

# **EECS 489**

# **Computer Networks**

**Winter 2023**

Z. Morley Mao

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

# Logistics

---

- Open book/text/notes, but OFFLINE
  - Except for taking the exam over the Internet
- You're NOT allowed to write/run any programs
- You're NOT allowed to collaborate with anyone

# General guidelines (1)

---

- Test only assumes material covered in lecture, discussion sections, quizzes, and assignments
  - Text: only to clarify details and context for the above
- The test doesn't require you to do complicated calculations
  - Use this as a hint to determine if you're on right track
- You don't need to memorize anything
- You do need to understand how things work

# General guidelines (2)

---

- Be prepared to:
  - Weigh design options outside of the context we studied them in
  - Contemplate new designs we haven't covered in detail but can be put together
    - »e.g., I introduce a new IP address format; how does this affect..”
  - Reason from what you know about the pros/cons of solutions we did study

# General guidelines (3)

---

- Exam format
  - Q1: True-False questions
    - » Wrong answer results in negative marks
  - Q2: MCQ questions
  - Q3-QN networking use cases
    - » Questions not ordered in terms of complexity
- ~75 minutes
  - About 10 minutes more than a typical EECS489 midterm *without* increasing complexity

# This review

---

- Walk through what you're expected to know at this point: key topics, important aspects of each
- Not covered in review **does NOT imply** you don't need to know it
  - But if it's covered today, you should know it
- Summarize, not explain
  - Stop me when you want to discuss something further!

# Topics

---

- Basics (lectures 1–2)
- Application layer (lectures 3–5)
  - HTTP, DNS, CDN, Video Streaming, and Cloud
- Transport layer (lectures 6–9)
  - UDP vs. TCP
  - TCP details: reliability and flow control
  - TCP congestion control: general concepts only
- Network layer (lecture 10–11)
  - Overview
  - Data plane

# Basic concepts

---

- You should know:
  - Packet vs. circuit switching
  - Statistical multiplexing
  - Link characteristics
  - Packet delays



# Switched networks

---

- End-systems and networks connected by switches instead of directly connecting them
- Allows us to **scale**
  - For example, directly connecting  $N$  nodes to each other would require  $N^2$  links!

# Two approaches to sharing

---

- Packet switching
  - Network resources consumed on demand per-packet
  - Admission control: per packet
- Circuit switching
  - Network resources reserved a priori at “connection” initiation
  - Admission control: per connection


# Statistical multiplexing

---

- Allowing more demands than the network can handle
  - Hoping that not all demands are required at the same time
  - Good for bursty traffic (average  $\ll$  peak demand)
  - Packet switching exploits statistical multiplexing better than circuit switching

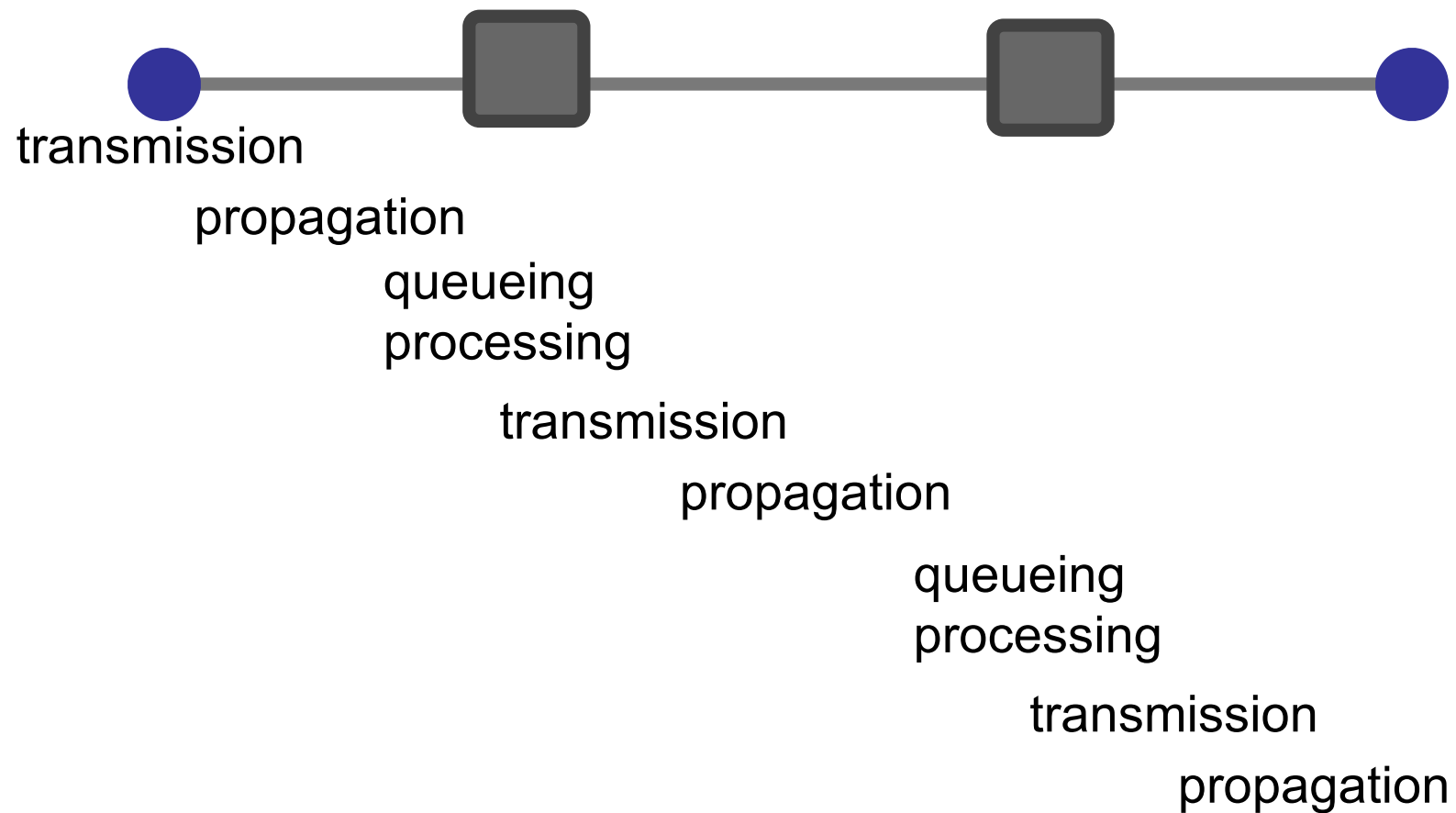
# Delay

---

- Consists of four components
    - Transmission delay
    - Propagation delay
    - Queuing delay
    - Processing delay
- due to link properties
- due to traffic mix and switch internals
- 

