Search Lookaside Buffer:

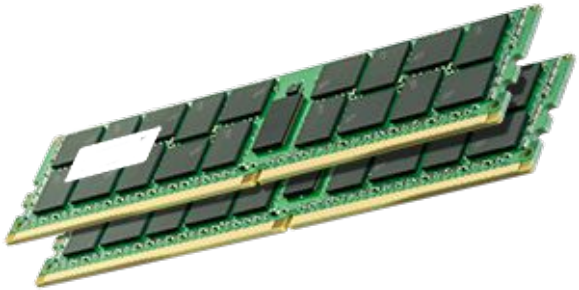
Efficient Caching for Index Data Structures

UNIVERSITY OF
TEXAS
ARLINGTON

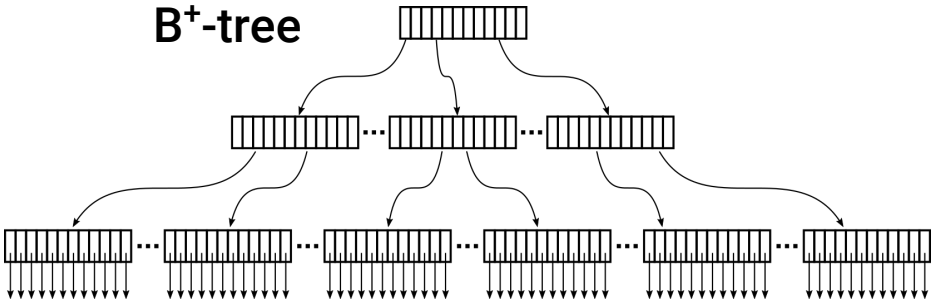
Xingbo Wu, Fan Ni, Song Jiang

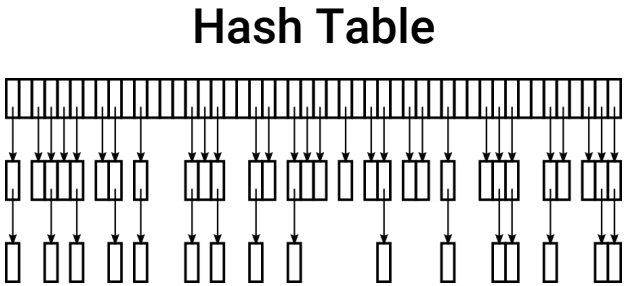
Background

- Large-scale in-memory applications.
  - In-memory databases
  - In-memory NoSQL stores and caches
  - Software routing tables



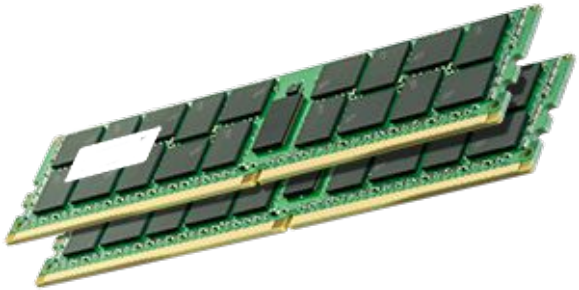
- They rely on **index data structures** to access their data.





Background

- Large-scale in-memory applications.
  - In-memory databases
  - In-memory NoSQL stores and caches
  - Software routing tables



- They rely on **index data structures** to access their data.
- *“hash index (i.e., hash table) accesses are the most significant single source of runtime overhead, constituting 14–94% of total query execution time.”* [Kocberber et al., MICRO-46]

CPU Cache is Not Effectively Used

- Indices are too large to fit in CPU cache.

  In-memory Database: "**55%** of the total memory". [Zhang et al., SIGMOD'16]

  In-memory KV caches: **20–40%** of the memory. [Atikoglu et al., Sigmetrics'12]

- Access **locality** has potential to address the problem.

  Facebook's Memcached workload study:

  "*All workloads exhibit the expected long-tail distributions, with a small percentage of keys appearing in most of the requests. . .*"

- However, data locality is compromised during index search.

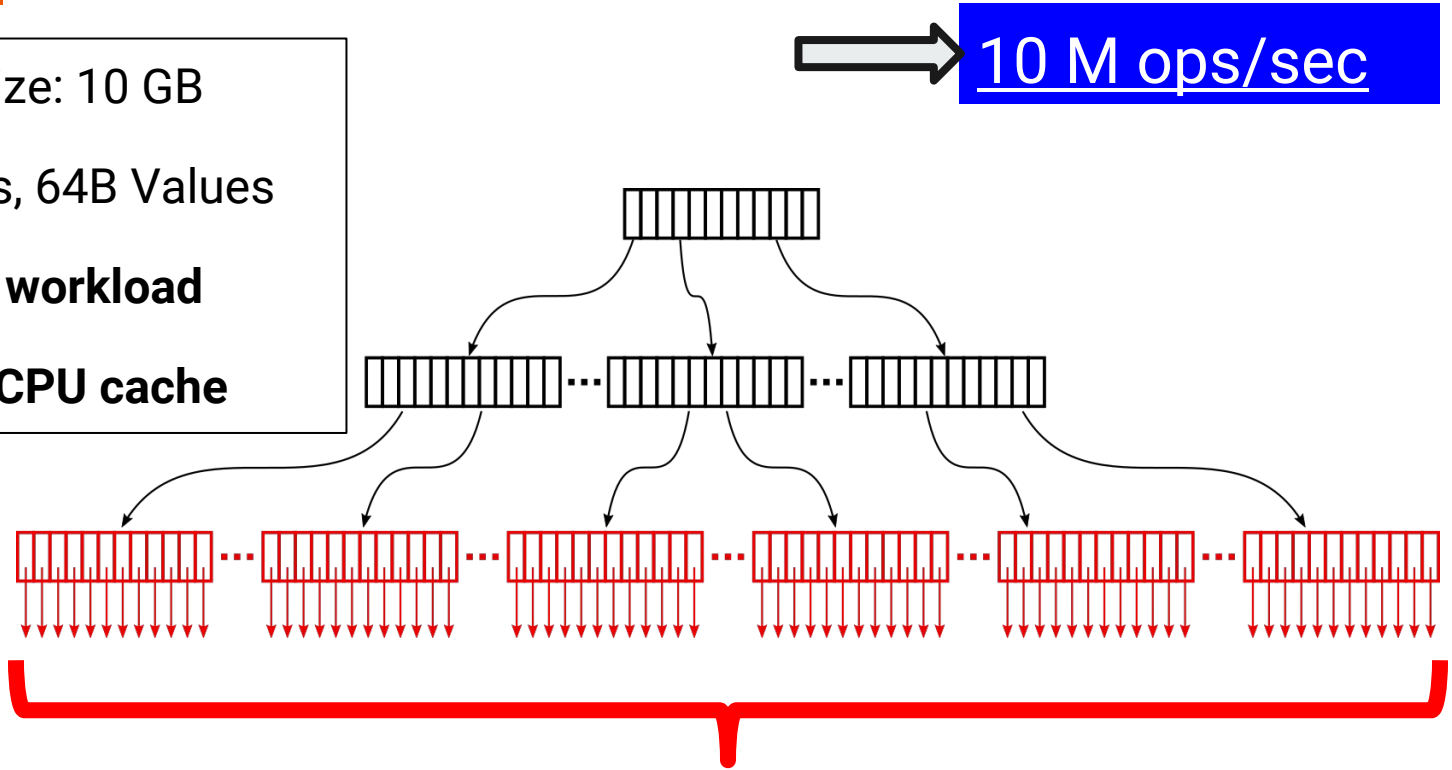
Case Study: Search in a B$^+$-tree-indexed Store

Store size: 10 GB

8B Keys, 64B Values

Zipfian workload

40 MB CPU cache



Case Study: Search in a B$^+$-tree-indexed Store

Store size: 10 GB

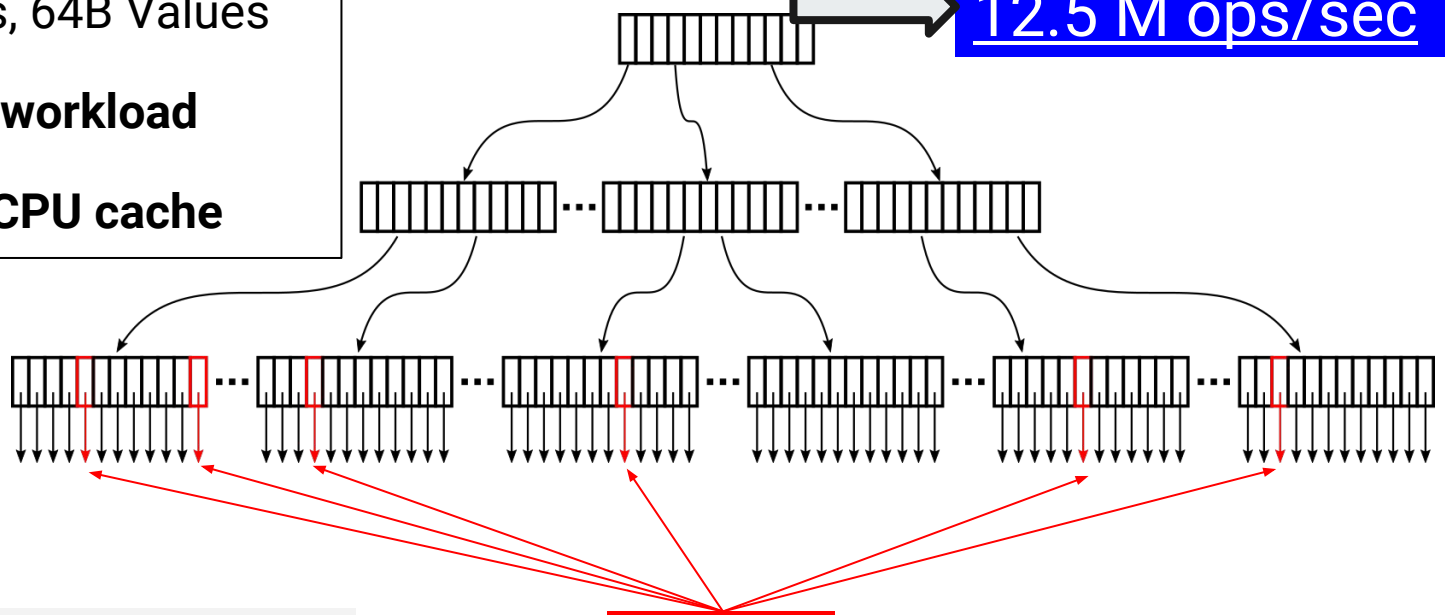
8B Keys, 64B Values

Zipfian workload

40 MB CPU cache

10 M ops/sec

12.5 M ops/sec



Accessed data set:

10 MB

Case Study: Search in a B$^+$-tree-indexed Store

Store size: 10 GB

8B Keys, 64B Values

Zipfian workload

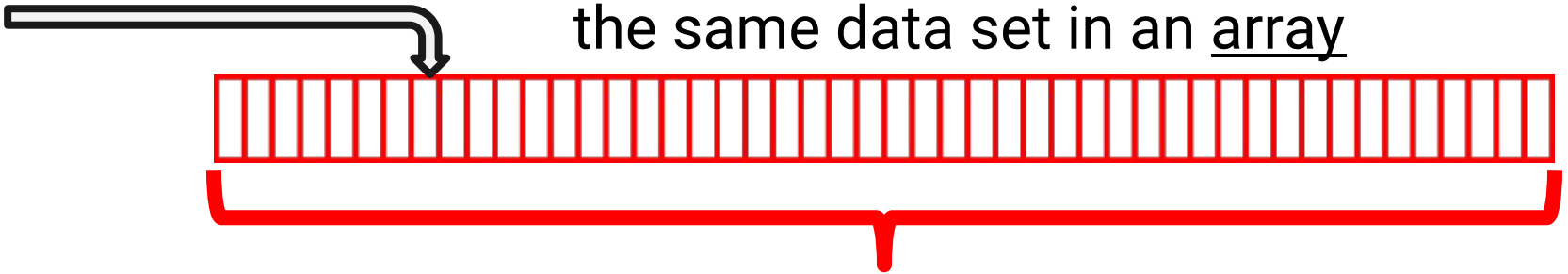
40 MB CPU cache

10 M ops/sec

12.5 M ops/sec

382 M ops/sec

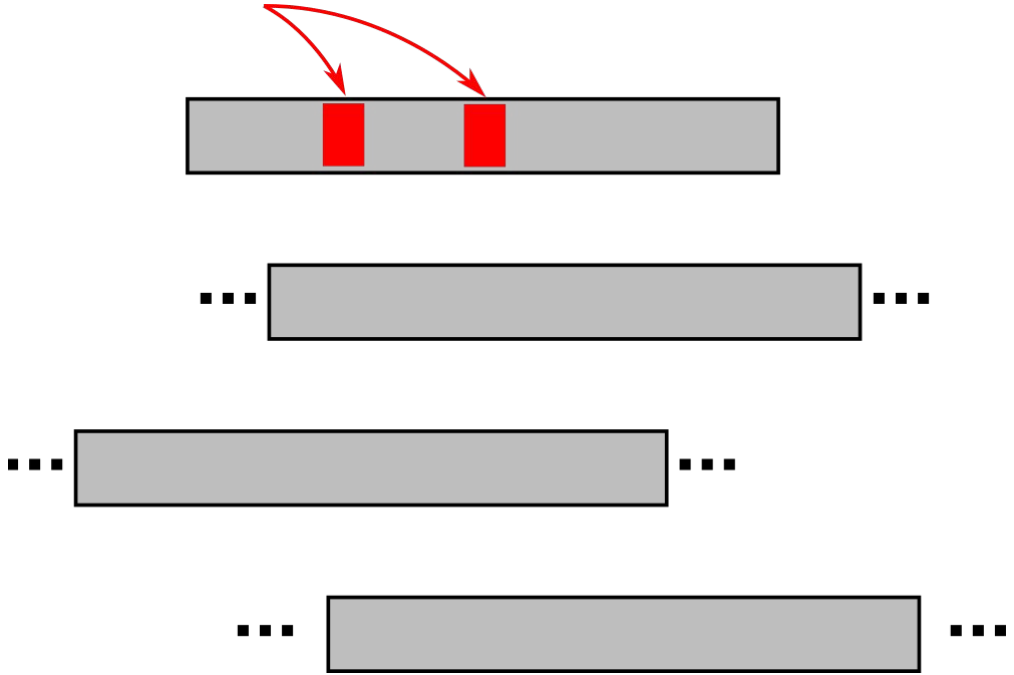
If we remove the index and put the same data set in an array



