# 6.888:
# Lecture 4
# Data Center Load Balancing

Mohammad Alizadeh

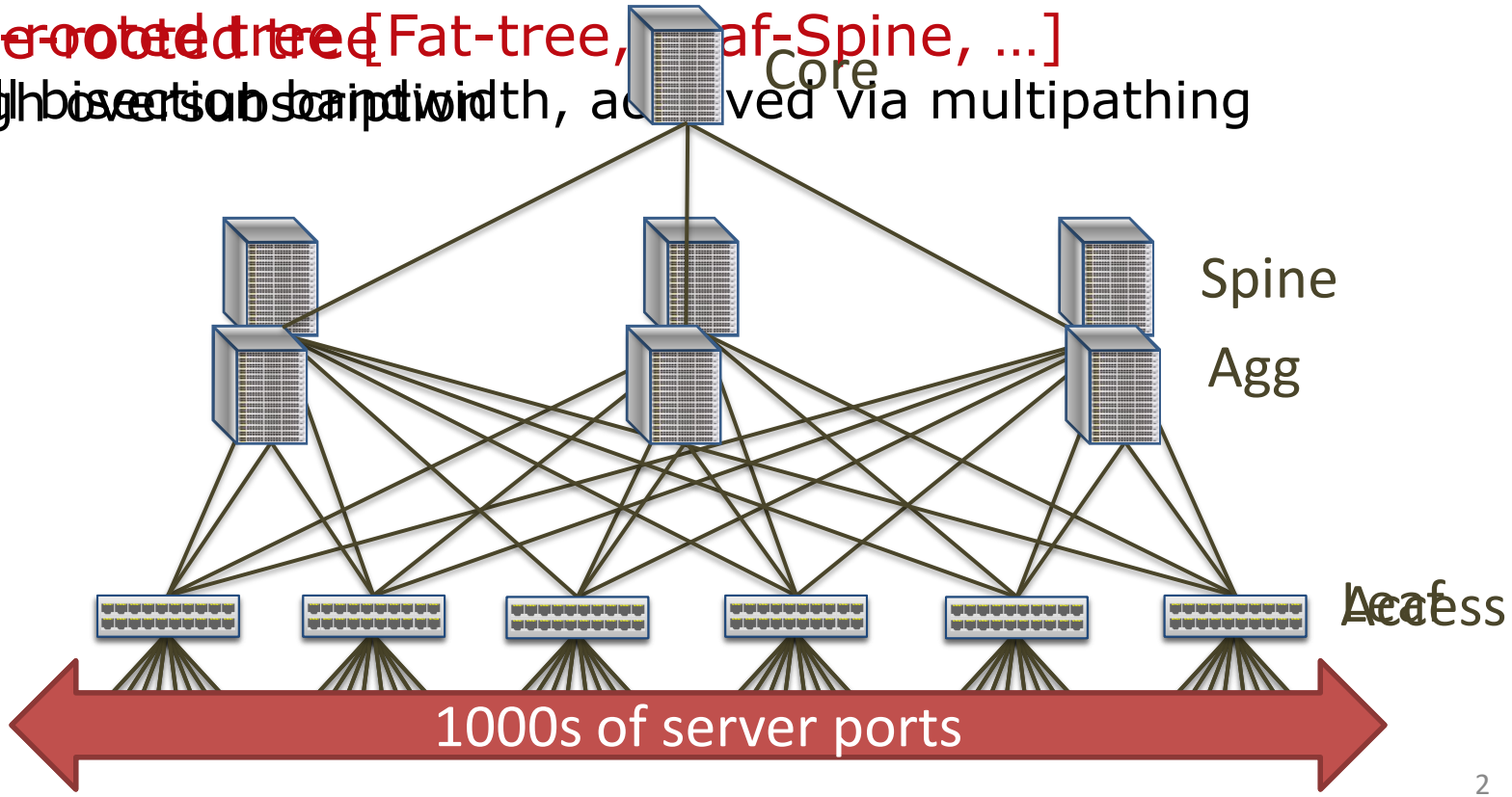
Spring 2016

# Motivation

DC networks need large bisection bandwidth for distributed apps (big data, HPC, web services, etc)

Multi-rooted tree [Fat-tree, Leaf-Spine, …]

- Single-rooted tree
- High oversubscription
- Full bisection bandwidth, achieved via multipathing

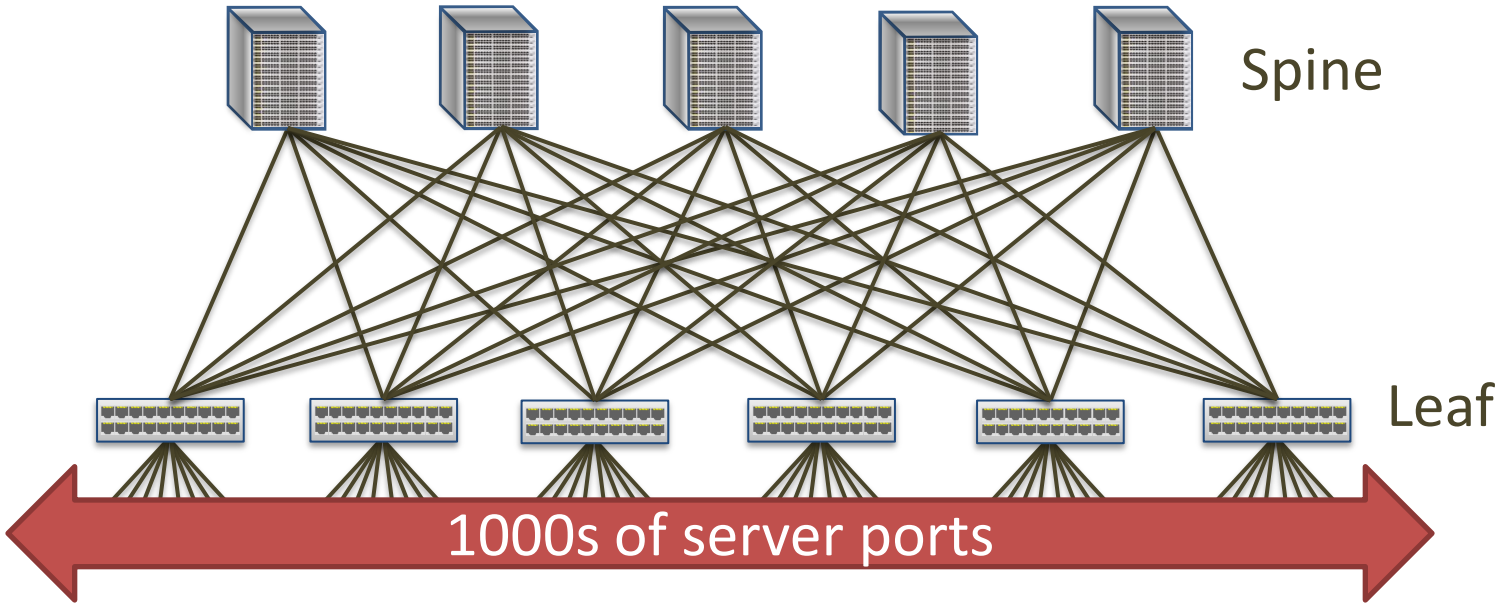


# Motivation

DC networks need large bisection bandwidth for distributed apps (big data, HPC, web services, etc)

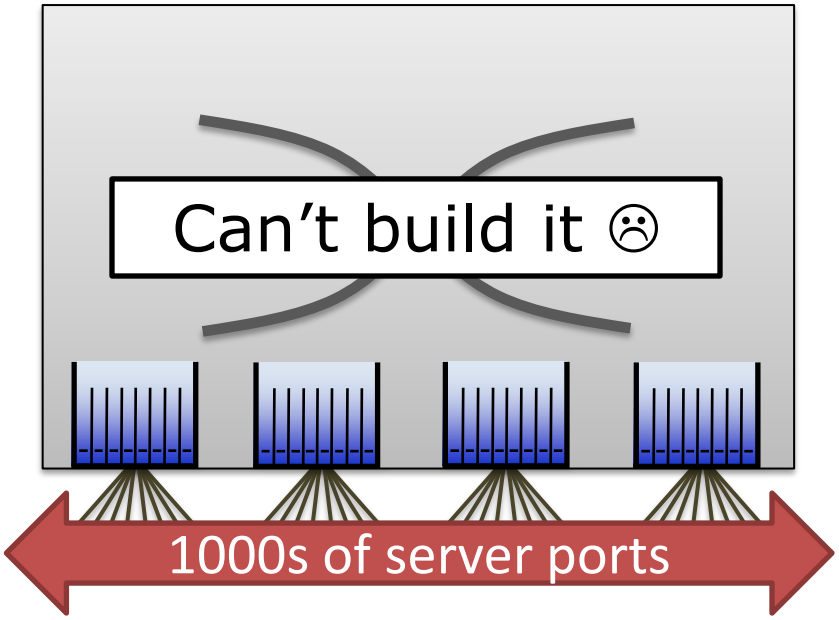
Multi-rooted tree [Fat-tree, Leaf-Spine, …]

- Full bisection bandwidth, achieved via multipathing



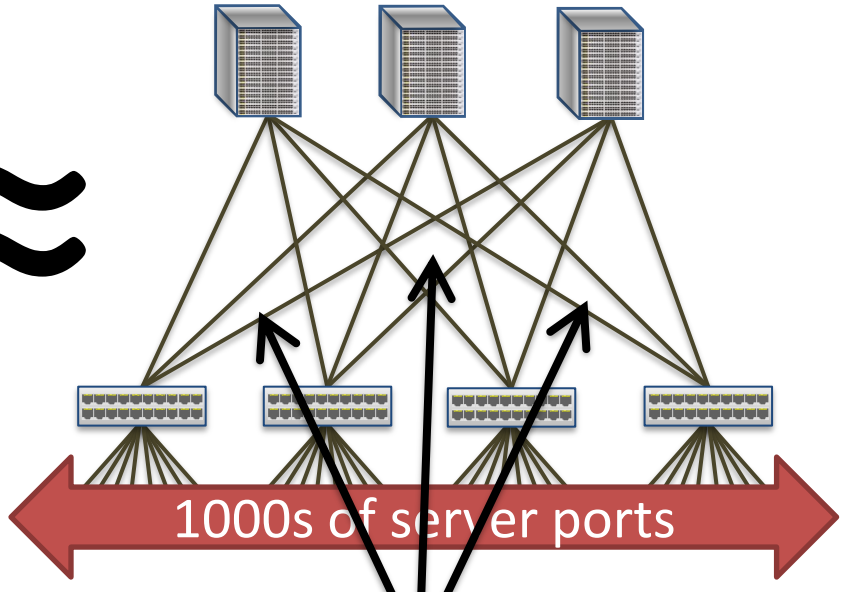
# Multi-rooted != Ideal DC Network

Ideal DC network:
Big output-queued switch




≈

Multi-rooted tree




Need efficient load balancing

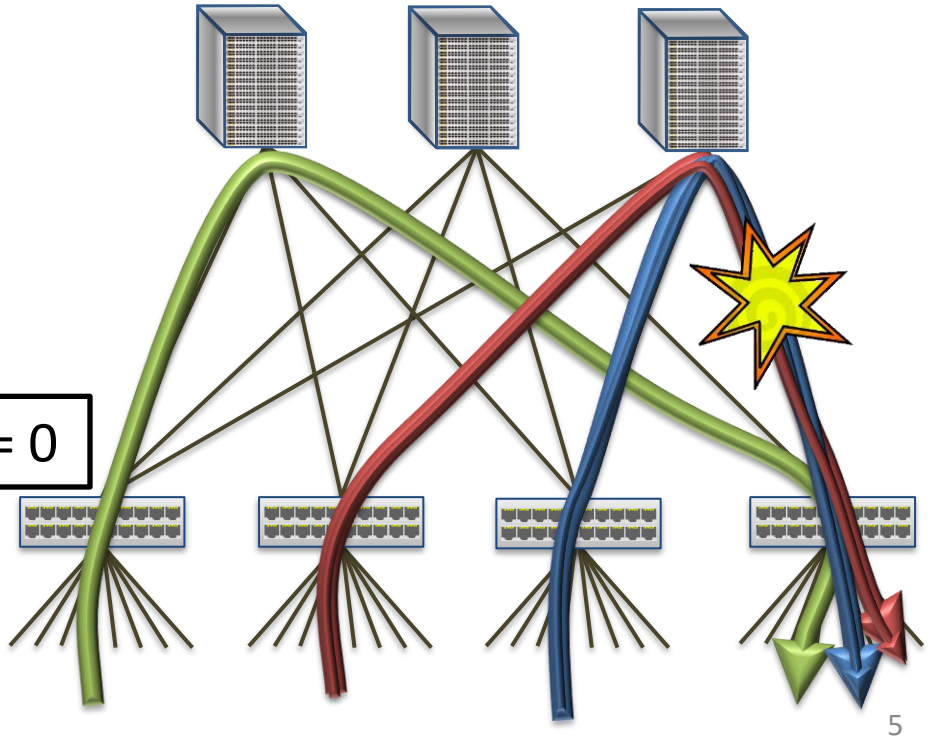
# Today: ECMP Load Balancing

Pick among equal-cost paths by a hash of 5-tuple

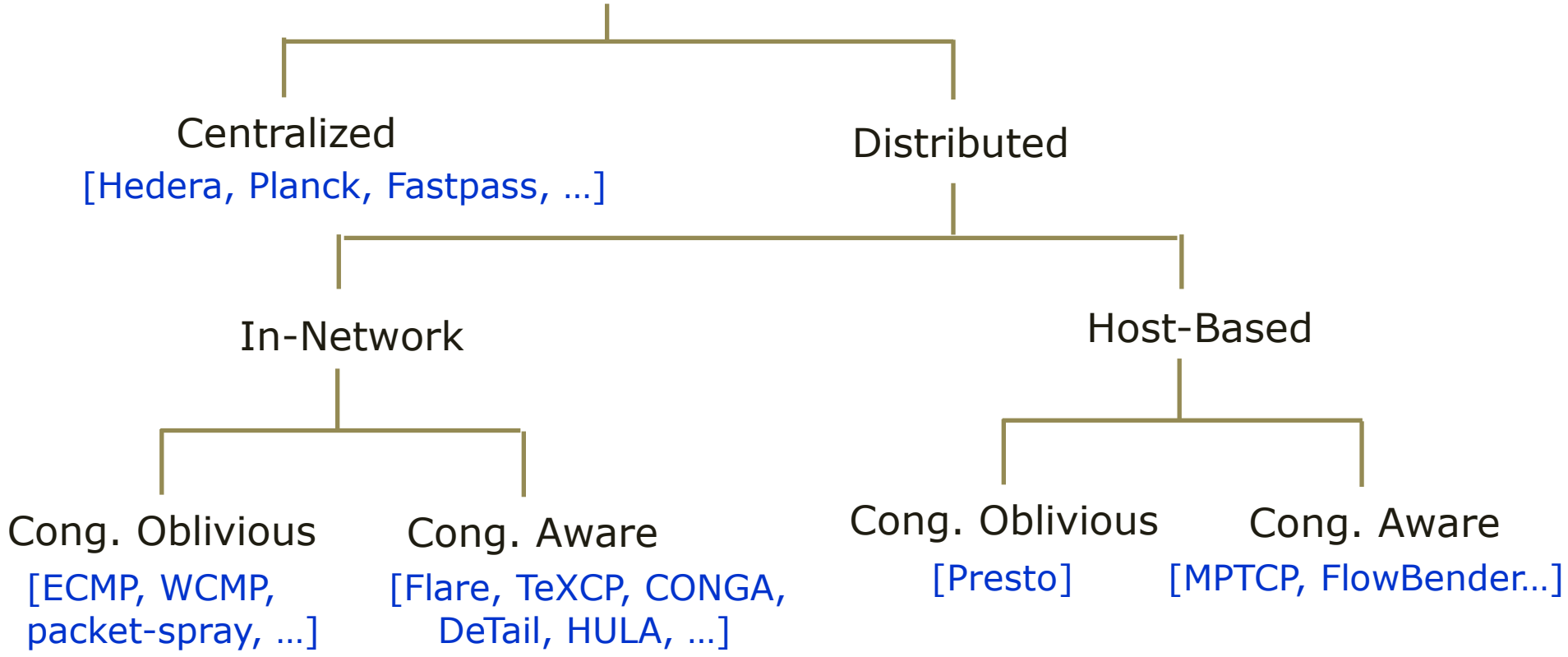
- Randomized load balancing
- Preserves packet order

Problems:

- Hash collisions
  (coarse granularity)
- Local & stateless
  (bad with asymmetry,
  e.g., due to link failures)



# Solution Landscape

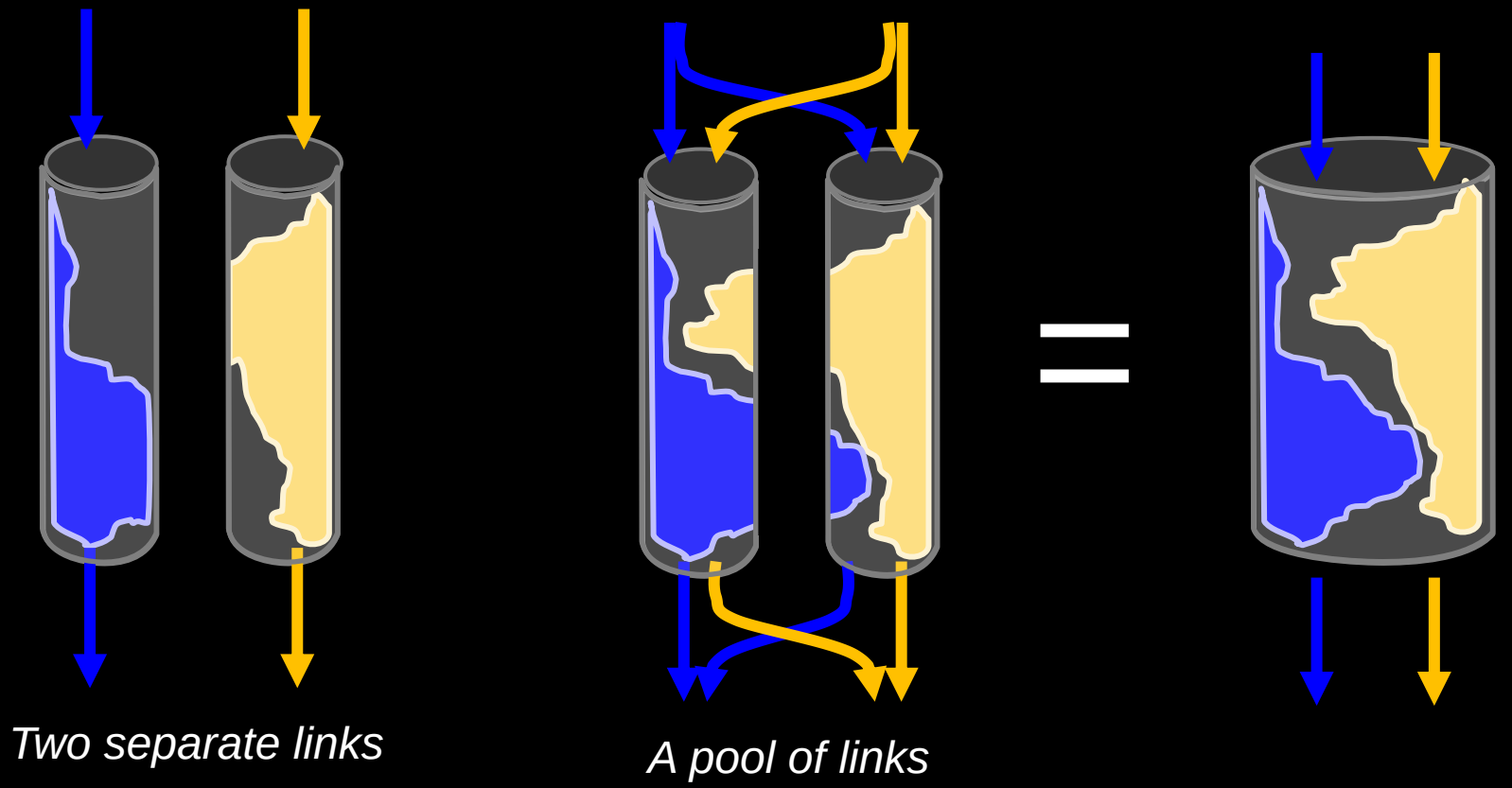


# MPTCP

✧Slides by Damon Wischik (with minor modifications)

# What problem is MPTCP trying to solve?

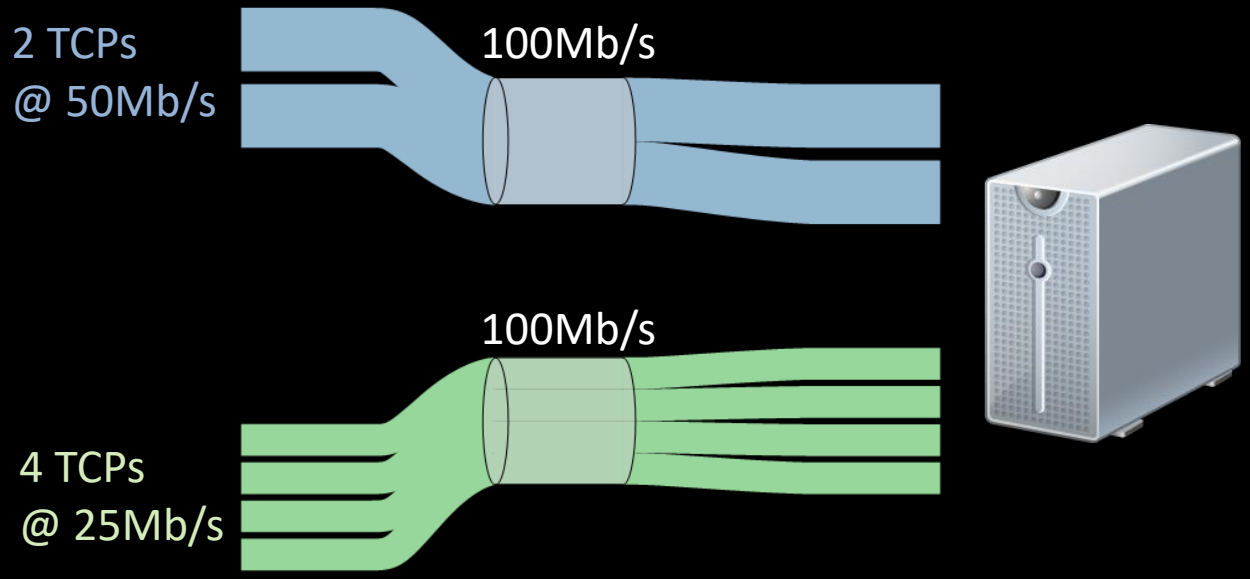
Multipath ‘pools’ links.



