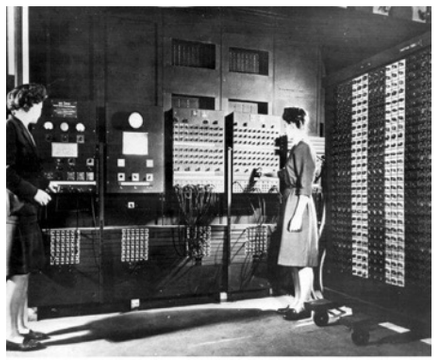
memory for large-scale systems

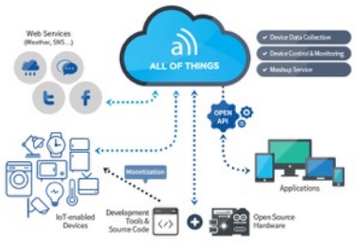
Prakhar Jawre

Outline

- Overview
- Current Technologies
- Challenges
- Conclusion

Timeline of IT and Computing





First generation computers

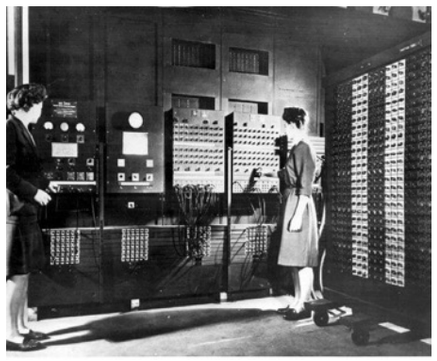
PC

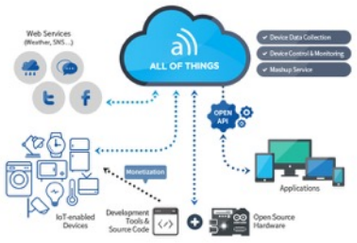
Mobile computing

Data Centres

onset of IOT & Big Data

Timeline of IT and Computing





First generation computers

PC

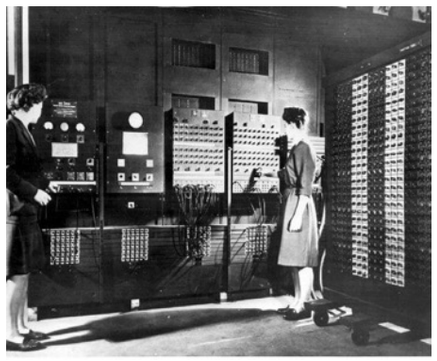
Mobile computing

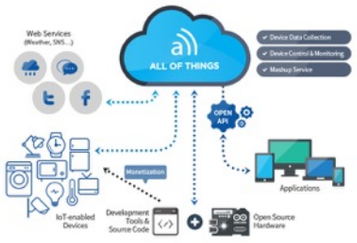
Data Centres

onset of IOT & Big Data

Rapid growth in digital data!

Timeline of IT and Computing





First generation computers

PC

Mobile computing

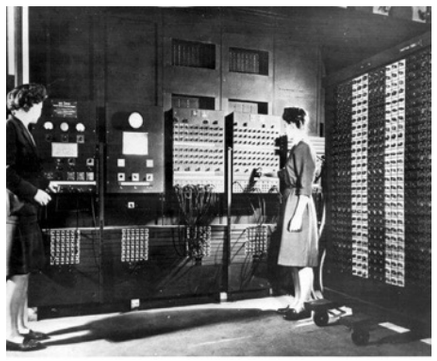
Data Centres

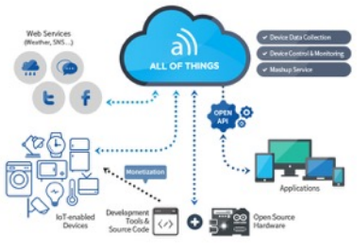
onset of IOT & Big Data

Rapid growth in digital data!

From Computing Centric to Data Centric

Timeline of IT and Computing





First generation computers | PC | Mobile computing | Data Centres | onset of IOT & Big Data

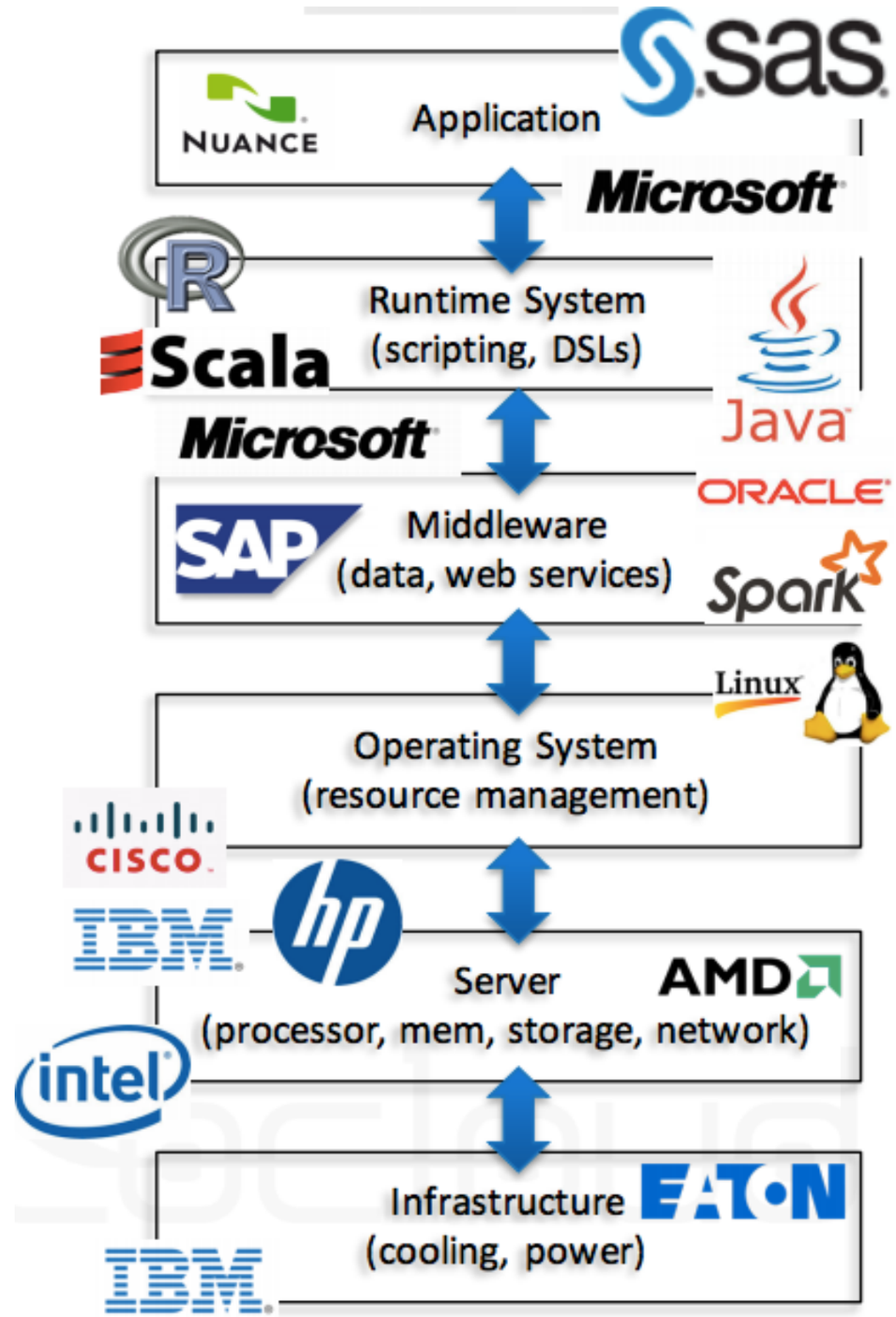
Rapid growth in digital data!

From Computing Centric to Data Centric

Data growth in 10 year(by 2015) = 100X

Today's Data Centers

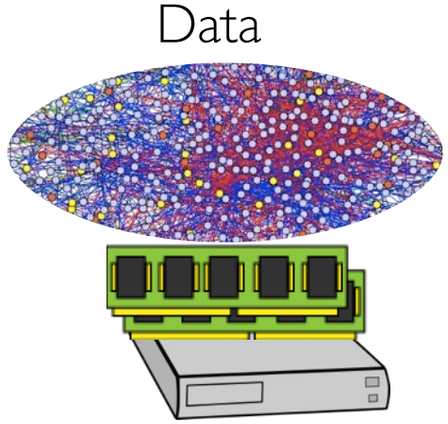
- Per-vendor Layer
- Product Based
- Near neighbor optimization
- Big vendors(Google, Amazon etc)
- Can do Cross Layer Optimization but,
  - Limited to service of Interest
  - Limited to Extend(Software)
  - Closed Technologies



Outline

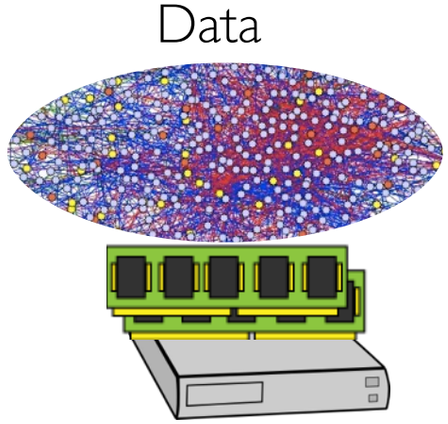
- Overview
- Current Technologies
- Challenges
- Conclusion

In-Memory Computing for high Performance



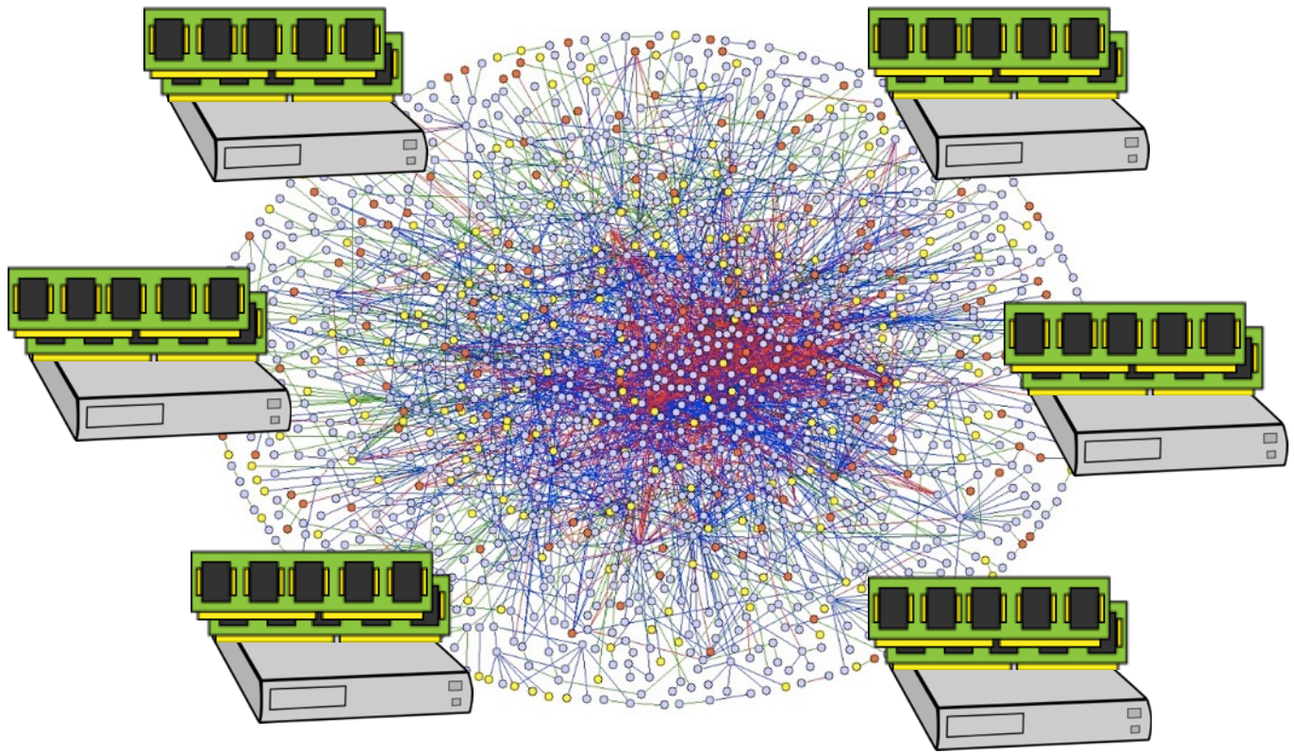
In-Memory Computing for high Performance

Why: Tight Latency Constraints
How: Place data in main memory



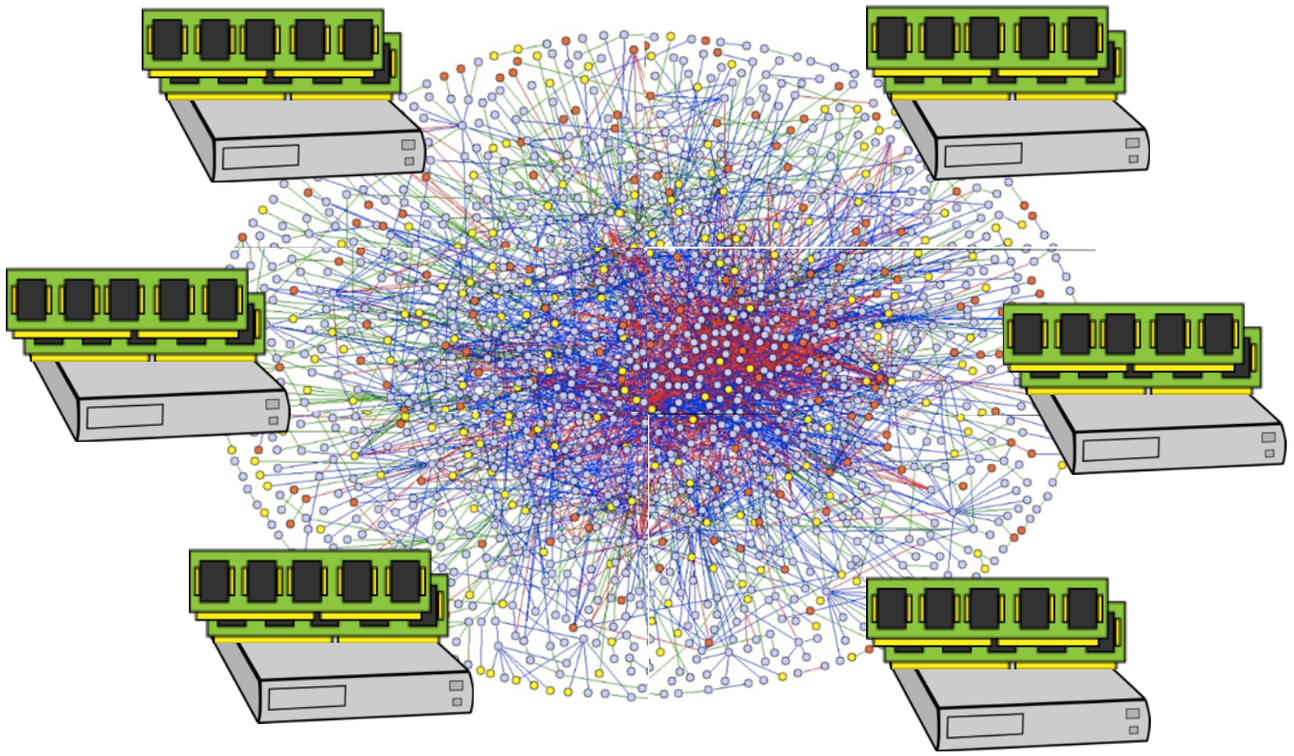
In-Memory Computing for high Performance

Why: Tight Latency Constraints
How: Place data in main memory

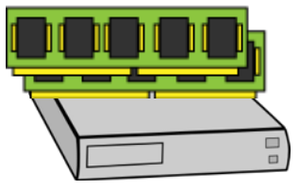
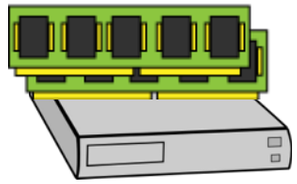
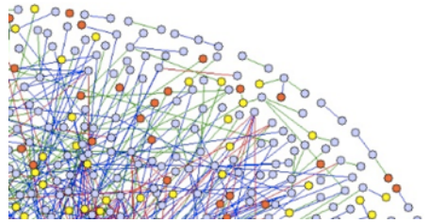
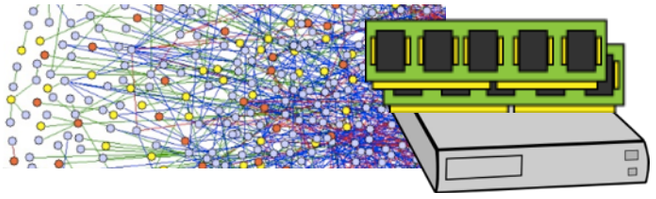
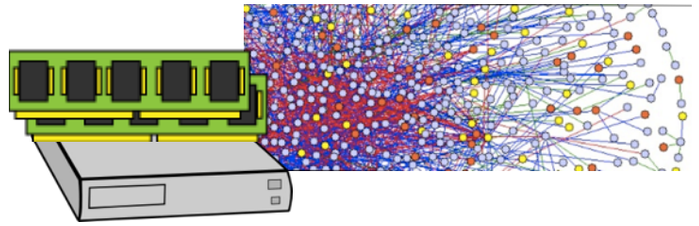
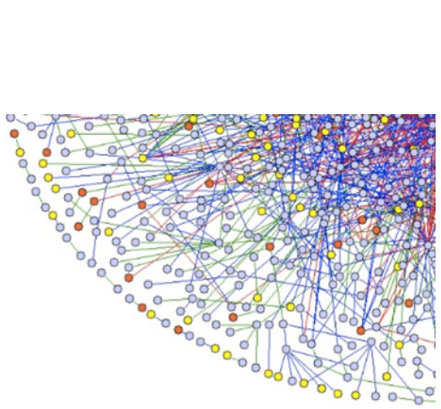
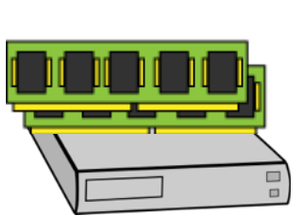
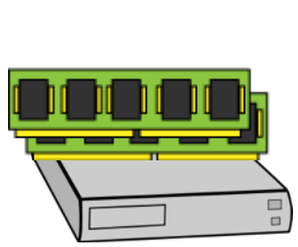
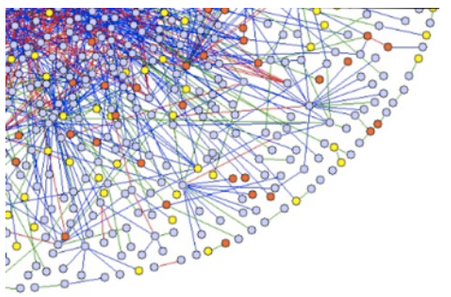


Need large memory pool to accommodate all data

Nodes Frequently Access Remote Memory



Nodes Frequently Access Remote Memory

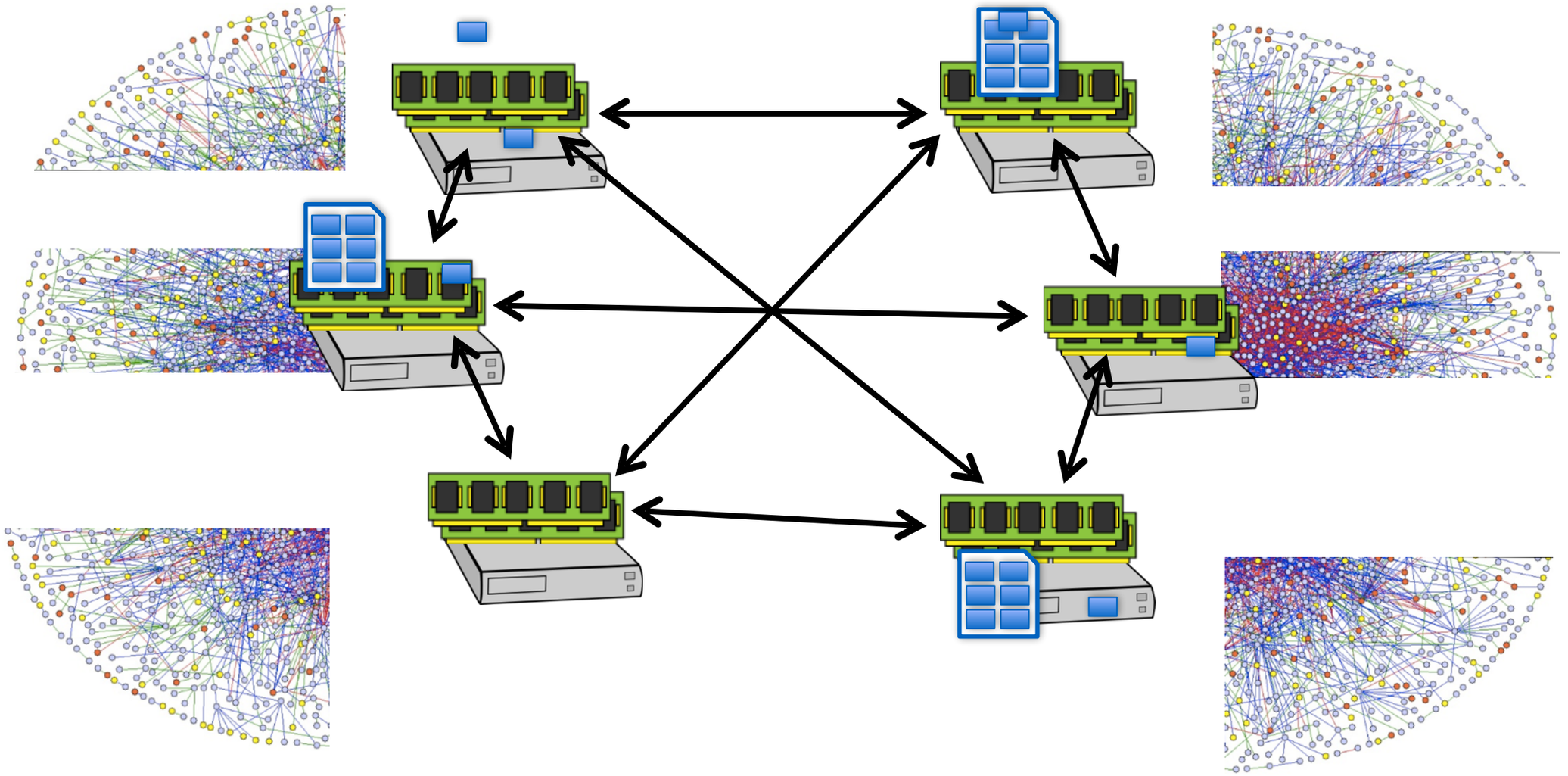
Graph Serving

- Fine-grained access

Graph Analytics

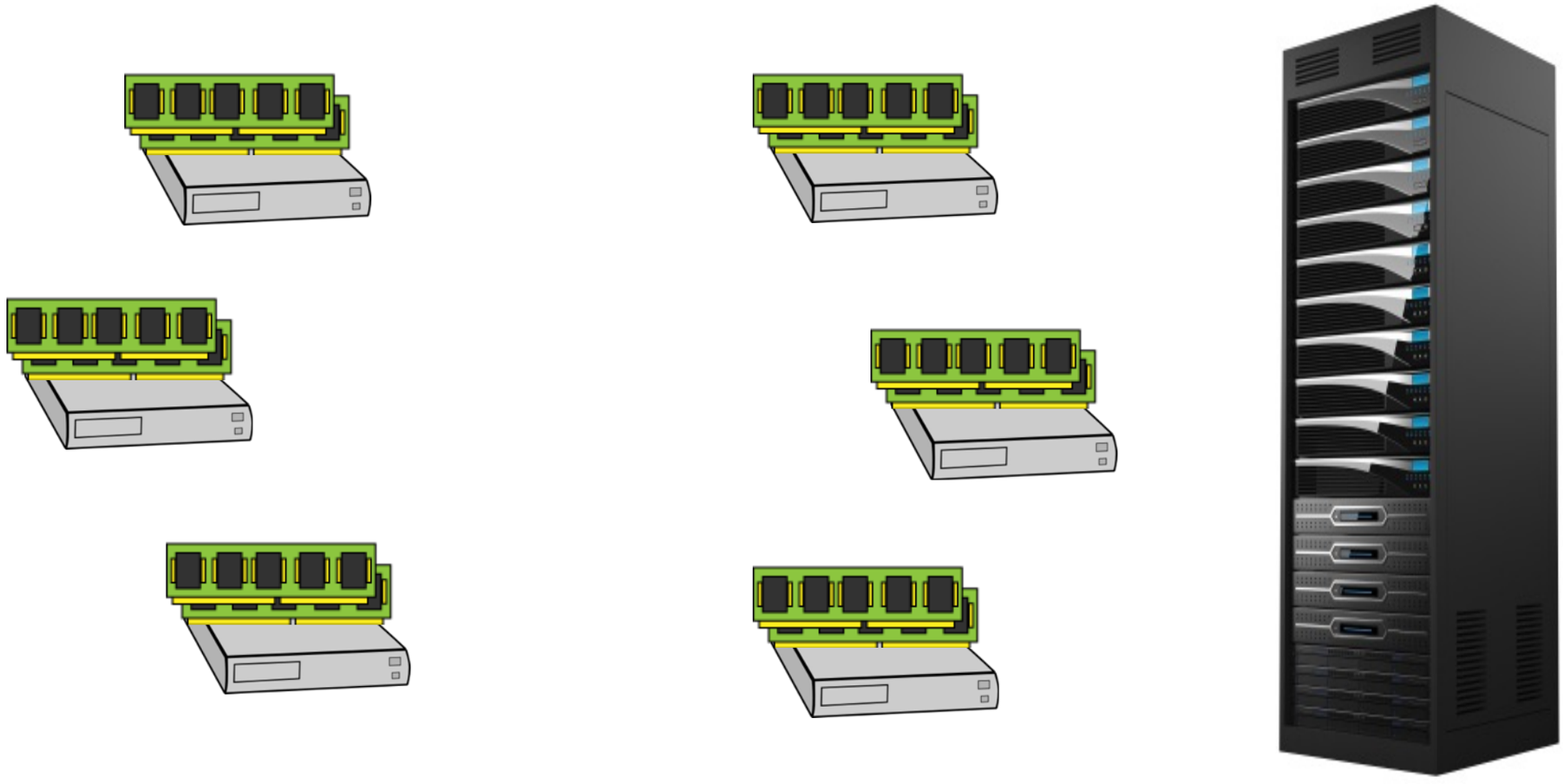
- Bulk acc

Nodes Frequently Access Remote Memory



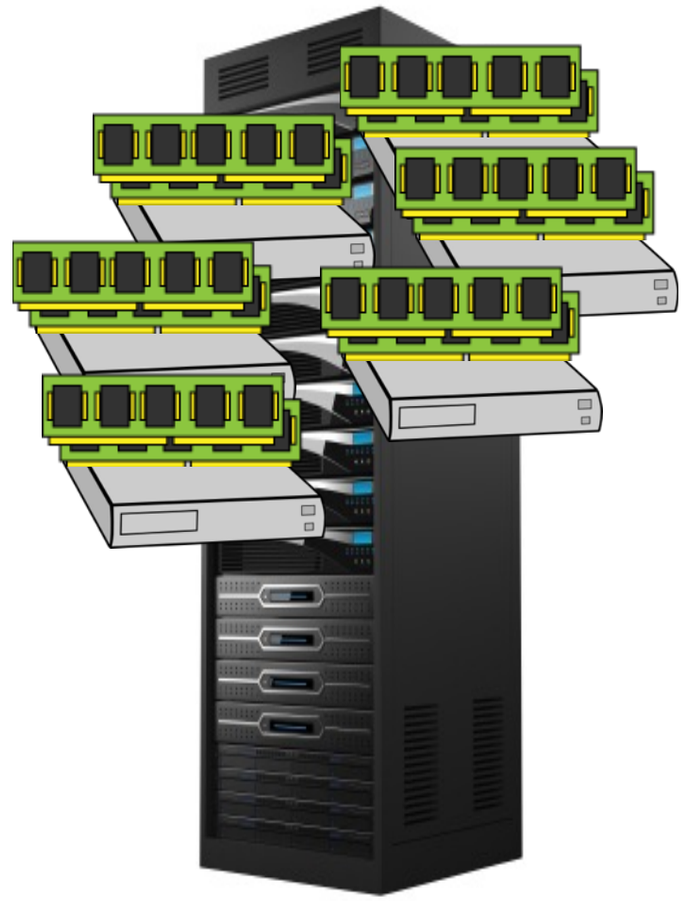
Need fast access to both small and large objects in remote memory