# End-to-end delay

---



# What we want

---

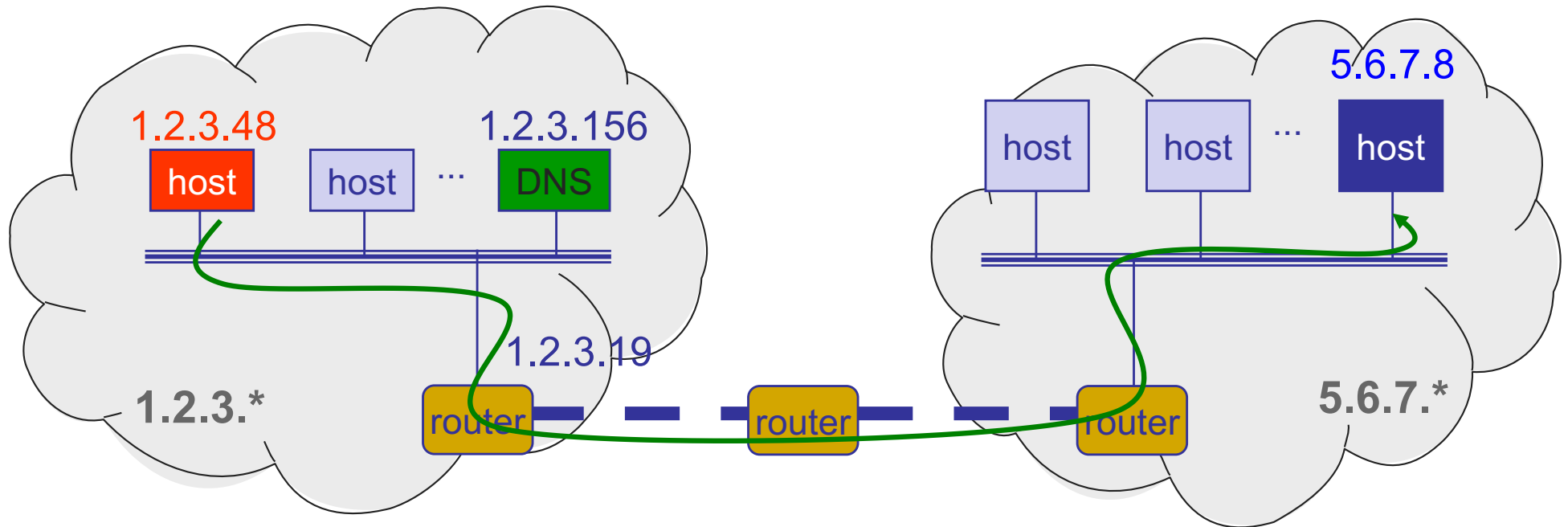
`http://123.xyz`



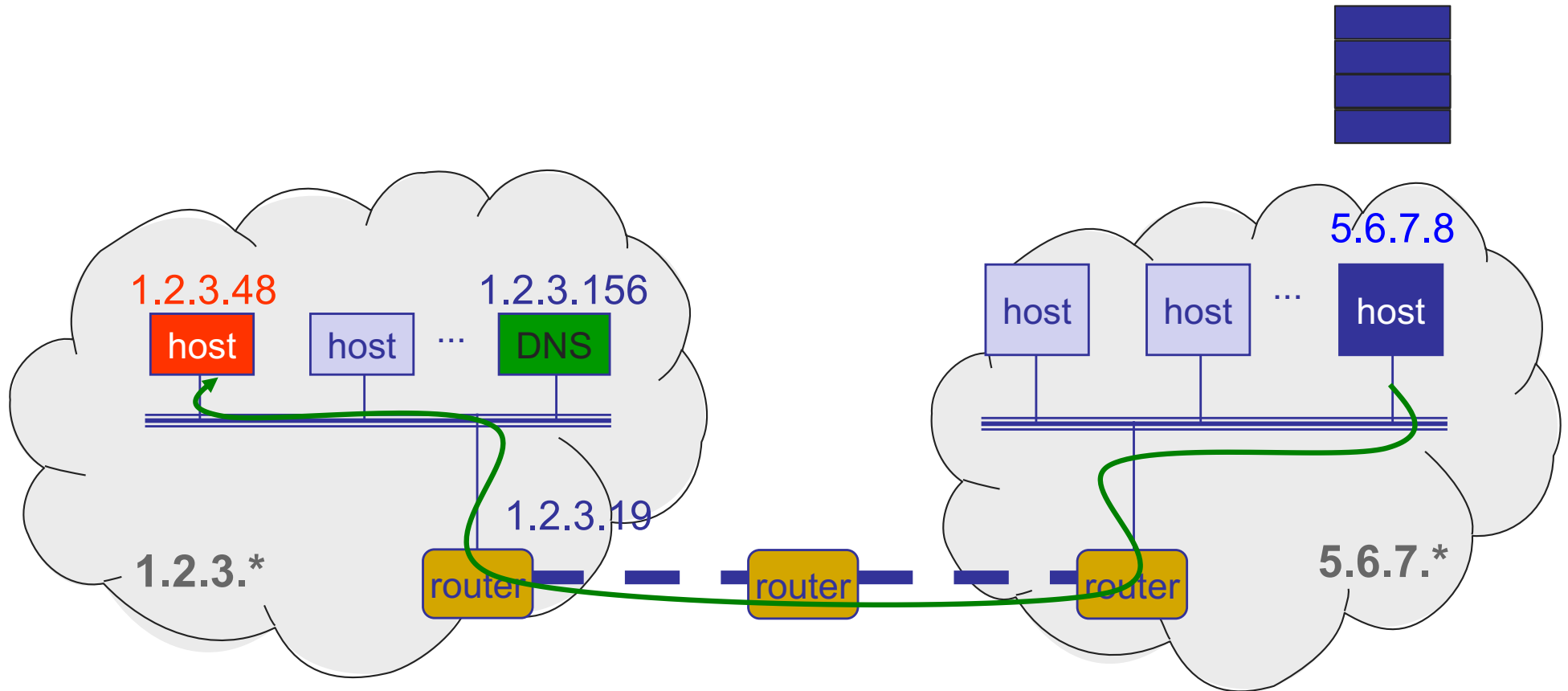
123.xyz server



# (Some of) What happens...



# (More of) What happens





# What we get

---



123.xyz server



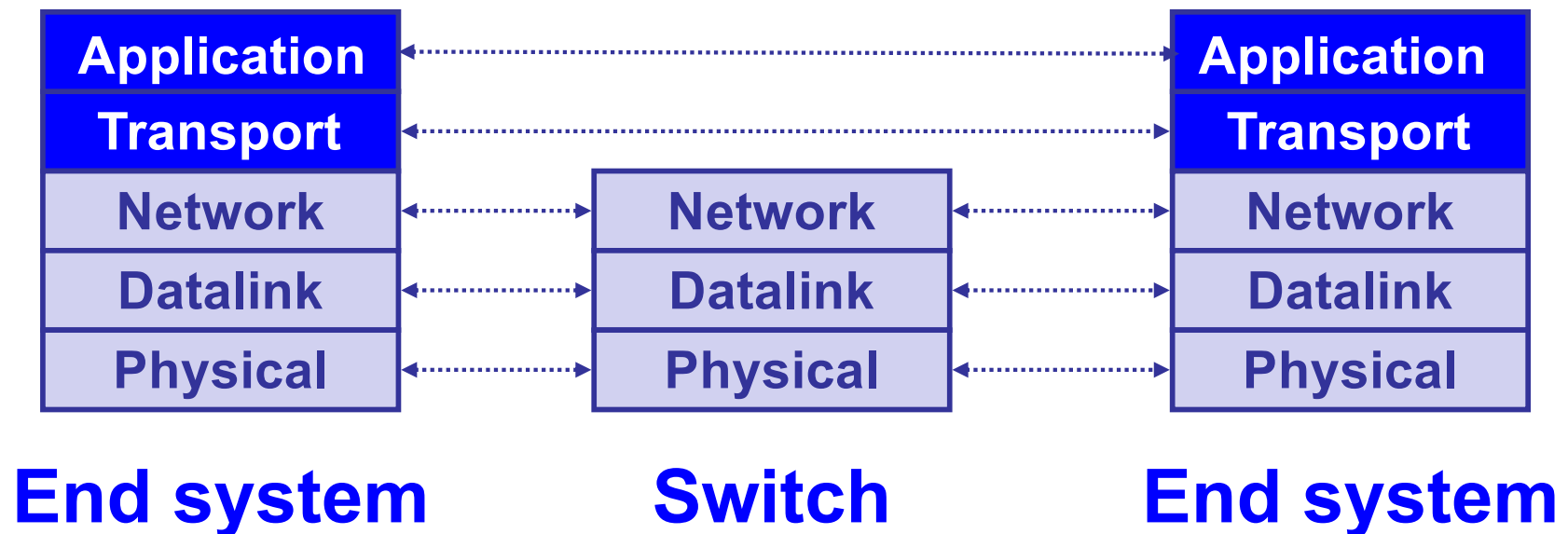
# Layers

---

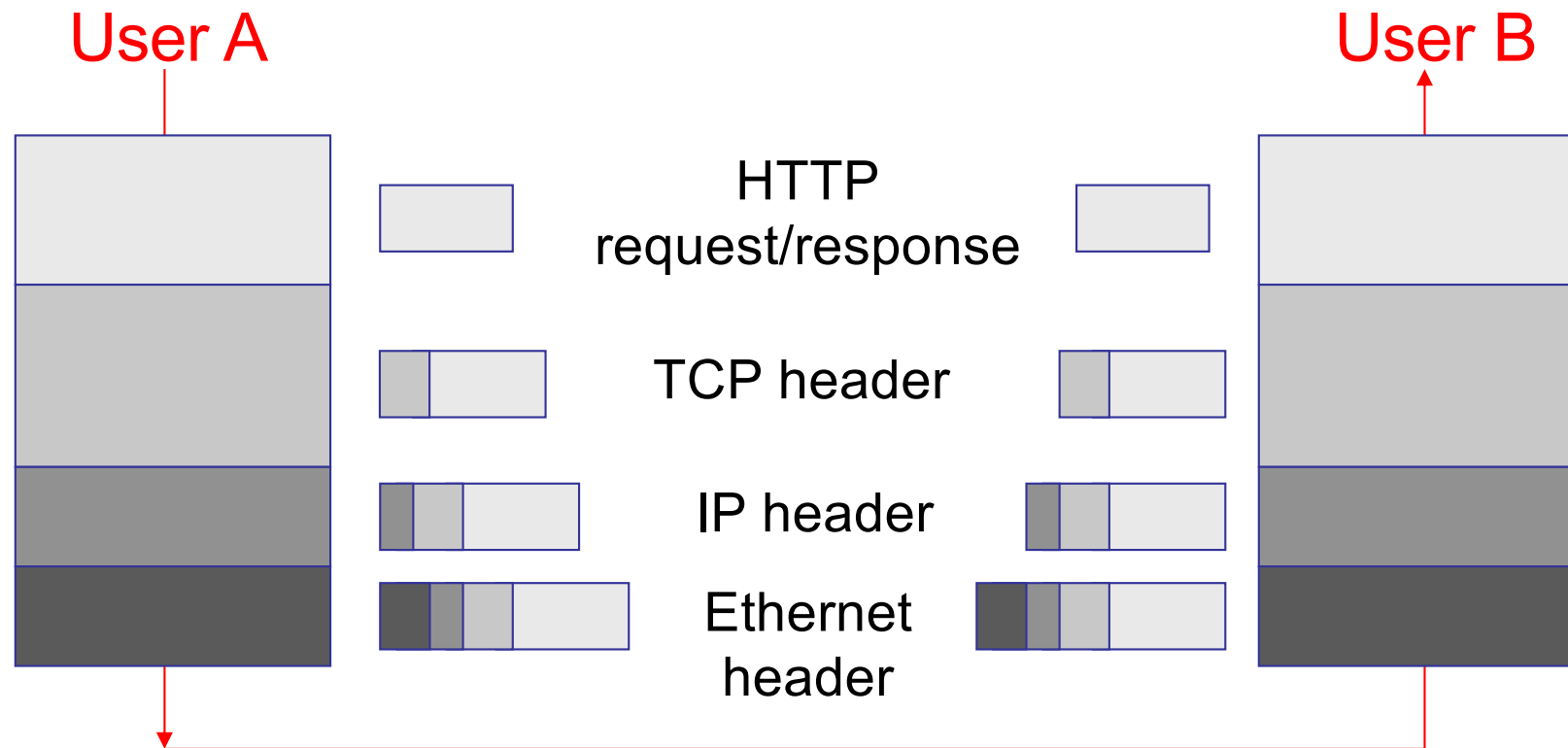
- Layer: a part of a system with well-defined interfaces to other parts
- One layer interacts only with layer above and layer below
- Two layers interact only through the interface between them

# Layers in practice

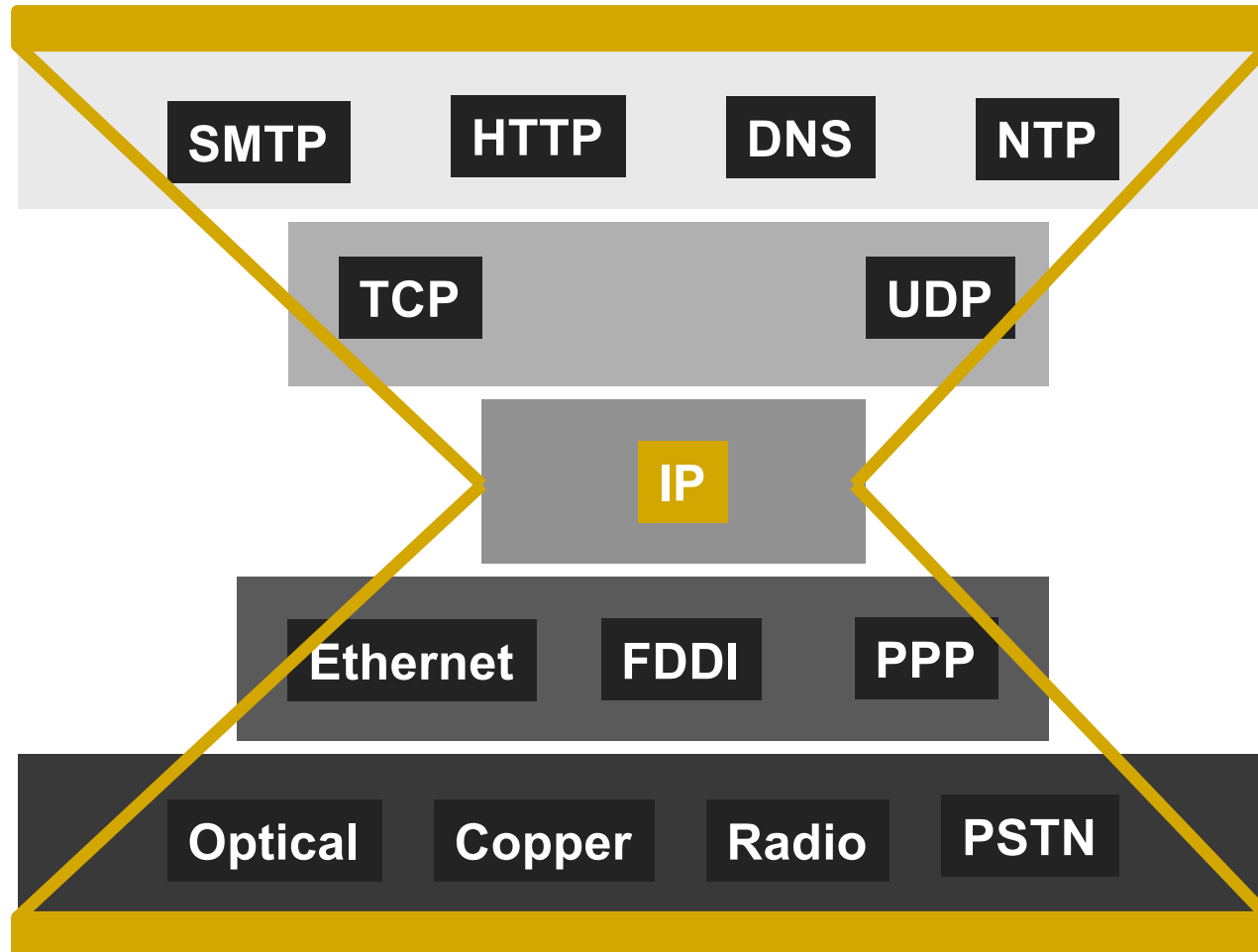
- Lower three layers implemented everywhere
- Top two layers implemented only at hosts



# Layer encapsulation: Protocol headers



# IP is the narrow waist of the layering hourglass



# Topics

---

- Basics (lectures 1–2)
- Application layer (lectures 3–5)
  - HTTP, DNS, CDN, Video Streaming, and Cloud
- Transport layer (lectures 6–9)
  - UDP vs. TCP
  - TCP details: reliability and flow control
  - TCP congestion control: general concepts only
- Network layer (lecture 10–11)
  - Overview
  - Data plane

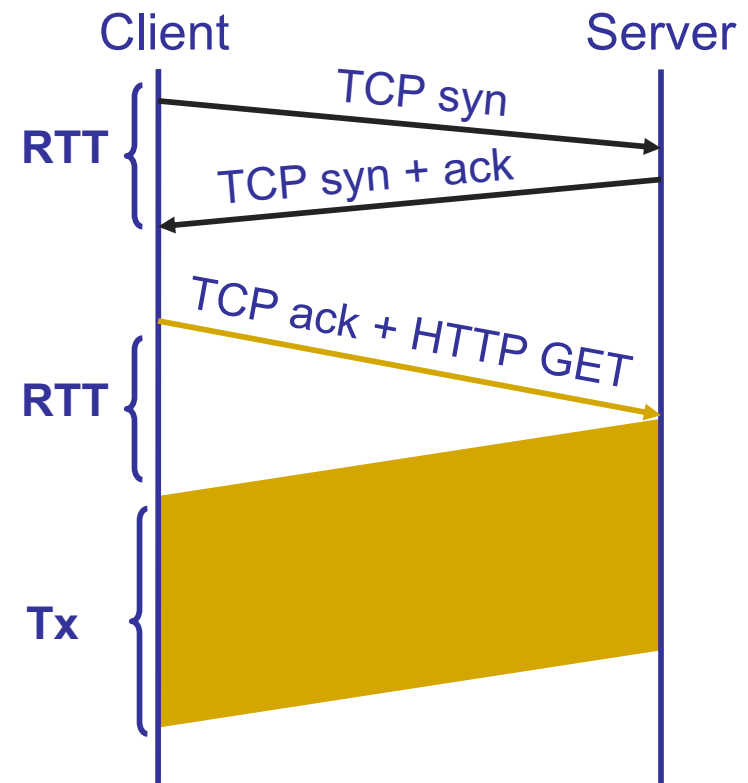
# Hyper Text Transfer Protocol (HTTP)

---

- Client-server architecture
  - Server is “always on” and “well known”
  - Clients initiate contact to server
- Synchronous request/reply protocol
  - Runs over TCP, Port 80
- Stateless
- ASCII format
  - Before HTTP/2

# Object request response time

- RTT (round-trip time)
  - Time for a small packet to travel from client to server and back
- Response time
  - 1 RTT for TCP setup
  - 1 RTT for HTTP request and first few bytes
  - Transmission time
  - **Total** =  $2\text{RTT} + \text{Transmission Time}$





# Improving HTTP performance

---

- Optimizing connections using three “P”s
  - Persistent connections
  - Parallel/concurrent connections
  - Pipelined transfers over the same connection
- Caching
  - Forward proxy: close to clients
  - Reverse proxy: close to servers
- Replication

# Scorecard: Getting $n$ small objects

---

- Time dominated by latency
- One-at-a-time:  $\sim 2n$  RTT
- $m$  concurrent:  $\sim 2\lceil n/m \rceil$  RTT
- Persistent:  $\sim (n+1)$  RTT
- Pipelined:  $\sim 2$  RTT
- Pipelined and Persistent:  $\sim 2$  RTT first time; RTT later for another  $n$  from the same site

# Scorecard: Getting $n$ large objects each of size $F$

---

- Time dominated by TCP throughput  $B_C$  ( $\leq B_L$ ), where link bandwidth is referred by  $B_L$
- One-at-a-time:  $\sim nF/B_C$
- $m$  concurrent:  $\sim nF/(mB_C)$ 
  - Assuming each TCP connection gets the same throughput and  $mB_C \leq B_L$
- Pipelined and/or persistent:  $\sim nF/B_C$ 
  - The only thing that helps is higher throughput