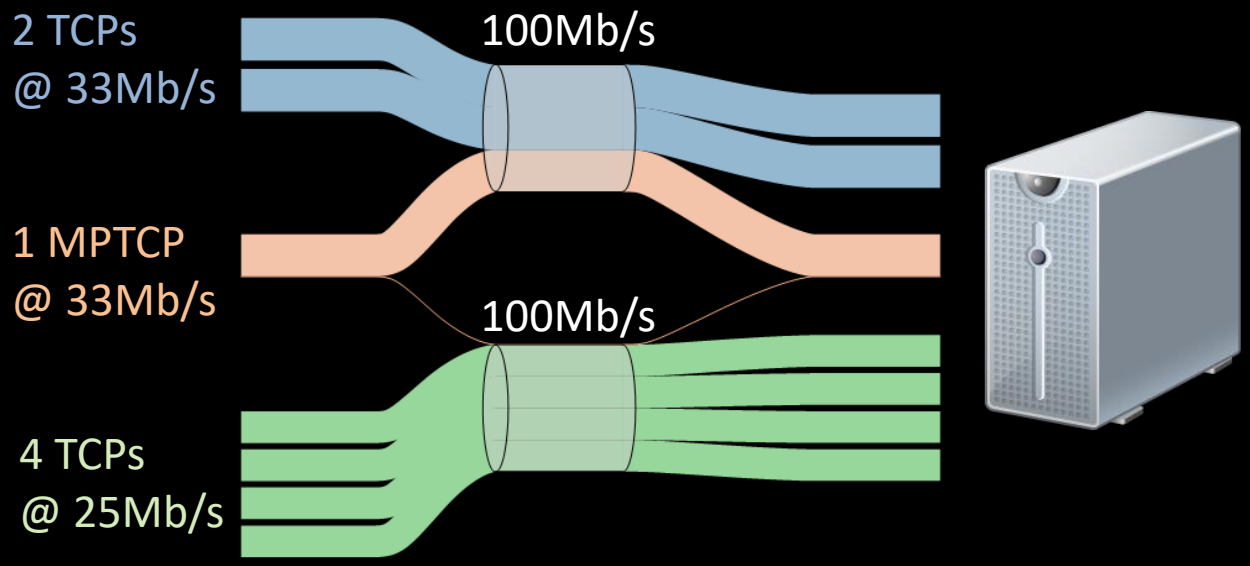

TCP controls how a link is shared.

How should a pool be shared?

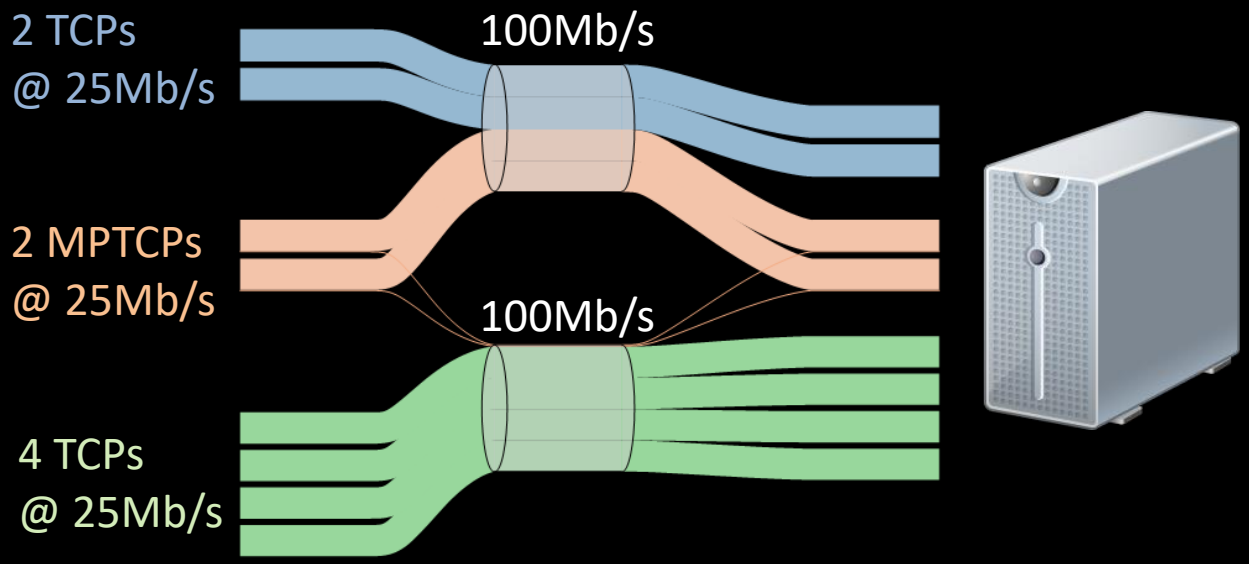
# Application: Multihomed web server



# Application: Multihomed web server

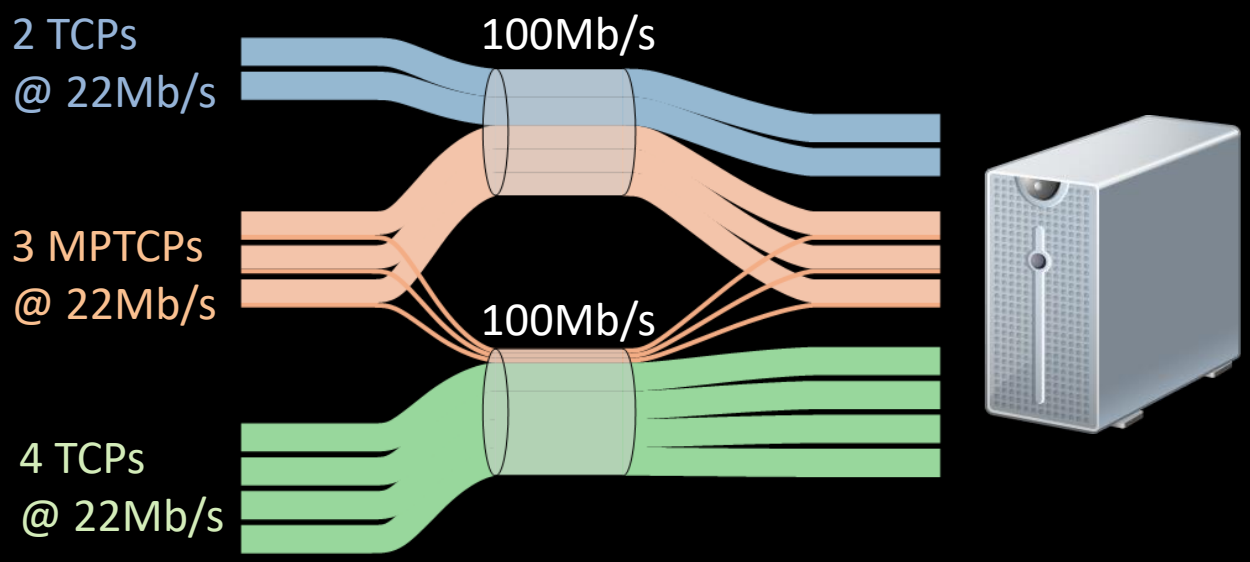


# Application: Multihomed web server




The total capacity, 200Mb/s, is shared out evenly between all 8 flows.

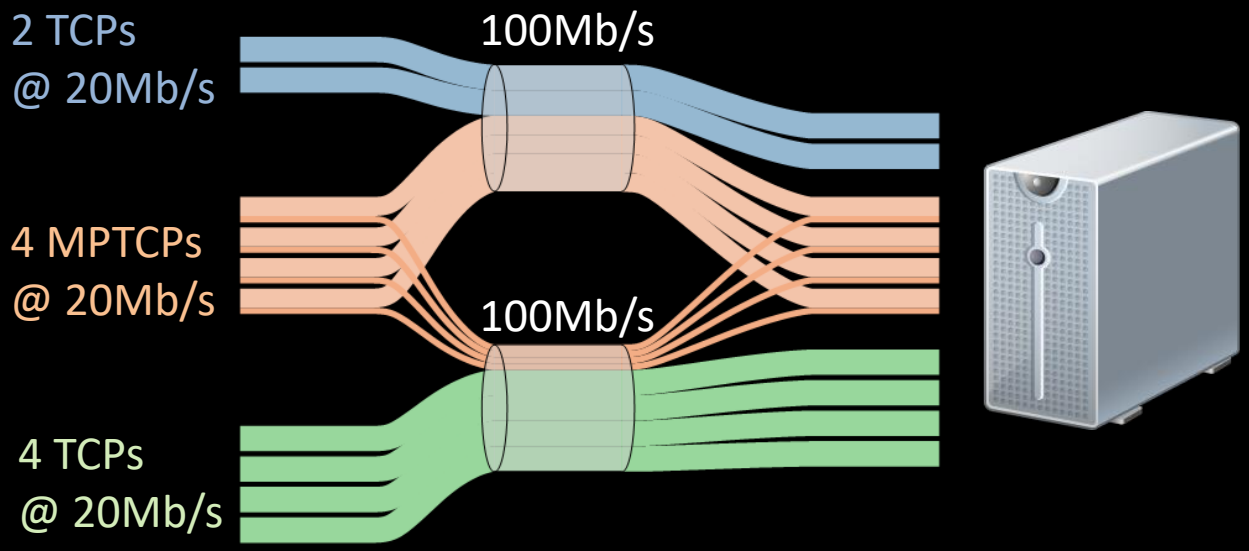
# Application: Multihomed web server




The total capacity, 200Mb/s, is shared out evenly between all 9 flows.

It's as if they were all sharing a single 200Mb/s link. The two links can be said to form a 200Mb/s pool.

# Application: Multihomed web server

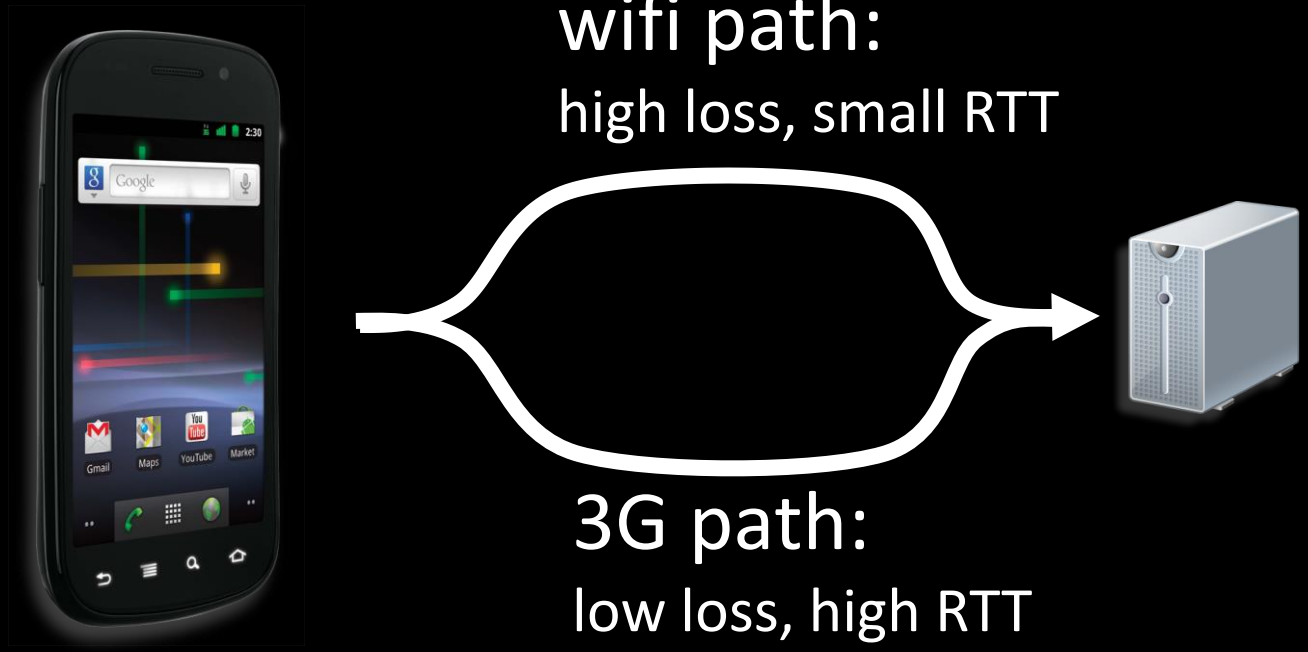



The total capacity, 200Mb/s, is shared out evenly between all 10 flows.

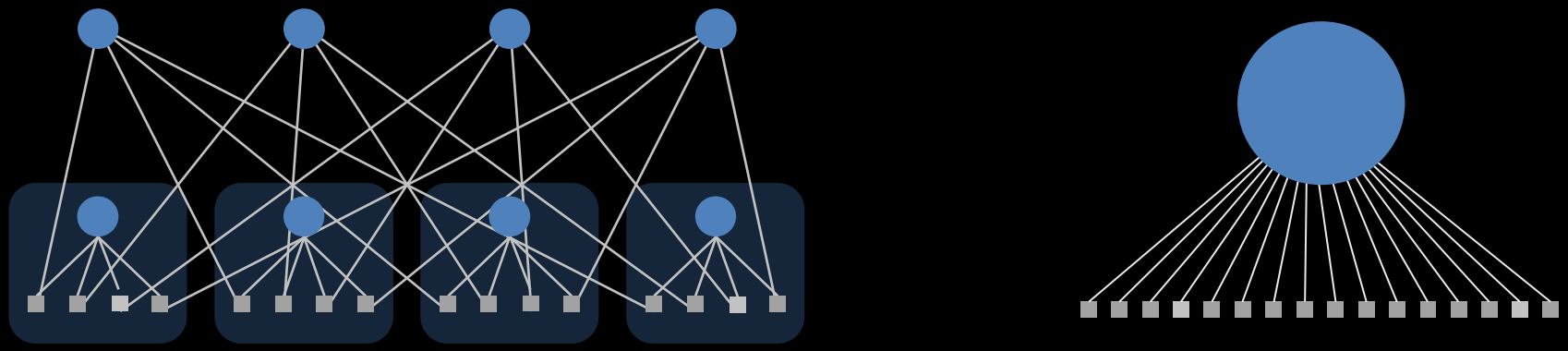
It's as if they were all sharing a single 200Mb/s link. The two links can be said to form a 200Mb/s pool.

# Application: WIFI & cellular together

How should your phone balance its traffic across very different paths?



# Application: Datacenters

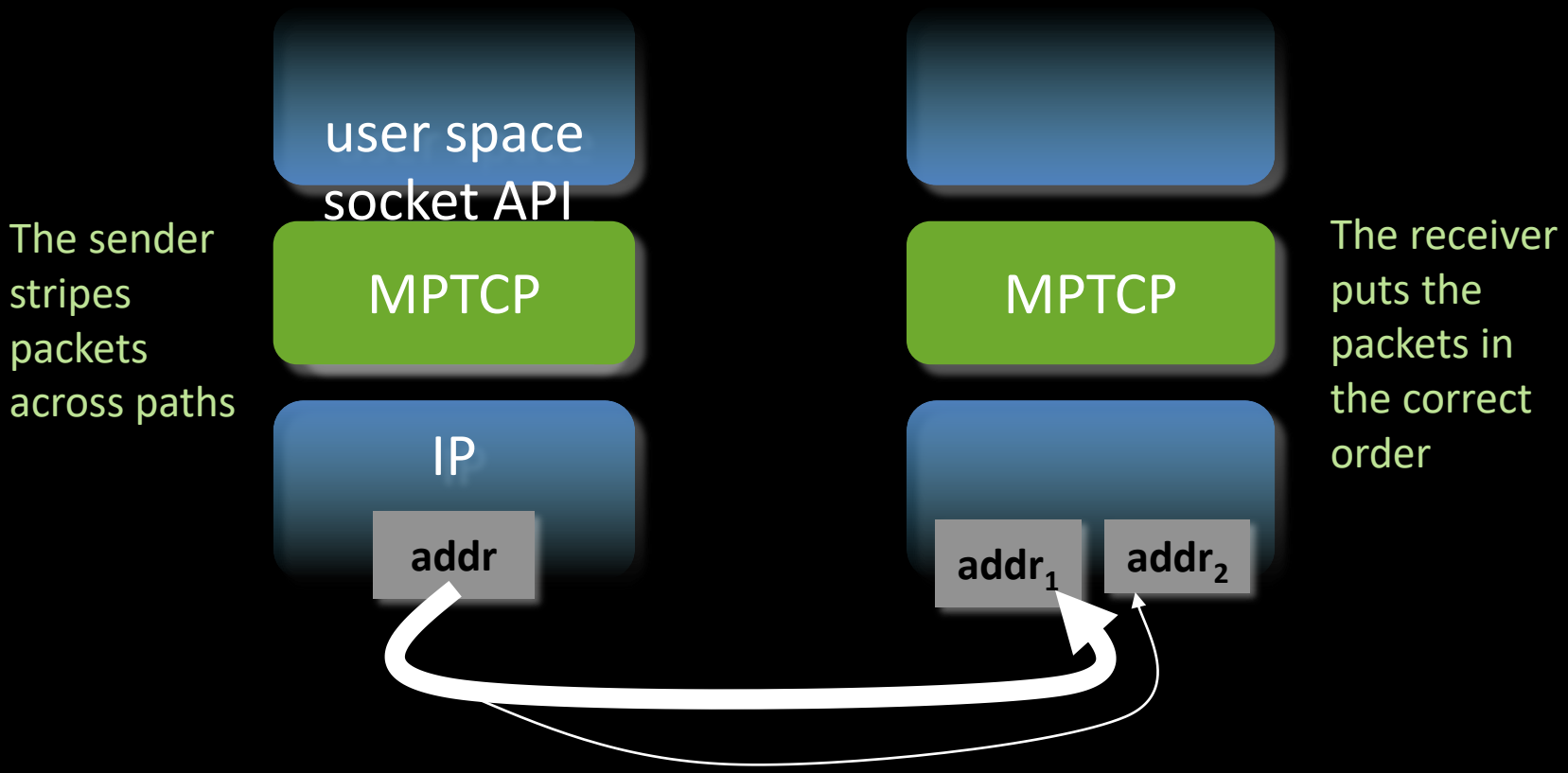


Can we make the network behave like a large pool of capacity?

MPTCP is a general-purpose multipath replacement for TCP.

