CS168
How the Internet Works (contd.)

Sylvia Ratnasamy

Fall 2024

Today

- Wrap up our discussion of circuit & packet switching
- Start our top-down overview

Recall, from last lecture...

Two canonical approaches to sharing

- **Reservations**: end-hosts explicitly reserve BW when needed (e.g., at the start of a flow)
- **Best-effort**: just send data packets when you have them and hope for the best ...

Two canonical designs to implementing these approaches

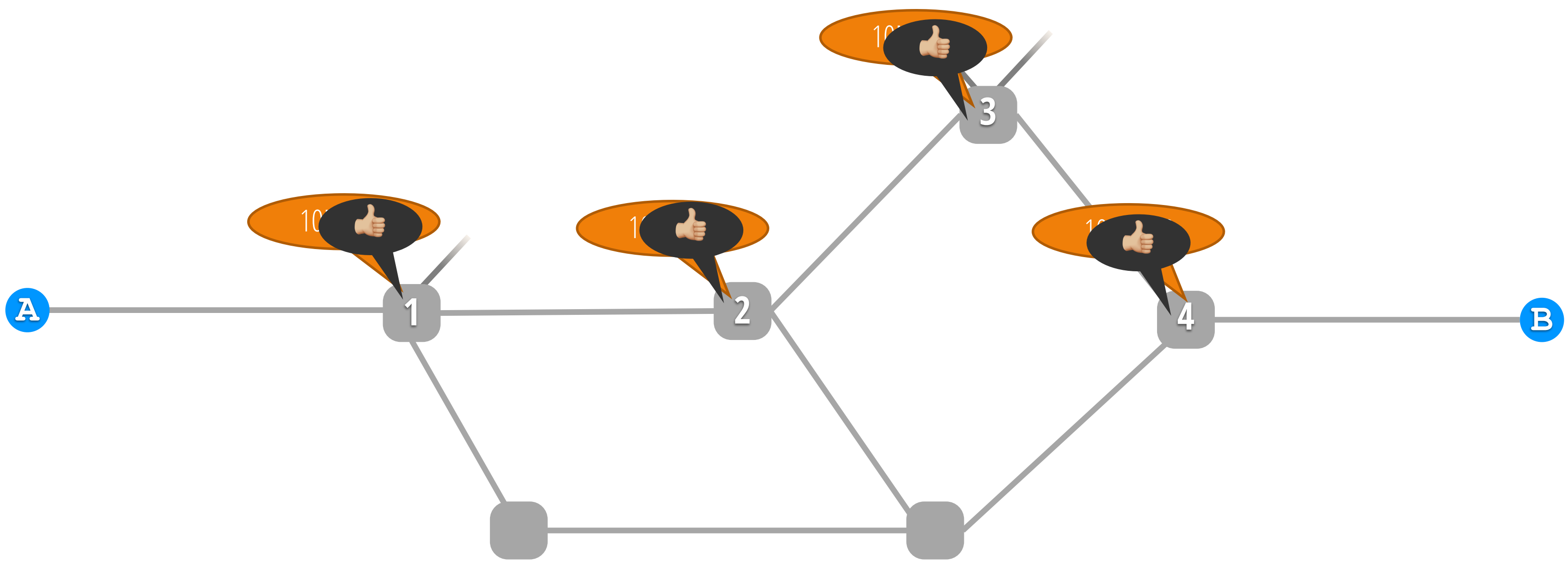
- Reservations via **circuit switching**
- Best-effort via **packet switching**

Circuit *vs.* Packet switching: which is better?

- What are the dimensions along which we should compare?
  - As an abstraction to applications (endhosts)
  - Efficiency
  - Handling failures (at scale)
  - Complexity of implementation (at scale)

Recall...

(1) source sends a reservation request to the destination



How do switches know that the reservation went through?

What happens if the reservation request is lost mid way?

What happens if the confirmation that the reservation made it is lost?

What should the endhost do if the reservation is declined?

What happens if the underlying route changes?

And on and on....

Recap: Circuit *vs.* Packet Switching

- Pros for circuit switching:
  - Better application performance (reserved bandwidth)
  - More predictable and understandable (w/o failures)
- Pros for packet switching:
  - Better efficiency
  - Faster startup to first packet delivered
  - Easier recovery from failure
  - Simpler implementation (avoids dynamic per-flow state management in switches)

What does the Internet use today?

- Packet switching is the default
- Limited use of RSVP (“Resource Reservation Protocol”)
  - Reservation is typically limited to a single domain
  - Not exposed to users; only invoked under operator control
  - Reservations for large aggregates of flows (*vs*. individual flows)
- You *can* also buy dedicated bandwidth along a path (e.g., “MPLS circuit”)
  - Often used by enterprises from one branch location to another (or to/from cloud)
  - Very expensive (e.g., 10-20x higher than a default connection)
  - Often statically set up (manually), long-lived (e.g., years), and per user (*vs.* per flow)
  - So, a far cry from the vision of dynamic reservations that we just discussed

Circuit *vs.* Packet Switching: A bit of history

- The early Internet (70-80s): packet switched
  - Well suited to (bursty) file transfer applications

- The next iteration (~90s): research & industry believed we'd need circuit switching
  - Envisioned that voice/live TV/ would be the Internet's true killer app
  - Spent 10+ years trying to realize this vision

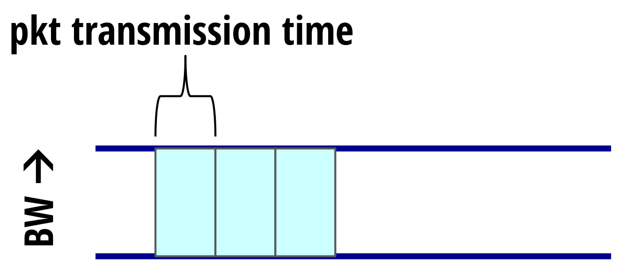
- Ultimately, a failed vision. Why?
  - All the reasons we discussed...
  - ...and Email and the web emerged as the killer apps of that time
  - ...and people rewrote apps to be adaptive (turns out we didn't really need guaranteed BW!)

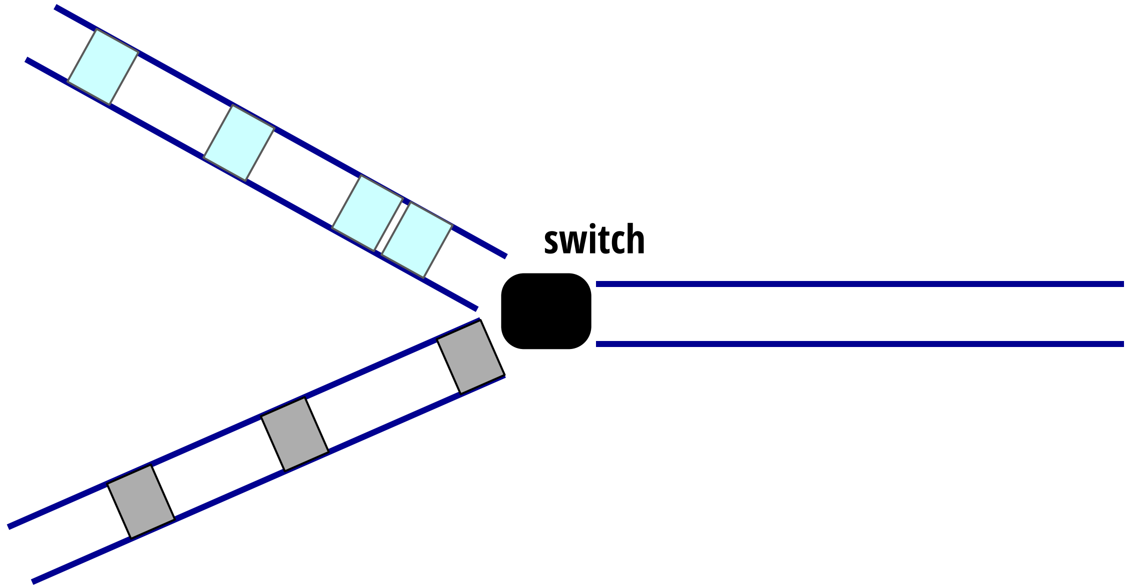
A lesson in how technology can transform user behavior!

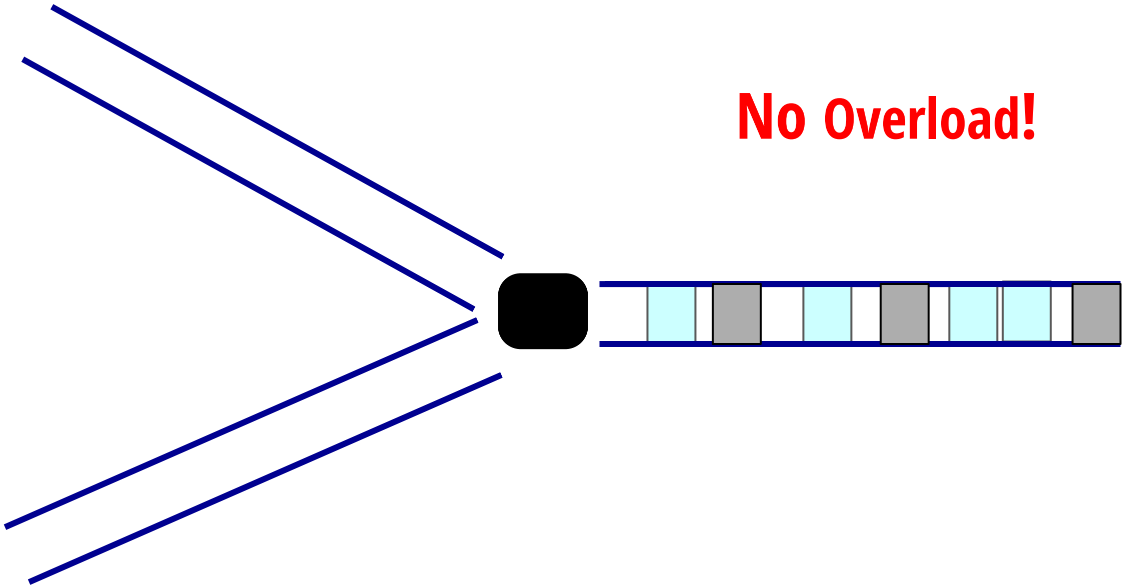
Questions??

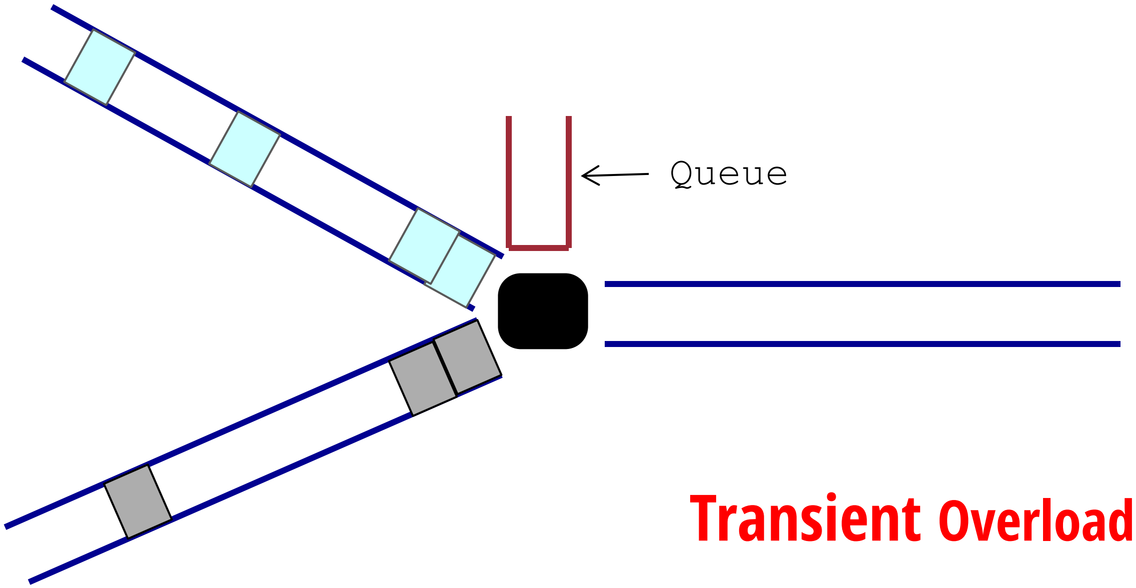
Let’s take a closer look at packet switching ….

