Bringing the Performance to the "Cloud"

Dongsu Han

KAIST
Department of Electrical Engineering
Graduate School of Information Security

The Era of Cloud Computing

Datacenters at Amazon, Google, Facebook

Customers of the Cloud

- Customers “rent” either physical or virtual machines from the cloud.
- Public and private cloud: External or internal

Scale of the "Cloud"

- Facebook: *"hundreds of thousands of machines."*
- Microsoft: 1 million servers
- Google envisions 10 million servers.
- Google spends about $3 Billion every year on data centers.*

***http://www.datacenterdynamics.com/focus/archive/2013/01/google's-data-center-spend-slows-2012**

Efficiency is important

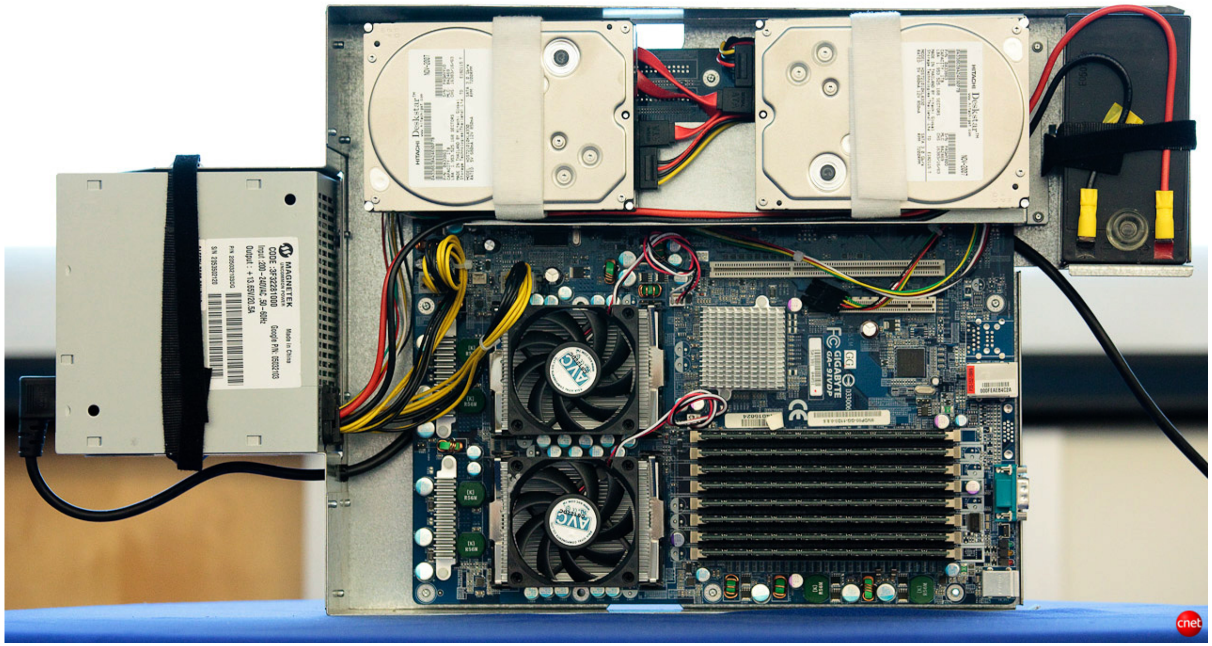
- What if we can increase a single machine's performance by e.g., 10, 20, 40%?
- Equipment cost savings
- Energy savings: By 2012, the cost of power for the data center is expected to exceed the cost of the original capital investment. [U.S DoE]
- Reduced complexity in system design as # of machine involved decreases

How do we improve the Cloud efficiency (and performance)?

We start with a single machine.

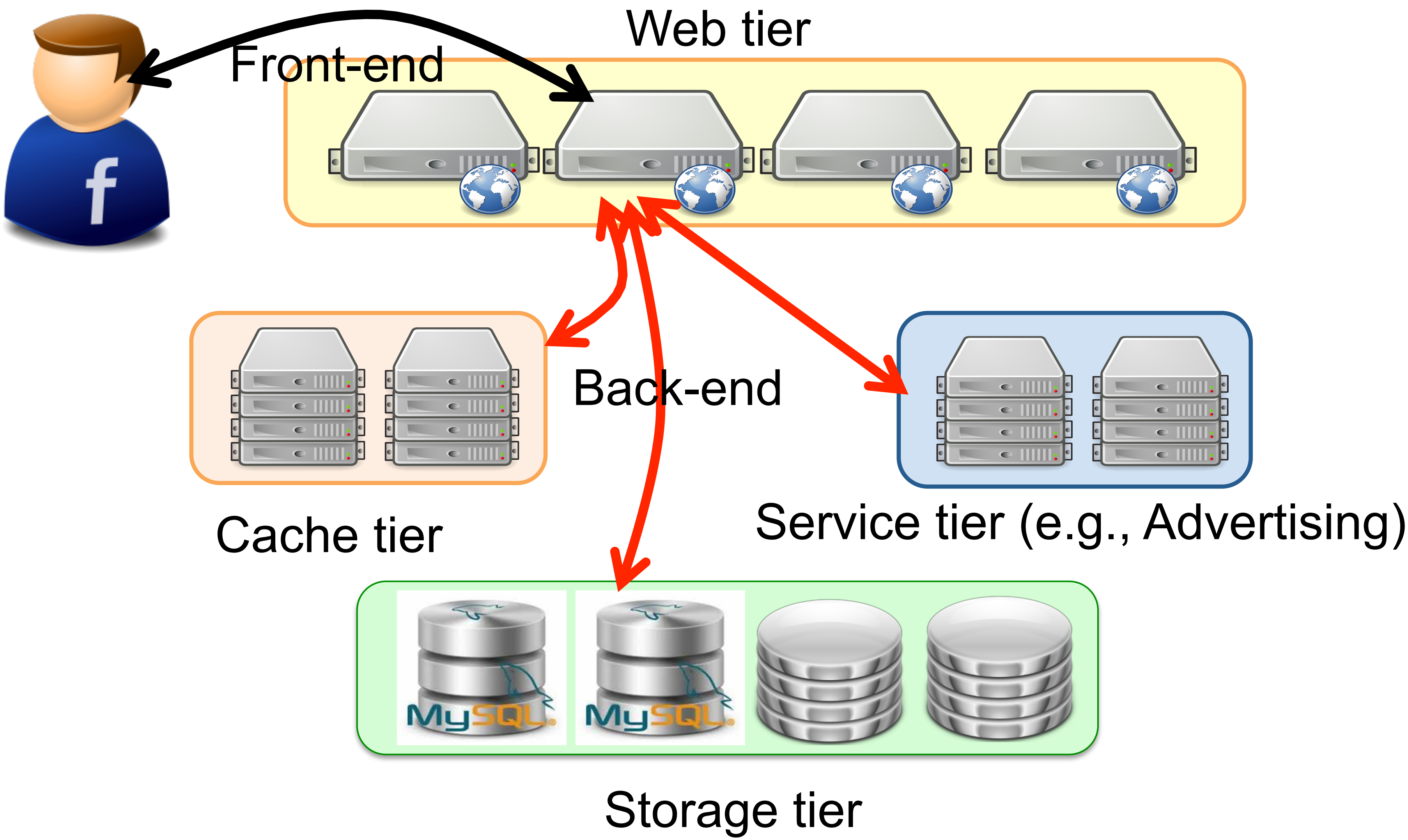
What does a machine look like?
What kind of workload does it handle?

What does a machine look like today?

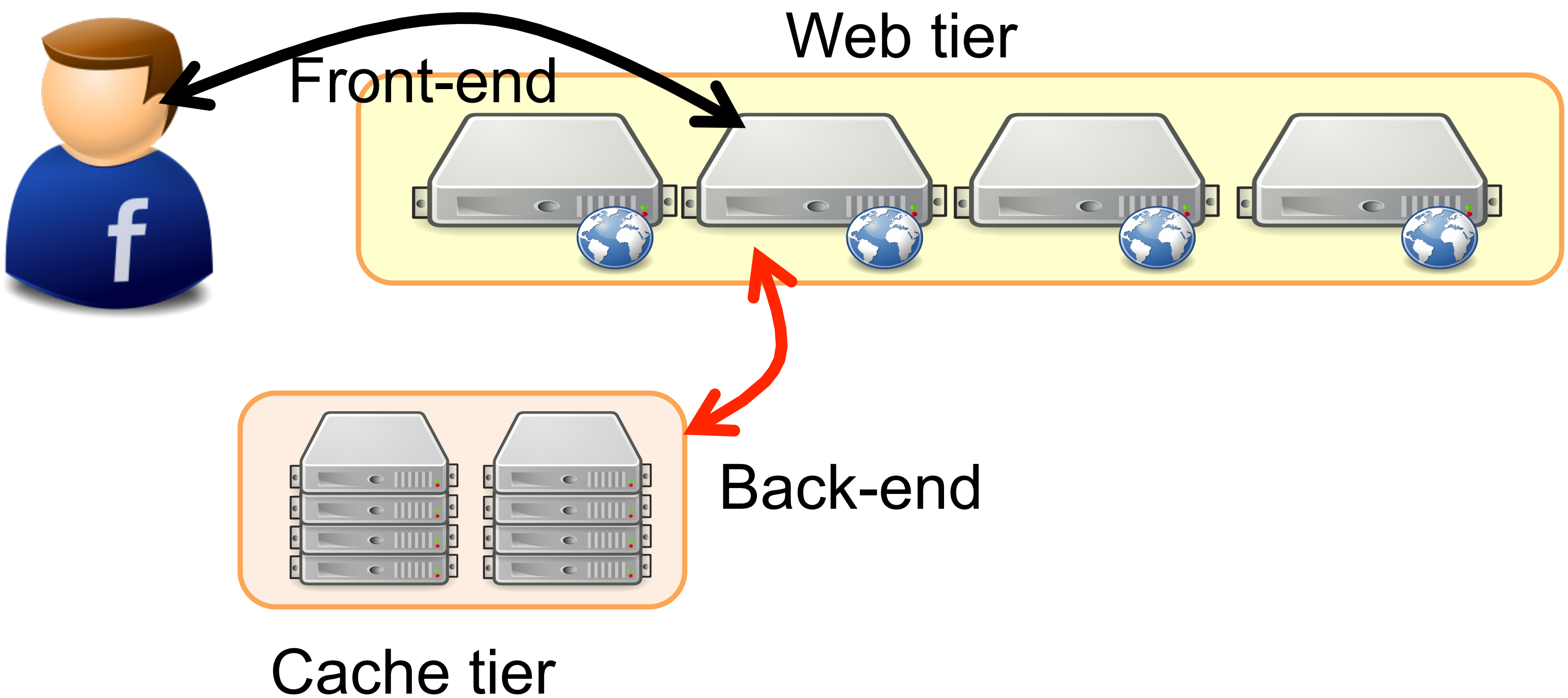


- General purpose hardware (x86 architecture)
- Multicore: 4 ~60 cores (tens of CPU cores)
- Multiple 10-Gigabit Ethernet (becoming the norm)

A Typical Cluster Configuration



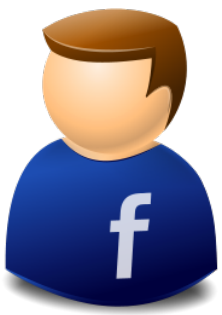
A Typical Cluster Configuration



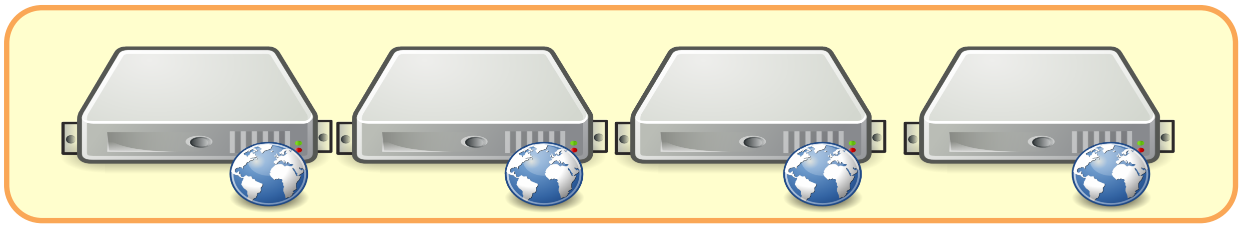
Front-end interaction happens over HTTP (TCP).

Back-end interaction can use any protocol.

