CS168, Lecture 3

How the Internet Works : A bottom-up view (contd.)

Sylvia Ratnasamy

Spring 2020

Today

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

Recall, from last lecture…

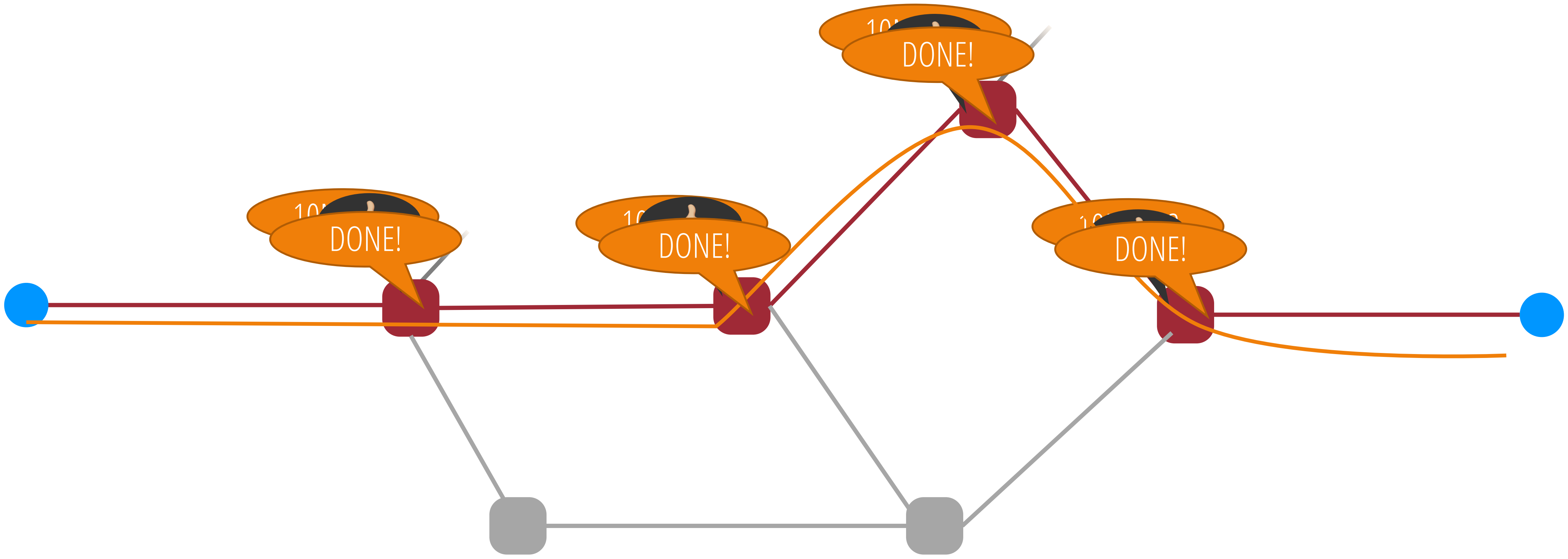
Two canonical approaches to sharing

- **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

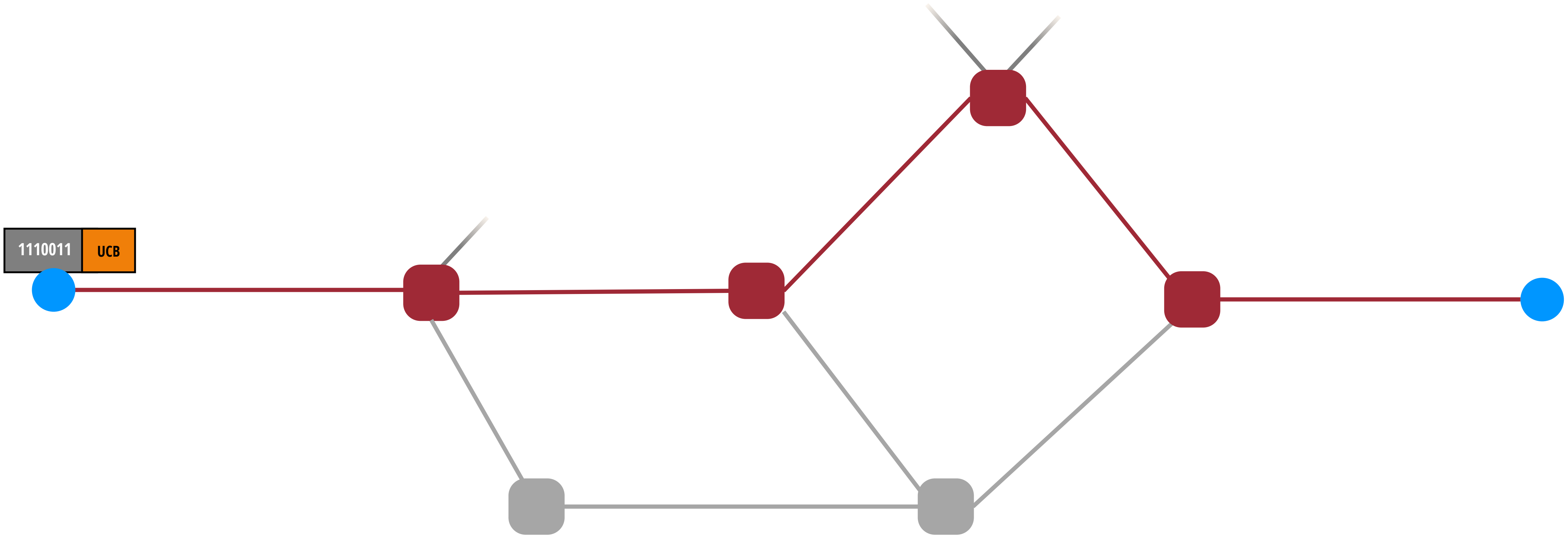
- Reservations via **circuit switching**
- Best-effort via **packet switching**

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: 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
  - Complexity of implementation

Which makes more efficient use of network capacity?

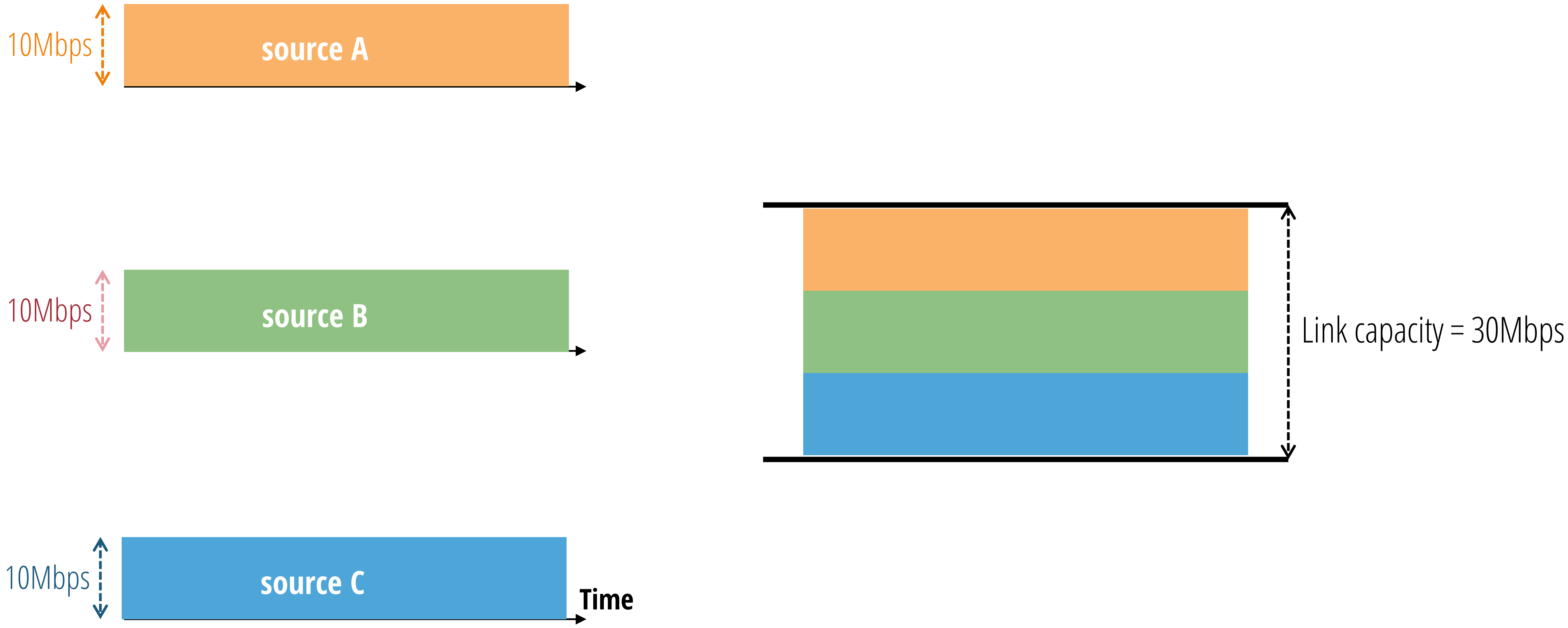
Answer: Packet switching is typically more efficient

