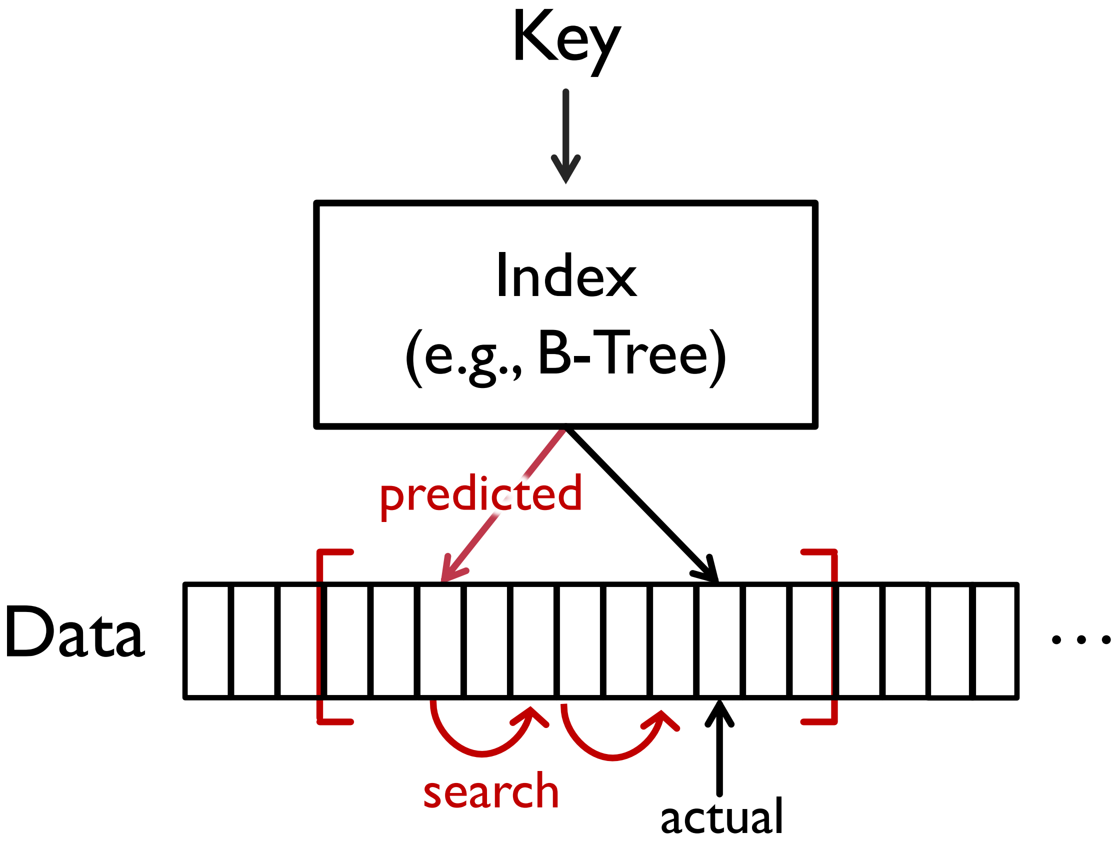
SIndex: A Scalable Learned Index for String Keys

Youyun Wang, Chuzhe Tang, Zhaoguo Wang, Haibo Chen

SHANGHAI JIAO TONG UNIVERSITY 1896

IPADS
INSTITUTE OF PARALLEL
AND DISTRIBUTED SYSTEMS

The learned index[1]



[1] Kraska et al., The Case for Learned Index Structures (SIGMOD '18)

The learned index: advantages

😀 **Fast query**

Up to 3× than B-Tree

😀 **Small memory footprint**

Save 99% memory usage than B-Tree

The learned index: disadvantages

😀 **Inefficient support for writes**

XIndex [PPoPP '20], ALEX [SIGMOD '20],

😀 **Dependence on workloads**

XIndex [PPoPP '20], PGM-Index [VLDB '20]

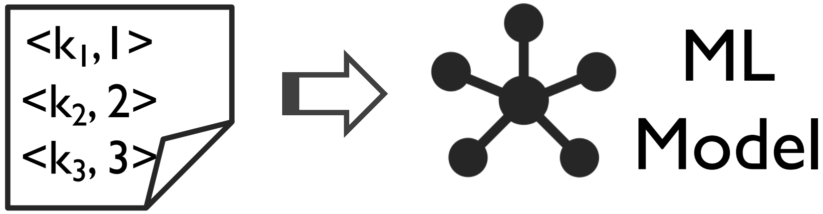
🤔 **Inefficient support for string key**

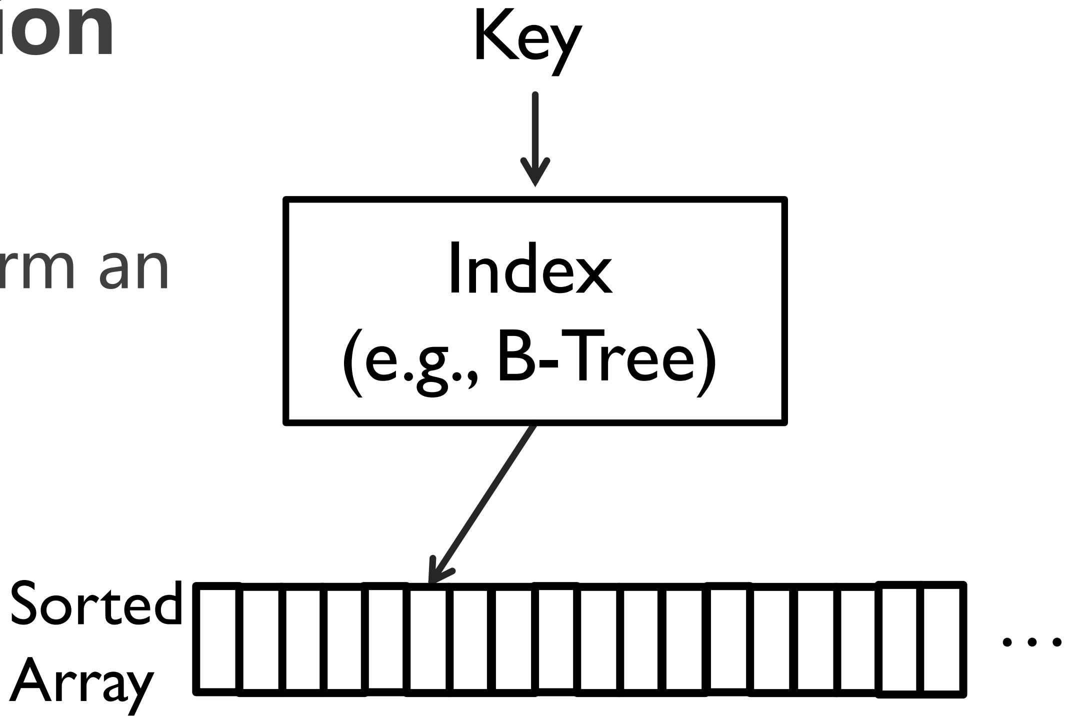
This talk: SIndex

- The first learned index that supports string keys efficiently
- Key idea: use **partial key** to reduce the costs of both model inference and data access
- Up to 91% better perf than the state-of-the-arts

Background: the learned index

1. **Train model with key-position mappings**
   - Memorize the min/max error to form an error range



