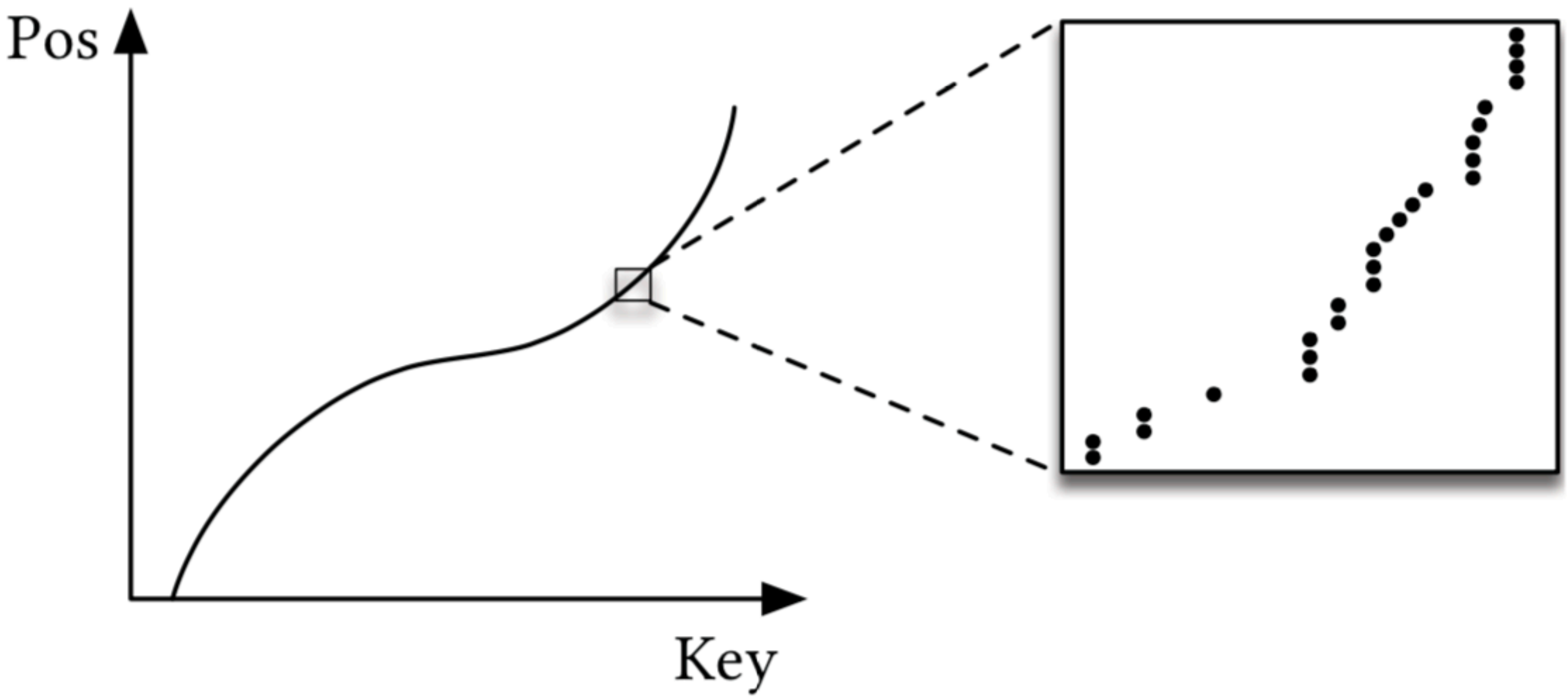


Figure 2: Indexes as CDFs

The RM-Index

- In order to solve challenges mentioned above, the authors developed
  - Learning Index Framework (LIF)
  - Recursive Model Indexes (RMI)
  - Standard-error-based search strategies

The RM-Index: LIF

- Learning Index Framework (LIF)
  - An index synthesis system: given an index specification, LIF generates different index configurations, optimizes them, and tests them automatically.

