# Host Networking

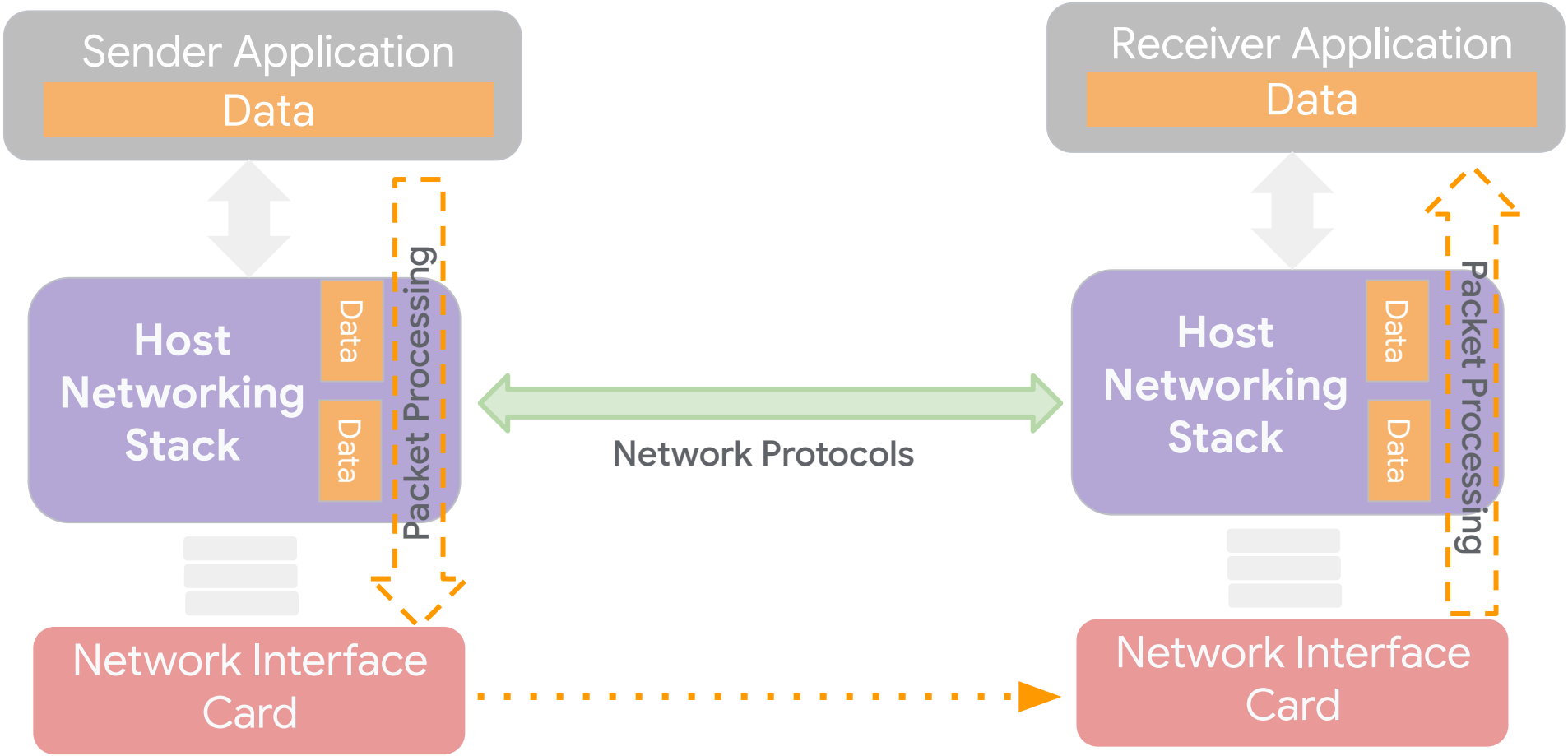
Sylvia Ratnasamy
CS168 Fall 2024

Based on slides by Nandita Dukkipati (guest lecture at UC Berkeley, April 2024)

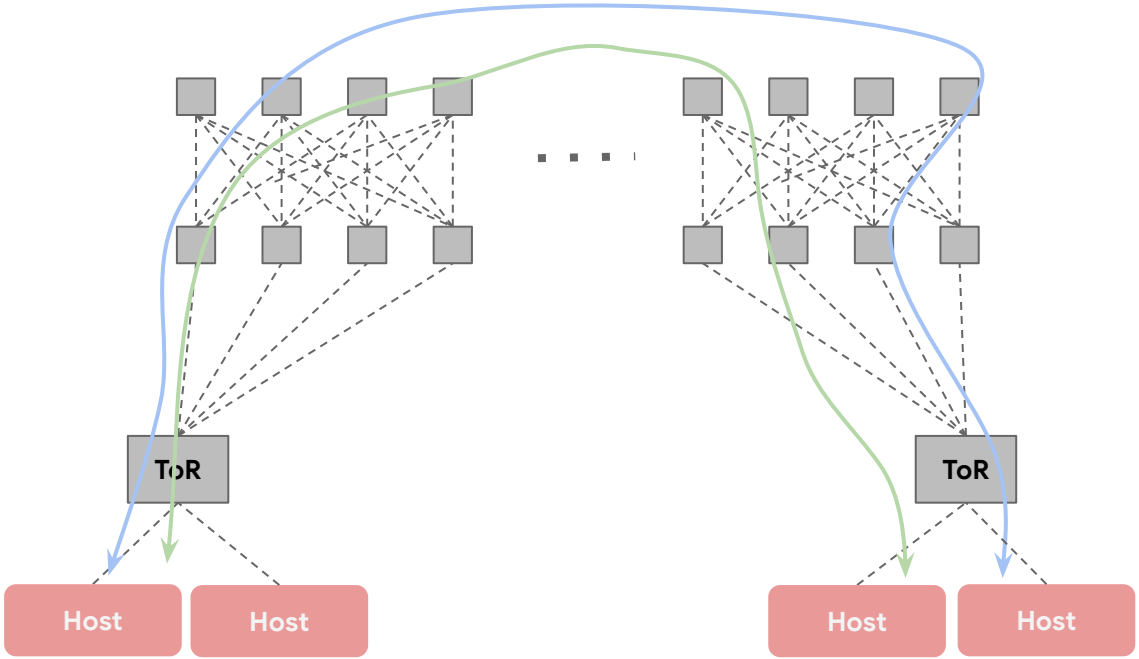
# Lecture Topics

- What is host networking and why it matters in the datacenter
- The role of Network Interface Cards (NICs)
- Interfacing with applications using Remote Direct Memory Access (RDMA)

# What constitutes Host Networking?



# End-to-End performance of applications *within* the datacenter




**Driver 1:**
Datacenter applications demand high bandwidth and low latency

**Driver 2:**
Server CPUs dominate datacenter costs
→ *efficient* use of CPUs is important!

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

# This lecture

- More about system implementation than network architecture or algorithms
- Opportunity to practice reasoning about system *performance*

# This lecture

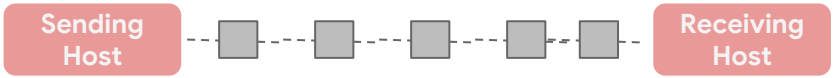
- More about system implementation than network architecture or algorithms
- Opportunity to practice reasoning about system *performance*
- Themes:
  - Hardware that is “specialized” for a given function is generally cheaper, faster, and lower power than using general-purpose processors for that same function
    - Why we use GPUs/TPUs for AI applications instead of x86 CPUs
    - Why we use specialized packet processing hardware in routers
  - Operating systems are large and complex codebases ⟶ introduce unwanted perf overheads
    - E.g., due to scheduling, interrupt handling, *etc.*

# Why host networking matters in the datacenter: a closer look (1)

Takeaway#1: host networking is a non-trivial contributor to end-to-end latency

**Assumptions**

- 1000 byte packet
- 5 switch hops between sending and receiving machines
- Average queue length at a switch = 10 packets
- Machines are 100 meters apart




**Circa 2011 [ref]**

**Technology specs**

- CPU speed: ~3GHz
- Link speeds: 10Gbps
- Propagation speed = 5 nanosecs/meter (speed of light in fiber)

**Per-component delays**

- Per switch: ~10 microsecs
  - Transmission: 0.8 microsecs (8000 bits @10Gbps)
  - Queueing: 10 x 0.8 = 8 microsecs
  - Switching: 1 microsecs
- Per-host: 10 microsecs

**Round-trip delay**

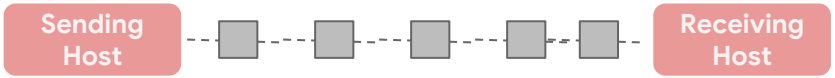
- Switch time: 10 hops x 10 /hop = **100 microsecs** (5 hops each way)
- Per-host: 10 x 4 = **40 microsecs** (4 traversals through OS)
- Propagation delay = **1 microsec** (5 nanosecs/meter x 100 meter x 2)

# Why host networking matters in the datacenter: a closer look (2)

**Takeaway#2: technology trends make host networking overheads a growing problem**

**Assumptions**

- 1000 byte packet
- 5 switch hops between sending and receiving machines
- Average queue length at a switch = 10 packets
- Machines are 100 meters apart




## Circa 2011 [ref]

**Technology specs**

- CPU speed: ~3GHz
- Link speeds: 10Gbps
- Propagation speed = 5 nanosecs/meter (speed of light in fiber)

**Per-component delays**

- Per switch: ~10 microsecs
  - Transmission: 0.8 microsecs (8000 bits @10Gbps)
  - Queueing: 10 x 0.8 = 8 microsecs
  - Switching: 1 microsecs
- Per-host: 10 microsecs

**Round-trip delay**

- Switch time: 10 hops x 10 /hop = **100 microsecs** (5 hops each way)
- Per-host: 10 x 4 = **40 microsecs** (4 traversals through OS)
- Propagation delay = **1 microsec** (5 nanosecs/meter x 100 meter x 2)

## Projected 2026 [ref]

**Technology specs**

- CPU speed: ~3GHz (Moore's law ended!)
- Link speeds: 1000 Gbps
- Propagation speed = 5 nanosecs/meter

**Per-component delays**

- Per switch: ~1 microsecs
  - Transmission: 0.008 microsecs
  - Queueing: 10 x 0.008 = 0.08 microsecs
  - Switching: 1 microsecs
- Per-host: 10 microsecs

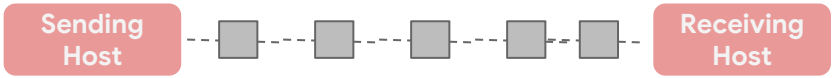
**Round-trip delay**

- Switch time: 10 x 1 = **10 microsecs**
- Per-host: 10 x 4 = **40 microsecs**
- Propagation delay = 1 microsec

# Why host networking matters in the datacenter: a closer look (3)

**Assumptions**

- 1000 byte packet
- 5 switch hops between sending and receiving machines
- Average queue length at a switch = 10 packets
- Machines are 100 meters apart




## Circa 2011 [ref]

**Technology specs**

- CPU speed: ~3GHz
- Link speeds: 10Gbps
- Propagation speed = 5 nanosecs/meter (speed of light in fiber)

**CPU overhead to drive a 10Gbps link**

- 1 3GHz CPU core can achieve ~20Gbps throughput
- Hence, need ½ a CPU core to drive a 10Gbps link

## Projected 2026 [ref]

**Technology specs**

- CPU speed: ~3GHz (Moore's law ended!)
- Link speeds: 1000 Gbps
- Propagation speed = 5 nanosecs/meter

**CPU overhead to drive a 1000 Gbps link**

- 1 3GHz CPU core can achieve ~20Gbps throughput
- *Would need 50 CPU cores to drive a 1000 Gbps link!*

Takeaway#3: traditional host networking stacks consume too many CPU resources!