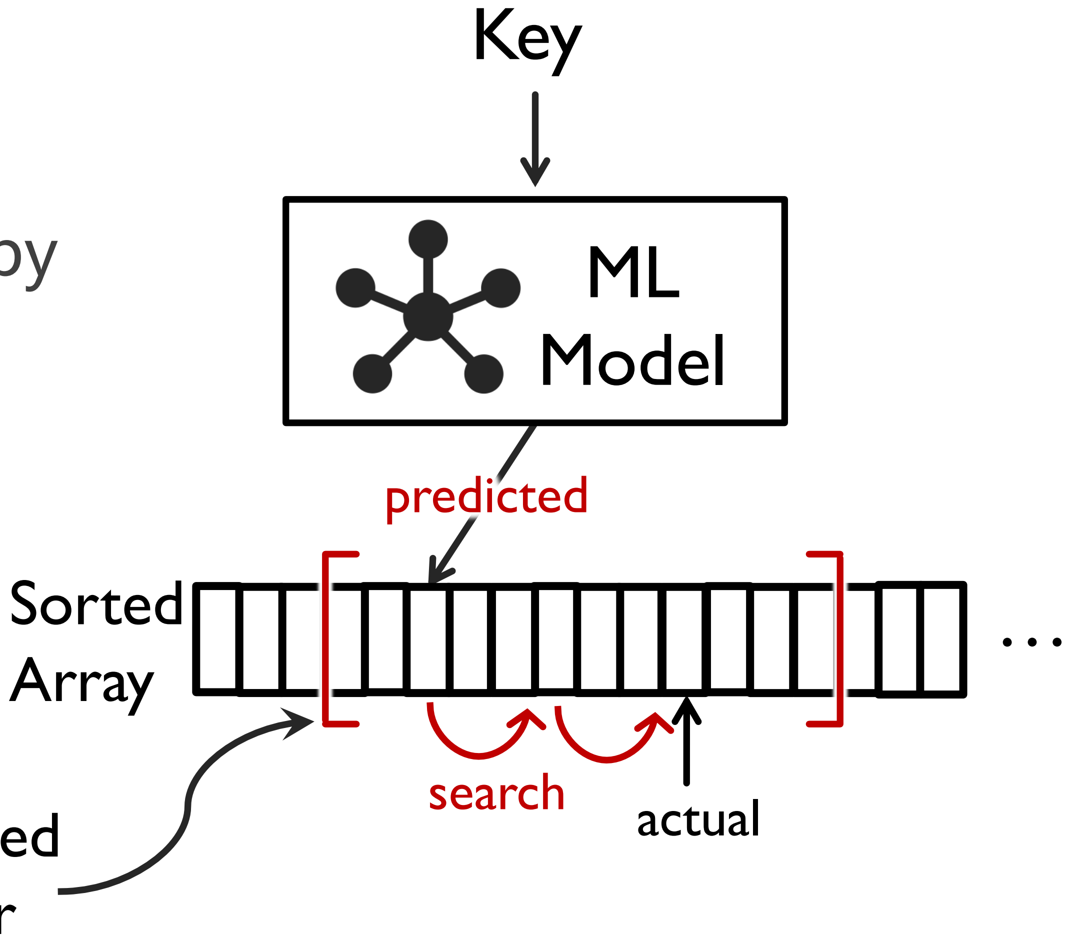


Background: the learned index

2. Perform a query

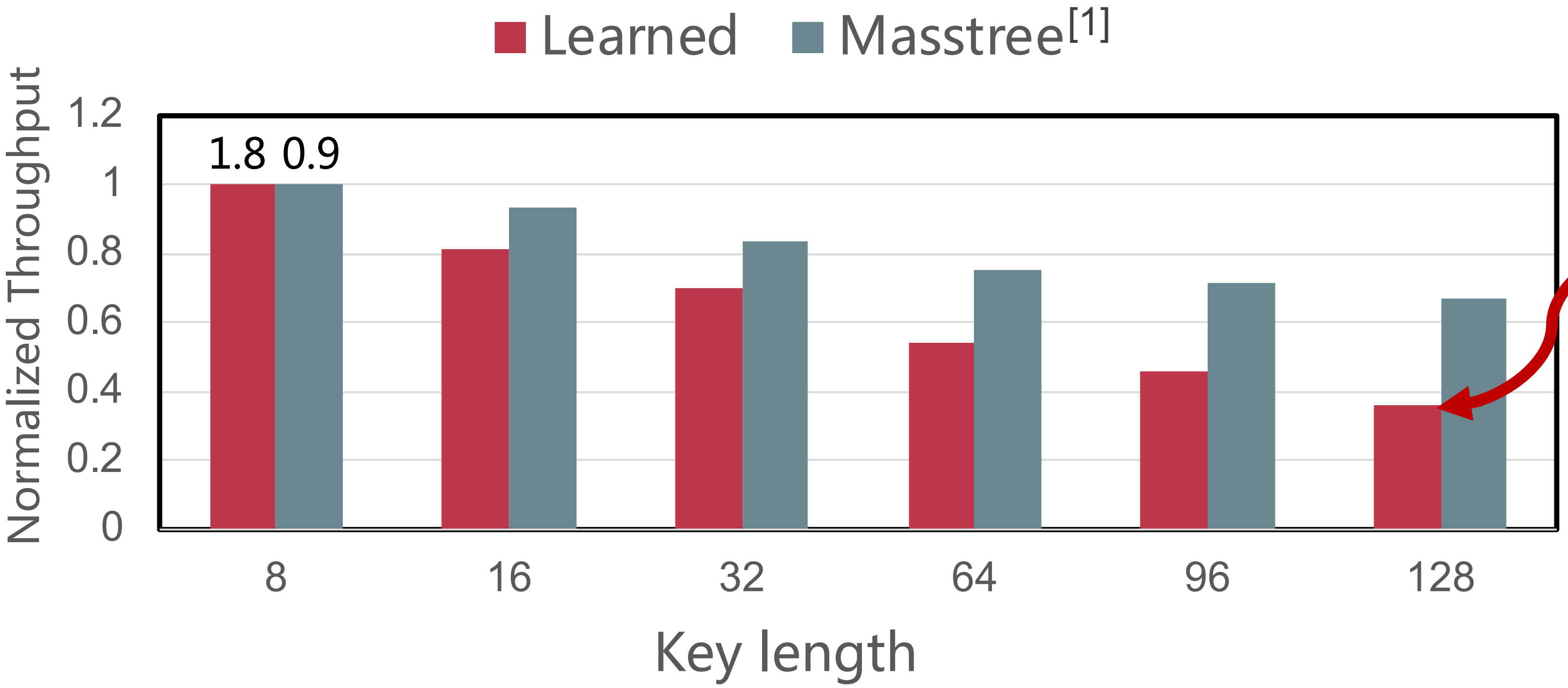
- Predict a position using model
- Search within the range bounded by the min/max error

Up to 3× perf than B-Tree



Issue under string keys

- **Read-only**

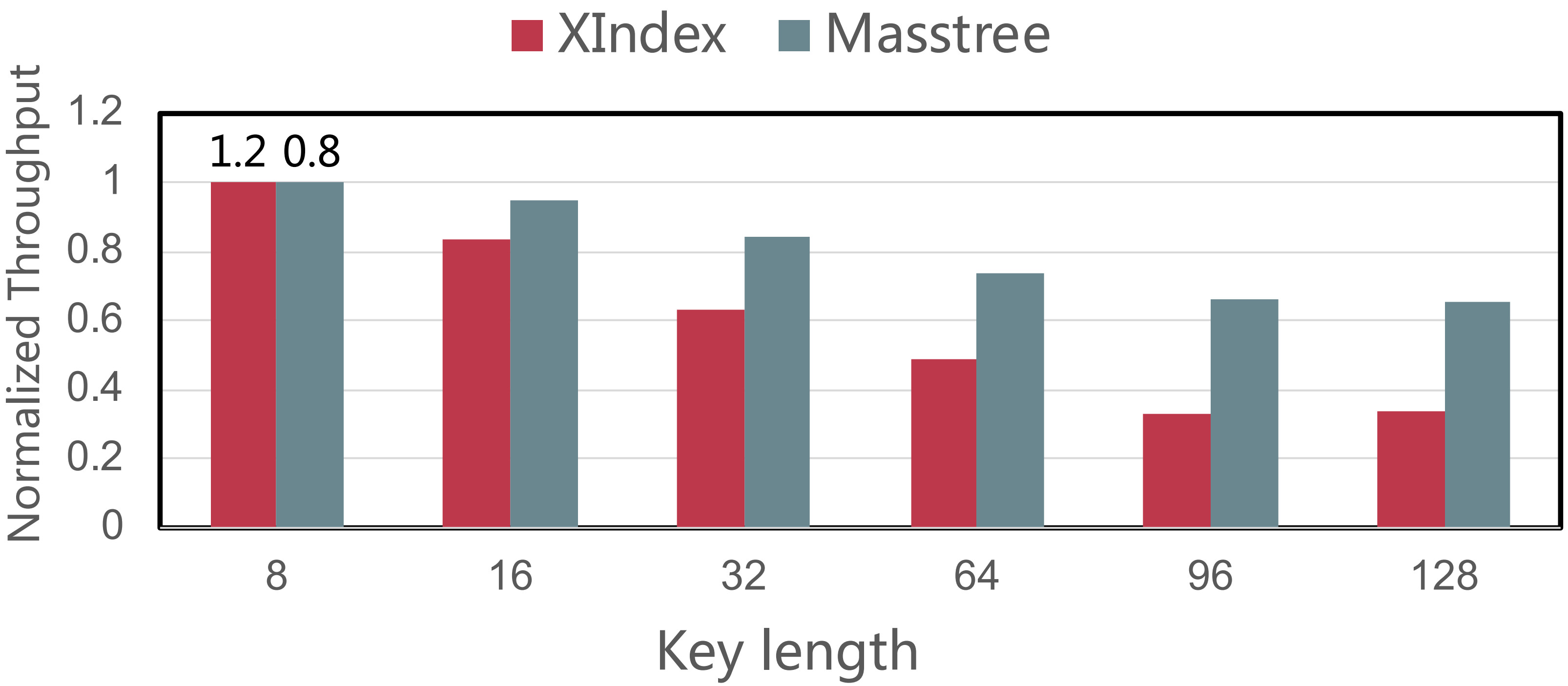


Only 35% of its 8-byte performance

[1] Mao et al. Cache Craftiness for Fast Multicore Key-Value Storage. (EuroSys'12)

Issue under string keys

- **Read-write (r:w = 9:1)**
  - XIndex[1] is a concurrent learned index



[1] Tang et al., XIndex: A Scalable Learned Index for Multicore Data Storage (PPoPP '20)

Why is perf poor under strings?

