Architecting the Internet

Lecture 2, Spring 2026

Internet Design Principles

- **Architecting the Internet**
- Narrow Waist, Demultiplexing
- End-to-End Principle

Designing Resource Sharing

- Statistical Multiplexing
- Circuit vs. Packet Switching
- Which is Better?
- A Brief History

The Internet Design Principles:

1. Decentralized control.
2. Best-effort service model.
3. Route around trouble.
4. Dumb infrastructure (with smart endpoints).
5. End-to-end principle.
6. Layering.
7. Federation via narrow-waist interface.

These are guidelines, not unbreakable rules.

- This is just one of many possible designs.
- We're still debating the big questions!

The Internet Design Principles

1. **Decentralized control**.
   - Each network device (e.g. router) runs on its own. No central mastermind.
2. **Best-effort service model**.
   - At Layer 3, routers only offer best-effort delivery.
3. **Route around trouble.**
   - Network must be resilient to failures.
   - If a router or link goes down, find a different path through the network.
4. **Dumb infrastructure (with smart endpoints)**.
   - Routers forward packets. They don't care about what's inside.
5. **End-to-end principle**.
   - Implement features at the end hosts, not at the routers.

6. **Layering**.

- Each layer relies on the layer below, and supports the layer above.
- Allows us to innovate at one layer, without disturbing other layers.
- Alternative: cross-layer protocols spanning multiple layers let us optimize several layers together.

7. **Federation via narrow-waist interface**.

- Federation works because all operators speak the same Layer 3 protocol.

Narrow Waist, Demultiplexing

Lecture 2, Spring 2026

Internet Design Principles

- Architecting the Internet
- **Narrow Waist, Demultiplexing**
- End-to-End Principle

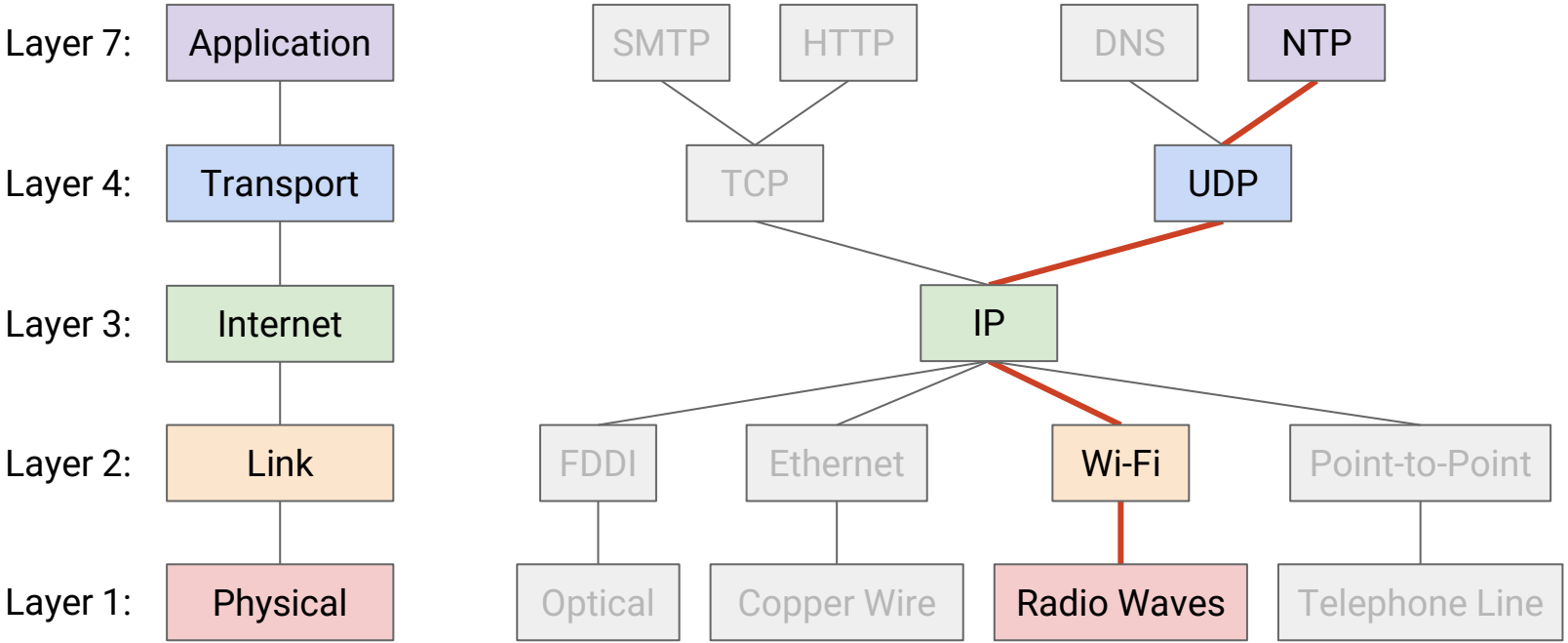
Designing Resource Sharing

- Statistical Multiplexing
- Circuit vs. Packet Switching
- Which is Better?
- A Brief History

Protocols at Different Internet Layers

Multiple protocols exist at each layer.

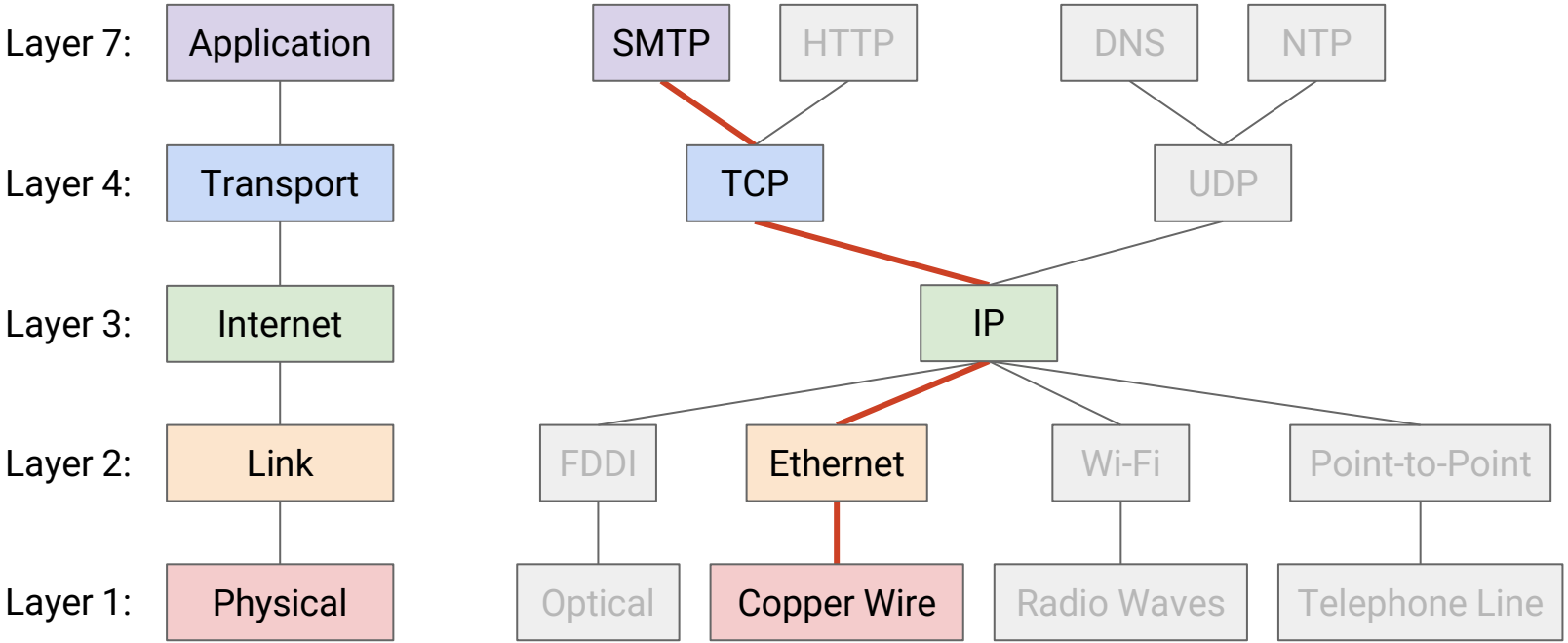
- End hosts can agree on the L4 and L7 protocols they want to use.
- Routers on each link can agree on the L1 and L2 protocols they want to use.



Protocols at Different Internet Layers

Multiple protocols exist at each layer.

