An Introduction to RDMA Networking

Animesh Trivedi
a.trivedi@vu.nl
https://animeshtrivedi.github.io/

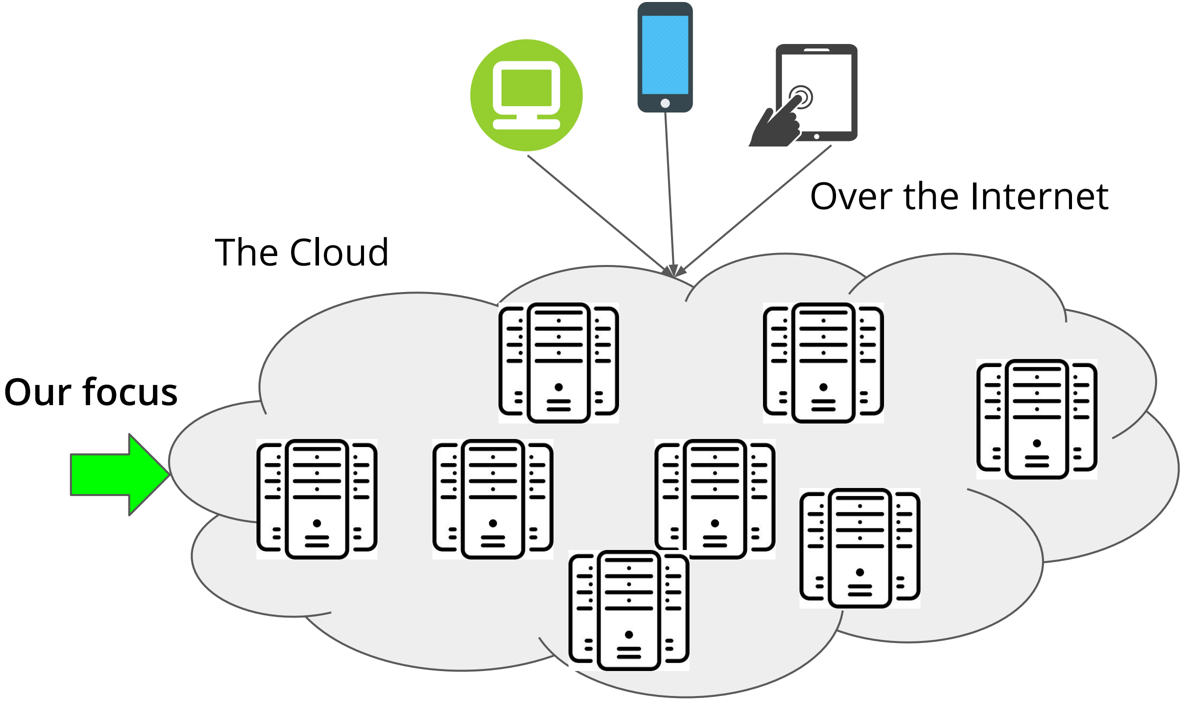
It is an Advanced Topic

- Networking knowledge
- Operating system knowledge
- CPU and architecture knowledge
- Low-level implementation details ...

Agenda

1. A closer look at the socket networking
2. Challenges with the classical/socket networking
3. The idea of User-space networking
4. Remote Direct Memory Access (RDMA) technology
    a. Performance
5. RDMA applications
6. Hands on experiments

A Basic Cloud Service Setup



A closely installed setup with 1000s of machines connected with high-performance network (e.g., the DAS-5 platform)

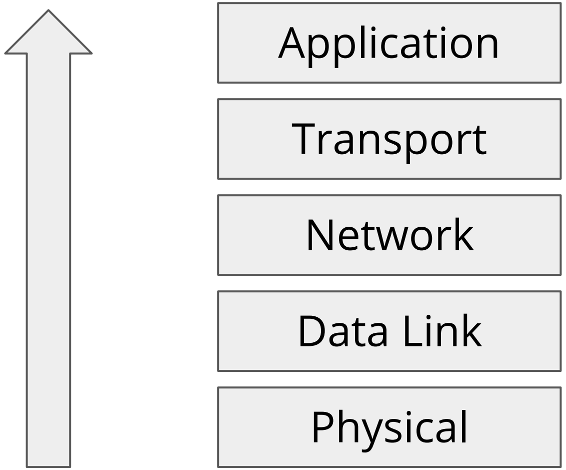
Google, Facebook, Microsoft and other big companies have large data center installation where they run services like Youtube, Search, Social platforms etc.

Example - (Rough) Datacenter Regions

How does our network work?

The Layered Model

Modularity: Layer by layer architecture where one layer provides service to the next one



Network access

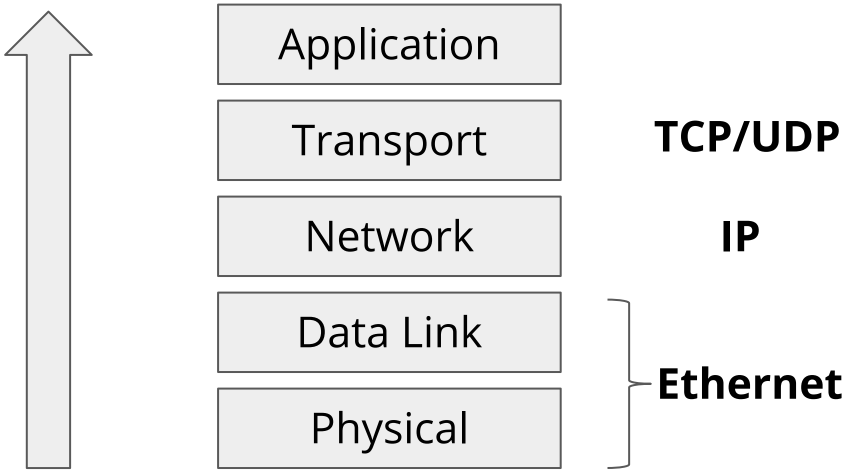
End-to-end delivery

Global routing and best-effort delivery

Link management / local area delivery

Physical bits/voltage/current movement

The Layered Model - Protocols



Some examples, but they are not the only one

How do applications use the network?

We use a network application programming interface (or API)

One example is a **Socket interface**

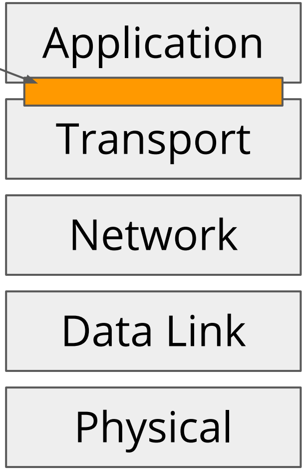
```
int sock = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in serv_addr;
[...]
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);
[...]
connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr);
// send data
send(sock , “Hello World!”, 12, …);
//recv data
recv(sock, buffer, 1024);
```

How do applications use the network?

We use a network application programming interface (or API)

One example is a **Socket interface**

```
int sock = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in serv_addr;
[...]
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);
[...]
connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr);
// send data
send(sock , “Hello World!”, 12, …);
//recv data
recv(sock, buffer, 1024);
```



History : Socket Interface

- One of the first implementations in the 4.2BSD Unix (1983)

1983!

MICROSOFT®
Microsoft Word
Version 1.15
Microsoft Word:

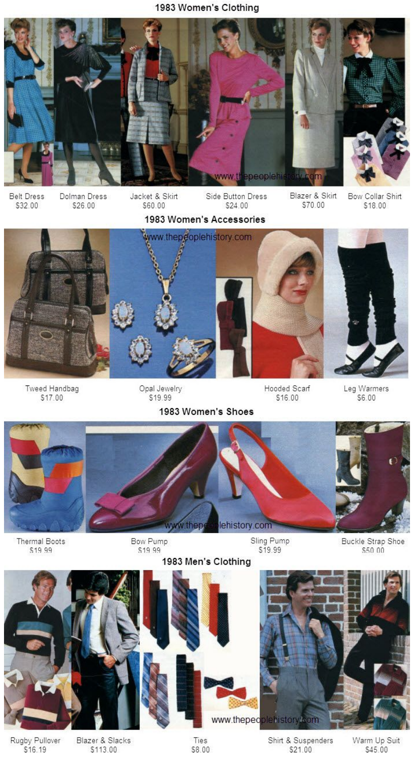
MARIO BROS.
★ MARIO BROS., BATTLE THE PESTS! TWO PLAYERS MAKE IT EASIER.

1983 Women's Clothing
Belt Dress $32.00
Dolman Dress $26.00
Jacket & Skirt $60.00
Side Button Dress $24.00
Blazer & Skirt $70.00
Bow Collar Shirt $18.00
1983 Women's Accessories
Tweed Handbag $17.00
Opal Jewelry $19.99
Hooded Scarf $16.00
Leg Warmers $6.00
1983 Women's Shoes
Thermal Boots $19.99
Bow Pump $19.99
Sling Pump $19.99
Buckle Strap Shoe $50.00
1983 Men's Clothing
Rugby Pullover $16.19
Blazer & Slacks $113.00
Ties $8.00
Shirt & Suspenders $21.00
Warm Up Suit $45.00

History : Socket Interface

- One of the first implementations in the 4.2BSD Unix (1983)
    - It is 36 years old
    - In 1983: Microsoft Word is first released
    - In 1983: Mario Bros. was first released as a Nintendo arcade game
    - In 1983: First mobile phones from Motorola

Modern derivatives: WinSock, BSD socket, POSIX socket, more



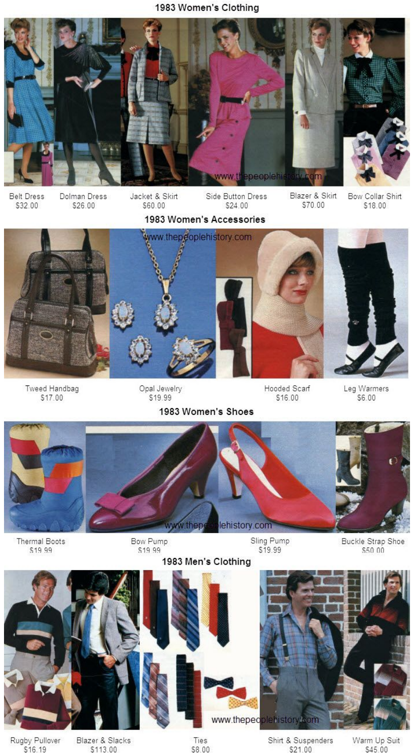
History : Socket Interface

- One of the first implementations in the 4.2BSD Unix (1983)
  - It is 36 years old
  - In 1983: Microsoft Word is first released
  - In 1983: Mario Bros. was first released as a Nintendo arcade game
  - In 1983: First mobile phones from Motorola

  Modern derivatives: WinSock, BSD socket, POSIX socket, more

- First reference RFC #147 (`The Definition of a Socket, 1971`) `https://tools.ietf.org/html/rfc147` `(only 2 pages)`

- Follows the UNIX philosophy
  - *Everything is a file* (a socket is a file descriptor)



Socket Interface - The Unofficial Standard

- Setting up and managing connections
  - `socket(), bind(), listen(), connect(), accept(), close()`
- Network operations
  - `send(), recv(), sendto(), and recvfrom()` (or `write()` and `read()` may also be used)
- Address/hostname management
  - `gethostbyname()` and `gethostbyaddr()` to resolve IPv4 host names and addresses
- Select activity and readiness of a socket for I/O
  - `select(), poll()`
- Setting up extra options
  - `getsockopt()` and `setsockopt()`

Not a complete list. There are OS specific (e.g., Linux) specific extensions.

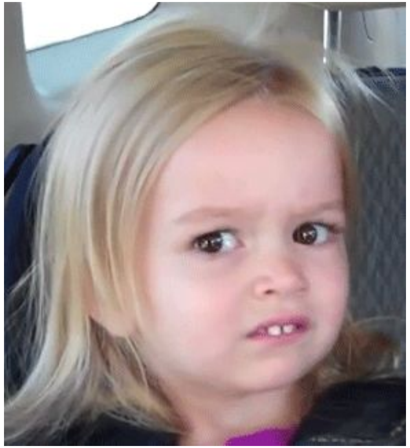
Socket - A Highly Successful Abstraction

- Socket is a **very successful abstraction**
  - A UNIX file with a bunch of basic functions
  - Applications are shielded away from managing anything but just "what to send" and "where to receive"
    - `send(int socket, void* buffer_address, size_t length, int flags);`
    - `recv(int socket, void* buffer_address, size_t length, int flags);`

- Worked **extremely well** all these years supporting different classes of applications
  - Web servers, video streaming, messaging applications, _your_favorite_application

But Wait...

- where are the rest of the networking layers?
- what happens after calling send / recv functions?
- who is running the TCP state machine?
- who is managing the TCP window and retransmission?
- who is doing IP routing?
- who is doing the MAC layer management?
- ...and so many more question



The Answer is …

The Operating System

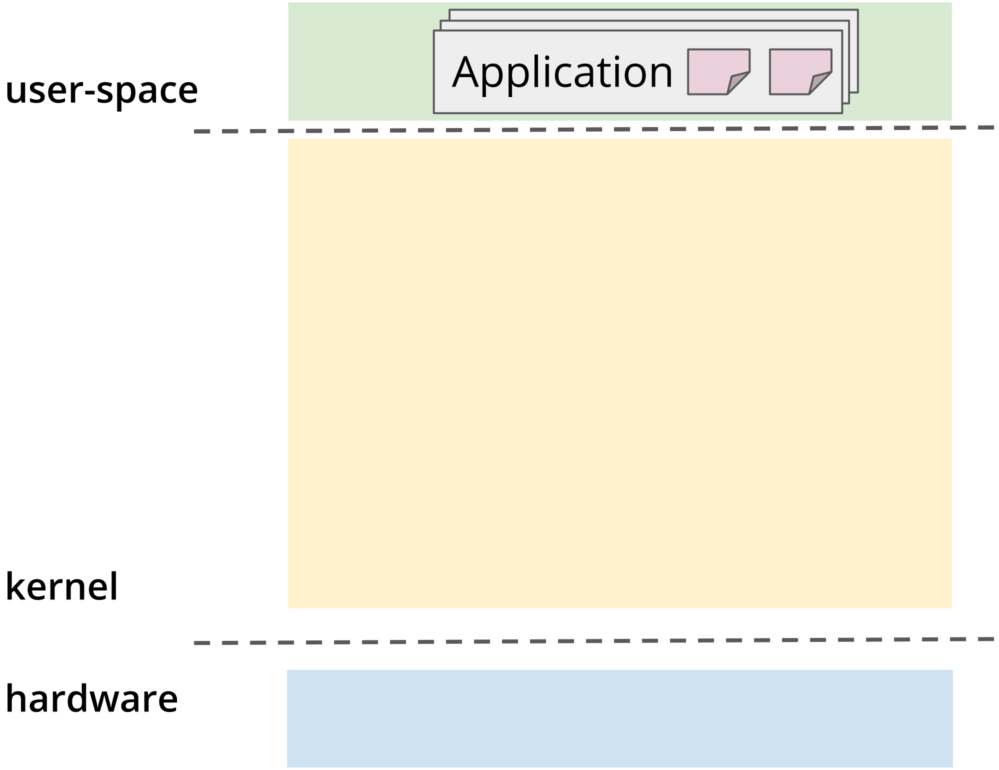
- Linux, Windows, Open/Free/NetBSD, Minix - whatever you are running

Why?

