Networking for large scale AI training

Mark Handley — OpenAI & University College London

Costin Raiciu — Broadcom & University Politehnica of Bucharest

Brad Karp — Google & University College London

Goals of today

General overview of what is happening at the frontier of networking for very large AI training clusters in the real world

- Trends – stuff people in the industry know, but not always obvious from outside
- For your research, what capabilities can you assume from current and near-term future hardware
- Open issues

Non-goals

- Lots of companies doing their own thing
  - Lots of secrets
  - We can’t cover the whole industry
  - We can’t even tell you a lot of details of what our companies are doing
- Not a sales talk
  - You don’t need a sales talk
  - We will try to avoid being partisan (but see above)
- Not a research talk
  - Topics we’re talking about are currently in use, or are already making the transition into use

Why is networking for AI training hard?

- Huge scale
- Synchronized computation => synchronized network traffic
- Massively parallel networks
- Very high speed links (too fast for normal software stacks)
- Hardware failures

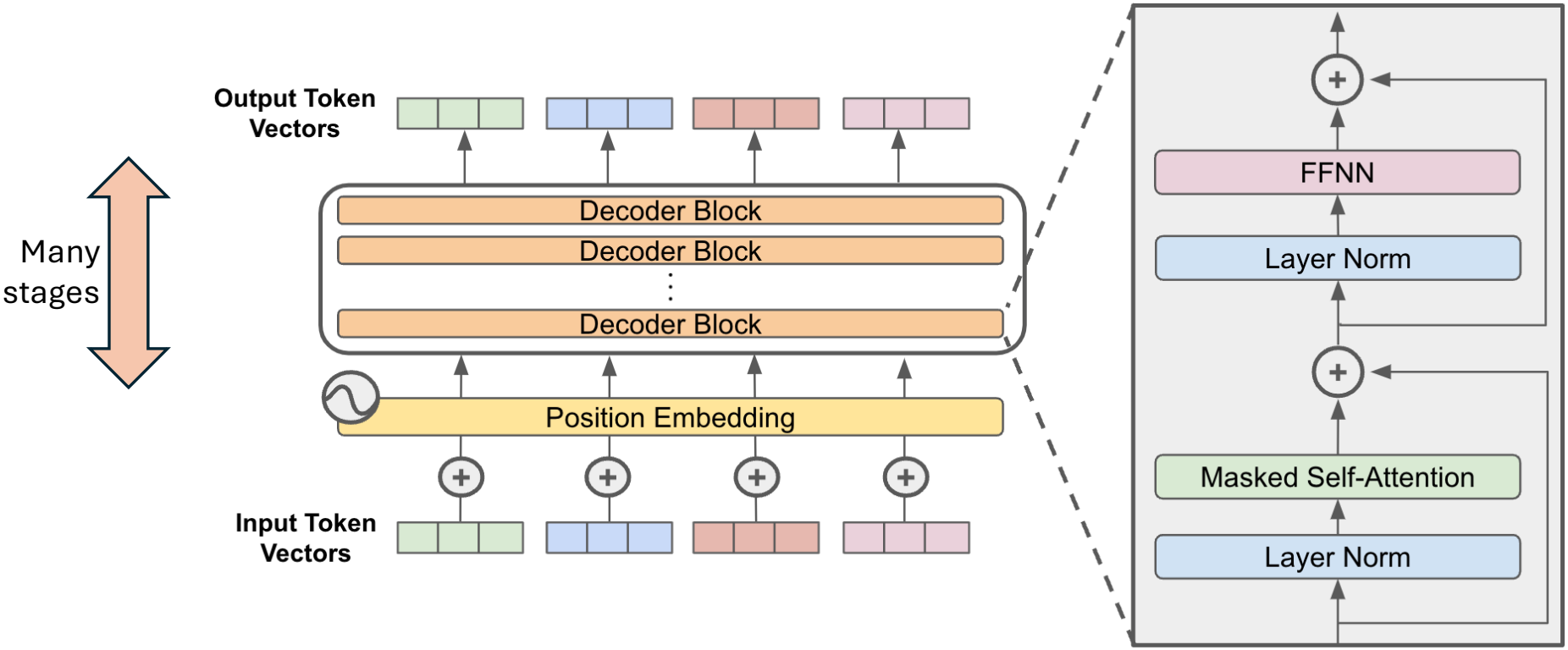
Why is networking for AI training hard?

- Workload
  - How to parallelize a large language model?
  - AI traffic patterns
- Topology
  - What is an AI accelerator?
  - How can we connect 100K GPUs?
- Transport
  - How can we best use the network to carry the workload?
  - How to cope with network failures

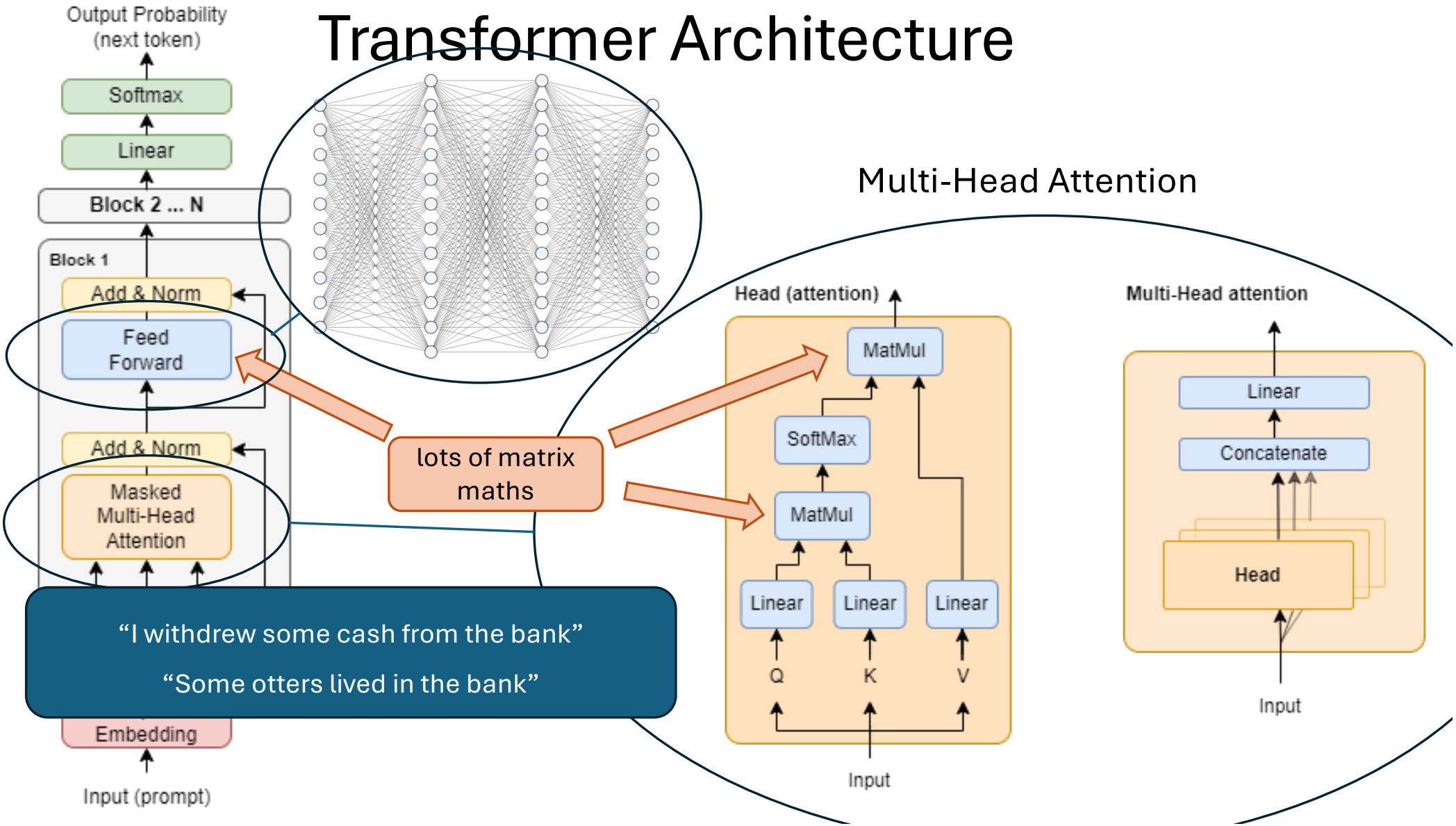
Outline

- Workload: what is a transformer?
- Problems of scale
- Parallelizing the workload
- Collective communication
- Transport: RDMA
- Scale-up vs scale-out
- Topology: load balancing
- Trimming
- Transport: UltraEthernet
- Congestion Signalling
- Topology: planes and rails
- Routing and fault tolerance
- Hands-on: htsim simulation

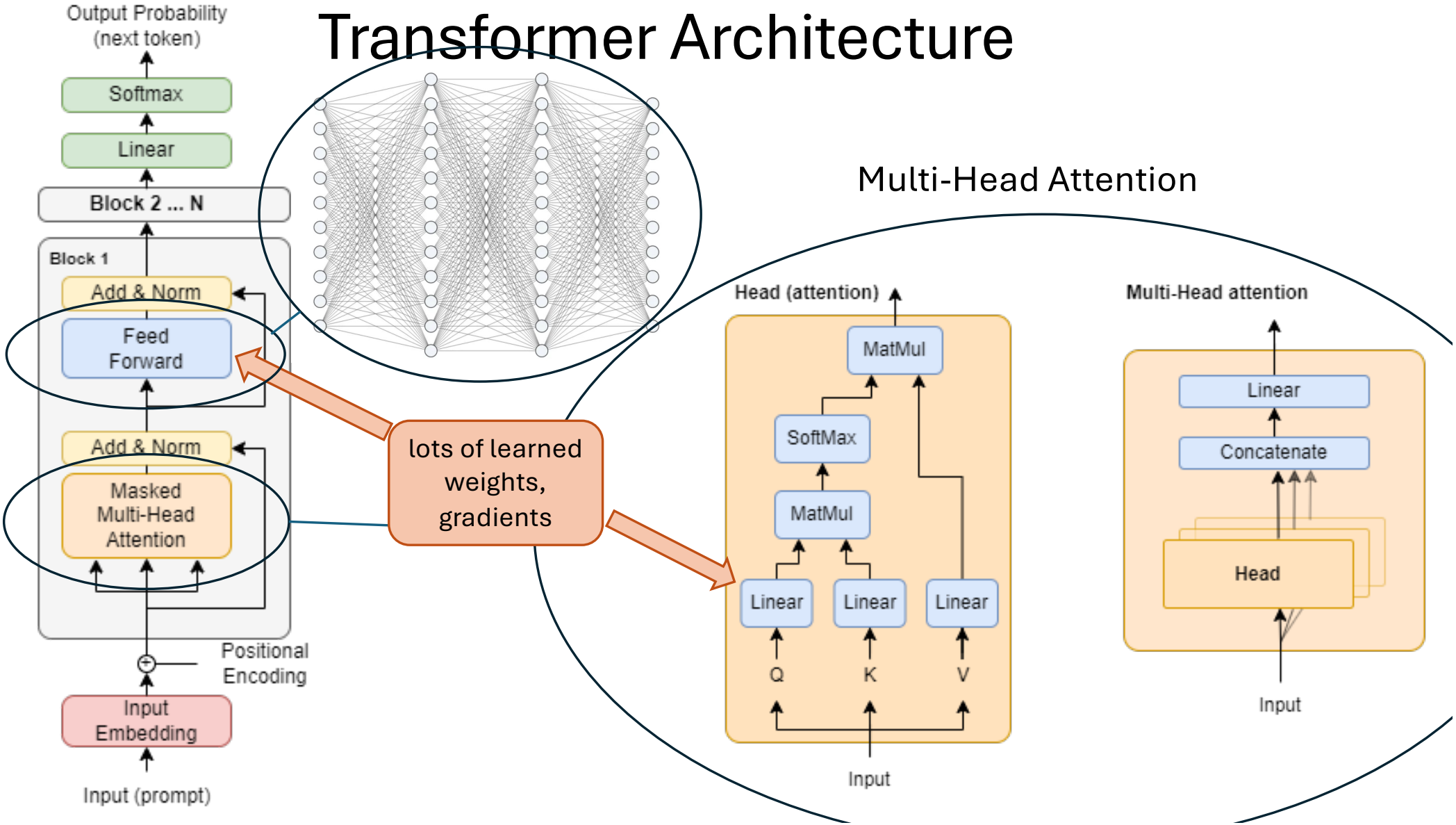
Transformer Architecture



Transformer Architecture

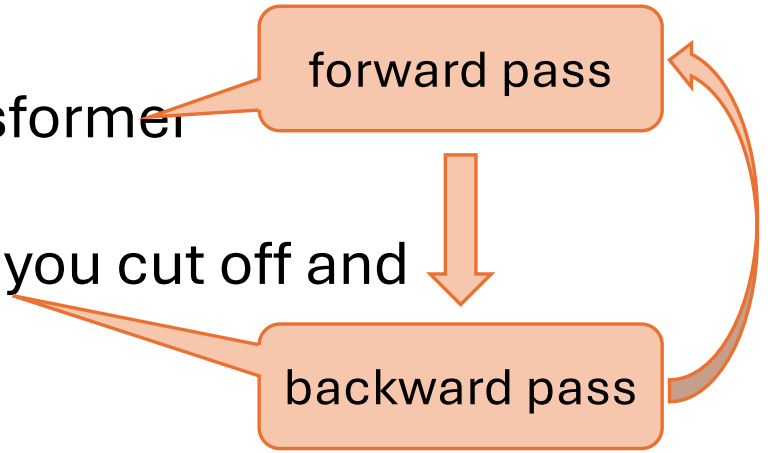


Transformer Architecture



Basic training idea

1. Gather lots of text
2. Strip the last word from that text
3. Feed remaining text as input into the transformer
   - At the output, it predicts the next word
4. Check if the prediction matches the word you cut off and backpropagate the error



- Back propagation involves using gradients stored on the forward pass to update the weights during the reverse pass based on how wrong the result was.

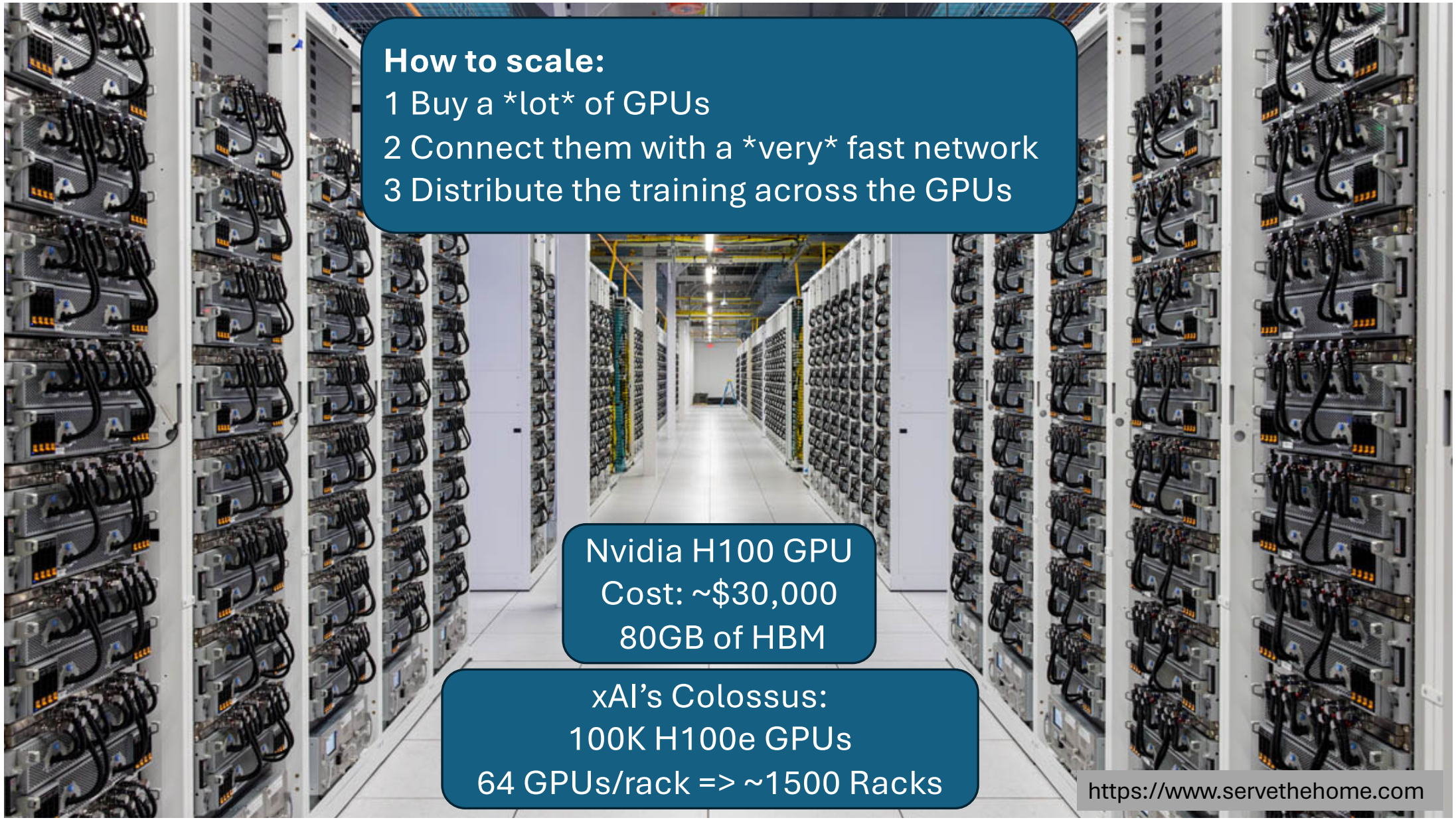
Problems of scale

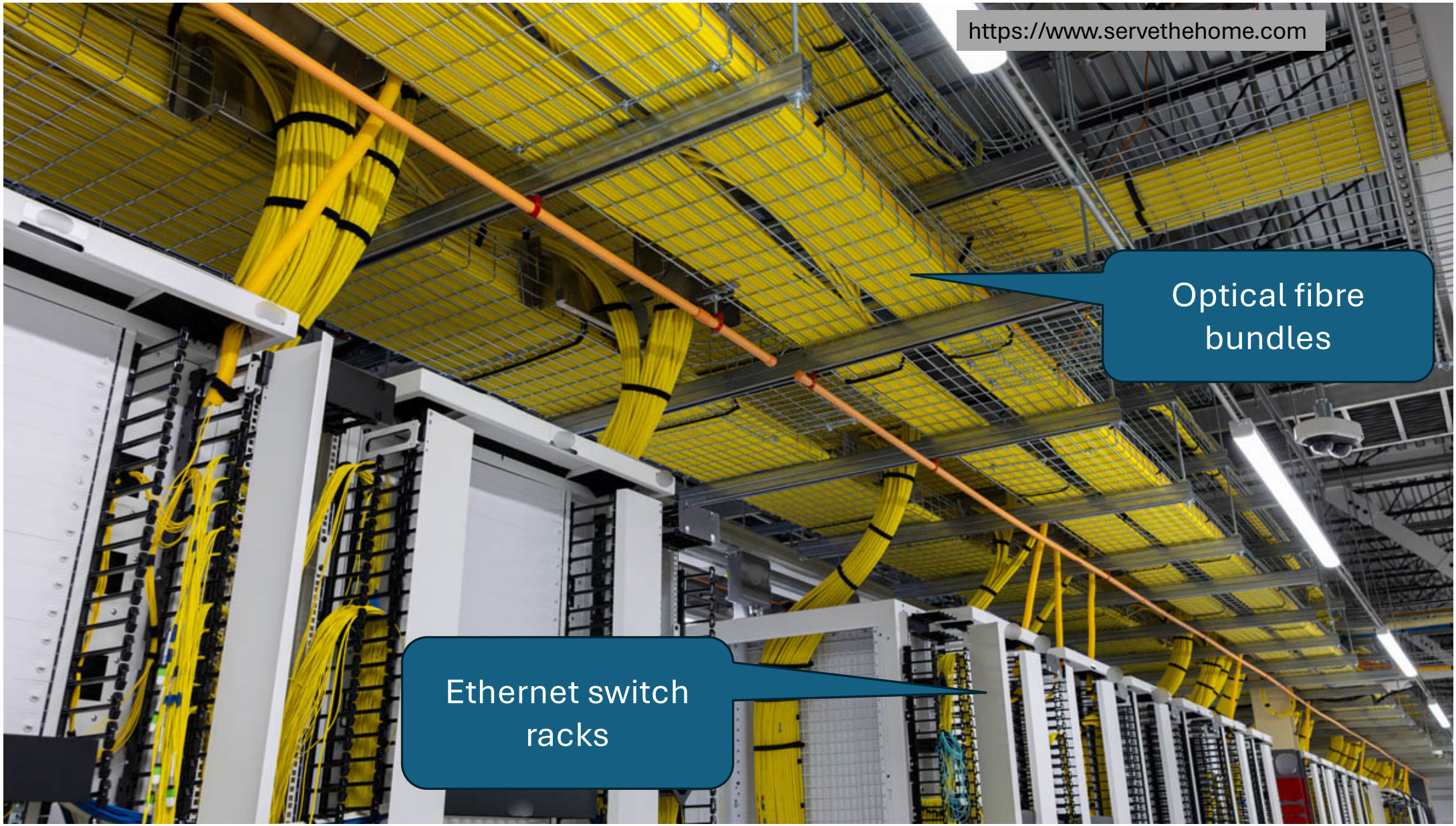
- Way too many parameters to hold in one GPU
  - weights
  - gradients
  - optimizer state

- Way too much data (all of the Internet!)
  - takes forever to train if we feed the data in sequentially

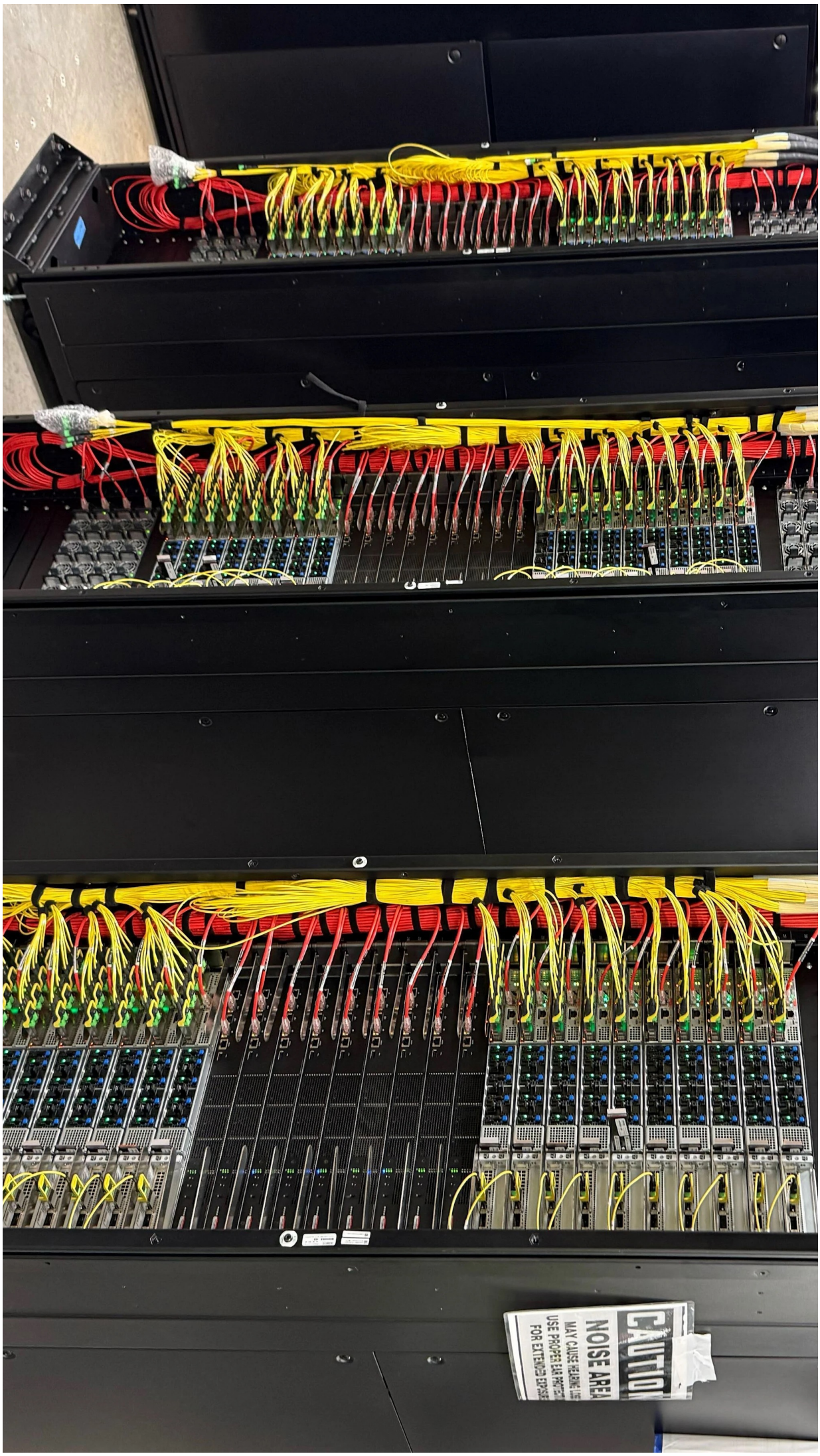
Llama 4 behemoth uses ~2 trillion parameters

Llama 4 trained on > 30 trillion tokens


How to scale:
1 Buy a *lot* of GPUs
2 Connect them with a *very* fast network
3 Distribute the training across the GPUs
Nvidia H100 GPU
Cost: ~$30,000
80GB of HBM
xAI's Colossus:
100K H100e GPUs
64 GPUs/rack => ~1500 Racks
https://www.servethehome.com


https://www.servethehome.com
Optical fibre bundles
Ethernet switch racks


https://openai.com/index/stargate-advances-with-partnership-with-oracle/


CAUTION
NOISE AREA
MAY CAUSE HEARING
USE PROPER EAR PROTECTION
FOR EXTENDED EXPOSURE

Problems of scale

- Way too many parameters to hold in one GPU
  - weights
  - gradients
  - optimizer state

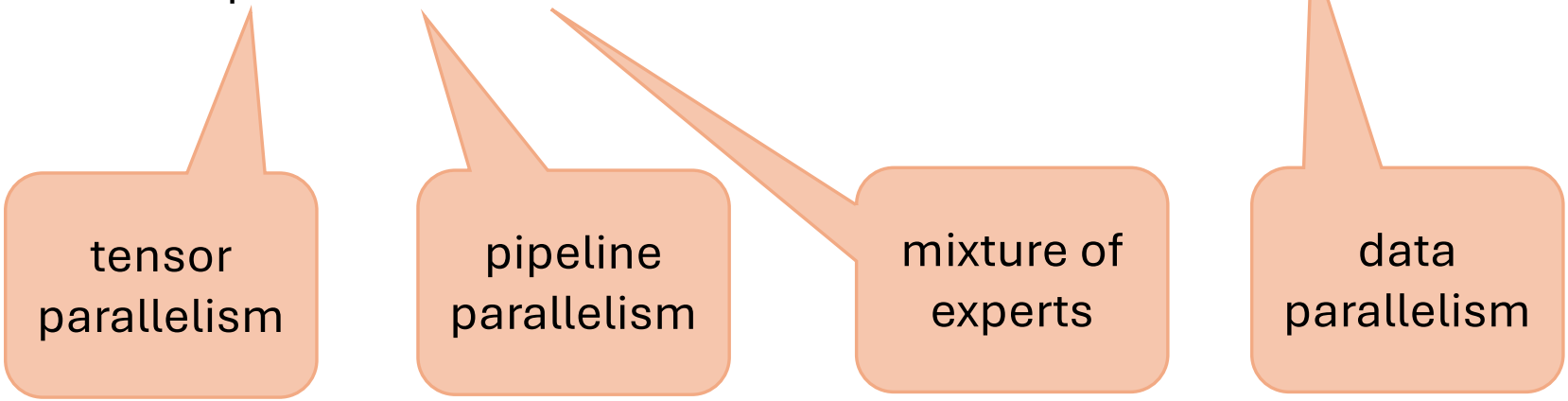
- Way too much data (all of the Internet!)
  - takes forever to train if we feed the data in sequentially

Llama 4 behemoth uses ~2 trillion parameters

Llama 4 trained on > 30 trillion tokens

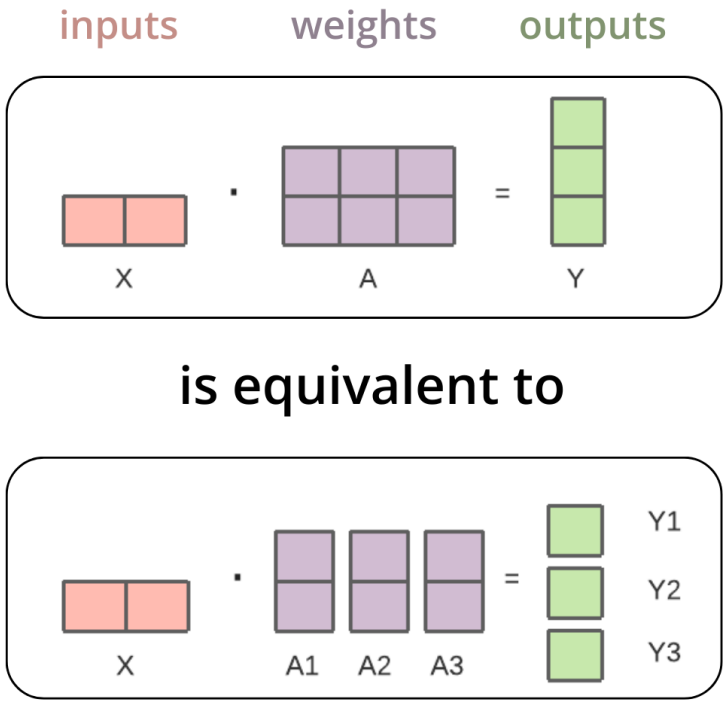
Problems of scale

- Way too many parameters to hold in one GPU
  - weights
  - gradients
  - optimizer state
- Way too much data (all of the Internet!)
  - takes forever to train if we feed the data in sequentially



Tensor Parallel

"my matrix is too big"

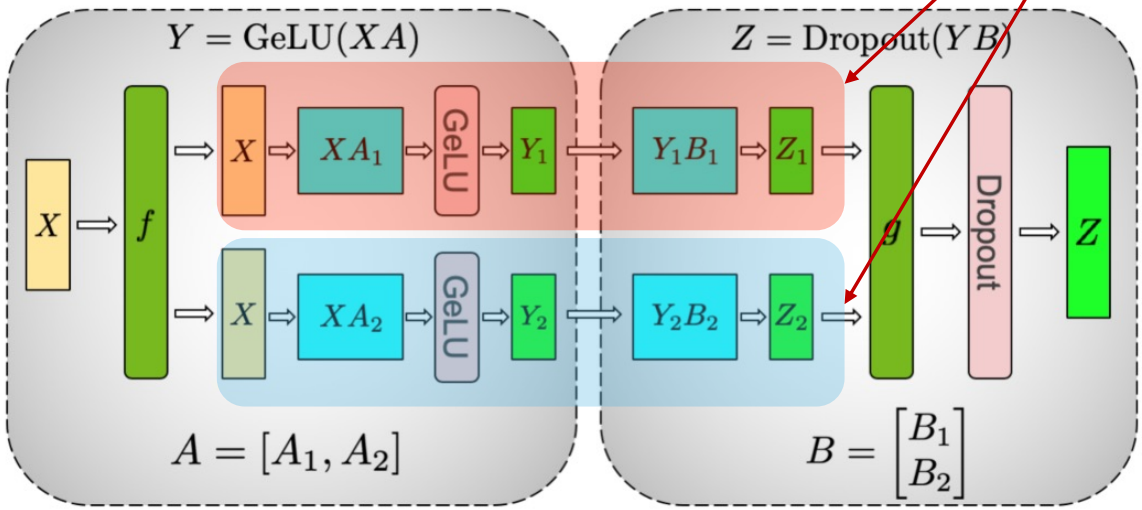


- Split weights (A) across multiple GPUs by column.
- Send same inputs (X) to all GPUs.
- Multiple columns separately
- Recombine output vector (Y)

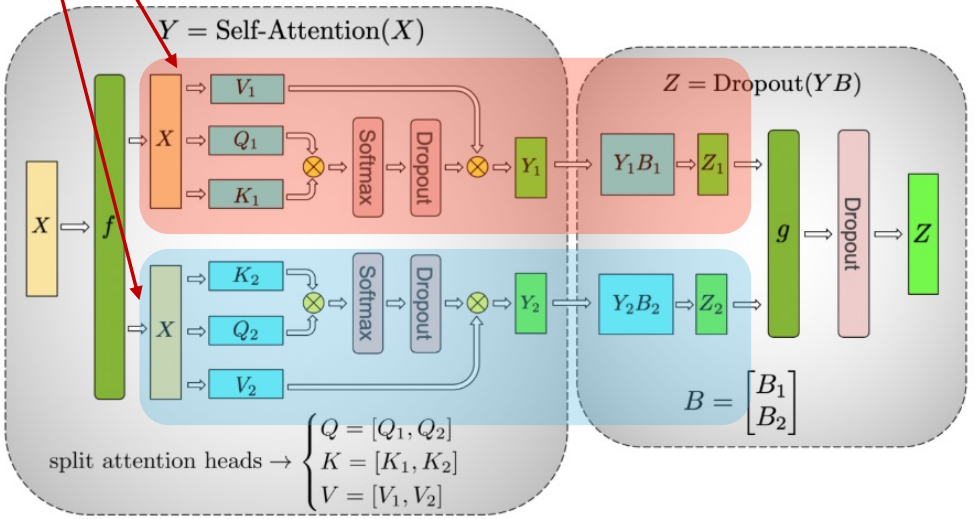
Tensor/Model Parallel

From Megatron-LM (Nvidia)

Runs on separate GPUs



(a) MLP



(b) Self-Attention

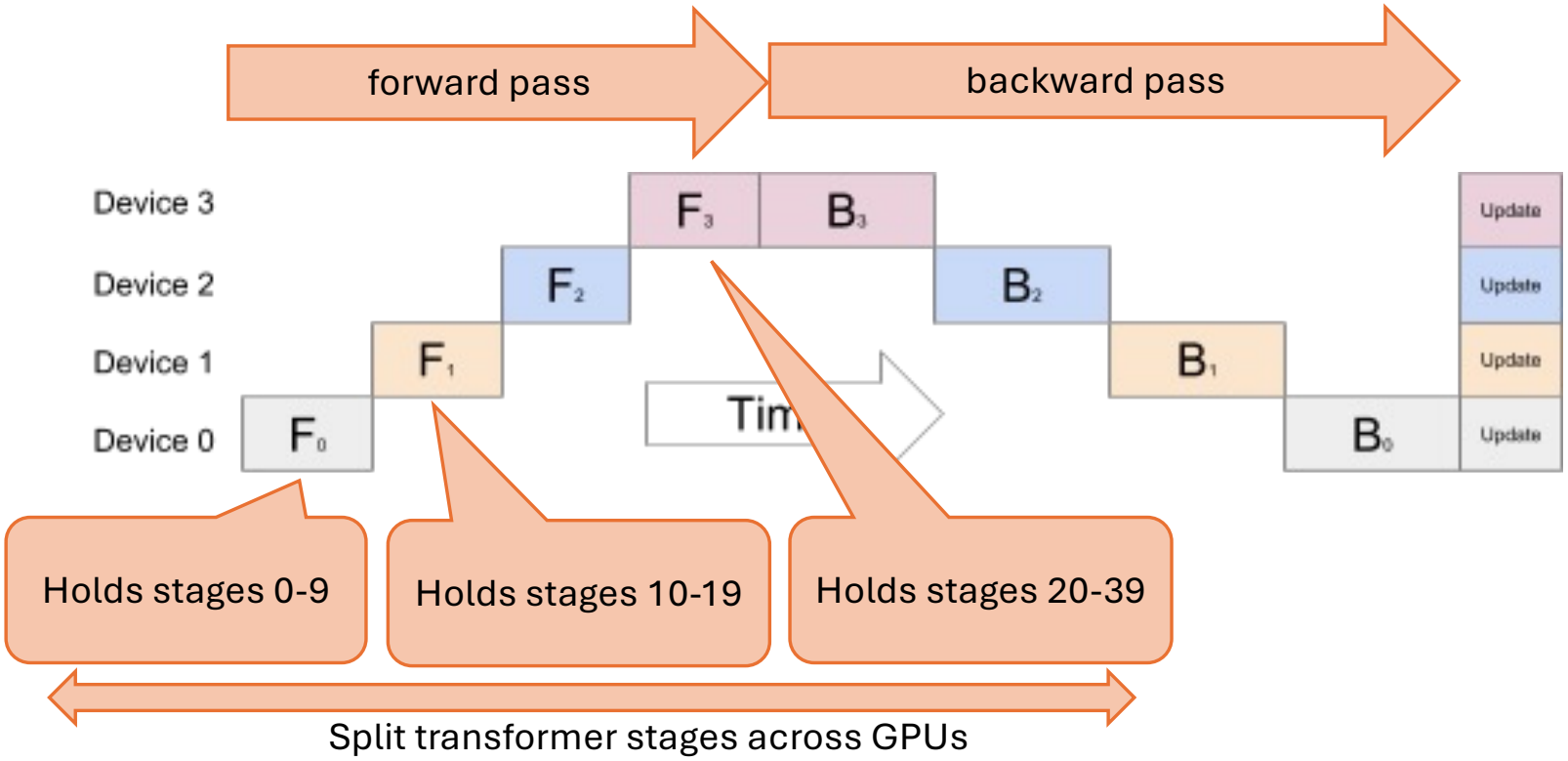
In forward pass, f is broadcast, g is all-reduce

In reverse pass, g is broadcast, f is all-reduce

average (sum) the data, send the average to everyone

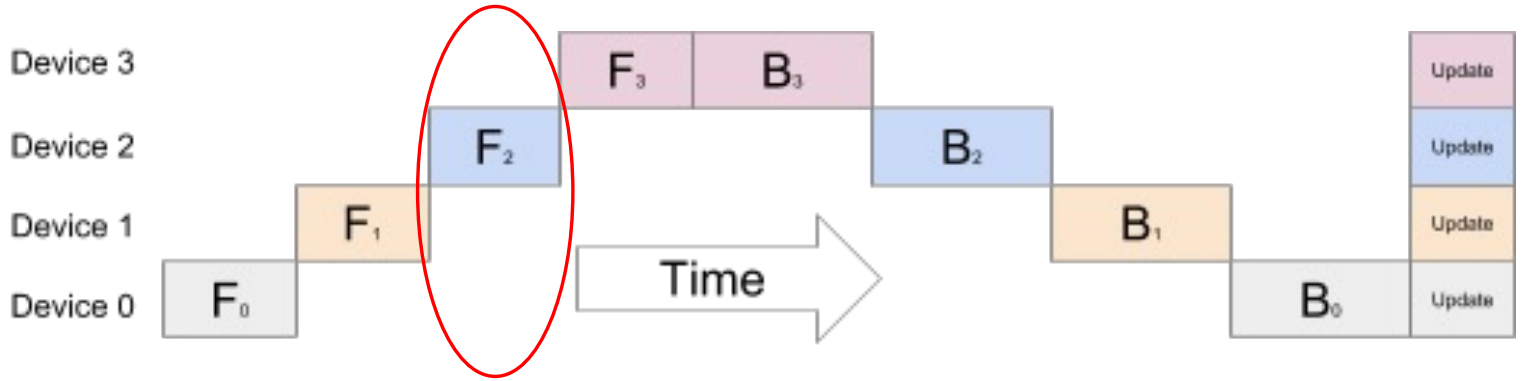
Pipeline Parallel

"I can't fit all the stages in RAM" (Google Gpipe)



Pipeline Parallel

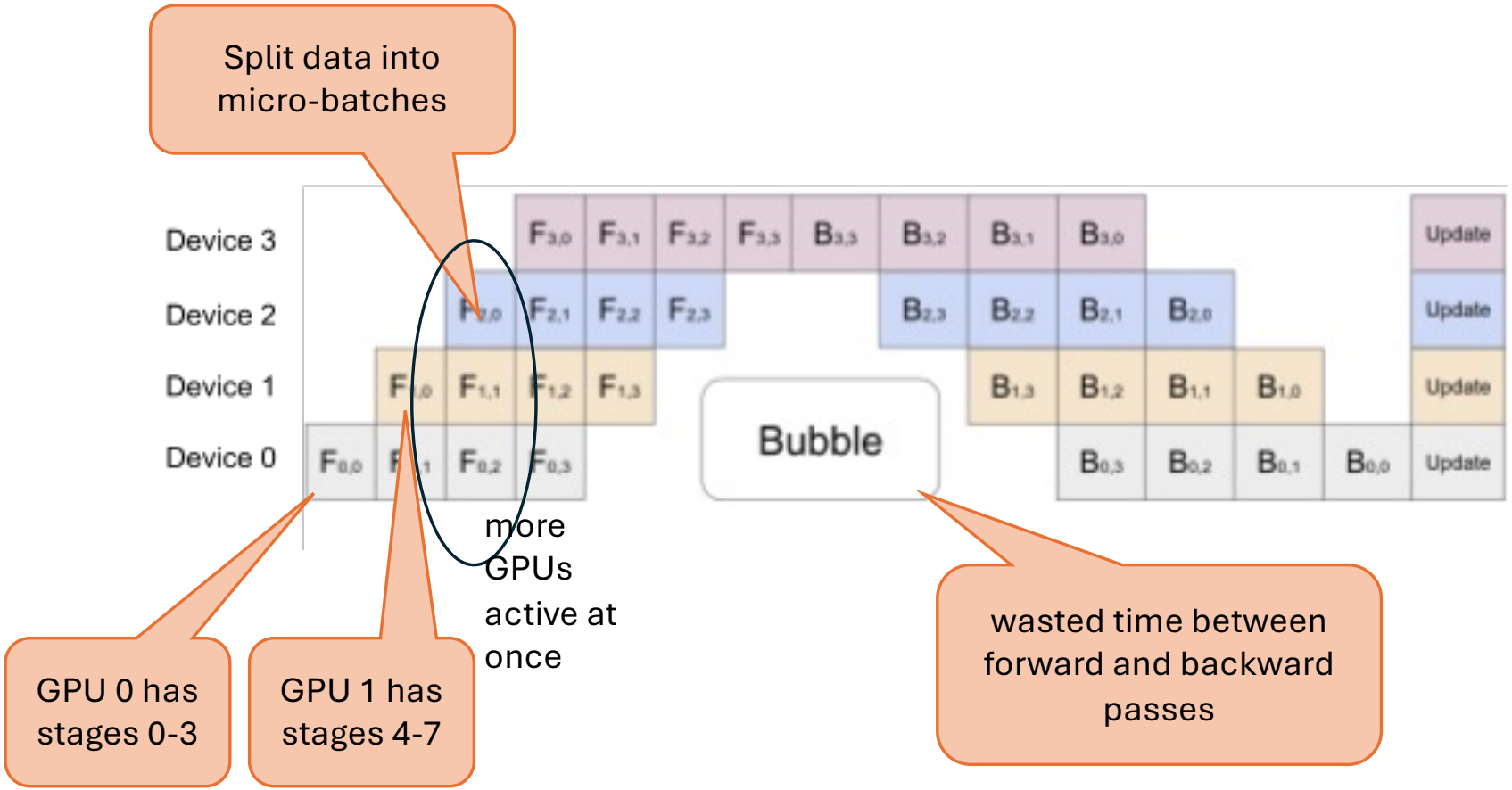
“I can’t fit all the layers in RAM” (Google Gpipe)



Only one GPU active at a time!