# Recap: Why host networking needs (more) attention in the datacenter

**Drivers**

- Datacenter applications want low latency and high throughput
- Server CPU resources dominate datacenter costs

**Tech trends**

- Datacenter networks enjoy low propagation delay and high bandwidth
- Link bandwidth continues to increase with every generation (~10x every 7-8 years)
- But CPUs are not getting faster (slowing / ending of Moore's law)

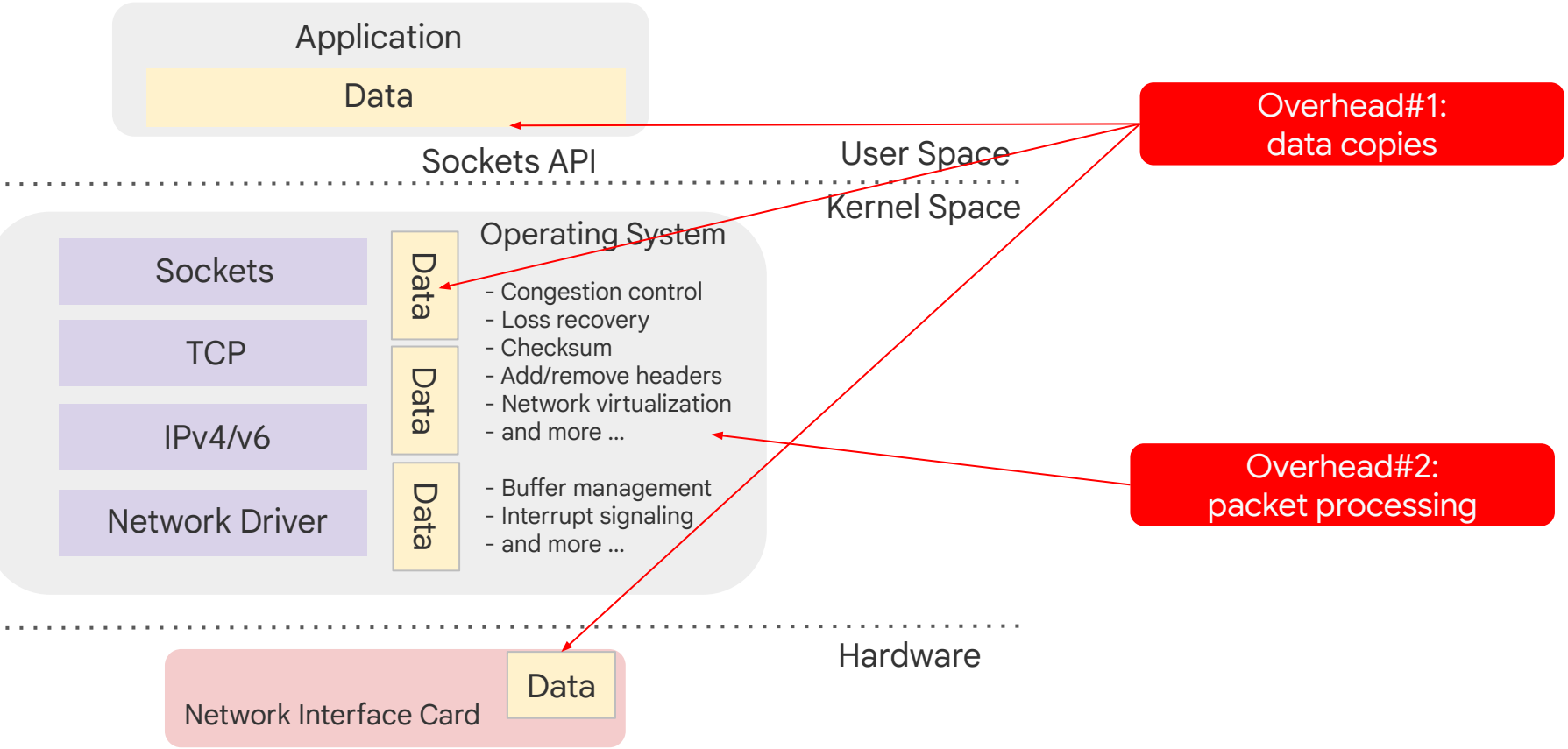
**Implications**

- Host networking is a significant contributor to end-to-end latency
  - *Tech trend*: latency due to switches is decreasing while the latency due to hosts is not
- Host networking consumes a significant amount of server CPU resources
  - *Tech trend*: as link bandwidth increases, so do the CPU resources needed to drive the link

Rest of this lecture: a closer look at host networking and what we can do to address these problems

# Questions?

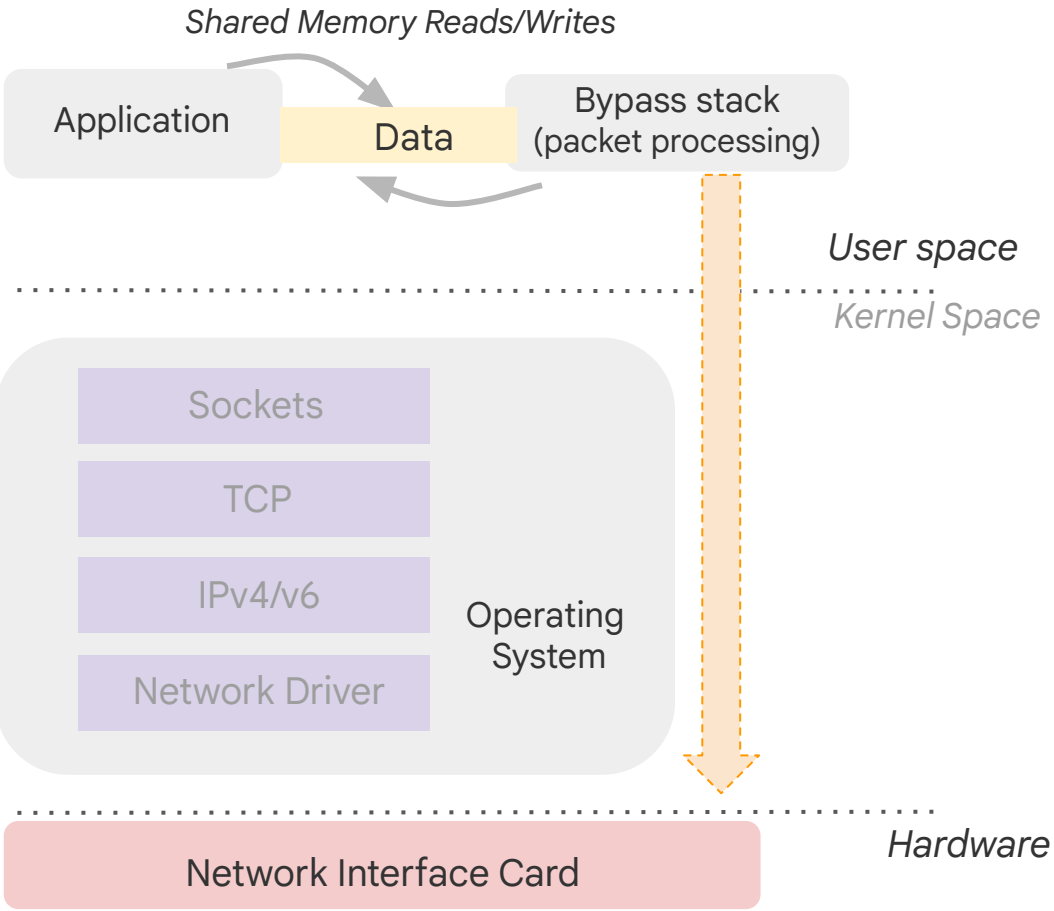
# Traditional Networking Stack in Operating System



# Recap: traditional network stacks in the OS

- Problem#1: high performance overheads due to:
  - Lots of data copies (between user-space and kernel, kernel and NIC)
  - Per-packet processing in software (checksums, CC, loss recovery, encryption, *etc.*)
- These introduce latency *and* consume CPU cycles
- Problem#2: kernel development and deployment is painful
  - Kernel code is complex, with a distributed developer ecosystem
  - Upgrades require machine reboots
- Two common optimizations: “bypass” stacks and NIC “offload”

# 1) Operating System “Bypass” Stacks




Key idea: implement the host networking stack in user space and avoid using OS networking components (“bypass”)

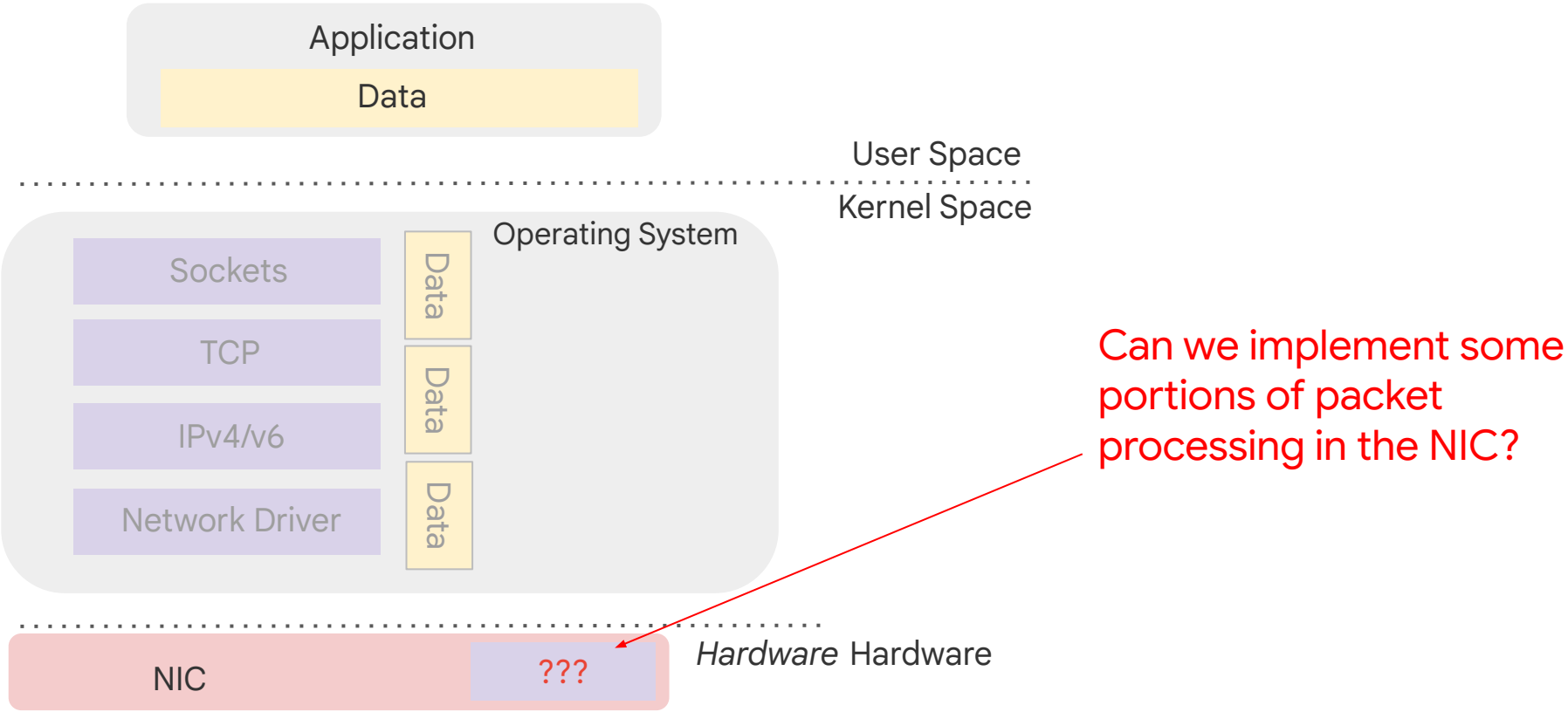
# 1) Operating System “Bypass” Stacks

- Key idea: implement the network stack as a user space process
  - Packet processing operations implemented in user space
  - With shared memory between the application and network stack
  - E.g., Intel DPDK, AWS EFA, Google Snap

- Benefits
  - Avoids copying data between the application and network stack
  - Larger developer pool and simpler deployment process (does not require machine reboots)

- But only a partial solution: bypass stack still consumes CPU cycles for packet processing

# 2) “Offload” the Network Stack to the Network Interface Card



# 2) “Offload” the Network Stack to the Network Interface Card

- Key idea: move packet processing functions from the host’s CPU to the NIC
  - Idea can be applied to kernel- or user-space host networking stacks
- Benefits
  - Frees up host CPU cycles that are otherwise consumed for host networking
  - NIC can use hardware specialized for networking → cheaper, faster, lower power than host CPUs
- These benefits fueled the development of “smart NIC” technology
  - E.g., Nvidia BlueField, Marvell Octeon, AWS Nitro, AMD Pensando
- A few different options for how much of the networking stack we offload (coming up)
  - Tradeoff: performance and efficiency benefits *vs.* NIC complexity and specialization

# Lecture Outline

What is Host Networking and Why it Matters

- Learnt before: network stack in the OS implements reliability, congestion control, flow control, *etc.*
- Problem in datacenters: extreme perf. requirements; valuable CPU cores; kernel development hard.
- Host networking in datacenters is heavily optimized for application performance and CPU efficiency.
- Optimization opportunities: OS bypass, and offloads to the NIC

The Role of Network Interface Cards (NICs)

