Challenges in Networking for AIML clusters

Costin Raiciu

BROADCOM®

Politehnica of Bucharest

Reuters

World | US Election | Business | Markets | Sustainability | Lega

Technology | Data Privacy

Microsoft, OpenAI plan $100 billion data-center project, media report says

By

Mar

THE CHANNEL CO.
CRN

News | Video | Companies | Awards & Lists | Eve

HOME ▸ NEWS ▸ DATA CENTER ▸ MICROSOFT AND OPENAI PLAN $100B AI-CENTRIC DATA CENTER: REI

Cloud ▸

Pro | Org Charts | Tech | Finance | Weekend | Community | Mor

Exclusive: Two OpenAI Business Partners Each Discuss $2 Billion Valuation

Save 25% and tune in

Subscribe to read the full article

Microsoft and OpenAI Plot $100 Billion Stargate AI Supercomputer

By Anissa Gardizy and Amir Efrati

Oper

How does infrastructure scale*

$814M

Frontier supercomputer

GPUs: **38k** AMD MI250X

Power consumption: **30** MW

$3.5B

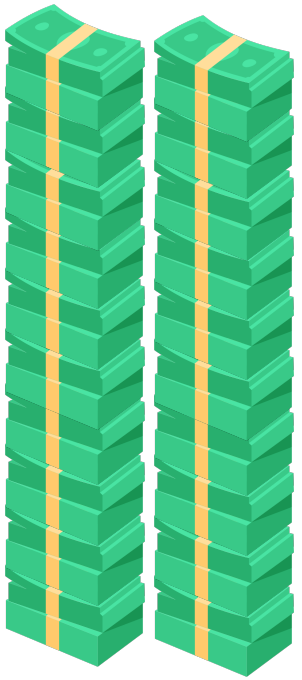
xAI's Colossus

GPUs: **100k+** Nvidia H100

Power consumption: **100** MW

Meta AI training cluster

$100B



OpenAI + Microsoft

GPUs: ***1.67M*** Nvidia B200 ?

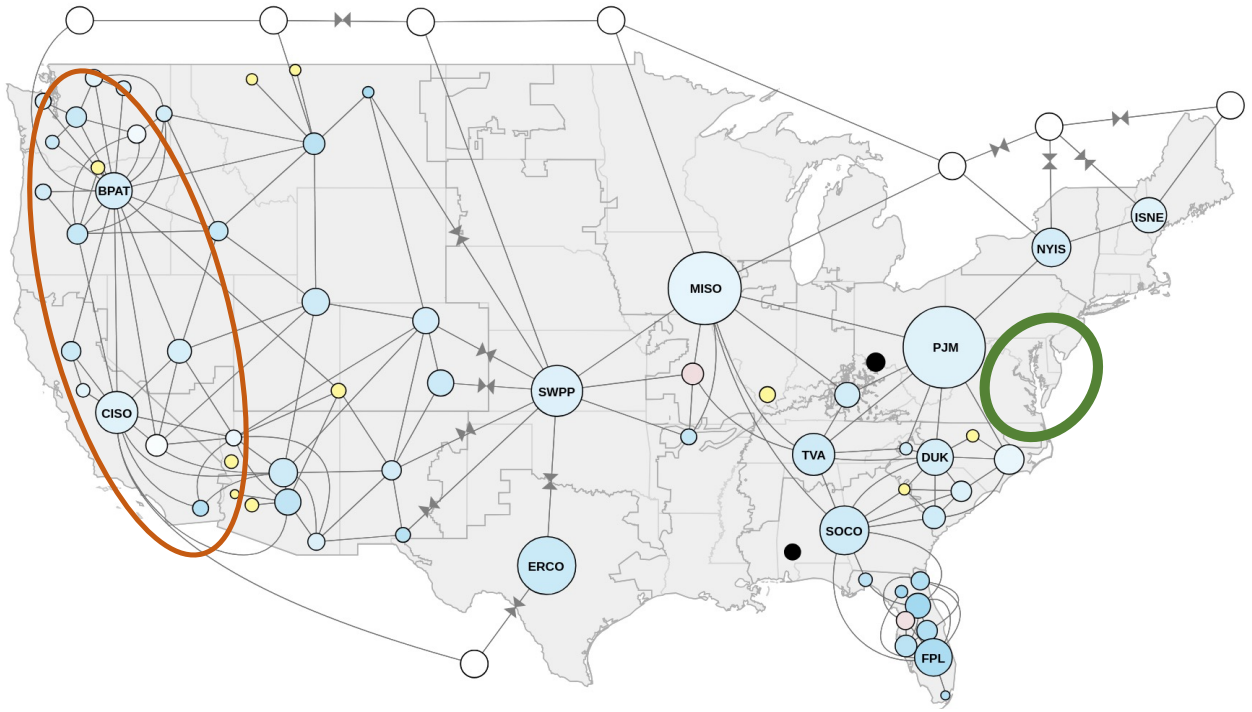
Power consumption: ***5 GW***

* estimates of cost and power consumption from public data

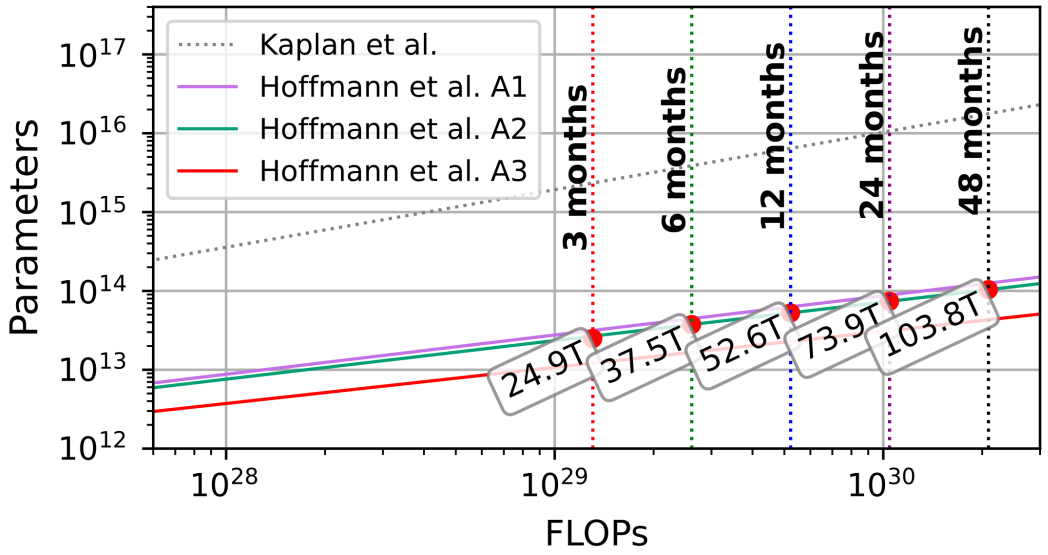
Where (in the US) can we place a ***5 GW*** AI/ML datacenter?

There's no single spot in the US with this much readily available power

The datacenter must be geographically split into multiple units



What Transformer model can we train with ***1.67M*** GPUs?



ChatGPT: reportedly close to **2T parameters**

100 trillion parameter model:

- 134 layers
- minimum 18 GPUs to hold 1 layer!

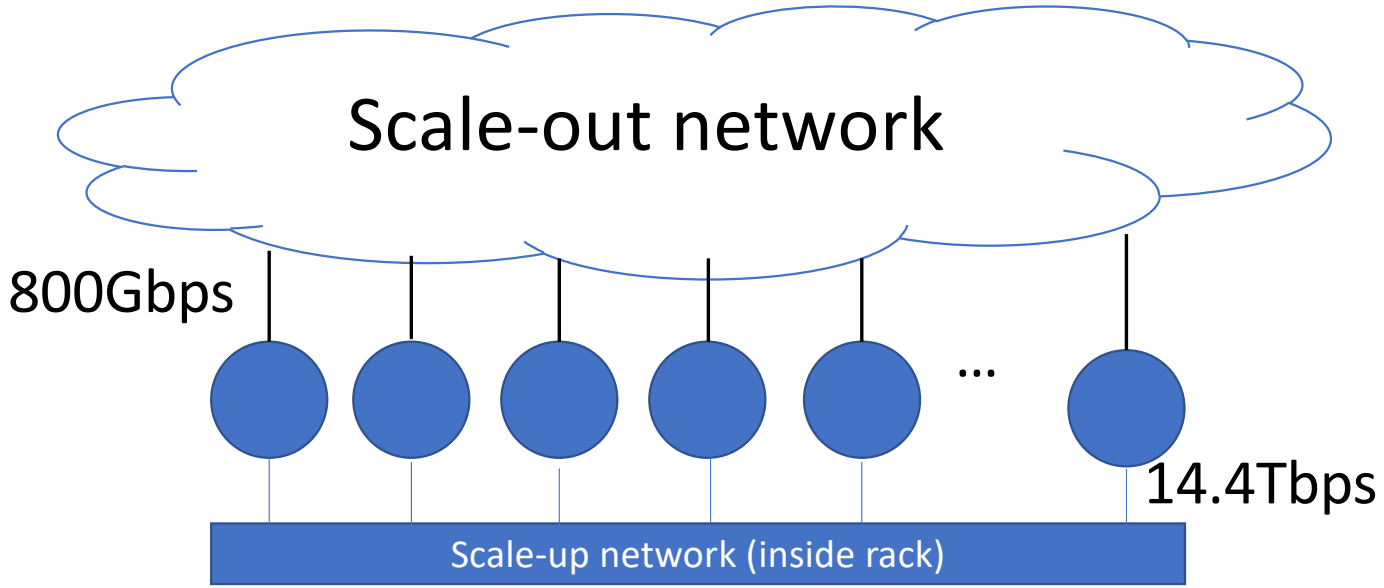
How do we train this model?

"Scaling Laws for Neural Language Models" - Kaplan et al. (OpenAI - Jan. 2020)