The RM-Index: RMI

- Recursive Model Index (RMI)
  - A hierarchy of models. At each stage the model takes the key as an input, and based on it picks another model, until the final stage predicts the position.

The RM-Index: RMI

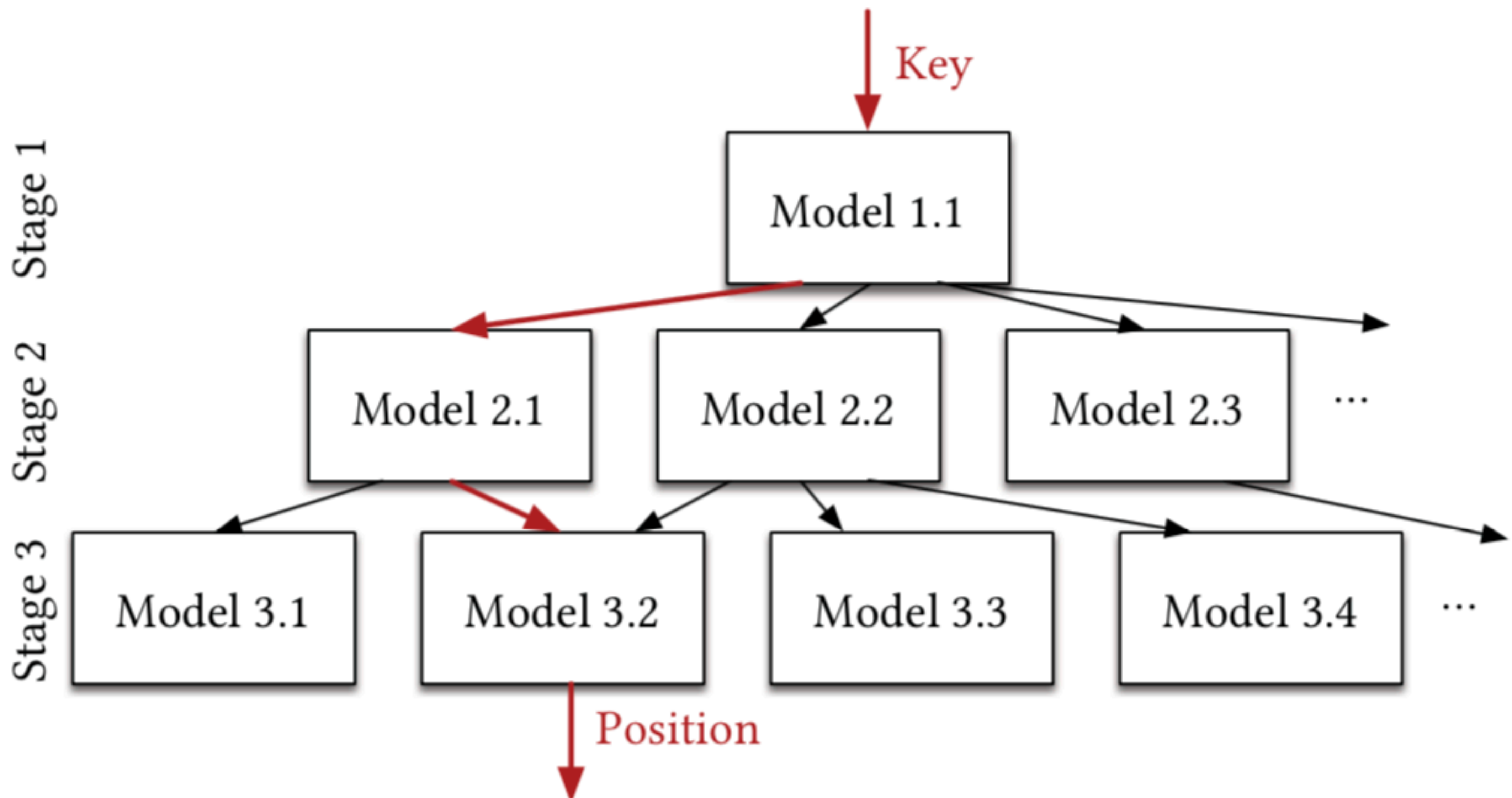


Figure 3: Staged models

The RM-Index: RMI Benefits

- It separates model size and complexity from execution cost.
- It leverages the fact that it is easy to learn the overall shape of the data distribution.
- It effectively divides the space into smaller sub-ranges, like a B-Tree, to make it easier to achieve the required “last mile” accuracy with fewer operations.
- There is no search process required in-between the stages.

The RM-Index: Hybrid Indexes

- Another advantage of the recursive model index is that mixtures of models can be built.

  - Top layer - a small ReLU neural net

  - Bottom - linear regression

The RM-Index: Search Strategy

- **Model Biased Search** - the first middle point is set to the value predicted by the model
- **Biased Quaternary Search** - three middle points of quaternary search as $pos - \sigma$, pos, $pos + \sigma$

Results

		Map Data			Web Data			Log-Normal Data		
Type	**Config**	Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)