# Content Distribution Networks (CDN)

---

- Caching and replication as a service
- Combination of caching and replication
  - **Pull**: Direct result of clients' requests (caching)
  - **Push**: Expectation of high access rate (replication)

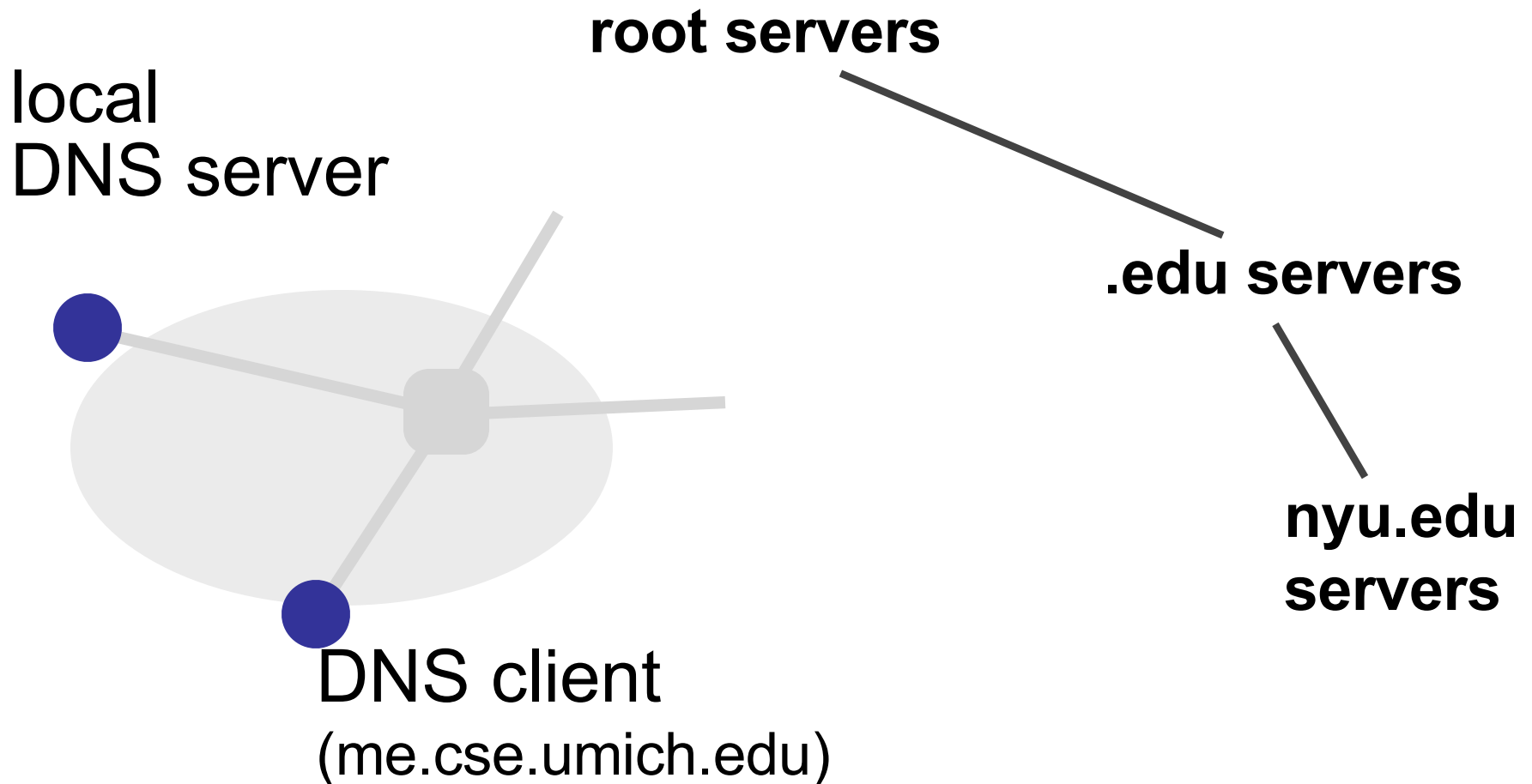
# Hierarchies in the DNS

---

- Three intertwined hierarchies
  - Hierarchical namespace
    - » As opposed to flat namespace
  - Hierarchically administered
    - » As opposed to centralized
  - (Distributed) hierarchy of servers
    - » As opposed to centralized storage