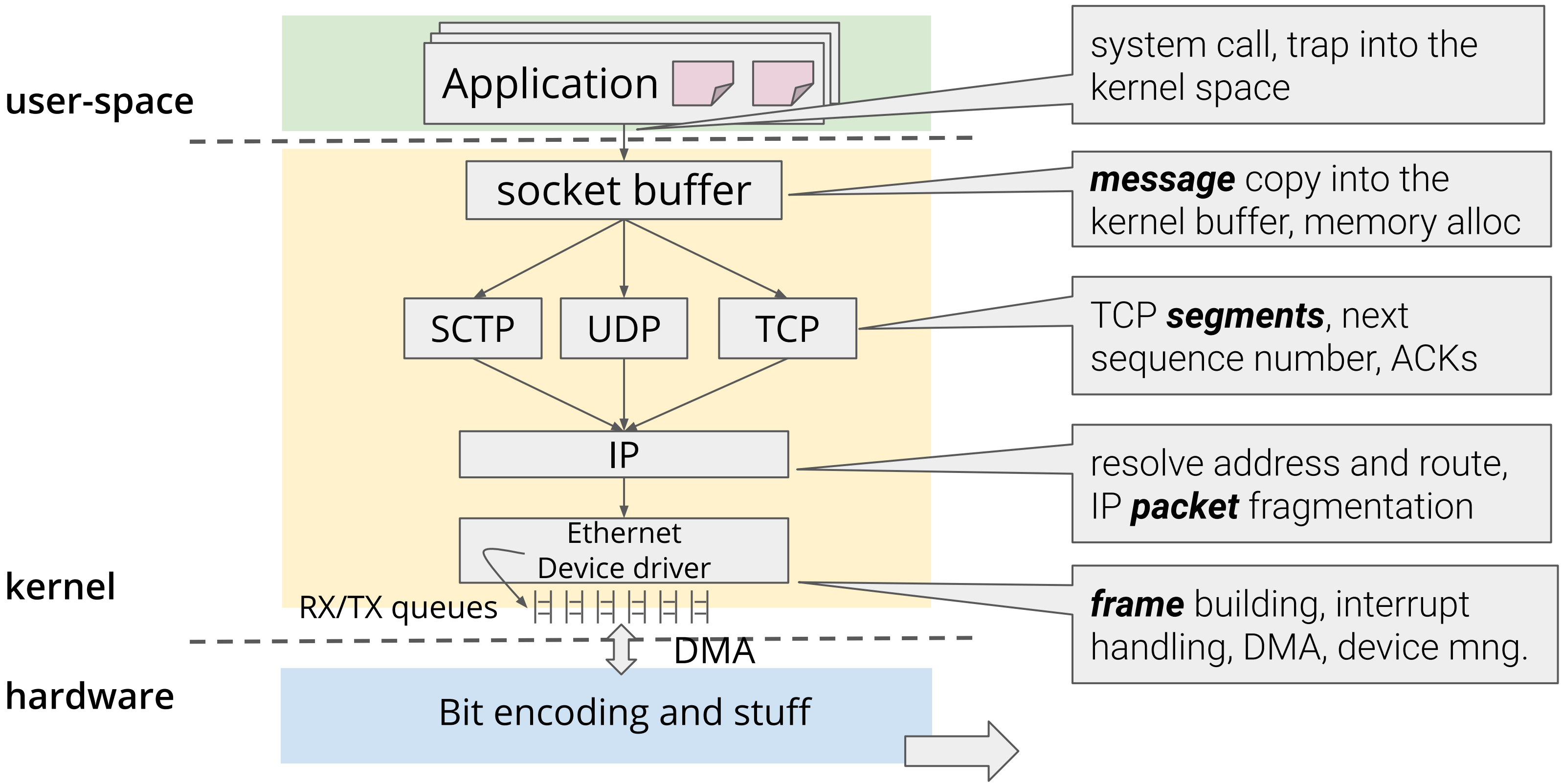
- Network connectivity is an important shared resource for all
- Every networking application benefits from a common implementation

`As we will see later, this is _NOT_ THE only way to arrange things`

All Together - sending a message



All Together - sending a message



All Together - receiving a message

Almost the same

- But in the other direction
- More difficult
    - Asynchronous nature of receive
    - Interrupts and polling

There are more, and much more details (*networking is _NOT_ the only thing that the OS is doing*)

- Process, memory, storage, user management

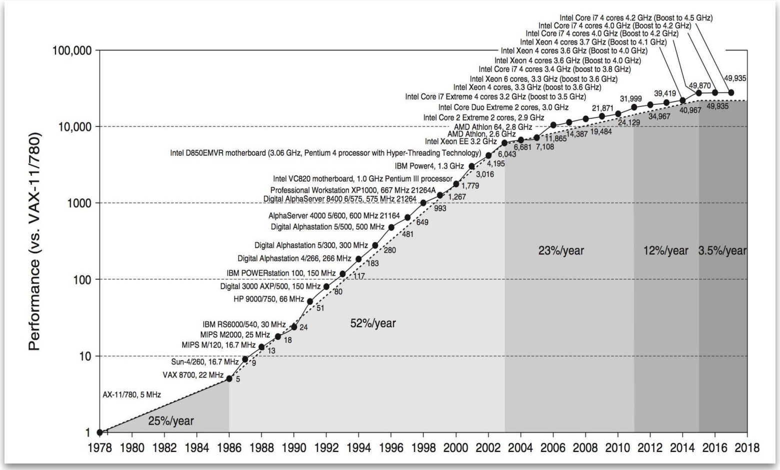
Key Message: There is a bit of work involved in doing network operations!

Agenda

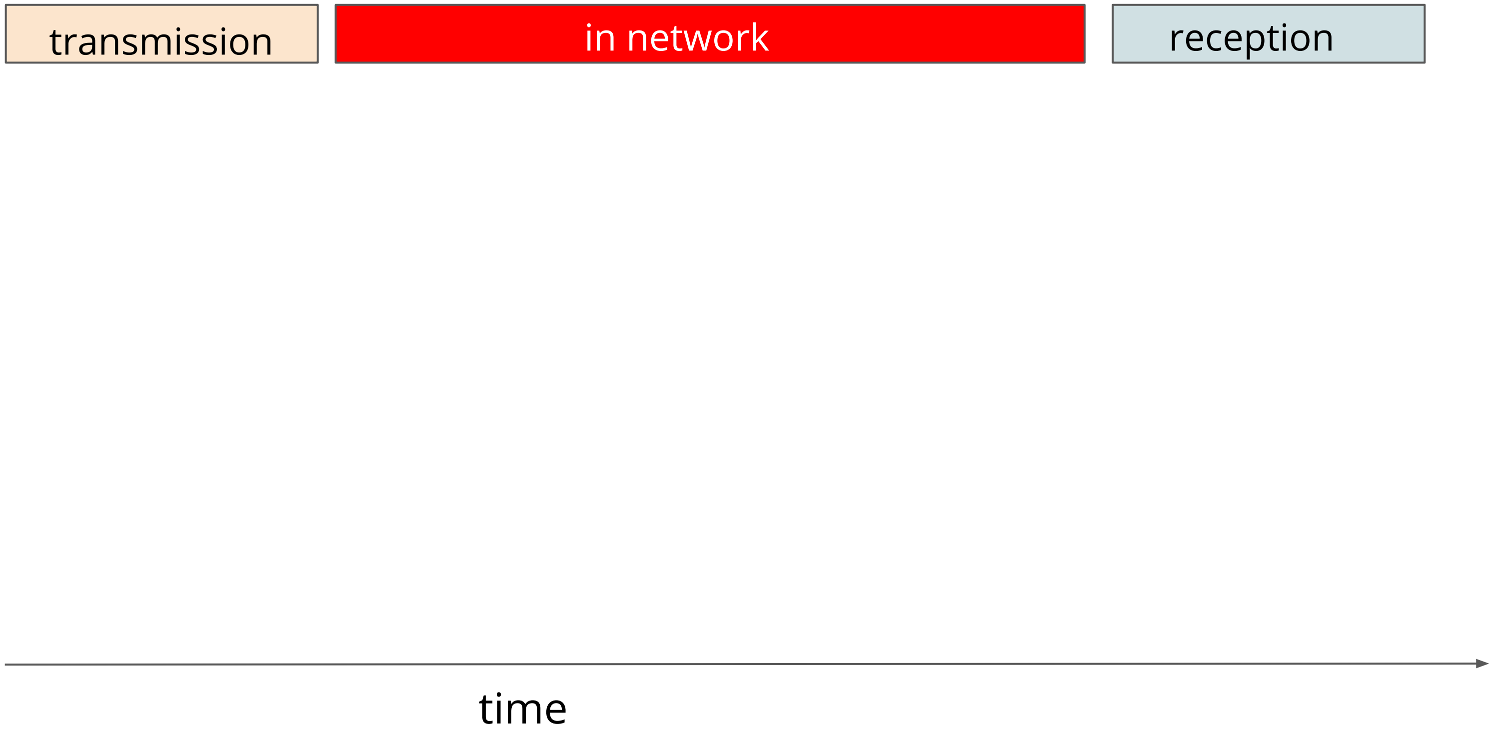
1. ~~A closer look at the socket networking~~
2. Challenges with the classical/socket networking
3. The idea of User-space networking
4. Remote Direct Memory Access (RDMA) technology
    a. Performance
5. RDMA applications
6. Hands on experiments

What is the Challenge?

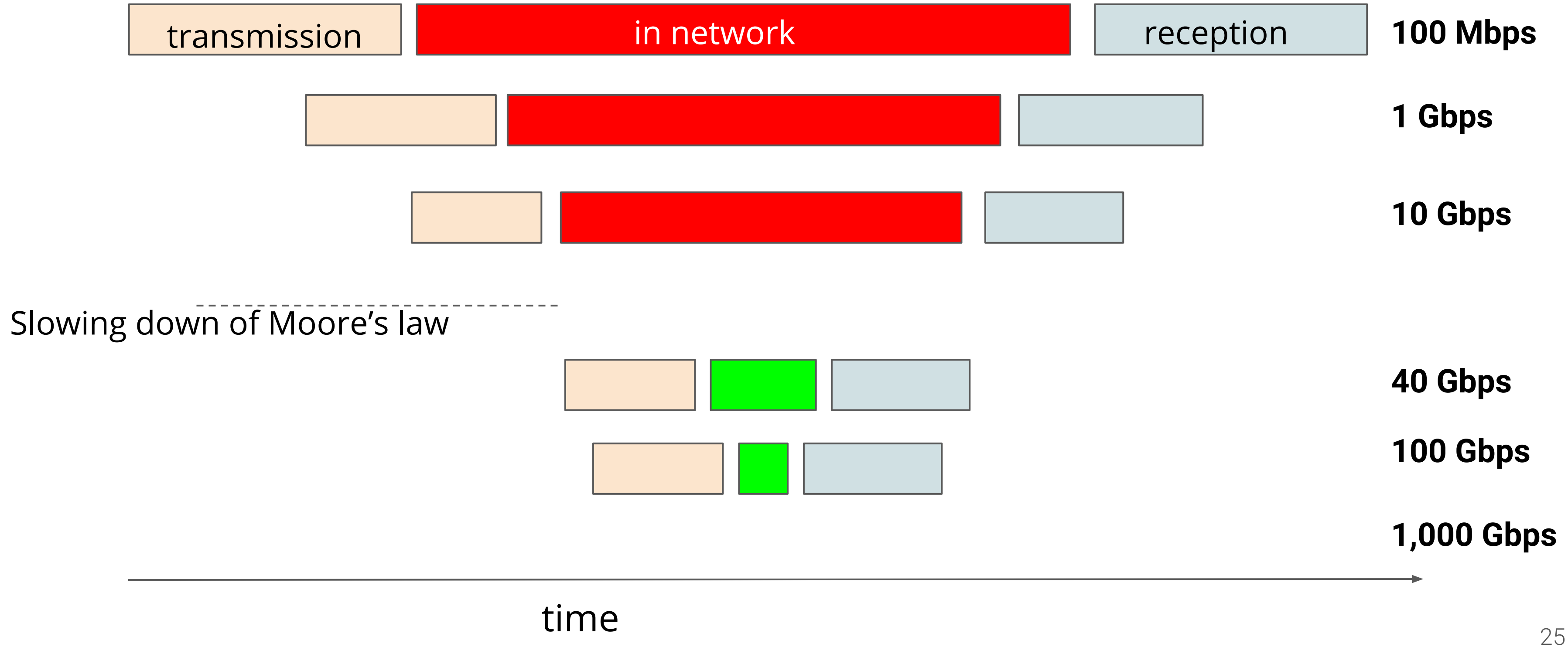
- Everything runs on the CPU
  - Applications, threads, processes
  - the operating system kernel
- **But the CPU is not getting any faster**
  - CPU was getting faster due to Moore's Law
- **But the network speeds are...**
  - 1 to 10, and now 100 Gbps
  - 200 and 400 Gbps are now available
  - **Be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



