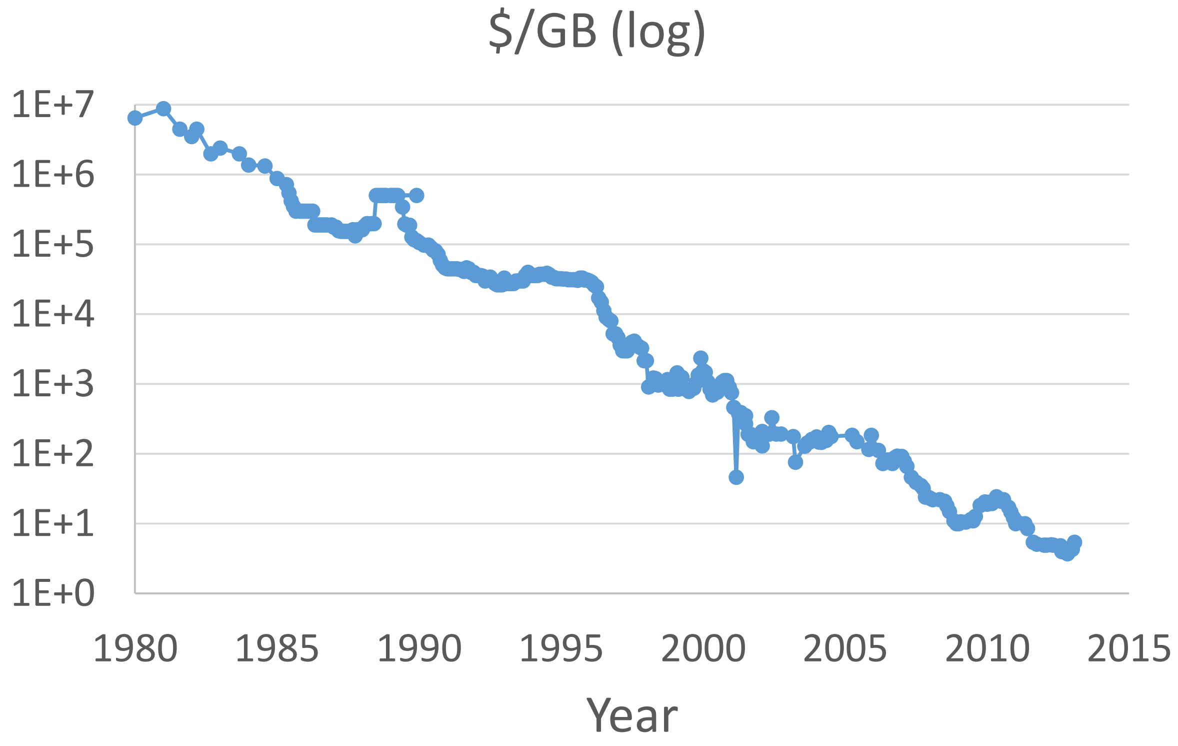
FaRM: Fast Remote Memory

Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, Miguel Castro

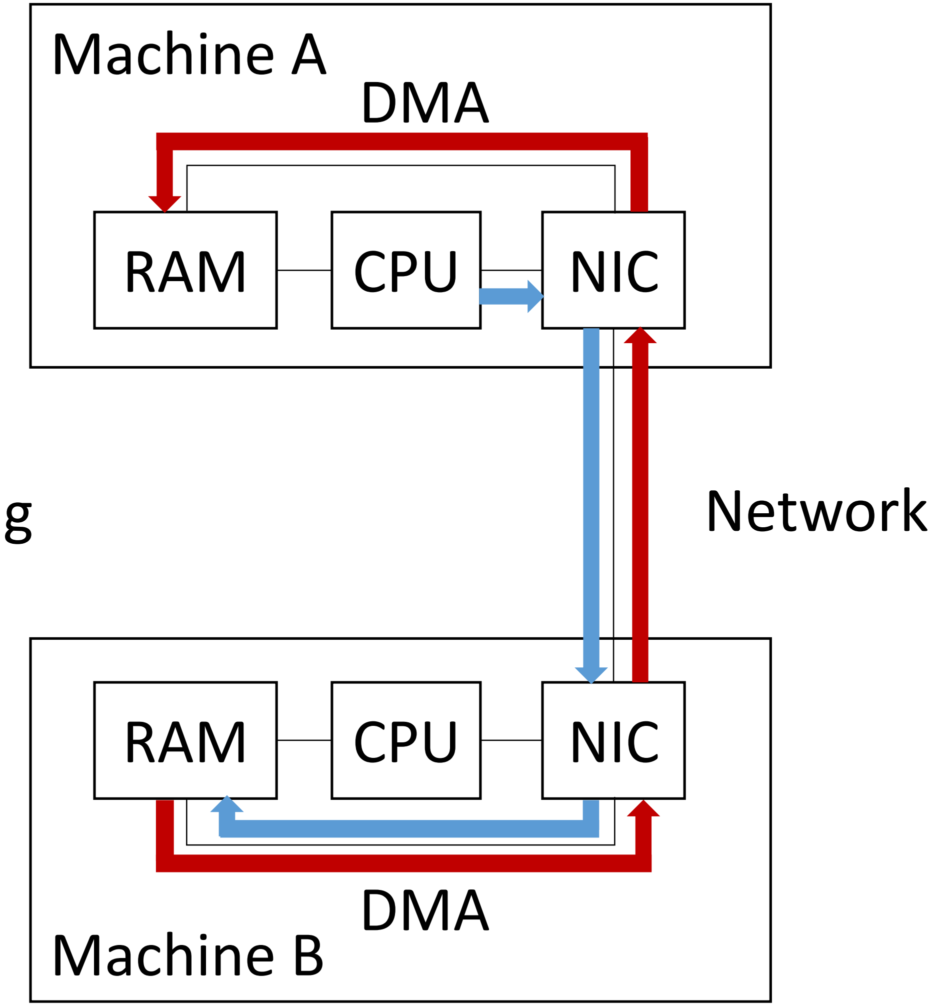
Hardware trends

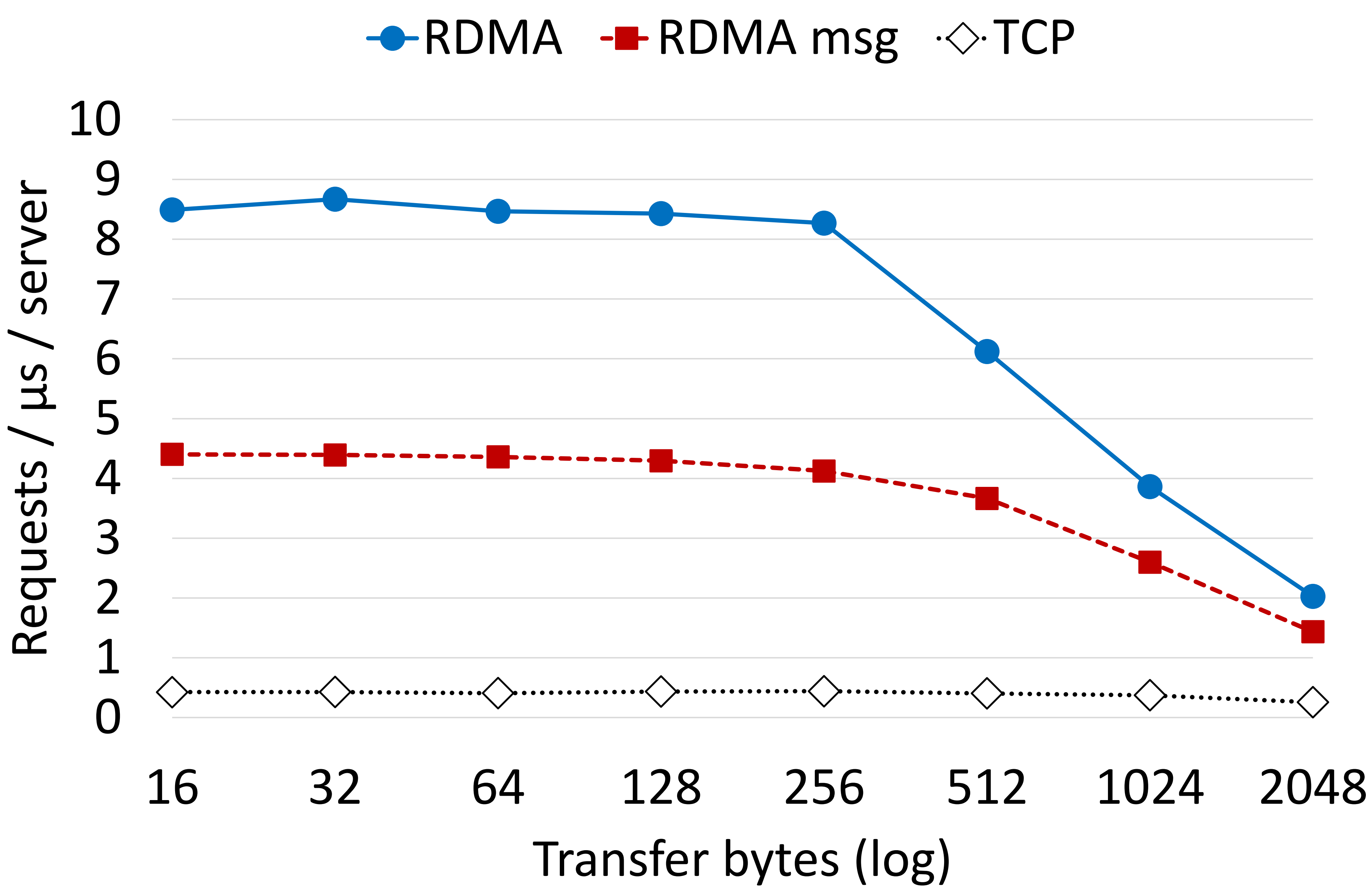
- Main memory is inexpensive
  - 100 GB – 1 TB per server
  - 10 – 100 TBs in a small cluster
- New data centre networks
  - 40 Gbps throughput (100 this year)
  - 1-3 µs latency
  - RDMA primitives



