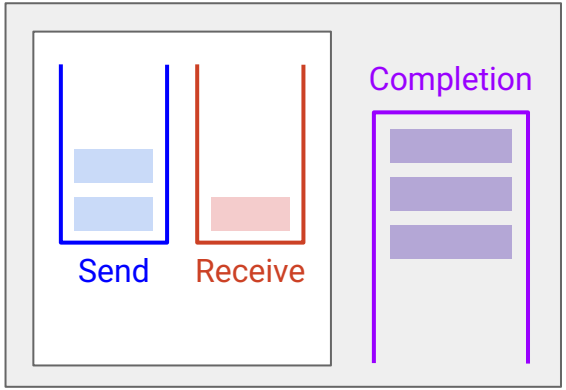


Lecture 22 (Datacenters 4)

Host Networking, RDMA

CS 168, Fall 2026 @ UC Berkeley

Slides credit: Sylvia Ratnasamy, Rob Shakir, Peyrin Kao, Nandita Dukkipati

What is Host Networking?

Lecture 20, CS 168, Spring 2026

Optimizations at the Host

- **What is Host Networking?**
- Shared Memory in User Space
- Offloading

Brief History of Offloading

- Epoch 0: No Offloading
- Epoch 1: Stateless Operations
- Epoch 2: Stateful Operations
- Epoch 3: Protocols (RDMA)
- RDMA Example

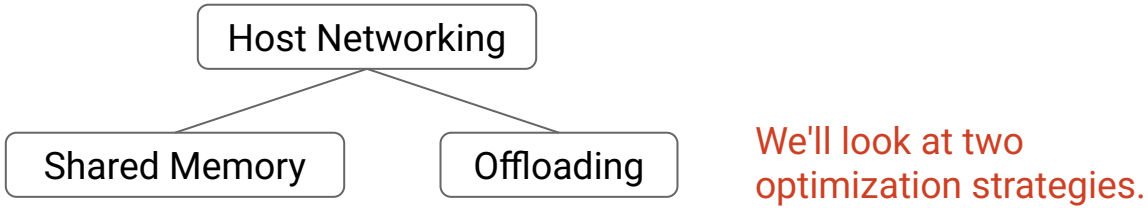
The New Bottleneck: Hosts

Historically, the network has been the bottleneck. *(Example: Congestion control.)*

In modern datacenters, hosts are the new bottleneck.

- Extremely high performance demands.
- Problem 1: CPUs can't keep up.
  - CPU working on network protocols → less resources for the application.
- Problem 2: Protocols can't keep up.
  - Traditional networking stack (TCP/IP) are insufficient.

Host networking: Optimizations **at the host** to meet modern performance demands.



Optimization: Shared Memory in User Space

Lecture 20, CS 168, Spring 2026

Optimizations at the Host

- What is Host Networking?
- **Shared Memory in User Space**
- Offloading

Brief History of Offloading

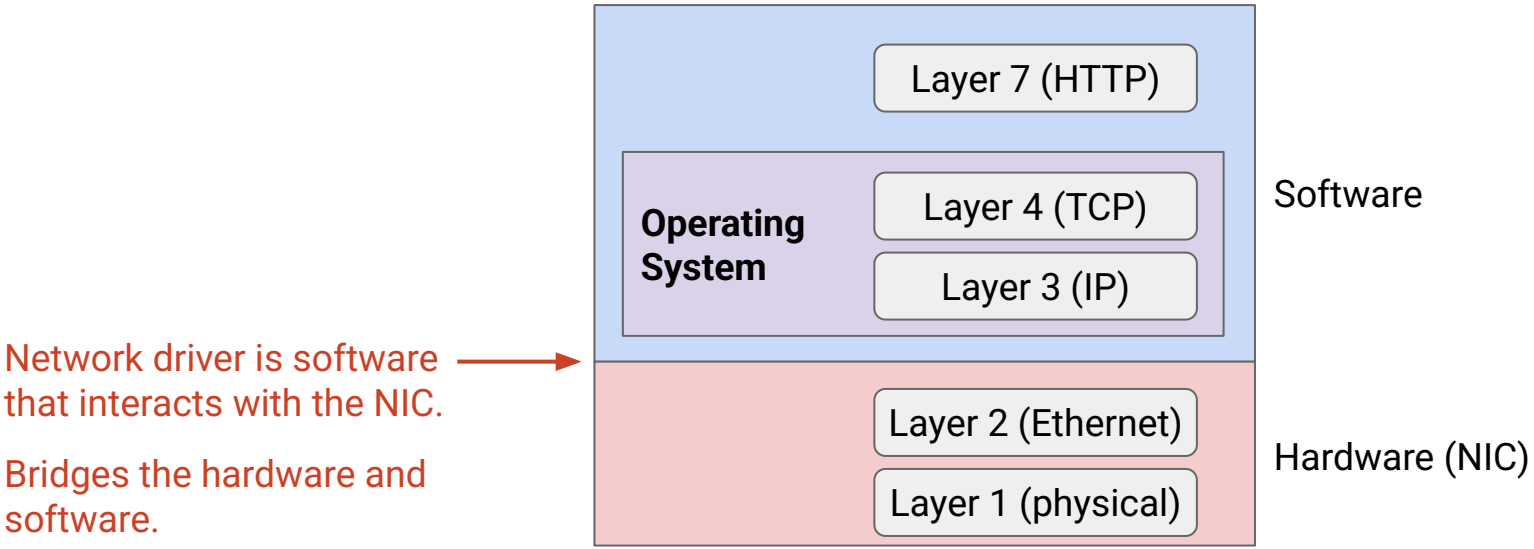
- Epoch 0: No Offloading
- Epoch 1: Stateless Operations
- Epoch 2: Stateful Operations
- Epoch 3: Protocols (RDMA)
- RDMA Example

Recall: Layers in the End Host

Layers 1 and 2 are implemented in hardware, on the network interface card (NIC).

Layers 3 and 4 are implemented in software, in the operating system.

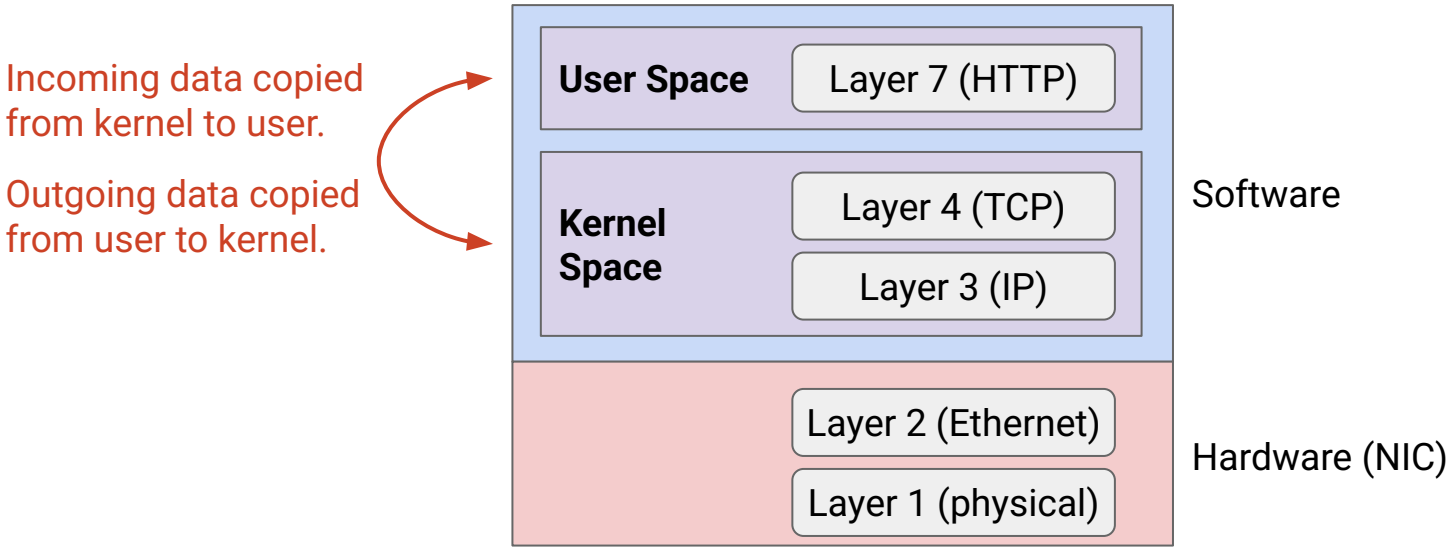
Layer 7 is the applications running in software.



