

# Datacenter Networks

CPSC 433/533, Spring 2021

Anurag Khandelwal

# What you should understand...

- **What is a datacenter network**
  - Scale, service-model, application characteristics
- **What makes it different?**
  - Characteristics, goals (w.r.t. Internet), degrees of freedom
- **How do we achieve goals by exploiting freedom?**
  - Topology redesign, L2/L3 redesign, L4 redesign
  - We will look at some approaches, not all

# What you should understand...

- **What is a datacenter network**
  - Scale, service-model, application characteristics
- **What makes it different?**
  - Characteristics, goals (w.r.t. Internet), degrees of freedom
- **How do we achieve goals by exploiting freedom?**
  - Topology redesign, L2/L3 redesign, L4 redesign
  - We will look at some approaches, not all

# Key Features of a DCN

- Scale
- Service model
  - Public clouds (jargon: SaaS, PaaS, DaaS, IaaS, ...)
  - Multi-tenancy
- Application characteristics
  - Large-scale computations (“big data”)
  - Customer-facing, revenue generating services

# Key Features of a DCN

- Scale
- Service model
  - *Public* clouds (jargon: SaaS, PaaS, DaaS, IaaS, ...)
  - Multi-tenancy
- Application characteristics
  - Large-scale computations (“big data”)
  - Customer-facing, revenue generating services

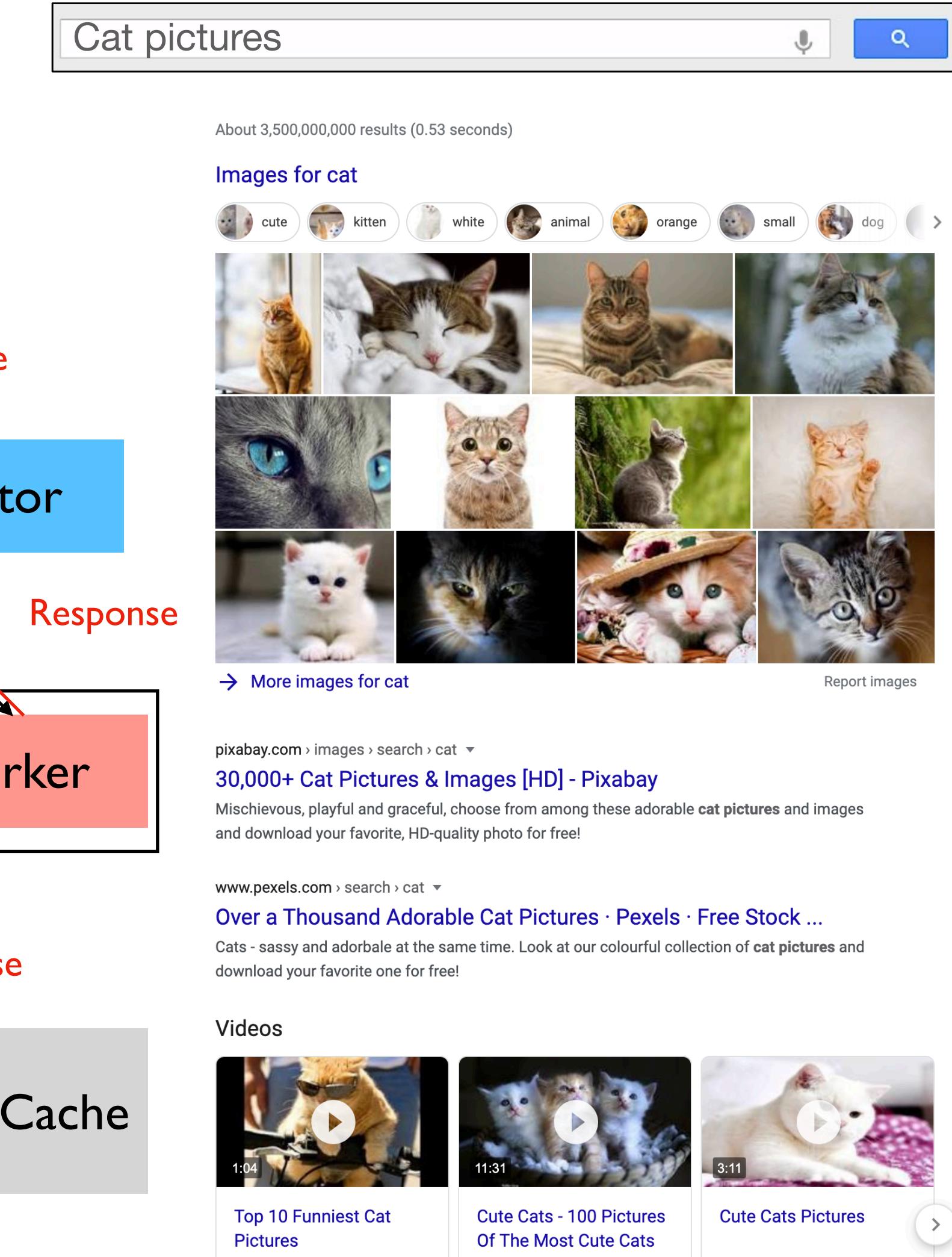
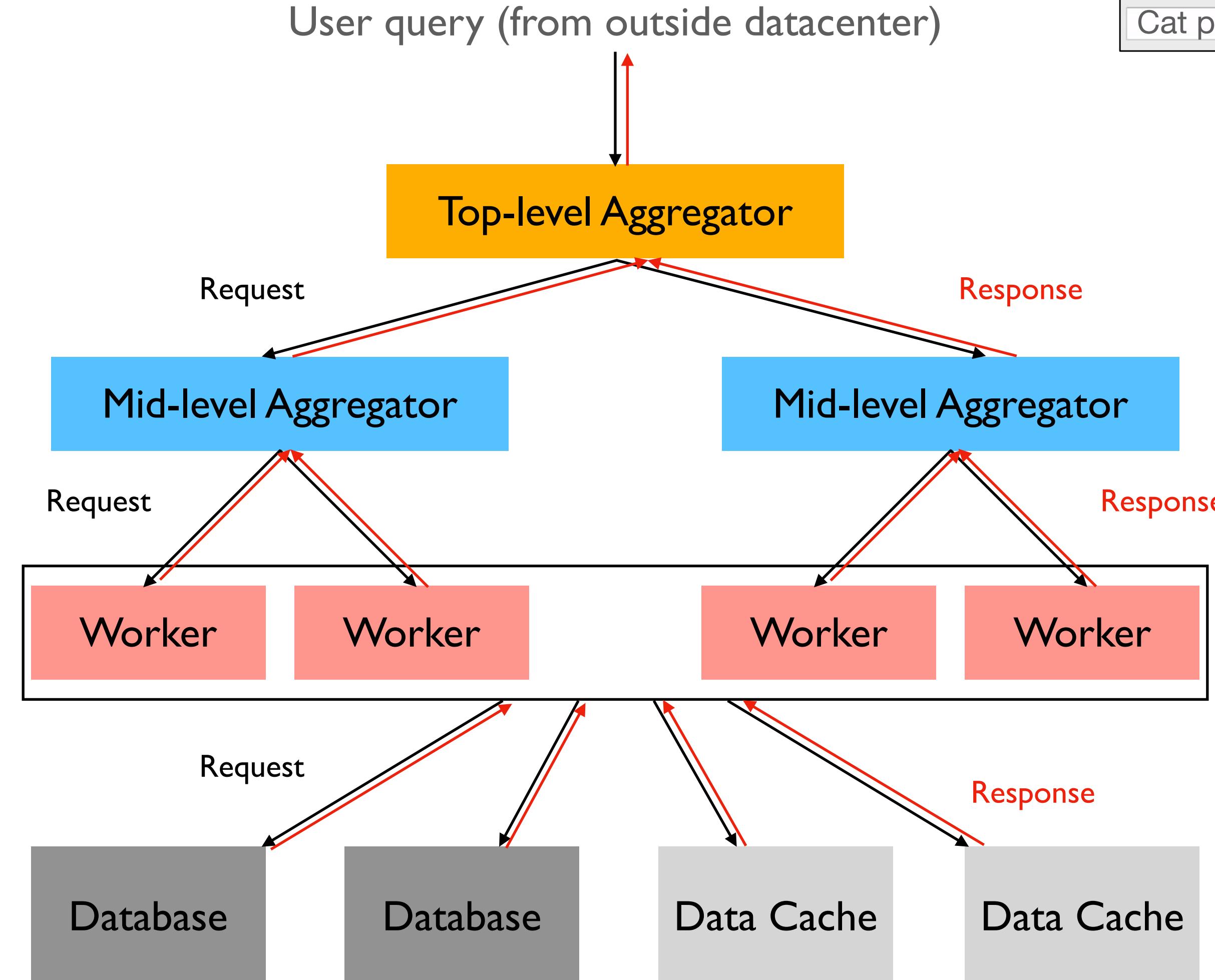
# Application Characteristics

- Common Theme: **Parallelism**
  - Application decomposed into tasks
  - Running in parallel on different servers
- Two common paradigms (not exhaustive)
  - Partition-Aggregate
  - Map-Reduce