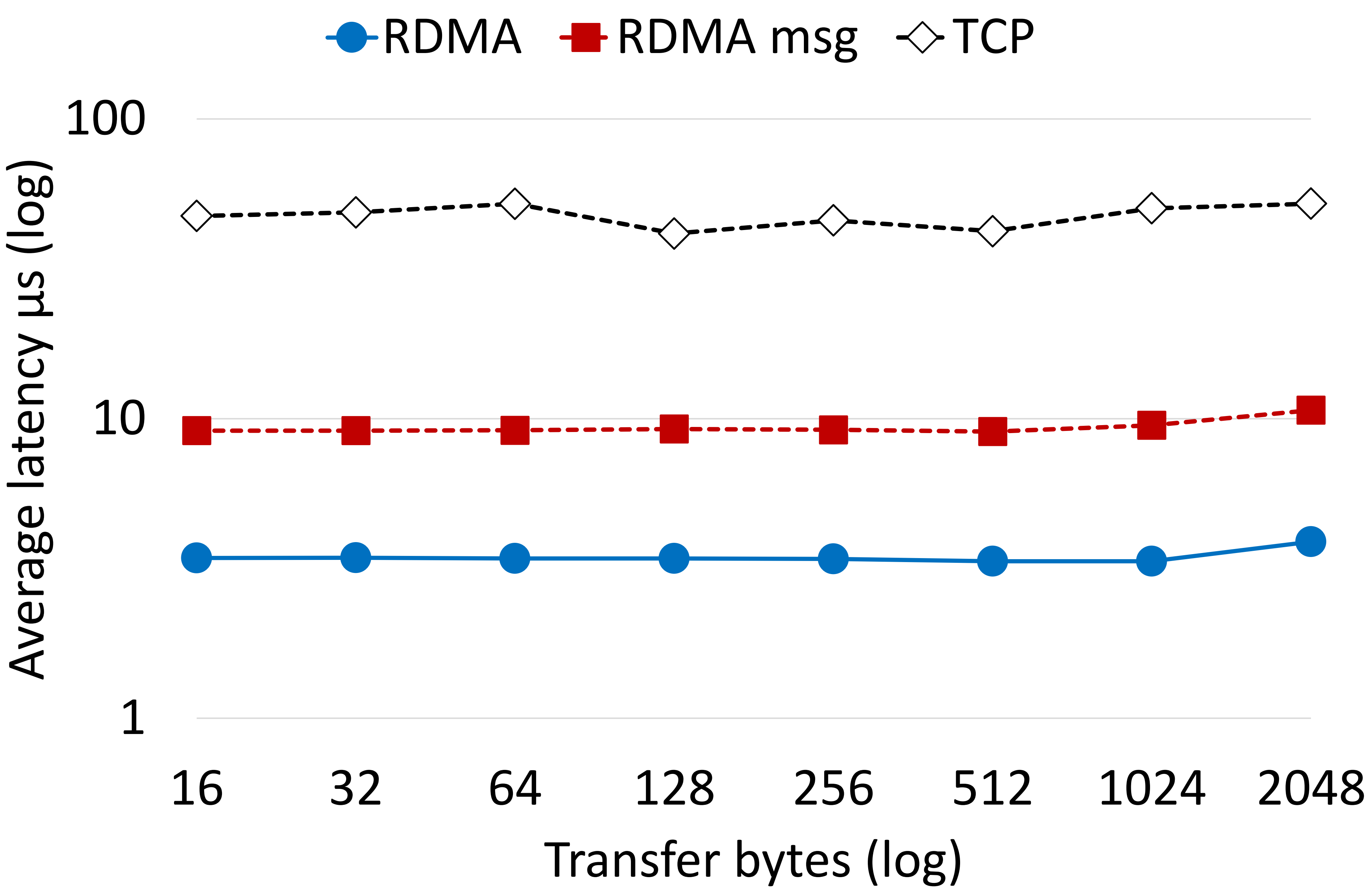
Remote direct memory access

- Read and write remote memory
  - NIC performs DMA requests
  - Remote CPU not involved
- We use RDMA extensively
  - Reads for directly reading data
  - Writes into remote buffers for messaging
- Great performance
  - Bypasses kernel
  - Bypasses remote CPU





RDMA
RDMA msg
TCP
Requests / µs / server
10
9
8
7
6
5
4
3
2
1
0
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
  - Low latency
- Data serving
  - Graph store
  - Key value-store
- Enabling new applications

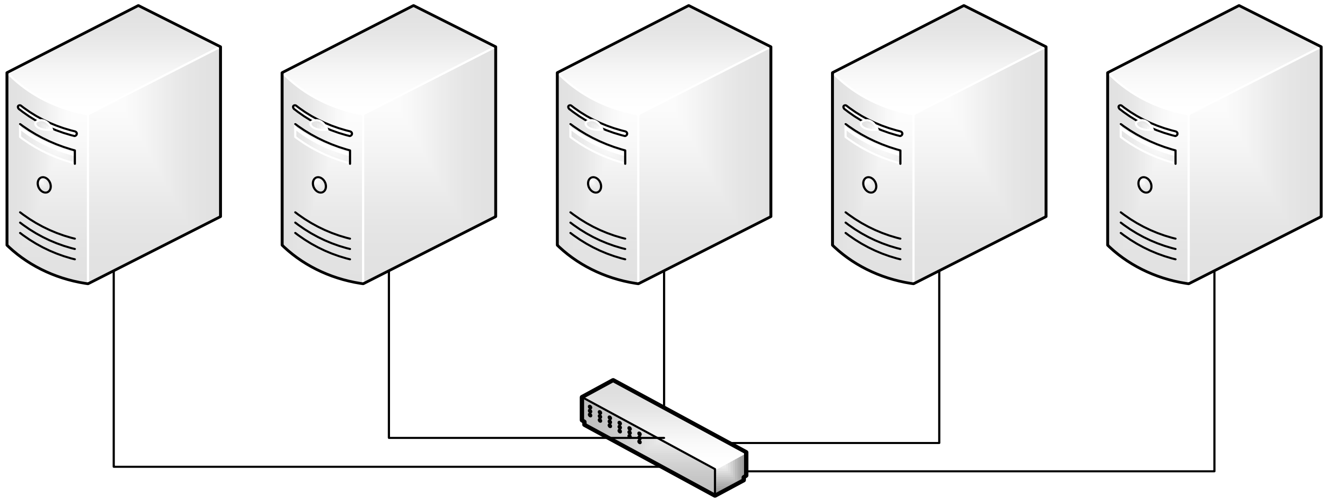
Outline

- FaRM programming model
- Design
  - Synchronization
  - Hashtable
- Experimental results
- Future work

How to program a modern cluster?

We have:
- TBs of DRAM
- 100s of CPU cores
- RDMA network

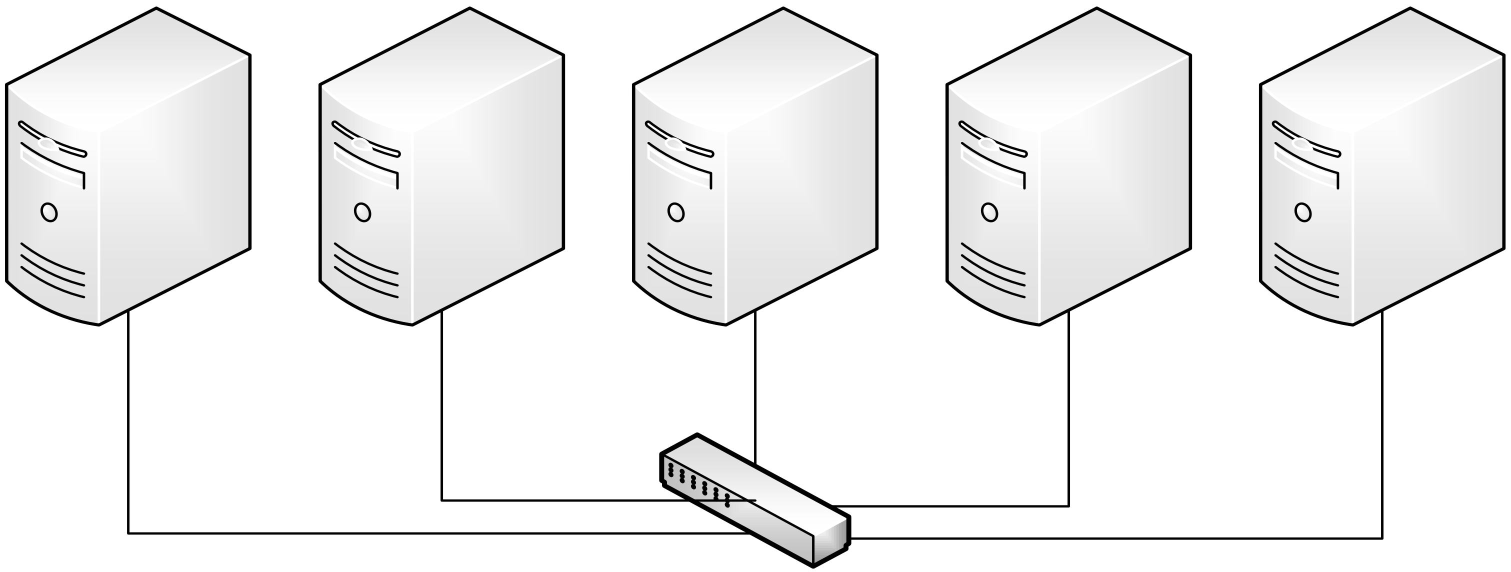
Desirable:
- Keep data in memory
- Access data using RDMA
- Collocate data and computation