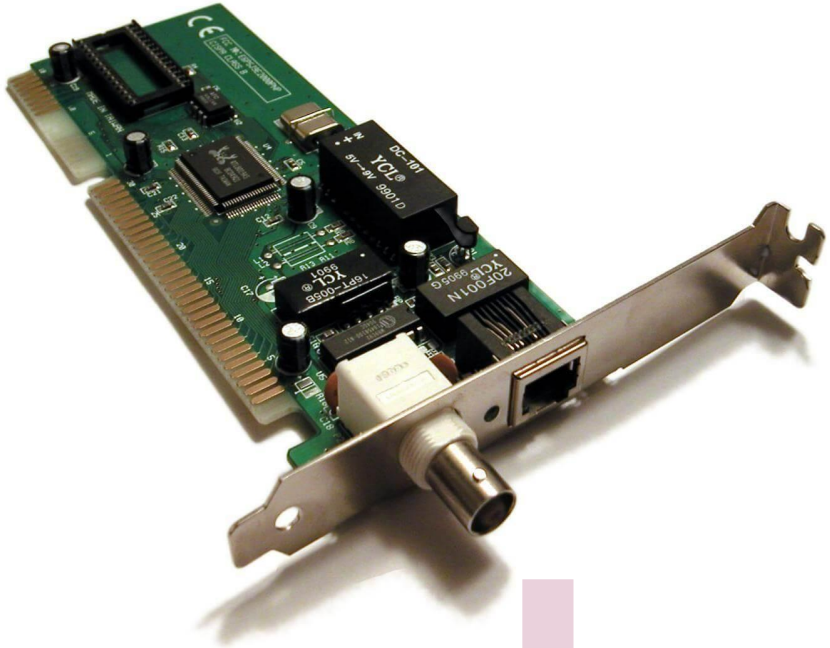
Remote Direct Memory Access

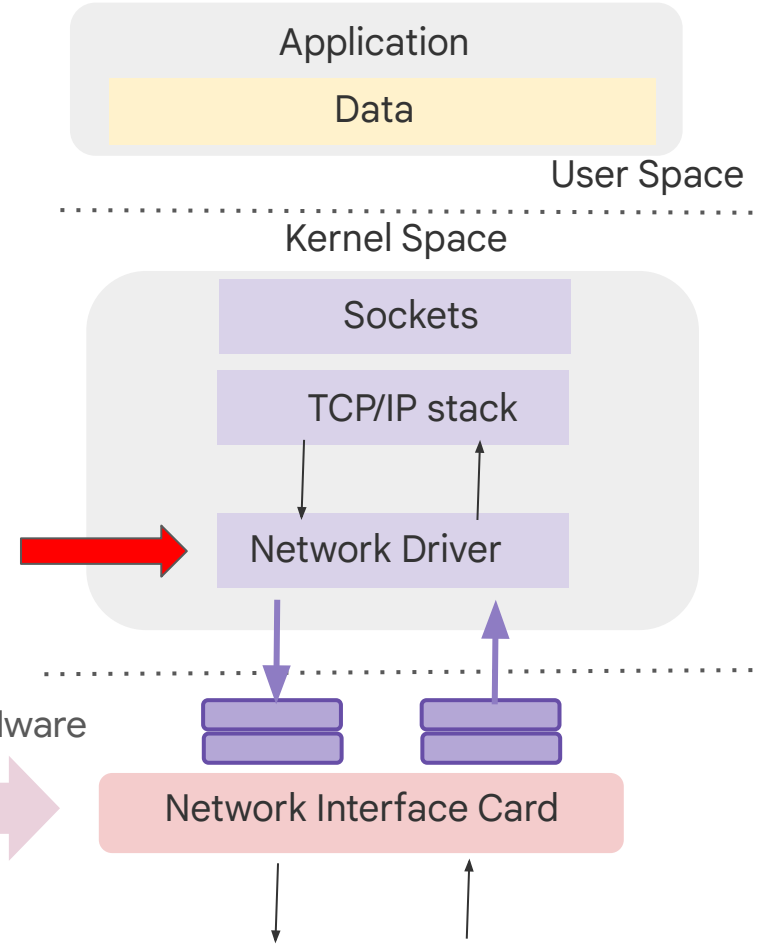
# Questions?

# Lecture Topics

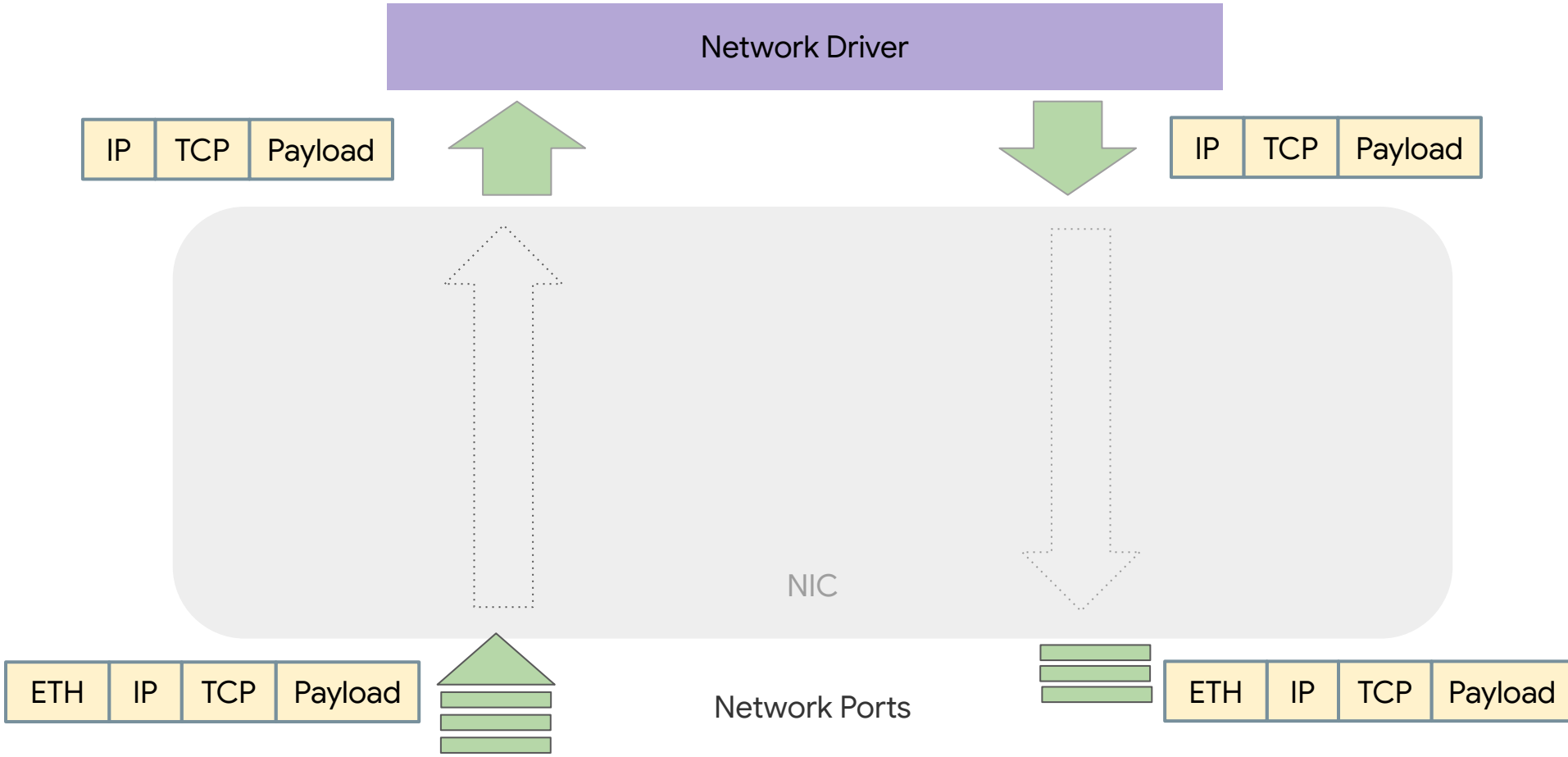
- What is Host Networking and Why it Matters
- The Role of Network Interface Cards (NICs) (Ref: IETF Talk)
- Interfacing with Applications using Remote Direct Memory Access (RDMA).