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

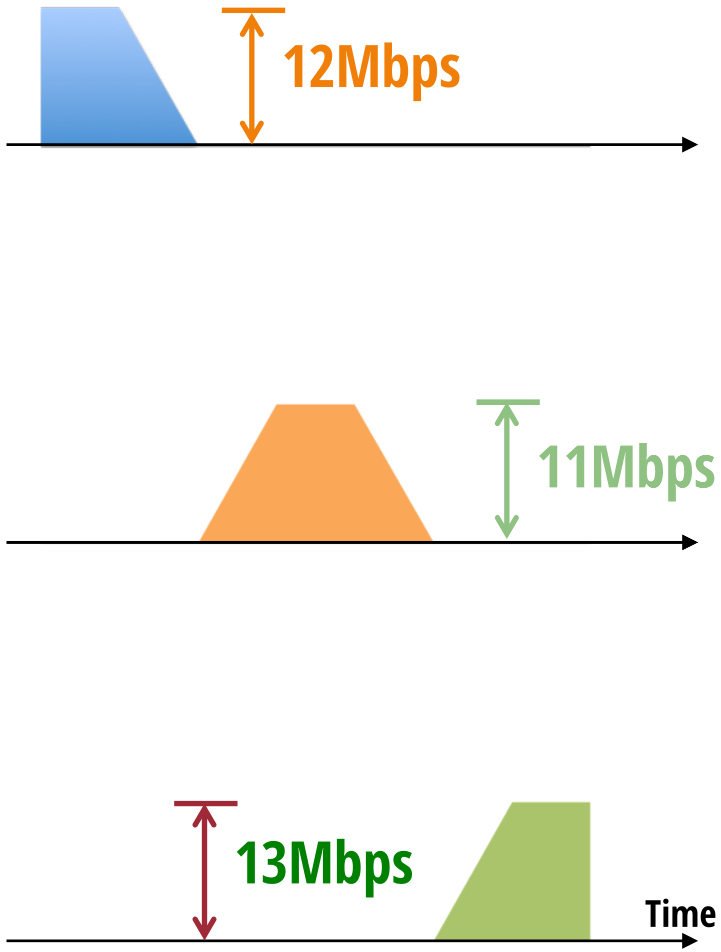
Example#1: Three constant rate sources sharing a link

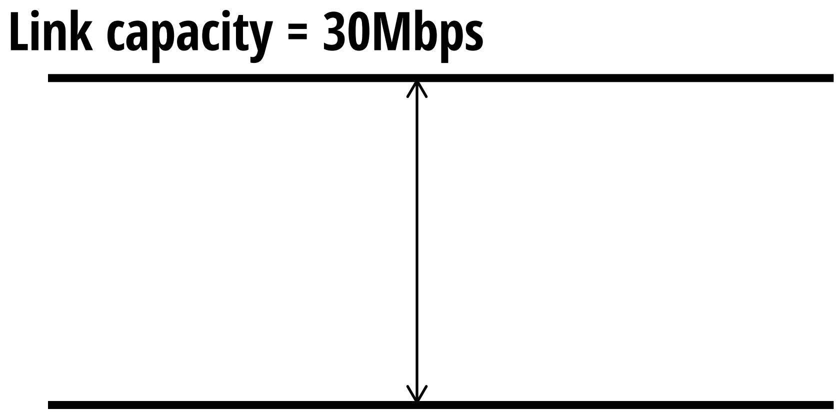
- Total link bandwidth is 30Mbps
- Demands: Each source needs a constant rate of 10Mbps
- Circuit and packet switching give approximately the same result
  - Every source gets what they need
  - No wasted bandwidth
  - ....

Example#1: Three constant sources



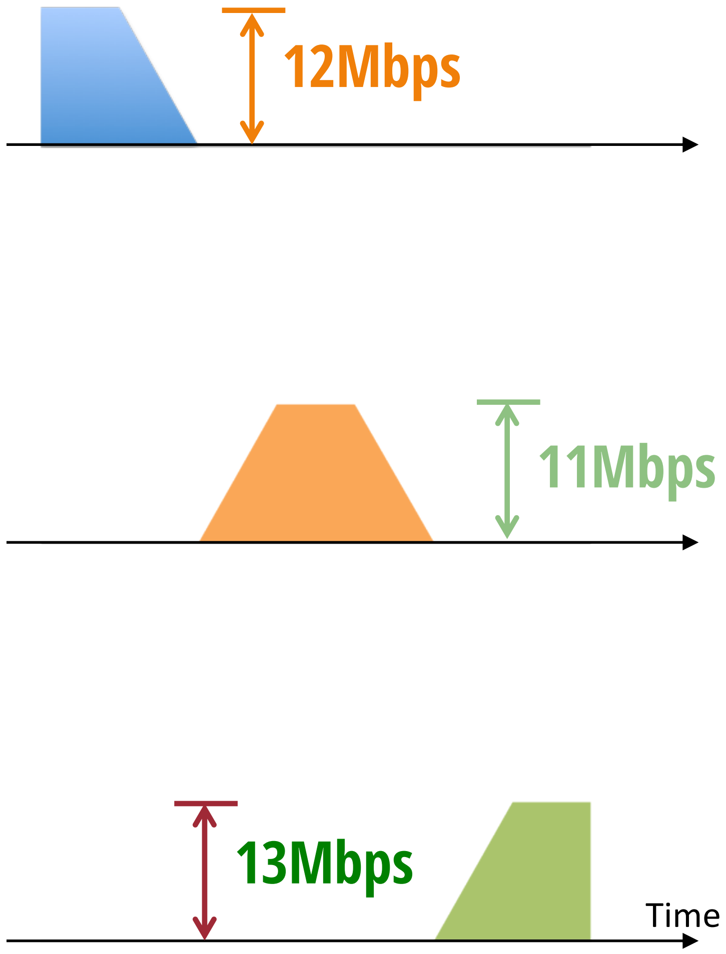
Example#2: Three “bursty” sources

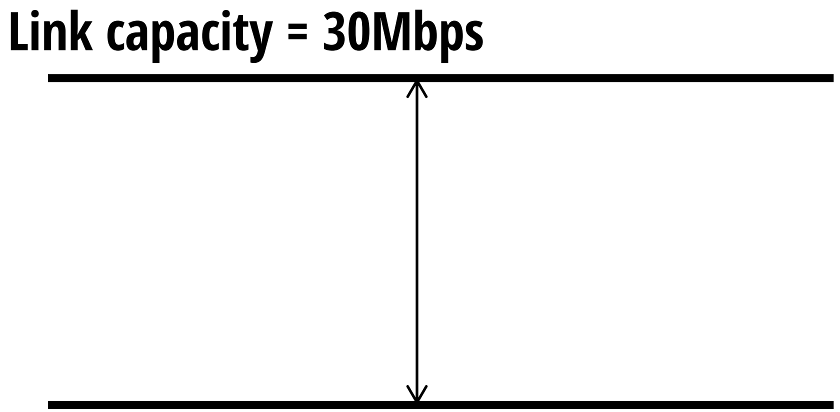




What happens with reservations?