- **Model computation cost**
  - Feature length equals key length
  - Increase by more than 20× from 8 bytes to 128 bytes
- **Data access cost**
  - Key comparison cost is proportional to key length
  - Increase by 2.3× from 8 bytes to 128 bytes

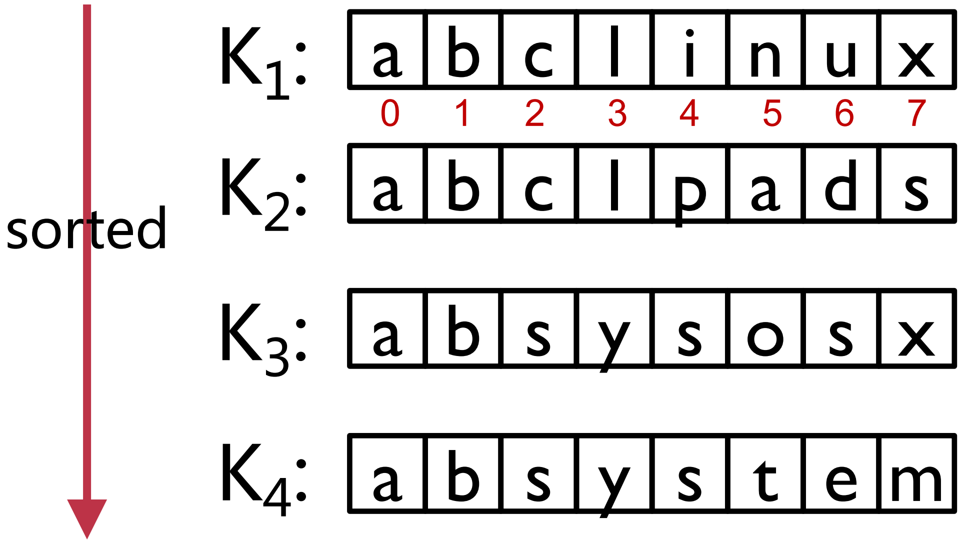
Our solution: SIndex

Main idea: use partial key (order-preserving substring)

- **As model features**
  - Reduce model inference cost
- **For key comparison**
  - Reduce data access cost

Partial key

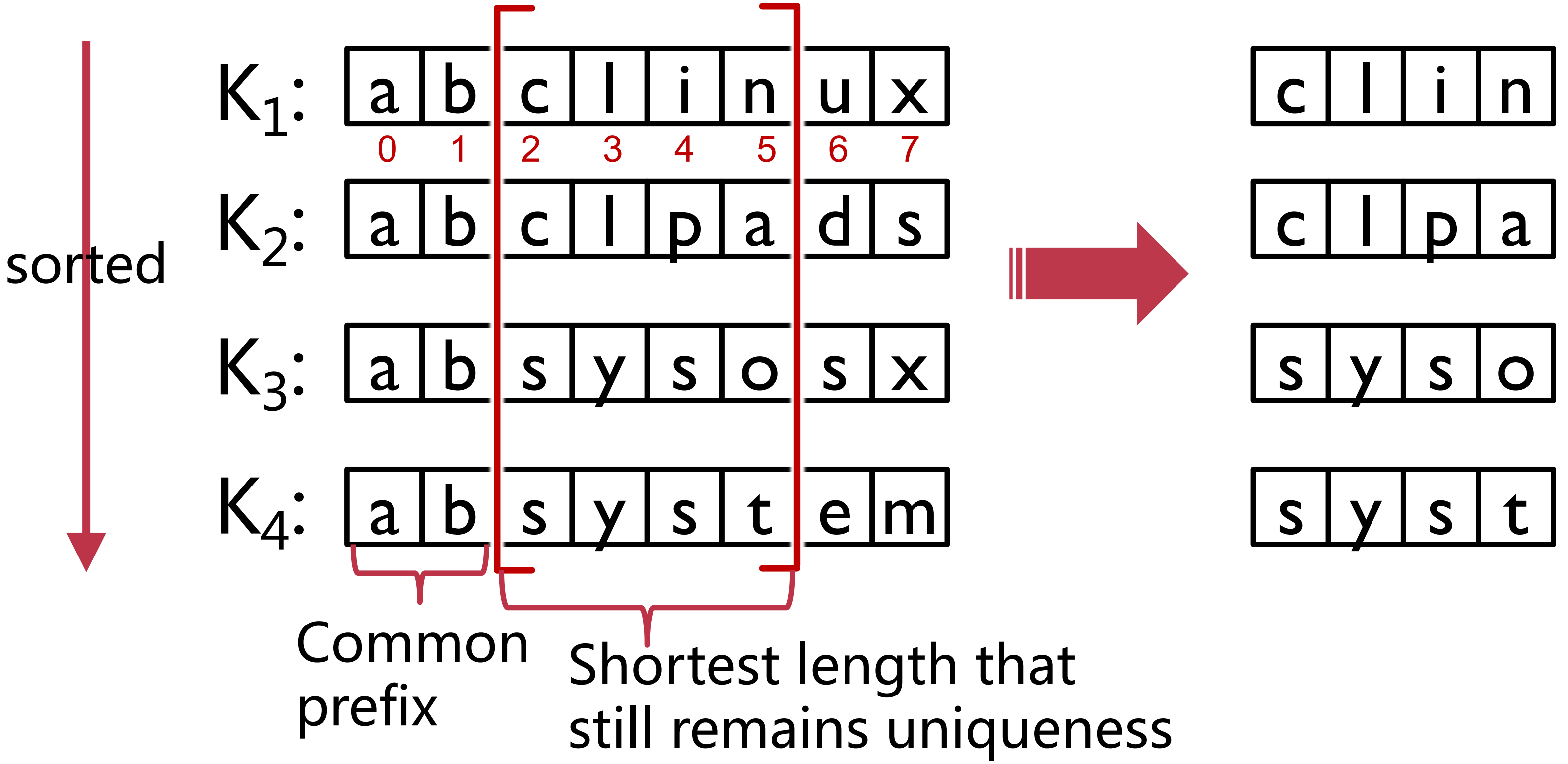
Partial key: the shortest order-preserving substring that remains uniqueness



	0	1	2	3	4	5	6	7
K_1:	a	b	c	l	i	n	u	x
K_2:	a	b	c	l	p	a	d	s
K_3:	a	b	s	y	s	o	s	x
K_4:	a	b	s	y	s	t	e	m

Partial key

Partial key: the shortest order-preserving substring that remains uniqueness



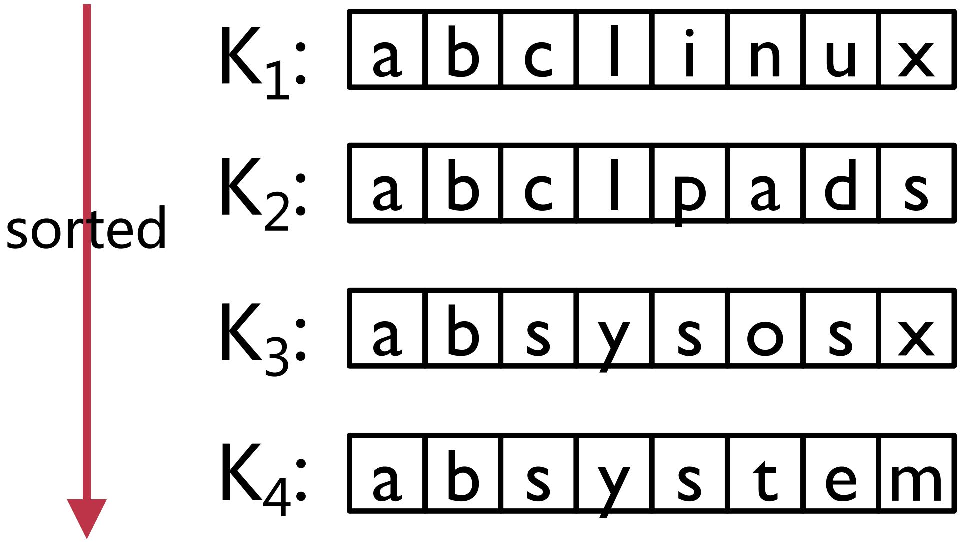
Partial key within each group

- **Applying partial key on the entire dataset may not be effective**
  - The length of partial keys may still be long
- **SIndex applies partial key within a group of keys**
  - Adopt a greedy grouping strategy to range-partition data into groups

