

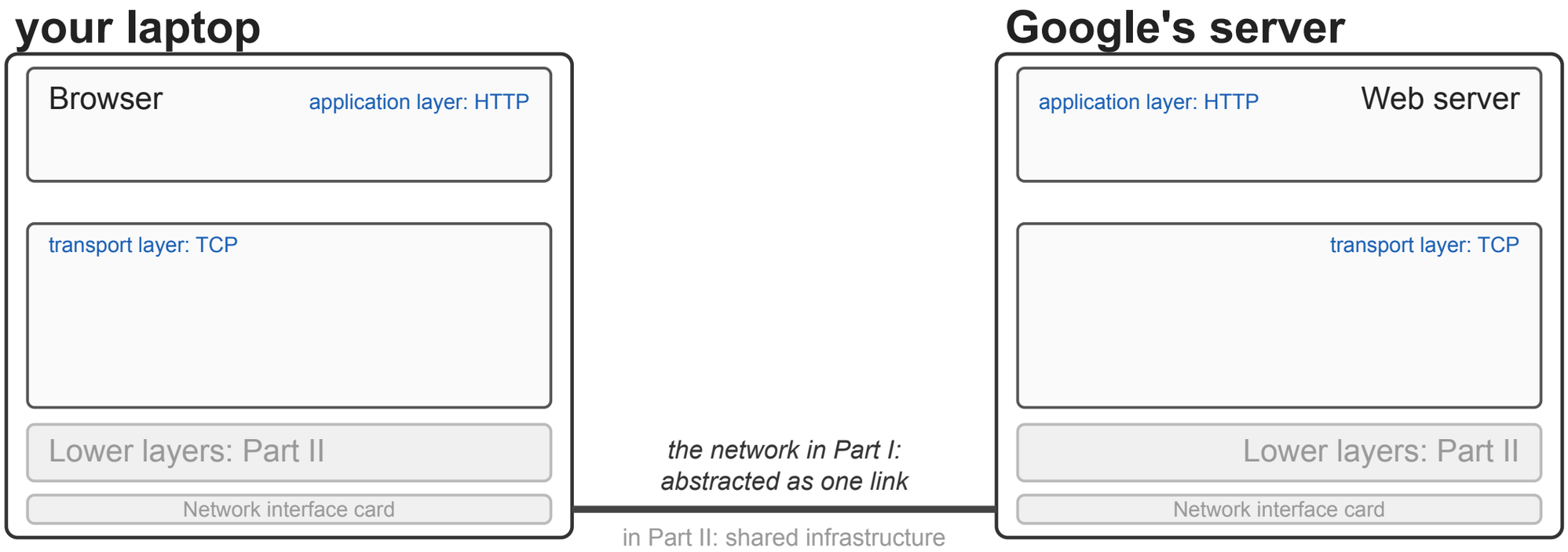
UNIVERSITY OF WATERLOO

How do many endpoints share a network?

CS 456/656, Fall 2026

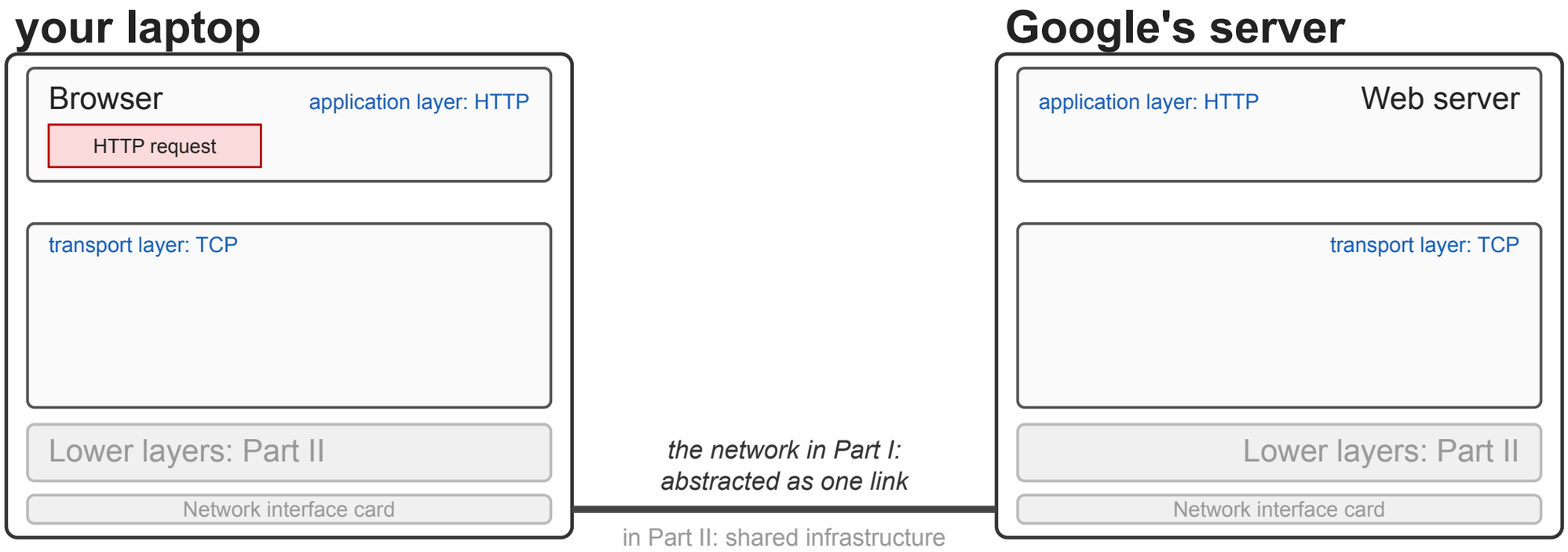
Mina Tahmasbi Arashloo

Part I, all together: typing www.google.com



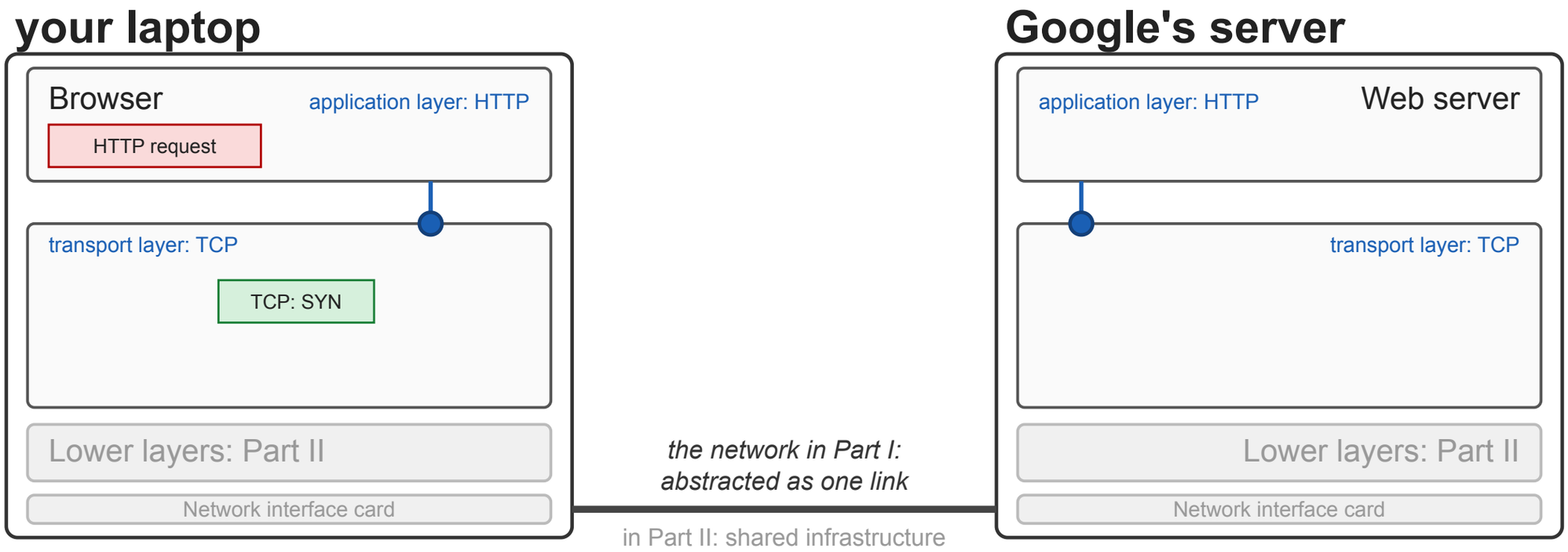
You come to school, open your laptop, and type www.google.com in the browser.

Part I, all together: typing www.google.com



The application, the browser, wants a page from the application on the other side, the web server.
It writes a request in the application protocol, HTTP: GET /index.html, Host: www.google.com.

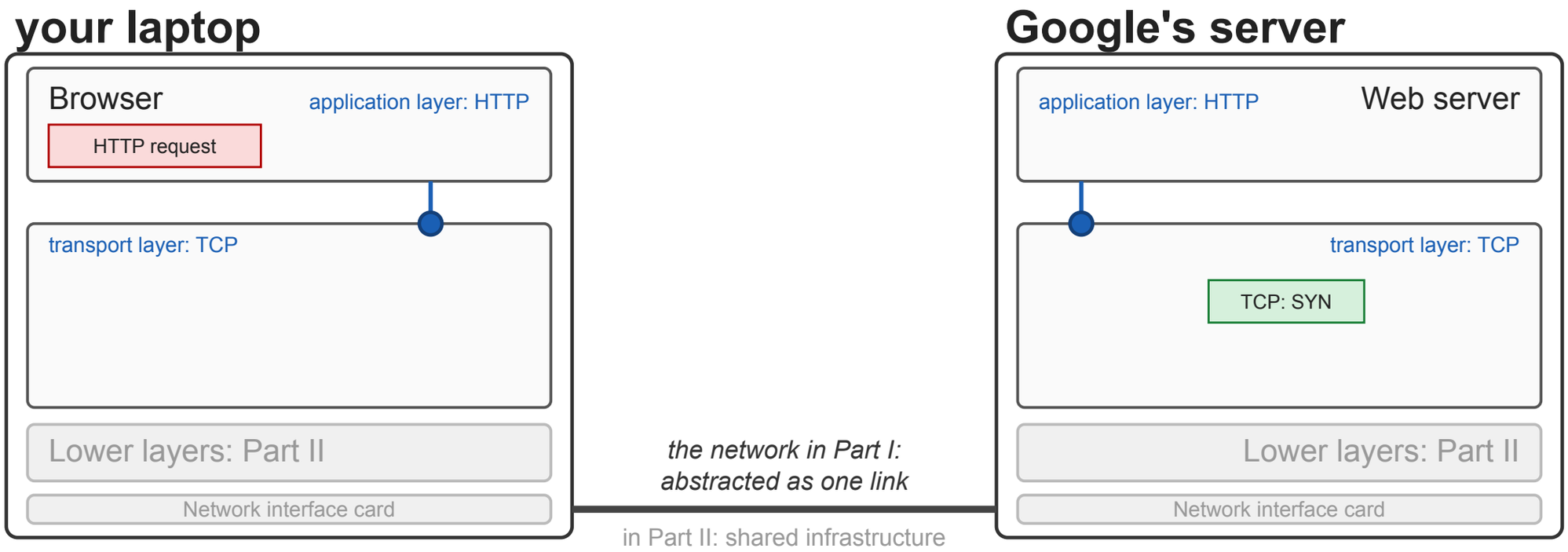
Part I, all together: typing www.google.com



It asks the transport layer for a channel to the server's application: socket(), connect().

The transport protocol, TCP, starts the handshake: a SYN segment with its first sequence number.

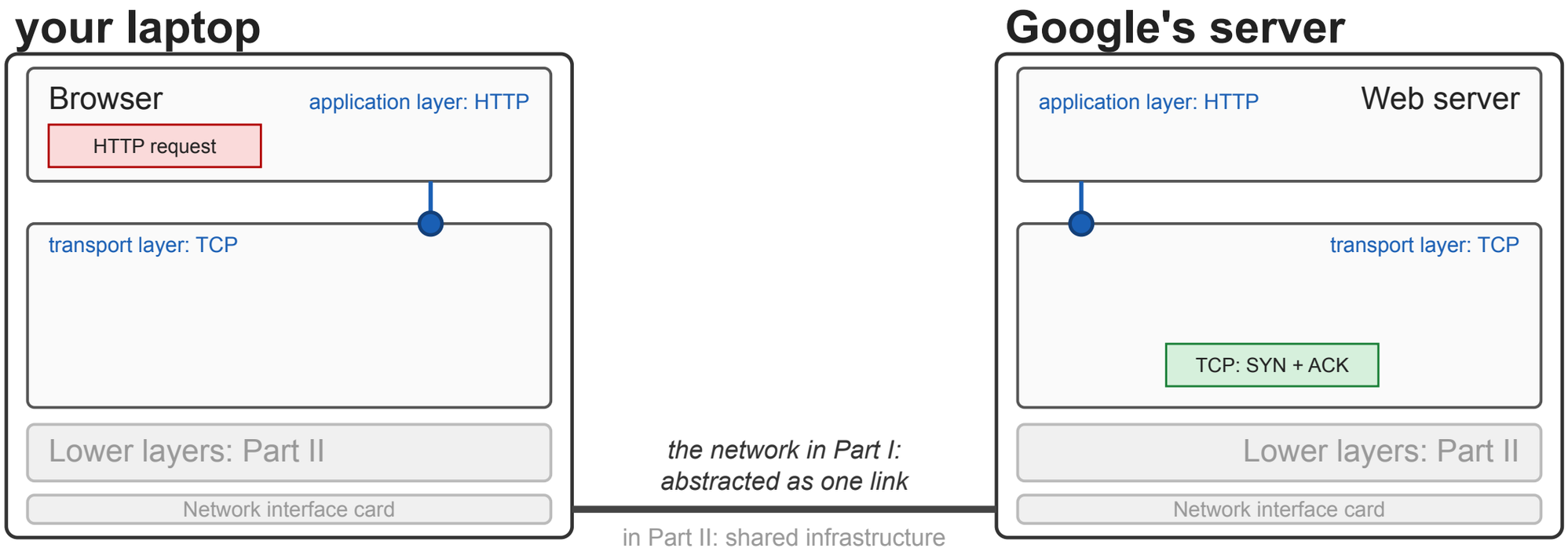
Part I, all together: typing www.google.com



It asks the transport layer for a channel to the server's application: socket(), connect().

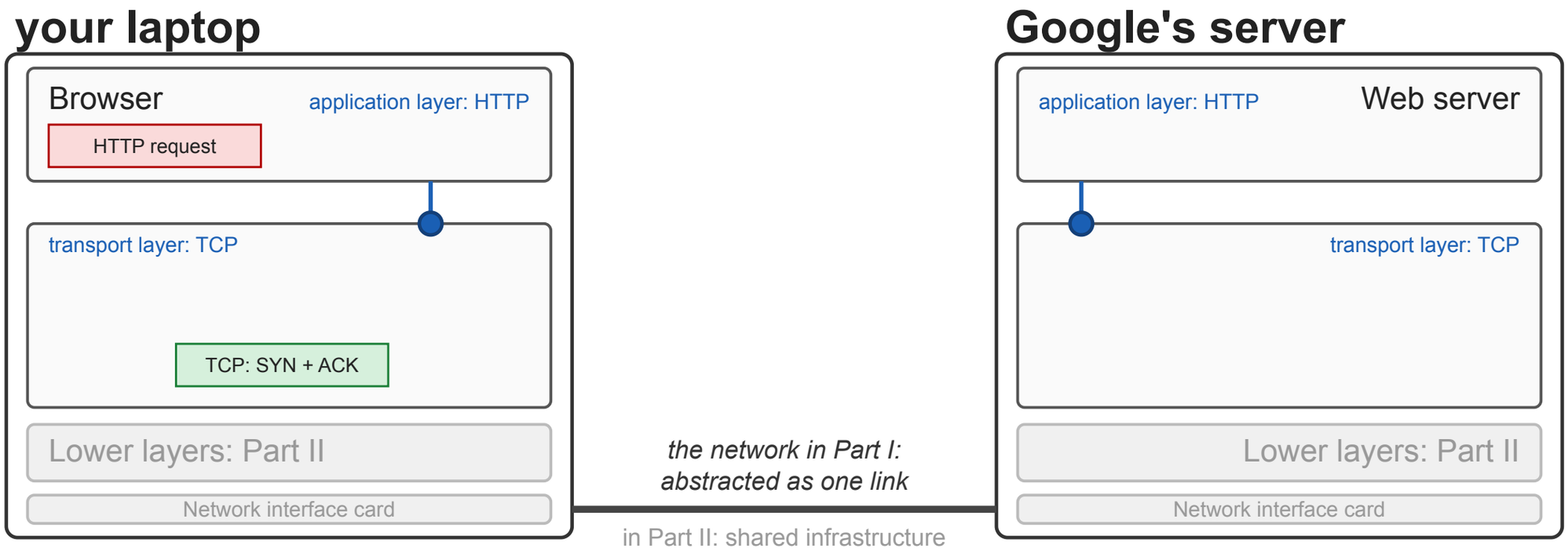
The transport protocol, TCP, starts the handshake: a SYN segment with its first sequence number.

Part I, all together: typing www.google.com



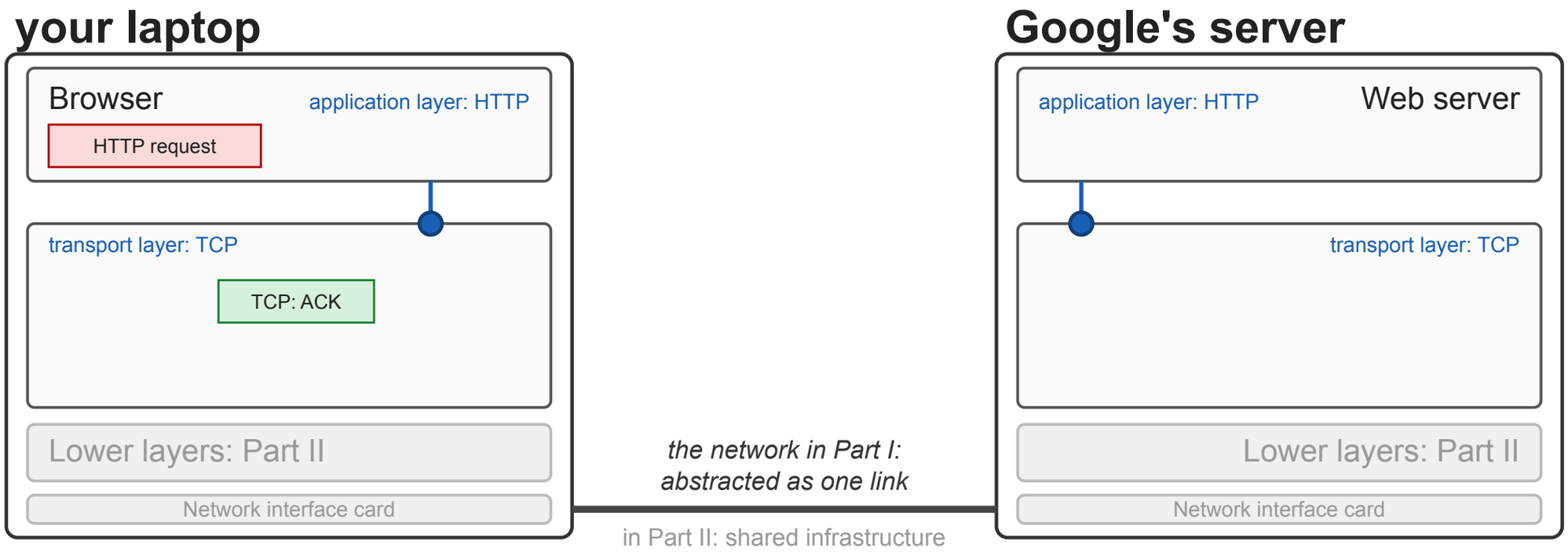
The server's transport layer, listening on port 80 (socket(), bind(), listen(), accept()), answers: SYN + ACK, with its own first sequence number, window and MSS.

Part I, all together: typing www.google.com



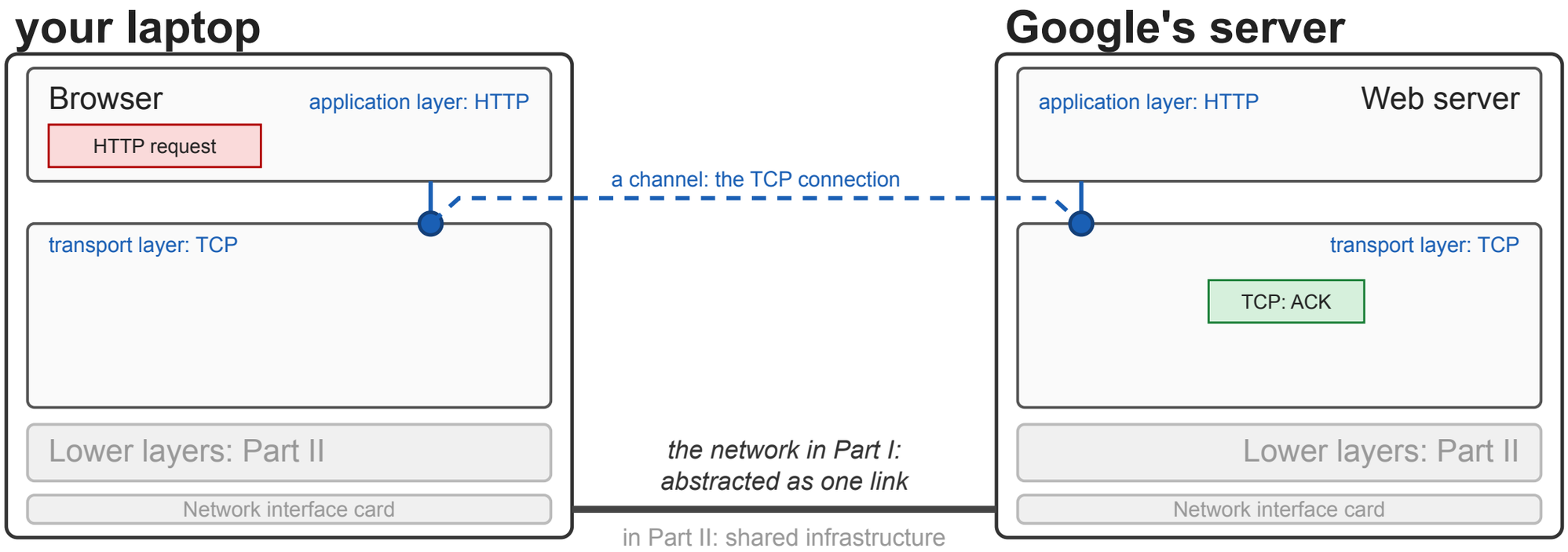
The server's transport layer, listening on port 80 (socket(), bind(), listen(), accept()), answers: SYN + ACK, with its own first sequence number, window and MSS.