# Fundamentals of Network Interface Card (NIC)

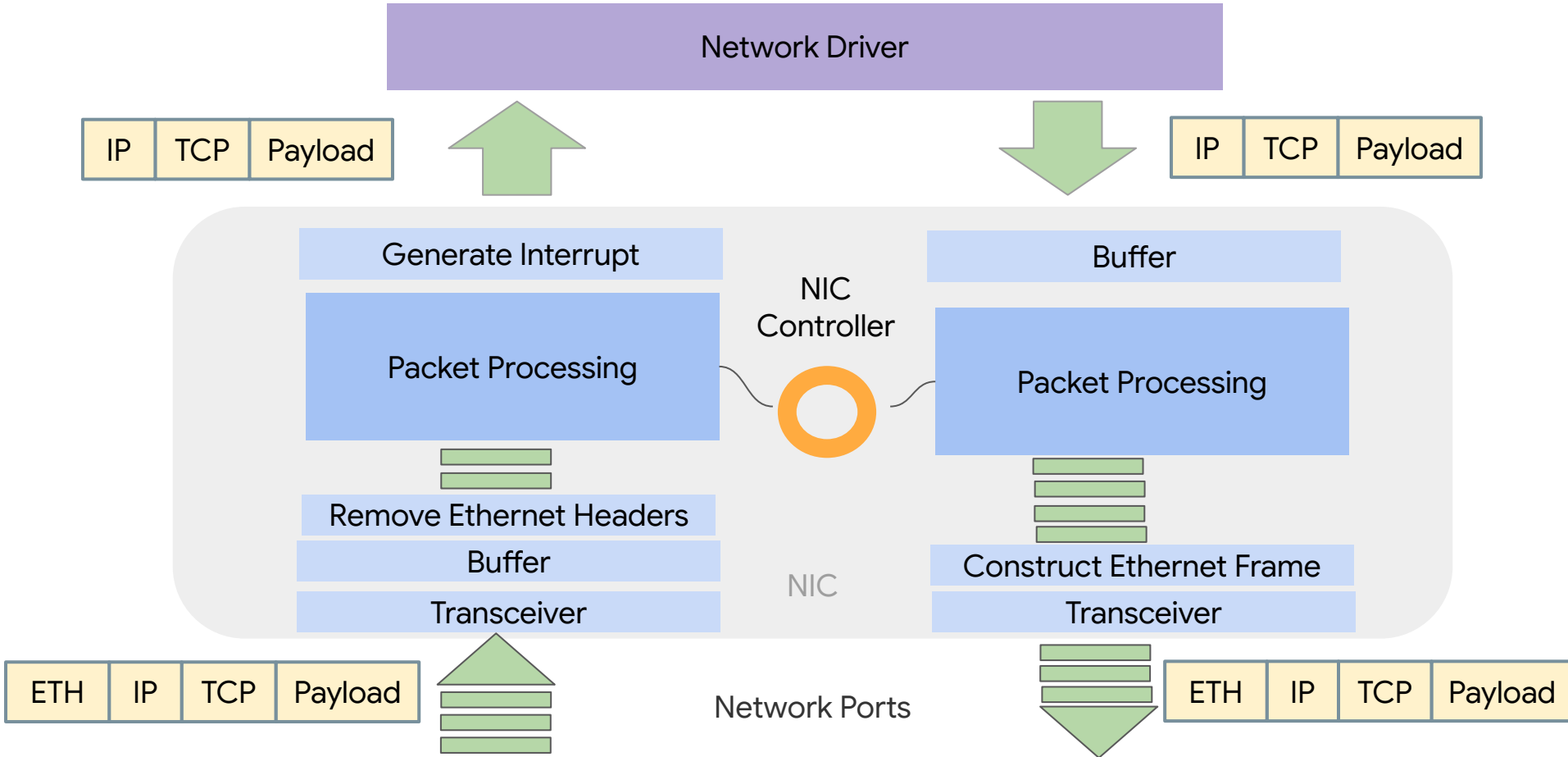




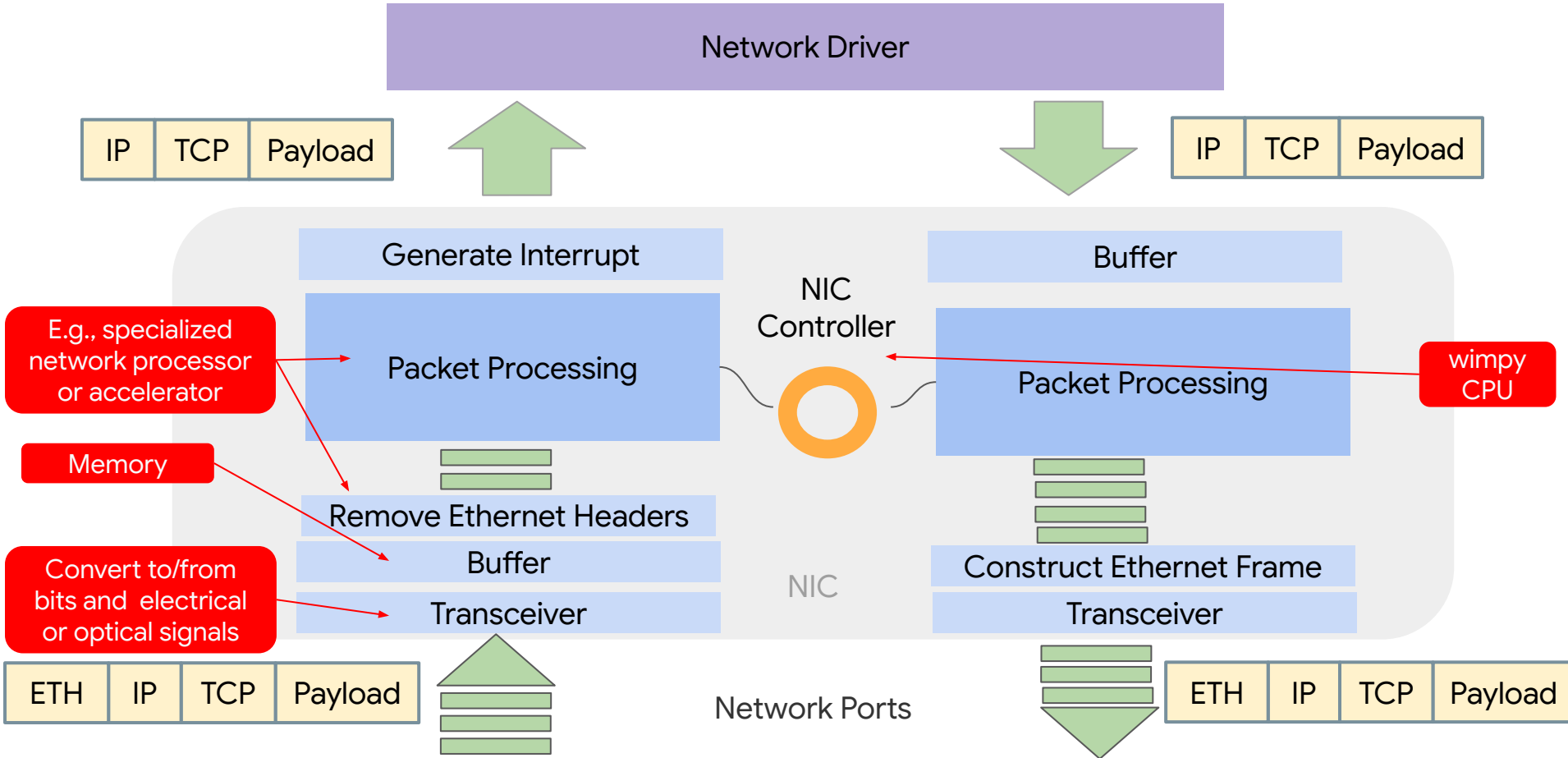
# Life of Packet in the NIC



# Life of Packet in the NIC



# Life of Packet in the NIC



# Why Offload?

- Free up host CPU cycles for applications
  - Network processing and applications no longer contend for CPU
- Efficiency
  - Specialized hardware can be more efficient in cost and power (*vs.* general-purpose CPU)
- Performance
  - Scaling throughput
  - Predictable low latency

# Spectrum of Offloads to the NIC

- Epoch 1 - Basic support
  - Simple stateless functions
- Epoch 2 - Accelerating the dataplane
  - More complex stateful offloads
- Epoch 3 - Protocol Offloads
  - Offload the entire protocol state machine (e.g., TCP, RDMA)

# Epoch 1: Simple, Stateless Offloads of Network Processing

Typically, operations that can be applied on individual packets without per-flow state or application-level information

Examples:

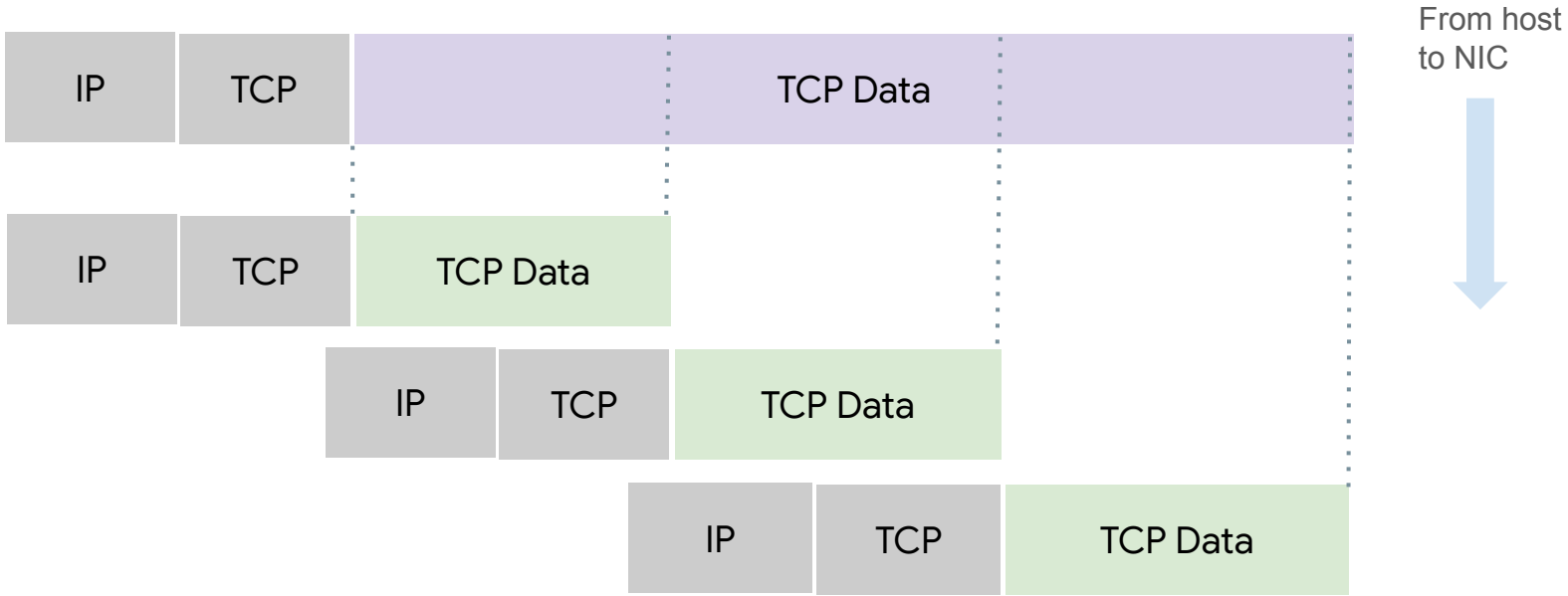
- Checksum Offload: compute the checksum on transmit; verify the checksum on receive [Example]
- Segmentation Offload: split large packets into MTU-sized ones on transmit; coalesce small packets into a large packet on receive

# Segmentation Offload

- Recall: TCP sends packets with “max segment size” (MSS) bytes
  - MSS picked based on link MTU (Maximum Transmission Unit)
- With Segmentation Offload: Host CPU sends/receives a much larger packet (>> MTU) and the NIC converts it into multiple smaller (~MTU sized) ones
  - E.g., host sends the NIC a 128KB packet and the NIC segments them into 1KB packets

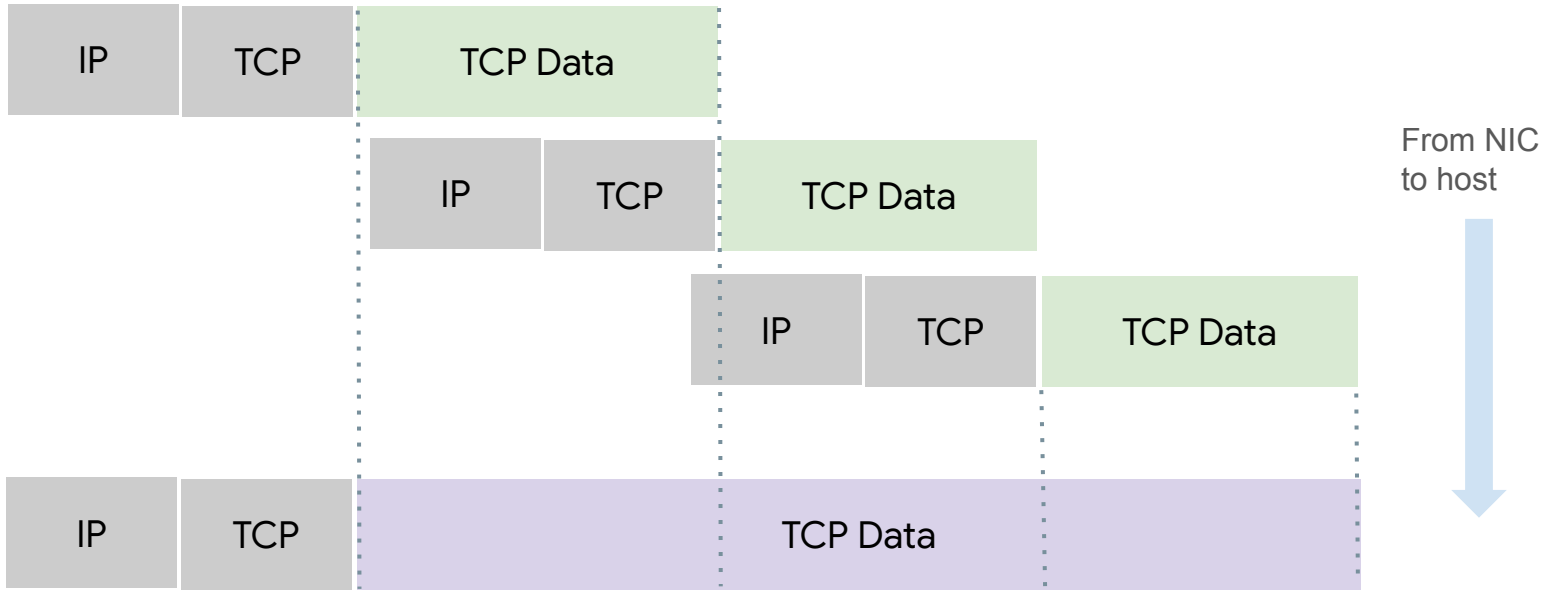
# Transmit Segmentation Offload (TSO)

- Split big packets into MTU sized ones.



# Receive Segmentation Offload (RSO)

- Coalesce small packets into bigger ones.



# Why offload segmentation?

- Allows the host networking stack to process **fewer** (though larger) packets
  - 10Gbps with 1 kilobyte packets ⟶ 1,250,000 packets per second
  - 10Gbps with 1 megabyte packets ⟶ 1,250 packets per second

# Why offload segmentation?

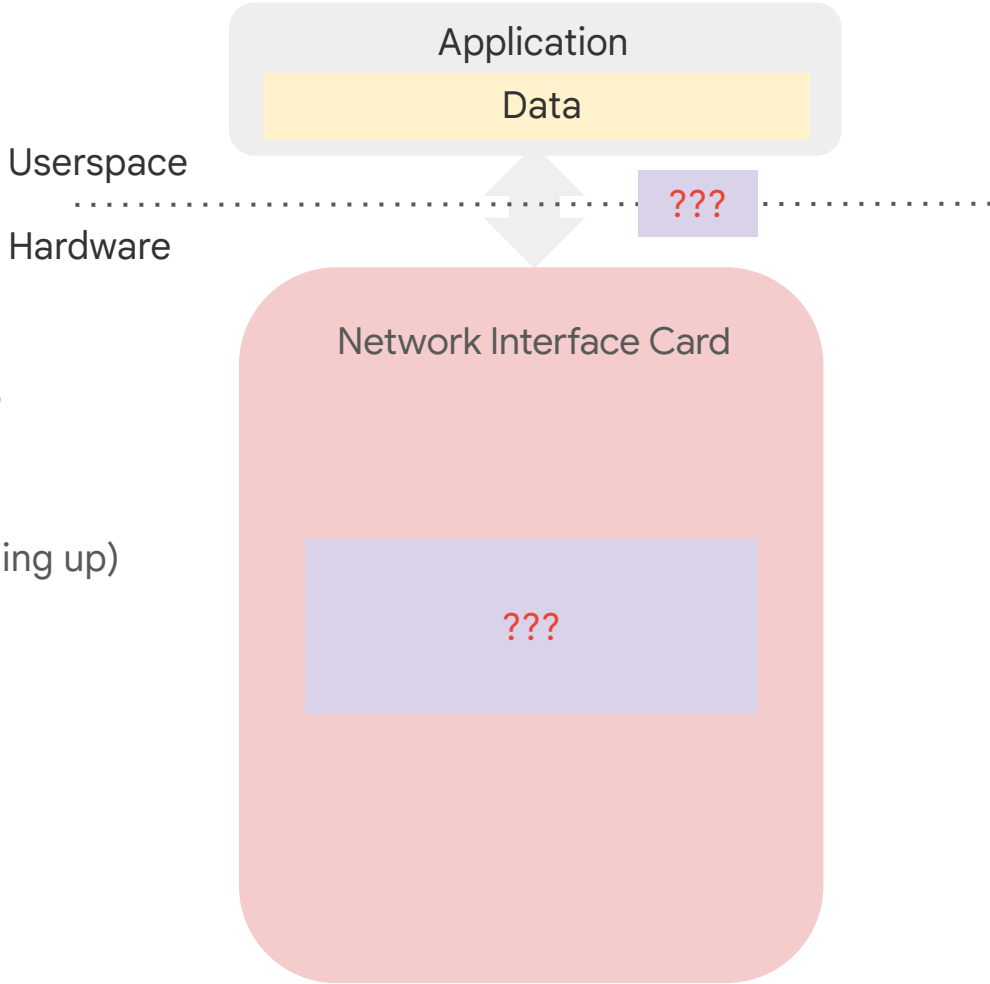
- Allows the host networking stack to process **fewer** (though larger) packets
  - 10Gbps with 1 kilobyte packets ⟶ 1,250,000 packets per second
  - 10Gbps with 1 megabyte packets ⟶ 1,250 packets per second
- Analogy: one shopping trip for N items *vs.* N shopping trips
- Some packet processing overheads depend on the length of the packet (e.g., checksum)
  - Still incurred with segmentation offload
- But many overheads don't; only incurred once *per packet* (e.g., interrupts, add/rm headers)
  - This portion of the overhead decreases if we send fewer packets!
- Hence, host can drive the same bandwidth while consuming fewer host CPU cycles