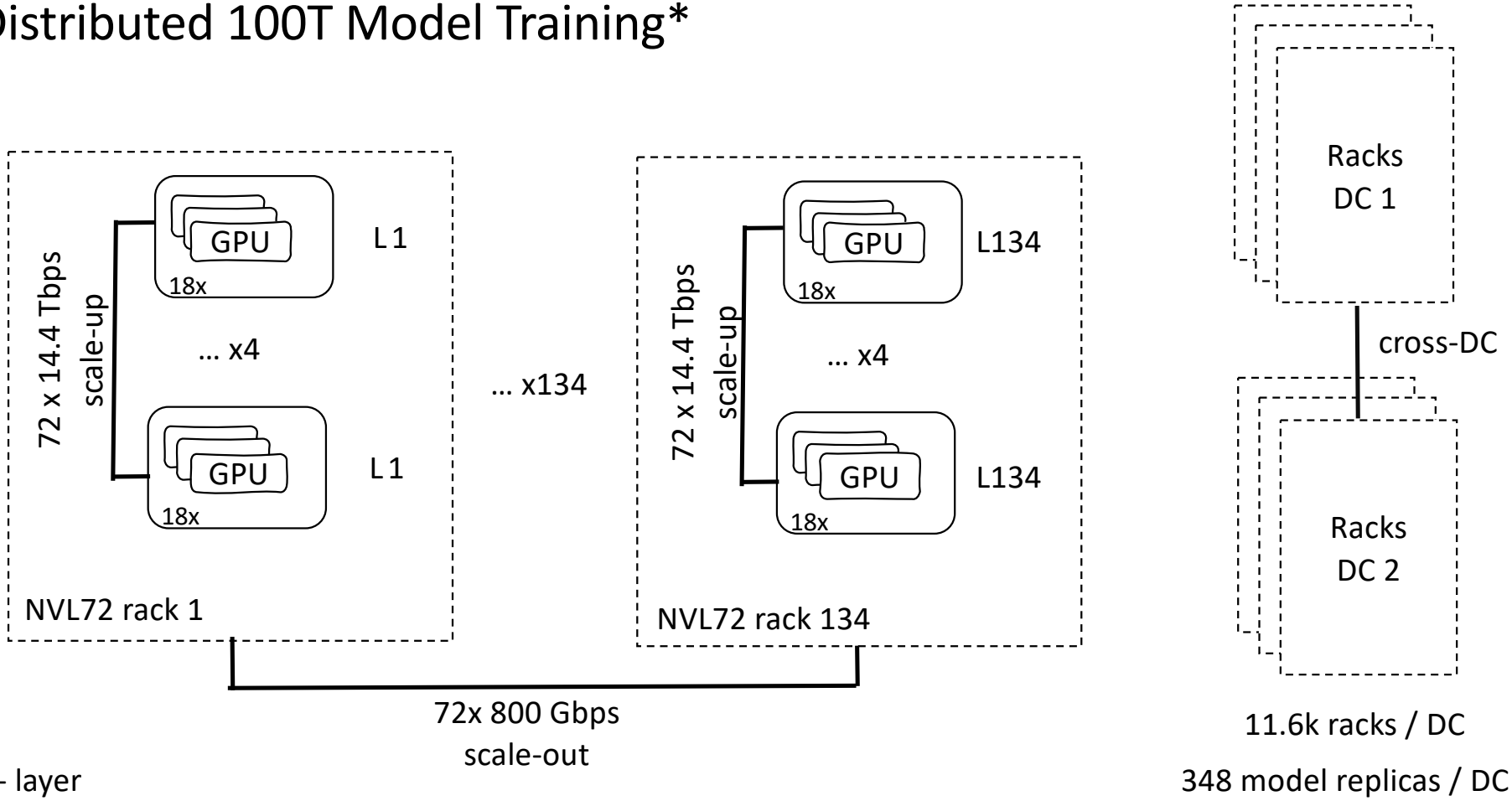
"Training Compute-Optimal Large Language Models" - Hoffmann et al. (Google DeepMind - Mar. 2022)

Networking for AI/ML: scale-up and scale-out

- Scale-up (e.g. NVLink) works inside an accelerator / rack
- Scale-out: Ethernet (or Infiniband) between racks



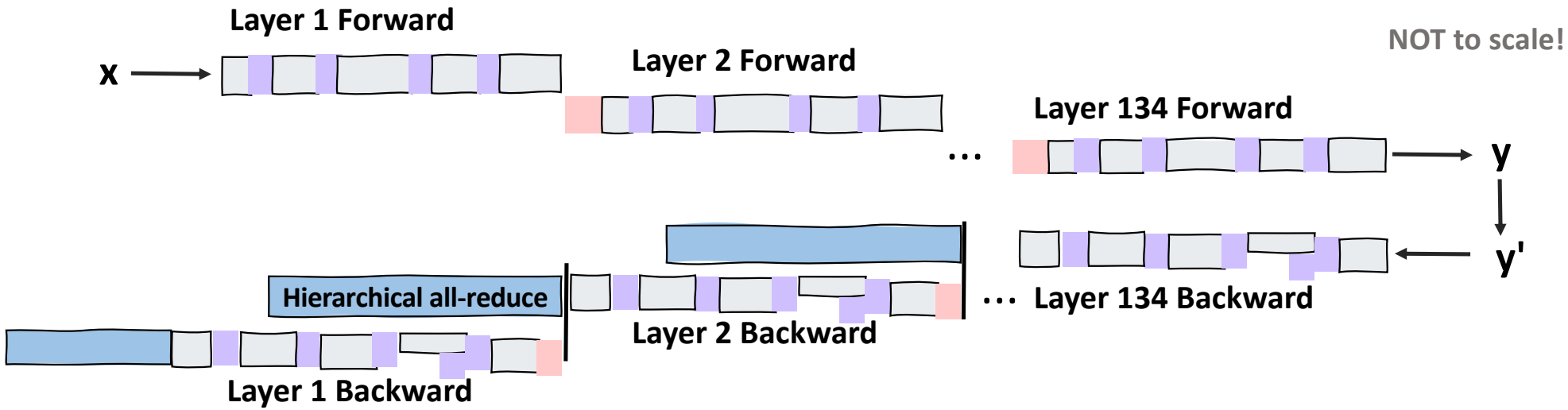
Distributed 100T Model Training*



L - layer

*State-of-the-art: 3D parallelism ["*Reducing Activation Recomputation in Large Transformer Models*" - Korthikanti et al. ('22)]

Computation / communication times for training (for one model pass)



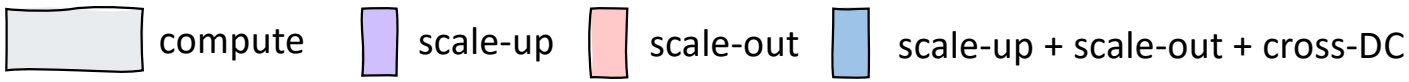
Per iteration per layer estimates:

- Computation: **400 ms**
- Communication: **150 ms** (**20 ms exposed**)

Exposed Transformer communication: **5%**

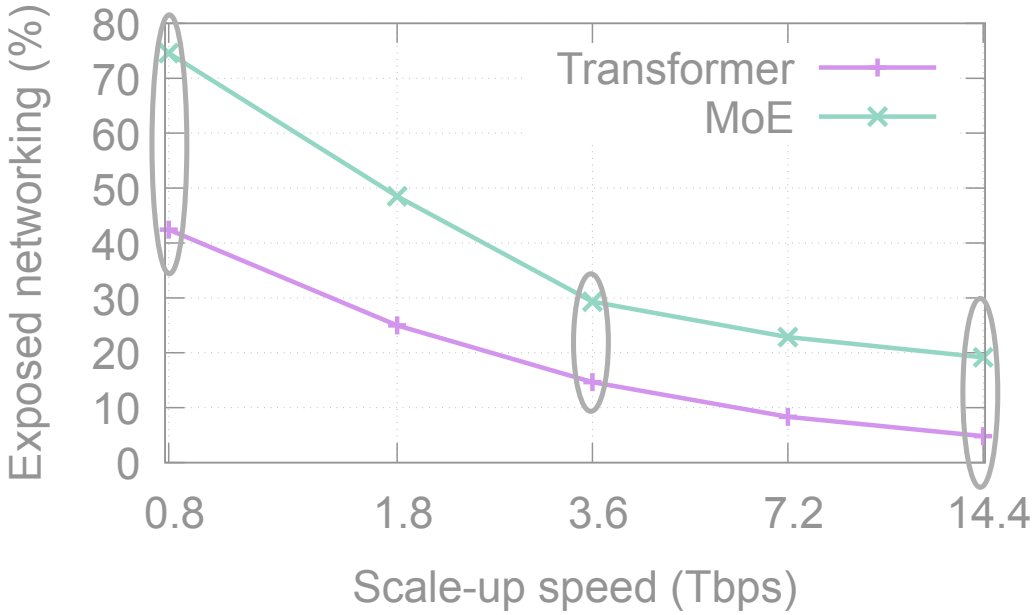
Exposed Mixture of Experts communication: **19%**

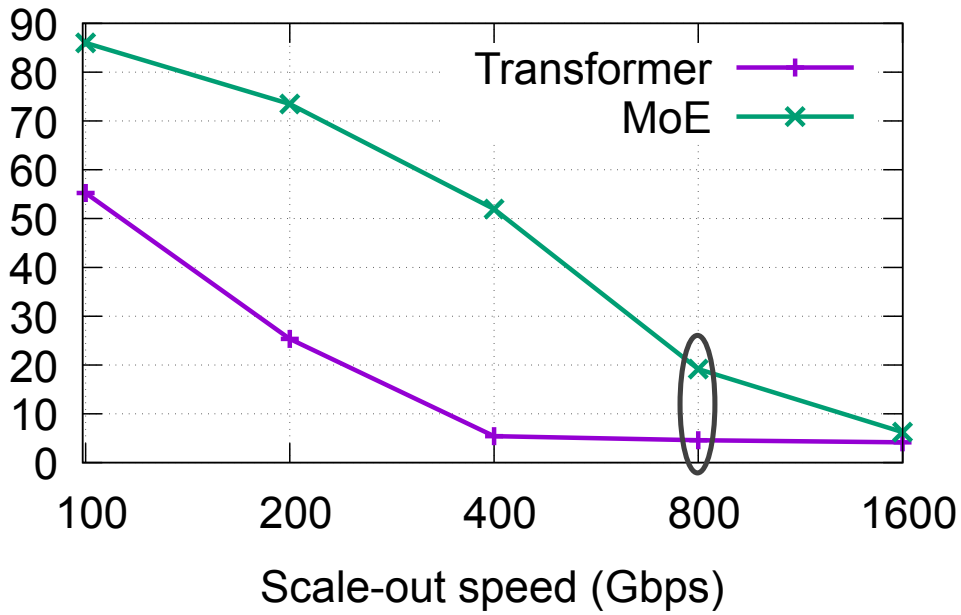
Takeaway: We need to keep communication times in check !



How should we dimension the network for AI/ML?

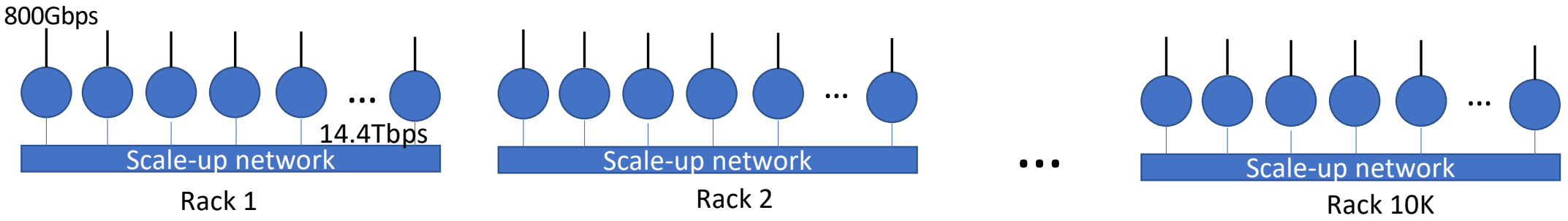
Minimize exposed networking : time per iteration spent while waiting for network



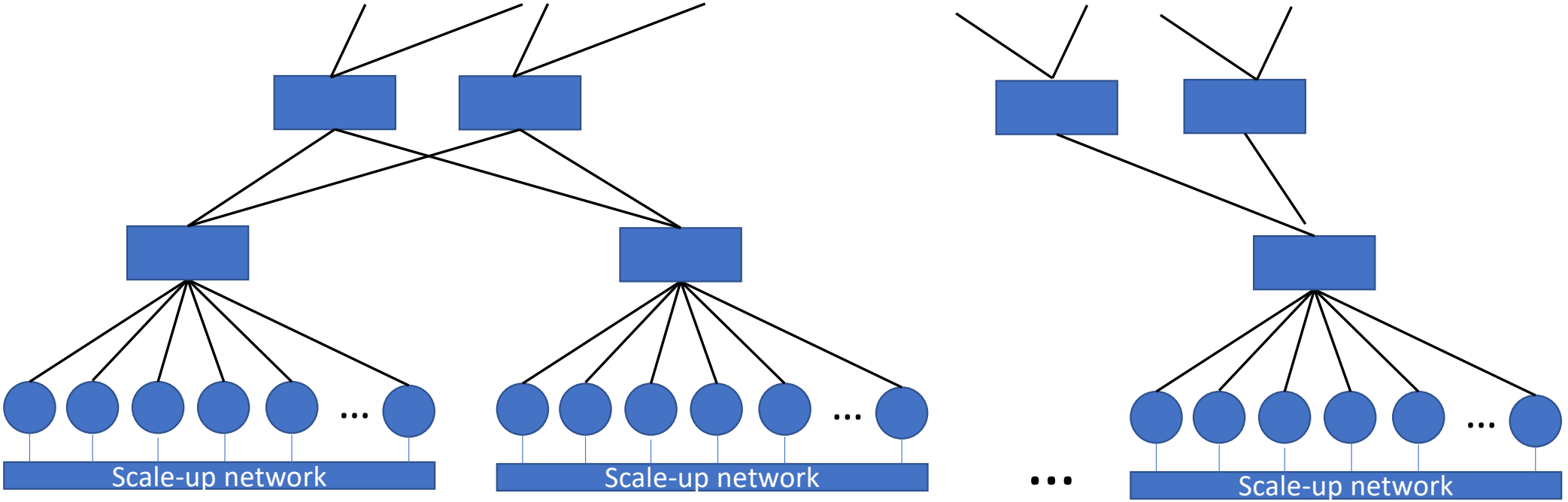


Takeaway: scale-up is crucial for distributed model training.
Scale-up domains growing (e.g. from 8 in H100 to 72 in NVL72).

