# CS168, Lecture 3

# How the Internet Works : overview

Sylvia Ratnasamy

Fall 2024

# Today

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

# Recall, from last lecture…

# Recall, from last lecture…

Two canonical approaches to sharing

# 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)

# 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 …

# 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 …

# 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

# 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

# Recall, from last lecture…

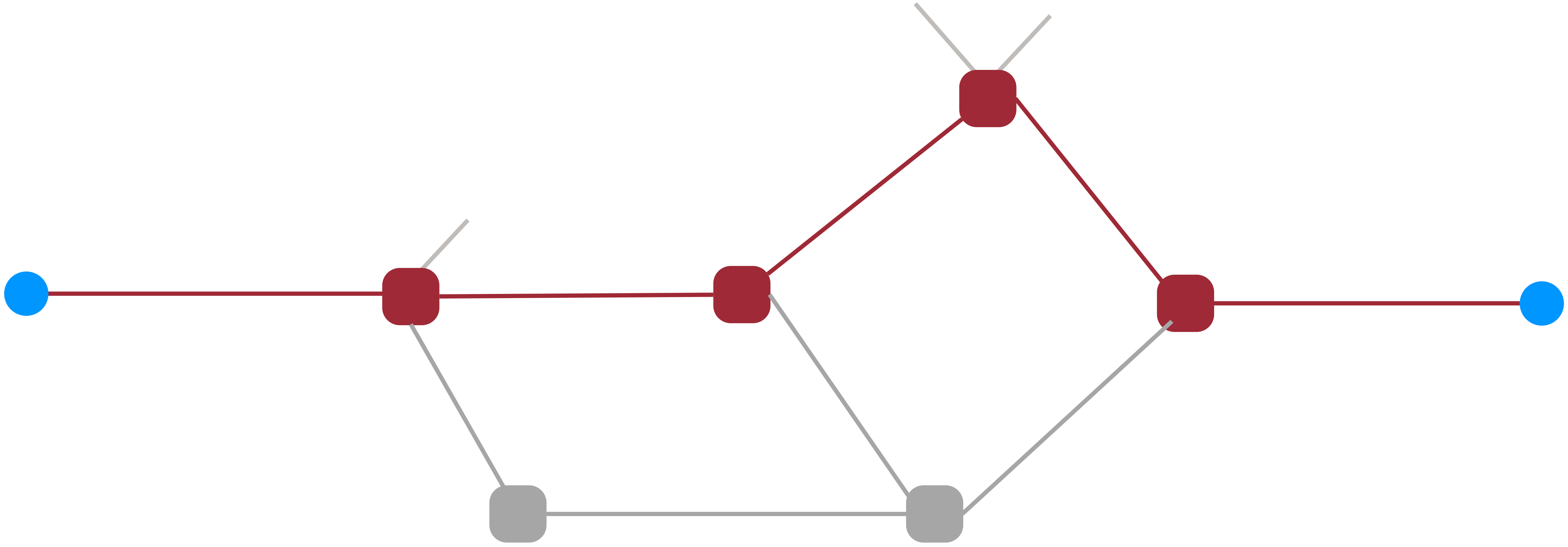
Two canonical approaches to sharing

- 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

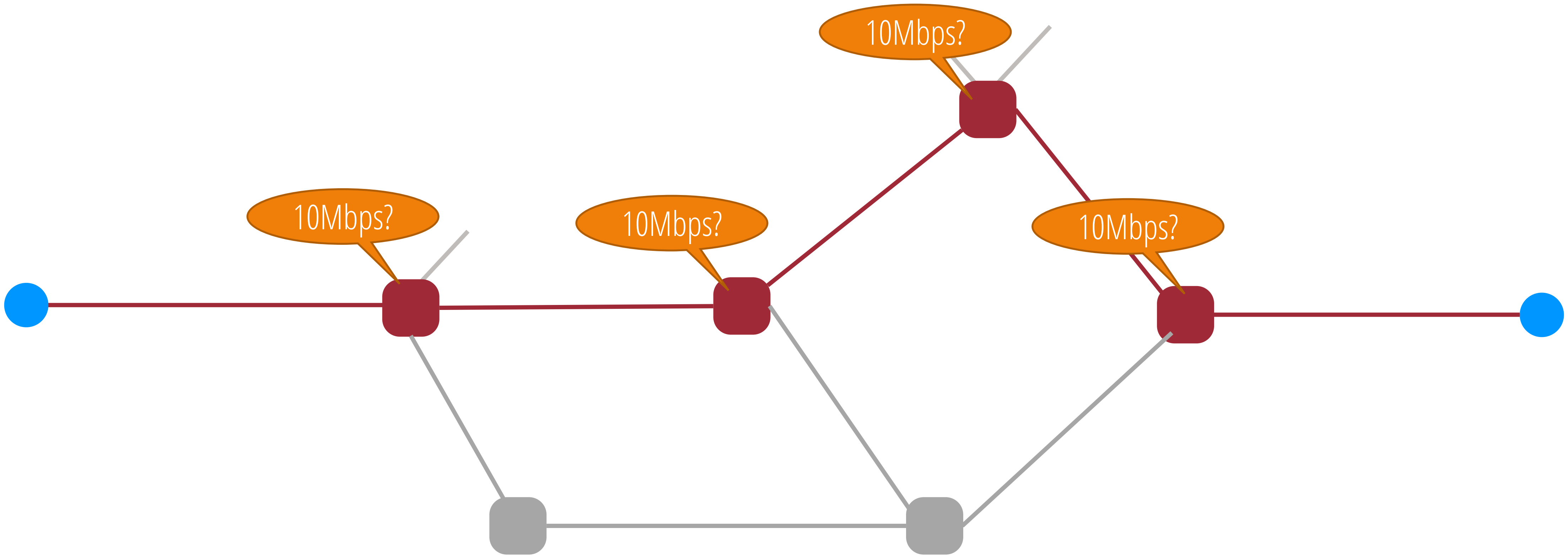
- Reservations via circuit switching
- Best-effort via packet switching

# Recall, from last lecture: circuit switching



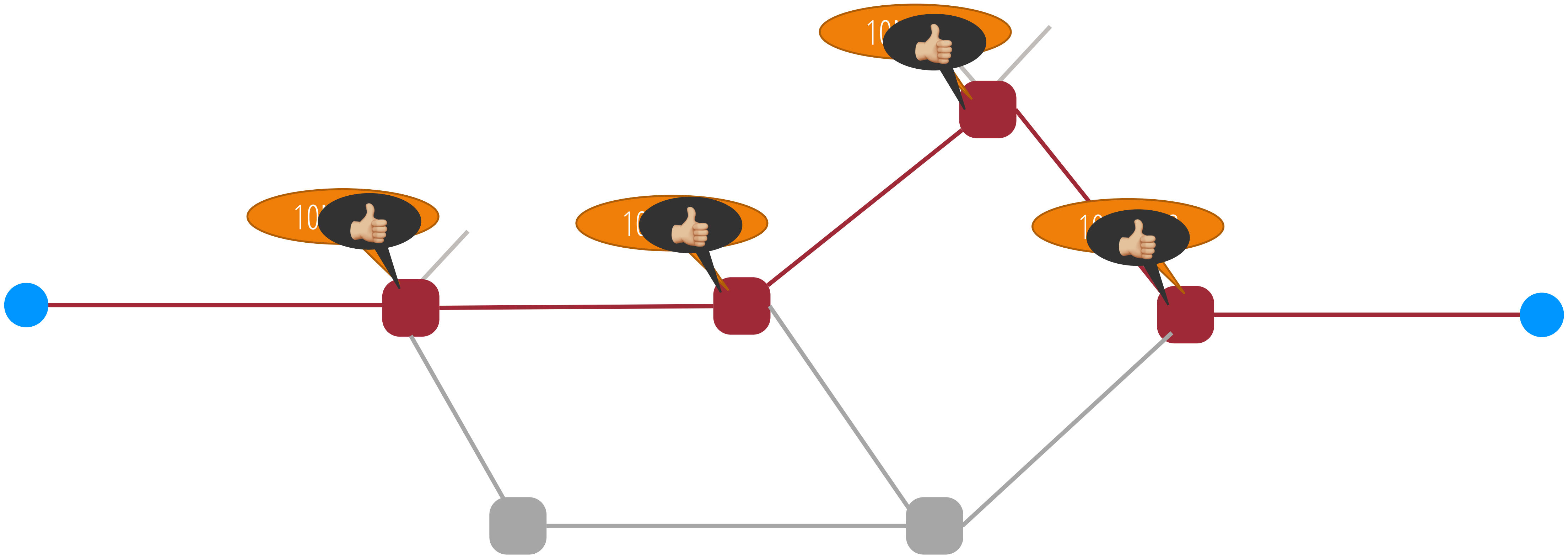
Idea: Reserve network capacity for all packets in a flow

# Recall, from last lecture: circuit switching



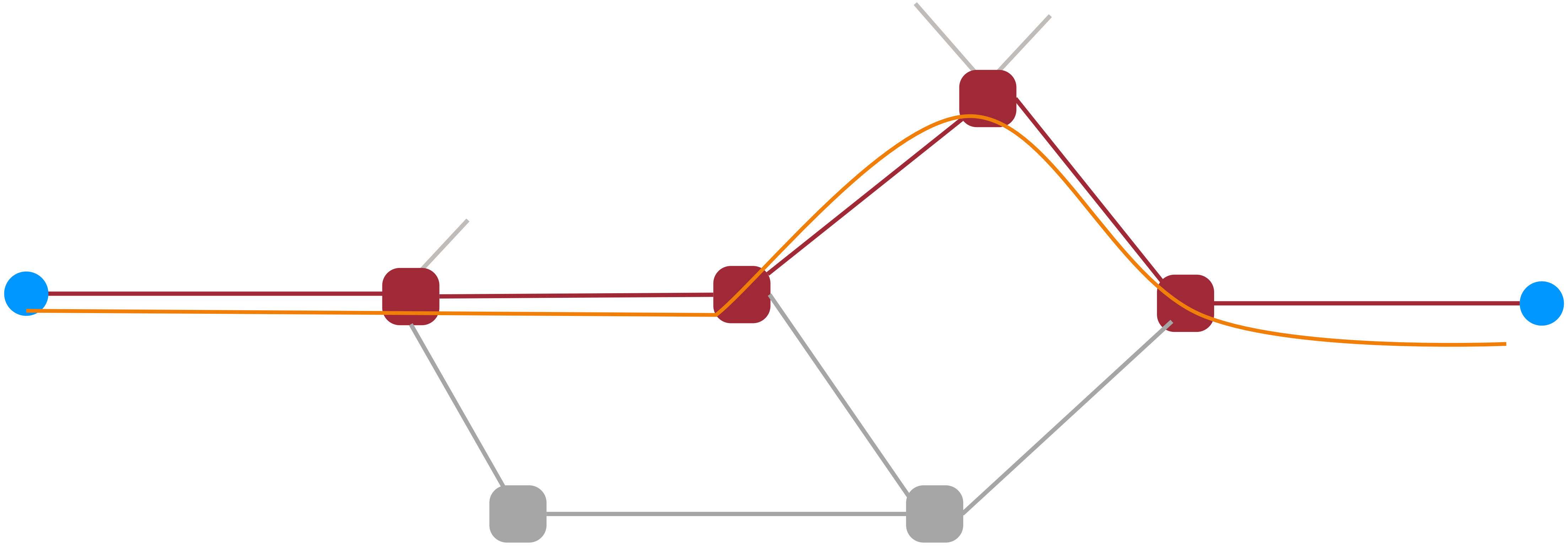

Idea: Reserve network capacity for all packets in a flow

# Recall, from last lecture: circuit switching



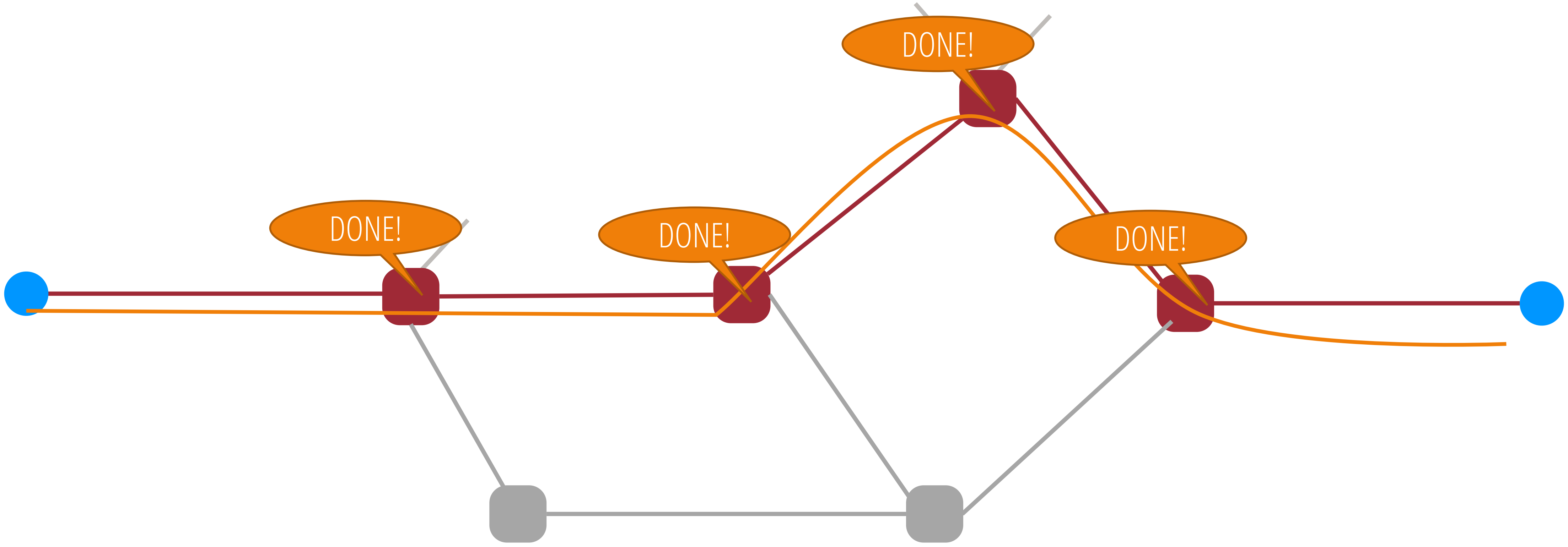
Idea: Reserve network capacity for all packets in a flow

# Recall, from last lecture: circuit switching



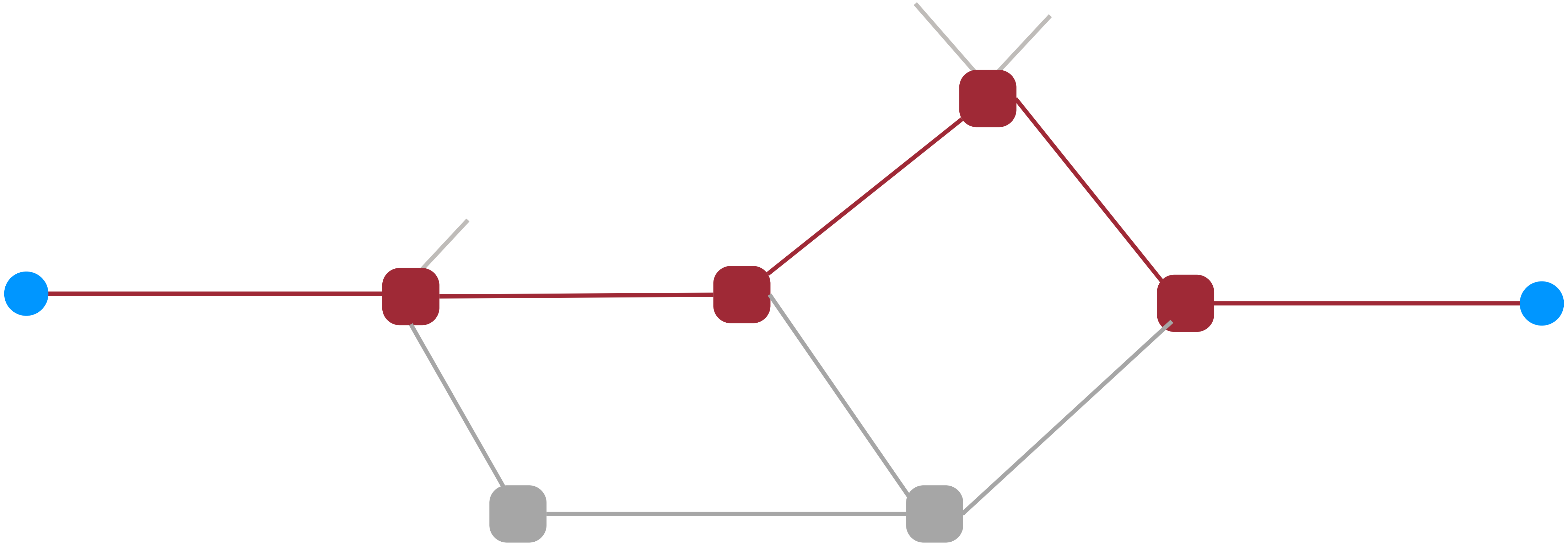
Idea: Reserve network capacity for all packets in a flow

# Recall, from last lecture: circuit switching



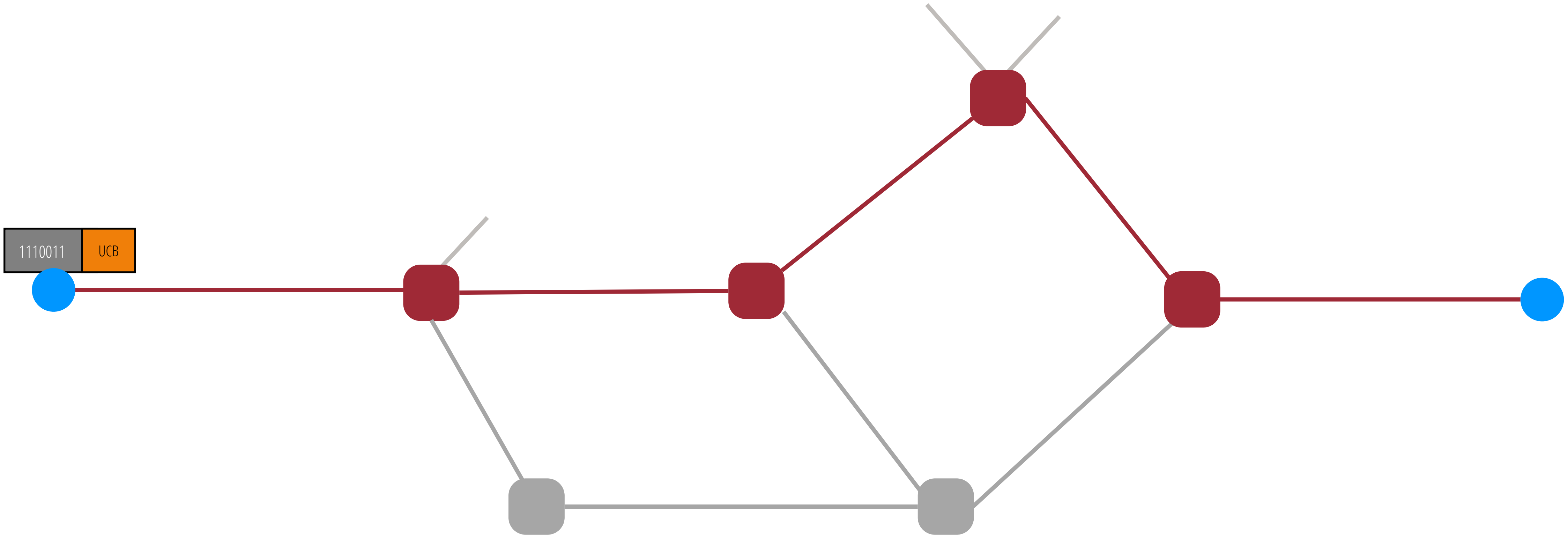

Idea: Reserve network capacity for all packets in a flow

# Recall, from last lecture: circuit switching



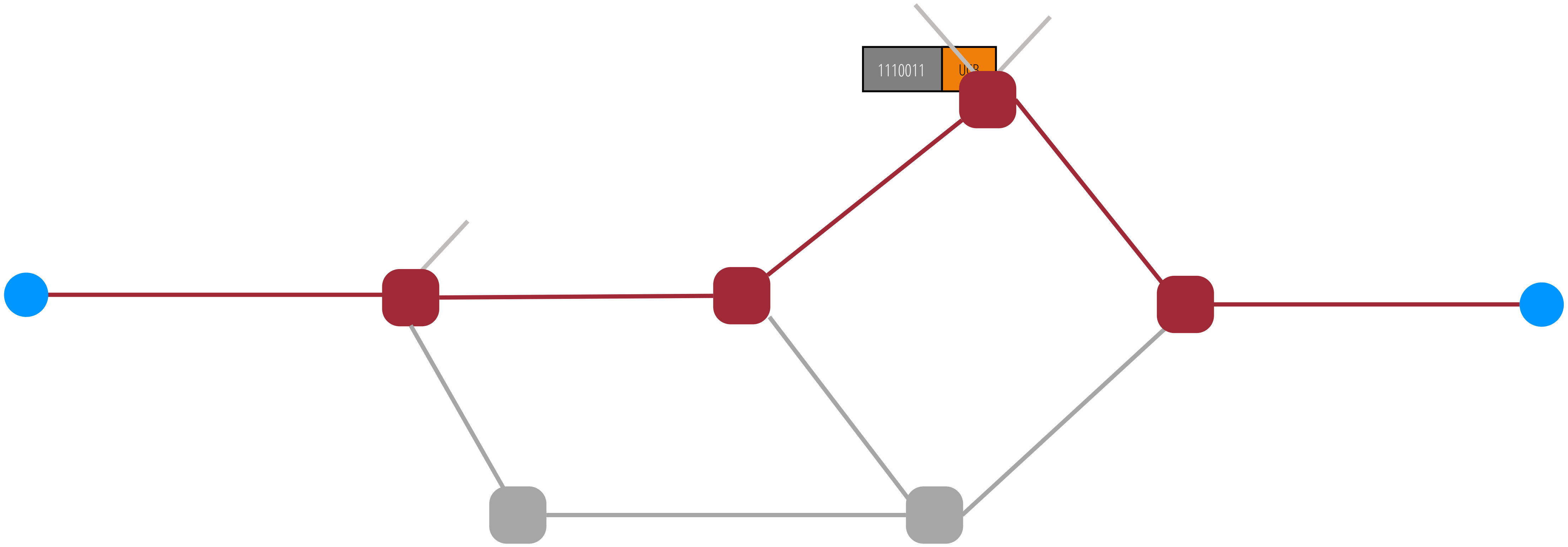
Idea: Reserve network capacity for all packets in a flow

# Recall, from last lecture: e.g., packet switching




Allocate resources to each packet independently

# Recall, from last lecture: e.g., packet switching




Allocate resources to each packet independently

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

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

# From an application viewpoint

# From an application viewpoint

- Circuits offer better application performance (reserved bandwidth)

# From an application viewpoint

- Circuits offer better application performance (reserved bandwidth)
- More predictable and understandable behavior (w/o failures)

# From an application viewpoint

- Circuits offer better application performance (reserved bandwidth)
- More predictable and understandable behavior (w/o failures)
- Also a very intuitive abstraction in support of business models