Problem: Kernel Space and User Space

The OS runs in **kernel space**, and applications run in **user space**.

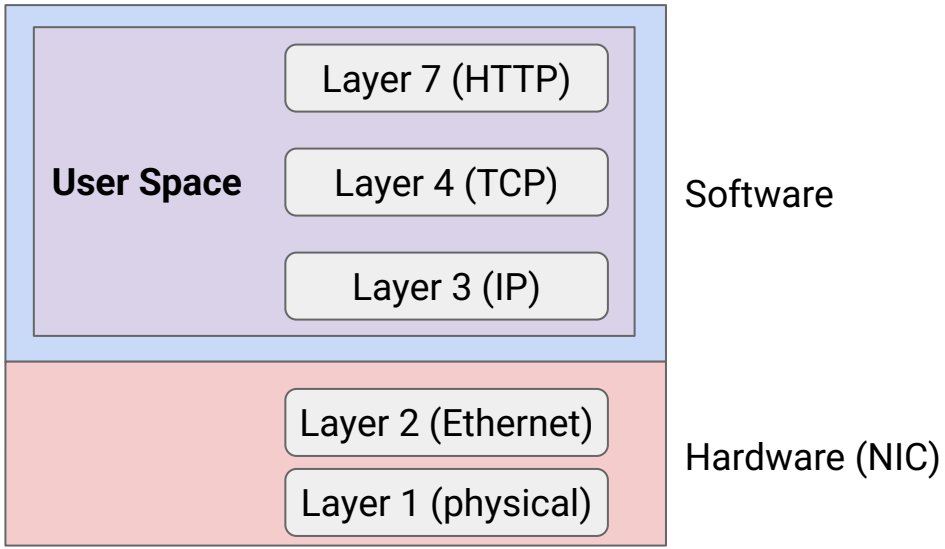
- The two address spaces are isolated from each other. *(Virtual memory from CS 61C.)*
- Problem: Data is constantly being copied between user/kernel space.
- Problem: Programming in the kernel is hard.



Solution: Networking Stack in User Space

Solution: Packet processing happens in user space.

- Application and network protocols access shared memory.
- No need to copy data back-and-forth!
- Easier to program (e.g. new protocols) in the user space!



Optimization: Offloading

Lecture 20, CS 168, Spring 2026

Optimizations at the Host

- What is Host Networking?
- Shared Memory in User Space
- **Offloading**

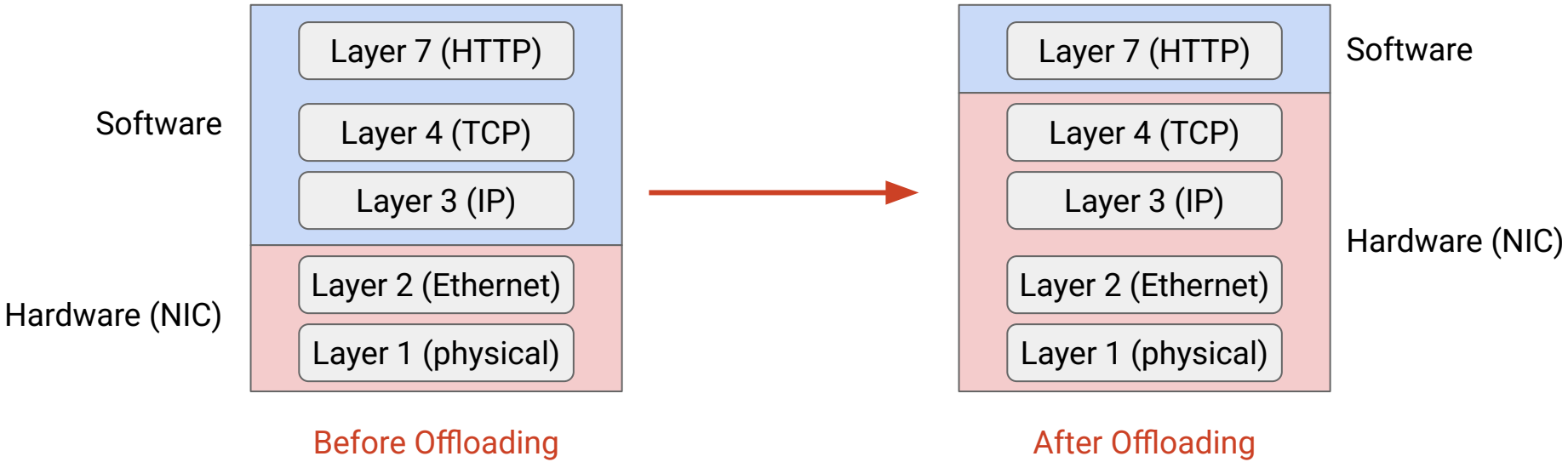
Brief History of Offloading

- Epoch 0: No Offloading
- Epoch 1: Stateless Operations
- Epoch 2: Stateful Operations
- Epoch 3: Protocols (RDMA)
- RDMA Example

Optimization: Offloading

Problem: CPUs can't keep up with modern performance demands.

Solution: **Offloading**. Process packets in hardware instead of software.



Benefits of Offloading

Frees up CPU cycles for applications.

- Network processing and applications no longer contend for CPU.

Efficiency.

- Specialized processing in hardware can be more efficient.
- Power savings.

Predictability.

- CPU has unpredictable scheduling delays.
- Hardware gives us more consistent latencies.

Offloading, Epoch 0: No Offloading

Lecture 20, CS 168, Spring 2026

Optimizations at the Host

- What is Host Networking?
- Shared Memory in User Space
- Offloading

Brief History of Offloading

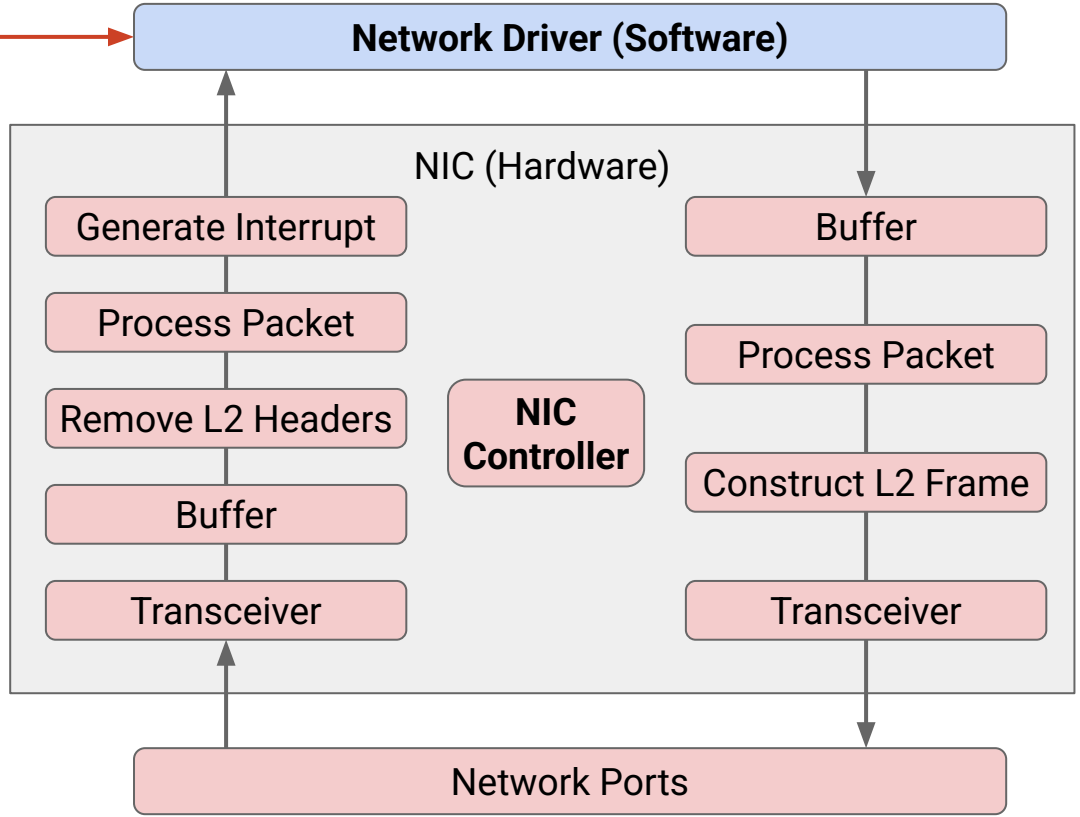
- **Epoch 0: No Offloading**
- Epoch 1: Stateless Operations
- Epoch 2: Stateful Operations
- Epoch 3: Protocols (RDMA)
- RDMA Example

NIC Before Offloading

Before any offloading, the NIC is a "doormat" for packets, with minimal processing.