Btree	page size: 32	52.45 (4.00x)	274 (0.97x)	198 (72.3%)	51.93 (4.00x)	276 (0.94x)	201 (72.7%)	49.83 (4.00x)	274 (0.96x)	198 (72.1%)
	page size: 64	26.23 (2.00x)	277 (0.96x)	172 (62.0%)	25.97 (2.00x)	274 (0.95x)	171 (62.4%)	24.92 (2.00x)	274 (0.96x)	169 (61.7%)
	page size: 128	13.11 (1.00x)	265 (1.00x)	134 (50.8%)	12.98 (1.00x)	260 (1.00x)	132 (50.8%)	12.46 (1.00x)	263 (1.00x)	131 (50.0%)
	page size: 256	6.56 (0.50x)	267 (0.99x)	114 (42.7%)	6.49 (0.50x)	266 (0.98x)	114 (42.9%)	6.23 (0.50x)	271 (0.97x)	117 (43.2%)
	page size: 512	3.28 (0.25x)	286 (0.93x)	101 (35.3%)	3.25 (0.25x)	291 (0.89x)	100 (34.3%)	3.11 (0.25x)	293 (0.90x)	101 (34.5%)
Learned Index	2nd stage models: 10k	0.15 (0.01x)	98 (2.70x)	31 (31.6%)	0.15 (0.01x)	222 (1.17x)	29 (13.1%)	0.15 (0.01x)	178 (1.47x)	26 (14.6%)
	2nd stage models: 50k	0.76 (0.06x)	85 (3.11x)	39 (45.9%)	0.76 (0.06x)	162 (1.60x)	36 (22.2%)	0.76 (0.06x)	162 (1.62x)	35 (21.6%)
	2nd stage models: 100k	1.53 (0.12x)	82 (3.21x)	41 (50.2%)	1.53 (0.12x)	144 (1.81x)	39 (26.9%)	1.53 (0.12x)	152 (1.73x)	36 (23.7%)
	2nd stage models: 200k	3.05 (0.23x)	86 (3.08x)	50 (58.1%)	3.05 (0.24x)	126 (2.07x)	41 (32.5%)	3.05 (0.24x)	146 (1.79x)	40 (27.6%)

Figure 4: Learned Index vs B-Tree

Point Index

Point Index:
Hash-map Index

- Conflict: too many distinct keys being mapped to the same position inside the Hash-map

