MICA: A Holistic Approach to Fast In-Memory Key-Value Storage

Hyeontaek Lim[1]

Dongsu Han,[2] David G. Andersen,[1] Michael Kaminsky[3]

[1]Carnegie Mellon University

[2]KAIST, [3]Intel Labs

Goal: Fast In-Memory Key-Value Store

- Improve per-node performance (op/sec/node)
  - Less expensive
  - Easier hotspot mitigation
  - Lower latency for multi-key queries

- Target: small key-value items (fit in single packet)

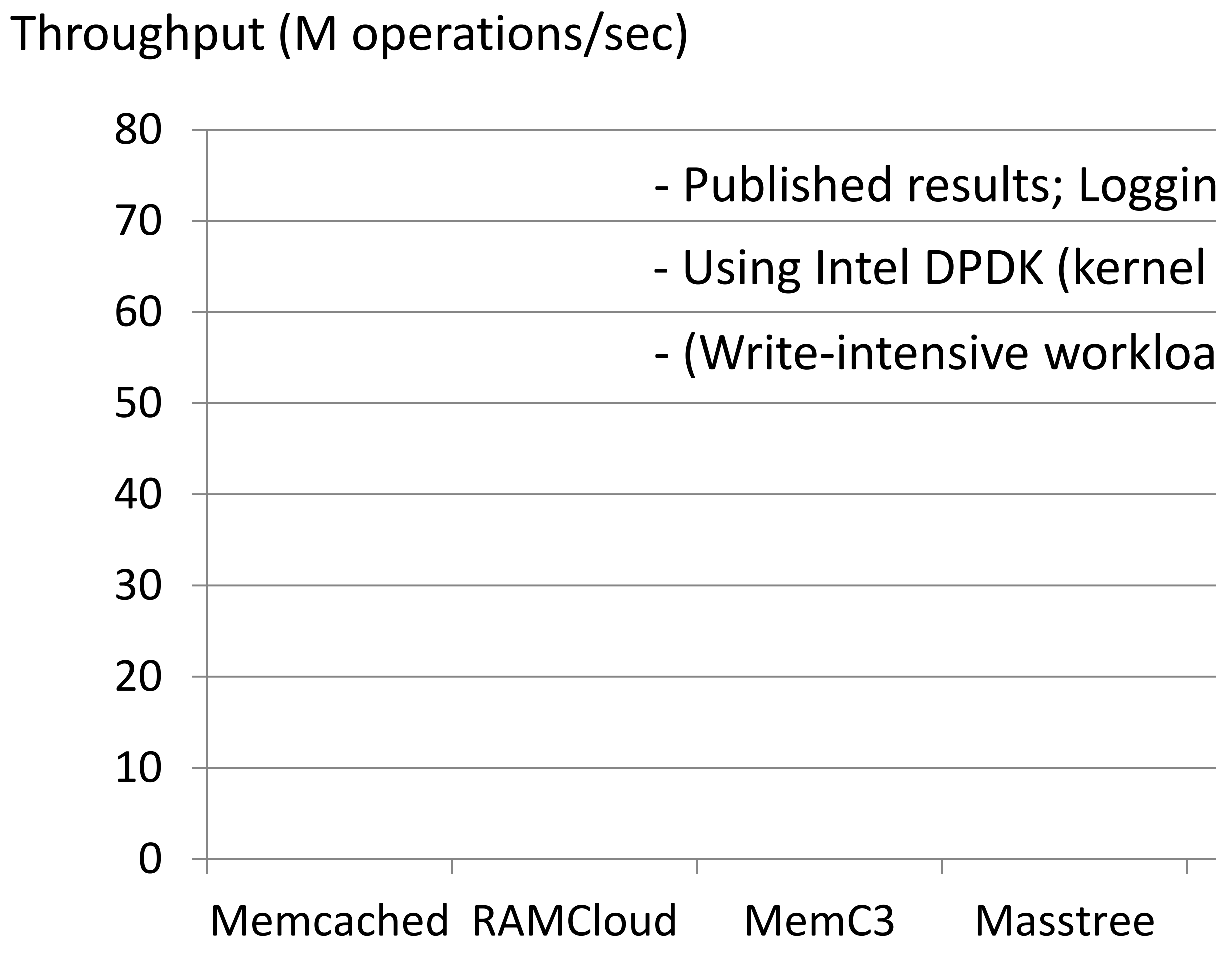
- Non-goals: cluster architecture, durability

Q: How Good (or Bad) are Current Systems?

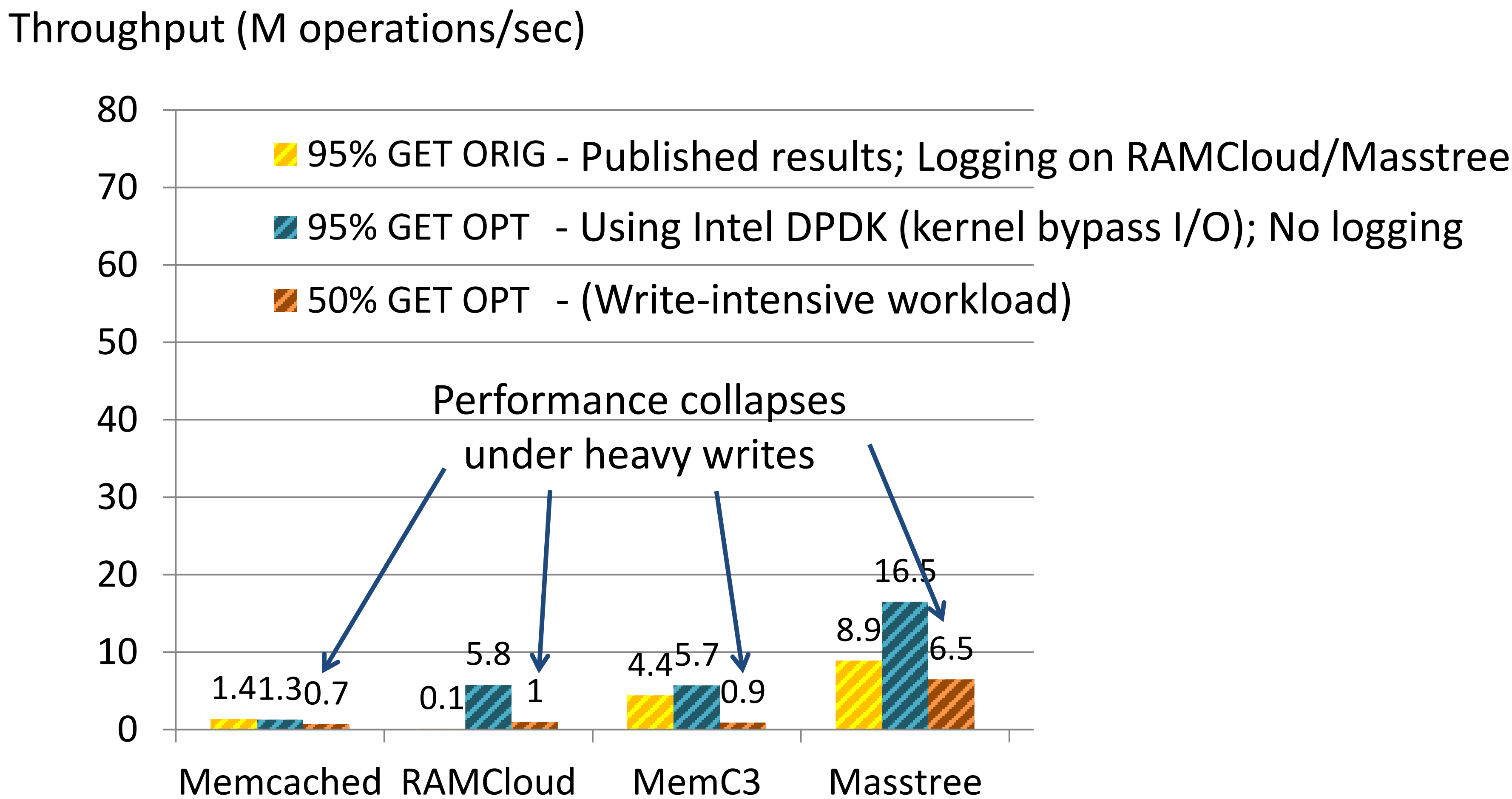
- Workload: YCSB [SoCC 2010]
  - Single-key operations
- In-memory storage
  - Logging turned off in our experiments
- End-to-end performance over the network
- Single server node