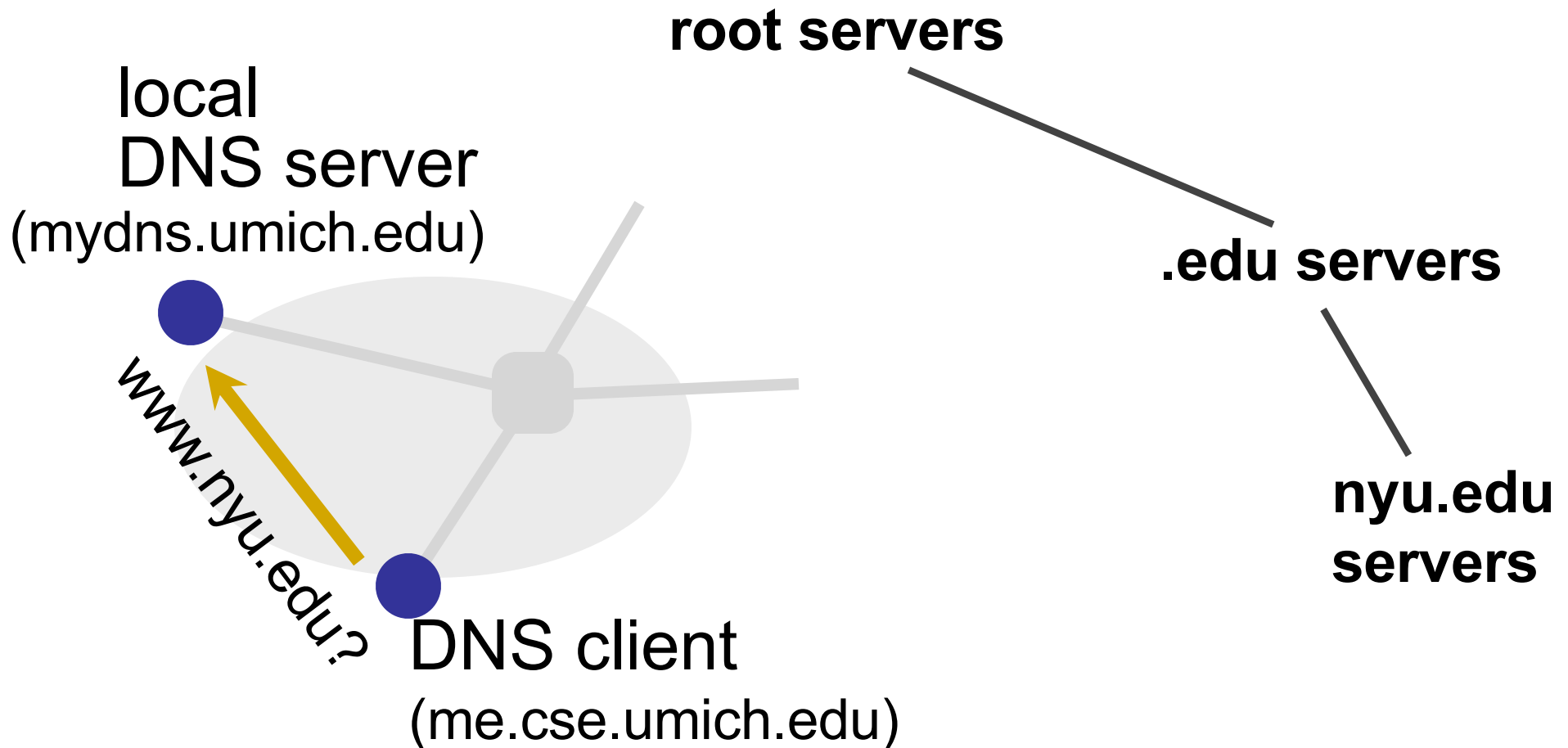
# Name resolution

---



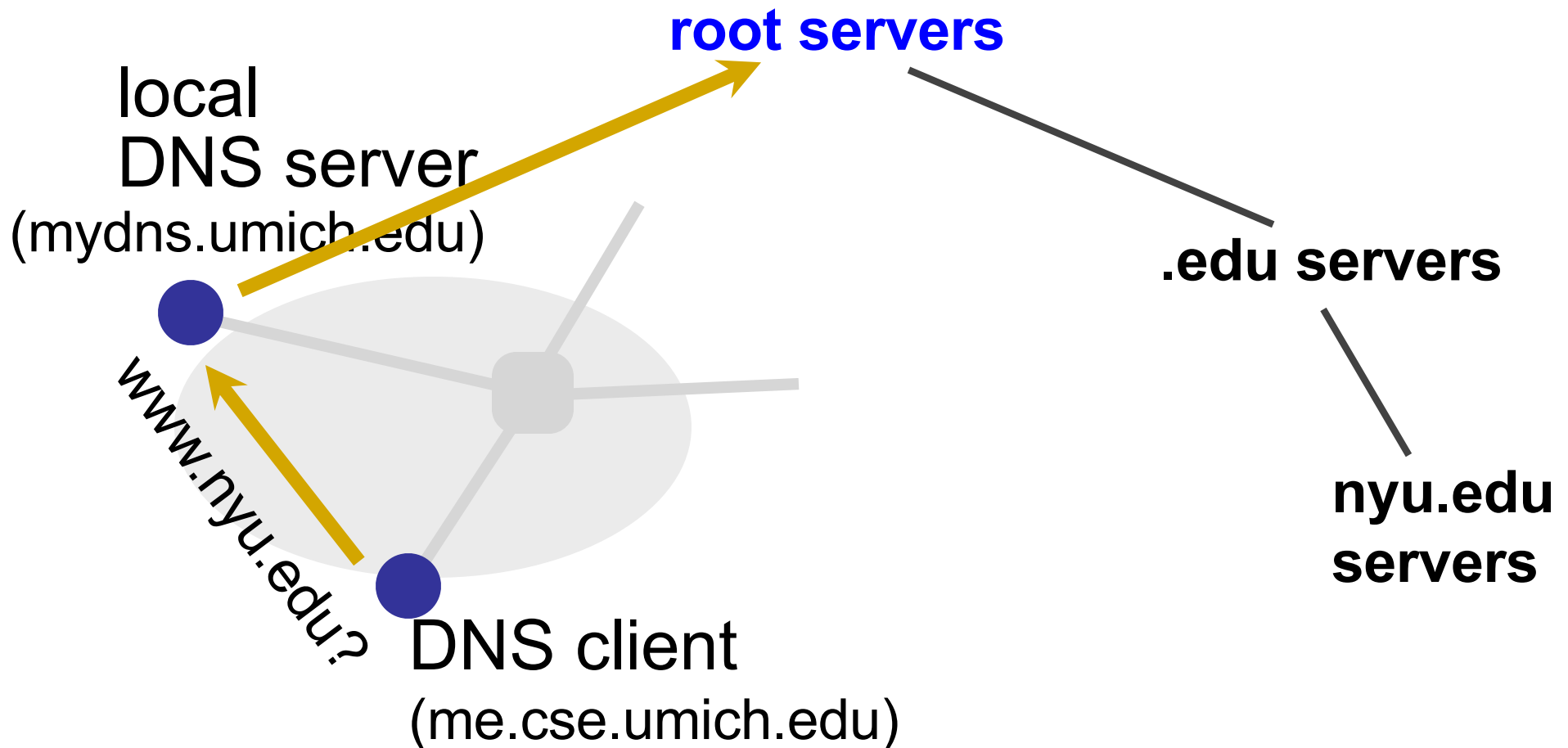
# Name resolution

---



# Name resolution

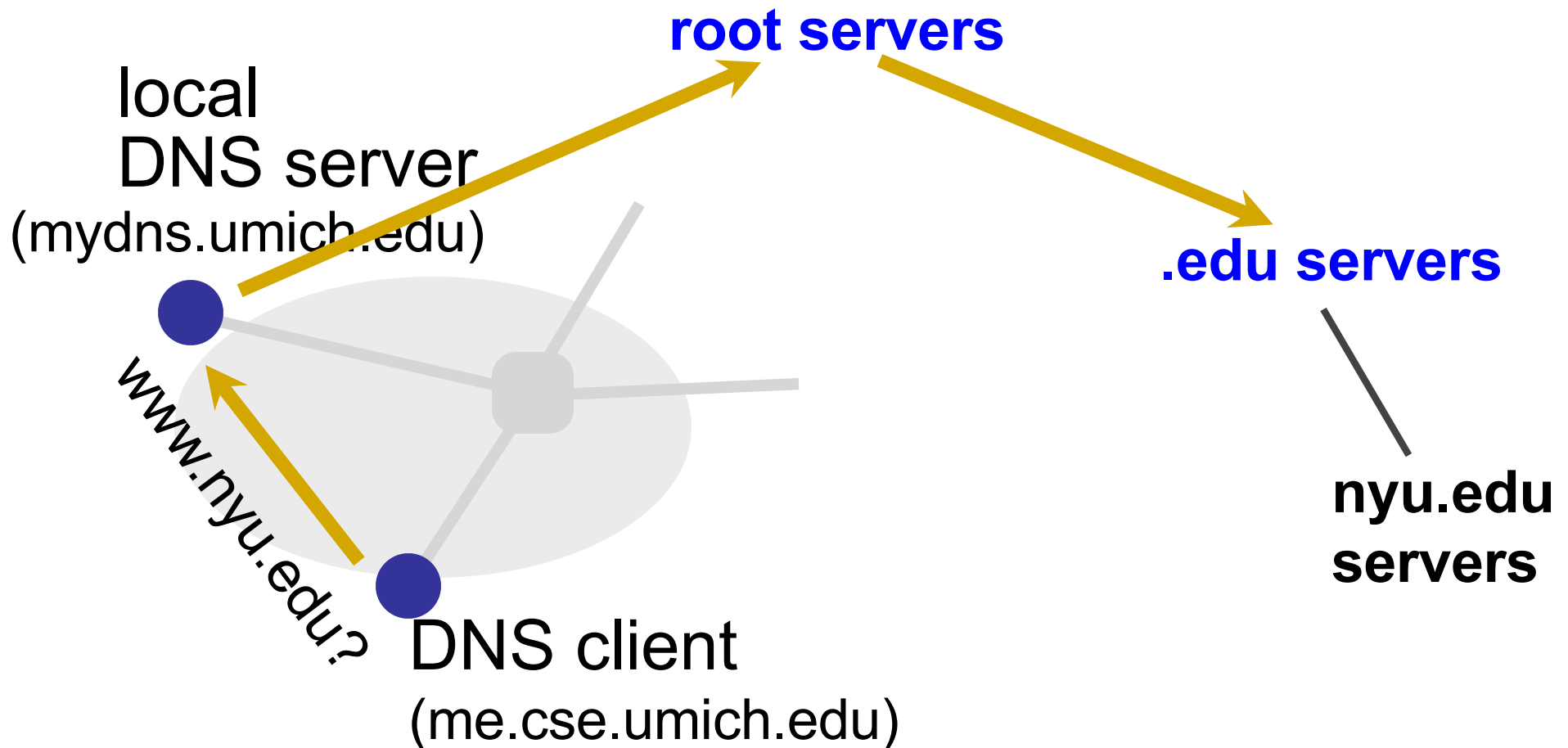
---



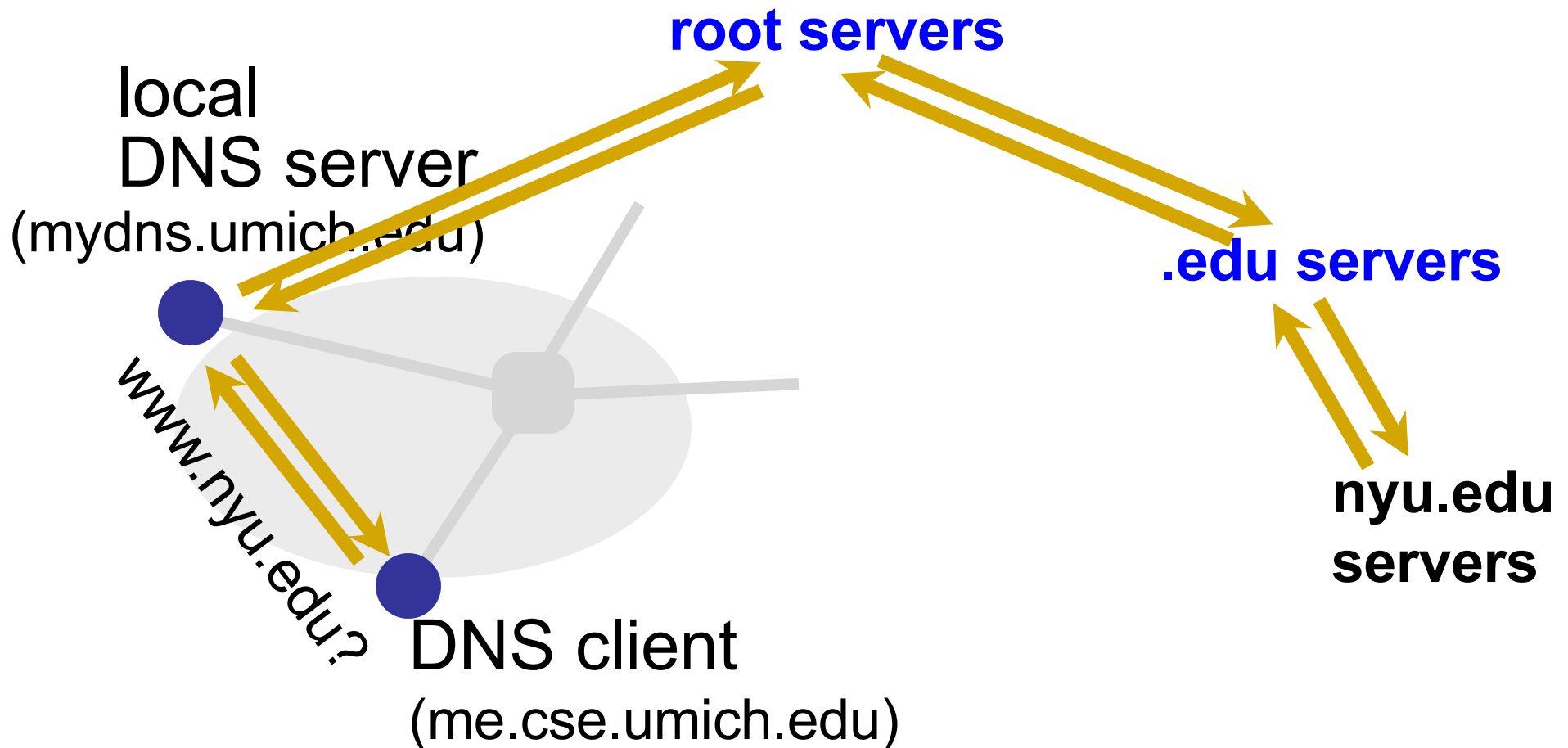


# Name resolution

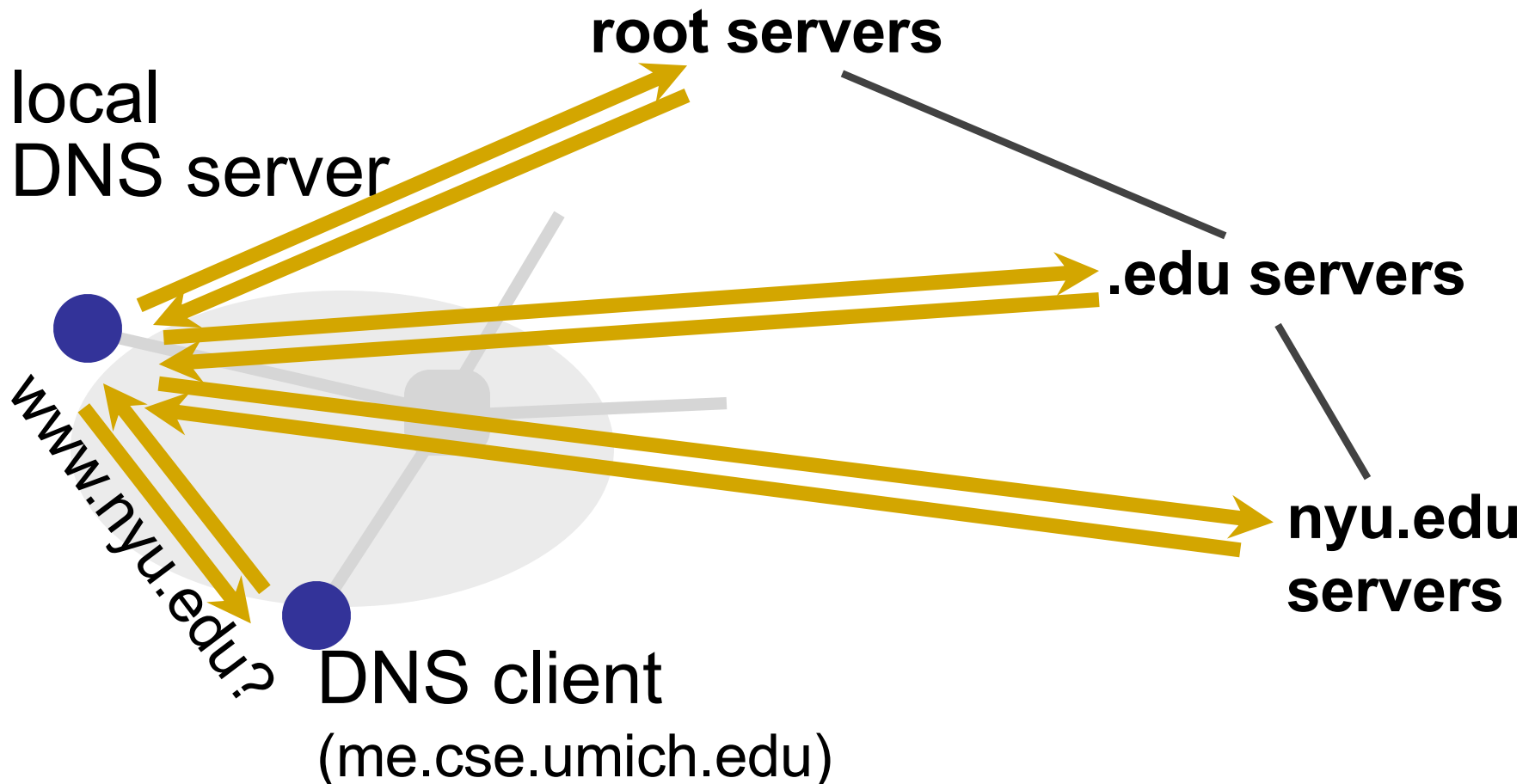
---



# Name resolution: Recursive



# Name resolution: Iterative



# DNS caching

---

- Performing all these queries takes time
  - Up to 1-second latency before starting download
- Caching can greatly reduce overhead
  - The top-level servers very rarely change
  - Popular sites (e.g., [www.google.com](http://www.google.com)) visited often
  - Local DNS server often has the information cached
- How DNS caching works
  - DNS servers cache responses to queries
  - Responses include a “time to live” (TTL) field
  - Server deletes cached entry after TTL expires

# HTTP streaming

---

- Video is stored at an HTTP server with a URL
- Clients send a GET request for the URL
- Server sends the video file as a stream
- Client first buffers for a while to minimize interruptions later
- Once the buffer reaches a threshold
  - The video plays in the foreground
  - More frames are downloaded in the background

# DASH : Dynamic Adaptive Streaming over HTTP

---

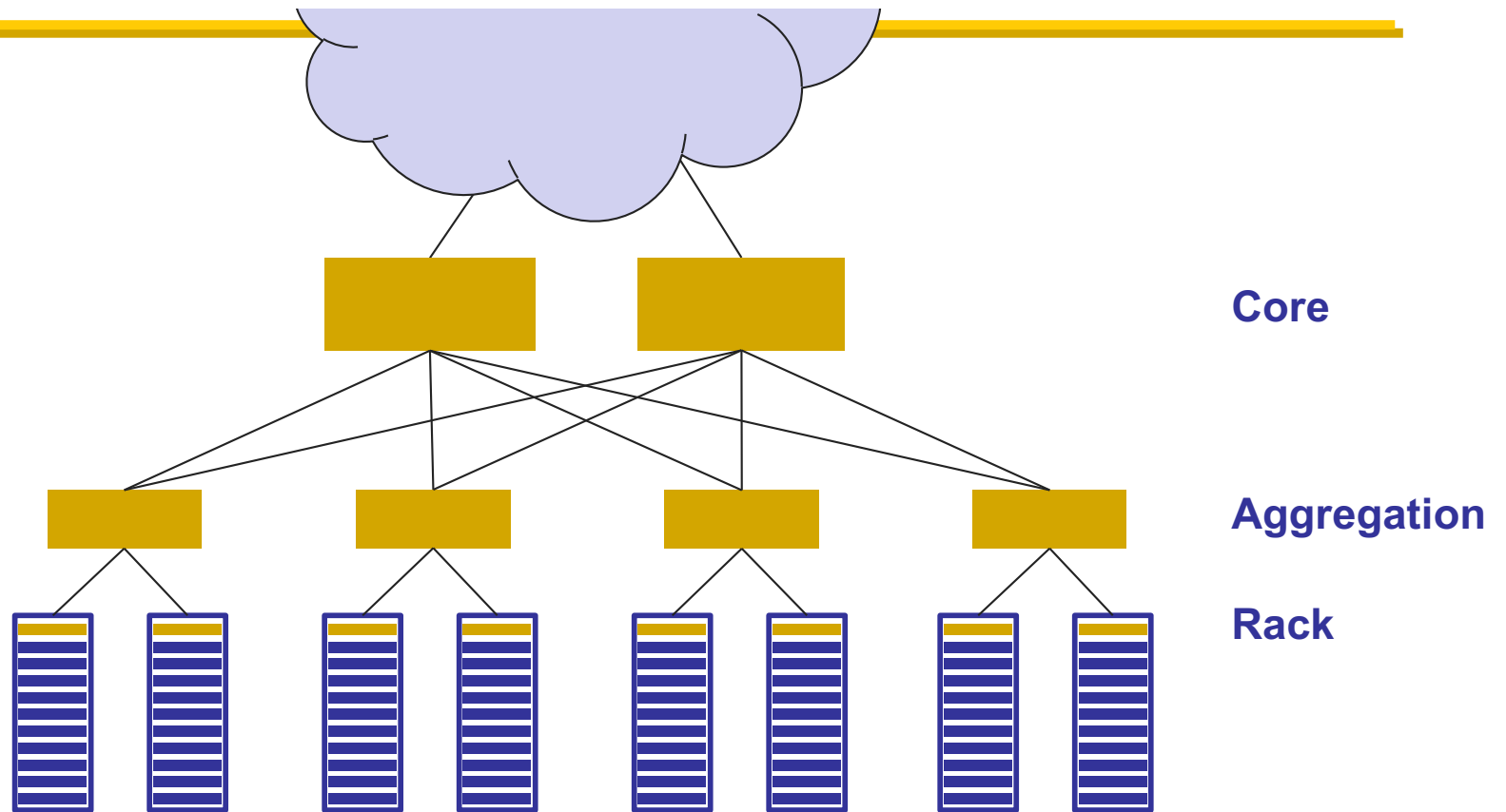
- Keep multiple resolutions of the same video
  - Stored in a manifest file in the HTTP server
- Client asks for the manifest file first to learn about the options
- Asks for chunks at a time and measures available bandwidth while they are downloaded
  - Low bandwidth  $\Rightarrow$  switch to lower bitrate
  - High bandwidth  $\Rightarrow$  switch to higher bitrate