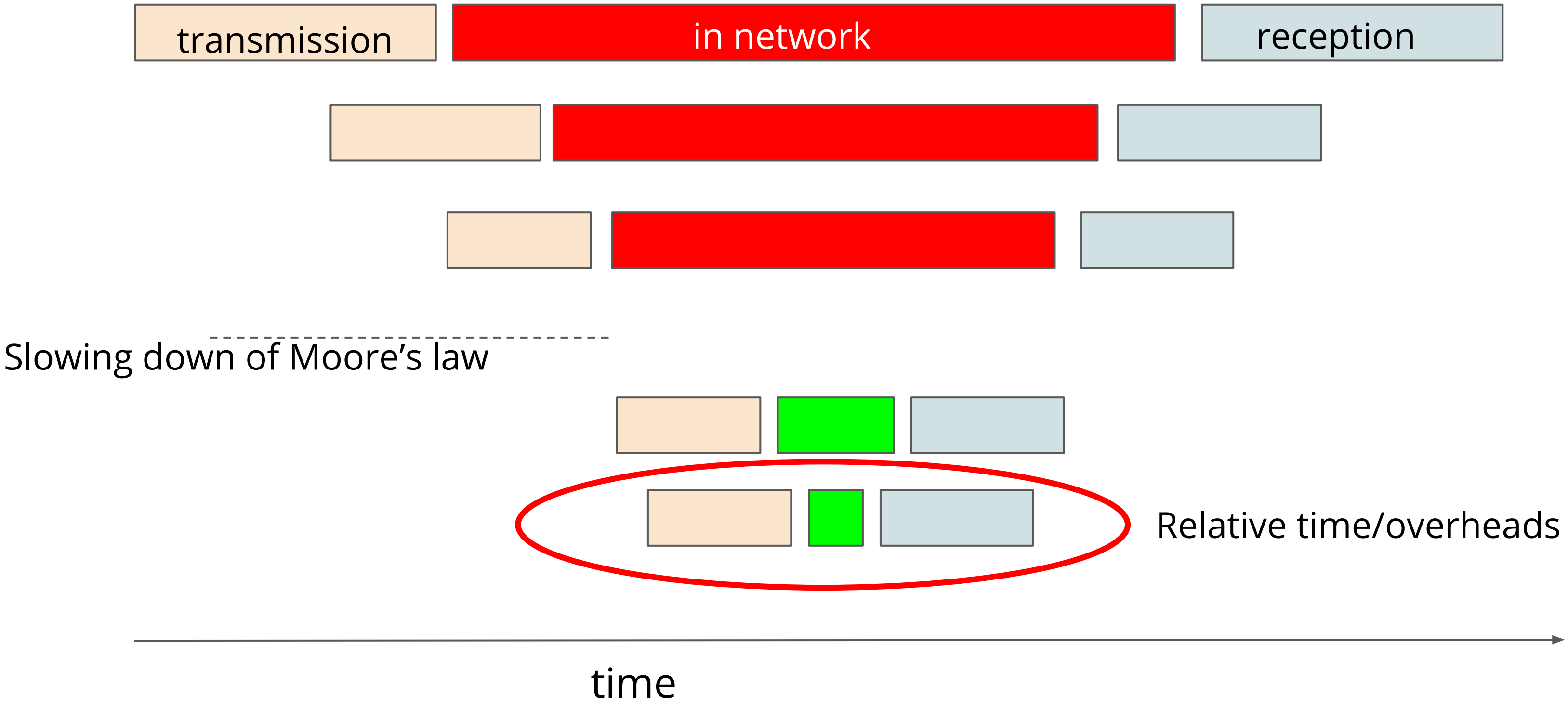
Shifting Performance Bottlenecks



Shifting Performance Bottlenecks



Shifting Performance Bottlenecks



Hardware Trends

	1980s	late 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

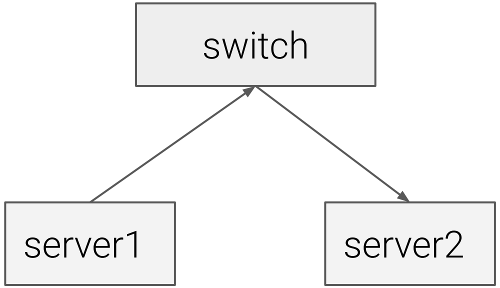
Trend 1: Network is getting faster

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

“Network performance is *increasingly* a software/CPU factor”

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:

(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)

Packet Rates

The smallest frame size on Ethernet is : 84 bytes (including all overheads)

- at 10 Gbps -> 67.2 ns between packets (14.88 Mpps)
- at 100 Gbps -> 6.72 ns between packets (148.8 Mpps)

Think about it, for a typical CPU

- L1$ = 1 ns, L2$ = 5 ns, LLC$ = 30 - 40 ns, DRAM = 60 - 100 ns
- How many cache misses can you afford on a 100 Gbps network?

https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html
https://people.netfilter.org/hawk/presentations/devconf2016/net_stack_challenges_100G_Feb2016.pdf

Is this all Socket's problem?

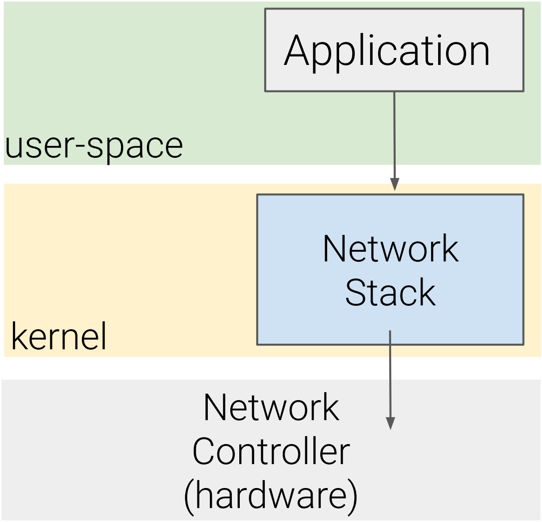
- Socket is an application-level interface - it does not say anything about lower layer protocols
- However, its simplistic design restricts many optimization opportunities to reduce the amount of work done for a network operation:
  - Ties to the OS "**process**" abstraction
    - Everything (i.e. the 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"
  - When to notify application about I/O completions and network events and how
  - **Blocking, non-blocking, synchronous, asynchronous I/O interfaces**

Agenda

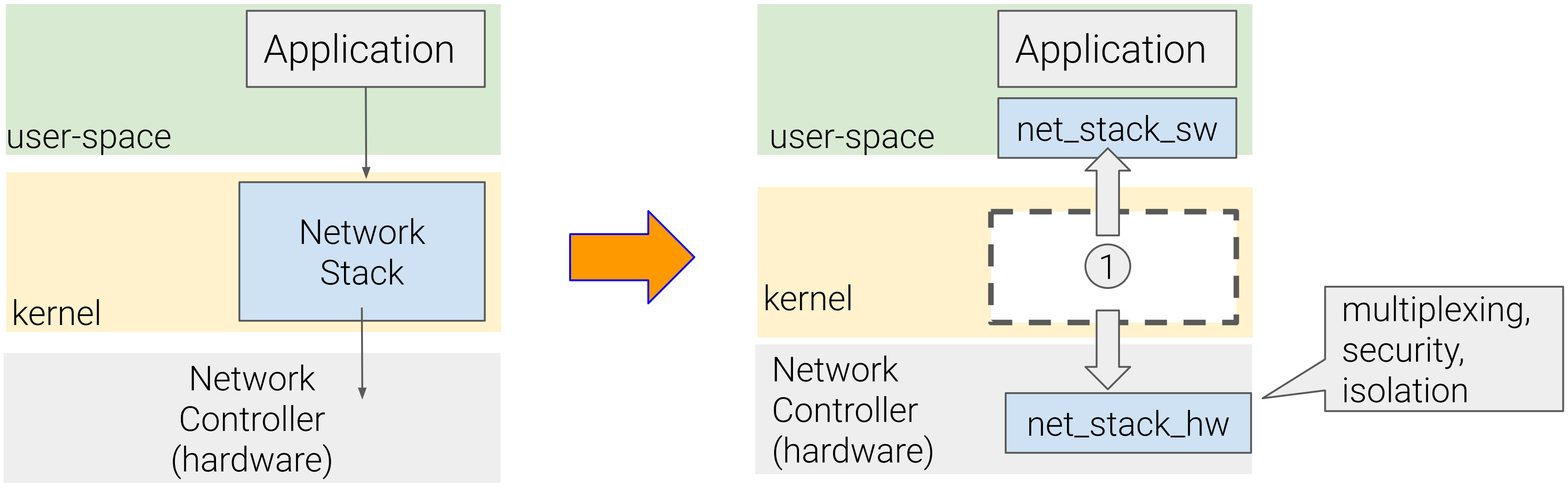
1. ~~A closer look at the socket networking~~
2. ~~Challenges with the classical/socket networking~~
3. The idea of User-space networking
4. Remote Direct Memory Access (RDMA) technology
    a. Performance
5. RDMA applications
6. Hands on experiments

Remote Direct Memory Access (RDMA)

Idea 1: User-space Networking

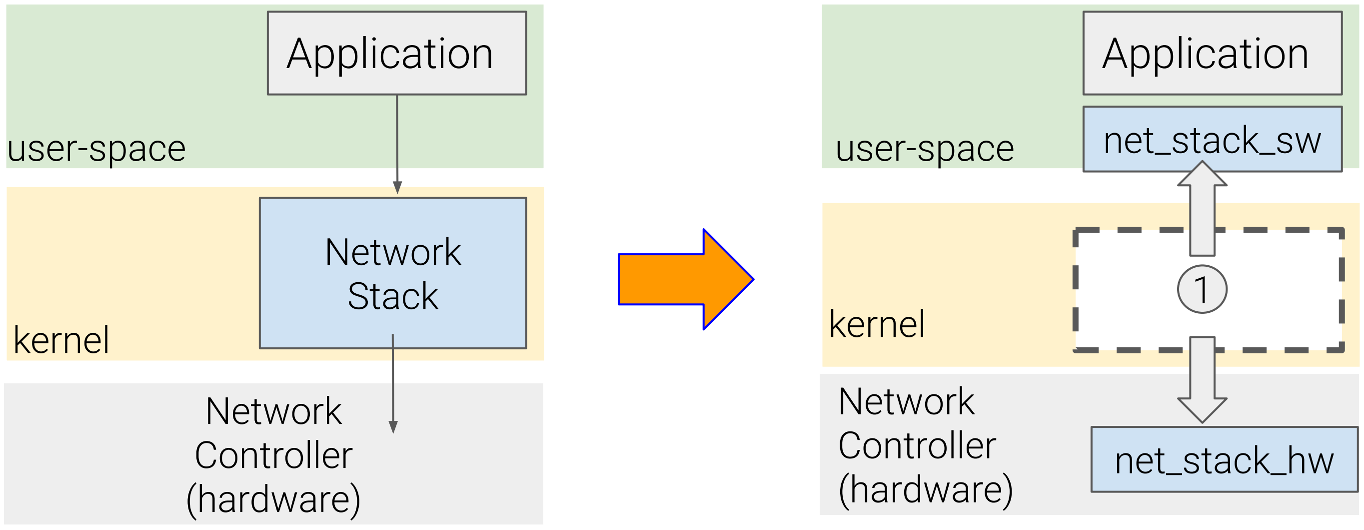


Idea 1: User-space Networking



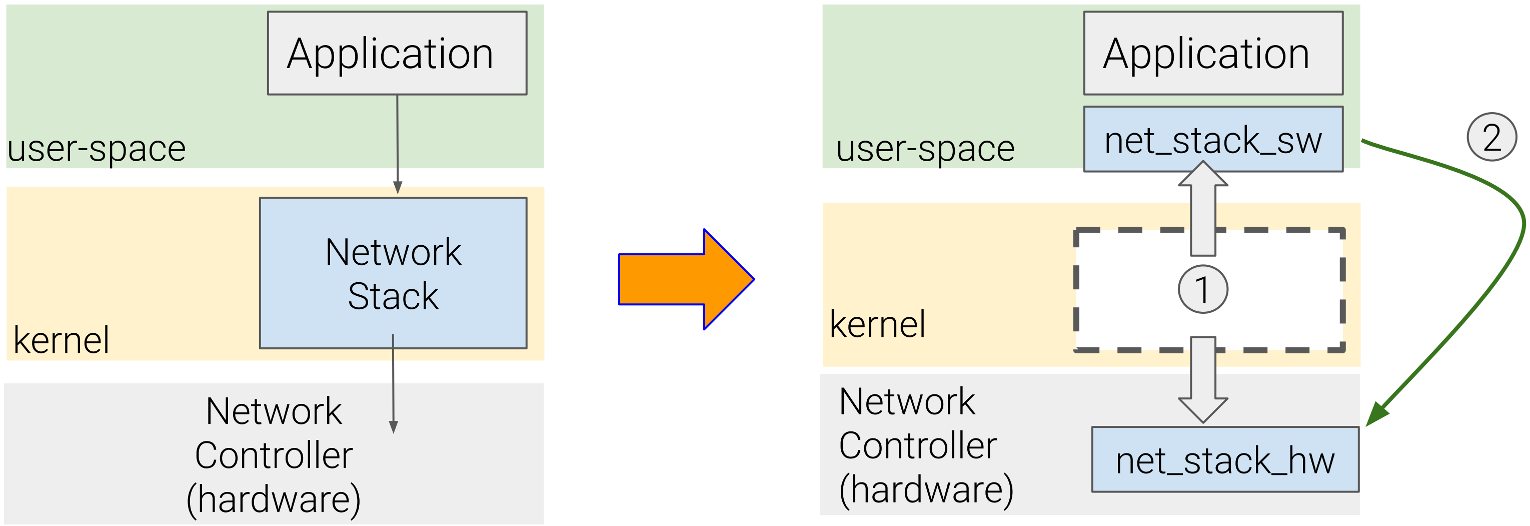
① **User-space networking** : Let the process manage its networking resources

Idea 1: User-space Networking



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

Idea 2: Kernel Bypass

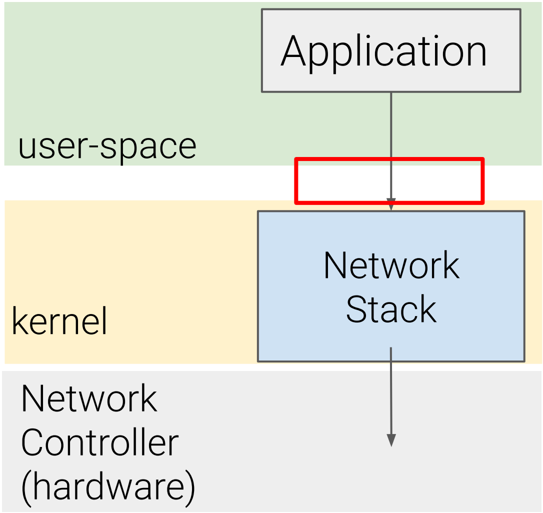


1 **User-space networking** : Let the process manage its networking resources

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

New Abstraction?

What is the right abstraction? Socket?

