Advanced Network Programming (ANP)
XB_0048

RDMA Networking

Animesh Trivedi
Autumn 2020, Period 1

VU VRIJE UNIVERSITEIT AMSTERDAM

Layout of upcoming lectures - Part 1

Sep 1st, 2020 (today): *~~Introduction and networking concepts~~*

Sep 3rd, 2020 (this Tuesday): *~~Networking concepts (continued)~~*

Sep 8th, 2020 : *~~Linux networking internals~~*

Sep 10th 2020: *~~Multicore scalability~~*

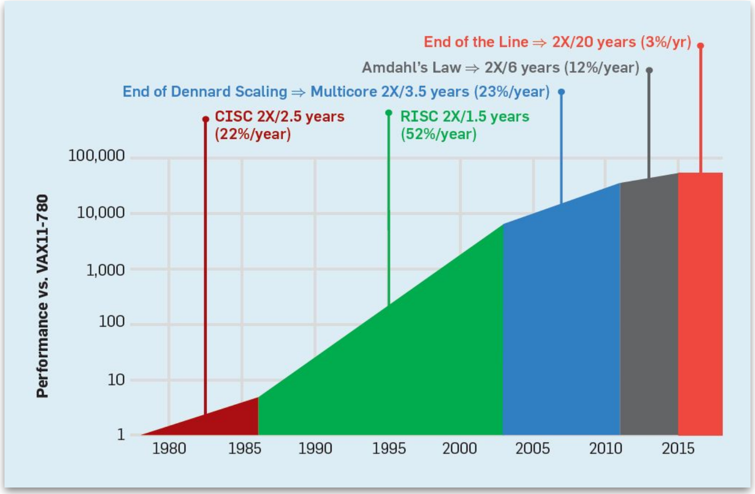
Sep 15th 2020: *~~Userspace networking stacks~~*

Packet processing
Netmap
mTCP
Userspace networking

Sep 17th 2020: *Introduction to RDMA networking*

What is the setup?

- Everything runs on the CPU
  - Applications, threads, processes
  - the operating system kernel
- **But the CPU is not getting any faster**
  - CPU was getting faster due to Moore's Law
- **But the network speeds are…**
  - 1 to 10, and now 100 Gbps
  - 200 and 400 Gbps are now available
  - **Be careful -** we are focussing on a closely installed datacenter setup



Hennessy and Patterson, A New Golden Age for Computer Architecture (Turing Award Lecture), https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

In the previous lectures

We have seen how to

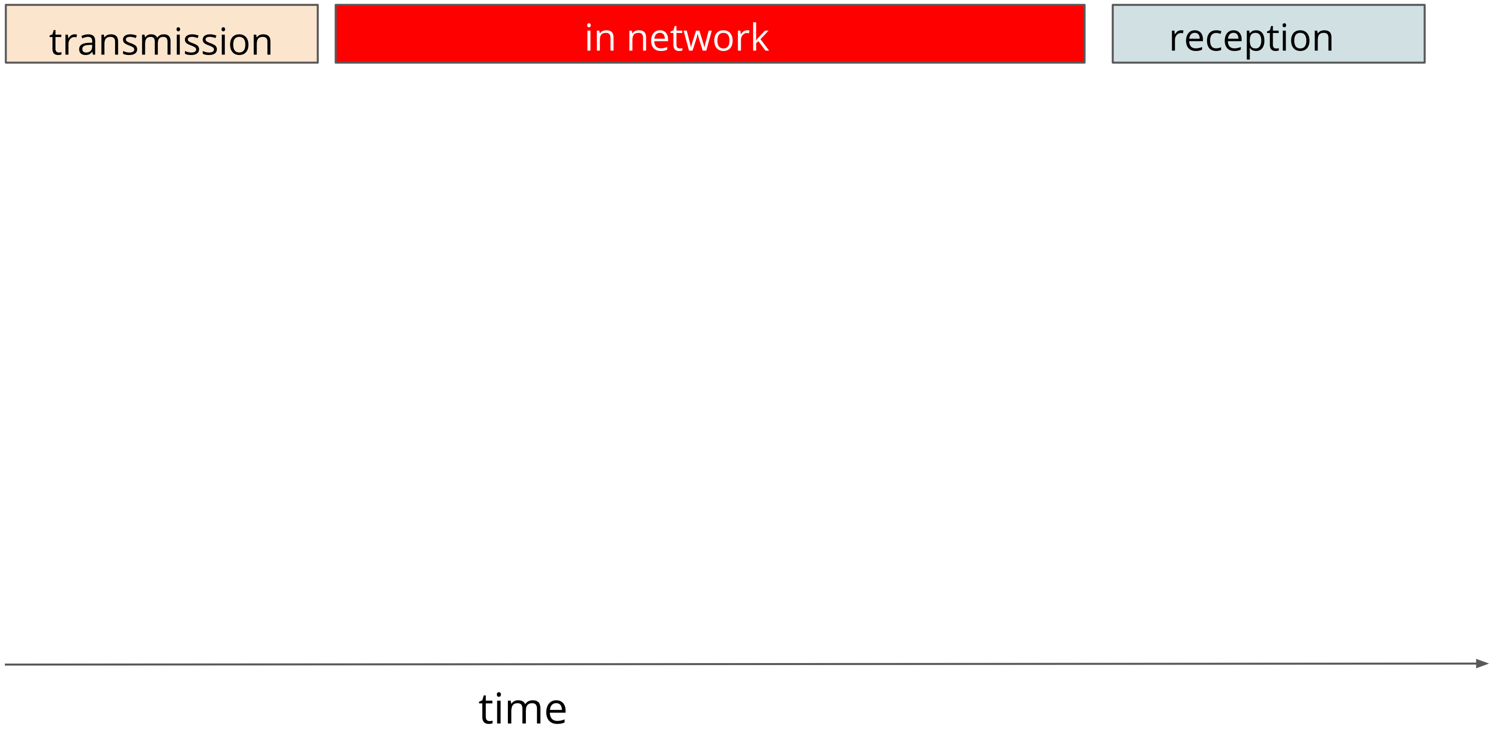
- Increase *bandwidth* of a single connection
  - TSO, LRO, GRO, Jumbo packets

- Utilize multicore systems - *packets / sec*
  - RSS, interrupt load balancing, connection state partitioning

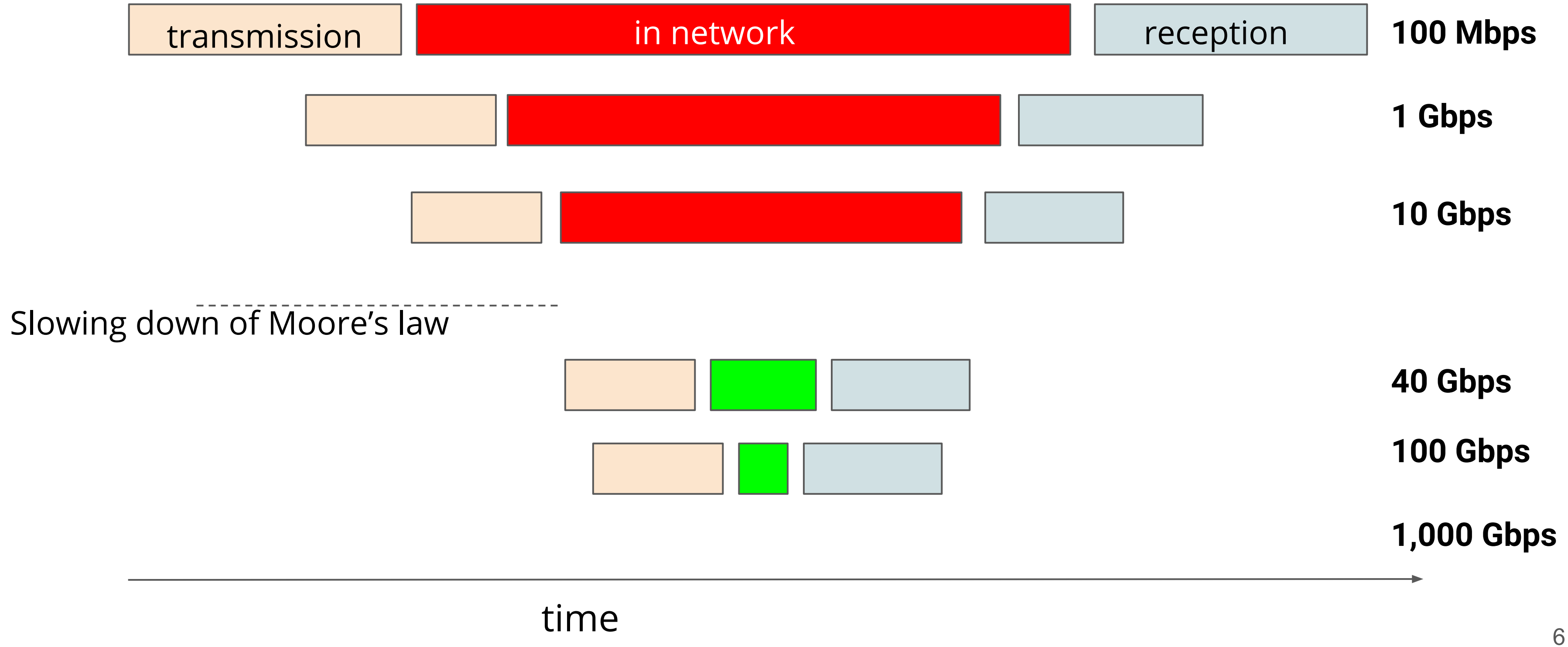
- Drive short, short-lived connections - *packets / sec*
  - Userspace networking, integrated processing, directly mapped queues, polling

How about latencies? How low can we go for accessing 1 byte, 4 bytes, 8 bytes?

Shifting performance bottlenecks

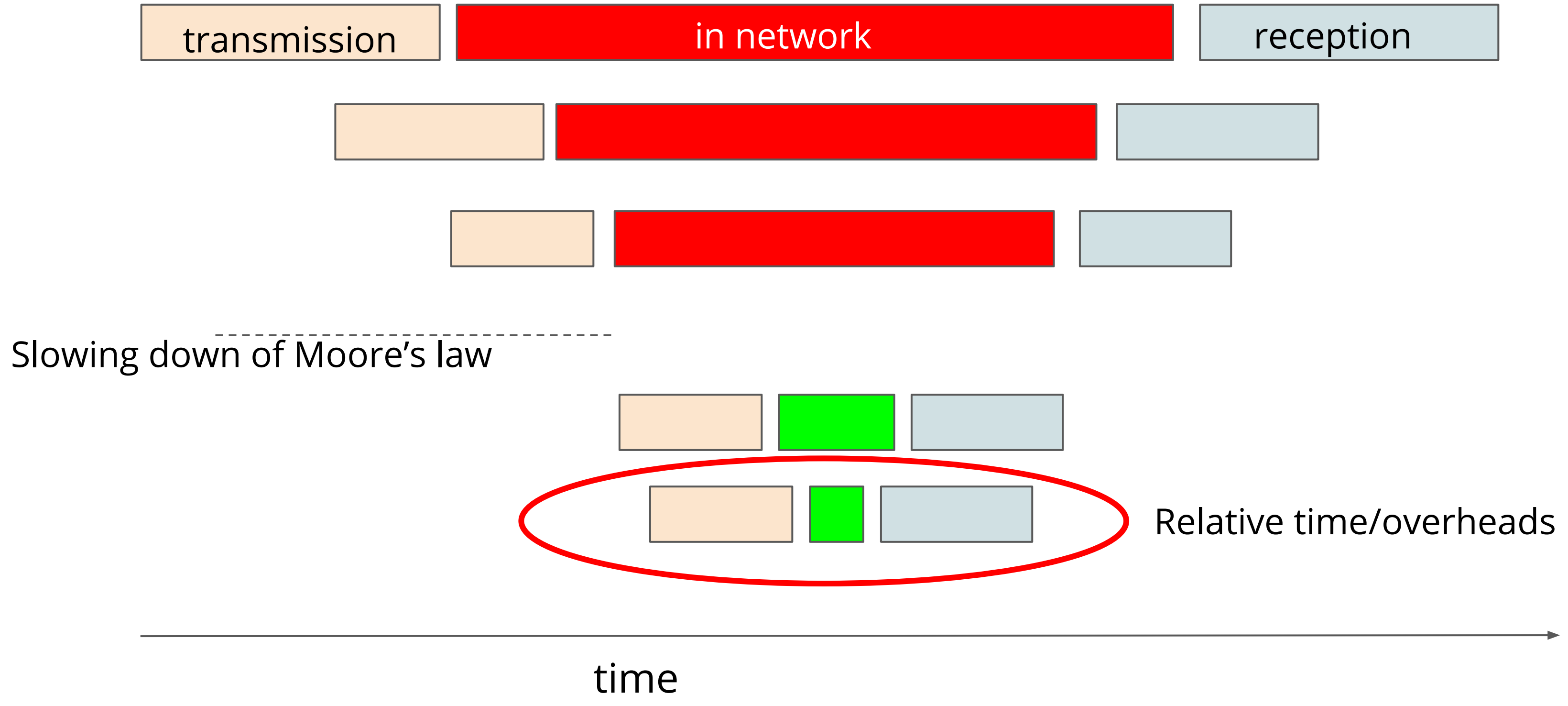


Shifting performance bottlenecks



Not to scale

Shifting performance bottlenecks

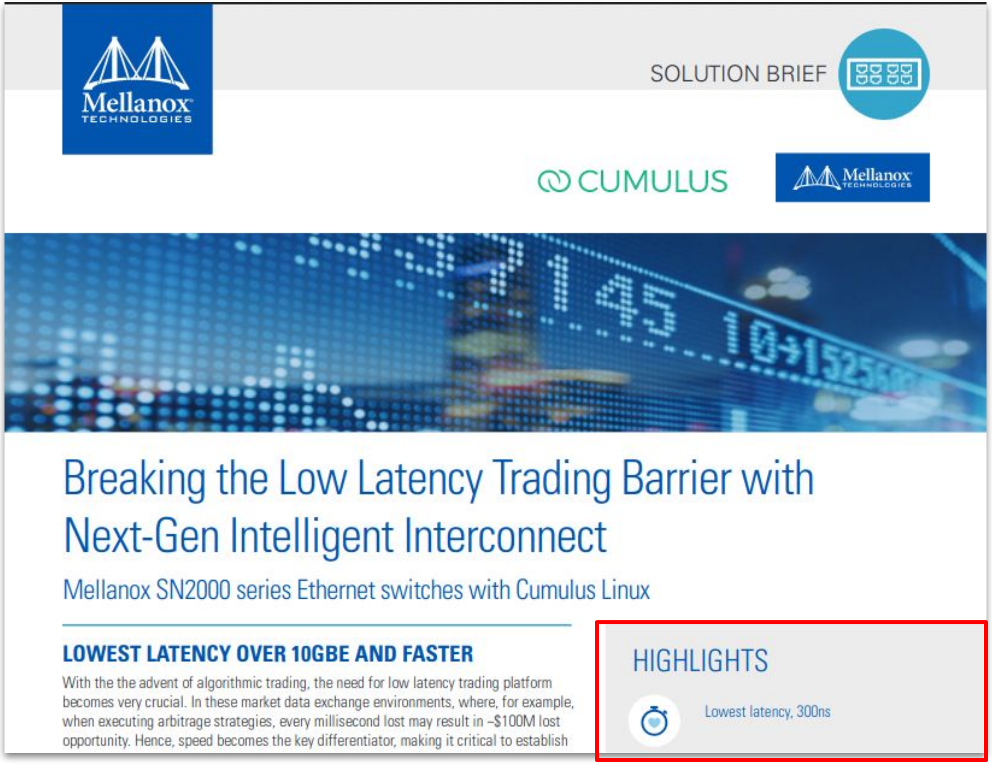


Trends in interconnect latencies

	ConnectX-4 EDR 100G*	Connect-IB FDR 56G	ConnectX-3 Pro FDR 56G
InfiniBand Throughput	100 Gb/s	54.24 Gb/s	51.1 Gb/s
InfiniBand Bi-Directional Throughput	195 Gb/s	107.64 Gb/s	98.4 Gb/s
InfiniBand Latency	0.61 us	0.63 us	0.64 us
InfiniBand Message Rate	149.5 Million/sec	105 Million/sec	35.9 Million/sec
MPI Bi-Directional Throughput	193.1 Gb/s	112.7 Gb/s	102.1 Gb/s

Historically, Ethernet has lagged a bit behind InfiniBand (1-2 generation) but usually catches up

Trends in interconnect latencies: Switches


SOLUTION BRIEF

Breaking the Low Latency Trading Barrier with Next-Gen Intelligent Interconnect

Mellanox SN2000 series Ethernet switches with Cumulus Linux

LOWEST LATENCY OVER 10GBE AND FASTER

With the the advent of algorithmic trading, the need for low latency trading platform becomes very crucial. In these market data exchange environments, where, for example, when executing arbitrage strategies, every millisecond lost may result in ~$100M lost opportunity. Hence, speed becomes the key differentiator, making it critical to establish

HIGHLIGHTS

Lowest latency, 300ns

Ethernet switches has typically a bit higher but still 100s of nanoseconds

300 nanoseconds

https://www.mellanox.com/related-docs/solutions/fsi/SB-breaking-the-low-latency-trading-barrier-with-next-gen-intelligent-interconnect.pdf

The year is 2011

It's Time for Low Latency

Stephen M. Rumble, Diego Ongaro, Ryan Stutsman,
Mendel Rosenblum, and John K. Ousterhout
Stanford University

Abstract

The operating systems community has ignored network latency for too long. In the past, speed-of-light delays in wide area networks and unoptimized network hardware have made sub-100μs round-trip times impossible. However, in the next few years datacenters will be deployed with low-latency Ethernet. Without the burden of propagation delays in the datacenter campus and network delays in the Ethernet devices, it will be up to us to finish the job and see this benefit through to applications. We argue that OS researchers must lead the charge in rearchitecting systems to push the boundaries of low-latency datacenter communication. 5-10μs remote procedure calls are possible in the short term – two orders of magnitude better than today. In the long term, moving the network interface on to the CPU core will make 1μs times feasible.

