KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC

Bojie Li[1,2] Zhenyuan Ruan[1,3] Wencong Xiao[1,4] Yuanwei Lu[1,2]
Yongqiang Xiong[1] Andrew Putnam[1] Enhong Chen[2]
Lintao Zhang[1]

[1]Microsoft Research [2]USTC [3]UCLA [4]Beihang University

Microsoft

Key-Value Store in Data Centers

Web Cache

MEMCACHED

redis

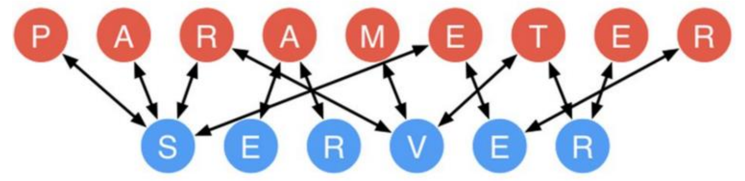
Key		Value
User ID	⟶	Recent Tweets
SQL	⟶	Query result
Session	⟶	Browser cookies

Key-Value Store in Data Centers

Web Cache

Key	Value
User ID →	Recent Tweets
SQL →	Query result
Session →	Browser cookies

Shared Data Structure

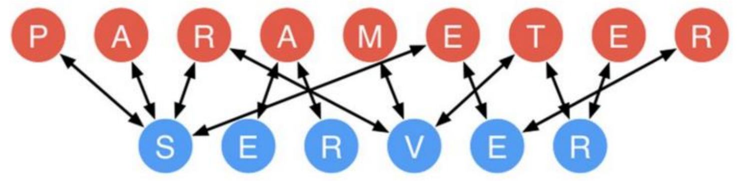


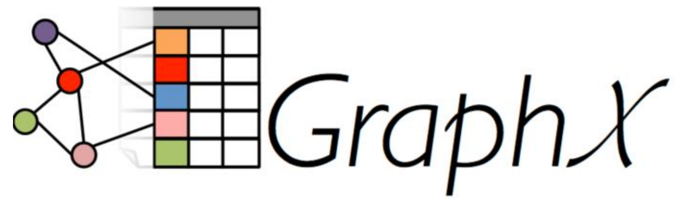
Key	Value
Graph node →	List of edges
Feature →	Weight
Shard →	Sequence number

More demanding workload

Key-Value Store in Data Centers

Shared Data Structure





Design Goals

1. High throughput
2. Low tail latency
3. Write intensive
4. Vector operations
5. Atomic operations

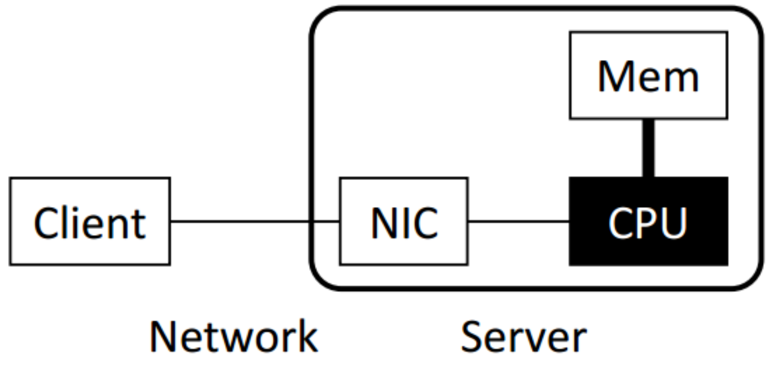
Key		Value
Graph node	→	List of edges
Feature	→	Weight
Shard	→	Sequence number

Graph node and Feature: 4. Vector operations

Shard: 5. Atomic operations

More demanding workload

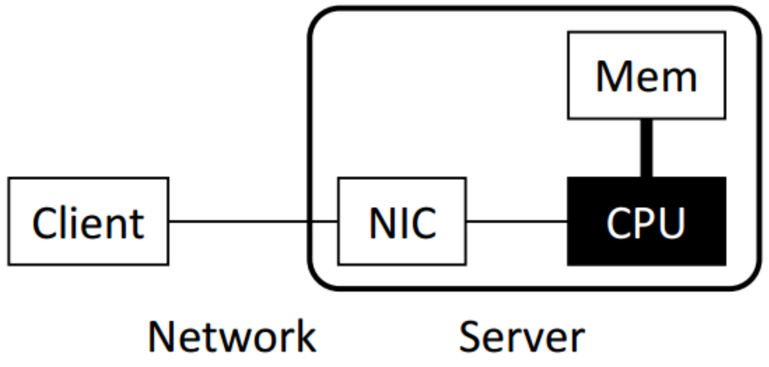
Key-Value Store Architectures



Software (Kernel TCP/IP)

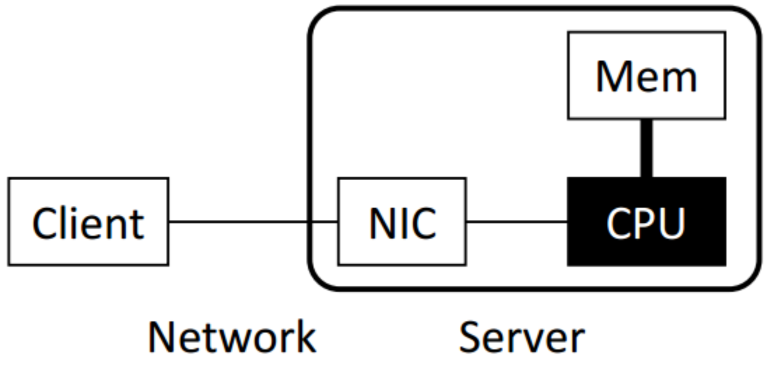
Bottleneck: Network stack in OS
(~300 Kops per core)

Key-Value Store Architectures



Software (Kernel TCP/IP)

Bottleneck: Network stack in OS
(~300 Kops per core)



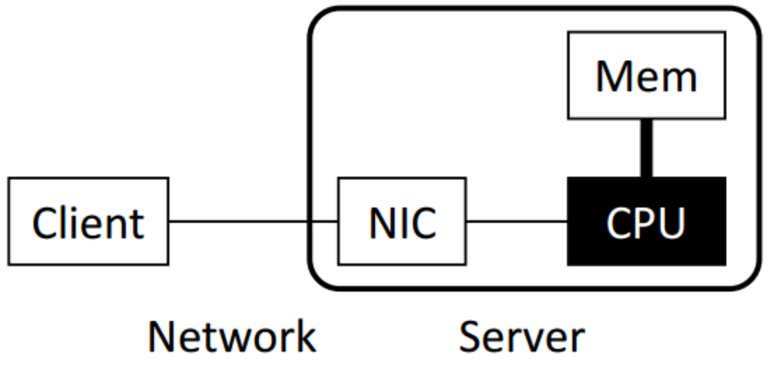
Software (Kernel Bypass)

e.g. DPDK, mtcp, libvma, two-sided RDMA

Bottlenecks: CPU random memory access and KV operation computation

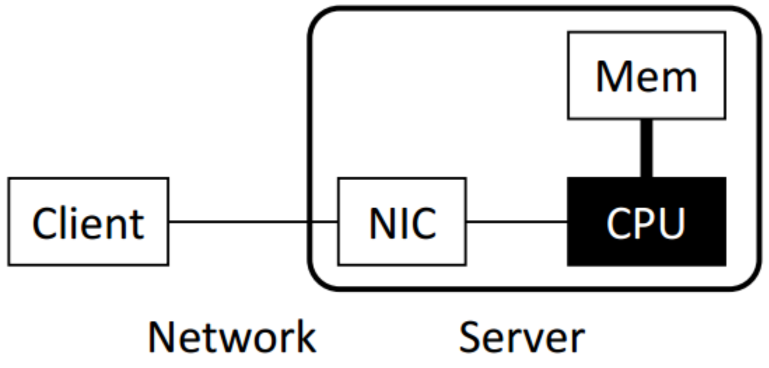
(~5 Mops per core)

Key-Value Store Architectures



Software (Kernel TCP/IP)

Bottleneck: Network stack in OS
(~300 Kops per core)

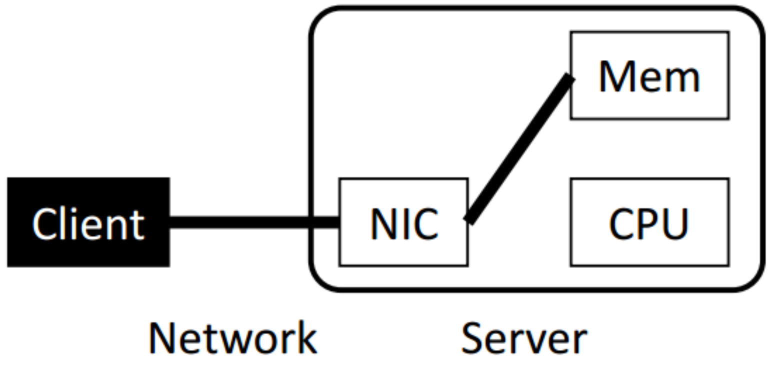


Software (Kernel Bypass)

e.g. DPDK, mtcp, libvma, two-sided RDMA

Bottlenecks: CPU random memory access and KV operation computation

(~5 Mops per core)

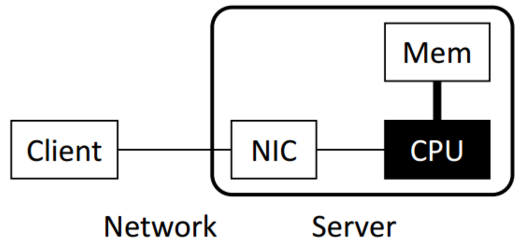


One-sided RDMA

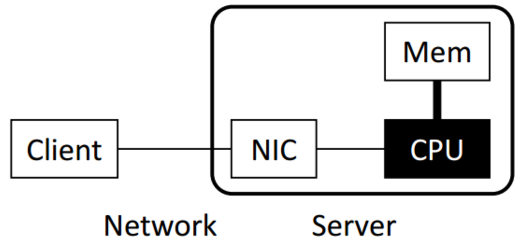
Communication overhead: multiple round-trips per KV operation (fetch index, data)

Synchronization overhead: write operations

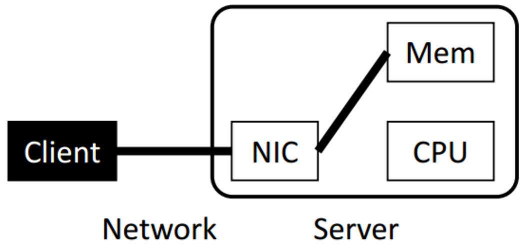
Key-Value Store Architectures



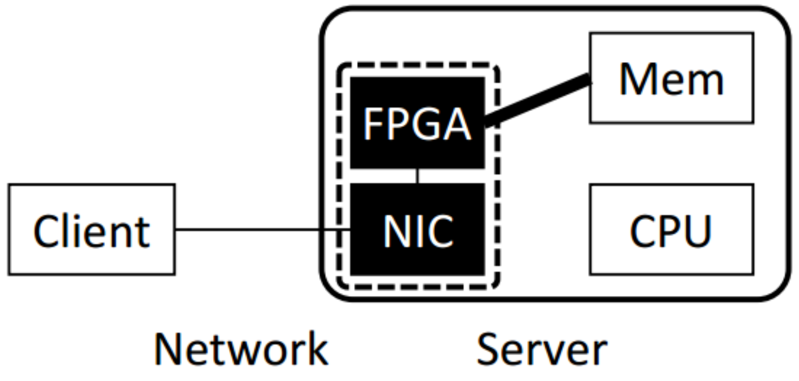
Software (Kernel TCP/IP)



Software (Kernel Bypass)



