Distributed Systems

Spring Semester 2020

Lecture 12: FaRM

John Liagouris
liagos@bu.edu

Why this paper

- 2015 — Data Center Scale Distributed Systems in light of Modern Hardware
- An outstanding achievement
  - 90 million Replicated, Persistent, Transactions per second! (TATP)
    - 1 million transactions/second per machine
    - ~10,000 tweets per second, ~3 million emails/s

Is this hard? Think of what we have seen to date.

How do they do it?

- Questions a very basic assumptions that the other papers have been predicated on
- The hardware! — Software carefully designed
  - EVERYTHING IN RAM
  - NO DISKS FOR PERSISTENCE
  - FASTER NETWORK
  - ELIMINATE PROCESSING OVERHEADS ON COMMUNICATON PATHS

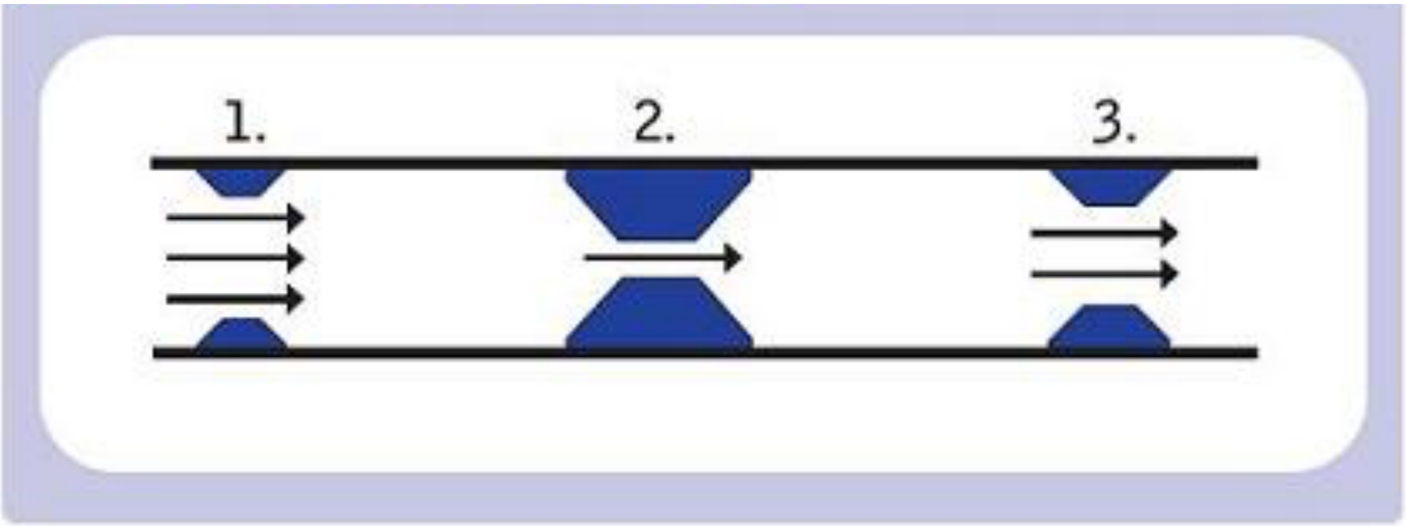
The HW Trends

128-512GB RAM/machine
<$12/GB —
2000 machines 1PetaByte

Lithium ION Batteries
and SSD’s — Cheap NVM

Datacenter networking which integrates
supercomputer interconnect capabilities
2x56Gbps NICS full bisectional bandwidth switch
— cpu-less communication — RDMA

All aspects required



Must alleviate or remove from hot paths

- Massive Ram required to remove disk bottleneck
- NVM required to have durability again to remove disk bottleneck
- Faster technology required to remove network bottleneck
- Advanced features to avoid CPU bottleneck

Big Memories

FaRM writes go to RAM, not disk -- eliminates a huge bottleneck

Can write RAM in 200 ns, but takes 10 ms to write hard drive, 100 us for SSD

ns = nanosecond 10^{-9},ms = millisecond10^{-3}, us = microsecond 10^{-6}

But RAM loses content in power failure! not persistent by itself.

Why not just write to RAM of f machines, to tolerate f failures?

Non-Volatile RAM

Batteries power the machines for a while if main power fails

s/w knows when main power fails

s/w halts all transaction processing

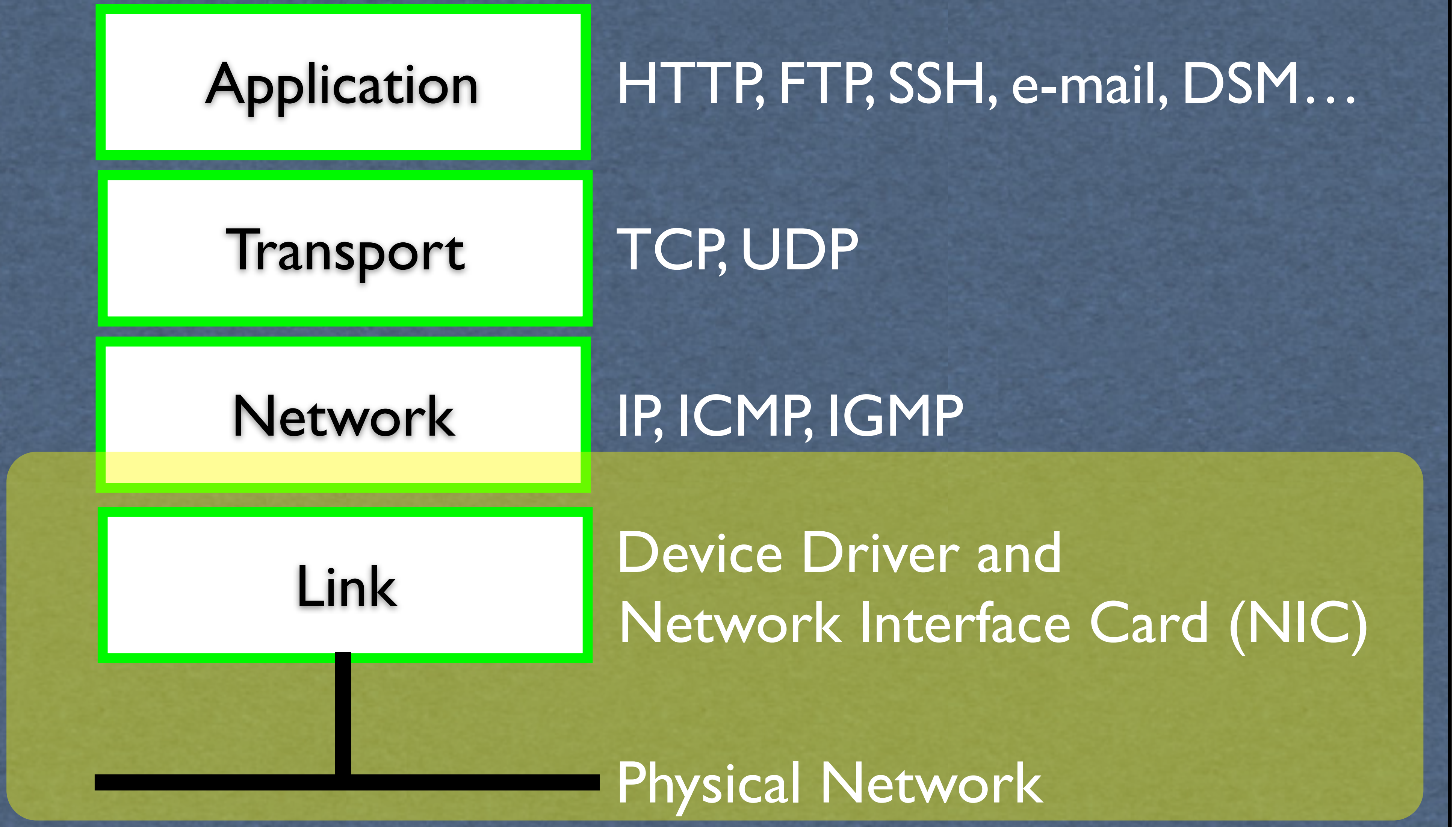
s/w writes FaRM's RAM to SSD; may take a few minutes then machine shuts down cleanly

On re-start, FaRM reads saved memory image from SSD

What if crash prevents s/w from writing SSD? e.g kernel bug, or cpu/memory/hardware error

Networking

The Layers of the TCP/IP protocol suite.



Application — HTTP, FTP, SSH, e-mail, DSM…

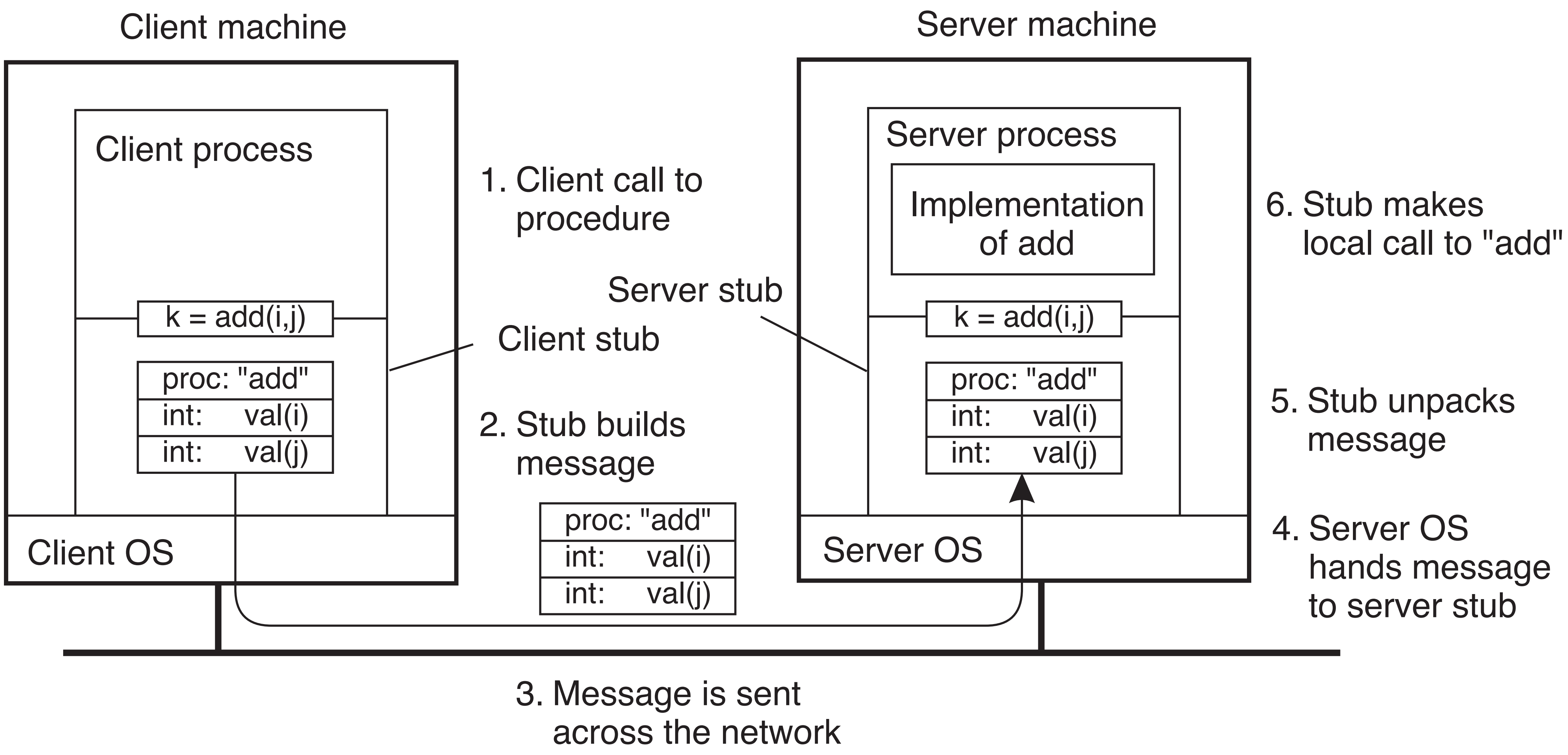
Transport — TCP, UDP

Network — IP, ICMP, IGMP

Link — Device Driver and Network Interface Card (NIC)