Rack-Scale Systems: Fast Access to Big Memory



Rack-Scale Systems: Fast Access to Big Memory

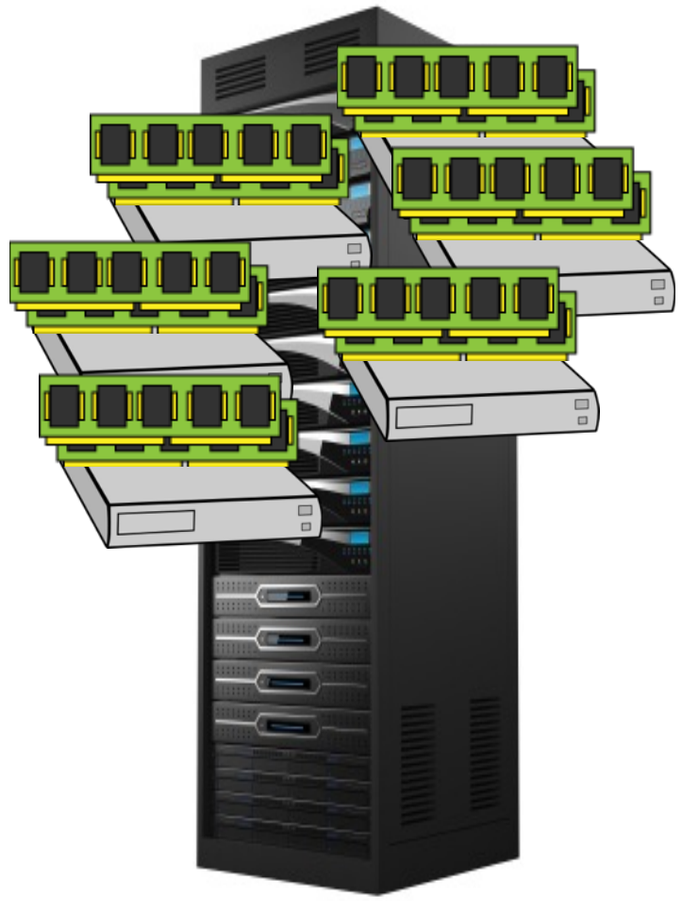
- Vast memory pool in small form factor
- On-chip integrated, cache-coherent NIs
  - Examples: Scale-Out NUMA [Novakovic et al., ’14]



Large memory capacity, low latency, high bandwidth

Rack-Scale Systems: Fast Access to Big Memory

- Vast memory pool in small form factor
- On-chip integrated, cache-coherent NIs
  - Examples: Scale-Out NUMA [Novakovic et al., '14]
- High-performance inter-node interconnect
  - NUMA environment
  - Low latency for fine-grained transfers
  - High bandwidth for bulk transfers



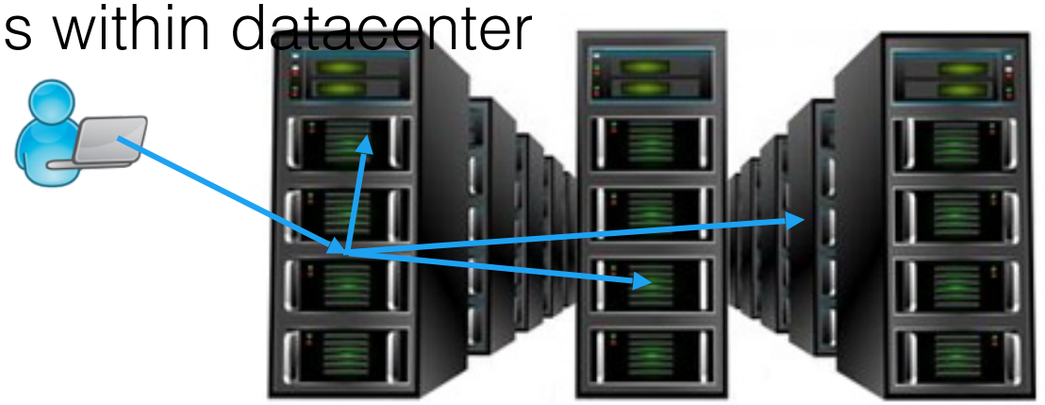
Large memory capacity, low latency, high bandwidth

Remote Direct Memory Access (RDMA)

- Frequent remote memory accesses within datacenter
- Networking traditionally expensive

Remote Direct Memory Access (RDMA)

- Frequent remote memory accesses within datacenter
- Networking traditionally expensive



Remote Direct Memory Access (RDMA)

- Frequent remote memory accesses within datacenter
- Networking traditionally expensive

<u>RDMA technology</u>

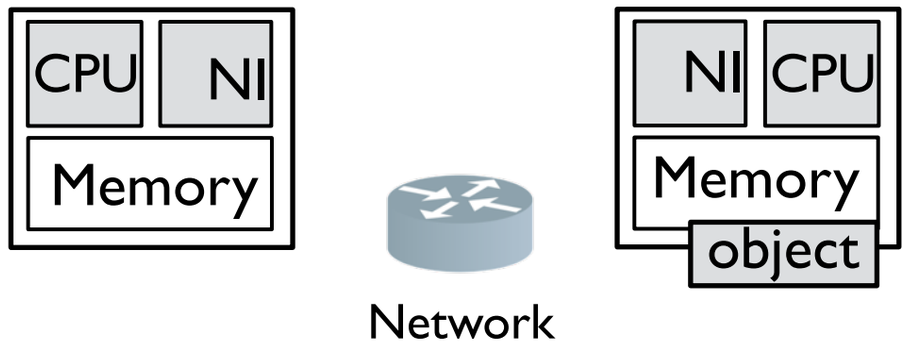
- Fast one-sided operations
- Destination CPU not involved

Outline

- Overview
- Current Technologies
- Challenges
  - Atomic Object Read
  - Network Interface for Many-Core Chip
- Proposed Solutions
- Conclusion

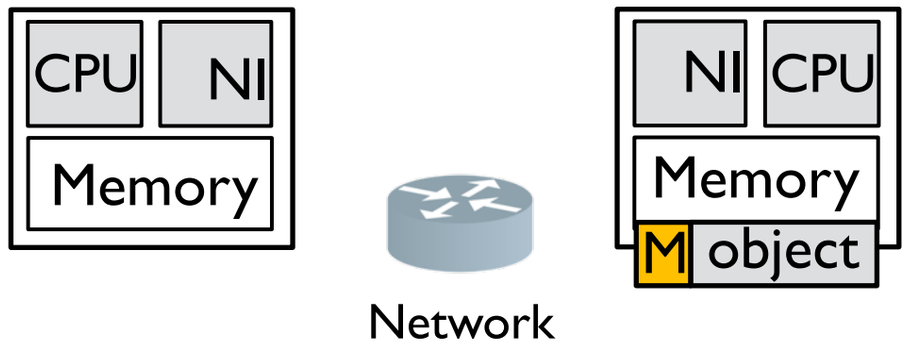
Object Atomicity in RDMA

- Atomicity limited to single cache line in one sided ops
- Software construct for object atomicity
  - Embedded MetaData



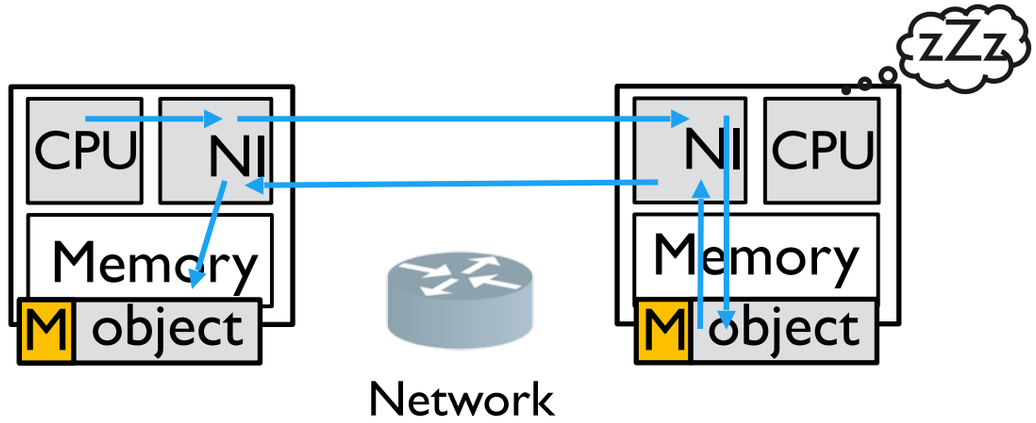
Object Atomicity in RDMA

- Atomicity limited to single cache line in one sided ops
- Software construct for object atomicity
  - Embedded MetaData



Object Atomicity in RDMA

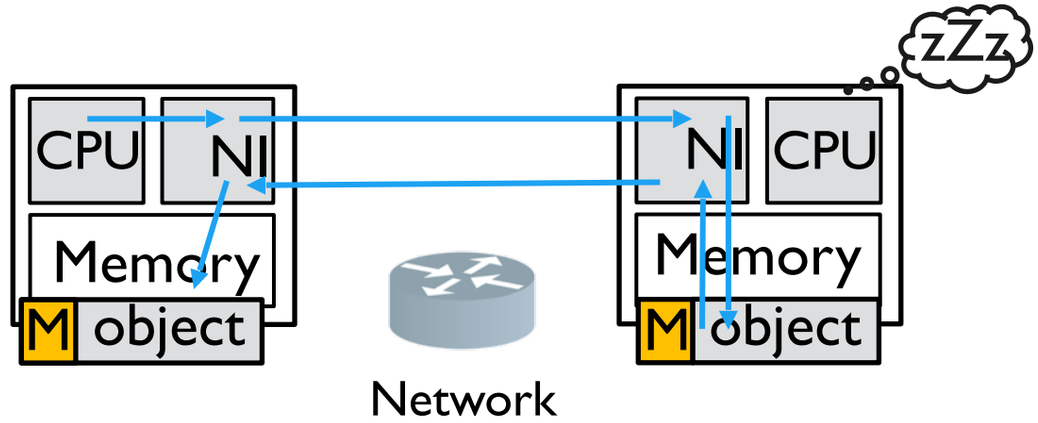
- Atomicity limited to single cache line in one sided ops
- Software construct for object atomicity
  - Embedded MetaData



Object Atomicity in RDMA

- Atomicity limited to single cache line in one sided ops
- Software construct for object atomicity
  - Embedded MetaData

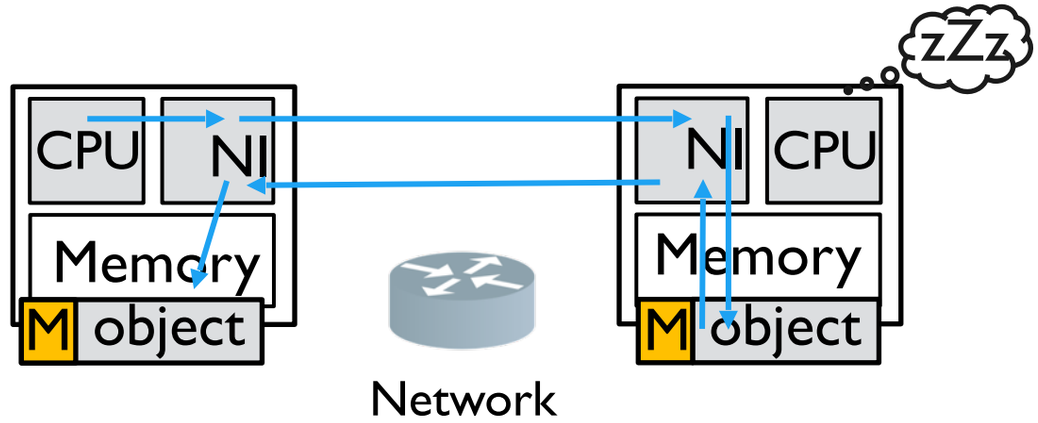
post-transfer atomicity check by CPU



Object Atomicity in RDMA

- Atomicity limited to single cache line in one sided ops
- Software construct for object atomicity
  - Embedded MetaData

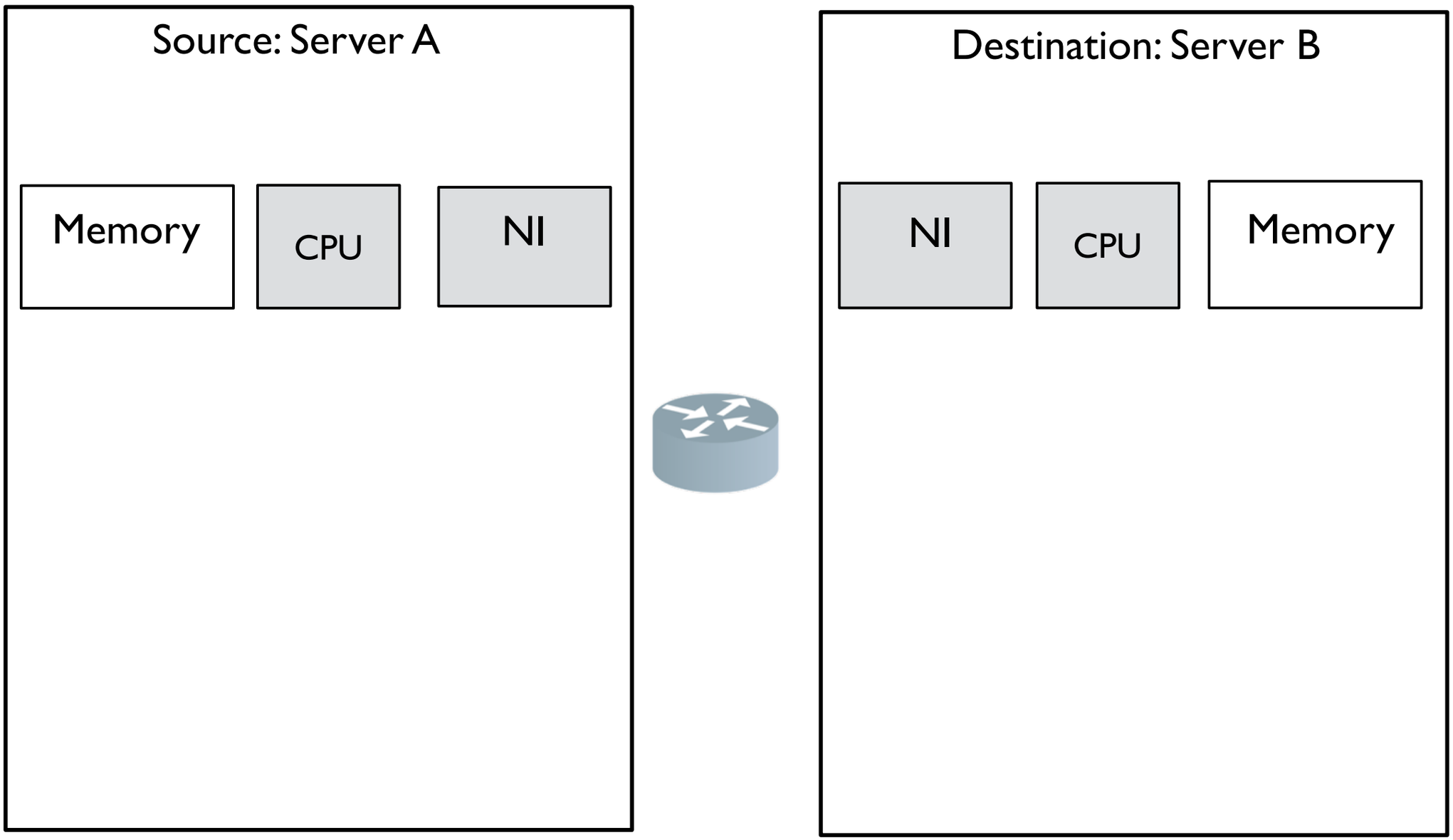
post-transfer atomicity check by CPU



- SW check expensive=>upto 50% of remote object read

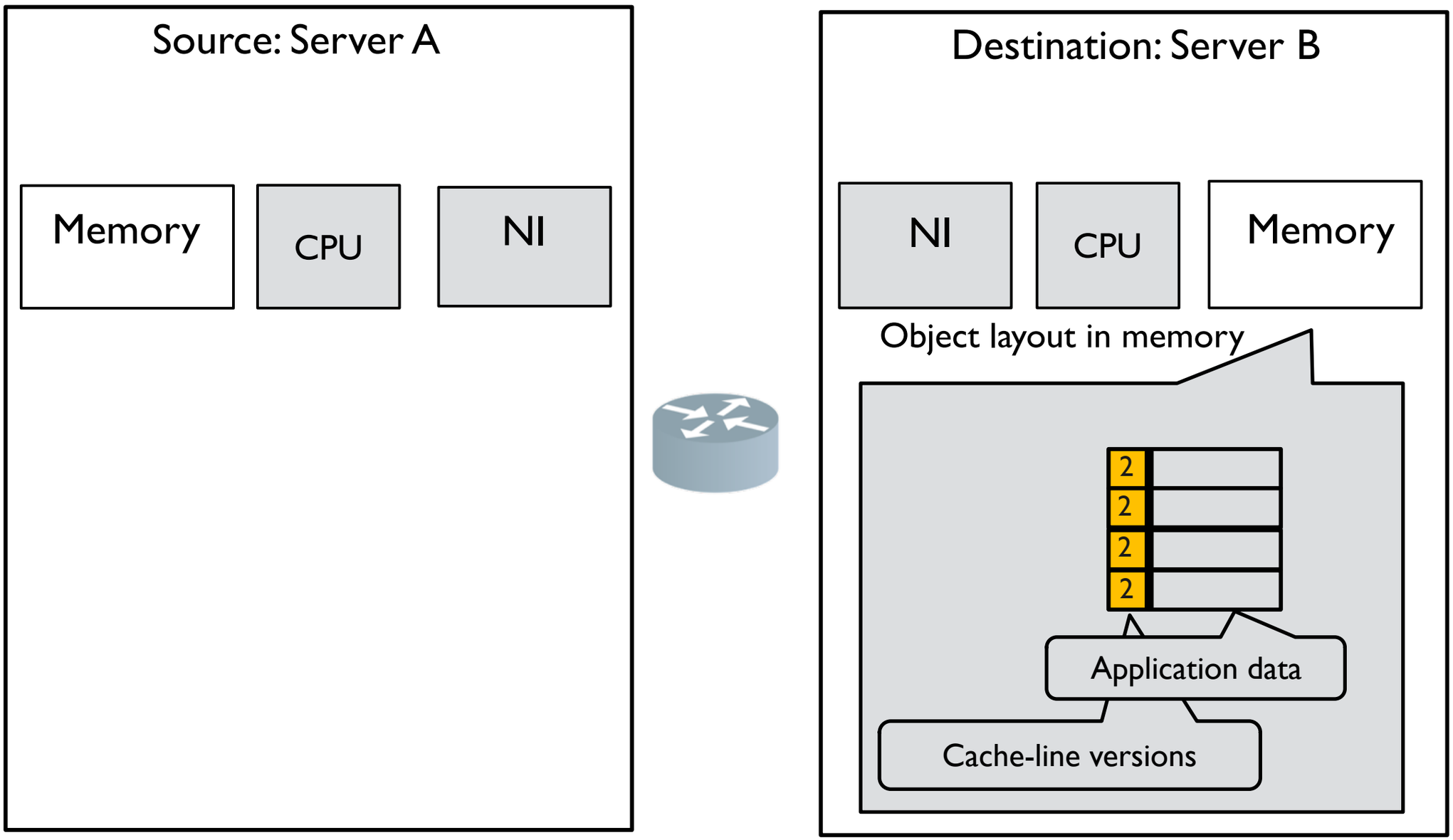
SW atomicity: How is it done?

Per-cache-line versions (FaRM [Dragojevic'14,'15])



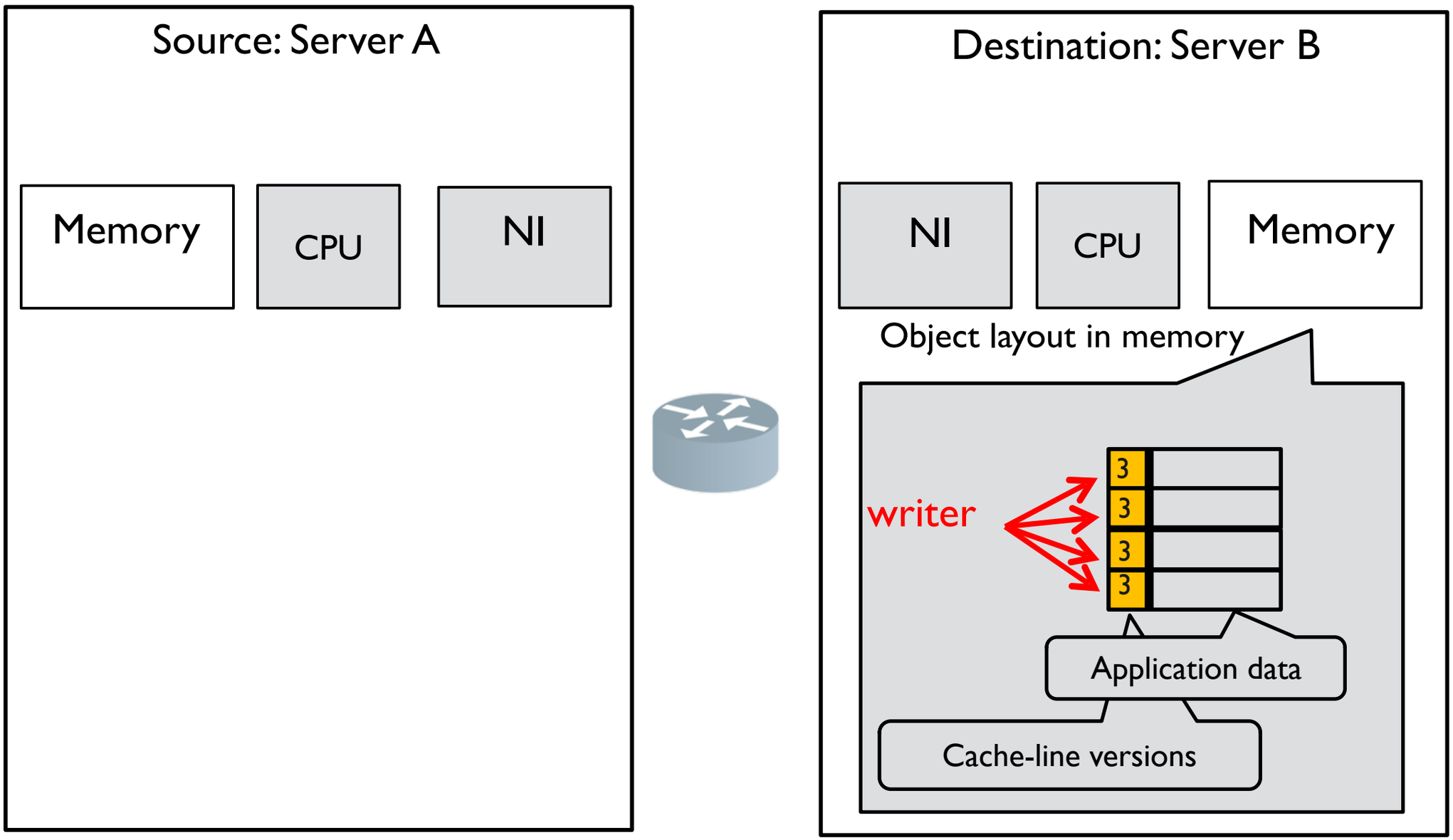
SW atomicity: How is it done?

Per-cache-line versions (FaRM [Dragojevic'14,'15])



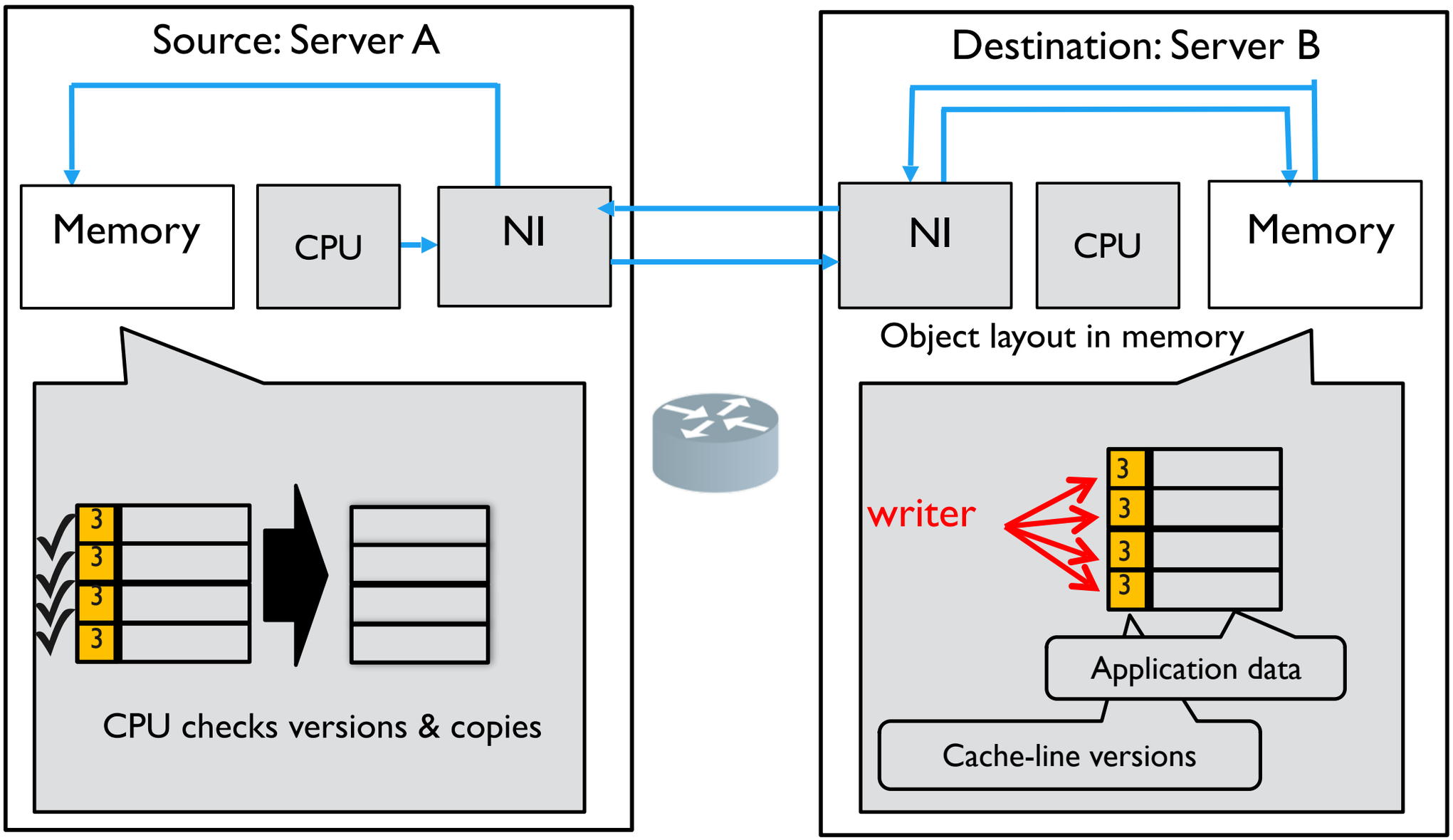
SW atomicity: How is it done?

