HIGH-PERFORMANCE NETWORKING :: USER-LEVEL NETWORKING :: REMOTE DIRECT MEMORY ACCESS

CS6410

Moontae Lee (Nov 20, 2014)

Part 1

Overview

00

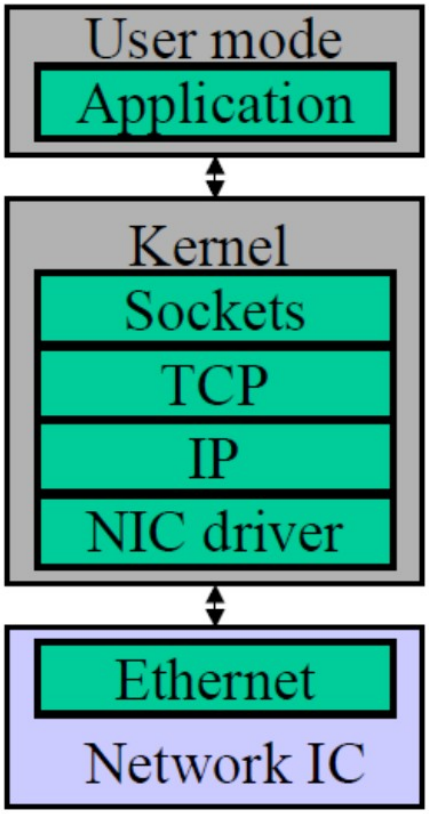
- Background
- User-level Networking (U-Net)
- Remote Direct Memory Access (RDMA)
- Performance

Index

00

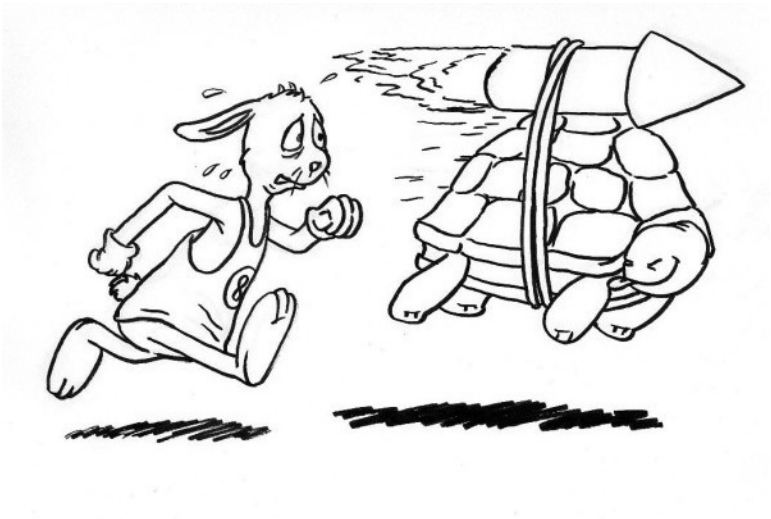
Background

Network Communication



- Send
  - Application buffer →→Socket buffer
  - Attach headers
  - Data is pushed to NIC buffer
- Receive
  - NIC buffer →→Socket buffer
  - Parsing headers
  - Data is copied into Application buffer
  - Application is scheduled (context switching)

Today's Theme



Faster and lightweight communication!

Terms and Problems

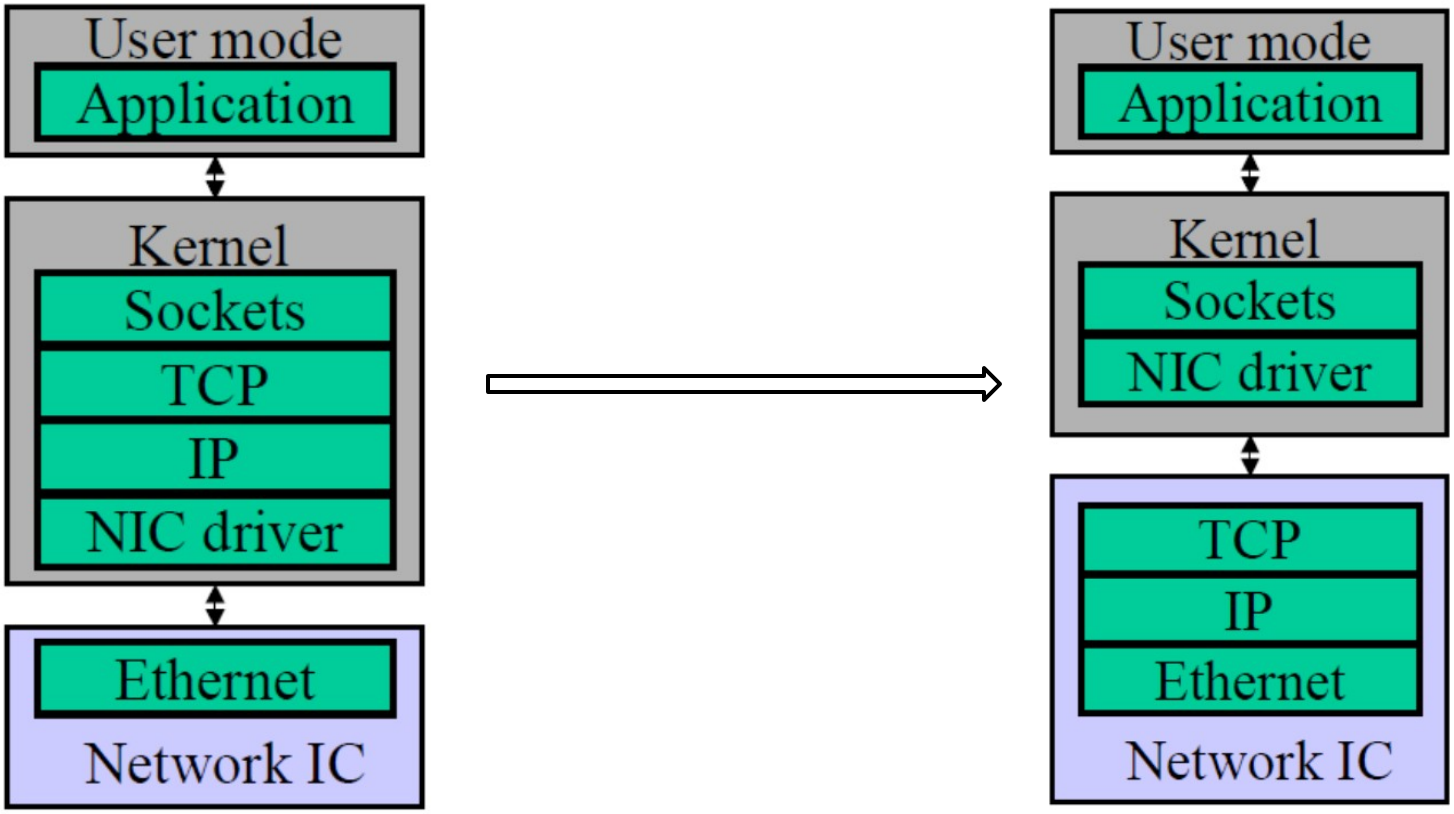
- Communication latency
  - Processing overhead: message-handling time at sending/receiving ends
  - Network latency: message transmission time between two ends (i.e., end-to-end latency)

Terms and Problems

- Communication latency
  - Processing overhead: message-handling time at sending/receiving ends
  - Network latency: message transmission time between two ends (i.e., end-to-end latency)

- If network environment satisfies
  - High bandwidth / Low network latency
  - Long connection durations / Relatively few connections

TCP Offloading Engine (TOE)



THIS IS **NOT** OUR STORY!

Our Story

- Large vs Small messages
  - Large: transmission dominant →→new networks improves
    (e.g., video/audio stream)
  - Small: processing dominant →→new paradigm improves
    (e.g., just a few hundred bytes)

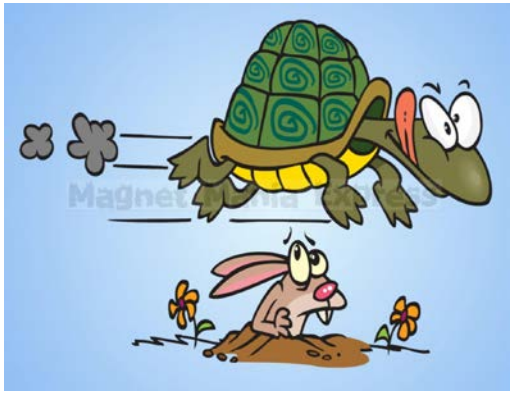
Our Story

- Large vs Small messages
  - Large: transmission dominant →→new networks improves
    (e.g., video/audio stream)
  - Small: processing dominant →→new paradigm improves
    (e.g., just a few hundred bytes)

- Our underlying picture
  - Sending many small messages in LAN
  - Processing overhead is overwhelming
    (e.g., buffer management, message copies, interrupt)

Traditional Architecture

- Problem: Messages pass through the kernel
  - Low performance
    - Duplicate several copies
    - Multiple abstractions between device driver and user apps
  - Low flexibility
    - All protocol processing inside the kernel
    - Hard to support new protocols and new message send/receive interfaces



History of High-Performance

- User-level Networking (U-Net)
  - One of the first kernel-bypassing systems
- Virtual Interface Architecture (VIA)
  - First attempt to standardize user-level communication
  - Combine U-Net interface with remote DMA service
- Remote Direct Memory Access (RDMA)
  - Modern high-performance networking
  - Many other names, but sharing common themes

Index

00

U-Net

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

- Focusing on small messages, key goals are:
  - High performance / High flexibility

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

- Focusing on small messages, key goals are:
  - High performance / High flexibility
    - Low communication latency in local area setting
    - Exploit full bandwidth
    - Emphasis on protocol design and integration flexibility
    - Portable to off-the-shelf communication hardware

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

- Focusing on small messages, key goals are:
  - High performance / High flexibility
    - Low communication latency in local area setting
    - Exploit full bandwidth
    - Emphasis on protocol design and integration flexibility
    - Portable to off-the-shelf communication hardware

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

- Focusing on small messages, key goals are:
  - High performance / High flexibility
    - Low communication latency in local area setting
    - Exploit full bandwidth
    - Emphasis on protocol design and integration flexibility
    - Portable to off-the-shelf communication hardware

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

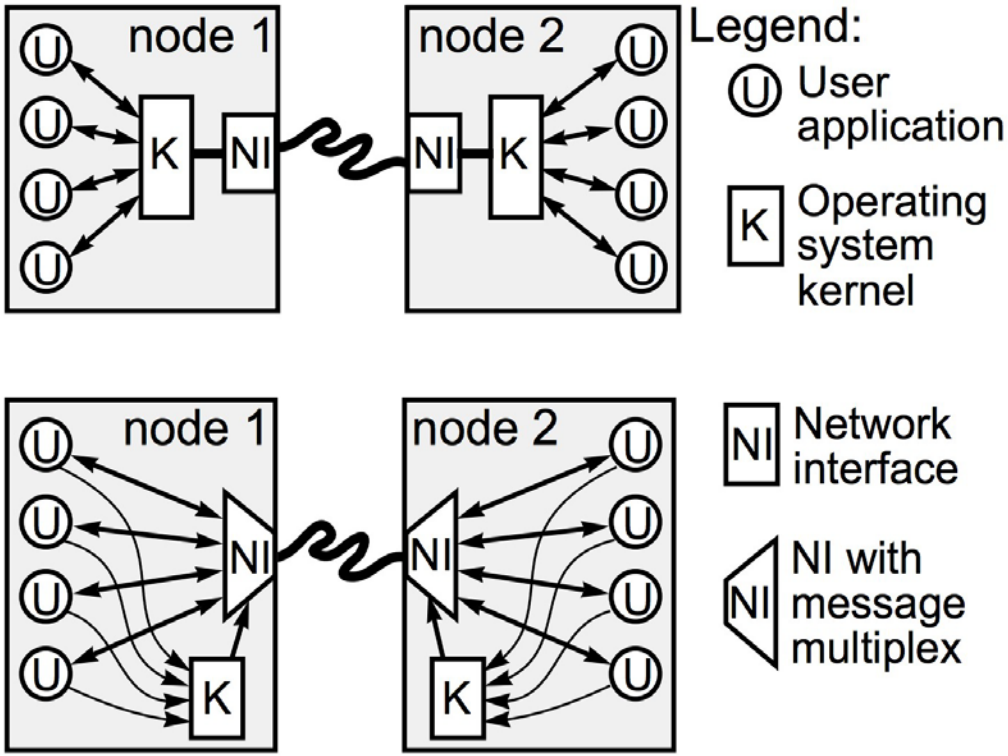
- Focusing on small messages, key goals are:
  - High performance / High flexibility
    - Low communication latency in local area setting
    - Exploit full bandwidth
    - Emphasis on protocol design and integration flexibility
    - Portable to off-the-shelf communication hardware

U-Net Ideas and Goals

- Move protocol processing parts into user space!
  - Move the entire protocol stack to user space
  - Remove kernel completely from data communication path