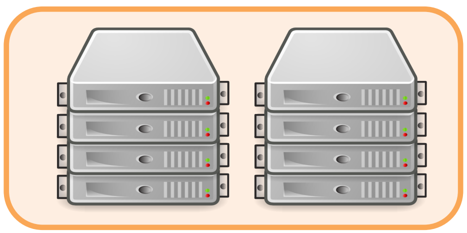
A Typical Cluster Configuration



Web tier



Up to 3x improvement in performance

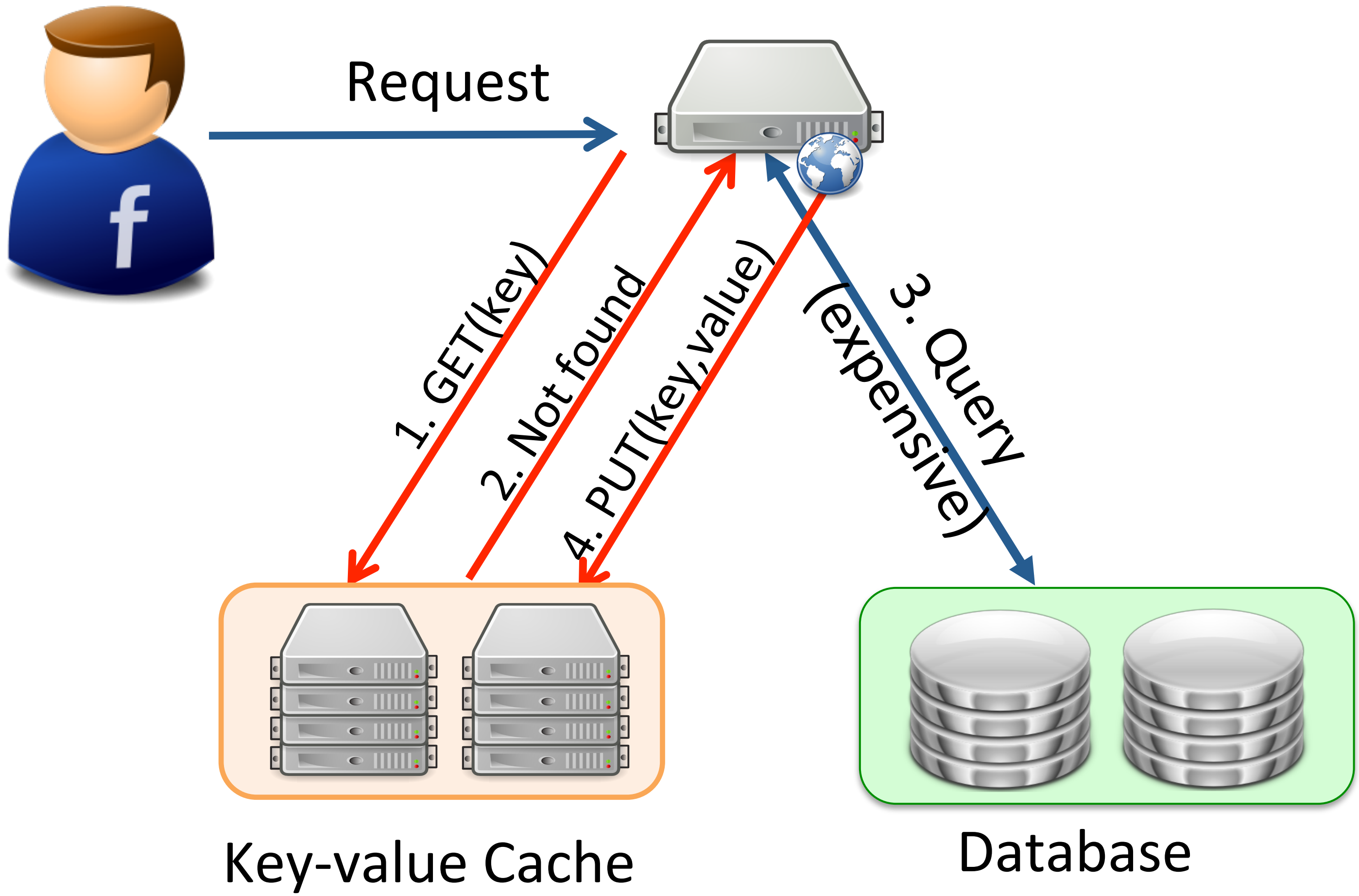


Cache tier

Up to 7x improvement in performance

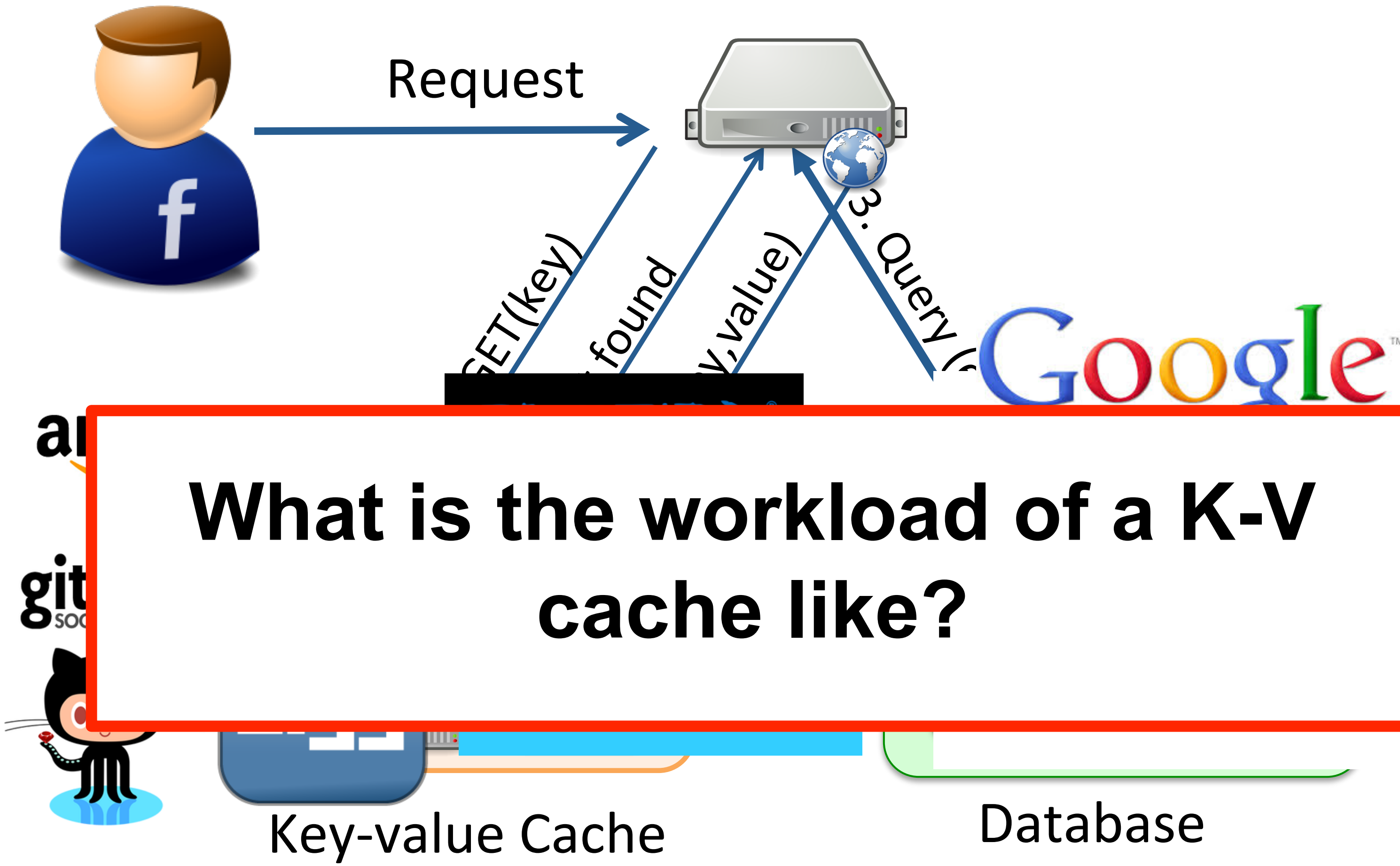
1. Improving the performance of a cache server [NSDI'14]
   Joint work with H. Lim, M. Kaminsky, and D. Andersen
2. Improving the performance of a Web server [NSDI'14]
   w/ E. Jeong, S. Woo, M. Jamshed, H. Jeong, S. Ihm, and K. Park

In-Memory Key-Value Cache



- Key-values are stored in DRAM ("Memcache")

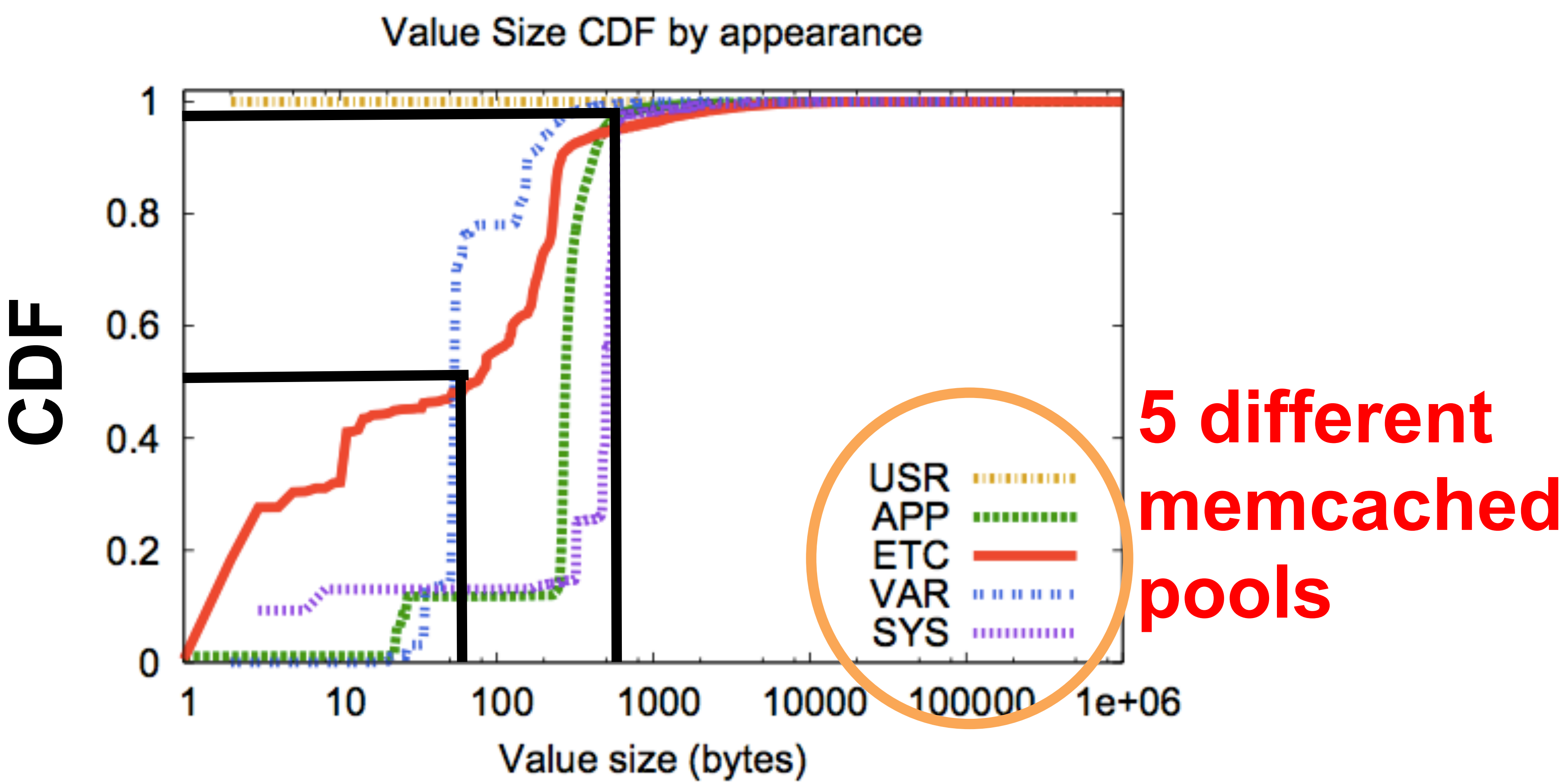
In-Memory Key-Value Cache



What is the workload of a K-V cache like?

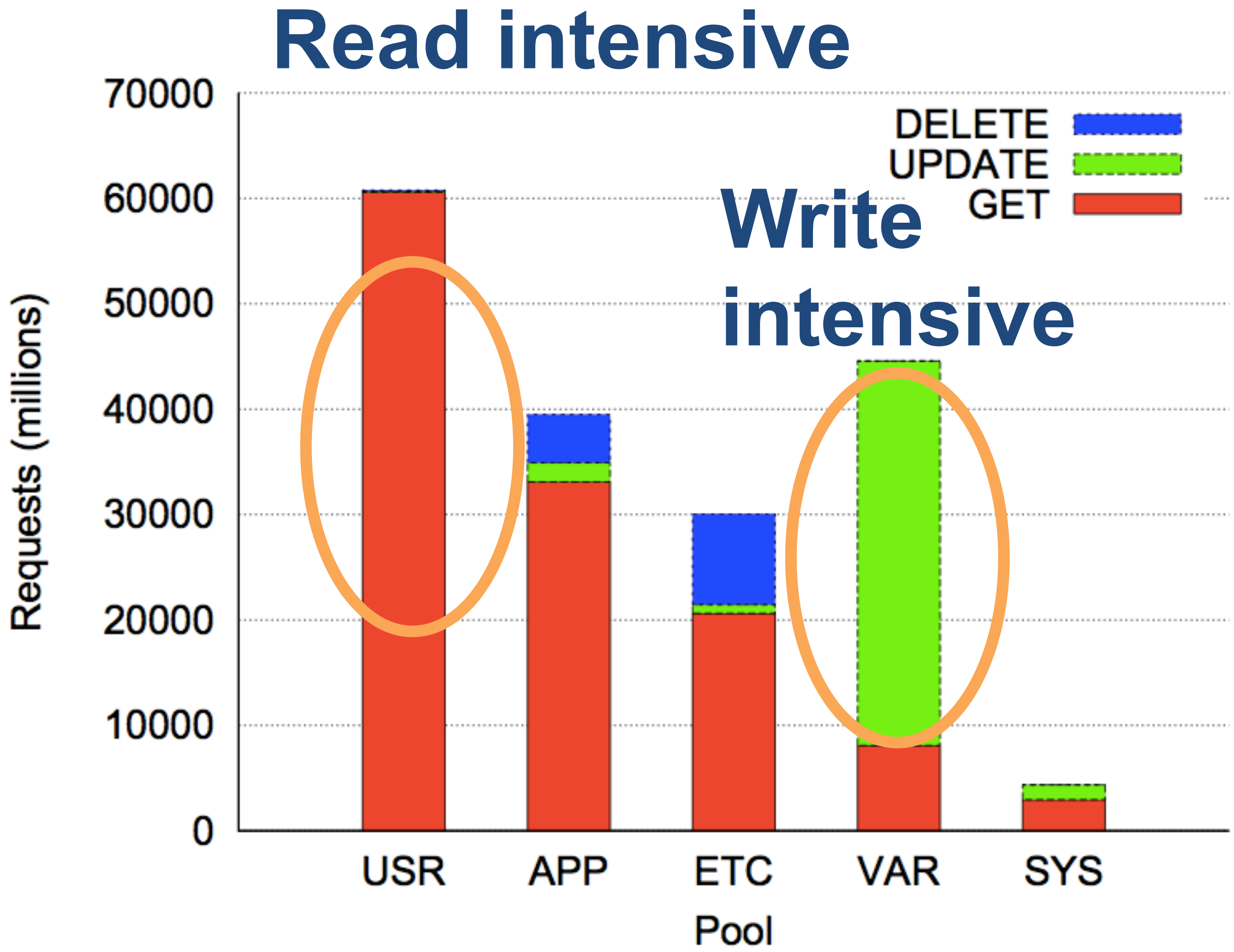
[SOSP] 2007, 2009, 2011 [NSDI] 2013a, 2013b [EuroSys] 2012 [SIGCOMM] 2012 [SOCC] 2010, 2012 [SIGMETRICS] 2012 [ATC] 2013

Workload of a Key-value cache



Facebook K-V size distribution [SIGMETRICS2012]

Diverse Workload [SIGMETRICS2012]



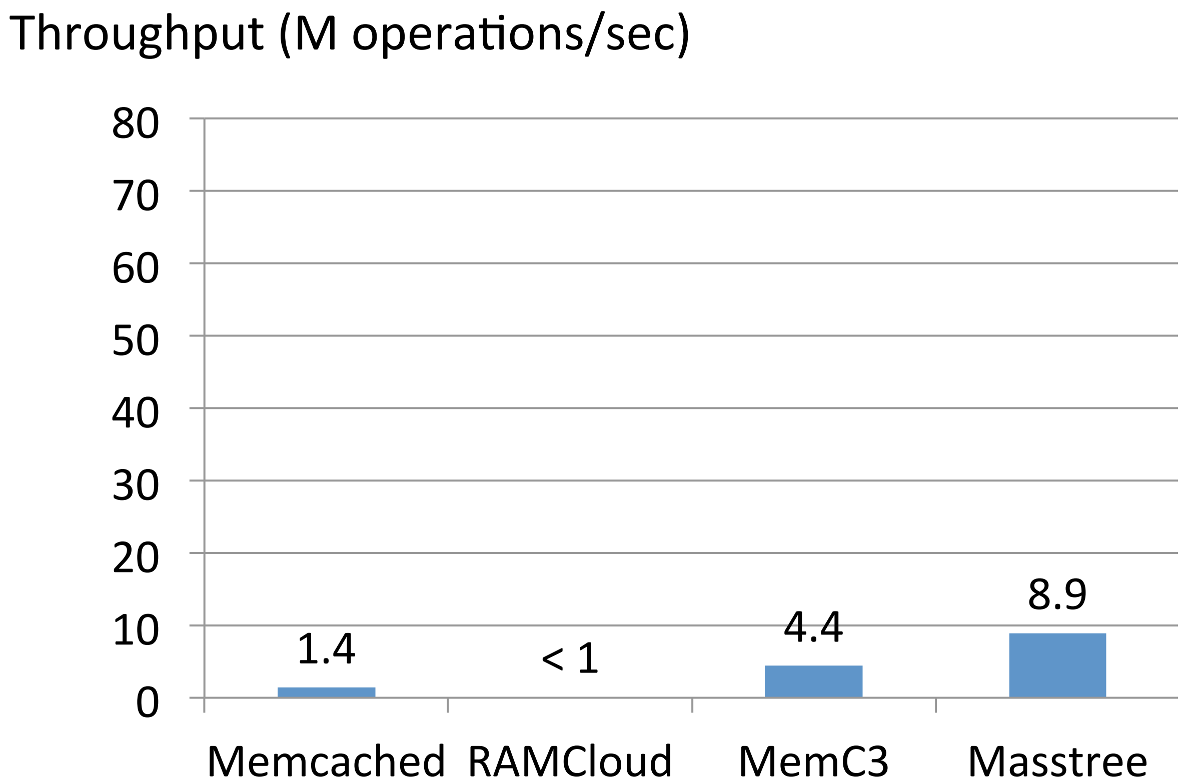
Fast In-Memory Key-Value Cache

Must handle small, variable-length items efficiently

Support both read- and write-intensive workloads

Current K-V Cache Performance

Workload: YCSB-B (95% GET, 5% PUT)



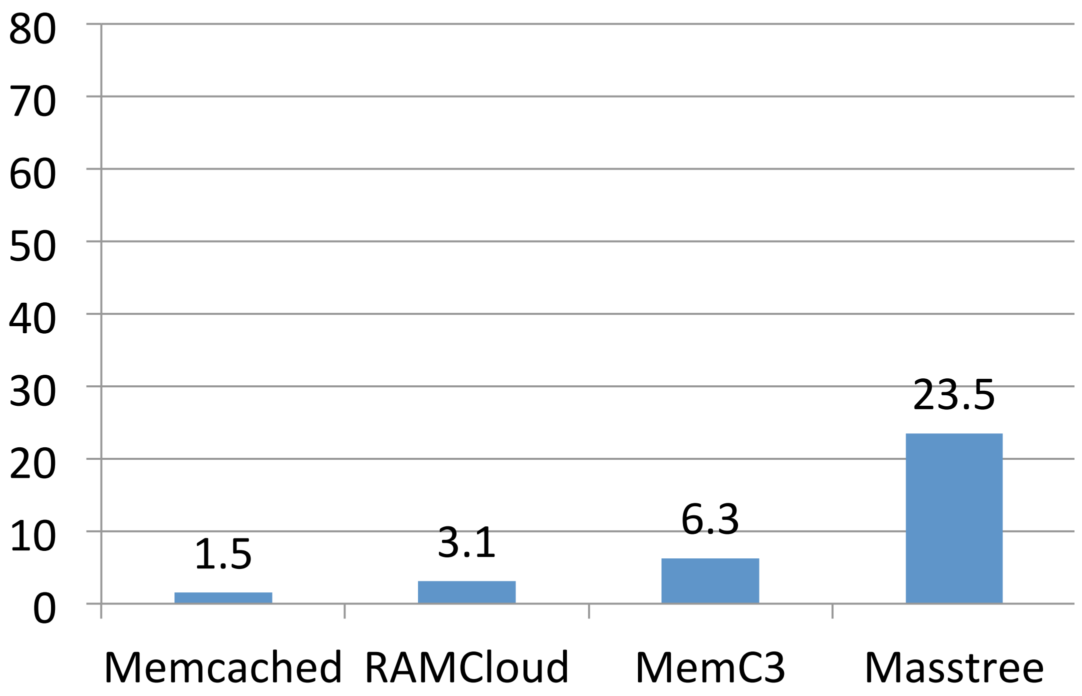
End-to-end performance using UDP

Server equipped with dual 8-core @2.7 GHz, 80 GbE

Read-Intensive Workload

Workload: YCSB-B (95% GET, 5% PUT)