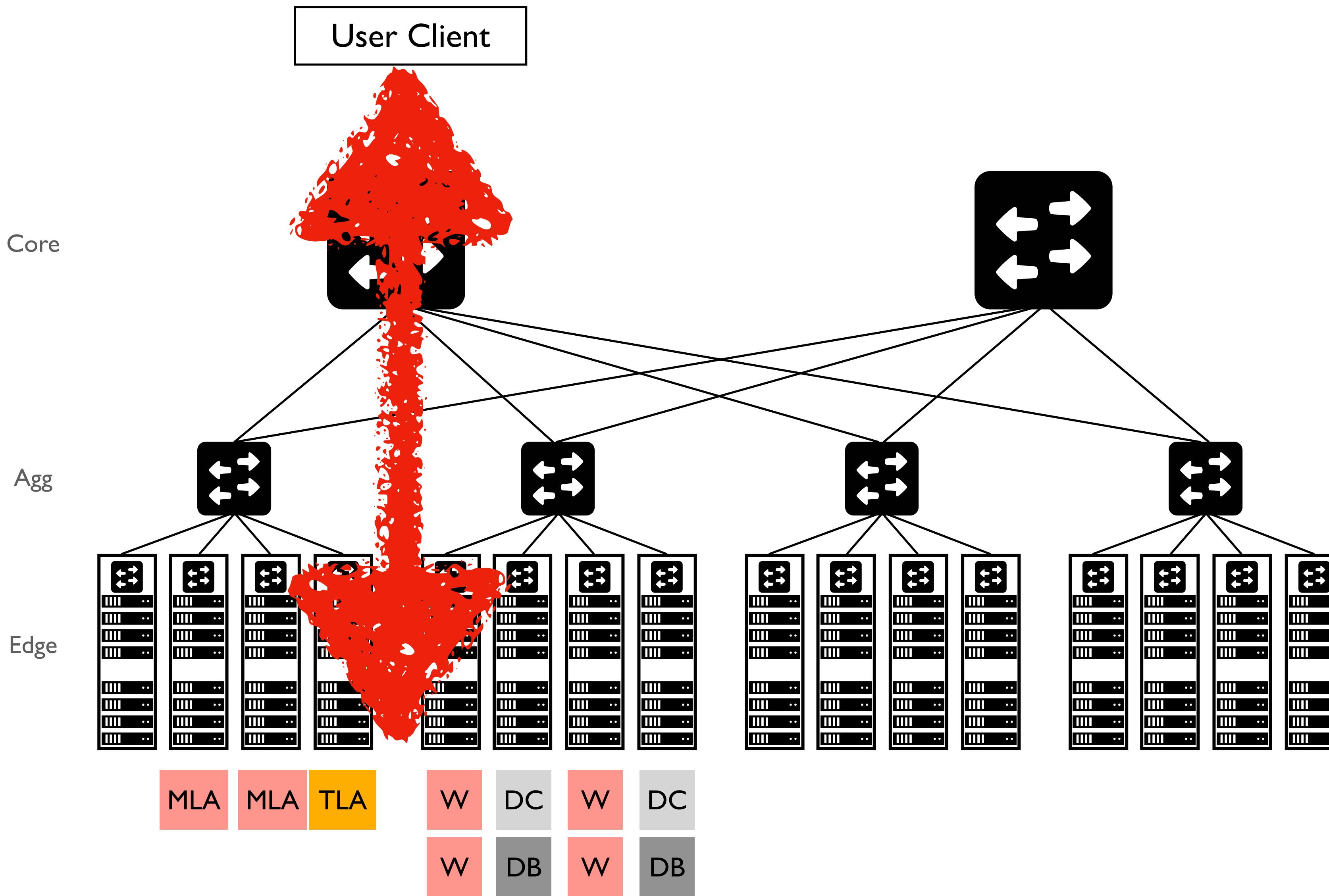
# Workload: Partition Aggregate

Most communications over network

Index

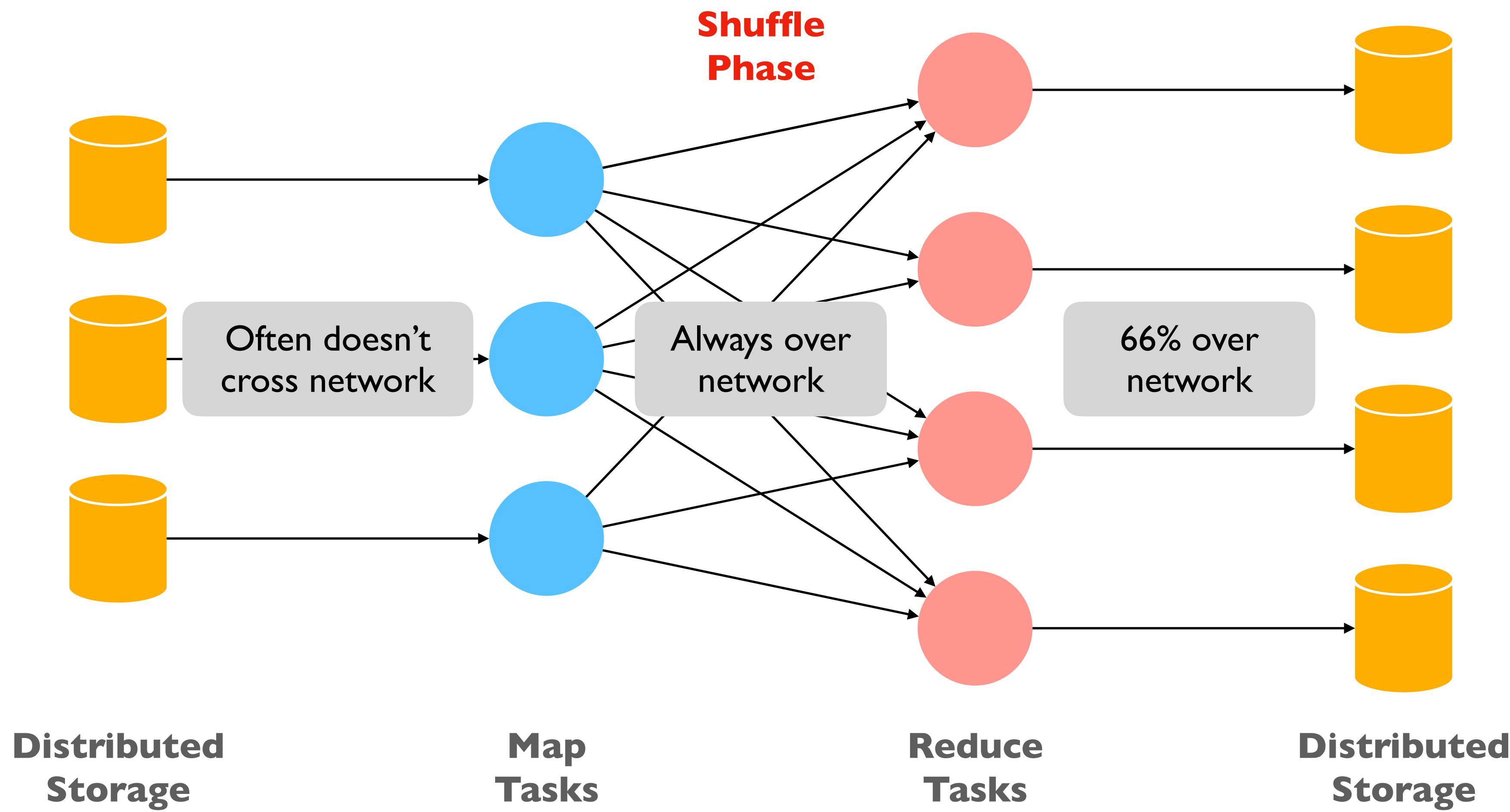


# “North-South” Traffic

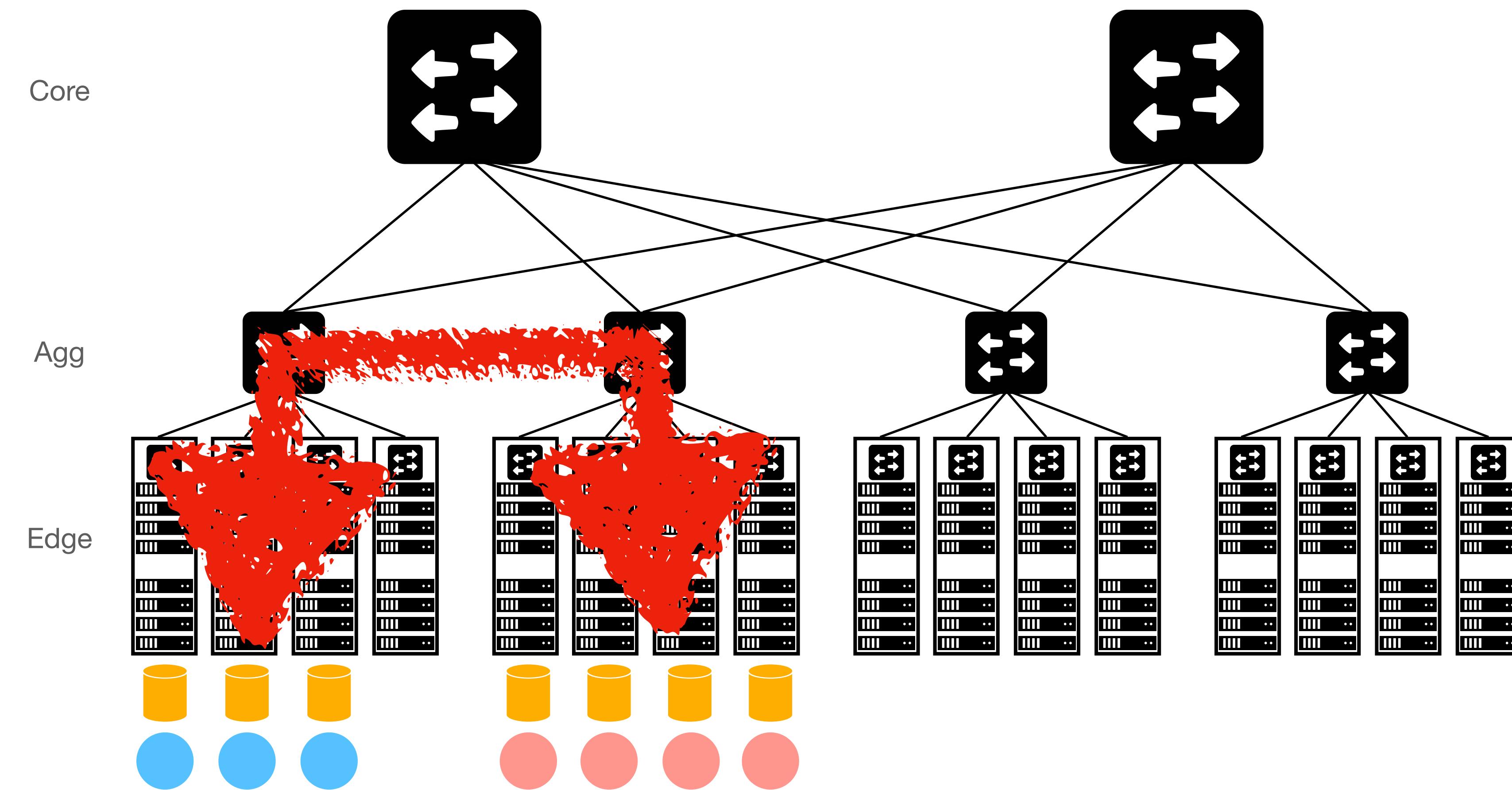


- Interactive query-response exchange b/w external clients & datacenter
  - Latency sensitive
  - $O(\text{milliseconds})$
- Handled by worker/aggregator tasks, databases & caches

# Workload: Map Reduce



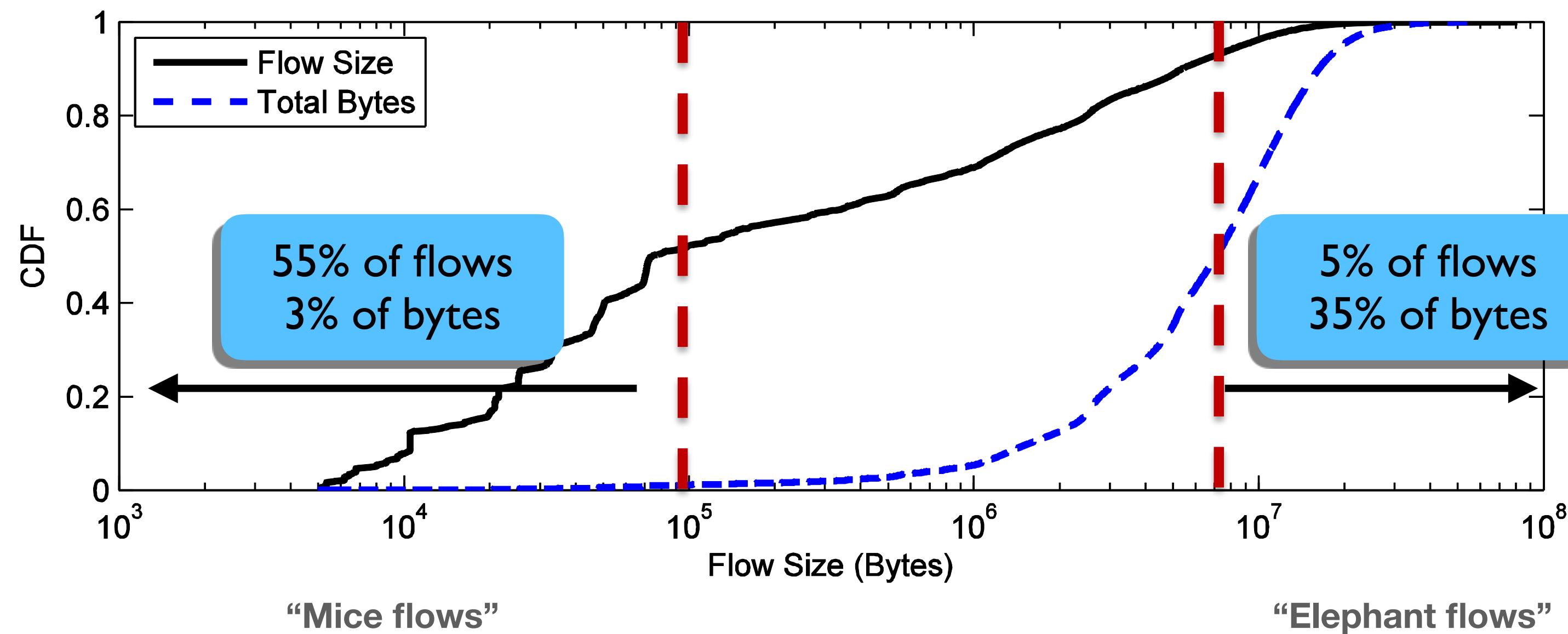
# “East-west” Traffic



- Traffic between servers in the datacenter
  - Bandwidth intensive
  - $O(\text{mins})$
- Handled by map/reduce tasks, distributed filesystem

# Characterizing Traffic Pattern: “Elephant” & “Mice” Flows

- Microsoft [Alizadeh et. al. 2010]
  - Web-search (north-south), data mining (east-west)



**Research:** How do you design the network protocols for such traffic?

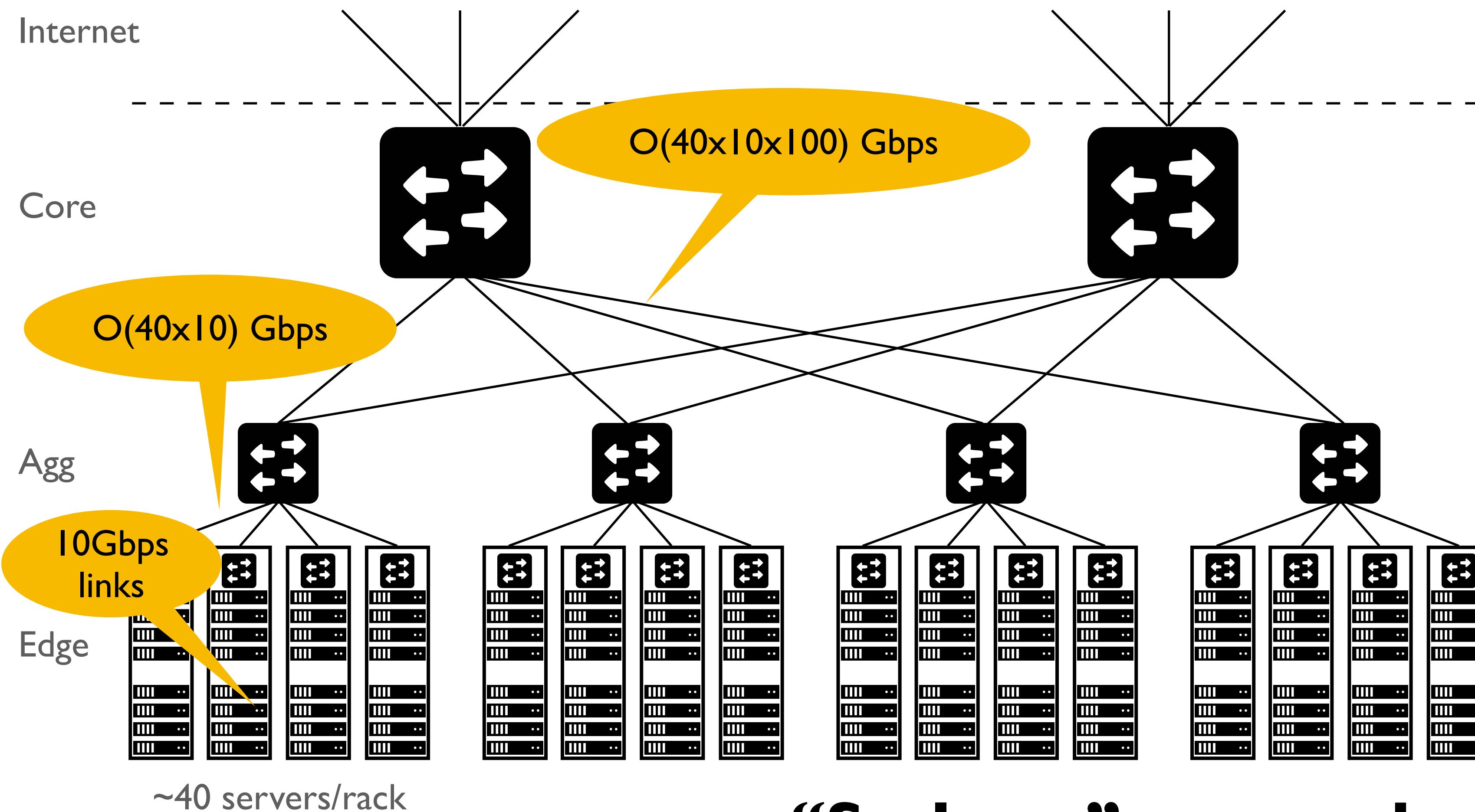
# Implications of Application Characteristics

- Low latency is critical (for north-south)
  - Also worst-case (“tail”) latency
- High bandwidth any-to-any communication (east-west)
  - ‘Bisection bandwidth’