Physical Network

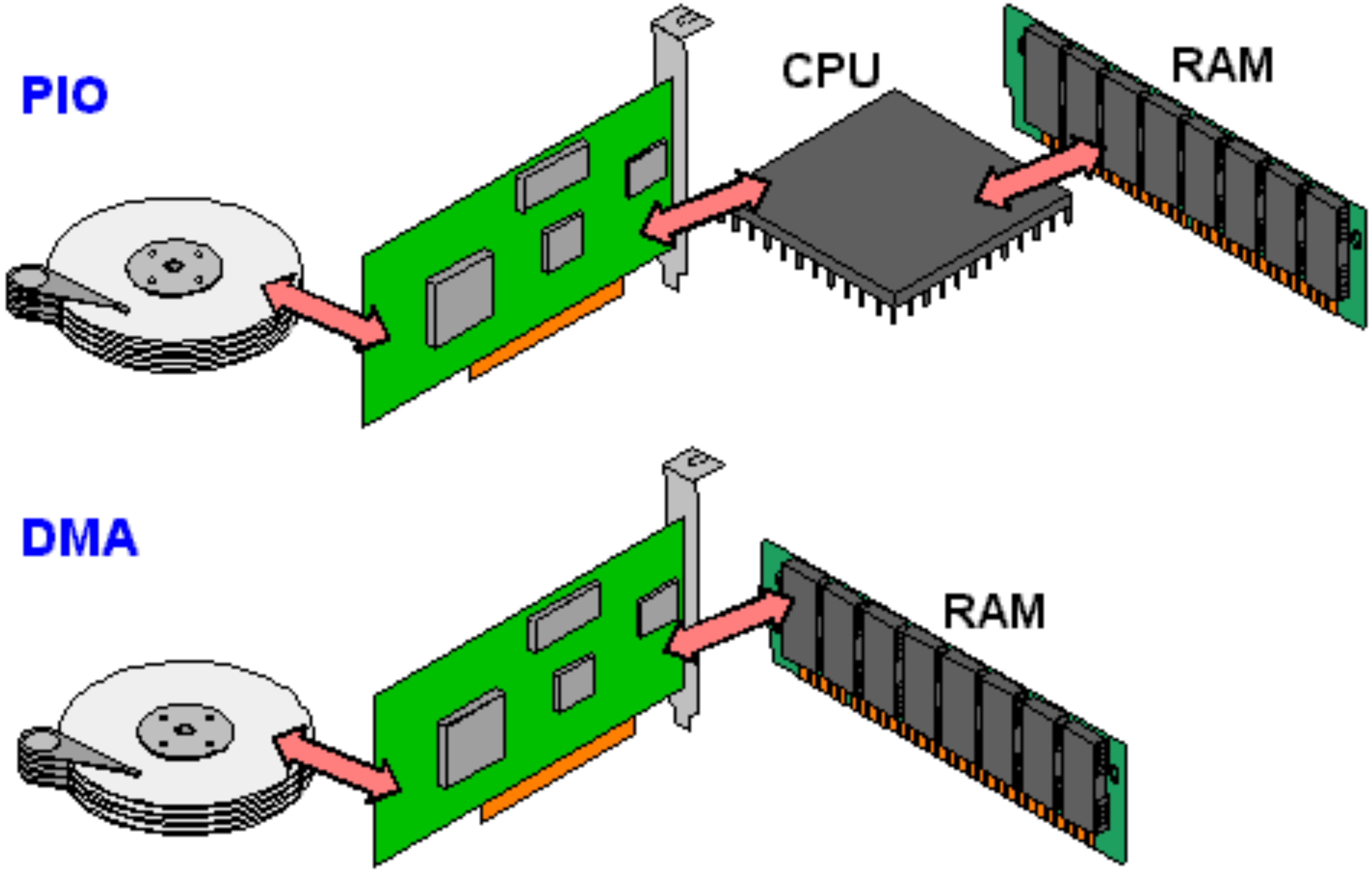
RPC: Under the covers



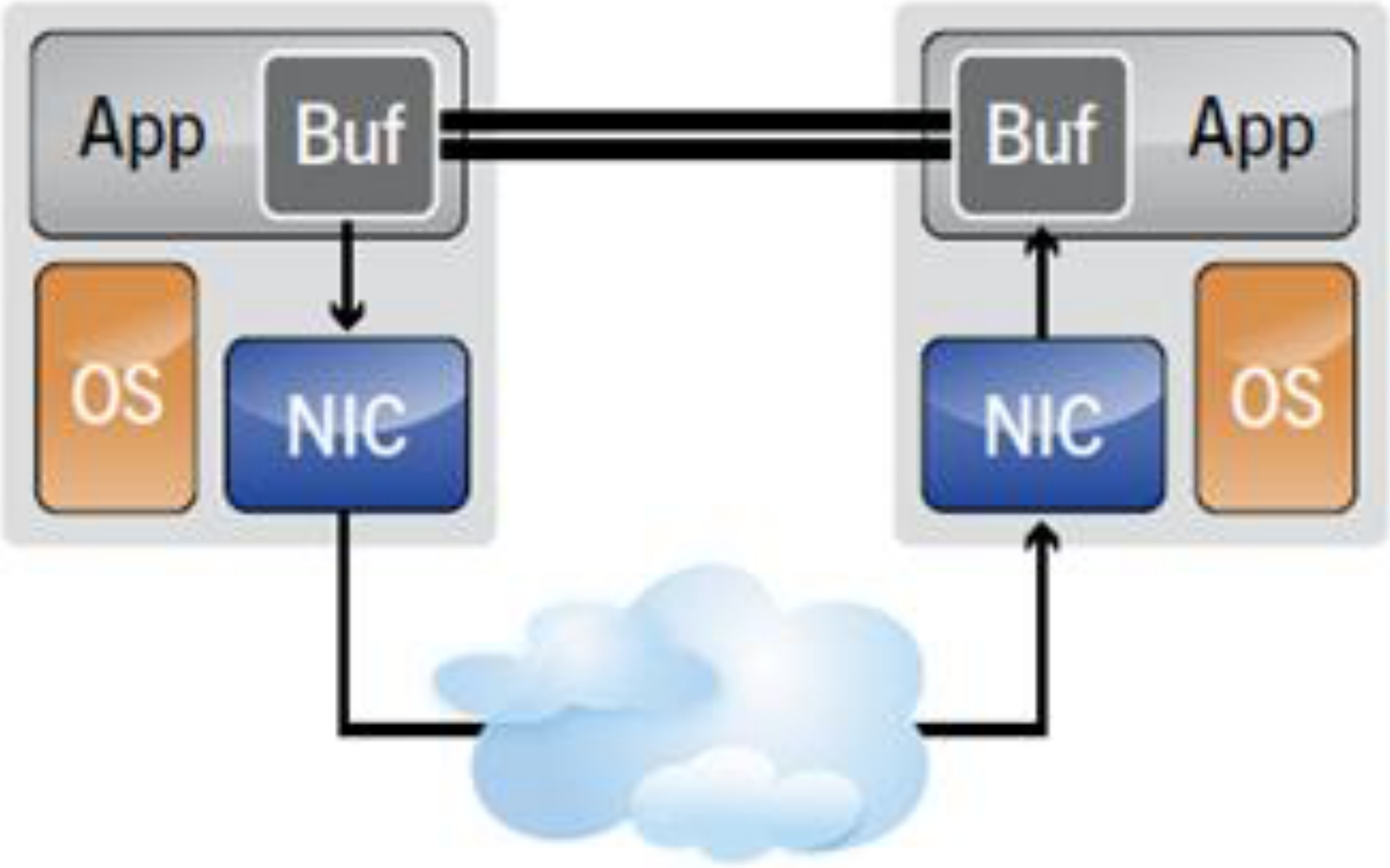
1. Client procedure calls client stub.
2. Stub builds message; calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters and calls server.
6. Server returns result to stub.
7. Stub builds message; calls OS.
8. OS sends message to client’s OS.
9. Client’s OS gives message to stub.
10. Client stub unpacks result and returns to the client.

What’s wrong with this story?

Memory Access



RDMA



One-Sided RDMA

NIC does "one-sided RDMA": memory read/write, not packet delivery

Sender says "write this data at this address", or "read this address"

NIC *hardware* executes at the far end

returns a "hardware acknowledgement"

FaRM

Distributed Transactions — Designed to exploit modern data center hardware

Distributed Transactions — single machine that executes operations on memory objects.

Optimistic Concurrency Control

Strict Serialisability

Fault-tolerance

Transaction

- Start transaction
- Read and Write memory objects
- Commit either succeeds or fails

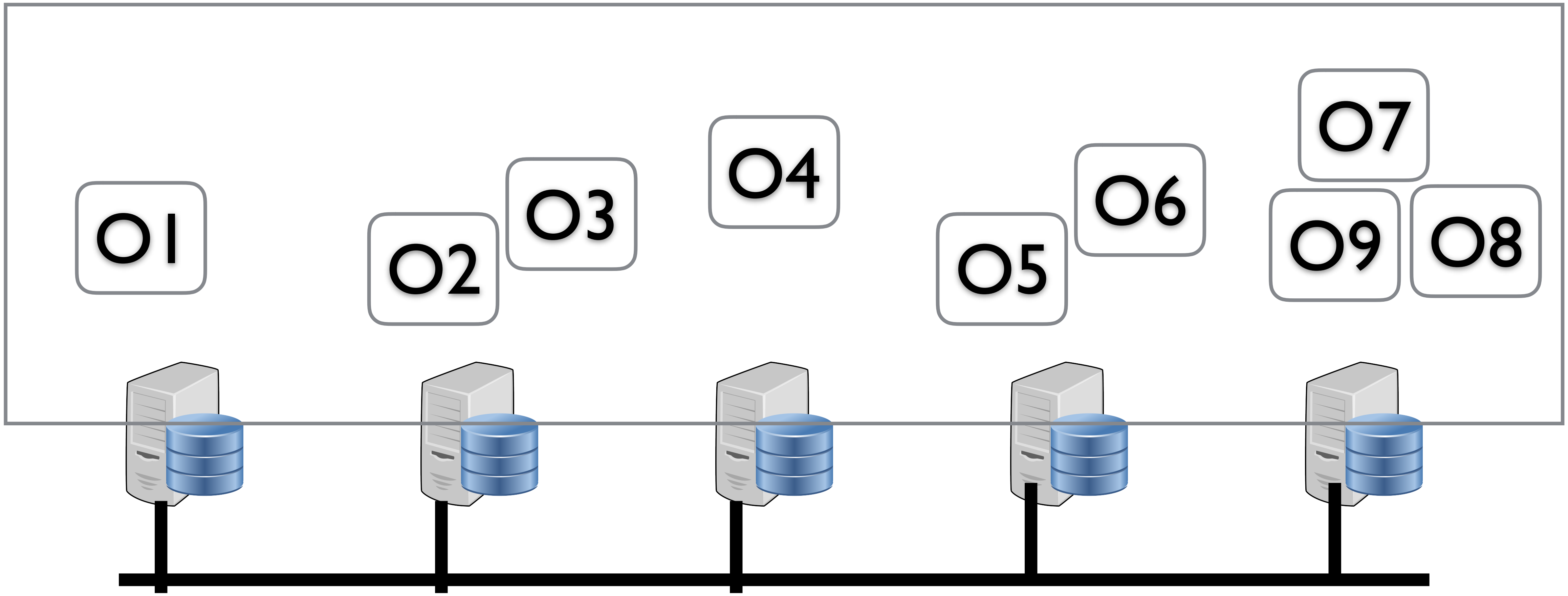
```
TxBegin()
   a = TxRead(oidA);
   b = TxRead(oidB);
   a.value -= amt;
   b.value += amt;
   TxWrite(oidA, a);
   TxWrite(oidB, b);
 TxCommit();
```

Real FaRM API more subtle — and Event Driven

3 Principles

- Use one-sided RDMA operations
- Reduce message counts
- Exploit parallelism — to increase availability

FaRM (NSDI '14)

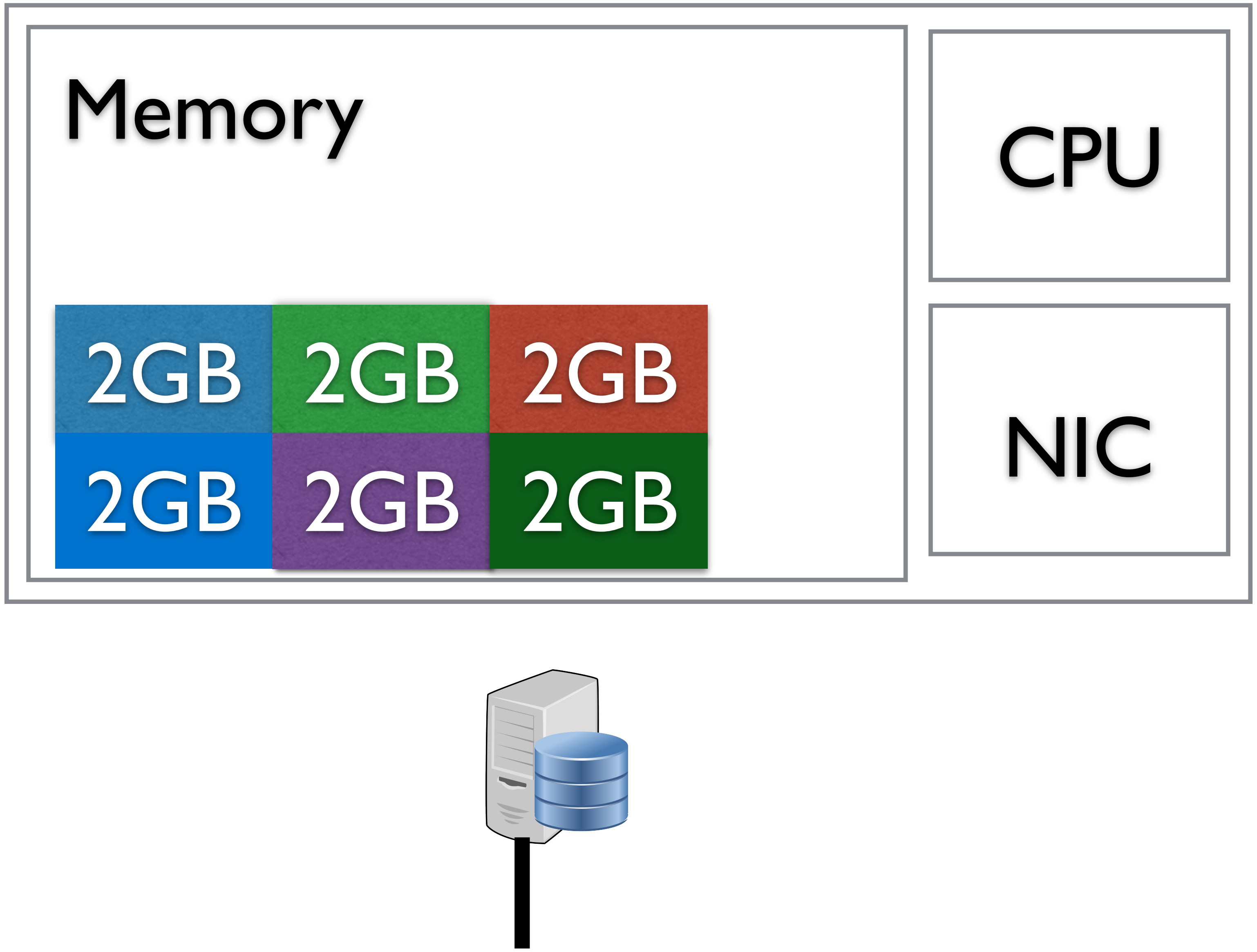


General Purpose Platform : KV Store, graph Store, OLTP databases, etc. — small objects, irregular access