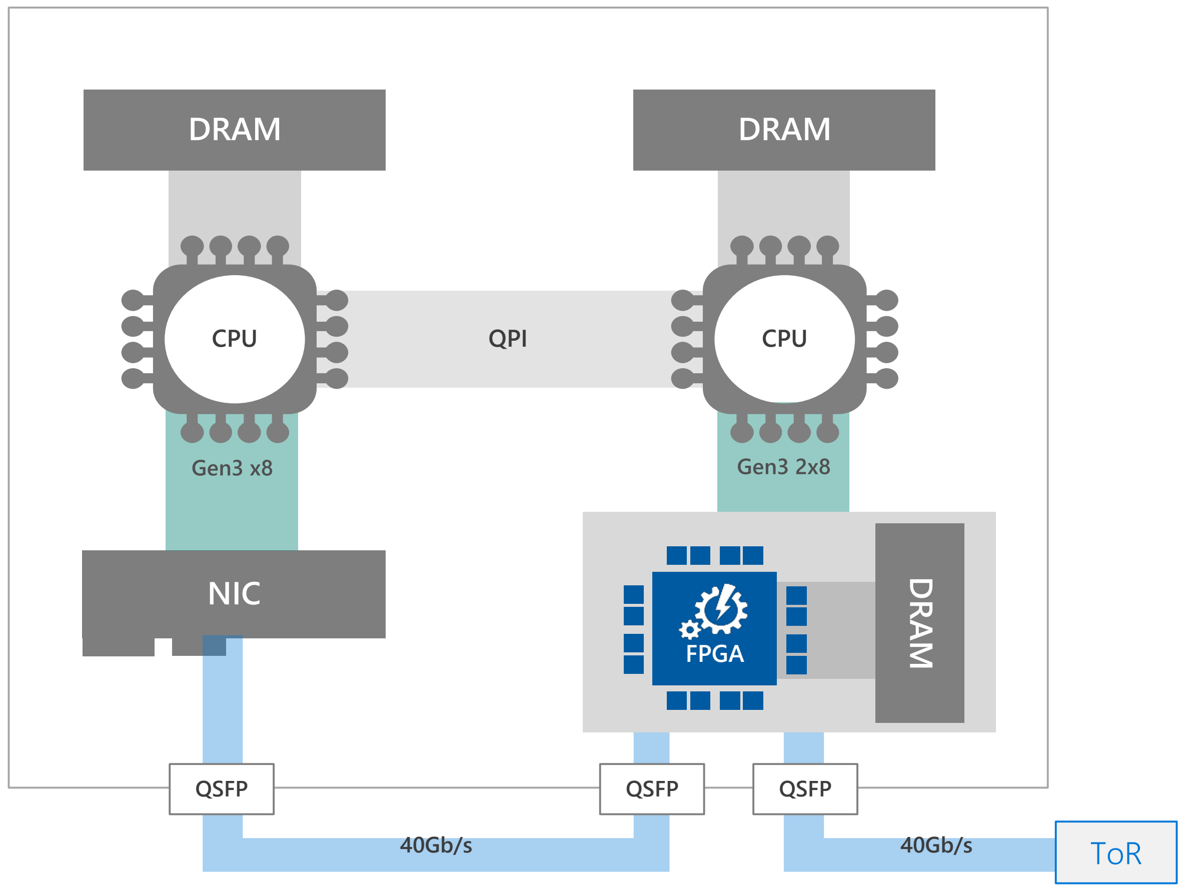
One-sided RDMA



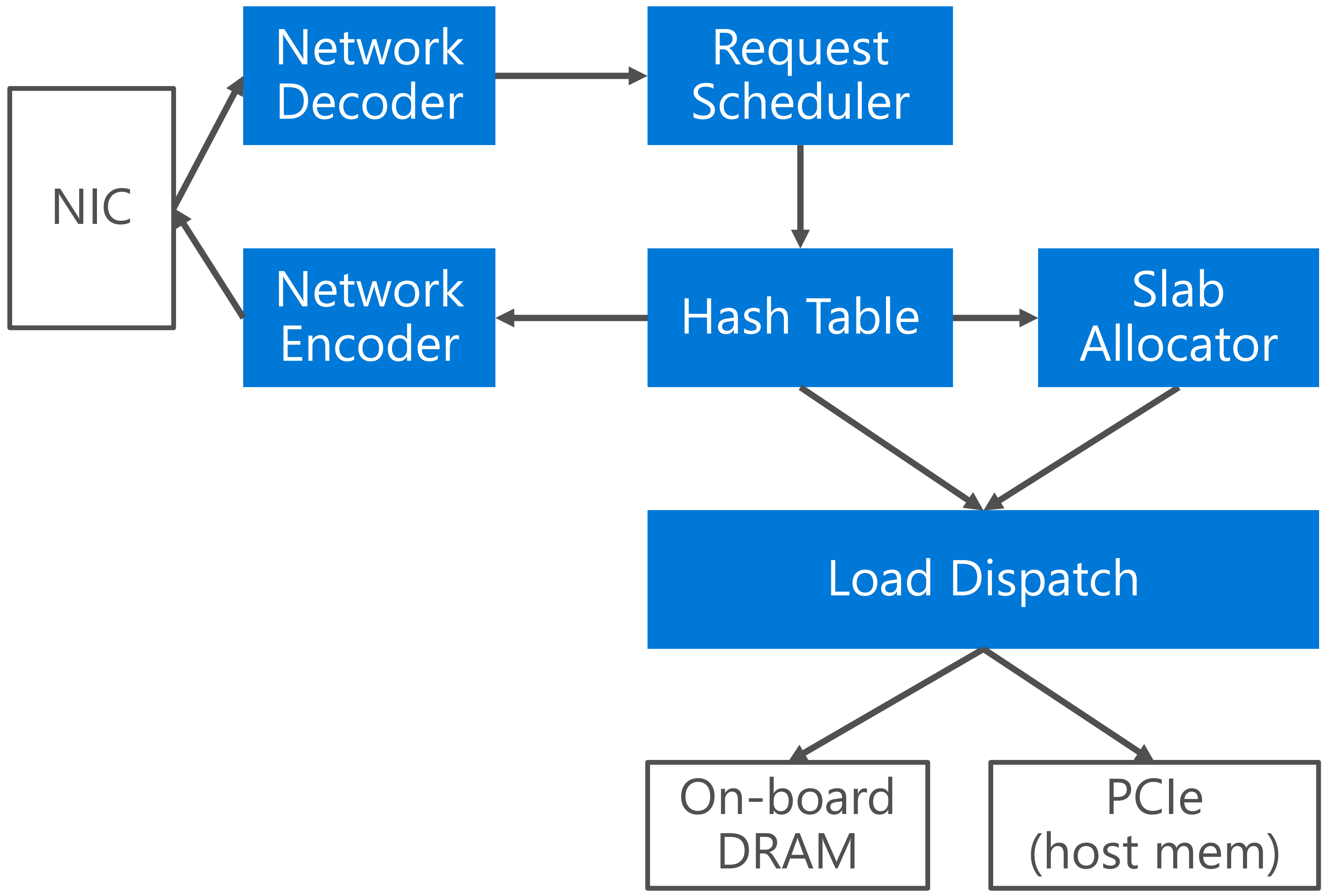
KV-Direct FPGA + NIC

Offload KV processing on CPU to Programmable NIC

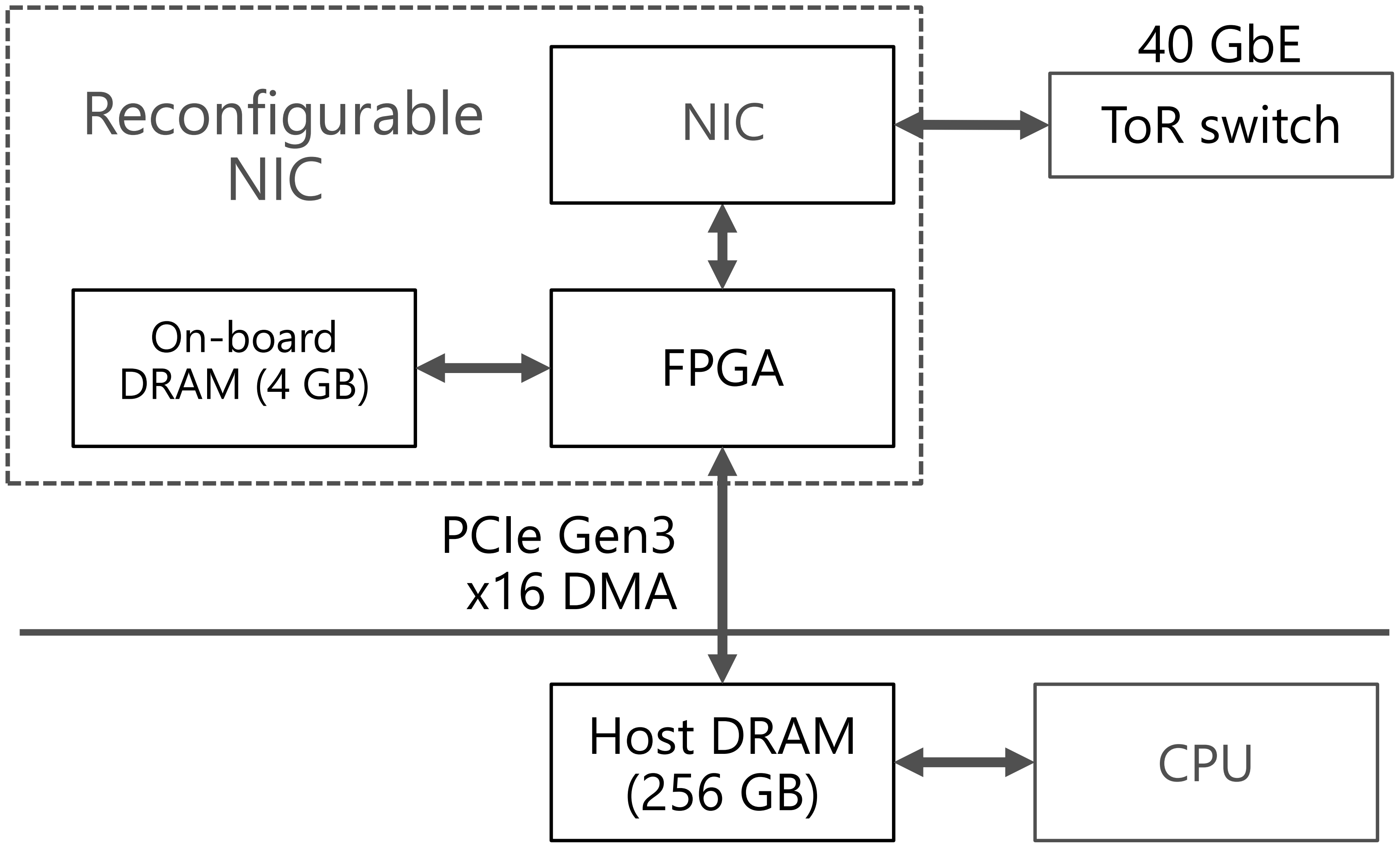
A Commodity Server with SmartNIC



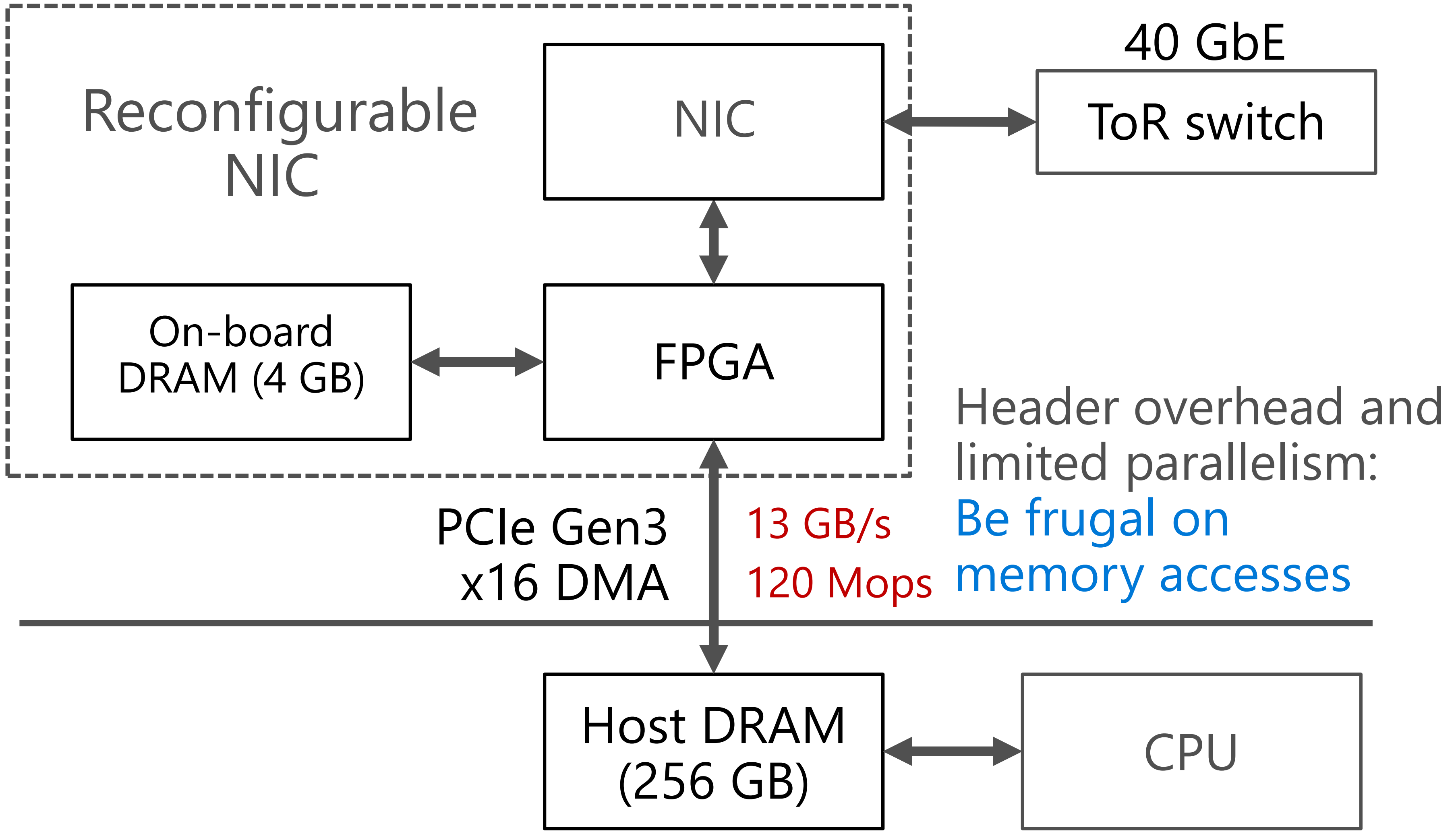
KV-Direct Architecture



Performance Challenges

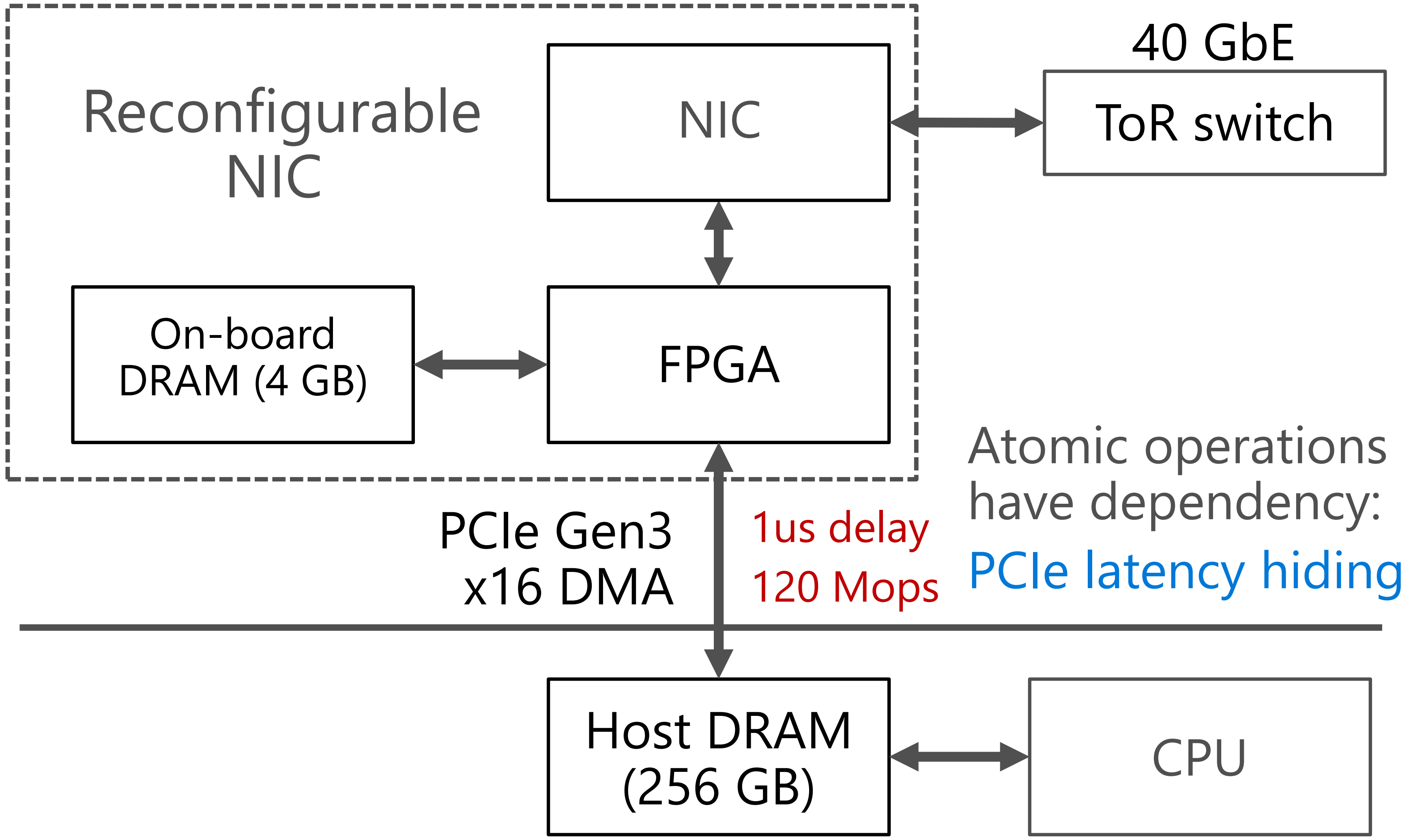


Performance Challenges



Header overhead and limited parallelism:
Be frugal on memory accesses

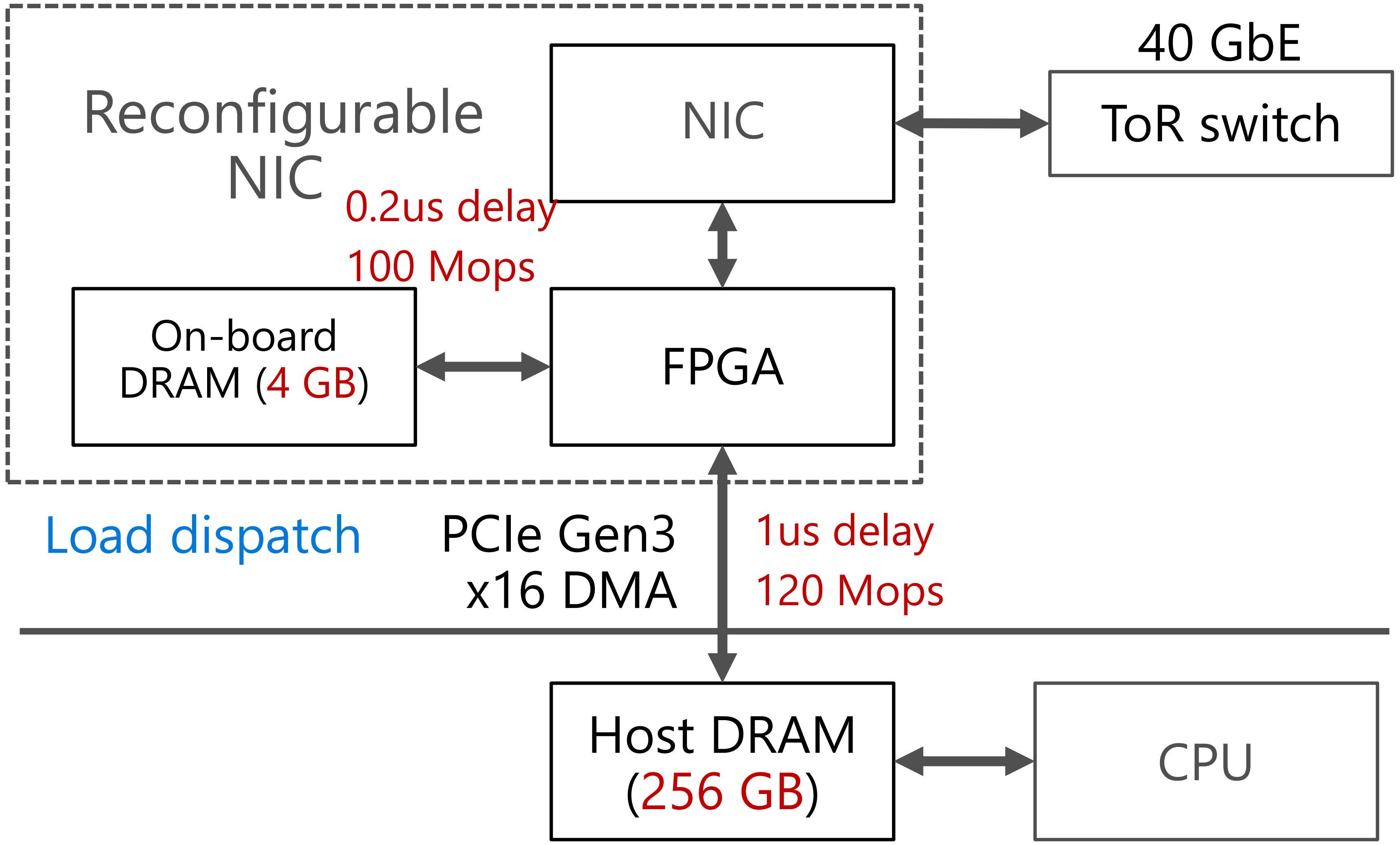
Performance Challenges



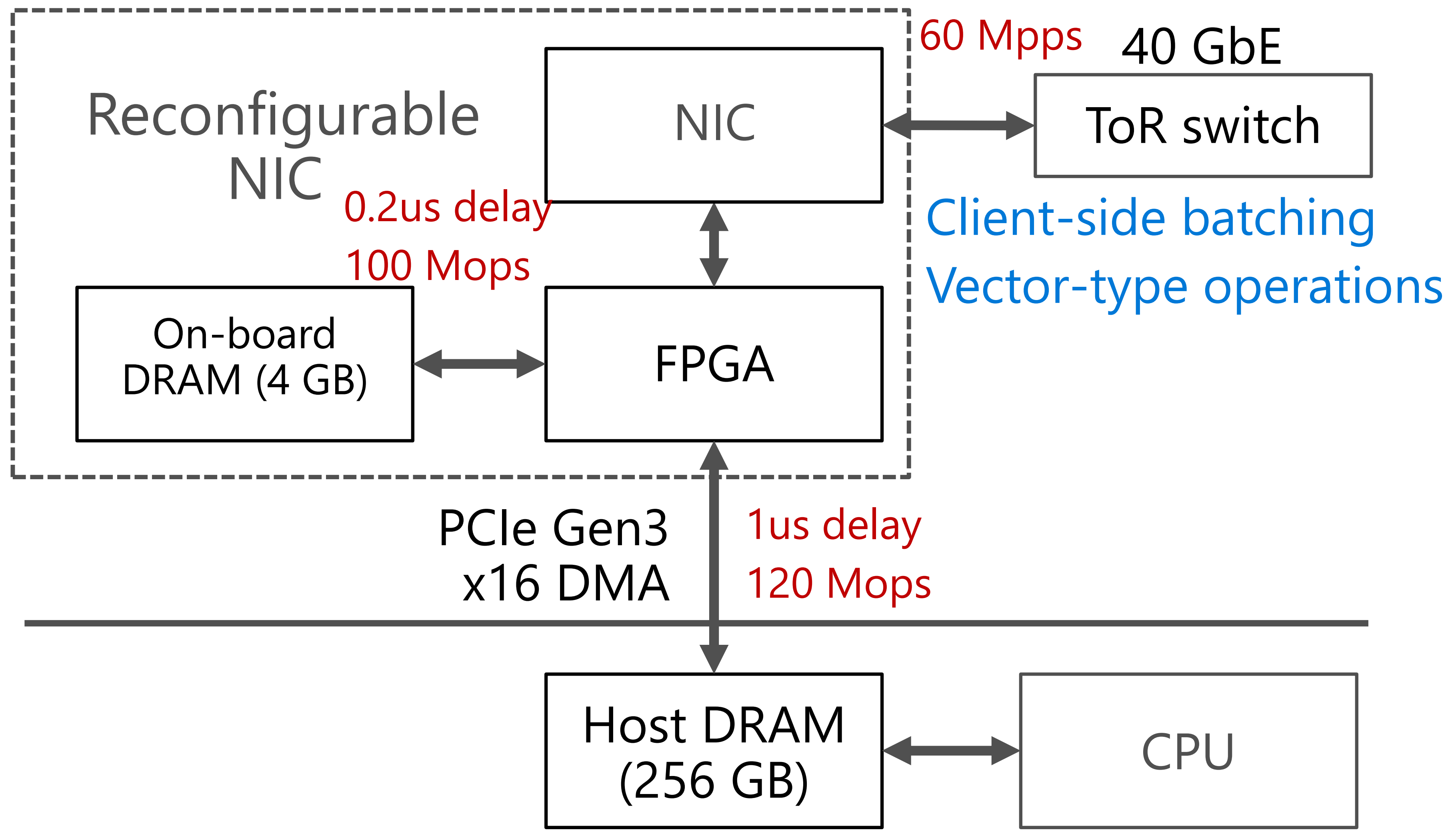
Atomic operations have dependency:

PCIe latency hiding

Performance Challenges



Performance Challenges



Client-side batching

Vector-type operations

Design Principles

1. Be frugal on memory accesses for both GET and PUT
2. Hide memory access latency
3. Leverage throughput of both on-board and host memory
4. Offload simple client computation to server

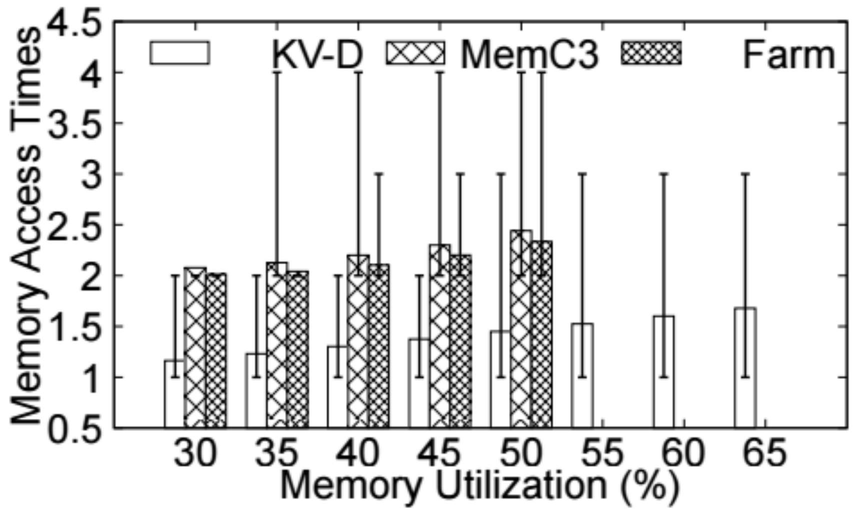
Be Frugal on Memory Accesses

Hash table: minimal access per GET & PUT

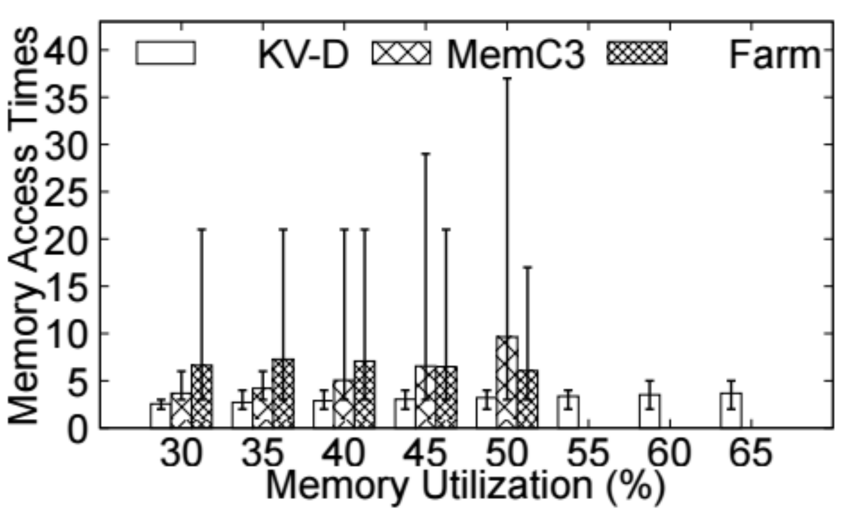
Cuckoo hashing: constant GET, sacrifice PUT.

Log-structured memory: fast PUT, sacrifice GET.

Our choice: Bucketized chaining: Close to 1 per GET, 2 per PUT



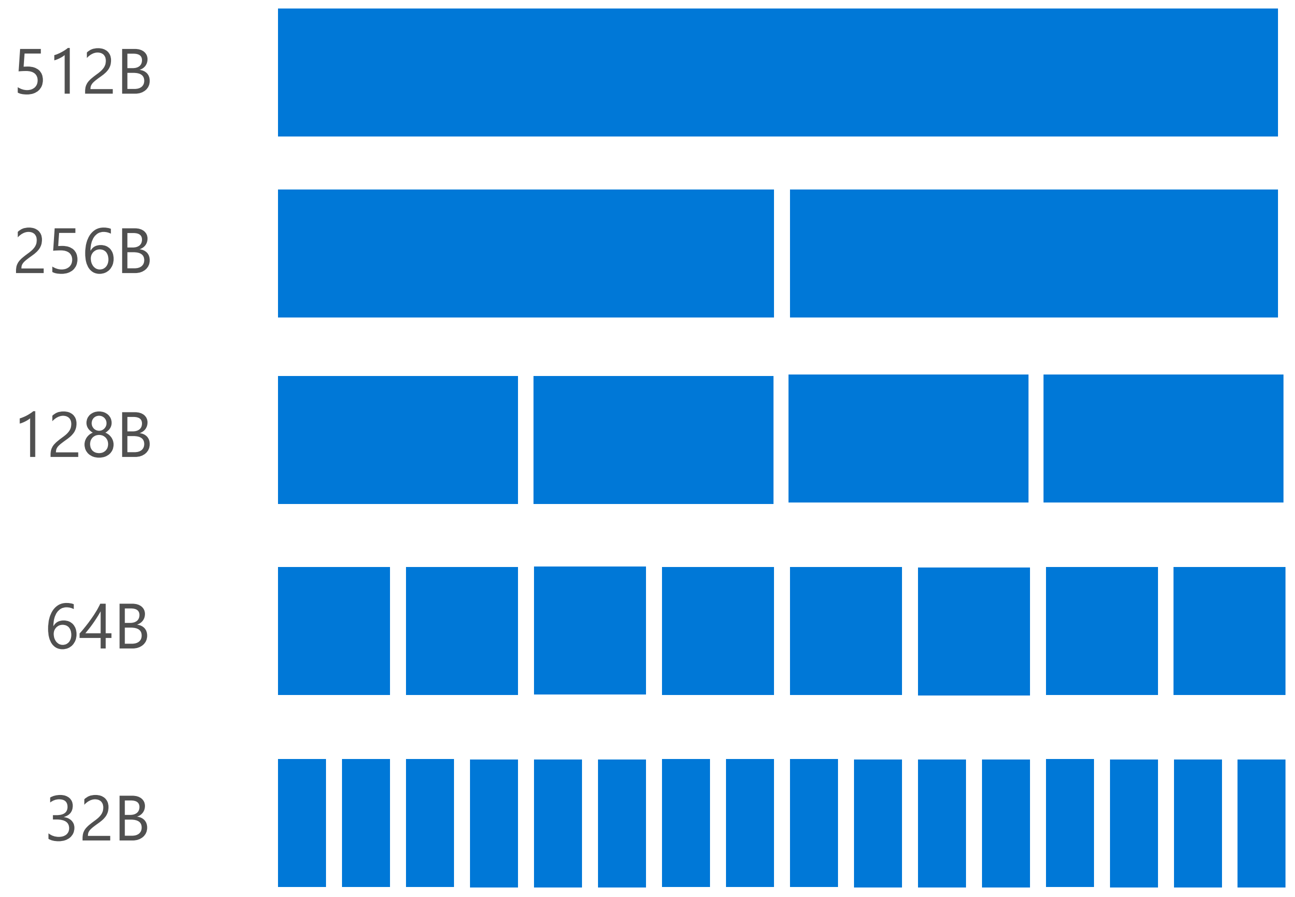
(a) 10B GET.



(b) 10B PUT.

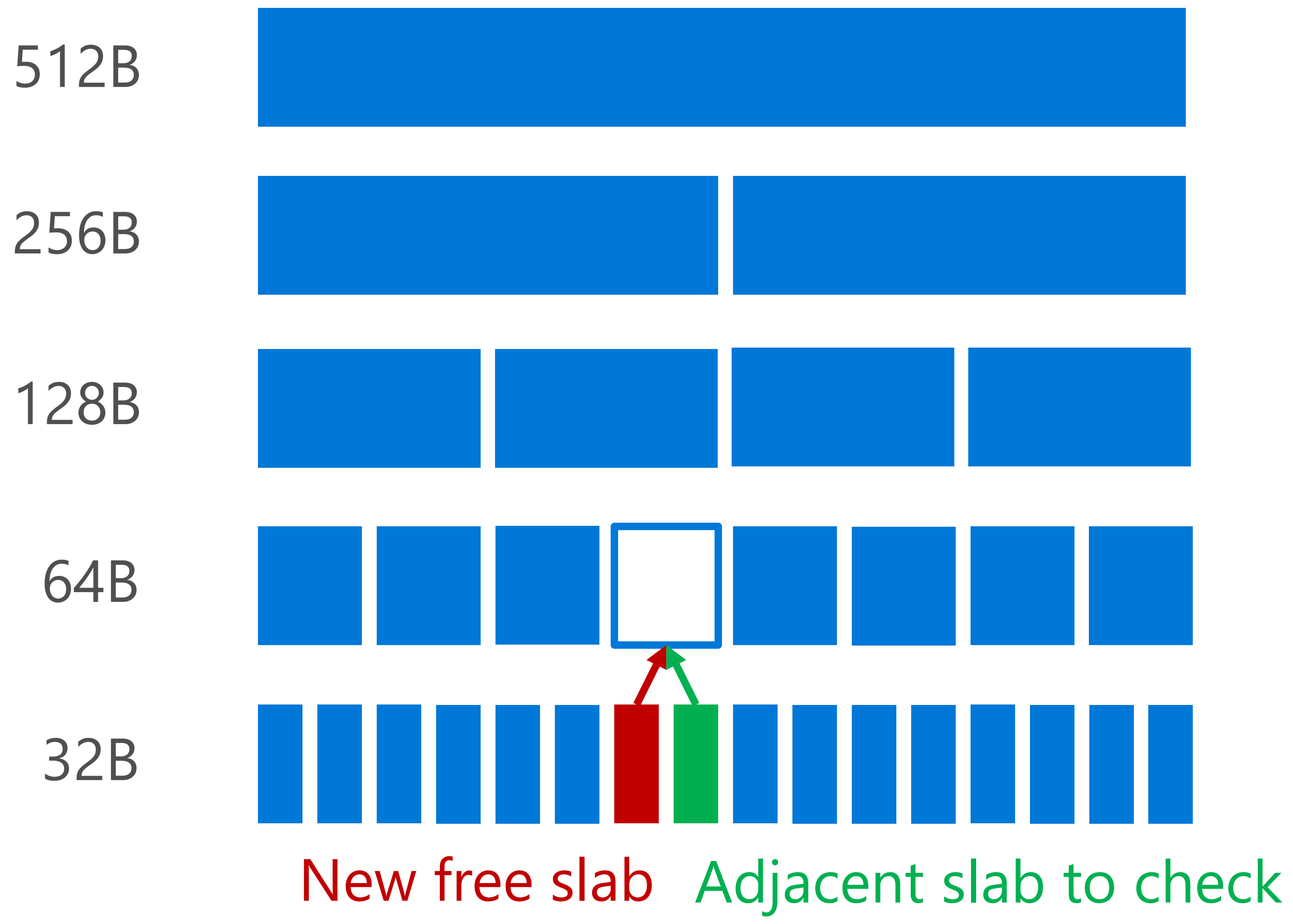
Be frugal on memory accesses

Slab allocator for variable-sized KVs



Be frugal on memory accesses

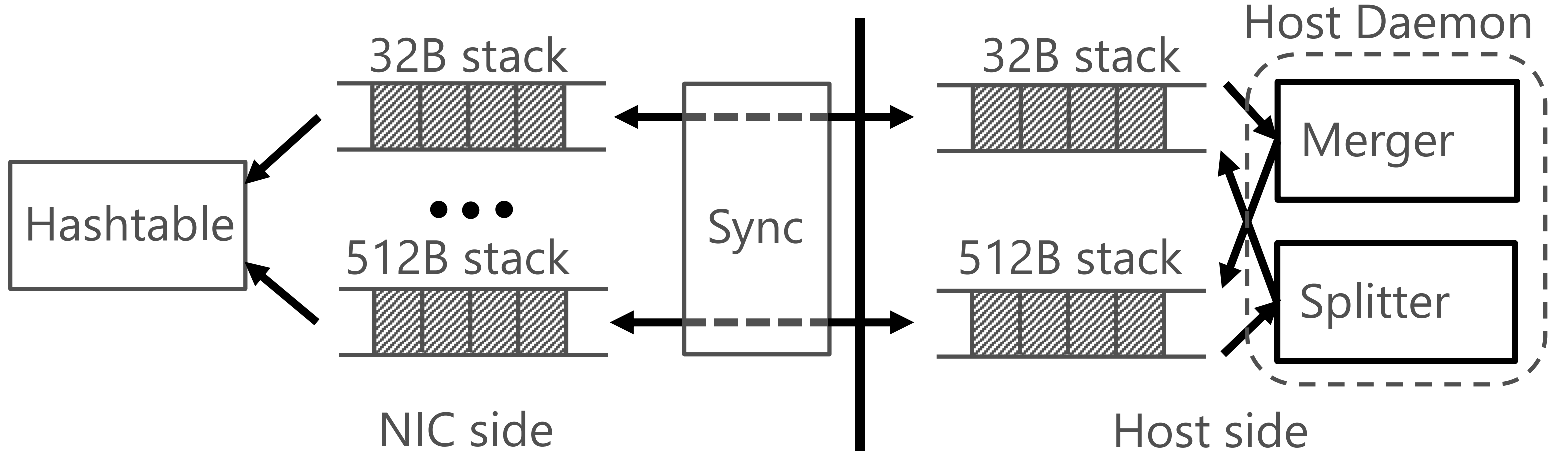
Random memory access to merge free slabs



Be Frugal on Memory Accesses

Solution: lazy slab merging on CPU

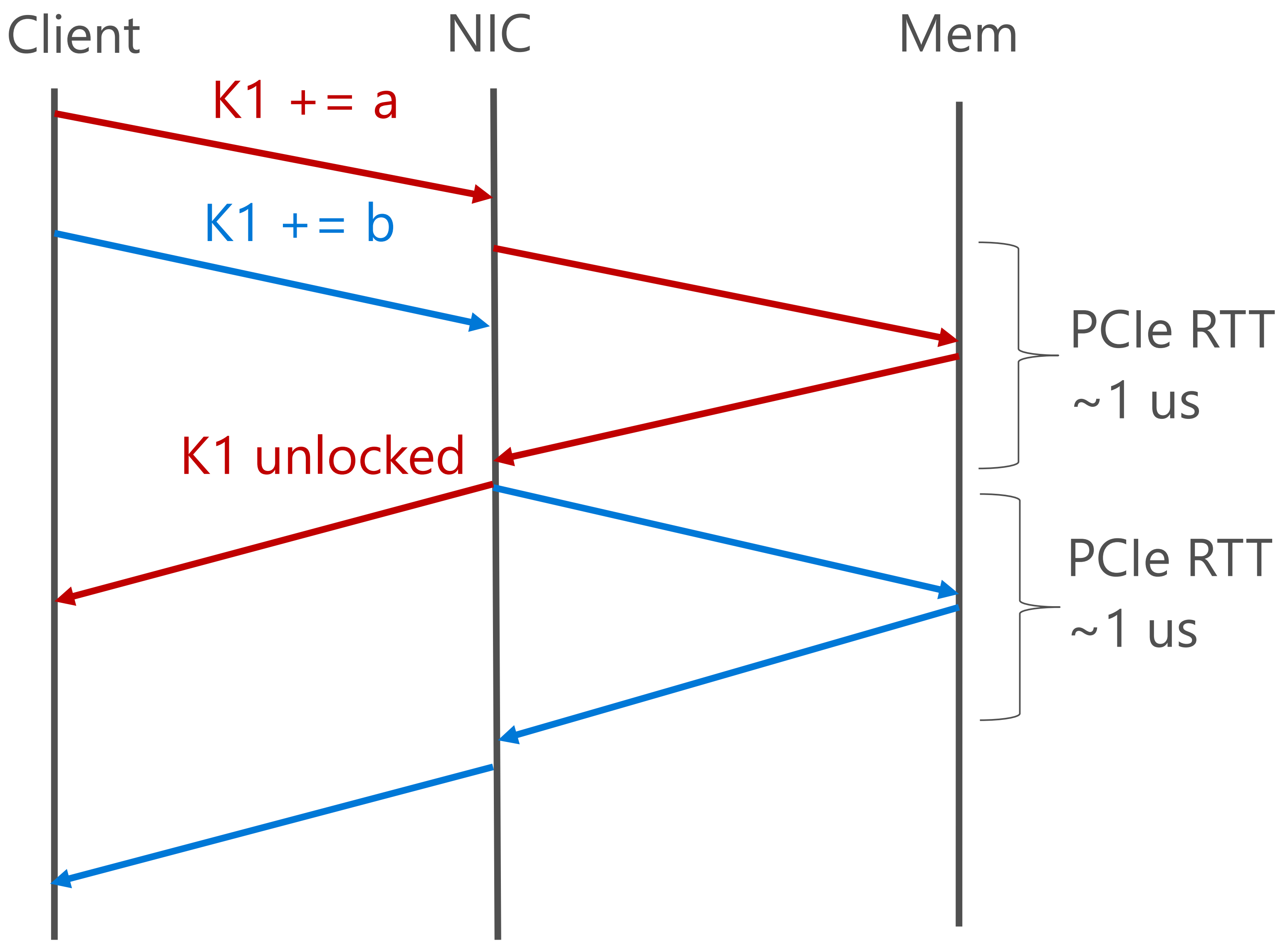
Worst case: 0.07 amortized DMA operations per (de)allocation.