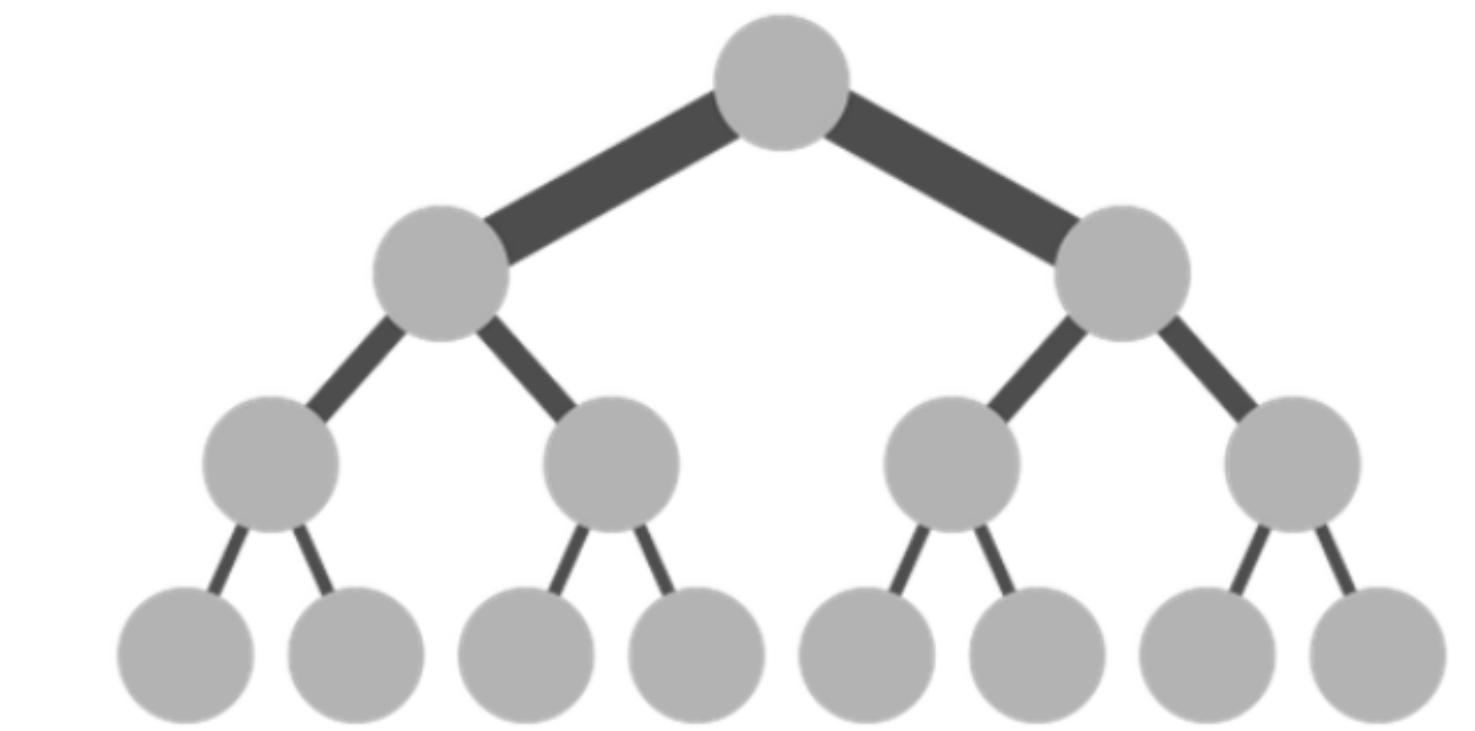
# Bisection Bandwidth

- Partition network into two equal parts
- Minimum bandwidth between the partitions is the *bisection bandwidth*
- *Full bisection bandwidth:* bisection bandwidth in an  $N$ -node network is  $N/2$  times the bandwidth of a single access link
  - Nodes of *any* two halves can communicate at full speed with each other

# Achieving Full Bisection Bandwidth



**“Scale up” approach**



# Achieving Full Bisection Bandwidth

- Challenge: “Scaling up” a traditional tree topology is expensive!
  - Requires non-commodity / impractical / link and switch components
- Solutions?
  - Later...

# Questions?

# What you should understand...

- What is a datacenter network
  - Scale, service-model, application characteristics
- What makes it different?
  - Characteristics, goals (w.r.t. Internet), degrees of freedom
- How do we achieve goals by exploiting freedom?
  - Topology redesign, L2/L3 redesign, L4 redesign
  - We will look at some approaches, not all

# What's different about DC Networks

## Characteristics

- Huge scale
  - ~20,000 switches/routers
  - Contrast: AT&T ~500 routers

# What's different about DC Networks

## Characteristics

- Huge scale
- Limited geographic scope
  - High bandwidth: 10/40/100G (contrast: DSL/WiFi)
  - Very low RTT: 1-10s usecs (contrast: 100s msec)

# What's different about DC Networks

## Characteristics

- Huge scale
- Limited geographic scope
- Limited heterogeneity
  - Link speeds, technologies, latencies, ...

# What's different about DC Networks

## Characteristics

- Huge scale
- Limited geographic scope
- Limited heterogeneity
- Regular/planned topologies (e.g., trees)
  - Contrast: ad-hoc evolution of wide-area topologies

# What's different about DC Networks

## Goals

- **Extreme bisection bandwidth requirements**
  - Recall: all that east-west traffic
  - Target: any server can communicate at its full link speed

# What's different about DC Networks

## Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
  - Real money on the line
  - Current target: 1 us RTTs

# What's different about DC Networks

## Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
  - “Your packet will reach in X ms, or not at all”
  - “Your VM will always see at least Y Gbps throughput”
  - How is still an open question

# What's different about DC Networks

## Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
- Differentiating between tenants is key
  - e.g., “No traffic between VMs of tenant A and tenant B”
  - “Tenant X cannot consume more than X Gbps”
  - “Tenant Y’s traffic is low priority”

# What's different about DC Networks

## Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
- Differentiating between tenants is key
- Scalability (of course)

# What's different about DC Networks

## Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
- Differentiating between tenants is key
- Scalability (of course)
- Cost/efficiency
  - Focus on commodity solutions, ease of management

# What's different about DC Networks

New degrees of (design) freedom

- **Single administrative domain!**
  - Can deviate from standards, invent your own, etc.
  - “Greenfield” deployment is still feasible

# What's different about DC Networks

New degrees of (design) freedom

- Single administrative domain!
- Control over network *and* endpoint(s)
  - Can change (say) addressing, congestion control, etc.
  - Can add mechanisms for security/policy/etc. at endpoints (typically in the hypervisor)

# What's different about DC Networks

New degrees of (design) freedom

- Single administrative domain!
- Control over network *and* endpoint(s)
- Control over the *placement* of traffic source/sink
  - e.g., map-reduce scheduler chooses where tasks run
  - Can control what traffic crosses which links

# Questions?

# What you should understand...

- What is a datacenter network
  - Scale, service-model, application characteristics
- What makes it different?
  - Characteristics, goals (w.r.t. Internet), degrees of freedom
- **How do we achieve goals by exploiting freedom?**
  - Topology redesign, L2/L3 redesign, L4 redesign
  - We will look at some approaches, not all

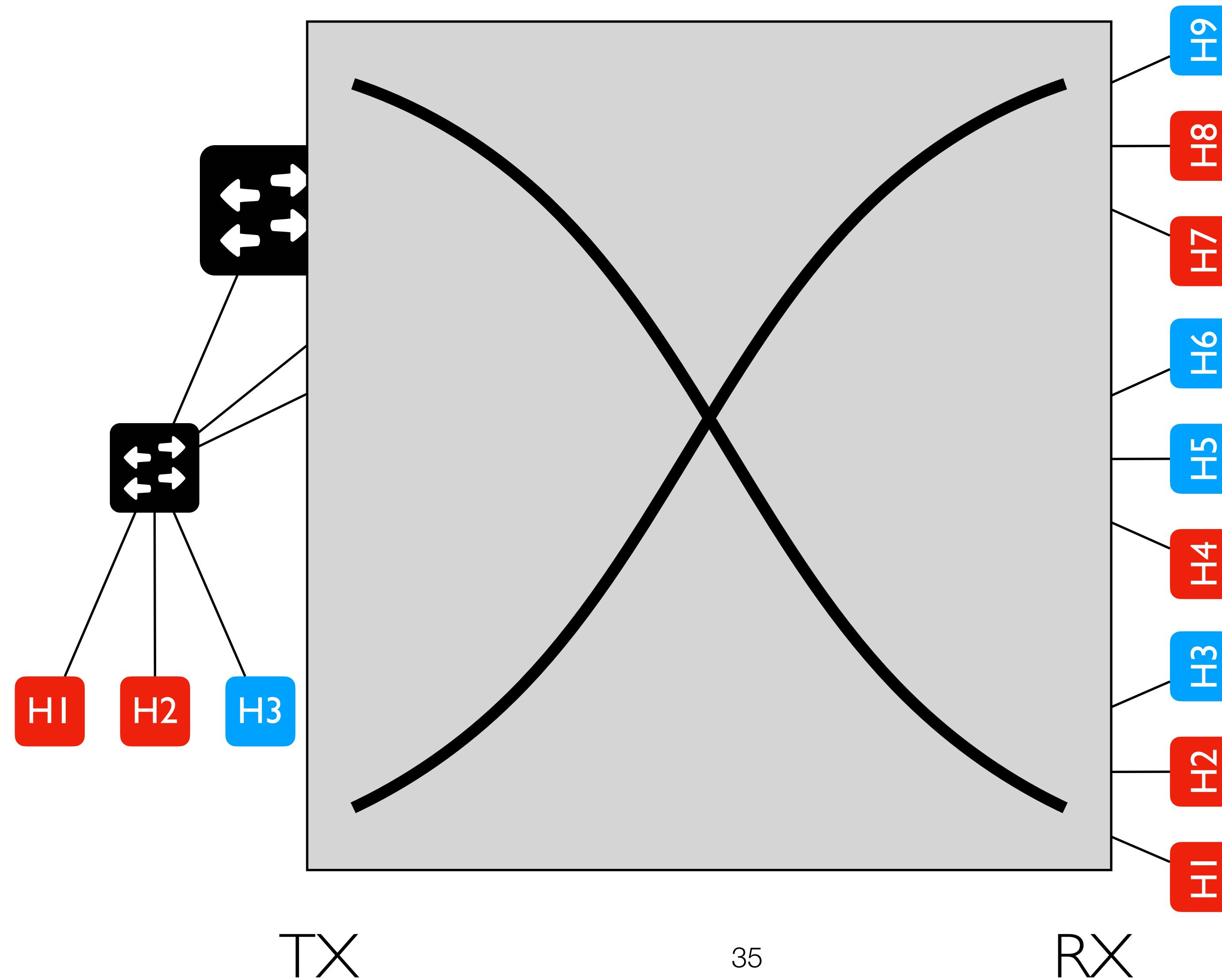
# How do we achieve DCN goals?

- **Network architecture design**
  - Rearchitect network topology to achieve full bisection bandwidth
- **L2/L3 design:**
  - Addressing / routing / forwarding in new topology
- **L4 design:**
  - Transport protocol design for new topology

# How do we achieve DCN goals?

- Network architecture design
  - Rearchitect network topology to achieve full bisection bandwidth
- Remember: we have the freedom to do this!
  - Single administration domain
- Conceptually: DC network as one giant switch!