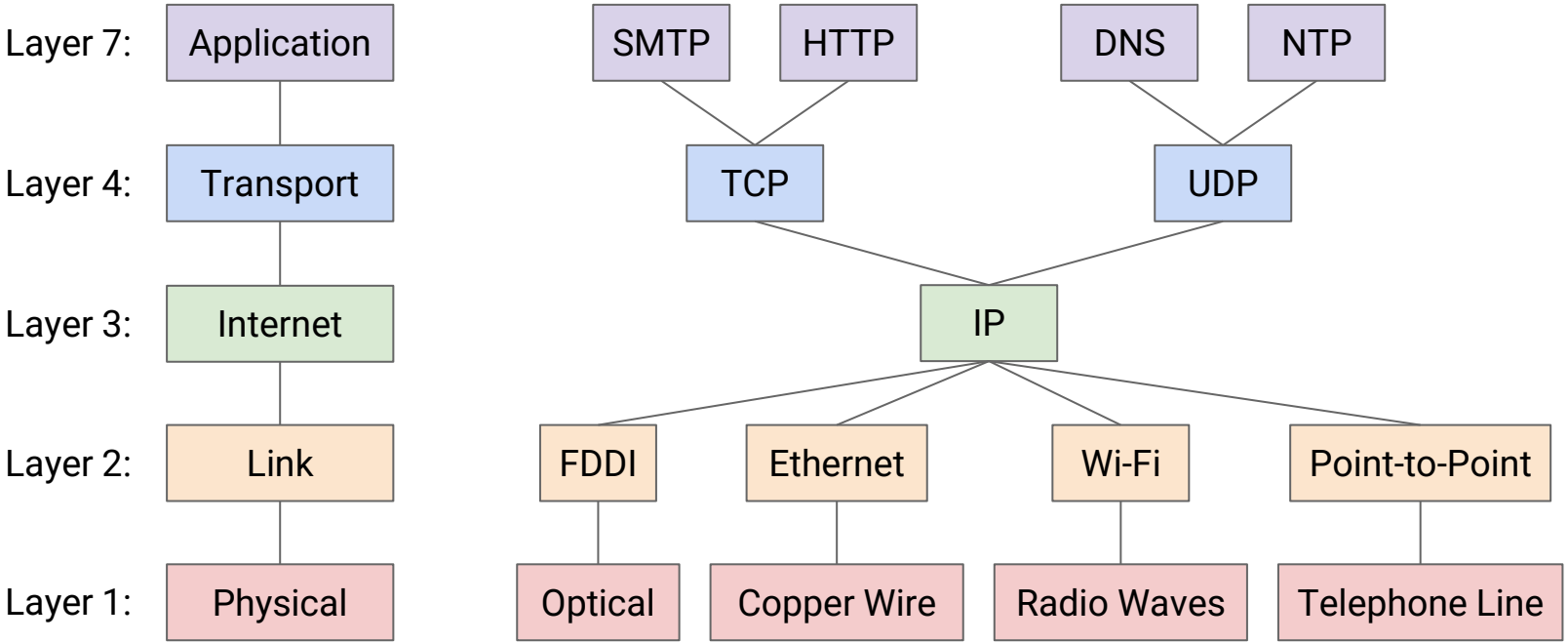
- End hosts can agree on the L4 and L7 protocols they want to use.
- Routers on each link can agree on the L1 and L2 protocols they want to use.



Protocols at Different Internet Layers

The narrow waist: IP (Internet Protocol) is the only protocol at Layer 3.

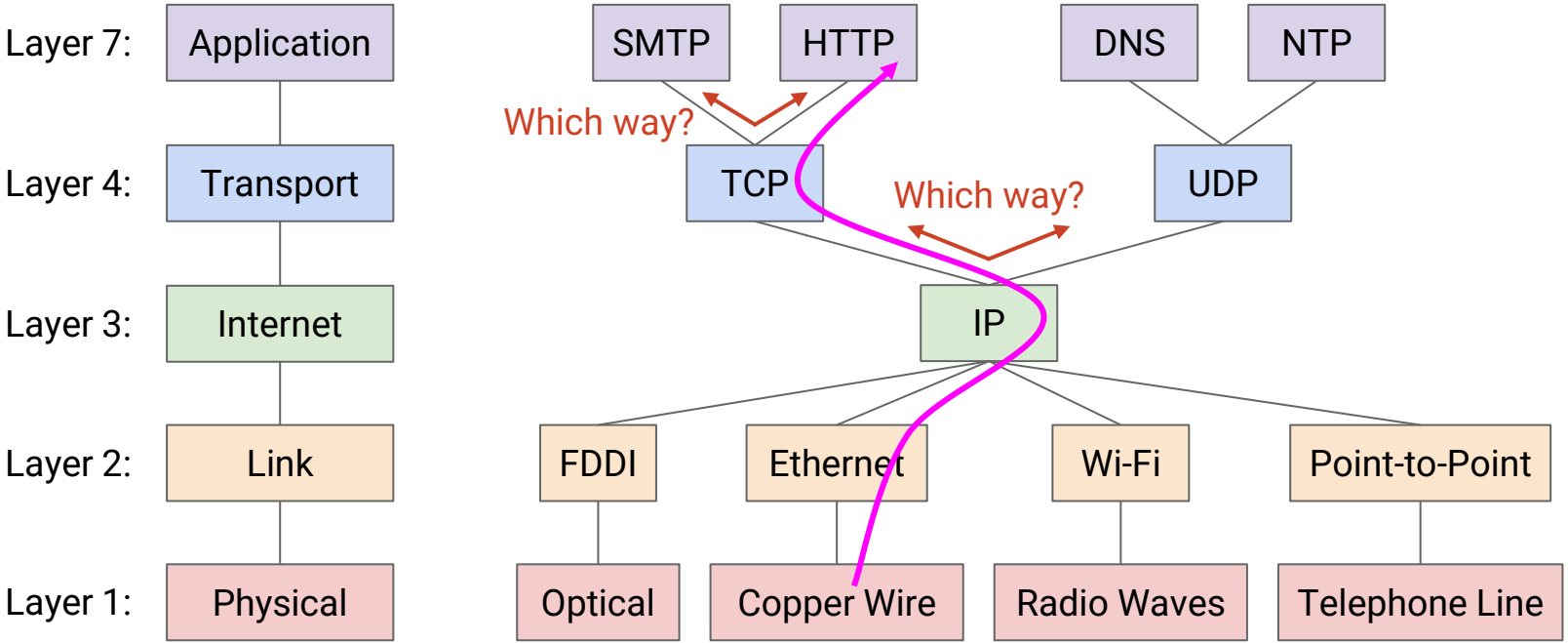
- All hosts and routers understand IP.
- This unifies the Internet and enables federation.



Demultiplexing

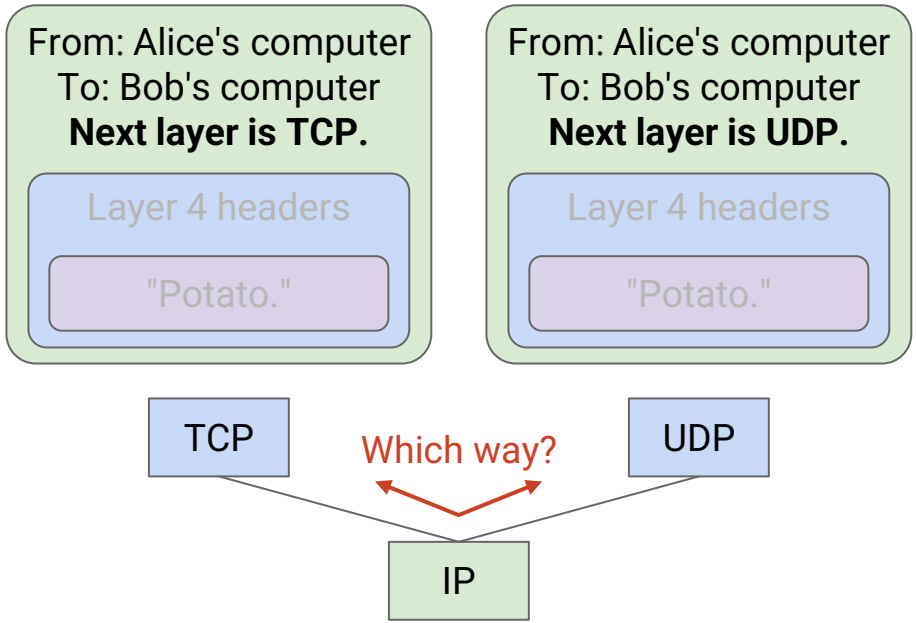
When you receive a packet, you pass it up the stack, to higher-layer protocols.

- How did IP know to pass up to TCP, not UDP?
- How did TCP know to pass up to HTTP, not SMTP?



Demultiplexing:

- Add a new header field that tells us what the next (higher) layer protocol is.
- Allows the IP code to pass the rest of the packet to the appropriate L4 code.

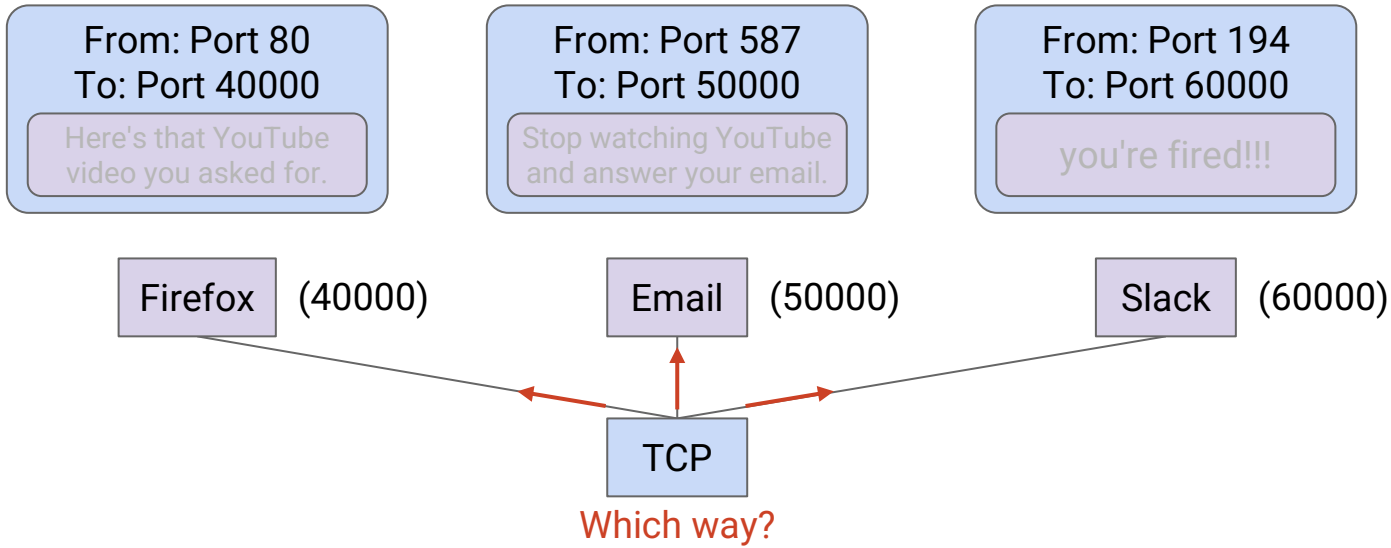


Demultiplexing at Layer 4

Demultiplexing also works at Layer 4.

More specifically, each open connection on the computer.

- Each running application on your computer is associated with a **port number**.
- When L4 receives a packet, it uses the port number to pass the packet to the corresponding application.