Point Index: Hash-map Index

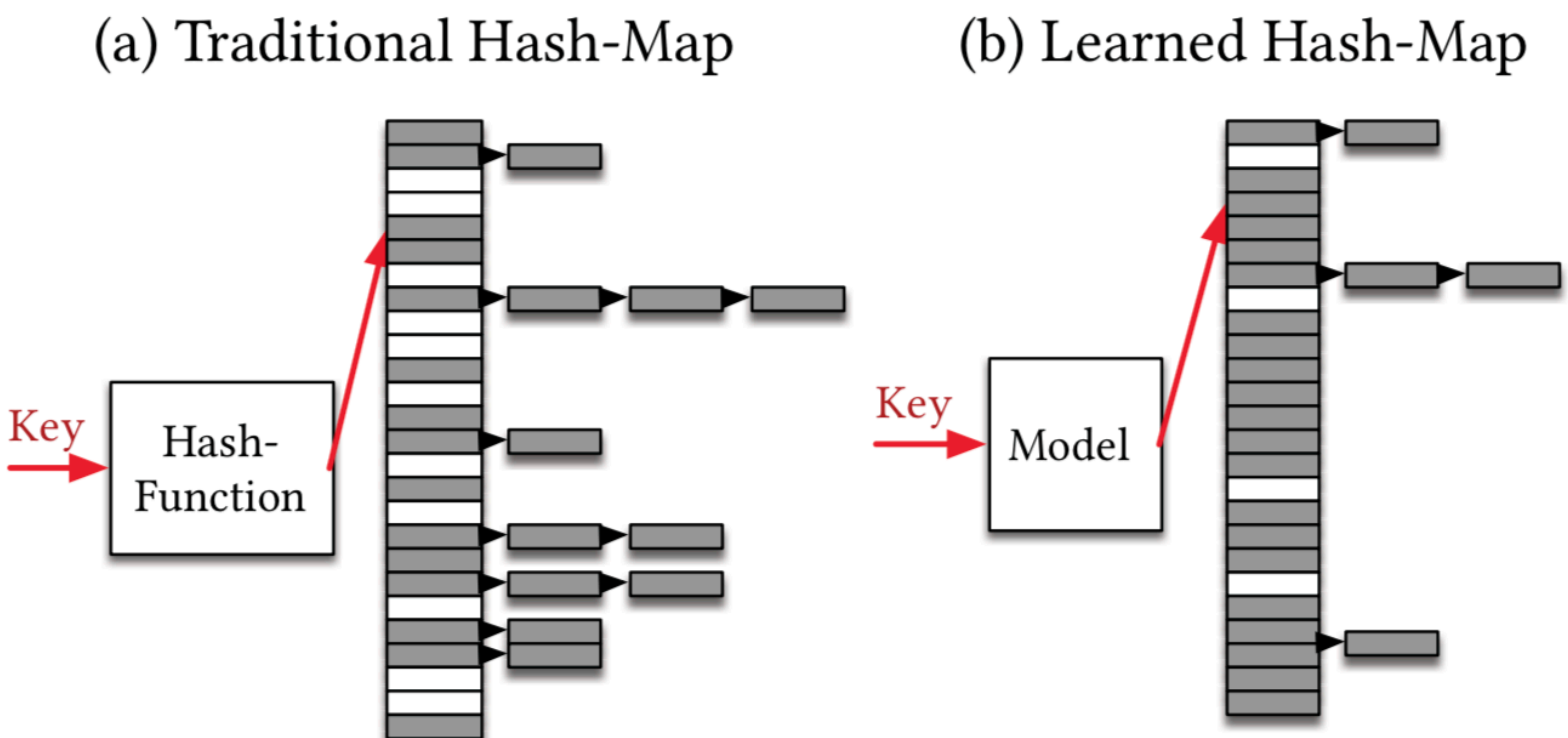


Figure 7: Traditional Hash-map vs Learned Hash-map

Point Index: Hash-map Index

- Learning the CDF of the key distribution is one potential way to learn a better hash function.
- Use $h(K) = F(K) * M$, with key K as our hash-function.
- If the model F perfectly learned the empirical CDF of the keys, no conflicts would exist

Point Index: Results

	% Conflicts Hash Map	% Conflicts Model	Reduction
Map Data	35.3%	07.9%	77.5%
Web Data	35.3%	24.7%	30.0%
Log Normal	35.4%	25.9%	26.7%