# Which makes more efficient use of network capacity?

# Which makes more efficient use of network capacity?

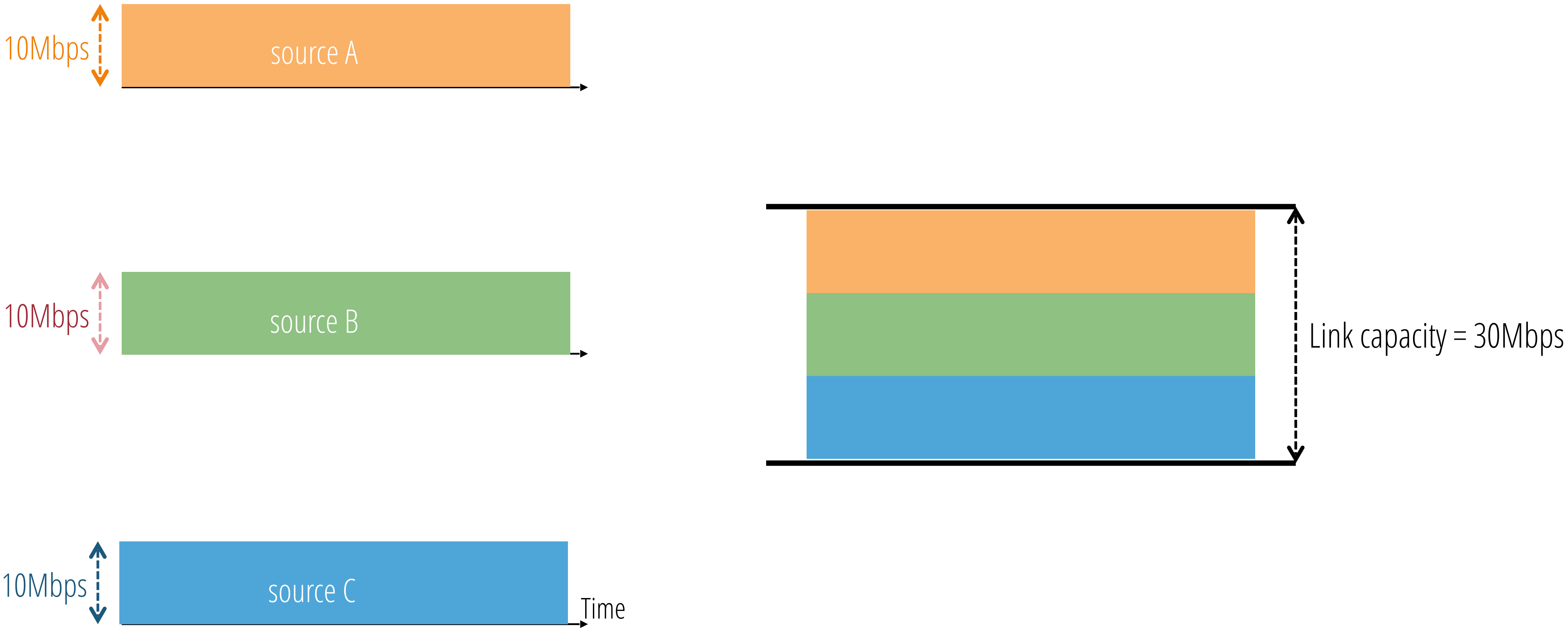
Answer:  Packet switching is typically more efficient

# Which makes more efficient use of network capacity?

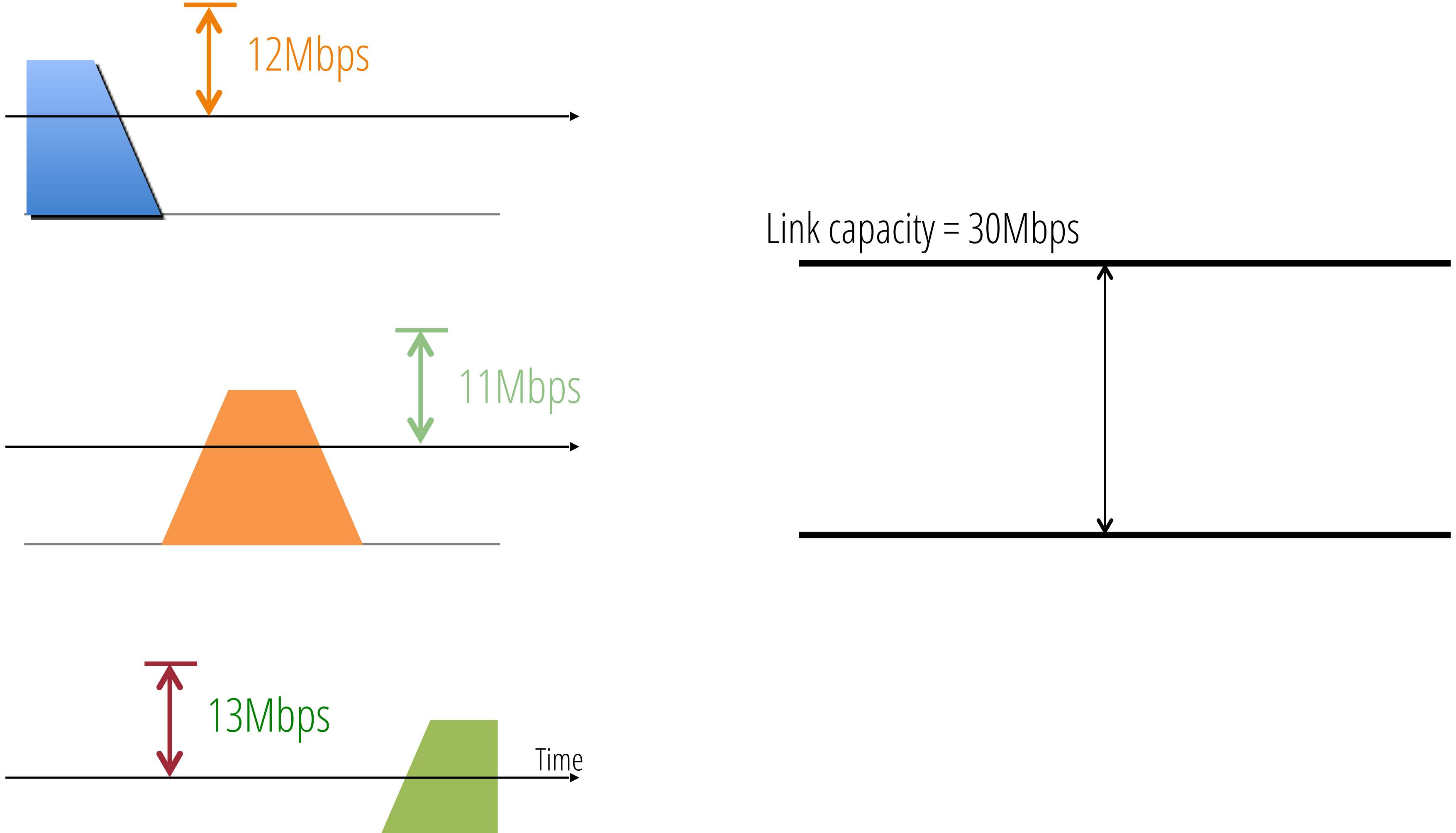
Answer:  Packet switching is typically more efficient

- But how much better depends on the “burstiness” of the traffic sources

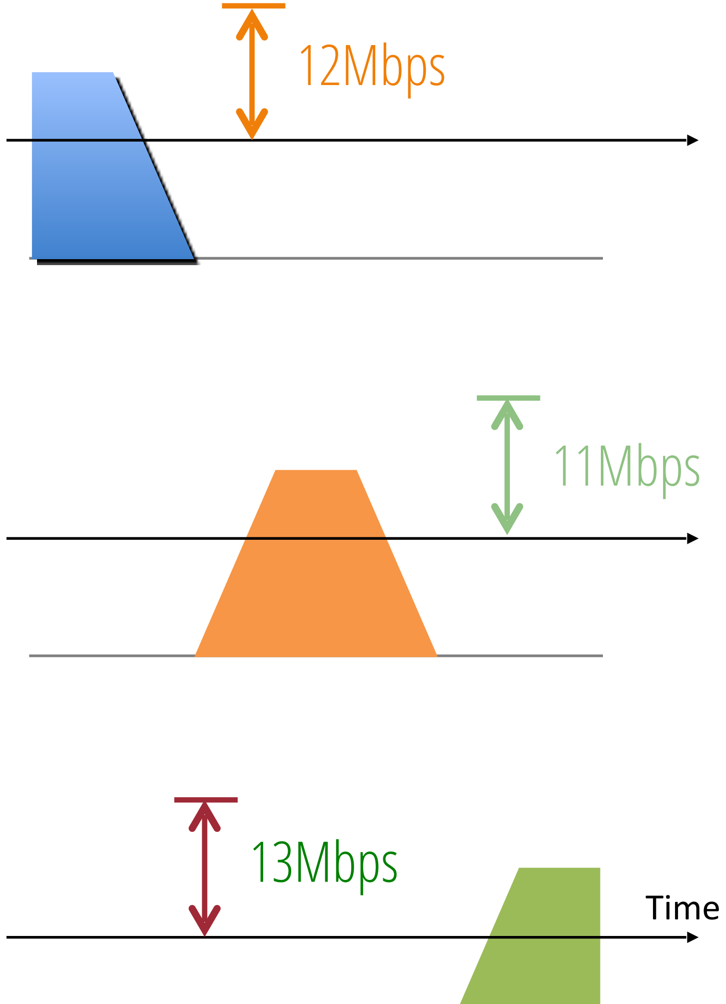
# Example#1: Three constant sources

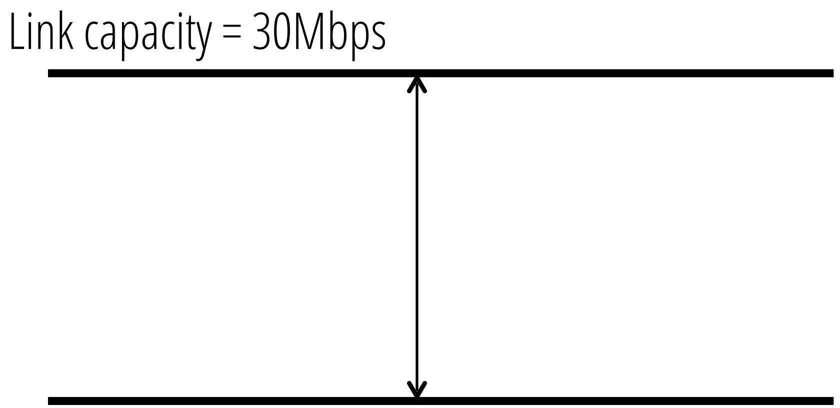


# Example#2: Three "bursty" sources



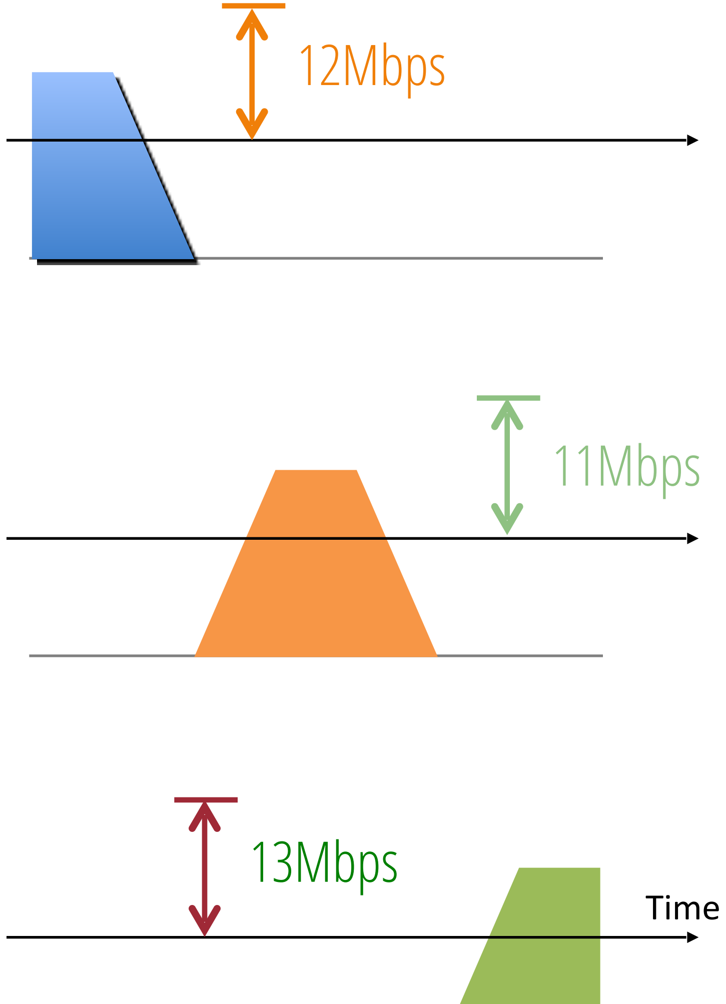
# What happens with reservations?

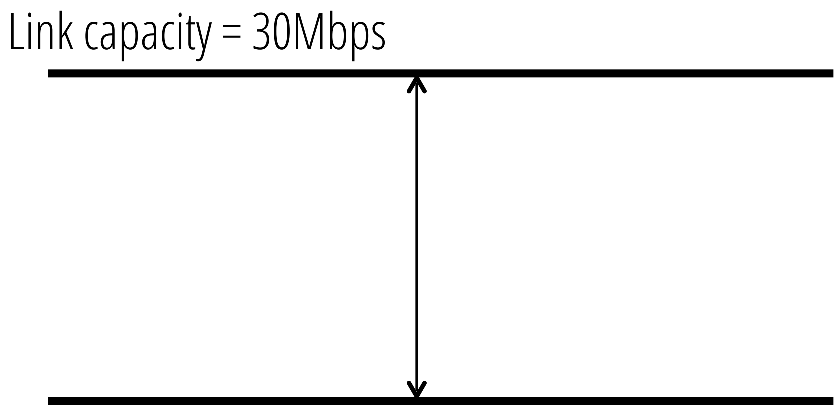


# What happens with reservations?

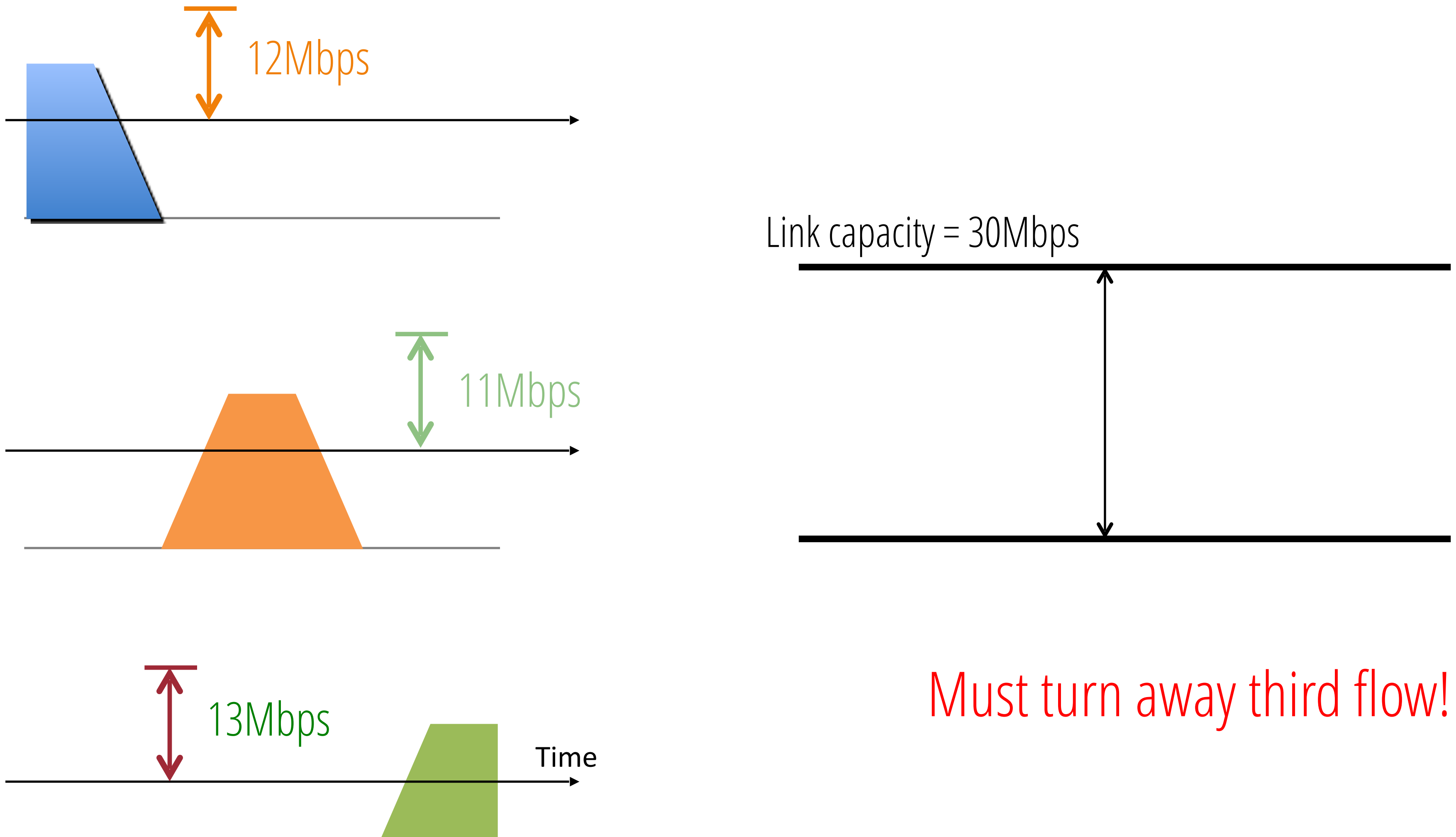
Allow two flows to reserve peak rate



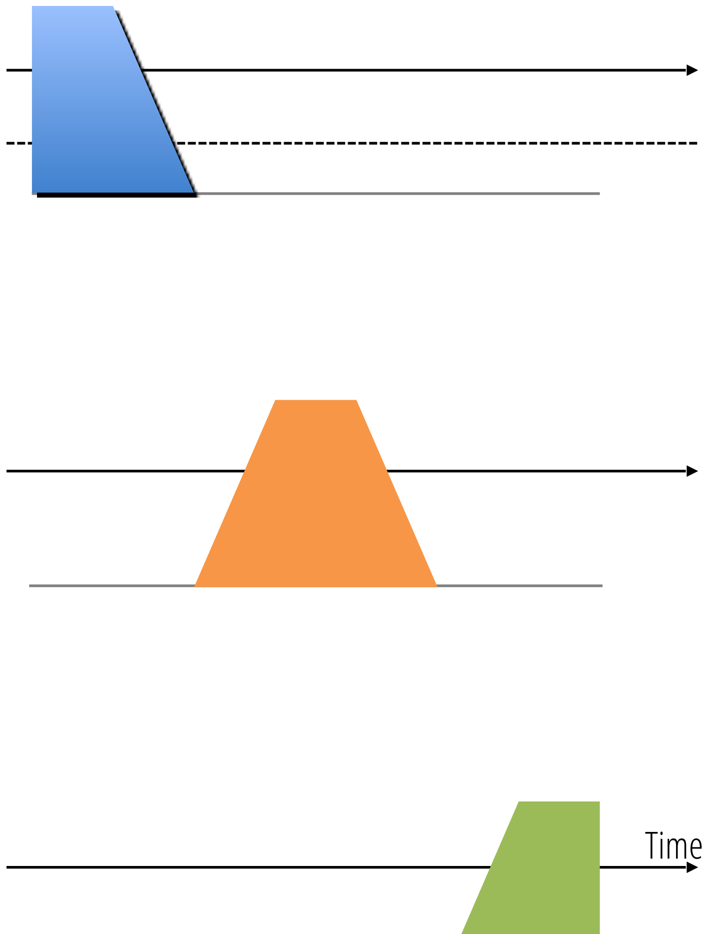
# What happens with reservations?

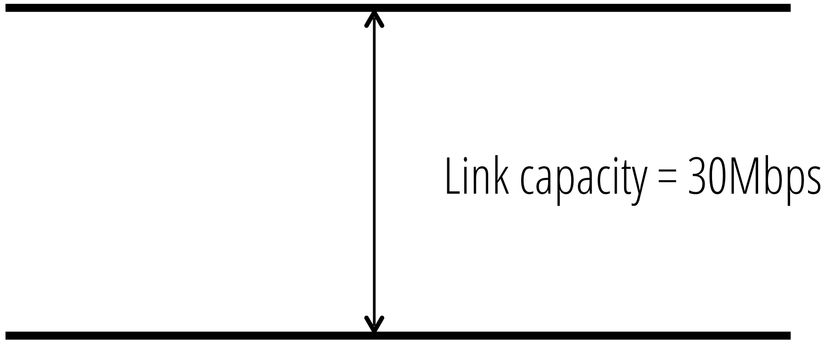
Allow two flows to reserve peak rate



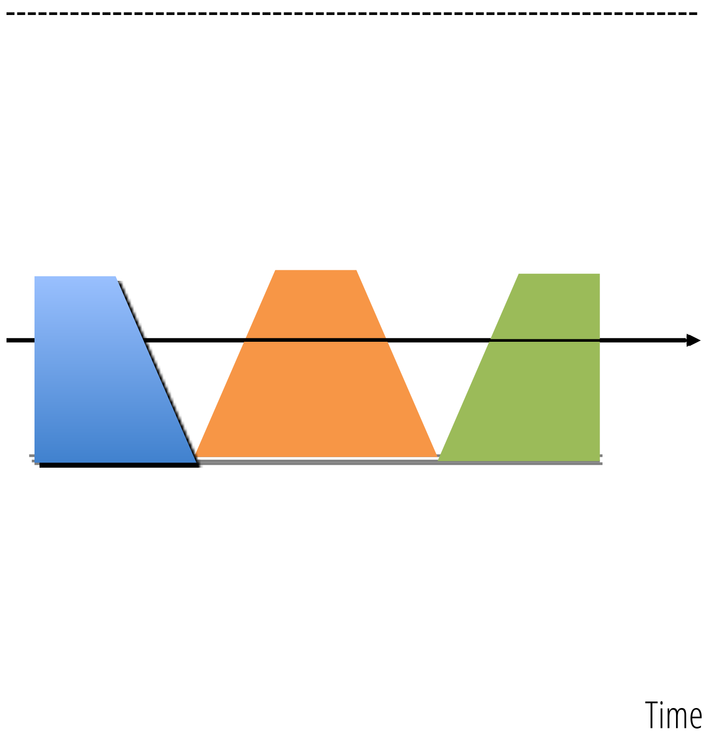

Must turn away third flow!

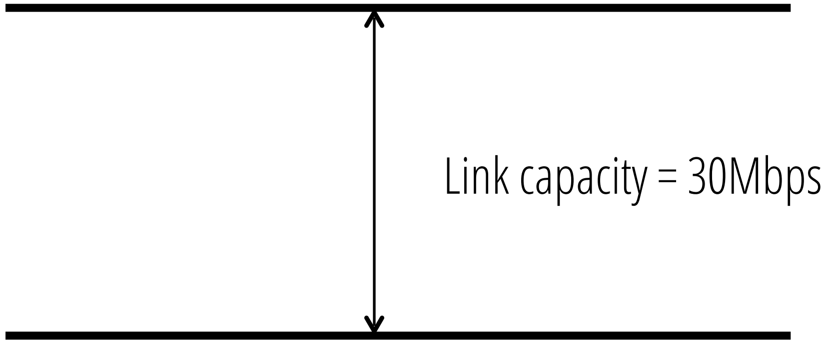
# What happens with best-effort?