# Epoch 2: Stateful offloads

## Simple functions that require per-flow or pre-user state:

- Virtual Routing Tables: e.g., forwarding entries for Coke vs. Pepsi VMs (last lecture)
- Rate limiting: e.g., enforcing tenant bandwidth limits (requires per-tenant counters)
- Access Control Lists: e.g., ensuring packets from Coke VMs can’t reach Pepsi VMs

# Epoch 3: Protocol Offloads




- Offload **all** of host networking to the NIC?
- Two broad approaches:
  - A new abstraction altogether: RDMA (coming up)
  - Retain TCP's abstraction, but offload the TCP/IP/Ethernet stack to the NIC

# Lecture Outline

What is Host Networking and Why it Matters

- Learnt before: network stack in the OS implements reliability, congestion control, flow control, *etc.*
- Problem in datacenters: extreme perf. requirements; valuable CPU cores; kernel development hard.
- Host networking in datacenters is heavily optimized for application performance and CPU efficiency.
- Optimization opportunities: OS bypass, and offloads to the NIC

**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 that maintain per-flow or per-tenant state
    - Most complex: offload the entire protocol, e.g. TCP, RDMA.

**Remote Direct Memory Access**

# RDMA: Topics we will cover

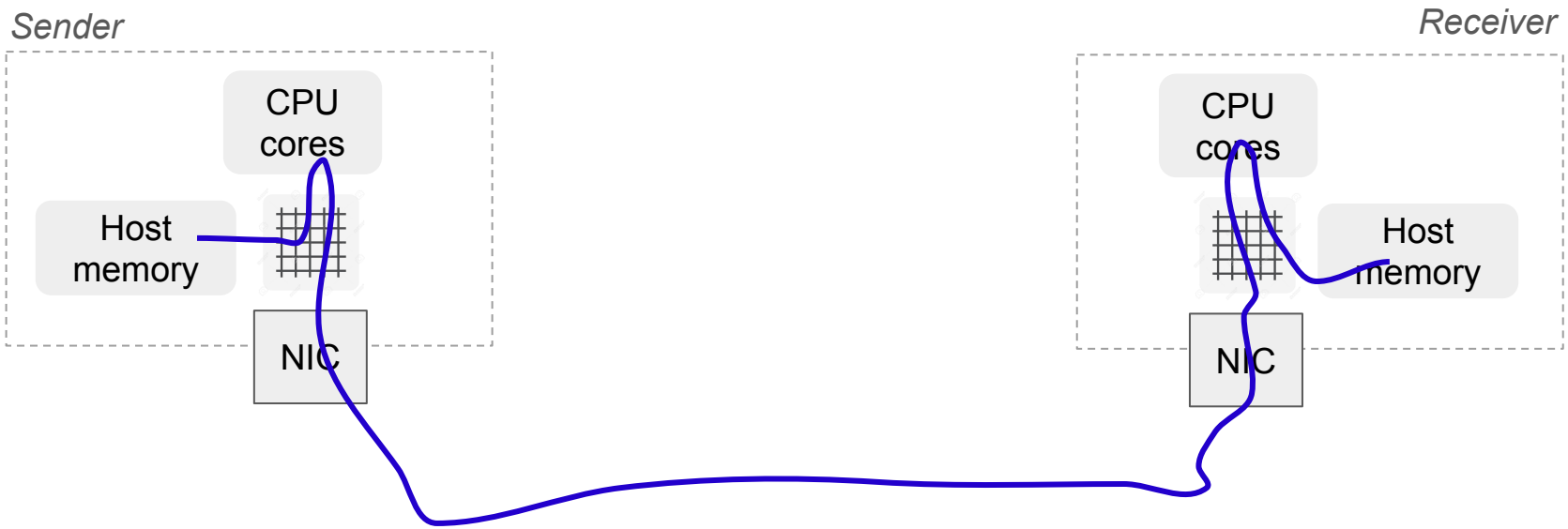
- An overview of RDMA
- RDMA pros, cons, and applications
- The RDMA abstraction
- A walk through of an RDMA Send operation

# Remote Direct Memory Access (RDMA)

- Originally developed as a network abstraction for supercomputer environments
  - Alternative to the TCP/IP network abstraction
- Goals: high performance and efficiency for “tightly coupled” distributed applications
- Increasingly prevalent in datacenter and AI/ML environments
- What RDMA offers: Direct transfer of data from application memory at server A to application memory at server B without consuming CPU time at server A or B

# 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

- CPU efficiency: CPUs are minimally involved in data transfers*
- High performance low latency, high bandwidth data transfers

## Cons

- More complex than traditional networking
- More limited ecosystem: RDMA requires specialized hardware and software
  - Not backwards compatible with TCP/IP applications.
  - Typically used between hosts that are in close proximity and coordination

For example: reduced CPU consumption due to host networking ~20% to < 1%

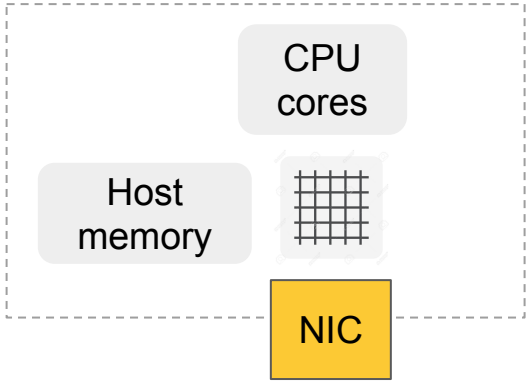
# Applications of RDMA

- High Performance Computing (HPC) applications
  - Scientific simulations: astrophysics, biological systems, weather forecasting, etc.
  - Key driver: all the benefits of RDMA (low latency, high throughput, CPU efficiency)
- Low Latency applications
  - ML inference, search queries, financial applications
  - Key driver: low and predictable latency for small messages
- Cloud Computing applications
  - VM migration, distributed file systems
  - Key driver: CPU efficient high throughput transfers
- ML Training
  - Key driver: Predictable latency for high bandwidth transfers. Also CPU efficiency.

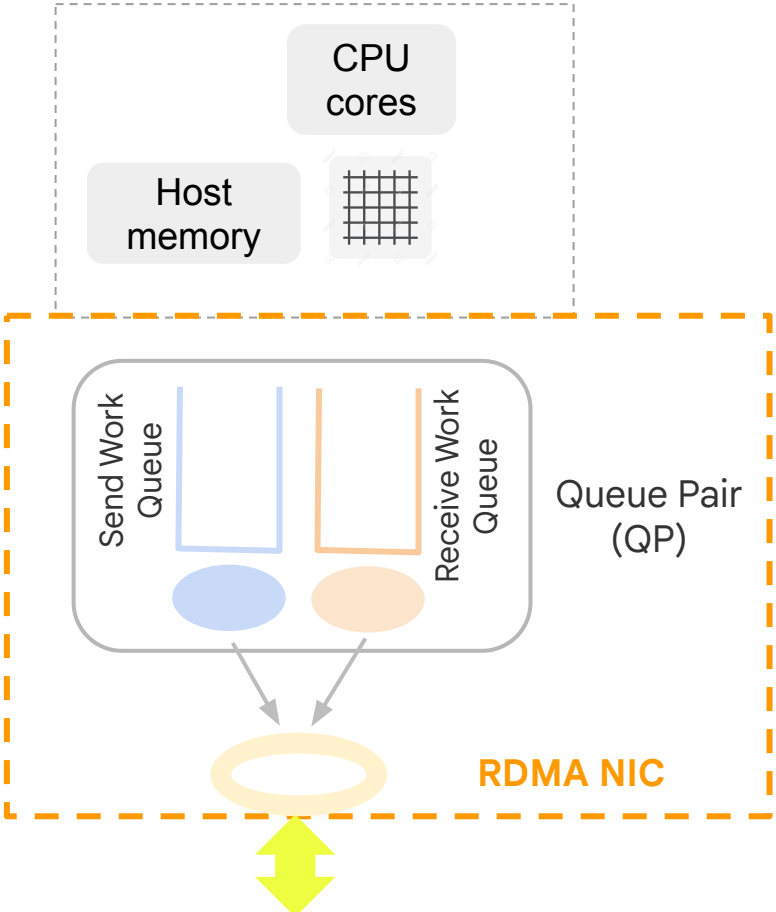
# RDMA overview

- 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 abstraction: RDMA “queue pairs”
  b. Offloading common tasks (congestion control, reliability, ordering, etc.) to the RDMA NIC and/or network fabric.

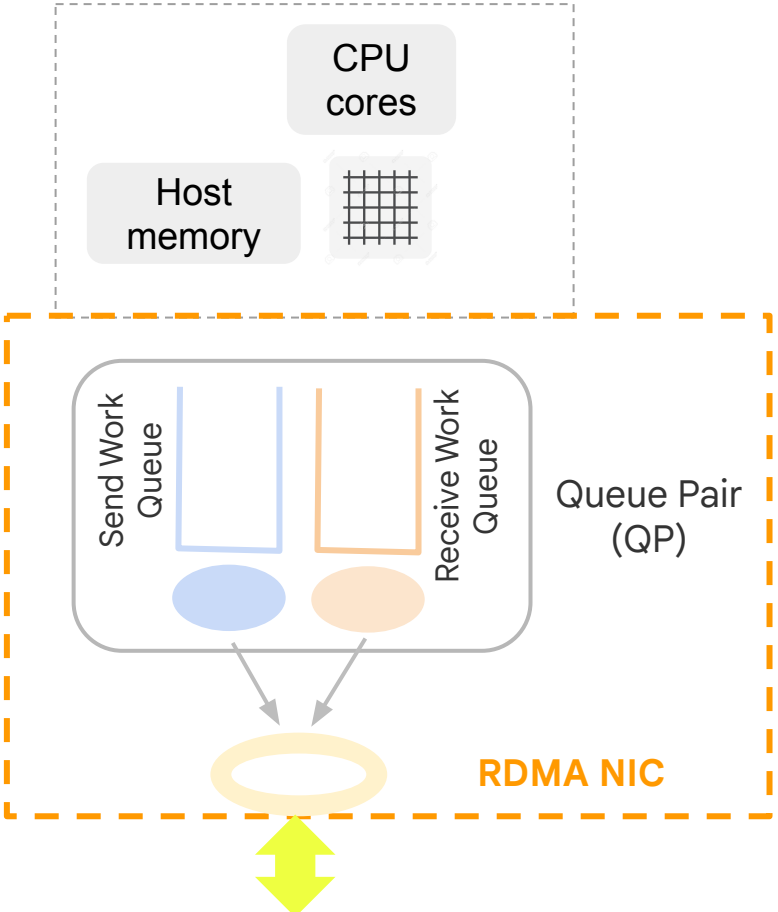
# RDMA Queue Pairs are the interface between the application and RDMA NIC