Per-cache-line versions (FaRM [Dragojevic'14,'15])

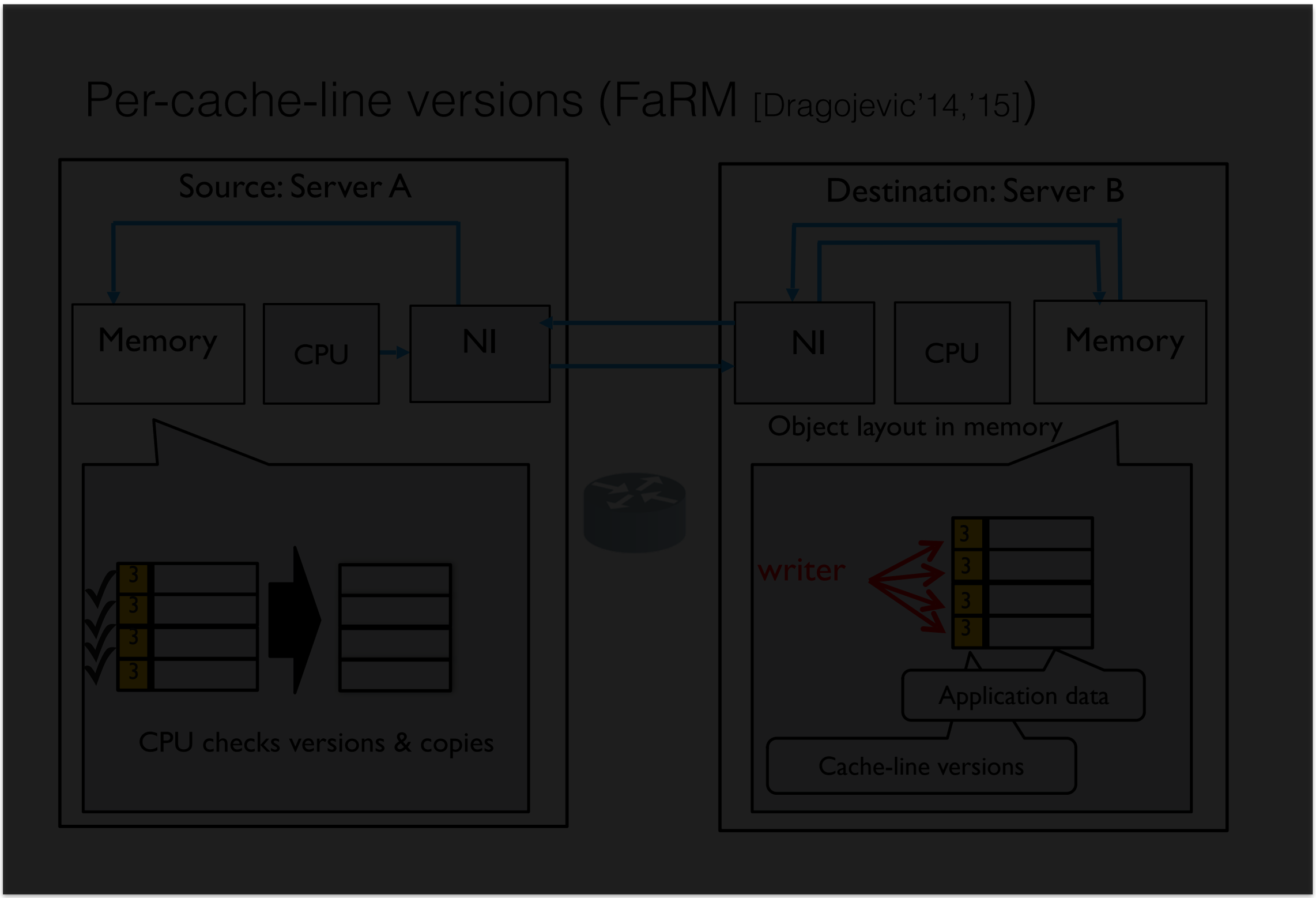


SW atomicity: How is it done?

Per-cache-line versions (FaRM [Dragojevic'14,'15])

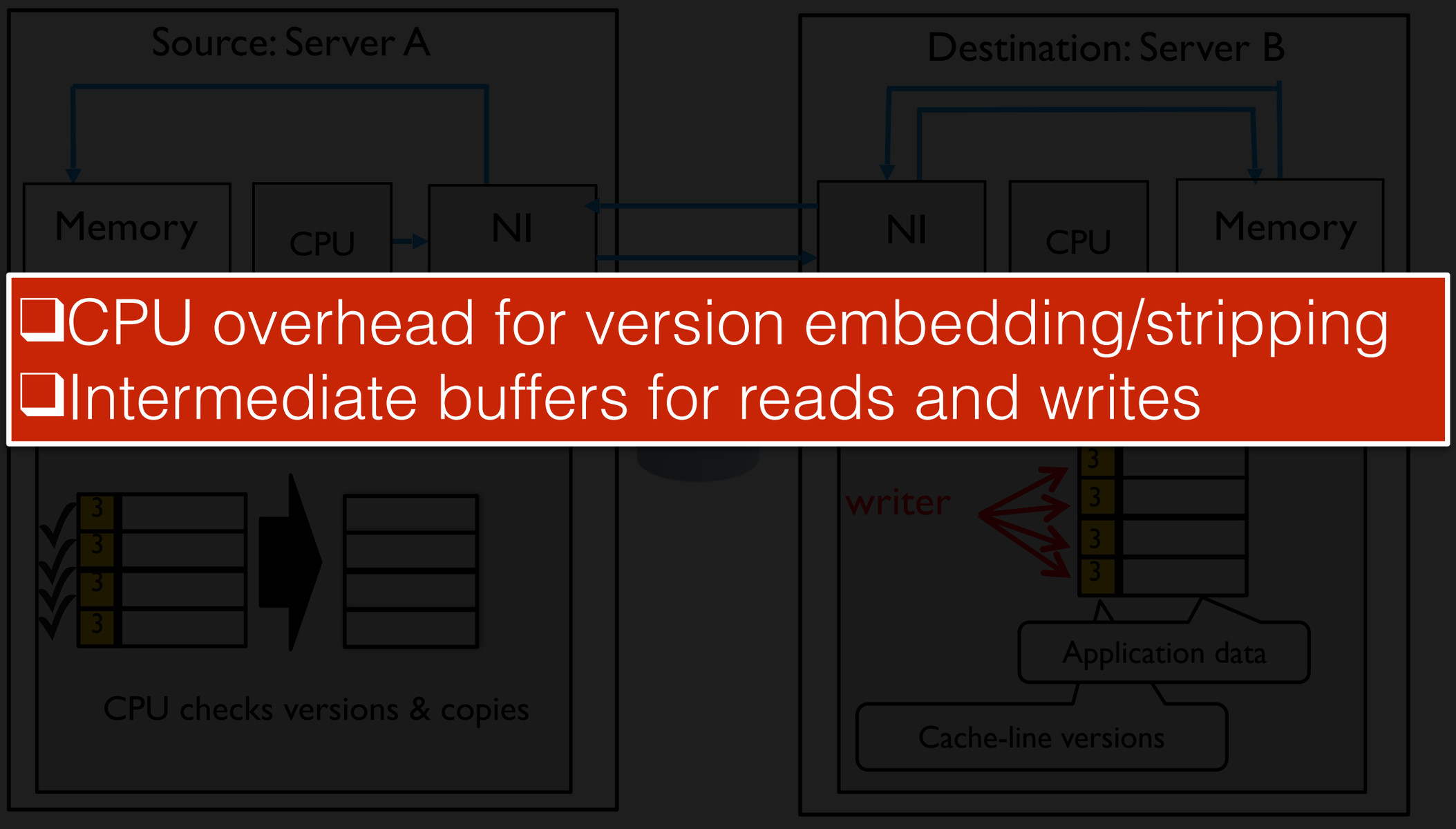


SW atomicity: How is it done?



SW atomicity: How is it done?

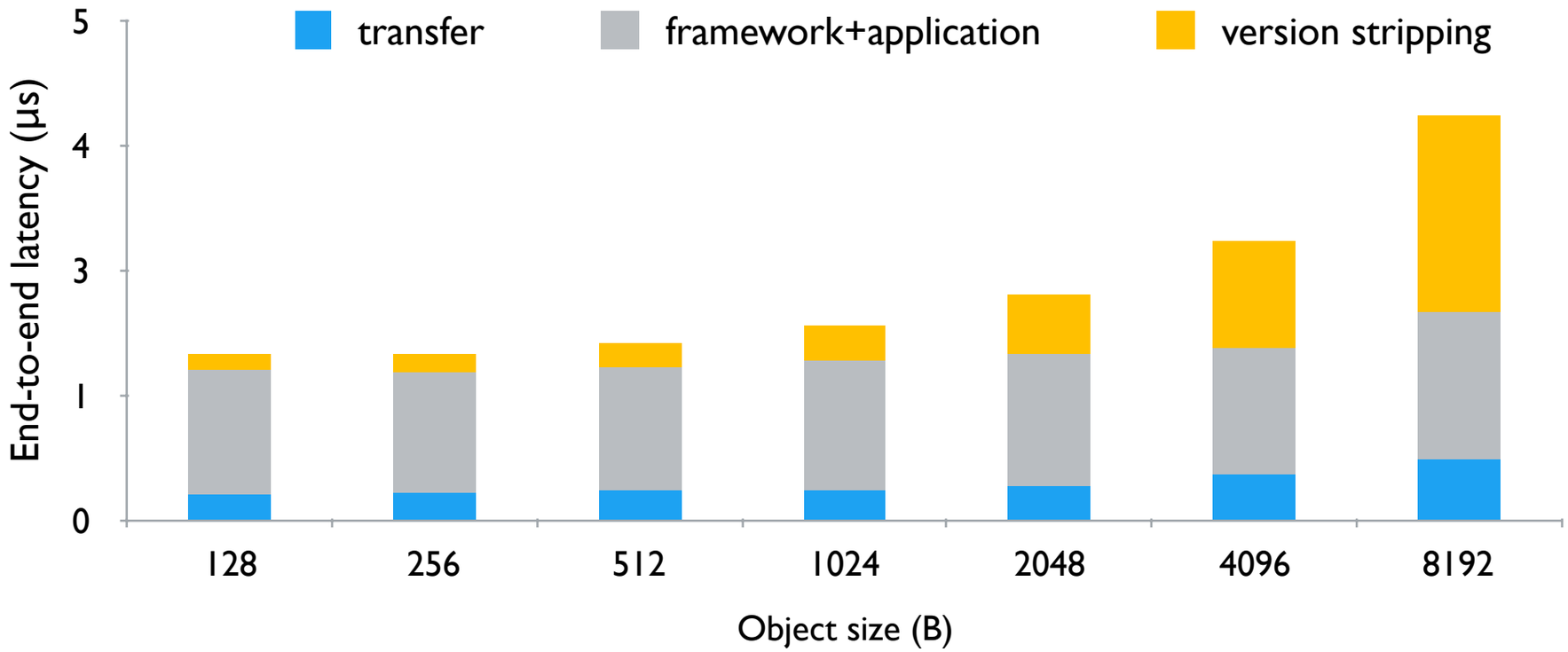
Per-cache-line versions (FaRM [Dragojevic'14,'15])



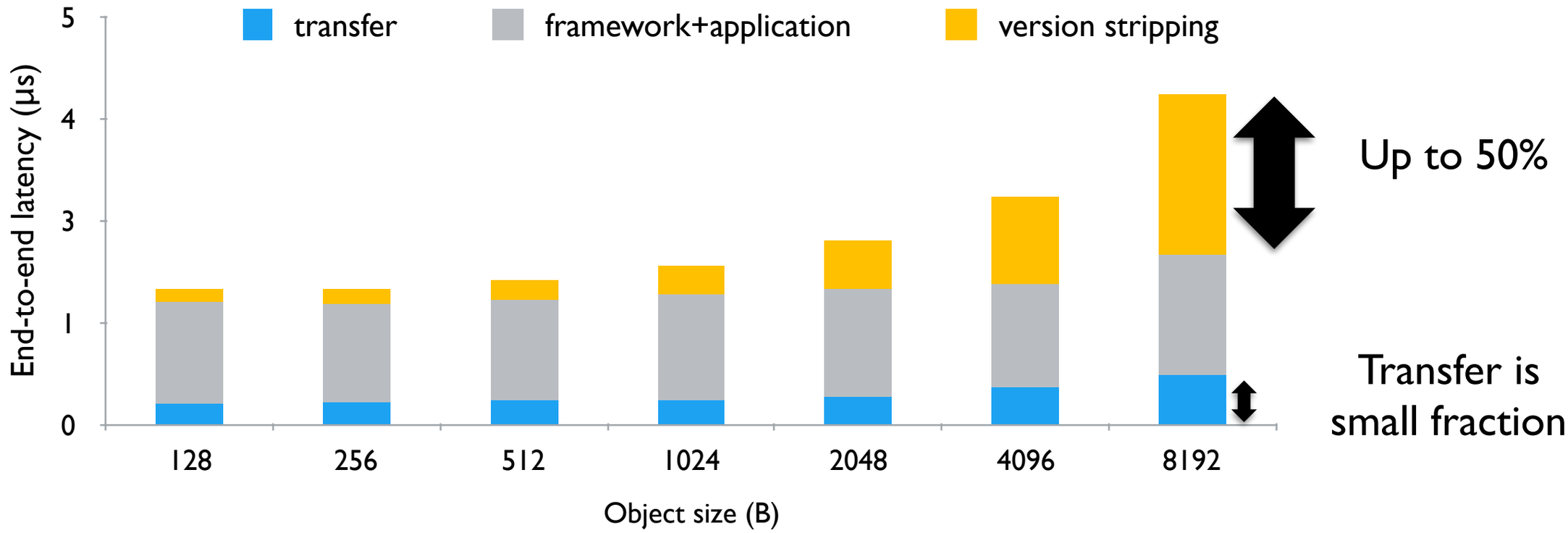
- CPU overhead for version embedding/stripping
- Intermediate buffers for reads and writes

SW atomicity overhead

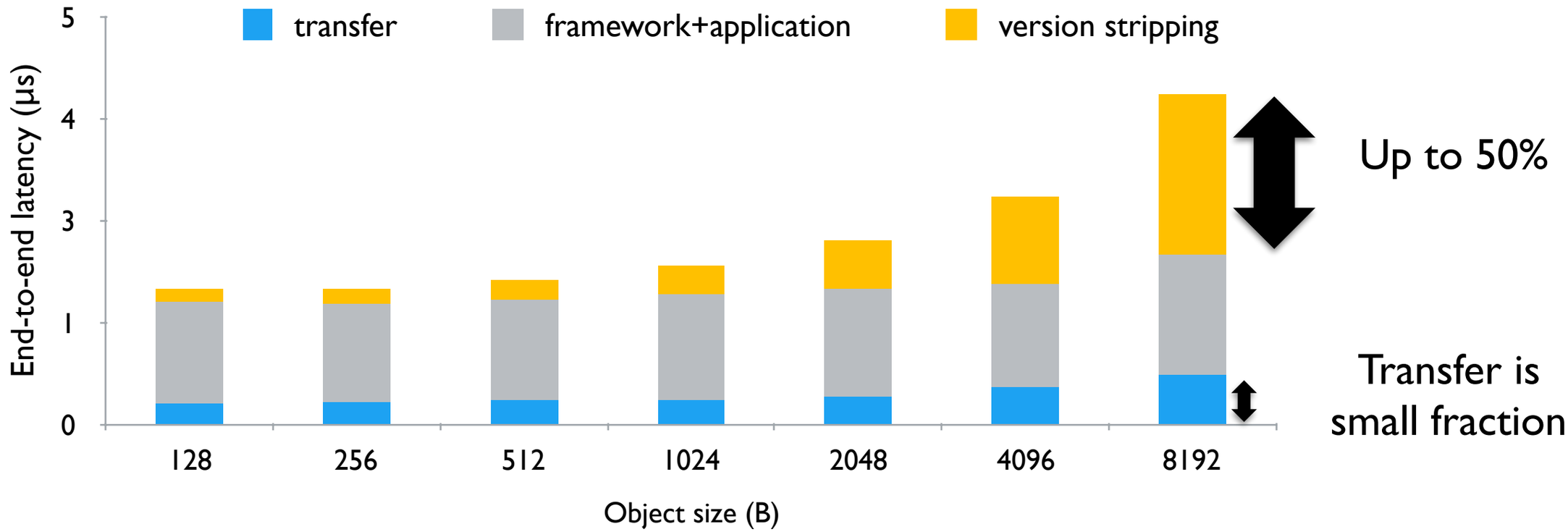
SW atomicity overhead



SW atomicity overhead

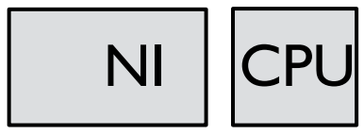


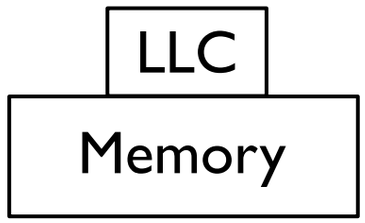
SW atomicity overhead



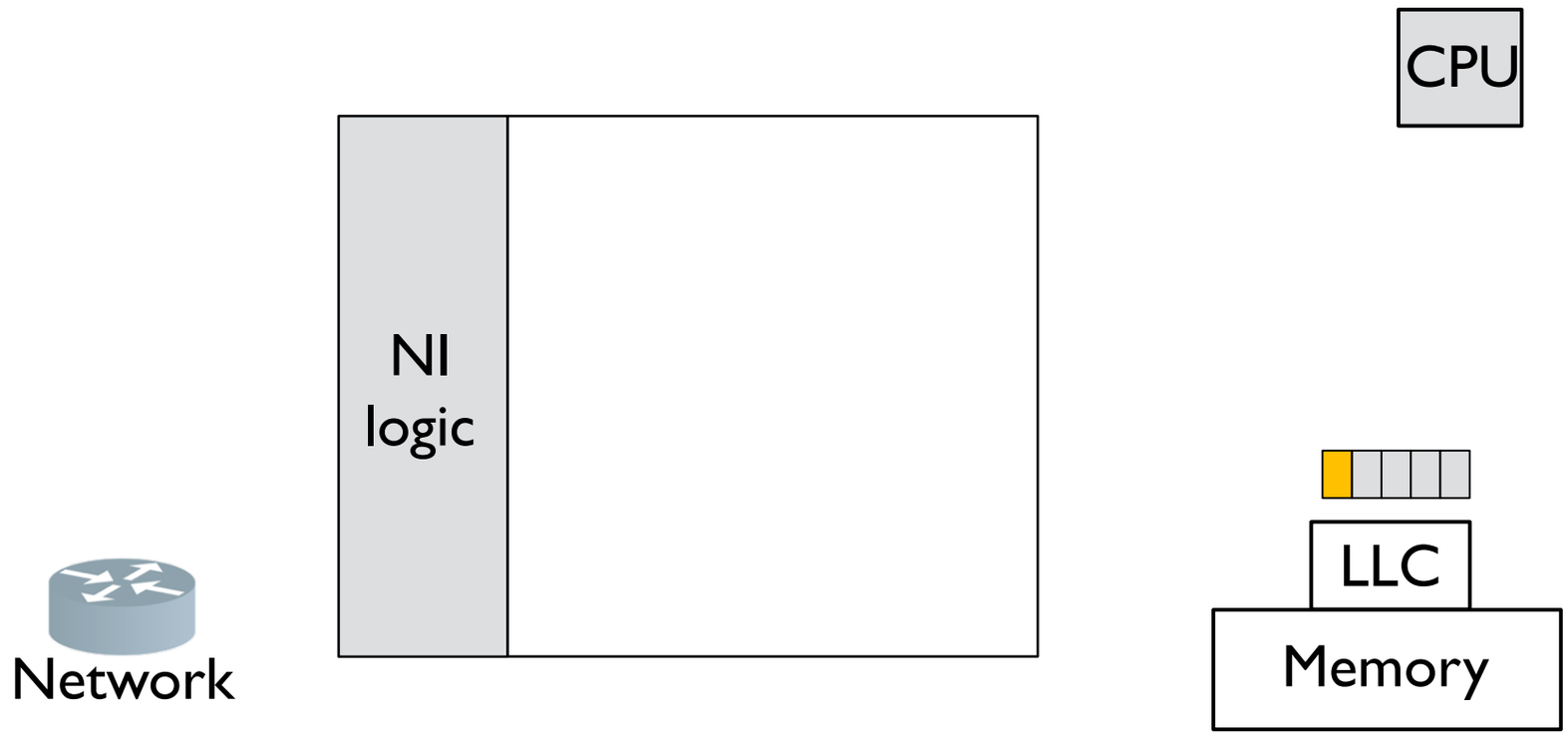
Can we do this using hardware?

