Microsoft Research

Microsoft

FaRM: Fast Remote Memory

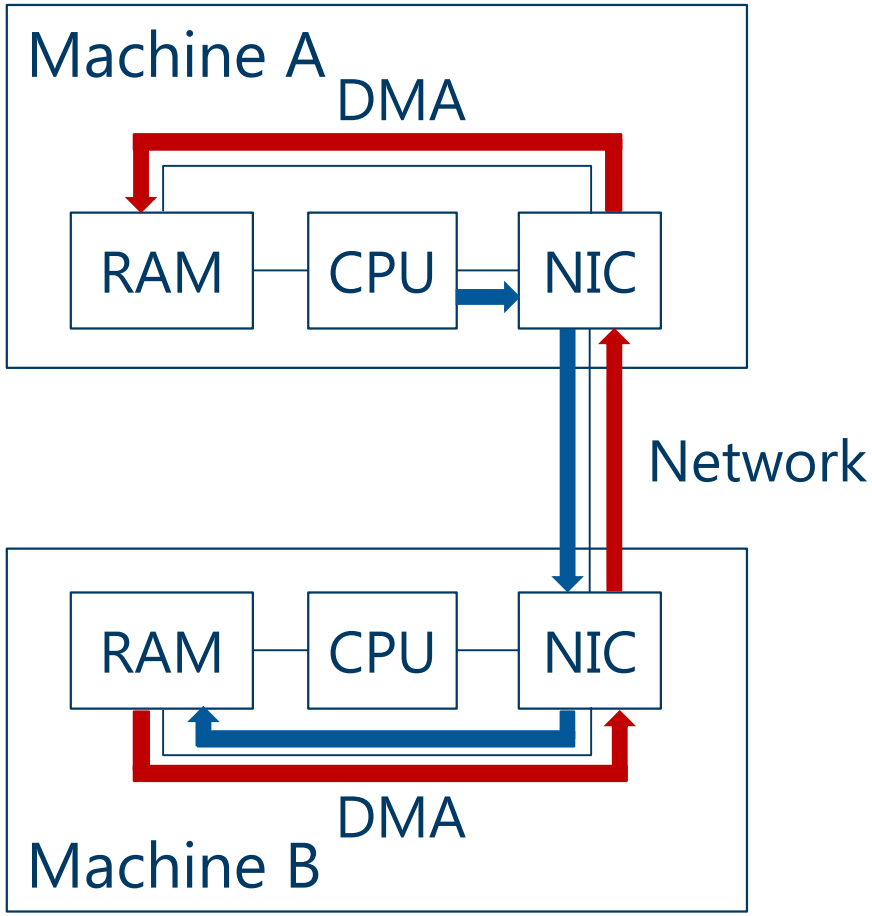
Aleksandar Dragojević, Dushyanth Narayanan,
Orion Hodson, Miguel Castro

