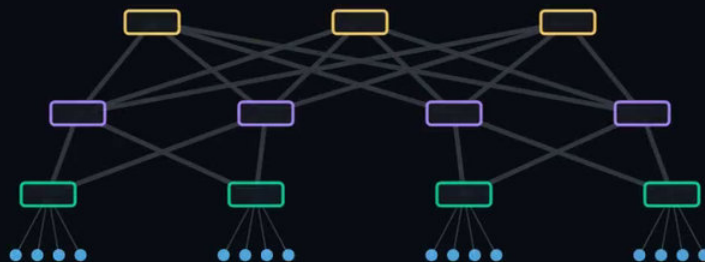


Fat-Tree



VL2

Many Paths, One Fabric

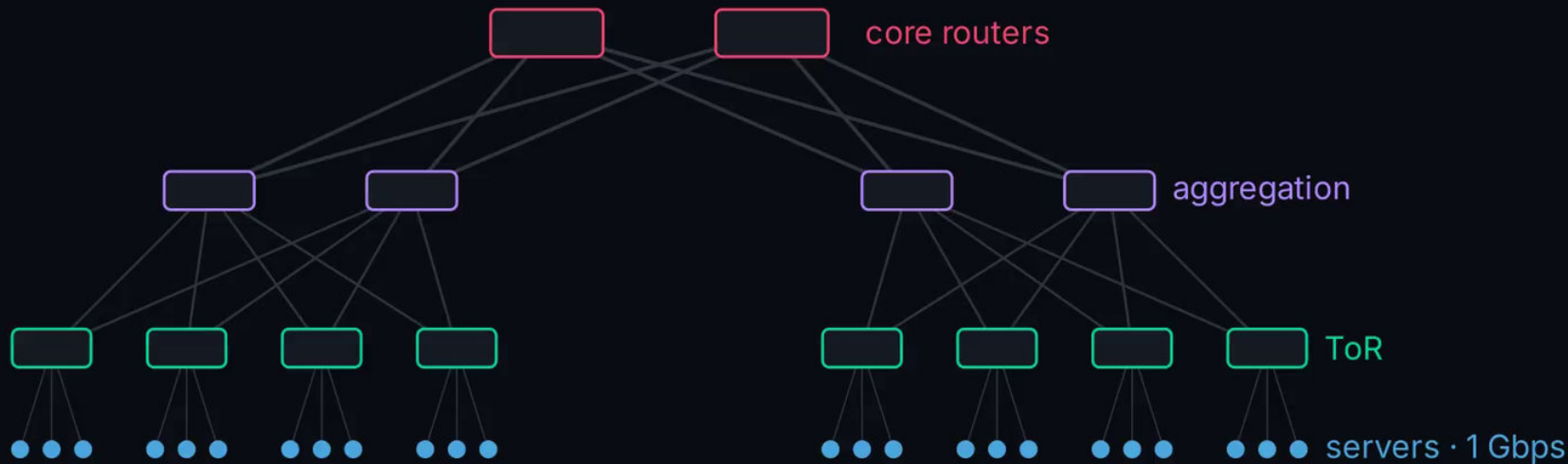
Fat-Tree and VL2 through the lens of multipath routing

Al-Fares, Loukissas, Vahdat — SIGCOMM 2008

Greenberg et al. (Microsoft Research) — SIGCOMM 2009

Both

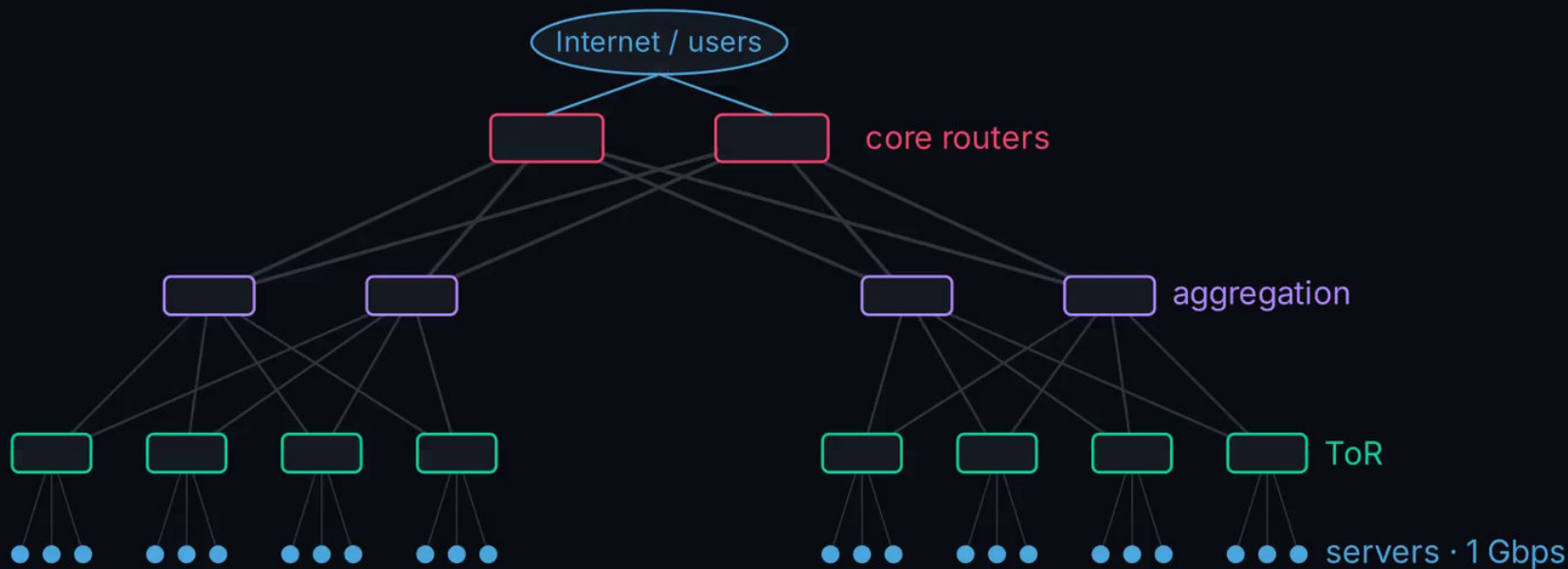
The problem: a tree has too few paths



The classic data-center tree

Both

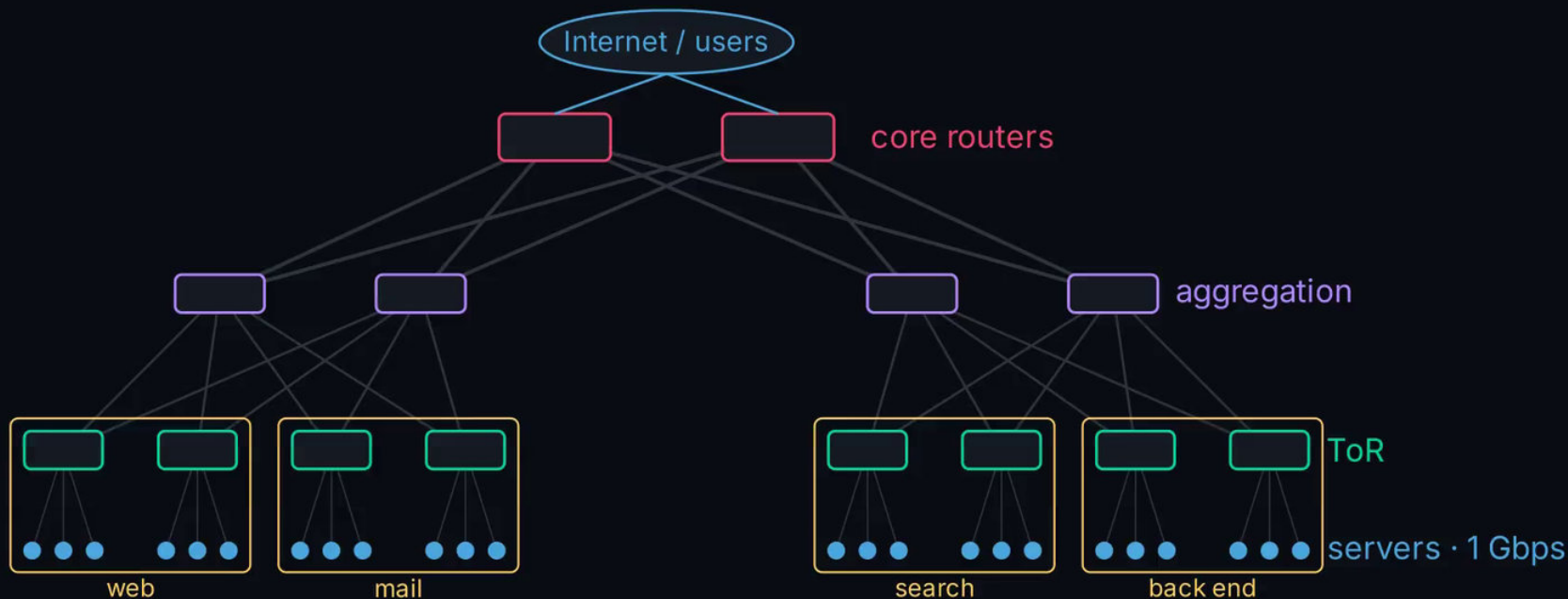
The problem: a tree has too few paths



The classic data-center tree

Both

The problem: a tree has too few paths

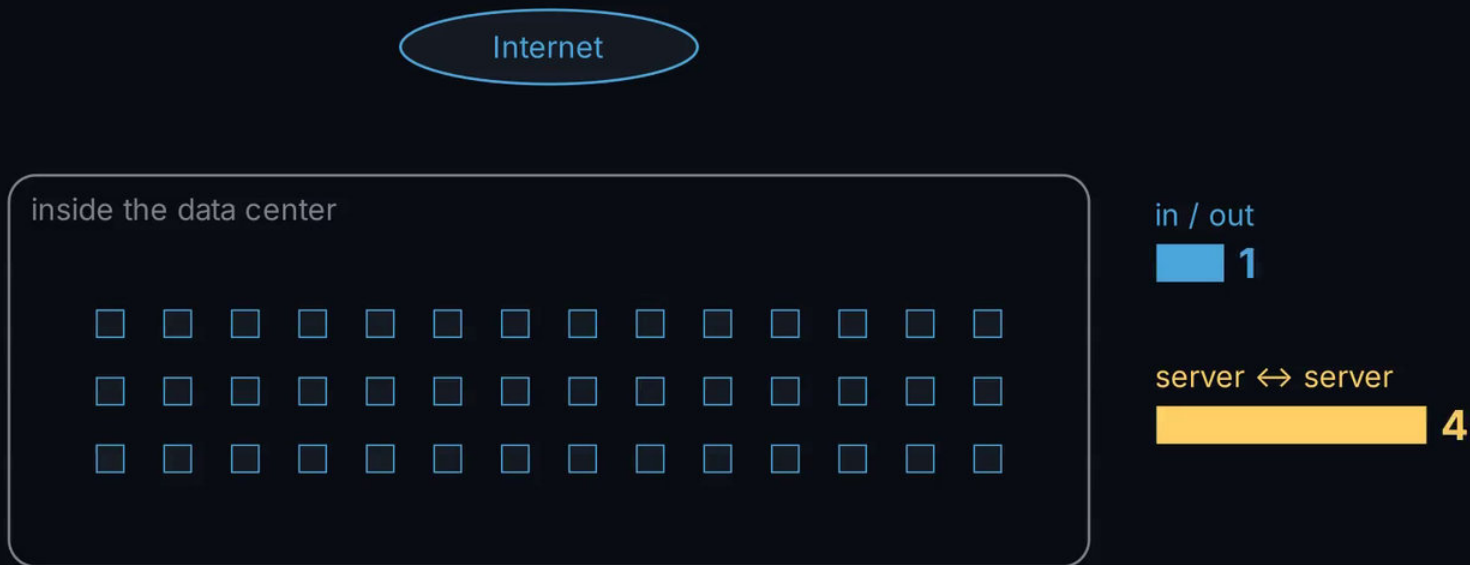


Built for: users $\leftrightarrow$ a service · each service in its own racks / VLAN

the dominant design pattern, adapted from Cisco's guide (VL2 §2, Fig. 1)

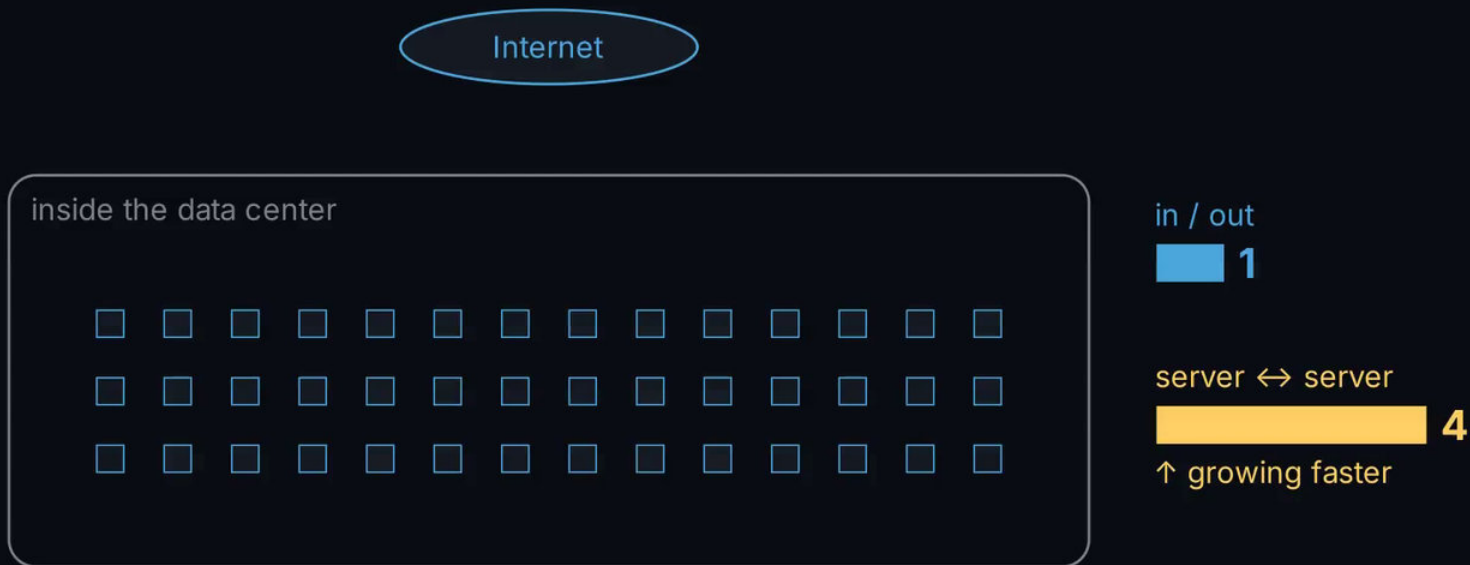
Both

The problem: a tree has too few paths



Both

The problem: a tree has too few paths

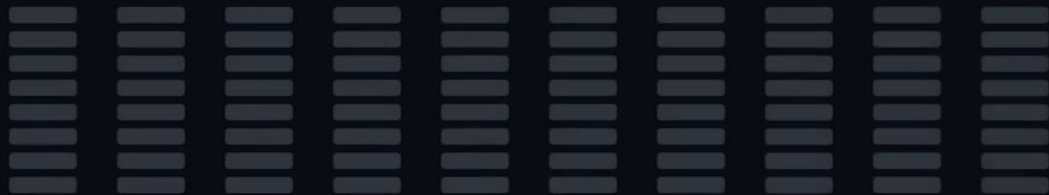


Most traffic is server ↔ server, inside the data center

VL2 §3.1: inside : in/out $\approx 4 : 1$, and the inside share is growing faster

Both

The problem: a tree has too few paths



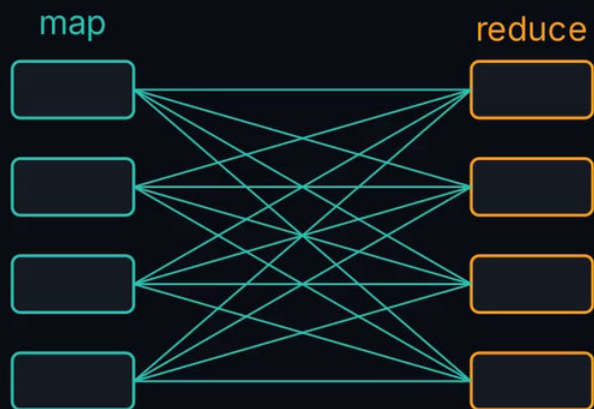
Both

The problem: a tree has too few paths



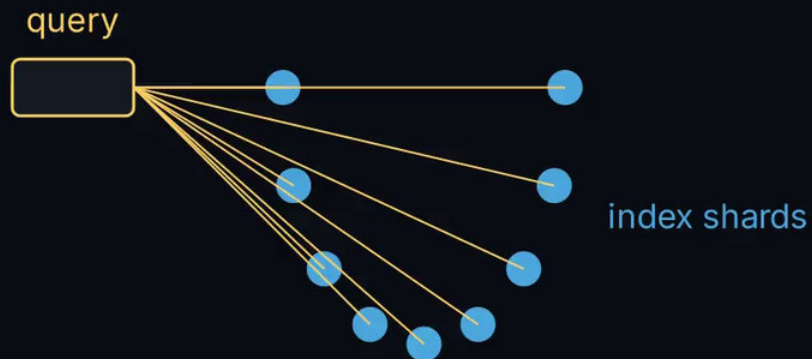
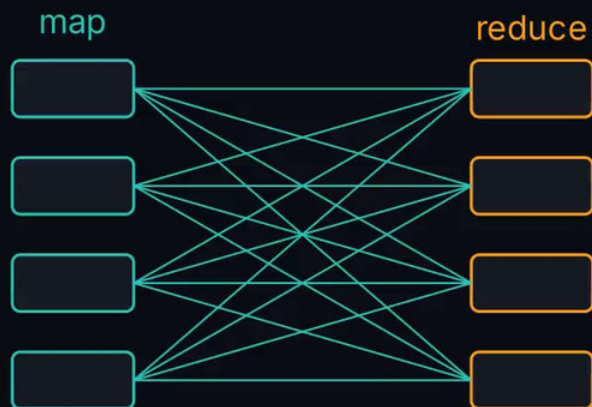
Both

The problem: a tree has too few paths



Both

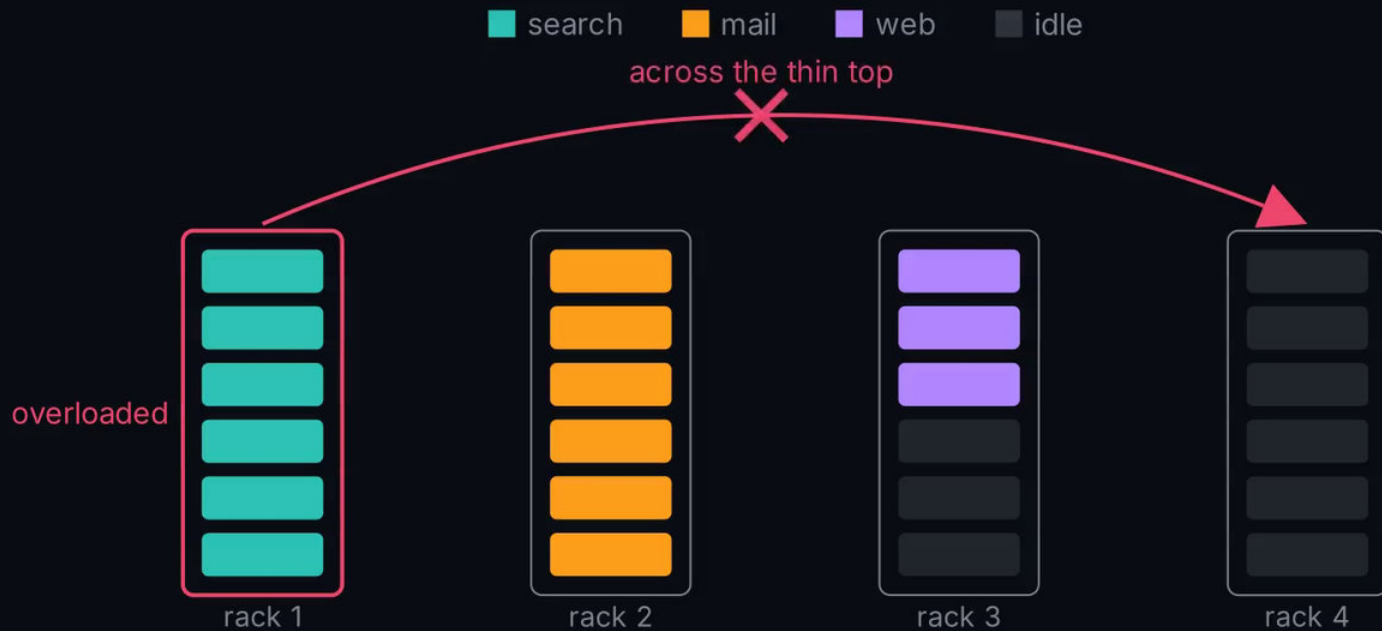
The problem: a tree has too few paths



Any server ↔ any server, at full NIC bandwidth

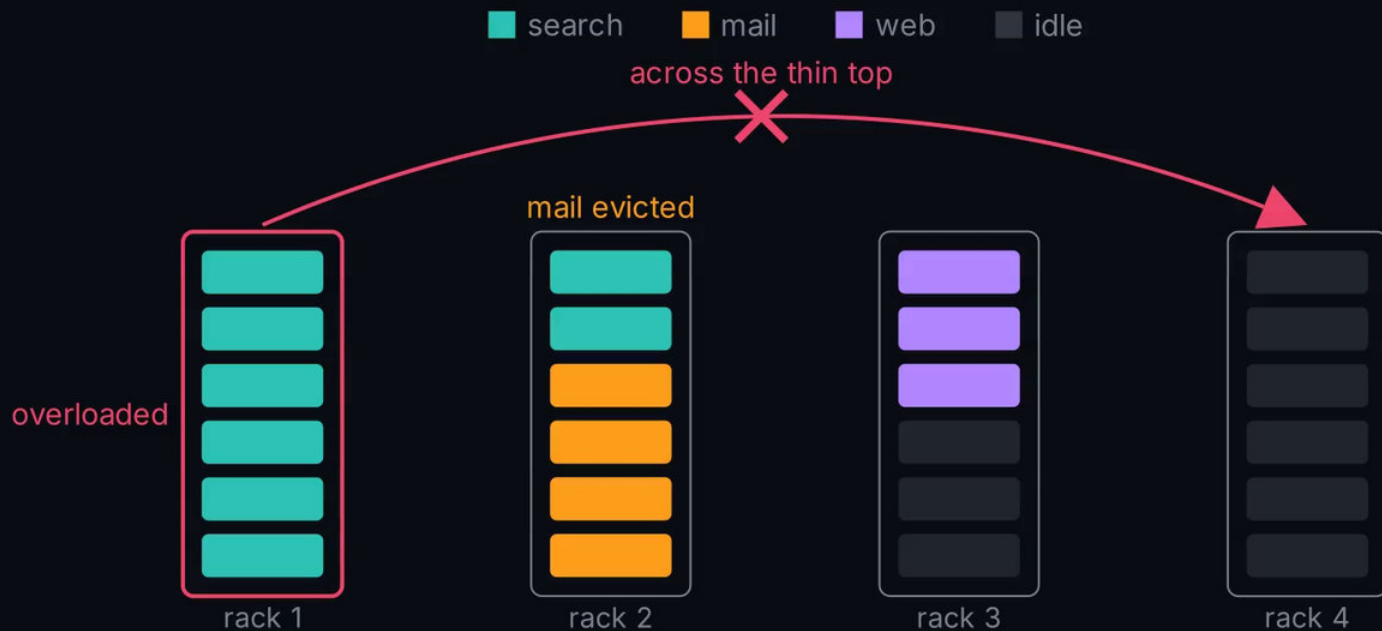
Both

The problem: a tree has too few paths



Both

The problem: a tree has too few paths



Without agility: hoard spares, evict neighbors, idle servers stay idle

VL2 §1-2

Both

The problem: a tree has too few paths

■ search ■ mail ■ web ■ idle

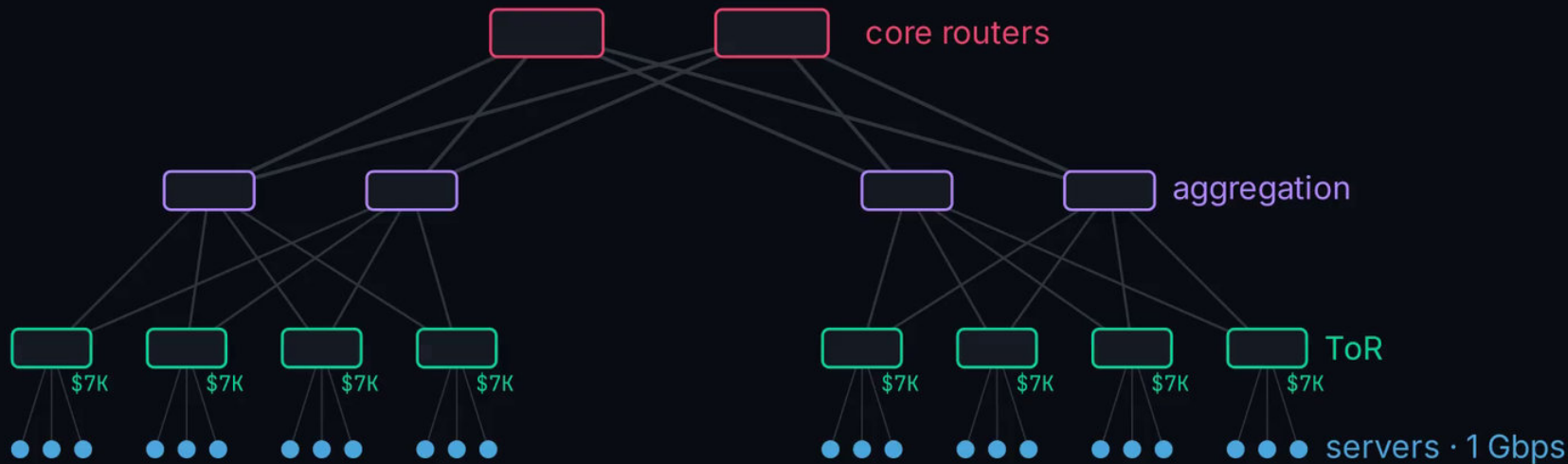


Agility: any server for any service

only possible if every pair of servers gets high capacity (VL2 §1)