Greedy grouping strategy

Greedily range-partition data into different groups

- Ensure the partial key length and model error are under thresholds



K_1: a b c l i n u x

K_2: a b c l p a d s

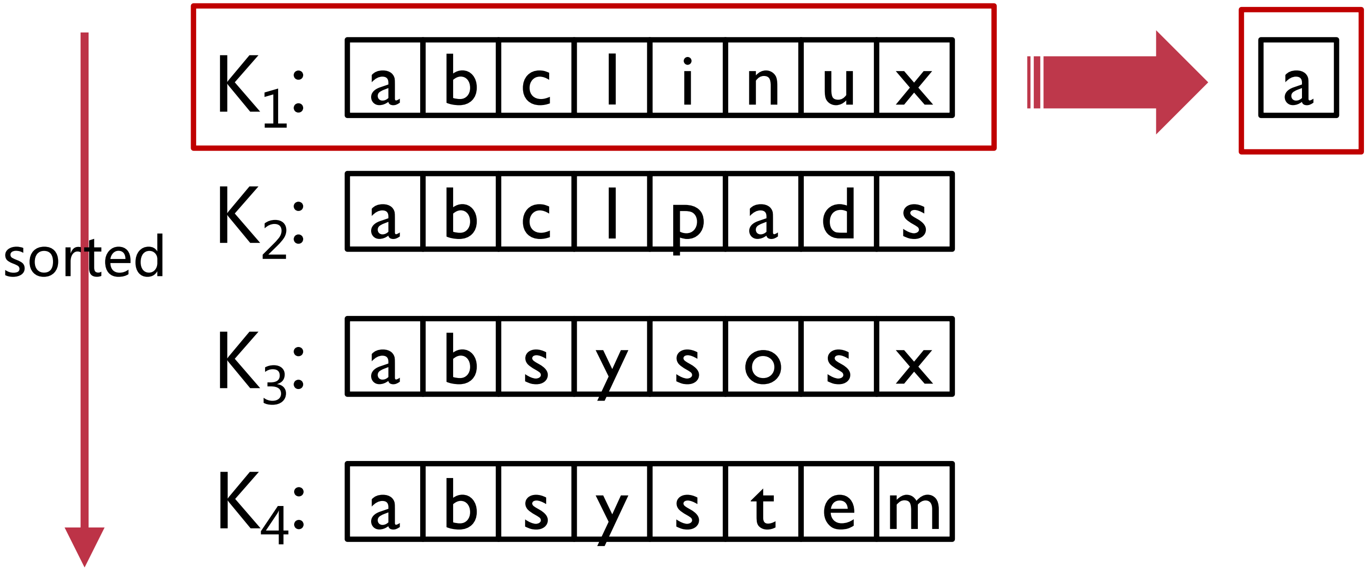
K_3: a b s y s o s x

K_4: a b s y s t e m

Greedy grouping strategy

Greedily range-partition data into different groups

Partial key length threshold: 2



sorted

K_1: a b c l i n u x

K_2: a b c l p a d s

K_3: a b s y s o s x

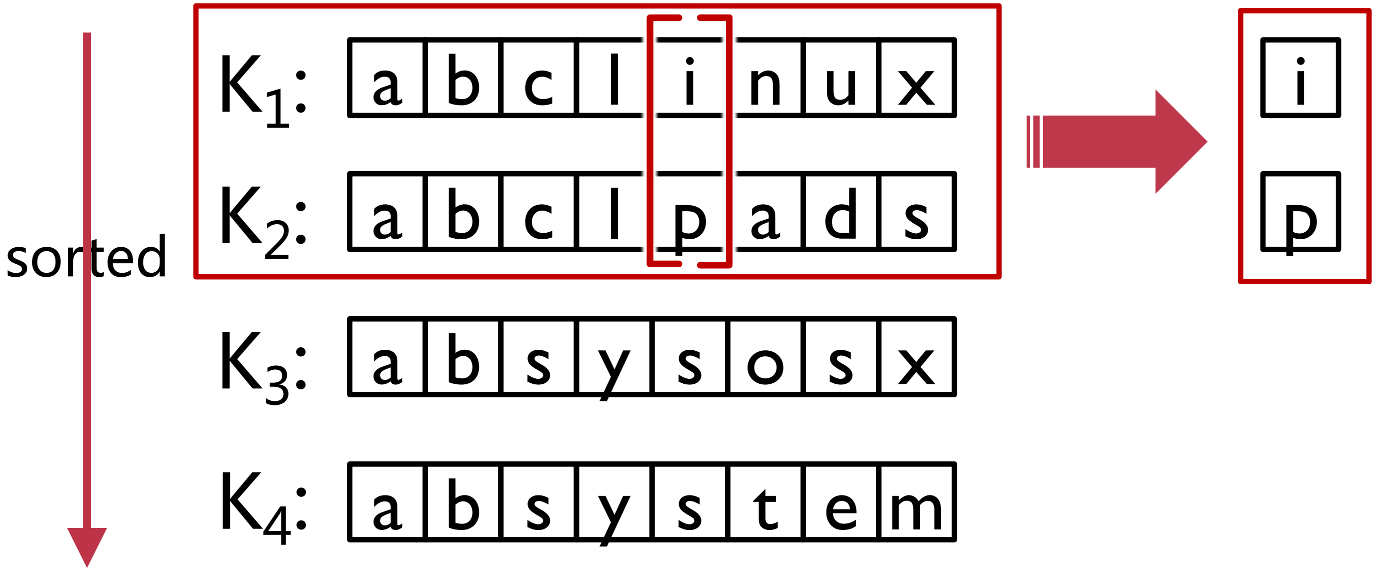
K_4: a b s y s t e m

a

Greedy grouping strategy

Greedily range-partition data into different groups

Partial key length threshold: 2

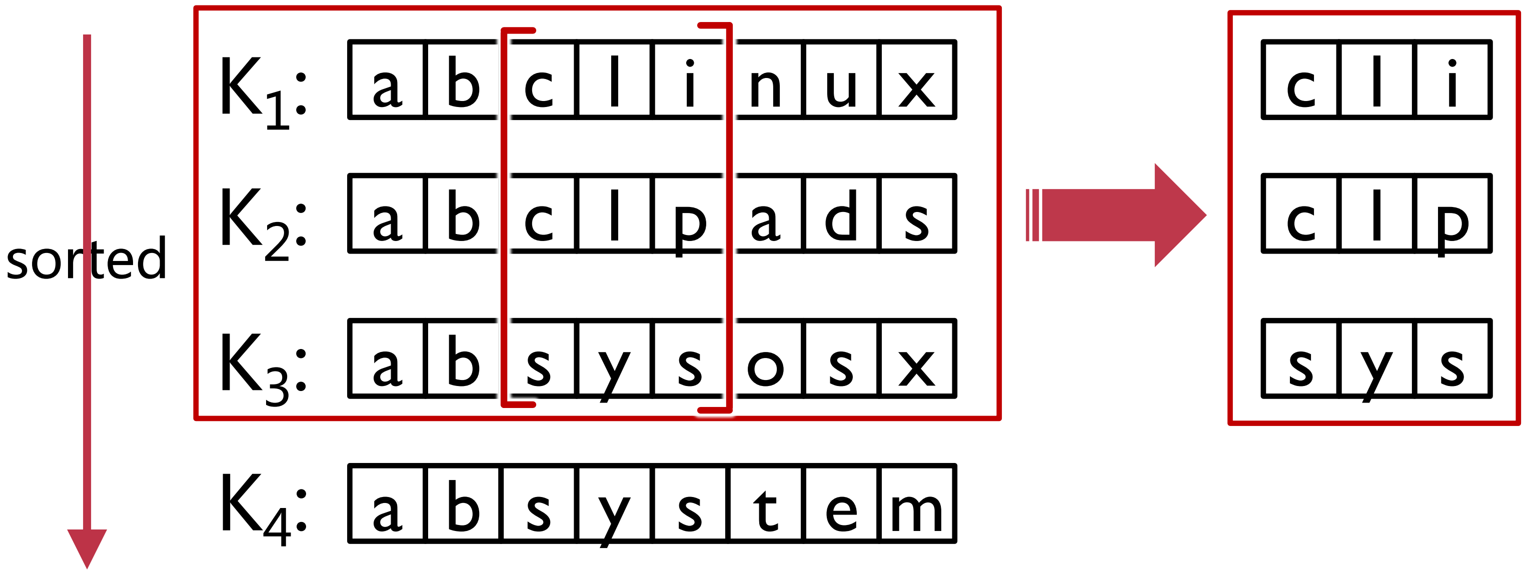


K_1:	a	b	c	l	i	n	u	x
K_2:	a	b	c	l	p	a	d	s
K_3:	a	b	s	y	s	o	s	x
K_4:	a	b	s	y	s	t	e	m