# DC Network: Just a Giant Switch!

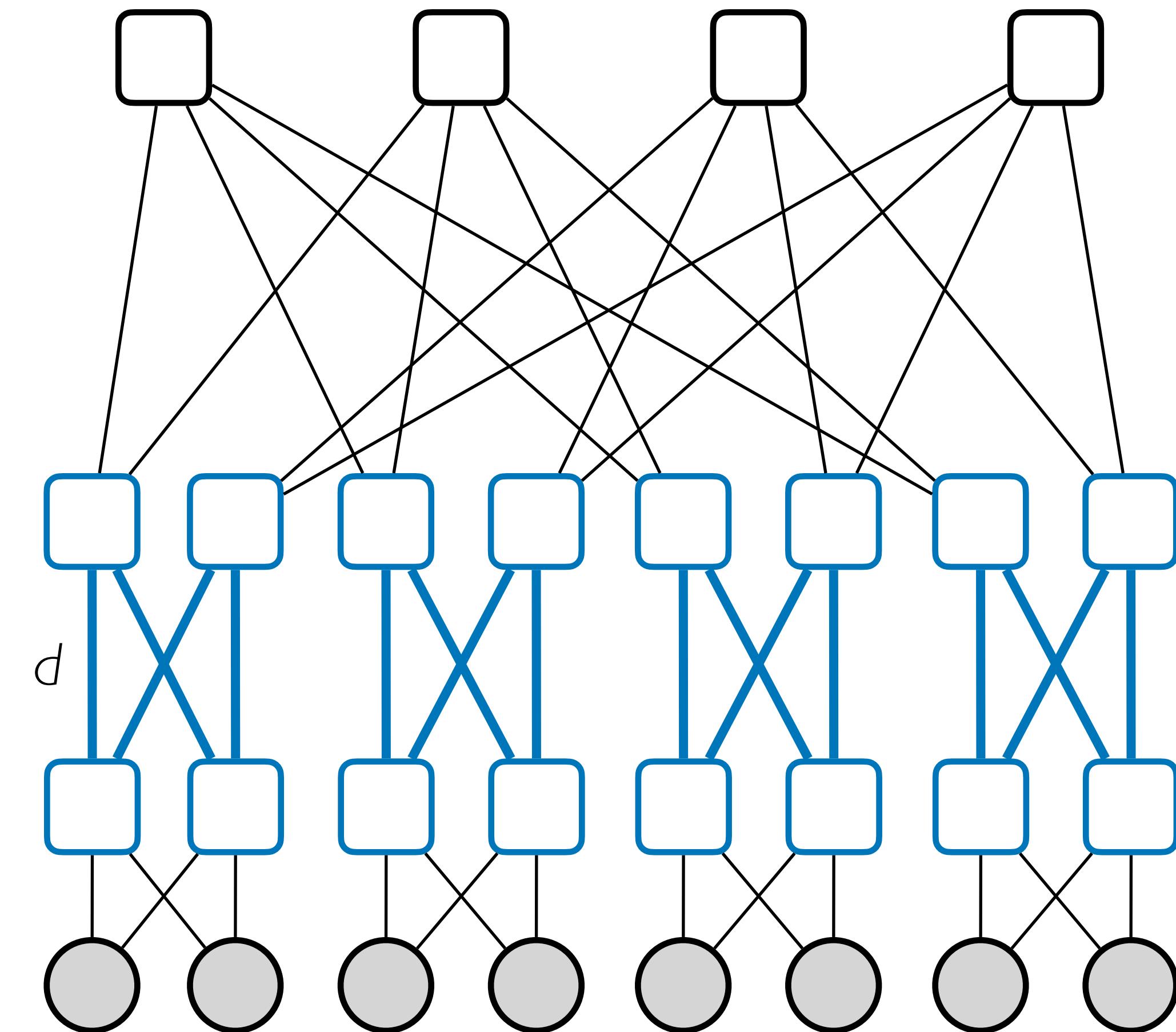


# High Bandwidth

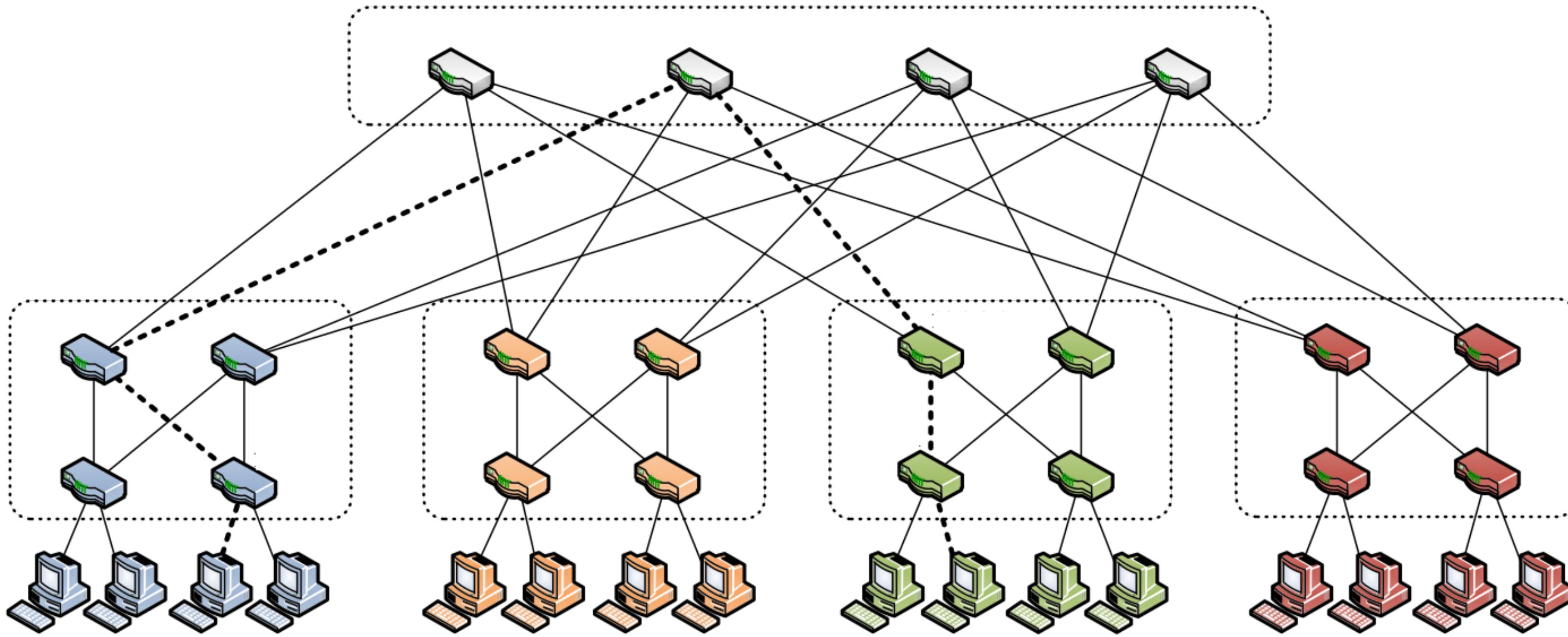
- Ideal: Each server can talk to any other server at its full access link rate
- Conceptually: DC network as one giant switch
  - Would require a 10 Pbits/sec switch!
    - 1M ports (one port per server)
    - 10 Gbps per port
- Practical approach: build a network of switches (“fabric”) with high bisection bandwidth
  - Each switch has practical #ports and link speeds

# Architecting Better Topologies

- E.g., ‘Clos’ topology
  - Multi-stage network with  $L$  stages
  - All switches have  $k$  ports
  - $k/2$  ports up,  $k/2$  down (except core)
  - All links have same speed
  - Guarantees full bisection bandwidth



# “Fat Tree” Topology [SIGCOMM’08]



- **Few things to notice:**
  - Special case of Clos Topology with 3 stages (edge/aggregation/core)
  - All switches have the same number of ports (=4)
  - All links have same bandwidth
  - Many paths between any two end hosts

# Bandwidth: Provisioning vs. Using

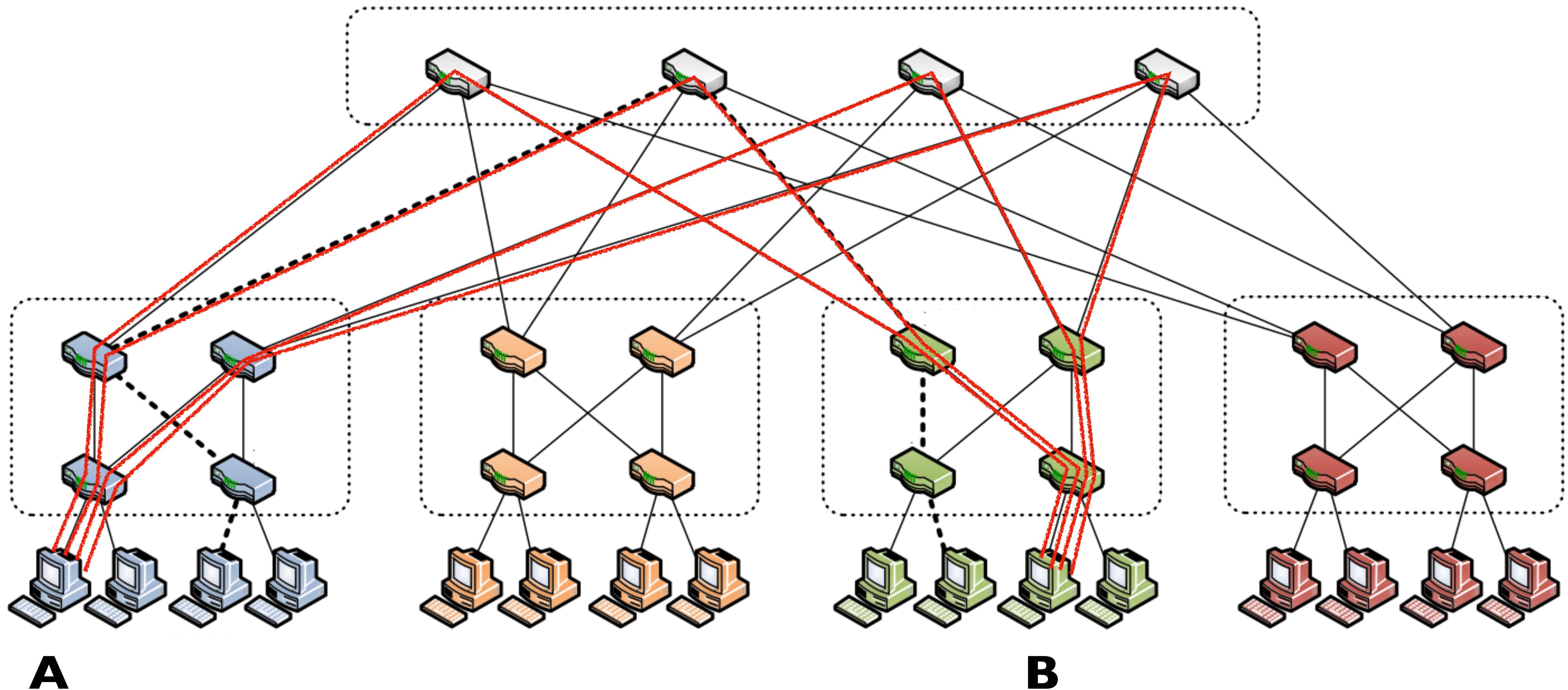
- Topology offers high bisection bandwidth
- All other system components must be able to exploit this available capacity
  - Routing must use all paths
  - Transport protocol must fill all pipes (fast)

# Questions?

# How do we achieve DCN goals?

- **L2/L3 design:**
  - Addressing / routing / forwarding in the Fat-Tree
- **Goals:**
  - Routing protocol must expose all available paths
  - Forwarding must spread traffic evenly all paths

# Using multiple paths well



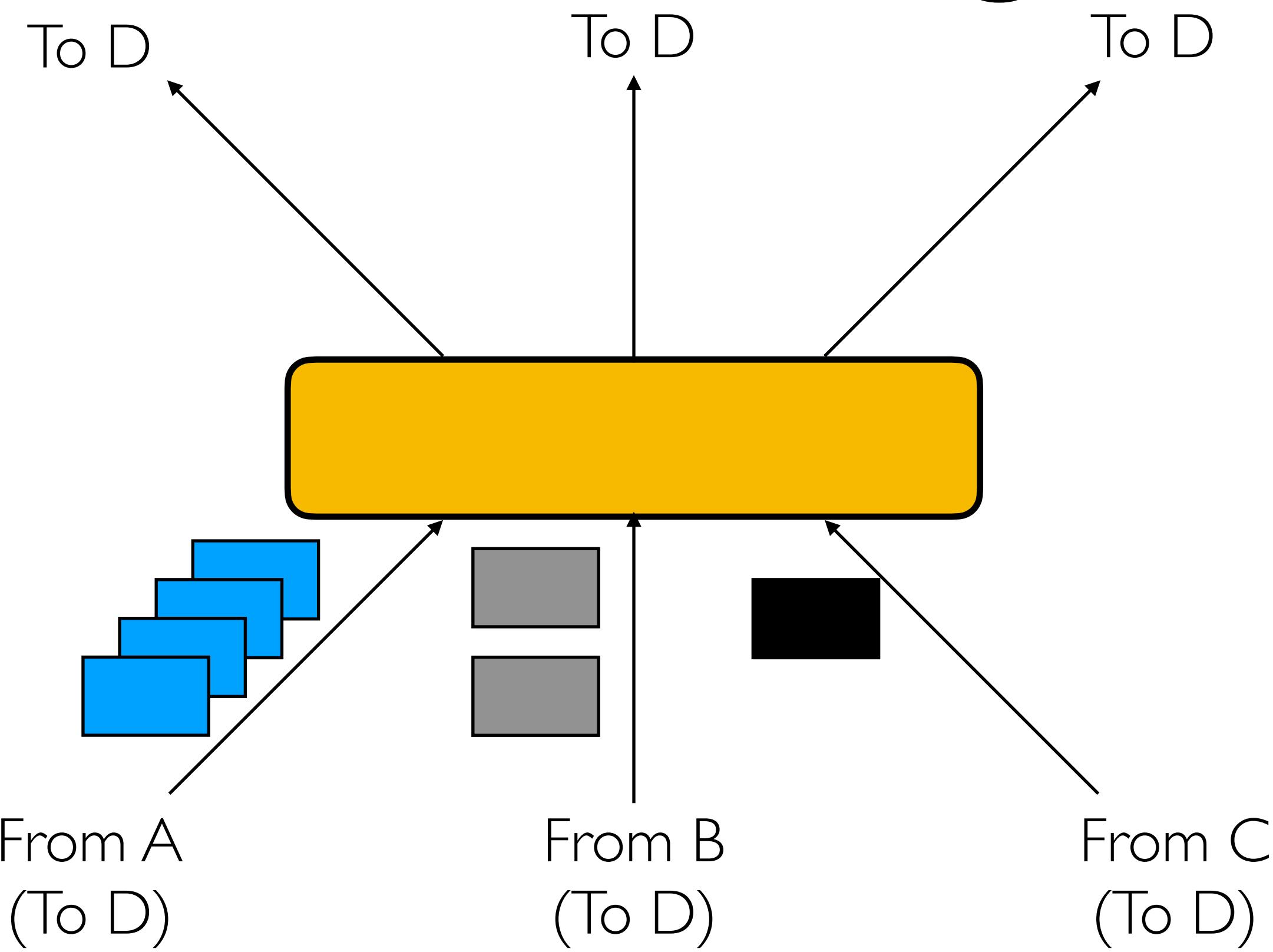
A

B

# Extend DV/LS to exploit multi-paths

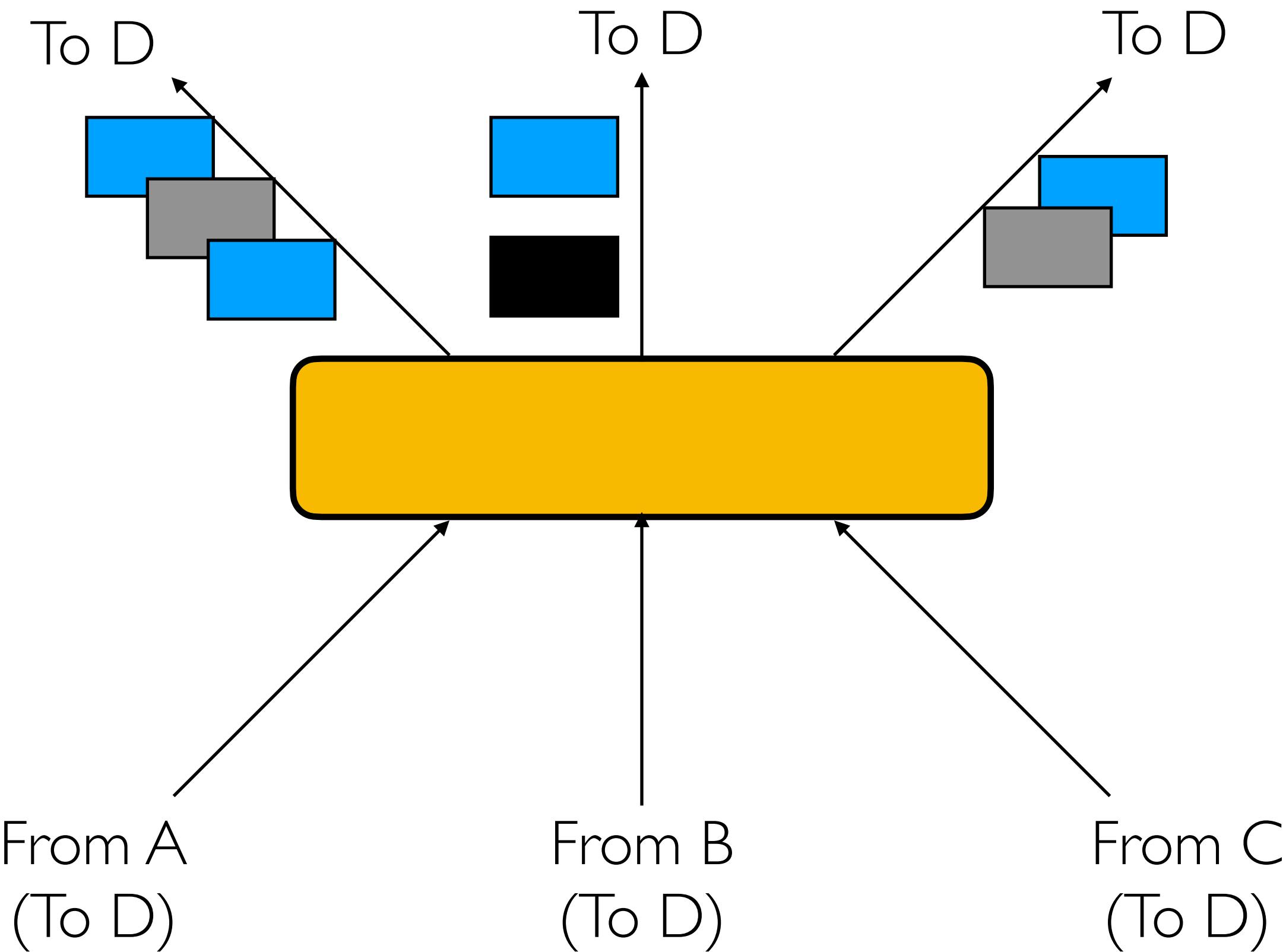
- Routing: how to use multiple paths?
  - Distance-vector: Remember *all* next hops that advertise equal cost to a destination
  - Link-state: Extend Dijkstra's to compute *all* equal cost shortest paths to each destination
- Forwarding: how to spread traffic across next hops?

