

Distributed Computing frameworks

The case of distributed training, review & RDMA

Xingda Wei, Yubin Xia

IPADS, Shanghai Jiao Tong University

<https://www.sjtu.edu.cn>

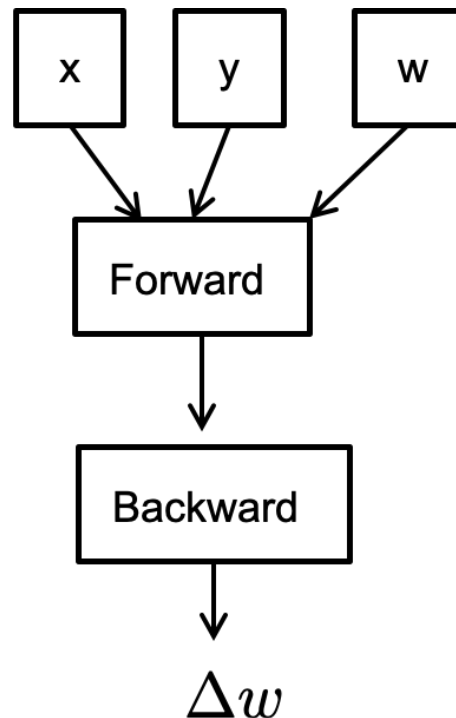
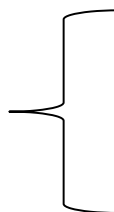
Pseudocode for Stochastic Gradient Descent

```
// execute on a master
```

```
for iter in num_iters:
```

```
    iter+1 iter
```

iter



Review: Parallelization Opportunities

Data Parallelism: *Distribute the processing of data to multiple PEs.*

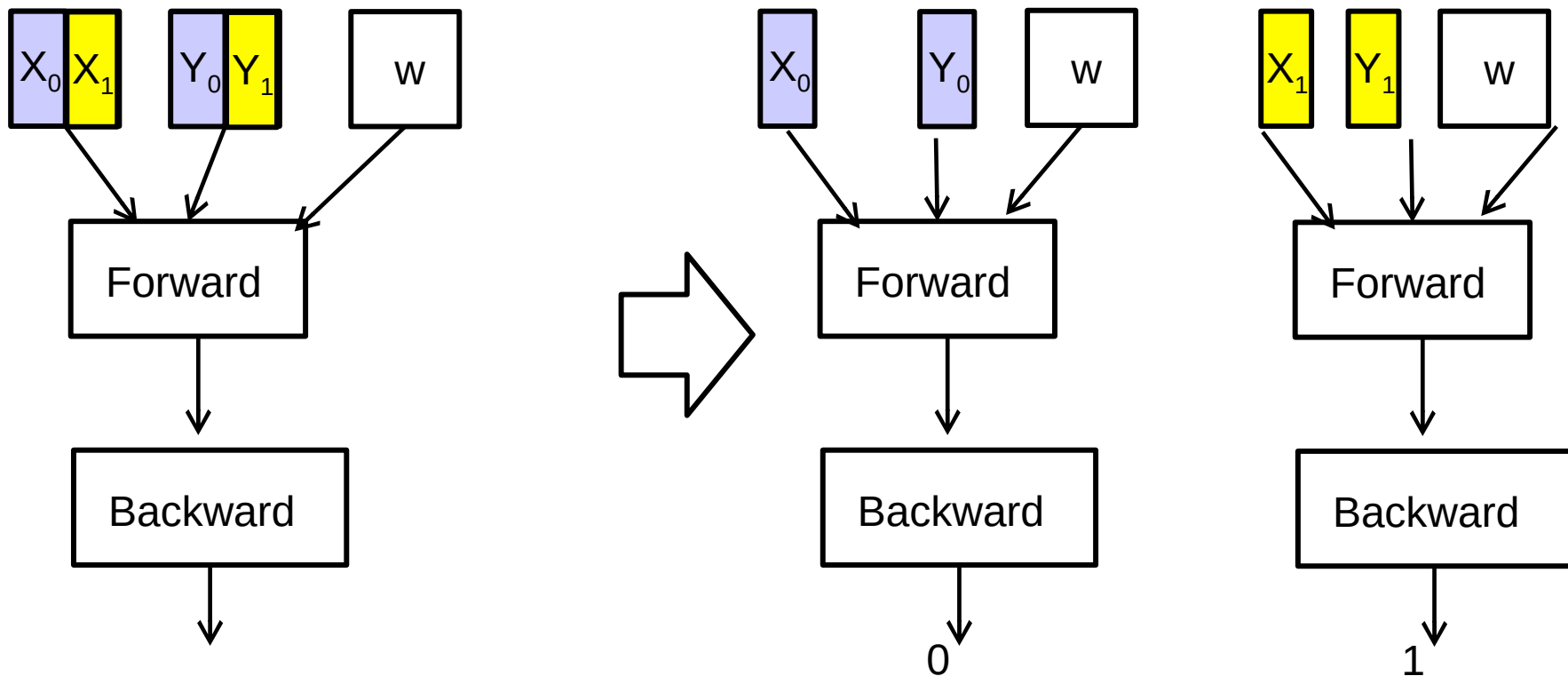
Model Parallelism: *Break the model and distribute processing of every layer to multiple PEs*

*For either approach it is also possible to use **synchronous** or **asynchronous** updates*

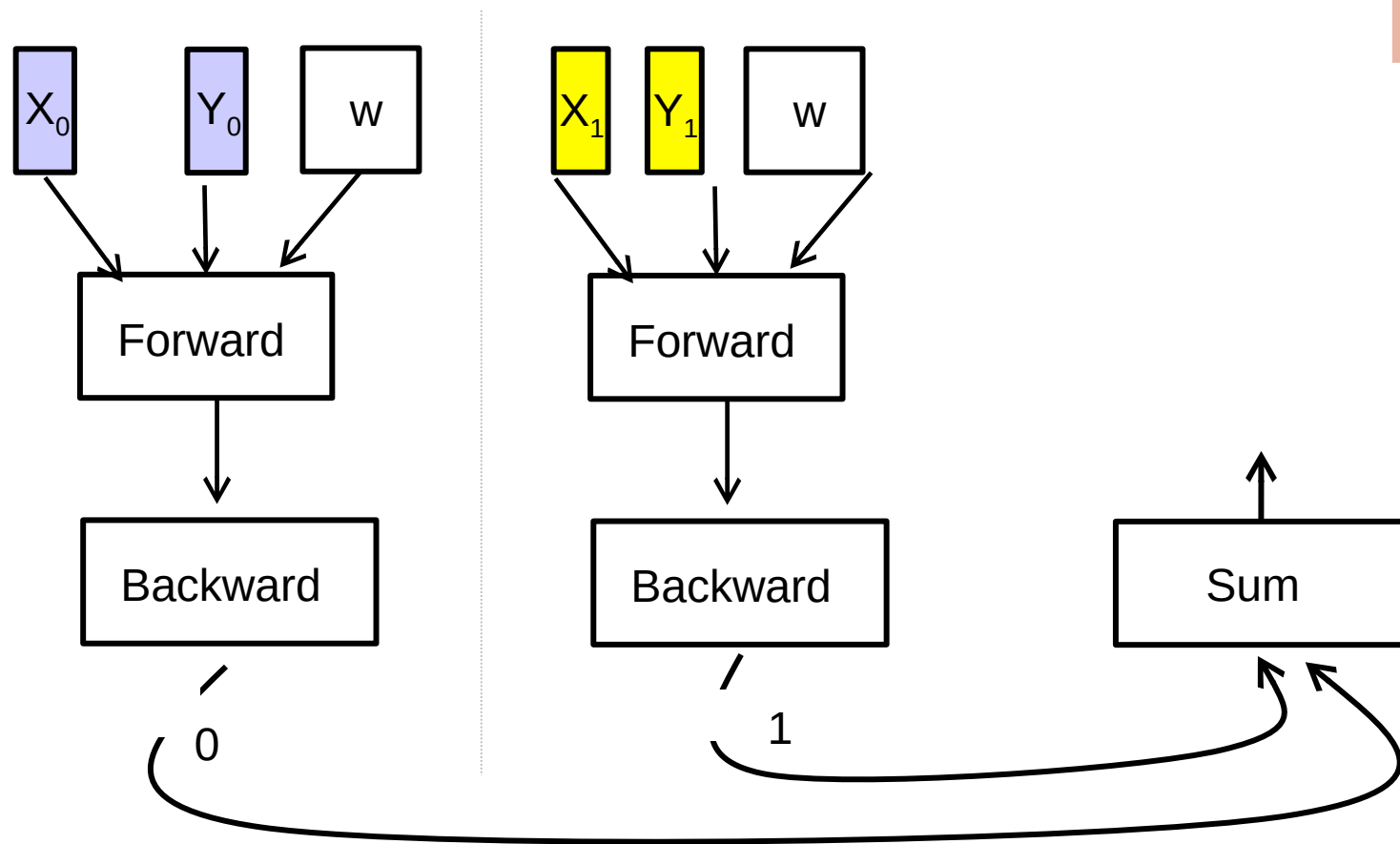
Synchronous Data Parallelism

Parallelize on the x (and y)

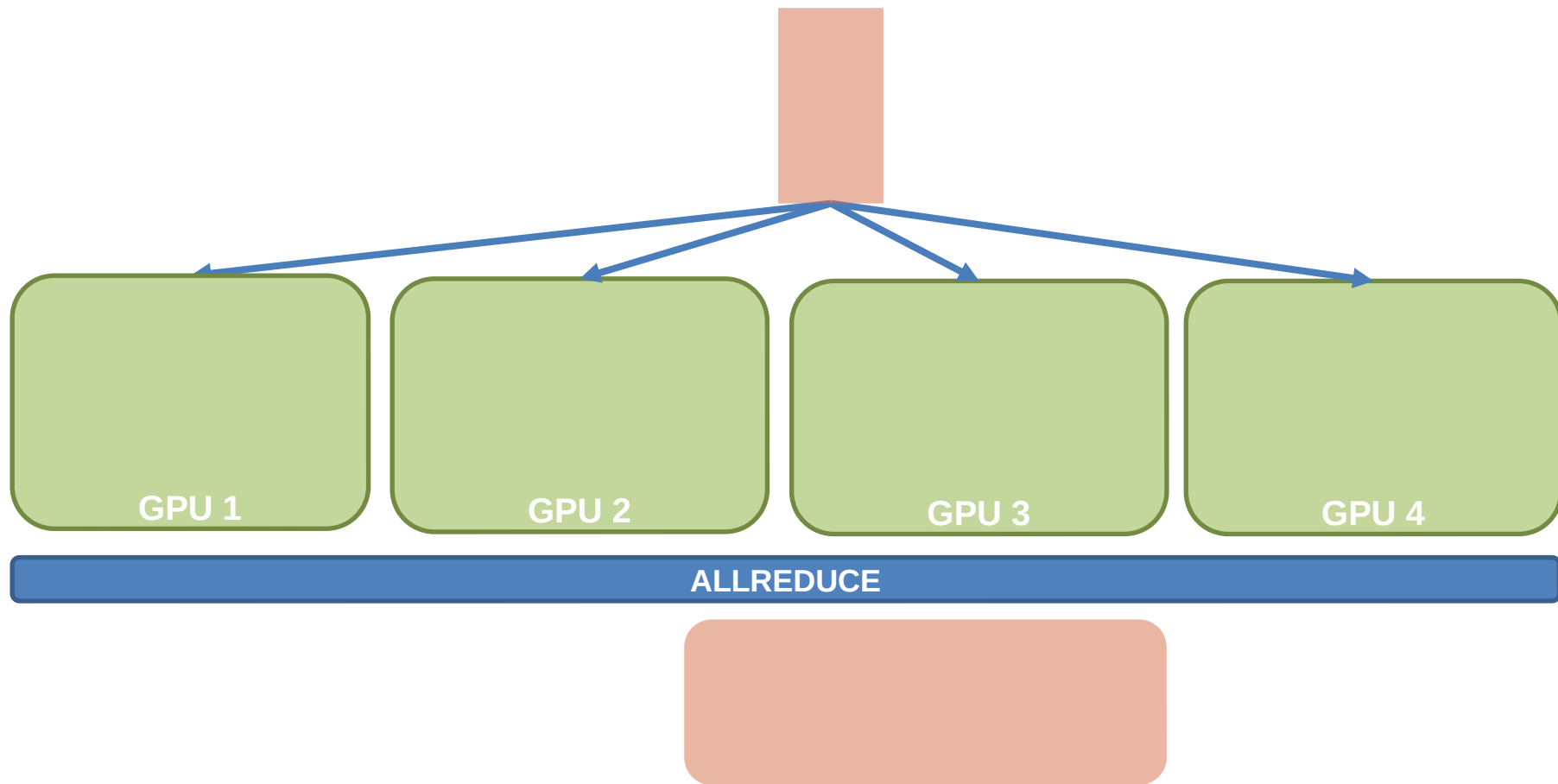
- Observation: the sum operator can be easily parallelized



Review: Data Parallelism



Review of Key communication operator: allreduce



Review: System requirements for AllReduce

Overhead analysis: computation + network

The overhead of computation is typically small

- Compute a sum operation on device is highly optimized, e.g., SIMD, GPU, TPU, etc.

Goal: reduce network overhead

- Reduce bytes sent per-message
- Reduce concurrent messages sent to a server

Review of Allreduce optimizations

Trade-off between rounds vs. fan-in performance

- P : # processors/machines/GPUs participated in the computation
- N : the size of the data to be reduced

	Round	Peak node bandwidth	Per-node fan-in
Parameter server (PS)	$O(1)$	$O(N * P)$	$O(P)$
Decentralized Allreduce	$O(P)$	$O(N)$	$O(1)$
Ring allreduce	$O(2 * P)$	$O(N/P)$	$O(1)$

Drawback of data parallelism: replicated model

Data parallelism assumes the model (parameters) are replicated on all the processes

- Such that they can do the forward & backward pass dependently
- What if the model cannot fit onto the device?

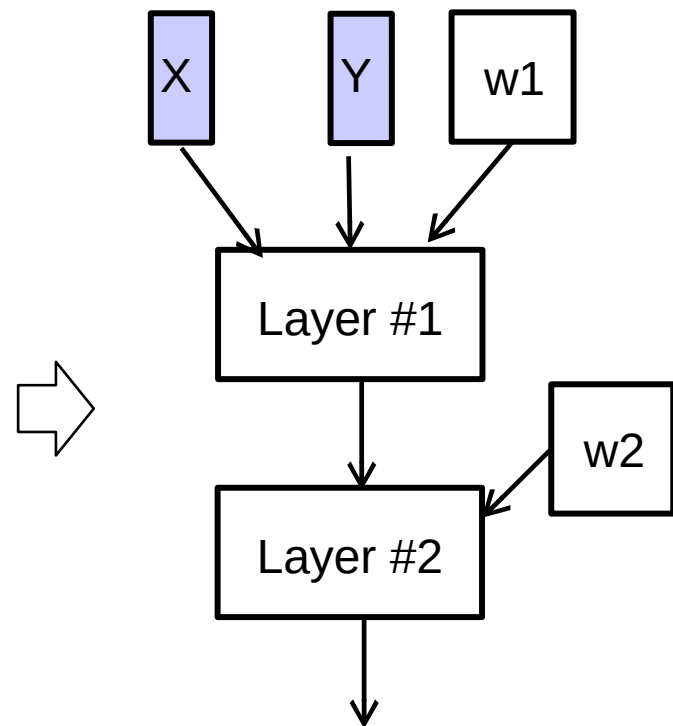
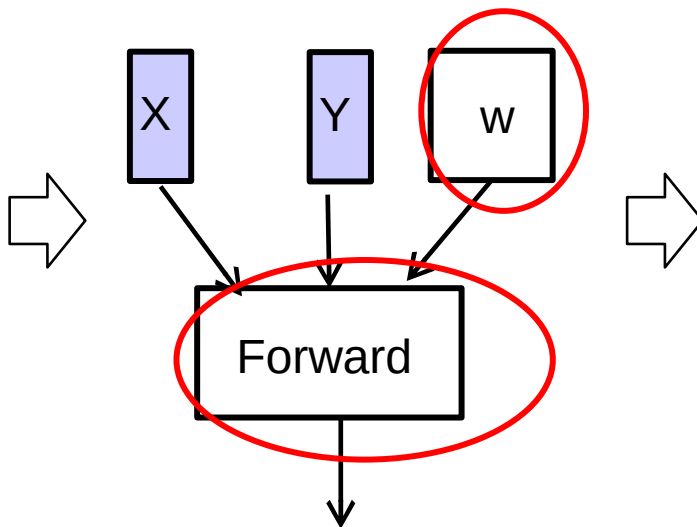
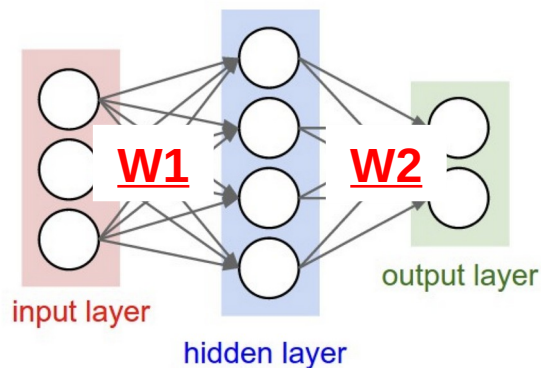
Current trends: models are becoming larger and larger

- No country (or company) can tolerate the risk of falling behind in the AI race

Review: Model parallelism

Partition the parameters of a model, typically done in two ways:

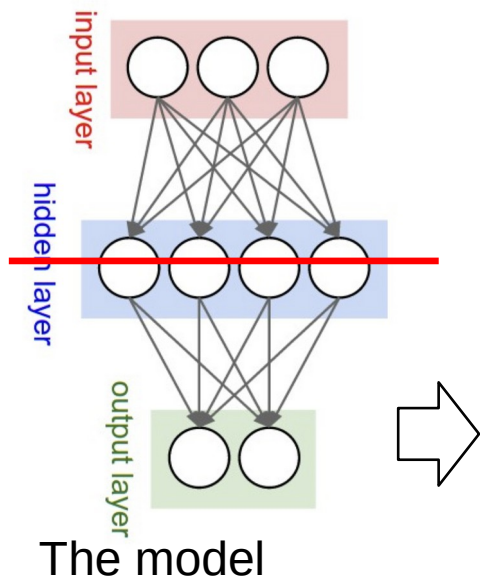
- **Partition on the layer**: pipeline parallelism
- **Partition on the W** : tensor parallelism



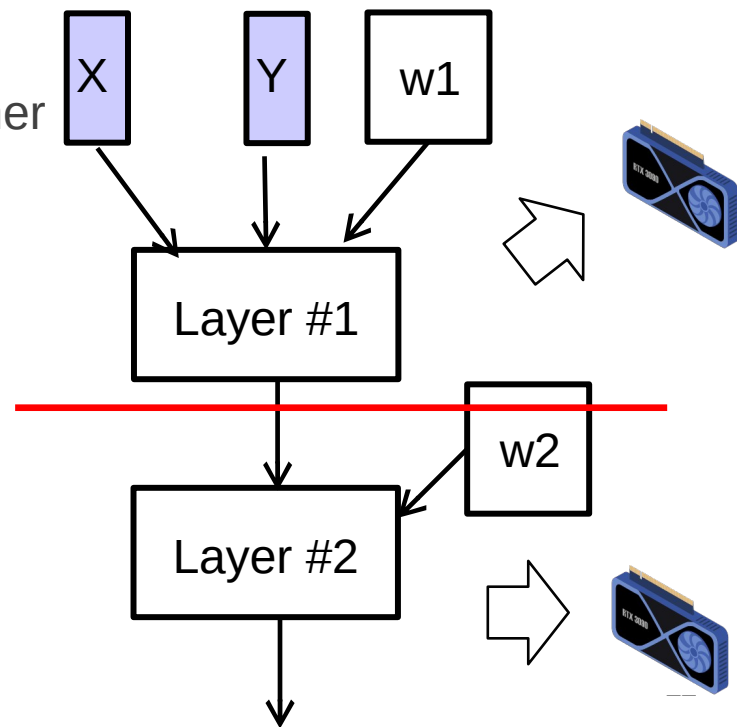
Review: Pipeline parallelism

Partition the computation layer-by-layer, each partition is deployed on a device

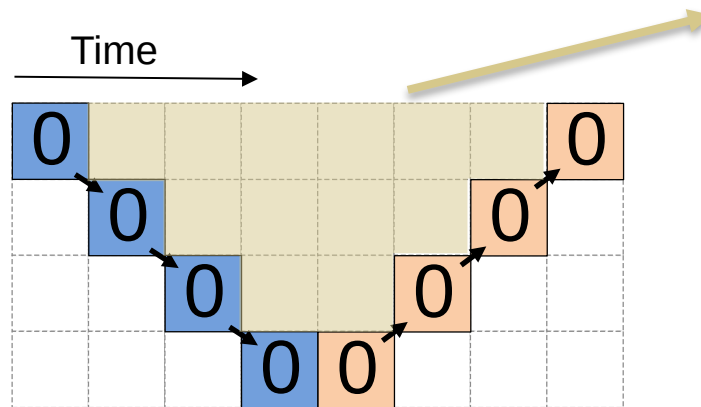
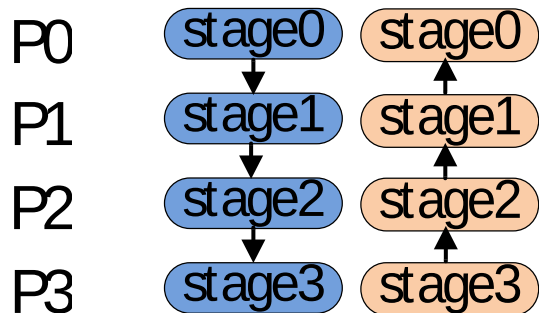
- Note that different layers can be grouped together
- The strategy is out of the scope of this course



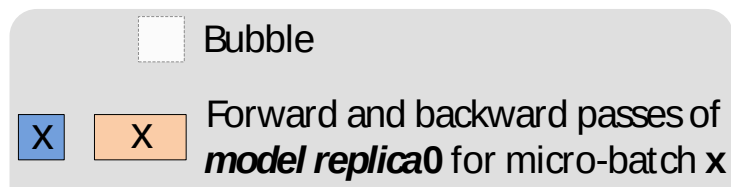
The partitioned
computation
graph



Review: Pipeline parallelism



Bubble where
processes are idle



How to reduce bubble?