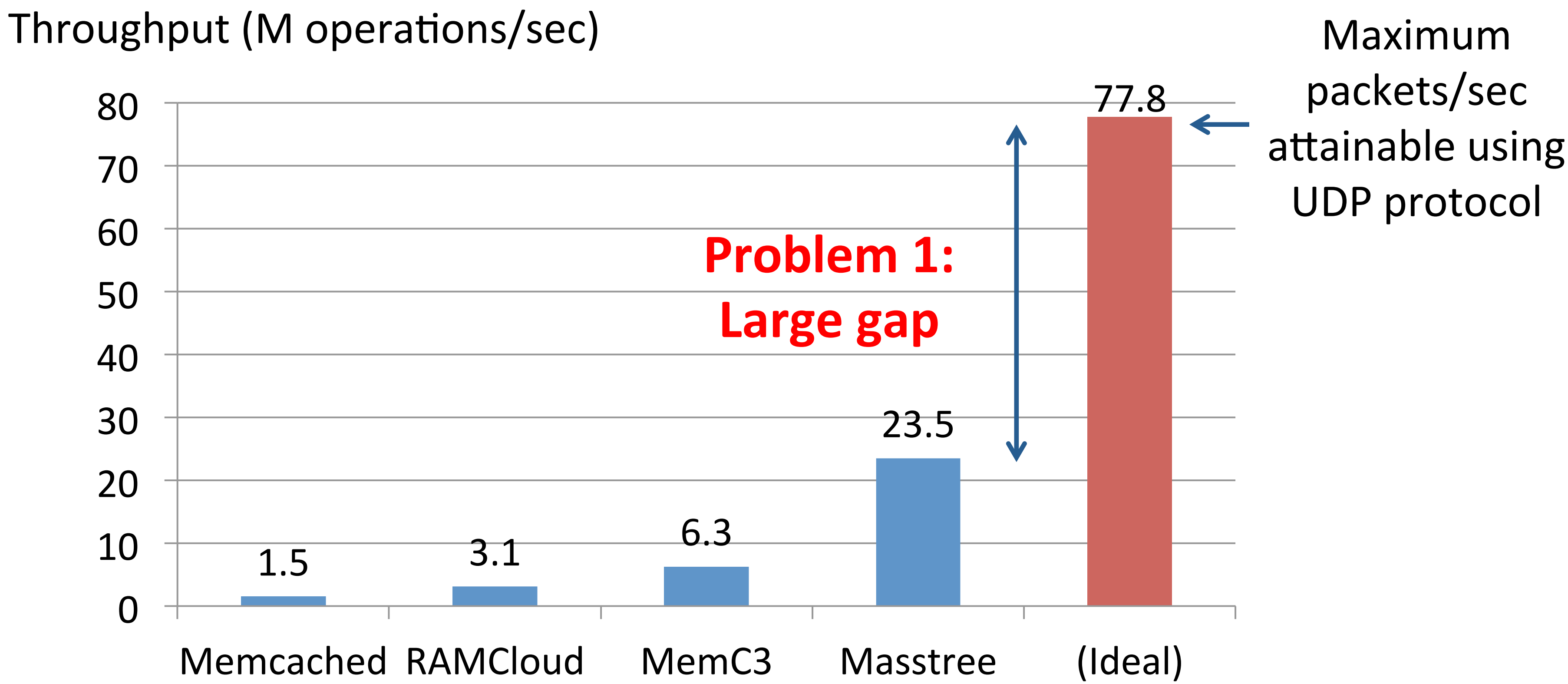
Throughput (M operations/sec)



End-to-end performance using our optimized network stack
Server equipped with dual 8-core @2.7 GHz, 80 GbE

Read-Intensive Workload

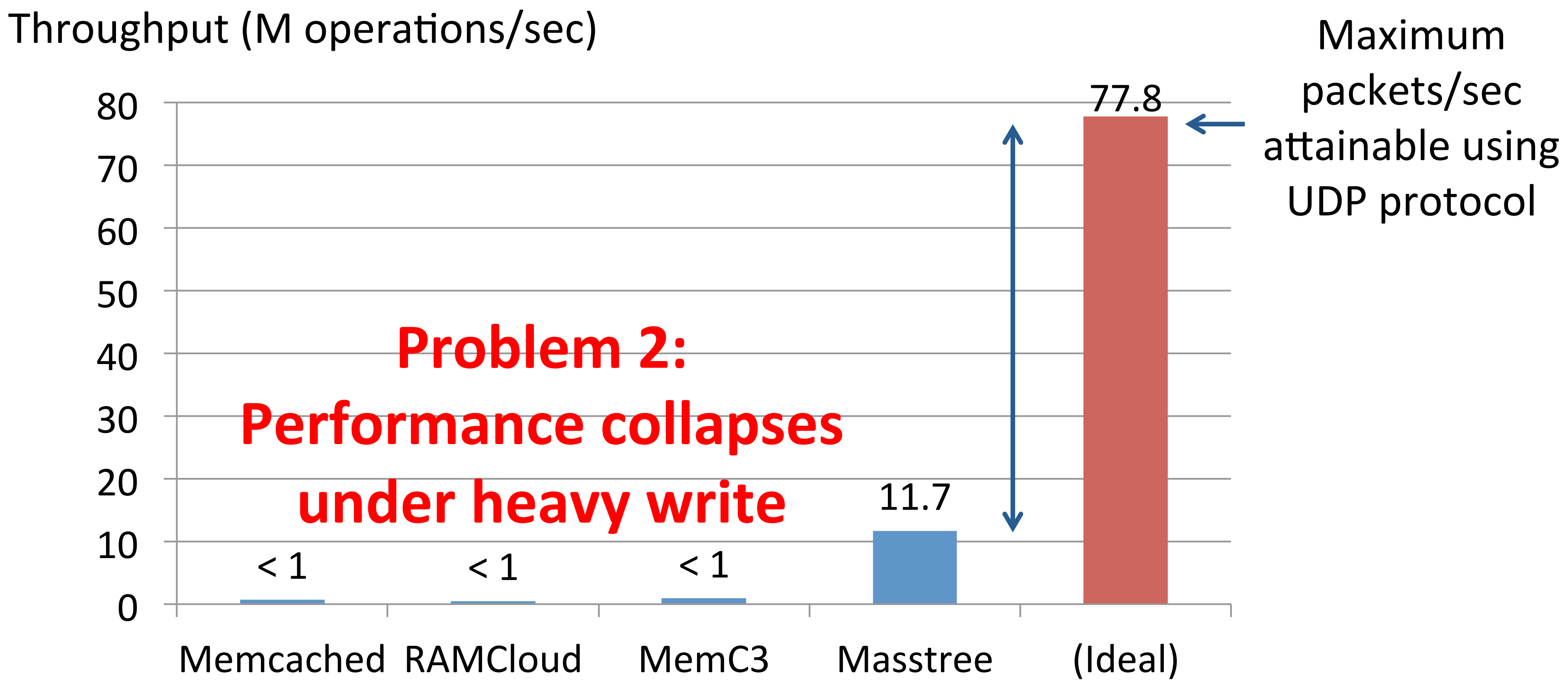
Workload: YCSB-B (95% GET, 5% PUT)



End-to-end performance using our optimized network stack
Server equipped with dual 8-core @2.7 GHz, 80 GbE

Write-Intensive

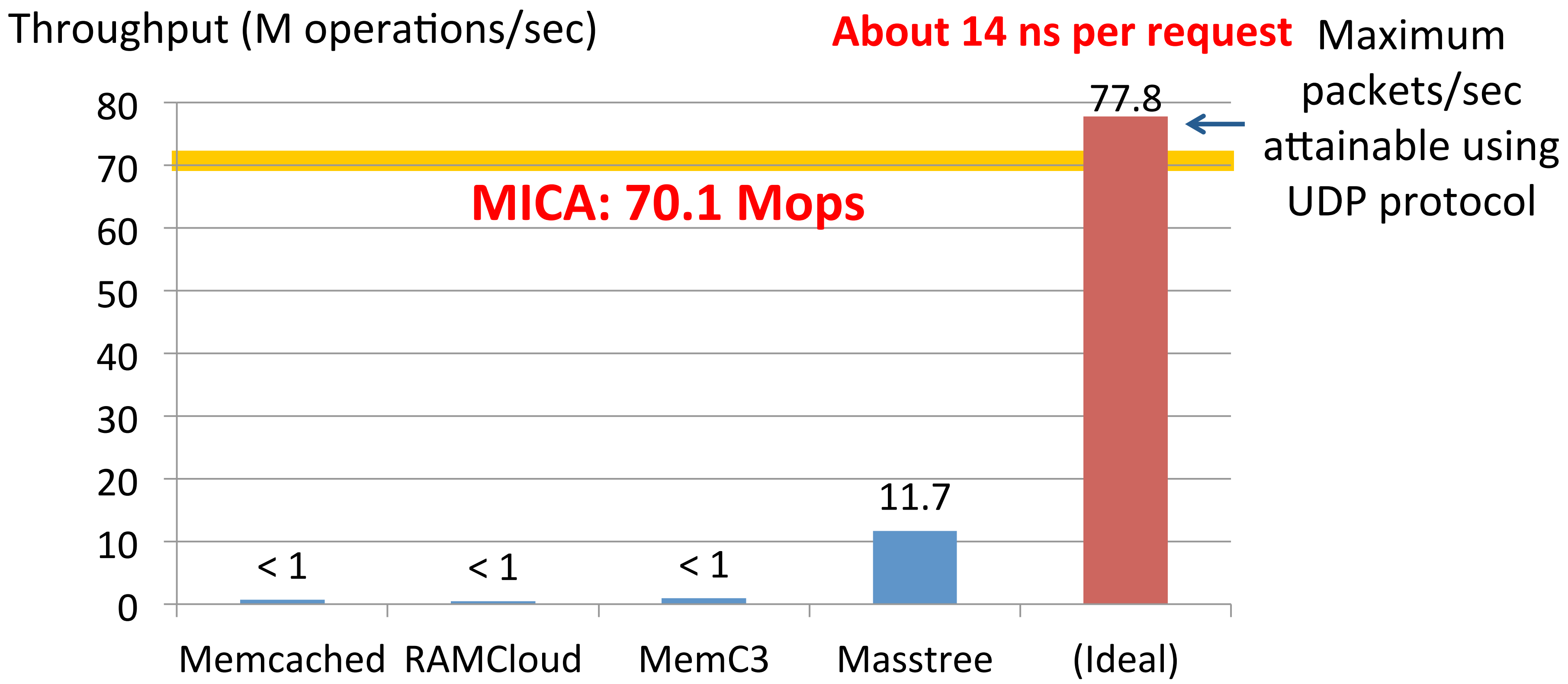
Workload: YCSB-A (50% GET, 50% PUT)



End-to-end performance using our optimized network stack
Server equipped with dual 8-core @2.7 GHz, 80 GbE

MICA Performance Preview

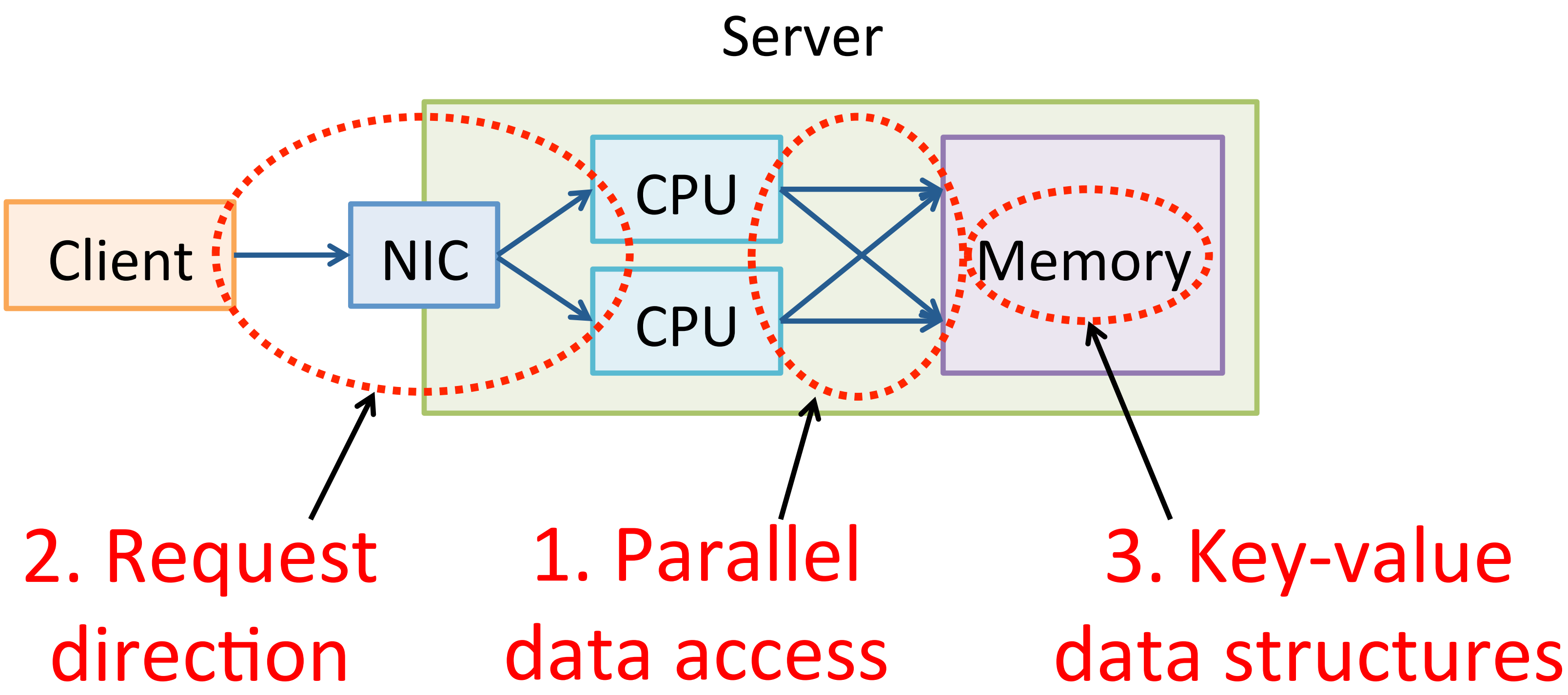
Workload: YCSB-A (50% GET, 50% PUT)



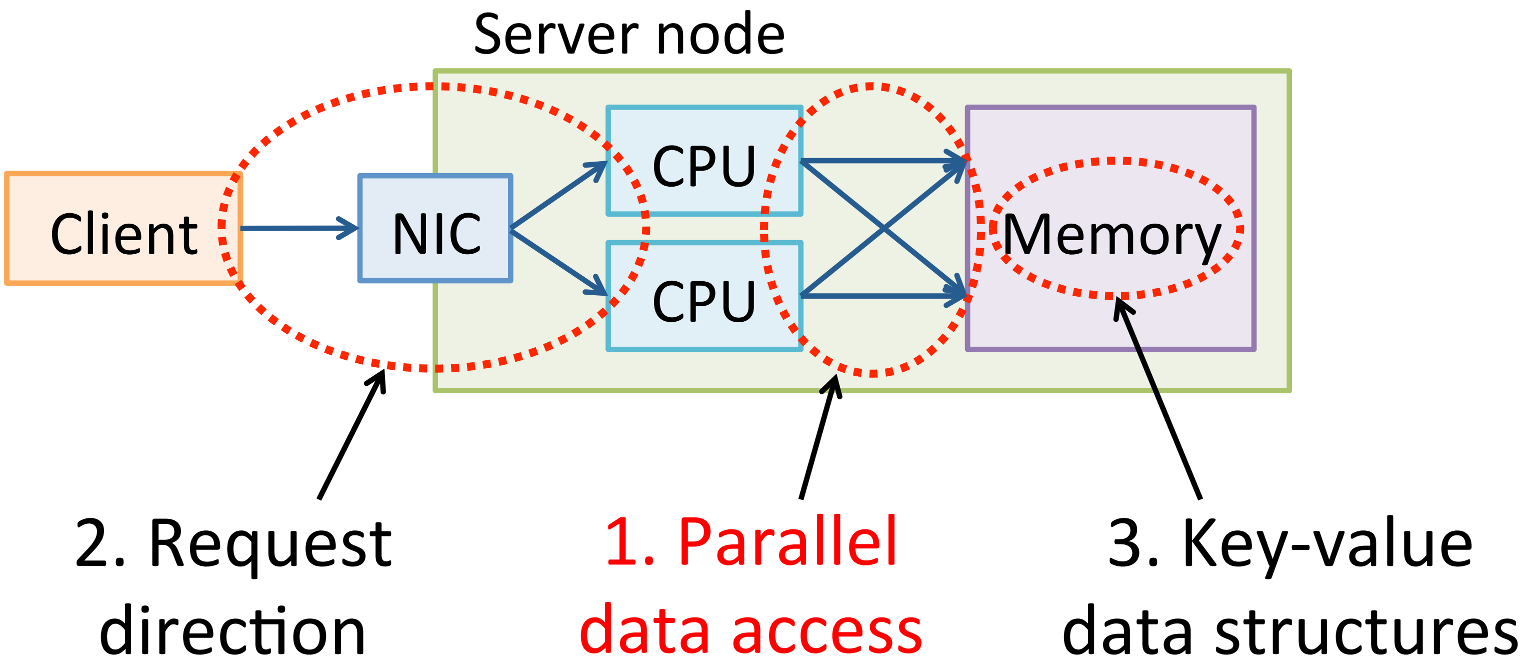
End-to-end performance using our optimized network stack
Server equipped with dual 8-core @2.7 GHz, 80 GbE

MICA Approach

MICA redesigns a K-V cache in a holistic way.



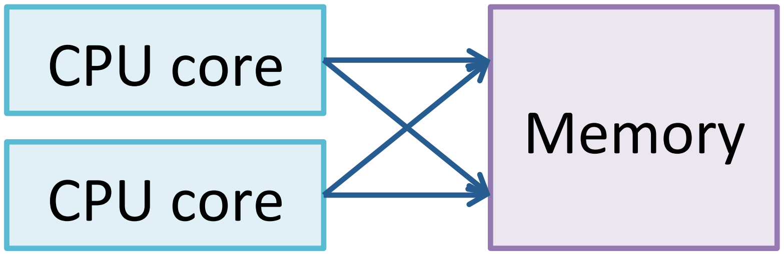
Parallel Data Access



- Modern CPUs have many cores (8, 12, …)
- We must exploit CPU parallelism <u>efficiently</u>.