Network driver is software that interacts with the NIC.

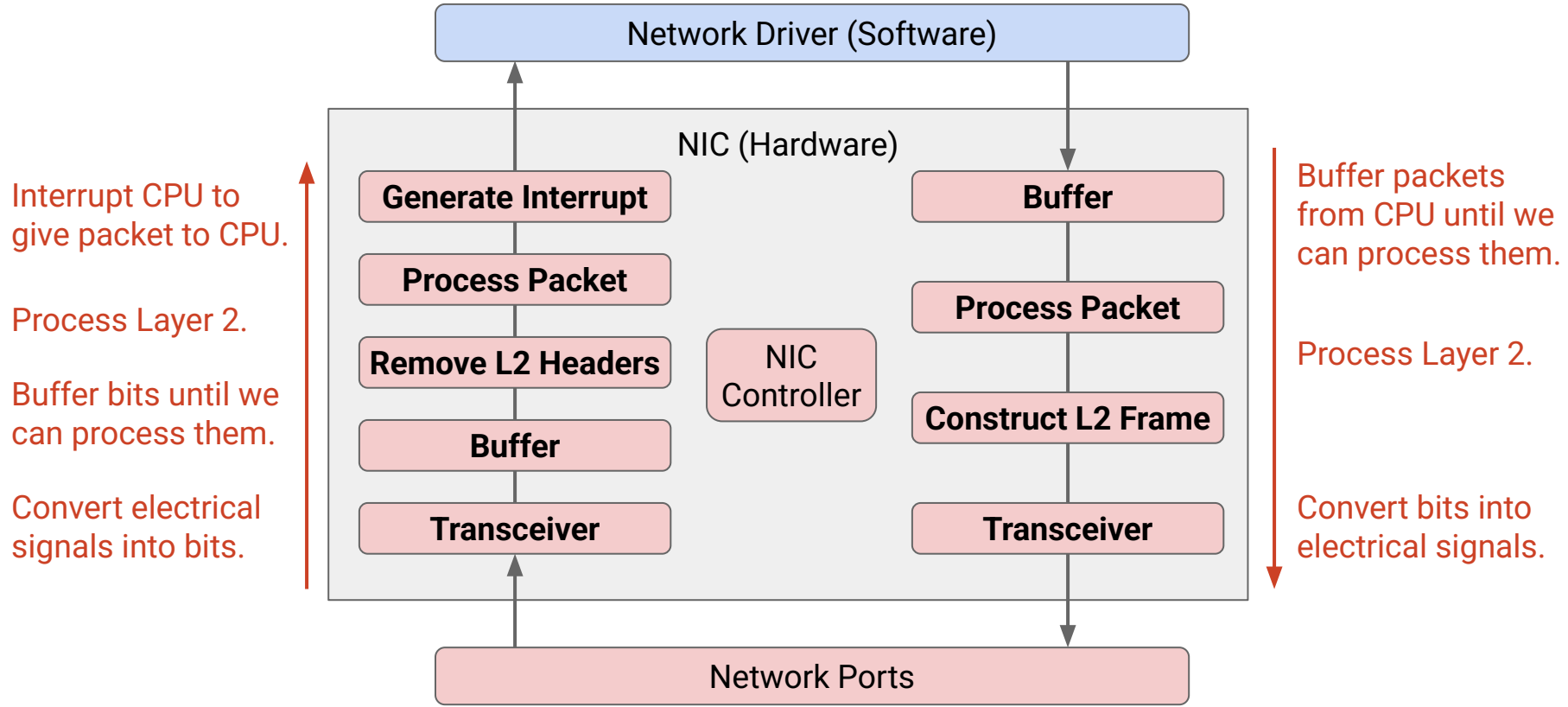
Bridges the hardware and software.



NIC controller manages operation on the card.

NIC Before Offloading

The NIC passes incoming packets up to CPU, and sends out packets from CPU.



Offloading, Epoch 1: Stateless Operations

Lecture 20, CS 168, Spring 2026

Optimizations at the Host

- What is Host Networking?
- Shared Memory in User Space
- Offloading

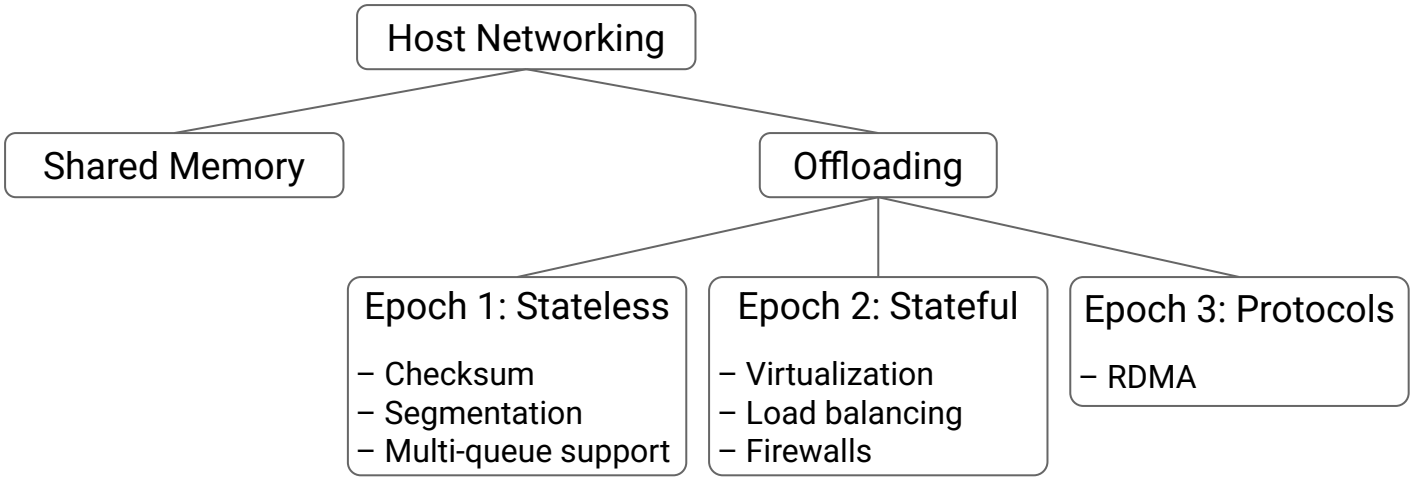
Brief History of Offloading

- Epoch 0: No Offloading
- **Epoch 1: Stateless Operations**
- Epoch 2: Stateful Operations
- Epoch 3: Protocols (RDMA)
- RDMA Example

Brief History of Offloading

Over time, increasingly complicated operations were offloaded to the NIC.

- Epoch 1: Simple stateless operations.
- Epoch 2: Stateful operations.
- Epoch 3: Protocols like RDMA.



Epoch 1 Offloads (1/3): Checksum

The NIC can *set* Layer 4 checksum for *outgoing* packets.