Technique for pipeline: micro-batching

Reducing the bubble size by breaking the batch size into smaller pieces to reduce the idle time of the processes

- Reduces bubble size in an easy way to implement

Question: to what extent can micro-batching improves performance?

- The bubble size depends on the #stages
- The ratio bubble overhead depends on the overall pipeline numbers

P0	0	1	2	3								0	1	2	3
P1		0	1	2	3					0	1	2	3		
P2			0	1	2	3			0	1	2	3			
P3				0	1	2	3	0	1	2	3				

Pipeline parallelism & the number of micro-batch

Question: what is the overhead of pipeline parallelism?

- Bubble time fraction:

Optimization directions

- 1. increase m (e.g., increase the batch size)
- 2. reduce p (reduce the #partitions)

Not always possible:

- Increase m may decrease accuracy (or convergence rate), and increase memory usage (due to extra storing of the
- Reduce p : cannot store the partitioned parameters in the precise GPU memory

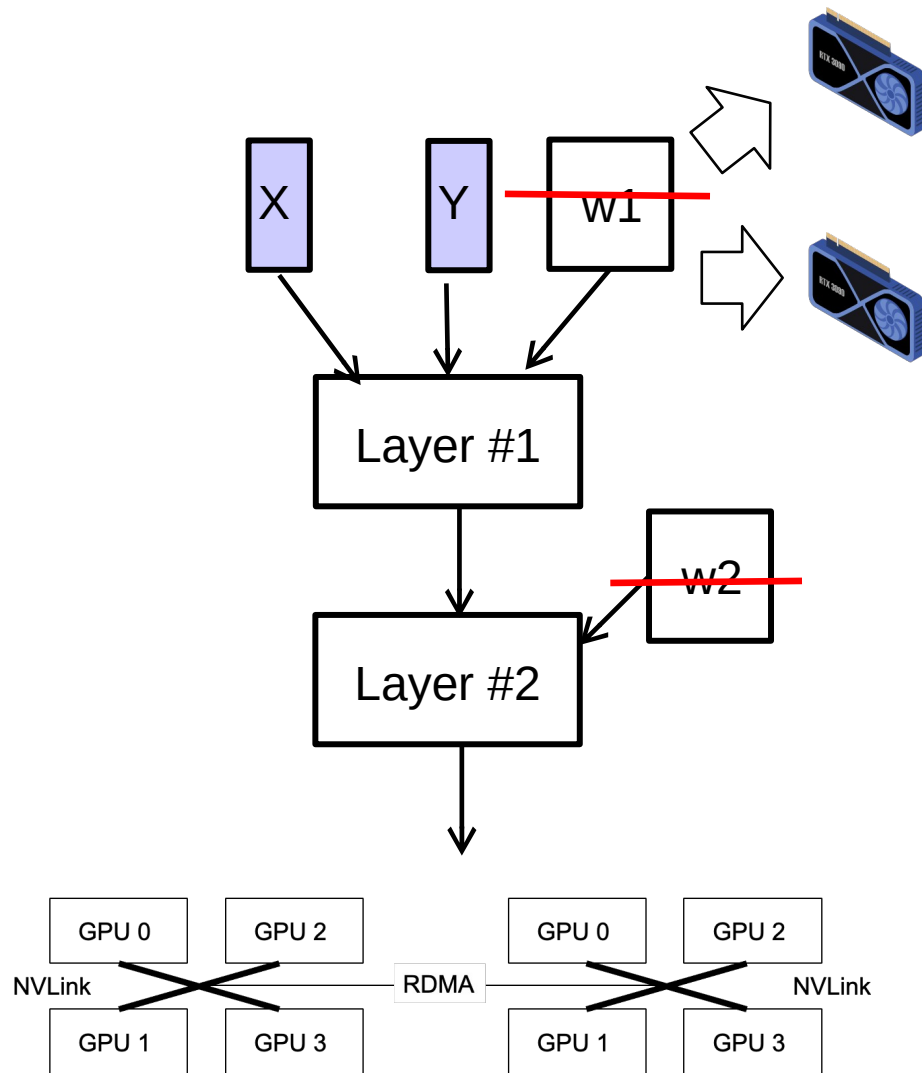
Tensor parallelism

Partition the parameters of a layer

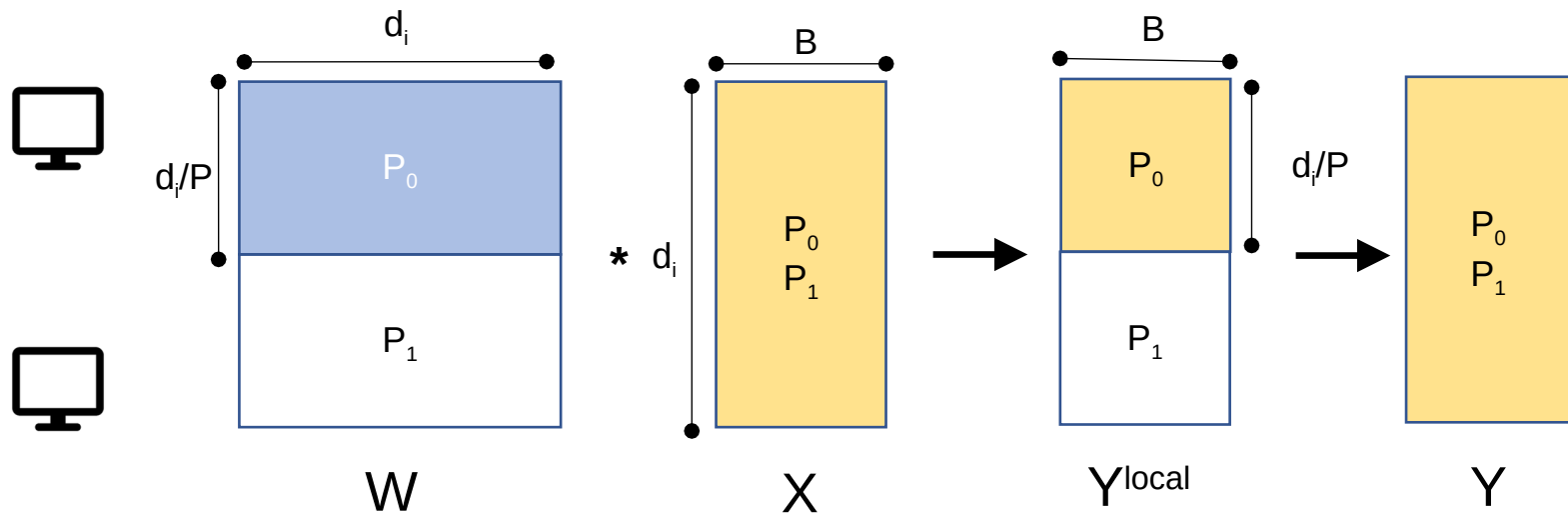
- Each partition is deployed on a separate GPU

Pros

- Support pipeline parallelism with a large layer
- Fit the hardware architecture of modern servers

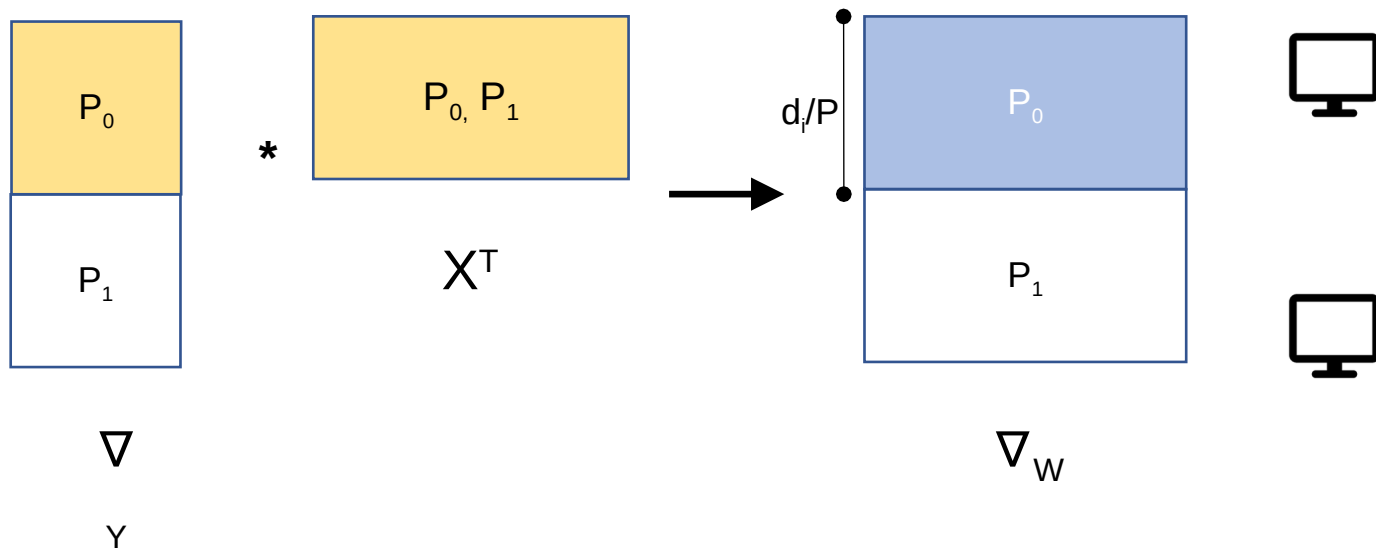


Model Parallelism: Forward Pass



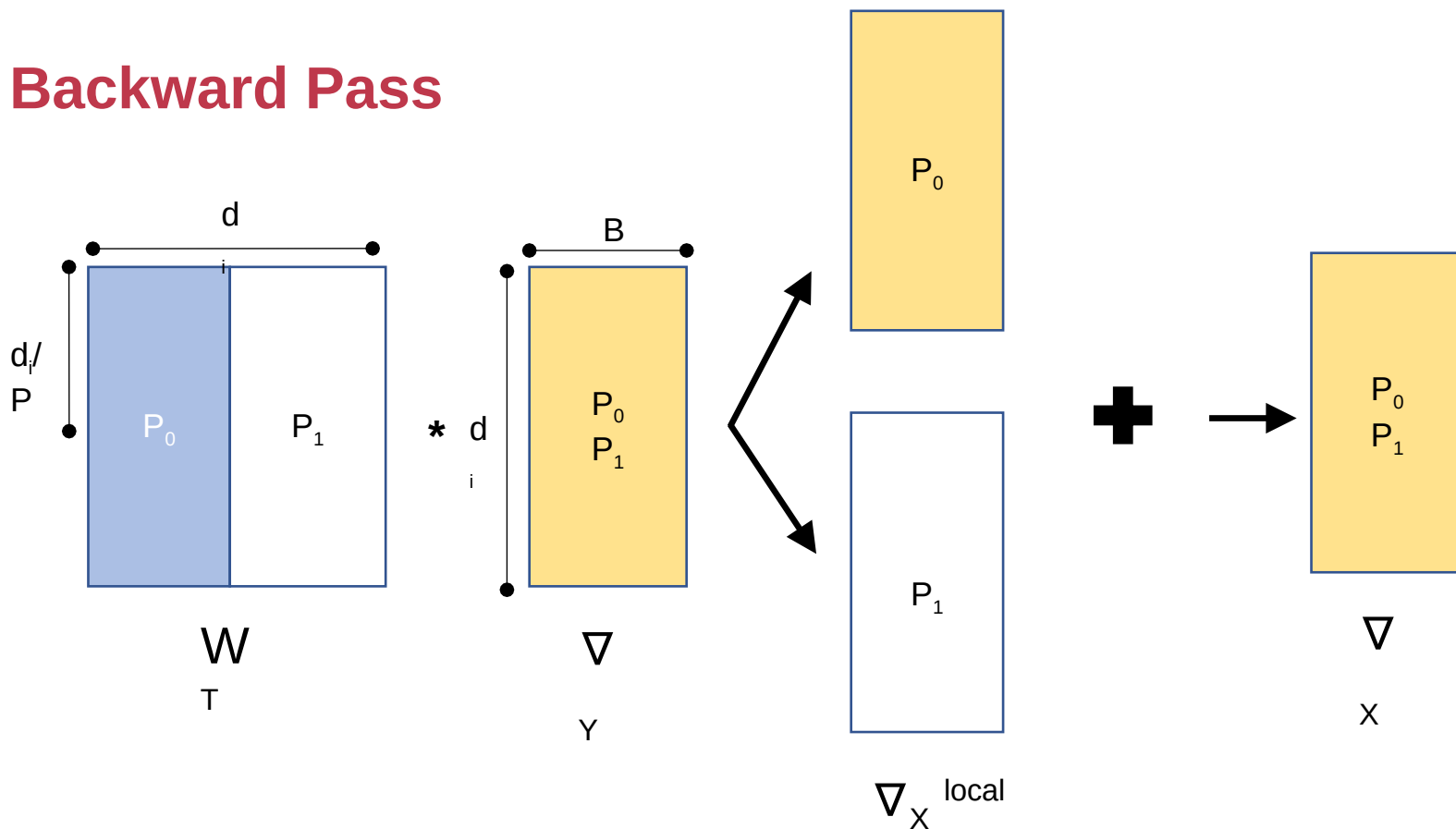
- Requires an all gather communication so that all processes get each others activation data
- Suppose with a simple forward, each node needs to send \sim

Model Parallelism: Backward Pass



No communication needed as every processor only needs the gradient of its own parameters

Backward Pass



- Aggregating input gradient requires an **allreduce** operation
- Cost (using ring):)

Tensor parallelism: communication analysis

Suppose W is d_i , and X is d_i , we have P partitions

Setup: partition the graph into P partitions w/ model parallelism

- What is the expected communication needed for a link?
- Typically, much higher than pipeline parallelism

Forward pass of tensor parallelism:

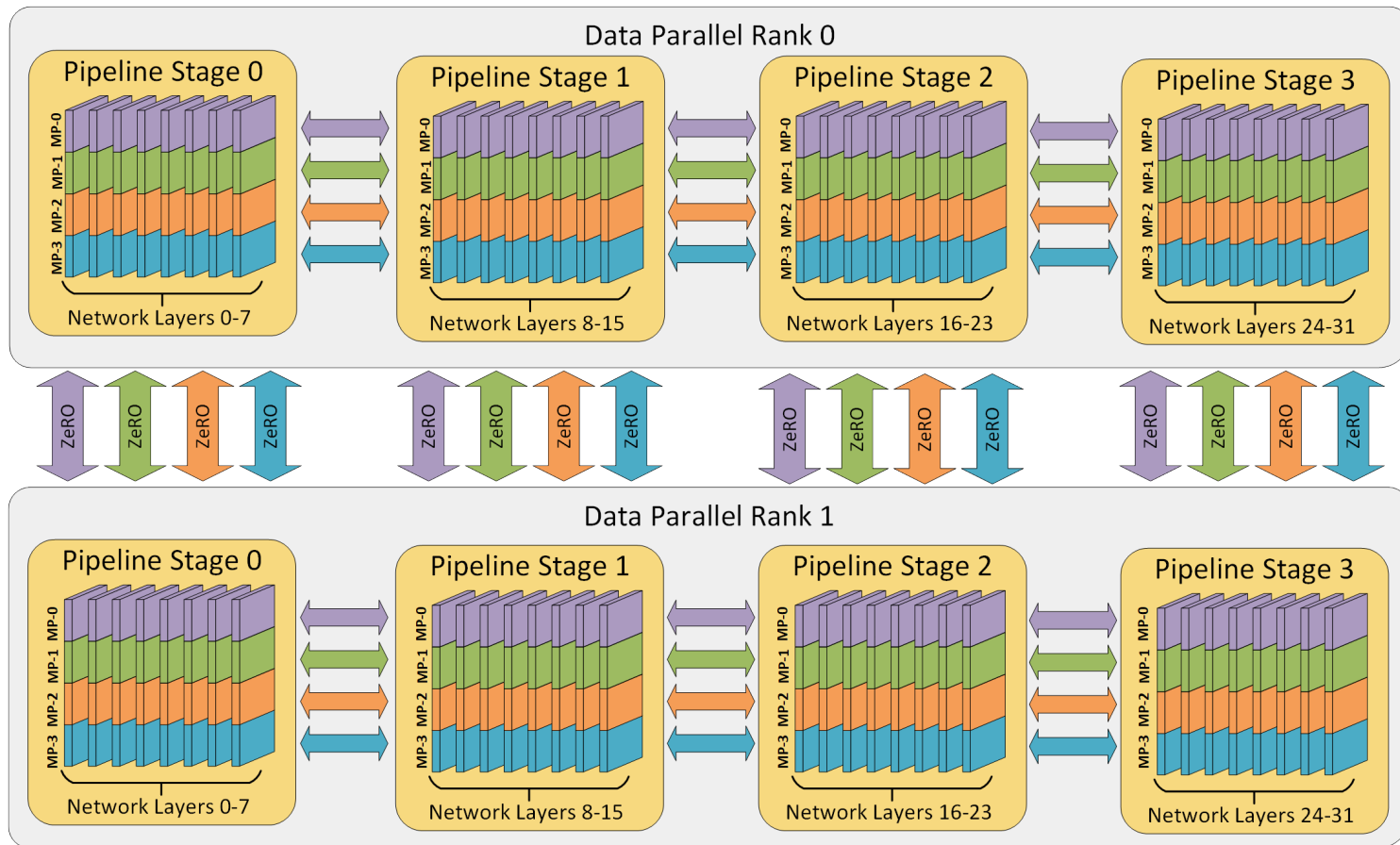
- $(d_i * B / P) * (P - 1)$ // must communicate with all the partitions

Forward pass of pipeline parallelism:

- $(d_i * B / P)$ // only needs to pass to the next stage

In general, tensor parallelism has a higher communication cost!

Put it altogether: parallelize distributed training



Summary: model parallelism

Two forms: pipeline parallelism + tensor parallelism

Pipeline parallelism

- Partition the computation graph by layers
- Pros: reduced communication
- Cons: bubbles

Tensor parallelism

- Partition the parameters of a layer (or a set of layers)
- Pros: better support for large models
- Cons: high communication cost

Case study: A100 vs. A800

The US conducted a GPU ban on China, mainly restricting GPU communication performance (some kind of out-dated)



Jensen H Huang

Jensen H Huang, Employees

11:56

Hi everyone,

The US Department of Commerce has imposed a new license requirement on A100 and H100 GPUs sold to China and Russia.

We will immediately work with our customers in China to satisfy their needs with our best alternatives or seek licenses where replacements are not sufficient. The replacement products will be worse for customer workloads that need the full performance of A100. However, for most customers, the alternative product should serve their requirements.

I realize this restriction comes during an already challenging environment.

Thank you all for doing your utmost to comply with the new license requirement and serve our customers.

I have every confidence we will get through this.

UNITED STATES
SECURITIES AND EXCHANGE COMMISSION
WASHINGTON, DC 20549

FORM 8-K

CURRENT REPORT
PURSUANT TO SECTION 13 OR 15(d) OF
THE SECURITIES EXCHANGE ACT OF 1934

Date of Report (Date of earliest event reported): August 26, 2022

NVIDIA CORPORATION
(Exact name of registrant as specified in its charter)

Delaware 0-23905 94-3177547
(State or other jurisdiction of incorporation) (Commission File Number) (IRS Employer Identification No.)

2788 San Tomas Expressway, Santa Clara, CA 95051
(Address of principal executive office) (Zip Code)

Registrant's telephone number, including area code: (408) 486-2000

Not Applicable
(Former name or former address, if changed since last report)

Check the appropriate box below if the Form 8-K filing is intended to simultaneously satisfy the filing obligation of the registrant under any of the following provisions.