How should we build the scale-out network?



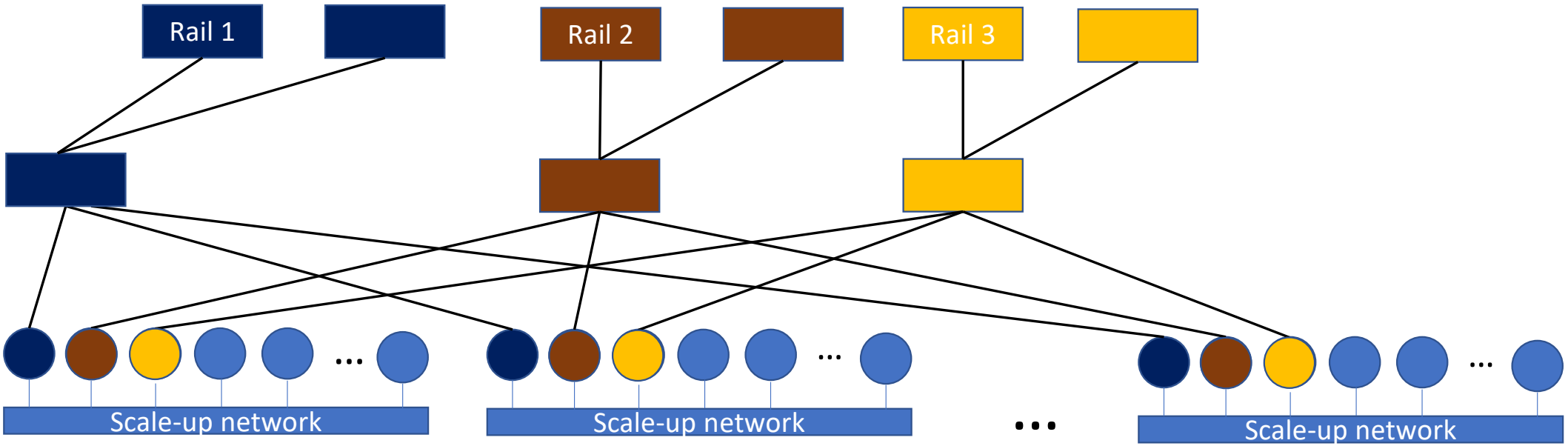
Default option: traditional multi-tier Clos



Max number of nodes ~ switch radix to #tiers

- 51.2T = 64 x 800Gbps ports => 4 tiers for 800K GPUs
- 75K switches, 4 cables / GPU (three or more optical).

Multi-rail: independent networks for subset of GPUs



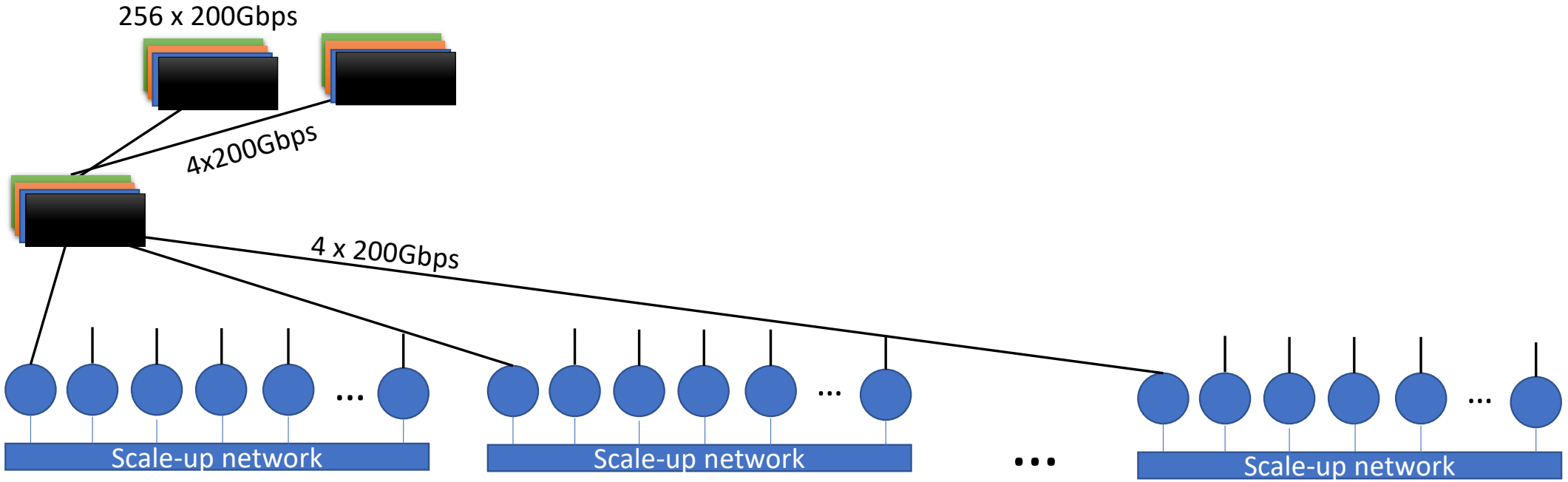
72 rails, ~8K GPUs per rail

- 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

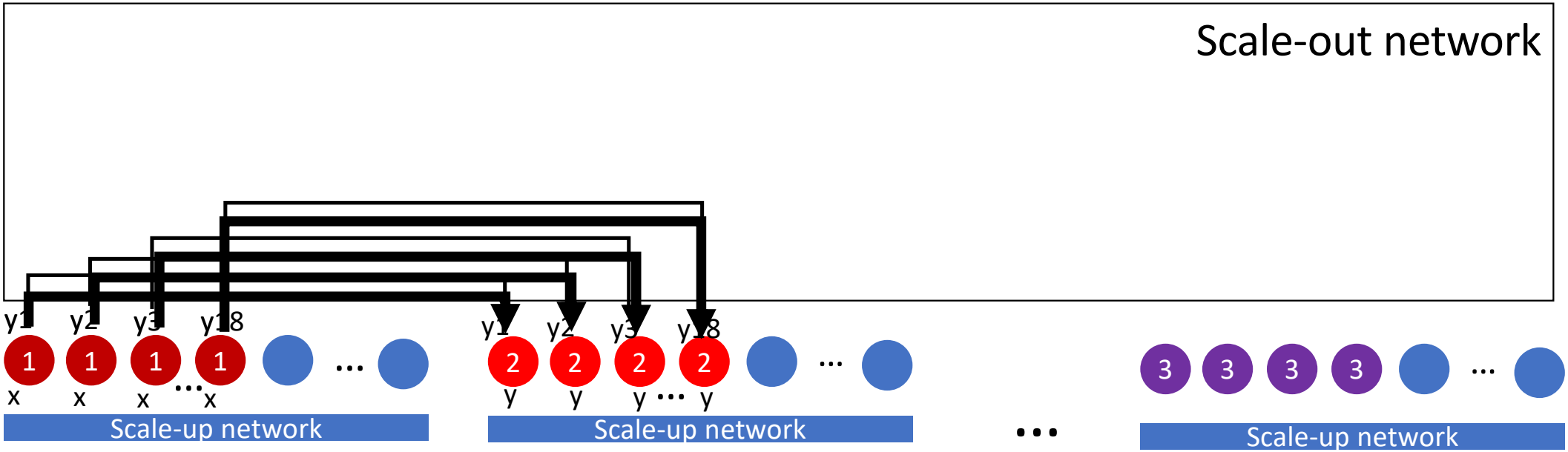


51.2T = 64x800Gbps or 256x200Gbps!

Four planes, 200Gbps per plane - two tier network. 50% cheaper

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

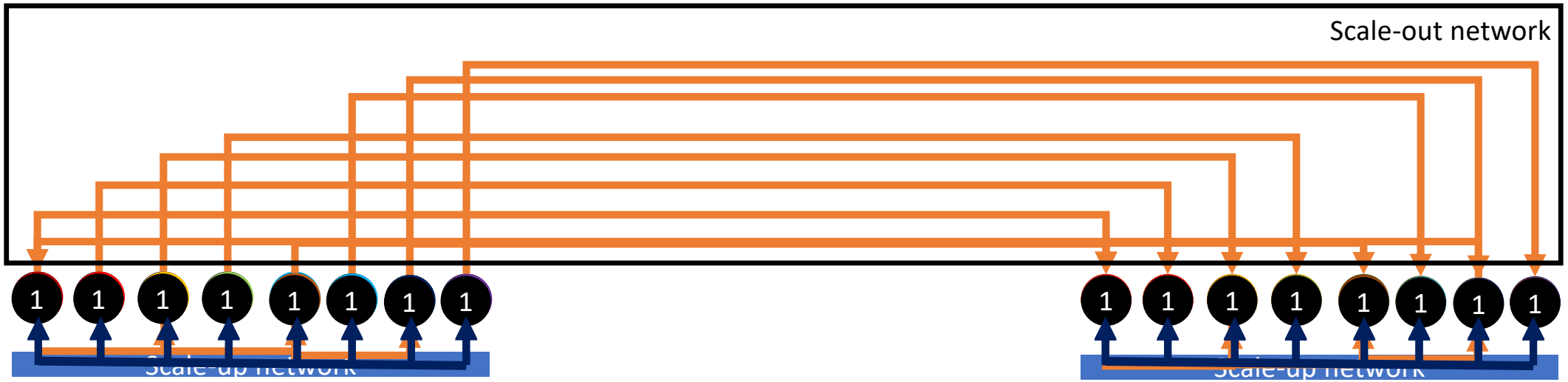
How do AIML workloads use the network?



100T model prediction:

- 18 B200 GPUs needed to hold 1 layer.
- size (x) = size (y) = 3.86GB
- Exposed networking: 38ms with 130ms fw / 260ms backward computation.
- Shard data to use scale-out efficiently: 215MB per GPU + all-gather at destination: 2.15ms only!

How do AI/ML workloads use the network?

