Host Networking (Part-I)

Nandita Dukkipati
Lecture at UC Berkeley, April 2024

Lecture Topics

(This Lecture) **Host Networking - Part I**

What is Host Networking and Why it Matters.

The Role of Network Interface Cards (NICs).

Interfacing with Applications using Remote Direct Memory Access (RDMA).

(Next Lecture) **Host Networking - Part II**
Techniques to reduce latency for applications

Congestion Control.

Load Balancing.

Shaping and Pacing traffic.

Quality-of-Service.

Reliable delivery of packets.

Lecture Outline

What is Host Networking and Why it Matters

- Learnt before: network does packet delivery; TCP implements reliability, congestion control, flow control.
- **Problem in Datacenters:** extreme performance requirements; valuable CPU cores; kernel development is hard.
- Host Networking is heavily optimized for application performance and CPU efficiency.
- Optimization Opportunities: **Operating System Bypass**, and **Offloads to the Network Interface Card**.

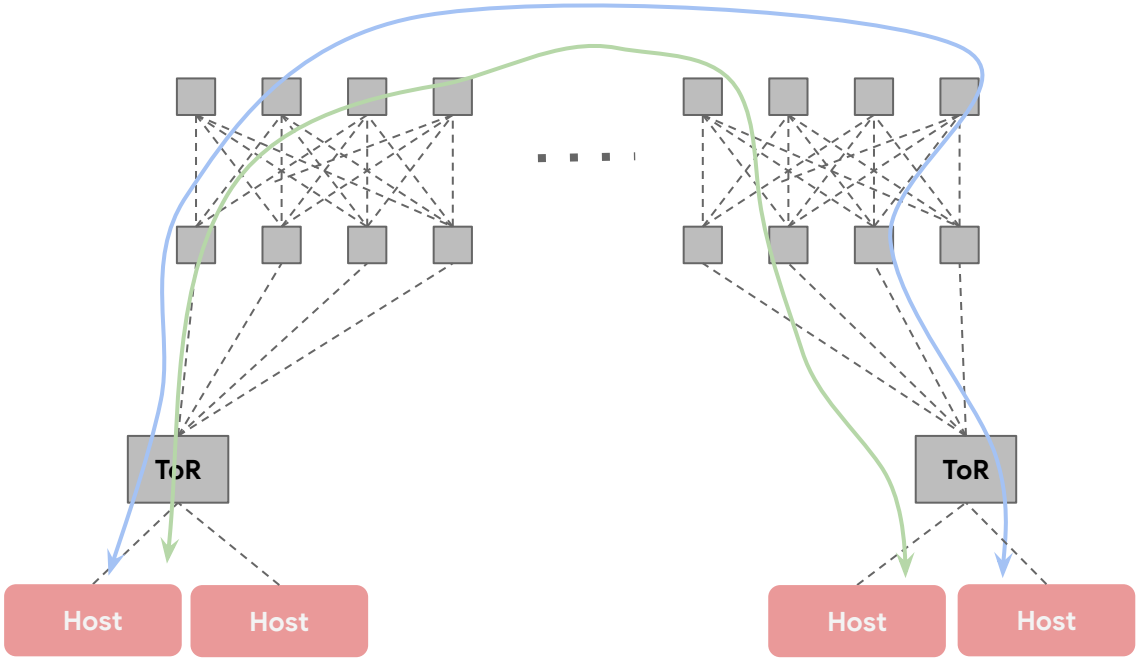
The Role of Network Interface Cards (NICs)

- What is the role of a NIC?
- What are Offloads and Why are they important?
- Offloads on a Spectrum:
    - Simple offloads like Checksum, Segmentation.
    - More complex offloads: Match Action Tables for Network Virtualization.
    - Most complex: offload the entire protocol, e.g. TCP, RDMA.

Remote Direct Memory Access

- What is RDMA; Pros and Cons.
- Applications of RDMA.
- RDMA Building Blocks.
- A walk through of RDMA Send Operation.

End-to-End Performance of Applications



Network starts and ends at Hosts.

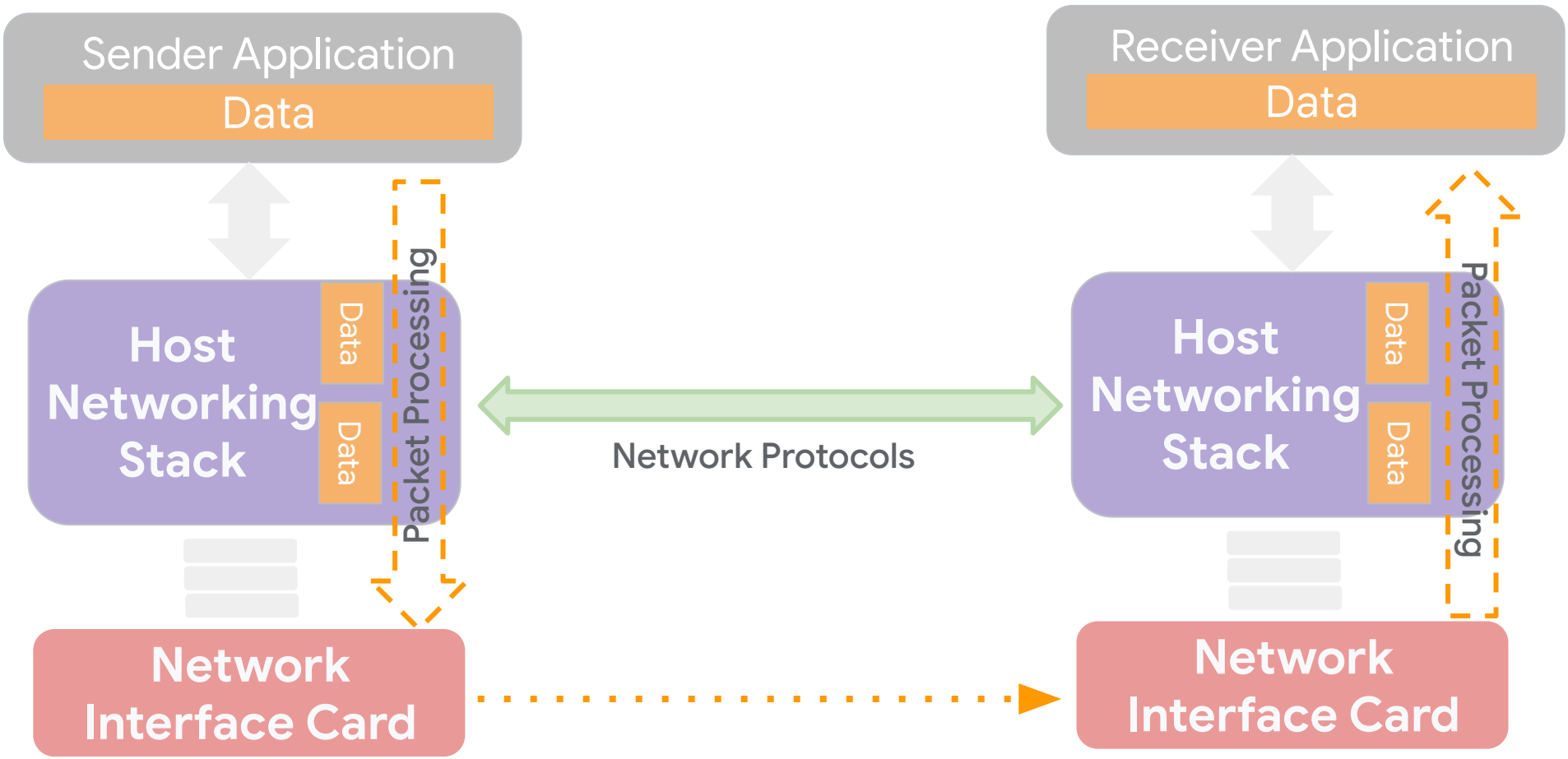
End-to-end application Performance in datacenters hinges critically on Host Networking.

Datacenter applications demand high bandwidth and low latency.

Problem 1:
Traditional networking stacks struggle to deliver high bandwidth and low latency with low CPU overhead.

Problem 2:
Transport protocols designed for Internet applications (e.g., TCP) are insufficient to meet low latency requirements of datacenter applications.

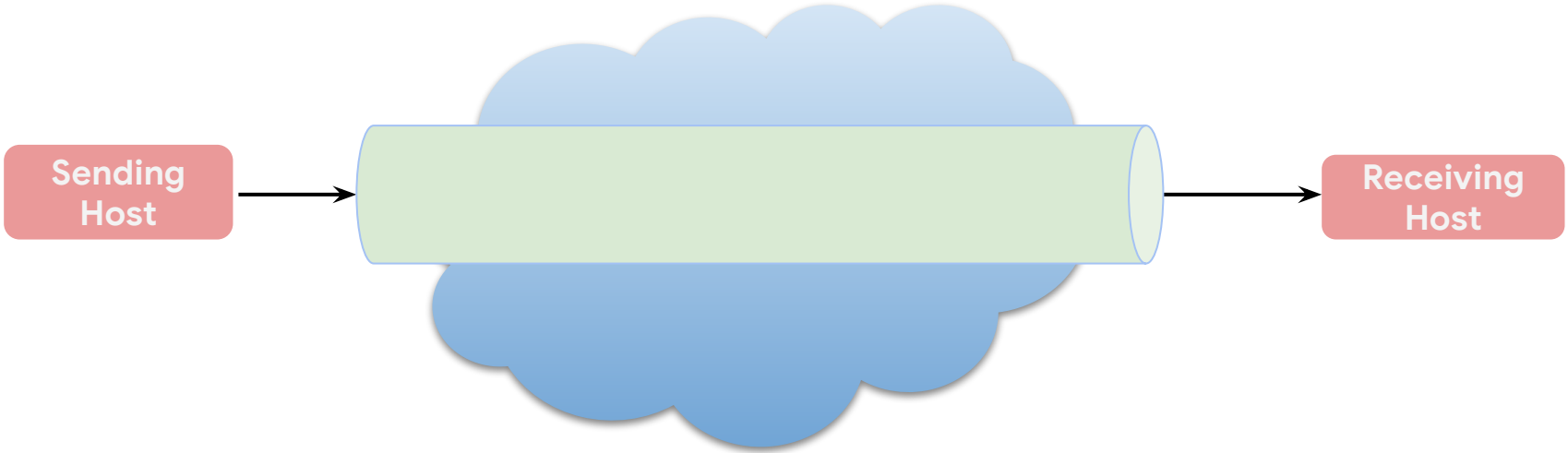
What constitutes Host Networking?



Protocols running in Hosts: the abstraction

Transport protocols, such as TCP, offer the abstraction of a fast, reliable, secure ordered byte stream.

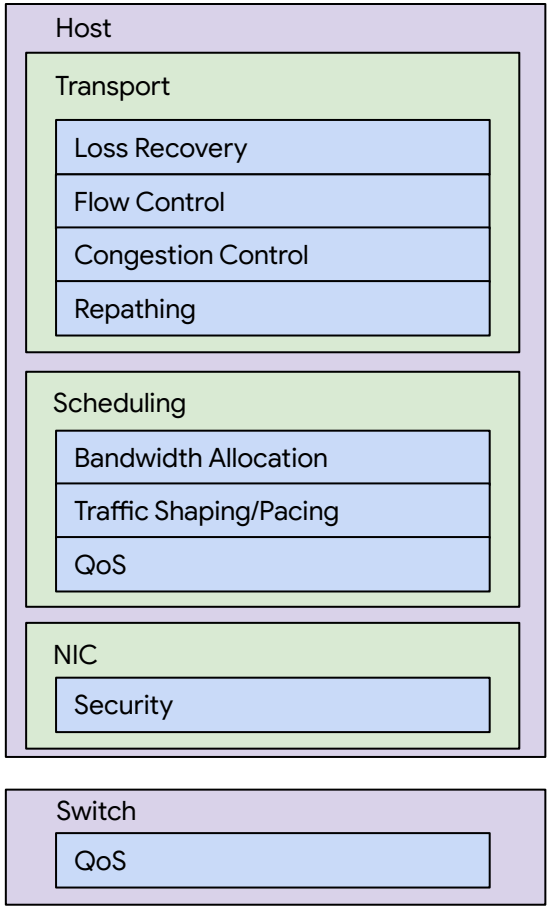
Implemented on an insecure, unordered, lossy datagram network with varying speed and reliability.



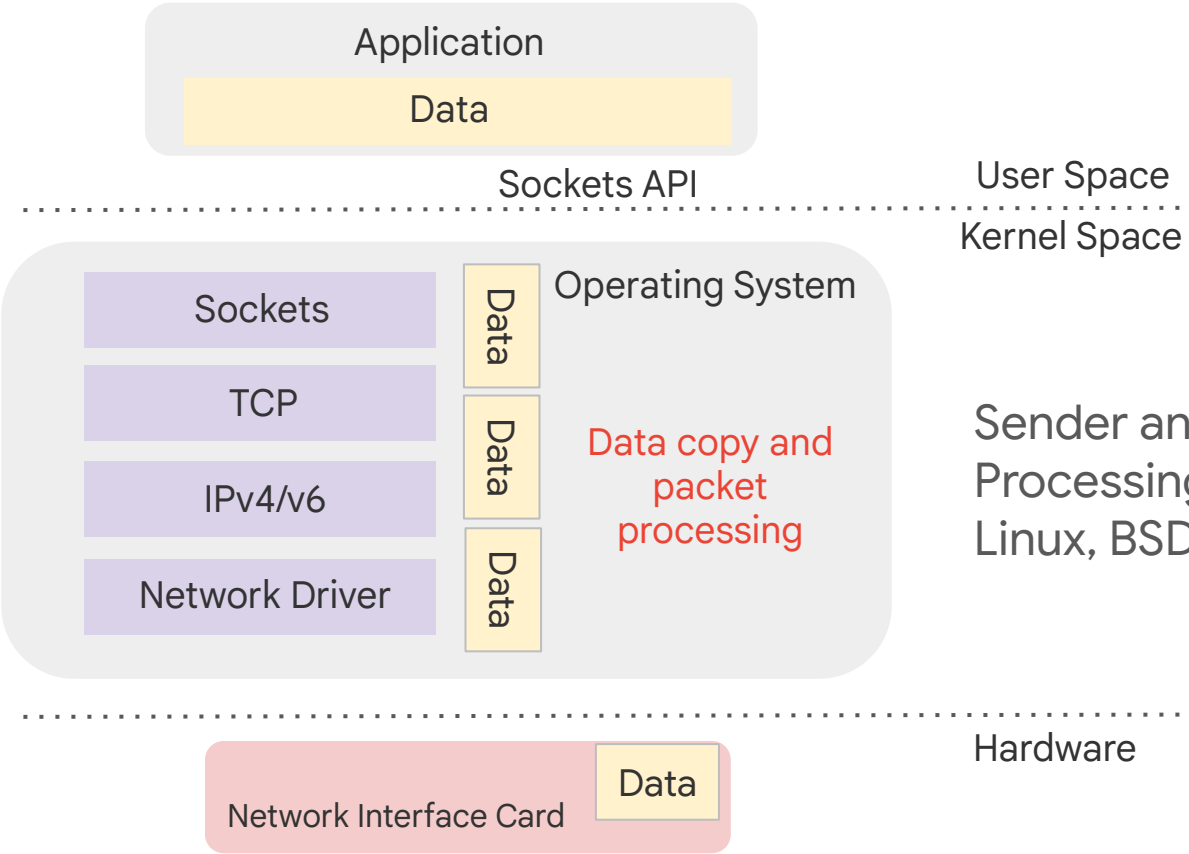
Protocols running in Hosts: Implementing the abstraction

Green: covered before
Purple: next lecture
Orange: not covered in either lecture

- **Congestion Control**: how fast to send, to avoid overloading the network?
- **Loss Recovery**: what to send: how to infer which packets were lost, for retransmissions for reliability?
- **Flow Control**: how fast to send, to avoid overloading the receiver's memory and/or CPU?
- **Load Balancing**: which path/paths should the traffic travel to avoid congestion and black holes?
- **Traffic Shaping/Pacing**: when should each packet be sent, to maintain short network queues?
- **Quality of Service (QoS)**: what's the priority of this traffic for allocating buffers and bandwidth at each hop, at <RTT time scales?
- **Bandwidth Allocation**: how much bandwidth is this user allowed on this path, on long time scales?
- **Security**: how to ensure traffic has a known src/dst user (authentication) and content is correct (integrity / checksums) and secret (encryption)?