SABRes: Single Site Atomic Bulk Read

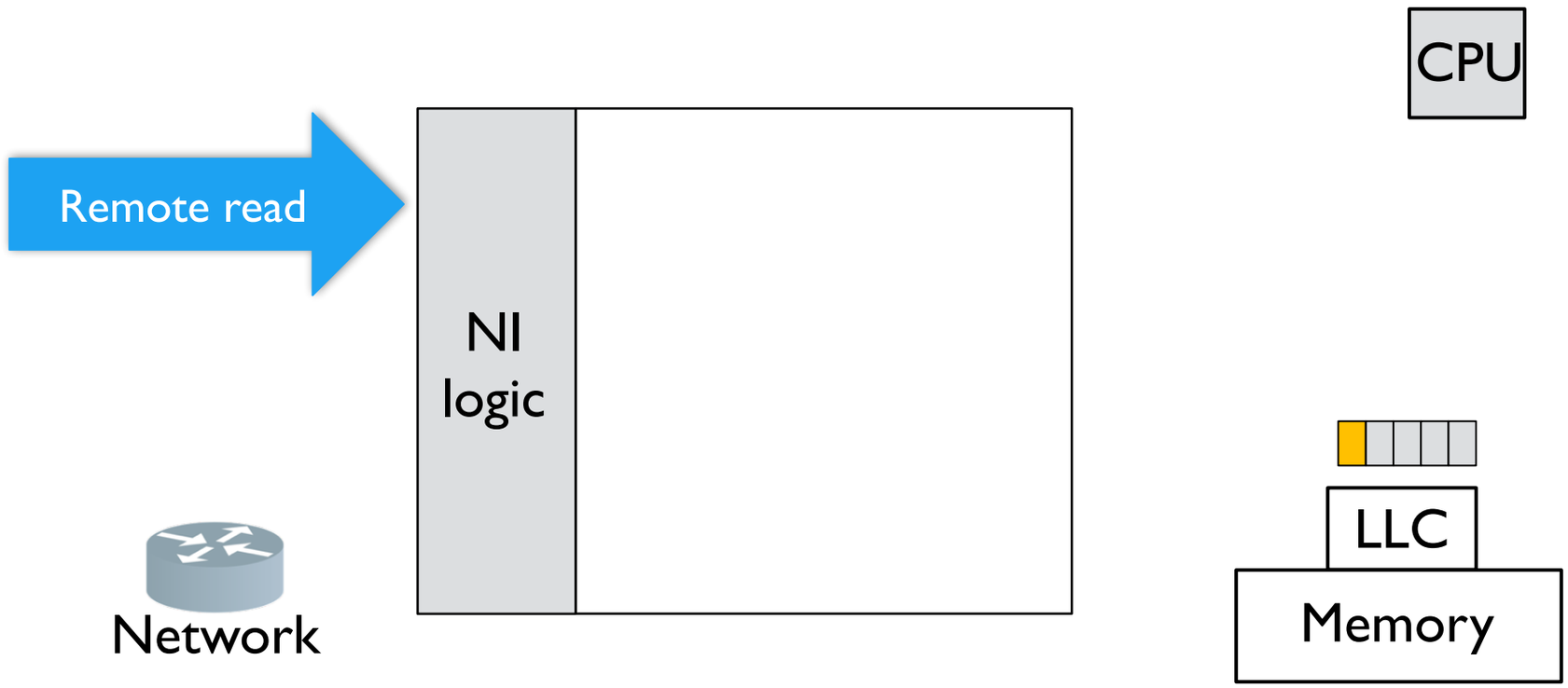




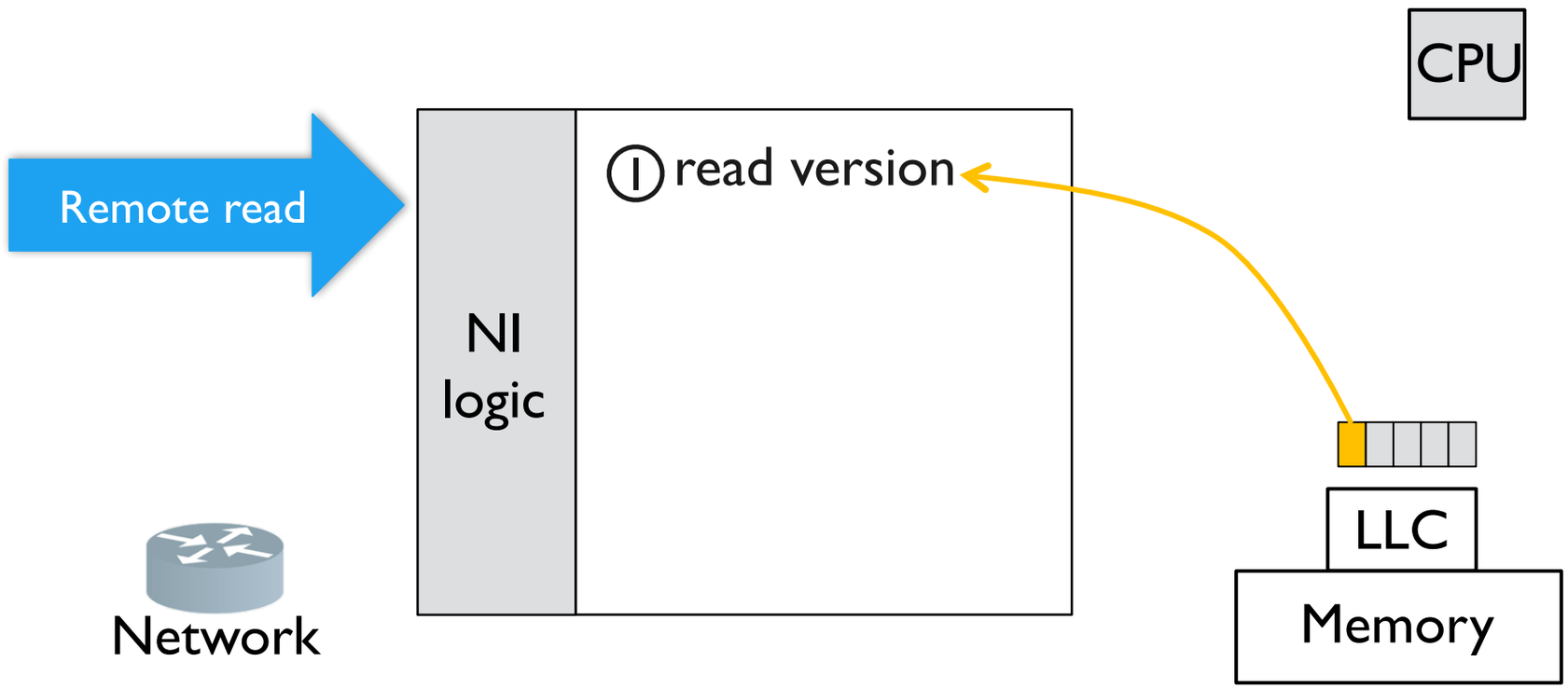
SABRes: Single Site Atomic Bulk Read



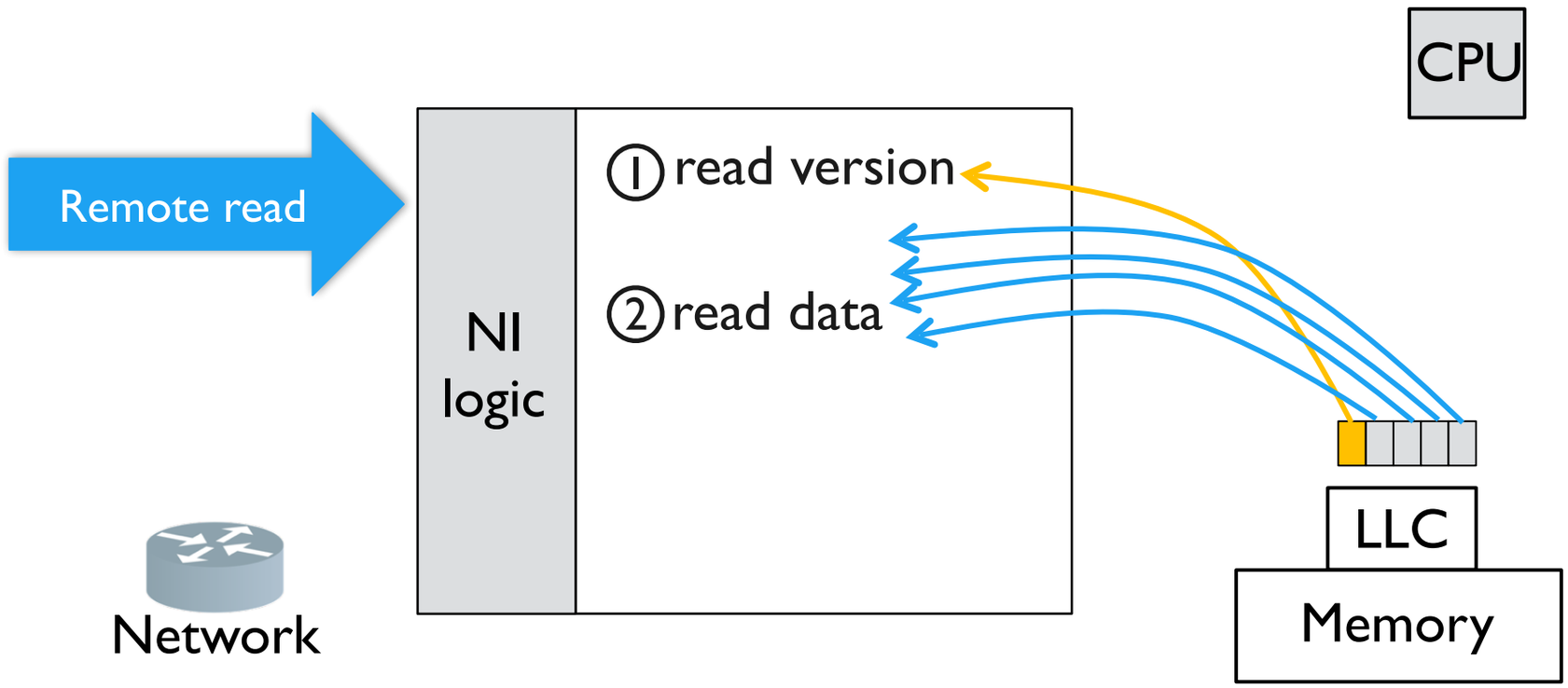
SABRes: Single Site Atomic Bulk Read



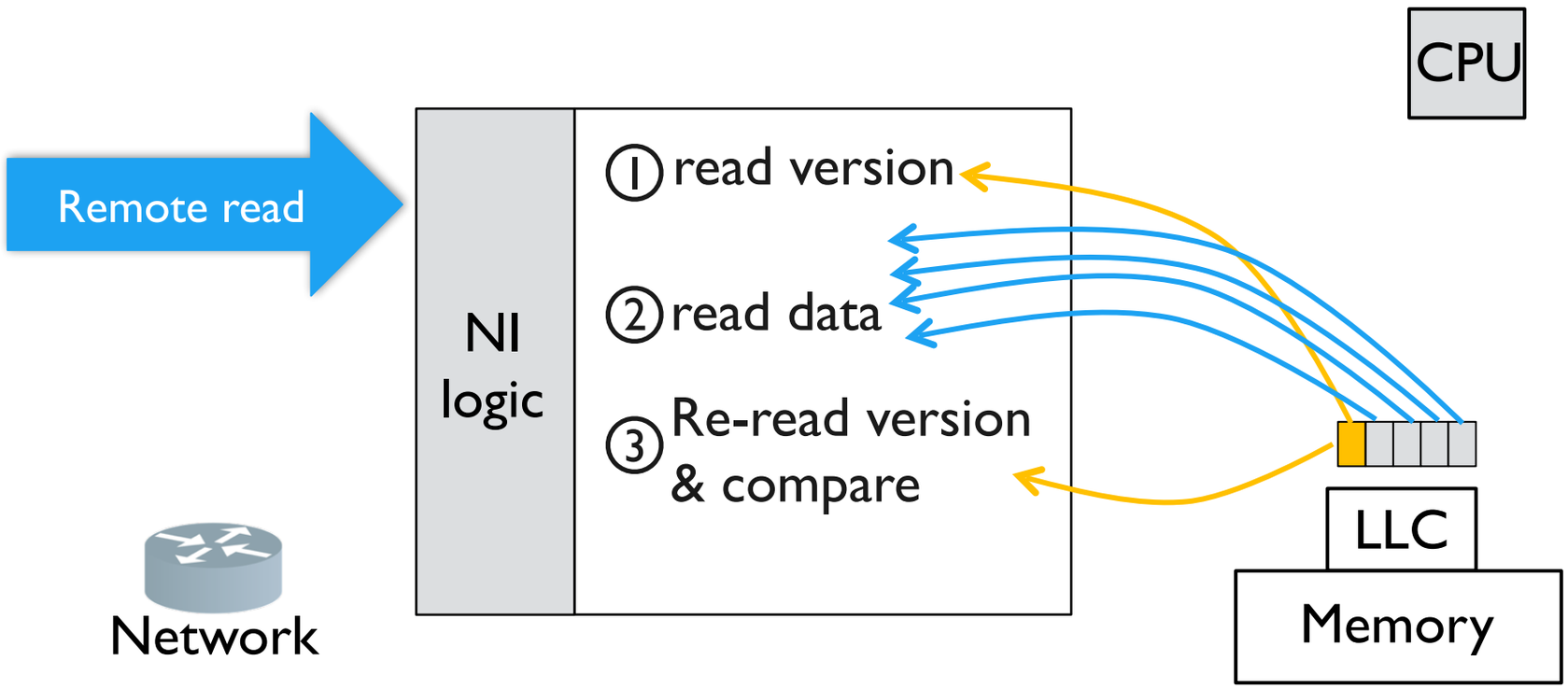
SABRes: Single Site Atomic Bulk Read



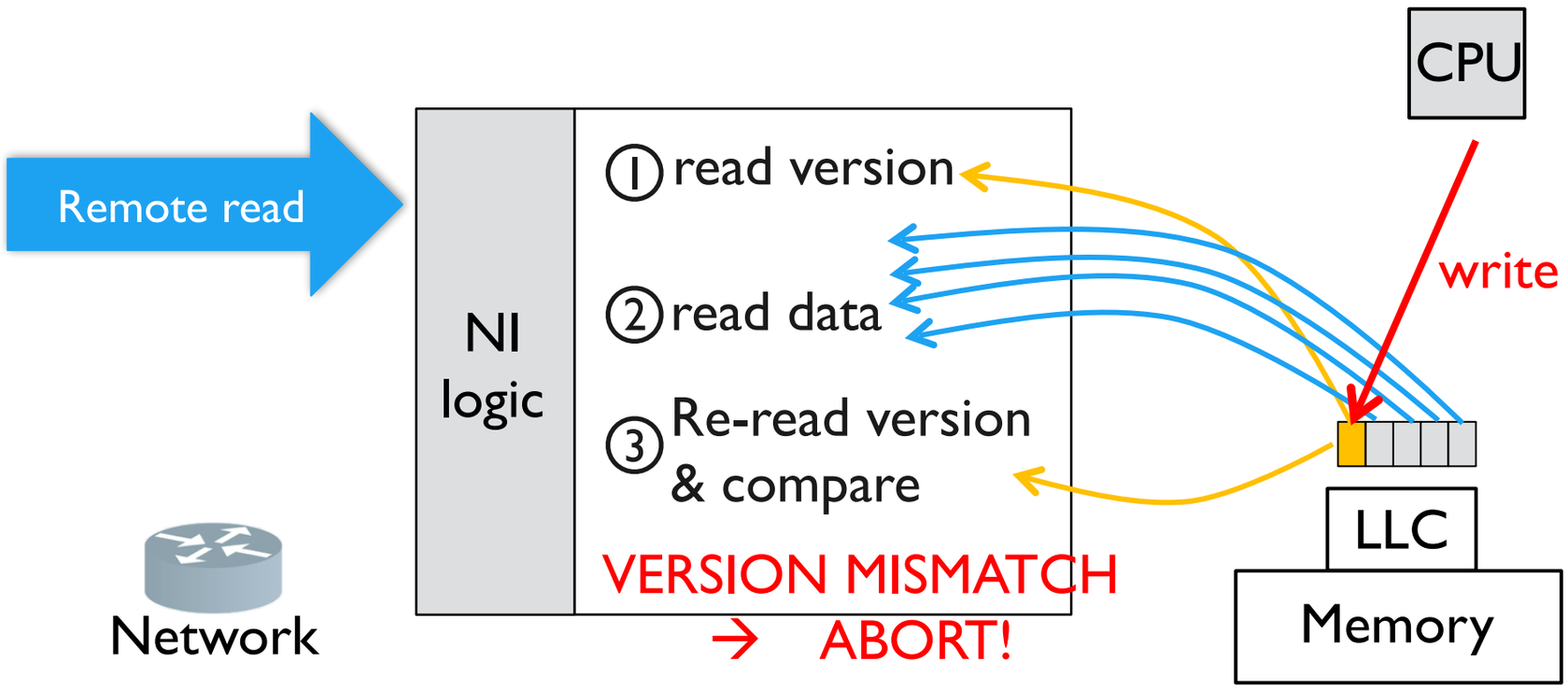
SABRes: Single Site Atomic Bulk Read



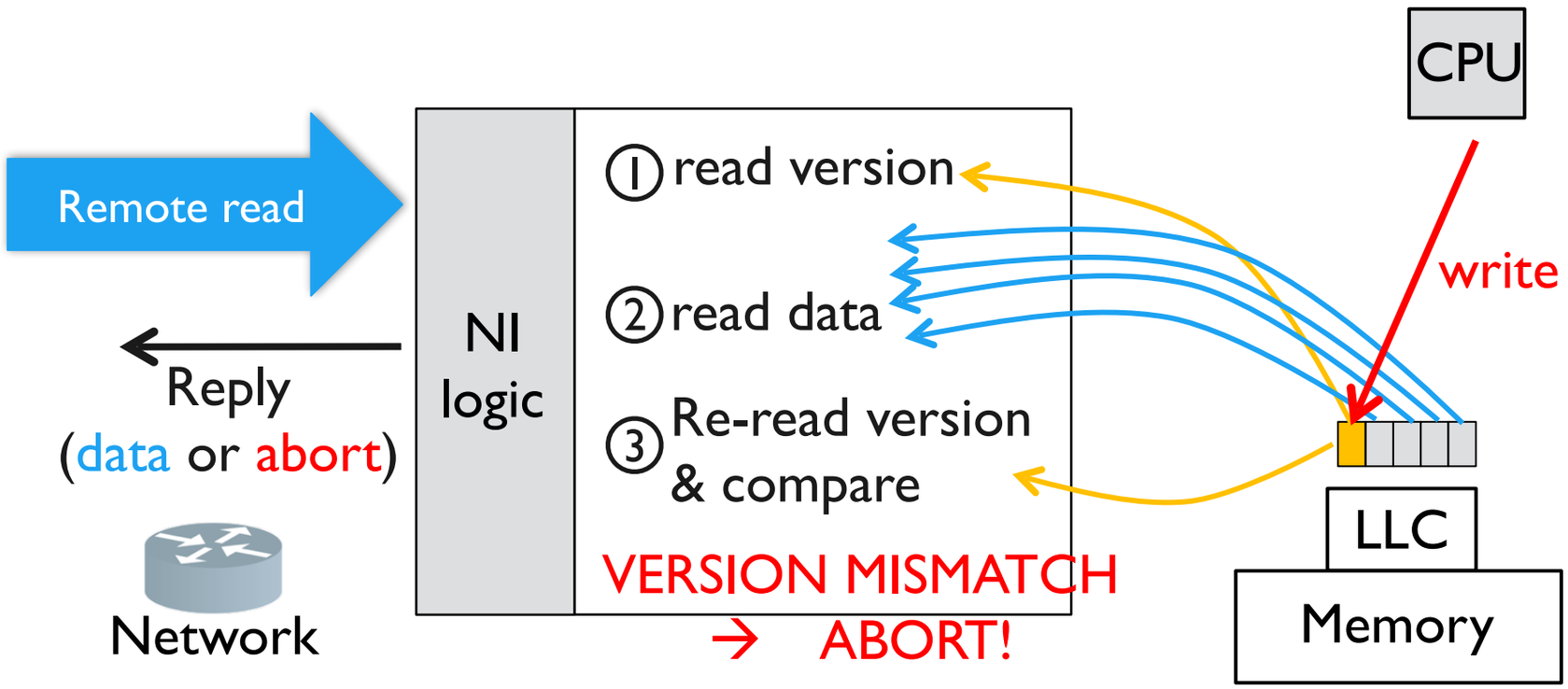
SABRes: Single Site Atomic Bulk Read



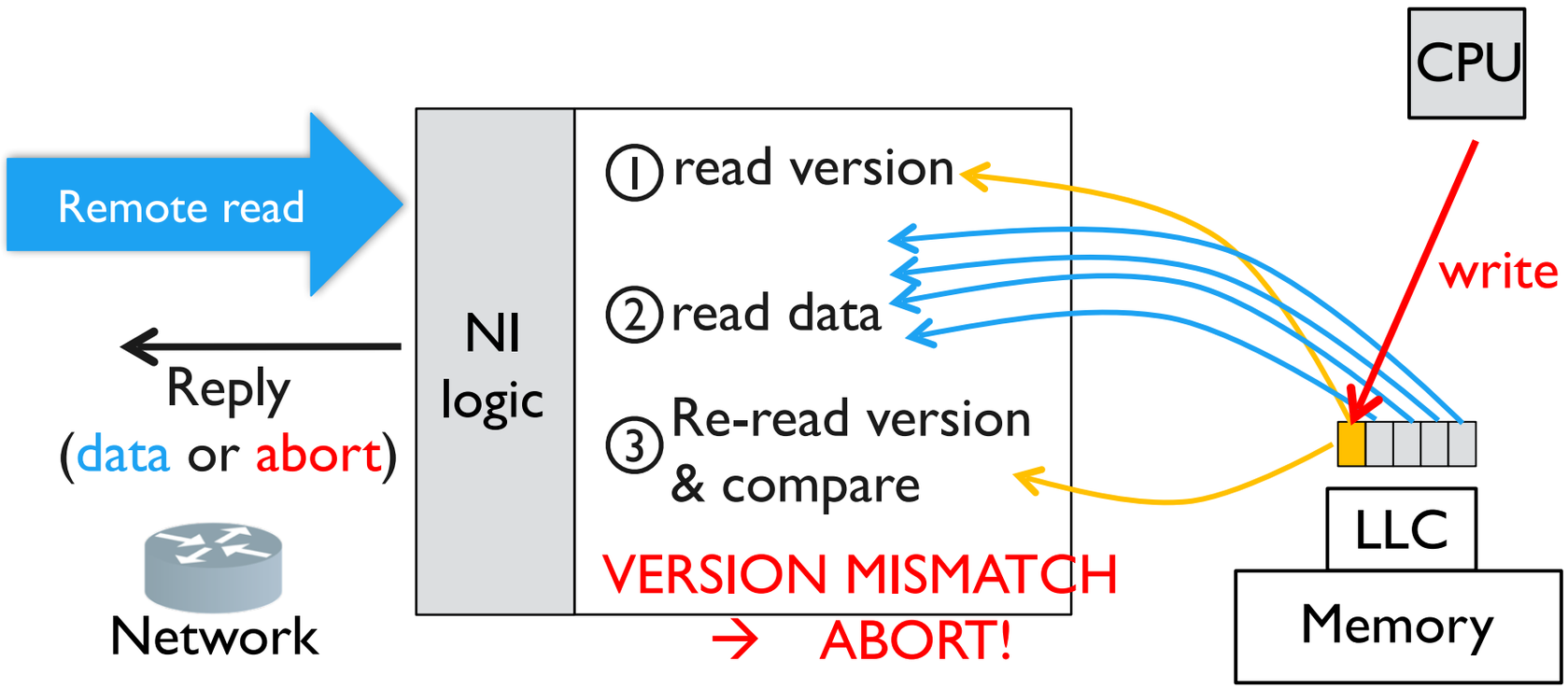
SABRes: Single Site Atomic Bulk Read



SABRes: Single Site Atomic Bulk Read

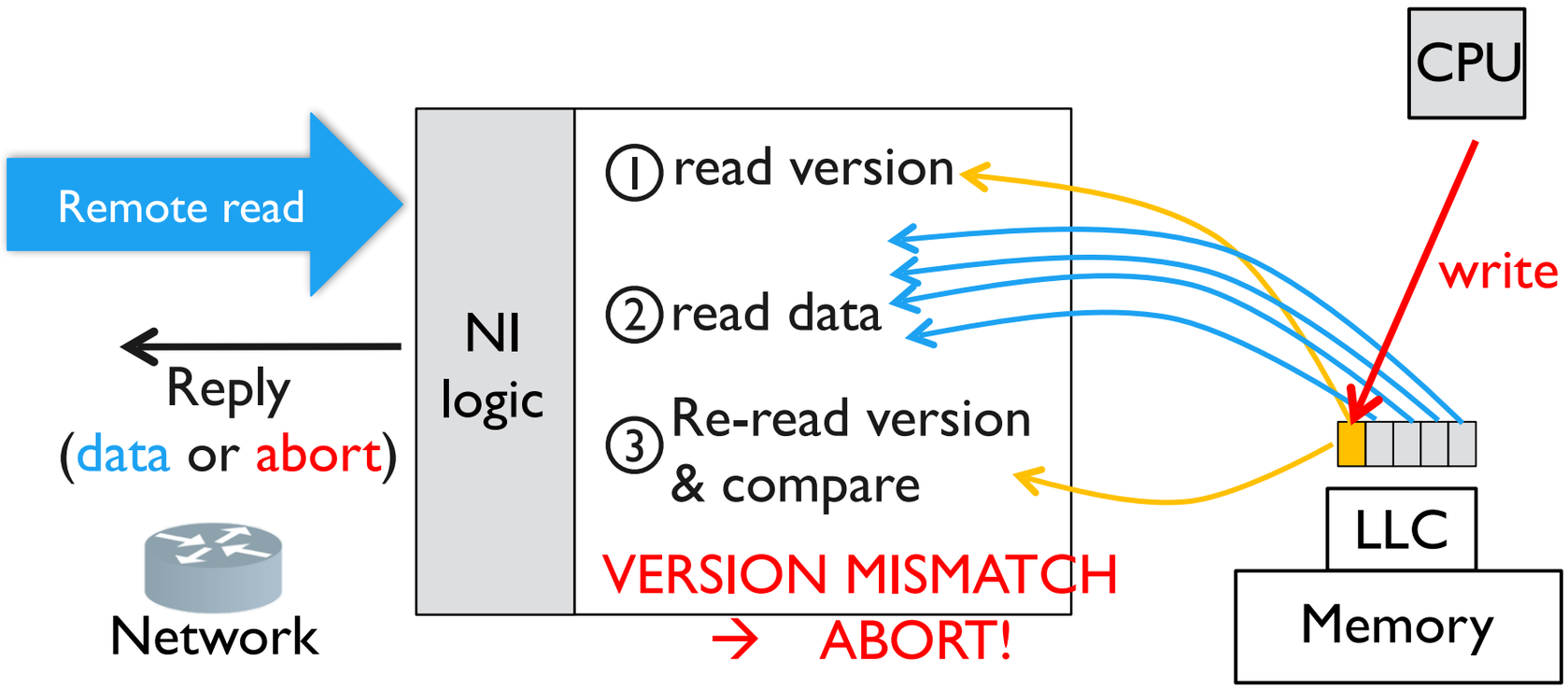


SABRes: Single Site Atomic Bulk Read



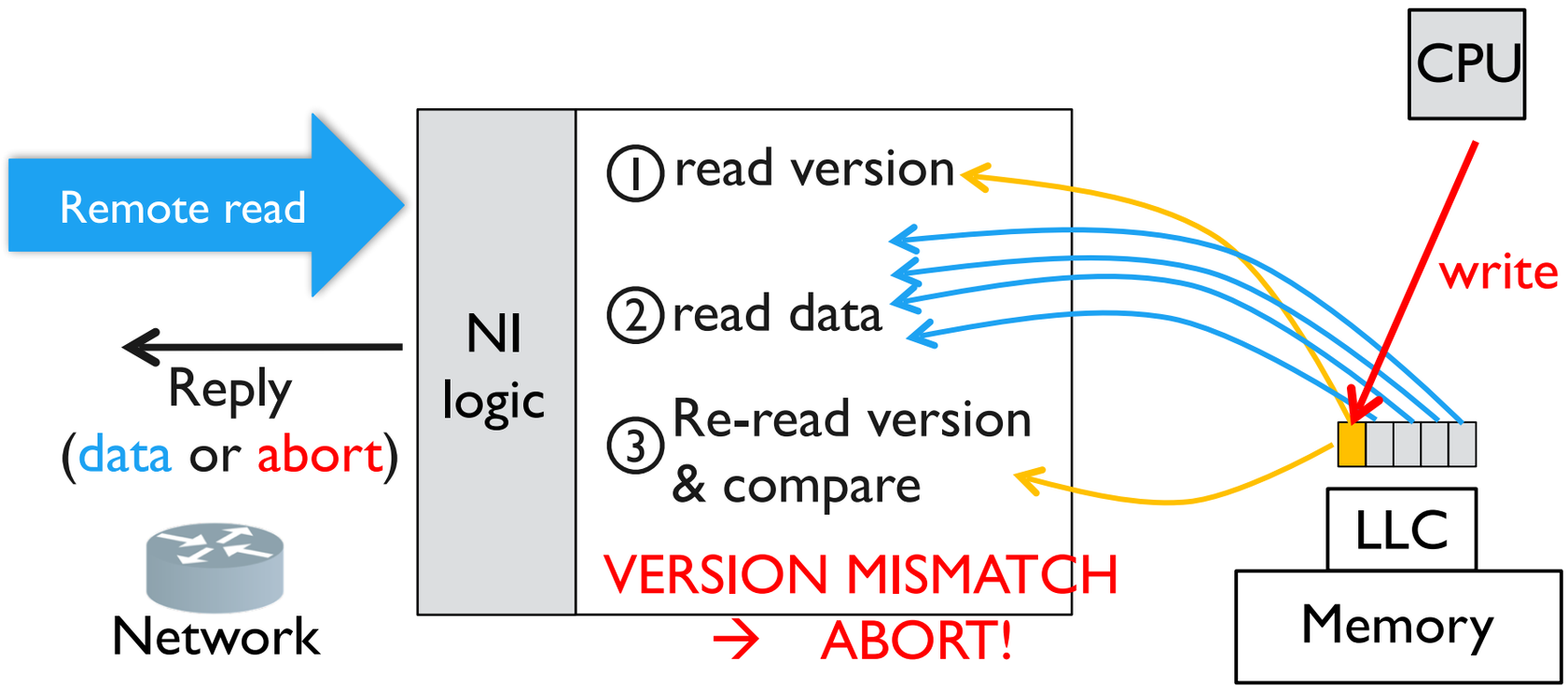
Speculatively read version and data → remove serial read latency