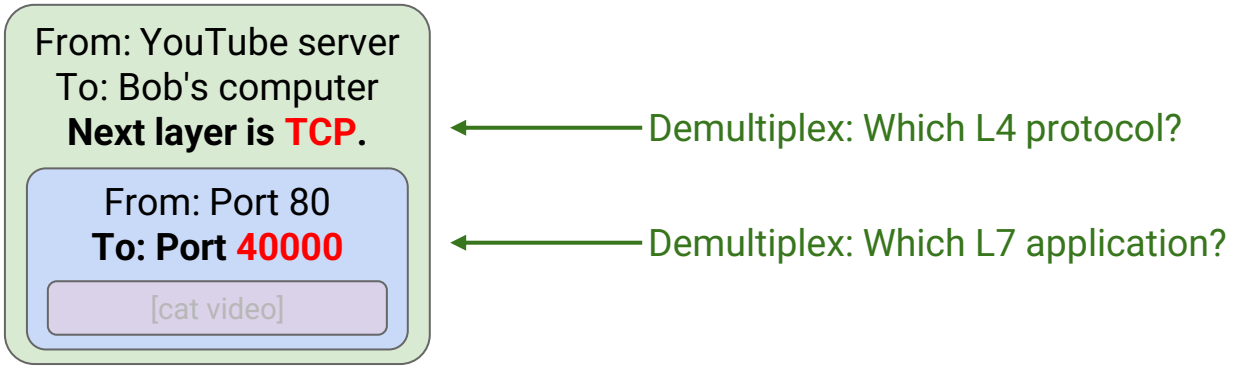
Ports at Layer 4

Port numbers help us distinguish between applications on the same computer.

- IP address (Layer 3) for all the applications is the same.
- But each connection is associated with a different port number.

Analogy: Room numbers.

- You and your housemate both have the same street address.
- If someone sends a letter to your house, who is it for?
- Distinguish by assigning room numbers to each housemate.



Ports at Layer 4

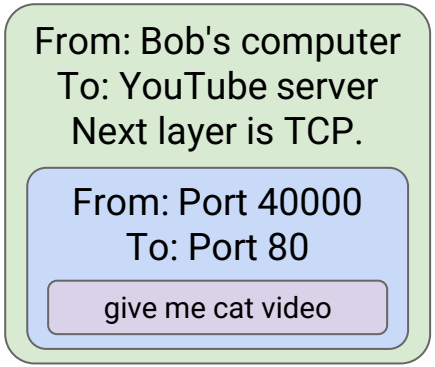
Both end hosts in a connection have a port number.

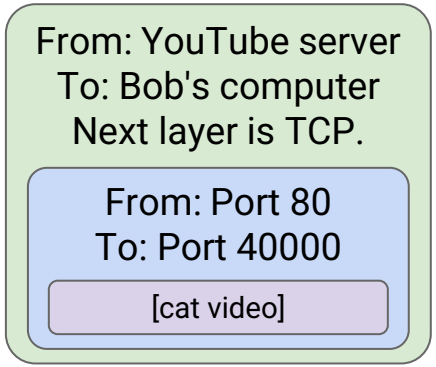
- A private client (e.g. your computer) can use a randomly-generated port number.
- A public server (e.g. YouTube) must use a fixed, well-known port number.

Analogy: Room numbers.

- Pick any number for your bedroom. No one cares.
- Public room numbers must be fixed and well-known.

Outgoing packet: Bob picks a random port number, but sends to YouTube's fixed port, 80.





Incoming reply: YouTube replies to Bob's chosen port. Bob's computer passes the packet to the correct application (e.g., Firefox).

Caution – Terminology Conflict

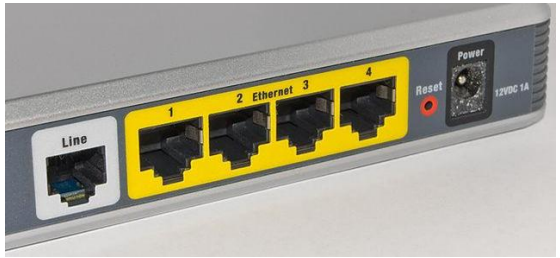
In networking, there are two different things, both called "ports."

- If it's unclear, we will specify "logical port" or "physical port."

From: Port 80
To: Port 40000

[cat video]

Logical port: A number identifying an application. Exists in software.



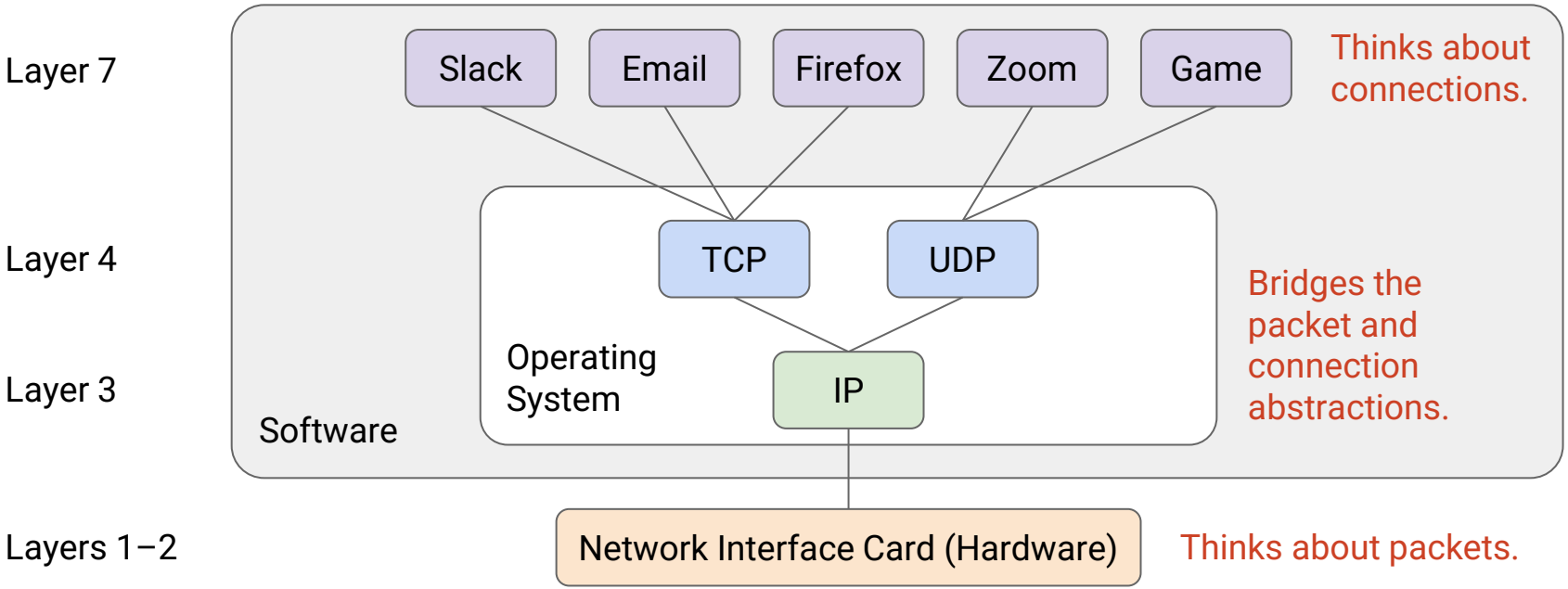
Physical port: The hole you plug a cable into. Exists in hardware.

Implementing Layers in the End Host

Layers 1 and 2 are implemented in hardware, on the network interface card (NIC).

Layers 3 and 4 are implemented in software, in the operating system.

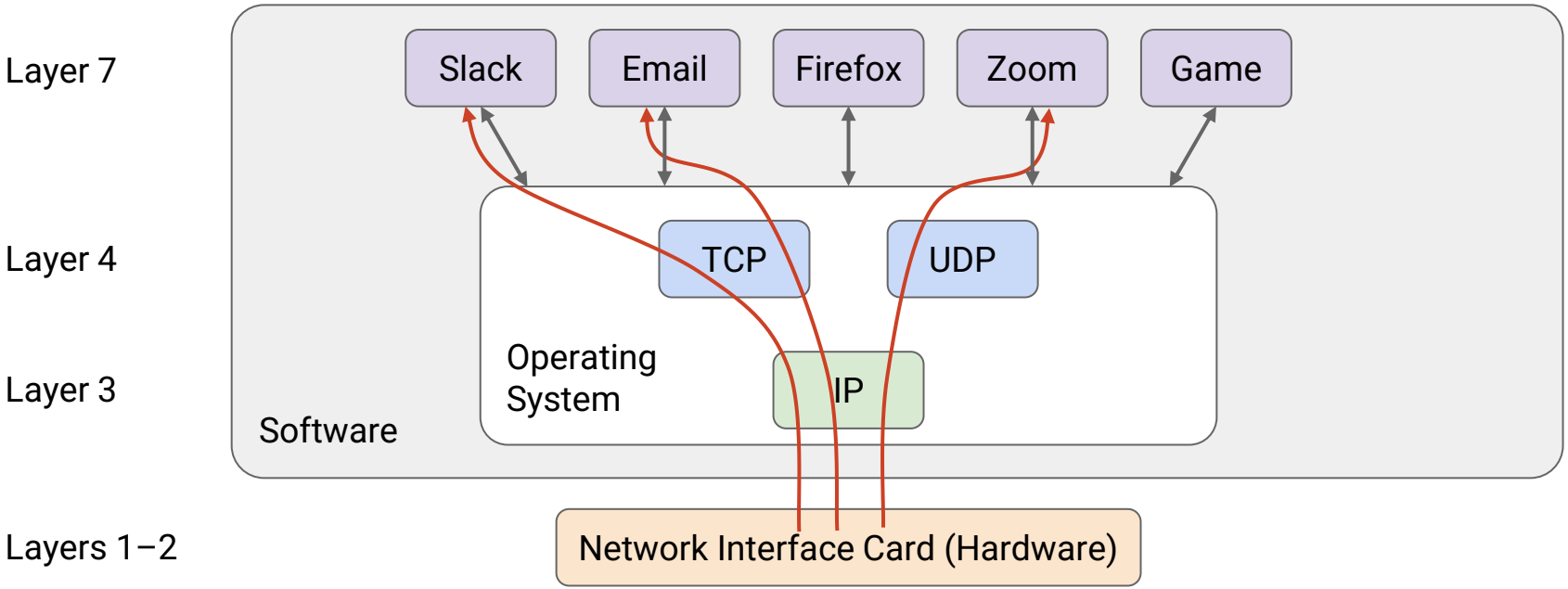
Layer 7 is the applications running in software.