☐ Written communications pursuant to Rule 425 under the Securities Act (17 CFR 230.425)

☐ Selecting material pursuant to Rule 144-12 under the Exchange Act (17 CFR 240.144-12)

☐ Pre-commencement communications pursuant to Rule 144-2(b) under the Exchange Act (17 CFR 240.144-2(b))

☐ Pre-commencement communications pursuant to Rule 136-4(c) under the Exchange Act (17 CFR 240.136-4(c))

Securities registered pursuant to Section 12(b) of the Act:

Title of each class	Trading Symbol(s)	Name of each exchange on which registered
Common Stock, \$0.01 par value per share	NVDA	The Nasdaq Global Select Market

Indicate by check mark whether the registrant is an emerging growth company as defined in Rule 405 of the Securities Act of 1933 (230.405 of this chapter) or Rule 120-2 of the Securities Exchange Act of 1934 (240.120-2 of this chapter).

Emerging Growth Company ☐

If an emerging growth company, indicate by check mark if the registrant has elected not to use the extended transition period for complying with any new or revised financial accounting standards provided pursuant to Section 13(a) of the Exchange Act: ☐

Item 8.01 Other Events

On August 26, 2022, the U.S. government, or USG, informed NVIDIA Corporation, or the Company, that the USG has imposed a new license requirement, effective immediately, for any future export to China (including Hong Kong) and Russia of the Company's A100 and forthcoming H100 integrated circuits, DGX or any other systems which incorporate A100 or H100 integrated circuits and the A100, are also covered by the new license requirement. The license requirement also includes any future NVIDIA integrated circuit achieving both peak performance and chip-to-chip I/O performance equal to or greater than thresholds that are roughly equivalent to the A100, as well as any system that includes those circuits. A license is required to export technology to support or develop covered products. The USG indicated that the new license requirement will address the risk that the covered products may be used in, or diverted to, a military end use or military end user in China and Russia. The Company does not sell products to customers in Russia.

The new license requirement may impact the Company's ability to complete its development of H100 in a timely manner or support existing customers of A100 and may require the Company to transition certain operations out of China. The Company is engaged with the USG and is seeking exemptions for the Company's internal development and support activities.

In addition, the Company is engaging with customers in China and is seeking to satisfy their planned or future purchases of the Company's Data Center products with products not subject to the new license requirement. To the extent that a customer requires products covered by the new license requirement, the Company may seek a license for the customer but has no assurance that the USG will grant any exemptions or licenses for any customer, or that the USG will act on them in a timely manner.

The Company's outlook for its third fiscal quarter provided on August 24, 2022 included approximately \$400 million in potential sales to China which may be subject to the new license requirement if customers do not want to purchase the Company's alternative product offerings or if the USG does not grant licenses in a timely manner or denies licenses to significant customers.

Certain statements in this Current Report on Form 8-K including statements regarding the Company's financial outlook for the third quarter of fiscal 2023, the Company's engagement with the USG seeking exemptions for the Company's internal development and support activities, the Company's intent to seek licenses for sales to customers of products subject to the new license requirement, and the Company's engagement with customers to satisfy their planned purchases of the Company's Data Center products with products not subject to the new license requirement are forward-looking statements that are subject to risks and uncertainties that could cause results to be materially different than expectations. Important factors that could cause such results to differ materially include: changes in consumer preferences or demands; as well as other factors detailed from time to time in our 15-K and quarterly reports on Form 10-Q. Copies of our annual and quarterly reports of financial performance and other information are available on our website.

Case study: A100 vs. A800

	A100 80GB PCIe	A100 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS 624 TFLOPS*	
INT8 Tensor Core	624 TOPS 1248 TOPS*	
GPU 显存	80GB HBM2	80GB HBM2e
GPU 显存带宽	1935 GB/s	2039 GB/s
最大热设计功耗 (TDP)	300W	400W ***
多实例 GPU	最大为 7 MIG @ 5GB	最大为 7 MIG @ 10GB
外形规格	PCIe 双插槽风冷式或单插槽液冷式	SXM
互连	NVIDIA® NVLink® 桥接器 2 块 GPU: 600 GB/s ** PCIe 4.0: 64 GB/s	NVLink: 600 GB/s PCIe 4.0: 64 GB/s
服务器选项	合作伙伴及配备 1 至 8 个 GPU 的 NVIDIA 认证系统™	NVIDIA HGX™ A100 合作伙伴和配备 4、8 或 16 块 GPU 的 NVIDIA 认证系统 配备 8 块 GPU 的 NVIDIA DGX™ A100

	A800 80GB PCIe	A800 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS 624 TFLOPS*	
INT8 Tensor Core	624 TOPS 1248 TOPS*	
GPU 显存	80GB HBM2e	80GB HBM2e
GPU 显存带宽	1935GB/s	2039GB/s
最大热设计功耗 (TDP)	300W	400W***
多实例 GPU	最多 7 个 MIG 每个 10GB	最多 7 个 MIG 每个 10GB
外形规格	PCIe (双插槽风冷式或单插槽液冷式)	SXM
互连技术	搭载 2 个 GPU 的 NVIDIA® NVLink® 桥接器: 400GB/s ** PCIe 4.0: 64GB/s	NVLink: 400GB/s PCIe 4.0: 64GB/s

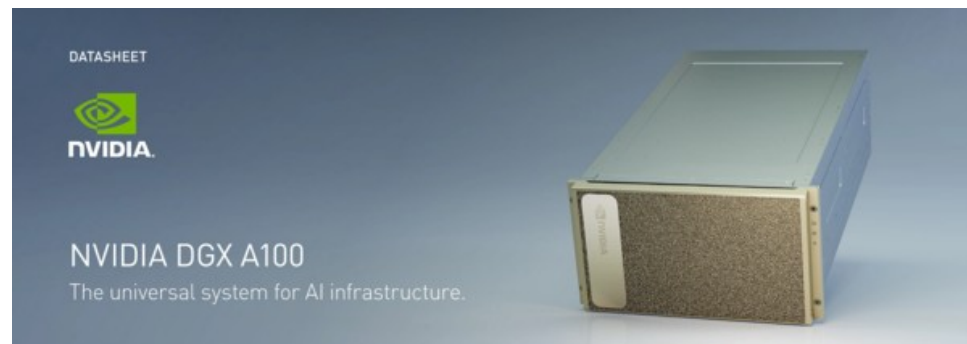
Case study: Distributed training on DGX A100 (8 X A100 server)

Key components

- 8 X A100 GPUs
- Up to 8 X ConnectX-6/7 NIC (50/100/200 Gbps, unidirectional)

GPUs are connected via NVLink

- 600GBps (bi-directional) w/ A100
- 400GBps (bi-directional) w/ A800



The Challenge of Scaling Enterprise AI

Every business needs to transform using artificial intelligence (AI), not only to survive, but to thrive in challenging times. However, the enterprise requires a platform for AI infrastructure that improves upon traditional approaches, which historically involved slow compute architectures that were siloed by analytics, training, and inference workloads. The old approach created complexity, drove up costs, constrained speed of scale, and was not ready for modern AI. Enterprises, developers, data scientists, and researchers need a new platform that unifies all AI workloads, simplifying infrastructure and accelerating ROI.

The Universal System for Every AI Workload

Part of the NVIDIA DGX™ platform, NVIDIA DGX A100 is the universal system for all AI workloads—from analytics to training to inference. DGX A100 sets a new bar for compute density, packing 5 petaFLOPS of AI performance into a 6U form factor, replacing legacy compute infrastructure with a single, unified system. DGX A100 also offers the unprecedented ability to deliver a fine-grained allocation of computing power, using the Multi-Instance GPU (MIG) capability in the NVIDIA A100 Tensor Core GPU. This enables administrators to assign resources that are right-sized for specific workloads.

Available with 320 gigabytes (GB) and 640 gigabytes of total GPU memory, DGX A100 can tackle the largest and most complex jobs, along with the simplest and smallest. DGX A100 640GB increases performance in large-scale training jobs up to 3X and doubles the size of MIG instances. This combination of dense compute power and complete workload flexibility makes DGX A100 an ideal choice for both single and scaled DGX deployments. Every DGX system is powered by **NVIDIA Base Command™** for managing single node as well as large-scale Surtm or Kubernetes clusters, and includes the NVIDIA AI Enterprise suite of optimized AI software containers.

Unmatched Level of Support and Expertise

NVIDIA DGX A100 is more than a server. It's a complete hardware and software platform built upon the knowledge gained from the world's largest DGX proving ground—NVIDIA DGX SATURNV—and backed by thousands of DGXperts at NVIDIA. DGXperts are AI-fluent practitioners who have built a wealth of know-how and experience over the last decade to help maximize the use of NVIDIA DGX. DGXperts also provide the critical production

SYSTEM SPECIFICATIONS

NVIDIA DGX A100 320GB	
GPUs	8x NVIDIA A100 40GB Tensor Core GPUs
GPU Memory	320 GB total
Performance	5 petaFLOPS AI 10 petaOPS INT8
NVIDIA NVSwitches	6
System Power Usage	6.5 kW max
CPU	Dual AMD Rome 7742, 128 cores total, 2.25 GHz (base), 3.4 GHz (max boost)

System Memory	
Networking	Up to 8x Single-Port NVIDIA ConnectX-7 200 Gb/s InfiniBand 1 Dual-Port ConnectX-7 VPI 10/25/50/100/200 Gb/s Ethernet
Storage	OS: 2x 1.92TB M.2 NVMe drives Internal Storage: 15TB (4x 3.84 TB) U.2 NVMe drives

Software	DGX OS / Ubuntu / Red Hat Enterprise Linux / Rocky – Operating System NVIDIA Base Command – Orchestration, scheduling, and cluster management NVIDIA AI Enterprise – Optimized AI software
----------	--

Support	Comes with 3-year business-standard hardware and software support
---------	---

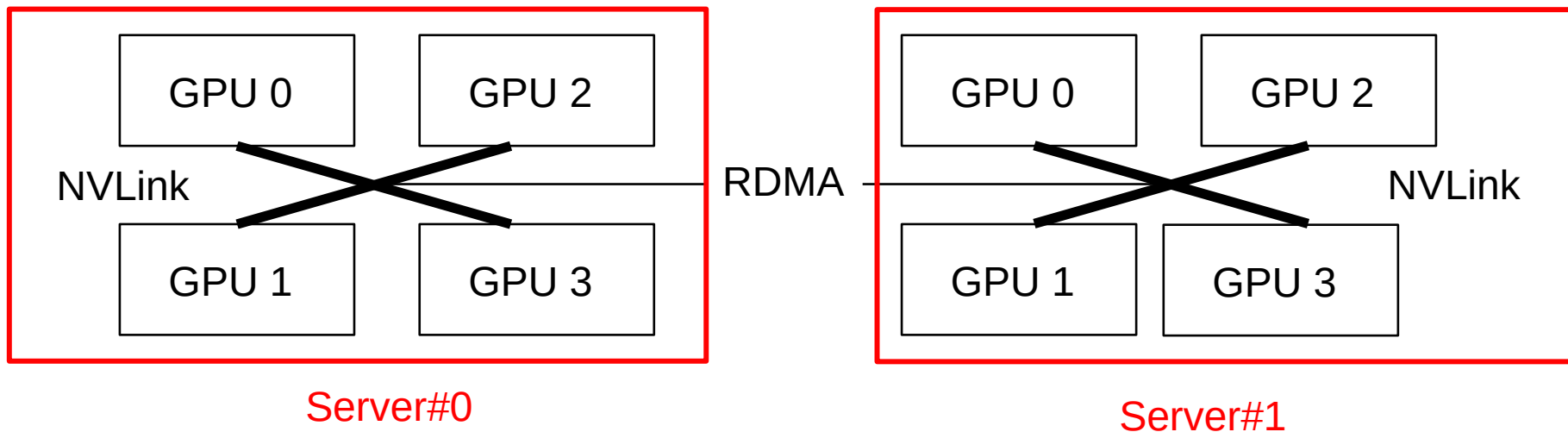
System Weight	271.5 lbs (123.16 kgs) max
---------------	----------------------------

Packaged System Weight	359.7 lbs (163.16 kgs) max
------------------------	----------------------------

System Dimensions	Height: 10.4 in (264.0 mm) Width: 19.0 in (482.3 mm) max Length: 35.3 in (897.1 mm) max
-------------------	---

Operating	5–30 °C (41–86 °F)
-----------	--------------------

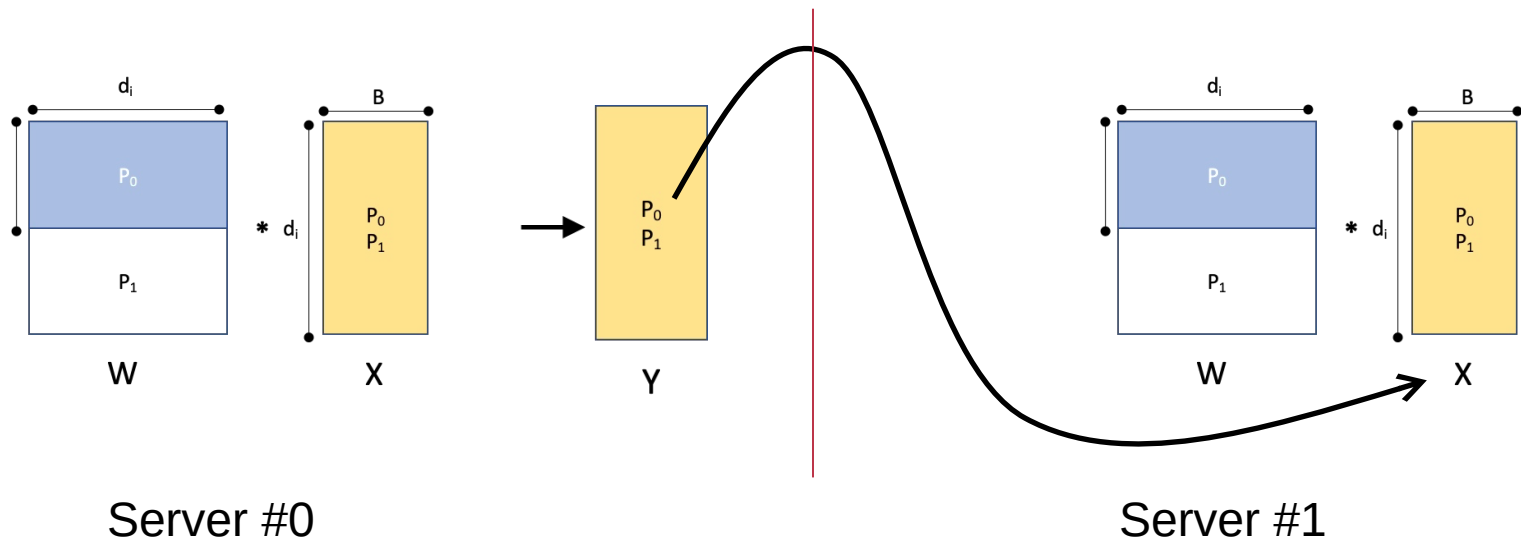
Thought experiment: why 600 -> 400GB/s?



Interconnect setup

- Between servers, up to 8 X 200Gbps ConnectX-6/7 (200GBps, uni-directional); however, in real evaluation, **typically measured 160GBps data (why?)**
- Between GPUs: 600/400 GBps NVLink (bi-directional) = 300/200 GBps NVLink (uni-directional), can **get the full speed** (empirically)

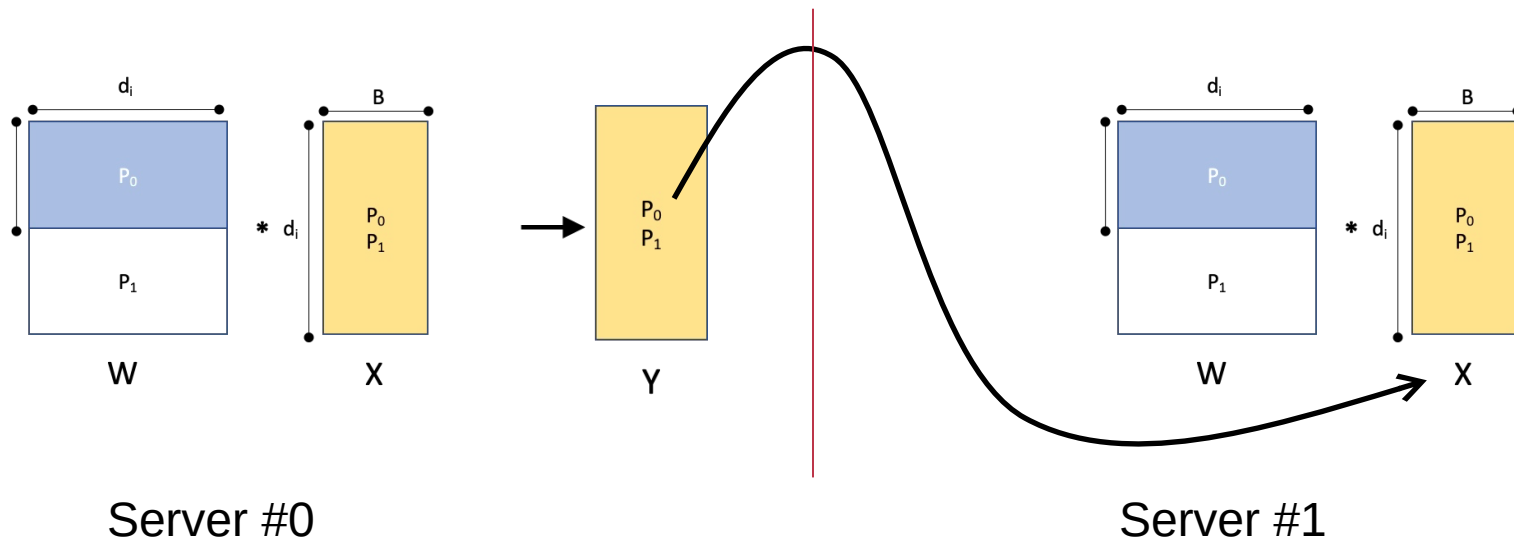
Thought experiment: why 600 -> 400GB/s?



Setup: pipeline + tensor parallelism

- Two servers, each server: 8 GPUs (e.g., DGX A100)
- Within each server, tensor parallelism (model partitioned on 8 GPUs)
- Between servers, pipeline parallelism

Thought experiment: why 600 -> 400GB/s?



Communication analysis (Backward path, assuming 8 GPUs)

- Between machine: bytes sent per iteration
- Between GPU: bytes sent per iteration

The bandwidth of NVLink should be 1.75X larger to prevent being the bottleneck

Thought experiment: why 600 -> 400GB/s?

Goal: NVLink should be 1.75X faster than RDMA

- To prevent becoming the communication bottleneck in our pipeline—tensor hybrid parallelism

A100 vs. RDMA

–

A800 vs. RDMA

–

With A800: the model parallelism will be bottlenecked by the NVLink !

Parallelization Opportunities

Data Parallelism: Distribute the processing of data to multiple PEs.