1 Introduction

Network latency has been an increasing source of frustration and disappointment over the last thirty years. While nearly every other metric of computer performance has improved drastically, the latency of network communication has not. System designers have consistently chosen to sacrifice latency in favor of other goals such as bandwidth, and software developers have focused their efforts

	1983	2011	Improved
CPU Speed	1x10Mhz	4x3GHz	> 1,000x
Memory Size	≤ 2MB	8GB	≥ 4,000x
Disk Capacity	≤ 30MB	2TB	> 60,000x
Net Bwidth	3Mbps	10Gbps	> 3,000x
RTT	2.54ms	80μs	32x

Table 1: Network latency has improved far more slowly over the last three decades than other performance metrics for commodity computers. The V Distributed System [5] achieved round-trip RPC times of 2.54ms. Today, a pair of modern Linux servers require 80μs for 16-byte RPCs over TCP with 10Gb Ethernet.

ments will also be required to reach this goal, the operating system community is in the best position to coordinate all of these changes and create the right end-to-end architecture. Over the longer-term, and with more radical hardware changes (such as moving the NIC onto the CPU chip), we think 1μs datacenter round-trips can be achieved.

There will be many benefits for datacenter computing if we succeed. Lower latency will simplify application development, increase web application scalability, and enable new kinds of data-intensive applications that are not possible today.

Really fun read!

Back then general commodity Ethernet was at 10 Gbps, and around **100s microseconds** for latencies (inside a data center)

Single CPU speed were stalled

Big data was booming

They came up with an ambitious goal of what would it take to deliver:

One microsecond of Req-Resp network operation (RPC)

Fun things to consider with latencies

Speed of light

- Copper vs optical cable - reflection, refraction, conversion to electrical signals
- *5 nanosecond/meter*, so a RTT of 1 useconds ~ 200 meter of installation distance
- ~14 meters x 14 meters of area (diagonal distance of 200 meter)

Density of computation, power delivery, cooling

- How many servers, switches, GPUs, CPUs, etc. can you pack here
- Electricity delivery, cooling, wiring

Computation diameter

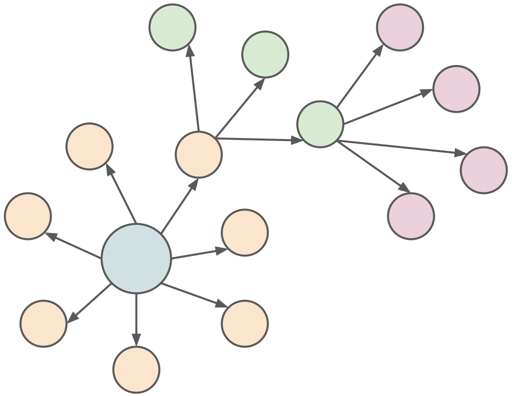
- Depending upon the number of machines, distance, there is only a certain numbers of machines you can contact in 1 microsecond

Amount of data you can access

- Bandwidths, machines, packet rates determines the data you can process

Is Latency an important factor?

- The fan-in and fan-out effect
  - Many web-scale, inside data center workloads touch hundreds of machines for processing a request
  - Facebook ~130 requests, Amazon 100-200 request
    - *How many servers would you contact within 1 sec?*
  - Latency critical processing at each step



- Latency sensitive workloads
  - Big data processing, graph processing, streaming
  - Hundreds/Thousands of servers needs to coordinate and work in unison to solve a problem
- Scientific workloads
  - High performance computing: Weather simulation, genomics, personalized medicine, computational chemistry
  - Small calculations + data access

The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM , https://web.stanford.edu/~ouster/cgi-bin/papers/ramcloud.pdf

How bad it might be?

	1980s	mid-2010s	Today
Bandwidth	1 Mbps	10 Gbps	100/200 Gbps
Latency	~2.5ms	~50 - 100μsec	1 - 2μsec
CPU	10 MHz	(2 - 4) x 3 GHz	n x 3 GHz

Trend 1: Network hardware is getting faster

Trend 2: CPUs are not (but network parallelization is hard)

"Network latencies are *increasingly* a software/CPU factor"

It's Time for Low Latency, http://www.scs.stanford.edu/~rumble/papers/latency_hotos11.pdf

Breakdown of a single server cost

Layer	Component	Time [µs]
Kernel	Driver RX	0.60
	Ethernet & IPv4 RX	0.19
	TCP RX	0.53
	Socket enqueue	0.06
Application	`epoll_wait()` syscall	0.15
	`read()` syscall	0.33
	Generate "OK" reply	0.48
	`write()` syscall	0.22
Kernel	TCP TX	0.70
	IPv4 & Ethernet TX	0.06
	Driver TX	0.43
Total		3.75

Setup a simple ping-pong (request-reply) setup

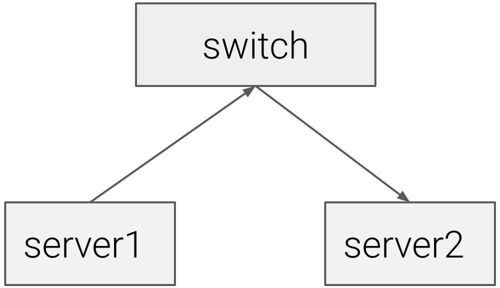
- 3.75 for server crossing
  - This is actually quite good already
- Typically scheduler will inject another 2-3 useconds of latencies
  - *Why scheduler is necessary?*

We could be seeing something between 3-6 useconds for a server crossing (in-out).

StackMap: Low-Latency Networking with the OS Stack and Dedicated NICs, https://www.usenix.org/system/files/conference/atc16/atc16-paper_yasukata.pdf

How Bad Does it Look?

Component	Delay
Switch	~1 µsec
NIC	~1 µsec
OS processing	3.4 - 6.2 µsec
Speed of light	5 nsec/meter



Delay calculation (one way):

(1µsec x 1 switch) + (1µsec x 2 NICs) + (2 OSes x *4.8 µsec*) + (2 meters * 5 nsecs) = **12.61 µsec**

(out of which 76.1% is OS/software cost)

Peter et al., Arrakis: The Operating System is the Control Plane (USENIX OSDI 2016)

Rumble et al., It's Time for Low Latency, (USENIX HotOS 2011)

What does socket contribute in it

- Socket is an application-level interface
    - It does not say anything about lower layer protocols
- However, its simplistic design restricts many optimization opportunities to reduce the amount of work done for a network operation:
    - Tied to the OS "**process**" abstraction with the file address space
        - Everything (i.e. the socket file) belongs to a process - multiplexing, security, isolation
    - When to do copy, when to do DMA - OS must decide on behalf of processes
    - Is the "file byte-stream" the right interface than "messages"
    - No control over when network I/O happens, ordering, and notification
- Hard API for the integration of any hardware help for applications
    - Sending/receiver buffers are told at the very last minute by the software
    - *There can not be any active network traffic without an application calling send/recv - application is involved*

Remote Direct Memory Access (RDMA)

RDMA: What is it?

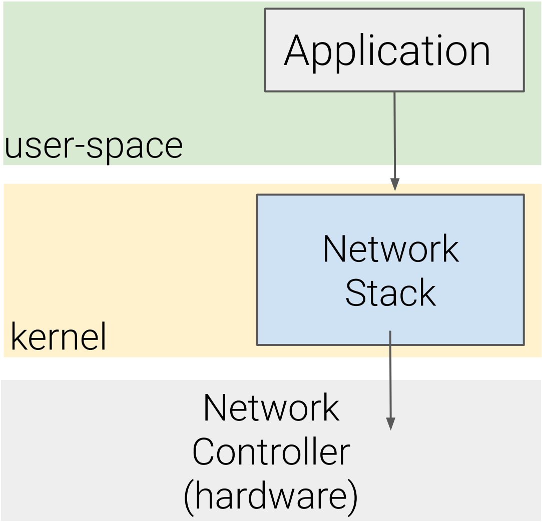
- It is a networking “*technology*” to enable high-performance, low-latency network operations

- **It is not socket** - has its own programming API and abstractions
    - We have seen so far: there is a difference between the network protocol (e.g., TCP) and the programming API (e.g., sockets, mTCP, MegaPipe)

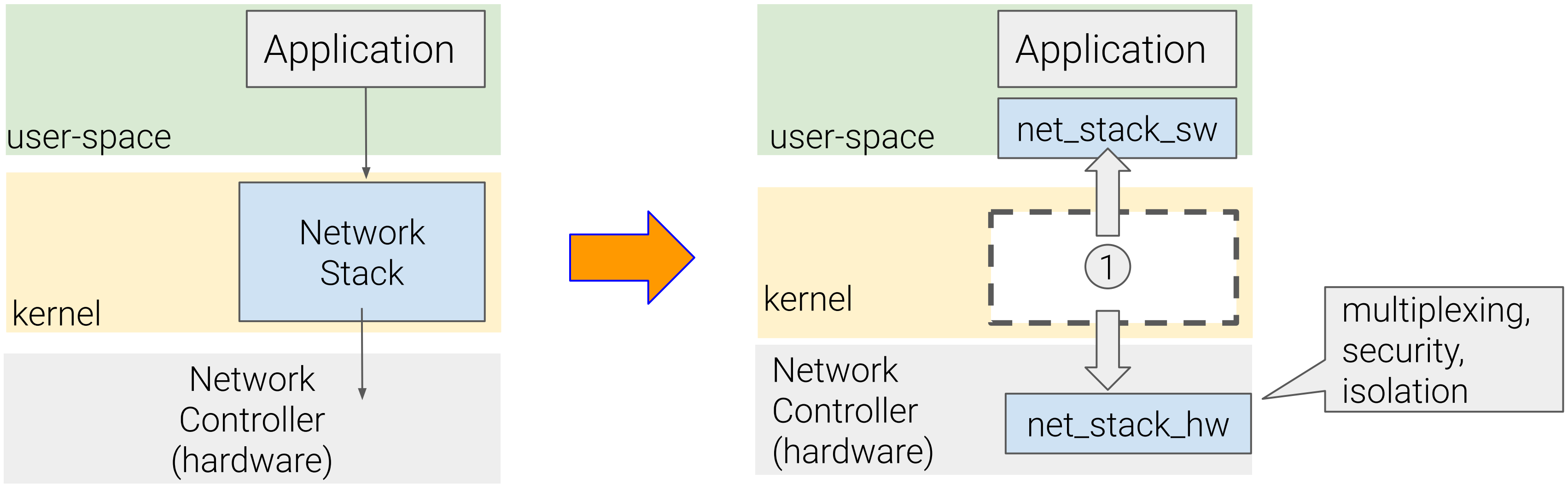
- Very successful, has a long history with research in Supercomputers
    - The goal: make network operations ~= local compute operations

- Since early ~2010, when a need for radical improvements in latency were needed, mainstream computing picked it up
    - Data centers were expanding, data was increasing, and we needed low latencies

Idea 1: User-space Networking

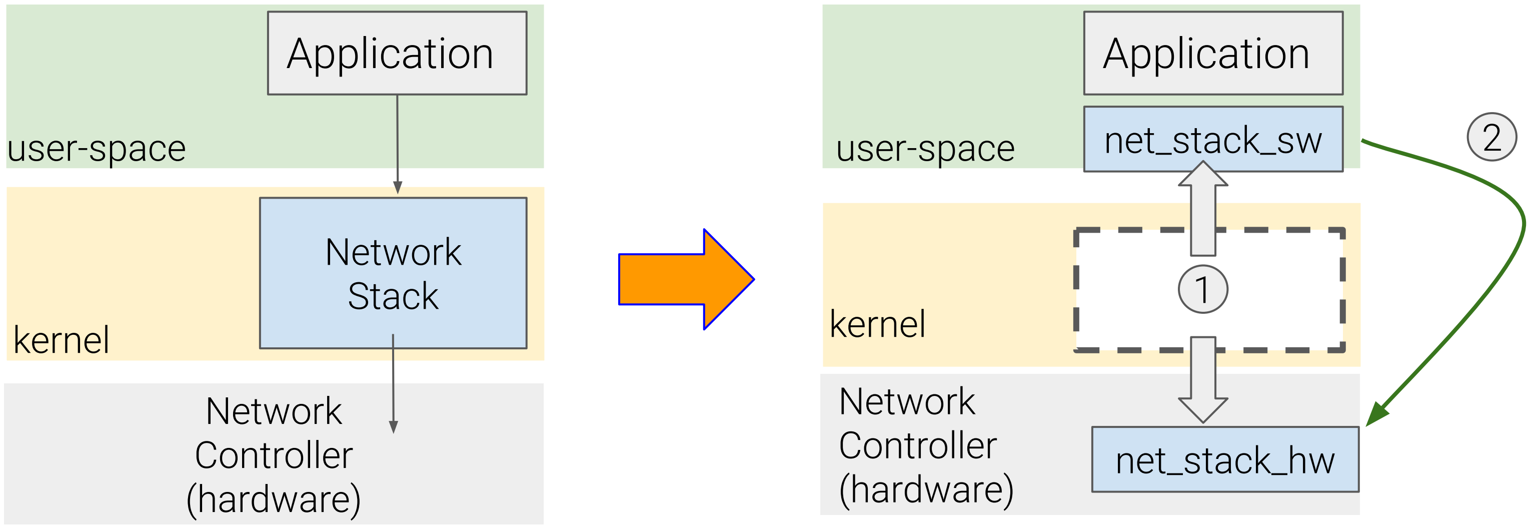


Idea 1: User-space Networking



1 **User-space networking** : Let the process manage its networking resources

Idea 2: Kernel Bypass

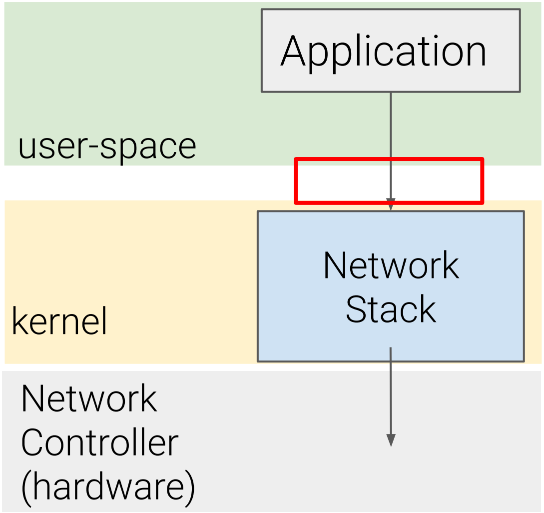


1 **User-space networking** : Let the process manage its networking resources

2 **Kernel Bypass** : Access hardware/NIC resources directly from the user-space

New Abstraction?

What is the right abstraction? Socket?

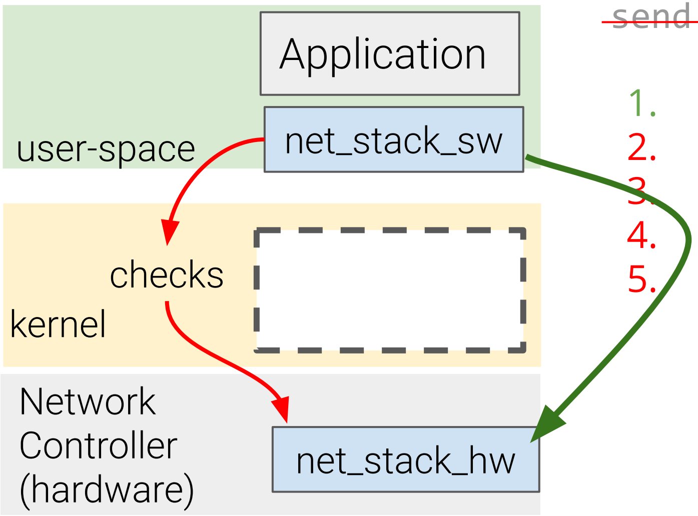



```
send (int sockfd, void *buf, size_t len, int flags);
```

1. transmit the data
2. allocate needed memory
3. manage buffers
4. schedule process (if necessary)
5. and everything else ...

New Abstraction?

What is the right abstraction? Socket?



~~`send (int sockfd, void *buf, size_t len, int flags);`~~

1. transmit the data
2. allocate needed memory
3. manage buffers
4. schedule process (if necessary)
5. and everything else ...