Option #1: allow two flows to reserve peak rate

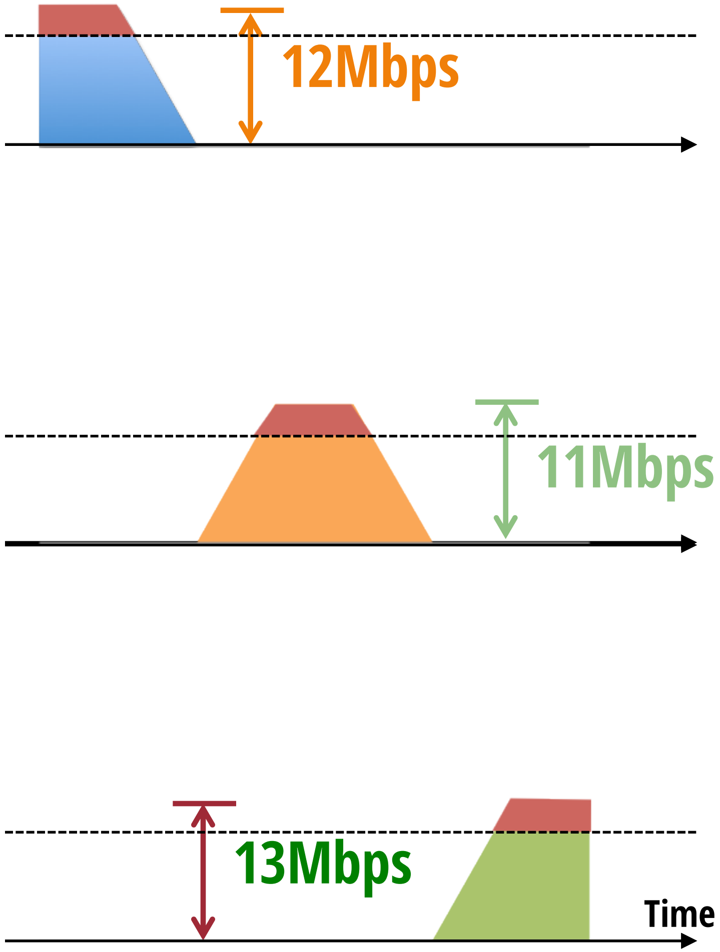




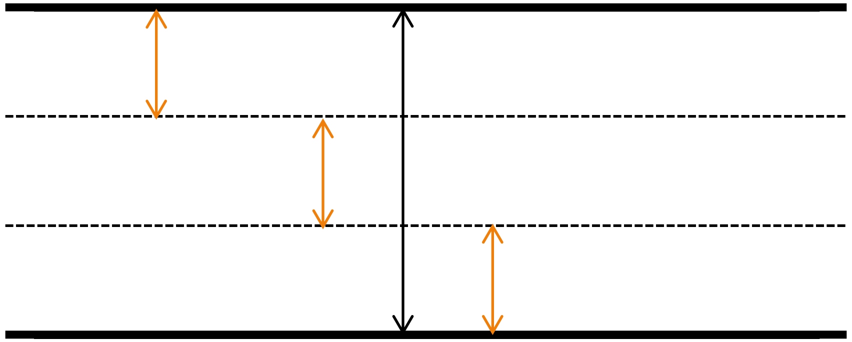
Must turn away third flow!

What happens with reservations?

Option #2: allow flows to reserve equal rates



Link capacity = 30Mbps

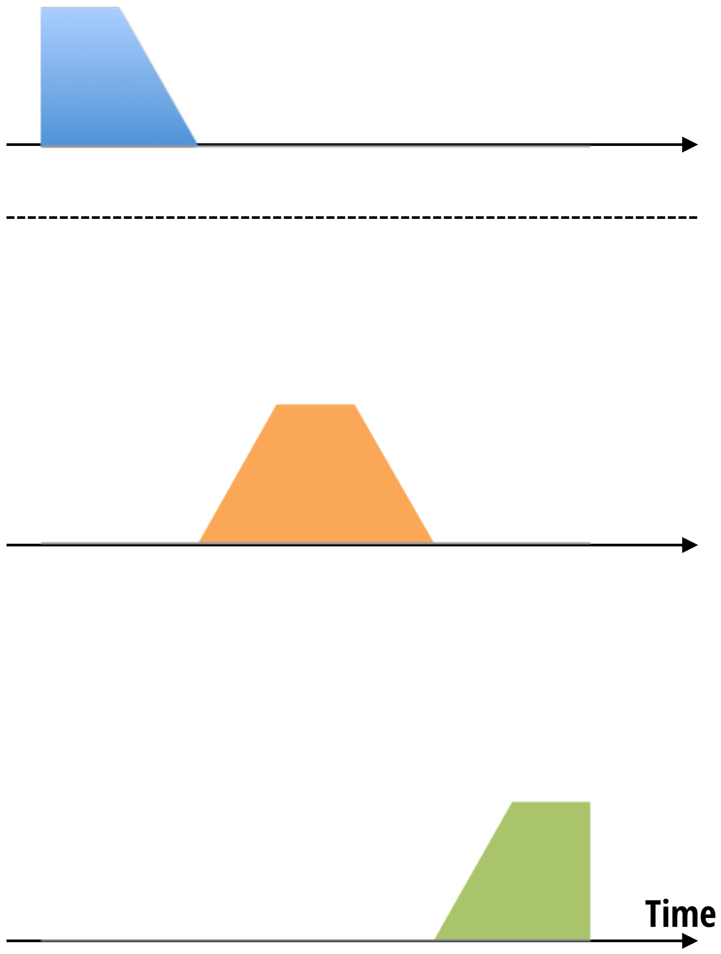


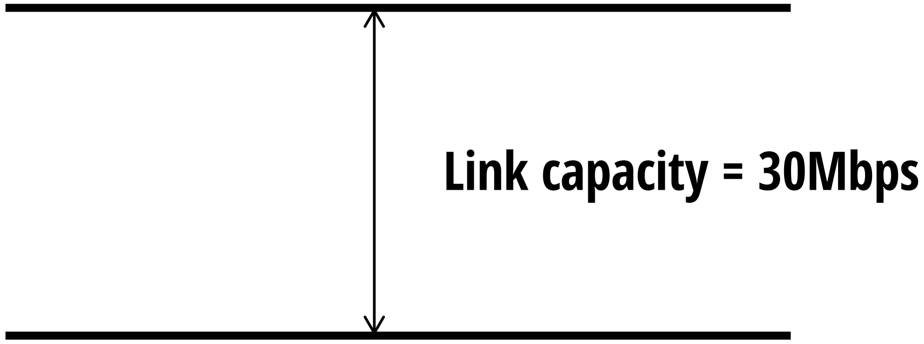
Each source gets 10Mbps

Frequent overloading!

What happens with best-effort?

All good! No overloading





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
  - Ratios of 100 or greater are common

- 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
- This is because packet switching implements statistical multiplexing at a finer granularity than circuit switching (packets vs. flows)

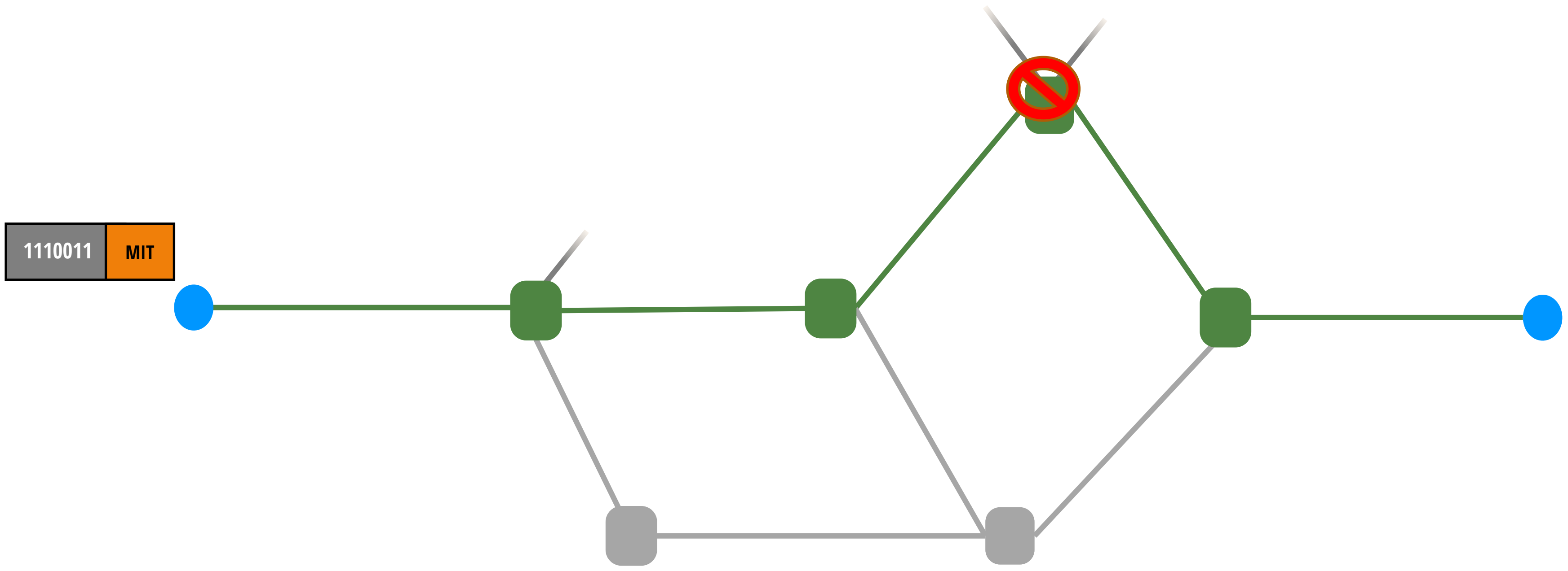
Other differences in efficiency?