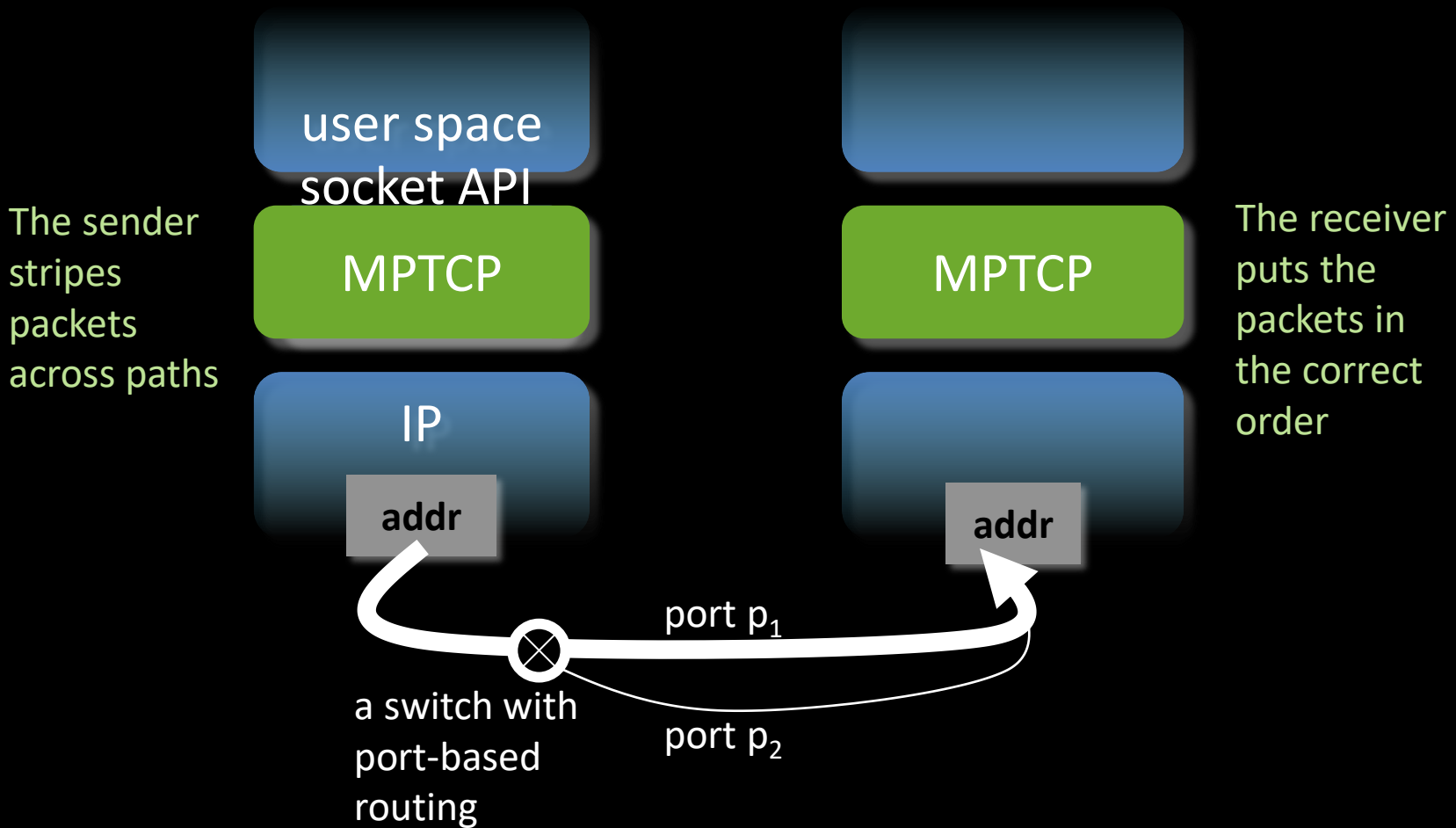
# What is the MPTCP protocol?

MPTCP is a replacement for TCP which lets you use multiple paths simultaneously.



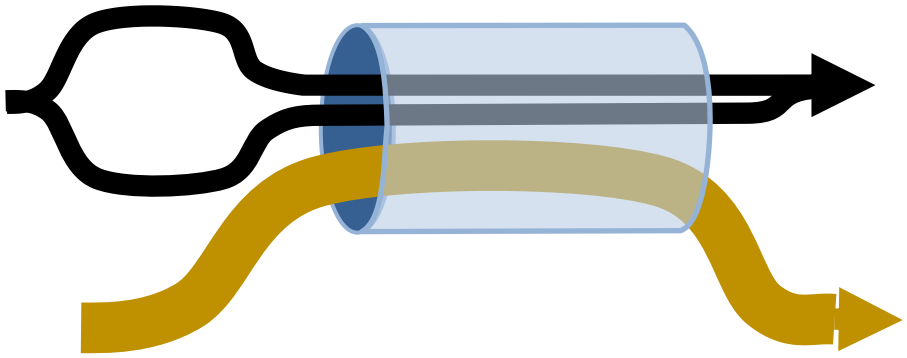
# What is the MPTCP protocol?

MPTCP is a replacement for TCP which lets you use multiple paths simultaneously.



Design goal 1:

# Multipath TCP should be fair to regular TCP at shared bottlenecks

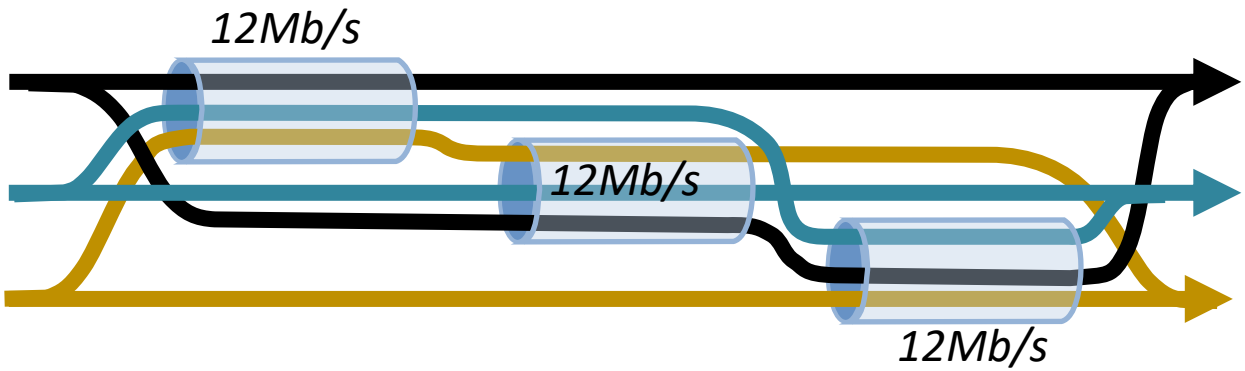



*To be fair, Multipath TCP should take as much capacity as TCP at a bottleneck link, no matter how many paths it is using.*

Strawman solution:

# Run “½ TCP” on each path

# MPTCP should use efficient paths

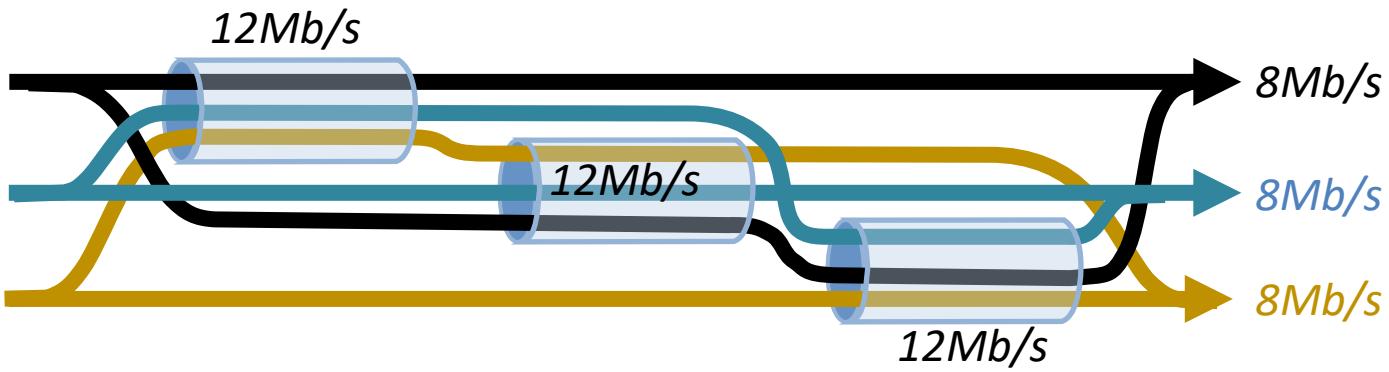



*Each flow has a choice of a 1-hop and a 2-hop path.*

*How should split its traffic?*

Design goal 2:

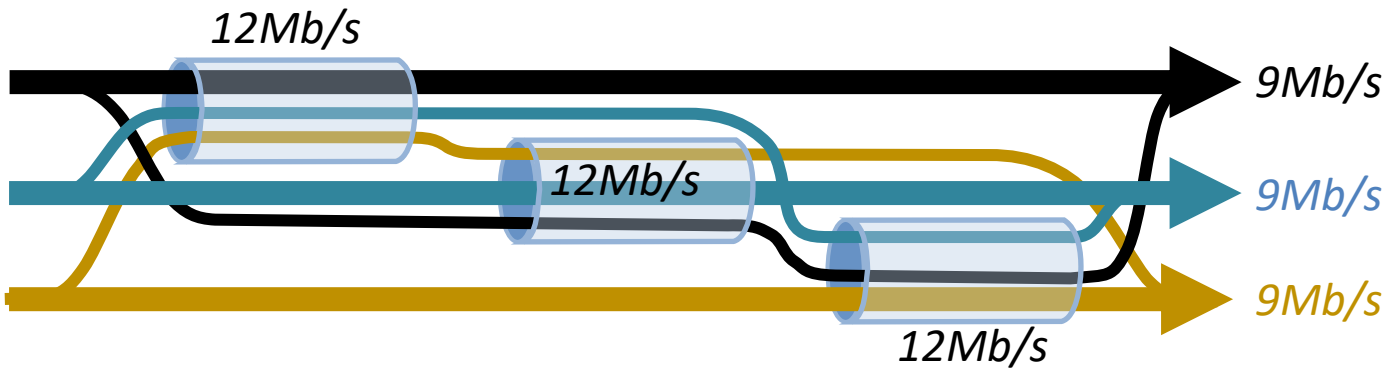
# MPTCP should use efficient paths




*If each flow split its traffic 1:1 ...*

Design goal 2:

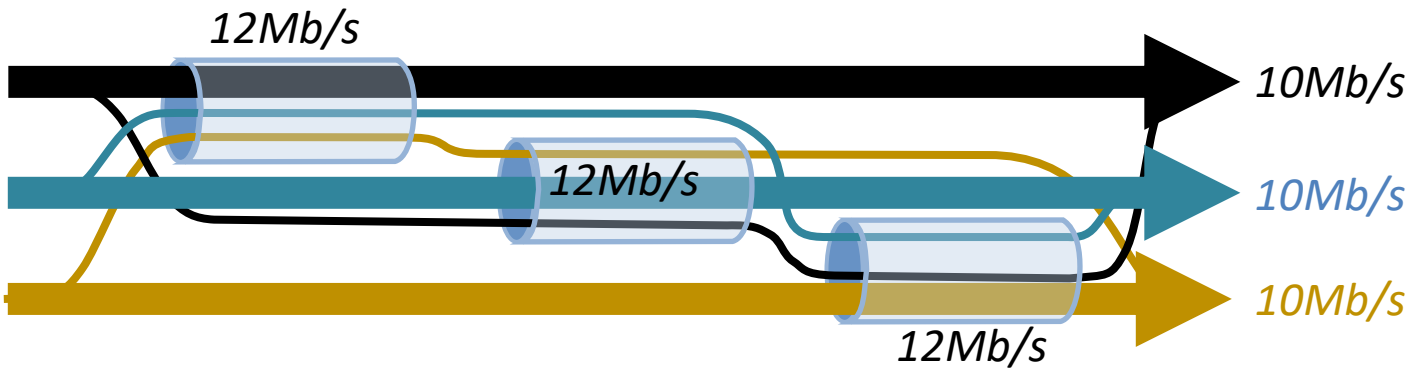
# MPTCP should use efficient paths




*If each flow split its traffic 2:1 ...*

Design goal 2:

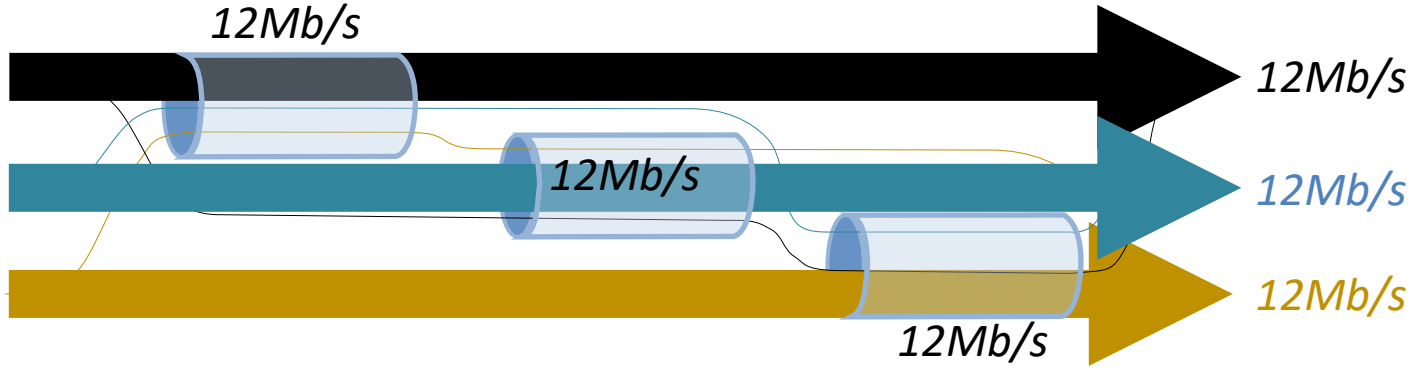
# MPTCP should use efficient paths




*If each flow split its traffic 4:1 ...*

Design goal 2:

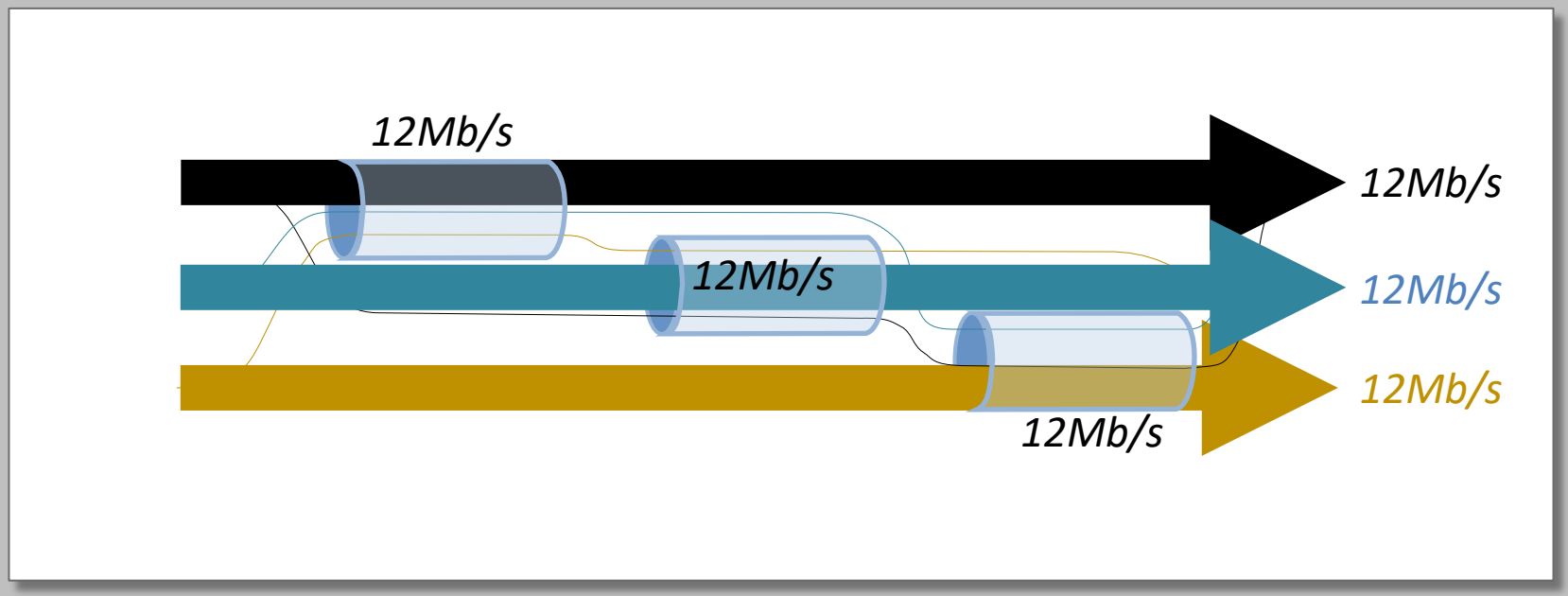
# MPTCP should use efficient paths




*If each flow split its traffic ∞:1 ...*

Design goal 2:

# MPTCP should use efficient paths



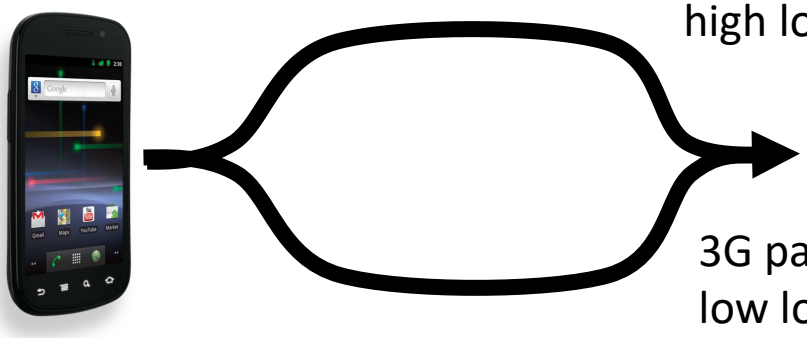

Theoretical solution (Kelly+Voice 2005; Han, Towsley et al. 2006)

MPTCP should send all its traffic on its least-congested paths.

**Theorem.** This will lead to the most efficient allocation possible, given a network topology and a set of available paths.

Design goal 3:

# MPTCP should be fair compared to TCP




*Design Goal 2 says to send all your traffic on the least congested path, in this case 3G. But this has high RTT, hence it will give low throughput.*

Goal 3a. A Multipath TCP user should get at least as much throughput as a single-path TCP would on the best of the available paths.

Goal 3b. A Multipath TCP flow should take no more capacity on any link than a single-path TCP would.

# Design goals

Goal 1. Be fair to TCP at bottleneck links

Goal 2. Use efficient paths ...

Goal 3. as much as we can, while being fair to TCP

Goal 4. Adapt quickly when congestion changes

Goal 5. Don't oscillate

## How does MPTCP achieve all this?

Read: "Design, implementation, and evaluation of congestion control for multipath TCP, NSDI 2011"

# How does TCP congestion control work?

Maintain a congestion window $w$.

- Increase $w$ for each ACK, by $1/w$
- Decrease $w$ for each drop, by $w/2$

# How does MPTCP congestion control work?

Maintain a congestion window $w_r$, one window for each path, where $r \in R$ ranges over the set of available paths.

