

EECS 489

Computer Networks

Winter 2024

Mosharaf Chowdhury

Material with thanks to Aditya Akella, Sugih Jamin, Philip Levis, Sylvia Ratnasamy, Peter Steenkiste, and many other colleagues.

Agenda

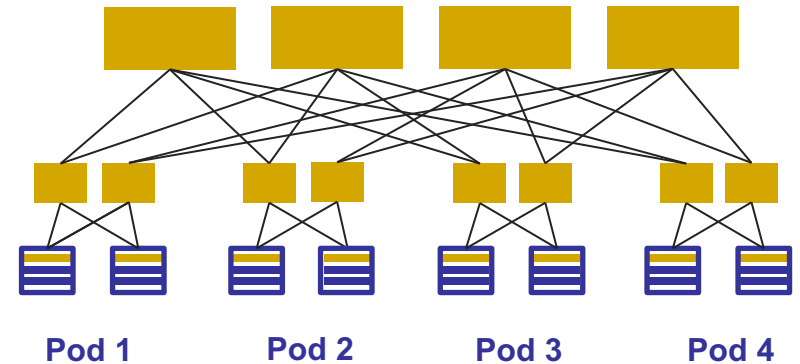
- Datacenter networking

Recap: Datacenter network requirements

- High “bisection bandwidth”
- Low latency, even in the worst-case
- Large scale
- Low cost

Recap: Clos topology

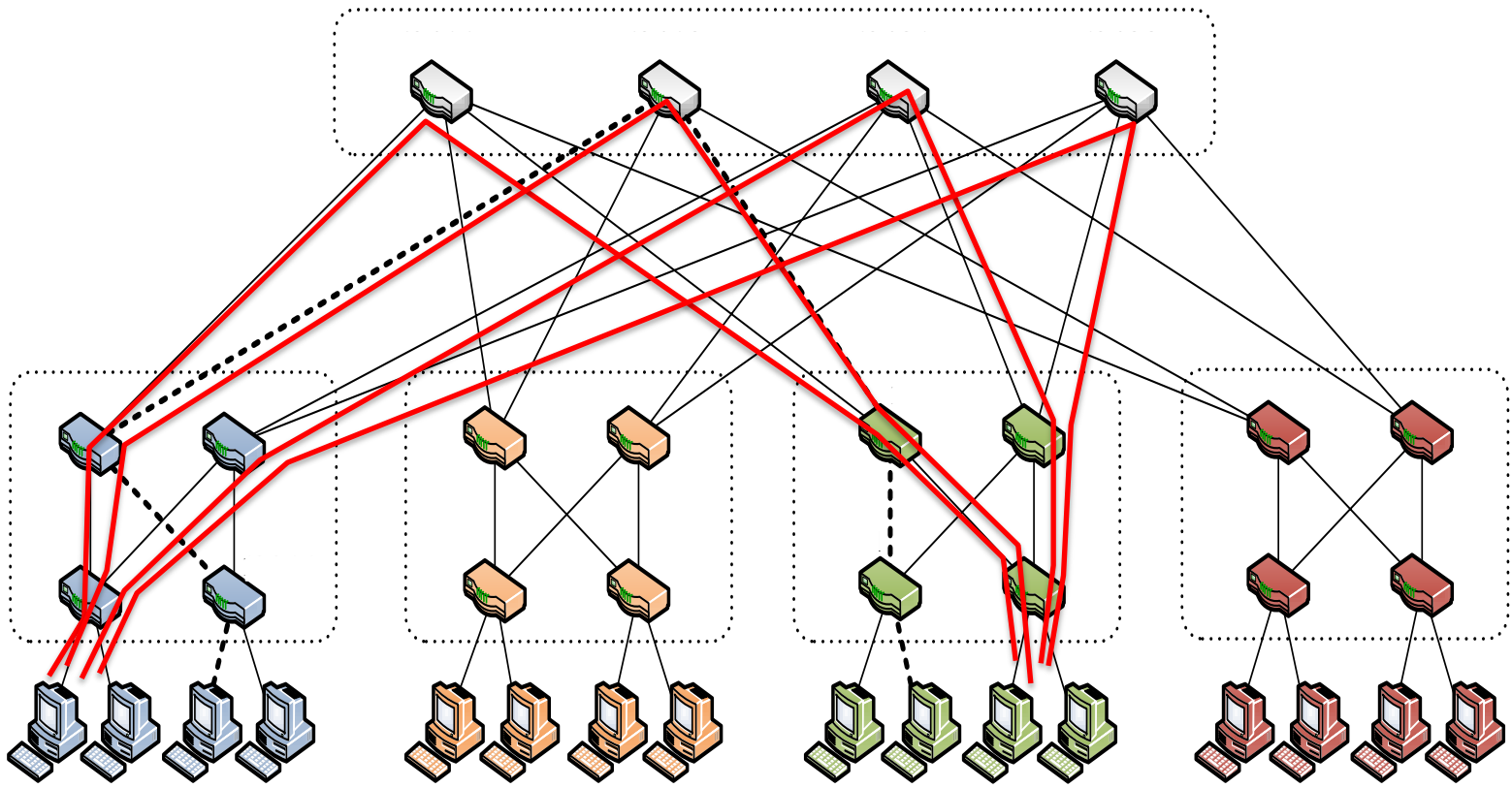
- Multi-stage network
- k pods, where each pod has two layers of $k/2$ switches
 - $k/2$ ports up and $k/2$ down
- All links have the same b/w
- At most $k^3/4$ machines
- Example
 - $k = 4$
 - 16 machines
- For $k=48$, 27648 machines



Agenda

- Networking in modern datacenters
 - L2/L3 design
 - » Addressing / routing / forwarding in the Fat-Tree
 - L4 design
 - » Transport protocol design (w/ Fat-Tree)
 - L7 design
 - » Exploiting application-level information (w/ Fat-Tree)

Using multiple paths well



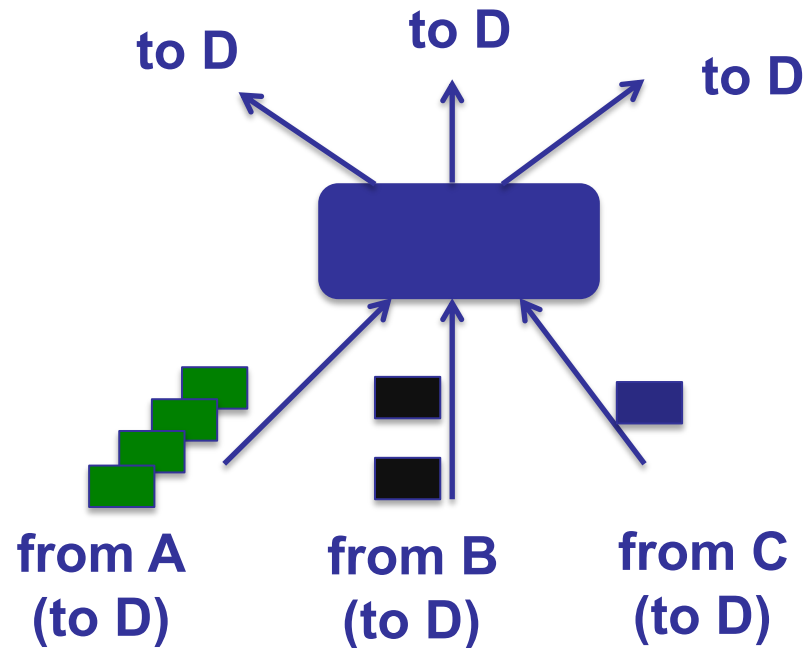
L2/L3 design goals

- Routing protocol must expose all available paths
- Forwarding must spread traffic evenly over all paths

Extend DV / LS ?