- 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?

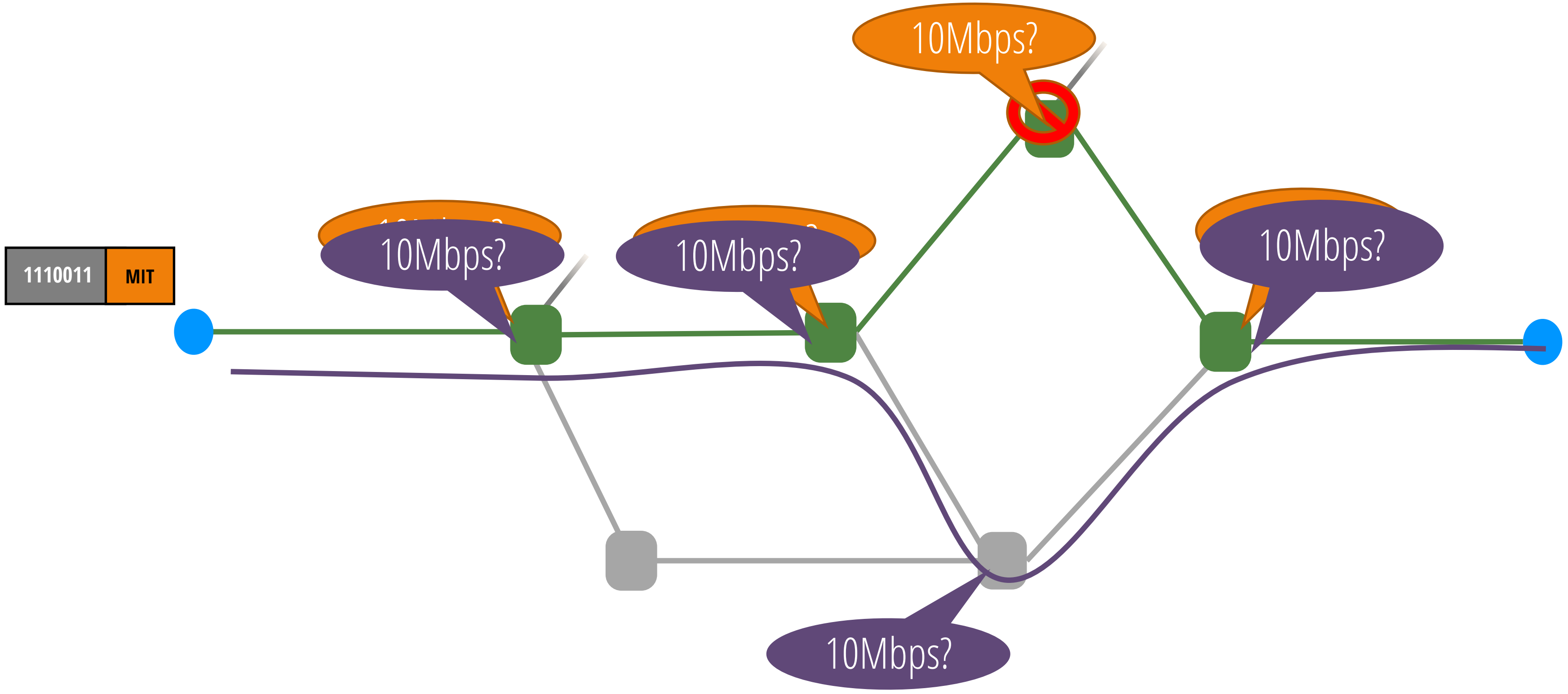
- What are the dimensions along which we should compare?
  - As an abstraction to applications (endhosts)
  - Efficiency
  - Handling failures
  - Complexity of implementation

What happens in the event of a failure?



With packet switching?

What happens in the event of a failure?



With circuit switching?

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

- 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!!

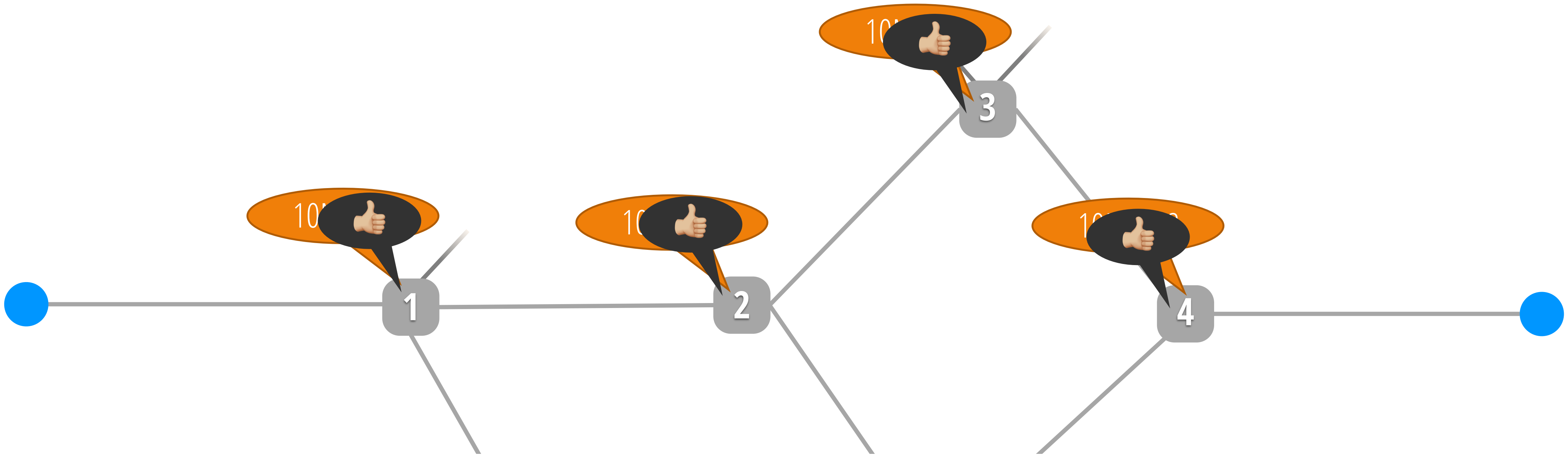
- If millions of flows were going through a switch, then millions of reservation requests are being simultaneously re-established!

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
  - Complexity of implementation

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)

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!

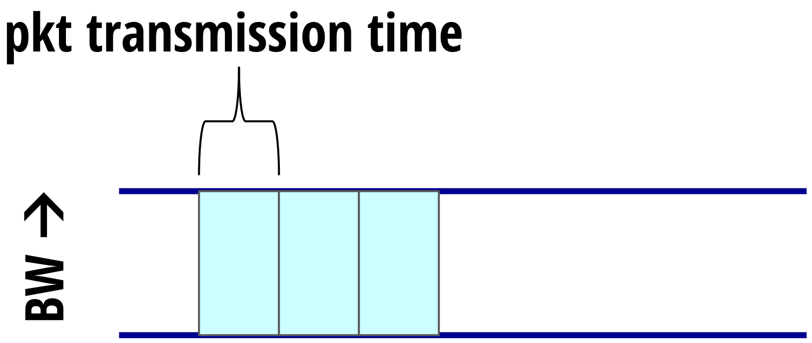
What does the Internet use today?