- Increase $w_r$ for each ACK on path $r$, by

$$\min\left(\frac{a}{\sum_r w_r}, \frac{1}{w_r}\right)$$

- Decrease $w_r$ for each drop on path $r$, by $w_r/2$

# Discussion

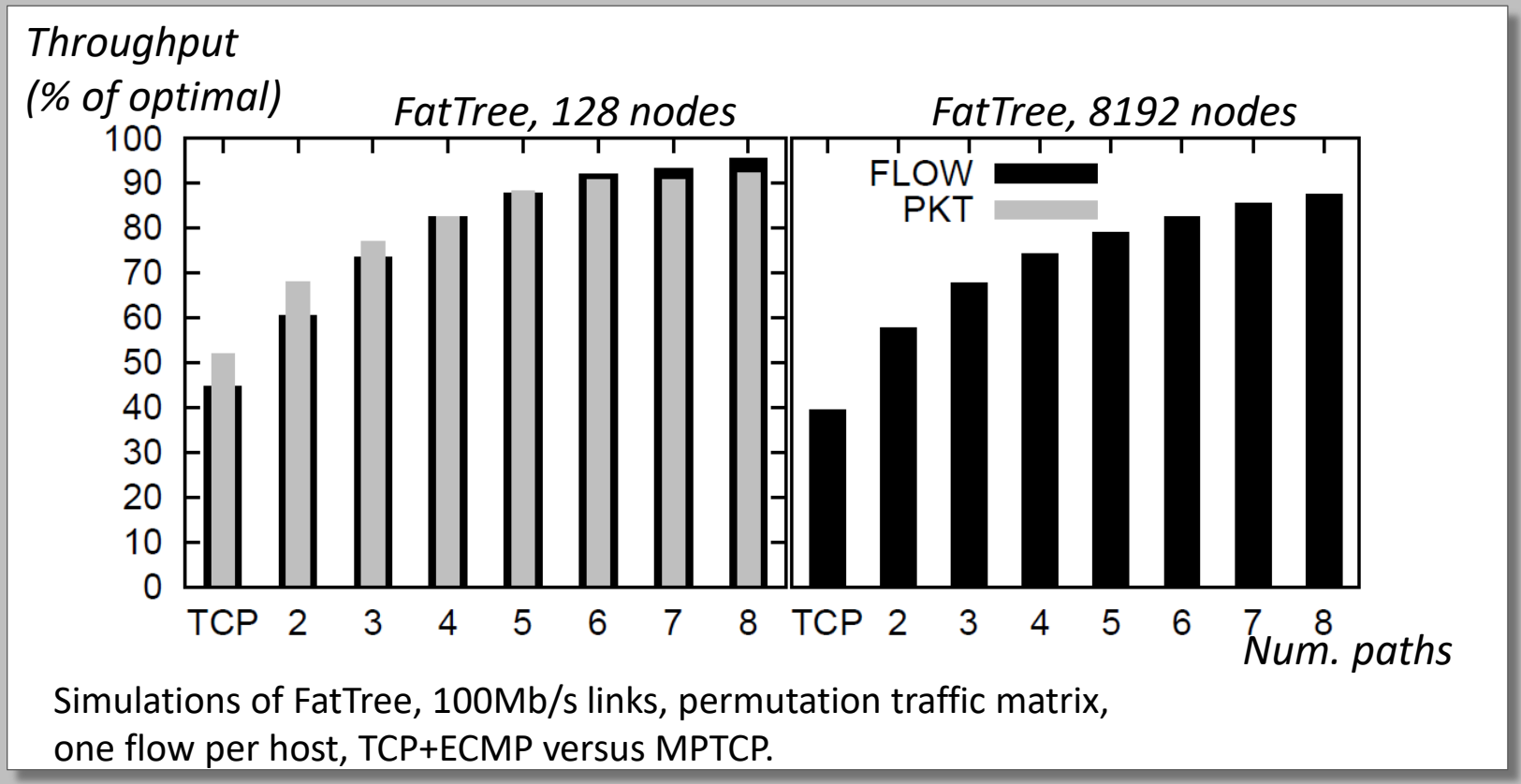
# What You Said

**Ravi:** *"An interesting point in the MPTCP paper is that they target a 'sweet spot' where there is a fair amount of traffic but the core is neither overloaded nor underloaded."*

# What You Said

**Hongzi:** *"The paper talked a bit of `probing' to see if a link has high load and pick some other links, and there are some specific ways of assigning randomized assignment of subflows on links. I was wondering does the `power of 2 choices' have some roles to play here?"*

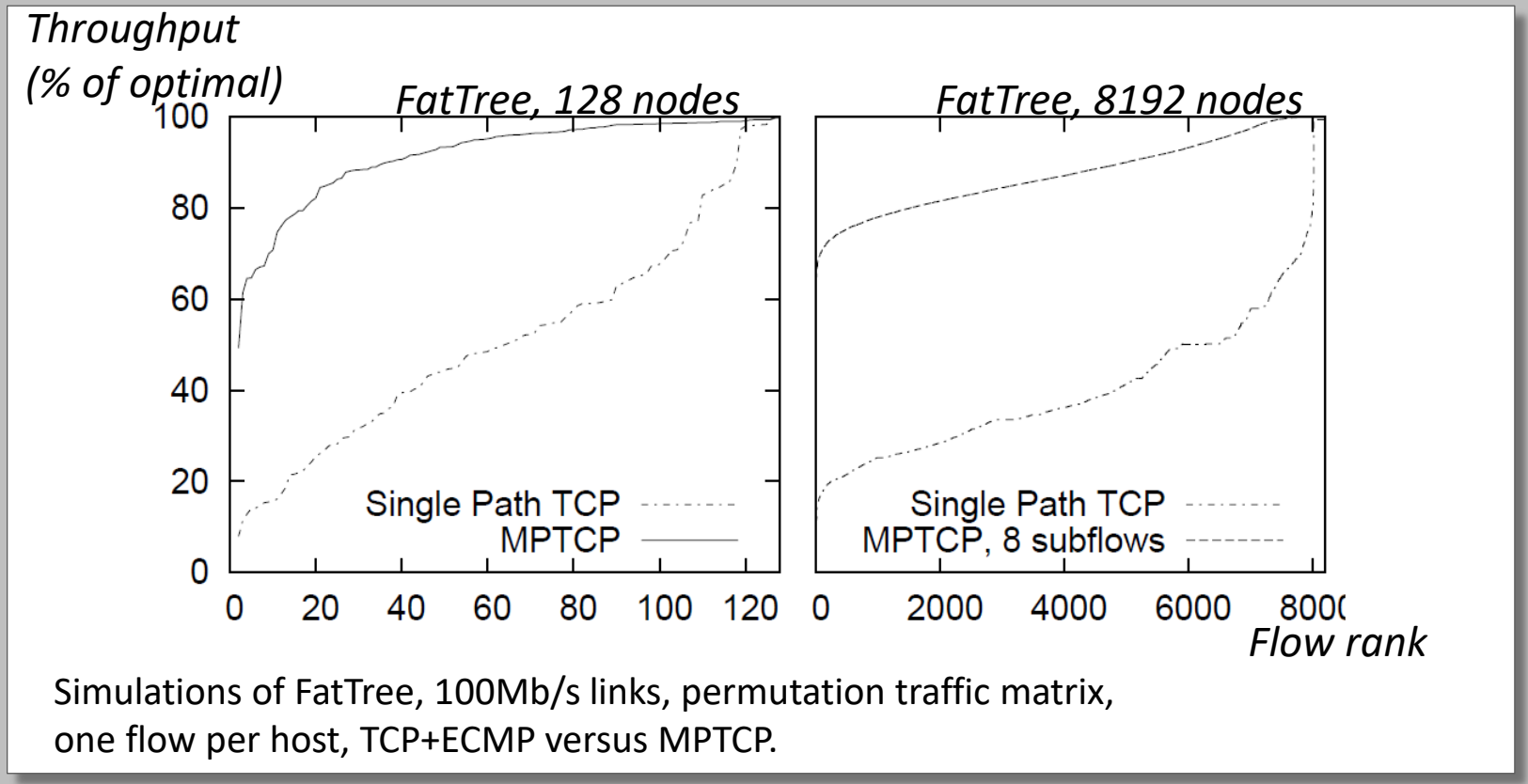
# MPTCP discovers available capacity, and it doesn't need much path choice.




Simulations of FatTree, 100Mb/s links, permutation traffic matrix, one flow per host, TCP+ECMP versus MPTCP.

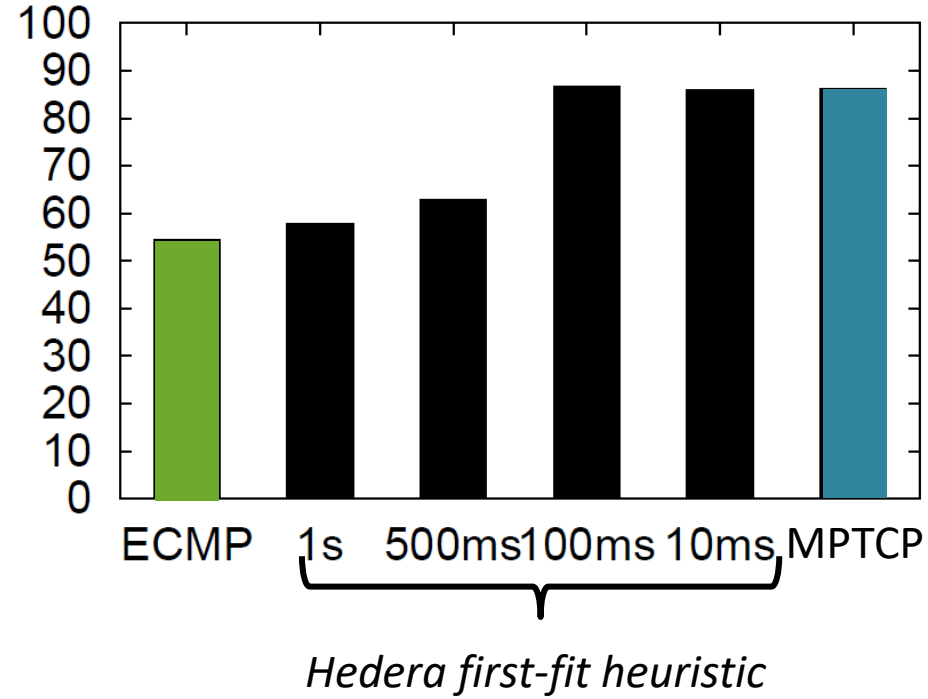
If each node-pair balances its traffic over 8 paths, chosen at random, then utilization is around 90% of optimal.

# MPTCP discovers available capacity, and it shares it out more fairly than TCP+ECMP.




Simulations of FatTree, 100Mb/s links, permutation traffic matrix, one flow per host, TCP+ECMP versus MPTCP.

# MPTCP can make good path choices, as good as a very fast centralized scheduler.

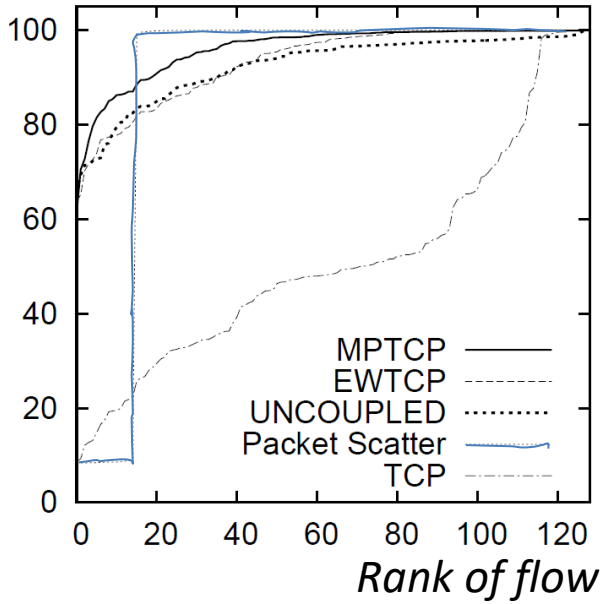


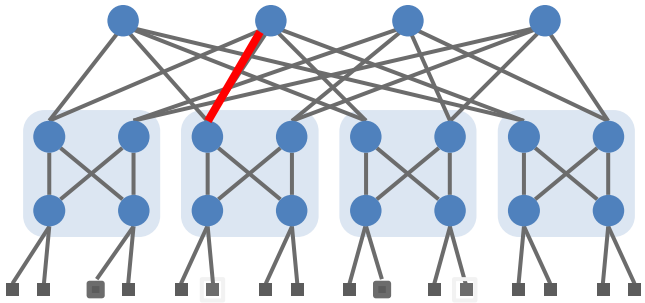

Simulation of FatTree with 128 hosts.

- Permutation traffic matrix
- Closed-loop flow arrivals (one flow finishes, another starts)
- Flow size distributions from VL2 dataset

# MPTCP permits flexible topologies

*Average throughput*
*[% of optimal]*

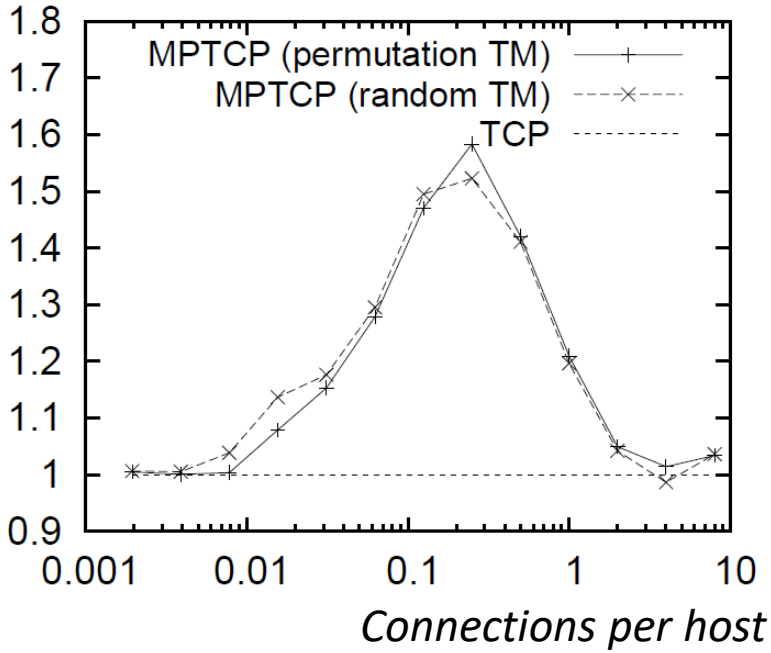


Simulation of 128-node FatTree, when one of the 1Gb/s core links is cut to 100Mb/s

Because an MPTCP flow shifts its traffic onto its least congested paths, congestion hotspots are made to “diffuse” throughout the network. Non-adaptive congestion control, on the other hand, does not cope well with non-homogenous topologies.

# MPTCP permits flexible topologies



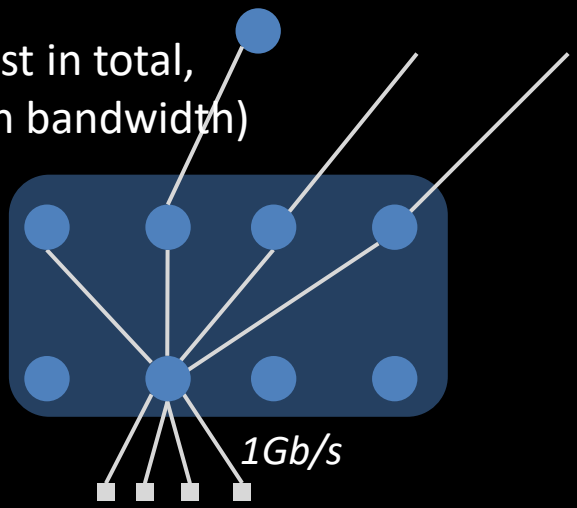

Simulation of a FatTree-like topology with 512 nodes, but with 4 hosts for every up-link from a top-of-rack switch, i.e. the core is oversubscribed 4:1.

- Permutation TM: each host sends to one other, each host receives from one other
- Random TM: each host sends to one other, each host may receive from any number

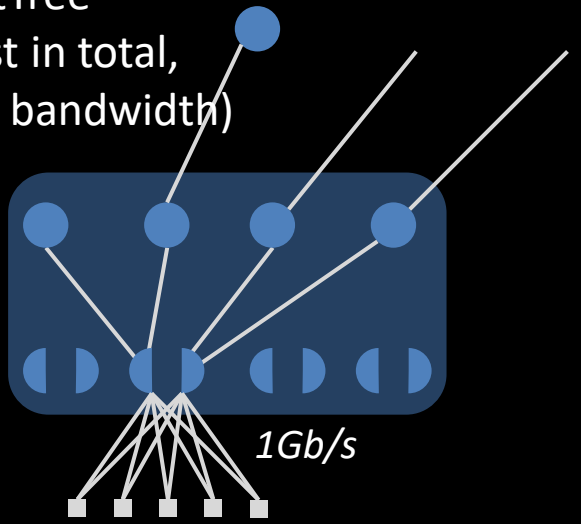
- At low loads, there are few collisions, and NICs are saturated, so TCP ≈ MPTCP
- At high loads, the core is severely congested, and TCP can fully exploit all the core links, so TCP ≈ MPTCP
- When the core is “right-provisioned”, i.e. just saturated, MPTCP > TCP

# MPTCP permits flexible topologies

FatTree
(5 ports per host in total,
1Gb/s bisection bandwidth)




Dual-homed FatTree
(5 ports per host in total,
1Gb/s bisection bandwidth)

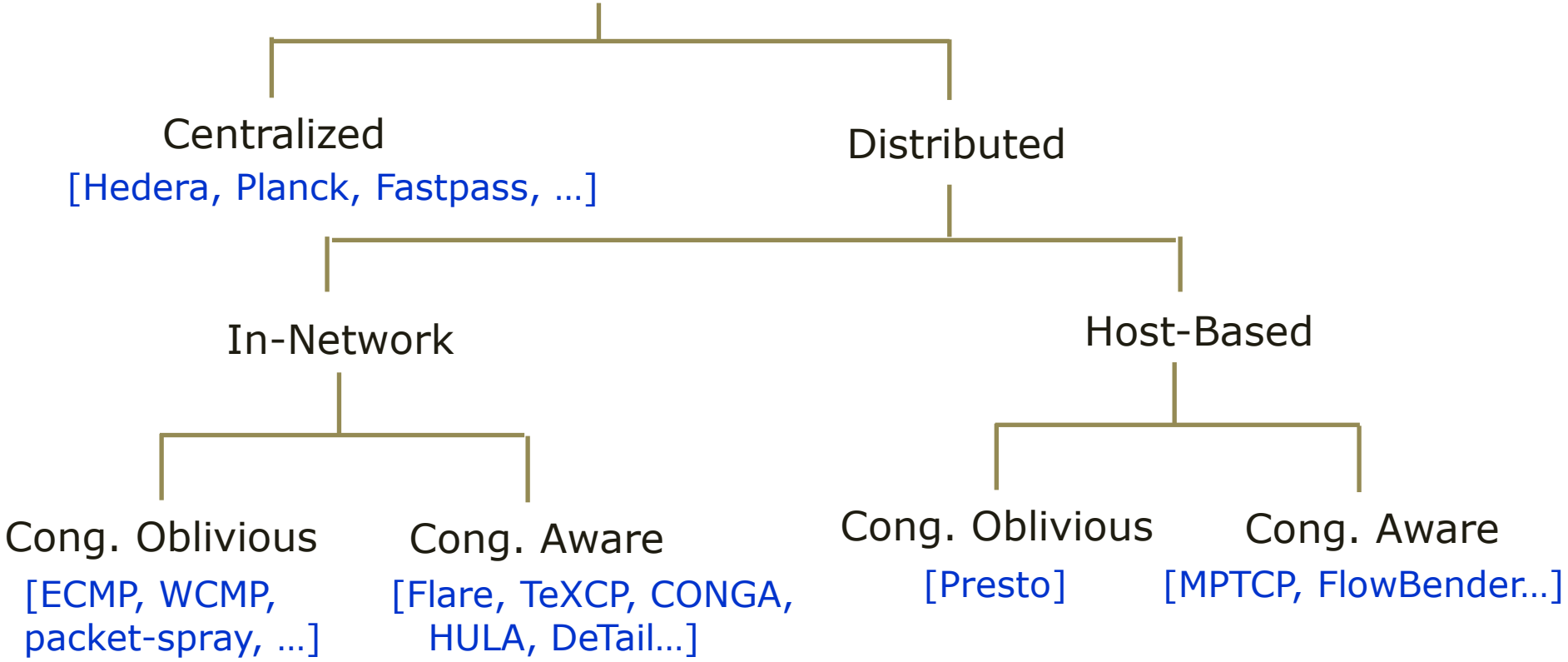



If only 50% of hosts are active, you'd like each host to be able to send at 2Gb/s, faster than one NIC can support.

# Presto

✧ Adapted from slides by Keqiang He (Wisconsin)

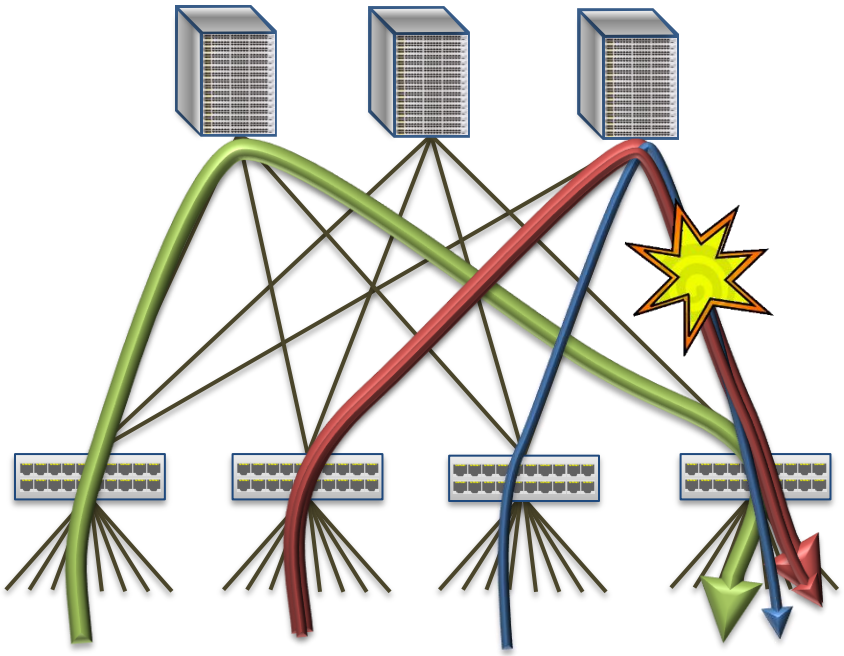
# Solution Landscape




Is congestion-aware load balancing overkill for datacenters?