Recall, packets in flight: "pipe" view



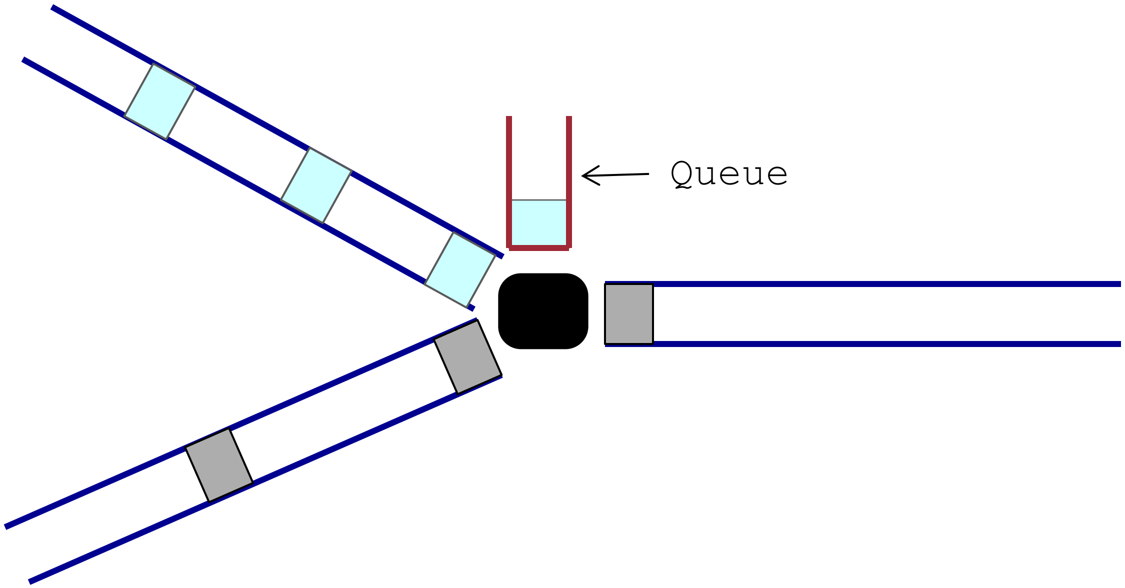

switch

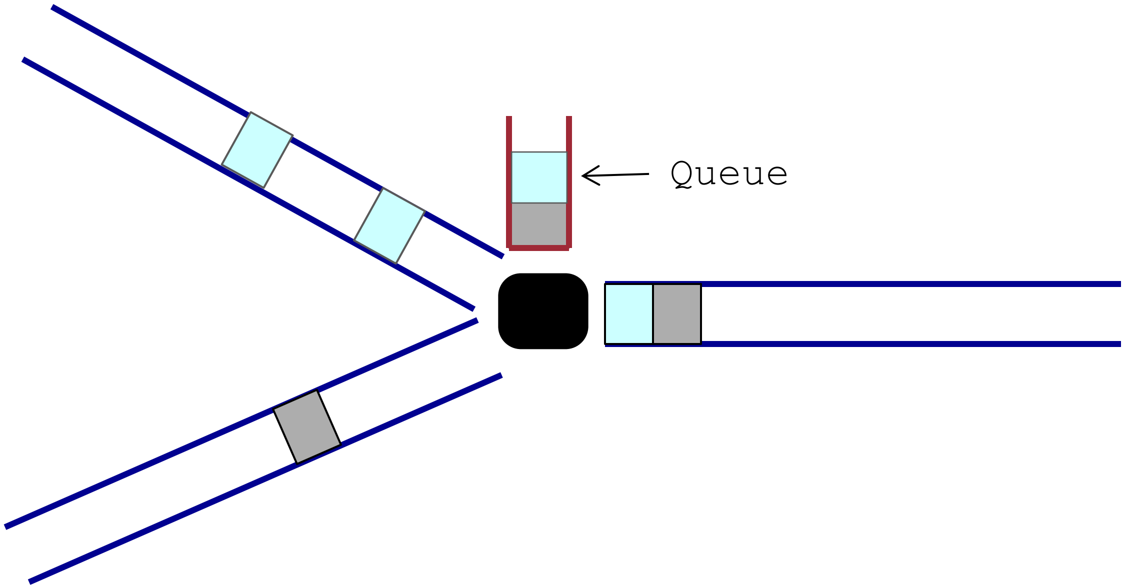

No Overload!

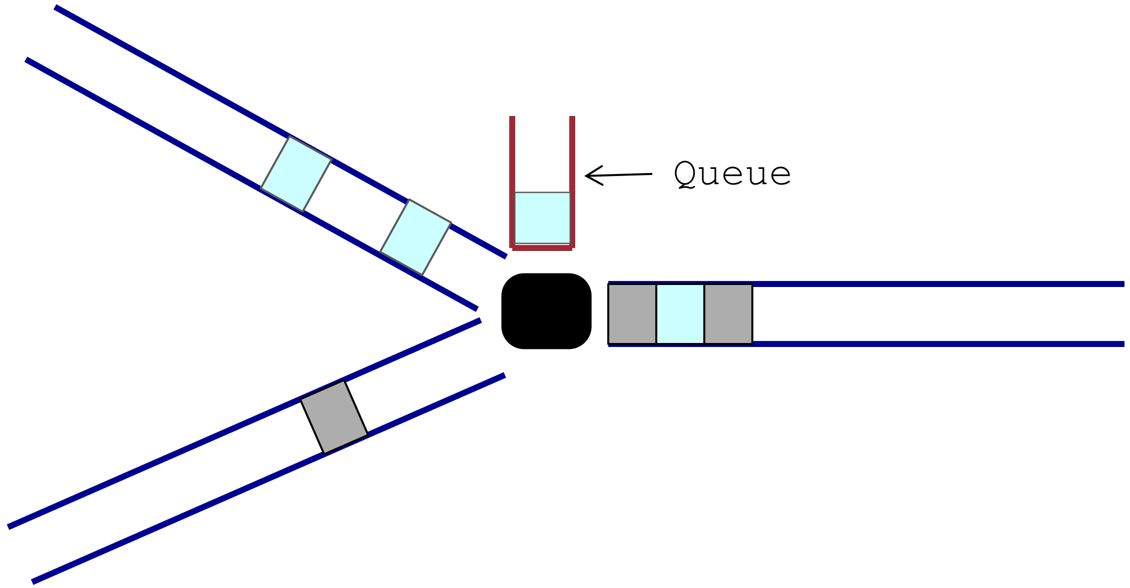


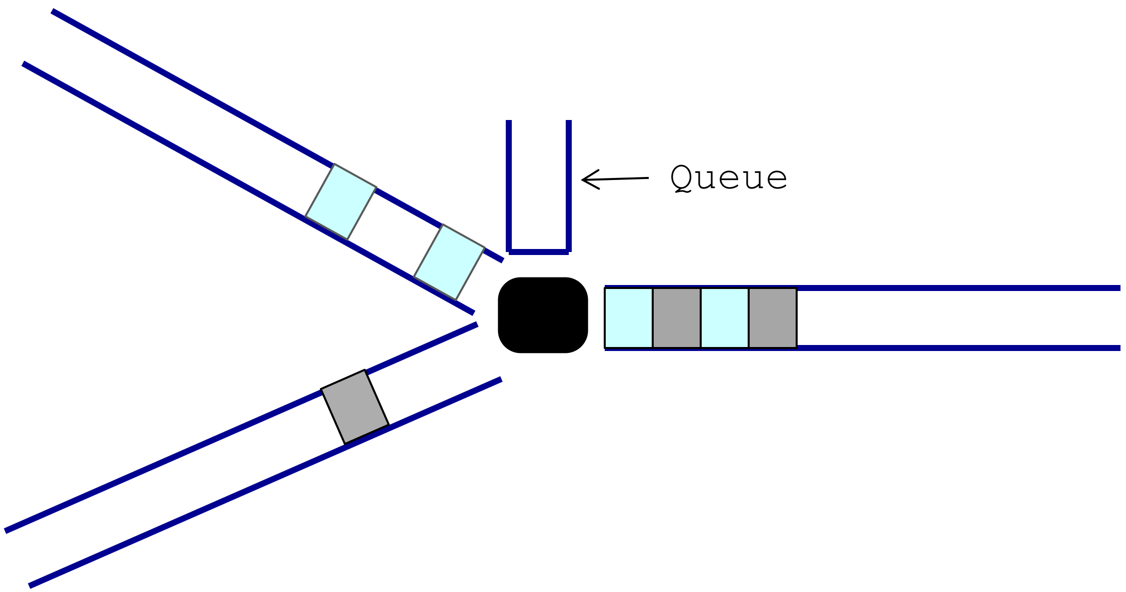
Transient Overload

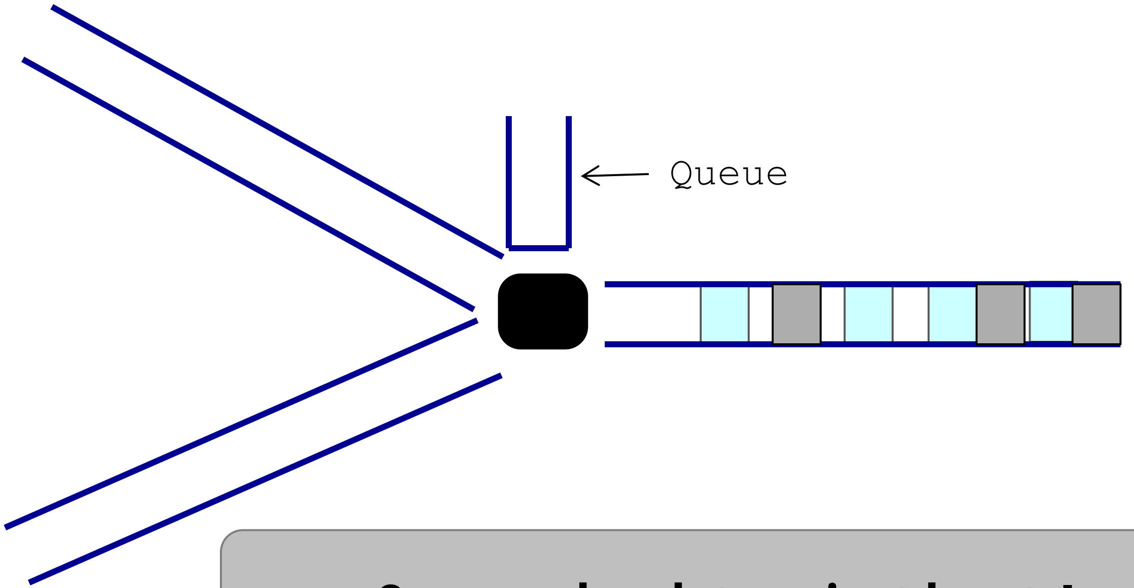
Not a rare event!


Queue

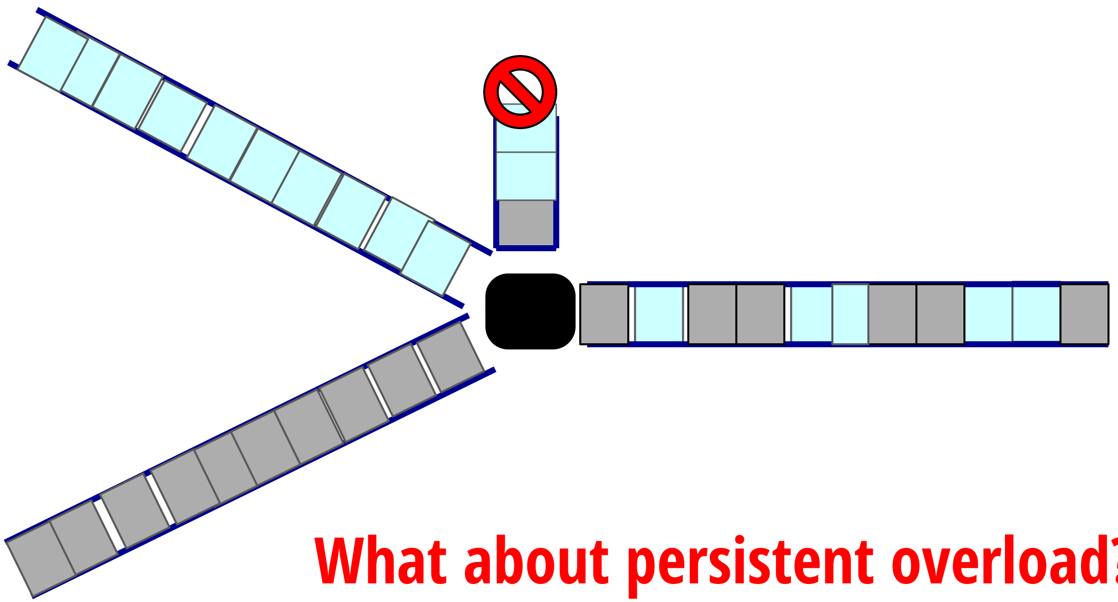

Queue


Queue


Queue



Queues absorb transient bursts!



What about persistent overload?

Will eventually drop packets

Queues introduce queuing delays

- Recall, packet delay = transmission delay + propagation delay
- With queues: packet delay = transmission delay + propagation delay + queueing delay

Recall: life of a packet so far...

- Source has some data to send to a destination
- Chunks it up into packets: each packet has a payload and a header
- Packet travels along a link
- Arrives at a switch; switch forwards the packet to its next hop
- And the last step repeats until we reach the destination ...

Recall: life of a packet so far… [updated]

- Source has some data to send to a destination
- Chunks it up into packets: each packet has a payload and a header
- Packet travels along a link
- Arrives at a switch; switch forwards the packet to its next hop
  - switch may buffer, or even drop, the packet
- And the last step repeats until we reach the destination …
  - or the packet is dropped

Challenge: Reliable packet delivery

- Packets can be dropped along the way
  - Buffers in switch can overflow
  - Switch can crash while buffering packets
  - Links can garble/corrupt packets

- Given an unreliable network, how do we make sure the destination receives its packets?
  - Or at least know if they are delivered….

Challenge: Congestion control

- Packet switching means network capacity is allocated on-demand
- But endhosts independently decide at what rate they will send packets!
- This can be tricky!
  - How fast I send packets impacts whether *your* packets are dropped
  - What's a good rate at which I should send my packets?
- Hence, congestion control:
  - How do we ensure that (endhosts') independent decisions lead to a good outcome?