Accessed data set: 10 GB

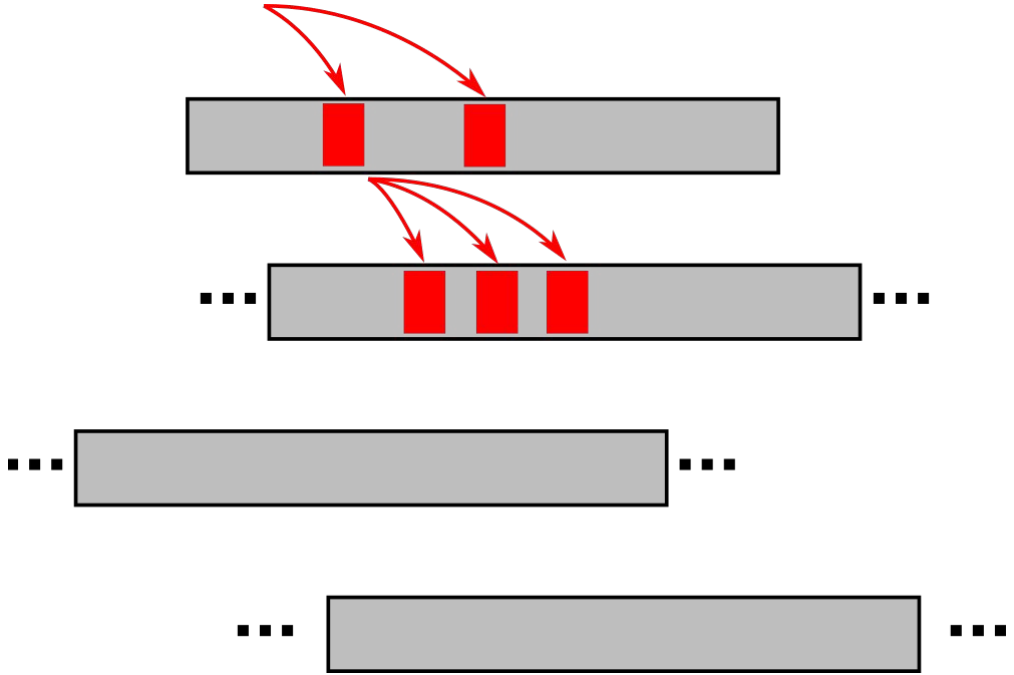
A Look at Index Traversal

- Index search in B^+-tree: binary search at each node



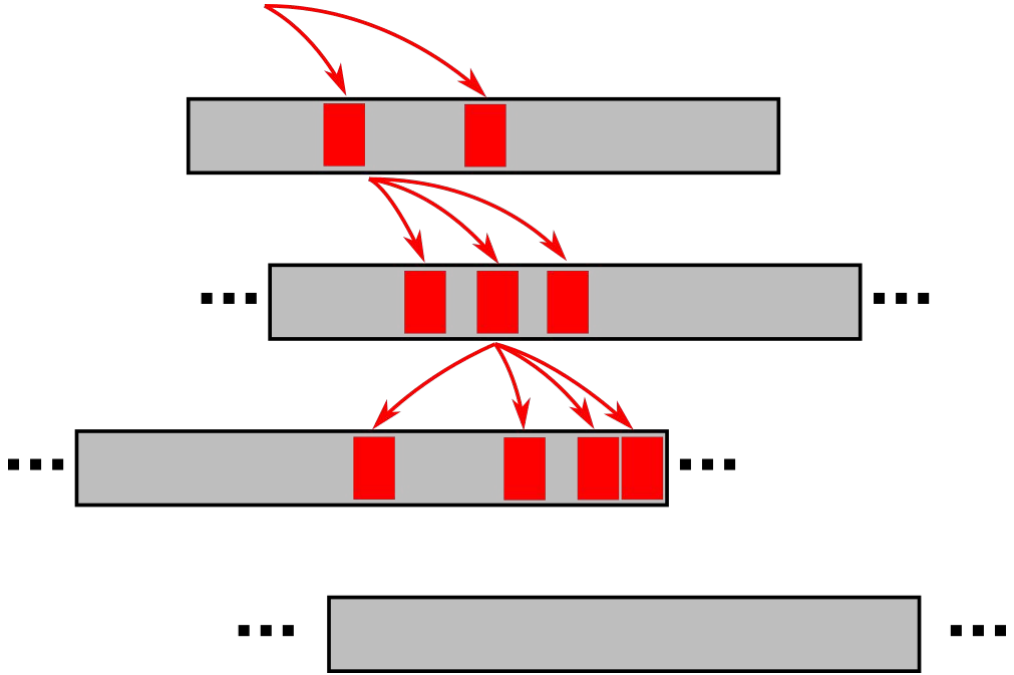
A Look at Index Traversal

- Index search in B^+-tree: binary search at each node



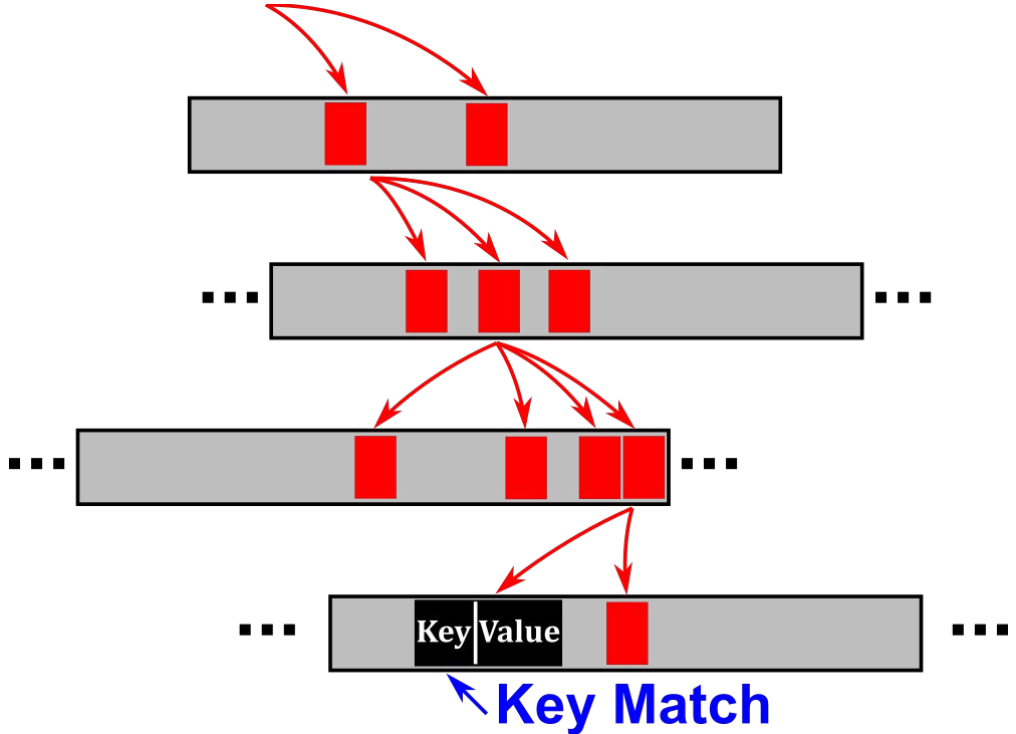
A Look at Index Traversal

- Index search in B^+-tree: binary search at each node



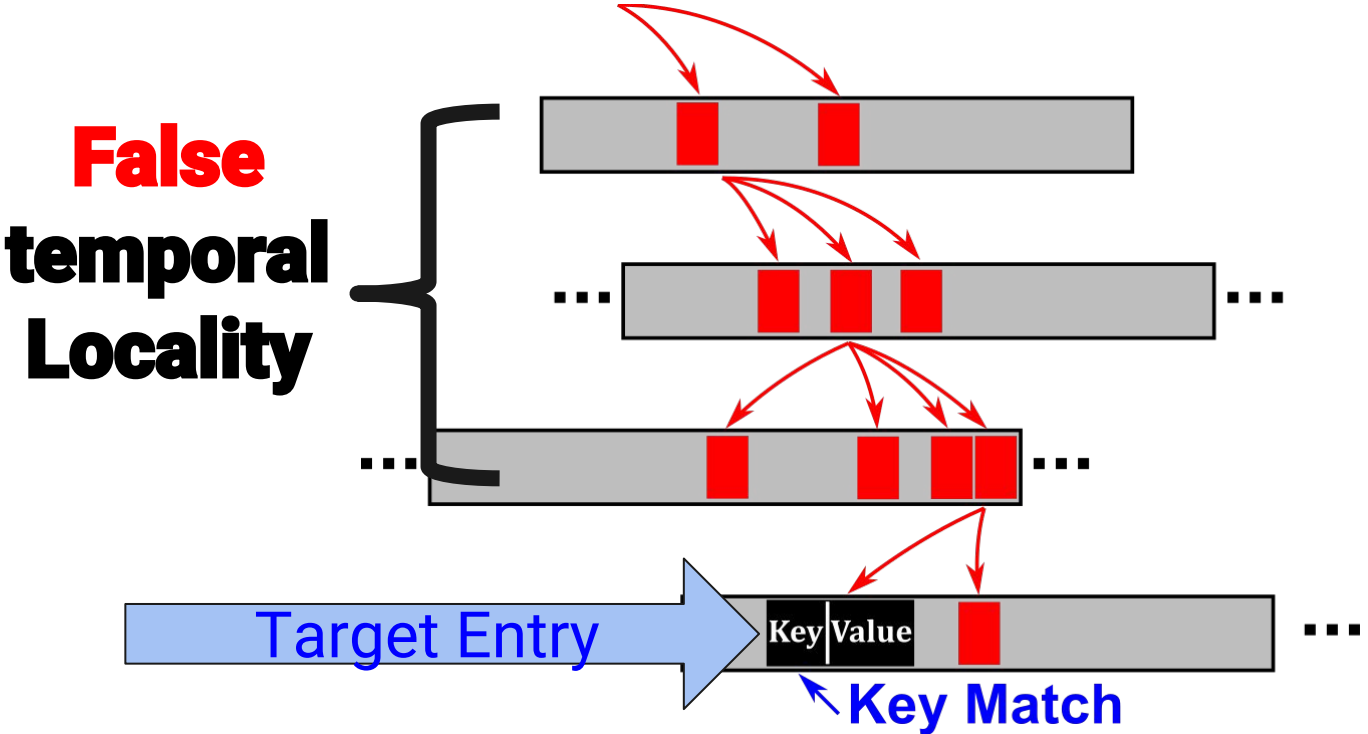
A Look at Index Traversal

- The intermediate entries on the path become **hot**.



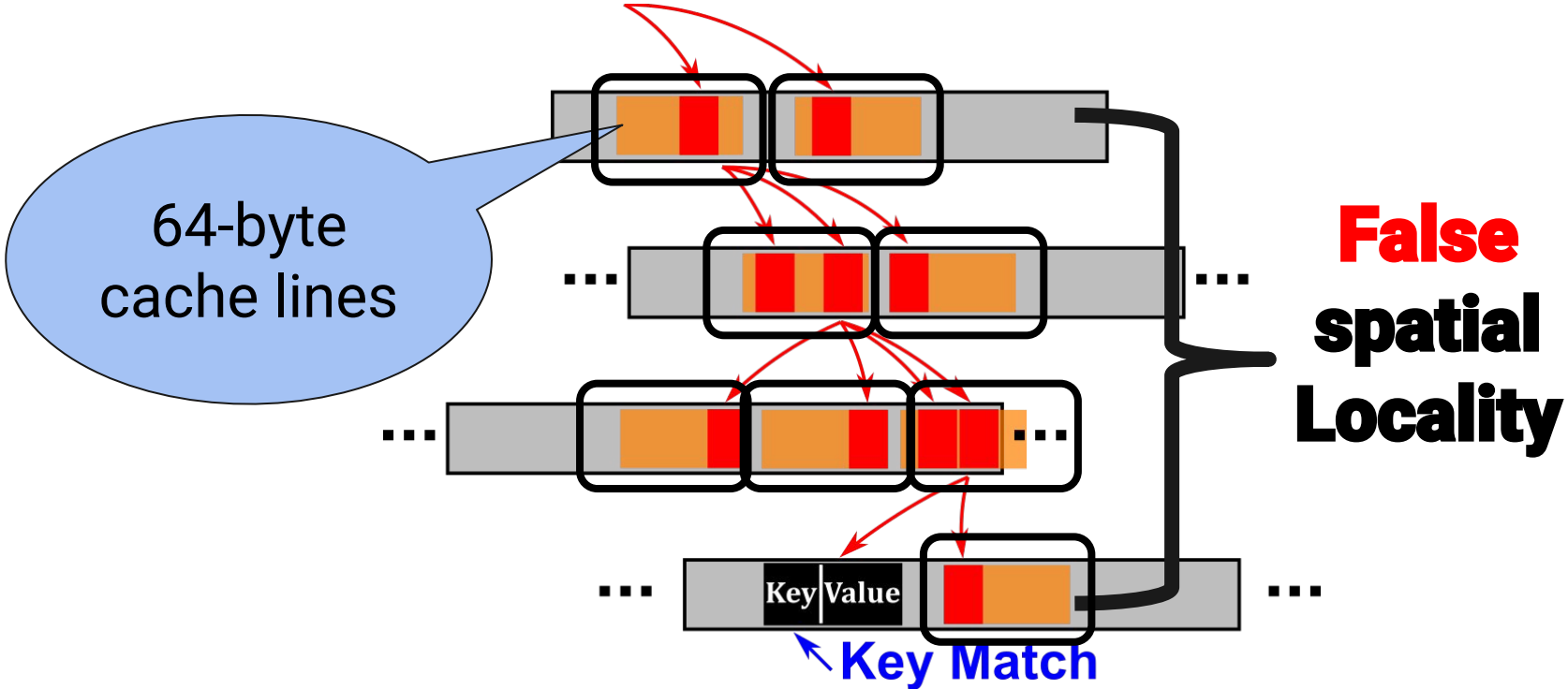
False Temporal Locality

- The intermediate entries on the path become **hot**.
- The purpose of index search is to find the **target entry**.



