RDMA

CS6450: Distributed Systems

Lecture 18

Ryan Stutsman

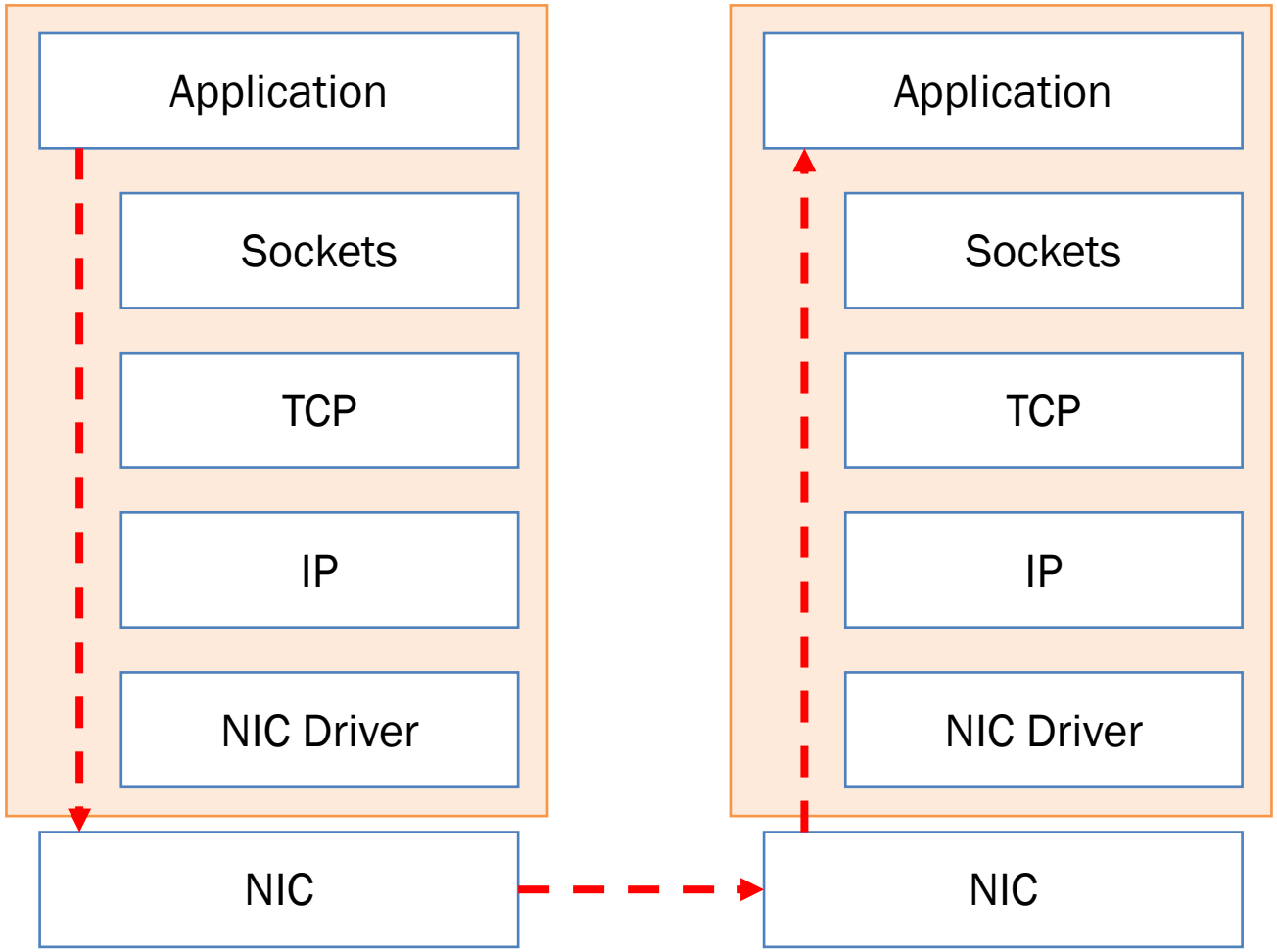
Some content adapted from Matei's NSDI talk.

Material taken/derived from Princeton COS-418 materials created by Michael Freedman and Kyle Jamieson at Princeton University.
Licensed for use under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.
Some material taken/derived from MIT 6.824 by Robert Morris, Franz Kaashoek, and Nickolai Zeldovich.