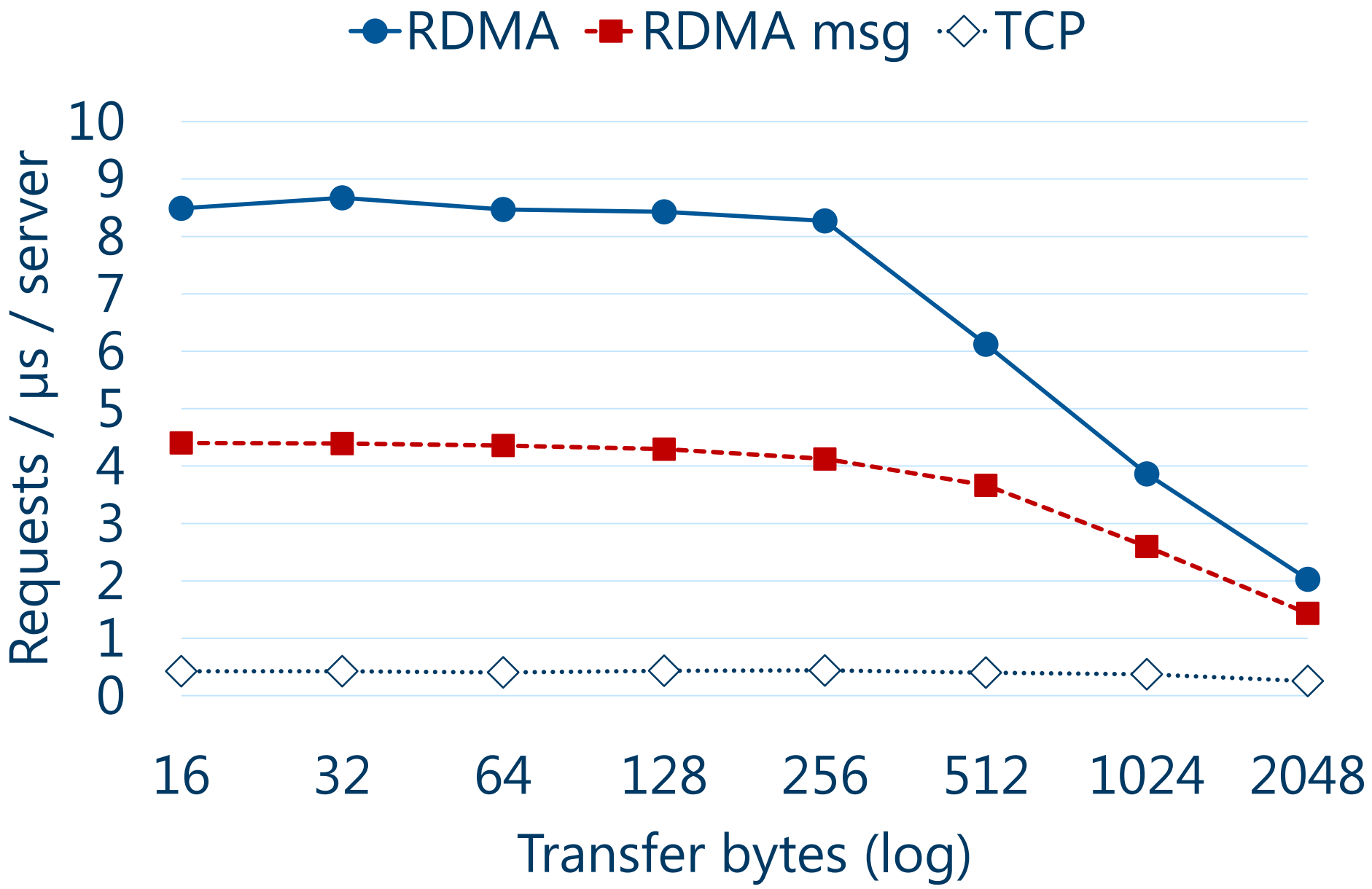
Hardware trends

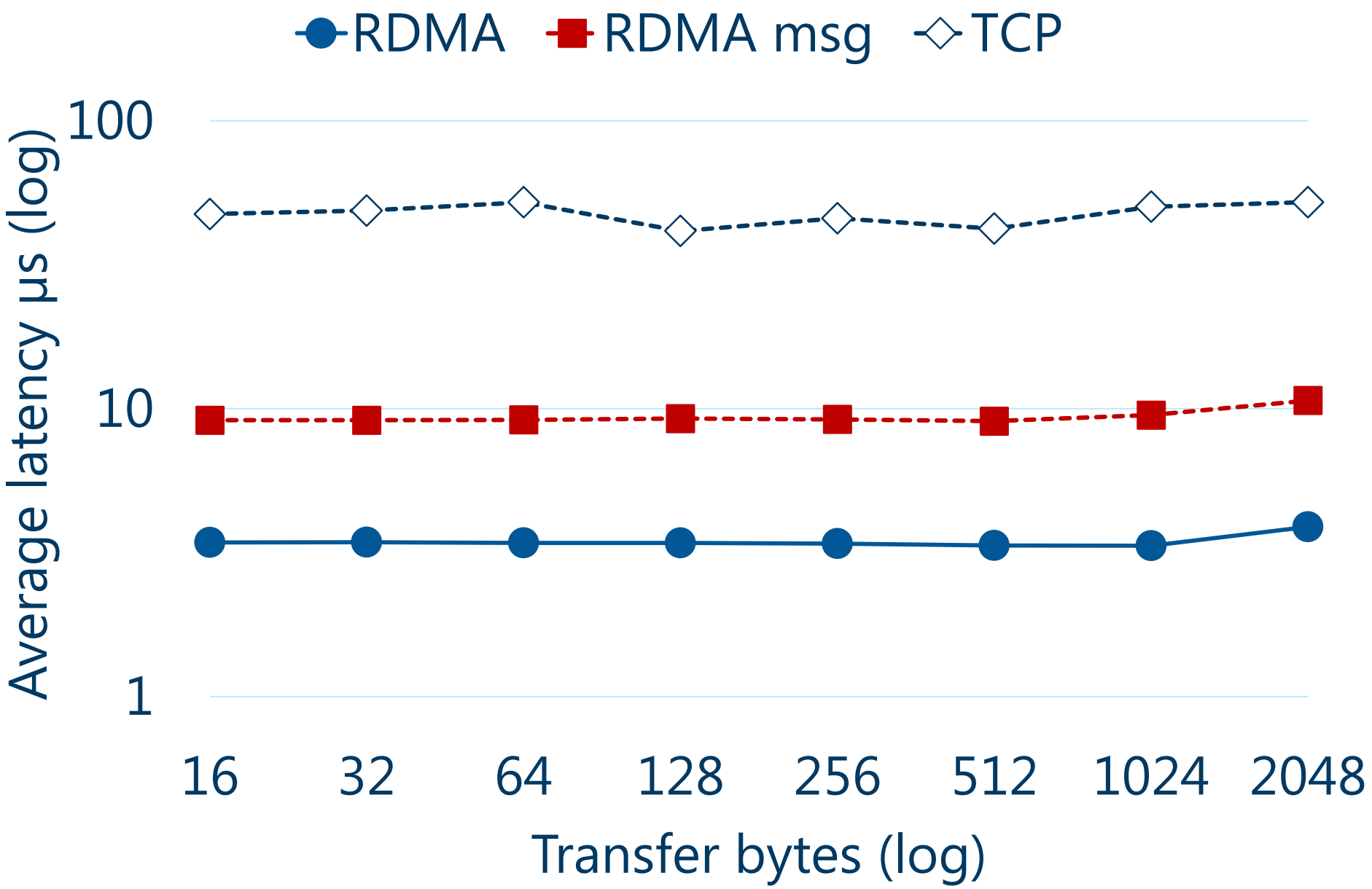
- Main memory is cheap
  - 100 GB – 1 TB per server
  - 10 – 100 TBs in a small cluster
- New data centre networks
  - 40 Gbps throughput (100 this year)
  - 1-3 μs latency
  - RDMA primitives

Remote direct memory access

- Read / write remote memory
  - NIC performs DMA requests
- FaRM uses RDMA extensively
  - Reads to directly read data
  - Writes into remote buffers for messaging
- Great performance
  - Bypasses the kernel
  - Bypasses the remote CPU




RDMA
RDMA msg
TCP
Requests / µs / server
0
1
2
3
4
5
6
7
8
9
10
16
32
64
128
256
512
1024
2048
Transfer bytes (log)


RDMA
RDMA msg
TCP
100
10
1
Average latency µs (log)
16
32
64
128
256
512
1024
2048
Transfer bytes (log)

Applications

- Data centre applications
  - Irregular access patterns
  - Latency sensitive
- Data serving
  - Key-value store
  - Graph store
- Enabling new applications

Paper

- RDMA communication
- Programming model 
- Address space management
- Transactions and lock-free operations
- Hashtable

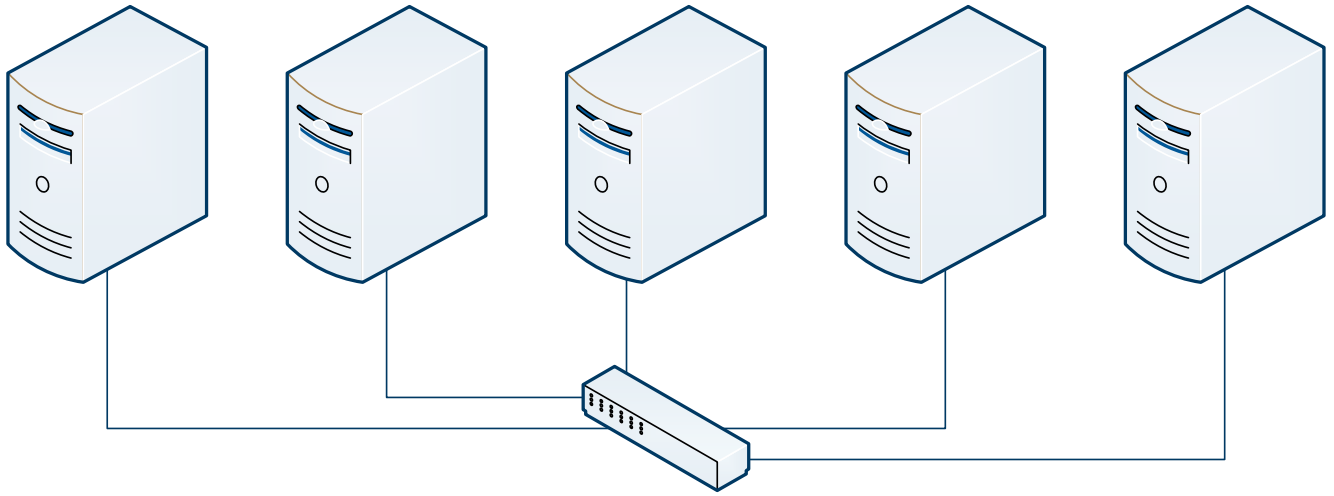
How to program a modern cluster?