Hence, our fundamental topics [updated]

- How do we name endhosts on the Internet? **(naming)**
- How do we address endhosts? **(addressing)**
- How do we map names to addresses? **(mapping names to addresses)**
- How do we compute forwarding tables? **(routing control plane → project 1)**
- How do we forward packets? **(routing data plane)**
- How do hosts communicate reliably? **(reliable packet delivery → project 3)**
- How do sources know at what rate they can send packets? **(congestion control)**
- **How do we build end-to-end applications? (naming, web, content delivery)**
- **How do our solutions change in modern environments (clouds, wifi, cellular)**

Recap: key takeaways from our bottom-up overview

- What is a packet?
- Approaches to sharing the network – circuit vs. packet switching -- and their tradeoffs
- An overall sense of the life of a packet
  - We'll continue to refine this picture over the course of the semester
- An overall sense of the topics we'll be studying and why they're fundamental

Questions??

Changing Perspective

- Designing the Internet: a top-down approach
- In the process, discuss a few enduring ideas:
  - Layering
  - The end-to-end principle
  - Fate sharing

The Internet's problem definition

- Support the transfer of data between endhosts
- ... across multiple networks
  - The Internet

How do you solve a problem?

1. Decompose it (into tasks and abstractions)

2. Assign tasks to entities (who does what)

Modularity

Modularity based on abstraction is the way things are done

– Barbara Liskov, Turing lecture

What is modularity?

- Decomposing systems into smaller units
  - Providing a “separation of concerns”

- Plays a crucial role in computer science…

- The challenge is to find the *right* modularity

<u>Network</u> Modularity

- The need for modularity still applies
  - **And is even more important!**
- Normal modularity organizes code
- But network implementations are not just distributed across many lines of code…
  - Also distributed across many devices (hosts, routers)
  - … *and* different players (clients, server, ISPs)

How do we decompose the job of transferring data between end-hosts?

Inspiration…

- **CEO A writes letter to CEO B**
  - Folds letter and hands it to administrative aide
- **Aide:**

Dear Elon,

  - Puts letter in envelope with CEO B's full name
  - Takes to FedEx

Your days are numbered.

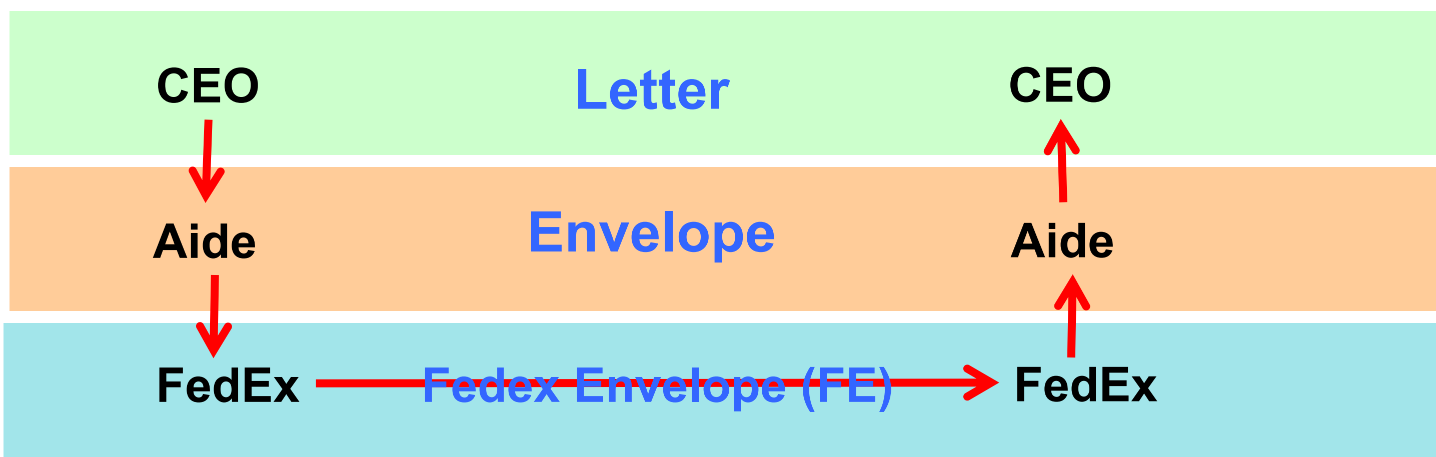
- **FedEx Office**
  - Puts letter in larger envelope

-- Satya

  - Puts name and street address on FedEx envelope
  - Puts package on FedEx delivery truck
- **FedEx delivers to other company**

The Path of the Letter

- "Peers" understand the same things
- No one else needs to
- Lowest level has most "packaging"



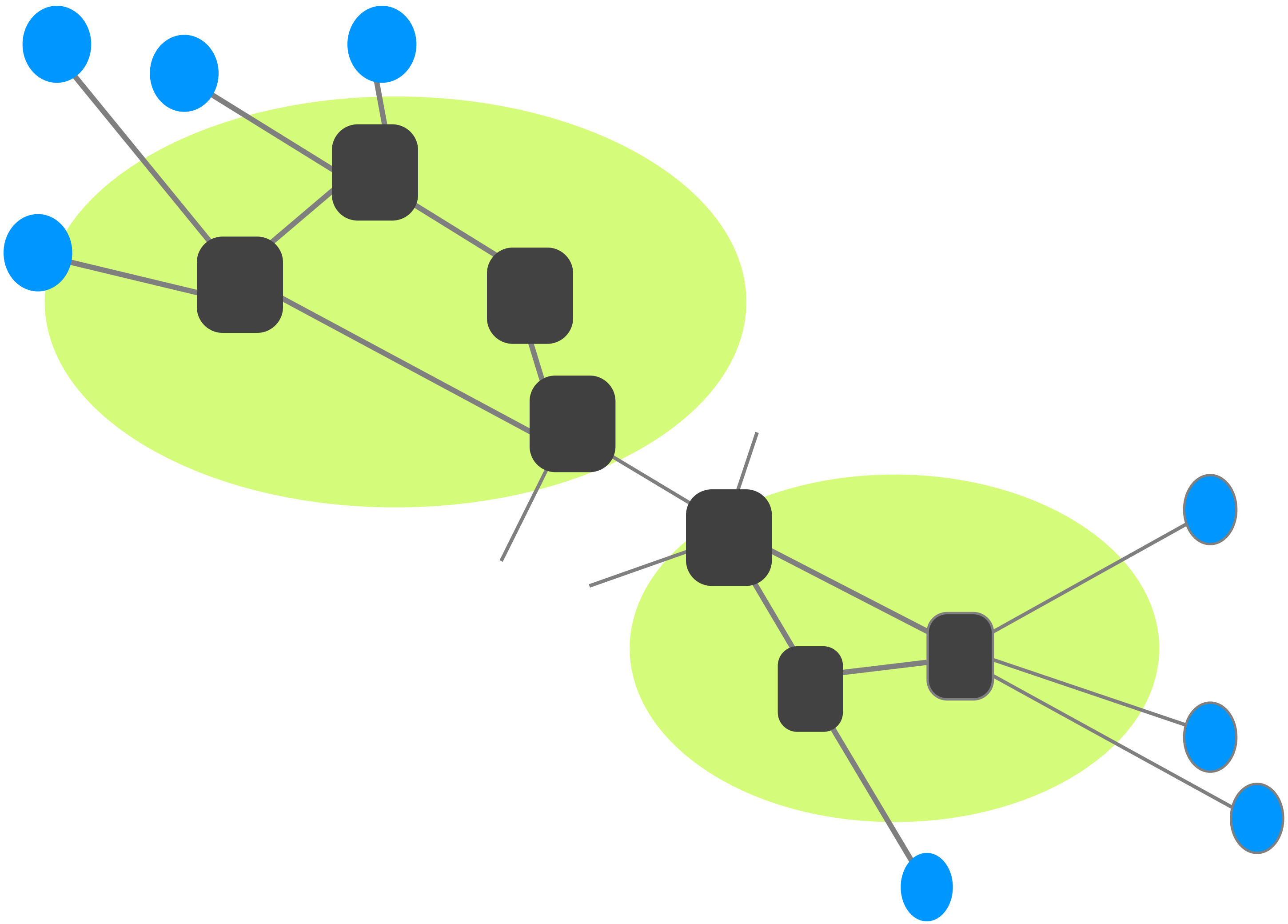
Thought Experiment

- How would ***you*** break the Internet into tasks?
- Just focus on what is needed to get packets between processes on different hosts….
- Do not consider application or control tasks
  - Naming, computing forwarding tables, etc.

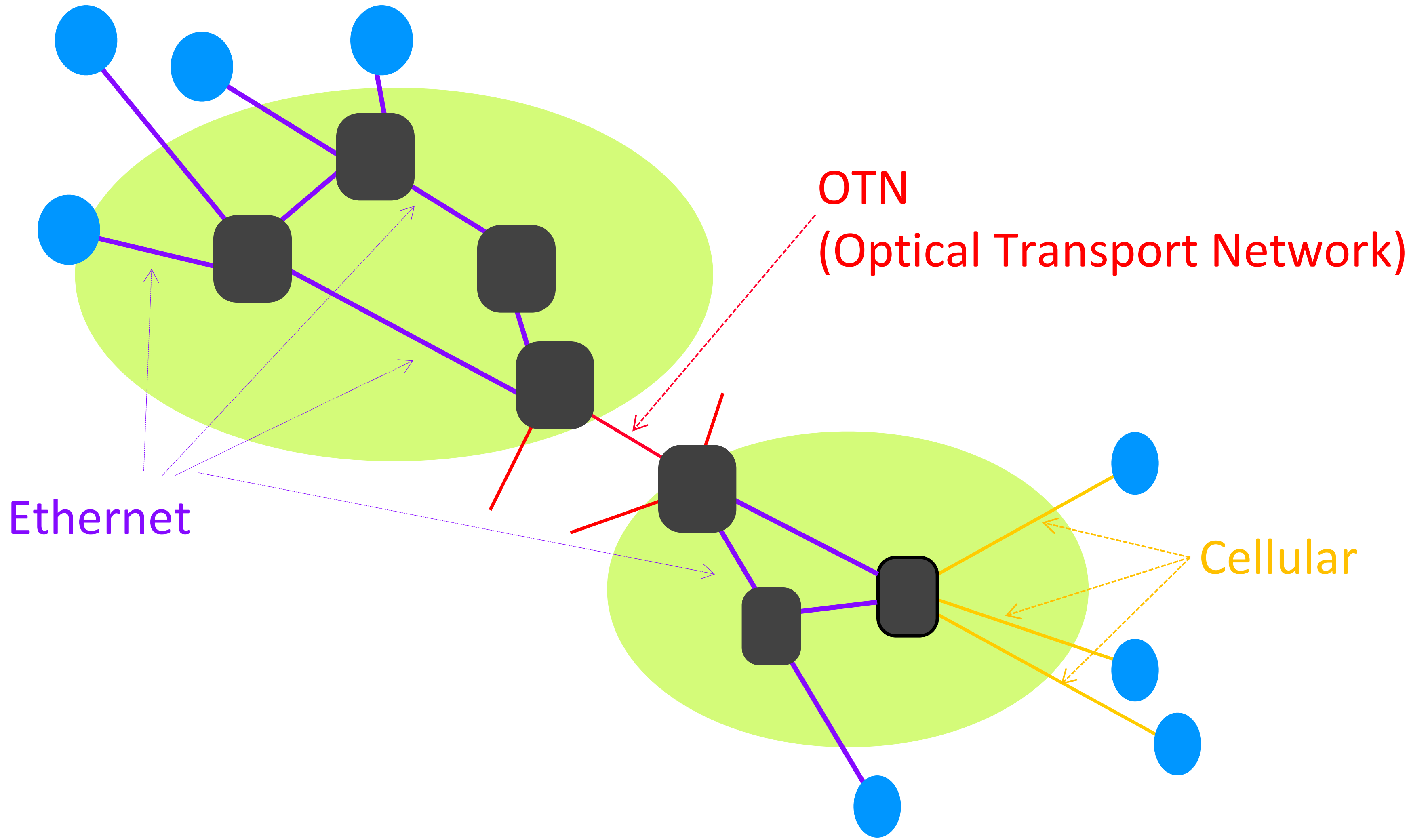
Breakdown into Tasks

- Bits across a link
- Packets across a link
- Deliver packets across a single network (“local” delivery)
- Deliver packets across multiple networks (“global” delivery)
- Deliver data reliably
- Do something with the data