SABRes: Single Site Atomic Bulk Read



Speculatively read version and data → remove serial read latency

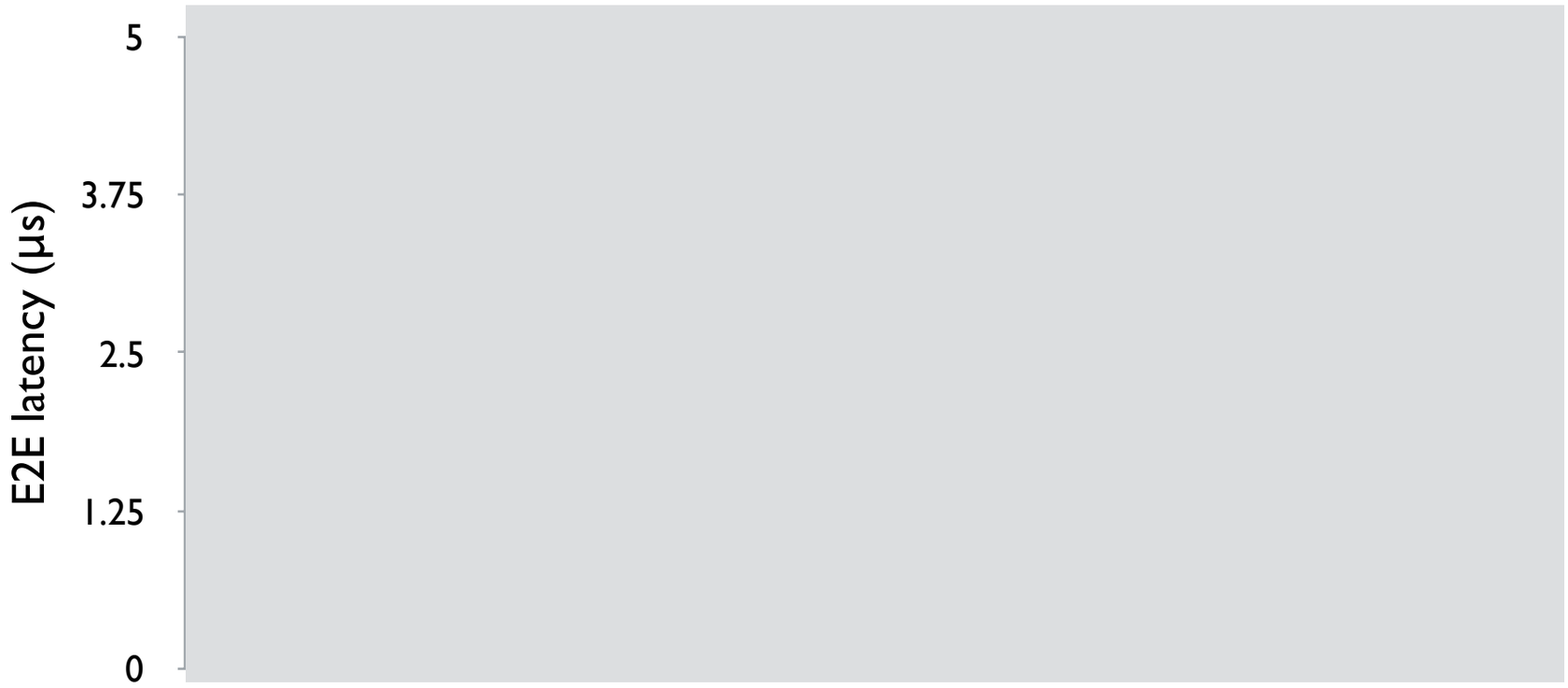
SABRes: Single Site Atomic Bulk Read



Speculatively read version and data → remove serial read latency

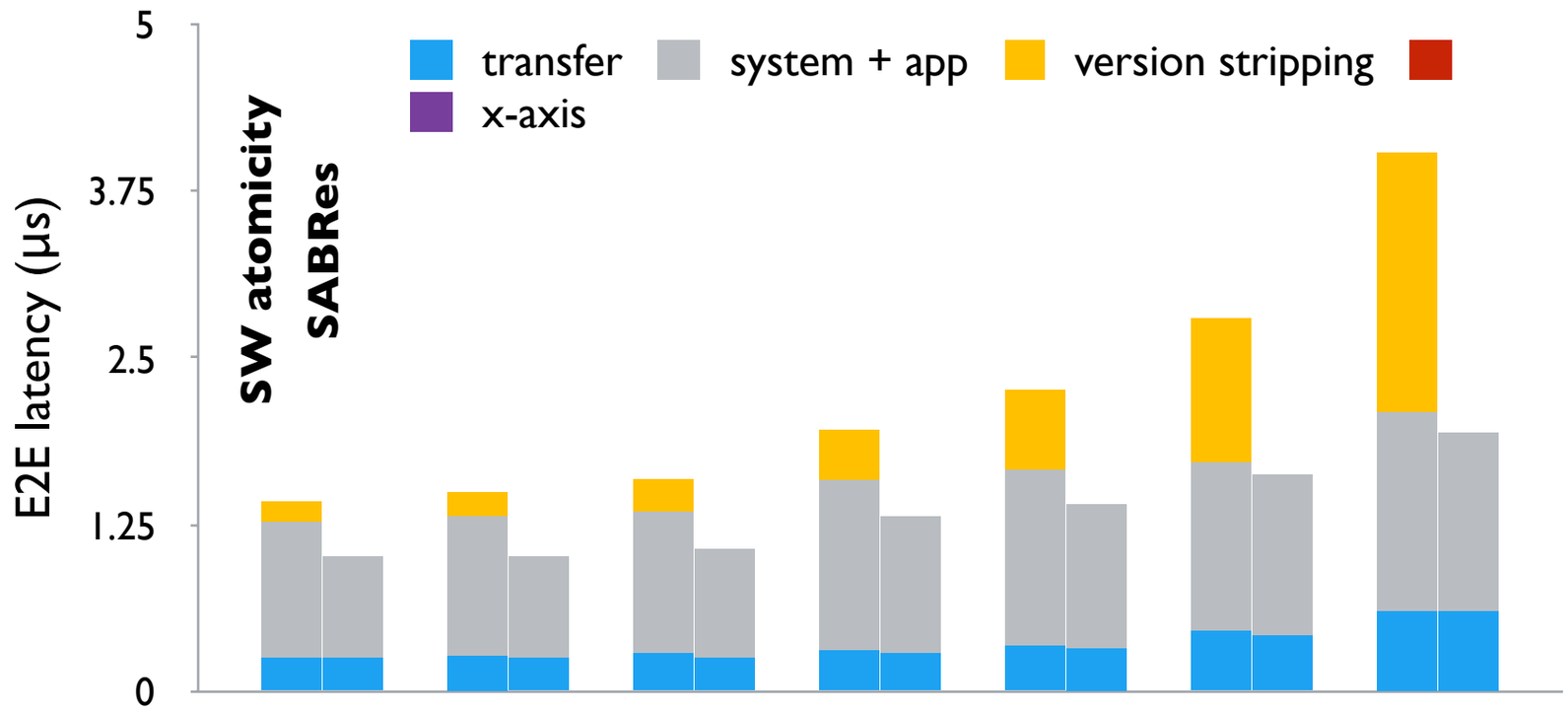
Simple hardware, atomicity w/ zero latency overhead

SABRes: Single Site Atomic Bulk Read



✓ Up to 2x latency & throughput benefit for distributed object stores

SABRes: Single Site Atomic Bulk Read



✓ Up to 2x latency & throughput benefit for distributed object stores

Outline

- Overview
- Current Technologies
- Challenges
  - Atomic Object Read
  - Network Interface for Many-Core Chip
- Conclusion

Interwork Interface Design for Manycore Chips

- NI placement & design is *key* for remote access performance
  - Obvious NI designs suffer from poor latency or bandwidth

- Contributions
  - NI design space exploration for manycore chips
  - Seamless integration of NI into chip's coherence domain
  - Novel *Split NI* design, optimizing for both latency and bandwidth

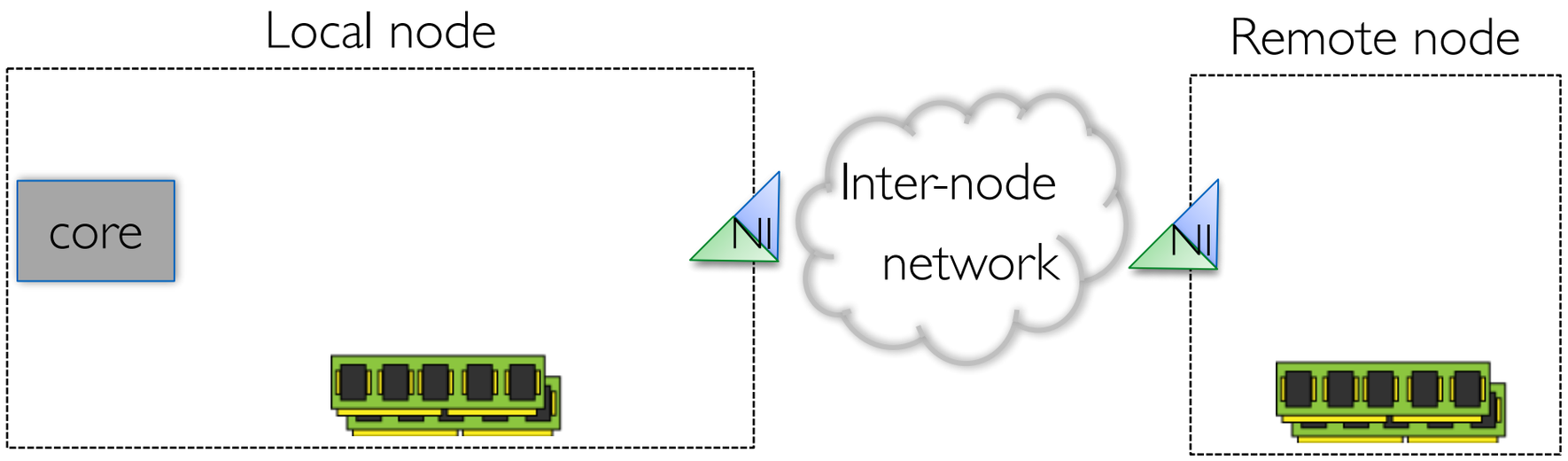
Goal: "Low-latency, high-bandwidth NI design for manycore chips"

User-level Remote Memory Access

RDMA-like Queue-Pair (QP) model

Cores and NIs communicate through *cacheable memory-mapped* queues

- Work Queue (WQ) and Completion Queue (CQ)

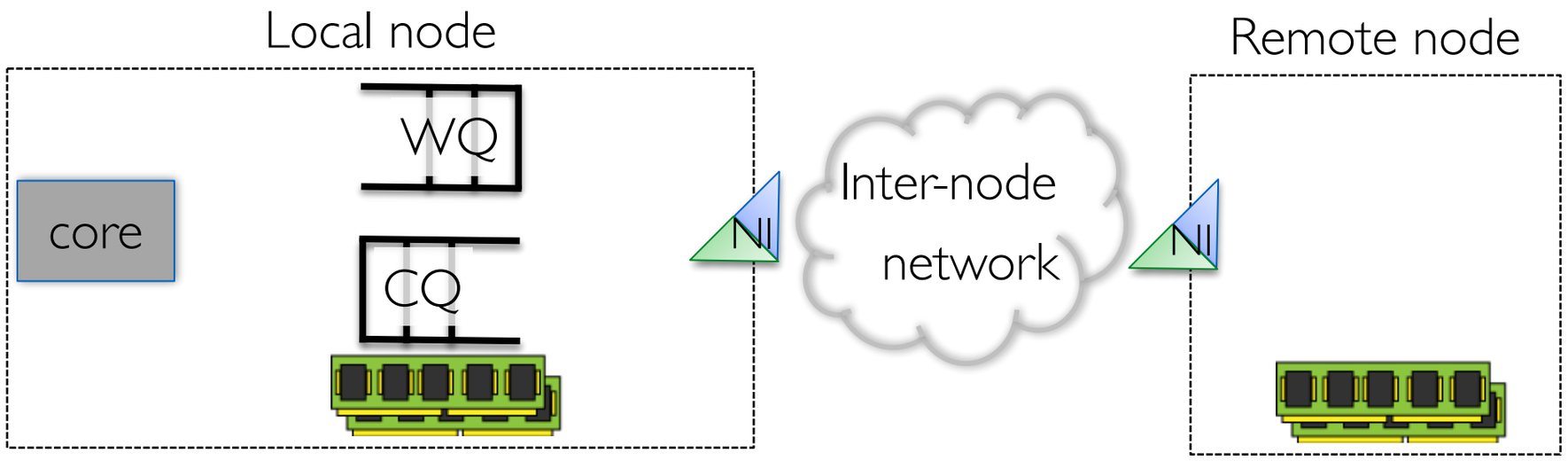


User-level Remote Memory Access

RDMA-like Queue-Pair (QP) model

Cores and NIs communicate through *cacheable memory-mapped* queues

- Work Queue (WQ) and Completion Queue (CQ)

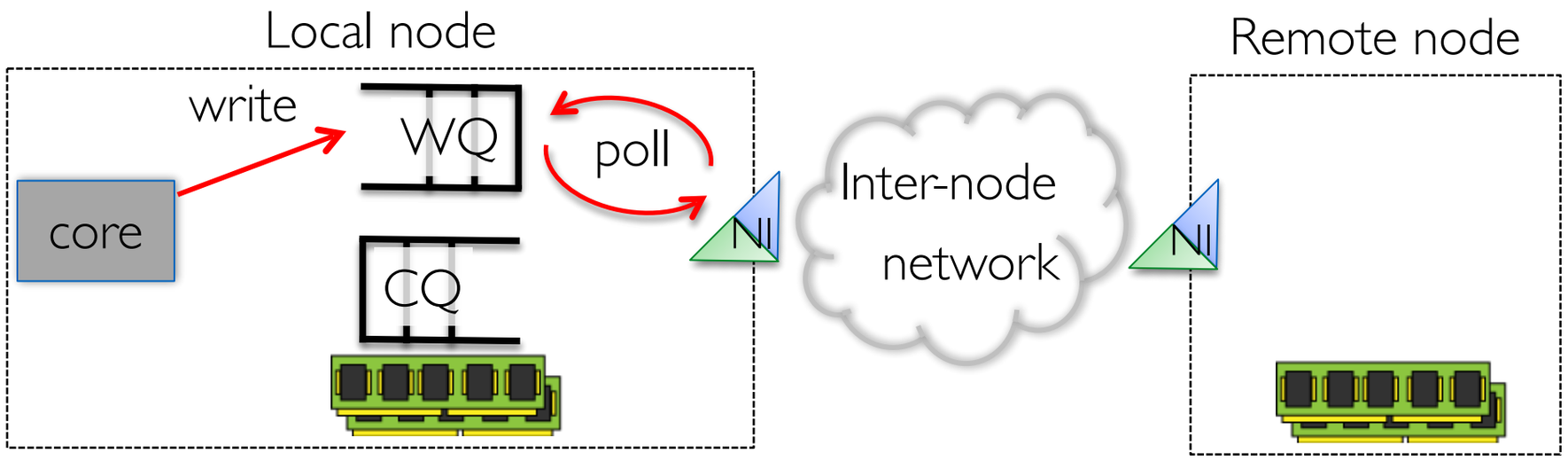


User-level Remote Memory Access

RDMA-like Queue-Pair (QP) model

Cores and NIs communicate through *cacheable memory-mapped* queues

- Work Queue (WQ) and Completion Queue (CQ)

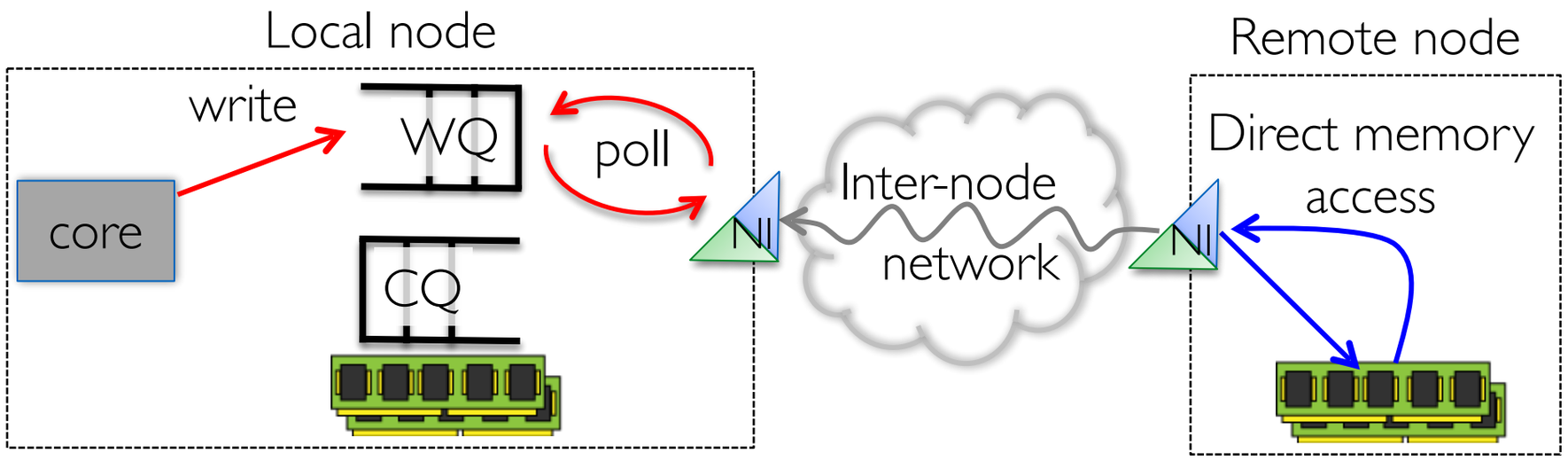


User-level Remote Memory Access

RDMA-like Queue-Pair (QP) model

Cores and NIs communicate through *cacheable memory-mapped* queues

- Work Queue (WQ) and Completion Queue (CQ)

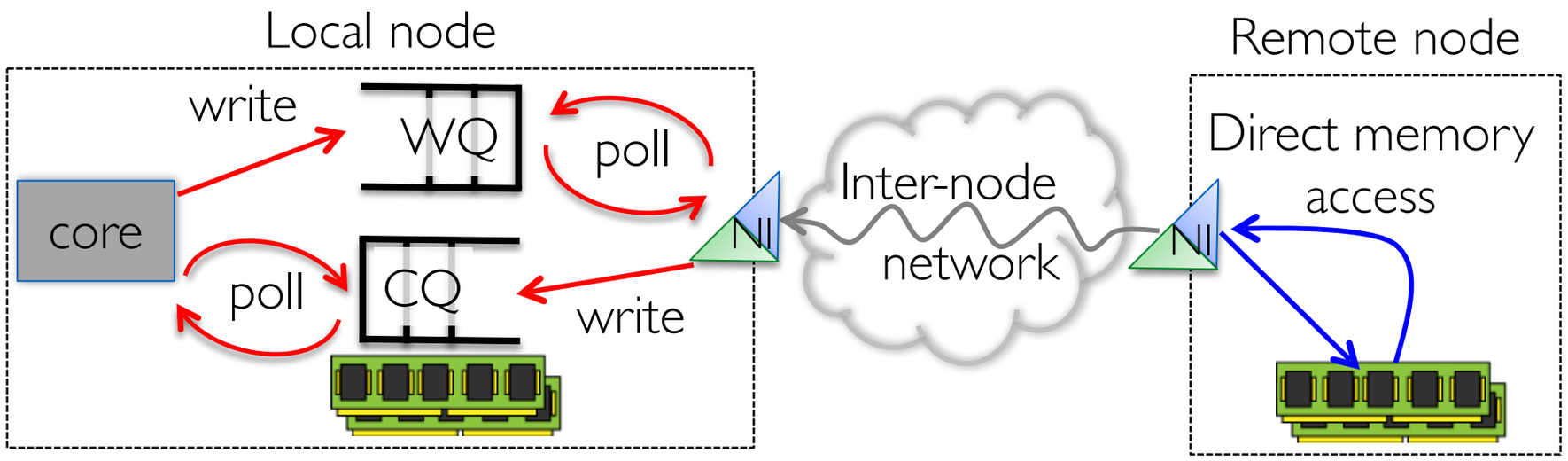


User-level Remote Memory Access

RDMA-like Queue-Pair (QP) model

Cores and NIs communicate through *cacheable memory-mapped* queues

- Work Queue (WQ) and Completion Queue (CQ)



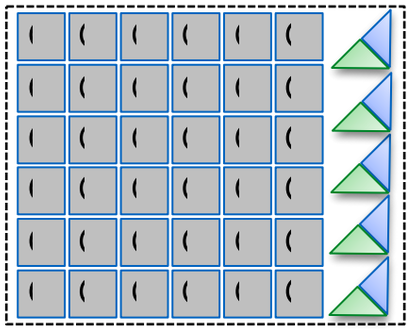
Implications of Manycore Chips on Remote Access

More Cores → Higher request rate

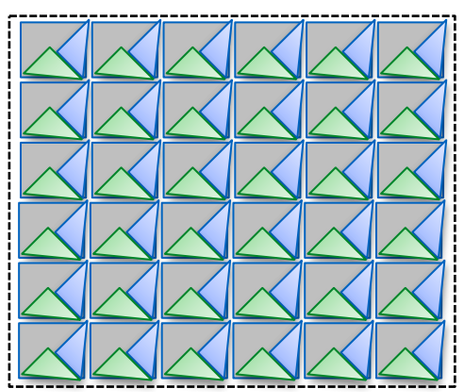
NI capabilities have to match chip's communication demands

Approaches →

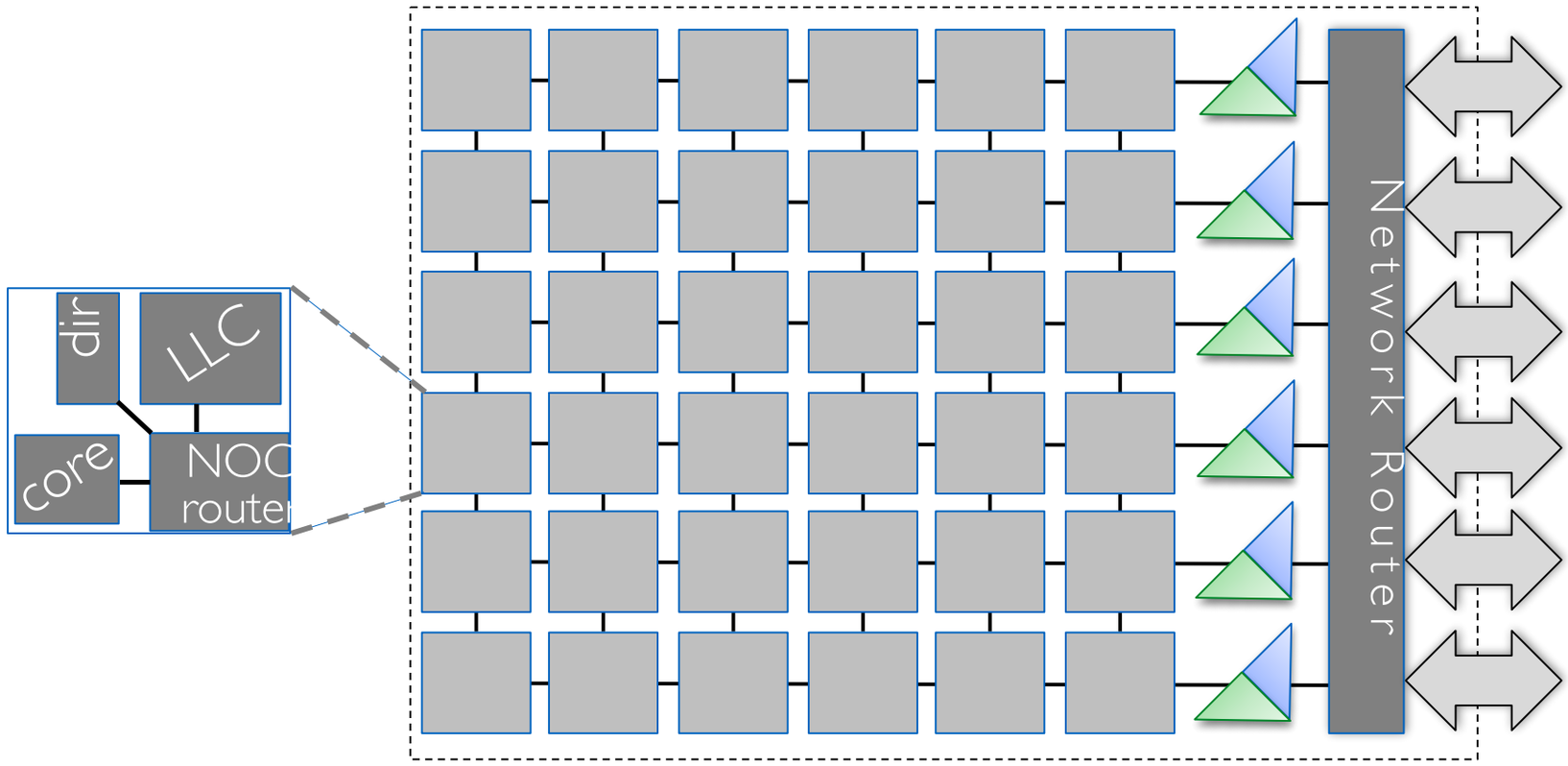
- ❑ <u>Scale NI across edge</u>
- ❑ → Close to network pins
- ❑ → Access to NOC's full bisection bandwidth
- ❑ → Large average core to NI distance ☹



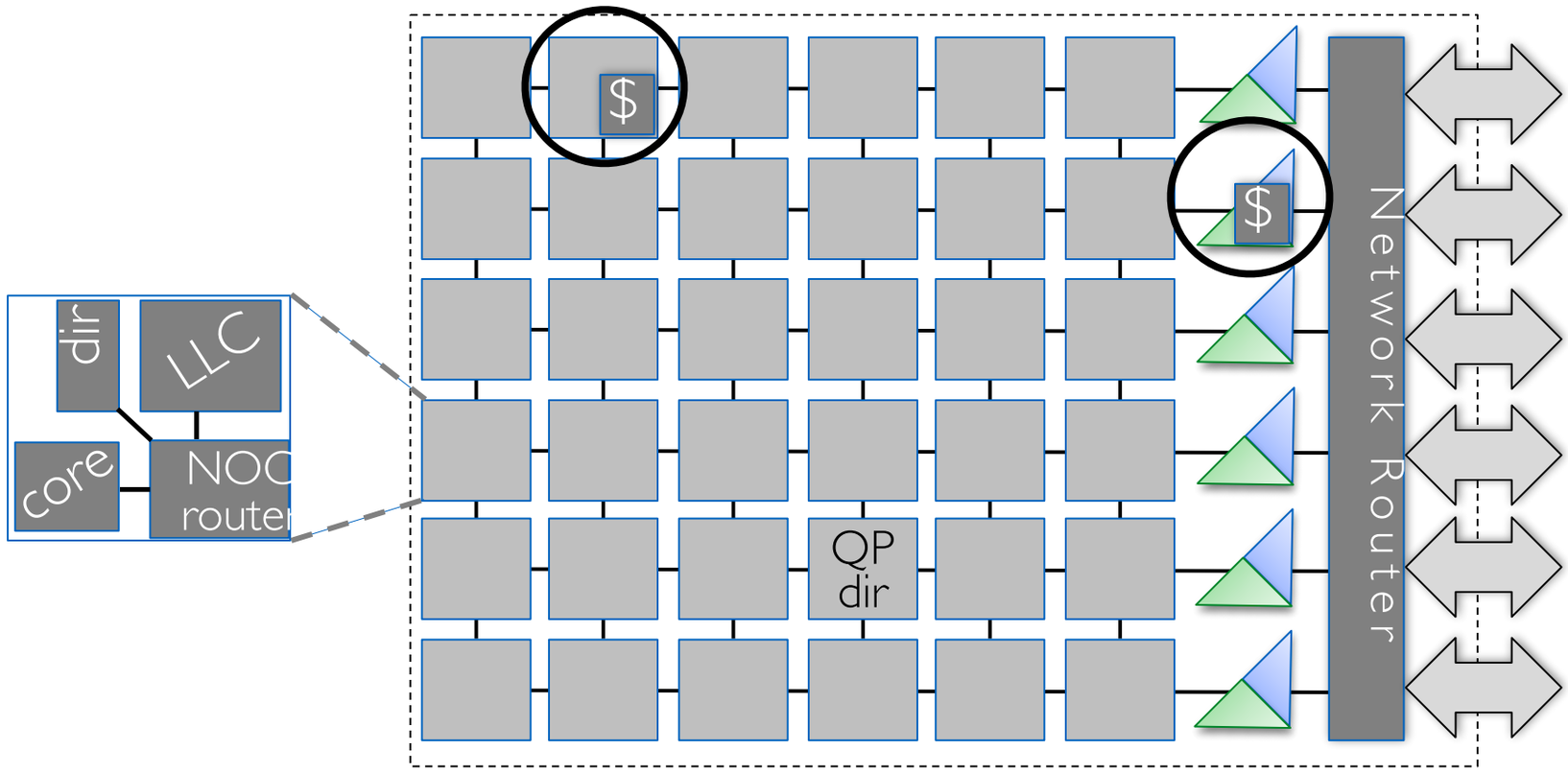
- ❑ <u>Collocate NI logic per core, to localize all interactions</u>
- ❑ → Transparent to coherence mechanism
- ❑ → No modifications to core's IP block