# We've already seen a congestion-oblivious load balancing scheme

ECMP



Presto is **fine-grained** congestion-oblivious load balancing

Key challenge is making this practical

# Key Design Decisions

Use software edge

- No changes to transport (e.g., inside VMs) or switches

LB granularity *flowcells [e.g. 64KB of data]*

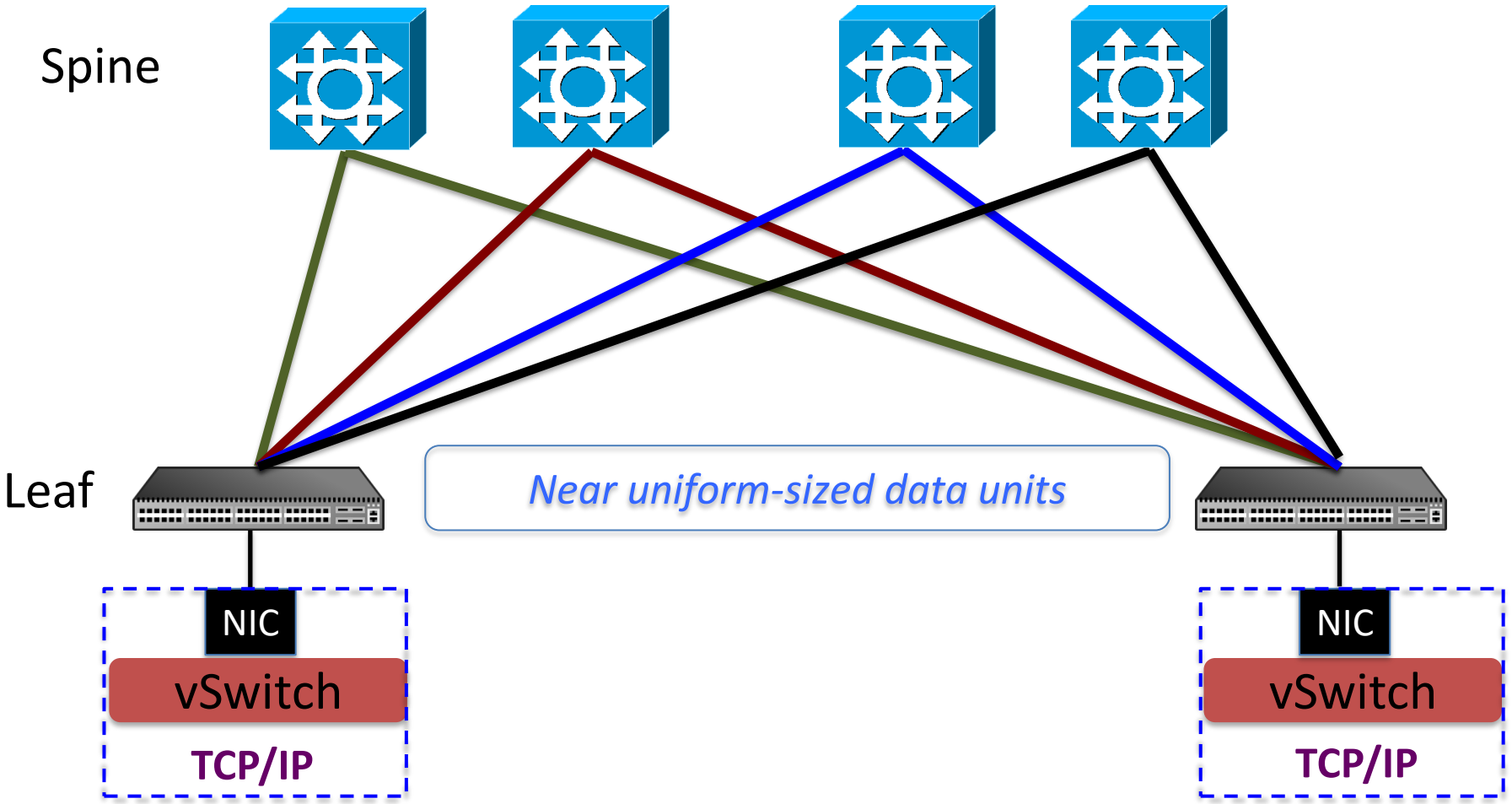
- Works with TSO (TSC Segmentation Offload) hardware offload
- No reordering for mice
- Makes dealing with reordering simpler (Why?)

"Fix" reordering at GRO (Generic Receive Offloading) layer

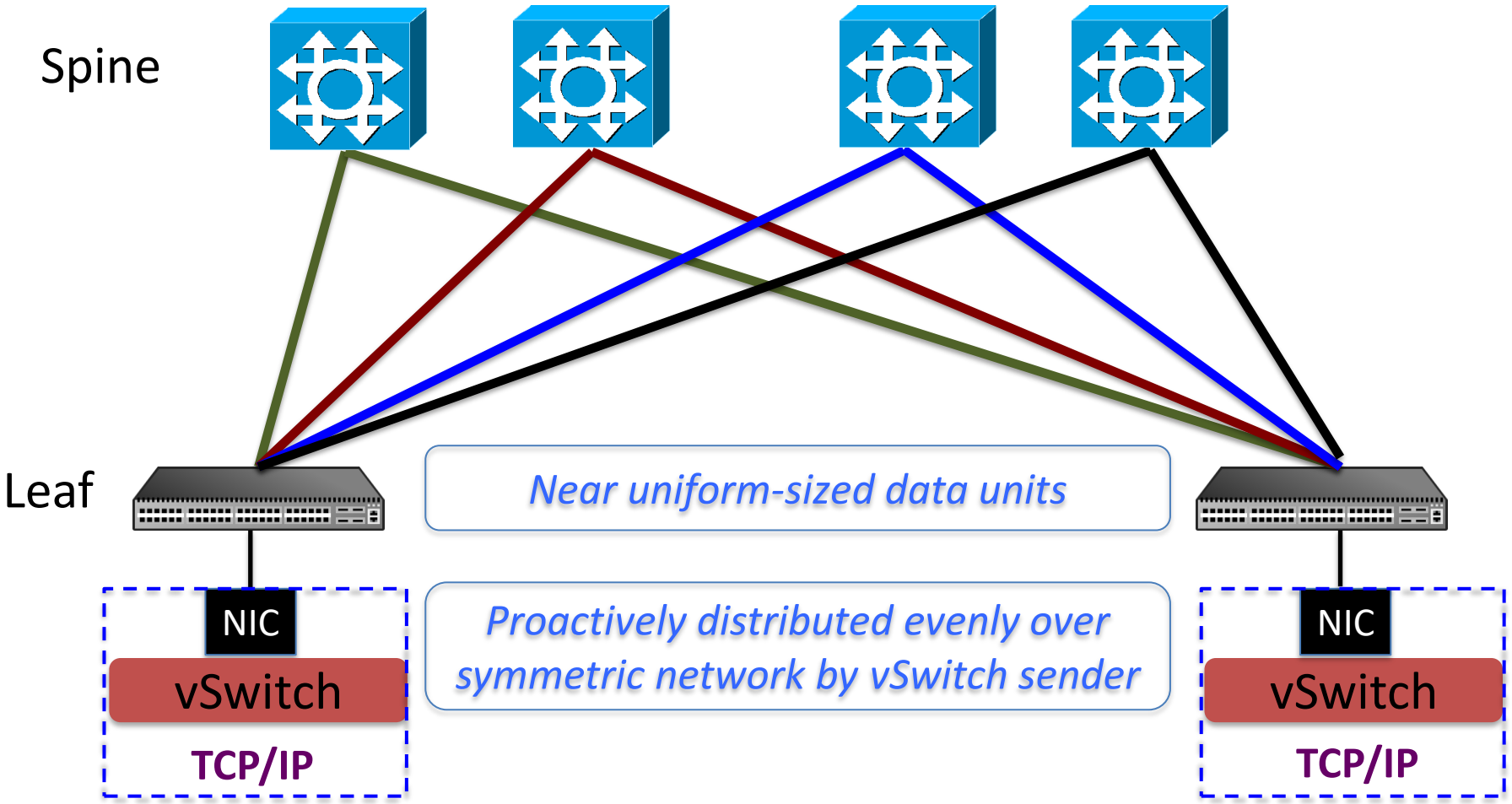
- Avoid high per-packet processing (esp. at 10Gb/s and above)

End-to-end path control

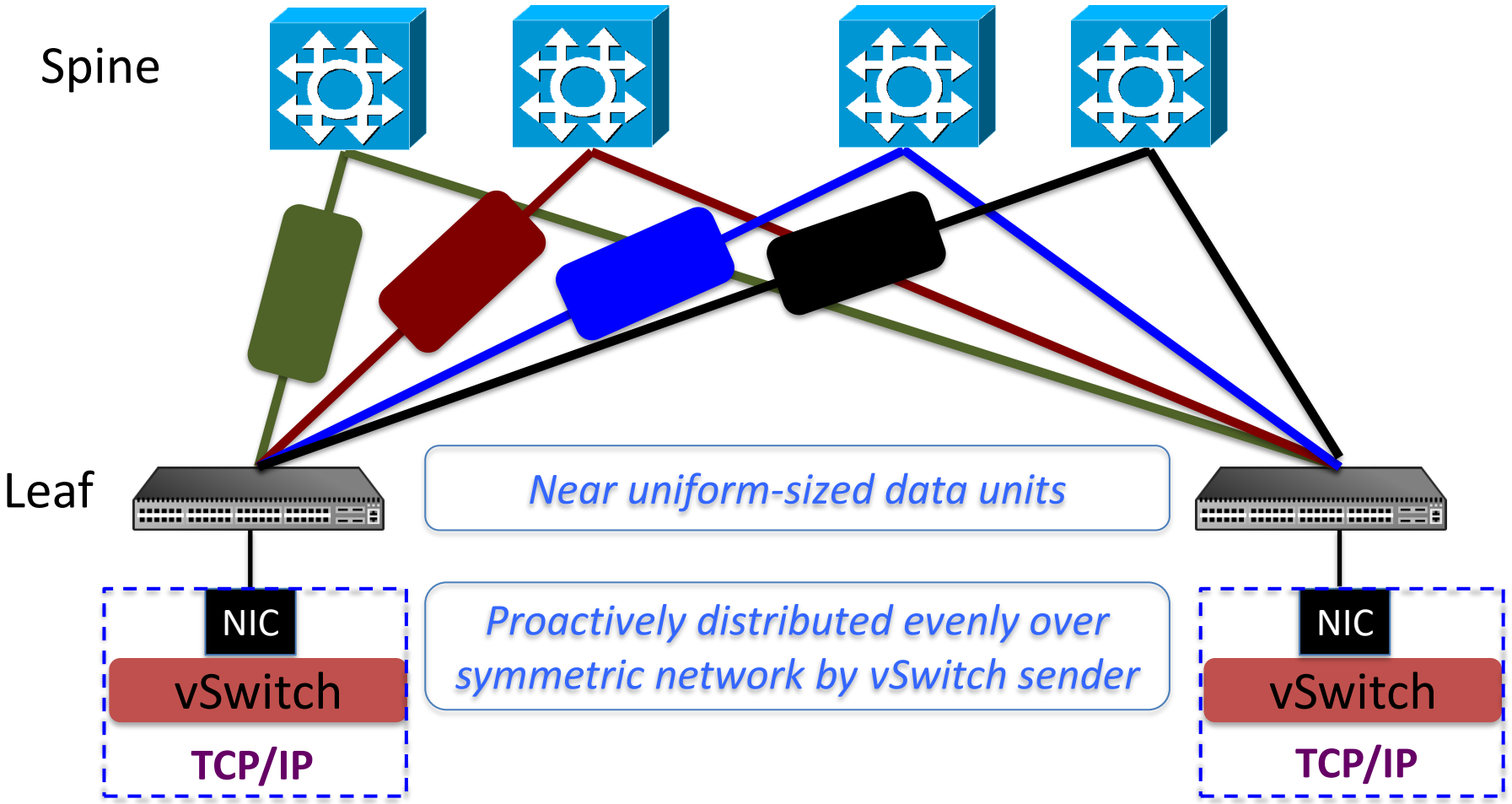
# Presto at a High Level



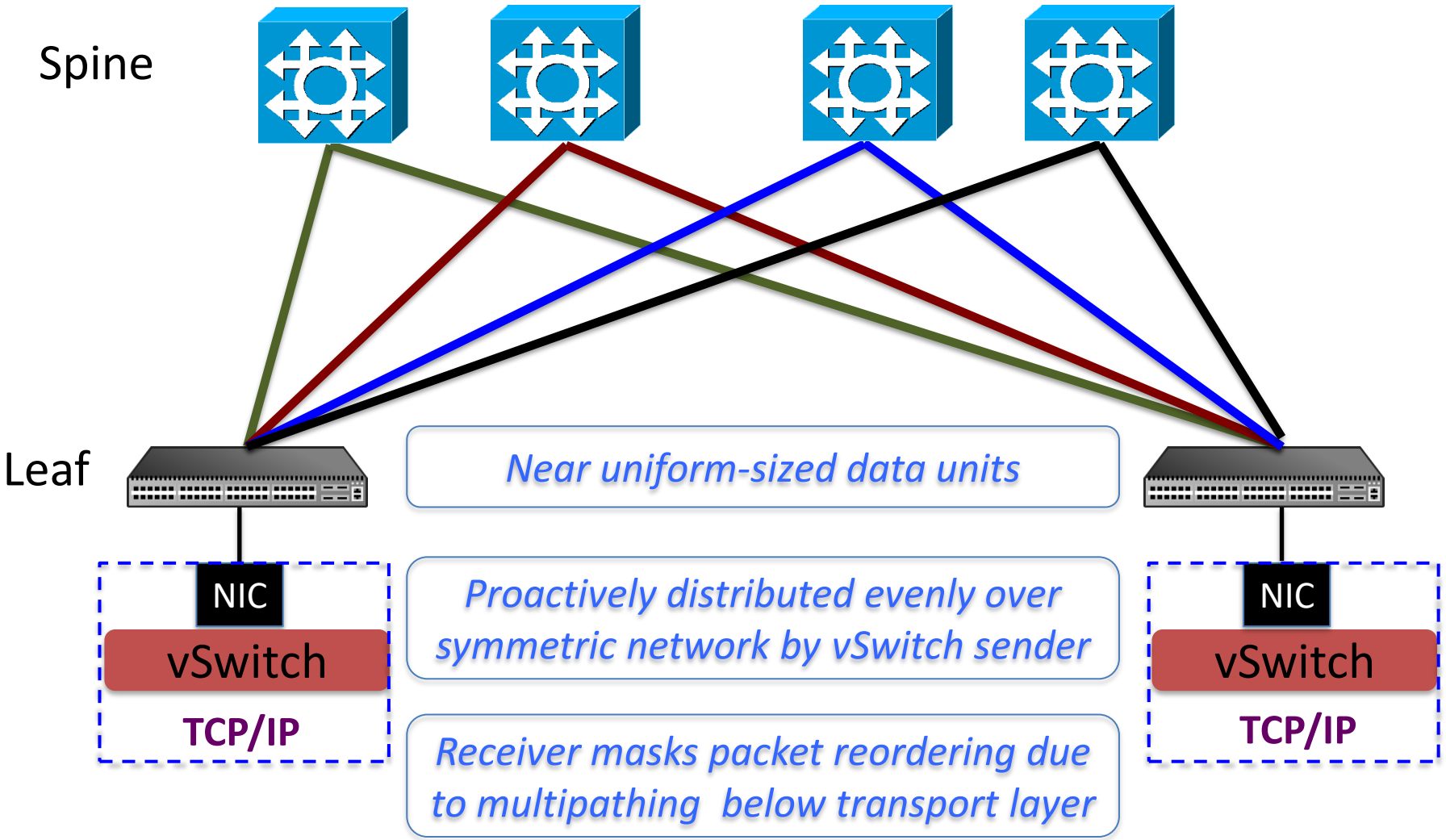
# Presto at a High Level



# Presto at a High Level



# Presto at a High Level



# Discussion

# What You Said

**Arman:** *"From an (information) theoretic perspective, order should not be such a troubling phenomenon, yet in real networks ordering is so important. How practical are "rateless codes" (network coding, raptor codes, etc.) in alleviating this problem?"*

**Amy:** *"The main complexities in the paper stem from the requirement that servers use TSO and GRO to achieve high throughput. It is surprising to me that people still rely so heavily on TSO and GRO. Why doesn't someone build a multi-core TCP stack that can process individual packets at line rate in software?"*

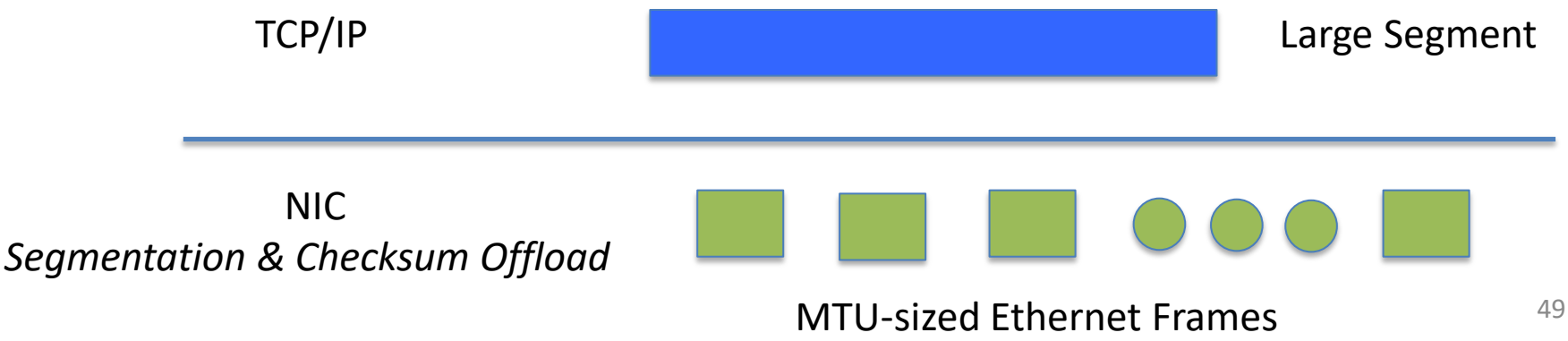
# Presto LB Granularity

Presto: load-balance on *flowcells*

What is flowcell?

- *A set of TCP segments with bounded byte count*
- Bound is maximal TCP Segmentation Offload (TSO) size
  - Maximize the benefit of TSO for high speed
  - *64KB* in implementation

What's TSO?



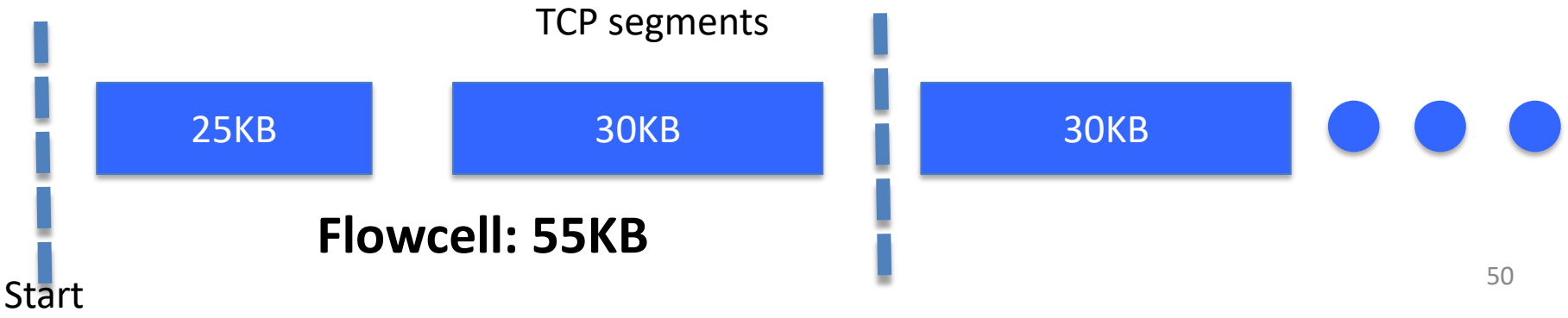
# Presto LB Granularity

Presto: load-balance on *flowcells*

What is flowcell?