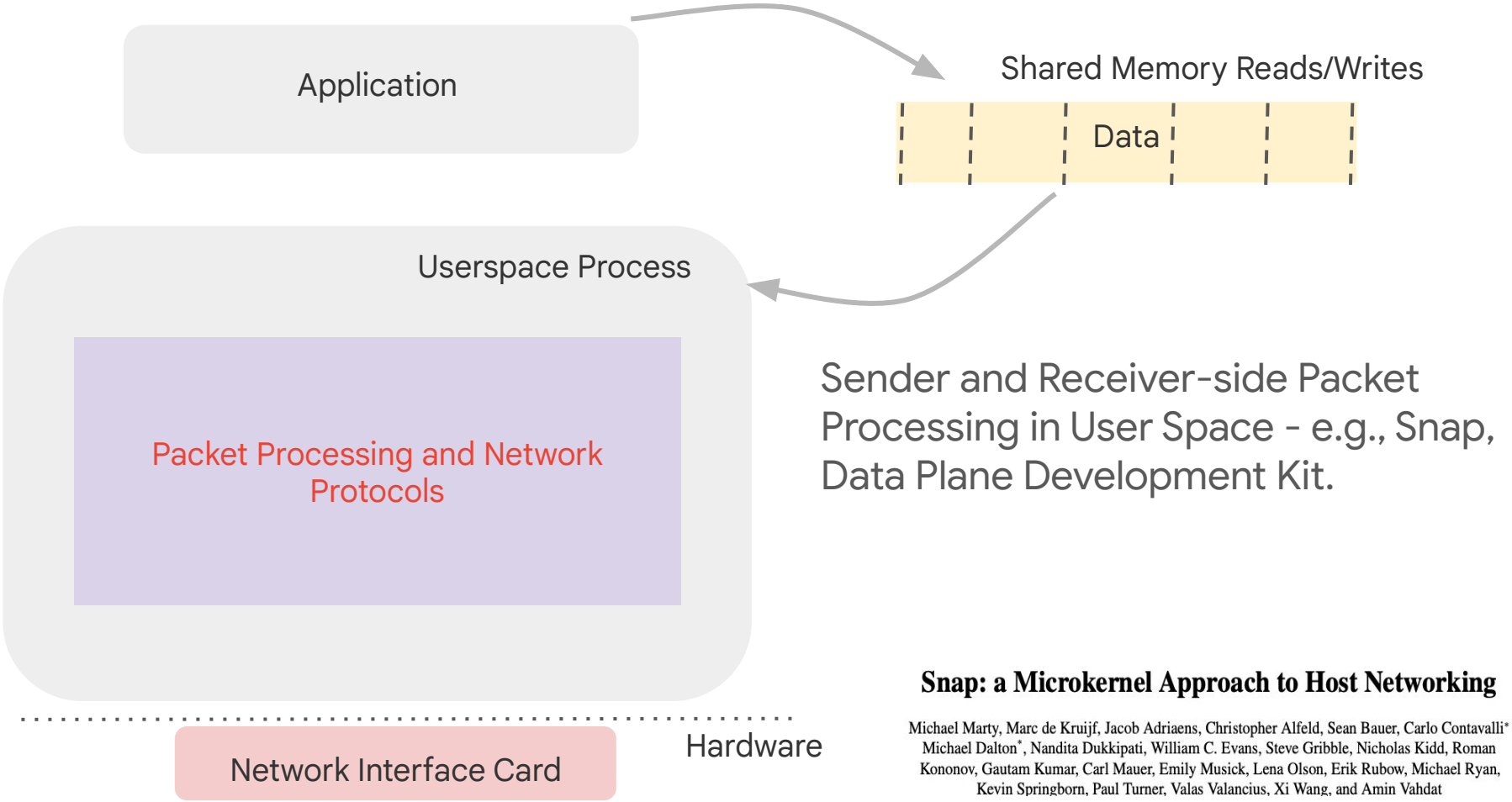


Traditional Networking Stack in Operating System



Sender and Receiver-side Packet Processing in the Operating System - Linux, BSD.

Operating System bypass Stacks: Networking Stack in Userspace



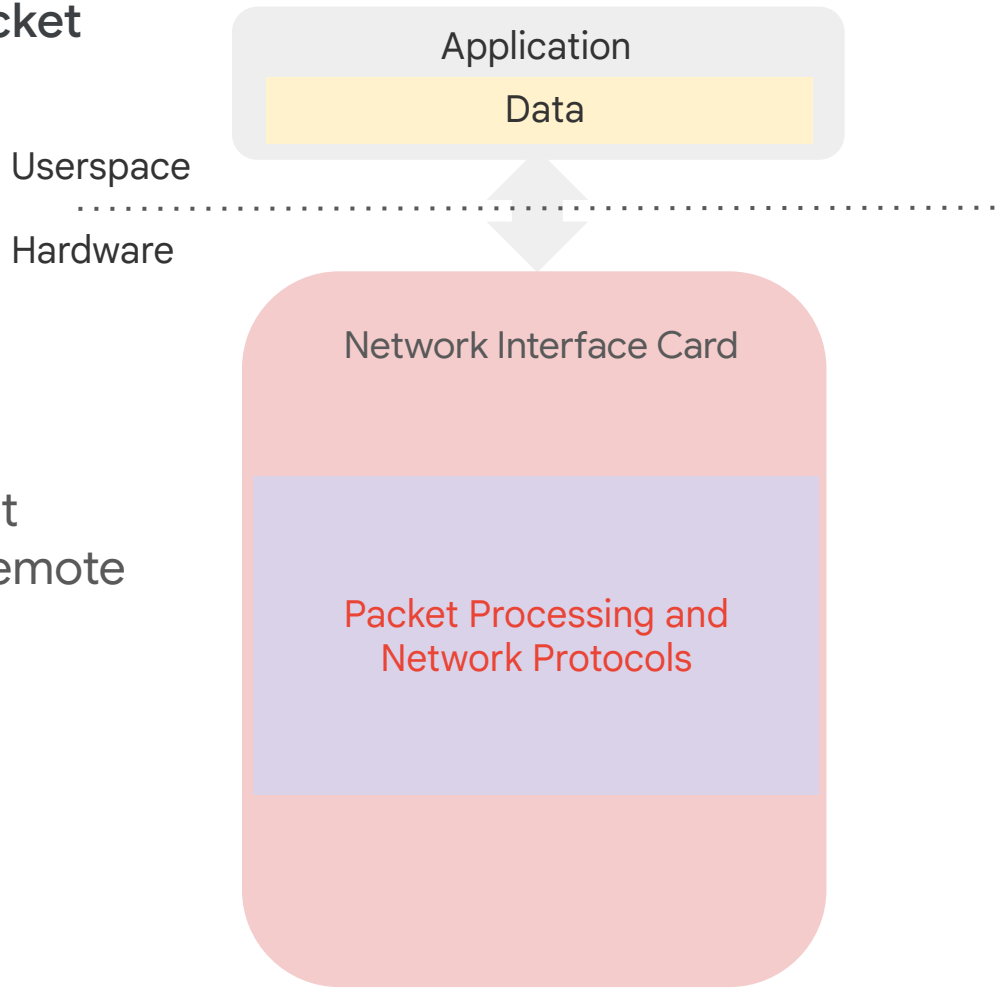
Sender and Receiver-side Packet Processing in User Space - e.g., Snap, Data Plane Development Kit.

Snap: a Microkernel Approach to Host Networking

Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli* Michael Dalton*, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat

Offloaded Networking Stack: Packet Processing in NIC Hardware

Sender and Receiver-side Packet Processing in Hardware - e.g., Remote Direct Memory Access (RDMA).



Lecture Outline

What is Host Networking and Why it Matters

- Learnt before: network does packet delivery; TCP implements reliability, congestion control, flow control.
- Problem in Datacenters: extreme performance requirements; valuable CPU cores; kernel development is hard.
- Host Networking is heavily optimized for application performance and CPU efficiency.
- Optimization Opportunities: Operating System Bypass, and Offloads to the Network Interface Card.

The Role of Network Interface Cards (NICs)

- What is the role of a NIC?
- What are Offloads and Why are they important?
- Offloads on a Spectrum:
  - Simple offloads like Checksum, Segmentation.
  - More complex offloads: Match Action Tables for Network Virtualization.
  - Most complex: offload the entire protocol, e.g. TCP, RDMA.

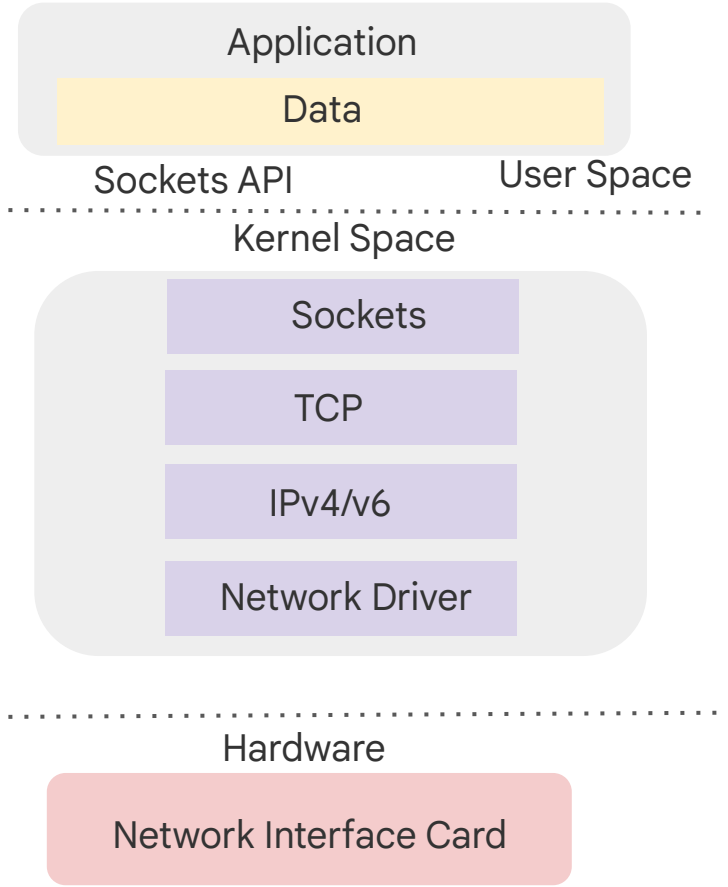
Ref: Link to an IETF Talk

Remote Direct Memory Access

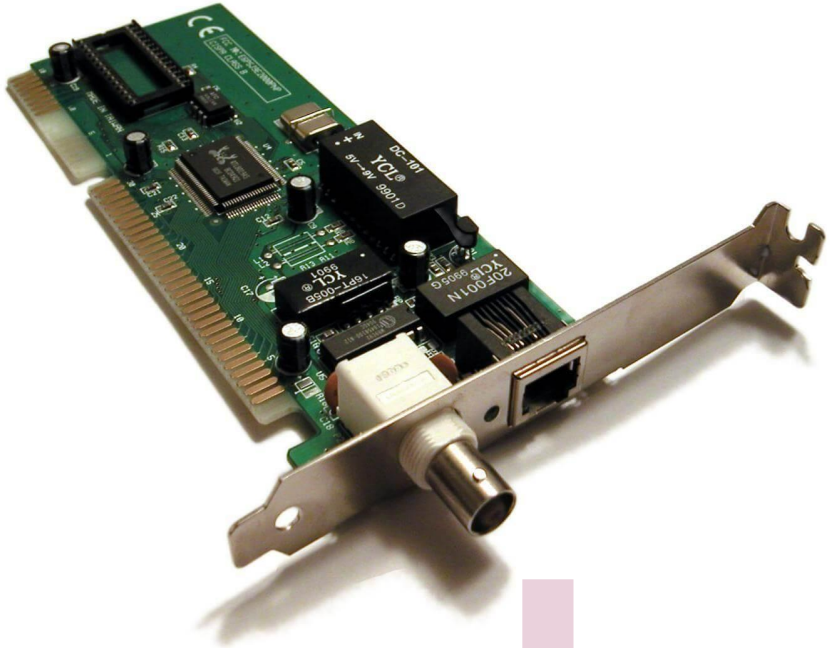
- What is RDMA; Pros and Cons.
- Applications of RDMA.
- RDMA Building Blocks.
- A walk through of RDMA Send Operation.

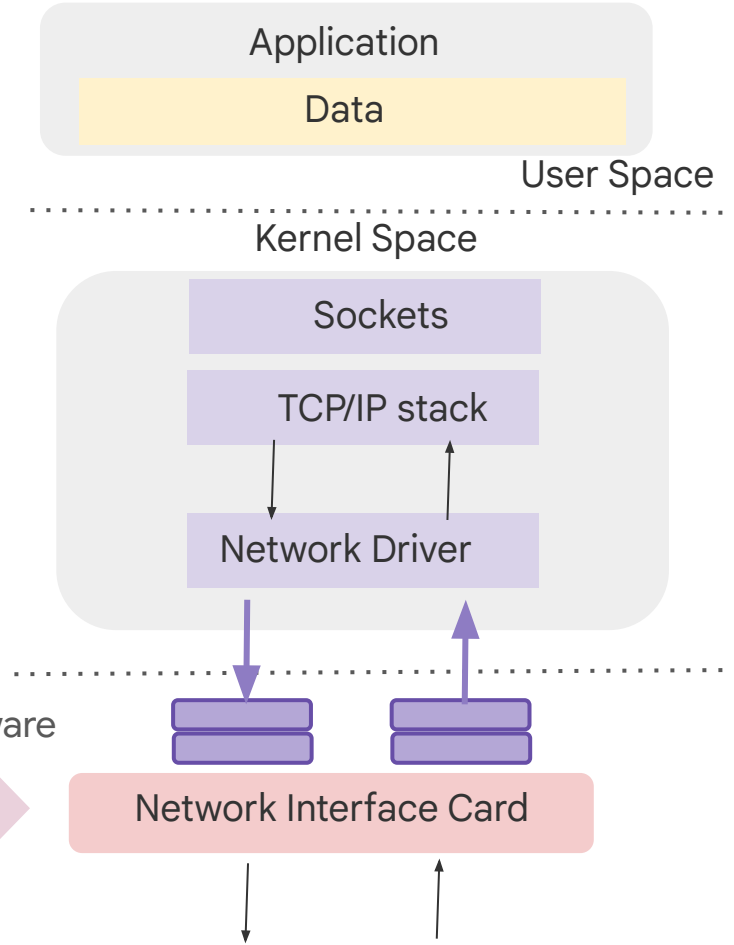
Fundamentals of Network Interface Card (NIC)

- Host Networking Stack is the software stack in the host that performs packet processing across L2/L3/L4 layers.
  - Kernel networking stacks are implemented in the Operating System.
  - OS-bypass stacks are implemented in user-space.
- Network Interface Card is host's interface to the network.
- Offloads - perform processing in the NIC, what one would normally do in host software stack.