End-to-End Performance Comparison

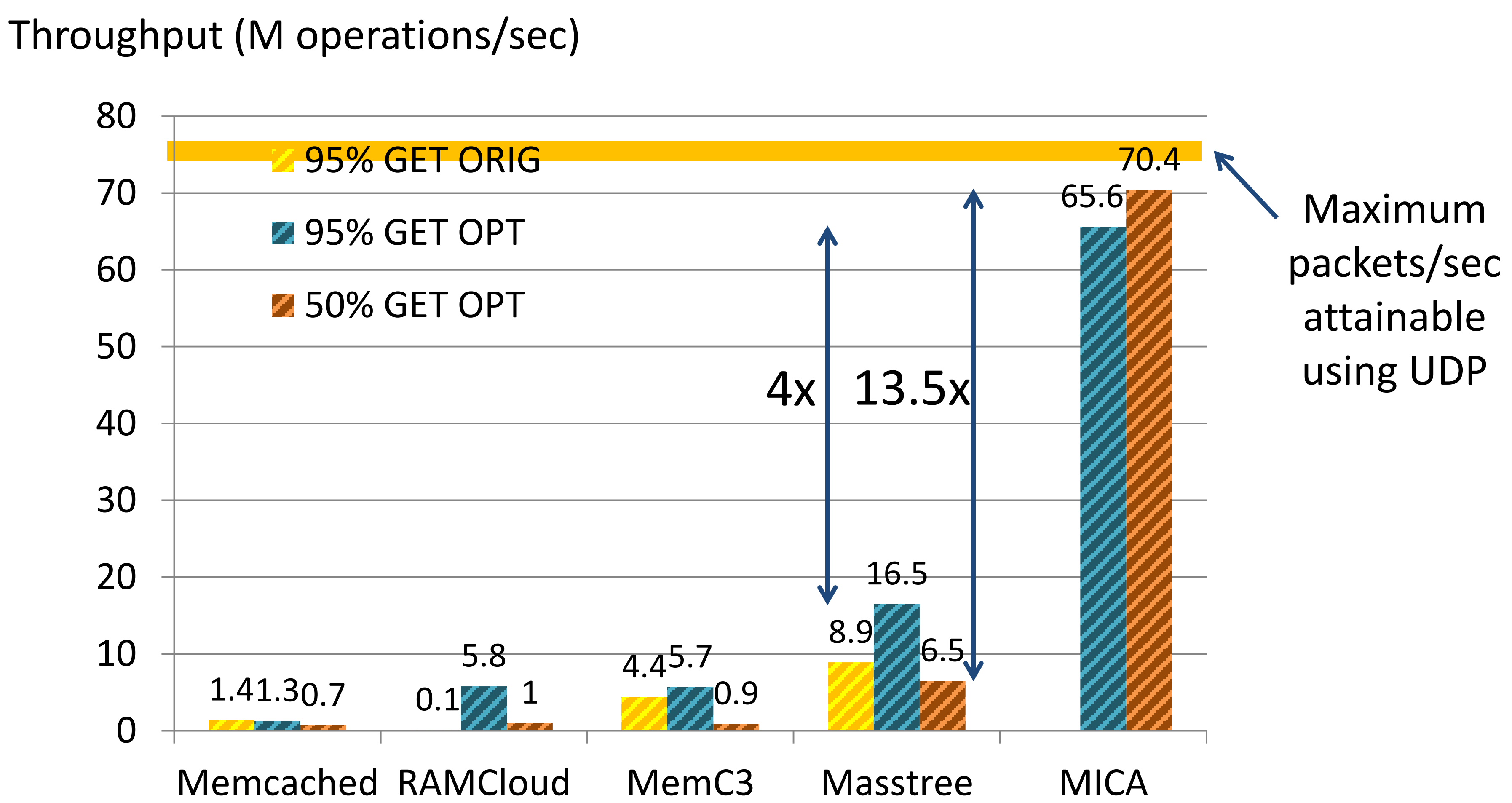
- Published results; Logging on RAMCloud/Masstree
- Using Intel DPDK (kernel bypass I/O); No logging
- (Write-intensive workload)