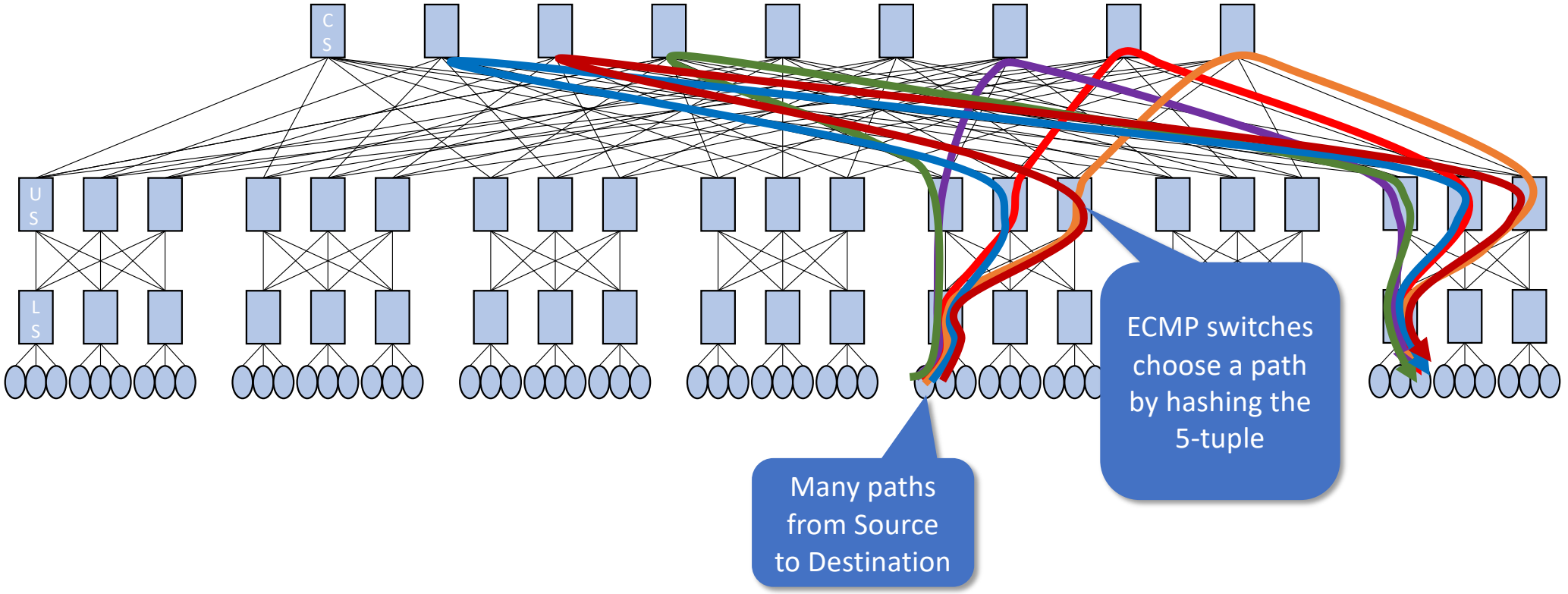


Takeaways: Scale-out collectives also use scale-up, have poor locality!

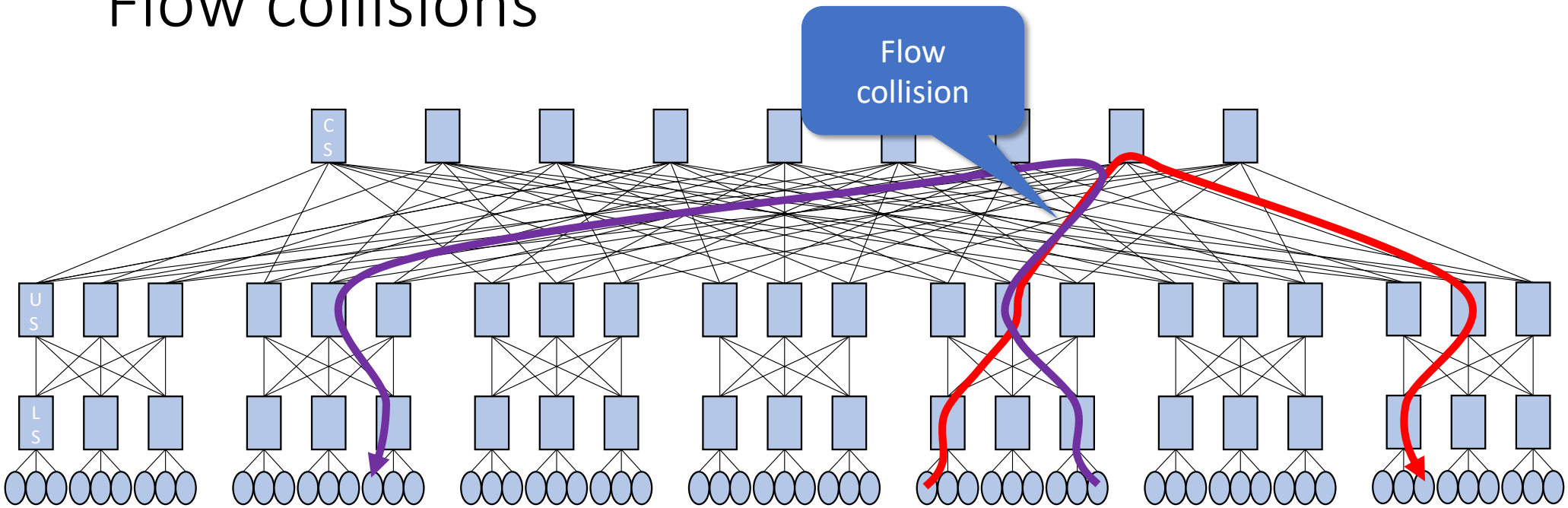
Pipelining used to reduce latency =>

- scale-out flows will be small.
- coupling between scale-up and scale-out flows.

Flow routing in a Clos topology

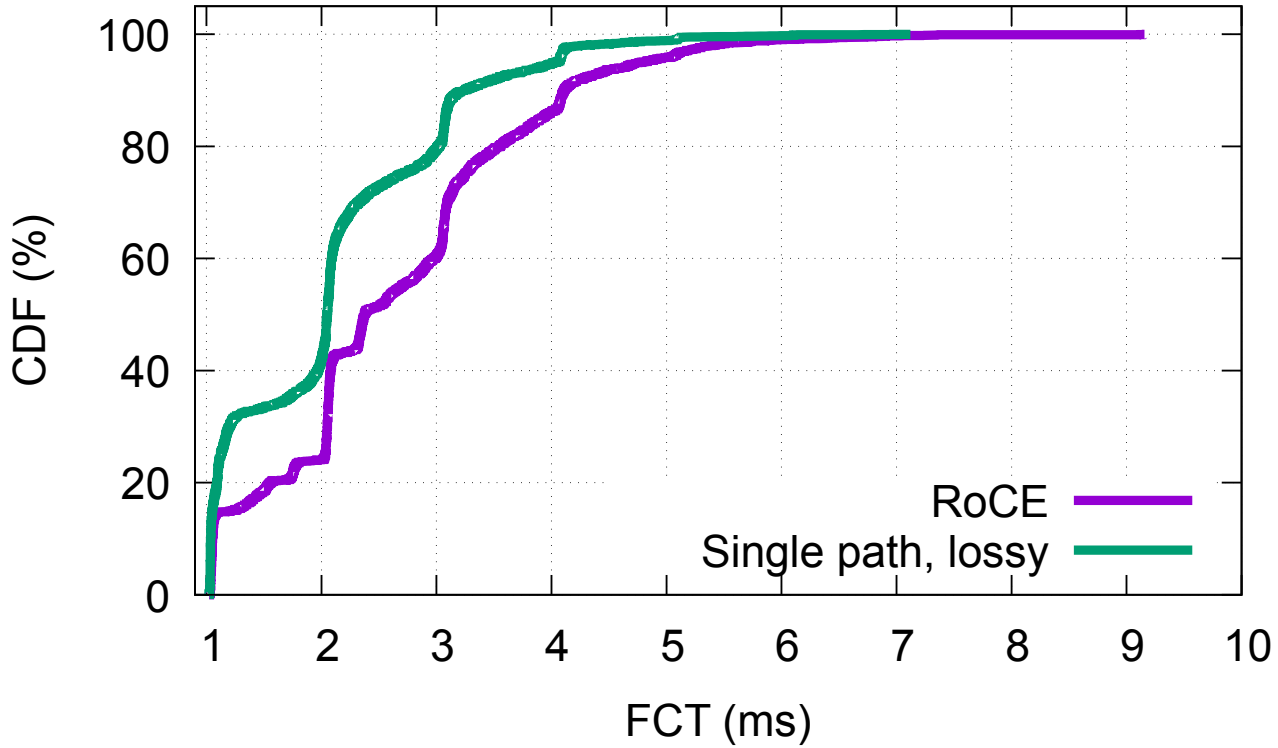


Flow collisions



Are RoCE / Infiniband good-enough for AIML?

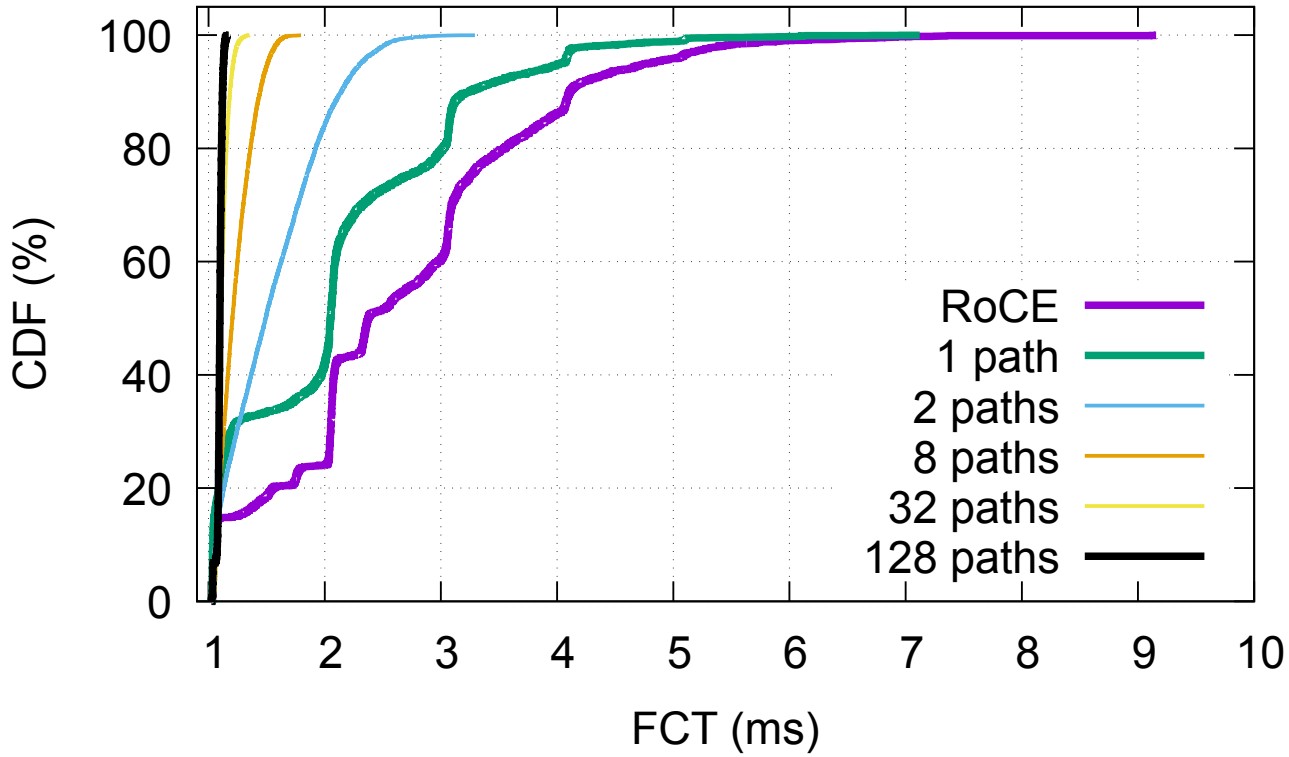
Simulated pipeline parallelism, 800Gbps links, 8K nodes leaf-spine, **100% load**



Flow collisions and head of line blocking in the main issue for RoCEv2

Ultra Ethernet Consortium standardizing a multipath transport protocol to replace RoCEv2.

Need multipath for pipeline parallelism



Takeaway: multipath is key to reduce tail FCT for ML training.