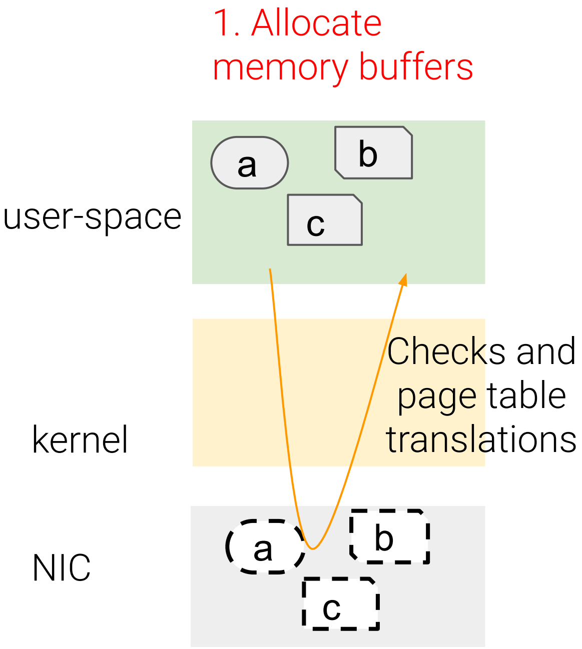
Data operation

Control operations

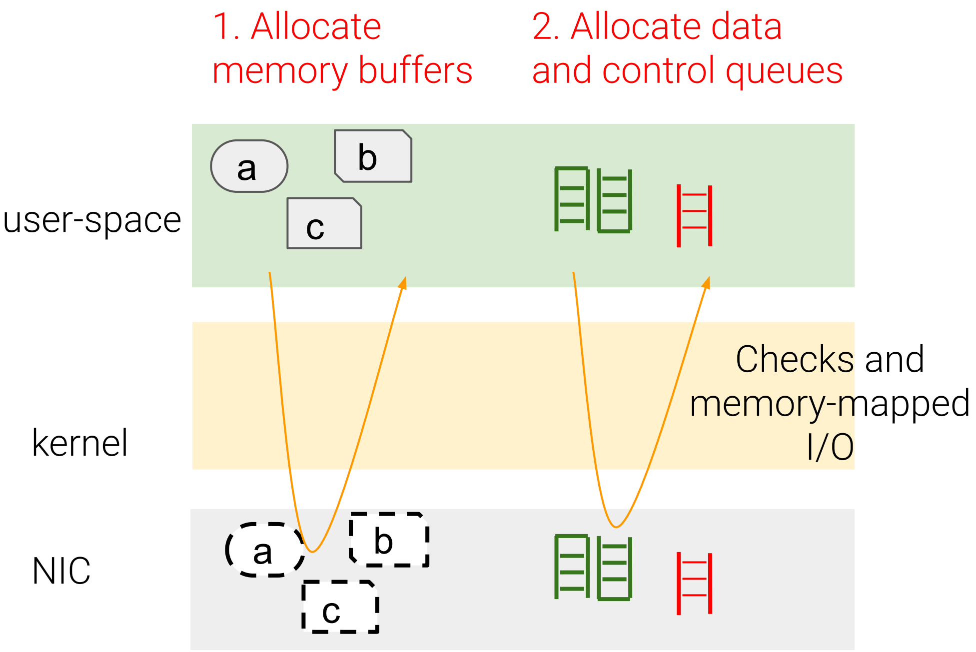
Idea: Make **control** and **data** operations explicit by separating the data from the control **abstractions**.

needs new interfaces

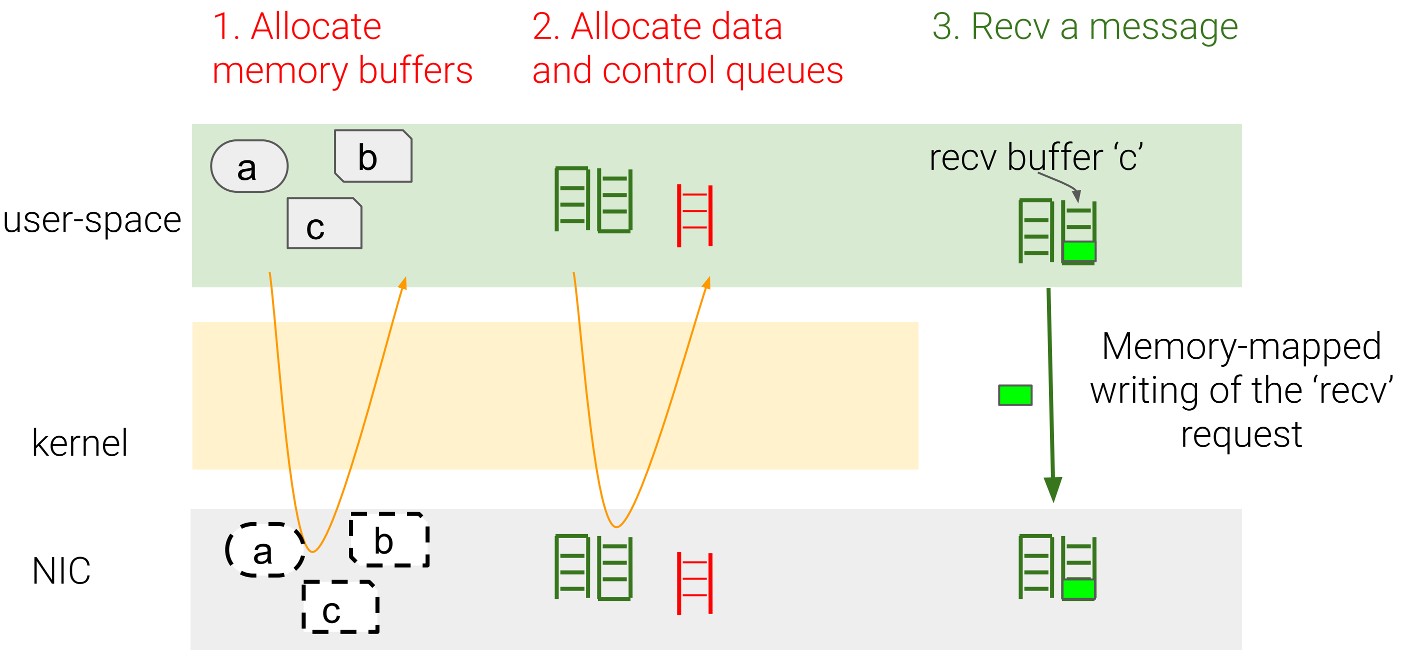
Putting All Together - User-Space Networking



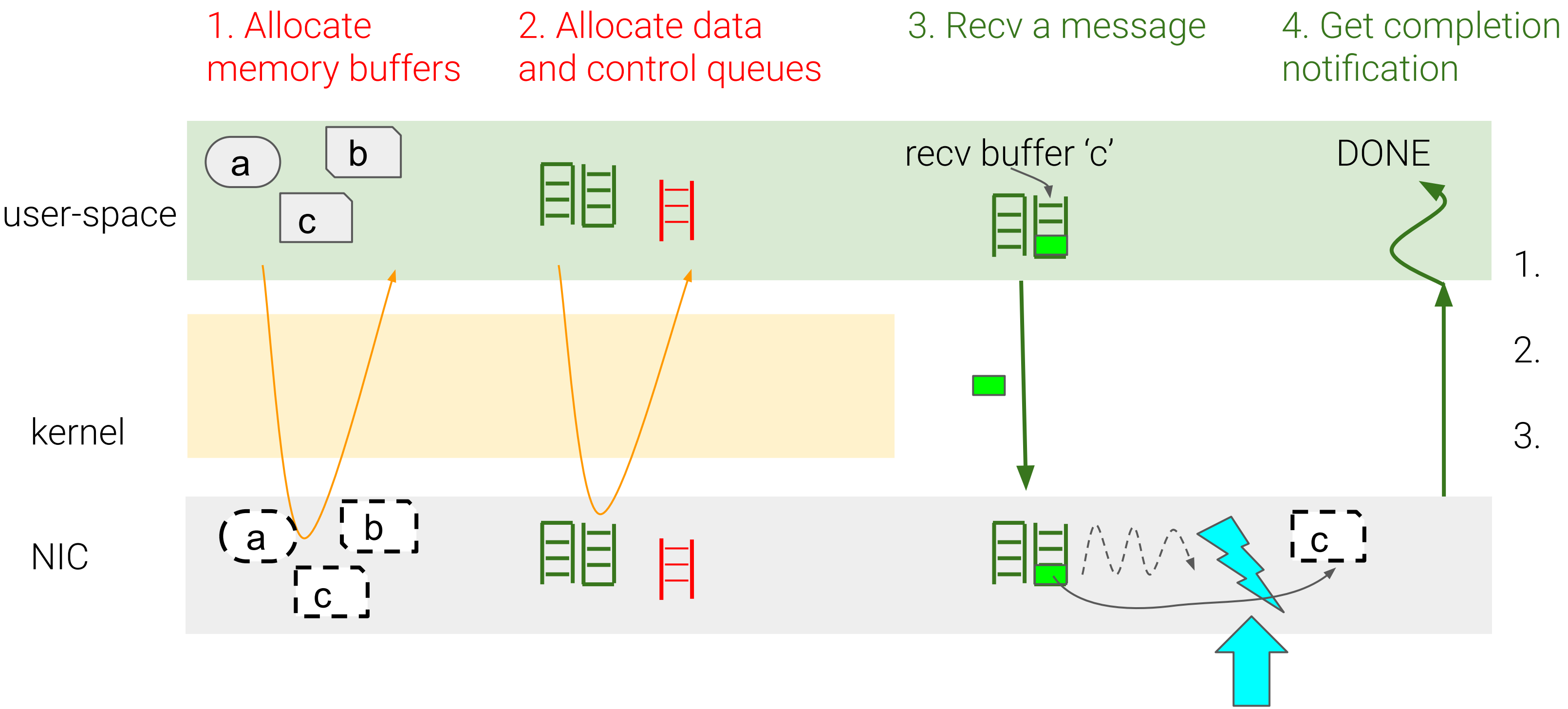
Putting All Together - User-Space Networking