# Forwarding



- Per-packet load balancing

# Forwarding

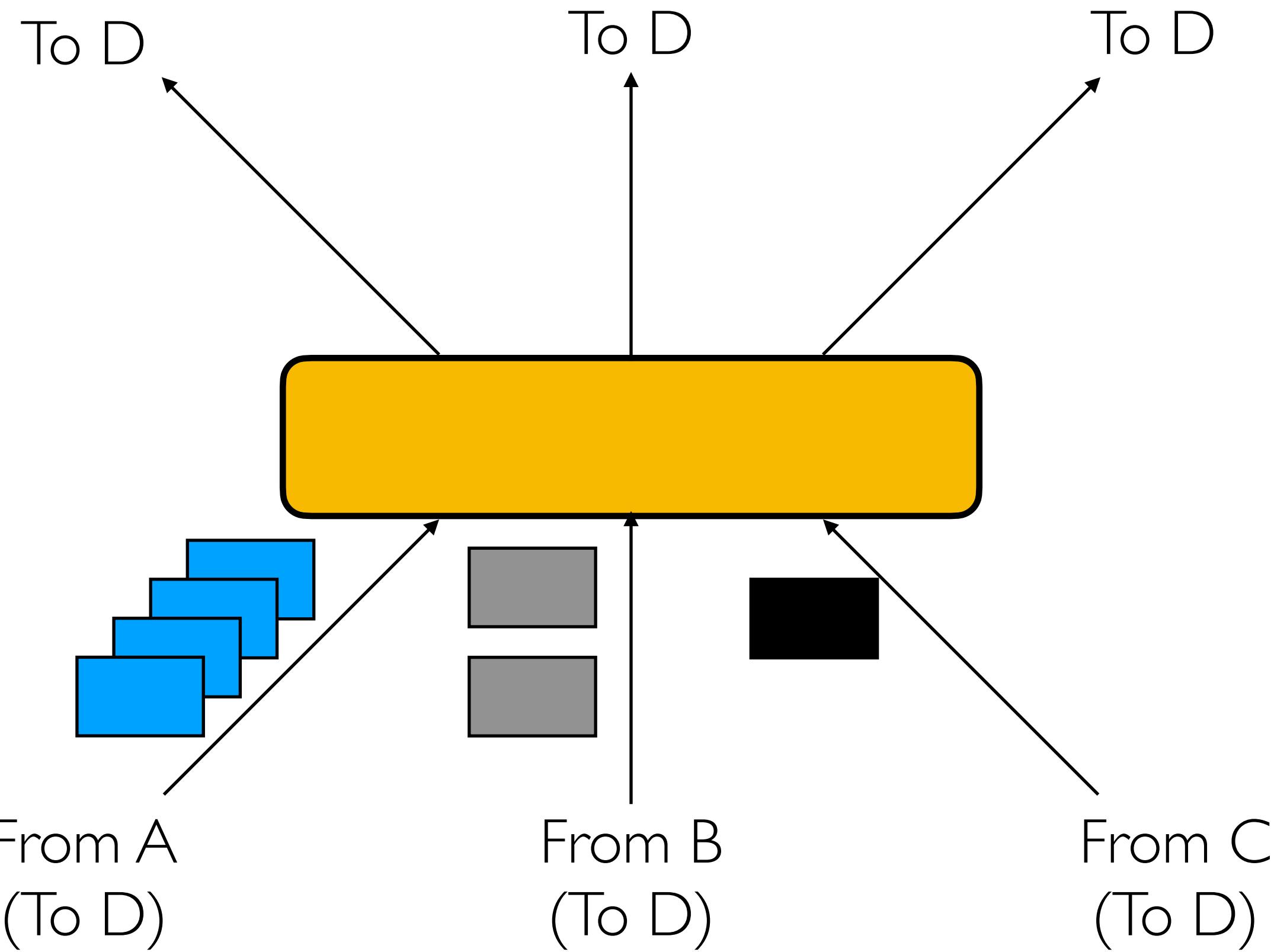


- Per-packet load balancing
  - Traffic well spread (even w/ elephant flows)
  - Interacts badly w/ TCP

# TCP w/ per-packet load balancing

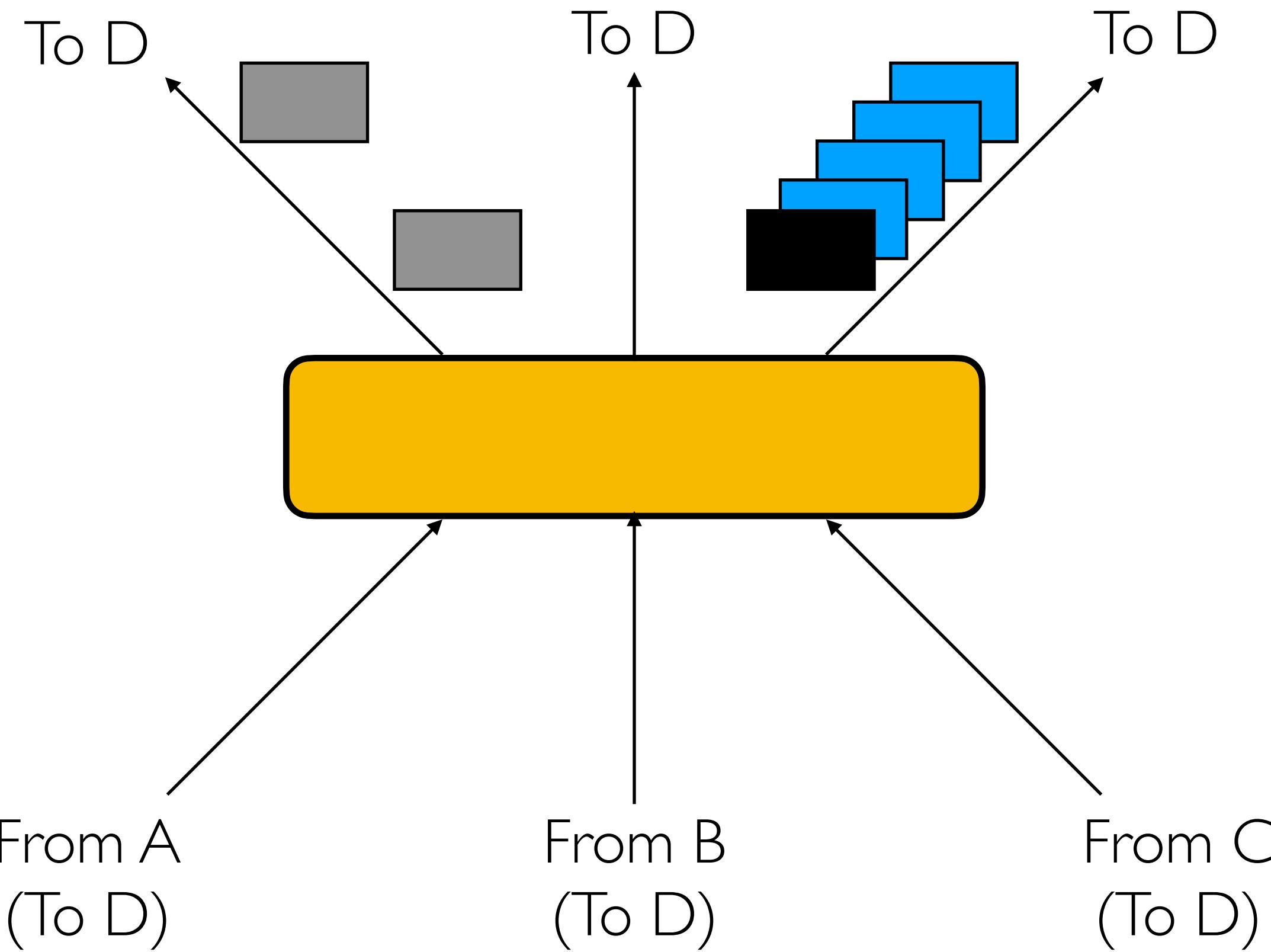
- Consider:
  - Sender sends seq#: 1, 2, 3, 4, 5
  - Receiver receives: 5, 4, 3, 2, 1
  - Sender will enter fast rtx, reduce CWND, rtx #1, ...
  - Repeatedly!
- Also: one RTT and timeout estimator for multiple paths
- Also: CWND halved when a packet is dropped on *any* path
- Multipath TCP (MP-TCP): Extend TCP with multi path routing
  - Value beyond datacenters (e.g., spread traffic across WiFi and 4G access)

# Forwarding



- **Per-flow load balancing (ECMP, “Equal Cost Multi Path”)**
  - E.g., based on  $\text{HASH}(\text{srcAddr}, \text{dstAddr}, \text{srcPort}, \text{dstPort})$

# Forwarding



- **Per-flow load balancing (ECMP, “Equal Cost Multi Path”)**
  - E.g., based on  $\text{HASH}(\text{srcAddr}, \text{dstAddr}, \text{srcPort}, \text{dstPort})$
  - Pro: a flow follows a single path (means TCP is happy!)
  - Con: non-optimal load balancing; elephants are a problem

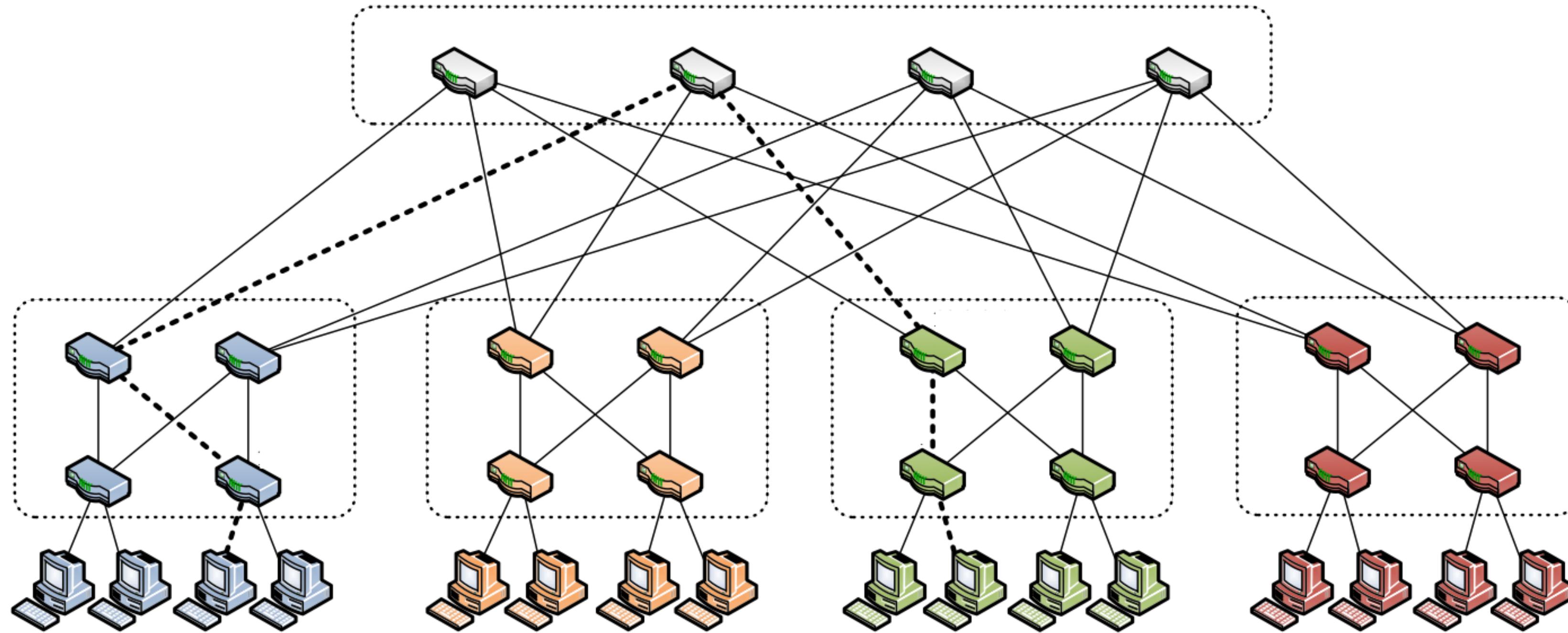
# Extend DV/LS to exploit multi-paths

- How:
  - Simple extensions to DV/LS
  - ECMP for load balancing
- Benefits
  - Simple; reuses existing solutions
- Problem: scales poorly
  - With  $N$  destinations,  $O(N)$  routing entries and messages
  - $N$  now in the millions!