Greedy grouping strategy

Greedily range-partition data into different groups

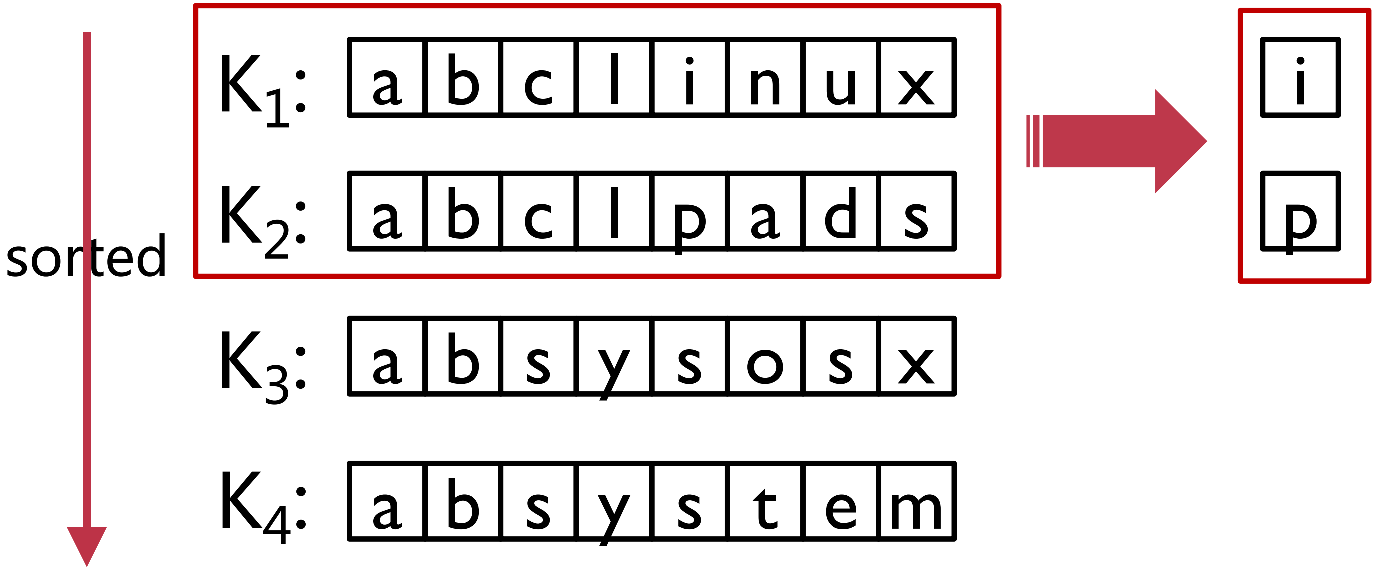
Partial key length threshold: 2



Greedy grouping strategy

Greedily range-partition data into different groups

Partial key length threshold: 2



sorted

K_1: a b c l i n u x

K_2: a b c l p a d s

K_3: a b s y s o s x

K_4: a b s y s t e m

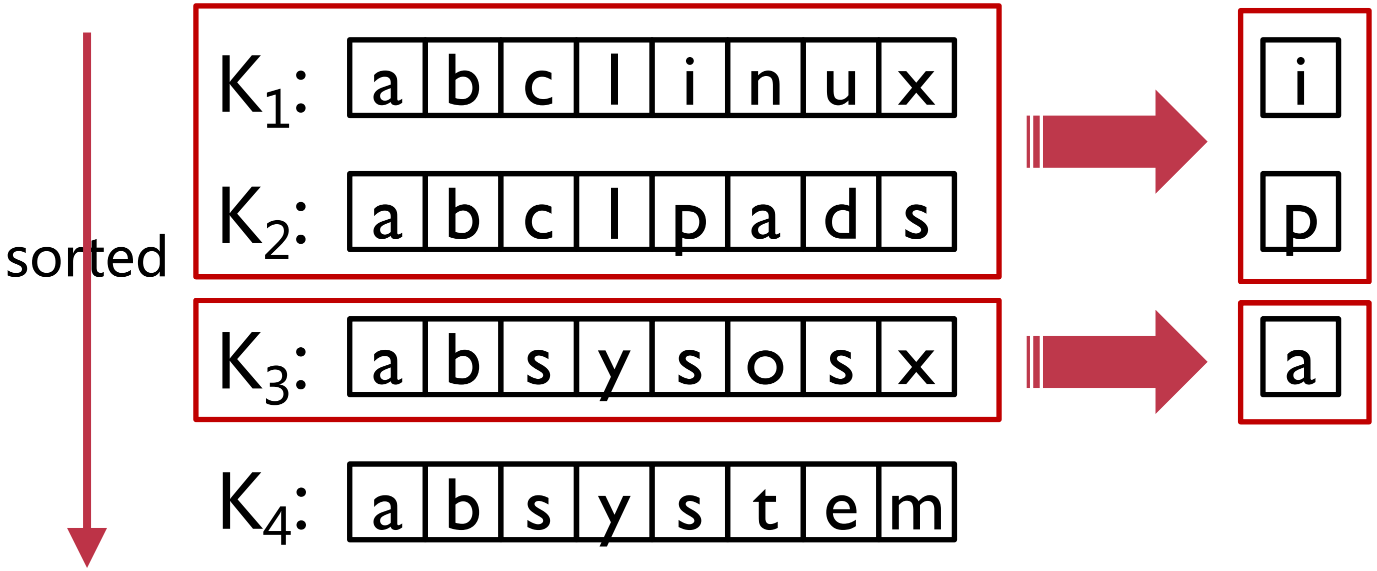
i

p

Greedy grouping strategy

Greedily range-partition data into different groups

Partial key length threshold: 2



sorted

K_1:	a	b	c	l	i	n	u	x
K_2:	a	b	c	l	p	a	d	s
K_3:	a	b	s	y	s	o	s	x
K_4:	a	b	s	y	s	t	e	m