Part I, all together: typing www.google.com



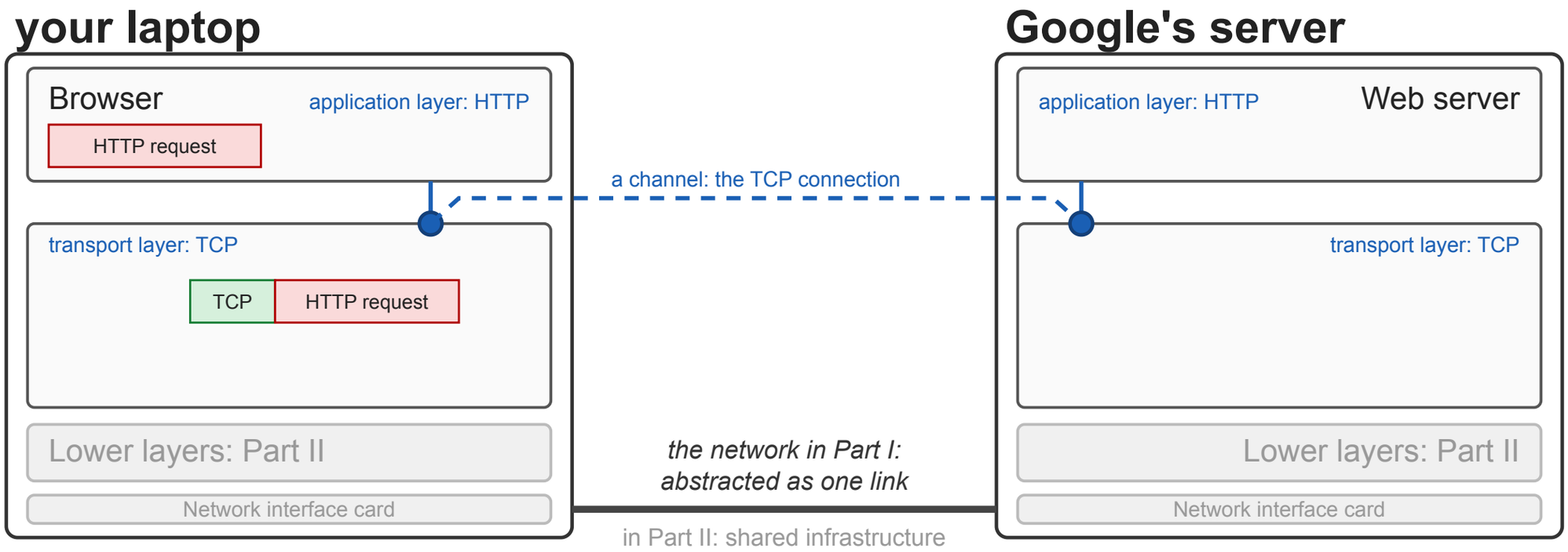
The laptop's transport layer answers with an ACK.

Part I, all together: typing www.google.com



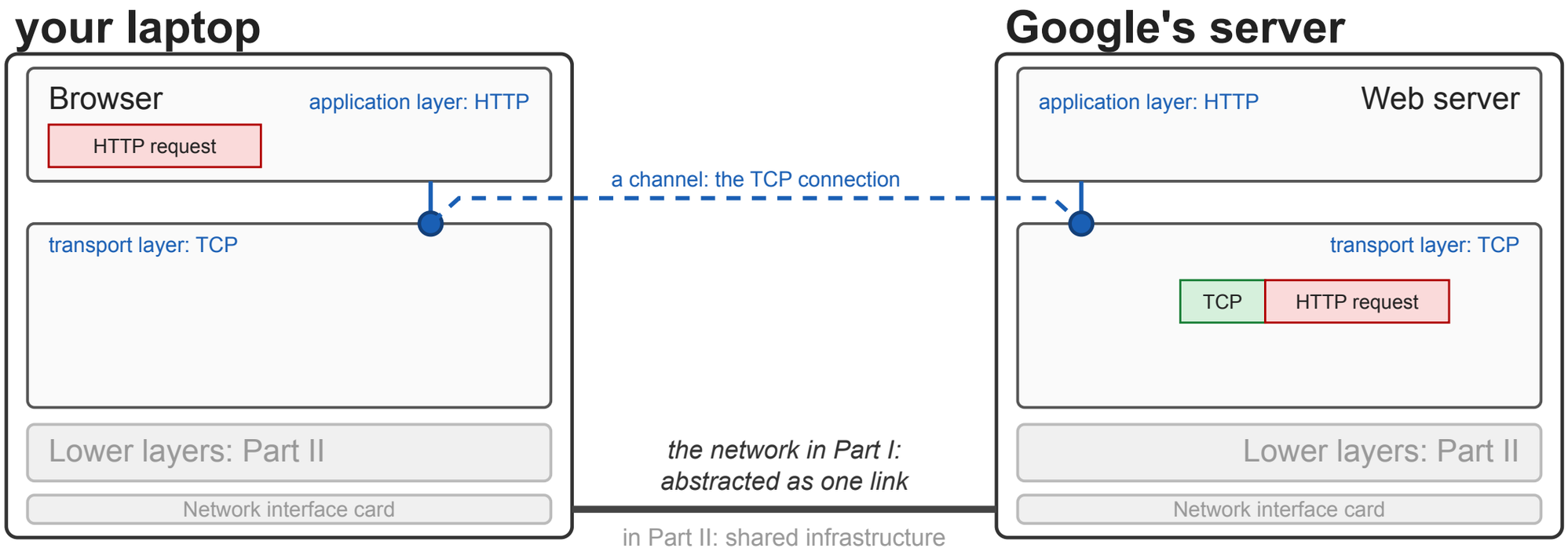
The handshake is complete: the connection (channel) is established and can be used for data transfer.

Part I, all together: typing www.google.com



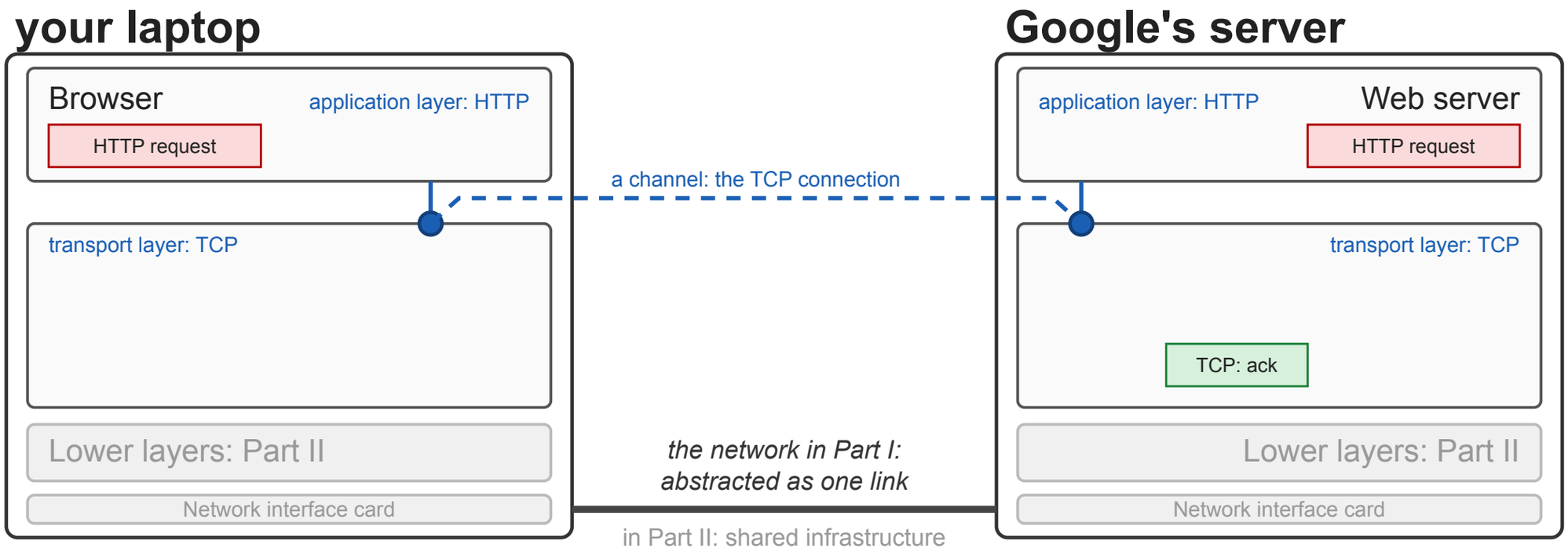
The browser writes its request into the channel: send(). TCP segments the request, adding its own metadata, and ensures it is transferred reliably to the server (the request fits in one segment in this example).

Part I, all together: typing www.google.com



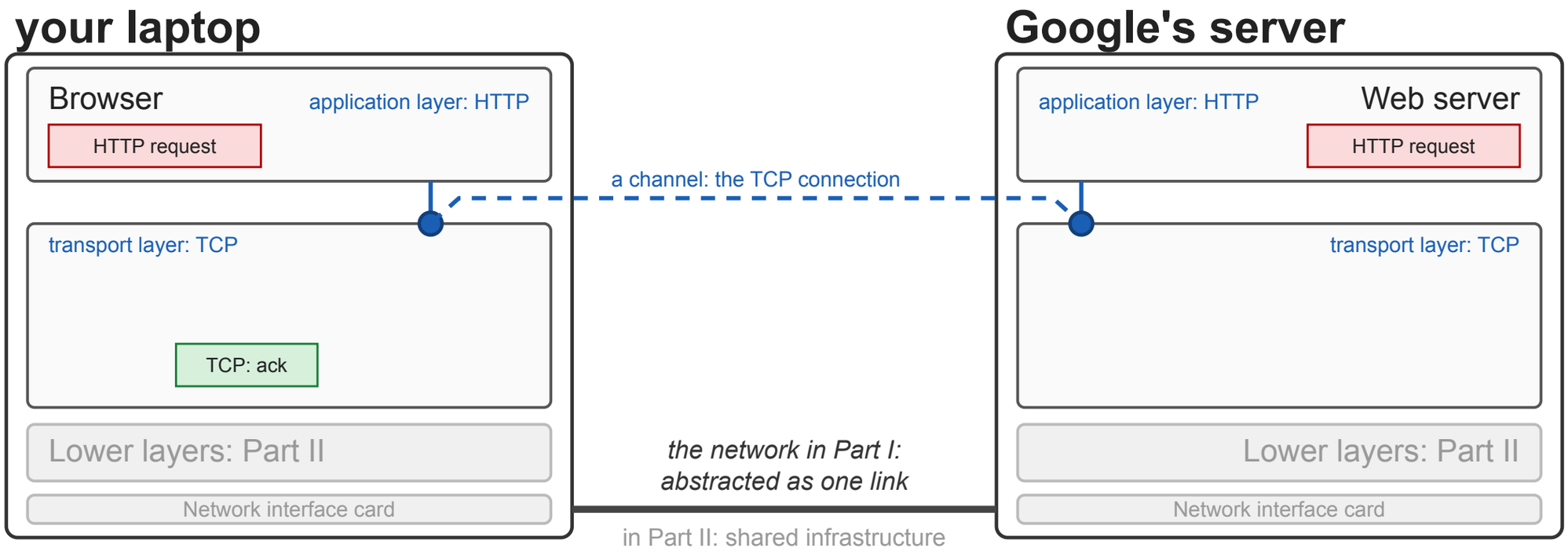
The browser writes its request into the channel: send(). TCP segments the request, adding its own metadata, and ensures it is transferred reliably to the server (the request fits in one segment in this example).

Part I, all together: typing www.google.com



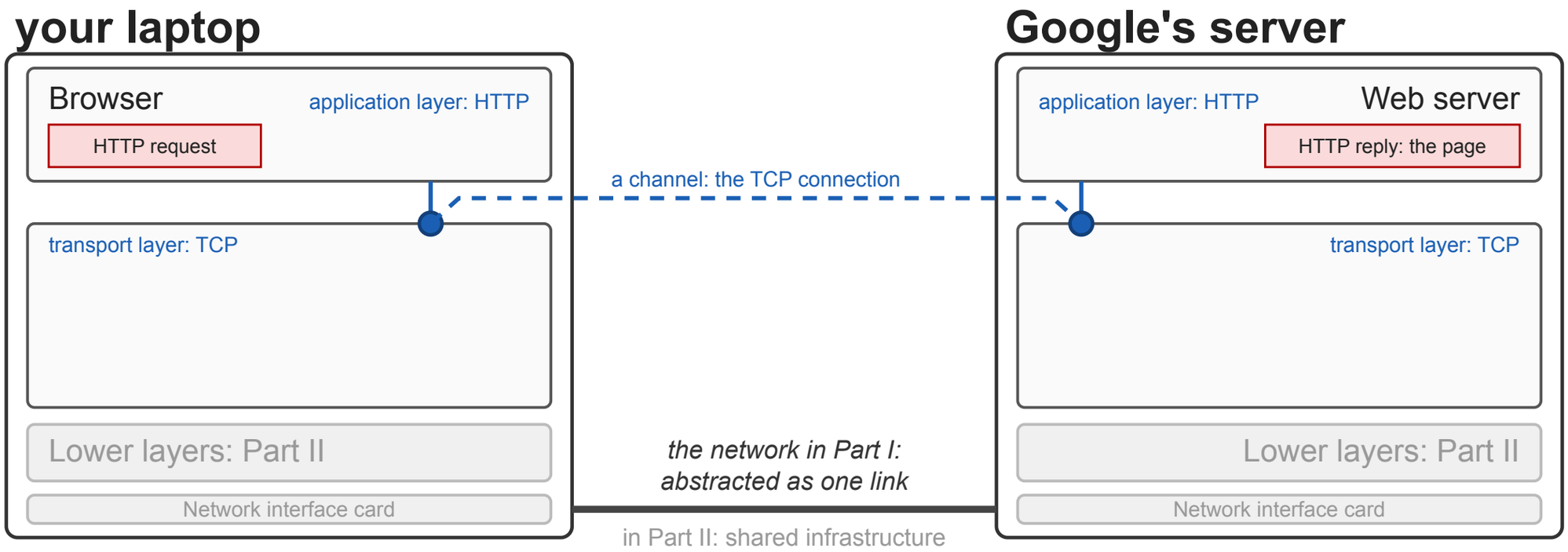
The server's transport layer checks the segment, hands the bytes up (recv() returns, and the web server reads the request), and acknowledges it.

Part I, all together: typing www.google.com



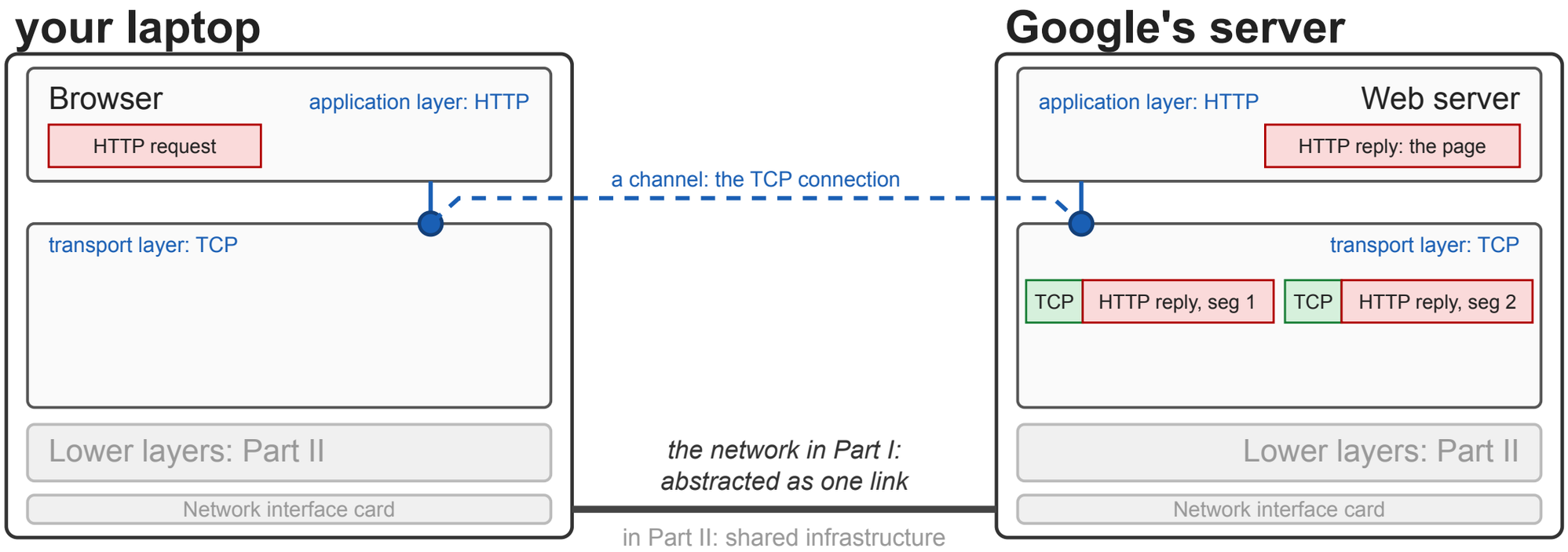
The server's transport layer checks the segment, hands the bytes up (recv() returns, and the web server reads the request), and acknowledges it.

Part I, all together: typing www.google.com



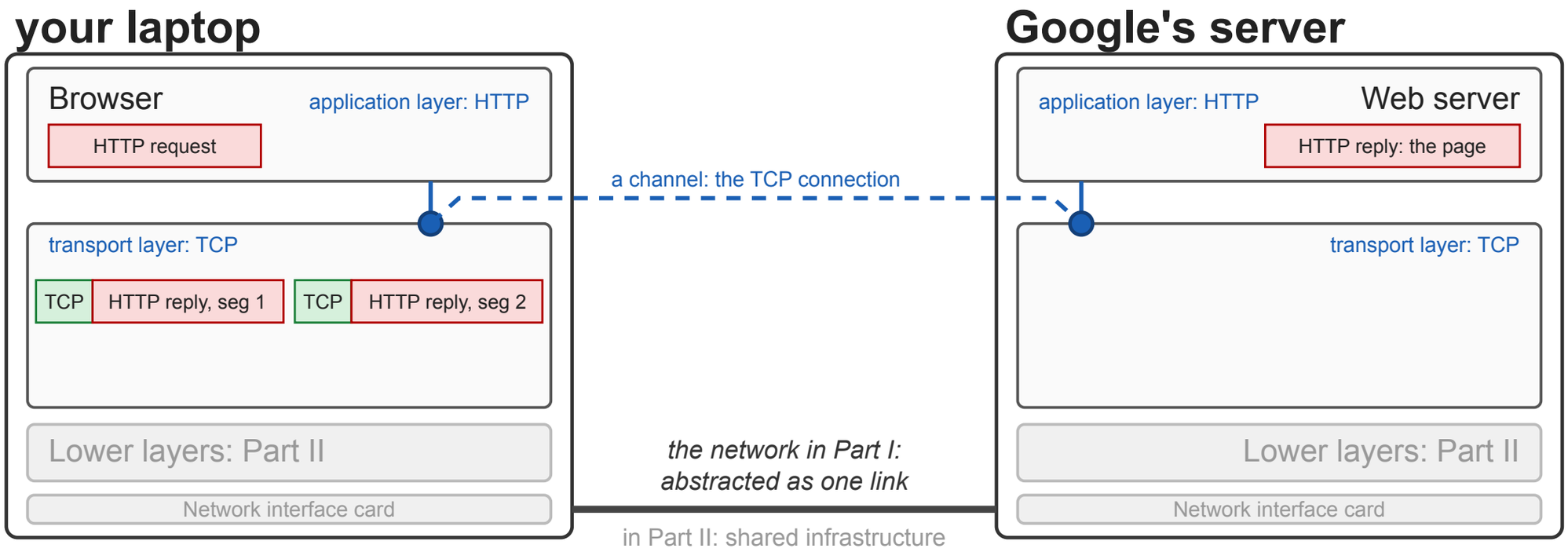
The web server writes the reply in HTTP, 200 OK and the page, and writes it into the channel: send().

Part I, all together: typing www.google.com



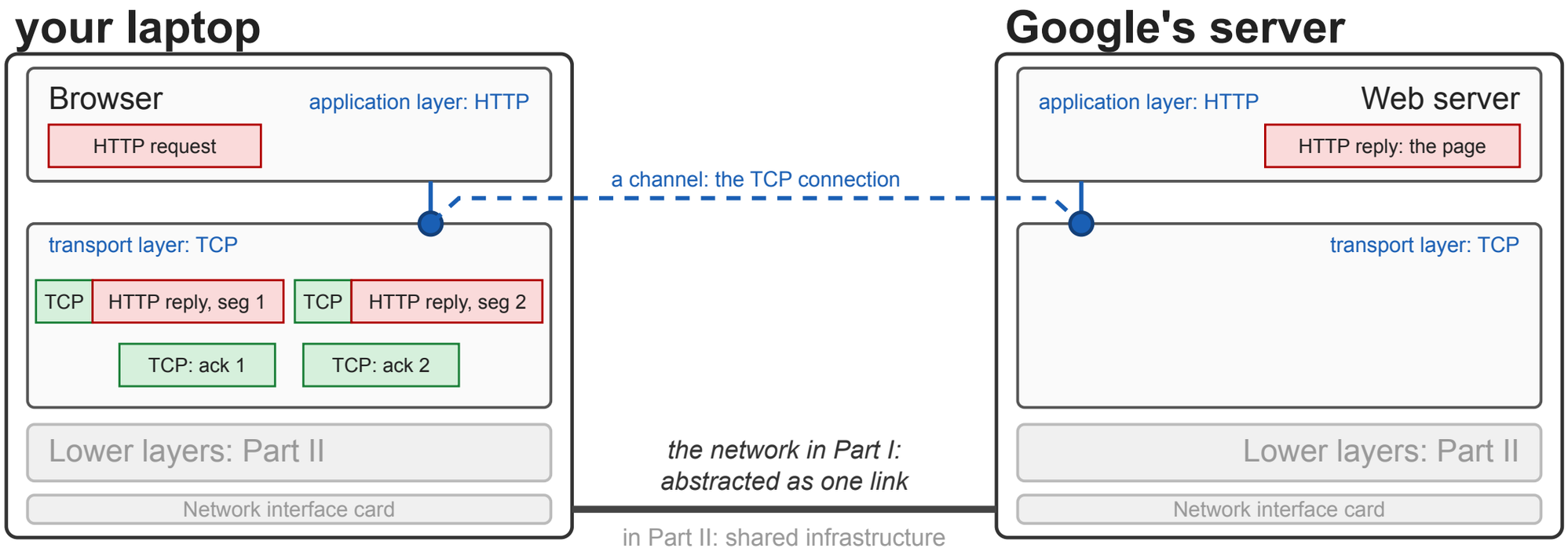
Its transport layer cuts the bytes into segments of at most one MSS and sends them out.