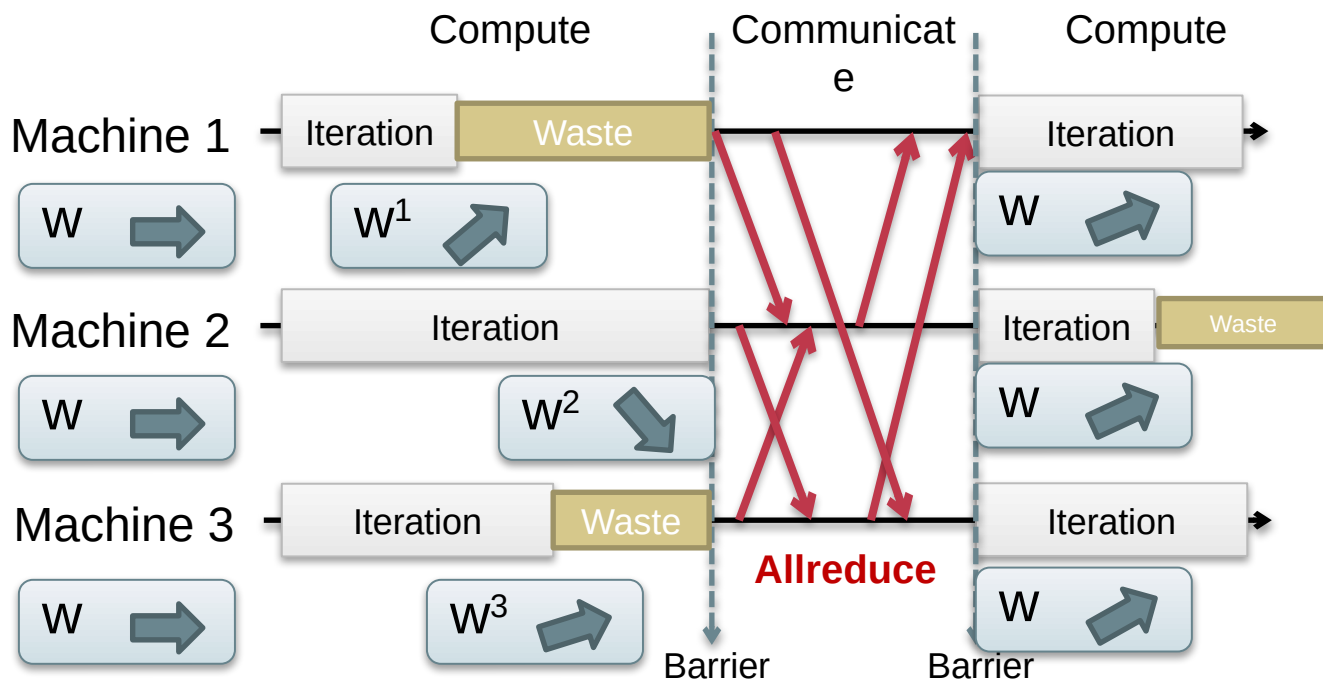
Model Parallelism: Break the model and distribute processing of every layer to multiple PEs

For either approach it is also possible to use **synchronous** or **asynchronous** updates

Recall: BSP for data parallel execution

Problem: may have idle time at each process

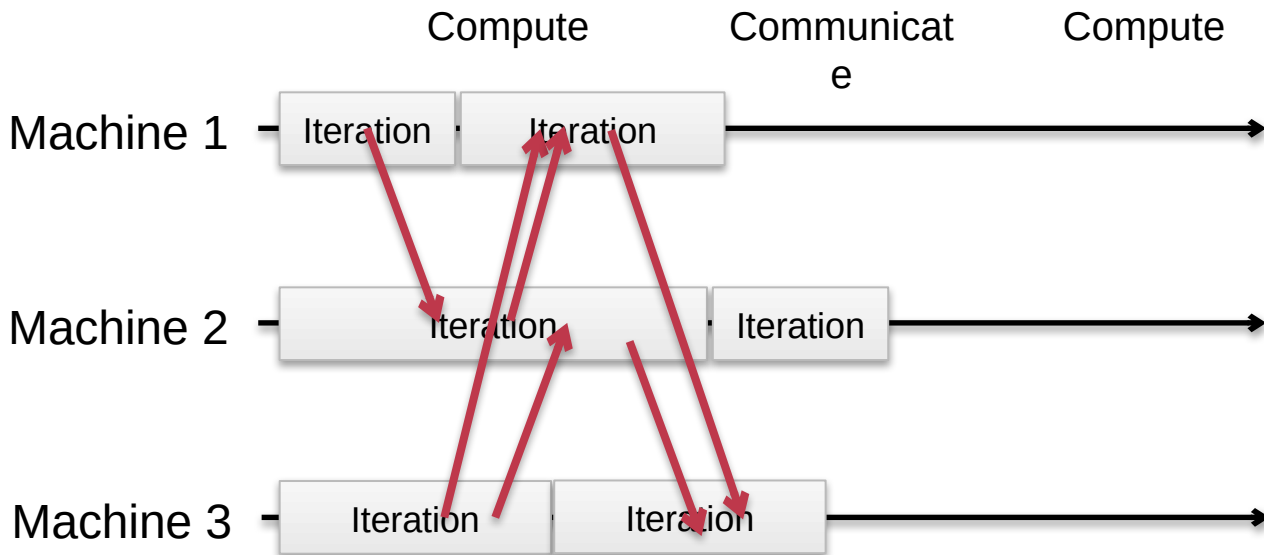
- Bulk Synchronous Parallel (BSP)



Asynchronous Execution

Key idea: each process does not wait for others

- Update the gradients upon receive a messages, no iteration barrier



Async vs. Sync execution

Async

- Pros: efficiency
- Cons: trade accuracy for performance

Sync

- Pros: Preserve the SGD training property
- Cons: poor performance

Which is better?

- Hard to tell. But, as a system designer, preserving the algorithms property is **very important**. Any design that alter the training property should be justified

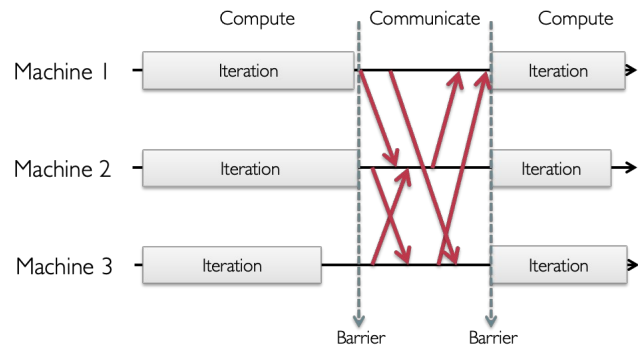
Common: nearly all training currently adopts a synchronous approach

Fault tolerance in distributed training

// execute on a master

```
for iter in num_iters:
```

```
    iter+1 iter
```



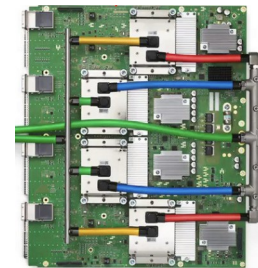
Really simple due to the BSP nature

- Checkpoint the W (& optimization states) per-iteration
- Or per-several iterations to reduce overheads
 - GPUs fail not so often: 40 minutes

Review of (distributed) & parallel computing

Review: Devices available for computation

Programmability ←



Intel i5-9600K,
single core:
6.3GFLOPS

Mate60 GPU
2.3 TFLOPs

Apple M2
GPU 3.6 TFLOPs

NVIDIA A100
GPU 19.5 TFLOPs

Google TPUv4
275 TFLOPS

Multiple
cores:
37.7GFLOPS

→ Performance

Scaling a single-device with parallelism

General methods

- Single core+: pipeline + super scalar with instruction level parallelism (ILP)
- Single core++: added SIMD support
- Single core+++: domain-specific accelerators (e.g., matrix accelerators)
- Multiple core: a single core (single core, single core+, single core++, single core+++) can be glued together !

Question

- What are the trade-offs?

Review: the roofline model

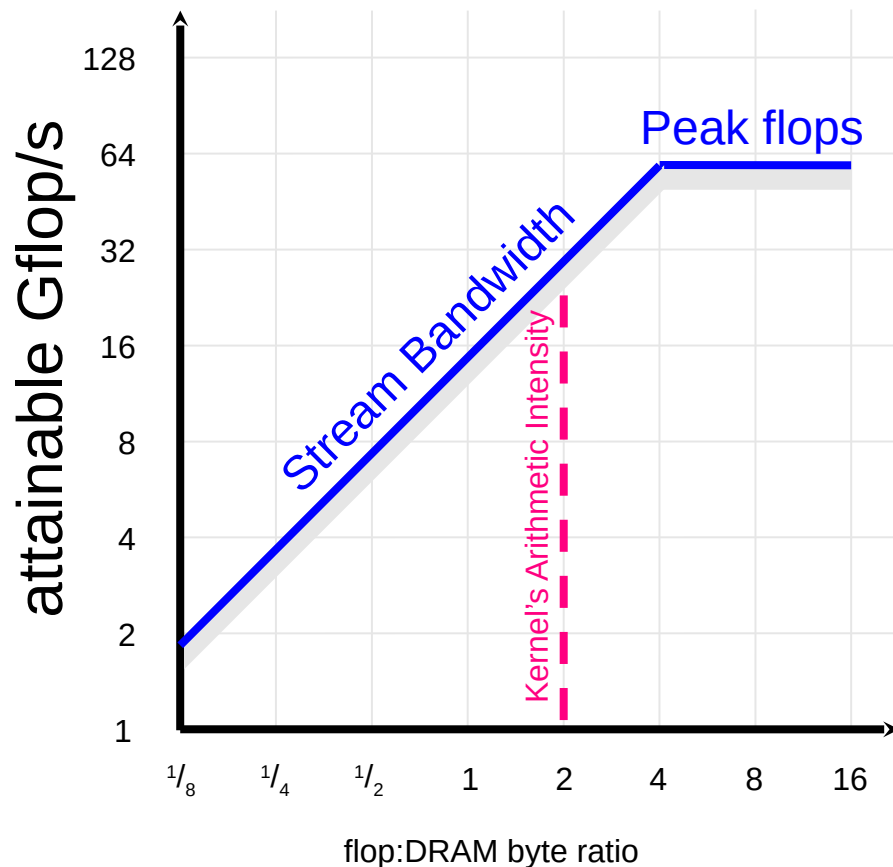
A computation task should look at capabilities other than the computation

Given an application

- If we know how many FLOPs performed per-memory read, we can see whether it is computation bound or memory bound

Benefit

- Give hints on the optimization directions



Review: Why distributed computing?

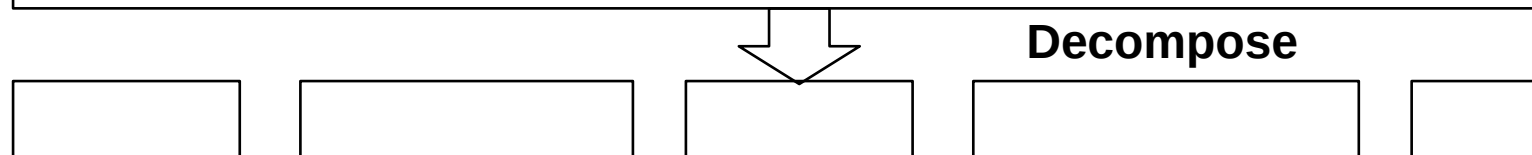
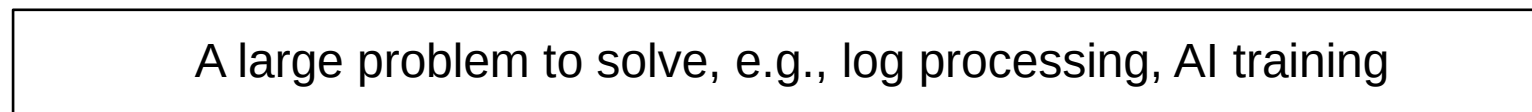
Large computation requirements

- E.g., training AI models, huge floating points required for each iteration
= $6 * \text{\#parameters} * \text{Batch size}$
- The trend continues

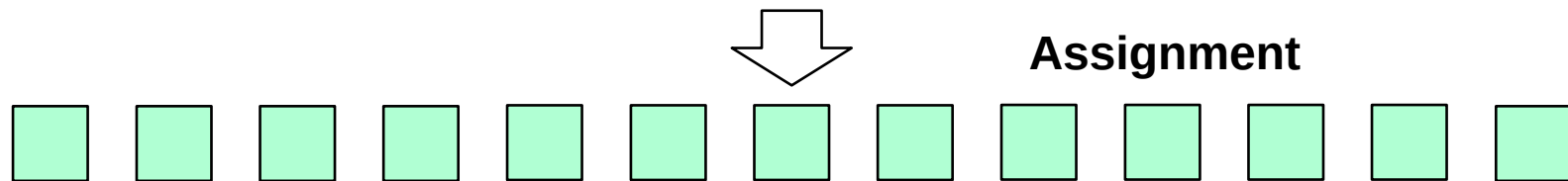
Single device FLOPS (floating points per second) is insufficient

- 2X speed improvements per-year
- Training computation required: 240+ X required per-year

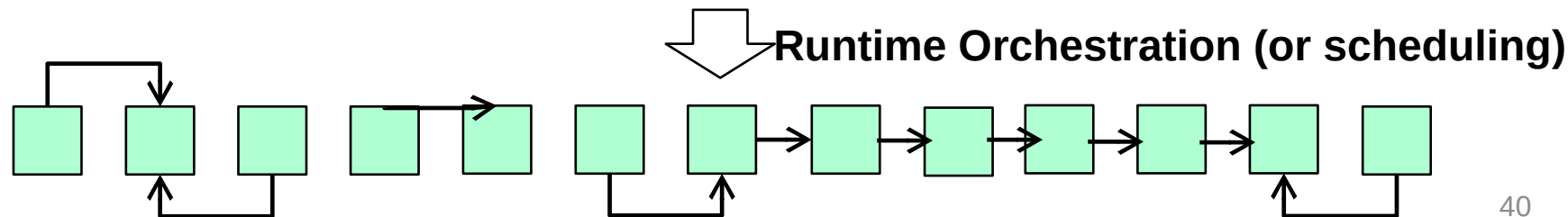
Logical process for doing a computation distributedly



Sub-problems. Many alias, e.g., sub-tasks, sub-jobs (or simply jobs)



A set of physical programs that can run on parallel units, e.g., a thread or a GPU kernel

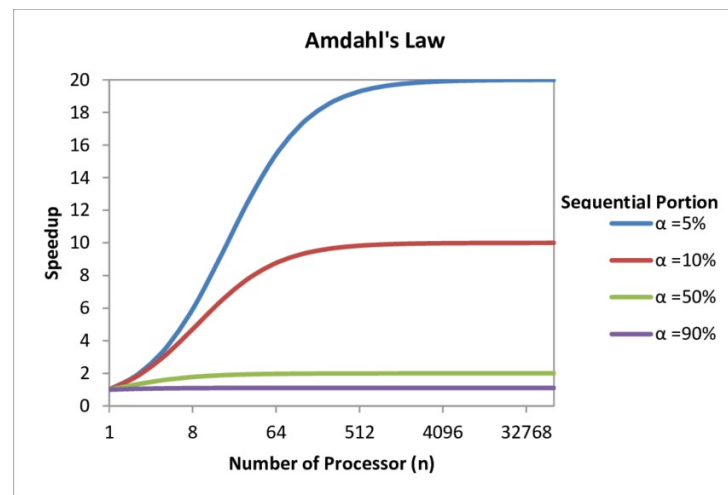


Review: Amdahl's Law for decomposing the tasks

Let α = the fraction of total work that is inherently sequential

Max speedup on M machines given by: Speedup

Takeaway: we should reduce sequential part during distributed computation



More implementation details to consider

1. Sending **data** to/from nodes
2. **Coordinating** among nodes
3. Recovering from node **failure**
4. Optimizing for **locality**
5. Partition data to to enable more **parallelism**

Common to many jobs! E.g., batch processing, graph processing, machine learning, etc.

Review: we build a framework to hide the above tasks

Goal

- Reduce programmer's effort in solving the above challenges

Challenges

- What are the abstraction? Thread & RPC is insufficient
- More general, harder to provide the above property
- E.g., if a thread fails, how can we know its progress?
 - It's a very hard problem, we will come back to it later

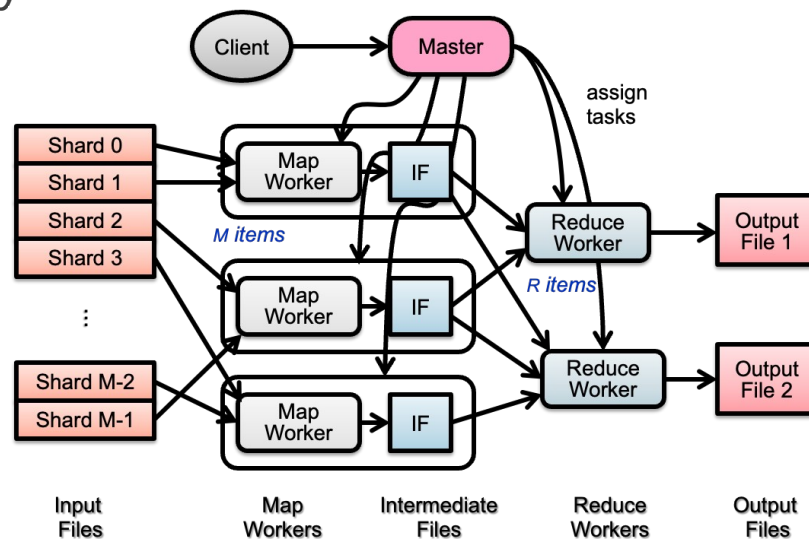
Map & Reduce abstraction in MapReduce

Map(input) -> [k,v]

- No f: the f is inside the Map itself (no high order function)
- A map is the same within the lifecycle of a MapReduce calculation

Reduce(k, values)

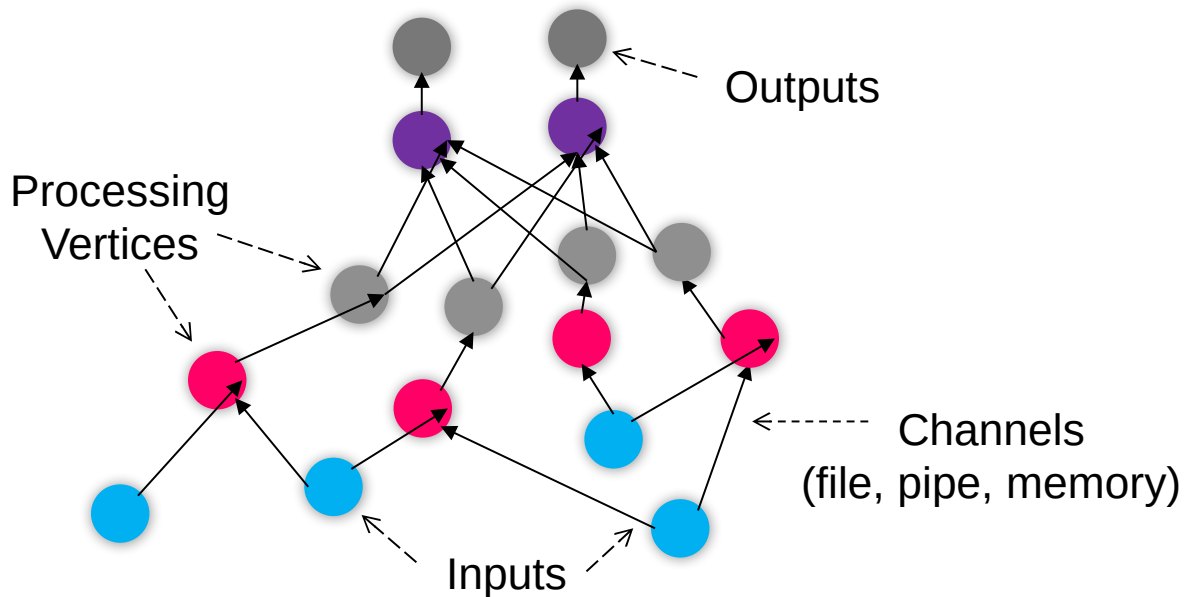
- Reduce on different keys can do parallelly



Review: The computation graph abstraction (DAG)

Computations are expressed as a **graph (Directly acrylic graph)**

- Vertices are computations (or data)
- Edges are communication channels
- Each vertex has several input and output edges



MapReduce vs. DAG

1. Sending **data** to/from nodes (both done)
2. **Coordinating** among nodes (both done)
3. Recovering from node **failure** (both done)
4. Optimizing for **locality** (both done)
5. Partition data to to enable more **parallelism** ?
 1. MapReduce: jobs are naturally parallelized w/ its abstraction
 2. But in DAG, the level of parallelism depends on the structure of DAG!

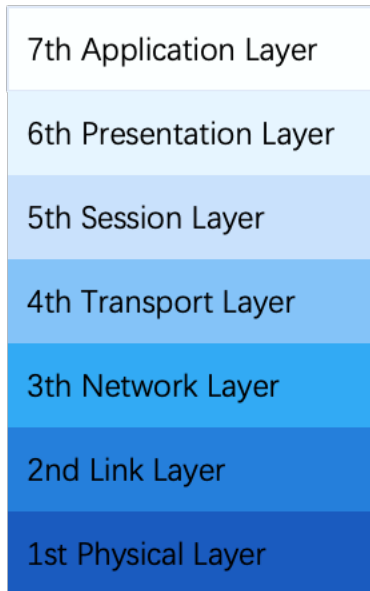
We need some domain-specific knowledge to help partition the graph node for more parallelism in DAG

Advanced topic: modern networking
for distributed computing

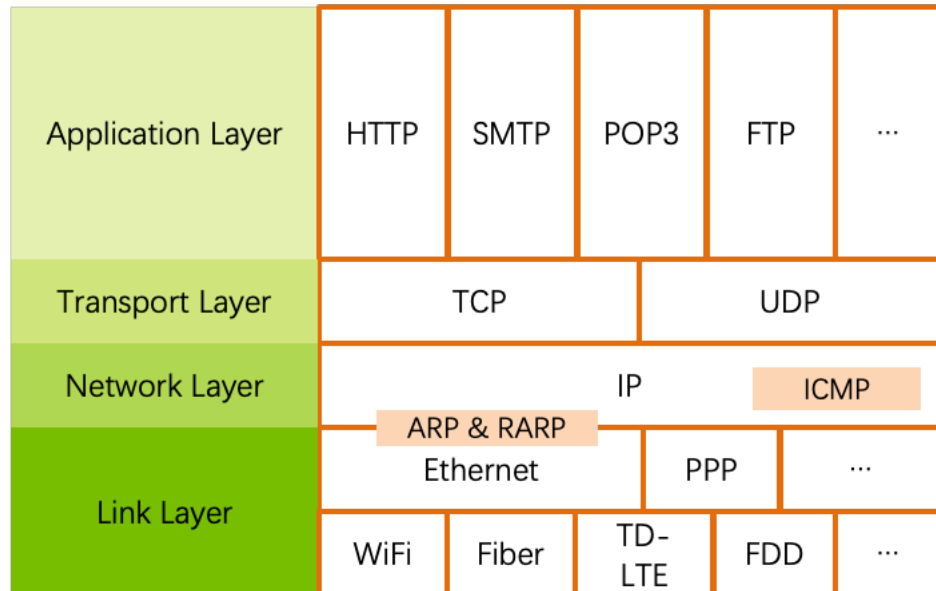
Review: traditional network stack

OSI

- The open systems interconnection (OSI) model
- 7-layer architecture



OSI



TCP/IP

What is the abstraction for networking?