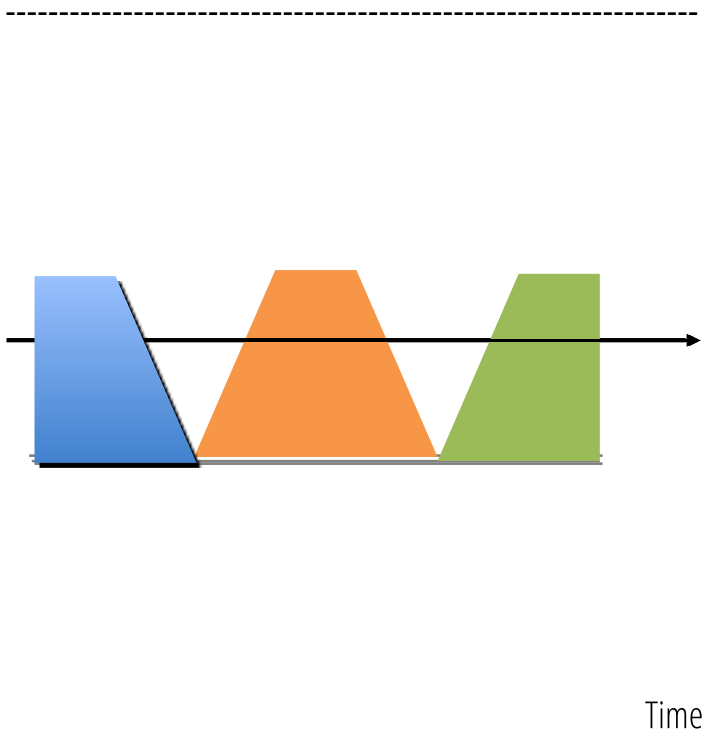
# What happens with best-effort?

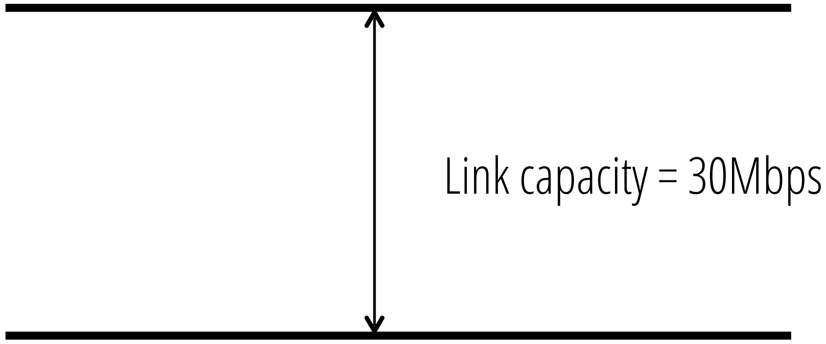


# What happens with best-effort?

All good! No overloading



# Smooth vs. Bursty Applications

# Smooth vs. Bursty Applications

- Characterized by the ratio between an app’s peak to average transmission rate

# Smooth vs. Bursty Applications

- Characterized by the ratio between an app’s peak to average transmission rate

- Some apps have relatively small peak-to-average ratios
  - Voice might have a ratio of 3:1 or so

# Smooth vs. Bursty Applications

- Characterized by the ratio between an app's peak to average transmission rate

- Some apps have relatively small peak-to-average ratios
  - Voice might have a ratio of 3:1 or so

- Data applications tend to be rather bursty
  - E.g., ratios of 100 or greater are common when web browsing

# Smooth vs. Bursty Applications

- Characterized by the ratio between an app's peak to average transmission rate

- Some apps have relatively small peak-to-average ratios
  - Voice might have a ratio of 3:1 or so

- Data applications tend to be rather bursty
  - E.g., ratios of 100 or greater are common when web browsing

- That's why the phone network used reservations and the Internet does not!

# Which makes more efficient use of network capacity?

Answer: Packet switching is typically more efficient

- But how much better depends on the "burstiness" of the traffic sources

# Other differences in efficiency?

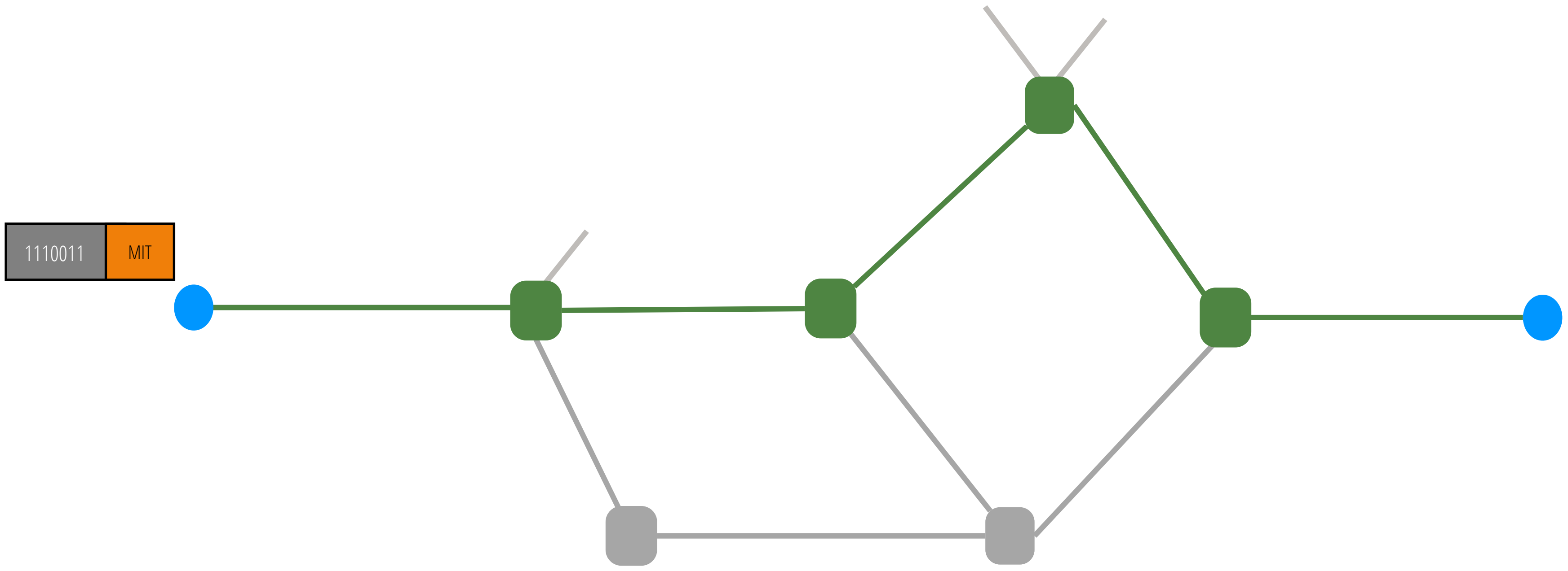
# Other differences in efficiency?

- Circuit switching spends some time to setup / teardown circuits
  - Very inefficient when you don’t have much data to send! (short flows)

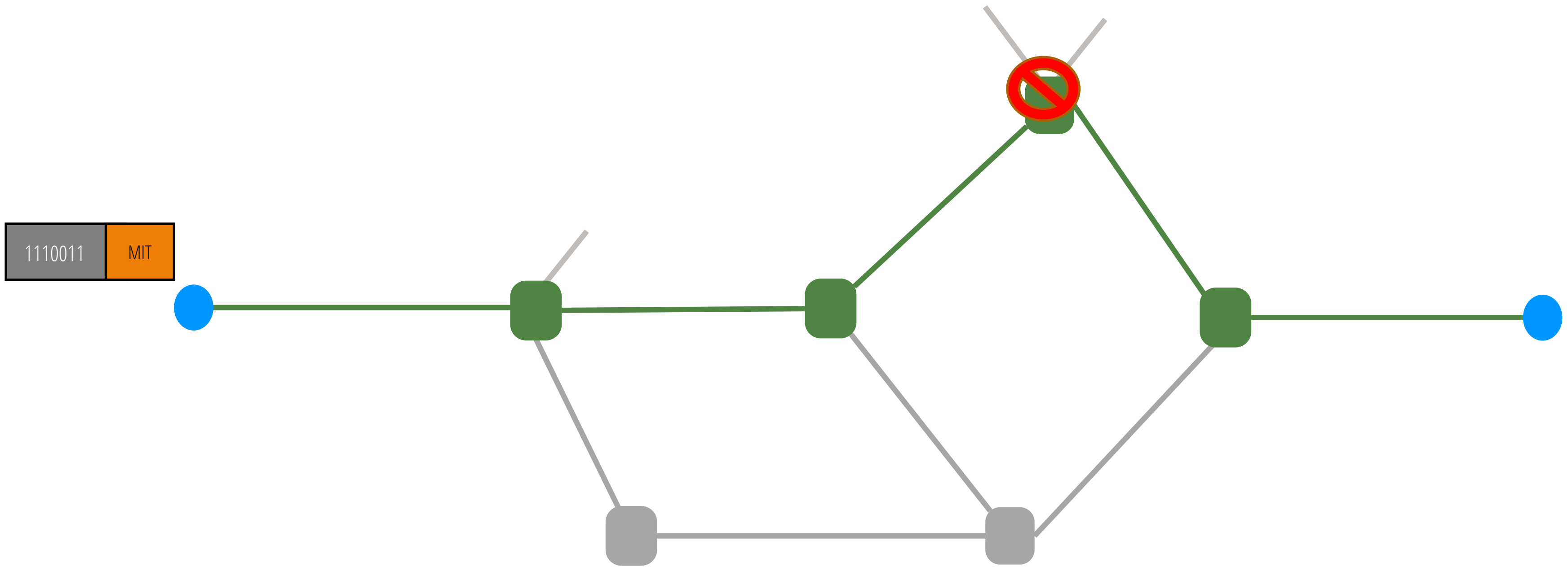
# Circuit *vs.* Packet switching: which is better?

- 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)

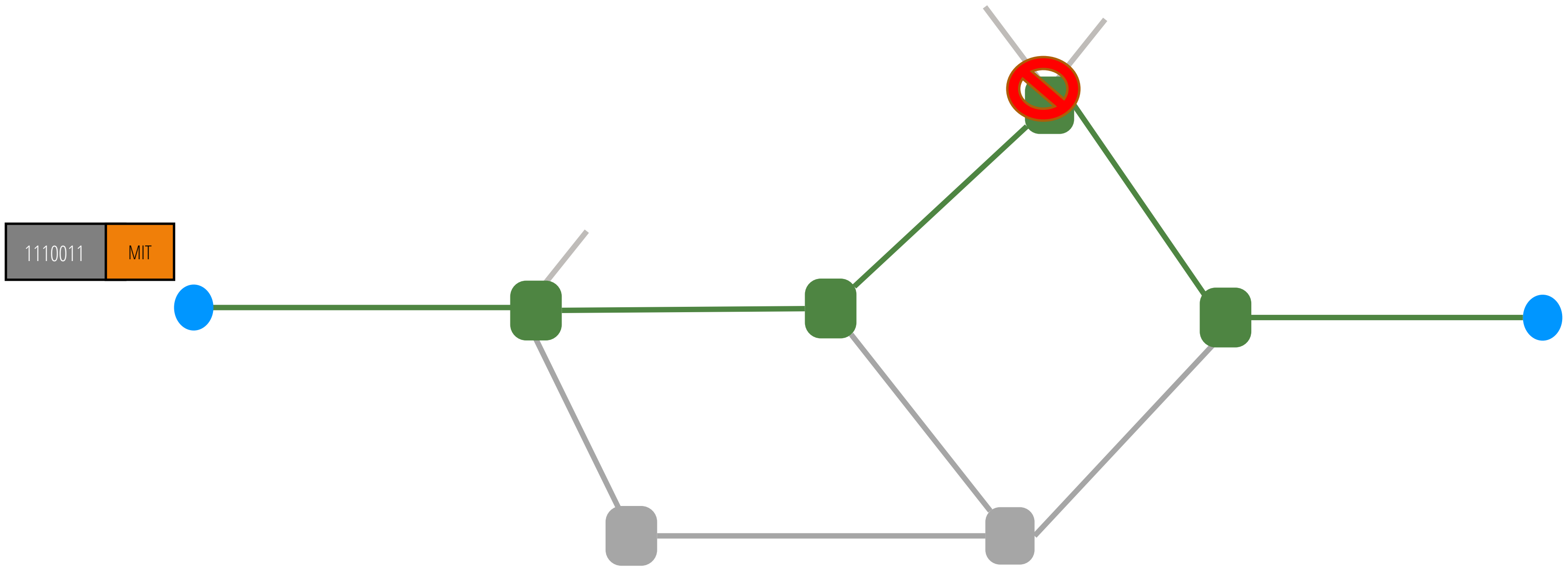
# What happens in the event of a failure?



# What happens in the event of a failure?

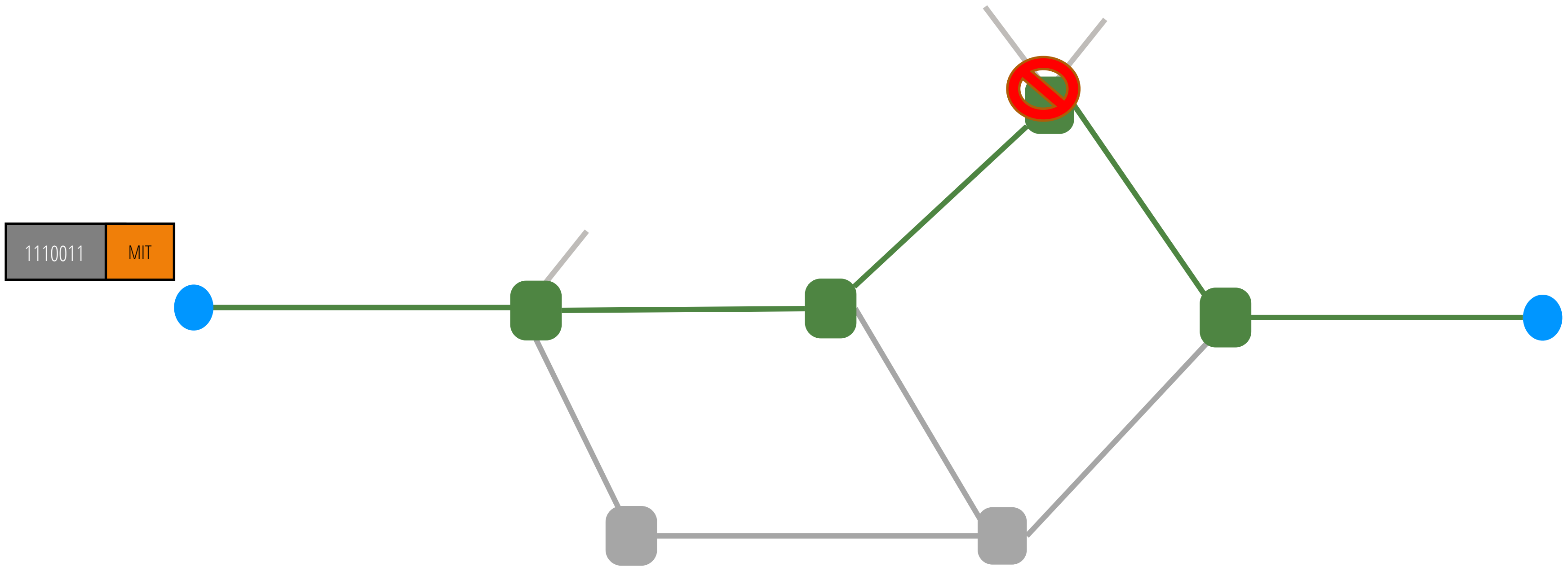


# What happens in the event of a failure?



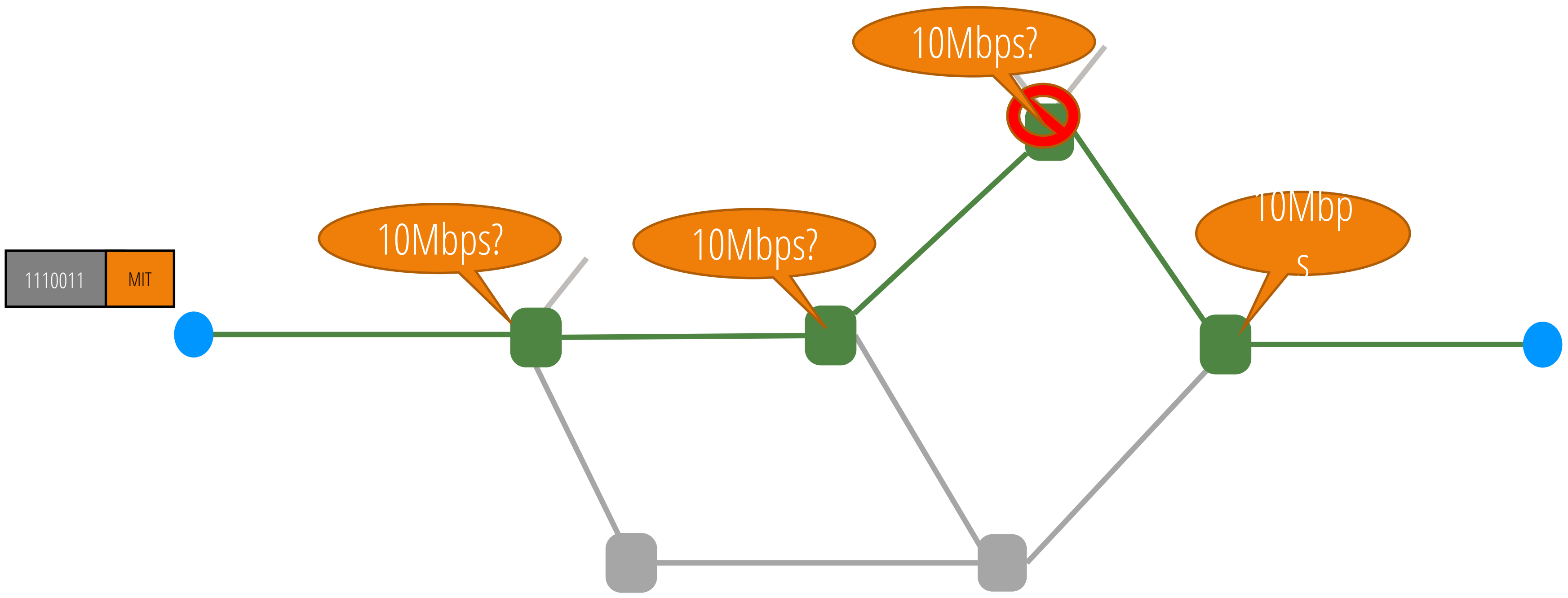

With packet switching?

# What happens in the event of a failure?



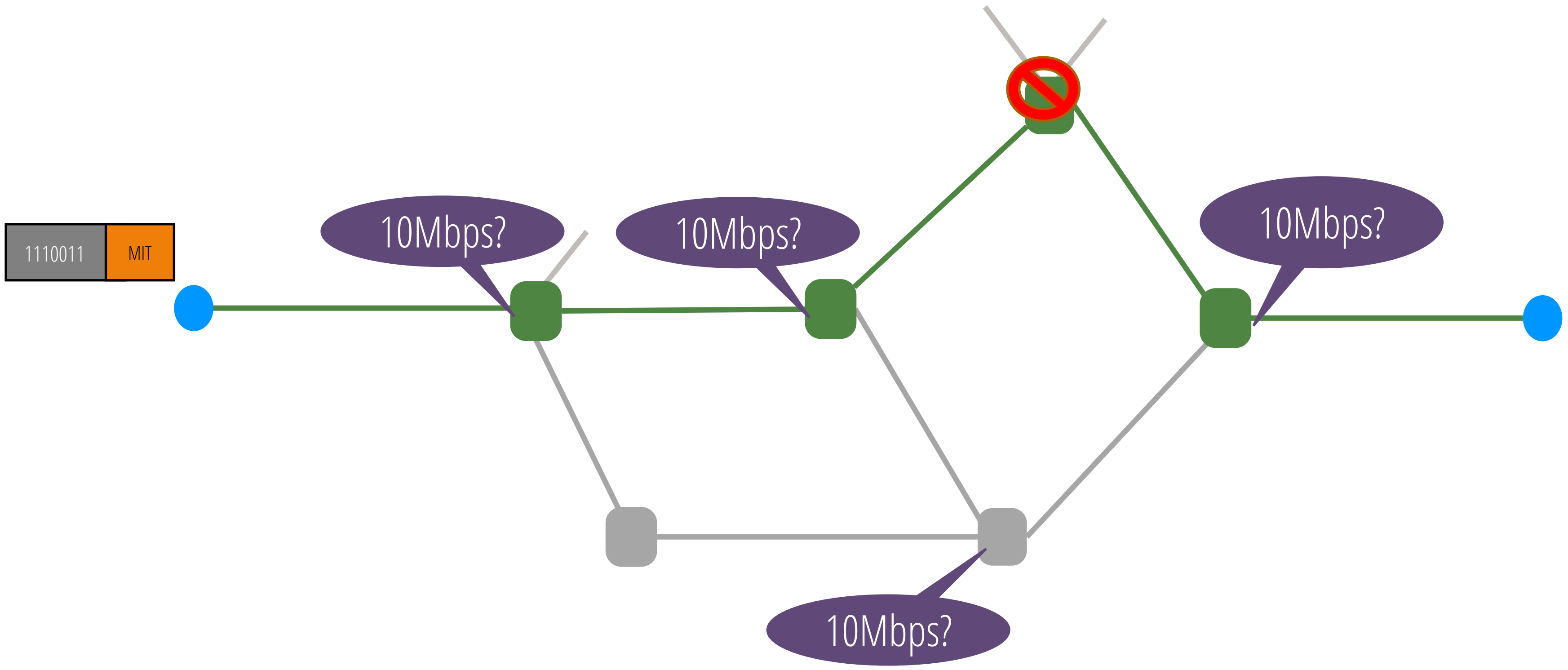

With circuit switching?

# What happens in the event of a failure?




With circuit switching?

# What happens in the event of a failure?




With circuit switching?

# Recap: Failure Recovery in Packet Switching

# Recap: Failure Recovery in Packet Switching

- Link goes down, then what?