Design Principles

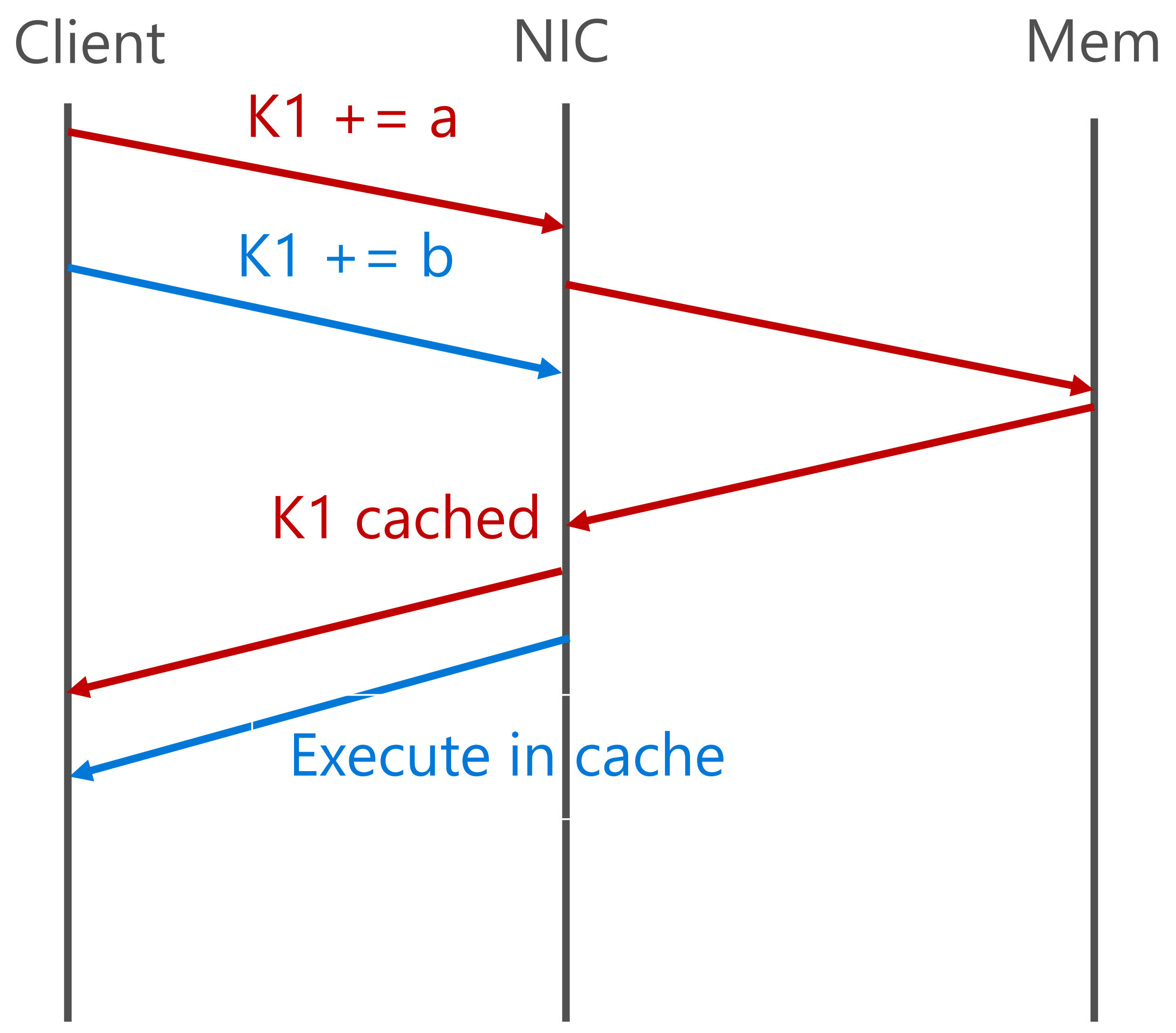
1. Be frugal on memory accesses for both GET and PUT
2. Hide memory access latency
3. Leverage throughput of both on-board and host memory
4. Offload simple client computation to server

Memory dependency

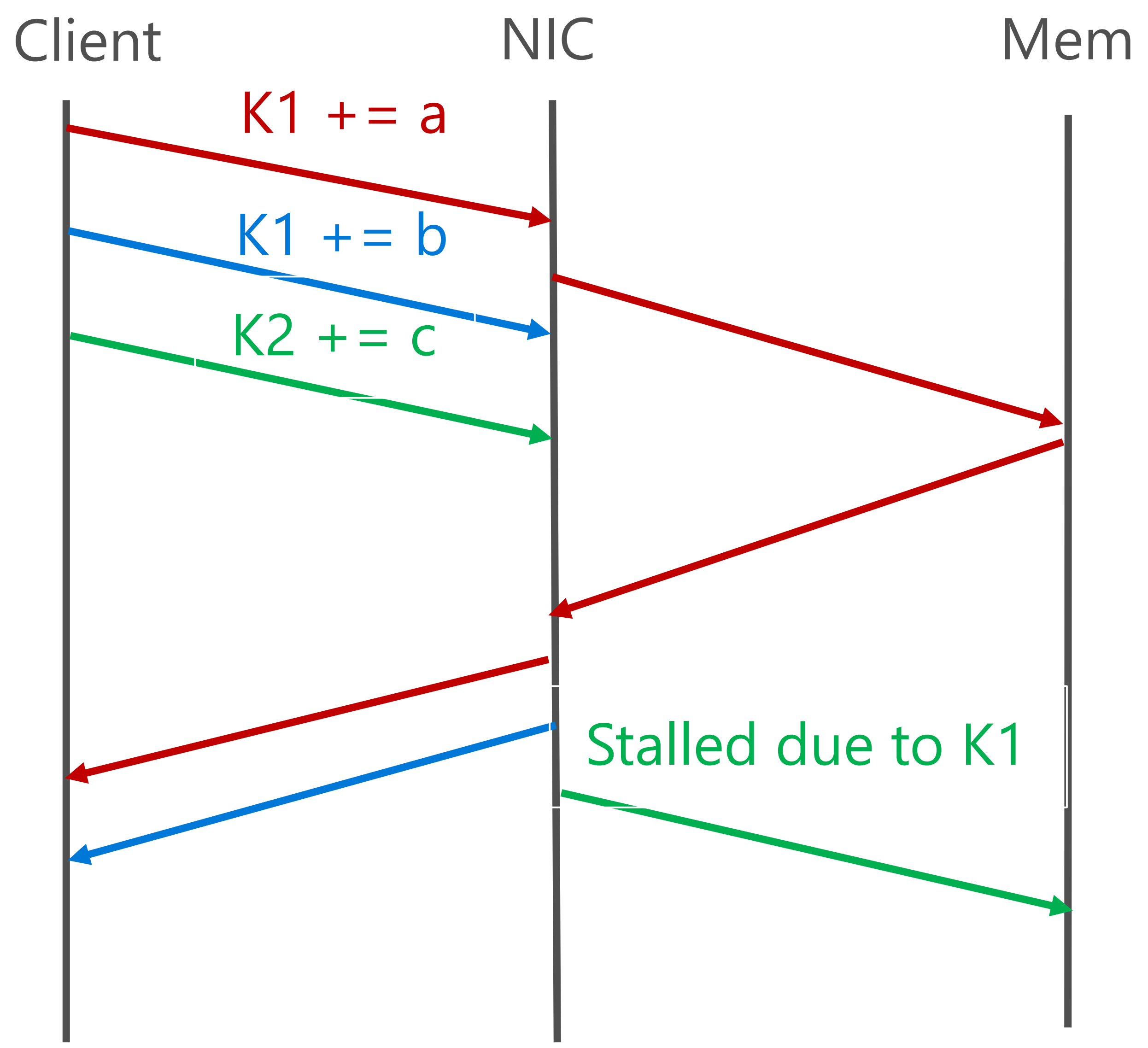


Reservation Station

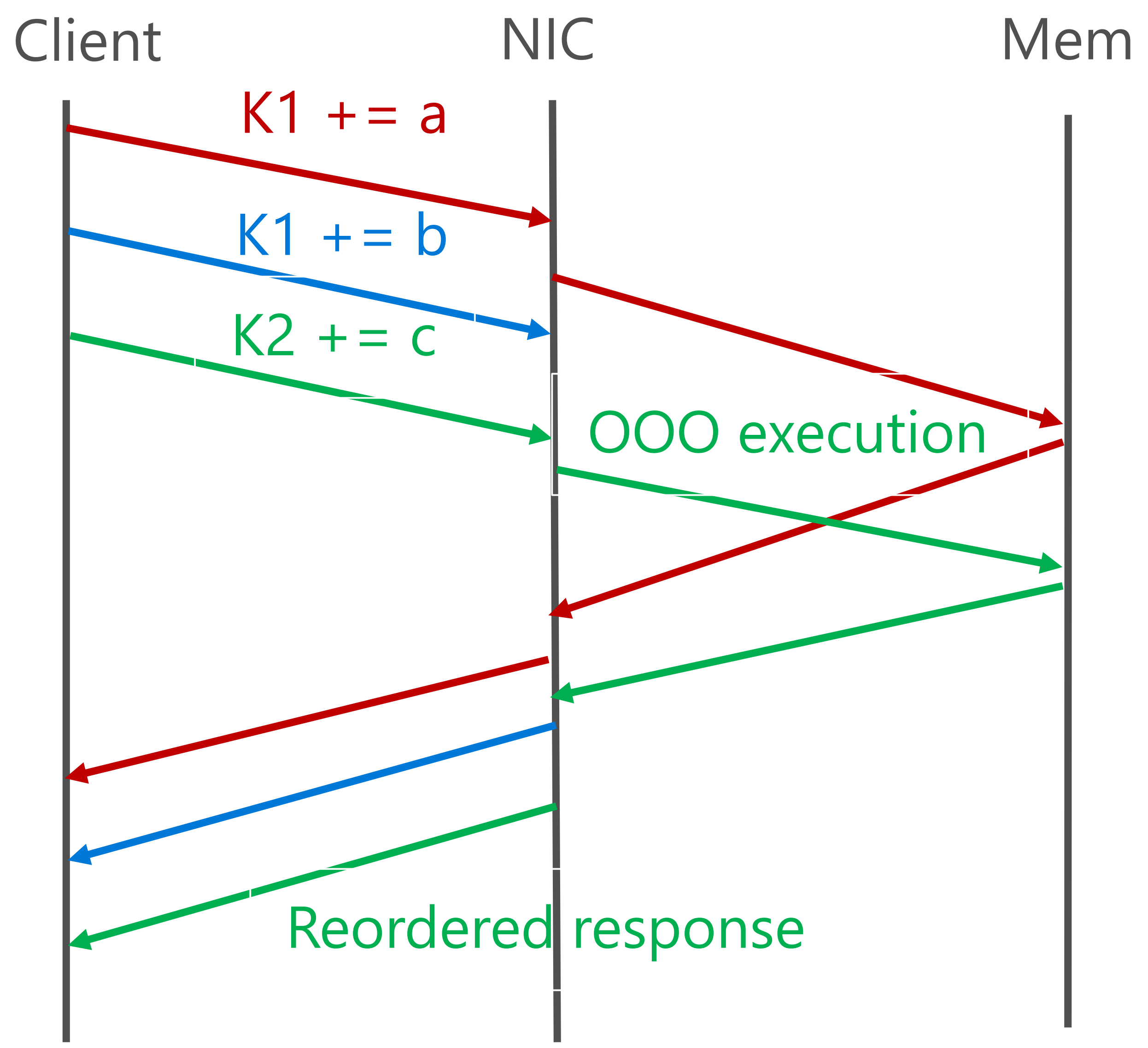
Cache most frequently accessed KVs



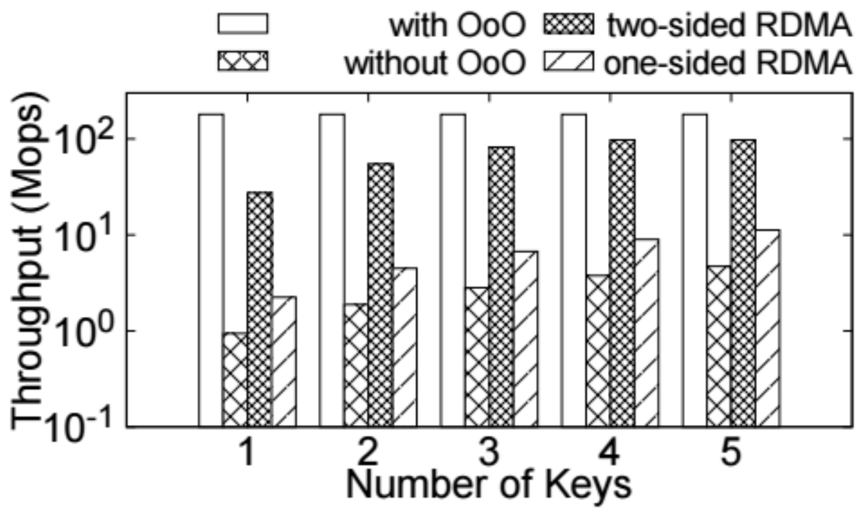
Pipeline stall



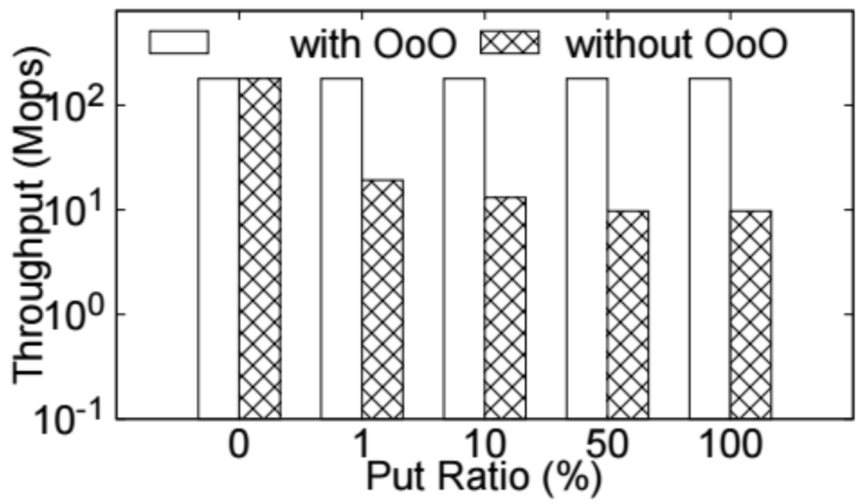
Out-of-order execution



Out-of-order Execution



(a) Atomics.



(b) Long-tail workload.

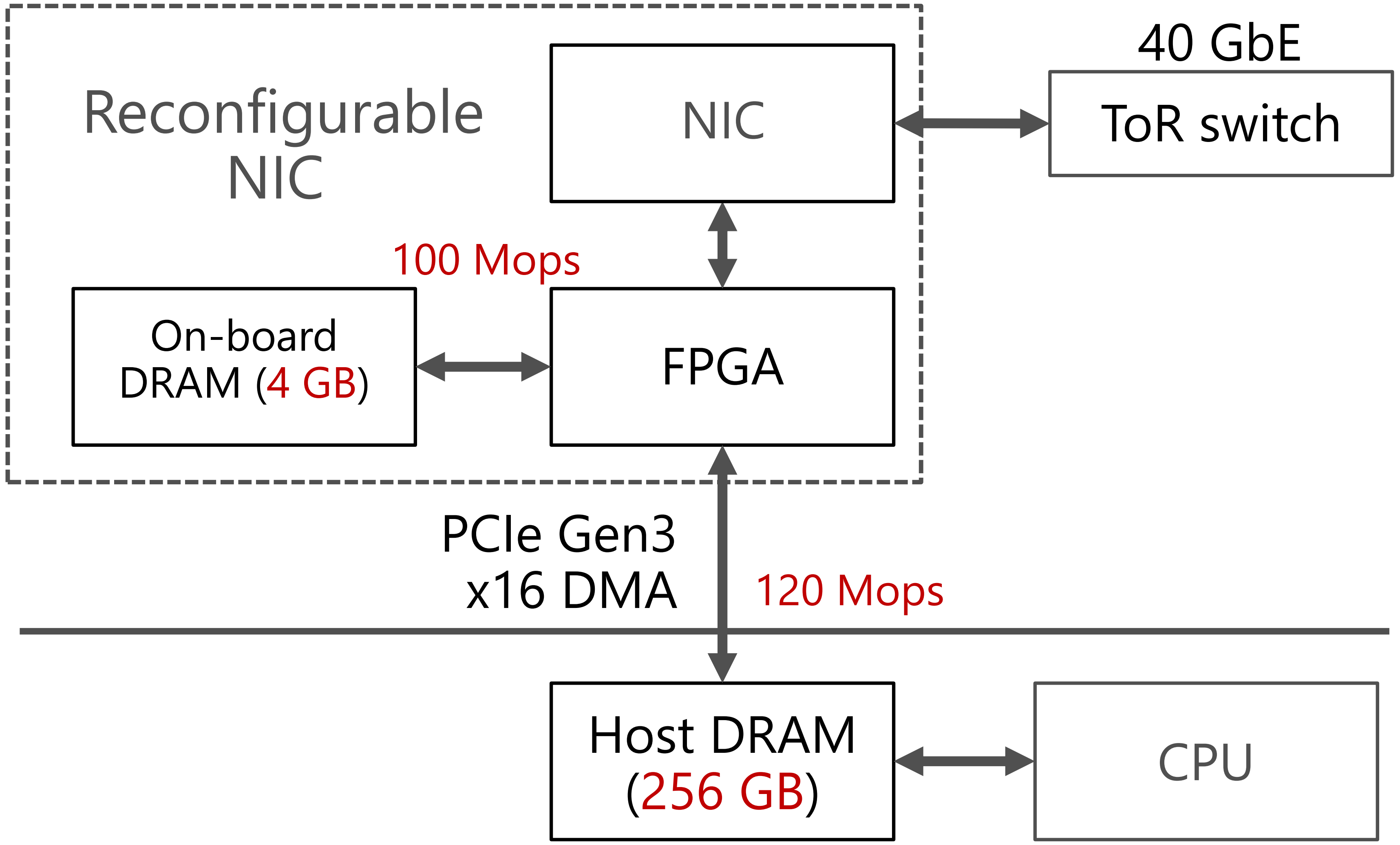
Throughput: 191x single-key atomics, 20x long-tail workload

We hope future RDMA NICs could adopt out-of-order execution for atomic operations!