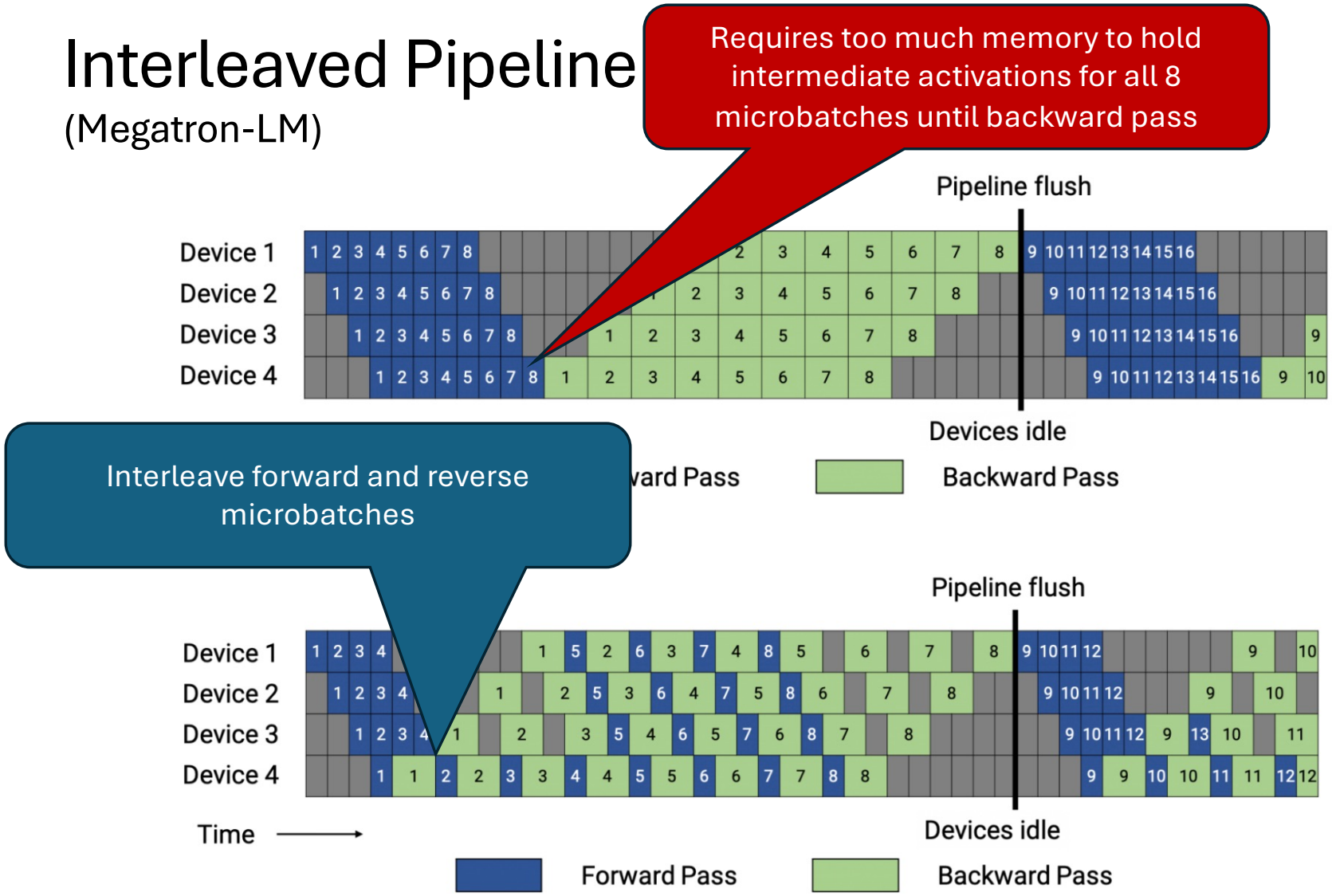
Pipeline Parallel

"I can't fit all the layers in RAM" (Google Gpipe)



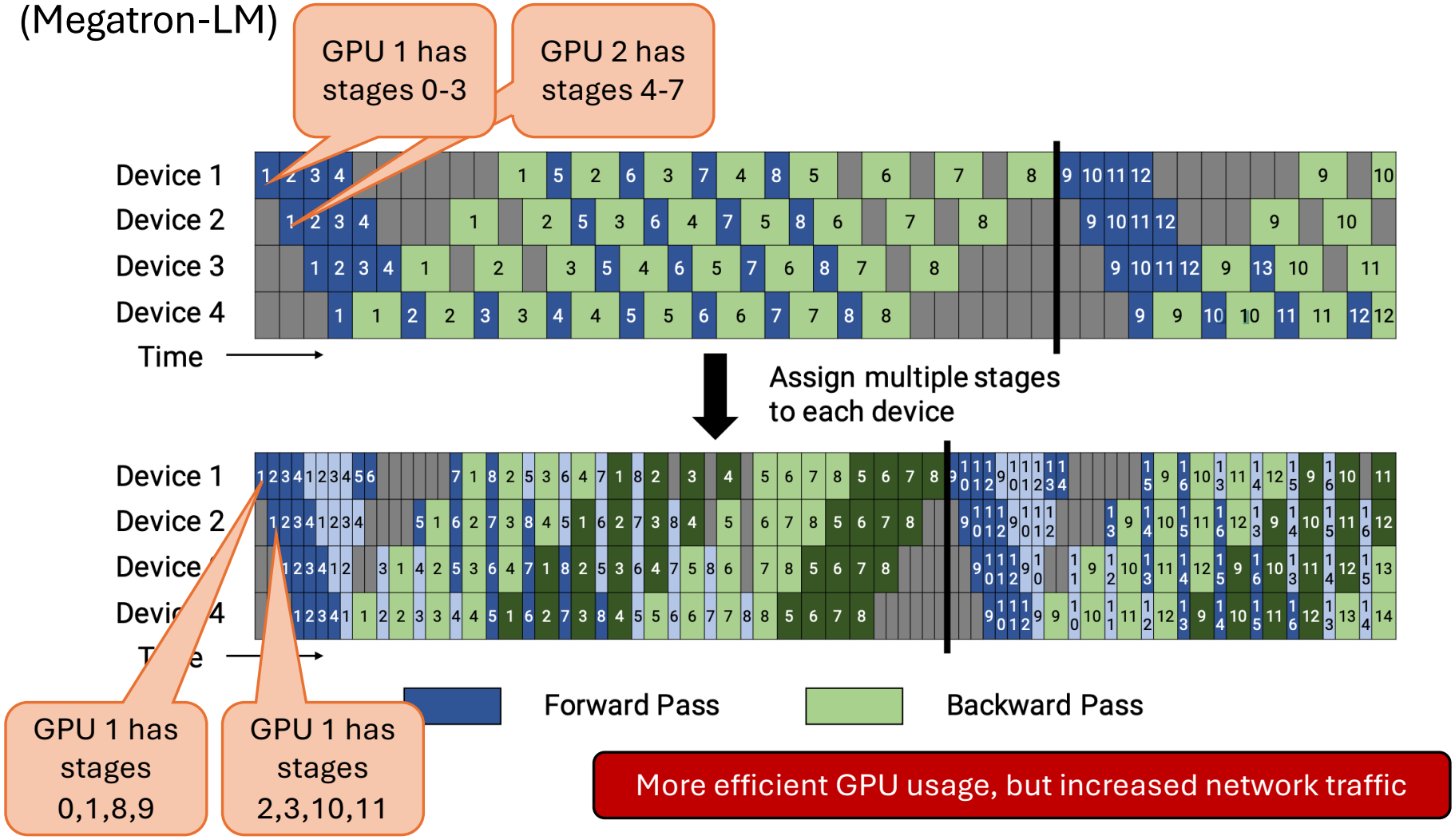
Interleaved Pipeline

(Megatron-LM)



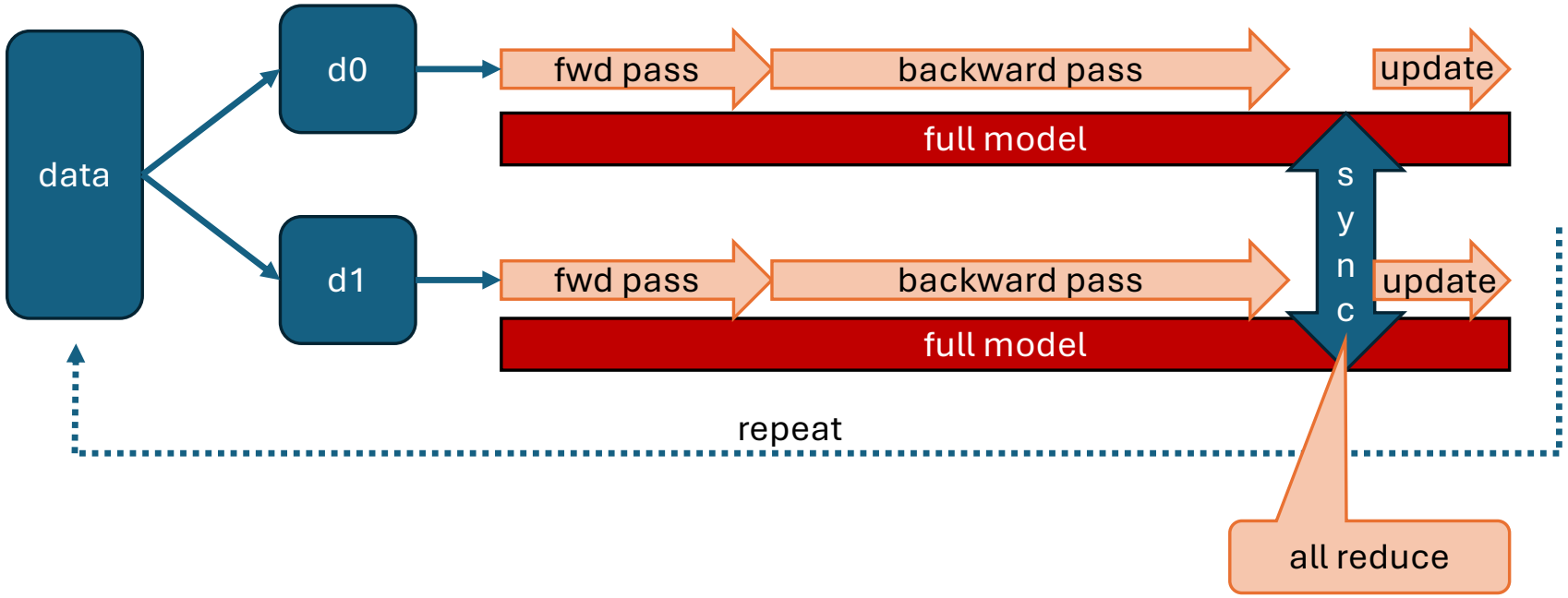
Interleaved Pipeline Parallel

(Megatron-LM)



Data Parallel

"It takes too long to process all the Internet if I do it sequentially"

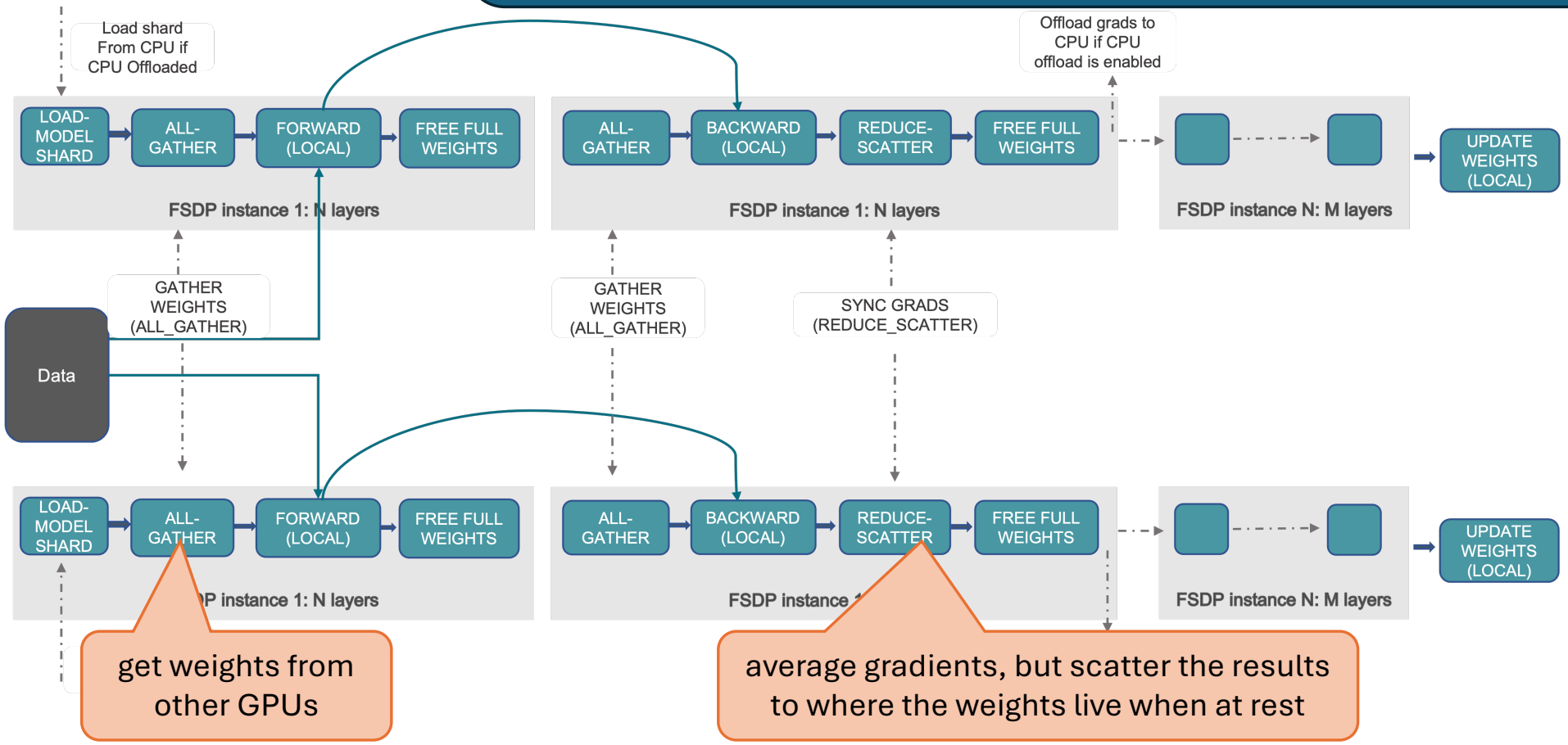


average the gradients, so all GPUs update their model in the same way

Fully Sharded

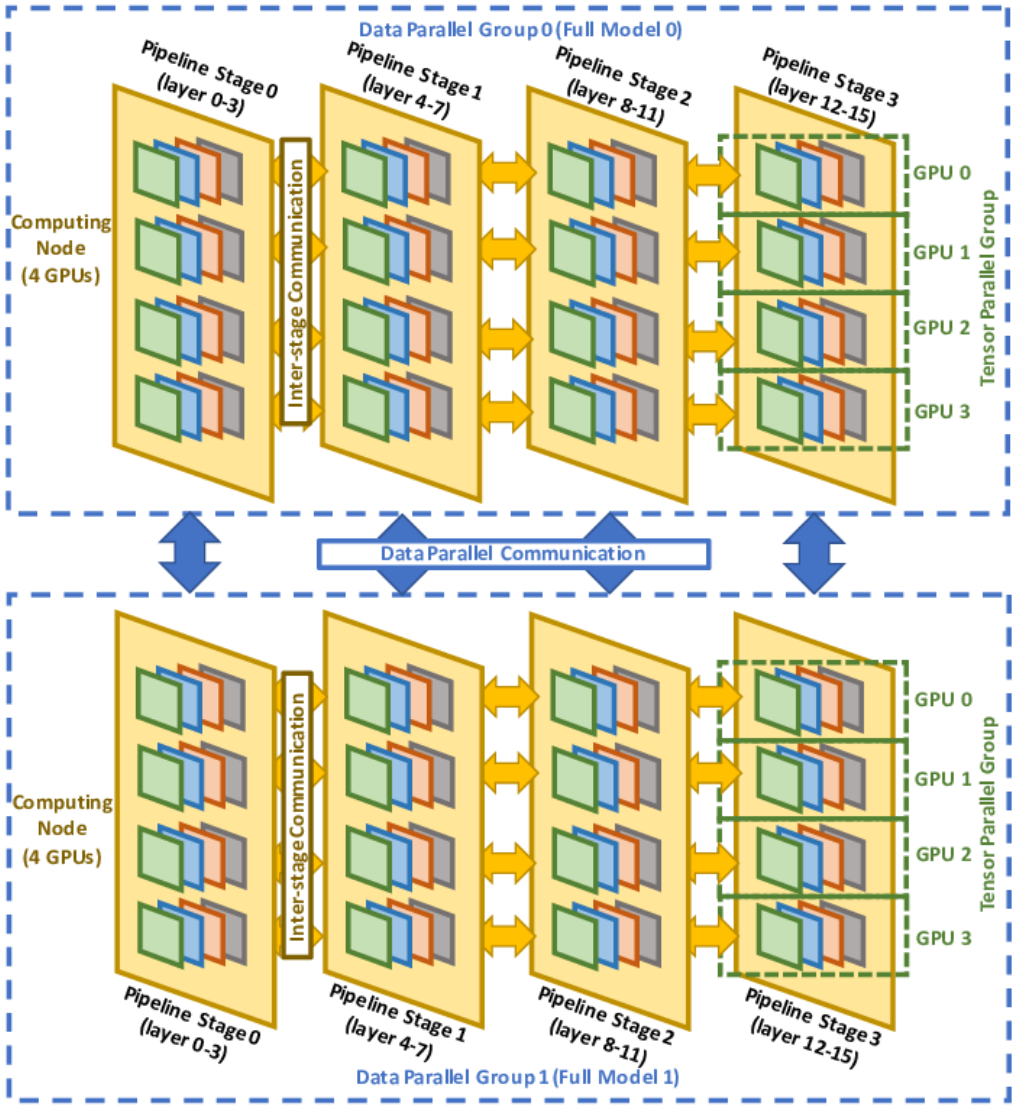
PyTorch. Combine data p

- Can trade-off memory against network traffic.
- Each GPU holds the master copy for one shard of model
- Fetch other shards on demand
- Compute, sync, forget the parts you don't own.
- Meanwhile loading shard for next stage of compute.



3d parallelism

- For large models, need to do all types of parallelism simultaneously
- Also other stuff like “mixture of experts” to reduce density of models



Communication patterns

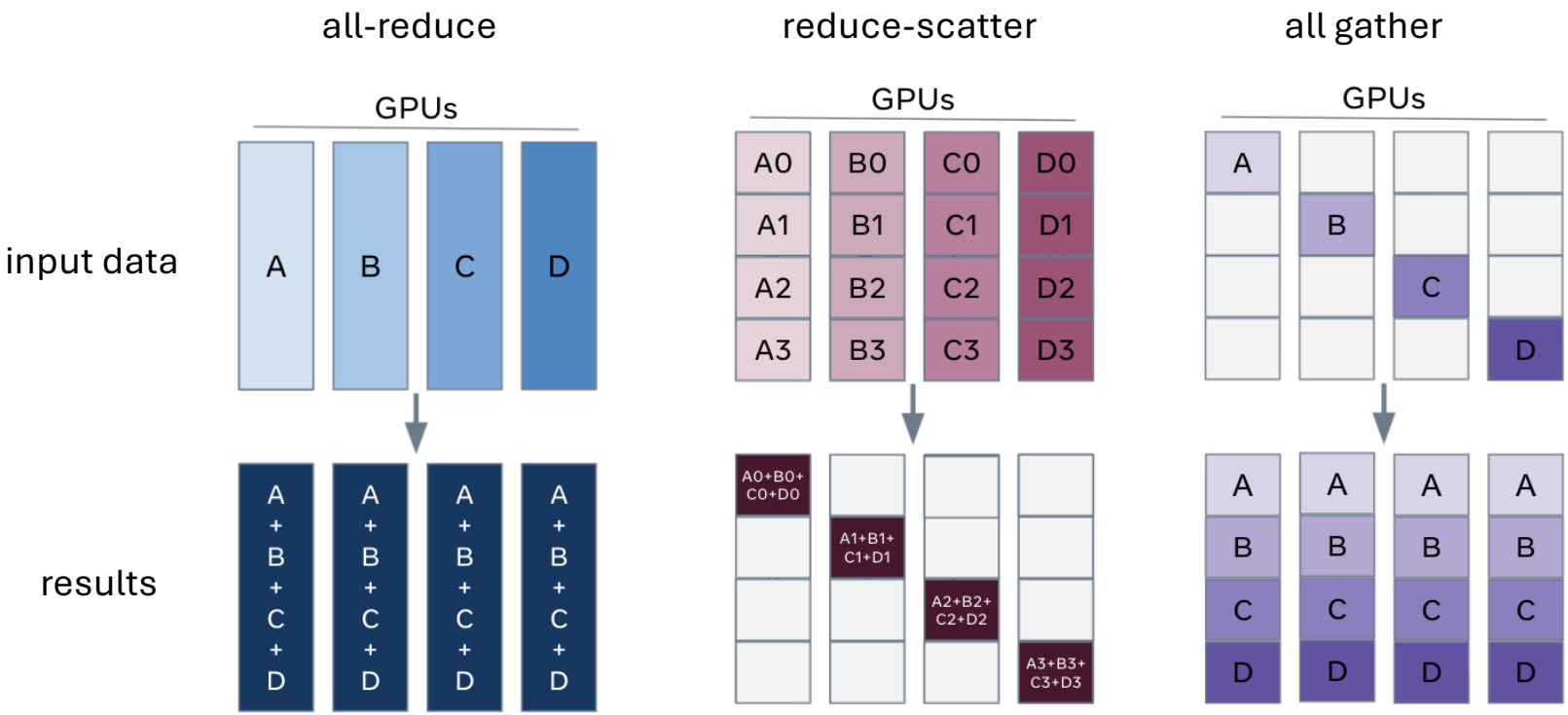
Communication is organized into collectives:

- all-gather
- all-reduce
- reduce-scatter
- all-to-all

Usually implemented as collective libraries such as NCCL

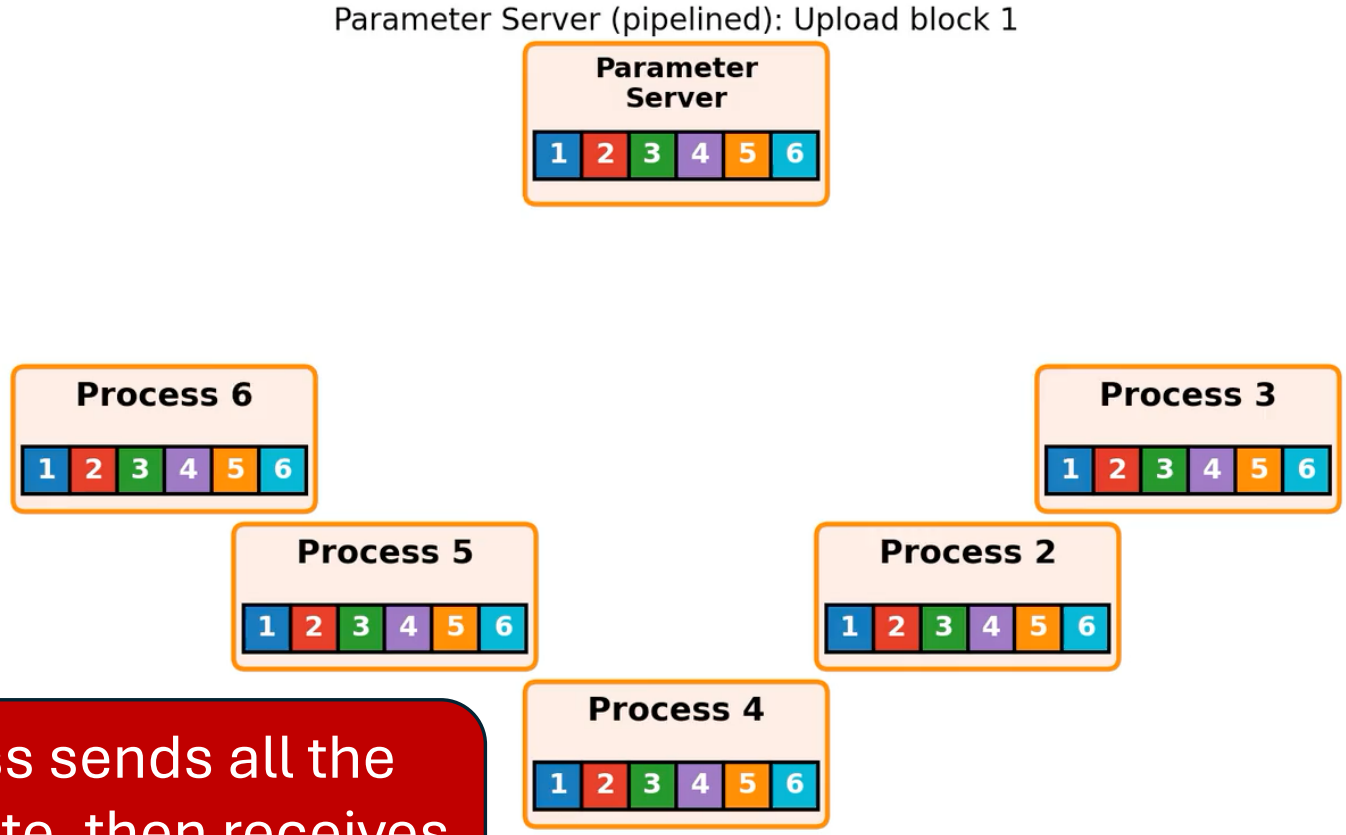
- The computation is synchronous across GPUs, so the start of each collective is synchronized too.
  - Everyone starts exactly together
  - Really bad for the network!
- If one is slow, everyone needs to wait for them!

Collectives



All-Reduce (parameter server)

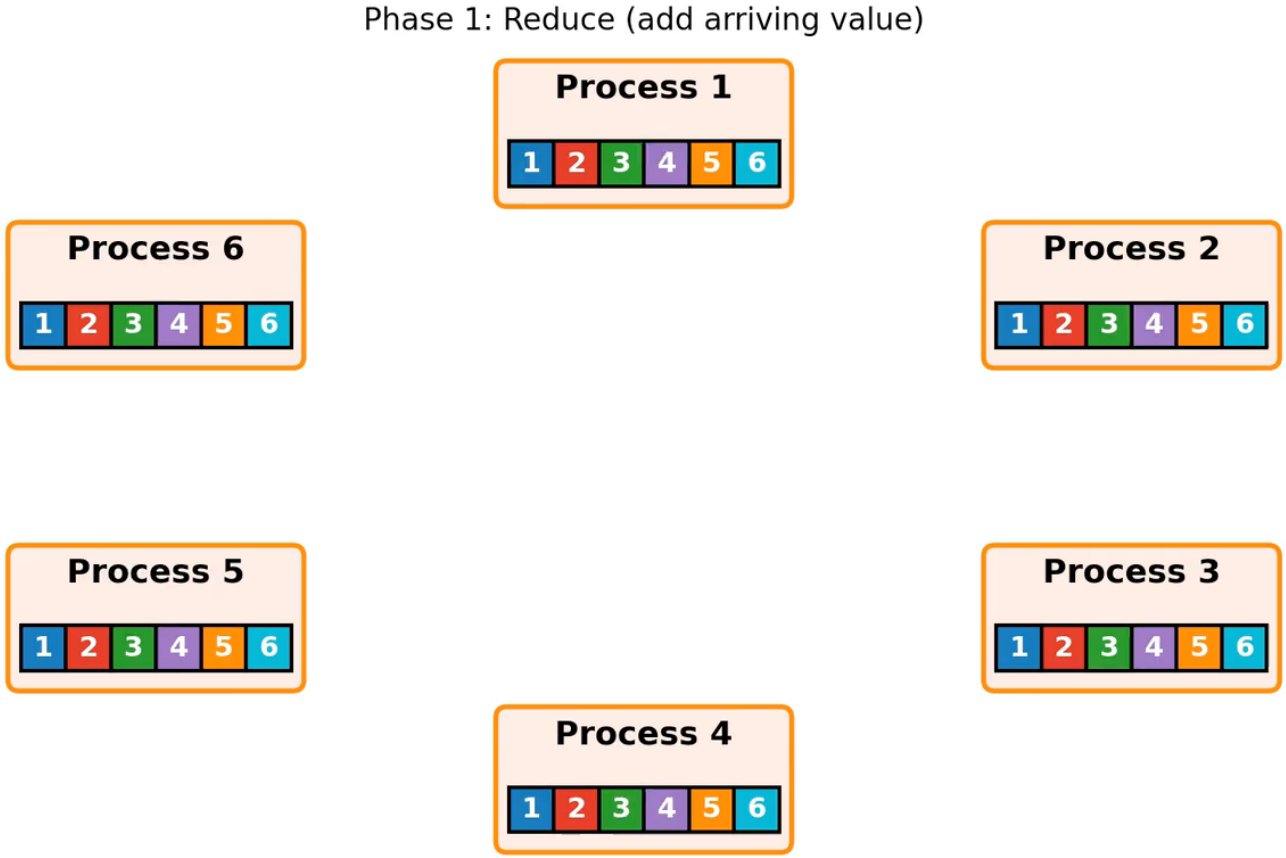
To see the animations, please refer to the powerpoint version of these slides posted on the Sigcomm Tutorial Github repository.



Each process sends all the data at $1/n^{th}$ rate, then receives all the data at $1/n^{th}$ rate.

Ring All-Reduce (NCCL)

To see the animations, please refer to the powerpoint version of these slides posted on the Sigcomm Tutorial Github repository.



Ring All-reduce

- Each GPU sends to one other GPU as fast as it can for the whole duration of the all-reduce.
- Each GPU receives from one other GPU as fast as it can.
- Additions are overlapped with data transfers.
  - Start adding as soon as data starts arriving.

Each process sends all the data twice at full rate no matter the number of processes (receives are overlapped with sends)

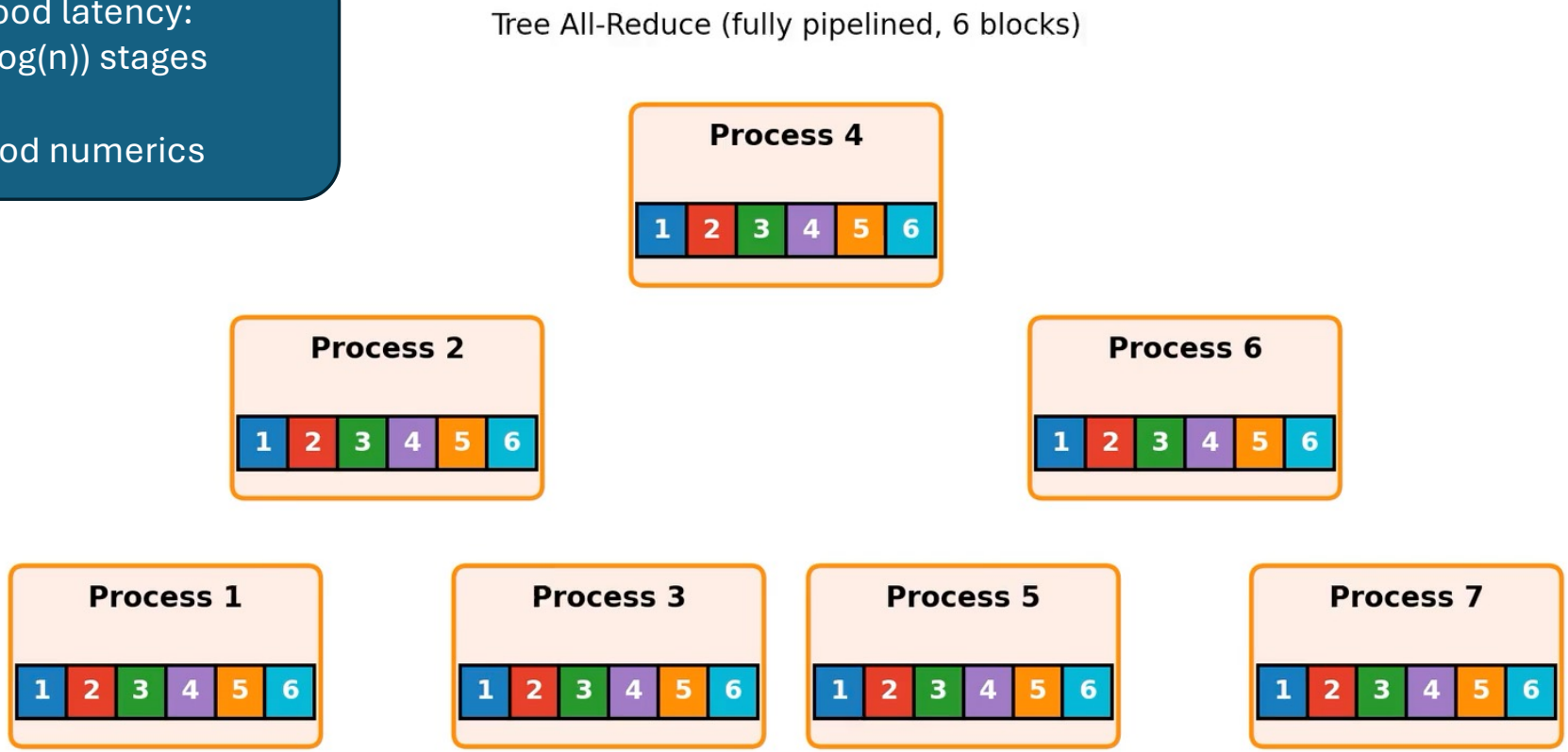
If anyone goes slow, the whole collective slows down.

Tree All-Reduce (pipelined)

To see the animations, please refer to the powerpoint version of these slides posted on the Sigcomm Tutorial Github repository.

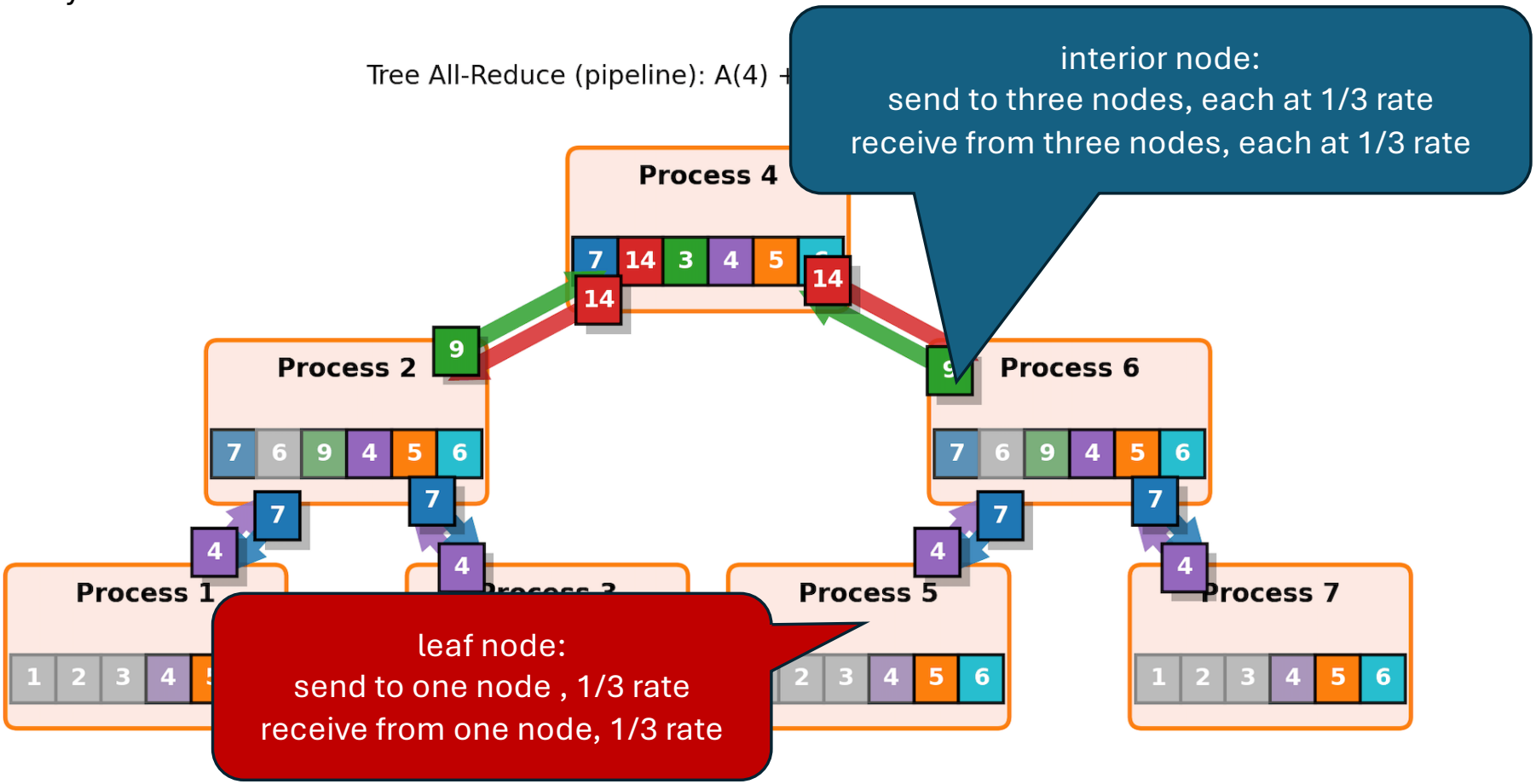
Good latency:
O(log(n)) stages

Good numerics



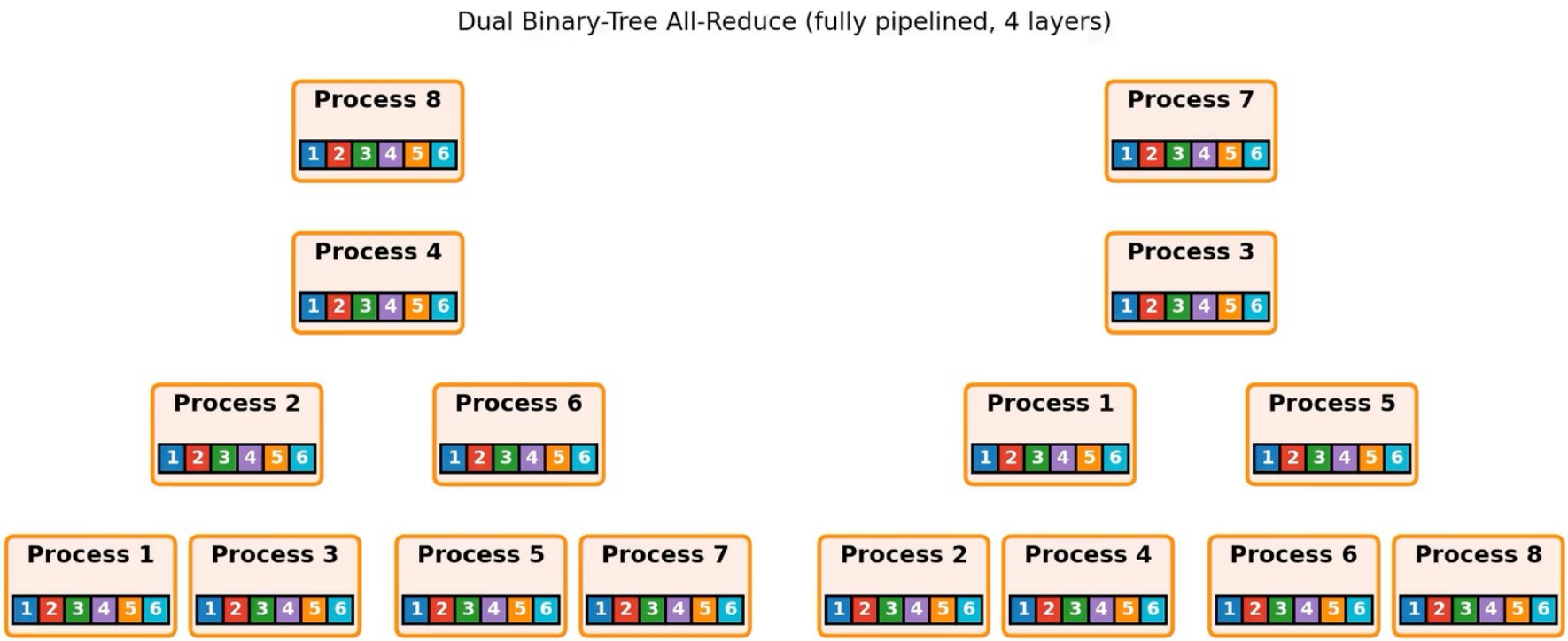
Tree All-Reduce

To see the animations, please refer to the powerpoint version of these slides posted on the Sigcomm Tutorial Github repository.