All Machines clients and servers

Transactions: Read, Write, Alloc, Free, …

FaRM



Memory

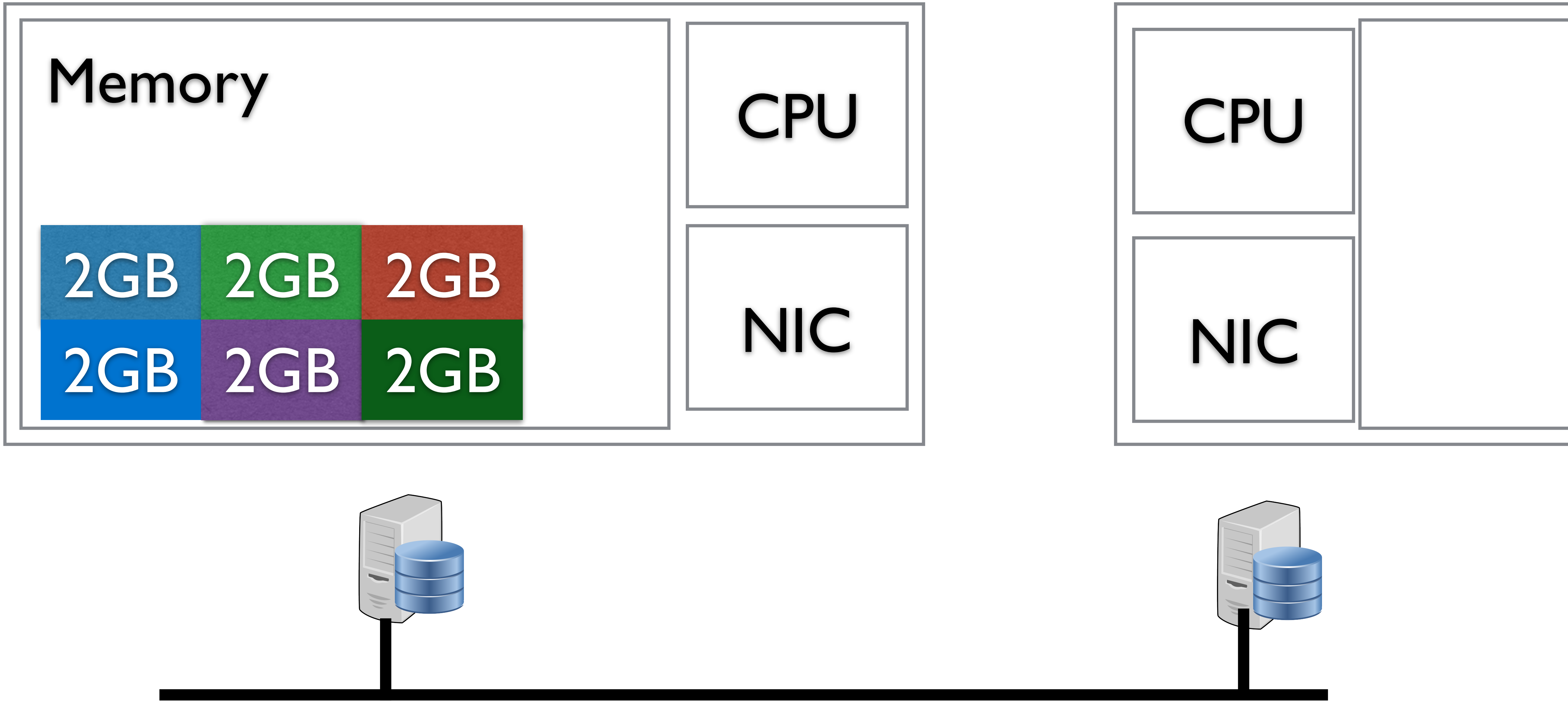
2GB 2GB 2GB

2GB 2GB 2GB

CPU

NIC

FaRM



Memory

2GB 2GB 2GB

2GB 2GB 2GB

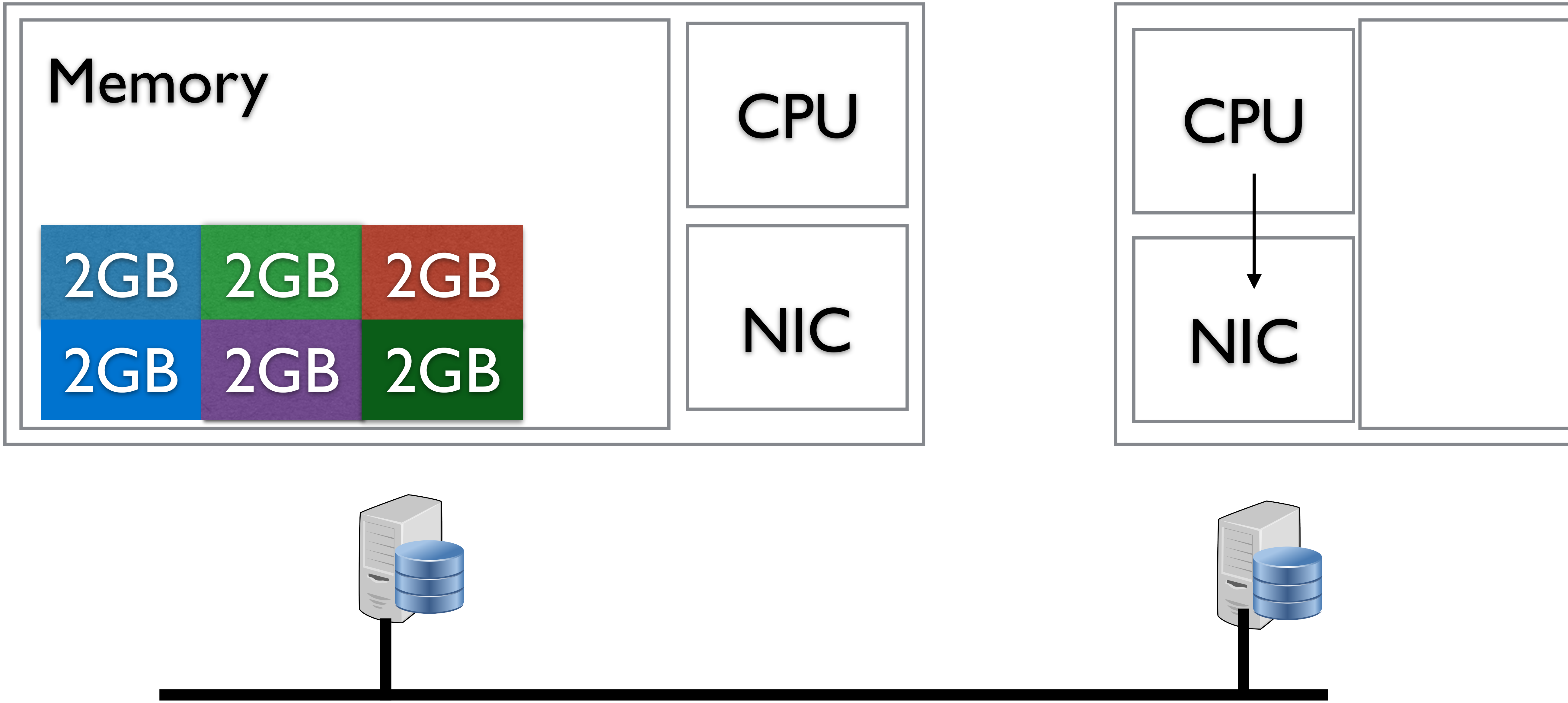
CPU

NIC

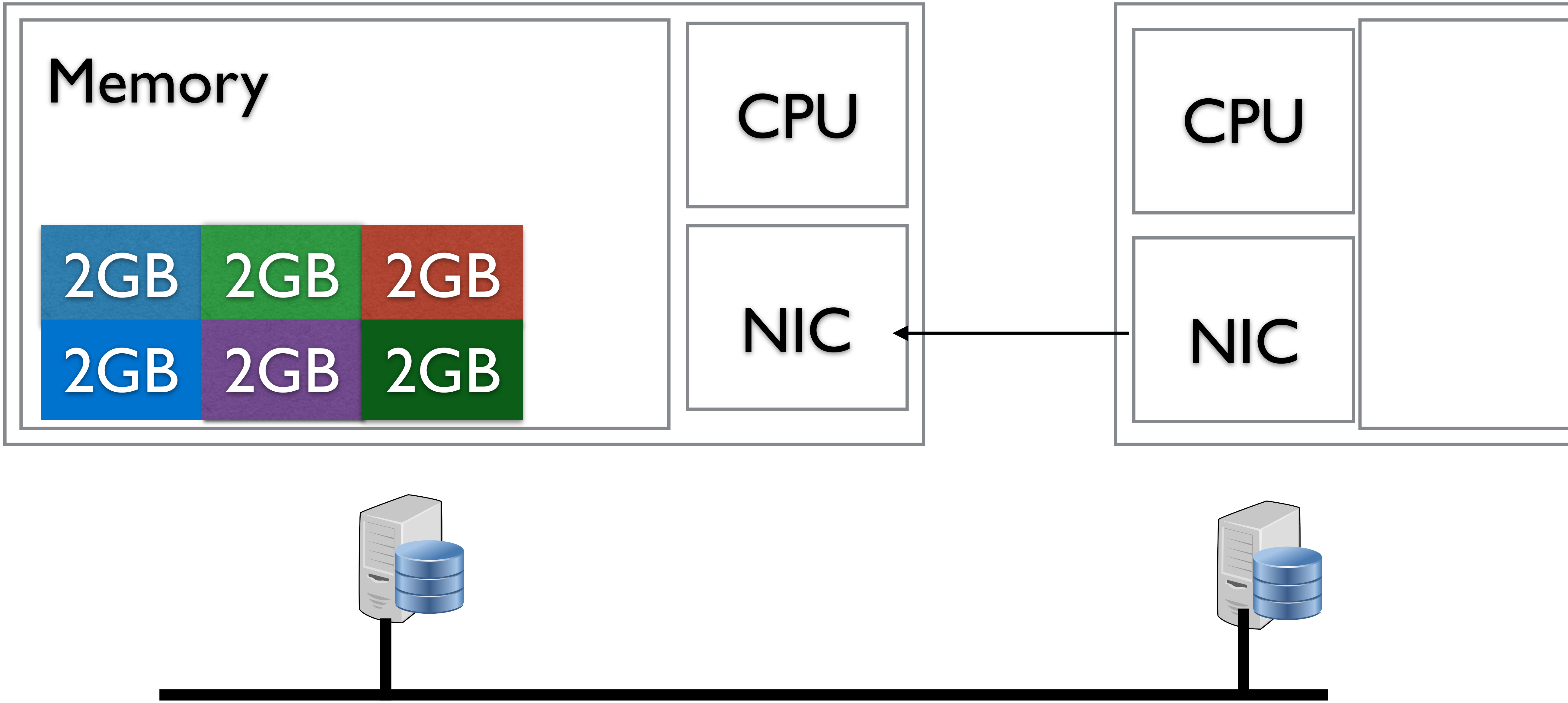