Implementing Layers in the End Host

Demultiplexing helps the operating system pass packets to the correct application.

Logical ports identify the attachment point between the application and the OS.



End-to-End Principle

Lecture 2, Spring 2026

Internet Design Principles

- Architecting the Internet
- Narrow Waist, Demultiplexing
- **End-to-End Principle**

Designing Resource Sharing

- Statistical Multiplexing
- Circuit vs. Packet Switching
- Which is Better?
- A Brief History

The End-to-End Question

Recall: Layer 3 (Internet) is best-effort.

- Routers implement Layers 1–3 only.
- Only end hosts implement Layer 4 (reliability).

Why did we choose this design?

Should we implement reliability in the network?

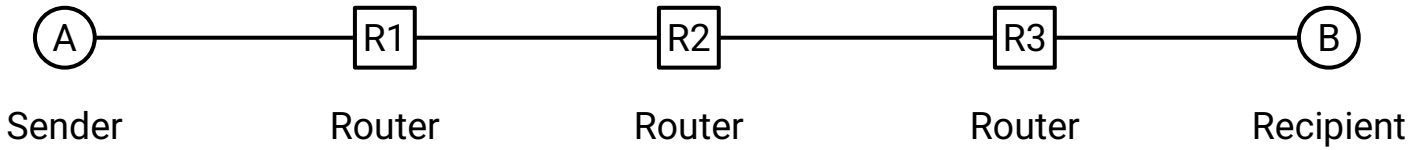
The end-to-end principle will help us answer these questions.

- Guides the debate about what functionality the network does or doesn't implement.

Running Example – Simple Reliability Protocol

We haven't discussed Layer 4 protocols yet, so let's use a super-simple protocol.

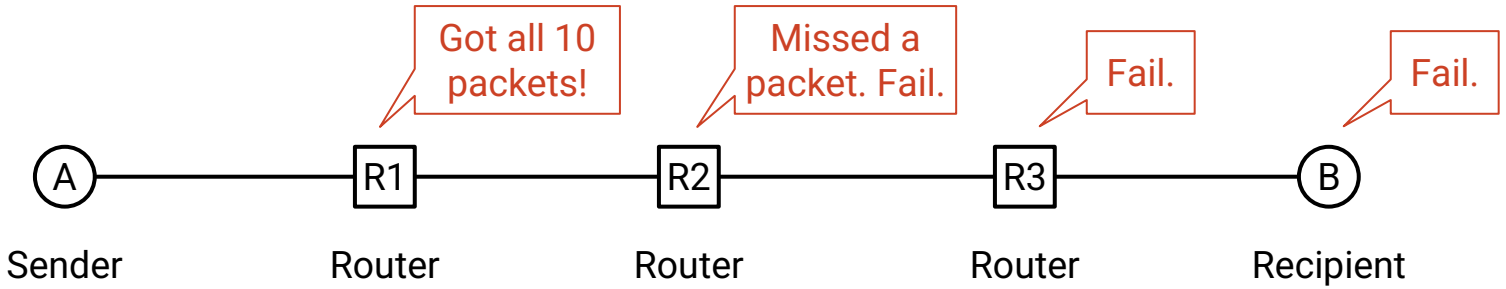
- Alice wants to send 10 packets to Bob (in any order).
- Alice numbers the packets 1 through 10 and sends them.
- Bob can either:
  - Receive all 10 packets and declare success, or
  - Detect that some packets were lost, and declare failure.
  - Bob cannot declare success when packets are lost.



Solution 1 – Reliability in the network:

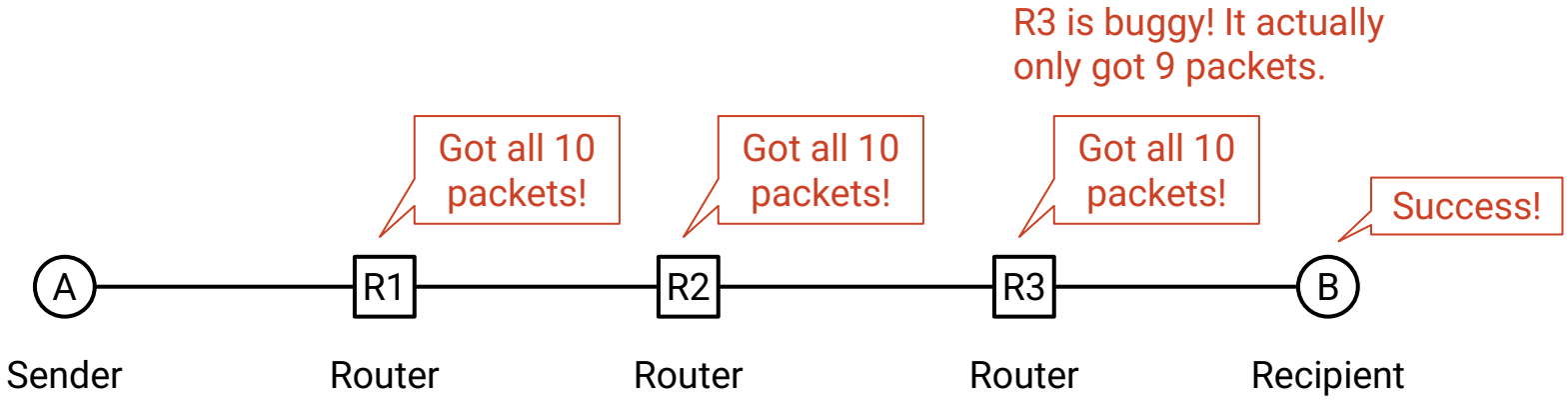
- Each router checks if it got all 10 packets.
- If success, send the 10 packets to the next hop.
- If failure, report the failure to the next hop.

Bob trusts whatever the last packet says. He doesn't count the packets himself.



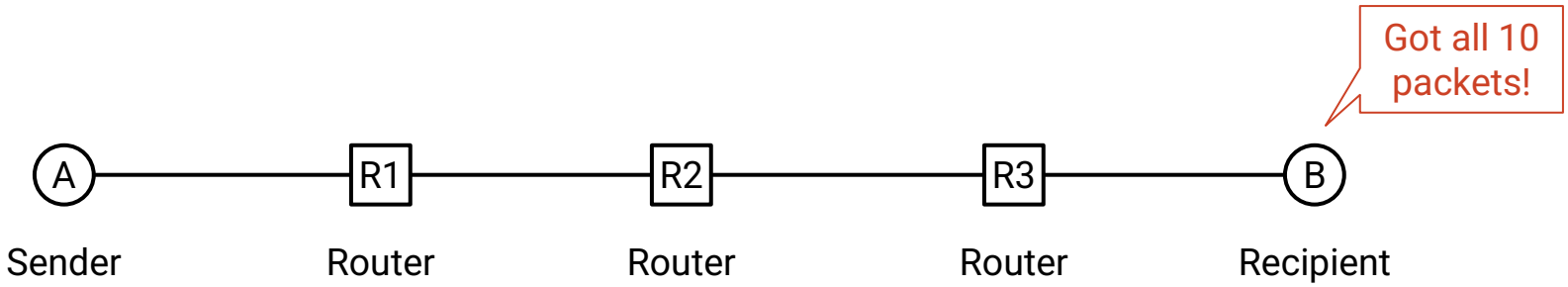
Surprising fact: Solution 1 cannot guarantee correctness.

- Suppose R3 is buggy and always reports success.
- Bob doesn't check, so he trusts R3's report...even if it's wrong.



Solution 2 – Reliability at end hosts:

- Routers are best-effort. They might drop packets (and not report it).
- Bob checks if he got all 10 packets.



Which Solution is Better?

Problem with Solution 1 (only routers check) : The end host (Bob) had to trust the network for correctness.

- If the reliability code in the network is buggy, there's nothing Bob can do.

Solution 2 (only end hosts check) can be correct by itself, where Bob only had to rely on himself for correctness.

- If the reliability code is buggy, Bob has the power to fix it.

Solution 2 is strictly better!

- End hosts checking alone is already sufficient.
- Router checks in solution 1 are unnecessary: Extra complexity for the network.

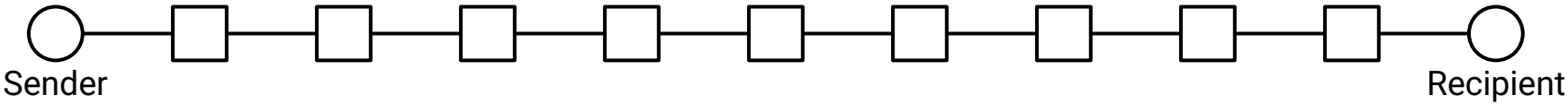
End-to-end principle: Certain application features (e.g. reliability) must be implemented at the end host for correctness.

Breaking the End-to-End Principle for Performance

The end-to-end principle can be relaxed.

- Could implement reliability in the network as a performance optimization.
- Must be done *in addition* to end-to-end checks, for correctness.

Example: Links can send duplicate packets to improve/optimize network performance. Crrectness is guaranteed by the end-to-end principle regardless of the network performance (by retransmission of lost packets)



10 links, 10% failure rate per link = ~**65%** end-to-end failure probablity. (Success rate per link = 1−0.1 = 0.9, so end-to-end success probability (when all 10 links must succeed): $0.9^{10} \approx 0.35$, hence End-to-end failure rate is 1-0.35 = 0.65.)