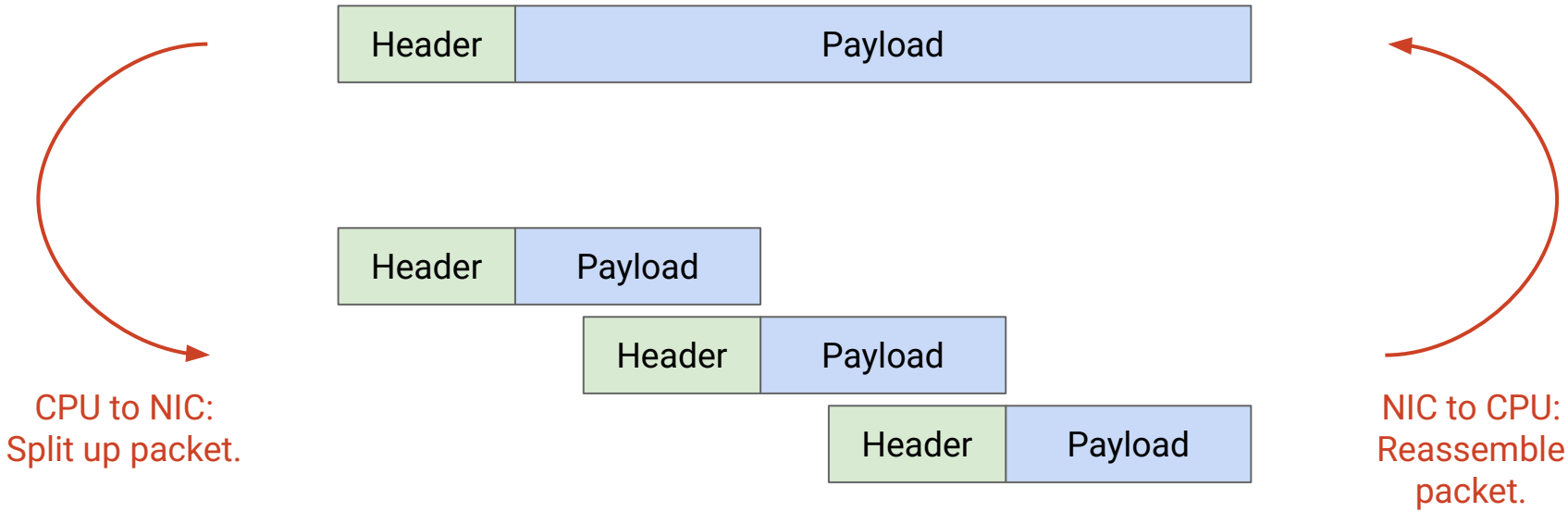
The NIC can *verify* Layer 4 checksum for *incoming* packets.

CPU tells the NIC where to read/write the checksum.

Epoch 1 Offloads (2/3): Segmentation

CPU can offload segmentation work (splitting and reassembling packets) to the NIC.

- CPU handles a few large packets.
- NIC handles lots of small packets.



Epoch 1 Offloads (2/3): Segmentation

Segmentation comes with trade-offs.

With segmentation:

- Less work for CPU to do.
- CPU hands large packets to NIC. Traffic is more bursty.

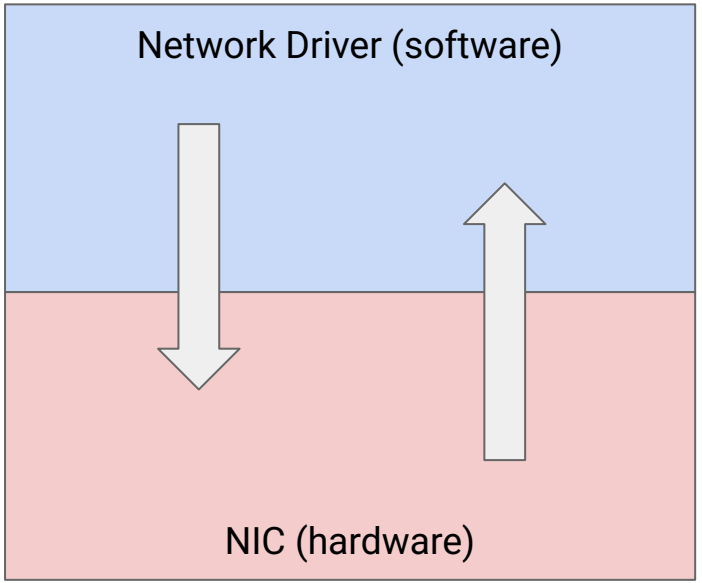
Without segmentation:

- More work for CPU to do.
- CPU hands small packets to NIC. Traffic is smoother.

Epoch 1 Offloads (3/3): Multi-Queue Support

Without offloading: NIC has 1 transmit queue and 1 receive queue.

Network driver is responsible for isolation and load balancing in a multi-processor system (multiple CPUs).



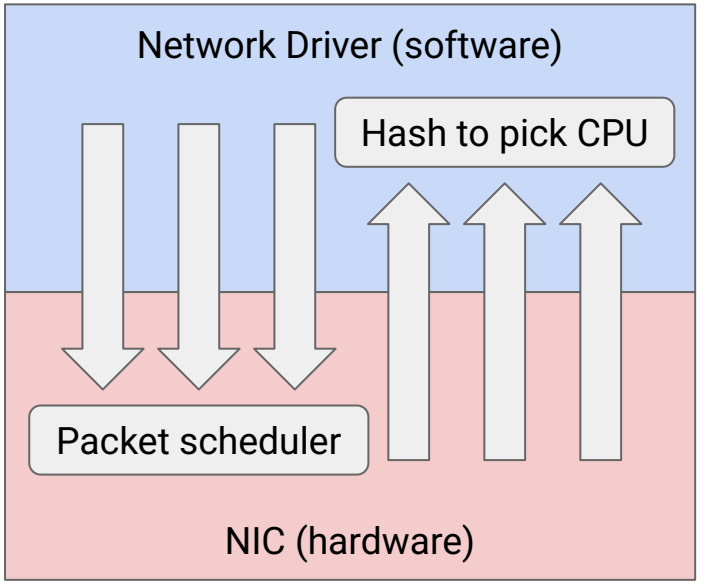
Epoch 1 Offloads (3/3): Multi-Queue Support

Idea: Multiple transmit queues.

- Packet scheduler decides which queue to send from next.

Idea: Multiple receive queues.

- Hash packets (like ECMP, equal-cost multi-path routing) to decide which CPU handles the packet.



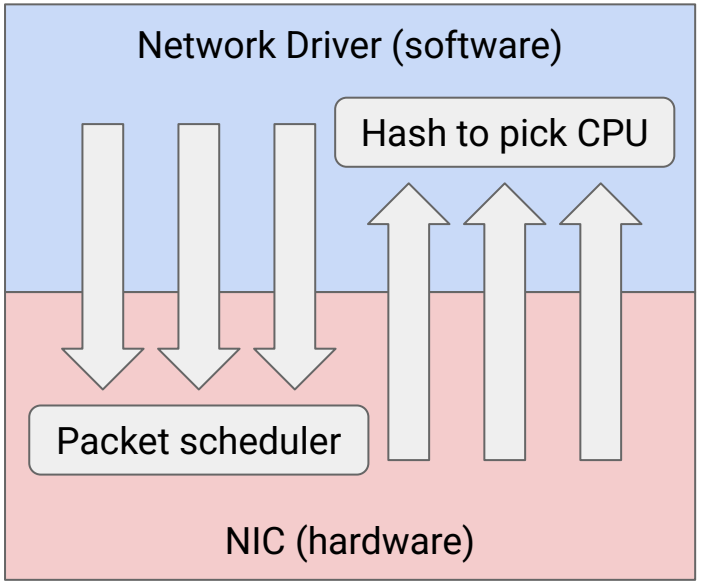
Epoch 1 Offloads (3/3): Multi-Queue Support

Benefits:

- Ensures isolation: Specific flows can use a dedicated queue.
- Ensure load-balancing: Heavy traffic in one queue doesn't affect others.
- Can prioritize specific flows.

Challenges:

- Must ensure packets in the same flow use the same queue, to avoid out-of-order packets (which slows down TCP).



Offloading, Epoch 2: Stateful Operations

Lecture 20, CS 168, Spring 2026

Optimizations at the Host

- What is Host Networking?
- Shared Memory in User Space
- Offloading

Brief History of Offloading

- Epoch 0: No Offloading
- Epoch 1: Stateless Operations
- **Epoch 2: Stateful Operations**
- Epoch 3: Protocols (RDMA)
- RDMA Example

Epoch 2: Stateful Offloads with Match-Action Tables

Stateful operations to offload:

- **Virtualization**: Virtual switch forwarding packets to/from VMs.
- **Bandwidth metering**: If user sends too many packets, drop the excess.
- **Firewalls**: Drop packets from malicious users.

Behavior specified by match-action tables (like OpenFlow).

Match	**Action**
Source IP = 10.1.8.9	Add encapsulation header, forward.
5-tuple matches (10.1.2.3, 50000, TCP, 24.1.3.0, 80)	Allow 100 packets per minute. Drop excess.
Source IP = 76.124.1.2	Malicious source. Drop.