Linux provides simple socket interfaces

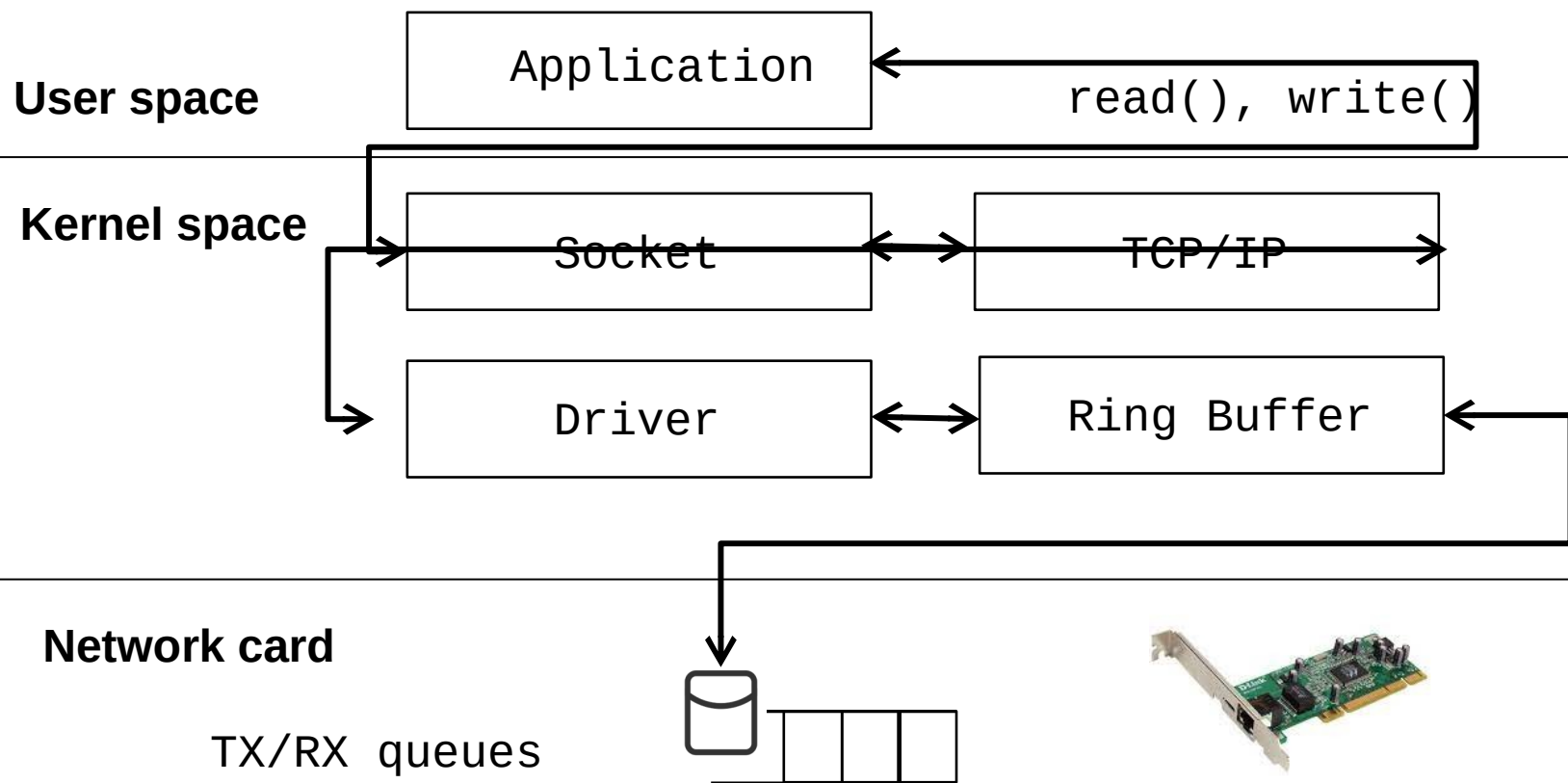
- `socket()`
- `bind()`
- `listen()`
- `connect()`
- `accept()`
- `read()`
- `write()`

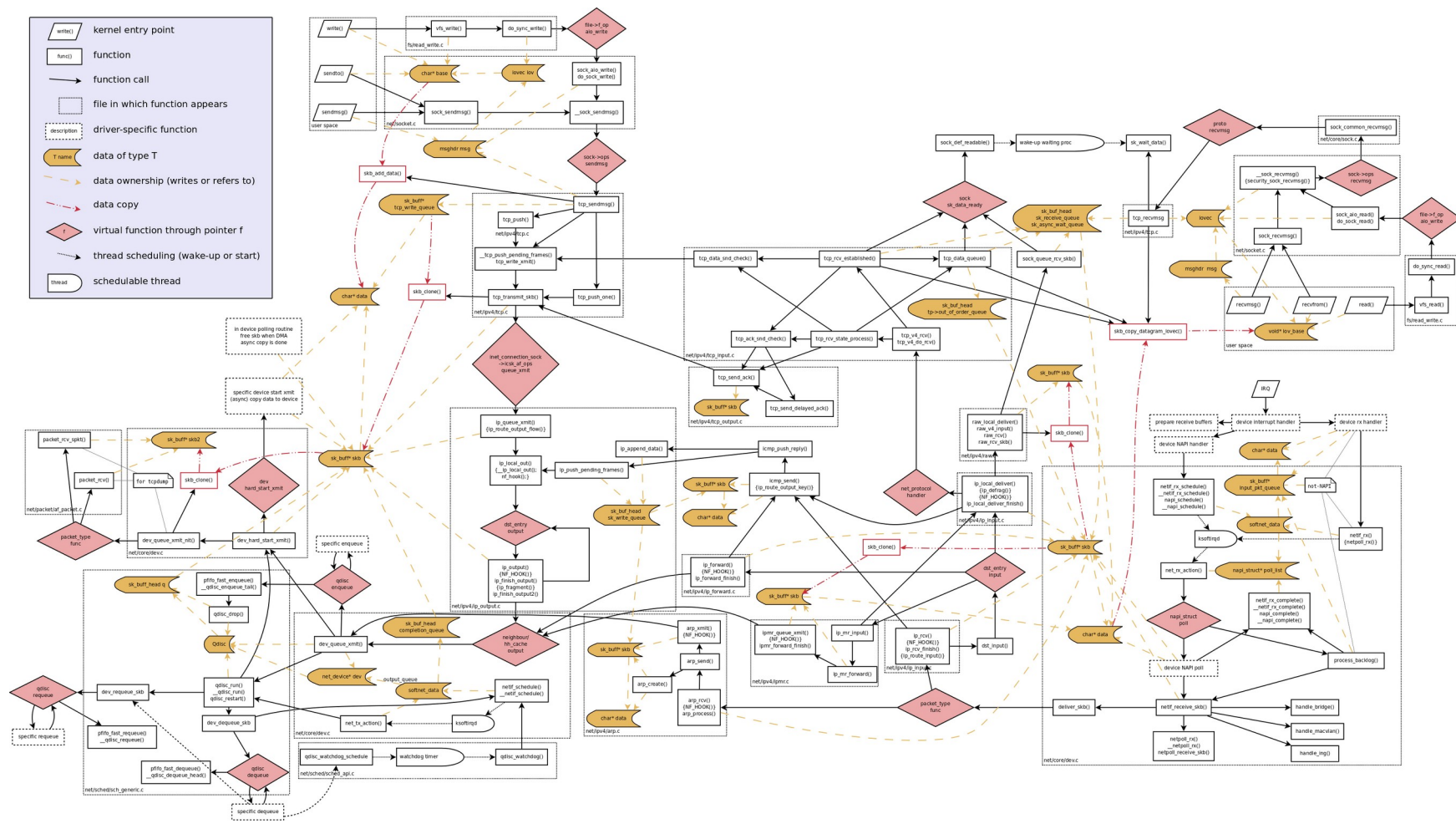
Does the software interfaces
that simple as it seems?

Can be a little complex if want high-performance

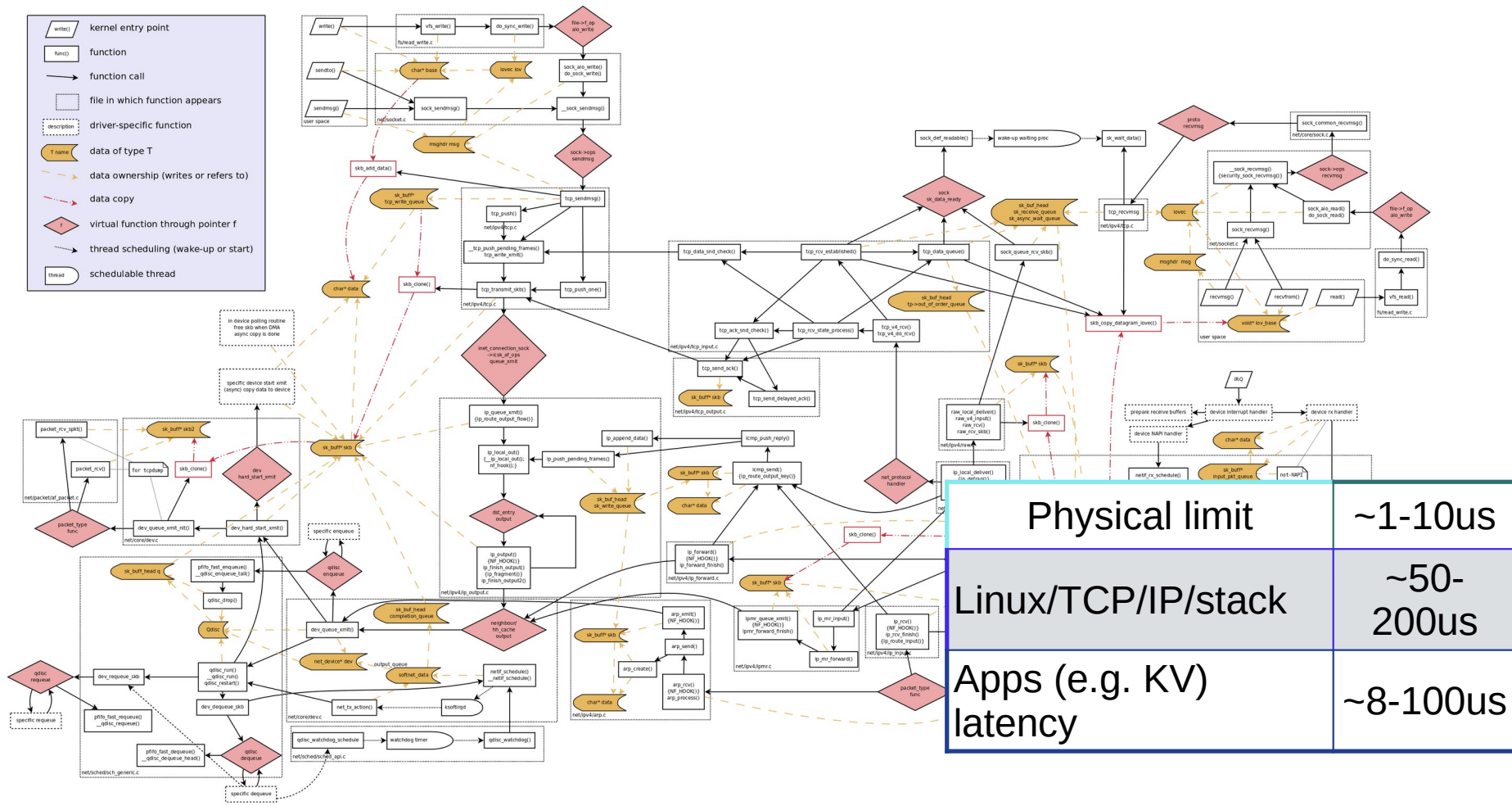
- `epoll()`, etc

Network processing in (traditional) Linux kernel



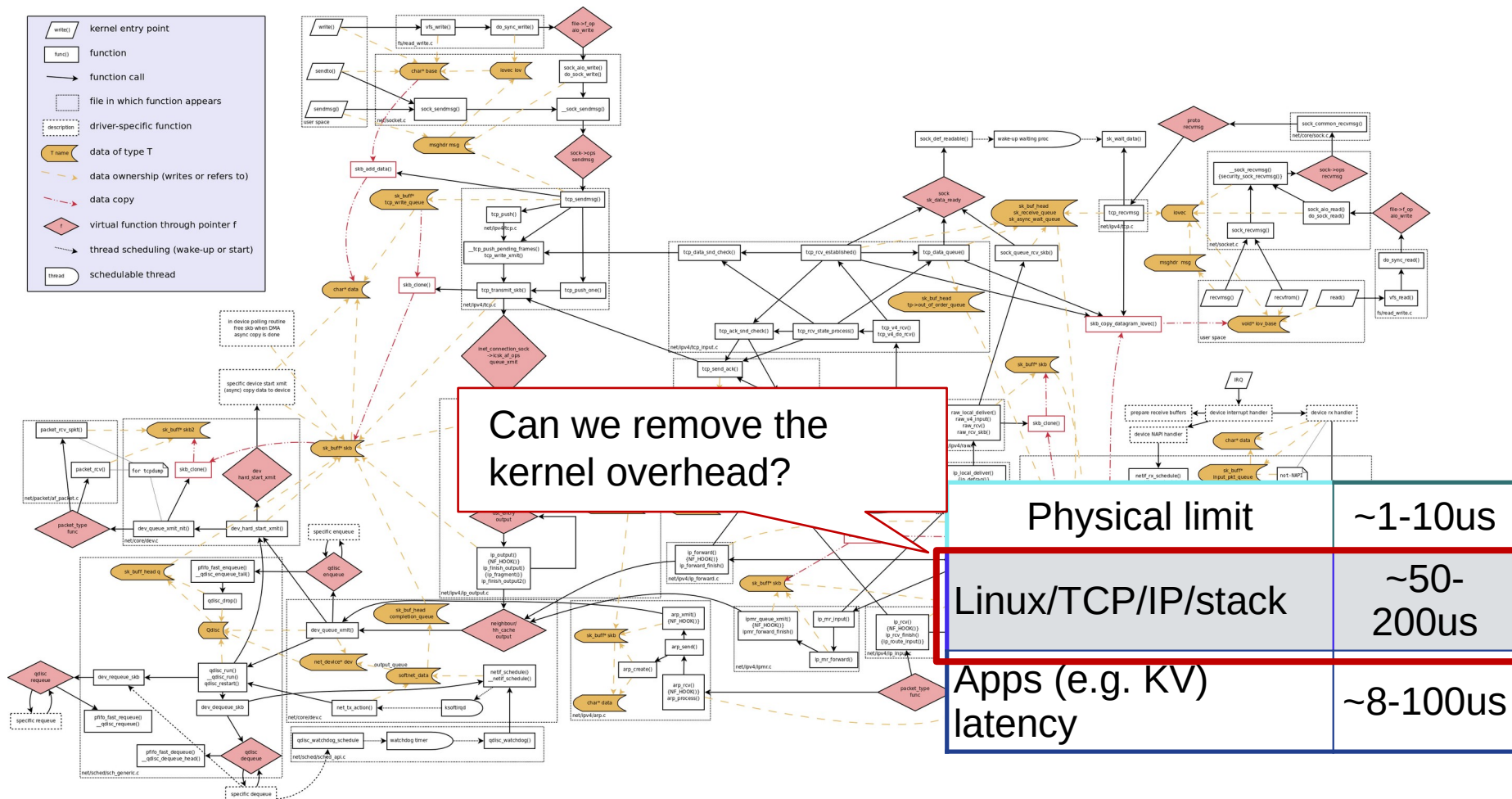


What are the costs of such a complex networking stack?



Maybe work for 10Gbps network
But what about 100Gbps?

What are the costs of such a complex networking stack?



Kernel-bypassing network



DPDK

- Set of **user-space** driver library to use the network card

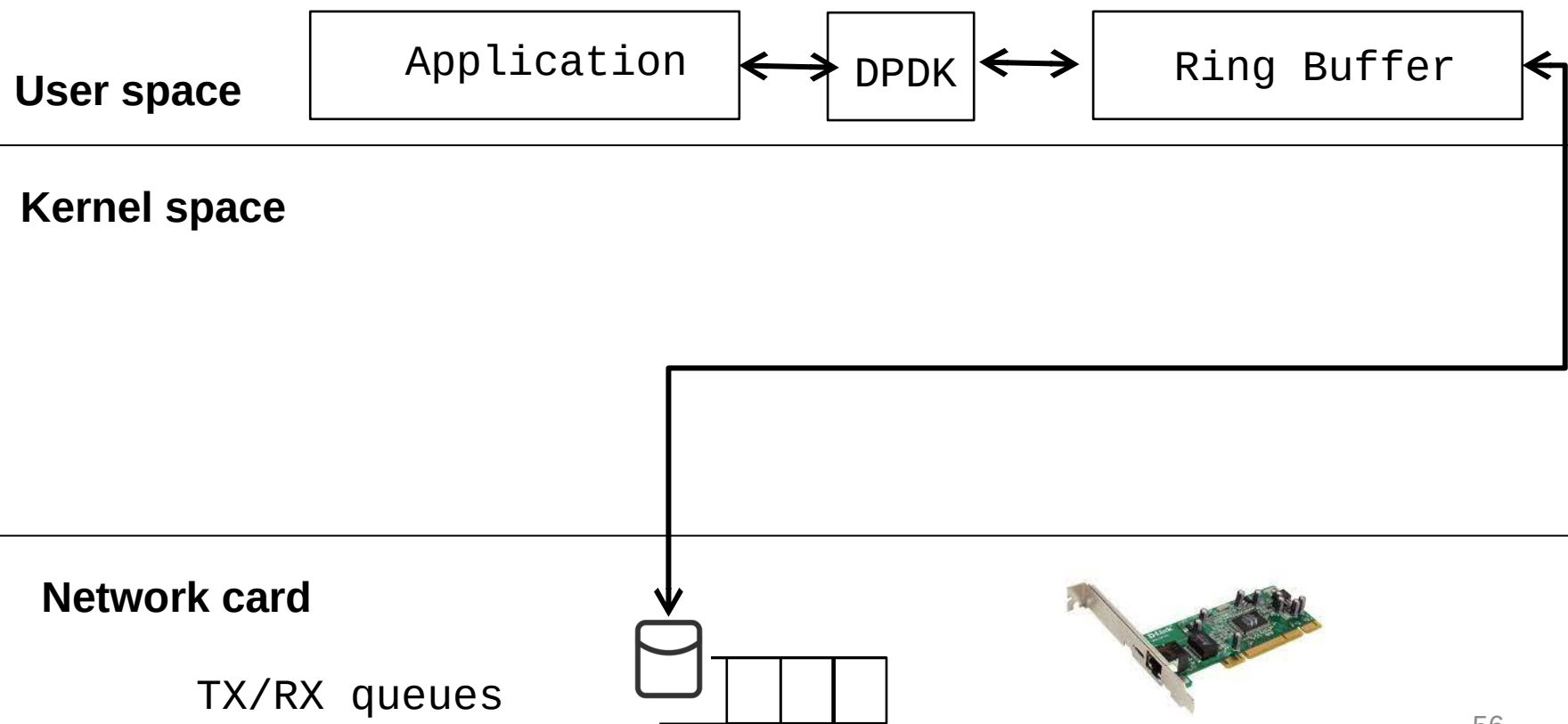
DPDK directly exposes network queues to the user-space

- Forward network packets to/from NIC from/to user applications
 - Without any kernel networking stack!

How to achieve so?

- OS will map the NIC's TX (transmit queue) and RX (receive queue) to the user-land

Packet processing with DPDK



Pros & Cons of DPDK

Pros

1. Bypasses the kernel
2. Reduced memcpy overhead
3. User can customize the networking, e.g., not using TCP/IP

E.g., co-design key-value store w/ DPDK (MICA@SIGCOMM'14)

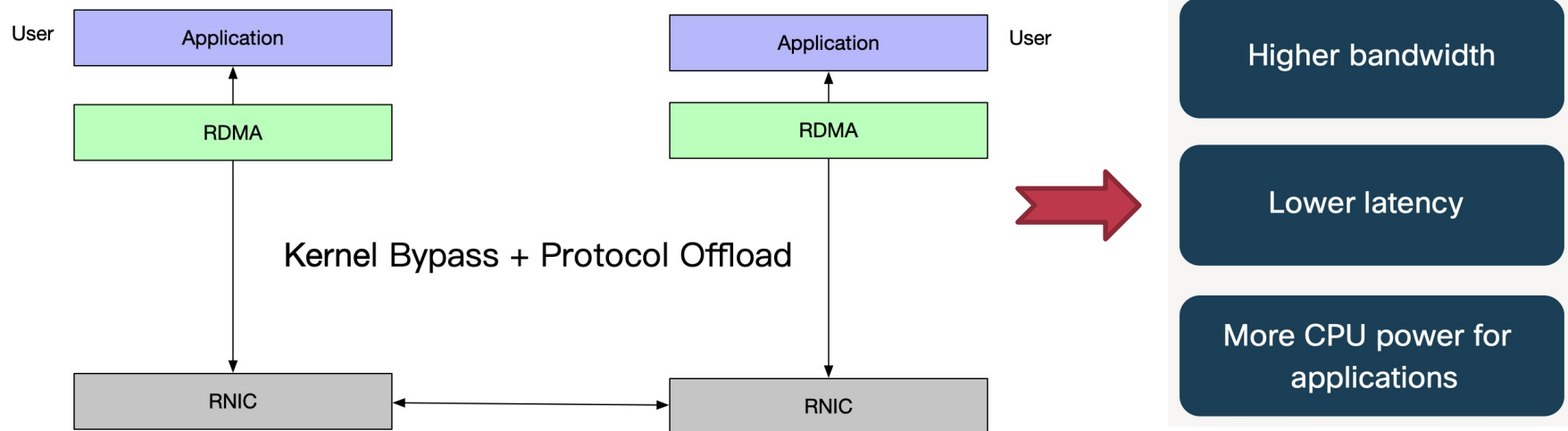
Cons

- Not compatible to traditional networking
- Still need a transport layer at the user applications, e.g., mTCP
 - Which has non-trivial overheads

Can we put the networking stack into the hardware?