We have:

- TBs of DRAM
- 100s of CPU cores
- RDMA network

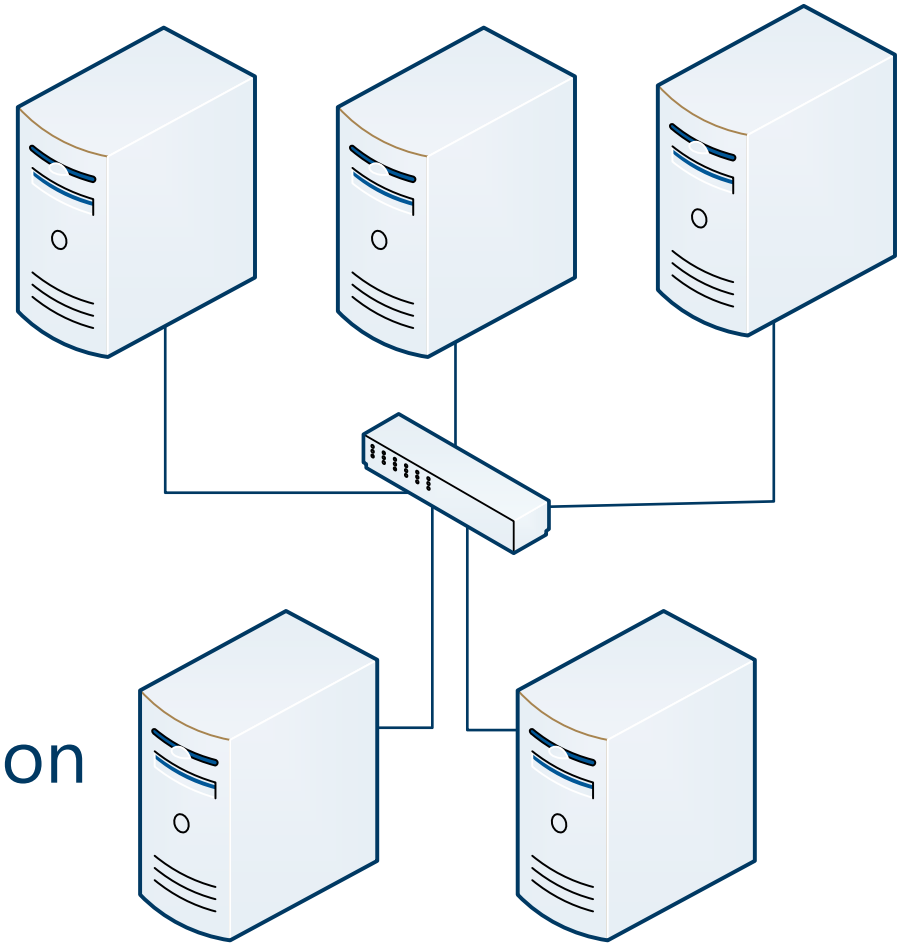
Desirable:

- Keep data in memory
- Access data using RDMA
- Collocate data and computation



Traditional model

Servers: store data

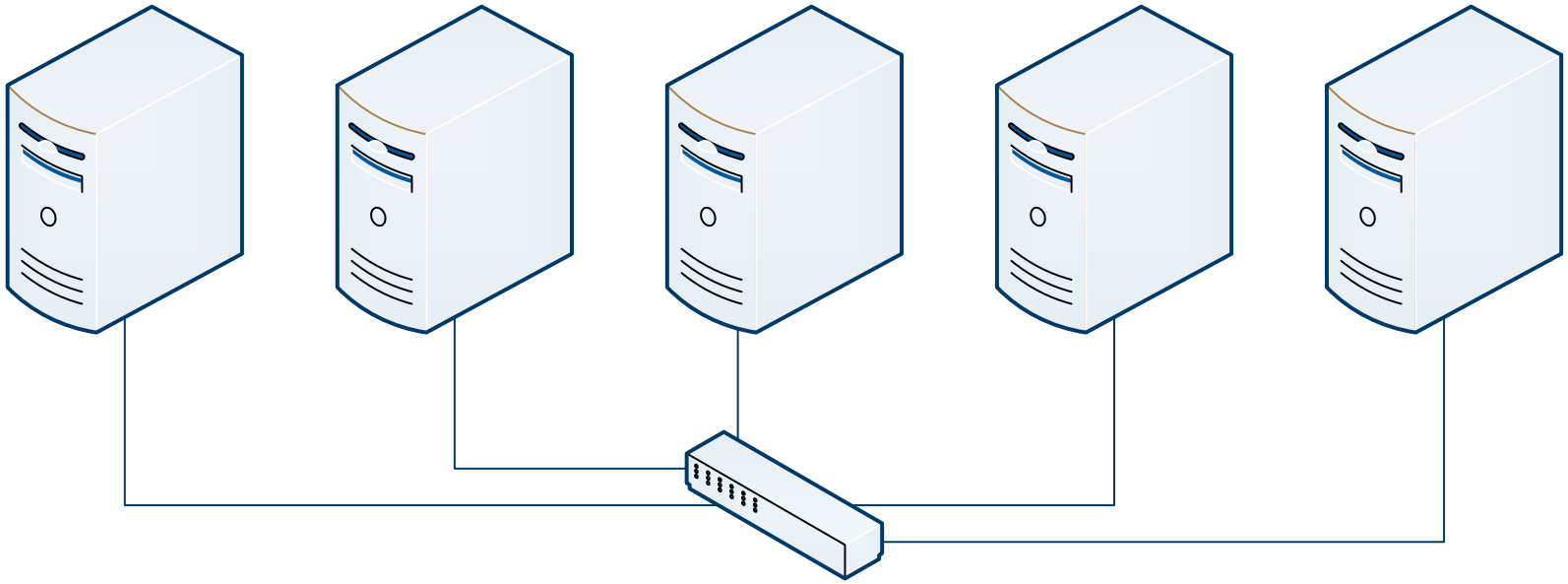


Clients: execute application

Symmetric model

Access to local memory is much faster

Server CPUs are mostly idle with RDMA

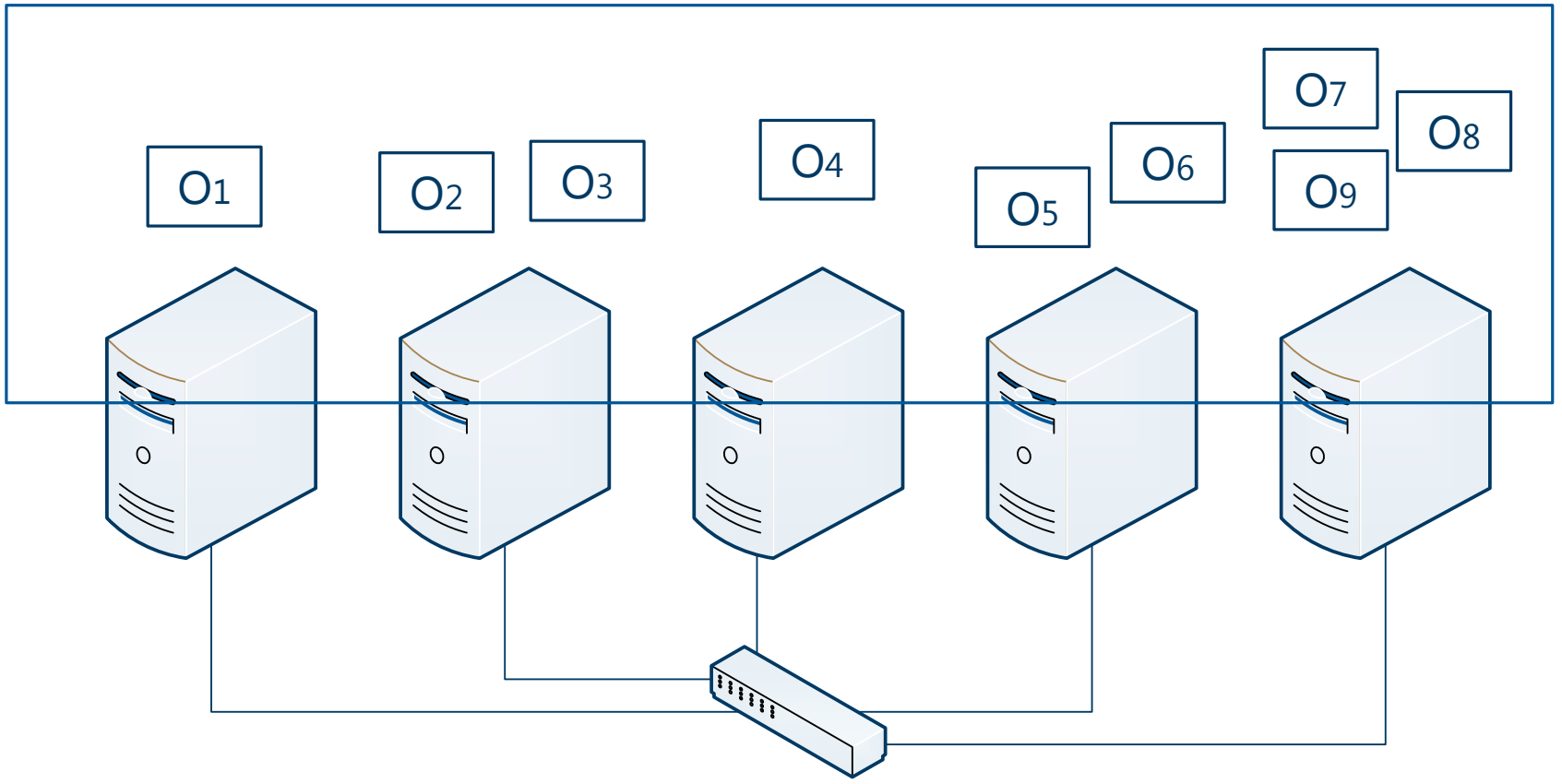


Machines store data and execute application

Shared address space

Supports direct RDMA of objects

Programmability a welcome bonus