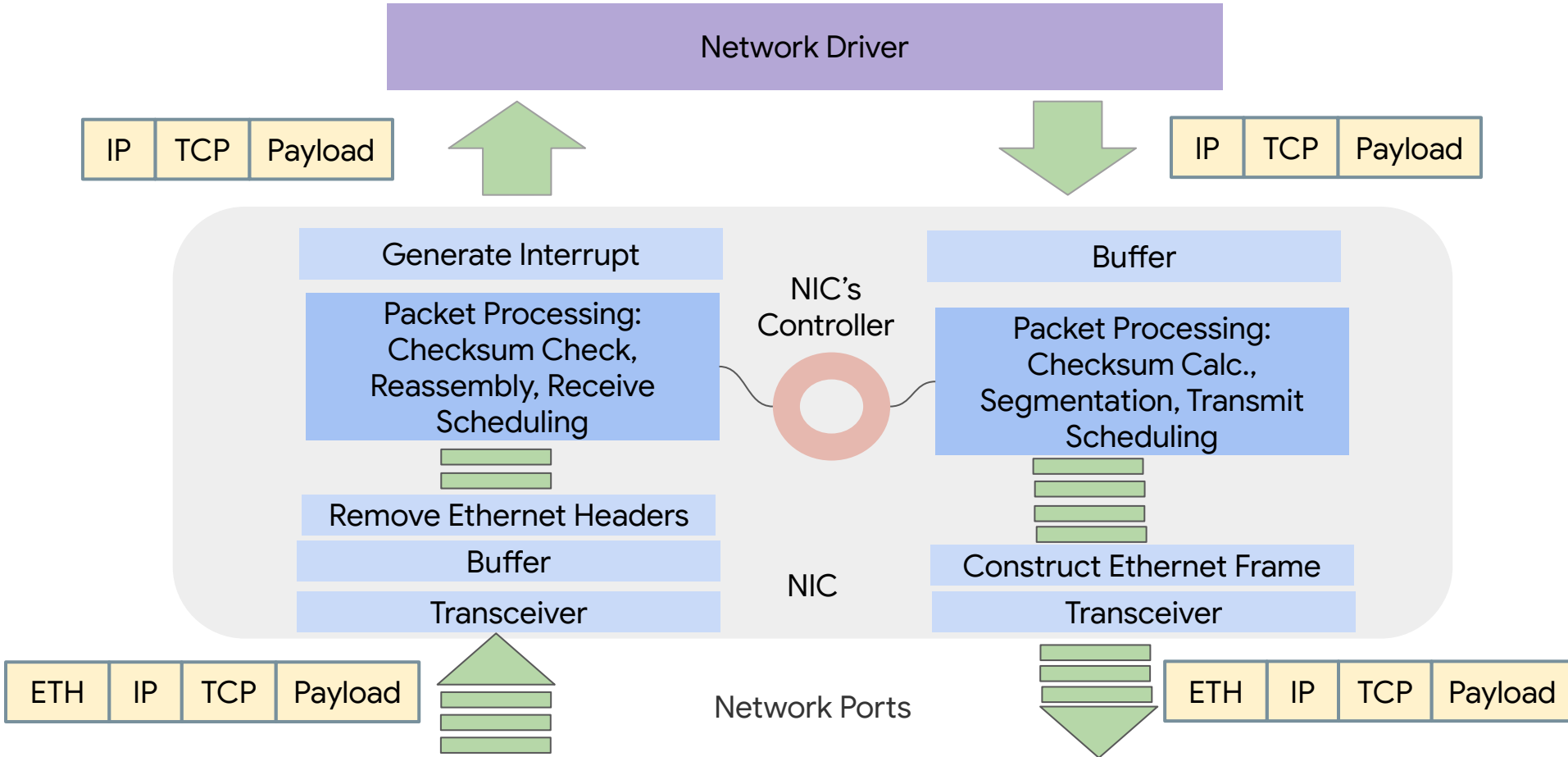


Fundamentals of Network Interface Card (NIC)





Life of Packet in the NIC



Spectrum of Offloads to the NIC

- Epoch 1 - Basic Support
  - Transmit and Receive packets.
  - Simple Stateless Offloads.
- Epoch 2 - Accelerating the dataplane
  - Accelerating more complex parts of host networking stack.
  - Stateful offloads - Bandwidth Management, Forwarding etc.
- Epoch 3 - Protocol Offloads
  - General purpose processor with programmable data plane.
  - Protocol State Machine Offloads (protocols like TCP, RDMA).

Why Offload?

- Free up host CPU cycles for applications
  - Network processing and applications no longer contend for CPU.
- Efficiency
  - Specialized processing in hardware can be more efficient.
  - Power savings.
- Performance
  - Scaling throughput.
  - Predictable low latency.

Epoch 1: Simple, Stateless Offloads of Network Processing

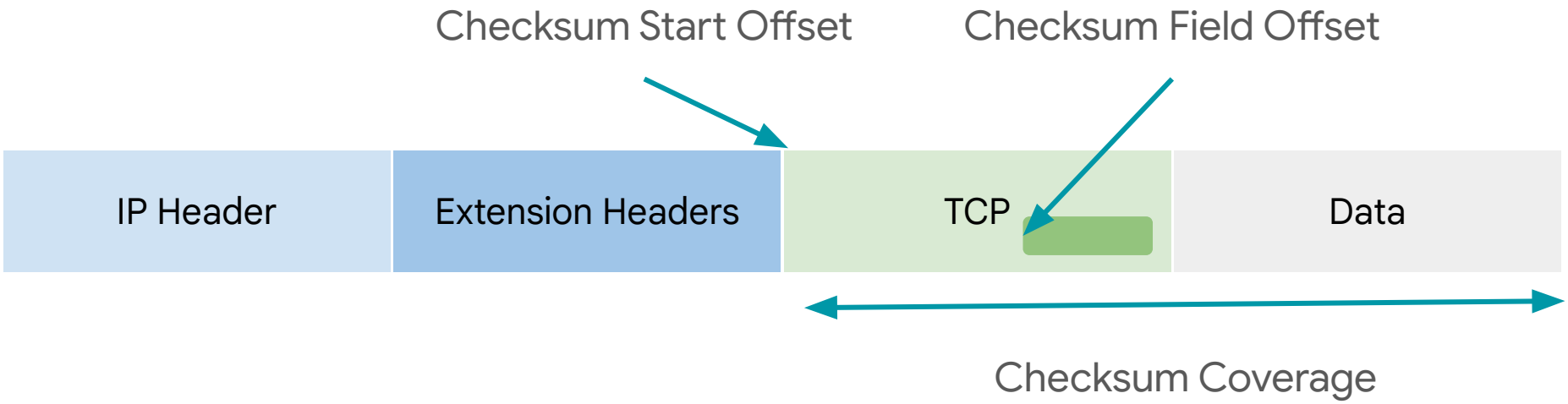
- Checksum Offload (skip in lecture; material in slides)
  - Transmit and Receive Checksum Offloads.
- Segmentation Offloads
  - Transmit and Receive Segmentation Offloads.
- Multi Queue Support
  - Transmit Queue Selection, Receive Packet Steering.

Checksum Offload

- Checksums are small blocks of data derived from a larger chunk of data and are used for error detection.
  - Sender calculates a checksum and includes it in the packet.
  - Receiver recalculates the checksum on their end. If the values match, the data is likely intact.
- Checksum Offload is NIC calculation over data.
  - Ubiquitous in NICs for TCP, UDP, and other protocols.
- Examples for checksum calculations using one's complement.

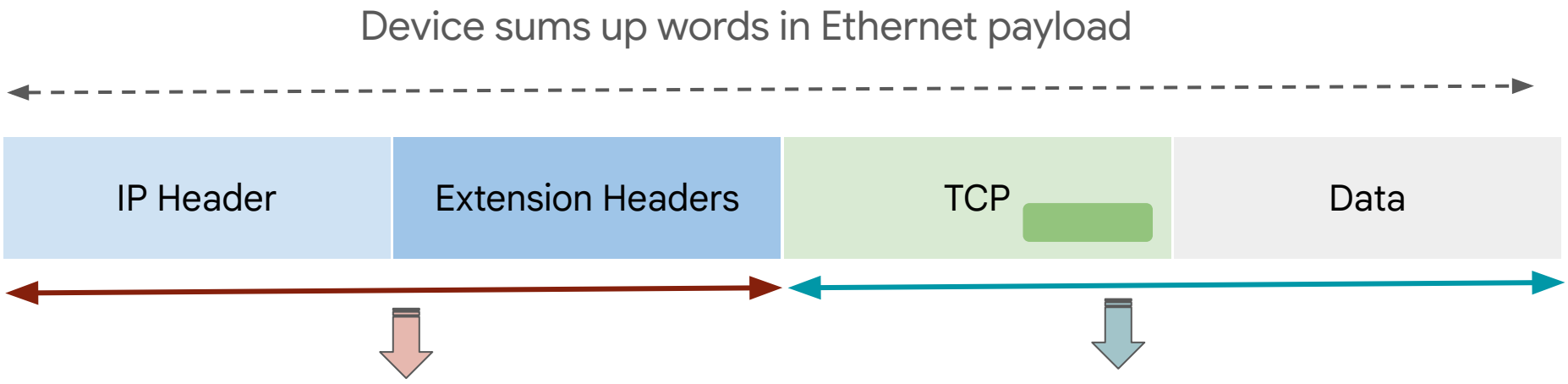
Transmit Checksum Offload

- NIC parses packets and **sets** TCP and UDP checksum.
- Networking stack instructs NIC where to start and write checksum.
  - Start offset for computing the checksum, and offset to write checksum to.



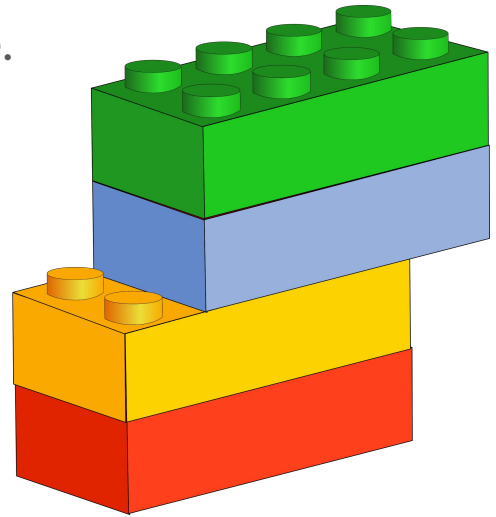
Receive Checksum Offload

- NIC parses packets and **verifies** TCP and UDP checksum.
- NIC computes 1's complement sum across words in the packet.



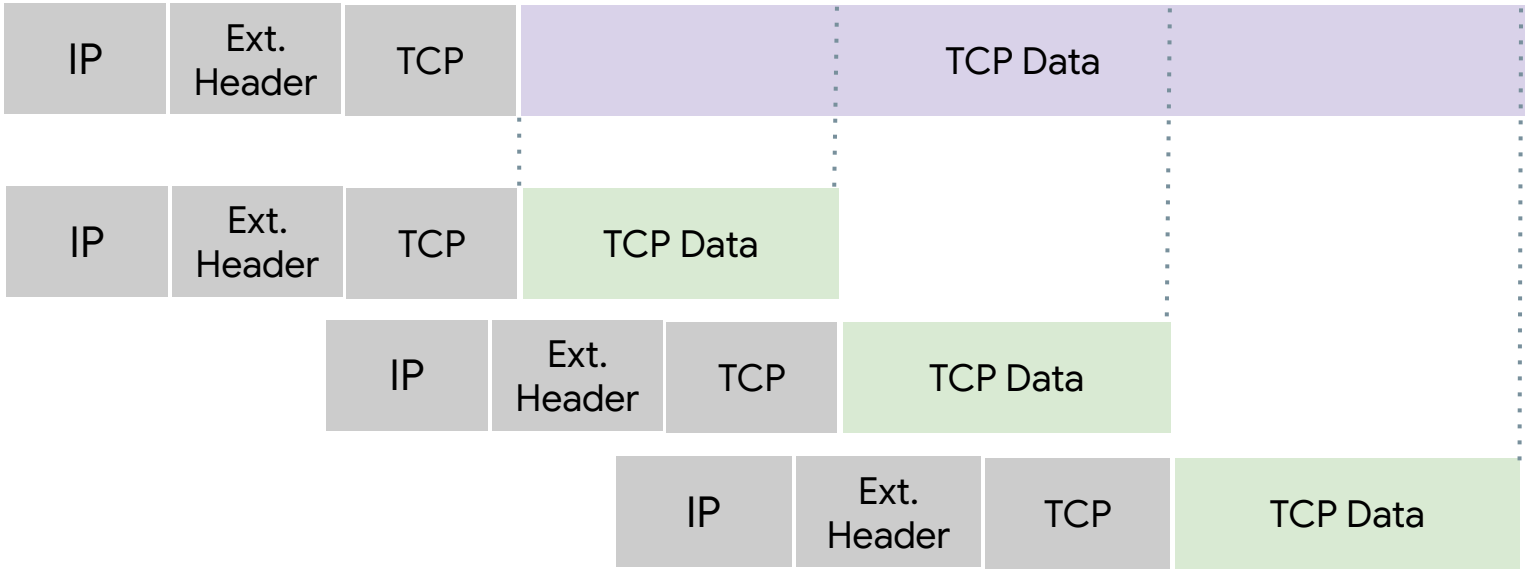
Segmentation Offload

- Key metrics for Networking Stack/NICs: Bandwidth, Packet latency, Packet Rate (small packets).
- Segmentation Offload: Host handles large packets, NIC handles MTU sized ones.
- Host Networking stack in SW is more efficient when operating on large packets.
    - Can maximize throughput while optimizing CPU usage.