# Questions?

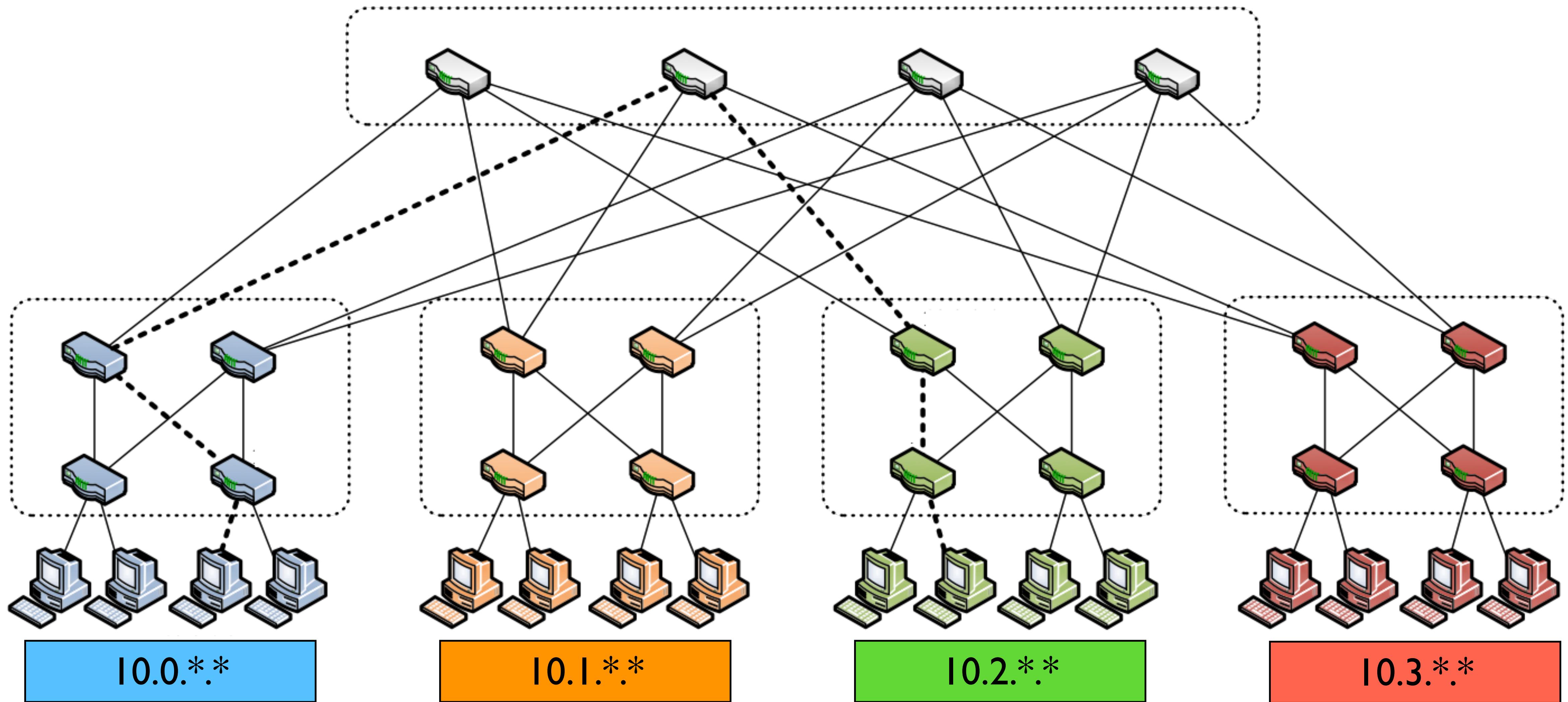
# You: design a scalable routing solution

- Design for this specific topology

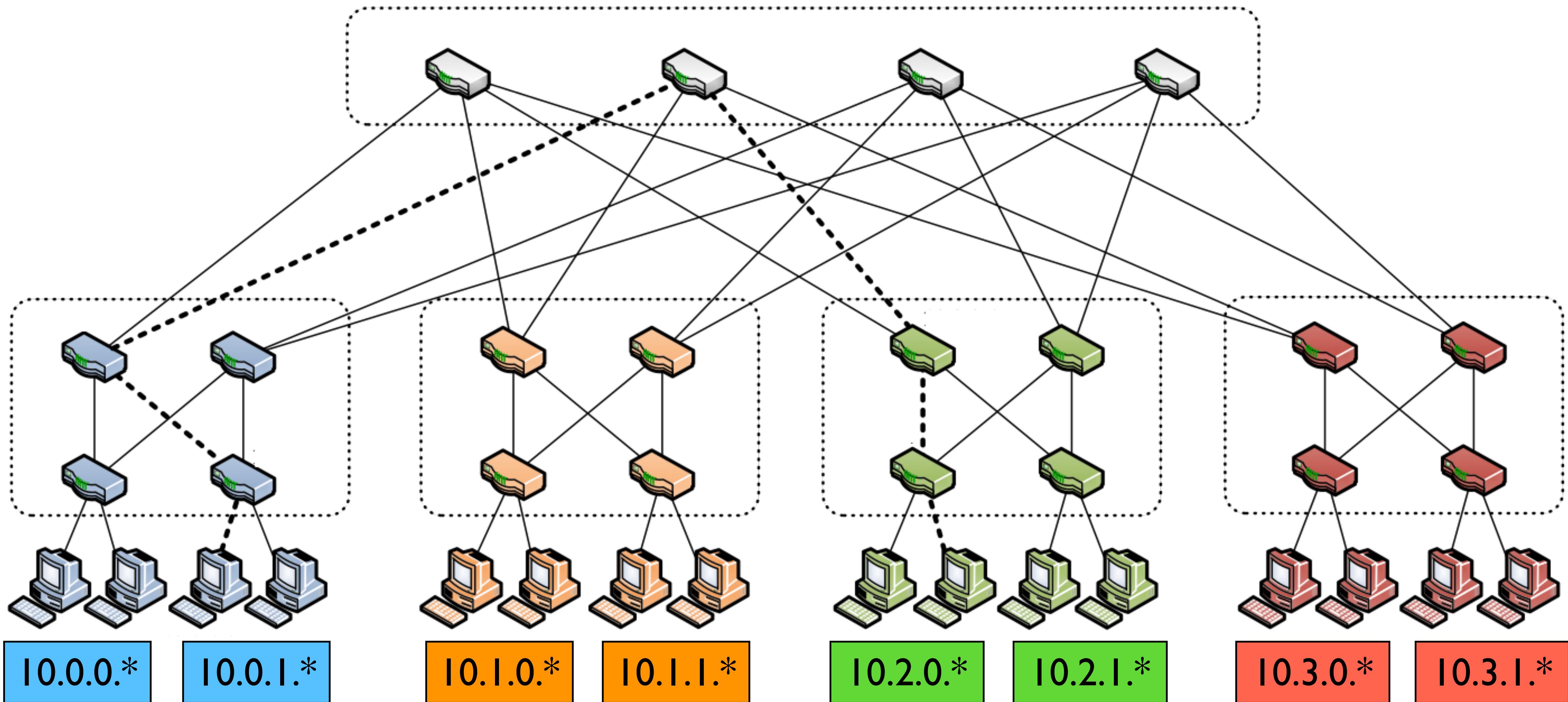


- Take 3 minutes, then tell me:
  - #routing entries per switch your solution needs
  - Many solutions possible; remember: you are now free from backward compatibility
  - Can redesign IP, routing algorithms, switch designs, etc.

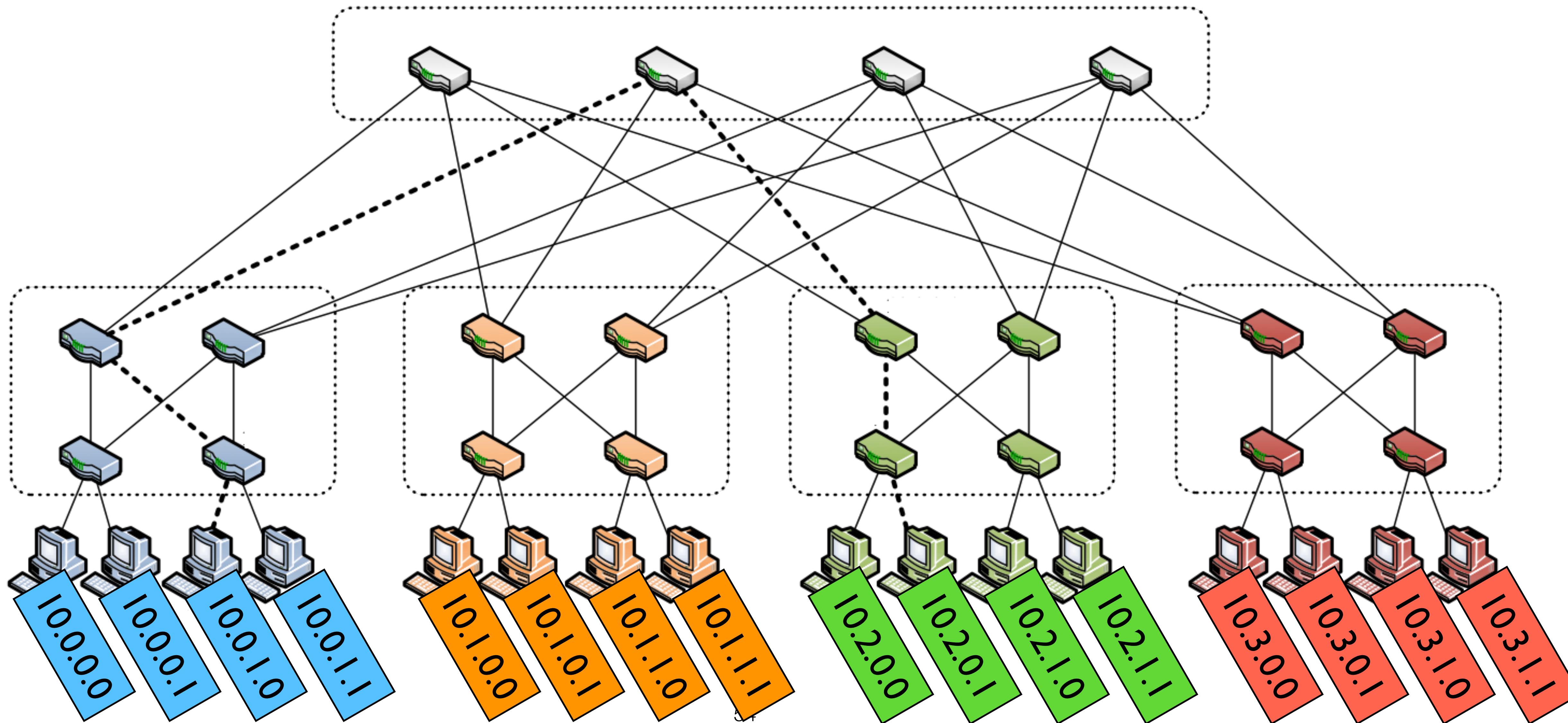
# Solution I:Topology-aware addressing



# Solution I: Topology-aware addressing

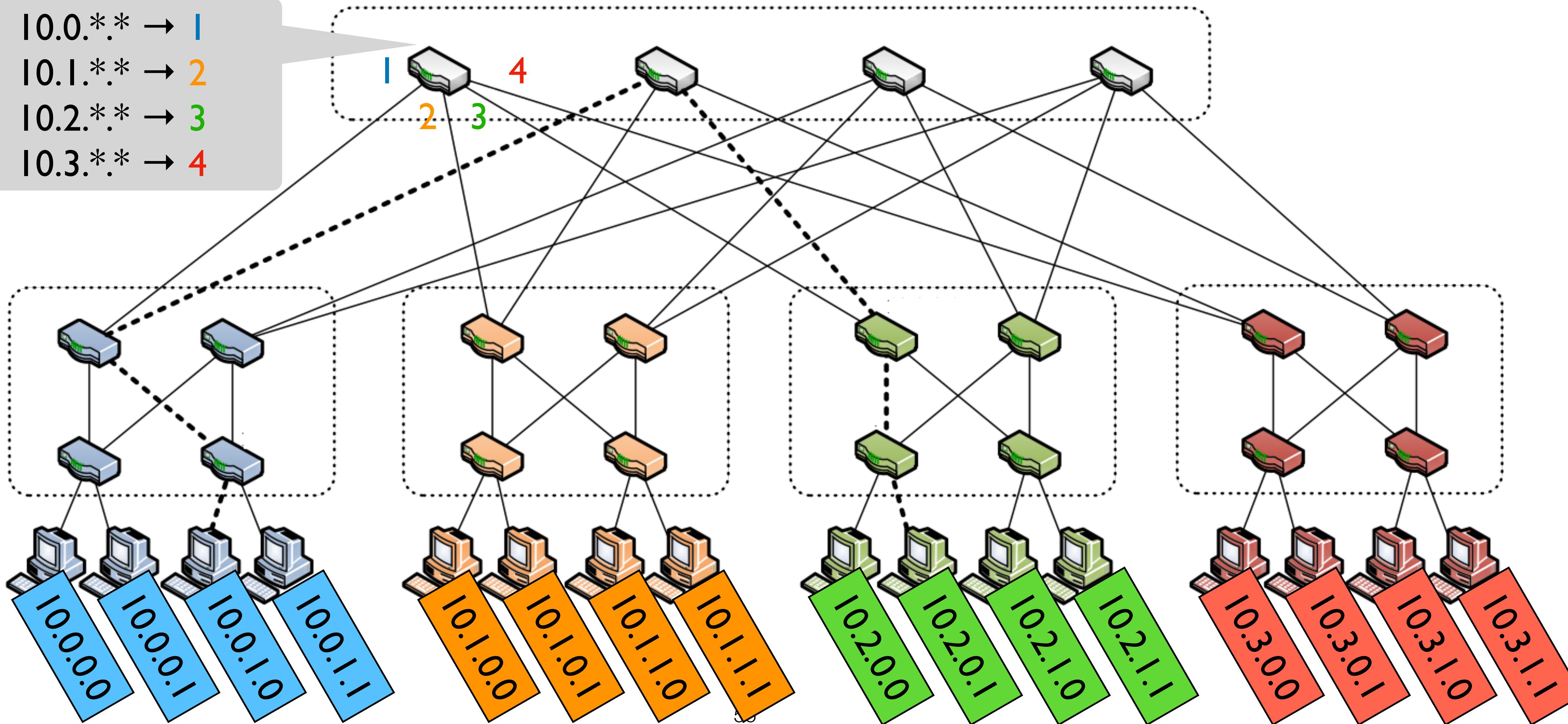


# Solution I: Topology-aware addressing



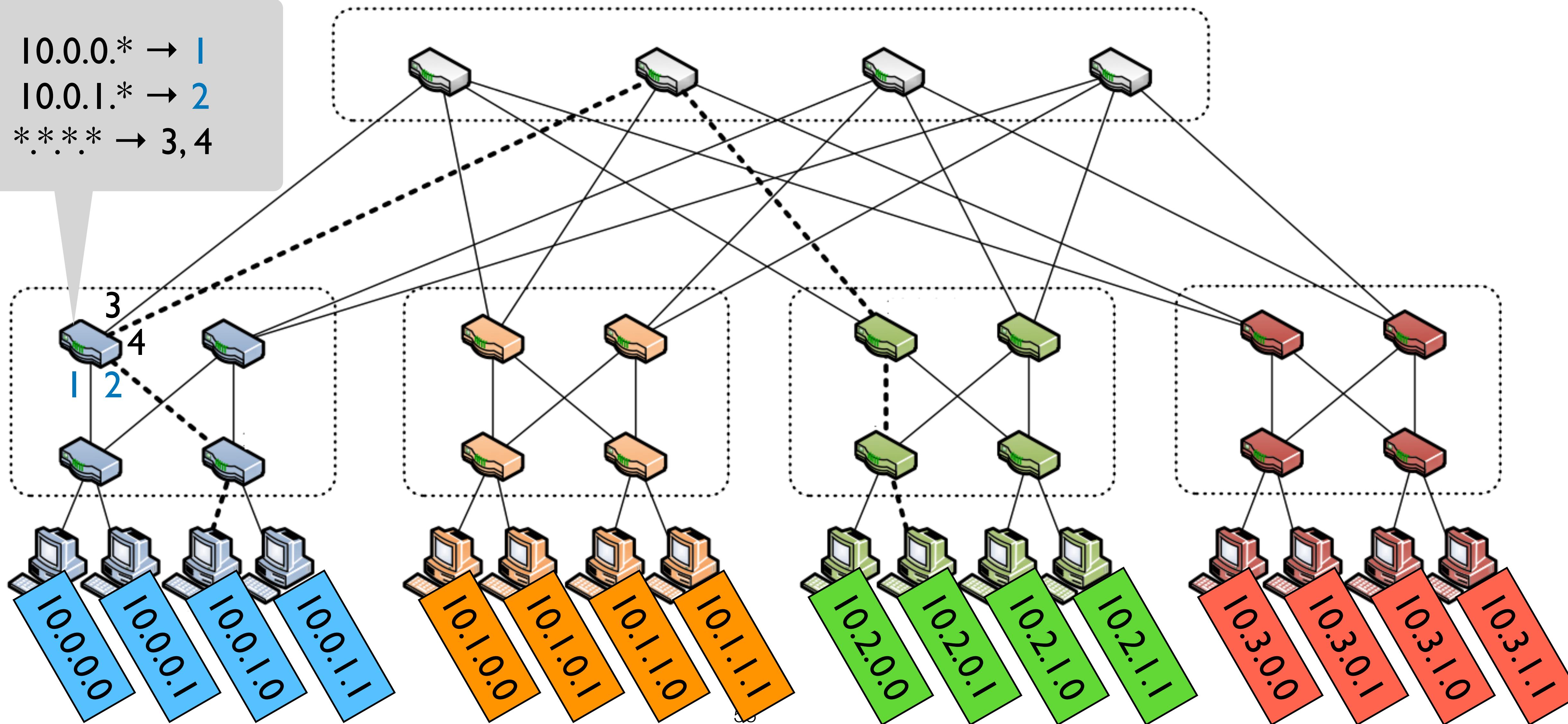
# Solution I: Topology-aware addressing

$10.0.*.* \rightarrow 1$   
 $10.1.*.* \rightarrow 2$   
 $10.2.*.* \rightarrow 3$   
 $10.3.*.* \rightarrow 4$