Offloading, Epoch 3: Protocols (RDMA)

Lecture 20, CS 168, Spring 2026

Optimizations at the Host

- What is Host Networking?
- Shared Memory in User Space
- Offloading

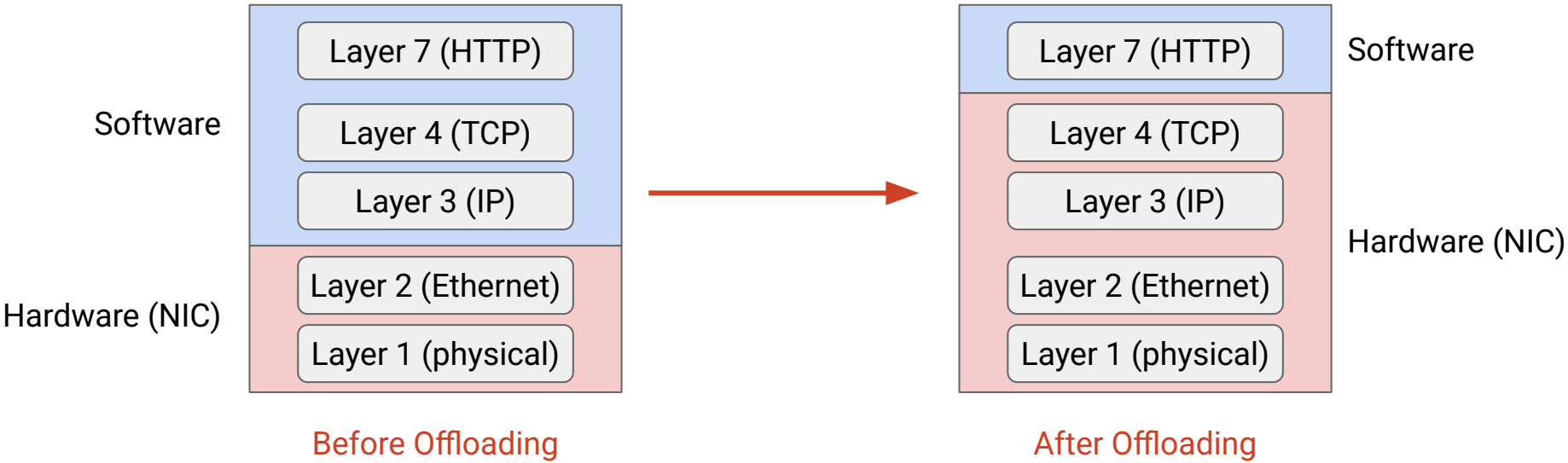
Brief History of Offloading

- Epoch 0: No Offloading
- Epoch 1: Stateless Operations
- Epoch 2: Stateful Operations
- **Epoch 3: Protocols (RDMA)**
- RDMA Example

Epoch 3: Protocols

Goal: Offload entire protocols, like TCP, into hardware.

- Active area of research!
- Instead of implementing TCP in hardware, design custom protocols for hardware.



RDMA: Remote Direct Memory Access

Transferring a file with traditional TCP/IP networking stack:

- Sender CPU reads file from memory.
- Sender CPU sends out packets.
- Recipient CPU processes packets.
- Recipient CPU writes file to memory.



Without RDMA: CPU involved in data transfer.

RDMA: Remote Direct Memory Access

RDMA (Remote Direct Memory Access):

- Abstraction: A can directly access memory in B's computer.
- Goal: Minimal CPU involvement in the data transfer.

Transferring a file with RDMA:

- CPUs set up the transfer and get out of the way.
- Sender NIC reads memory and sends out data.
- Recipient NIC receives data and writes to memory.

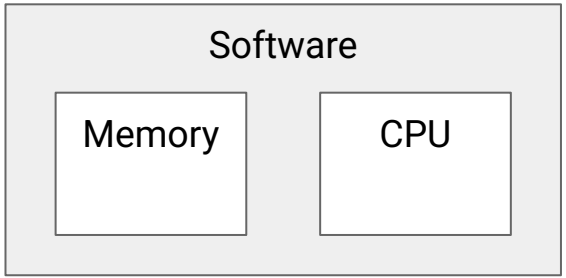


RDMA: CPU is minimally involved in transfer!

Abstraction: Queue Pairs

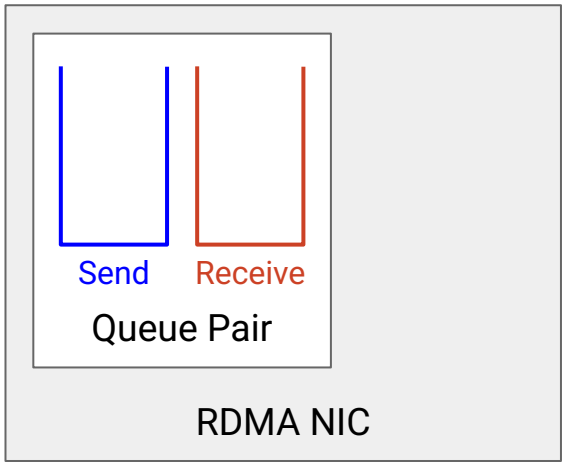
TCP used bytestreams, but RDMA uses the **queue pair** abstraction between the application and NIC.

- **Send queue** has all pending jobs, where I'm sending data to someone else.
- **Receive queue** has all pending jobs, where someone else is sending data to me.



A single NIC can have multiple queue pairs.

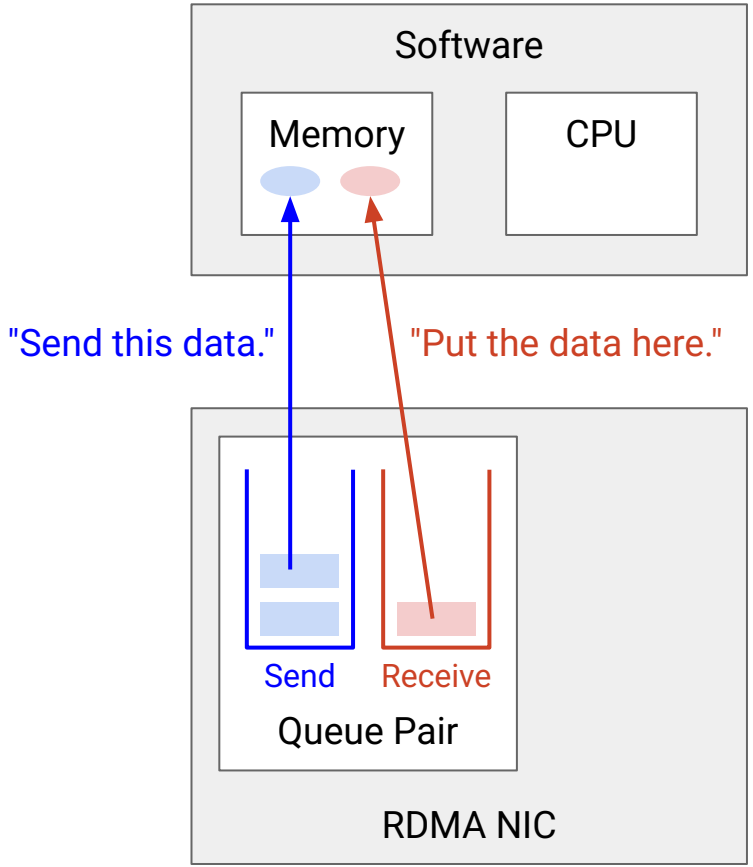
- Each queue pair has a different service guarantee. (Reliable? Ordered?)
- Reliable, in-order queue pair is most similar to TCP.



Abstraction: Work Queue Elements

CPU schedules transfers by adding **Work Queue Elements (WQEs)** to the queues.