Symmetric model

Access to local memory is much faster

Server CPUs are mostly idle with RDMA

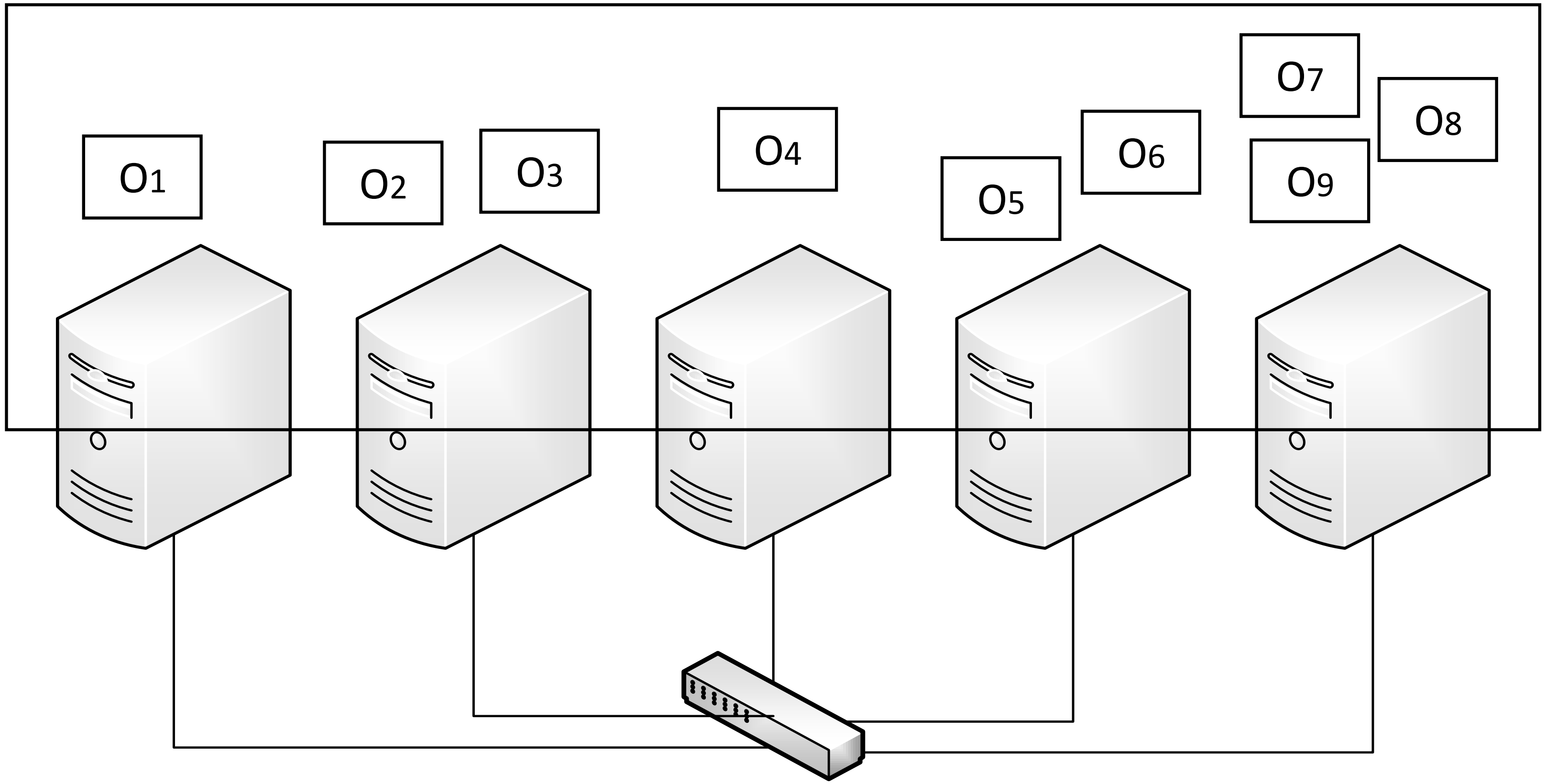


Machines store data and execute application

Shared address space

Supports direct RDMA of objects

Programmability a welcome bonus



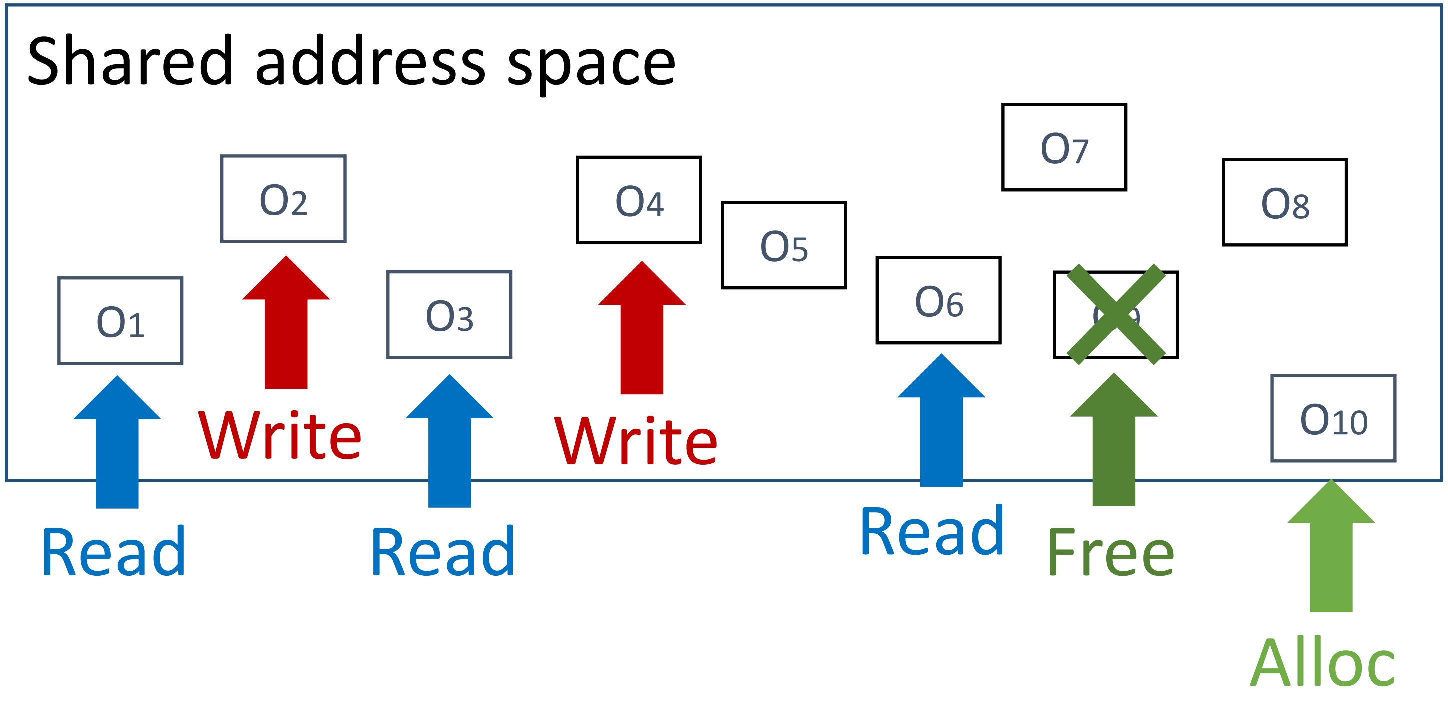
Transactions: simplify programming

General primitive

Strong consistency: serializability

Transparent:

- location
- concurrency
- failures



Atomic execution of multiple operations

FaRM API: transactions

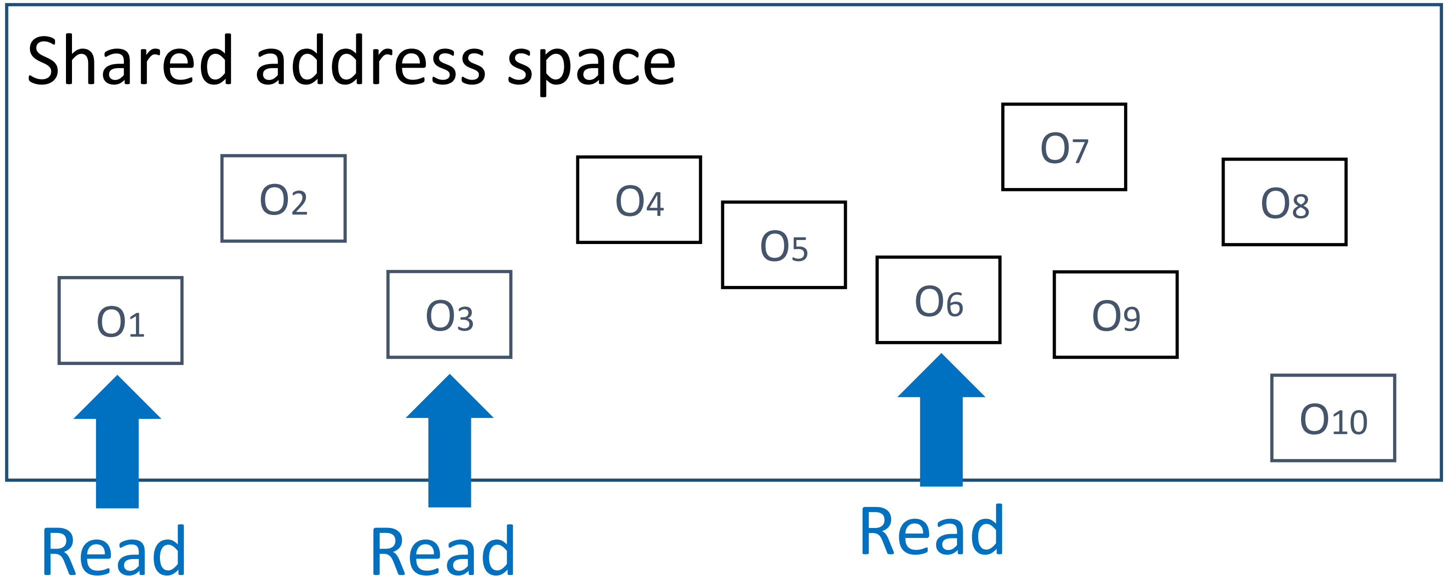
```
Tx *TxStart();

Addr TxAlloc(Tx *tx, int size, Addr hint);
void TxFree(Tx *tx, Addr addr);

ObjBuf *TxRead(Tx *tx, Addr addr, int size);
ObjBuf *TxOpenForWrite(Tx *tx, ObjBuf *obj);

bool TxCommit(Tx *tx);
```