Flows start at line rate, but don't like PFC. Packet trimming being standardized.

Network Landscape Changing Dramatically

	CPU Cloud Networks		Large-Scale GPU Networks
Network	Single network	→ → →	Scale-Up / Scale-out
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

Transport protocol requirements for AIML networks

- Efficiently spread traffic across available paths.
- Short flows (1-10RTTs), some longer – need line-rate start!
- Switch buffers not scaling with linkspeeds.
  - Few BDPs / port available.

Multipath TCP (RFC6856)

- MPTCP opens multiple subflows between the source and destination.
  - ECMP hashes each subflow to a (probabilistically) different path.
- Multipath congestion control (e.g. MPTCP)
  - One window per path, each with its own ACK clock (cwnd_j), 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

Why not use MPTCP for AIML networks?

- Load balancing works well for long flows lasting hundreds of RTTs
  - Not so well for shorter flows.
- Need to use many paths.
- But minimum MPTCP window depends on #paths.
  - E.g. 256 paths means min 256 packet window.
  - (This equals BDP at 800Gbps).
  - Congestion collapse in incast.

Packet spraying transport design.

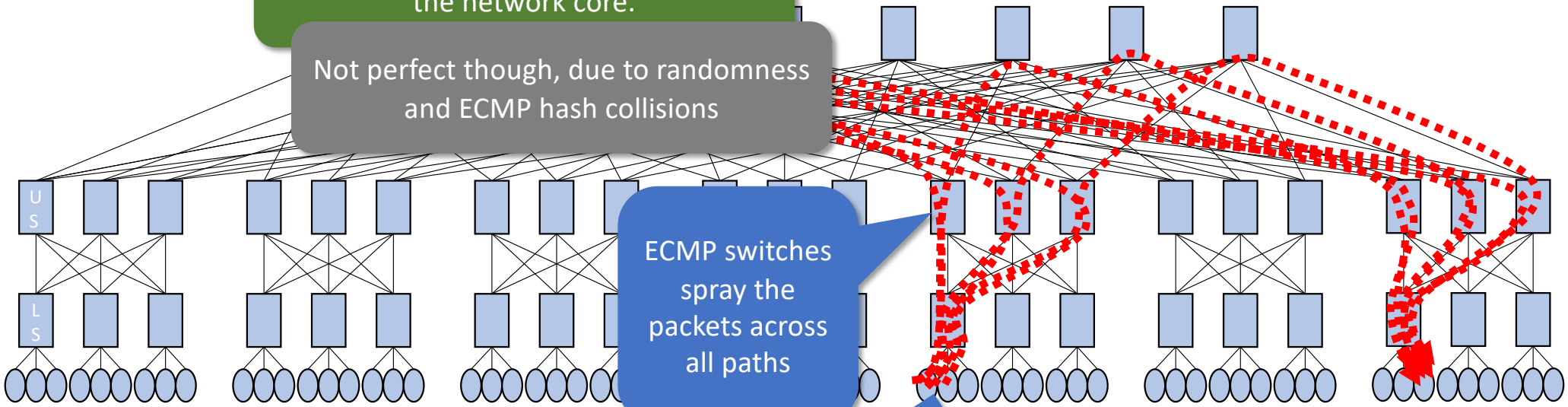
- Network-wide RTT and linkspeed – known to transport.
- Global window limits total amount of traffic.
  - Upper bounded to 1.5BDPs.
- Packet spraying: use multiple entropy values in packets.
  - Switches hash EV & 5 tuple = packets take probabilistically different paths.
- Load balancing to spread traffic across available paths.
  - How should we choose EV for outgoing packets?

Packet spraying

Good load balancing across all paths in the network core.

Reordering makes life harder for the transport protocol to detect loss

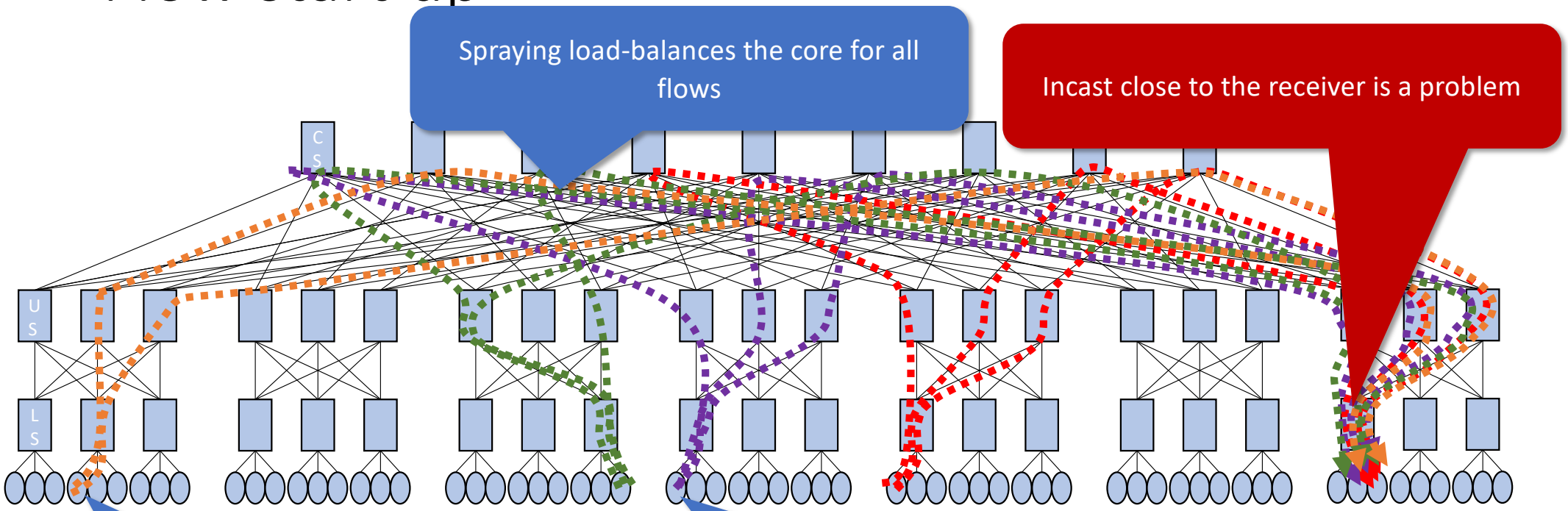
Not perfect though, due to randomness and ECMP hash collisions



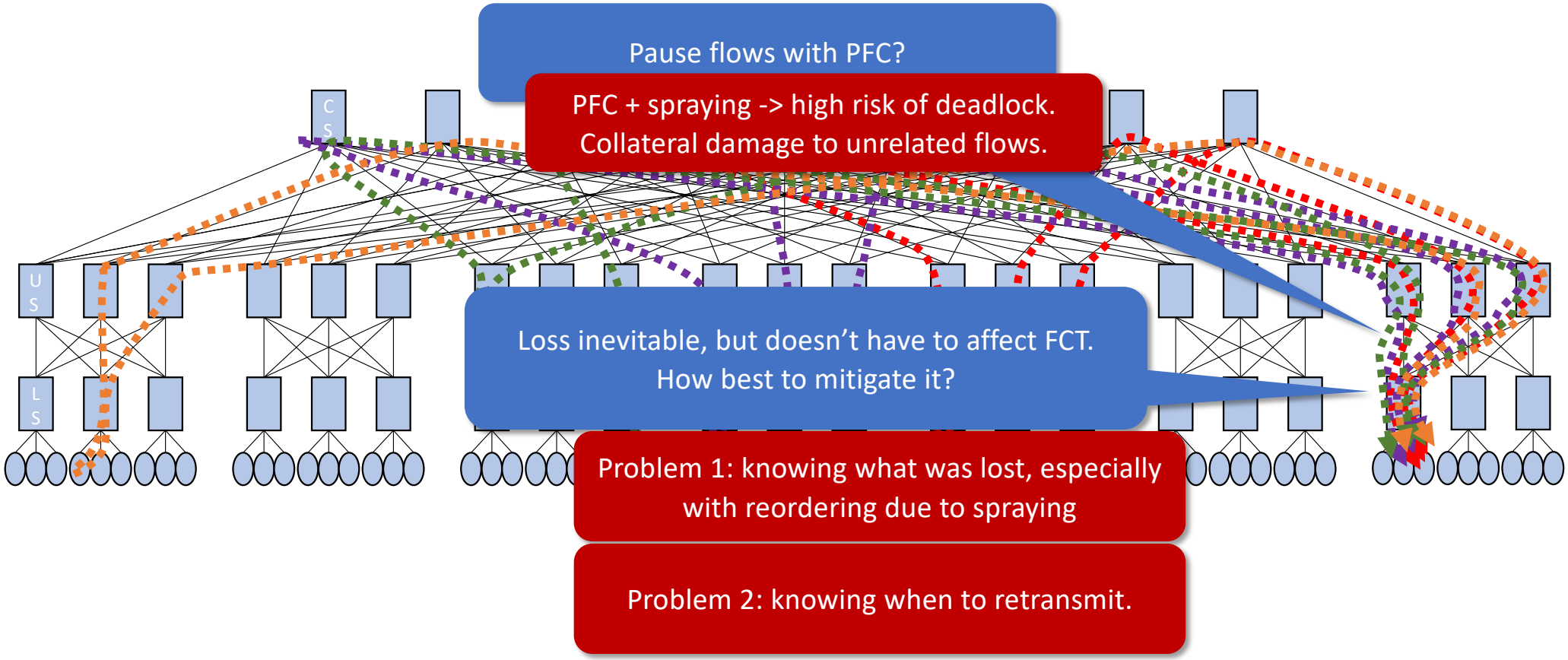
ECMP switches spray the packets across all paths