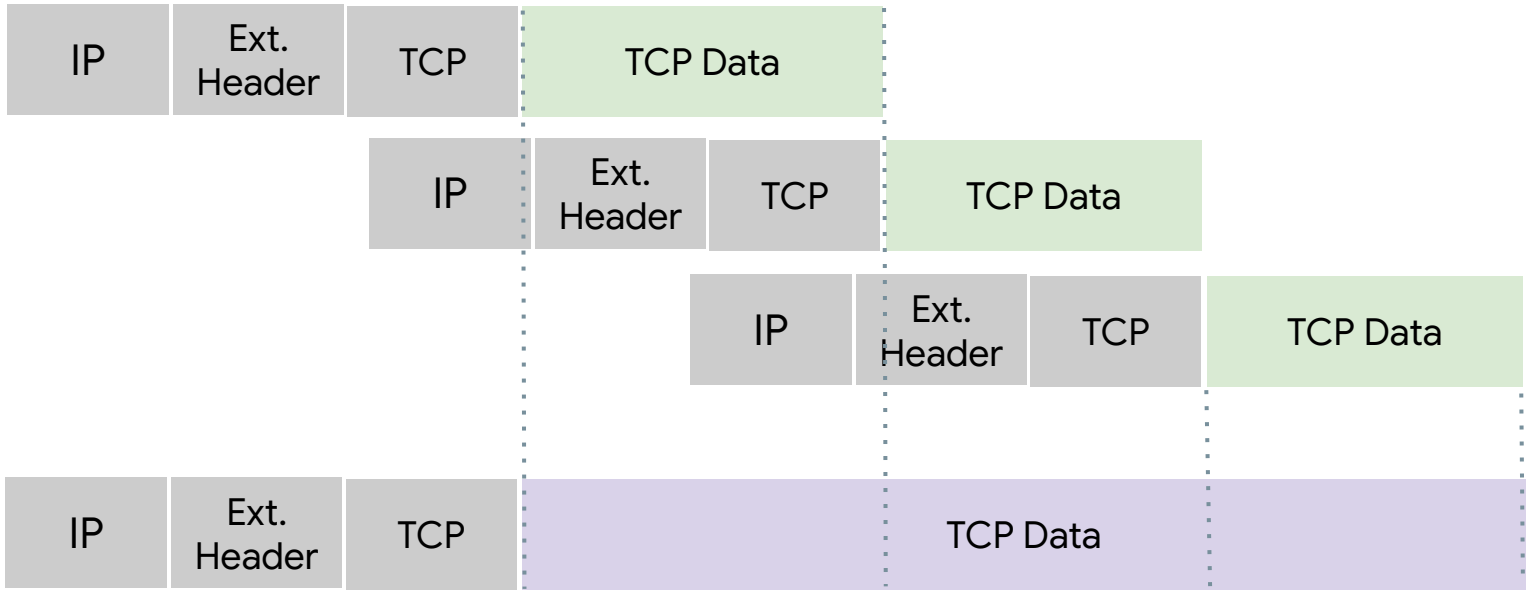
Transmit Segmentation Offload

- Split big packets into MTU sized ones.



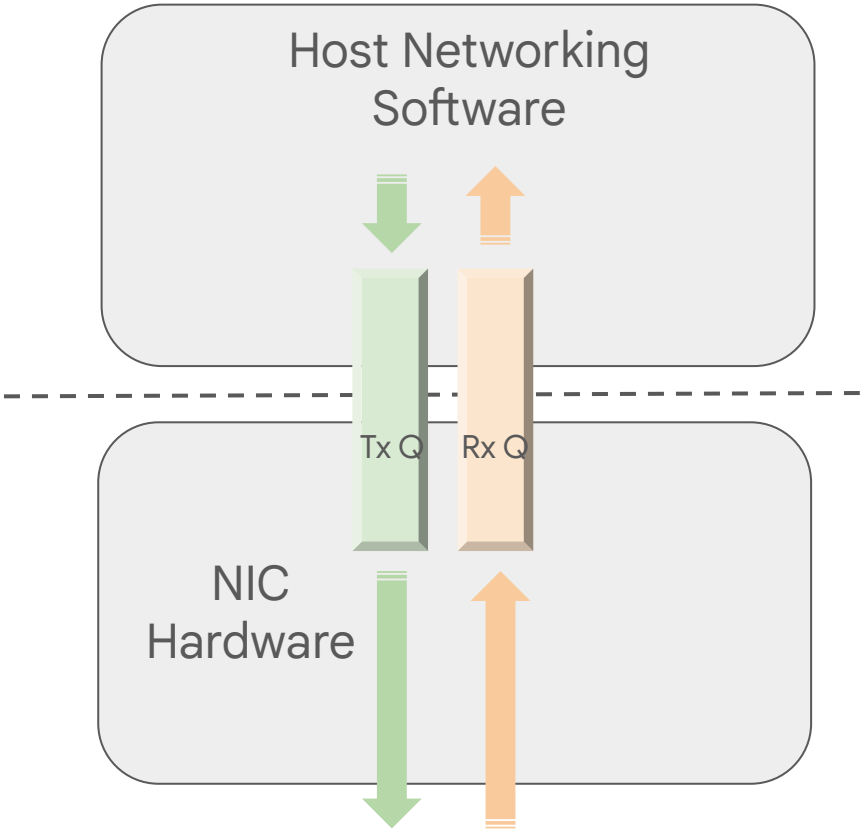
Receive Segmentation Offload

- Coalesce small packets into bigger ones.
- Difficult to make it work efficiently **and** make it work correctly with congestion signals.



NIC with Single Queue

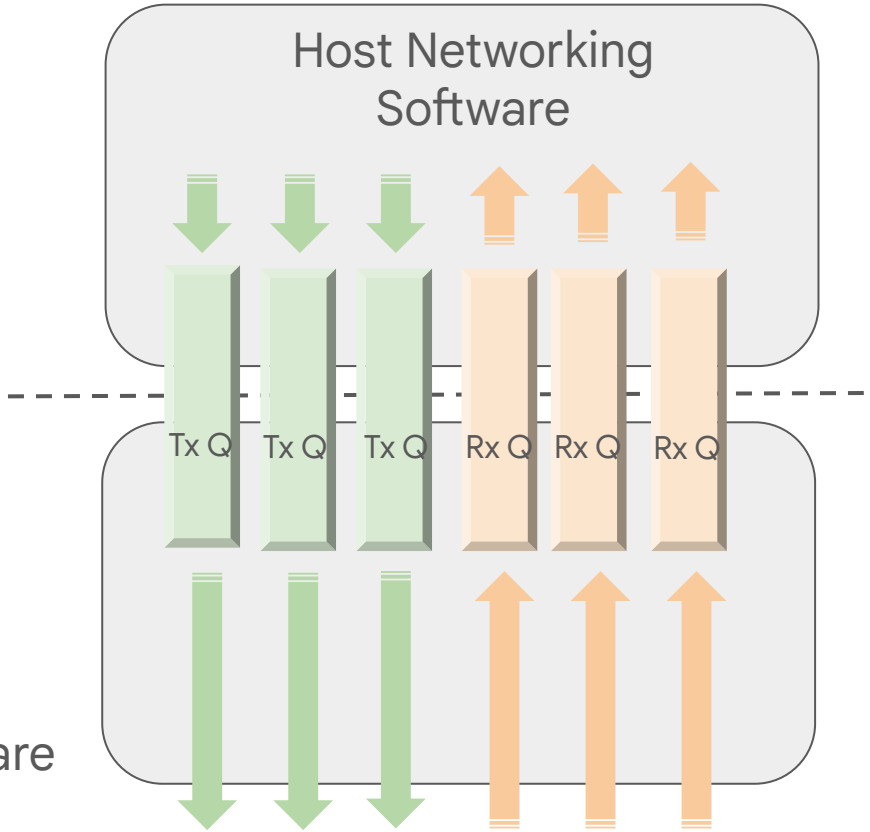
What happens with just one Transmit and one Receive Queue in NIC?



Multi-Queue NICs

Multiple Queues exposed by the NIC.

- Queues accessed for load balancing network processing across CPU cores.
- Prioritized Scheduling of Queues.
- Dedicated for certain applications.



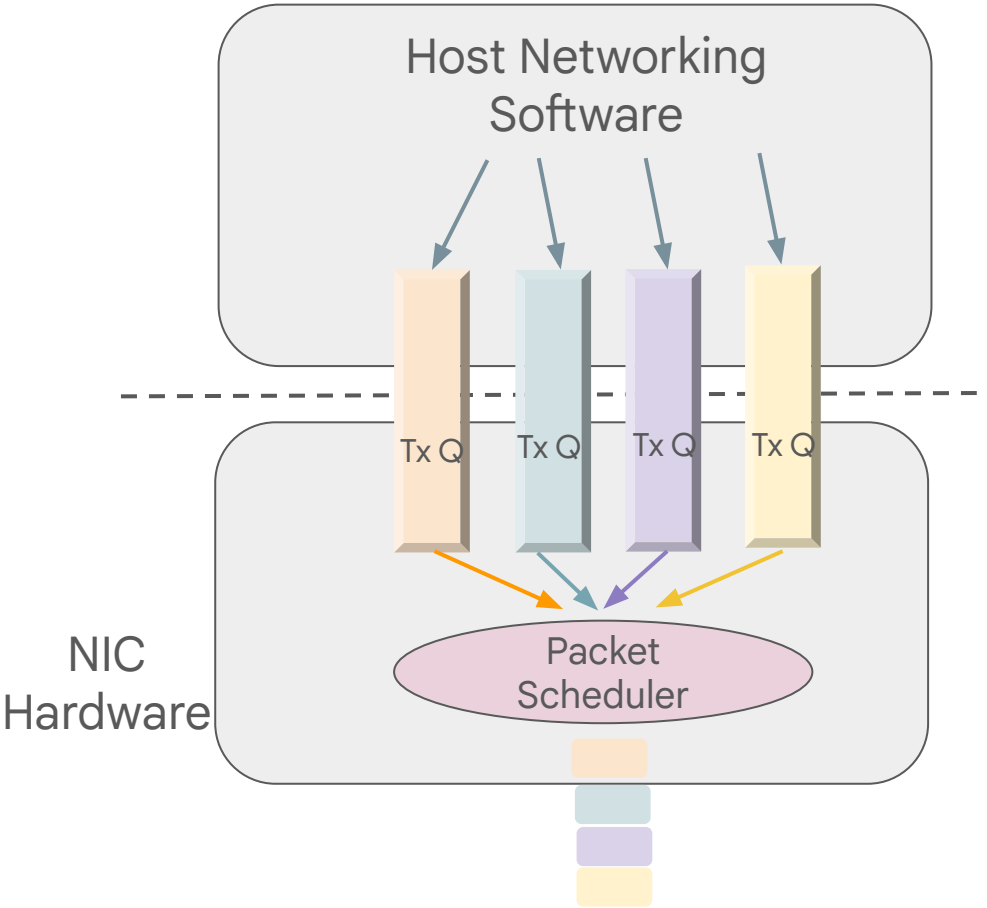
Transmit Queue Selection

Transmit Packet Steering.

- Send packet on the queue associated with specific CPU or application.

Considerations

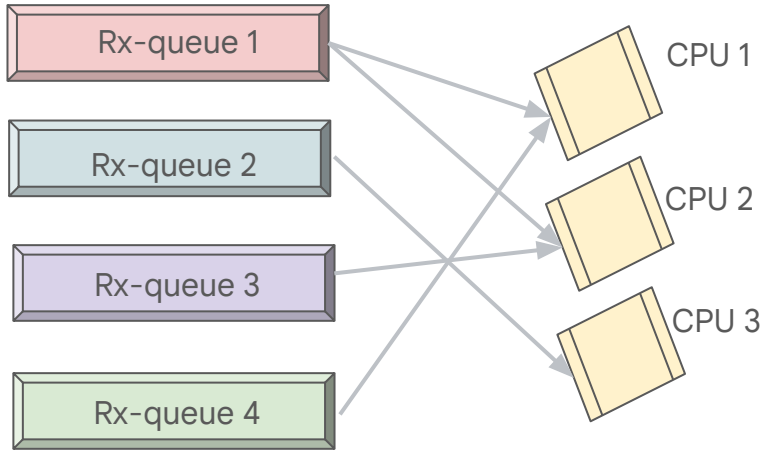
- Avoid out of order packets within a TCP flow. Maintain a TCP flow to queue affinity.



Receive Side Scaling

Receive Side Scaling in NIC hardware

- NIC distributes packets across multiple Receive queues; steer to queue based on hash.
- Selected CPU to run interrupt handler on.



Epoch 2: Stateful offloads of Host Networking Stack

Offloads in support of Cloud Virtual Machines

- Bandwidth metering
- Many others functions also offloaded.
  - Virtual Routing Tables, Forwarding
  - Load balancers
  - Access Control Lists
  - Quality of Service

Why Offload?

- Free up host CPUs for customer Virtual Machines (VMs).
- Reduce overall cost of providing cloud services.
- Performance: higher throughput, lower latency and variability.

Match Action Tables

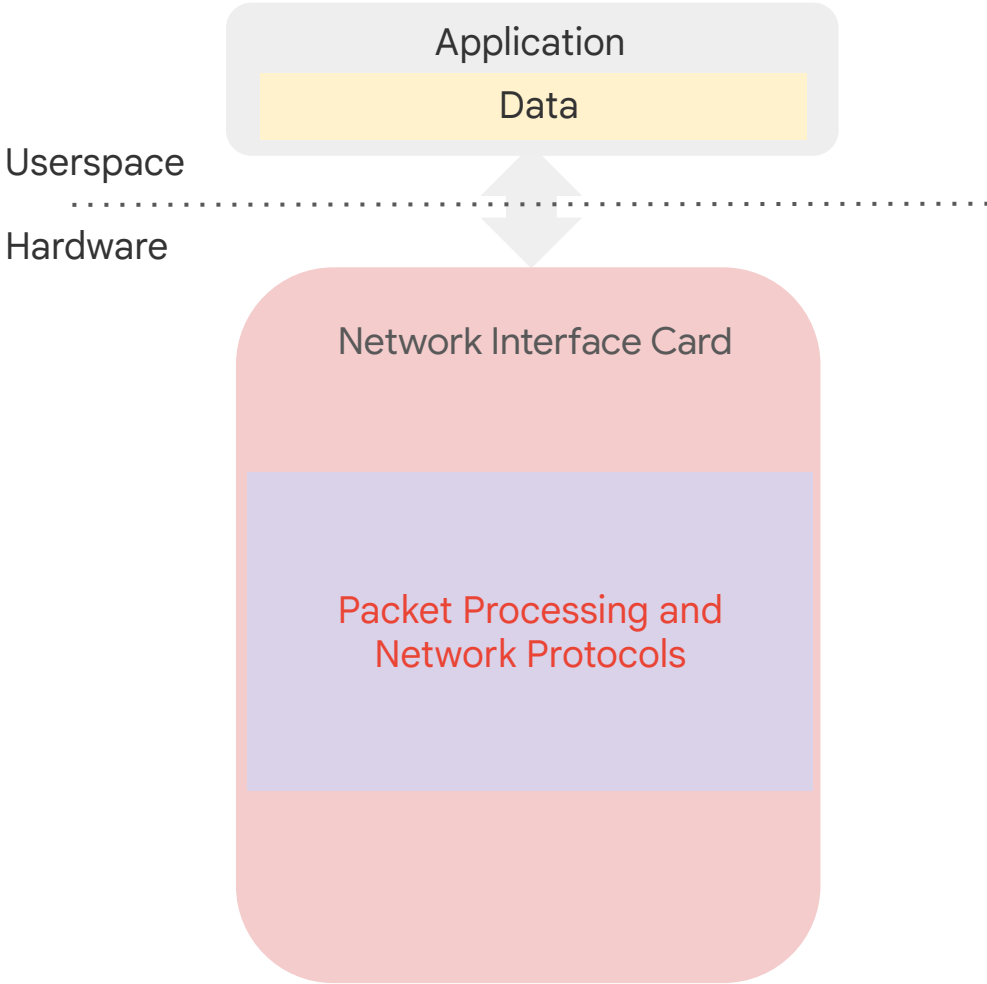
- Match packets based on headers and metadata.
- Execute Actions based on Match.
  - **Pass through, Drop.**
  - Packet / metadata modification.
  - Connection Tracking.
  - Firewall actions.
  - Billing.

src = 1.2.*.*, dest=*.*.*.* src = 10.1.2.3, dest=*.*.*.*	Pass packet if burst < X within 30 second interval.