- Focusing on small messages, key goals are:
  - High performance / High flexibility
    - Low communication latency in local area setting
    - Exploit full bandwidth
    - Emphasis on protocol design and integration flexibility
    - Portable to off-the-shelf communication hardware

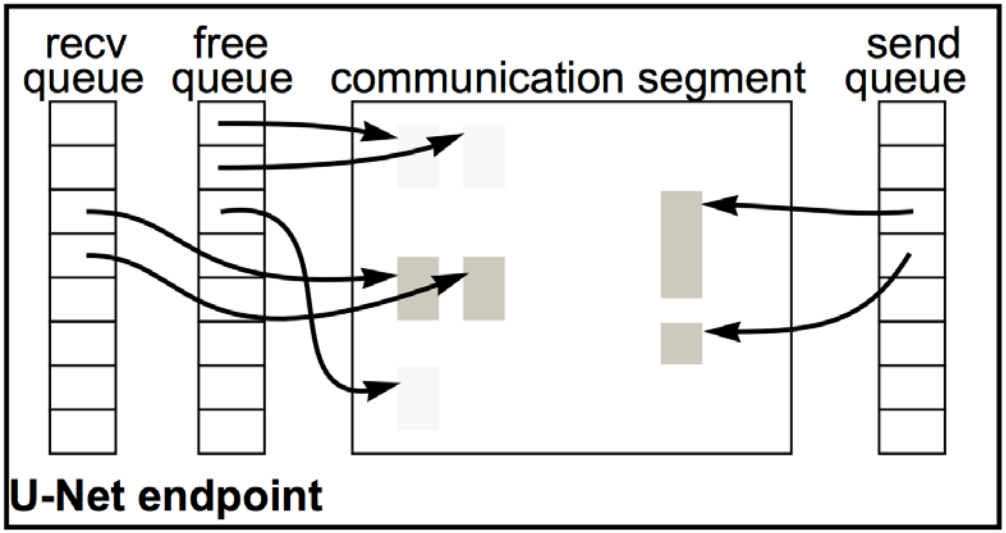
U-Net Architecture



- Traditionally
  - Kernel controls network
  - All communications via the kernel
- U-Net
  - Applications can access network directly via MUX
  - Kernel involves only in connection setup

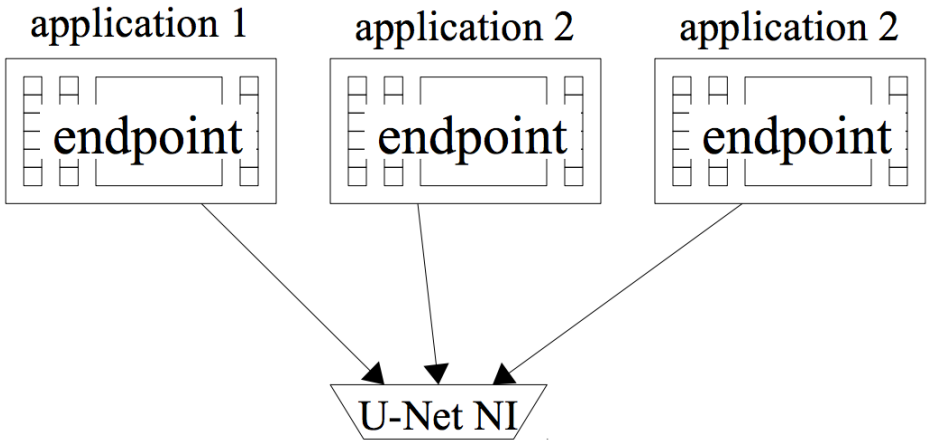
* Virtualize NI →provides each process the illusion of owning interface to network

U-Net Building Blocks



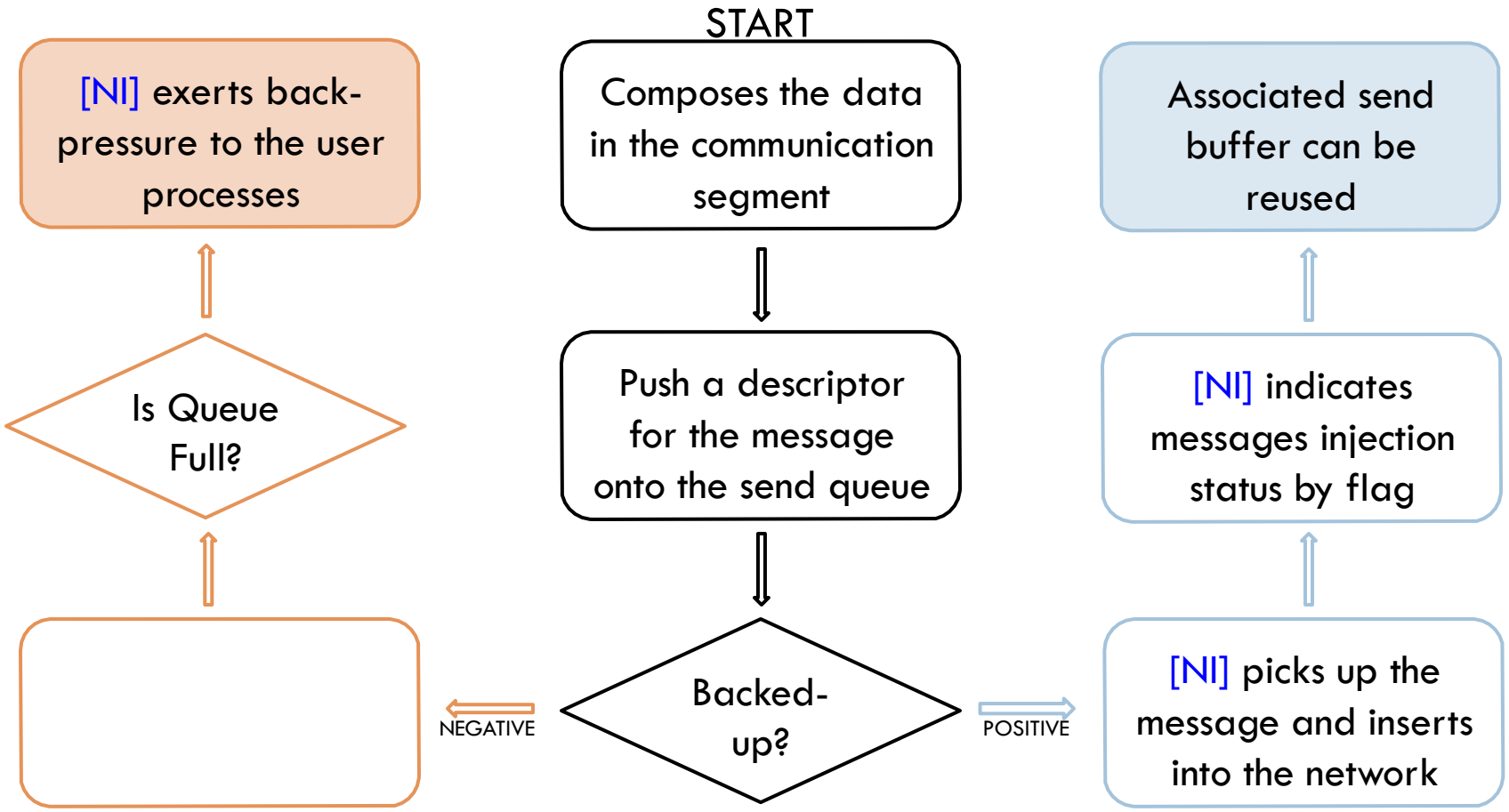
- **End points:** application's / kernel's handle into network
- **Communication segments:** memory buffers for sending/receiving messages data
- **Message queues:** hold descriptors for messages that are to be sent or have been received

U-Net Communication: Initialize



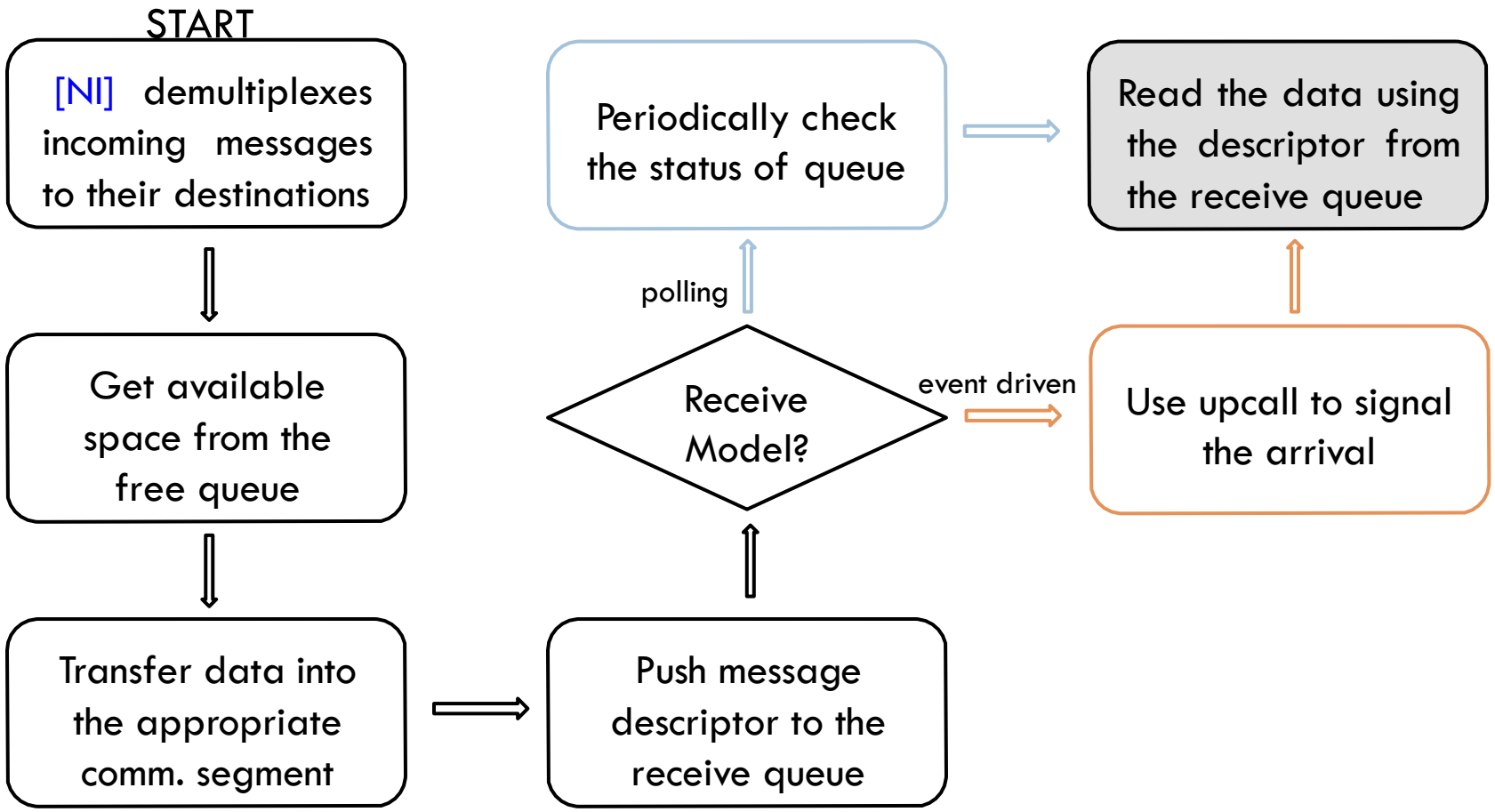
- Initialization:
  - Create single/multiple endpoints for each application
  - Associate a communication segment and send/receive/free message queues with each endpoint

U-Net Communication: Send



Send as simple as changing one or two pointers!

U-Net Communication: Receive



Receive as simple as NIC changing one or two pointers!

U-Net Protection

- Owning process protection
  - Endpoints
  - Communication segments
  - Send/Receive/Free queues

Only owning process can access!