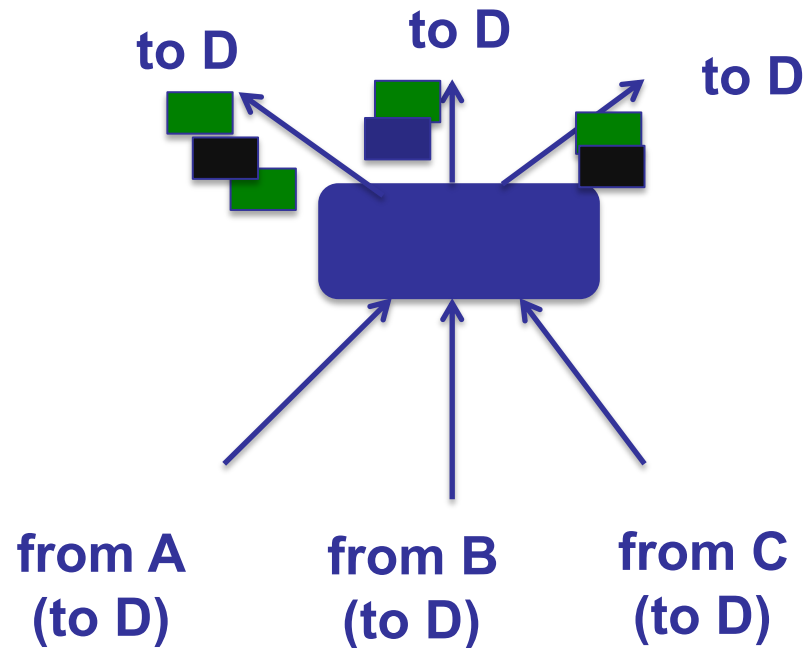
- Routing
 - Distance-Vector: Remember all next-hops that advertise equal cost to a destination
 - Link-State: Extend Dijkstra's to compute all equal cost shortest paths to each destination
- Forwarding: how to spread traffic across next hops?

Forwarding



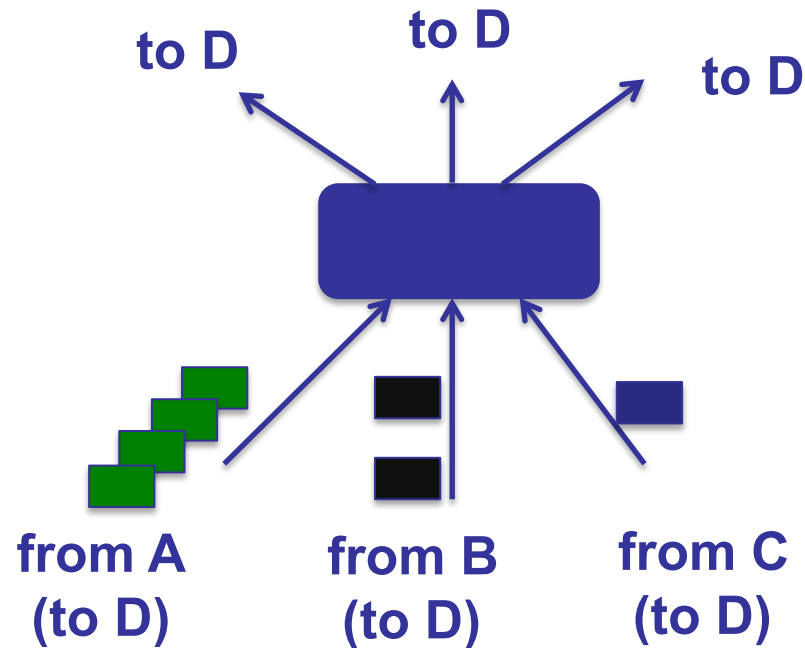
- Per-packet load balancing

Forwarding



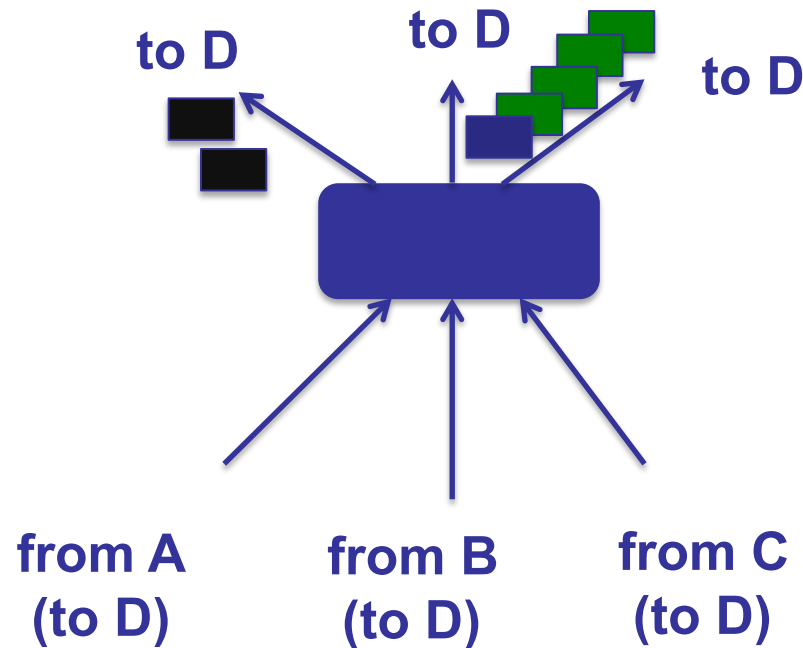
- Per-packet load balancing
 - Traffic well spread (even w/ elephant flows)
 - **BUT** Interacts poorly w/ TCP

Forwarding



- Per-flow load balancing (ECMP, “Equal Cost Multi Path”)
 - E.g., based on (src and dst IP and port)