# Applications

---

- Common theme: **parallelism**
  - Applications decomposed into tasks
  - Running in parallel on different machines
- Two common paradigms
  - Partition-Aggregate
  - Map-Reduce

# Traditional datacenter networks



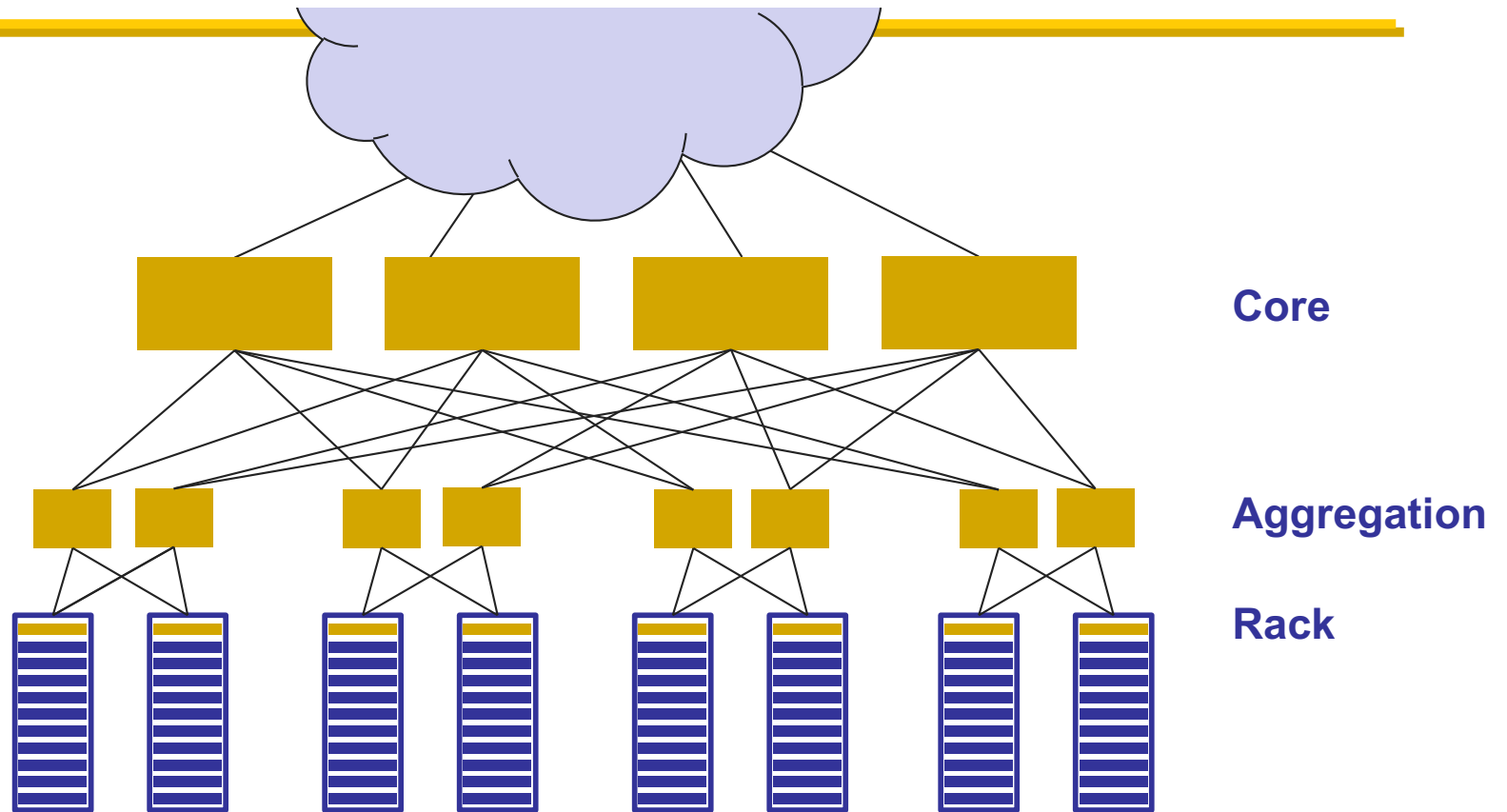


# Challenges

---

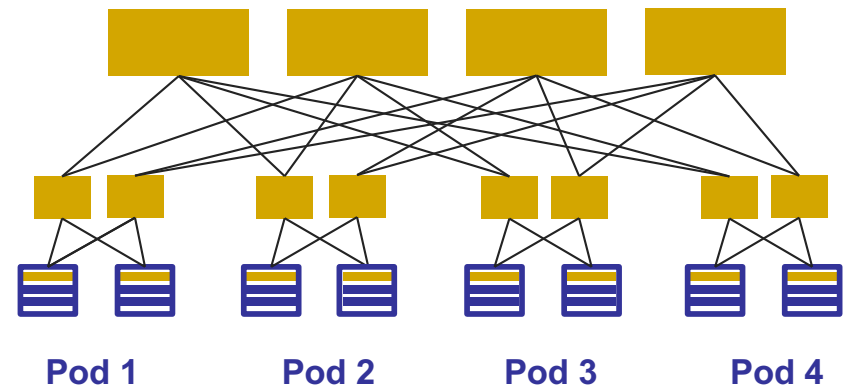
- Not enough bandwidth
  - **Oversubscription**: Less bandwidth in the ToR-Agg links than all the servers' bandwidth in the rack
  - **Oversubscription ratio**: Ratio between bandwidth underneath and bandwidth above
- Not enough paths between server pairs
  - Load balancing issues
  - Failure recovery issues

# Modern datacenter networks: More bandwidth, more paths



# 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



---

**5-MINUTE BREAK!**

# Announcements

---

- 80-minute midterm exam starts on
  - **Feb 22 Wednesday 9am – 10:20am**
  - Students receiving accommodations should have received an email confirmation by now. (Reach out to [eeecs489staff-w23@umich.edu](mailto:eeecs489staff-w23@umich.edu) if you haven't).

# Topics

---

- Basics (lectures 1–2)
- Application layer (lectures 3–5)
  - HTTP, DNS, CDN, Video Streaming, and Cloud
- Transport layer (lectures 6–9)
  - UDP vs. TCP
  - TCP details: reliability and flow control
  - TCP congestion control: general concepts only
- Network layer (lecture 10–11)
  - Overview
  - Data plane

# Role of the transport layer

---

- (1) Communication between application processes
  - Mux and demux from/to application processes
  - Implemented using ports
- (2) Provide common end-to-end services for app layer
  - Reliable, in-order data delivery
  - Well-paced data delivery

# UDP vs. TCP

---

- Both UDP and TCP perform mux/demux via ports

|                         | UDP                      | TCP   |
|-------------------------|--------------------------|---|
| <b>Data abstraction</b> | Packets (datagrams)      | Stream of bytes of arbitrary length   |
| <b>Service</b>          | Best-effort (same as IP) | <ul style="list-style-type: none"><li>•Reliability</li><li>•In-order delivery</li><li>•Congestion control</li><li>•Flow control</li></ul> |



# Reliable transport: General concepts

---

- Checksums (for error detection)
- Timers (for loss detection)
- Acknowledgments (feedback from receiver)
  - Cumulative: “received everything up to X”
  - Selective: “received X”
- Sequence no (detect duplicates, accounting)
- Sliding windows (for efficiency)

You should know:

- what these concepts are
- why they exist
- how TCP uses them

# Designing a reliable transport protocol

---

- **Stop and wait** is correct but inefficient
  - Works packet by packet (of size DATA)
  - Throughput is  $(\text{DATA} / \text{RTT})$
- **Sliding window**: use pipelining to increase throughput
  - $n$  packets at a time results in higher throughput
  - $\text{MIN}(n * \text{DATA} / \text{RTT}, \text{Link Bandwidth})$

# The TCP abstraction

---

- TCP delivers a reliable, in-order, byte stream
- **Reliable**: TCP resends lost packets (recursively)
  - Until it gives up and shuts down connection
- **In-order**: TCP only hands consecutive chunks of data to application
- **Byte stream**: TCP assumes there is an incoming stream of data, and attempts to deliver it to app

# Things to know about TCP

---

- How TCP achieves reliability
  - RTT estimation
  - Connection establishment/teardown
  - Flow Control
  - Congestion Control (concepts only)
- 
- For each, know how the functionality is implemented and why it is needed

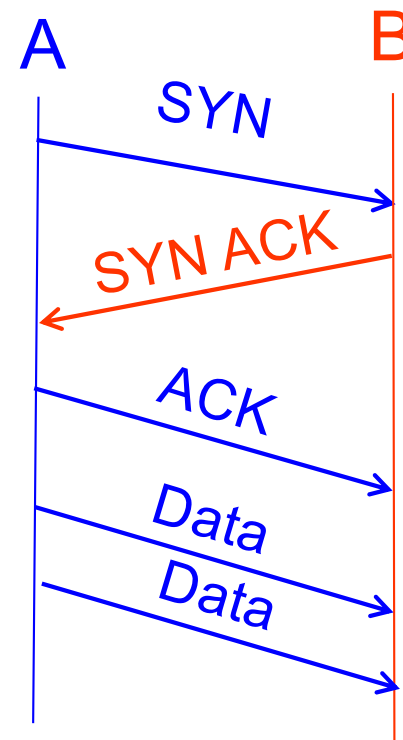
# Reliability

---

- Having TCP take care of it simplifies application development
- How
  - Checksums and timers (for error and loss detection)
  - Fast retransmit (to detect faster-than-timeout loss)
  - Cumulative ACKs (receiver feedback: what's lost?)
  - Sliding windows (for efficiency)
  - Buffers at sender (hold packets until ACKs arrive)
  - Buffers at receiver (to reorder packets before delivery to application)

# Establishing/terminating a TCP connection

- Three-way handshake to establish connection
  - Host A sends a SYN (open; “synchronize sequence numbers”) to host B
  - Host B returns a SYN acknowledgment (SYN ACK)
  - Host A sends an ACK to acknowledge the SYN ACK
- Three-way handshake to terminate (normal operation)



# Flow control

---

- Why?

- TCP at the receiver must buffer a packet until all packets before it (in byte-order) have arrived and the receiving application has consumed available bytes
- Hence, receiver advances its window when the receiving application consumes data
- Sender advances its window when new data ACK'd
- Risk of sender overrunning the receiver's buffers

- How?

- “Advertised Window” field in TCP header

# Congestion control

---

- Why?
  - Because the network itself can be the bottleneck
  - Should make efficient use of available network capacity
    - » While sharing available capacity fairly with other flows
    - » And adapting to changes in available capacity
- How?
  - Dynamically adapts the size of the sending window



# Put together

---

- Flow Control
  - Restrict window to RWND to make sure that the receiver isn't overwhelmed
- Congestion Control
  - Restrict window to CWND to make sure that the network isn't overwhelmed
- Together
  - Restrict window to  $\min\{\text{RWND}, \text{CWND}\}$  to make sure that neither the receiver nor the network are overwhelmed

# CC implementation

---

- States at sender
  - **CWND** (initialized to a small constant)
  - **ssthresh** (initialized to a large constant)
  - **dupACKcount** and **timer**
- Events
  - **ACK** (new data)
  - **dupACK** (duplicate ACK for old data)
  - **Timeout**

# Event: ACK (new data)

---

- If  $CWND < ssthresh$

- $CWND += 1$

- *$CWND$  packets per RTT*
  - *Hence, after one RTT with no drops:*  
 $CWND = 2 \times CWND$

# Event: ACK (new data)

- If  $CWND < ssthresh$

- $CWND += 1$

***Slow start phase***

- Else

- $CWND = CWND + 1/CWND$

***Congestion avoidance phase***

- *CWND packets per RTT*
- *Hence, after one RTT with no drops:*  
 $CWND = CWND + 1$

# Event: TimeOut

---

- On Timeout
  - $\text{ssthresh} \leftarrow \text{CWND}/2$
  - $\text{CWND} \leftarrow 1$