CPU

NIC

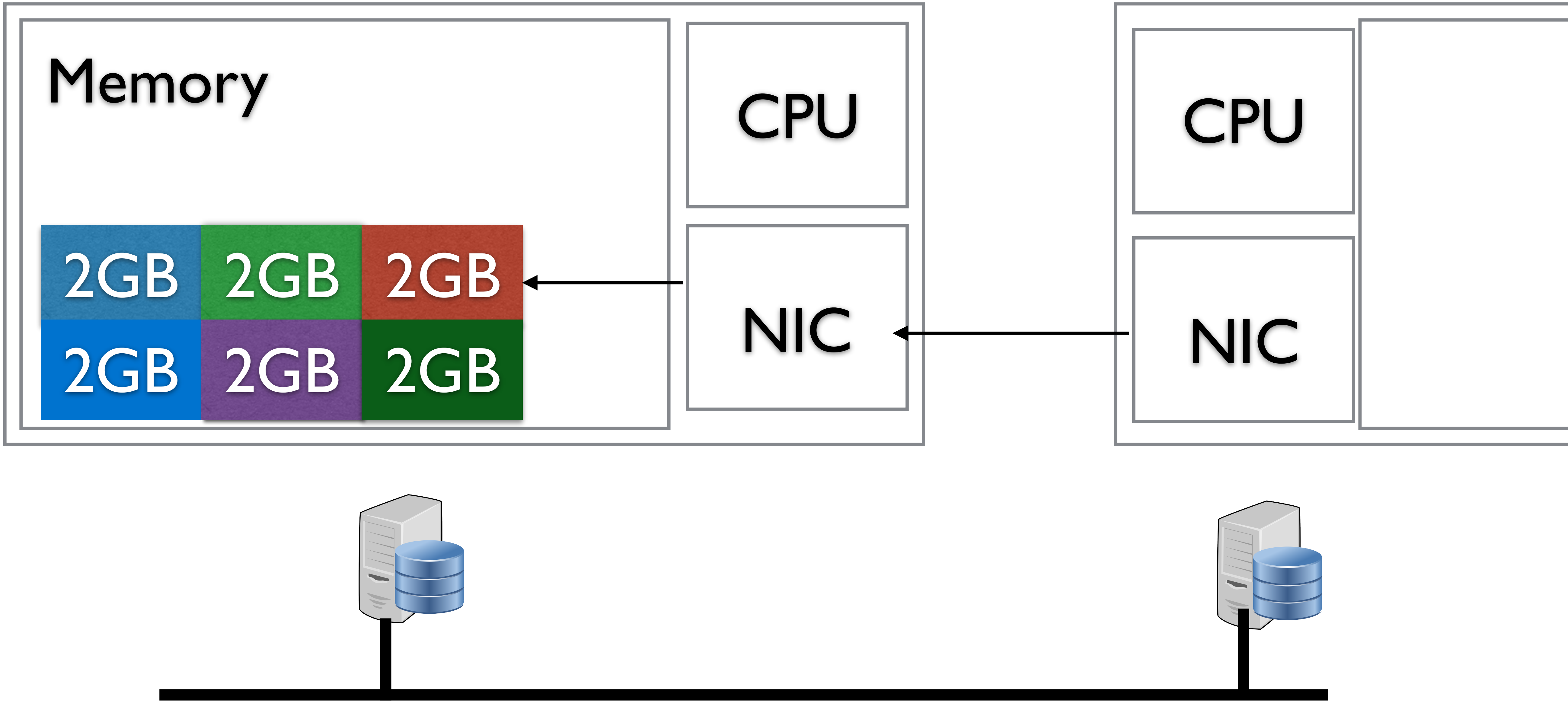
FaRM: txRead



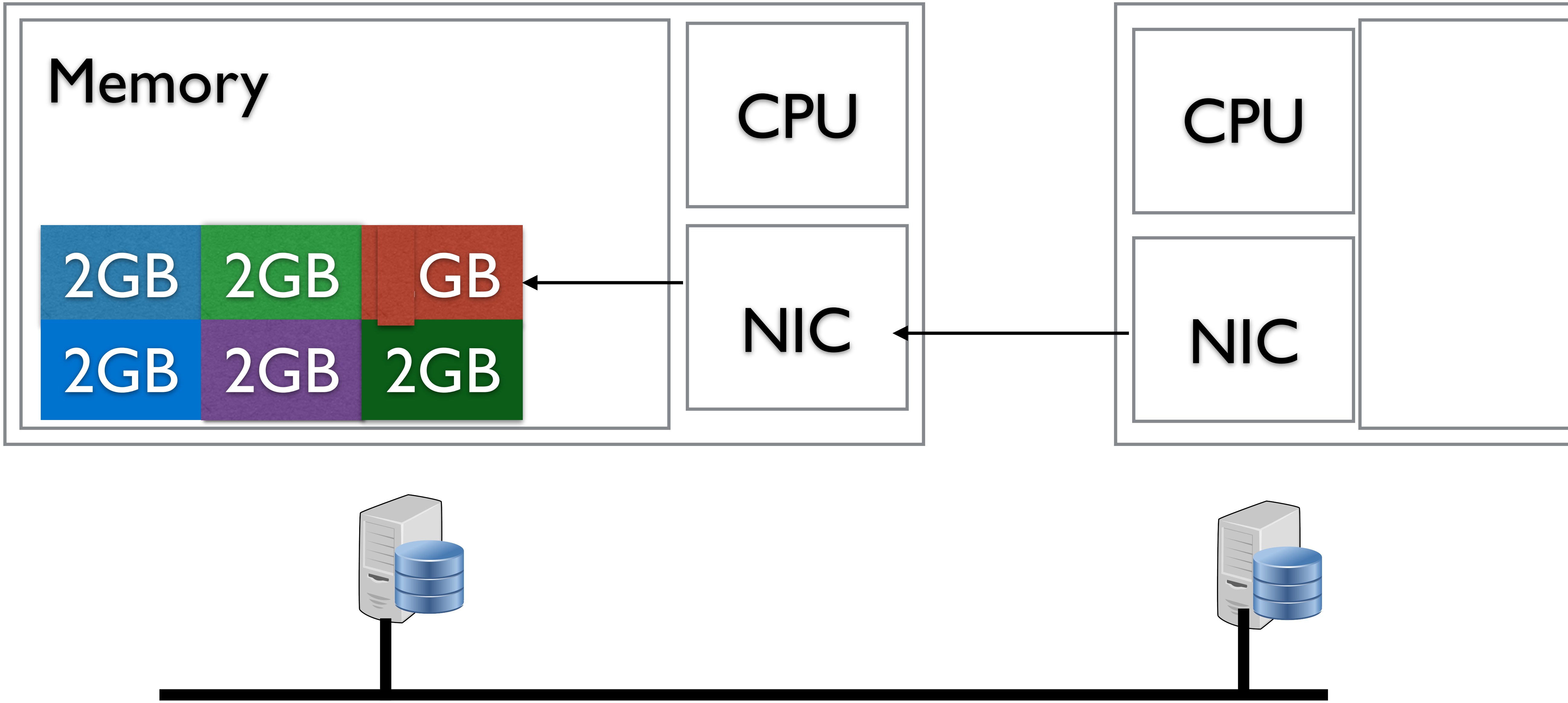
FaRM: txRead



FaRM: txRead

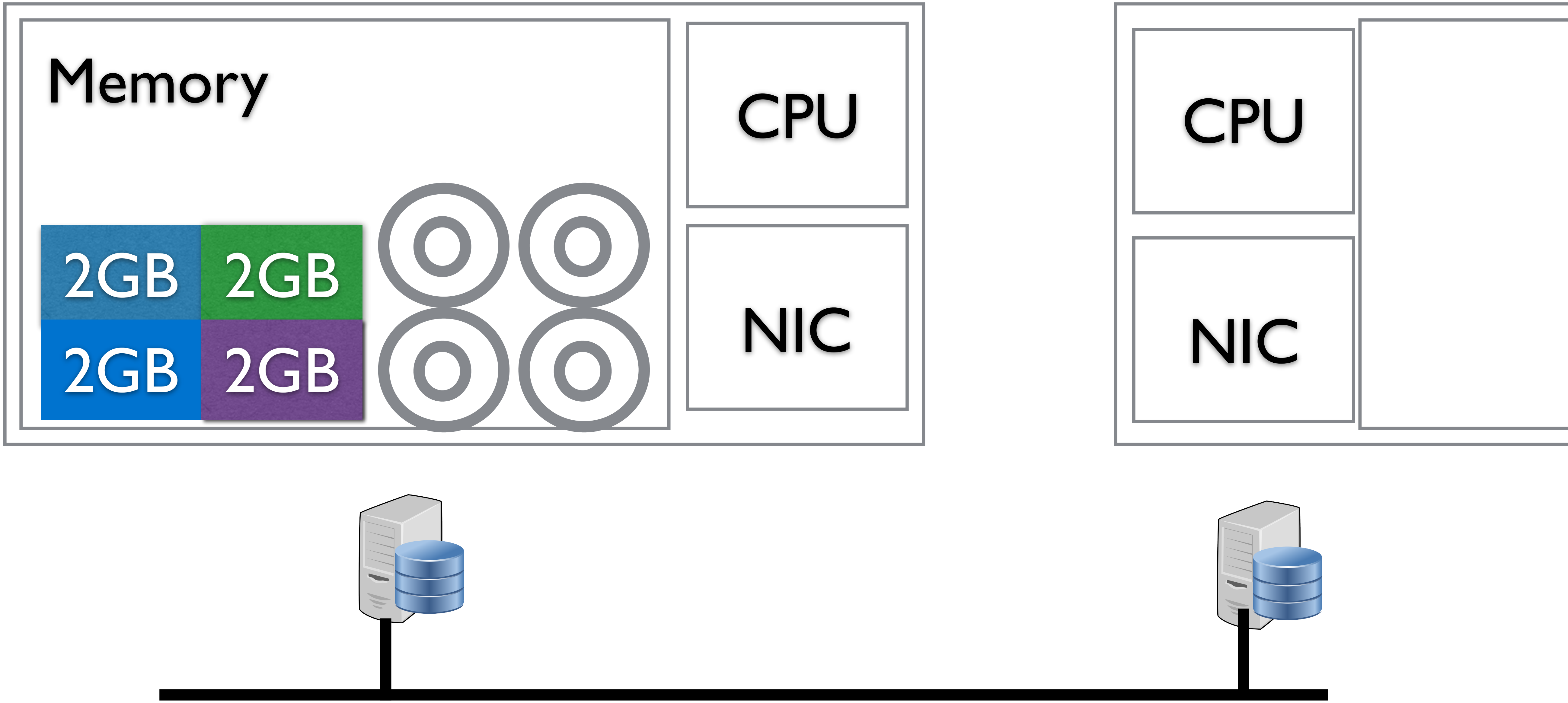


FaRM: txRead



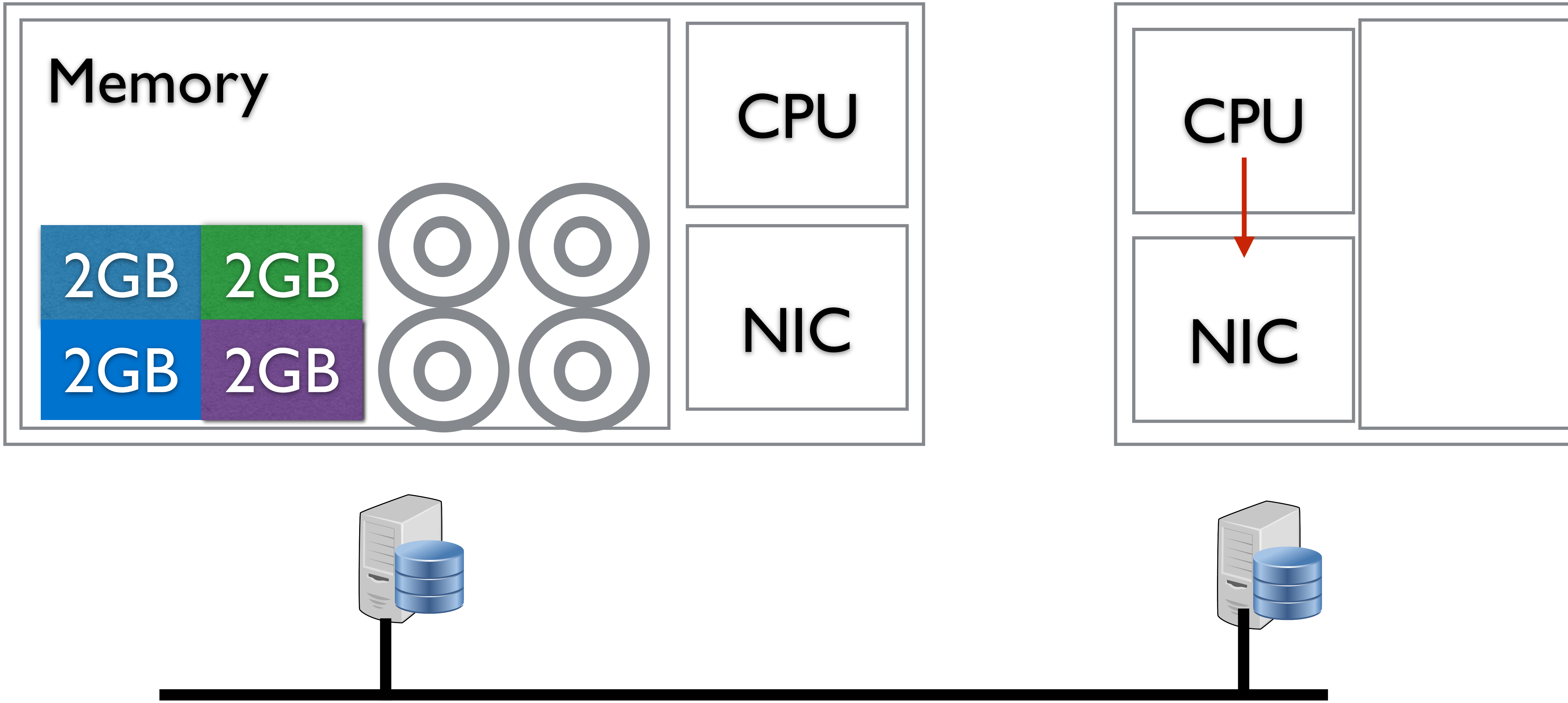
No CPU involvement at "Server"