Change entropy value (e.g. UDP src port) with **every** packet sent

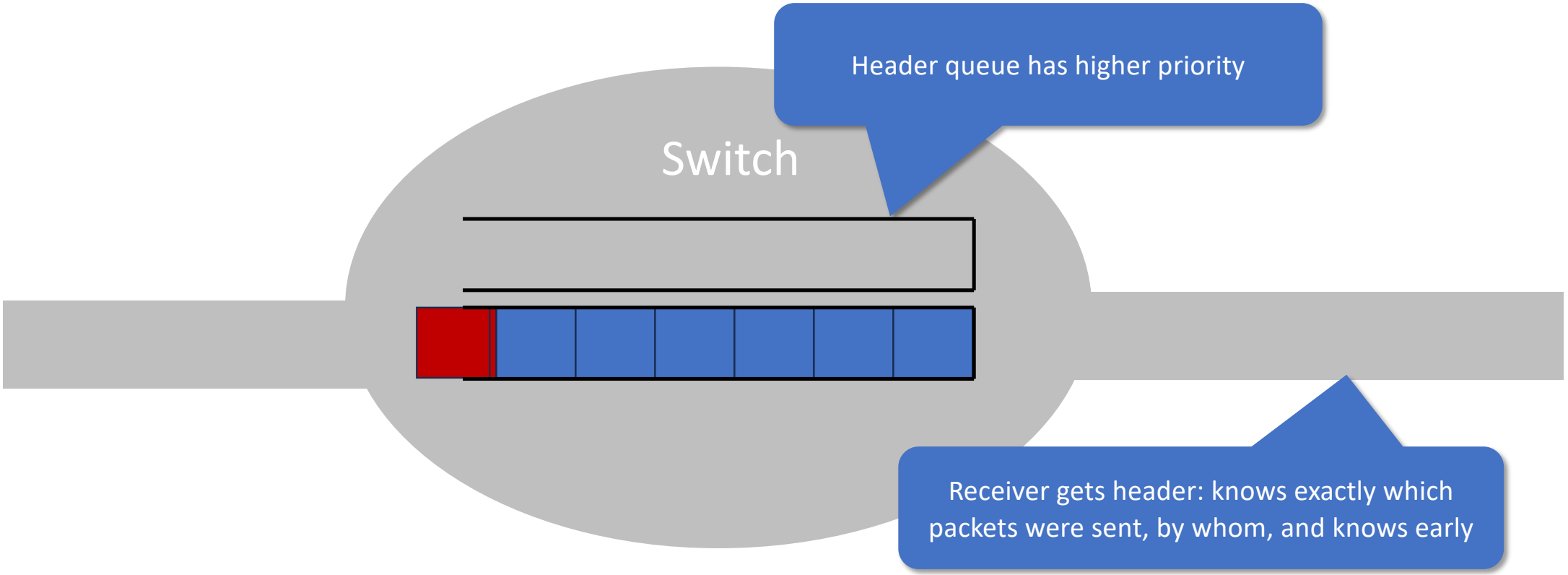
Flow start up



Flow start up



Packet trimming



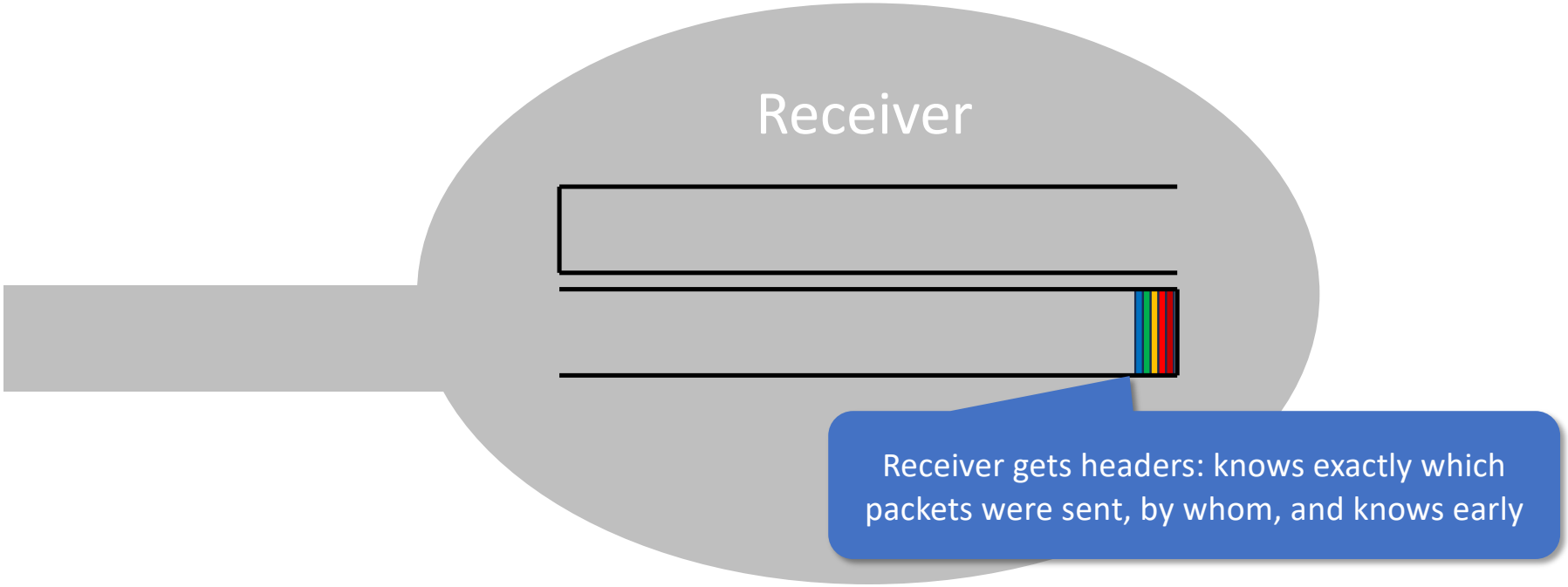
What about congestion control?

- Always start at line rate, after 1st RTT congestion control kicks in.
- Trims at receiver lead to NACKs which notify sender of lost packets.

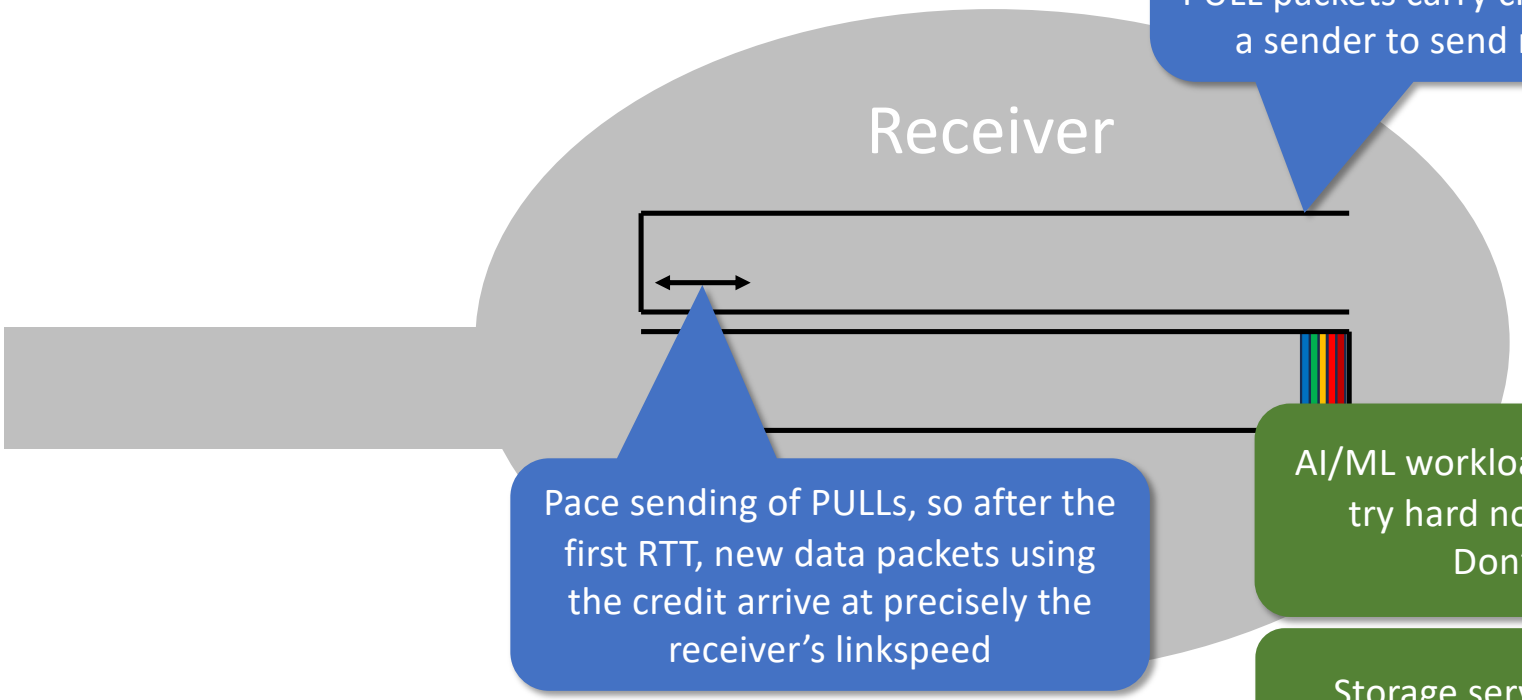
- Sender-driven congestion control (e.g. Smartt)
  - Targets sub-BDP standing queue at the bottleneck.
  - Use ECN and delay simultaneously.
    - Aggressive increase when queue ~ 0. Linear increase otherwise.
    - Multiplicative decrease when ECN & delay above threshold.
  - Average delay across all paths.
  - Handles both incast and oversubscription.

- Receiver-driven control – see next slide (e.g. EQDS).

Receiver credit pacing



Receiver credit pacing



For each unique sender, place a PULL packet in the pull queue.

PULL packets carry credit, allowing a sender to send more data.

Pace sending of PULLs, so after the first RTT, new data packets using the credit arrive at precisely the receiver's linkspeed

AI/ML workloads (and collective libraries) try hard not to create large incasts. Don't always succeed!

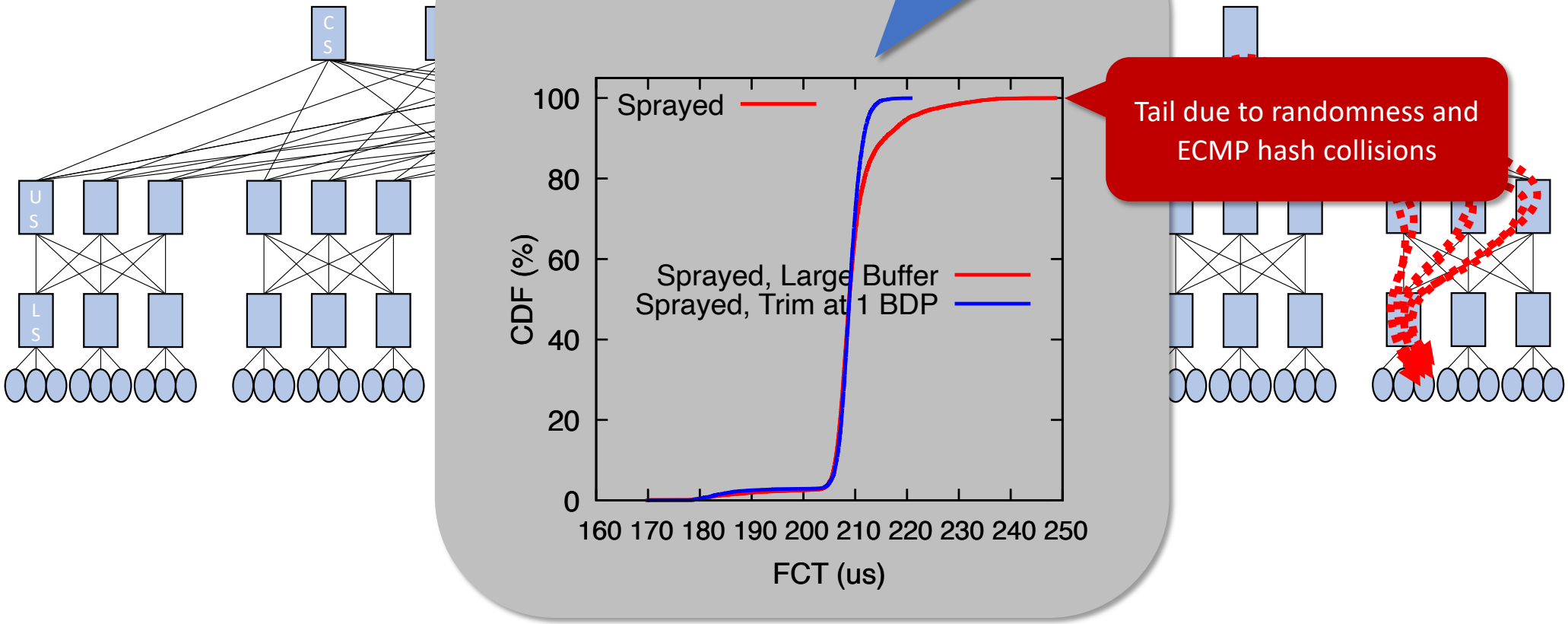
Storage servers can see large incasts. Receiver credit controls incasts really well.

What is the best way to spray packets?