RDMA: offload the complete network stack to the NIC

RDMA = Remote Direct Memory Access



Direct memory-to-memory data communication over network.

Why not offloading TCP to the hardware?

There exists proposals, e.g., TCP offloading (ToE), but typically less-efficient

RDMA is protocol designed for offloading to hardware

- Reduce complexities to improve performance (and easy-of-offloading)

Bottom-up vs. Top-down approach

- RDMA: bottom-up to find which protocol is easier to implement in the hardware
- ToE: top-down finds how to offload the complete TCP/IP protocol to the hardware

Programming w/ RDMA (verbs)

High-level concepts of RDMA

Queue Pairs (Send Queue, Complete Queue and Receive Queue)

- SQ, CQ & RQ
- A group of queues form an RDMA connection
 - Analog to connections in TCP/IP

User posts networking requests to the SQ

- Two-sided: **message datagram** (like send/recv)
- One-sided: remote read/write (will be talked about later)

Get the completion events of the send requests from the CQ

Receive two-sided messages from the RQ

Queue Pairs have different models

Reliable connected (RC)

- Like TCP/IP socket, provides reliable communications, supports all RDMA requests

Unreliable datagram (UD)

- Like UDP socket
- No reliability guarantees, limited RDMA requests type supported

Others

- Less commonly used

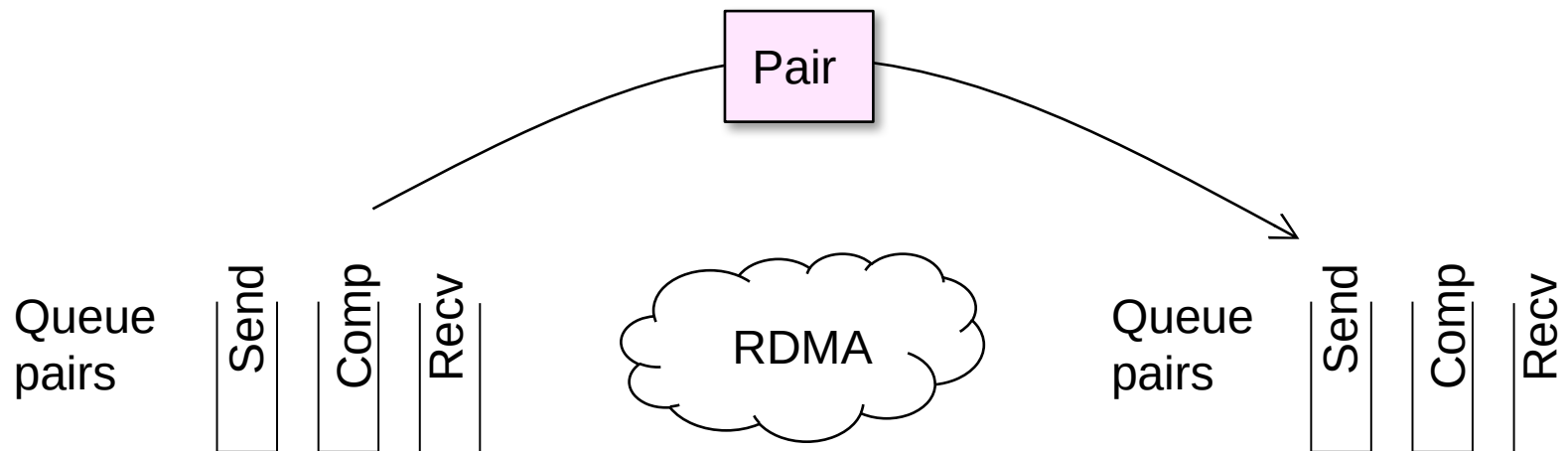
We focus on RC queue pair in this lecture

Reliable connection Queue Pairs (QP)s

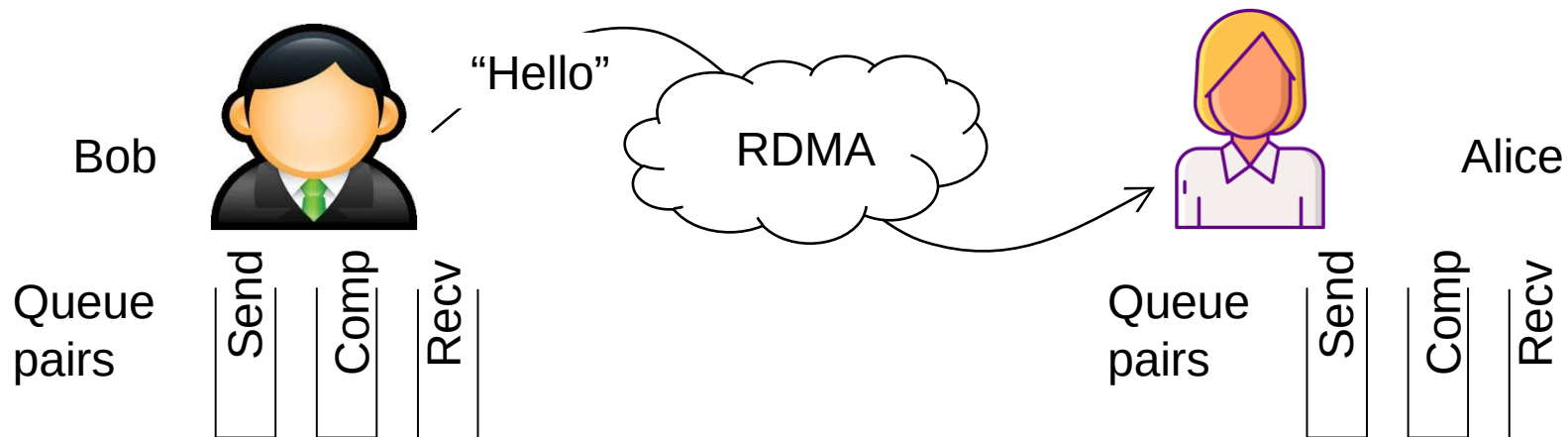
Two QP must first establish a connection via handshake so as to proceed\

- Similar to handshakes in TCP/IP
- We will talk about the connection establishment later

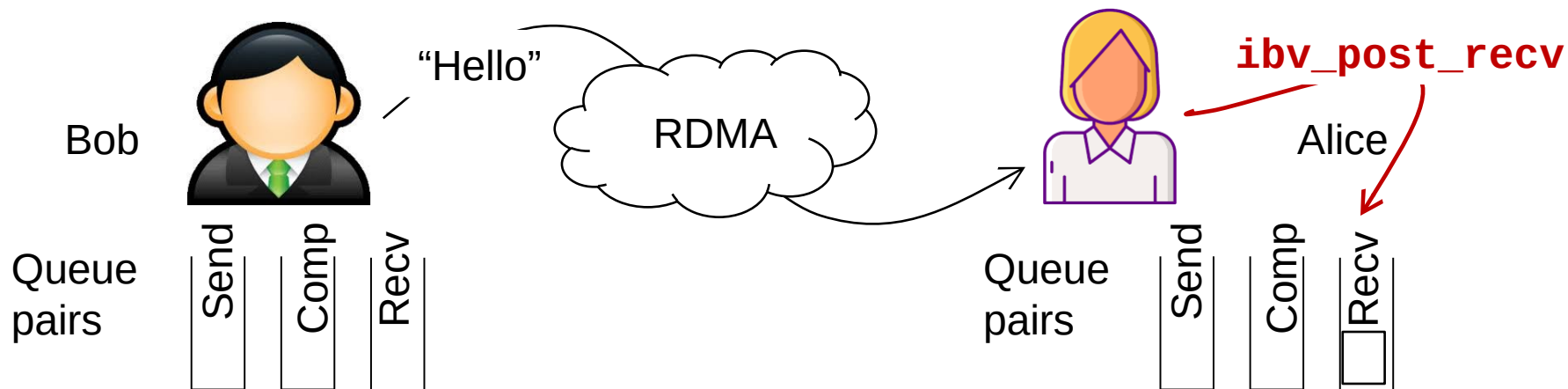
An reliable connection QP must be only connected to one other



Example: send message with RDMA



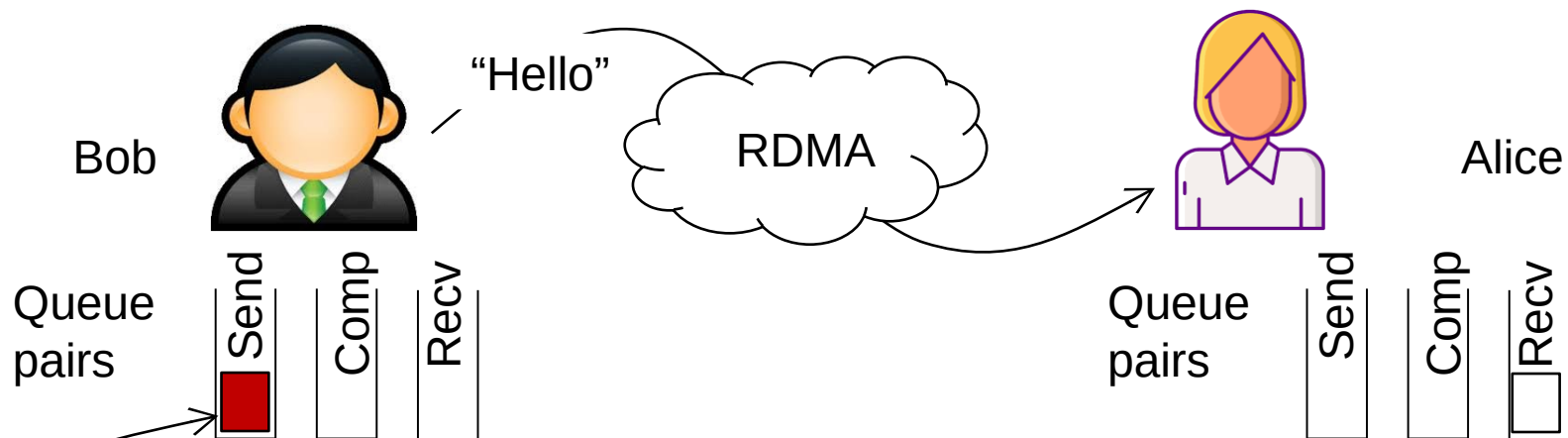
Register a message buffer to receive incoming message



```
struct ibv_sge list = { .addr = msg_buf, .length=5 };
struct ibv_recv_wr wr = {
    .sg_list = &list,
    ..., // other fields
};
ibv_post_recv(alice_qp, &wr, ...); // post to the receive
queue
```

Alice

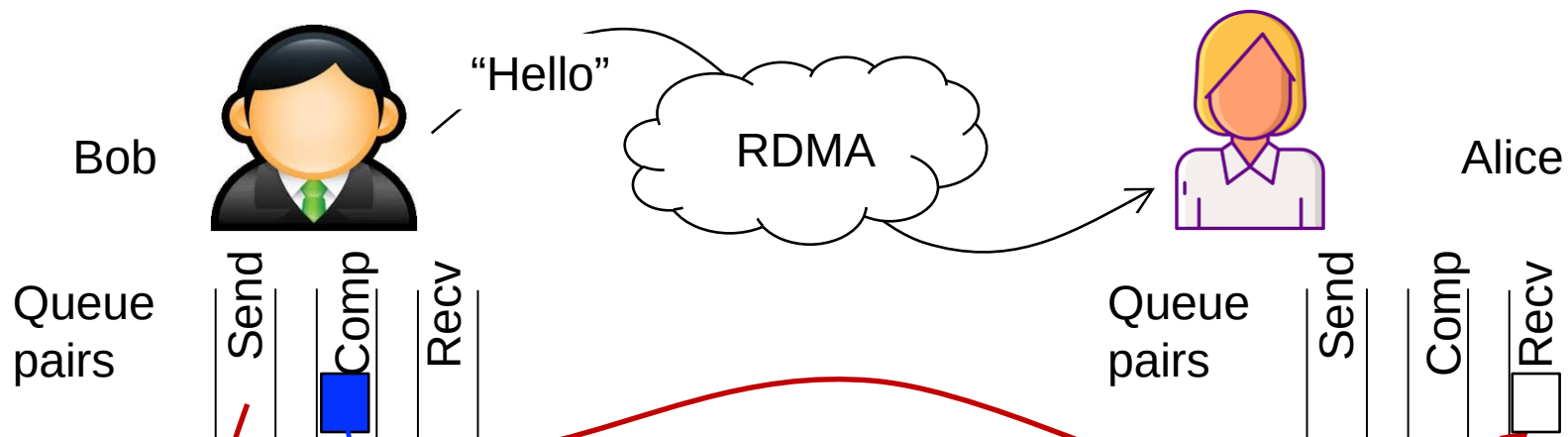
Send the message with post_send



```
struct ibv_sge list = { .addr = (uintptr_t)"Hello", .length = 5 };
struct ibv_send_wr wr = {
    .sg_list = &list,
    .op_code = IBV_WR_SEND // send a two-sided message
    ..., // other fields
};
ibv_post_send(bob_qp, &wr, ...); // send queue
```

bob

Poll the completion of the send via polling

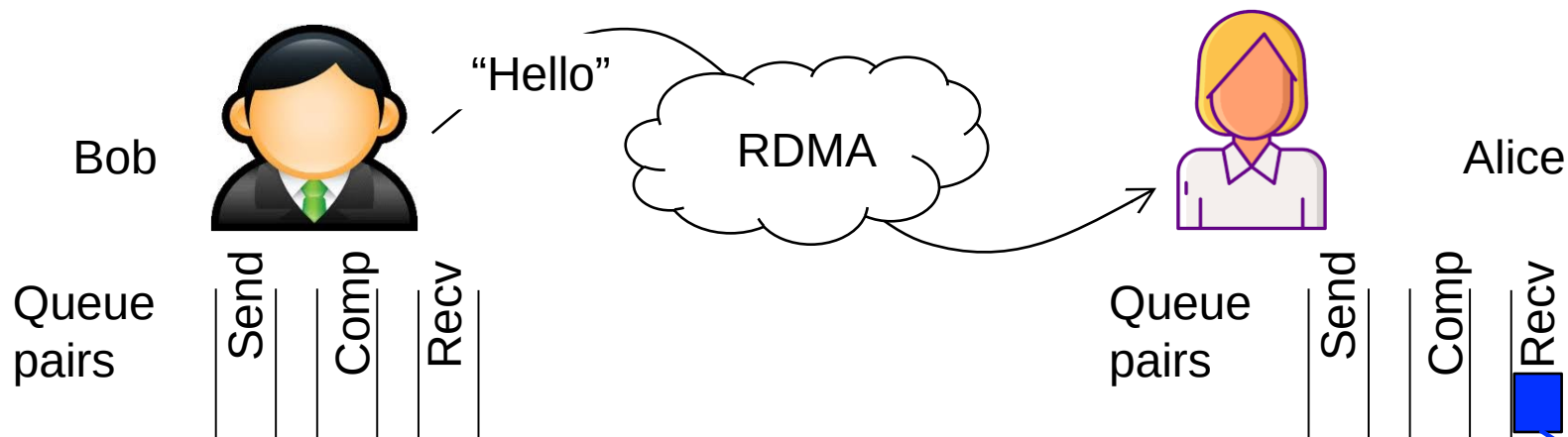


```
do {  
    num_completions = ibv_poll_cq(bob_com_qp, ...);  
} while (num_completions == 0);
```

```
// The request has been successfully sent to the remote  
// or an error happens
```

bob

Poll the received message from Alice



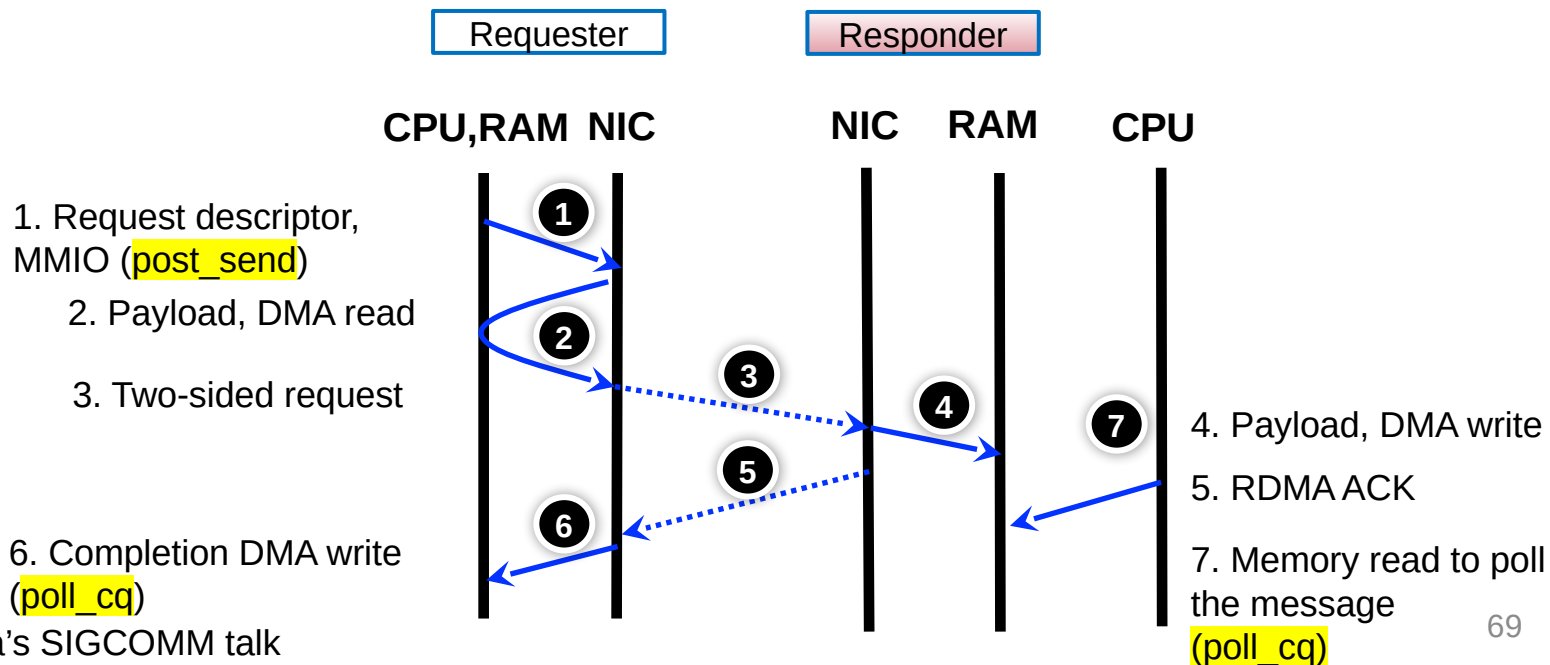
```
do {  
    num_completions = ibv_poll_cq(alice_receive_qp, ...);  
} while (num_completions == 0);  
  
// The message must be stored in msg_buf if no error  
Print(msg_buf) // Hello
```

bob

Lifecycles of the sent message

Secrets behind RDMA: MMIO & DMA

- MMIO: Using LOAD and STORE
- DMA: Device directly read/write the memory (provided by PCIe)



RDMA vs. DPDK

Common

- Both are kernel bypassing

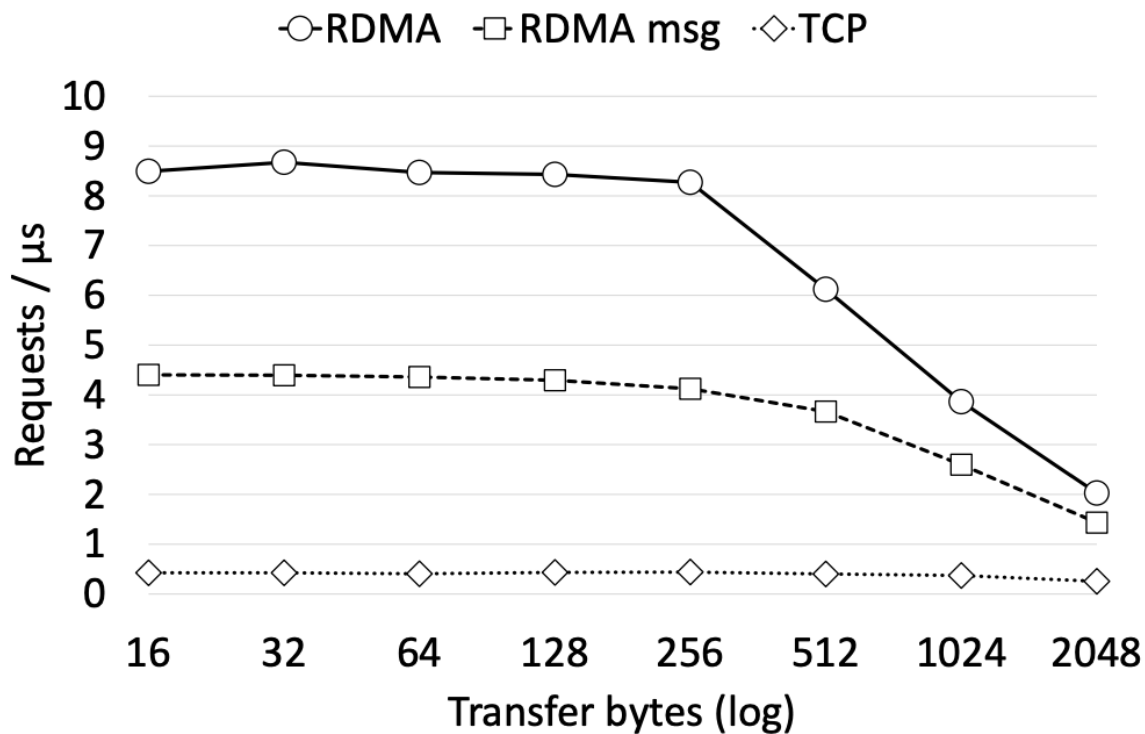
Advantage of RDMA

- Fully supported network in the hardware
- Fully zero copy

Advantage of DPDK

- More flexible, i.e., can use a customized network protocol
 - RDMA's black-boxed network protocol may result in poor performance if not using properly! (we will see)

Performance of RDMA (message)



Is DPDK & RDMA (messaging) sufficient?