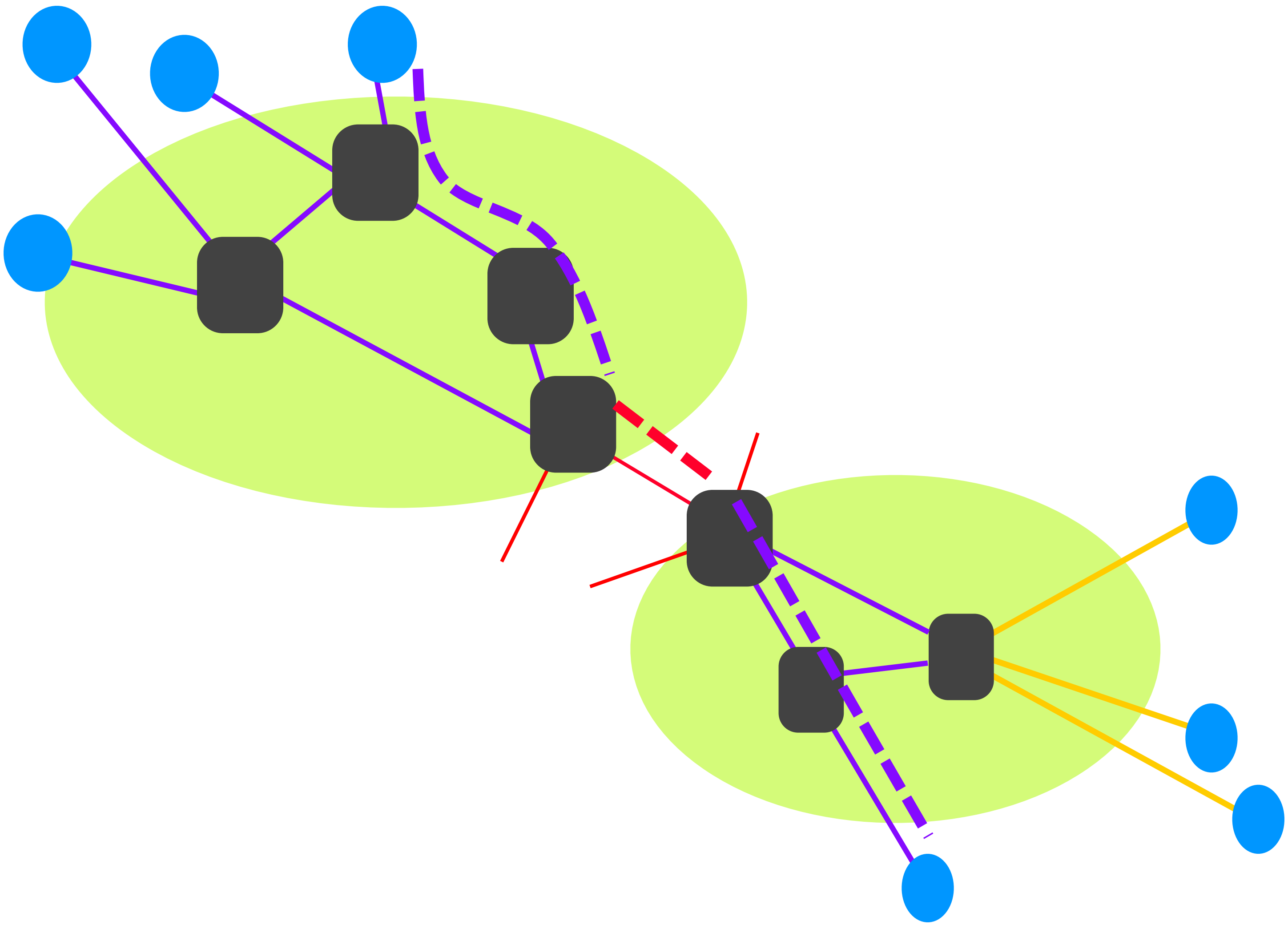
Local *vs.* Global Delivery



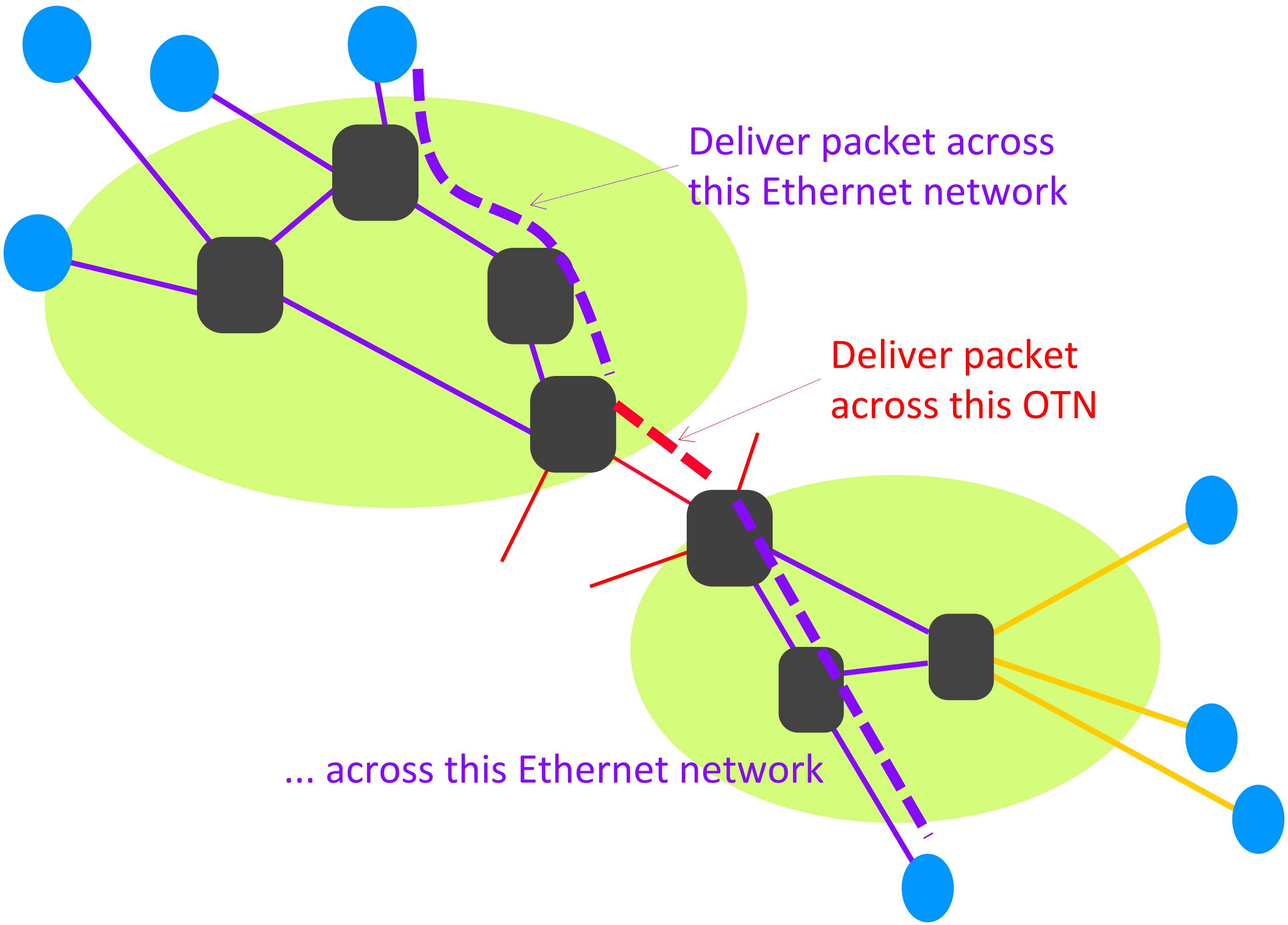
Local *vs.* Global Delivery



Local *vs.* Global Delivery



Local *vs.* Global Delivery



Breakdown into Tasks

- Bits across a link
- Packets across a link
- Deliver packets across a single network (“local” delivery)
- Deliver packets across multiple networks (“global” delivery)
- Deliver data reliably
- Do something with the data

Breakdown into Tasks

- Bits across a link
- Packets across a link and local network (“local” delivery)
- Deliver packets across multiple networks (“global” delivery)
- Deliver data reliably
- Do something with the data

Breakdown into Tasks

- Bits across a link
- Packets across a link and local network (“local” delivery)
  - Local addresses
- Deliver packets across multiple networks (“global” delivery)
  - Global addresses
- Deliver data reliably
- Do something with the data

In the Internet: organization

Applications

Reliable (or unreliable) data delivery

Best-effort *global* packet delivery

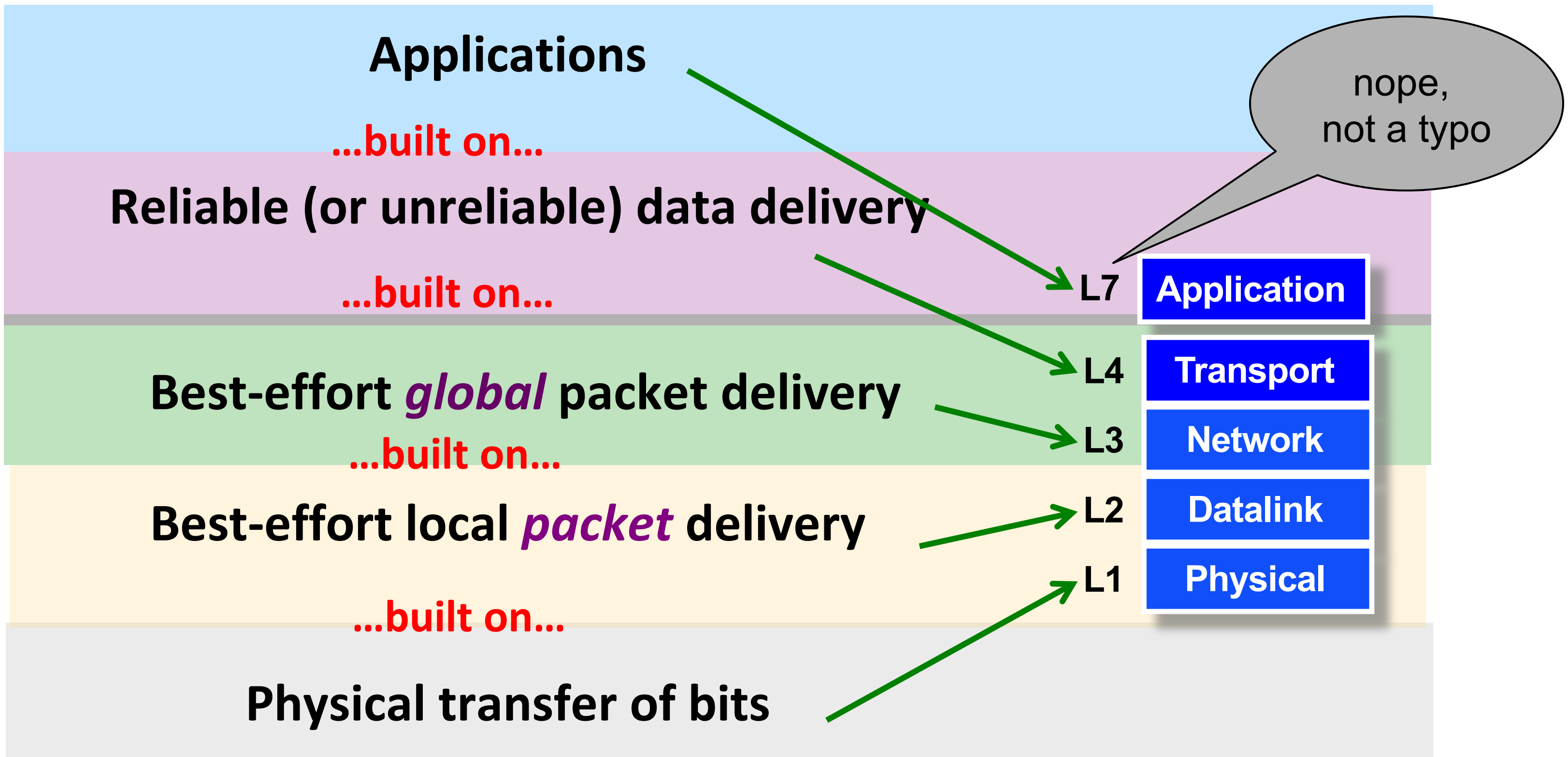
Best-effort local *packet* delivery

Physical transfer of bits

A <u>layered</u> architecture

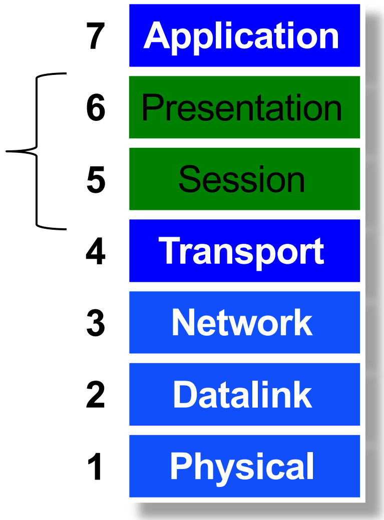
- Layer = a part of a system with well-defined interfaces to other parts
- **One layer interacts only with layer above and layer below**
- Two layers interact only through the interface between them

In the Internet: organization



Ancient history (late 1970s)

The Open Systems Interconnect (OSI) model developed by the International Organization for Standardization (ISO) included two additional layers that are often implemented as part of the application



Questions?