i

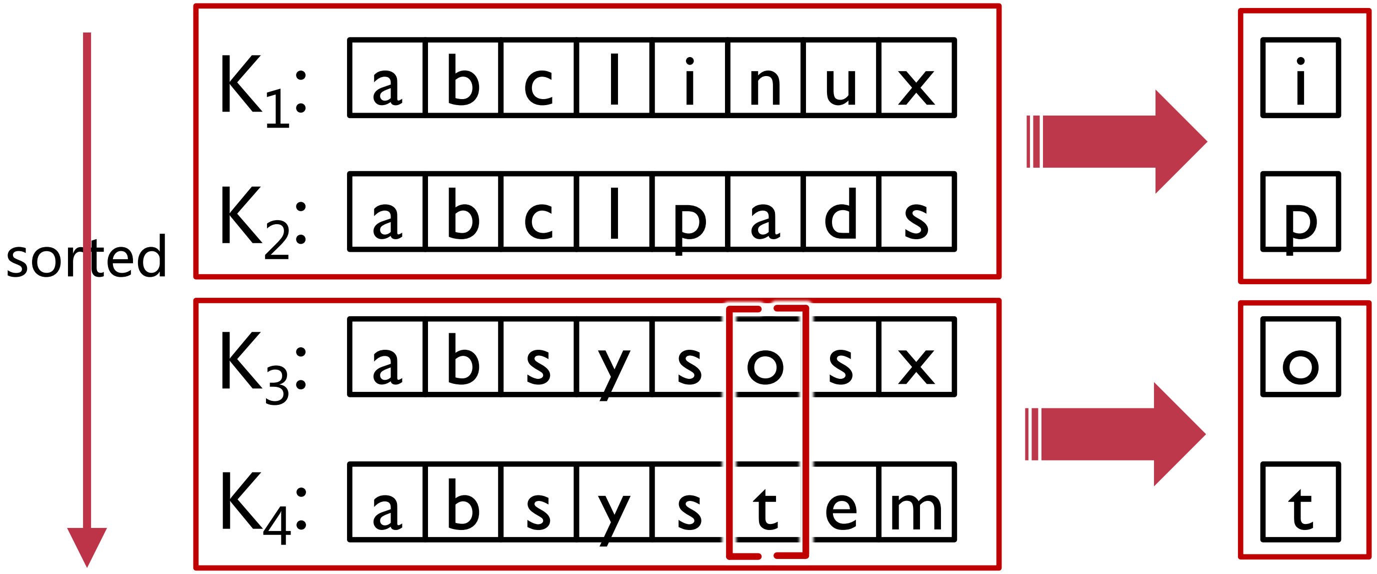
p

a

Greedy grouping strategy

Greedily range-partition data into different groups

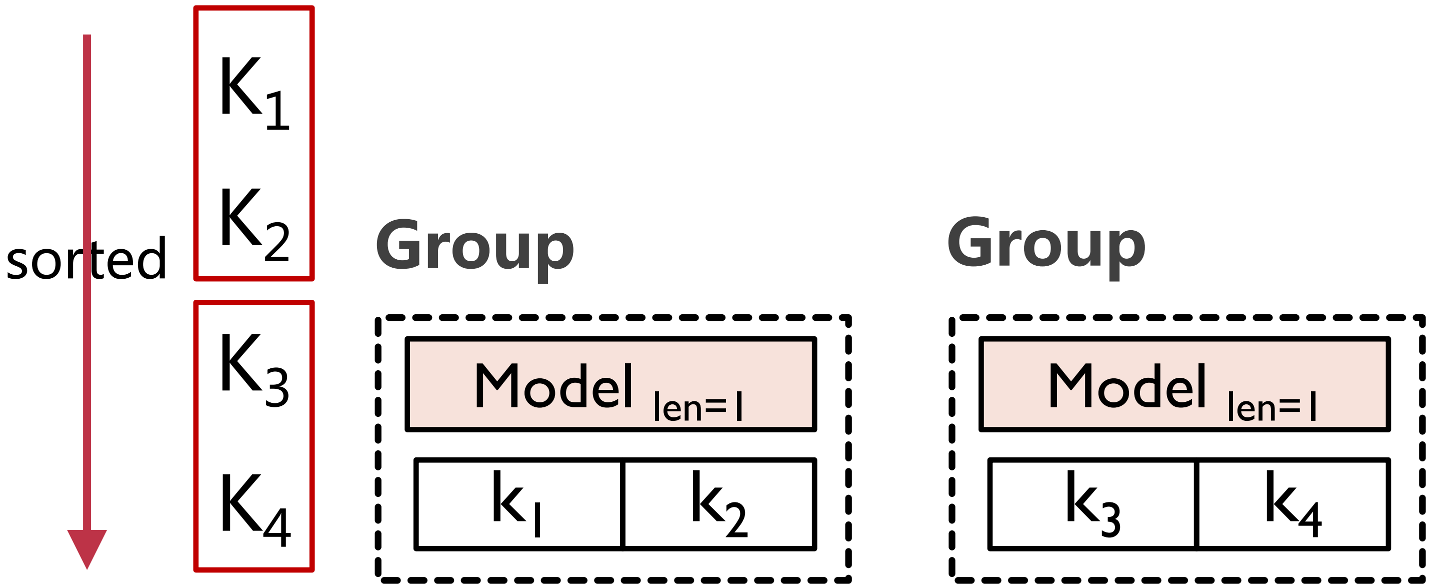
Partial key length threshold: 2



Greedy grouping strategy

Greedily range-partition data into different groups

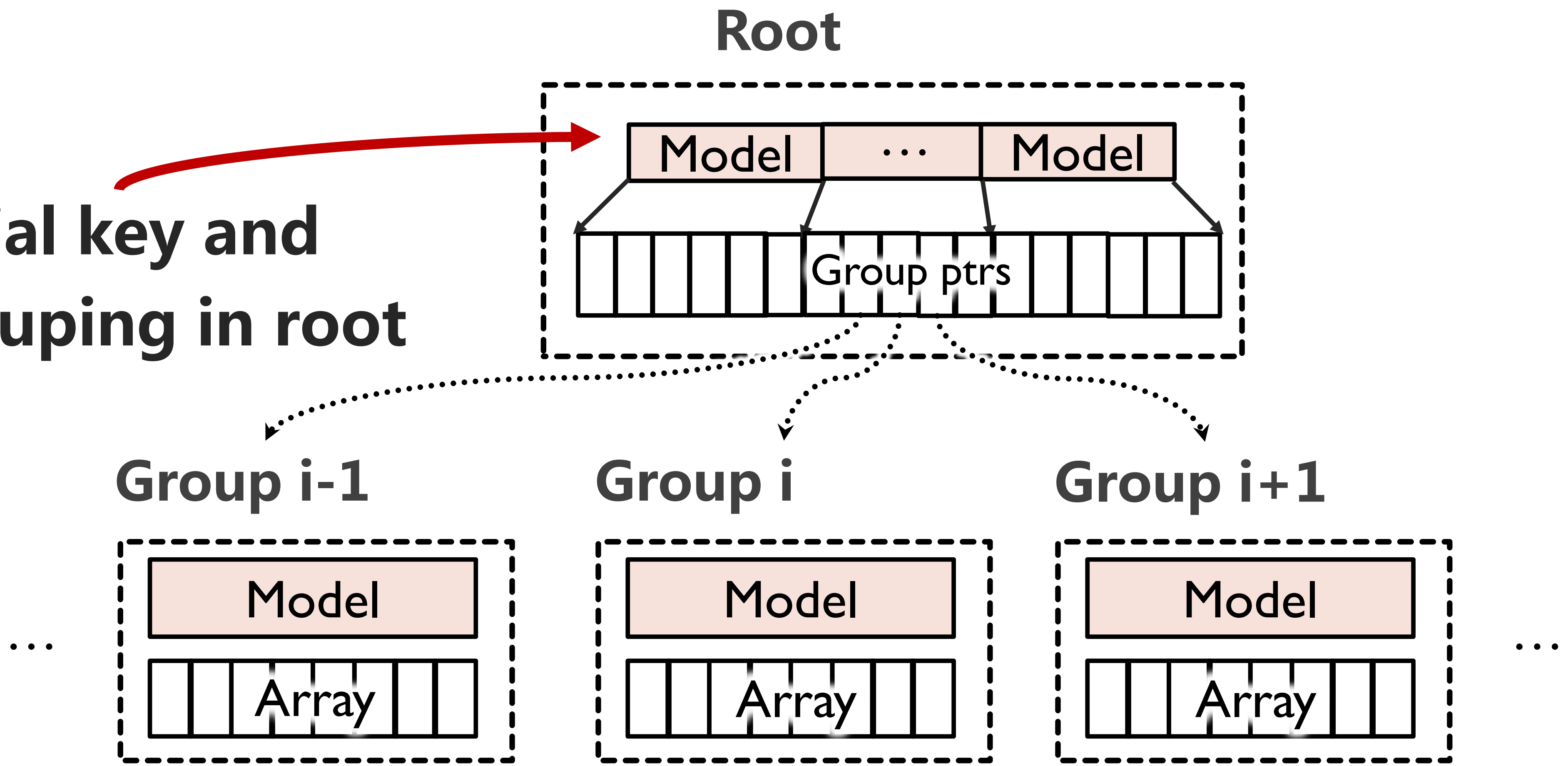
Partial key length threshold: 2



Two groups, with partial key length of 1 byte

SIndex: two-layer design

Apply partial key and greedy grouping in root

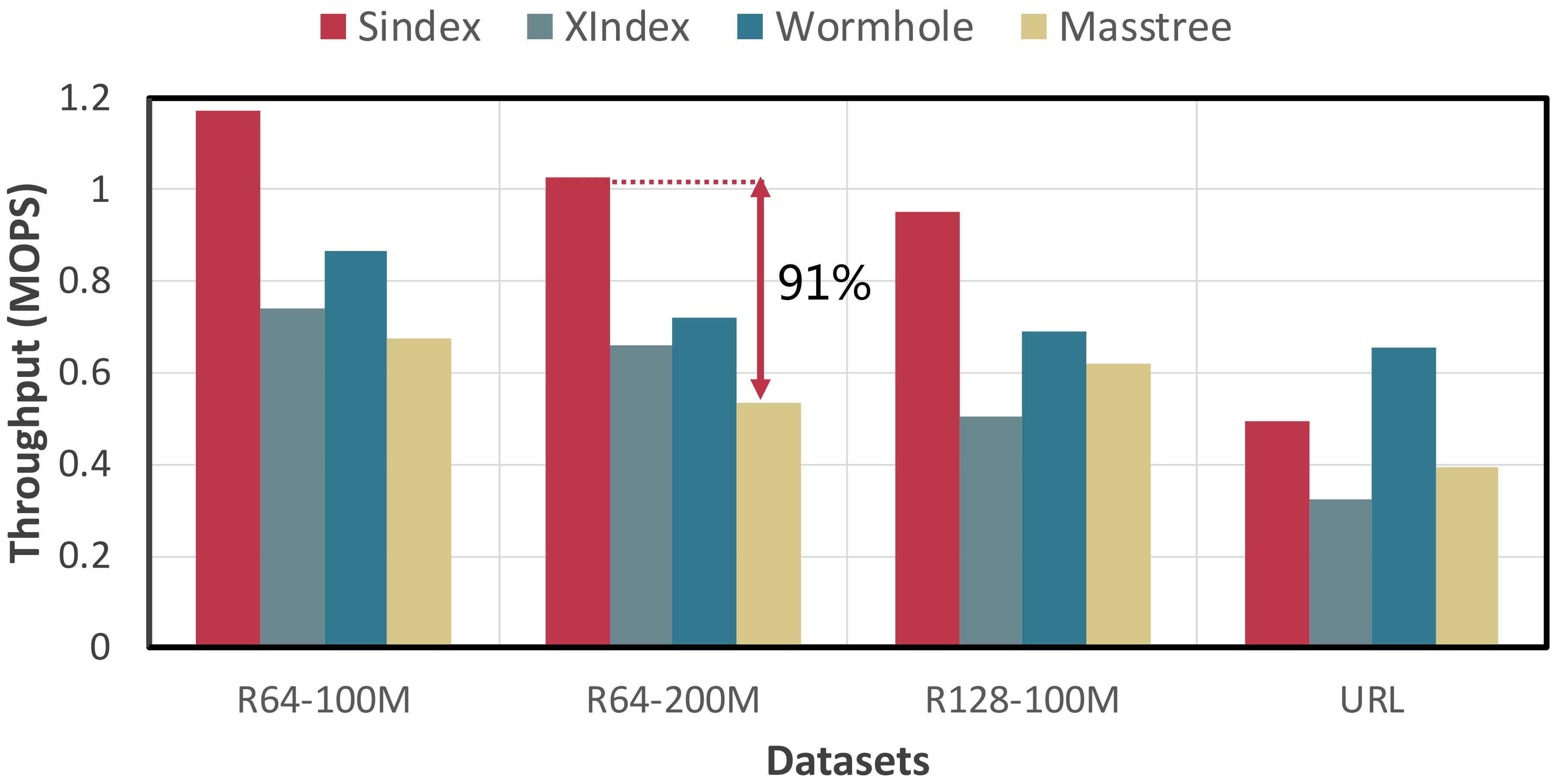


Evaluation

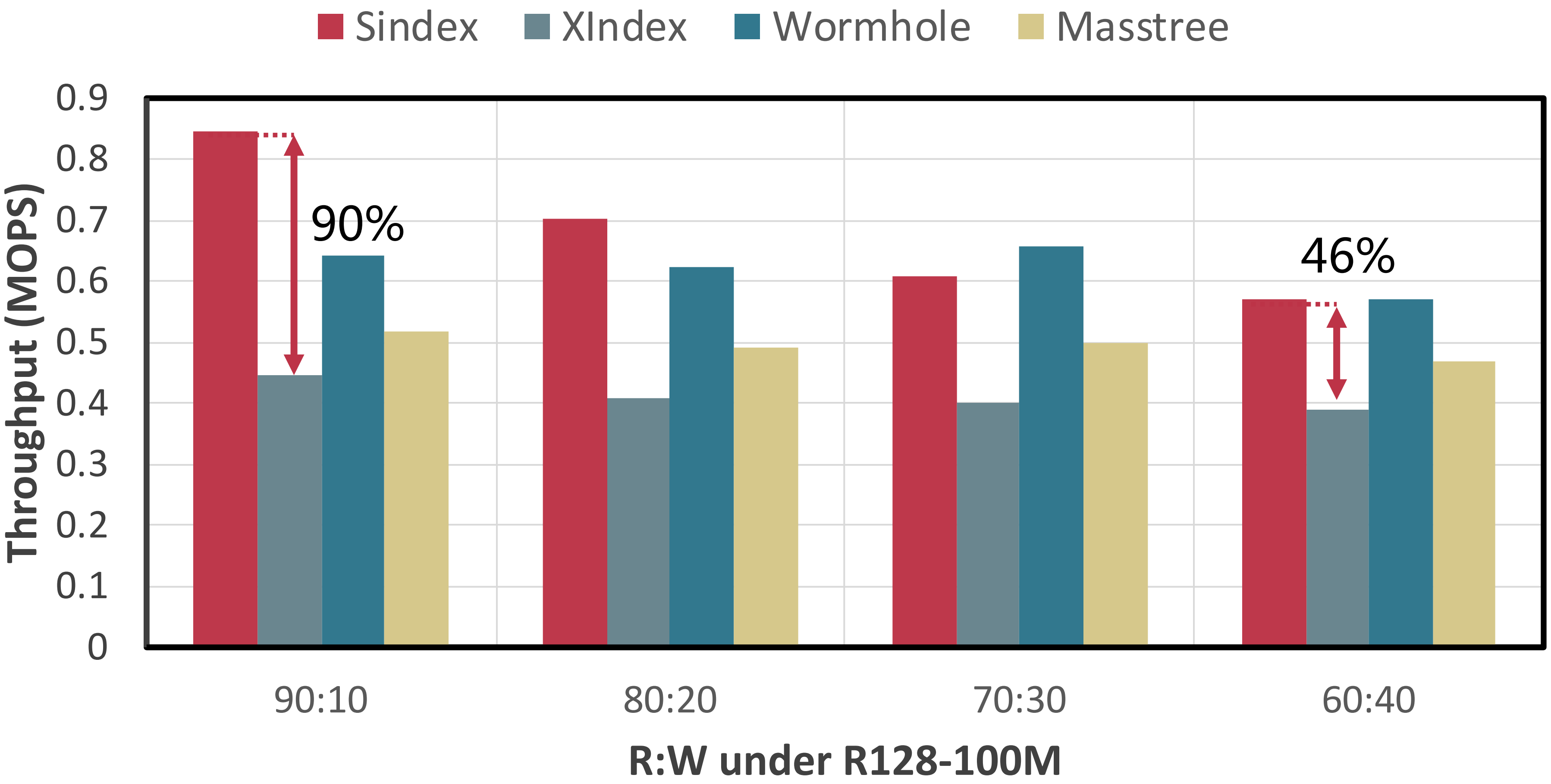
- **Implementation**
  - Adopt SIMD instructions to perform model computation
  - Use the compaction mechanism in XIndex to handle writes
- **Counterparts**