CPU is becoming the problem



CPU is the problem

CPU cannot become faster

- Power wall (e.g., dennard scaling), The decline of Moore's Law

NIC is continuously becoming faster

- Lower latency (down to 600ns)
- Higher bandwidth (up to **400Gbps**)

Maybe we do not need such high bandwidth?

Most of the time the workload is idle

- Latency is more important than the bandwidth when the workload is idle

CPU achieves relatively low latency if idle w/ kernel-bypassing network

- No request **queuing** effect
- e.g., Two-sided RDMA achieves 2.2us RTT
- NIC RTT: ~1.5us

Energy: another problem of CPU

A hidden assumption of low-latency two-sided RDMA is using **polling**

Example: Bob uses polling to receive messages from the receive queue

- The while loop can easily burn the CPU!

```
do {  
    num_completions = ibv_poll_cq(alice_receive_qp, ...);  
} while (num_completions == 0);  
  
// The message must be stored in msg_buf if no error  
Print(msg_buf) // Hello
```

bob

Polling uses a lot of energy

CPU is not that energy-efficient as network cards

- And polling utilize all the energy of CPU



TDP: 105W

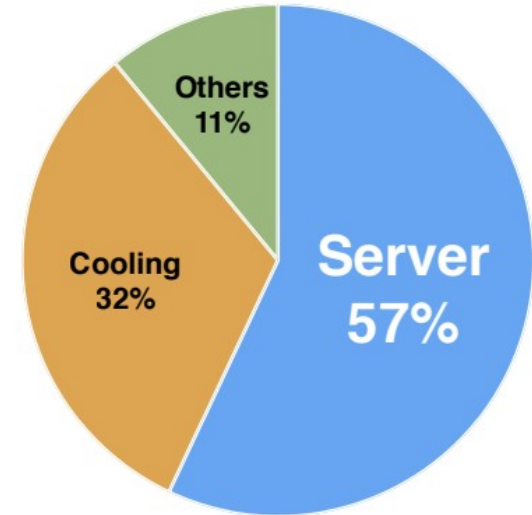
24 cores



TDP: 3.8W

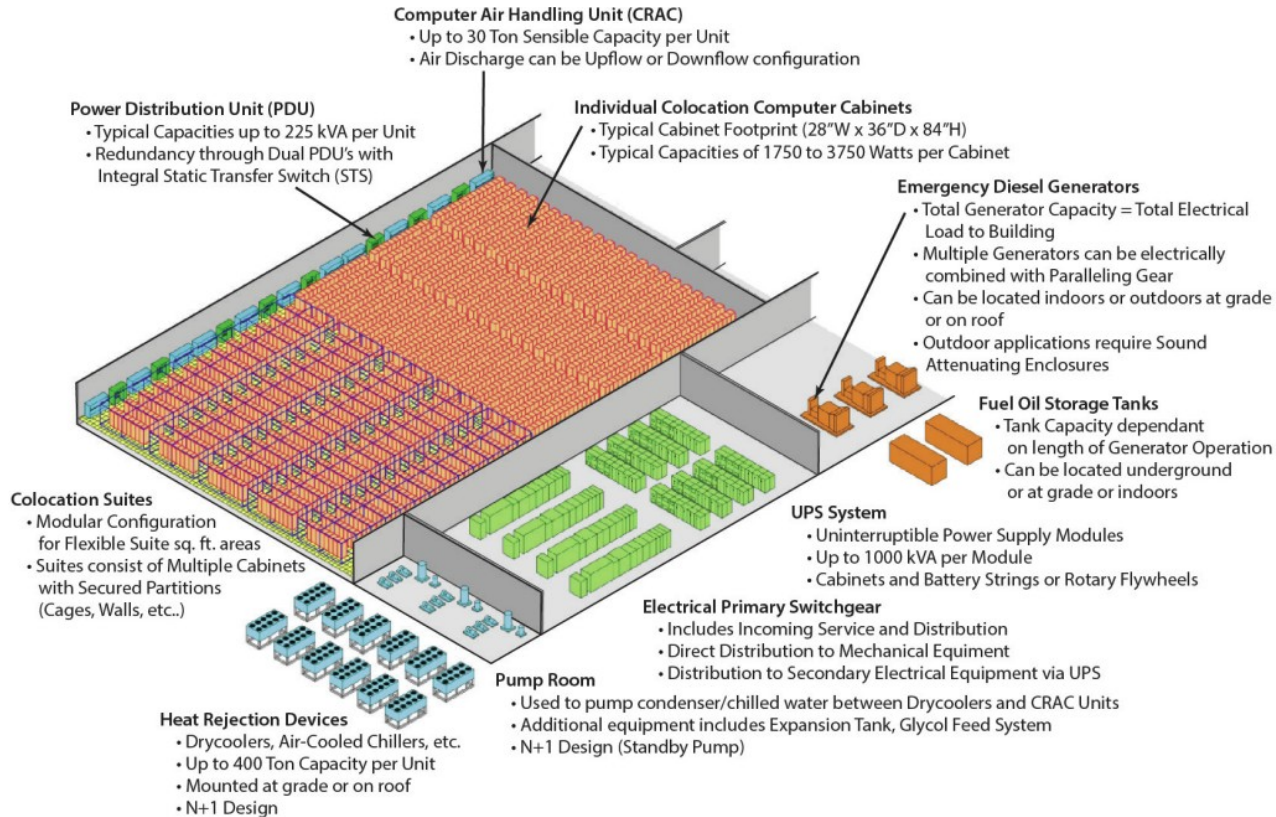
Energy is also very important in datacenter

- ❖ US data centers consume 70 billion kilowatt-hours of energy per year
- ❖ Server CPUs consume the most energy



Source: United States Data Center Energy Usage Report.

Power and Cooling Management in Datacenter

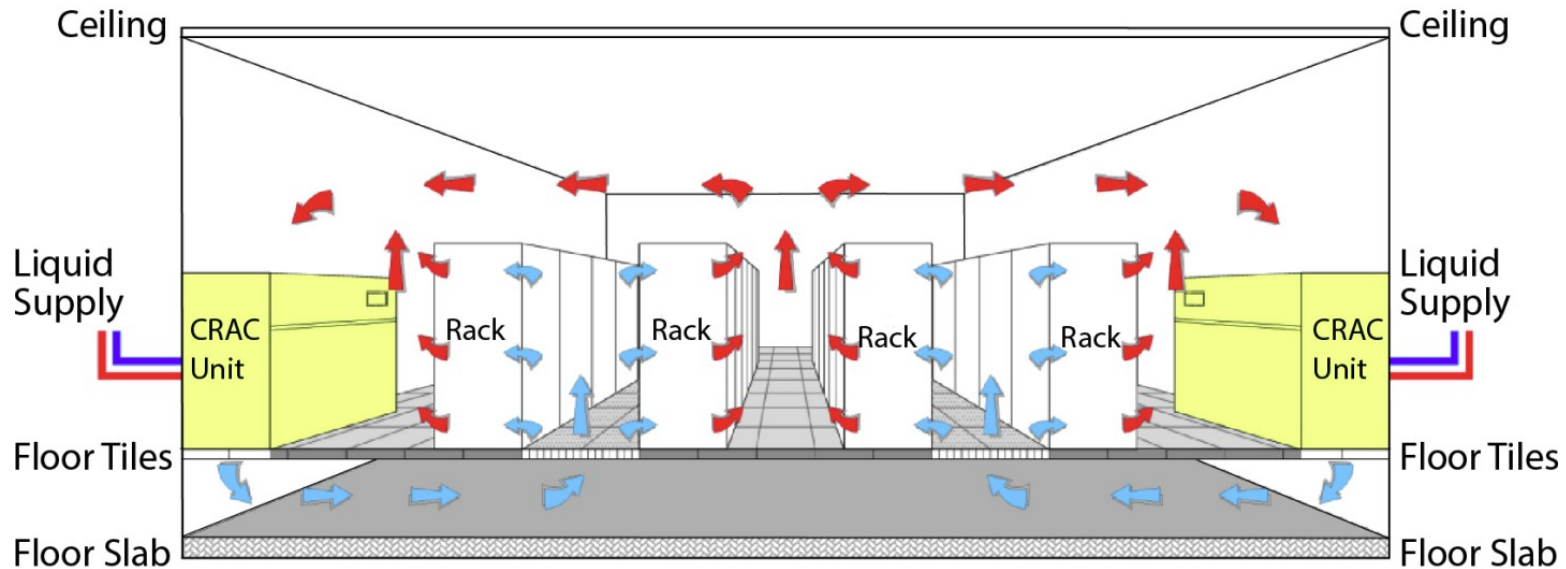


Datacenter Cooling Systems

Loop system to bring heat outside and cool medium in

Two-loop system

- CRAC Units: computer room air conditioning
- Liquid supply discharge heat to outside environment



“Free Cooling”

Cold weather

Close to water

Cheap energy bill

TECH

World's largest data center to be built in Arctic Circle

PUBLISHED TUE, AUG 15 2017-9:10 AM EDT | UPDATED TUE, AUG 15 2017-9:50 AM EDT

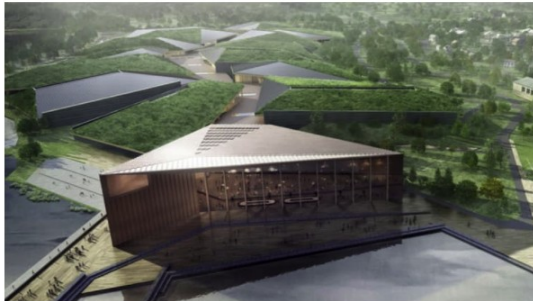


David Reid
@DAVYREID73

SHARE [f](#) [t](#) [in](#) [e](#)

KEY POINTS

- Data center is predicted to draw on a record amount of power.
- The facility is being developed by a US/Norwegian joint venture.
- Company claims site will be a “fortress for data”.



An artist impression of the proposed Kolos data center
Image copyright KOLOS

TV

Dateline

WATCH LIVE

UP NEXT | Shepard Smith 4:00 AM ET

Listen

← Ads by Google

Send feedback

Why this ad? ⓘ

The Kolos facility is being developed by a US-Norwegian partnership, also called Kolos, who say the site will eventually draw on a record-setting 1000 megawatts of power.

On their website, Kolos claim Ballangen's cold climate and access to hydropower will help trim energy costs by as much as 60 percent.

The company added that Kolos will be a “fortress for data”.



Can we use interrupt to reduce the energy?

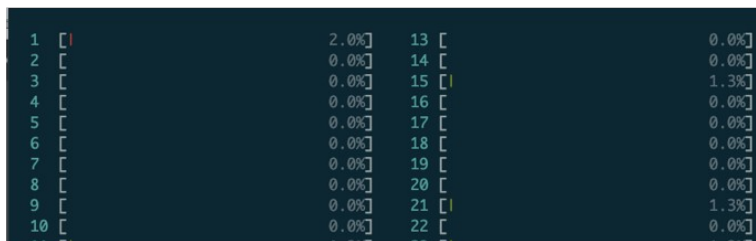
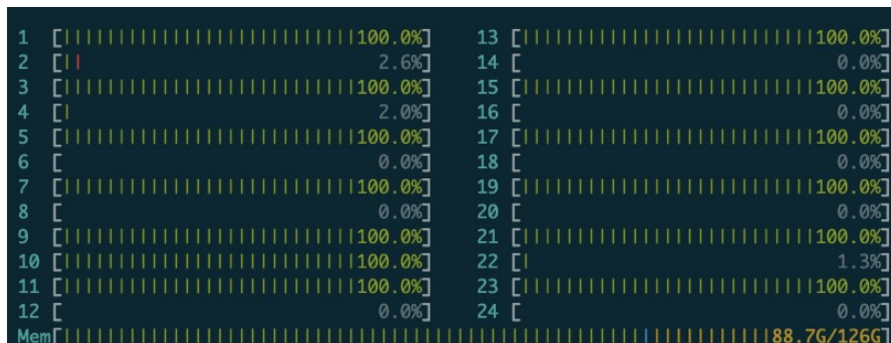
Interrupt:

- Yield to the kernel, will be scheduled back once the message is received
- **Energy efficient**: CPU has nearly zero costs if no interrupt comes

Polling

vs.

Interrupts



Can we use interrupt to reduce the energy?

Interrupt:

- Yield to the kernel, will be scheduled back once the message is received

Problem:

- **1000X** latency increases! (2 μ s vs. 2ms)
 - Due to the kernel and scheduling effects of the CPU

Dilemma between energy & performance!

Does kernel-bypassing networking
sufficient?

Not sufficient!

Efficiency = performance + low power consumption



+



We also need CPU bypassing!

One-sided RDMA

Observation:

- Remote read/write operations is common in applications
 - We can let NIC do all these operations!

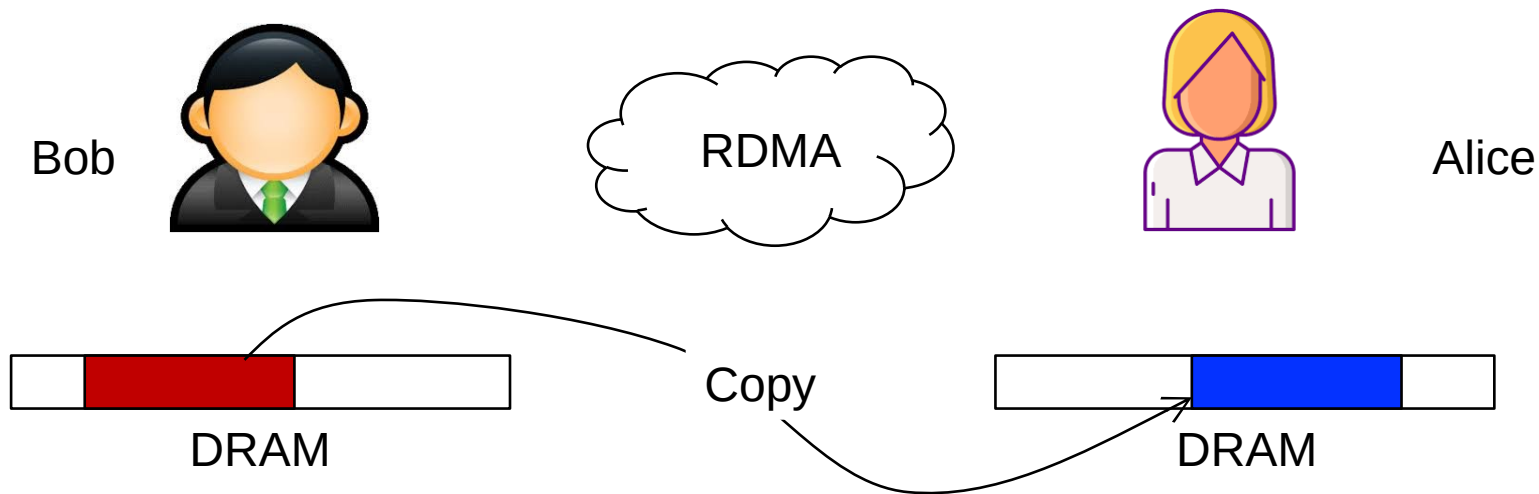
One-sided RDMA

- **READ**
 - The NIC can directly read the server memory bypassing server CPU
- **WRITE**
 - The NIC can directly write the server memory bypassing server CPU
- **ATOMICS** (fetch & add, compare & swap)
 - The NIC can directly execute atomic operations bypassing server CPU

Example: remote memory copy

Suppose we want to implement a remote procedure call (RPC) with RDMA

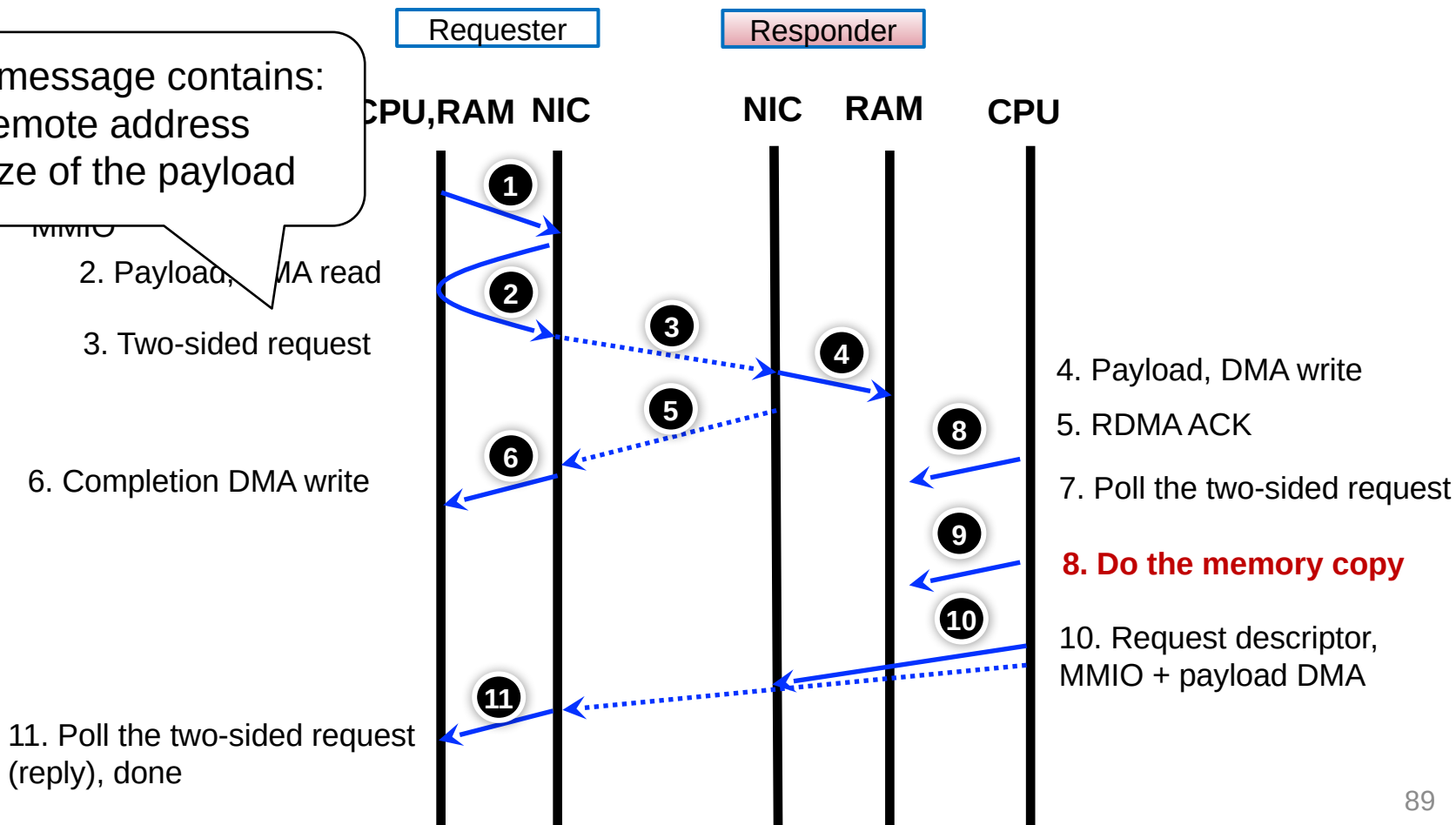
- The RPC will simply do a memory copy (memcpy)
- For simplicity, we assume only the requester copies to the responder