Putting All Together - User-Space Networking

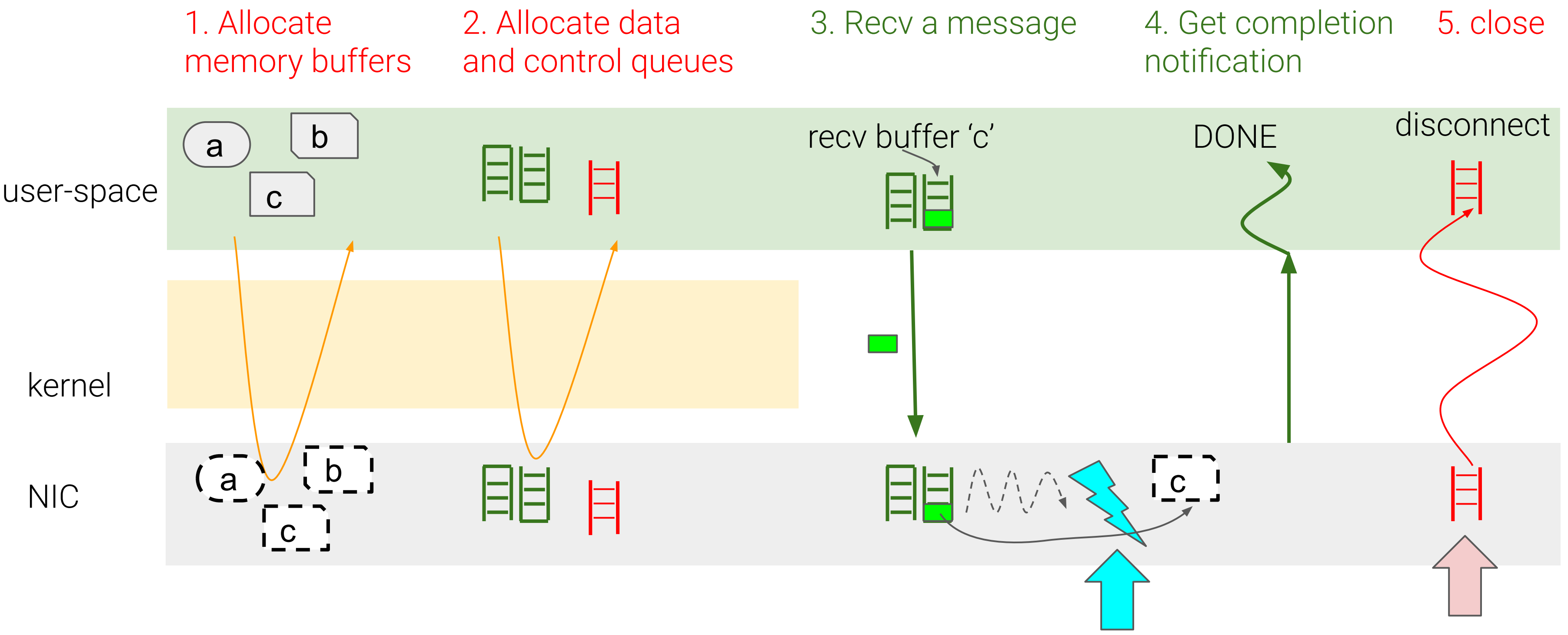


Putting All Together - User-Space Networking

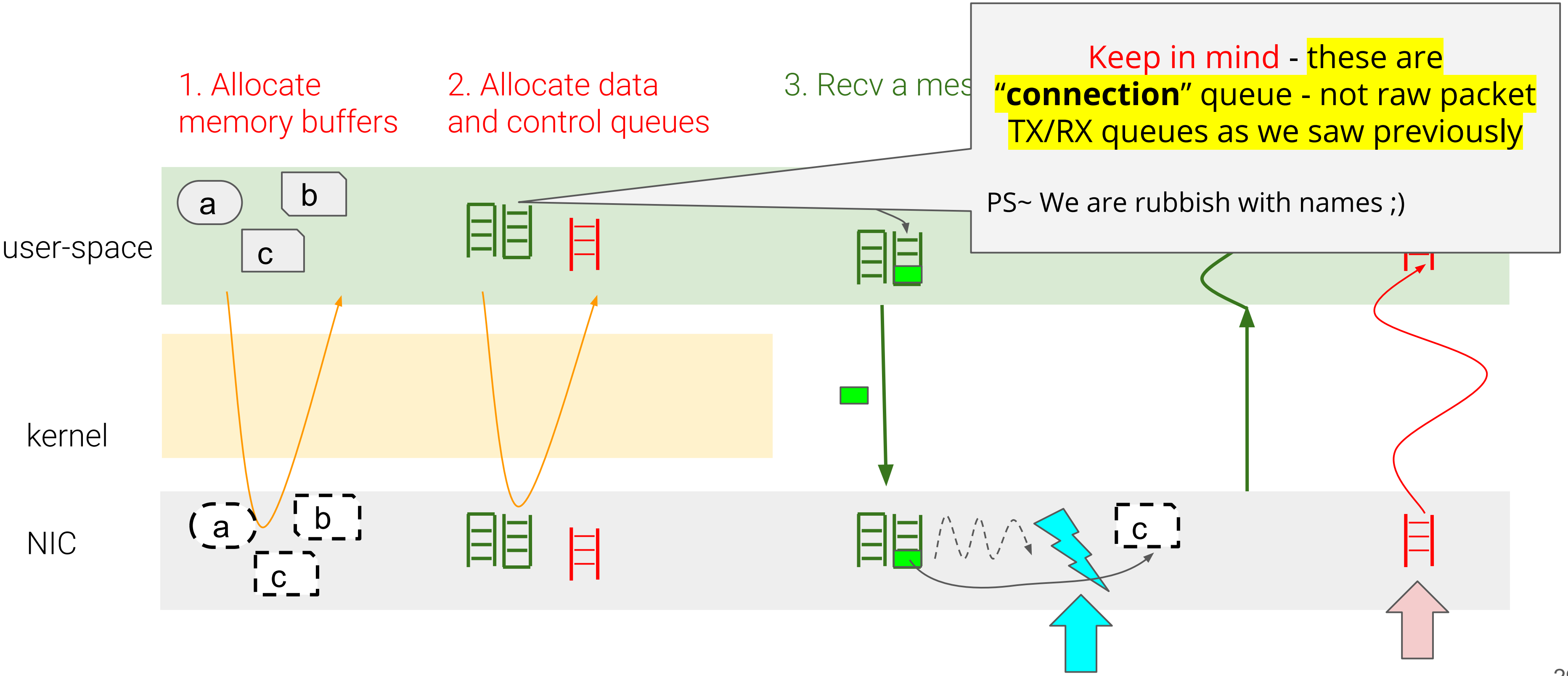


1. Network processing,
2. DMA to the buffer ‘c’,
3. Notification to the application

Putting All Together - User-Space Networking

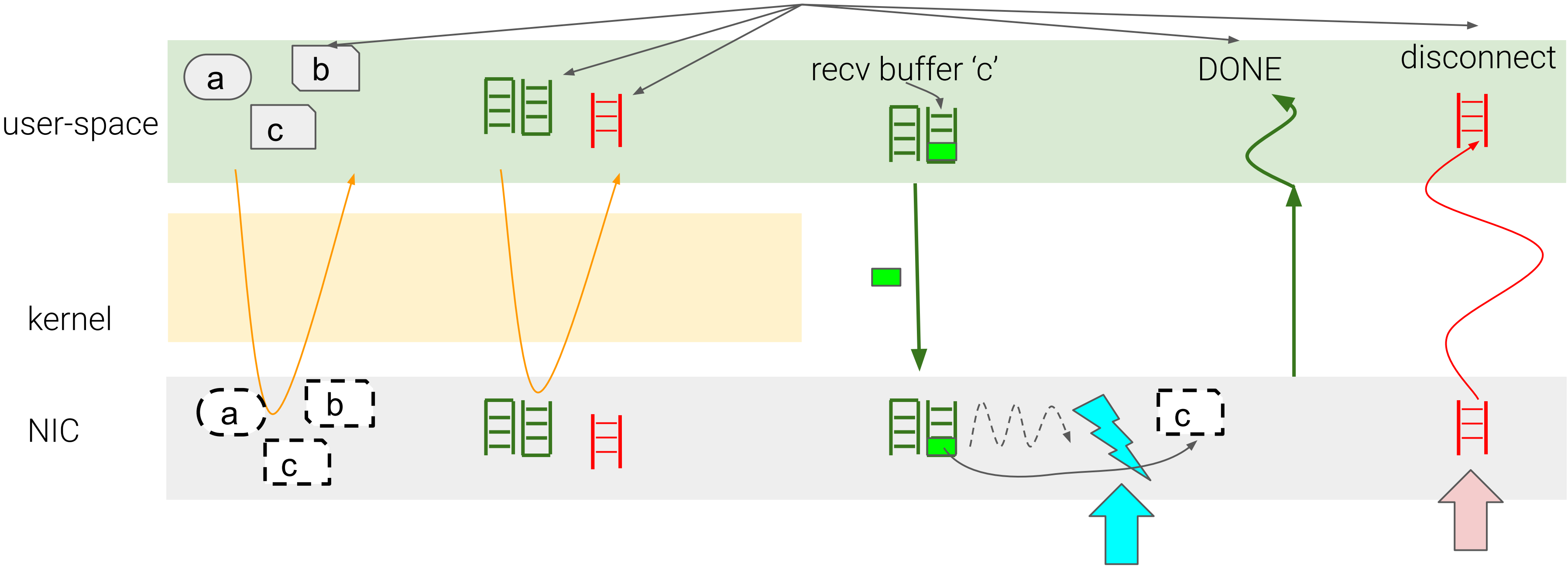


Putting All Together - User-Space Networking



Putting All Together - User-Space Networking

What new abstractions, objects, do you see here?



New RDMA Objects

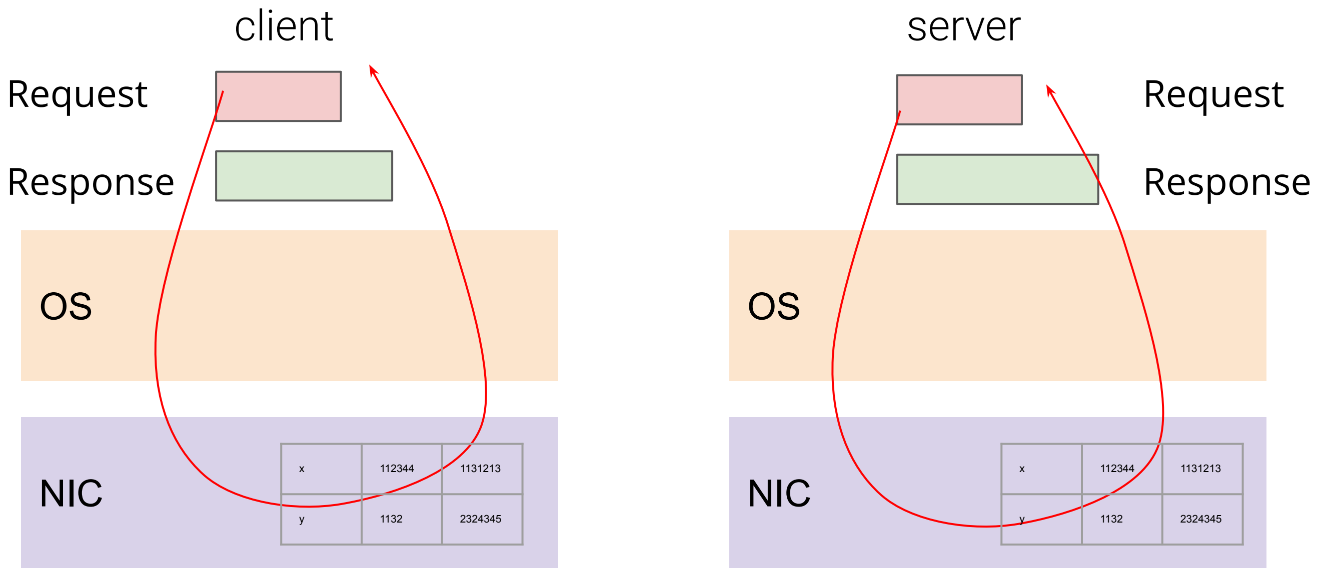
1. **Memory buffers**: from where the data transmission/reception happens
2. **Connection send/receive queues**: represents a *connection*, also known as a queue pair **(QP)**
3. **Control queues** (a few flavors)
   a. Network control queue: new connection, disconnect, NIC up/down
   b. I/O event queue: network I/O finished, error

There are a few more as we will see later, but for now these are high level objects we can see

Network I/O happens by posting work requests (**WRs**) on the QP

- Contains - buffer, length (+*more*)
- Supports SGE, batching, linking independent requests, enforce ordering
- Key difference from socket is that you can only use pre-registered buffers for I/O, not any arbitrary locations for send/recv (which you can for sockets)

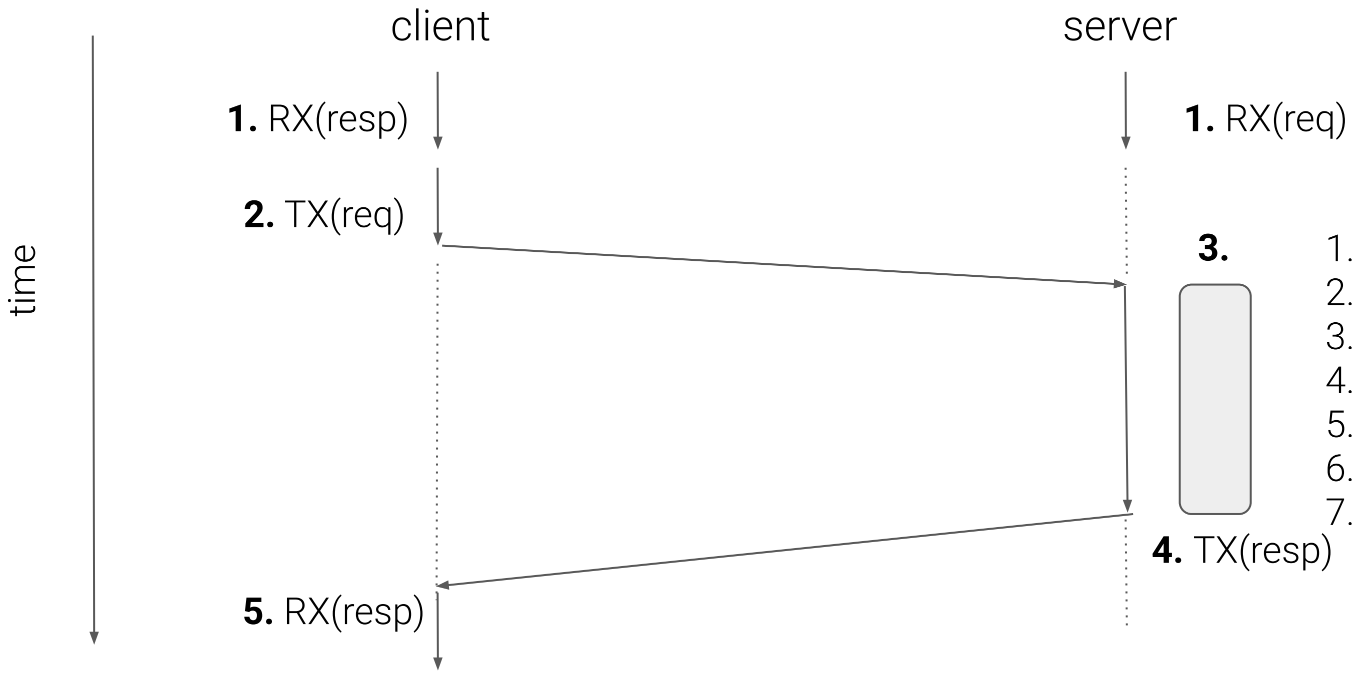
Timeline of a send/recv exchange



We first have to do ***memory registration*** for any buffer on which we have to do I/O

- The RDMA NIC ***remembers*** the address and length of such buffers, and returns an ***identifier***
    - This identifier can be used multiple times, no need to program NIC multiple times for DMA
- This is different from normal NIC/DMA where NIC does not remember the buffer

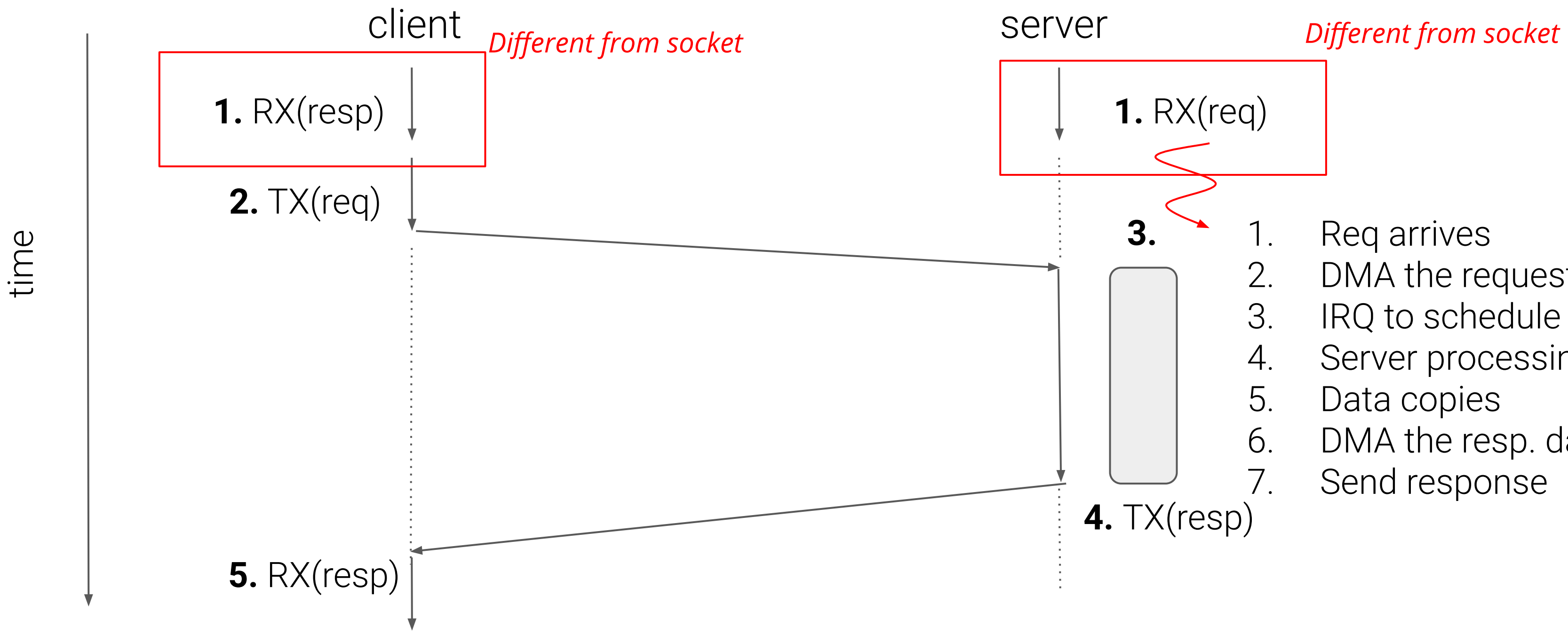
Timeline of a send/recv exchange



1. Req arrives
2. DMA the request
3. IRQ to schedule
4. Server processing
5. Data copies
6. DMA the resp. data
7. Send response

Remember: all this interaction is directly between application and the NIC hardware

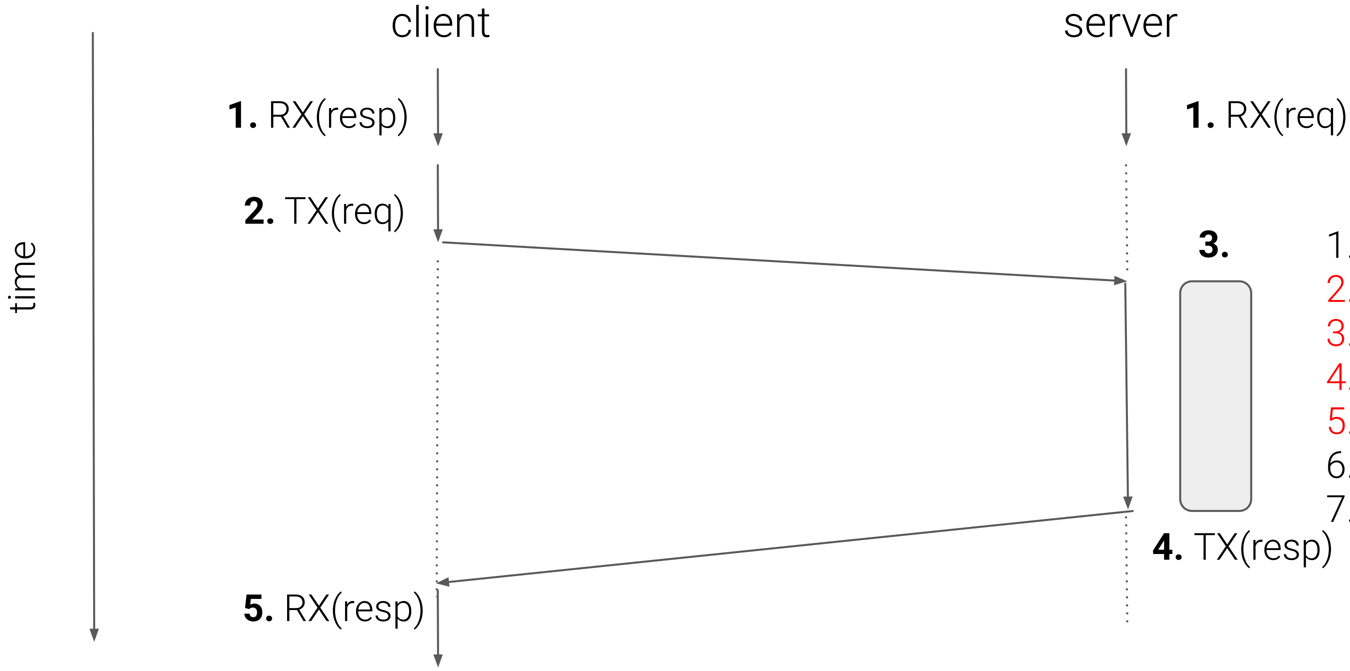
Timeline of a send/recv exchange



1. Req arrives
2. DMA the request
3. IRQ to schedule
4. Server processing
5. Data copies
6. DMA the resp. data
7. Send response

Two-sided message exchange between processes

Timeline of a send/recv exchange



1. Req arrives
2. DMA the request
3. IRQ to schedule
4. Server processing
5. Data copies
6. DMA the resp. data
7. Send response

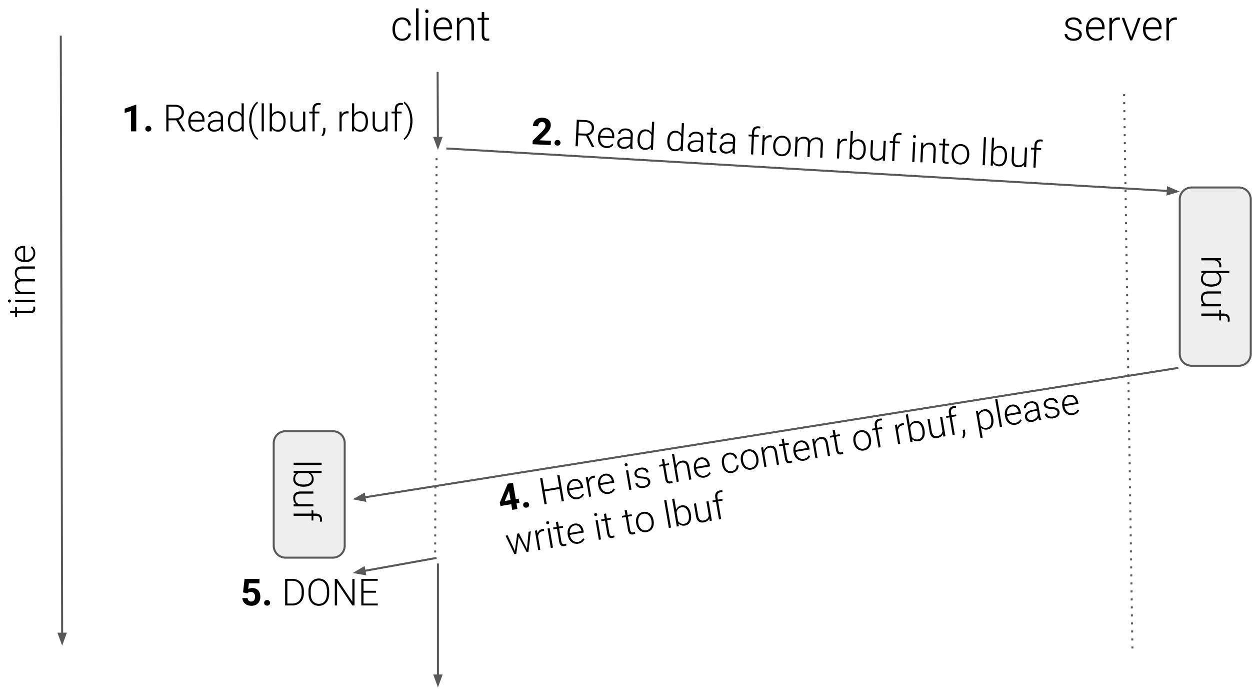
Two-sided message exchange between processes

Can we eliminate server entirely?