- Tag protection
  - Outgoing messages are tagged with the originating endpoint address
  - Incoming messages are only delivered to the correct destination endpoint

U-Net Zero Copy

- Base-level U-Net (might not be ‘zero’ copy)
  - Send/receive needs a buffer
  - Requires a copy between application data structures and the buffer in the communication segment
  - Can also keep the application data structures in the buffer without requiring a copy
- Direct Access U-Net (true ‘zero’ copy)
  - Span the entire process address space
  - But requires special hardware support to check address

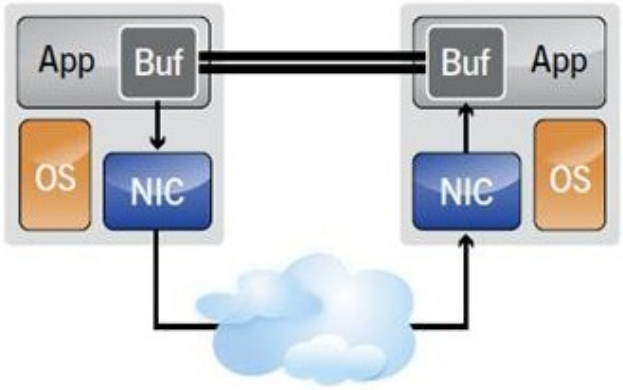
Index

00

RDMA

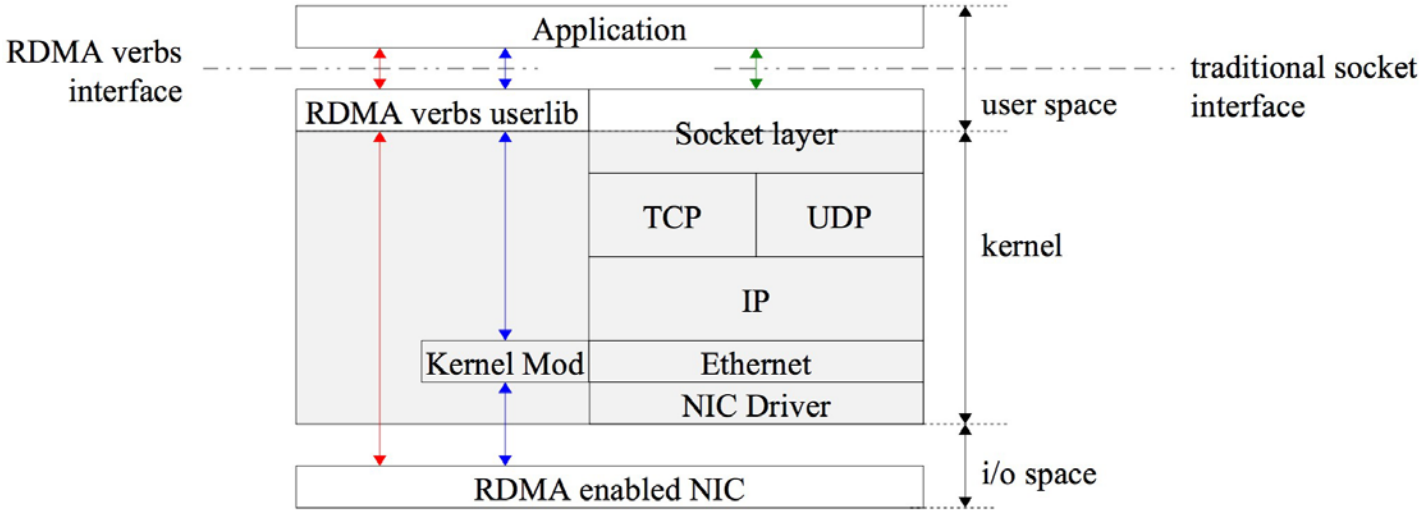
RDMA Ideas and Goals

- Move buffers between two applications via network



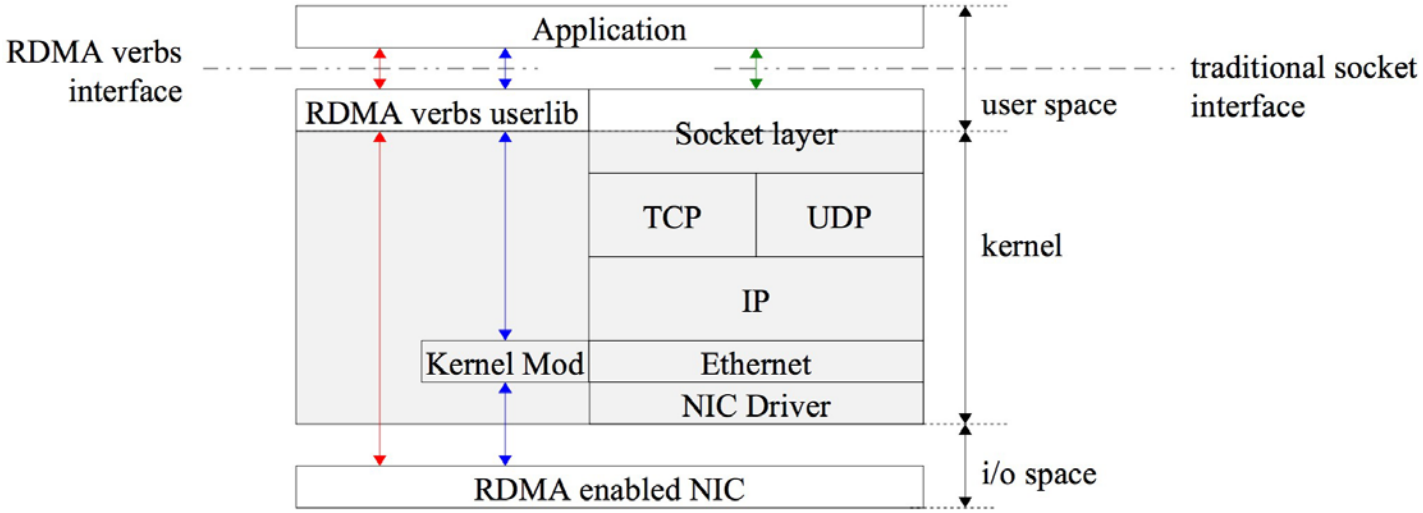
- Once programs implement RDMA:
  - Tries to achieve lowest latency and highest throughput
  - Smallest CPU footprint

RDMA Architecture (1/2)



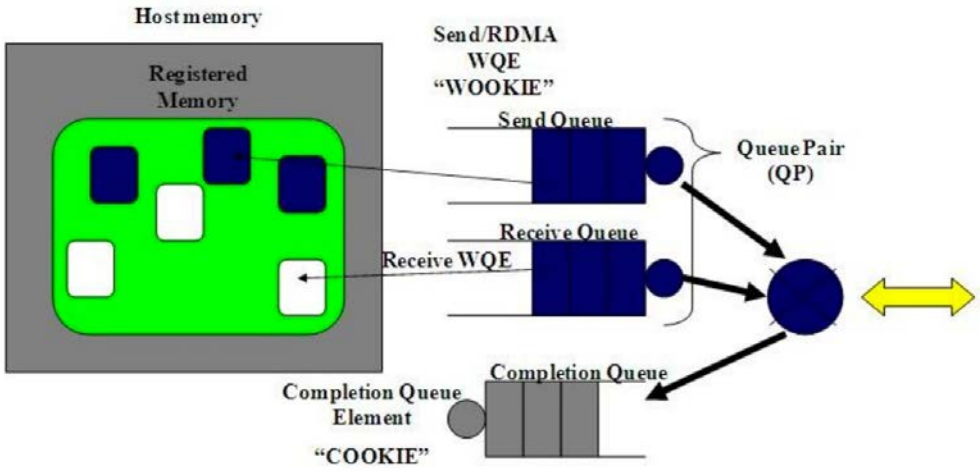
- Traditionally, socket interface involves the kernel
- Has a dedicated **verbs interface** instead of the socket interface
- Involves the kernel only on control path
- Can access rNIC directly from user space on data path bypassing kernel

RDMA Architecture (2/2)



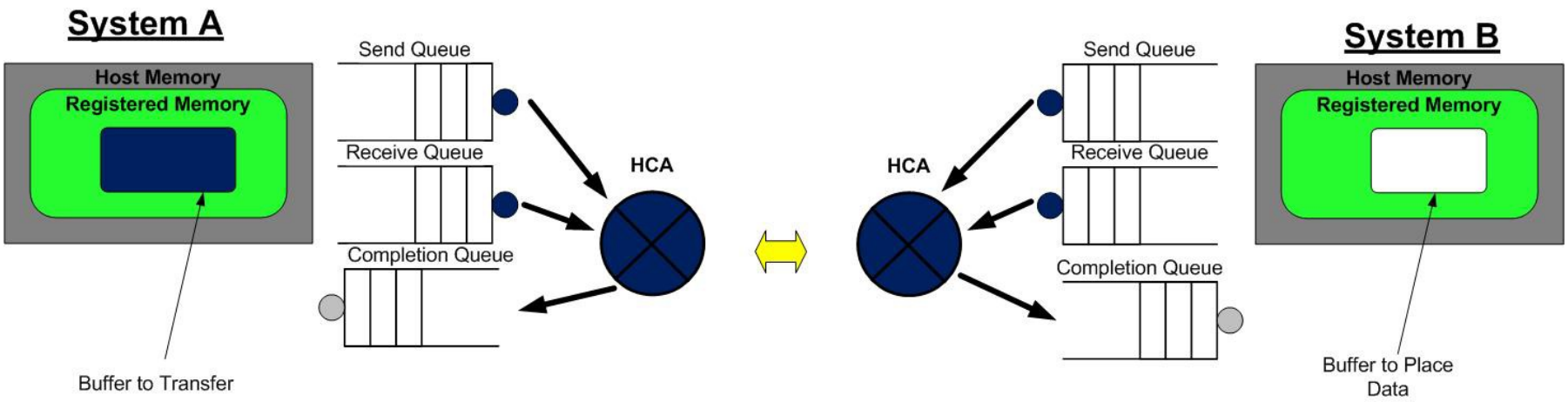
- To initiate RDMA, establish data path from RNIC to application memory
- **Verbs interface** provide API to establish these data path
- Once data path is established, directly read from/write to buffers
- Verbs interface is different from the traditional socket interface.

RDMA Building Blocks



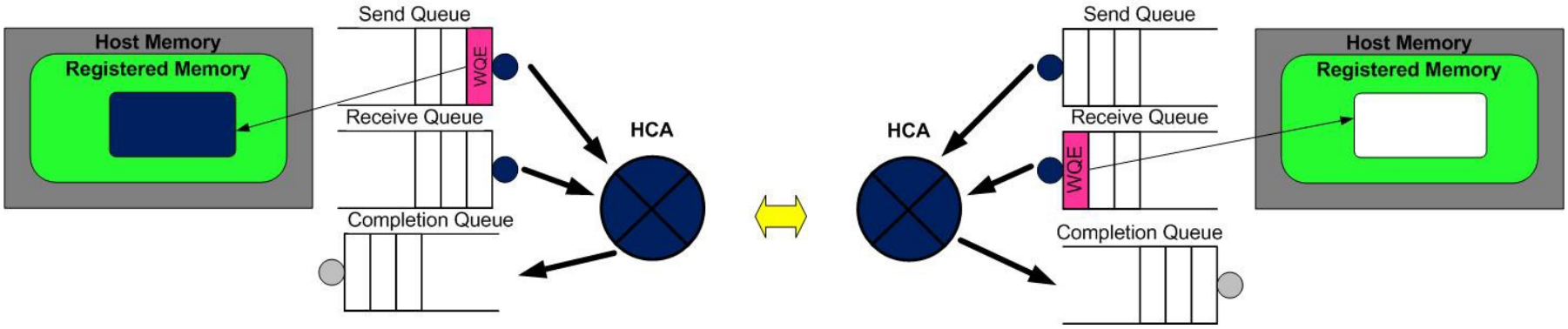
- Applications use **verb interfaces** in order to
  - Register memory: kernel ensures memory is pinned and accessible by DMA
  - Create a queue pair (QP): a pair of send/receive queues
  - Create a completion queue (CQ): RNIC puts a new completion-queue element into the CQ after an operation has completed.
  - Send/receive data

RDMA Communication (1/4)



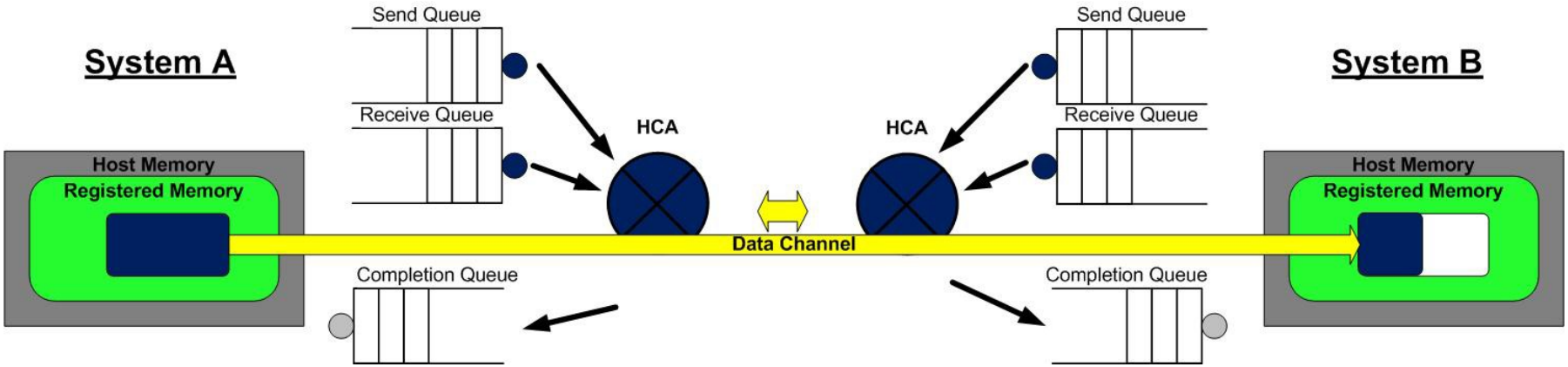
- Step 1

RDMA Communication (2/4)