Part I, all together: typing www.google.com



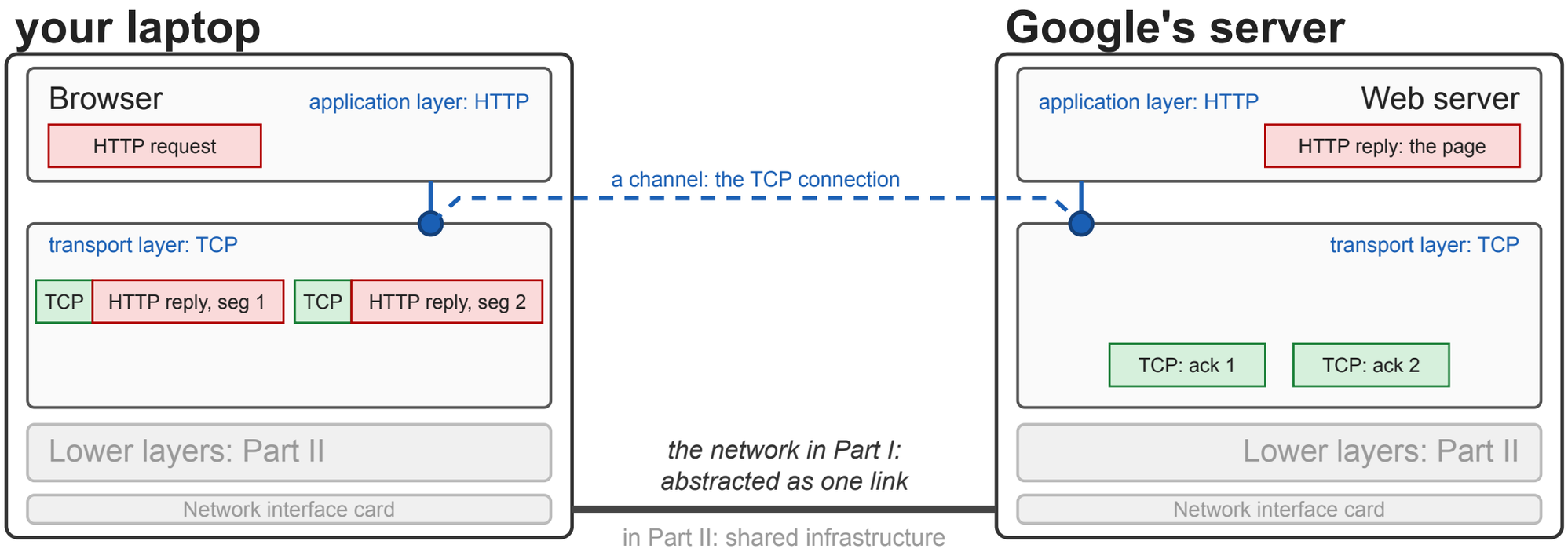
Its transport layer cuts the bytes into segments of at most one MSS and sends them out.

Part I, all together: typing www.google.com



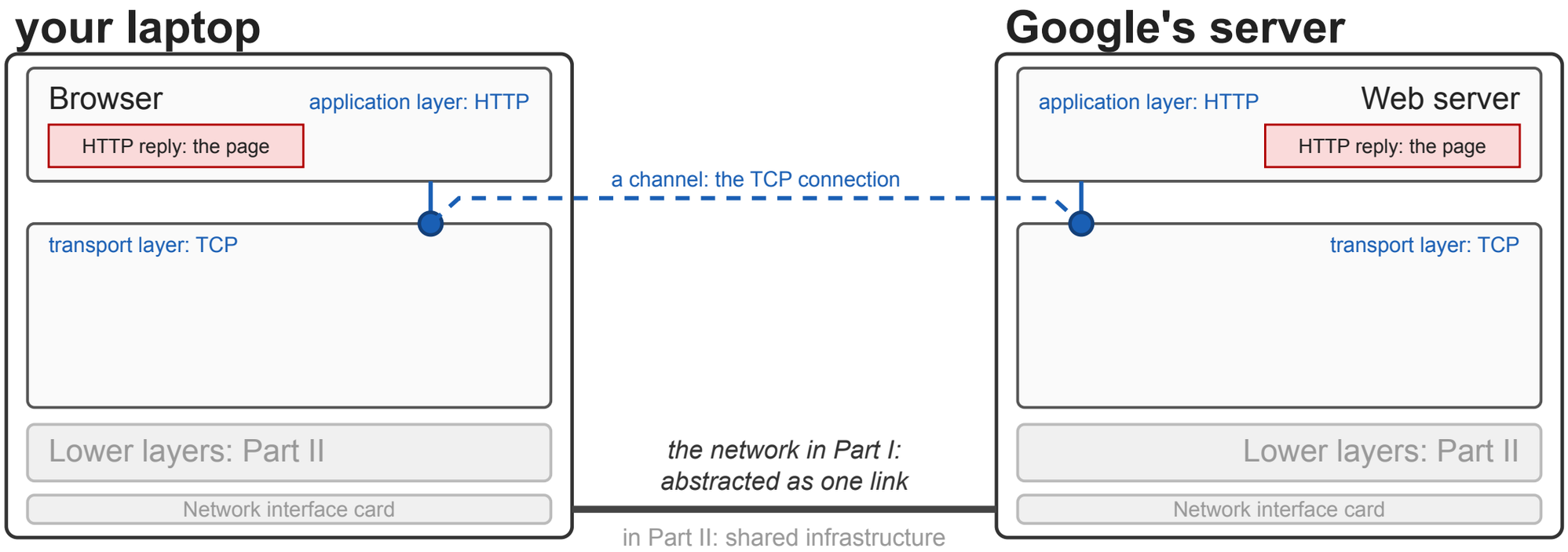
The laptop's transport layer (TCP) checks each segment, puts them in order, and acknowledges each one, also with its receive window. A lost segment is sent again when the timer goes off, or after three duplicate acks.

Part I, all together: typing www.google.com



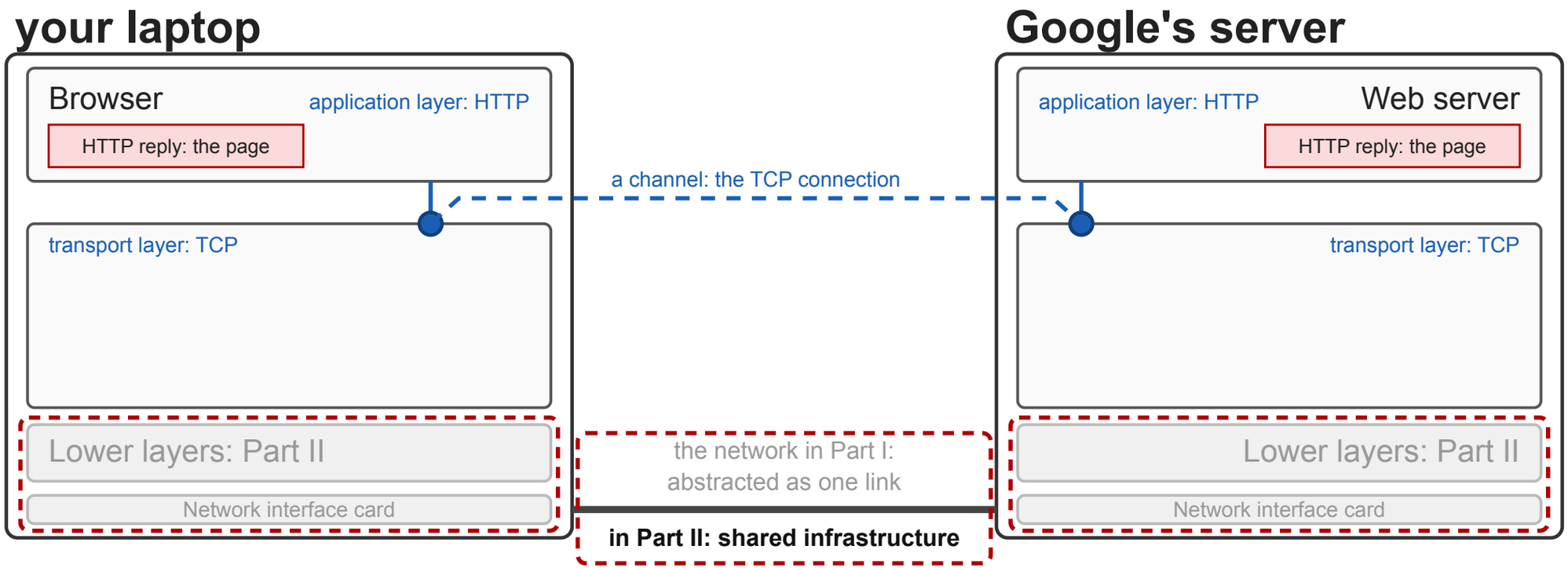
With the acks, the server's transport layer slides its window; here, there is nothing more to send.

Part I, all together: typing www.google.com



recv() returns the bytes in order: the browser renders the page. When both sides are done, each closes its direction of the channel.

Part I, all together: typing www.google.com

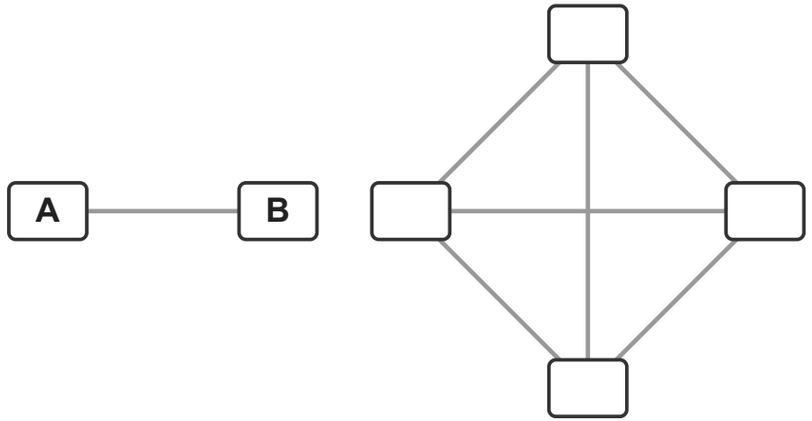


What if there are more than two machines?



2 machines
1 link

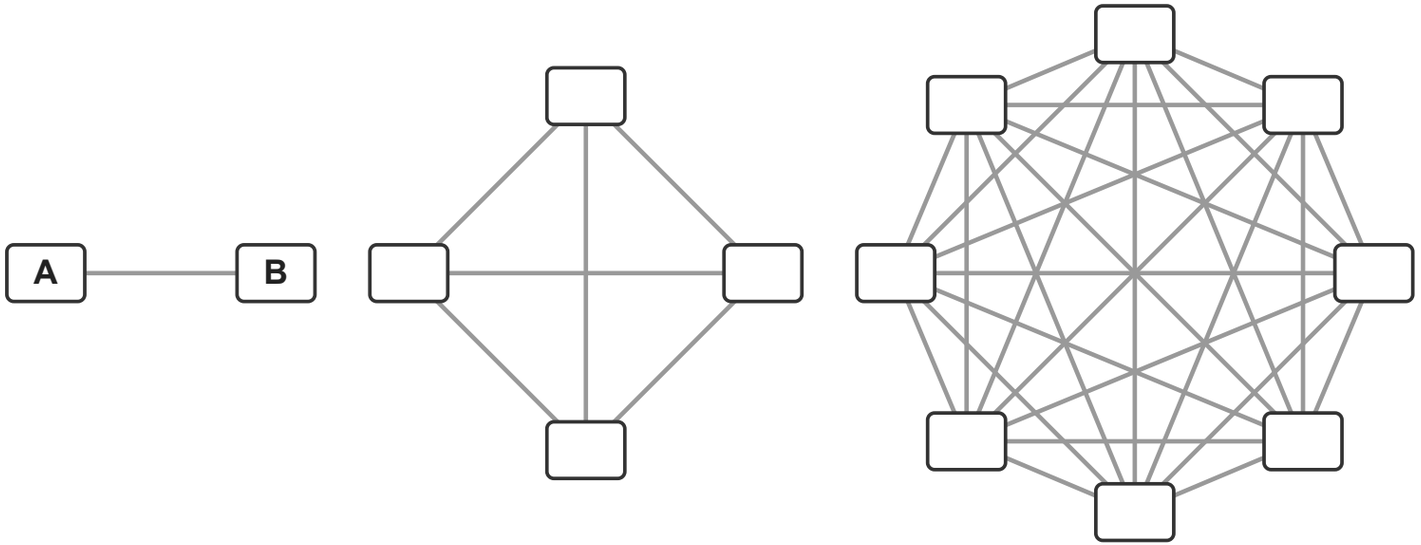
What if there are more than two machines?



2 machines
1 link

4 machines
6 links

What if there are more than two machines?

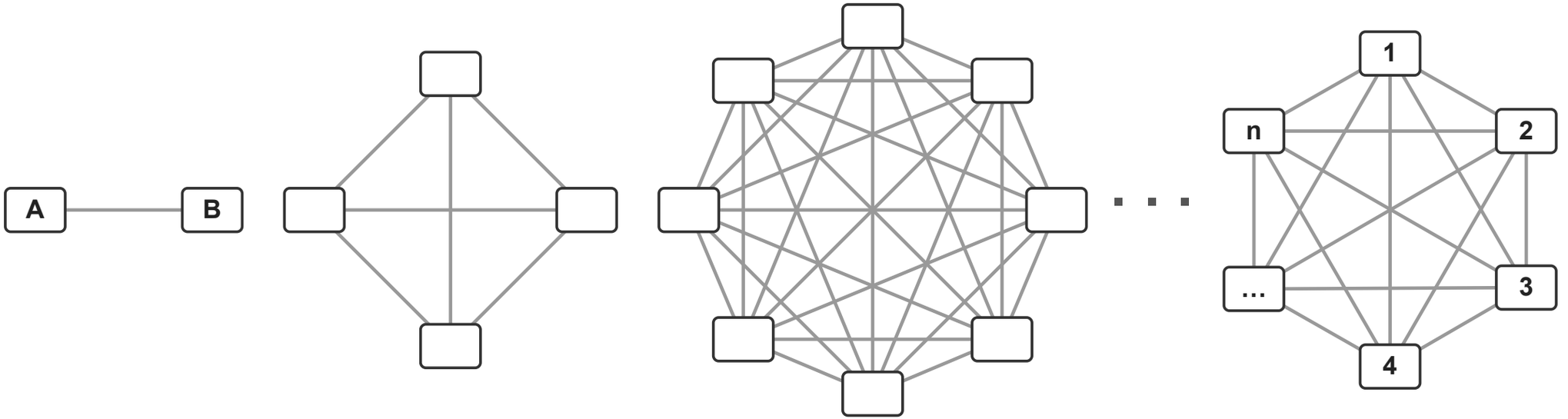


2 machines
1 link

4 machines
6 links

8 machines
28 links

What if there are more than two machines?



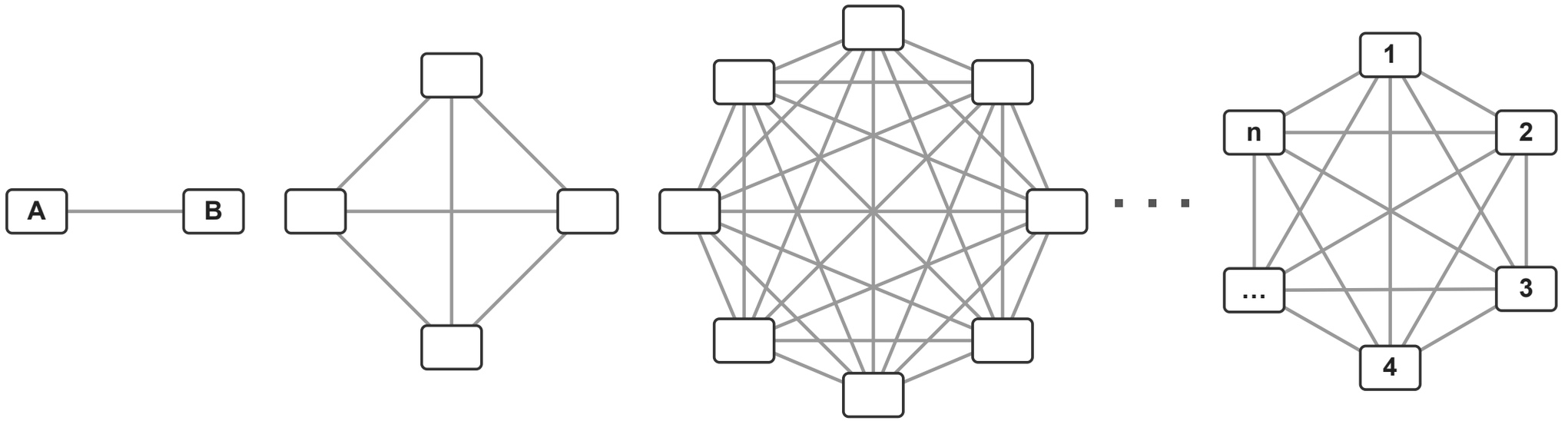
2 machines
1 link

4 machines
6 links

8 machines
28 links

n machines
n (n − 1) / 2 links
n − 1 cards per machine

What if there are more than two machines?



2 machines
1 link

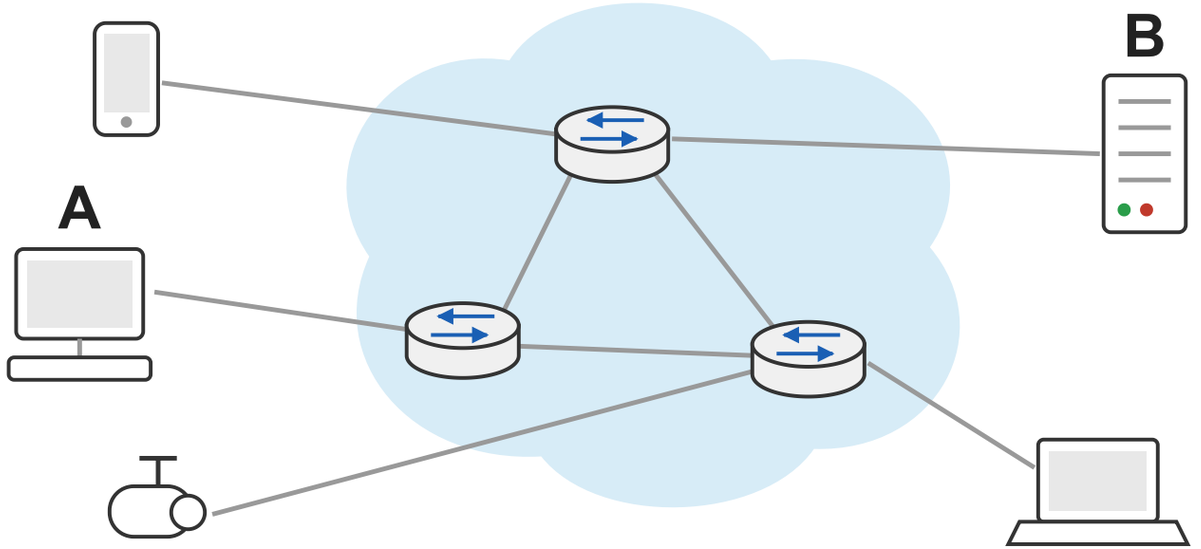
4 machines
6 links

8 machines
28 links

n machines
n (n − 1) / 2 links
n − 1 cards per machine

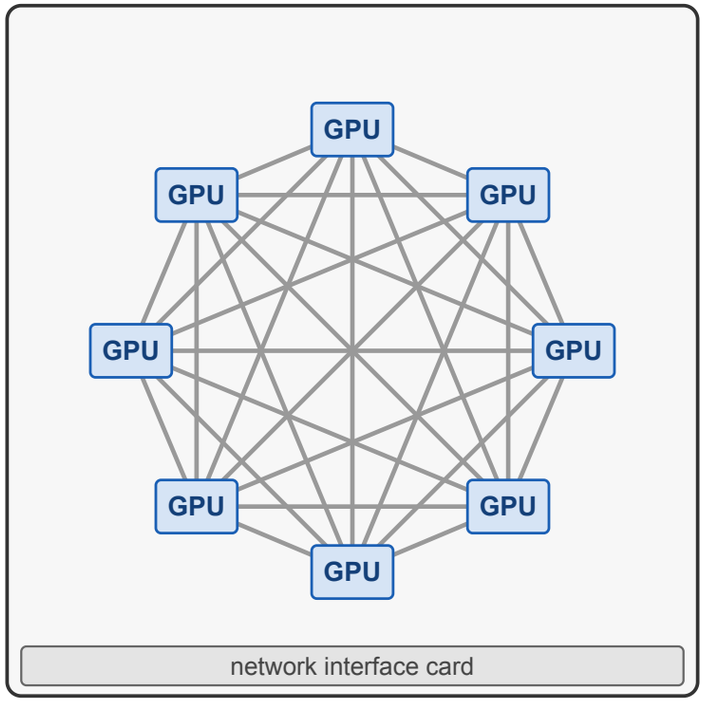
> Would you design your network that way? Why or why not?

A network of devices in between



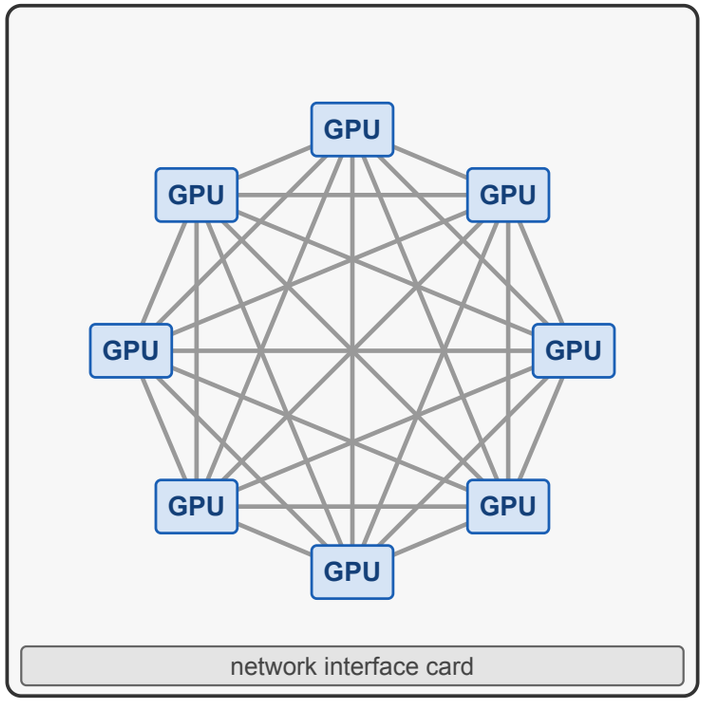
- Each machine has one link, to a device near it.
- The devices have links to each other.
- A packet from A to B crosses several links and devices on the way, which are shared by packets from other data transfers.