Design Principles

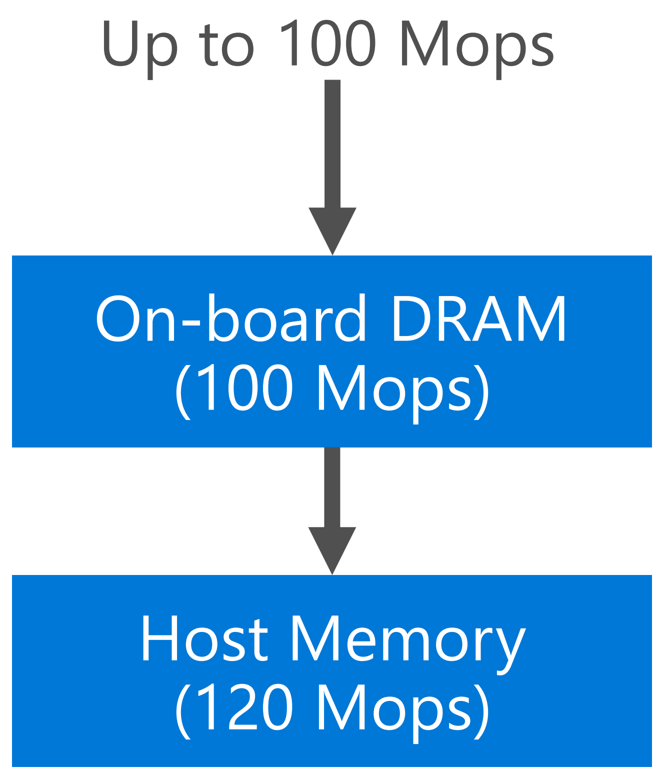
1. Be frugal on memory accesses for both GET and PUT
2. Hide memory access latency
3. Leverage throughput of both on-board and host memory
4. Offload simple client computation to server

How to Use On-board DRAM?

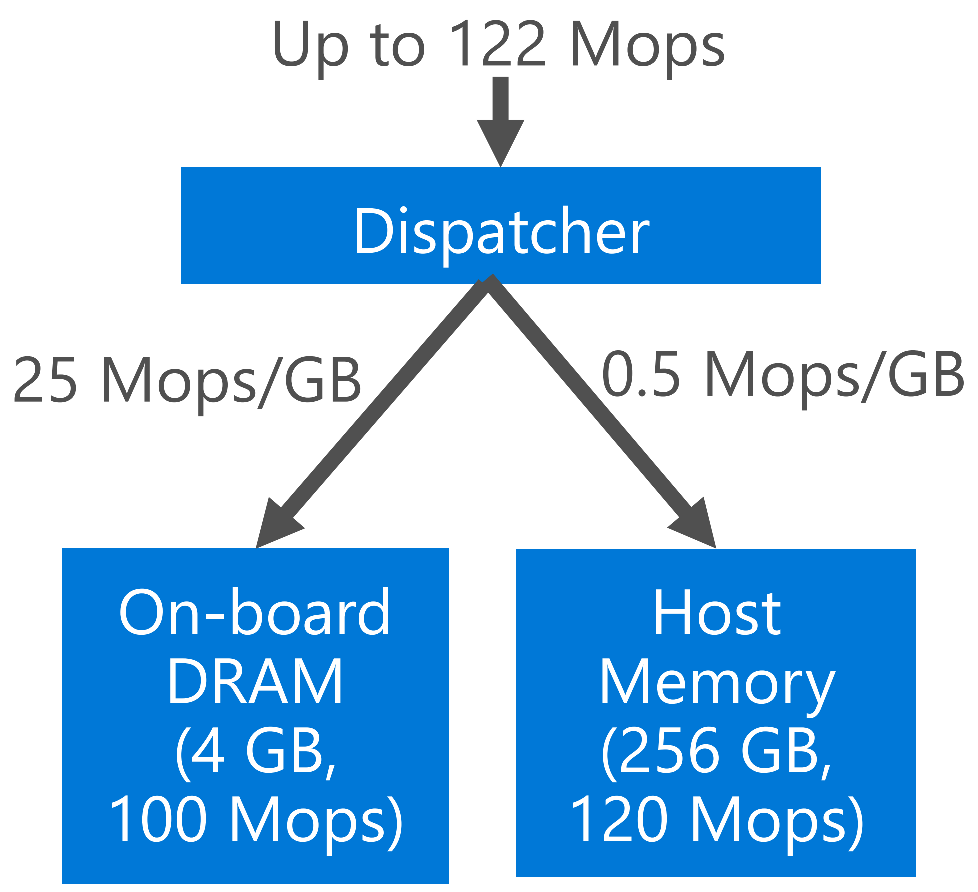


How to Use On-board DRAM?

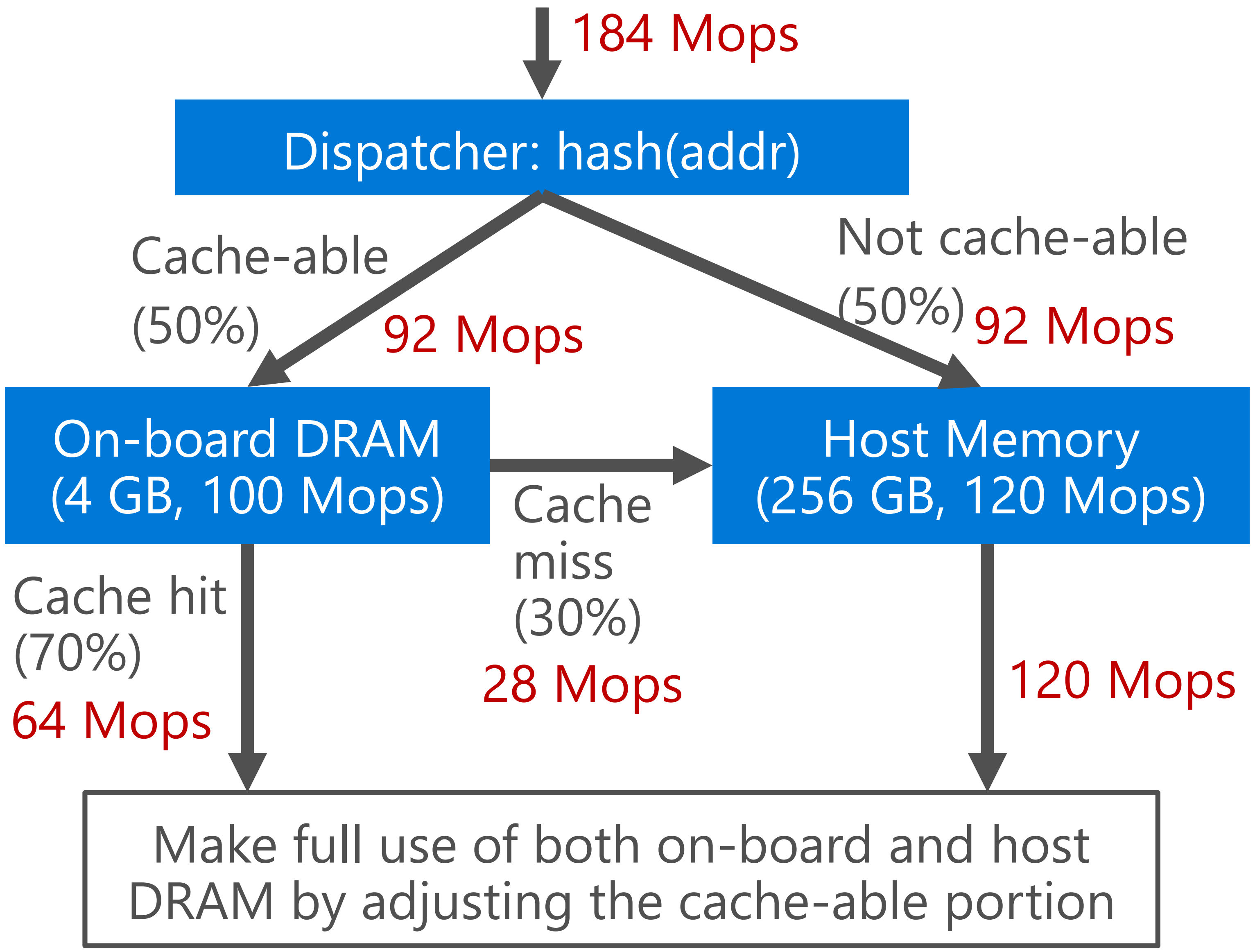
Cache?



Load balance?


Up to 122 Mops
Dispatcher
25 Mops/GB
0.5 Mops/GB
On-board DRAM (4 GB, 100 Mops)
Host Memory (256 GB, 120 Mops)

Load Dispatch = Cache + Load Balance



Make full use of both on-board and host DRAM by adjusting the cache-able portion

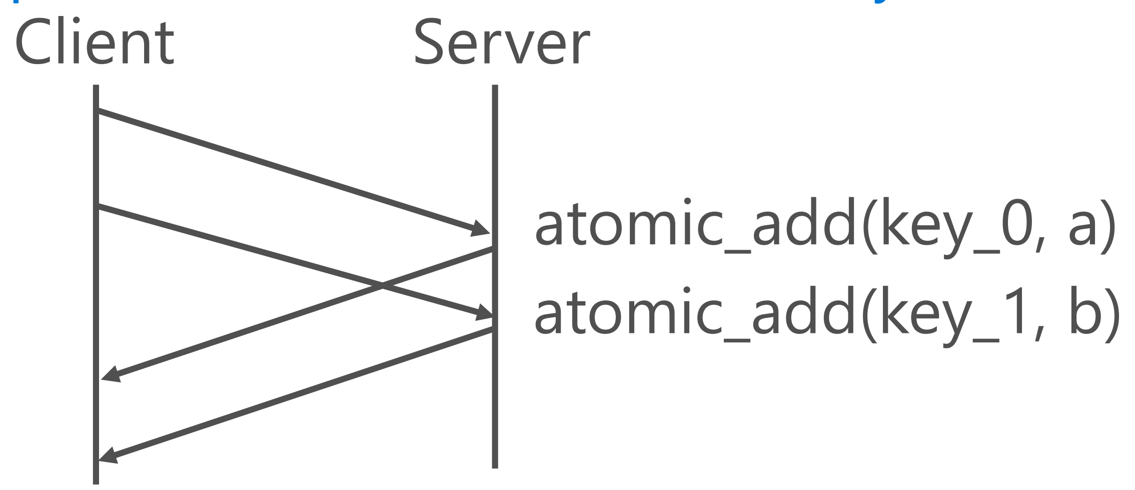
Design Principles

1. Be frugal on memory accesses for both GET and PUT
2. Hide memory access latency
3. Leverage throughput of both on-board and host memory
4. Offload simple client computation to server

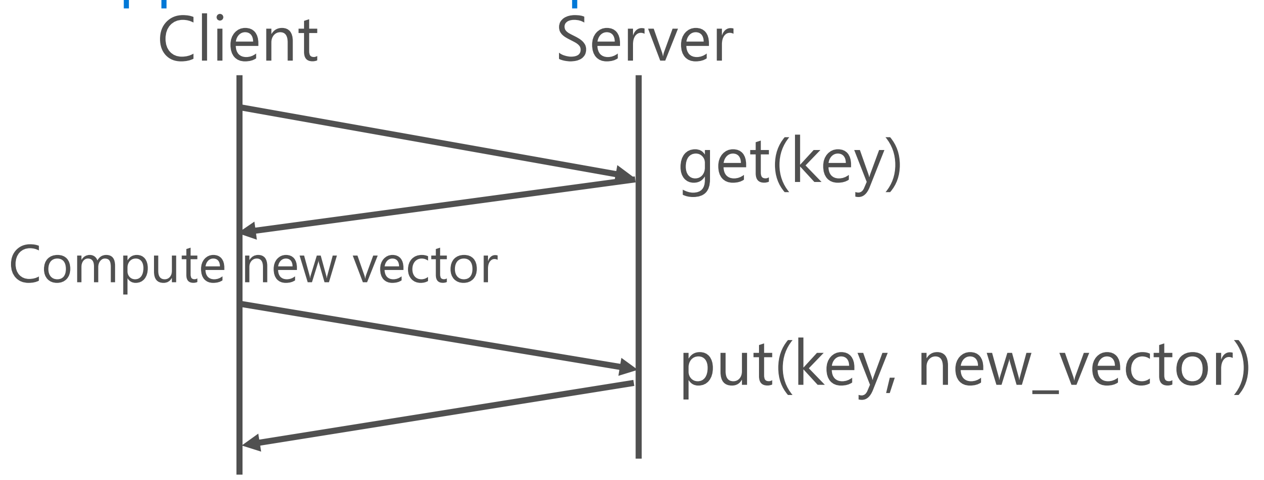
Vector-Type Operations

Example: key[0] += a, key[1] += b

Approach 1: Each element as a key



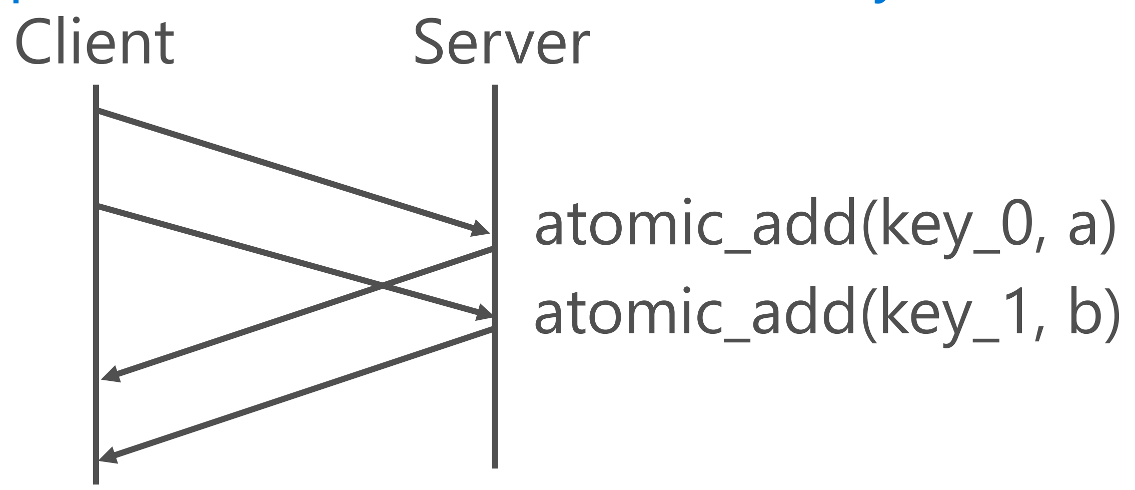
Approach 2: Compute at client



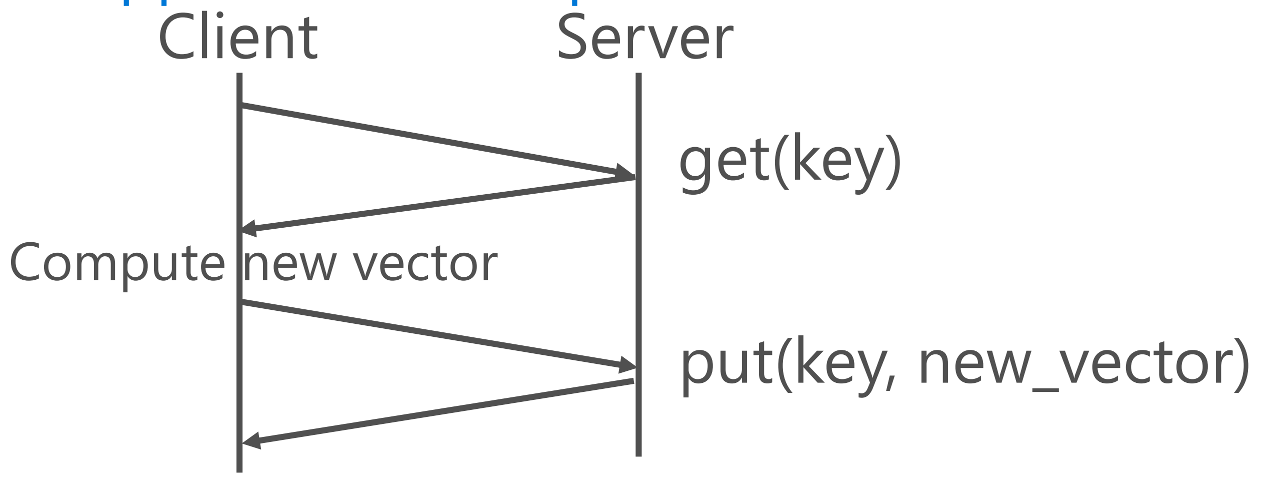
Vector-Type Operations

Example: key[0] += a, key[1] += b

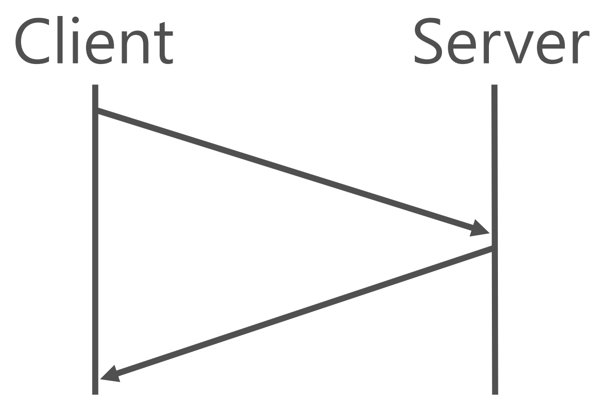
Approach 1: Each element as a key



Approach 2: Compute at client



Our approach: Vector operations

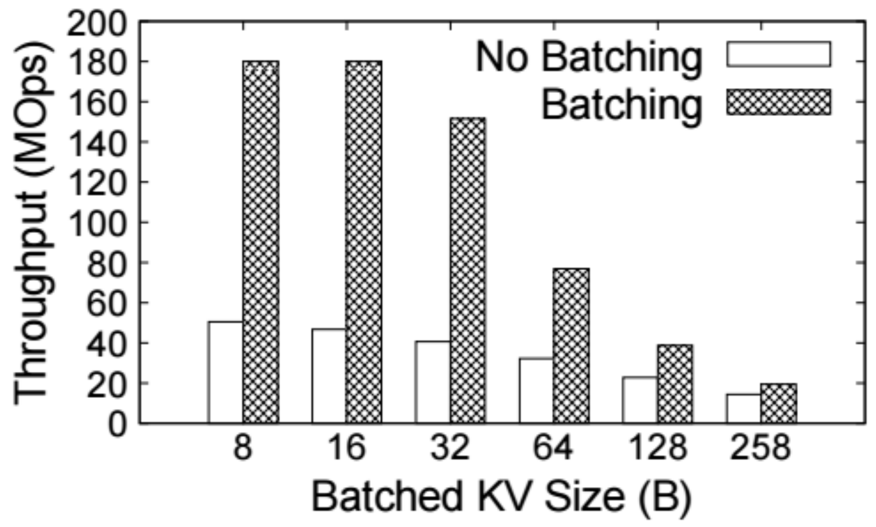


vector_atomic_add
(key, [a,b])

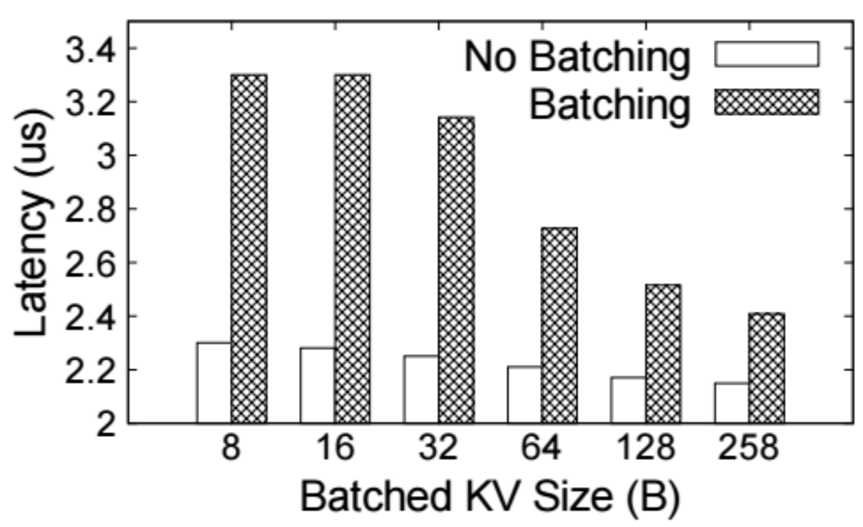
Actually the "atomic add" function can be user-defined.