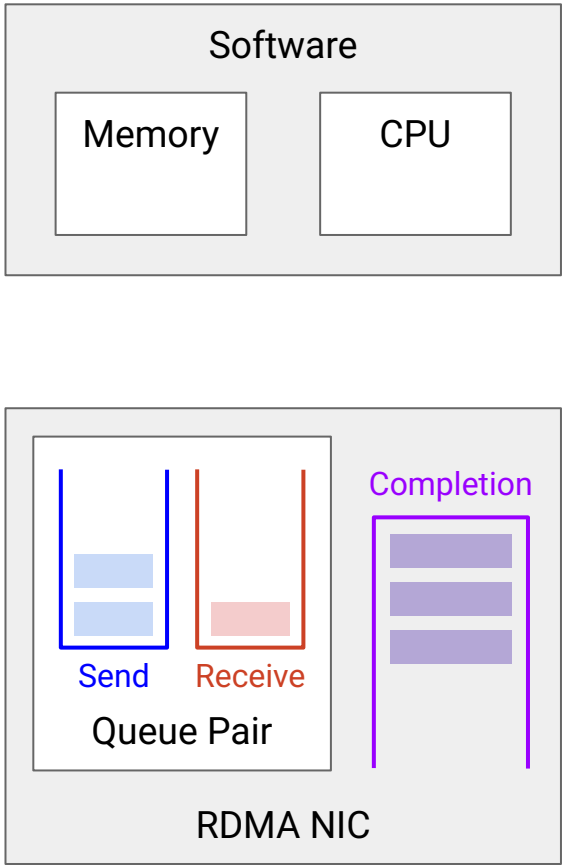
- Struct with instructions about the transfer.
  - Contains pointers to where in memory to read/write data.
- Gives the NIC an application-level view of the transfer, e.g. where the file starts/ends.
  - Contrast with TCP, which just sees a bytestream.



Abstraction: Asynchronous Scheduling, Completion Queue

RDMA is asynchronous.

- CPU schedules jobs in the queue pair.
- NIC processes jobs in the queue.
- When job completes, NIC adds a **Completion Queue Element (CQE)** to the Completion Queue, describing what happened (e.g. success or error).
- Application reads CQE to see what happened with the job.



RDMA Example

Lecture 20, CS 168, Spring 2026

Optimizations at the Host

- What is Host Networking?
- Shared Memory in User Space
- Offloading

Brief History of Offloading

- Epoch 0: No Offloading
- Epoch 1: Stateless Operations
- Epoch 2: Stateful Operations
- Epoch 3: Protocols (RDMA)
- **RDMA Example**

RDMA Example

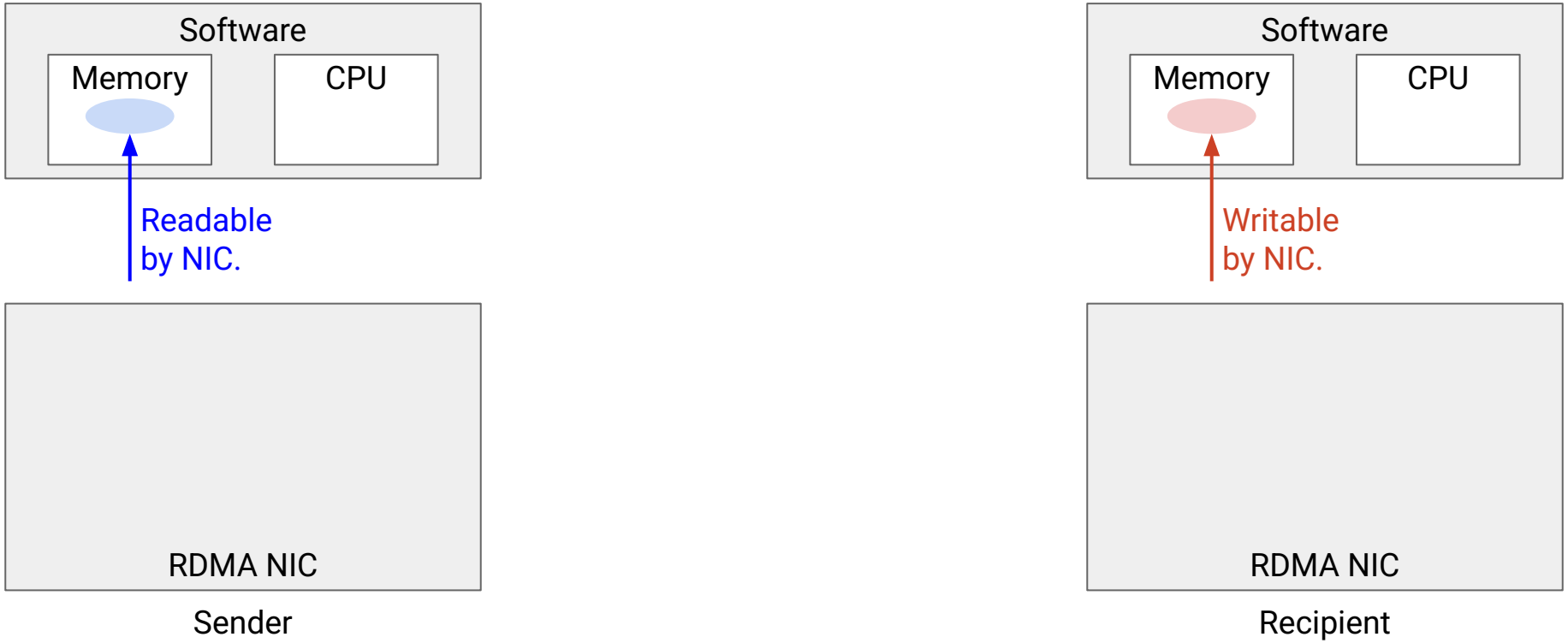
There are many different RDMA operations:

- RDMA Write.
- RDMA Read.
- RDMA Atomic. Perform an operation in remote host's memory.
- RDMA Write with Immediate. Send additional value along with writing data.

Let's look at an RDMA Send/Receive.

Steps of RDMA (1/6)

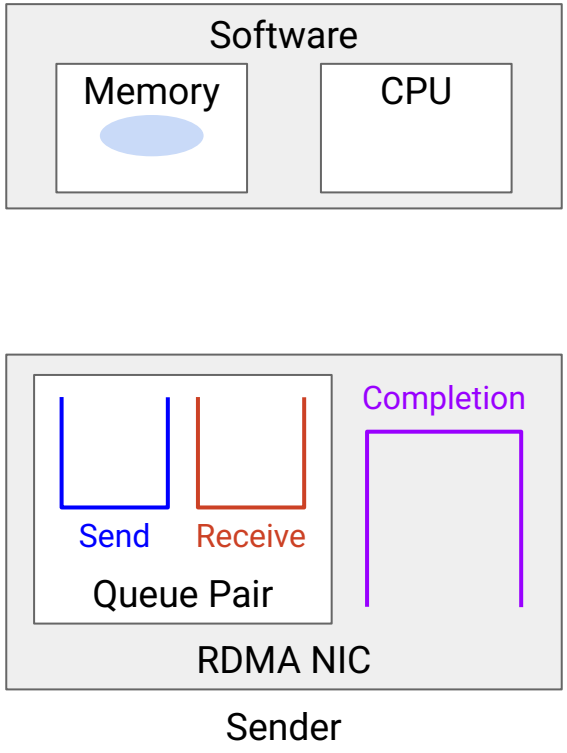
1. Each server designates some memory to be accessible by NIC for RDMA transfers.

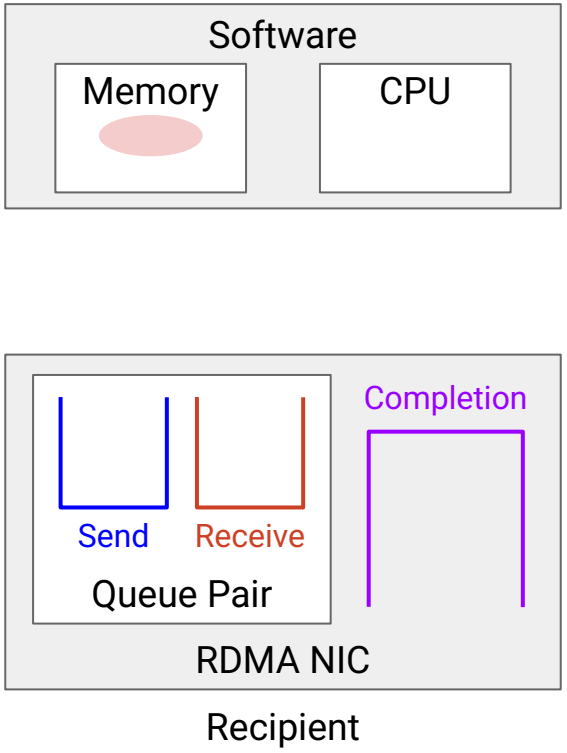


Steps of RDMA (2/6)

2. Each server sets up queues.

This can be done out-of-band, e.g. use TCP to coordinate between servers.