FaRM: txWrite



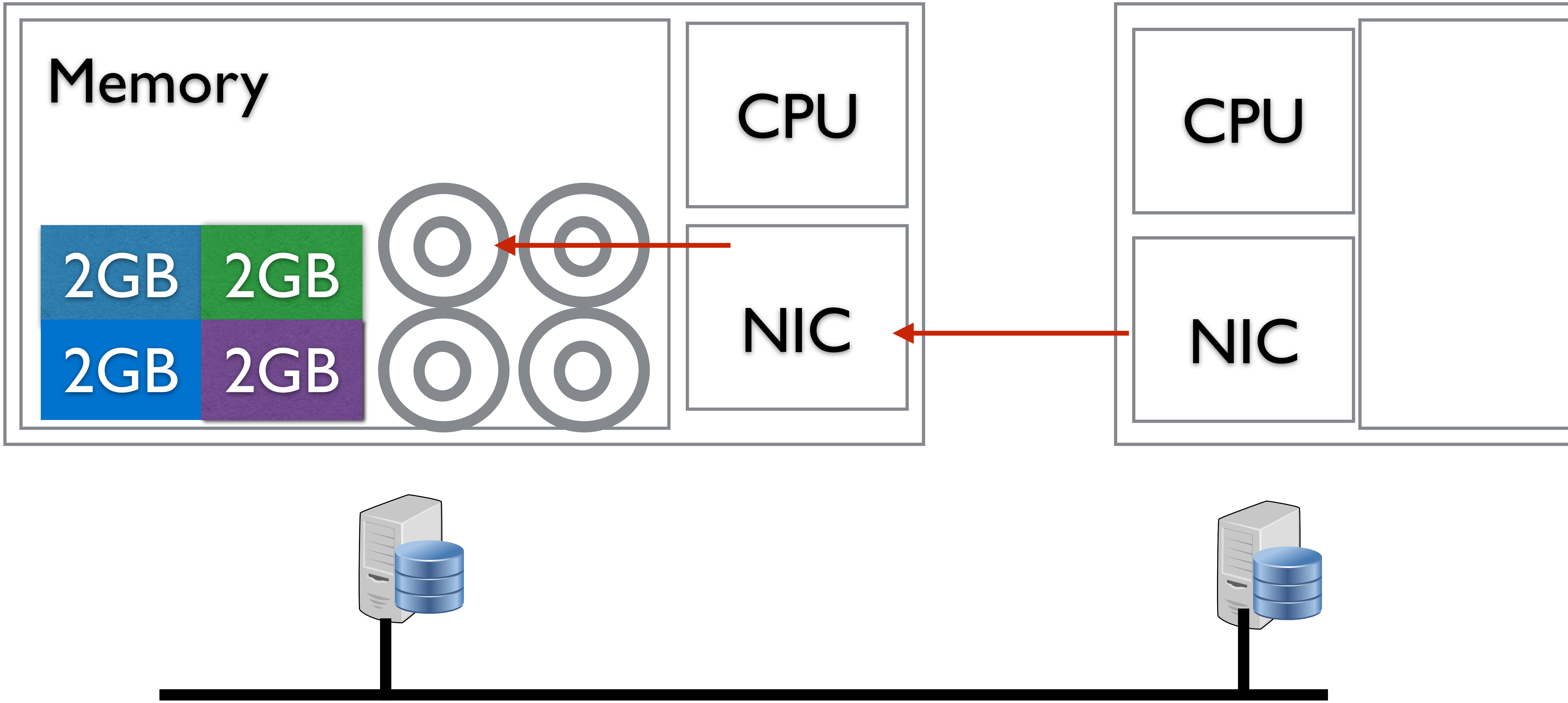
No CPU involvement at "Server"

FaRM: txWrite



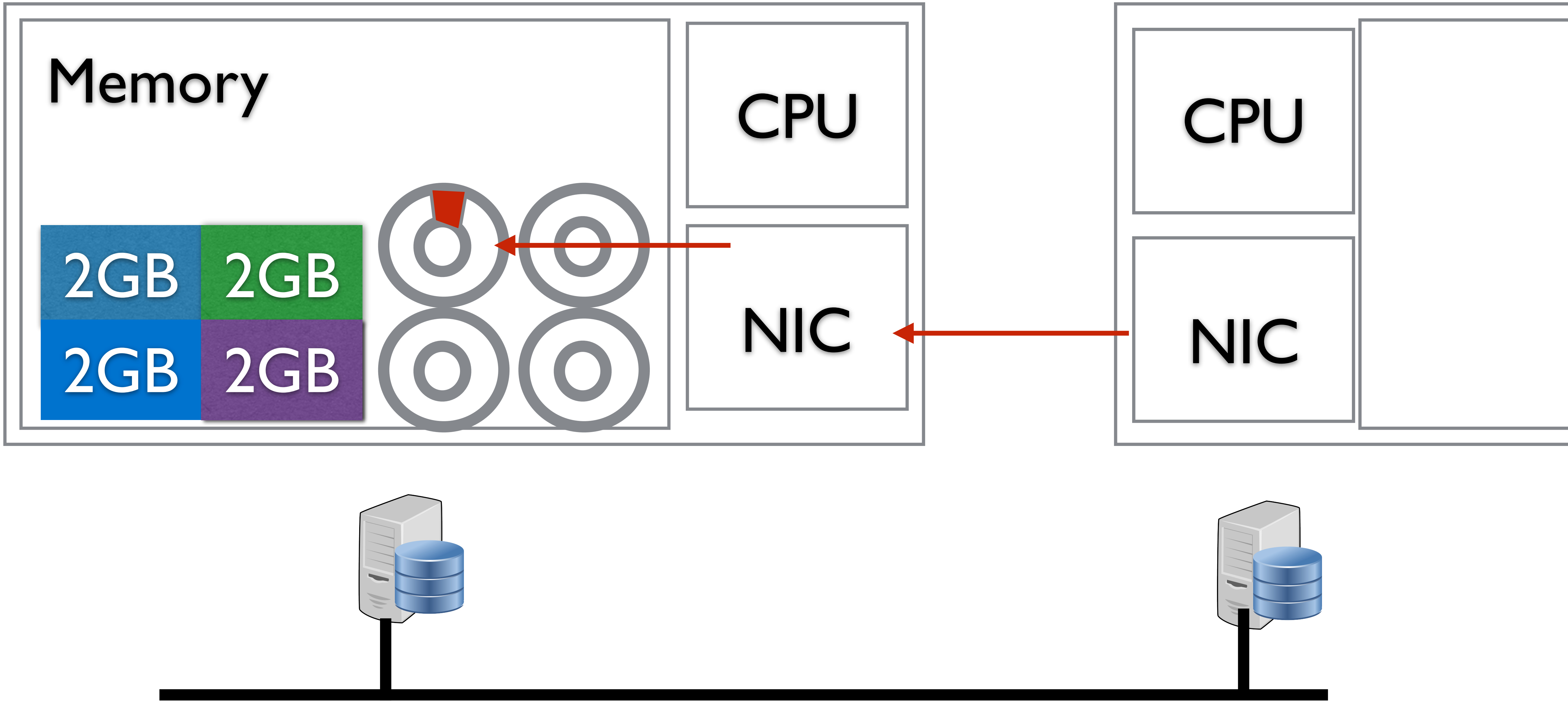
No CPU involvement at "Server"

FaRM: txWrite



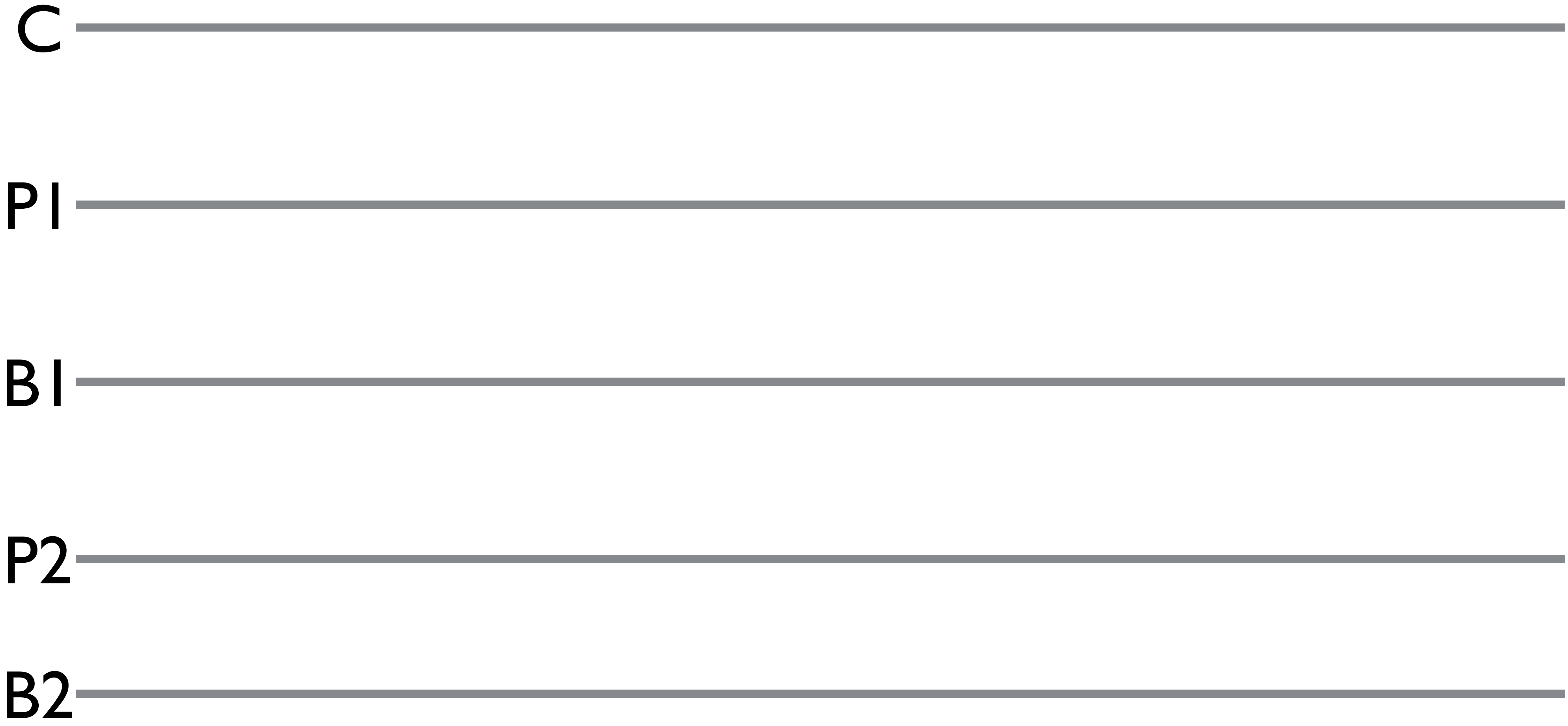
No CPU involvement at “Server”

FaRM: txWrite



No CPU involvement at “Server”