Optimizations: lock-free reads

Efficient: read is a single RDMA

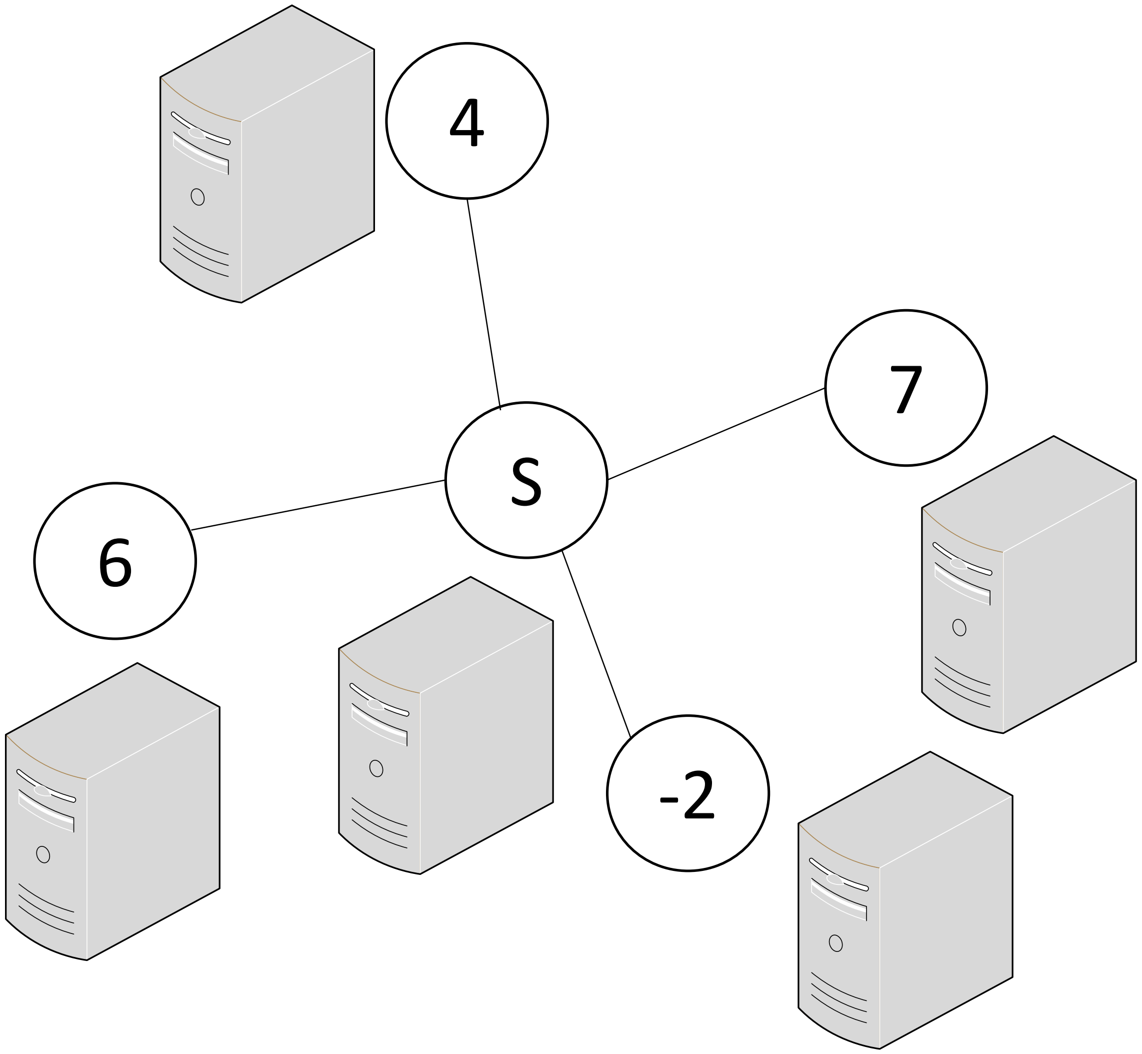
Strong consistency: serializability

Harder to compose: custom validation



Atomic execution of a single read

Optimizations: locality awareness

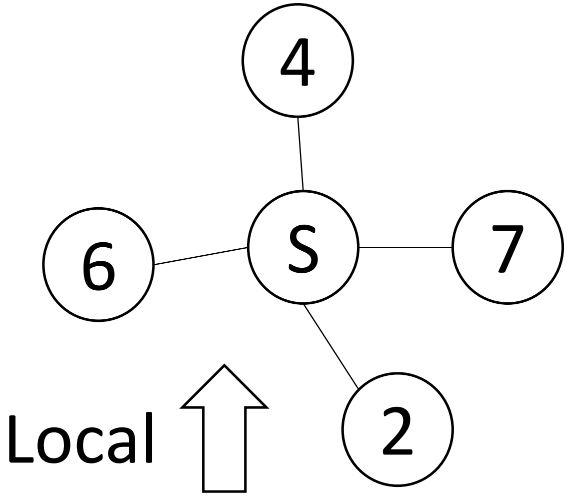


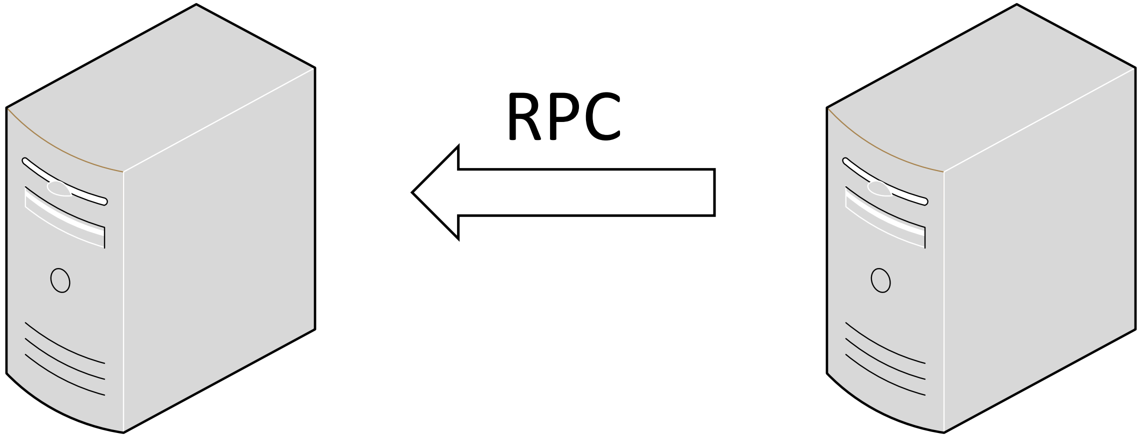
Optimizations: locality awareness

Collocate data accessed together

Ship computation to target data

Optimized single-server transactions

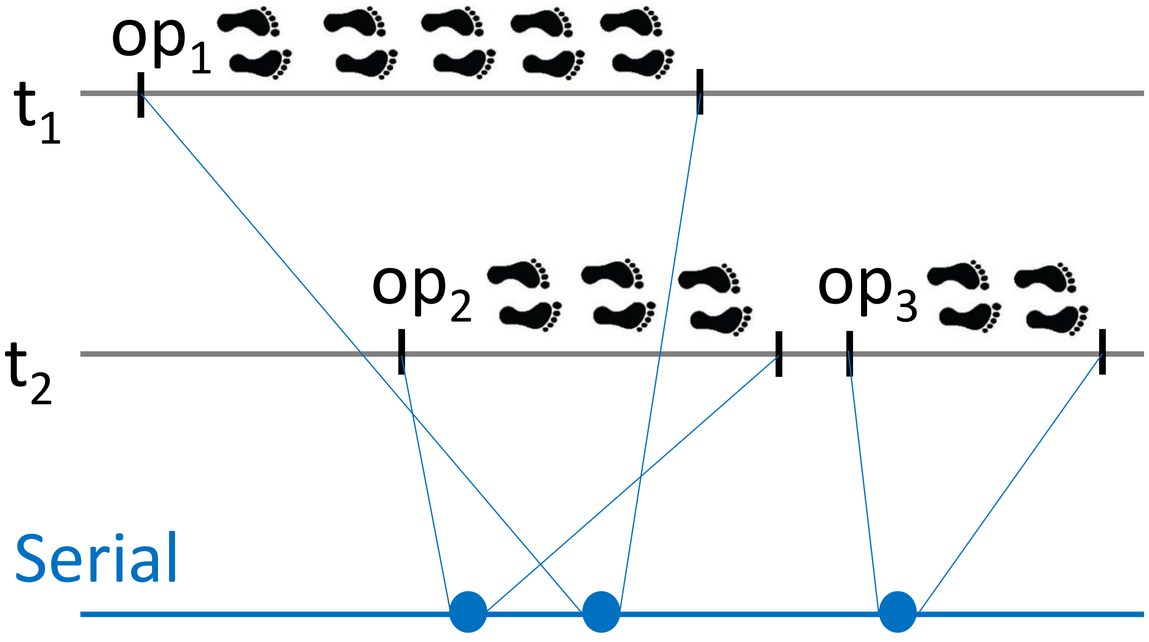





```
Addr TxAlloc(Tx *tx,
             int size,
             Addr hint);
void SendMsg(Addr a,
             Msg *m);
```

Consistency model

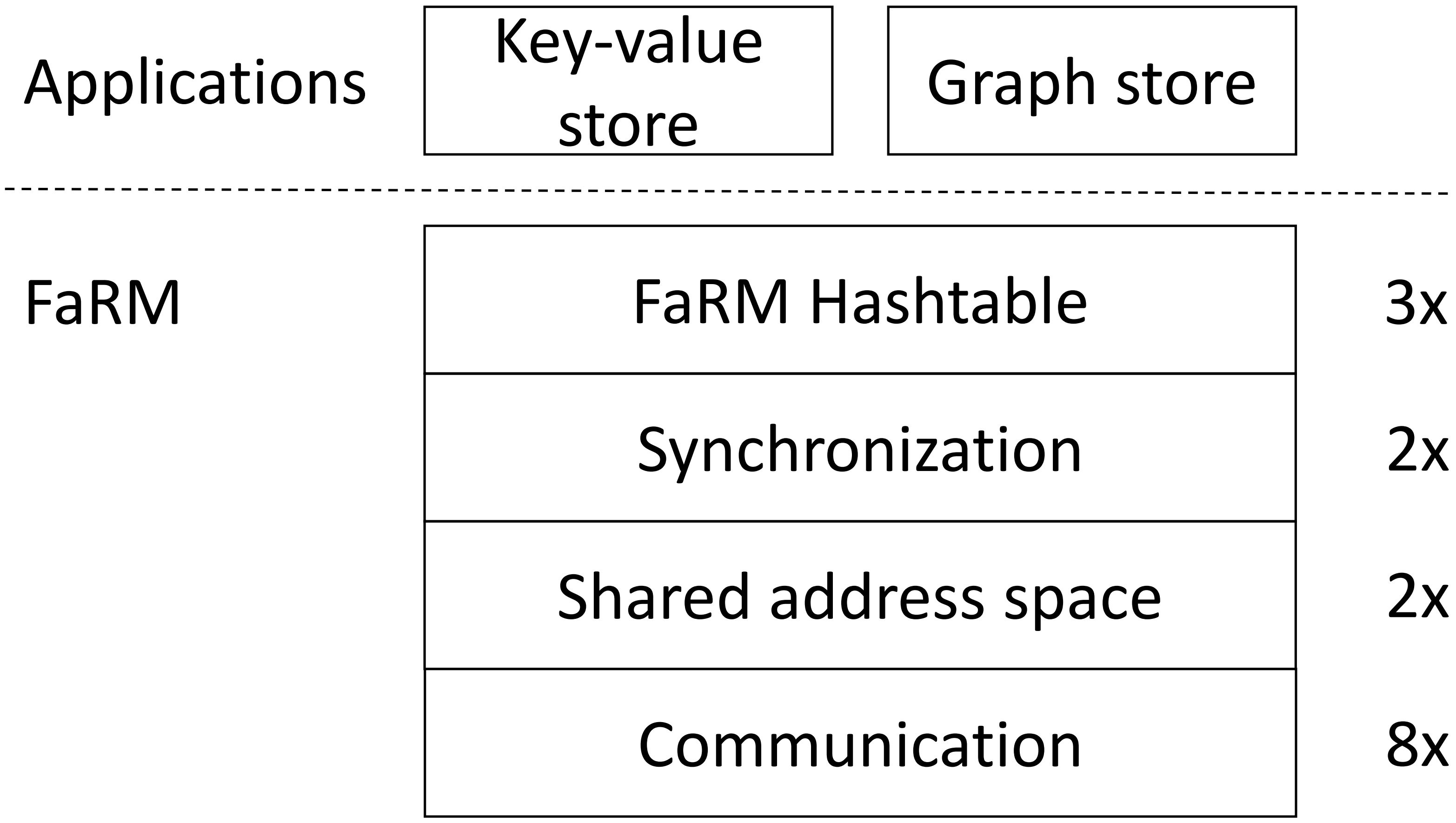
- Strong consistency
  - Strict serializability for transactions
  - Linearizability for data structures
- Weak timing assumptions
  - Eventual synchrony
  - Bounded clock drift



Outline

- FaRM programming model
- Design
  - Synchronization
  - Hashtable
- Experimental results
- Future work