# RDMA Queue Pairs are the interface between the application and RDMA NIC

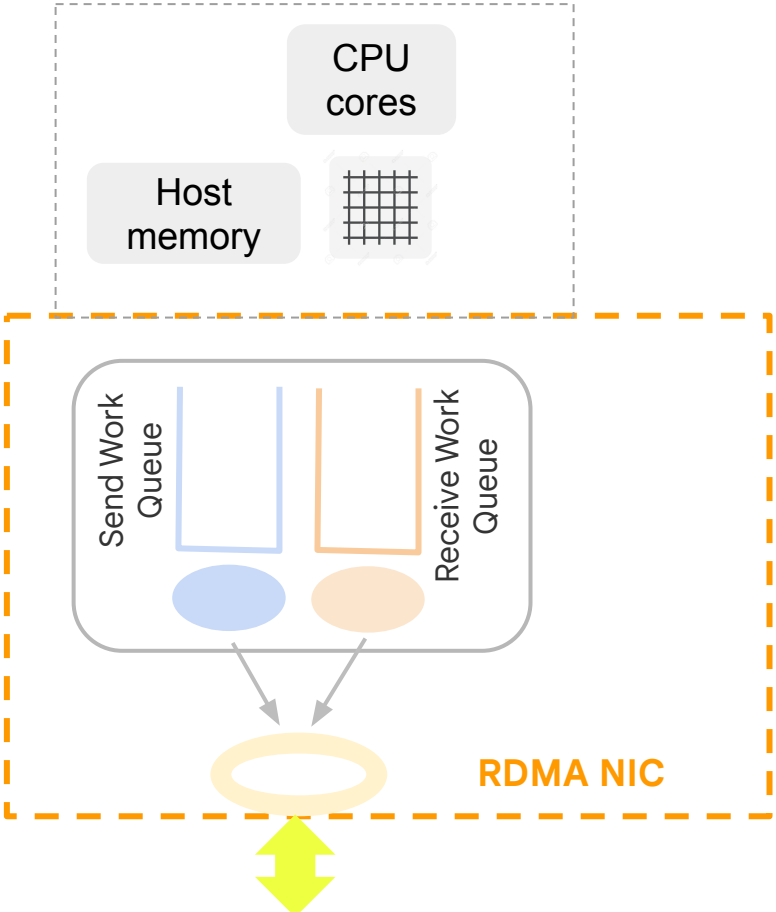


# RDMA Queue Pairs are the interface between the application and RDMA NIC



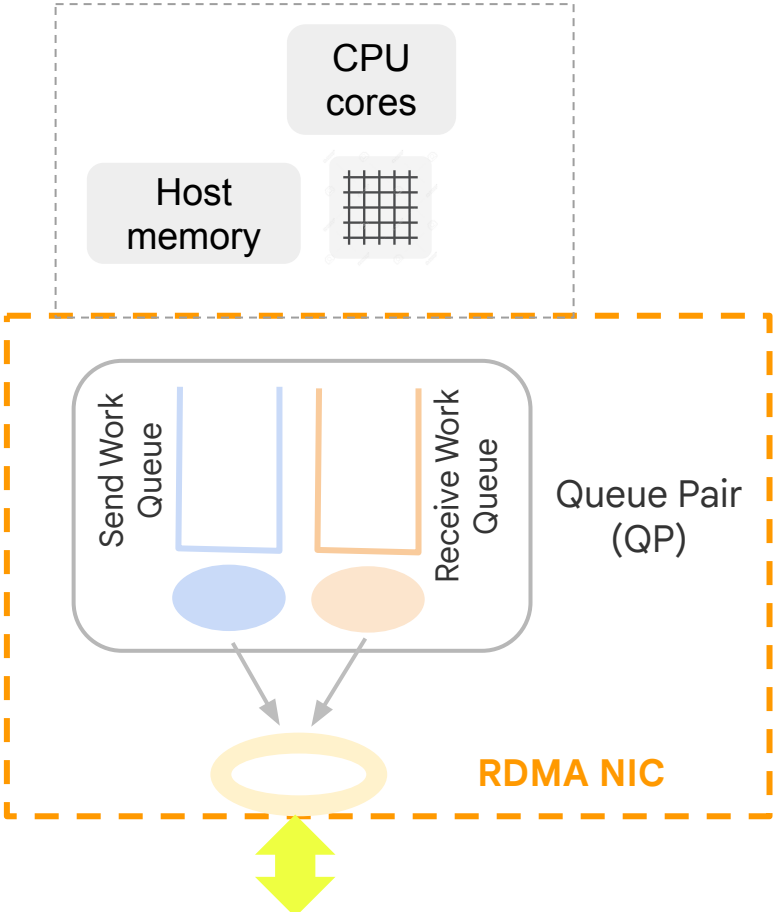

- Send and Receive Work Queues are created in pairs. Used by the CPU to *schedule* transfers
  - Not to hold data!
- Different types of Queue Pairs:
  - Ordered vs. unordered.
  - Reliable vs. unreliable
  - *Etc.*
- “Reliable Connected” Queue Pairs
  - Closest to traditional TCP connections.
  - Connection establishment is out of band, exchanges Queue Pair Ids, *etc.*

# RDMA: Send and Receive Work Queues



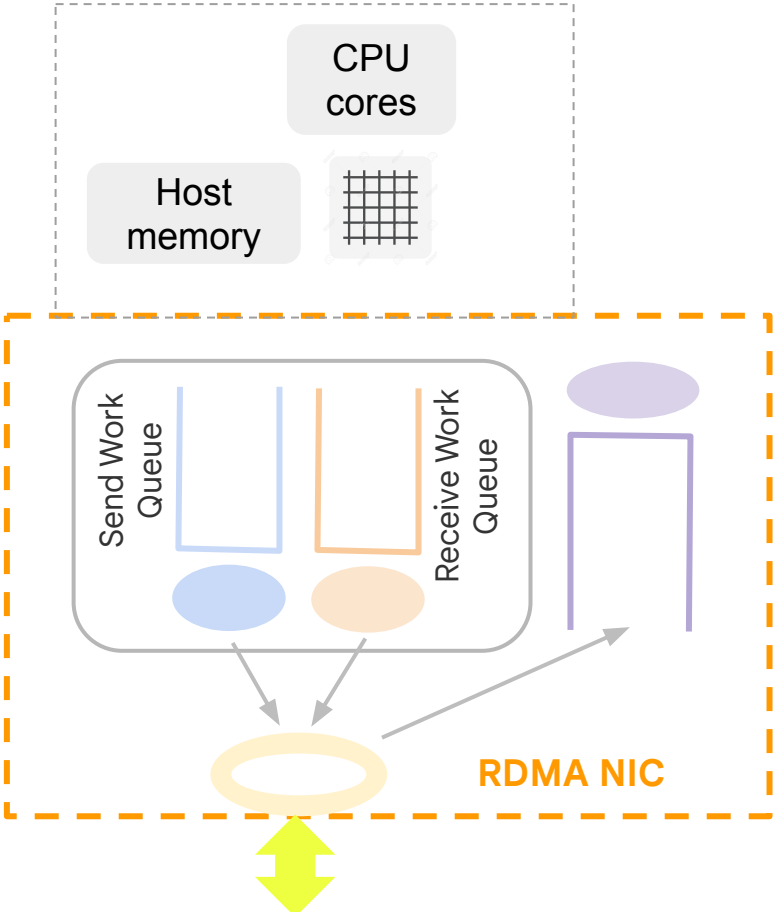

- **Send Work Queue** is responsible for managing outgoing RDMA transfers initiated by the local host.
- **Receive Work Queue** responsible for managing incoming RDMA operations from a remote host.

# RDMA: Work Queue Entry (WQE)




- Application instructs the NIC about data it wants to send/receive by writing Work Queue Entries (WQEs, pronounced “wookie”) to the Send/Receive Work Queue
- 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: Completion Queues



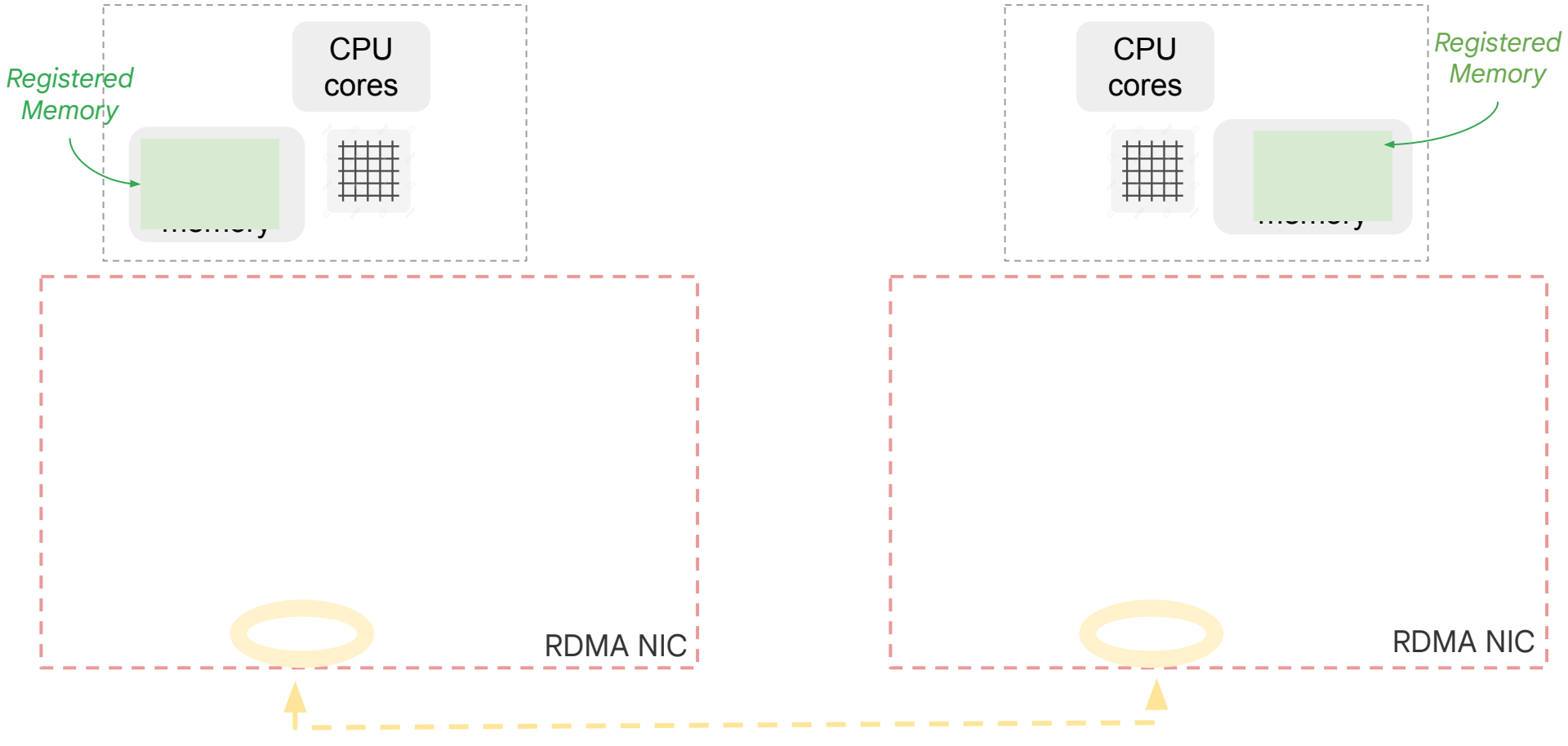

- **Completion Queue (CQ)** Stores completion notifications for both Send and Receive Queue transfers.
- The application reads the Completion Queue to understand the status of its RDMA transfers.

# RDMA 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 ...

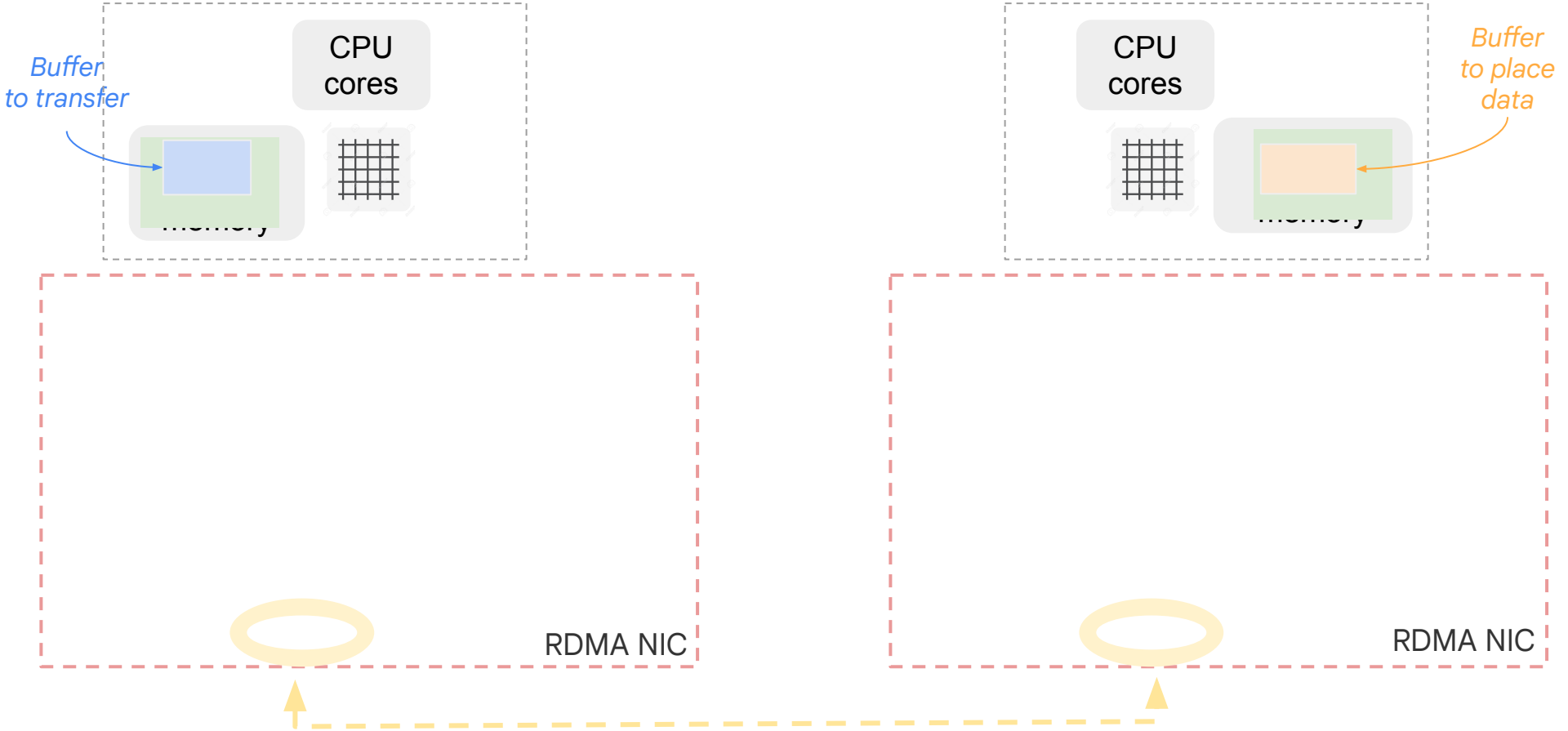
Application (sender side)