Transactions



C

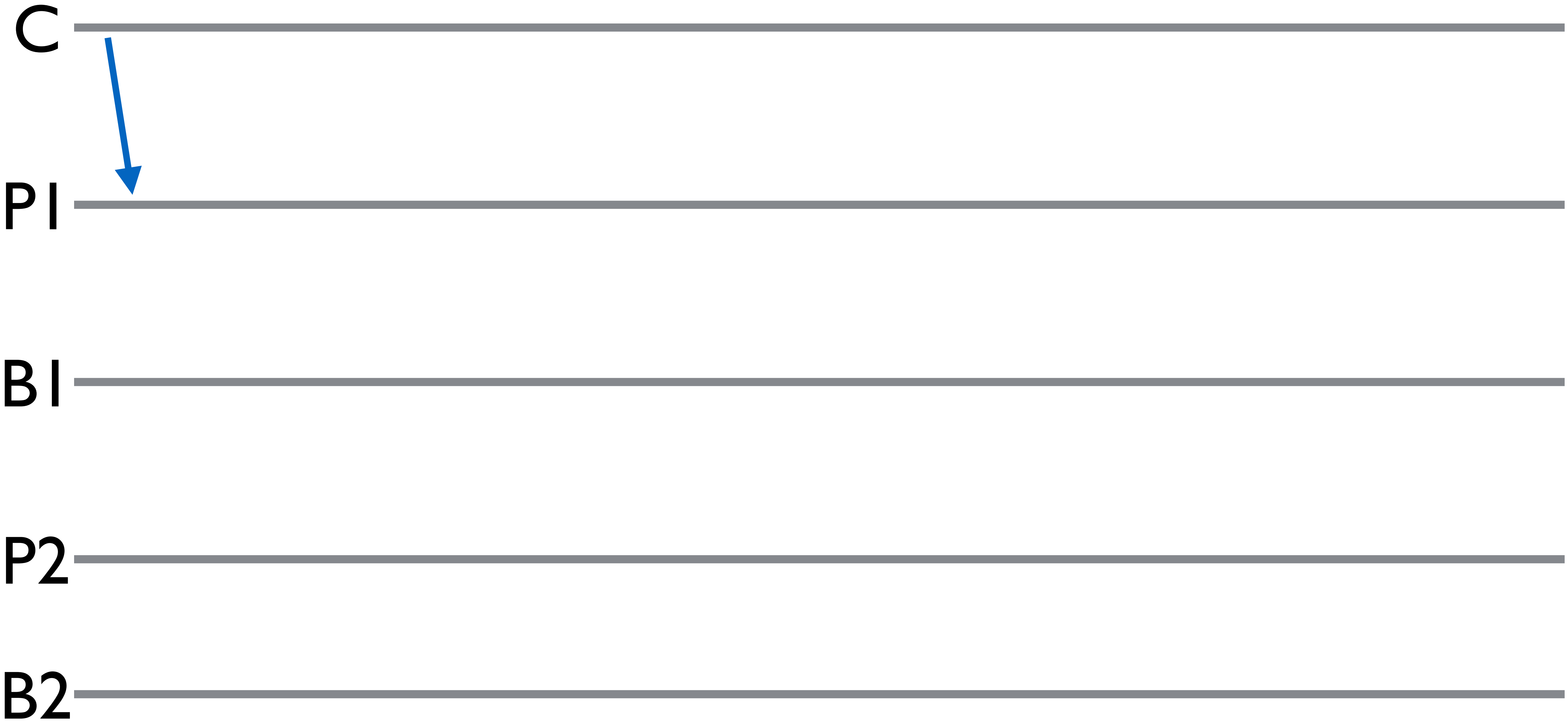
P1

B1

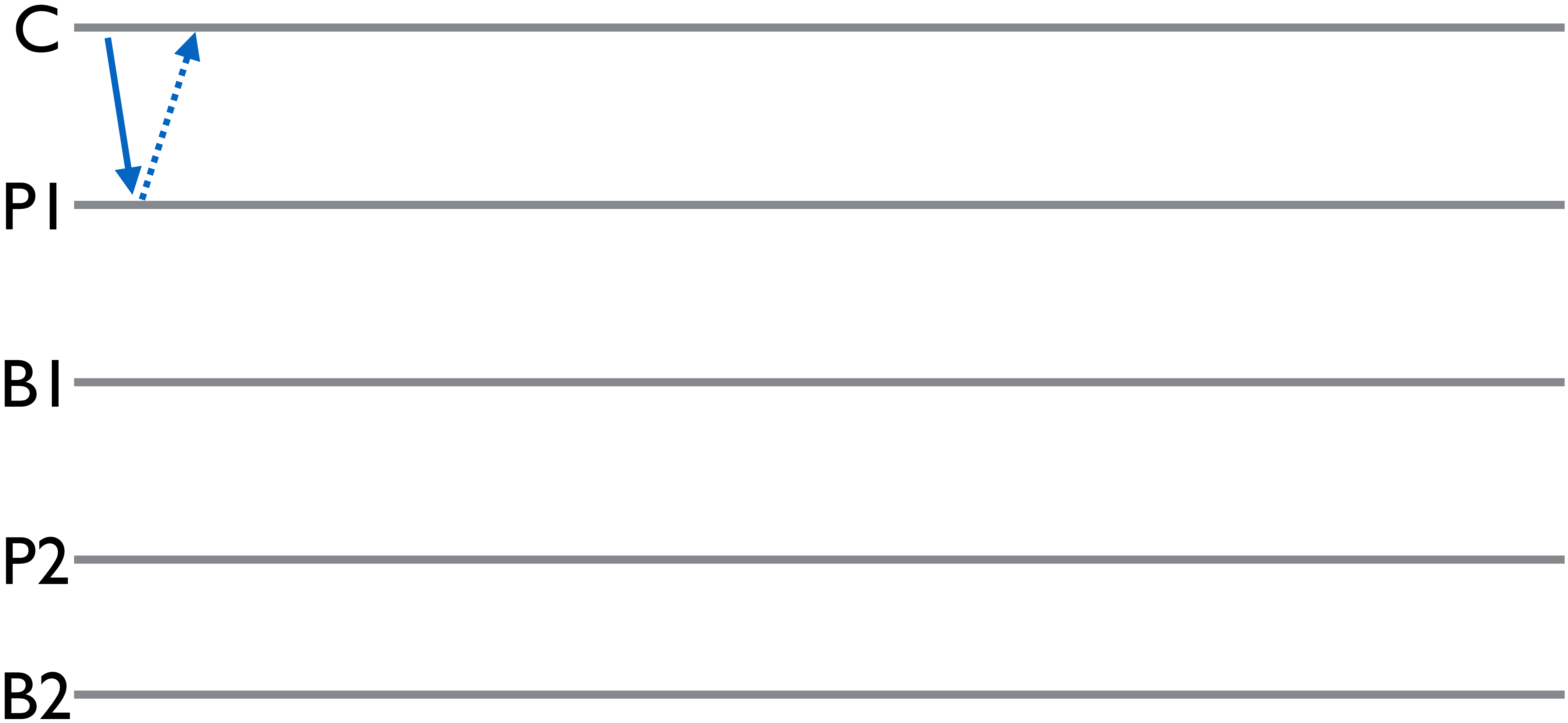
P2

B2

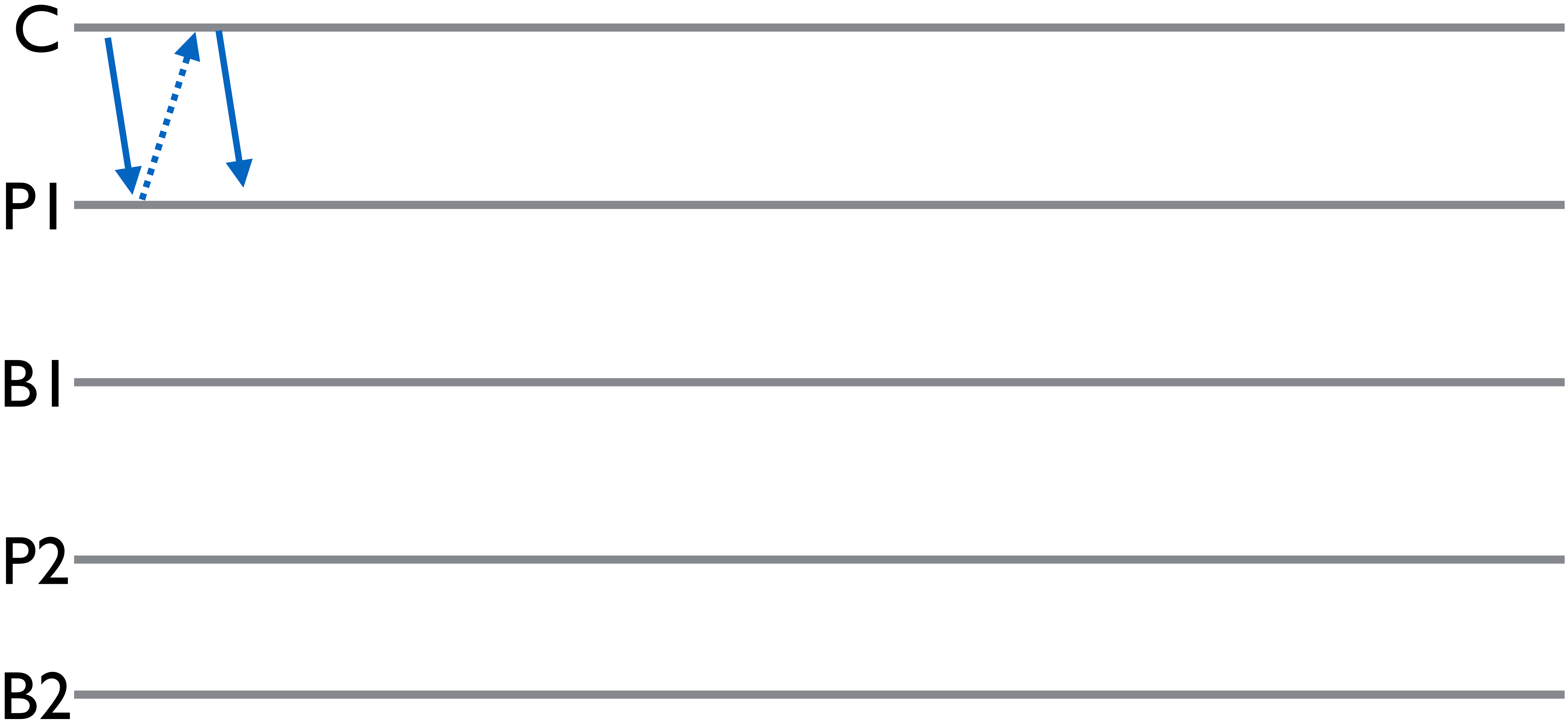
Transactions



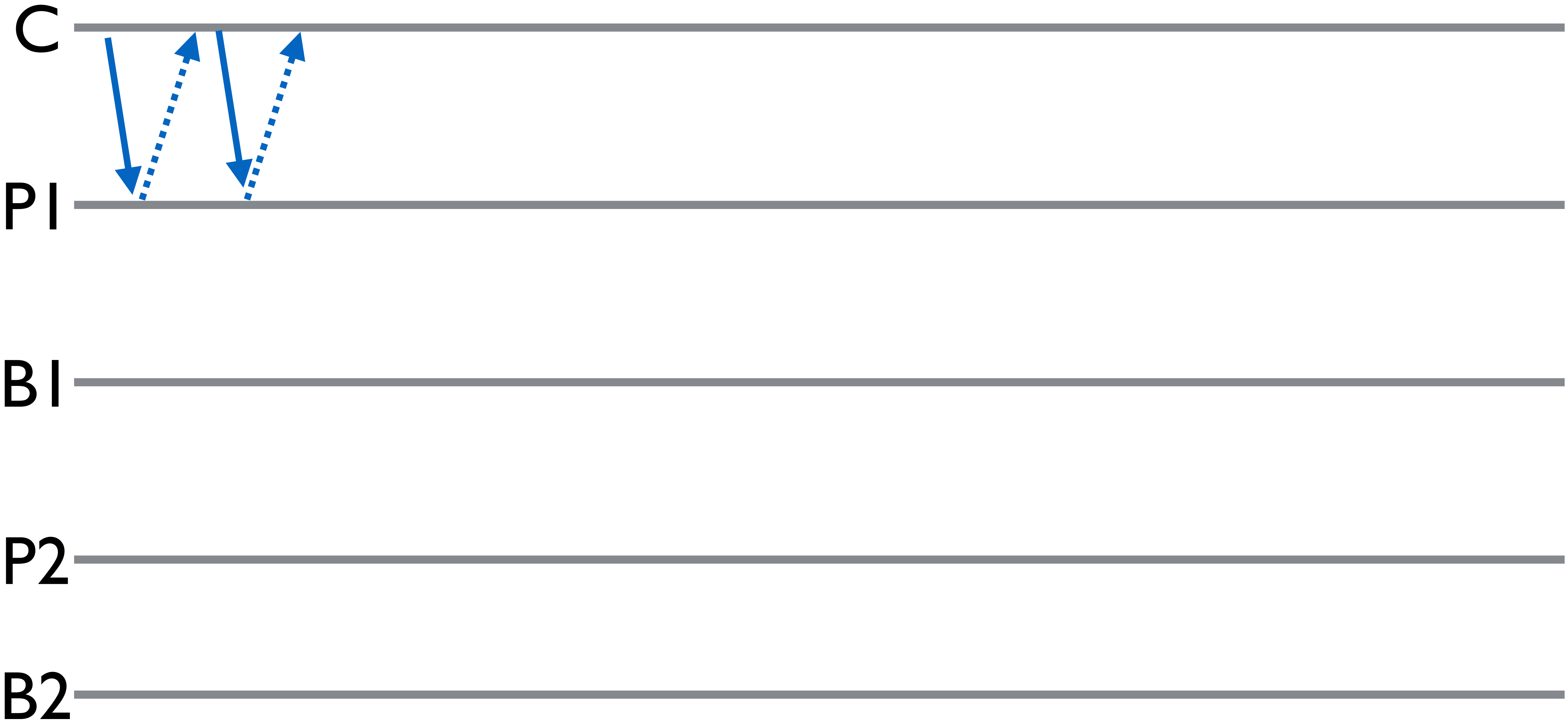
Transactions



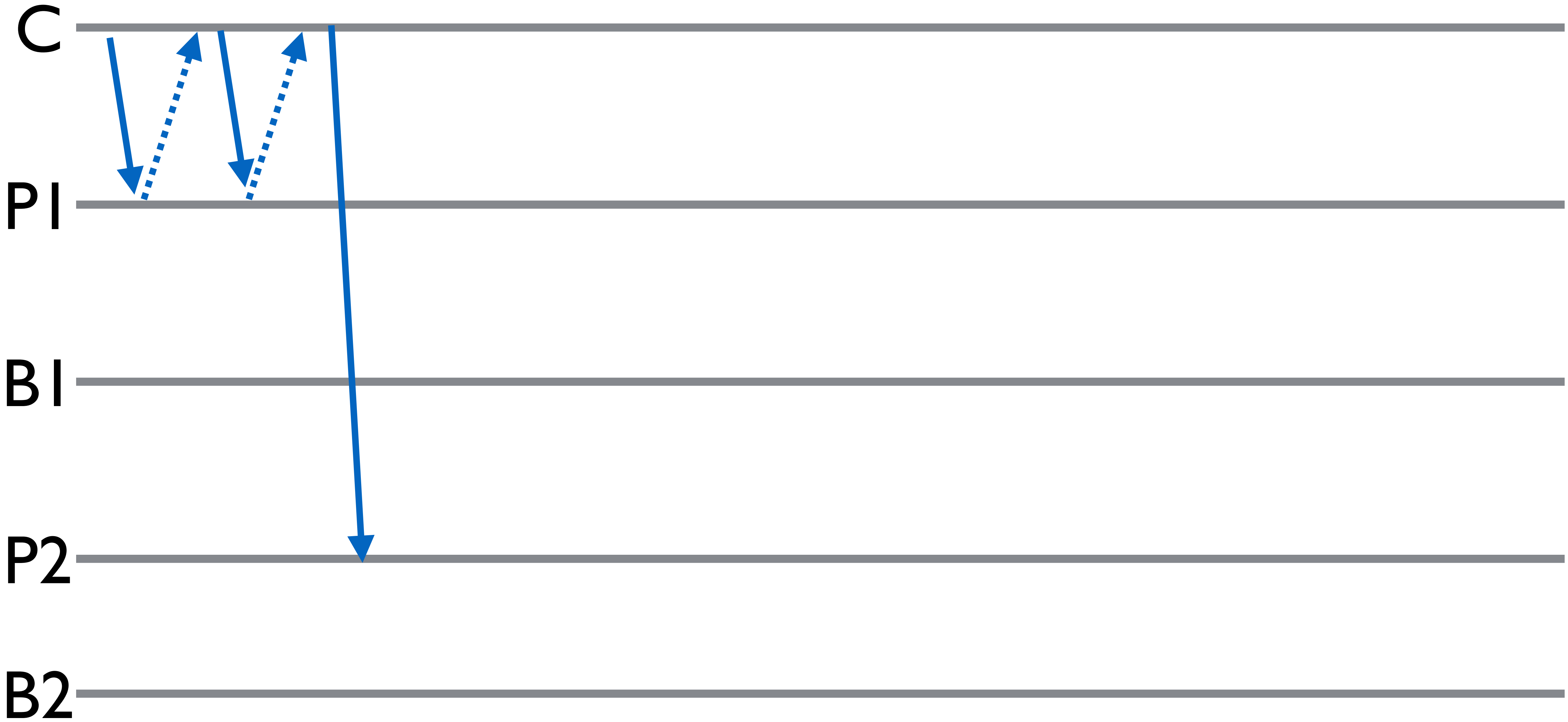
Transactions



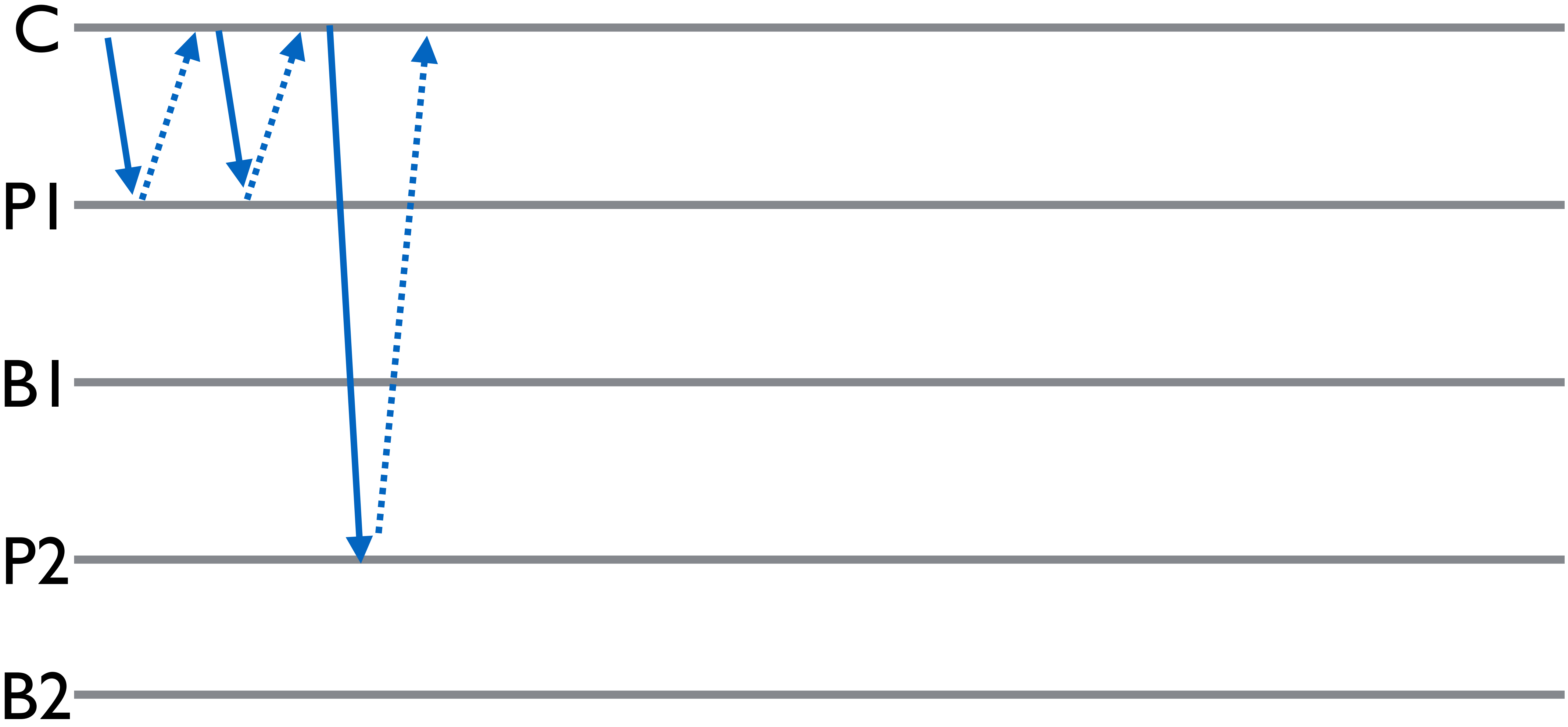
Transactions



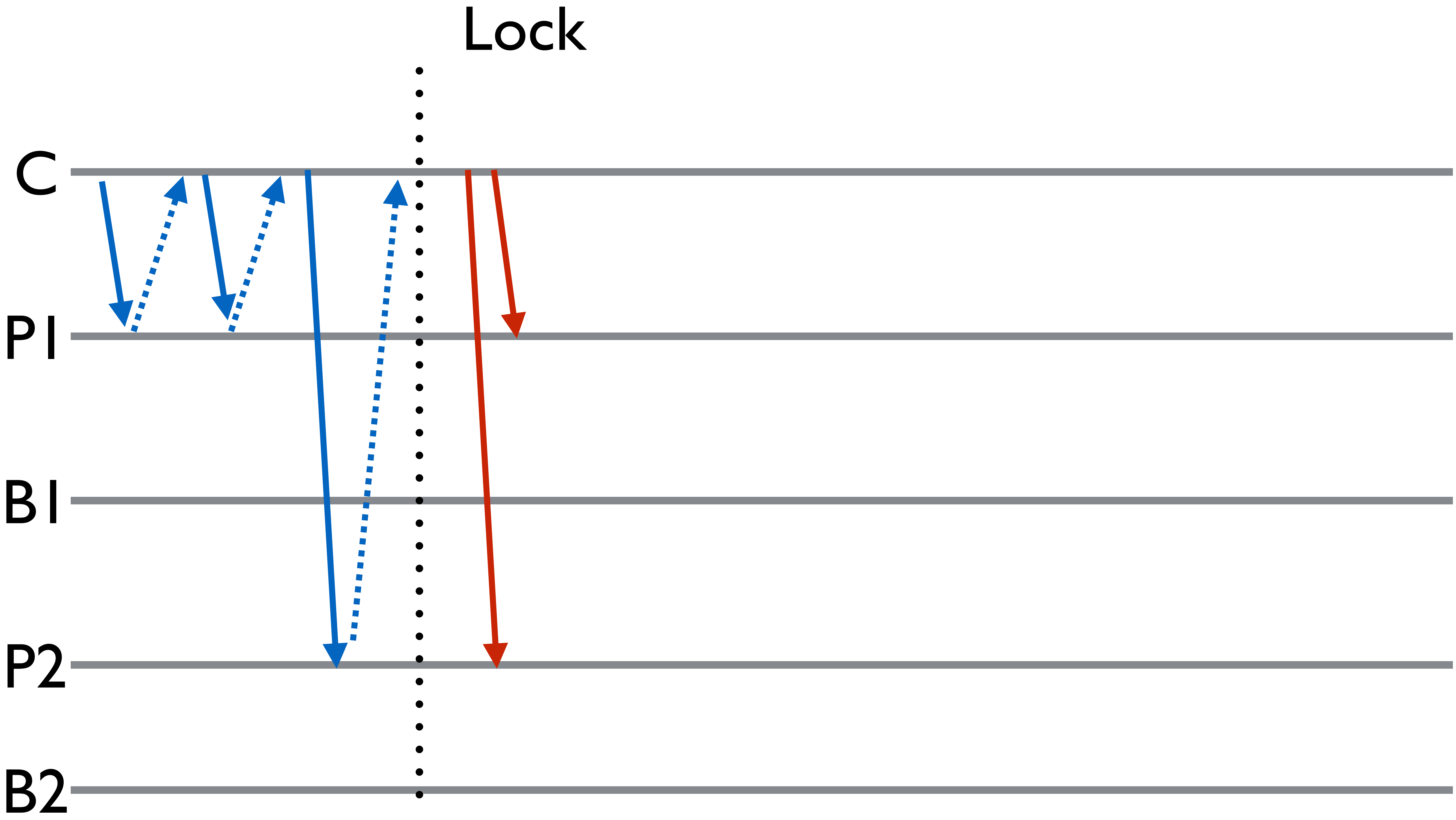
Transactions



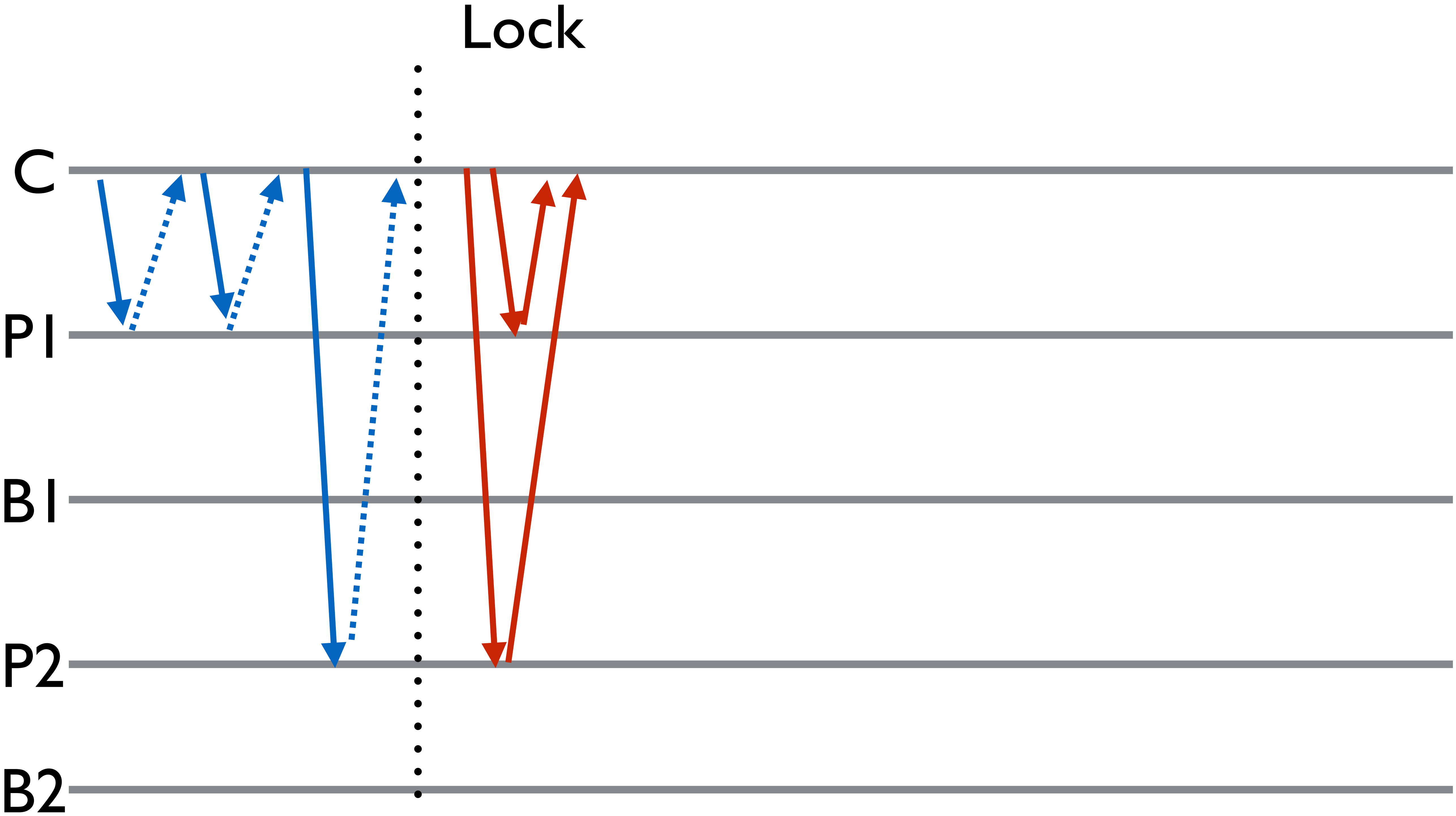