- Network must detect failure

# Recap: Failure Recovery in Packet Switching

- Link goes down, then what?
- Network must detect failure
- Network recalculates routes
  - (Job of the routing control plane)

# Recap: Failure Recovery in Packet Switching

- Link goes down, then what?
- Network must detect failure
- Network recalculates routes
  - (Job of the routing control plane)
- Endhosts and individual flows do nothing special
  - Except cope with the temporary loss of service

# Recap: Failure Recovery in Circuit Switching

- Network must do all the things needed for packet switching

# Recap: Failure Recovery in Circuit Switching

- Network must do all the things needed for packet switching

- And in addition, endhosts must
  - detect failure
  - teardown old reservations
  - send a new reservation request

# Recap: Failure Recovery in Circuit Switching

- Network must do all the things needed for packet switching

- And in addition, endhosts must
  - detect failure
  - teardown old reservations
  - send a new reservation request
- All impacted endhosts must do this, for each impacted flow

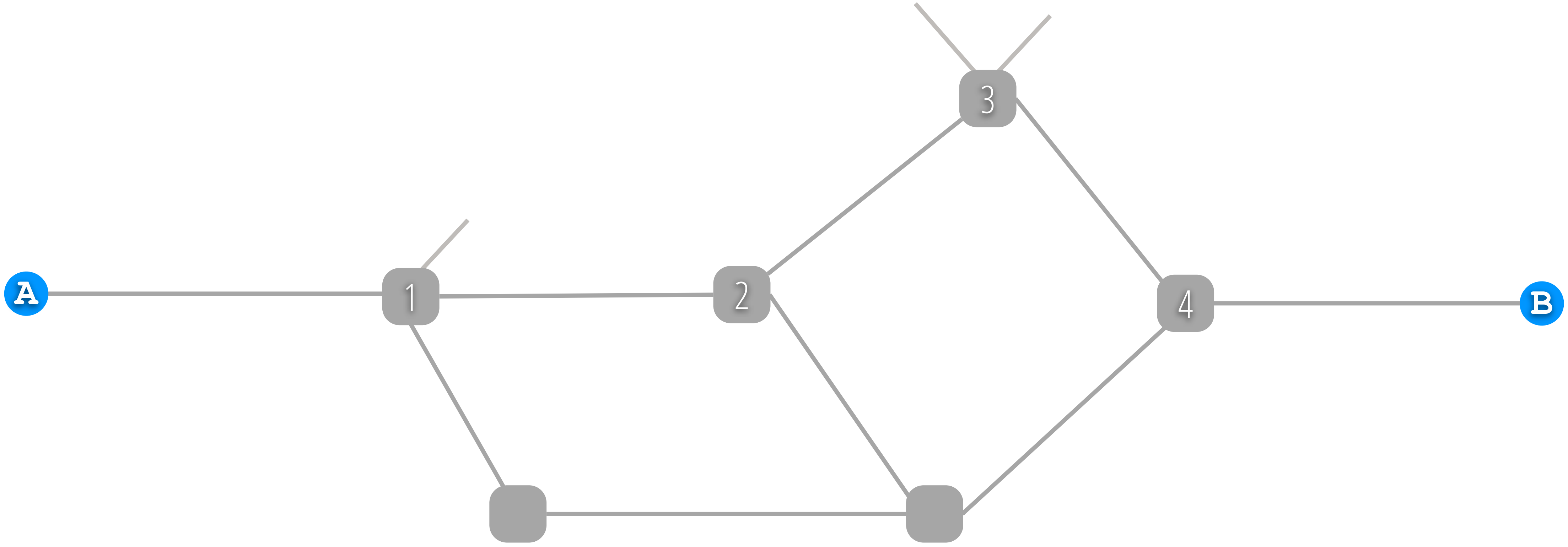
# Recap: Failure Recovery in Circuit Switching

- Network must do all the things needed for packet switching
- And in addition, endhosts must
  - detect failure
  - teardown old reservations
  - send a new reservation request
- All impacted endhosts must do this, for each impacted flow
- If millions of flows were going through a switch, then millions of reservation requests are being simultaneously re-established!

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

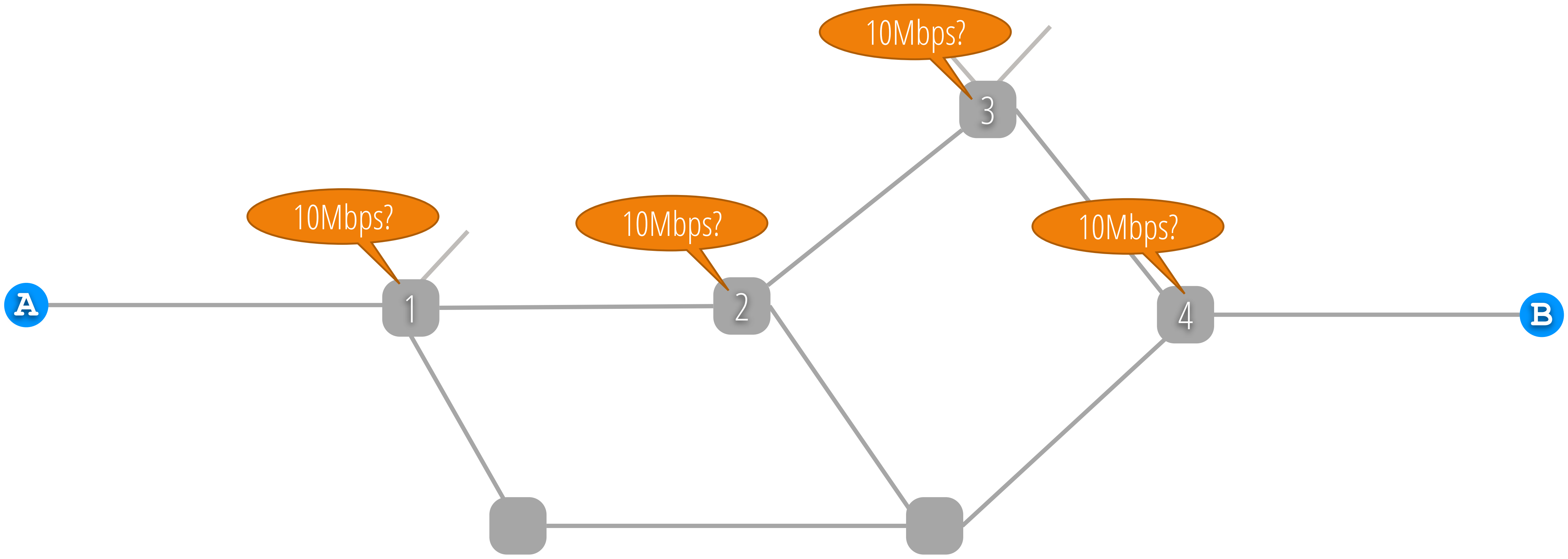
- What are the dimensions along which we should compare?
  - As an abstraction to applications (endhosts)
  - Efficiency
  - Handling failures (at scale)
  - Complexity of implementation (at scale)

# Recall...



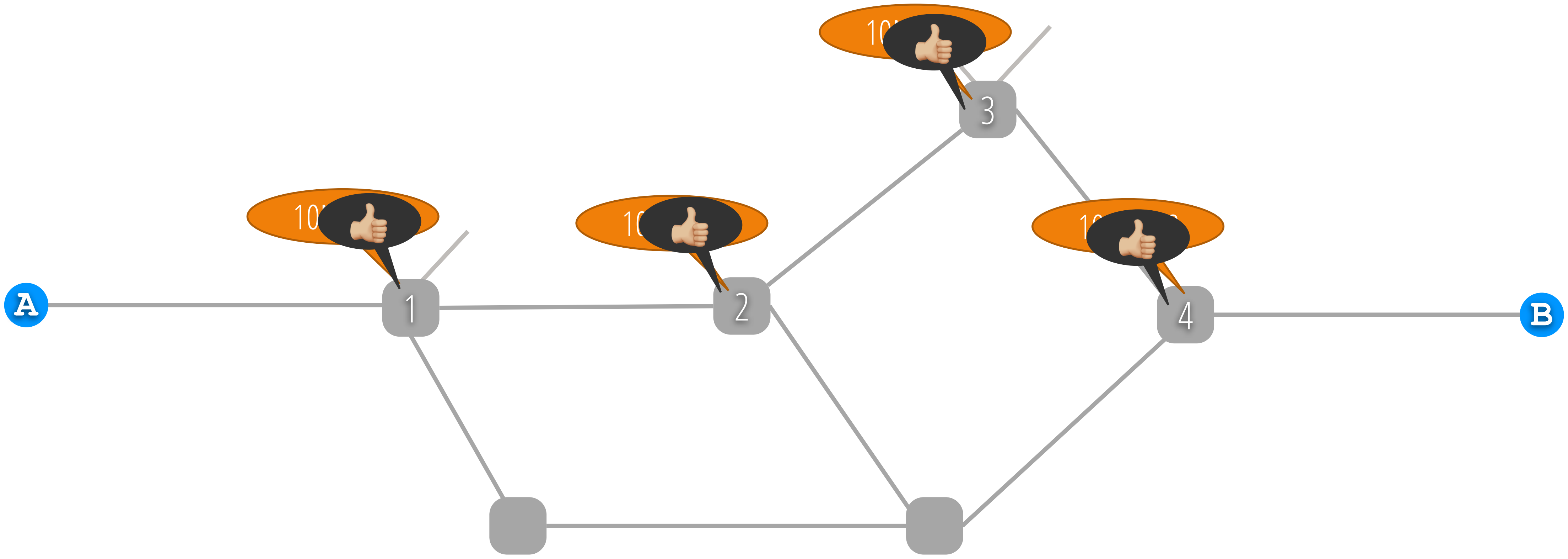
# Recall...

(1) source sends a reservation request to the destination



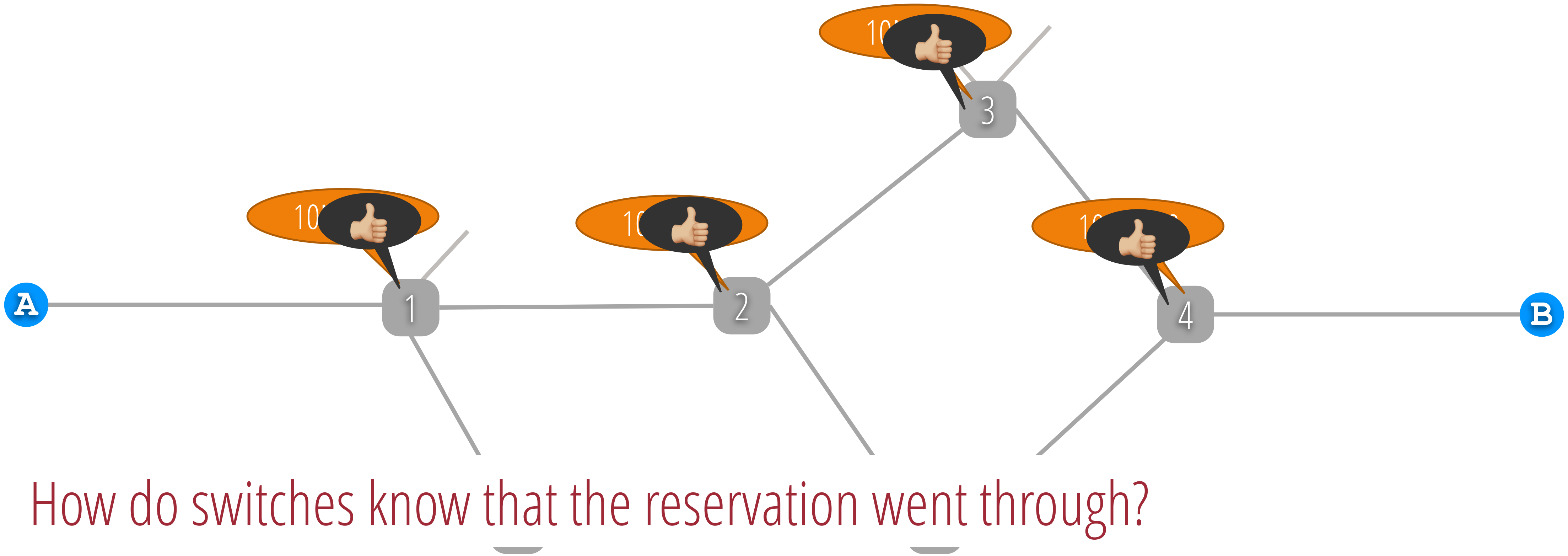
# Recall...

(1) source sends a reservation request to the destination



# Recall...

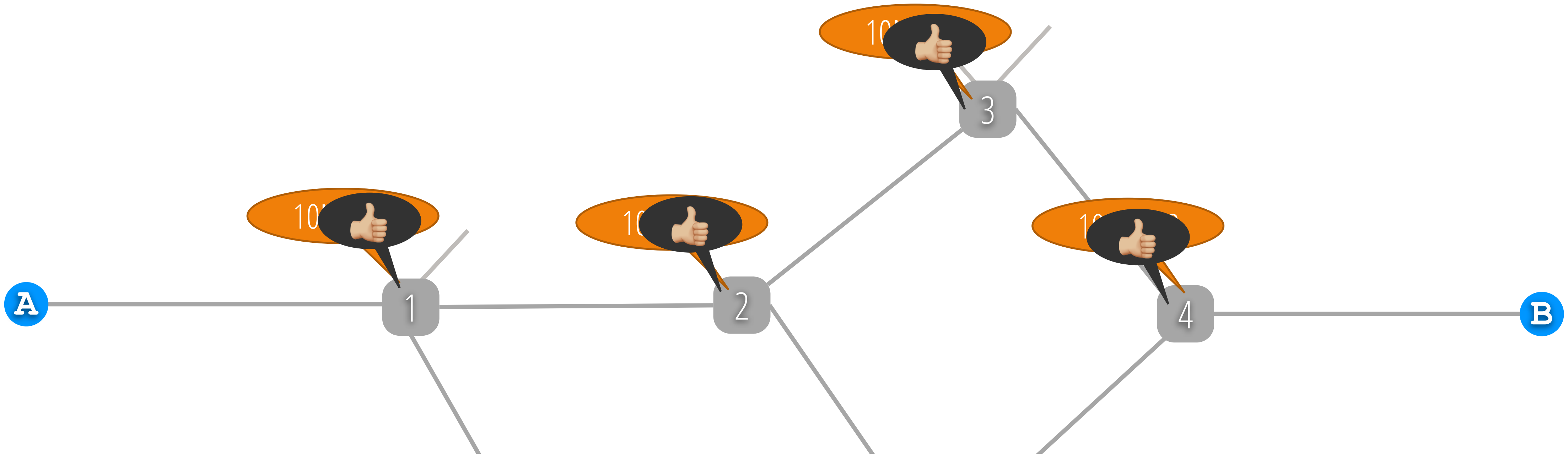
(1) source sends a reservation request to the destination




How do switches know that the reservation went through?

# Recall...

(1) source sends a reservation request to the destination

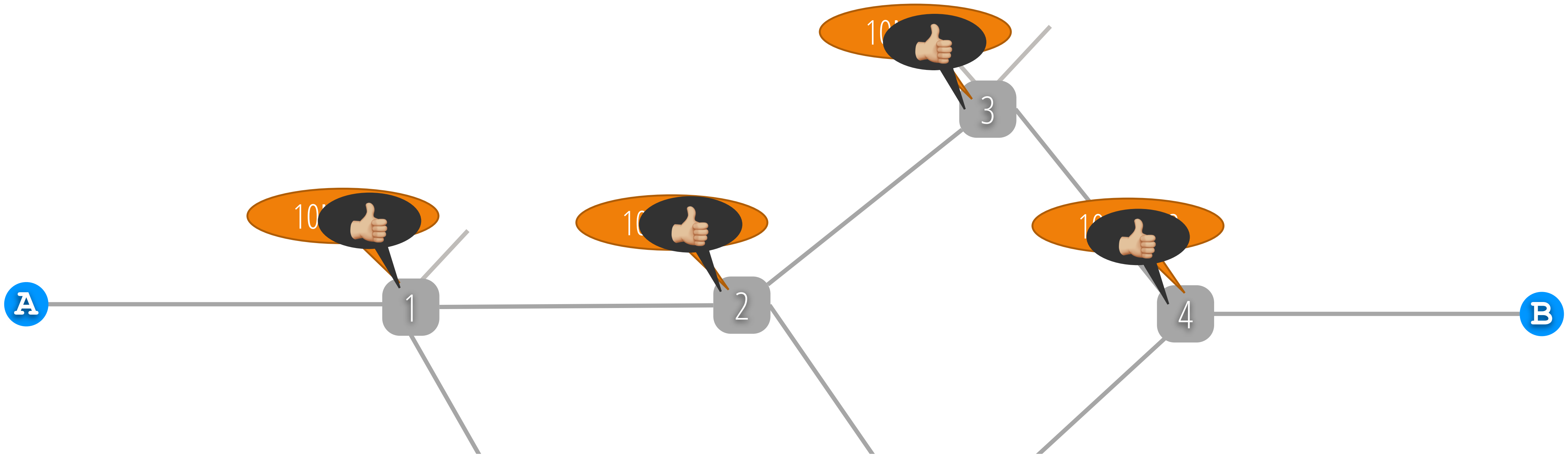



H

What happens if the reservation request is lost mid way?

# Recall...

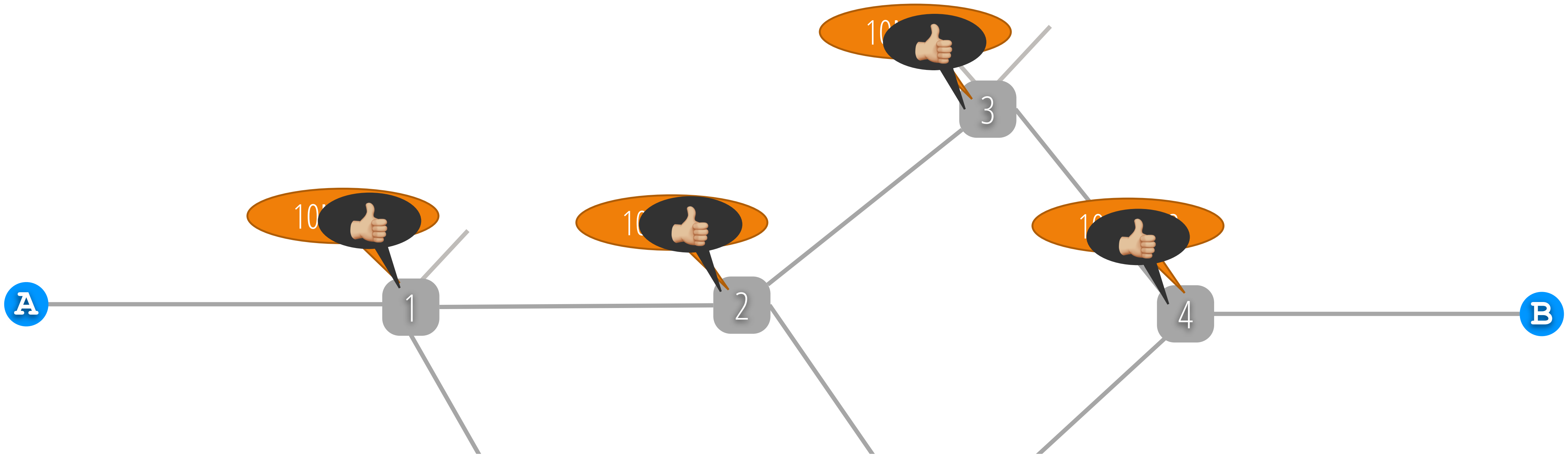
(1) source sends a reservation request to the destination




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

# Recall...

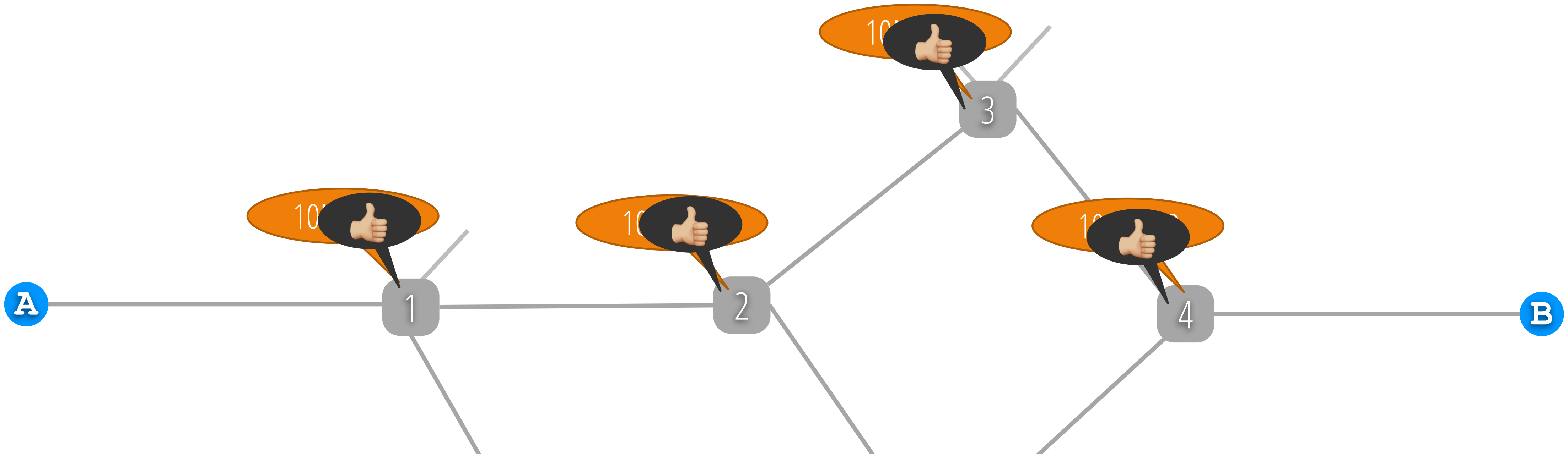
(1) source sends a reservation request to the destination




What should the endhost do if the reservation is declined?

# Recall...

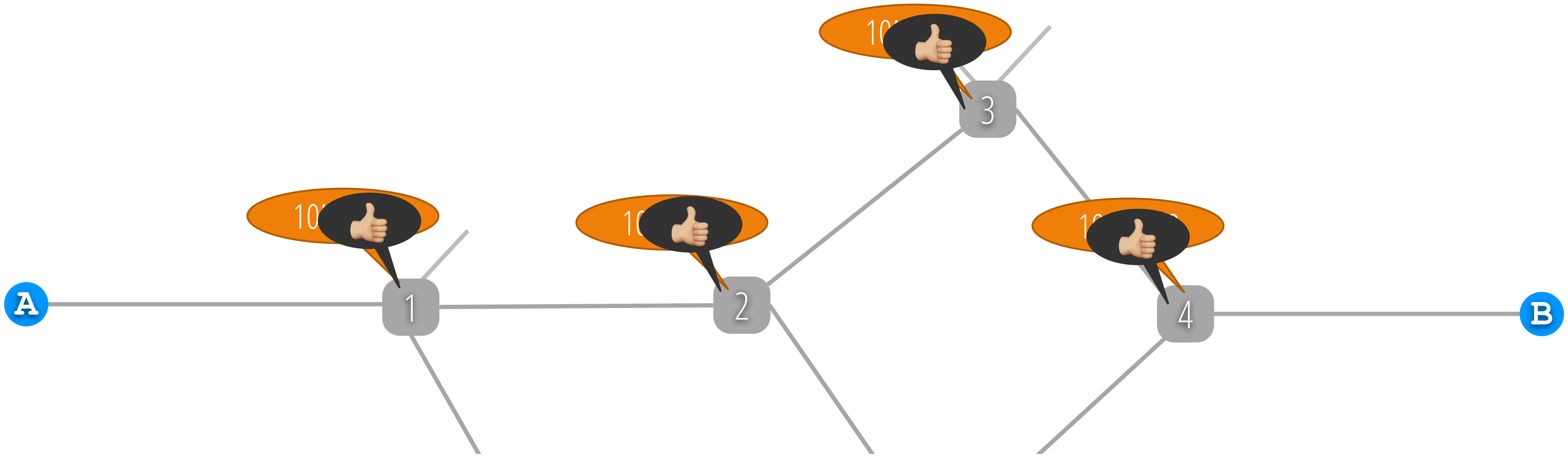
(1) source sends a reservation request to the destination




What happens if the underlying route changes?

# Recall...

(1) source sends a reservation request to the destination




And on and on....

# Recap: Circuit *vs.* Packet Switching

# Recap: Circuit *vs.* Packet Switching

- Pros for circuit switching:
  - Better application performance (reserved bandwidth)
  - More predictable and understandable (w/o failures)

# 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?

# What does the Internet use today?

- Packet switching is the default

# What does the Internet use today?

- Packet switching is the default
  - Some use of RSVP (“Resource Reservation Protocol”) within one domain

# What does the Internet use today?

- Packet switching is the default
  - Some use of RSVP (“Resource Reservation Protocol”) within one domain

- But you *can* also buy a dedicated circuit (e.g., MPLS circuits, leased lines, etc.)