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

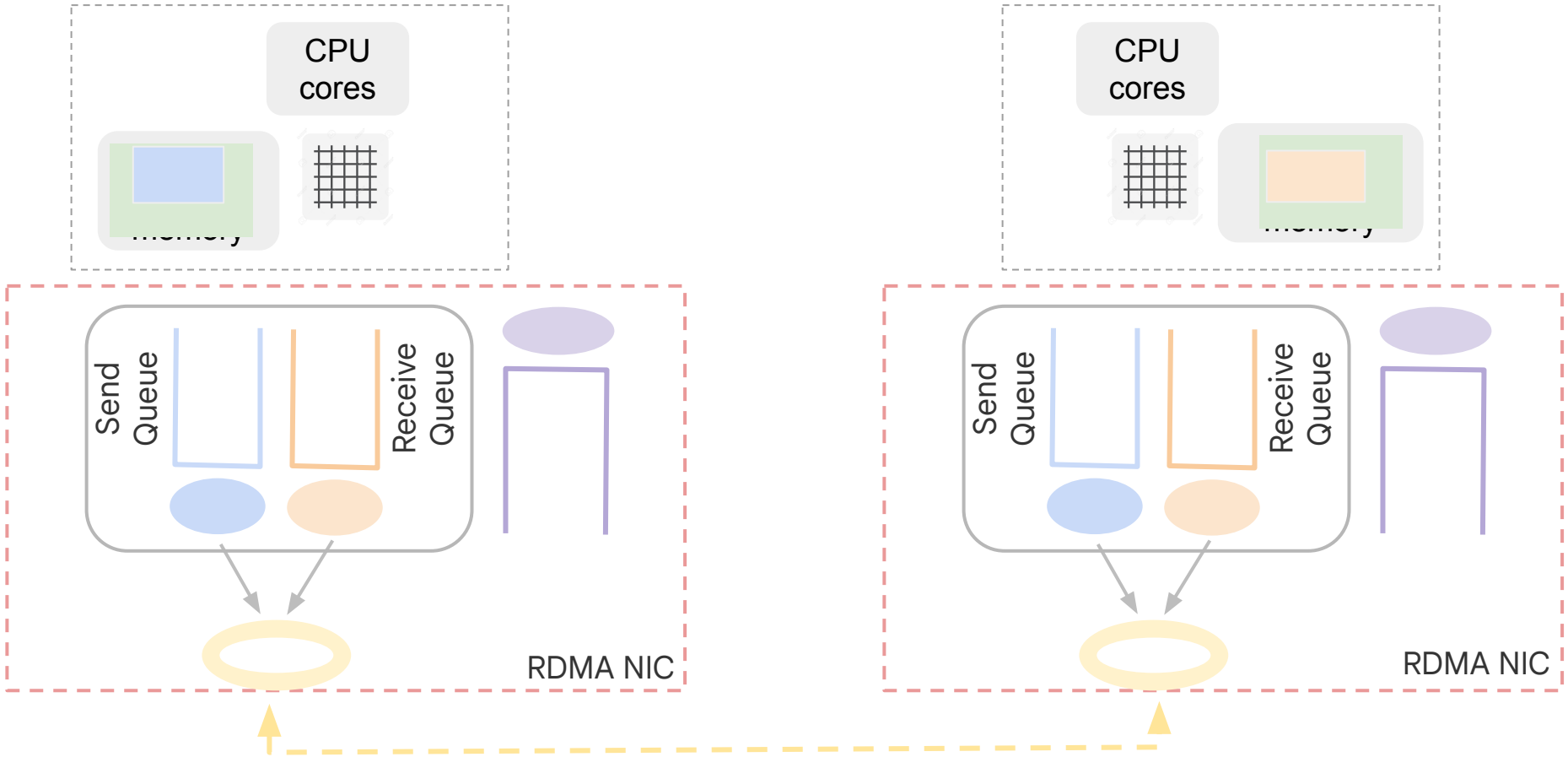


Application (sender side)

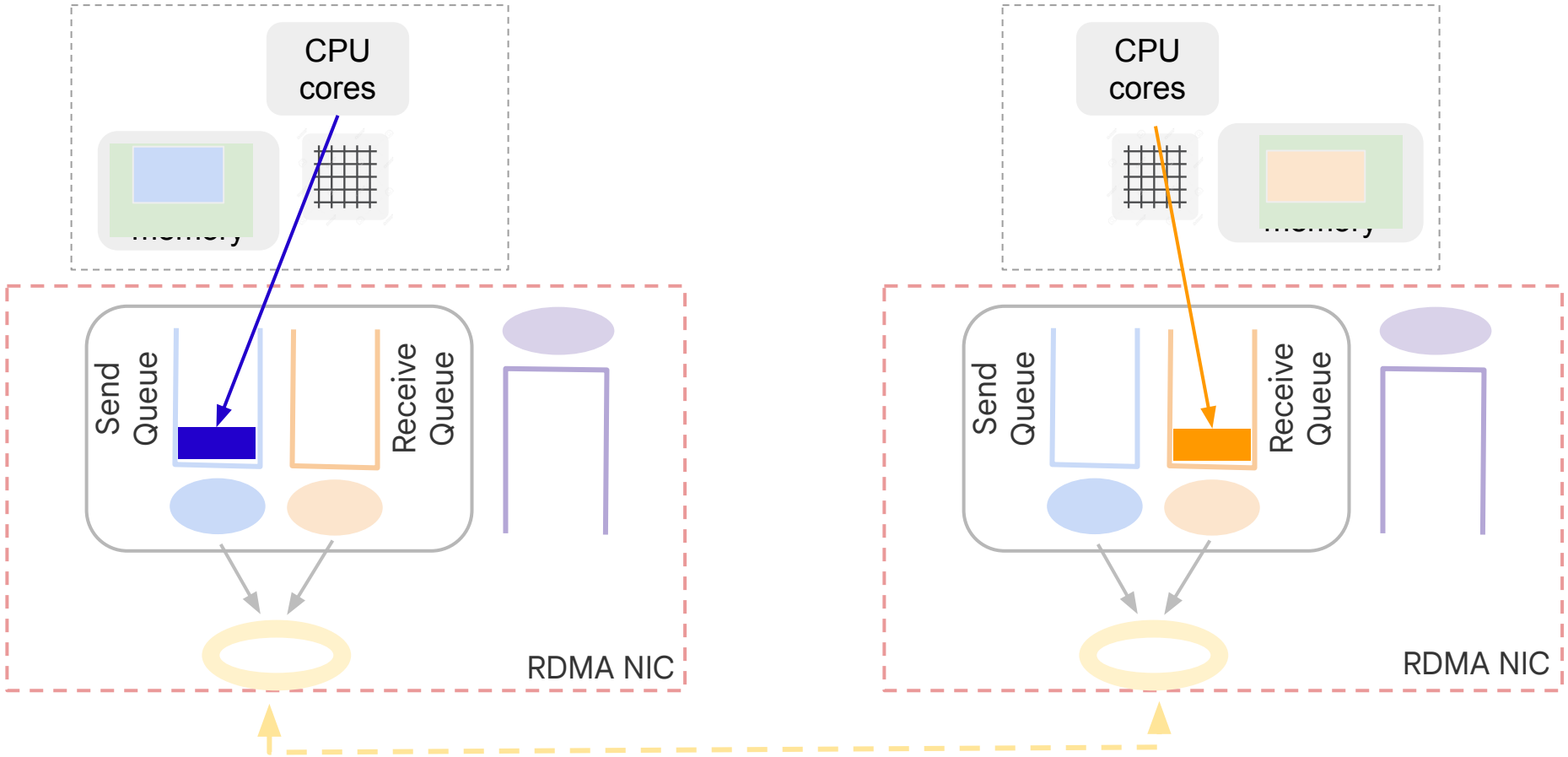
# [Step 0] RDMA transfers “messages” from/to this registered memory region



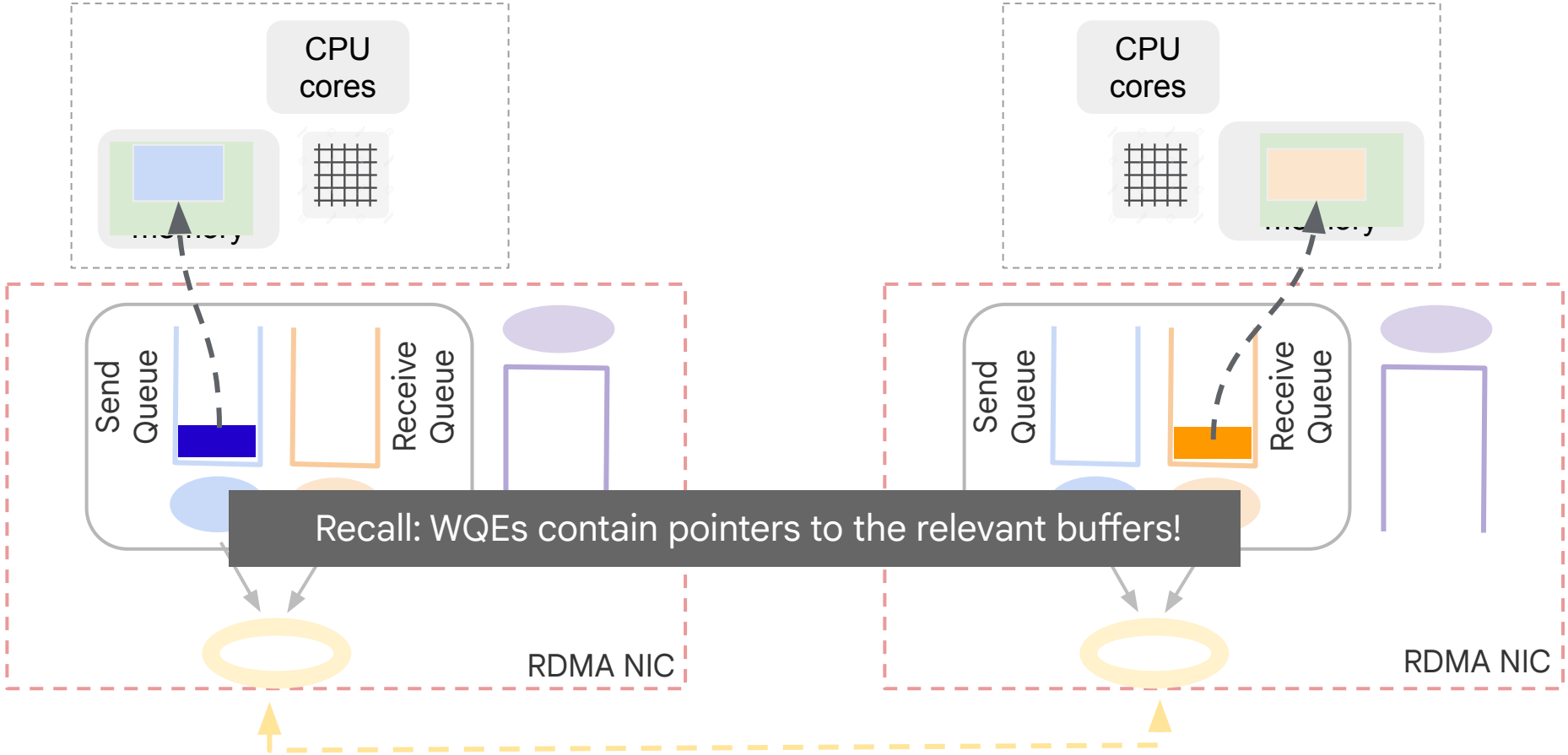
# [Step 1] Create Queue Pairs, Completion Queues