Concurrent Read/Write (CRCW)

- Any core can read/write any part of memory



+) Can distribute load to multiple cores

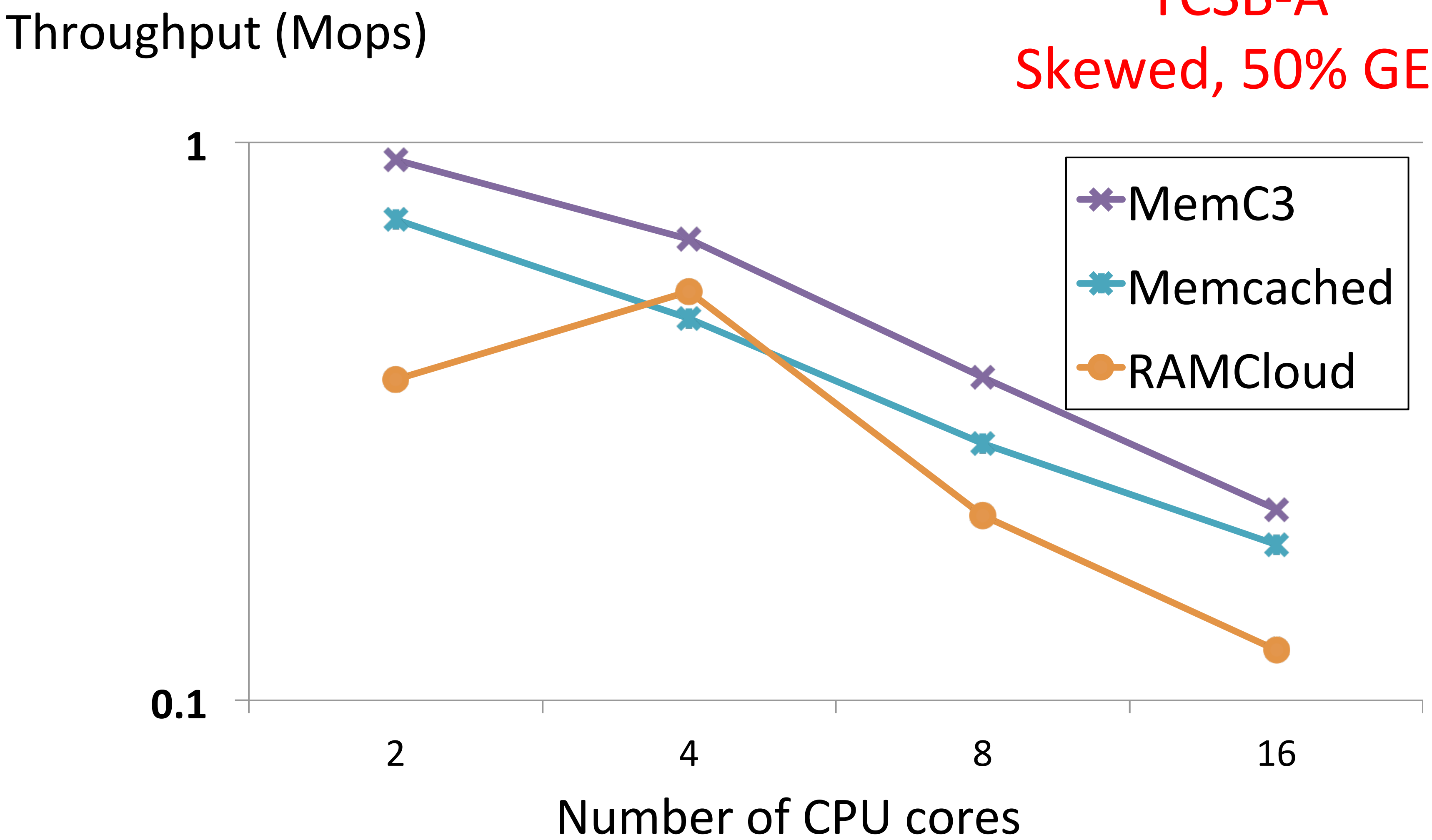
- Memcached, RAMCloud [SOSP], MemC3 [NSDI], Masstree [EuroSys]

-) Limits scalability with multiple cores

- Lock contention
- Expensive cacheline transfer caused by concurrent writes on the same memory location

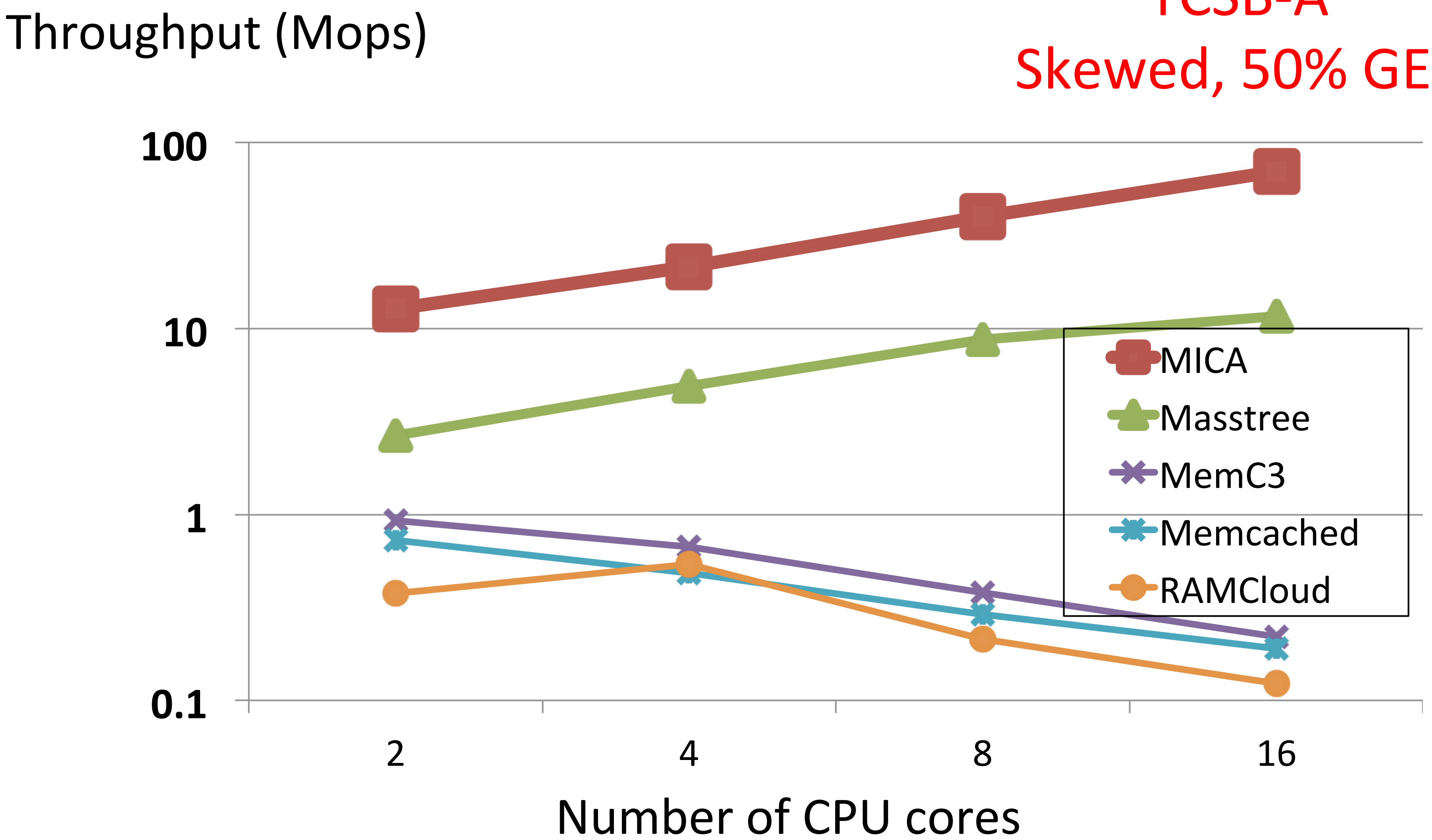
MICA Scales Well with Many Cores

YCSB-A
Skewed, 50% GET



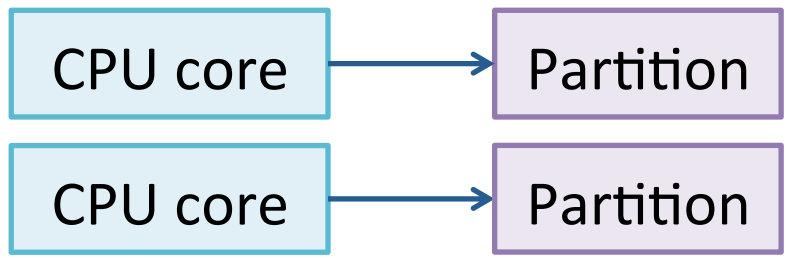
MICA Scales Well with Many Cores

YCSB-A
Skewed, 50% GET



MICA's Parallel Data Access

- Partition data using the hash of keys
- Exclusive Read/Write (EREW)
  - Only one core accesses a particular partition

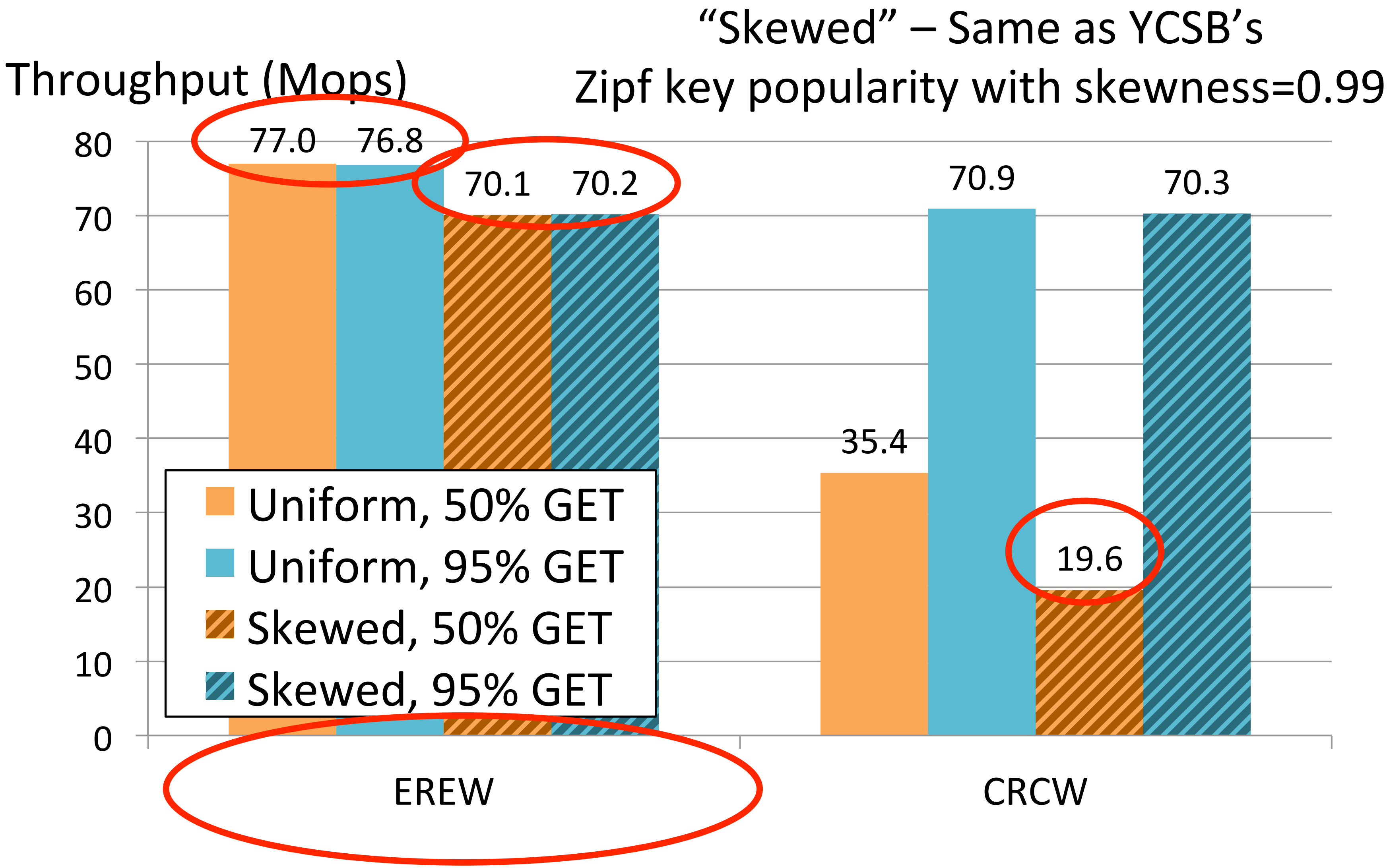


+) Avoids synchronization/inter-core communication [H-Store,VoltDB]

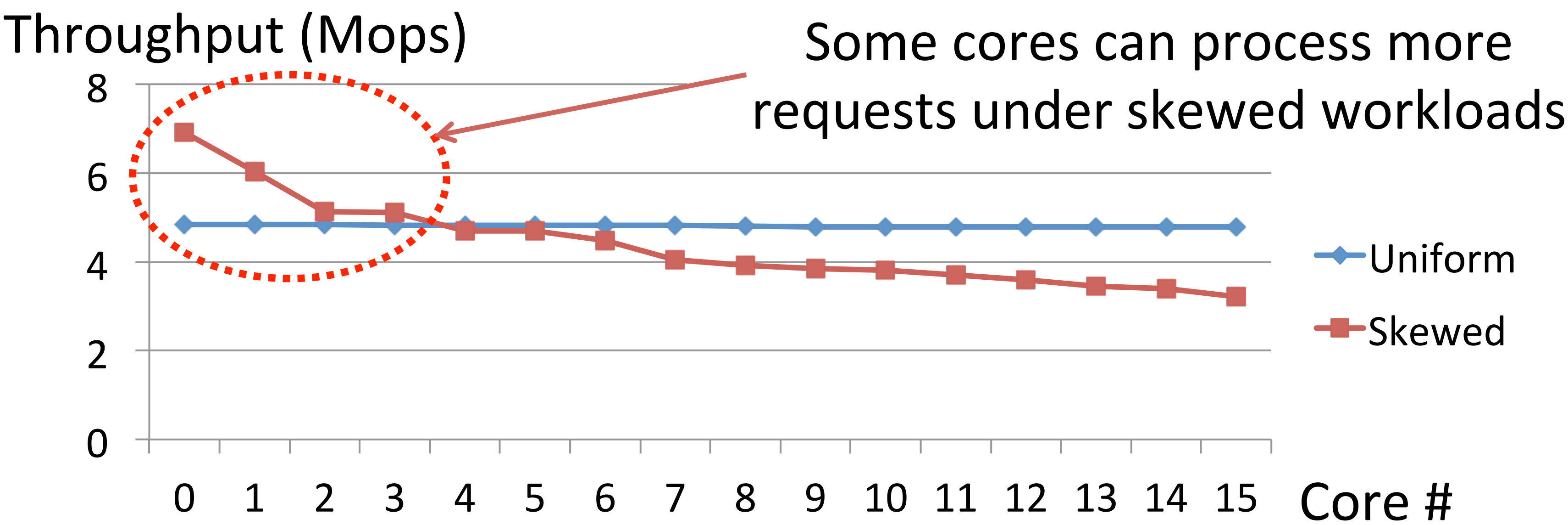
-) Can be slow under skewed key popularity

- A popular item cannot be served by multiple cores

EREW Outperforms CRCW



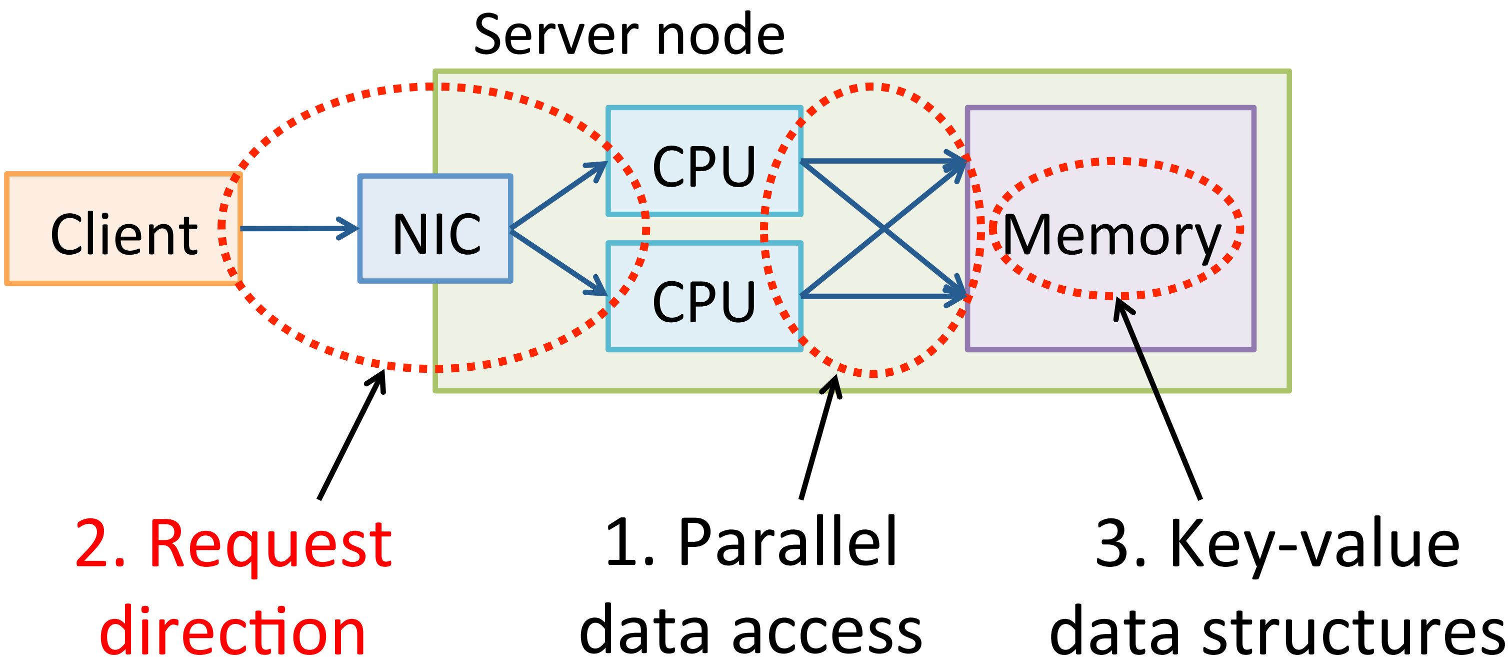
Skew Does Not Hurt (Much)



Per-core performance breakdown

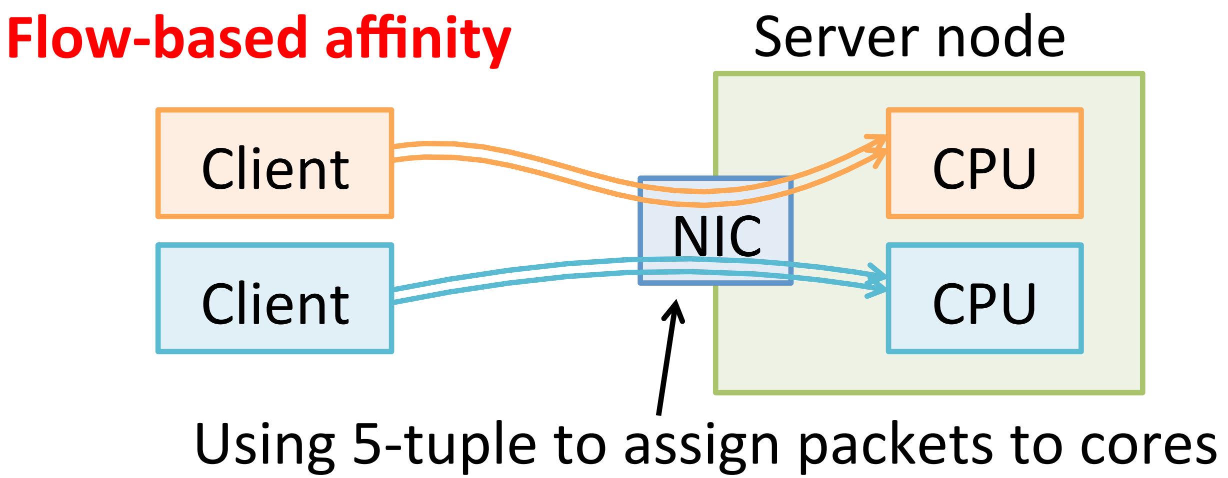
- Hot partitions contain a few popular keys, making CPU cache very effective

Request Direction



- EREW requires correct request direction.
- A request must be sent to the core/partition that handles the requested key.