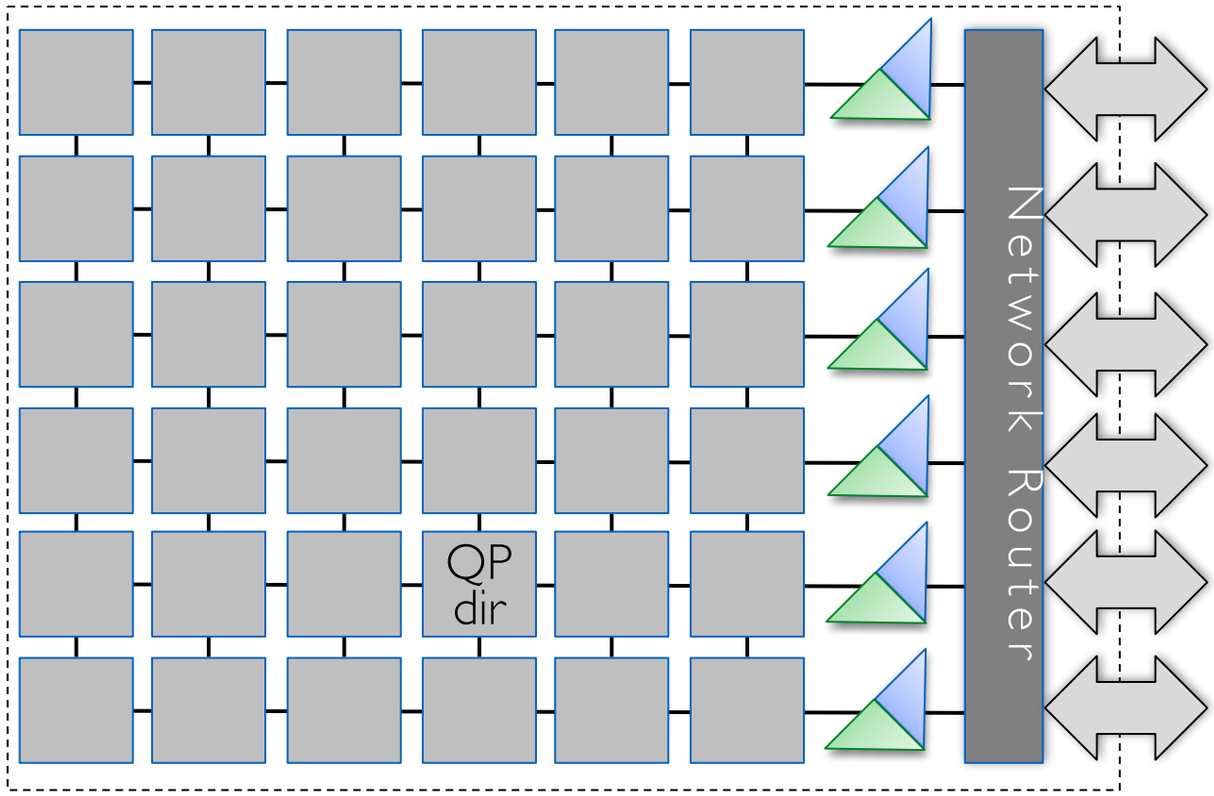
Edge NI 4-Cache-Block Remote Read



Edge NI 4-Cache-Block Remote Read



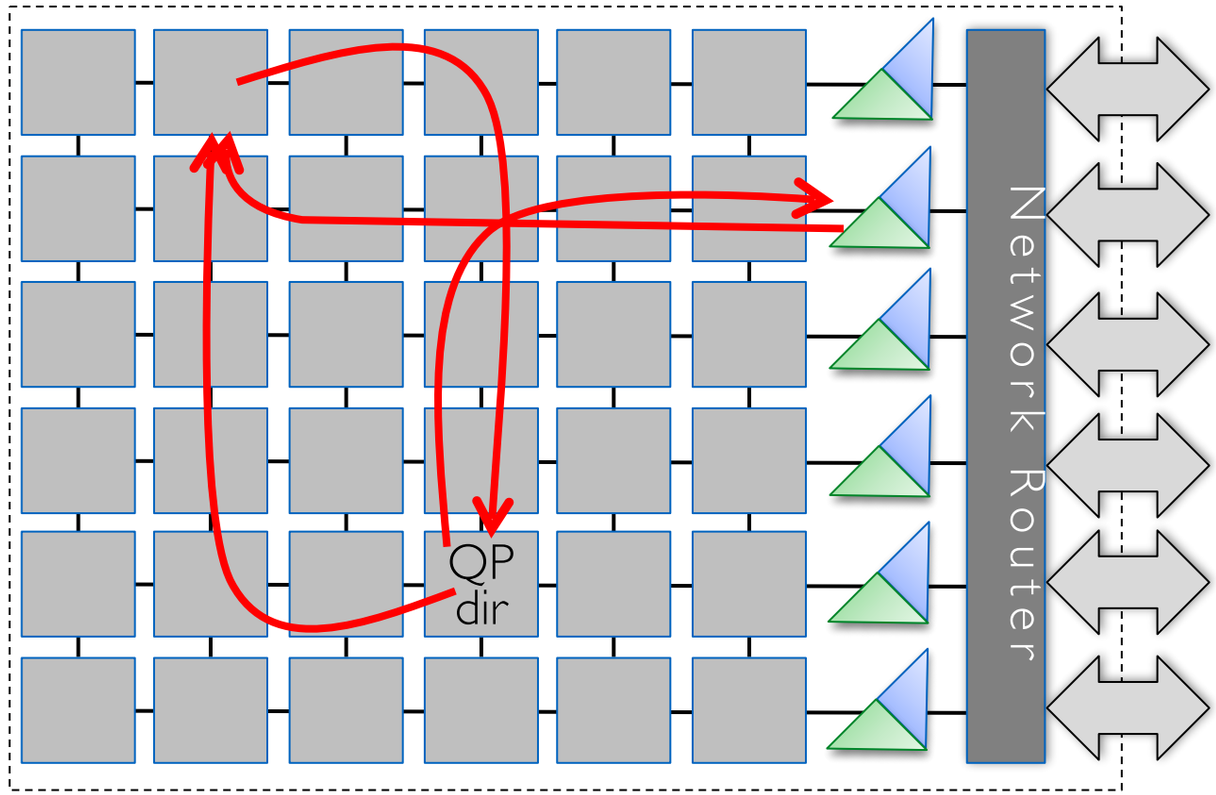
Edge NI 4-Cache-Block Remote Read



Edge NI 4-Cache-Block Remote Read

QP interactions

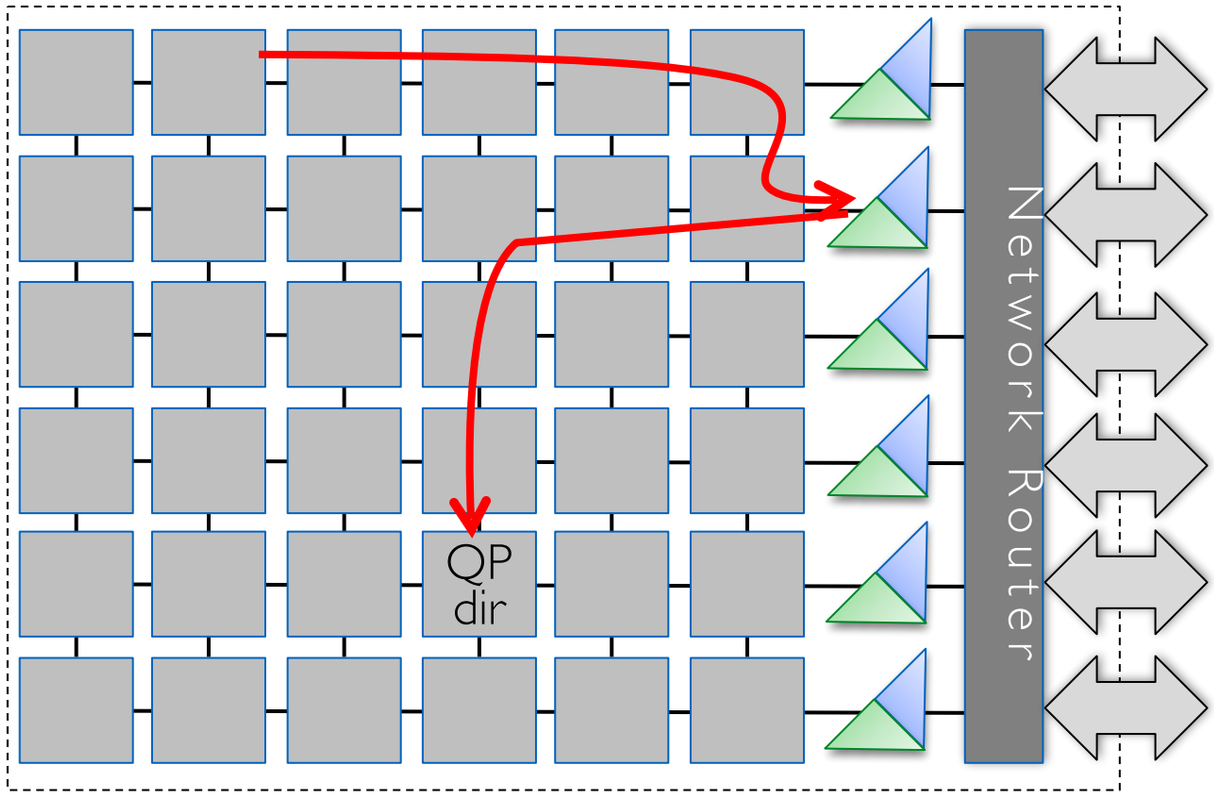
Core writes WQ



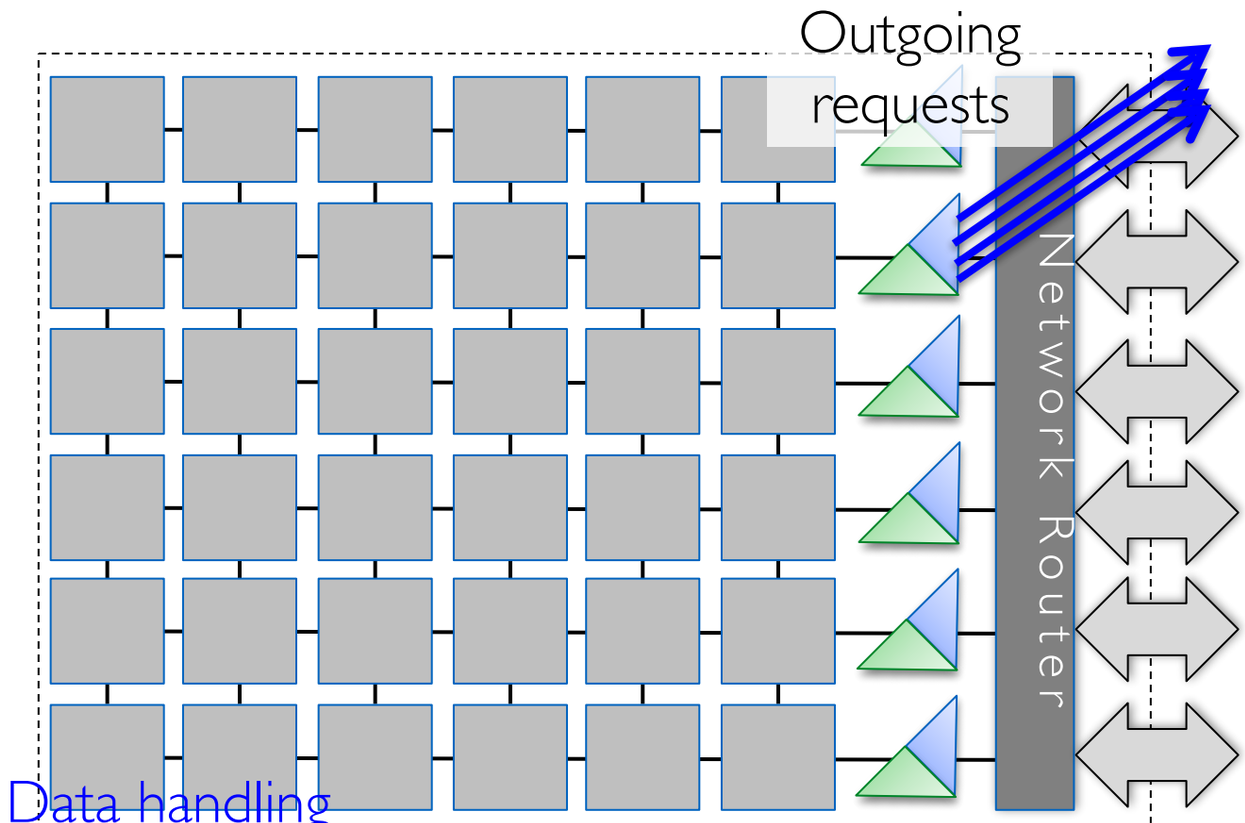
Edge NI 4-Cache-Block Remote Read

QP interactions

NI reads WQ

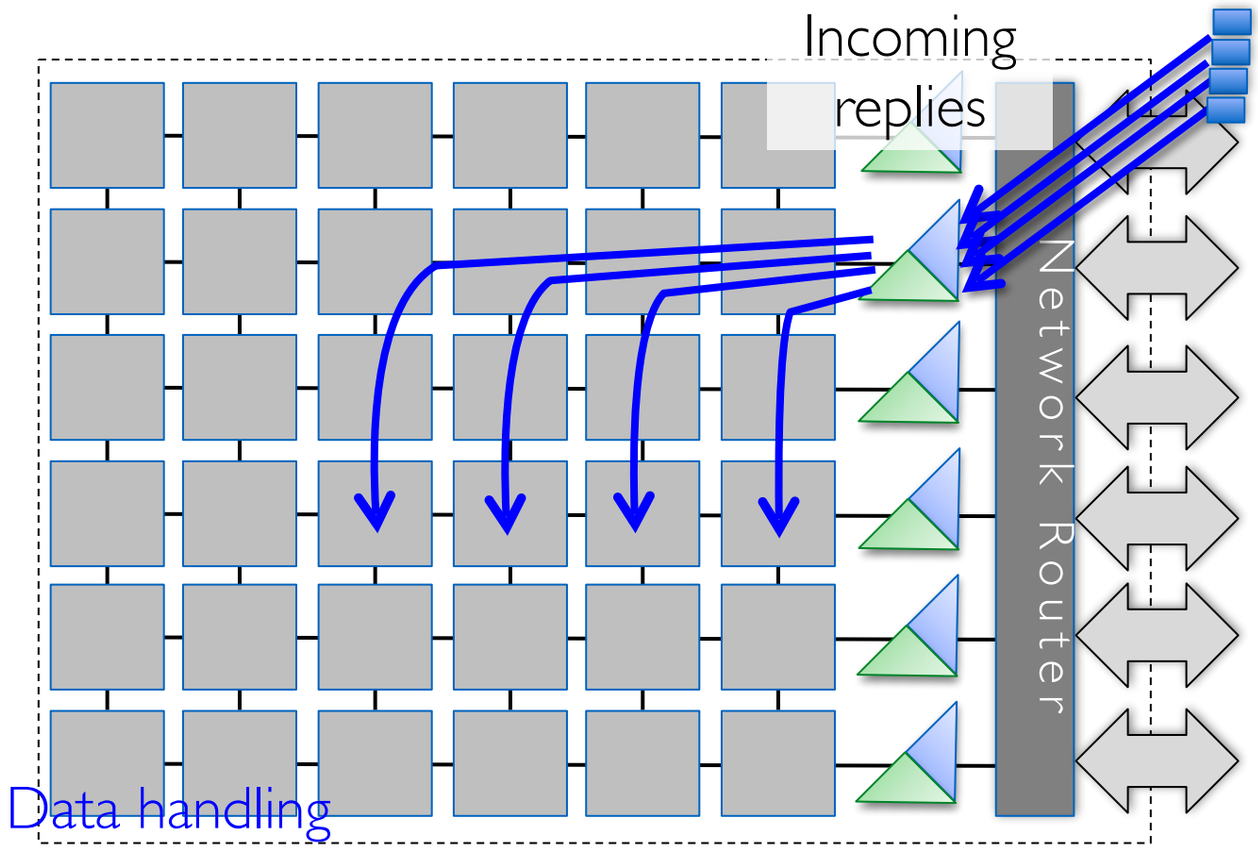


Edge NI 4-Cache-Block Remote Read



NI unrolls to cache-block-sized requests

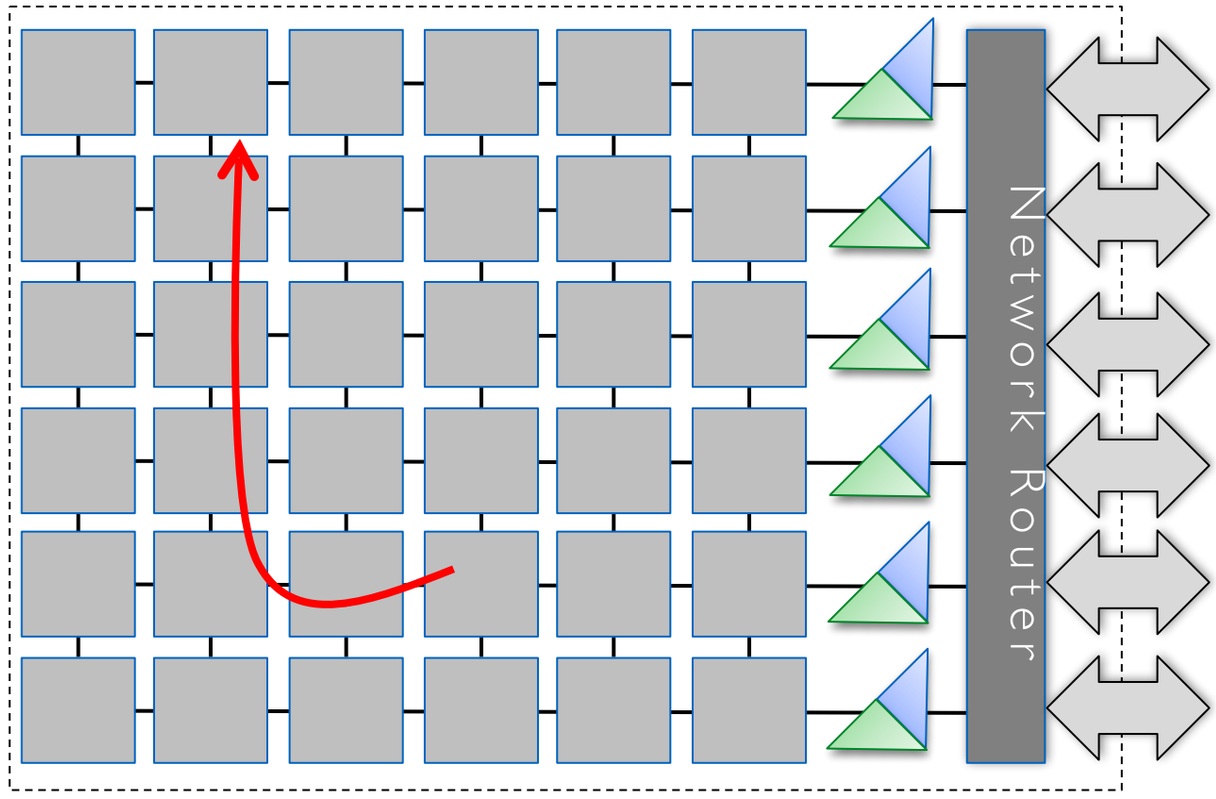
Edge NI 4-Cache-Block Remote Read



NI unrolls to cache-block-sized requests

Edge NI 4-Cache-Block Remote Read

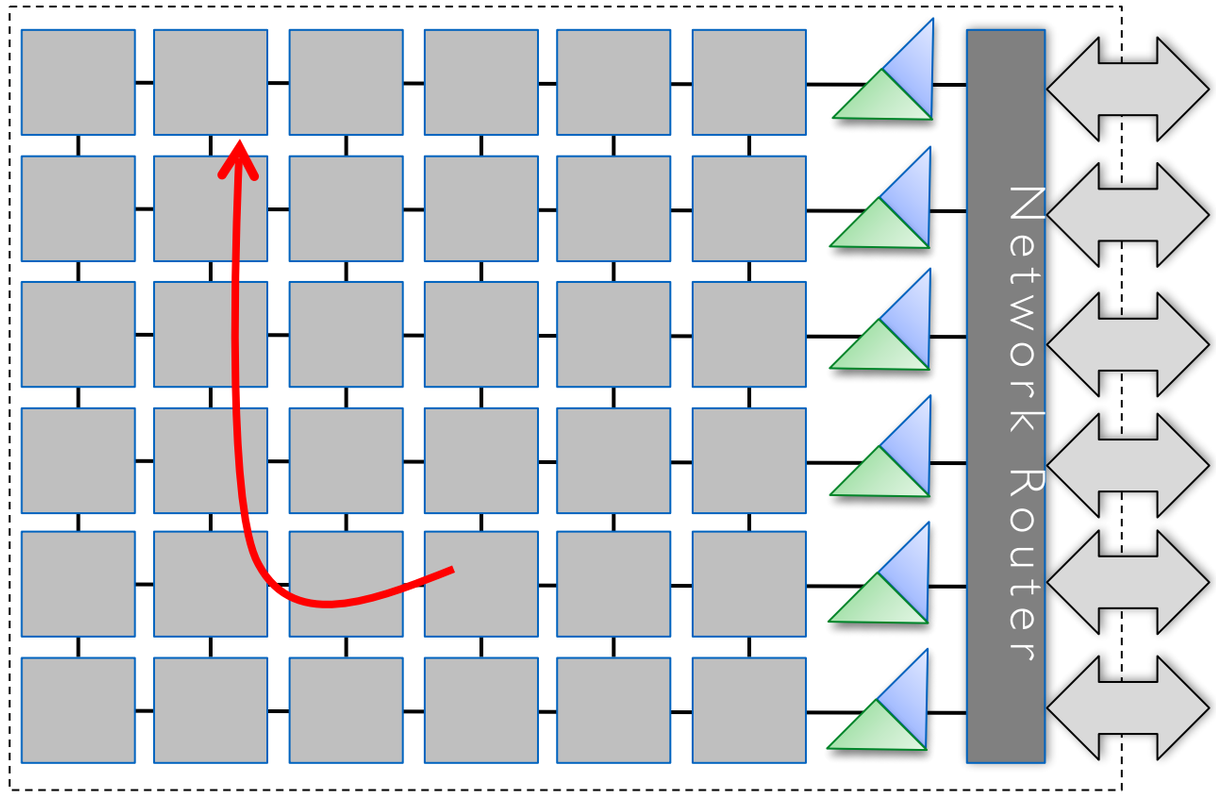
NI writes CQ
Core reads CQ



Data transfer completed. Repeat QP interactions to write CQ

Edge NI 4-Cache-Block Remote Read

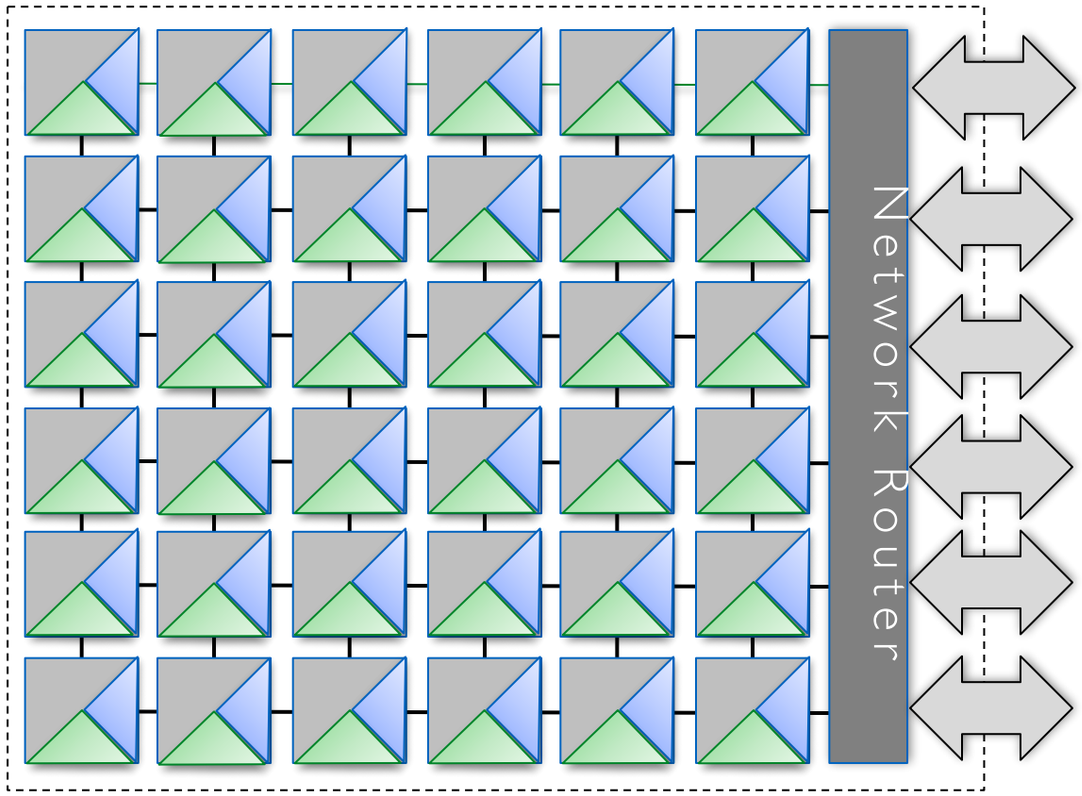
NI writes CQ
Core reads CQ



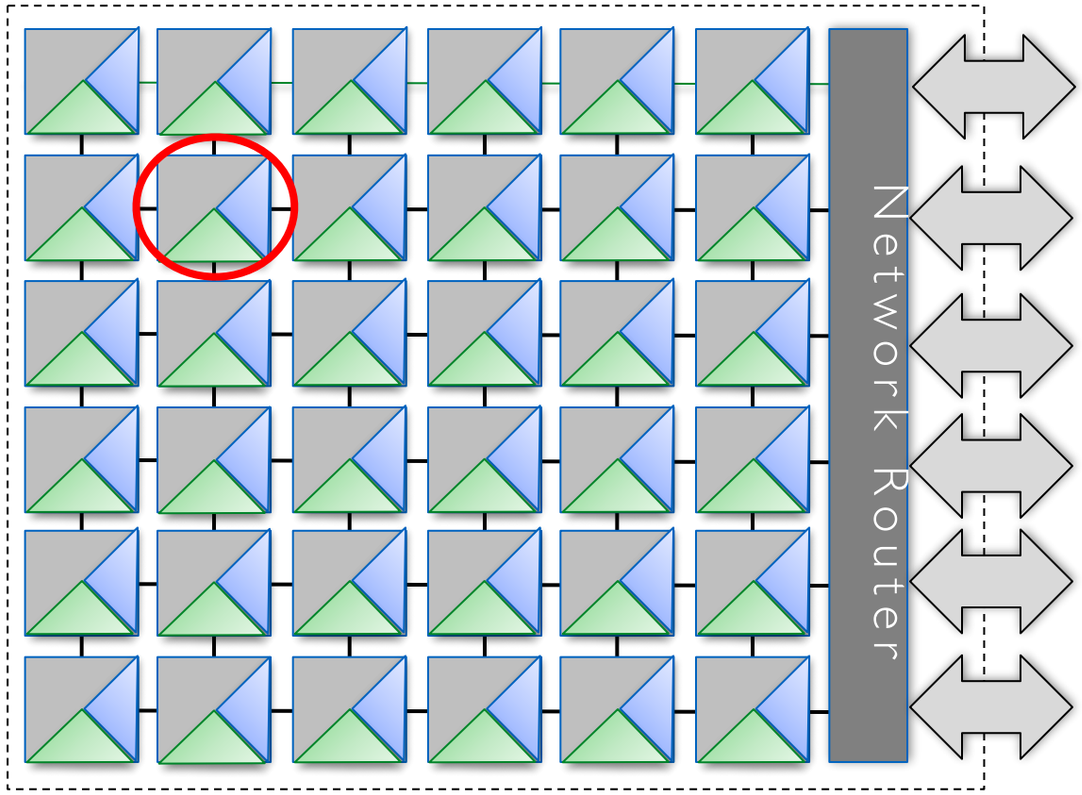
Bandwidth ✓
Latency ✗

QP interactions account for up to 50% of end-to-end latency

Per-Core NI 4-Cache-Block Remote Read

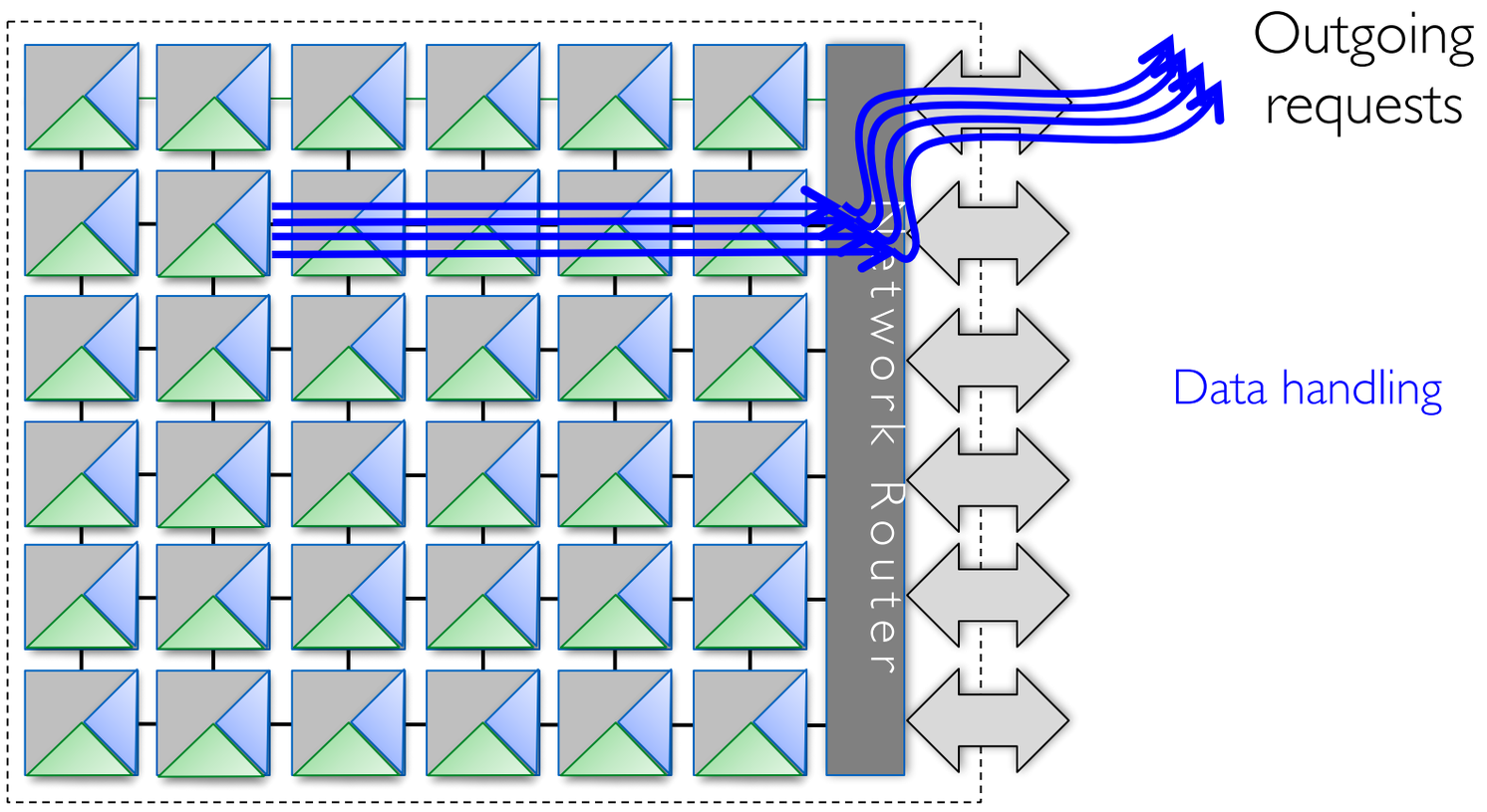


Per-Core NI 4-Cache-Block Remote Read

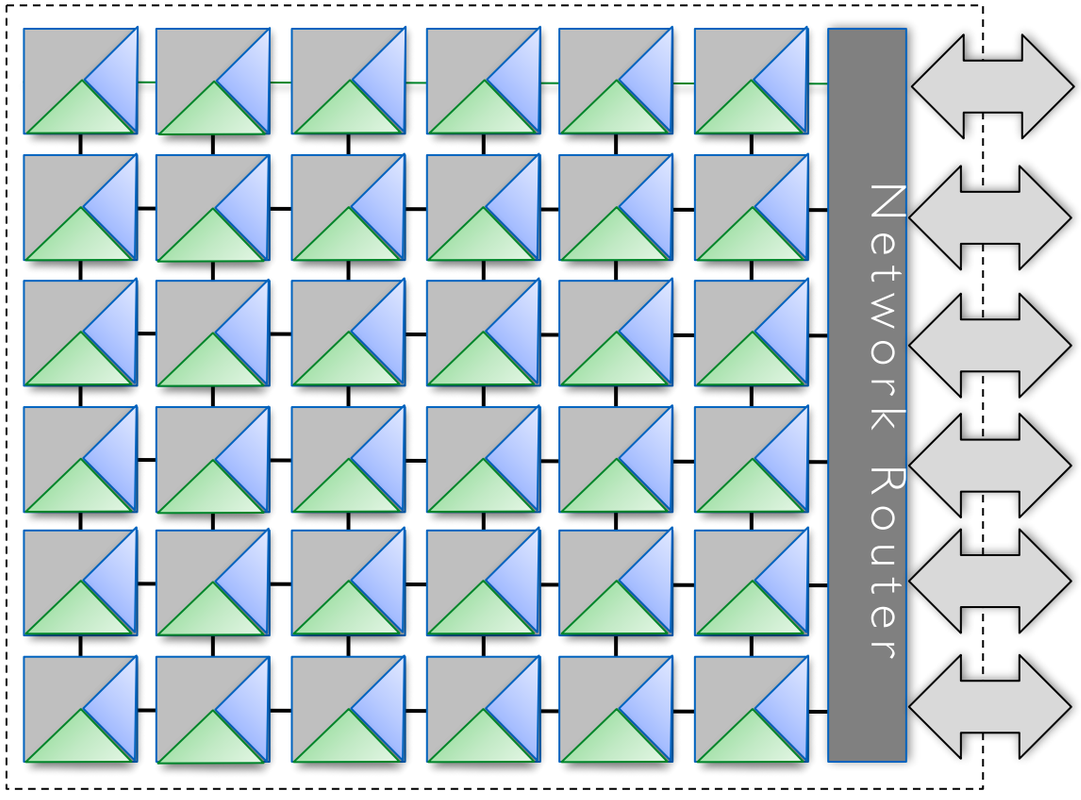


All QP interactions local!