# What does the Internet use today?

- Packet switching is the default
  - Some use of RSVP (“Resource Reservation Protocol”) within one domain

- But you *can* also buy a dedicated circuit (e.g., MPLS circuits, leased lines, etc.)
  - Often used by enterprises from one branch location to another (or to/from cloud)

# What does the Internet use today?

- Packet switching is the default
  - Some use of RSVP (“Resource Reservation Protocol”) within one domain

- But you *can* also buy a dedicated circuit (e.g., MPLS circuits, leased lines, etc.)
  - Often used by enterprises from one branch location to another (or to/from cloud)
  - Very expensive (e.g., 10-20x higher than a normal connection)

# What does the Internet use today?

- Packet switching is the default
  - Some use of RSVP (“Resource Reservation Protocol”) within one domain

- But you *can* also buy a dedicated circuit (e.g., MPLS circuits, leased lines, etc.)
  - Often used by enterprises from one branch location to another (or to/from cloud)
  - Very expensive (e.g., 10-20x higher than a normal connection)
  - Often statically set up (manually), long-lived (e.g., years), and per user (*vs.* per flow)

# What does the Internet use today?

- Packet switching is the default
  - Some use of RSVP (“Resource Reservation Protocol”) within one domain

- But you *can* also buy a dedicated circuit (e.g., MPLS circuits, leased lines, etc.)
  - Often used by enterprises from one branch location to another (or to/from cloud)
  - Very expensive (e.g., 10-20x higher than a normal 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

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

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

# 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 (late 80s-90s): research & industry believed we'd need circuit switching

# 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 (late 80s-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 (many Berkeley folks were pioneers in this space!)

# 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 (late 80s-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 (many Berkeley folks were pioneers in this space!)
- Ultimately, a failed vision. Why?
  - All the reasons we 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 (late 80s-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 (many Berkeley folks were pioneers in this space!)
- Ultimately, a failed vision. Why?
  - All the reasons we discussed…
  - …and people rewrote apps to be adaptive (turns out we didn't really need guaranteed BW!)

# 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 (late 80s-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 (many Berkeley folks were pioneers in this space!)

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

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

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

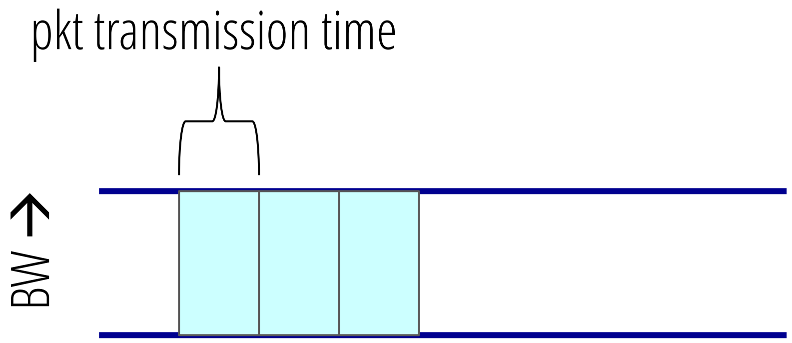
- The next iteration (late 80s-90s): research & industry believed we'd need circuit switching
  - Envisioned that voice/live TV/ would be the Internet's true killer app
  - Spent 10+ years trying to realize this vision (many Berkeley folks were pioneers in this space!)

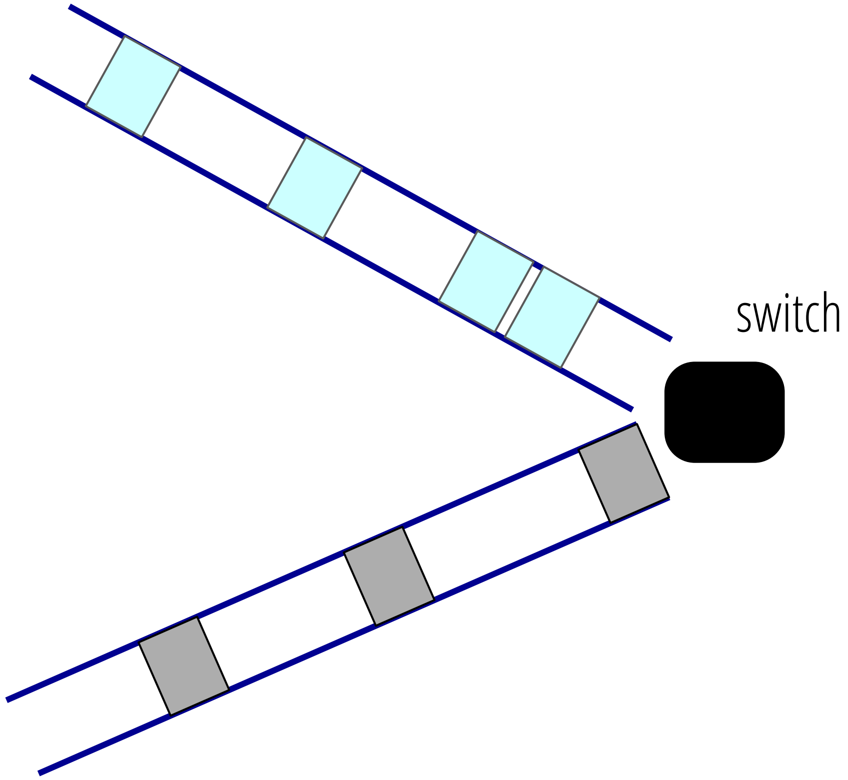
- Ultimately, a failed vision. Why?
  - All the reasons we discussed…
  - …and people rewrote apps to be adaptive (turns out we didn't really need guaranteed BW!)
  - …and Email and the web emerged as the killer apps (of the time)

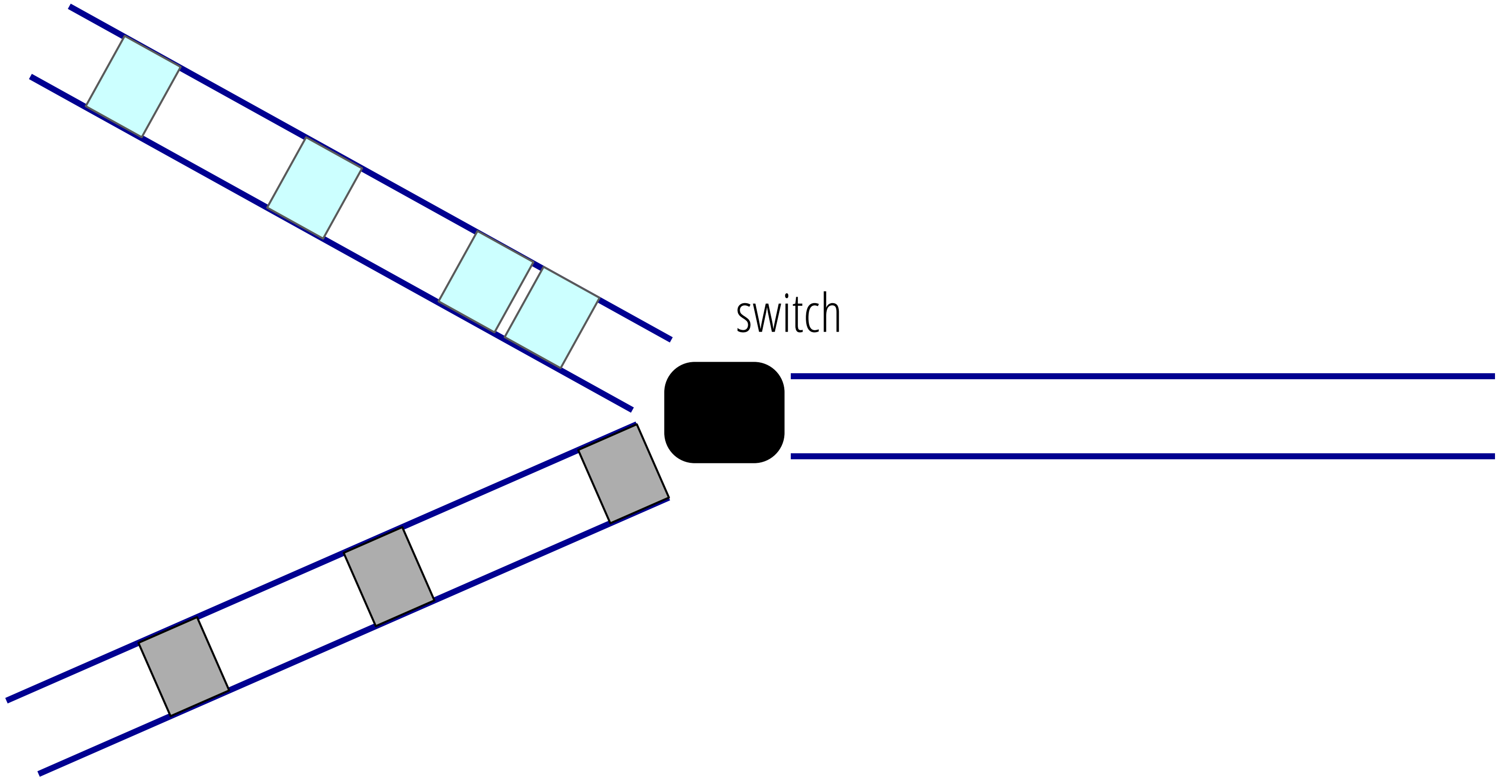
*A lesson in how technology can transform user behavior!*

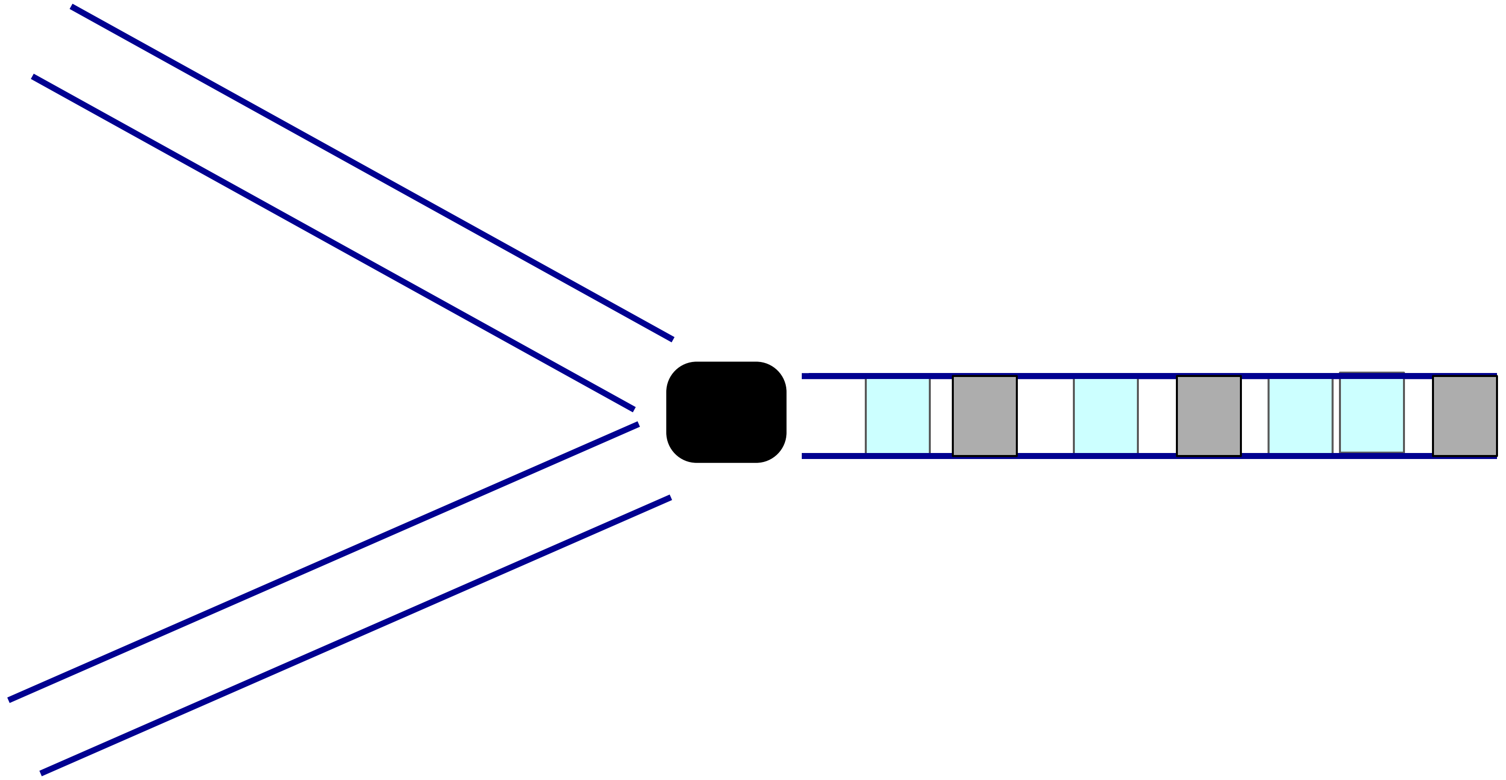
Let's take a closer look at packet switching ....

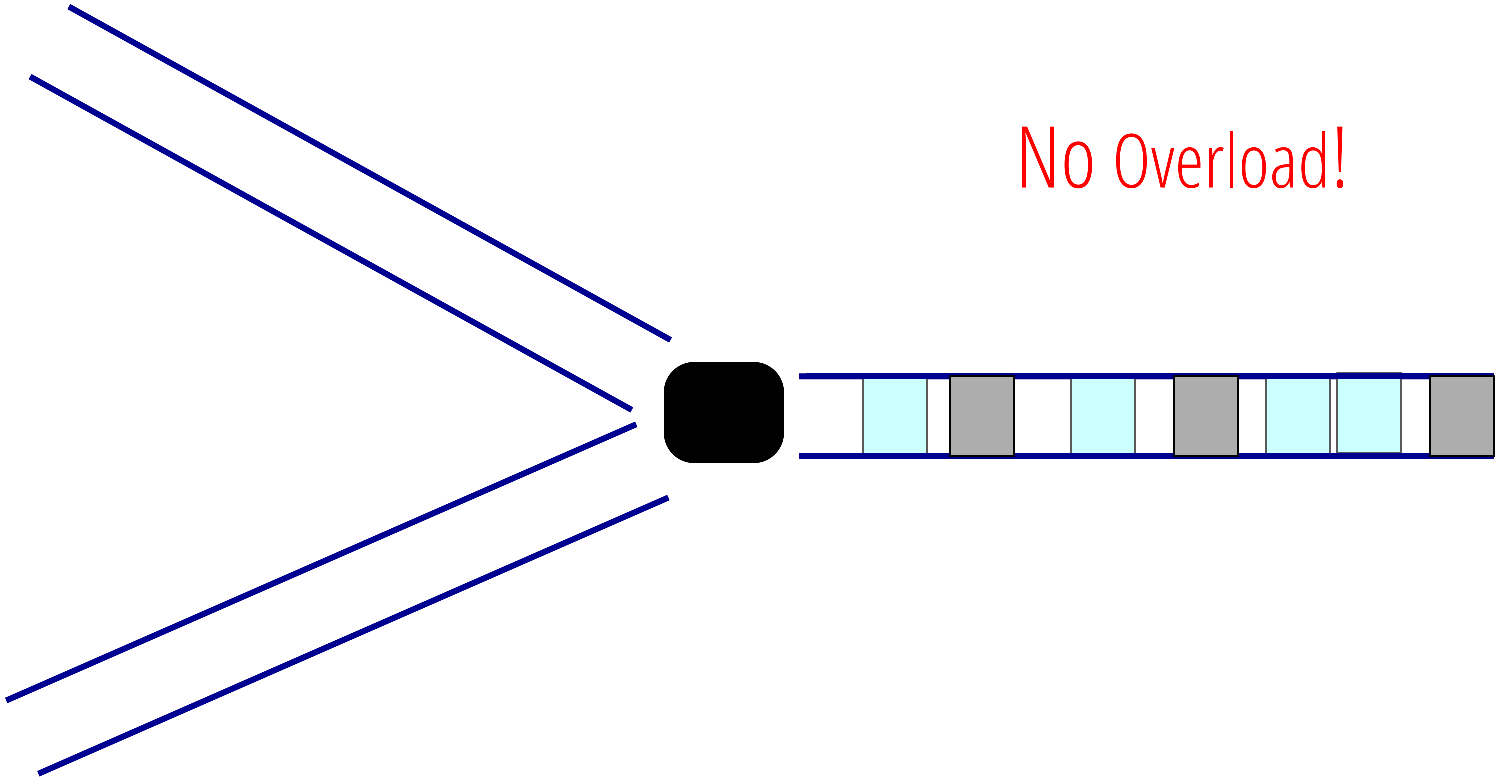
# Recall, packets in flight: “pipe” view