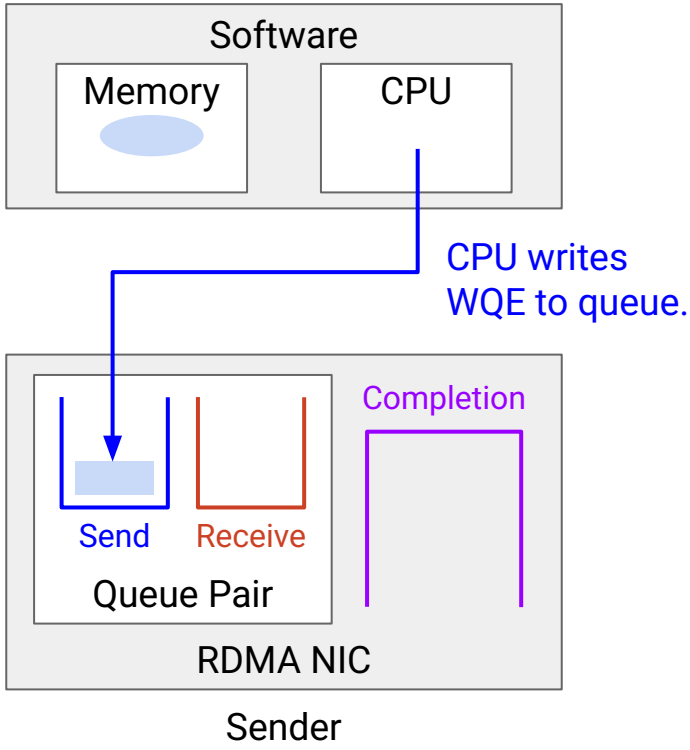


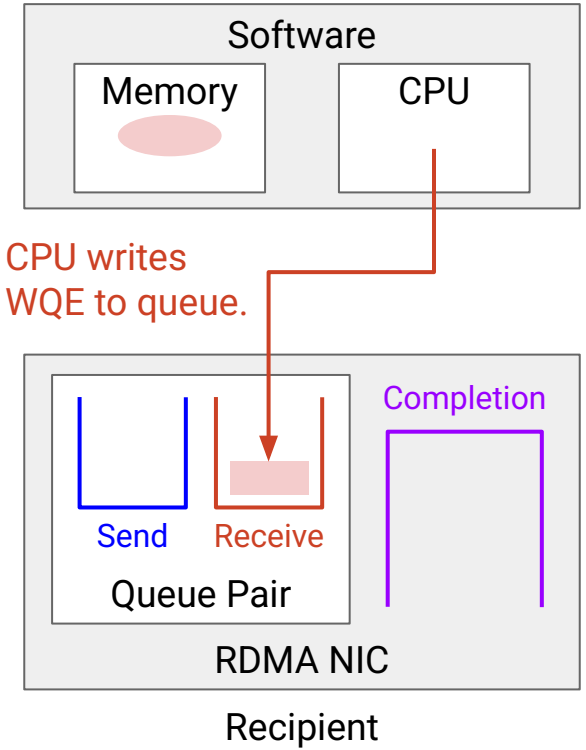


Steps of RDMA (3/6)

3. Servers set up Work Queue Entries (WQEs) in the queues.

WQE contains pointer to buffer in memory.

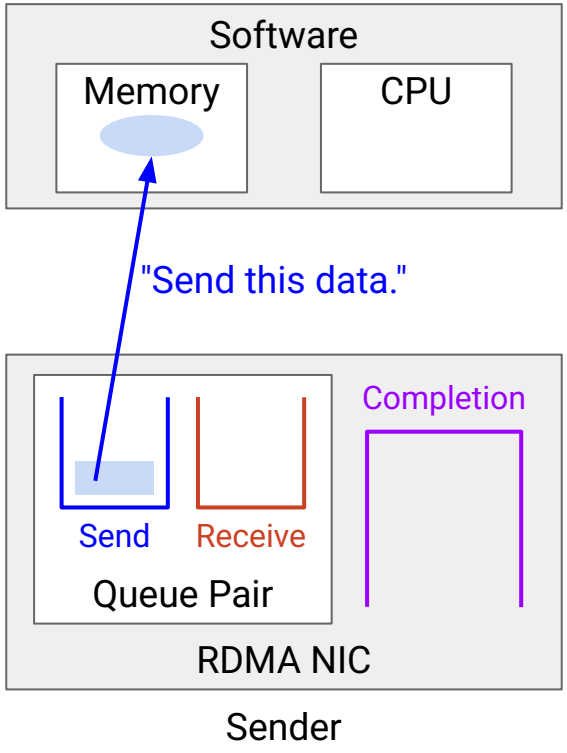


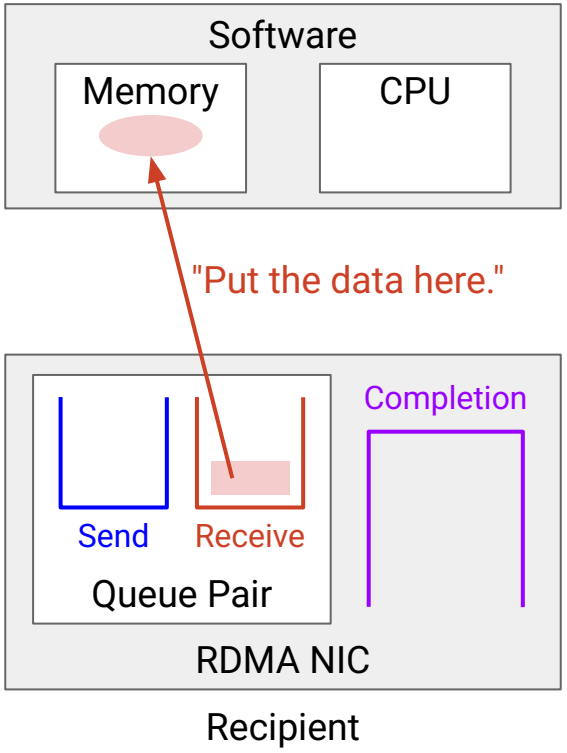


Steps of RDMA (3/6)

3. Servers set up Work Queue Entries (WQEs) in the queues.

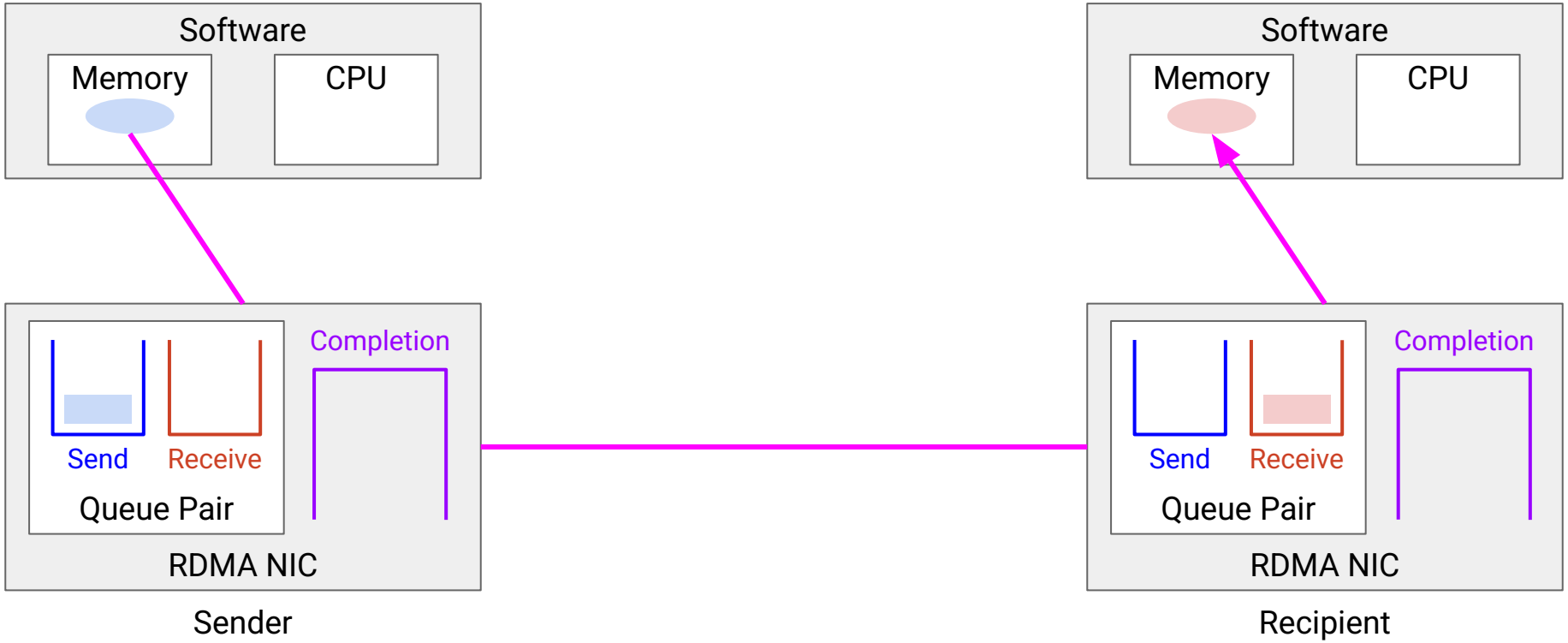
WQE contains pointer to buffer in memory.