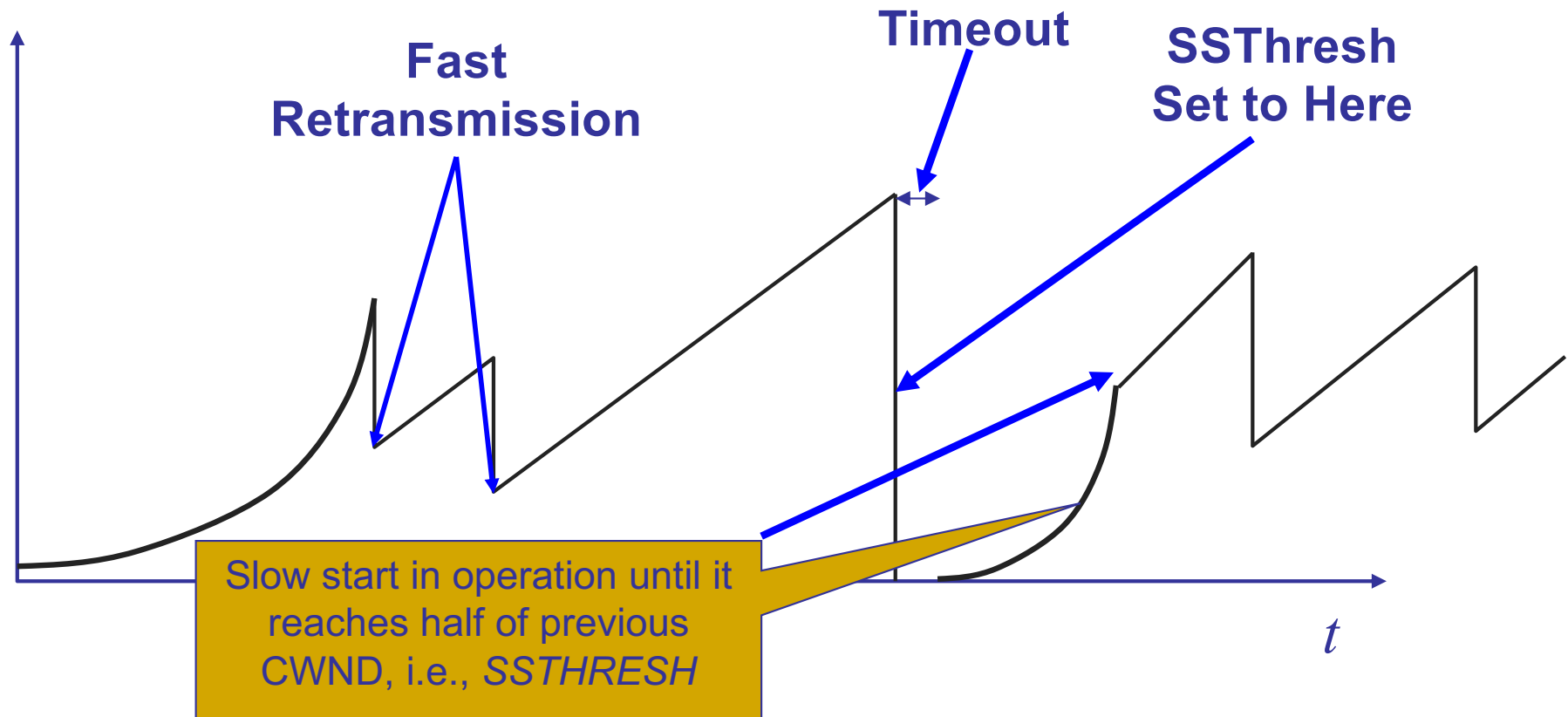
# Event: dupACK

---

- dupACKcount ++
- If dupACKcount = 3 /\* fast retransmit \*/
  - ssthresh = CWND/2
  - CWND = CWND/2

# Example

*Window*



Slow-start restart: Go back to  $\text{CWND} = 1 \text{ MSS}$ , but take advantage of knowing the previous value of  $\text{CWND}$

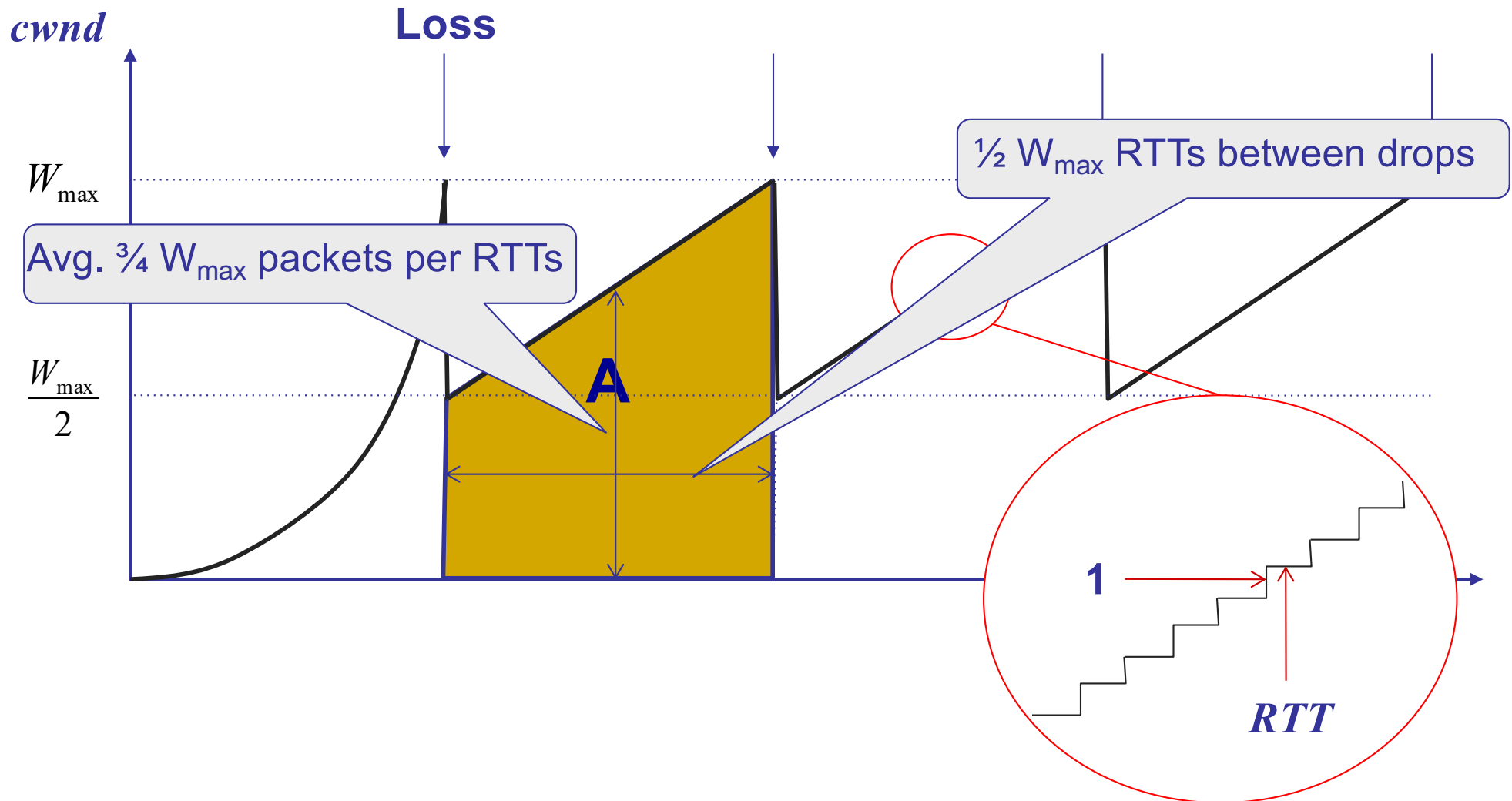
# TCP flavors

---

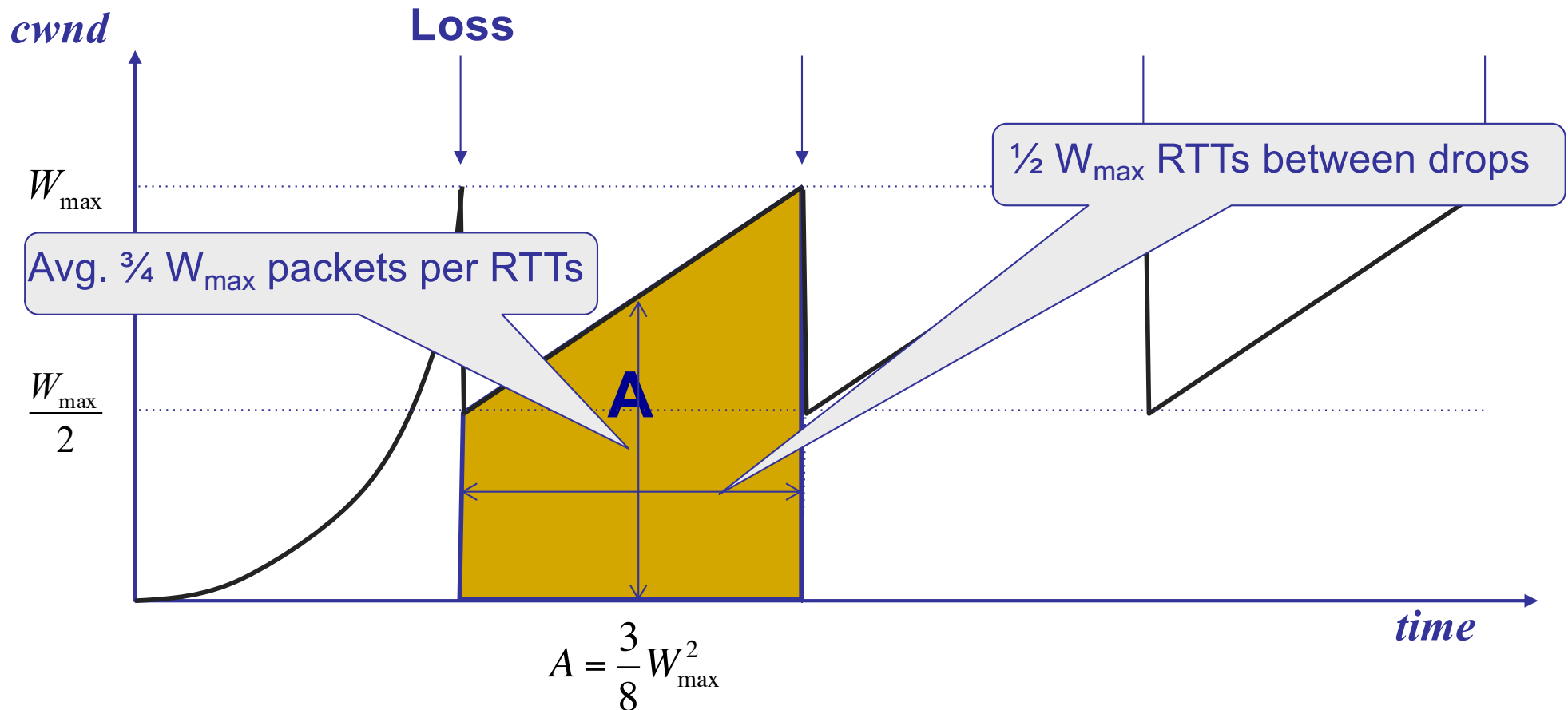
- TCP-Tahoe
  - CWND = 1 on 3 dupACKs
- TCP-Reno
  - CWND = 1 on timeout
  - CWND = CWND/2 on 3 dupACKs
- TCP-newReno
  - TCP-Reno + improved fast recovery
- TCP-SACK
  - Incorporates selective acknowledgements



# A simple model for TCP throughput



# A simple model for TCP throughput



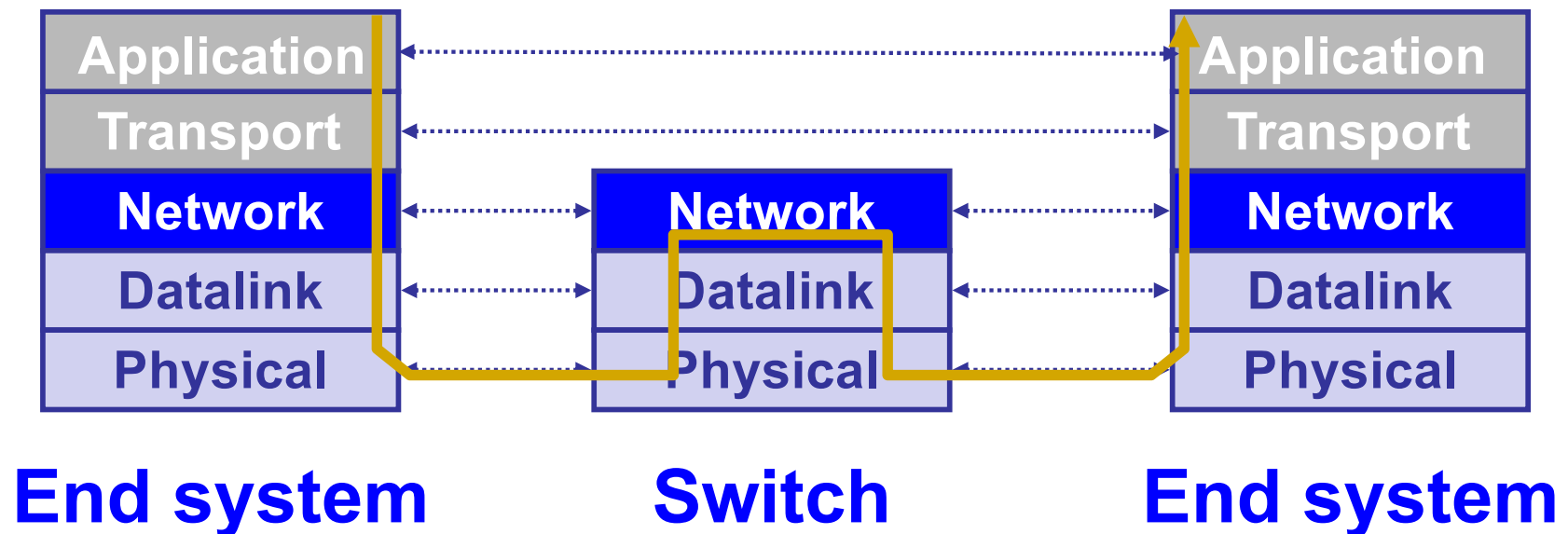
# Topics

---

- Basics (lectures 1–2)
- Application layer (lectures 3–5)
  - HTTP, DNS, CDN, Video Streaming, and Cloud
- Transport layer (lectures 6–9)
  - UDP vs. TCP
  - TCP details: reliability and flow control
  - TCP congestion control: general concepts only
- Network layer (lecture 10–11)
  - Overview
  - Data plane