End-to-End Performance Comparison

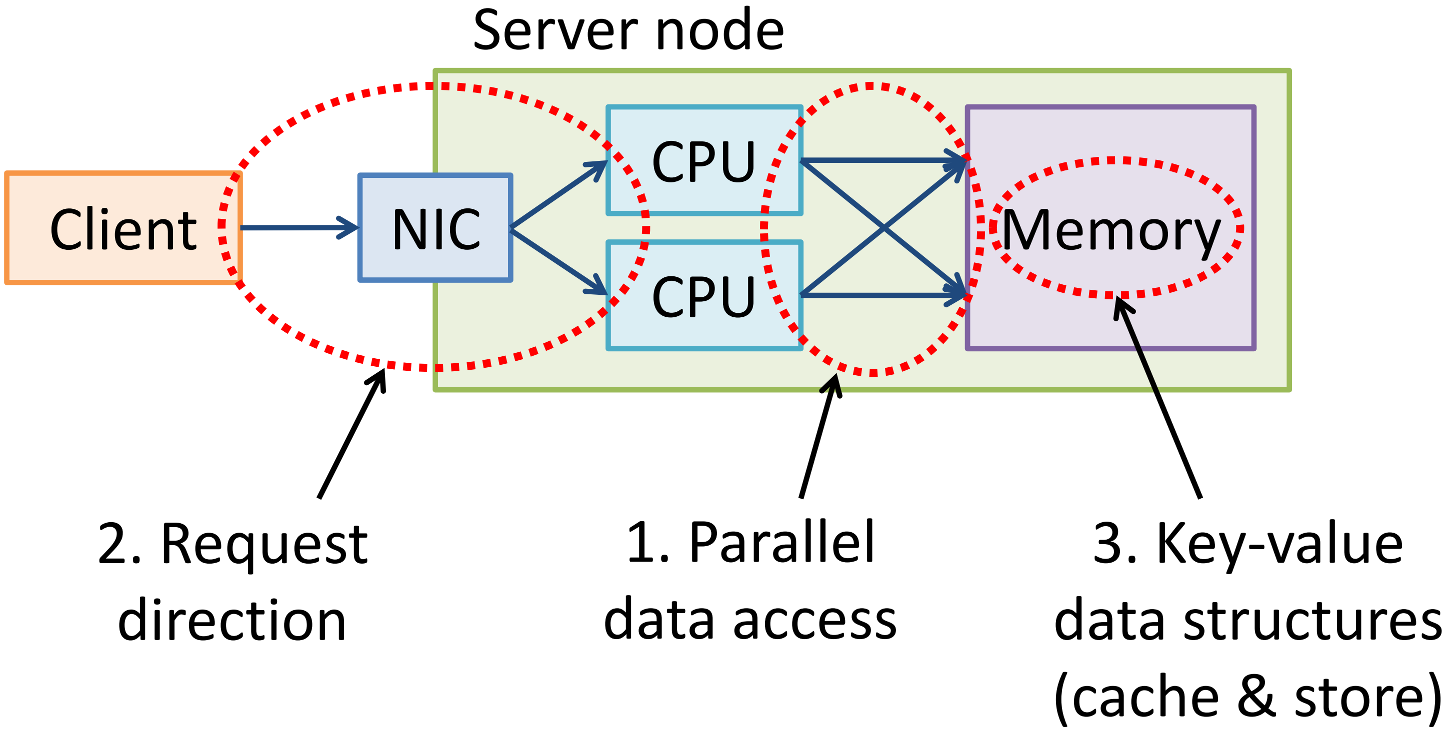


End-to-End Performance Comparison

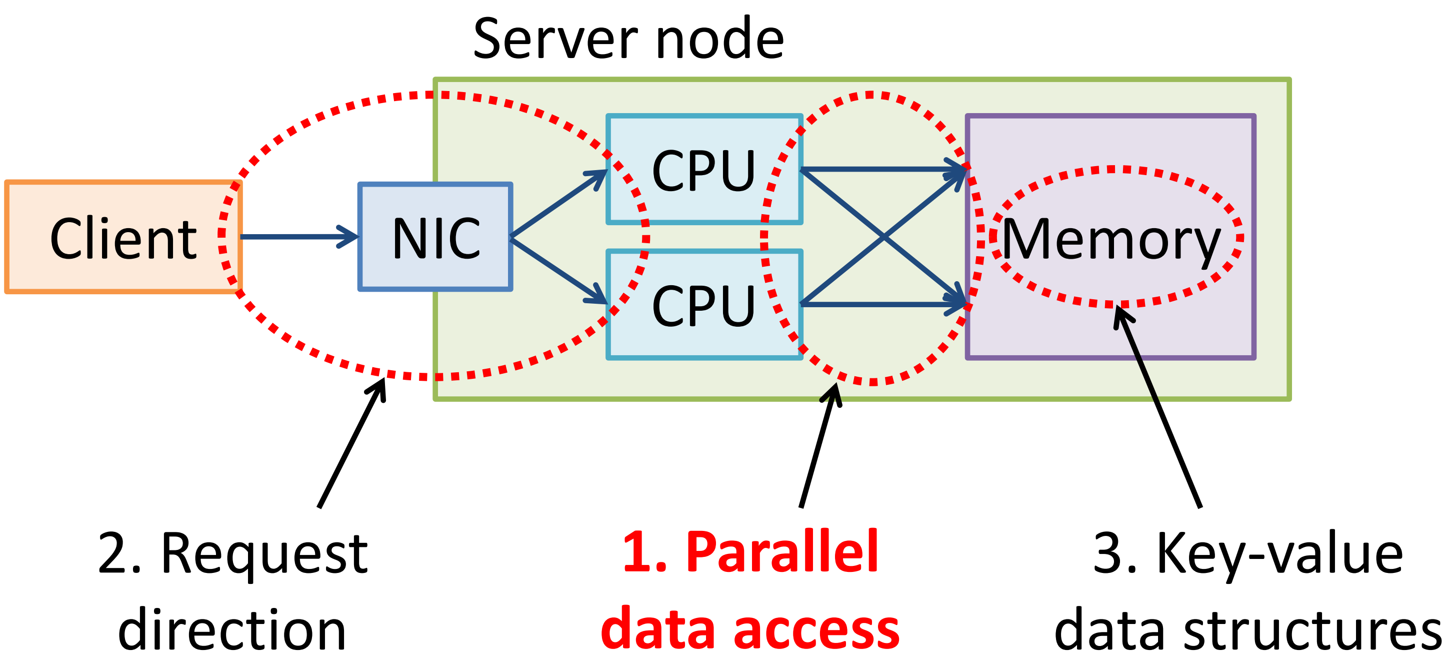


MICA Approach

- **MICA**: Redesigning in-memory key-value storage
  - Applies new SW architecture and data structures to general-purpose HW in a holistic way



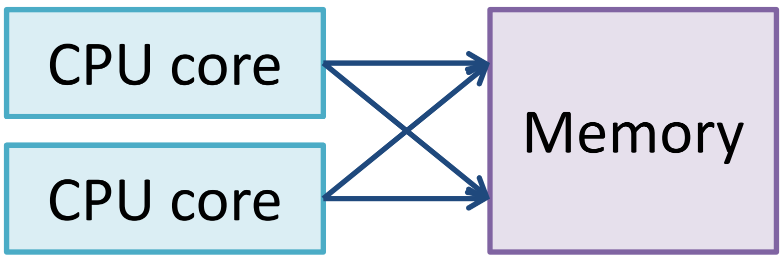
Parallel Data Access



- Modern CPUs have many cores (8, 15, ...)
- How to exploit CPU parallelism efficiently?

