DSC-204A: Scalable Data Systems, Fall 2025

1: Introduction (Cloud & Networking Foundations)

Lecturer: Hao Zhang *Scribe: Felipe Lorenzi, Ryan Clement, Thuy Nguyen, Taylor Martinez*

Course Context and Motivation

Where we are: This course spans three pillars of modern scalable data systems: (i) cloud & data centers, (ii) networking primitives leading into collective communication, and (iii) distributed storage/compute with ML workloads as recurring case studies. The first unit emphasizes networking because collective communication for ML/HPC builds directly on these fundamentals.

Student feedback and focus: Class feedback requested more advanced, ML/LLM systems content (many expect ≥ 30

Learning objectives for this lecture:

- Explain the evolution from Cloud 1.0 $\rightarrow$ Cloud 3.0 and why disaggregation is increasingly viable.
- Compare renting models (on-demand, reserved, spot) and the operational trade-offs they induce.
- Define core network concepts (addresses, ports, packets) and performance metrics (latency, throughput, jitter, loss, reordering).
- Connect basic send/recv abstractions to collective communication patterns (reduce/all-reduce) used in distributed deep learning.

Cloud Generations and Why They Matter

Cloud 1.0 (Past): Renting Whole Machines

What it is: Users rent entire, networked servers. The model offers strong isolation and conceptual simplicity, but utilization is poor: you pay for idle capacity whenever your workload is bursty or imbalanced.

Implications: This model is easy to reason about operationally but expensive for variable or spiky workloads. It predates widespread virtualization.

Cloud 2.0 (Current): Virtualized Fleets

What it is: Providers virtualize servers into VMs/containers so many tenants share a physical host. Rent resource capacity (e.g., vCPU, RAM, storage) rather than whole boxes. Compute and storage are routinely decoupled to scale independently (e.g., compute on VMs, data on a shared store like S3).

Implications: Cheaper than Cloud $\sim$ 1.0 and far more flexible (multi-tenancy, load balancing, elasticity), but still wastes resources by forcing users to rent fixed-size "units." Cross-node parallelism increasingly relies on high-speed datacenter networks (100 GbE $\rightarrow$ TbE links) and hybrids of shared-disk plus shared-nothing designs.

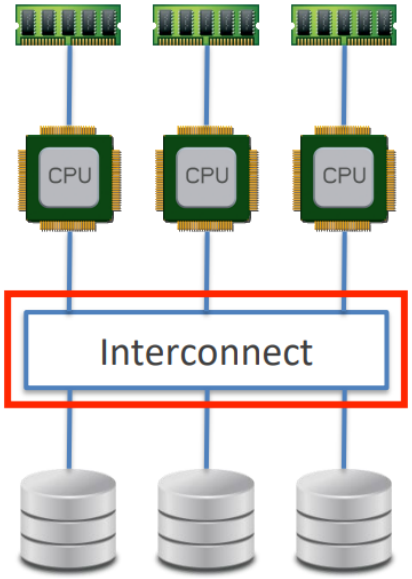


Figure 1: Parallelism in the Cloud (use Slide 7).

Cloud 3.0 (Ongoing Research): Fully Disaggregated & Serverless

What it is: Compute, memory, and storage are disaggregated and network-attached. The platform elastically composes the right mix per request. Users submit a function or program and provide resource hints (e.g., CPU, DRAM). The provider manages provisioning transparently (Function-as-a-Service / serverless).

Why now? Very fast datacenter networks make remote memory/storage accesses practical at scale. For certain workloads (e.g., online ML inference), serverless can deliver much higher utilization and significantly lower cost (illustratively, often order-of-magnitude cheaper than spot for suitable patterns), at the price of cold-starts, more complex scheduling, and stricter security/privacy boundaries.