Lowering Networking Overheads

- Kernel TCP & UDP
  - RTT today between two hosts largely dominated by kernel time and abstractions
    - Inherent copying in (sockets) interface
    - Privilege boundary crossings
    - Scheduler interactions (in both directions)
      - Possible context switch, page table switch, TLB flush
    - Interrupts on receive side
      - ISR and softirq handlers – coherence traffic
      - Wake core from C-state, schedule process, more context switch
- Turns < 1 µs wire time into > 40 µs RTT
- TCP processing at > 100 Gbps uses all CPU resources
  - 15 ns of wall time per packet even at MTU

Kernel-bypass



Kernel-bypass Frameworks/APIs

- DPDK
  - Work with raw packet buffers for Ethernet frames
  - NIC control structures mapped into app
  - App “posts” buffer descriptors into tx and rx queues
  - App has to deal with frames, reorder, loss, etc.
    - Designed for “dataplanes”; Ethernet focus
    - No bells and whistles
- ibverbs
  - Work with messages
  - NIC control structures mapped into app
  - App “posts” buffer descriptors into tx and rx queues
  - Both reliable (RC) and unreliable modes
    - Designed for Infiniband; low-latency messaging focus
    - High-level API, extra features like one-sided RDMA

Fast Networks

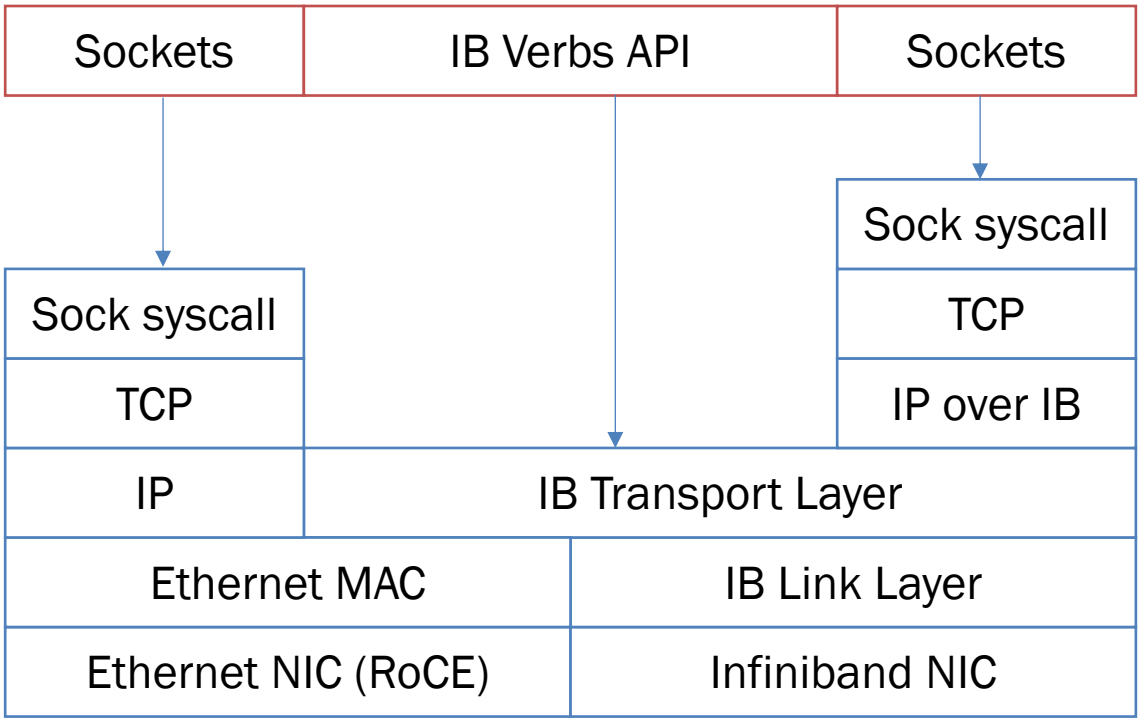
- Ethernet

Gbps	Year
10	2002
25	
40	2010
100	2010
200	2017
400	2017

- Infiniband

	Gbps	Year
SDR	8	2001
DDR	16	2005
QDR	32	2007
FDR	54	2011
EDR	100	2014
HDR	200	2017
NDR	400	2020
XDR	1000	2023?

ibverbs



Verbs usually work directly with NIC control structures in app
Different interface than sockets, but still usually “connected” client-server

Why the hype?