Figure 8: Reduction of Conflicts

Existence Index

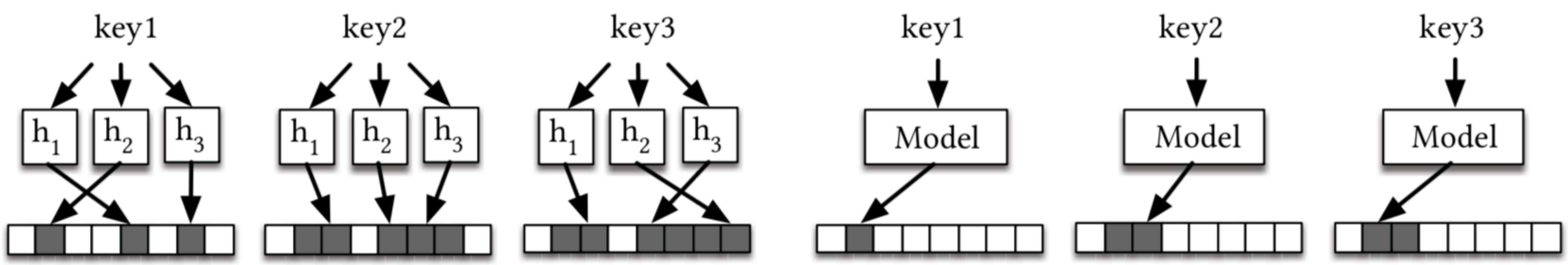
Existence Index: Learned Bloom Filters

- Separate keys from everything else
- Provide a specific FPR for realistic queries in particular while maintaining a FNR of zero
- Non-keys come from observable histori-cal queries
- Use recurrent neural network (RNN)

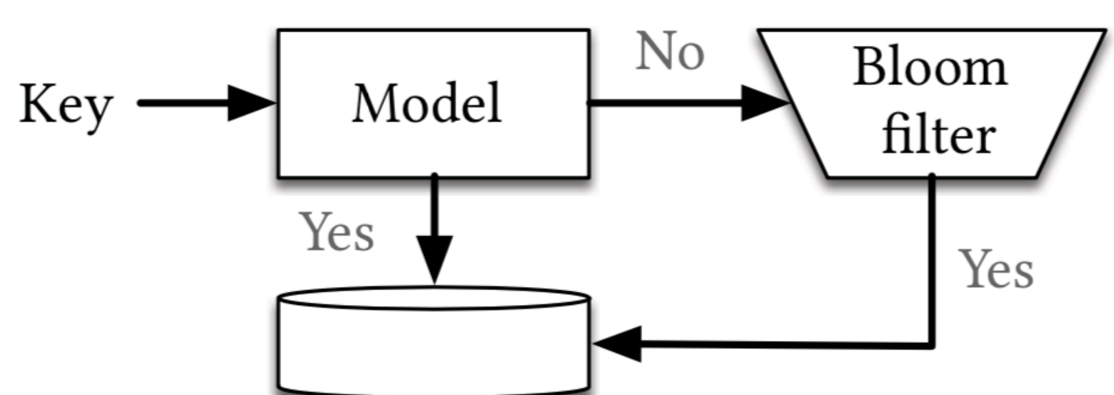
Existence Index:
Learned Bloom Filters as a Classification Problem