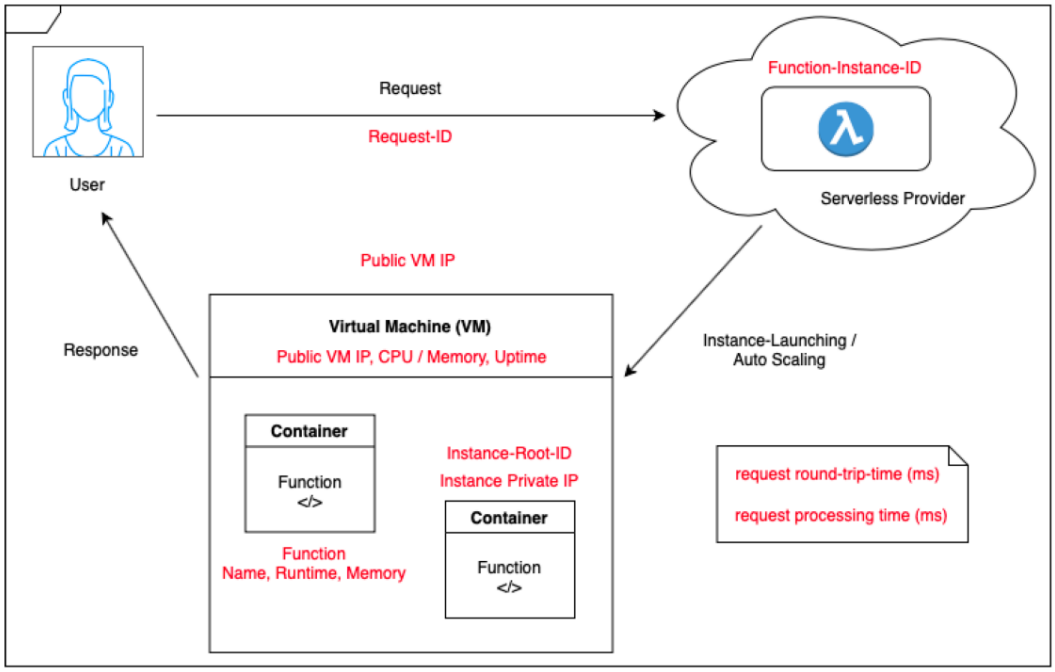


Figure 2: Cloud 3.0 disaggregation & serverless (Slides 8–10).

Renting Models in Practice

On-Demand. Pay-as-you-go; most flexible; highest unit cost. You are guaranteed the node while you pay.

Reserved. Commit to capacity & term (months/years) for discounts. Good for steady-state fleets; less flexible for changing needs.

Spot/Preemptible. Deep discounts for spare capacity that can be reclaimed by the provider at any time. Best for fault-tolerant jobs with checkpointing/migration.

Microservices vs. Cloud 3.0

Microservices represent a key trend in modern cloud computing, often associated with what is called "Cloud 3.0." This architectural approach breaks applications into smaller, loosely coupled services that are independently deployable and highly scalable, making it easier to allocate computing resources flexibly for different users and workloads.

While microservices offer significant advantages in terms of agility and scalability compared to traditional monolithic architectures, they can still be challenging to coordinate due to the complexity of managing many independent components.

	Spot Instances	On-Demand Instances
Launch time	Can only be launched immediately if the Spot Request is active and capacity is available.	Can only be launched immediately if you make a manual launch request and capacity is available.
Available capacity	If capacity is not available, the Spot Request continues to automatically make the launch request until capacity becomes available.	If capacity is not available when you make a launch request, you get an insufficient capacity error (ICE).
Hourly price	The hourly price for Spot Instances varies based on demand.	The hourly price for On-Demand Instances is static.
Rebalance recommendation	The signal that Amazon EC2 emits for a running Spot Instance when the instance is at an elevated risk of interruption.	You determine when an On-Demand Instance is interrupted (stopped, hibernated, or terminated).
Instance interruption	You can stop and start an Amazon EBS-backed Spot Instance. In addition, the Amazon EC2 Spot service can interrupt an individual Spot Instance if capacity is no longer available, the Spot price exceeds your maximum price, or demand for Spot Instances increases.	You determine when an On-Demand Instance is interrupted (stopped, hibernated, or terminated).

Figure 3: Spot vs. On-Demand trade-offs (Slides 10–11).

Advantages and Disadvantages (At a Glance)

- **Cloud 1.0:** + Simple, strong isolation; − Expensive (idle waste).
- **Cloud 2.0:** + Cheaper than 1.0; − Some resource waste persists due to fixed unit sizes.
- **Cloud 3.0:** + Often cheapest (utilization wins); − Cold starts; security/privacy complexity; harder operations.

Ecosystem and Strategic Notes

Stack view. Hardware → cloud infrastructure → platforms (e.g., data services) → applications (e.g., web apps, LLM-backed services). In today's "AI-heavy" landscape, a large share of immediate value accrues to hardware vendors and clouds, with applications racing to improve utilization and unit economics. This hierarchy technology hierarchy can be found reflected in the profit chain of Silicon Valley.