  - XIndex, Masstree [EuroSys '12], Wormhole [EuroSys '19]
- **Datasets**
  - Random (denote as R-[key length]-[size]), URL (128 bytes) from Memetracker[1]

[1] Leskovec et al., MemeTracking and the Dynamics of the News Cycle

Read-only


Sindex
XIndex
Wormhole
Masstree
Throughput (MOPS)
1.2
1
0.8
0.6
0.4
0.2
0
91%
R64-100M
R64-200M
R128-100M
URL
Datasets

Read-write



Sindex
XIndex
Wormhole
Masstree
Throughput (MOPS)
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
90%
46%
90:10
80:20
70:30
60:40
R:W under R128-100M

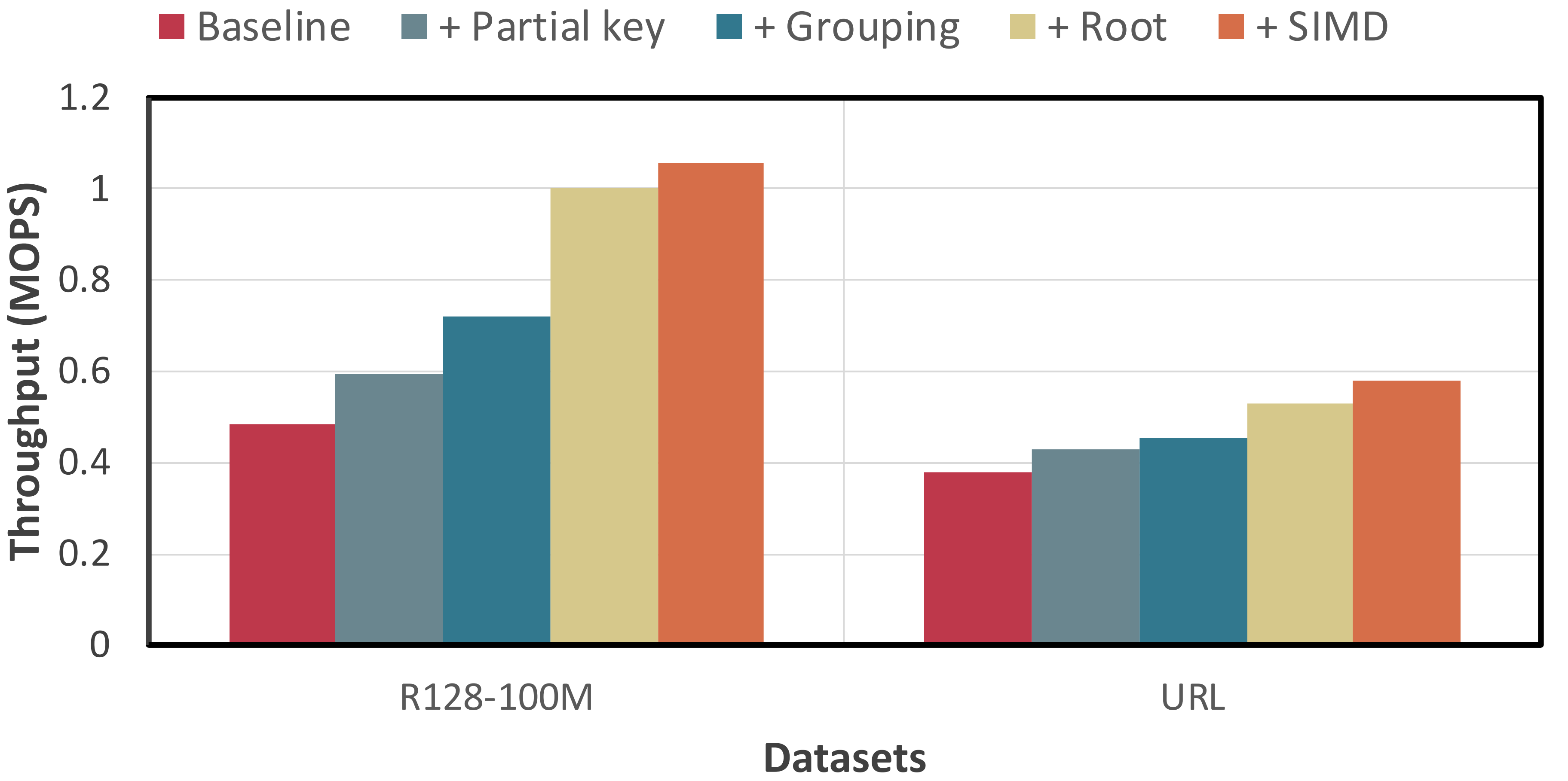
SIndex

- The first learned index for string keys
- Exploit **partial key** to reduce model inference and data access overheads
- Up to 91% perf improvement compared with state-of-the-arts