- Remote Direct Memory Access
  - Similar to local DMA in some respects
- Schedule NIC for remote xfers, asynchronous without remote host disruption
- Low-latency (low as µs one-sided); no kernel time
- Zero-copy; no copying in/out of buffers
- Hardware handles transport, app works at message/op level, saving more CPU
- End result: low-latency, higher-throughput at finer-message granularity

Verbs

- Two sided: send and recv
- One sided: read and write and atomics

- Unreliable Datagram mode
  - Only send and recv, and only up to 2 to 4 kB messages
  - No integrity, ordering, or delivery guarantee
  - Supports multicast
- Reliable Connected mode
  - Supports send, recv, read, write, atomics
  - At-most once on connection (ack when in-memory)
  - Up to 2 GB message size
  - Scaling issues with many QPs; problematic past 100s of QPs

High-level RC Workflow

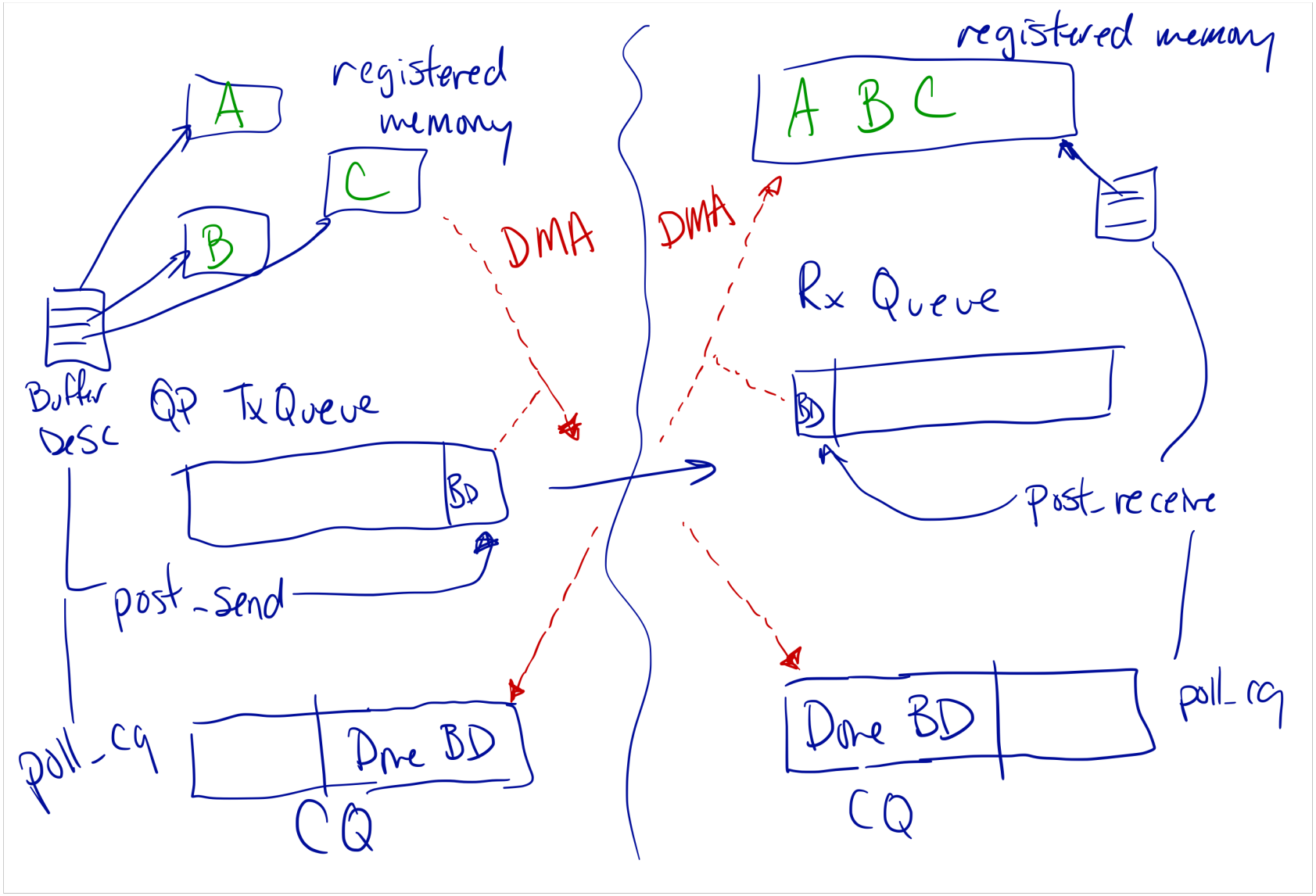
- Call kernel module to map NIC control registers
- Register regions of process address space
- Create a “completion queue”
  - NIC fills completion queue with notifications of completed “work requests”, e.g. send has completed
- Create an RC “queue pair” with the other end
  - Need to exchange a QP number, LID, …
- Receiver: post receive buffer descriptors
- Send: post send buffer descriptors
- Both: “poll” CQ to find incoming events