Flow Table	
Match	Action

Epoch 3: Protocol Offloads

- Insatiable demand for bandwidth, low latency along with high efficiency.
- Protocol level offloads for low latency in support of AI/ML and High Performance Computing workloads.
- TCP like protocols (reliability, congestion control, ordering) are offloaded to the NIC.



Lecture Outline

What is Host Networking and Why it Matters

- Learnt before: network does packet delivery; TCP implements reliability, congestion control, flow control.
- Problem in Datacenters: extreme performance requirements; valuable CPU cores; kernel development is hard.
- Host Networking is heavily optimized for application performance and CPU efficiency.
- Optimization Opportunities: Operating System Bypass, and Offloads to the Network Interface Card.

The Role of Network Interface Cards (NICs)

- What is the role of a NIC?
- What are Offloads and Why are they important?
- Offloads on a Spectrum:
    - Simple offloads like Checksum, Segmentation.
    - More complex offloads: Match Action Tables for Network Virtualization.
    - Most complex: offload the entire protocol, e.g. TCP, RDMA.

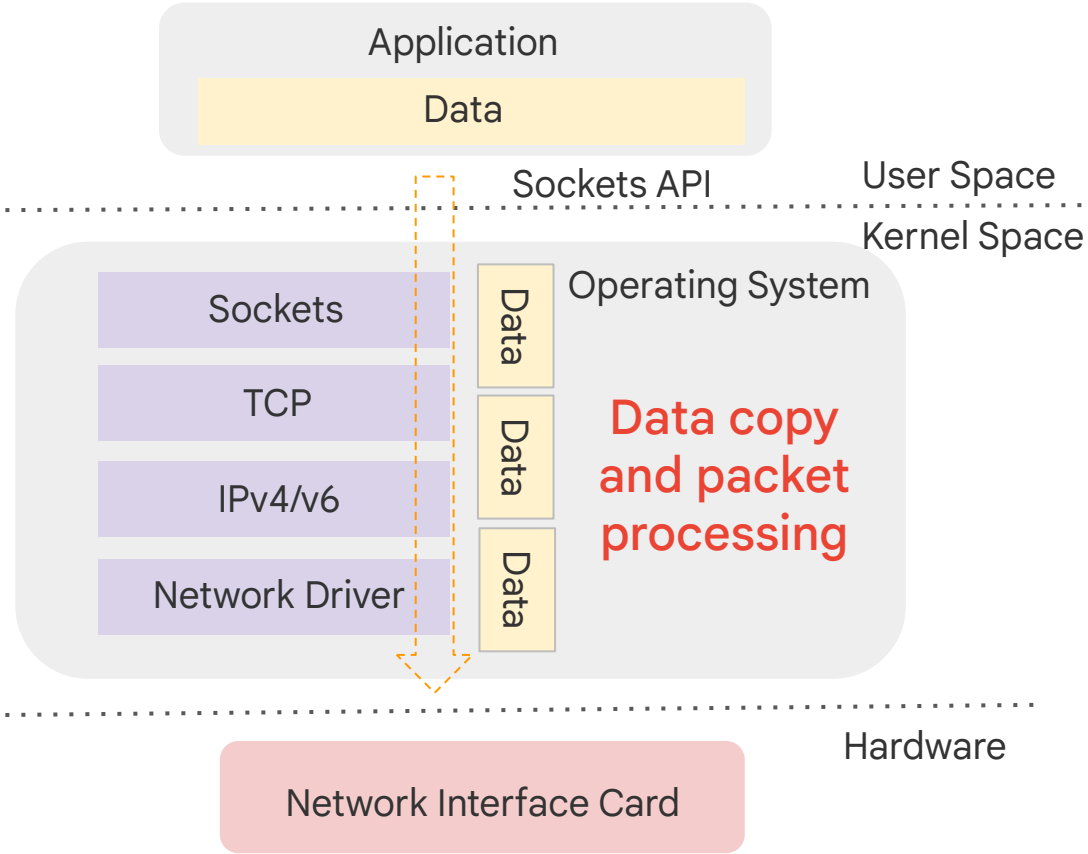
Remote Direct Memory Access

- What is RDMA; Pros and Cons.
- Applications of RDMA.
- RDMA Building Blocks.
- A walk through of RDMA Send Operation.

Topics we will cover

- Problems with Byte streaming transfers in traditional networking stacks.
- An overview of RDMA.
- RDMA Pros/Cons and Applications.
- Key Components for transferring data in RDMA.
- A walk through of an RDMA Send operation.

Traditional Networking Stack in Operating System

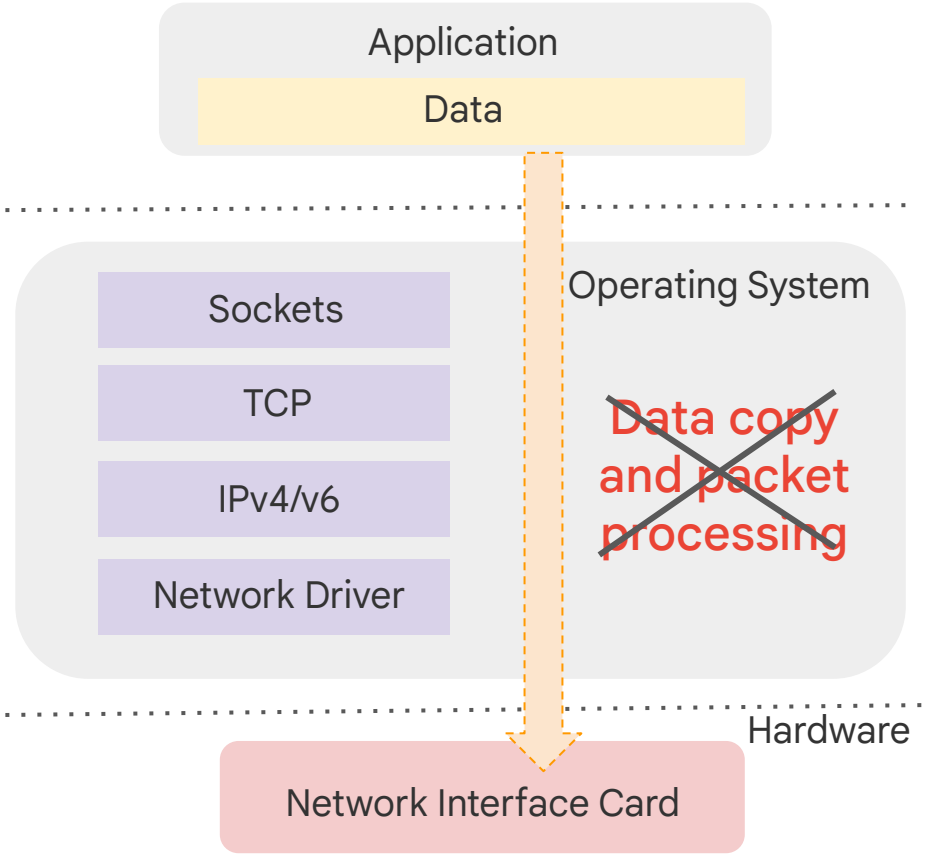


Packet Processing in host networking stack – Operating System (Linux, BSD) or user-space stacks (Snap, Data Plane Development Kit).

Problems:

- Latency variation in microseconds.
- CPU inefficiencies (expensive, CPU unavailable for applications).
- Limited packet rates or Op rates.

Remote Direct Memory Access (RDMA): Introduction



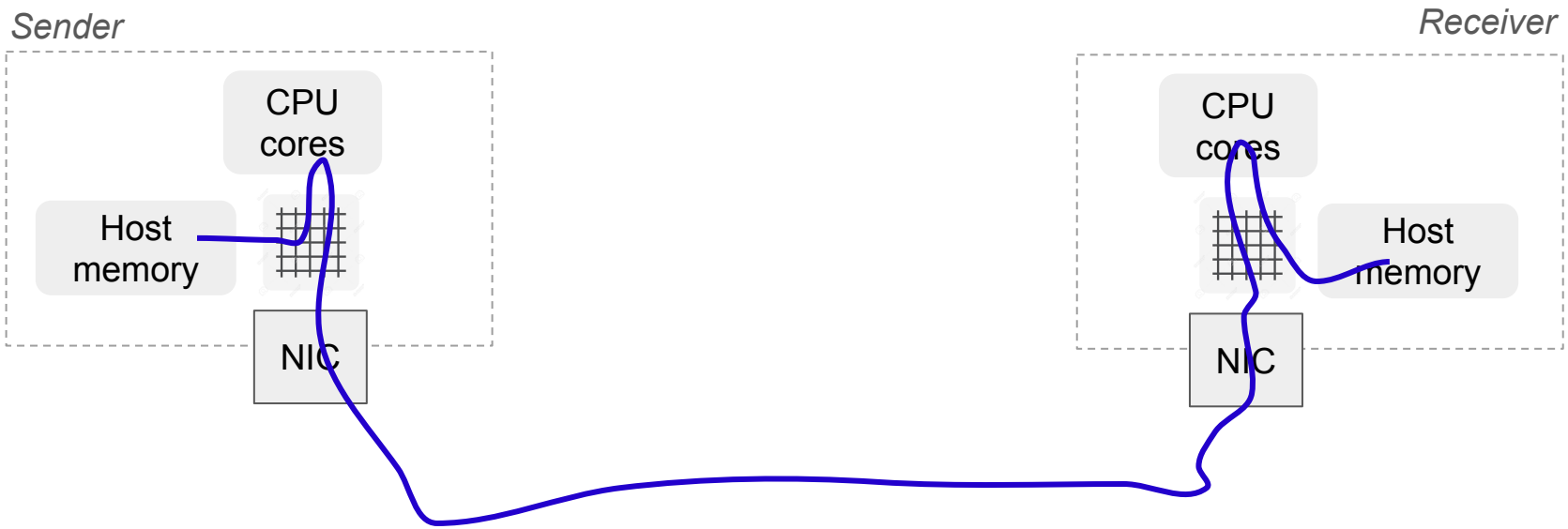
Goal: An application at server A can access data directly from the memory at server B without consuming (much) CPU time at server A or B.

RDMA is a network abstraction/implementation that achieves this goal.

- Replaces traditional TCP/IP
- Increasingly prevalent in datacenter.

Reminder: high level view of the internals of a server

Data transfers without RDMA



CPU is involved in moving data between memory and the NIC at both the sender and receiver

Data transfers with RDMA

CPU is (mostly) no longer involved!

Pros and Cons of RDMA

Pros

- High performance for data transfer between systems - low latency, high bandwidth.
- CPU efficiency: RDMA transfers data directly between senders/receivers without involving the CPU, which frees up the system resources for applications.

Cons

- More complex than traditional networking: RDMA requires specialized hardware and software.
- Limited protocols:
    - RDMA is only compatible with certain transport protocols, which limits its general use in datacenters.
    - RDMA is typically used for data transfer between systems that are in close proximity, limiting its range of application.

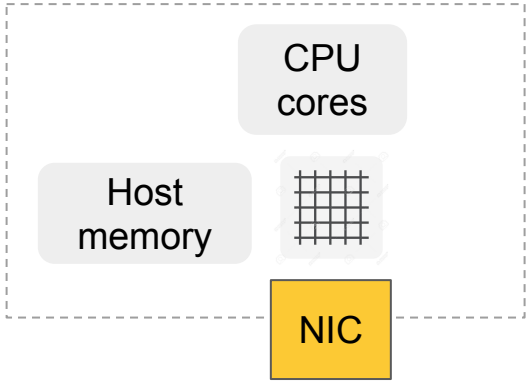
Applications of RDMA

- High Performance Computing (HPC) Applications
  - Scientific research, financial modeling, weather forecasting.
  - Key driver: all the benefits of RDMA (low latency for small messages, high packet rate, CPU efficiency)
- Low Latency applications.
  - ML inference, search queries, financial applications.
  - Key driver: low and predictable latency for small messages.
- Cloud Computing, e.g. migrating Virtual Machines from one physical server to another.
  - Key driver: CPU efficiency.
- Distributed Storage
  - key-value stores, Distributed File Systems.
  - Key driver: CPU efficiency.
- ML Training
  - Key driver: Predictable latency for high bandwidth transfers.
  - CPU efficiency is also a key consideration.

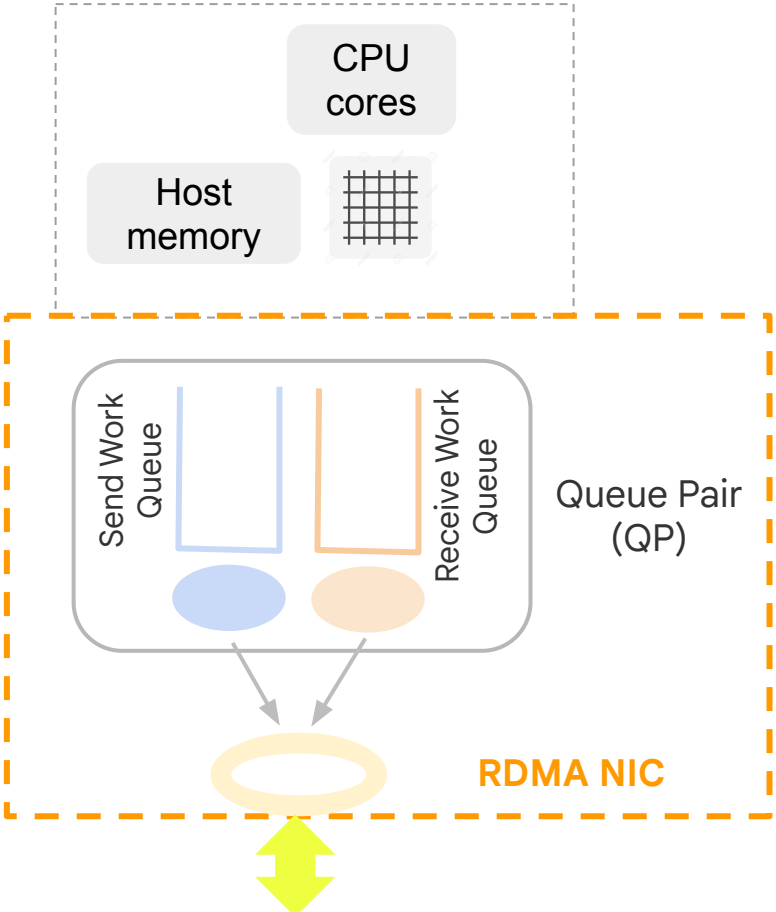
RDMA overview

- Basic idea: CPU sets up the transfer and then “gets out of the way”.
- Two high-level aspects to how this is achieved
  a. A new application interface: RDMA “queue pairs”
  b. Offloading common tasks (congestion control, reliability, ordering, etc.) to the RDMA NIC and/or network fabric.