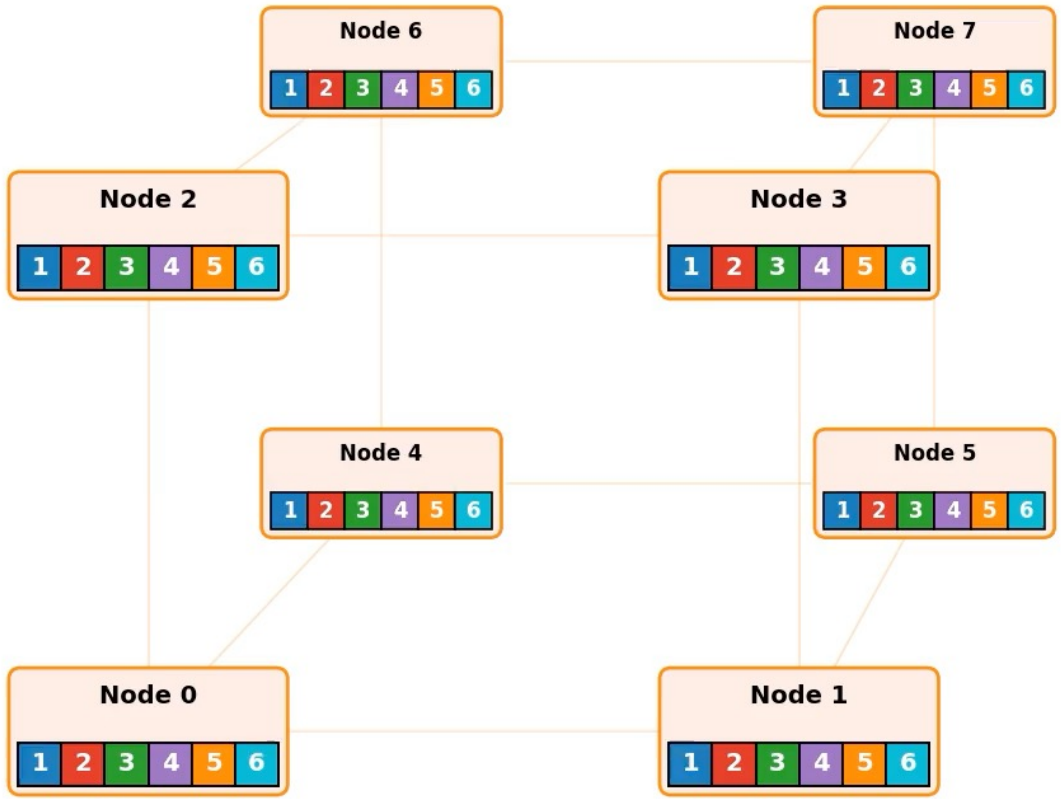
Dual Binary Tree

To see the animations, please refer to the powerpoint version of these slides posted on the Sigcomm Tutorial Github repository.



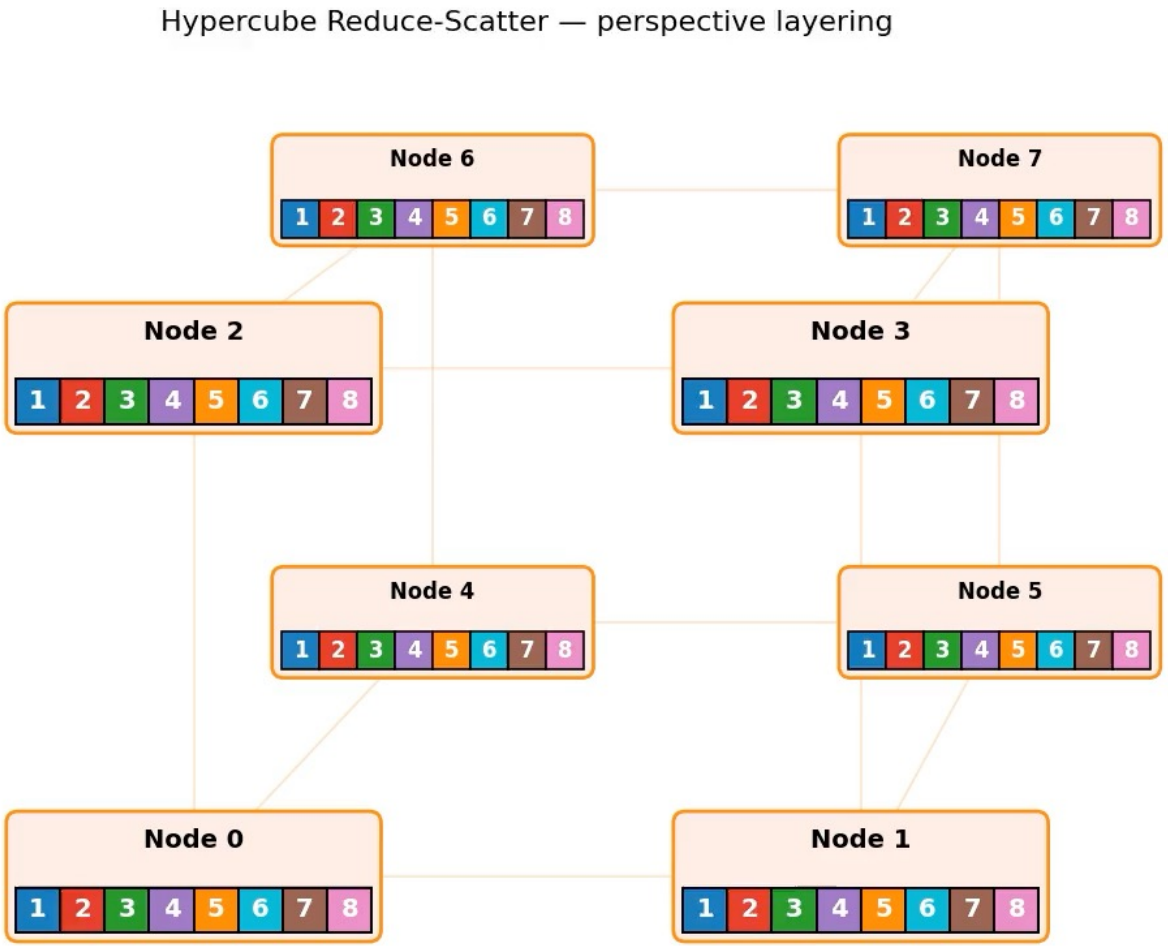
All-reduce (hypercube)

To see the animations, please refer to the powerpoint version of these slides posted on the Sigcomm Tutorial Github repository.

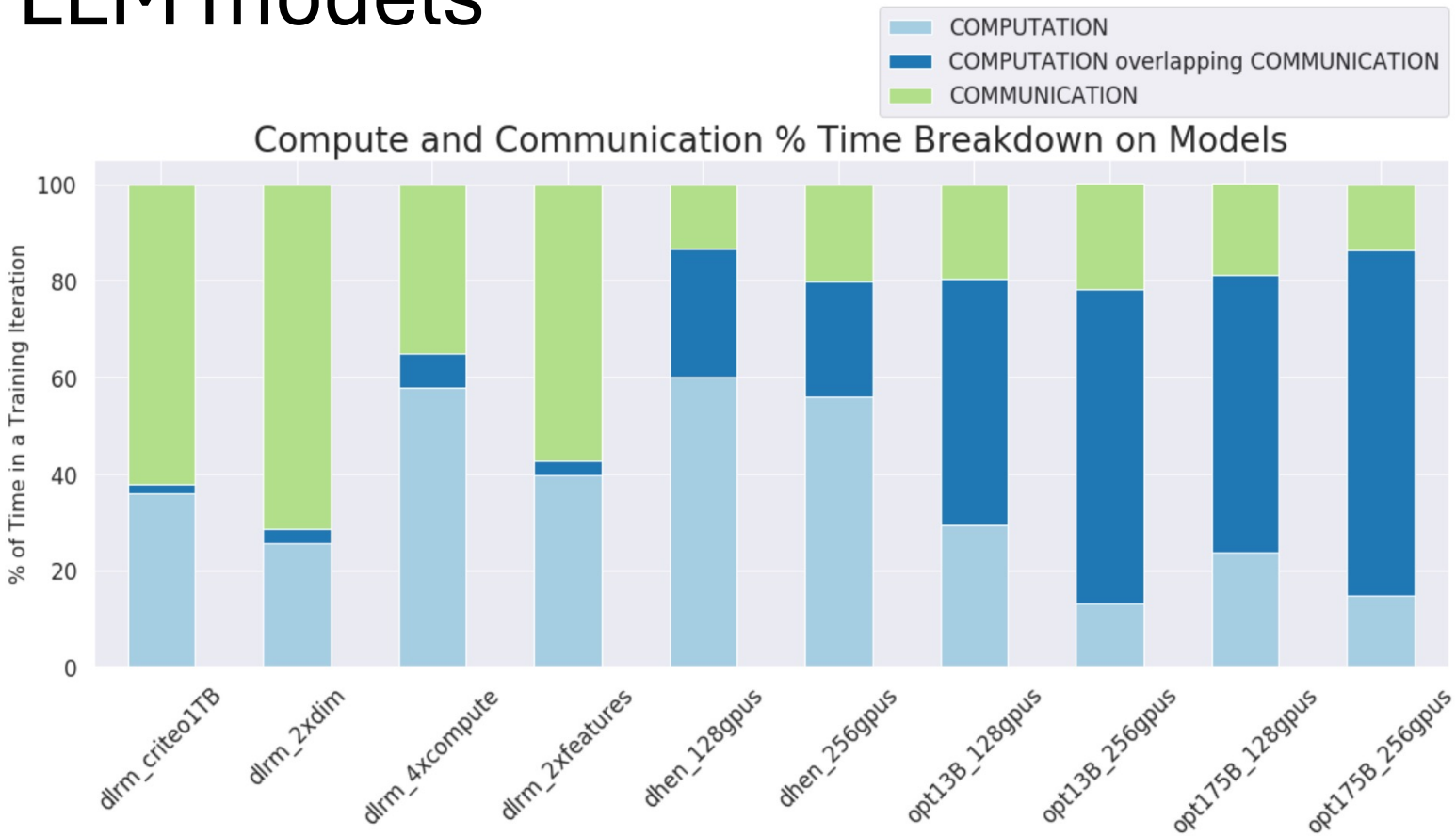


Reduce-scatter (hypercube)

To see the animations, please refer to the powerpoint version of these slides posted on the Sigcomm Tutorial Github repository.



Exposed communication in Recommendation and LLM models



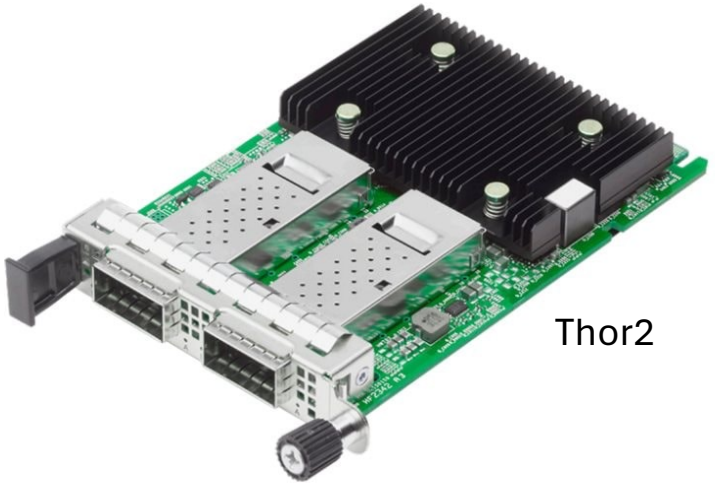
Source: Meta, OCP 2022

How to minimize exposed communication

- Interleave processing of different parts of the data
  - Eg: Megatron's Interleaved Pipeline Parallel
- Minimise tail latency in collectives
  - Each round of communication is limited by the last transfer to finish
  - We only care about last flow completion time (FCT).
- With 3d parallelism, can be doing two collectives at once.
  - Ensure they co-exist gracefully.

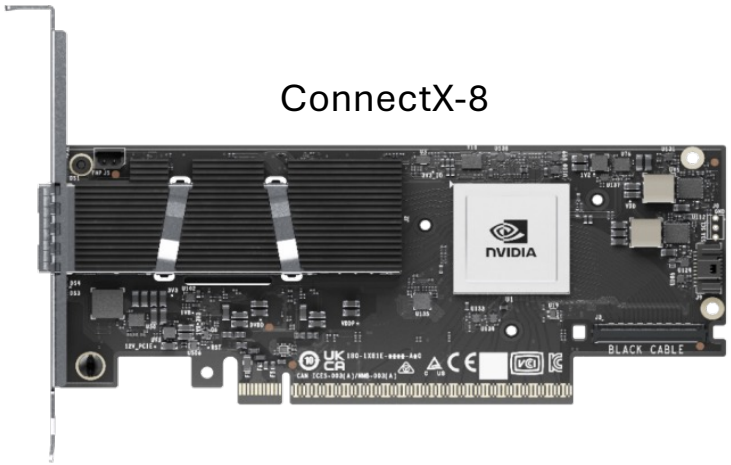
Ethernet NICs

- Previous generation (400Gb/s) RoCE NICs:
  - Nvidia ConnectX-7
  - Broadcom Thor2
- Programmable 400Gb/s NICs/DPUs:
  - Nvidia Bluefield 3
  - AMD Pollara, Salina
- Current/Next generation (800Gb/s):
  - Nvidia ConnectX-8
  - soon, Broadcom, AMD, ...



Thor2

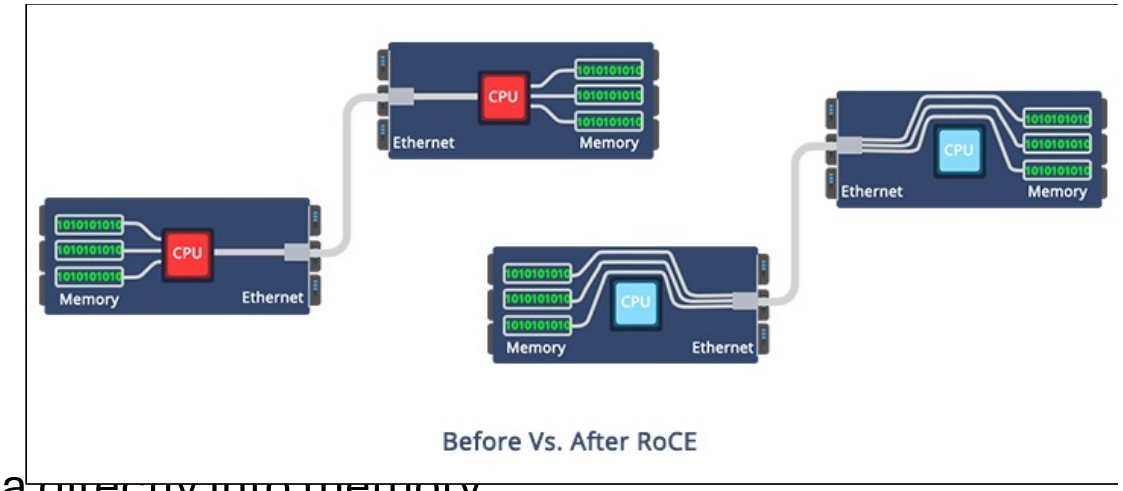
ConnectX-8



Transport Protocols

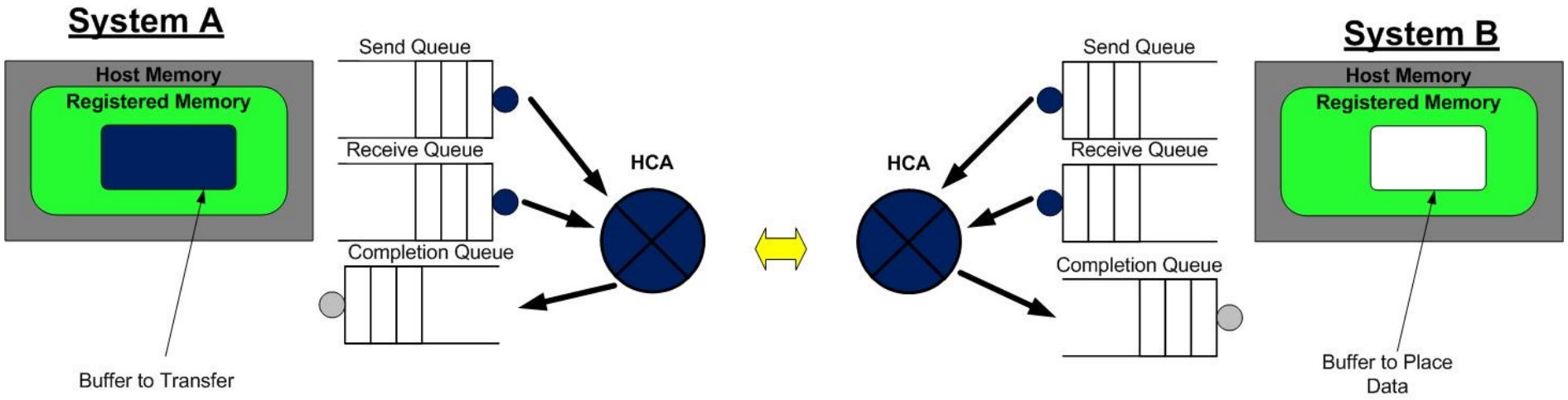
- Current linkspeeds in AI training clusters:
  - 400Gb/s, currently moving to 800Gb/s
- We need a reliable transfer, but can’t use TCP:
  - 400Gb/s or 800Gb/s is too fast to process in software on the CPU
  - OS network stack would create too much latency
  - Really want to do GPU to GPU transfers

Transport Protocols: RDMA



- Remote Direct Memory Access
  - DMA: old technique, hardware places data directly into memory
  - RDMA: place data directly into memory on a remote computer
- Local CPU says what data to write where on which computer, and hands the task to the local NIC
- Local NIC sends to remote NIC. Remote NIC writes to memory.
- RDMA originated on Infiniband networks
  - RoCE extends RDMA to Ethernet-based networks
- GPUdirect RDMA can copy from one GPU's HBM direct to another GPU's HBM without going through the CPU's DRAM at all.

RDMA concept: Queue Pair



- Send Queue – software enqueues work requests
- Receive Queue – software pre-posts receive buffers for incoming requests
- Completion Queue – NIC posts notifications of completed work queue entries or errors.

Types of Queue Pair

- **RC (Reliable Connection)**
  - Point-to-point, reliable, in-order delivery (like TCP over RDMA).
  - Supports **all verbs**: Send/Recv, RDMA Read/Write, Atomics.
- **UC (Unreliable Connection)**
  - Point-to-point, unreliable, unordered delivery.
  - Supports **Send/Recv, RDMA Write** (no Read/Atomics).
- **UD (Unreliable Datagram)**
  - One-to-many communication.
  - Supports **Send/Recv only**, no RDMA Read/Write.
  - Used for broadcast/multicast, sometimes for control messages.

Types of Queue Pair

- **RC (Reliable Connection)**
  - Point-to-point, reliable, in-order delivery (like TCP over RDMA).
  - Supports **all verbs**: Send/Recv, RDMA Read/Write, Atomics.
- **UC (Unreliable Connection)**
  - Point-to-point, unreliable, unordered delivery.
  - Supports **Send/Recv, RDMA Write** (no Read/Atomics).
- **UD (Unreliable Datagram)**
  - One-to-many communication.
  - Supports **Send/Recv only**, no RDMA Read/Write.
  - Used for broadcast/multicast, sometimes for control messages.

RDMA verbs API

- **RDMA Write** (one-sided) →
  Widely used for large tensor transfers.
  Example: GPU memory is registered, and peers perform RDMA writes directly into each other's memory.
- **Send/Recv** (two-sided) →
  Still used in some collective implementations, particularly for **control traffic** (small messages, synchronization).
- **RDMA Read** →
  Less common than write, but sometimes used in implementations where the receiver "pulls" data instead of the sender "pushing."
- **Atomics (CAS, FAA)** →
  Rare in AI training, but sometimes used in **barrier synchronization** or experimental parameter-server style approaches.

RDMA verbs API

- **RDMA Write** (one-sided) →
  Widely used for large tensor transfers.
  Example: GPU memory is registered, and peers perform RDMA writes directly into each other's memory.
- **Send/Recv** (two-sided) →
  Still used in some collective implementations, particularly for **control traffic** (small messages, synchronization).
- **RDMA Read** →
  Less common than write, but sometimes used in implementations where the receiver "pulls" data instead of the sender "pushing."
- **Atomics (CAS, FAA)** →
  Rare in AI training, but sometimes used in **barrier synchronization** or experimental parameter-server style approaches.

Write vs Write-with-immediate

- **RDMA Write**
  - **One-sided operation**: The initiator (local RNIC) writes data directly into the remote peer's registered memory.
  - **No involvement from remote CPU/application**: The remote side doesn't get an explicit signal.
  - **Completion**: Only the initiator sees a completion entry (in its own Completion Queue).
- **RDMA Write with Immediate**
  - Same as **RDMA Write, but adds a notification** to the remote side.
  - In addition to writing data into the remote memory, the initiator sends a **32-bit immediate value**.
  - The remote RNIC places this immediate value into the remote **Completion Queue (CQ)**.
  - **Effect**: The remote process is notified that "new data has arrived" without having to poll memory.

RoCE Write: Protocol

- NIC segments transfer into packets, typically 4KB.
- Sends open-loop (no TCP ack-clock).
- Receiver sends cumulative ACKs:
  - with outgoing data
  - on various event triggers
  - periodically (eg every 32 packets)
- Receiver sends a NACK if an out-of-order data packet is arrived
  - gives next expected packet
  - Sender performs go-back-n on receipt of NACK.

RoCE Performance

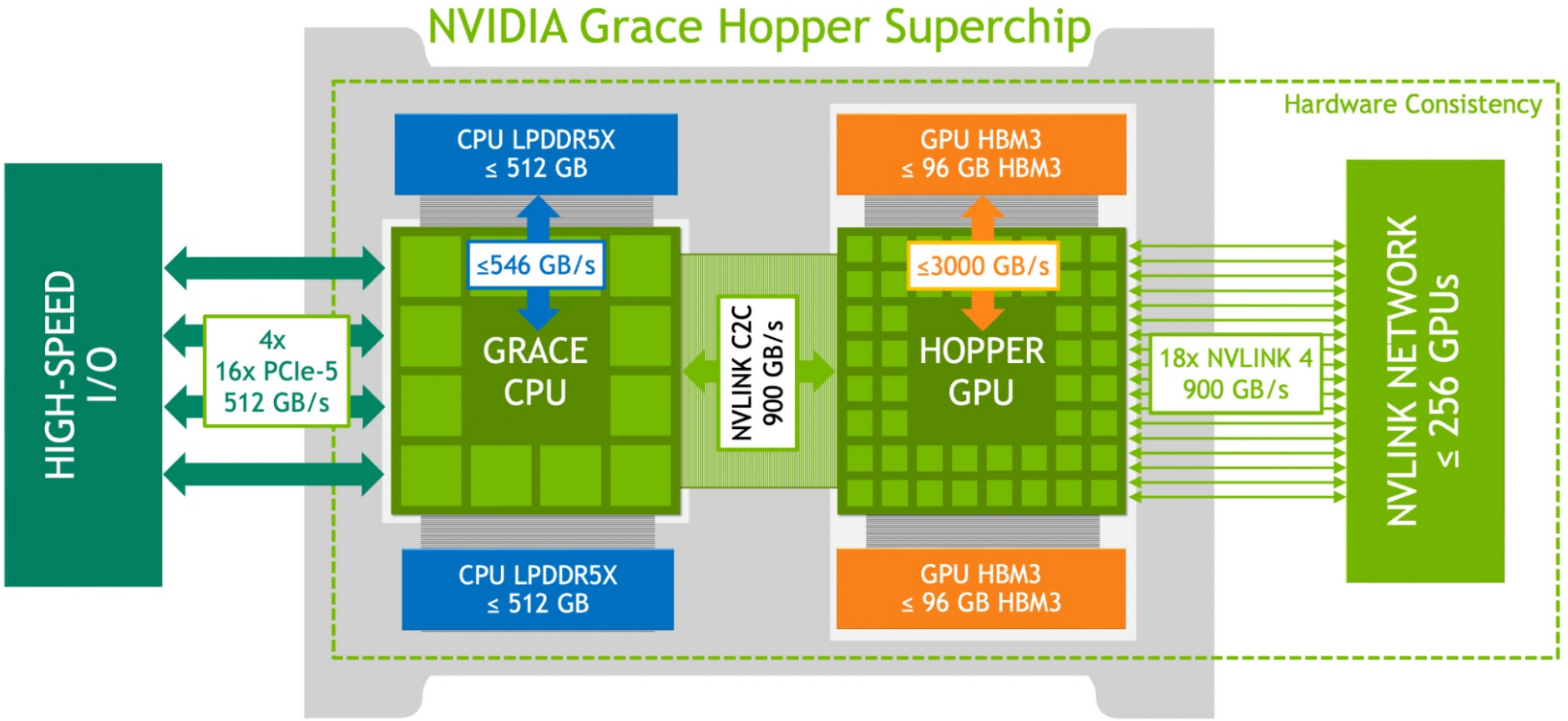
- Assumes negligible loss rate.
  - Can do go-back-n retransmission, but kills performance.
- Assumes negligible reordering
  - Reordering will trigger unnecessary go-back-n

- Solution:
  - lossless Ethernet, via PFC
  - upcoming: link-layer-retransmit

PFC and scaling

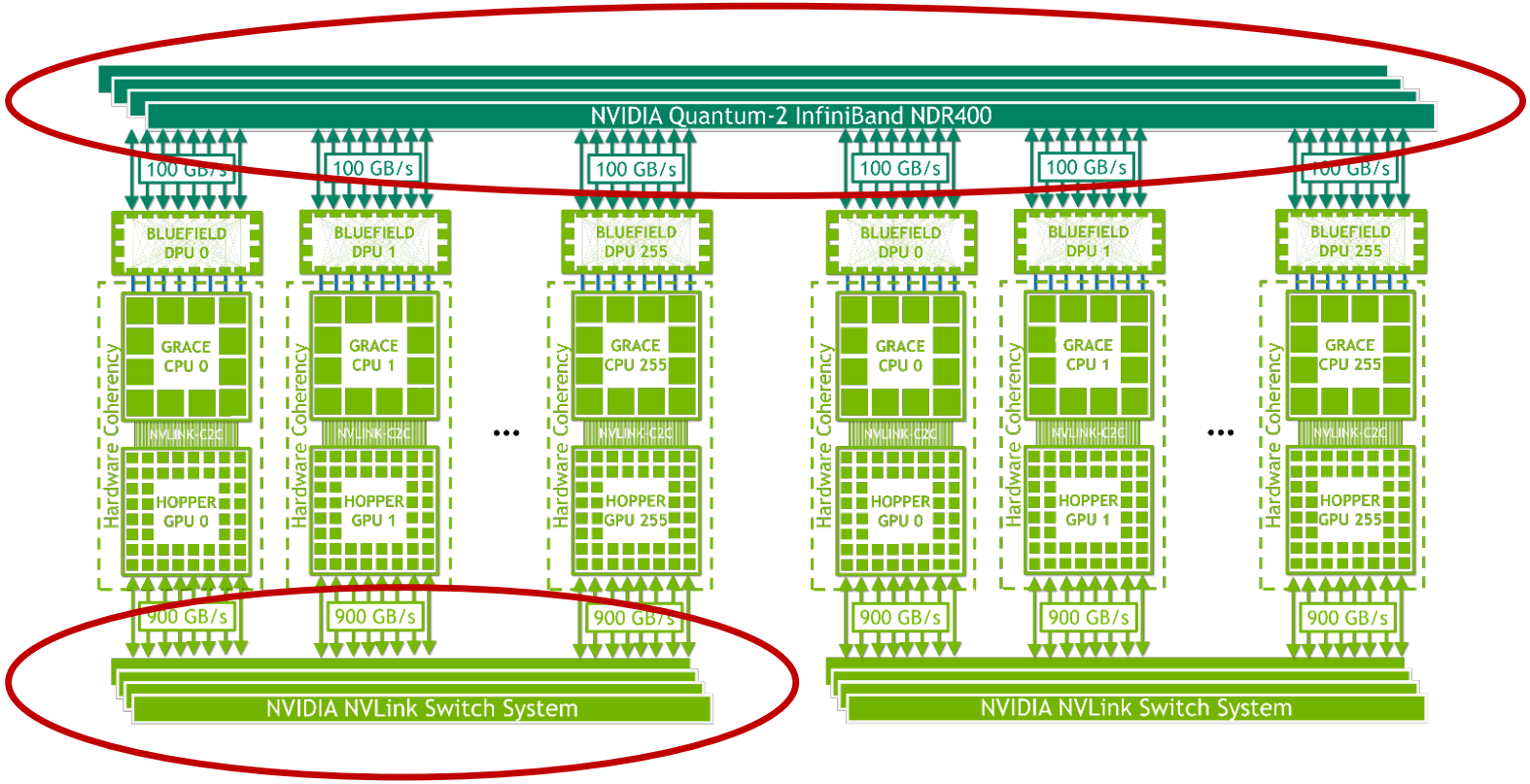
- As networks scale, especially with synchronized workloads, becomes increasingly important to avoid PFC spreading
  - PFC stalls other flows, increasing overall FCT
- Avoiding PFC requires good congestion control
  - Slow down sender before PFC has stalled the network
- RoCE commonly uses DCQCN
  - Switches set ECN on data packets on congestion
  - Receiver sends Congestion Notification Packet (CNP) on receipt of ECN
  - Multiplicative decrease on CNP, then hold time
  - Additive increase on no congestion

GPU Hardware (GH100)

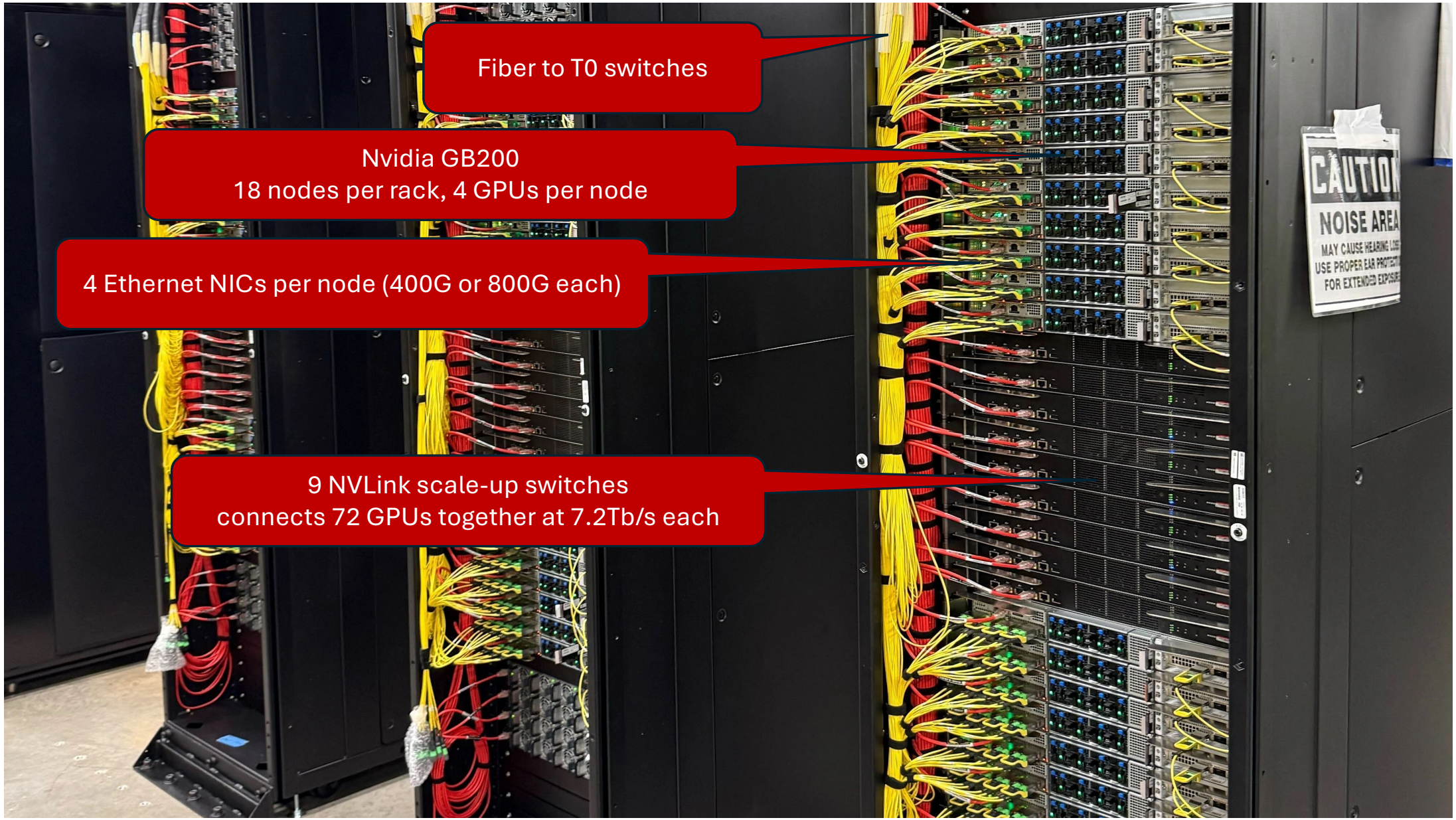


Scaling networking

Scale-out network. High bandwidth, extreme scale



Scale-up network. Extreme bandwidth, low latency, typically 8 GPUs w/ H100


Fiber to T0 switches
Nvidia GB200
18 nodes per rack, 4 GPUs per node
4 Ethernet NICs per node (400G or 800G each)
9 NVLink scale-up switches
connects 72 GPUs together at 7.2Tb/s each
CAUTION
NOISE AREA

Scale-up vs Scale-out vs Front-end

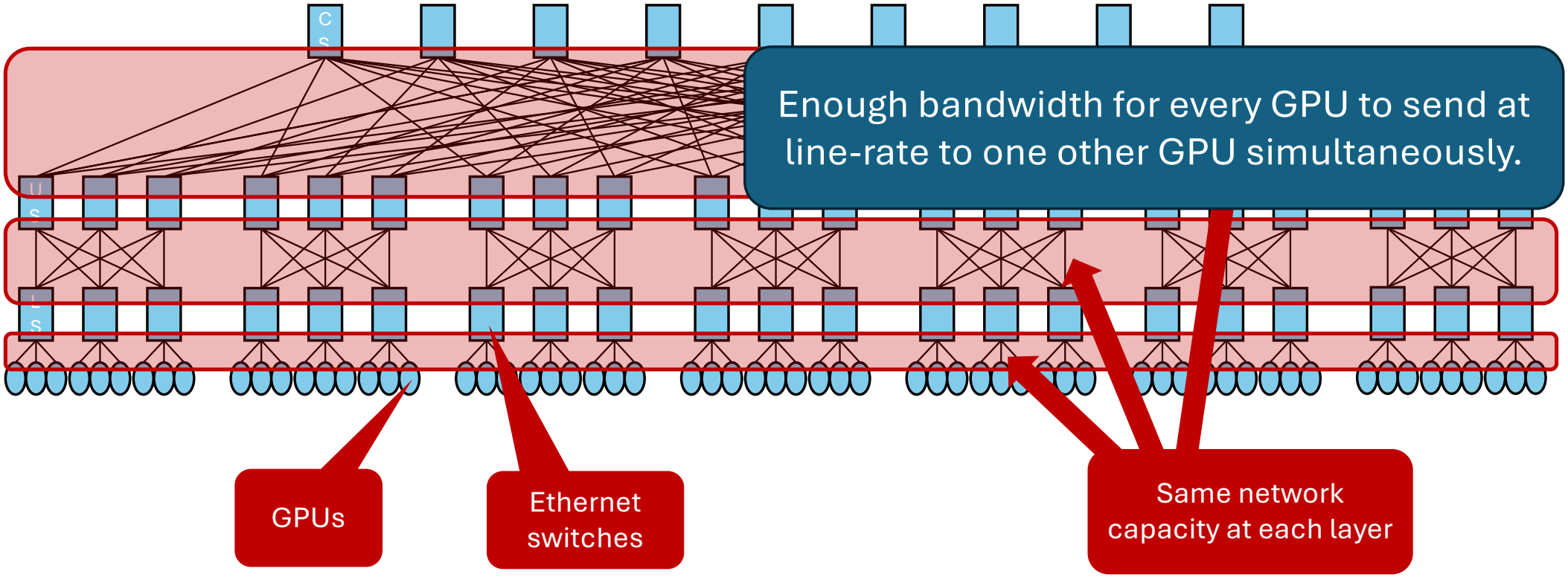
- Scale-up:
  - Very high bandwidth (8-10x faster than scale-out)
  - Typically copper
  - API: memory fabric
  - Within a rack 8-72 GPUs.
  - Direct connect, or single tier of switches
- Scale-out:
  - Fast: 400-800Gb/s per GPU
  - RDMA direct from GPU memory
  - 2-4 tiers of switches
  - Scale to 100K+ GPUs
- Front-end:
  - 200-400Gb/s per node (2 x CPUs)
  - Access to storage, etc
  - 2-4 tiers of switches
  - TCP, maybe RDMA from CPU

Scale-out: how can we connect 100K GPUs?

- Ideally, want to allow full line rate between any pair of GPUs
  - “full bisection bandwidth”
  - Means we don’t care which GPU group we assign for each scale-out task
    - But want to put tensor-parallel traffic on scale-up, so some constraints.