Recall: peers understand the same things

CEO	Letter	CEO
Aide	Envelope	Aide
FedEx	Fedex Envelope (FE)	FedEx

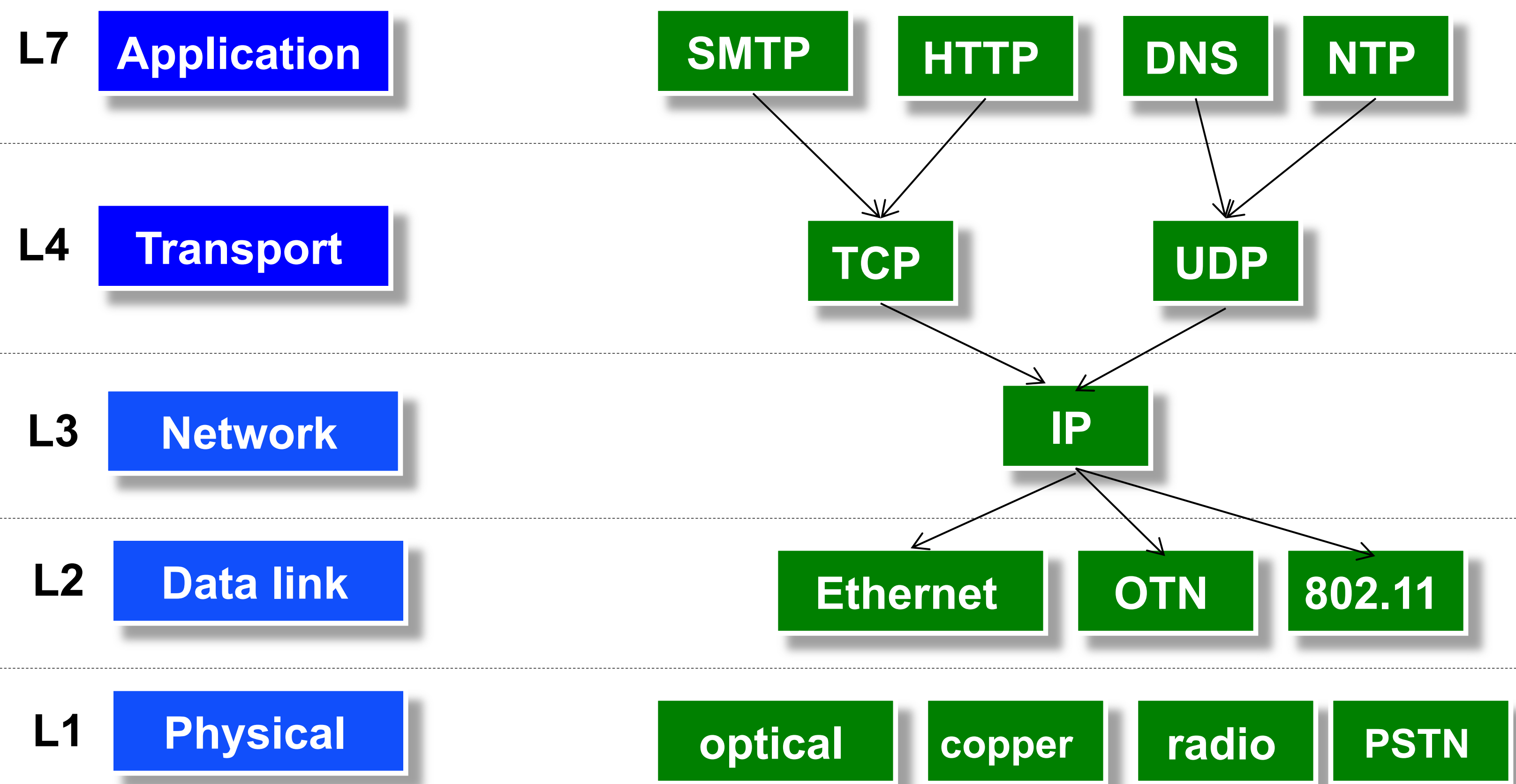
Protocols and Layers

Communication between peer layers on different systems is defined by protocols

What is a Protocol?

- An agreement between parties on how to communicate
- Defines the syntax of communication
- And semantics
  - “first a hullo, then a request…”
  - essentially, a state machine
  - we’ll study many protocols later in the semester
- Protocols exist at many layers
  - defined by a variety of standards bodies (IETF, IEEE, ITU)

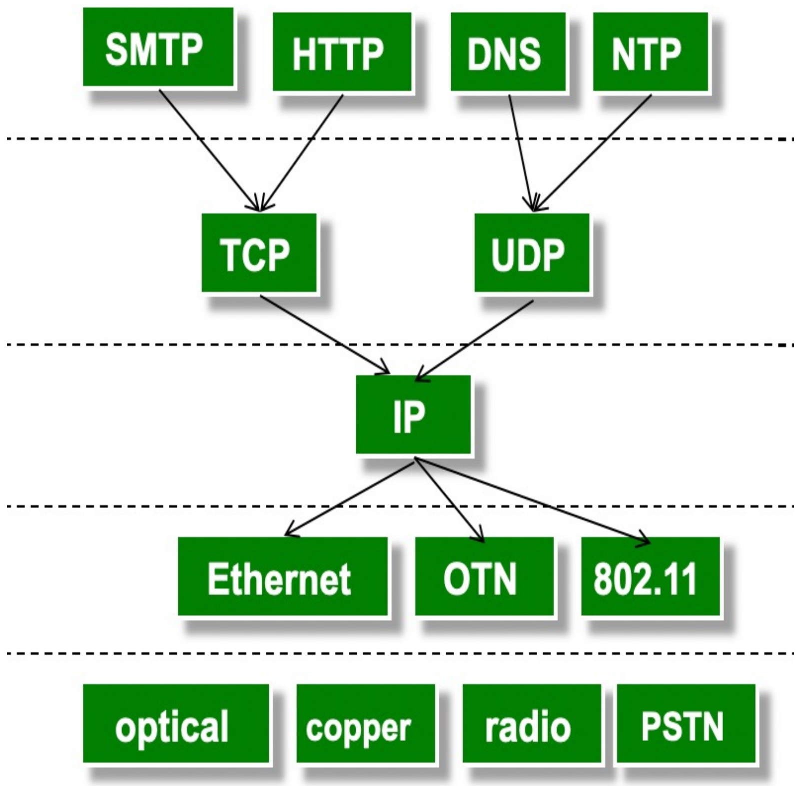
Protocols at different layers



There is just one network-layer protocol!

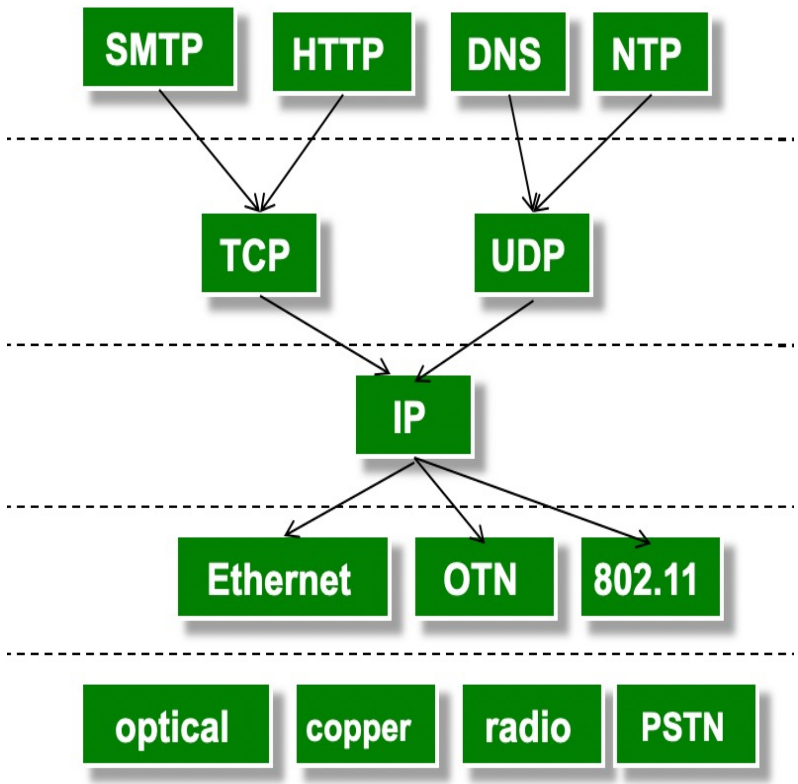
Recap: Three important properties

- Each layer:
  - Depends on layer below
  - Supports layer above
  - Independent of others
- Multiple versions in a layer
  - Interfaces differ somewhat
  - Components at one layer pick which lower-level protocol to use
- But only one IP layer
  - Unifying protocol



Why is layering important?

- Innovation can proceed largely in parallel!
- Pursued by very different communities
  - App devs (L7), chip designers (L1/L2)
- Leading to innovation at most levels
  - Applications (lots)
  - Transport (some)
  - Network (few)
  - Physical (lots)



Questions?

How do you solve a problem?

1. Decompose it (into tasks and abstractions)
2. Assign tasks to entities (who does what)

Distributing Layers Across Network

- Layers are simple if only on a single machine
  - Just stack of modules interacting with those above/below
- But we need to implement layers across:
  - Hosts
  - Routers (switches)
- What gets implemented where?

What gets implemented at the end host?

- Bits arrive on wire ... → must implement L1
- ... must make it up to app → must implement L7

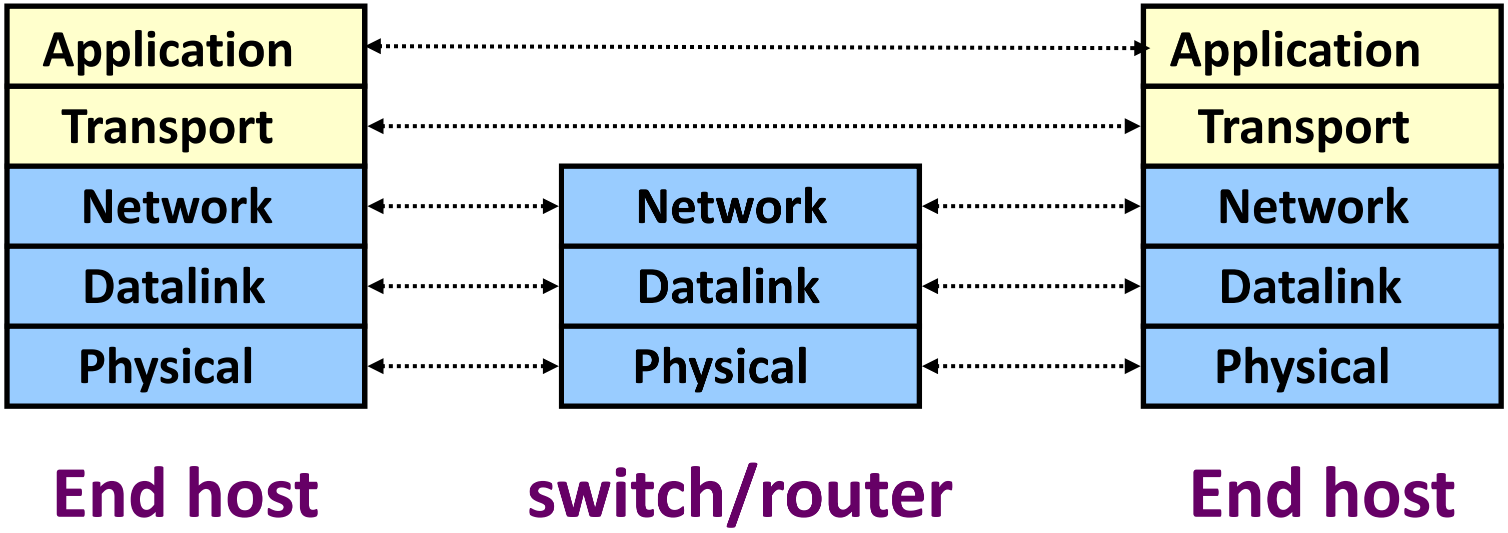
- Therefore, all layers must exist at host!

What gets implemented in the network?