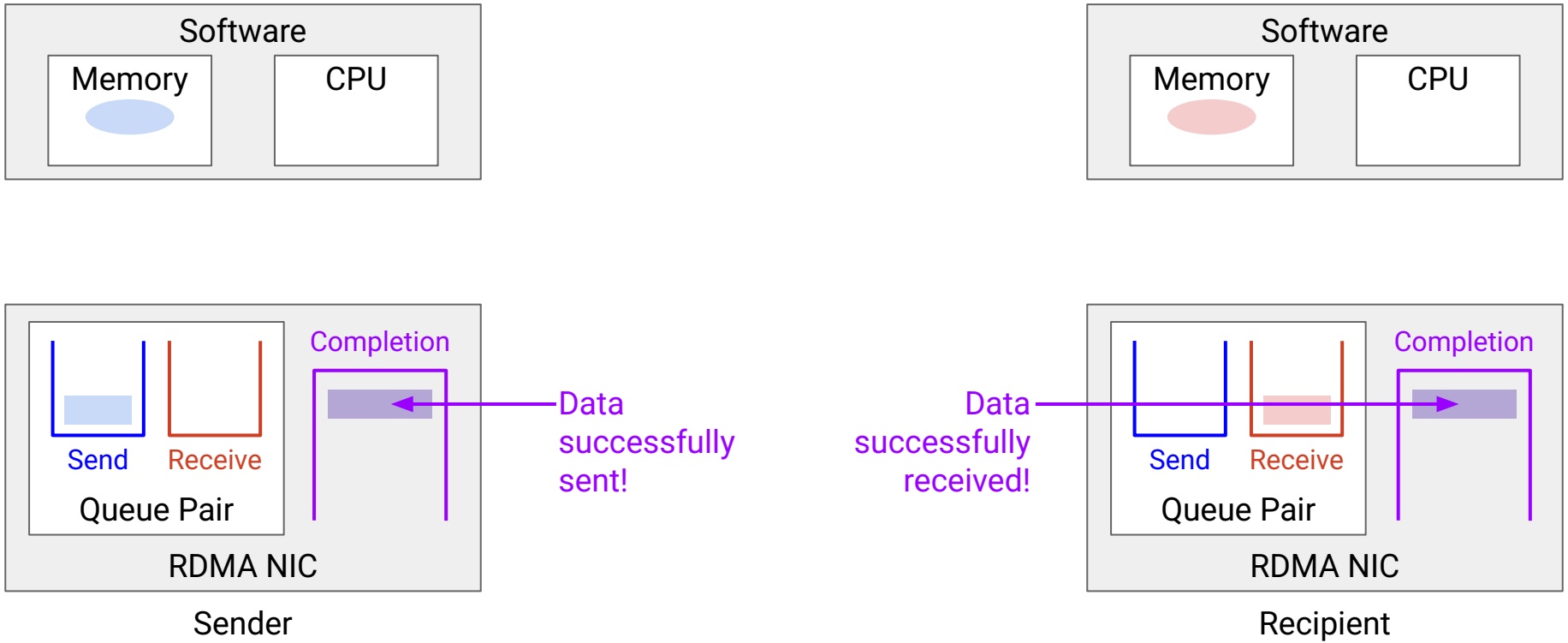
Steps of RDMA (4/6)

4. NIC transfers data between memory, without CPU involvement!



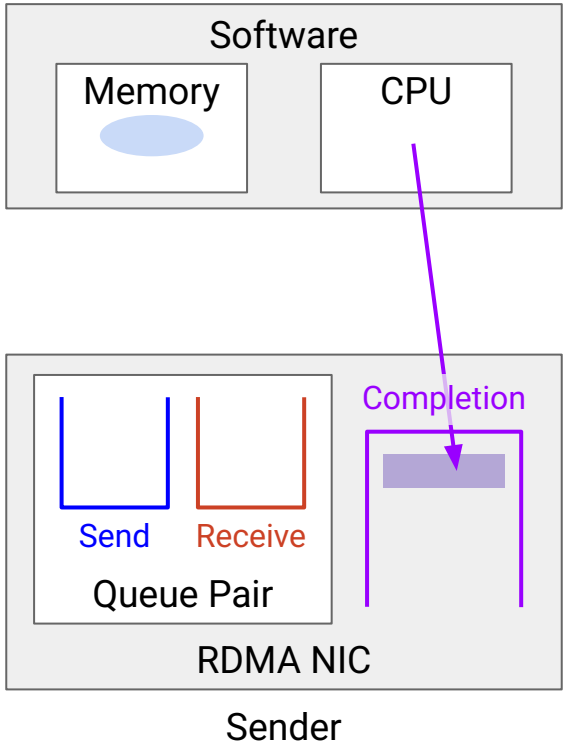
Steps of RDMA (5/6)

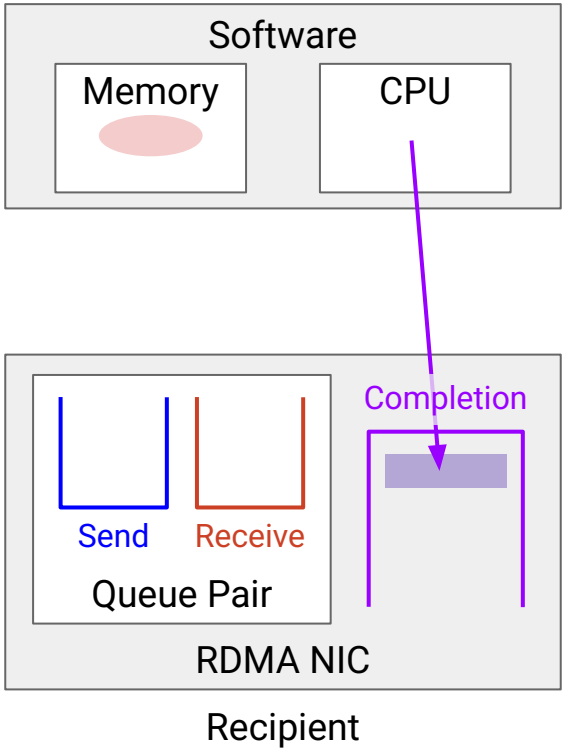
5. NICs generate Completion Queue Entries (CQEs), and delete WQEs.



Steps of RDMA (6/6)

6. Applications read CQE to understand what happened to the transfer.





RDMA: Pros and Cons

Pros:

- High-performance data transfer.
- Frees up CPU for applications.

Limitations:

- More complex. Requires specialized hardware and software.
  - RDMA replaces TCP/IP, so features like reliability must be implemented in hardware.
- Works best when servers are nearby, e.g. in same datacenter.
  - If servers are far away, dominant delay is the network. RDMA savings are insignificant.

RDMA: Applications

RDMA useful for high-performance computing.

- Scientific research, financial modeling, weather forecasting, etc.
- Cloud computing, e.g. migrating large VM from one physical server to another.
- AI/ML training.

Goals:

- Free up CPU for applications.
- Low and predictable latency for data transfers.

RDMA: Implementation

How do you handle losses, reordering, congestion, etc.?

Two broad options:

- Build features into the datacenter network, e.g. at the switches. (*Nvidia's InfiniBand.*)
- Build features in the NIC, under the queue pair abstraction. (*Google.*)

In both cases, applications and OS both see the abstraction of reliable delivery.