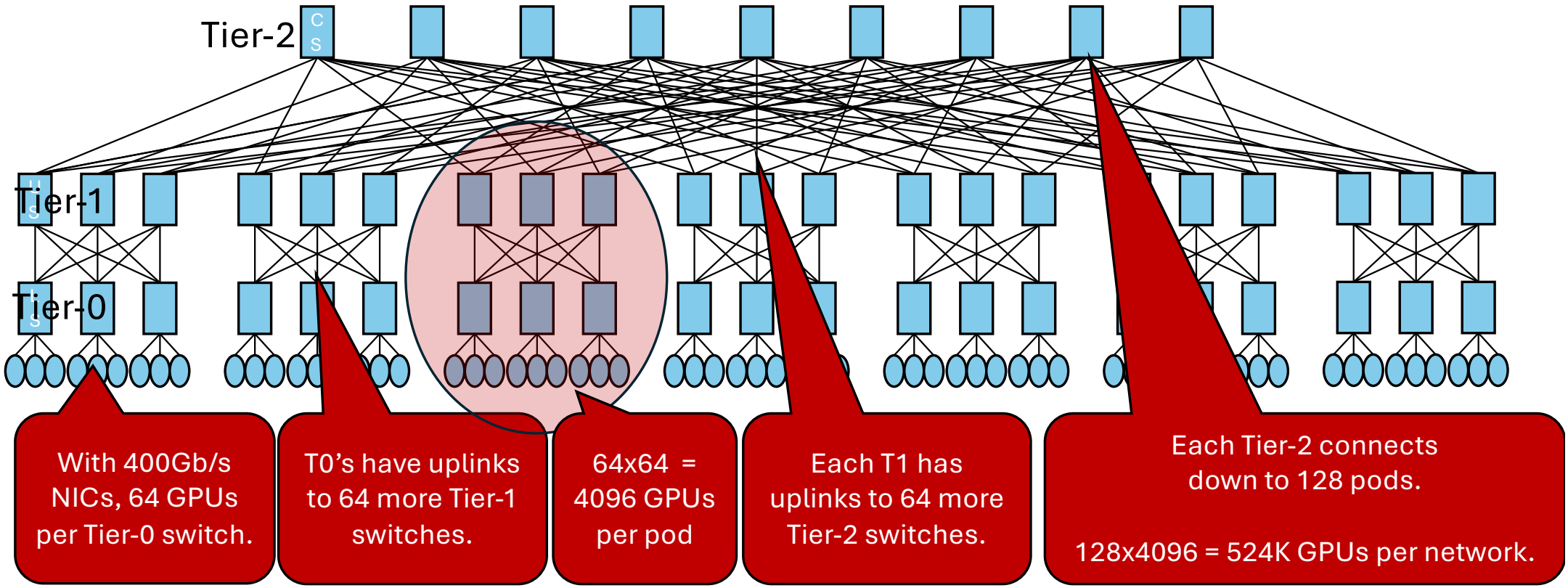
- With many GPUs, things will be failing constantly.
  - Meta’s Llama 3 (406B params), trained on 16K GPUs over 54 days.
  - 419 unplanned job interruptions
  - One every 3 hours on average.
  - 58% due to failed GPUs.
  - 8% due to failed network switches/cables

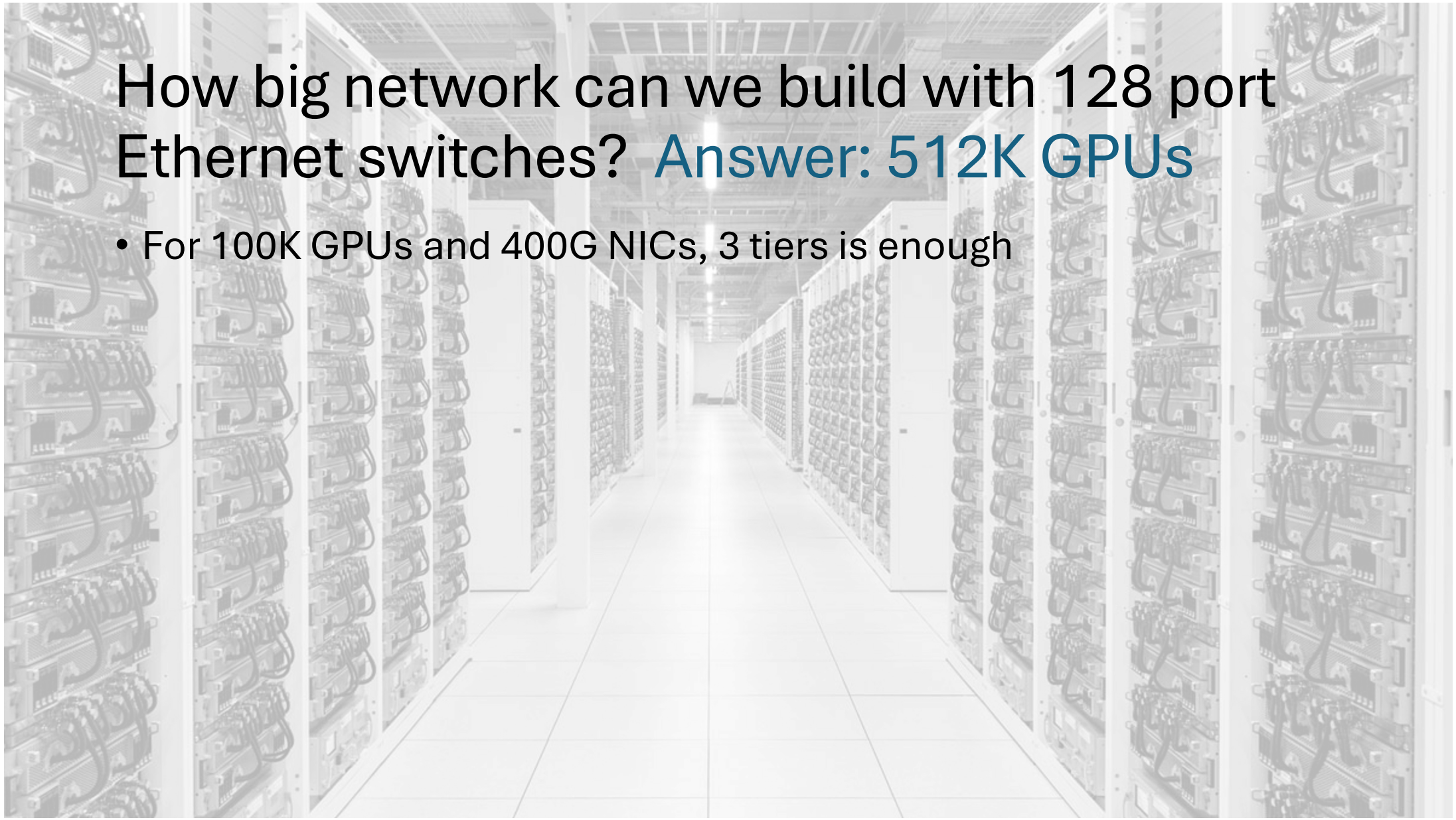
Clos topology (“FatTree”)




Broadcom Tomahawk 5:
51.2 Tb/s non-blocking Ethernet switch.
128 x 400Gb/s or 64 x 800Gb/s ports
TE
Amphenol
https://www.servethehome.com

How big a network can we build with 128 port Ethernet switches and 400Gb/s NICs?

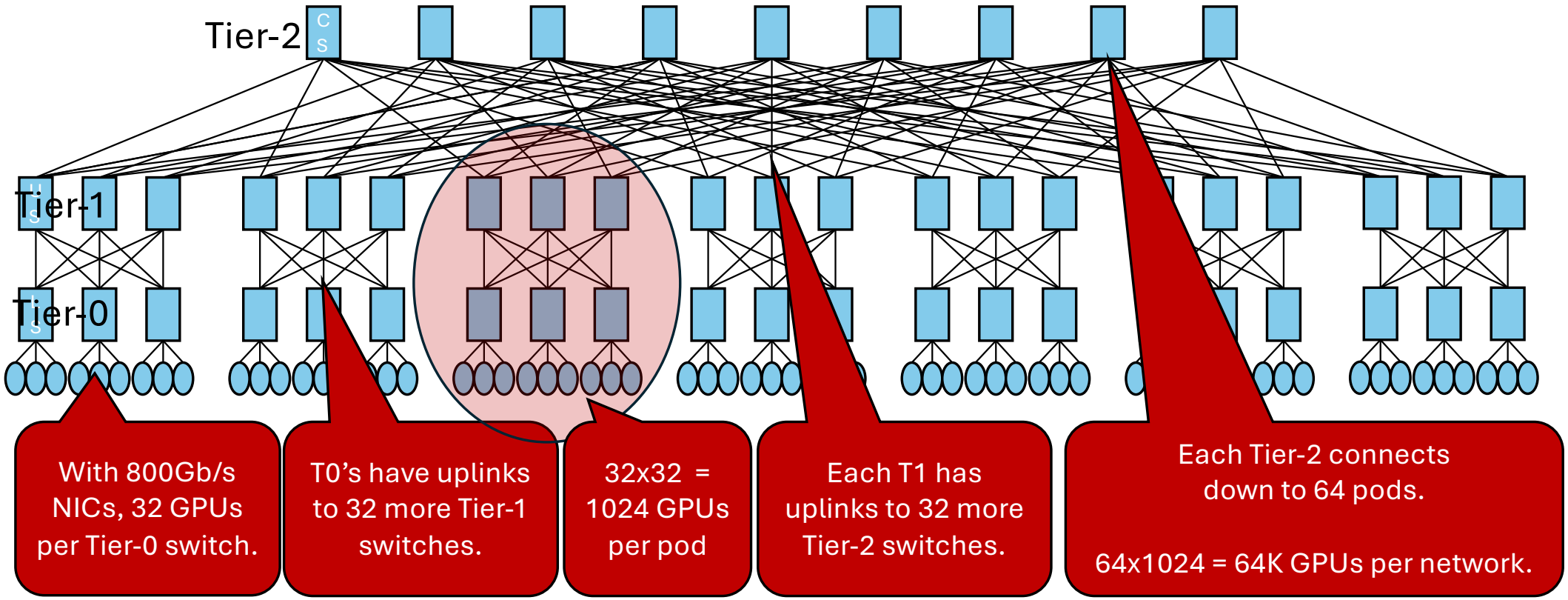


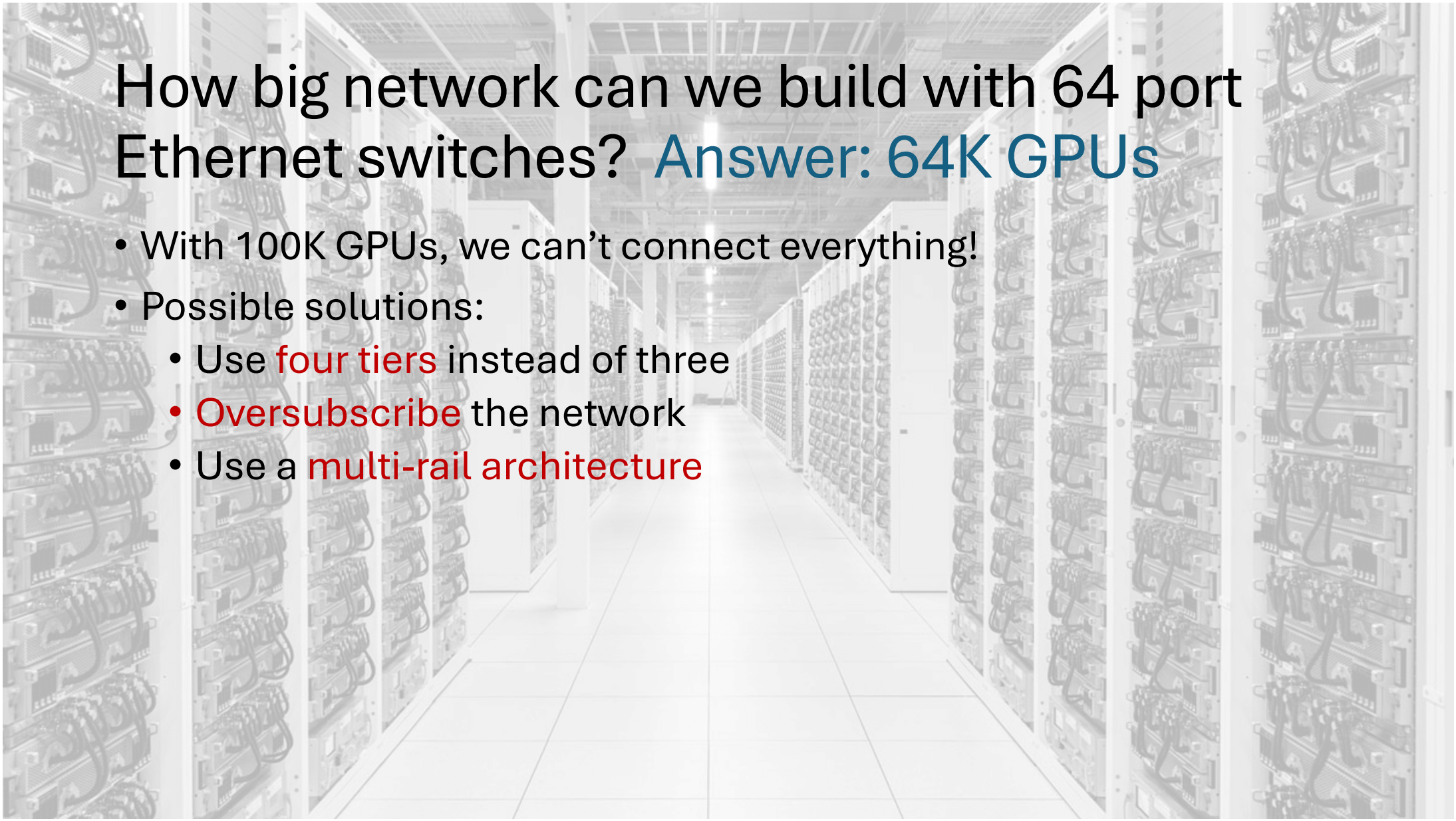


How big network can we build with 128 port Ethernet switches? Answer: 512K GPUs

- For 100K GPUs and 400G NICs, 3 tiers is enough

How big a network can we build with 64 port Ethernet switches? And 800Gb/s NICs

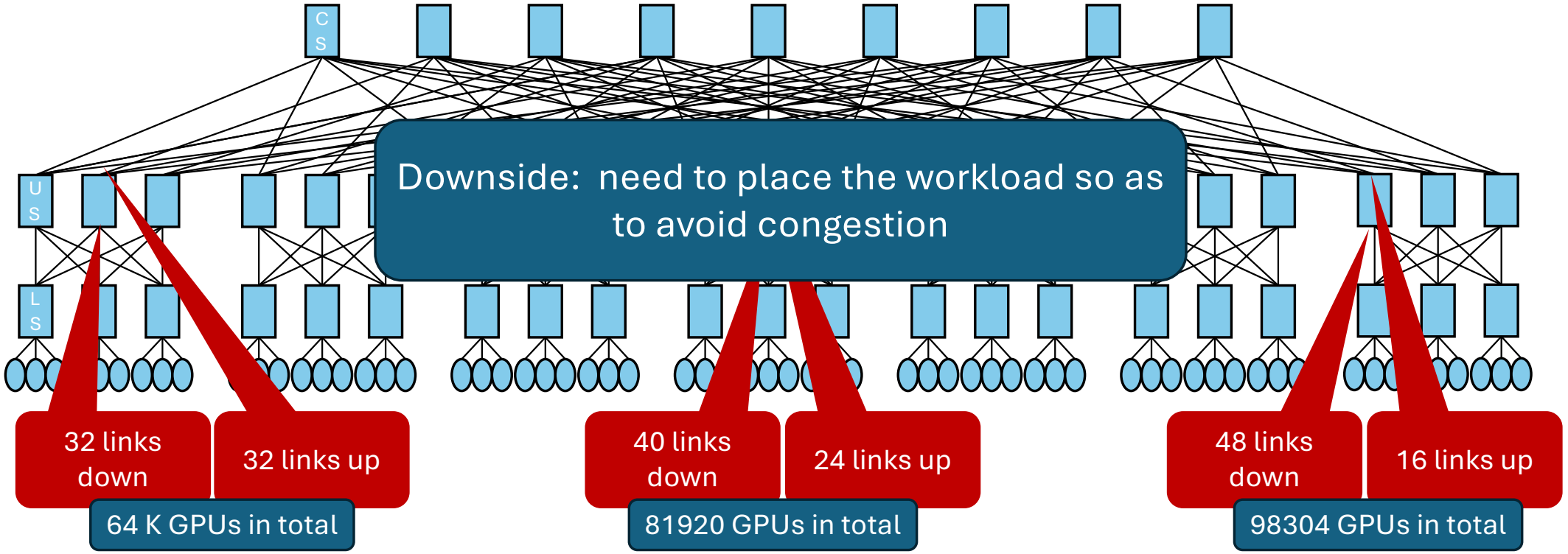




How big network can we build with 64 port Ethernet switches? Answer: 64K GPUs

- With 100K GPUs, we can't connect everything!
- Possible solutions:
  - Use four tiers instead of three
  - Oversubscribe the network
  - Use a multi-rail architecture

Oversubscribed FatTree

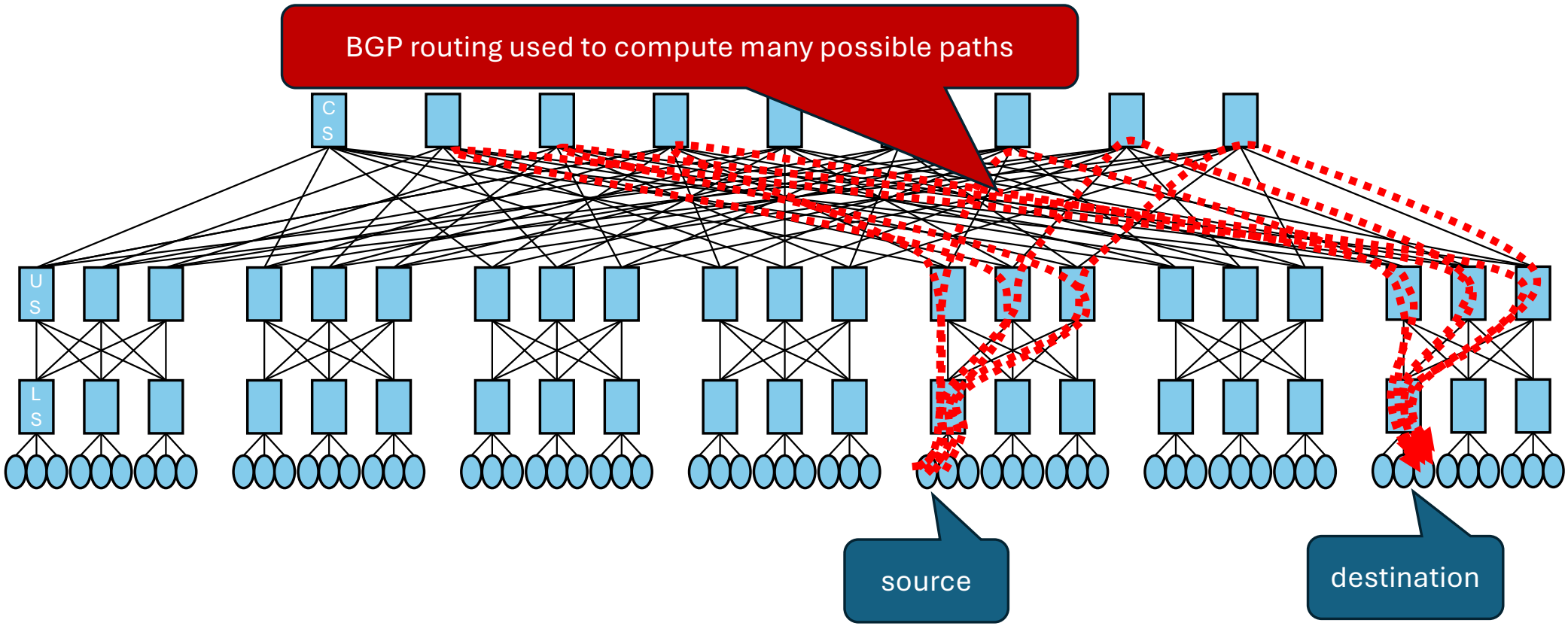


RoCE is stupid

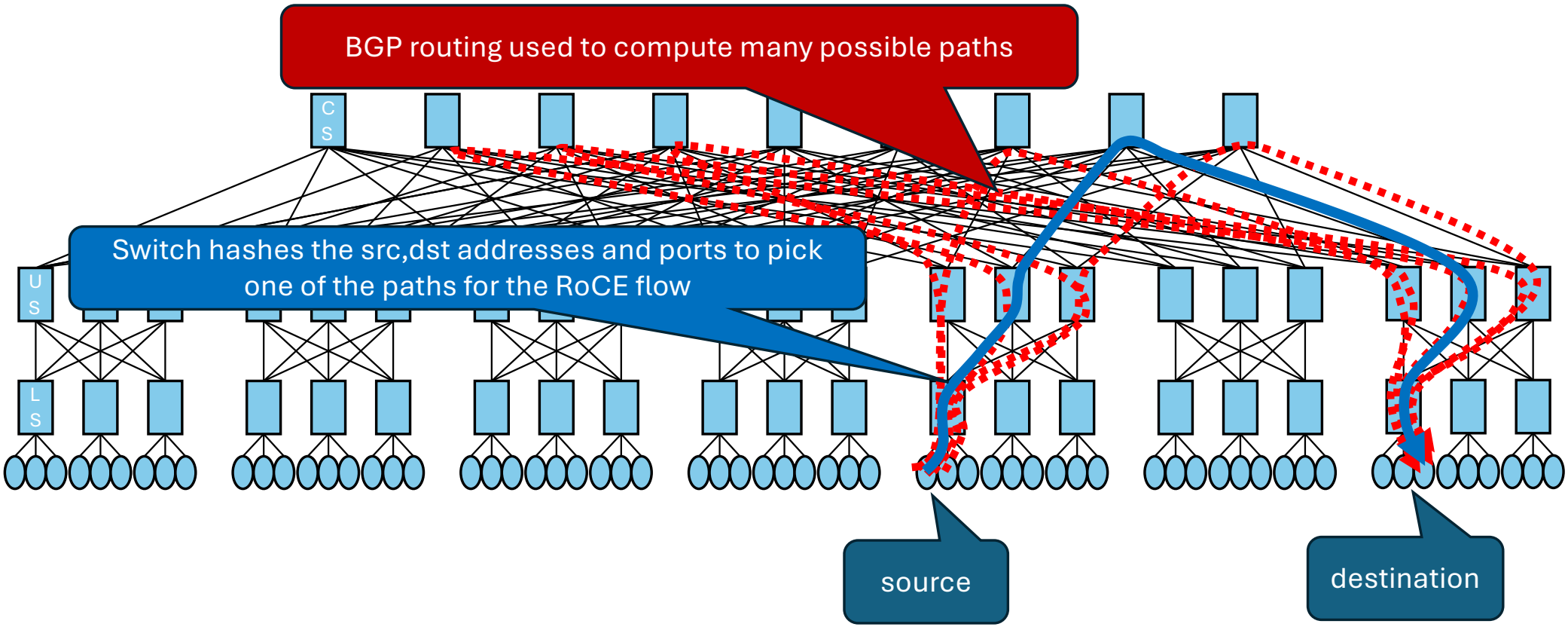
RDMA is ubiquitous for AI networking

- Implemented in NIC hardware.
- Assumes a lossless network
  - Ethernet can support pause frames/PFC to prevent congestion causing packet loss
  - This is hard to scale – need good congestion control to avoid deadlocks
- When loss does occur (eg corrupted packet), isn't smart enough to figure out what happened.
  - Unexpected packet causes a NACK to be sent.
  - Does go-back-n retransmission.
- If network reorders packets, RoCE thinks loss occurred and does go-back-n.

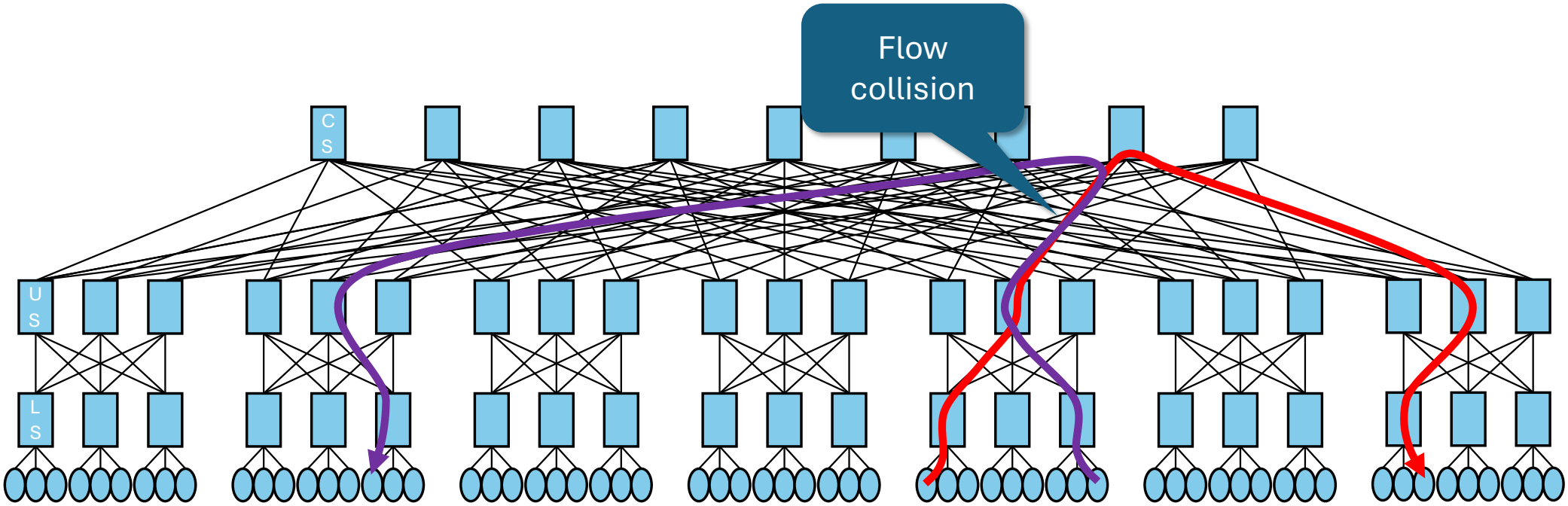
Equal Cost Multipath (ECMP) Routing



Equal Cost Multipath (ECMP) Routing

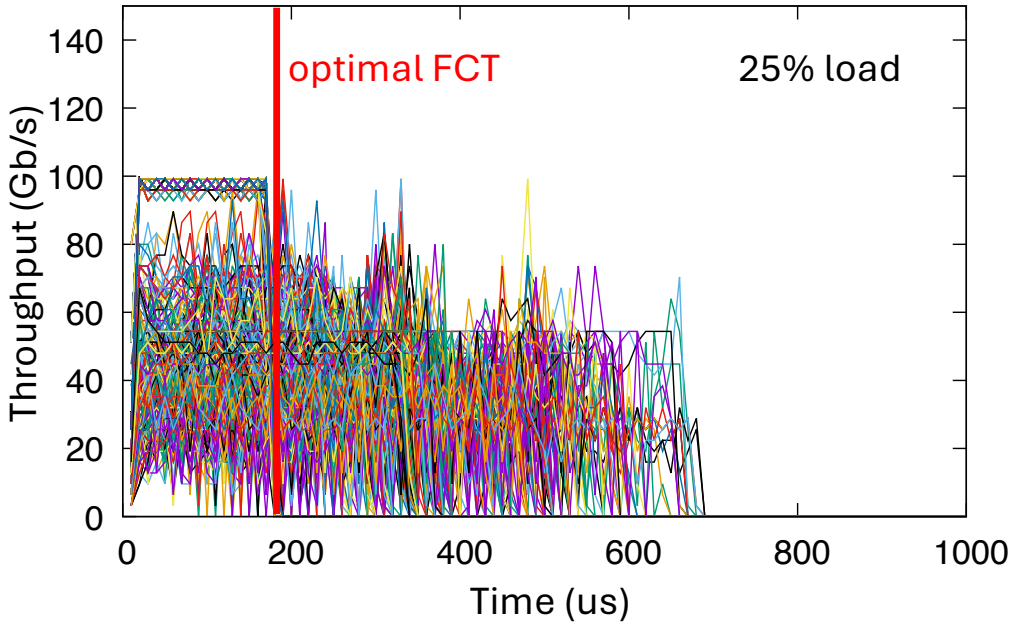


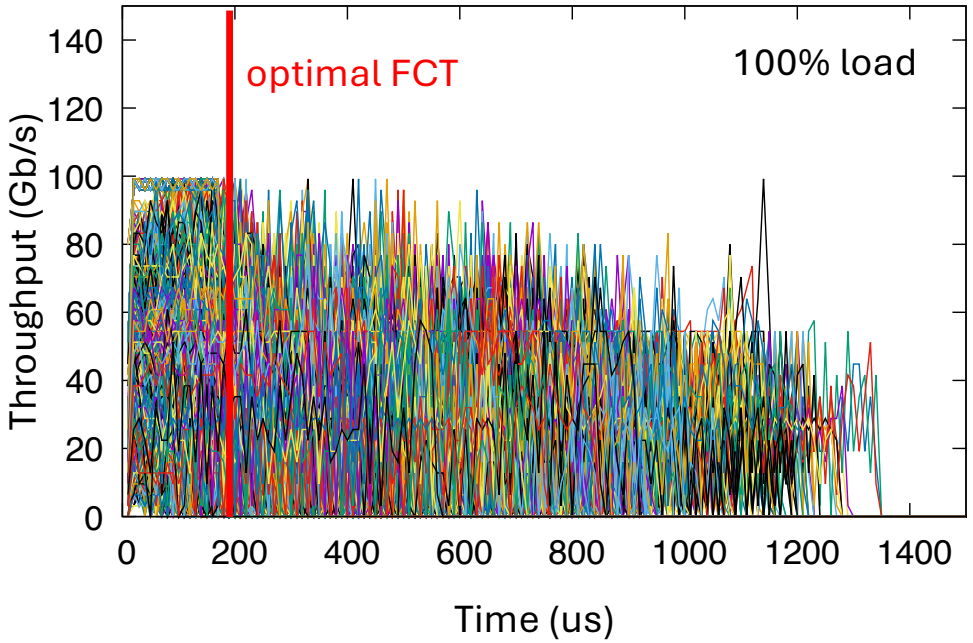
ECMP Flow collisions



RoCE under load

Permutation TM, 8192-node 100Gb/s Full bisection FatTree, 2MB transfers (htsim simulation)



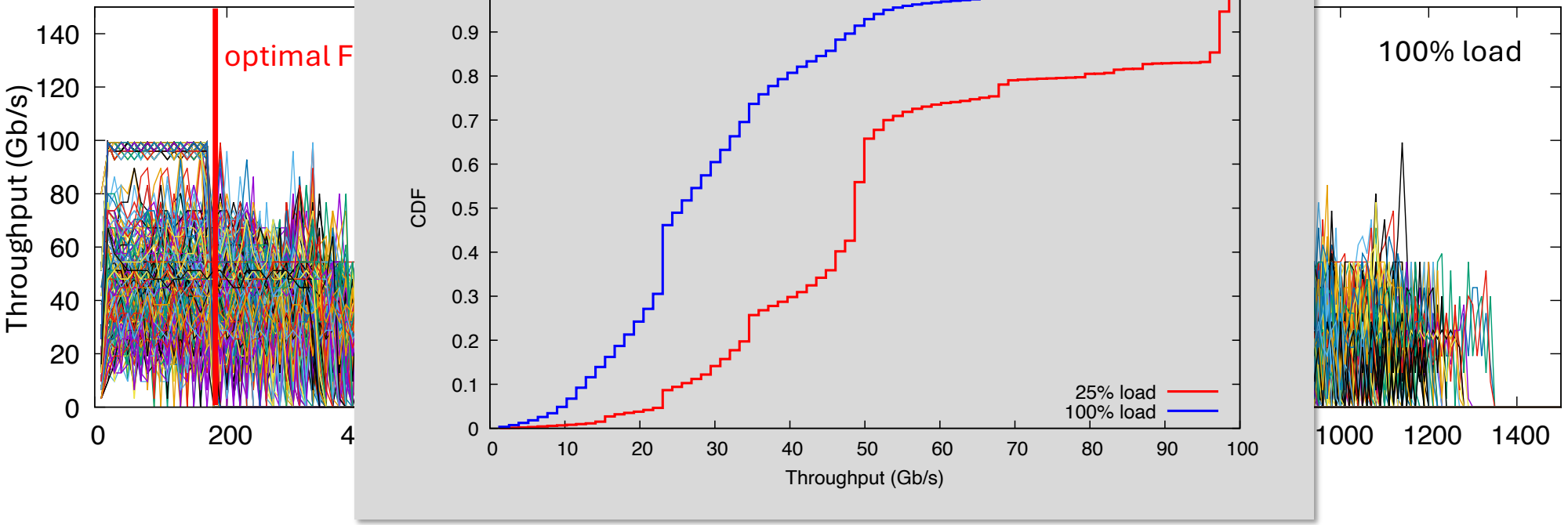


RoCE under load

Permutation TM, 8192-node 100G

RoCE doesn't tolerate reordering, so each transfer must use a single path through the network.

Many flows suffer flow collisions.



Can the switch help?

- The switch can see which ports are congested.
  - On T0 uplink, T1 uplink, there are multiple ports it could pick.
  - Pick the least congested for each new flow
    - Remember the choice for that flow to avoid reordering.

- Problems:
  - No path choice on T1, T2 downlinks.
  - With collectives, many flows start simultaneously
    - No congestion when choice is made.
    - May be able to reroute in idle time between bursts

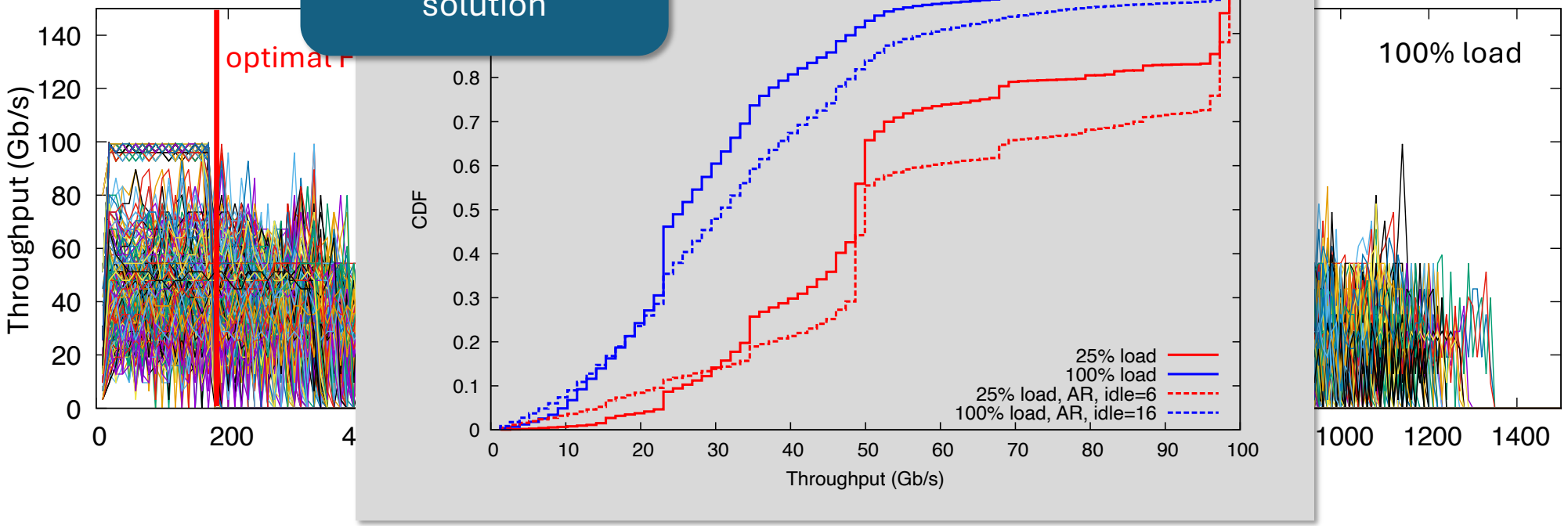
RoCE under load

Permutation TM, 8192-node 100G

RoCE doesn't tolerate reordering, so each transfer must use a single path through the network.

Many flows suffer flow collisions.

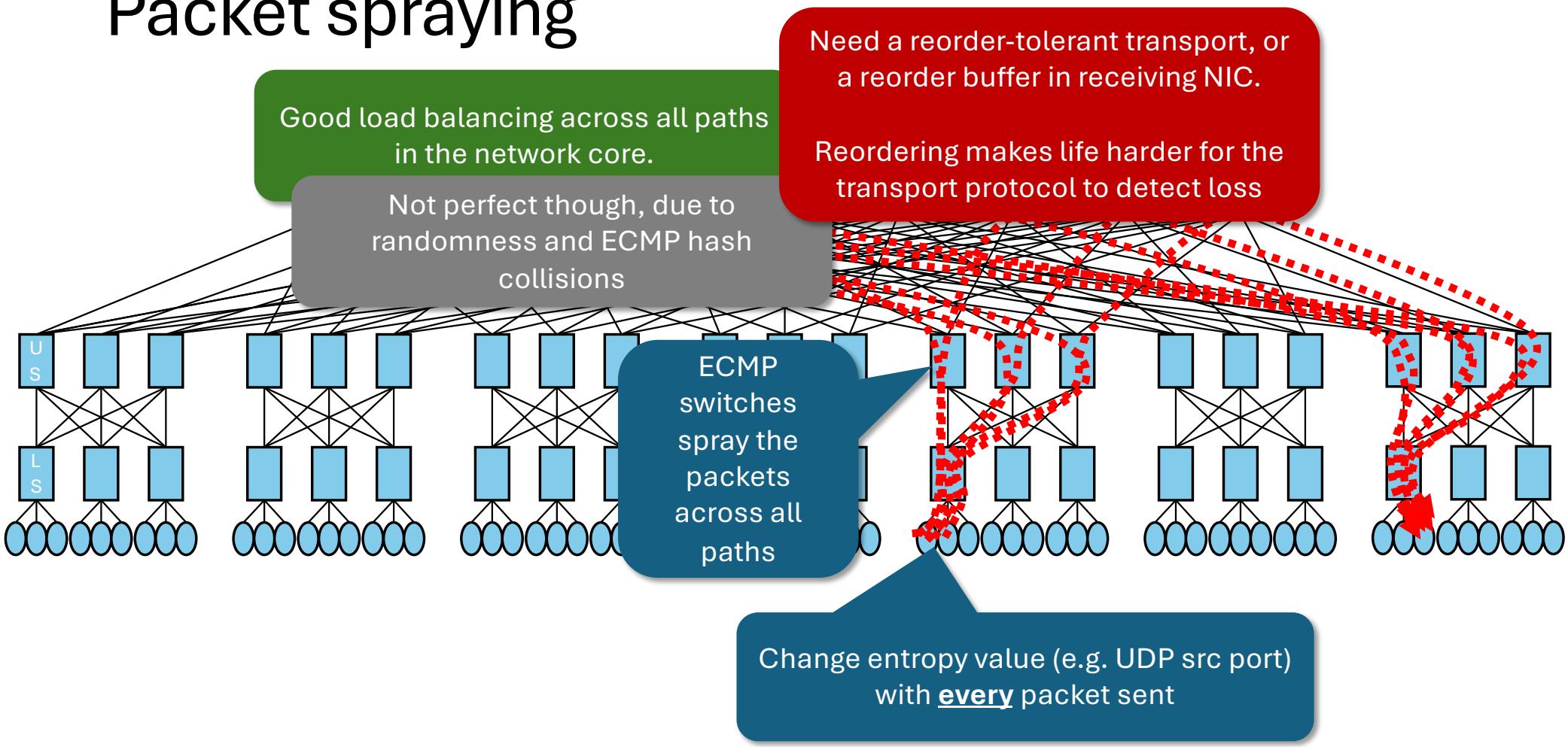
Adaptive routing helps, but not a full solution



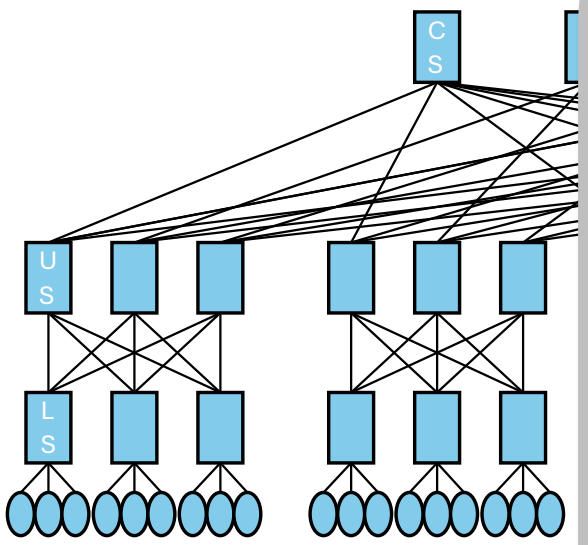
QP Scaling

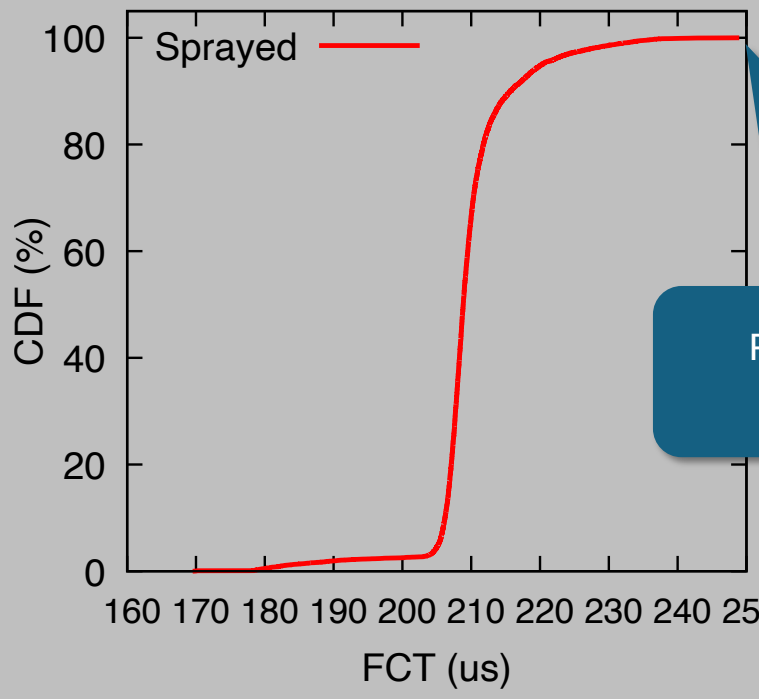
- In the collective library, stripe each transfer across many QPs.
  - Each QP no longer needs to go at line rate
  - Flow collisions still happen
    - Some QPs go slow (collide more), some go faster (collide less)
    - For long transfers, app can send more on faster QPs.
- Helps quite a bit, but still far from optimal.
- More work for collective software.