Common Request Direction Scheme

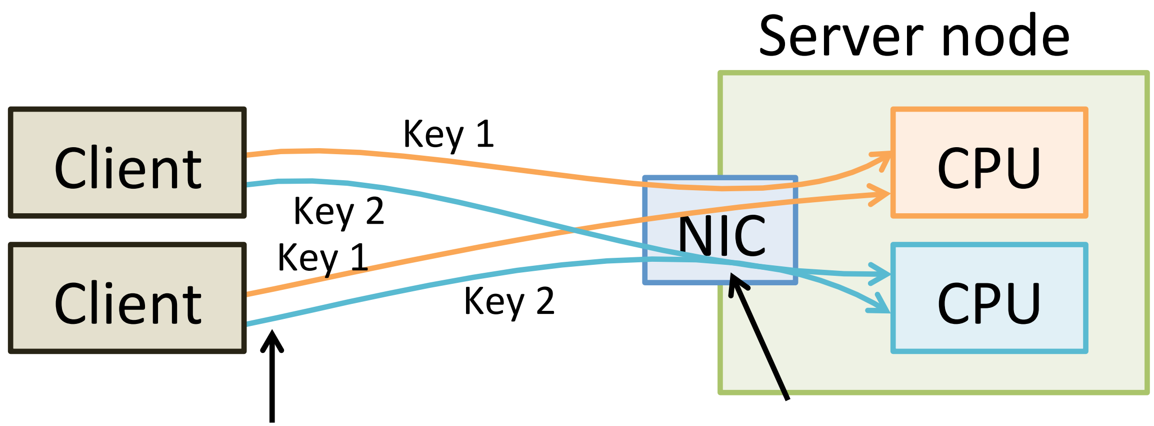


- Useful for flow-based protocols (e.g., TCP)
- Does not work well with MICA's EREW
  - A client can request keys from different partitions

MICA's Request Direction

Object-based affinity

- MICA overcomes commodity NICs' limited programmability by using client assistance

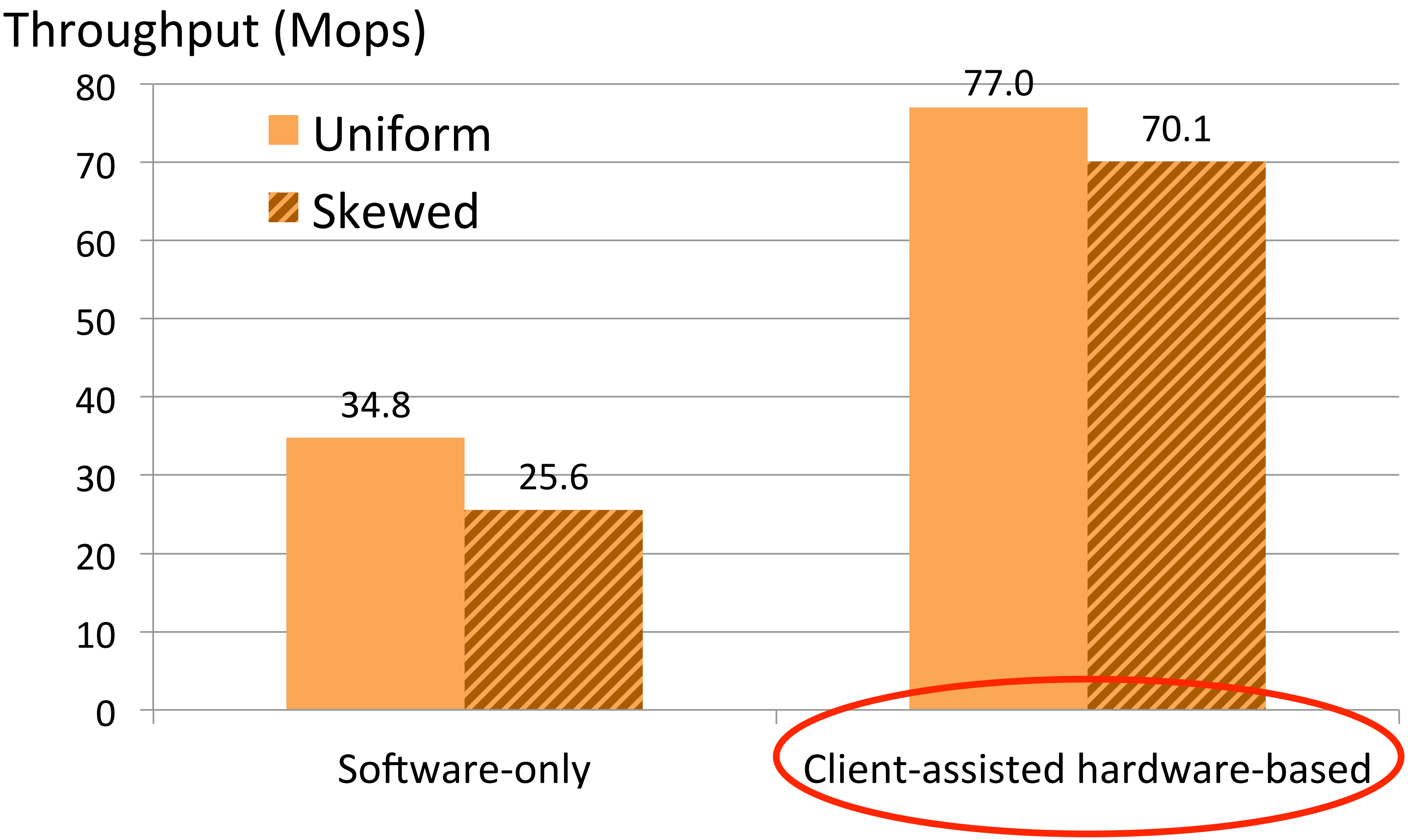


Key's partition ID is put into <u>packet header</u> (UDP dest port)

Programmed to use the embedded partition ID to direct packets to cores

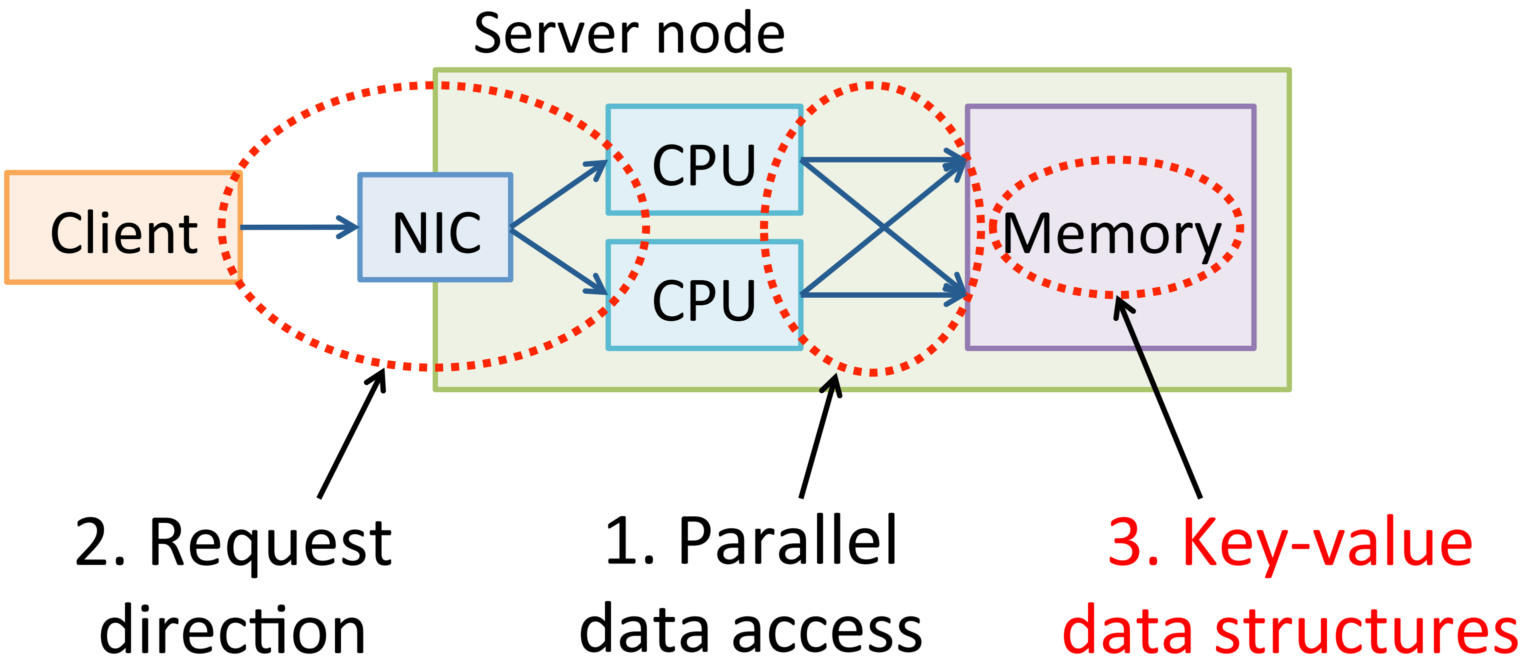
- Uses Intel Data Plane Development Kit (DPDK) for low-overhead burst packet I/O bypassing OS kernel

NIC HW for Request Direction



Using EREW for parallel data access

Key-Value Data Structures



- Significant impact on key-value processing speed
- New design required to support both read and write operations at high speeds

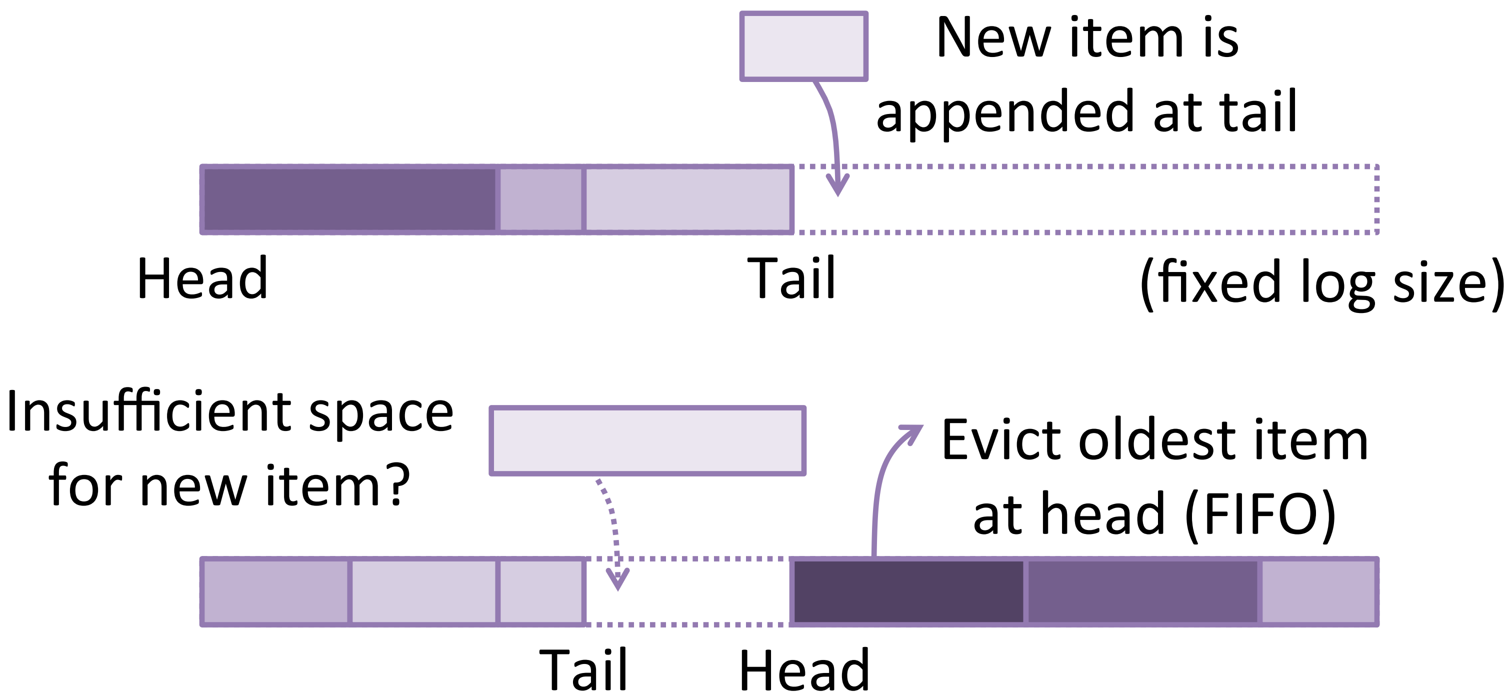
MICA's Key-Value Data Structures

- Each partition has two data structures:
  - Circular log store
  - Lossy concurrent hash index

- Omitted in this talk: numerous optimizations
  - Garbage collection
  - LRU approximation
  - NUMA-aware memory allocation, memory mapping
  - memory prefetching, cache-friendly data structures
  - Concurrency support (for "CREW")

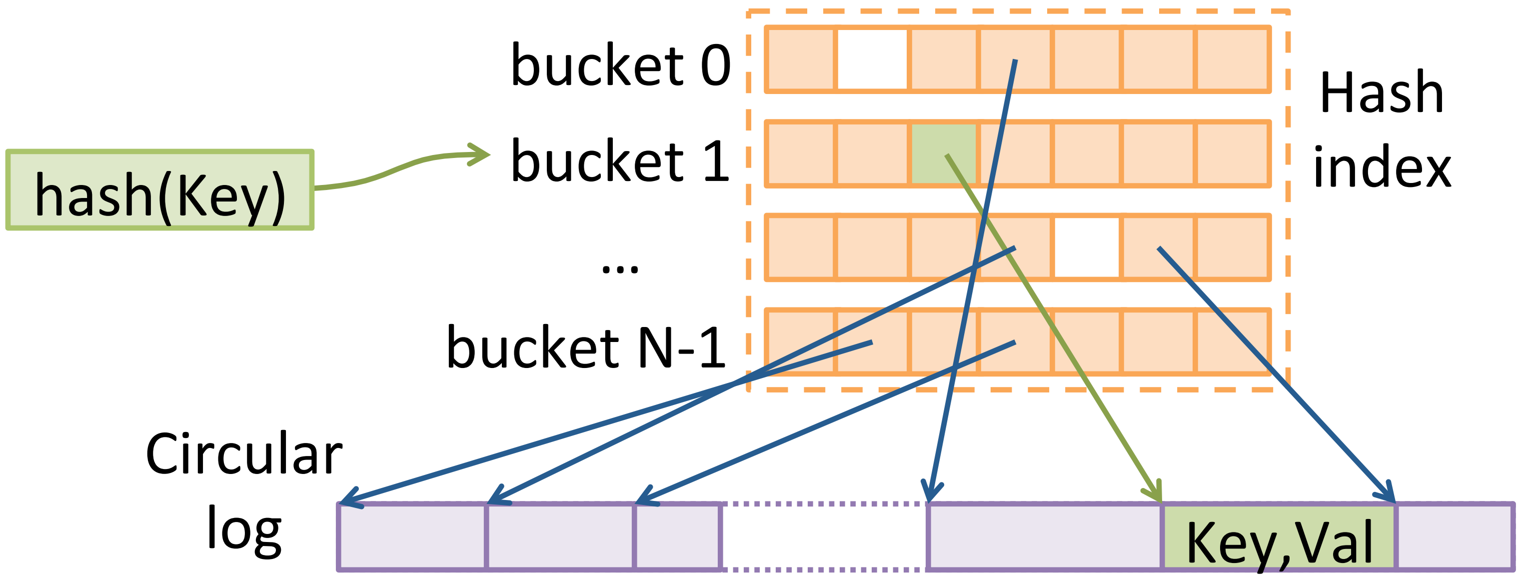
Circular Log Store

- Allocates space for key-value items of any length
- Simple garbage collection and free space defragmentation