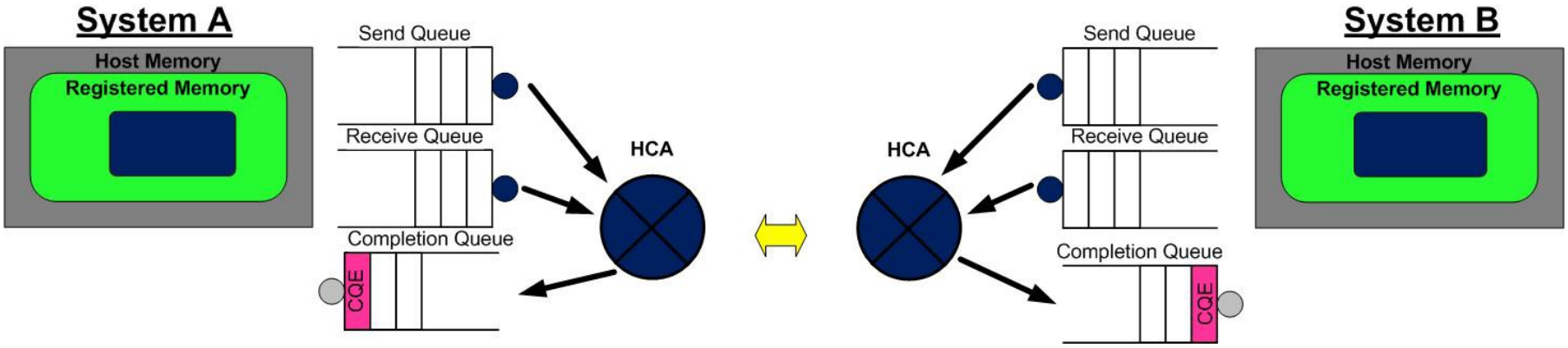
- Step 2

RDMA Communication (3/4)



- Step 3

RDMA Communication (4/4)



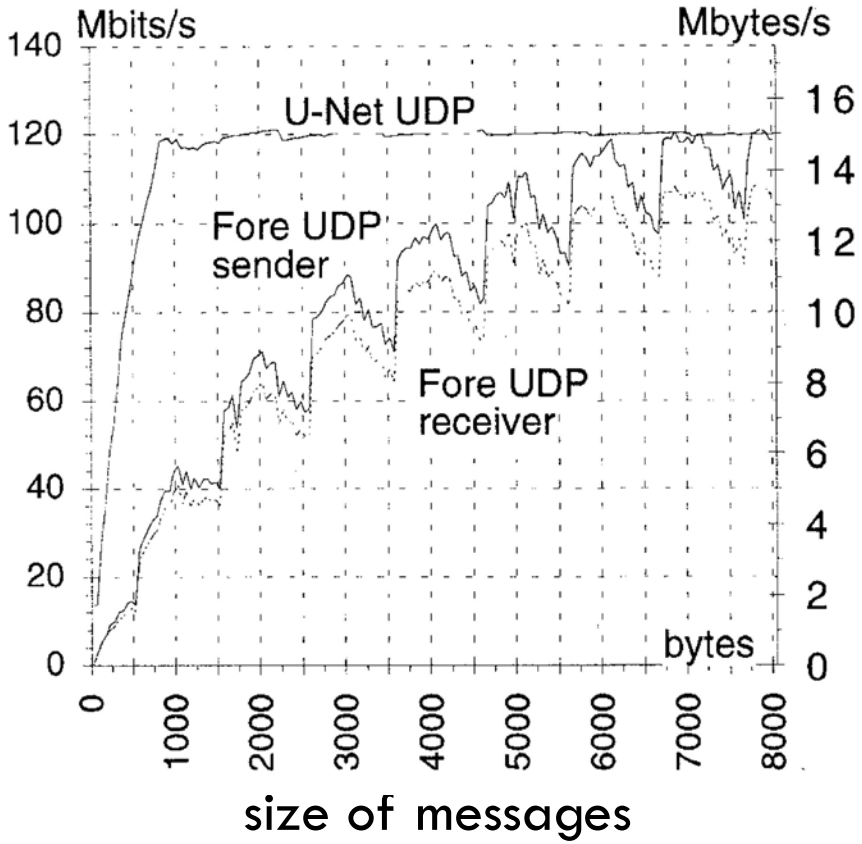
- Step 4

Index

Performance

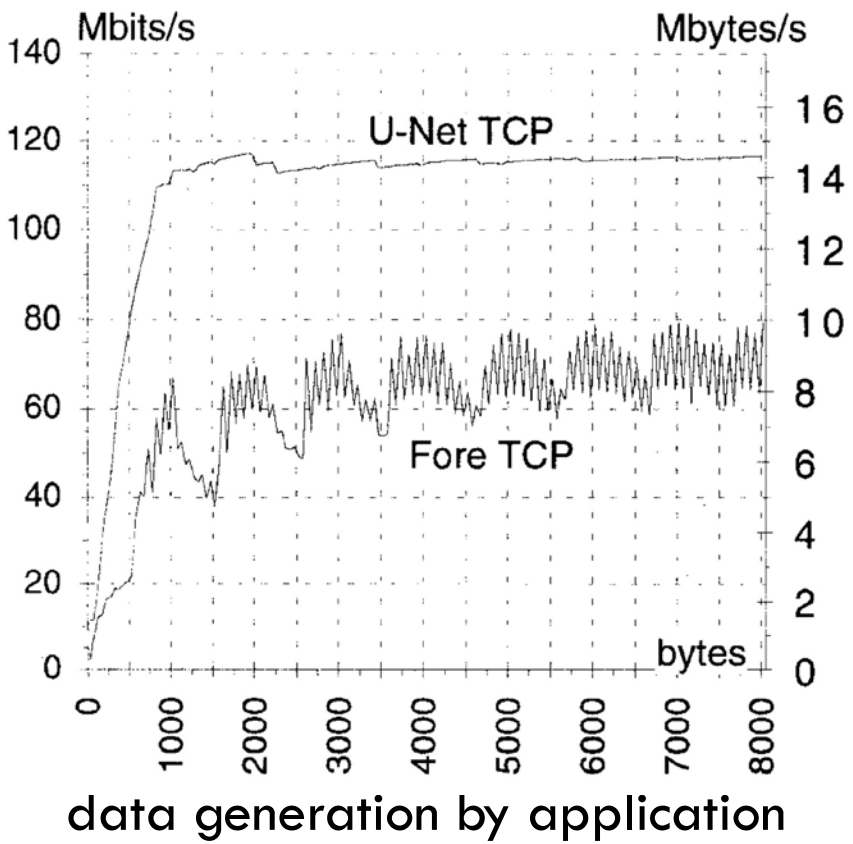
U-Net Performance: Bandwidth

* UDP bandwidth



size of messages

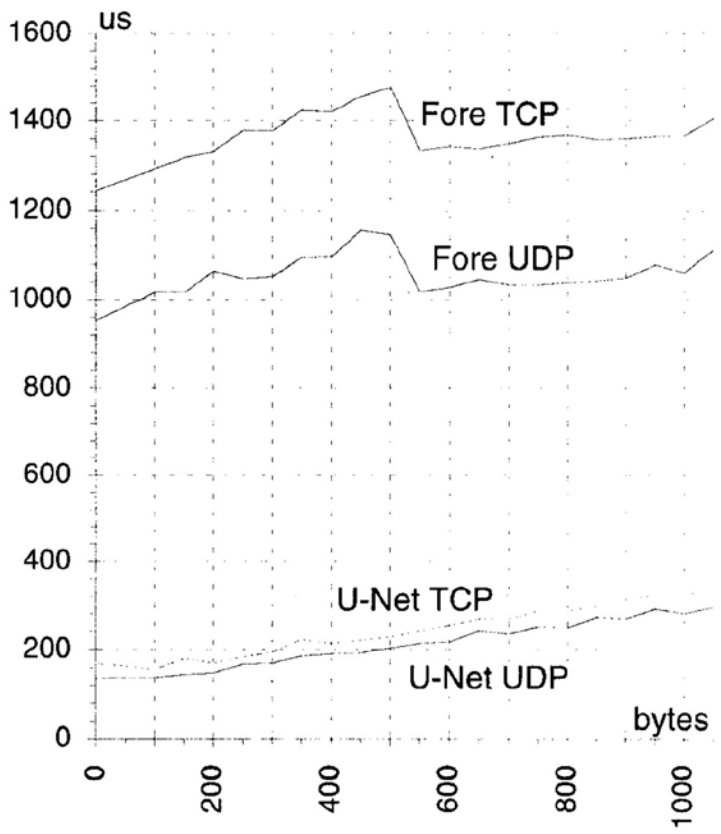
* TCP bandwidth



data generation by application

U-Net Performance: Latency

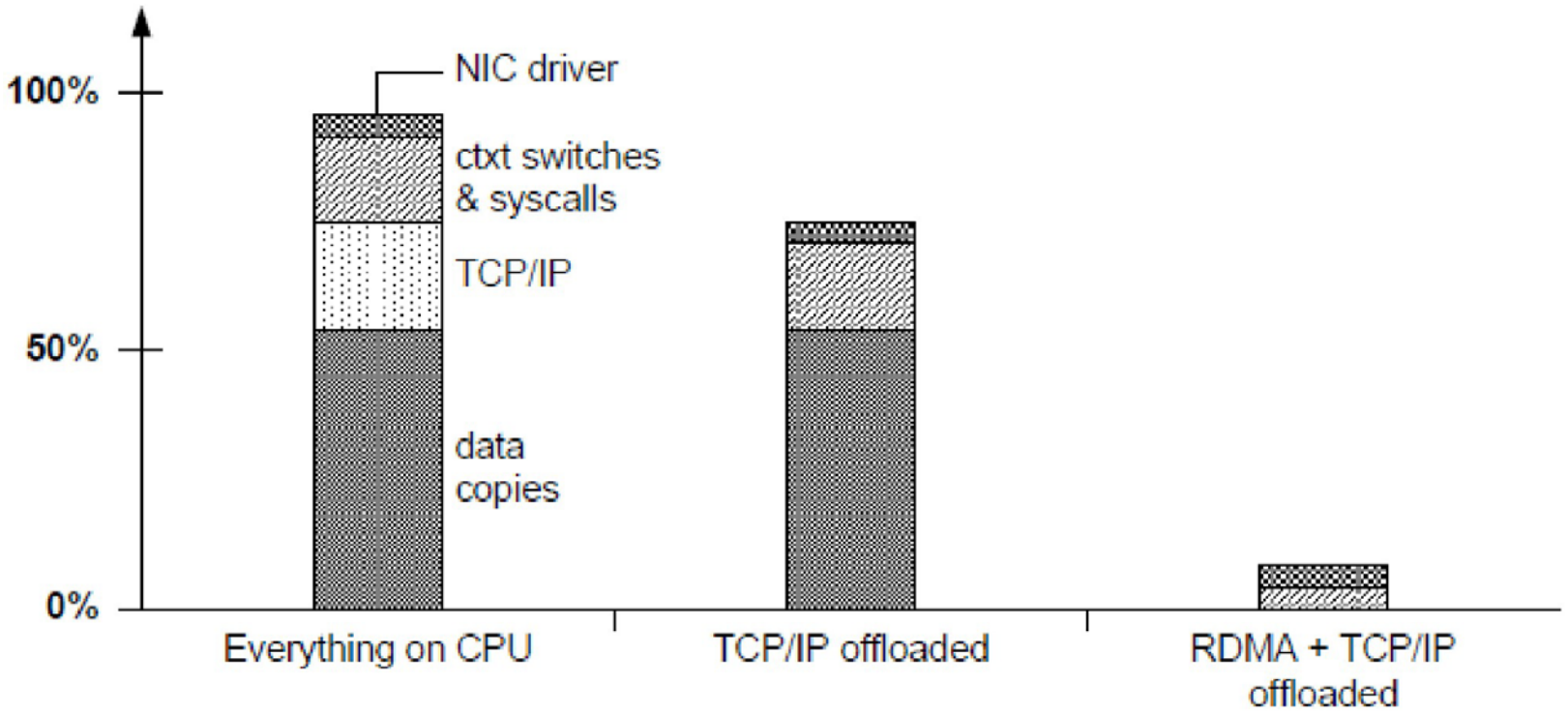
- End-to-end round trip latency



size of messages

RDMA Performance: CPU load

- CPULoad



Modern RDMA

- Several major vendors: Qlogic (Infiniband), Mellanox, Intel, Chelsio, others

- RDMA has evolved from the U/Net approach to have three "modes"
  - Infiniband (Qlogic PSM API): one-sided, no "connection setup"
  - More standard: "qpair" on each side, plus a binding mechanism (one queue is for the sends, or receives, and the other is for sensing completions)
  - One-sided RDMA: after some setup, allows one side to read or write to the memory managed by the other side, but pre-permission is required
  - RDMA + VLAN: needed in data centers with multitenancy