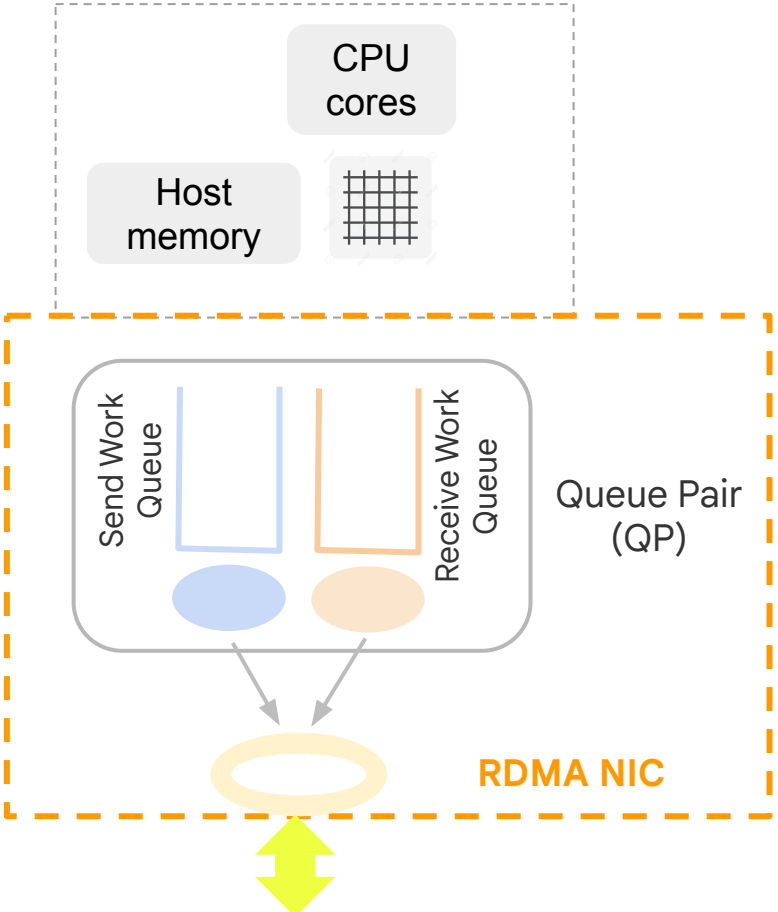
RDMA Queue Pairs



RDMA Queue Pairs



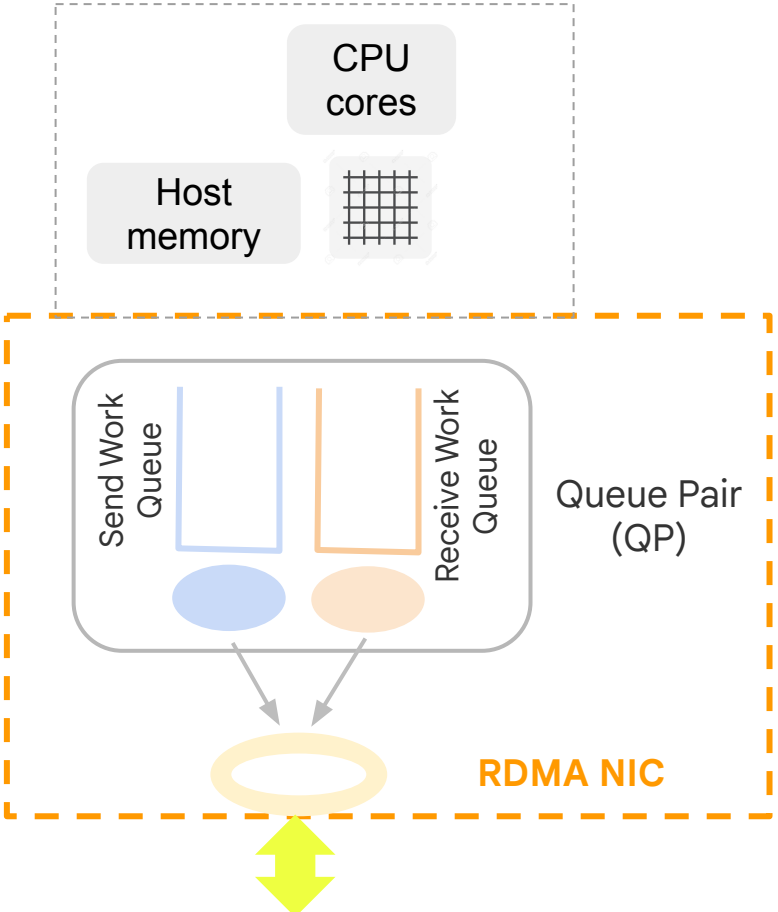
RDMA Queue Pairs



Queue Pairs (QP) are the interface between the application and RDMA NIC.

- Send Queue and Receive Queue are always created in pairs. Used by the CPU to schedule transfers.
- Different types of Queue Pairs:
  - Ordered vs. unordered.
  - Reliable vs. unreliable.
- Reliable Connected Queue Pairs
  - Closest to traditional TCP connections.
  - Connection establishment is out of band, exchanges Queue Pair Ids, etc.

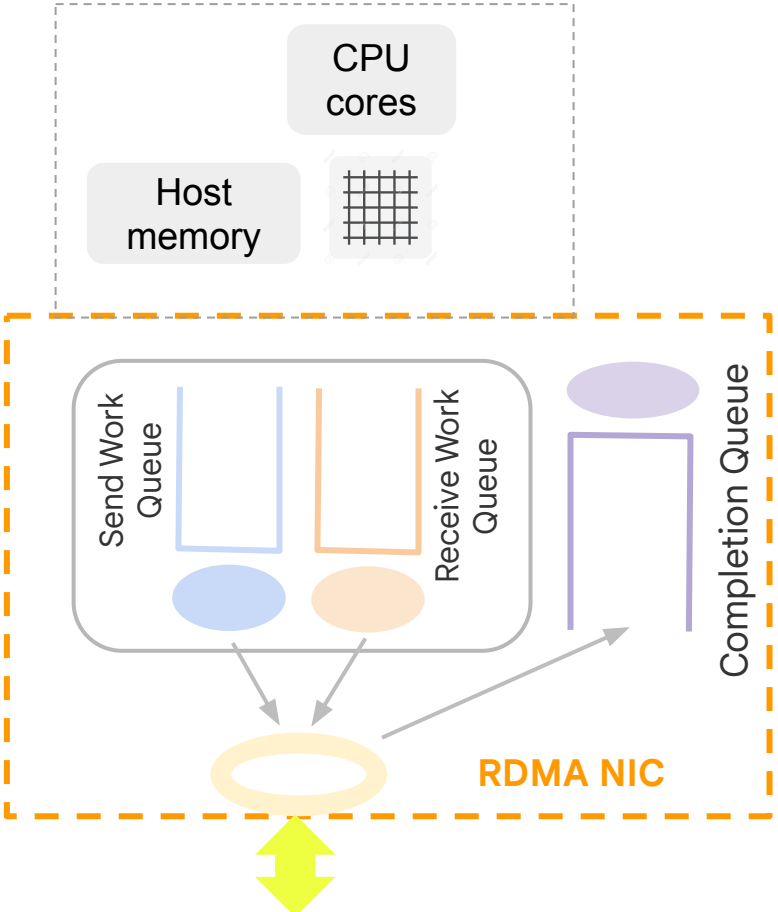
RDMA Queue Pairs



High-level idea behind Queue-Pair operation

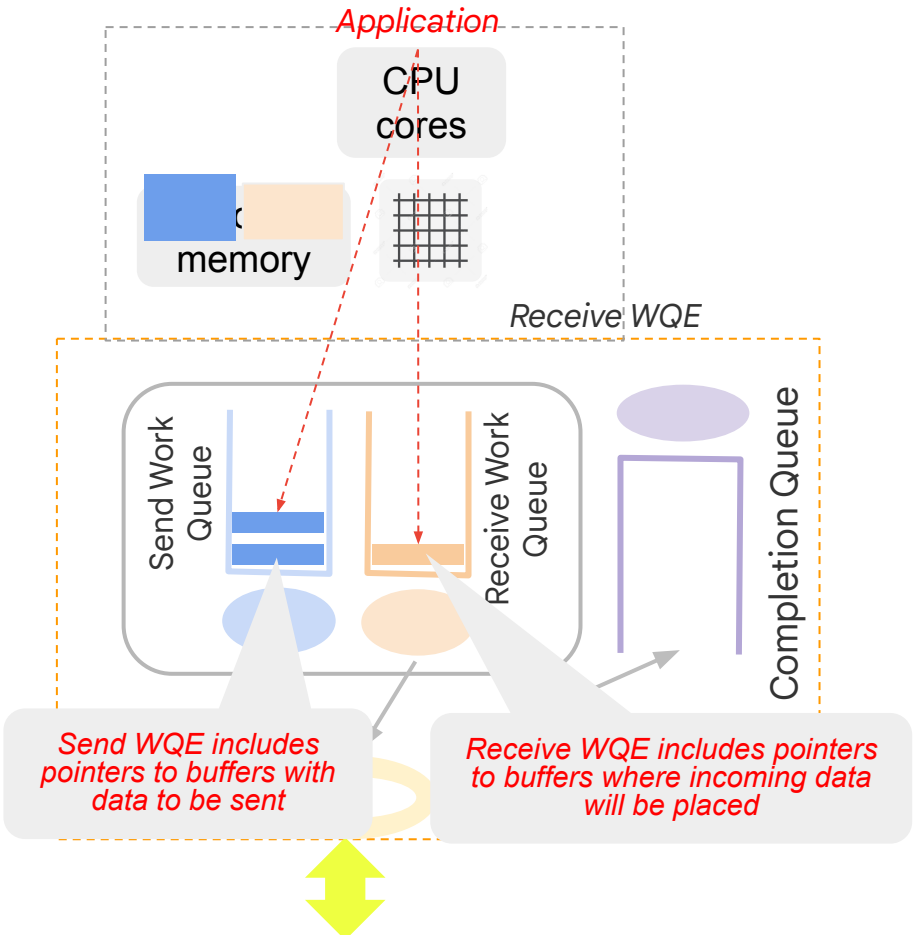
Application instructs the NIC about data it wants to send/receive by writing **“Work Queue Elements”** (WQEs) to the Send/Receive Work Queue.

(RDMA Data Transfer Mechanisms) Send, Receive and Completion Queues



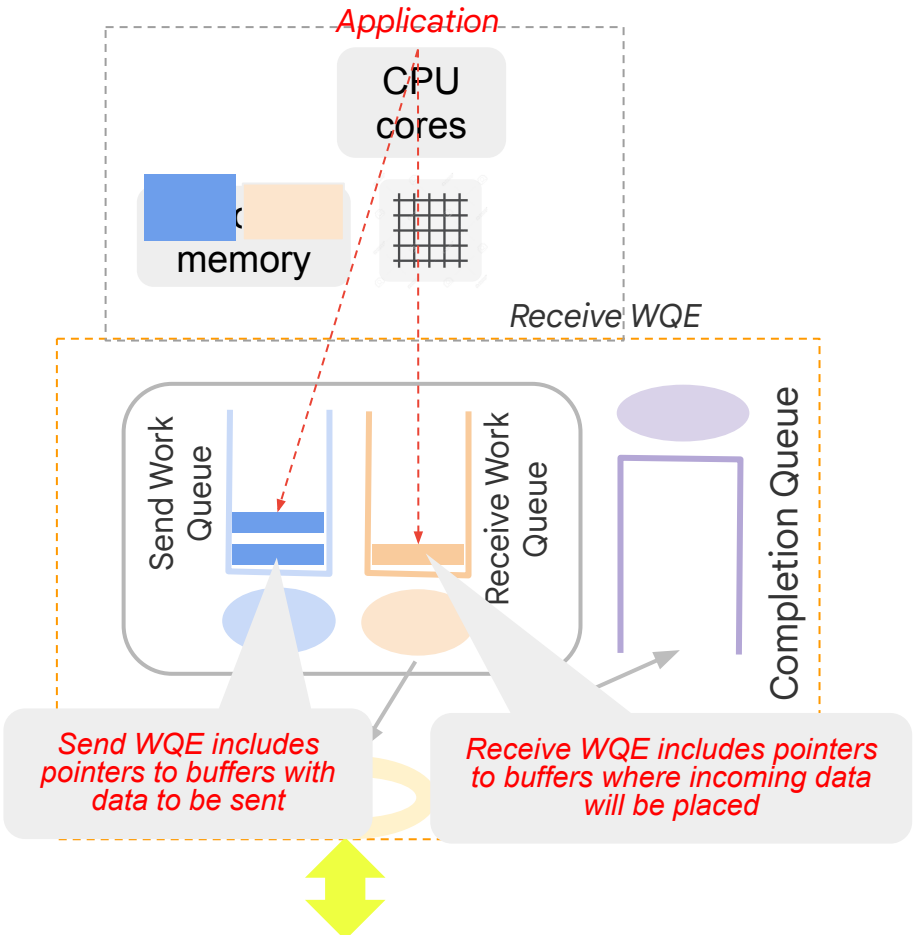
- **Send Work Queue** is responsible for managing outgoing RDMA transfers initiated by the local host.
- **Receive Work Queue** responsible for managing incoming RDMA operations from a remote host.
- **Queue Pair:** Send + Receive Queues form a complete Queue Pair (QP) for bidirectional communication.
- **Completion Queue (CQ)** Stores completion notifications for both Send and Receive Queue transfers.
- The application reads the Completion Queue to understand the status of its RDMA transfers.

(RDMA Data Transfer Mechanisms) Work Queue Elements



- Work Queue Elements (WQE): Instructions placed by application in its Send/Receive Work Queue Pairs telling NIC what buffers it wants to send or receive.
- A WQE primarily contains a pointer to a buffer.
  - WQE on send queue contains a pointer to the transfer to be sent.
  - WQE on the receive queue contains a pointer to a buffer where an incoming transfer from the wire can be placed.