- *A set of TCP segments with bounded byte count*
- Bound is maximal TCP Segmentation Offload (TSO) size
  - Maximize the benefit of TSO for high speed
  - *64KB* in implementation

Examples

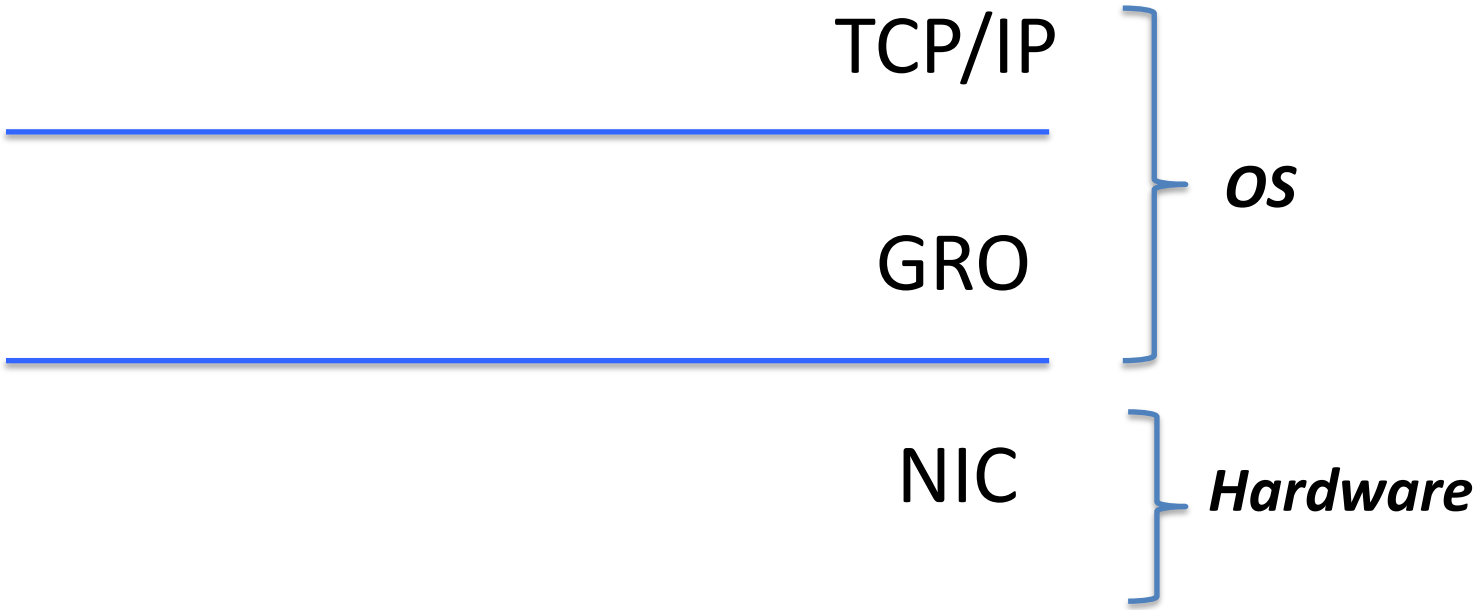


# Intro to GRO

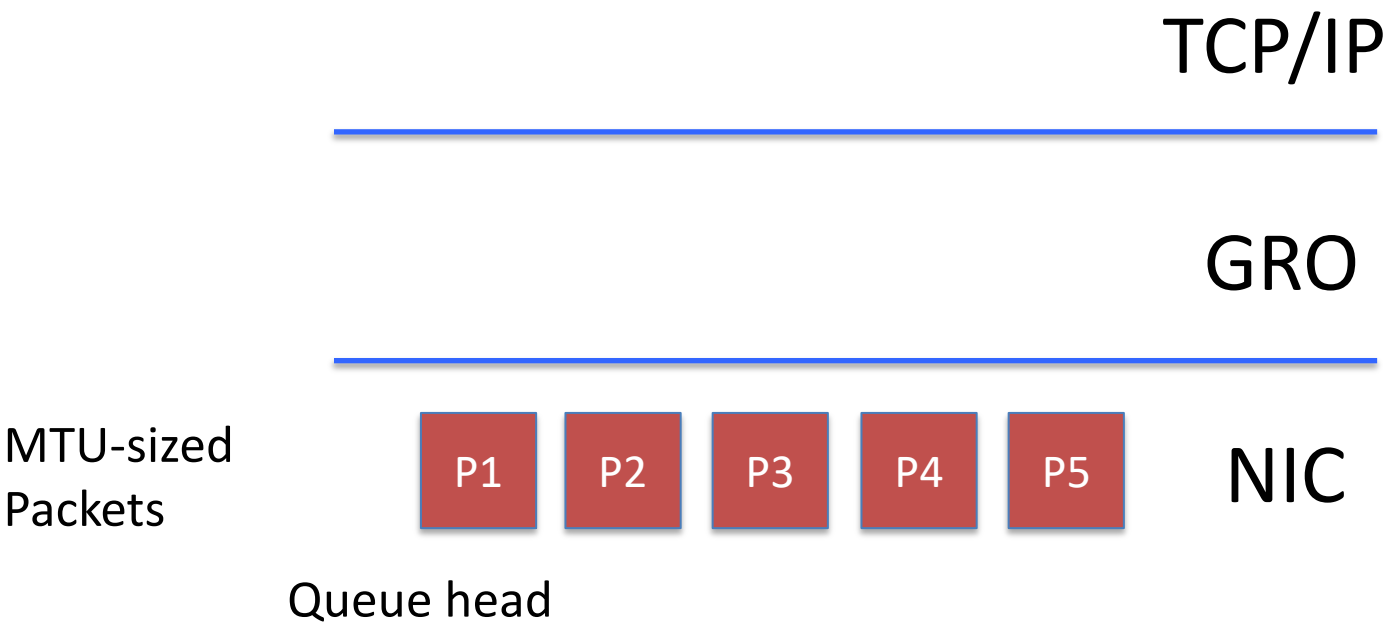
Generic Receive Offload (GRO)

- The reverse process of TSO

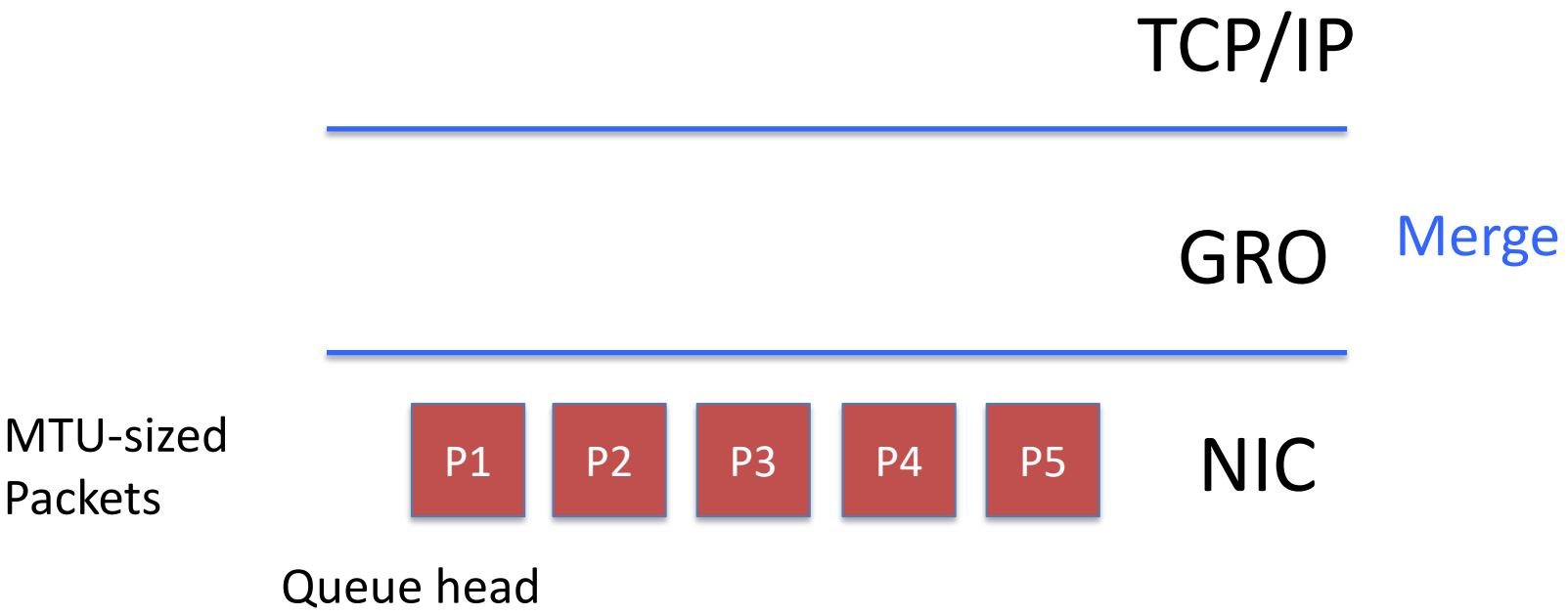
# Intro to GRO



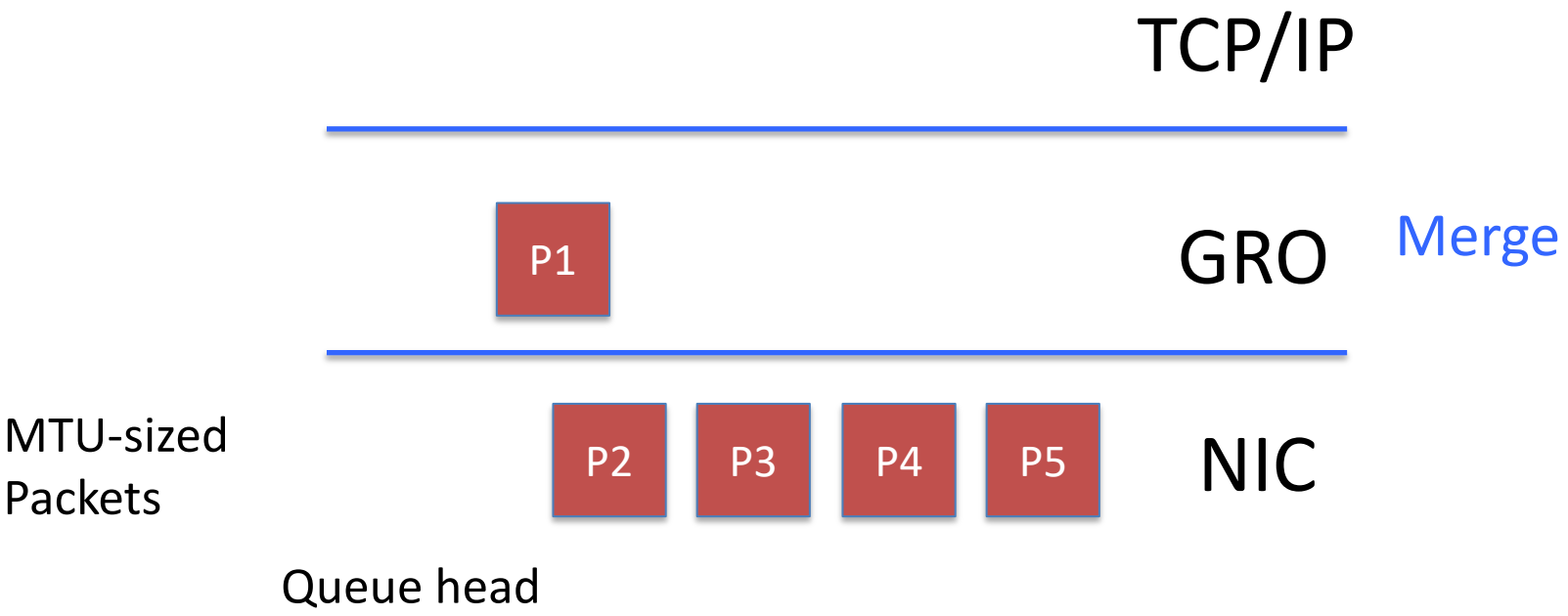
# Intro to GRO



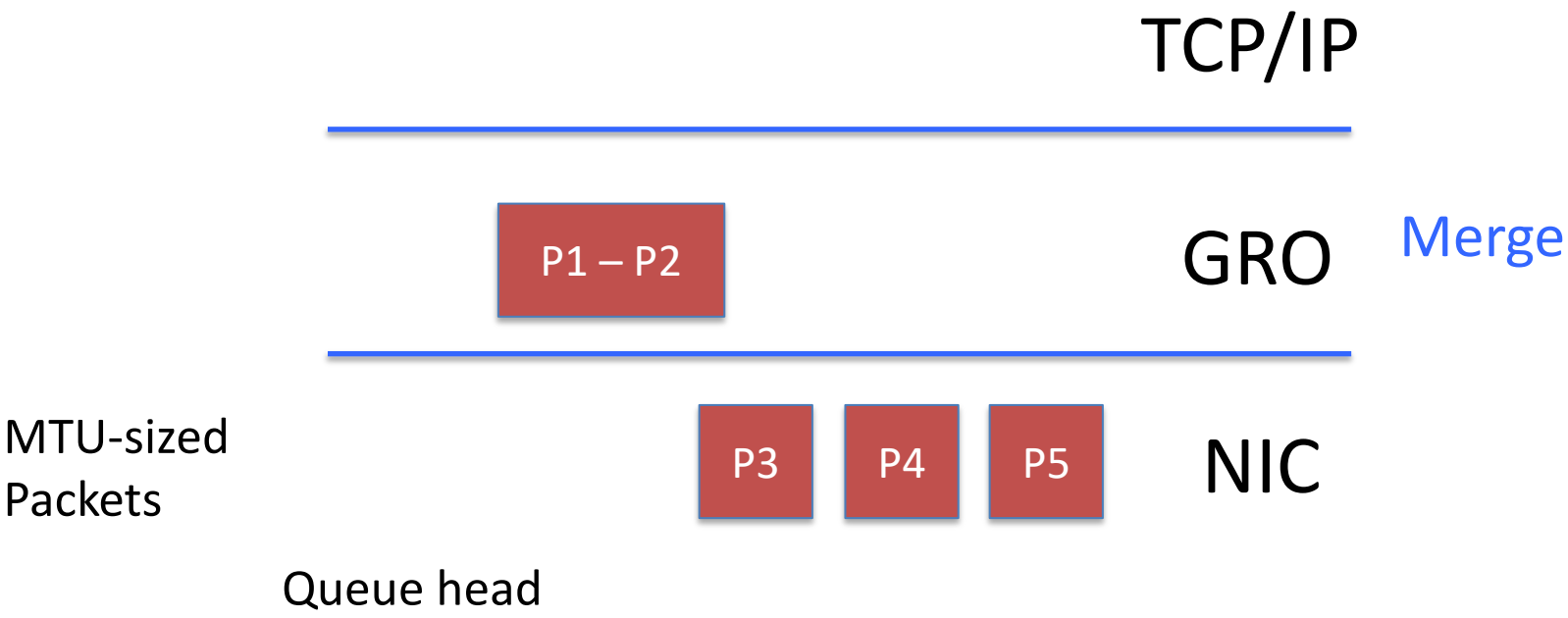
# Intro to GRO


TCP/IP
GRO
Merge
MTU-sized
Packets
P1
P2
P3
P4
P5
NIC
Queue head

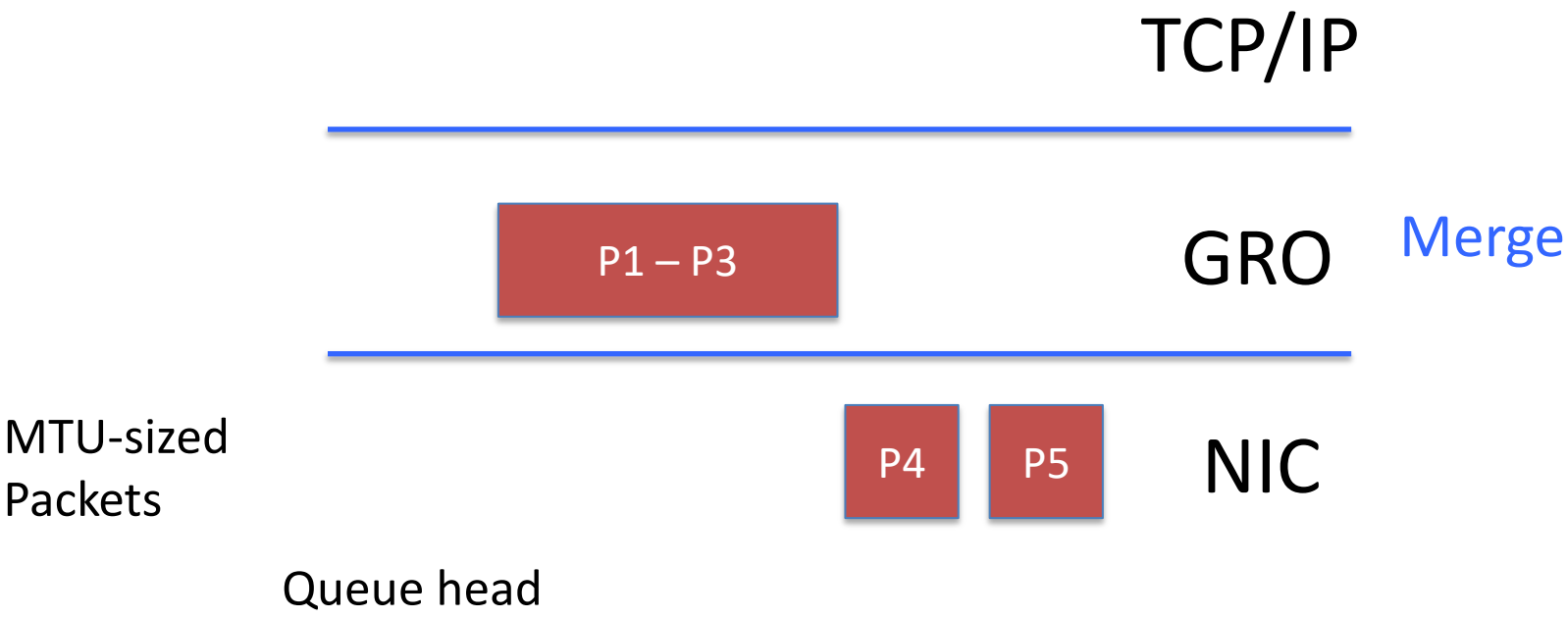
# Intro to GRO


TCP/IP
P1
GRO
Merge
MTU-sized
Packets
P2
P3
P4
P5
NIC
Queue head

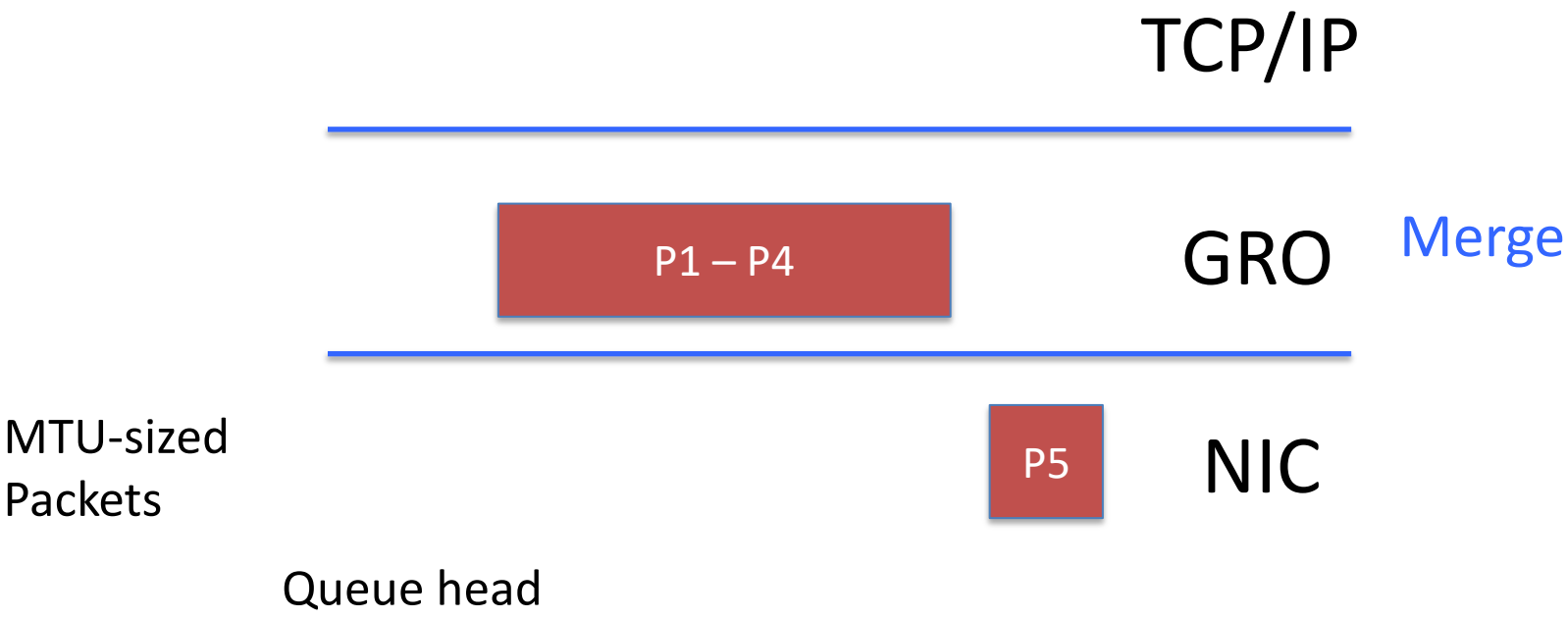
# Intro to GRO


TCP/IP
P1 – P2
GRO
Merge
MTU-sized
Packets
P3
P4
P5
NIC
Queue head

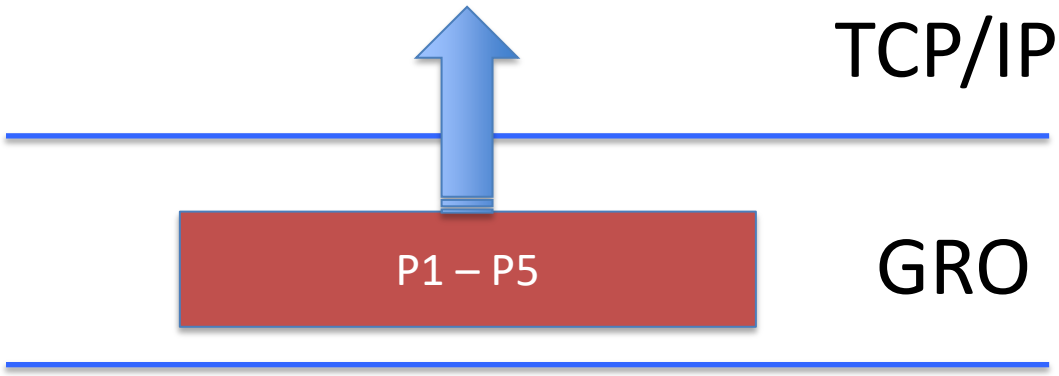
# Intro to GRO



# Intro to GRO



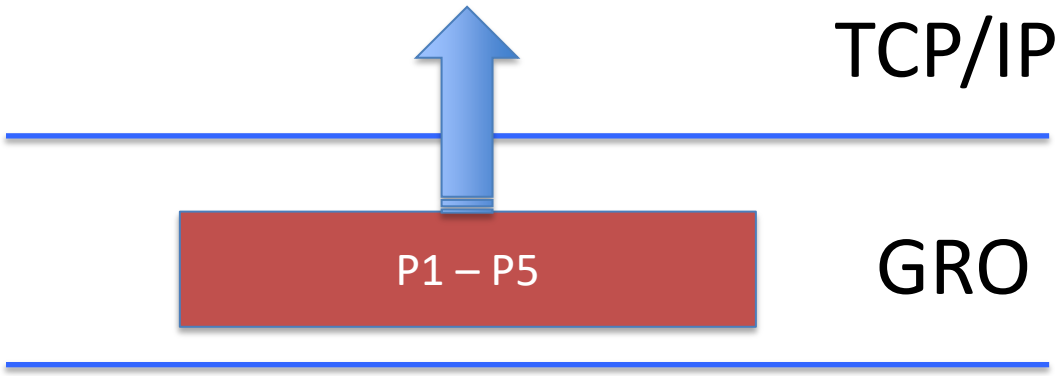
# Intro to GRO




MTU-sized
Packets

*Large TCP segments are pushed-up at the end of a batched IO event (i.e., a polling event)*