Parallel Data Access Schemes

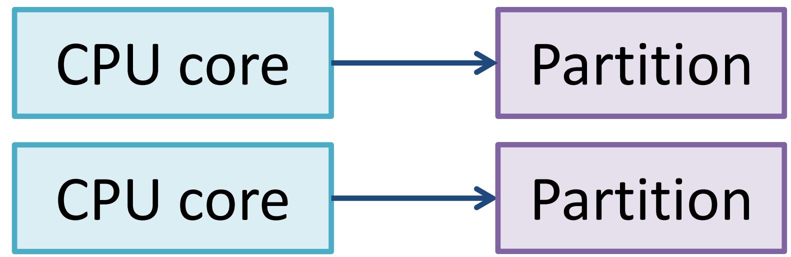
Concurrent Read
Concurrent Write



+ Good load distribution

- Limited CPU scalability (e.g., synchronization)
- Cross-NUMA latency

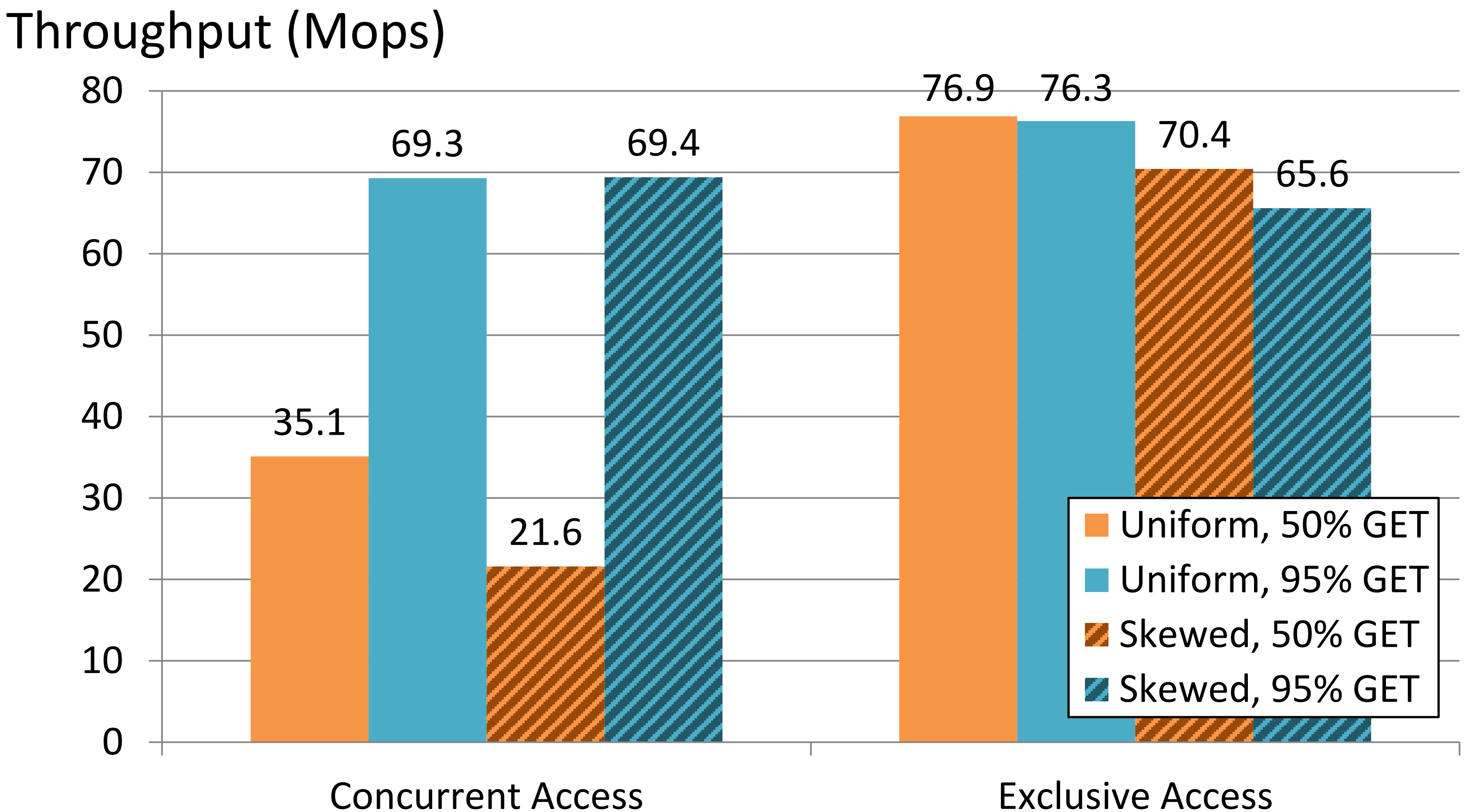
Exclusive Read
Exclusive Write



+ Good CPU scalability

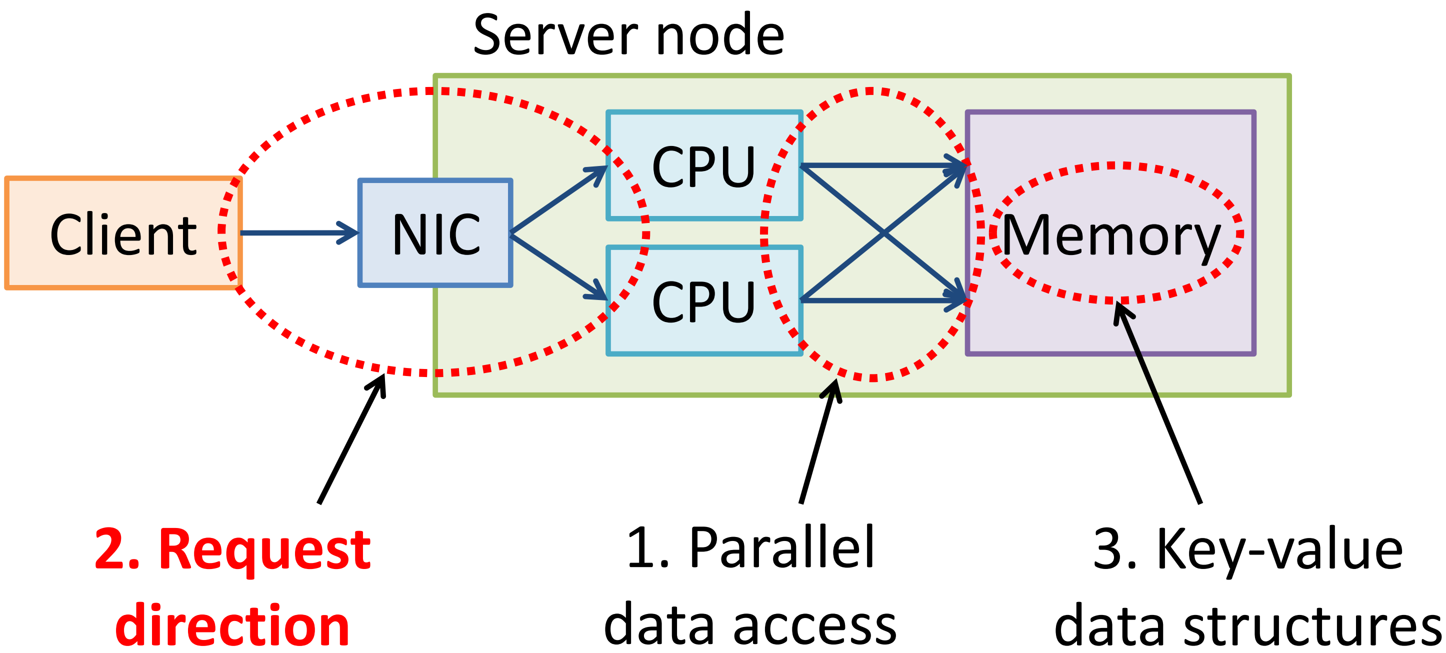
- *Potentially* low performance under skewed workloads