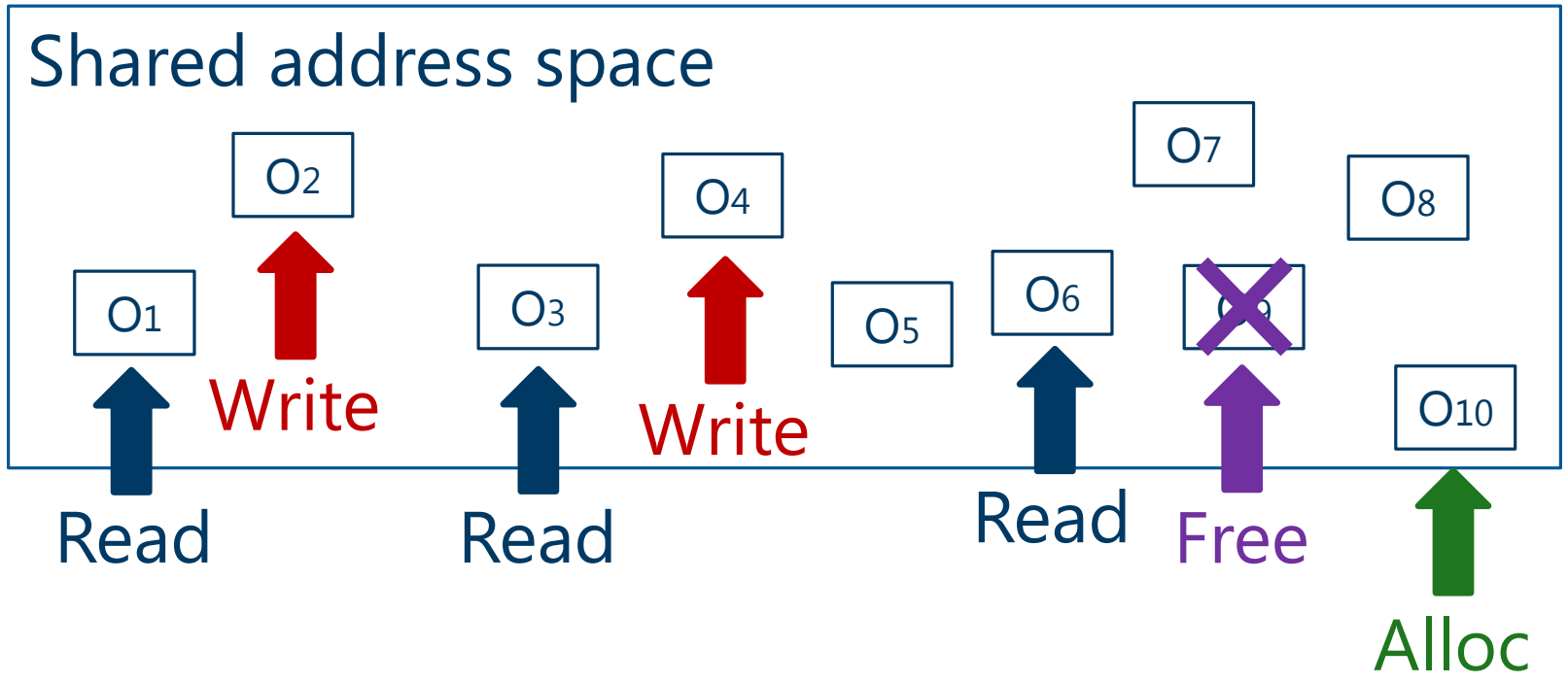


Transactions: simplify programming

General primitive

Strong consistency: serializability

Transparent:
- location
- concurrency
- failures



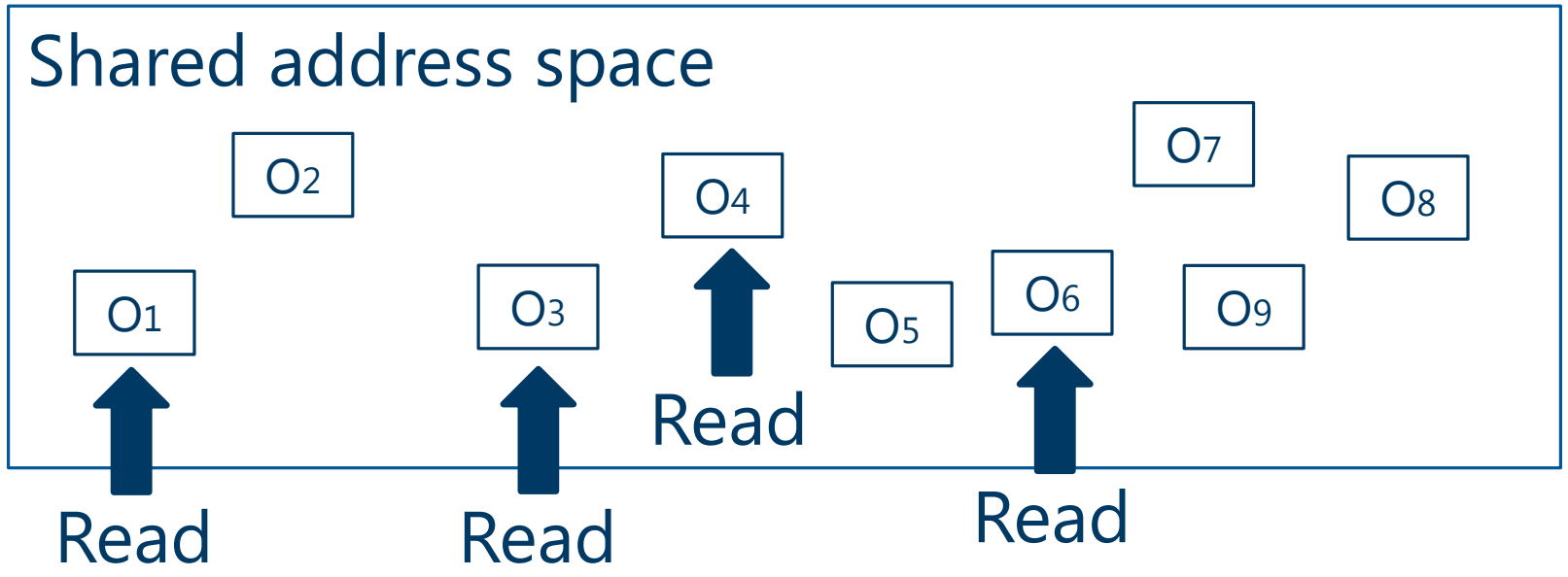
Atomic execution of multiple operations

Optimizations: lock-free reads

Efficient: read is a single RDMA

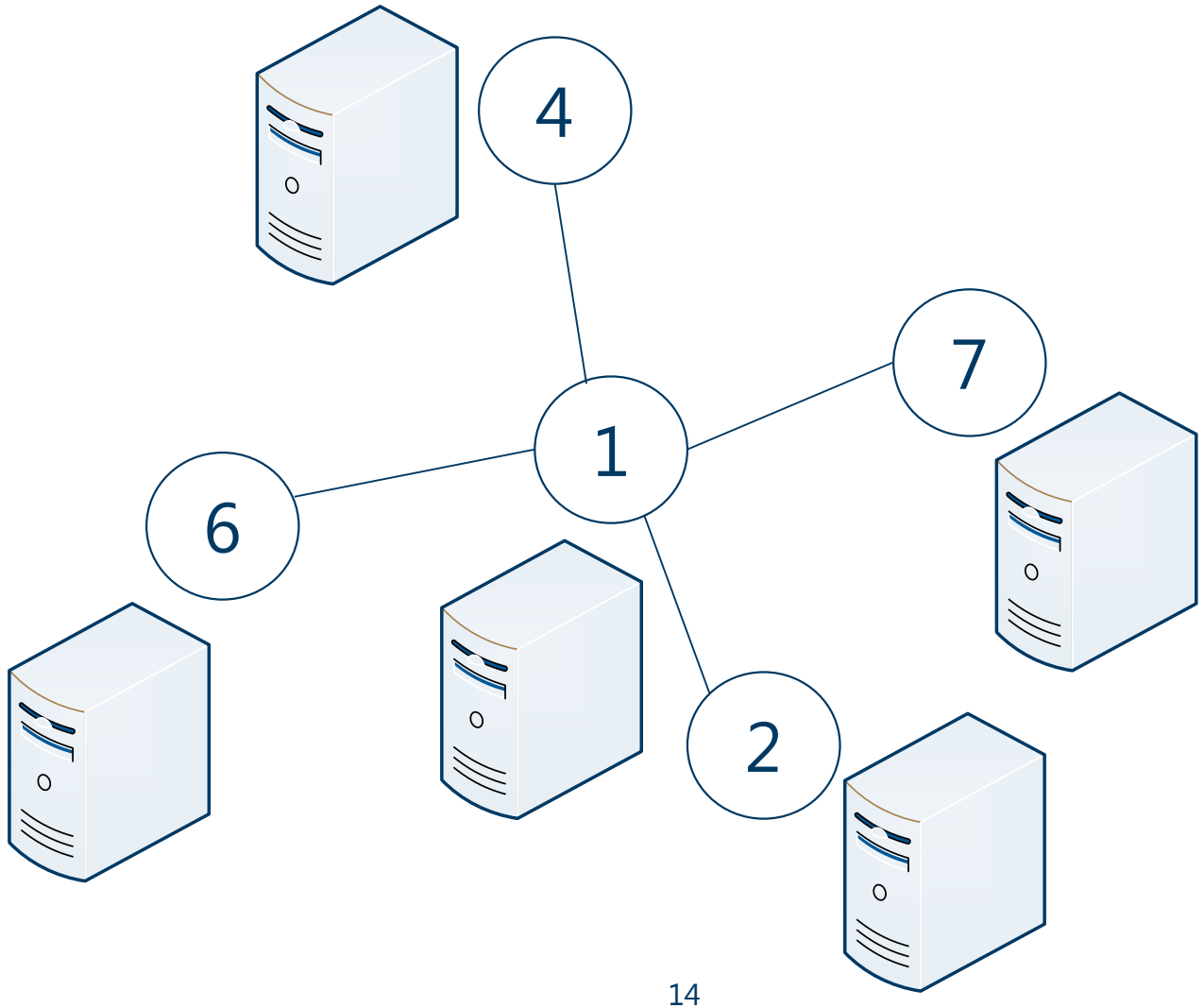
Strong consistency: serializable

Harder to compose: custom validation



Atomic execution of a single read

Optimizations: locality awareness