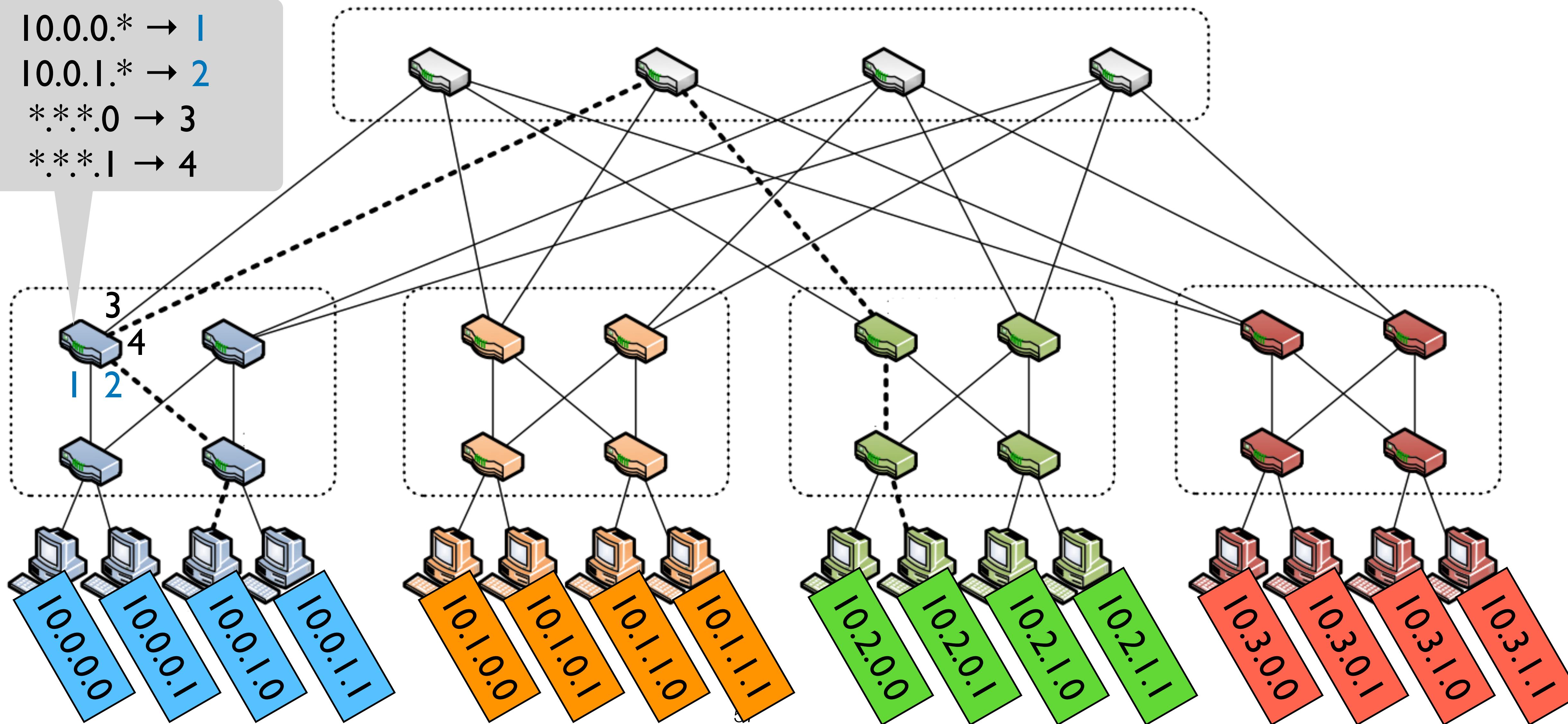
# Solution I: Topology-aware addressing

$10.0.0.* \rightarrow 1$   
 $10.0.1.* \rightarrow 2$   
 $*.*.* \rightarrow 3, 4$

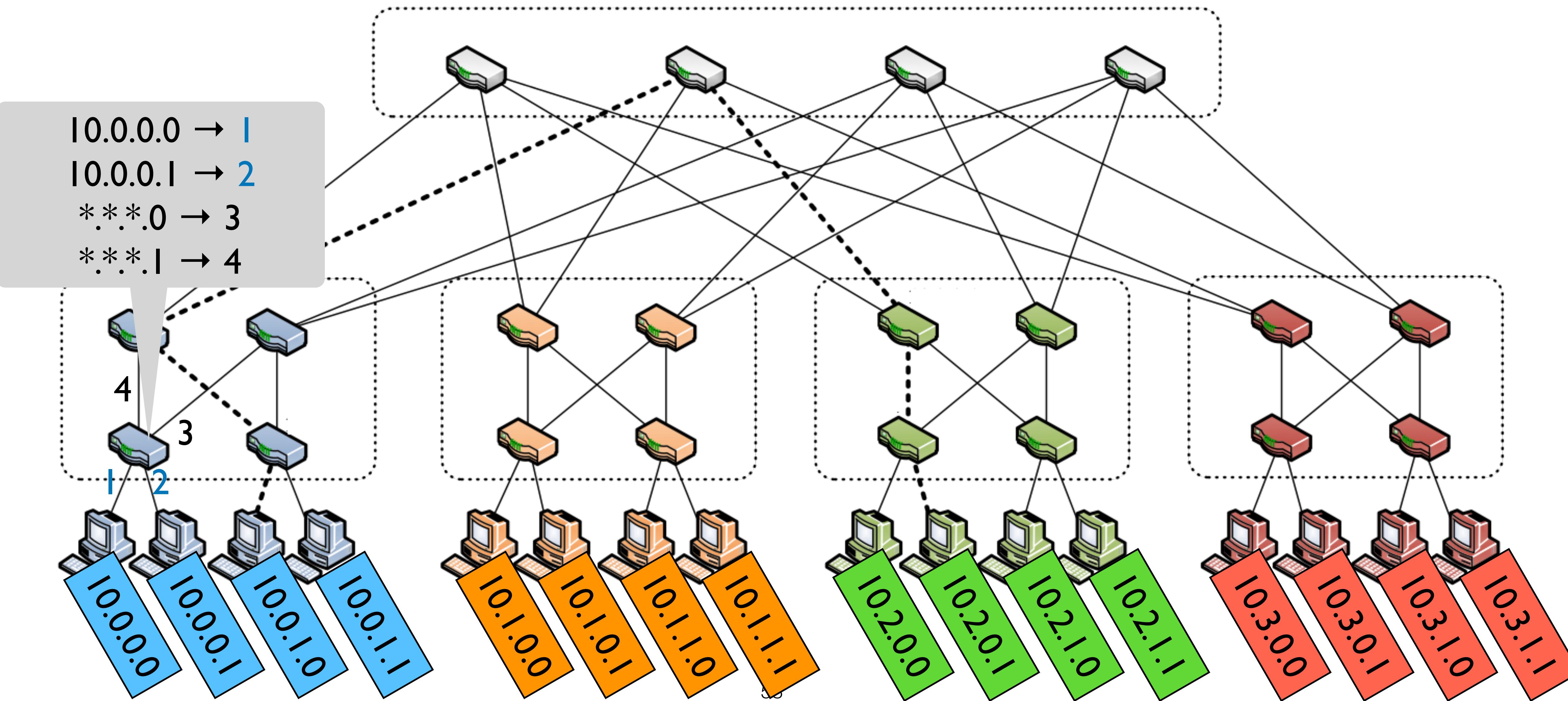


# Solution I: Topology-aware addressing

$10.0.0.* \rightarrow 1$   
 $10.0.1.* \rightarrow 2$   
 $*.*.*.0 \rightarrow 3$   
 $*.*.*.1 \rightarrow 4$



# Solution I: Topology-aware addressing



# Solution I: Topology-aware addressing

- Idea: addresses embed location in regular topology
- Maximum #entries/switch:  $k$  ( $=4$  in example)
  - Constant, independent of #destinations!
- No route computation / messages / protocols
  - Topology is “hard coded”
- Problems?
  - VM migration: ideally, VM keeps its IP address when it moves
  - Vulnerable to misconfiguration (in topology or addresses)

# Solution 2: Centralize + Source Routes

- Centralized “controller” server knows topology & computes routes
- Controller hands server all paths to each destination
  - $O(\# \text{destinations})$  state per server
  - But server memory cheap (e.g., 1M routes  $\times$  100B route = 100MB)
- Server inserts entire path vector into packet header (“source routing”)
  - E.g., header=[dst=D | index=0 | path={S5, S1, S2, S9}]
- Switch forwards based on packet header
  - $\text{index}++; \text{next-hop} = \text{path}[\text{index}]$

# Solution 2: Centralize + Source Routes

- #entries per switch?
  - None!
- #routing messages?
  - Akin to a broadcast from controller to all servers
- Pro:
  - Switches very simple and scalable
  - Flexible: end-points (hence applications) control route selection
- Cons:
  - Scalability / robustness of controller (SDN addresses this)
  - Clean-state design of everything

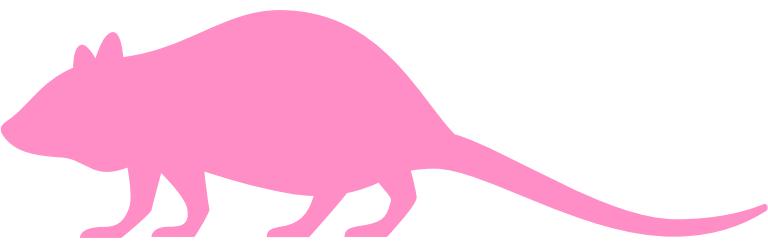
# Questions?

# How do we achieve DCN goals?

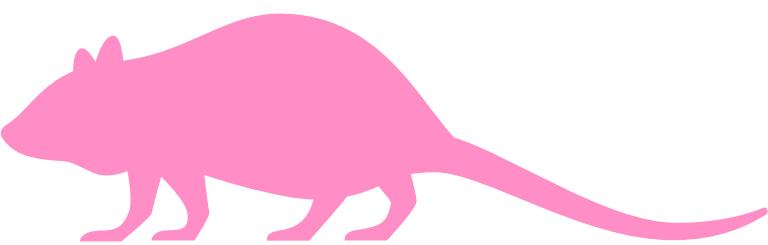
- L4 design:
  - Transport protocol design (w/ Fat-Tree)

# Workloads

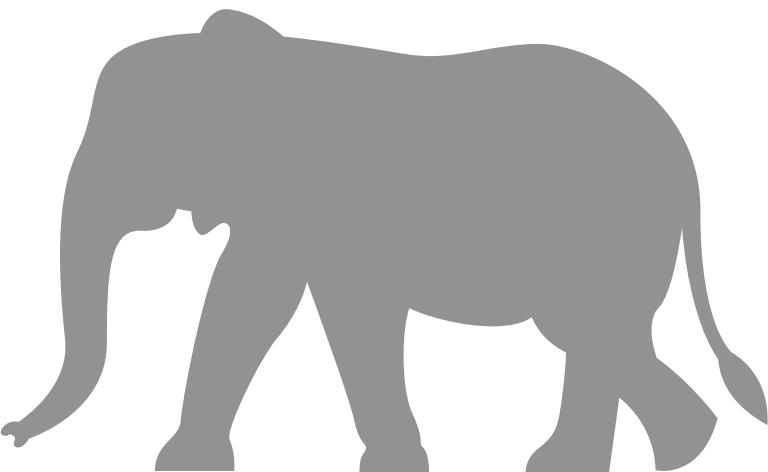
- Partition/aggregate
  - Query
- Short messages [50KB-1MB]
  - Coordination, control state
- Large flows [1MB-50MB]
  - Data update