Client-side Network Batching

Amortize packet header overhead



(a) Throughput.

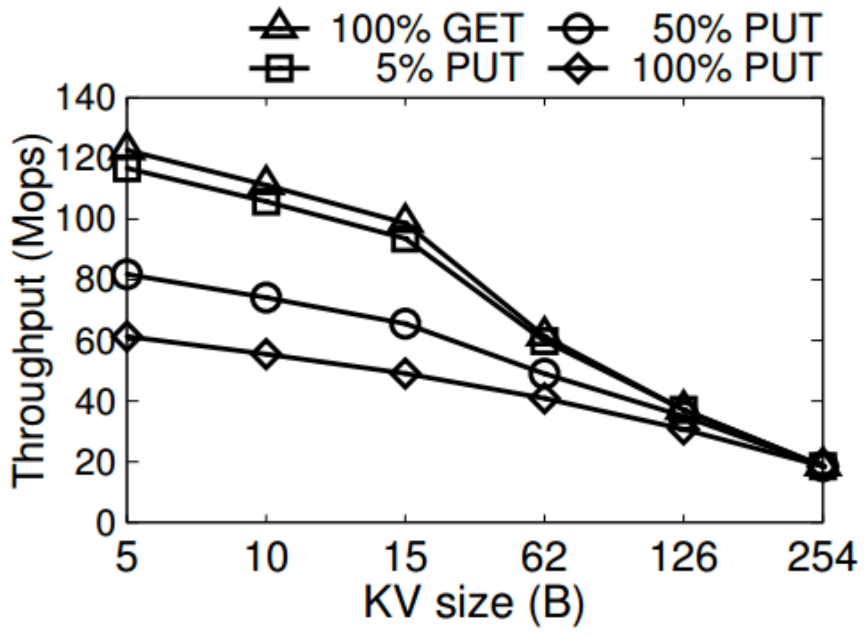


(b) Latency.

3.6x throughput
(50 Mops -> 180 Mops)

+1 us latency
(2.2 us -> 3.3 us)

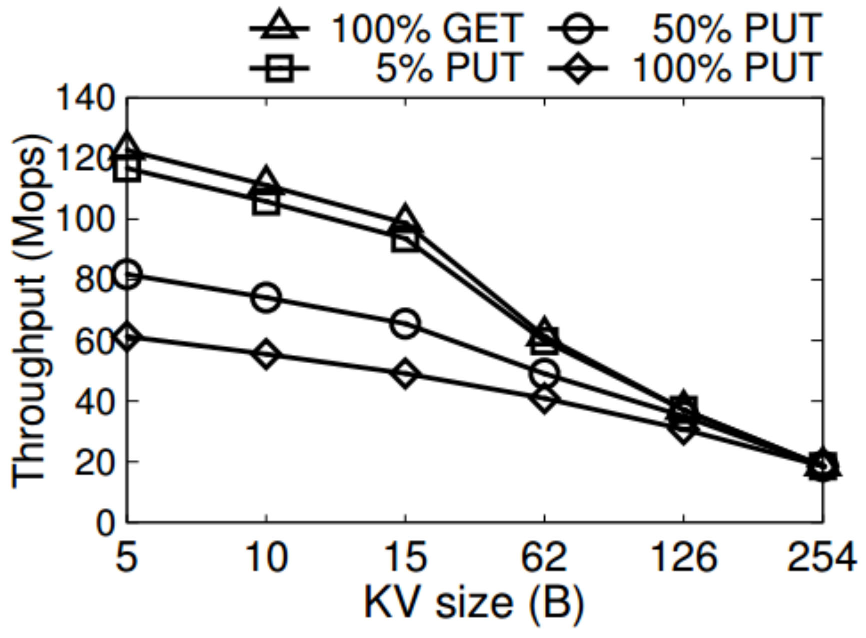
KV-Direct System Performance



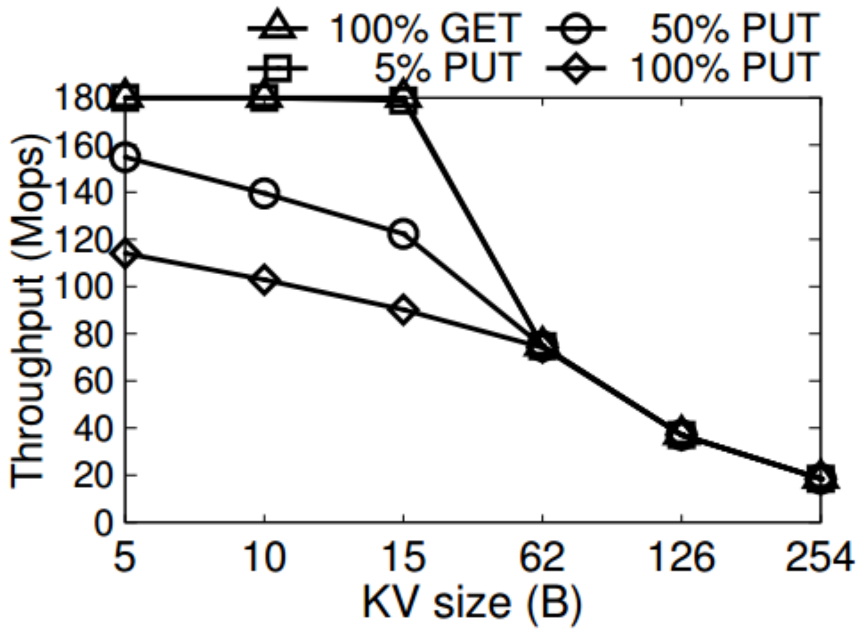
(a) Uniform.

Throughput

KV-Direct System Performance



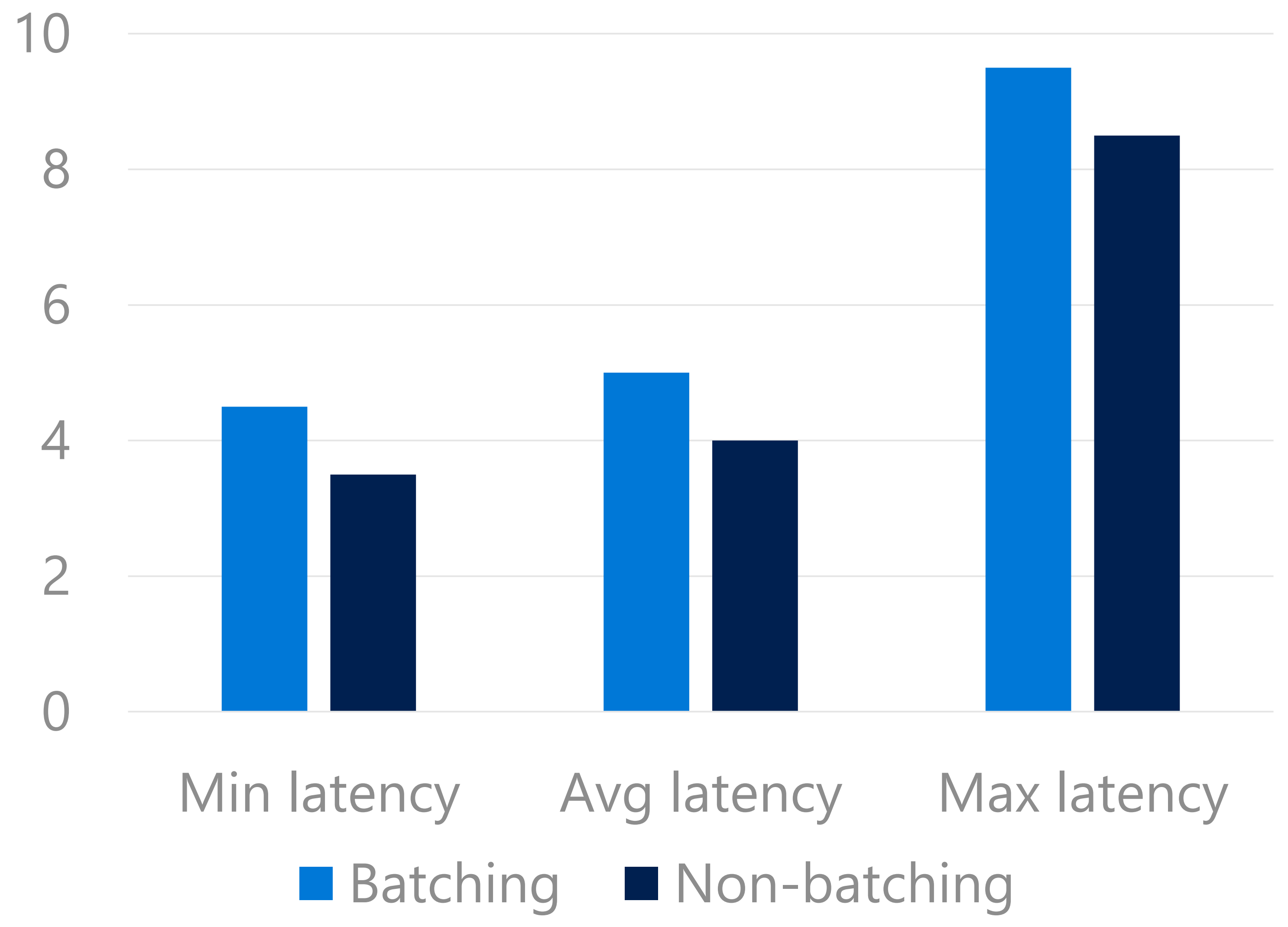
(a) Uniform.



(b) Long-tail.

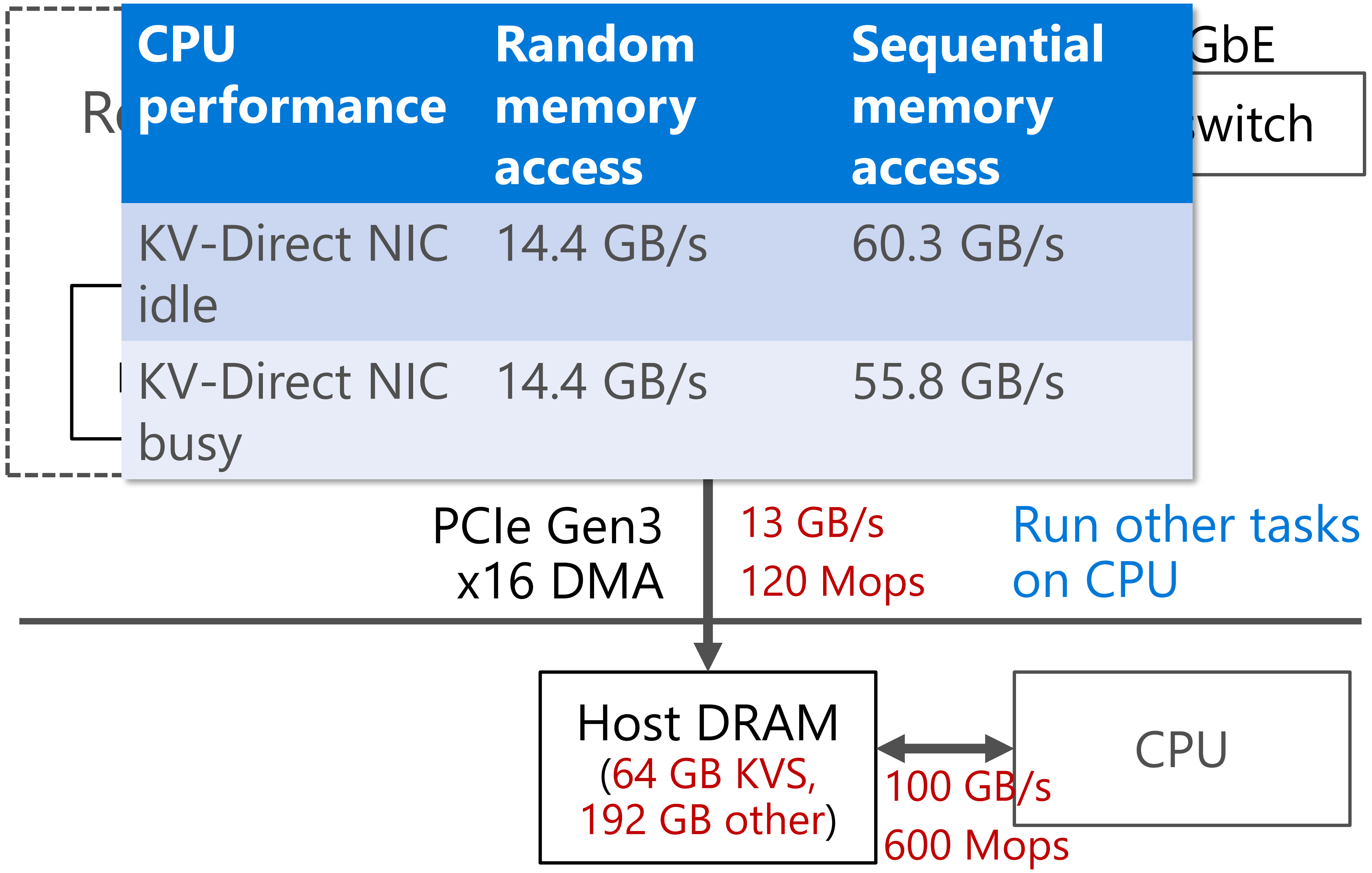
Throughput

KV-Direct System Performance

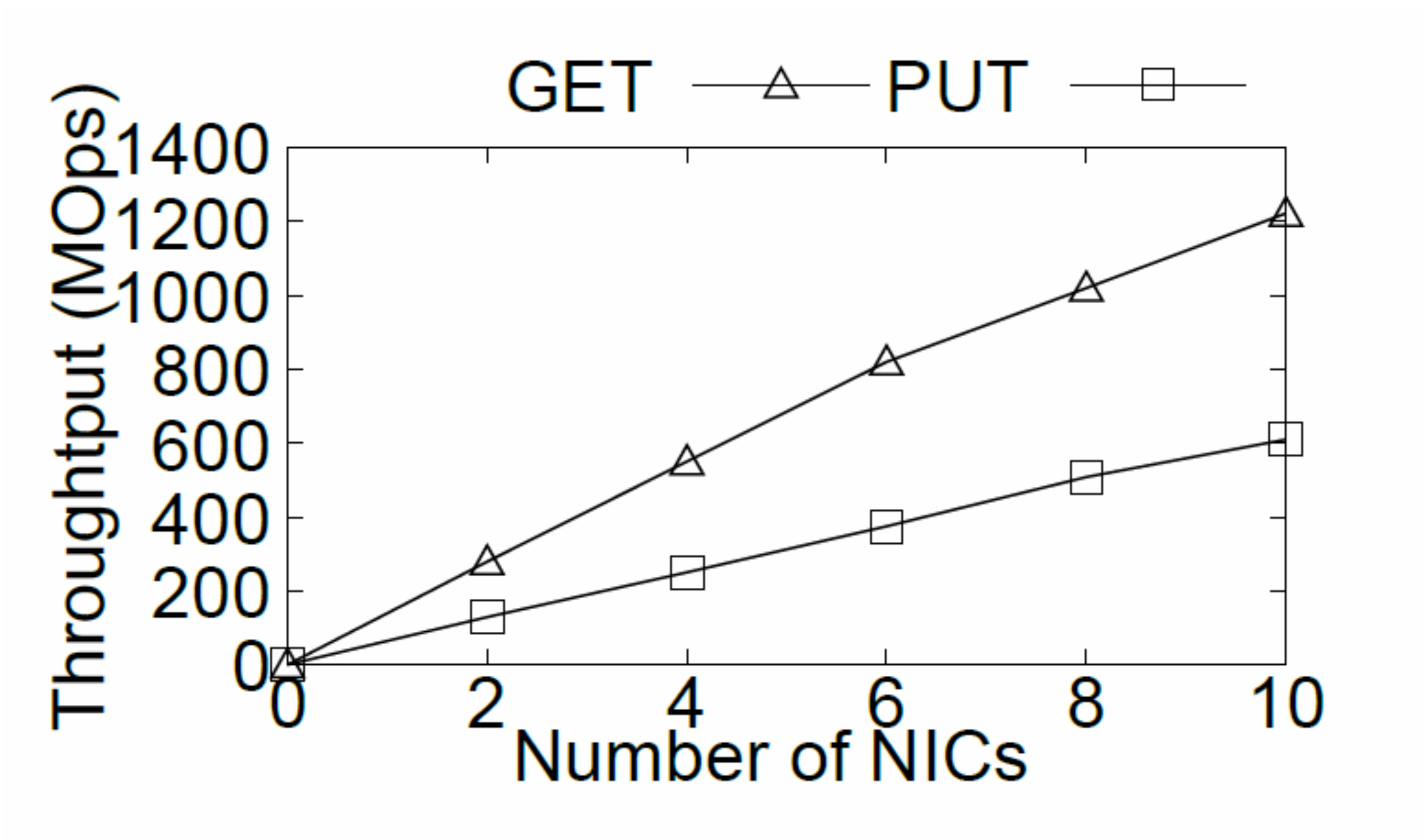


Little Impact on CPU Performance

CPU performance	Random memory access	Sequential memory access
KV-Direct NIC idle	14.4 GB/s	60.3 GB/s
KV-Direct NIC busy	14.4 GB/s	55.8 GB/s