# Intro to GRO

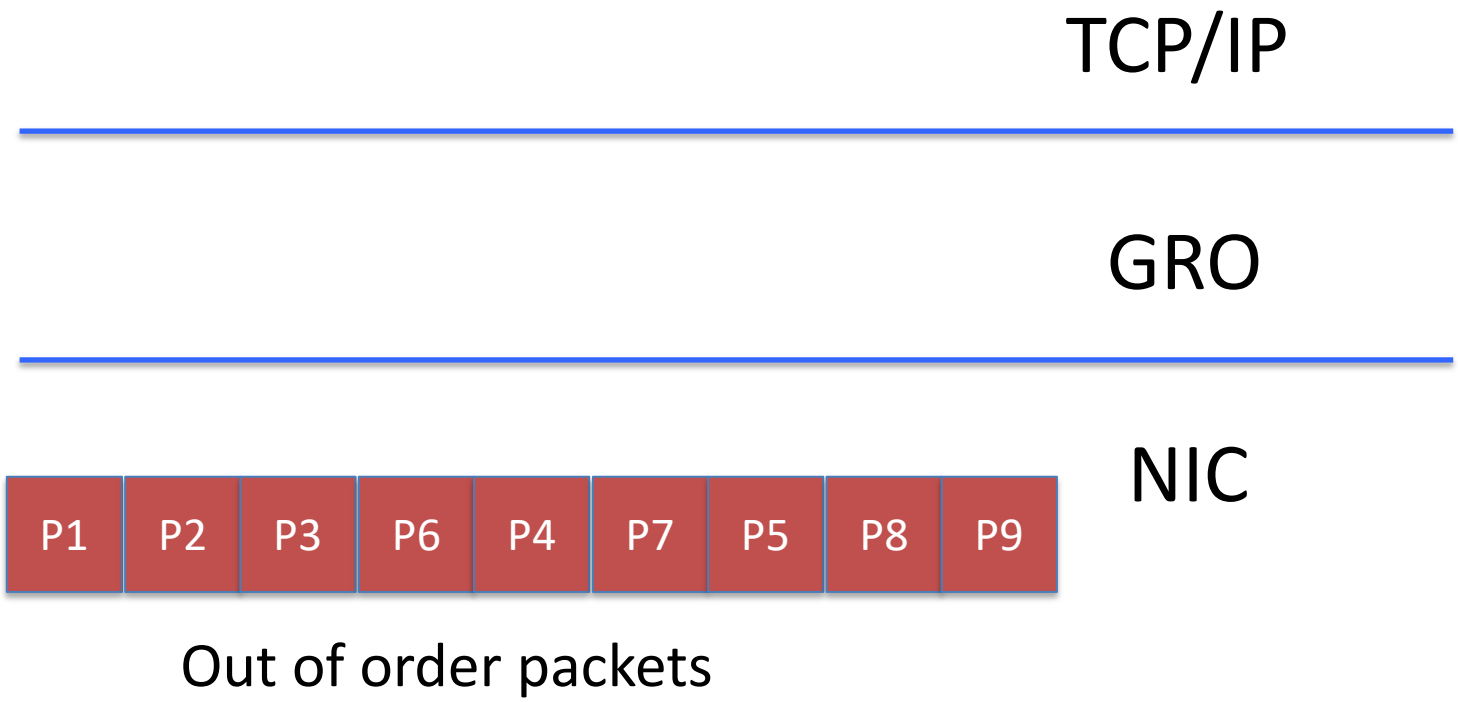



MTU-sized
Packets

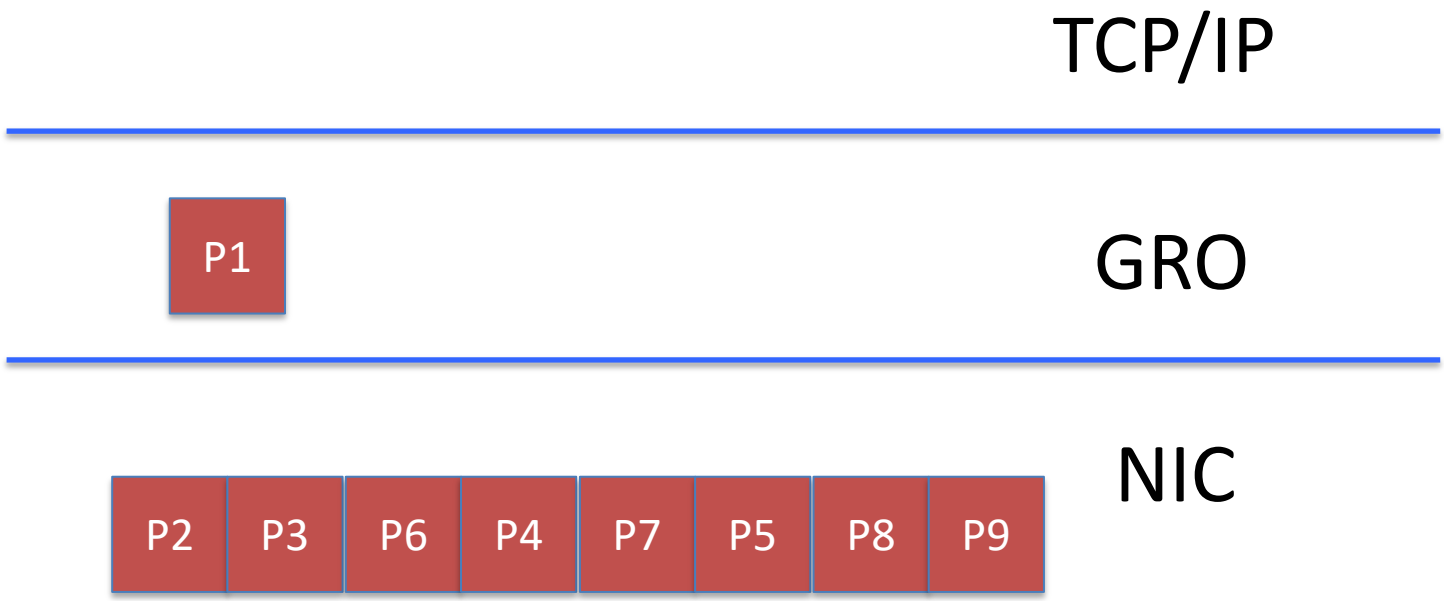
*Merging pkts in GRO creates less segments & avoids using substantially more cycles at TCP/IP and above [Menon, ATC'08]*

If GRO is disabled, ~6Gbps with 100% CPU usage of one core

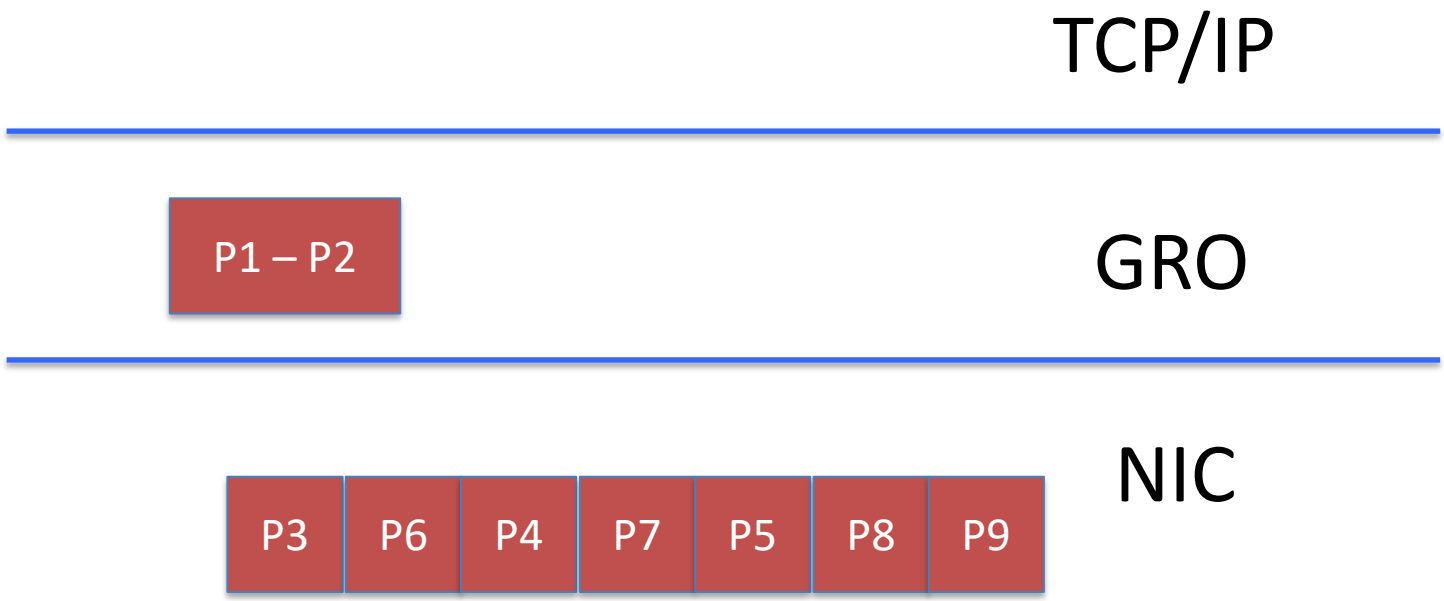
# Reordering Challenges



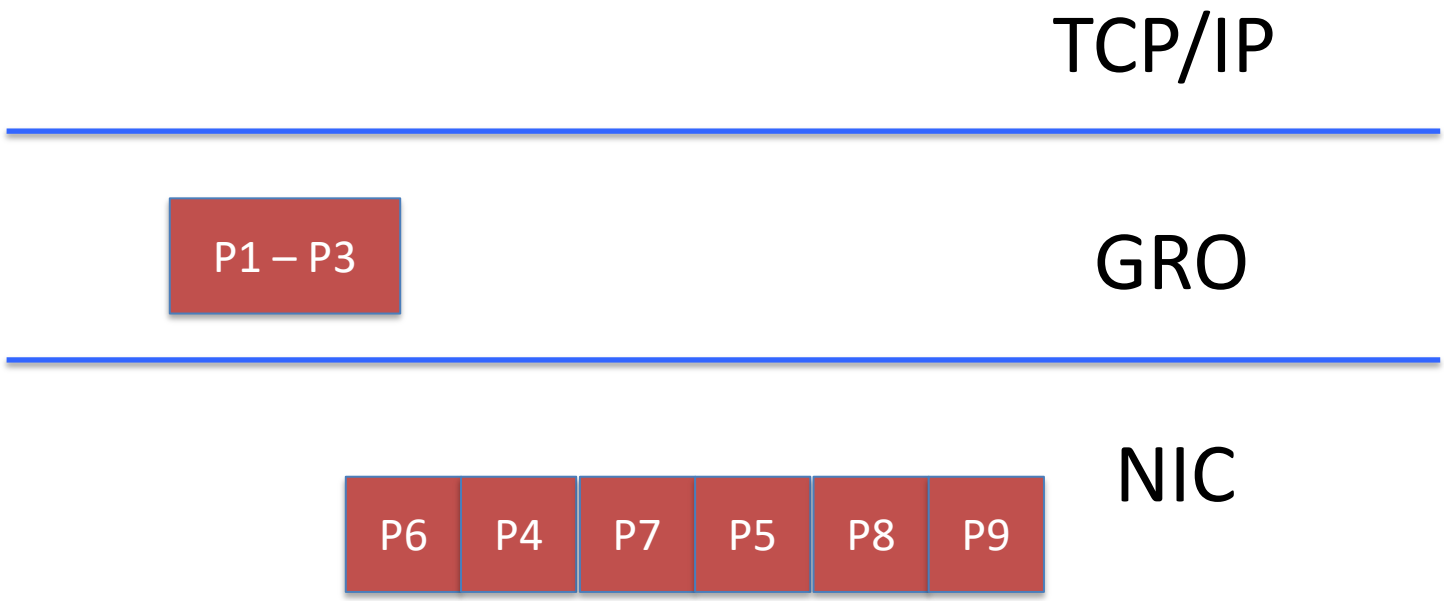
# Reordering Challenges



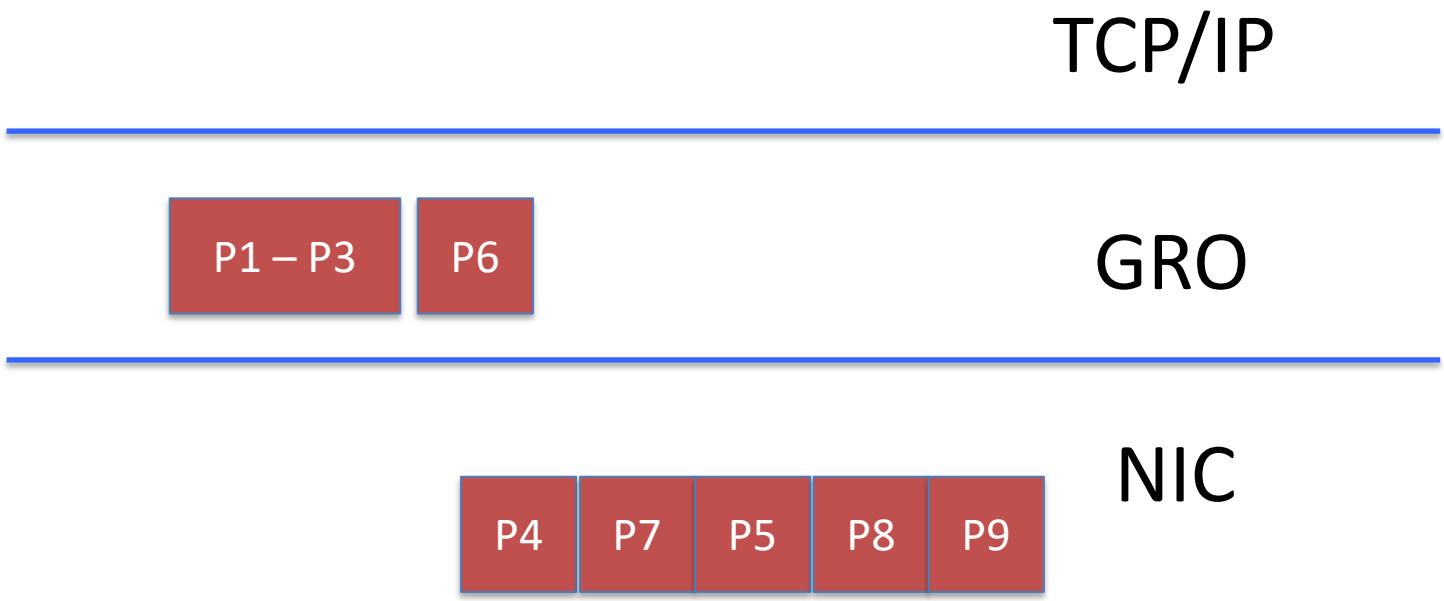
# Reordering Challenges



# Reordering Challenges

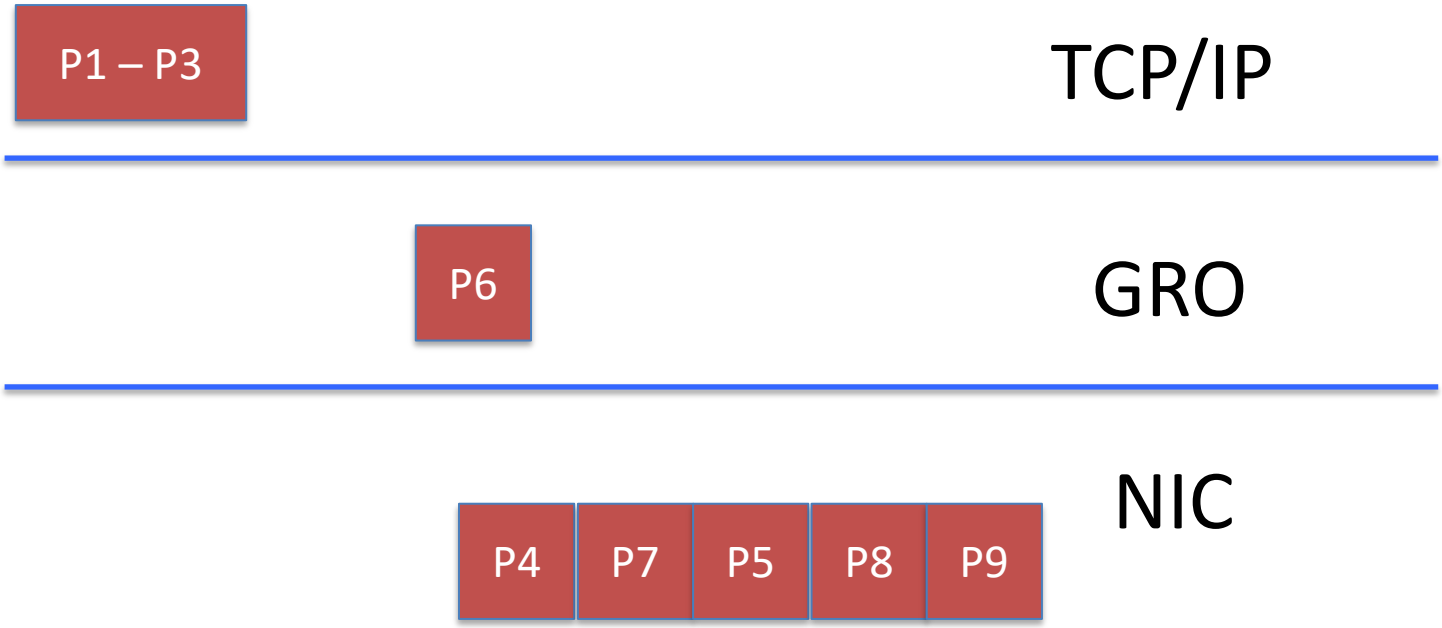


# Reordering Challenges

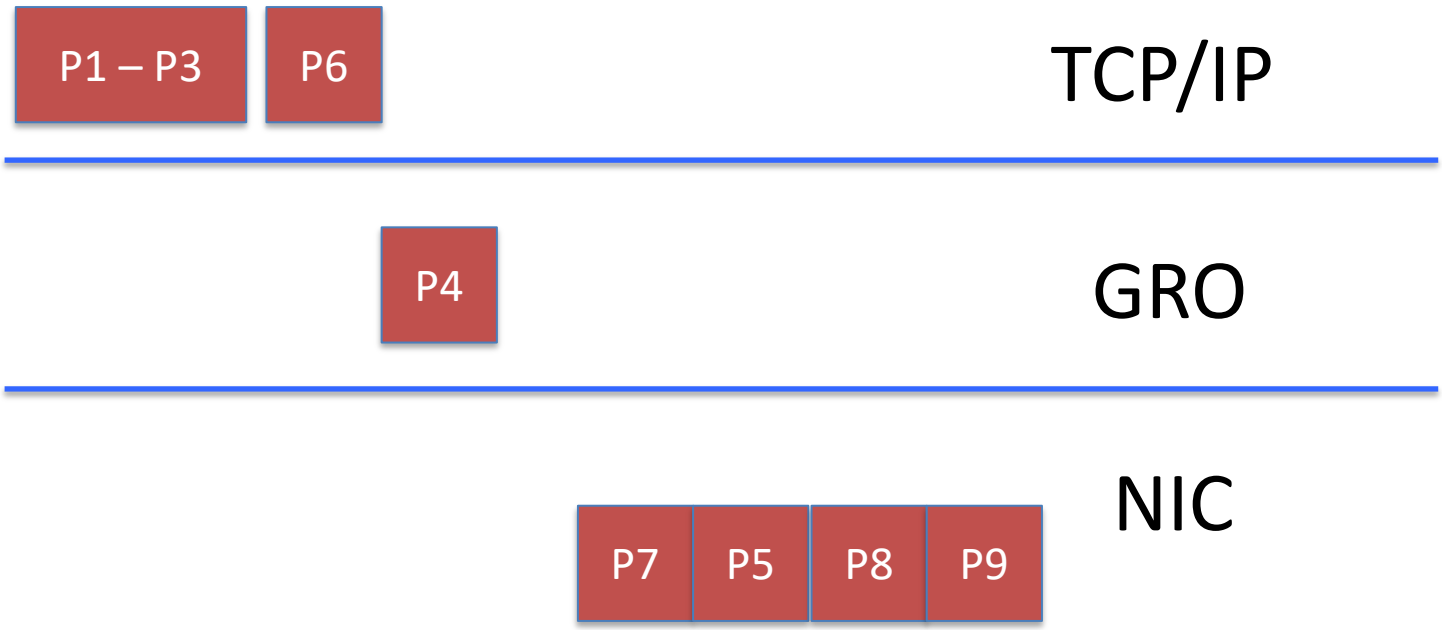



GRO is designed to be fast and simple; it pushes-up the existing segment immediately when 1) there is a gap in sequence number, 2) MSS reached or 3) timeout fired