Memory Registration

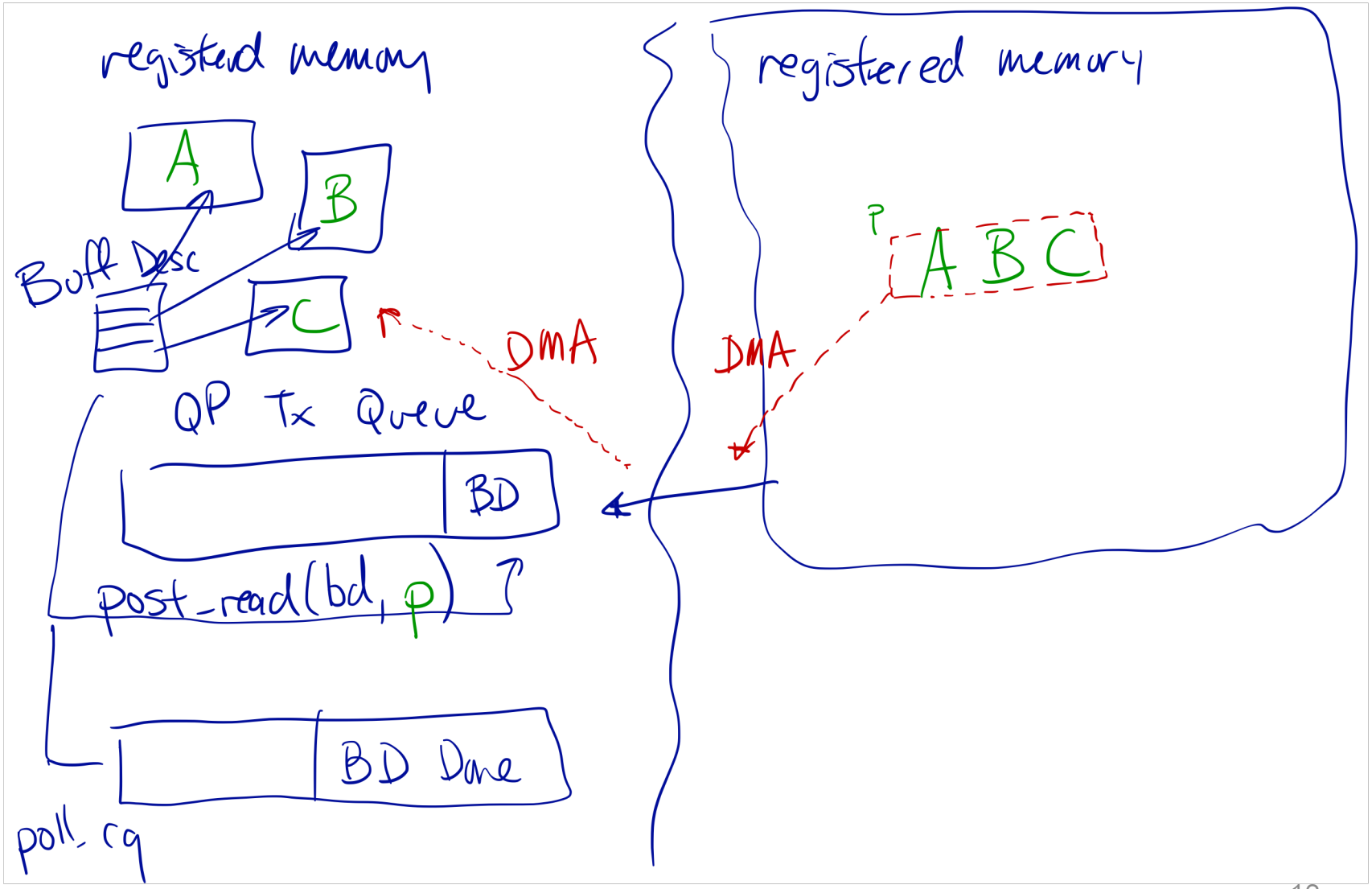
- NIC must do address translation
  - App knows virtual addresses, NIC DMAs physical
- OS cannot swap out/evict registered memory
  - If buffer posted in send/recv, NIC may be accessing it!