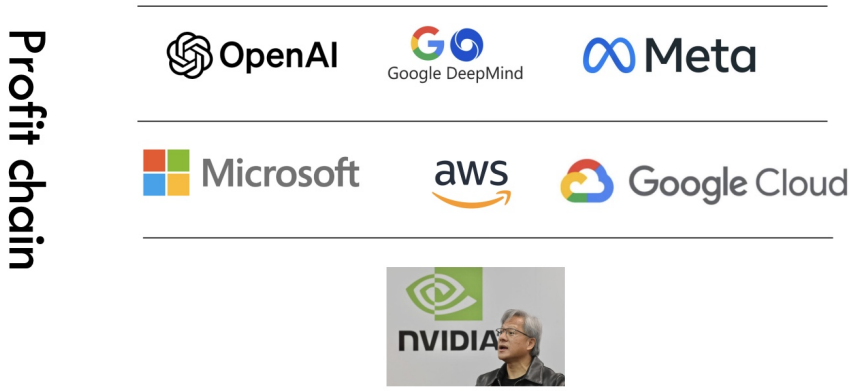


Figure 4: Profit chain (Slide 15).

Trends. Capacity is scarce for top-tier accelerators; large users increasingly reserve long-term fleets. Some organizations are exploring on-prem "supercomputer-style" builds to better control cost and dependency.

Architecture & culture. The lecture highlights that microservice-centric organizations aligned naturally with the rise of disaggregated services. As an open question, students are asked why AWS (and even Azure) outpaced GCP in market share despite Google's historical systems leadership; one suggested factor is organizational architecture and productization around microservices.

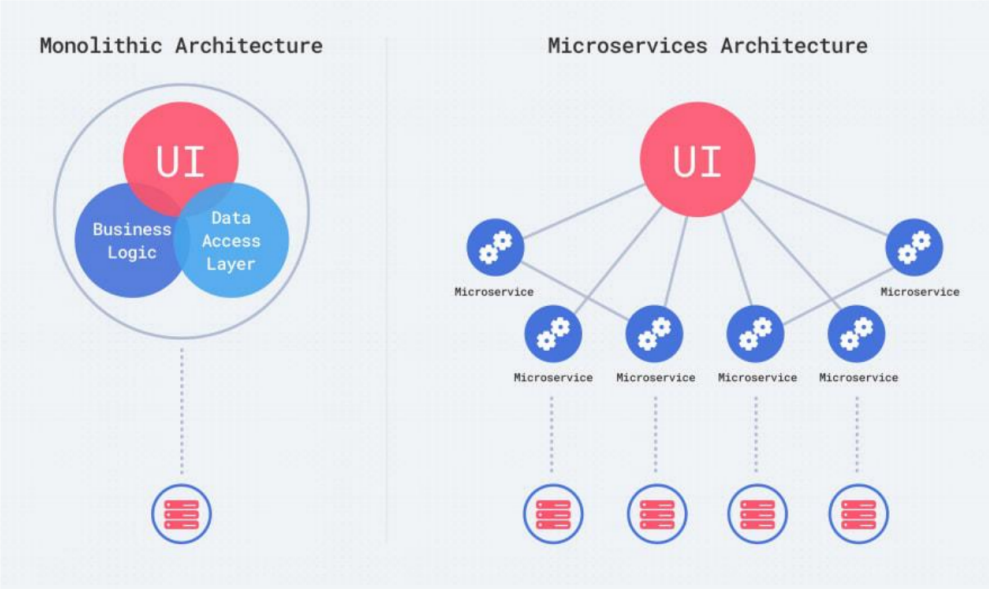


Figure 5: Today's cloud trend: disaggregated resources and services (Slide 20).

From Cloud to Networking

A ChatGPT Request as a Motivating Example

A single user request may traverse DNS, internet paths, load balancers, and GPU-backed compute nodes; responses may stream tokens while accessing caches or storage. This end-to-end path underscores why networking performance is a first-order concern for ML systems and cloud services.