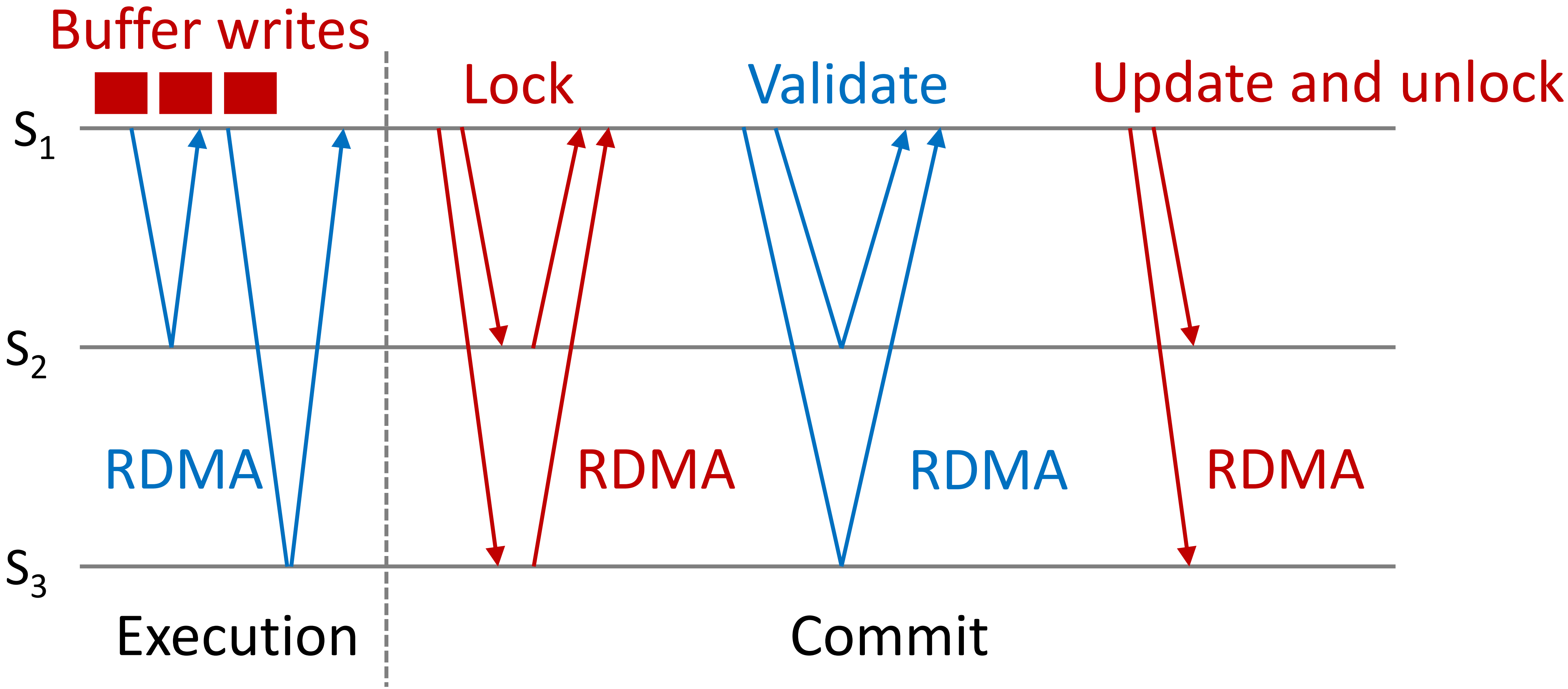
FaRM runtime



24x better than published RDMA key-value store

10x-40x better than TCP state-of-the-art key-value store

Transactions

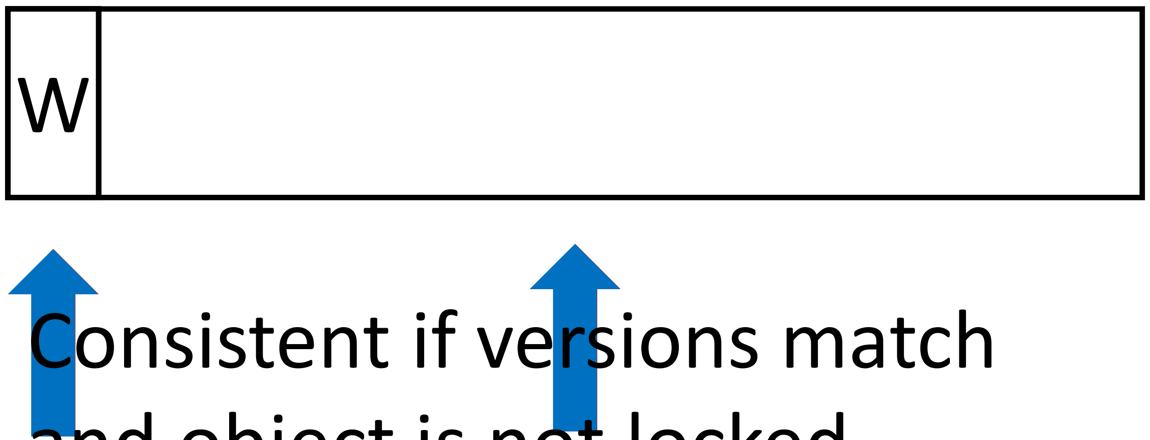


Lock-free reads

- Transactions can be expensive
  - Require many messages
- FaRM exposes lock-free reads
  - Consistent object state
  - One RDMA operation
- Strictly serializable with transactions
  - Equivalent to a one-read transaction

Lock-free reads

Header version



64-bit version to avoid overflow

Consistent if versions match and object is not locked

Unlock and increment

Read version

Lock

Update

Read data

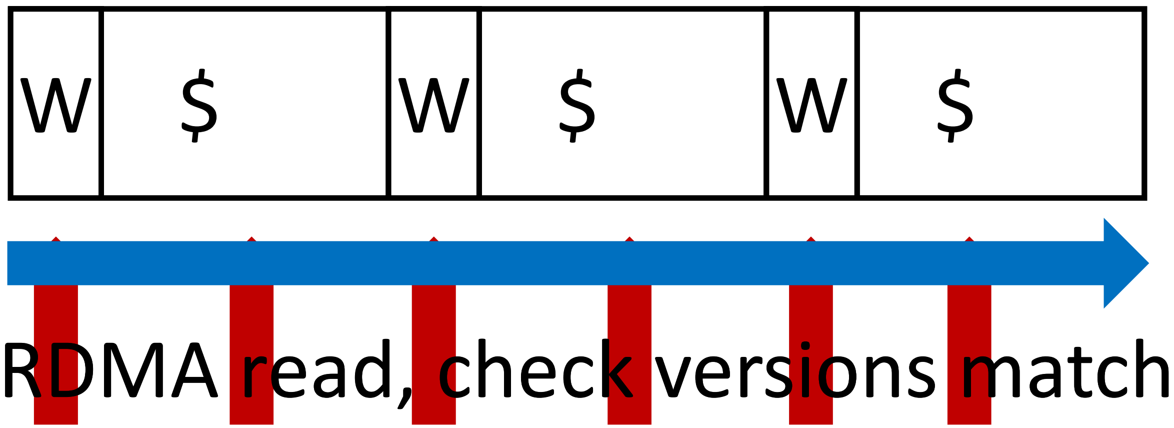
Update

Read

Read requires three network accesses

FaRM lock-free reads

Header version



Space efficiency:
16-bit cache-line
versions

Cache-line versions

RDMA read, check versions match and read does not take too long

Unlock and increment

Lock update

t_{update_min} = 40 ns

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

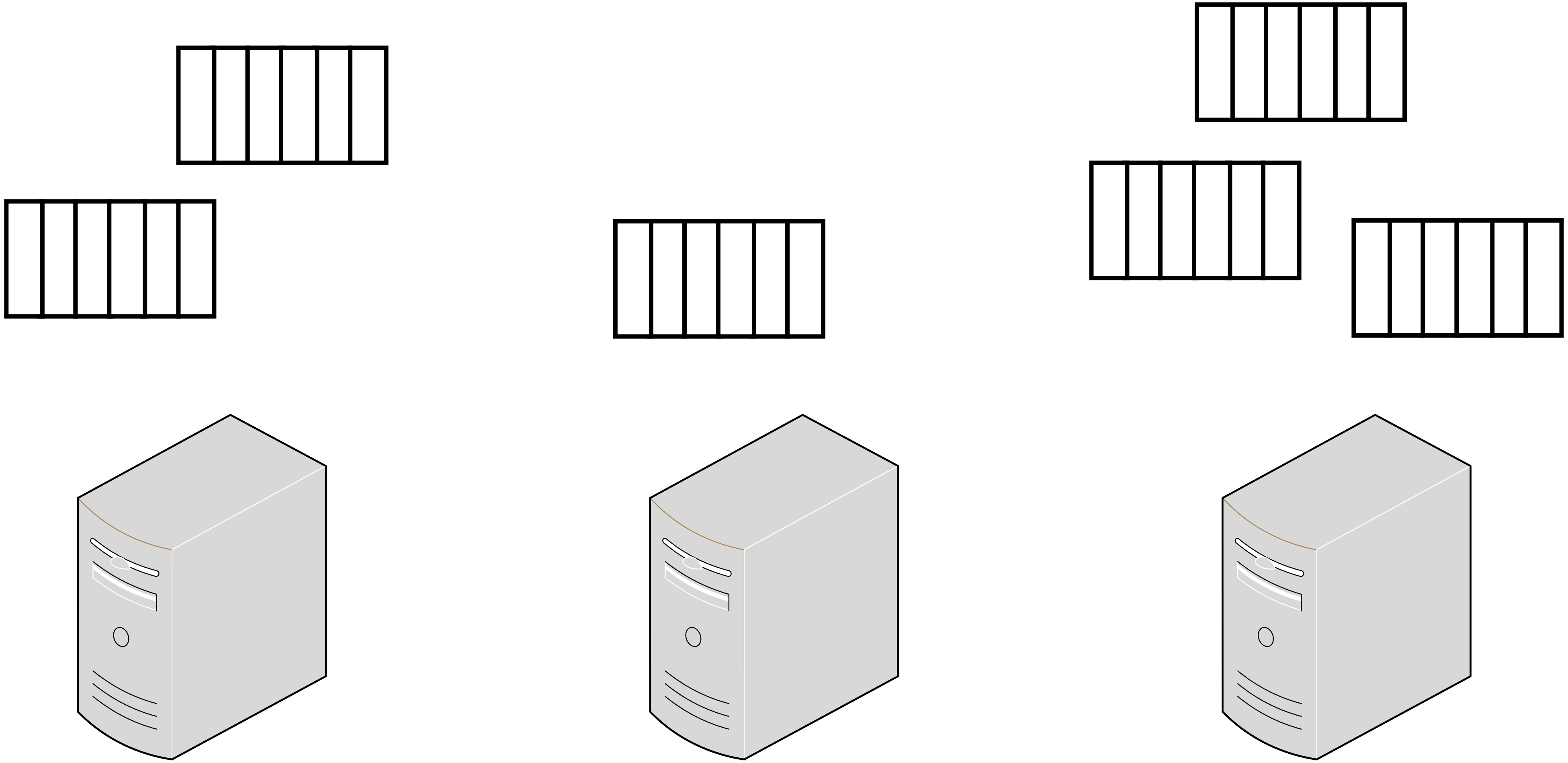
Outline

- FaRM programming model
- Design
  - Synchronization
  - Hashtable
- Experimental results
- Future work