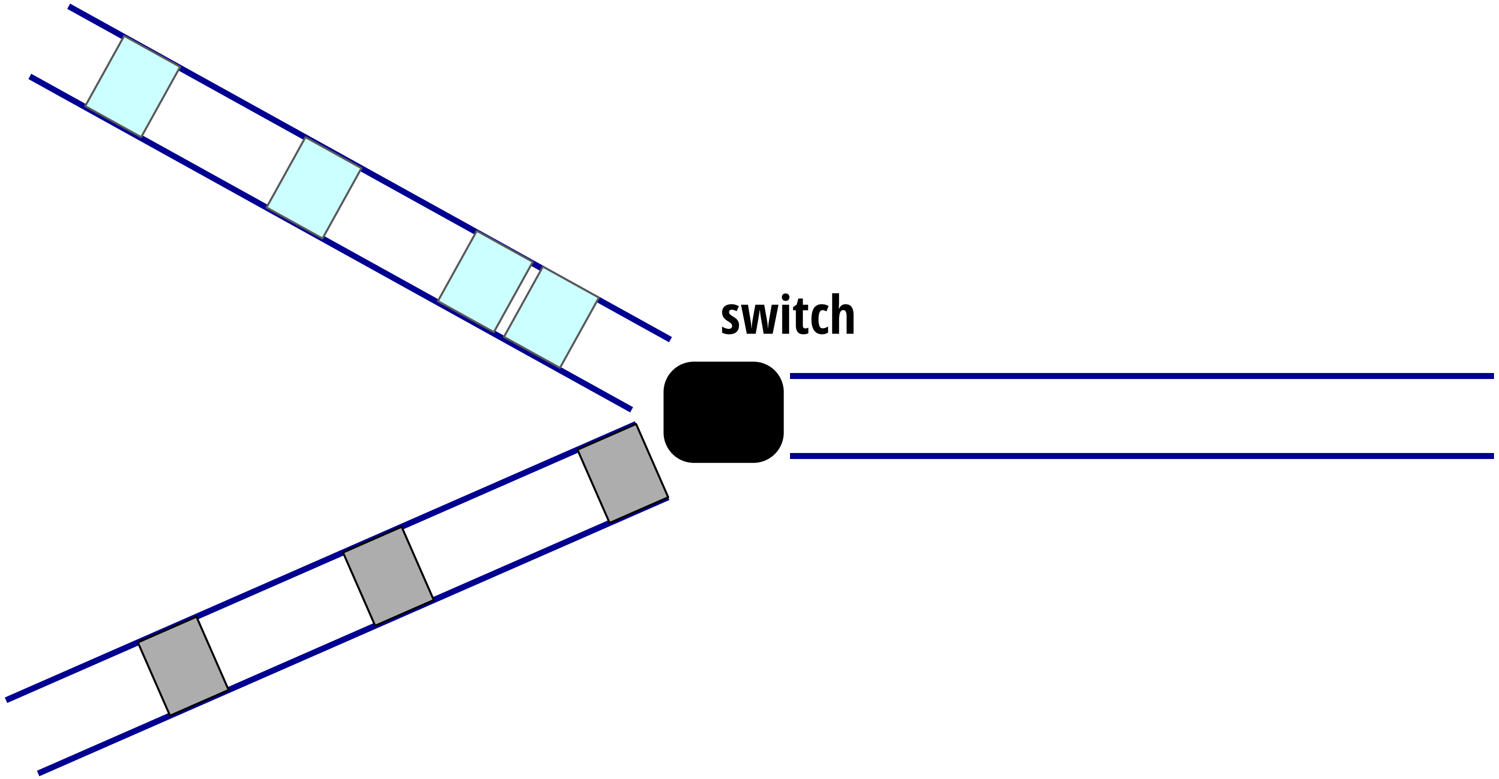
- Packet switching is the default → **the "public" Internet**
- But you *can* also buy a reserved circuit (MPLS circuits, leased lines, *etc.*)
  - Often used by enterprises from one branch location to another (or to/from cloud)
  - Very expensive (e.g., 10-20x higher than a normal connection)
  - Statically (often manually) setup between two locations (e.g., month+ setup times)
  - So, a far cry from the vision of dynamic reservations that we just discussed

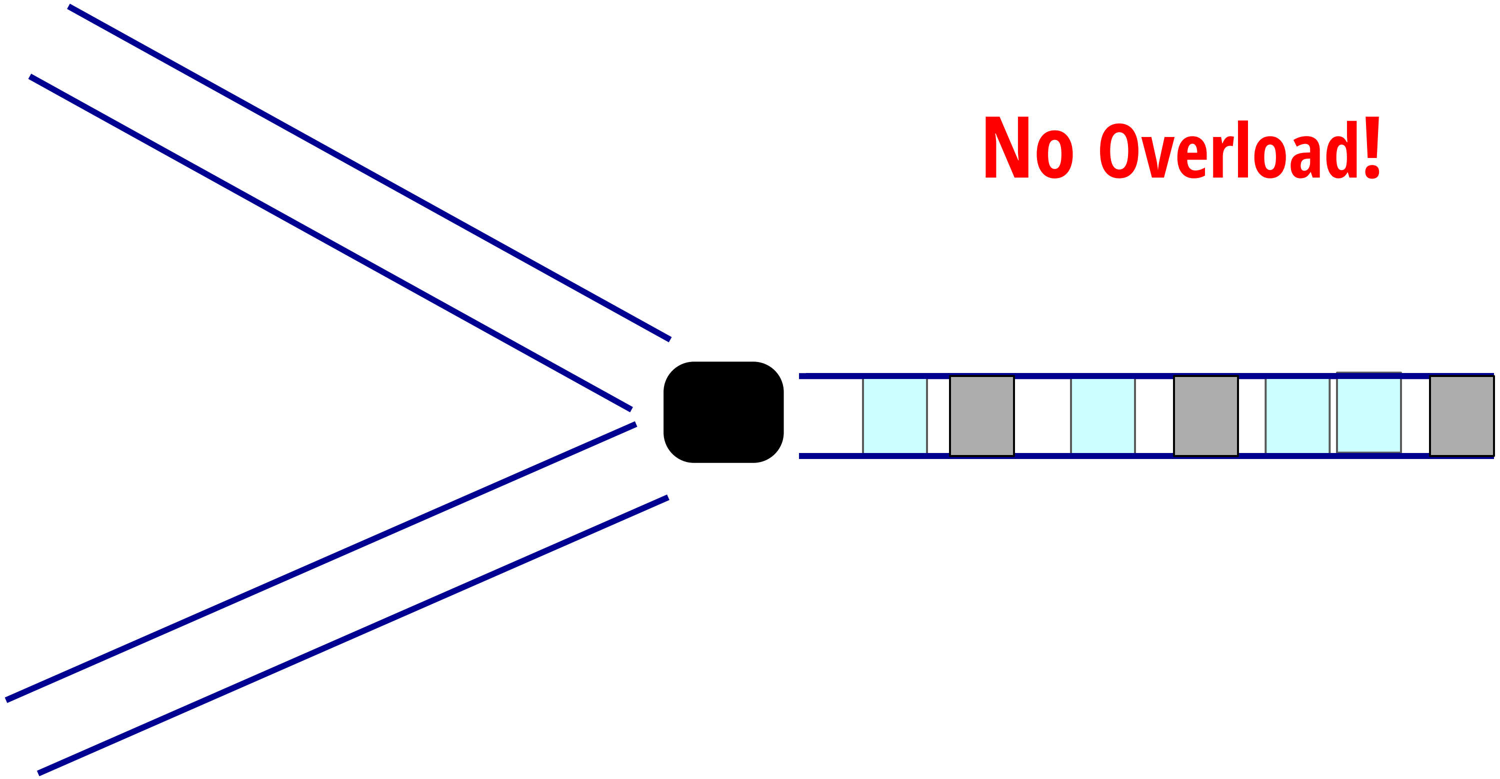
Questions??