If each link sends 2 extra copies of every packet (3 total): 0.1% failure rate per link, ~**1%** end-to-end failure rate. (A link only fails if all 3 copies fail. Single-copy failure probability: q = 0.1. Failure rate per link when all 3 copies fail is: q^3 = 0.001. Success rate per link = 1− 0.001 = 0.999, so end-to-end success rate is: 0.999^{10}= 0.990, hence end-to-end failure rate is 1 - 0.99 = 0.01.)

Designing Resource Sharing: Statistical Multiplexing

Lecture 2, Spring 2026

Internet Design Principles

- Architecting the Internet
- Narrow Waist, Demultiplexing
- End-to-End Principle

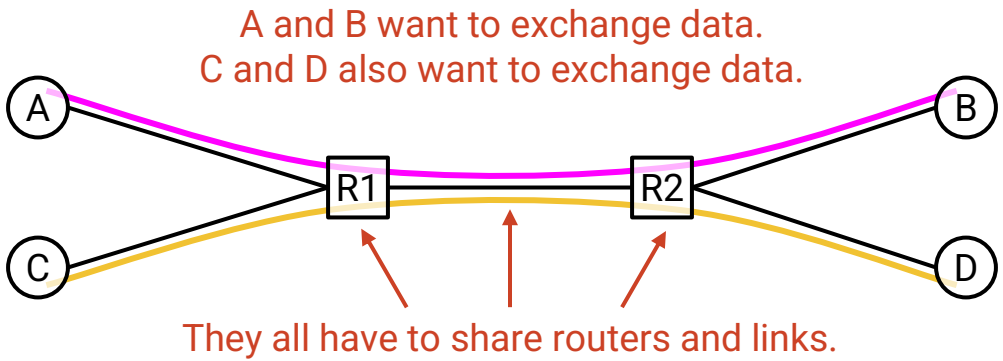
Designing Resource Sharing

- **Statistical Multiplexing**
- Circuit vs. Packet Switching
- Which is Better?
- A Brief History

Sharing Network Resources

The network must support many simultaneous flows.

- Recall: A flow is a stream of packets sent between two end hosts.
- This means network resources are shared between end hosts.

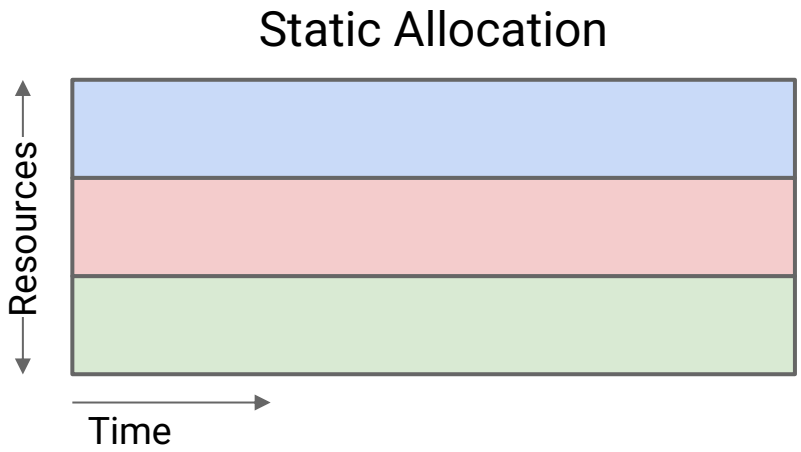


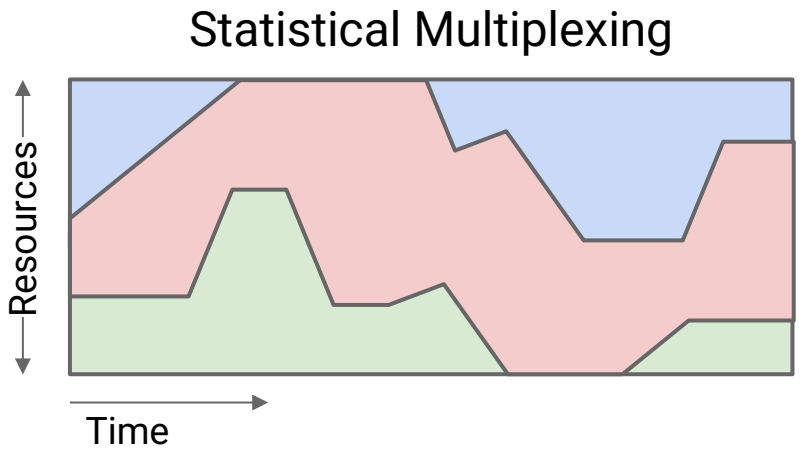
Sharing Network Resources

Two ways to allocate resources to users:

- Static allocation: Give a fixed amount to each user.
- Statistical multiplexing: Dynamically allocate to users based on their demand.
  - Example: Your computer allocates CPU to apps based on demand.

Network resources are **statistically multiplexed**.





Statistical Multiplexing is More Efficient

Statistical multiplexing (dynamic) is more efficient than static allocation (fixed).

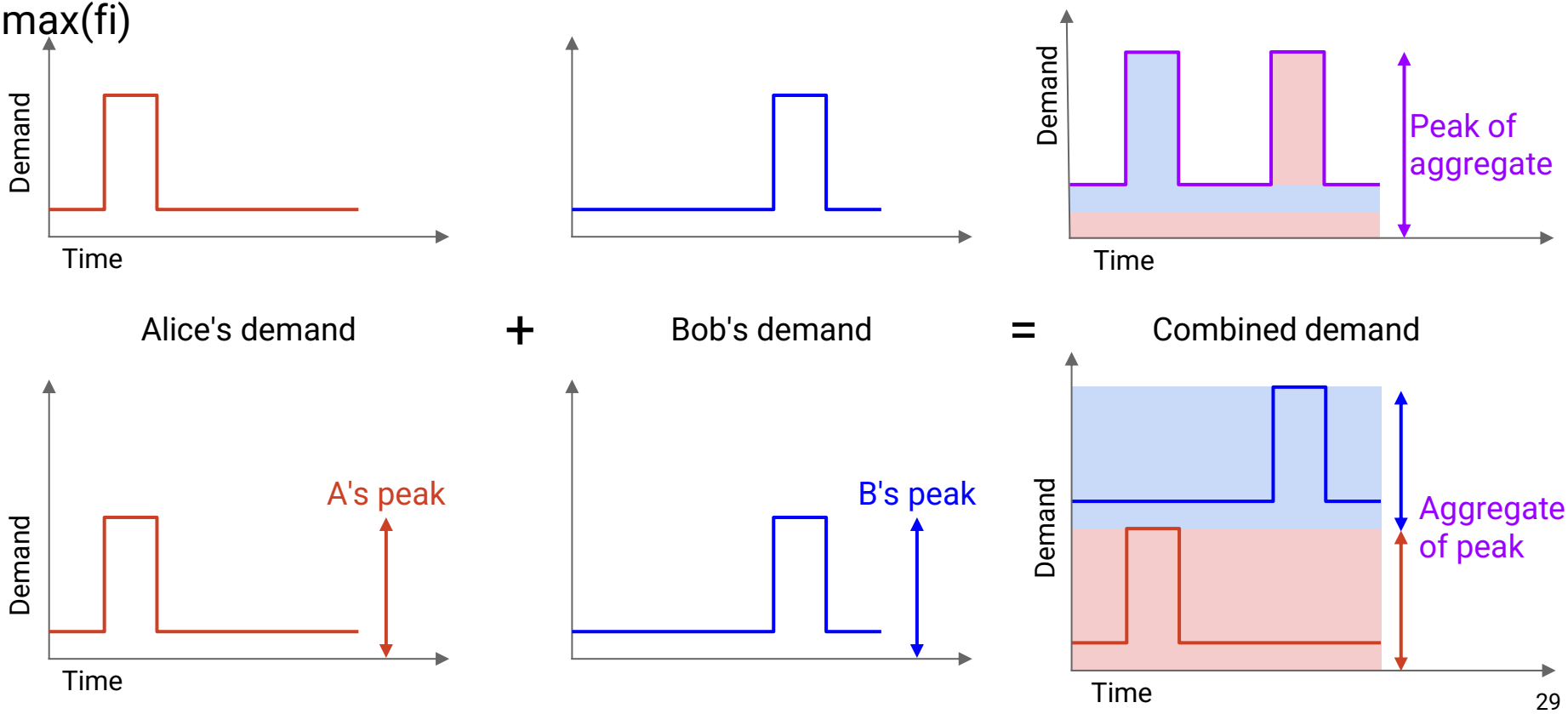
- Fixed: You have to give everyone enough for their peak demand.
- Dynamic: Give a user more when their demand peaks.

Example:

- Alice needs 10 in the morning, and 2 all other times.
- Bob needs 10 at night, and 2 all other times.
- Fixed: Alice and Bob each get 10 at all times.
  - We need 20 to satisfy demand.
  - Alice's 10 is wasted most of the time. Same for Bob.
- Dynamic: Give each user 10 at their peak time, and 2 at other times.
  - We only need 12 to satisfy demand!