Forwarding

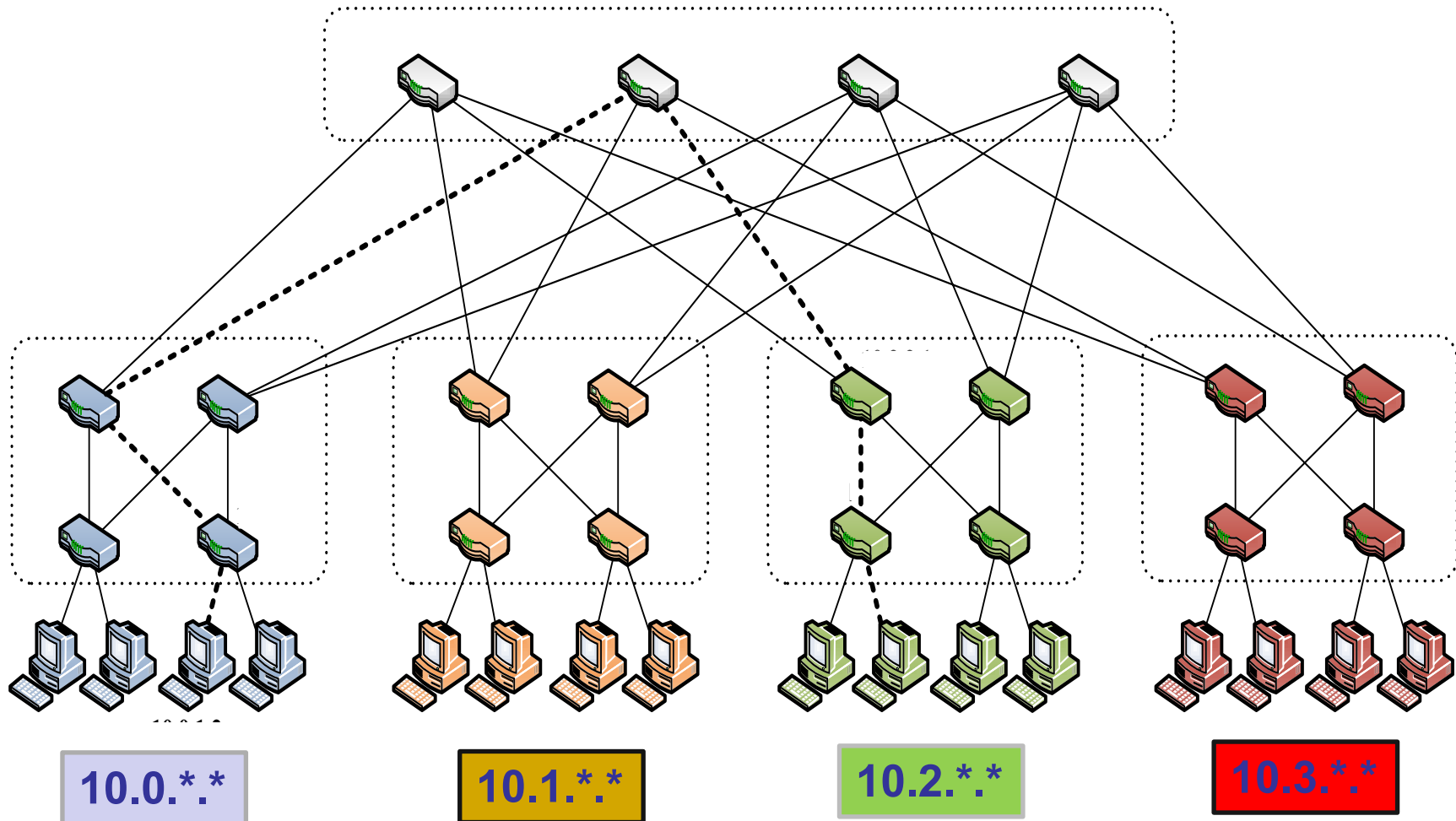


- Per-flow load balancing (ECMP)
 - A flow follows a single path (→ TCP is happy)
 - Suboptimal load-balancing; elephants are a problem¹²

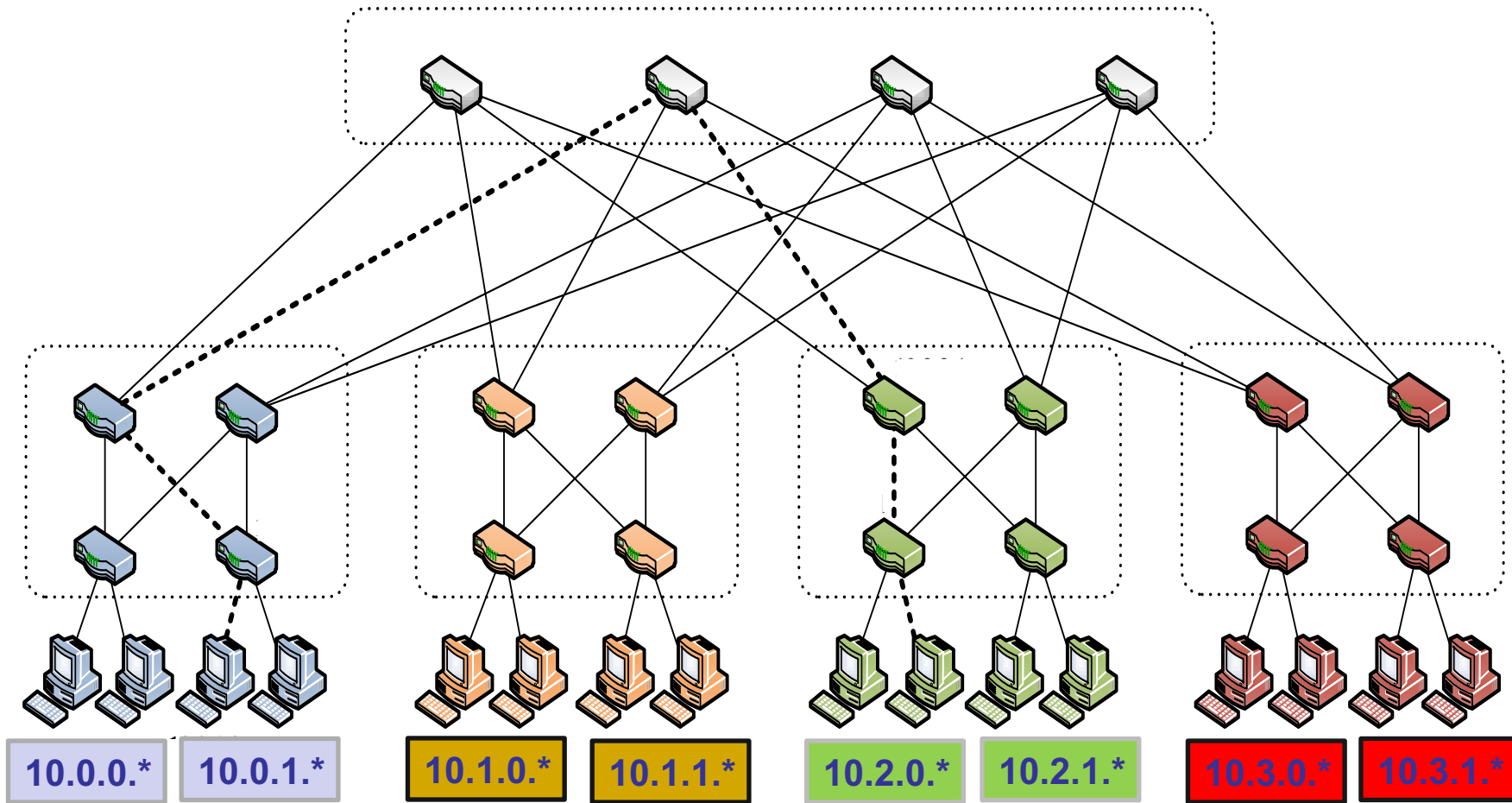
Extend DV / LS ?