(RDMA Data Transfer Mechanisms) Work Queue Elements

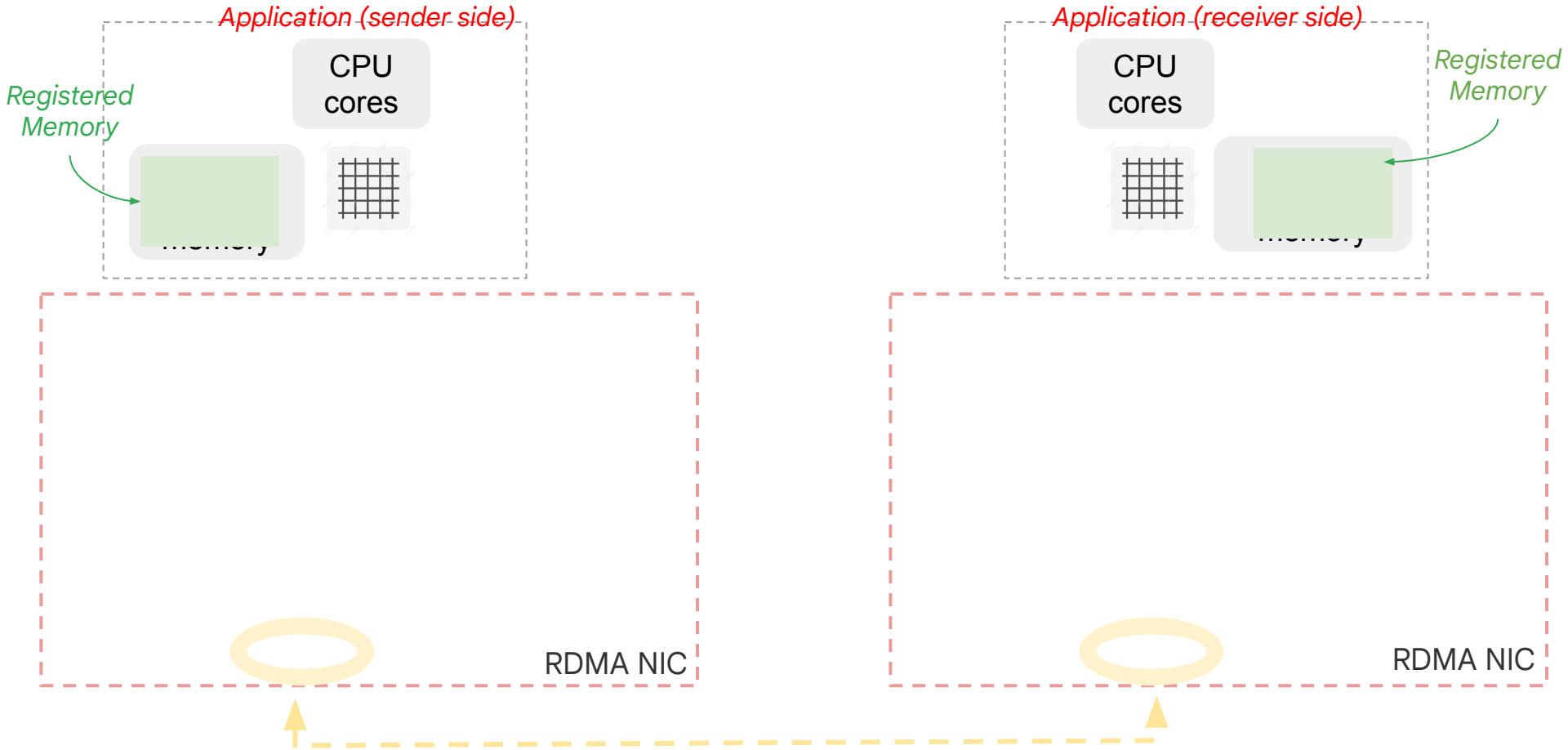


- Application can queue a number of send or receive WQEs at a time. The NIC will process these WQE in order as fast as it can.
- When the WQE is processed the data is moved. Once the transaction completes, a Completion Queue Element (CQE) is created and placed on the Completion Queue.

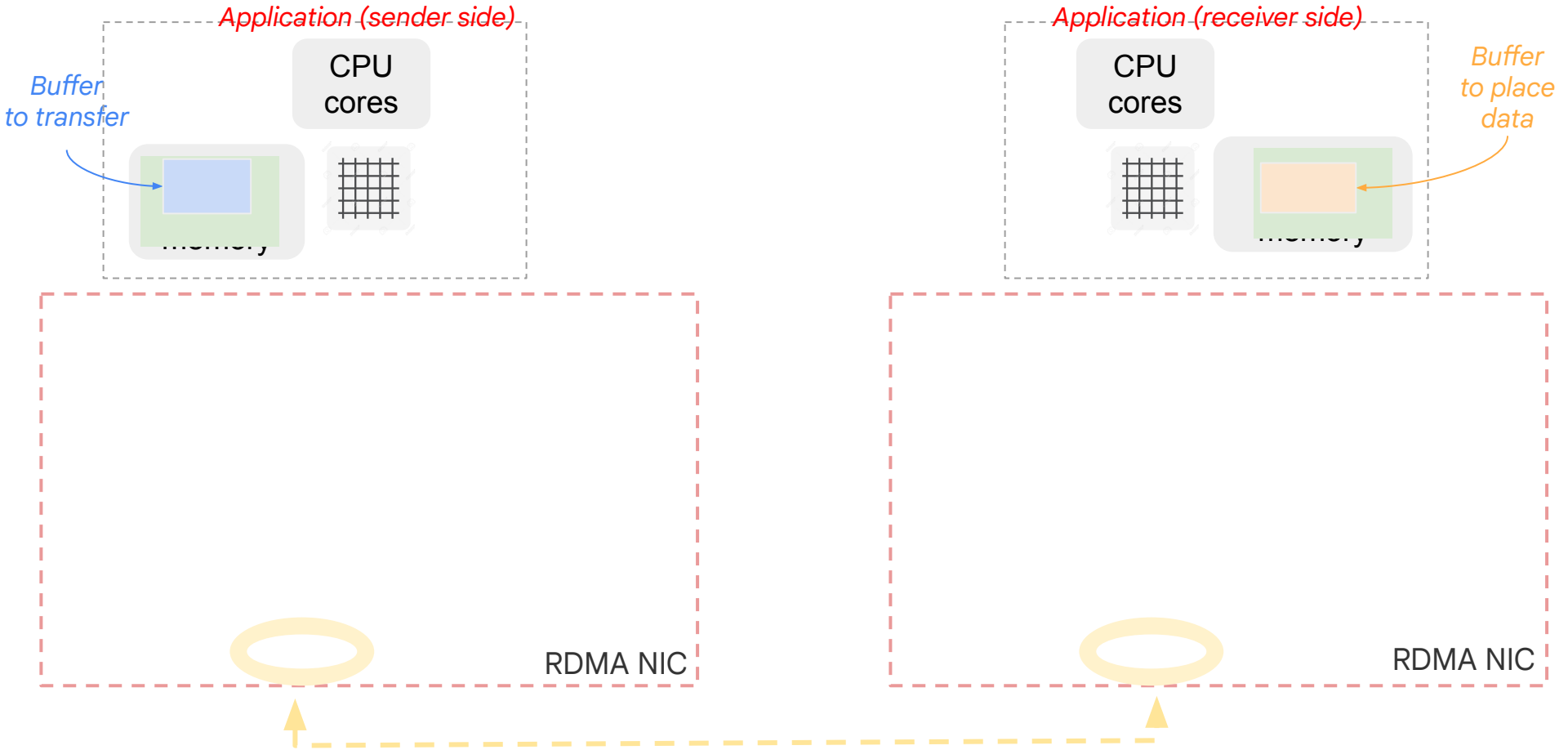
RDMA Operations

- Basic form of RDMA Operation: Send
  - Sends data to remote node.
- Other RDMA Operations:
  - RDMA Write.
  - RDMA Read.
  - RDMA Atomic.
  - RDMA WRITE with Immediate.
- Let’s step through the operation of an RDMA Send operation ...

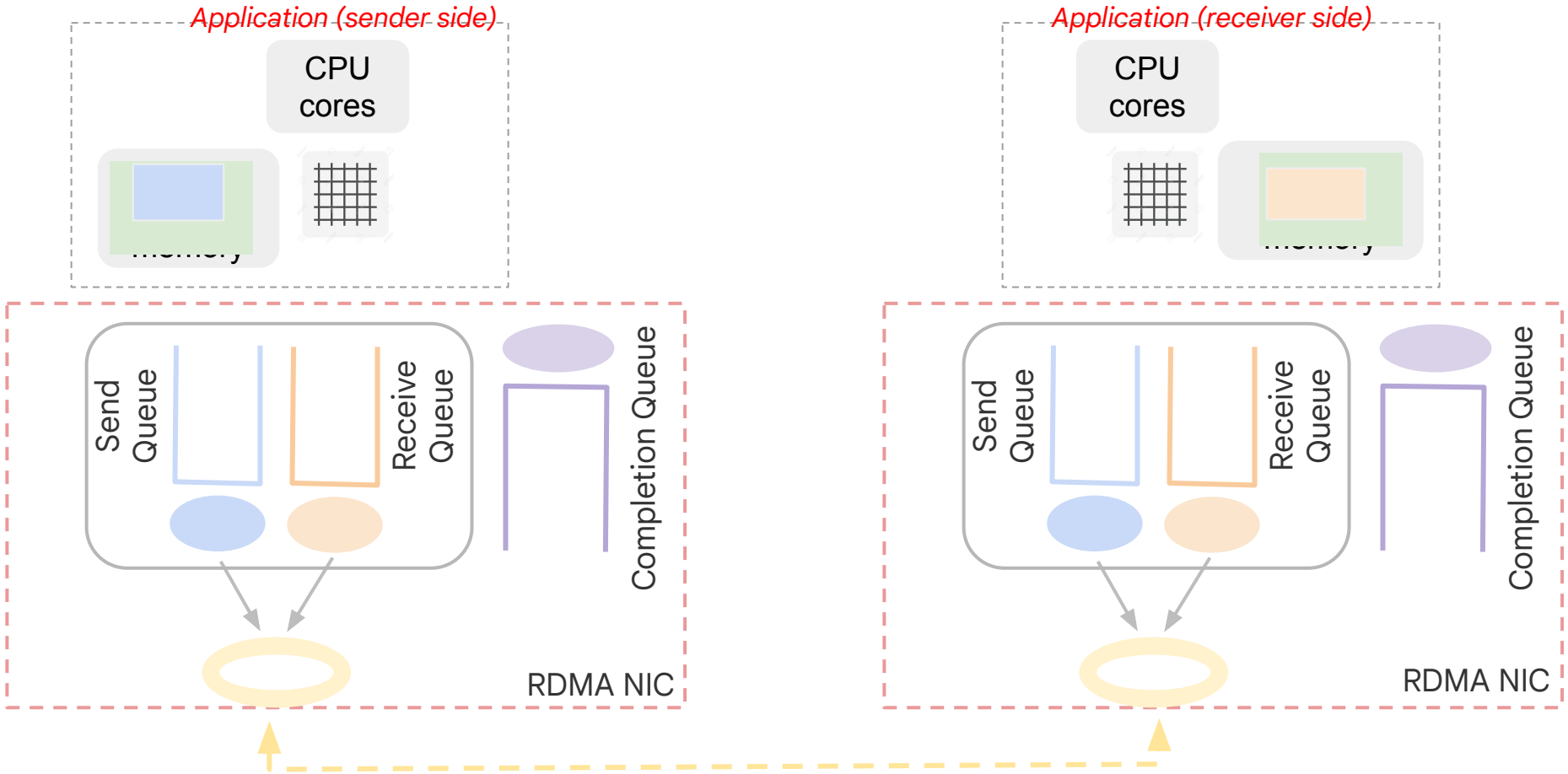
[Step 0] Application registers memory region accessible by NIC for RDMA transfers



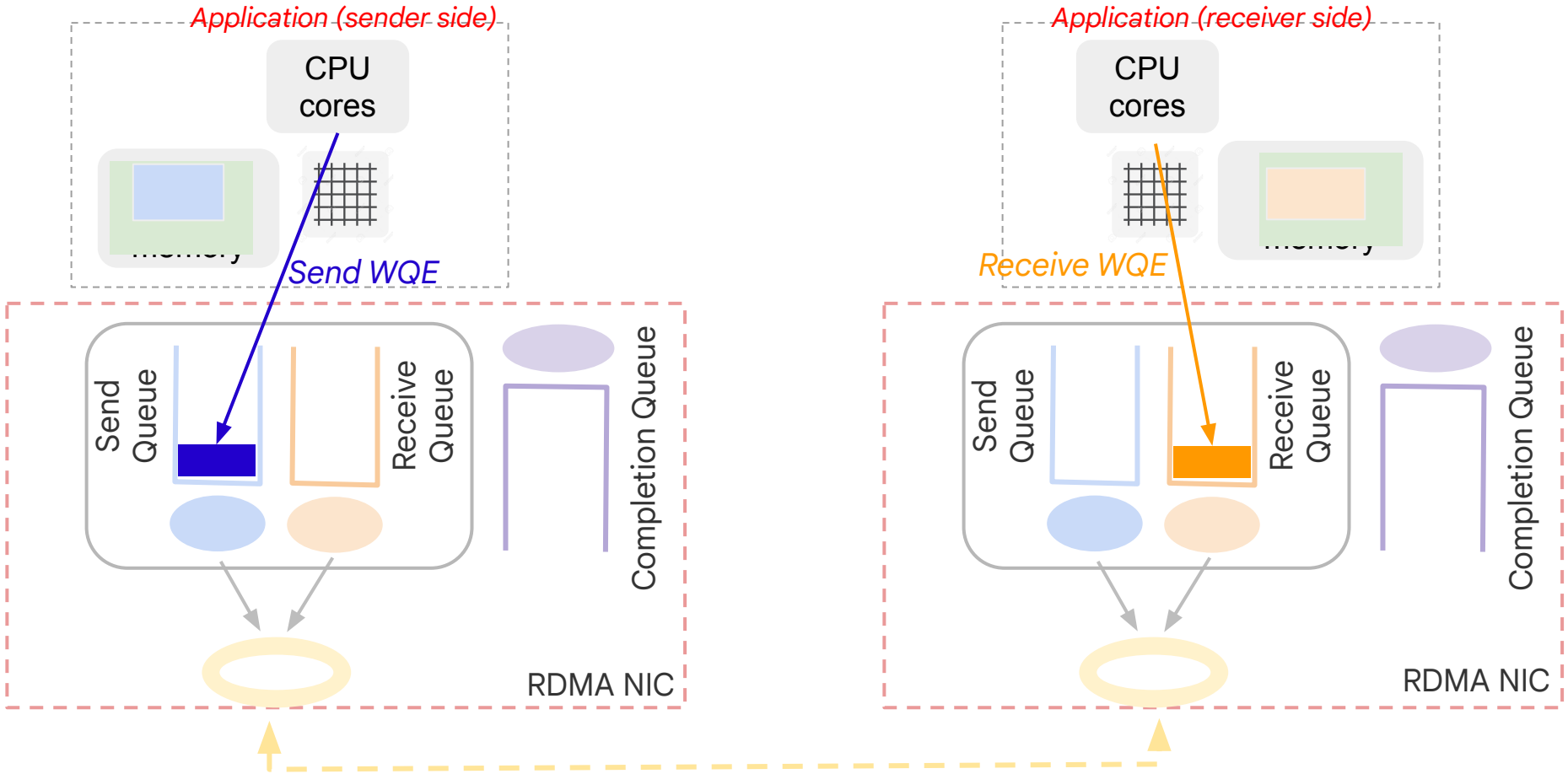
[Step 0] RDMA transfers “messages” from/to this registered memory region