- Solution: register and “pin” memory
  - NIC knows VA to PA mappings for registered region
  - Implements on-card address translation
  - Scaling issues if registered region is large and physically discontinuous, so use huge pages
  - Applies to both send/recv and read/write
  - Too expensive to do this on the critical path

Two-sided "RDMA"



One-sided RDMA



One-sided vs Two-sided

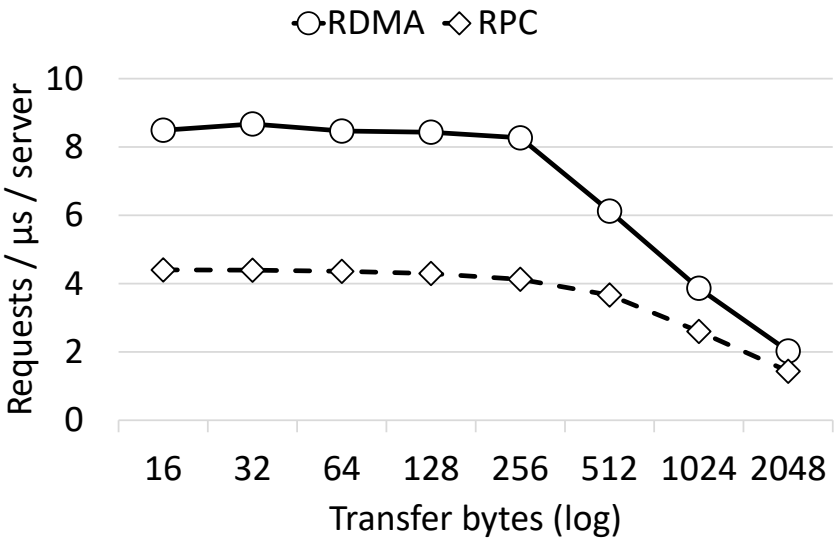
- One-sided
  - No CPU use at target
  - Messaging scales at NIC, then scales indep. of CPU
  - Constrained semantics

- Two-sided
  - Busy-wait plus CQ/RX interaction costs
  - Hw DMA/transport still offloads most of work
  - Recvr is active, so can implement RPC-like ops

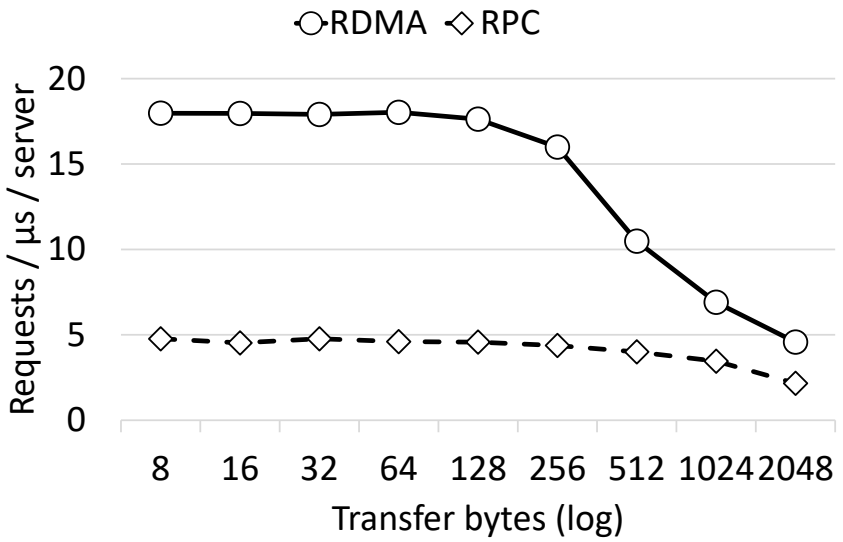
Key Question

- Use RDMA reads to offload the target?
  - Or implement richer functionality at target and reduce number of ops that it needs to handle?
  - Different systems have taken different approaches
- Considerations beyond CPU scaling?