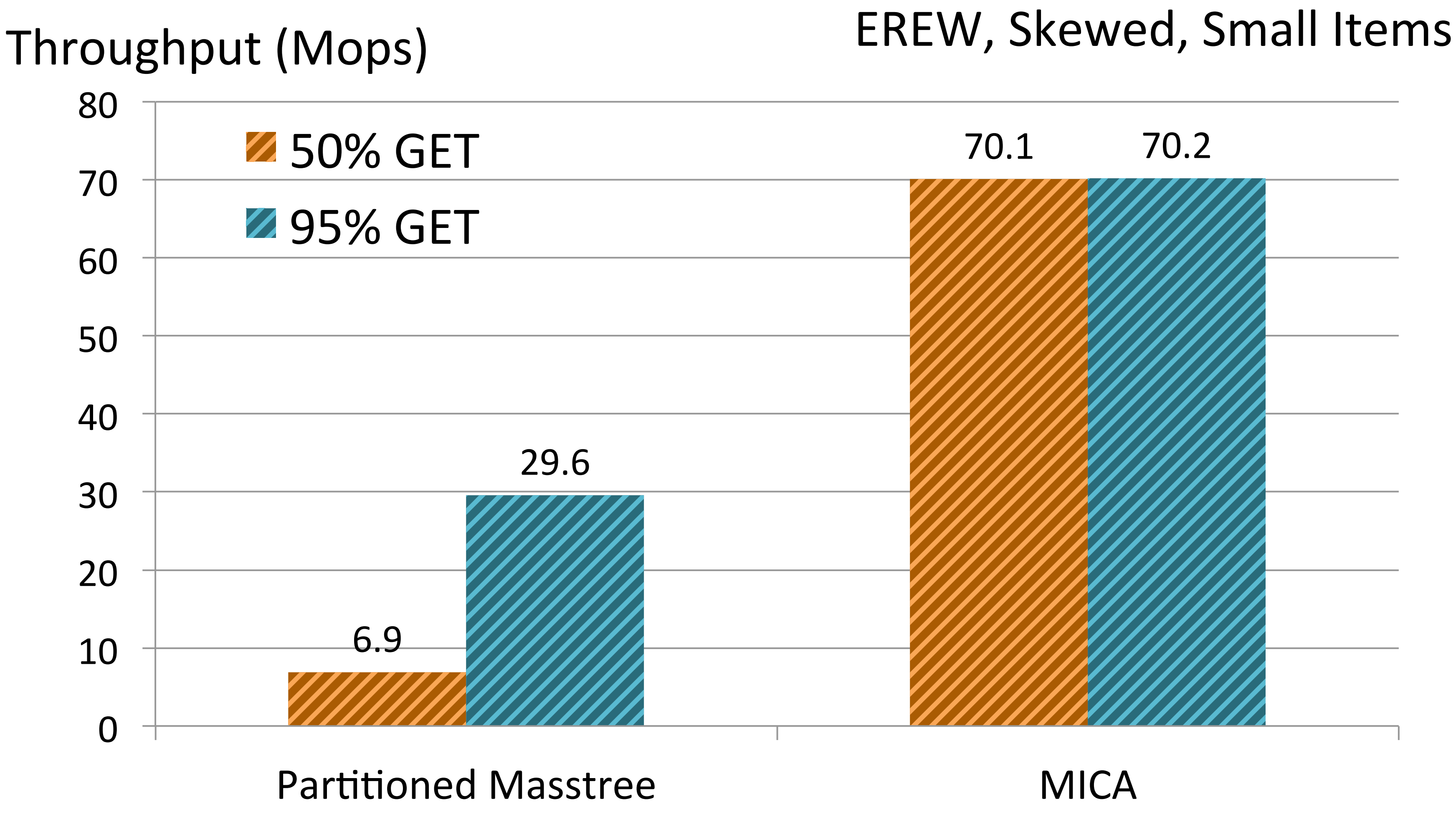
Lossy Concurrent Hash Index

- Indexes items in the circular log with a set-associative hash index

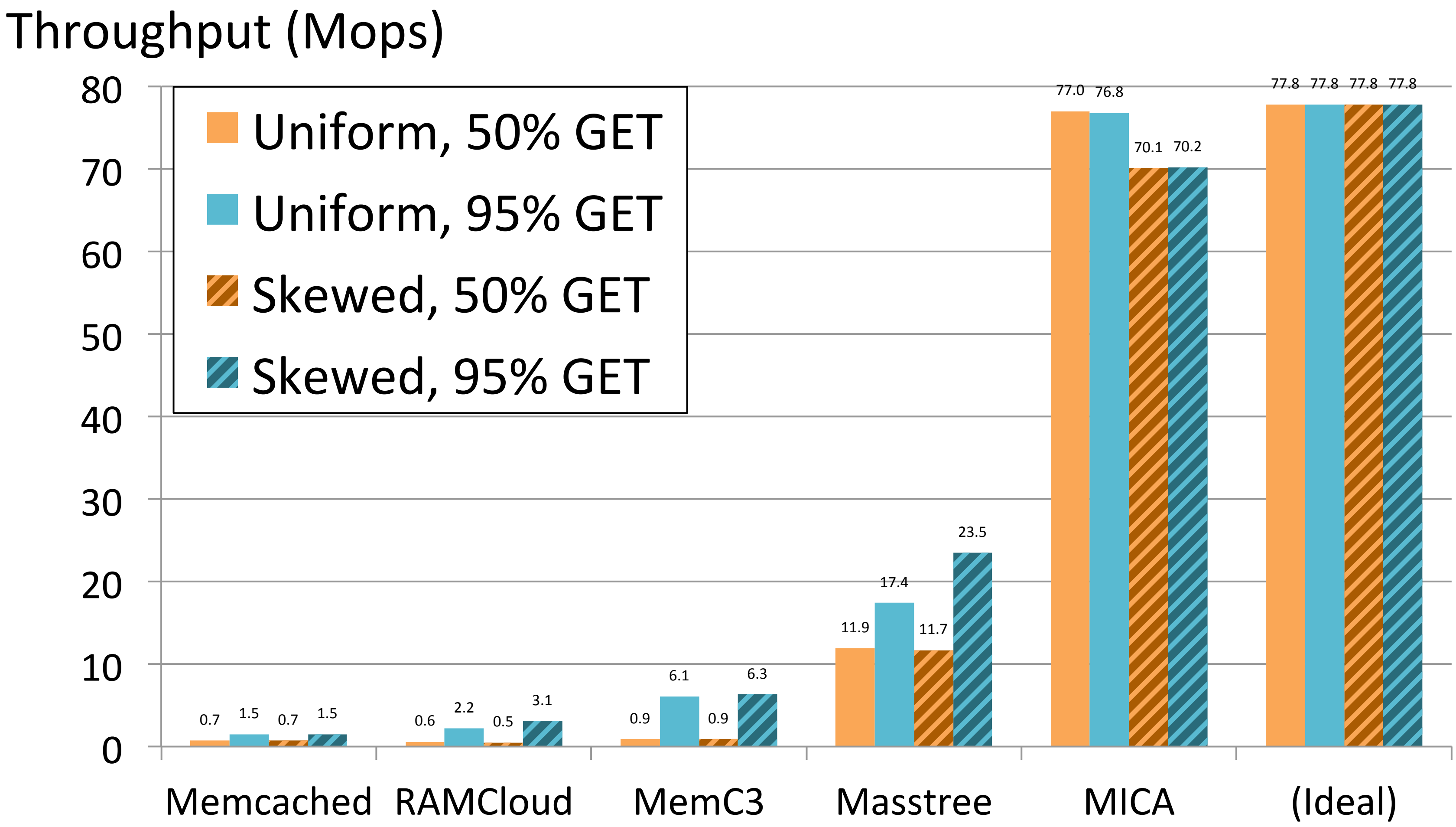


- Full bucket? Evict oldest entry from it (FIFO)
  - Allows fast indexing of new key-value items

Key-Value Data Structure Comparison

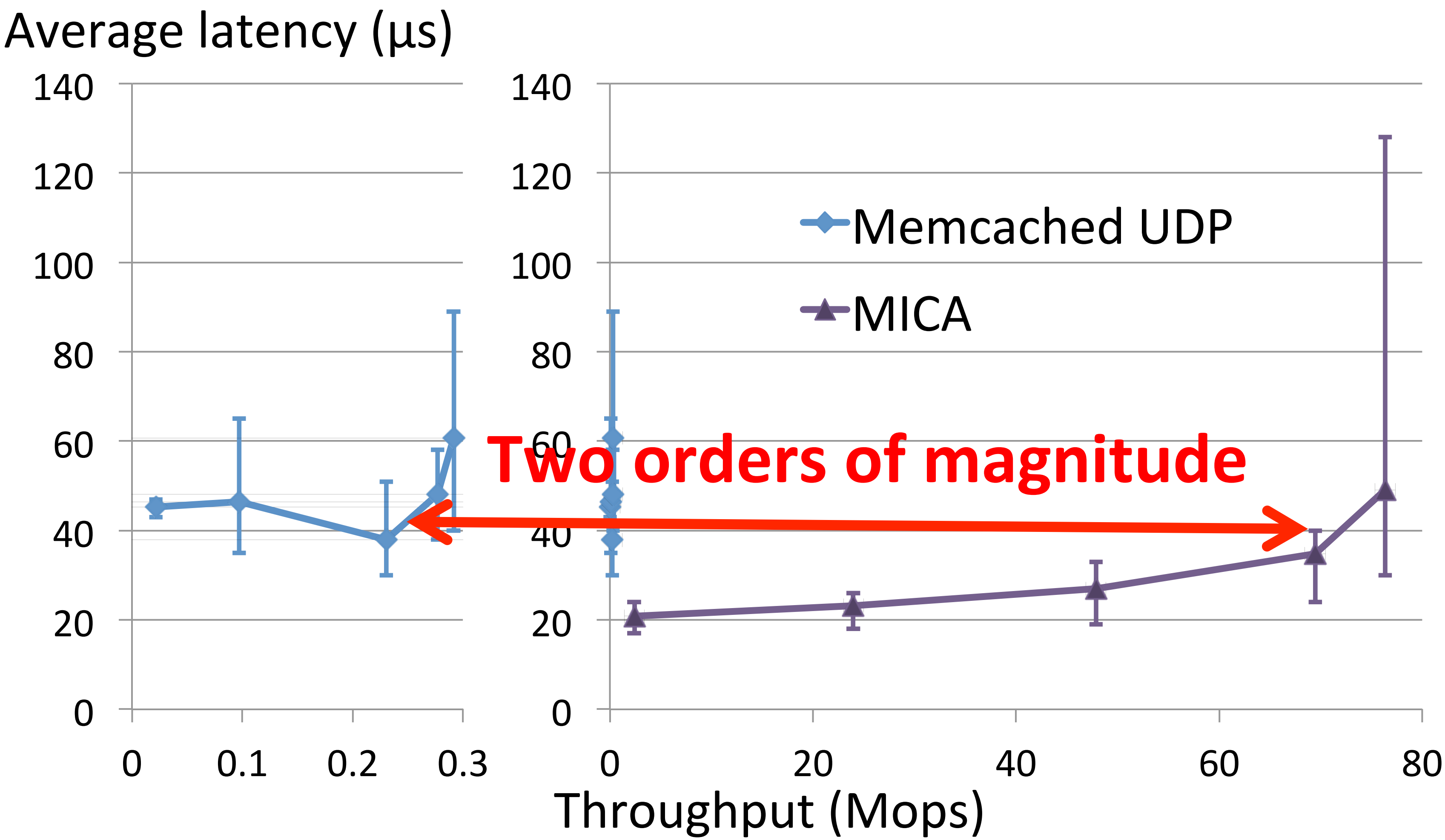


Throughput Comparison



End-to-end performance using our optimized network stack

Throughput-Latency (on Ethernet)

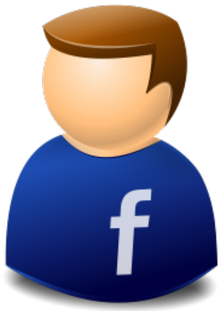


Memcached UDP using standard socket I/O

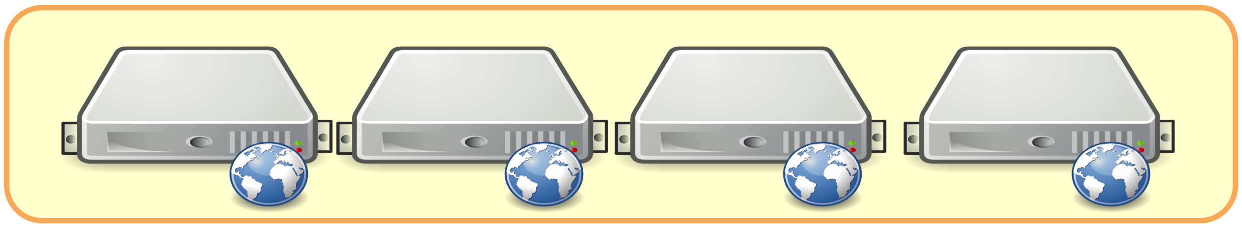
Summary

- MICA takes a holistic approach to designing fast in-memory key-value caches.
  - Efficient parallel data access
  - Hardware-based request direction
  - Optimized data structures for key-value caching
- MICA consistently achieves high performance under diverse workloads.

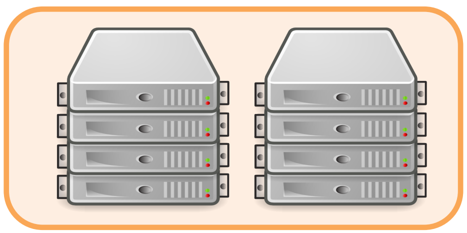
Typical Server Cluster Configuration



Web tier



Up to 3x improvement in performance



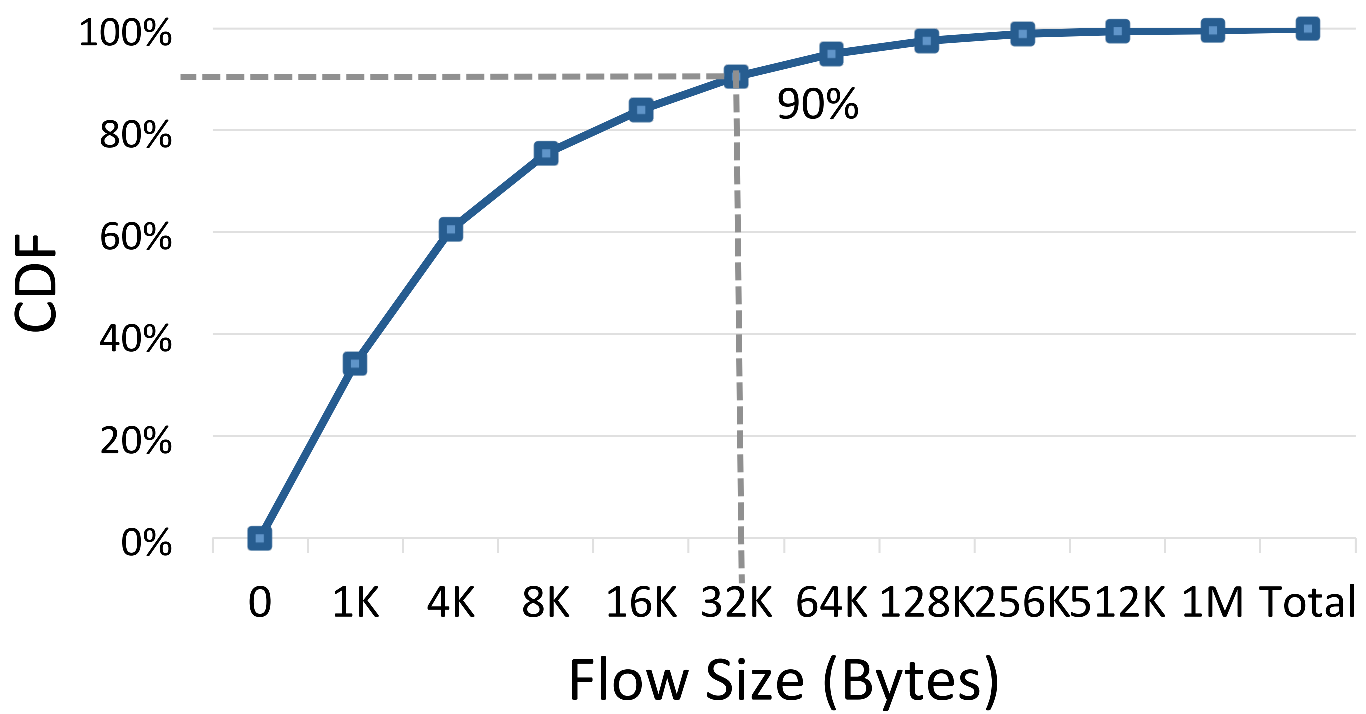
Cache tier

Up to 7x improvement in performance

1. Improving the performance of a cache server [NSDI'14]
   Joint work with H. Lim, M. Kaminsky, and D. Andersen
2. Improving the performance of a Web server [NSDI'14]
   E. Jeong, S. Woo, M. Jamshed, H. Jeong, S. Ihm, and K. Park

Workload for User-facing Servers

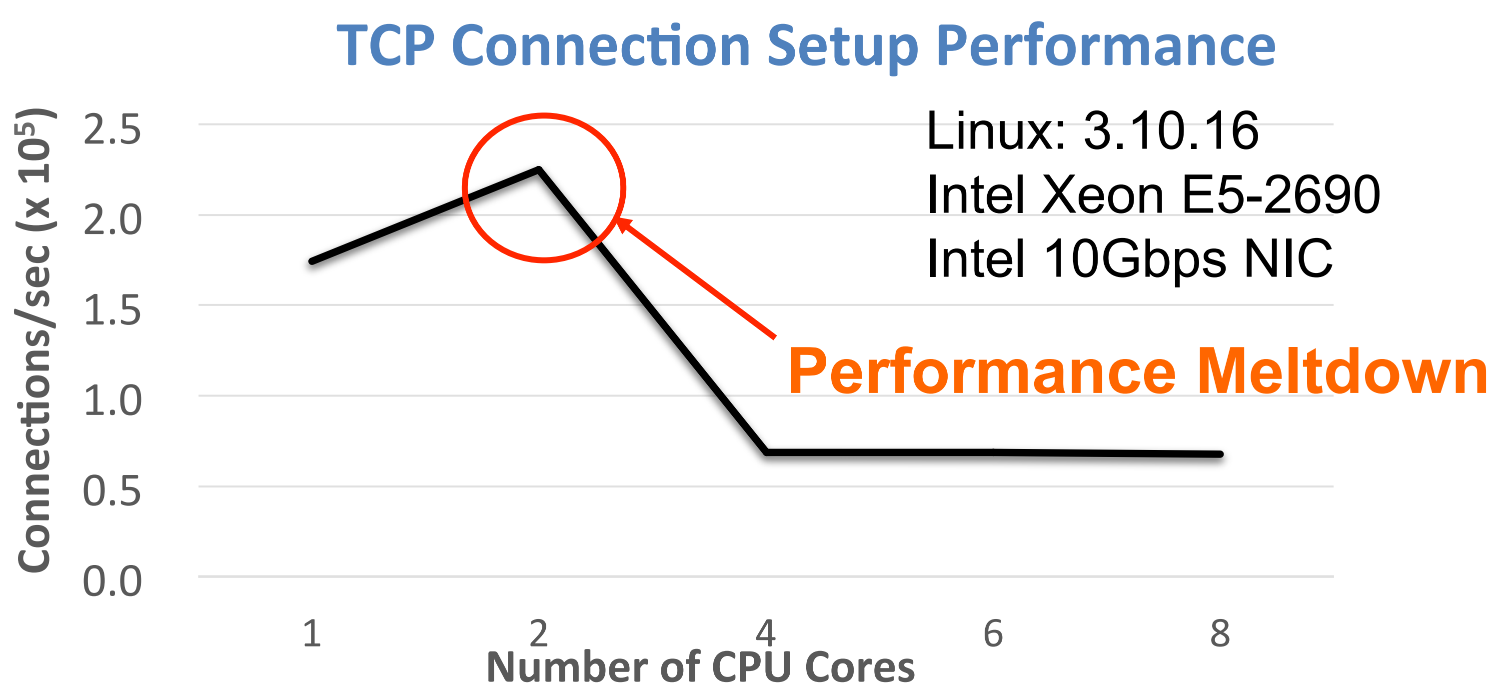
Measurement of TCP flows in commercial cellular backbone [Woo,mobisys'13]



Over 90% (50%) of TCP flows are smaller than 64 KB (4 KB).