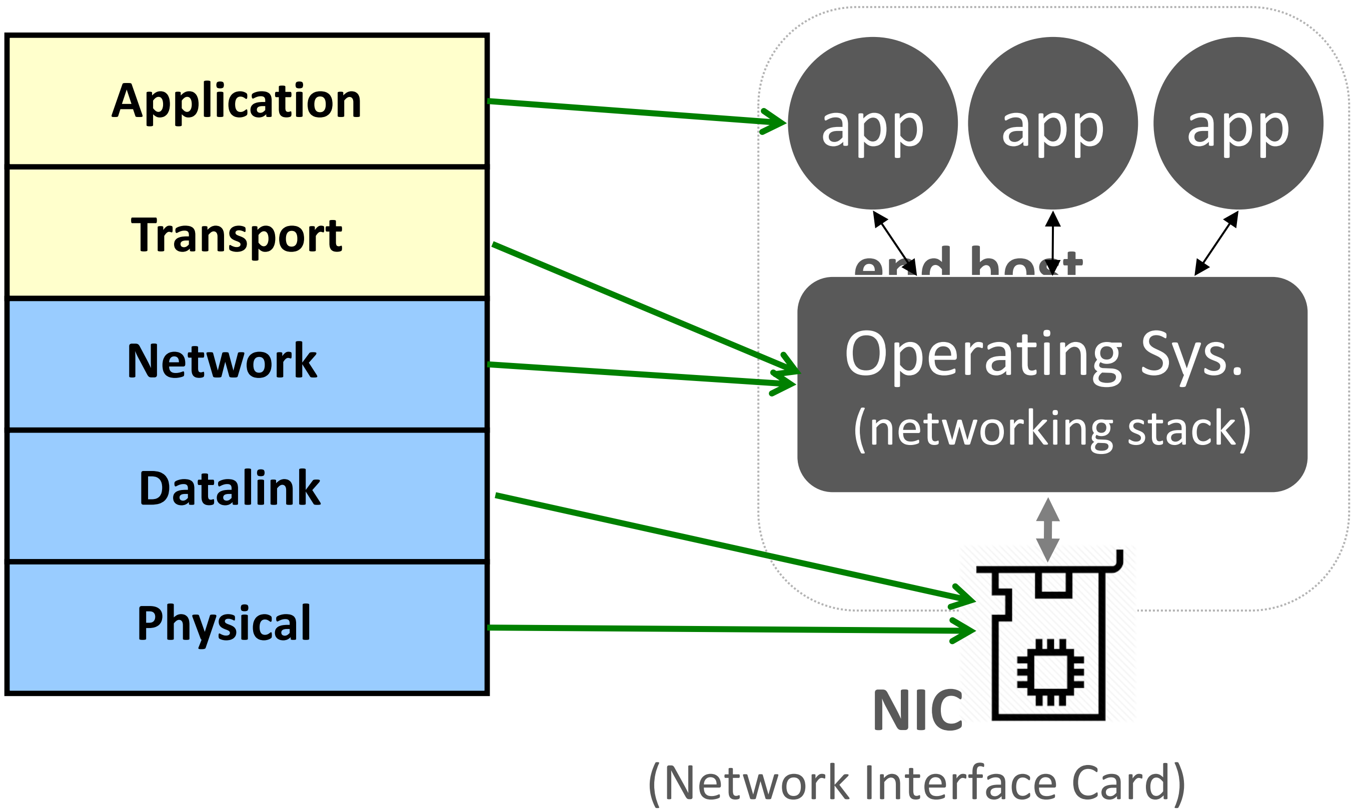
- Bits arrive on wire → physical layer (L1)
- Packets must be delivered across links and local networks → datalink layer (L2)
- Packets must be delivered between networks for global delivery → network layer (L3)
- The network does not support reliable delivery
  - Transport layer (and above) ***not*** supported

Simple Diagram

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

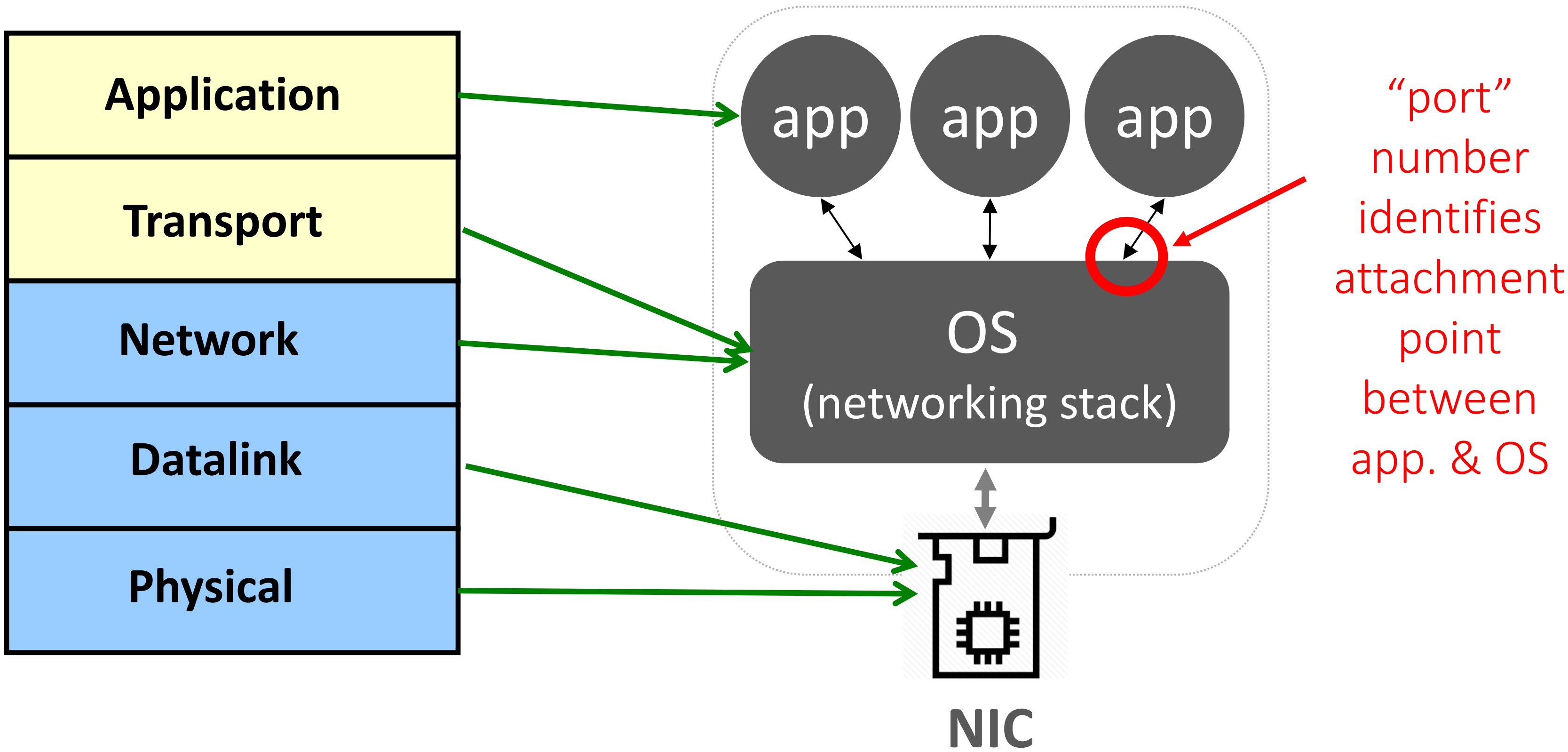


A closer look: end host



Note: addressing *within* the end host

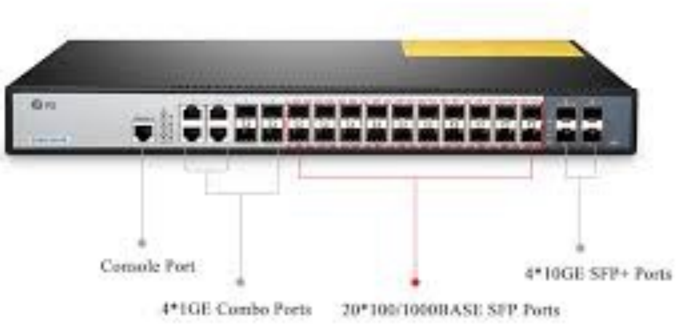
Recall: packet contains the destination host's address



When a packet arrives at the host, how does the OS know which app to send the packet to?

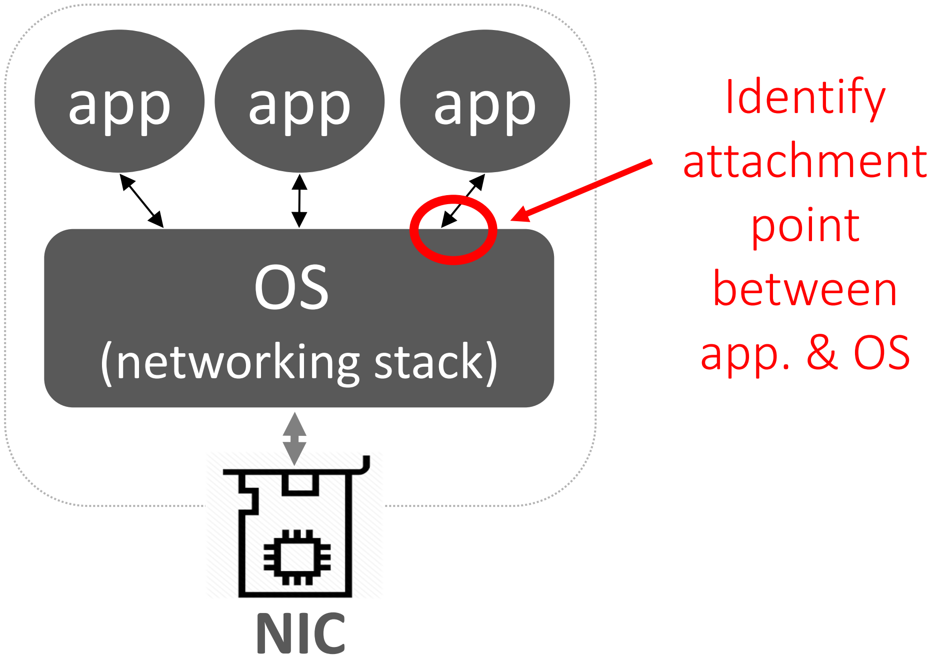
Network "ports": two types

- Switches/routers have **physical ports**:
  - Places where links connect to switches



Network "ports": two types

- Switches/routers have **physical ports**:
  - Places where links connect to switches
- The OS supports **logical ports**:
  - Place where app connects to OS network stack



Of Sockets and Ports

- **Socket:** an OS mechanism that connects app processes to the networking stack
- When an app wants access to the network, it opens a **socket**, which is associated with a **port**
  - *This is not a physical port, just a logical one*
- The **port number** is used by the OS to direct incoming packets to its associated socket

Section will cover sockets in detail

Implications for Packet Header

- Packet header must include:
  - Destination host address (used by network to reach host)
  - Destination port (used by host OS to reach app) [new!]

- When a packet arrives at the destination end-host, it is delivered to the socket (process) associated with the packet's destination port

OS Network Stack Is An Intermediary

- Application has very clear task (w.r.t. network)
  - Thinks about data
- NIC/driver has very clear task
  - Thinks about packets
- Network stack in the intermediary between them
  - Translates between their abstractions

Recap: layers at the end host

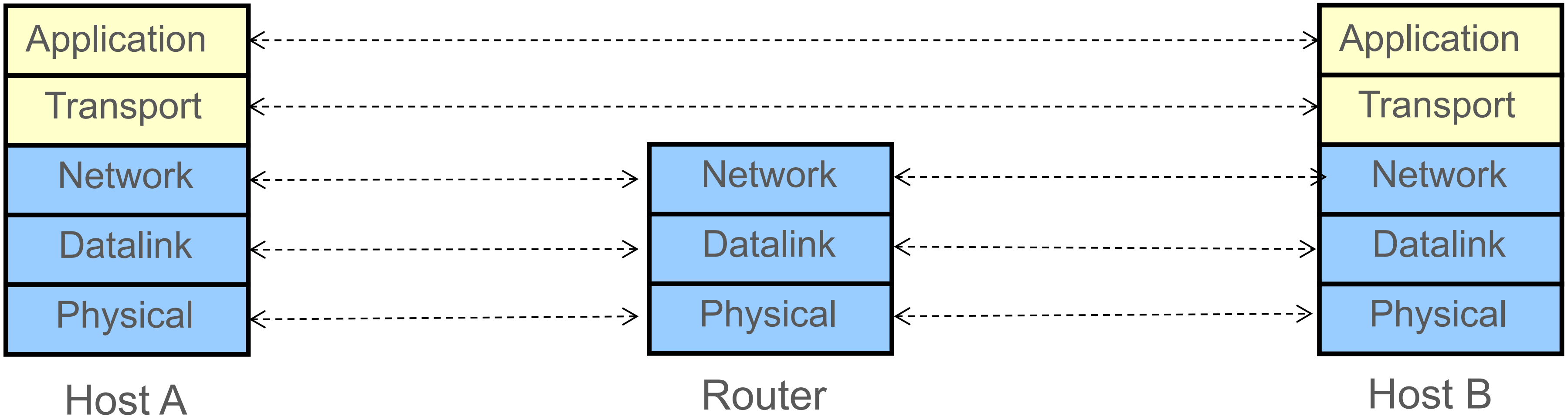
- **Application layer (L7)**
  - part of the app: browser, mail client,…
- **Transport and network layer (L3, L4)**
  - typically part of the OS
- **Datalink and physical layer (L1, L2)**
  - hardware/firmware/drivers

A closer look: network

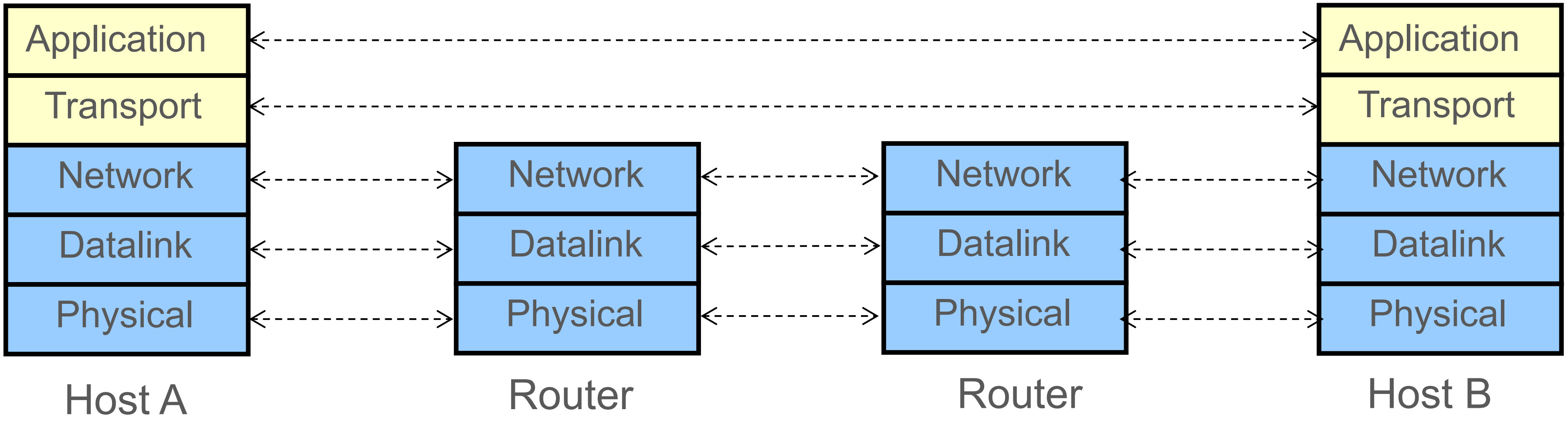
- Bits on wire → physical layer (L1)
- Local delivery of packets → datalink layer (L2)
- Global delivery of packets → network layer (L3)