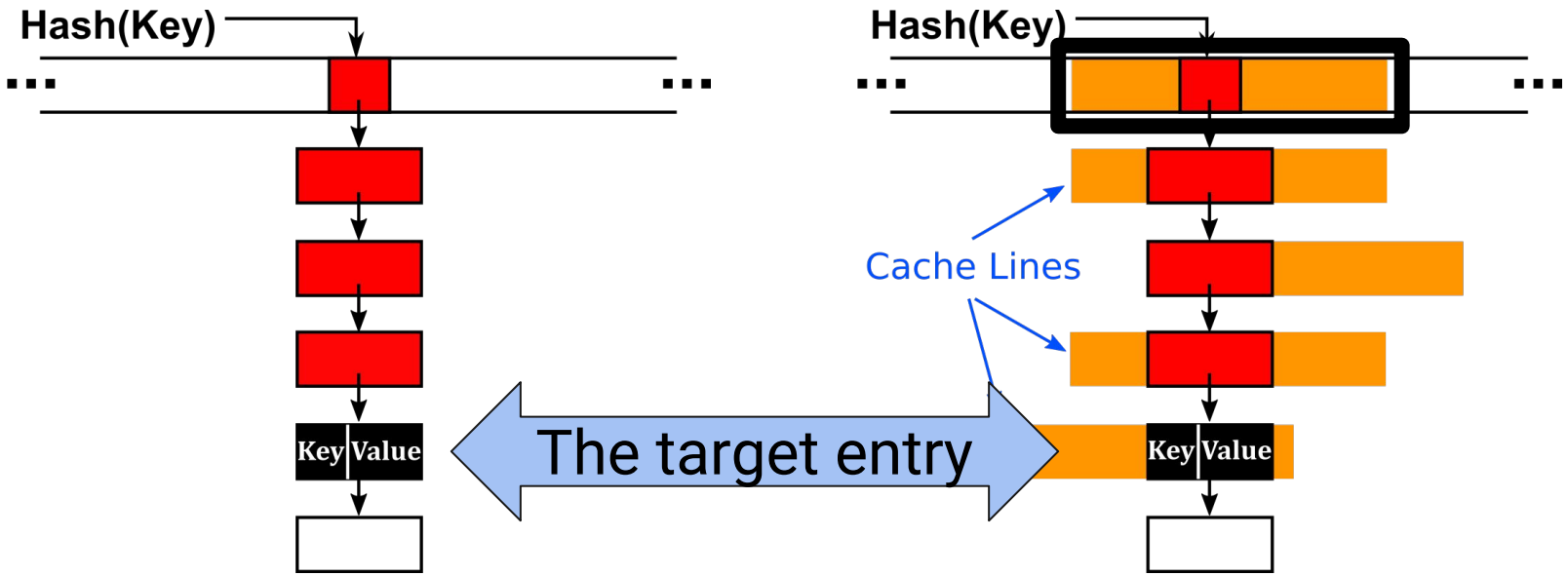
False Spatial Locality

- Each hot intermediate entry occupies a whole **cache line**.
- Touched cache lines ≫ entries required in the search.



False Localities on a Hash Table

- Chains or open addressing lead to false temporal locality.
- False spatial locality is significant even with short chains.



A Closer Look at Your CPU Cache

- Cache space is occupied by index entries of false localities.

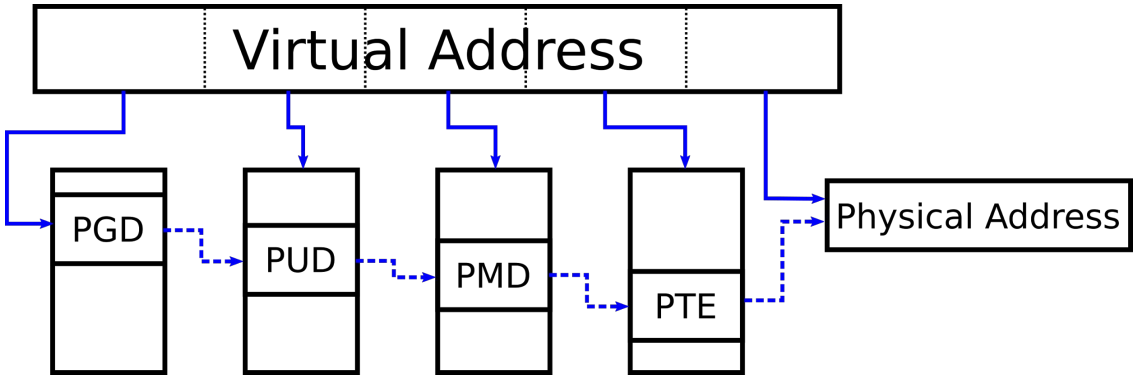
Existing Efforts on Improving Index Search

- Redesigning the data structure: Cuckoo hash, Masstree..
  - Must be an expert of the data structure
  - Optimizations are specific to certain data structures
  - May add overhead to other operations (e.g., expensive insertions)
- Hardware accelerators: Widx, MegaKV, etc.
  - High design cost
  - Hard to adapt to new index data structures
  - High latency for out-of-core accelerators (e.g., GPUs, FPGAs)

The Issue of Virtual Address Translation

Use of page tables shares the **same challenges** of index search.

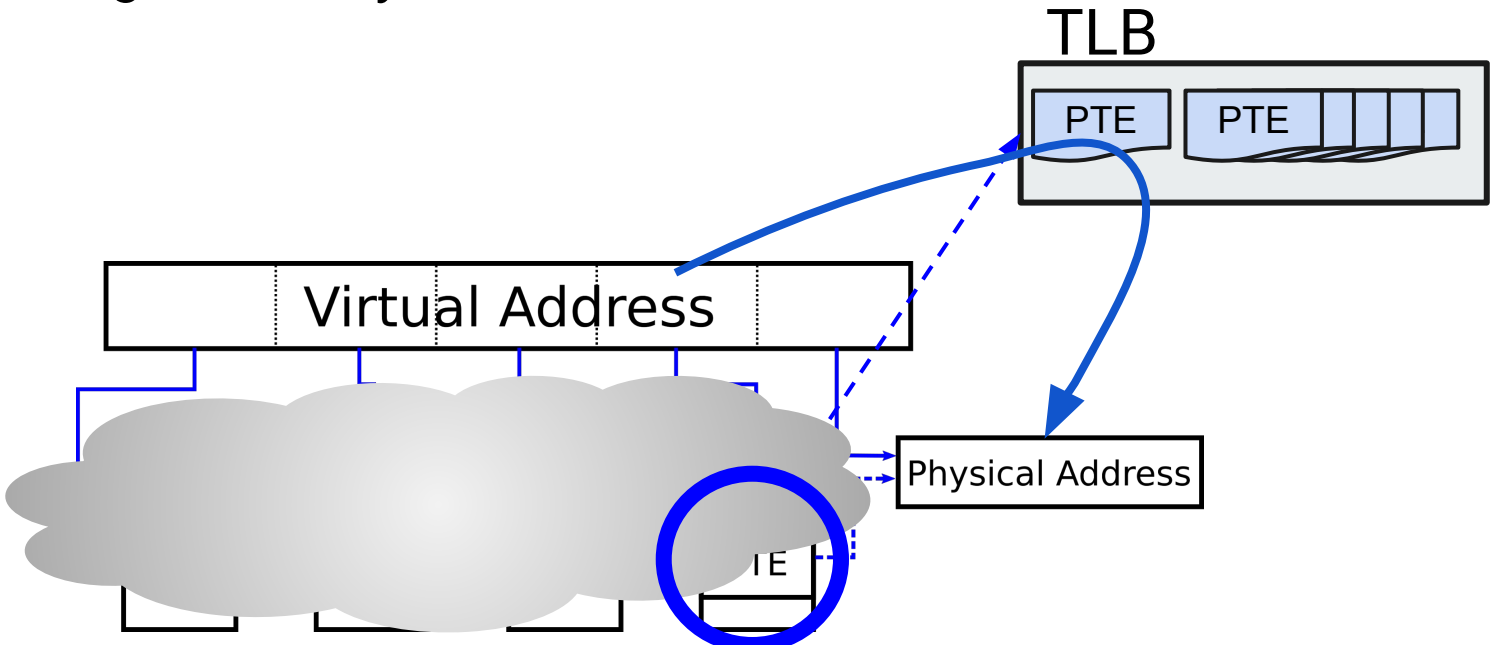
- Large index: every process has a page table.
- Frequently accessed: consulted in every memory access.
- False temporal locality: tree-structured tables.
- False spatial locality: intermediate page-table directories.