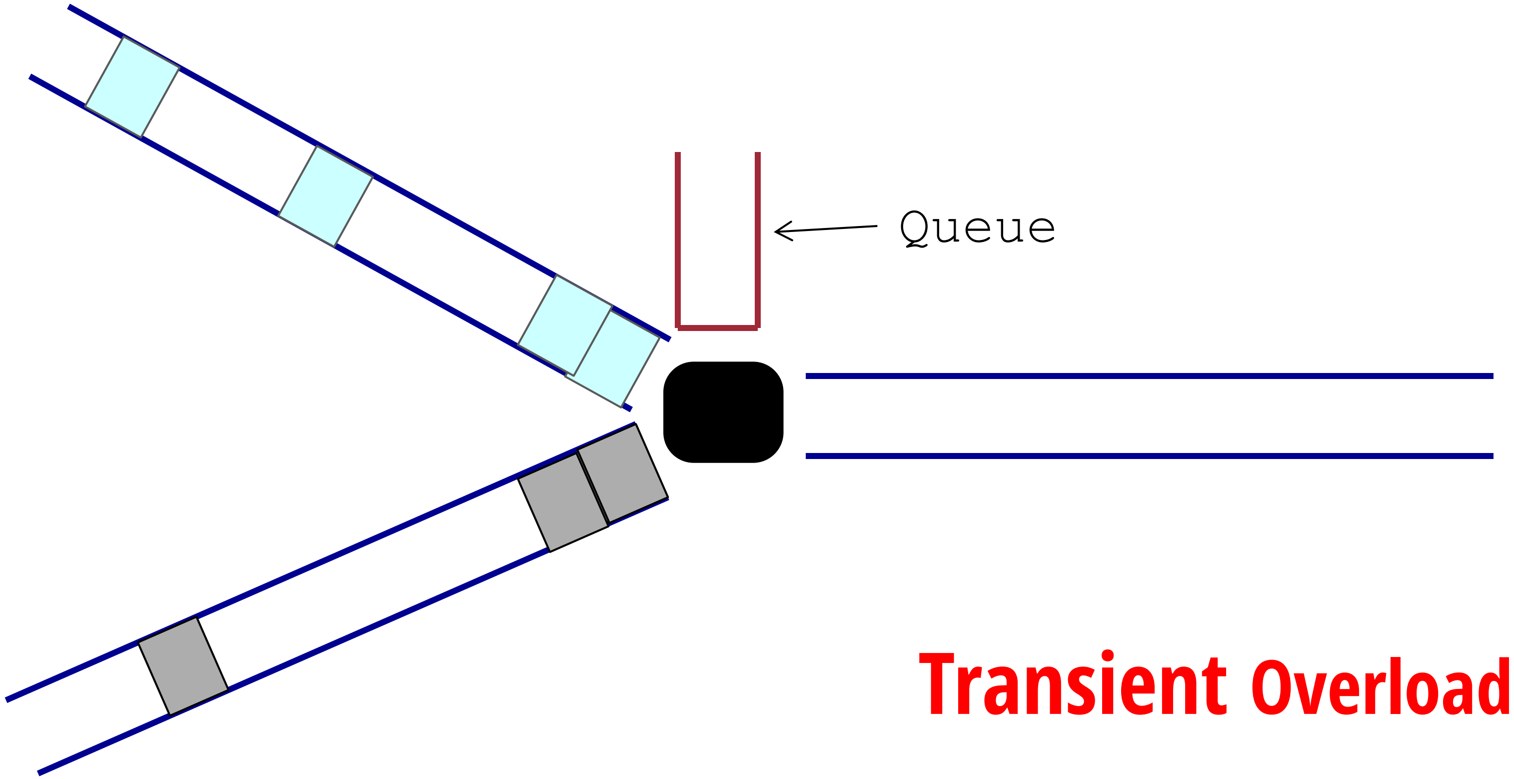
Let’s take a closer look at packet switching ….

Recall, packets in flight: “pipe” view



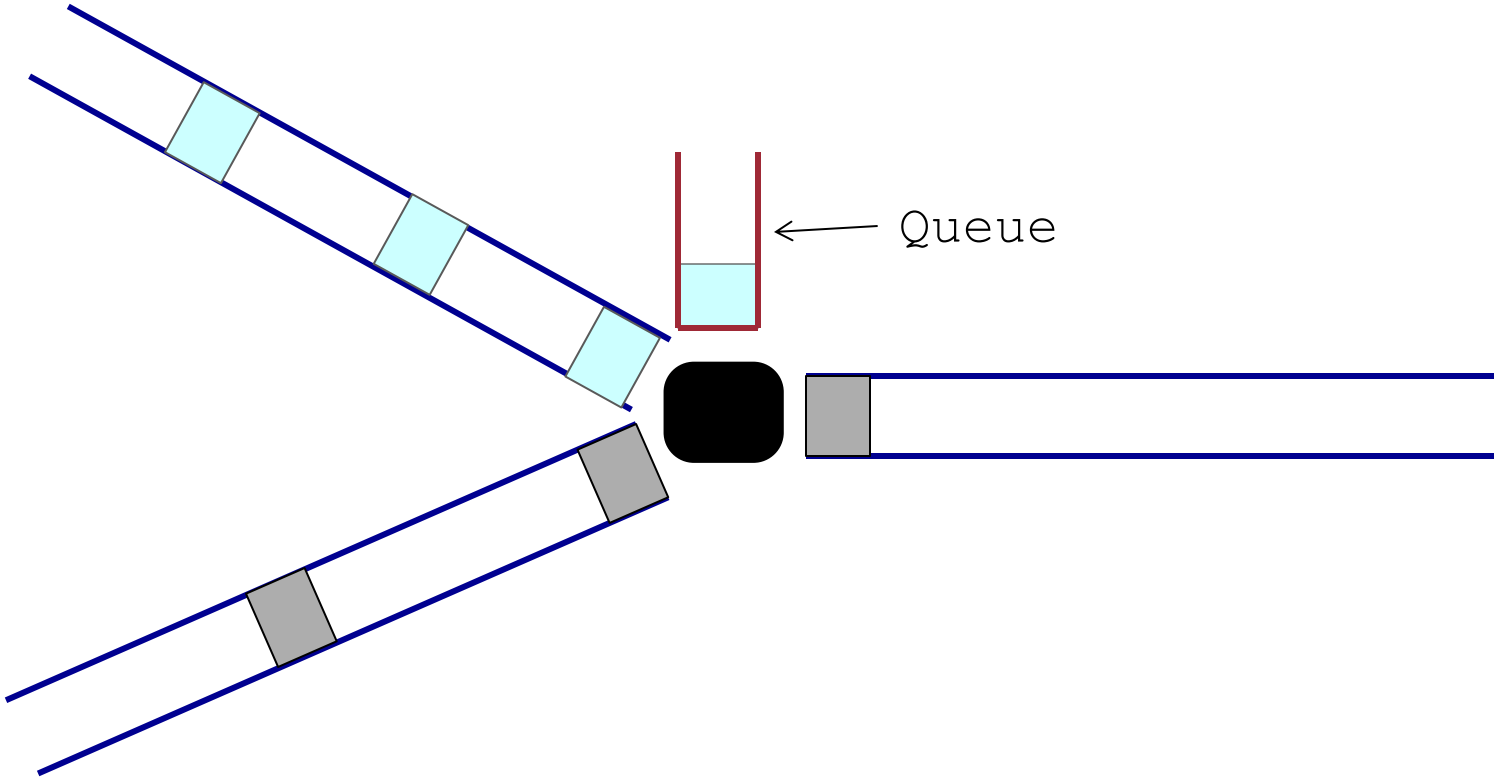

switch

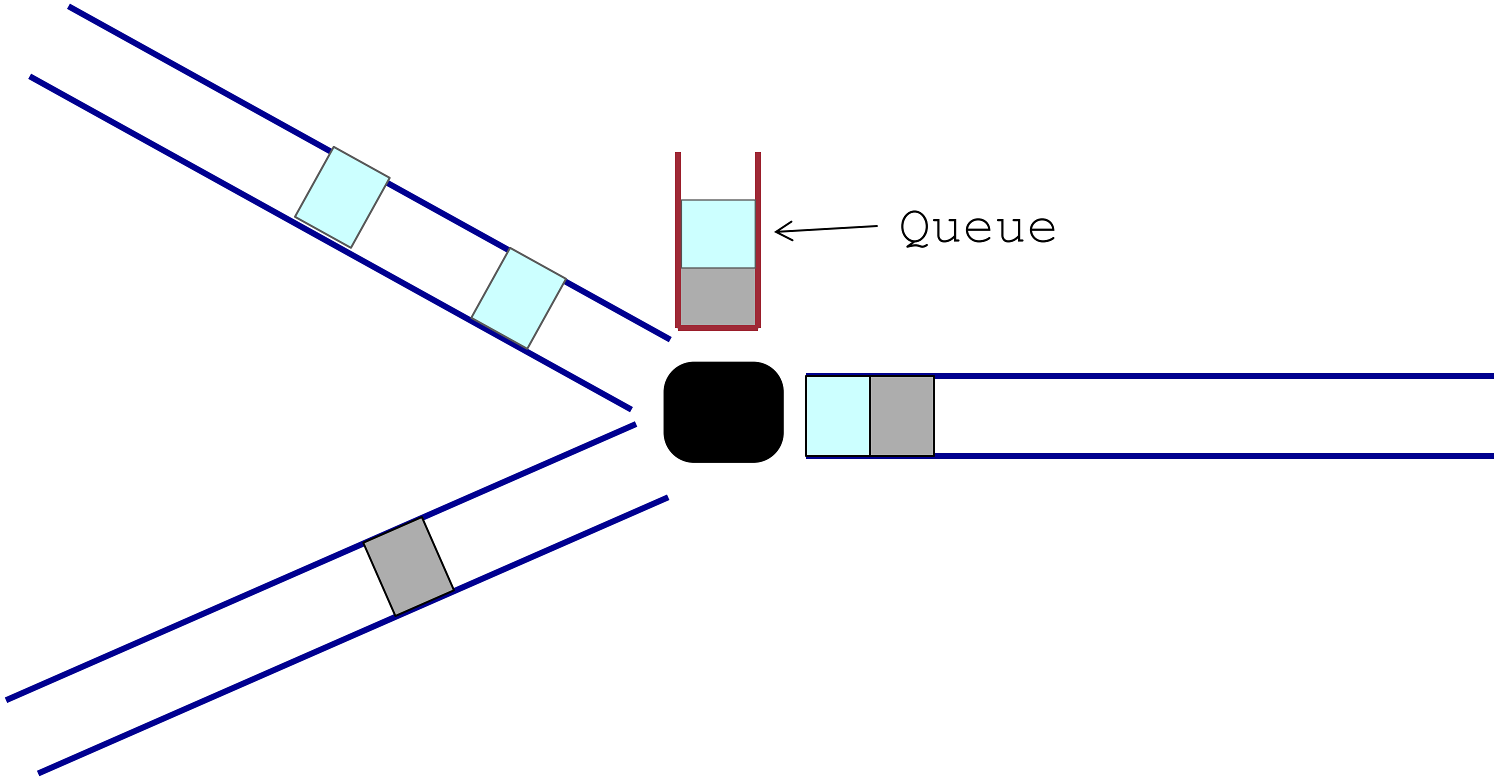

No Overload!