The Idea of Remote Direct Memory Access

- Imagine, if all buffers are known up front to everyone in a cluster / data center
- A client/peer can initiate a network transfer by itself
  - "**One-sided**" operations (instead of ***two-sided*** where 2 peers are involved)
- An RDMA **WRITE** specifies:
  - Which local/client buffer data should be read from
  - Which remote/server buffer data should be written to
- An RDMA **READ** specifies:
  - Which remote/server buffer data should be read from
  - Which local/client buffer data should be written into

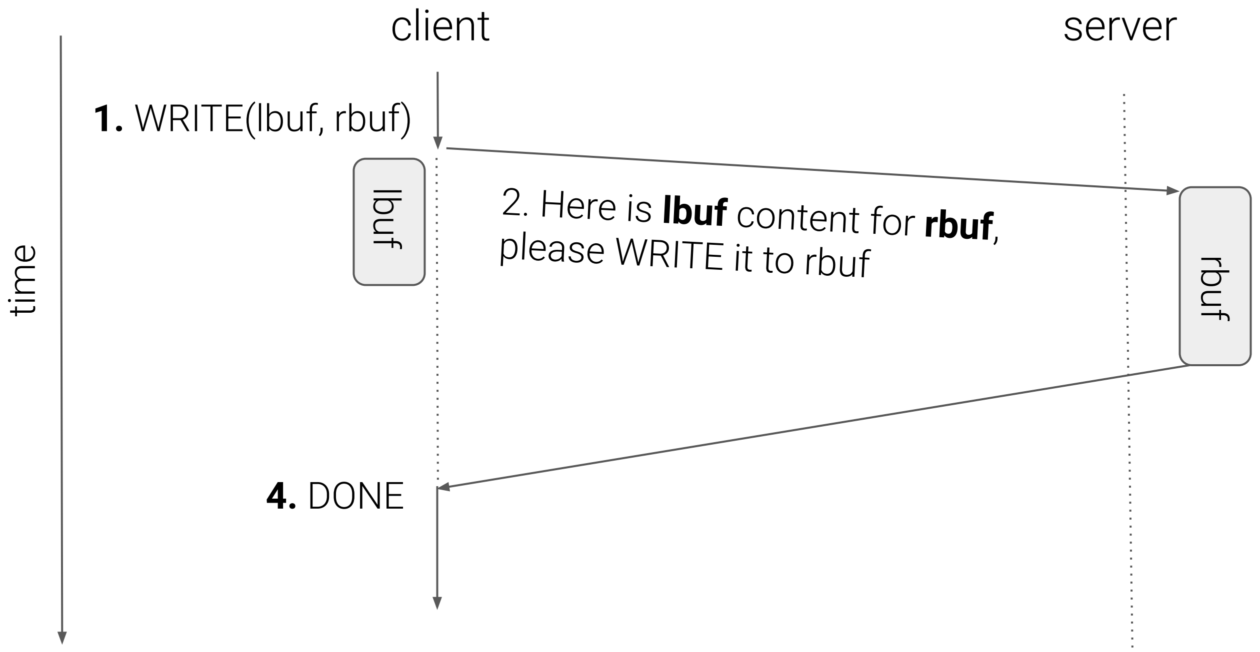
RDMA READ Operation



1. Req arrives
2. ~~DMA the request~~
3. ~~IRQ to schedule~~
4. ~~Server processing~~
5. ~~Data copies~~
6. DMA from rbuf
7. Send READ response

Example one-sided RDMA READ operation

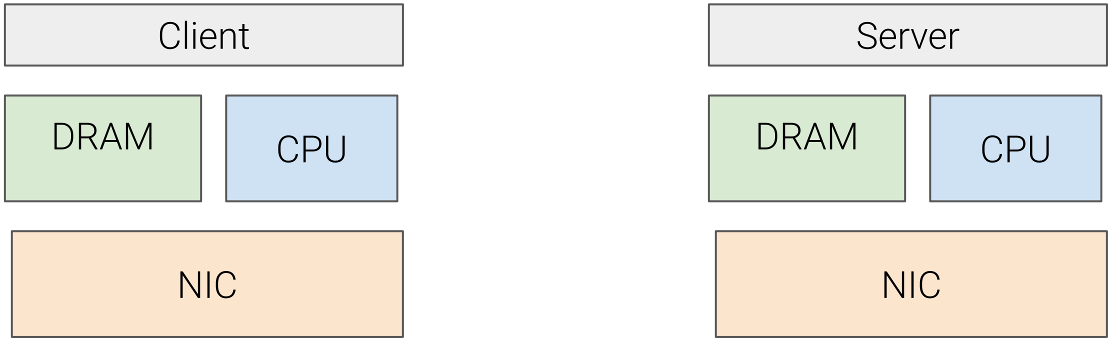
RDMA WRITE Operation



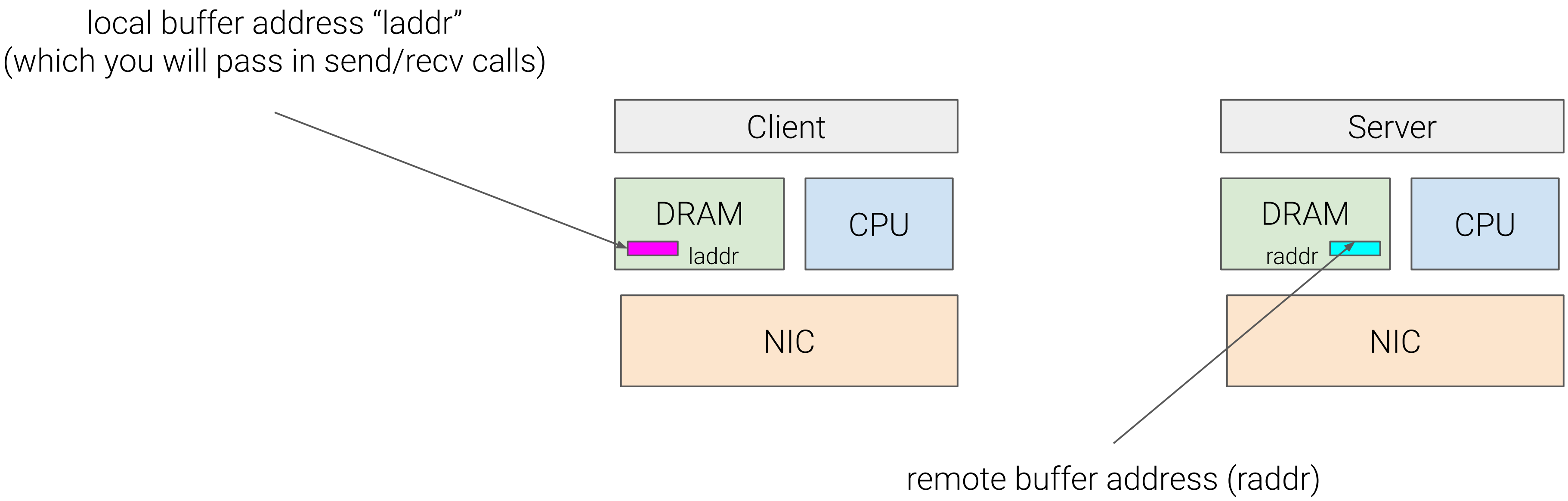
1. Req arrives
2. ~~DMA the request~~
3. ~~IRQ to schedule~~
4. ~~Server processing~~
5. ~~Data copies~~
6. DMA to rbuf
7. Send WRITE response

Example one-sided RDMA WRITE operation

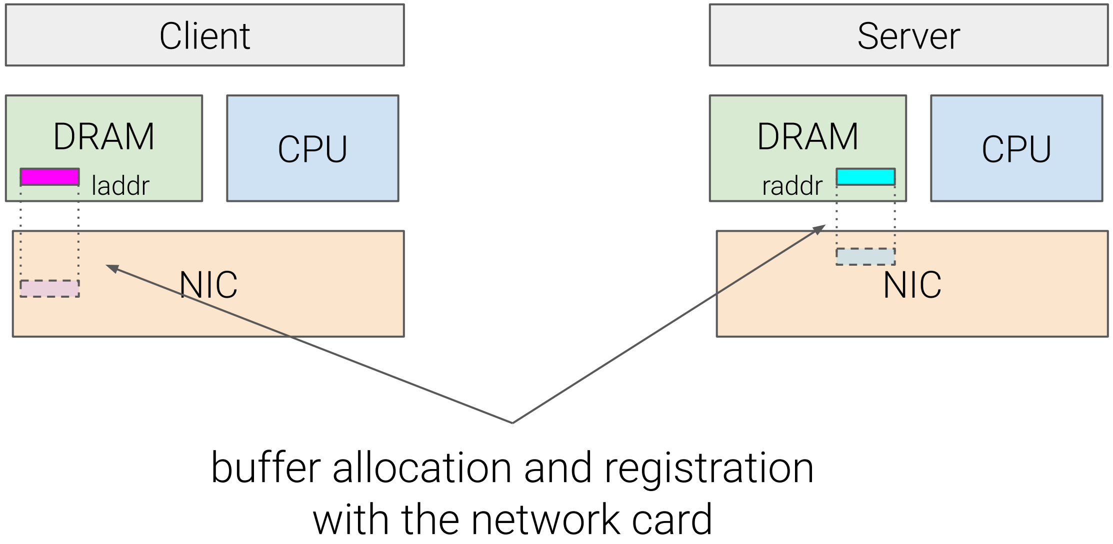
RDMA: Architectural View



RDMA: Architectural View

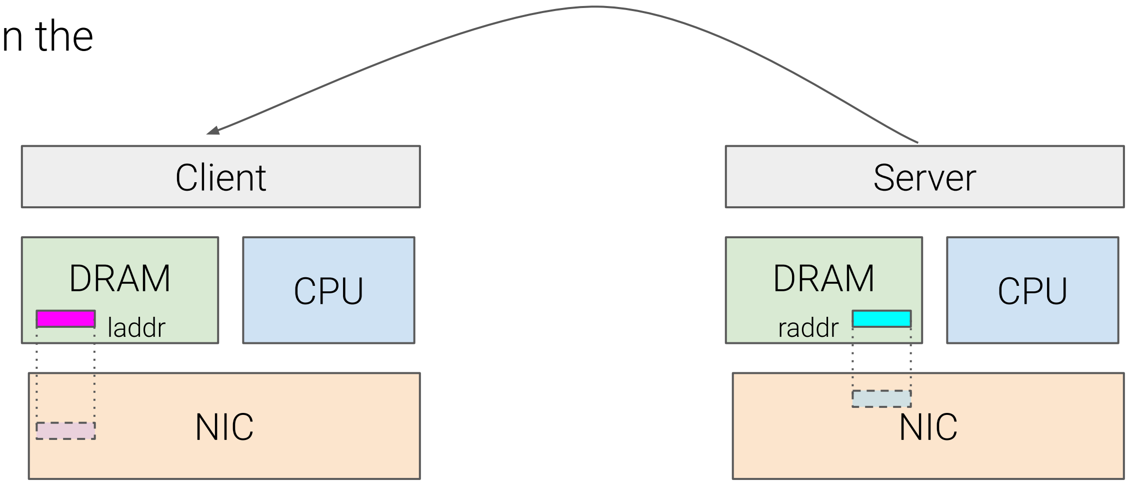


RDMA: Architectural View



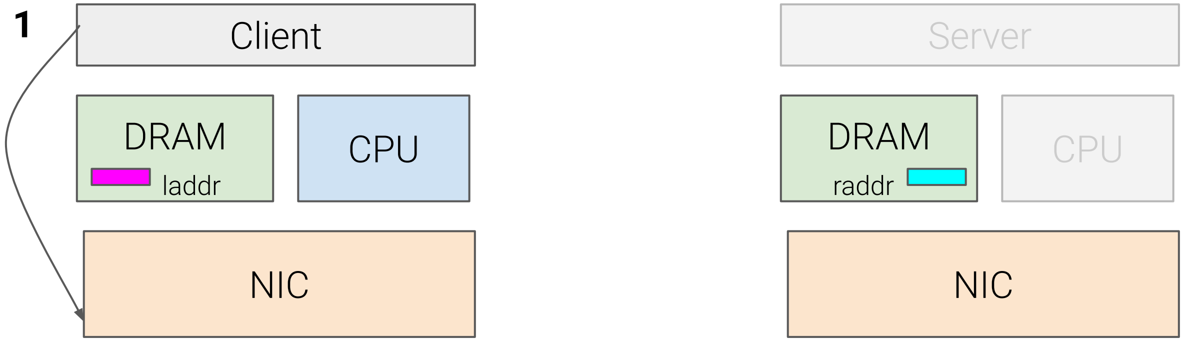
RDMA: Architectural View

Hey! Your content is stored in the buffer at 'raddr'



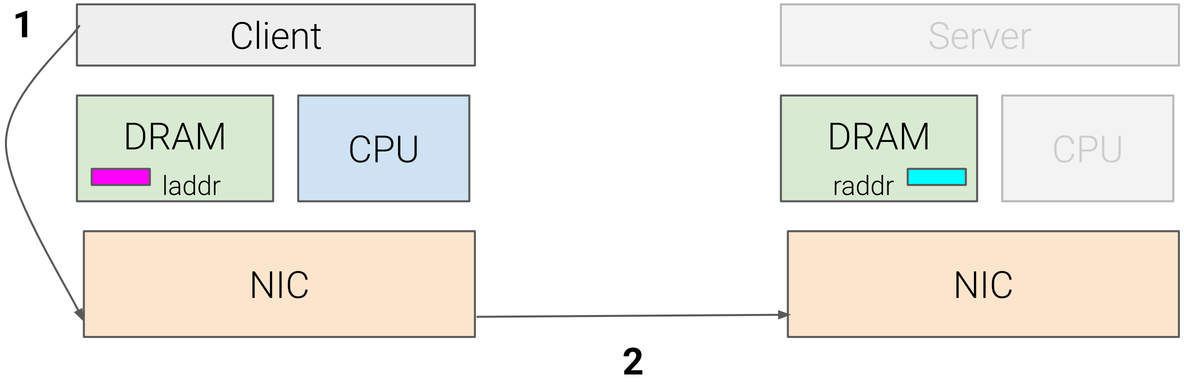
RDMA: Architectural View

1. Client: READ remote memory address (raddr) to local address (laddr)



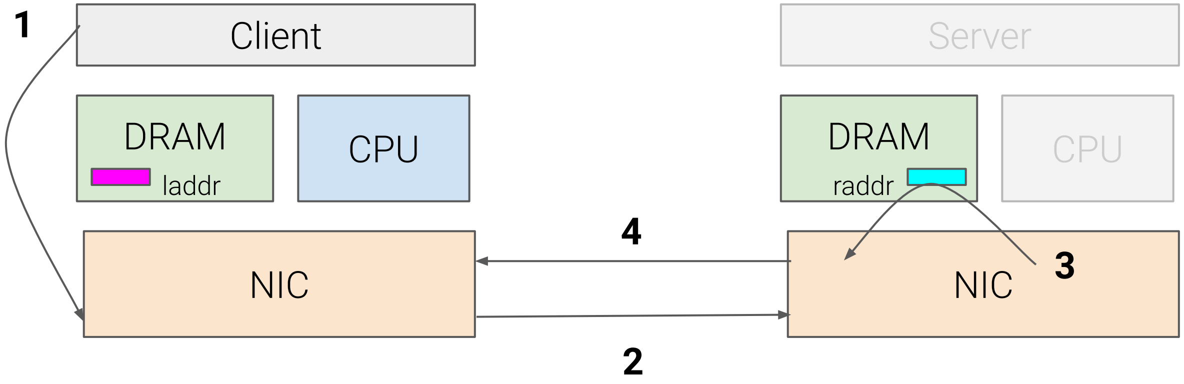
RDMA: Architectural View

1. Client: READ remote memory address (raddr) to local address (laddr)
2. Client: posts READ request



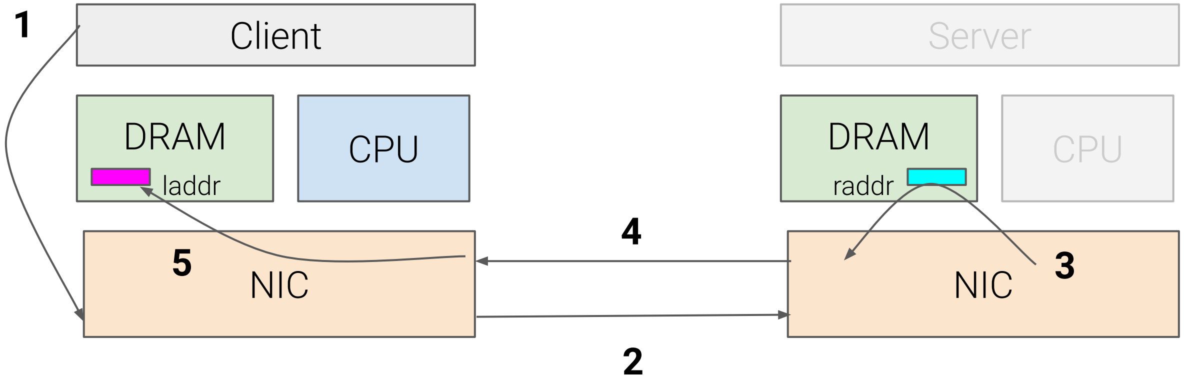
RDMA: Architectural View

1. Client: READ remote memory address (raddr) to local address (laddr)
2. Client: posts READ request
3. Server: read local (raddr) - local DMA operation
4. Server: TX data back to client NIC