(a) Traditional Bloom-Filter Insertion

(b) Learned Bloom-Filter Insertion



(c) Bloom filters as a classification problem



Existence Index: Results

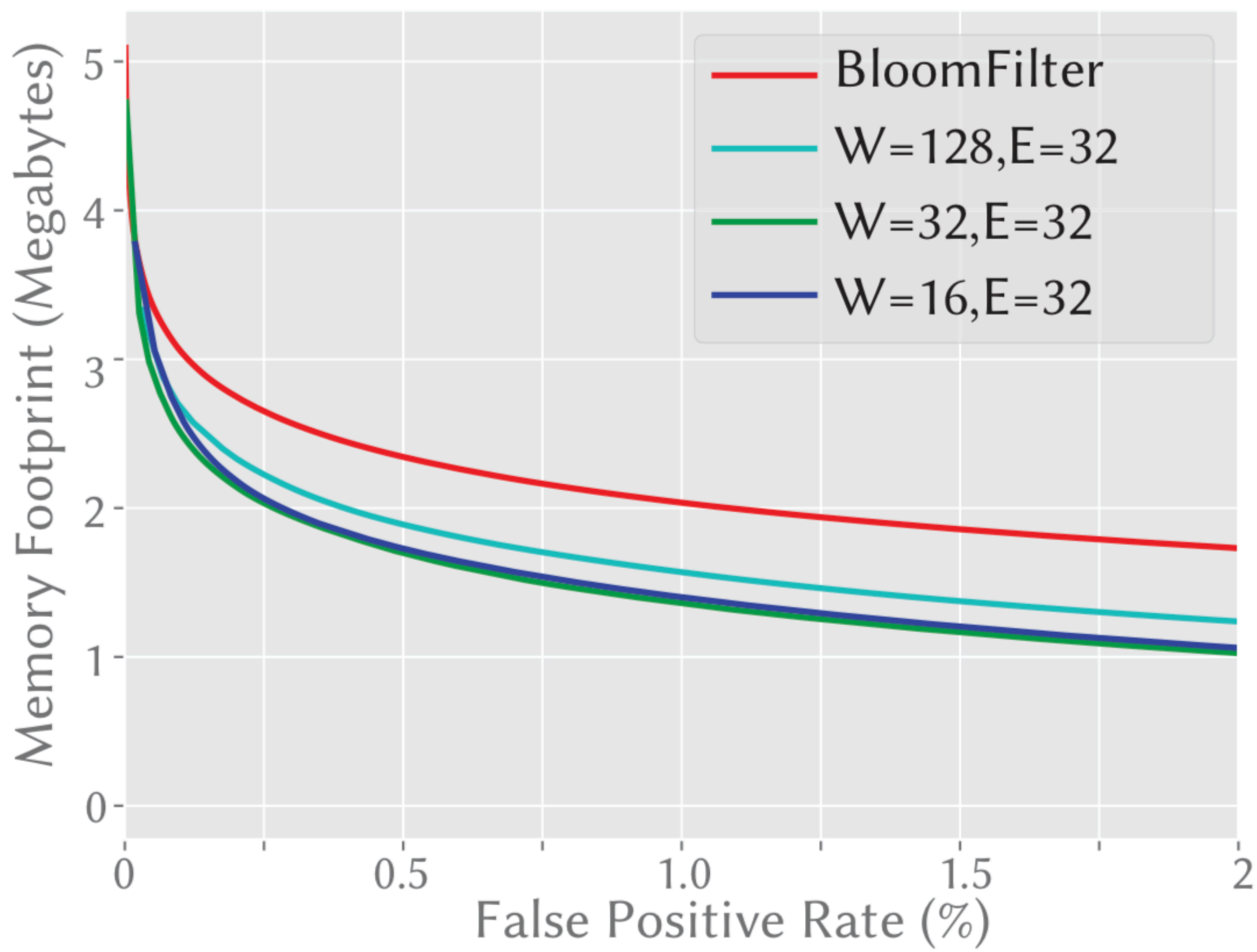


Figure 10: Learned Bloom filter improves memory footprint at a wide range of FPRs. (Here *W* is the RNN width and *E* is the embedding size for each character.)

Conclusion

Conclusion

- “In summary, we have demonstrated that machine learned models have the potential to provide significant benefits over state-of-the-art indexes, and we believe this is a fruitful direction for future research. ”

Future Work

- Other ML Models
- Multi-dimensional Indexes
- Learned Algorithm - sorting or join
- GPU/TPU

Thanks

Q&A