In MICA, Exclusive Outperforms Concurrent



End-to-end performance with kernel bypass I/O

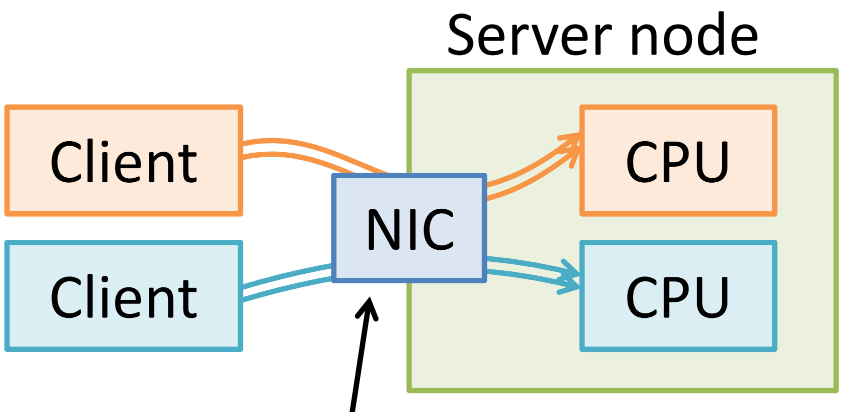
Request Direction



- Sending requests to appropriate CPU cores for better data access locality
- Exclusive access benefits from correct delivery
  - Each request must be sent to corresp. partition's core

Request Direction Schemes

Flow-based Affinity

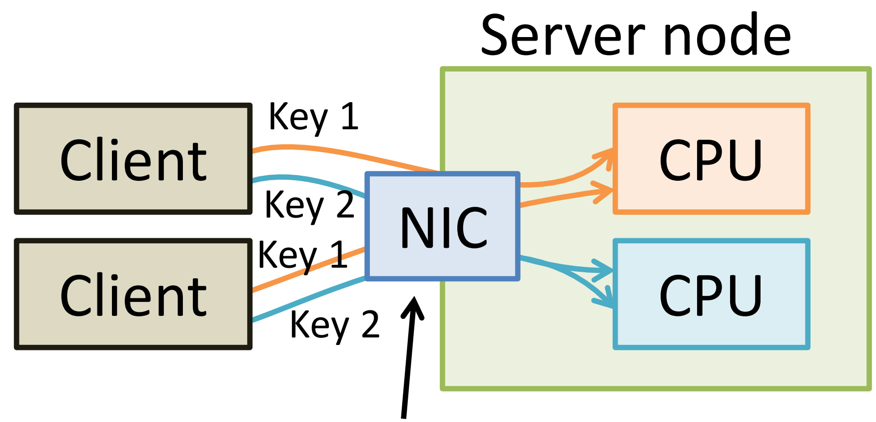


Classification using 5-tuple

+ Good locality for flows (e.g., HTTP over TCP)