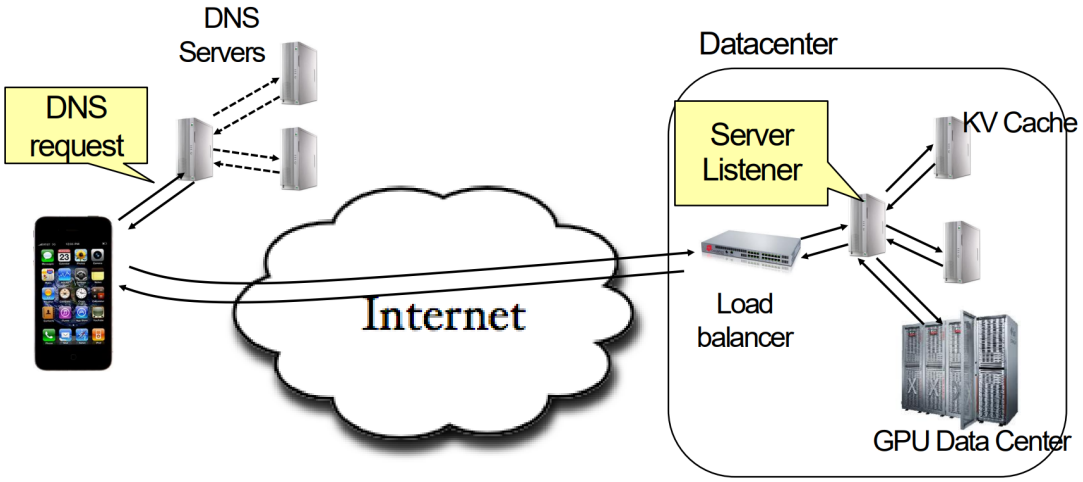


Figure 6: "What's in a ChatGPT query?" (Slide 23).

Why Networking Matters

Nearly all modern apps are networked, with a significant fraction of functionality running remotely in provider data centers. Connectivity problems and datacenter incidents manifest directly as user-visible failures. The OS and datacenter stack therefore prioritize robust connectivity, throughput under load, and predictable latency for quality of service.

Networking Basics

Hardware and Identifiers

NICs: A network interface card/controller connects a host to the network.

Addresses: Each NIC has a hardware MAC address (48-bit). Hosts are also assigned IP addresses (IPv4 or IPv6), and user-space processes are identified by port numbers at transport endpoints. A *connection* is a channel between two processes, identified by a tuple including ports.

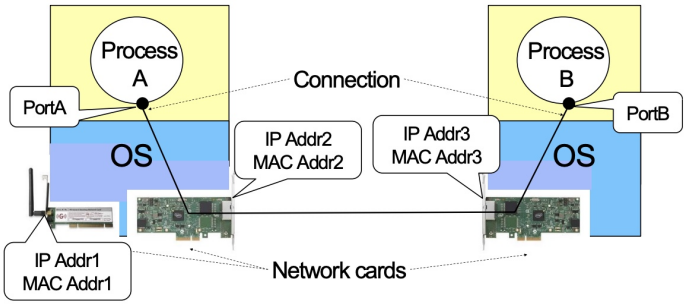


Figure 7: Network communication (Slide 27).

Links, Switching, and Multiplexing

At scale, we share link capacity among many flows using switches. Historically, *circuit switching* reserved a dedicated path for each session (telephone-era). Modern packet-switched networks instead split messages into *packets* (bounded-size frames with headers/trailers) that carry source/destination/type and payload; switches forward each packet independently (store-and-forward), keeping links busy whenever there is traffic.

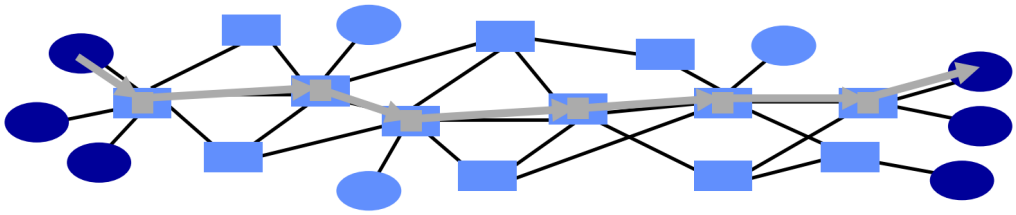


Figure 8: Packet switching (Slide 35).

Congestion, Buffers, and Control

Each link has finite bandwidth. Switch/host ingress buffers absorb bursts, but when queues overflow, packets are dropped. Rising losses trigger congestion control at senders to back off rates until aggregate load matches available resources. This dynamic explains phenomena like "Wi-Fi collapse" in dense venues when buffers and links saturate.