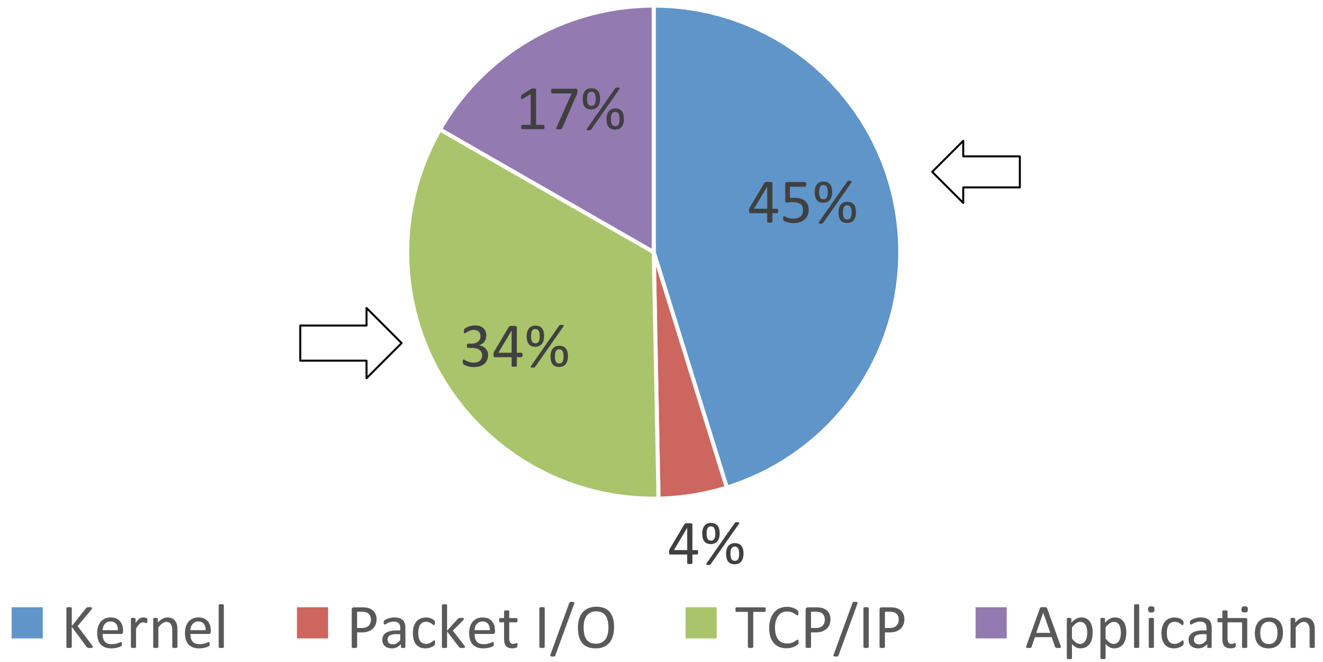
Web Server Performance

- Large transfers: easy to fill up 10 Gbps
- Small transactions: 1.2 Gbps under SpecWeb
- Kernel is not designed well for multicore systems.



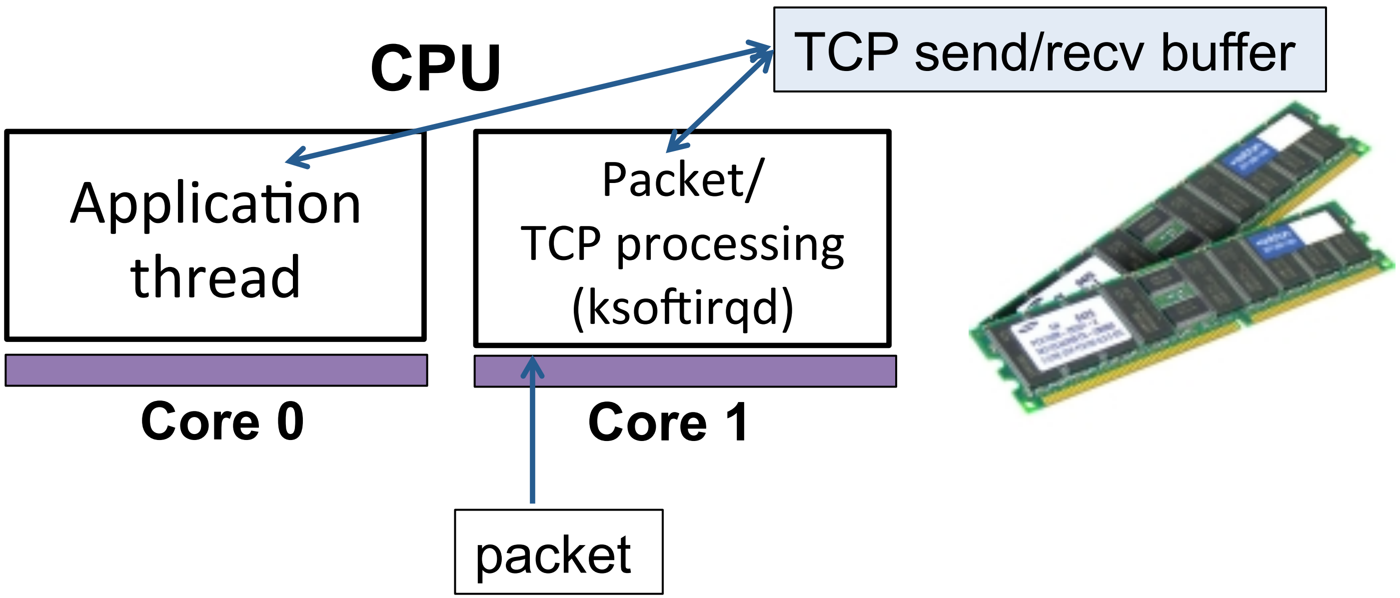
Performance Analysis of a Web Server

83% of CPU usage spent inside kernel!



Inefficiencies in Kernel TCP/IP Stack

1. Lack of connection locality



Inefficiencies in Kernel TCP/IP Stack

1. Lack of connection locality

```
while (1) {
        epoll_wait(…)
        fd = accept(listen_fd, NULL);
        …
        read(fd, buf, 1024);
        …
        write(fd, buf, 1024);
}
```

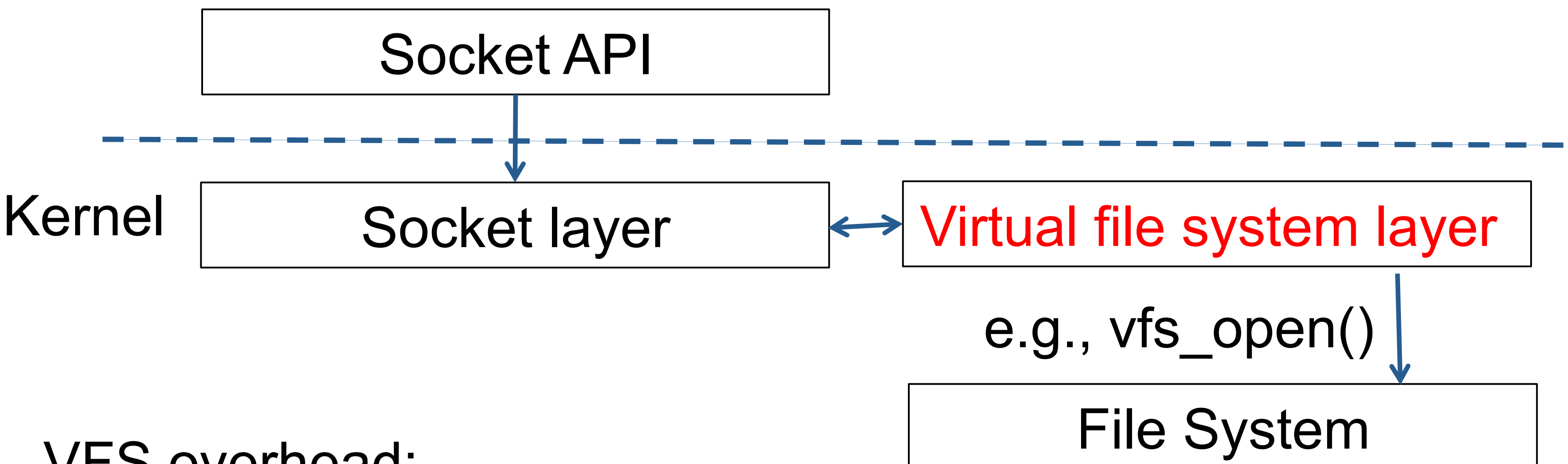
```
while (1) {
        epoll_wait(…)
        fd = accept(listen_fd, NULL);
        …
        read(fd, buf, 1024);
        …
        write(fd, buf, 1024);
}
```

Accept-Affinity[EUROSYS'12]: connection affinity (only)
Linux SO_REUSEPORT option (v3.9.4): per-core listen socket

Inefficiencies in Kernel TCP/IP Stack

2. Shared file descriptor space

fd = accept(listen_fd, NULL)



VFS overhead:
Creates inode for each socket file descriptor.
Finds the lowest available integer [POSIX]

Inefficiencies in Kernel TCP/IP Stack

3. System call overhead (frequent and expensive)

```
while (1) {
        epoll_wait(…)
        fd = accept(listen_fd, NULL);
        …
        read(fd, buf, 1024);
        …
        write(fd, buf, 1024);
}
```

4. Inefficient per-packet processing
    - Per-packet memory allocation/deallocation overhead

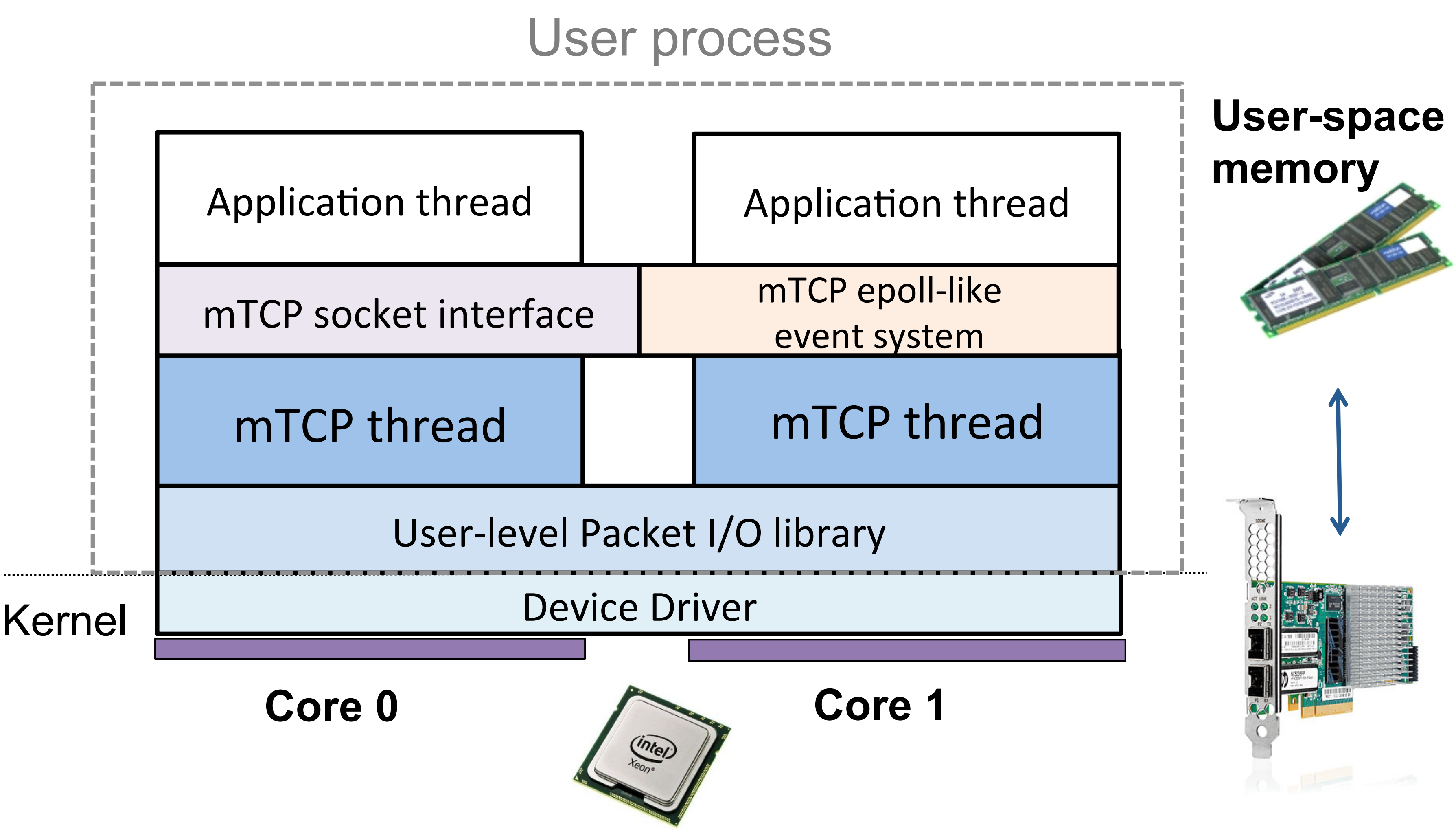
MegaPipe[OSDI'12]: partially address problems 1,2,3

⇒ All prior work reuses kernel's TCP/IP.

mTCP Approach

- mTCP: a high-performance user-level TCP design for multicore systems
- Clean-slate approach to divorce kernel's complexity
  1. Leverage user-level packet I/O
  2. Support multicore-aware flow processing
  3. Provide a user-level socket API

mTCP Overview



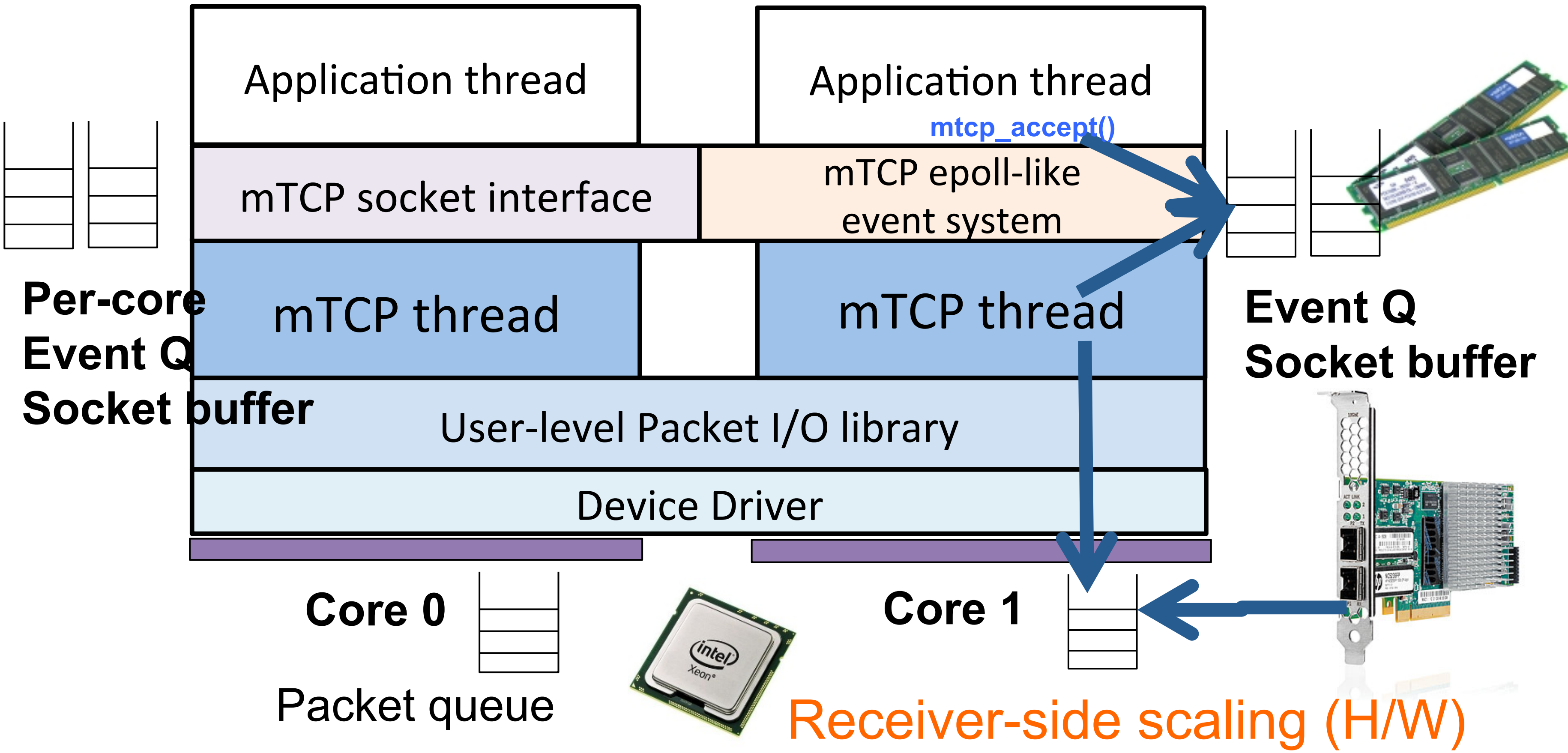
mTCP Overview

Core-affinity (1)