Both

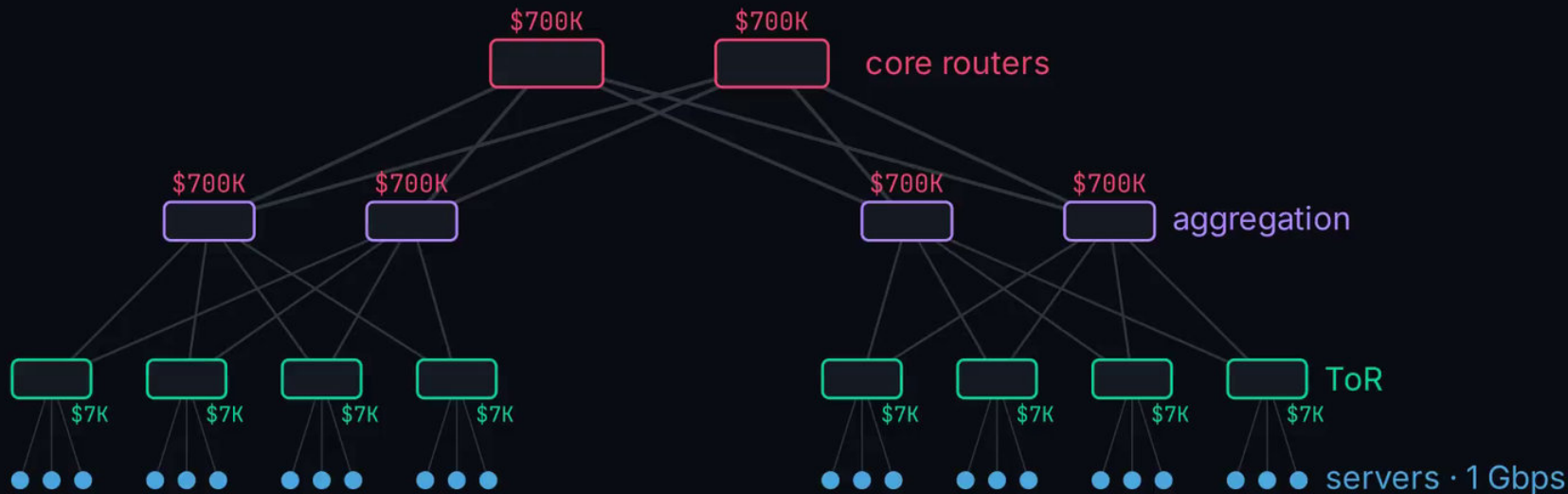
The problem: a tree has too few paths



ToR: 48-port GigE $\approx$ \$7,000

Both

The problem: a tree has too few paths

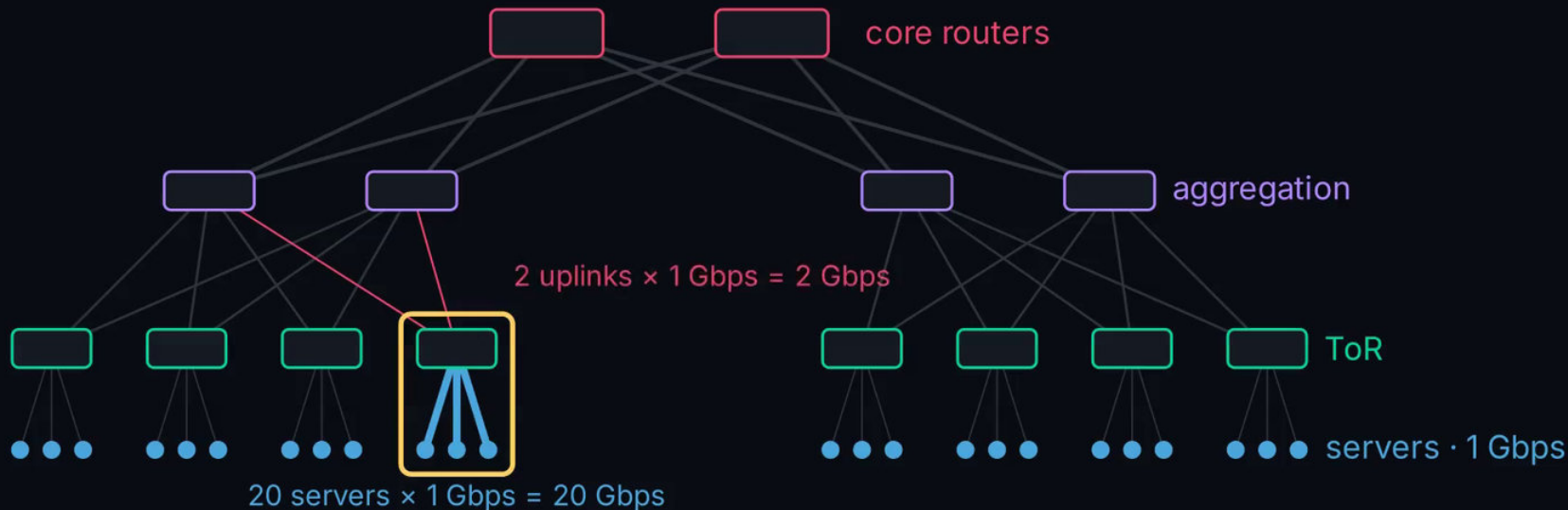


Above the ToR: 128-port 10 GigE $\approx$ \$700,000 each

full bandwidth for 20,000 hosts this way $\approx$ \$37M $\rightarrow$ buy far less capacity up top

Both

The problem: a tree has too few paths

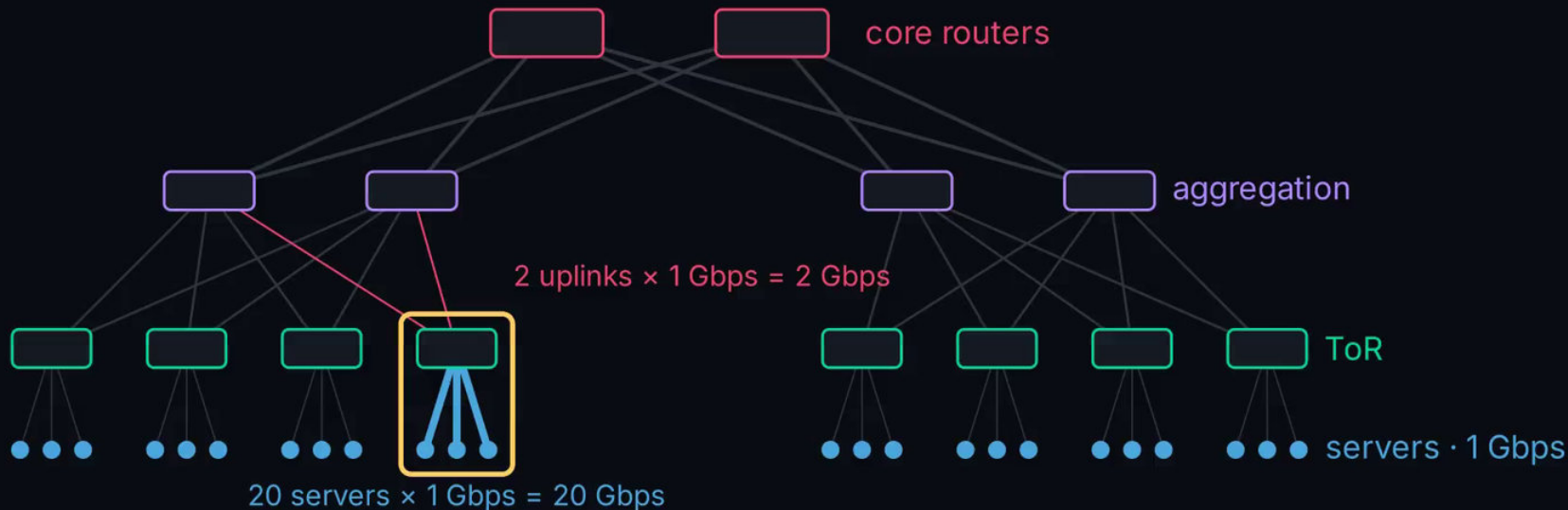


Above the ToR: 128-port 10 GigE ≈ \$700,000 each

full bandwidth for 20,000 hosts this way ≈ \$37M → buy far less capacity up top

Both

The problem: a tree has too few paths

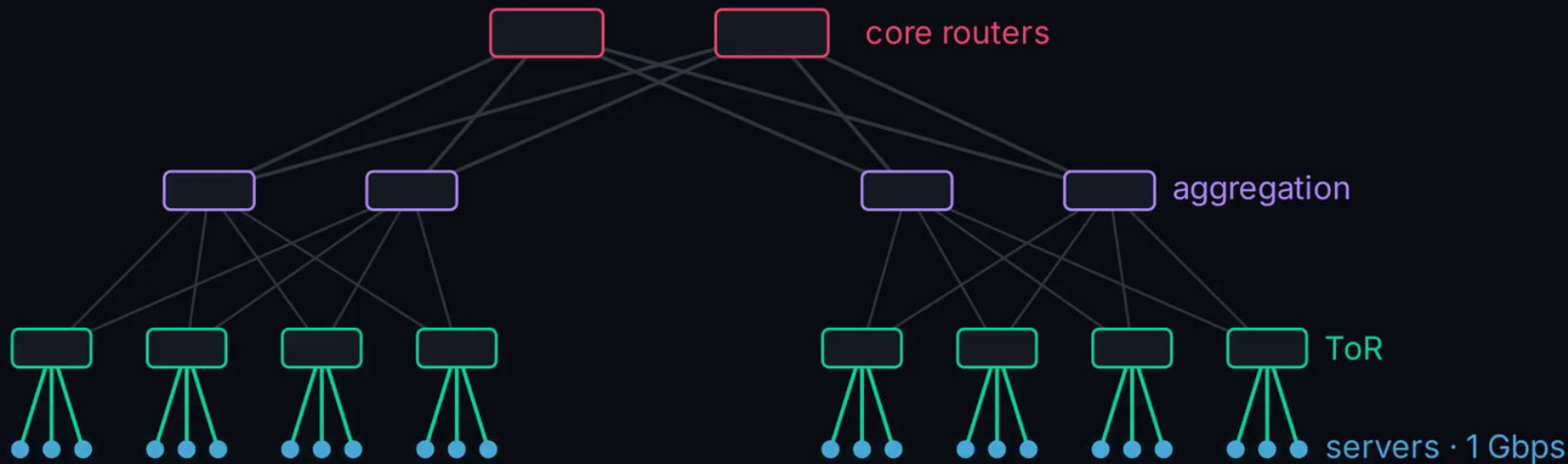


$2 : 20 = 1 : 10$ oversubscribed at the ToR

VL2 §2, measured in production: ToR uplinks 1:5 – 1:20 (1–4 Gbps up for 20 servers)

Both

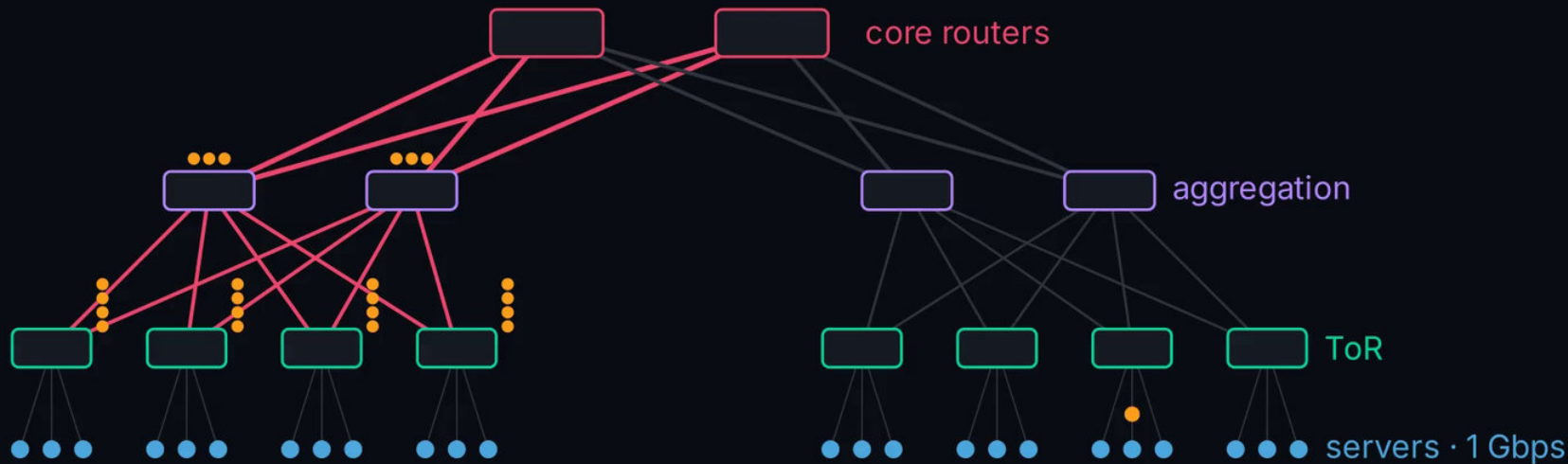
The problem: a tree has too few paths



Inside a rack: every server talks at its full 1 Gbps

Both

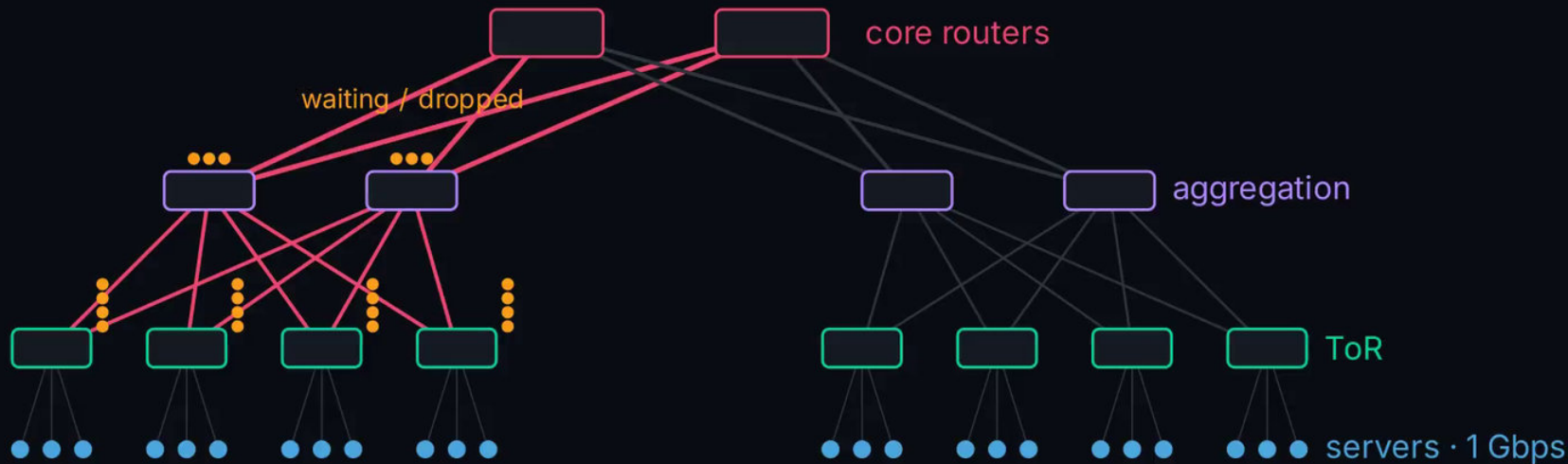
The problem: a tree has too few paths



Across racks: the same servers now need the uplinks

Both

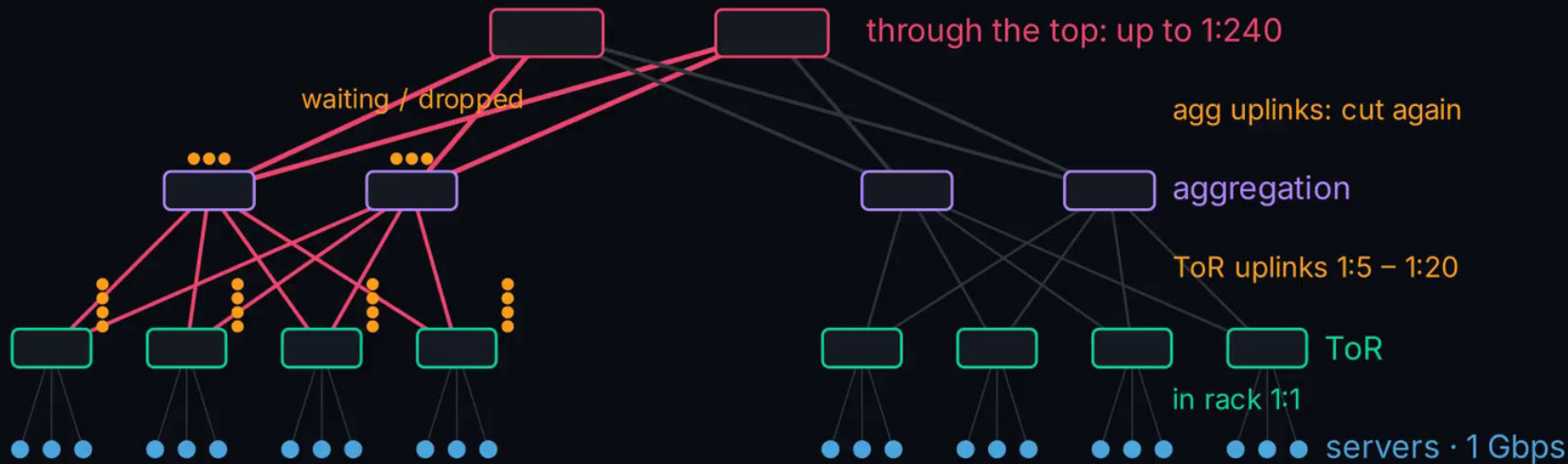
The problem: a tree has too few paths



Cross-rack: 24 packets leave the servers, 2 reach the other side
all 20 servers off-rack: $2 \text{ Gbps} / 20 = 100 \text{ Mbps}$ each

Both

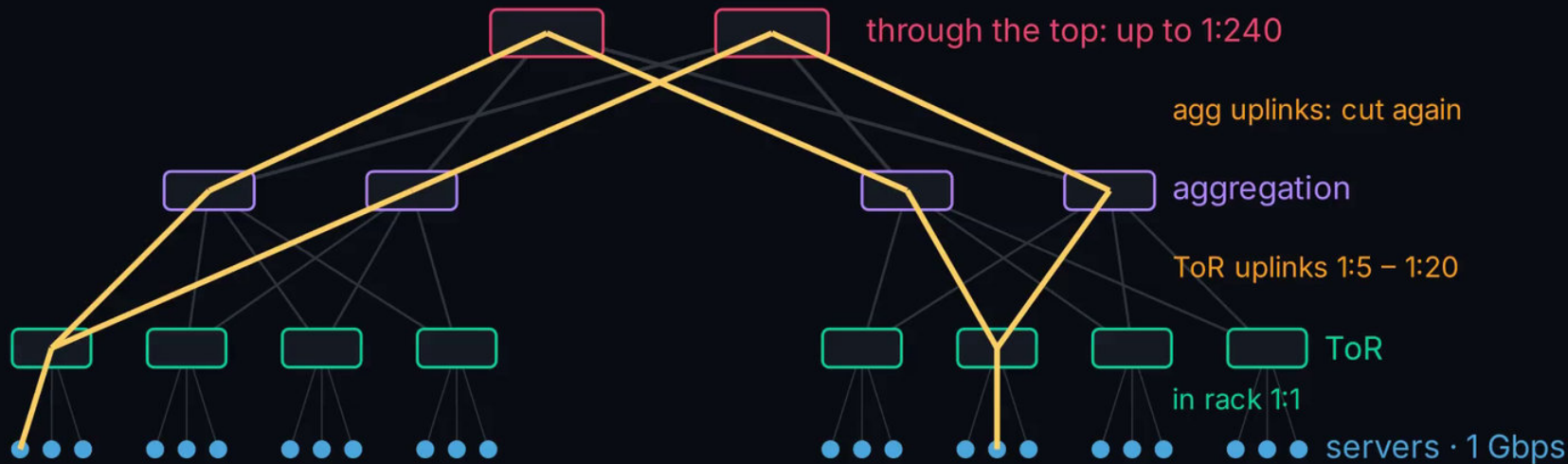
The problem: a tree has too few paths



across the top at 1:240: $1 \text{ Gbps} / 240 \approx 4 \text{ Mbps}$
measured in Microsoft's data centers (VL2 §2) → services get crammed into one rack

Both

The problem: a tree has too few paths

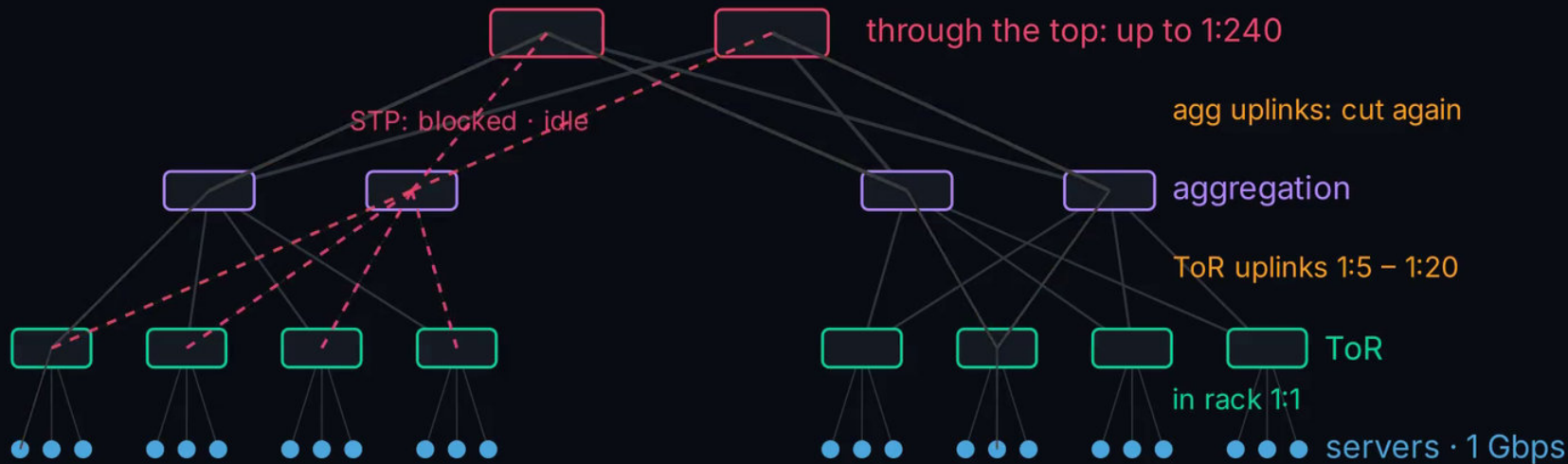


Only 2 paths between any two racks

the conventional topology offers at most two paths (VL2 §2)

Both

The problem: a tree has too few paths

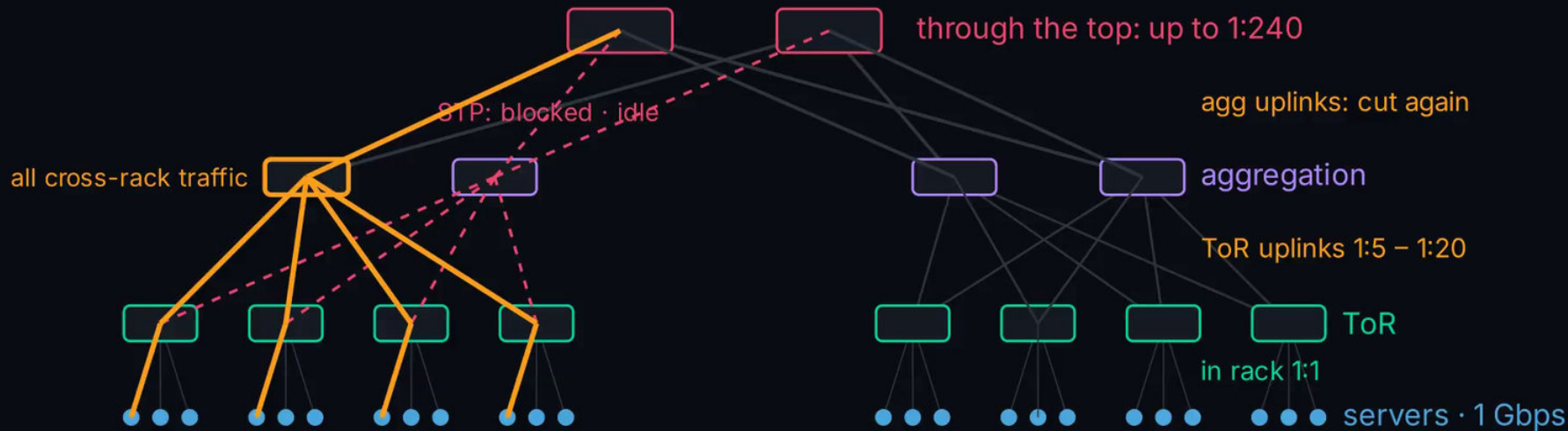


Only 2 paths between any two racks

the conventional topology offers at most two paths (VL2 §2)

Both

The problem: a tree has too few paths

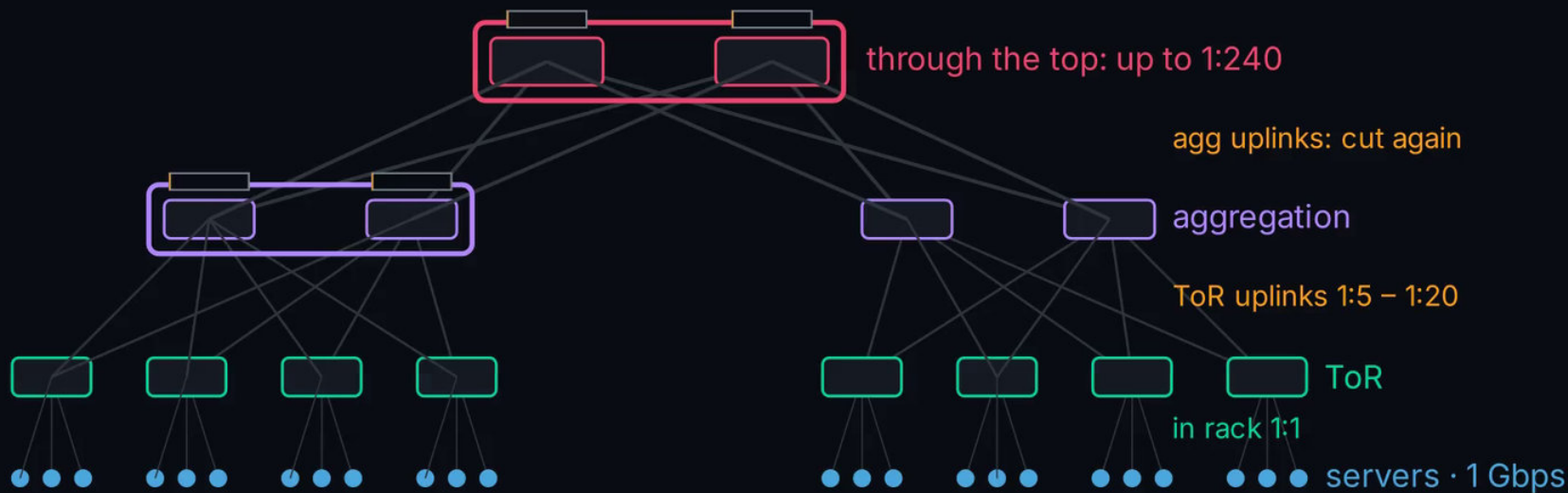


Spanning Tree: one aggregation switch carries the whole domain

its twin is blocked and idle (VL2 §2: "only a single path to be used")