Statistical Multiplexing is More Efficient

In summary: peak of aggregate demand < aggregate of peak demands. max(Σ fi) < Σ max(fi)



Circuit Switching vs. Packet Switching

Lecture 2, Spring 2026

Internet Design Principles

- Architecting the Internet
- Narrow Waist, Demultiplexing
- End-to-End Principle

Designing Resource Sharing

- Statistical Multiplexing
- **Circuit vs. Packet Switching**
- Which is Better?
- A Brief History

Circuit Switching vs. Packet Switching

There are 2 canonical designs for implementing statistical multiplexing:

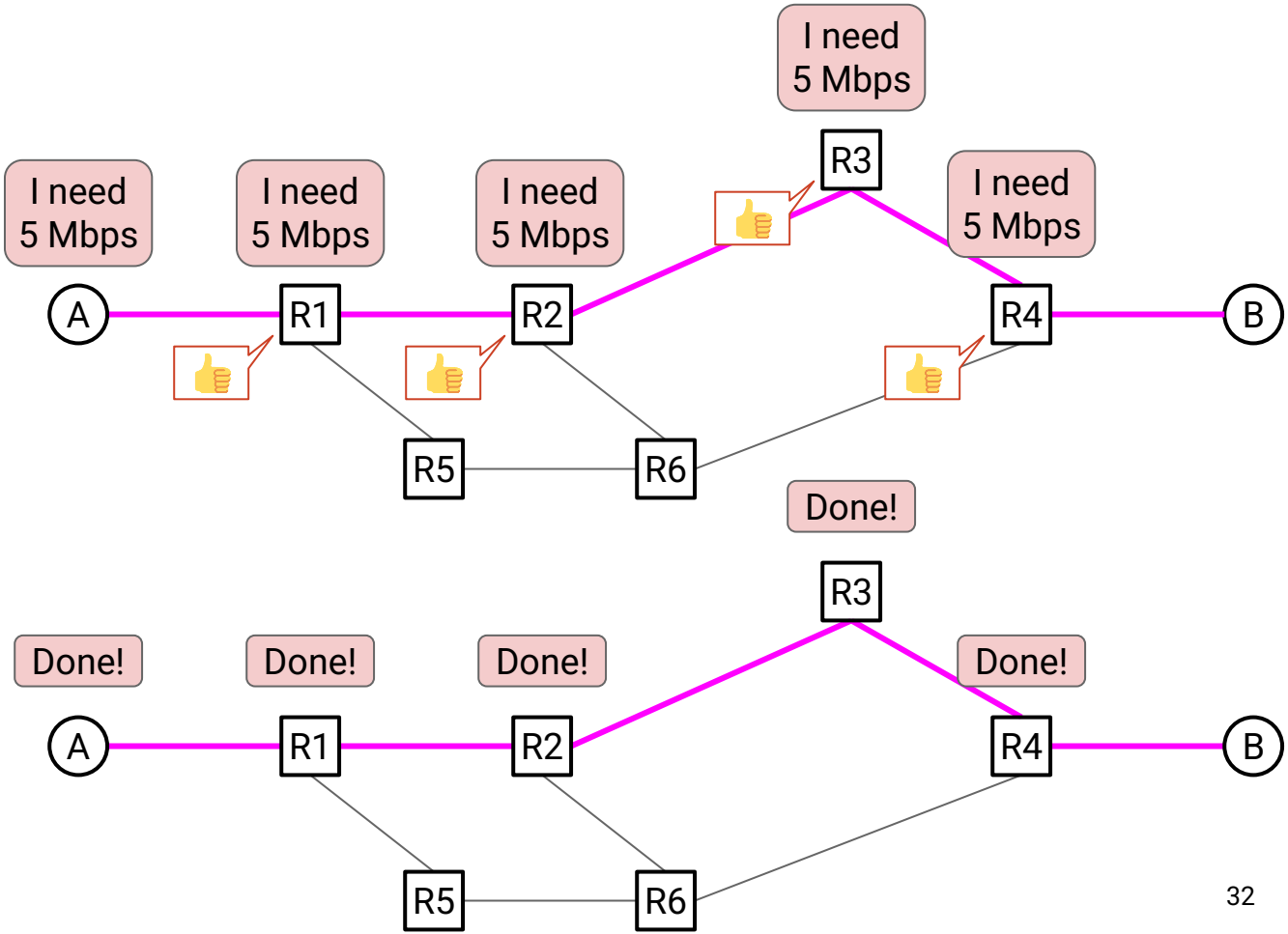
- **Reservations** via circuit switching:
  - At start of connection, end-hosts explicitly request and reserve resources.
  - During connection, use the reserved resources to send packets.
  - At end of connection, release resources.
- **Best-effort** via packet switching:
  - Just use the resources (send packets) and hope for the best.

Analogy: In a restaurant, reservations vs. first-come, first-serve.

Circuit Switching

Reservations via circuit switching: Reserve capacity for the connection.

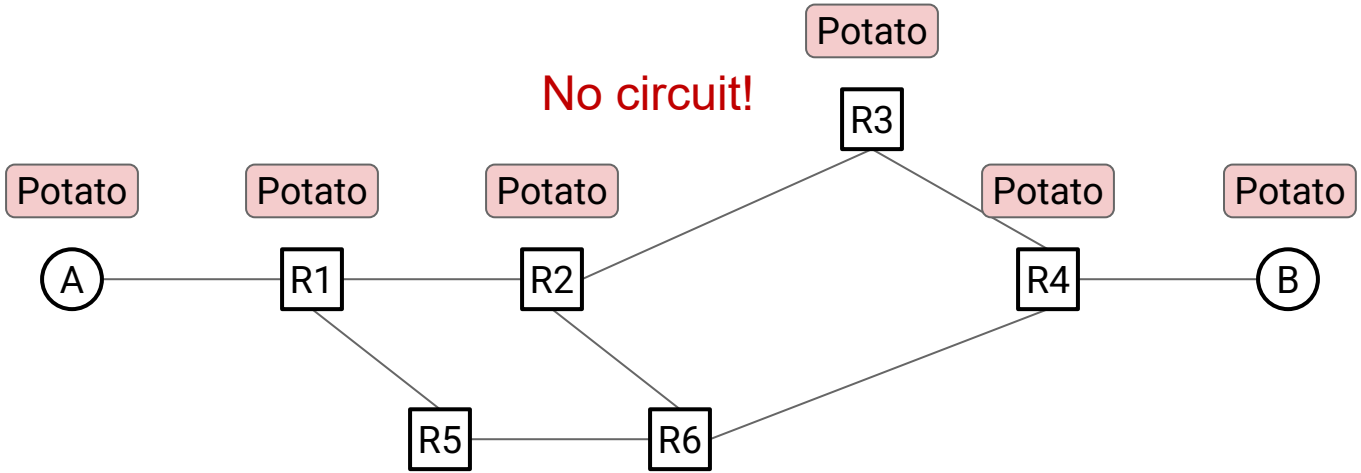
- A sends a reservation to B. Along the way, routers hear the reservation and allocate resources. Routers establish a circuit, and A can start sending data.
- When connection is done, A sends a teardown message to B. Along the way, routers see the message and free up resources.



Packet Switching

Best-effort via packet switching: Allocate resources to each packet independently.

- Each router considers the packet independently.
- Each packet in the flow is considered independently.



Circuit vs. Packet Switching: Which is Better?

Lecture 2, Spring 2026

Internet Design Principles

- Architecting the Internet
- Narrow Waist, Demultiplexing
- End-to-End Principle

Designing Resource Sharing

- Statistical Multiplexing
- Circuit vs. Packet Switching
- **Which is Better?**
- A Brief History

Circuit Switching vs. Packet Switching: Which is Better?

Which is better? We can compare along several dimensions.

1. Which one offers a better abstraction to applications?
2. Which one is more efficient at scale?
3. Which one is better at handling failures at scale?
4. Which one is easier (less complex) to implement at scale?

Which is Better? (1/4) – Application Abstraction

As a programmer, **circuit switching** is more convenient.

- You get a guarantee of reserved resources.
- More predictable and understandable behavior.
- Leads to an intuitive business model for companies.
  - Charge a user depending on what they reserve.

1. **Better abstraction to applications?**
2. More efficient at scale?
3. Better at handling failures at scale?
4. Easier (less complex) to implement at scale?

Which is Better? (2/4) – Efficiency

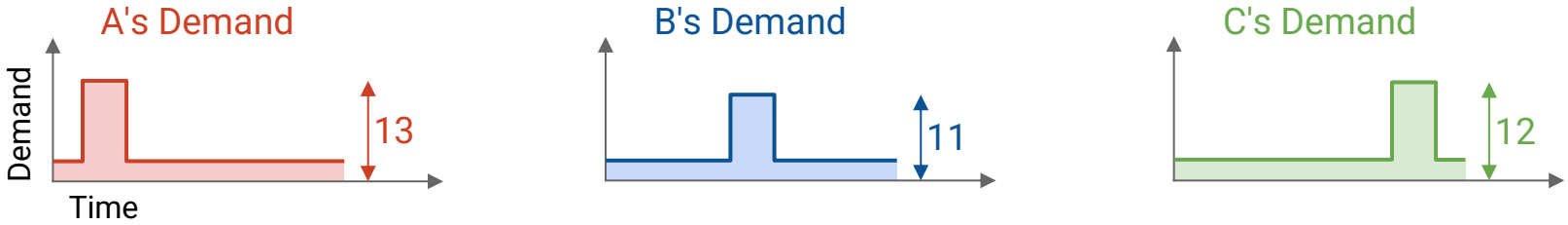
Packet switching is typically more efficient.

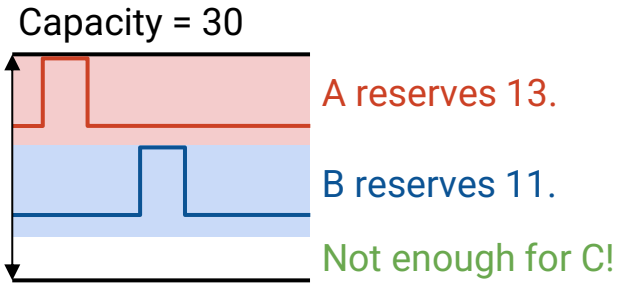
- Circuit switching takes time for setup/teardown.
  - Very inefficient if you don't have much data to send, e.g. short flows.
- Circuit switching can lead to wasted resources.
  - If I reserve 5 Mbps for an hour, I might not need 5 Mbps the whole hour.
  - Other people could be using the resources I'm not using.
  - How much better depends on the "burstiness" of the traffic sources.

1. Better abstraction to applications?
2. **More efficient at scale?**
3. Better at handling failures at scale?
4. Easier (less complex) to implement at scale?

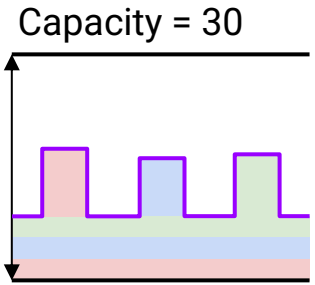
Which is Better? (2/4) – Efficiency – Packet Burstiness

Circuit switching with *bursty* traffic leads to inefficient resource allocation.