Per-Core NI 4-Cache-Block Remote Read

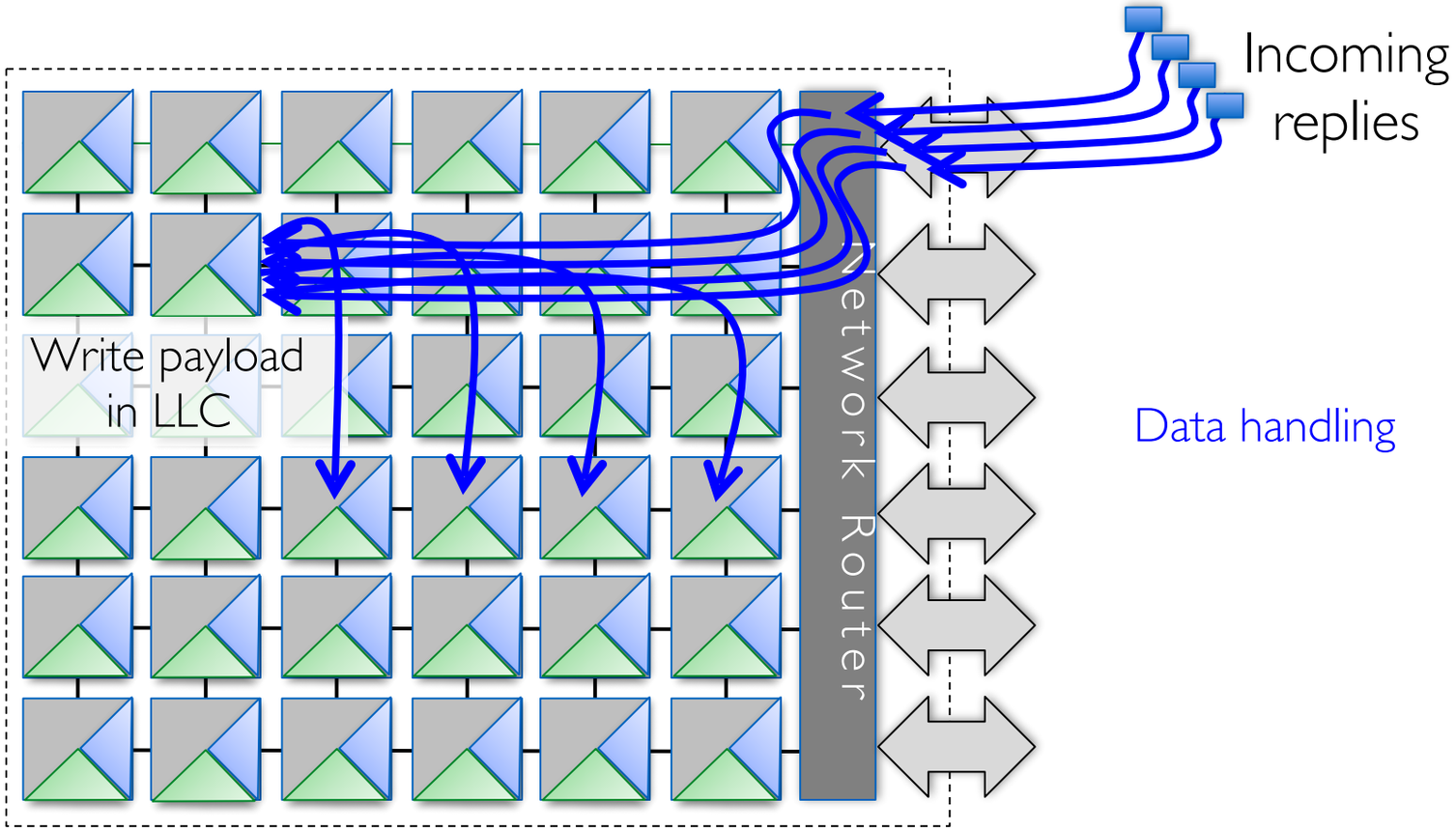


Per-Core NI 4-Cache-Block Remote Read

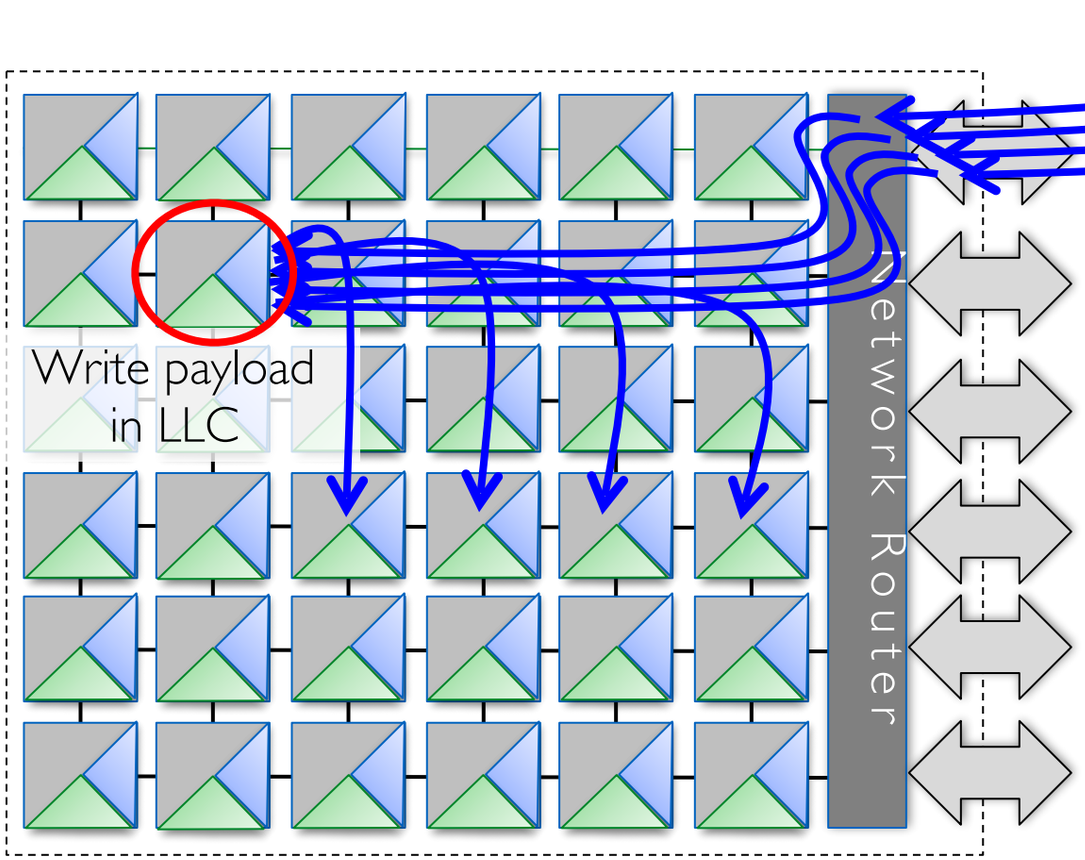


Data handling

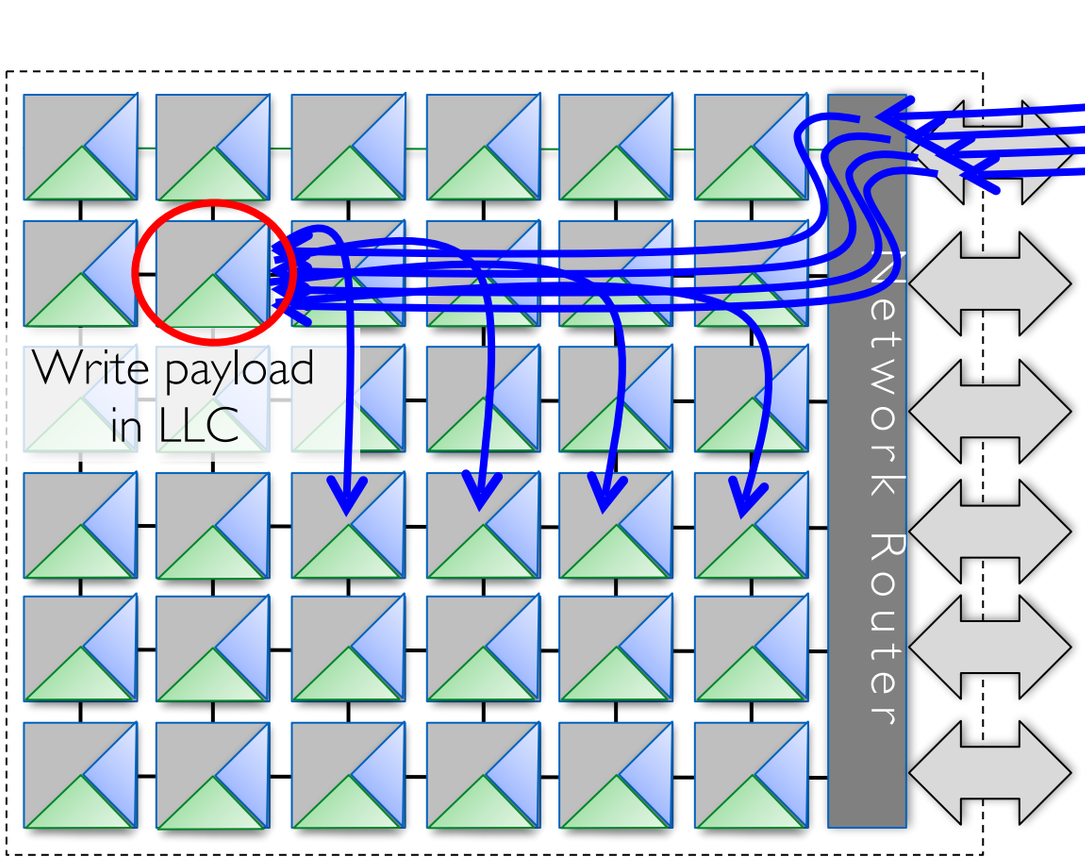
Per-Core NI 4-Cache-Block Remote Read



Per-Core NI 4-Cache-Block Remote Read



Per-Core NI 4-Cache-Block Remote Read



Incoming
replies

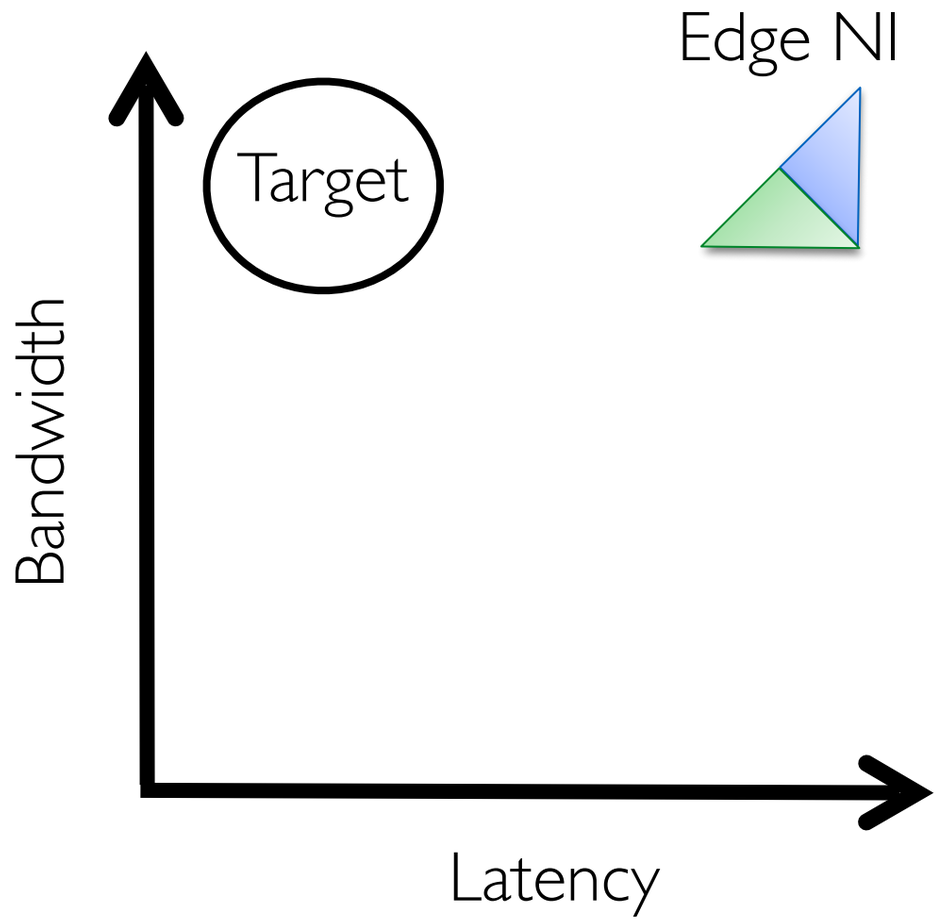
Data handling

All reply packets
received!
Write CQ *locally*

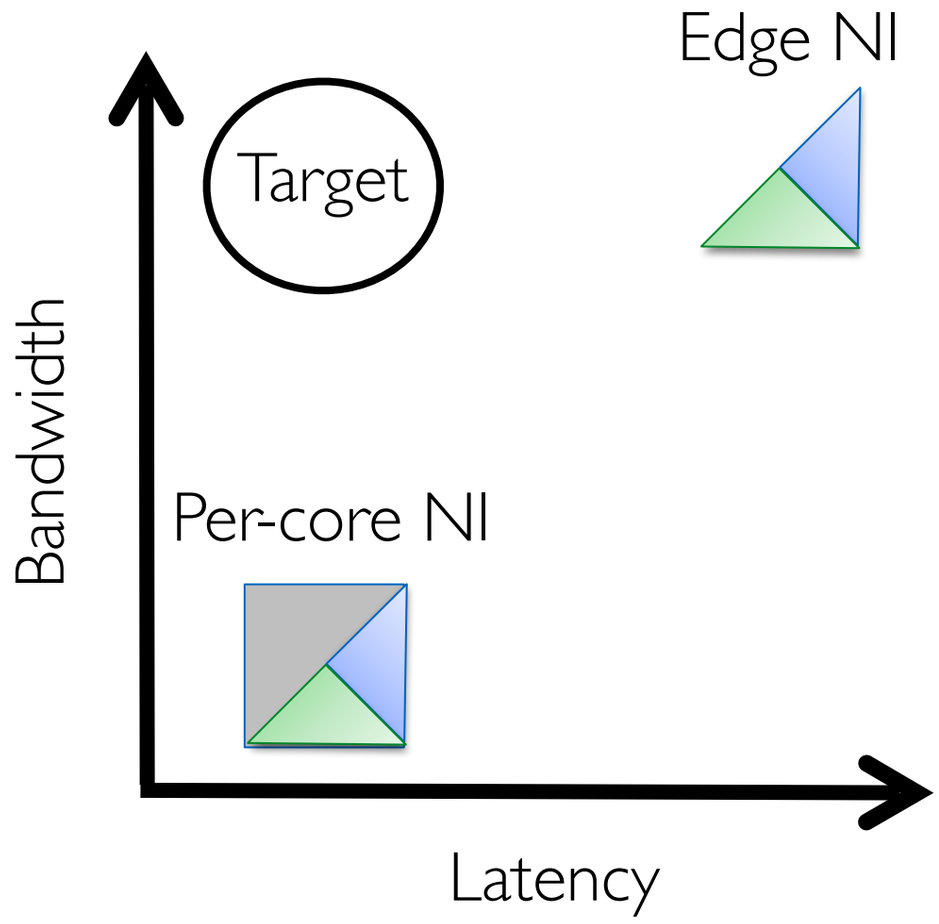
Bandwidth ✗
Latency ✓

Minimized latency but bandwidth misuse for large requests

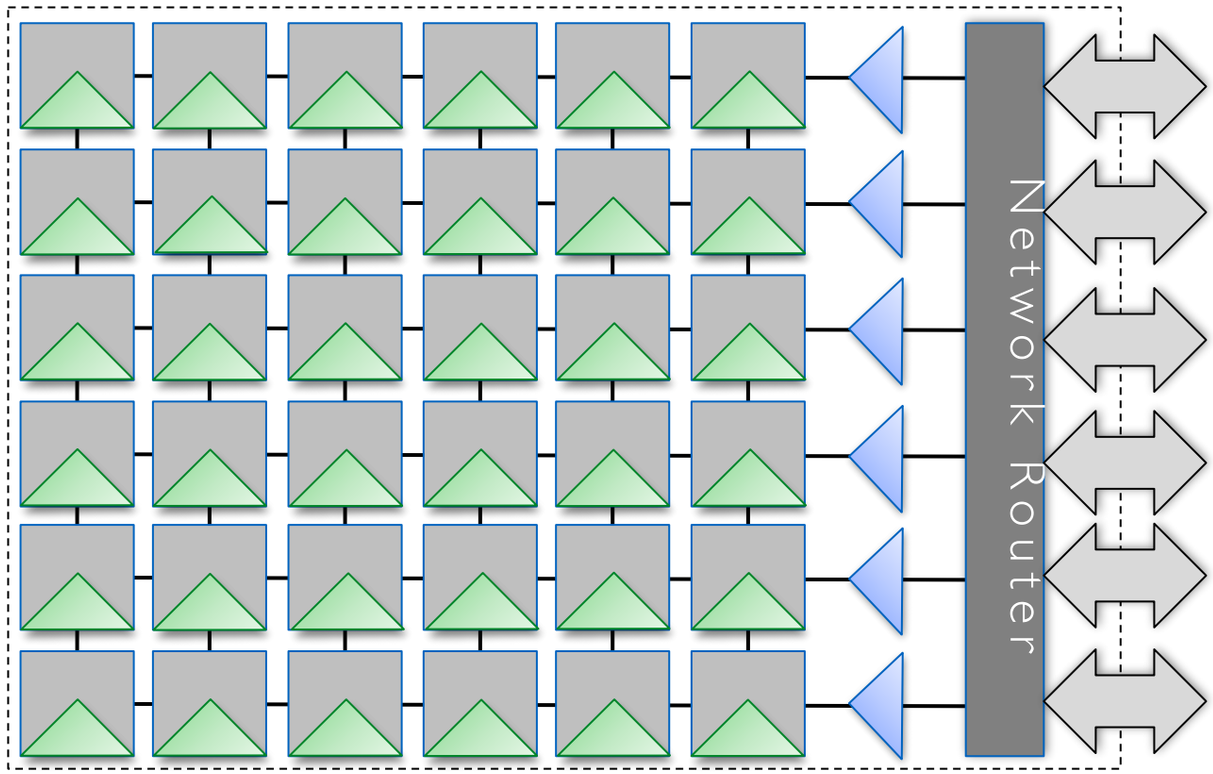
NI Design Space



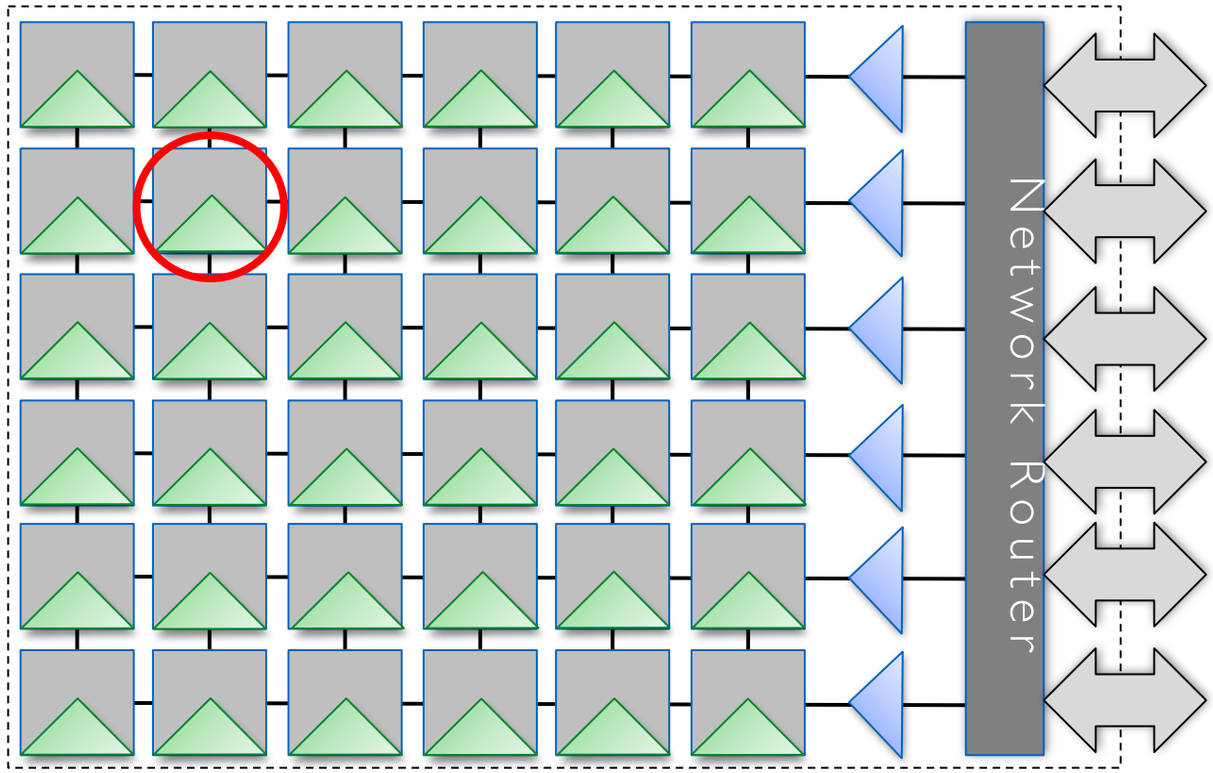
NI Design Space



Split NI 4-Cache-Block Remote Read

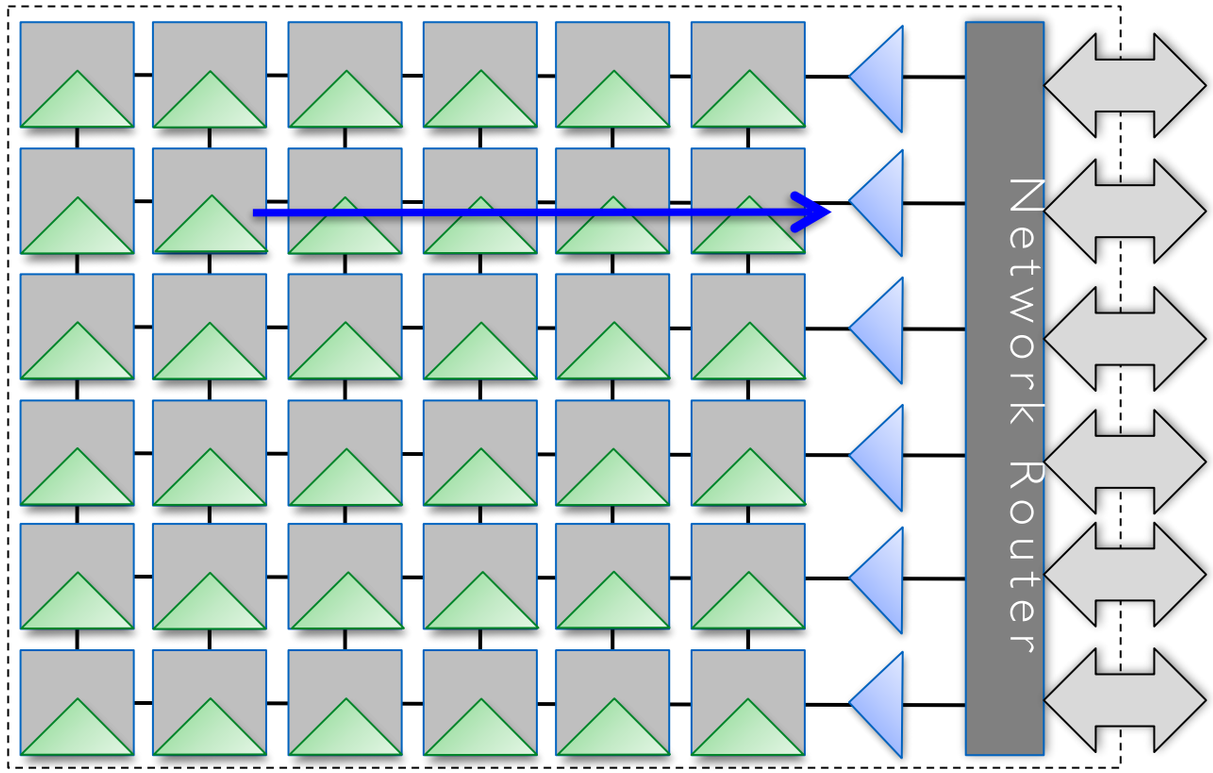


Split NI 4-Cache-Block Remote Read



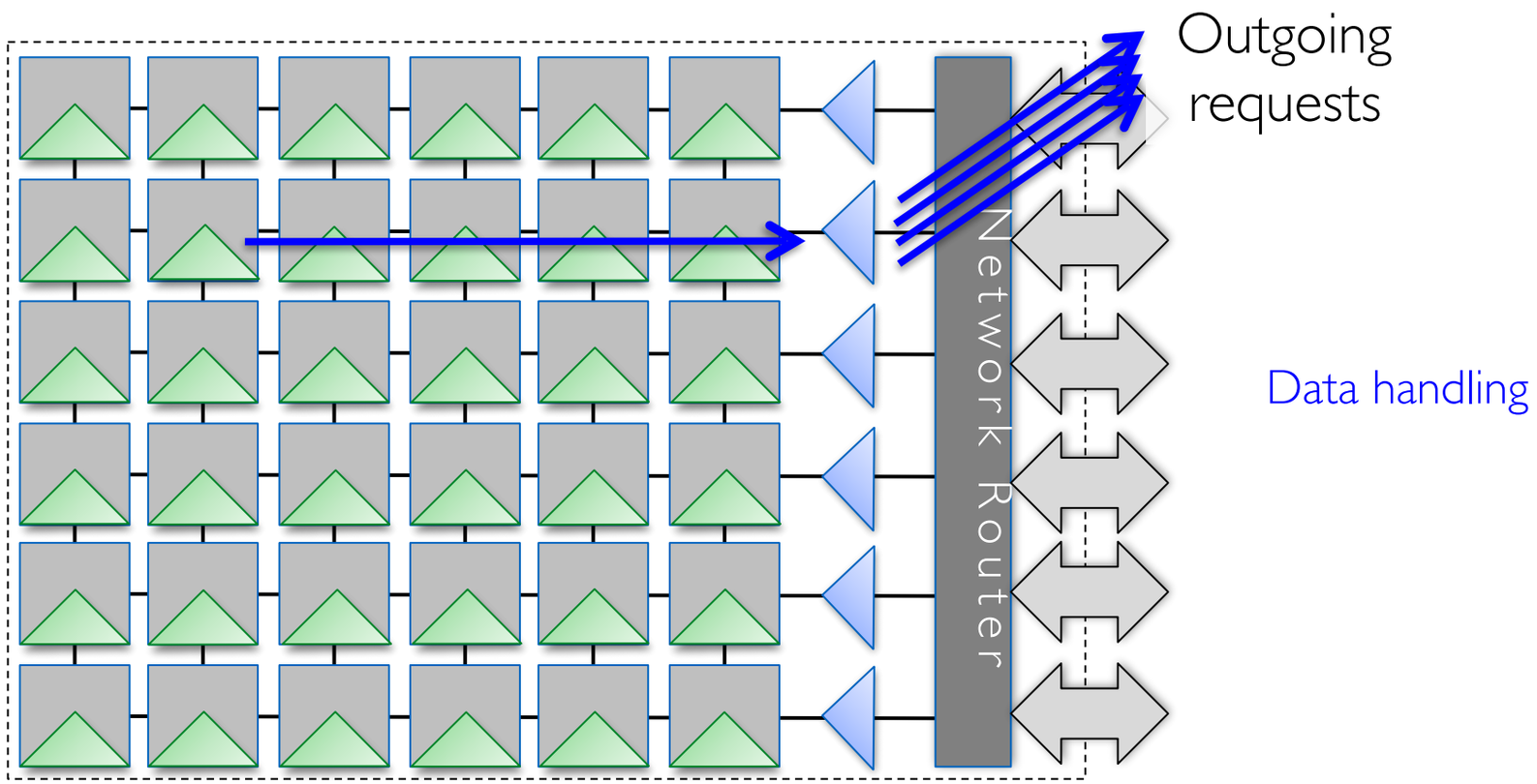
All QP interactions local!

Split NI 4-Cache-Block Remote Read

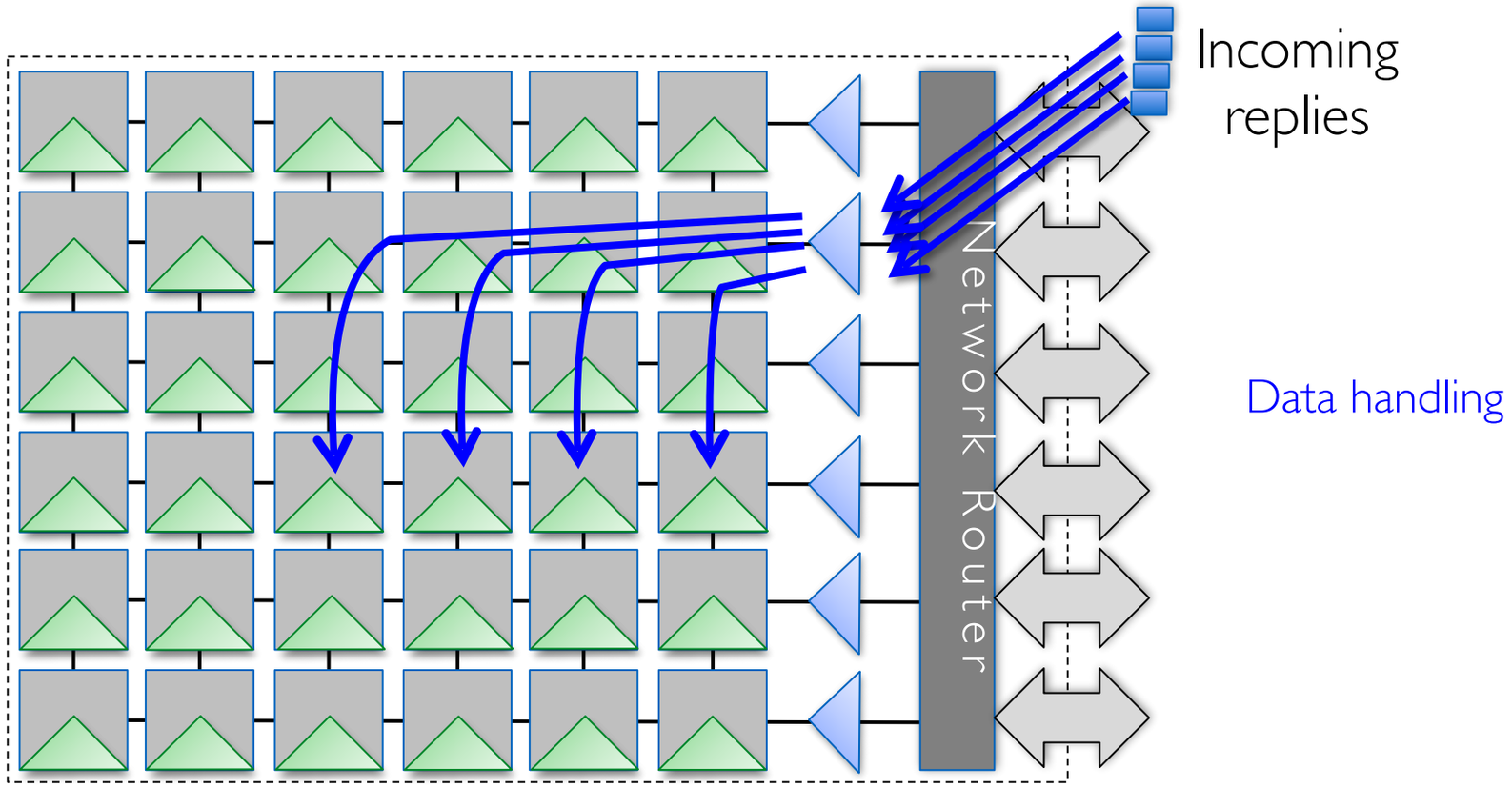


Data handling

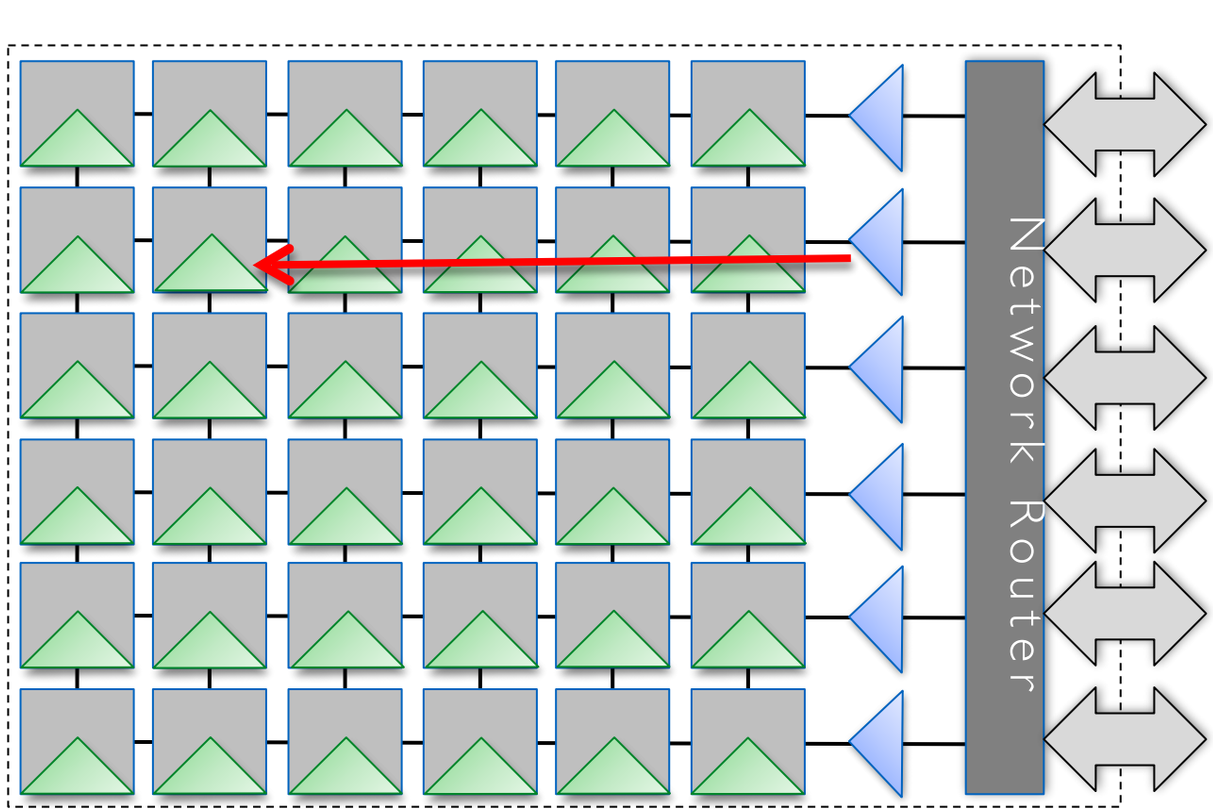
Split NI 4-Cache-Block Remote Read



Split NI 4-Cache-Block Remote Read



Split NI 4-Cache-Block Remote Read

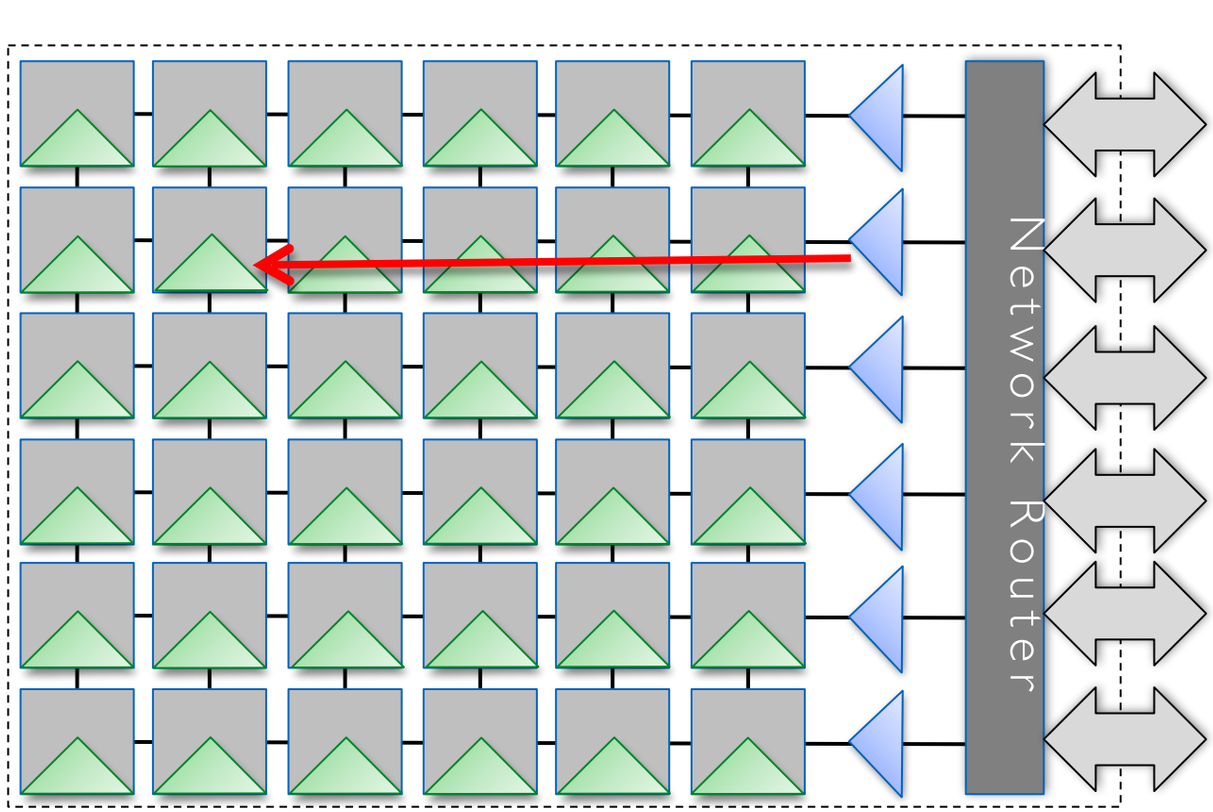


Incoming replies

All reply packets received!

Write CQ *locally*