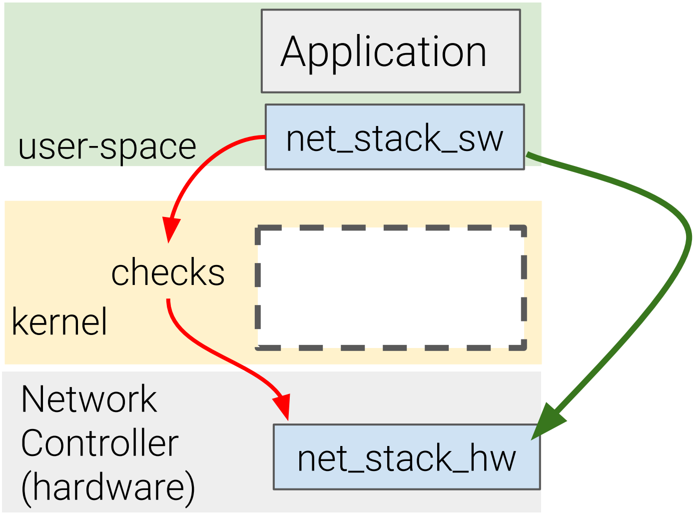


send (int sockfd, void *buf, size_t len, int flags);

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 → **Data operation**
2. Allocate needed memory
3. Manage buffers
4. Schedule process (if necessary)
5. And everything else …

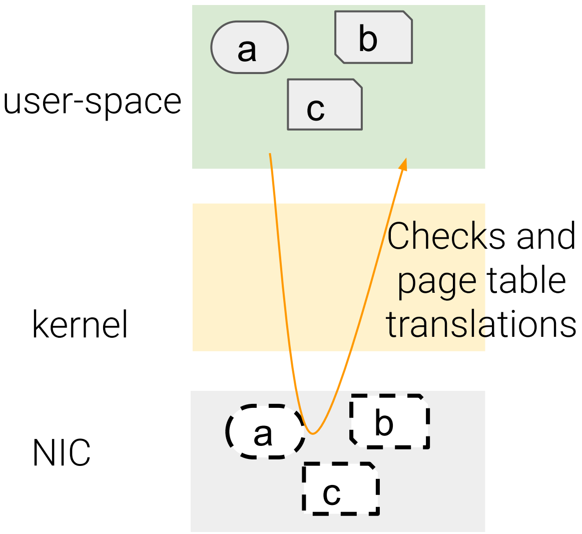
} **Control operations**

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

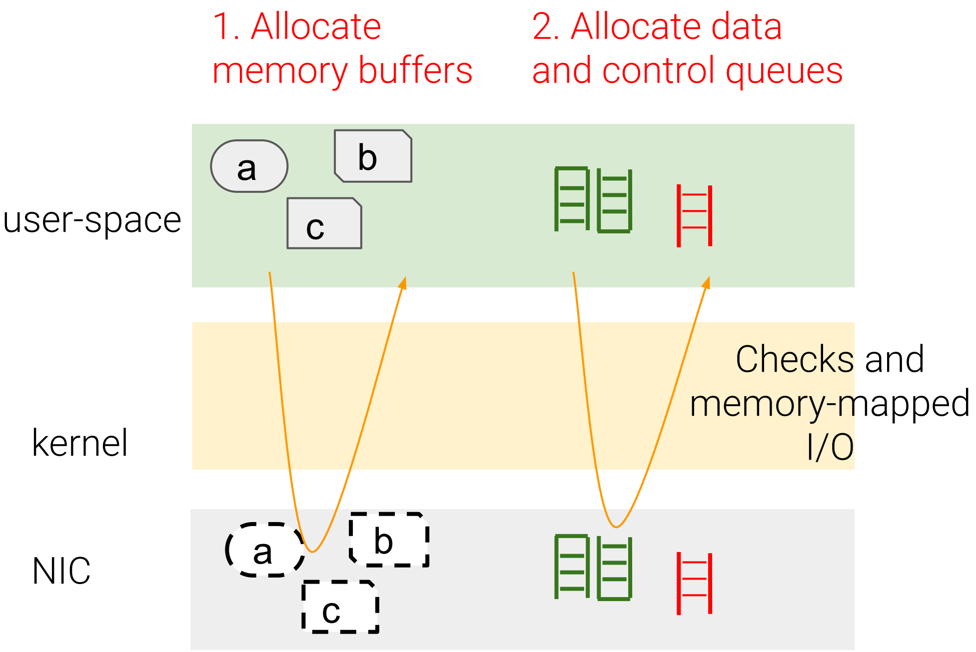
needs new interfaces

Putting All Together - User-Space Networking

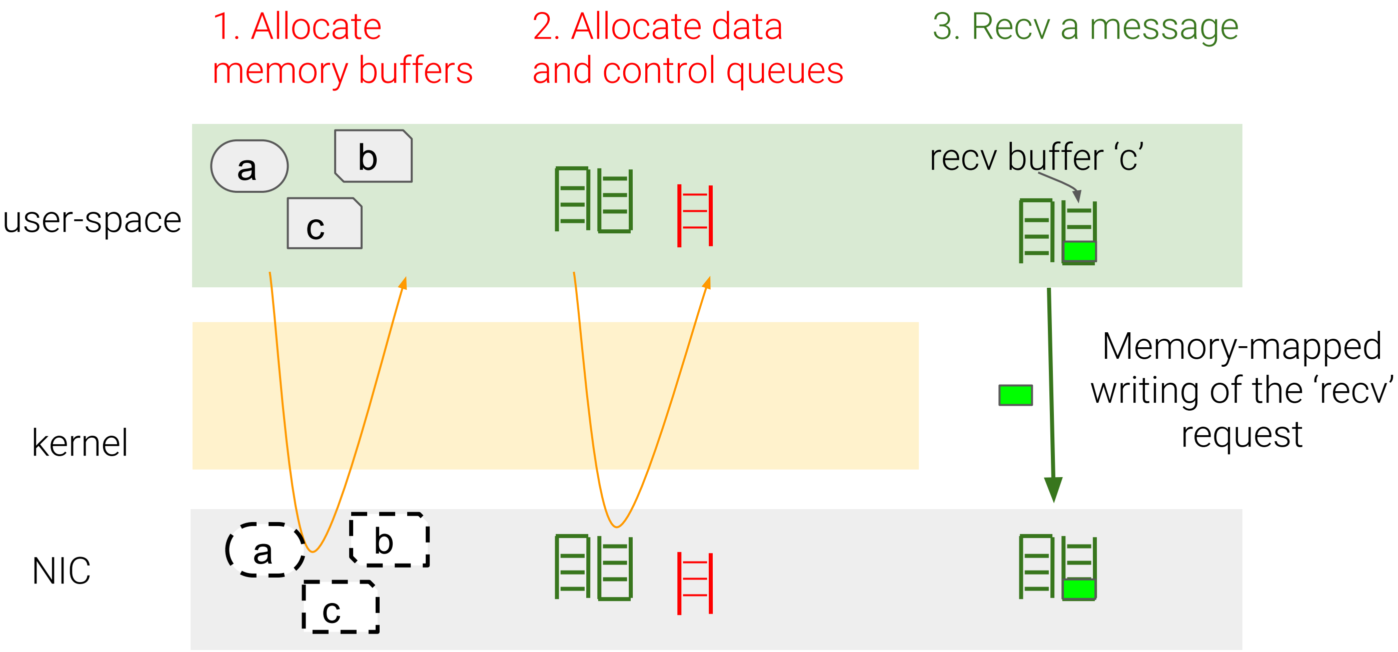
1. Allocate memory buffers



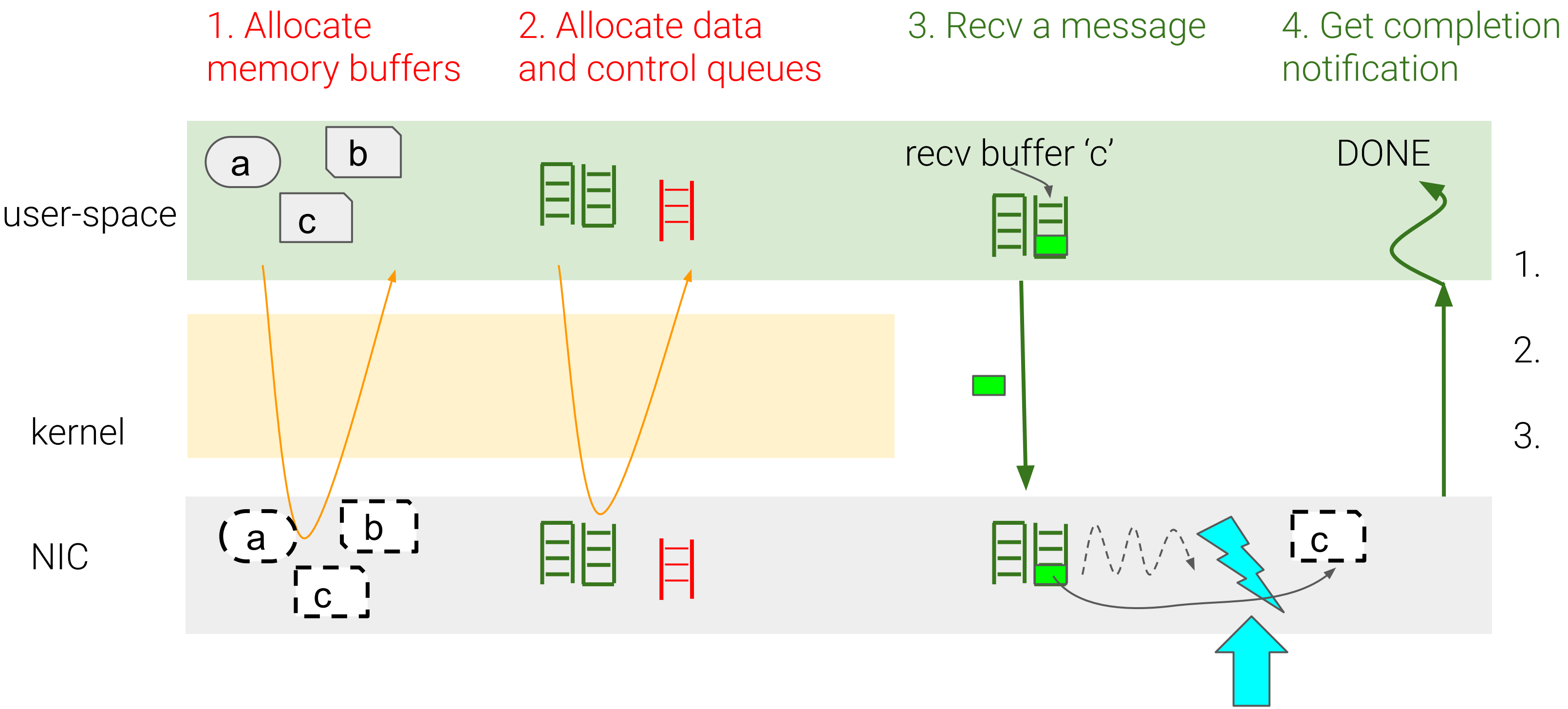
Putting All Together - User-Space Networking



Putting All Together - User-Space Networking

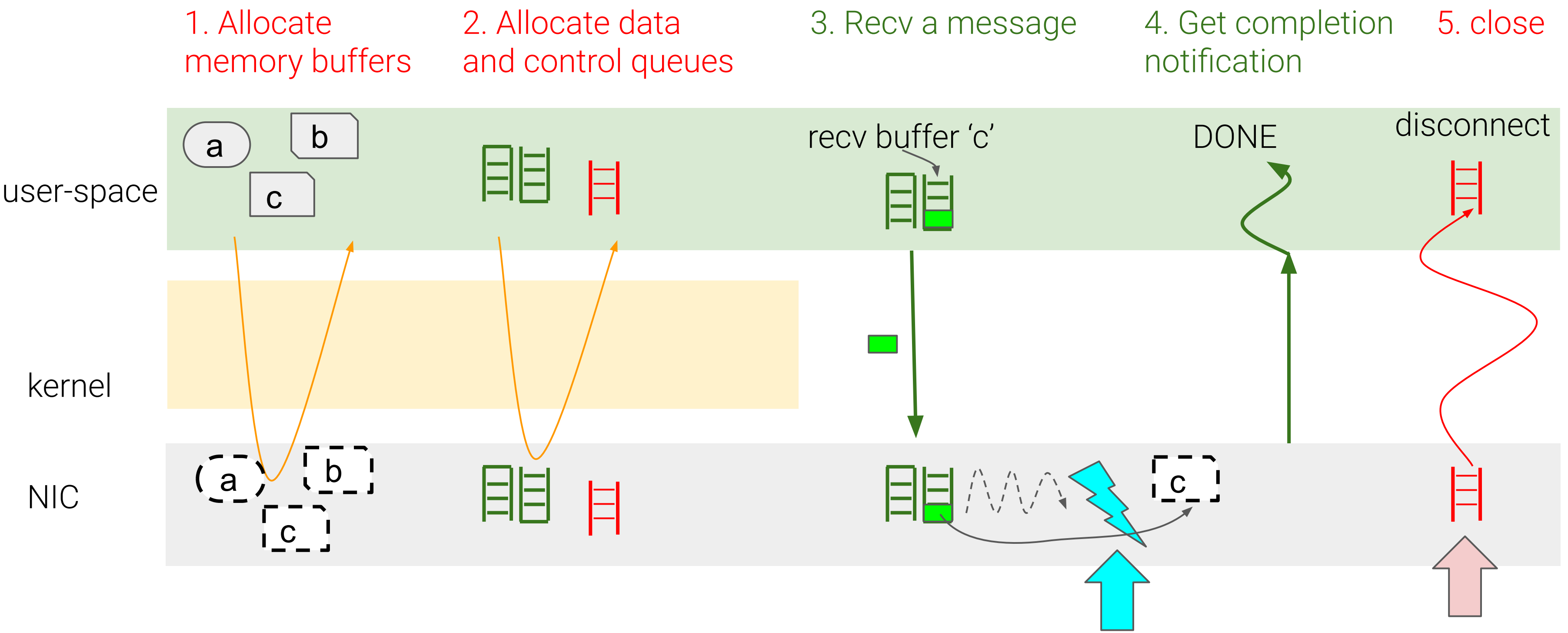


Putting All Together - User-Space Networking



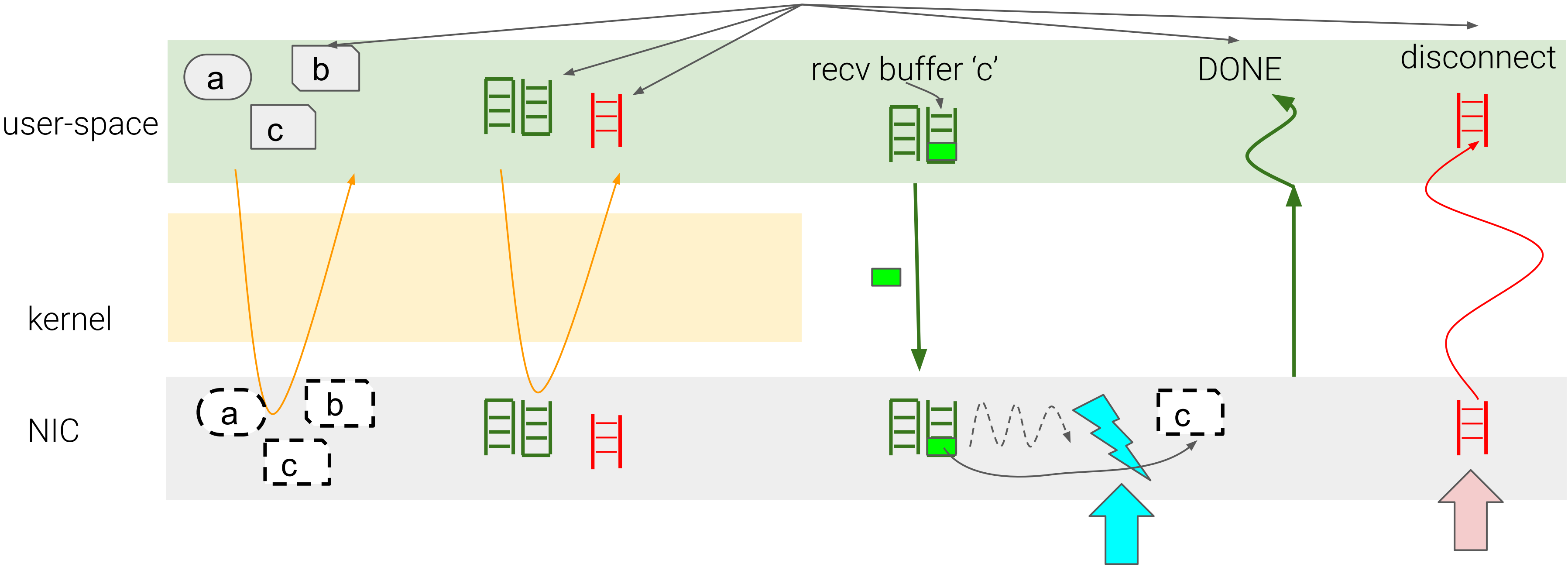
1. Network processing,
2. DMA to the buffer 'c',
3. Notification to the application