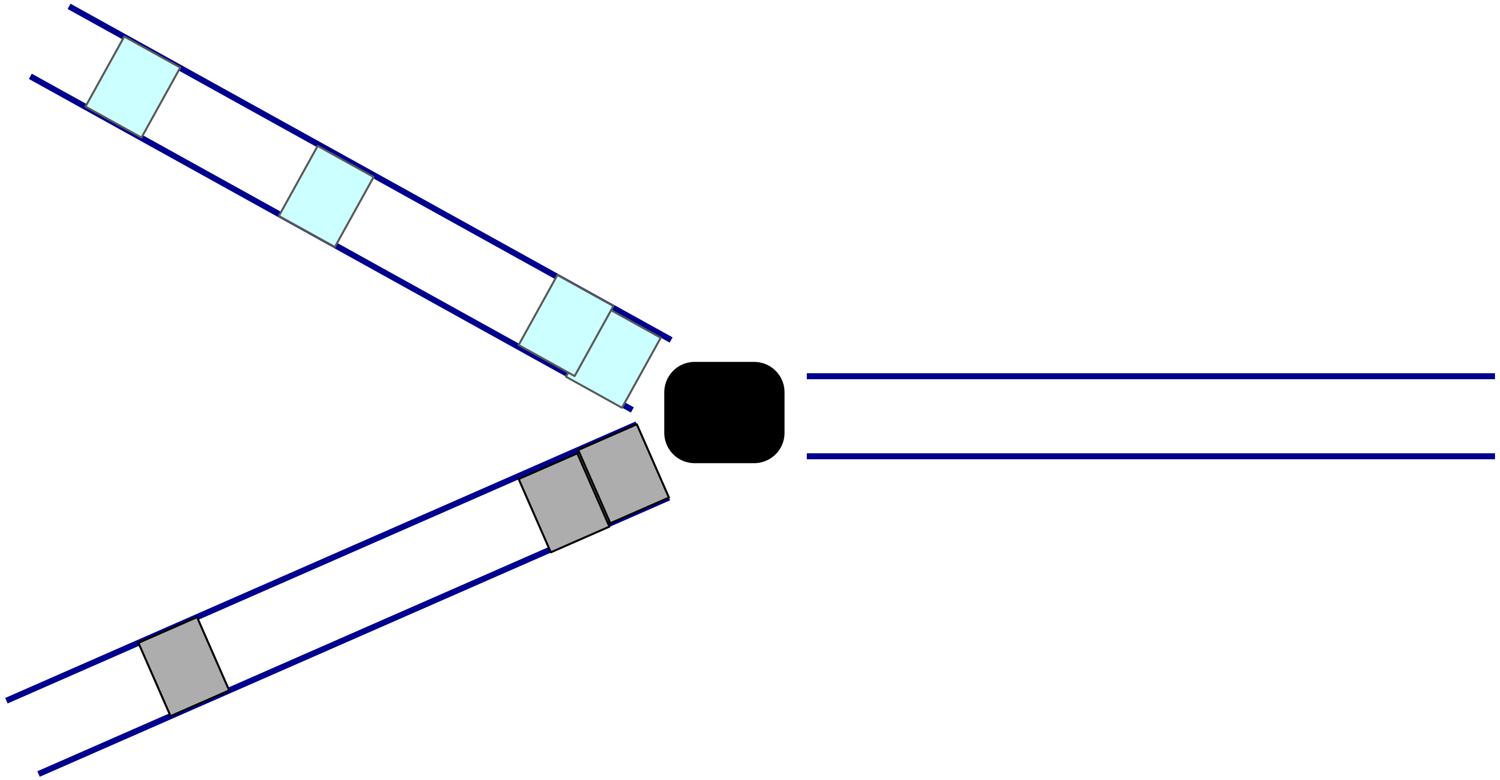


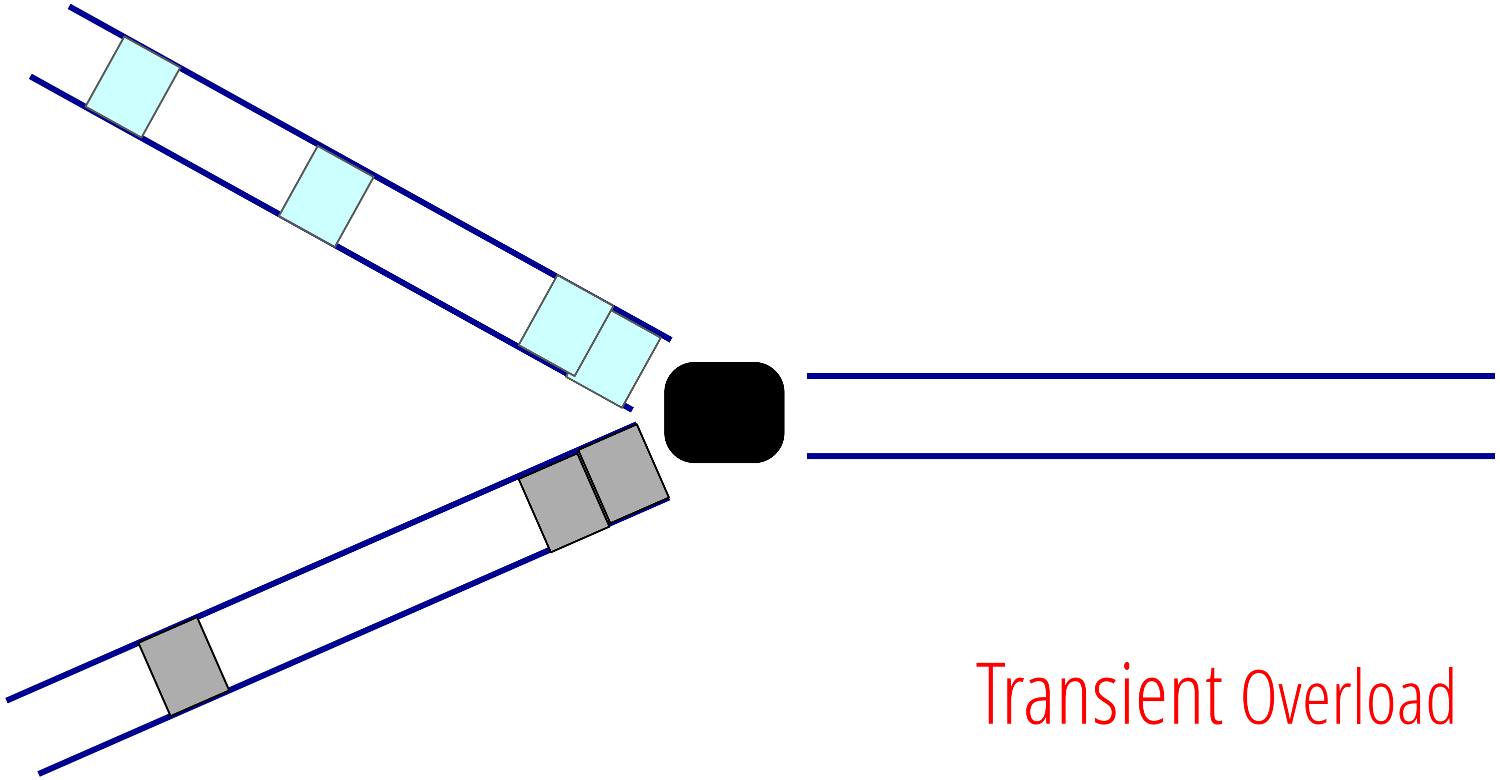

switch


switch

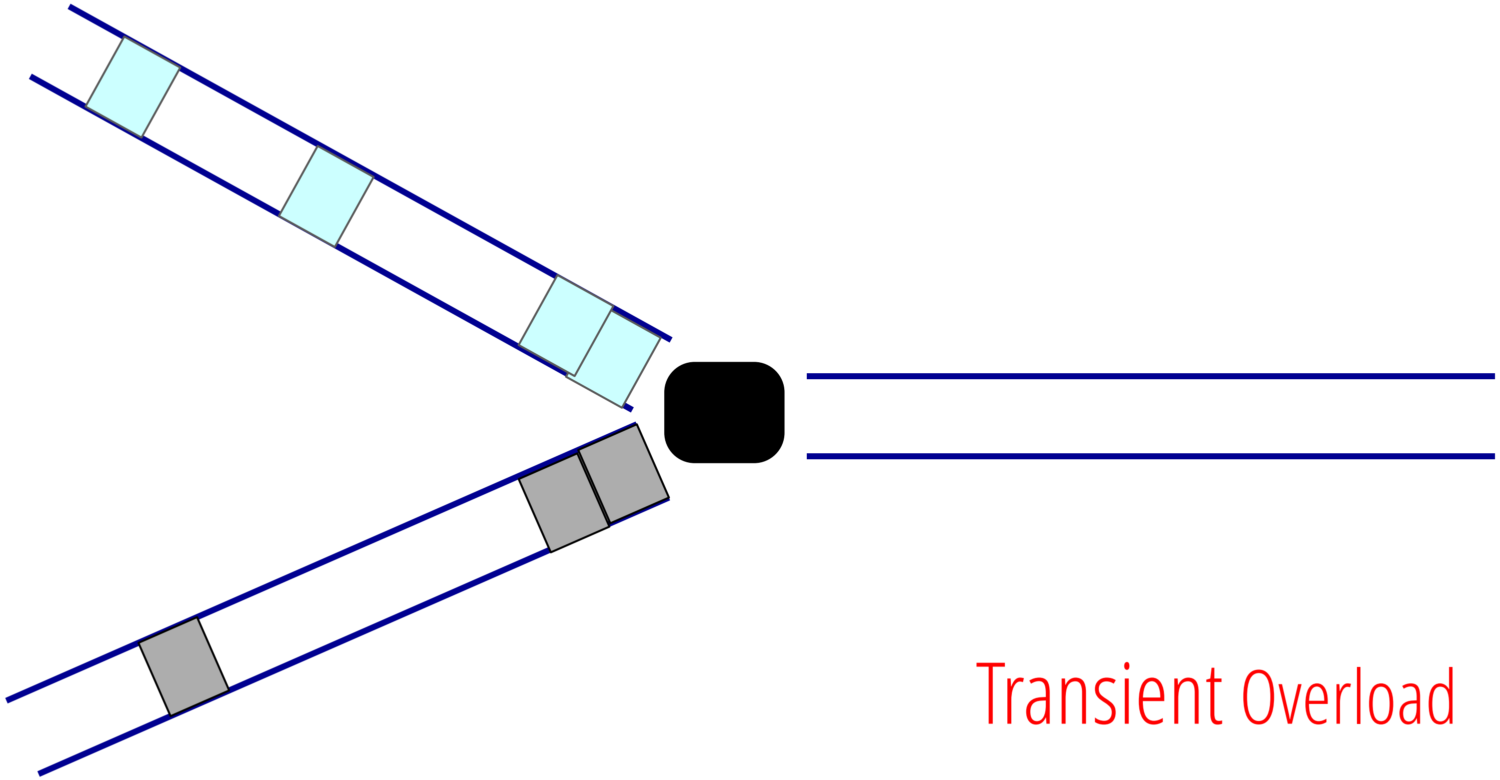



No Overload!



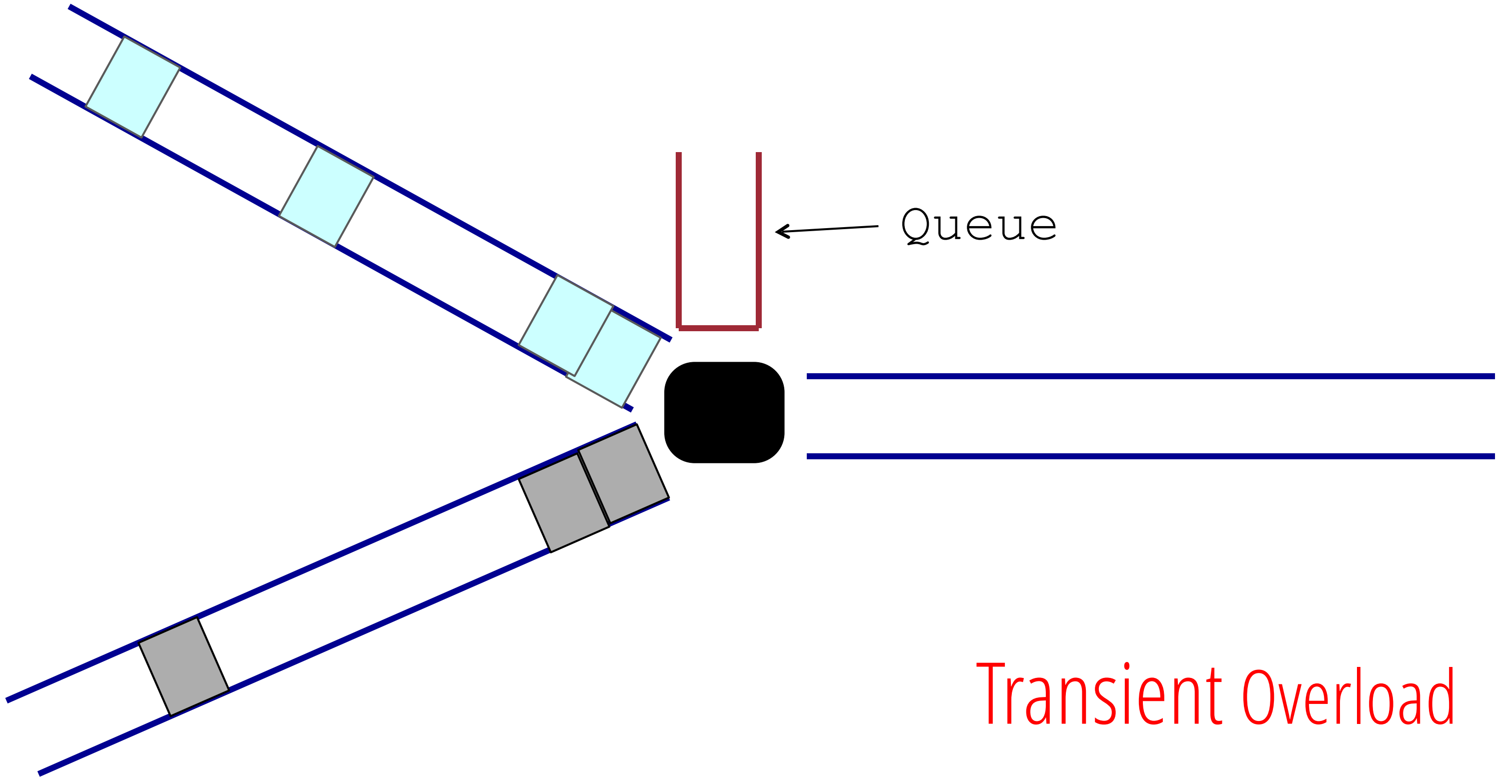


Transient Overload



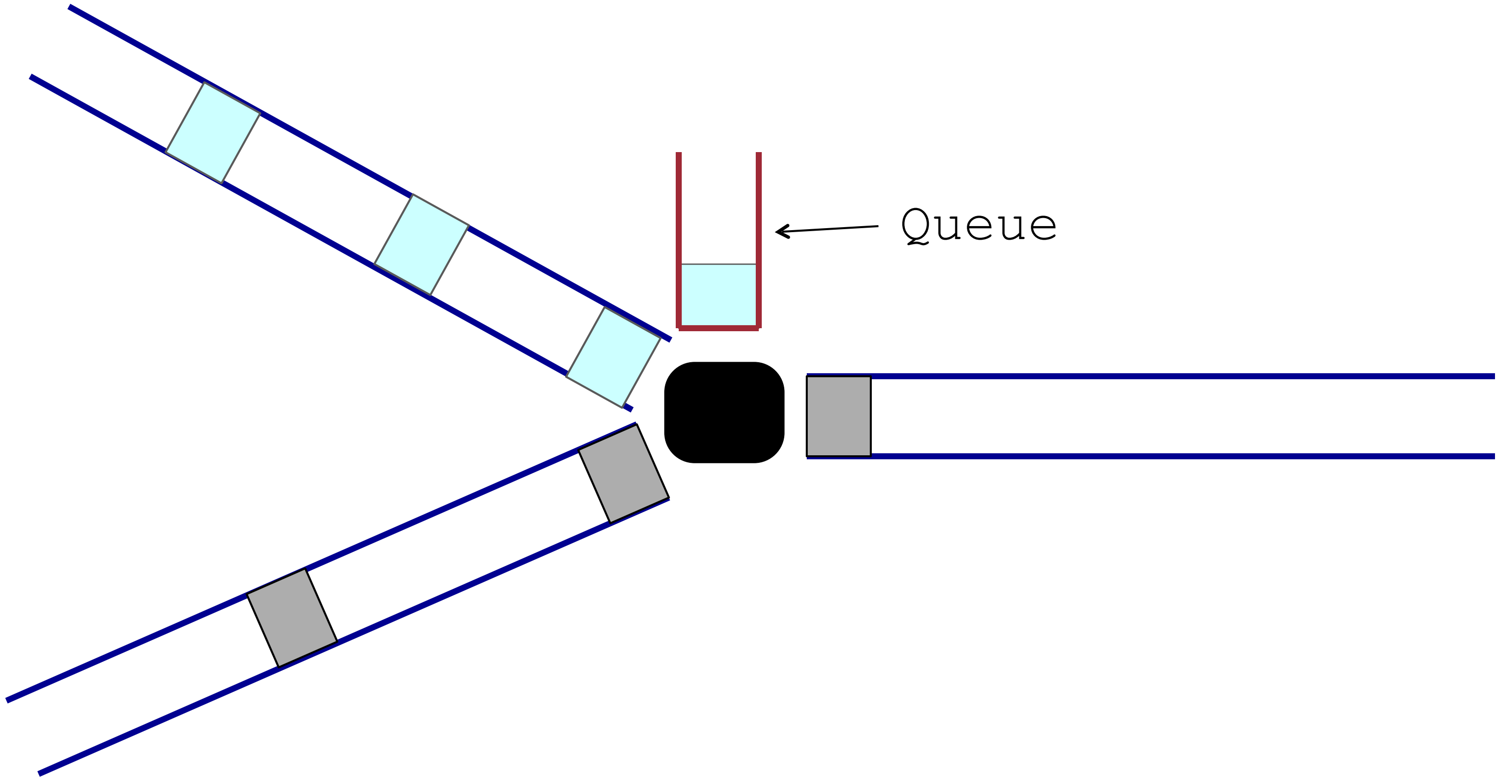
Transient Overload

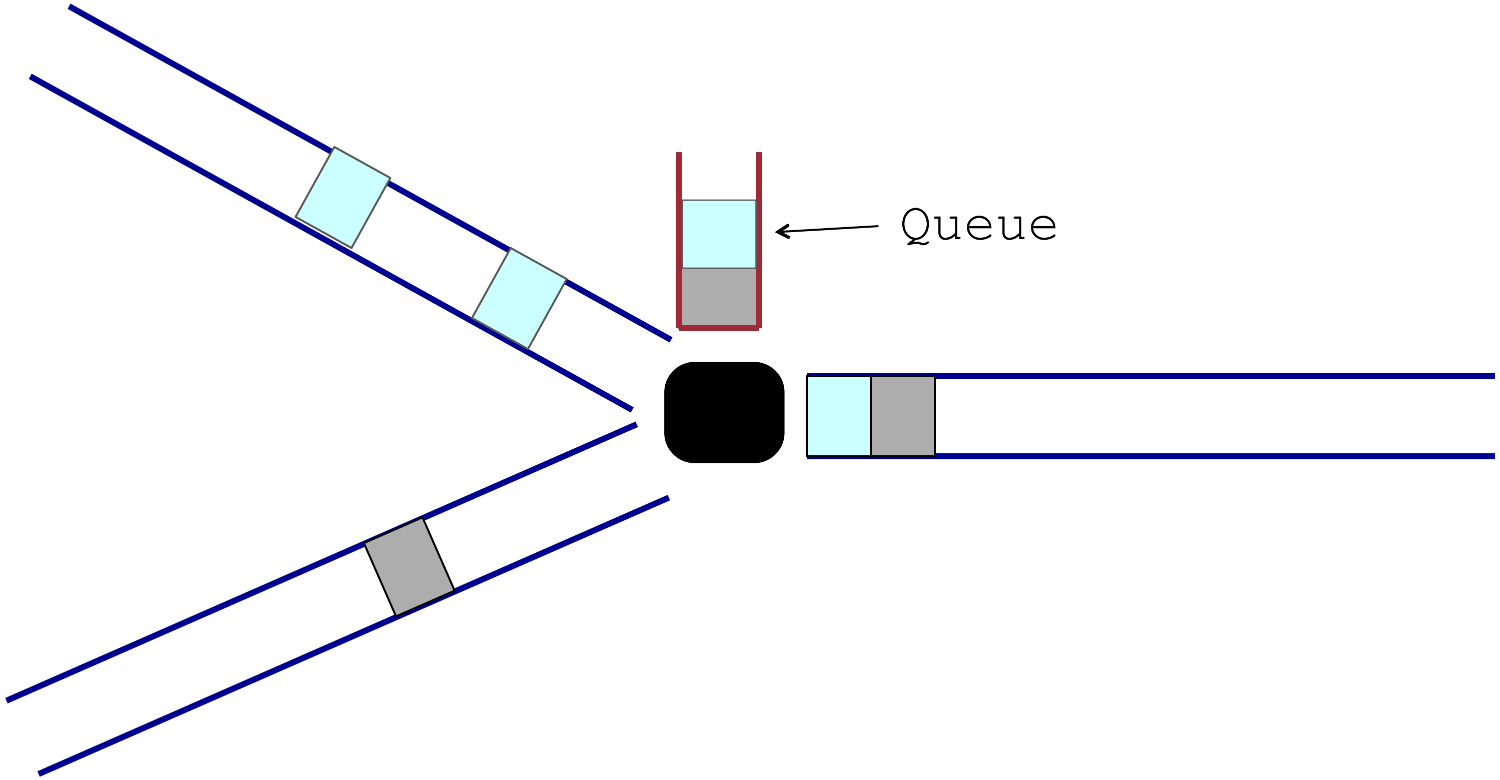
Not a rare event!

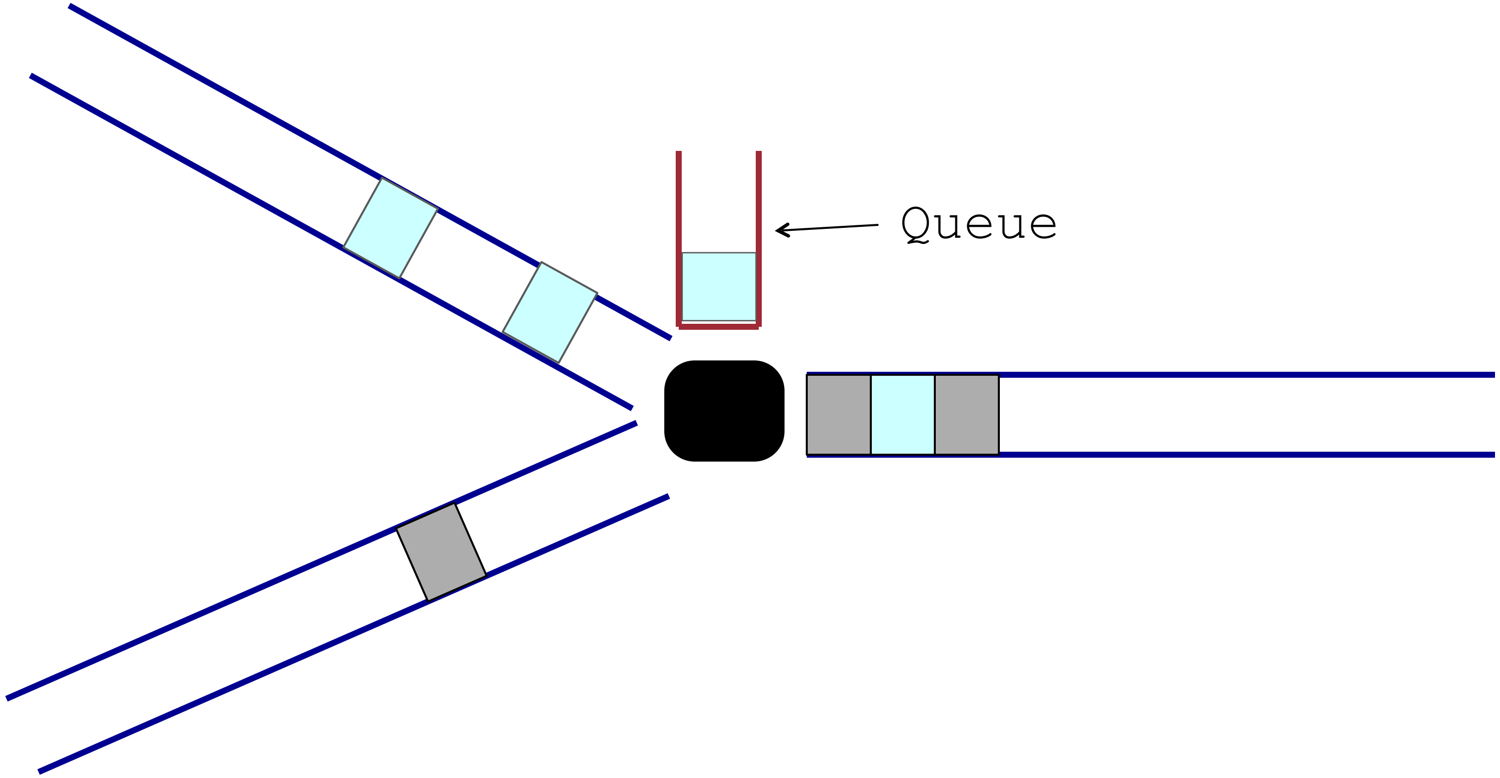


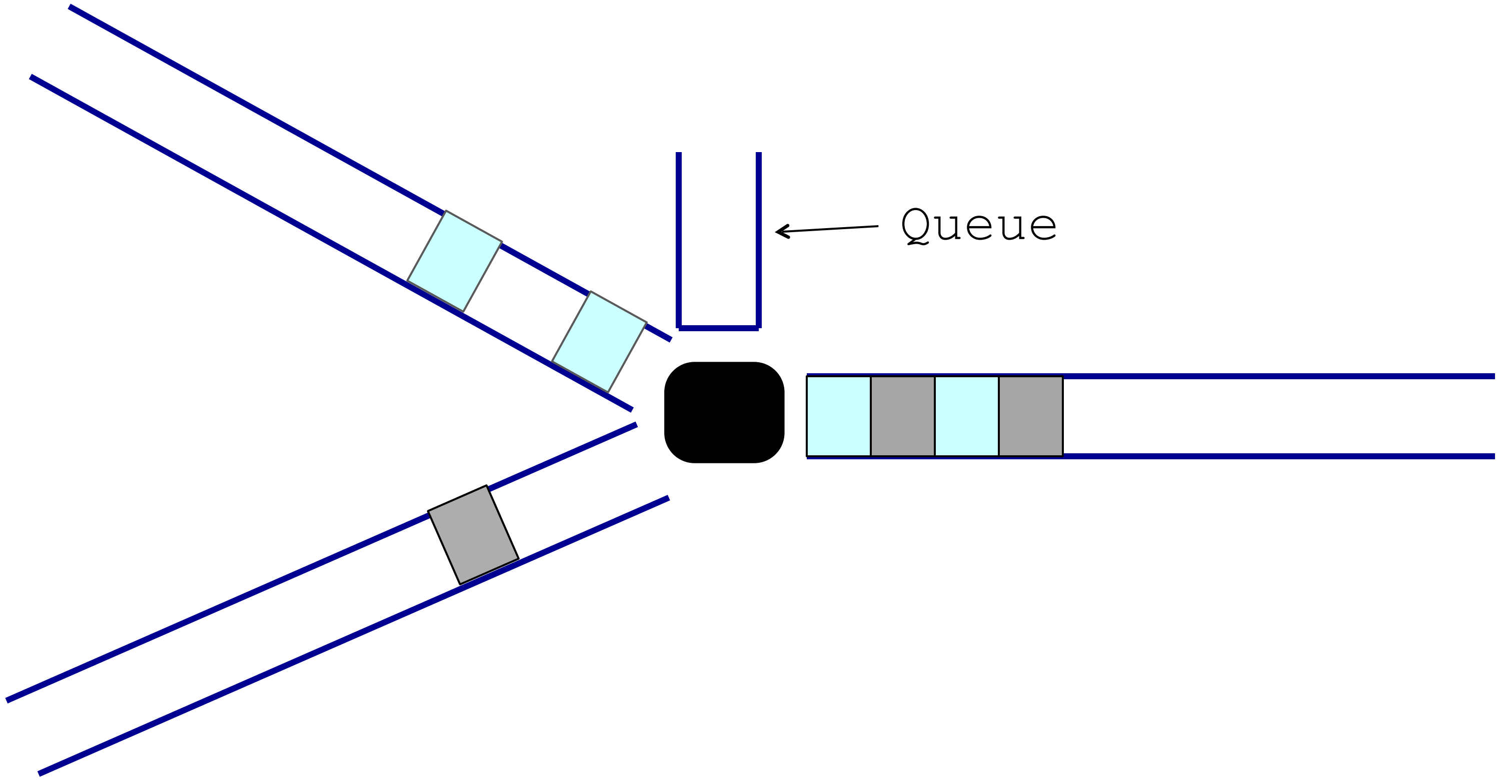

Transient Overload

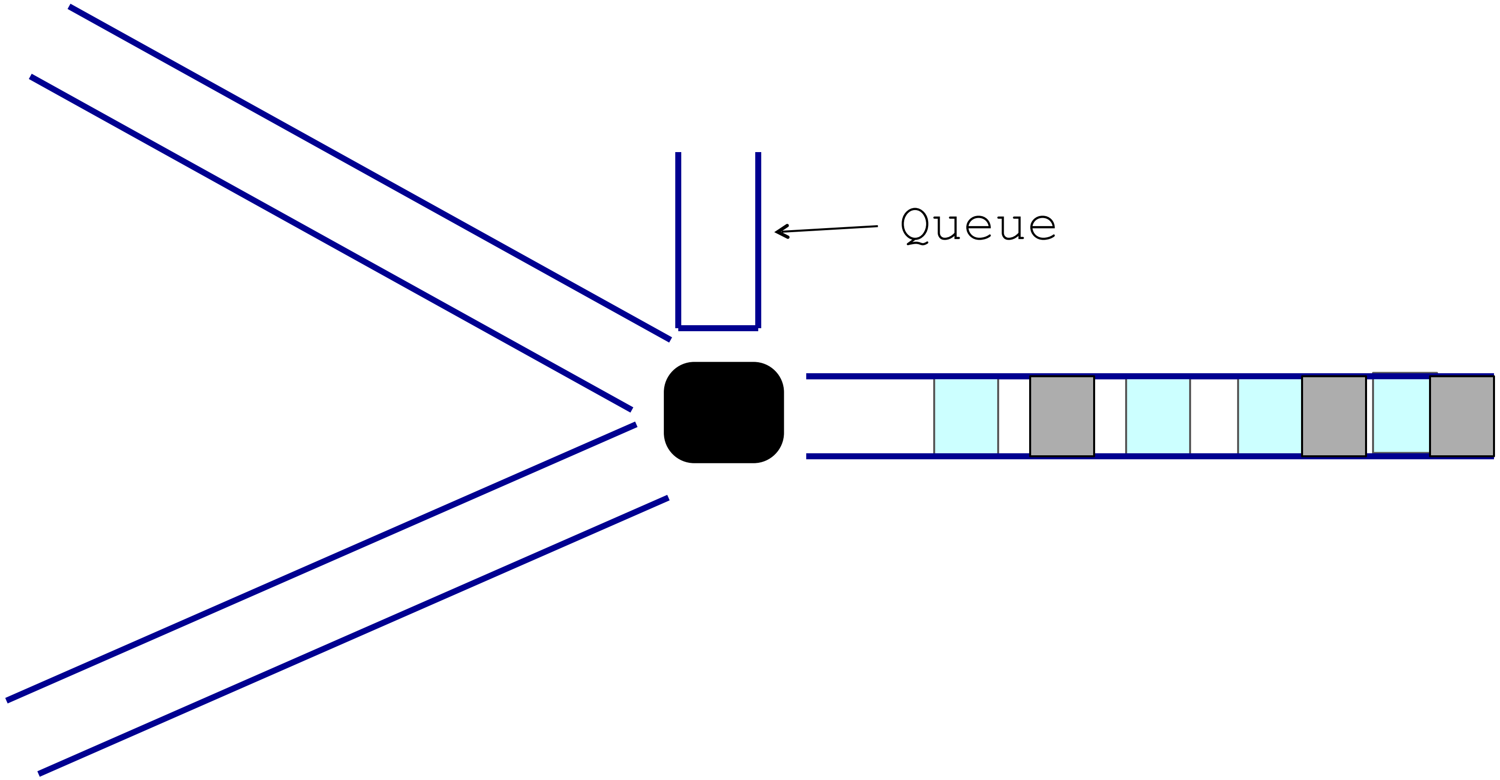
Not a rare event!