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

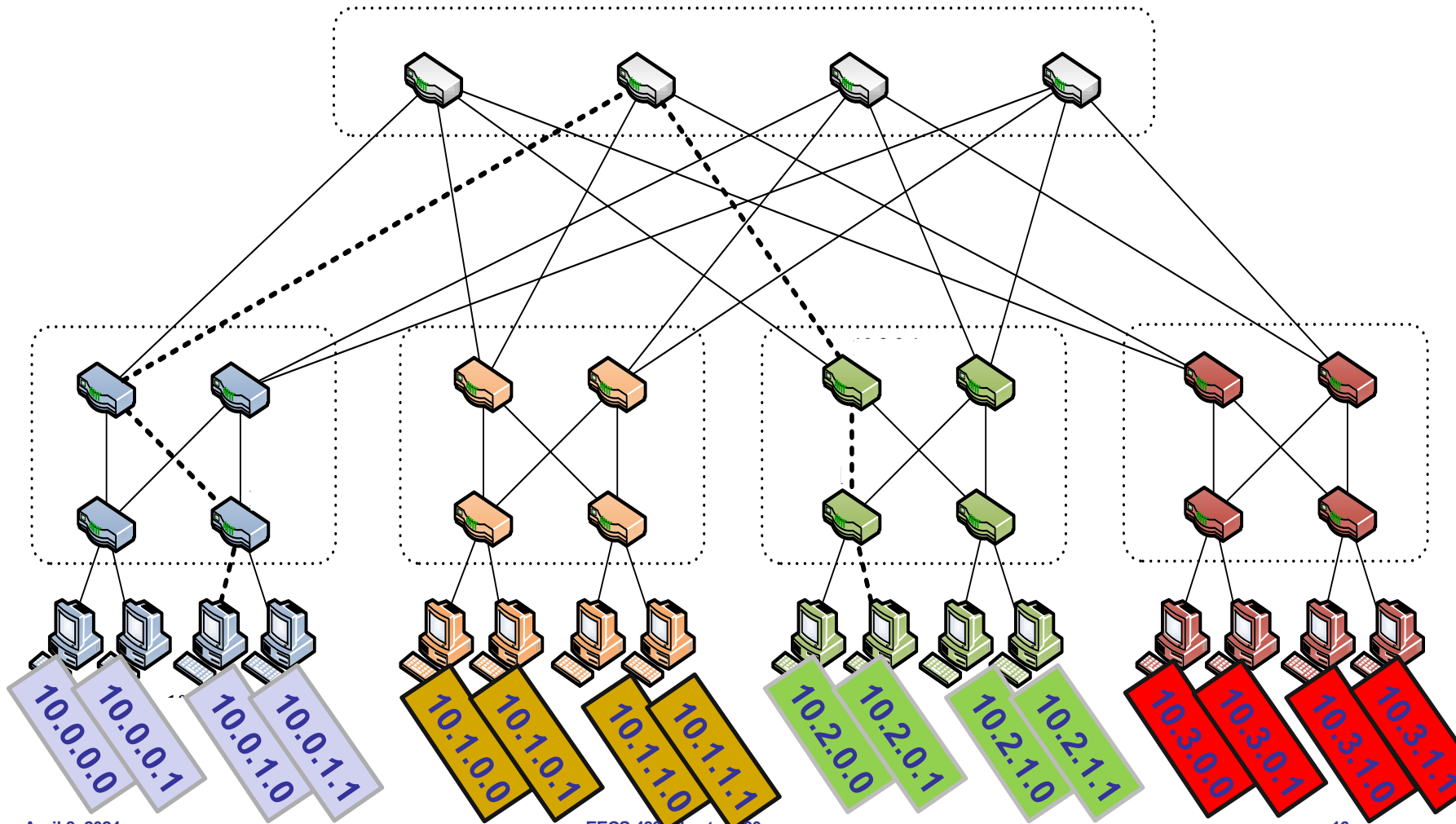
Solution 1: Topology-aware addressing



Solution 1: Topology-aware addressing

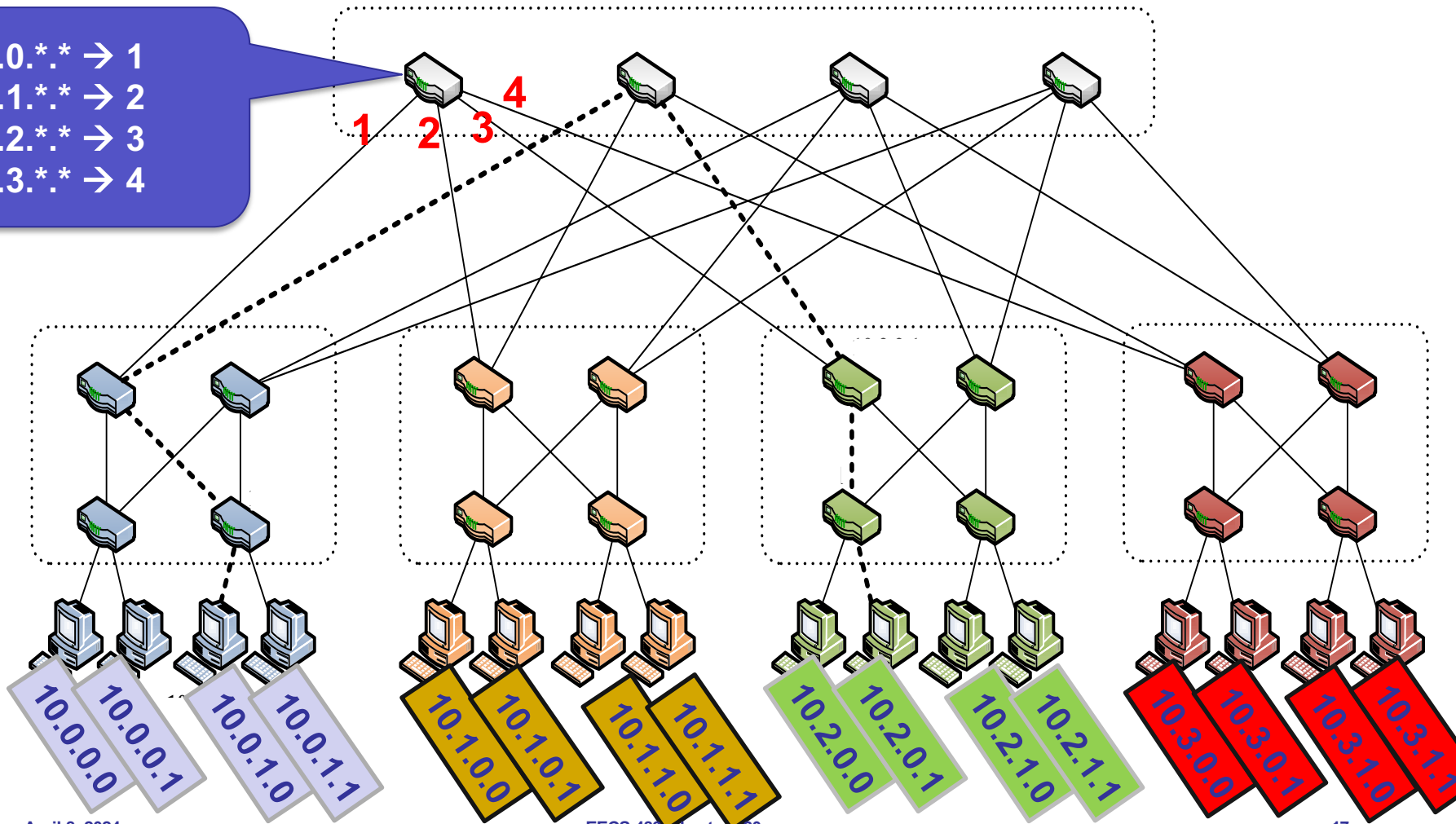


Solution 1: Topology-aware addressing

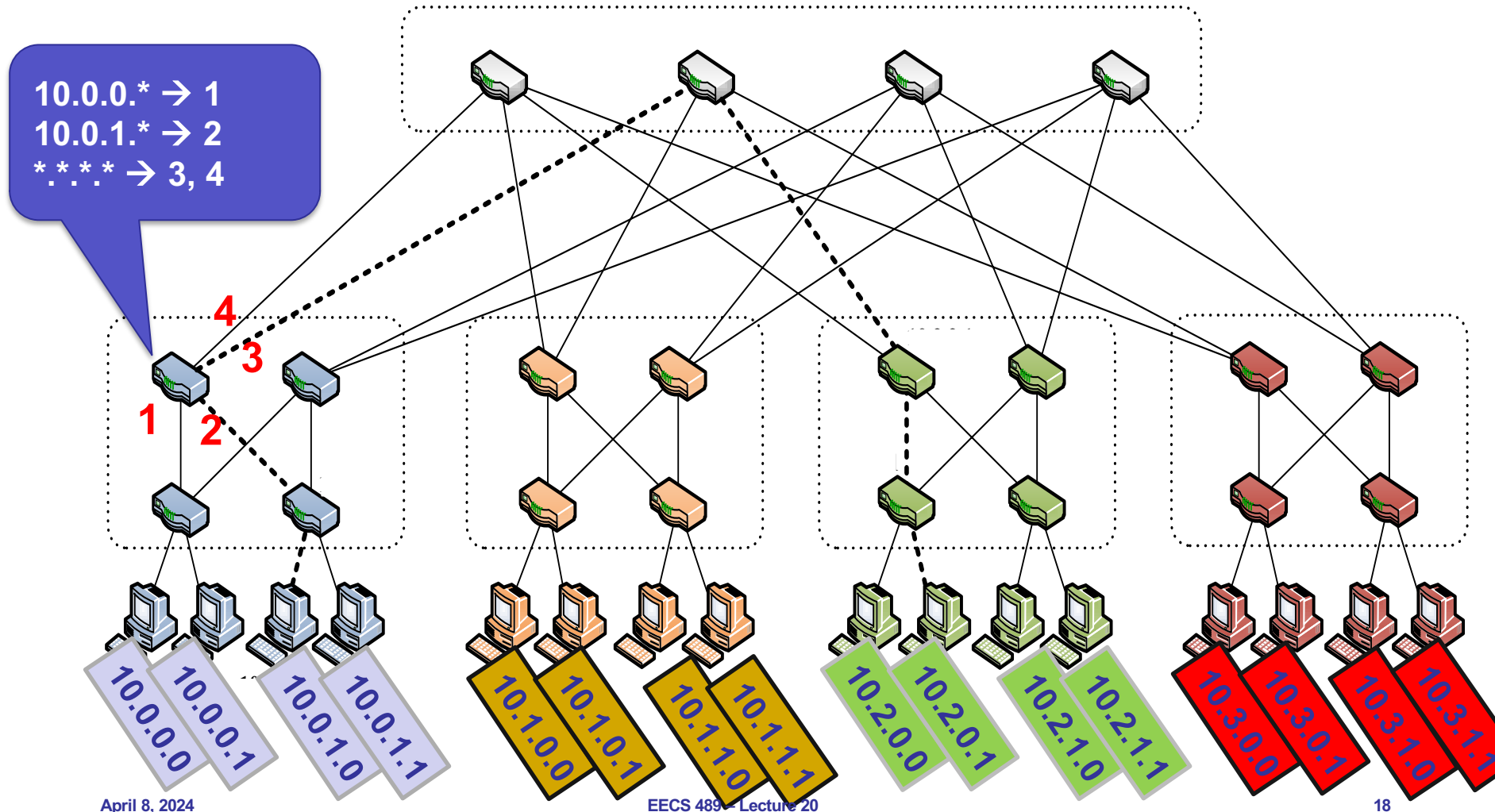


Solution 1: Topology-aware addressing

10.0.* → 1
10.1.* → 2
10.2.* → 3
10.3.* → 4

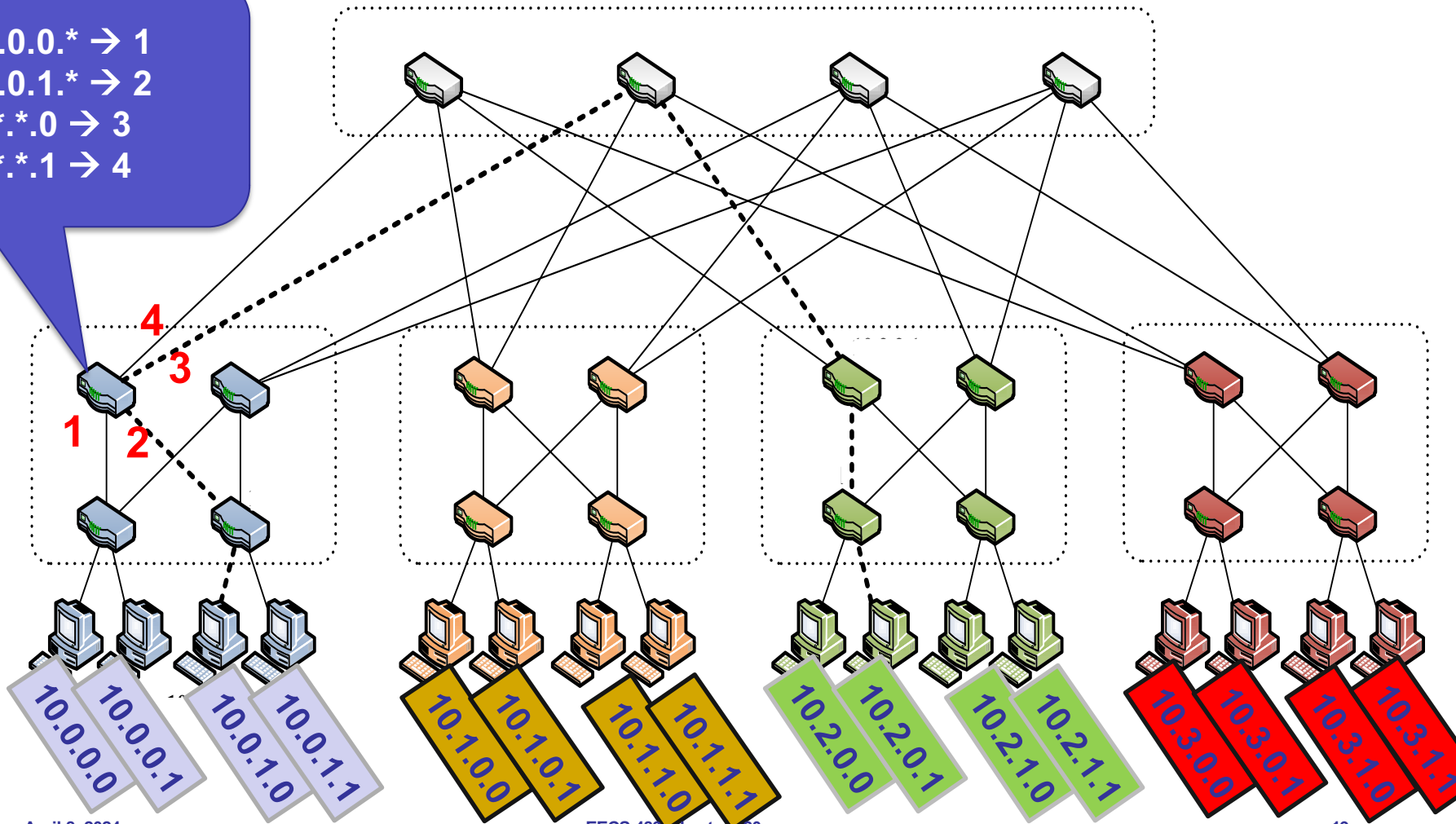


Solution 1: Topology-aware addressing

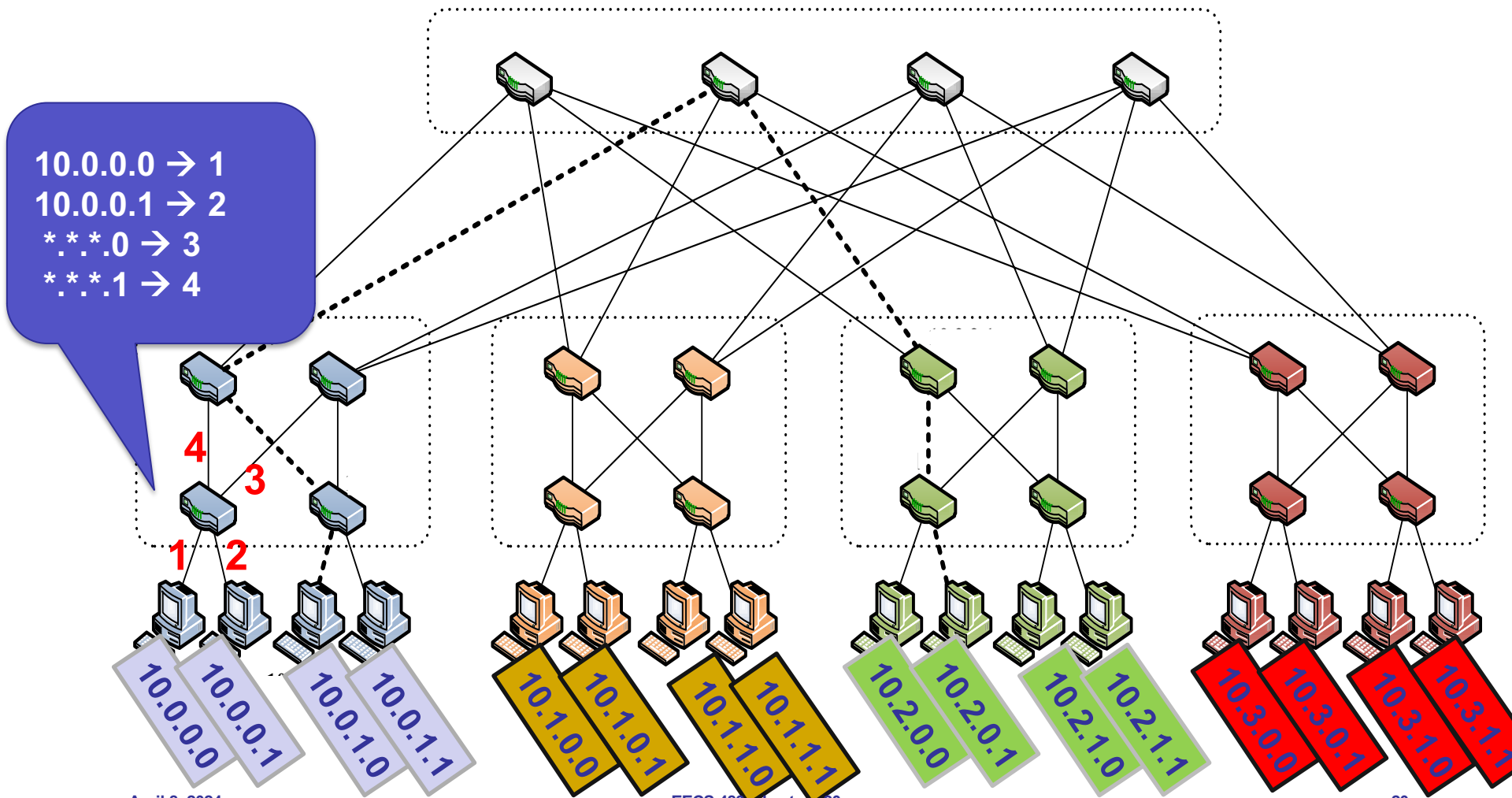


Solution 1: Topology-aware addressing

10.0.0.* → 1
10.0.1.* → 2
..0 → 3
..1 → 4



Solution 1: Topology-aware addressing



Solution 1: Topology-aware addressing

- 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, but still need localized link failure detection
- Problems?
 - VM migration: ideally, VM keeps its IP address when it moves
 - Vulnerable to (topology/addresses) misconfiguration

Solution 2: Centralize + Source routes