- Suboptimal for small key-value processing

Object-based Affinity

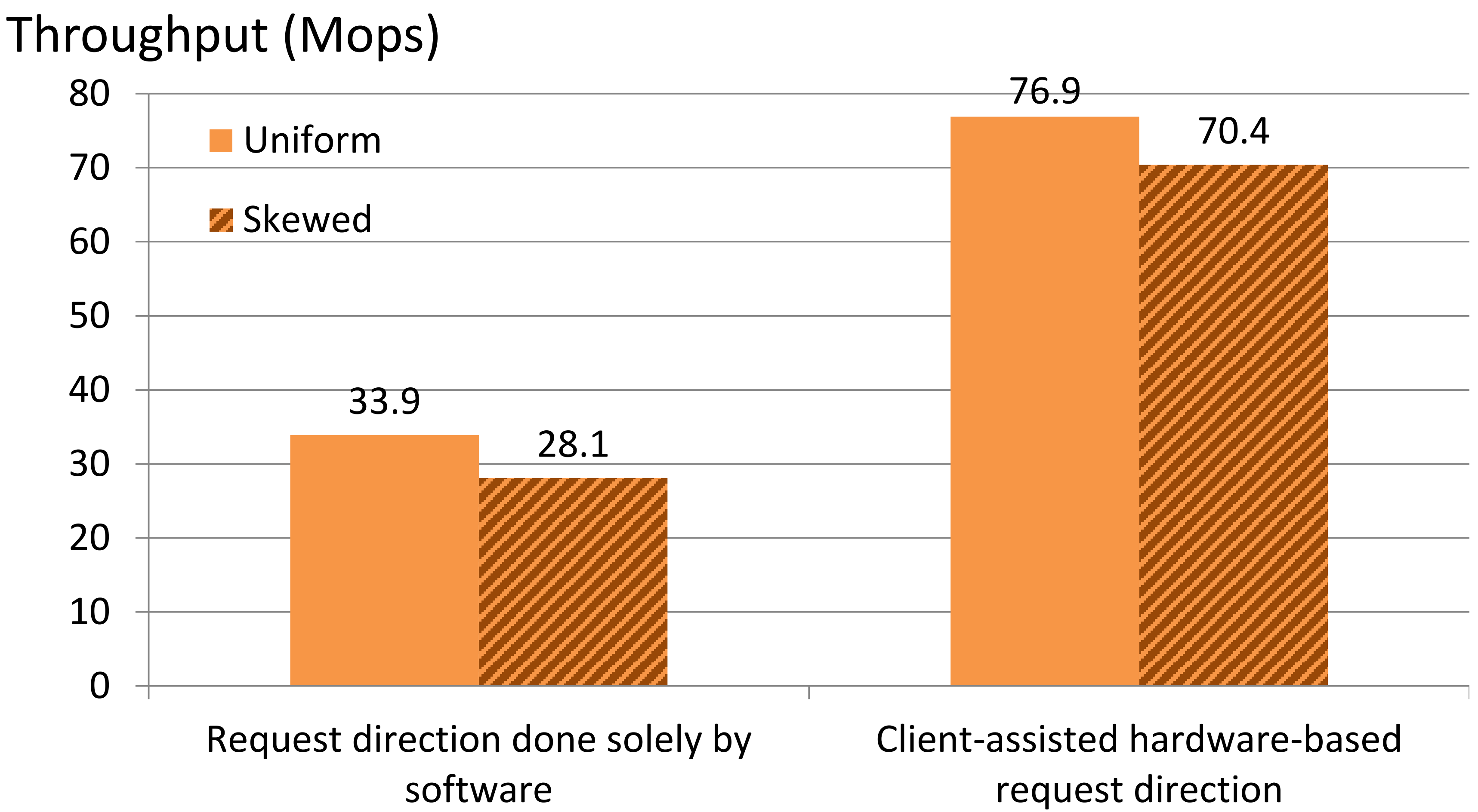


Classification depends on request content

+ Good locality for key access

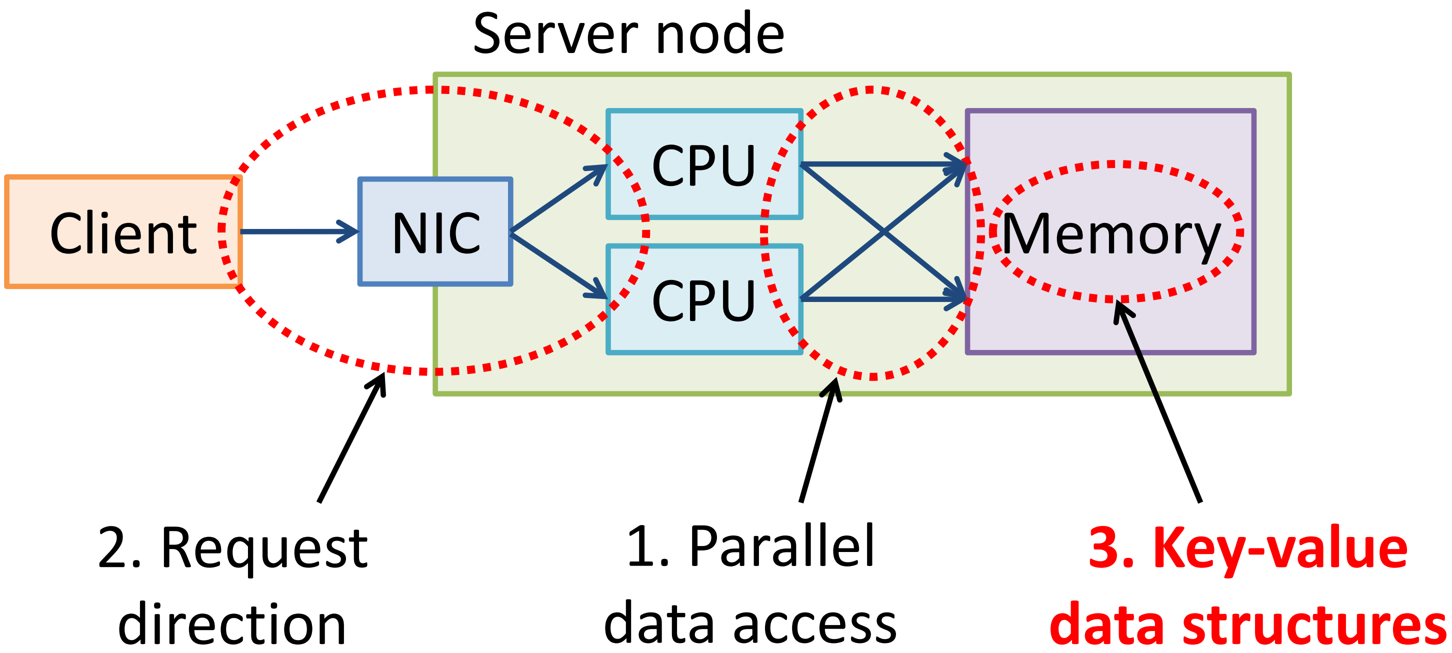
- Client assist or special HW support needed for efficiency

Crucial to Use NIC HW for Request Direction



Using exclusive access for parallel data access

Key-Value Data Structures



- Significant impact on key-value processing speed
- New design required for very high op/sec for both read and write
- "Cache" and "store" modes