Recall: Logical Communication

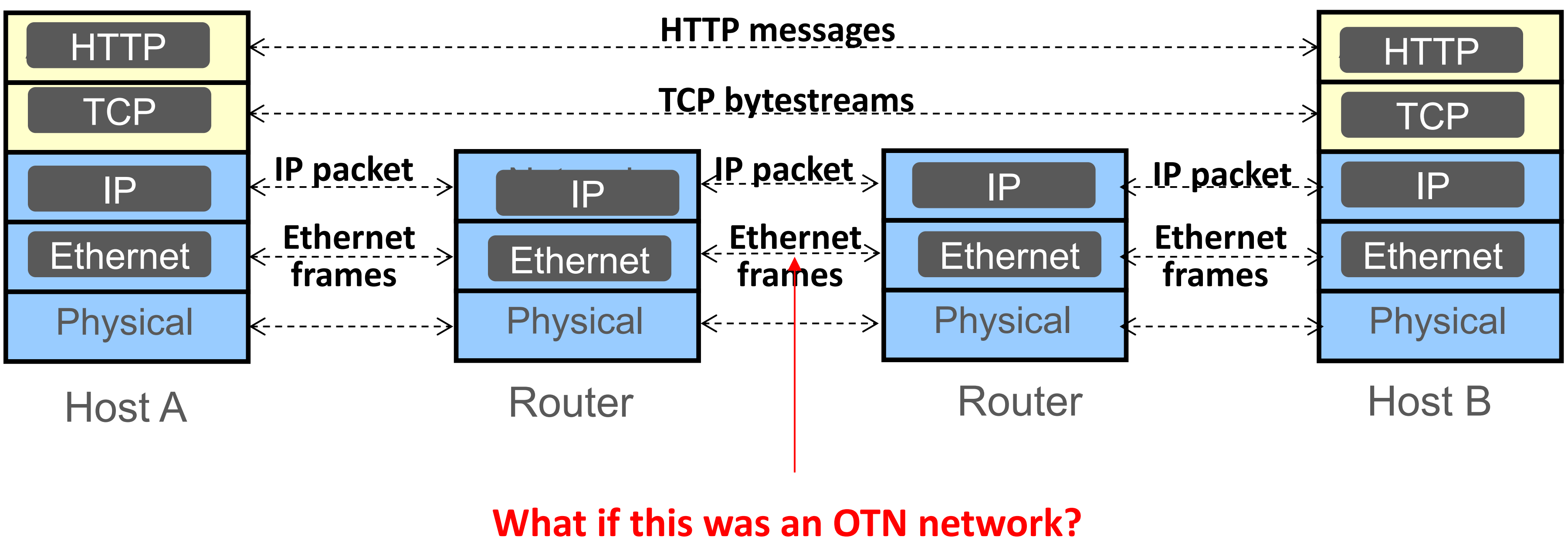
- Layers interact with peer's corresponding layer
- Lower three layers implemented everywhere
- Top two layers implemented only at hosts



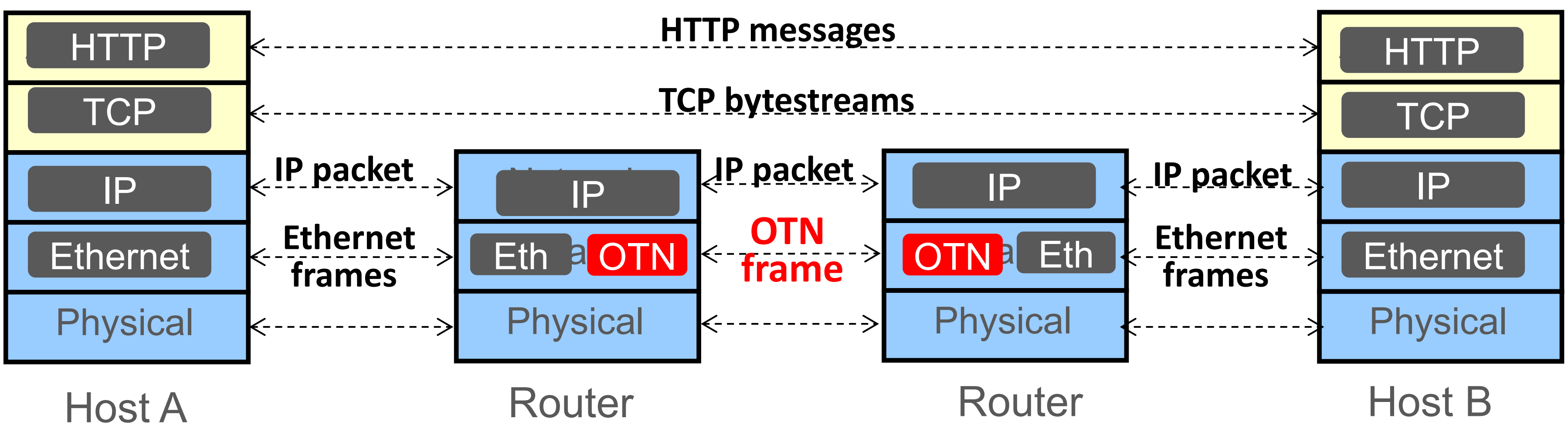
A closer look: network



Example: simple protocol diagram



Example: simple protocol diagram



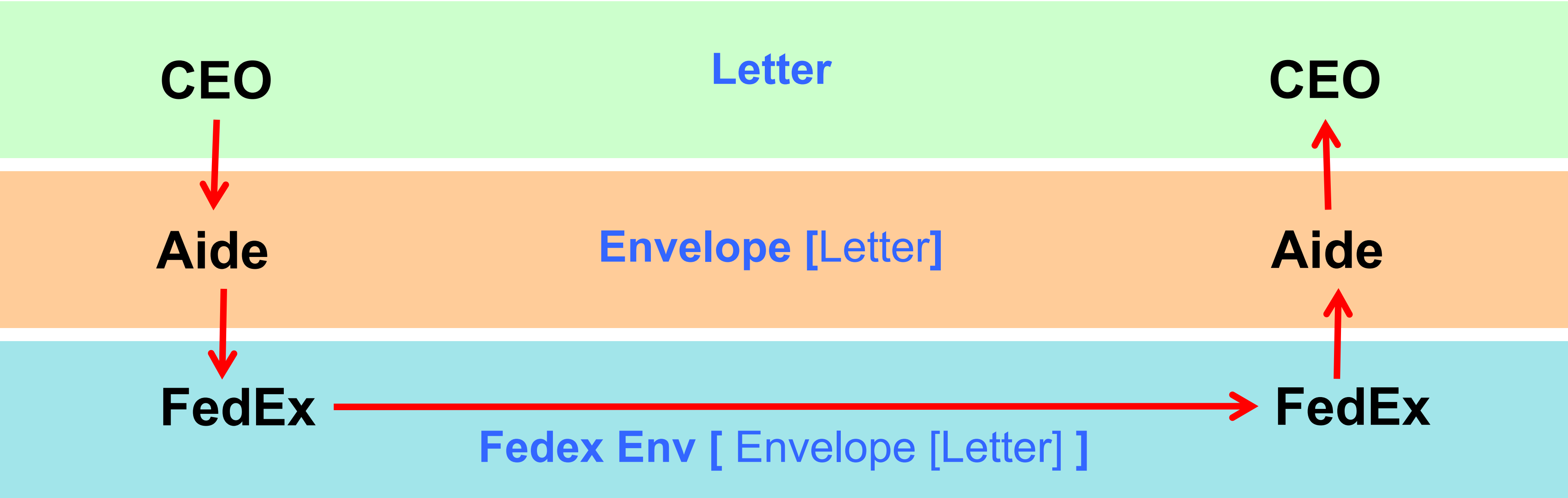
Recap: Physical Communication

- Communication goes down to physical network
- Then up to relevant layer

Recall: the path of the letter

Recap: Physical Communication

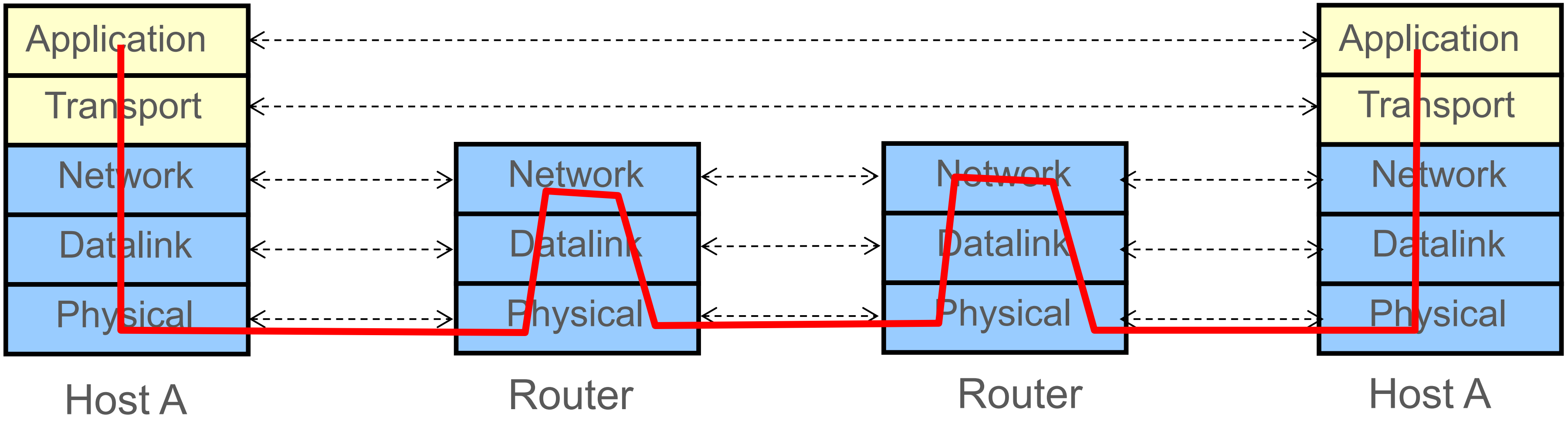
- Communication goes down to physical network
- Then up to relevant layer
- Lowest layer has the most "packaging"