Putting All Together - User-Space Networking



Putting All Together - User-Space Networking

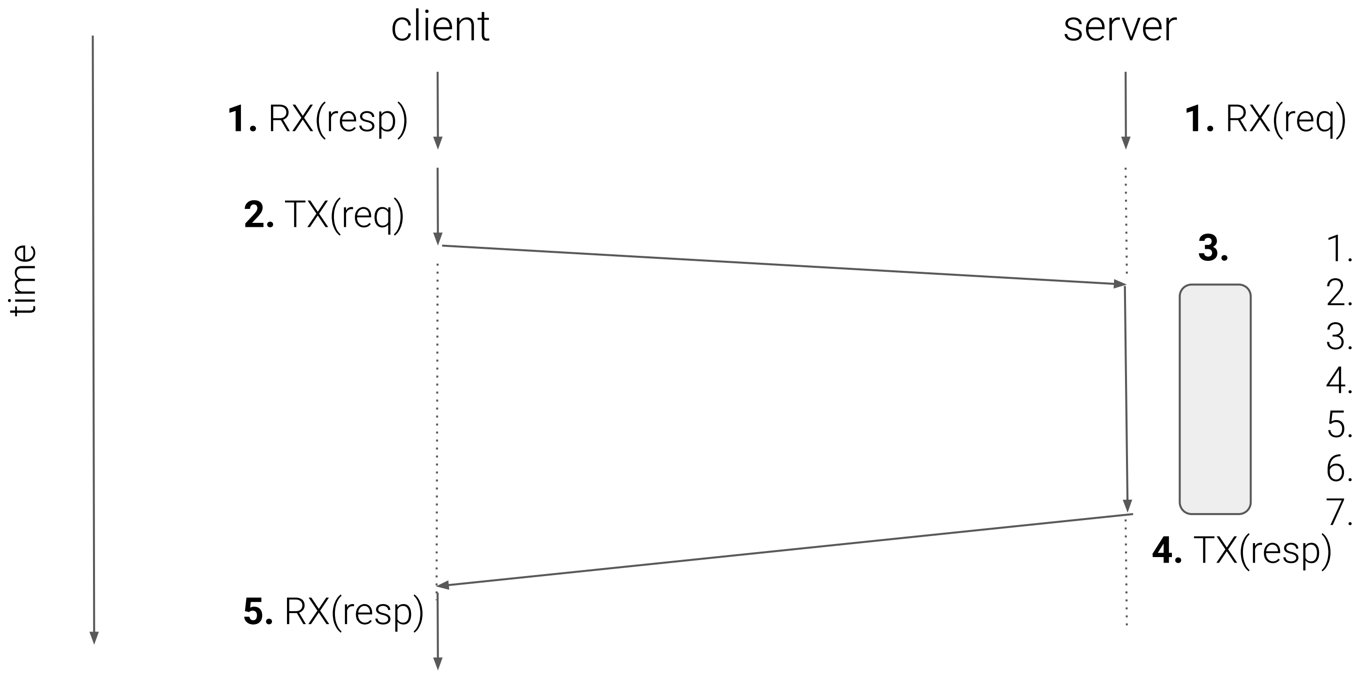
What new abstractions, objects, do you see here?



Agenda

1. ~~A closer look at the socket networking~~
2. ~~Challenges with the classical networking~~
3. ~~User-space networking~~
4. Remote Direct Memory Access (RDMA)
    a. Performance
5. RDMA applications
6. Hands on experiments

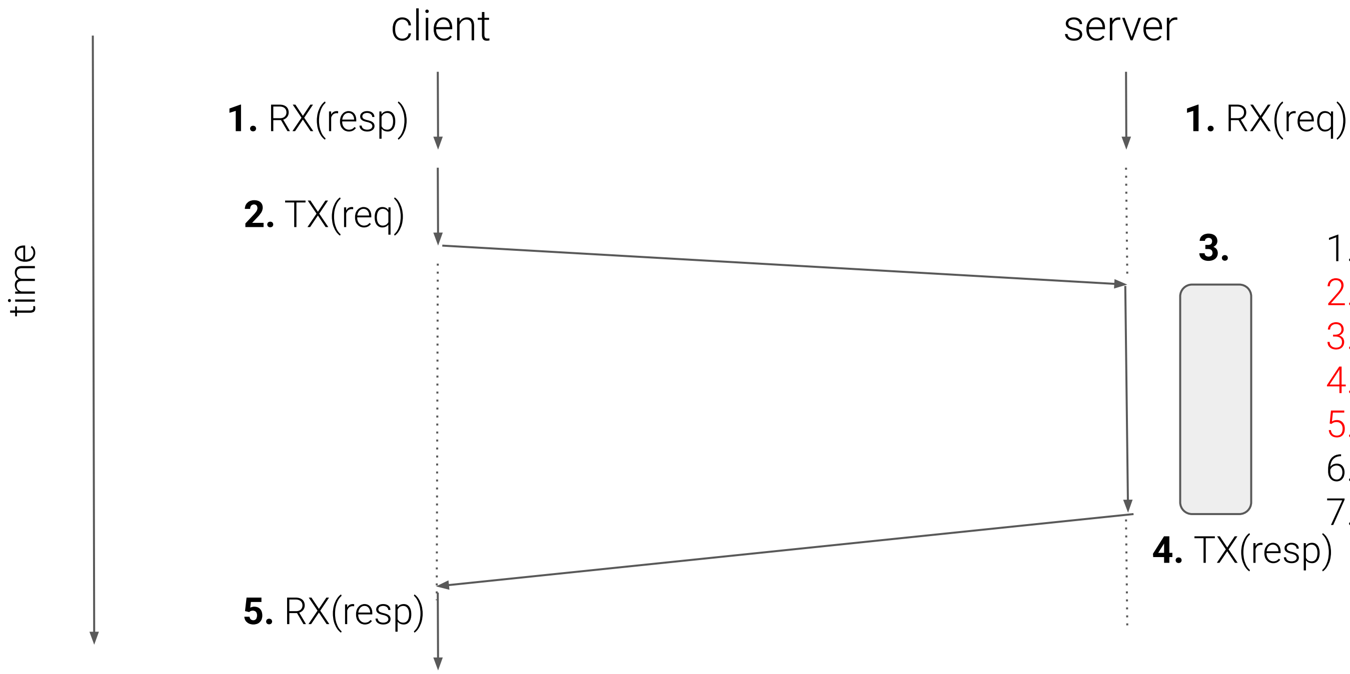
Timeline of a send/recv Exchange



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

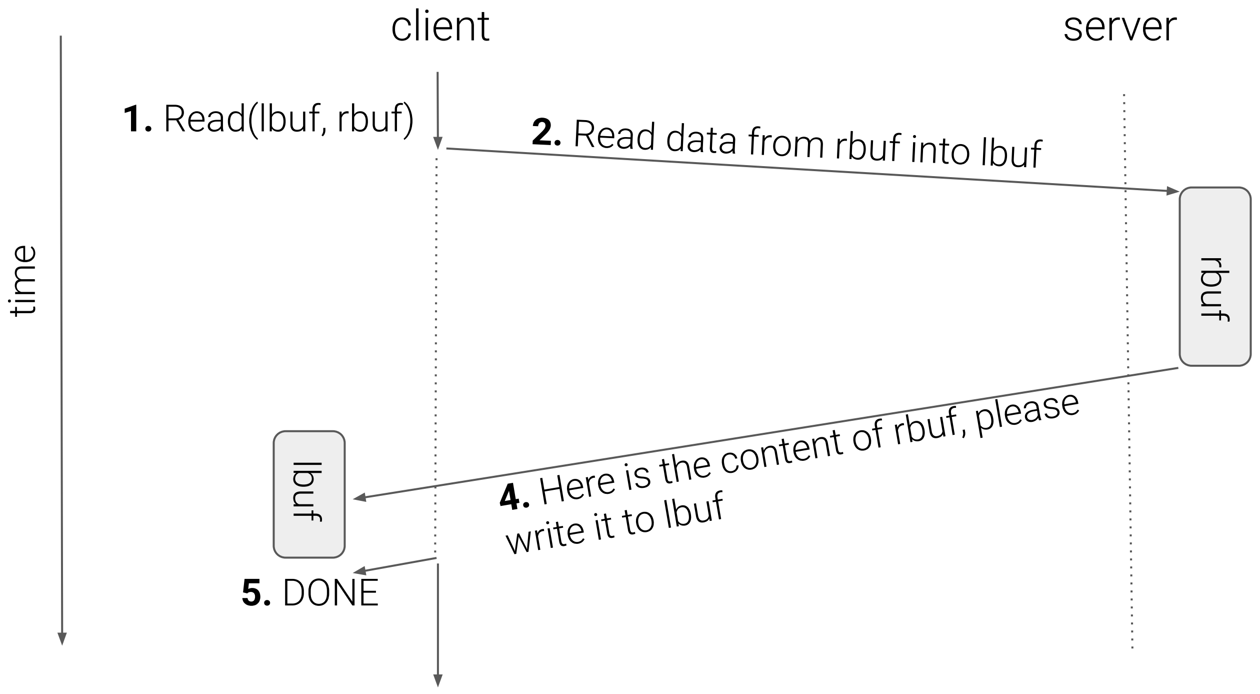
Two-sided message exchange between processes

Can we eliminate server entirely?

The Idea of Remote Direct Memory Access

- Imagine, if all buffers are known up front to everyone
- 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