Per-core file descriptor, listen socket (2)

Kernel bypass, No system call (3)

Batched packet processing (4)



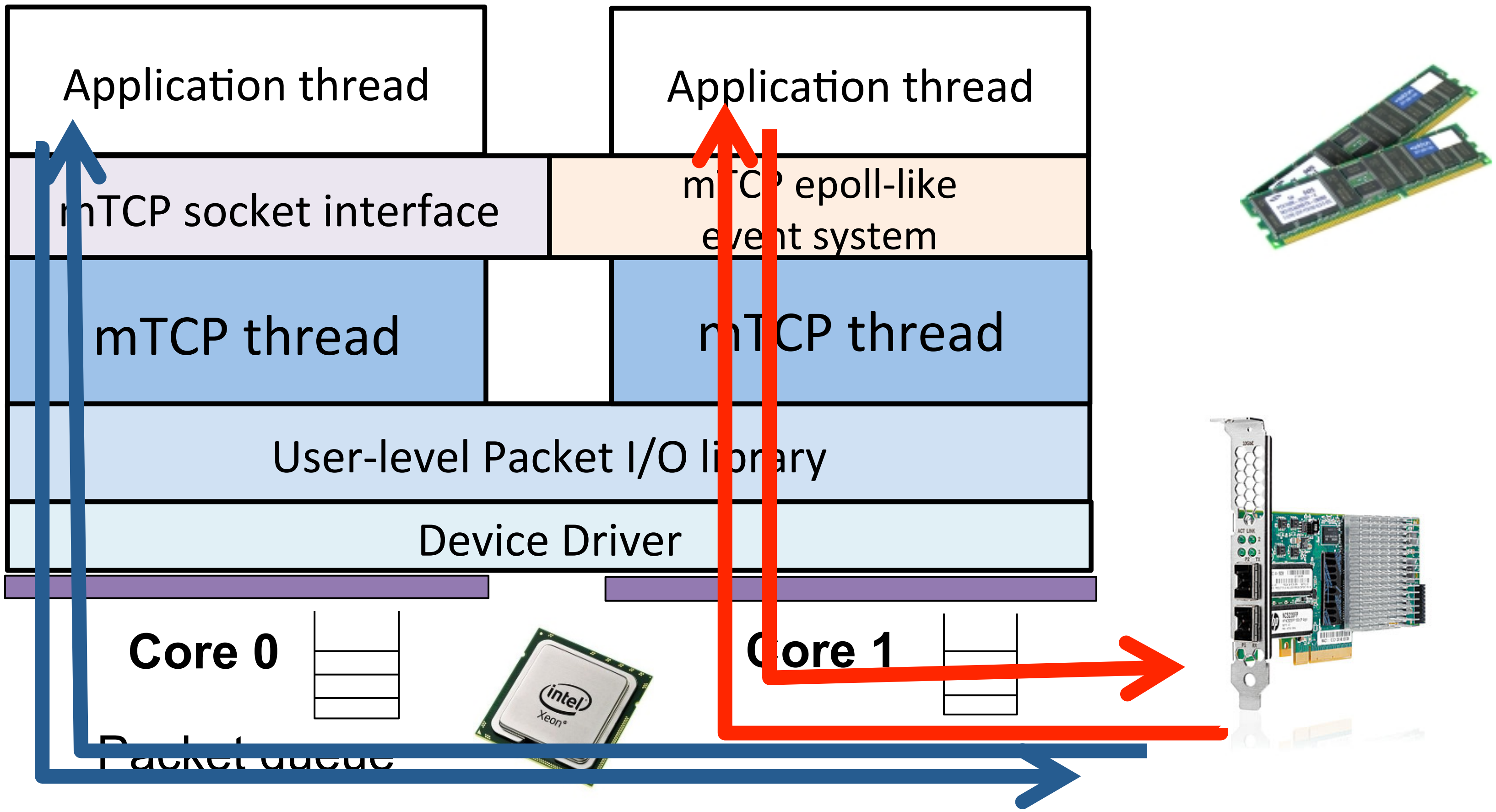
mTCP Overview

Core-affinity (1)

Per-core file descriptor, listen socket (2)

Kernel bypass, No system call (3)

Batched packet processing (4)

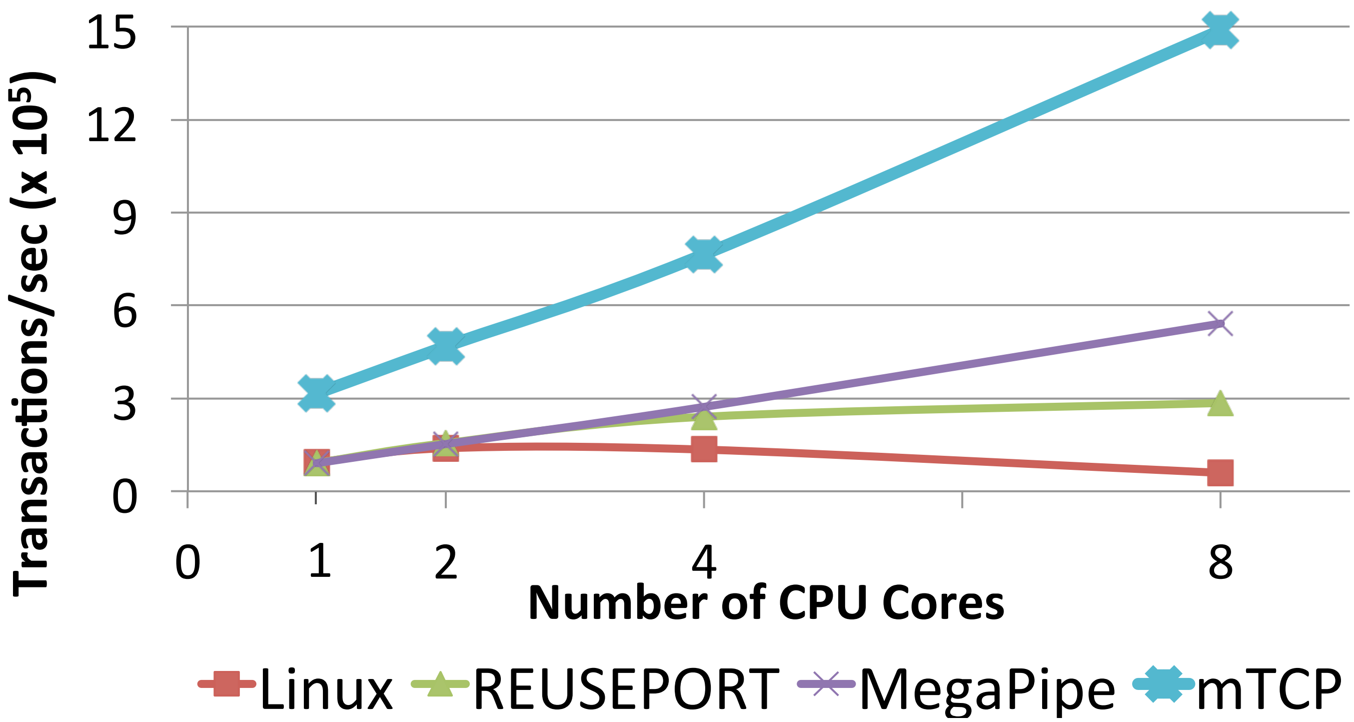


mTCP Design

- Highly scalability on multicore systems
  - 25x faster than latest Linux version
  - 3x faster than MegaPipe
- Easy to use; little porting effort
  - Modified 29 lines of the Apache library (out of 66,493)
- Evaluation
  - HTTP server/client: lighttpd, Apache
  - Web Replayer: replays cellular backbone traffic (Korea)
  - SSL Proxy

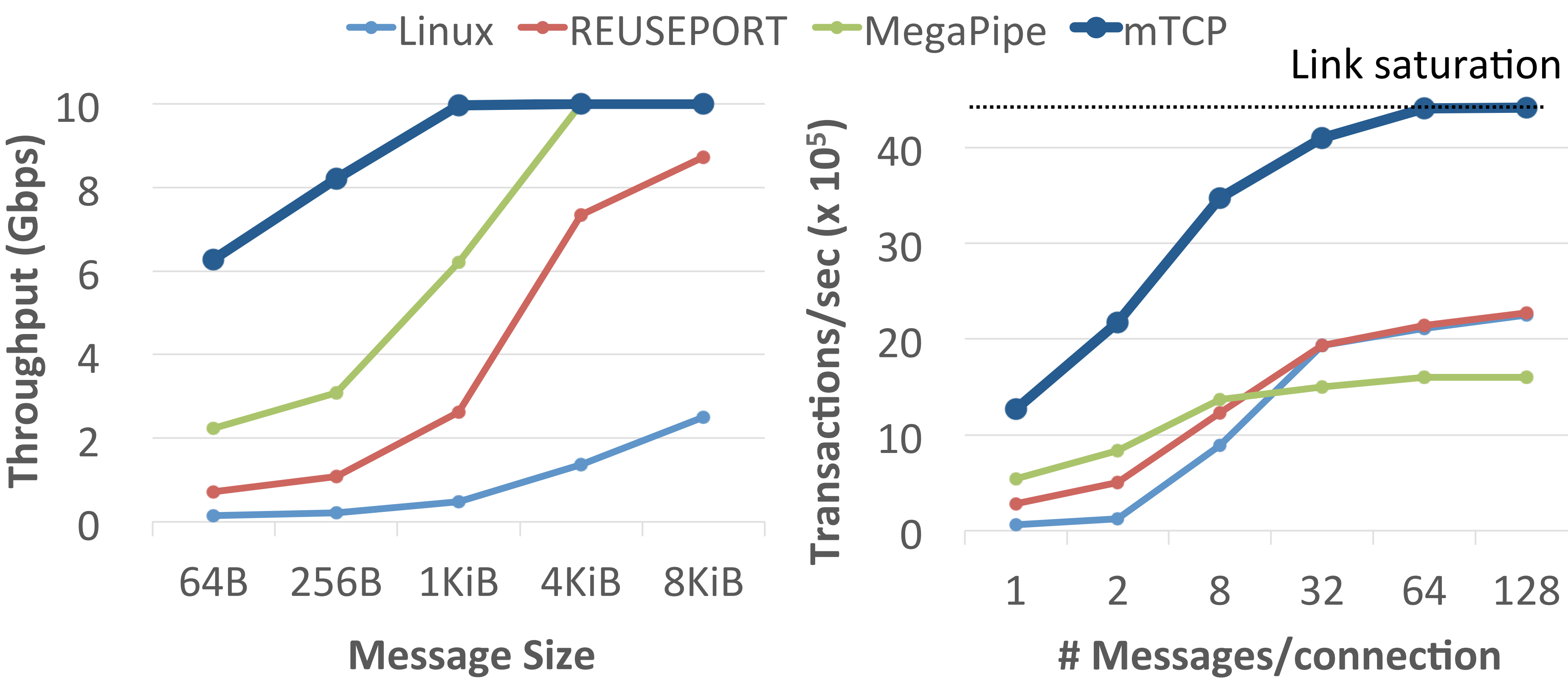
Multicore Scalability

- 64B message per each connection
- Heavy connection, small packet processing overhead
- 25x Linux, 5x REUSEPORT, 3x MegaPipe [OSDI 2012]



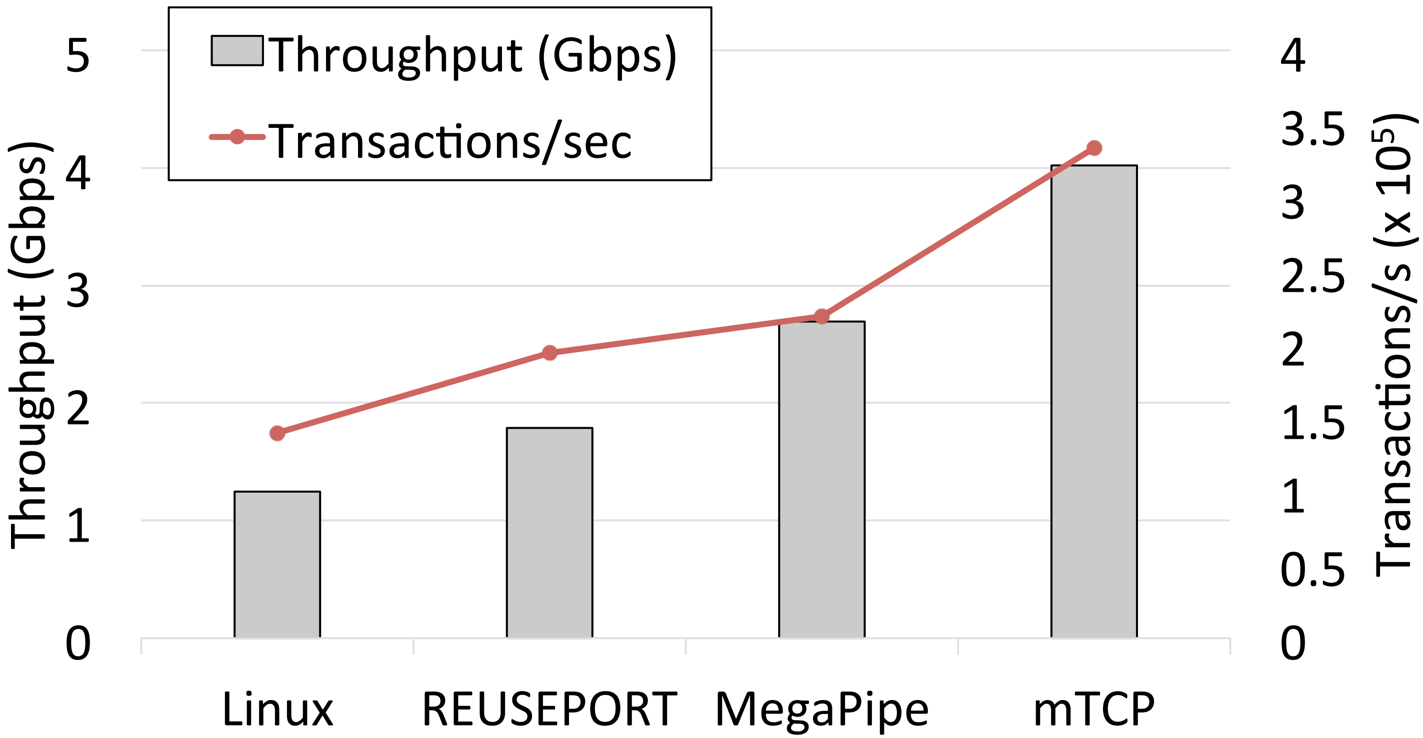
Message Benchmark

- Scaling by message size
- Persistent connection with 64 byte messages



Web Server (Lighttpd) Performance

- SpecWeb2009 static file workload (738B average)
- 3.2x faster than Linux, 1.5x faster than MegaPipe



Summary

- mTCP: a high-performance user-level TCP stack for multicore systems
- Efficiently utilize multicore resources by
  - Eliminating system call overhead
  - Reducing context switch cost by event batching
  - Using per-core resource management
  - Using cache-aware threading
- Achieve high performance scalability
  - Small message transactions: 3x to 25x
  - Existing applications: 33% (SSLShader) to 320% (lighttpd)

Conclusion

- Despite many efforts from academia and industry, there still exists lots of room for innovations for Cloud-based systems and services.
- Essential building blocks for Cloud services can benefit from a holistic, multicore-aware design that leverages the underlying H/W and that carefully considers the workload.
- More research is ahead in bringing new applications to the Cloud.

Reference

- [DPDK] http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/packet-processing-is-enhanced-with-software-from-intel-dpdk.html
- [FacebookMeasurement] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. In *Proc. SIGMETRICS 2012.*
- [Masstree] Yandong Mao, Eddie Kohler, and Robert Tappan Morris. Cache Craftiness for Fast Multicore Key-Value Storage. In *Proc. EuroSys 2012.*
- [MemC3] Bin Fan, David G. Andersen, and Michael Kaminsky. MemC3: Compact and Concurrent MemCache with Dumber Caching and Smarter Hashing. In *Proc. NSDI 2013.*
- [Memcached] http://memcached.org/
- [RAMCloud] Diego Ongaro, Stephen M. Rumble, Ryan Stutsman, John Ousterhout, and Mendel Rosenblum. Fast Crash Recovery in RAMCloud. In *Proc. SOSP 2011.*