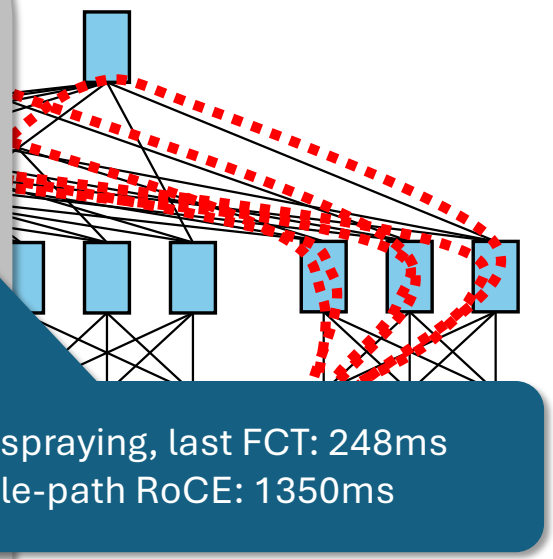
Packet spraying



Packet spraying



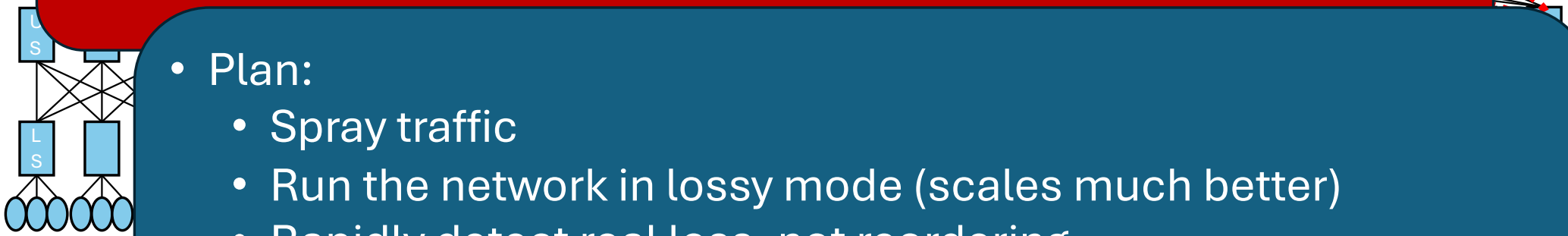




Packet spraying, last FCT: 248ms
Single-path RoCE: 1350ms

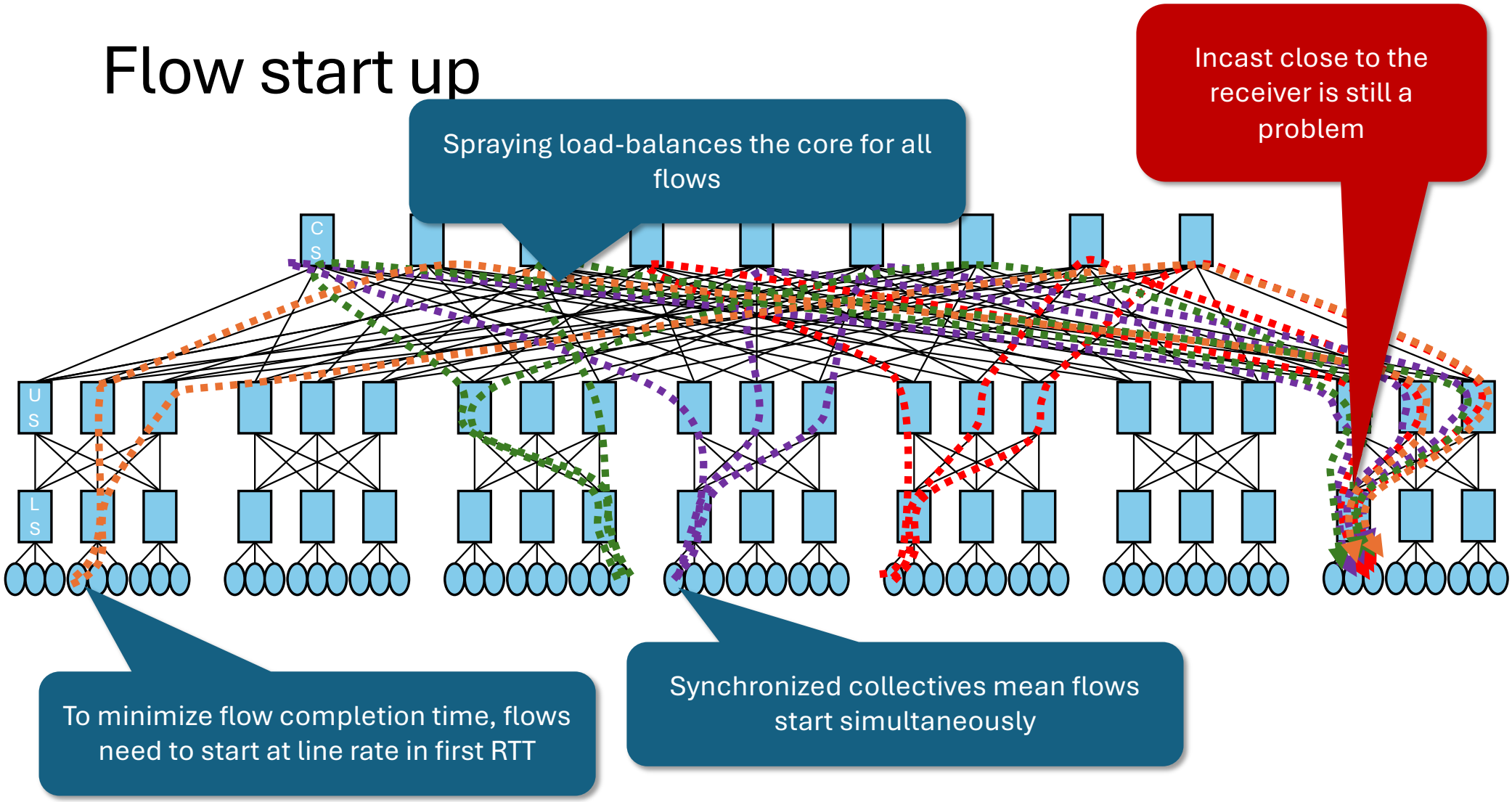
Downside: packet spraying causes reordering

- Different paths have slightly different load, so queue sizes vary.
- Packets will arrive in a slightly random order.
- How can the receiver quickly know which ones are missing?

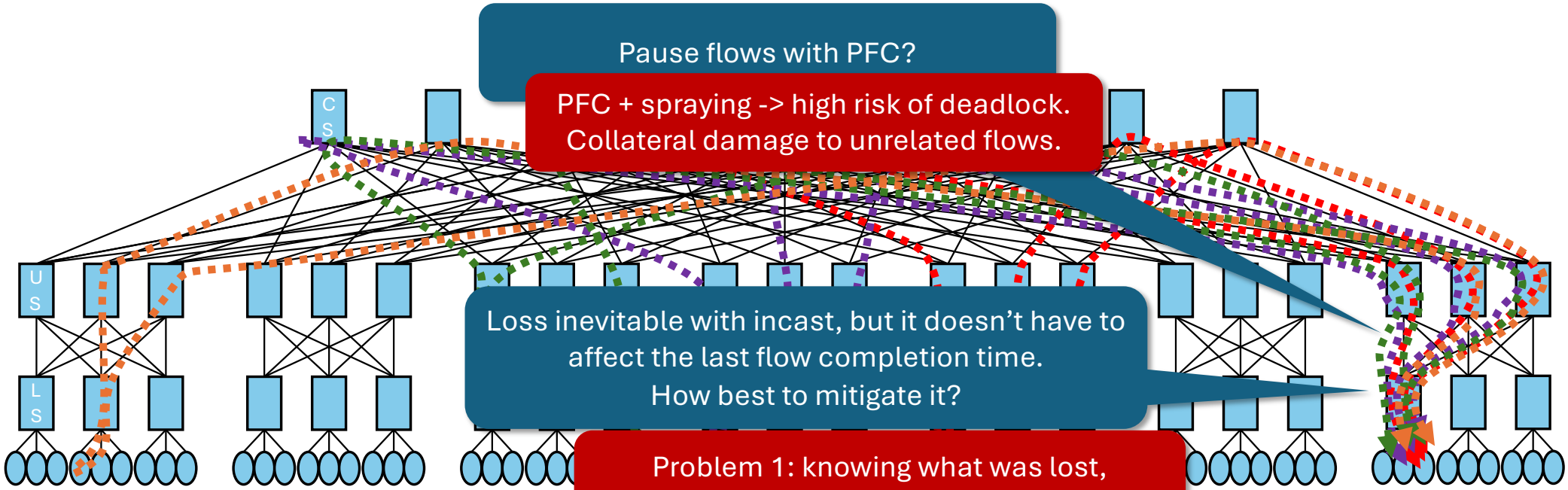


- Plan:
  - Spray traffic
  - Run the network in lossy mode (scales much better)
  - Rapidly detect real loss, not reordering
  - NACK only the missing packets
  - Fast retransmit in hardware

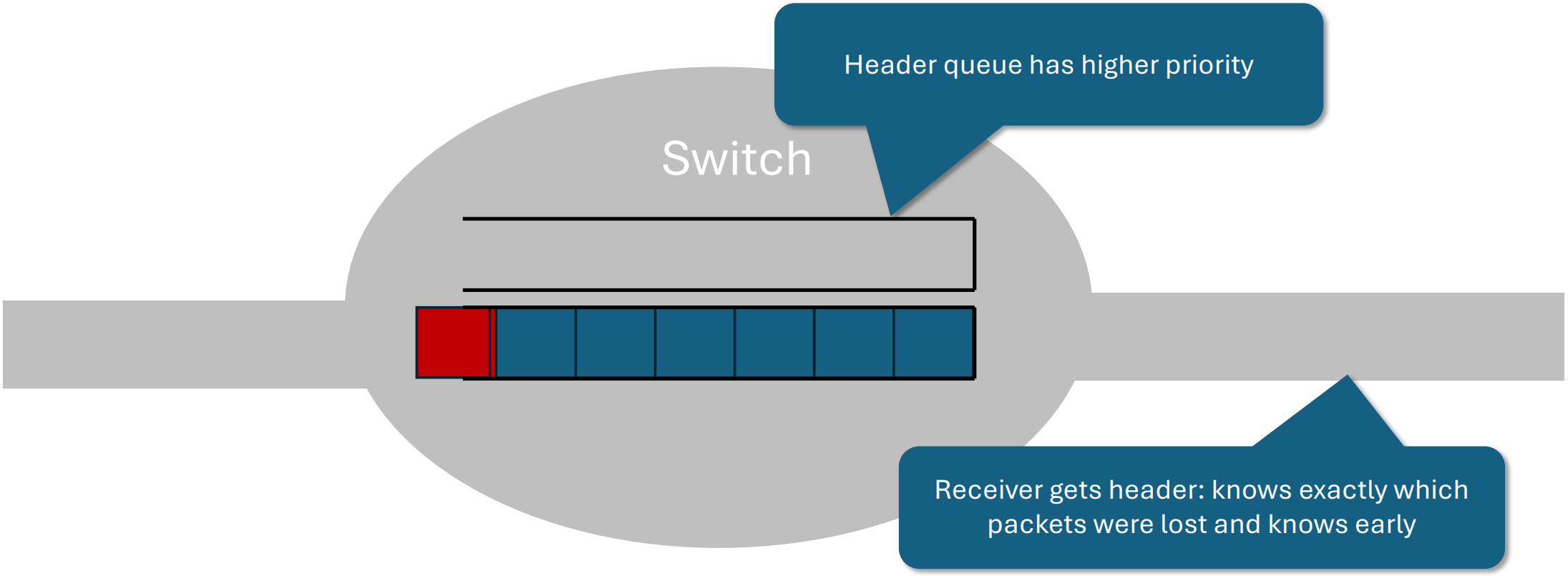
Flow start up



Flow start up

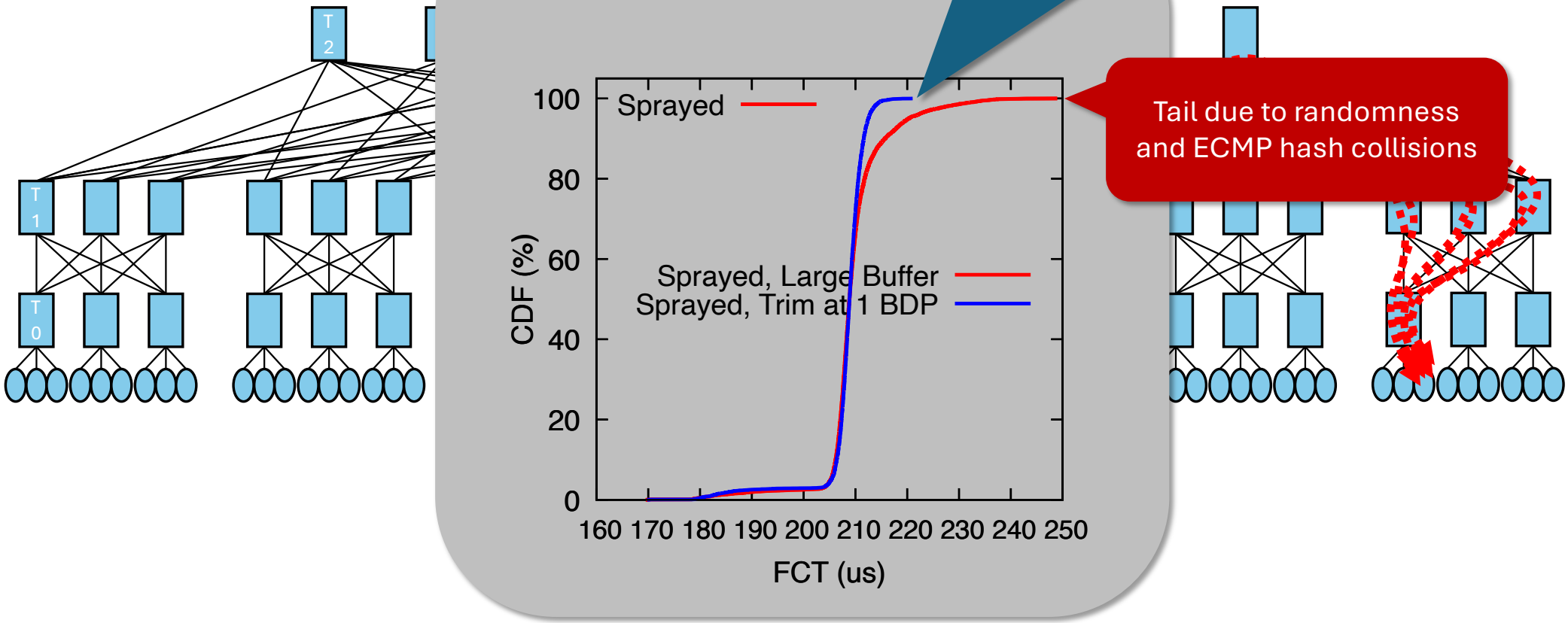


Packet trimming



Packet spraying

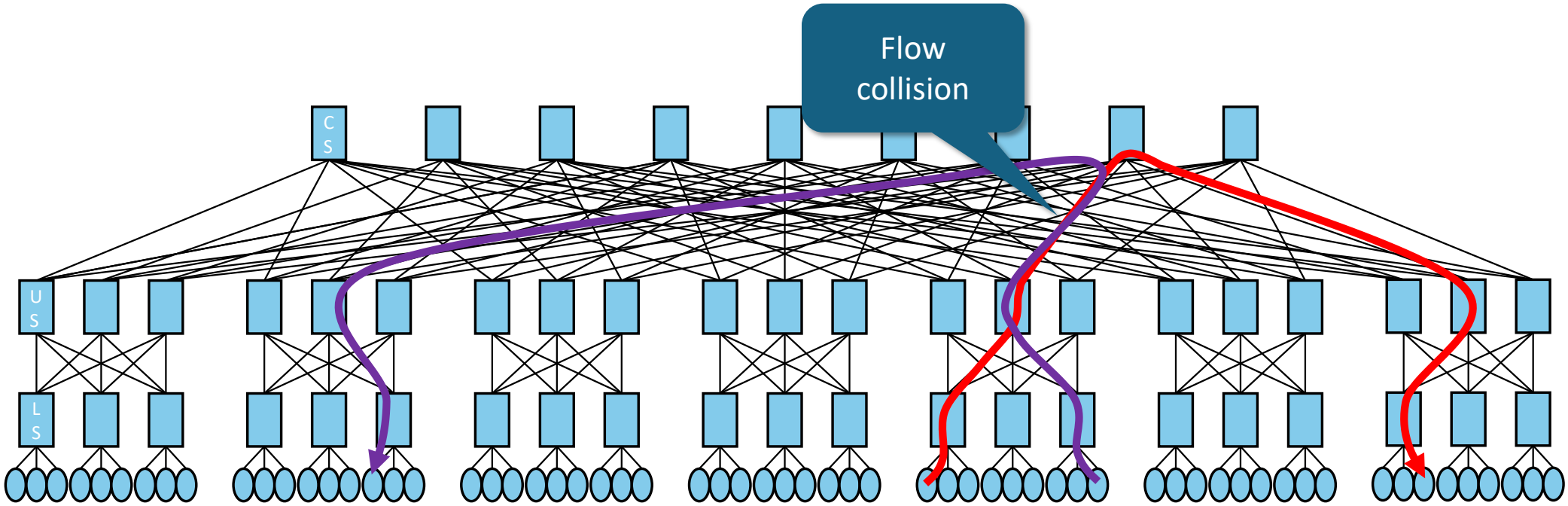
When sprayed load balancing is imperfect,
queues can still build.
Trimming prevents queue building.
Packet gets trimmed, NACKed,
RTX on a different less loaded path.



Tail due to randomness
and ECMP hash collisions

Load balancing in AI networks

Flow collisions



Background: avoiding flow collisions

- A transport connection must use multiple paths.
  - Equal split is optimal for symmetric networks.
  - Asymmetries may appear due to faults or single path traffic.

Two categories of solutions

- Multipath transport:
  - Congestion window and sequence numbers per path (subflow).
  - Subflow window adapted using per path congestion information.
  - Load balancing driven by each subflow CWND.

- Packet spraying:
  - Single congestion window across all paths.
  - Aggregate congestion information drives congestion window evolution.
  - Load balancing balances load across all paths.

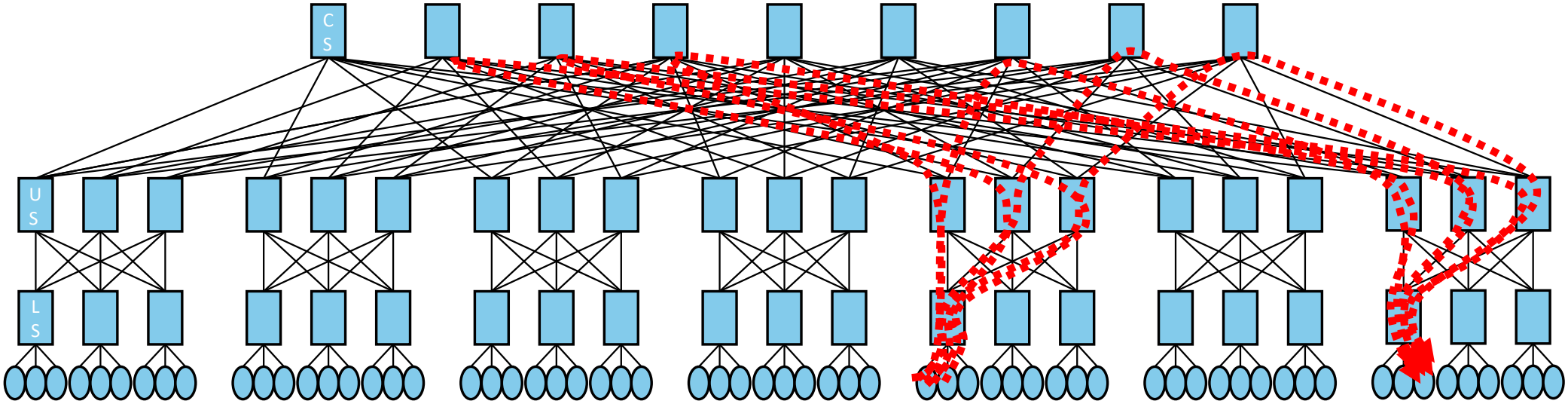
Multipath TCP
RFC 8684, 6356

- Open multiple subflows between the source and destination.
  - ECMP hashes each subflow to a (probabilistically) different path.
  - MPTCP connection setup: 2RTTs.
- Multipath congestion control
  - One window per path ($cwnd_j$), each with its own ACK clock, sequence space.
    - Subflow acts like a TCP connection from the network point of view.
    - But its cwnd is not independent.
  - Per path window depends on that window at all other windows.
    - On each ACK for subflow j, $cwnd_j$ += *a* / total_cwnd
    - On each loss for subflow j, $cwnd_j = cwnd_j / 2$

MPTCP for AIML networks?

- Great at load balancing for long flows.
- Need to use many paths for the common case of
  - Symmetric highly loaded networks + short flows.
  - Best way to load balance short flows is to use many paths
  - Load balancing works well for long flows, not so well for shorter flows.
- But with many paths, minimum MPTCP total window is #paths.
  - E.g. 256 paths means min 256 packet window. This equals BDP at 800Gbps.
  - Congestion collapse in incast.
- Path state for MPTCP is quite costly.
  - CWND, flight_size, sequence numbers, etc. (tens of bytes).
  - 256 * 20 = 5KB per connection!

Packet spraying in AIML networks

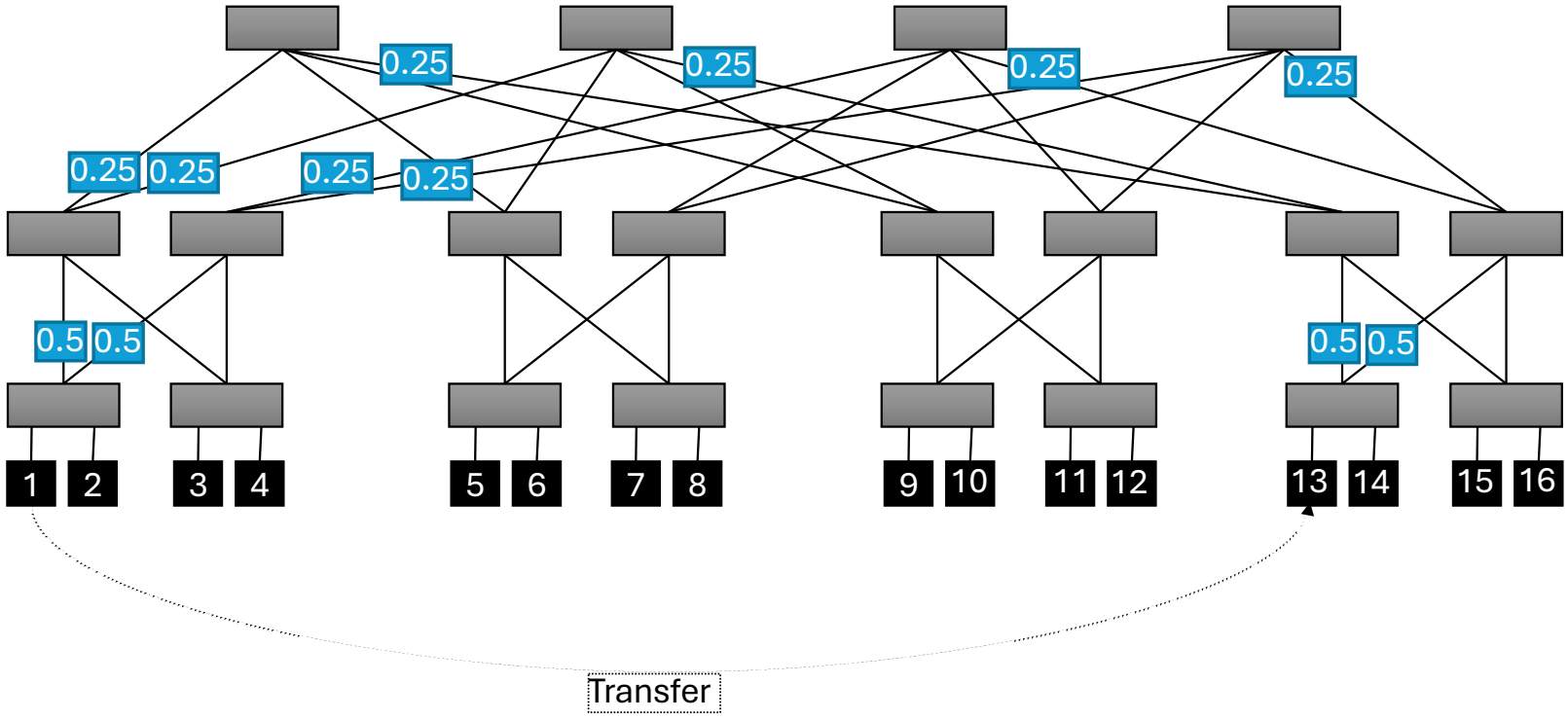


Congestion control with packet spraying

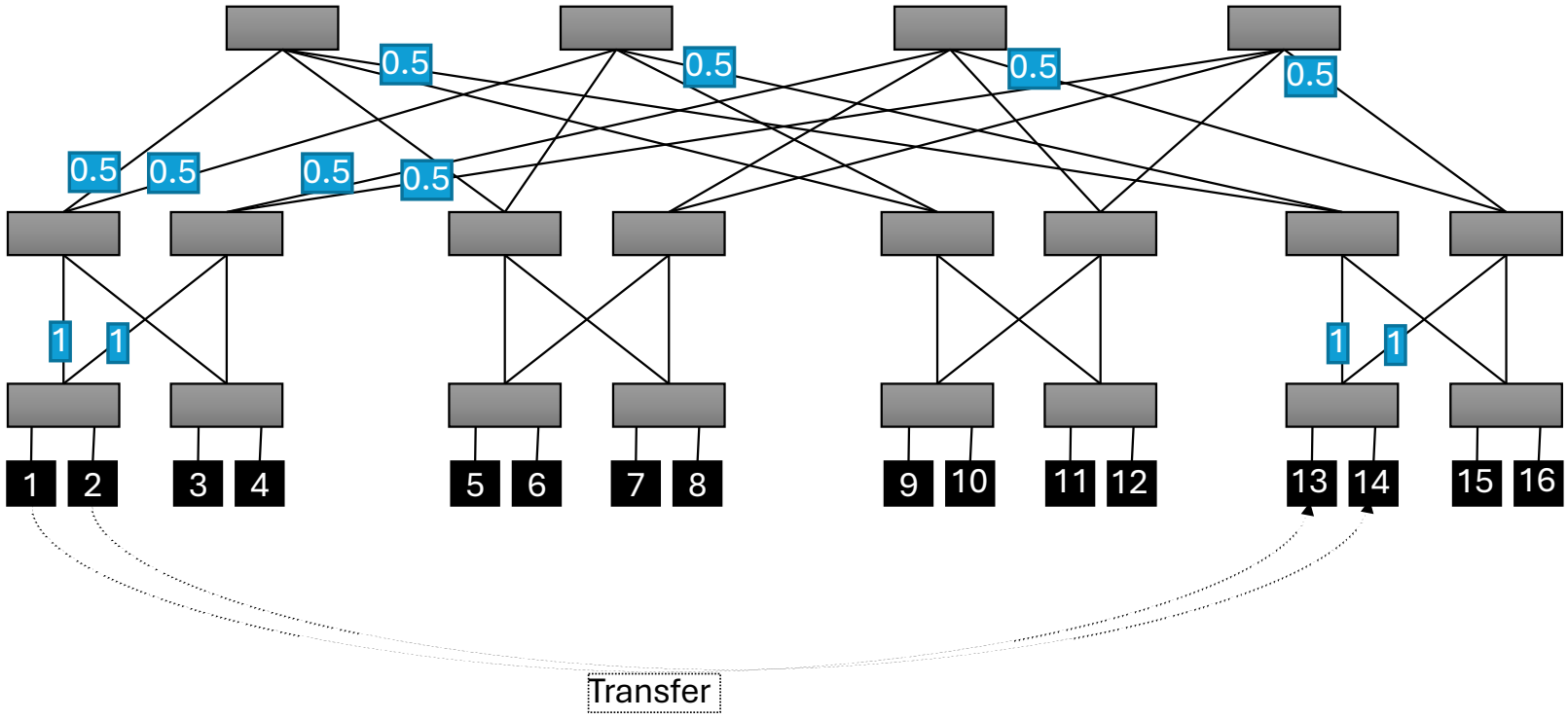
- Maintain a single congestion window that upper bounds flight size.
- Sender-driven congestion control (e.g. UET NSCC)
  - Targets sub-BDP standing queue at the bottleneck.
  - Use ECN and delay simultaneously.
    - Aggressive increase when queue ~ 0. Linear increase otherwise.
    - Multiplicative decrease when ECN mark & average delay above threshold.
  - Average queueing delay across all paths used as control target.

- But how to load balance packets across paths?
  - Bad load balancing results in reducing CWND (across all paths).
  - Good load balancing with asymmetric capacity is hard!

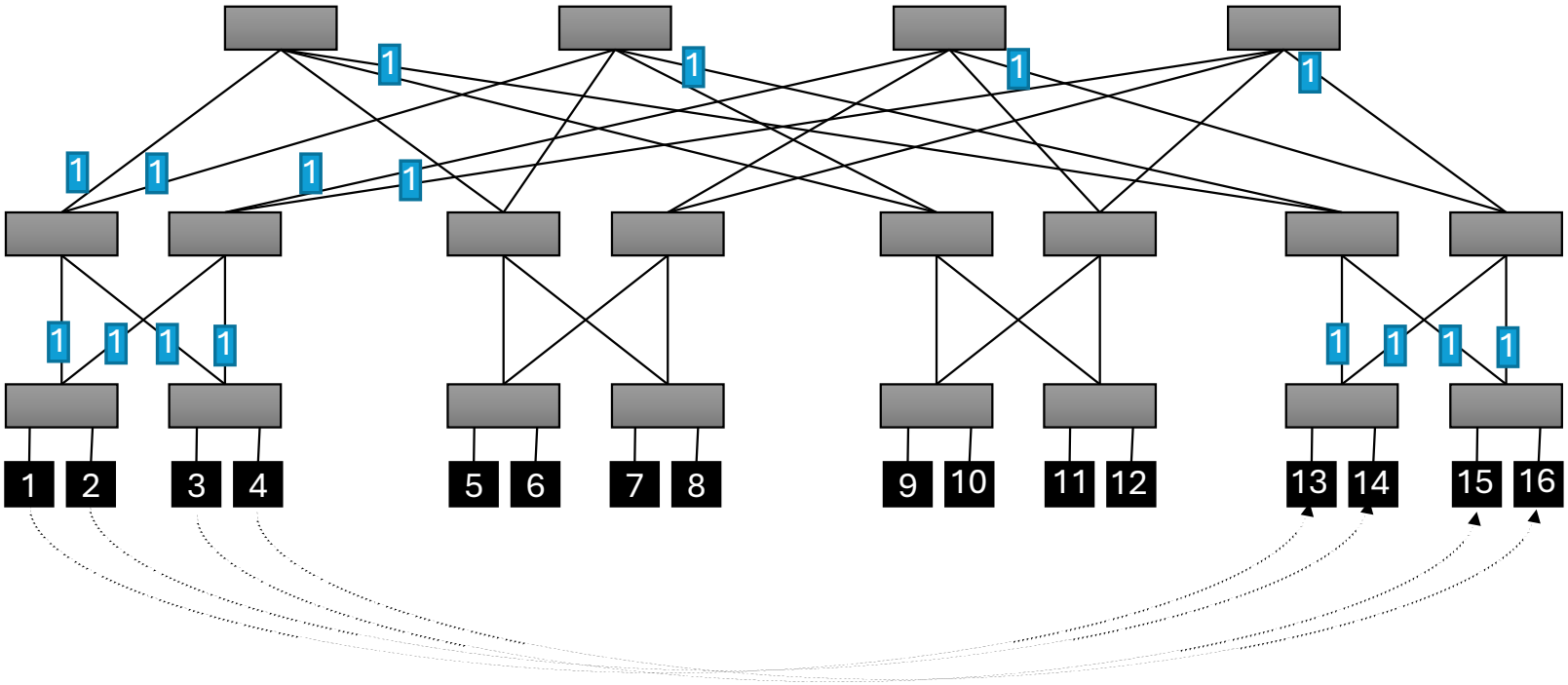
Spraying aims to equalize load on all paths



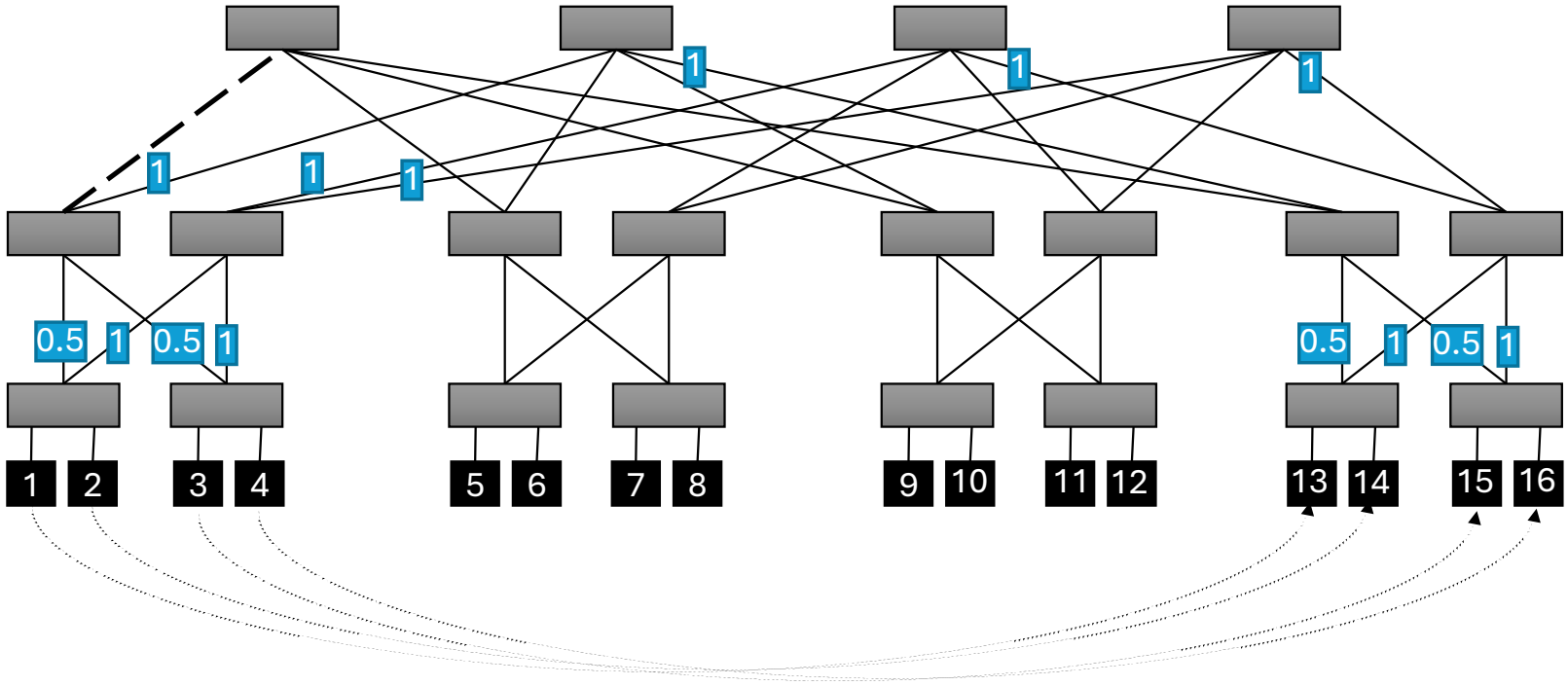
Spraying aims to equalize load on all paths



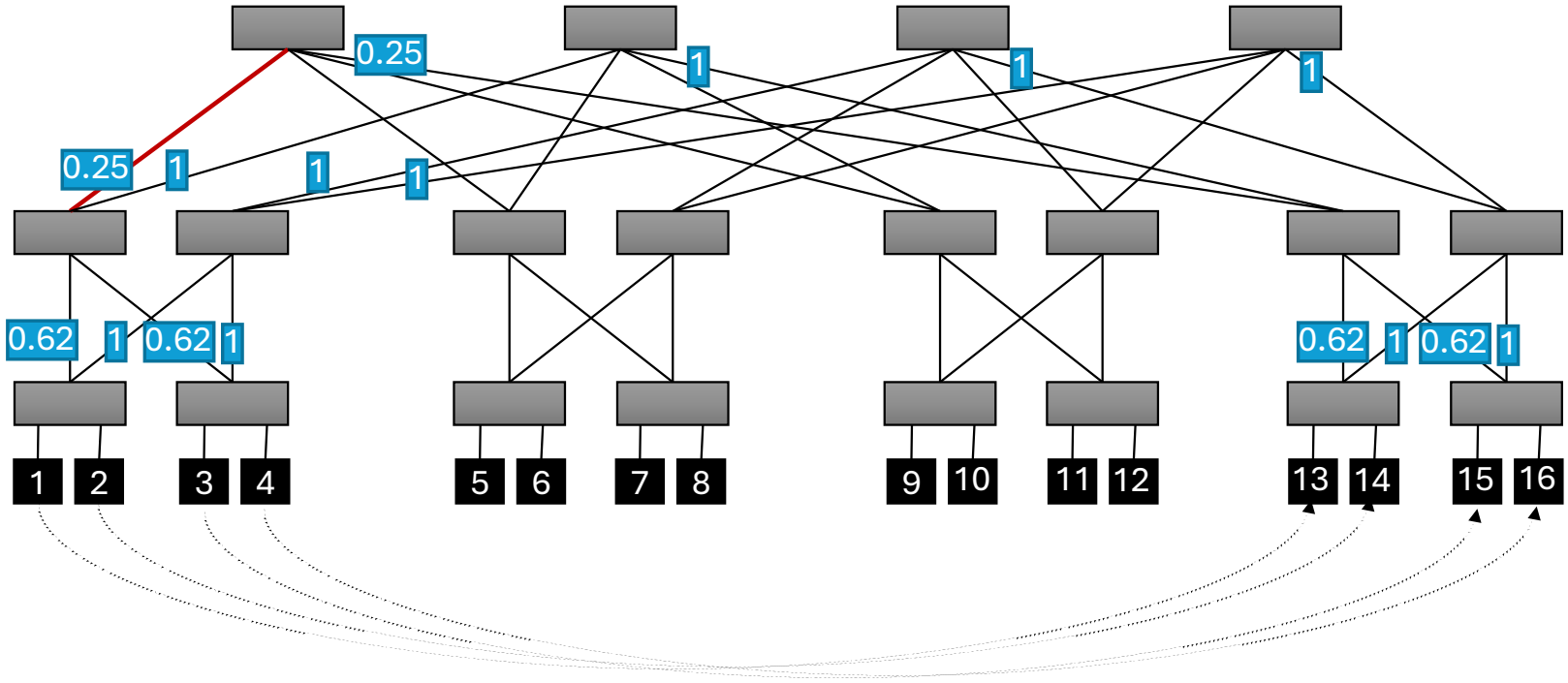
Spraying aims to equalize load on all paths



Spraying with asymmetries: failed links



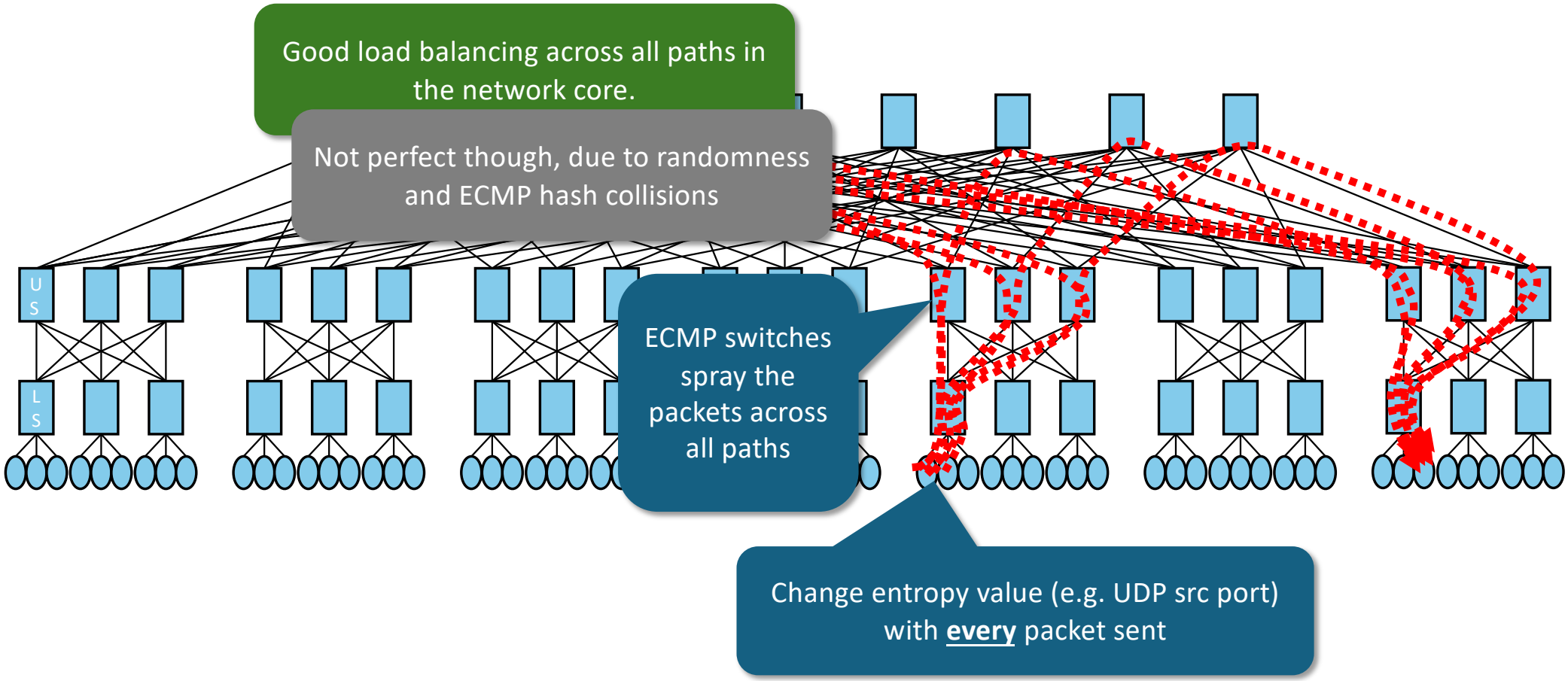
Spraying with asymmetries: unequal capacity links