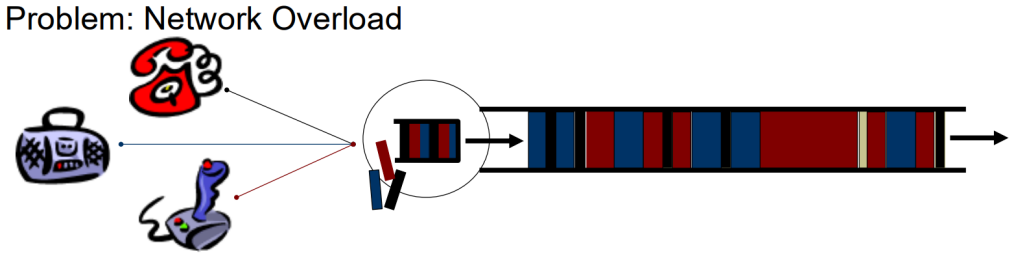


Figure 9: Buffering and congestion control under overload (Slide 38).

Performance Metrics and Delay Components

Latency: (time to first bit),
Throughput/capacity: (bits/s),
Jitter: (variation in latency),
Loss/reliability: (drop rate),
Reordering: characterizes network behavior.

End-to-end delay is the sum of:

- **Propagation** delay (distance-limited per hop),
- **Transmission** delay (packet size / link speed),
- **Processing** delay (router/switch speed),
- **Queuing** delay (load and queue size dependent).

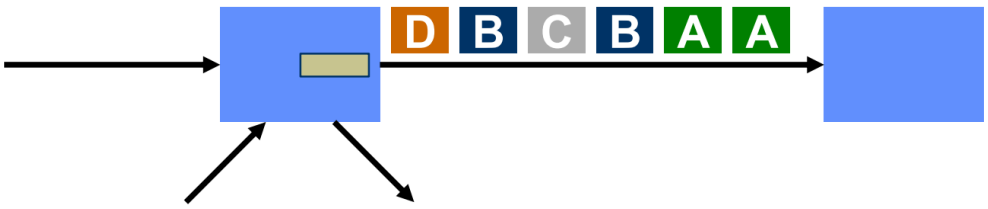


Figure 10: Packet delay components (Slide 40).

Worked Example: Latency and Effective Throughput

A back-of-the-envelope calculation illustrates latency dominance for small transfers across long distances. With an illustrative cross-country path of $\approx 5 \times 10^6$ m and propagation $\approx 2 \times 10^8$ m/s (fiber), one-way propagation is $\approx$ 25 ms (RTT $\approx$ 50 ms). For 100 Mbps link rate and 10 kbit packets, transmission is $\approx$ 0.1 ms, negligible against propagation. Effective throughput for a single-packet exchange with an ACK can thus be only $\sim$ 200 kbit/s, limited by RTT rather than nominal link speed.

- Cross country latency
  - Distance/speed = 5 * 10^6m / 2x10^8m/s = 25 * 10^-3 s = 25ms
  - 50ms RTT
- Link speed (capacity) 100Mbps
- Packet size = 1250 bytes = 10 kbits
  - Packet size on networks usually = 1500 bytes across wide area or 9000 bytes in local area
- 1 packet takes
  - 10k/100M = .1 ms to transmit
  - 25ms to reach there
  - ACKs are small → so 0ms to transmit
  - 25ms to get back
- Effective bandwidth = 10kbits/50.1ms = 200kbits/sec

Figure 11: Simple RTT and throughput example (Slide 42).

Layering and End-to-End Abstractions

A canonical five-layer model (Physical, Datalink, Network, Transport, Application) provides common abstractions. The lower three layers exist on all network elements; the top layers live at hosts. Each layer speaks to its peer via well-defined headers and semantics, allowing applications to rely on simple APIs while the stack handles reliability, routing, and flow control.