- Centralized “controller” server knows topology and 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
 - index++; next-hop = path[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
 - Flexibility: end-points control route selection
- Cons:
 - Scalability / robustness of controller (SDN issue)
 - Clean-slate design of everything

5-MINUTE BREAK!

Announcements

- Teaching evaluations
 - 75% or higher completion rate will result in +1 on the final grade for everyone

Agenda

- Networking in modern datacenters
 - L2/L3 design
 - » Addressing / routing / forwarding in the Fat-Tree
 - L4 design
 - » Transport protocol design (w/ Fat-Tree)
 - L7 design
 - » Exploiting application-level information (w/ Fat-Tree)

Workloads

- Partition-Aggregate traffic from user-facing queries
 - Numerous short flows with small traffic footprint
 - Latency-sensitive
- Map-Reduce traffic from data analytics
 - Comparatively fewer large flows with massive traffic footprint
 - Throughput-sensitive

Tension between requirements

High throughput

- Deep queues at switches
 - Queueing delays increase latency

Low latency

- Shallow queues at switches
 - Bad for bursts and throughput

Objective:
Low Queue Occupancy & High Throughput

Data Center TCP (DCTCP)

- Proposal from Microsoft Research, 2010
 - Incremental fixes to TCP for DC environments
 - Deployed in Microsoft datacenters (~rumor)
- Leverages Explicit Congestion Notification (ECN)

Recap: Explicit Congestion Notification (ECN)

- Defined in RFC 3168 using ToS/DSCP bits in the IP header
- Single bit in packet header; set by congested routers
 - If data packet has bit set, then ACK has ECN bit set
- Routers typically set ECN bit based on average queue length
- Congestion semantics exactly like that of drop
 - I.e., sender reacts as though it saw a drop

DCTCP: Key ideas

- React early, quickly, and with certainty using ECN
- React in **proportion to the extent of congestion**, not its presence

ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%

Actions due to DCTCP

- At the switch
 - If **instantaneous** queue length $> k$
 - » Set ECN bit in the packet
- At the receiver
 - If ECN bit is set in a packet, set ECN bit for its ACK
- At the sender
 - Maintain an EWMA of the fraction of packets marked (α)
 - Adapt window based on α : $W \leftarrow (1 - \alpha/2) W$
 - $\alpha = 1$ implies high congestion: $W \leftarrow W/2$ (like TCP)

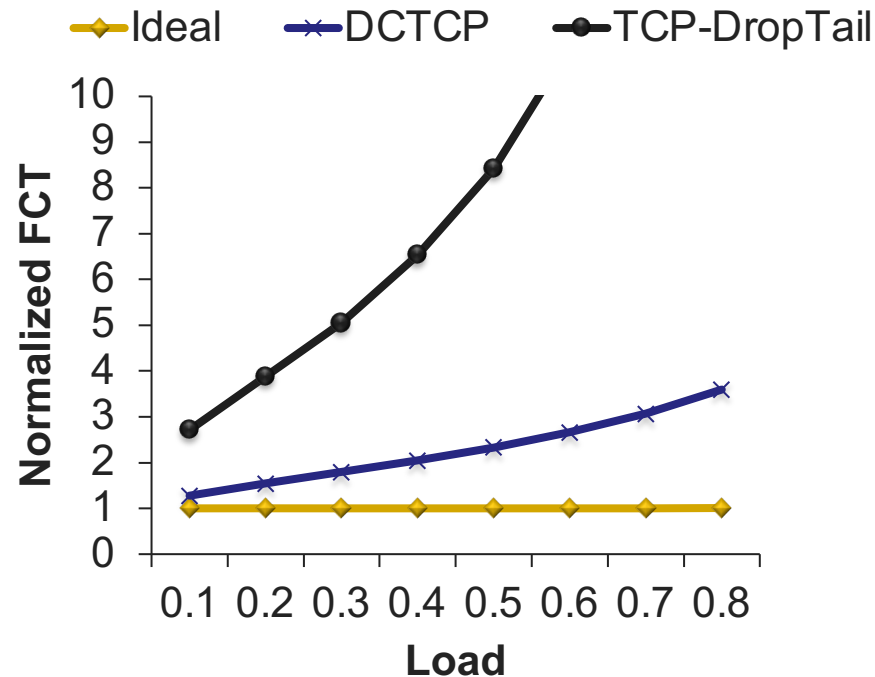
DCTCP: Why it works

- React early and quickly: use ECN
 - Avoid large buildup in queues → lower latency
- React in proportion to the extent of congestion, not its presence
 - Maintain high throughput by not over-reacting to congestion
 - Reduces variance in sending rates, lowering queue buildups
- Still far from ideal

Flow Completion Time (FCT)

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

FCT with DCTCP



Queues are still shared \Rightarrow Head-of-line blocking

Solution: Use priorities!

- Packets carry a single priority number
 - Priority = remaining flow size
- 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)
- Provides FCT close to the ideal

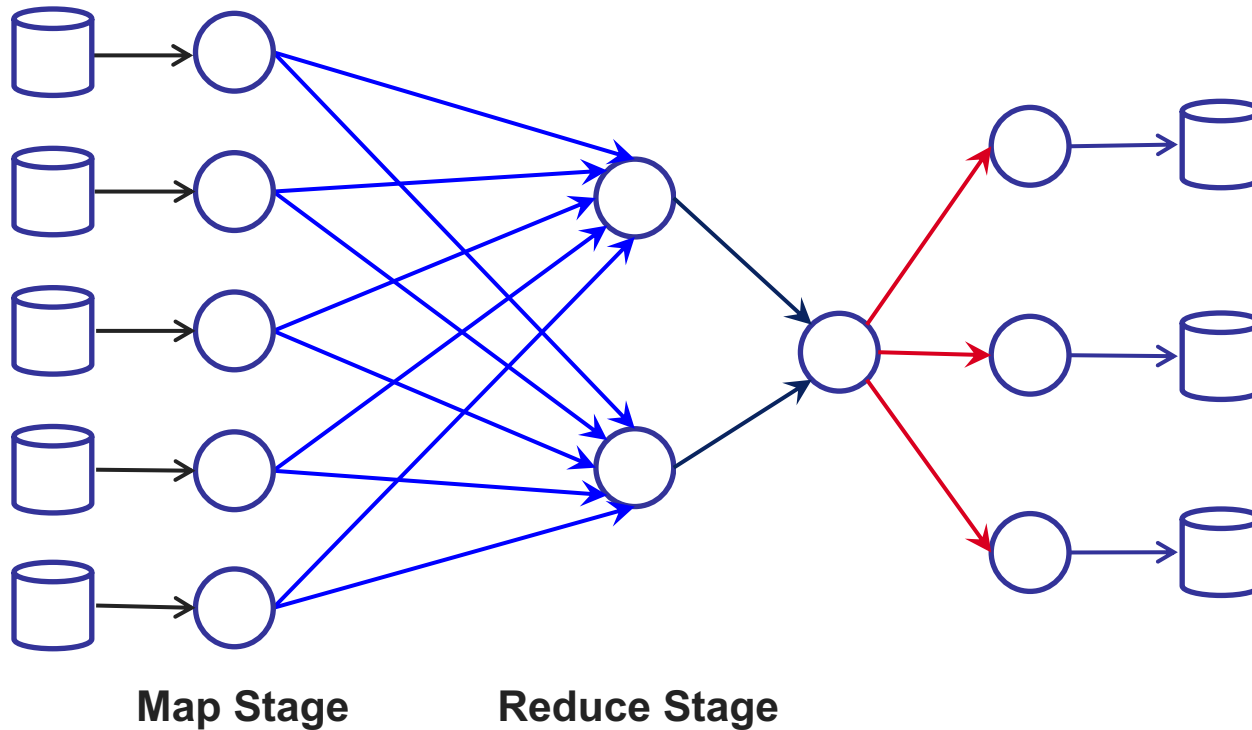
Are we there yet?

- Nope!
- Someone asked “What do datacenter applications *really* care about?”