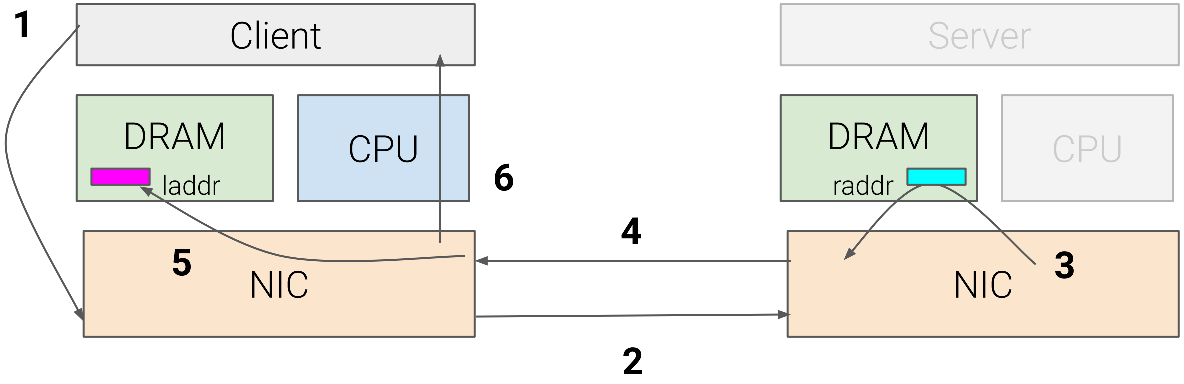
RDMA: Architectural View

1. Client: READ remote memory address (raddr) to local address (laddr)
2. Client: posts READ request
3. Server: read local (raddr) - local DMA operation
4. Server: TX data back to client NIC
5. Client: local DMA to (laddr) buffer in DRAM



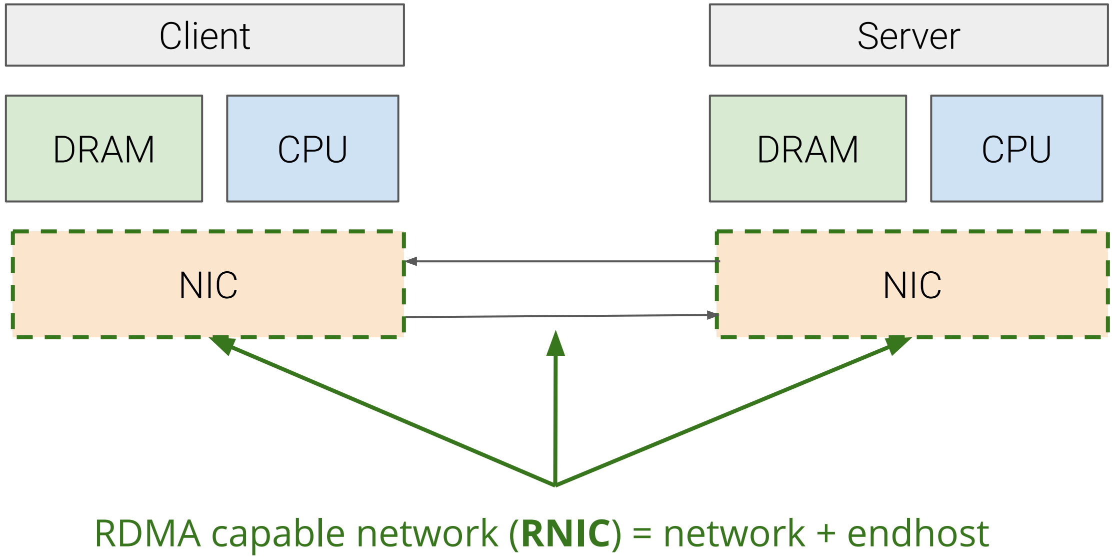
RDMA: Architectural View

1. Client: READ remote memory address (raddr) to local address (laddr)
2. Client: posts READ request
3. Server: read local (raddr) - local DMA operation
4. Server: TX data back to client NIC
5. Client: local DMA to (laddr) buffer in DRAM
6. Client: interrupt the local CPU/OS to notify completion about the client's READ operation



RDMA: Architectural View

1. Client: READ remote memory address (raddr) to local address (laddr)
2. Client: posts READ request
3. Server: read local (raddr) - local DMA operation
4. Server: TX data back to client NIC
5. Client: local DMA to (laddr) buffer in DRAM
6. Client: interrupt the local CPU/OS to notify completion about the client's READ operation



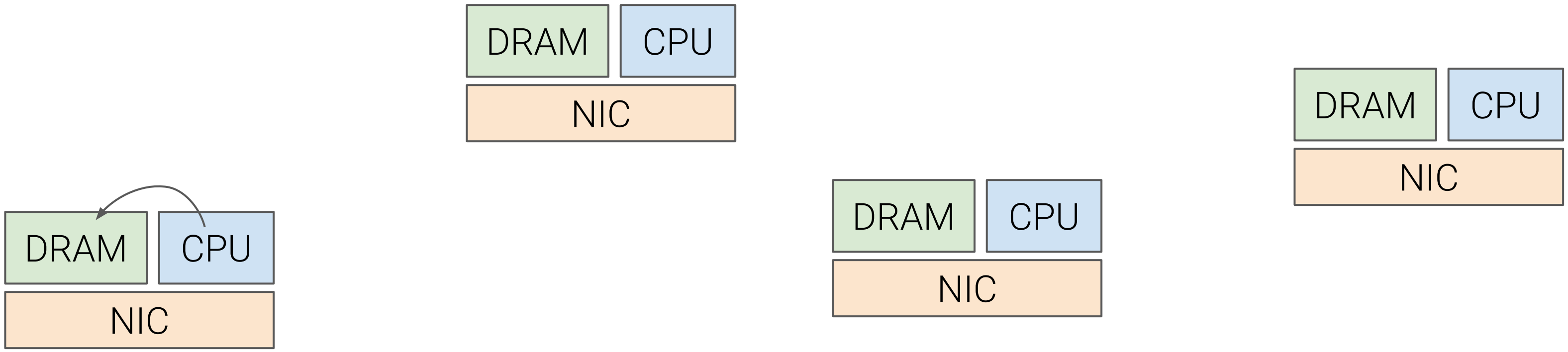
So, this is RDMA

- It is very powerful idea - no remote application/OS/CPU involvement in data transfer
- Once you have capability to access remote memory from network you can imagine doing a lot of things without having to worry about if the remote application is ready to send or receive
- Different types of operations: READ, WRITE, ATOMICS (update, CAS), even transactions (!)
- Not limited just to system DRAM, think of DRAM in the GPU (hint: its possible)

We did not develop all of this in last 10 years

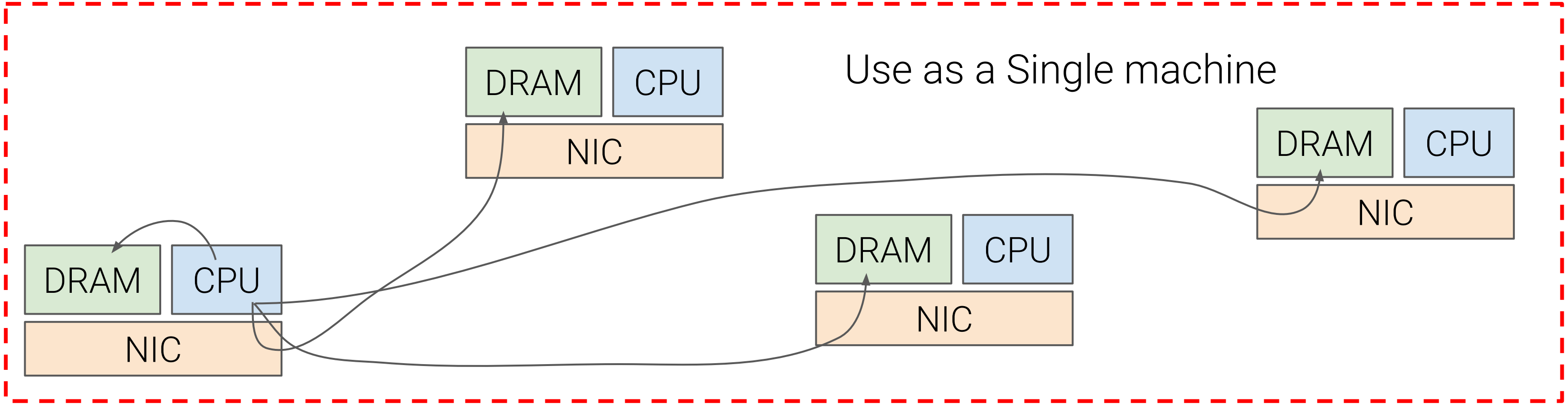
A Brief History

- **1980s-1990s**: a long history of high-performance networking research
  - Building networked multi-processor systems - commodity, cheap vs. supercomputers
  - Berkeley NOW, Stanford FLASH, Princeton SHRIMP, Cornell U-Net, HP labs Hamlyn
  - The goal was to connect and integrate CPUs via network as efficiently as possible



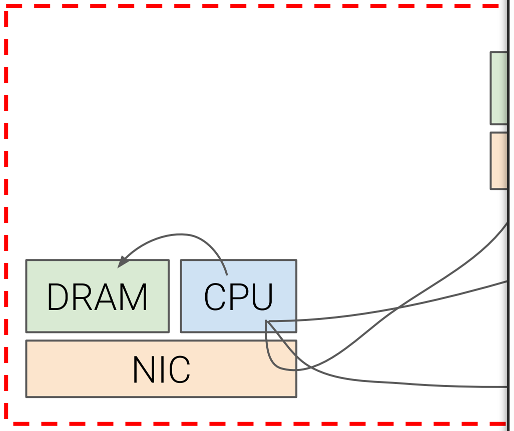
A Brief History

- **1980s-1990s**: a long history of high-performance networking research
  - Building networked multi-processor systems - commodity, cheap vs. supercomputers
  - Berkeley NOW, Stanford FLASH, Princeton SHRIMP, Cornell U-Net, HP labs Hamlyn
  - The goal was to connect and integrate CPUs via network as efficiently as possible



A Brief History

- **1980s-1990s**: a long history of
    - Building networked mul
    - Berkeley NOW, Stanford
    - The goal was to connect



ORCA: A LANGUAGE FOR PARALLEL

PROGRAMMING OF DISTRIBUTED SYSTEMS†

*Henri E. Bal **
M. Frans Kaashoek
Andrew S. Tanenbaum

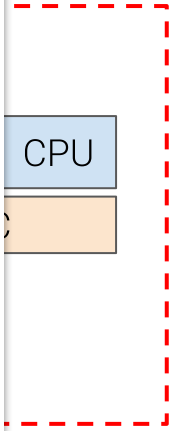
Dept. of Mathematics and Computer Science
Vrije Universiteit
Amsterdam, The Netherlands

ABSTRACT

Orca is a language for implementing parallel applications on loosely coupled distributed systems. Unlike most languages for distributed programming, it allows processes on different machines to share data. Such data are encapsulated in data-objects, which are instances of user-defined abstract data types. The implementation of Orca takes care of the physical distribution of objects among the local memories of the processors. In particular, an implementation may replicate and/or migrate objects in order to decrease access times to objects and increase parallelism.

This paper gives a detailed description of the Orca language design and motivates the design choices. Orca is intended for applications programmers rather than systems programmers. This is reflected in its design goals to provide a simple, easy to use language that is type-secure and provides clean semantics.

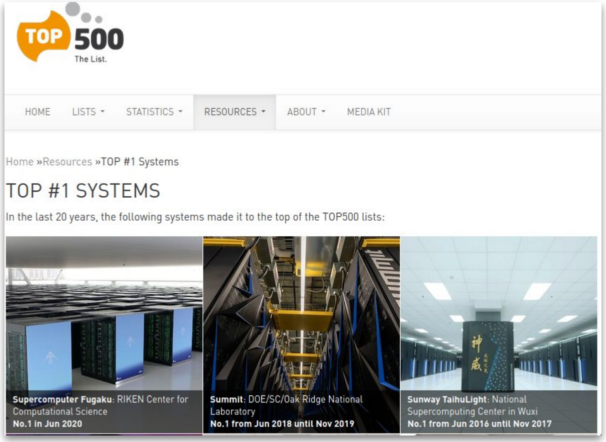
The paper discusses three example parallel applications in Orca, one of which is described in detail. It also describes one of the existing implementations, which is based on reliable broadcasting. Performance measurements of this system are given for three parallel applications. The measurements show that significant speedups can be obtained for all three applications. Finally, the paper compares Orca with several related languages and systems.

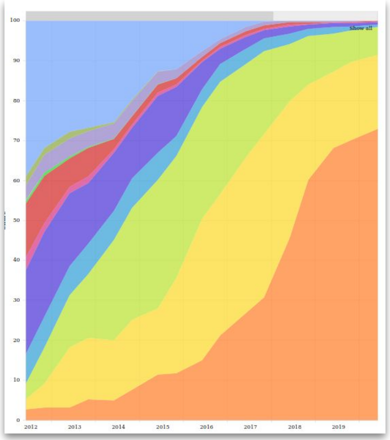


A Brief History

- **1980s-1990s**: a long history of high-performance networking research
  - Building networked multi-processor systems - commodity, cheap vs. supercomputers
  - Berkeley NOW, Stanford FLASH, Princeton SHRIMP, Cornell U-Net, HP labs Hamlyn
  - The goal was to connect and integrate CPUs via network as efficiently as possible

- **1990s:** but CPUs were getting fast, so these efforts finally focussed on HPC workloads
  - Infiniband, Myrinet, QsNet (Quadrics), BlueGene, Cray, NUMAlink (see the SC Top500 list from 2000s)





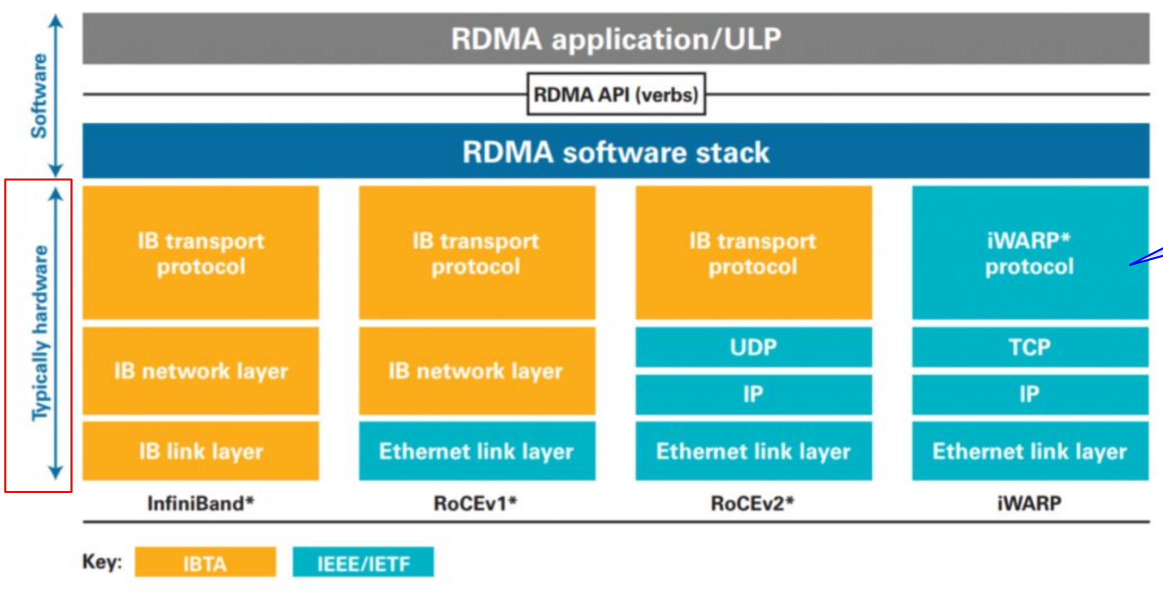
https://www.top500.org/

2012-2019 - different color, different network

A Brief History

- **1980s-1990s**: a long history of high-performance networking research
    - Building networked multi-processor systems - commodity, cheap vs. supercomputers
    - Berkeley NOW, Stanford FLASH, Princeton SHRIMP, Cornell U-Net, HP labs Hamlyn
    - The goal was to connect and integrate CPUs via network as efficiently as possible
- **1990s**: but CPUs were getting fast, so these efforts finally focussed on HPC workloads
    - Infiniband, Myrinet, QsNet (Quadrics), BlueGene, Cray, NUMAlink (see the SC Top500 list from 2000s)
- **Late-2000s**: CPU performance falters and the focus is back on high-performance networking
    - Ethernet improved significantly and caught up Infiniband performance
- **Today commodity:** *InfiniBand, RoCE, iWARP, OminiPath* support RDMA networking stacks
- **Today supercomputers:** TOFU interconnect (Fujitsu), Sunway, CRAY Aries and Gemini, Bull BXI (Atos), IBM...
    - https://www.top500.org/statistics/list/