# Network layer

- Present everywhere
- Performs **addressing**, **forwarding**, and **routing**, among other tasks

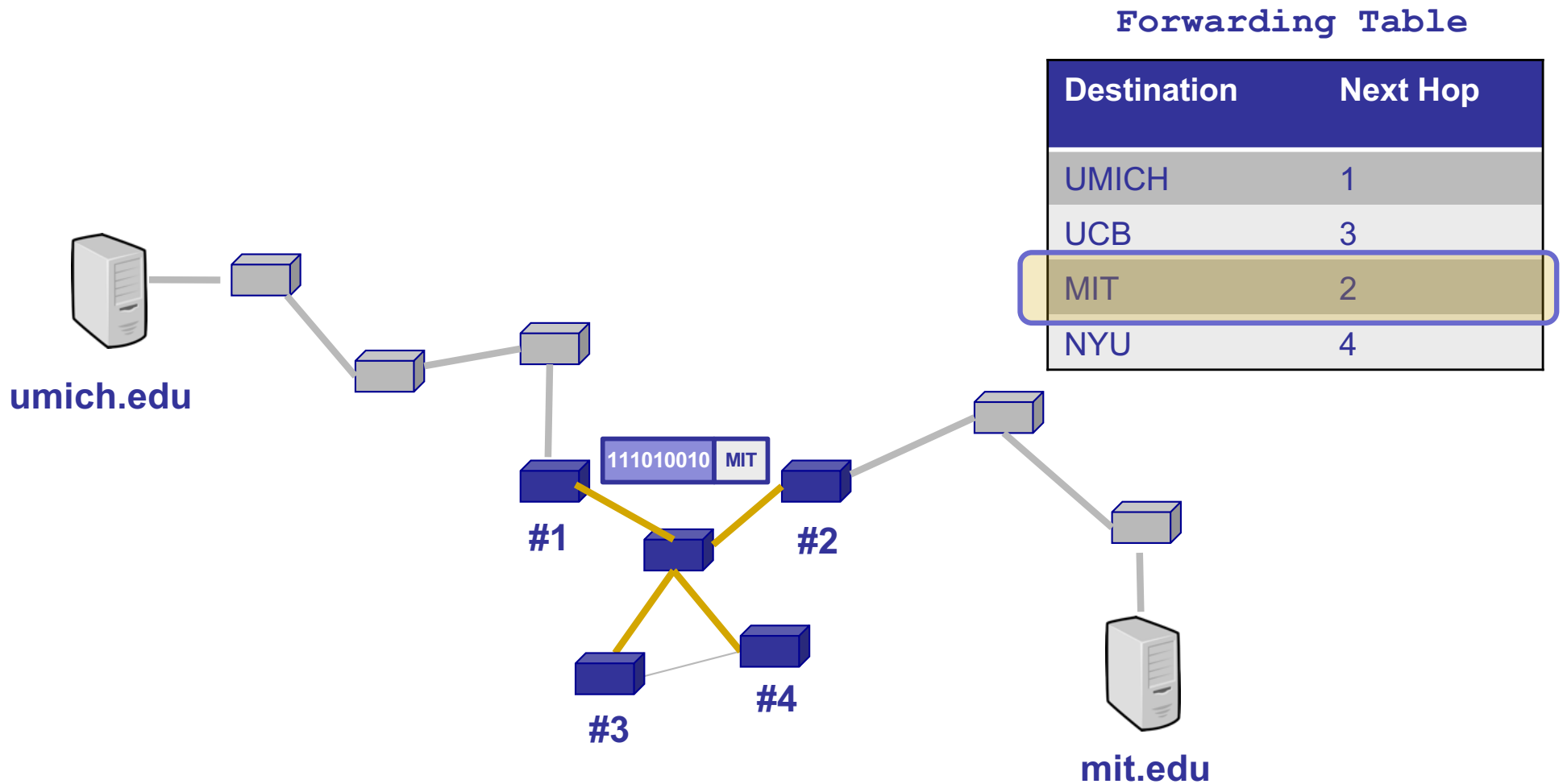


# Forwarding vs. routing

---

- Forwarding: “data plane”
  - Directing one data packet
  - Each router using local routing state
- Routing: “control plane”
  - Computing the forwarding tables that guide packets
  - Jointly computed by routers using a distributed algorithm

# Forwarding



# Designing the IP header

---

- Think of the IP header as an interface
  - Between the source and destination end-systems
  - Between the source and network (routers)
- Designing an interface
  - What task(s) are we trying to accomplish?
  - What information is needed to do it?
- Header reflects information needed for basic tasks

# What information do we need?

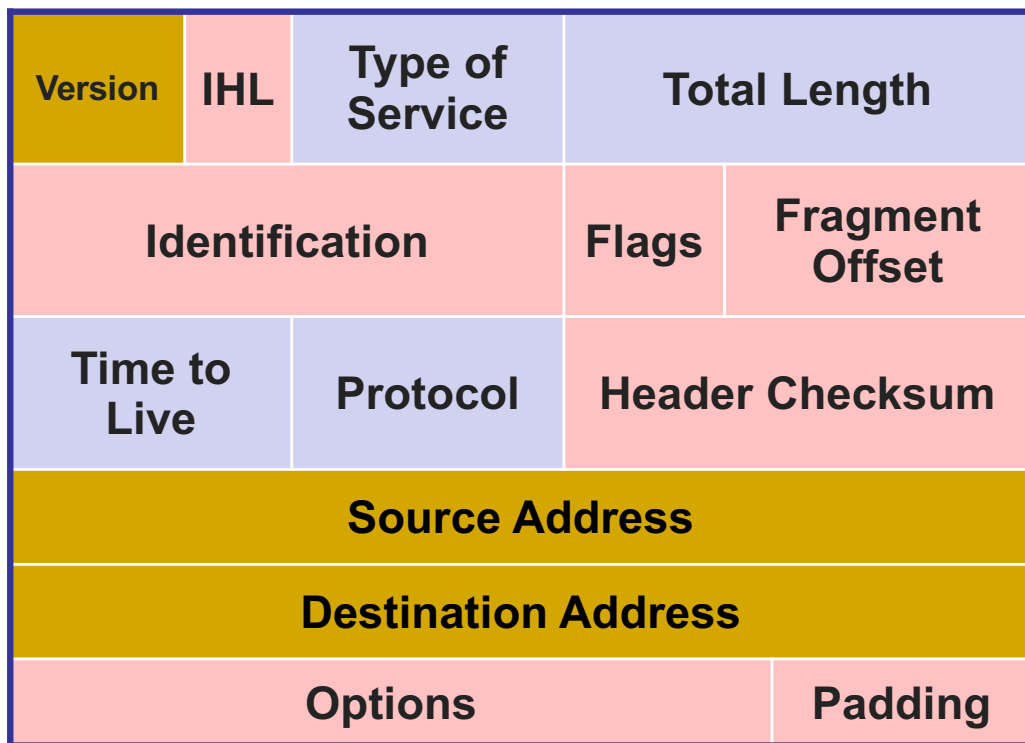
---

- Parse packet
  - IP version number (4 bits), packet length (16 bits)
- Carry packet to the destination
  - Destination's IP address (32 bits)
- Deal with problems along the way
  - Loops: TTL (8 bits)
  - Corruption: checksum (16 bits)
  - Packet too large: fragmentation fields (32 bits)

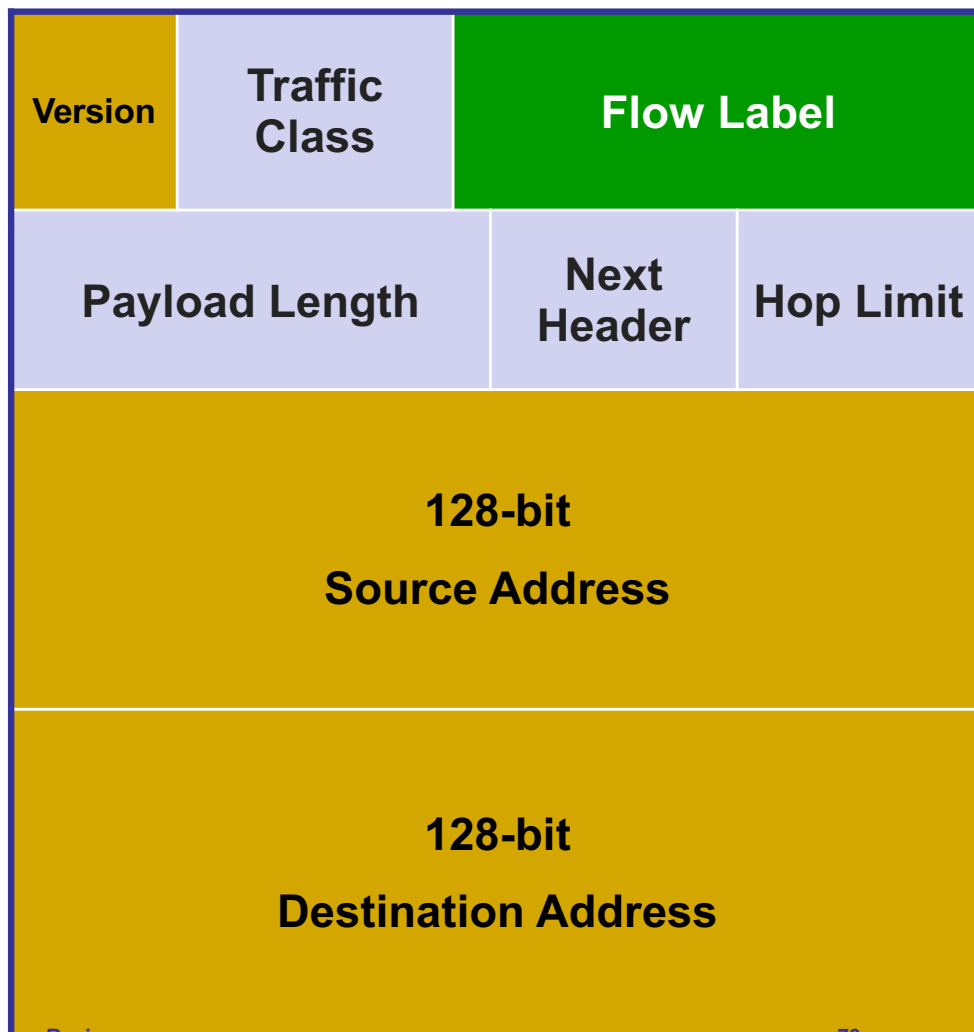


# IPv4 and IPv6 header comparison

IPv4



IPv6



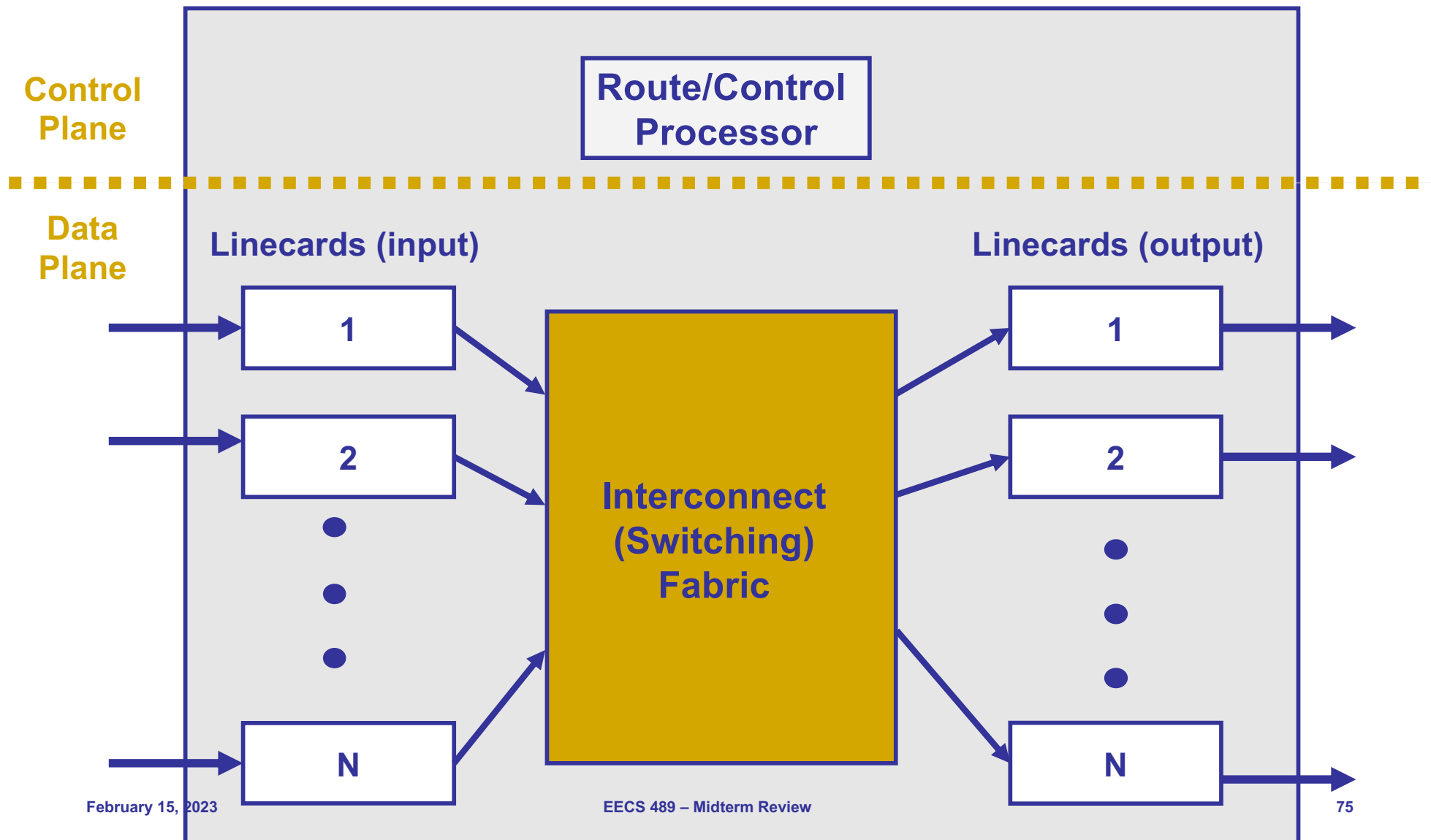
- Field name kept from IPv4 to IPv6
- Fields not kept in IPv6
- Name & position changed in IPv6
- New field in IPv6