Split NI 4-Cache-Block Remote Read



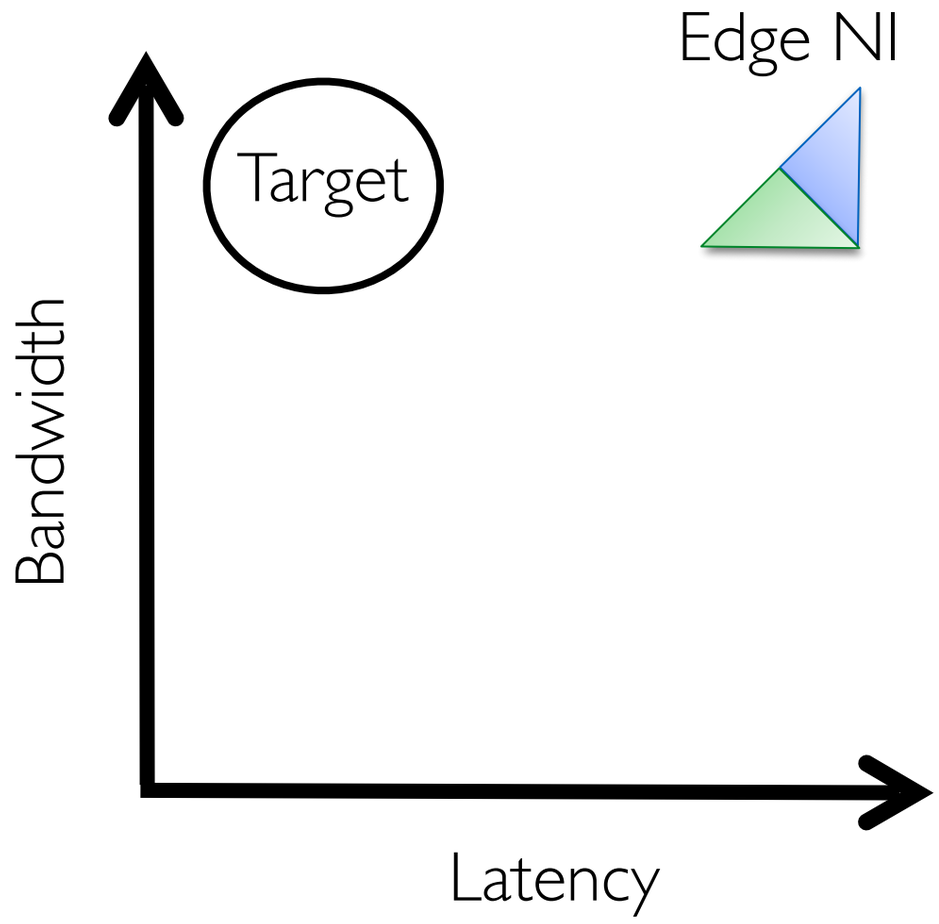
Incoming replies

All reply packets received!
Write CQ *locally*

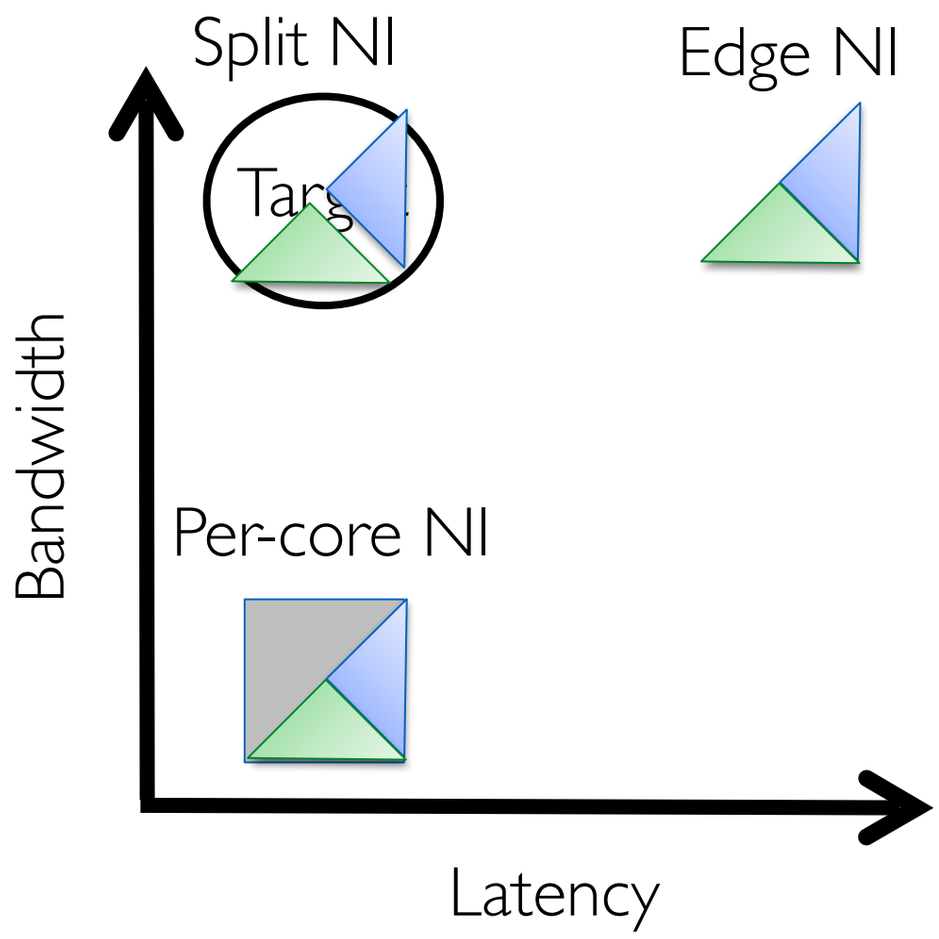
Bandwidth ✓
Latency ✓

Both low-latency small transfers and high-bandwidth large transfers

NI Design Space



NI Design Space



Outline

- Overview
- Current Technologies
- Challenges
- Conclusion

Conclusion

- Software construct for object atomicity incur heavy overheads.
- Need for hardware solutions to atomic object reads.
- NI design for manycore chips is crucial to remote memory access.
- Fast core-NI interaction critical to achieve low latency
- Large transfers best handled at the edge
- Split NI: low-latency, high-bandwidth remote memory access

Reference

- Alexandros Daglis, Stanko Novaković, Edouard Bugnion, Babak Falsafi, and Boris Grot. 2015. Manycore network interfaces for in-memory rack-scale computing. *SIGARCH Comput. Archit. News* 43, 3 (June 2015)

- Daglis, A, Ustiugov, D, Novakovic, S, Bugnion, E, Falsafi, B & Grot, B 2016, SABRes: Atomic Object Reads for In-Memory Rack-Scale Computing. in *In Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2016)*

- https://parsa.epfl.ch/~falsafi/talks/specialized-server.pdf