In the Layer Model



Defines how run a message protocol for RDMA operations on top of TCP byte-stream
RFC 5040-5044

A Survey of End-System Optimizations for High-Speed Networks, ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 51 Issue 3, July 2018. https://dl.acm.org/citation.cfm?doid=3212709.3184899

Image reference: https://fakecineaste.blogspot.com/2018/02/

Key items to understand - Part 1

- **RDMA** can refer to many things - unfortunately there is no one definition
    - Broadly speaking the idea represents capability to read remote memory w/o *remote OS/application involvement, **hence, remote DMA***
        - Physical, or virtual addressing or referencing capability for remote/local memory
    - With such capability, you can also do send/recv (but less exciting)

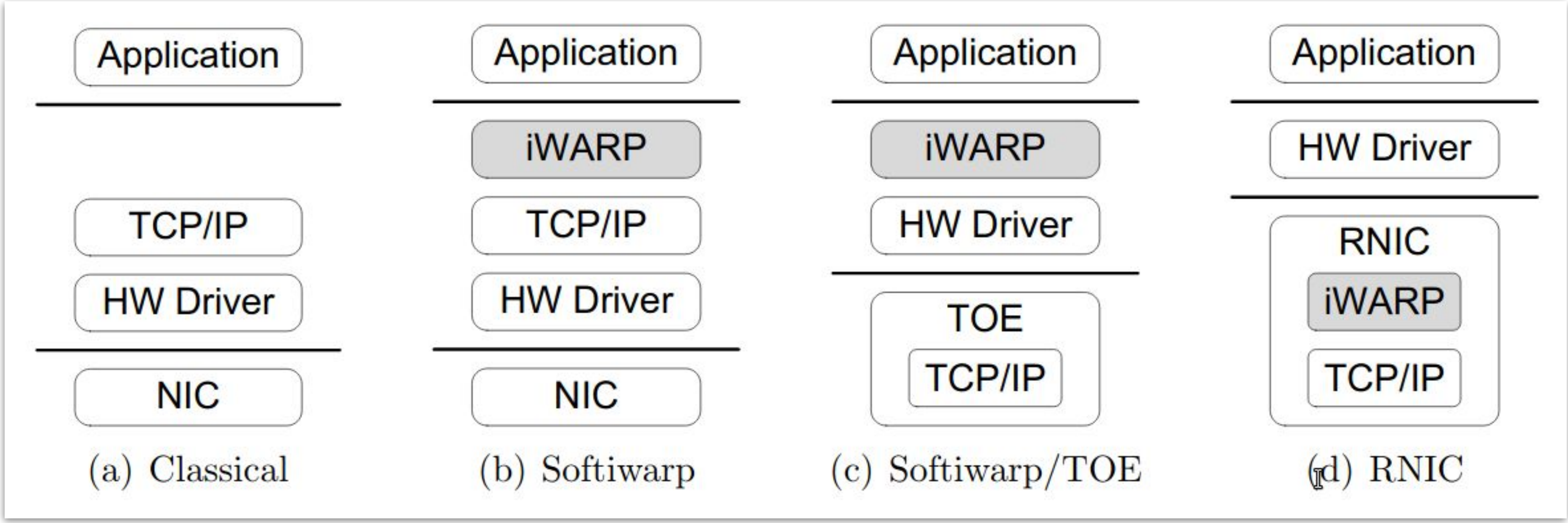
- There is no **ONE** RDMA API like socket
    - As we saw previously, back in days with many supercomputer vendor they had different link technology which had its own implementation of RDMA “semantics”
    - Often wrapped under another high-level API like MPI (Standard, message passing)
    - A (pseudo) standard stack is **Open-Fabric Alliance (OFA)**

Key items to understand - part 2

- The RDMA idea is independent of the networking technology and the programming interfaces given to the user:
  - Today, there are Infiniband, iWARP, RoCE, OminiPath - all support RDMA operations on top of different networking layers
  - They all support the OFED RDMA Stack (which is part of the Linux main line)
- The idea of RDMA can be implemented in software also
  - SoftiWARP, SoftRoCE - software kernel devices that run the RDMA protocol inside the kernel
  - Full API - but limited performance gains. Remote OS cannot be skipped if the software device is running in the kernel, but the remote application can

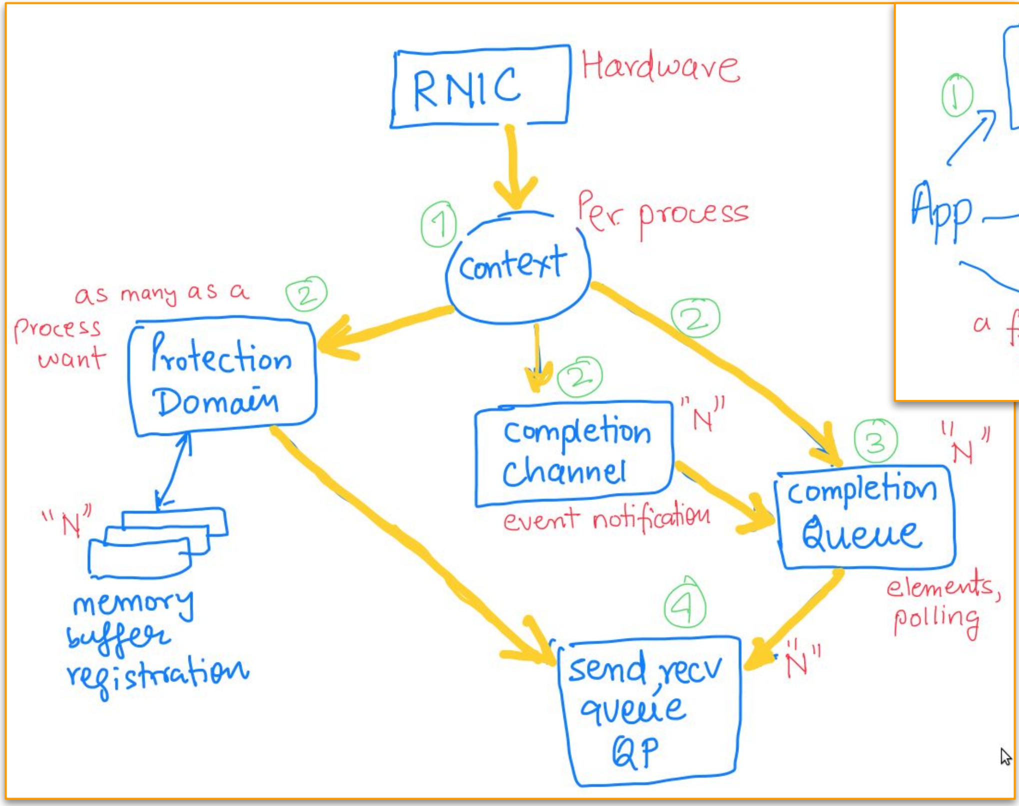
Trivedi et al., A case for RDMA in clouds: turning supercomputer networking into commodity, https://dl.acm.org/doi/10.1145/2103799.2103820

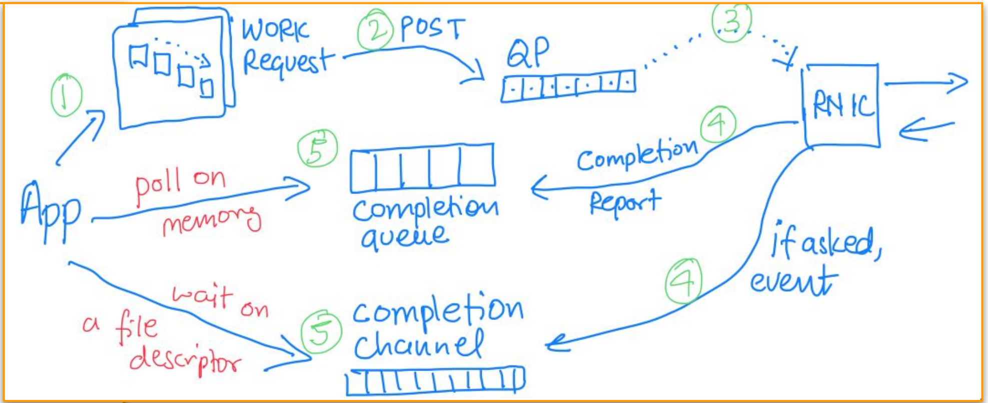
iWARP - RDMA over TCP/IP



Mogul in his paper (*TCP offload is a dumb idea*) was talking about ToE with RDMA-type API (not socket)

RDMA Object Relationships and Workflow





Objects relationship

Workflow

What does NIC need to know

RDMA capable network cards or **RNICs** typically maintain "*some state*" on the network card hardware:

- In case of iWARP: TCP offload engine, connection state (see here: *ToE but without sockets*)
- Memory buffers (virtual, physical addresses, length, permissions)
- Queue pairs (QPs) : for posting network I/O requests
- Pending/in-progress network I/O requests
  - Send/Recv, READs, WRITEs, Atomics
  - Their execution and notification orders
- Completion queues (CQs): to get completion status
- Some protection domain related context to identify resources belonging to one process from another
- And more ...

If you want to try RDMA and see its programs

Option 1: Build and run things on DAS at VU

- Currently has 56 Gbps InfiniBand
- DAS-6 will have 100 Gbps RoCE network
- Full power of the RDMA operations

Option 2: run a software kernel device (it's actually not that difficult)

- https://github.com/zrlio/softiwarp
- https://animeshtrivedi.github.io/blog/2019/06/26/siw.html
- RDMA server/client example, https://github.com/animeshtrivedi/rdma-example
  - You see how large a server client example in the TCP/socket is ~50 lines
  - This one is around ~1000 lines :)

Why go through so much pain

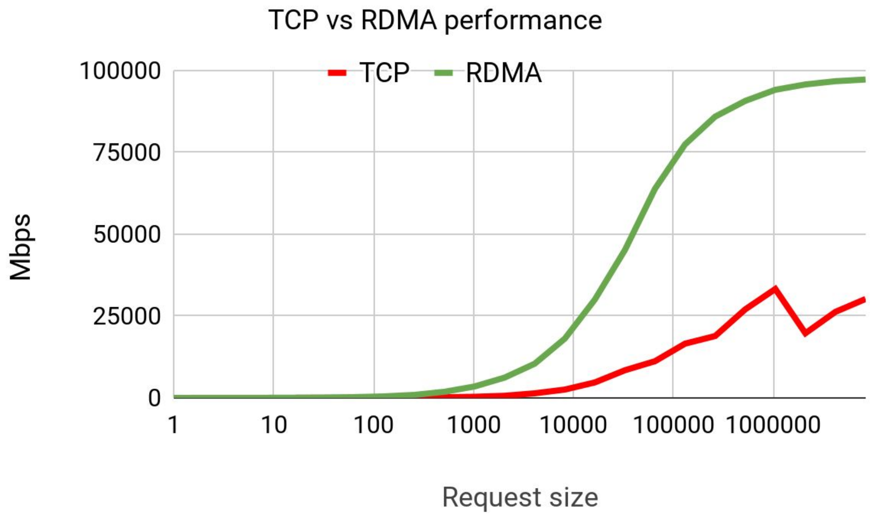
What does RDMA promise to deliver?

- On 100 Gbps RoCE network
- Dual-socket Sandy-Bridge Xeon CPUs
- DDR3 DRAM

Bandwidth and network operation latencies in a simple request-response setup

- client sends a request for ‘x’ bytes of data
- Server sends back ‘x’ bytes of data