Agenda

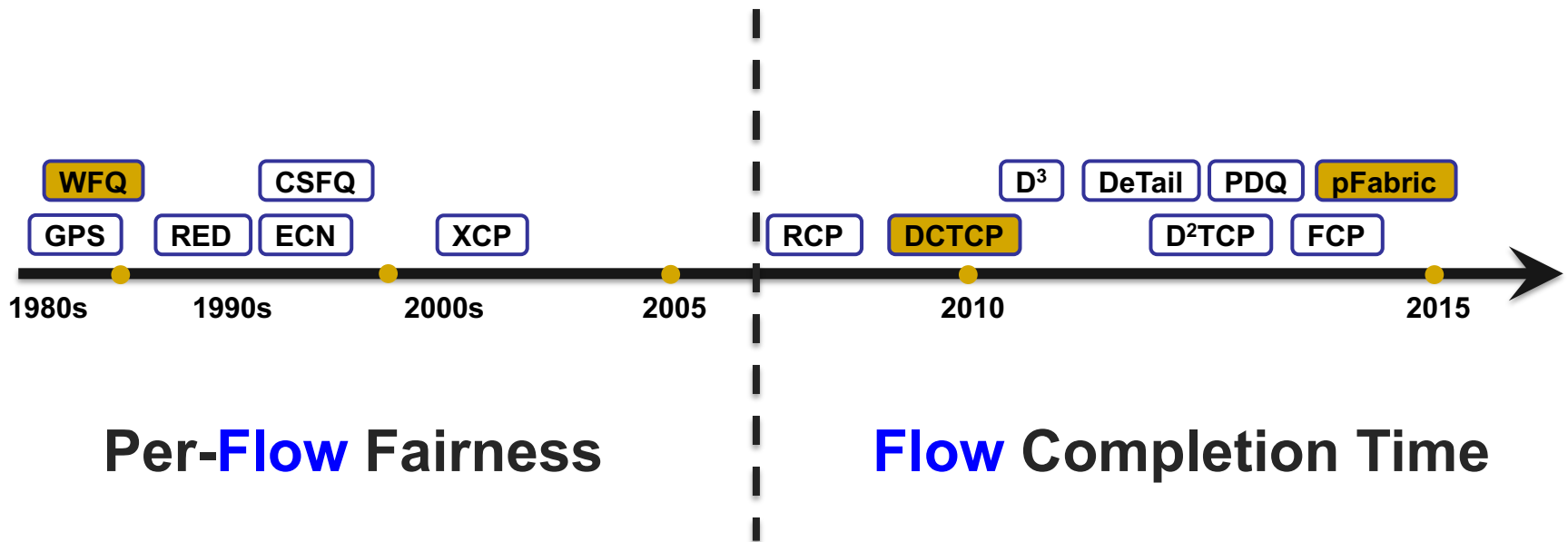
- Networking in modern datacenters
 - L2/L3 design
 - » Addressing / routing / forwarding in the Fat-Tree
 - L4 design
 - » Transport protocol design (w/ Fat-Tree)
 - L7 design
 - » Exploiting application-level information (w/ Fat-Tree)

The Map-Reduce Example



Observation:
A communication stage cannot complete until all its flows have completed

Flow-based solutions

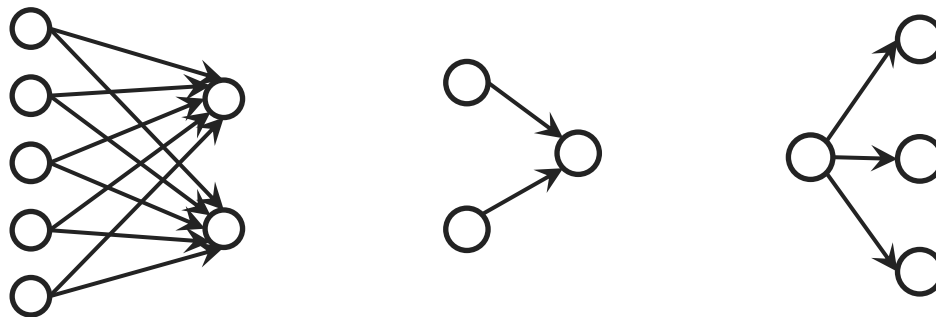


Independent flows cannot capture collective communication patterns that are common in data-parallel applications

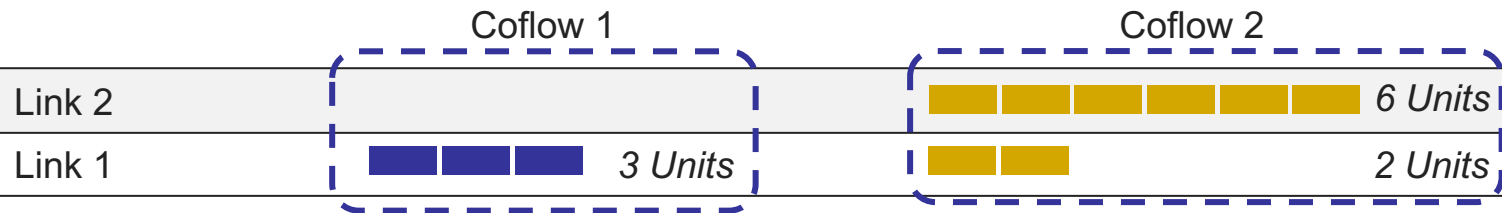
The Coflow abstraction

[SIGCOMM'14]

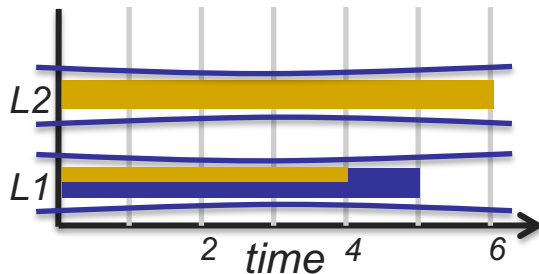
- Coflow is a communication abstraction for data-parallel applications to express their performance goals; e.g.,
 - Minimize completion times,
 - Meet deadlines, or
 - Perform fair allocation
- Not for individual flows; for entire stages!



Benefits of inter-coflow scheduling

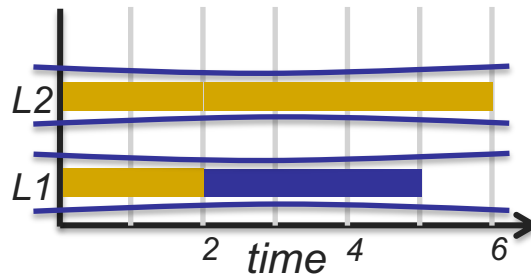


Fair Sharing (TCP, DCTCP)



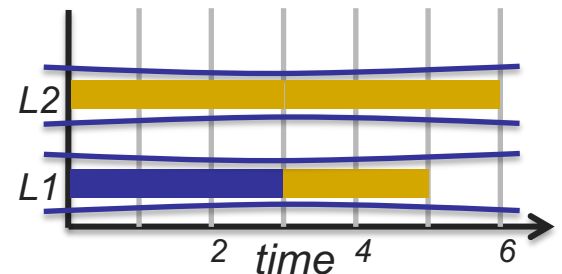
Coflow1 comp. time = 5
Coflow2 comp. time = 6
Average FCT = 5

Smallest-Flow First (pFabric)



Coflow1 comp. time = 5
Coflow2 comp. time = 6
Average FCT = 4.33

Smallest-Coflow First



Coflow1 comp. time = 3
Coflow2 comp. time = 6
Average FCT = 4.67

Coflow completion time (CCT) is a better predictor of job-level performance than FCT

Summary

- Networking in modern datacenters
 - L2/L3: Source routing and load balancing to exploit multiple paths over the Clos topology
 - L4: Find a better balance between latency and throughput requirements
 - L7: Exploit application-level information with **coflows**
- Last class: Final Review



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 retransmit, reduce CWND, retransmit #1, ...
 - Repeatedly!
- Information sharing between multiple paths affects TCP
 - One RTT and timeout estimator for multiple paths
 - CWND halved when a packet is dropped on any path

Multipath TCP

- Multipath TCP (MPTCP) is an ongoing effort to extend TCP to coexist with multipath routing
 - Value beyond datacenters (e.g., spread traffic across WiFi and 4G access)

What's ideal for a 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?

How to implement coflows?

- Modify applications to annotate coflows
 - Possible to infer them as well [SIGCOMM'16]
- Managed communication
 - Applications do not communicate; instead, a central entity does the communication on their behalf
- Centralized scheduling