Modern RDMA

- Memory management is tricky:
  - Pages must be pinned and mapped into IOMMU
  - Kernel will zero pages on first allocation request: slow
  - If a page is a mapped region from a file, kernel may try to automatically issue a disk write after updates, costly
  - Integration with modern NVRAM storage is “awkward”
  - On multicore NUMA machines, hard to know which core owns a particular memory page, yet this matters

- Main reason we should care?
  - RDMA runs at 20, 40Gb/s. And soon 100, 200… 1Tb/s
  - But memcpy and memset run at perhaps 30Gb/s

SoftROCE

- Useful new option
- With standard RDMA may people worry programs won't be portable and will run only with one kind of hardware
- SoftROCE allows use of the RDMA software stack (libibverbs.dll) but tunnels via TCP hence doesn't use hardware RDMA at all
  - Zero-copy sends, but needs one-copy for receives

- Intel iWarp is aimed at something similar
  - Tries to offer RDMA with zero-copy on both sides under the TCP API by dropping TCP into the NIC
  - Requires a special NIC with an RDMA chip-set

HIGH-PERFORMANCE NETWORKING :: USER-LEVEL NETWORKING :: REMOTE DIRECT MEMORY ACCESS

CS6410

Jaeyong Sung (Nov 20, 2014)

Part 2

Hadoop

- Big Data

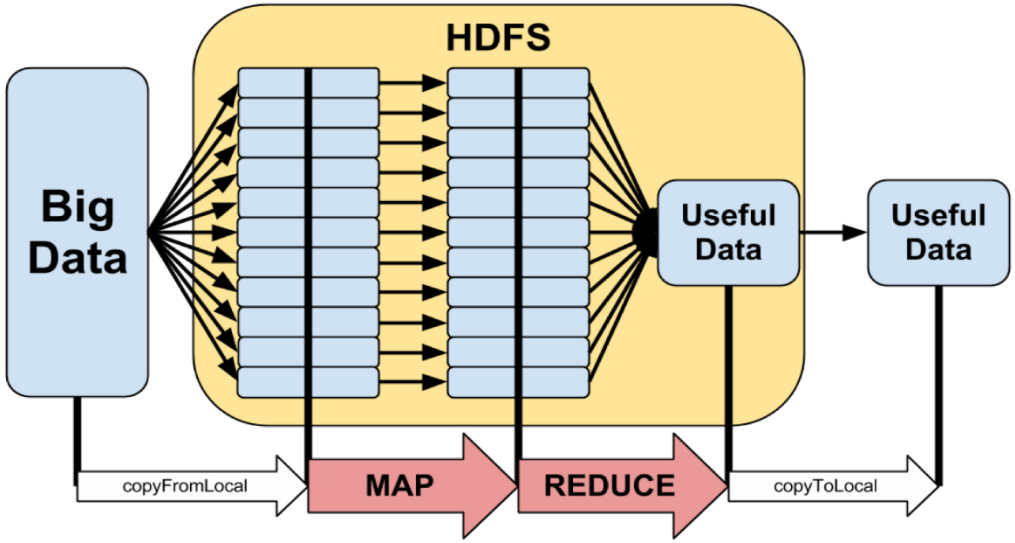
  very common in industries

  - e.g. Facebook, Google, Amazon, …

- Hadoop

  open source MapReduce for handling large data

  require lots of data transfers



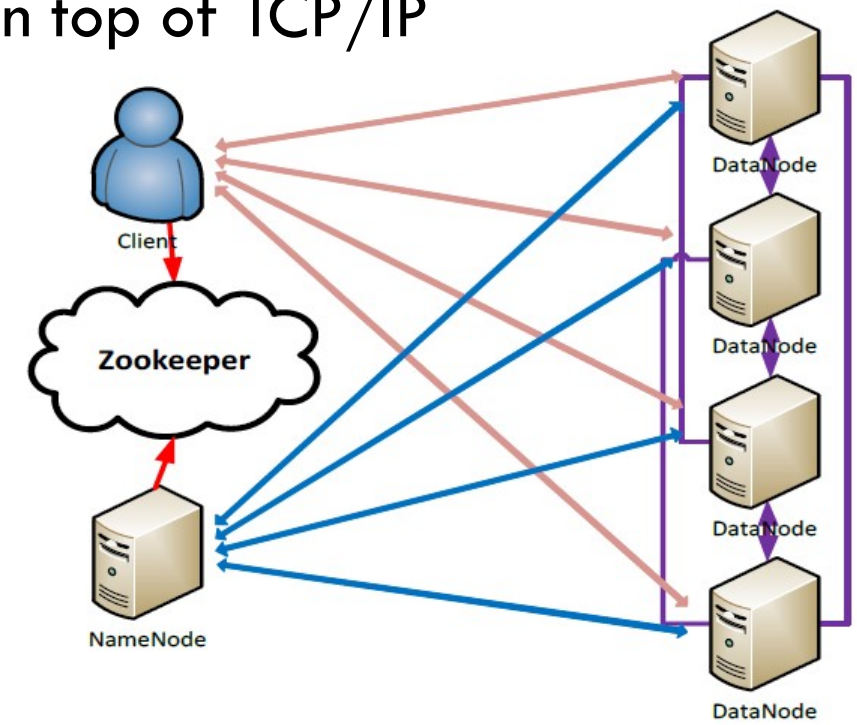
Hadoop Distributed File System

- primary storage for Hadoop clusters

  both Hadoop MapReduce and HBase rely on it

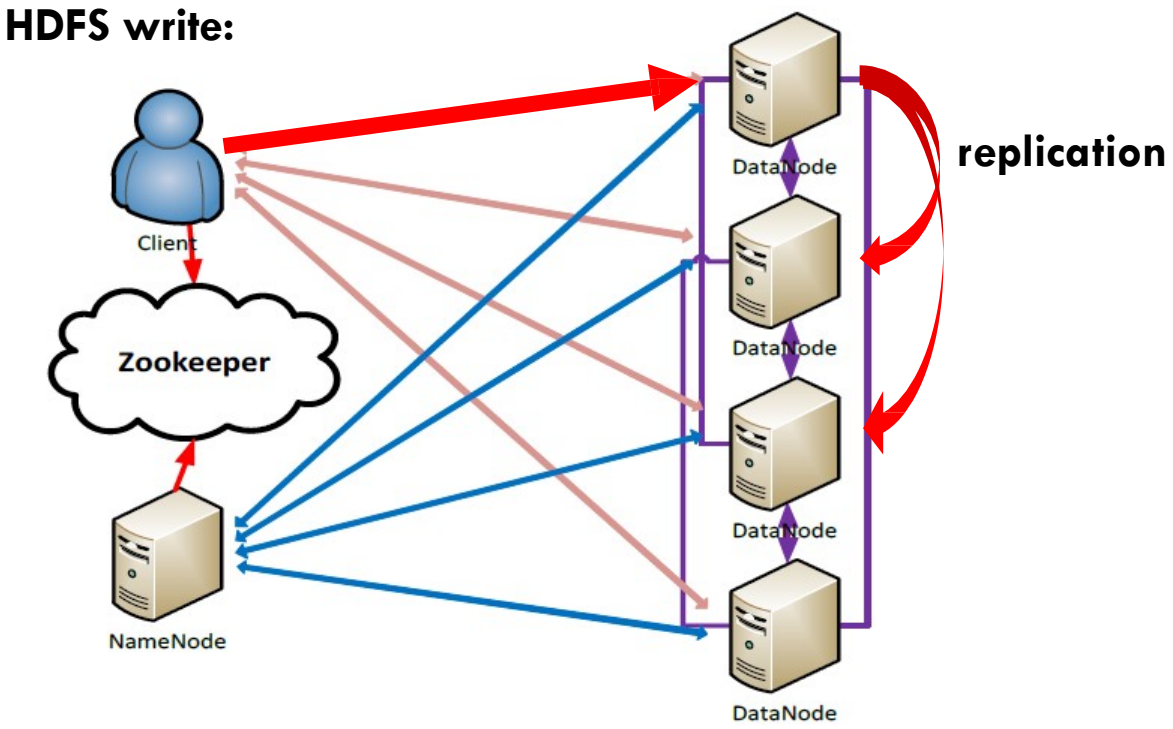
- communication intensive middleware

  layered on top of TCP/IP



Hadoop Distributed File System

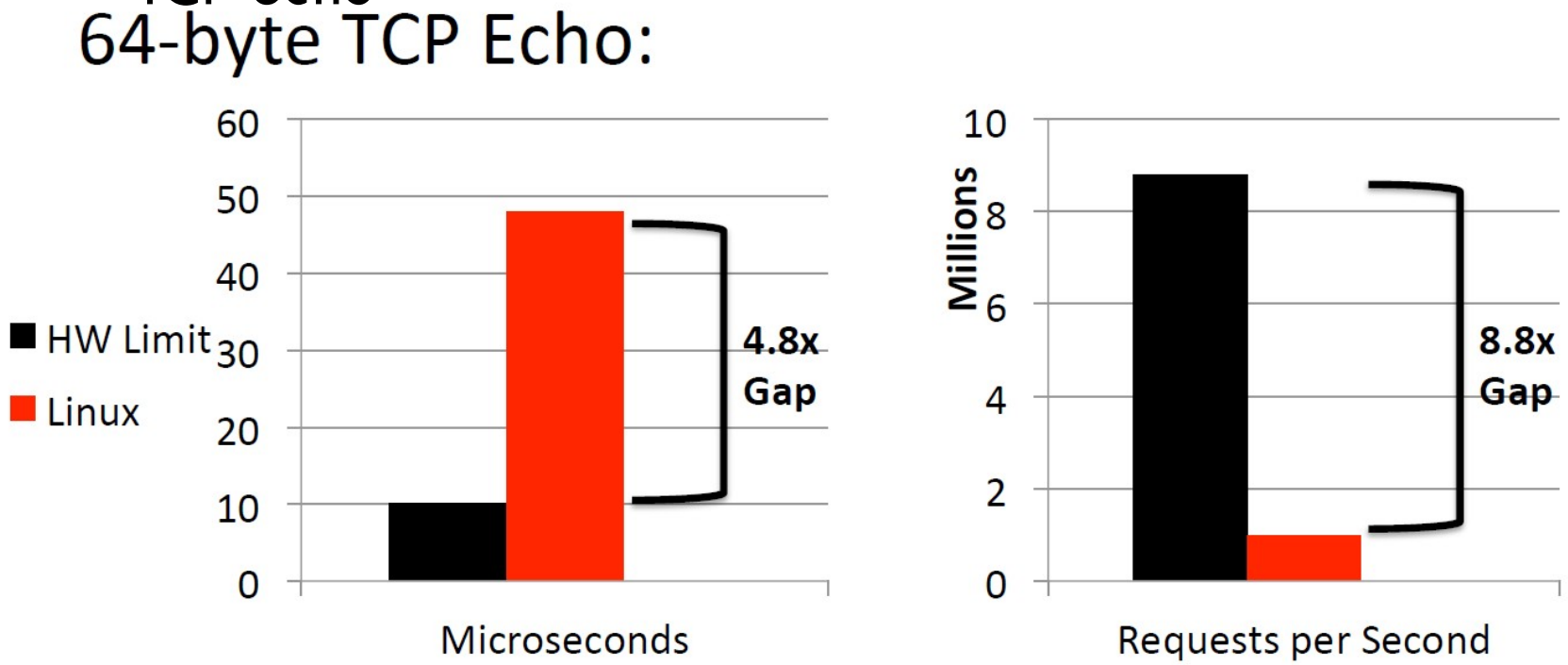
- highly reliable fault-tolerant replications
- in data-intensive applications, network performance becomes key component