Fast Address translation with TLB

TLB directly caches **P**age **T**able **E**ntries for translation.

➔ Bypasses page table walking
➔ Covers large memory area with a small cache



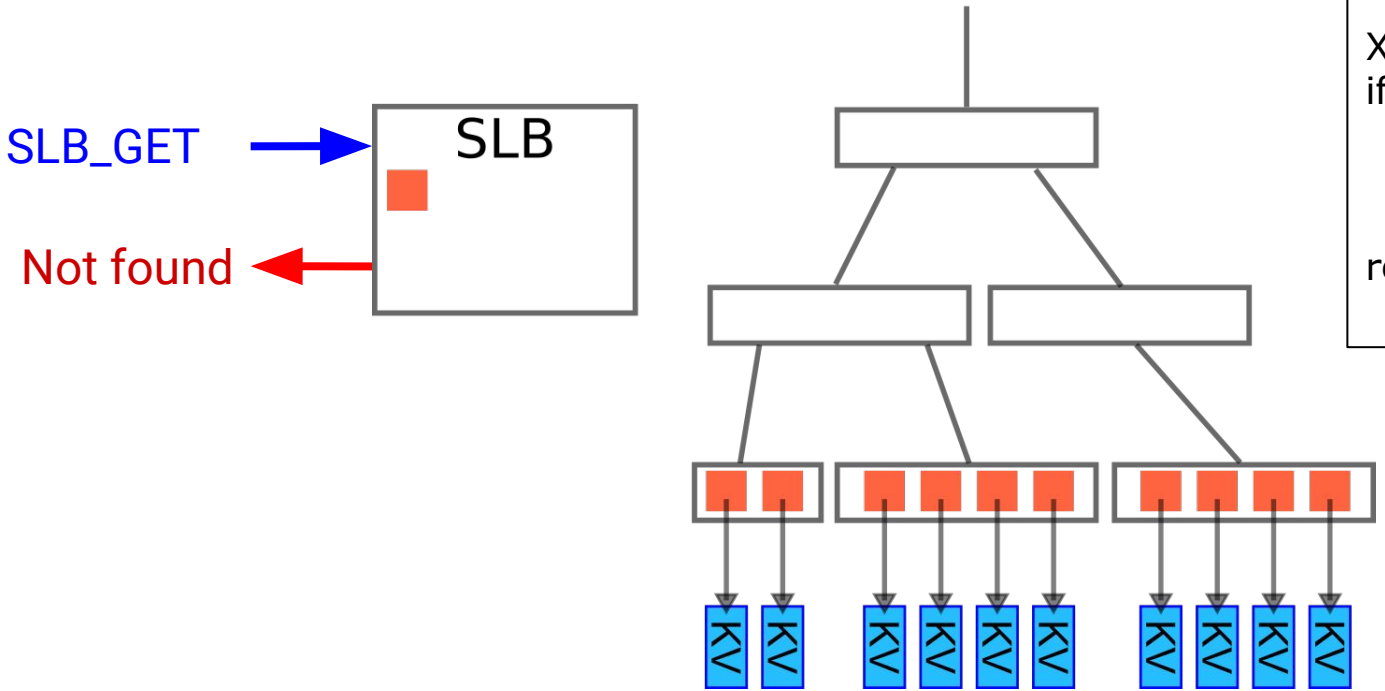
Our Solution: Search Lookaside Buffer

- Pure software library
- Easy integration with any index data structure
- Negligible overhead even in the worst case

Index Search with SLB

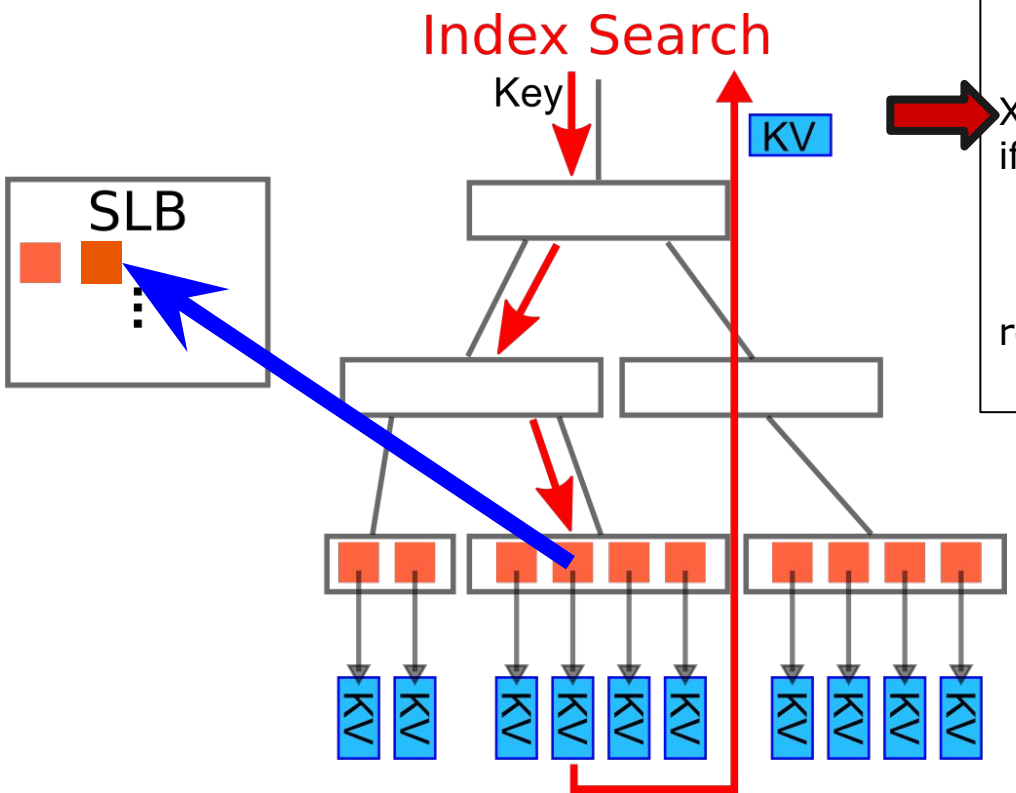
Every lookup first consults SLB.

```
X = SLB_GET(key)
if X:
  return X

X = INDEX_GET(key)
if X:
  SLB_EMIT(key, X)
  return X

return NULL
```



Index Search with SLB

Emits a target entry after successful search.

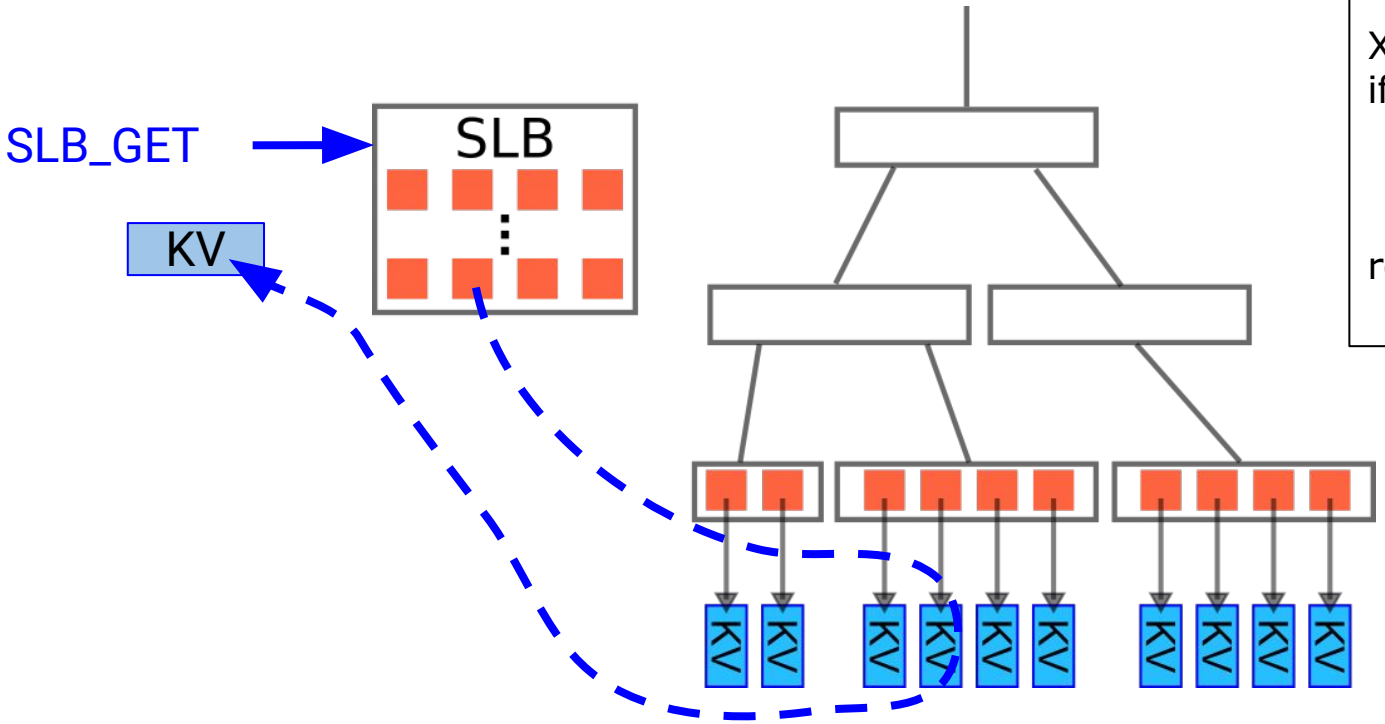



```
X = SLB_GET(key)
if X:
  return X

X = INDEX_GET(key)
if X:
  SLB_EMIT(key, X)
  return X

return NULL
```