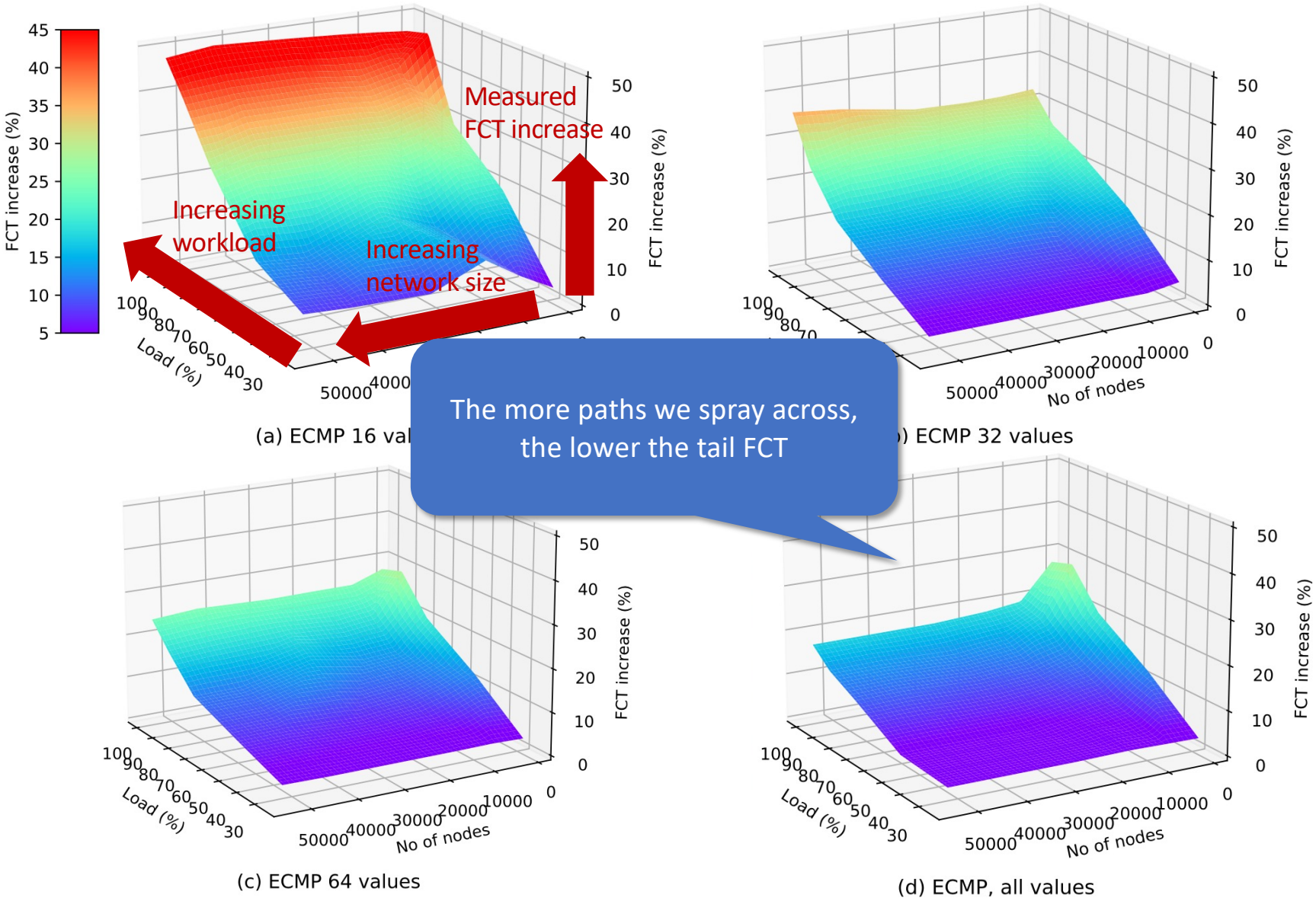
- **Simplest: oblivious load balancing**
- Pick a random EV for each packet.
- Works very well if network capacity is uniform.

Oblivious packe

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

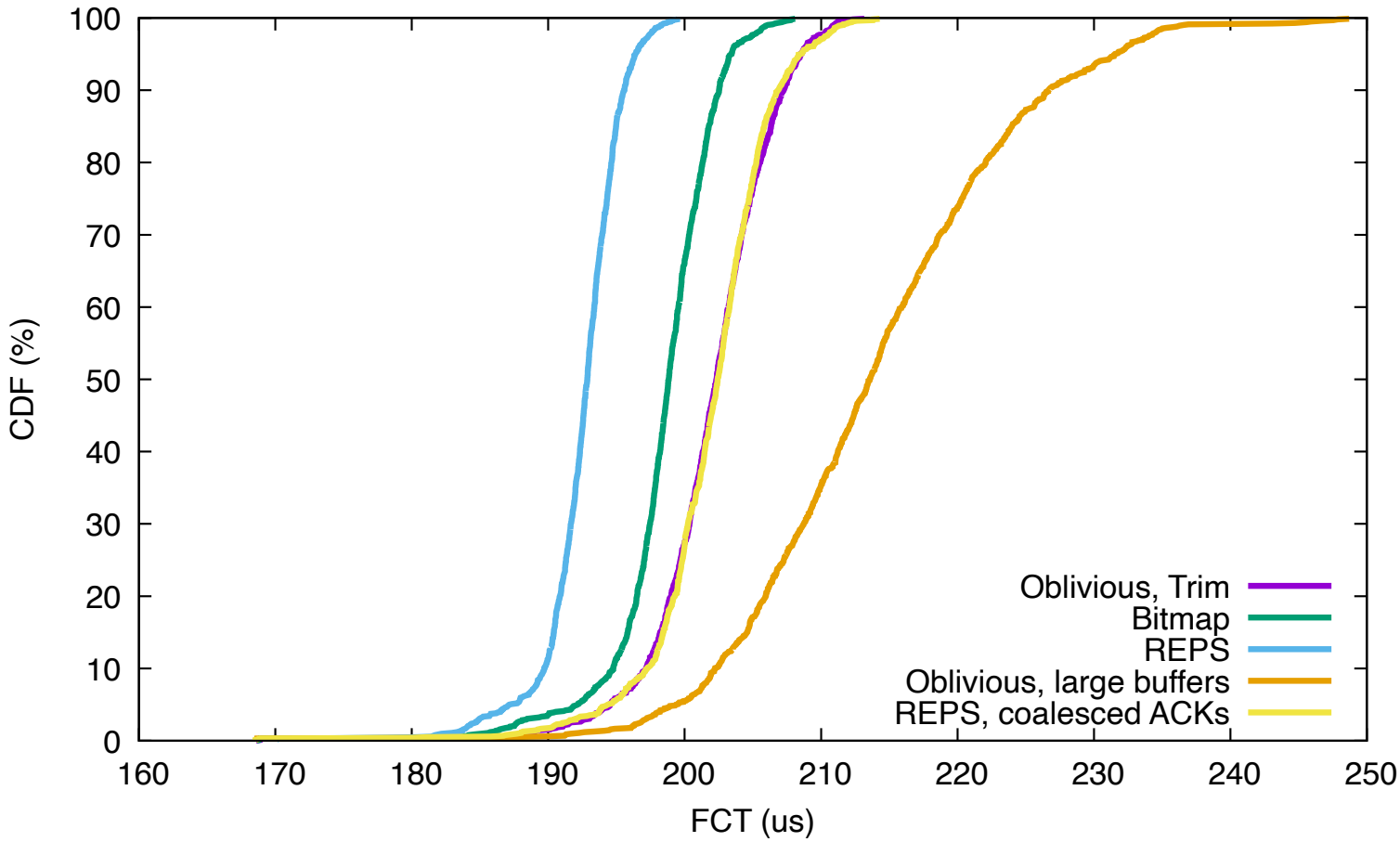

FCT increase (%)
45
40
35
30
25
20
15
10
5
Measured
FCT increase
Increasing
workload
Increasing
network size
Load (%)
100
90
80
70
60
50
40
30
No of nodes
50000
40000
30000
20000
10000
0
FCT increase (%)
50
40
30
20
10
0
The more paths we spray across,
the lower the tail FCT
(a) ECMP 16 val
) ECMP 32 values
(c) ECMP 64 values
(d) ECMP, all values

Keeping per path state

- Bitmap load balancing (e.g. Strack)
  - 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):
  - 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.

Load balancing algorithms comparison

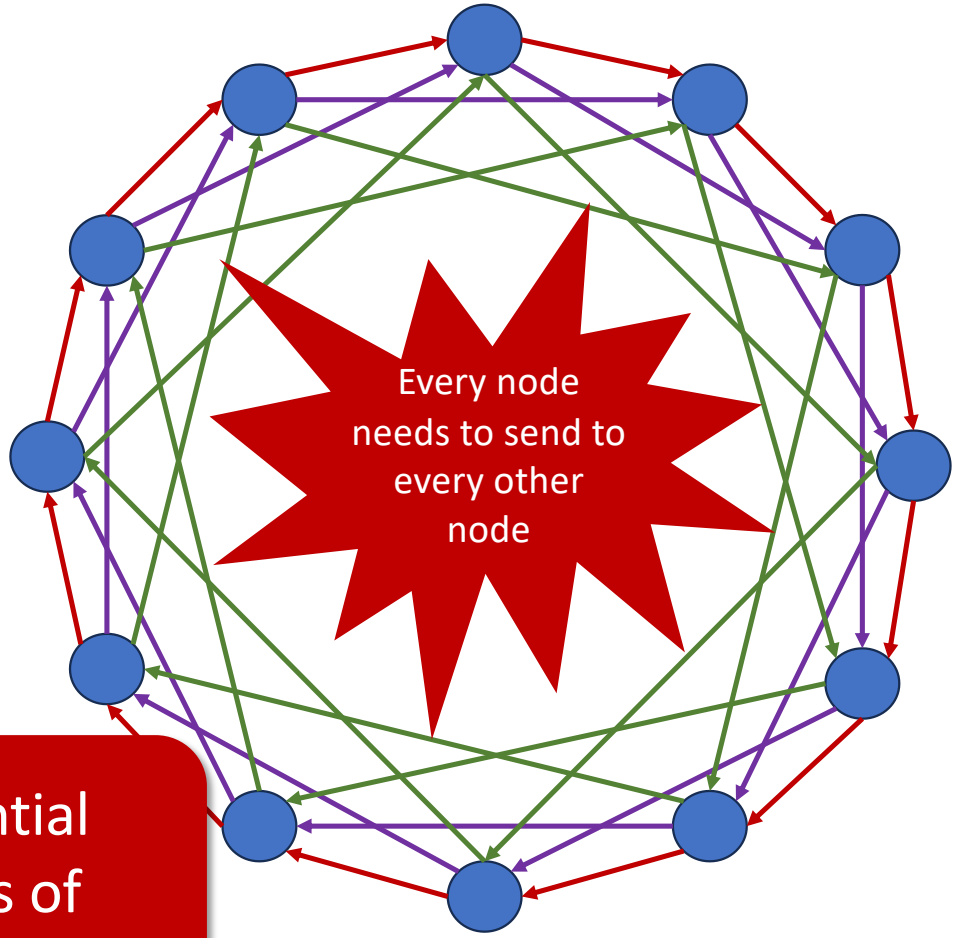


All-to-all Traffic Matrix

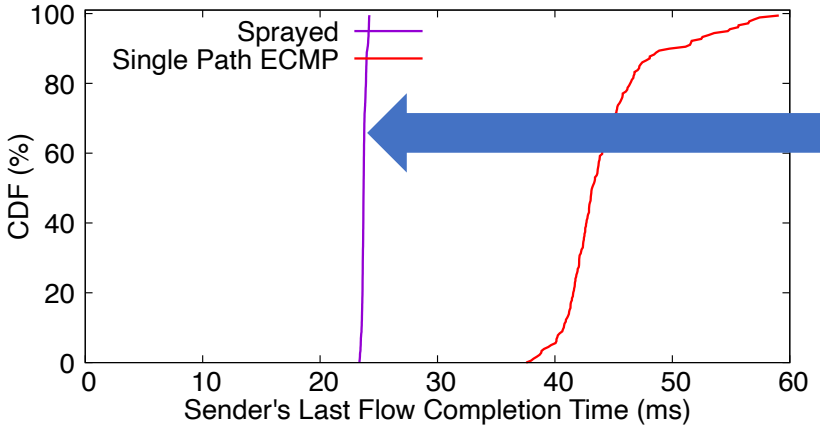
Parallel
Each node sends to n-1 nodes at one

Sequential
Each sends to one node, then send to next when finished, and so on.