Software Bottleneck

- Using TCP/IP on Linux,

TCP echo



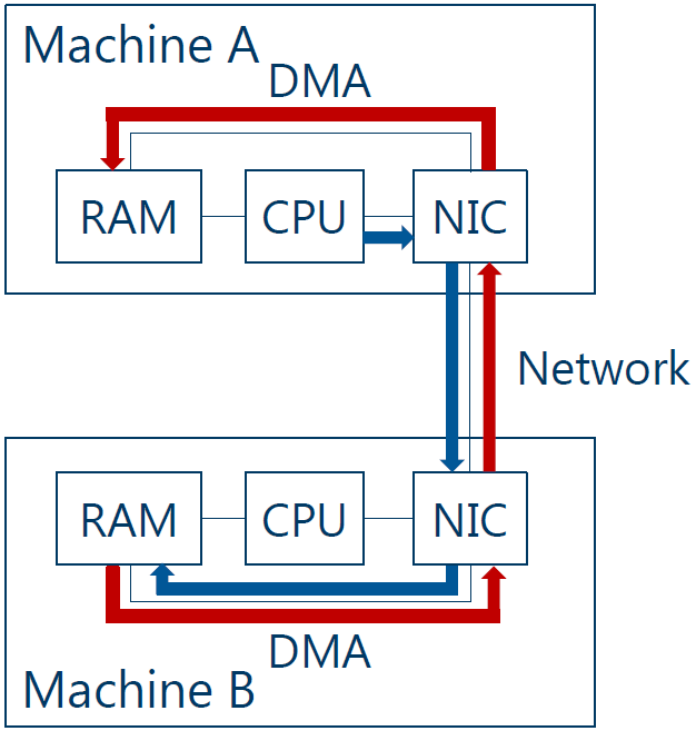
RDMA for HDFS

- data structure for Hadoop

  <key, value> pairs

  stored in data blocks of HDFS

- Both write(replication) and read can take advantage of RDMA

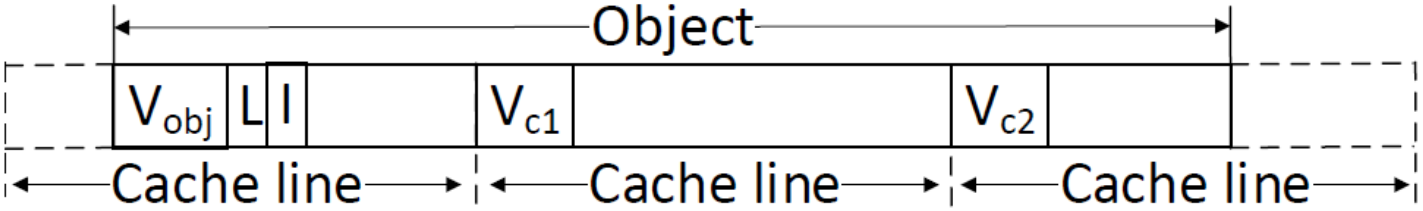


FaRM: Fast Remote Memory

- relies on cache coherent DMA

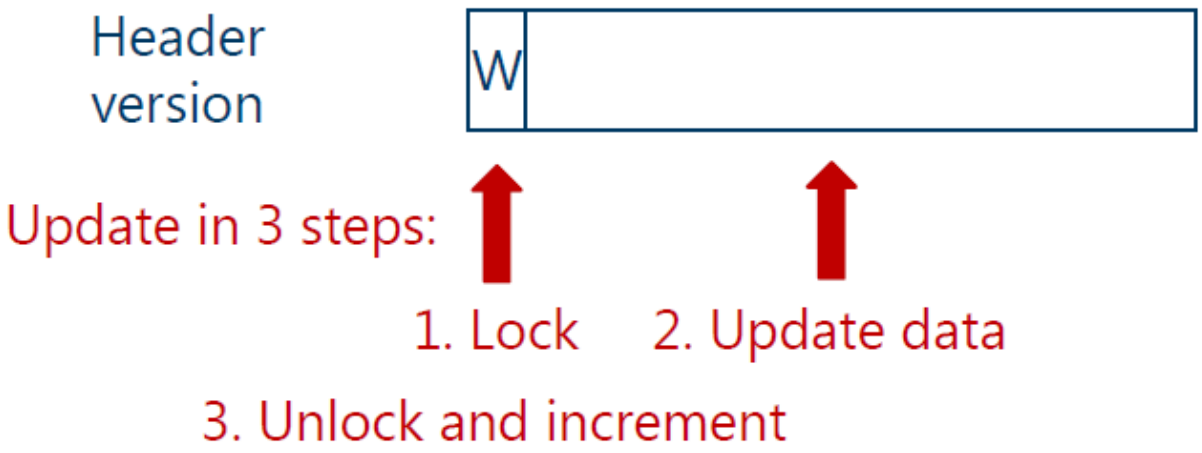
  object's version number

  - stored both in the first word of the object header and at the start of each cache line
  - NOT visible to the application (e.g. HDFS)