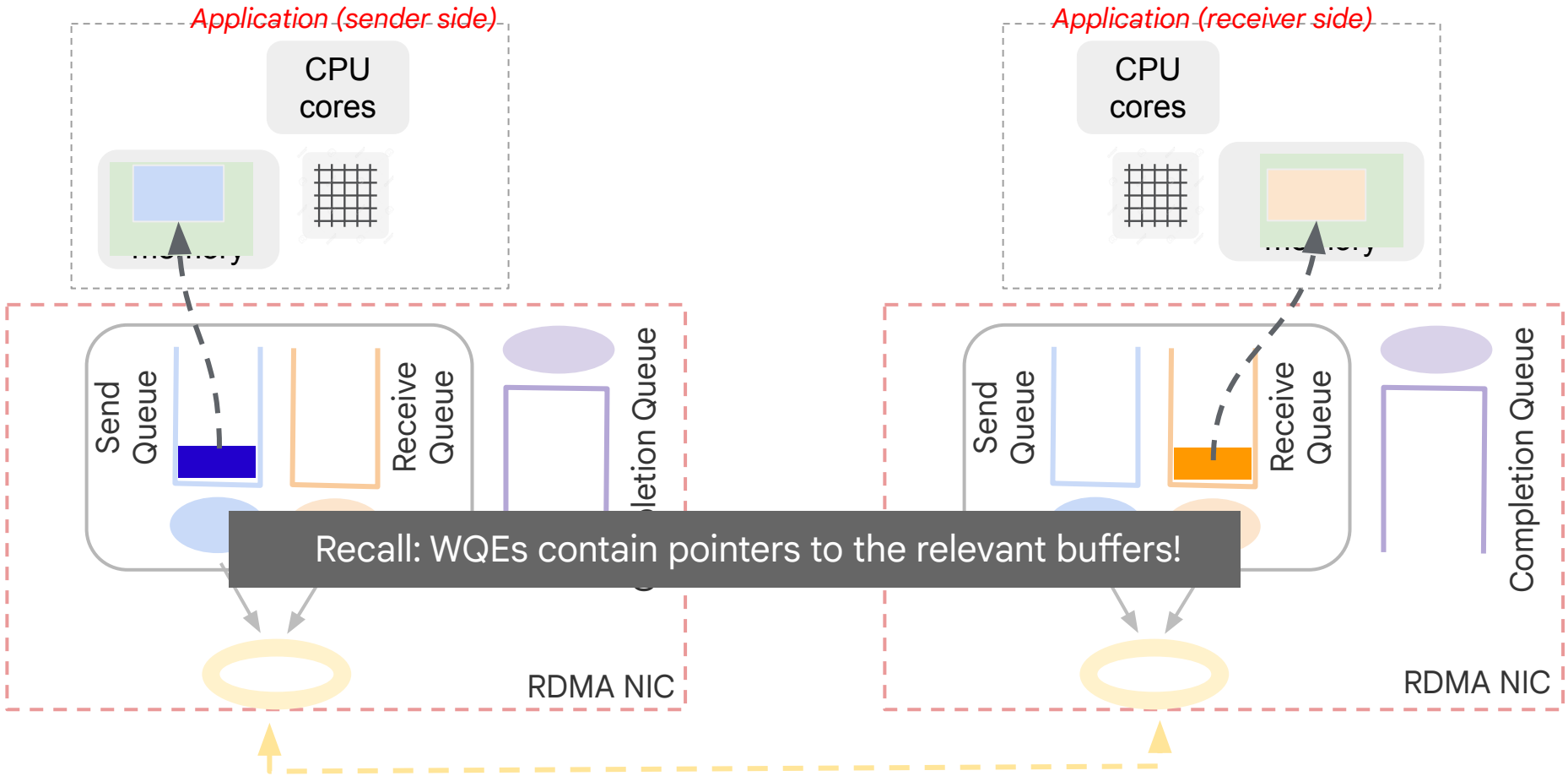
[Step 1] Create Queue Pairs, Completion Queues



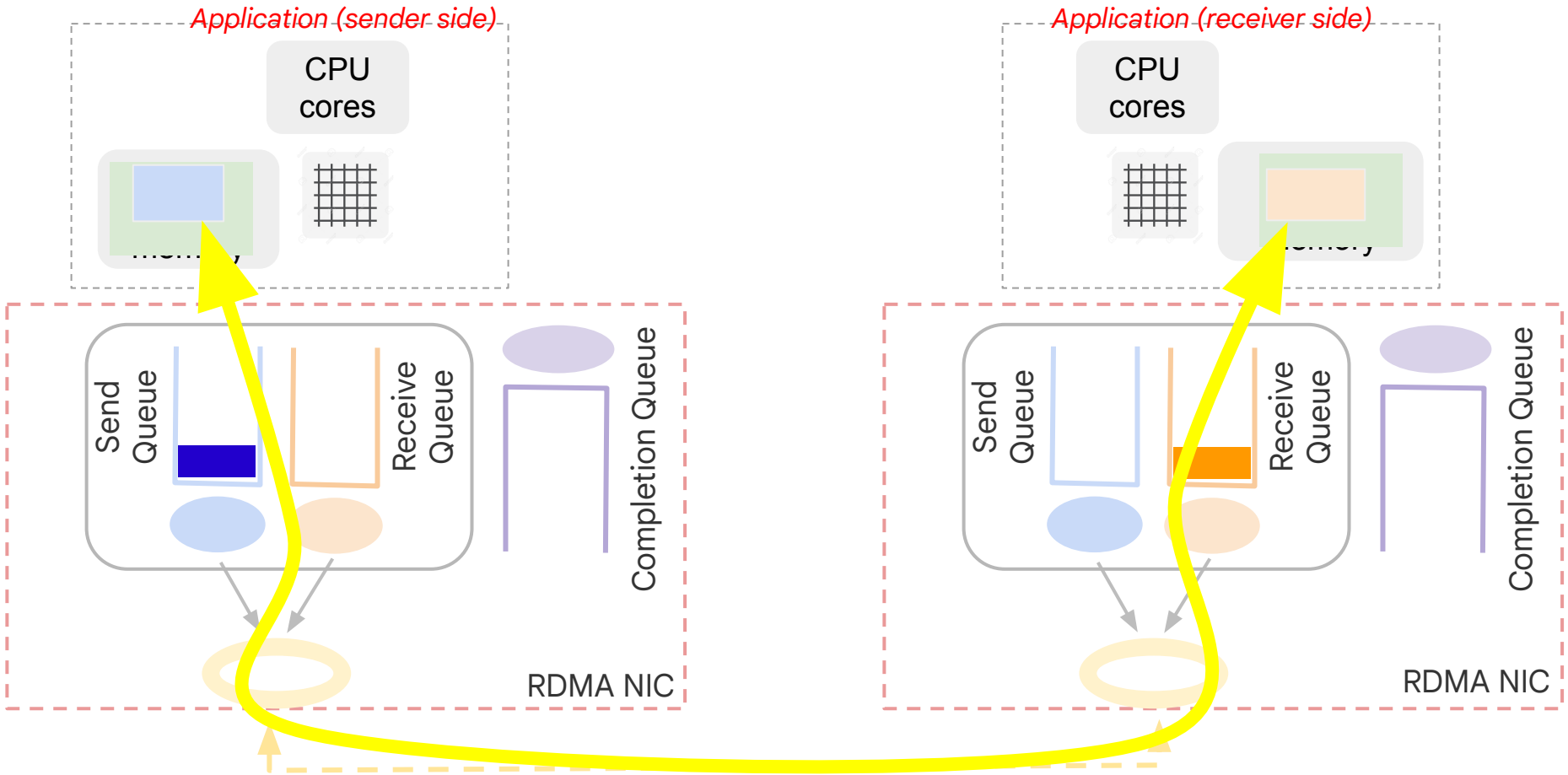
[Step 2] Application creates Work Queue Entries on Send and Receive Queues



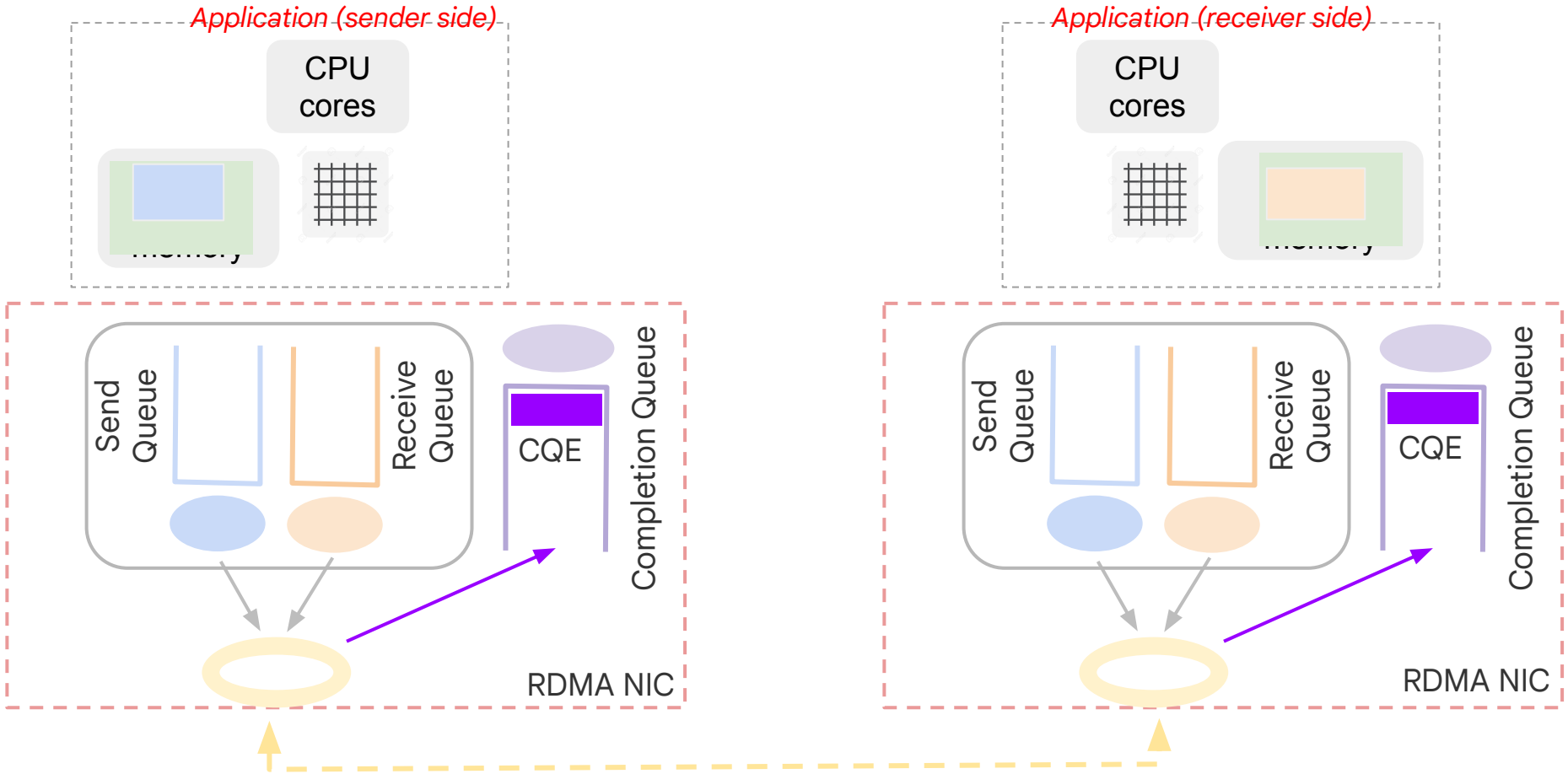
[Step 2] Application creates Work Queue Entries on Send and Receive Queues



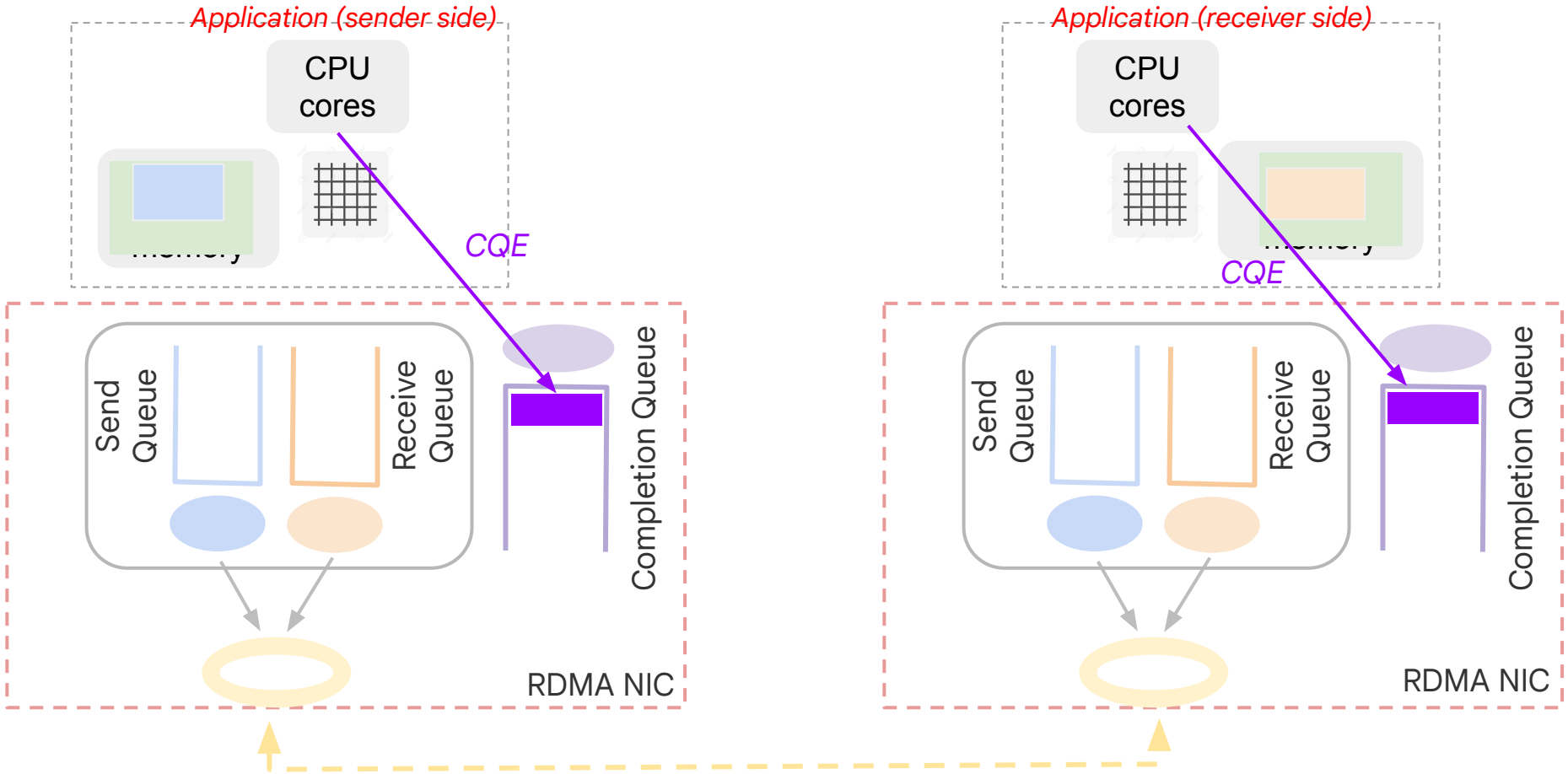
[Step 3] Memory to Memory Data Transfer (without CPU involvement!)



[Step 4] NICs Generate Completion Queue Entries (and remove relevant WQEs)



[Step 5] Application Processes Completion Queue Entries



Wait, what about losses, reordering, congestion, etc.?

Two broad options:

1. Build a datacenter network that is reliable, preserves ordering, etc. [Nvidia's Infiniband]
2. Implement Reliability, Ordering, Congestion Control in NIC under queue-pair abstraction [Google].

In both cases, applications and OS "see" the abstraction of reliable and in-order message delivery.