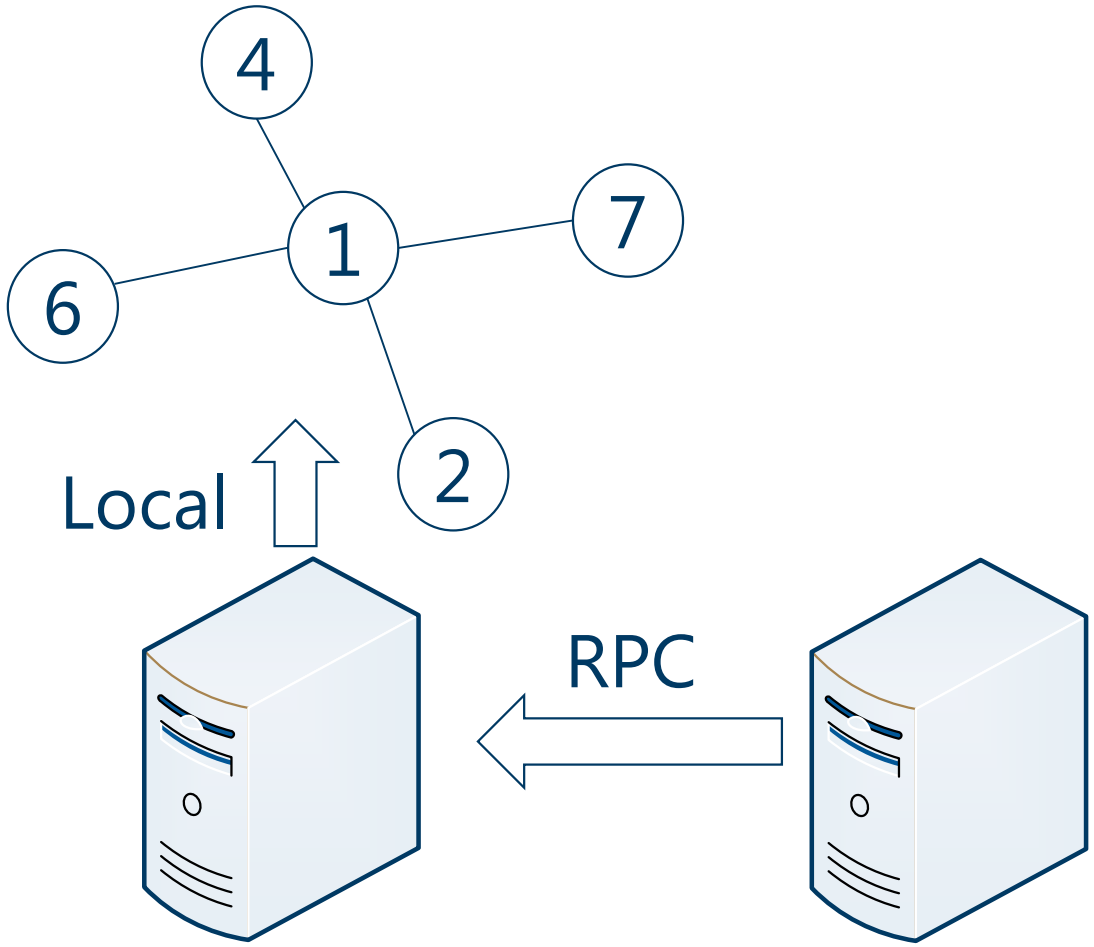


Optimizations: locality awareness

Collocate data accessed together

Ship computation to target data

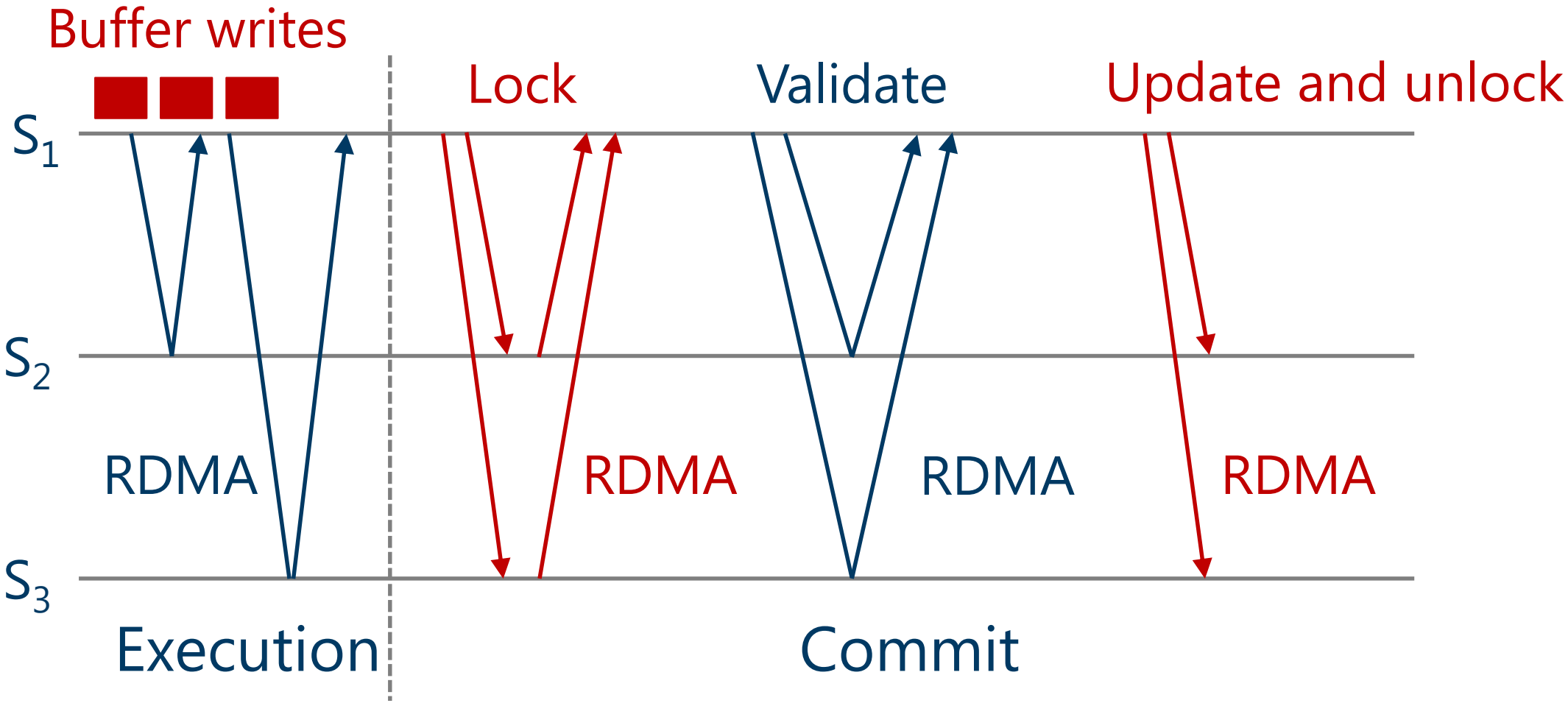
Optimized single-server transactions



Paper

- RDMA communication
- Programming model
- Address space management
- Transactions and lock-free operations 
- Hashtable

Transactions



Traditional lock-free reads

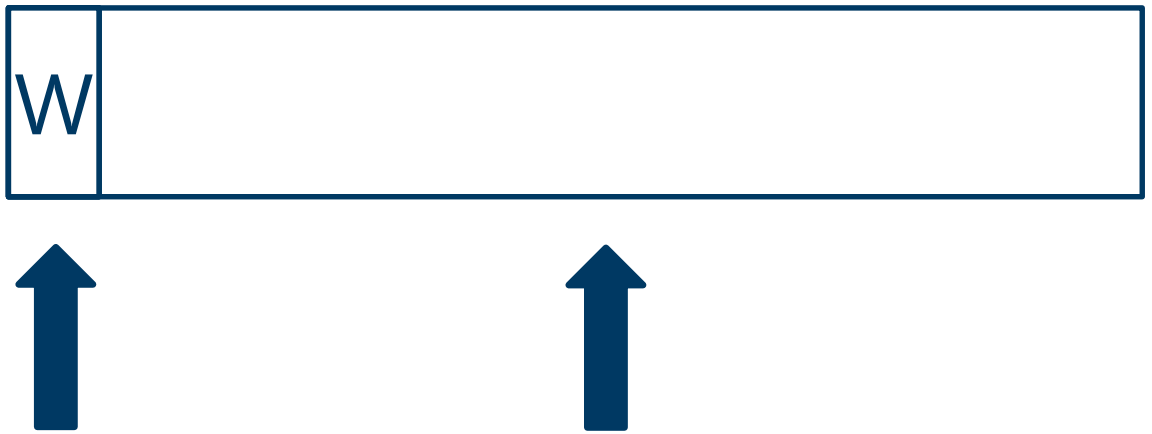
Header version

Update in 3 steps:

1. Lock
2. Update data
3. Unlock and increment

Traditional lock-free reads

Header version



Read in 3 steps:

1. Read version
2. Read data
3. Read version

Consistent if versions in steps 1. and 3. are equal

Traditional lock-free reads

Header version

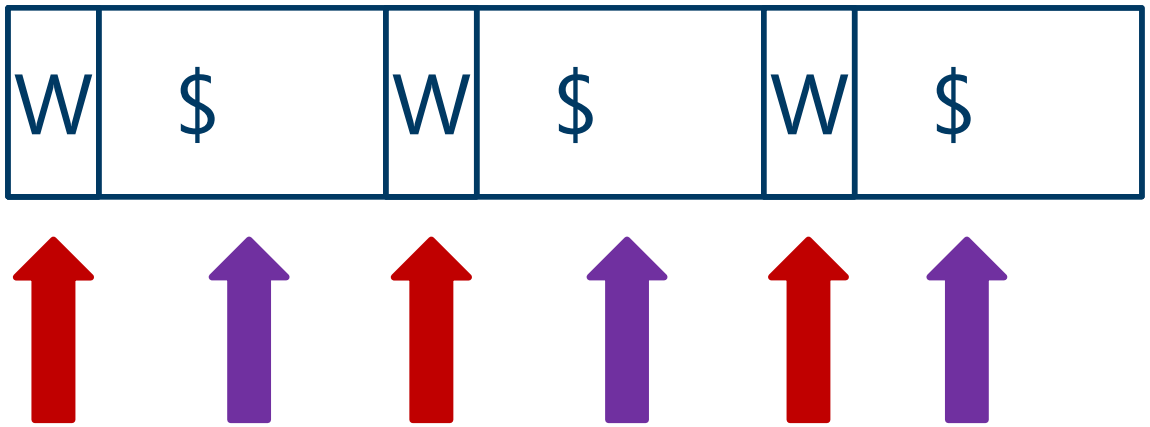
64-bit version to avoid overflow

Problem: read requires three network accesses, so it is not well suited to RDMA

FaRM lock-free reads

Header version

Use cache-line versions



1. Lock versions
2. Update data
3. Unlock and increment

FaRM lock-free reads

Header version

Cache-line versions

One RDMA read of the whole object, check that all versions are equal

FaRM lock-free reads

Space efficiency: 16-bit cache-line versions

To ensure cache line versions don't overflow, measure read time and discard it too long

t_{update_min} = 40 ns

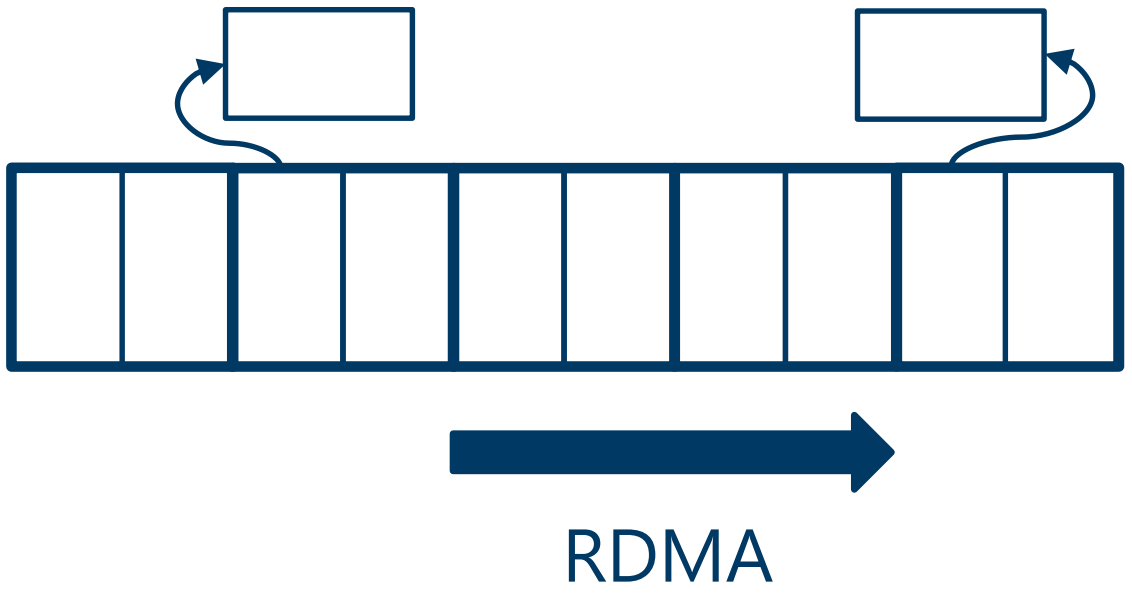
t_{read_max} = 40 ns * 2^{16} * (1 – ε) = 2 ms