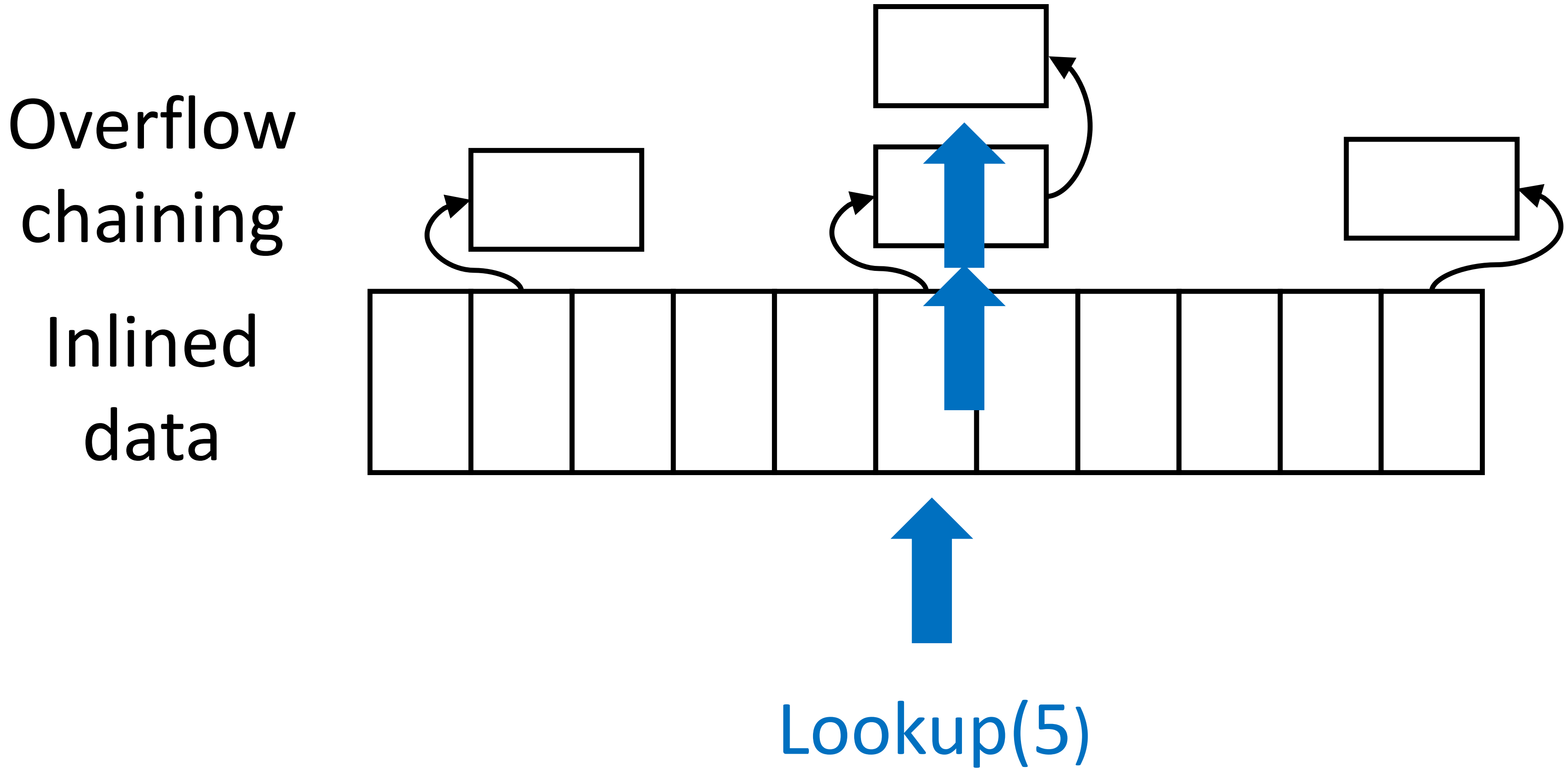
FaRM hashtable

- Optimize for lookups
  - Majority of accesses are lookups
  - Goal: lookup with a single RDMA read
- Update with transactions
  - Simplifies updates
  - Performance: ship updates to data owner
- Correctness
  - Goal: linearizability

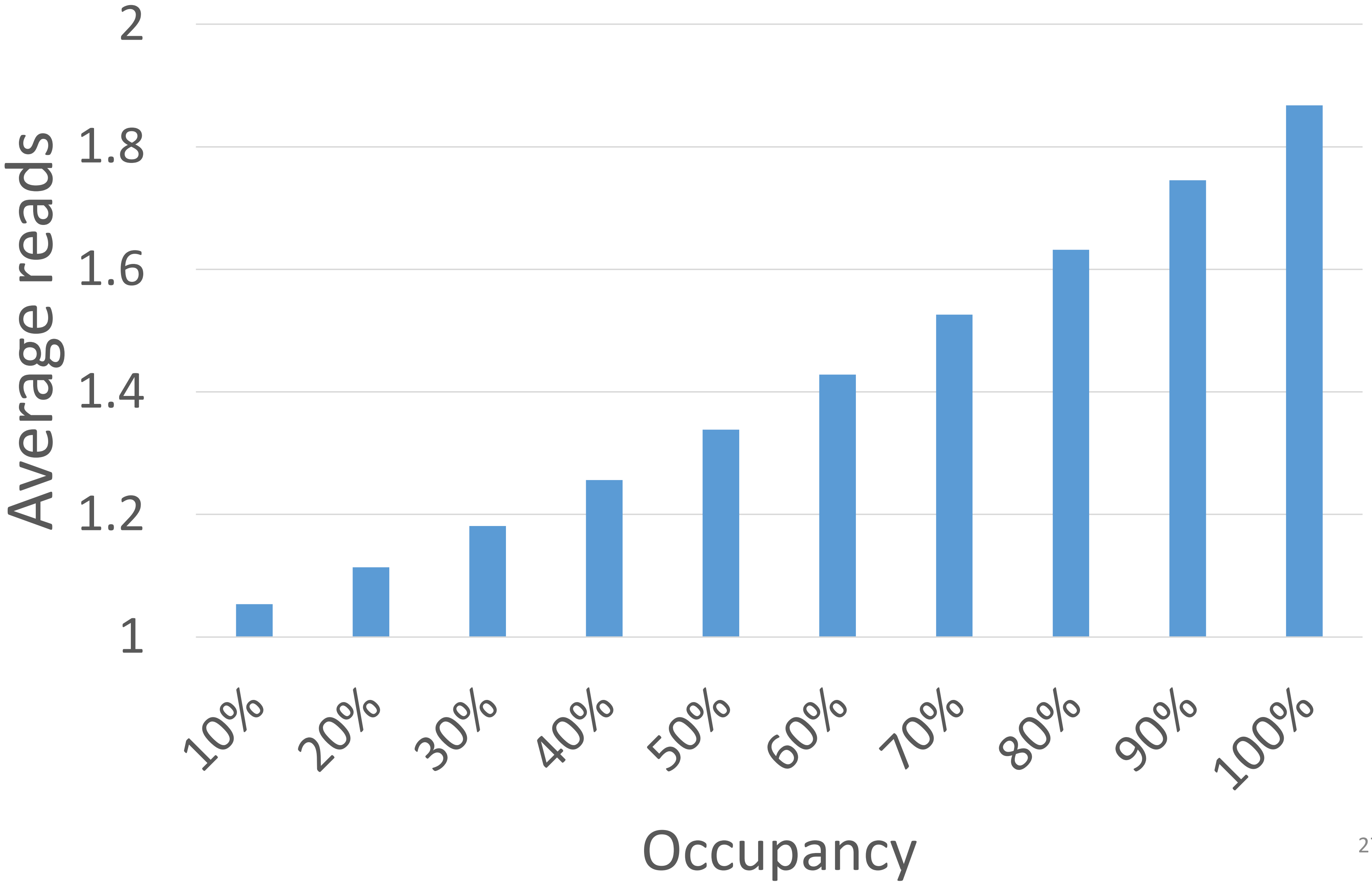
Distributed hashtable



First attempt: chaining

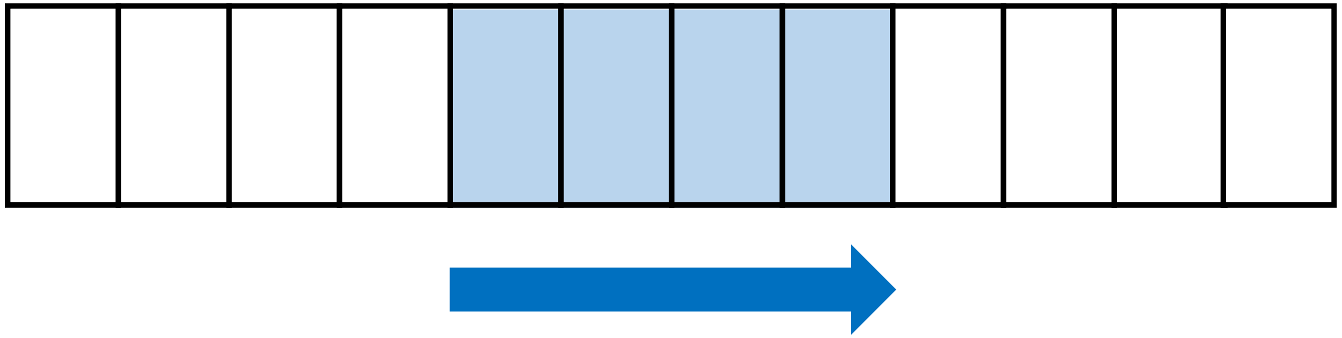


One read in the common case. Not quite.


2
1.8
1.6
1.4
1.2
1
Average reads
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
Occupancy

Hopscotch hashtable [Herlihy '08]