Both

The problem: a tree has too few paths

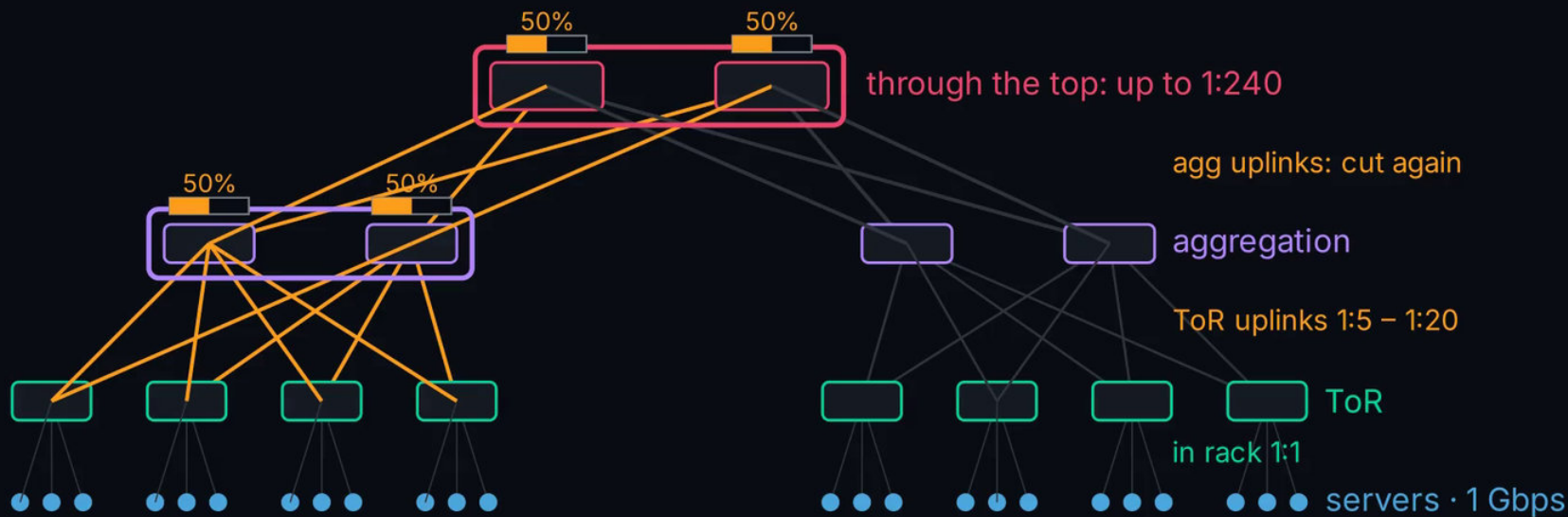


Above the ToR, every switch is paired with a twin (1:1)

VL2 §2

Both

The problem: a tree has too few paths

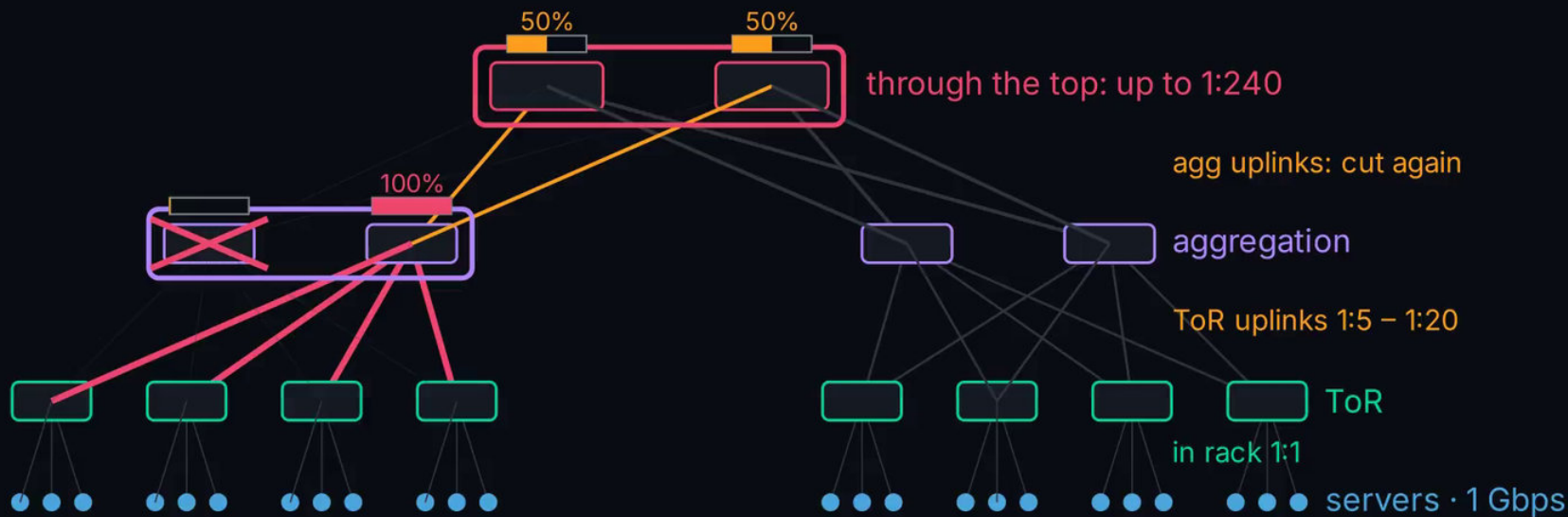


Both carry traffic, but each keeps half its capacity free for its twin

planned this way every day, not only during failures

Both

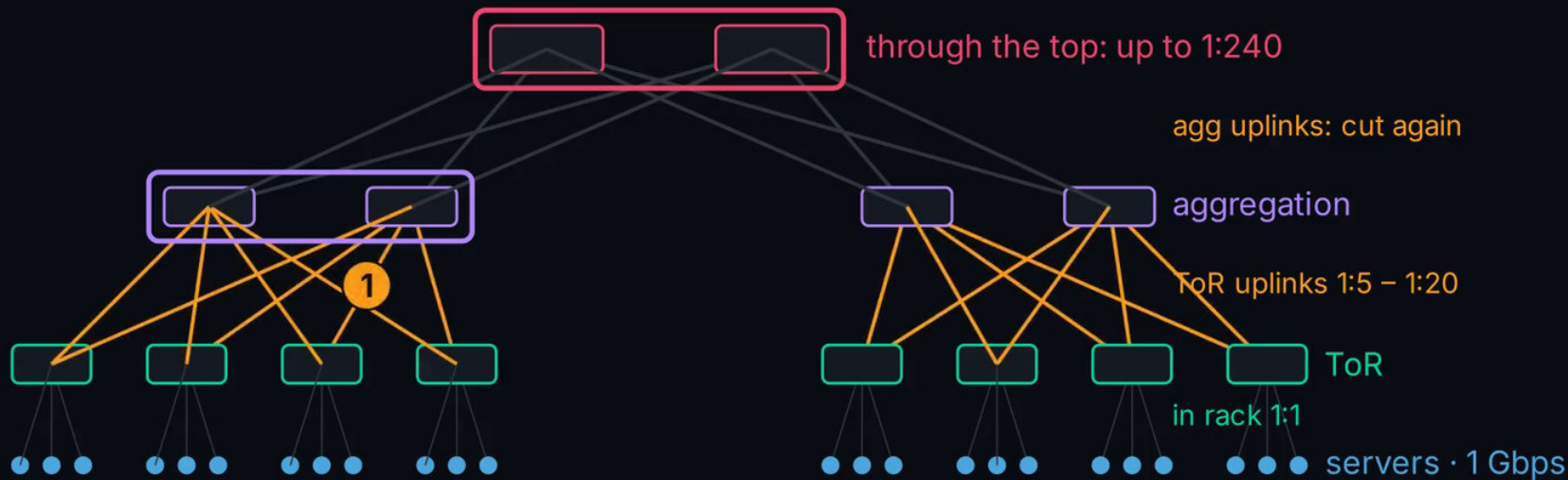
The problem: a tree has too few paths



Why the cap: if one fails, its partner must absorb everything → exactly 100%

Both

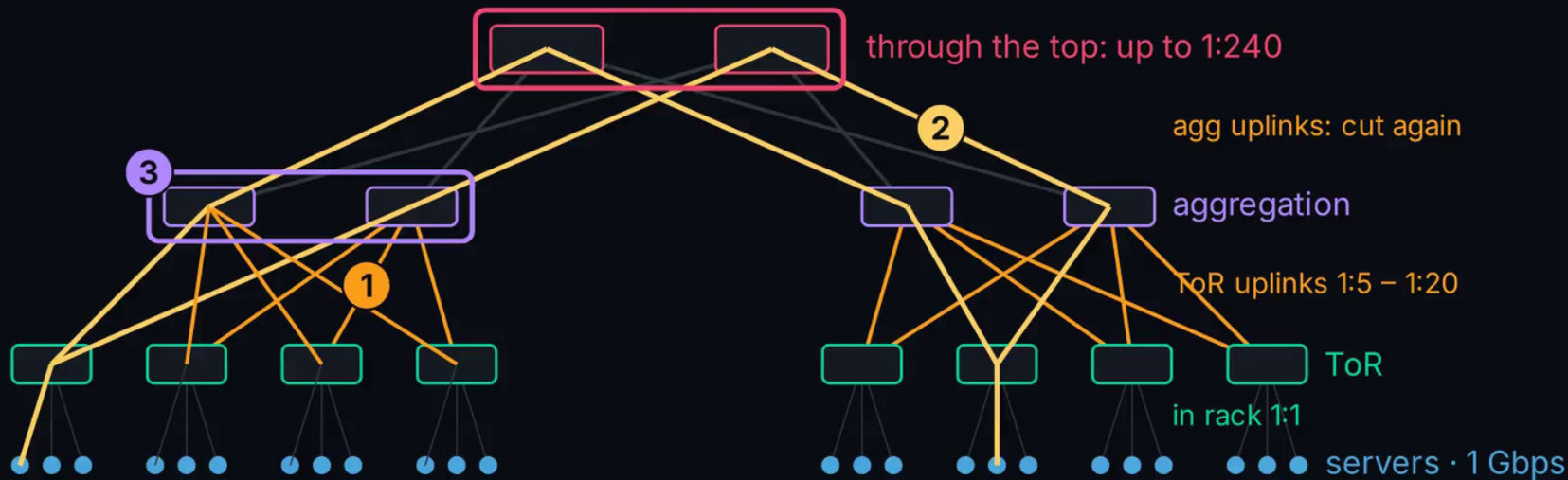
The problem: a tree has too few paths



Why the cap: if one fails, its partner must absorb everything → exactly 100%

Both

The problem: a tree has too few paths



① capacity cut: 1:20 ... 1:240

② 2 paths; STP uses 1

③ every box $\leq 50\%$ loaded

Both papers make the same first move:

build a Clos topology from many cheap commodity switches

- ① cheap switches everywhere → no need to cut capacity (1:1)
- ② hundreds of equal-cost paths, all active
- ③ lose 1 of n switches → lose only $1/n$, not $1/2$ (VL2 §4.1)

Both

The problem: a tree has too few paths

Both papers make the same first move:

build a Clos topology from many cheap commodity switches

- ① cheap switches everywhere → no need to cut capacity (1:1)
- ② hundreds of equal-cost paths, all active
- ③ lose 1 of n switches → lose only $1/n$, not $1/2$ (VL2 §4.1)

Then the real question:

who picks the path for each flow, and how?

Both Act 4 · Same blueprint, two builders

Fat-Tree · UC San Diego · 2008

budget: commodity 48-port 1 GigE switches only

can change: the switch's route lookup

can't assume: anything about the hosts

Both Act 4 · Same blueprint, two builders

Fat-Tree · UC San Diego · 2008

budget: commodity 48-port 1 GigE switches only

can change: the switch's route lookup

can't assume: anything about the hosts

VL2 · Microsoft Research · 2009

runs production data centers today

can change: the OS on every server

can't change: the switches (stock only)

Fat-Tree · UC San Diego · 2008

budget: commodity 48-port 1 GigE switches only

can change: the switch's route lookup

can't assume: anything about the hosts

VL2 · Microsoft Research · 2009

runs production data centers today

can change: the OS on every server

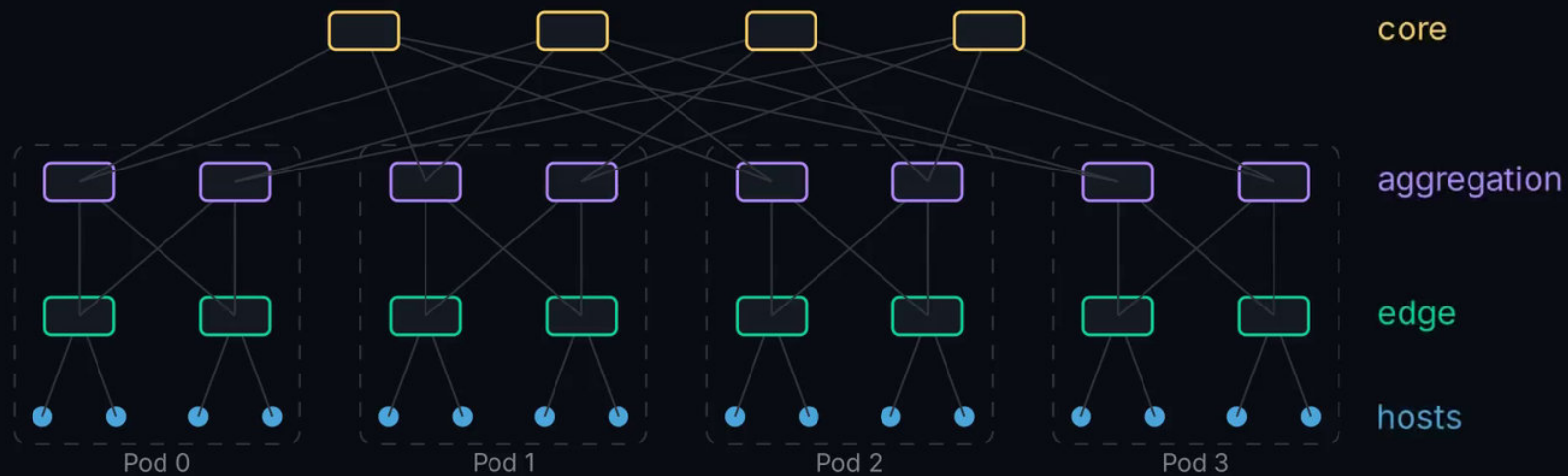
can't change: the switches (stock only)

Same goal

many paths × many cheap switches → full bandwidth, low cost, and a failure costs $1/n$, not $1/2$

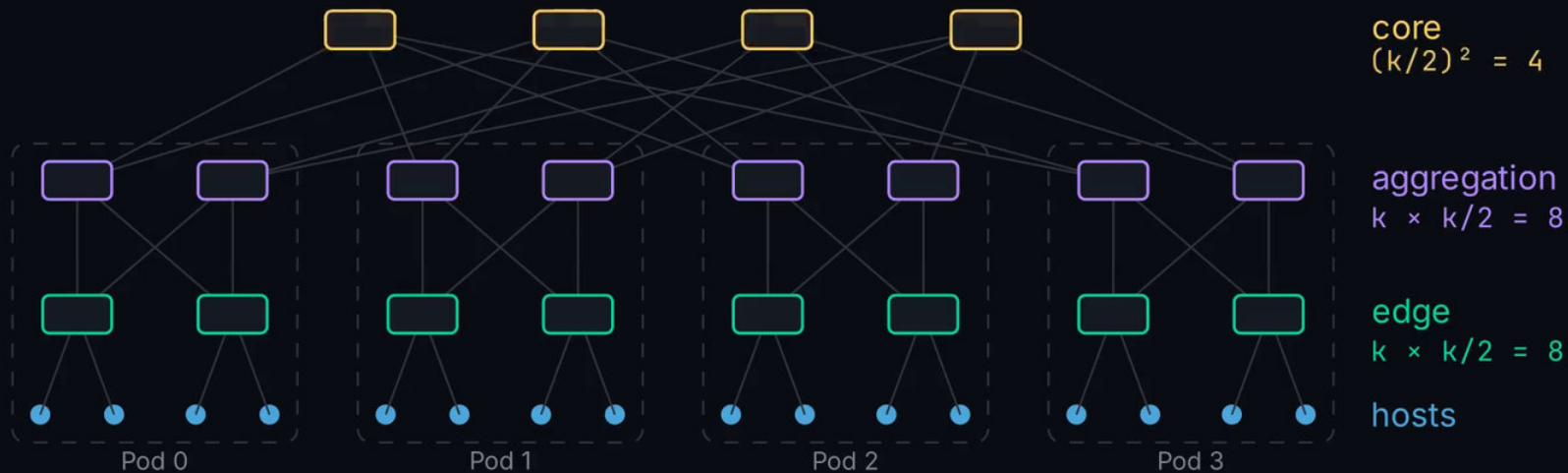
Both

Act 4 · Same blueprint, two builders



Both

Act 4 · Same blueprint, two builders

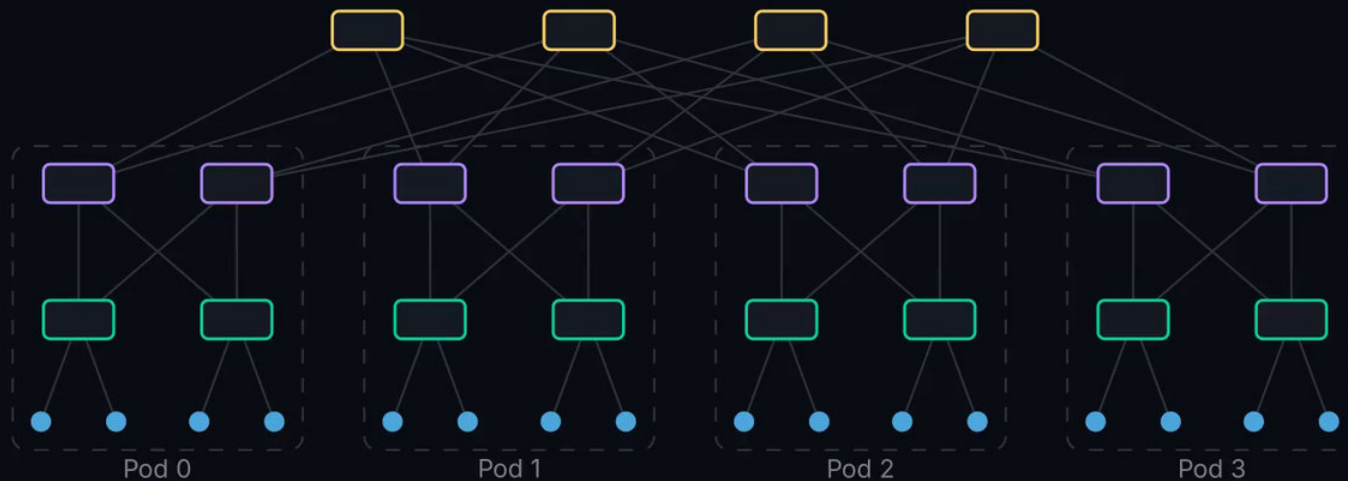


k = ports per switch (here k = 4)

each edge switch: $k/2 = 2$ ports down to hosts, $k/2 = 2$ up

Both Act 4 · Same blueprint, two builders

$k = 4$ ports / switch



core
 $(k/2)^2 = 4$

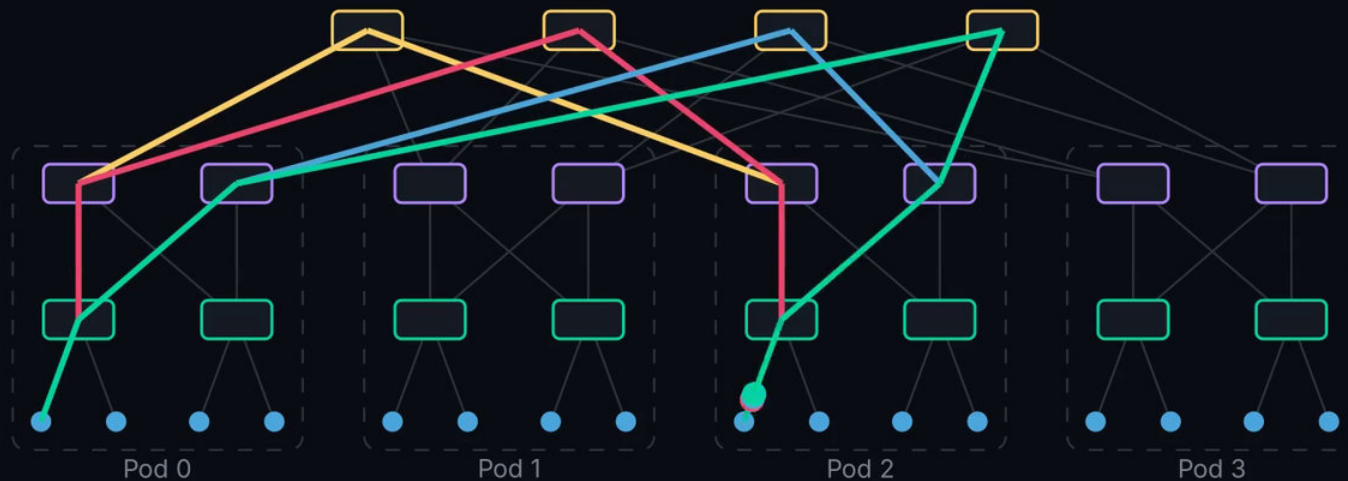
aggregation
 $k \times k/2 = 8$

edge
 $k \times k/2 = 8$

hosts
 $k^3/4 = 16$

Both Act 4 · Same blueprint, two builders

$k = 4$ ports / switch



core
 $(k/2)^2 = 4$

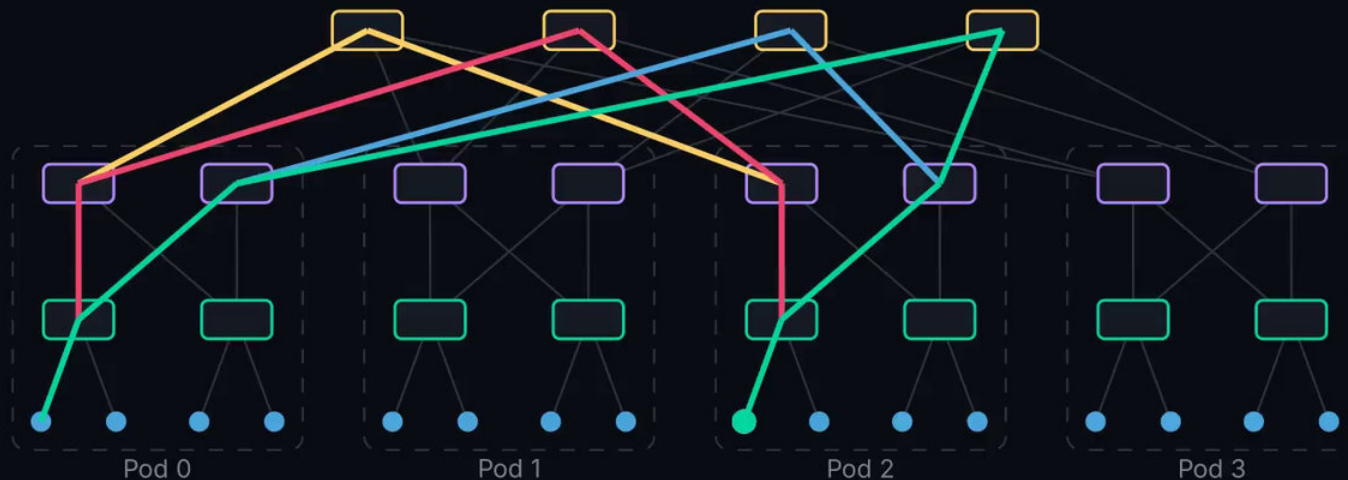
aggregation
 $k \times k/2 = 8$

edge
 $k \times k/2 = 8$

hosts
 $k^3/4 = 16$

Both Act 4 · Same blueprint, two builders

$k = 4$ ports / switch



core
 $(k/2)^2 = 4$

aggregation
 $k \times k/2 = 8$

edge
 $k \times k/2 = 8$

hosts
 $k^3/4 = 16$

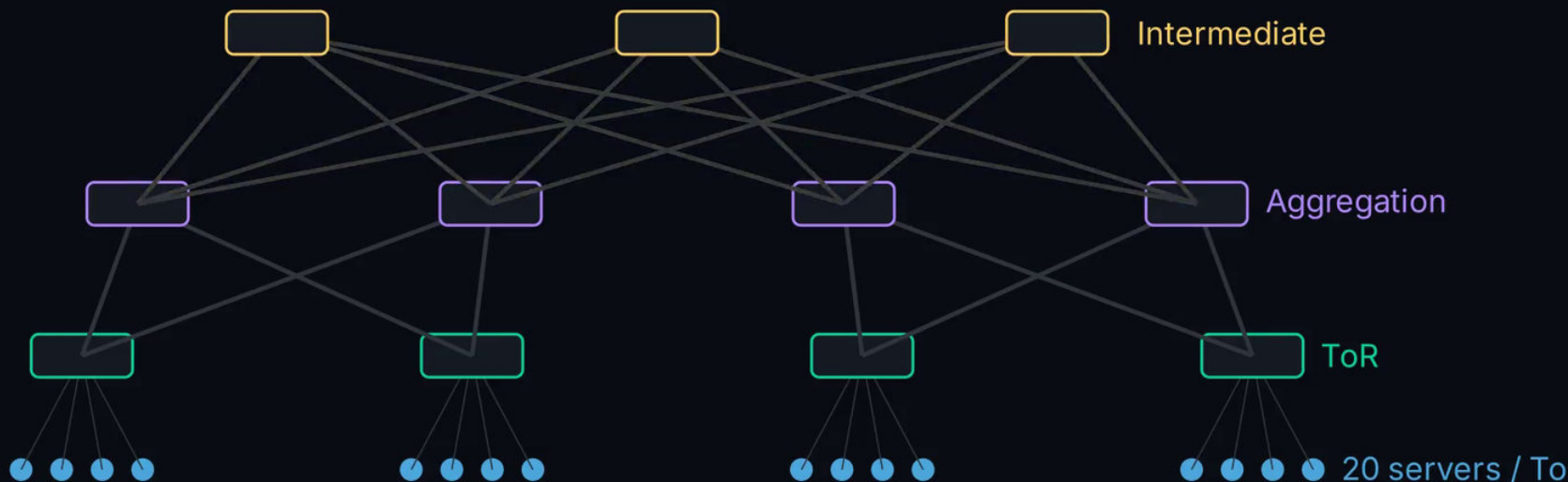
$(k/2)^2$ core paths between any two pods

$k = 48$: 27,648 hosts · 576 paths · $\approx$ \$8.64M (the tree: $\approx$ \$37M)

Fat-Tree §2.2; est. 56.6% less power than the hierarchy (§3.9)