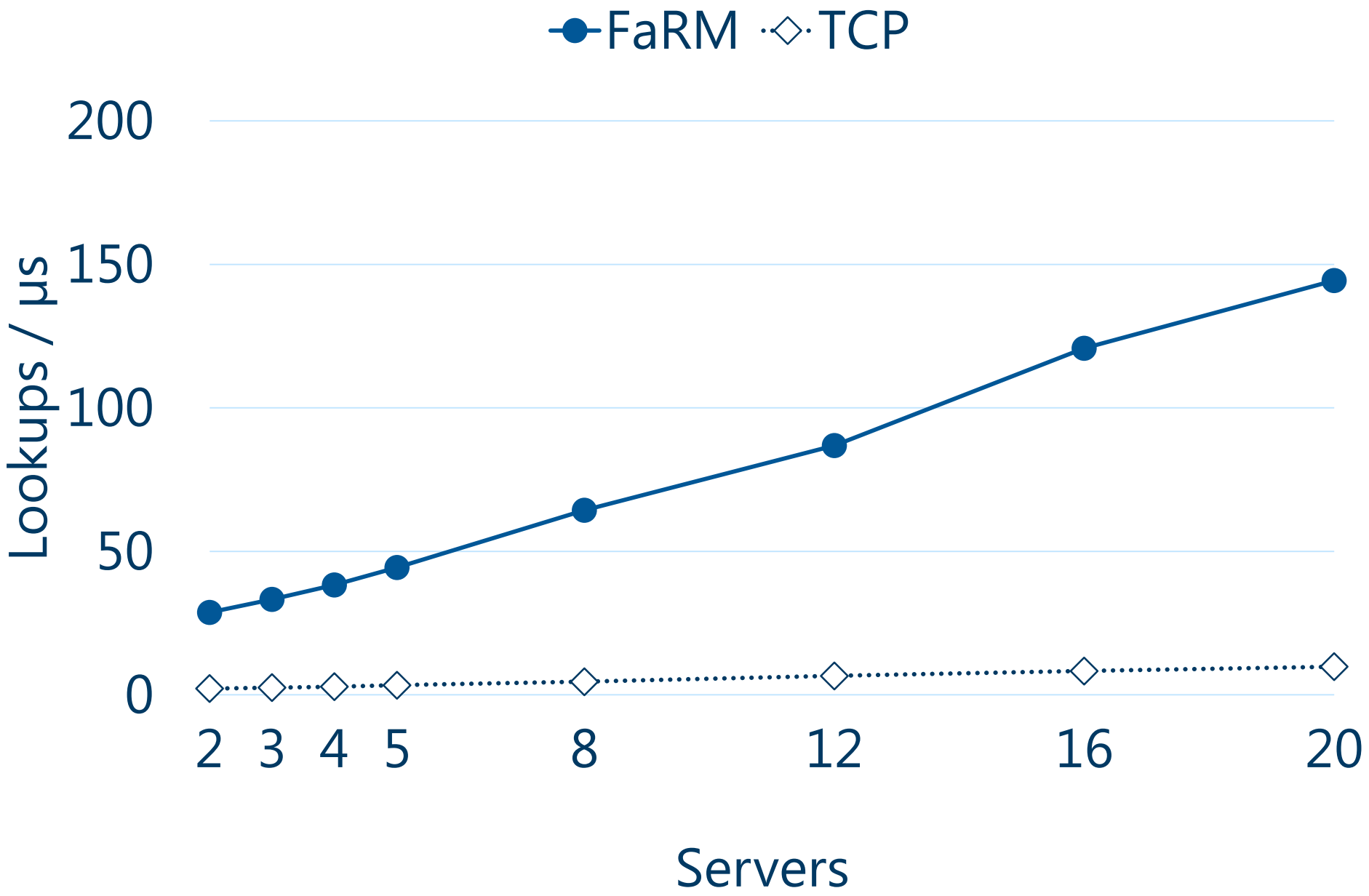
Paper

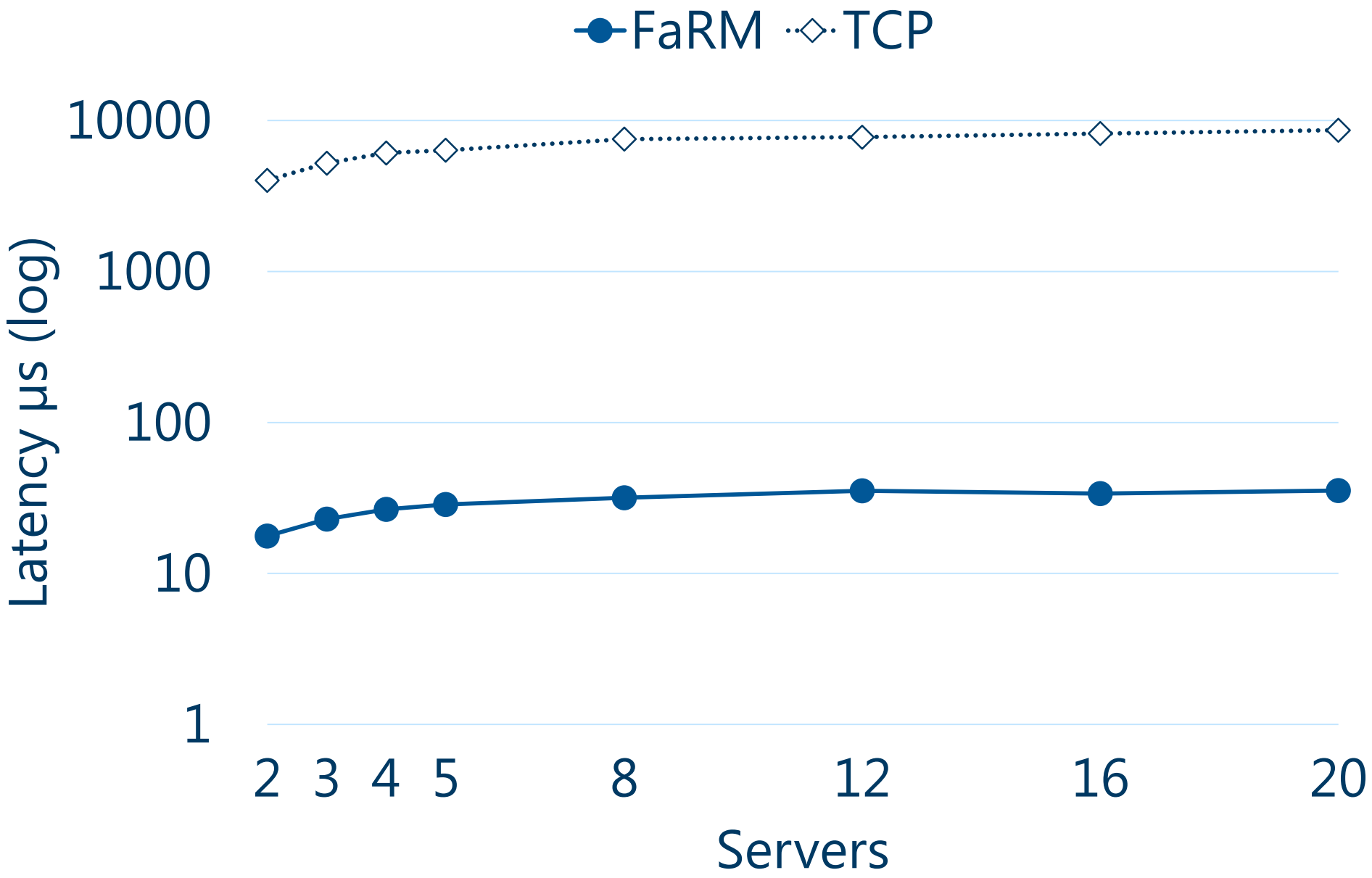
- RDMA communication
- Programming model
- Address space management
- Transactions and lock-free operations
- Hashtable ⟸

FaRM hashtable

- Important building block
  - FaRM makes it possible to easily try out different designs
- Optimized for lookups
  - One RDMA in the common case
- Good space utilization




FaRM
TCP
200
150
100
50
0
Lookups / µs
2
3
4
5
8
12
16
20
Servers


FaRM
TCP
10000
1000
100
10
1
Latency µs (log)
2
3
4
5
8
12
16
20
Servers

TAO [Bronson '13, Armstrong '13]

- Facebook's in-memory graph store
- Workload
  - Read-dominated (99.8%)
  - 10 operation types
- FaRM implementation
  - Nodes and edges are FaRM objects
  - Lock-free reads for lookups
  - Transactions for updates

6 Mops/s/srv
(10x improvement)

42 µs average latency
(40 – 50x improvement)

FaRM

- Platform for distributed computing
  - Data is in memory
  - RDMA
- Shared memory abstraction
  - Transactions
  - Lock-free reads
- Order-of-magnitude performance improvements
  - Enables new applications