Invariant:
element in
neighbourhood



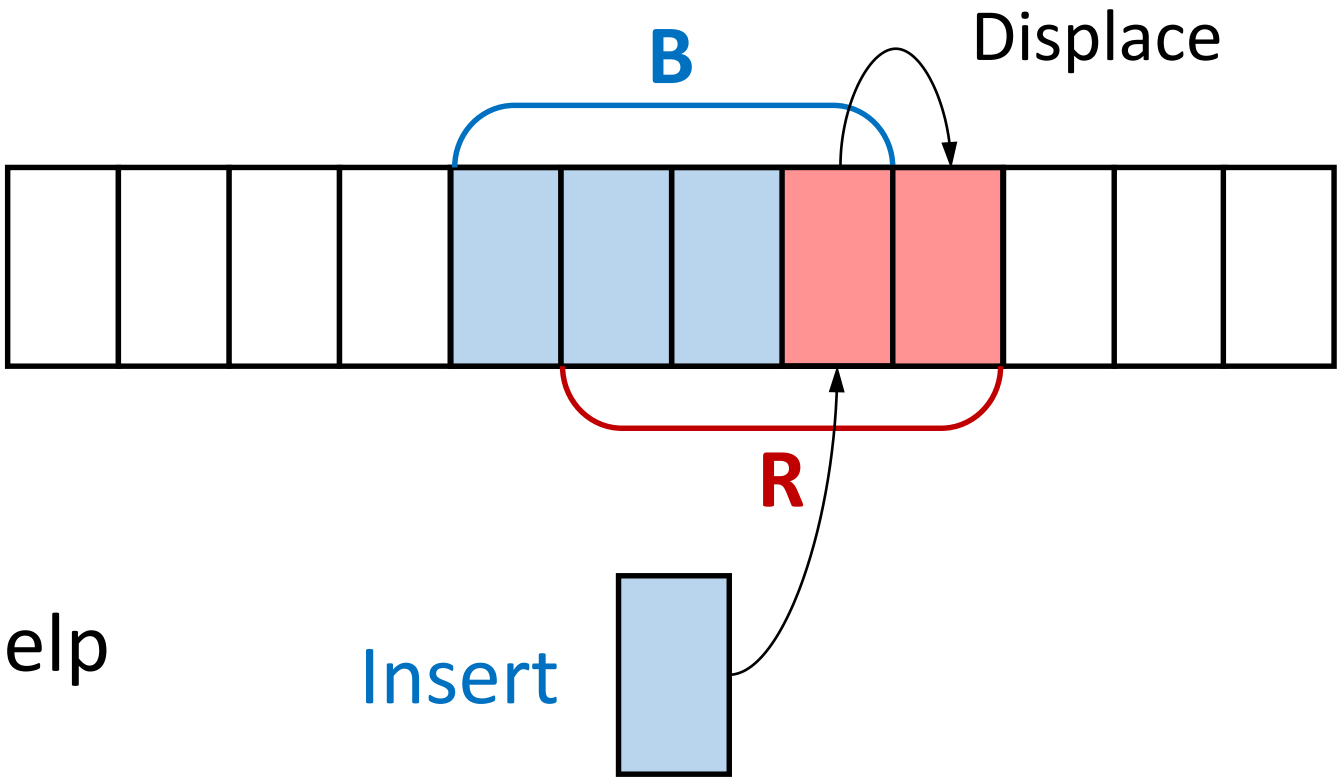
Lookup(5)

Hashtable lookup with a single RDMA

Maintaining invariant

Displace items to make room in neighbourhood

Resize when displacing does not help

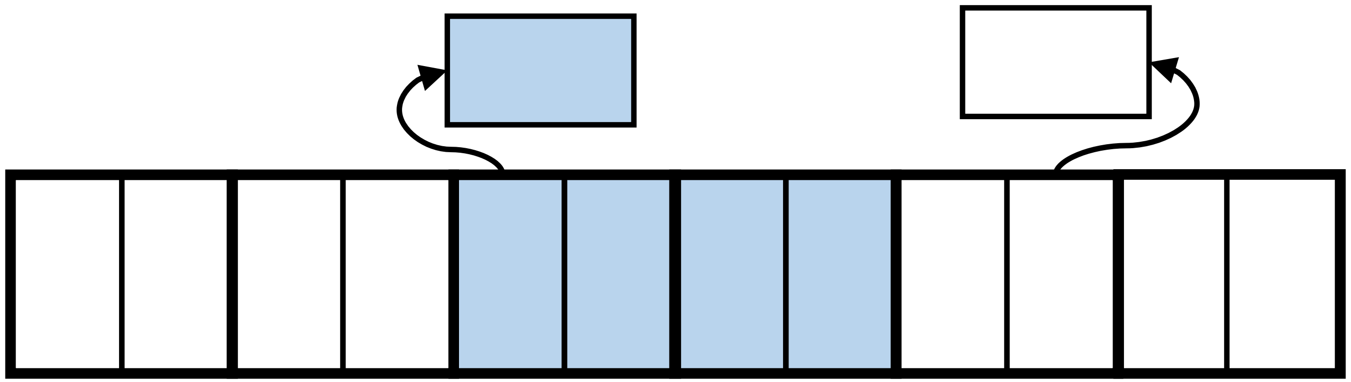


Use large neighbourhoods: 32 elements

FaRM hashtable

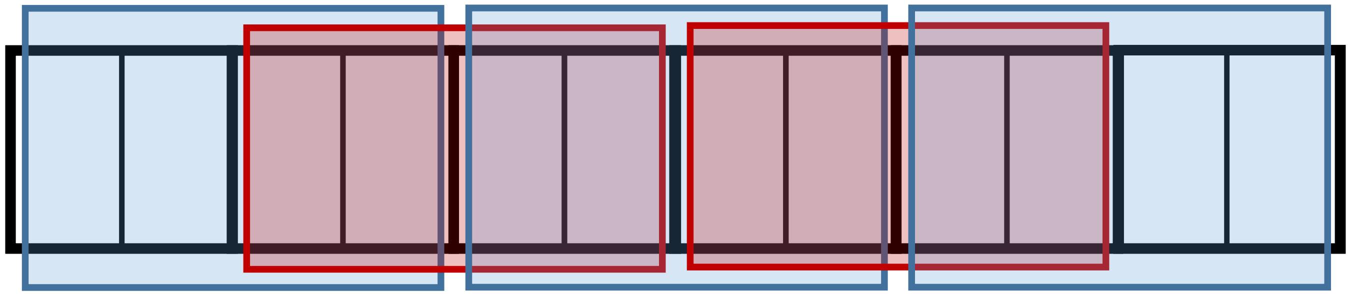
Overflow chaining

Element in neighbourhood



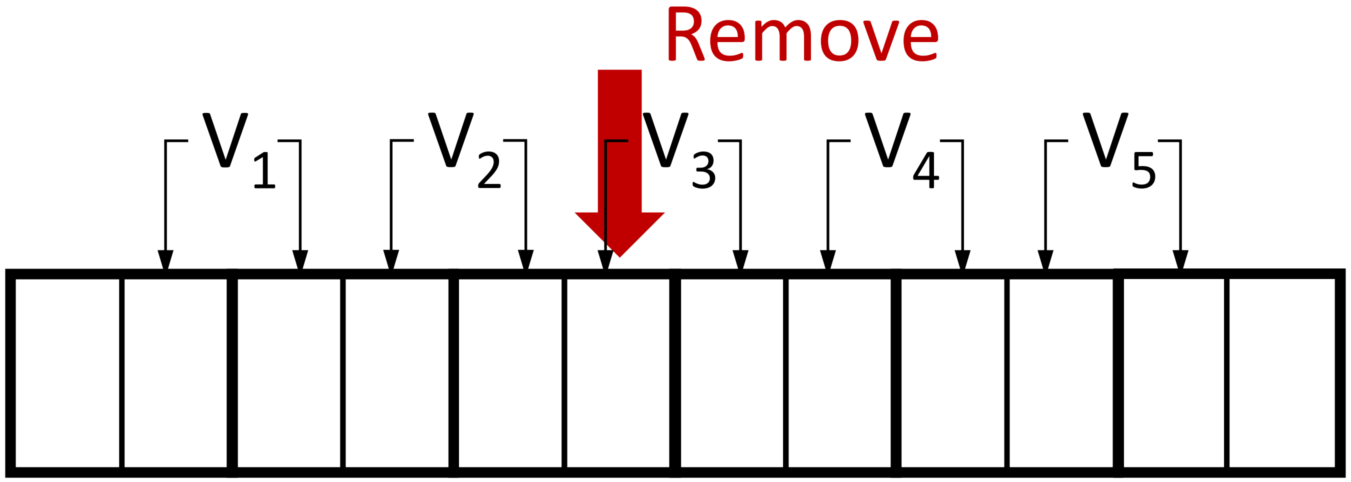
Space efficiency: multiple items per FaRM object

Overlapping neighbourhoods



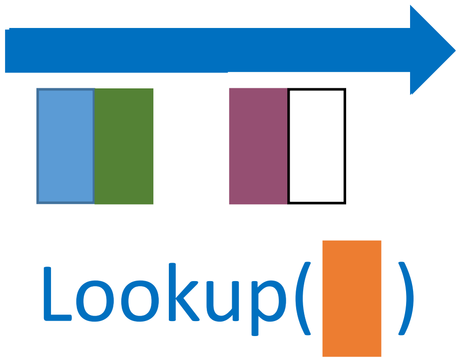
Consistent neighbourhoods

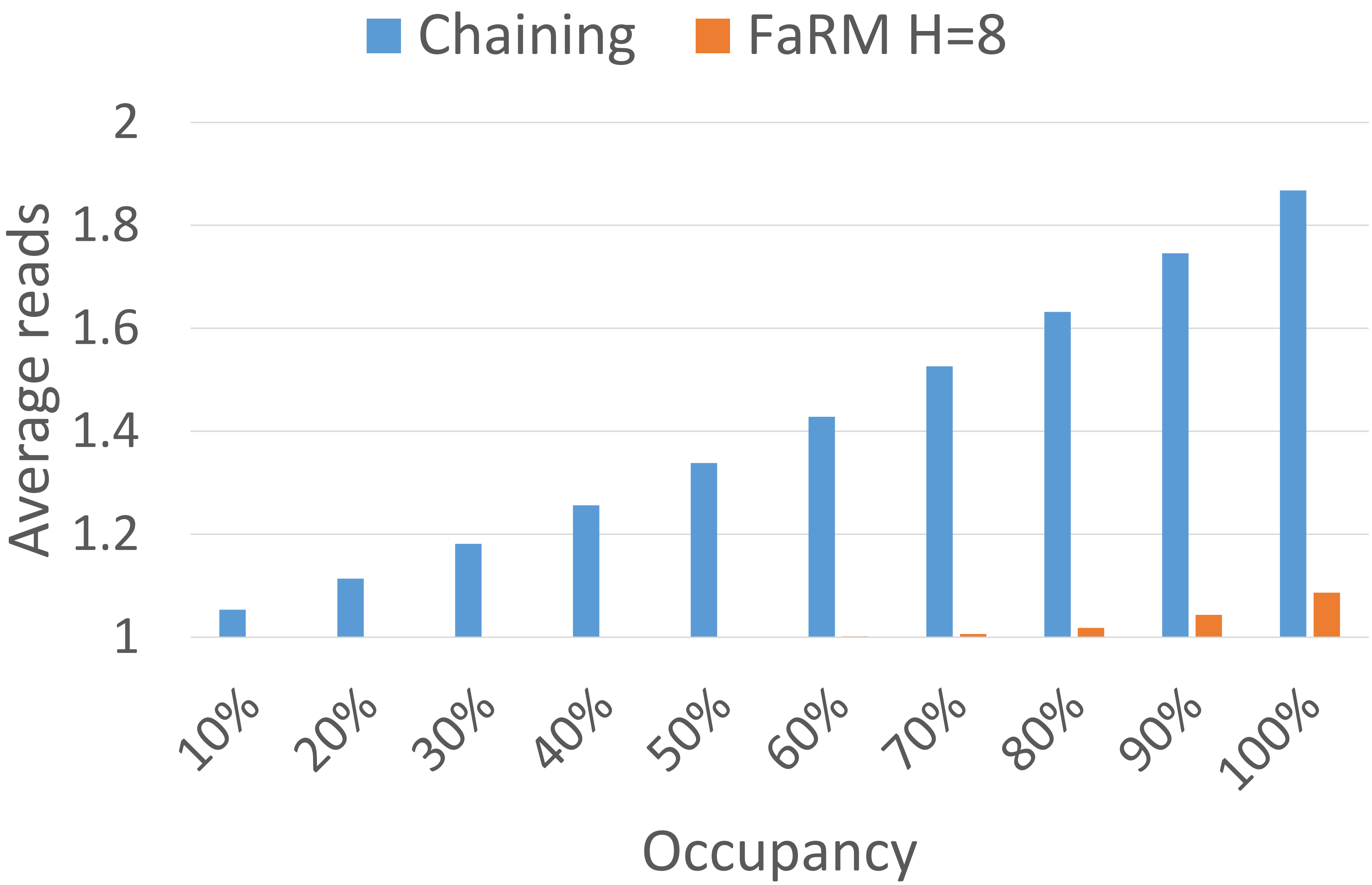
Neighbourhood versions



$\texttt{if obj}_1\texttt{.nv}_{next}\texttt{ != obj}_2\texttt{.nv}_{prev}\texttt{:}$