RDMA Reads vs RPC



(a) 1 NIC (network-bound)



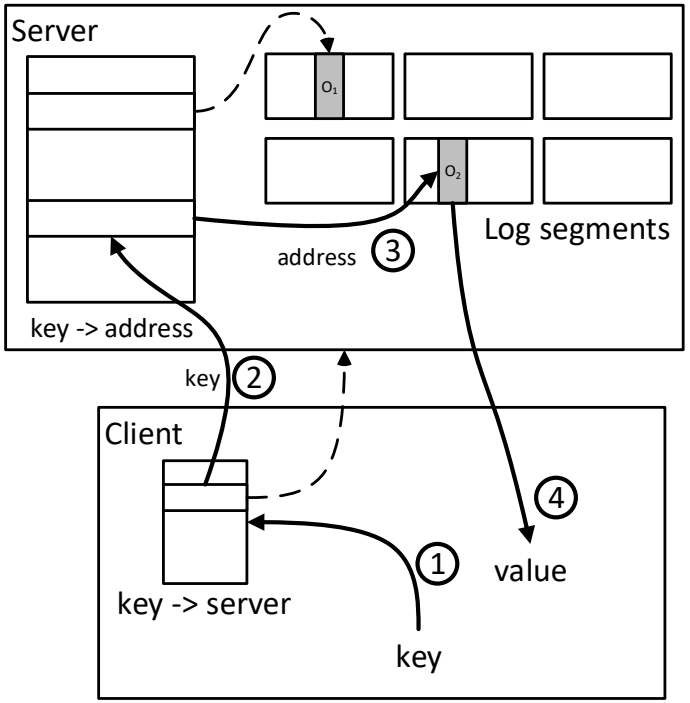
(b) 2 NICs (CPU-bound)

Tight client-coupling challenges

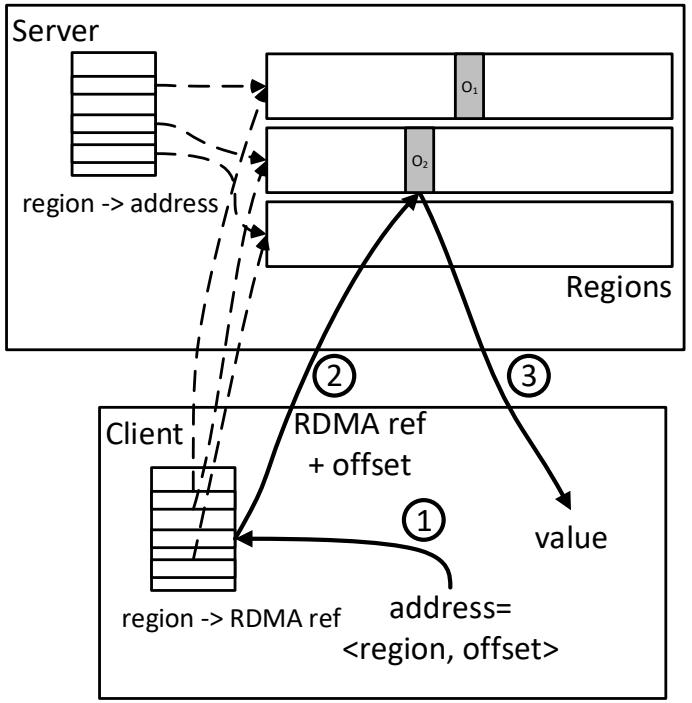
- Key issue: lack of indirection, lack of single central canonical reference

- Look up by key?
- Variable length objects?
- Compacting live space?
  - How does the system determine if an object has a reference somewhere?
- Fine-grained data migration and rebalancing?
- Eviction of data to cold tier?

RAMCloud vs FaRM



(a) RAMCloud



(b) FaRM

Questions

- RDMA-based B-tree?
- RDMA-based MVCC?
- Are real in-memory data center systems CPU bound?