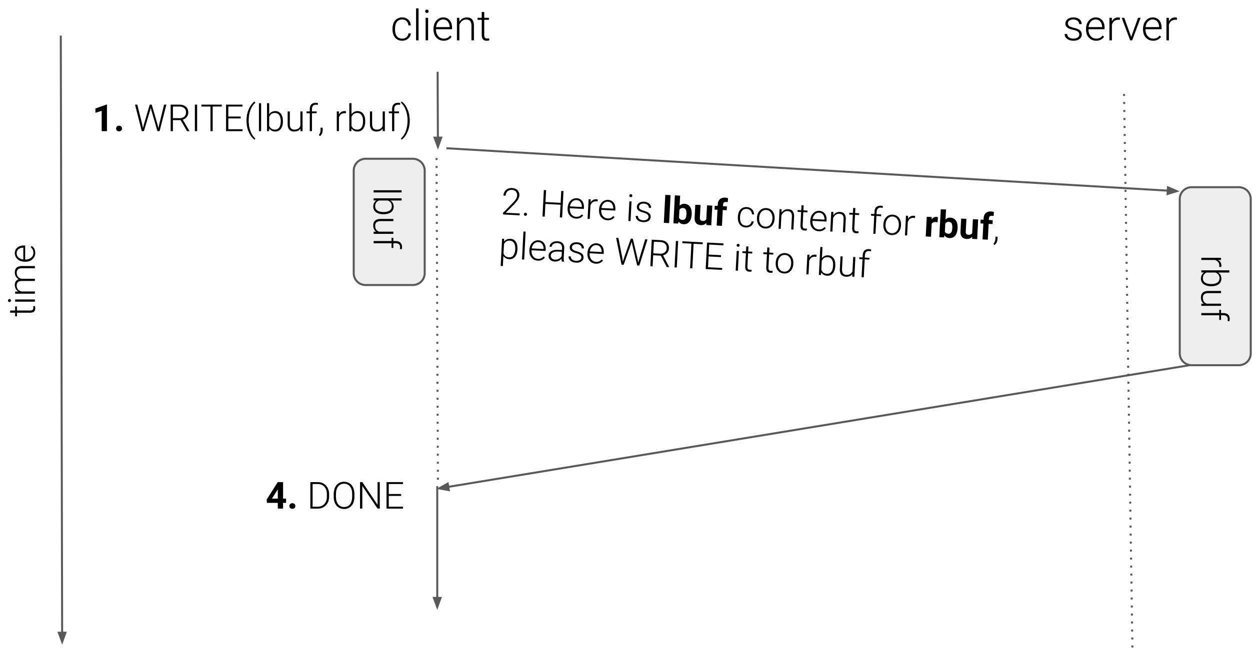
RDMA READ Operation



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

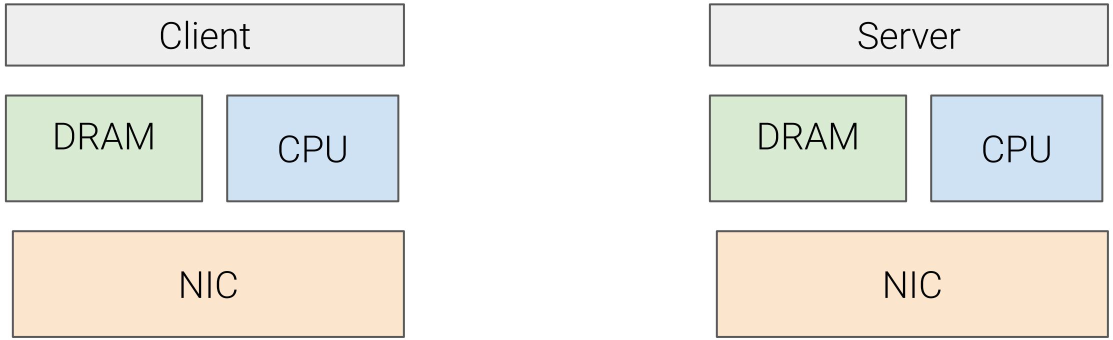
RDMA WRITE Operation



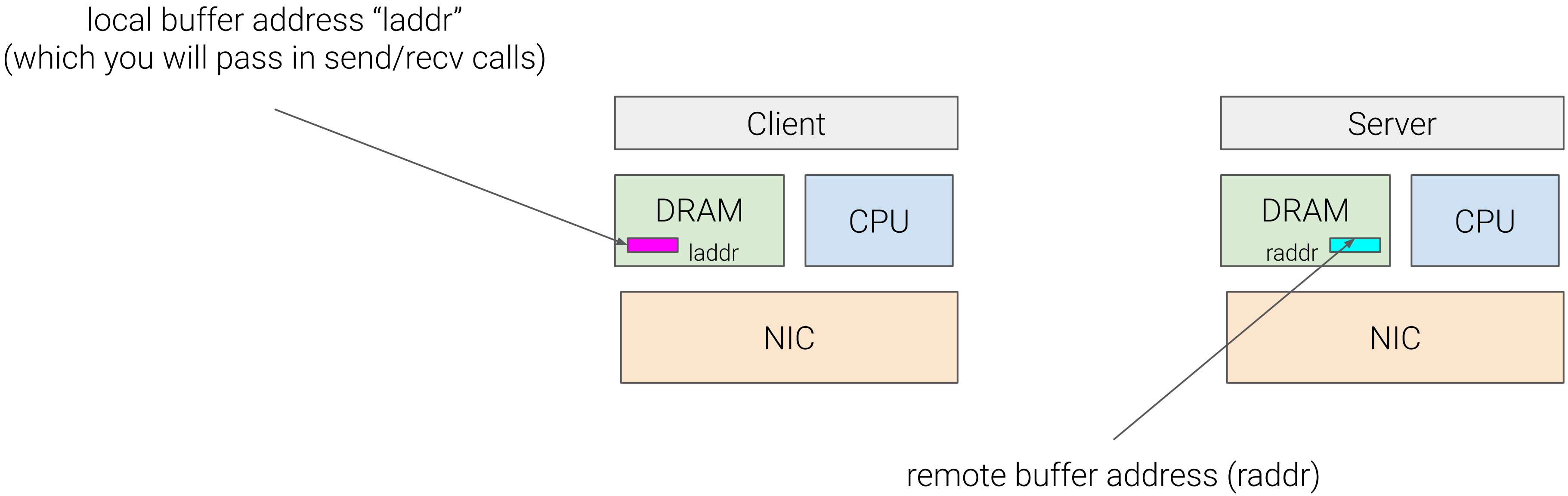
1. Req arrives
2. ~~DMA the request~~
3. ~~IRQ to schedule~~
4. ~~Server processing~~
5. ~~Data copies~~
6. DMA to rbuf
7. Send WRITE response

Example one-sided RDMA WRITE operation

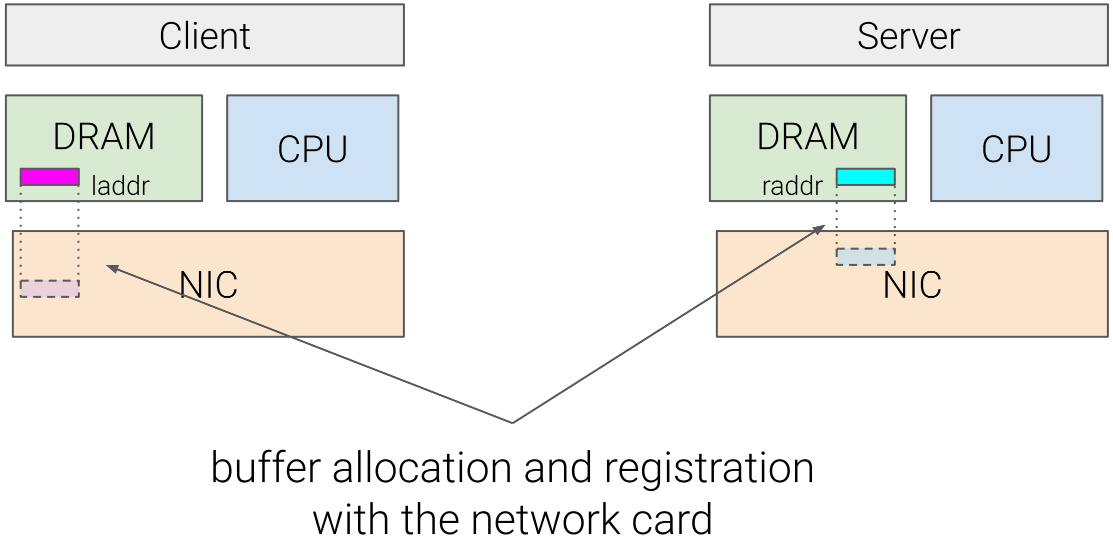
RDMA: Architectural View



RDMA: Architectural View

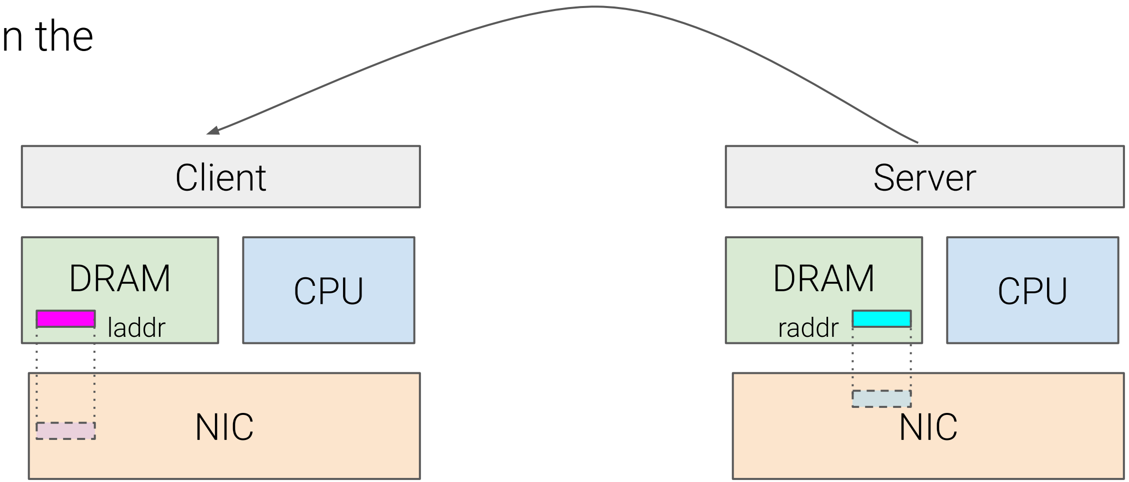


RDMA: Architectural View



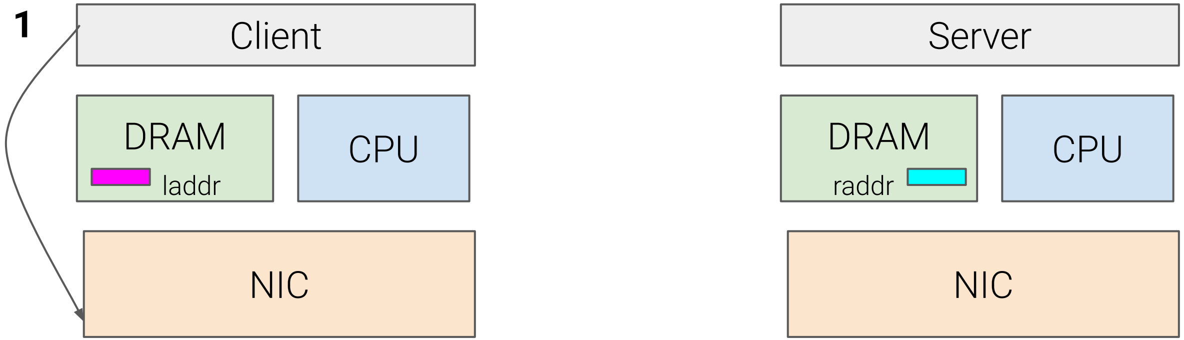
RDMA: Architectural View

Hey! Your content is stored in the buffer at ‘raddr’