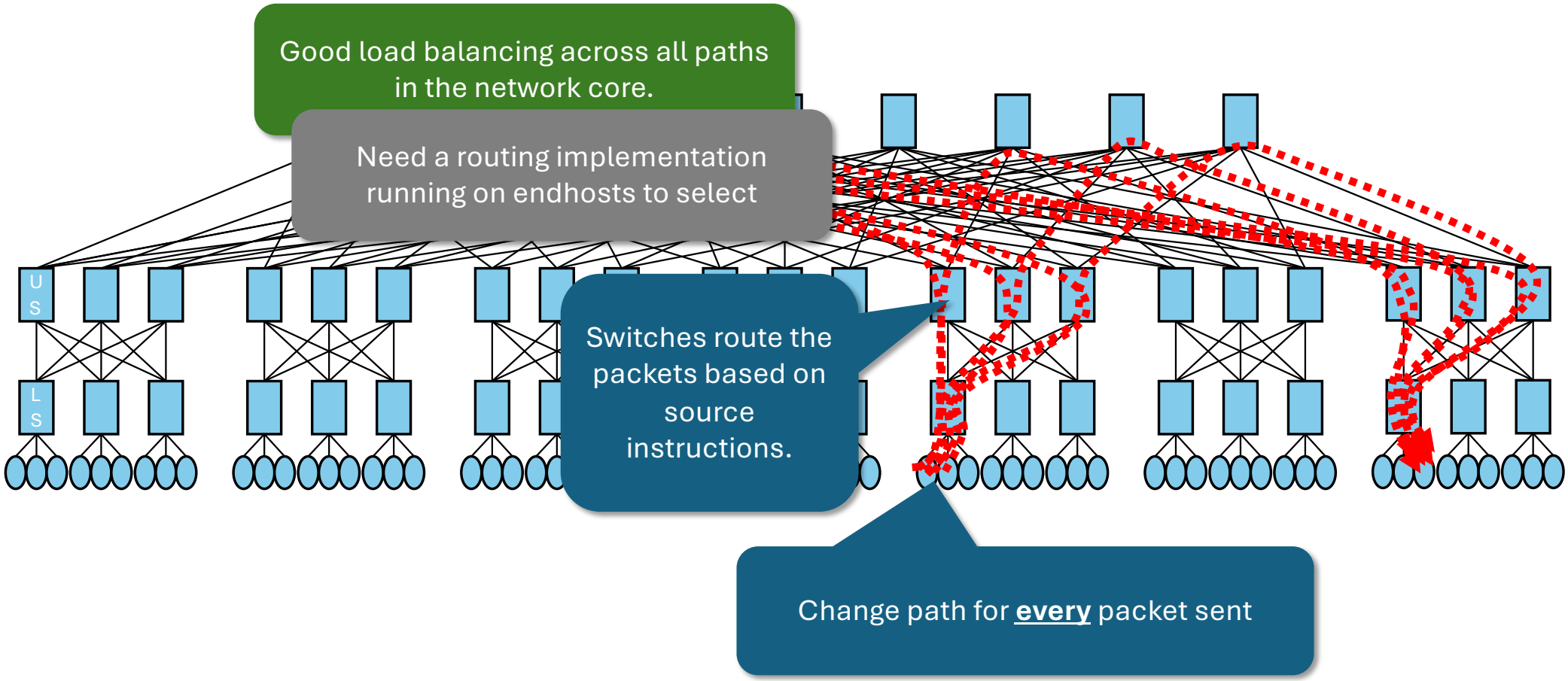
Spraying: picking a path for each packet

Who picks path?	How is path enforced?	Per path state?	Example solution
Host	Source routing	**Oblivious**	NDP, Homa
	Entropy + ECMP	**State-aware**	Multipath TCP Falcon Bitmap, REPS [UET]
Switch		**Oblivious**	RPS, pHost Round robin W/ECMP
		Load-aware	Drill Adaptive Routing / Dynamic Load Balancing Global Load Balancing

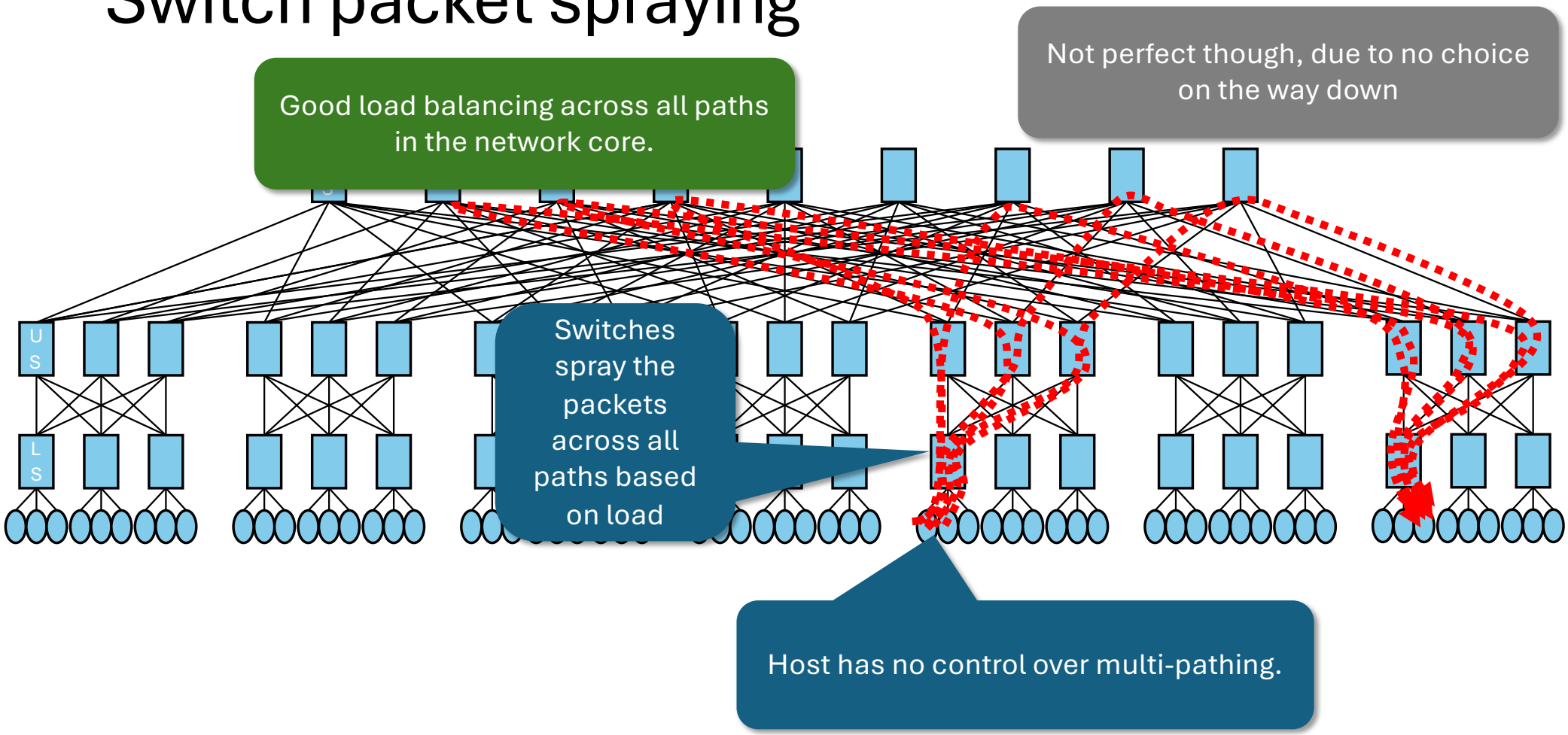
End-to-end packet spraying via ECMP



End-to-end packet spraying with source routing.



Switch packet spraying

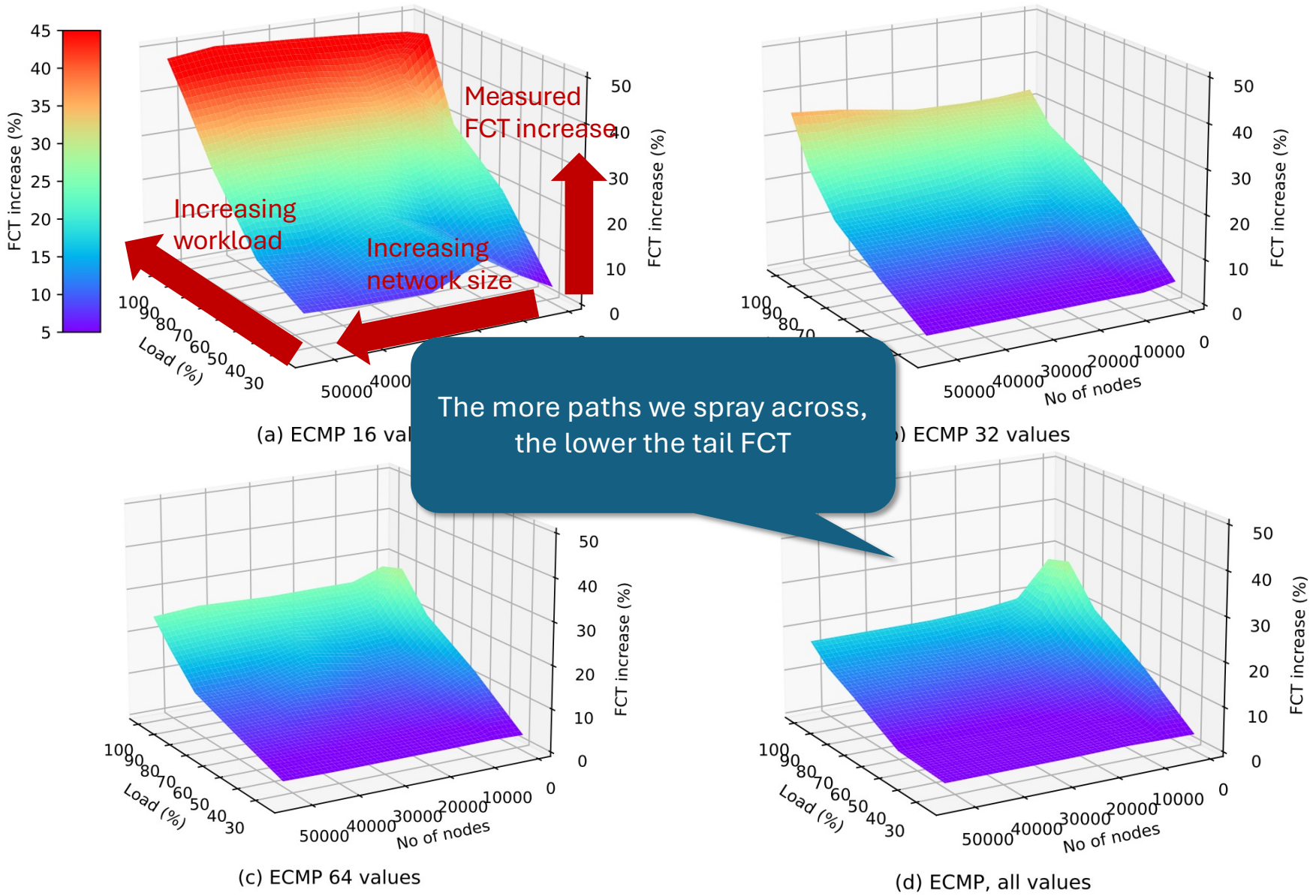


Host based spraying
What is the best way to spray packets?

- **Simplest: oblivious load balancing**
  - Pick a random EV for each packet.
  - Works very well if network capacity is uniform.

- Bitmap load balancing (e.g. UET bitmap algorithm)
  - Per EV state - one or a few bits.
  - When ACK indicates ECN mark, increment EV state.
  - When EV is next to be picked but non-zero state, decrement state, skip.

- Recycled entropies (REPS, UET):
  - Keep EV cache for which we got an ACK without ECN set.
  - Path selection: pick EV from cache if non-empty. Otherwise pick random EV.


FCT increase (%)
Measured FCT increase
Increasing workload
Increasing network size
Load (%)
No of nodes
The more paths we spray across, the lower the tail FCT
(a) ECMP 16 val
) ECMP 32 values
(c) ECMP 64 values
(d) ECMP, all values

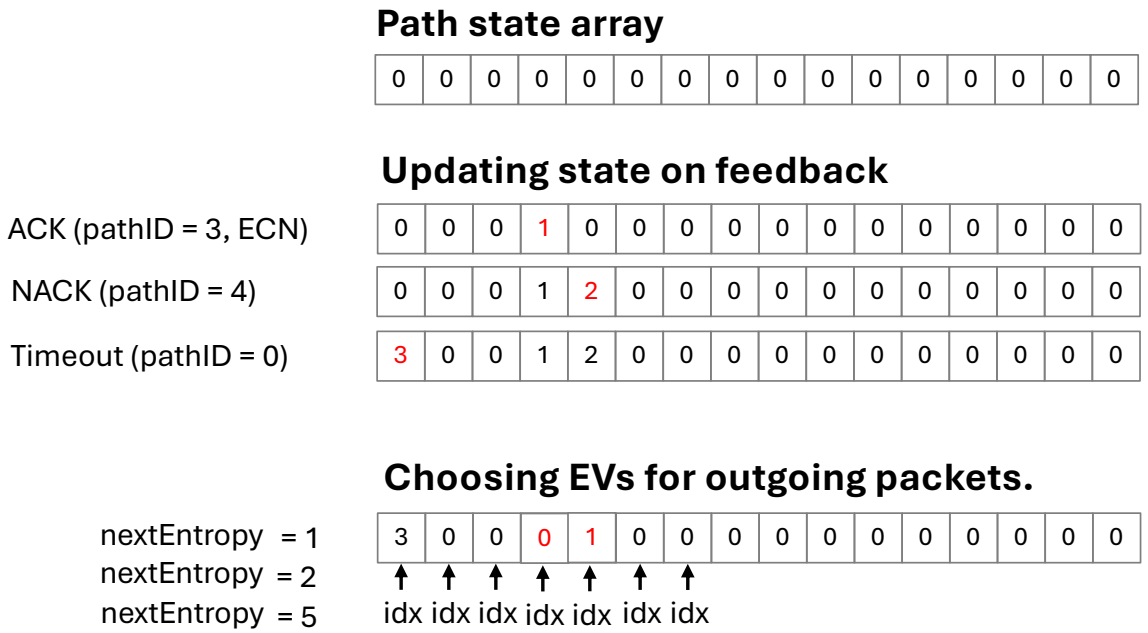
State-based host based spraying

- Bitmap load balancing (e.g. UET bitmap algorithm)
  - Per EV state - one or a few bits.
  - When ACK indicates ECN mark, increment EV state.
  - When EV is next to be picked but non-zero state, decrement state, skip.

- Recycled entropies (REPS, UET):
  - Keep EV cache for which we got an ACK without ECN set.
  - Path selection: pick EV from cache if non-empty. Otherwise pick random EV.

Host stateful load balancing: bitmap [UET]

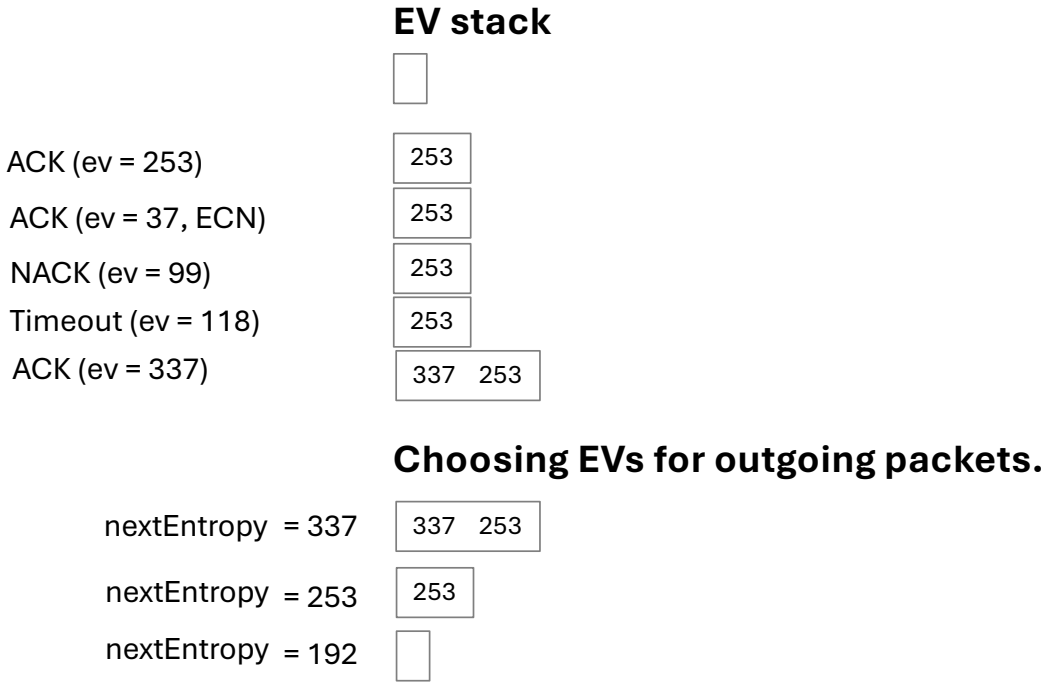
Path EVs chosen at connection setup. 2 bits of state per EV (0=good; 1,2=congested; 3=down)



Details not covered: from path IDs to entropy values. Random permutation through path IDs.

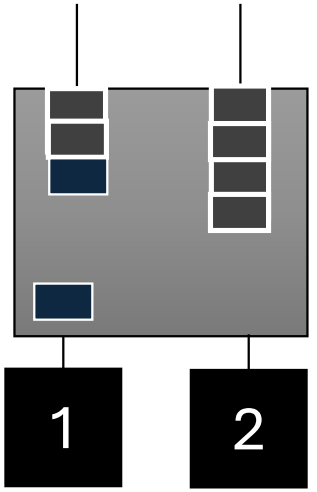
Host stateful load balancing: recycled entropies [UET]

Keeps stack of good EVs. Uses them for new EVs, otherwise generates random EVs.

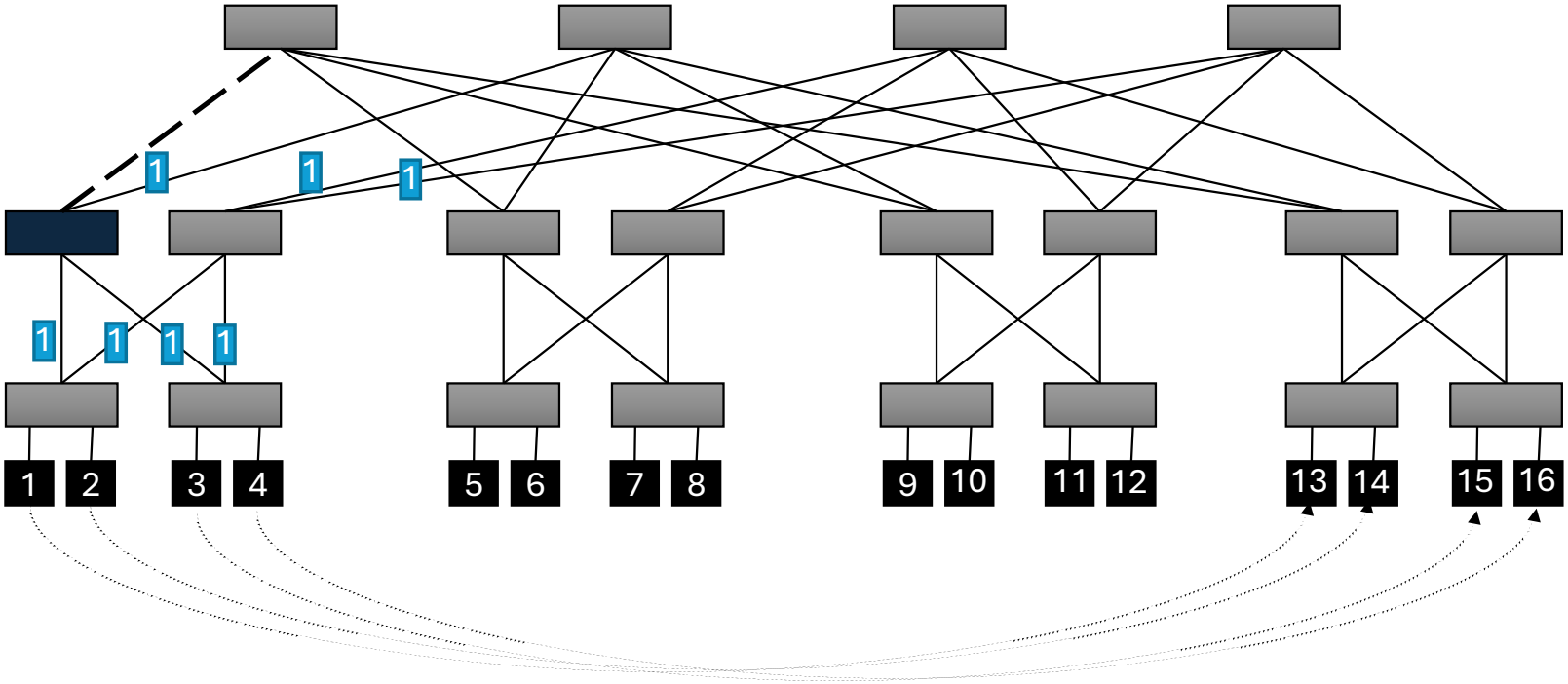


How does switch spray work?

- ECMP group => DLB group.
- During route lookup, routes in DLB group are consulted.
  - Contains all available paths towards destination.
  - Switch uses local information to decide which route (and associated egress port) to pick
  - Example metrics:
    - Queue length
    - Bandwidth utilization
    - PFC Port state.
    - Combination of the above possible.
- Works very well when path choice exists (e.g. up the tree).
- Less well on the downward path / with asymmetries.
  - At the limit, behaves like oblivious endhost spraying.



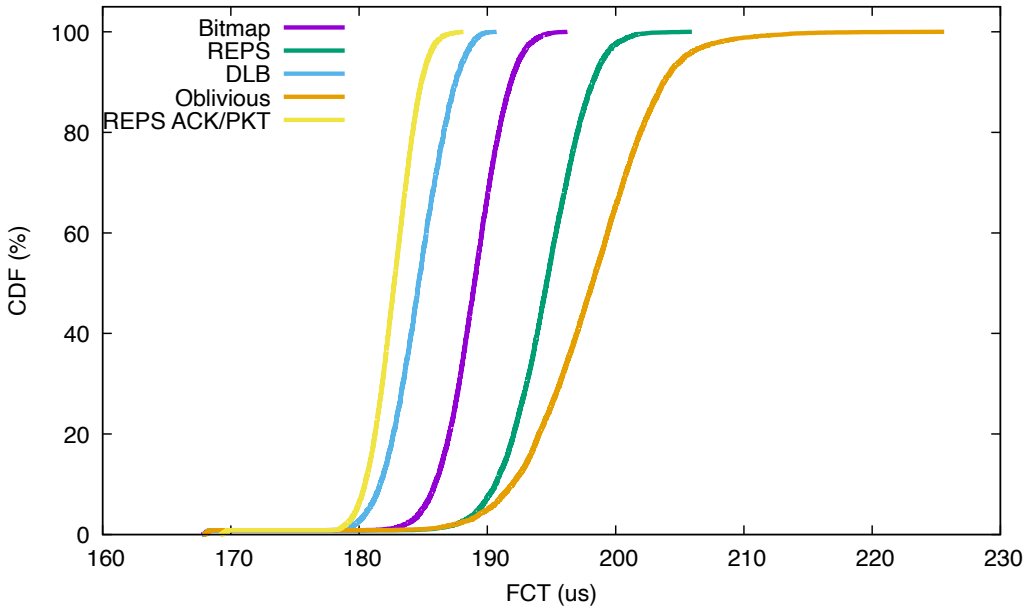
Switch spray with asymmetries



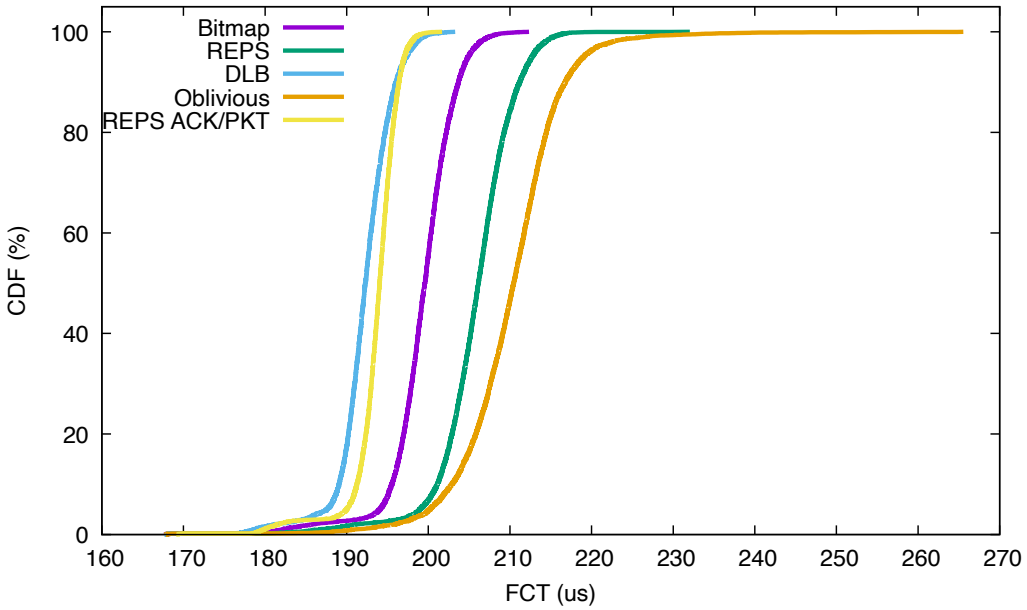
OK, so how does this look in simulation?

- 8192 nodes; 100Gbps topology, no failures. Trimming at 1BDP.

Two tier topology



Three tier topology

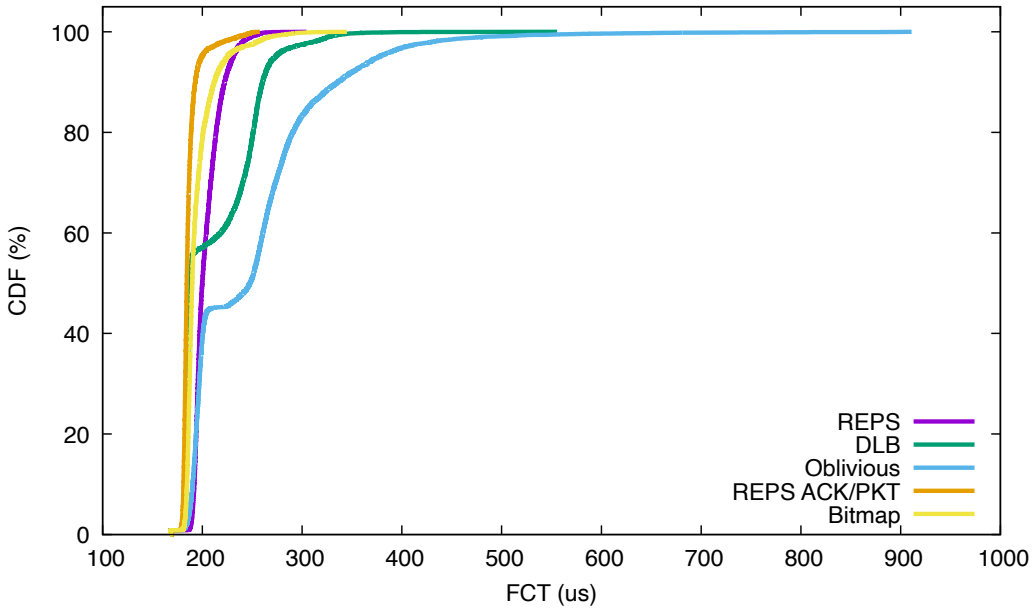


Take aways:

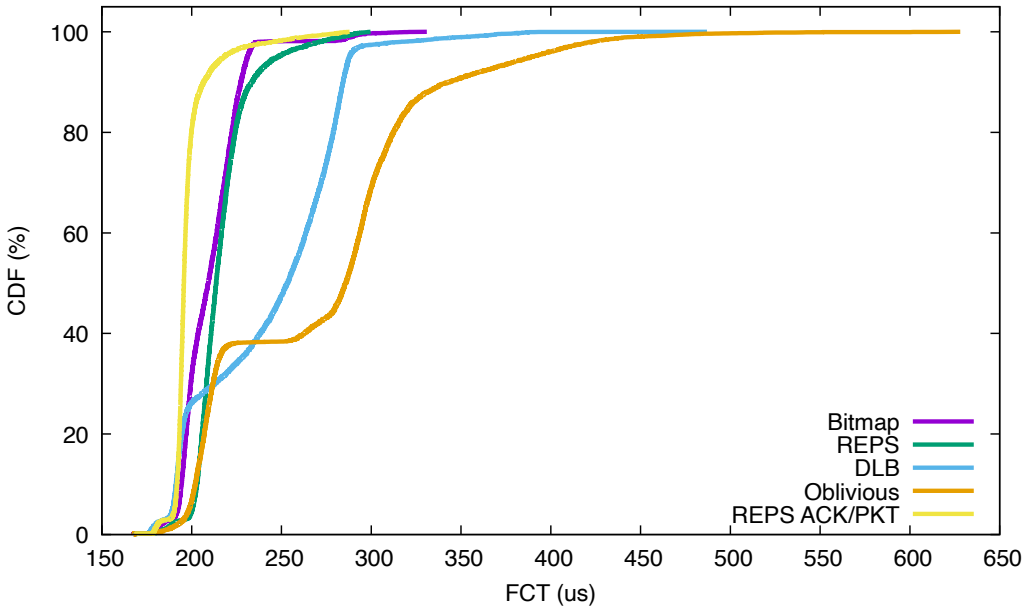
- Random collisions slightly increase FCTs even with spraying.
- Very similar performance for host-stateful and DLB. Host oblivious good-enough.
- REPS requires per packet ACK to work (really) well.

1% of top tier links are blackholes.

Two tier topology



Three tier topology

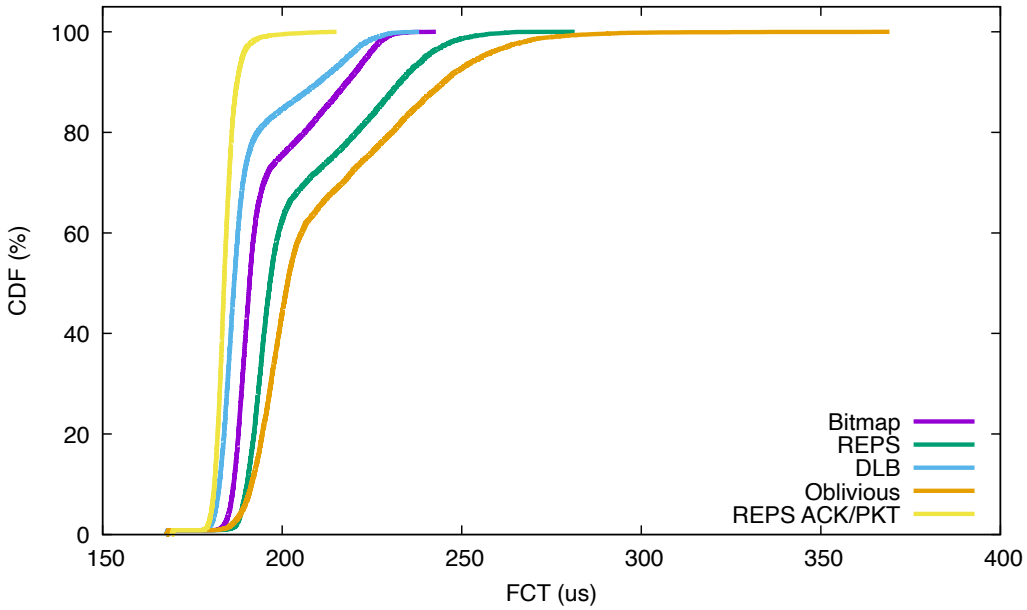


Take aways:

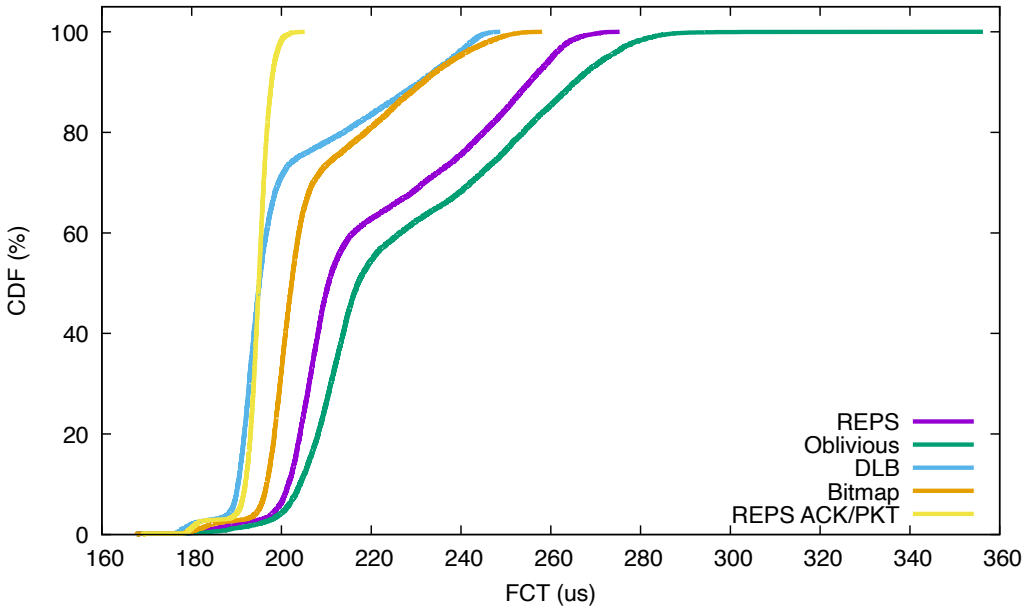
- Oblivious host spraying results in continuous timeouts which affect FCT.
- Switch spraying using only local information affected heavily too on downlinks.

1% of top-tier links have reduced capacity (25%)

Two tier topology



Three tier topology

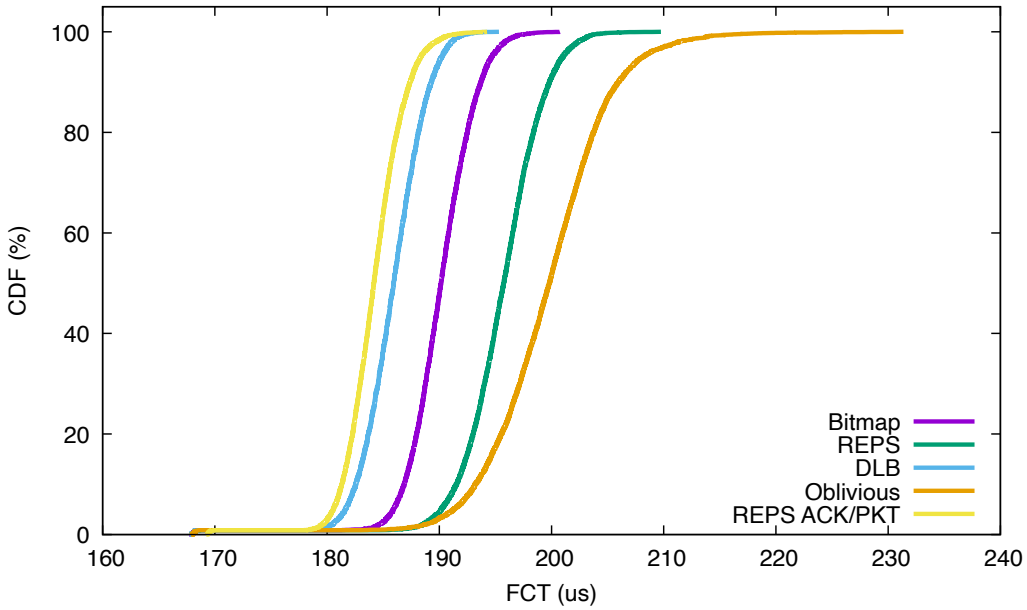


Take aways:

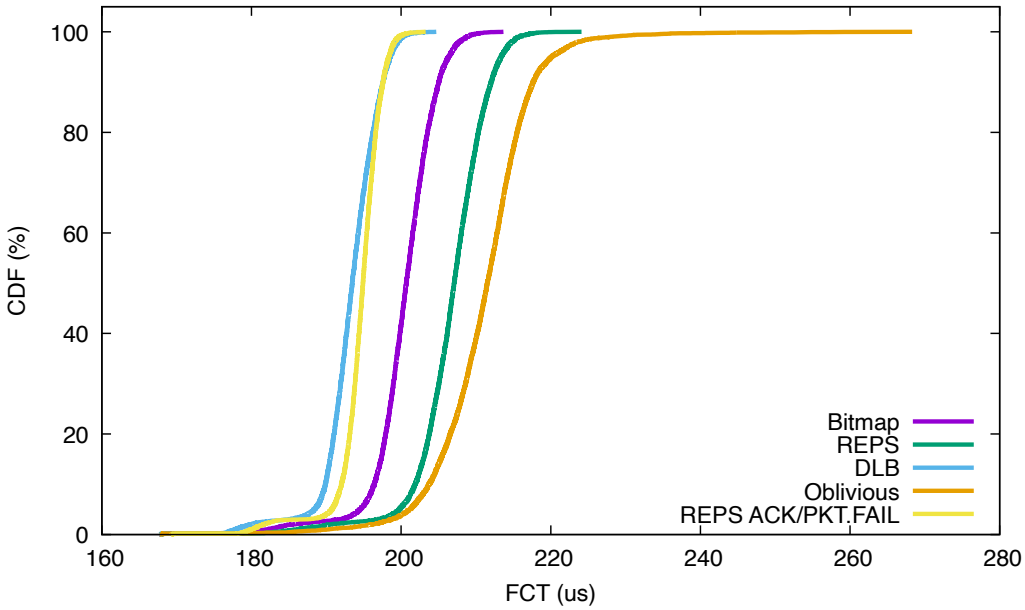
- Oblivious host spraying and switch spraying suffers from increased FCT.
- REPS within 10% of optimal. Bitmap struggles to load balance effectively.

1% of top-tier links failed and removed from routing

Two tier topology



Three tier topology

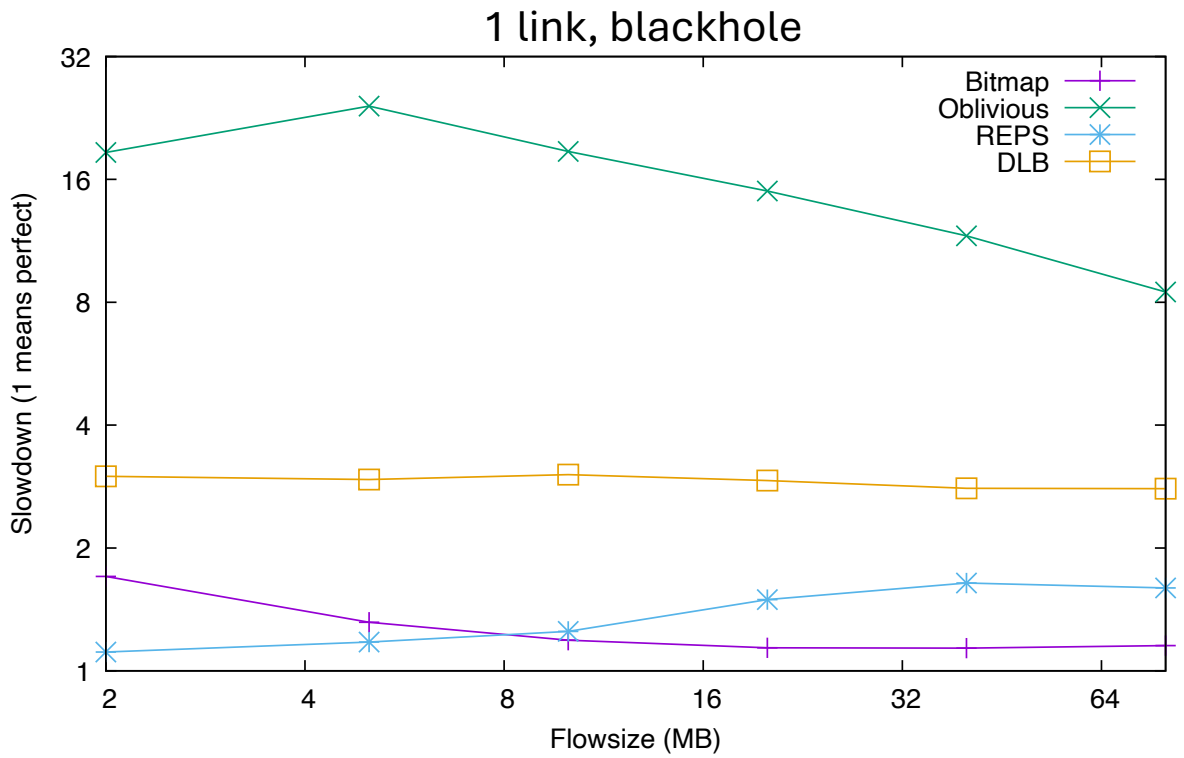


Take aways:

- Performance after routing reconvergence is quite good.
- No big differences from no failure case.