Index Search with SLB

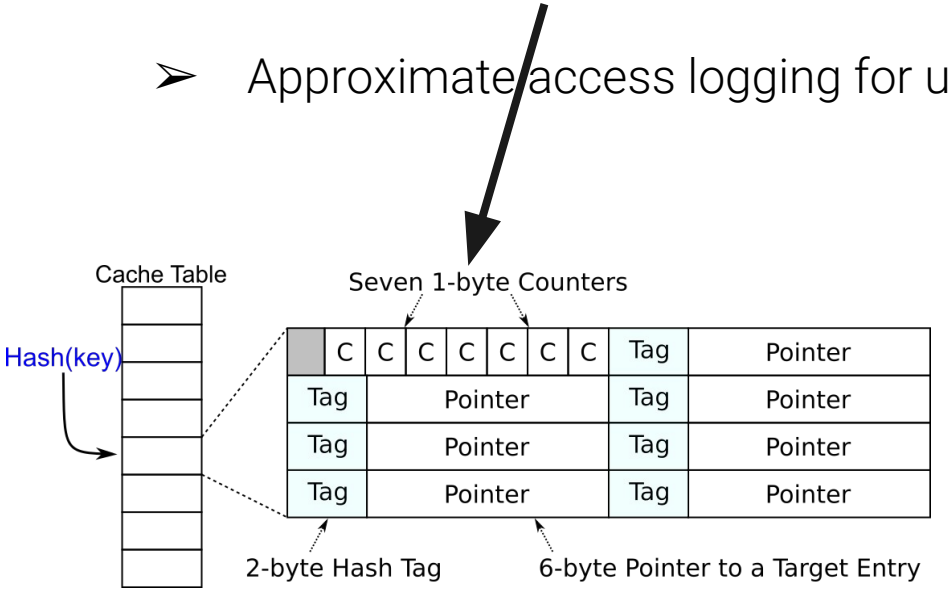
A hit in SLB cache completes the search.

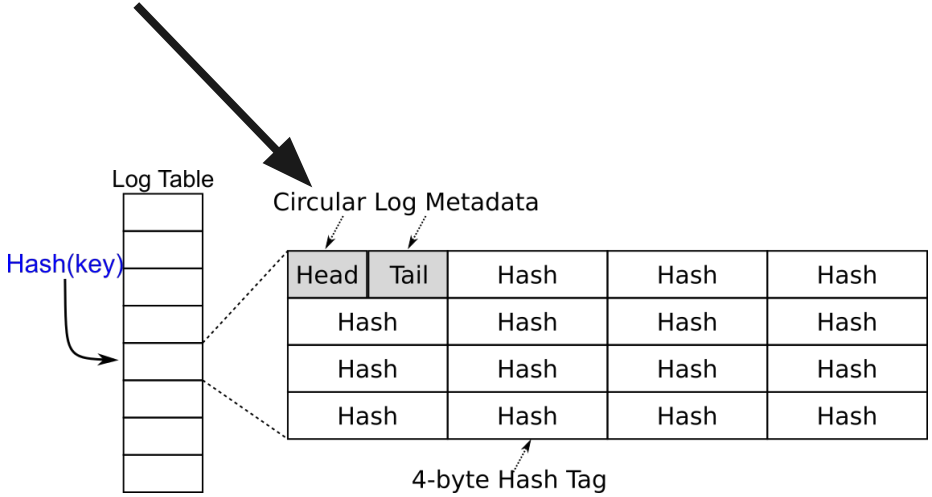
```
X = SLB_GET(key)
if X:
  return X

X = INDEX_GET(key)
if X:
  SLB_EMIT(key, X)
  return X

return NULL
```



Design challenges

- Tracking KV temperatures can pollute CPU cache
  - Cache-line-local access counters for cached items.
  - Approximate access logging for uncached items.



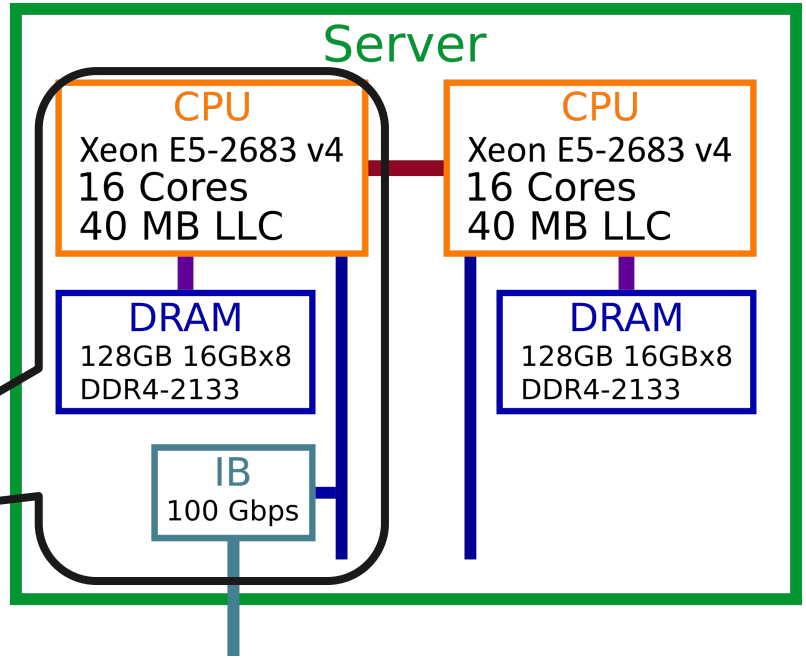


Design challenges

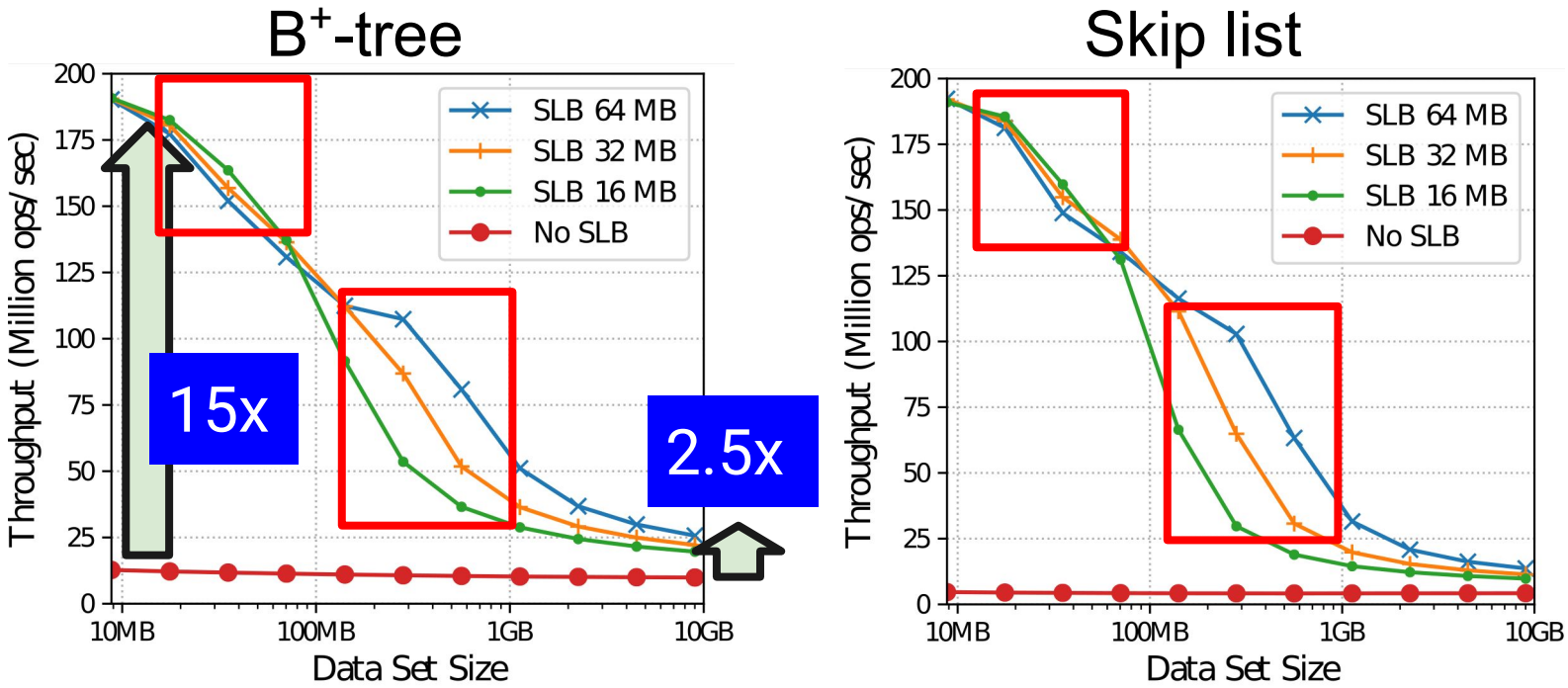
- Tracking temperatures of items can pollute CPU cache
  - Cache-line-local access counters for cached items.
  - Approximate access logging for uncached items.
- Frequent replacement hurts index performance
  - Adaptive logging throttling for uncached items.
- More details in the paper...