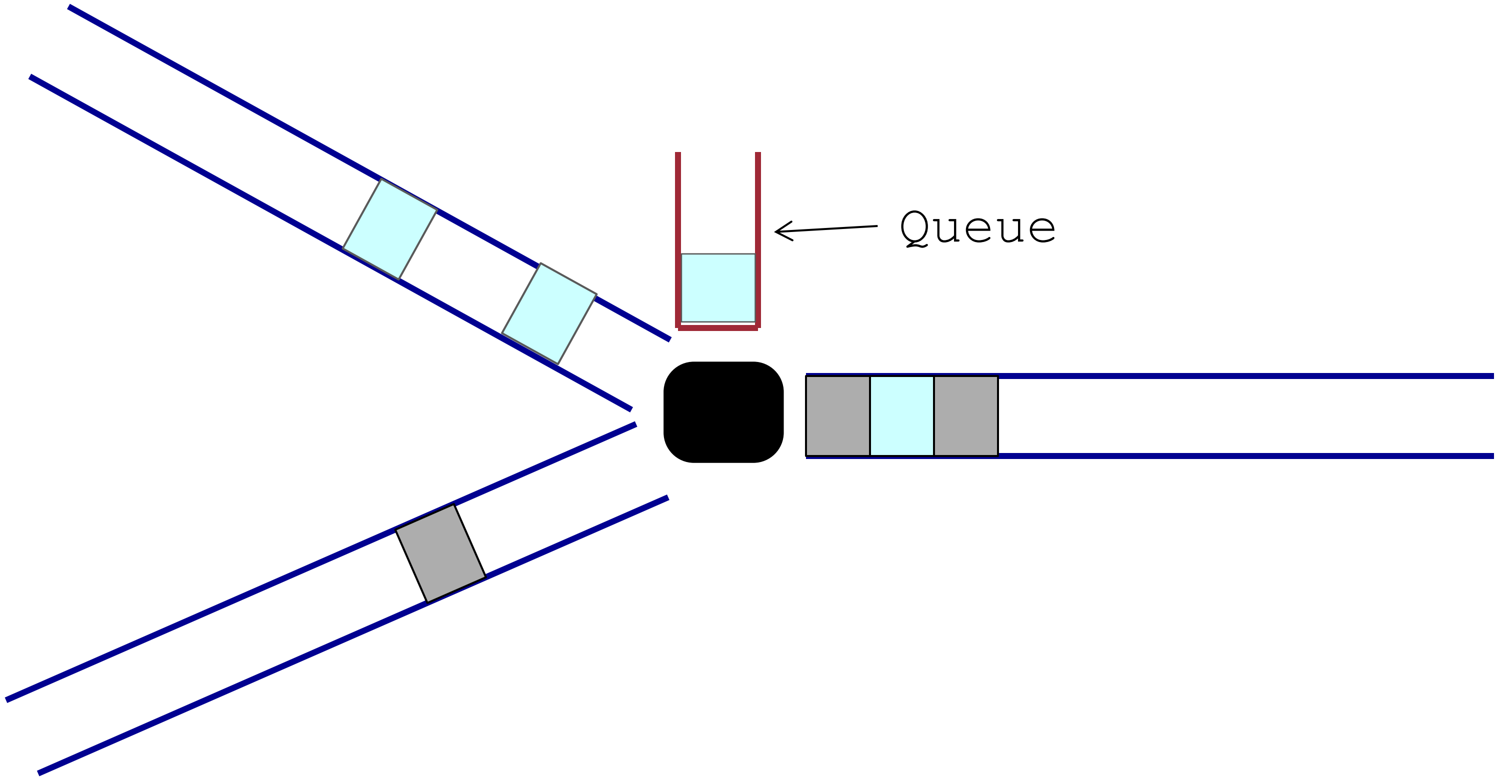


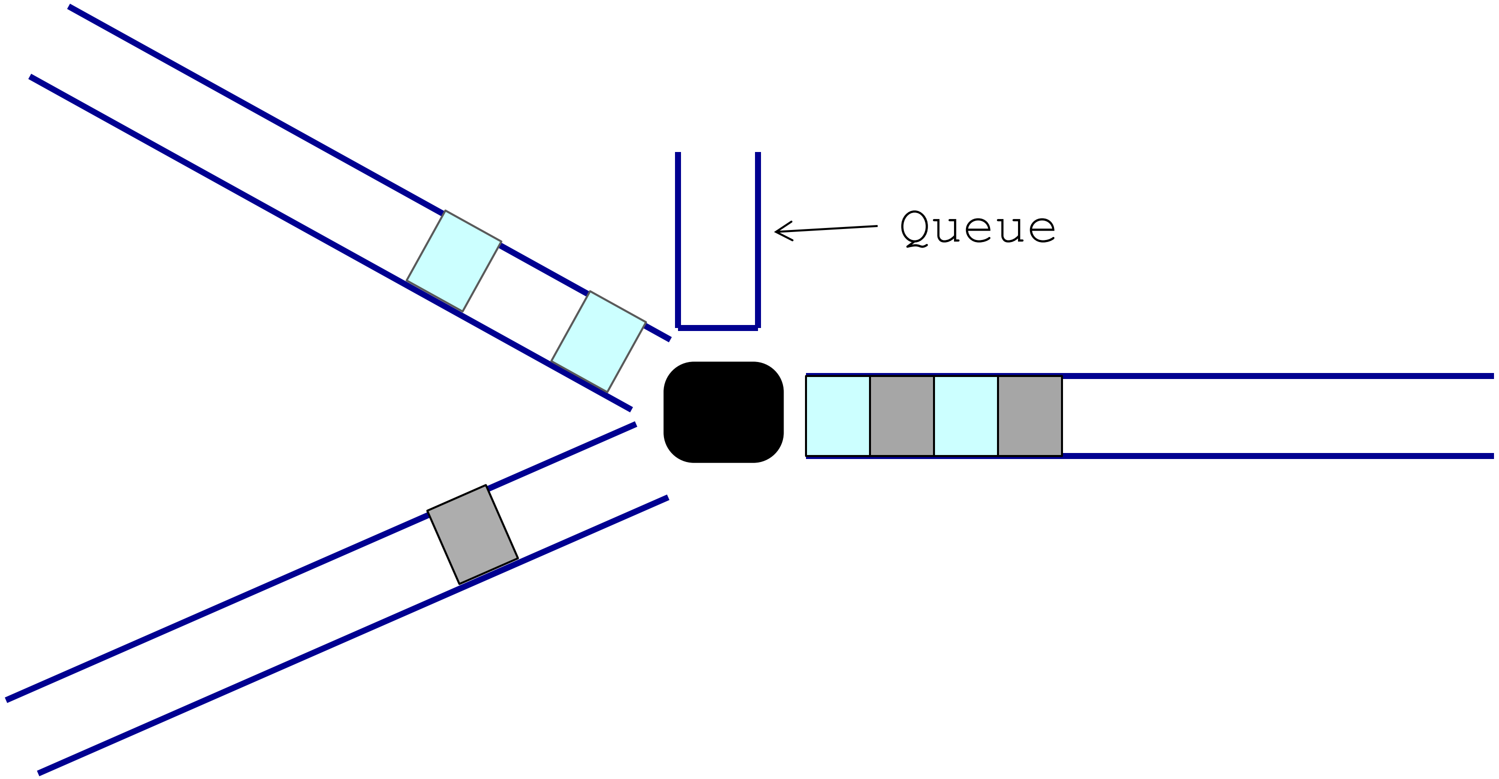
Transient Overload

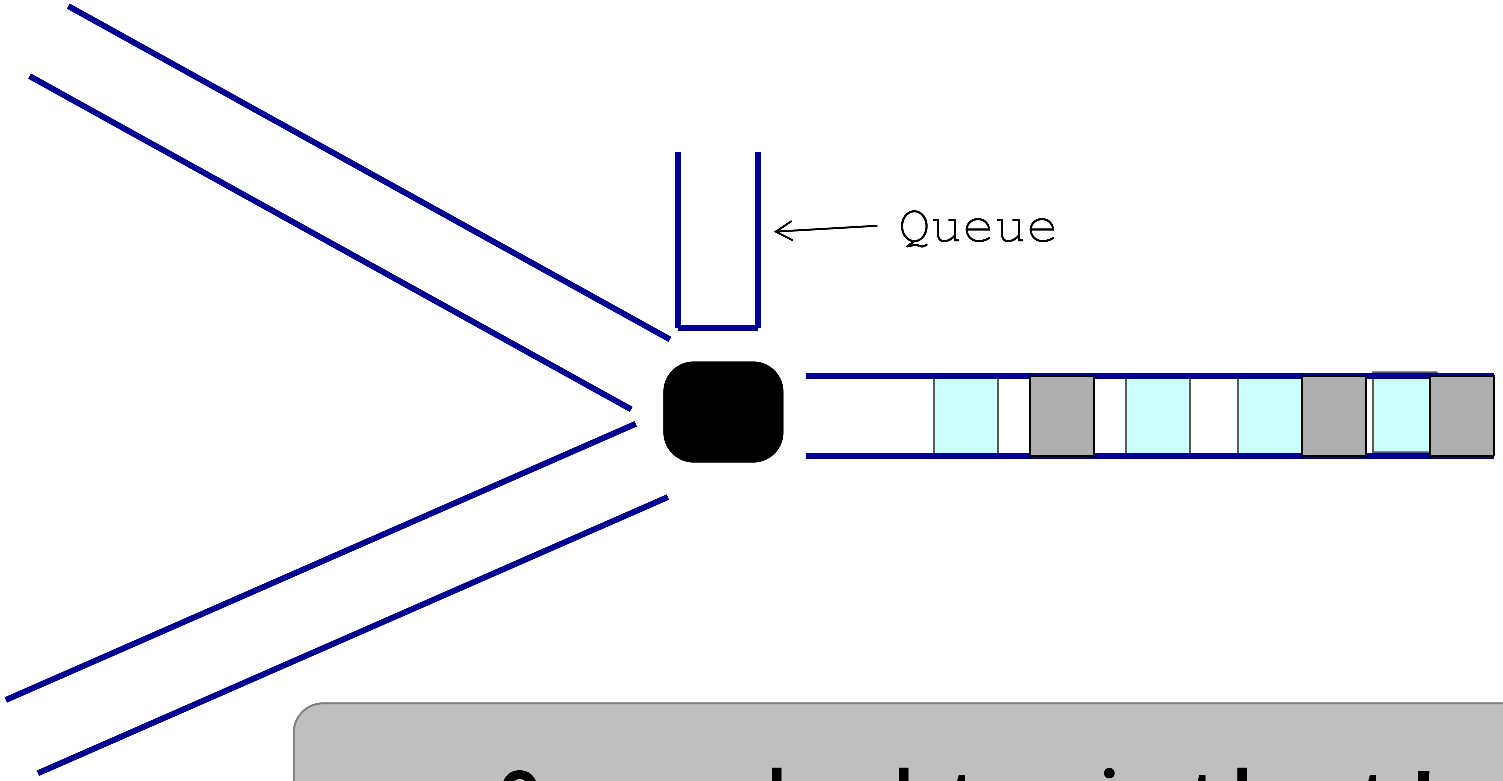
Not a rare event!


Queue

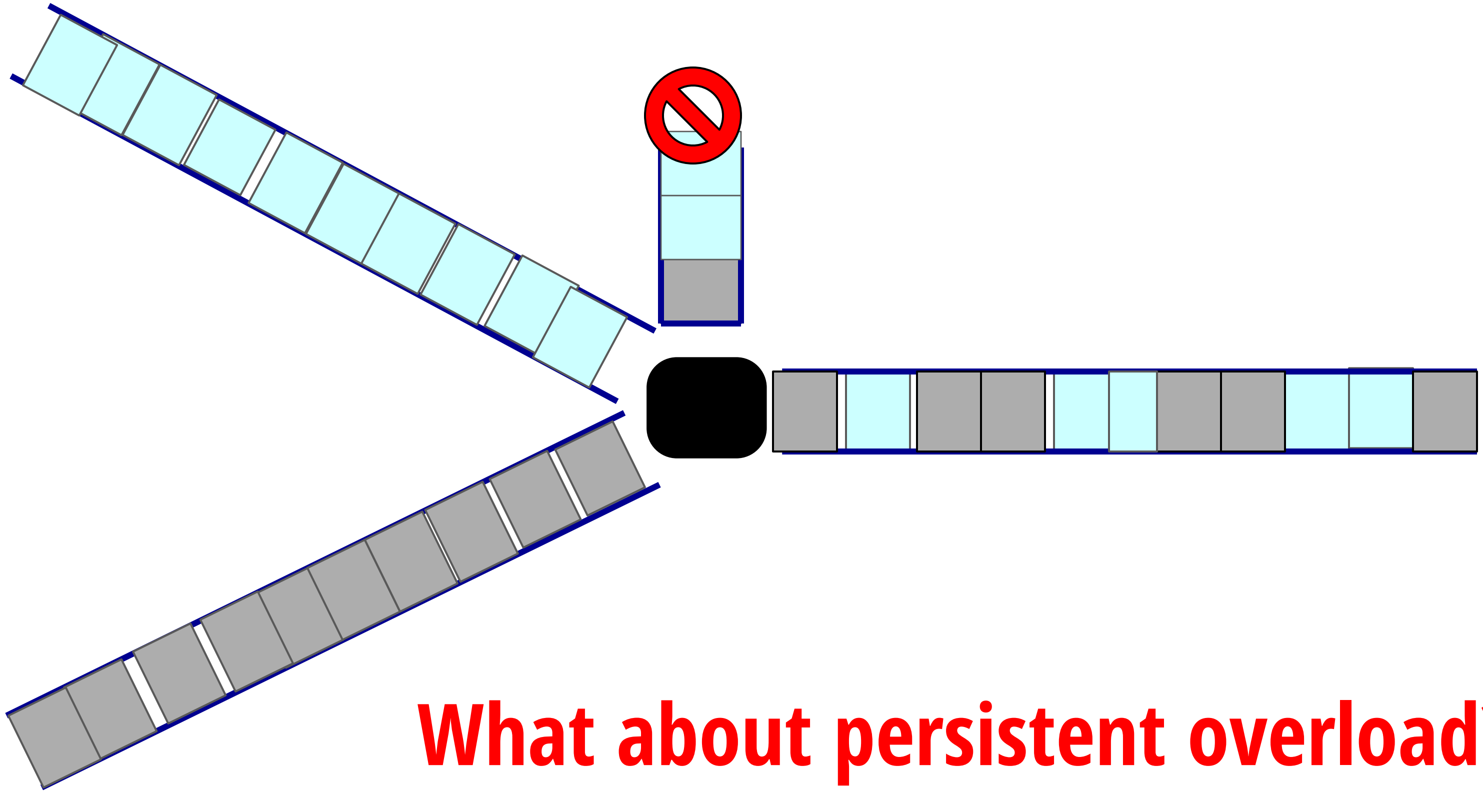

Queue


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
- What will queueing delays depend on?
  - Topic for future lectures …

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

See any new challenges that this introduces?

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….
  - Want no false positives, and high chance of success

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 important topics for the semester

- 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)**

What Else?

- Security? **(< 1 lecture)**
- How to download content efficiently **(1 lecture)**
  - HTTP and the web
- How do we manage our networks? **(1-2 lectures)**
  - SDN and more
- What else happens to packets on path? **(< 1 lecture)**
  - Middleboxes and NFV
- What about specialized networks? **(2-3 lectures)**
  - Wireless, Datacenters, Ethernets (L2)

Recap: key takeaways from our bottom-up overview

- What is a packet?

- Approaches to sharing the network – circuit vs. packet switching -- and their tradeoffs
  - *[just for fun: can you think of a way to make reservations practical?]*

- 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??

Break