Both

Act 4 · Same blueprint, two builders

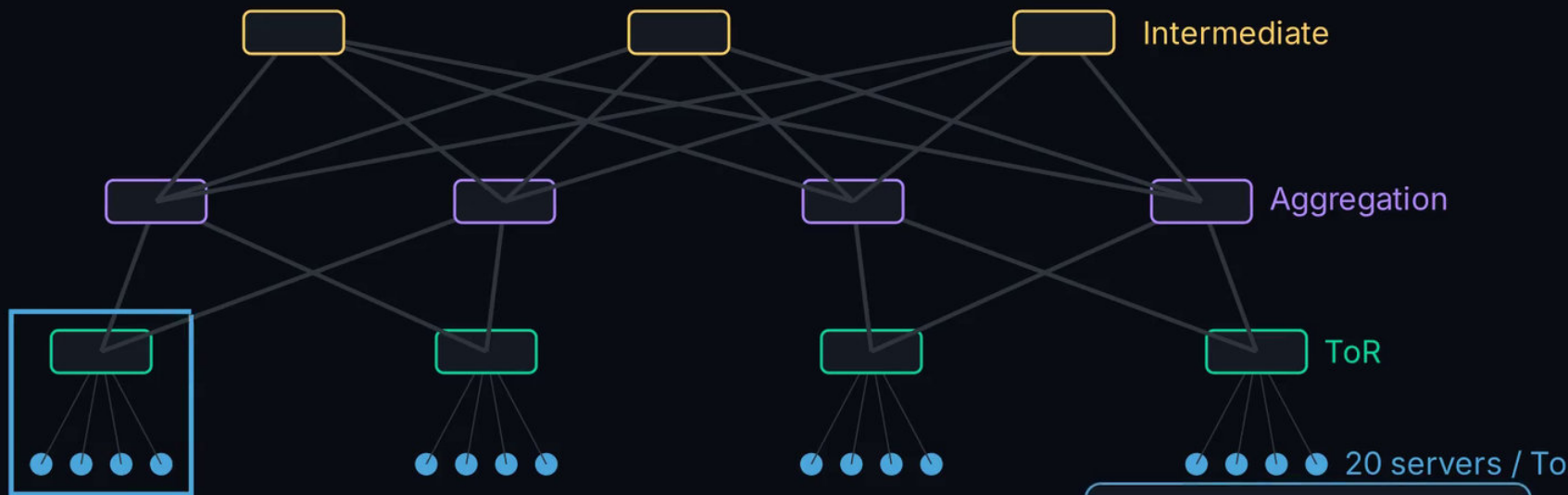


thick = 10G switch links

thin = 1G server links

Both

Act 4 · Same blueprint, two builders



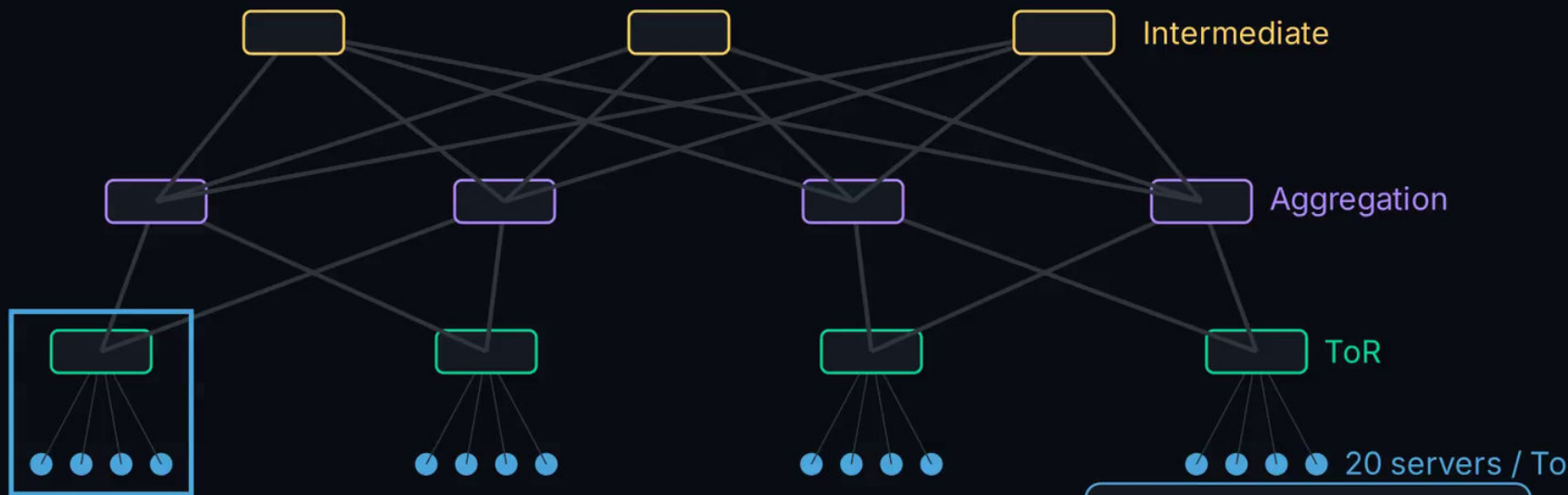
One rack stays 1:1

down: $20 \times 1\text{G} = 20\text{G}$

up: $2 \times 10\text{G} = 20\text{G}$

Both

Act 4 · Same blueprint, two builders



VL2 §6: 1:1 VL2 costs the same as today's 1:240 network
a conventional 1:1 network would cost $\approx 14\times$ more

One rack stays 1:1

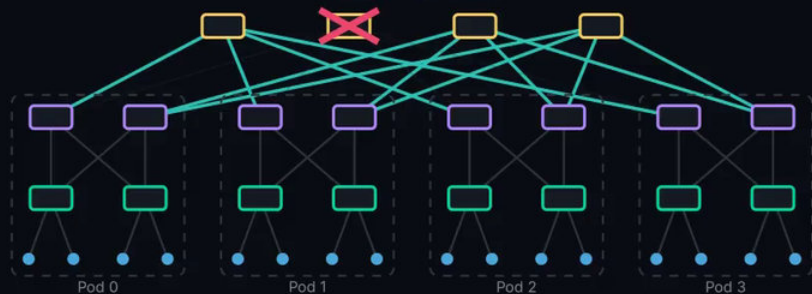
down: $20 \times 1\text{G} = 20\text{G}$

up: $2 \times 10\text{G} = 20\text{G}$

Both

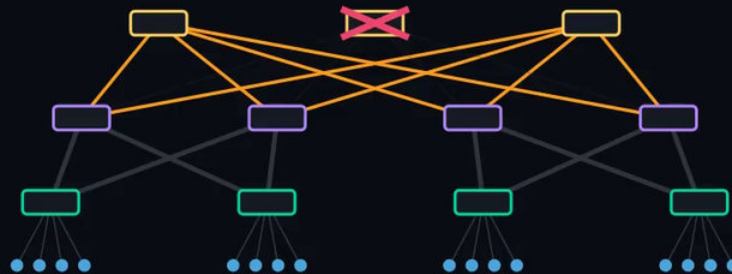
Act 4 · Same blueprint, two builders

Fat-Tree



lose 1 of $(k/2)^2$ cores $\rightarrow$ lose $1/(k/2)^2$

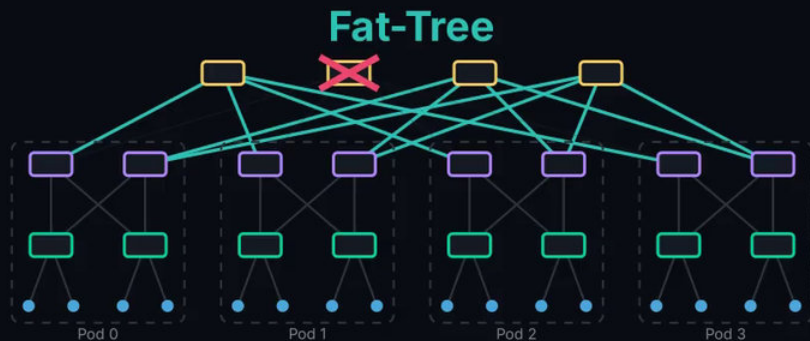
VL2



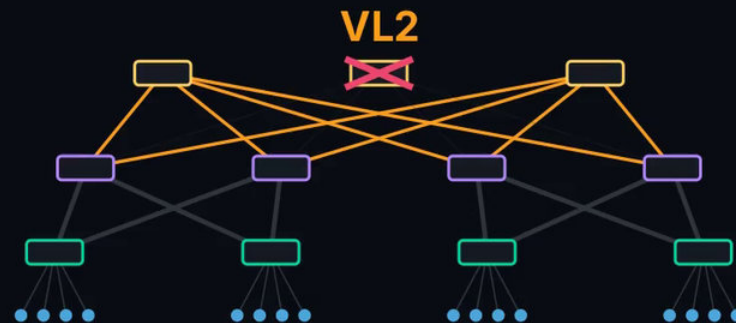
lose 1 of n Intermediates $\rightarrow$ lose $1/n$ (VL2 §4.1)

Both

Act 4 · Same blueprint, two builders



lose 1 of $(k/2)^2$ cores $\rightarrow$ lose $1/(k/2)^2$

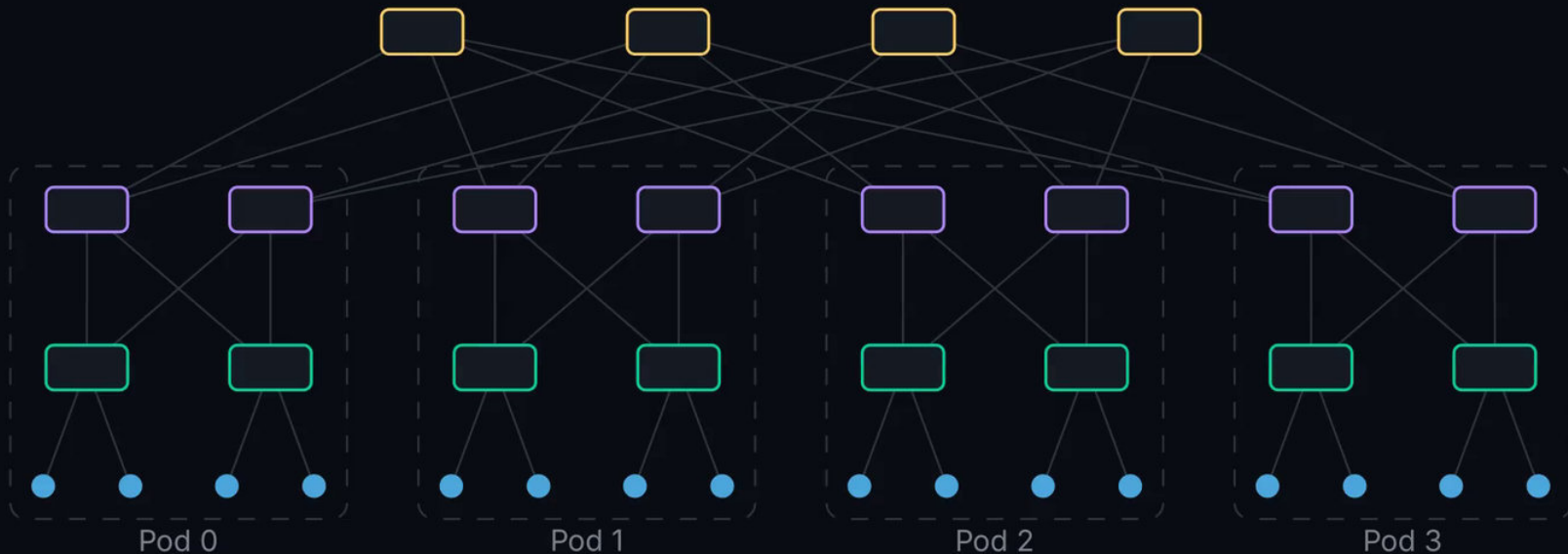


lose 1 of n Intermediates $\rightarrow$ lose $1/n$ (VL2 §4.1)

No idle twin, no cliff: 1:1 redundancy becomes $n:m$ (VL2 §3.4)

Both

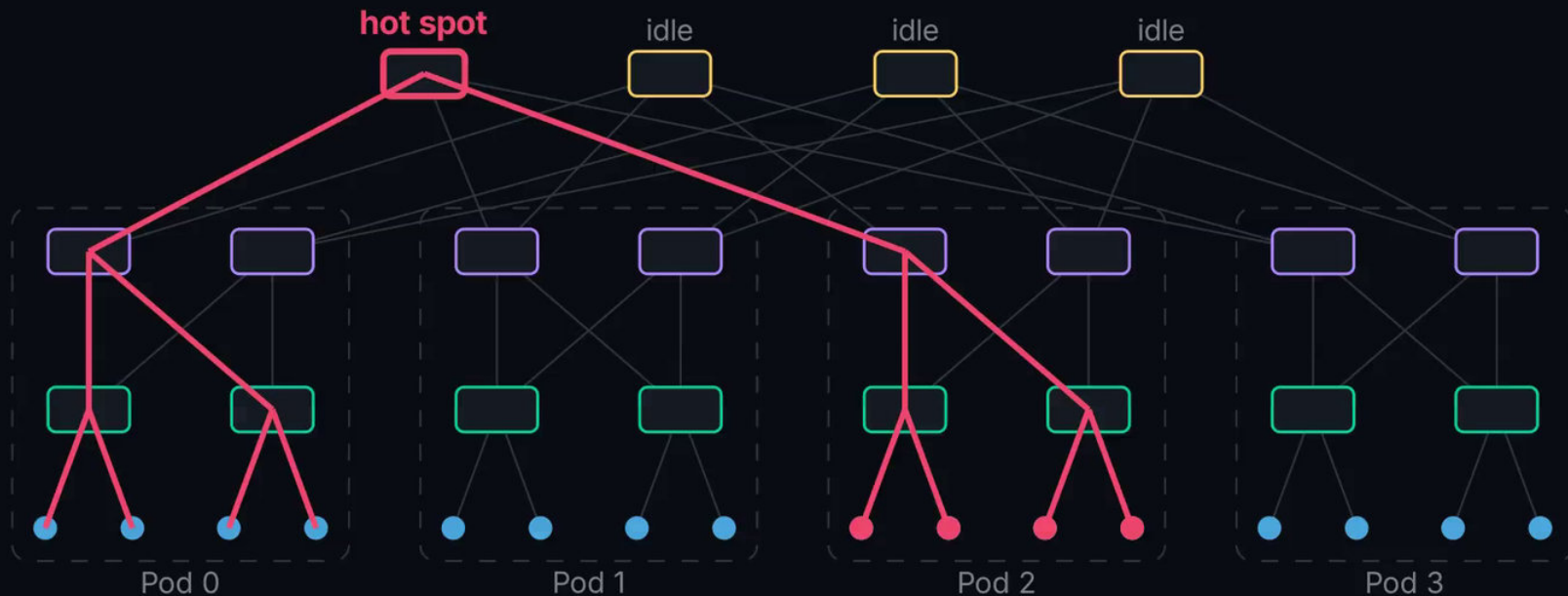
Act 5 · Many paths $\neq$ used paths



Plain shortest-path routing: one next hop per destination subnet

Both

Act 5 · Many paths $\neq$ used paths



Plain shortest-path routing: one next hop per destination subnet

ECMP: hash the 5-tuple, pick one of N equal paths

```
hash(src, dst, sport, dport, proto) mod N
```

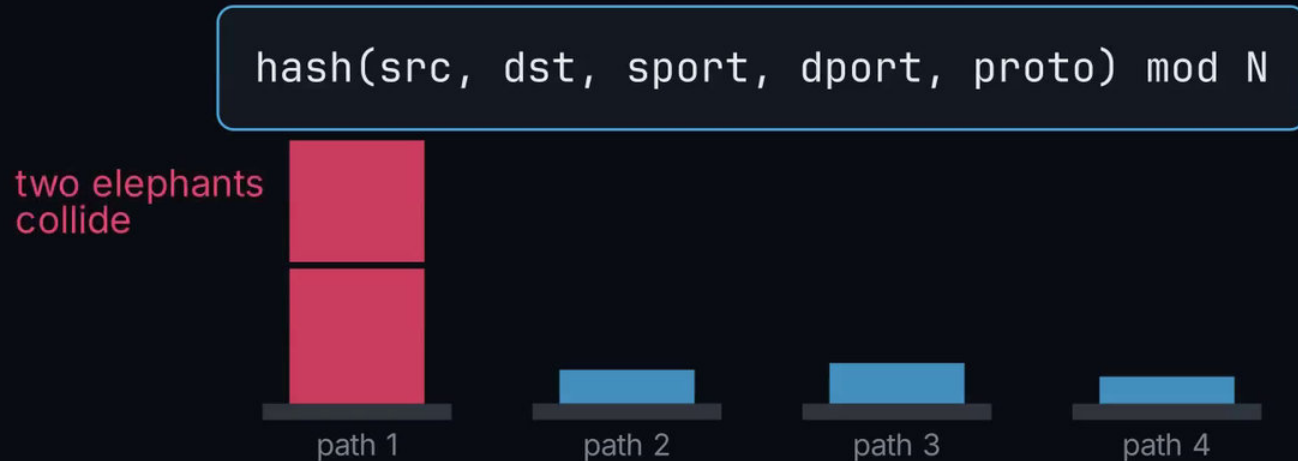
path 1

path 2

path 3

path 4

ECMP: hash the 5-tuple, pick one of N equal paths



ECMP: hash the 5-tuple, pick one of N equal paths



- × static: blind to flow size
- × switches: 8–16 ways (Fat-Tree §3.1)
- × 16-way ECMP, 256 just appearing (VL2 §4.2)

Both

Act 5 · Many paths $\neq$ used paths

Many equal-cost paths. Who spreads the traffic?

Many equal-cost paths. Who spreads the traffic?



Fat-Tree

can change the switch

→ the switches decide

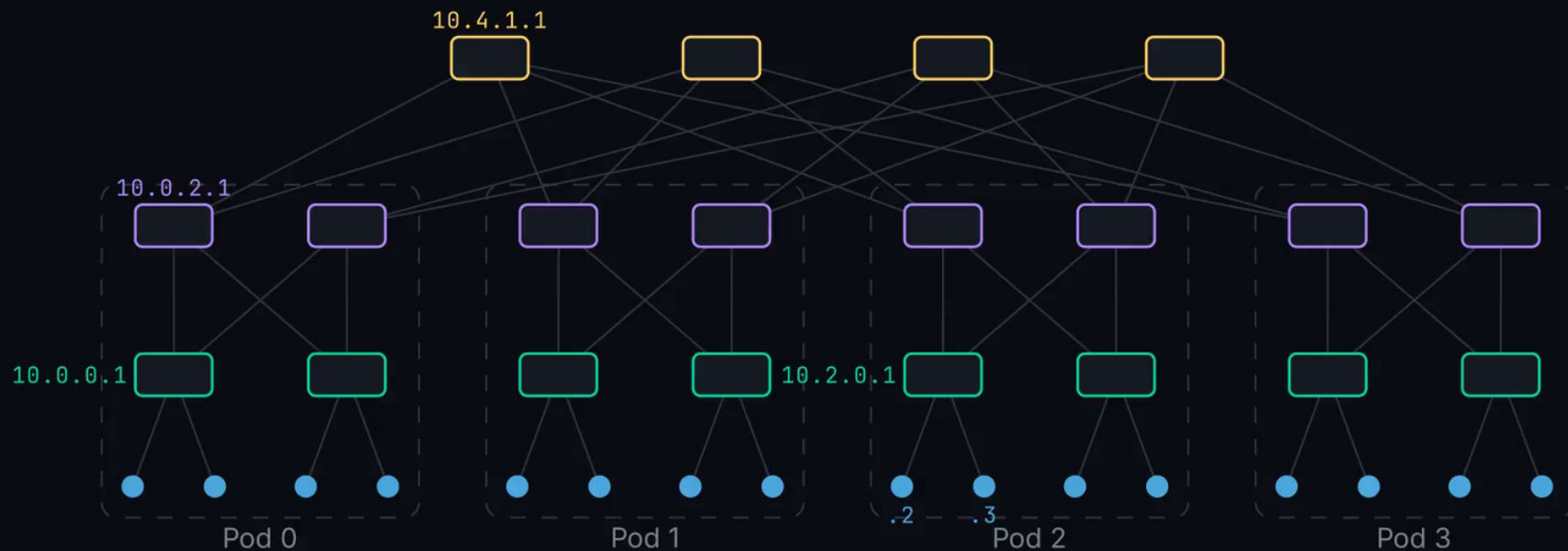
deterministic: $\text{path} = f(\text{dst host ID})$

VL2

can change the host

→ the hosts decide

random: Valiant Load Balancing



dst 10.2.0.3 = pod 2 · edge switch 0 · host ID 3

pod switch 10.pod.switch.1

host 10.pod.switch.ID

core 10.k.j.i

Two-level table at an aggregation switch (pod 2)

Prefix table (destination in my pod)

10.2.0.0/24 port 0 ↓

10.2.1.0/24 port 1 ↓

0.0.0.0/0 → suffix

Two-level table at an aggregation switch (pod 2)

Prefix table (destination in my pod)

10.2.0.0/24 port 0 ↓

10.2.1.0/24 port 1 ↓

0.0.0.0/0 → suffix

Suffix table (match host ID)

0.0.0.2/8 port 2 ↑

0.0.0.3/8 port 3 ↑

Two-level table at an aggregation switch (pod 2)

Prefix table (destination in my pod)

10.2.0.0/24 port 0 ↓

10.2.1.0/24 port 1 ↓

0.0.0.0/0 → suffix

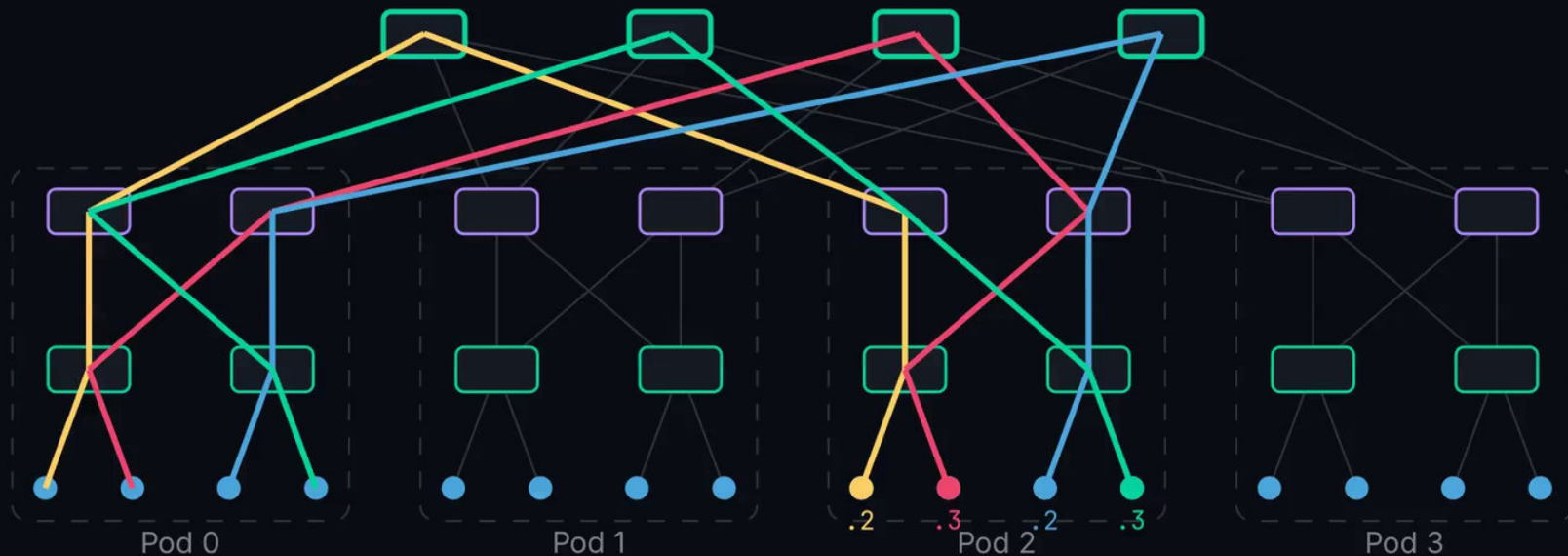
Suffix table (match host ID)