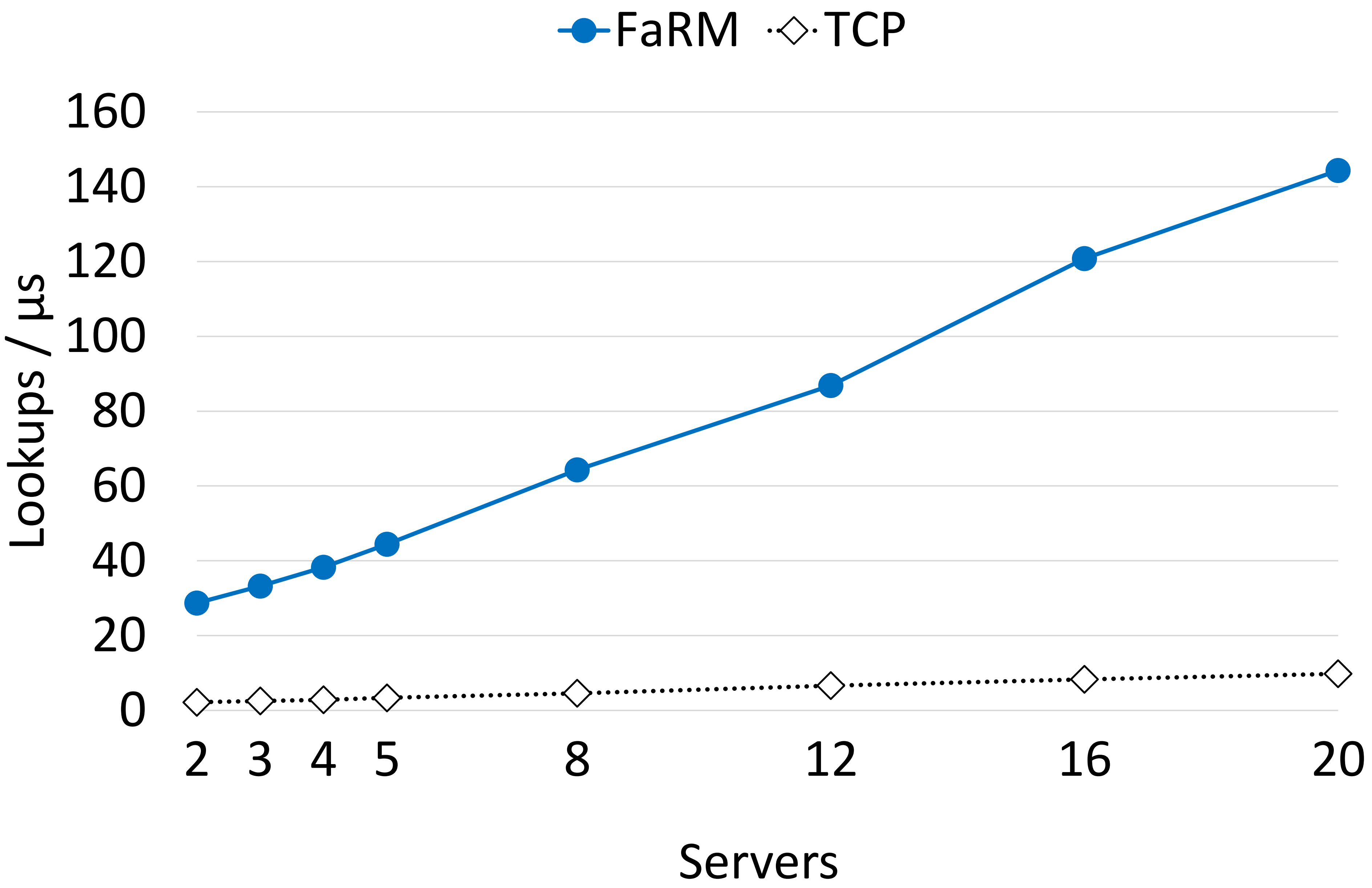
    retry



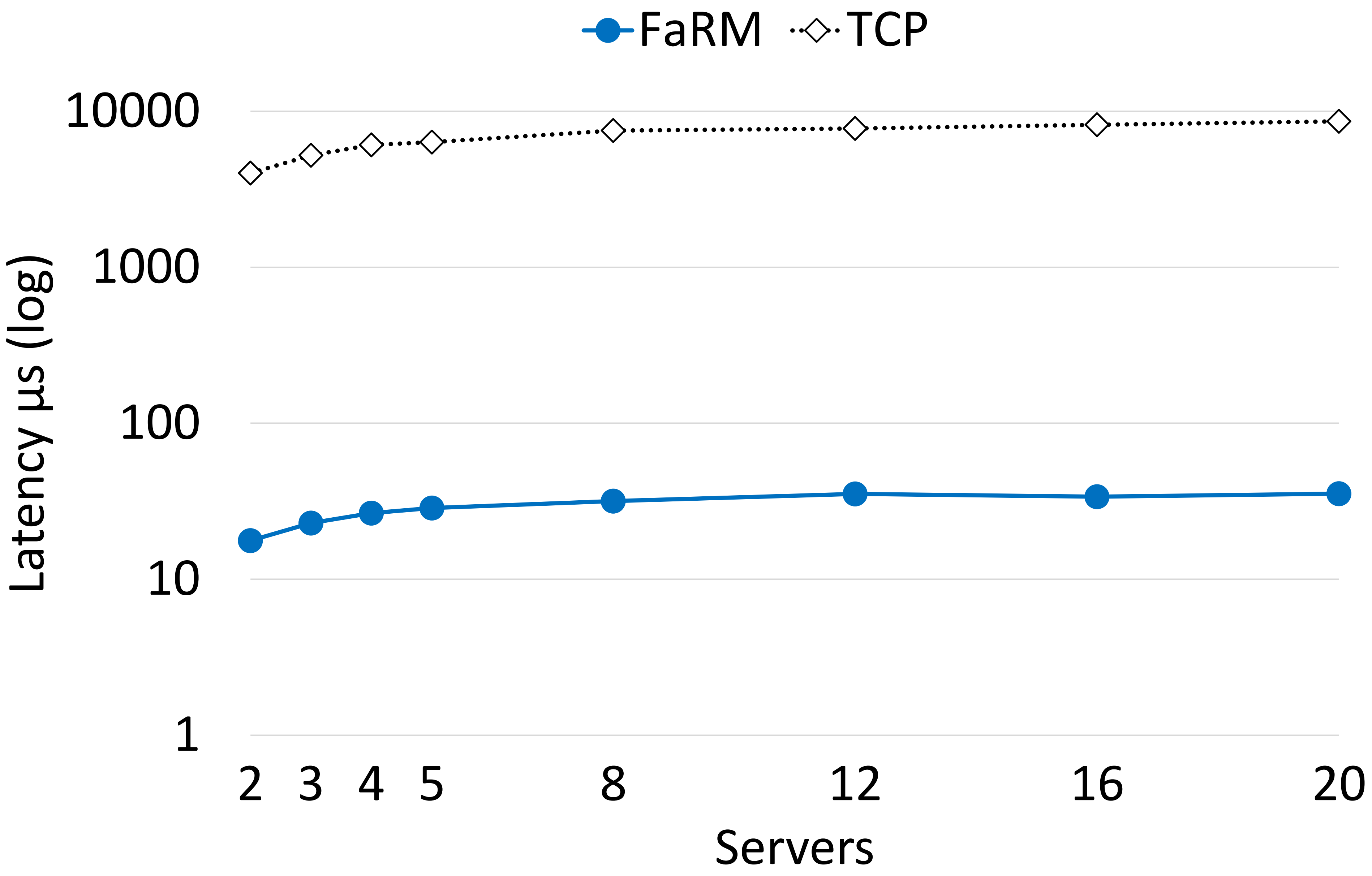

Chaining
FaRM H=8
2
1.8
1.6
1.4
1.2
1
Average reads
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
Occupancy

Outline

- FaRM programming model
- Design
  - Synchronization
  - Hashtable
- Experimental results
- Future work



FaRM
TCP
160
140
120
100
80
60
40
20
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]

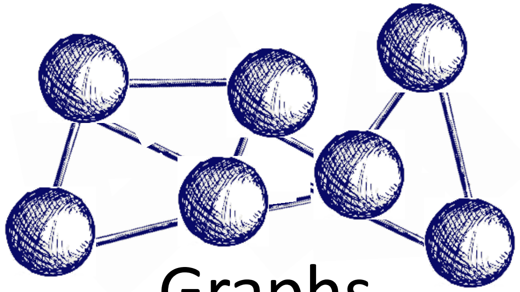
- Facebook’s in-memory graph store
- Workload
  - Read-dominated (99.8%)
  - 10 operation types
- FaRM implementation
  - Nodes and edges as FaRM objects
  - FaRM pointers between them
  - Lock-free reads for lookups
  - Transactions for updates

6 Mops/s/srv
(10x improvement)

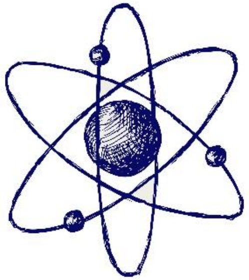
42 µs average latency
(40 – 50x improvement)

A step towards future data centres

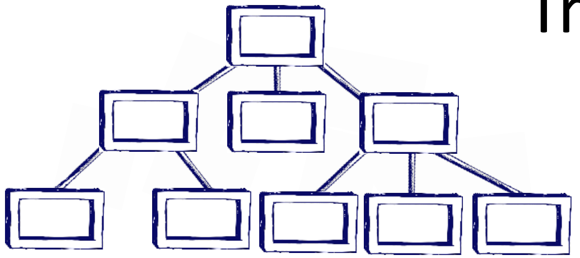
- Enabling new applications
  - Graph processing
  - Scale-out OLTP
  - Deep neural networks
- Future hardware
  - Software hardware co-design
  - Integrated network
  - Non-volatile memory



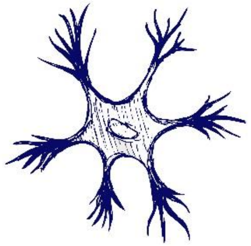
Graphs



Transactions



Data structures



Deep neural networks

Fault tolerance

FaRM [NSDI ‘14]

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