Aside: sometimes a mesh is the right answer

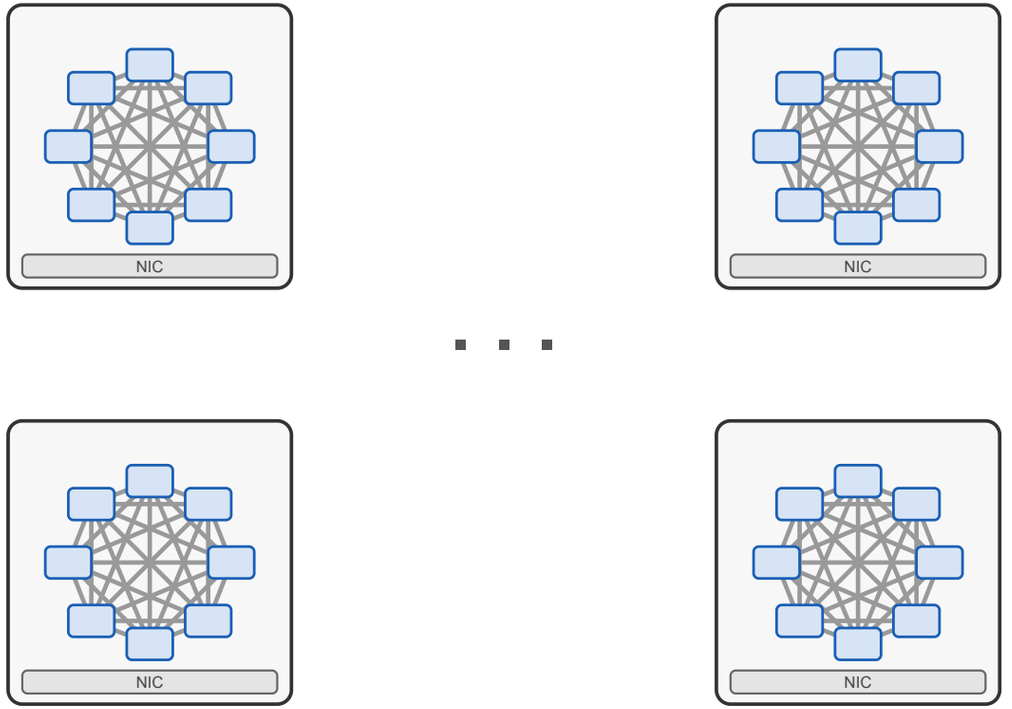


one server, 8 GPUs

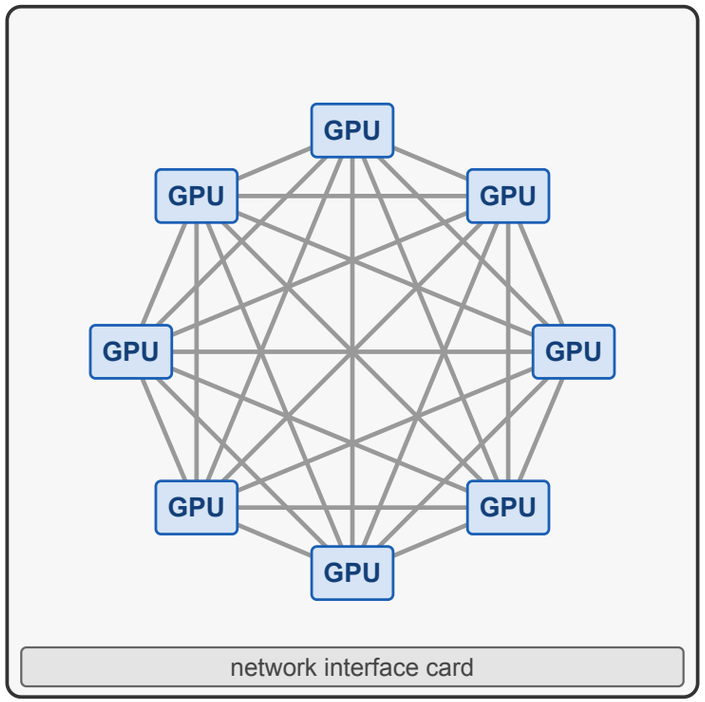
Aside: sometimes a mesh is the right answer



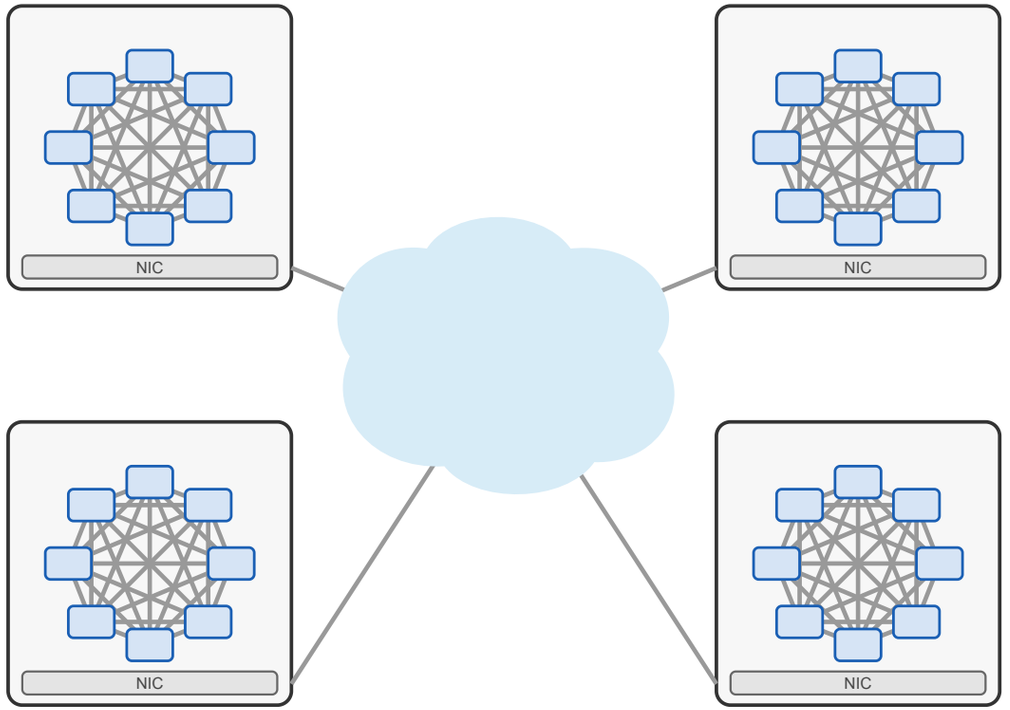
one server, 8 GPUs



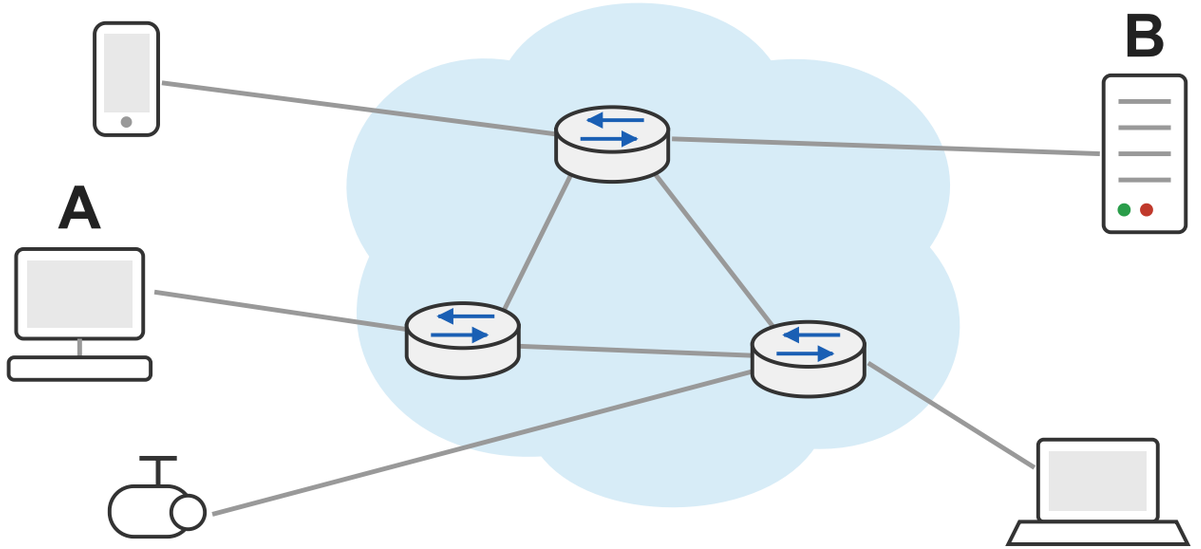
Aside: sometimes a mesh is the right answer



one server, 8 GPUs

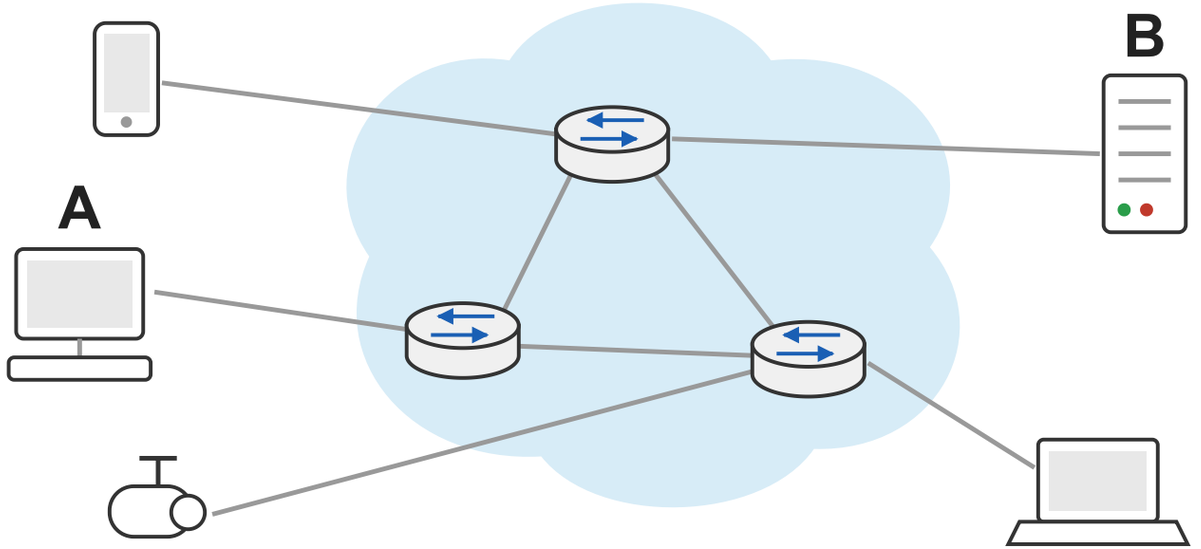


A network of devices in between



- Each machine has one link, to a device near it.
- The devices have links to each other.
- A packet from A to B crosses several links and devices on the way, which are shared by packets from other data transfers.

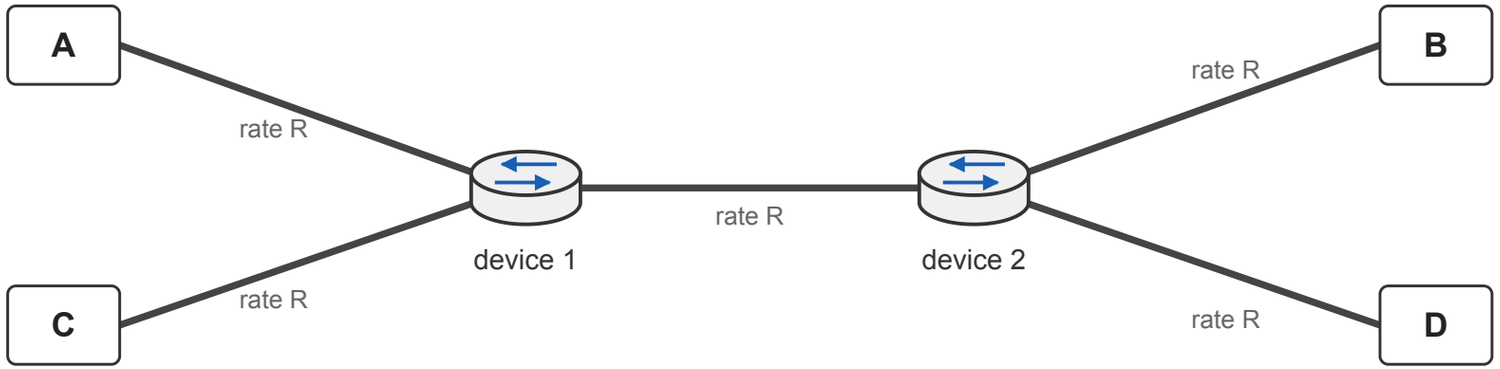
A network of devices in between



- Each machine has one link, to a device near it.
- The devices have links to each other.
- A packet from A to B crosses several links and devices on the way, which are shared by packets from other data transfers.

How do concurrent data transfers share network devices and links?

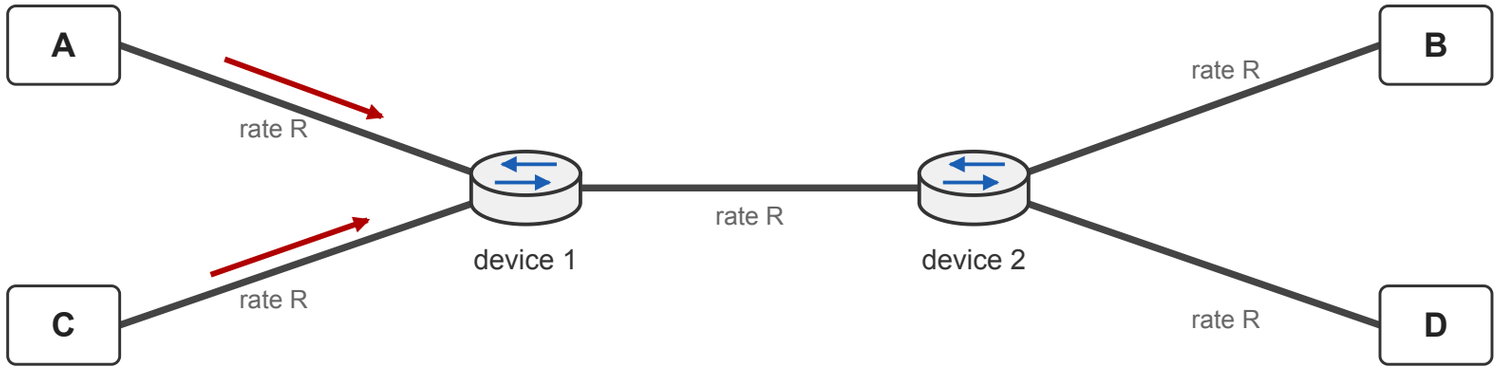
A wants to send a large file to B



C is a web server answering client requests

How do concurrent data transfers share network devices and links?

A wants to send a large file to B

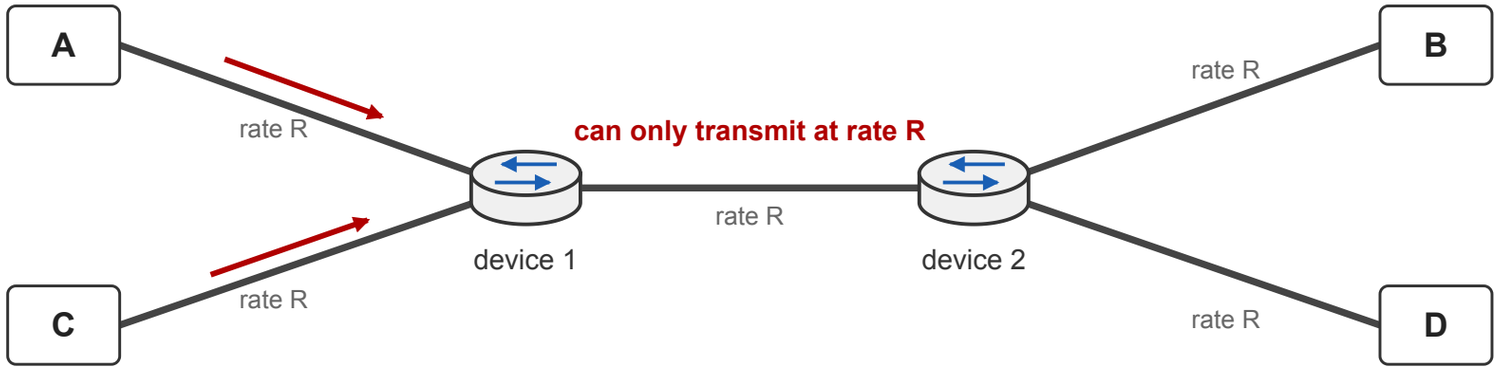


C is a web server answering client requests

They can both send at rate R

How do concurrent data transfers share network devices and links?

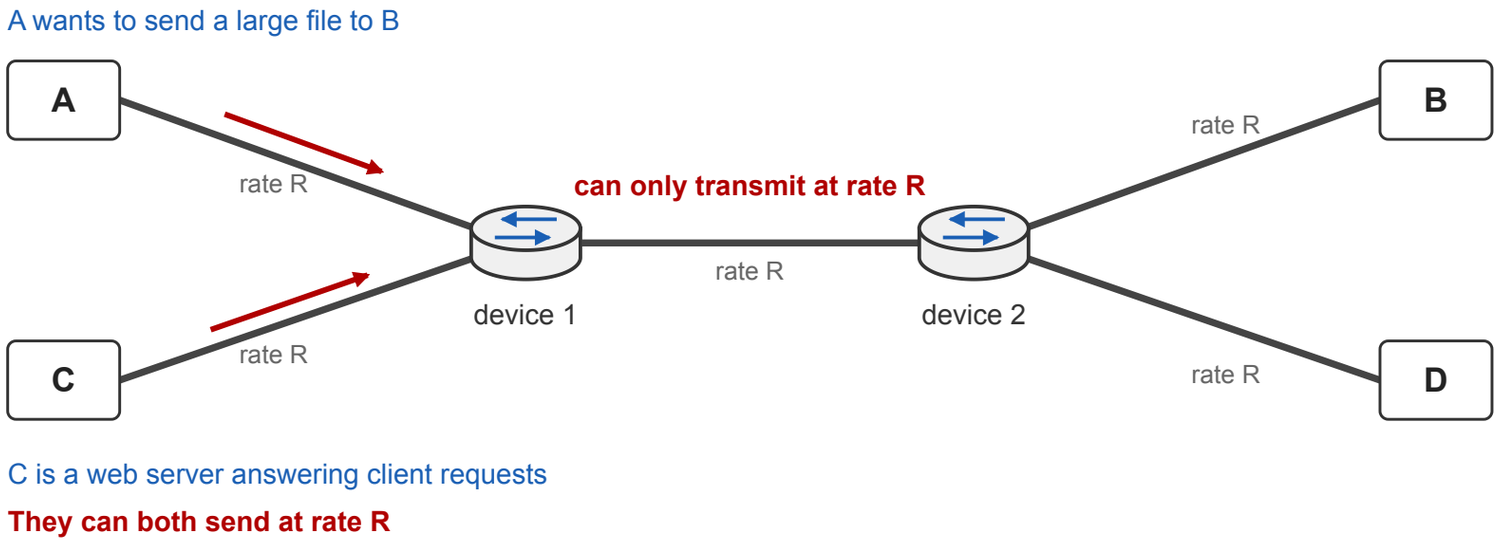
A wants to send a large file to B



C is a web server answering client requests

They can both send at rate R

How do concurrent data transfers share network devices and links?



> How would you design your network to share device 1's link to device 2 between multiple data transfers?

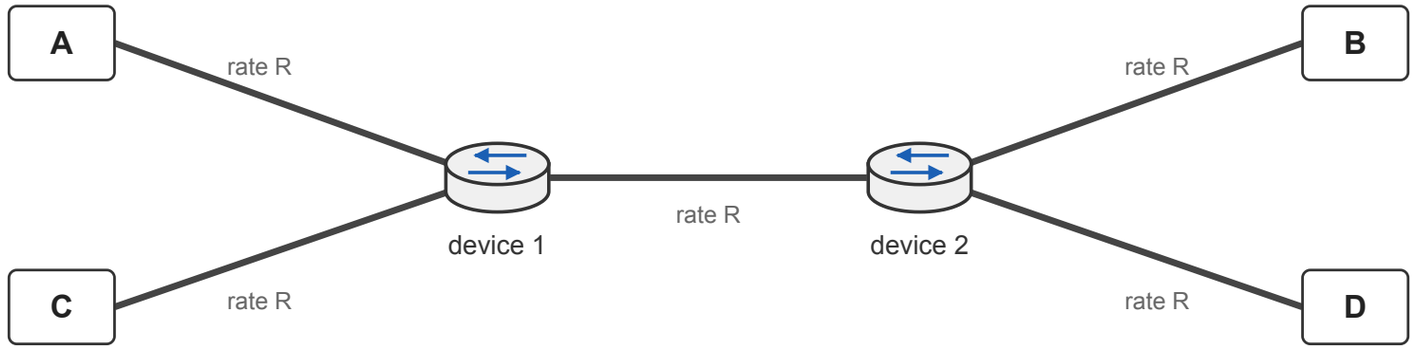
Two families of answers

Reserve before sending

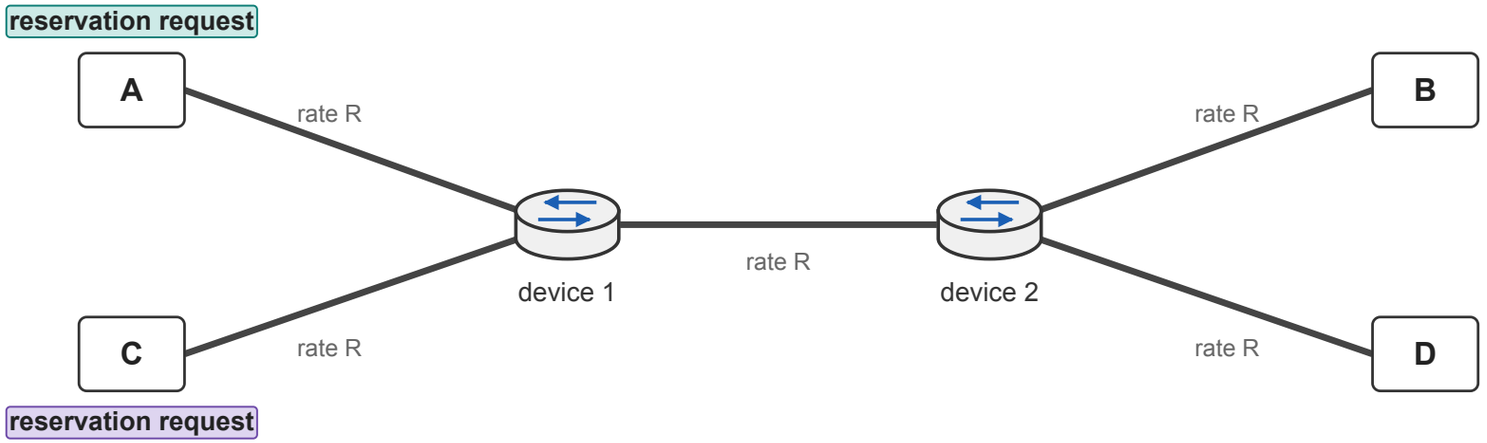
Before A sends anything, it asks every device on the path to set aside a share of its link for A, e.g., half of R.

A only sends after its reservation is confirmed.