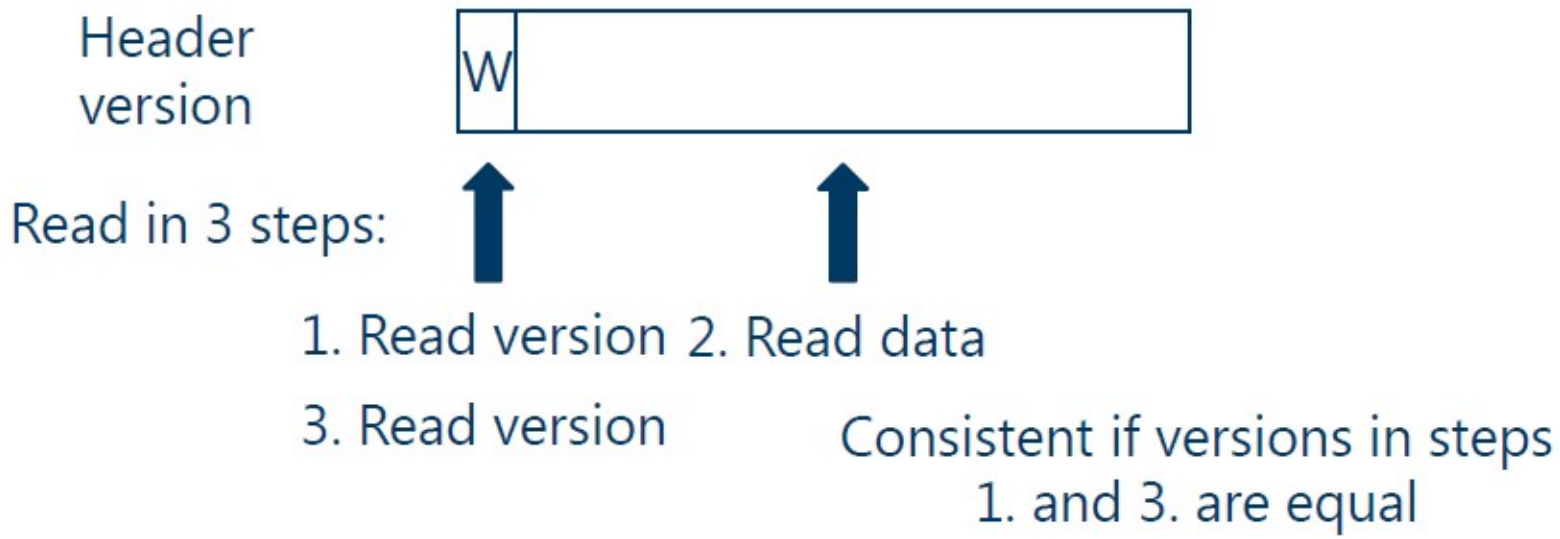
Traditional Lock-free reads

- For updating the data,



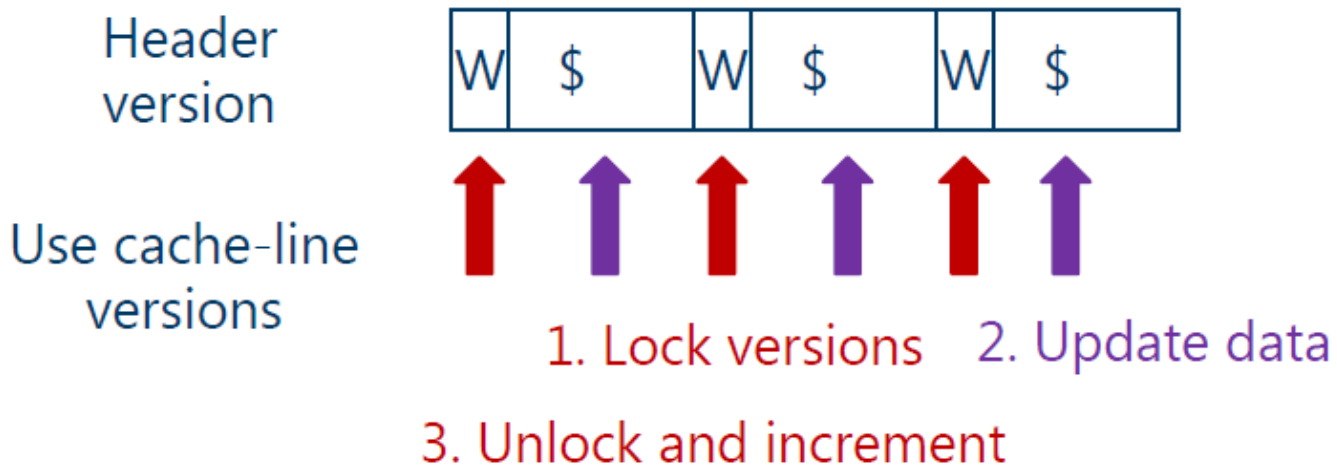
Traditional Lock-free reads

- Reading requires three accesses



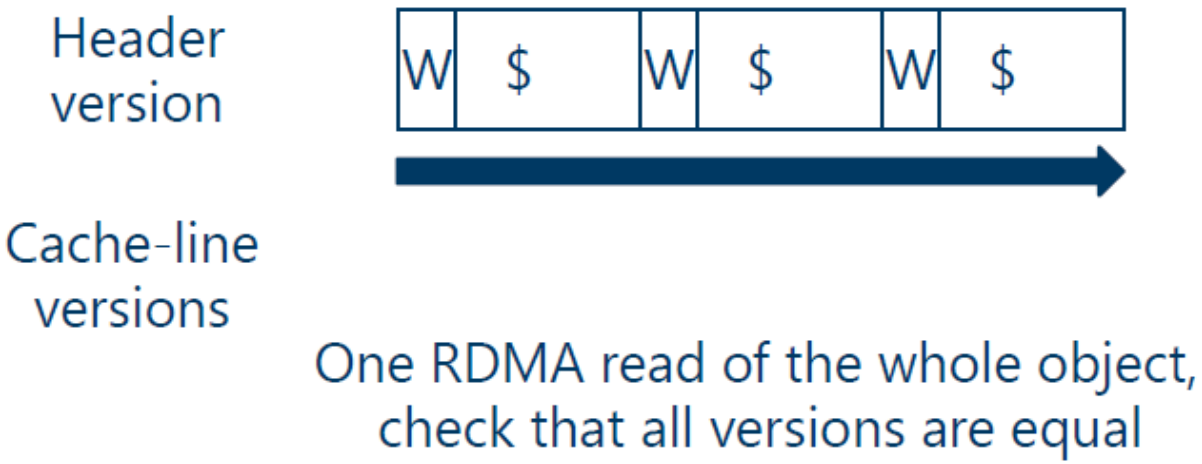
FaRM Lock-free Reads

- FaRM relies on cache coherent DMA
- Version info in each of cache-lines



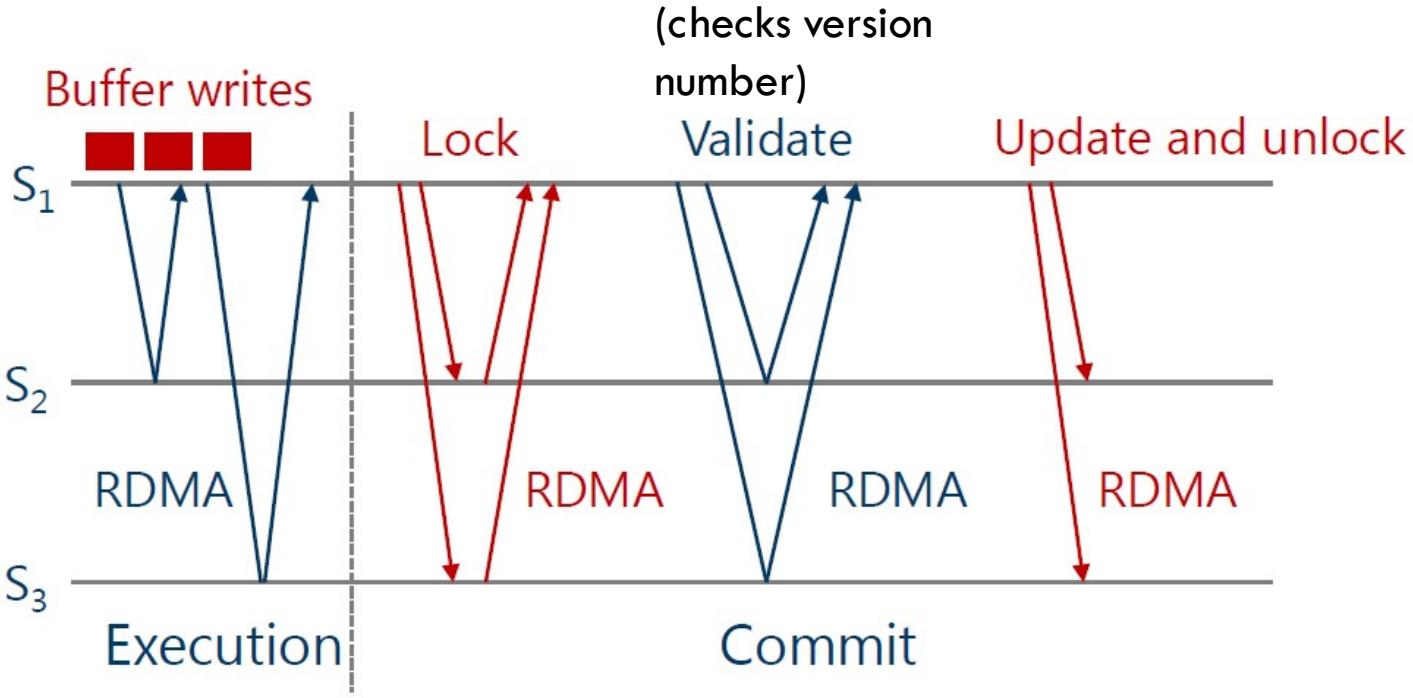
FaRM Lock-free Reads

- single RDMA read



FaRM: Distributed Transactions

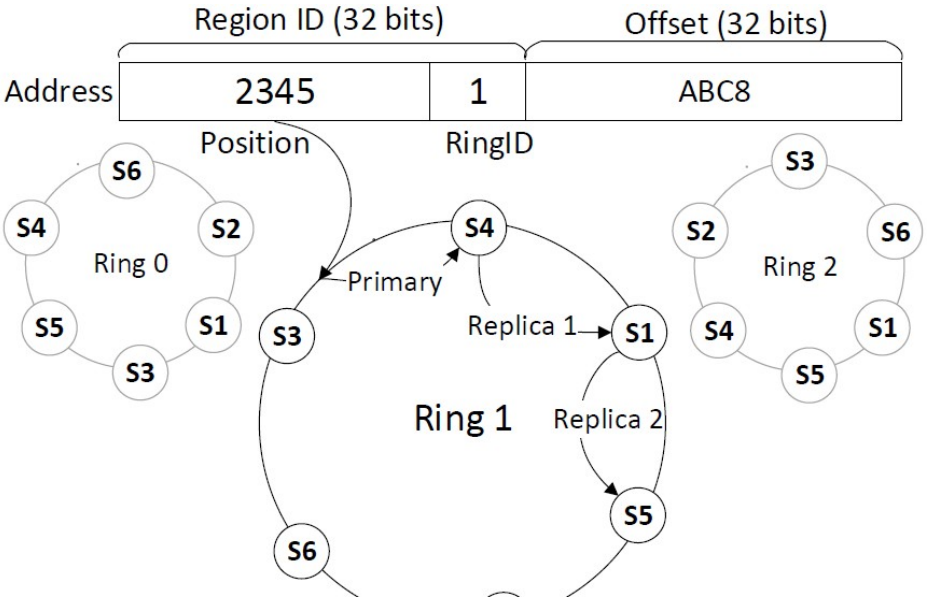
- general mechanism to ensure consistency
- Two-stage commits



Shared Address Space

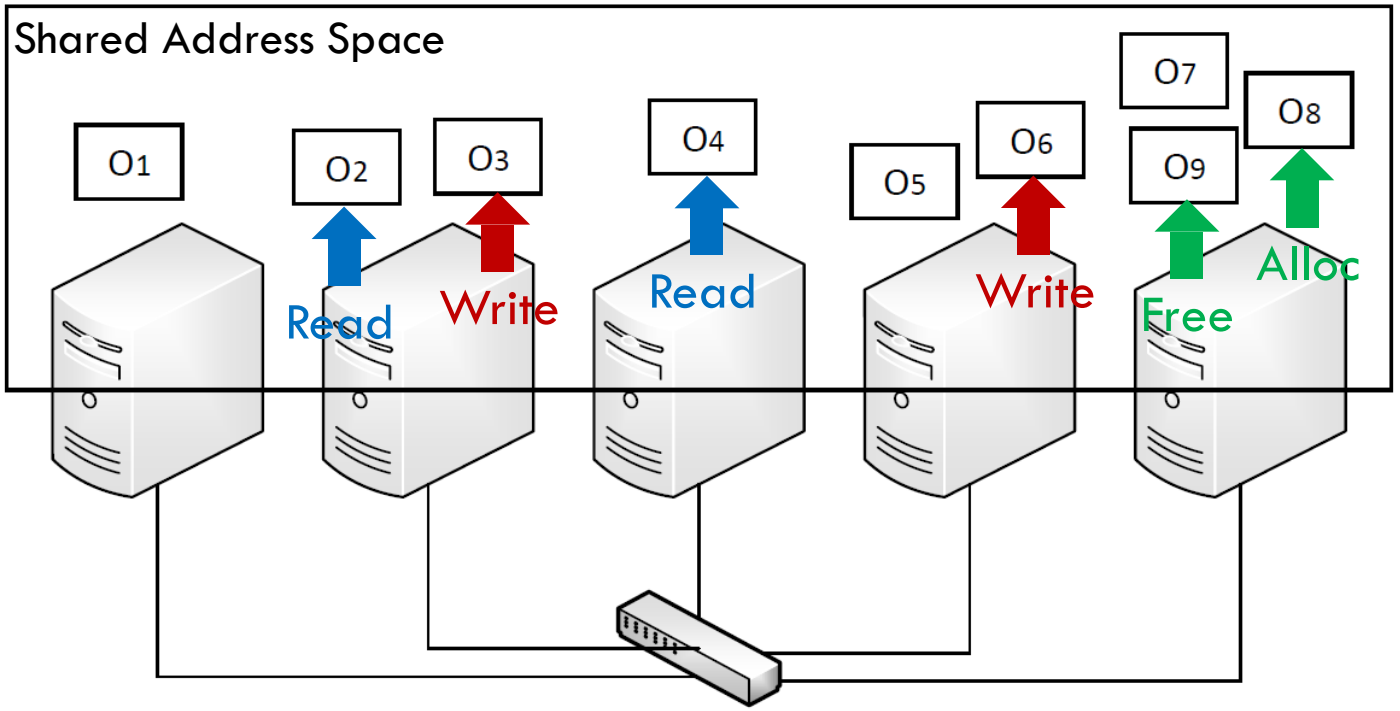
- shared address space consists of many shared memory regions
- consistent hashing for mapping region identifier to the machine that stores the object

  each machine is mapped into k virtual rings



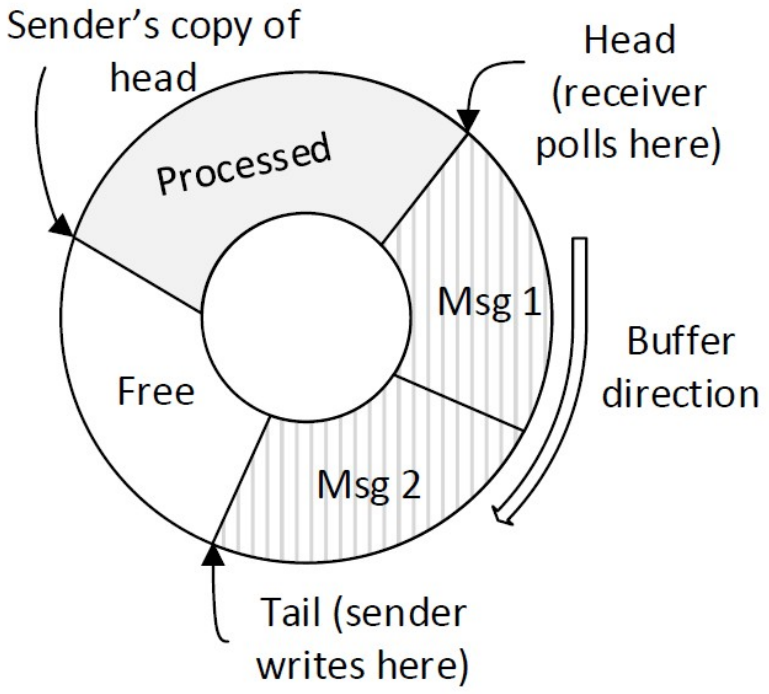
Transactions in Shared Address Space

- Strong consistency
- Atomic execution of multiple operations



Communication Primitives

- One-sided RDMA reads
  - to access data directly
- RDMA writes
  - circular buffer is used for unidirectional channel
  - one buffer for each sender/receiver pair
  - buffer is stored on receiver



benchmark on communication primitives

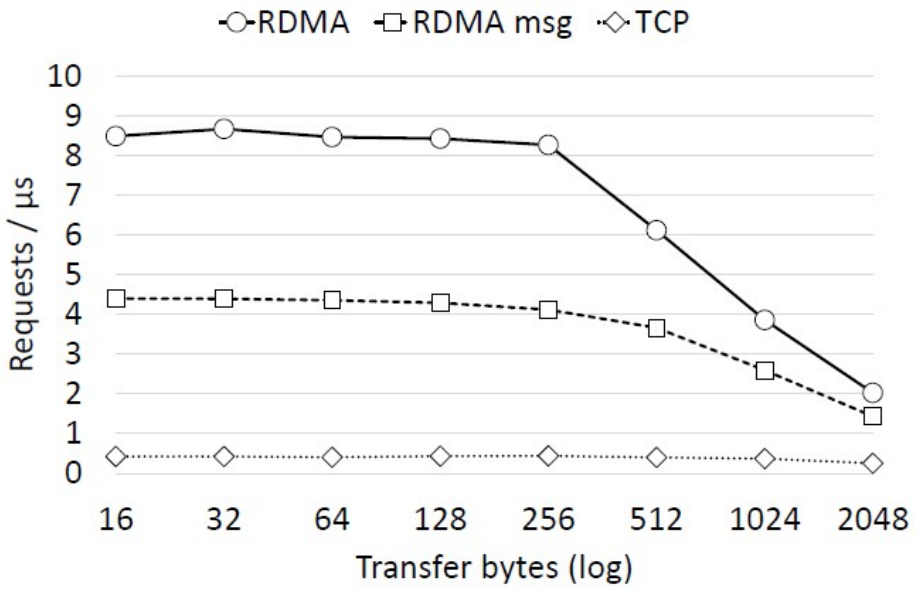


Figure 2: Random reads: request rate per machine

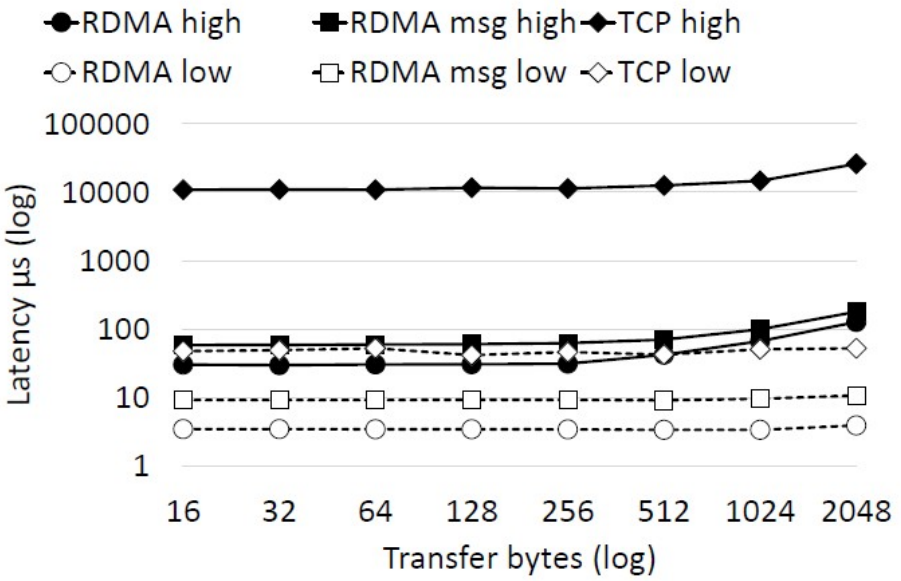
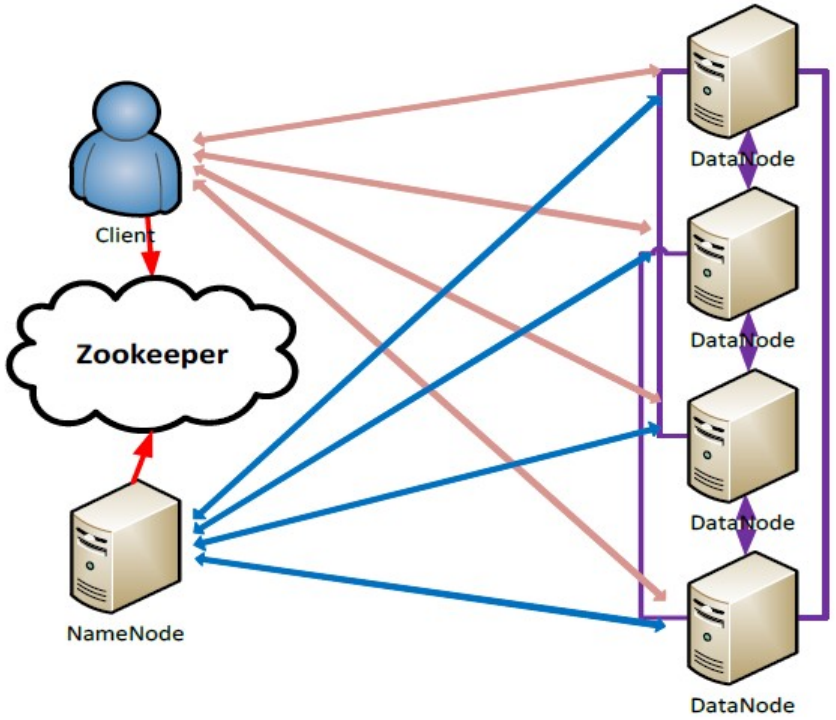