0.0.0.2/8 port 2 ↑

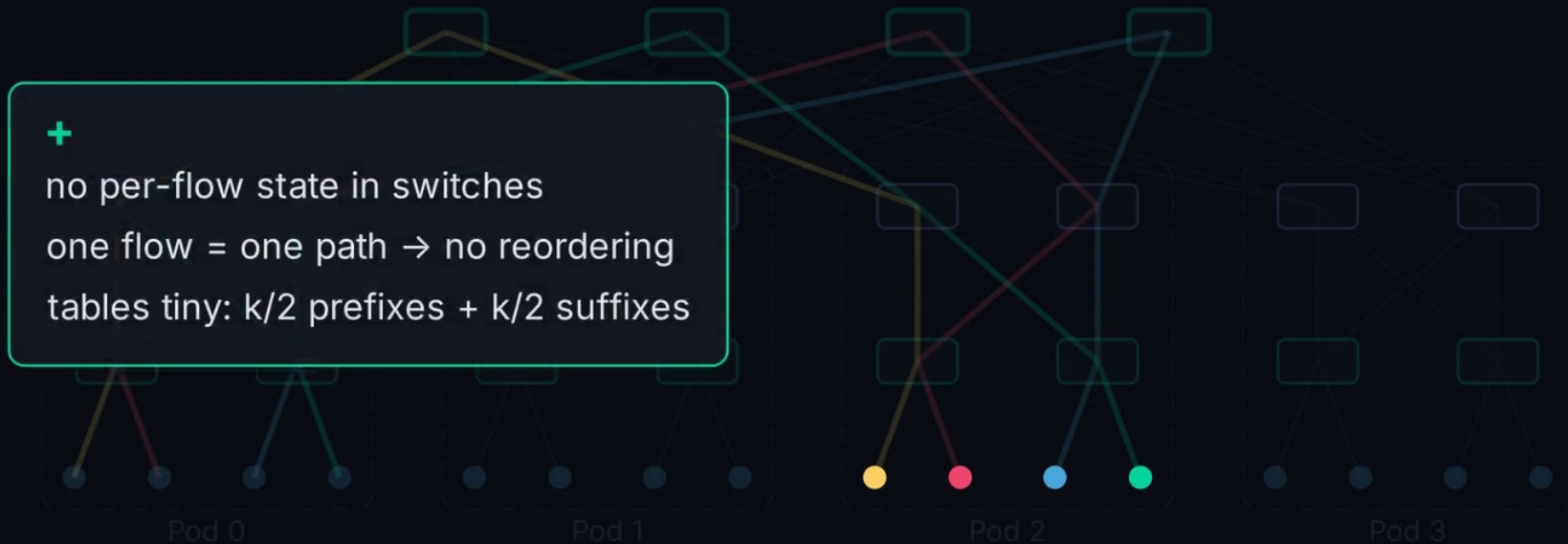
0.0.0.3/8 port 3 ↑

Going down: prefix → the one correct port

Going up: last byte of dst → spreads hosts over all uplinks



Same 4 flows as before, now one per core switch



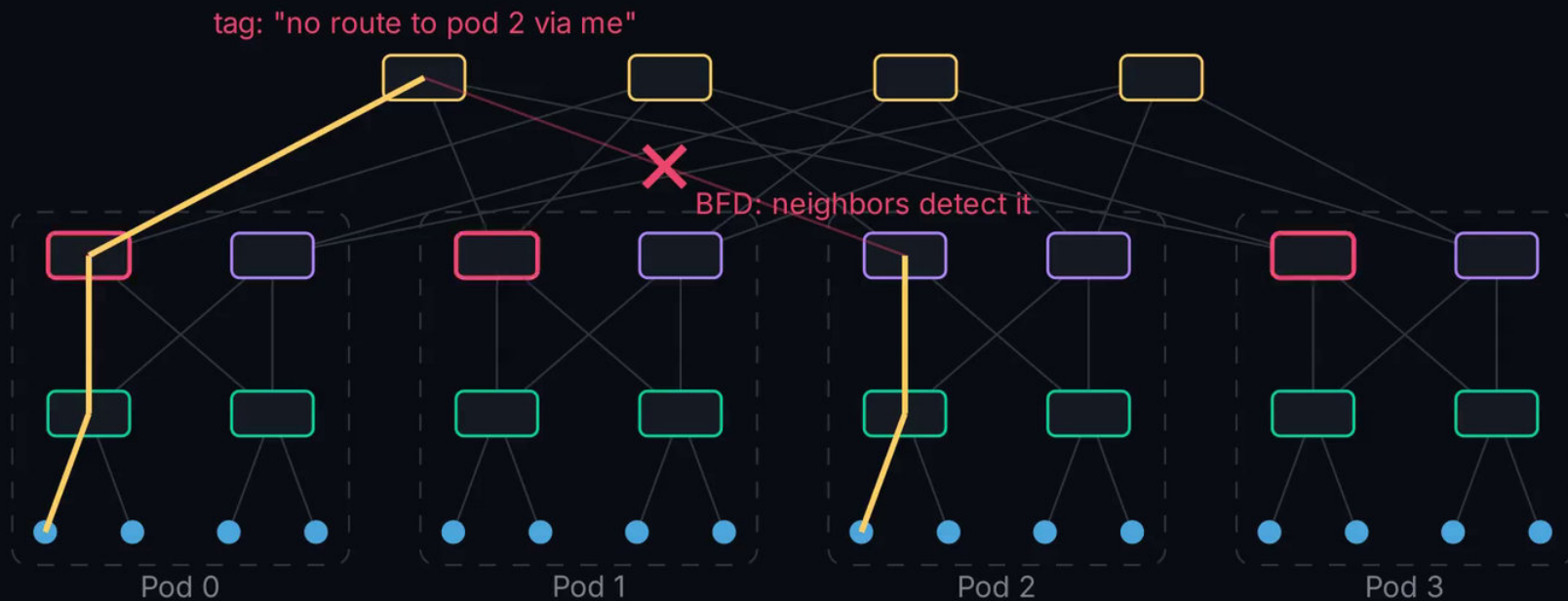
+

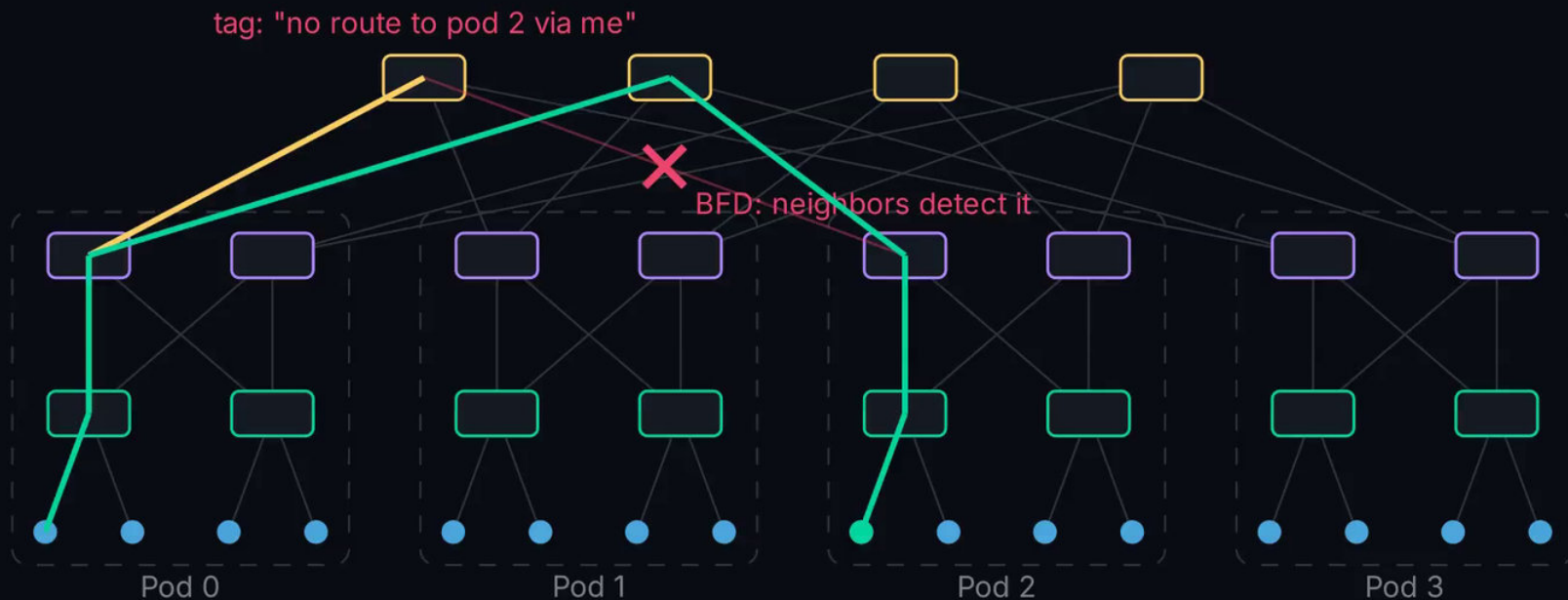
no per-flow state in switches
one flow = one path → no reordering
tables tiny: $k/2$ prefixes + $k/2$ suffixes

-

static: ignores flow sizes
two big flows, same host ID → collide
new lookup in the switch (NetFPGA: < 100 lines)







Fat-Tree §3.8: switches route around a failure one or two hops away

Everything lives in the switches

new lookup: two-level tables in every pod switch

BFD + failure broadcast in every switch

(and, for big flows, more switch logic: coming up in Act 8)

works only once vendors ship it

Everything lives in the switches

new lookup: two-level tables in every pod switch

BFD + failure broadcast in every switch

(and, for big flows, more switch logic: coming up in Act 8)

works only once vendors ship it

Microsoft can't wait for vendors. Can the hosts do this with stock switches?

Production data center measurements (VL2 §3)

Production data center measurements (VL2 §3)

flows < 100 MB

**99% of flows**

bytes in 100 MB–1 GB flows

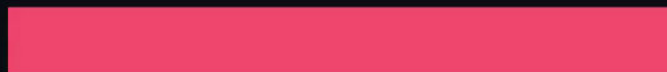
**> 90% of bytes**

Production data center measurements (VL2 §3)

flows < 100 MB

**99% of flows**

bytes in 100 MB–1 GB flows

**> 90% of bytes**

Concurrent flows per server:

~10 more than 50% of the time · > 80 at least 5% of the time

Traffic matrices jump around



one block = one 100 s traffic matrix · color = which cluster it fits

Traffic matrices jump around



one block = one 100 s traffic matrix · color = which cluster it fits

Even 50–60 representative matrices leave ~60% fitting error

Pattern changes quickly and unpredictably

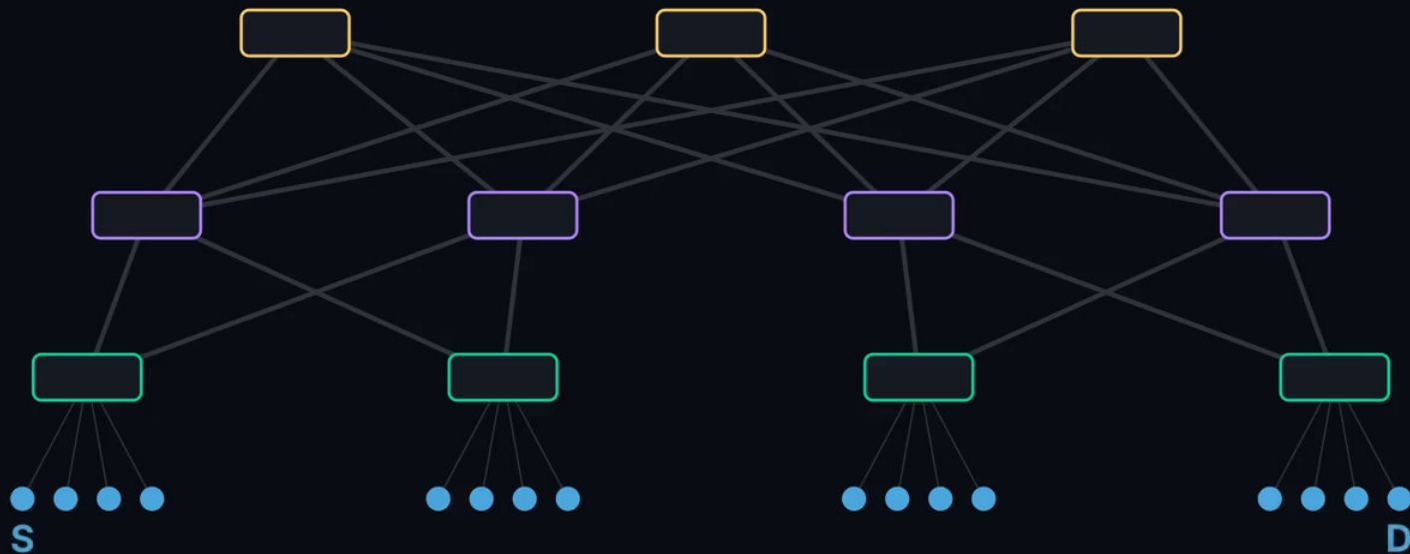
→ optimizing routes for a few matrices won't work

Can't predict the traffic?

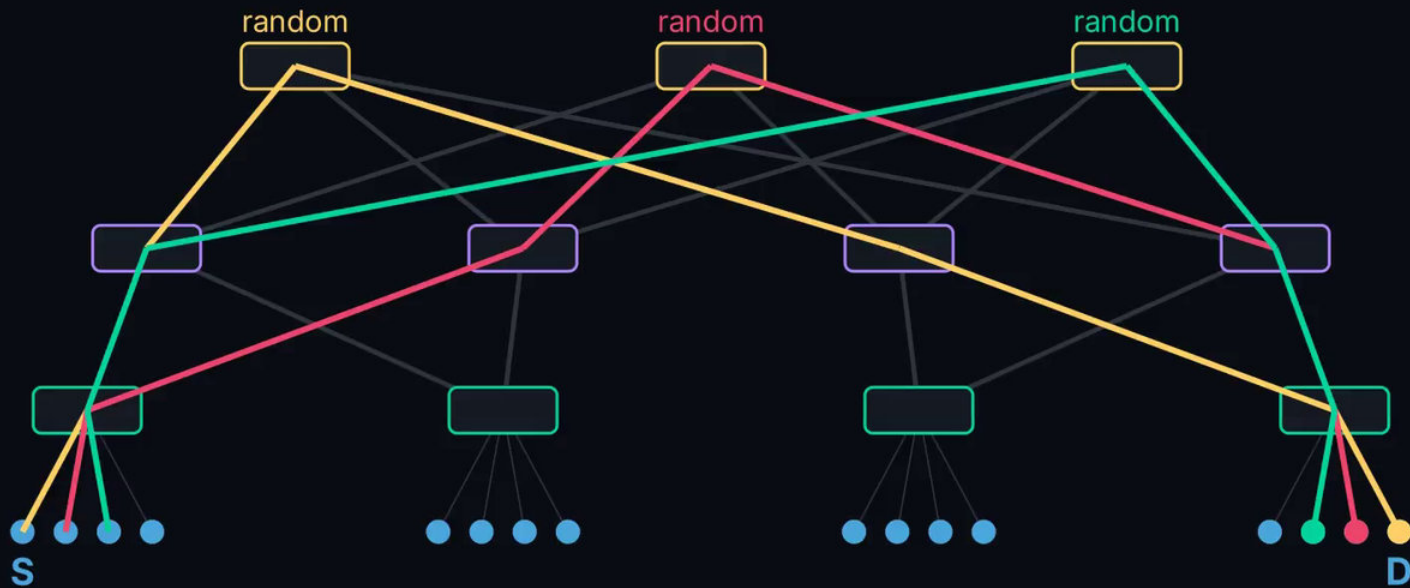
Can't predict the traffic?

Don't try. Spread it randomly.

Valiant Load Balancing: oblivious routing, no traffic engineering



Each flow: up to a random Intermediate, then down to the destination



Each flow: up to a random Intermediate, then down to the destination

Why random bouncing works: the hose model

Each server sends $\leq 1G$ and receives $\leq 1G$ (its NIC)

Why random bouncing works: the hose model

Each server sends $\leq 1G$ and receives $\leq 1G$ (its NIC)

Split every flow evenly over the n Intermediates

→ every Intermediate link carries $1/n$ of any traffic matrix

→ no hot spots, whatever the traffic matrix

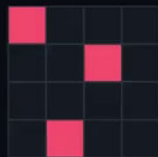
Why random bouncing works: the hose model

Each server sends $\leq 1G$ and receives $\leq 1G$ (its NIC)

Split every flow evenly over the n Intermediates

→ every Intermediate link carries $1/n$ of any traffic matrix

→ no hot spots, whatever the traffic matrix



or



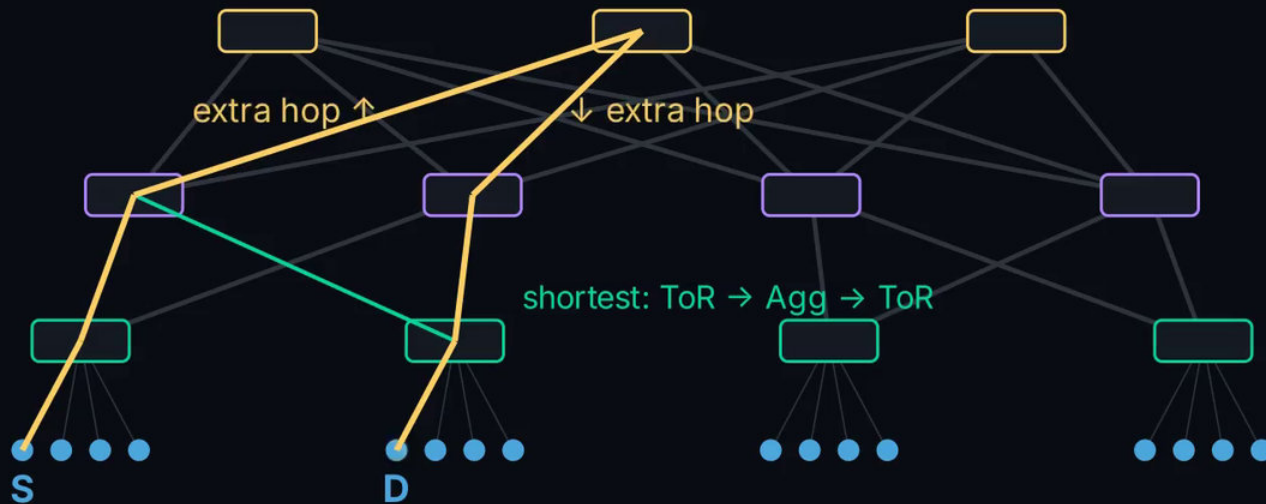
same uniform load

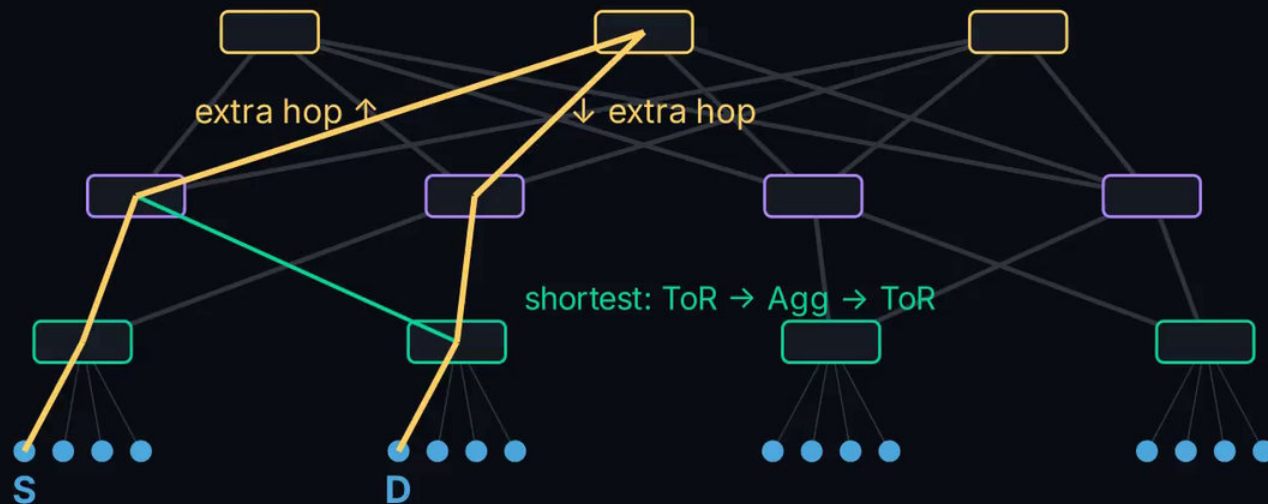
Theory wants

- (a) random split per packet
- (b) traffic obeys the hose model

VL2 does

- (a) random per flow → no reordering
- (b) relies on TCP to keep senders in line

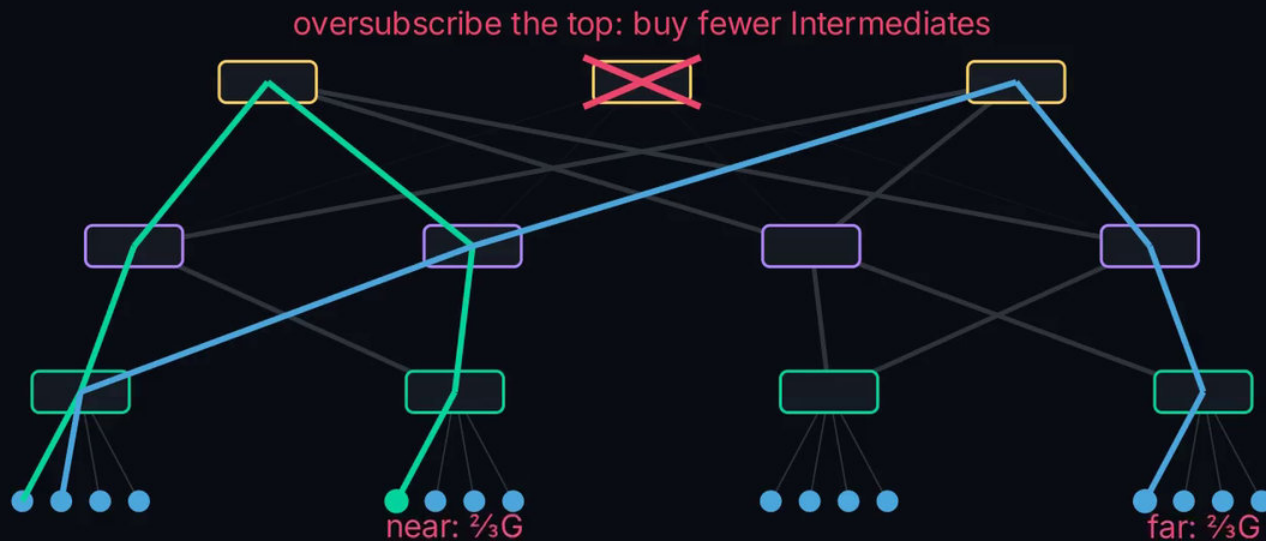


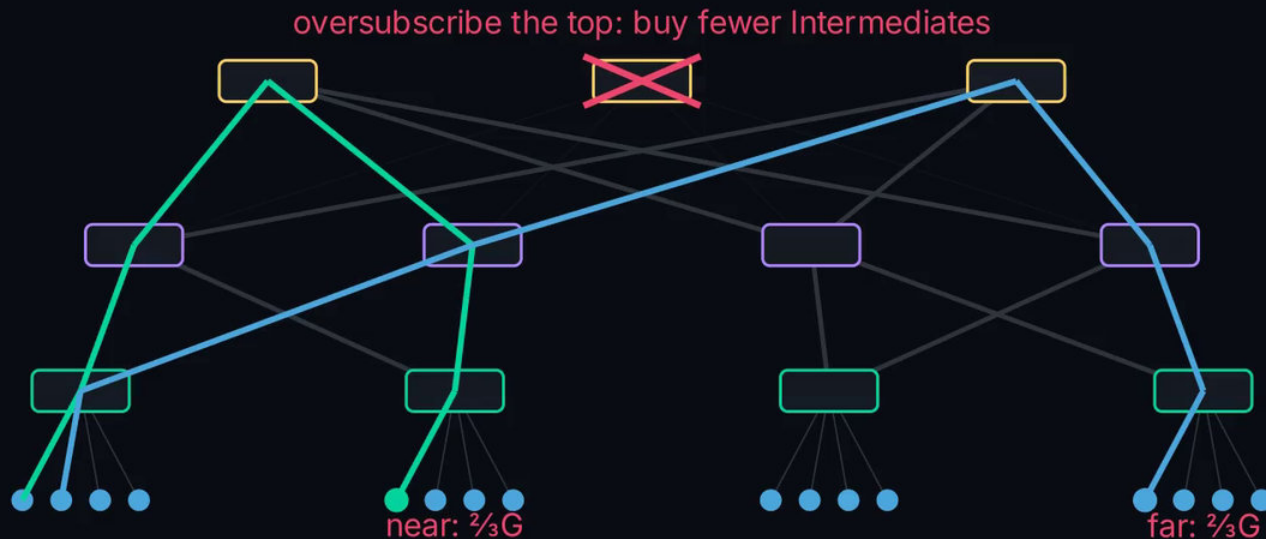


price: microseconds of fiber + one switch, inside one building

capacity: Agg $\leftrightarrow$ Int is a full mesh of 10G, sized for exactly this (hose model)

in return: uniform $1/n$ load on every top link, for any traffic matrix

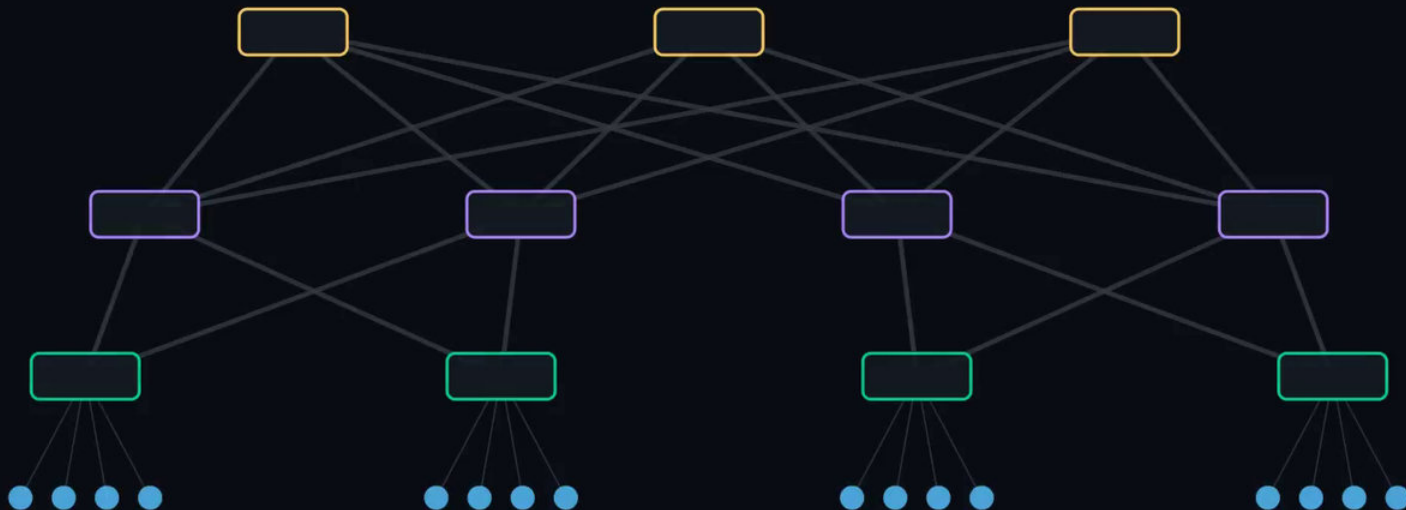




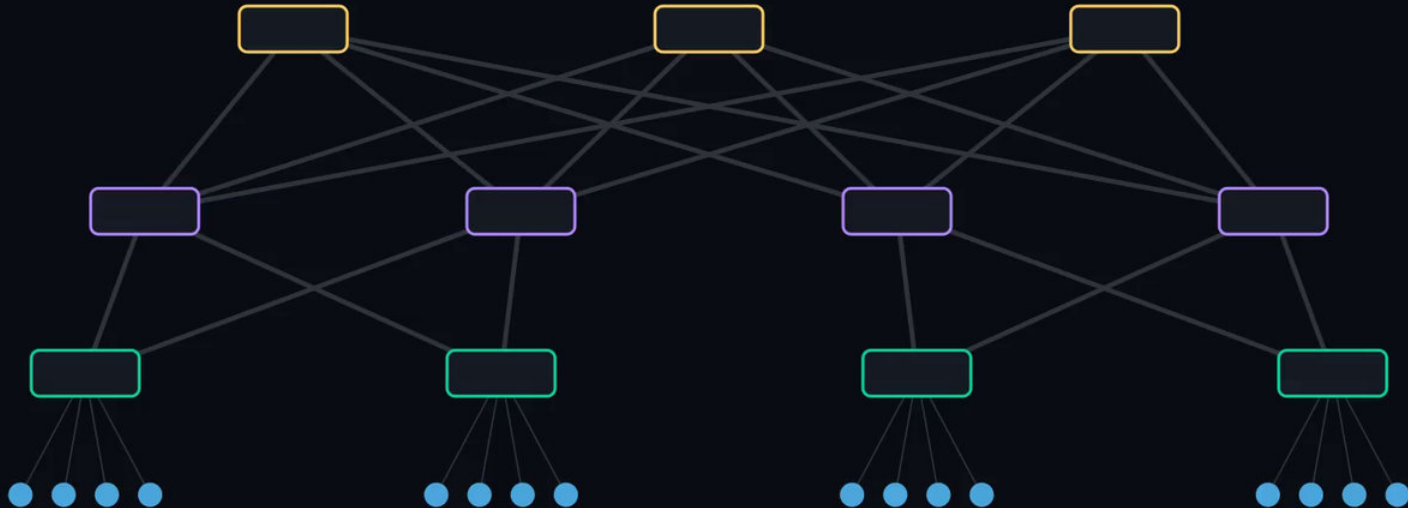
VLB drops locality on purpose: uniform capacity = agility (page 6)

VL2 §6: a 1:23 VL2 costs 70% less, "probably not worth the loss in performance"
and a 1:1 VL2 already costs about the same as today's 1:240 network

Rule: switches stay stock. What does a stock switch already do?

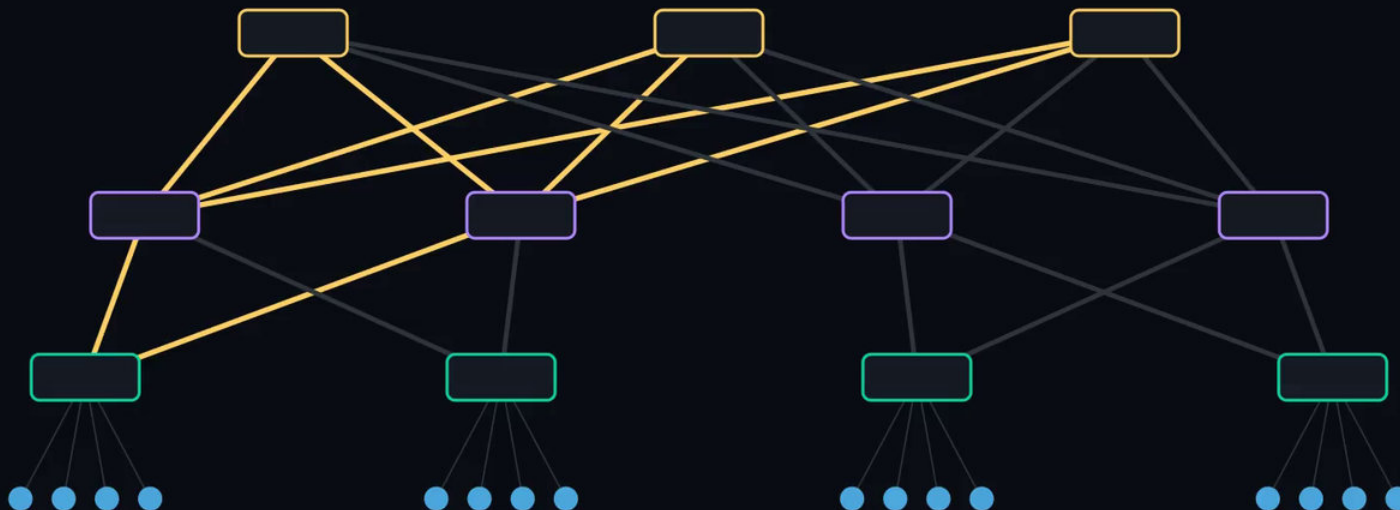


Rule: switches stay stock. What does a stock switch already do?