Reserve before sending

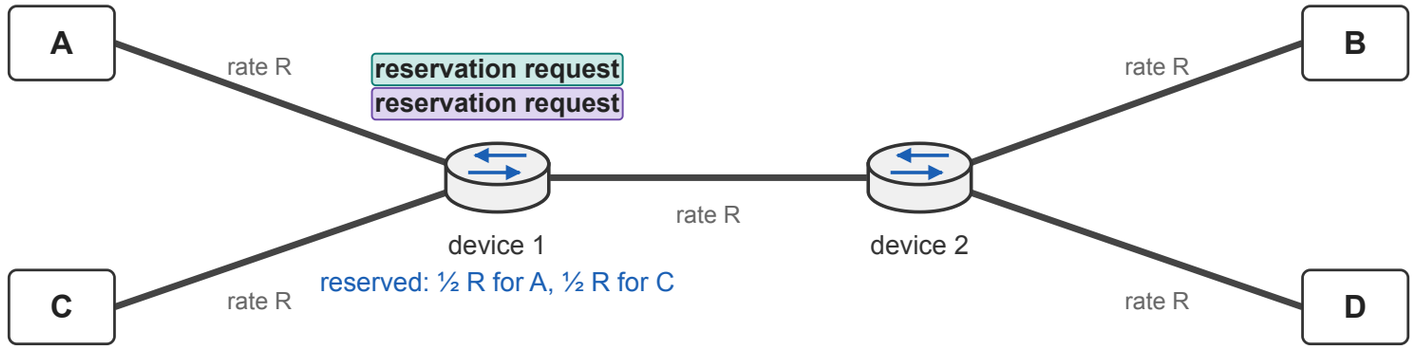


Reserve before sending



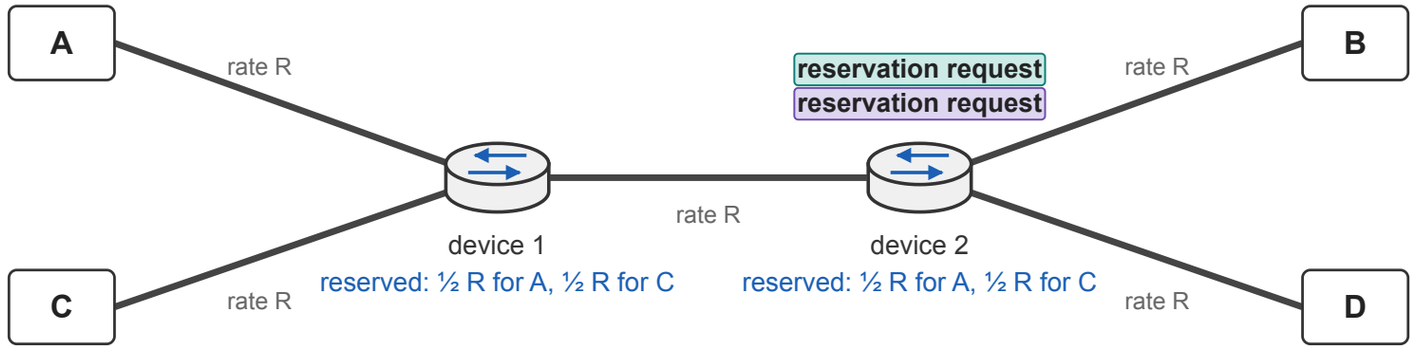
A sends a reservation request for a path to B. C sends one for a path to D

Reserve before sending



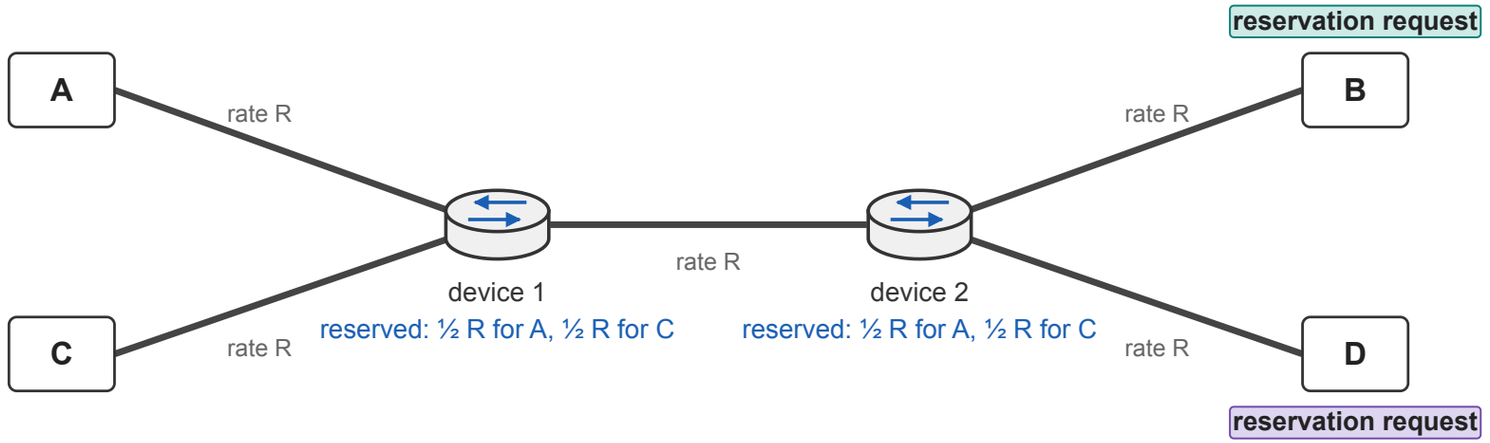
Device 1 sets aside half of its link to device 2 for A, and the other half for C

Reserve before sending



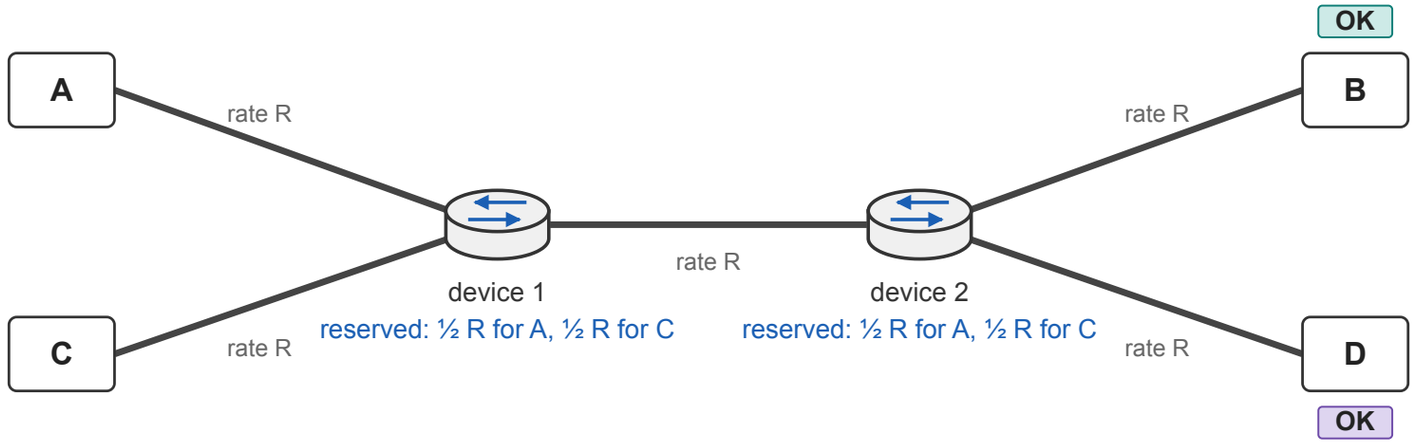
Device 2 does the same for its links, and remembers it, as device 1 does

Reserve before sending



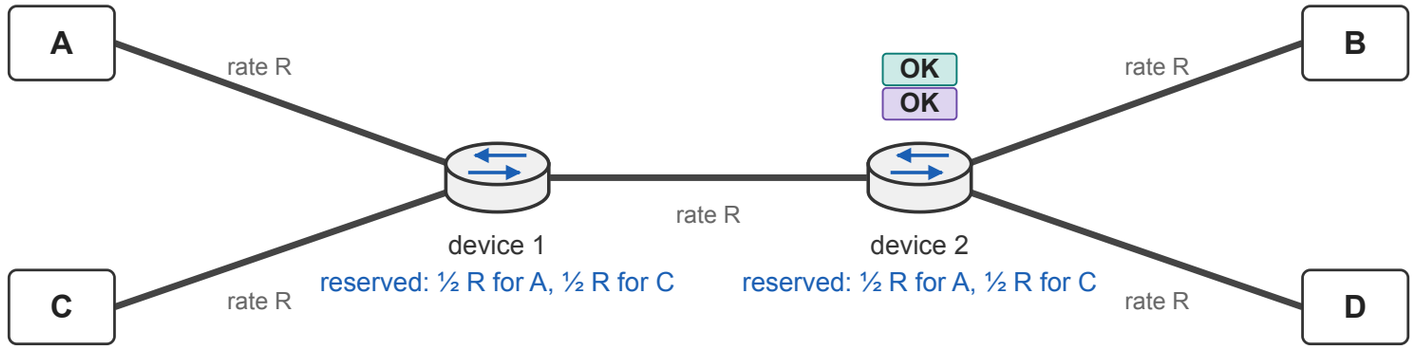
The requests reach B and D

Reserve before sending



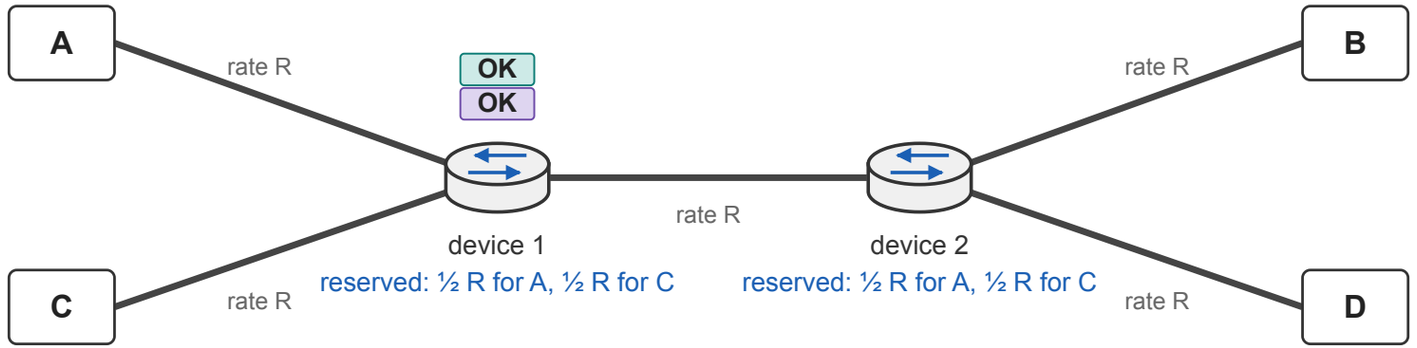
B and D confirm

Reserve before sending



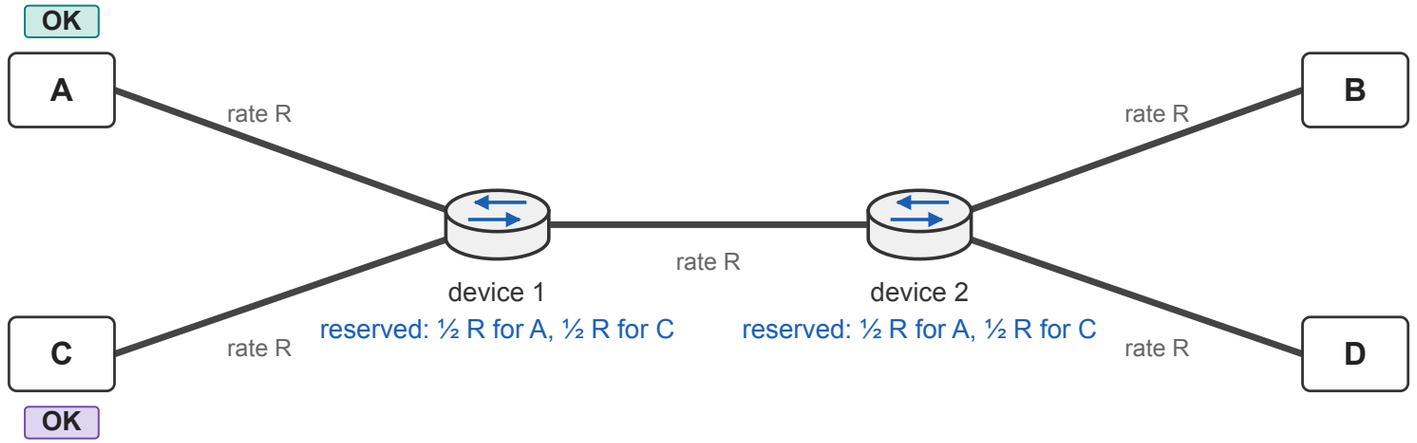
The confirmations travel back

Reserve before sending



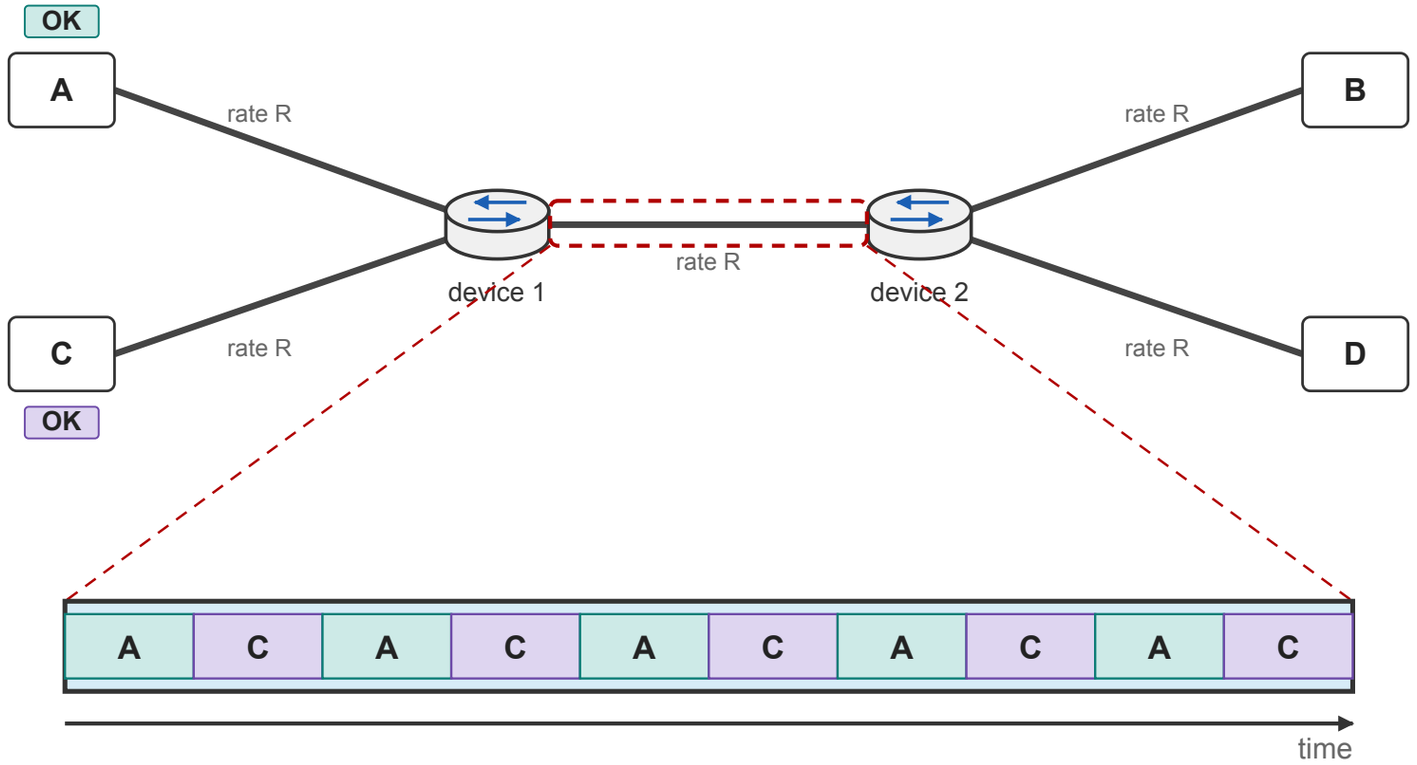
Every device on the way has already set the shares aside

Reserve before sending



A and C can start sending, each at rate ½ R

Reserve before sending



Time-shared link. A can only send in the green slots, C can only send in the purple slots.

Two families of answers

Reserve before sending

Before A sends anything, it asks every device on the path to set aside a share of its link for A, e.g., half of R.

A only sends after its reservation is confirmed.

Two families of answers

Reserve before sending

Before A sends anything, it asks every device on the path to set aside a share of its link for A, e.g., half of R.

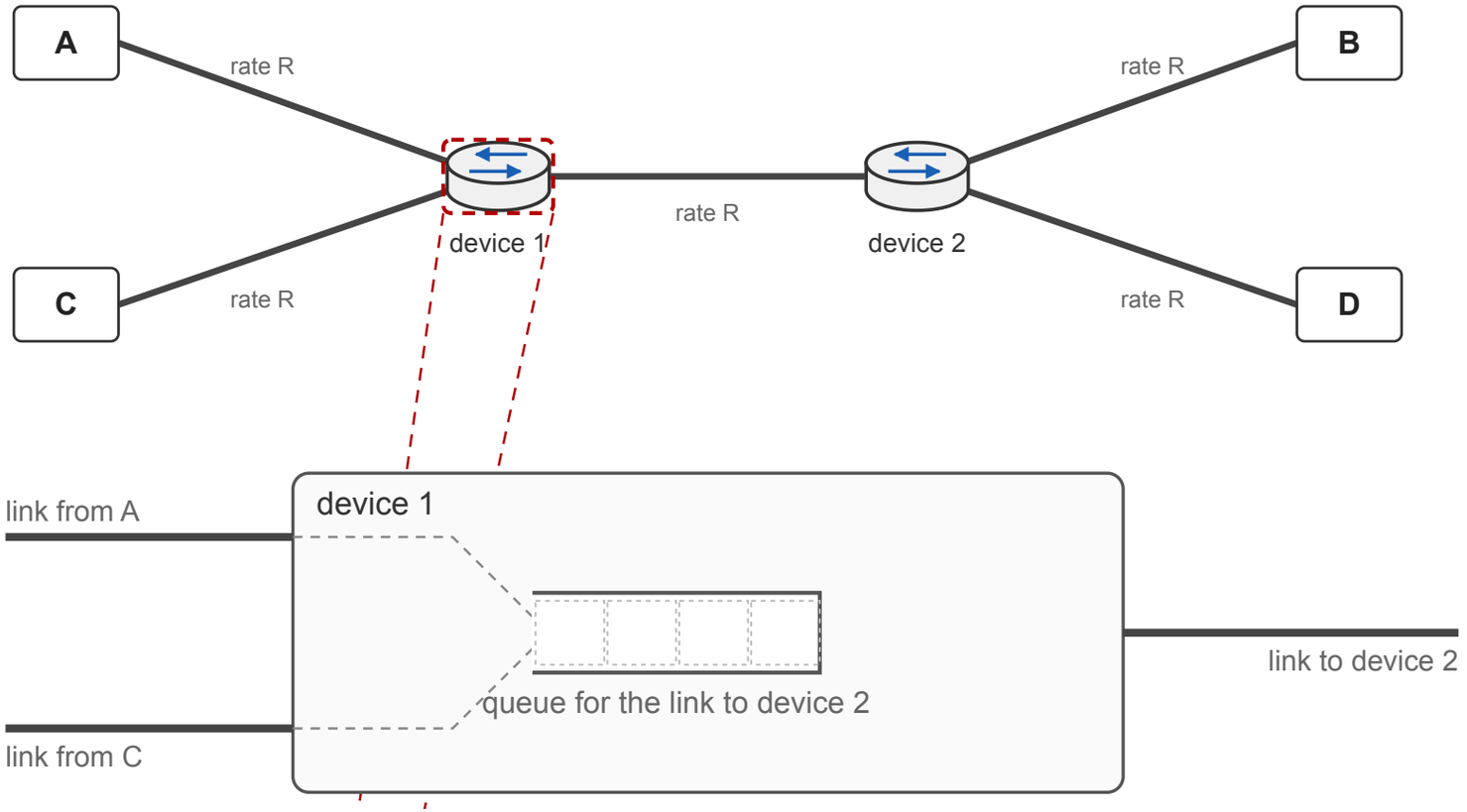
A only sends after its reservation is confirmed.

Send on demand

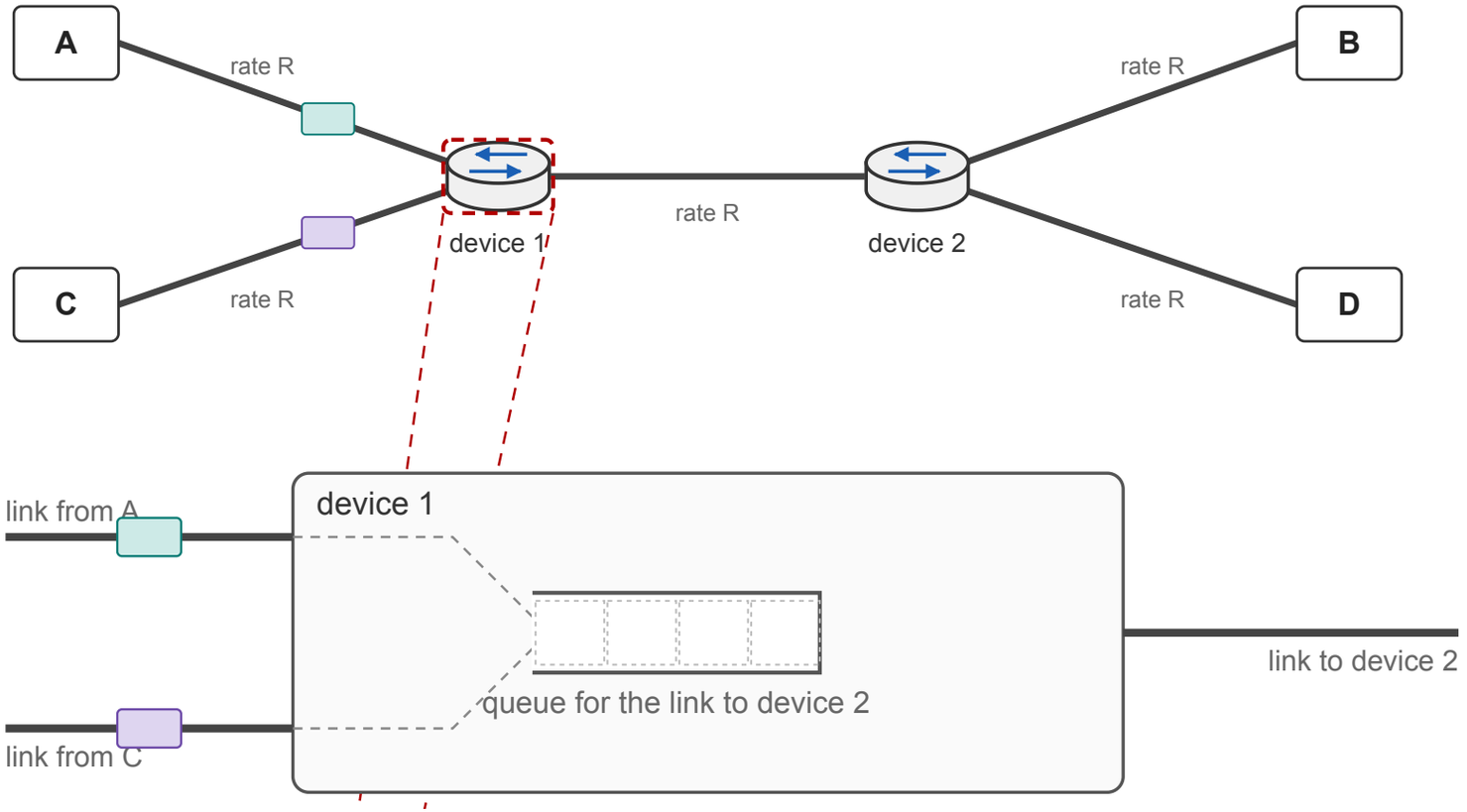
A just sends. Device 1 takes each packet as it comes in and sends it on toward B.