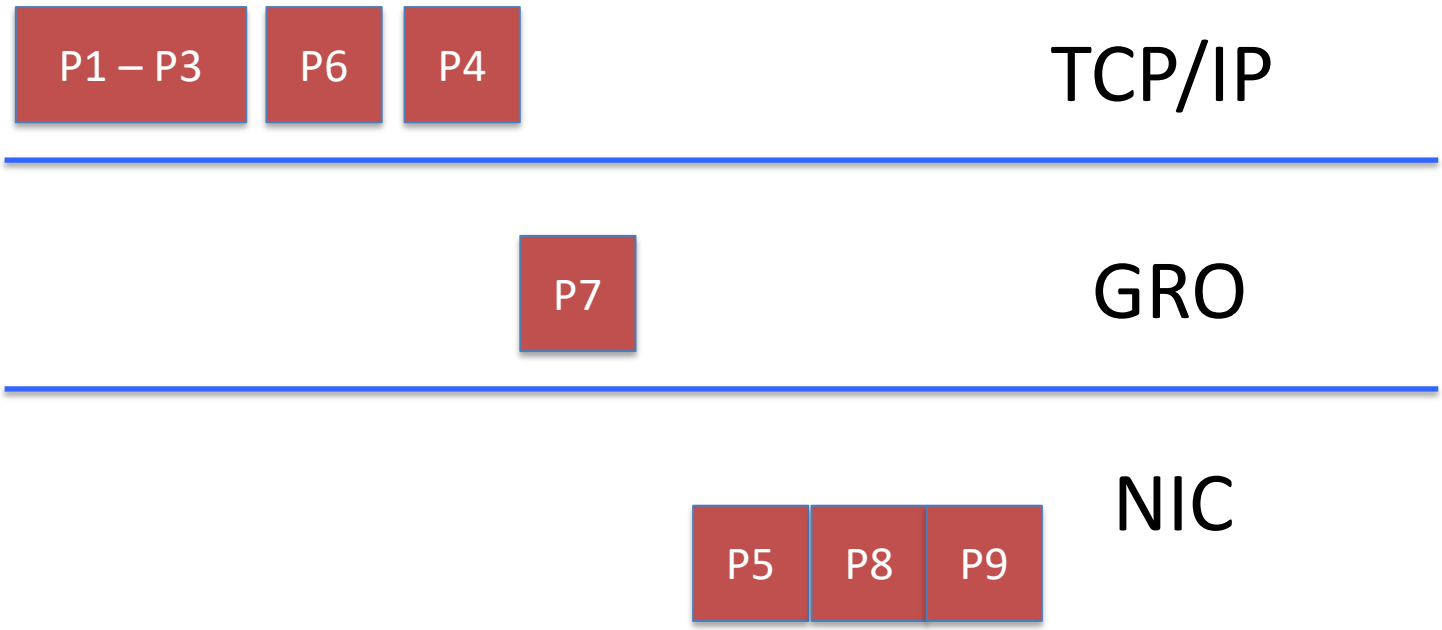
# Reordering Challenges



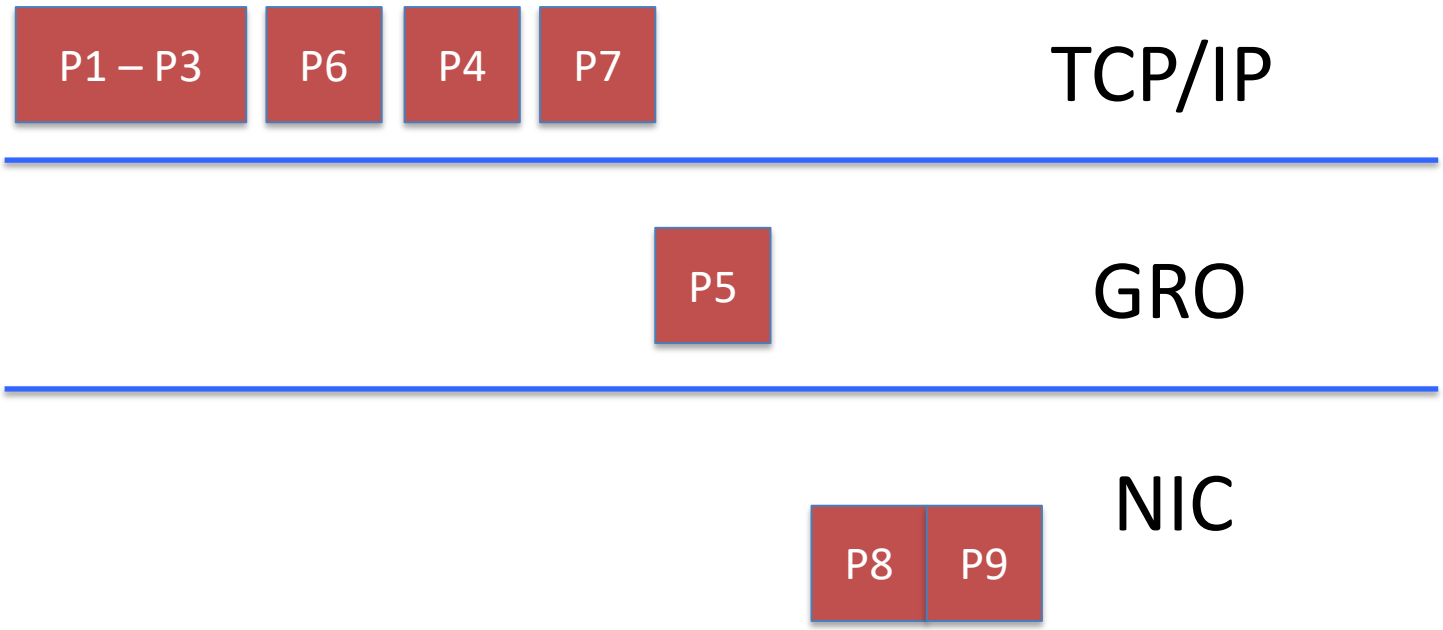
# Reordering Challenges



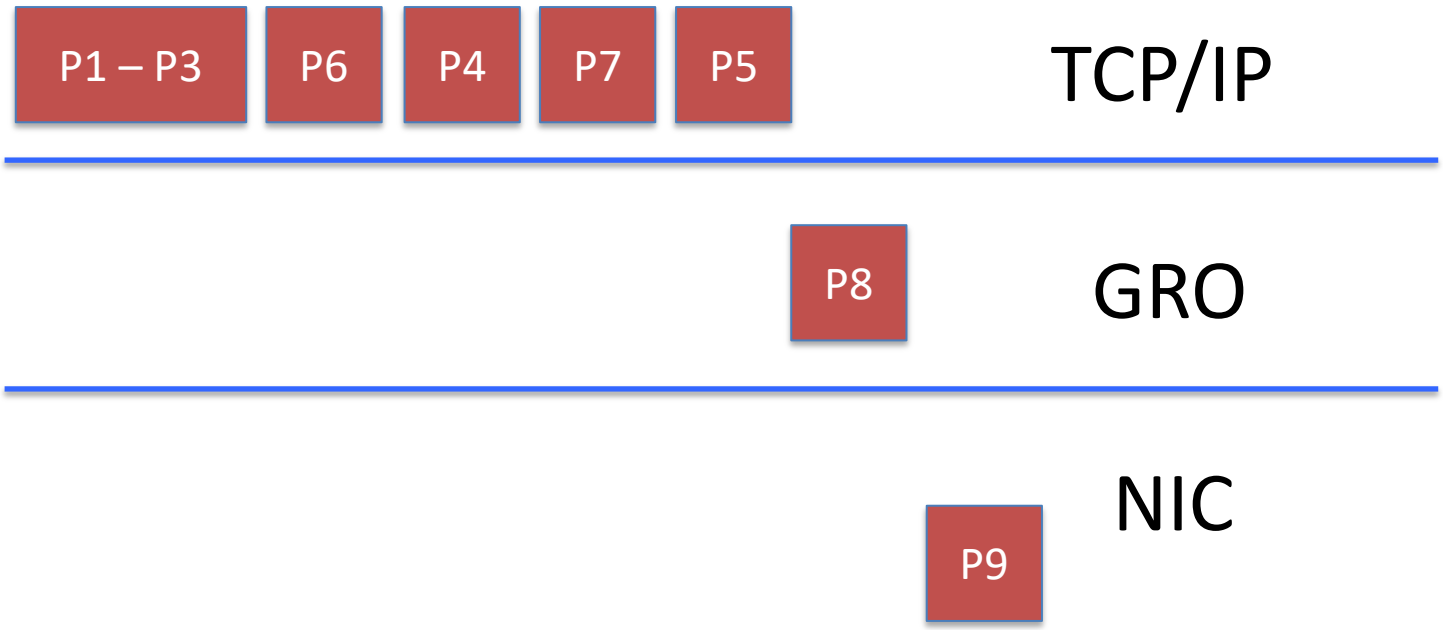
# Reordering Challenges



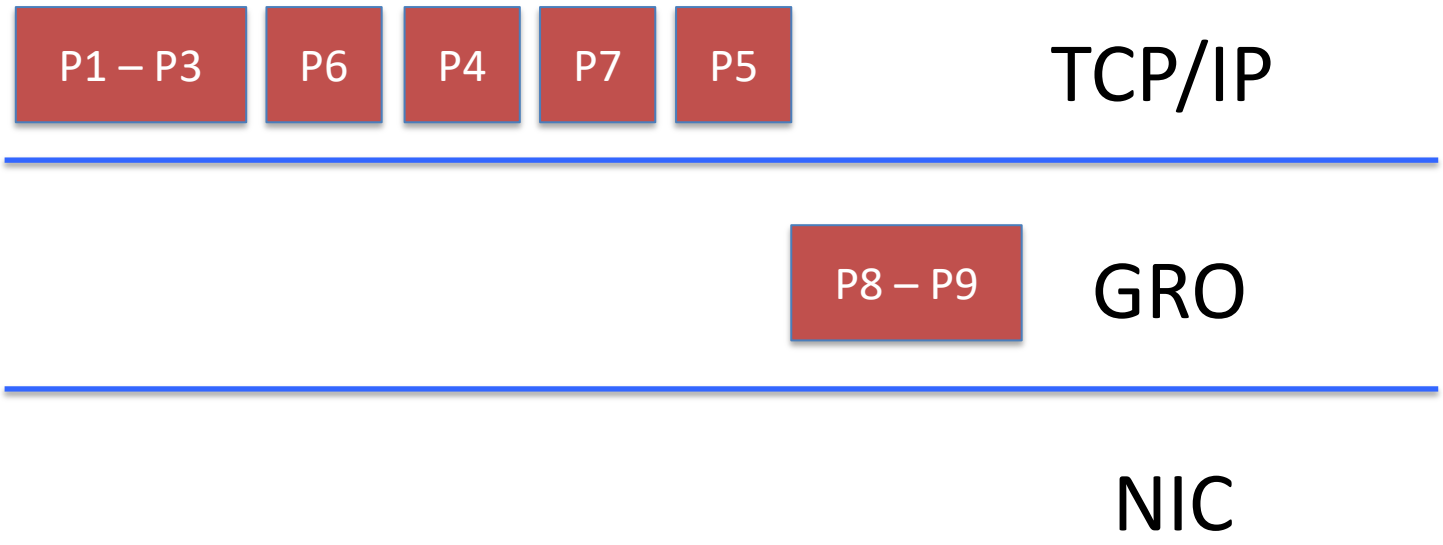
# Reordering Challenges



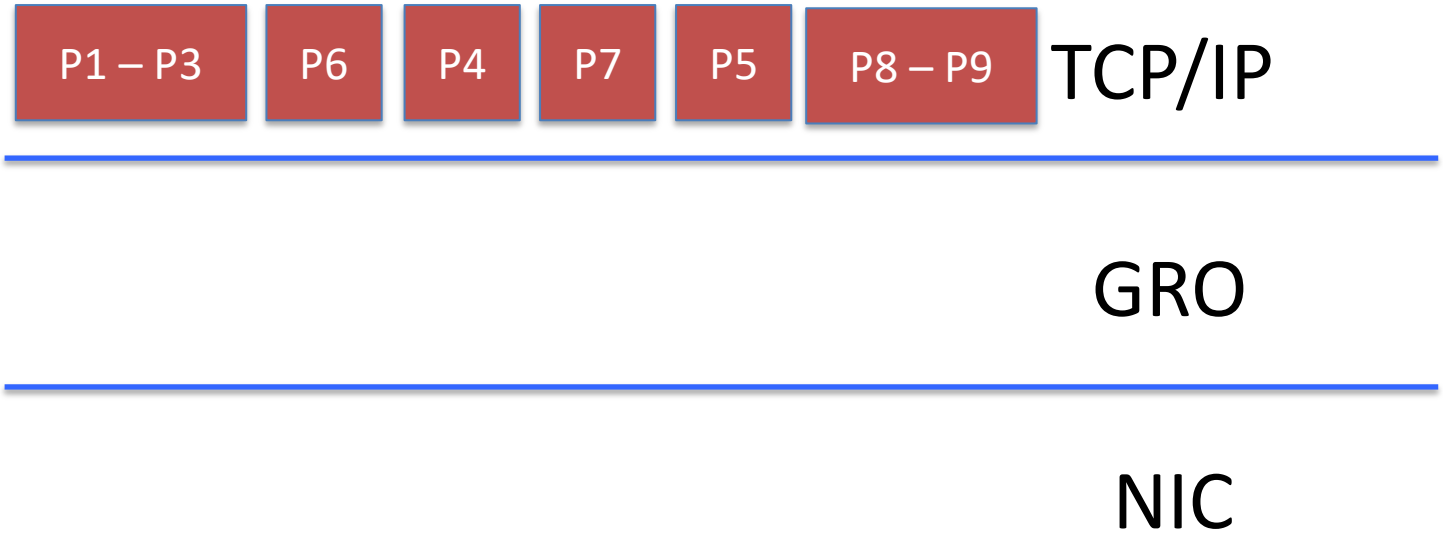
# Reordering Challenges



# Reordering Challenges



# Reordering Challenges



# Reordering Challenges

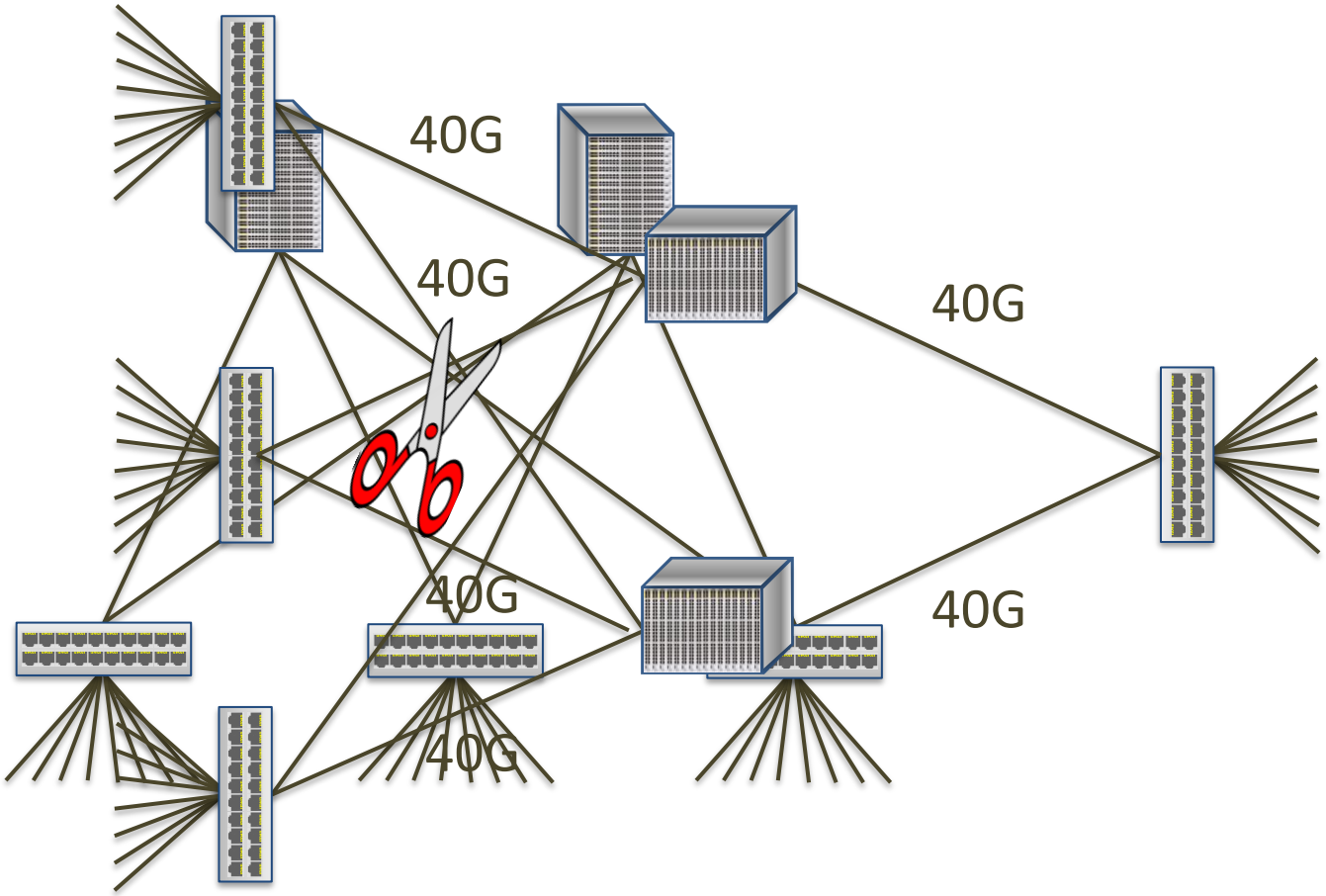
***GRO is effectively disabled***
***Lots of small packets are pushed up to TCP/IP***

Huge CPU processing overhead

Poor TCP performance due to massive reordering

# Handling Asymmetry

Handling asymmetry optimally needs traffic awareness



# Handling Asymmetry

Handling asymmetry optimally needs traffic awareness