Does the size of the flows change the relative performance?

- 32 nodes; 100Gbps topology, single failed link. Leaf-spine. Trimming at 1BDP. Average of 10 runs.

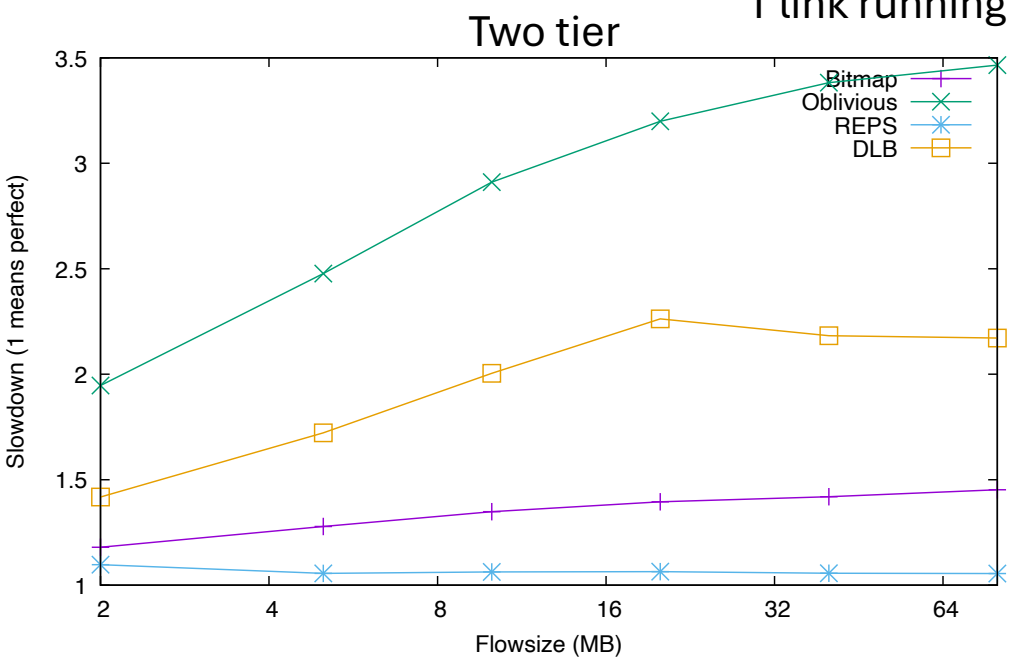


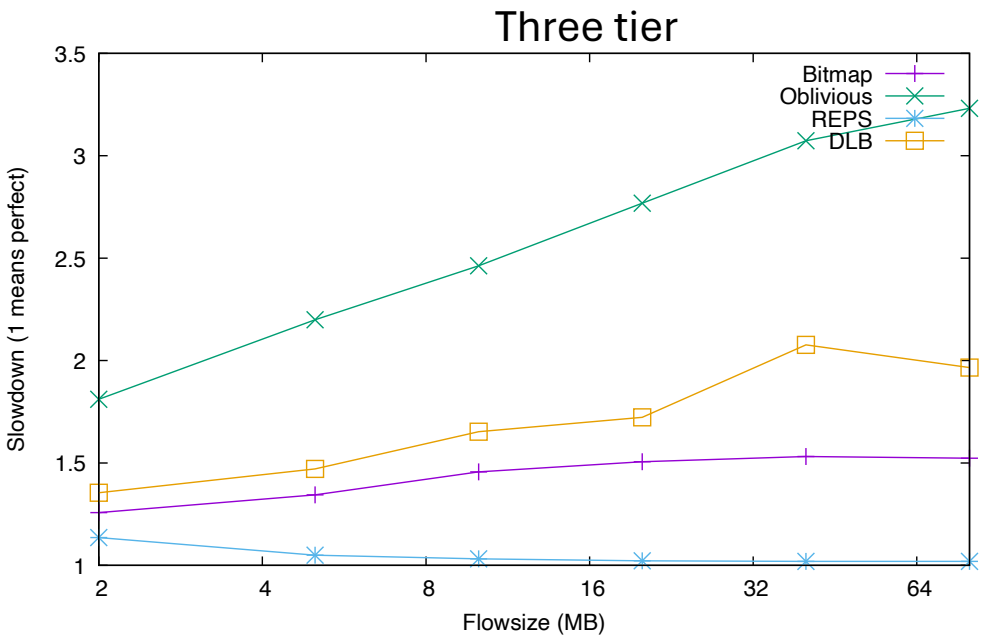
Take-away: bitmap becomes better than REPS from longer flows as it stops using bad paths after 1 RTO

Does the size of the flows change the relative performance?

- 32 nodes; 100Gbps topology, single failed link. Trimming at 1BDP.. Average of 10 runs with different seeds.

1 link running at 25%





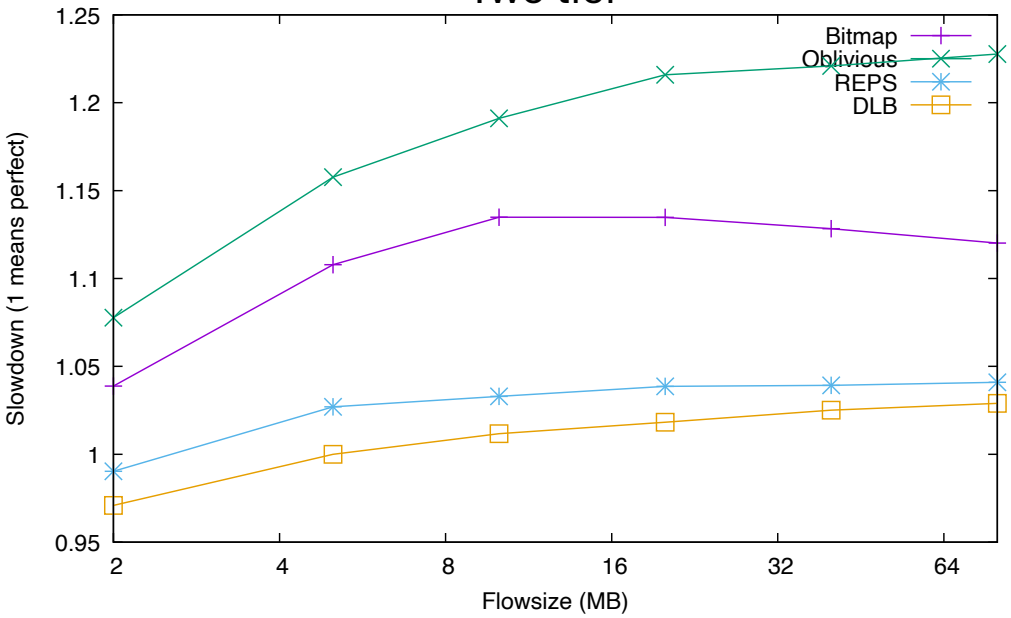
Take-away: DLB is susceptible to FCT increase in asymmetric networks compared to host-based spray.

Does the size of the flows change the relative performance?

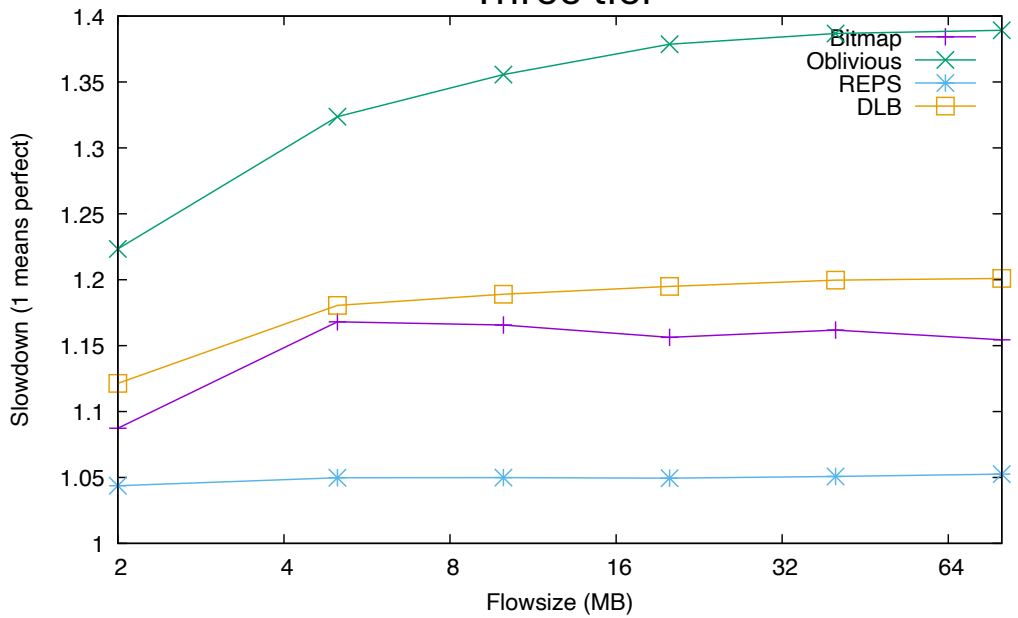
- 32 nodes; 100Gbps topology, single failed link. Trimming at 1BDP.. Average of 10 runs with different seeds.

1 link failed, routing reconverged.

Two tier



Three tier



Take-away: Long term failures hurt DLB in three-tier networks, less so in two tier ones.

Spraying is emerging as the go-to technique for AI

- Spraying:
  - Single congestion window plus load balancing to decide packet to path mapping.
  - Enables exploring many paths,
  - Multipath transport with less state per connection.
- The key to spraying is the load balancing algorithm.
- Switch spraying works very well in symmetric networks.
  - Less well with black holes or asymmetric capacity links.
  - Additional mechanism needed to handle these cases:
    - e.g. Broadcom chips have Global Load Balancing.
- Host spraying works almost as well and also handles asymmetries
  - It is harder to implement in the NIC.
- No clear winner yet.

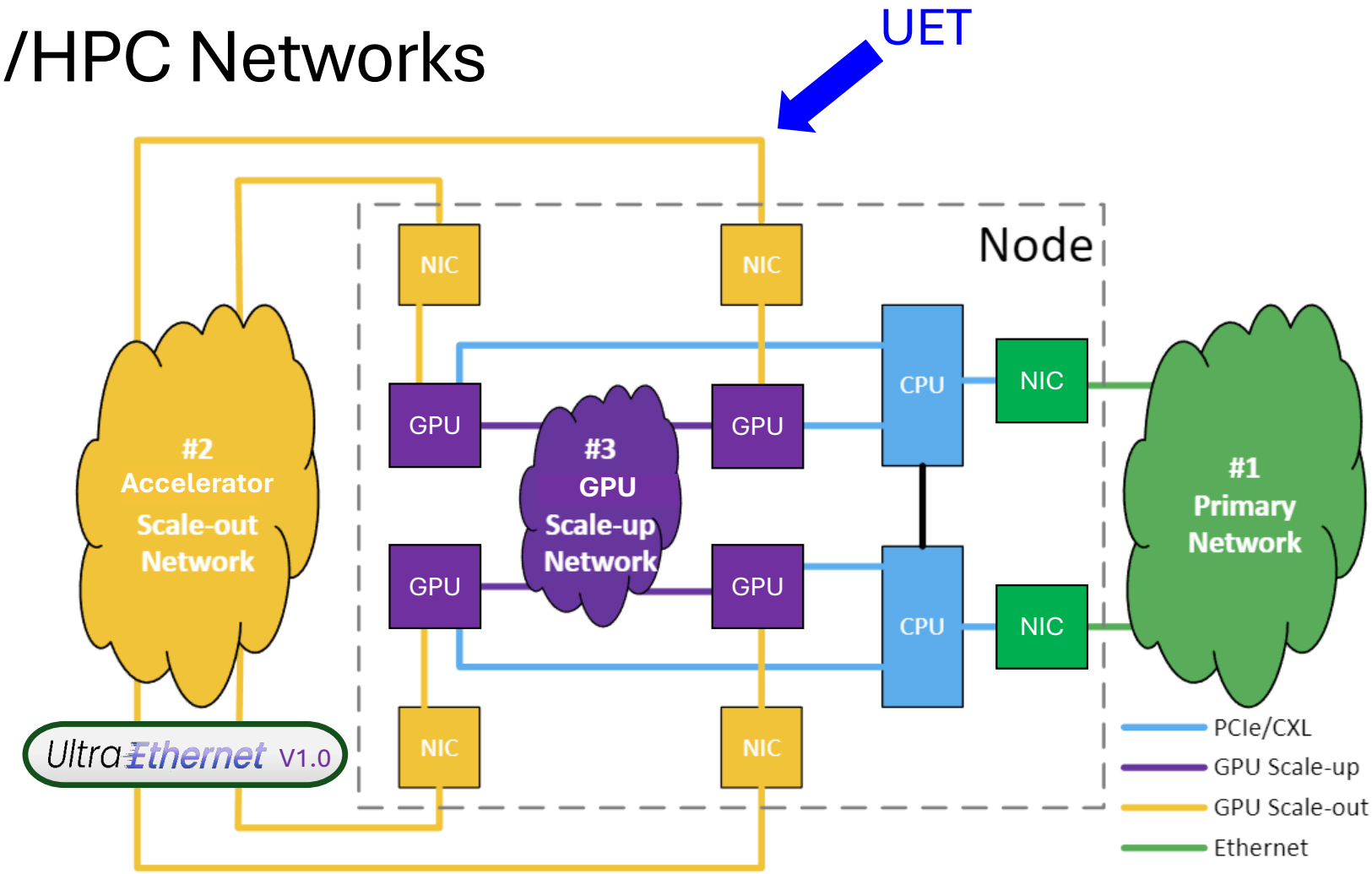
Ultra Ethernet Consortium

v1.0 spec now available

- Industry forum , standardizing next-gen protocols for high-performance large scale AI networks.
  > 200 members, inc Broadcom, Nvidia, Intel, AMD, MSFT, Google

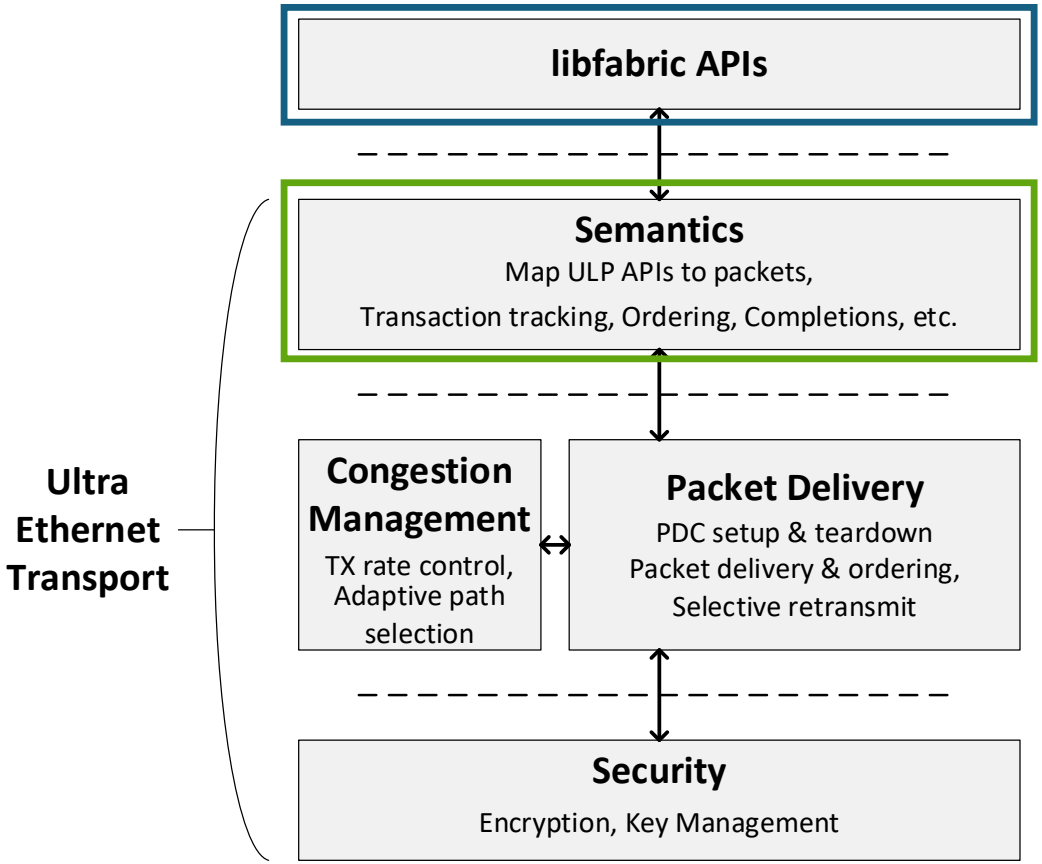
- New transport protocol, UET, features:
  - Replacement for RDMA
  - Packet spraying
  - Trim-NACK, Selective ACK and selective fast retransmit
  - Two new congestion control schemes
- New network features:
  - Packet Trimming
  - Link-layer Retransmit
  - Congestion Signalling (CSIG)

AI/HPC Networks



UEC v1.0 targets the Scale Out Network

UET – UltraEthernet Transport

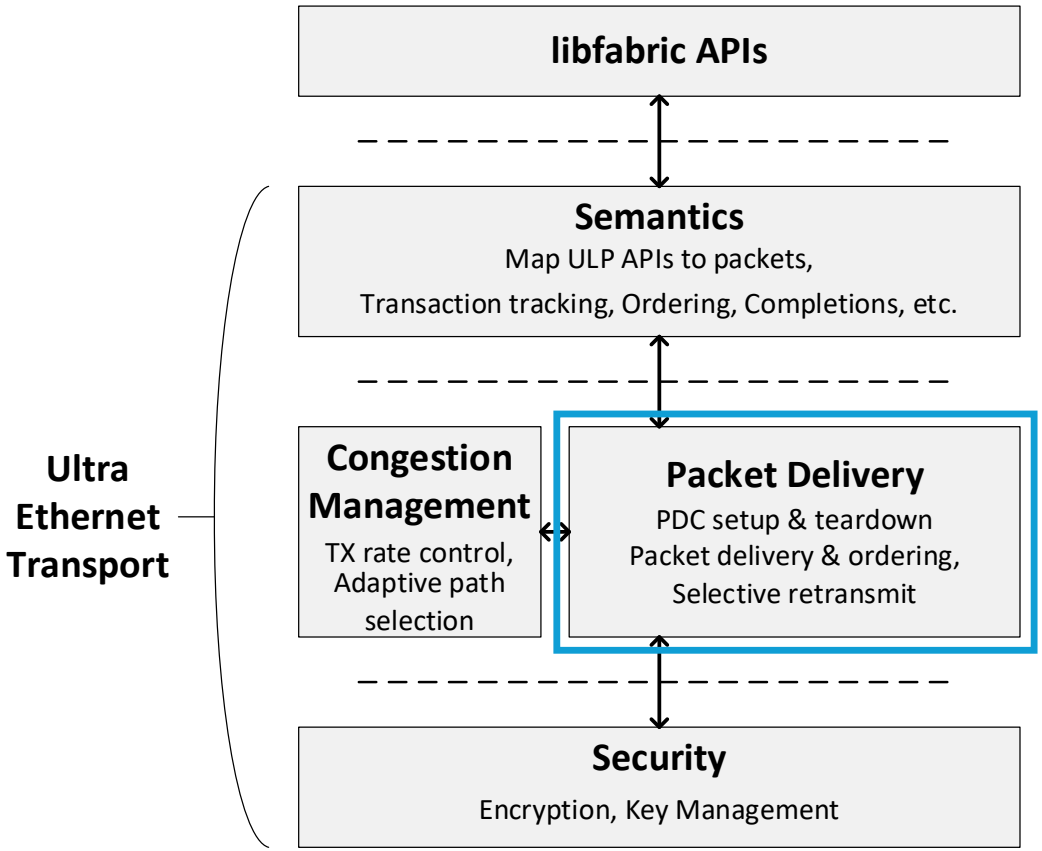


High scale, high utilization, low tail latency

- Simplify adoption by using existing, open standard
- Easy to port applications

- RDMA services: Send/Rcv, Write, Read, Atomics
  - Focus on MPI and *CCL
  - Shared receive queue
- Scalable addressing
- Optimized extensions:
  - Deferred Send improves RNR
  - Rendezvous using Send/Read
  - Exact match tags for HW offload of ordering between endpoints using shared receive queues

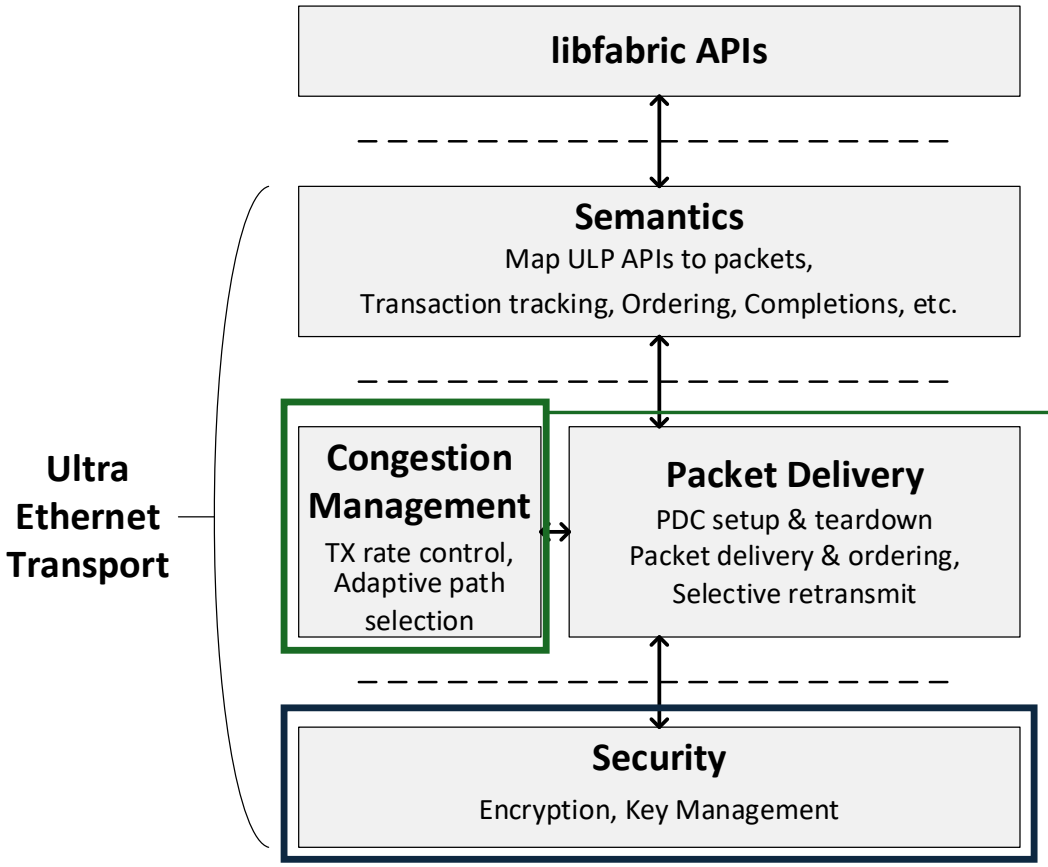
UET – UltraEthernet Transport



- Dynamic, ephemeral connections
  - Zero start up time, 1-RTT close
- 4 delivery services:
  - ROD – Reliable, ordered
  - RUD – Reliable, unordered
  - RUDI – Reliable, unordered, idempotent (Write/Read)
  - UUD – Unreliable, unordered
- Out-of-order packet arrival
- Selective acknowledgement and retransmission for RUD & RUDI
  - ROD uses Go-BackN

High scale, high utilization, low tail latency

UET – UltraEthernet Transport



- Multipath with congestion avoidance
  - Leveraging ECMP
- Trimming with NACK signal
- Network Signaled CC (NSCC)
  - Window based at sender using RTT and ECN
- Receiver Controlled CC (RCCC)
  - Credit based at receiver

- End-to-end AES encryption
- Key derivation for additional security
- Replay protection
- Scalable security domains
- Optional within UET

High scale, high utilization, low tail latency

UET Congestion Control Goals

- UET designed primarily for best-effort FatTree style networks
  - Many equal-cost paths from source to destination.
  - Need to load balance evenly to avoid congestion due to flow collisions.
- To minimize latency, default to starting at line rate.
  - Sometimes will cause congestion – need to react very fast when this happens
- Workloads often create small incasts.
  - Try to avoid large incasts in application.
  - 10:1 common, 100:1 should be handled gracefully.
  - Incast can be an emergent property of all-to-all.
- Some networks are oversubscribed.
  - Intent is for oversubscription to never cause congestion (or you'd have oversubscribed less - GPUs are expensive!)
  - But when it happens, core congestion should be handled gracefully.

NSCC (Network-signal based CC)

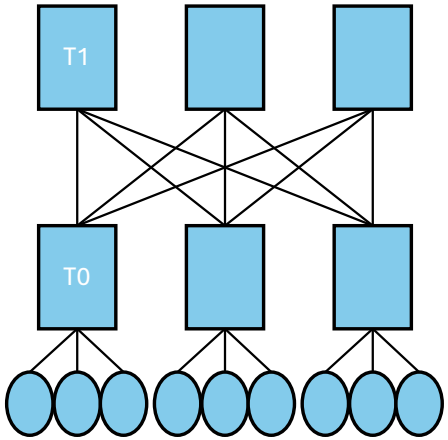
- Assumption: ECN is set based on queue size in switch at dequeue
  - ECN is a leading indicator of congestion.
- Delay is a lagging multi-bit indicator of congestion level
- Use both to adjust window of data allowed in flight

	Delay ~0 for some time	**Delay < target**	**Delay > target**	**Delay > FC target (or trim / drop)**
No ECN	(underloaded) fast increase	(uncongested) proportional increase	(congestion gone away) fair increase	N/A
ECN	N/A	(new congestion) fair decrease	(congested) multiplicative decrease	(overload) fast convergence to delivered bytes

Congestion Signaling

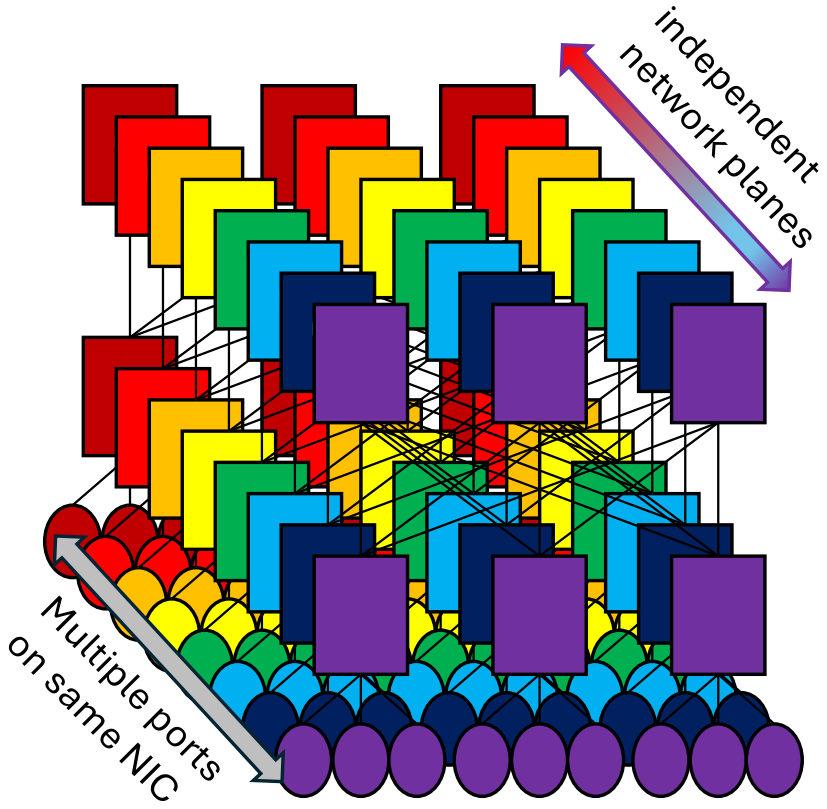
- Please refer to separate slides from Brad Karp (Google) on this topic.

Multi-plane topology



- Three-tier topologies add extra latency.
  - Can we scale to >100K GPUs in two tiers?
- TH5 switch can support 64 ports at 400Gb/s, but 512 ports at 100Gb/s.
  - 256 GPUs per T0
  - T1s downlinks per T1 => 512 T0s per network
  - 256x512 = 131K GPUs per network.
  - But slow network links!

Multi-plane topology



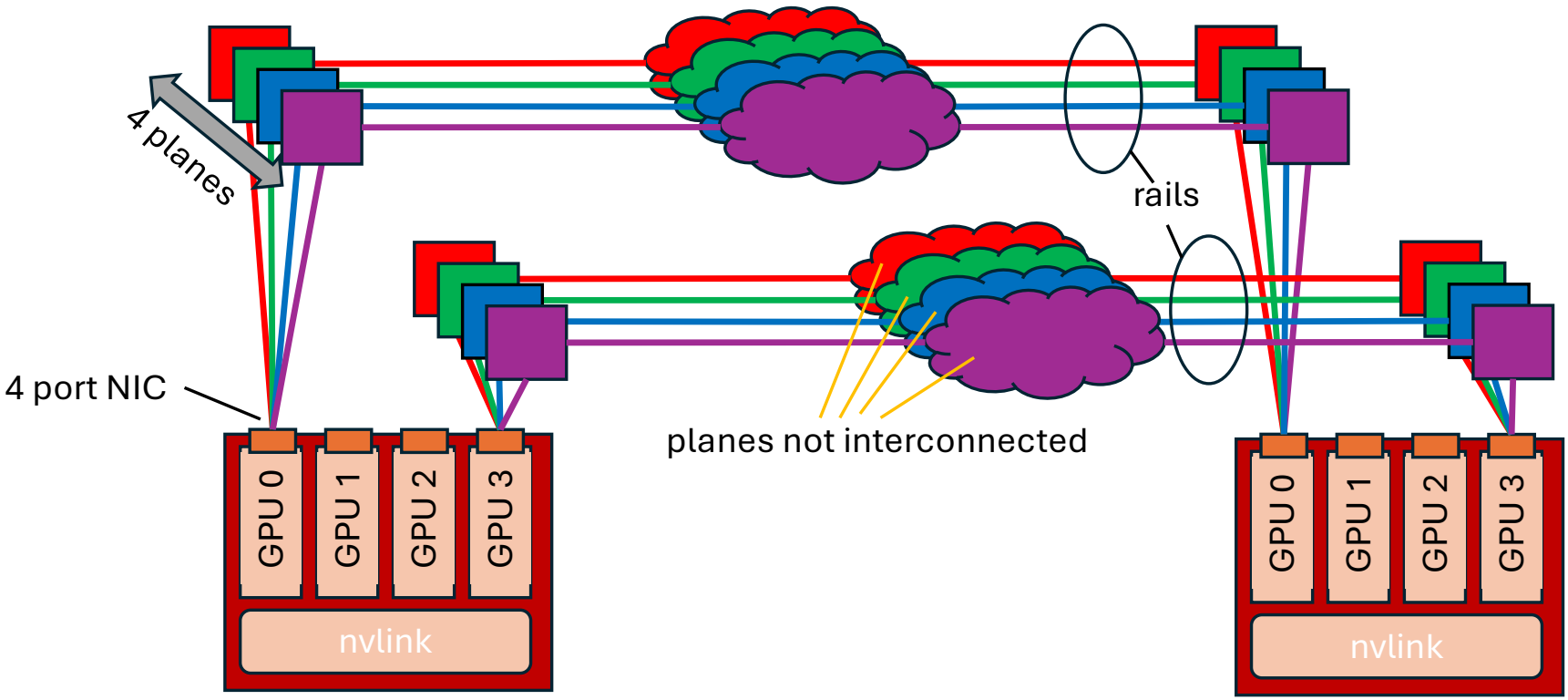
Build an 8-port UEC-capable NIC.

- 8 x 100Gb/s ports
- Spray each flow across all 8 ports in hardware
- Build 8 separate parallel two-tier networks

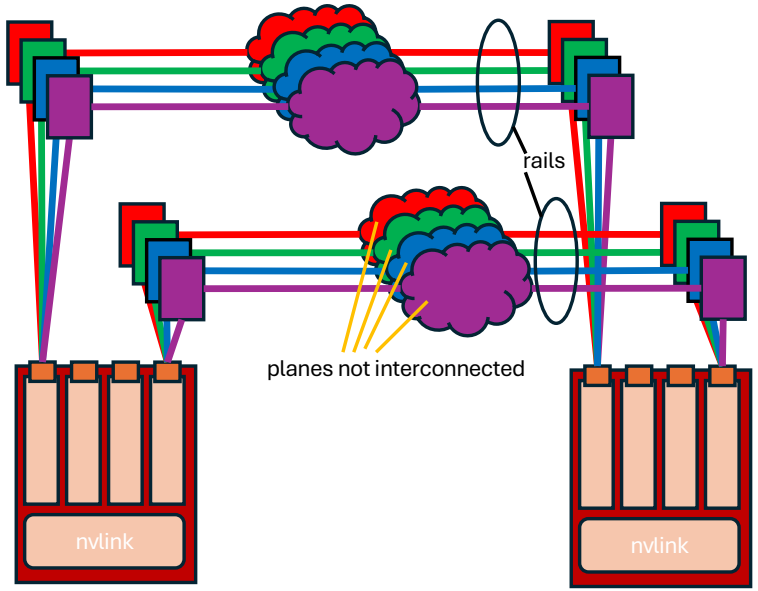
Connects 131K GPUs

- each flow can achieve 800Gb/s by using all eight networks in parallel

Multi-plane, multi-rail topology

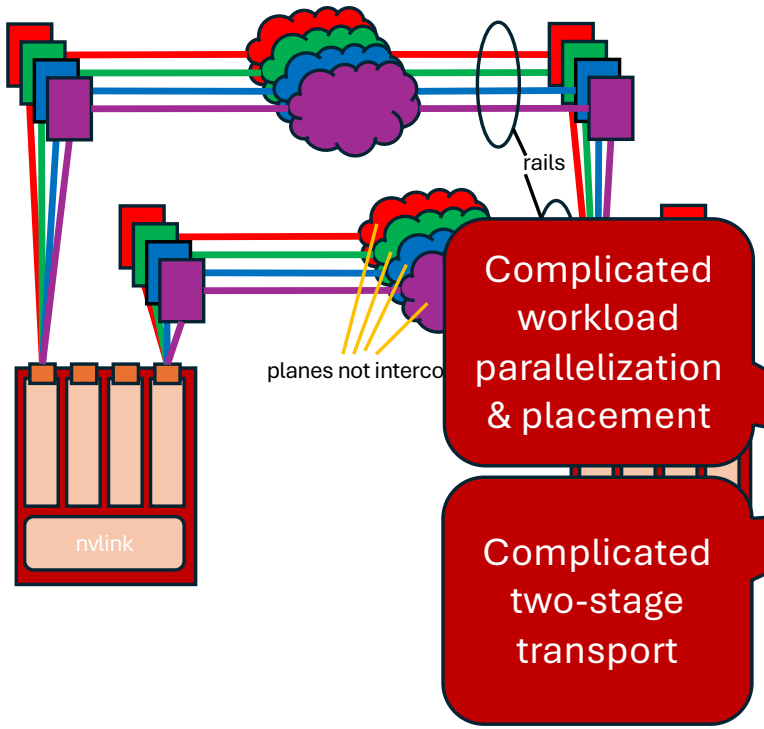


Multi-plane, multi-rail topology



- H100 GPUs can support 8 GPUs via nvlink scale-up
- Connect all the GPU0s in each scale-up network to each other via an 8-plane scale-out network
  - We call this a “rail”
  - Similarly for all the GPU1s, etc.
- Can’t get from GPU0 in one rack to GPU1 in another rack via the scale-out network
  - Can place the workload so you never need to
  - Or go via scale-up, then scale-out
- But can support 1 million GPUs in a two-tier network

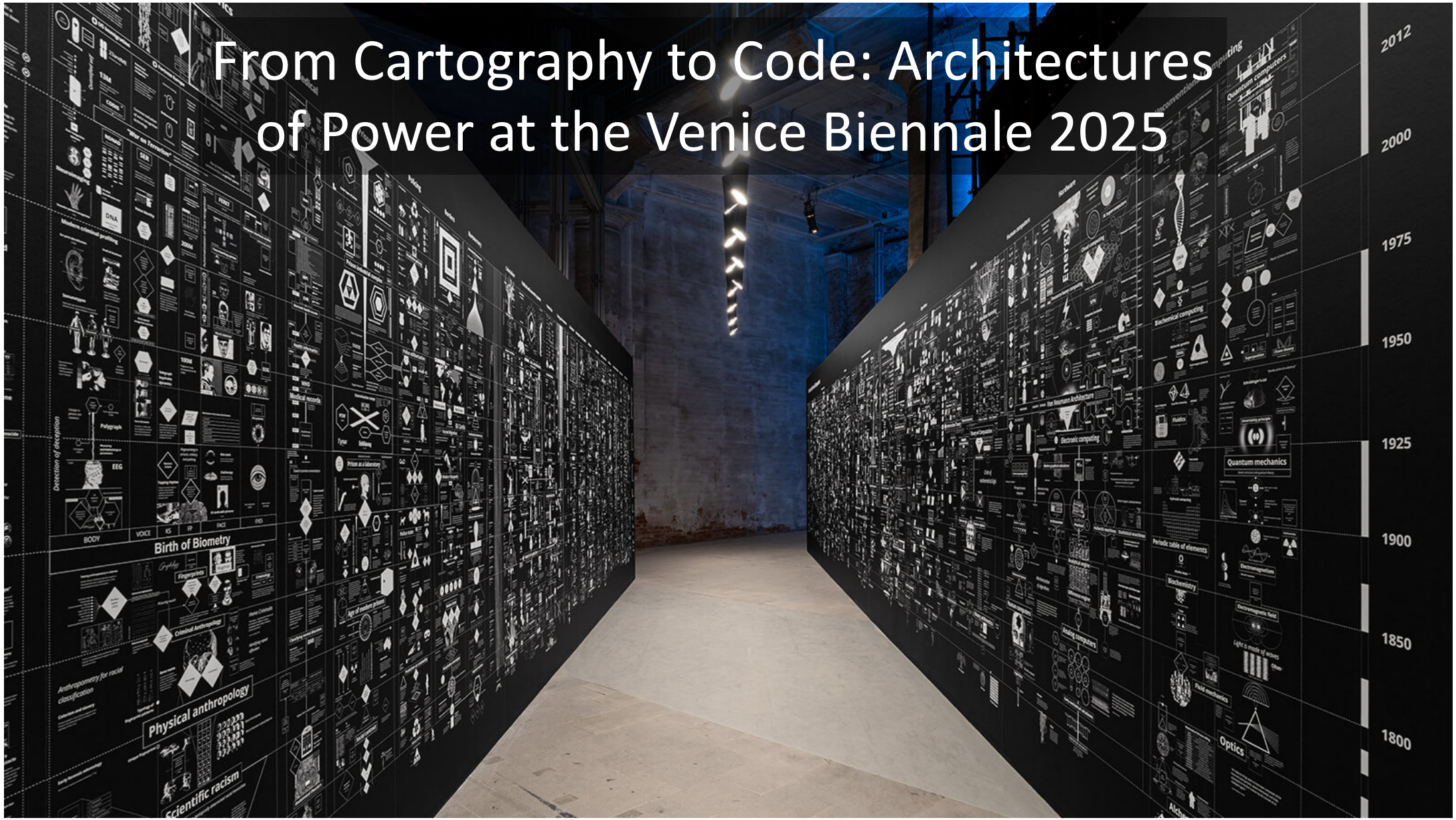
Multi-plane, multi-rail topology

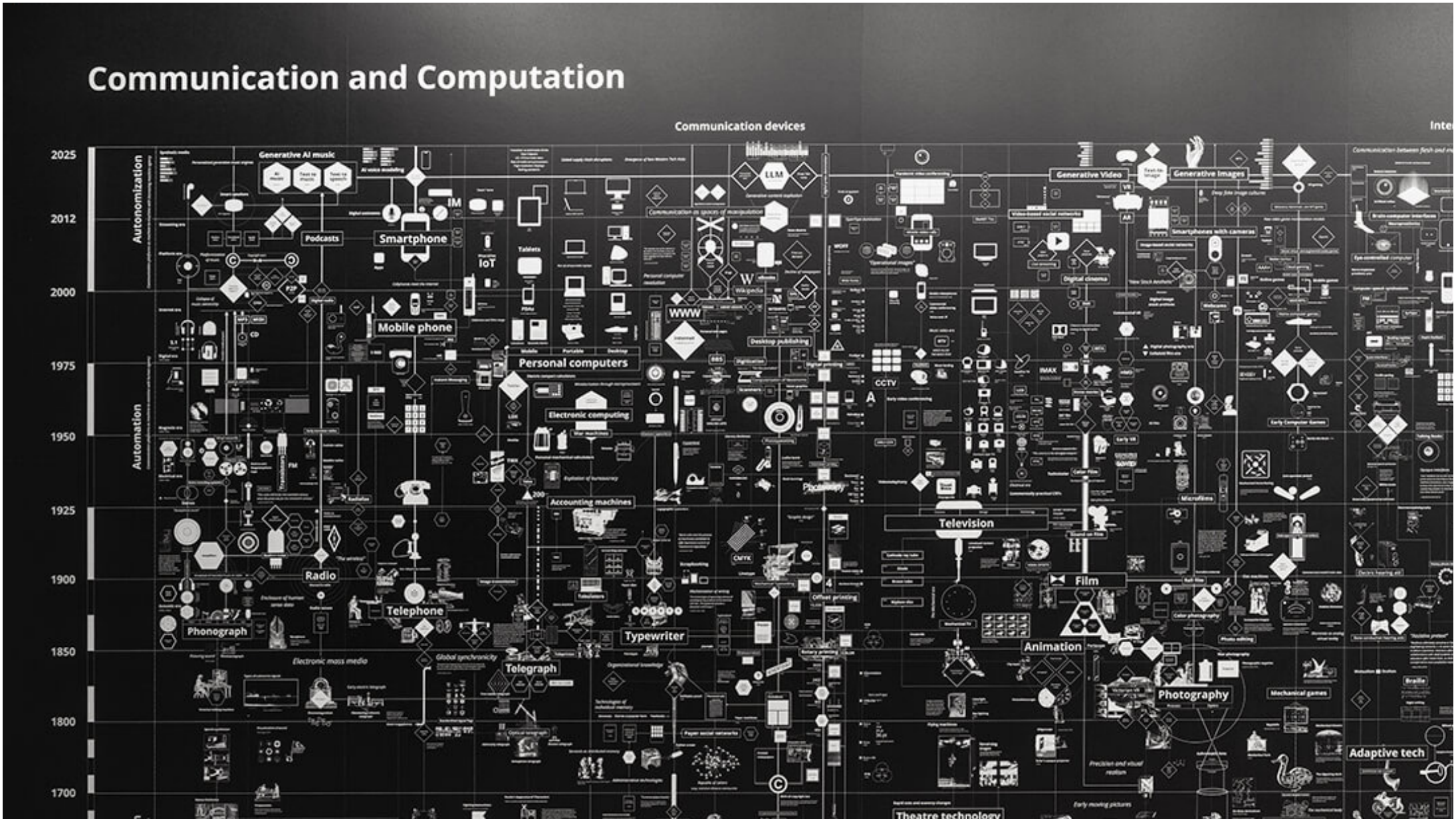


- H100 GPUs can support 8 GPUs via nvlink scale-up
- Connect all the GPU0s in each scale-up network to each other via an 8-plane scale-out network
  - We call this a “rail”
  - Similarly for all the GPU1s, etc.
- Can’t get from GPU0 in one rack to GPU1 in another rack via the scale-out network
  - Can place the workload so you never need to
  - Or go via scale-up, then scale-out
- But can support 1 million GPUs in a two-tier network

Trends in Ethernet Switching for AI

From Cartography to Code: Architectures of Power at the Venice Biennale 2025




Communication and Computation
Communication devices
2025
2012
2000
1975
1950
1925
1900
1850
1800
1700
Automonization
Automation
Generative AI music
Podcasts
Smartphone
IM
IoT
Tablets
Mobile phone
Personal computers
Mobile
Portable
Desktop
Electronic computing
Accounting machines
Radio
Telephone
Phonograph
Typewriter
Telegraph
Electronic mass media
Global synchronicity
Desktop publishing
WWW
Wikipedia
LLM
CCTV
Generative Video
Generative Images
Digital cinema
Smartphones with cameras
Television
Film
Animation
Photography
Color photography
Mechanical games
Adaptive tech
Braille
Theatre technology
Early moving pictures

What are the trends shaping datacenter switching sillicon?