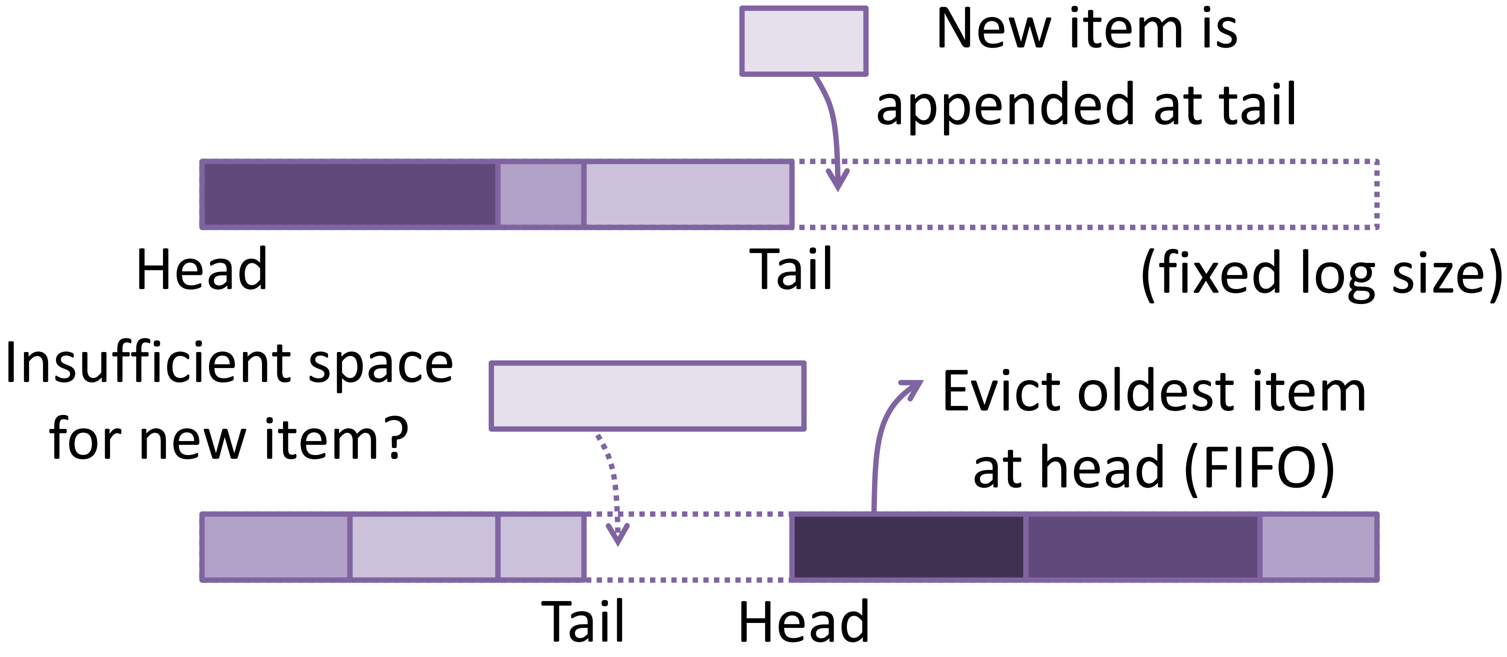
MICA's "Cache" Data Structures

- Each partition has:
  - **Circular log** (for memory allocation)
  - **Lossy concurrent hash index** (for fast item access)

- Exploit Memcached-like cache semantics
  - Lost data is easily recoverable (not free, though)
  - Favor fast processing
- Provide good memory efficiency & item eviction

Circular Log

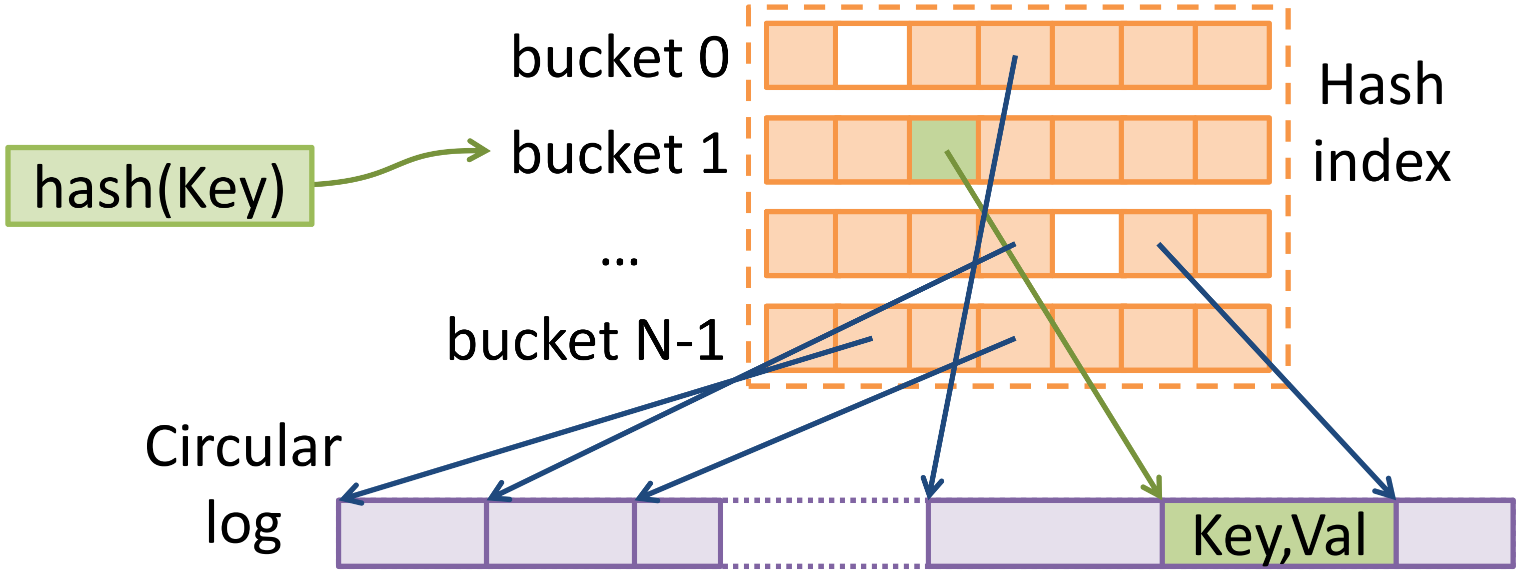
- Allocates space for key-value items of any length
- Conventional logs + Circular queues
  - Simple garbage collection/free space defragmentation



Support LRU by reinserting recently accessed items

Lossy Concurrent Hash Index

- Indexes key-value items stored in the circular log
- Set-associative table



- Full bucket? Evict oldest entry from it
  - Fast indexing of new key-value items