# Philosophy of changes

---

- Don't deal with problems: leave to ends
  - Eliminated fragmentation and checksum
  - Why retain TTL?
- Simplify handling:
  - New options mechanism (uses next header)
  - Eliminated header length
    - » Why couldn't IPv4 do this?
- Provide general flow label for packet
  - Not tied to semantics
  - Provides great flexibility

# What's inside a router?



# Input linecards

---

- Main challenge is processing speeds
- Tasks involved:
  - Update packet header (easy)
  - LPM lookup on destination address (harder)
- Mostly implemented with specialized hardware

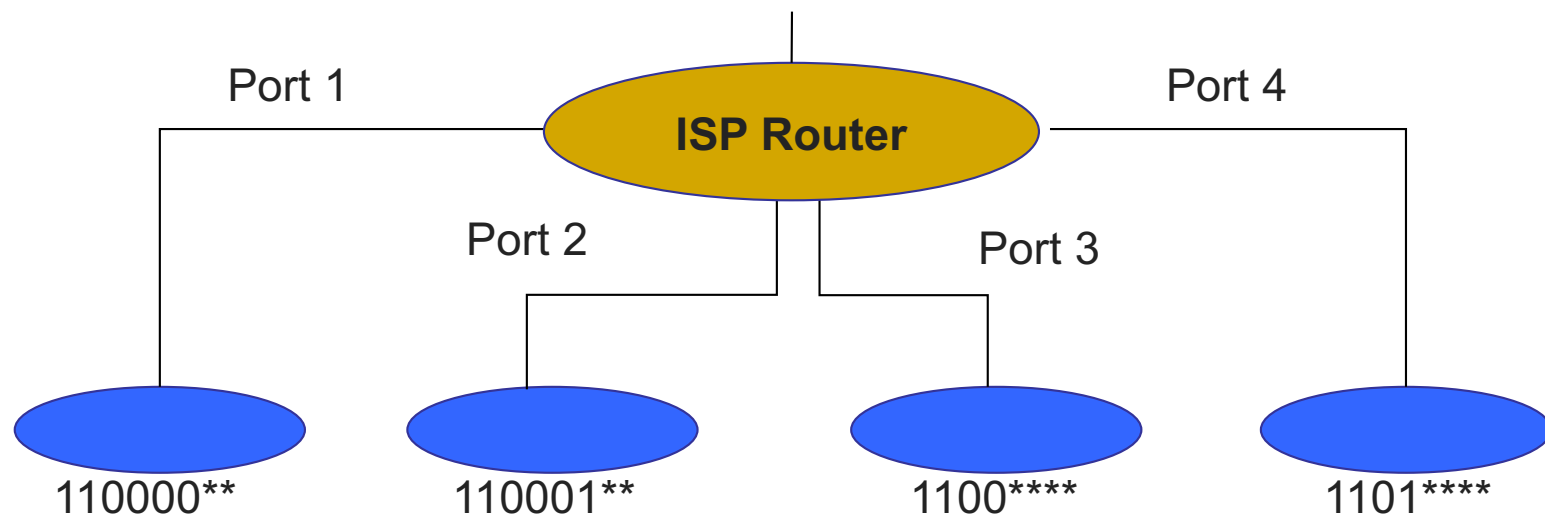
# Looking up the output port

---

- One entry for each address → 4 billion entries!
- For scalability, addresses are aggregated

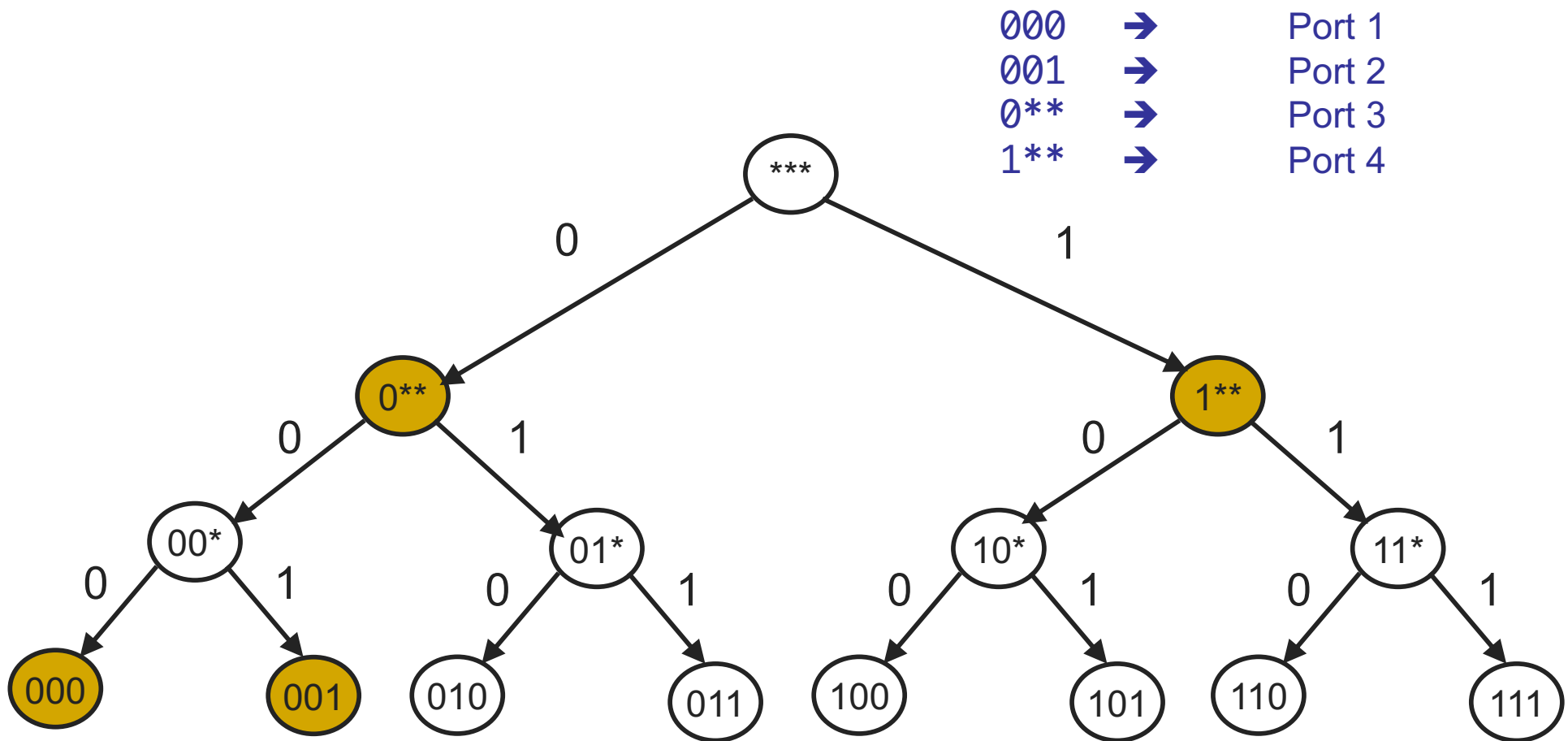
# Longest prefix matching

---



Send to the port with the longest prefix match

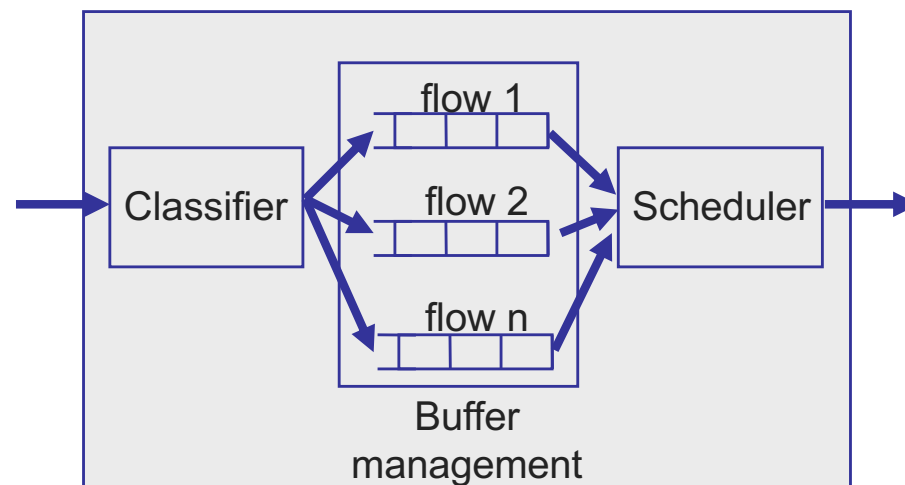
# Tree structure



Record port associated with latest match, and only override when it matches another prefix during walk down tree

# Output linecards

- **Packet classification**: map packets to flows
- **Buffer management**: decide when and which packet to drop
- **Scheduler**: decide when and which packet to transmit

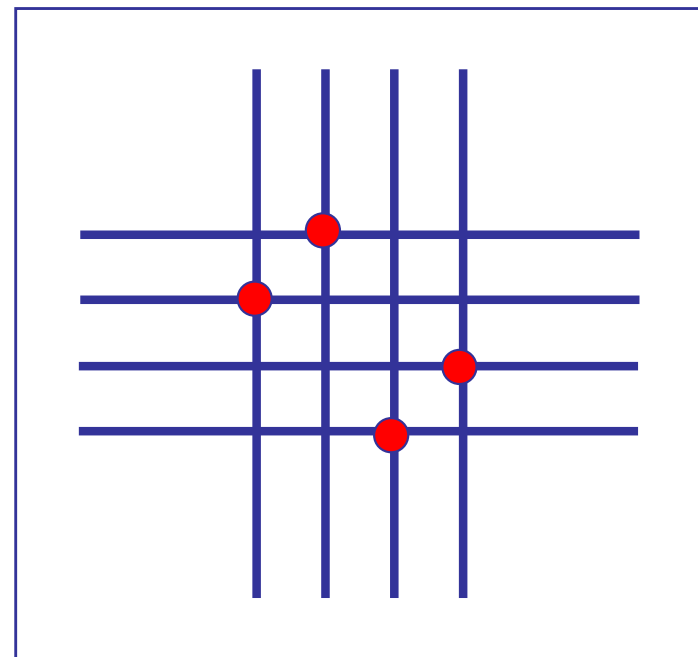




# Crossbar interconnect

- $2N$  buses intersecting with each other:
  - $N$  input
  - $N$  output
- Non-blocking

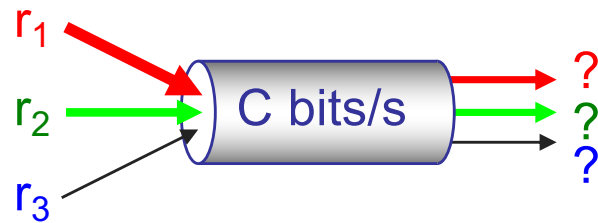
Input  
ports



Output ports

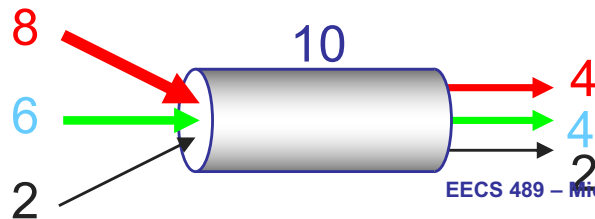
# Max-Min fairness

- Given set of bandwidth demands  $r_i$  and total bandwidth  $C$ , max-min bandwidth allocations are:
  - $a_i = \min(f, r_i)$
  - where  $f$  is the unique value such that  $\text{Sum}(a_i) = C$



# Example

- $C = 10$ ;  $r_1 = 8$ ,  $r_2 = 6$ ,  $r_3 = 2$ ;  $N = 3$
- $C/3 = 3.33 \rightarrow$ 
  - $r_3$  needs only 2
    - » Can service all of  $r_3$
  - Remove  $r_3$  from the accounting:  $C = C - r_3 = 8$ ;  $N = 2$
- $C/2 = 4 \rightarrow$ 
  - Can't service all of  $r_1$  or  $r_2$
  - So hold them to the remaining fair share:  $f = 4$



$f = 4$ :  
 $\min(8, 4) = 4$   
 $\min(6, 4) = 4$   
 $\min(2, 4) = 2$

# Max-Min fairness

---

- Given set of bandwidth demands  $r_i$  and total bandwidth  $C$ , max-min bandwidth allocations are:
  - $a_i = \min(f, r_i)$
  - where  $f$  is the unique value such that  $\text{Sum}(a_i) = C$
- If you don't get full demand, no one gets more than you
- This is what round-robin service gives if all packets are the same size

# Summary

---

- Demo Exam on Canvas