Recall: the path of the letter

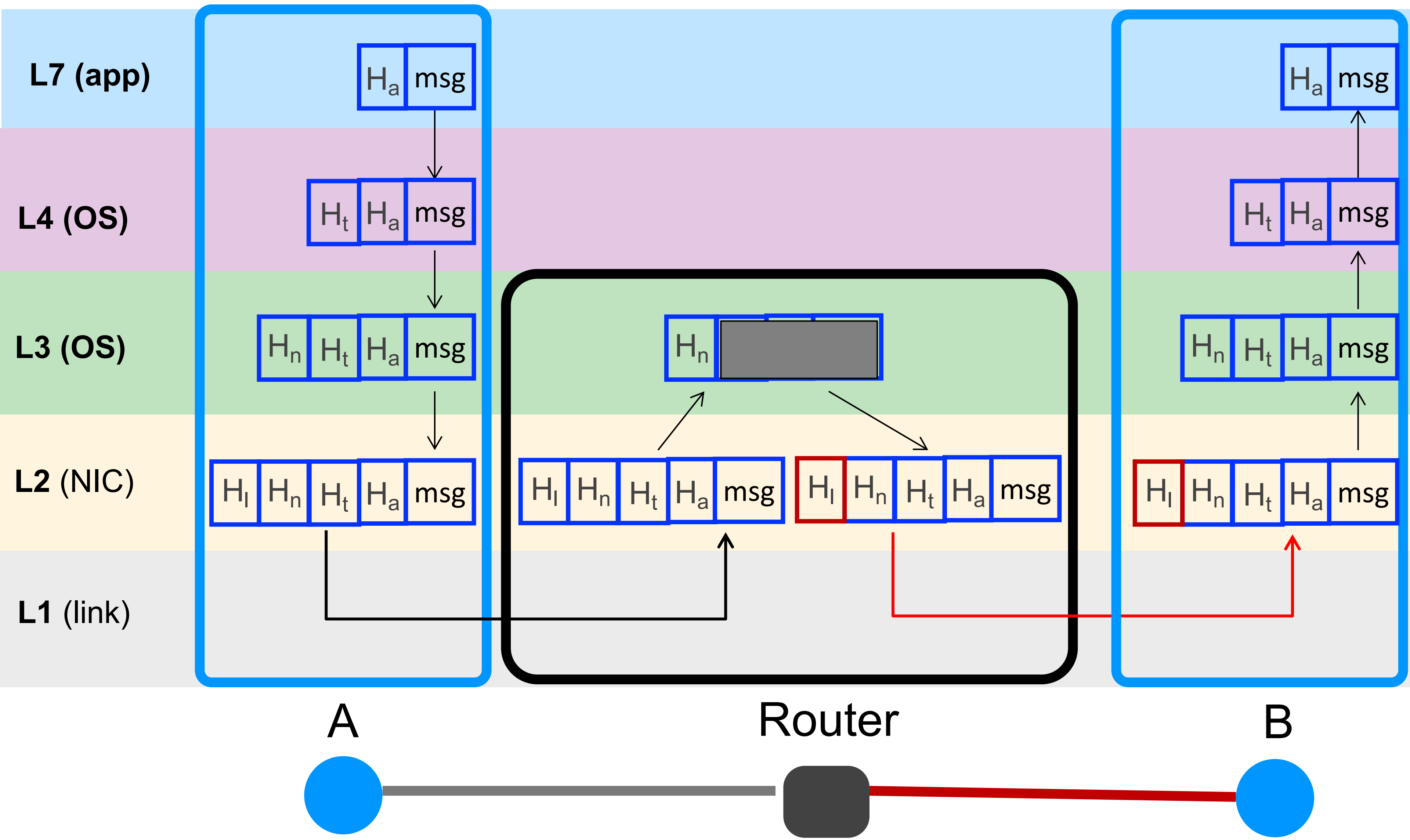
Recap: Physical Communication

- Communication goes down to physical network
- Then up to relevant layer

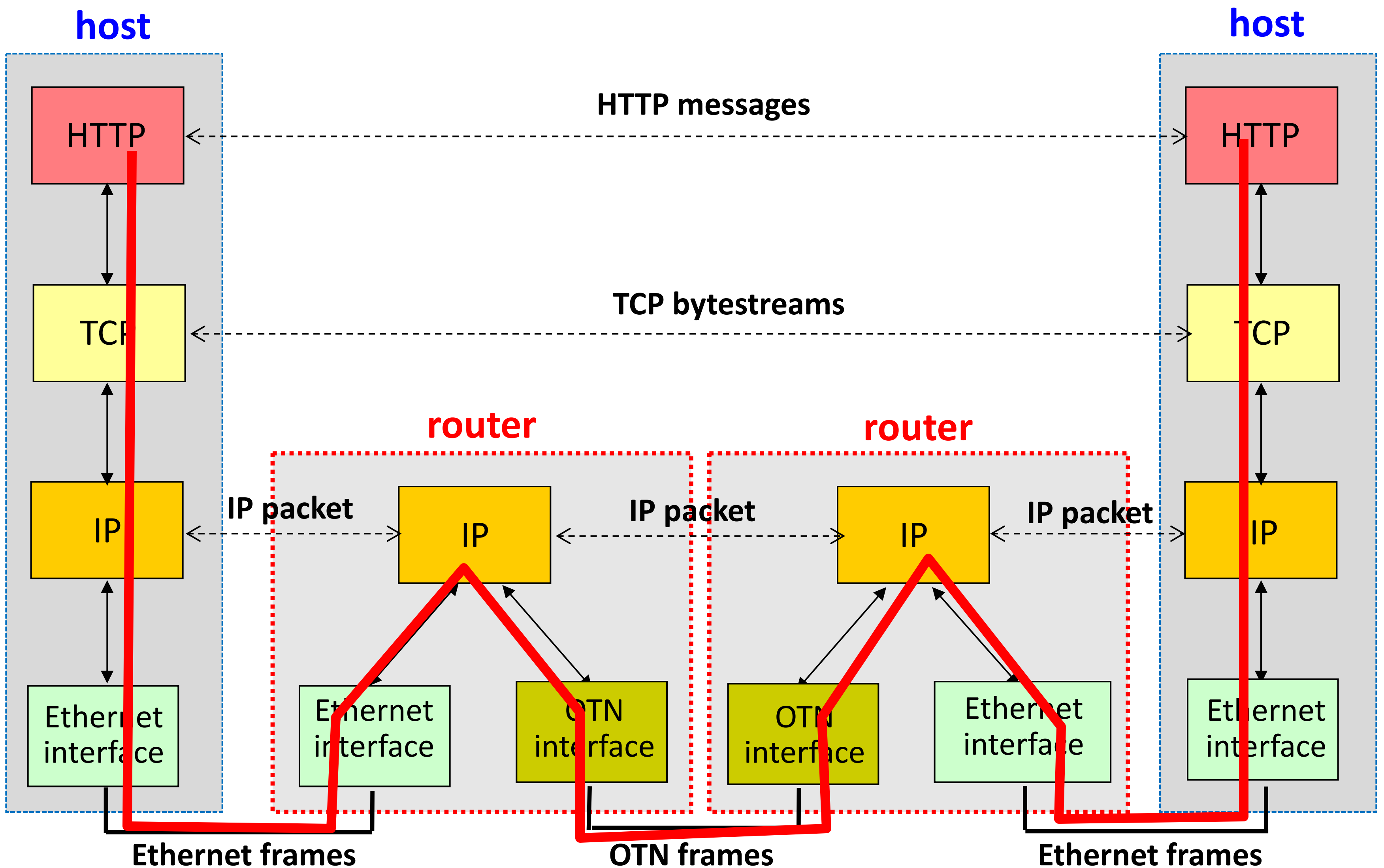


Layer Encapsulation

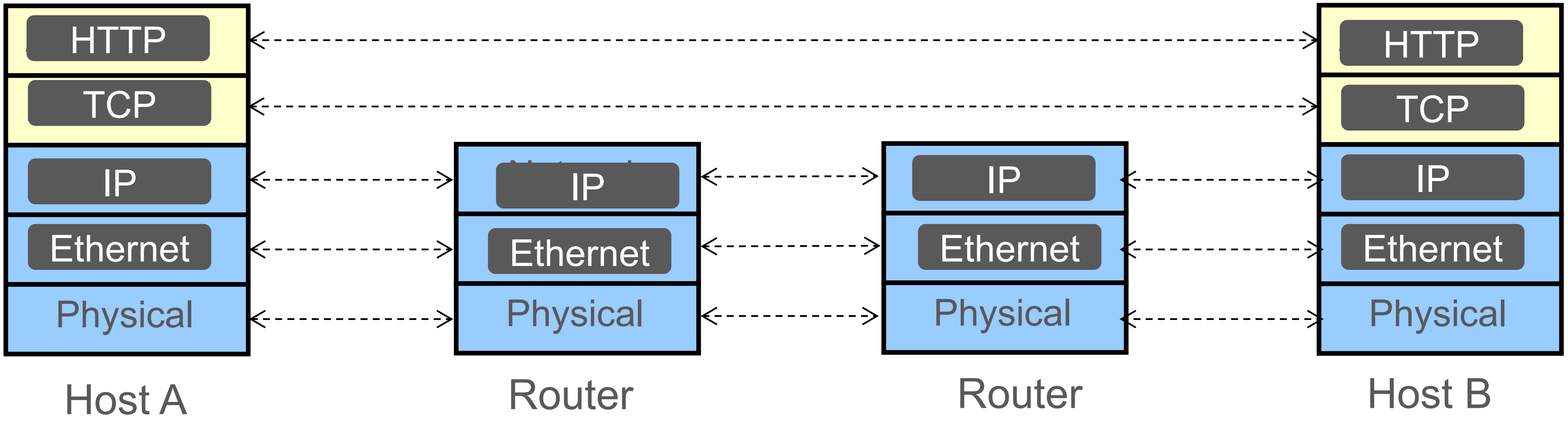
On the wire: packet has data + headers from all layers



Complicated protocol diagram

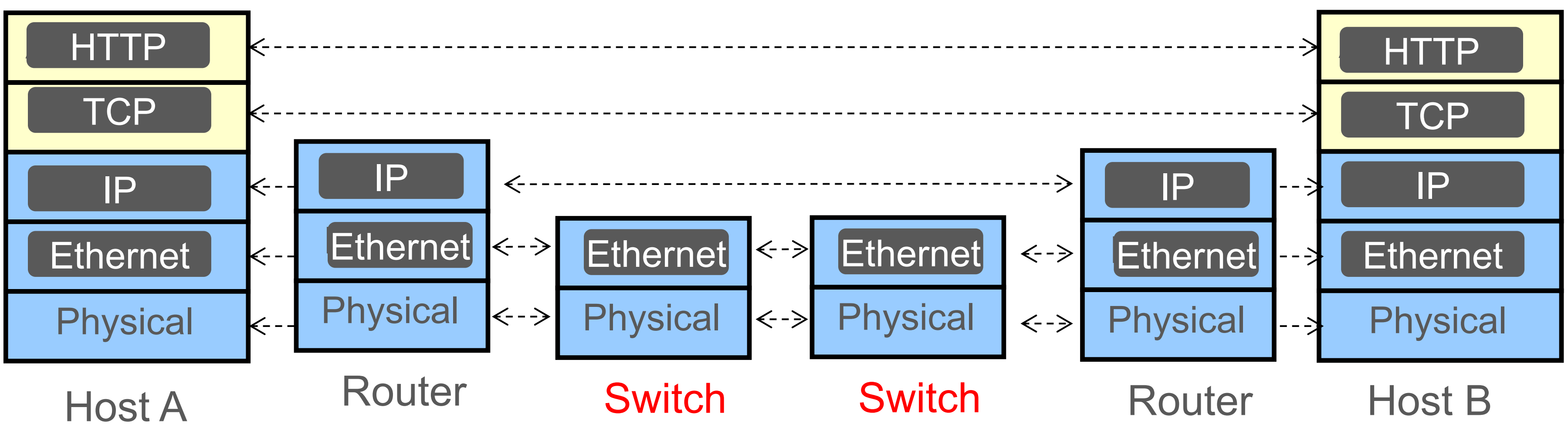


A side note: switches vs. routers

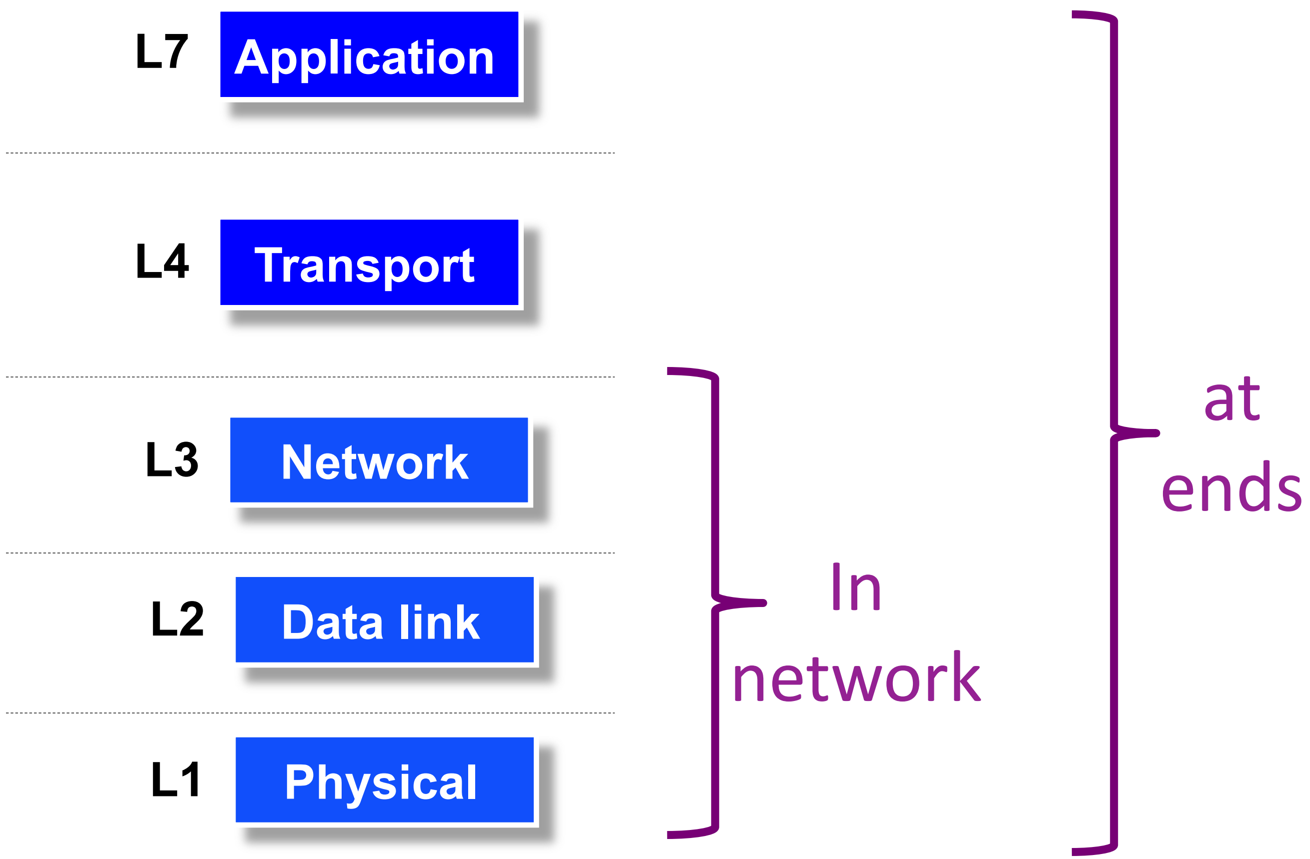


A side note: switches vs. routers

- Historically: switches implemented L1, L2 and routers L1, L2, L3
- These days, most switches also implement L3 hence we use the term switches and router interchangeably



Review



Next lecture: why is *this* a good assignment of?