Software control plane (2008)

- Software defined networking (Ethane, Openflow).
- Breaks monolithic coupling between switch software stack and switching ASIC.

Programmable dataplanes (2014)

- 2014: seminal P4 paper from McKeown, Vahdat, Rexford & al.
- Tofino and Tofino 2 from Barefoot Networks
- Trident switch product line from Broadcom programmable via NPL.

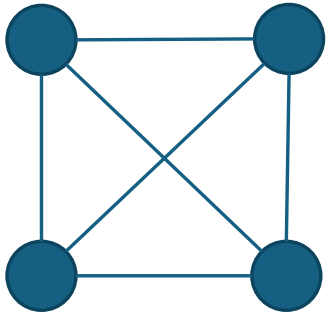
Main trend: economic efficiency

- More bandwidth, fewer switches.
- Merchant sillicon and white-box switches.

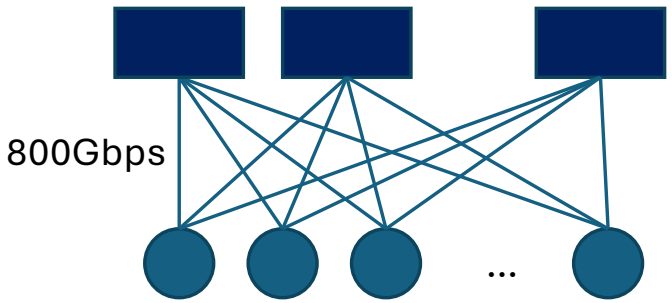
Network Landscape Changing Dramatically

	CPU Cloud Networks		Large-Scale GPU Networks
Network	Single network	→ → →	Scale-Up / Scale-out / Inter-DC
Topology	3-4 tier folded Clos	→ → →	2-tier Clos, Multi-rail, multi-plane
Switch support	ECN, PFC.	→ → →	Trimming, ECN, PFC.
Transport	TCP/IP, QUIC	→ → →	RoCE, Multipath / Ultra Ethernet
Transport impl.	Kernel (software)	→ → →	NIC
Datacenter Scale	100,000+	→ → →	~1M
Job Scale	10K	→ → →	~1M
Flows per host	High	→ → →	Low
Flow throughput	1-10Gbps	→ → →	Line-rate (e.g. 800Gbps)
Load Balancing	ECMP	→ → →	Multipath

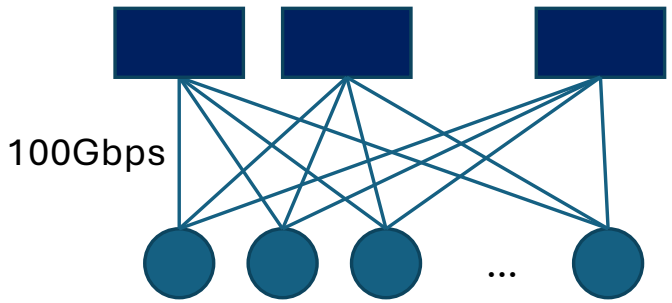
How do we build the scale-up network for 4 nodes?



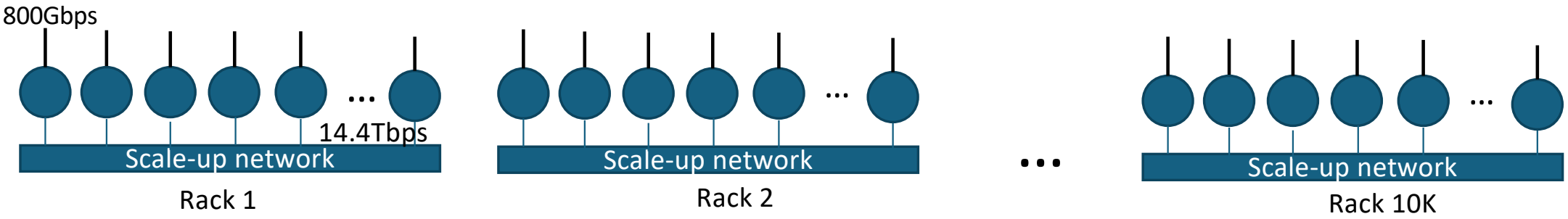
How do we build the scale-up network for 64 nodes?



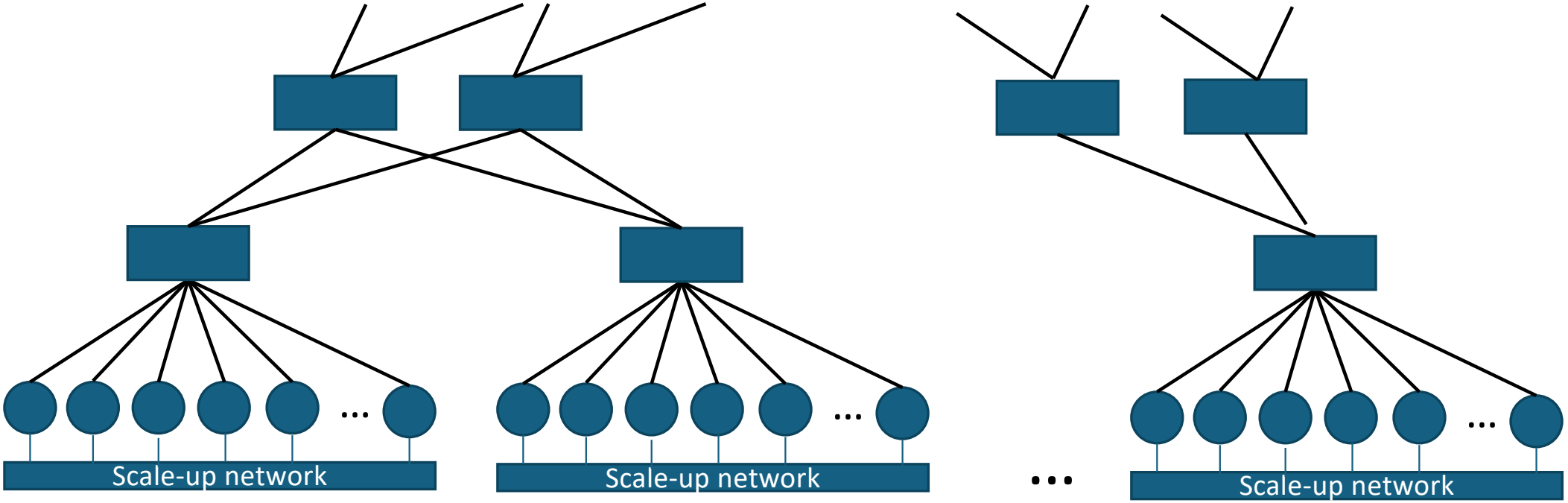
How do we build the scale-up network for 512 nodes?



How should we build the scale-out network for 1M nodes?



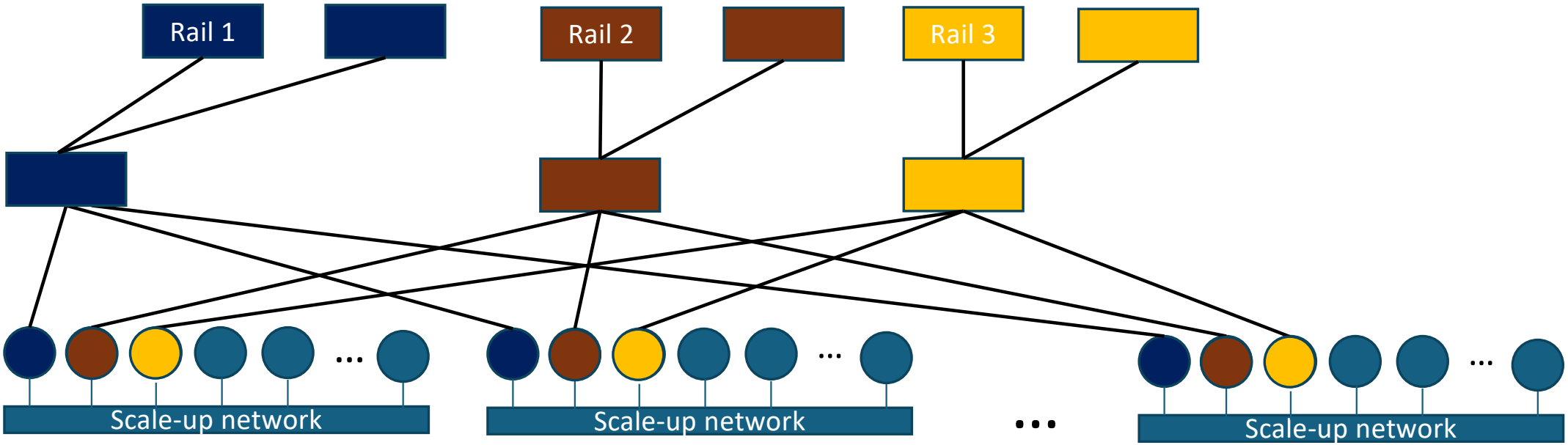
Default option: traditional multi-tier Clos



Max number of nodes ~ switch_radix $^{\#tiers}$

- 102.4T = 128 x 800Gbps ports => 4 tiers for 1M GPUs
- 55K switches, 4 cables / GPU (three or more optical).

Multi-rail: independent networks for subset of GPUs



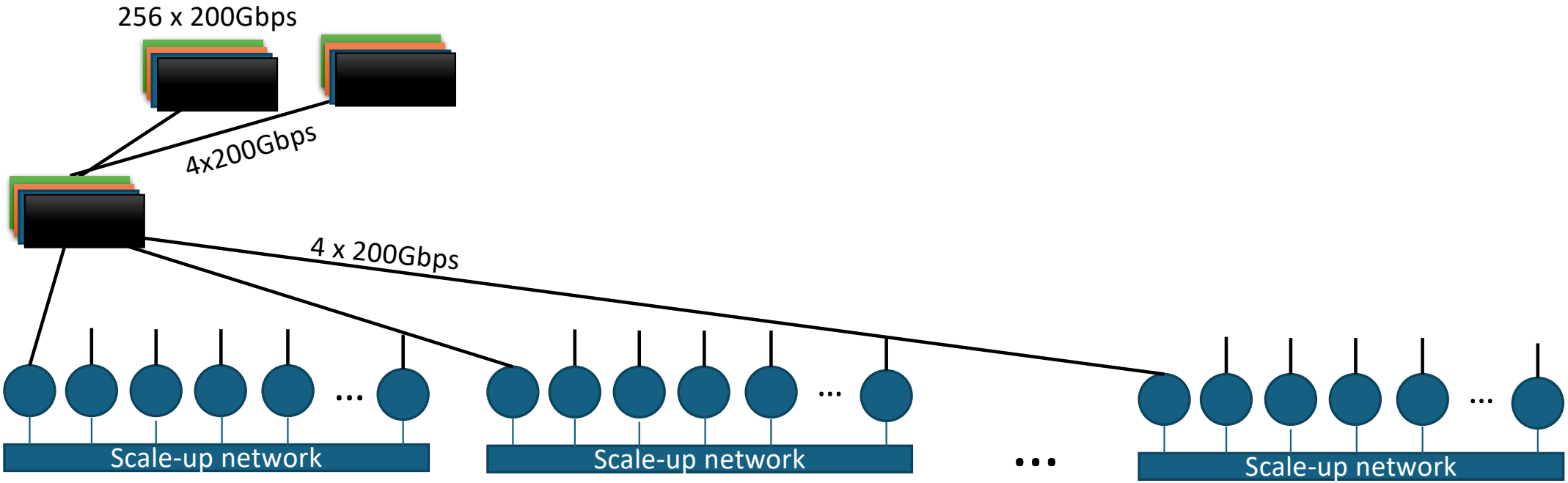
72 rails, ~14K GPUs per rail

- Three tier network sufficient; 1/3 cheaper than FatTree.

Challenge: no direct connectivity for most GPU pairs!

- Use scheduling + scale-up / scale-out flows instead.

Multi-plane: more switch chips in one box



102.4T = 128x800Gbps or 512x200Gbps!

Four planes, 200Gbps per plane - two tier network for 131K nodes.

8 rails + four planes = two tier network for 1M GPUs

Challenge: load balance, monitor, debug traffic across multiple planes.

Switch trends driven by need for higher density

	2014	2016	2018	2020	2022	2024
Bandwidth	3.2T	6.4T	12.8T	25.6T	51.2T	102.4T
Serdes	25Gbps	25Gbps	50Gbps	50/100Gbps	100Gbps	100/200Gbps
Port count	128	128	128	256	512	512

8x serdes speed

Higher bandwidth
32x in 10 years

4x more ports

AI Networking Trends: direction Ethernet

Scale-up
1-2us RTT

- Higher bandwidth / XPU
- Proprietary today (NVLink, InfinityFabric, some Ethernet).
- Moving to Ethernet.

Scale-out across racks
10us RTT

- Initial deployments used Infiniband / HPC interconnects.
- All large-scale deployments now on Ethernet.

Scale-out across regions
100-1000us RTT

- Cross-region training.
- Large buffers needed.
- All on Ethernet.

Tomahawk Ultra
51.2 Tbps

Tomahawk 6
102.4 Tbps

Jericho4
51.2T Scalable router

Implications for transport protocols

Buffer memory not scaling with chip bandwidth

- Buffers are SRAM.
- More buffers increase chip area and power consumption.
- Congestion control is harder with less buffers to absorb bursts.

Improved support for congestion control and visibility:

- Packet trimming (UET 1.0 standard)
- Congestion Signaling (UET, in progress)
- Vendor specific extensions.

Broadcom-specific extensions

Tomahawk 6

- Back-to-source trimming.
- Fast link failover.
- In-band flow analysis (IFA) – e.g. support for HPCC.

Tomahawk Ultra

- 250ns switching latency.
- Line-rate 64B packets.
- Programmable visibility framework.
- Support for in-network collectives.
- Optimized header.

Switch trends for AI

- AI networks moving to/on Ethernet for scale-up, scale-out and cross region.
- Key trends driven by economics:
  - Chip bandwidth doubling every two years.
  - Fewer network tiers, higher radix.
  - Buffer memory not scaling with speeds.
- Trimming and congestion signaling to support transport protocols.
- Specialized chips address different parts of AI networks.

Fault tolerance

- Consider a hypothetical 100K GPU datacenter.
  - One 800Gb/s scale-out NIC per GPU
  - 8 x 100Gb/s ports per NIC
  - 512-port NICs, so two-tier FatTree
  - Full bisection b/w
- We need:
  - 800K NIC-T0 links
  - ~3000 T0 switches in 8 planes
  - 800K T0-T1 links
  - T1 switches depends on number of rails, but >1500

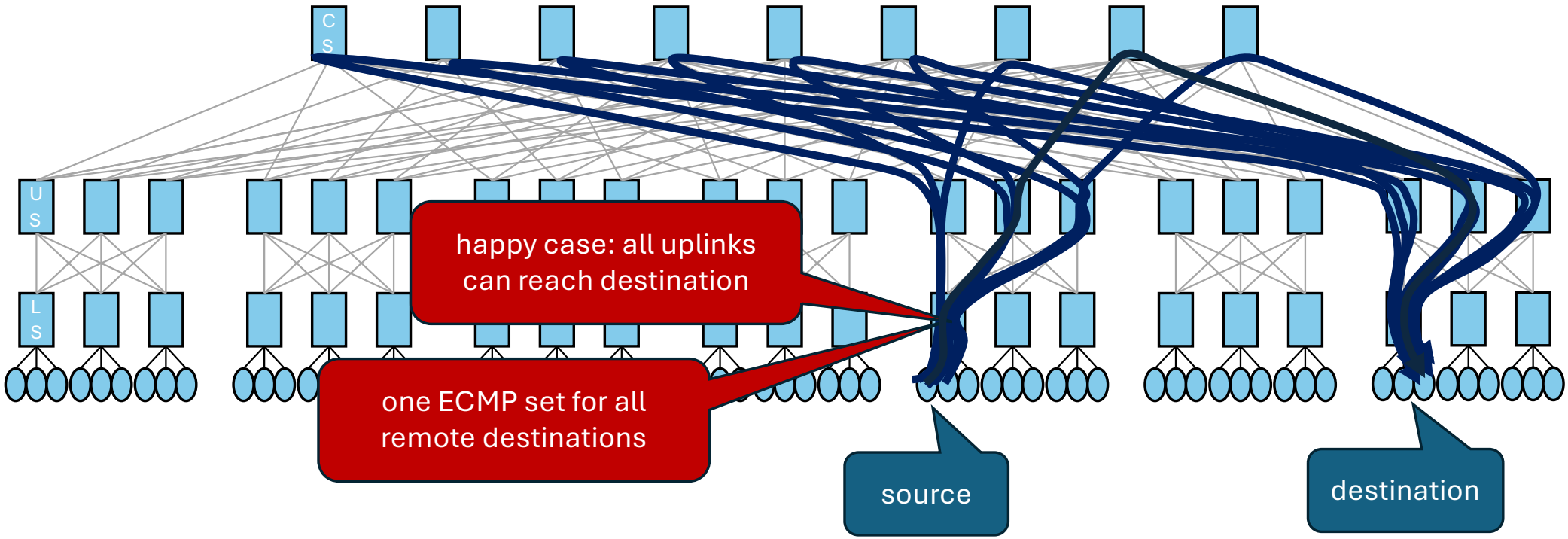
- With 1.6M links and 5000 switches, expect lots of failures.

Routing

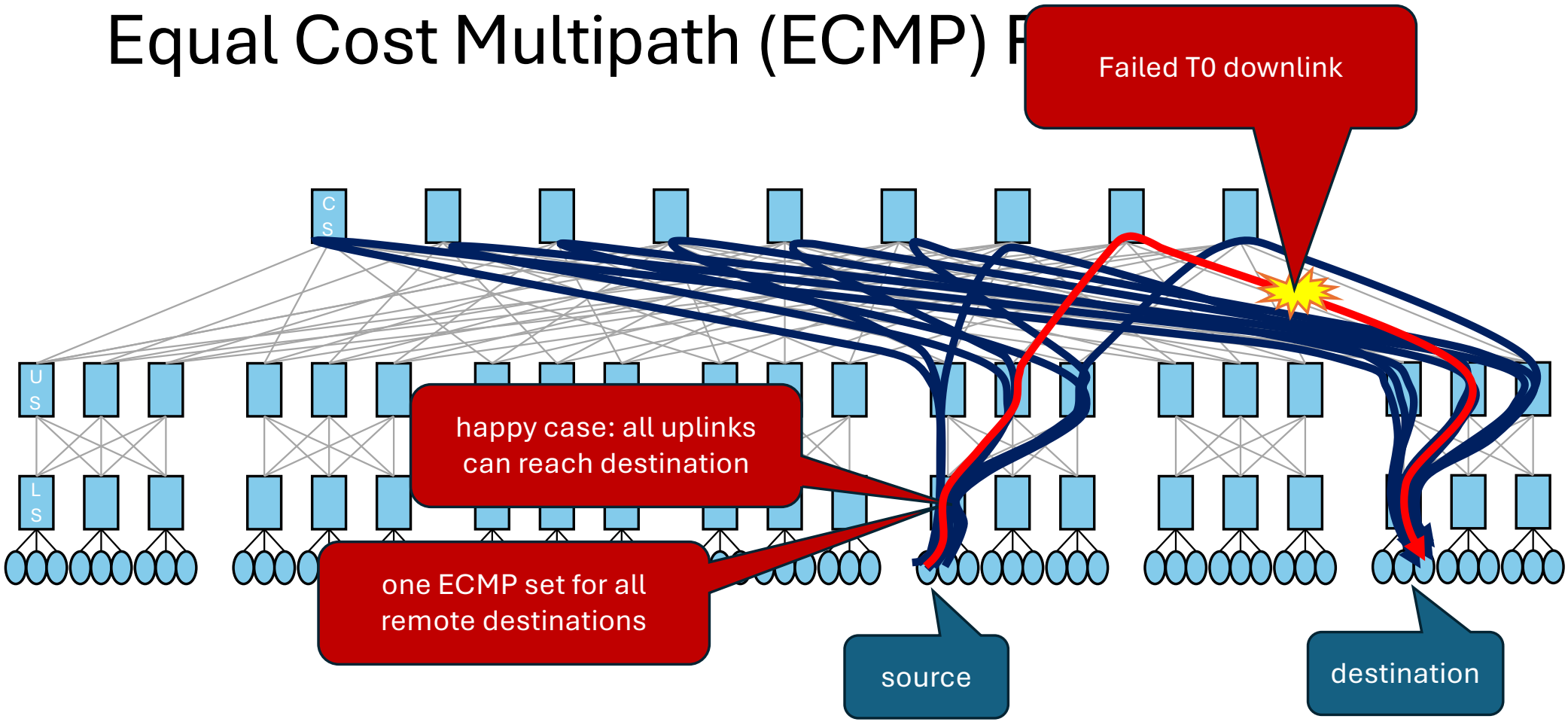
- BGP has traditionally been the tool of choice for routing in back-end networks
  - Stateful, scales well to very large numbers of routes.
  - Supports ECMP

- What happens with failures?

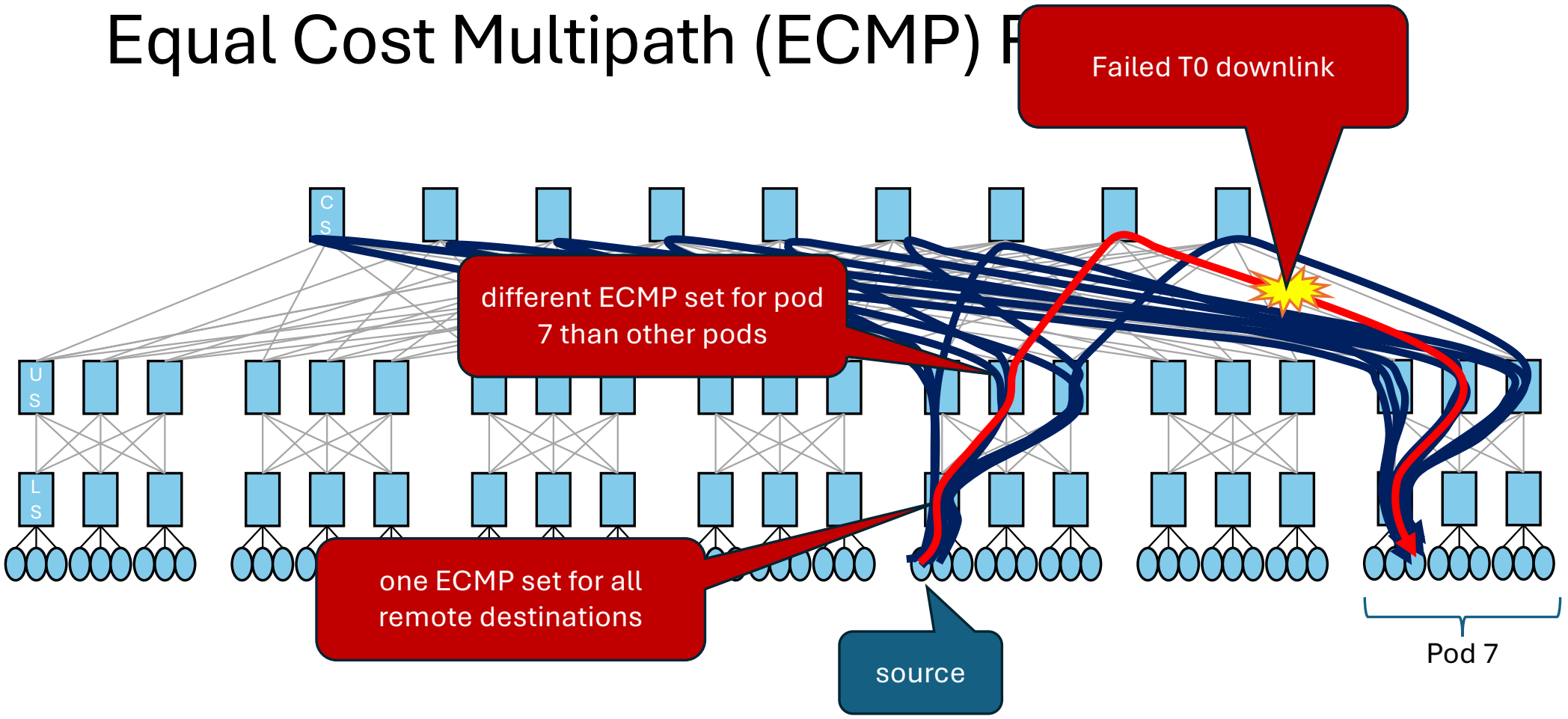
Equal Cost Multipath (ECMP) Routing



Equal Cost Multipath (ECMP) F



Equal Cost Multipath (ECMP) F

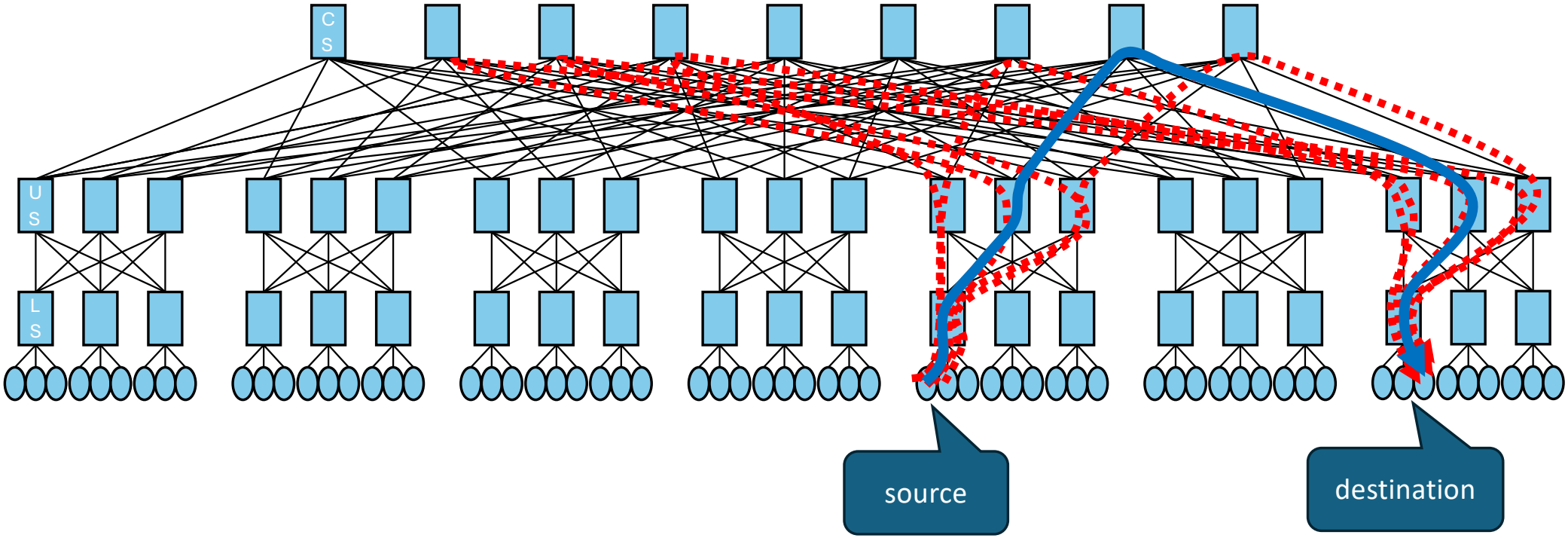


Routing

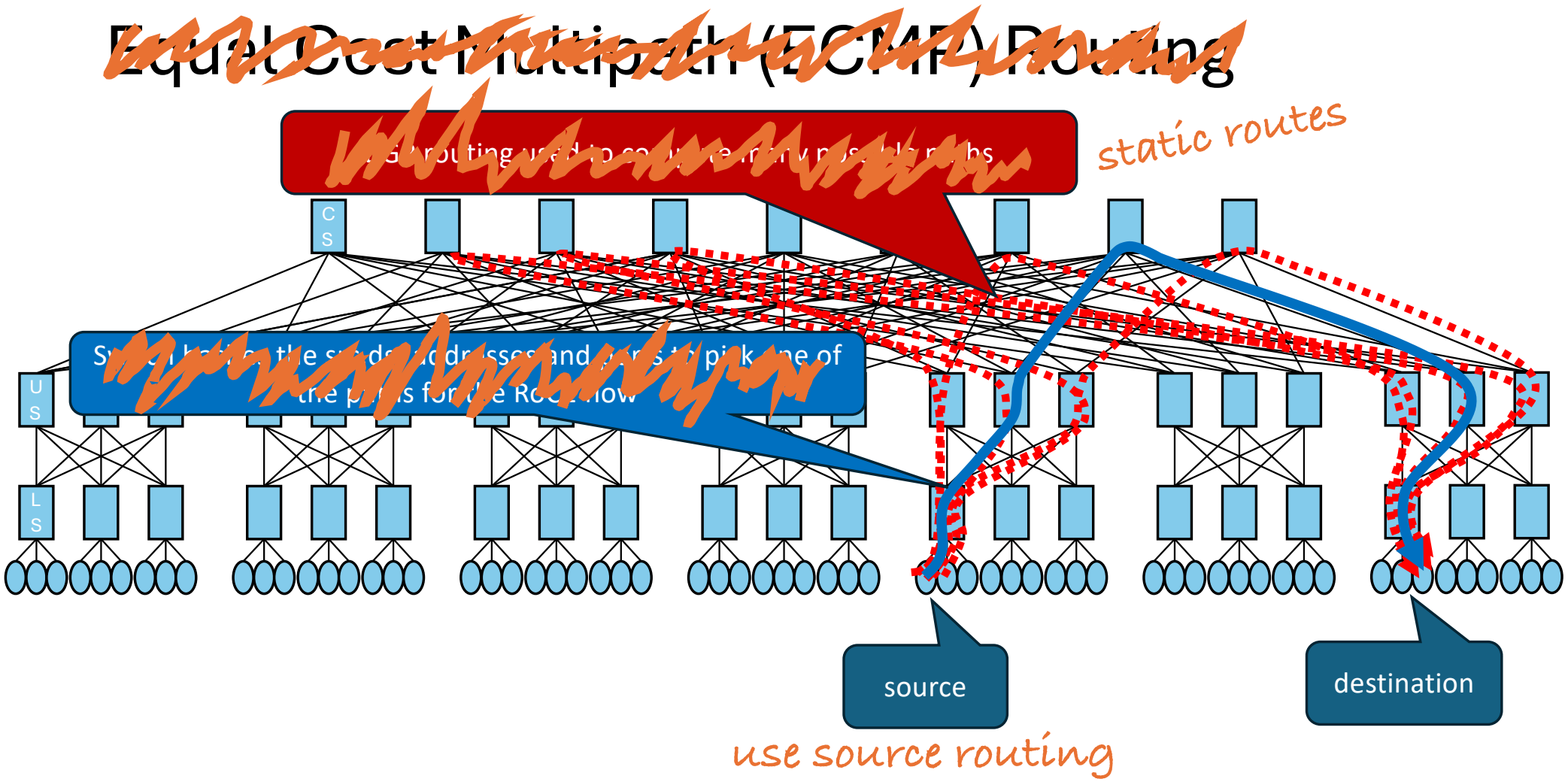
- BGP has traditionally been the tool of choice for routing in back-end networks
  - Stateful, scales well to very large numbers of routes.
  - Supports ECMP

- What happens with failures?
  - Lots of different routes, each with a different ECMP group.
  - State scales O(destinations * failures)
  - With lots of failures, even BGP may struggle.

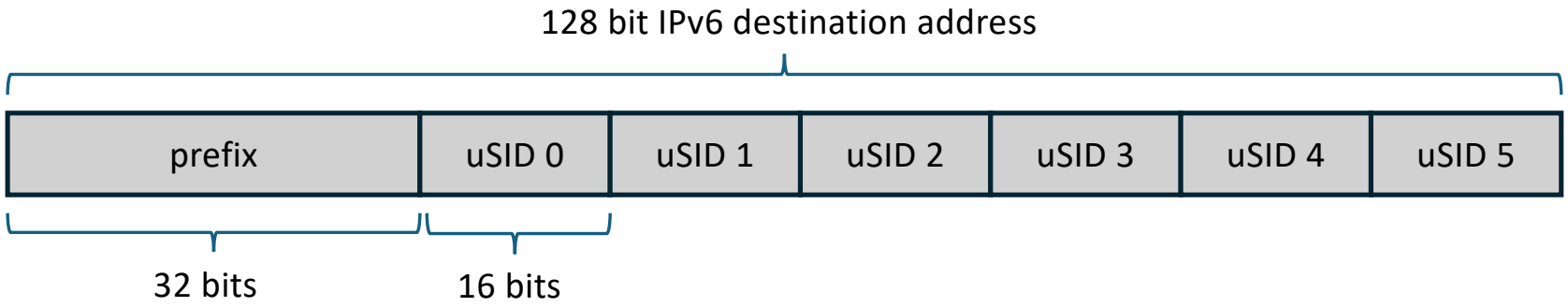
Equal Cost Multipath (ECMP) Routing



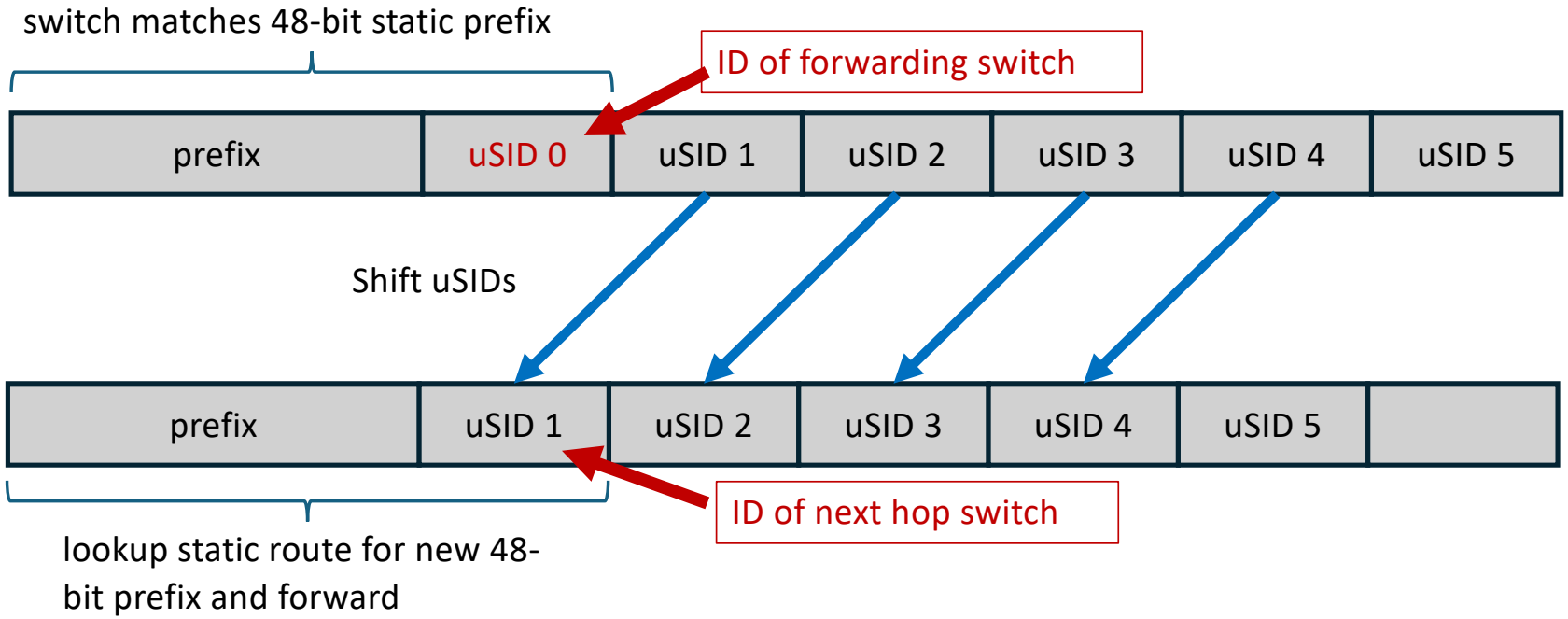
Equal Cost Multipath (ECMP) Routing



IPv6 Segment Routing (SRv6)



IPv6 Segment Routing (SRv6)



Static source routing

- Needs a transport that sprays across many paths
  - Enough paths that losing packets on one causes only a transient glitch
  - One explicit SRv6 route per path
- Transport detects failure
  - Dropped packets, no trim-nack.
  - Stops using failed path
- No dynamic routing
  - Nothing moving under you
  - Once transport stops using a path, it stays out of use.

Why is AI networking hard?

- Huge scale
- Synchronized computation => synchronized network traffic
- Massively parallel networks
- High speed links (too fast for normal software stacks)

AI networks are co-evolving:

- GPU design
- Application parallelism
- Collective algorithms
- Network hardware
- Transport protocols
- Network topology

Big Open Issues

- Scale-up vs scale-out
  - Most demanding traffic moving to scale-up, as it gets big enough
  - Can we unify scale-up and scale-out?
- Scale-up transport design
  - Multi-plane, but single tier?
  - 10x high speed compared to scale-out.
  - Need 10x less power per bit!
- Scale-out transport protocols
  - Currently, it's the wild west.
  - Many many solutions from many companies.
  - Have we reached the point where industry stabilizes for a while on a small number of good enough solutions?
    - Is UET a good baseline for future industry-wide innovation? Or Falcon? Or something else?
- The role of optical switching
  - Circuit switches currently used to reconfigure between tasks
    - Avoid latency + power of another tier of switches
  - Can silicon photonic switching replace electronic switching?