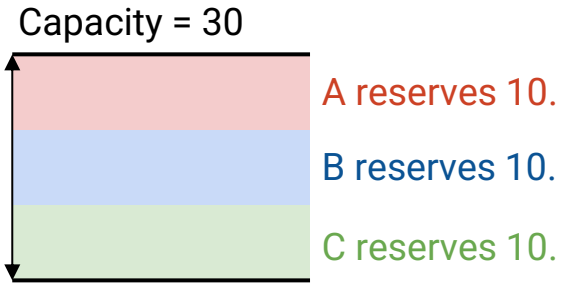
Circuit switching:
Must reject one of the flows!



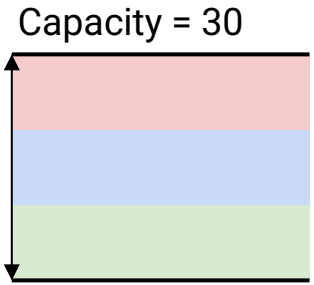
Packet switching:
All demands satisfied!

Which is Better? (2/4) – Efficiency – Packet Burstiness

If demand over time is constant, circuit and packet switching both work well.



Circuit switching:
All demands satisfied!

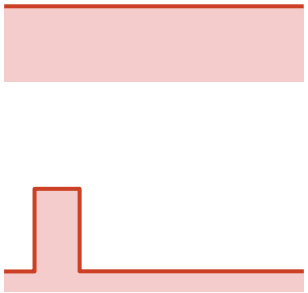


Packet switching:
All demands satisfied!

Which is Better? (2/4) – Efficiency – Packet Burstiness

Flows can be smooth or bursty.

- Characterized by the ratio between the flow's peak demand and average demand.
- **Smooth** applications have a small peak-to-average ratio.
  - Voice has a ratio of ~3:1.
  - This is why the phone network uses reservations!
- **Bursty** applications have a large peak-to-average ratio.
  - Data applications tend to be rather bursty.
  - Web browsing can have a ratio of 100:1 or more.



Which is Better? (3/4) – Handling Failure

What happens if a link or router fails?

- Network must detect failure and send packets along a different route.

Packet switching is better at handling failure.

- End hosts don't need to do anything extra.

Circuit switching requires extra work from end hosts.

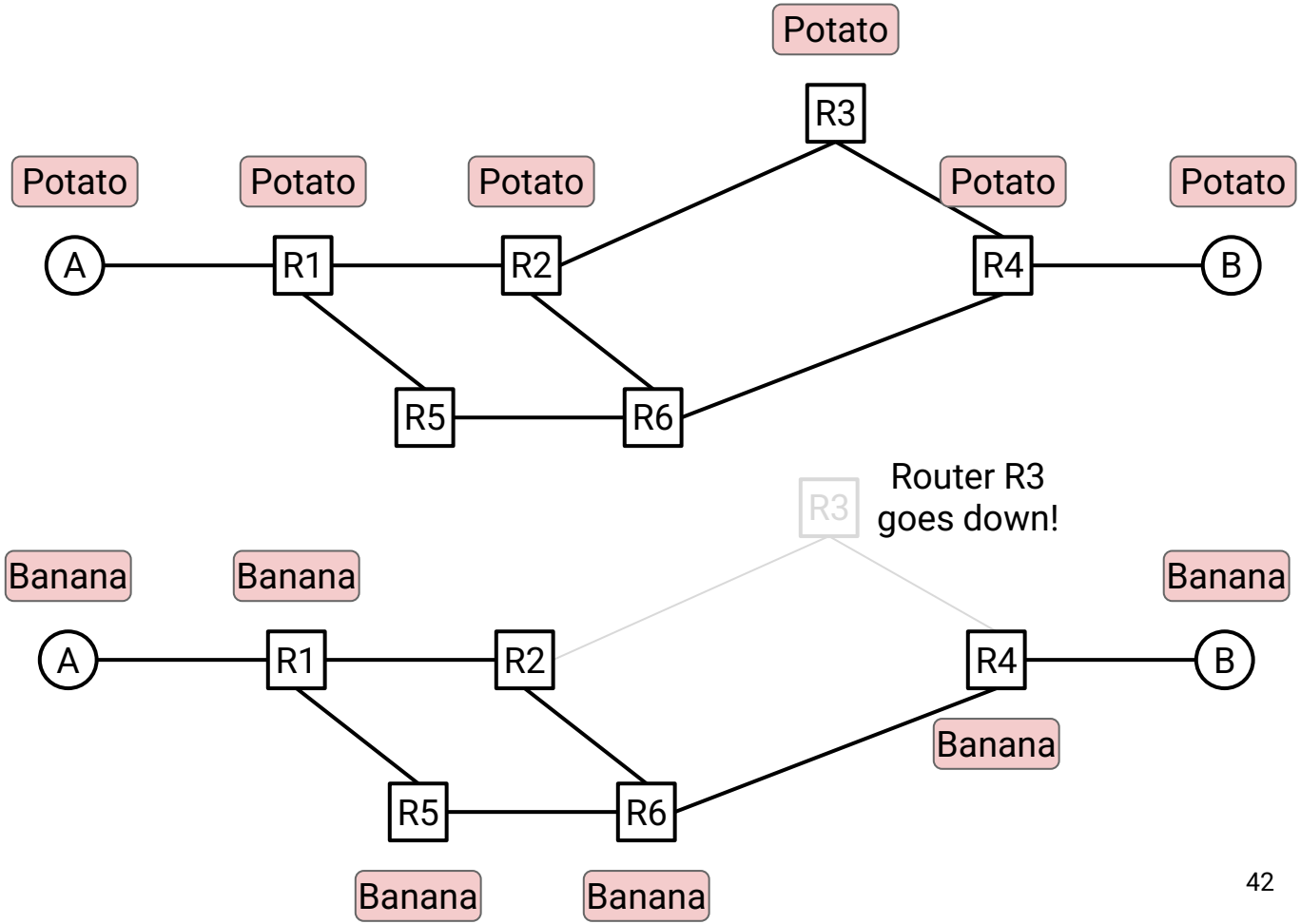
- End host must also detect failure, tear down old reservations, and send new reservation request.
- All flows using that link must redo reservations.
  - Potentially millions of flows simultaneously re-establishing reservations!

1. Better abstraction to applications?
2. More efficient at scale?
3. **Better at handling failures at scale?**
4. Easier (less complex) to implement at scale?

Which is Better? (3/4) – Handling Failure with Packet Switching

If a failure occurs in packet switching:

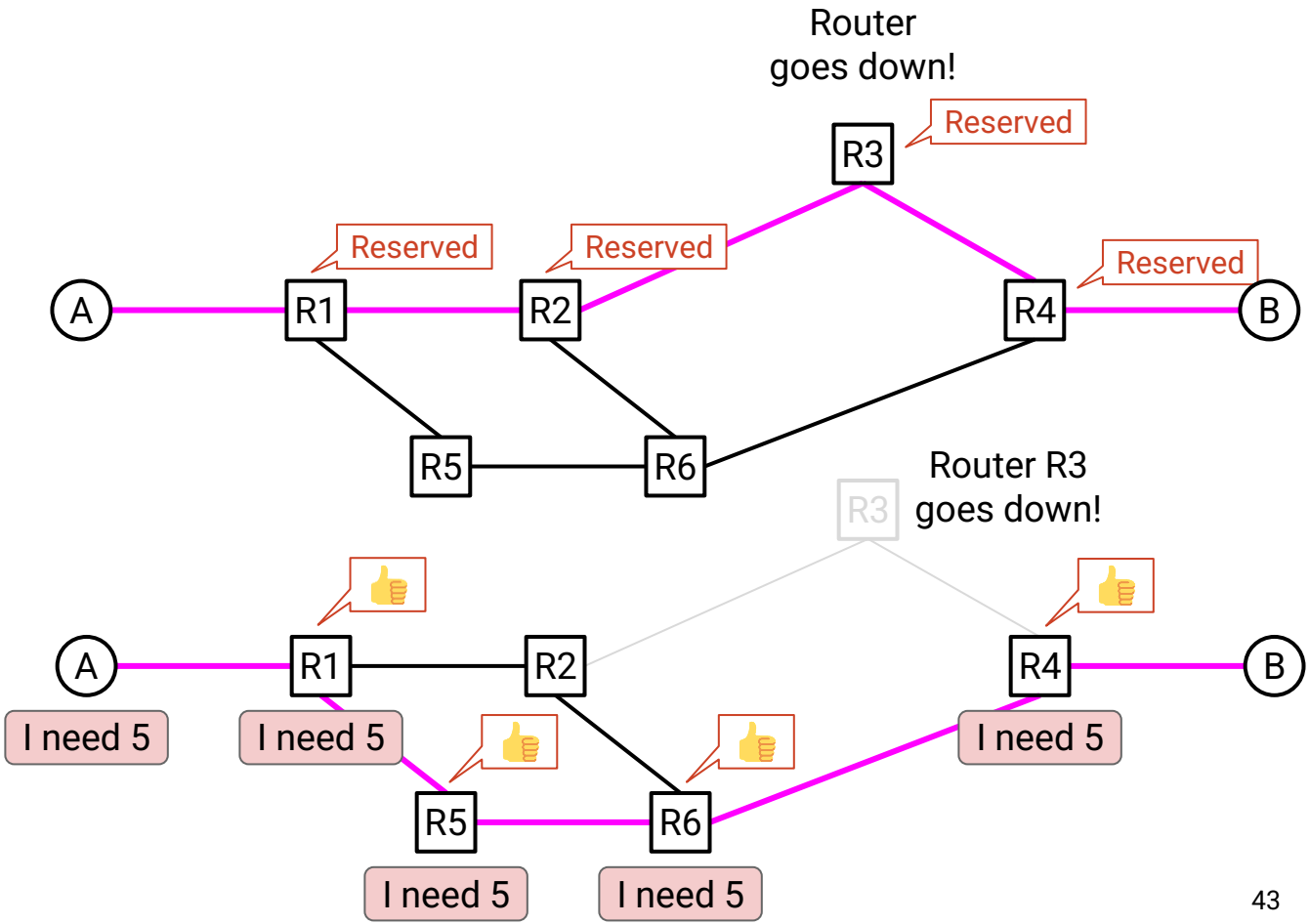
- Routers send packets along a different route.
- End hosts don't need to do anything extra.