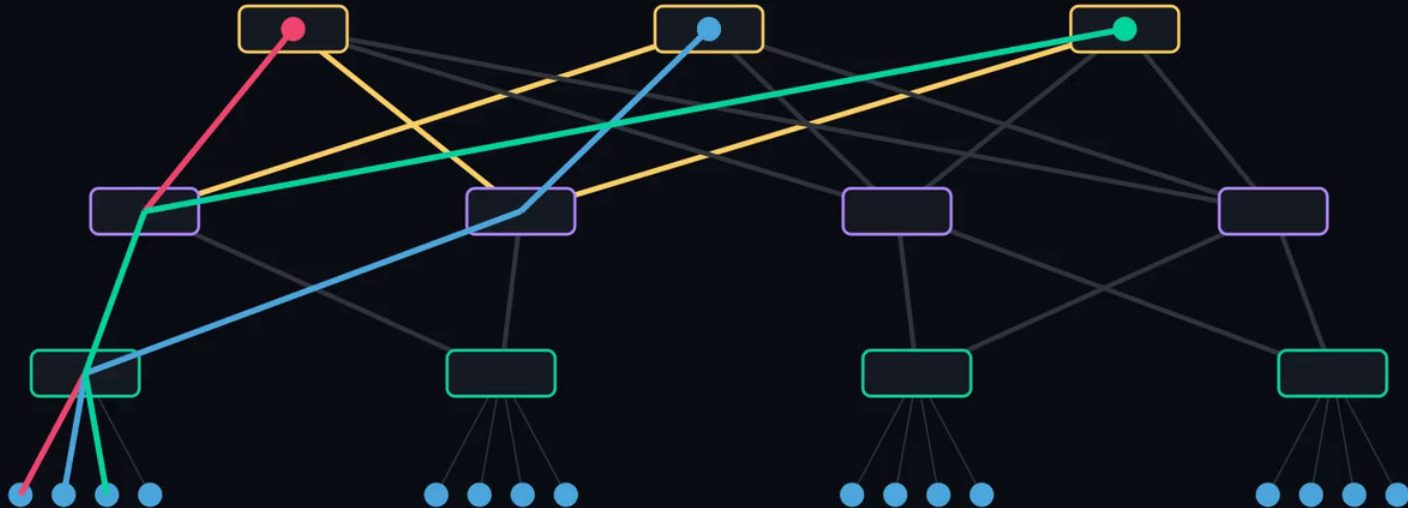
OSPF (link-state): flood link state → same map everywhere → shortest paths

Rule: switches stay stock. What does a stock switch already do?



equal-cost paths: OSPF installs every one of them as a next hop

Rule: switches stay stock. What does a stock switch already do?



layer 3 → no spanning tree → every link is used (vs. page 11)

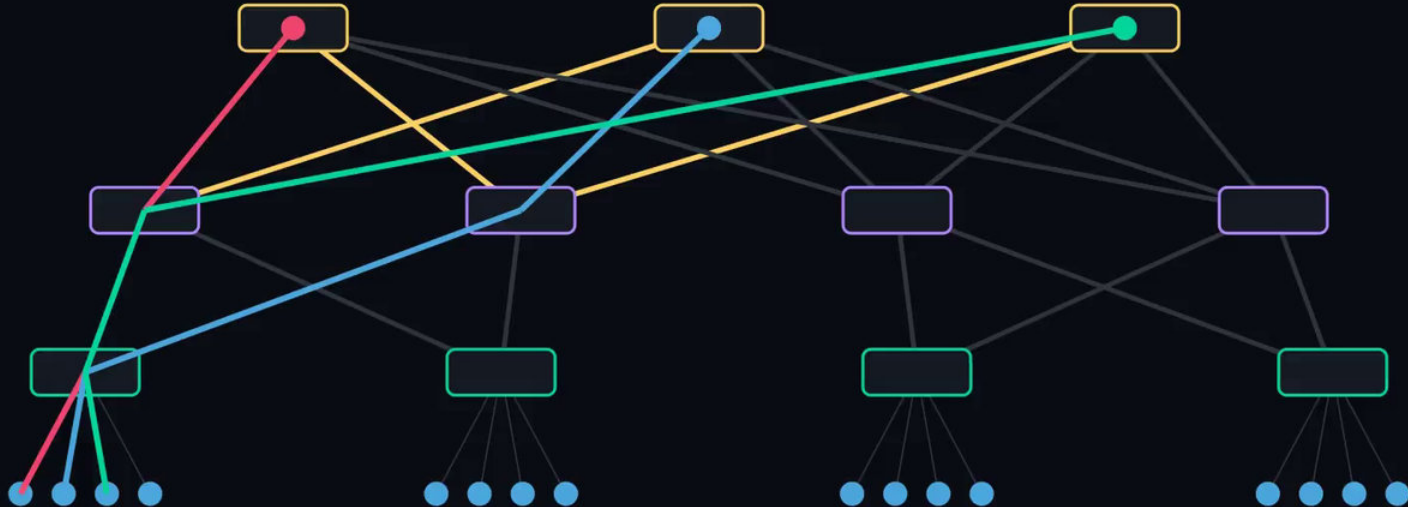
Rule: switches stay stock. What does a stock switch already do?

OSPF

ECMP

IP anycast

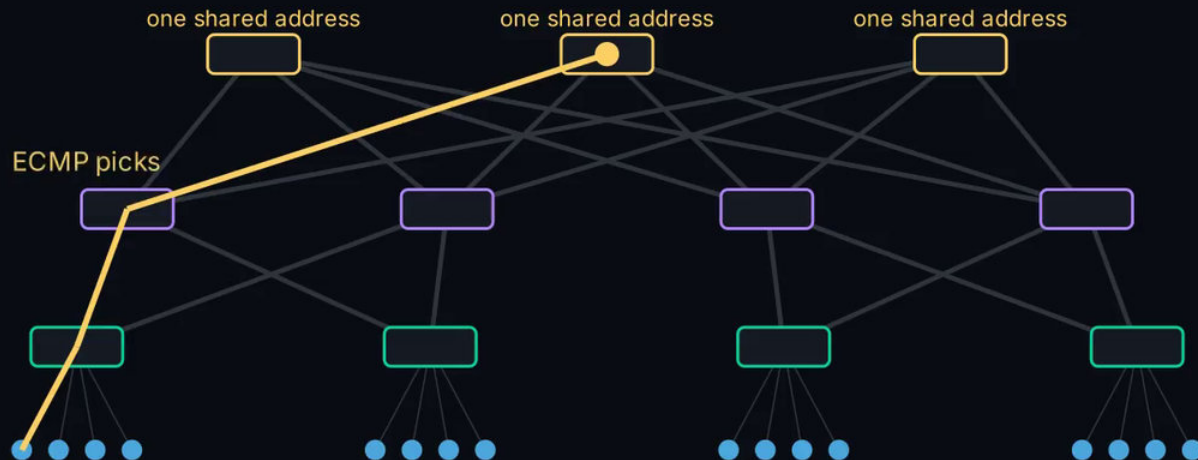
IP-in-IP decap



layer 3 → no spanning tree → every link is used (vs. page 11)

VL2 agent

shim in the host OS



IP-in-IP: two extra headers

dst: Int anycast 10.1.1.1

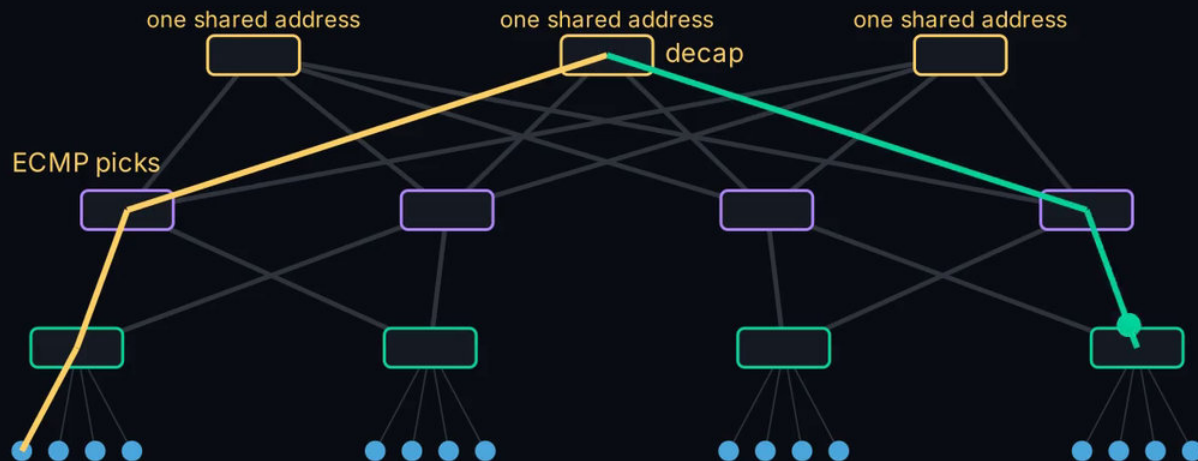
dst: D's ToR 10.0.0.6

S → D (untouched)

payload

VL2 agent

shim in the host OS



IP-in-IP: two extra headers

dst: D's ToR 10.0.0.6

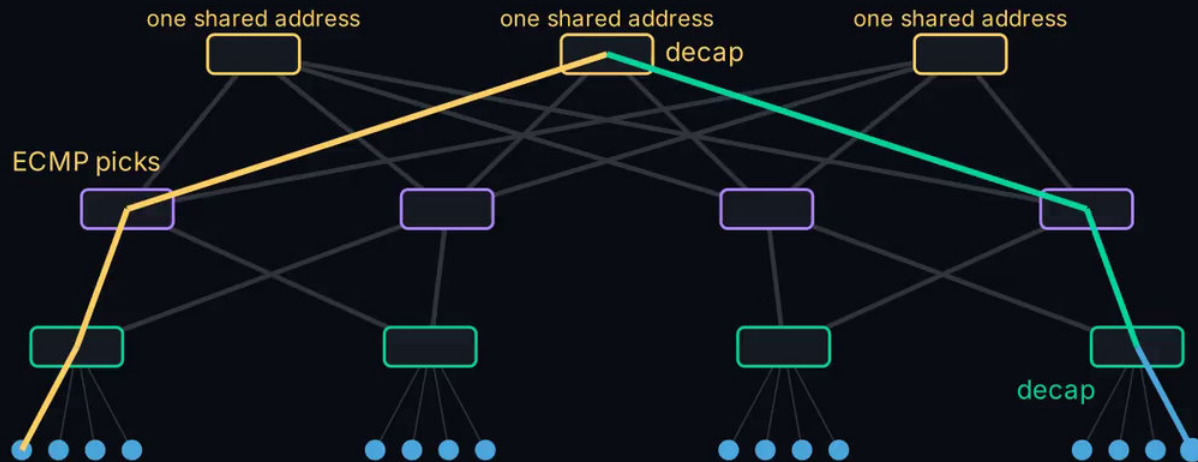
S → D (untouched)

payload

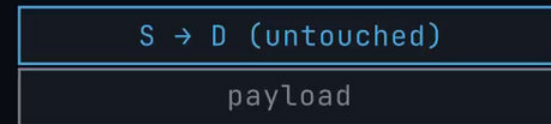
VL2 Act 7 · The hosts decide, on stock switches

VL2 agent

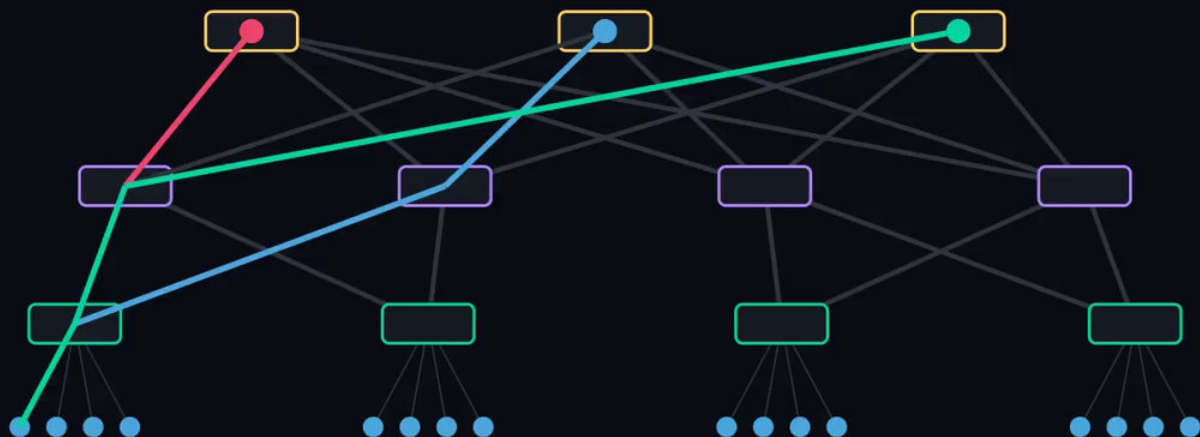
shim in the host OS



IP-in-IP: two extra headers



how the agent learns D's ToR (names vs. locations): later, with NVP
each switch does one stock thing: ECMP up · decap at Int · OSPF down · decap at ToR

Snag: cheap switches can't see inside the wrapping`dst: Int · src: H(5-tuple)``dst: D's ToR``S:port → D:port`

ports hidden from ECMP

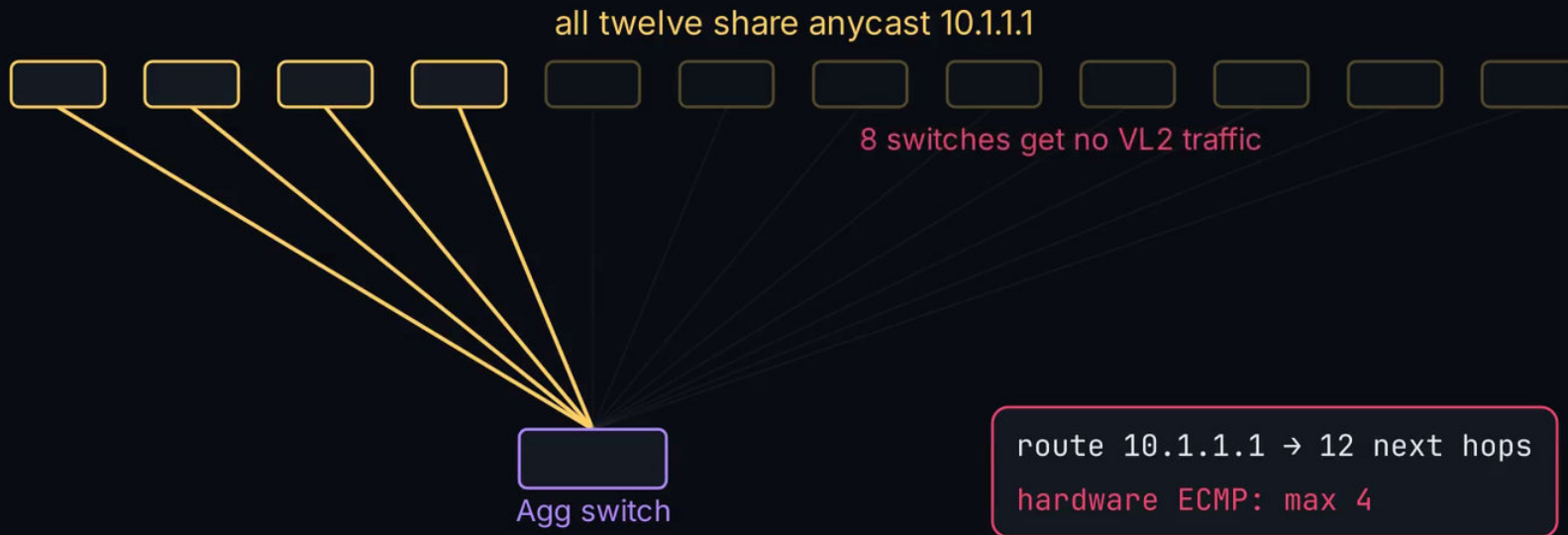
Snag 2: a switch hashes over at most 16 next hops

drawn with a 4-way limit so it fits on screen; real switches: 16 (256 just arriving in 2009)



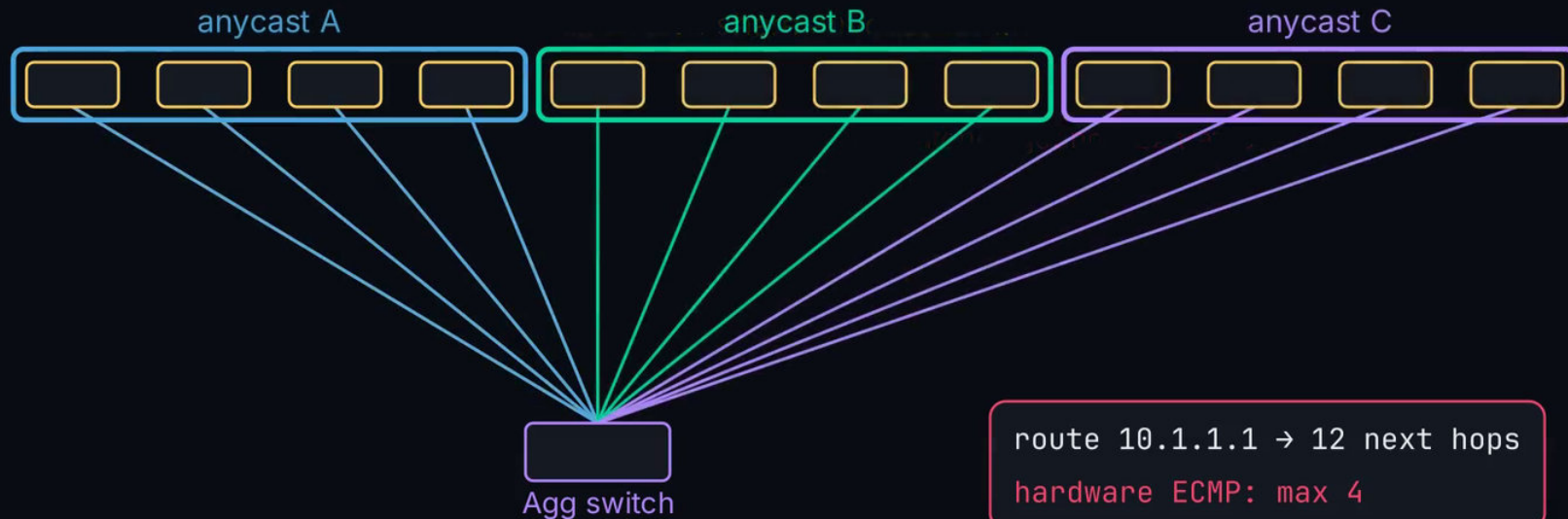
Snag 2: a switch hashes over at most 16 next hops

drawn with a 4-way limit so it fits on screen; real switches: 16 (256 just arriving in 2009)



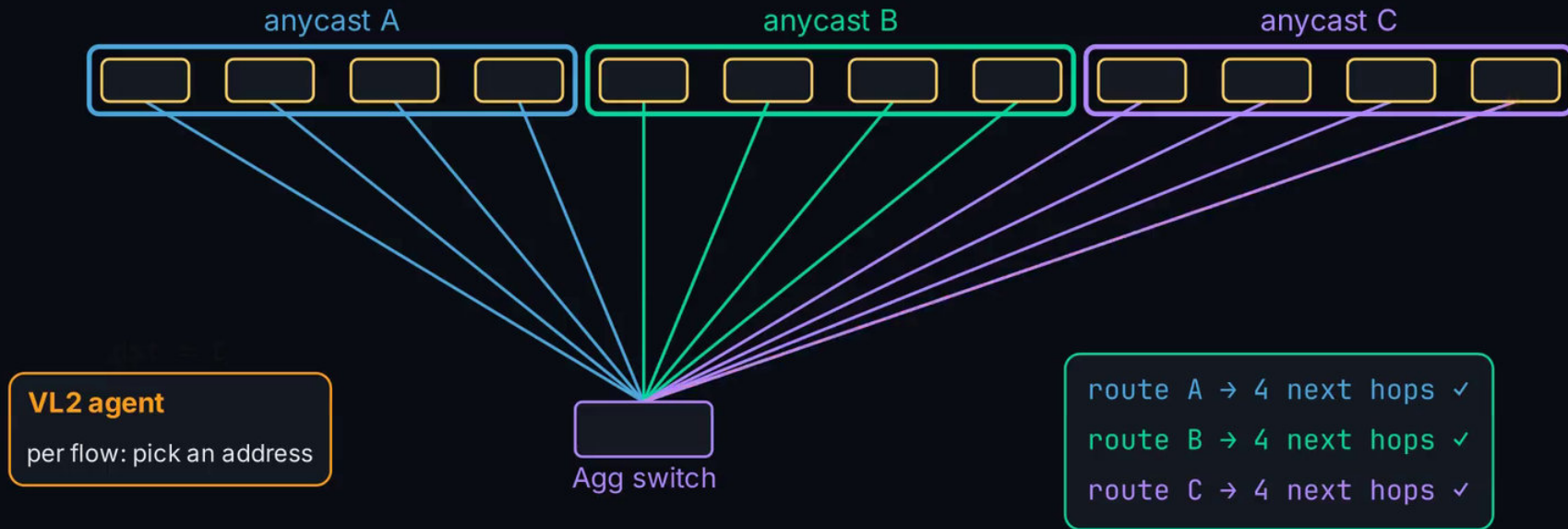
Snag 2: a switch hashes over at most 16 next hops

drawn with a 4-way limit so it fits on screen; real switches: 16 (256 just arriving in 2009)



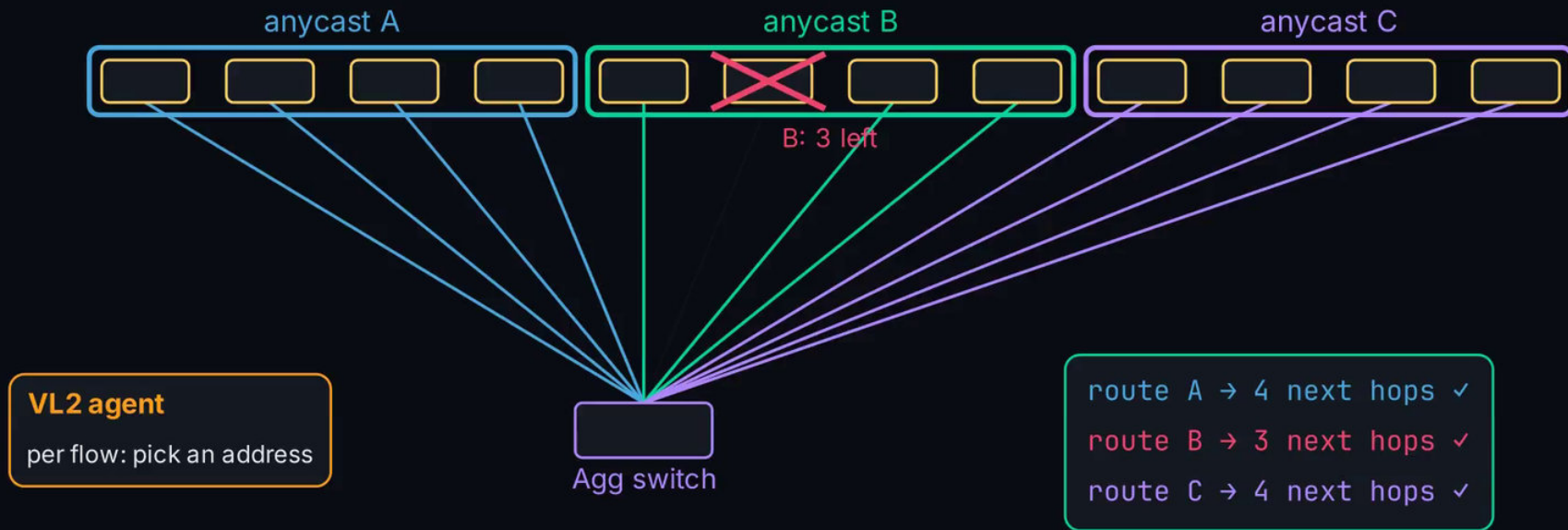
Snag 2: a switch hashes over at most 16 next hops

drawn with a 4-way limit so it fits on screen; real switches: 16 (256 just arriving in 2009)

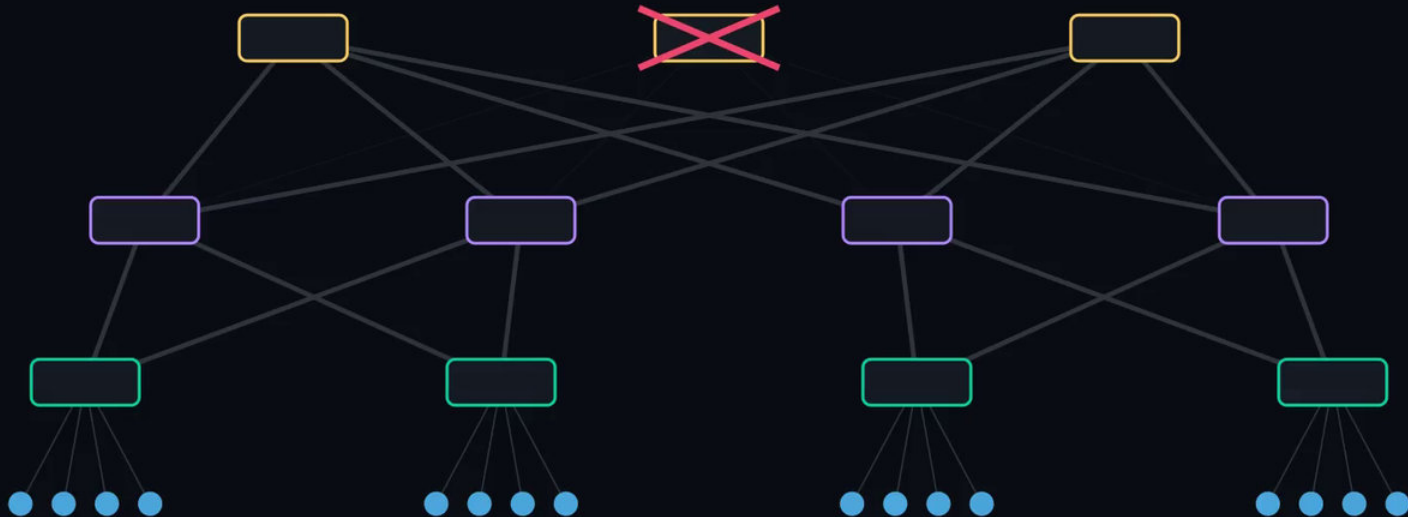


Snag 2: a switch hashes over at most 16 next hops

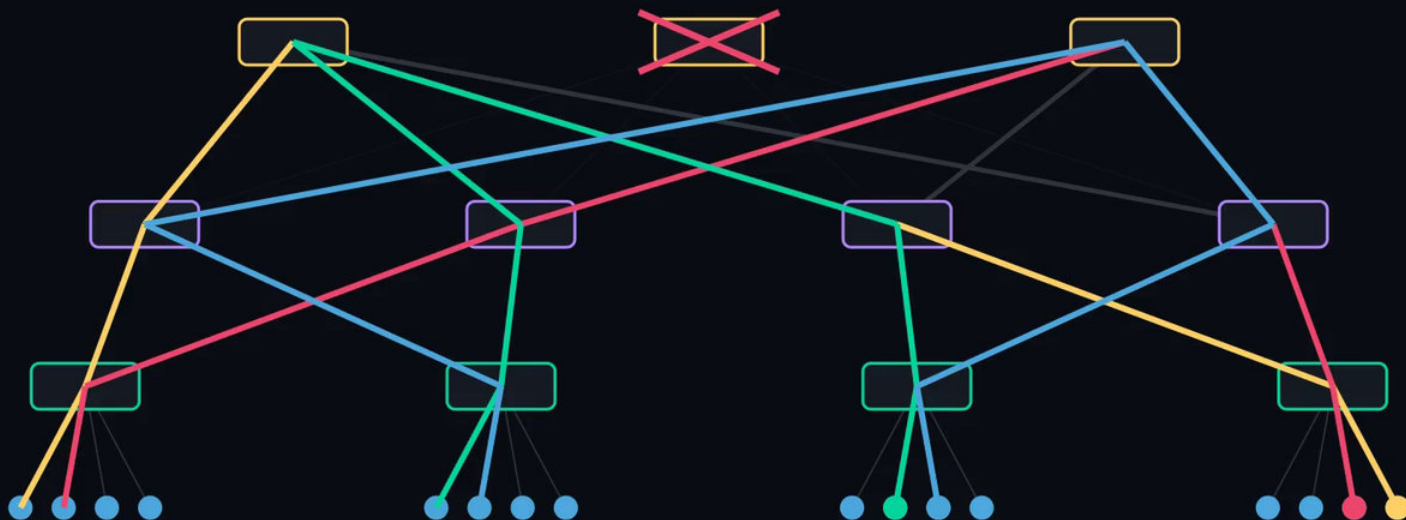
drawn with a 4-way limit so it fits on screen; real switches: 16 (256 just arriving in 2009)



OSPF reconverges · ECMP drops it from the anycast set



OSPF reconverges · ECMP drops it from the anycast set



capacity drops by $1/n$; no host is told (VL2 §4.2, §5.4)

Both

Act 8 · Load balancing: who notices the elephant?