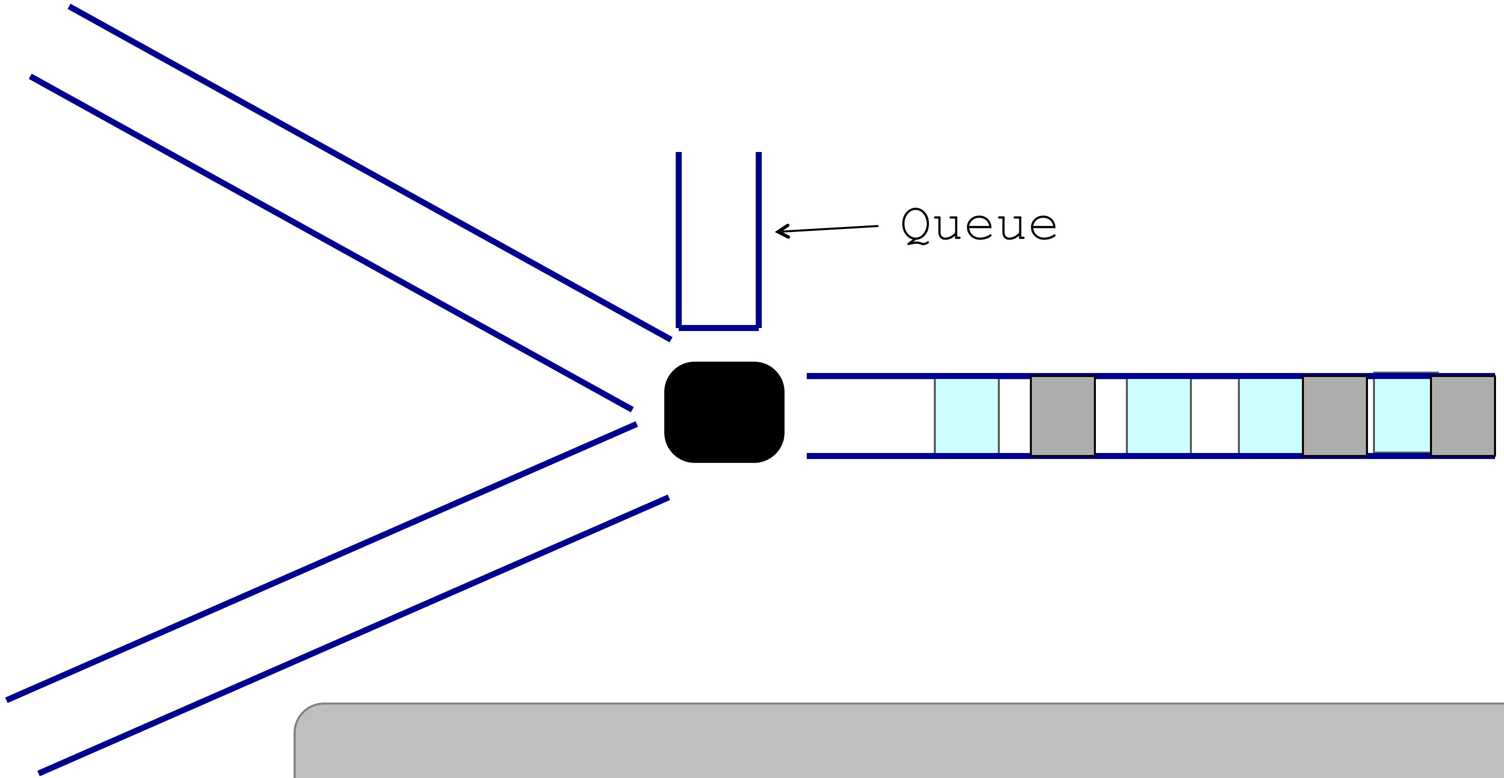

Queue


Queue

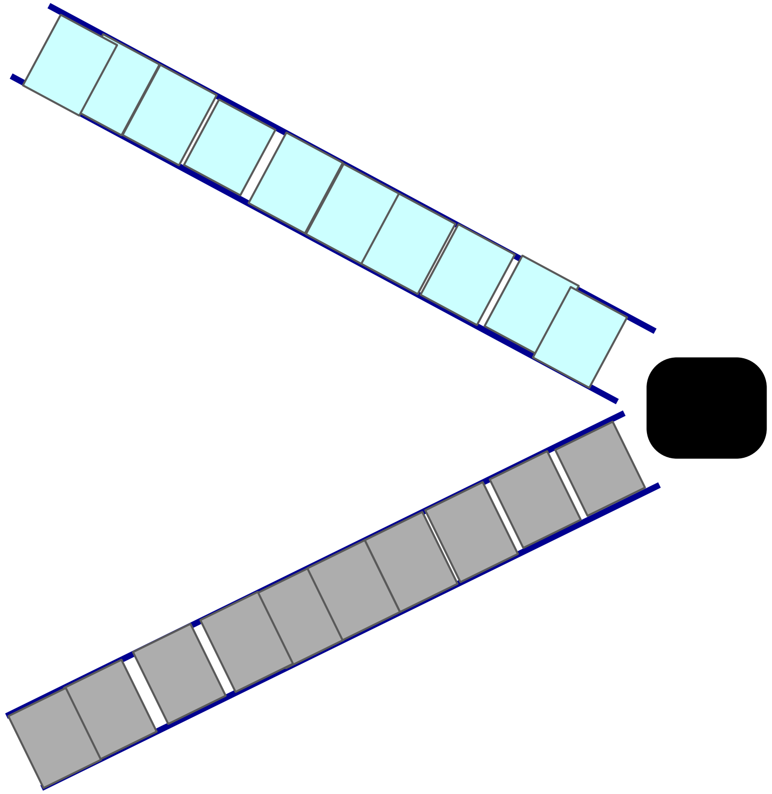

Queue

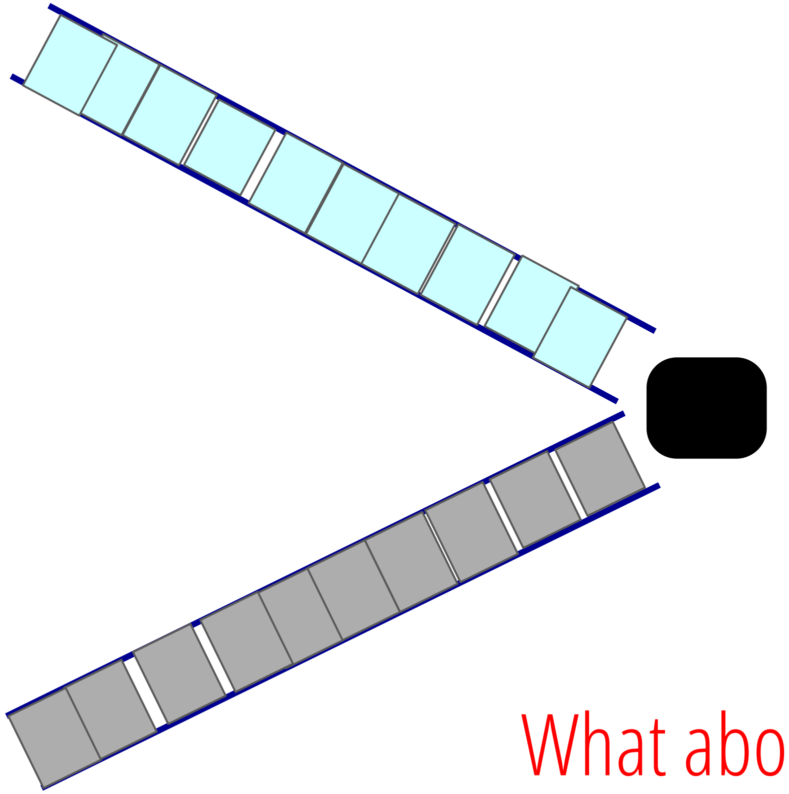

Queue


Queue

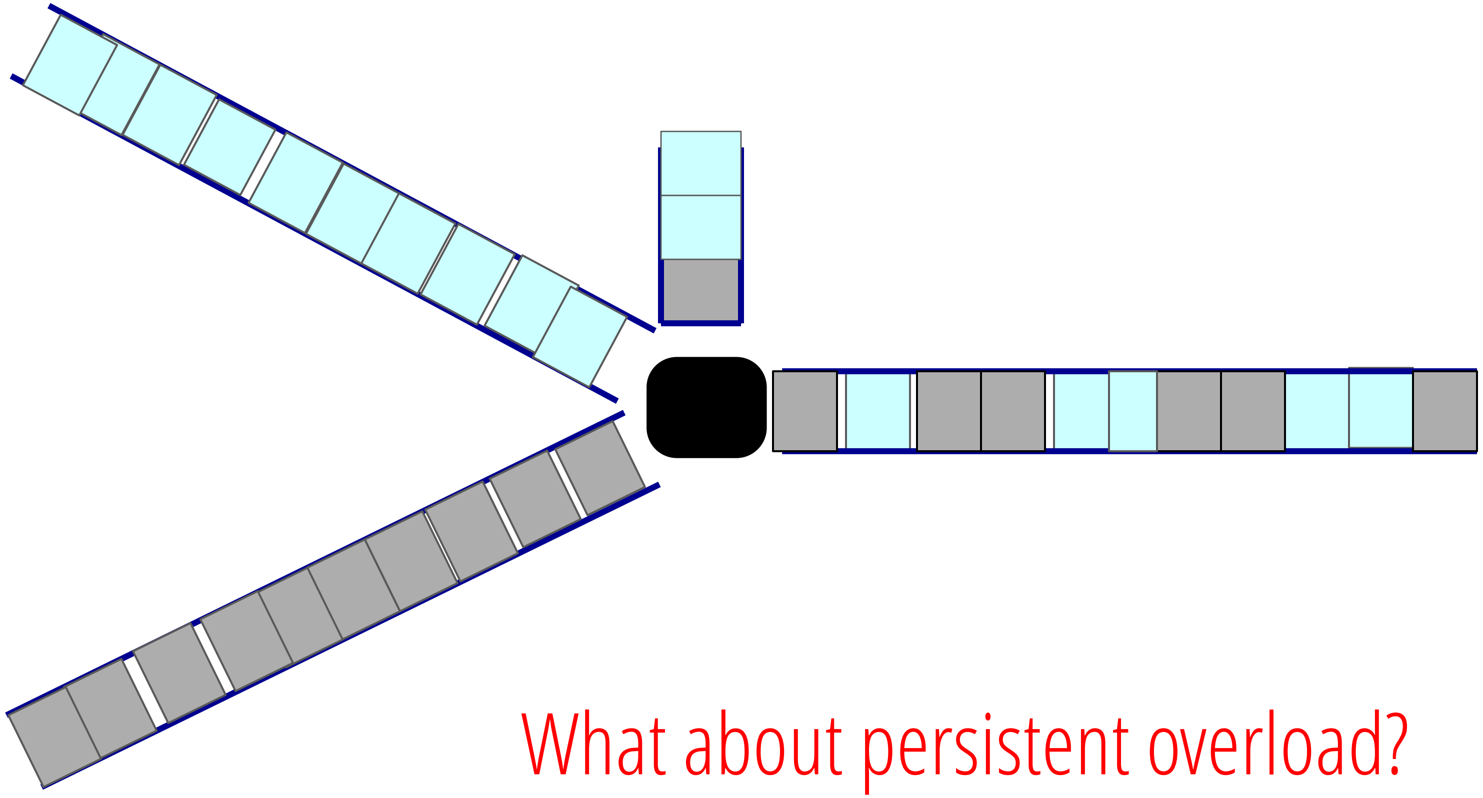



Queues absorb transient bursts!

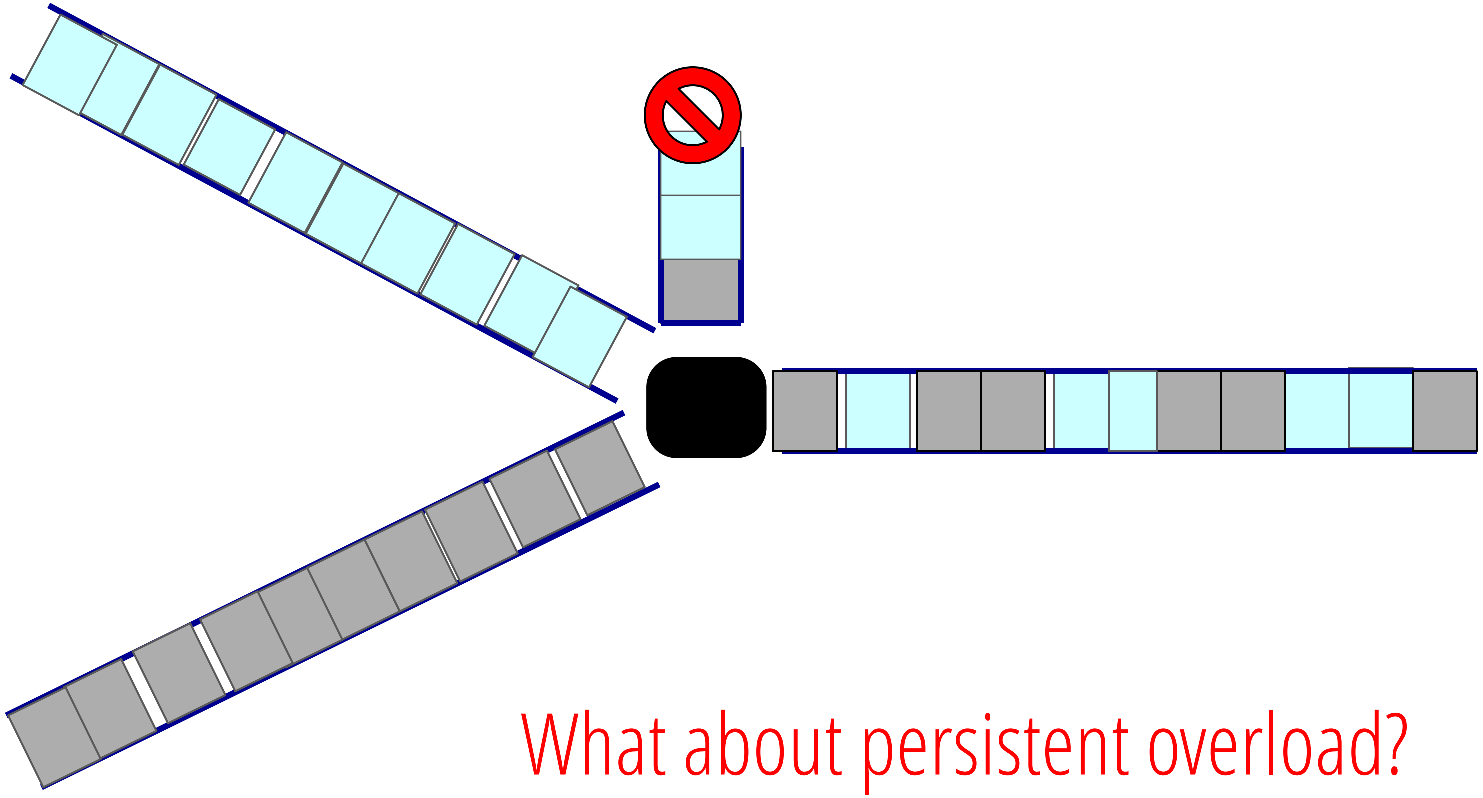




What about persistent overload?



What about persistent overload?



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

# 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 2)
- How do sources know at what rate they can send packets? (congestion control)
- Plus advanced topics (the web, SDN, cellular, datacenters, etc.)

# 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?

# How do you solve a problem?

1. Decompose it (into tasks and abstractions)

# How do you solve a problem?

1. Decompose it (into tasks and abstractions)

# 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?

# What is Modularity?

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

# What is Modularity?

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

- Plays a crucial role in computer science…

# What is Modularity?

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

- Plays a crucial role in computer science…

# 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

# Network Modularity

- The need for modularity still applies

# Network Modularity

- The need for modularity still applies
  - **And is even more important!**

# Network Modularity

- The need for modularity still applies
  - **And is even more important!**

- Normal modularity organizes code

# Network 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…

# Network 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)

# Network 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…

# Inspiration…

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

Dear Sundar,

Your days are numbered.

-- Satya

# Inspiration…

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

# Inspiration…

- **CEO A writes letter to CEO B**
  - Folds letter and hands it to administrative aide
- **Aide:**
  - Puts letter in envelope with CEO B’s full name
  - Takes to FedEx

# Inspiration…

- **CEO A writes letter to CEO B**
  - Folds letter and hands it to administrative aide
- **Aide:**
  - Puts letter in envelope with CEO B’s full name
  - Takes to FedEx
- **FedEx Office**
  - Puts letter in larger envelope
  - Puts name and street address on FedEx envelope
  - Puts package on FedEx delivery truck

# Inspiration…

- **CEO A writes letter to CEO B**
  - Folds letter and hands it to administrative aide
- **Aide:**
  - Puts letter in envelope with CEO B's full name
  - Takes to FedEx
- **FedEx Office**
  - Puts letter in larger envelope
  - Puts name and street address on FedEx envelope
  - Puts package on FedEx delivery truck
- **FedEx delivers to other company**

# The Path of the Letter

CEO

Aide

FedEx

CEO

Aide

FedEx

# The Path of the Letter

CEO

Aide

FedEx

CEO

Aide

FedEx

# The Path of the Letter

CEO

Aide

↓

FedEx

CEO

Aide

FedEx

# The Path of the Letter

| | |
|---|---|
| **CEO** | **CEO** |
| **Aide** | **Aide** |
| **FedEx** → | **FedEx** |

# The Path of the Letter

**CEO**

**Aide**

**FedEx**

**CEO**

**Aide**

**FedEx**

# The Path of the Letter

CEO

Aide

FedEx

CEO

Aide

FedEx

# The Path of the Letter

**CEO**

**Aide**

**FedEx**

**CEO**

**Aide**

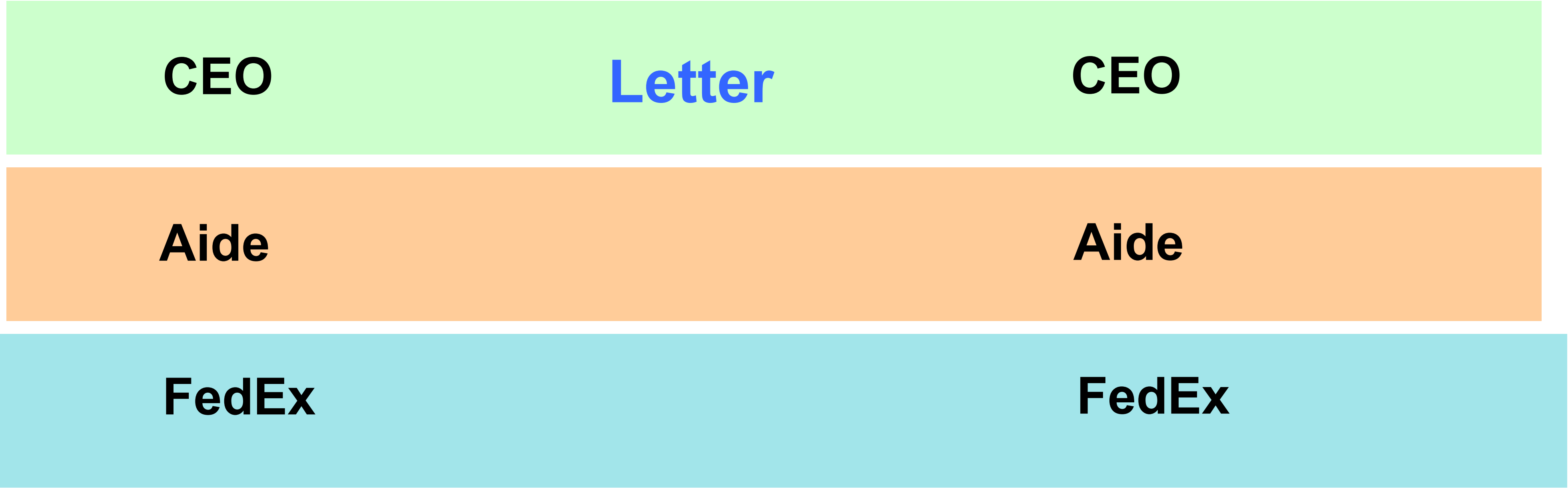
**FedEx**

# The Path of the Letter

| | |
|---|---|
| **CEO** | **CEO** |
| **Aide** | **Aide** |
| **FedEx** | **FedEx** |