Experimental Setup

- B^{+}-tree, Skip list, and hash tables
- Filled with 10^{8} KVs (8B K, 64B V)
- **Store size: ~10GB**
- Zipfian workload
- **Accessed data set: 10MB->10GB**
- SLB size: 16/32/64 MB
- Uses one NUMA node (16 cores)



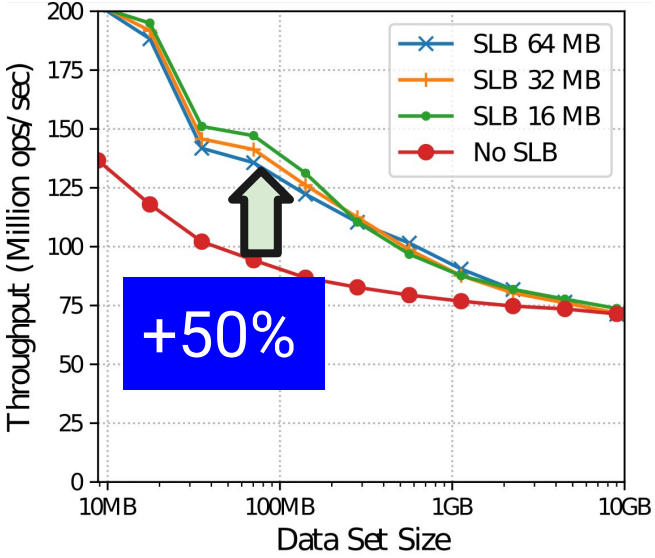
B+-tree and Skip List



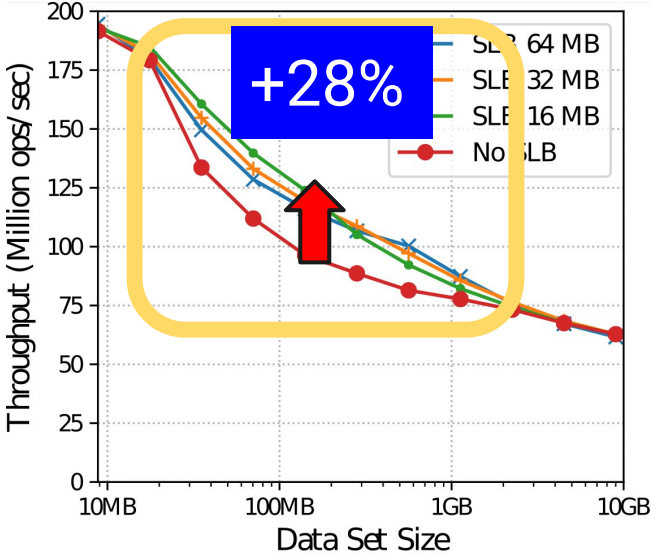
- Significant improvements for ordered data structures
  - Substantial False localities caused by index traversal

Hash Tables

Cuckoo



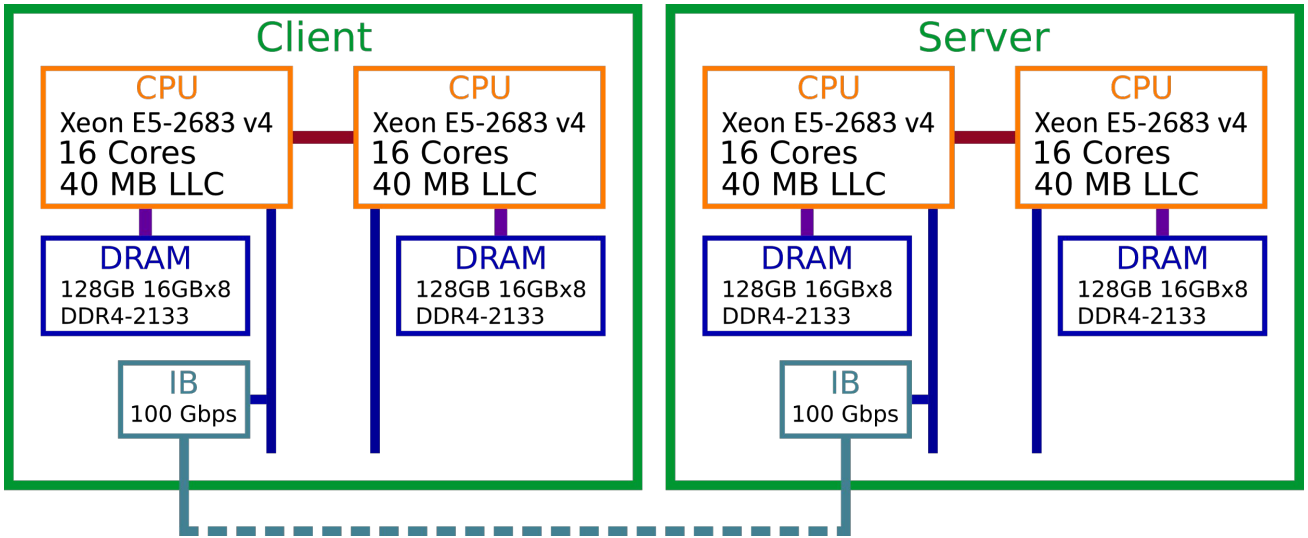
Chaining



- Chaining hash table: average chain length <= 1
  - The index has no false temporal locality.
  - improves by up to 28% by removing false spatial locality

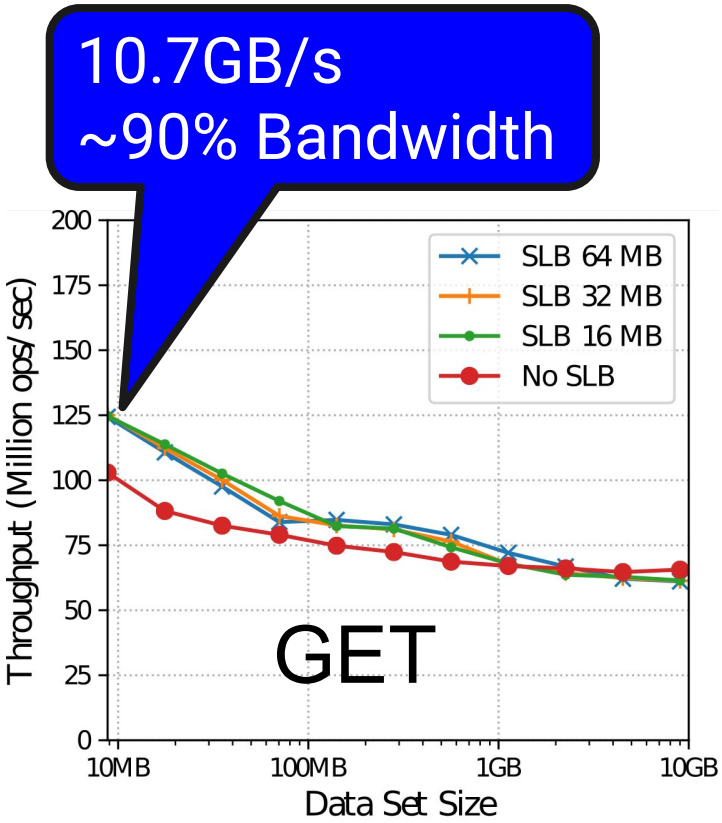
High-performance KV Server

- An RDMA-port of MICA [Lim et al., NSDI'14]
  - In-memory KV store
  - Bulk-chaining partitioned hash tables
  - Batch-processing
  - Lock-free accesses

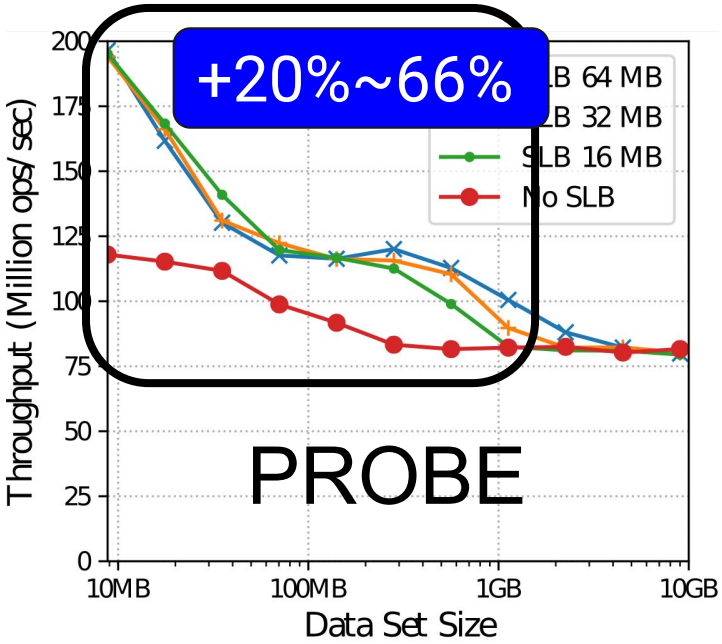


MICA over 100Gbps Infiniband

- GET: Limited improvements due to network bandwidth.