If the link is busy with C’s packet, A’s waits its turn. If too many packets are waiting, some are thrown away.

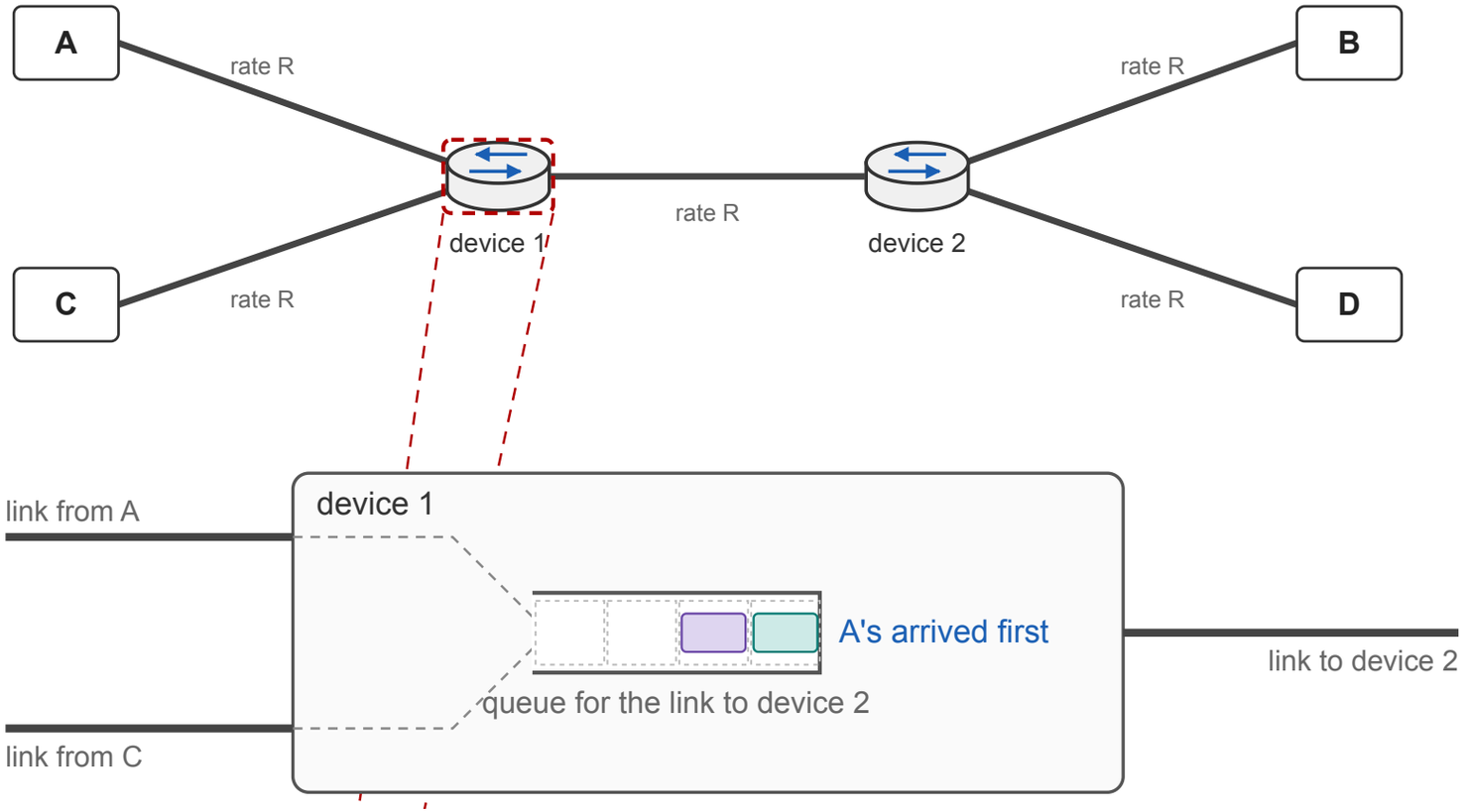
Send on demand



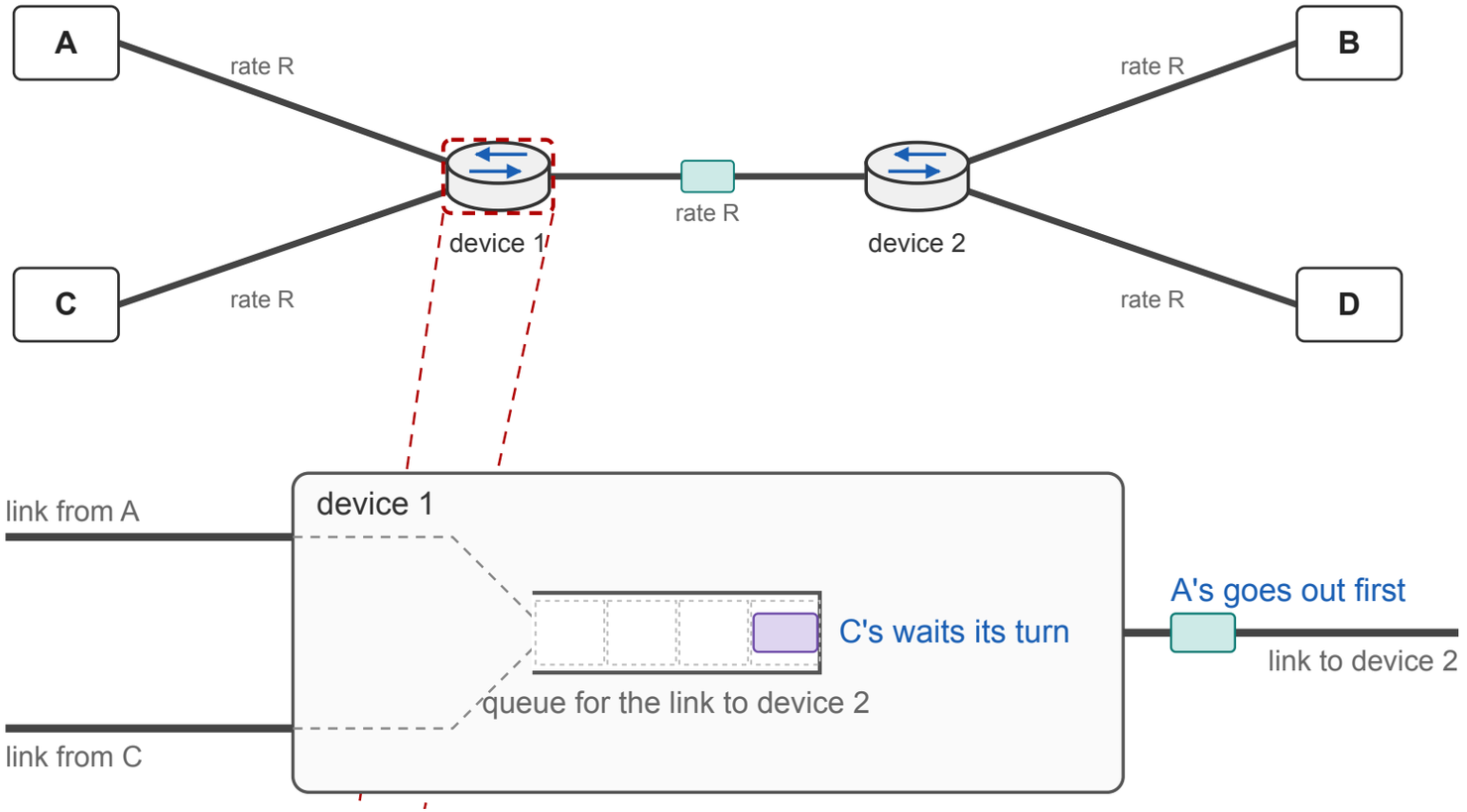
Send on demand



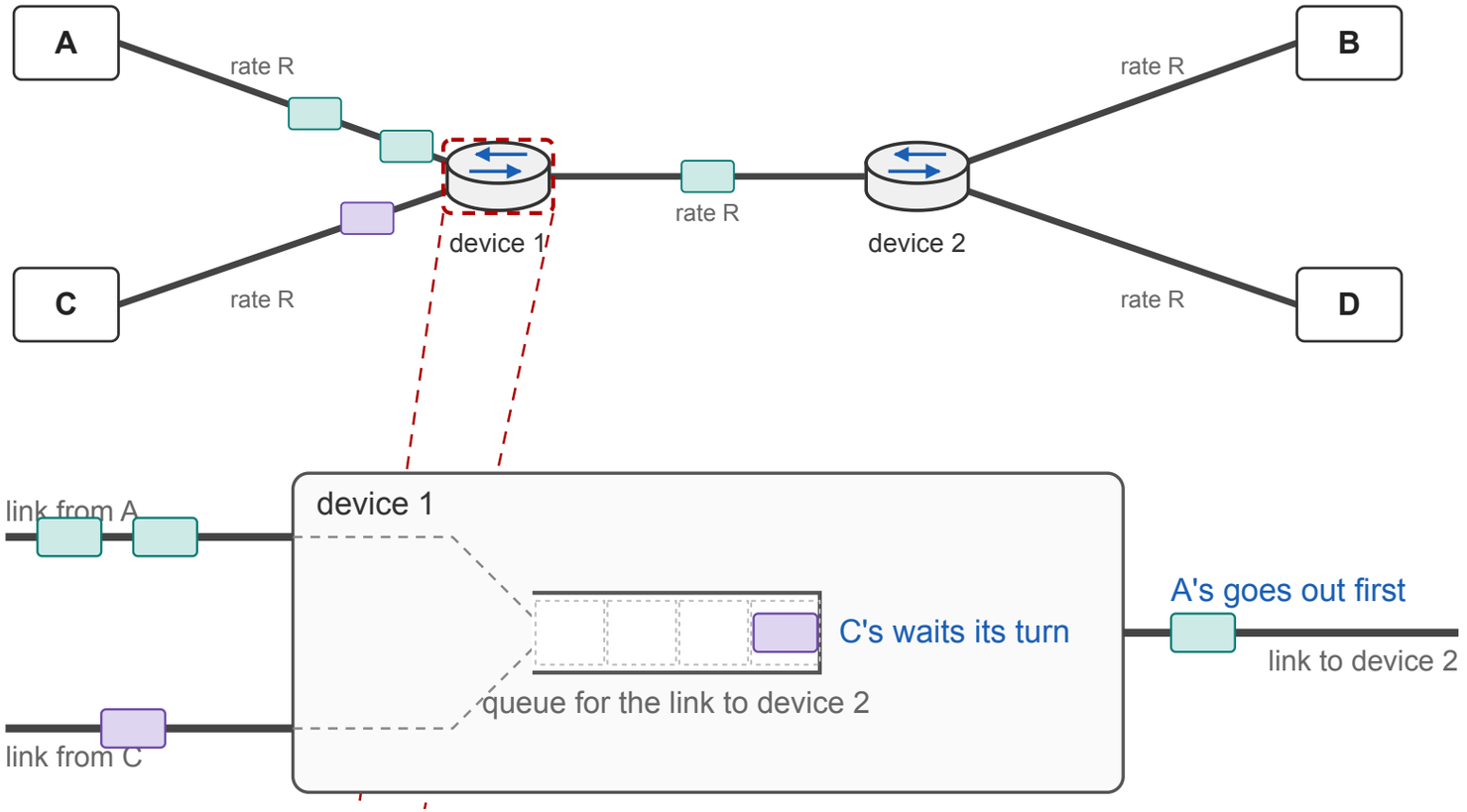
Send on demand



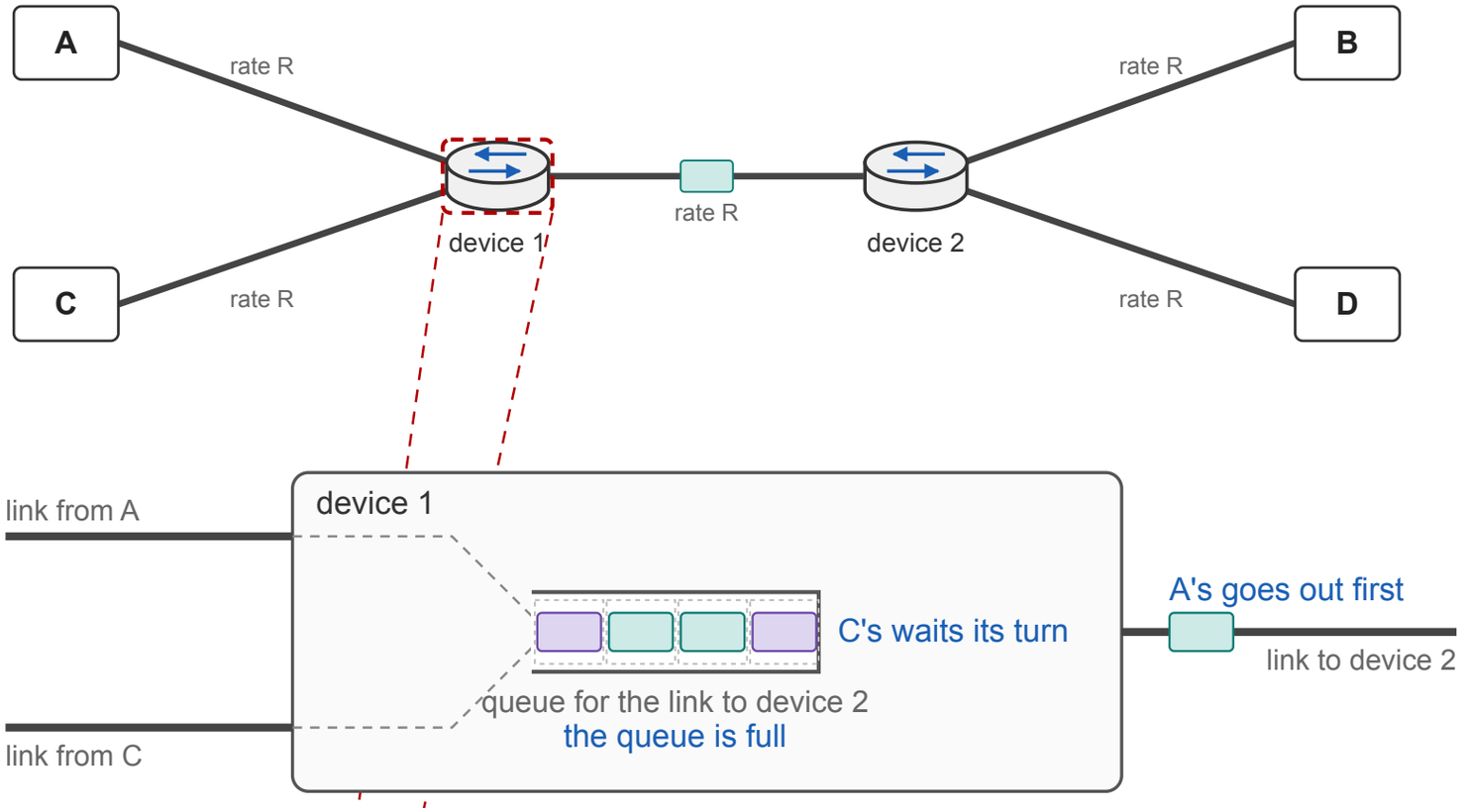
Send on demand



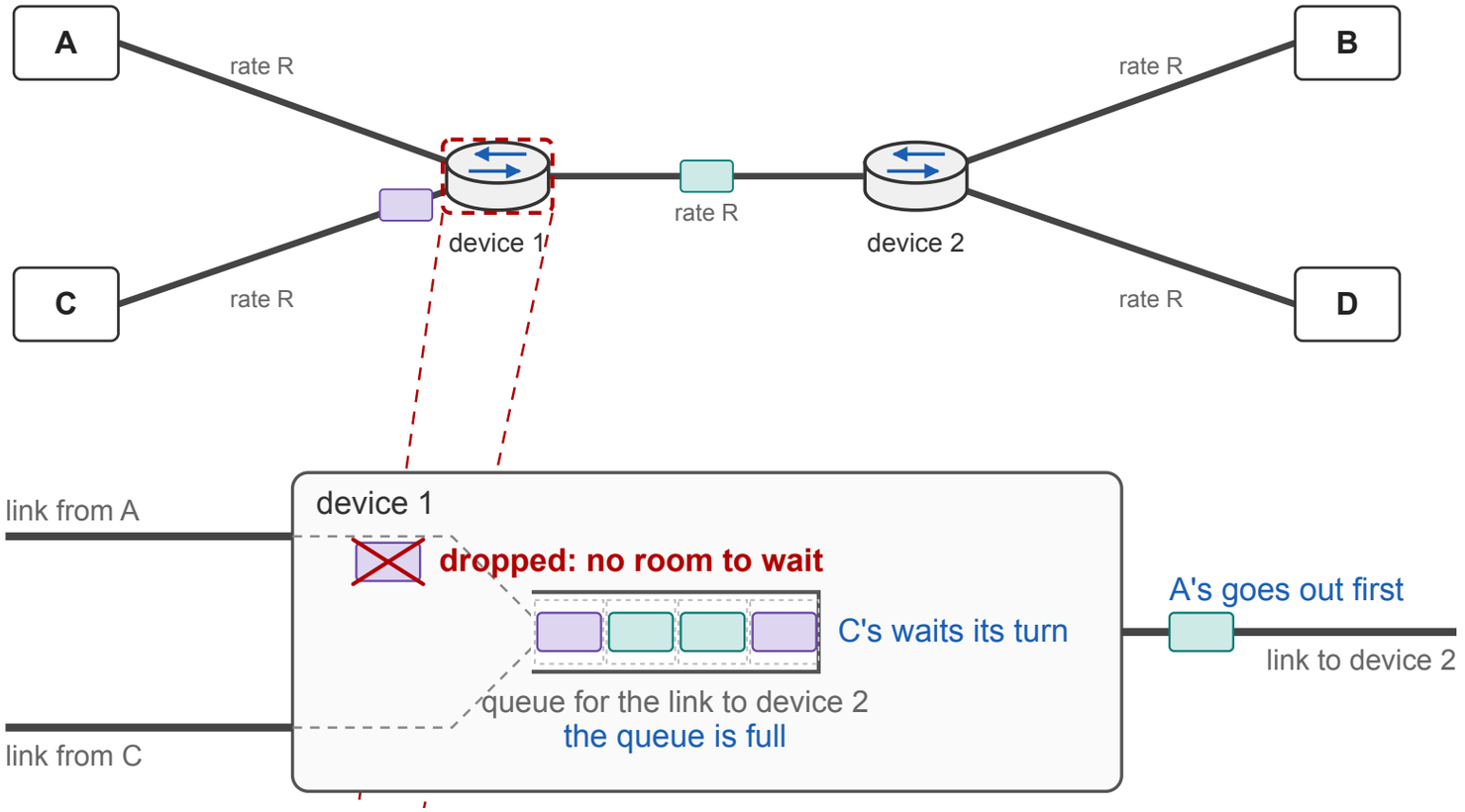
Send on demand



Send on demand



Send on demand



Two families of answers

Reserve before sending

Before A sends anything, it asks every device on the path to set aside a share of its link for A, e.g., half of R.

A only sends after its reservation is confirmed.

Send on demand

A just sends. Device 1 takes each packet as it comes in and sends it on toward B.

If the link is busy with C’s packet, A’s waits its turn. If too many packets are waiting, some are thrown away.

Two families of answers

Reserve before sending

Before A sends anything, it asks every device on the path to set aside a share of its link for A, e.g., half of R.

A only sends after its reservation is confirmed.

circuit switching

Send on demand

A just sends. Device 1 takes each packet as it comes in and sends it on toward B.

If the link is busy with C's packet, A's waits its turn. If too many packets are waiting, some are thrown away.

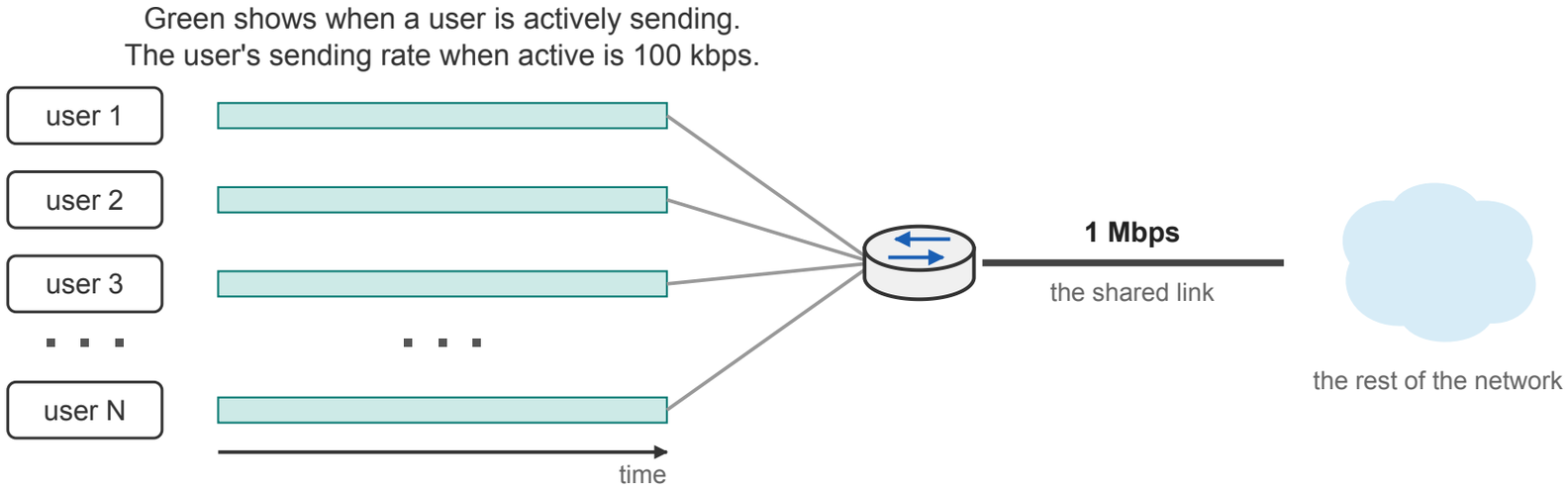
packet switching

Circuit switching or packet switching? (in-class)

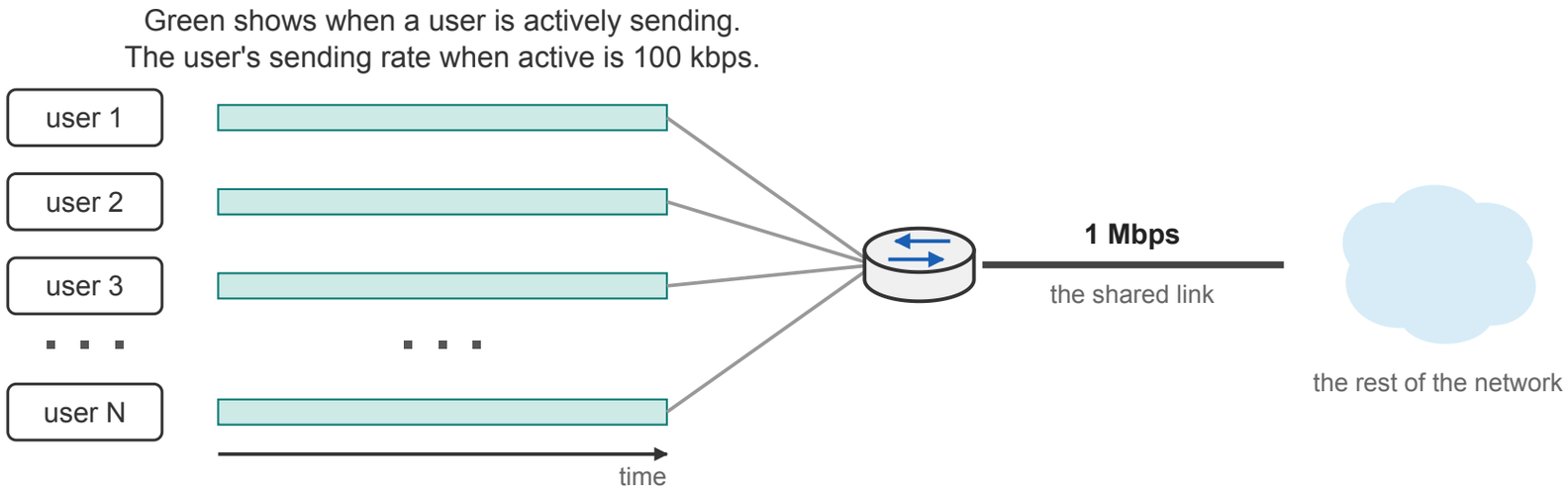
> Think about the pros and cons of each. Come up with two scenarios: one where circuit switching works better, and one where packet switching works better.