Transactions



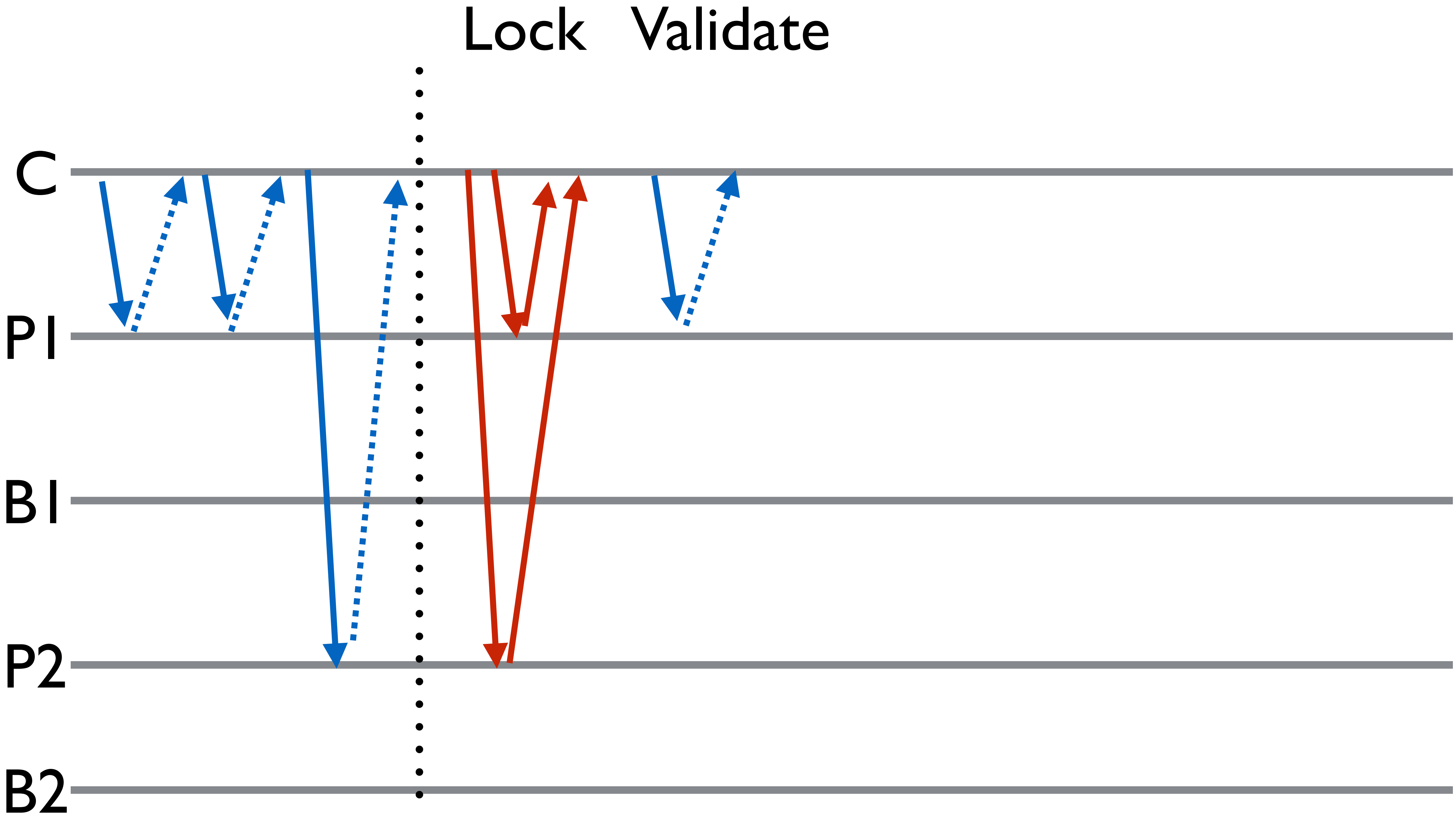
Transactions



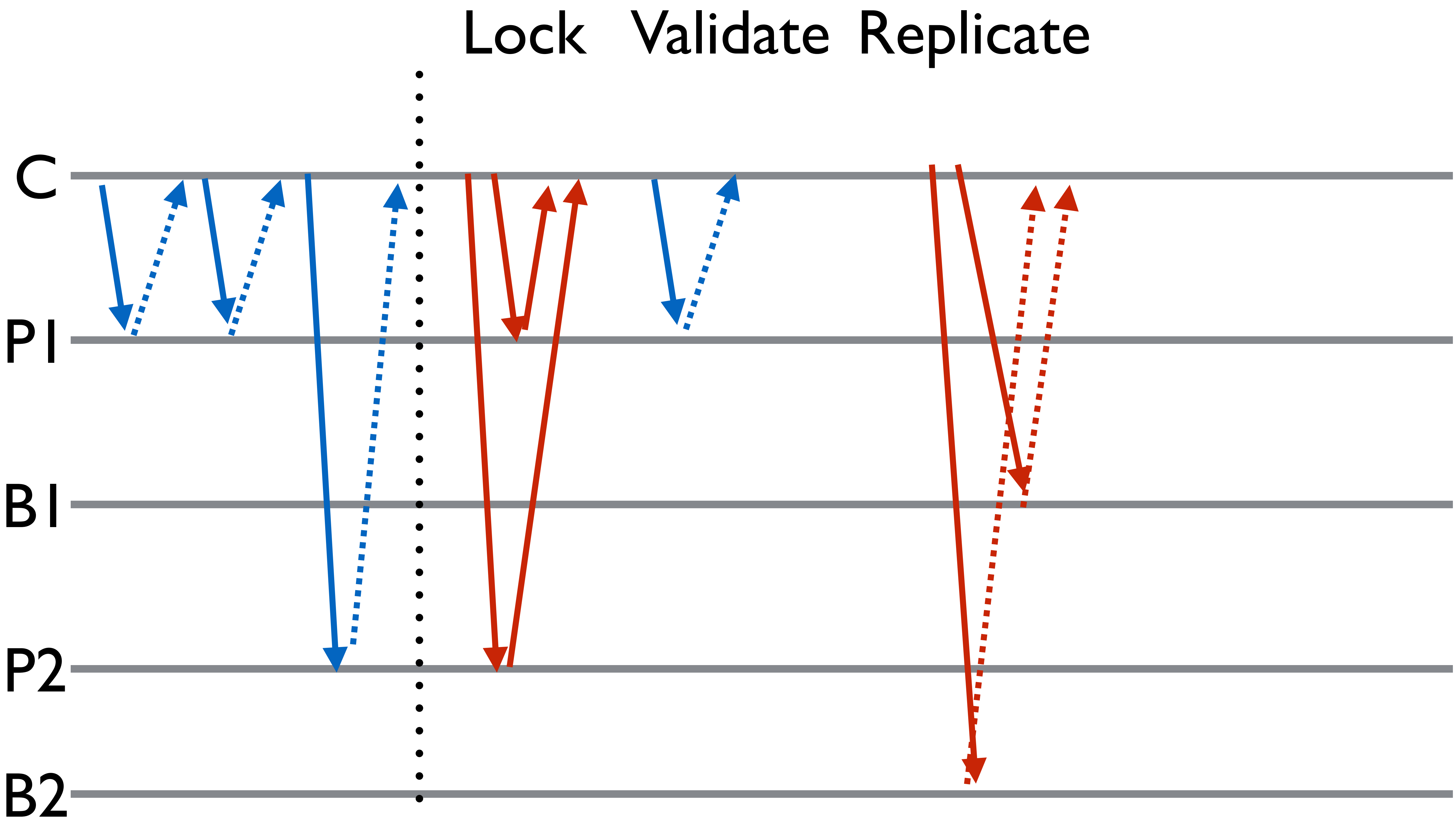
Transactions



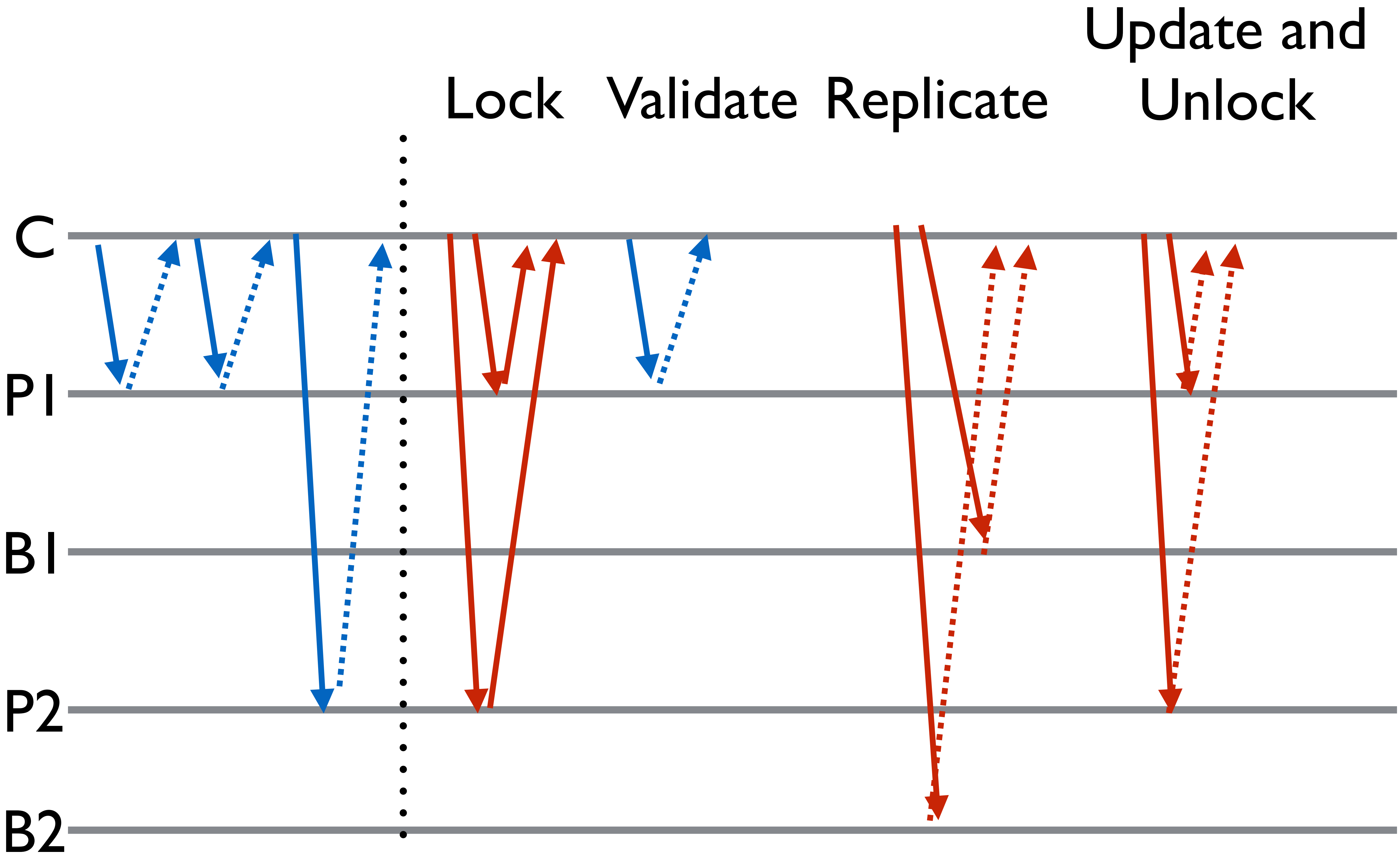
Transactions



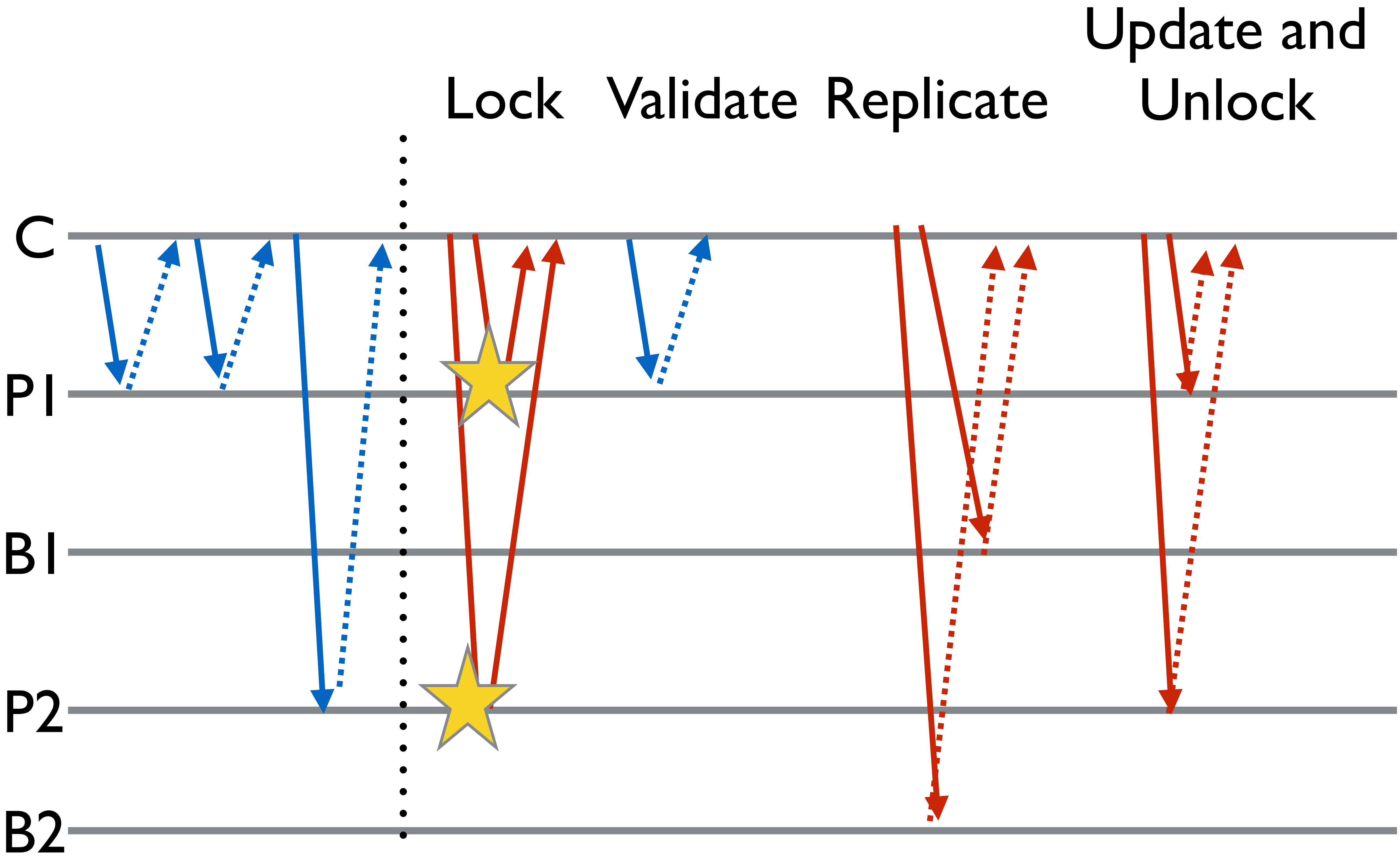
Transactions



Transactions



Transactions



Failures and Recovery

- Complicated by one-side operations — no software running on message reception — can’t reject
- They need to have a way of establishing exactly when and who can continue to send messages — “Precise membership”
- Recovery — transactions might have been reported as committed but the logs are not processed so must drain logs

High Availability

- Fast switch — backups are in memory so can switch fast
- Networks and implementation allows very small leases — 10ms
- lots of parallelism — unaffected regions can proceed and data recover in parallel