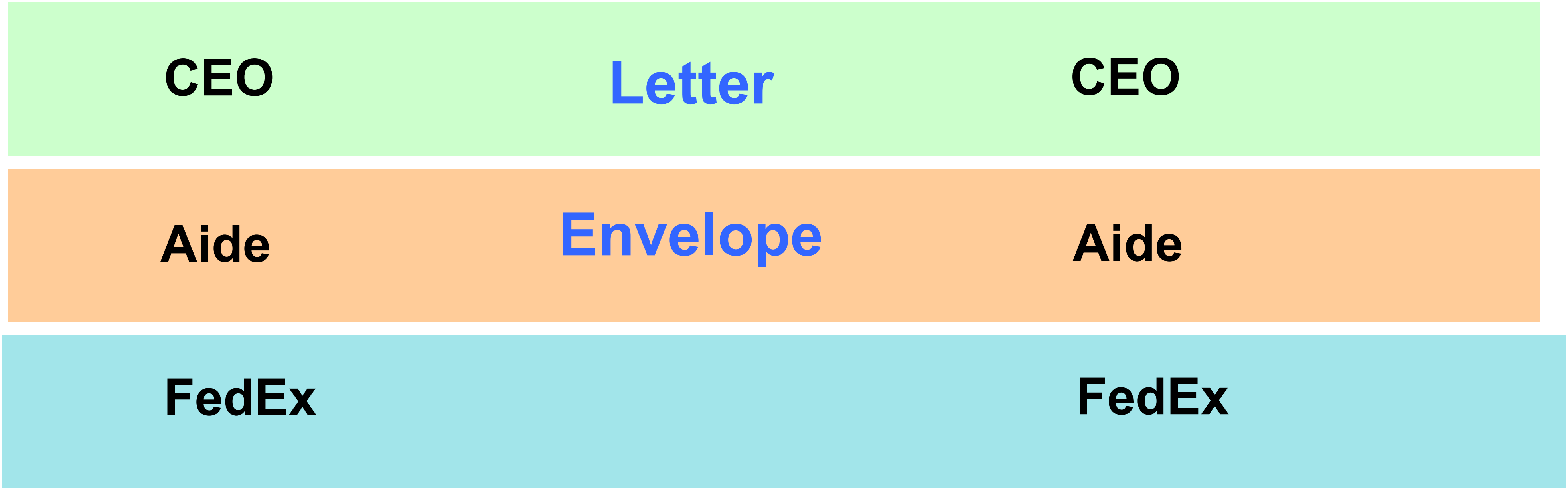
# The Path of the Letter

- “Peers” understand the same things



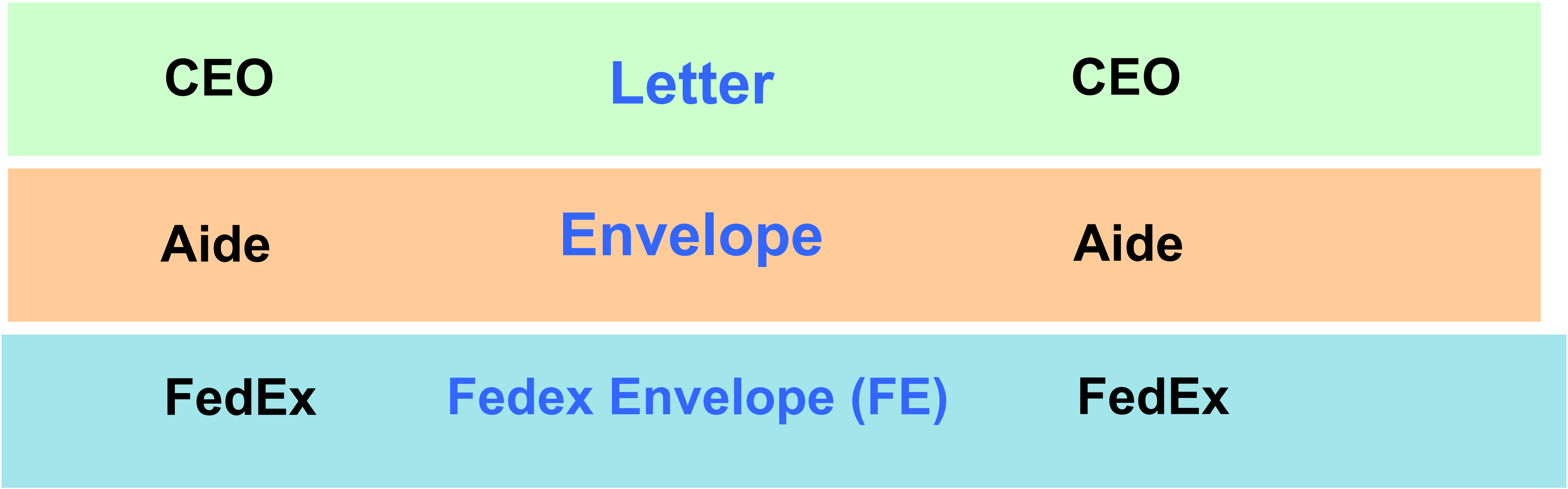
# The Path of the Letter

- “Peers” understand the same things



# The Path of the Letter

- “Peers” understand the same things



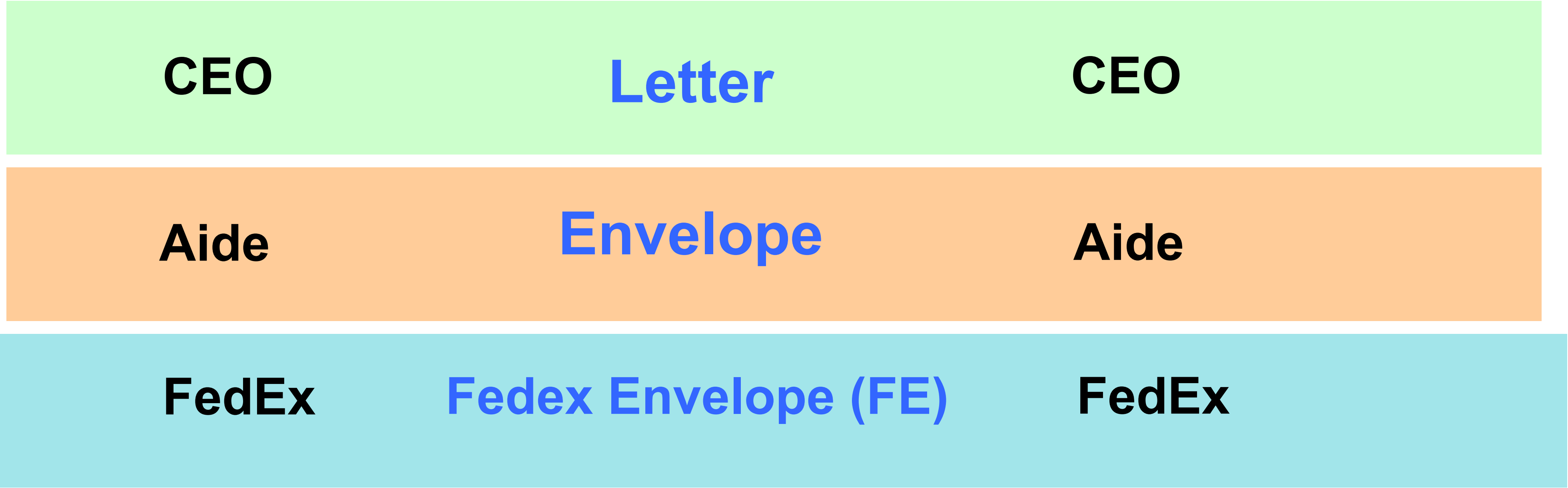
# The Path of the Letter

- “Peers” understand the same things
- No one else needs to

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

# The Path of the Letter

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



# Thought Experiment

# Thought Experiment

- How would ***you*** break the Internet into tasks?
- Just focus on what is needed to get packets between processes on different hosts….

# Thought Experiment

- How would ***you*** break the Internet into tasks?
- Just focus on what is needed to get packets between processes on different hosts….

# 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

# Breakdown into Tasks

- Bits across a link

# Breakdown into Tasks

- Bits across a link
- Packets across a link

# Breakdown into Tasks

- Bits across a link
- Packets across a link
- Deliver packets across local network
  - Local addresses

# Breakdown into Tasks

- Bits across a link
- Packets across a link
- Deliver packets across local network
  - Local addresses
- Deliver packets across multiple networks
  - Global addresses

# Breakdown into Tasks

- Bits across a link
- Packets across a link
- Deliver packets across local network
  - Local addresses
- Deliver packets across multiple networks
  - Global addresses
- Deliver data reliably

# Breakdown into Tasks

- Bits across a link
- Packets across a link
- Deliver packets across local network
  - Local addresses
- Deliver packets across multiple networks
  - Global addresses
- Deliver data reliably
- Do something with the data

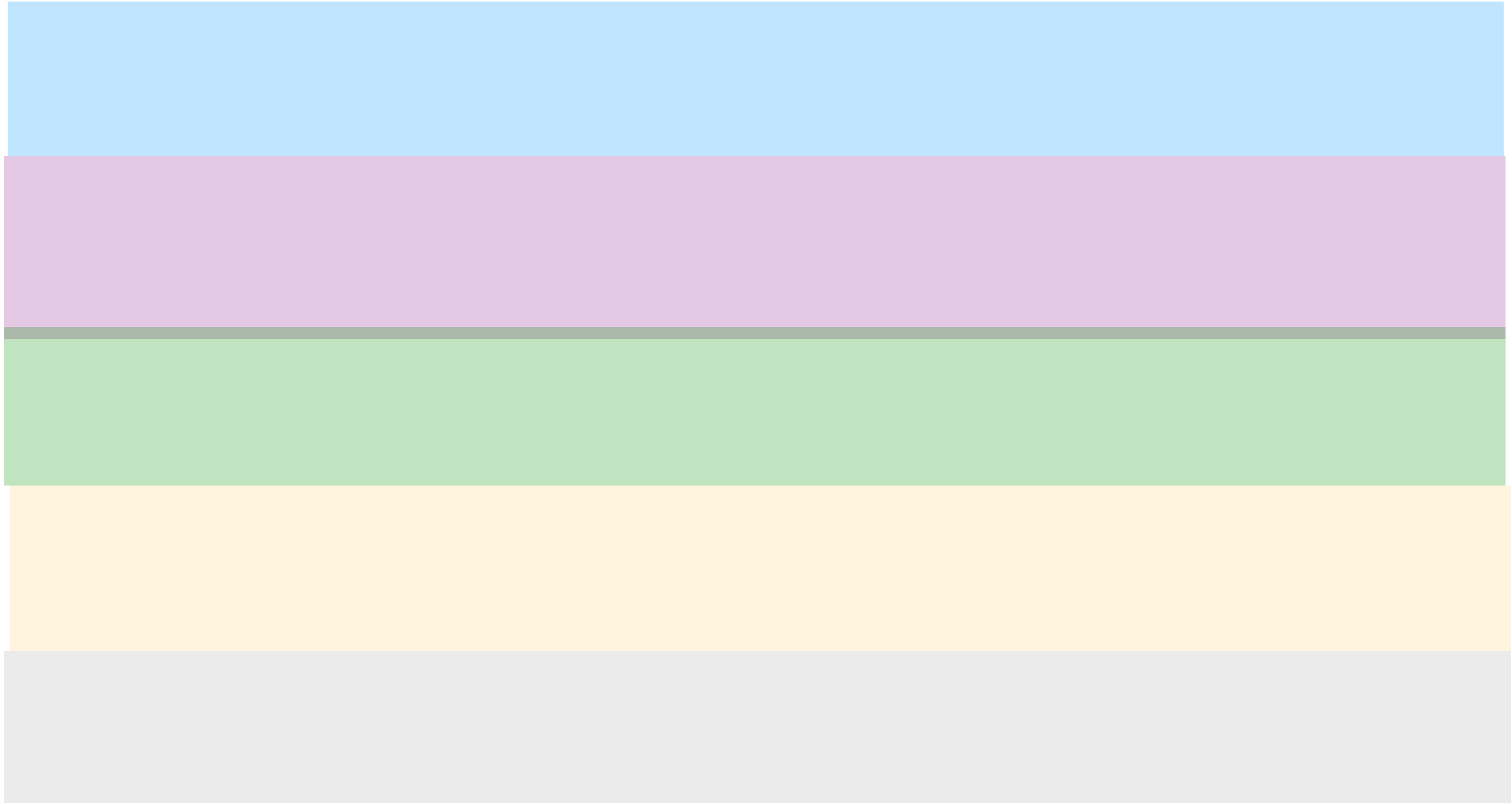
# Breakdown into Tasks

- Bits across a link
- Packets across a link
- Deliver packets across local network
    - Local addresses
- Deliver packets across multiple networks
    - Global addresses
- Deliver data reliably
- Do something with the data

# Breakdown into Tasks

- Bits across a link
- Packets across a link and local network
    - Local addresses
- Deliver packets across multiple networks
    - Global addresses
- Deliver data reliably
- Do something with the data

# In the Internet: organization



# In the Internet: organization

# In the Internet: organization

Best-effort local *packet* delivery

Physical transfer of bits

# In the Internet: organization

**Best-effort *global* packet delivery**

**Best-effort local *packet* delivery**

**Physical transfer of bits**

# In the Internet: organization

Reliable (or unreliable) data delivery

Best-effort *global* packet delivery

Best-effort local *packet* delivery

Physical transfer of bits

# In the Internet: organization

**Applications**

**Reliable (or unreliable) data delivery**

**Best-effort *global* packet delivery**

**Best-effort local *packet* delivery**

**Physical transfer of bits**

# In the Internet: organization

**Applications**

**Reliable (or unreliable) data delivery**

**Best-effort *global* packet delivery**

**Best-effort local *packet* delivery**

**Physical transfer of bits**

# In the Internet: organization

Applications

...built on...

Reliable (or unreliable) data delivery

Best-effort *global* packet delivery

Best-effort local *packet* delivery

Physical transfer of bits

# In the Internet: organization

**Applications**

**…built on…**

**Reliable (or unreliable) data delivery**

**…built on…**

**Best-effort *global* packet delivery**

**Best-effort local *packet* delivery**

**Physical transfer of bits**

# In the Internet: organization

**Applications**

**...built on...**

**Reliable (or unreliable) data delivery**

**...built on...**

**Best-effort *global* packet delivery**

**...built on...**

**Best-effort local *packet* delivery**

**Physical transfer of bits**

# In the Internet: organization

**Applications**

**...built on...**

**Reliable (or unreliable) data delivery**

**...built on...**

**Best-effort *global* packet delivery**

**...built on...**

**Best-effort local *packet* delivery**

**...built on...**

**Physical transfer of bits**

# A <u>layered</u> architecture

- Layer = a part of a system with well-defined interfaces to other parts

# A layered architecture

- Layer = a part of a system with well-defined interfaces to other parts

- **One layer interacts only with layer above and layer below**

# 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

**Applications**

...built on...

**Reliable (or unreliable) data delivery**

...built on...

**Best-effort global packet delivery**

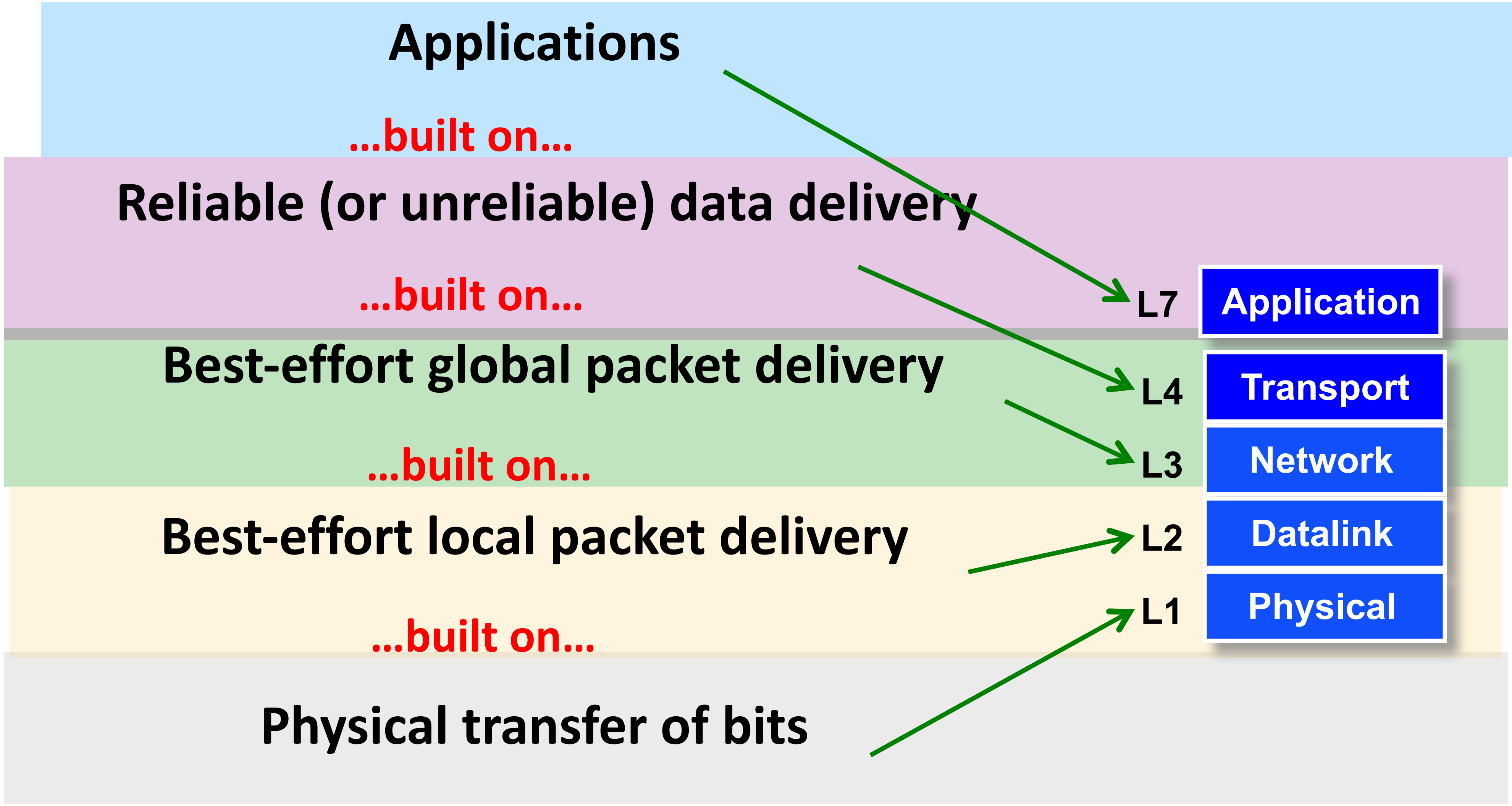
...built on...

**Best-effort local packet delivery**

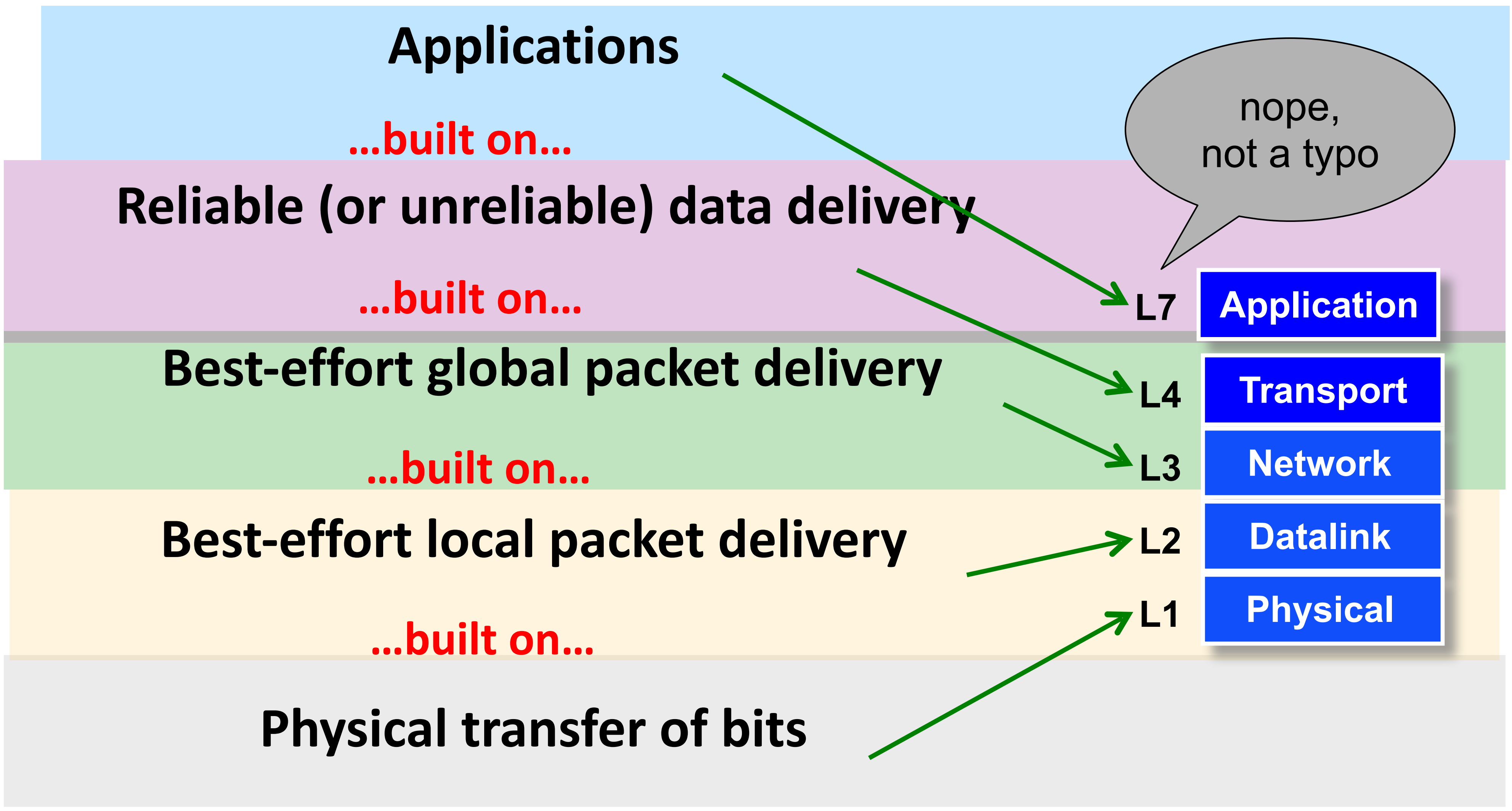
...built on...

**Physical transfer of bits**

# In the Internet: organization

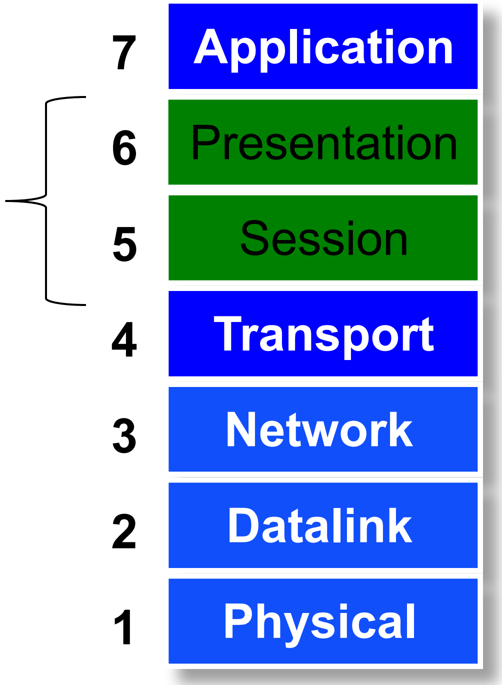


# In the Internet: organization



# 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?