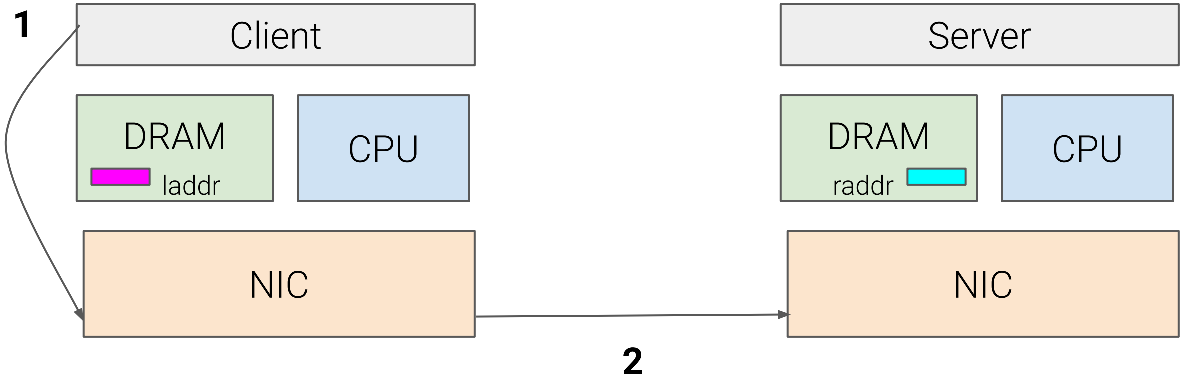
RDMA: Architectural View

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



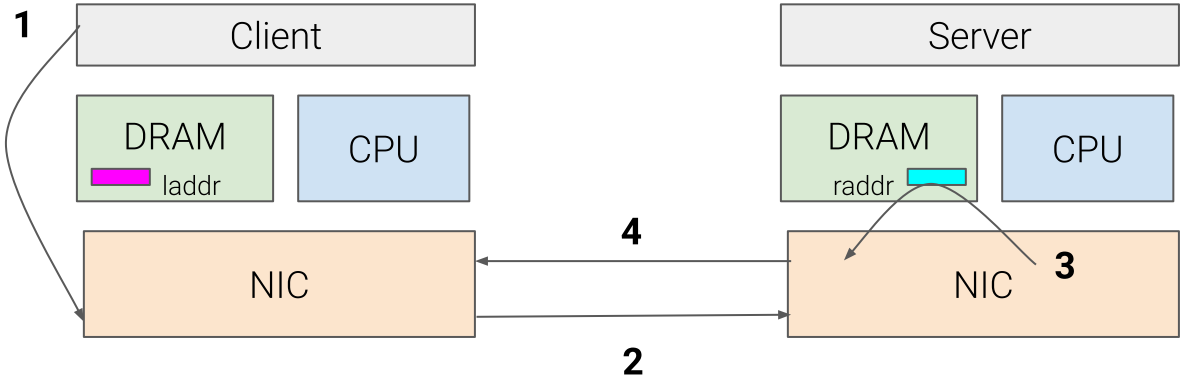
RDMA: Architectural View

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



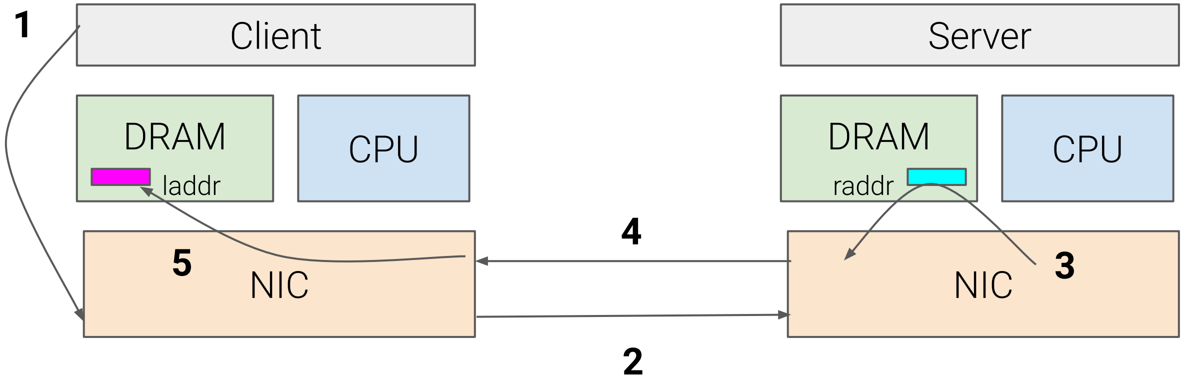
RDMA: Architectural View

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



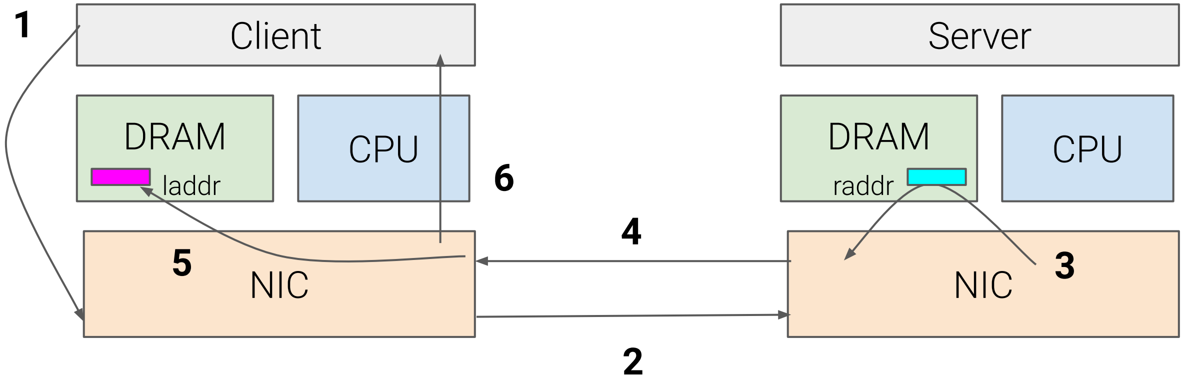
RDMA: Architectural View

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



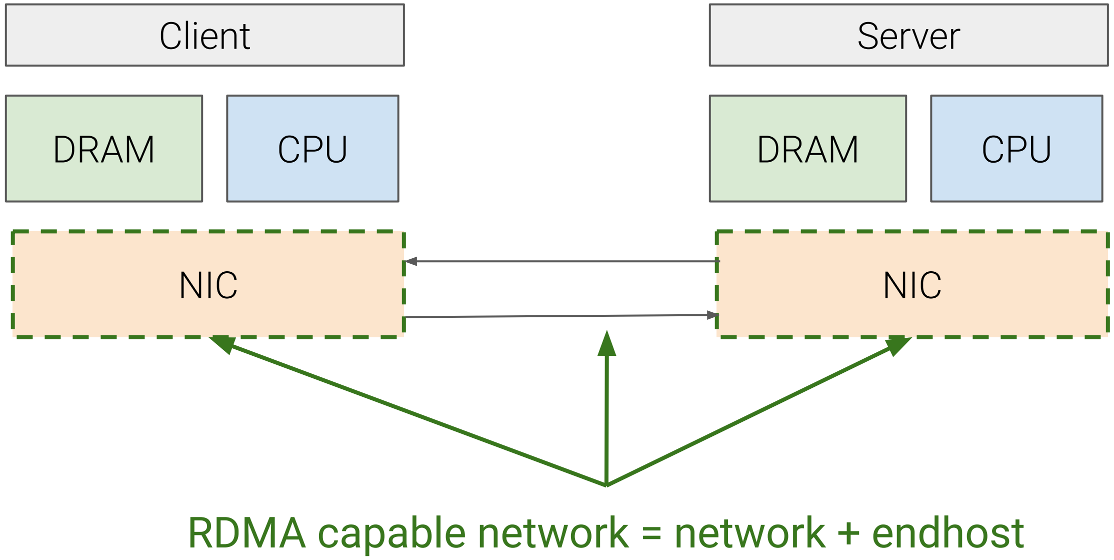
RDMA: Architectural View

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



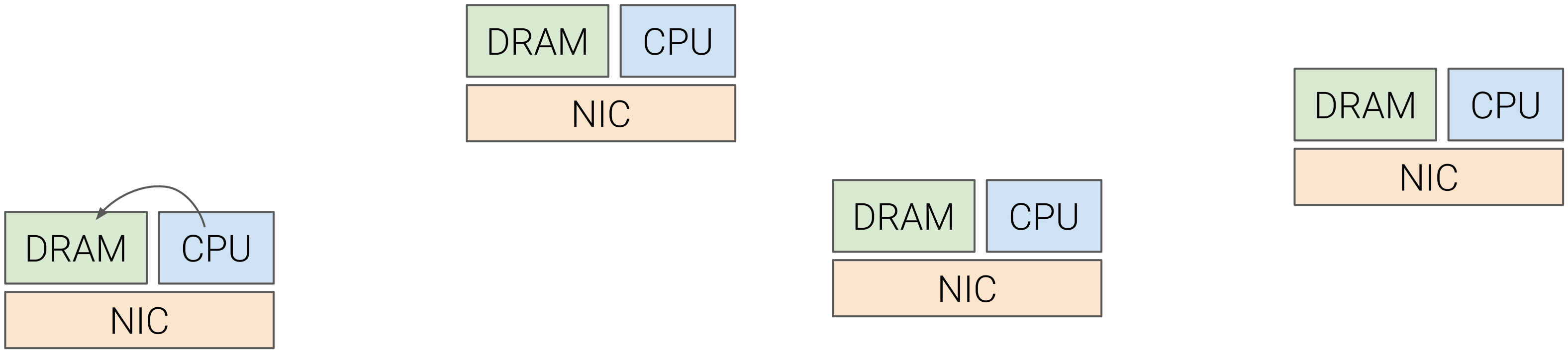
RDMA: Architectural View

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



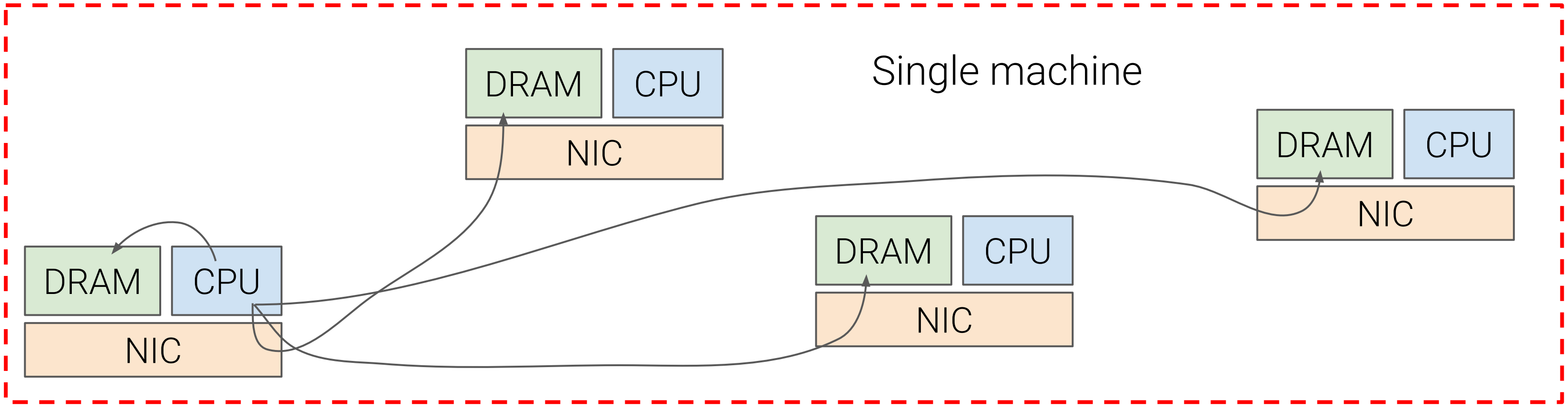
A Brief History

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



A Brief History

- 1980s: a long history of high-performance networking research
  - Building networked multi-processor systems/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



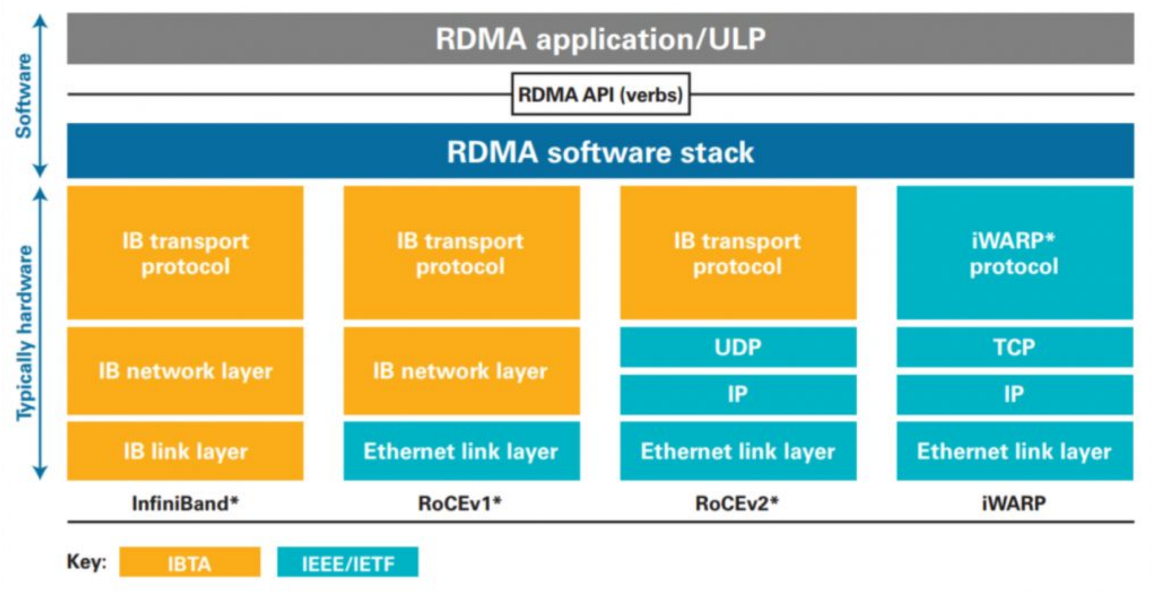
A Brief History

- 1980s: a long history of high-performance networking research
    - Building networked multi-processor systems/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 Top500 list from 2000s)

A Brief History

- 1980s: a long history of high-performance networking research
  - Building networked multi-processor systems/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 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



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

- There is no ONE RDMA API like socket
- There is no ONE RDMA framework - you can write your own from scratch!
  - Each interconnect provider can give its own (MLX)
  - Often wrapped under another high-level API like MPI
  - A (pseudo) standard stack is Open-Fabric Alliance (OFA)
- The RDMA idea is independent of the networking technology and the programming interfaces used
  - Infiniband, iWARP, RoCE - all support RDMA operations on top of different networking layers

Agenda

1. ~~A closer look at the socket networking~~
2. ~~Challenges with the classical networking~~
3. ~~User-space networking~~
4. ~~Remote Direct Memory Access (RDMA)~~
    a. Performance
5. RDMA applications
6. Hands on experiments

Performance

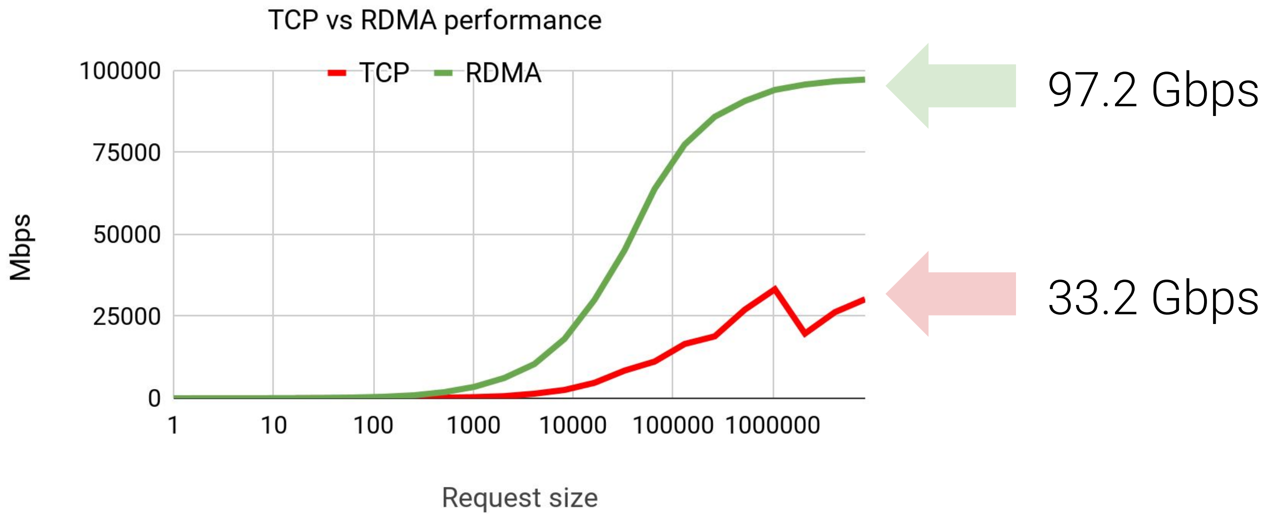
What does RDMA promise to deliver?

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

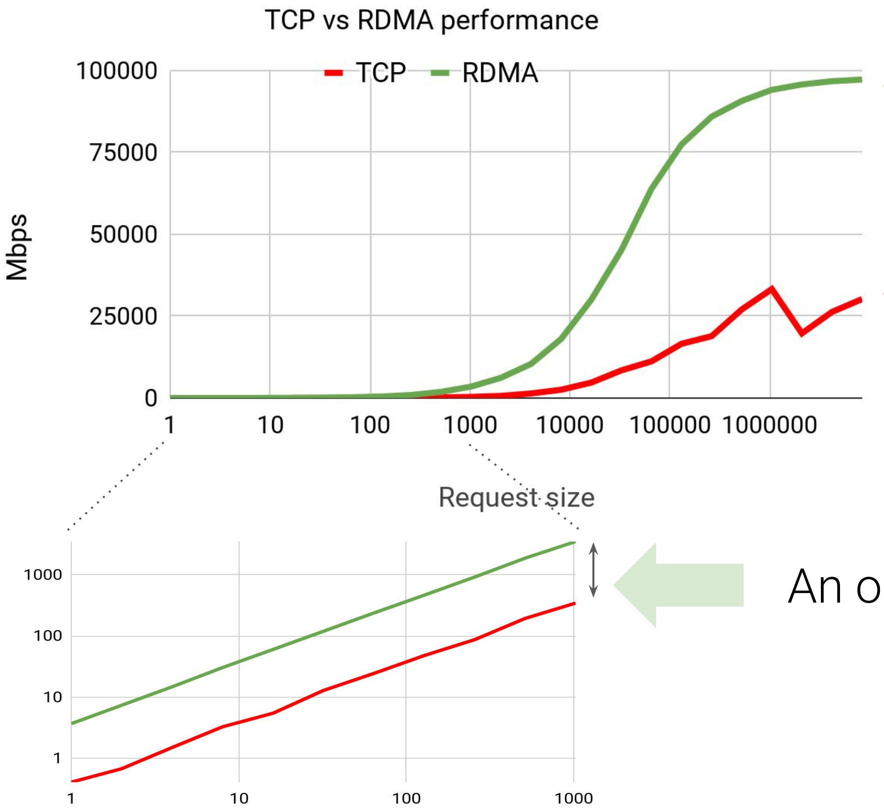
Bandwidth and network operation latencies in a simple request-response setup

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

Performance



Performance

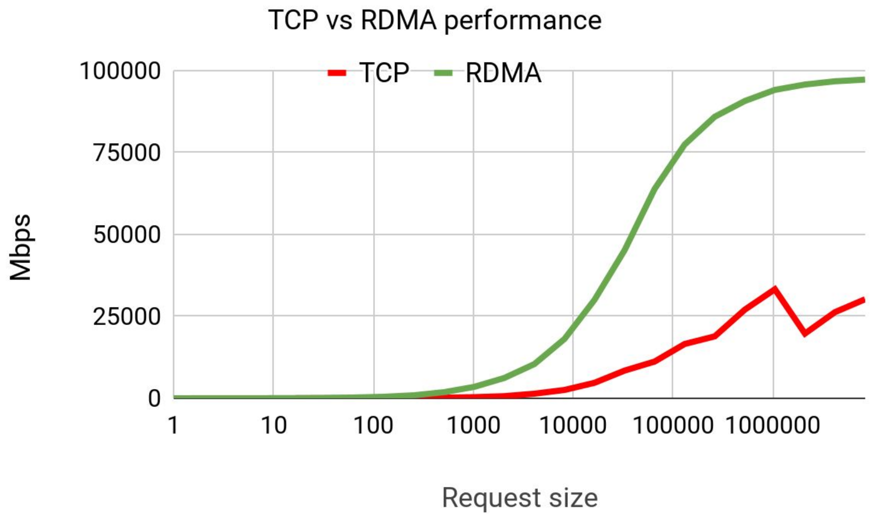


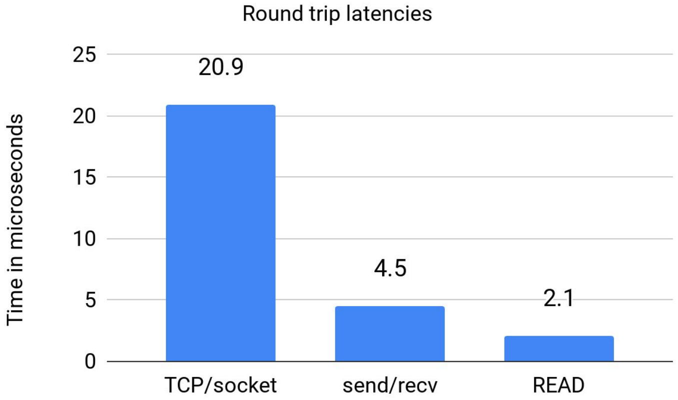
97.2 Gbps

33.2 Gbps

An order of magnitude gap for small requests

Performance





4.6x

10x

Where do the Performance Gains come from?