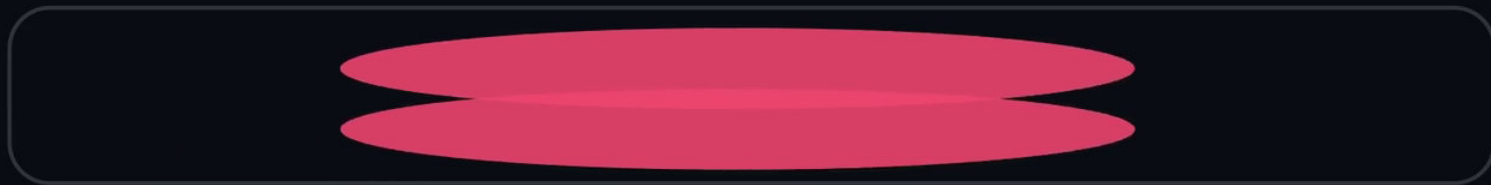
Not all flows are equal



Both

Act 8 · Load balancing: who notices the elephant?

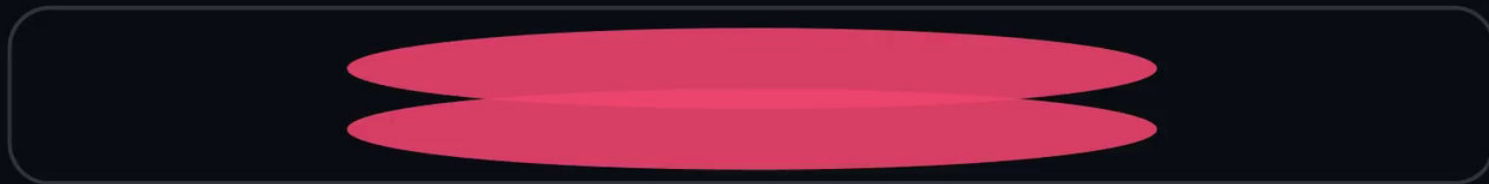
Not all flows are equal



Both

Act 8 · Load balancing: who notices the elephant?

Not all flows are equal



share of flows



99% of flows < 100 MB

share of bytes



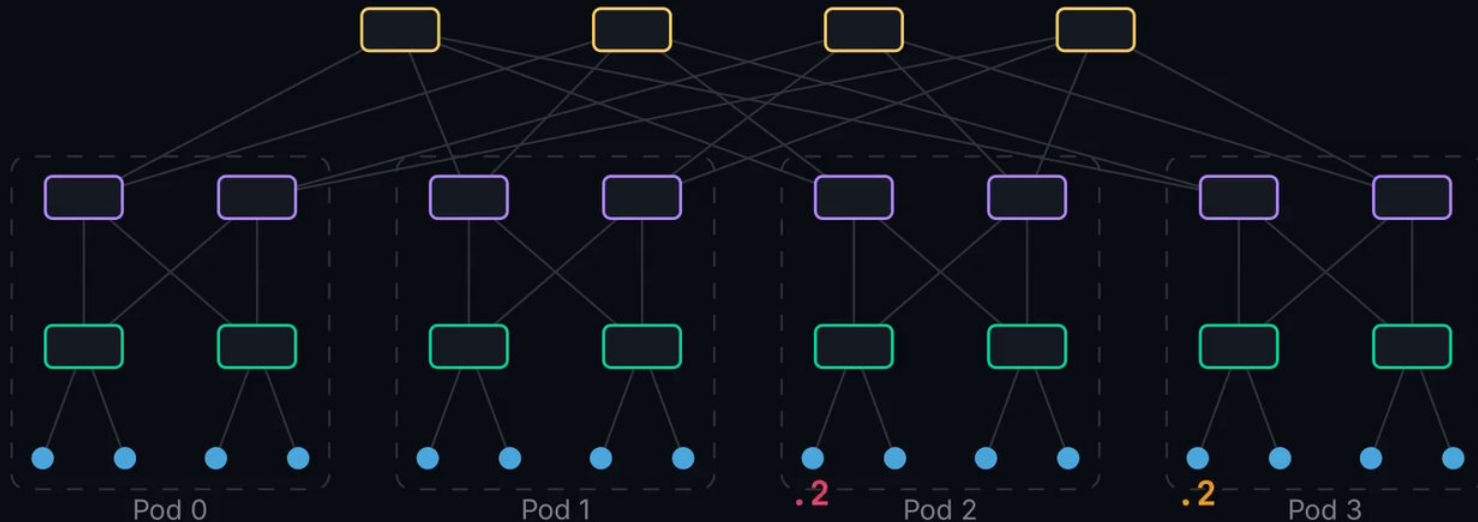
> 90% of bytes in 100 MB – 1 GB flows

VL2 §3, measured (page 28)

Both

Act 8 · Load balancing: who notices the elephant?

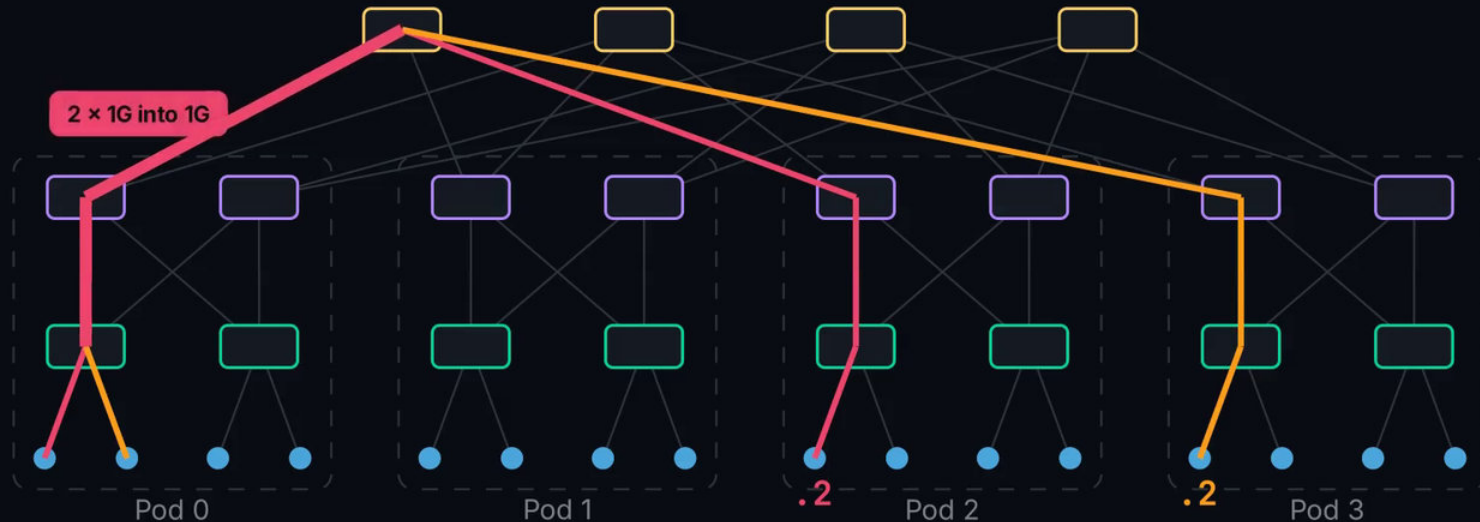
At UCSD: two elephants, one link



Both

Act 8 · Load balancing: who notices the elephant?

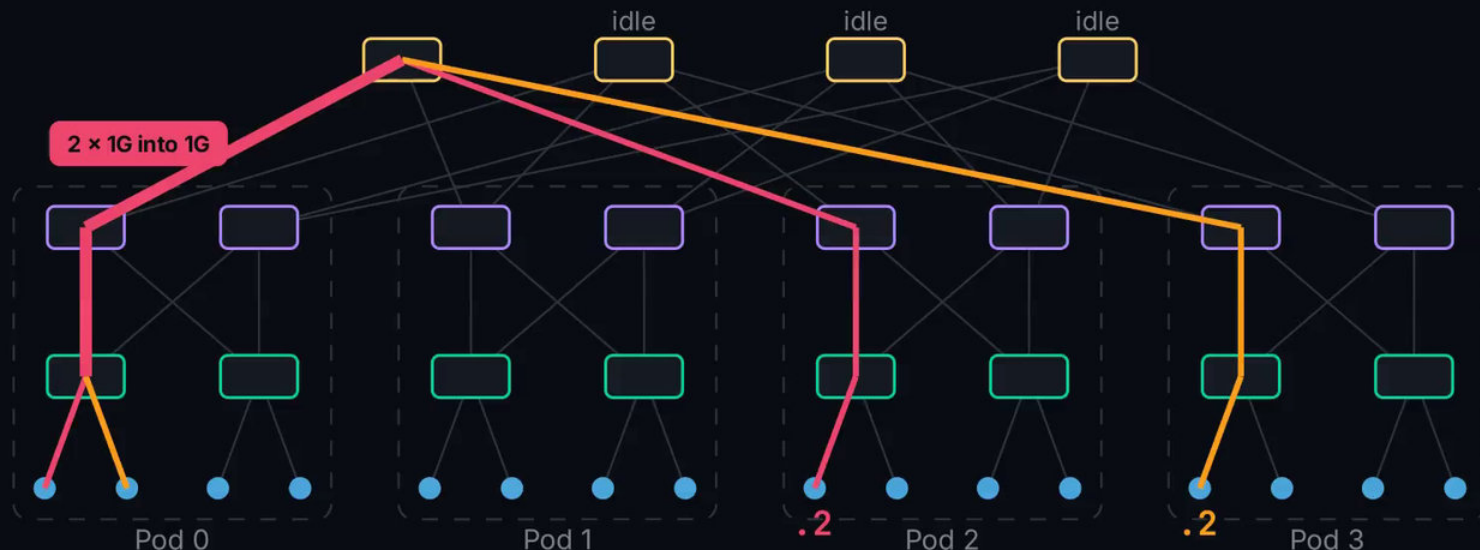
At UCSD: two elephants, one link



Both

Act 8 · Load balancing: who notices the elephant?

At UCSD: two elephants, one link

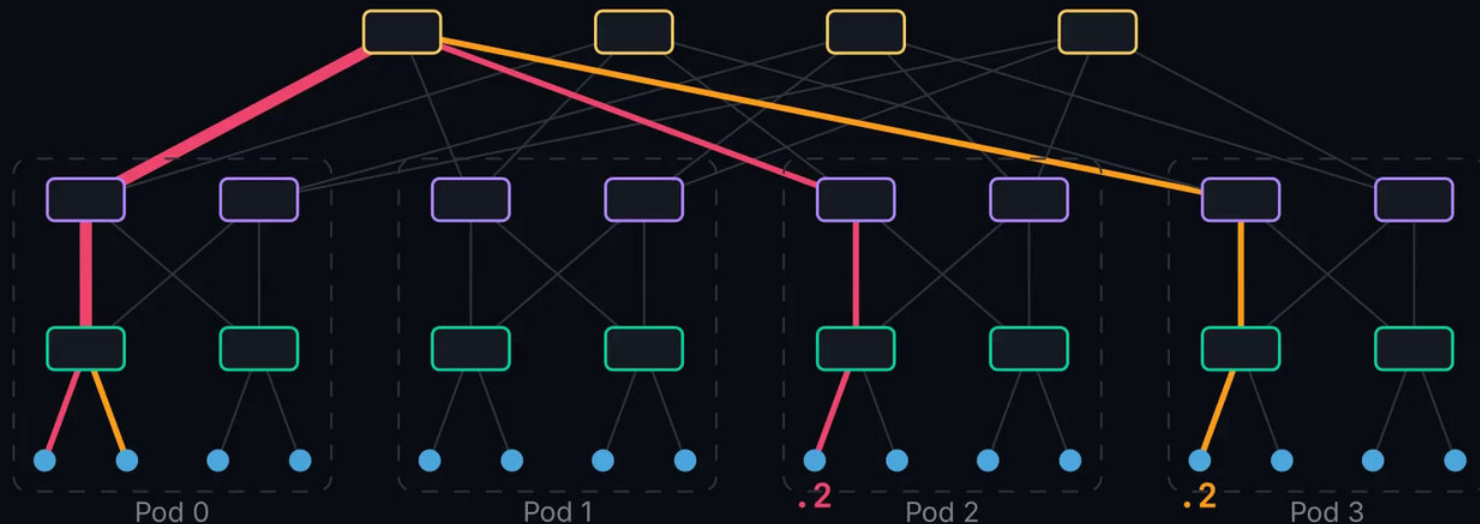


every Fat-Tree link is 1G, the same as a server's NIC → 500 Mbps each
the table is static: it never looks at flow size

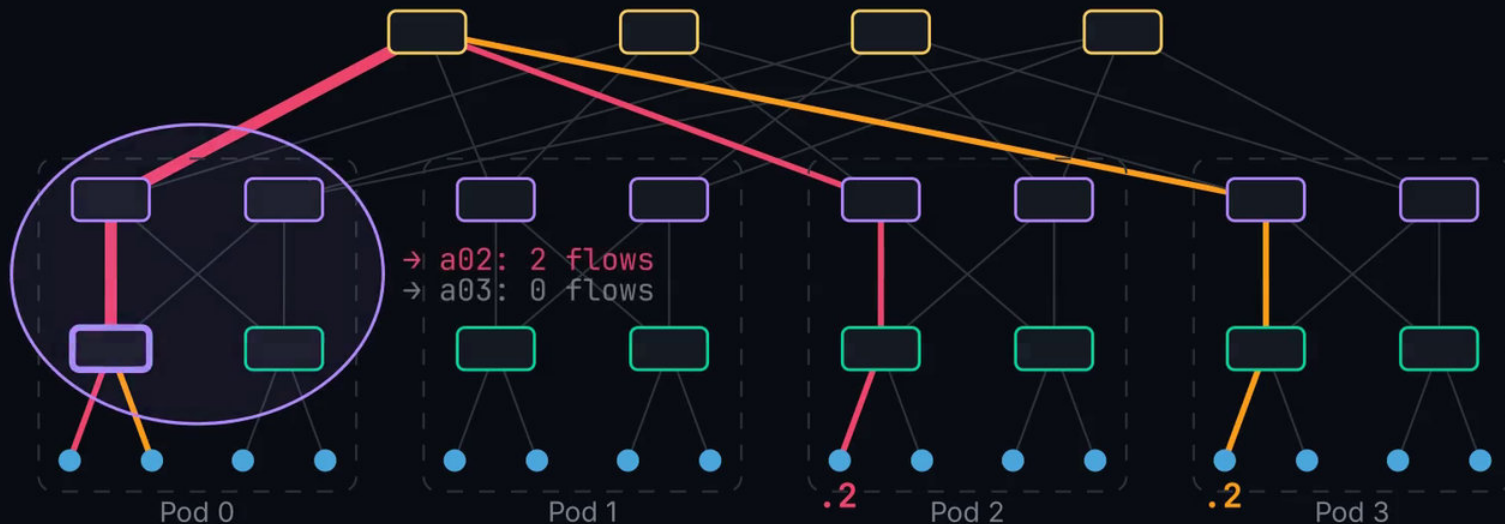
Both

Act 8 · Load balancing: who notices the elephant?

First idea: the switch that sees it fixes it



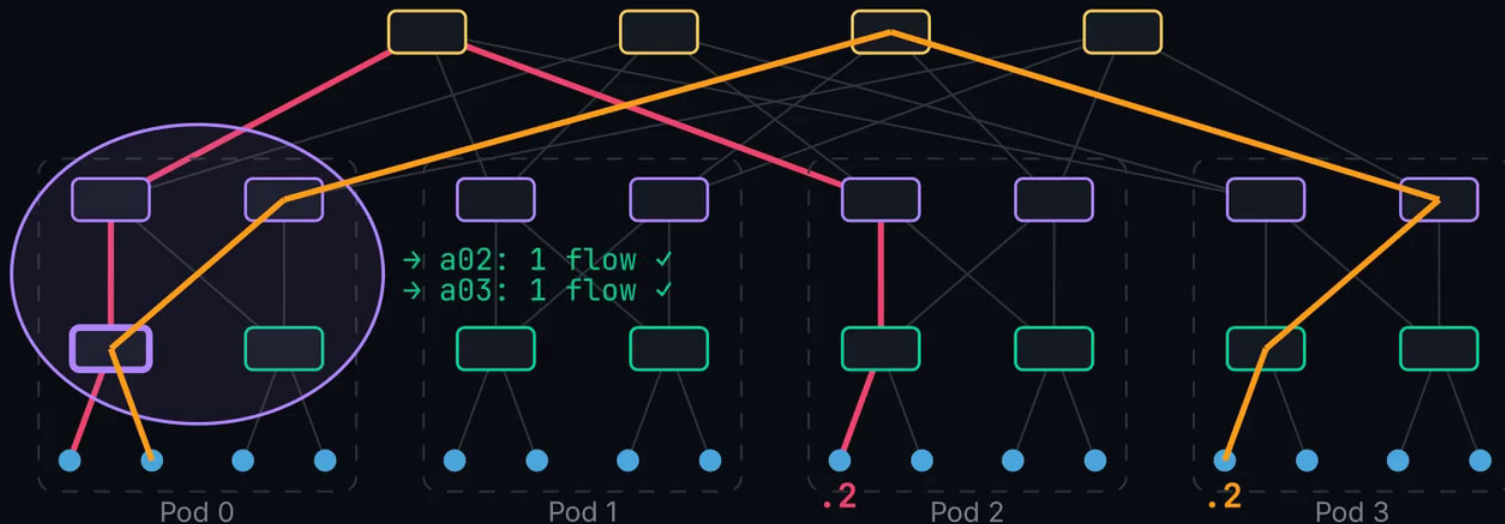
First idea: the switch that sees it fixes it



flow classification (§3.6)

- ① remember each flow's port → its packets stay in order
- ② every few seconds, move as few flows as possible to even out its uplinks

First idea: the switch that sees it fixes it

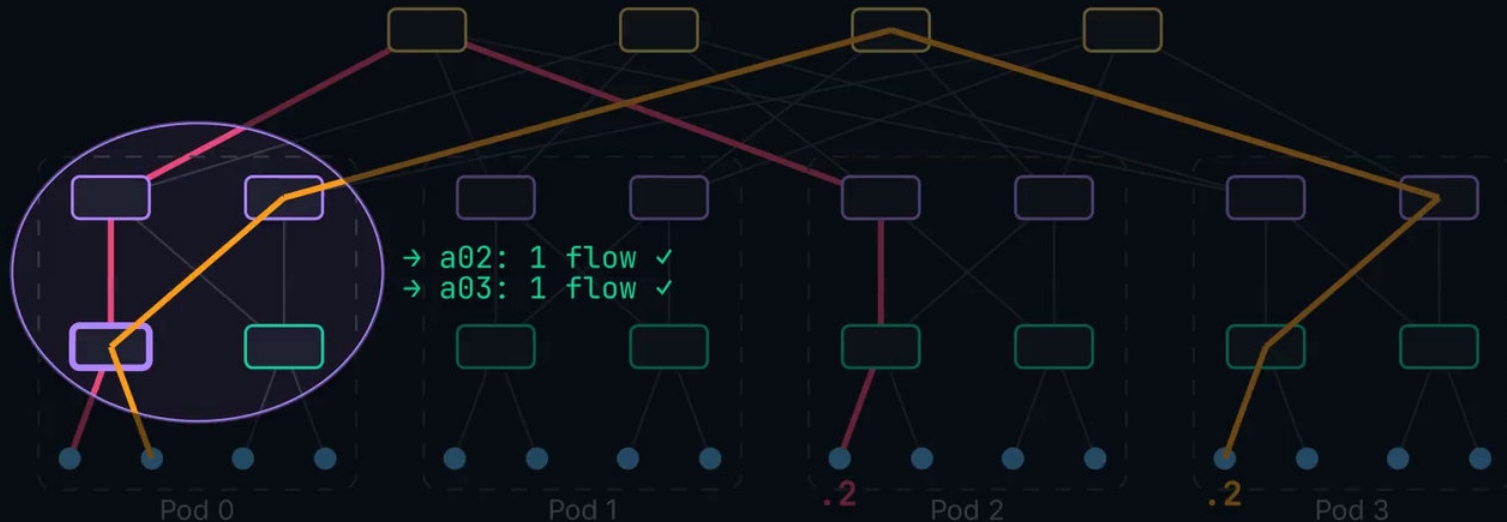


flow classification (§3.6)

- ① remember each flow's port → its packets stay in order
- ② every few seconds, move as few flows as possible to even out its uplinks

First idea: the switch that sees it fixes it

but this circle is everything it can see



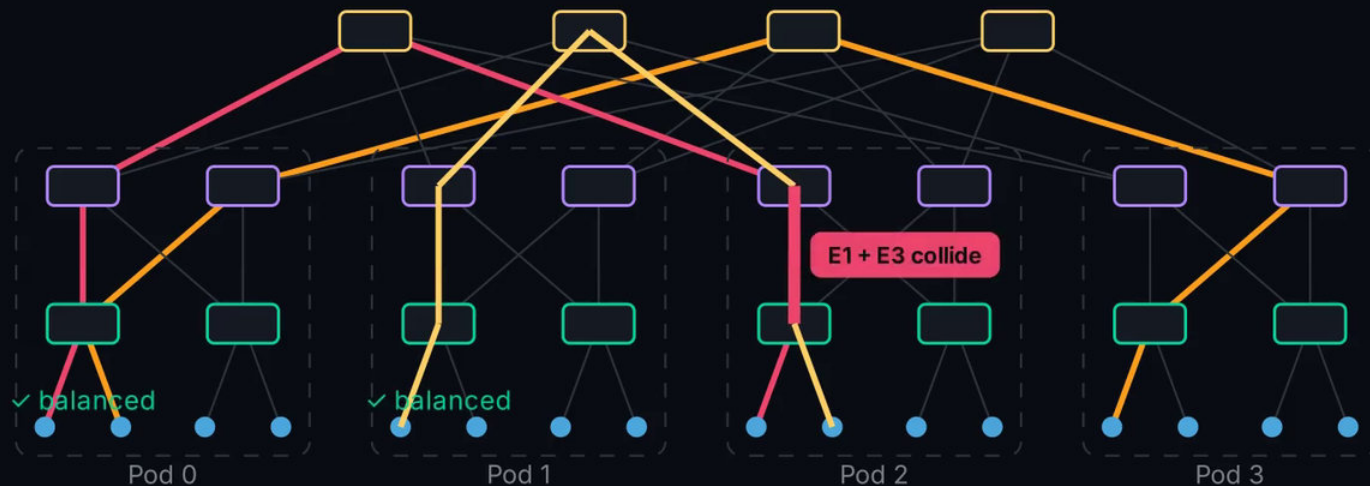
flow classification (§3.6)

- ① remember each flow's port → its packets stay in order
- ② every few seconds, move as few flows as possible to even out its uplinks

Both

Act 8 · Load balancing: who notices the elephant?

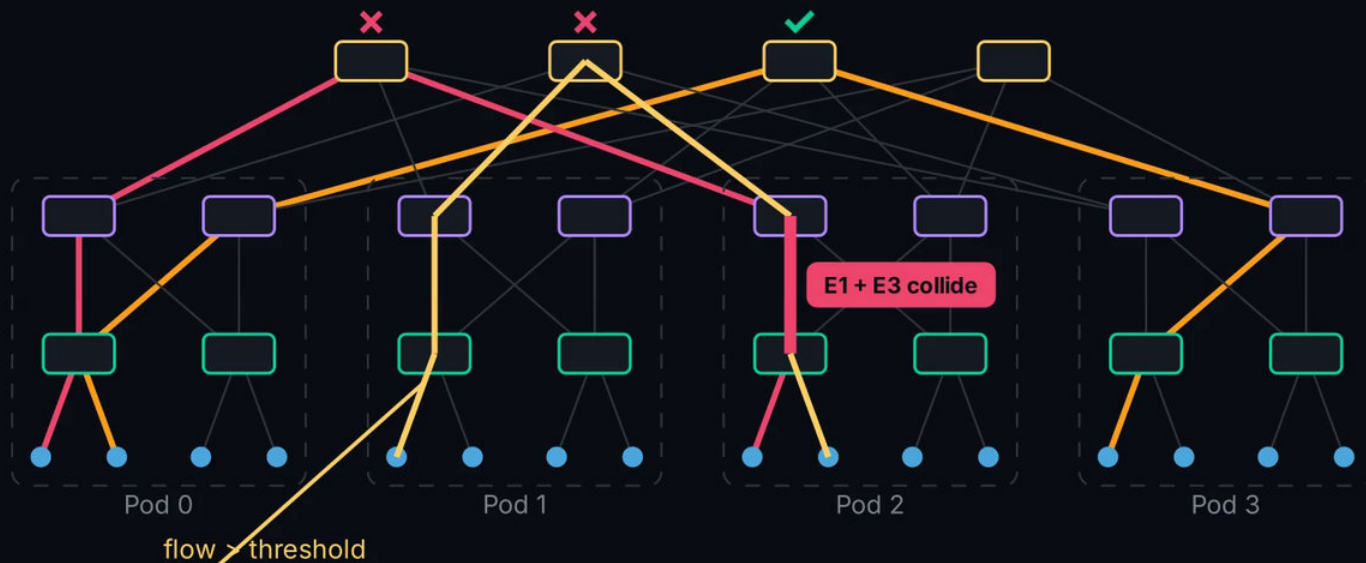
Where local breaks: a collision on the way down



Both

Act 8 · Load balancing: who notices the elephant?