Scalability with Multiple NICs



KVS Performance Comparison

	Tput (Mops) (GET / PUT)	Power (Kops/W)	Comment	Latency (us) (GET / PUT)
Memcached	1.5 / 1.5	5 / 5	TCP/IP	50 / 50
MemC3	4.3 / 4.3	14 / 14	TCP/IP	50 / 50
RAMCloud	6 / 1	20 / 3.3	Kernel bypass	5 / 14
MICA (12 NICs)	137 / 135	342 / 337	Kernel bypass	81 / 81
FARM	6 / 3	30 (261) / 15	One-side RDMA	4.5 / 10
DrTM-KV	115 / 14	500 (3972) / 60	One-side RDMA	3.4 / 6.3
HERD	35 / 25	490 / 300	Two-side RDMA	4 / 4
FPGA-Xilinx	14 / 14	106 / 106	FPGA	3.5 / 4.5
Mega-KV	166 / 80	330 / 160	GPU	280 / 280
KV-Direct (1 NIC)	180 / 114	1487 (5454) / 942 (3454)	Programmable NIC	4.3 / 5.4
KV-Direct (10 NICs)	1220 / 610	3417 (4518) / 1708 (2259)	Programmable NIC	4.3 / 5.4

* Number in parenthesis indicates power efficiency based on power consumption of NIC only, for server-bypass systems.

KVS Performance Comparison

	Tput (Mops) (GET / PUT)	Power (Kops/W)	Comment	Latency (us) (GET / PUT)
Memcached	1.5 / 1.5	5 / 5	TCP/IP	50 / 50
MemC3	4.3 / 4.3	14 / 14	TCP/IP	50 / 50
RAMCloud	6 / 1	20 / 3.3	Kernel bypass	5 / 14
MICA (12 NICs)	137 / 135	342 / 337	Kernel bypass	81 / 81
FARM	6 / 3	30 (261) / 15	One-side RDMA	4.5 / 10
DrTM-KV	115 / 14	500 (3972) / 60	One-side RDMA	3.4 / 6.3
HERD	35 / 25	490 / 300	Two-side RDMA	4 / 4
FPGA-Xilinx	14 / 14	106 / 106	FPGA	3.5 / 4.5
Mega-KV	166 / 80	330 / 160	GPU	280 / 280
KV-Direct (1 NIC)	180 / 114	1487 (5454) / 942 (3454)	Programmable NIC	4.3 / 5.4
KV-Direct (10 NICs)	1220 / 610	3417 (4518) / 1708 (2259)	Programmable NIC	4.3 / 5.4

* Number in parenthesis indicates power efficiency based on power consumption of NIC only, for server-bypass systems.

KVS Performance Comparison

	Tput (Mops) (GET / PUT)	Power (Kops/W)	Comment	Latency (us) (GET / PUT)
Memcached	1.5 / 1.5	5 / 5	TCP/IP	50 / 50
MemC3	4.3 / 4.3	14 / 14	TCP/IP	50 / 50
RAMCloud	6 / 1	20 / 3.3	Kernel bypass	5 / 14
MICA (12 NICs)	137 / 135	342 / 337	Kernel bypass	81 / 81
FARM	6 / 3	30 (261) / 15	One-side RDMA	4.5 / 10
DrTM-KV	115 / 14	500 (3972) / 60	One-side RDMA	3.4 / 6.3
HERD	35 / 25	490 / 300	Two-side RDMA	4 / 4
FPGA-Xilinx	14 / 14	106 / 106	FPGA	3.5 / 4.5
Mega-KV	166 / 80	330 / 160	GPU	280 / 280
KV-Direct (1 NIC)	180 / 114	1487 (5454) / 942 (3454)	Programmable NIC	4.3 / 5.4
KV-Direct (10 NICs)	1220 / 610	3417 (4518) / 1708 (2259)	Programmable NIC	4.3 / 5.4

* Number in parenthesis indicates power efficiency based on power consumption of NIC only, for server-bypass systems.

What application?

Back-of-envelope calculations show potential performance gains when KV-Direct is applied in end-to-end applications. In PageRank, because each edge traversal can be implemented with one KV operation, KV-Direct supports 1.2 billion TEPS on a server with 10 programmable NICs. In comparison, GRAM (Ming Wu on SoCC'15) supports 250M TEPS per server, bounded by interleaved computation and random memory access.

Are the optimizations general?

The discussion section of the paper discusses NIC hardware with different capacity. First, the goal of KV-Direct is to leverage existing hardware in data centers instead of designing a specialized hardware to achieve maximal KVS performance. Even if future NICs have faster or larger on-board memory, under long-tail workload, our load dispatch design still shows performance gain. The hash table and slab allocator design is generally applicable to cases where we need to be frugal on memory accesses. The out-of-order execution engine can be applied to all kinds of applications in need of latency hiding.

How will the load dispatcher with different hit rate?

Throughput is similar to state-of-art

With a single KV-Direct NIC, the throughput is equivalent to 20 to 30 CPU cores. These CPU cores can run other CPU intensive or memory intensive workload, because the host memory bandwidth is much larger than the PCIe bandwidth of a single KV-Direct NIC. So we basically save tens of CPU cores per programmable NIC. With ten programmable NICs, the throughput can grow almost linearly.

Really Save Money?
What about cost of 1 Smart NIV v.s. 10 CPUs?

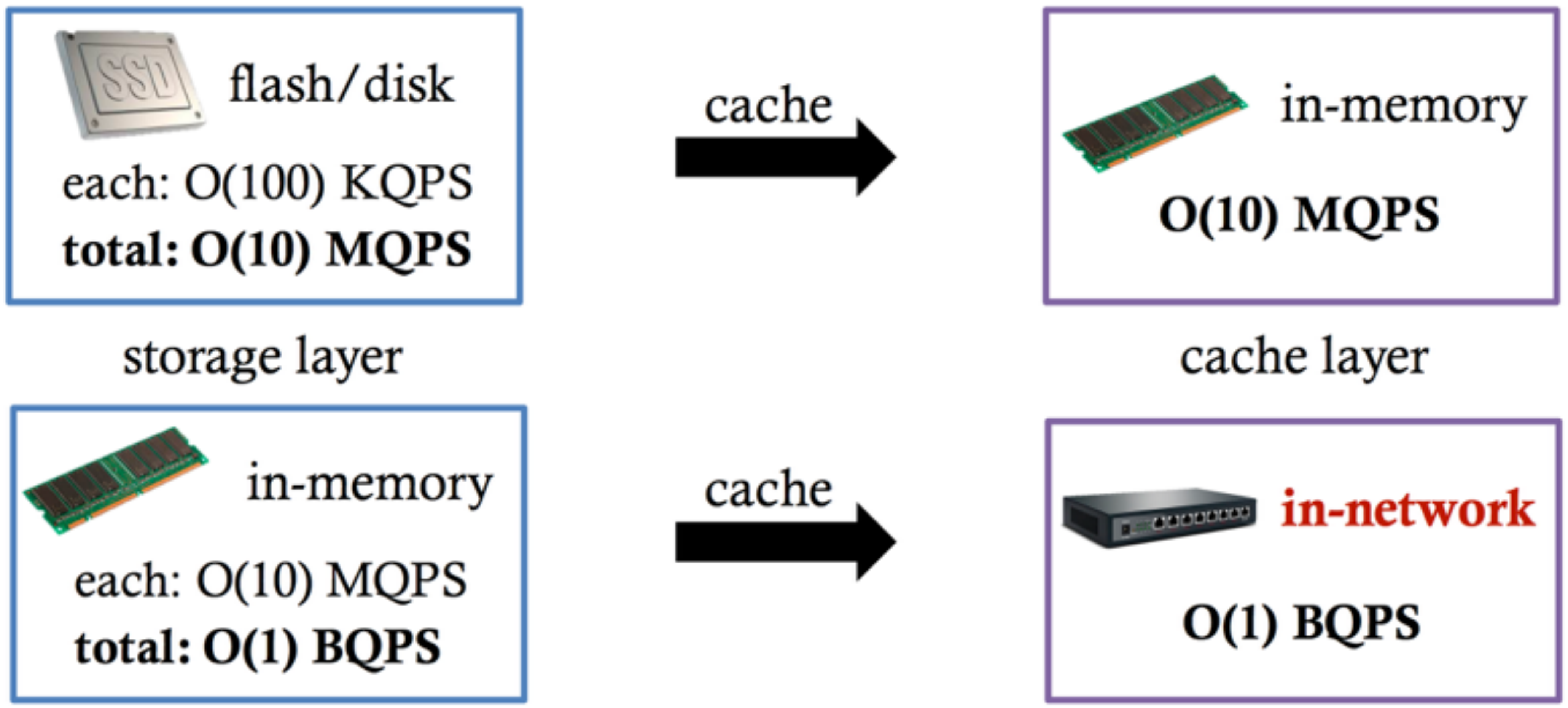
Consistency among multiple NICs

Each NIC behaves as if it is an independent KV-Direct server. Each NIC serves a disjoint partition of key space and reserves a disjoint region of host memory. The clients distribute load to each NIC according to the hash of keys, similar to the design of other distributed key-value stores. Surely, the multiple NICs suffer load imbalance problem in long-tail workload, but the load imbalance is not significant with a small number of partitions. The NetCache system in this session can also mitigate the load imbalance problem.

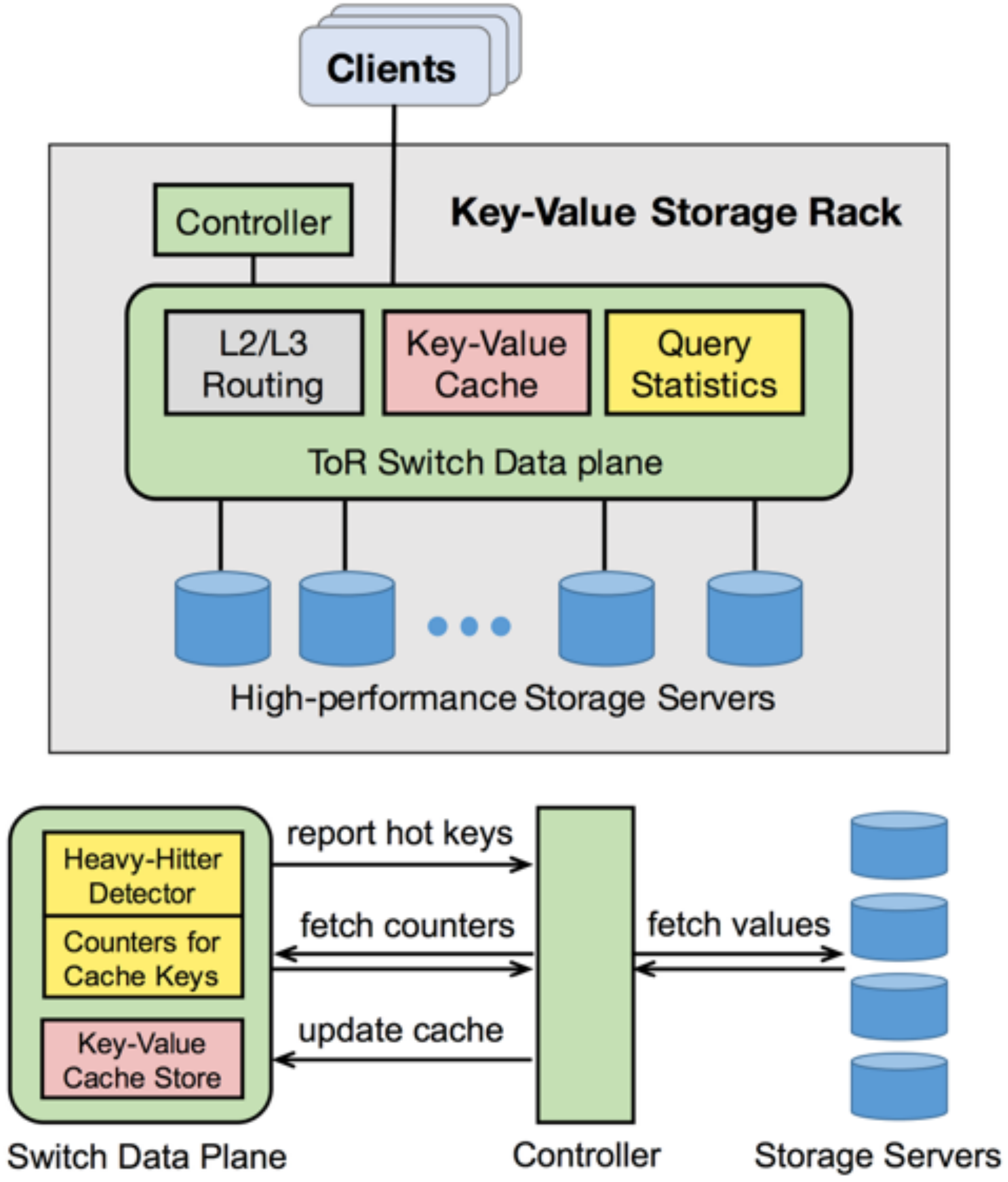
Can NetCache balance loads for per SmartNIC?

Cache - NetCache (SOSP 17')

- Goal: Make the fast, small-cache viable for modern in-memory Key Value servers.
- Storage scale: per-rack



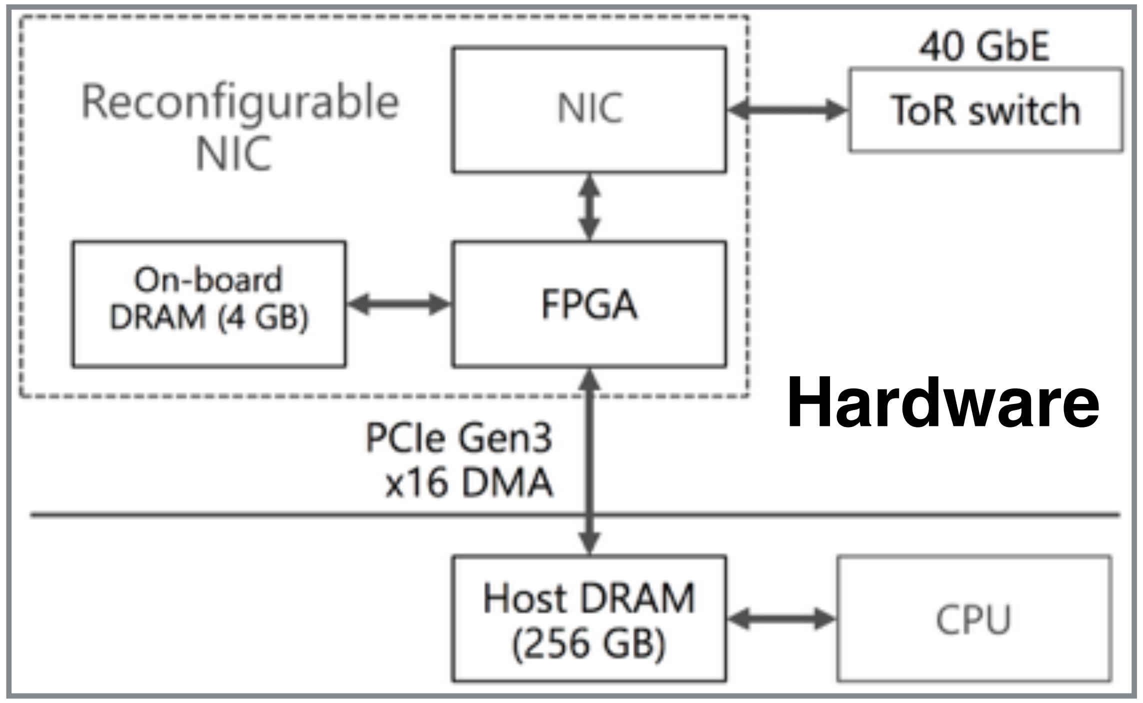
Cache - NetCache (SOSP 17')