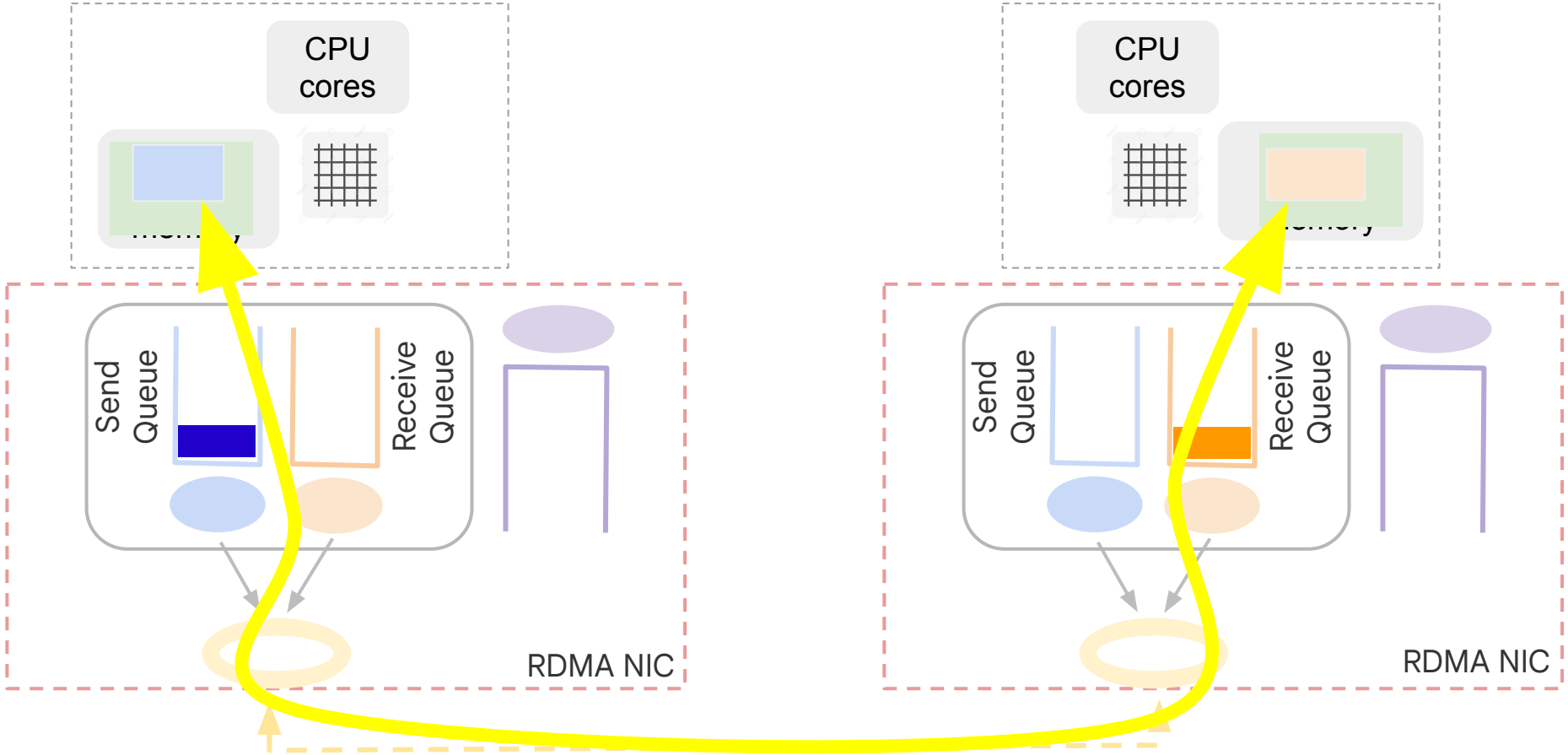
# [Step 2] Application creates Work Queue Entries on Send and Receive Queues



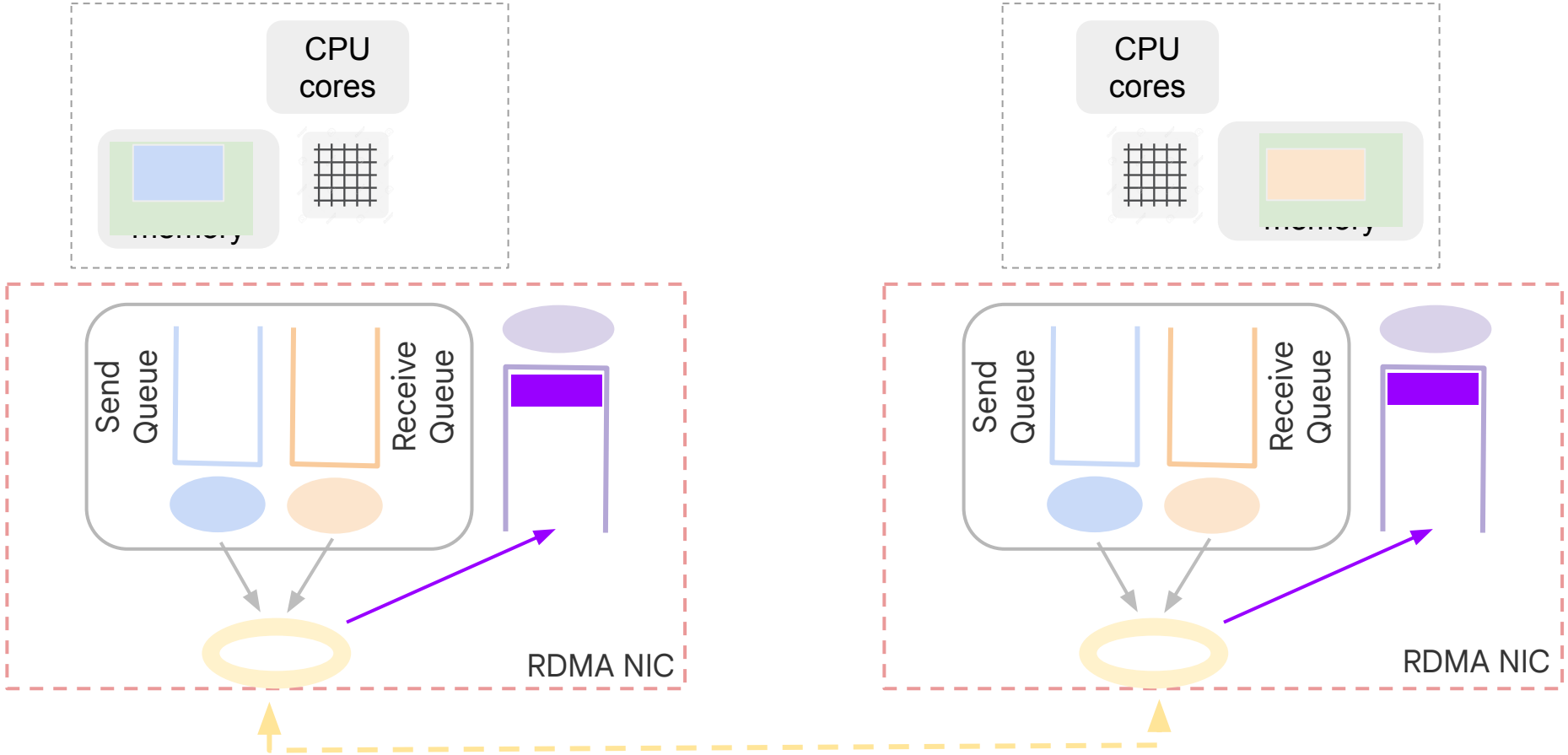
# [Step 2] Application creates Work Queue Entries on Send and Receive Queues



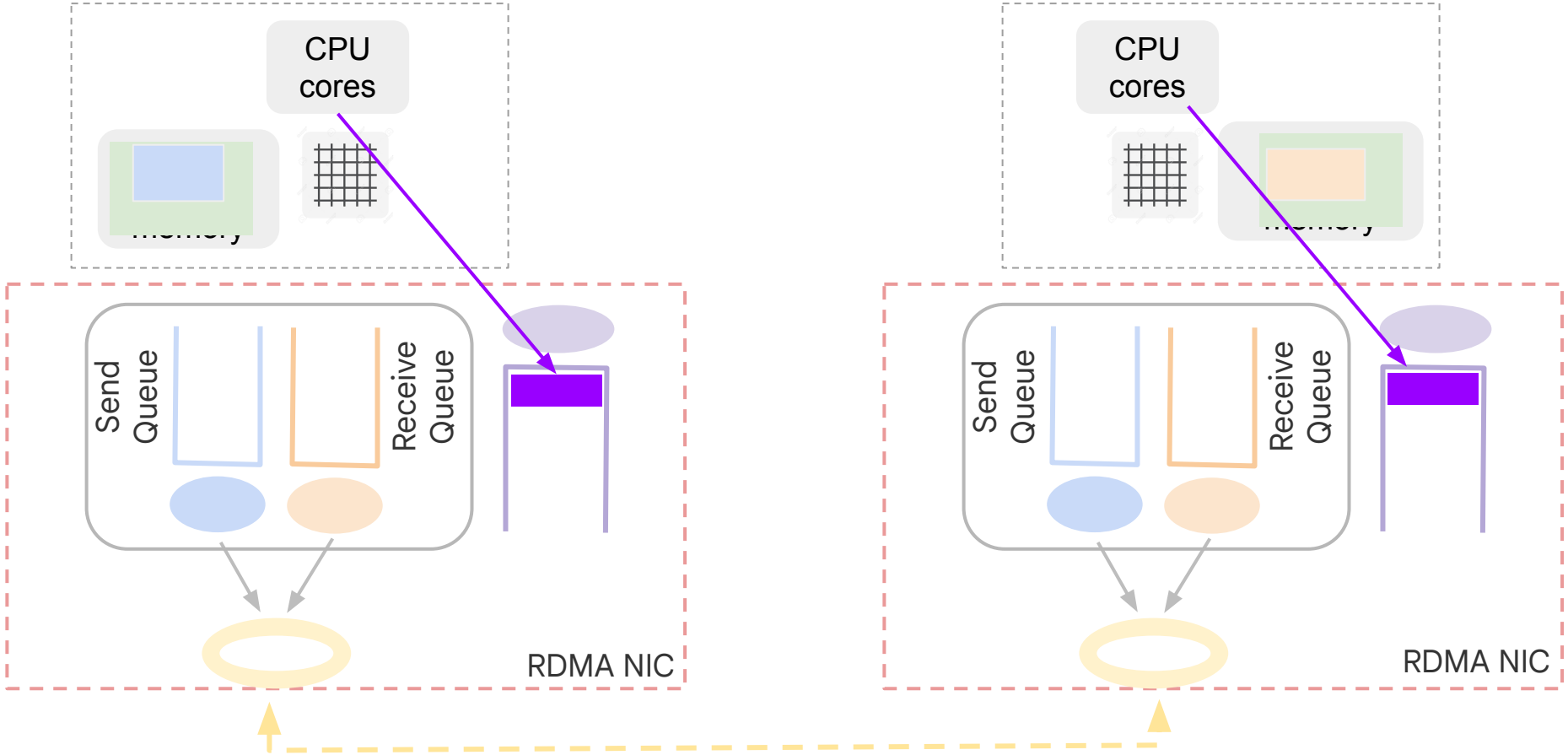
# [Step 3] Memory to Memory Data Transfer (without CPU involvement!)



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



# [Step 5] Application Processes Completion Queue Entries



# Implementing the RDMA abstraction

- What about loss, congestion, ordering, *etc.*? Still need to address these issues!
- Addressed by the lower layers of RDMA, implemented in the NIC and switches
- Two broad approaches
  1. Infiniband: uses switch support for reliability, CC, *etc.* Big departure from TCP/IP/Ethernet
  2. RDMA over converged Ethernet (RoCE): Allows running RDMA over modified Ethernet, IP
- Jury still out on which approach is best!

# Taking stock: RDMA

- Departure from the traditional sockets, TCP/IP, Ethernet stack
  - New application abstraction
  - New division of functionality between hosts and switches
  - New division of functionality within hosts (app *vs.* OS *vs.* NIC)

- Adoption driven by modern app needs and enabled by the unique characteristics of datacenter environments
  - Applications need high performance and efficiency
  - In datacenters we can co-design apps, hosts *and* switches to adopt a new architecture

# Lecture Outline

## What is Host Networking and Why it Matters

- Learnt before: network stack in the OS implements reliability, congestion control, flow control, *etc.*
- Problem in datacenters: extreme perf. requirements; valuable CPU cores; kernel development is hard.
- Host networking in datacenters is heavily optimized for application performance and CPU efficiency.
- Optimization opportunities: OS bypass, and offloads to the NIC

## 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 that maintain per-flow or per-tenant state
    - 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.

# Closing Thoughts

- Original Internet protocols and implementations not designed for performance
- Datacenter environments have changed / are changing / the game
- Lessons
  - Understand your constraints
  - Contemplate technology trends
  - Master “what if” reasoning