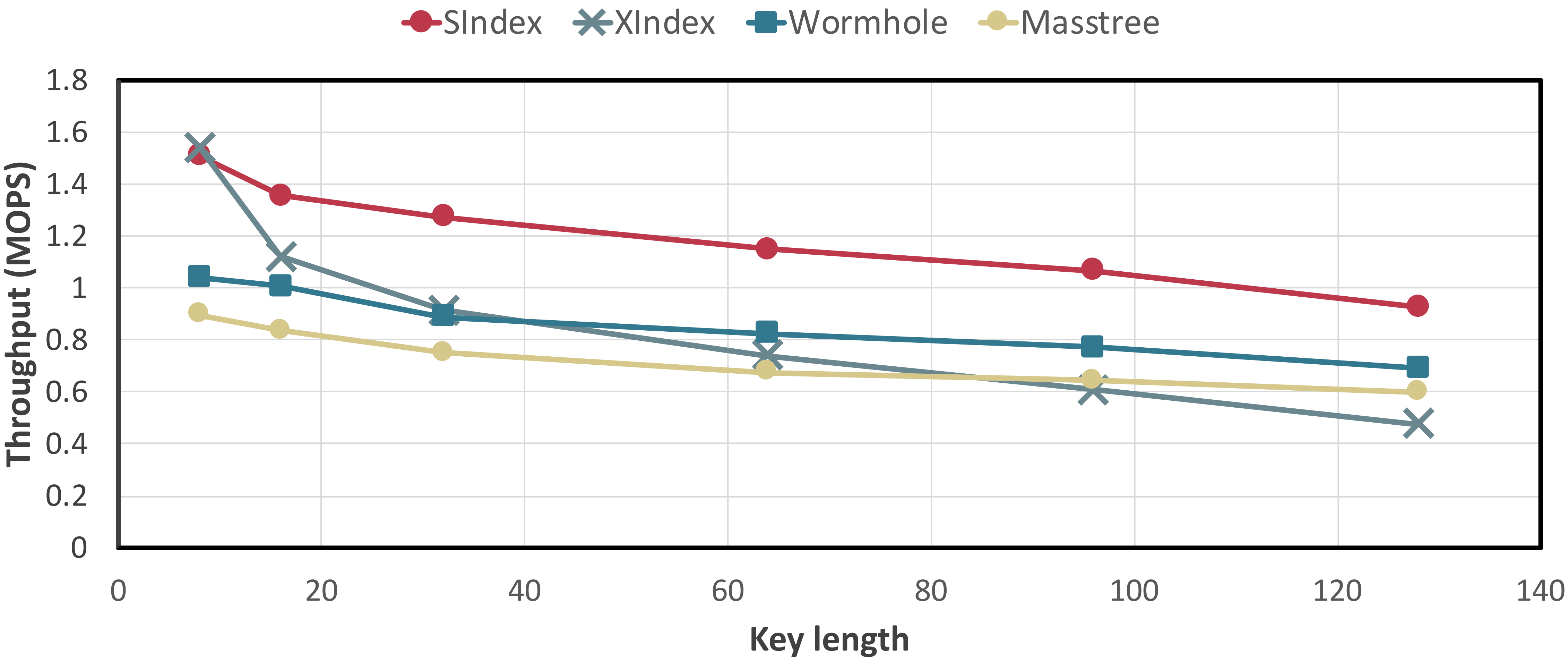
Source code is available

https://ipads.se.sjtu.edu.cn:1312/opensource/xindex/-/tree/sindex

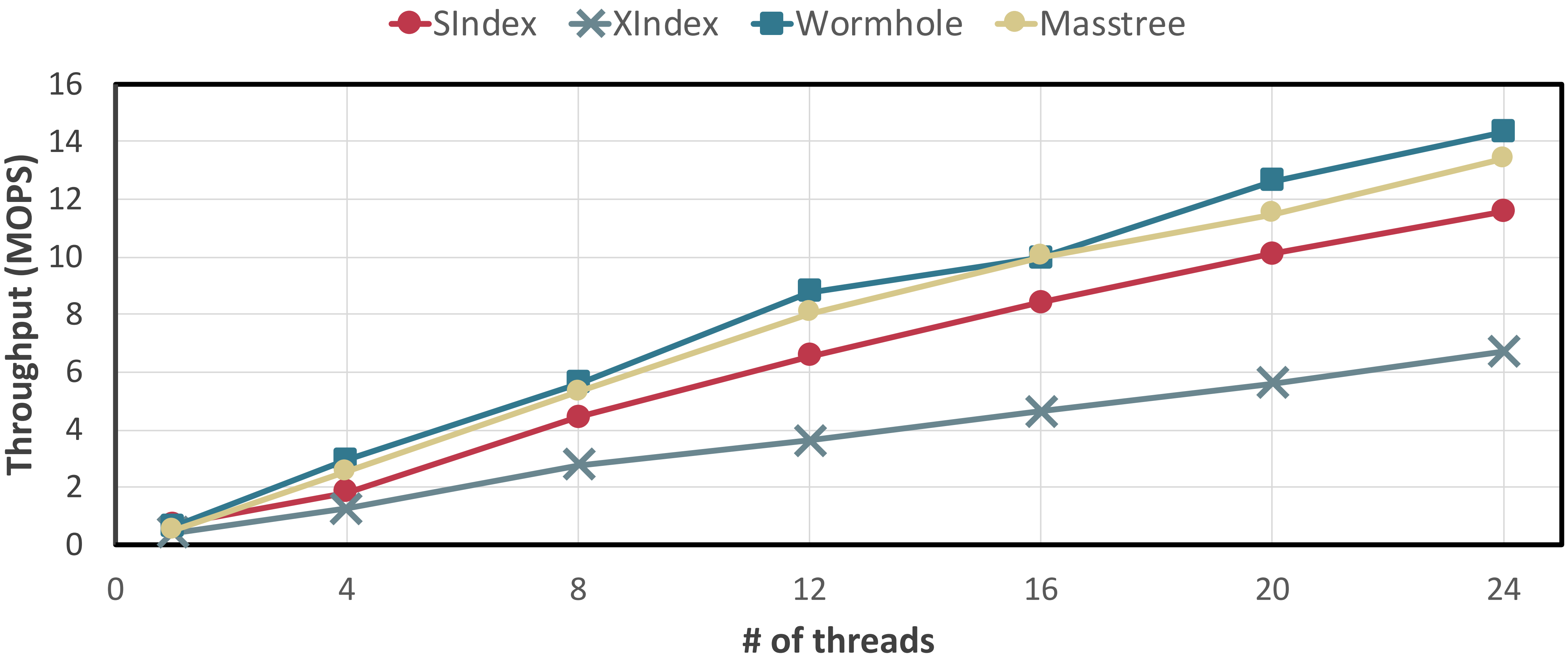
Breakdown


Baseline
+ Partial key
+ Grouping
+ Root
+ SIMD
Throughput (MOPS)
1.2
1
0.8
0.6
0.4
0.2
0
R128-100M
URL
Datasets

Read-only

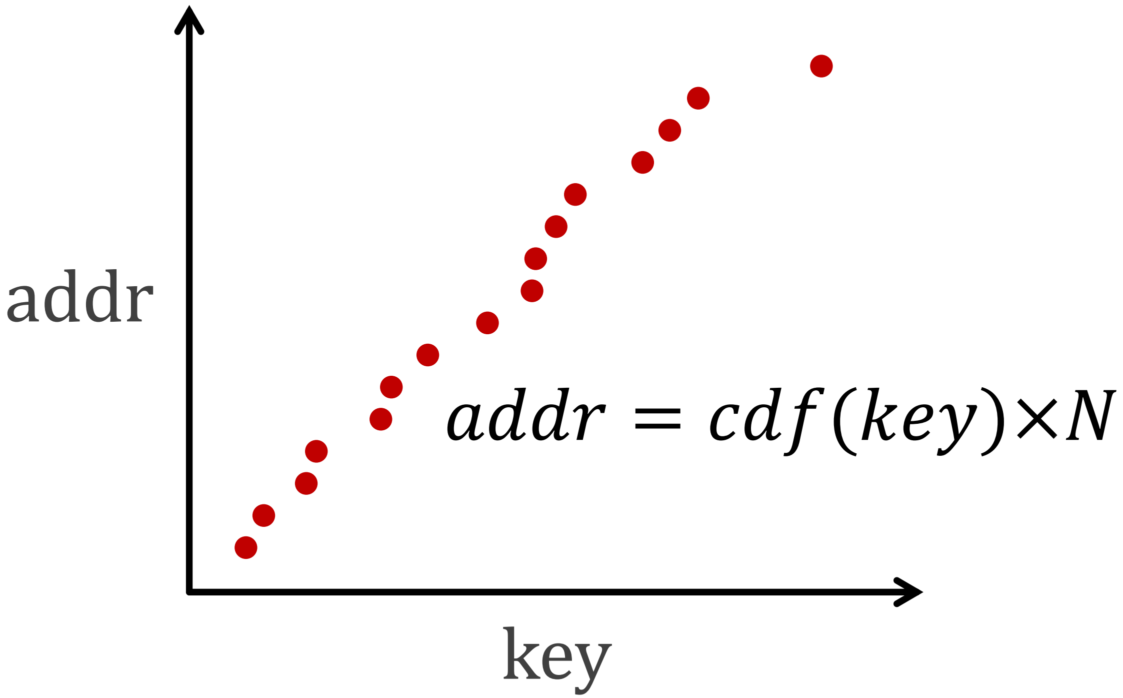

SIndex
XIndex
Wormhole
Masstree
Throughput (MOPS)
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
100
120
140
Key length

Scalability

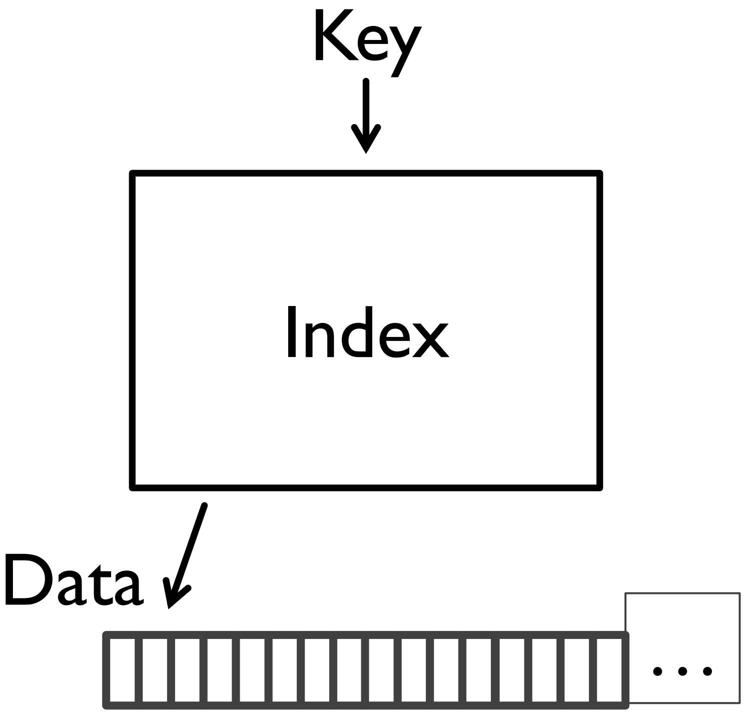


Background: the learned index

- **With contiguously sorted data, index functions are CDFs (cumulative distribution functions)**



$$addr = cdf(key) \times N$$


Key
Index
Data
…