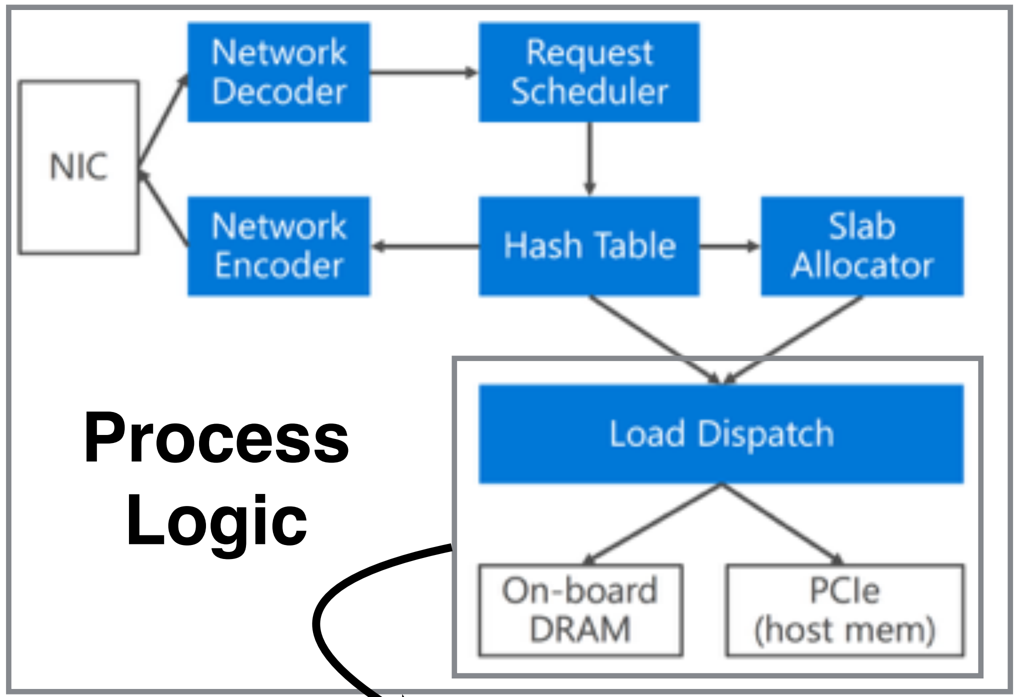


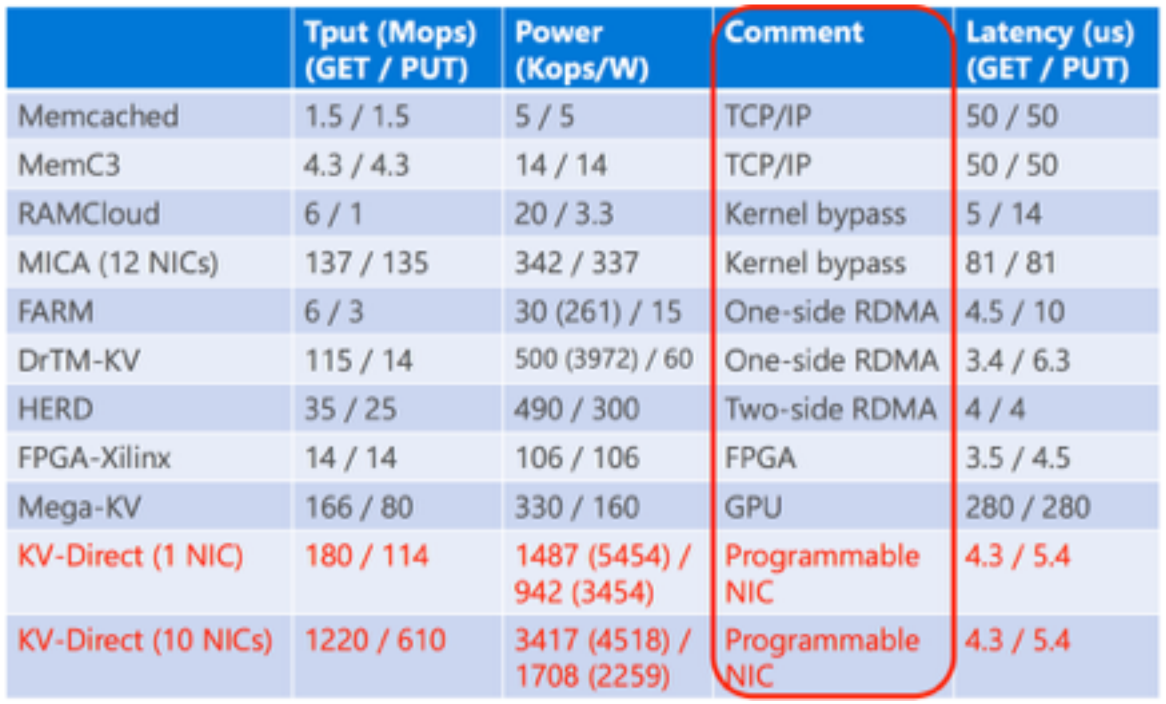
- Switch Data Plane
  1. Key-value store to serve queries for cached keys
  2. Query statistics to enable efficient cache updates

- Switch Control Plane
  1. Insert hot items into the cache and evict less popular items
  2. Manage memory allocation for on-chip key-value store

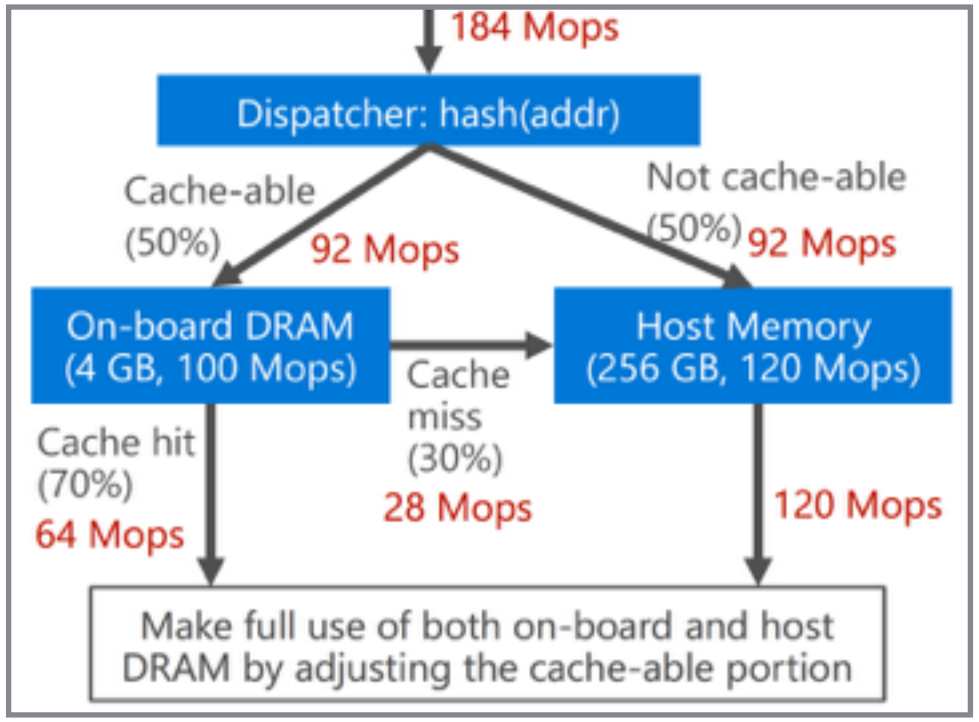
Cache - KVDirect (SOSP 17')



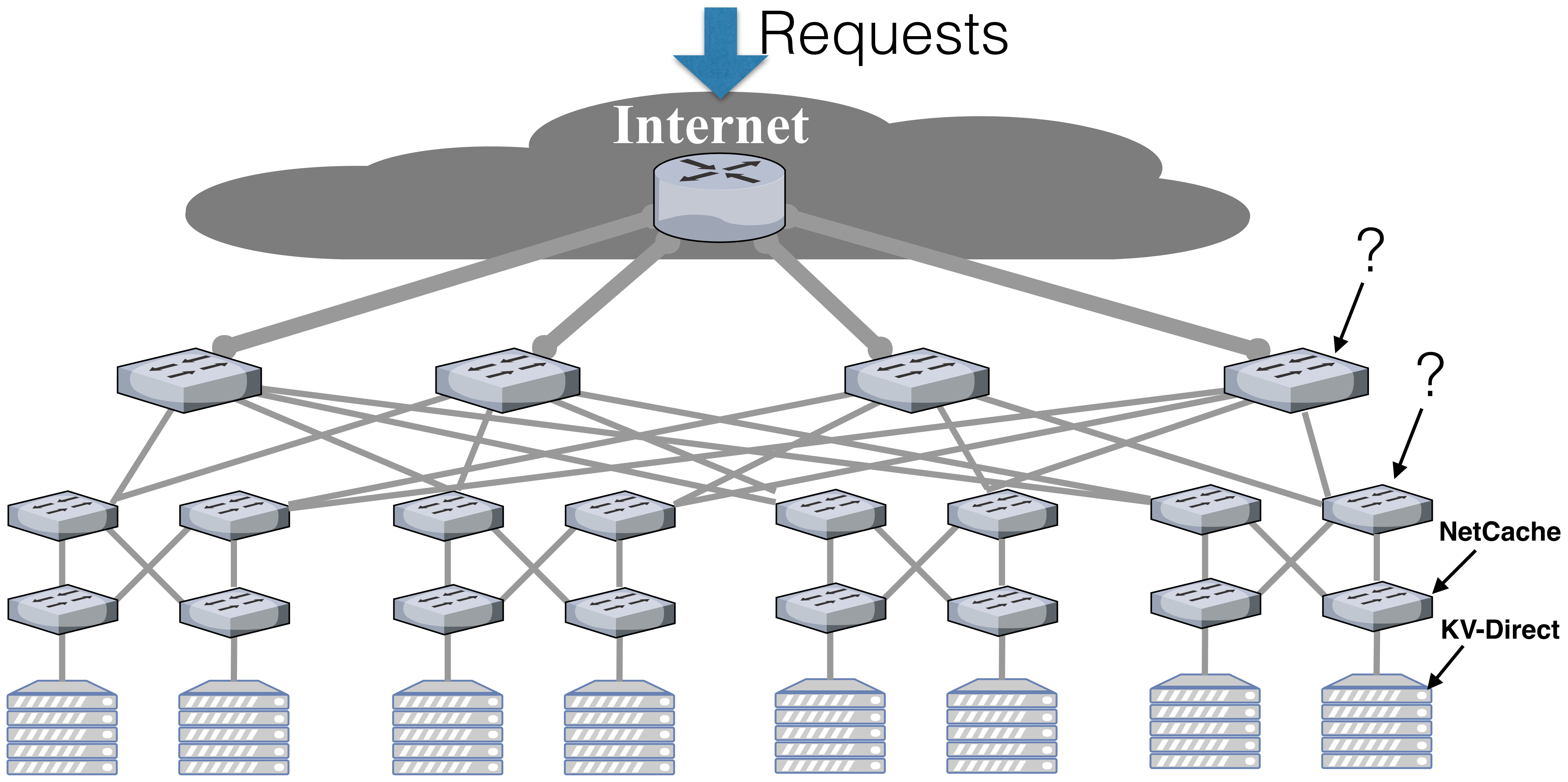




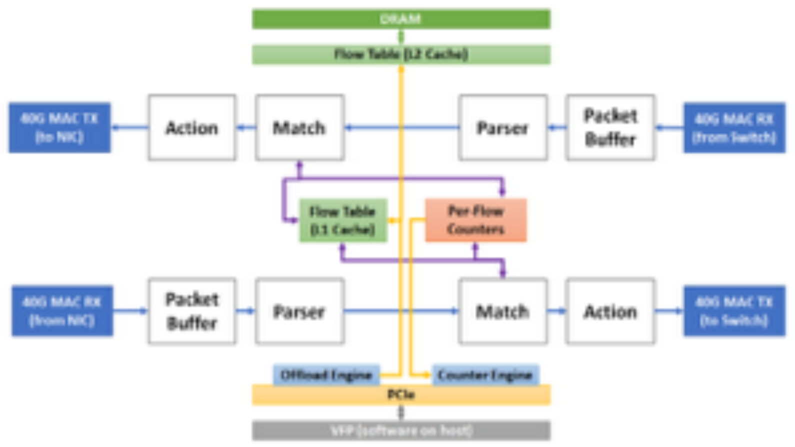
	Tput (Mops) (GET / PUT)	Power (Kops/W)	Comment	Latency (us) (GET / PUT)
Memcached	1.5 / 1.5	5 / 5	TCP/IP	50 / 50
MemC3	4.3 / 4.3	14 / 14	TCP/IP	50 / 50
RAMCloud	6 / 1	20 / 3.3	Kernel bypass	5 / 14
MICA (12 NICs)	137 / 135	342 / 337	Kernel bypass	81 / 81
FARM	6 / 3	30 (261) / 15	One-side RDMA	4.5 / 10
DrTM-KV	115 / 14	500 (3972) / 60	One-side RDMA	3.4 / 6.3
HERD	35 / 25	490 / 300	Two-side RDMA	4 / 4
FPGA-Xilinx	14 / 14	106 / 106	FPGA	3.5 / 4.5
Mega-KV	166 / 80	330 / 160	GPU	280 / 280
KV-Direct (1 NIC)	180 / 114	1487 (5454) / 942 (3454)	Programmable NIC	4.3 / 5.4
KV-Direct (10 NICs)	1220 / 610	3417 (4518) / 1708 (2259)	Programmable NIC	4.3 / 5.4

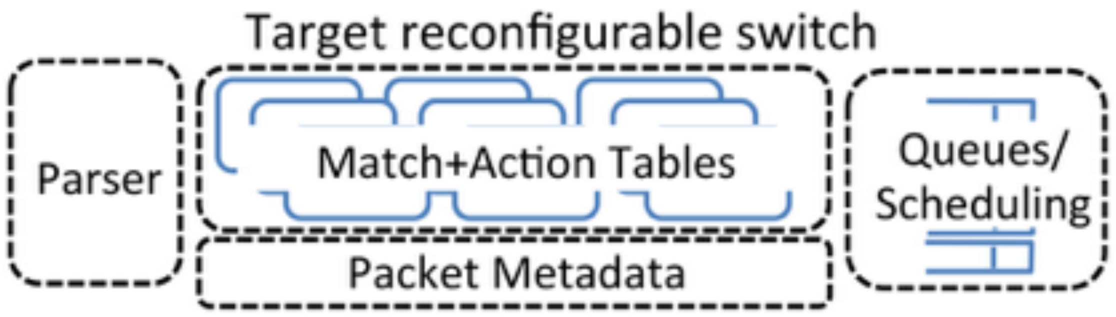

184 Mops
Dispatcher: hash(addr)
Cache-able (50%)
92 Mops
Not cache-able (50%)
92 Mops
On-board DRAM (4 GB, 100 Mops)
Host Memory (256 GB, 120 Mops)
Cache miss (30%)
28 Mops
Cache hit (70%)
64 Mops
120 Mops
Make full use of both on-board and host DRAM by adjusting the cache-able portion

Hierarchical Cache



Smart NIC & Programmable Switch



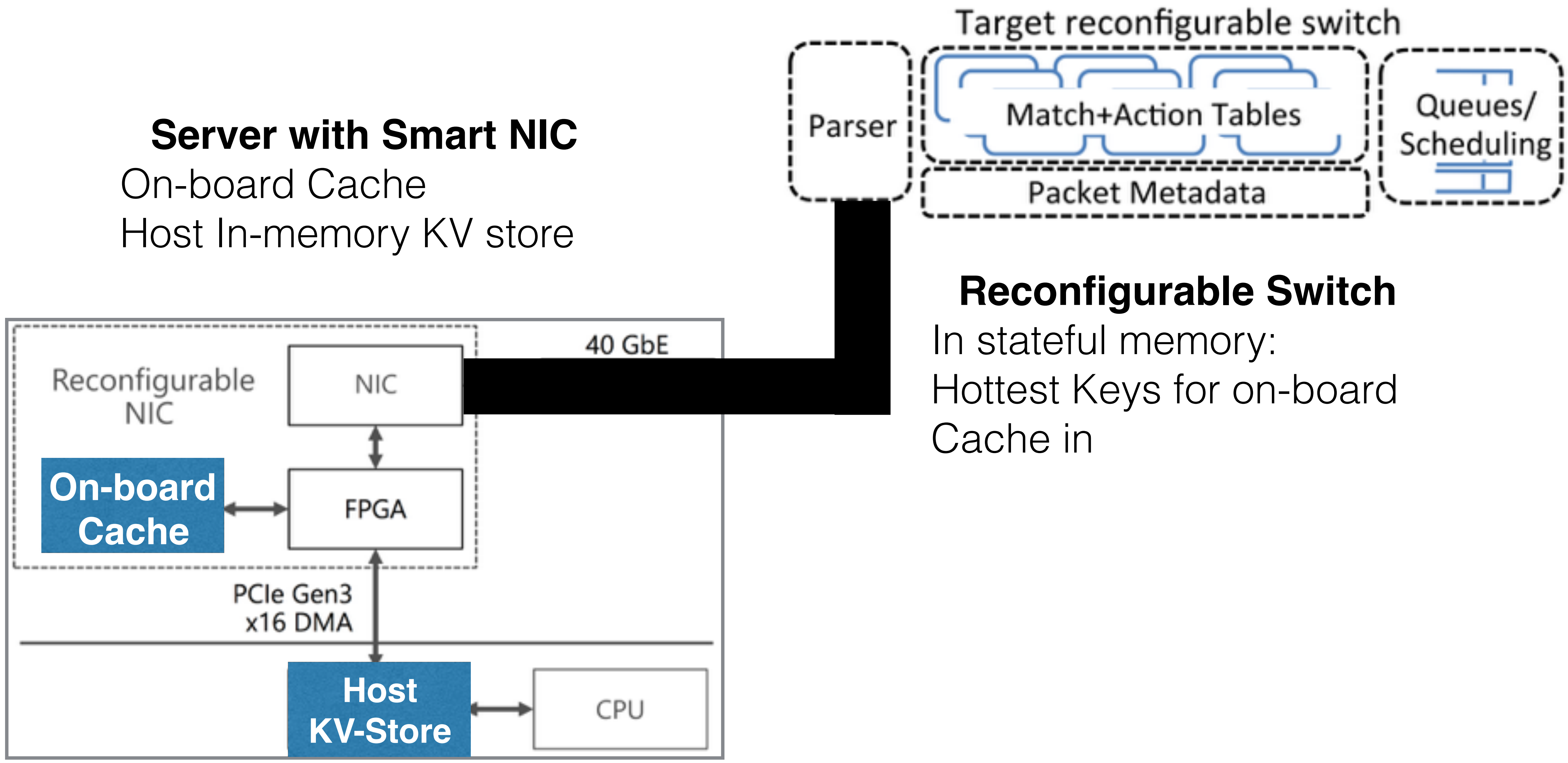


Similarity	Protocol Independent User Programmable	
Difference	SmartNIC: GBs on-board SRAM; complex computing power	Programmable Switch: MBs on-board SRAM; limited ALU elements

Hierarchical Cache

Server with Smart NIC
On-board Cache
Host In-memory KV store

Reconfigurable Switch
In stateful memory:
Hottest Keys for on-board Cache in



Thanks

Q & A