Figure 3: Random reads: latency with high and low load

Limited cache space in NIC

- Some Hadoop clusters can have hundreds and thousands of nodes
- Performance of RDMA can suffer as amount of memory registered increases
  - NIC will run out of space to cache all page tables



(a) High-level overview of HDFS architecture

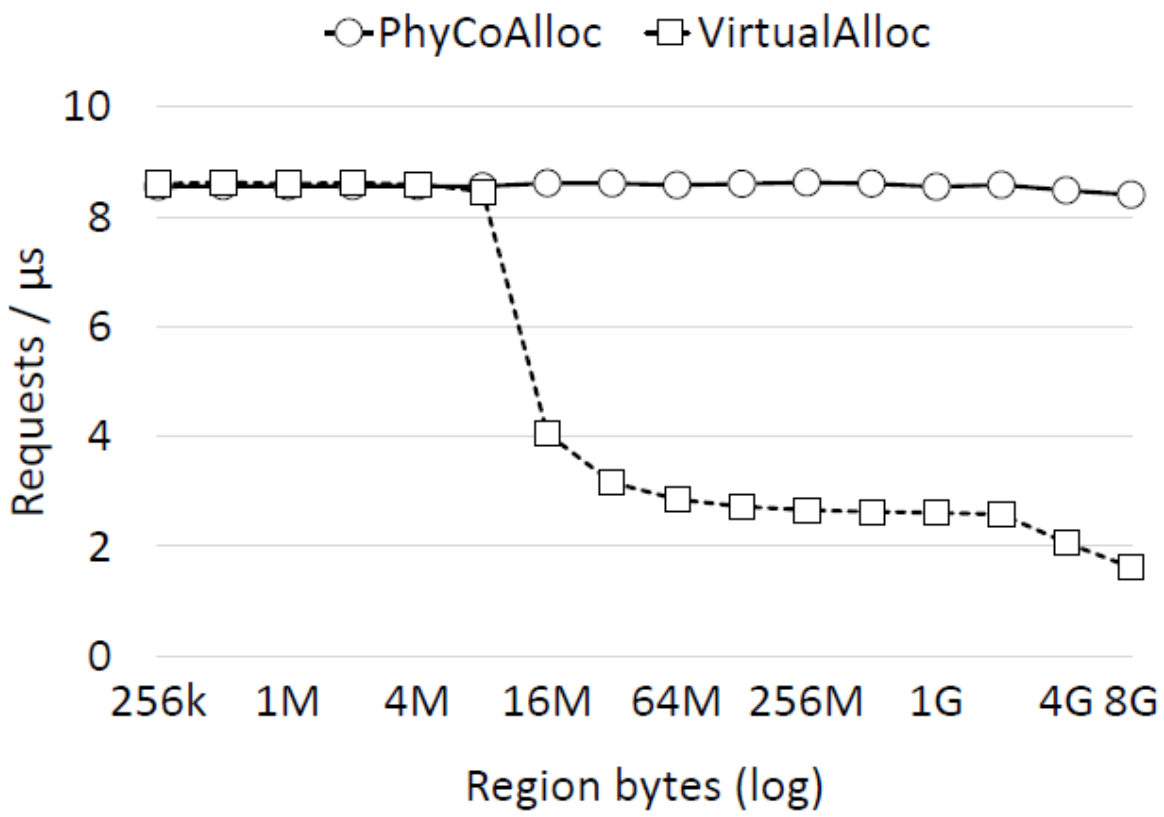
Limited cache space in NIC

- FaRM's solution: PhyCo

  kernel driver that allocates a large number of physically contiguous and naturally aligned 2GB memory regions at boot time

  maps the region into the virtual address space aligned on a 2GB boundary

PhyCo


PhyCoAlloc
VirtualAlloc
10
8
6
4
2
0
Requests / µs
256k
1M
4M
16M
64M
256M
1G
4G
8G
Region bytes (log)

Limited cache space in NIC

- PhyCo still suffered as number of clusters increased

  because it can run out of space to cache all queue pair

  - $2 \times m \times t^2$ queue pairs per machine
    - m = number of machines, t = number of threads per machine

  single connection between a thread and each remote machine

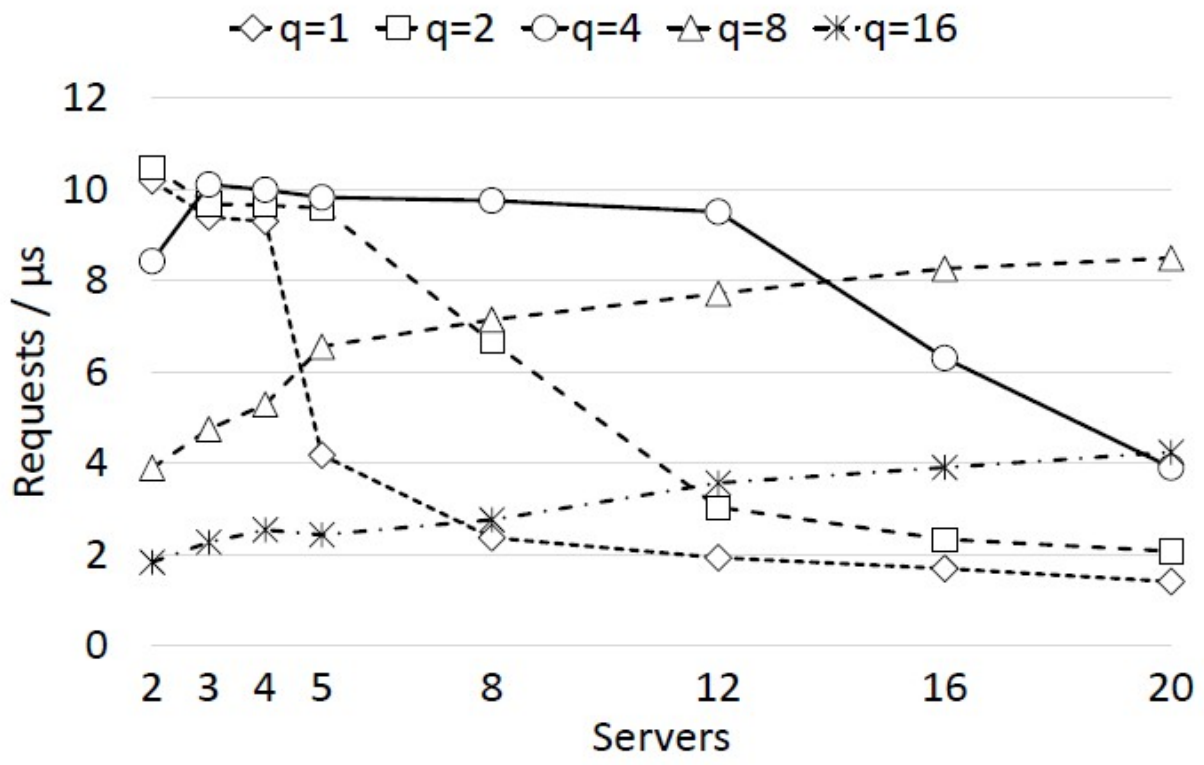
  - $2 \times m \times t$

  queue pair sharing among q threads

  - $2 \times m \times t / q$

Connection Multiplexing

- best value of q depends on cluster size



Experiments

- Key-value store: lookup scalability

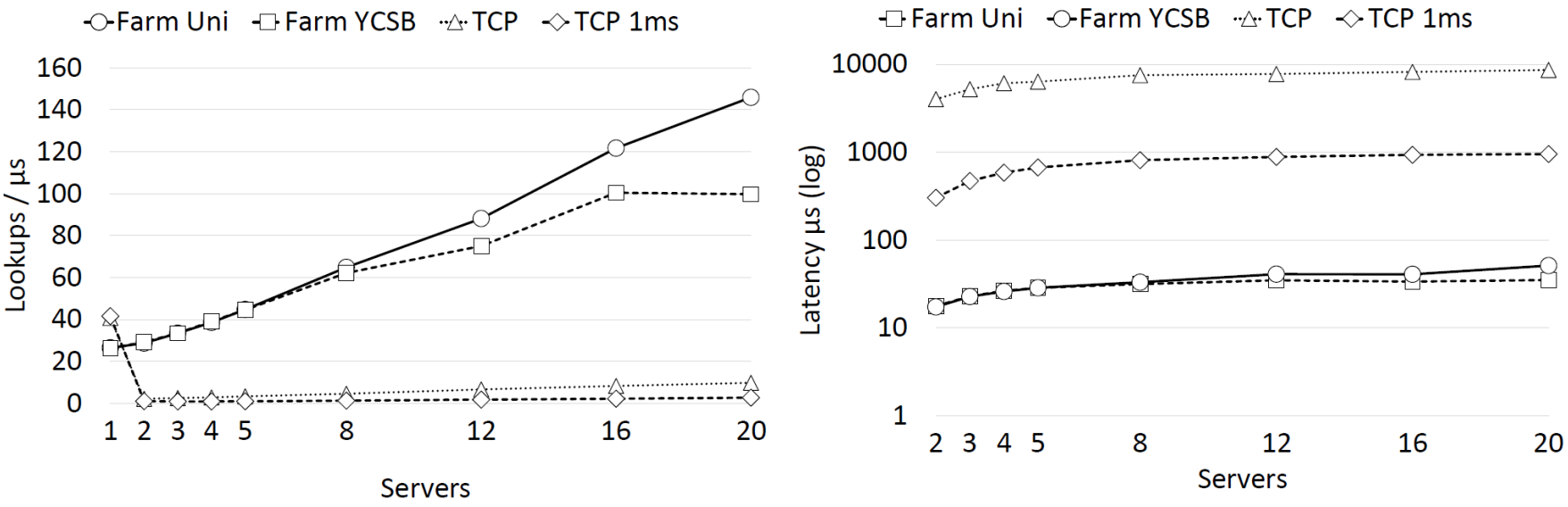
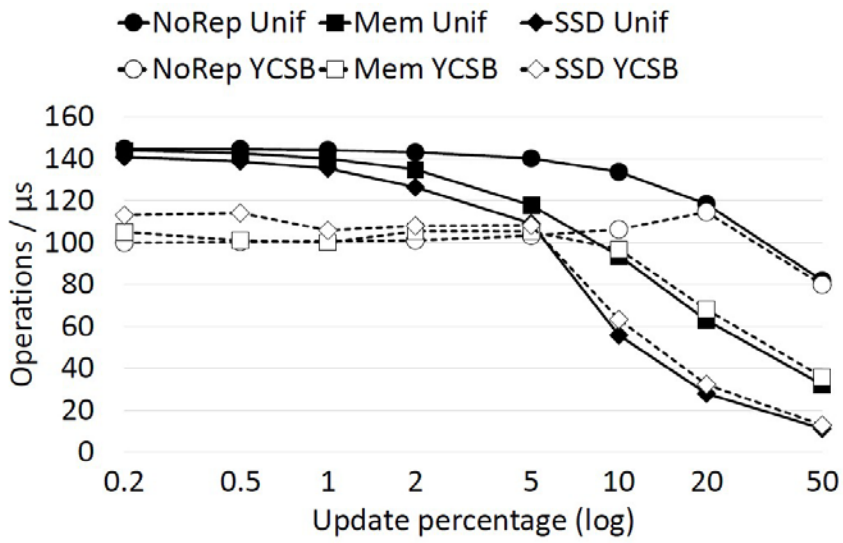


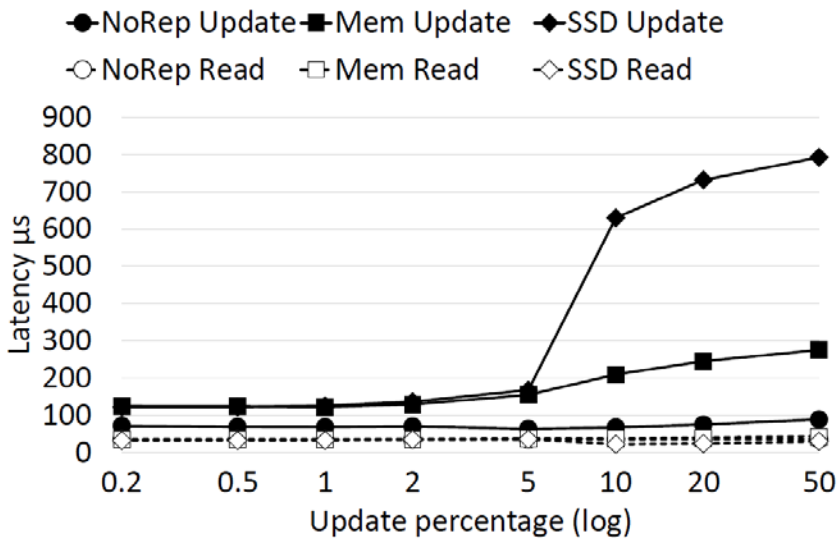
Figure 12: Key-value store: lookup scalability

Experiments

- Key-value store: varying update rates



(a) Throughput (YCSB and uniform)



(b) Latency for lookups and updates (uniform)

Figure 15: Key-value store: varying update rates