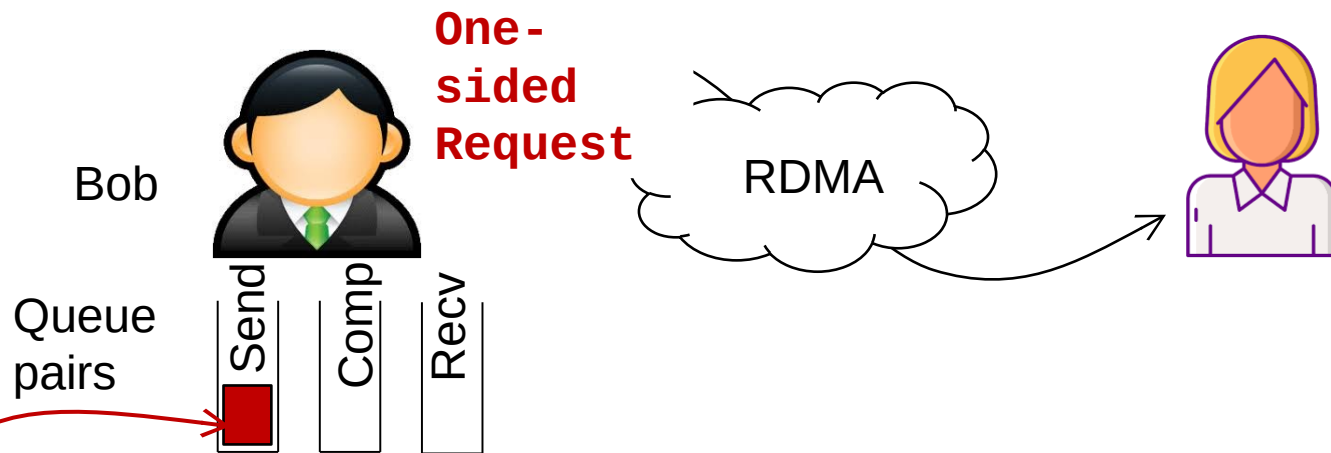
Existing approach: two-sided RDMA based RPC

The message contains:

1. Remote address
2. Size of the payload



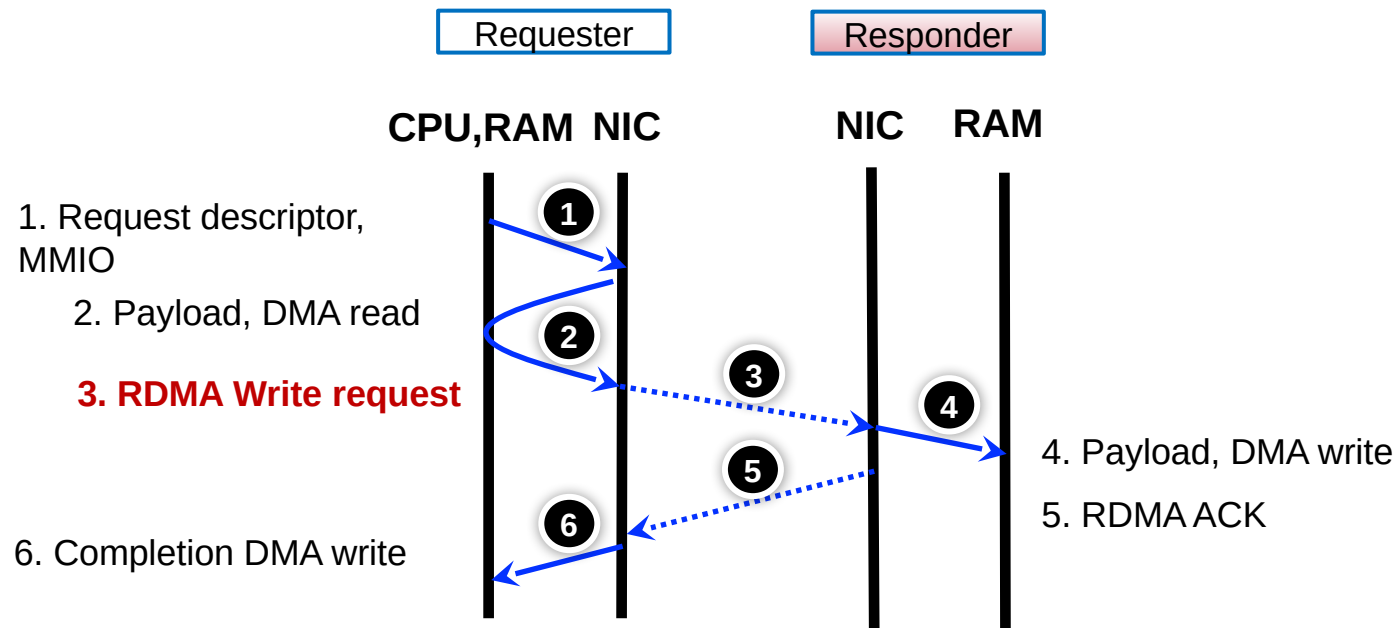
Remote memory copy with one-sided RDMA WRITE



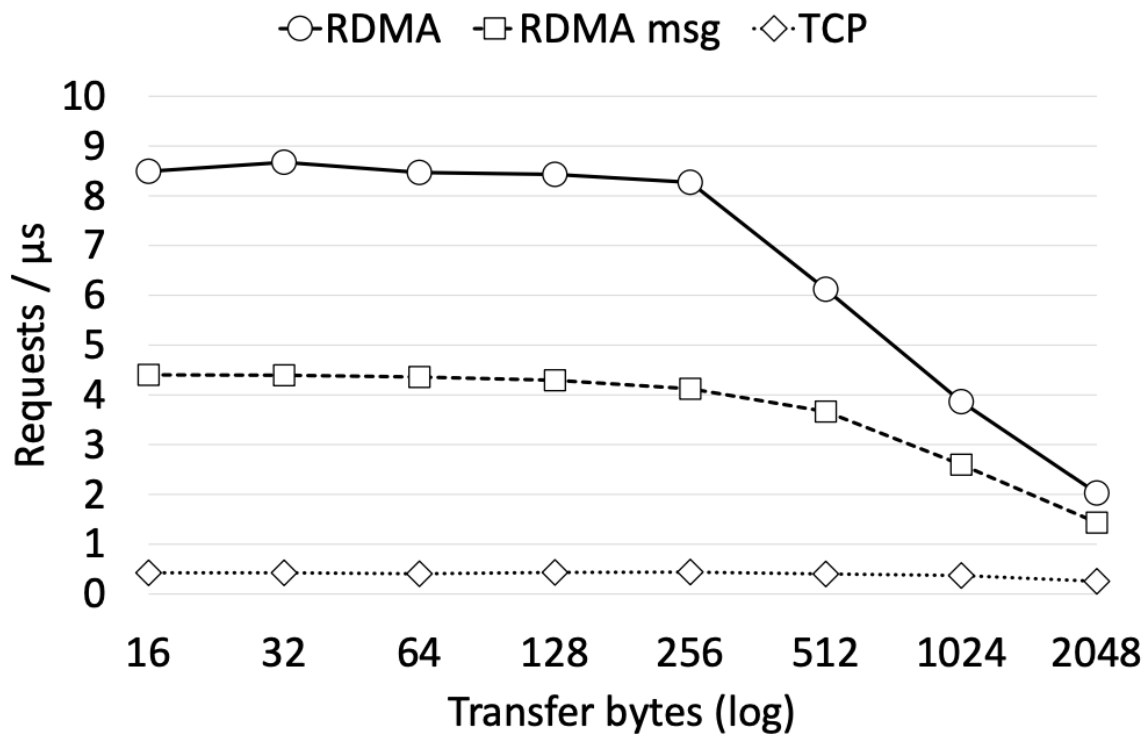
```
struct ibv_sge list = { .addr = src, length=size };
struct ibv_send_wr wr = {
    .sg_list = &list,
    .op_code = IBV_WR_RDMA_WRITE, // to the the one-sided WRITE
    .wr.wr.rdma.remote_addr = dst,
    ..., // other fields
};
ibv_post_send(bob_qp, &wr, ...); // send queue
```

bob

Remote memory copy with one-sided RDMA WRITE



One-sided RDMA WRITE vs. Two-sided RDMA



Question: how can RDMA access the remote memory?

RDMA can directly access the virtual address of the remote end

- How can the NIC find the DMA address?
- How can the remote end do the authentication?
 - E.g., some virtual address cannot be exposed to the others

```
struct ibv_sge list = { .addr = src, length=size };
struct ibv_send_wr wr = {
    .sg_list = &list,
    .op_code = IBV_WR_RDMA_WRITE, // to the the one-sided WRITE
    .wr.wr.rdma.remote_addr = dst,
    ..., // other fields
};
ibv_post_send(bob_qp, &wr, ...); // send queue
```

Memory registration (MR)

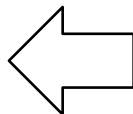
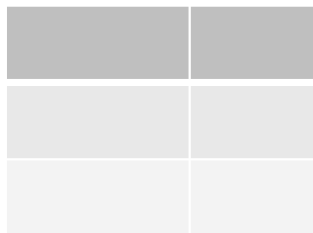
RDMA uses memory registrations to expose the virtual memory

If Alice wants to expose the memory, she must first register it to the RNIC

- Via `ibv_reg_mr()`
- Access flags: read/write, etc.
- Return: rkey

`ibv_reg_mr(..., start, end-
start, access_flags, ...)`

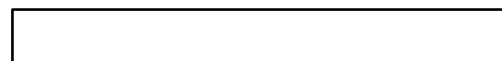
NIC will create an internal mapping



Alice

Start

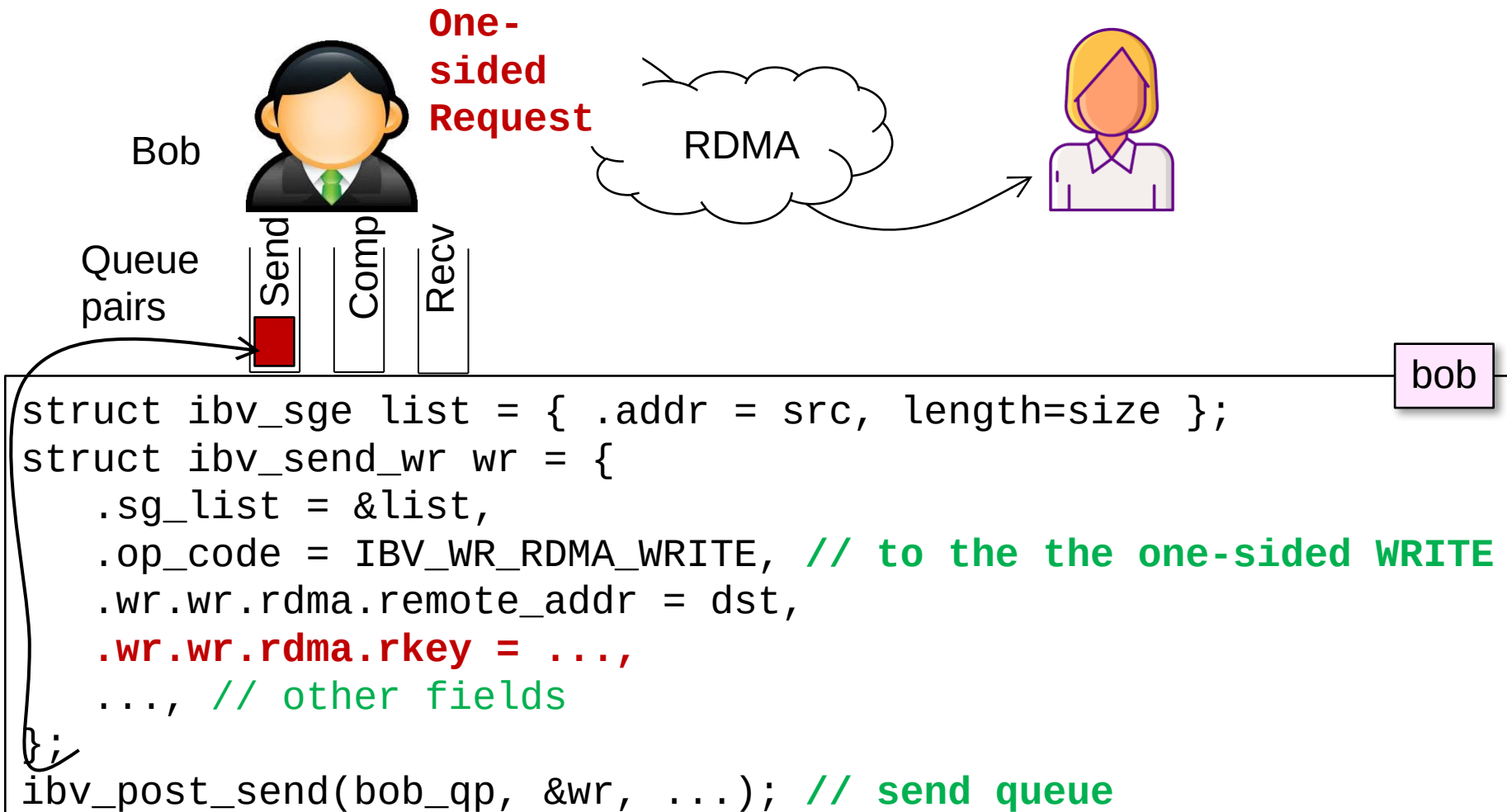
End



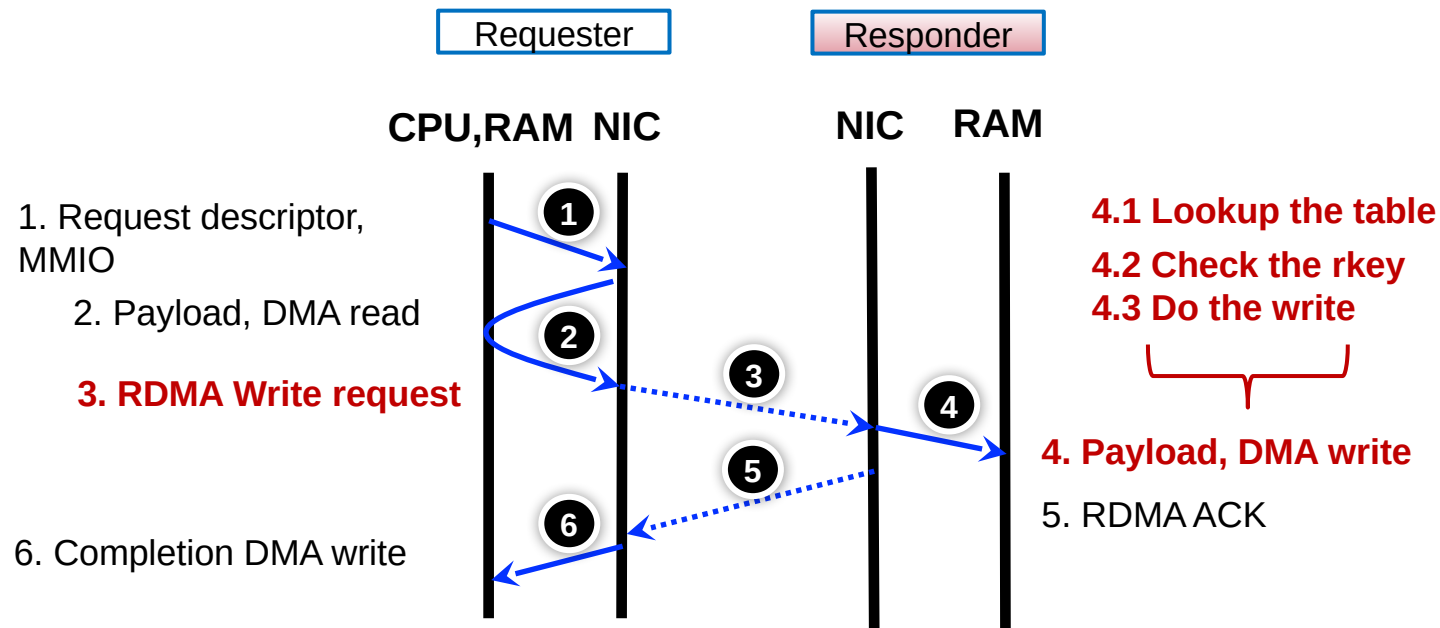
DRAM

Virtual – Physical mapping

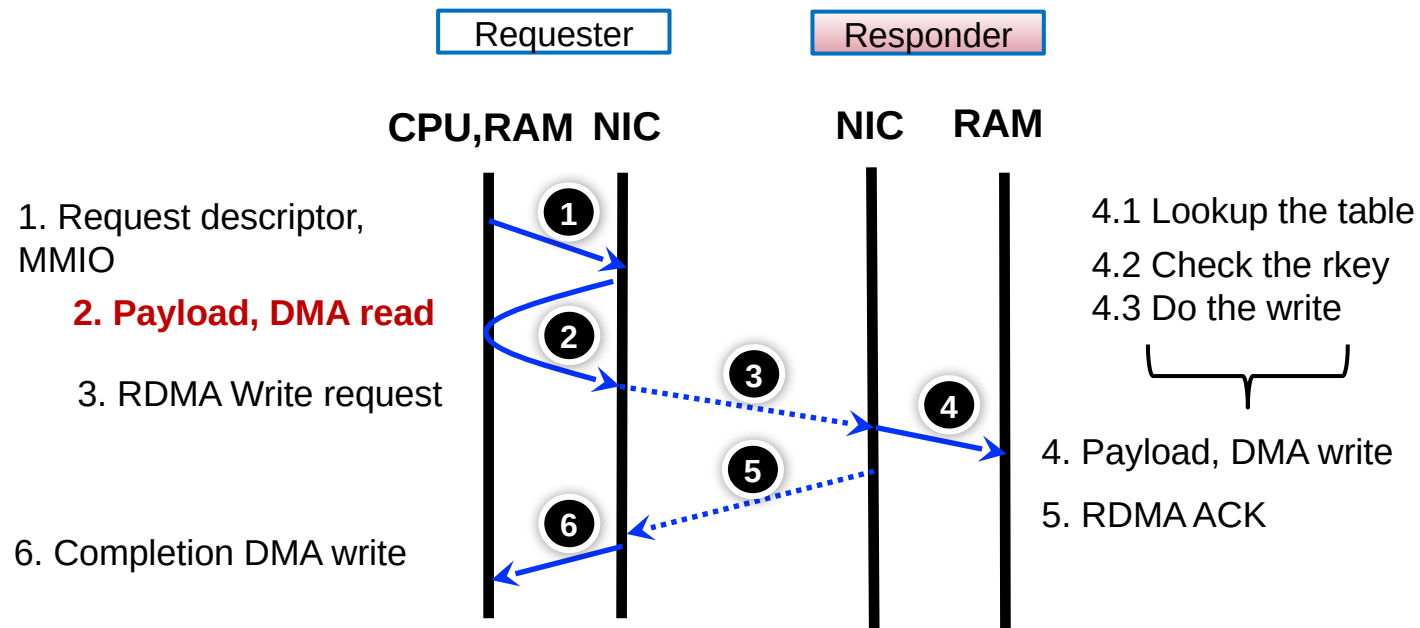
One-sided RDMA with Memory Registration



One-sided RDMA with Memory Registration



One-sided RDMA with Memory Registration



Question: what about the local DMA?

Local RDMA access also needs MR

The same API `ibv_reg_mr()`

- Will return two keys: `lkey` & `rkey`
 - `Lkey` is used for local accesses
 - `Rkey` is used for remote accesses

bob

```
struct ibv_sge list = { .addr = src, length=size, lkey = lkey };
struct ibv_send_wr wr = {
    .sg_list = &list,
    .op_code = IBV_WR_RDMA_WRITE, // to the the one-sided WRITE
    .wr.wr.rdma.remote_addr = dst,
    .wr.wr.rdma.rkey = ...,
    ..., // other fields
};
ibv_post_send(bob_qp, &wr, ...); // send queue
```

Question: where is the mapping table stored?

Stored in DRAM

- Problem: NIC will use one DMA to fetch for the check (inefficient!)

NIC will use a small SRAM for the cache

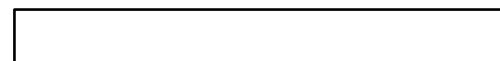
- Avoiding extra lookups for the MR

`ibv_reg_mr(..., start, end-
start, access_flags, ...)`

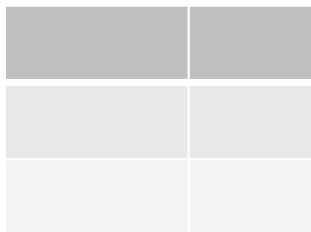
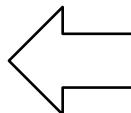


Alice

End



DRAM



Virtual – Physical mapping

Performance summary of RDMA

We use one-sided RDMA WRITE as an example

- ① MMIO: 200-800ns (on Intel x864 machines)
- ② + (4) DMA: (about) 1us
- ③ + (5): network, depends on the distance & speed of light
 $\ll 1\mu\text{s}$ in rack-scale clusters
- ⑥ Memory access
100-400ns

Total RTT: 2-5us