Second idea: someone who sees everything



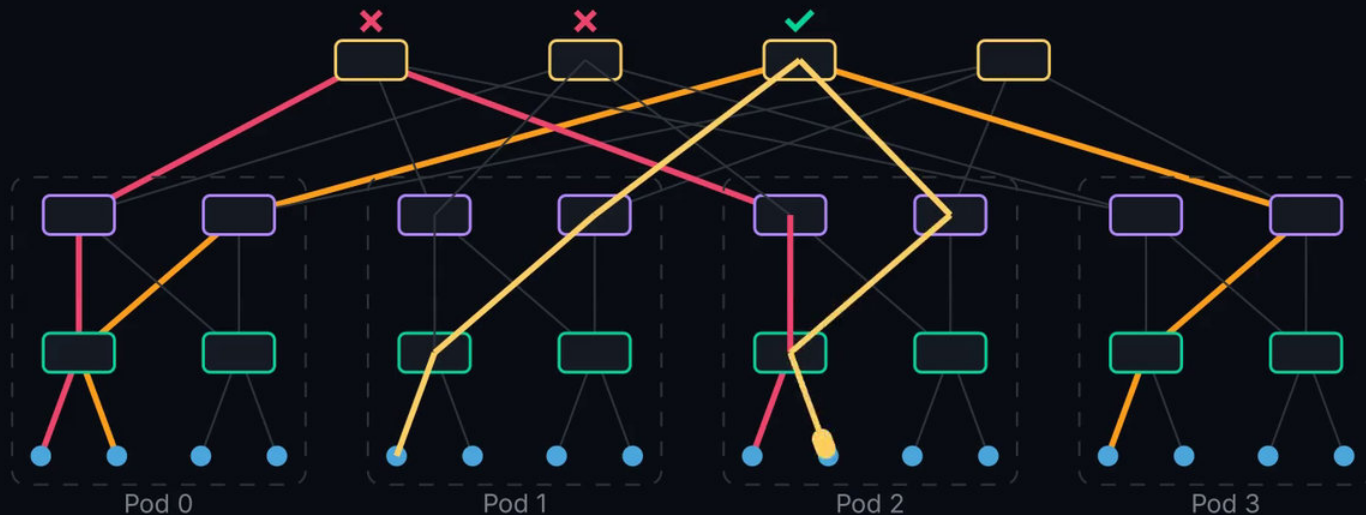
Central scheduler (§3.7)

map of reserved links

Both

Act 8 · Load balancing: who notices the elephant?

Second idea: someone who sees everything



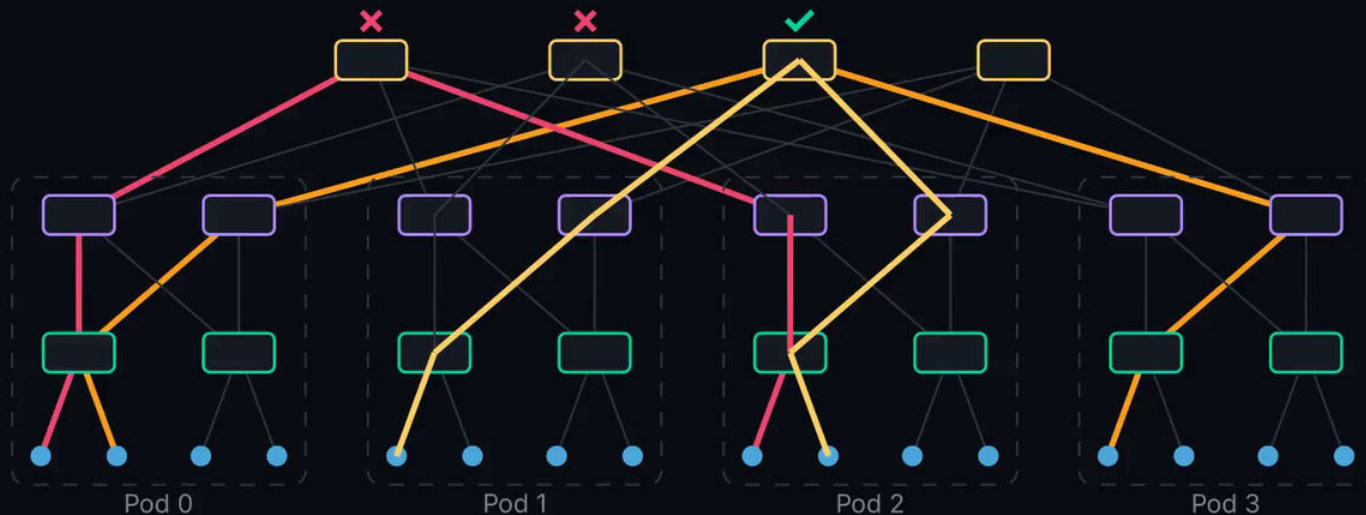
Central scheduler (§3.7)

map of reserved links

Both

Act 8 · Load balancing: who notices the elephant?

Second idea: someone who sees everything



Central scheduler (§3.7)

map of reserved links

greedy placement: 94% of ideal in simulation (27k hosts)

price: central state + still more switch logic

Both

Act 8 · Load balancing: who notices the elephant?

■ traditional tree

■ two-level tables

■ flow classification

■ flow scheduling

100% of ideal bisection

Random

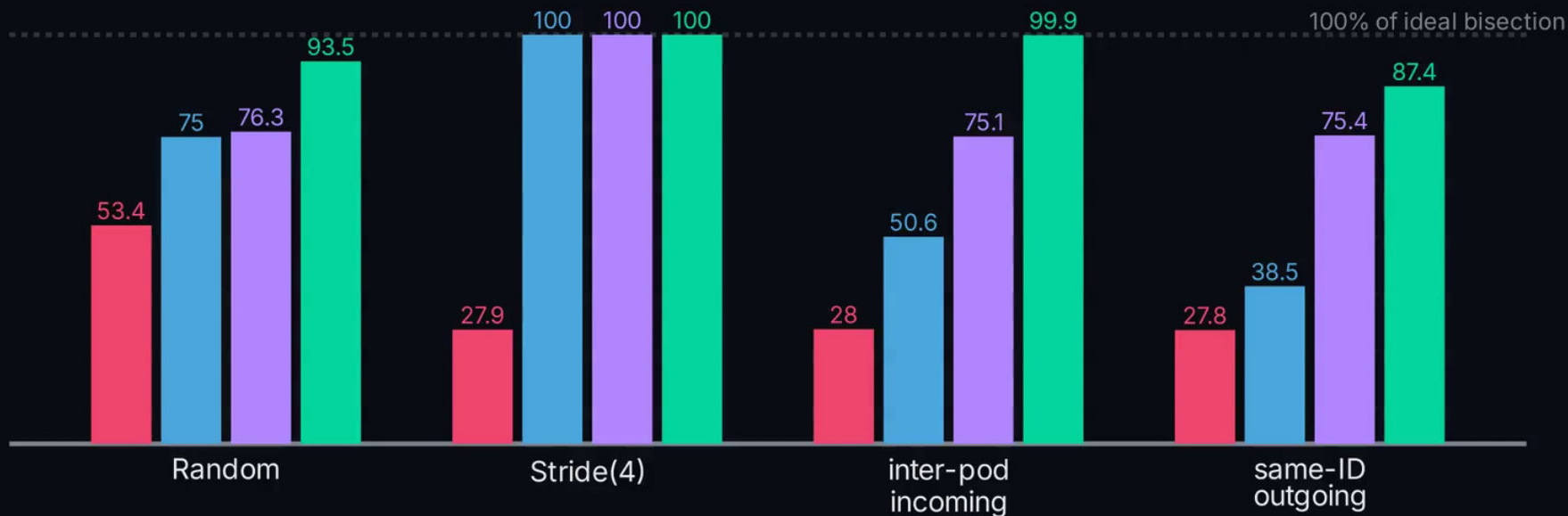
Stride(4)

inter-pod
incoming

same-ID
outgoing

Act 8 · Load balancing: who notices the elephant?

■ traditional tree ■ two-level tables ■ flow classification ■ flow scheduling



Both

Act 8 · Load balancing: who notices the elephant?

At Microsoft: nobody notices, and mostly nobody needs to

Fat-Tree core link: 1G



VL2 core link: 10G (same width, 10× the capacity)



Both

Act 8 · Load balancing: who notices the elephant?

At Microsoft: nobody notices, and mostly nobody needs to

Fat-Tree core link: 1G



each gets 500 Mbps

VL2 core link: 10G (same width, 10× the capacity)



At Microsoft: nobody notices, and mostly nobody needs to

Fat-Tree core link: 1G



each gets 500 Mbps

VL2 core link: 10G (same width, 10× the capacity)



each still gets 1G

At Microsoft: nobody notices, and mostly nobody needs to

Fat-Tree core link: 1G



each gets 500 Mbps

VL2 core link: 10G (same width, 10× the capacity)



11 → each < 1G

At Microsoft: nobody notices, and mostly nobody needs to

Fat-Tree core link: 1G



each gets 500 Mbps

VL2 core link: 10G (same width, 10× the capacity)



11 → each < 1G

VL2 doesn't avoid collisions; 10× links dilute them (load drawn for illustration)

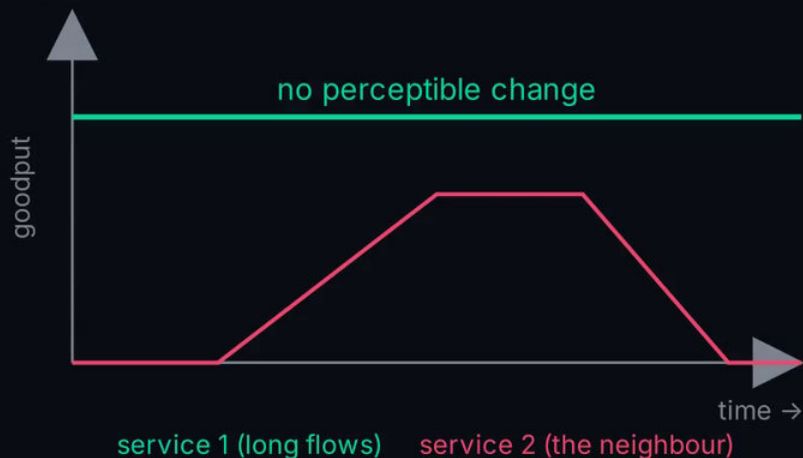
a bet: many medium, fast-changing flows, so > 10 elephants on one link is rare (§3, §5.2)

elephants are the paper's "greatest concern"; fallback: re-hash after heavy TCP loss (§4.2.2)

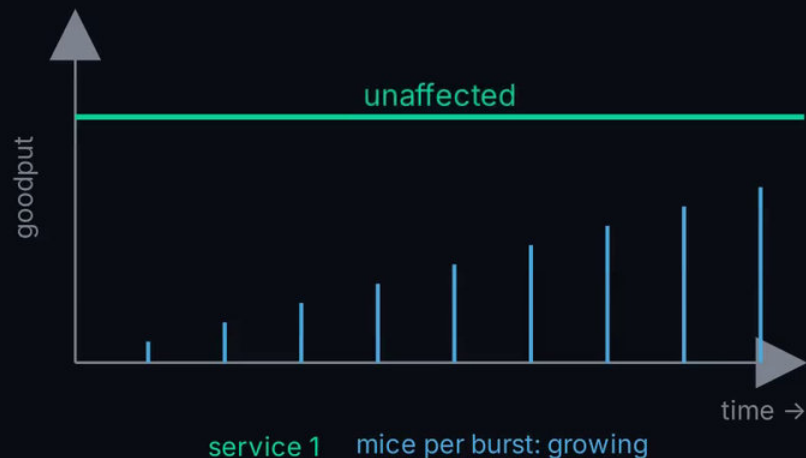
Does the bet hold? A noisy neighbour

VL2 testbed: two services, servers mixed across the same 4 ToRs (§5.3)

① neighbour ramps up 19 elephants (8 GB each)



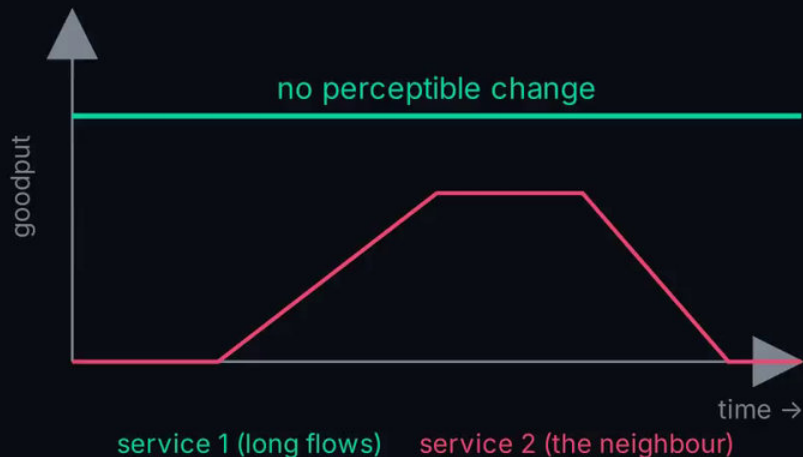
② neighbour fires bursts of mice (1–20 KB)



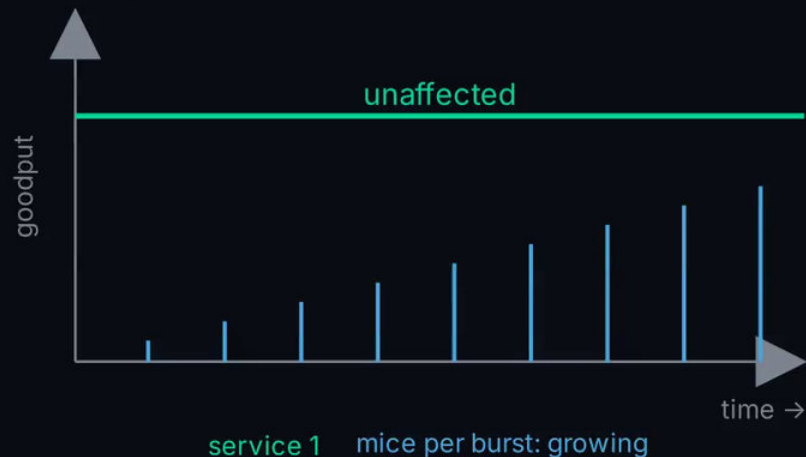
Does the bet hold? A noisy neighbour

VL2 testbed: two services, servers mixed across the same 4 ToRs (§5.3)

① neighbour ramps up 19 elephants (8 GB each)



② neighbour fires bursts of mice (1–20 KB)

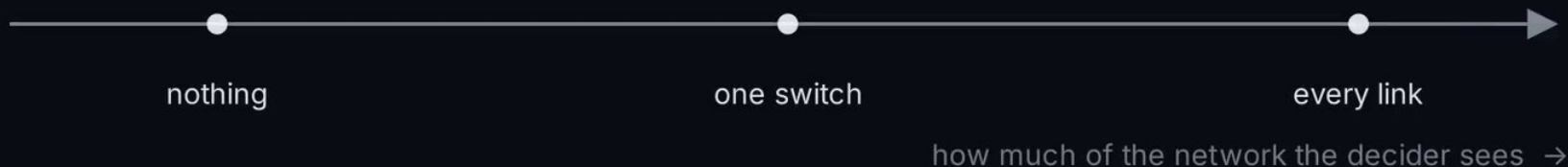


VLB spreads every service evenly + TCP holds each flow to its fair share + no oversubscription
→ isolation without admission control or rate shaping · curves redrawn in shape from Fig. 11/12

Both

Act 8 · Load balancing: who notices the elephant?

Who notices the elephant?



Both

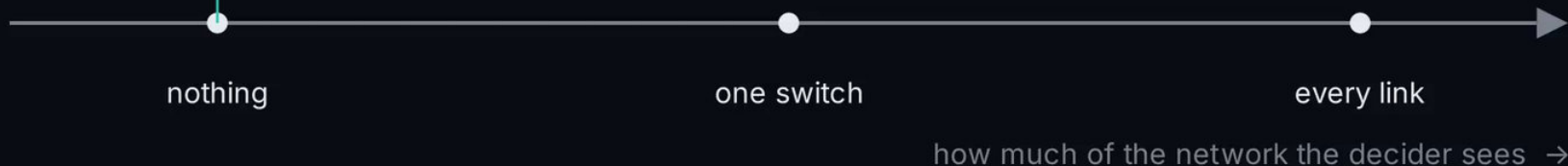
Act 8 · Load balancing: who notices the elephant?

Who notices the elephant?

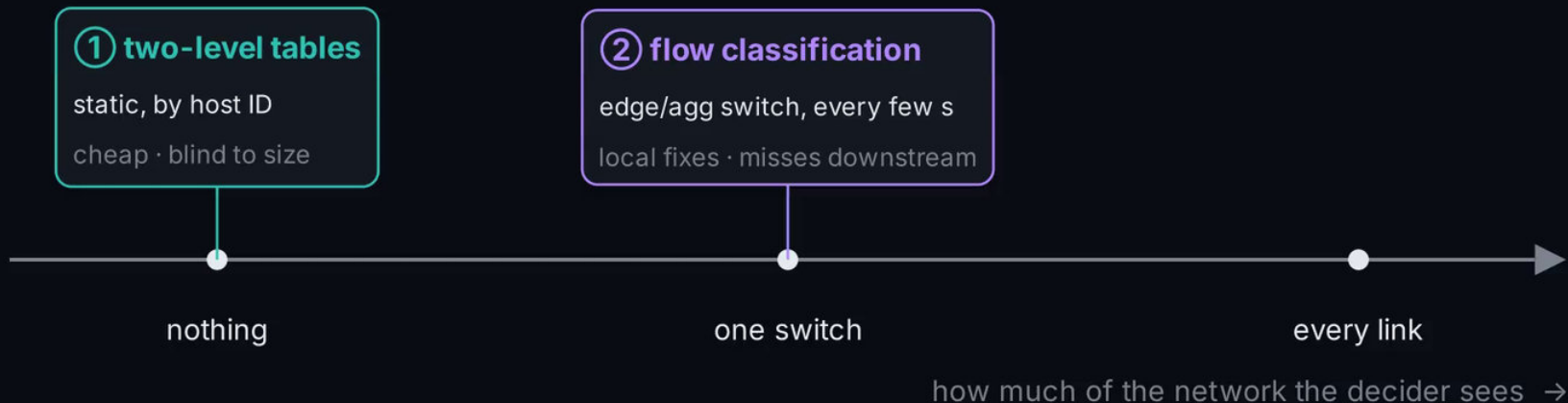
① two-level tables

static, by host ID

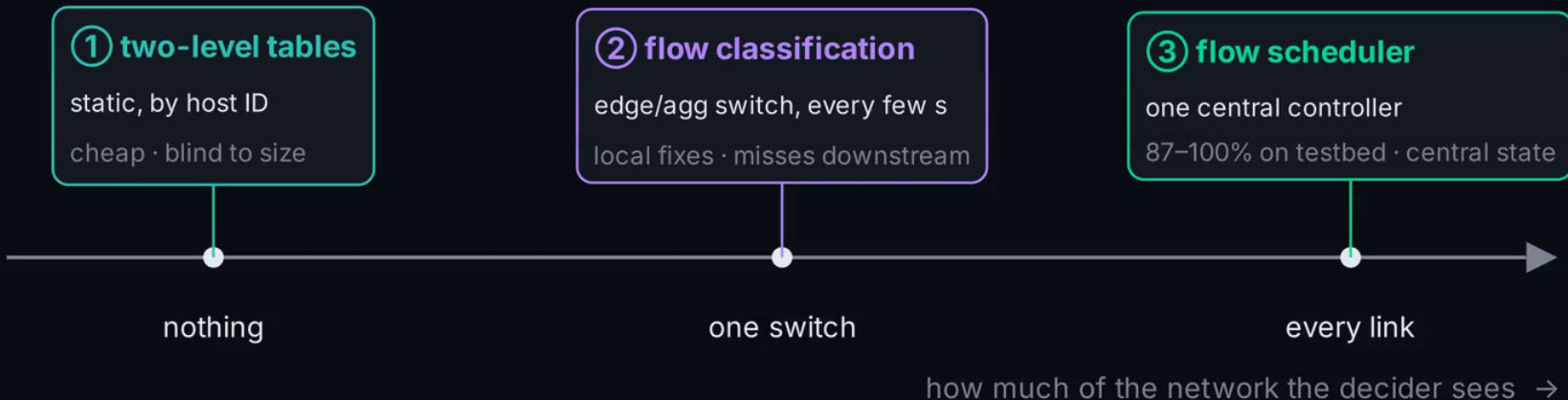
cheap · blind to size



Who notices the elephant?



Who notices the elephant?



Who notices the elephant?



Who notices the elephant?

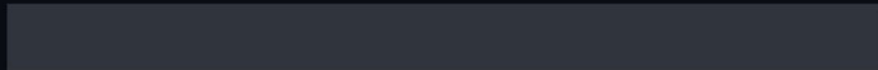


Fat-Tree manages elephants with switch logic; VL2 outspends them with link capacity

VL2 prototype: all-to-all shuffle

80 servers, 10 commodity switches (3 Int · 3 Agg · 4 ToR) · 75 servers × 500 MB to each other = 2.7 TB

max achievable



62.3 Gbps

VL2 goodput



58.8 Gbps

VL2 prototype: all-to-all shuffle

80 servers, 10 commodity switches (3 Int · 3 Agg · 4 ToR) · 75 servers × 500 MB to each other = 2.7 TB

max achievable



62.3 Gbps

VL2 goodput



58.8 Gbps

94% efficiency

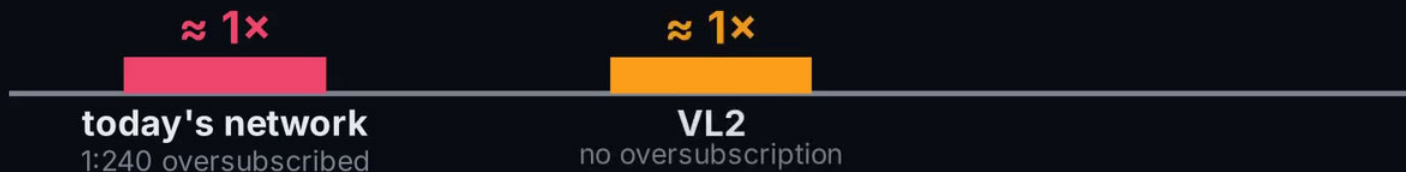
fairness 0.995

395 s

A conventional tree with 1:20 at the ToR would take ~11× longer

And what does it cost?

35K servers · street prices for switches · ToRs and cabling excluded (VL2 §6)



And what does it cost?

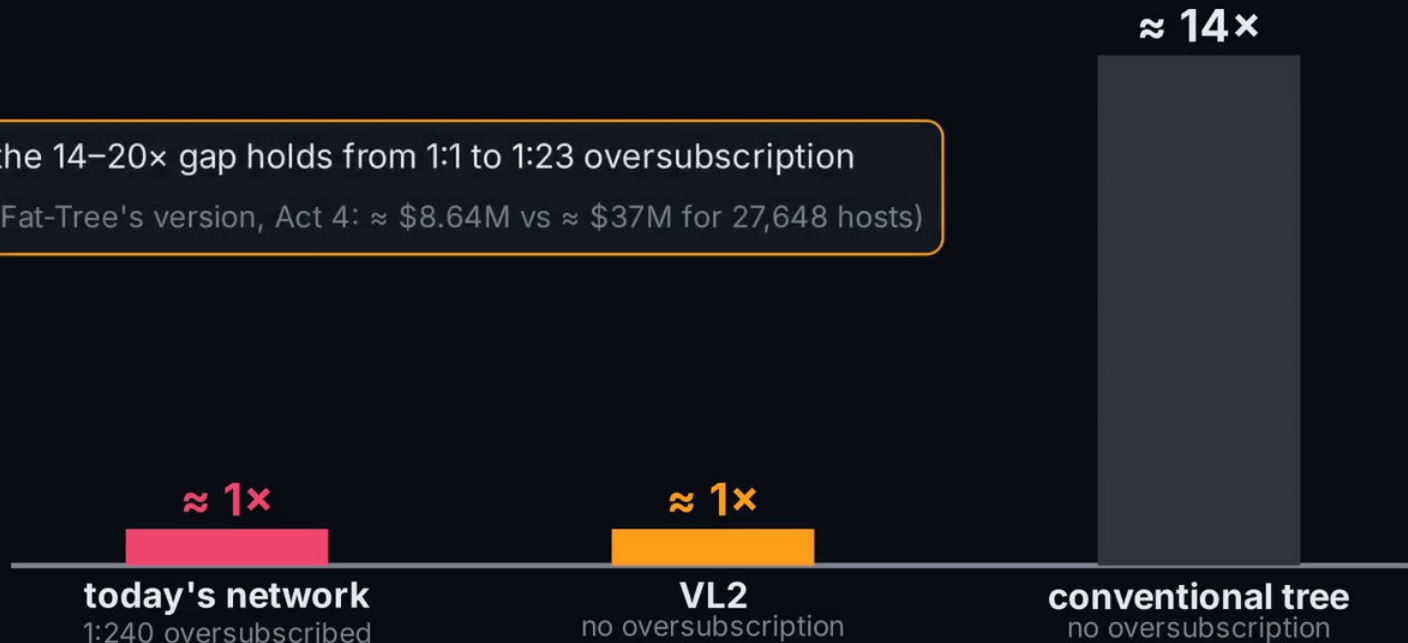
35K servers · street prices for switches · ToRs and cabling excluded (VL2 §6)



And what does it cost?

35K servers · street prices for switches · ToRs and cabling excluded (VL2 \$6)

the 14–20× gap holds from 1:1 to 1:23 oversubscription
(Fat-Tree's version, Act 4: $\approx \$8.64\text{M}$ vs $\approx \$37\text{M}$ for 27,648 hosts)



Both Same goal, two answers

	Fat-Tree (2008)	VL2 (2009)
Shared goal	many paths, cheap switches	fault tolerance at low cost
Topology	k-ary fat-tree, all 1G	Clos, 1G servers / 10G fabric
Who picks the path	switches	end hosts (VL2 agent)
How	deterministic: dst host ID	random: VLB via anycast + ECMP
Big flows	classifier / central scheduler	statistical multiplexing on 10G
Failures	BFD + tags between switches	OSPF + ECMP; hosts never told
What changes	switch forwarding logic	host OS shim; switches stock

Both Same goal, two answers

1. Bandwidth and fault tolerance both come from many paths through many cheap switches
2. Using the paths is the hard part: structure (Fat-Tree) vs. randomness (VL2)
3. Both spread per flow: no reordering, but elephants can collide
4. Where the logic lives decides what you can deploy: switches vs. hosts

Both Same goal, two answers

1. Bandwidth and fault tolerance both come from many paths through many cheap switches
2. Using the paths is the hard part: structure (Fat-Tree) vs. randomness (VL2)
3. Both spread per flow: no reordering, but elephants can collide
4. Where the logic lives decides what you can deploy: switches vs. hosts

Legacy: Clos fabrics + ECMP are how data centers are built today

Next: VL2's other half, virtualization (names vs. locations), alongside NVP