Reset

Example of one main difference

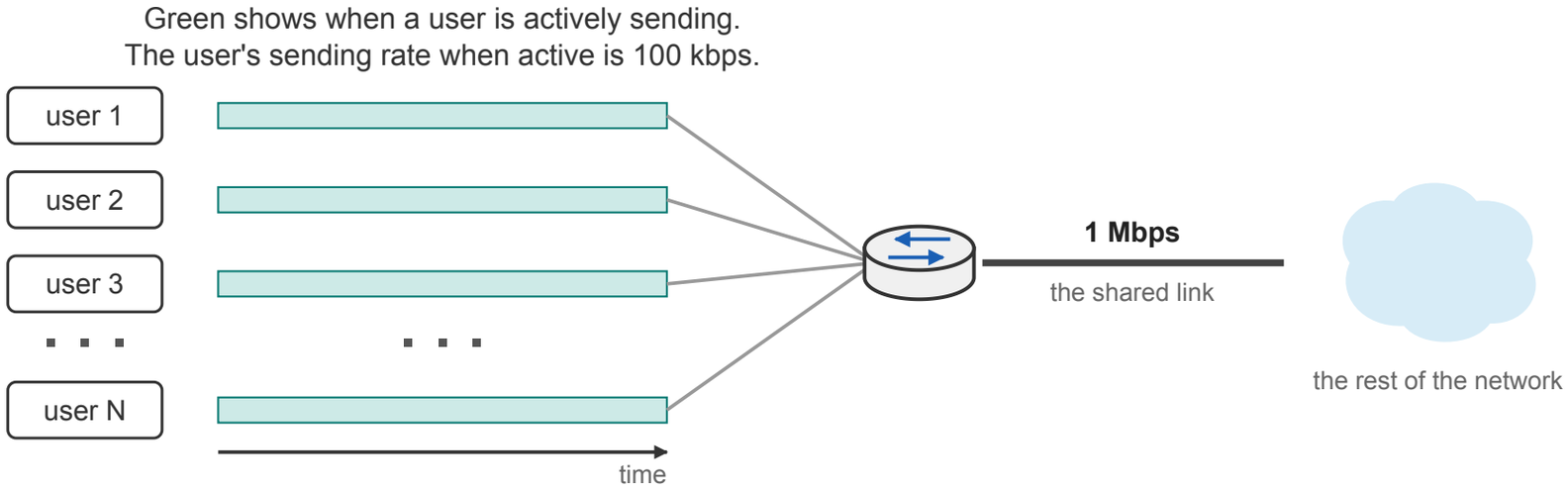


Example of one main difference



Circuit switching: can support 10 reservations, each at 100 kbps, so 10 concurrent users. If N > 10, the rest wait for a successful reservation.

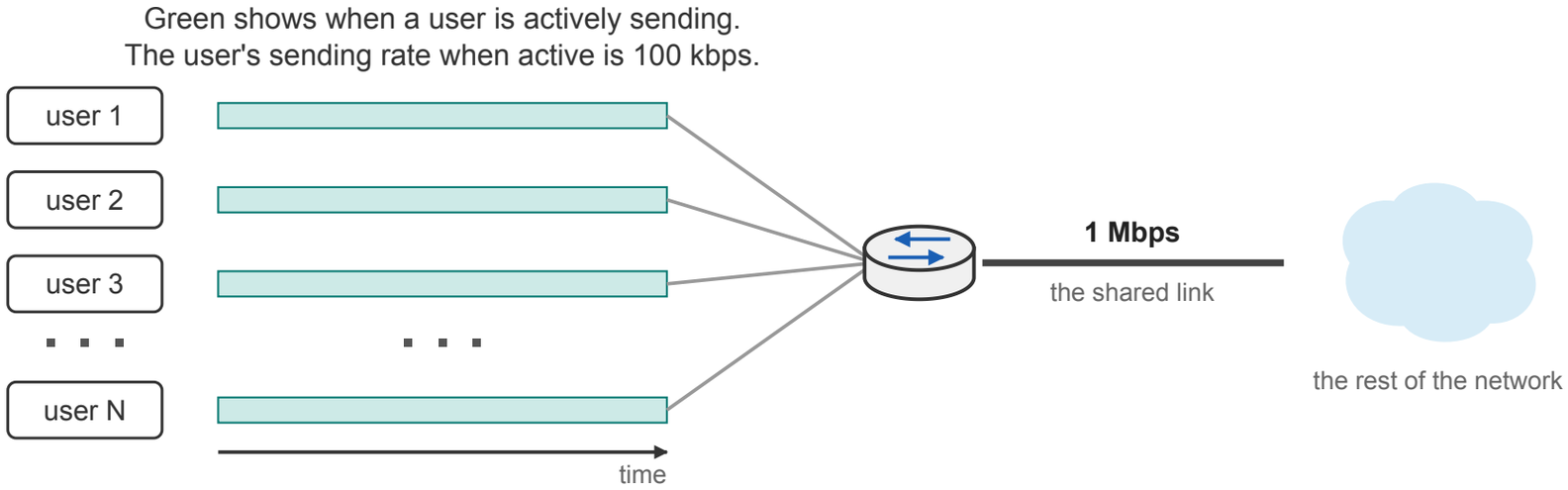
Example of one main difference



Circuit switching: can support 10 reservations, each at 100 kbps, so 10 concurrent users. If N > 10, the rest wait for a successful reservation.

Packet switching: for N > 10, Pr(more than 10 active at a time) = 1. The packets collide in the queue, keep getting dropped, keep getting retransmitted: congestion collapse.

Example of one main difference

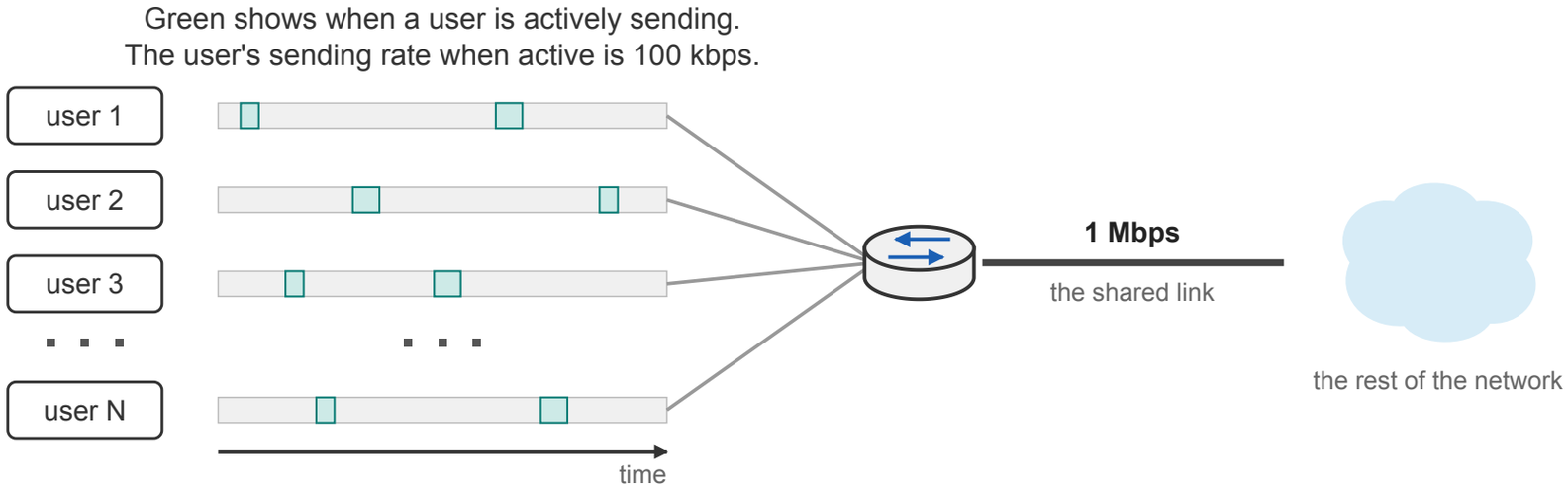


Circuit switching: can support 10 reservations, each at 100 kbps, so 10 concurrent users. If N > 10, the rest wait for a successful reservation.

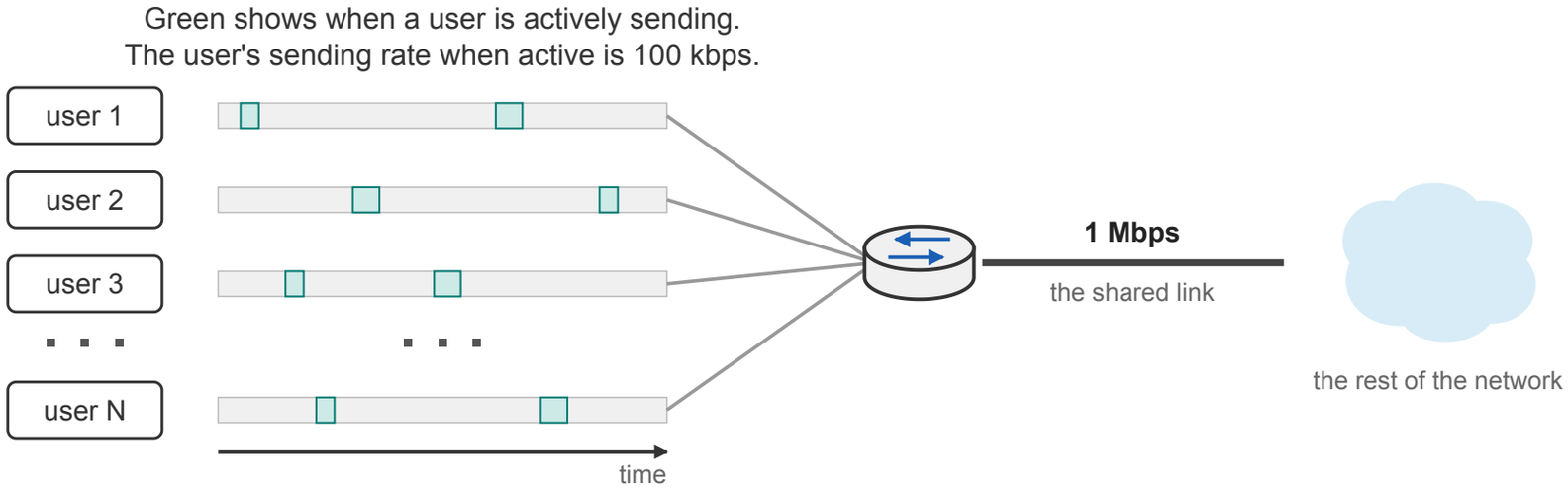
Packet switching: for N > 10, Pr(more than 10 active at a time) = 1. The packets collide in the queue, keep getting dropped, keep getting retransmitted: congestion collapse.

Would you do circuit switching or packet switching? Why?

Example of one main difference

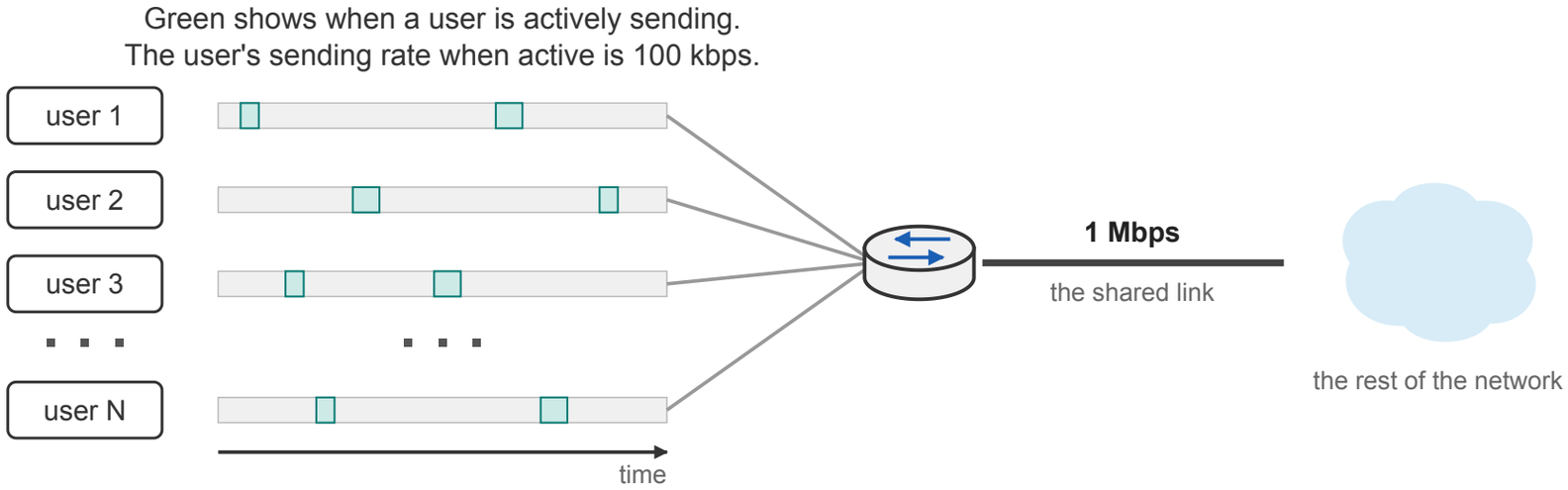


Example of one main difference



Circuit switching: still at most 10 concurrent users (10 reservations, each at 100 kbps).

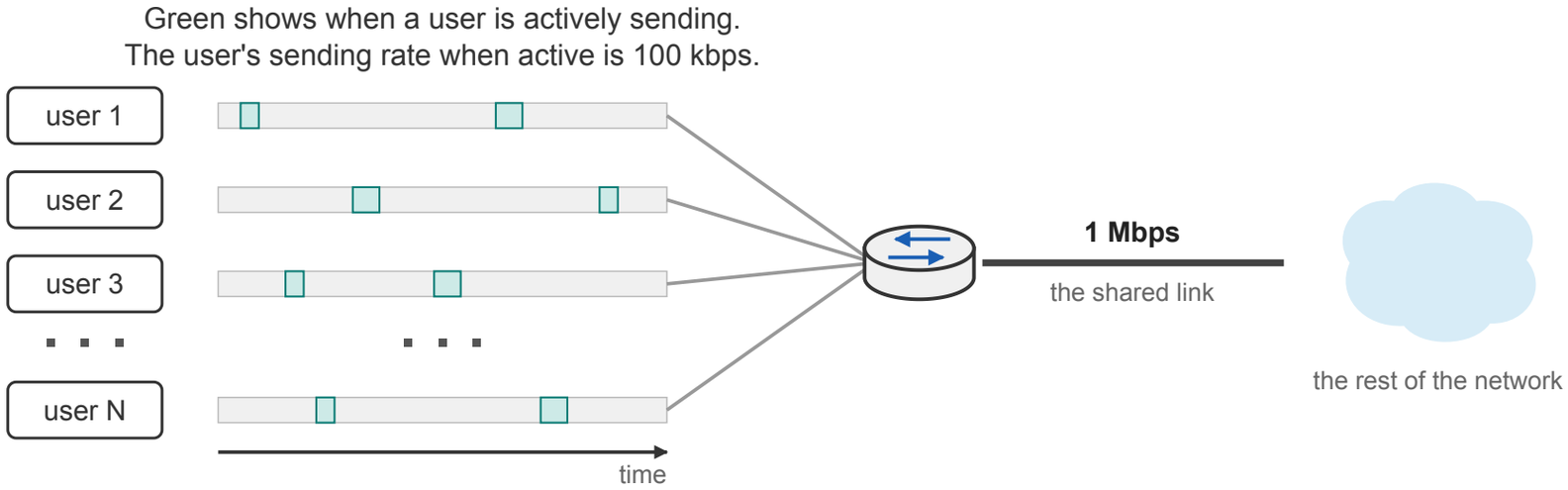
Example of one main difference



Circuit switching: still at most 10 concurrent users (10 reservations, each at 100 kbps).

Packet switching: with 35 users, Pr(more than 10 active at a time) = $\sum_{k=11..35}$ (35 choose k) 0.1^k $0.9^{35-k} \approx 0.0004$. So 35 users can use the network with a small probability of a persistent queue build-up.

Example of one main difference



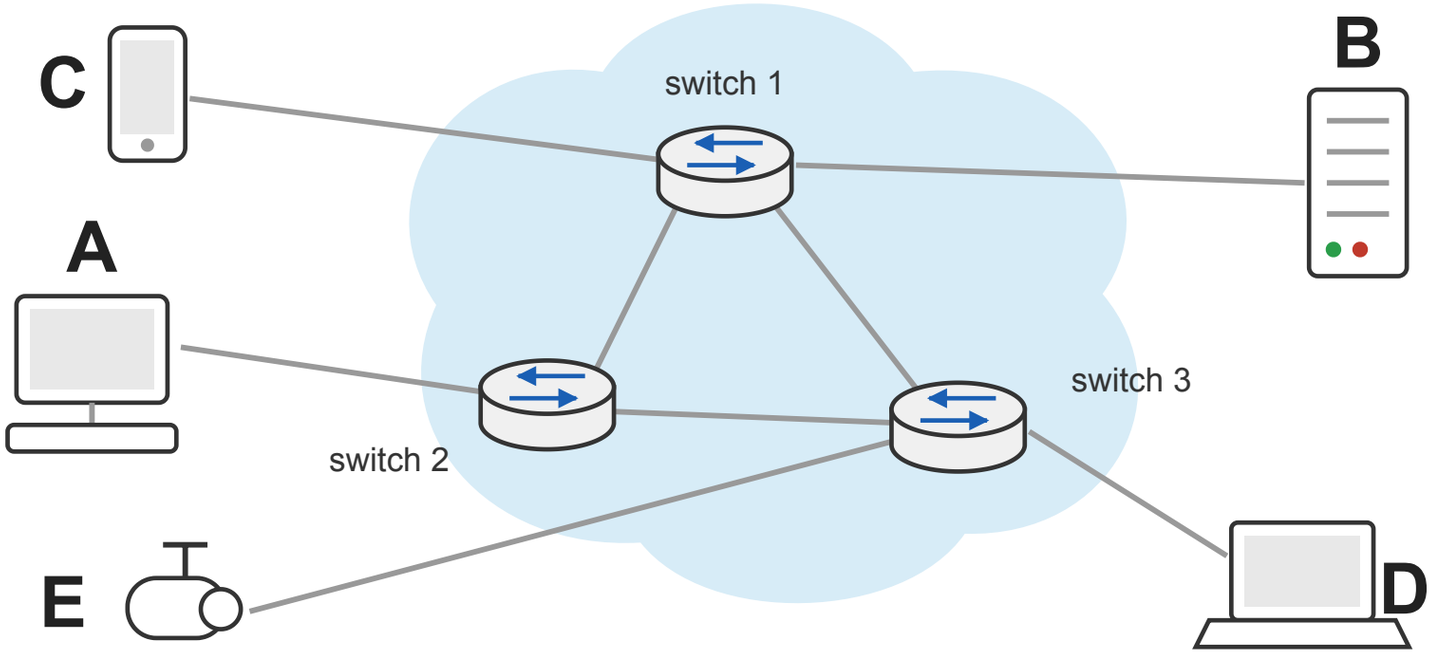
Circuit switching: still at most 10 concurrent users (10 reservations, each at 100 kbps).

Packet switching: with 35 users, Pr(more than 10 active at a time) = $\sum_{k=11..35}$ (35 choose k) 0.1^k $0.9^{35-k} \approx 0.0004$. So 35 users can use the network with a small probability of a persistent queue build-up.

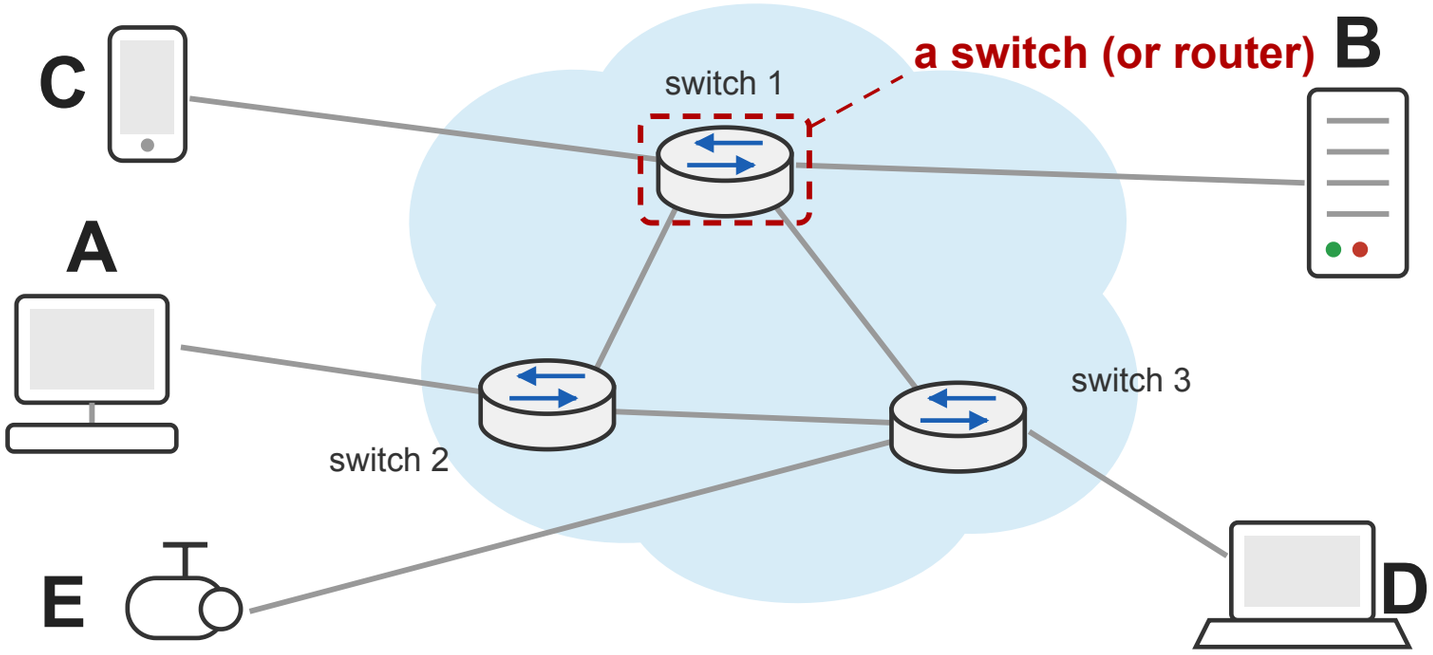
Would you do circuit switching or packet switching? Why?