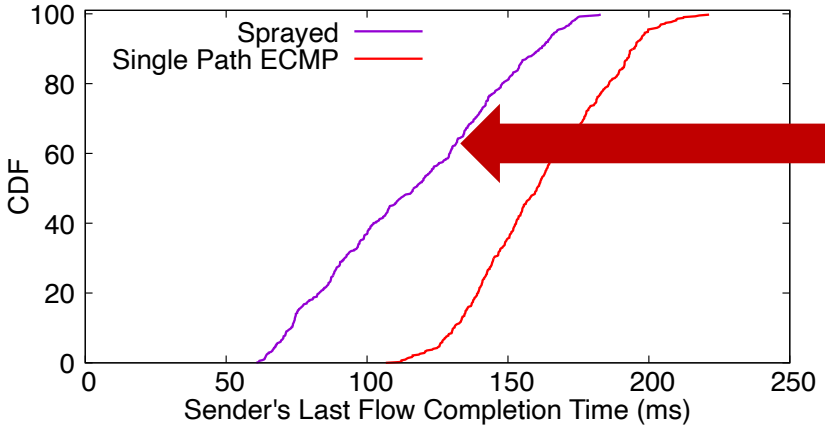
With spraying, sequential should just be a series of permutations. Easy!



Sequential All-to-all (1MB flows)



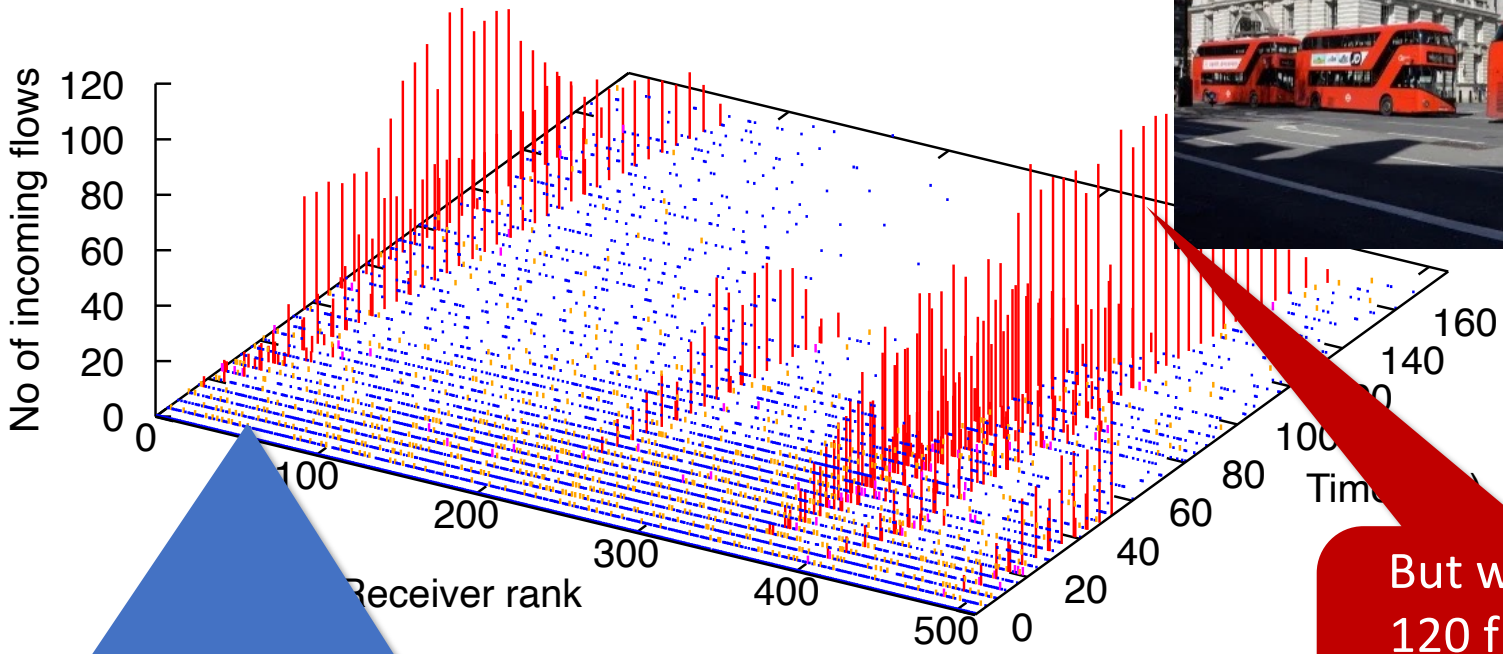
216 node all-to-all.
Spraying gives happy flows.



Same setup, but 512 node all-to-all.
Spraying gives unhappy flows.

What happened?

Sequential all-to-all: a closer look

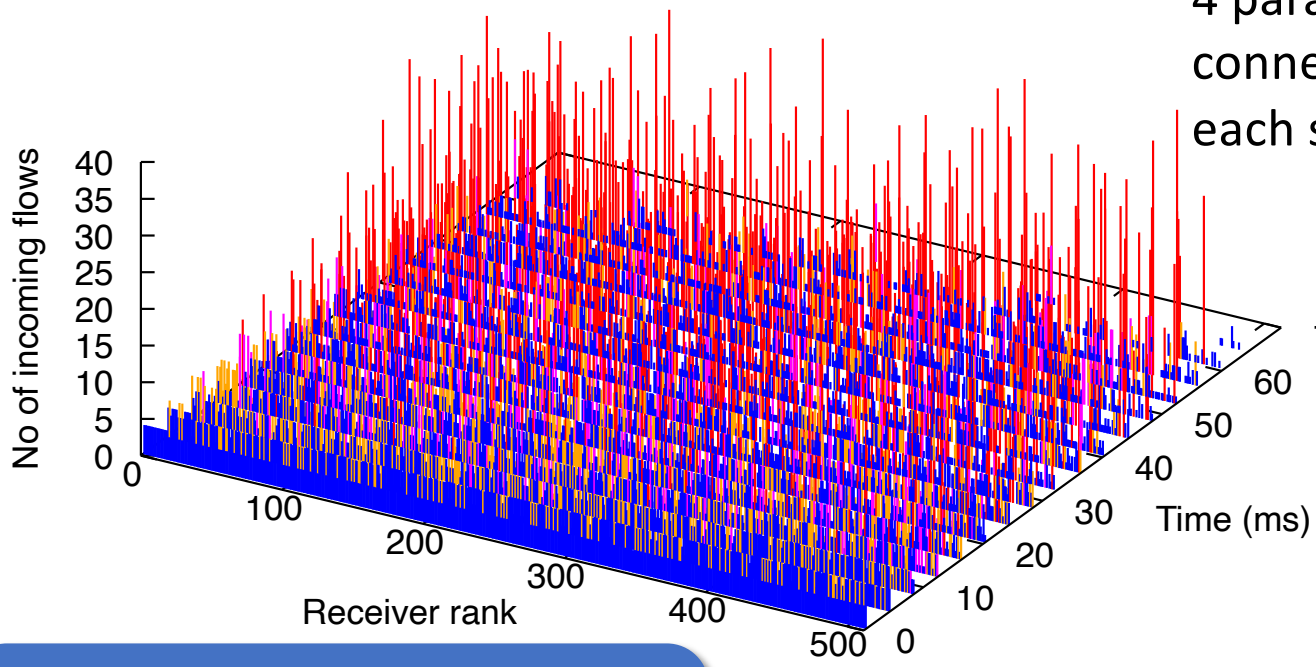


Each sender is sending to one receiver at a time
Each receiver *should* be receiving from one sender at a time.

But we see up to 120 flows to one receiver, with others idle

(Semi) parallel all-to-all

4 parallel connections from each sender at once



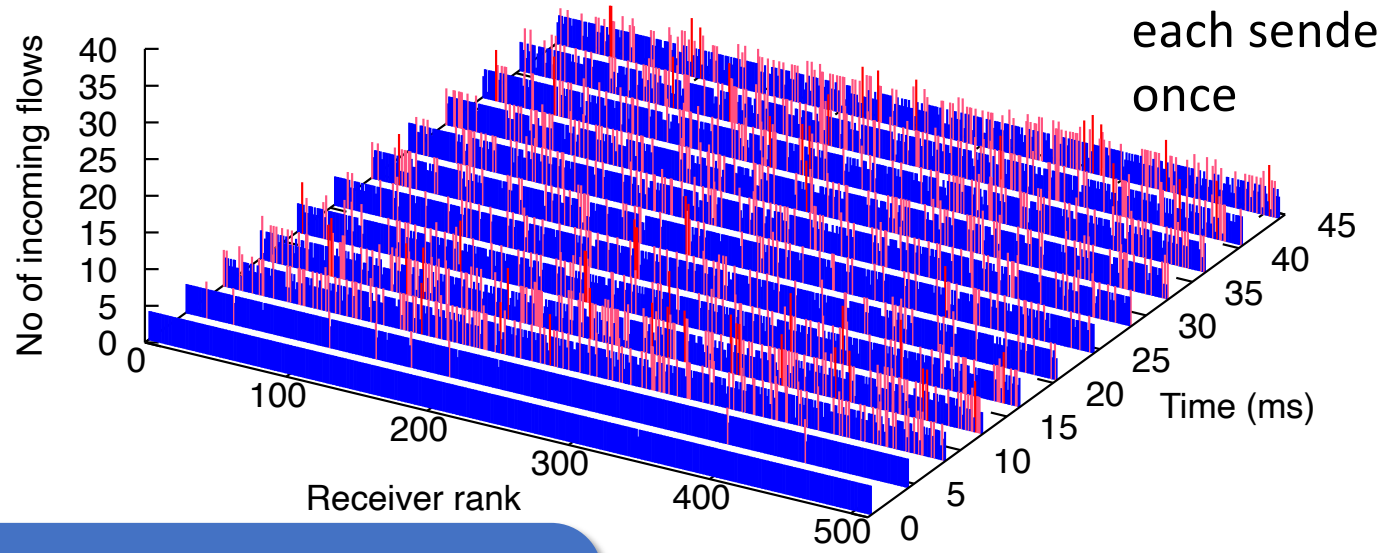
FCT was 170ms with sequential

Adding parallel connections really helps

Need > 20 connections from each sender at once to get close to optimal.

Prioritizing receiver credit

4 parallel connections from each sender at once



FCT was 65ms with 4 parallel connections and no priority

Each receiver can prioritize sending credit to its sender from the earliest round

AI/ML workloads are very demanding.

Full bisection bandwidth networks, multi-rail and multi-plane!

- Many flows take just a few BDPs.
- Transport protocol uses packet spraying.
  - –Flows start at line-rate.
  - –Packets spread on available paths by using (quasi-random)entropies.
  - –Switches trim packets on overload.
  - –Congestion control:
    - – Receiver or sender driven.
    - – Can be enabled simultaneously.

References

- I’ve got 99 problems but flops ain’t one – Gherghescu et al, Hotnets 2024.
- An edge-queued datagram service for all datacenter traffic – Olteanu et al, NSDI 2022.
- SMaRTT-REPS: Sender-based Marked Rapidly-adapting Trimmed & Timed Transport with Recycled Entropies – Bonato et al – Arxiv preprint, 2024.
- STrack: A Reliable Multipath Transport for AI/ML Clusters – Le at al. Arxiv preprint, 2024.