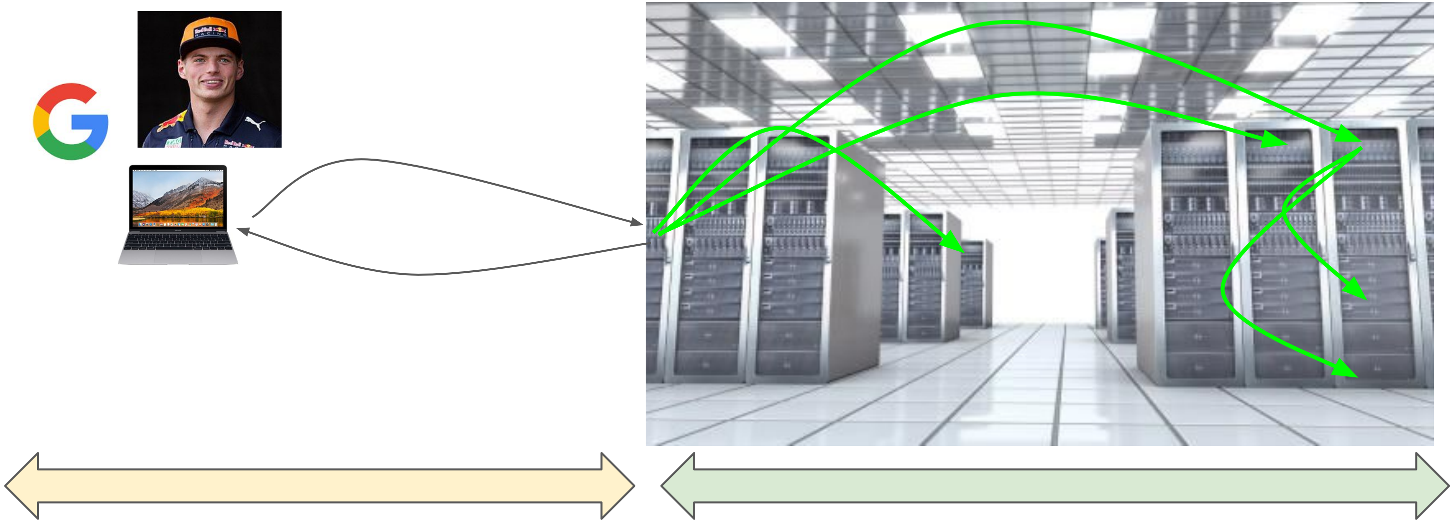
- Closer application network integration
  - When, how, where of network processing
- Better(?), high-performance code
  - Pushing setup at the beginning, resource allocation
- Offloading
  - Hardware acceleration
- Bypassing the operating system
  - Lot of boilerplate code skipped
  - Processing close to the metal
- An active area of research - the RIGHT application/network integration framework

Agenda

1. ~~A closer look at the socket networking~~
2. ~~Challenges with the classical networking~~
3. ~~User-space networking~~
4. ~~Remote Direct Memory Access (RDMA)~~
    a. ~~Performance~~
5. RDMA applications
6. Hands on experiments

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

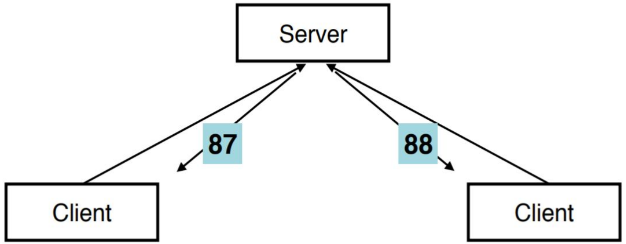
Inside a datacenter
- 100s of Gbps
- 1-10s of usec RTTs

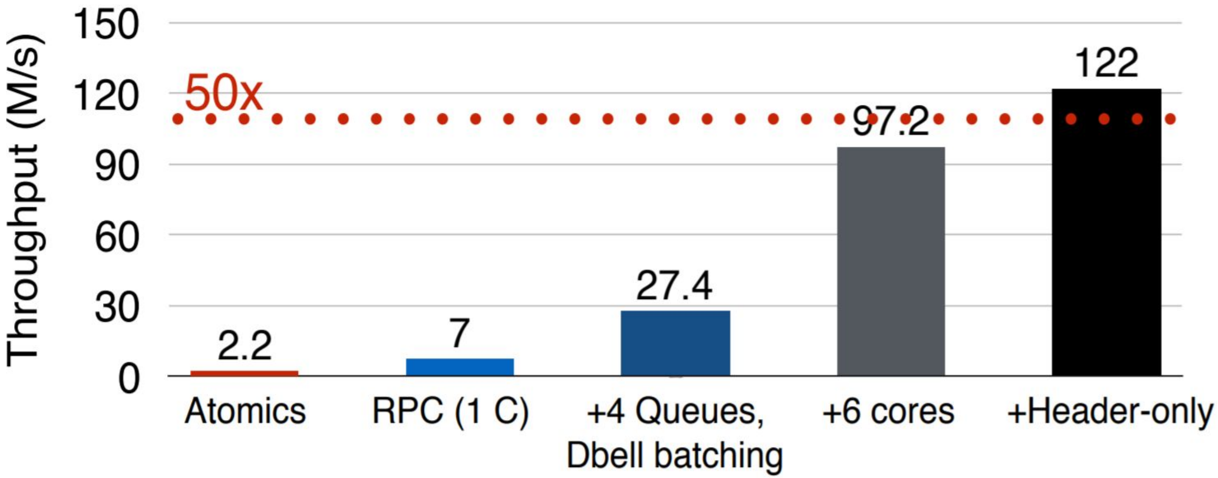
RDMA Design Space

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

Example - Sequencer Throughput



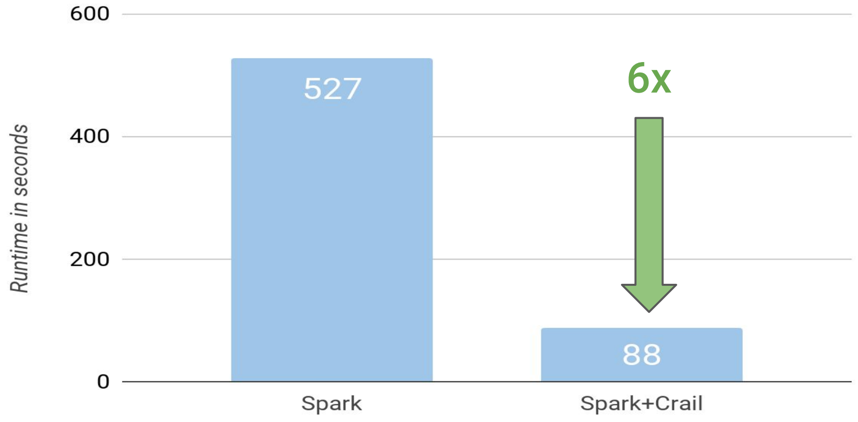


The design space is large, and performance margins are 1-2 orders of magnitude

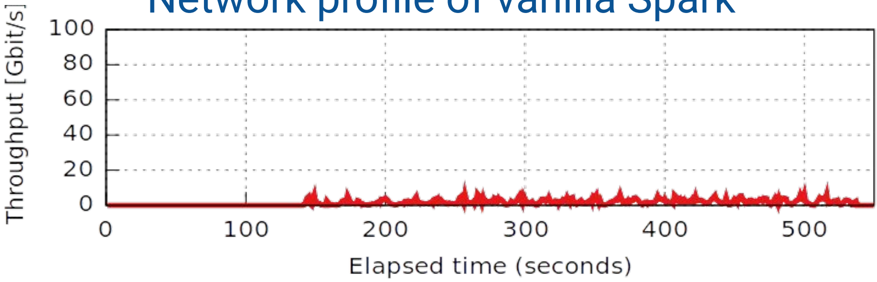
Paper: Design Guidelines for High Performance RDMA Systems, Usenix 2016

Workload-level Acceleration

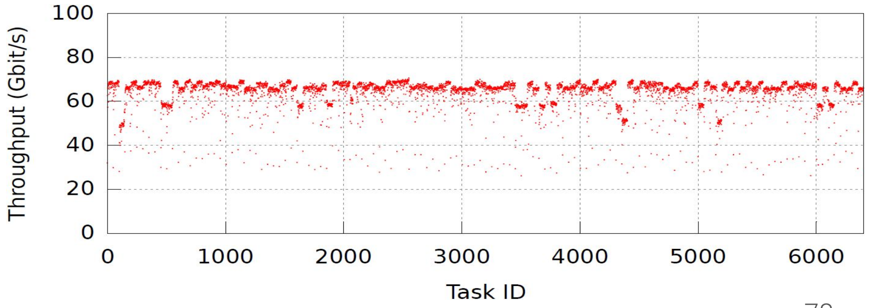
Sorting 12.8 TB of data on 128 machines
- 100 Gbps network
- 4 x NVMe devices (source and sink)
- Apache Spark



Network profile of vanilla Spark



Optimized Network profile of Spark



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

End Summary

- NICs are getting faster, but the CPU is not
    - CPU/software governs the performance, not the networking devices
    - Next-generation of programmable devices are coming (NICs, GPUs, FPGAs, storage, etc.)
- New ways of doing network I/O are being explored
    - The idea of User-space networking, kernel bypassing, and separating data from control paths
    - New interfaces (not socket) and ways of doing networking - e.g., RDMA operations
- Applications of RDMA networking in distributed systems/data centers
    - Large design space, and lots of new applications
    - Apache Crail project - accelerating data sharing in distributed systems (www.crail.io)
    - Challenges with deployments at scale

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)


Talk is cheap. Show me the code.
— Linus Torvalds

Things to Know about RDMA Programming

- There are more than one ways to get started with working with RDMA
  - We are using the OFA OFED environment
- There is a bit of a setup involved before getting started
  - Setting up the kernel modules and user-space libraries
- For historical reasons functions are (often) prefixed by `ib_ or ibv_`
- I recommend (for now) to stick with the RDMA CM interfaces (`rdmacm`)
  - But there more, feel free to peak around or ask me

Setting up the VM

Follow instructions at : https://github.com/animeshtrivedi/blog/blob/master/post/2019-06-26-siw.md

So How Do I Program RDMA?

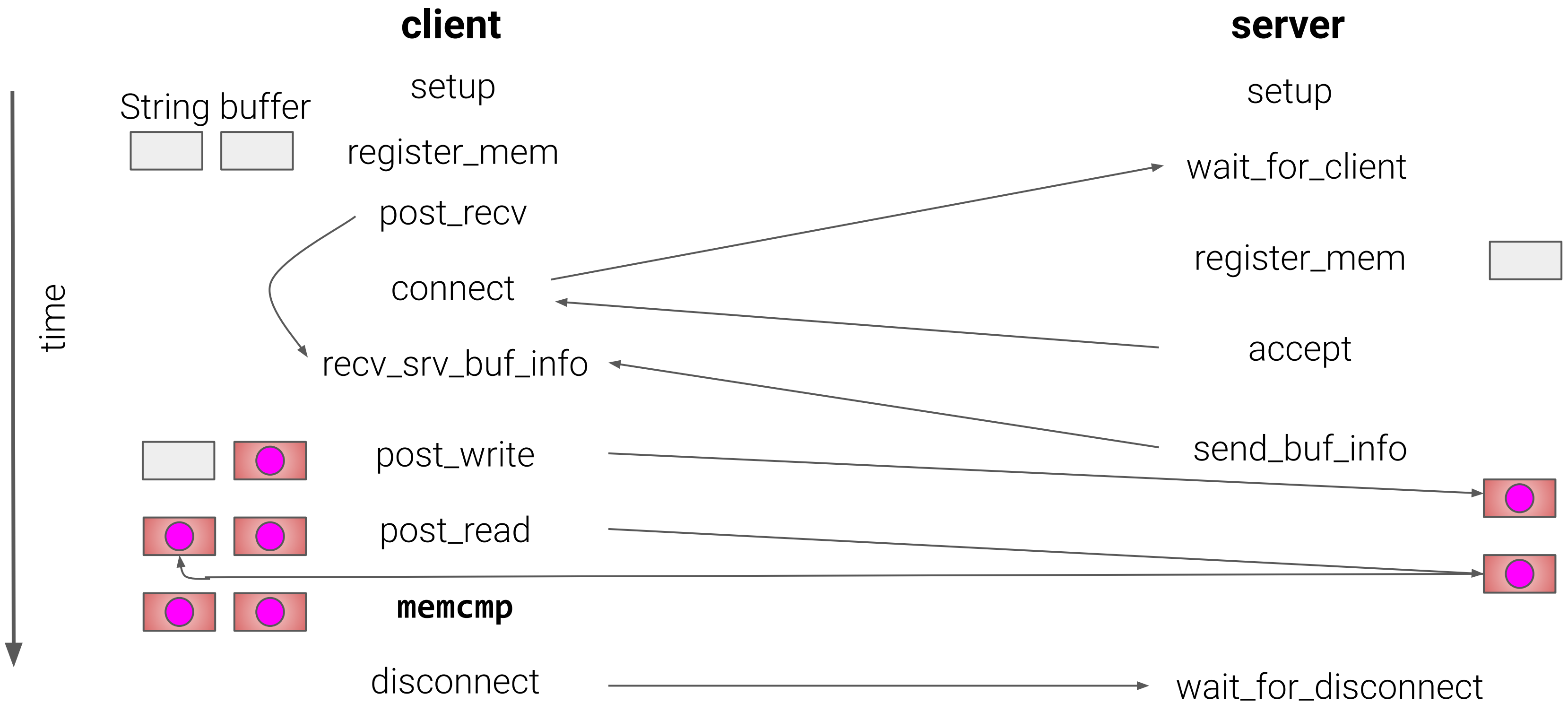
You will need some resources (their usage will become clearer later)

- Connection identifier : a connection identifier
- A transmission and reception queue, or a queue pair (QP)
- Work queue element (WQEs)
  - Scatter gather elements (SGEs) - contains buffer description
- Completion queue (CQ) : completion completion elements
- Completion channel : work completion notifications
- Memory regions : registered application buffers
- Protection domain : a security container
- Event channel : network event notification
- ...

Let's have a look at the code now

https://github.com/animeshtrivedi/rdma-example.git

What the code is doing?



Your Goal

Is to replicate the same "`memcmp`" steps at the server side

Can sockets be used over RDMA to hide its complexity?