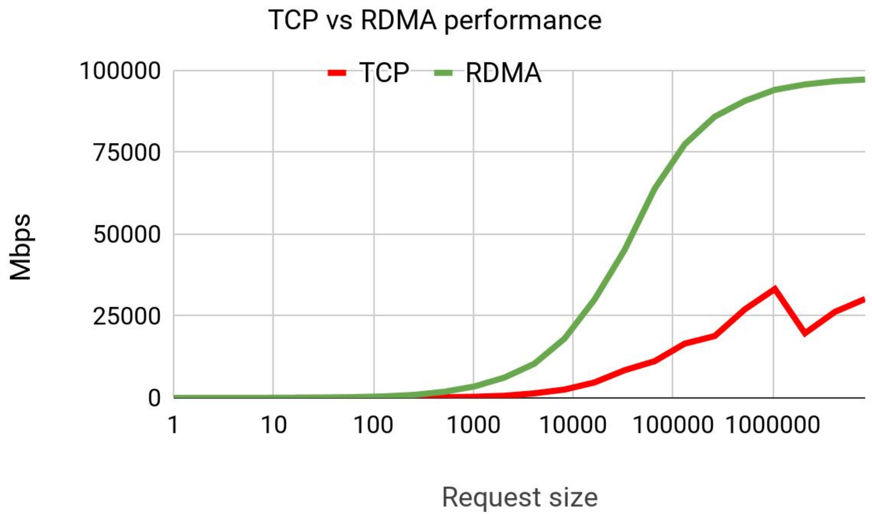
Performance



97.2 Gbps

33.2 Gbps

Performance

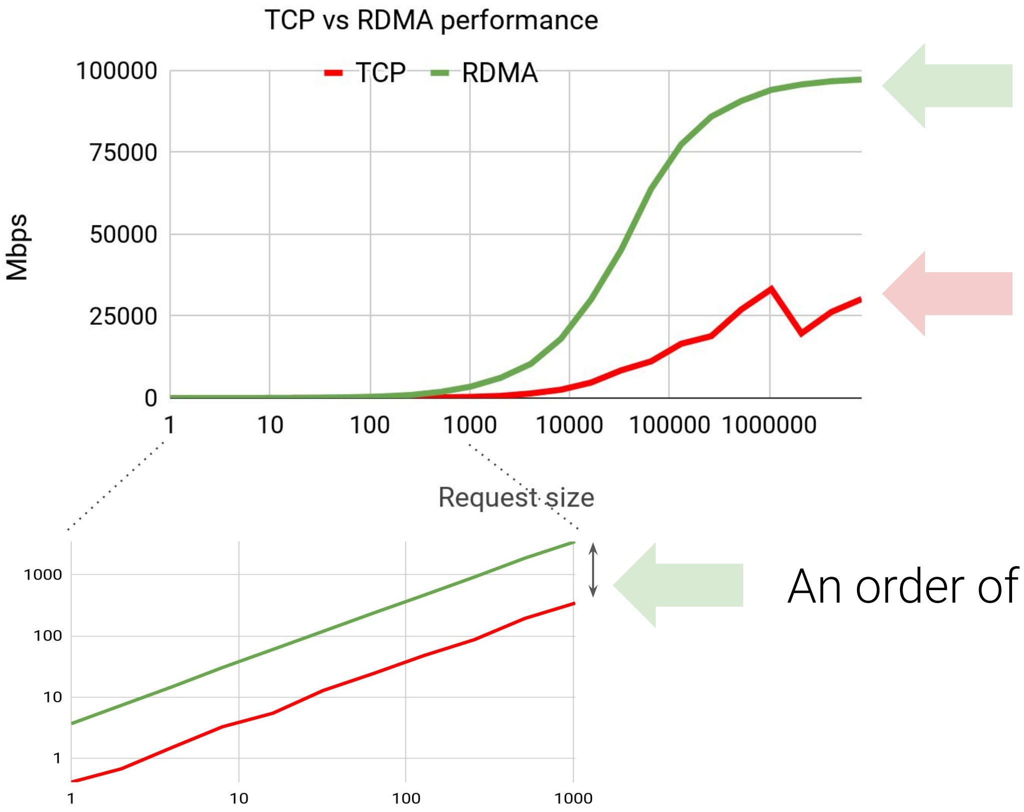


97.2 Gbps

33.2 Gbps

0% CPU utilization

Performance

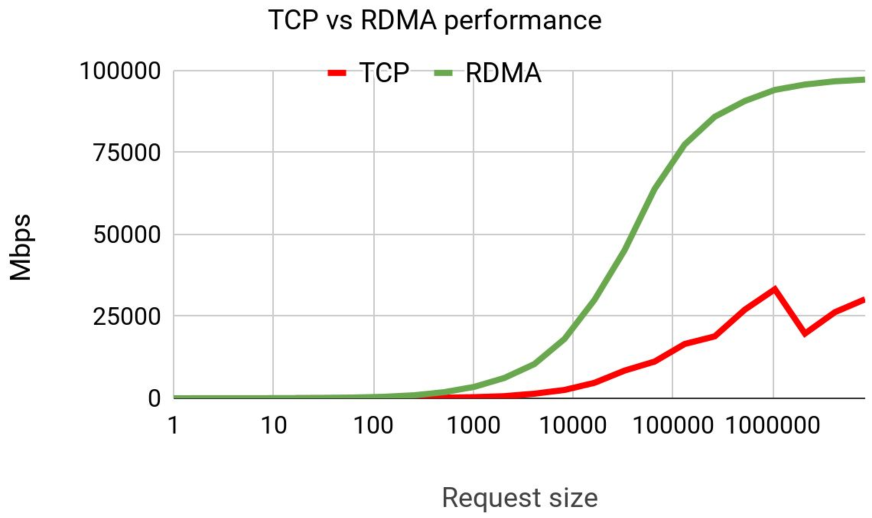


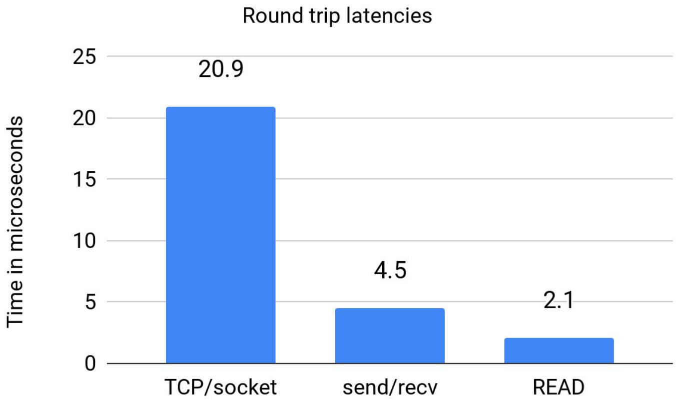
97.2 Gbps

33.2 Gbps

An order of magnitude gap for small requests

Performance





4.6x

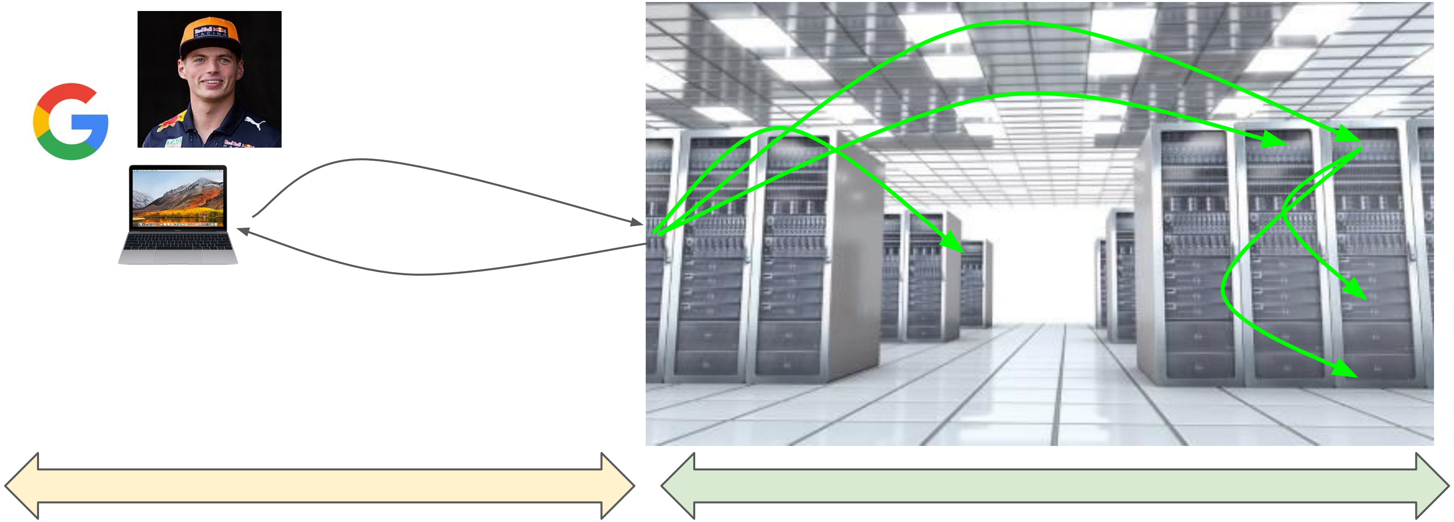
10x

Where do the Performance Gains come from?

- Closer application network integration
  - When, how, where of network processing, closely integrated with application's needs
  - Full control over the network processing
- Better(?), high-performance code
  - The API forces to pushing setup at the beginning, resource allocation
- Hardware offloading
  - Hardware acceleration, less CPU/software involvement
- Bypassing the operating system
  - Lot of boilerplate code skipped
  - Processing close to the metal
- An active area of research - the RIGHT application/network integration framework

Where can you use RDMA?

Data-Center Environment / Rack-scale computing



Over the Internet
- Mbps to Gbps
- 1-10s of msec of RTT

Inside a datacenter
- 100s of Gbps
- 1-10s of usec RTTs

Where can you use RDMA?

Data-Center Environment / Rack-scale computing

- Shared memory
- Key-Value stores
- Caches
- RPCs
- Sync/locking
- File systems
- Services
- ...

Over the Internet

- Mbps to Gbps
- 1-10s of msec of RTT

Inside a datacenter

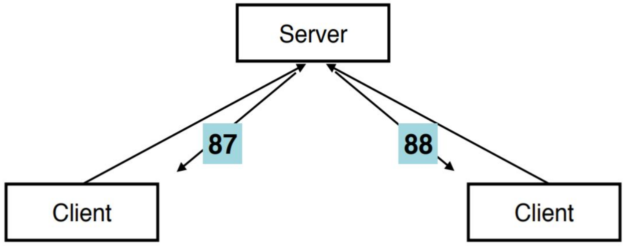
- 100s of Gbps
- 1-10s of usec RTTs

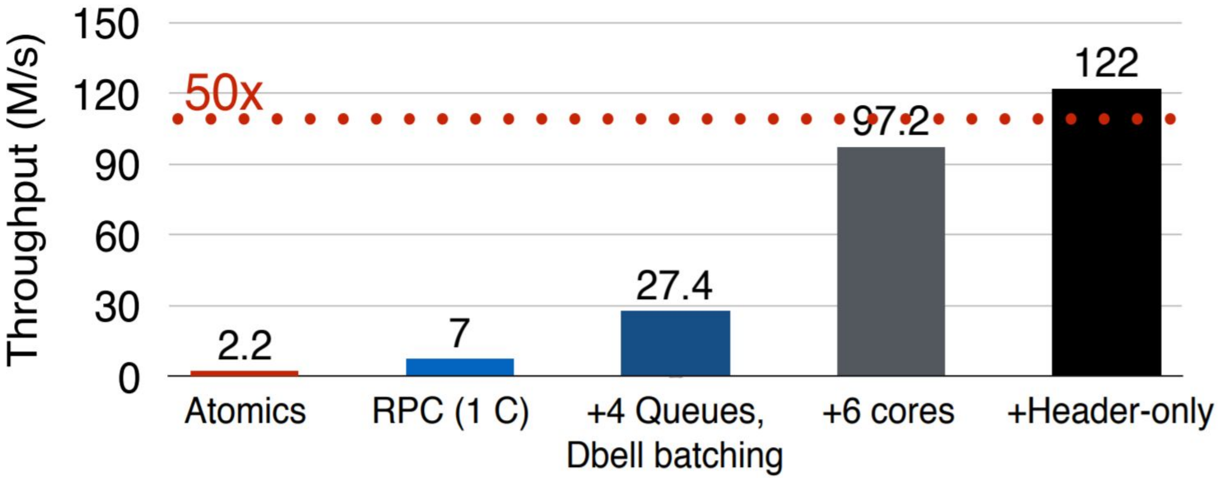
RDMA Programming and Design Space

Operations	READ	WRITE	ATOMIC	SEND	RECV
Transport	Connected	Datagram	Reliable	Unreliable	
Optimizations	Inline	Polling/ Unsignaled	Doorbell batching	WQE scheduling	0len-recvs

Paper: Design Guidelines for High Performance RDMA Systems, Usenix 2016, https://www.usenix.org/system/files/conference/atc16/atc16_paper-kalia.pdf

Example - Sequencer Throughput





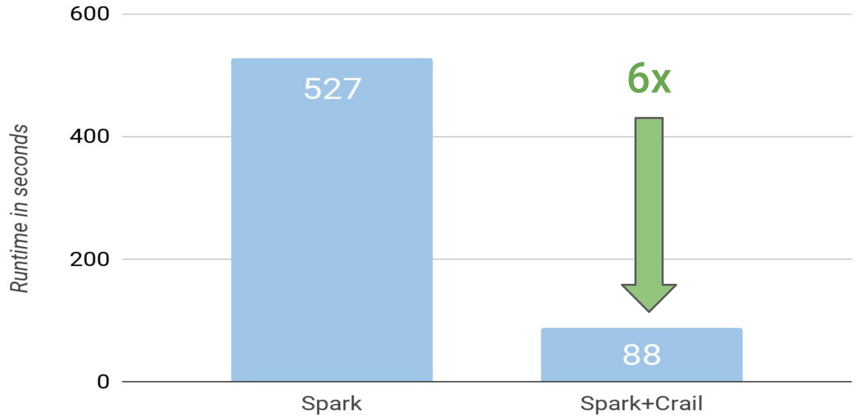
The design space is large, and performance margins are 1-2 orders of magnitude

Paper: Design Guidelines for High Performance RDMA Systems, Usenix 2016

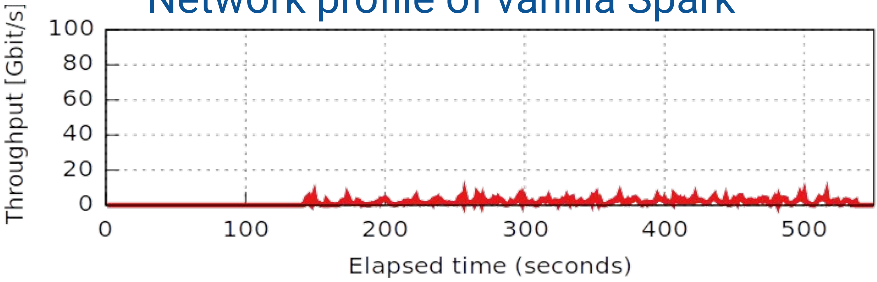
Workload-level Acceleration

Sorting 12.8 TB of data on 128 machines

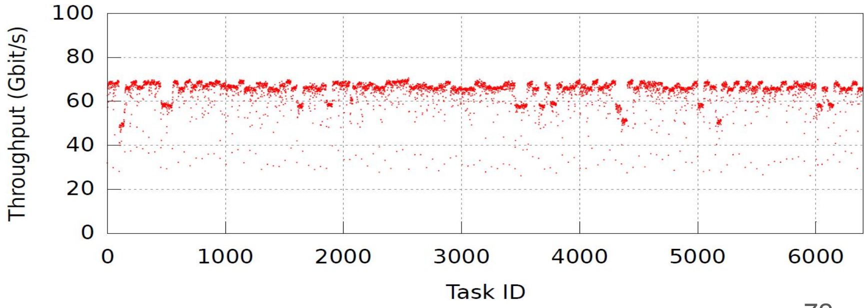
- 100 Gbps network
- 4 x NVMe devices (source and sink)
- Apache Spark



Network profile of vanilla Spark



Optimized Network profile of Spark



http://crail.incubator.apache.org/blog/2017/01/sorting.html

Challenges with RDMA

- Debugging
  - Operation failed, connection down, what went wrong?
  - Logging and introspection can be hard, e.g., log4j, printf -> string manipulation@10s of usec!
- Performance
  - Takes a while to get used to the new way of writing code - event driven, lots of resources
  - Performance isolation (e.g., local PCIe vs remote NIC traffic BUG)
  - Quality of service, traffic management, firewall, filtering, compliance
- Fragility
  - In the cloud (performance vs. flexibility, e.g. VM migration)
  - Correctness and verification (e.g., 32 bit ADD circuit on 64-bit addresses in one RNIC)
  - Small eco-system and vendors
- Scalability
  - How many concurrent socket connections can you support in your server?
  - How many memory buffers an RNIC can remember?

Comment on networking research

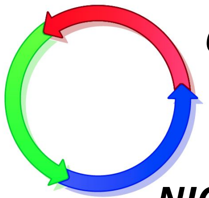
Between all the papers and topics we have discussed

- Networking research has a long history and many ideas are repeated, find their applications in different deployment context (HPC, DC, Internet, Edge, etc.), with different applications

The grand question - *what should be the work division between*

Easy of APIs and abstractions, support, and expressiveness

Generability, multi-tenancy, hardware utilization



hardware complexity, cost, on-chip resources, time for development

Always changing, dictated by the current

- application needs (latency, bandwidth, packets/sec)
- hardware trends (power balance between devices and CPU)
- interplay between OS and hardware

What you should know from this lecture

1. The idea of RDMA
2. Why it becomes possible and what kind of support it expects from the hardware
3. What is userspace networking and kernel bypassing
4. What is the relationship between socket programing and RDMA
5. What advantages does RDMA offer
6. What are its (potential) disadvantages
7. In what kind of setting it should be used
8. A bit of historical context and where today's RDMA network evolved from

Forwarding looking

These are **very exciting times** for systems research (OS, network, storage, devices, architecture):

- Everything is changing - infinite amount of customization and optimization possible
- There are SmartNICs, programmable hardware, GPUs, FPGAs, TPUs
- Computation is becoming more heterogeneous and distributed - *even inside a single machine*

Pushing the boundaries of what is possible with computation:

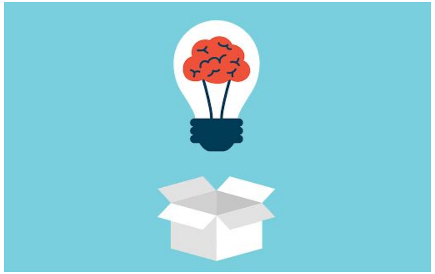
- Fighting for the last nano-micro seconds
- Pushing to deliver millions/billions of packets per second
- Delivering Terabits/sec bandwidths

Also in storage

Remember: *nothing runs in isolation storage, CPU, OS designs, application designs, scheduling, memory management, process management, architecture research ....virtualization (!)*

Time to think outside the box

What kind of questions researchers are tackling?



- *How do you write software, cloud services for such infrastructure?*
- *What is the right API and abstraction? Files and sockets?*
- *How do you debug at this speed? Failures?*
- *Is C still a good choice? (Rust, Go?)*
- *How do you write a portable software with so much hardware dependencies?*
- *Is Linux a good operating system for such diverse, heterogeneous hardware setup?*
- *How do we integrate network + storage + GPU computation in a single application?*
- *How do we support multiple applications with different performance needs? Multi-tenancies?*
- *How do we integrate such devices in the cloud, edge computing?*
- *What applications can we run on this infrastructure - AI/ML, Bioinformatics, Astronomy, Banking, Precision Agriculture, autonomous driving?*
- *Do we need all that generality if we are just running ML/AI on our cluster?*

Speaking of ML/AI ...

http://www.keypersonofinfluence.com/how-can-we-think-outside-the-box-from-inside-the-box/

WILL KNIGHT BUSINESS 01.21.2020 07:00 AM

AI Can Do Great Things—if It Doesn't Burn the Planet

The computing power required for AI landmarks, such as recognizing images and defeating humans at Go, increased 300,000-fold from 2012 to 2018.

LAST MONTH, RESEARCHERS at OpenAI in San Francisco revealed an algorithm capable of learning, through trial and error, how to manipulate the pieces of a Rubik's Cube using a robotic hand. It was a remarkable research feat, but it required more than 1,000 desktop computers plus a dozen machines running specialized graphics chips crunching intensive calculations for several months.

The effort may have consumed about 2.8 gigawatt-hours of electricity, estimates Evan Sparks, CEO of Determined AI, a startup that provides software to help companies manage AI projects. That's roughly equal to the output of three nuclear power plants for an hour. A

*This is not to say that AI/ML is bad, but to point out that we have an opportunity to **smartly design hardware/software** to deliver efficiency*

- *Cost*
- *Energy*
- *Sustainability*

Recommended Reading

Animesh Trivedi, *End-to-End Considerations in the Unification of High-Performance I/O*, PhD thesis, ETH Zurich, January, 2016.

https://doi.org/10.3929/ethz-a-010651949

Chapter 2, Evolution of High-Performance I/O

Chapter 3, Remote Direct Memory Access (example and details)