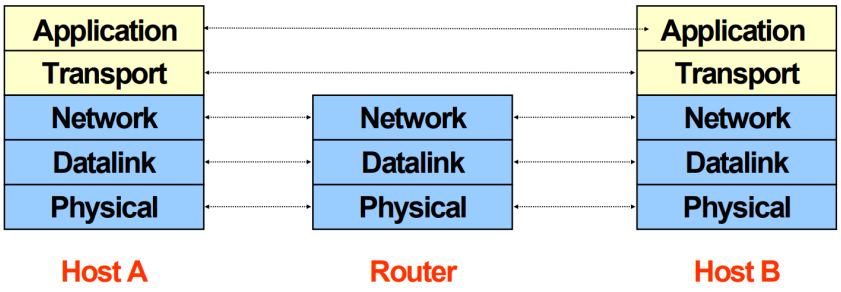
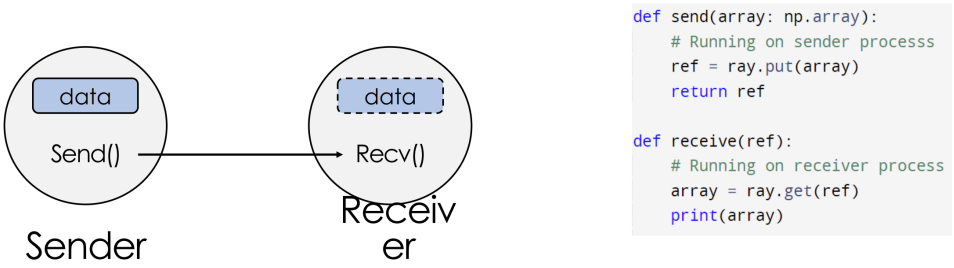


Figure 12: Five-layer network stack (Slide 43).

Programming Model: Point-to-Point to Collectives

Point-to-Point (P2P) Send/Recv

At the application level, a basic model is one *sender* and one *receiver*, with corresponding `send()` and `recv()` operations and receive buffers. In distributed runtimes like Ray, a straightforward baseline is to `put` data into an object store and send a reference for the receiver to `get`. This baseline is simple but adds an extra hop through the store, which can double latency; an optimization path is to approach direct process-to-process transfers.




```python
def send(array: np.array):
    # Running on sender processs
    ref = ray.put(array)
    return ref

def receive(ref):
    # Running on receiver process
    array = ray.get(ref)
    print(array)
```

Figure 13: P2P communication with Ray (Slide 45).

Case Study: Gradient Aggregation in Distributed ML

In data-parallel training, P workers compute gradients on different data shards and must aggregate them to update the global model parameters. A naive "funnel" design that routes all gradients into a single node and then fans out the result creates a hotspot both inbound and outbound.

We can see this in figure 14, where even though we only need to generate the gradient once, out worker node still needs to send this updated gradient to all previous nodes, which represents the "funnel" bottleneck previously mentioned

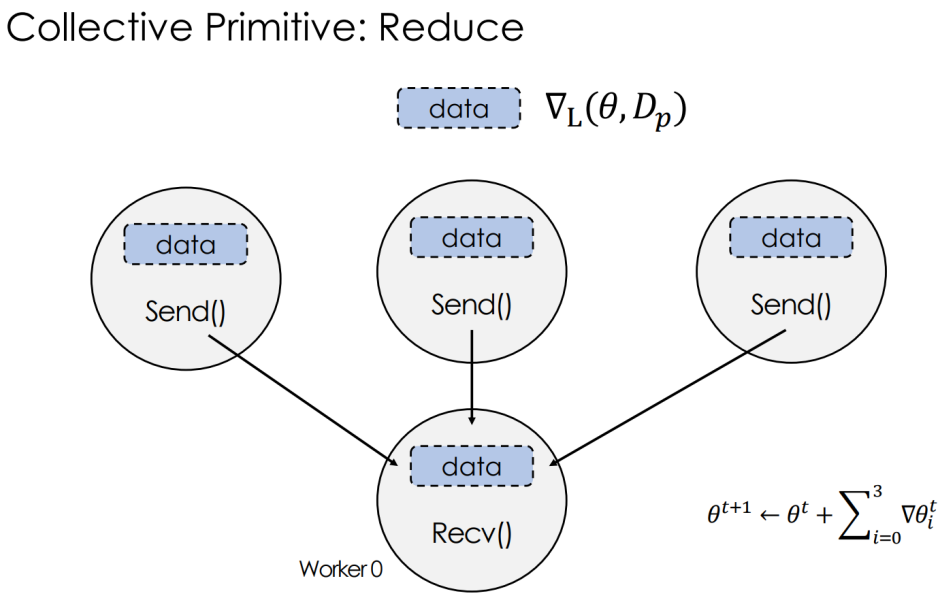


Figure 14: Collective Primitive Reduce and why naive aggregation bottlenecks (Slides 47–50, 52–53).