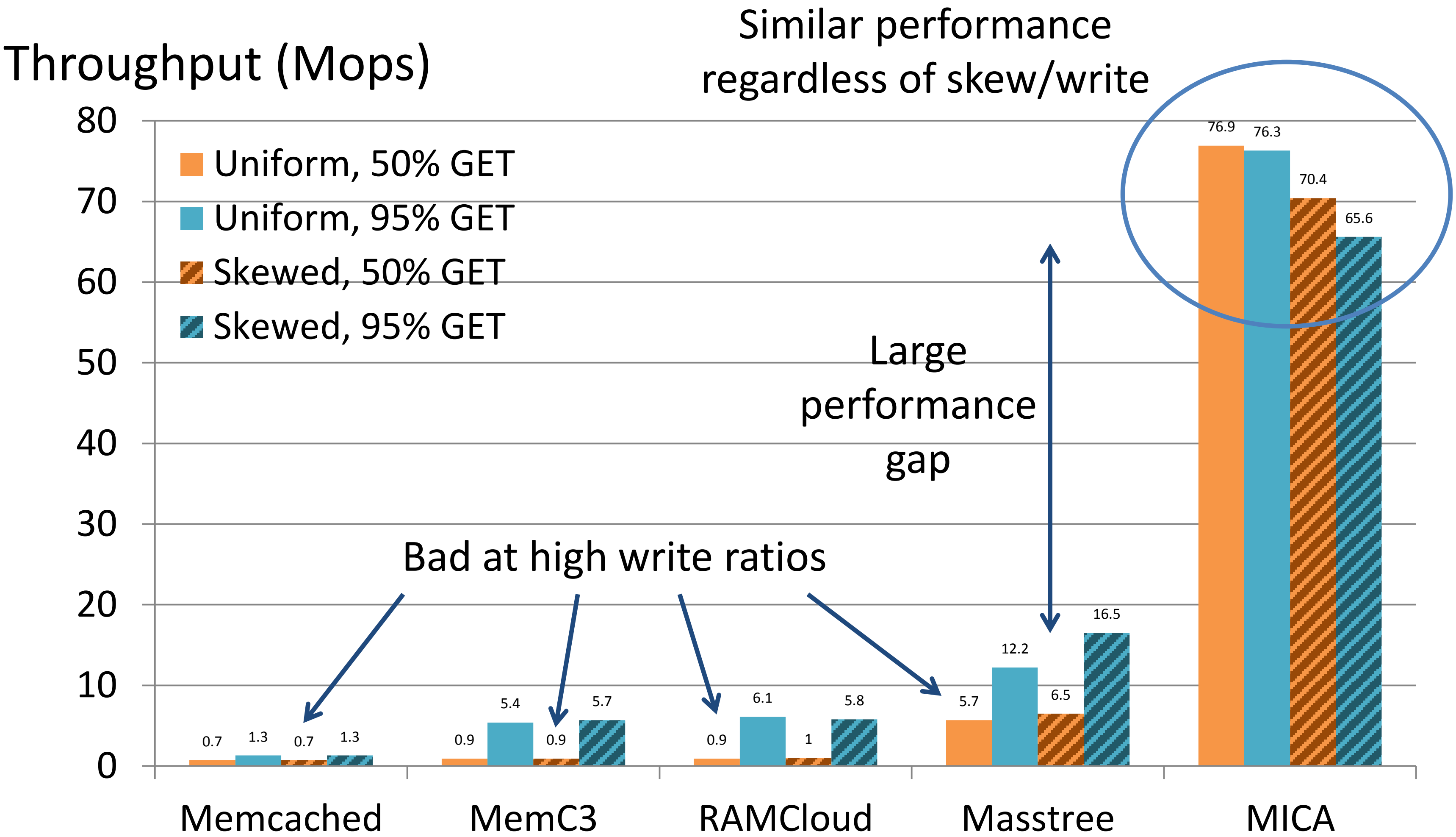
MICA's "Store" Data Structures

- Required to preserve stored items
- Achieve similar performance by trading memory
  - Circular log -> Segregated fits
  - Lossy index -> Lossless index (with bulk chaining)
- See our paper for details

Evaluation

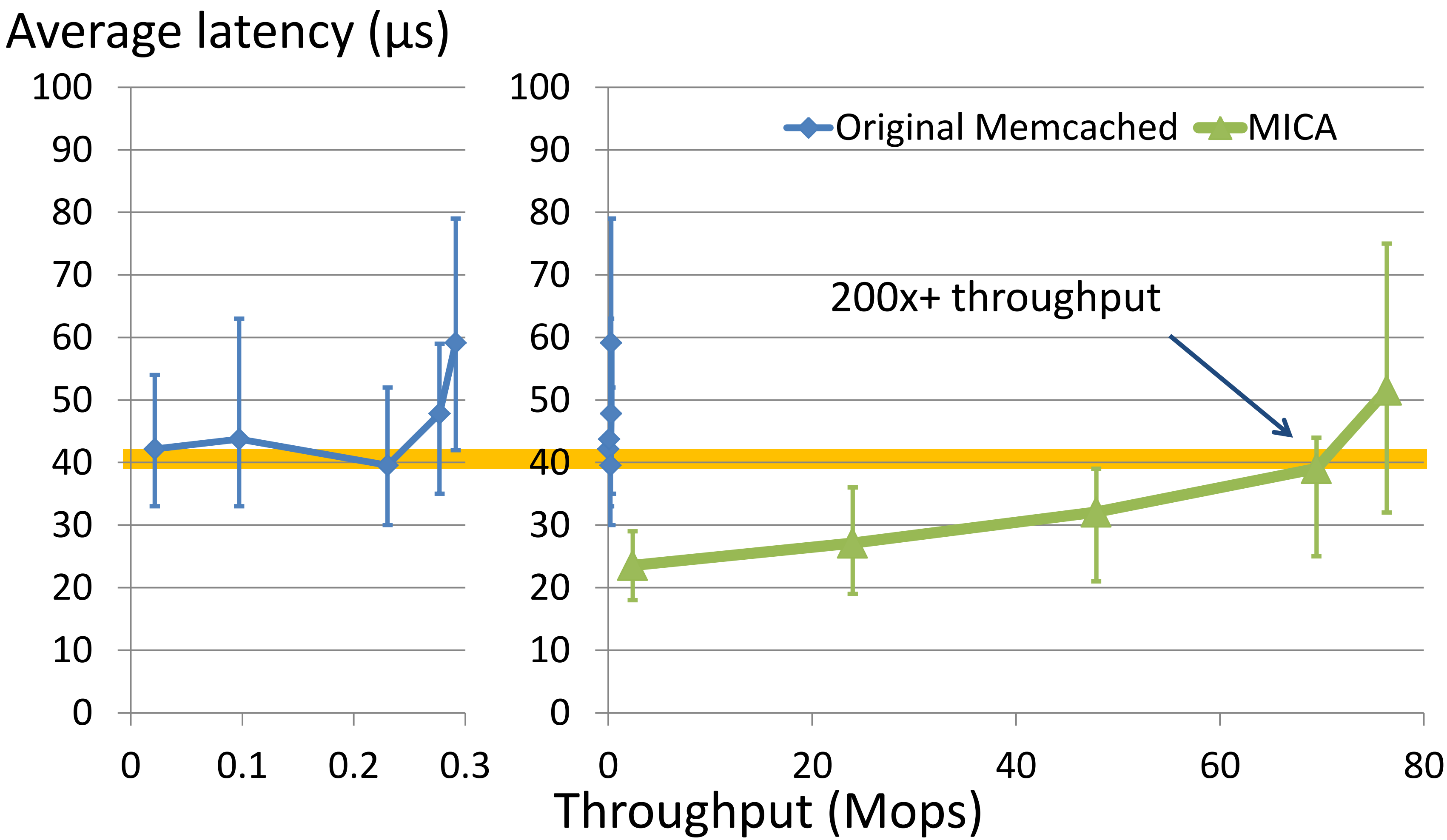
- Going back to end-to-end evaluation…
- Throughput & latency characteristics

Throughput Comparison



End-to-end performance with kernel bypass I/O

Throughput-Latency on Ethernet



Original Memcached using standard socket I/O; both use UDP

MICA

- Redesigning in-memory key-value storage
- 65.6+ Mops/node even for heavy skew/write
- Source code: github.com/efficient/mica