- PROBE: only returns True/False



Conclusion

- We identify the issue of **false temporal/spatial locality** in index search.
- We propose SLB, a general software solution to improve search for **any index data structure** by removing the false localities.
- SLB improves index search for workloads with strong locality, and imposes **negligible overhead** with weak locality.

Thank You !

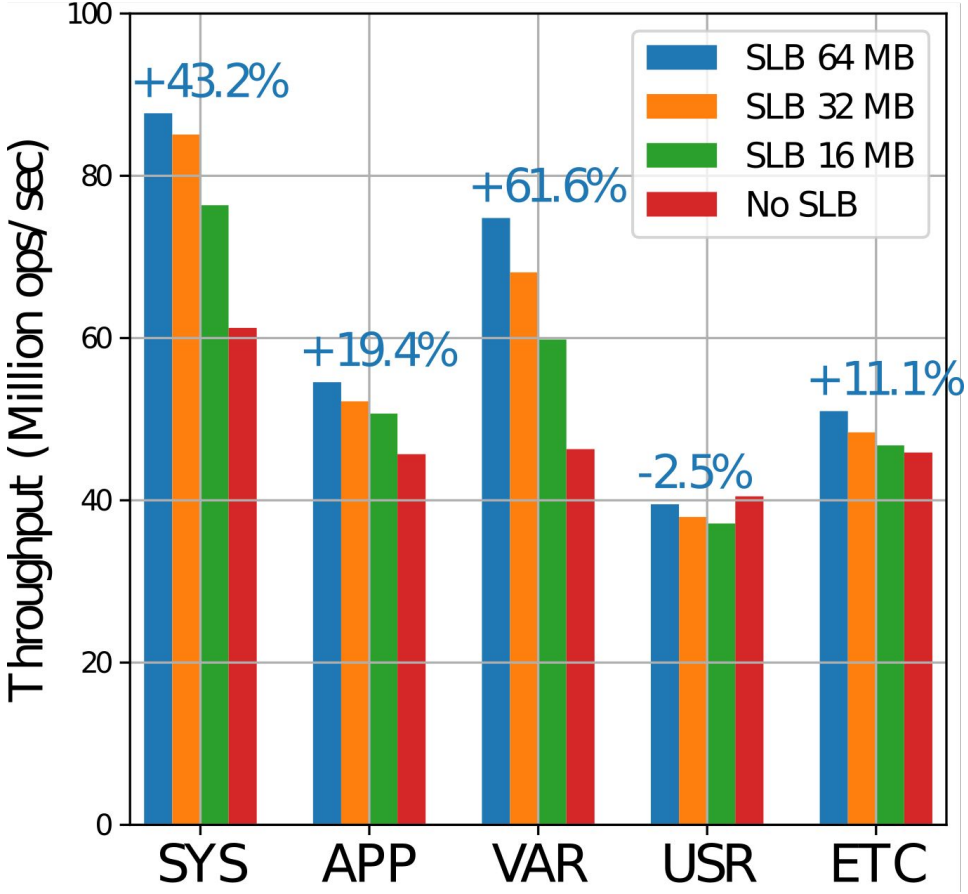
☺ Questions?

Backup slides

Replaying Facebook KV Workloads

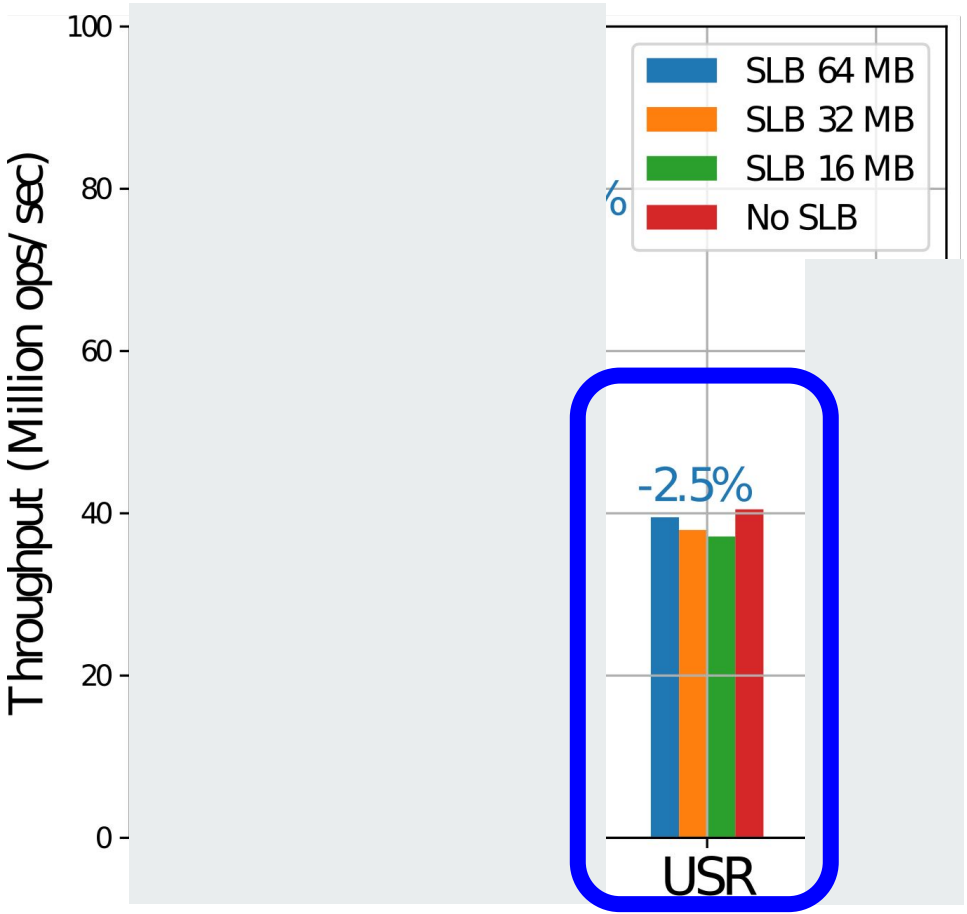
Five key-value traces collected on production memcached servers

[Atikoglu et al., Sigmetrics'12]



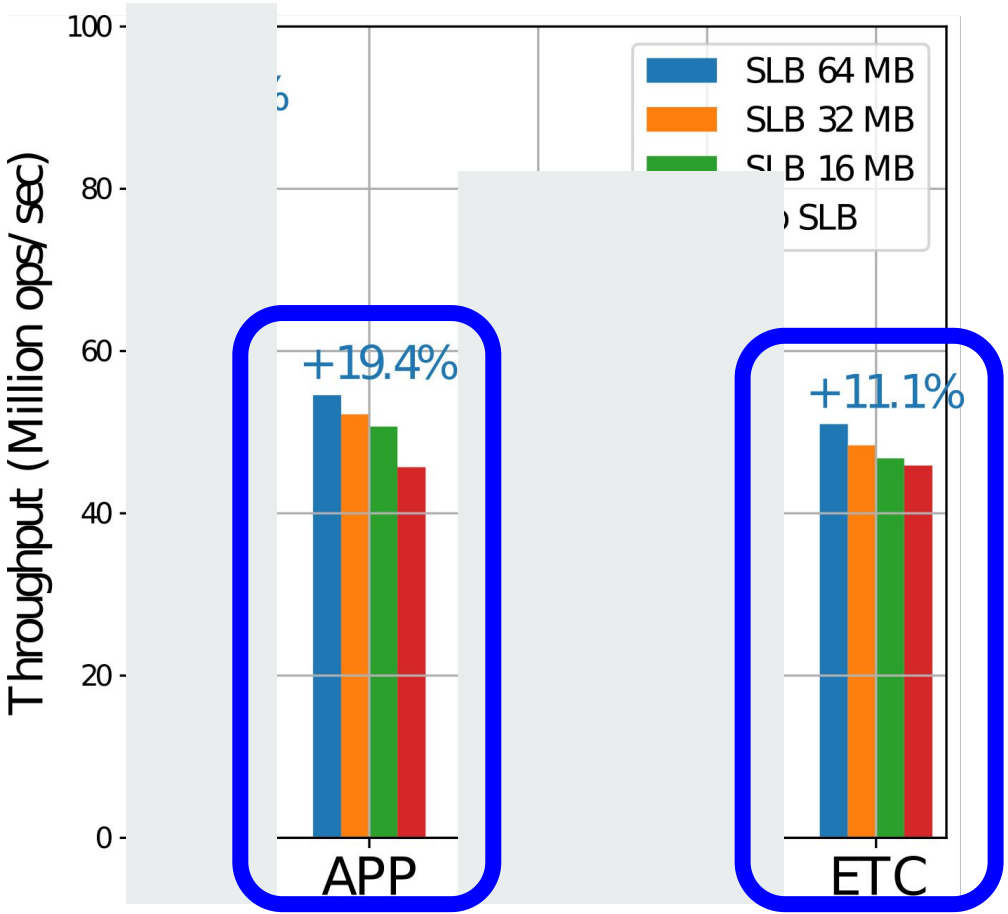
Replaying Facebook KV Workloads

USR:
GET-dominant
Less skewed
Working set >>> cache
No improvement



Replaying Facebook KV Workloads

APP & ETC:
More skewed
Working set fits the cache
10%-30% DELETE
frequent invalidations in SLB
Improvement < 20%



Replaying Facebook KV Workloads

SYS & VAR:
GET & UPDATE
Working set fits the cache
Improvement > 43%