Collective Primitives: Reduce and All-Reduce

Reduce: Each worker sends its gradient; the receiver aggregates (e.g., sum) and produces a single result. Message cost with a single aggregator is roughly $3N$ for N inputs (N sends into, one result out to N? — motivating better patterns).

All-Reduce: Every worker needs the aggregated result. Structured collectives (e.g., tree- or ring-based) avoid a central hotspot by arranging communication so each participant contributes/receives partial results in rounds, improving scalability over the naive design.

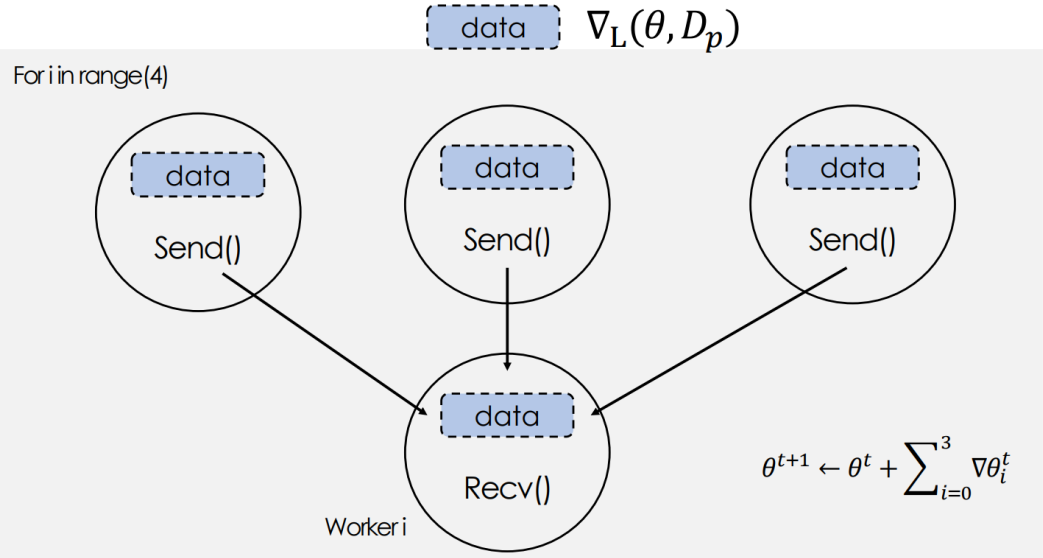


Figure 15: Motivation for all-reduce in synchronized training (use Slide 54).

Operational Takeaways

- **Utilization dominates cost:** Disaggregation + fast networks enable serverless patterns that squeeze idle time, often lowering cost materially for the right workloads.
- **Choose renting models by risk tolerance:** On-demand is simple but pricey; reserved trades flexibility for savings; spot offers lowest price if you can tolerate preemption (checkpoint/migrate).
- **Performance is pipeline-limited:** Throughput follows the slowest link/stage; latency has immutable components (speed-of-light propagation).
- **From send/recv to collectives:** Understanding P2P costs clarifies why structured collectives (reduce, all-reduce) are essential in ML training backends.

Checklist / What You Should Be Able to Do After This Lecture

- Sketch the three cloud generations and articulate pros/cons of each.
- Given a workload description, pick a plausible renting model and justify it.
- Decompose end-to-end latency into propagation, transmission, processing, and queuing; identify when RTT dominates.
- Explain how packetization, buffering, and congestion control interact under load.
- Describe why naive gradient aggregation hotspots appear and how all-reduce mitigates them.

Glossary (Selected)

Disaggregation: Separating compute, memory, and storage into network-attached pools that can be recombined elastically. **Serverless / FaaS:** A model where users submit functions and the platform handles provisioning; billing generally tracks fine-grained usage. **On-Demand / Reserved / Spot:** Three broad renting paradigms with a price–flexibility–risk trade-off. **NIC (Network Interface Card):** Hardware that connects a host to the network and exposes MAC/IP addressing to the stack. **Latency / Throughput / Jitter / Loss:** Standard metrics for network performance and quality. **Reduce / All-Reduce:** Collective communication patterns for aggregating values across many workers.