Which is Better? (3/4) – Handling Failure with Circuit Switching

If a failure occurs in circuit switching:

- End host must tear down the circuit, and request a new reservation.
- What if the new request gets declined?



Which is Better? (4/4) – Implementation Complexity

Packet switching is easier to implement.

- Routers don't have to keep track of reservations.

Circuit switching implementation questions:

- How do all the routers know that the request was approved?
- What if the request packet gets dropped?
- What if the teardown packet gets dropped?
- What should the end host do if the request is declined?
- What if routers say: "I can't give you 5, but I can give you 3"?

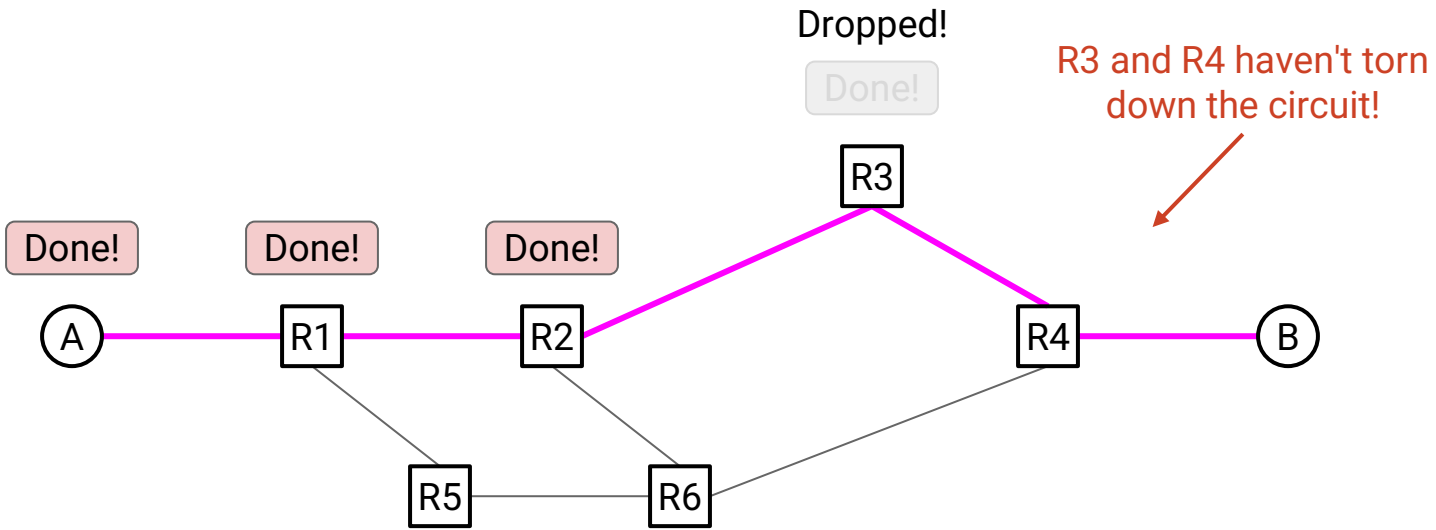
1. Better abstraction to applications?
2. More efficient at scale?
3. Better at handling failures at scale?
4. **Easier (less complex) to implement at scale?**

Which is Better? (4/4) – Implementation Complexity in Circuit Switching

Circuit switching implementation question:

What if the teardown packet gets dropped? Doesn't reach R3 and R4.

- Possible solution: Reservation expires after some time of inactivity.



Circuit Switching vs. Packet Switching: Which is Better?

1. Which one offers a better abstraction to applications? (*Circuit switching.*)
2. Which one is more efficient at scale? (*Packet switching.*)
3. Which one is better at handling failures at scale? (*Packet switching.*)
4. Which one is easier (less complex) to implement at scale? (*Packet switching.*)

Circuit switching pros:

- Reservations give applications better performance.
- Reservations are more predictable and understandable.

Packet switching pros:

- More efficient.
- Faster startup to first packet delivered.
- Easier recovery from failure.
- Simpler implementation.

Circuit vs. Packet Switching: A Brief History

Lecture 2, Spring 2026

Internet Design Principles

- Architecting the Internet
- Narrow Waist, Demultiplexing
- End-to-End Principle

Designing Resource Sharing

- Statistical Multiplexing
- Circuit vs. Packet Switching
- Which is Better?
- **A Brief History**

Circuit vs. Packet Switching in Practice

Packet switching is the default in the modern Internet.

Circuit switching used in limited settings.

- RSVP (Resource Reservation Protocol) inside a local network.
- Users can buy a dedicated link, e.g. MPLS circuits, leased lines.
  - Companies might buy dedicated links between their offices.
  - Very expensive. 10–20 times more than a normal connection.
  - Reservation requires manual set-up, and lasts for years.
  - Reservation is per-company, not per-flow.
- These settings are more constrained than reservations on the whole Internet.

Circuit vs. Packet Switching – A Brief History

Early Internet (1970s, 1980s) used packet switching.

- Well-suited for bursty file transfer applications.

Next iteration (late 1980s, 1990s) tried to move toward circuit switching.

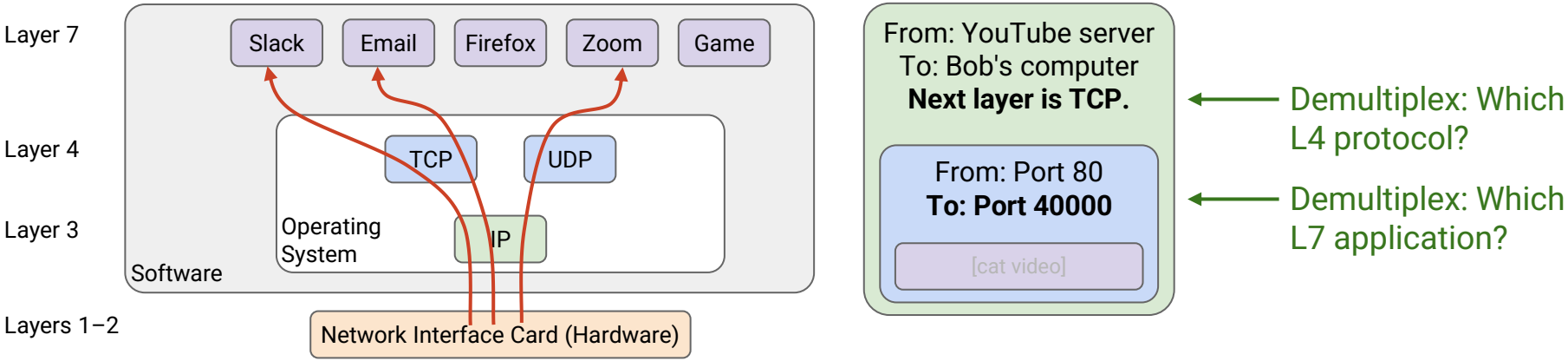
- Internet control shifted from the US government to companies.
  - Circuit switching offers a more intuitive business model.
- Envisioned smooth voice/TV applications to dominate traffic.

Spent 10+ years trying to realize the vision of circuit switching, but ultimately failed.

- Because of all the reasons we discussed.
- Bursty email and web browsing applications ended up dominating traffic.
- Users ended up adapting to packet switching realities.
  - Example: Video quality decreases if connection is poor. That’s just life.

Summary: Internet Design Principles

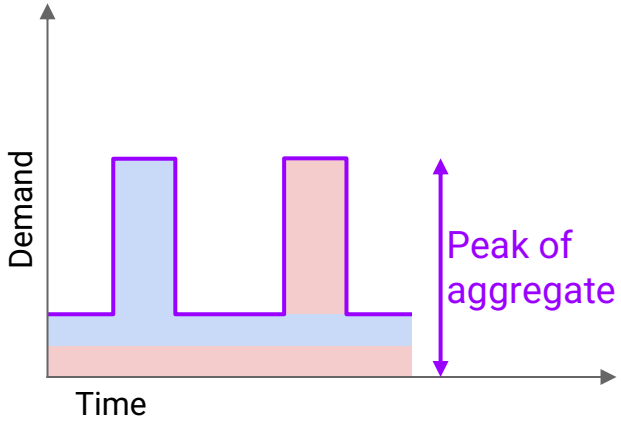
- Narrow-waist for federation: Everyone uses IP at Layer 3.
- Demultiplexing is used to pass a packet up to the correct higher-layer protocol.
- Ports are used to uniquely identify applications on an end host.



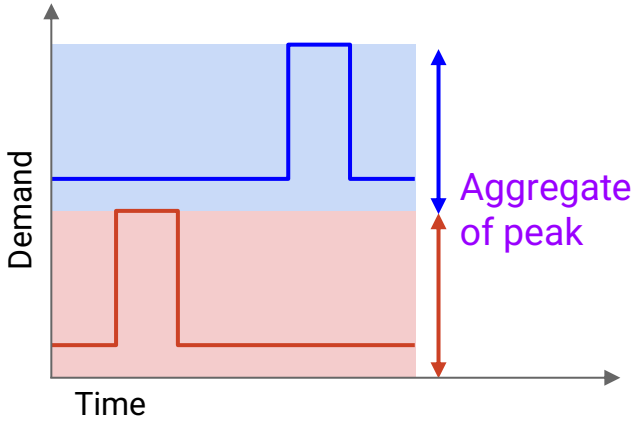
- End-to-end principle: Reliability must be implemented in end hosts (not routers) for correctness.
- Reliability could be added to routers as a performance optimization, though.

Summary: Designing Resource Sharing

- Statistical multiplexing is more efficient than static allocation.
- Circuit switching offers better abstraction. Packet switching is more efficient, better at handling failures, and easier to implement.
- Flows can be smooth or bursty. Packet switching is more efficient on bursty flows.



Statistical Multiplexing



Static Allocation