Delay-sensitive



Delay-sensitive



Throughput-sensitive

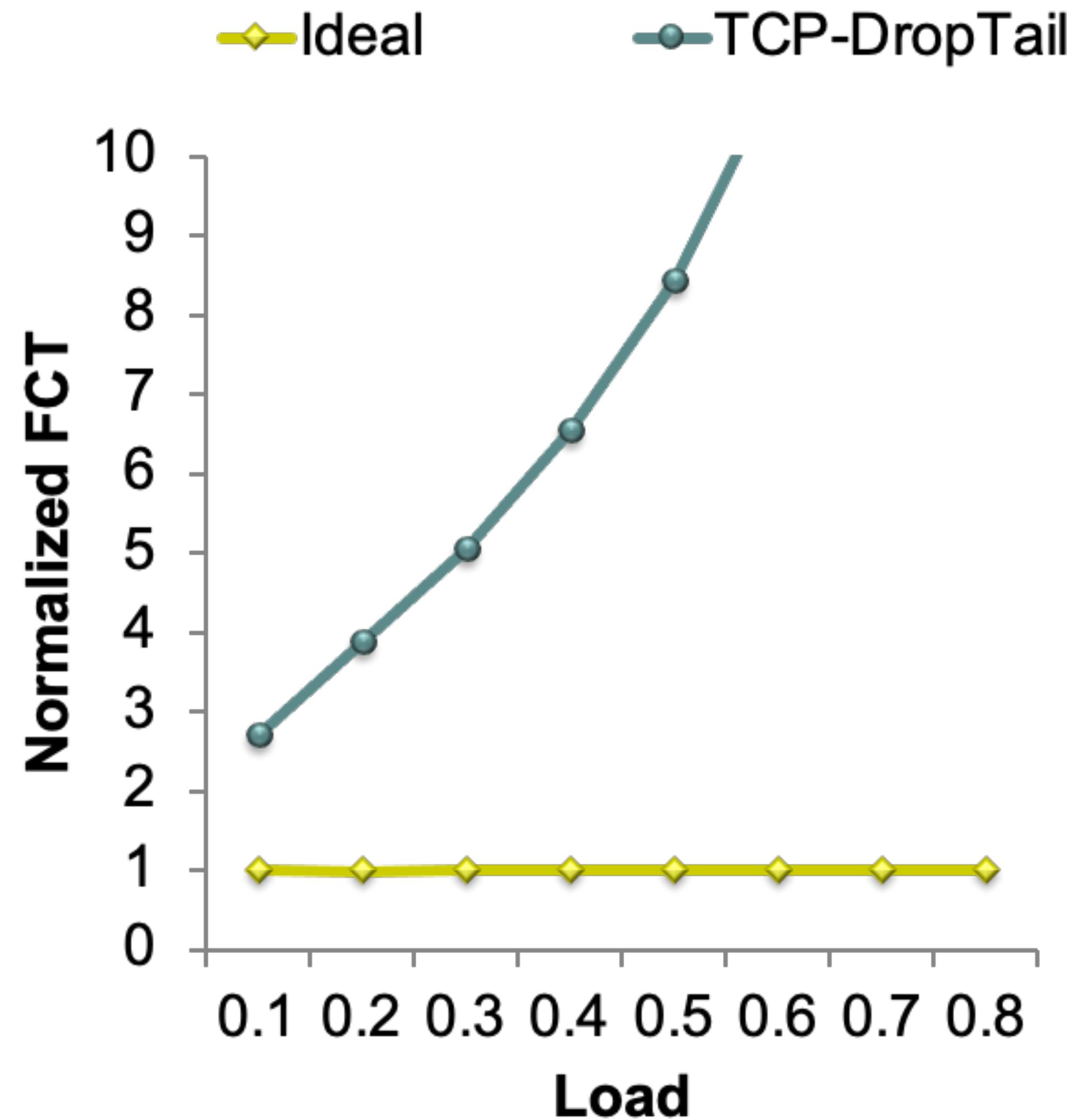
# What's “Ideal”?

- What is the best measure of performance for a data center transport protocol?
  - When the flow is completely transferred?
  - Latency of each packet in the flow?
  - Number of packet drops?
  - Link utilization?
  - Average queue length at switches?

# Flow Completion Time (FCT)

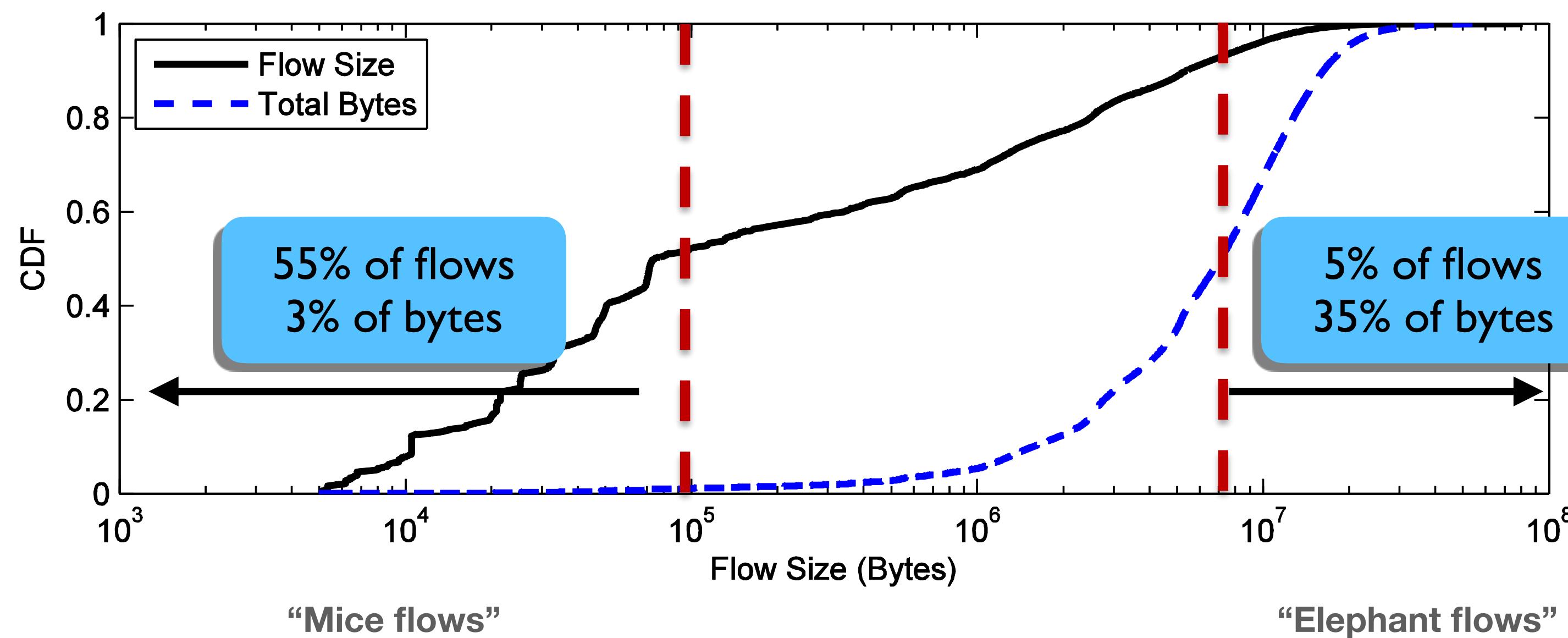
- Time from when flow started at the sender, to when all packets in the flow were at the receiver

# FCT with TCP

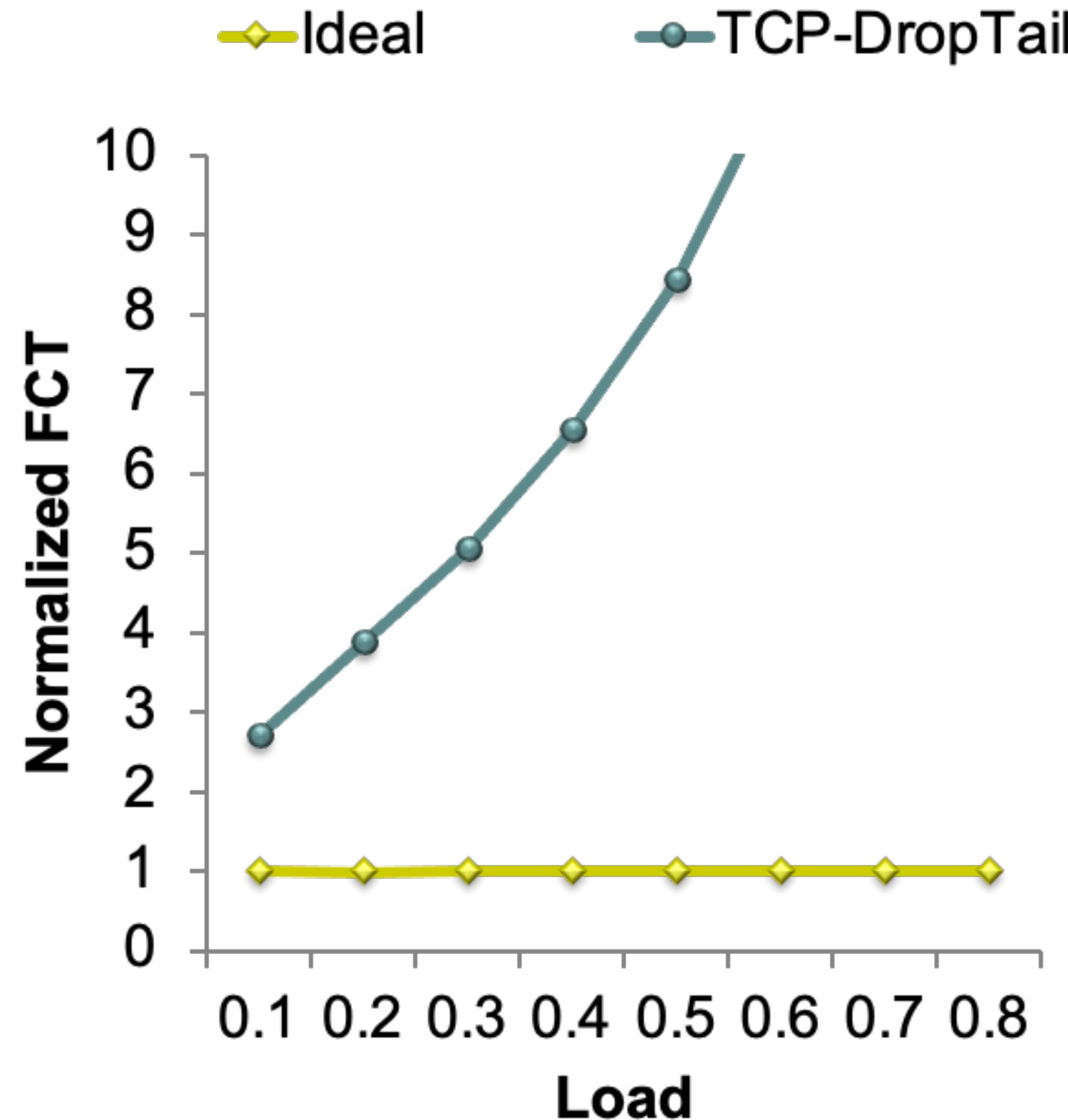


# Recall: “Elephants” and “Mice”

- Microsoft [Alizadeh et. al. 2010]
  - Web-search (north-south), data mining (east-west)



# FCT with TCP

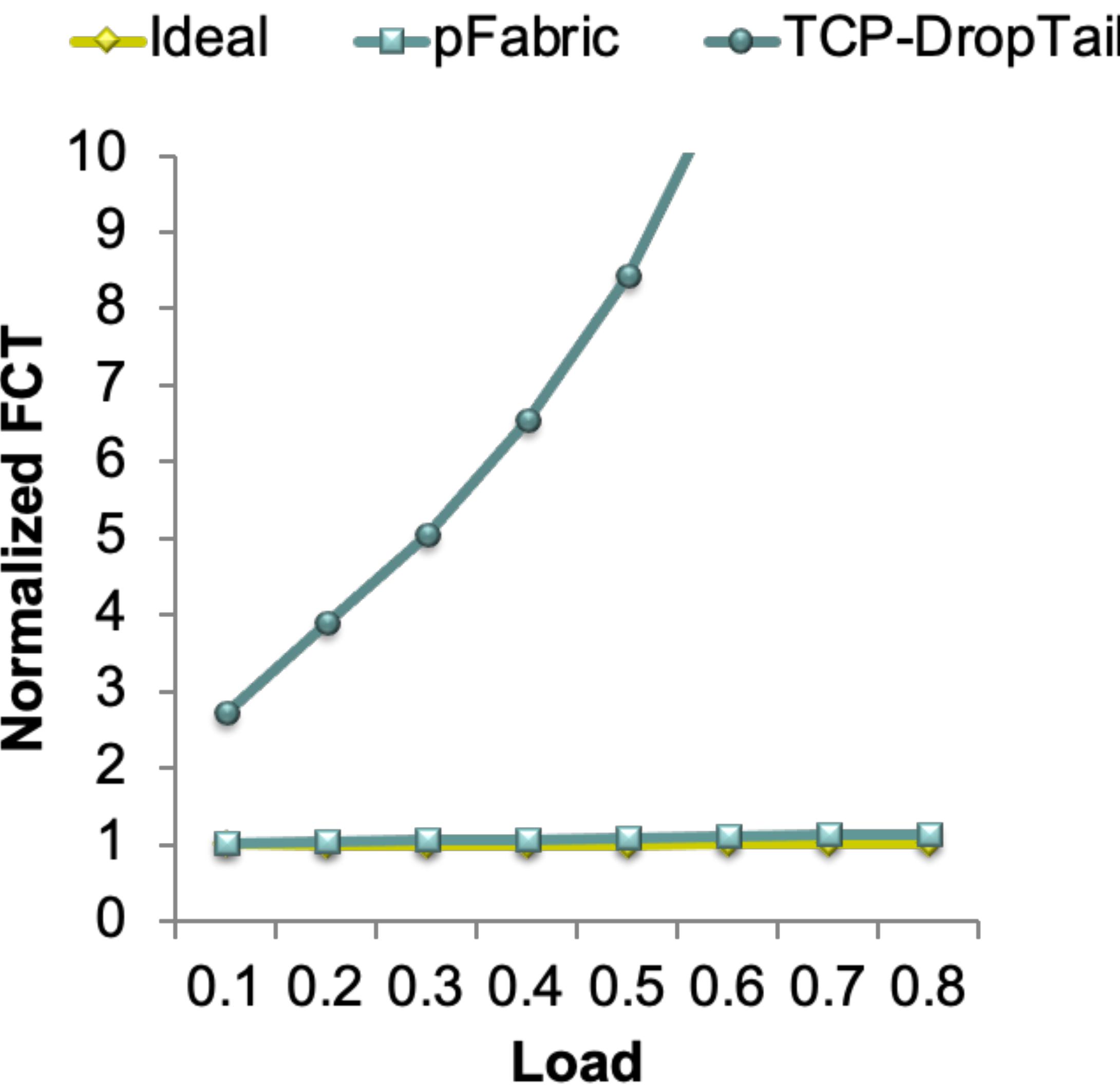


Problem: the mice are delayed by the elephants

# Solution: Use Priorities! [*pFabric, SIGCOMM 2013*]

- Packets carry a single priority number
  - Priority = remaining flow size (e.g., #bytes unACKed)
- Switches
  - Very small queues (e.g., 10 packets)
  - Send highest priority / drop lowest priority packet
- Servers
  - Transmit/retransmit aggressively (at full link rate)
  - Drop transmission rate only under extreme loss (timeouts)

# FCT



# Why does pFabric work?

- Consider problem of scheduling  $N$  jobs at a single queue/processor
  - $J_1, J_2, \dots, J_n$  with duration  $T_1, T_2, \dots, T_n$  respectively
- “Shortest Job First” (SJF) scheduling minimizes average Job Completion Time
  - Pick job with minimum  $T_i$ ; de-queue and run; repeat
  - i.e., job that requires minimum runtime has maximum priority
- Solution for a network of queues is NP-hard
- Setting priority=remaining flow size is a heuristic to approximate SJF

# Questions?