For the rest of this course, we assume packet switching, the more prominent of the two over the Internet.

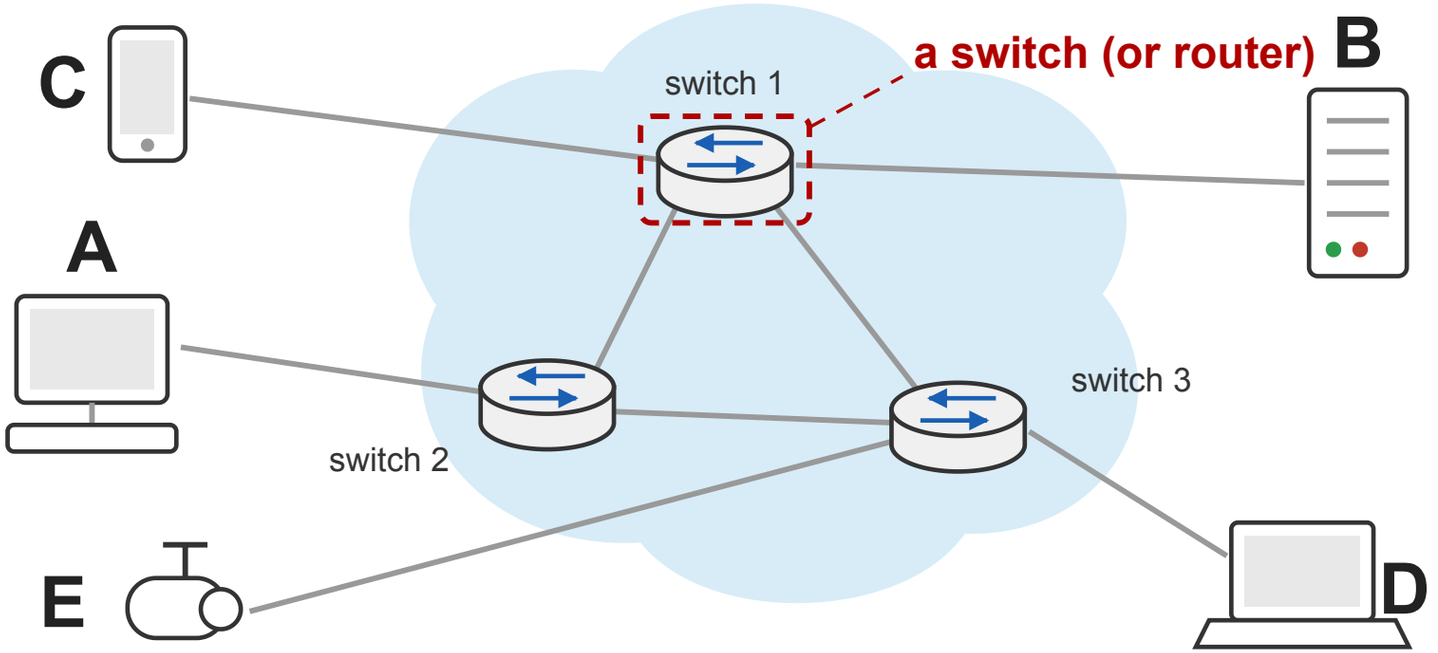
Inside a network device



Inside a network device

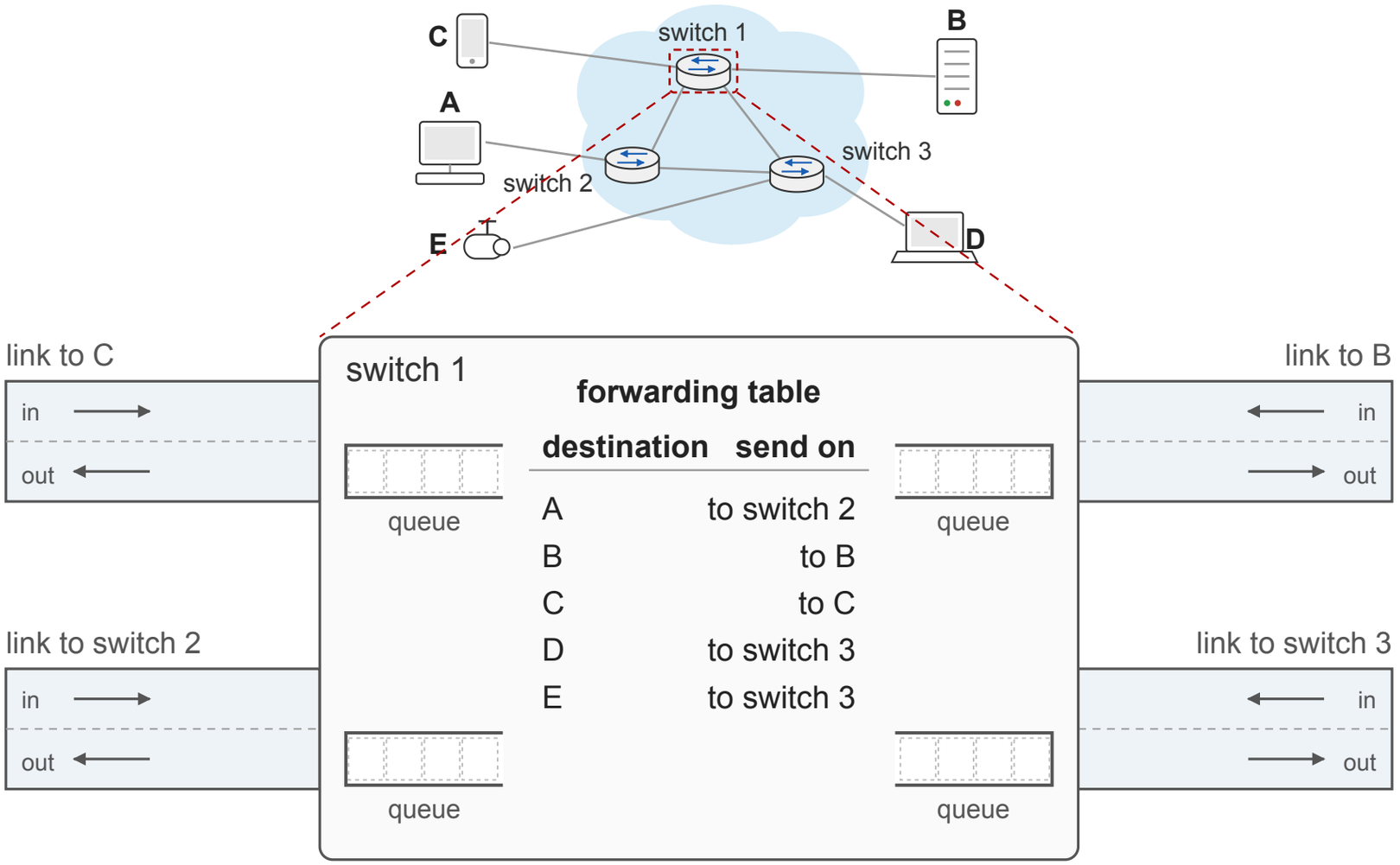


Inside a network device

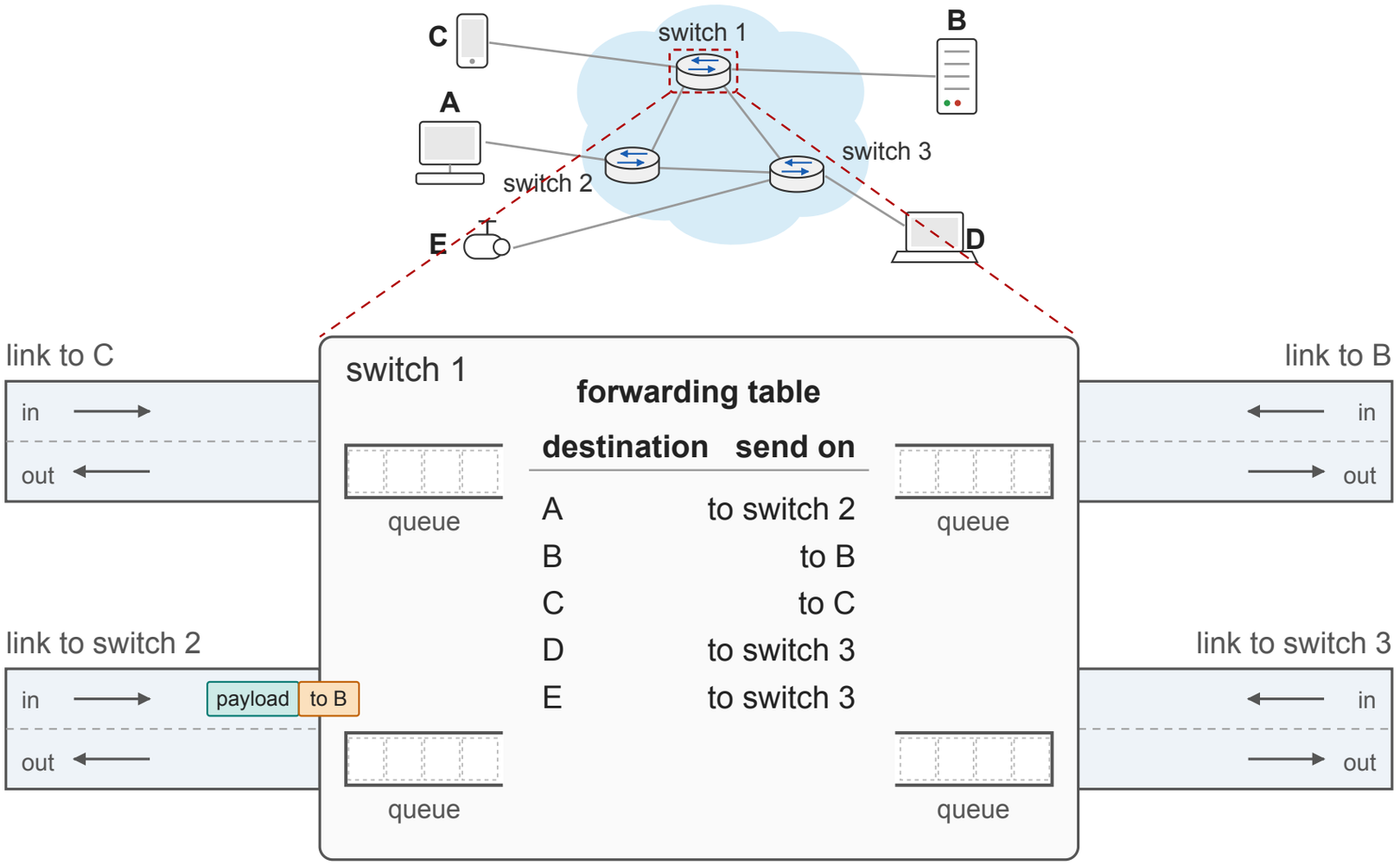


Switches and routers differ in some ways. For now, we call both of them switches.

Inside a switch

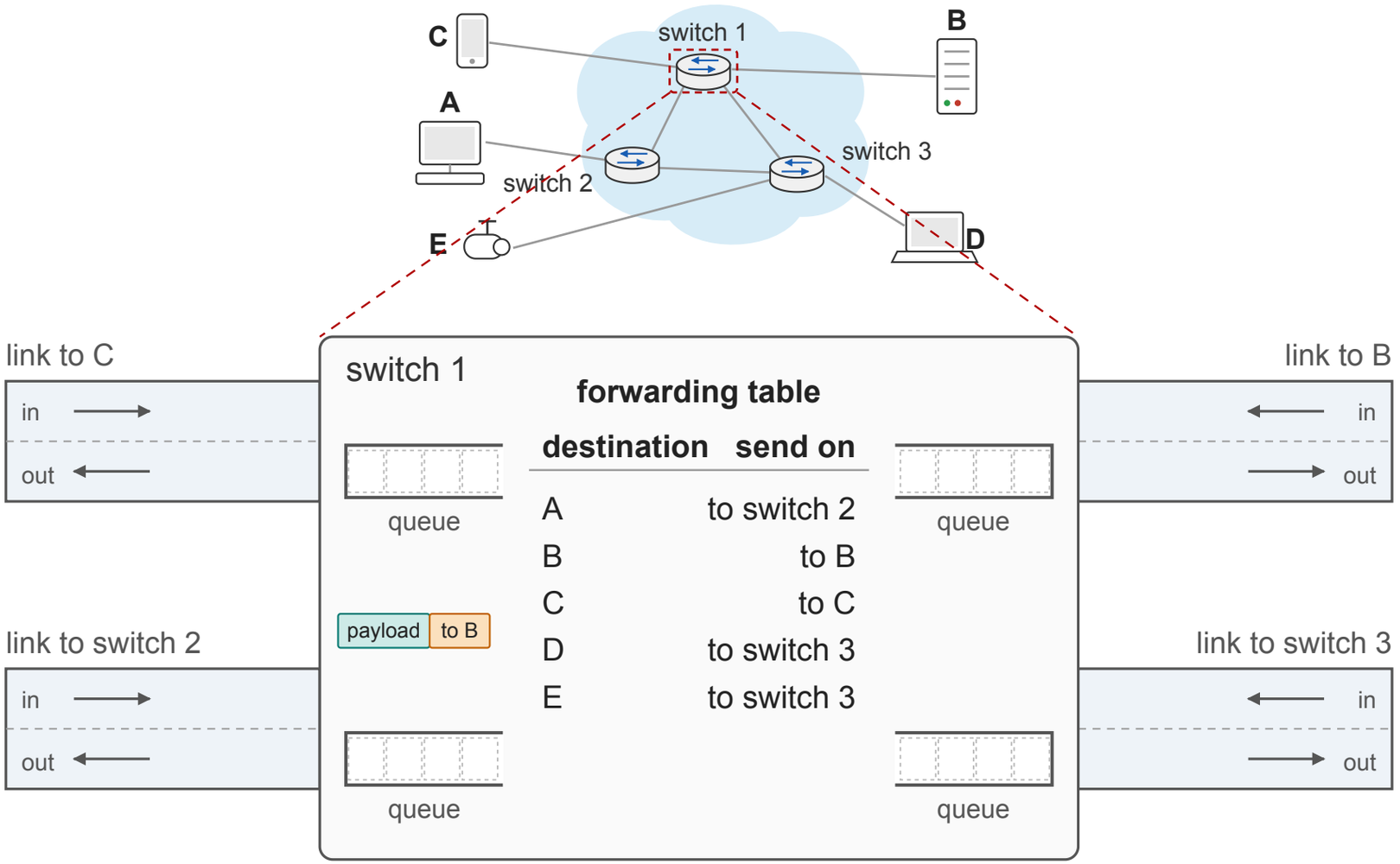


Inside a switch



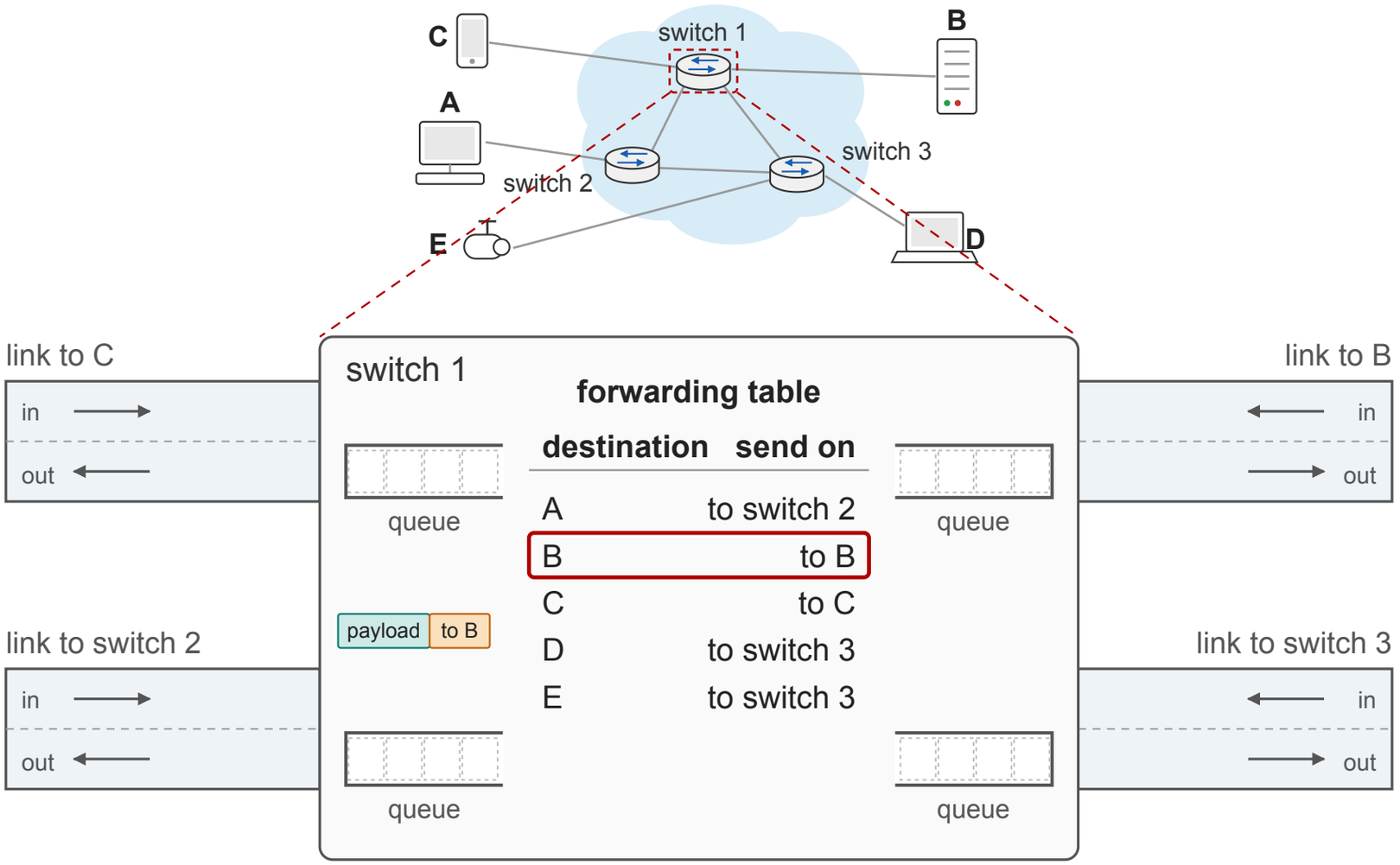
A packet from A arrives bit by bit on the link from switch 2. Its metadata says: to B

Inside a switch



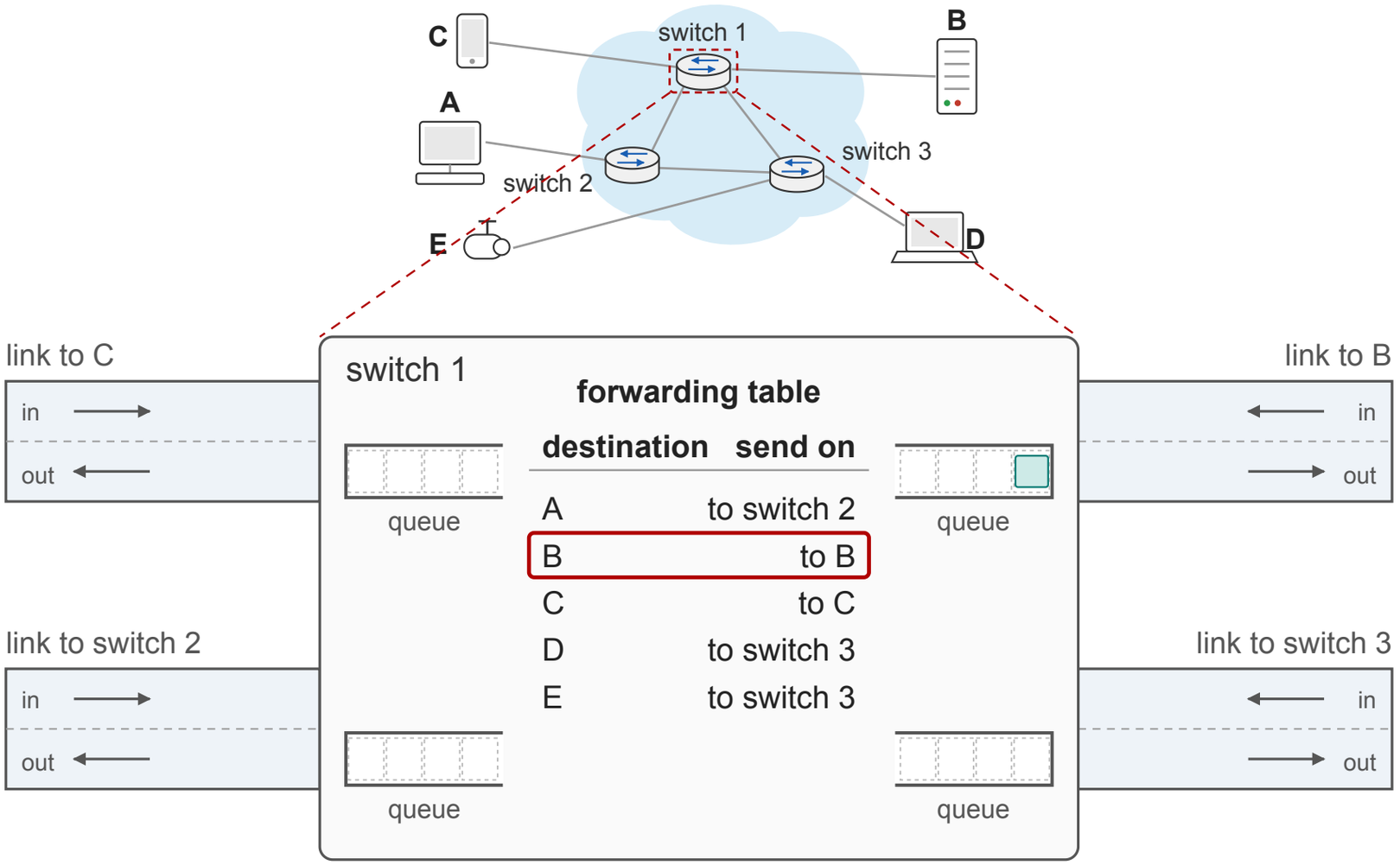
Store and forward: the switch waits for the whole packet before it does anything with it. Why?

Inside a switch



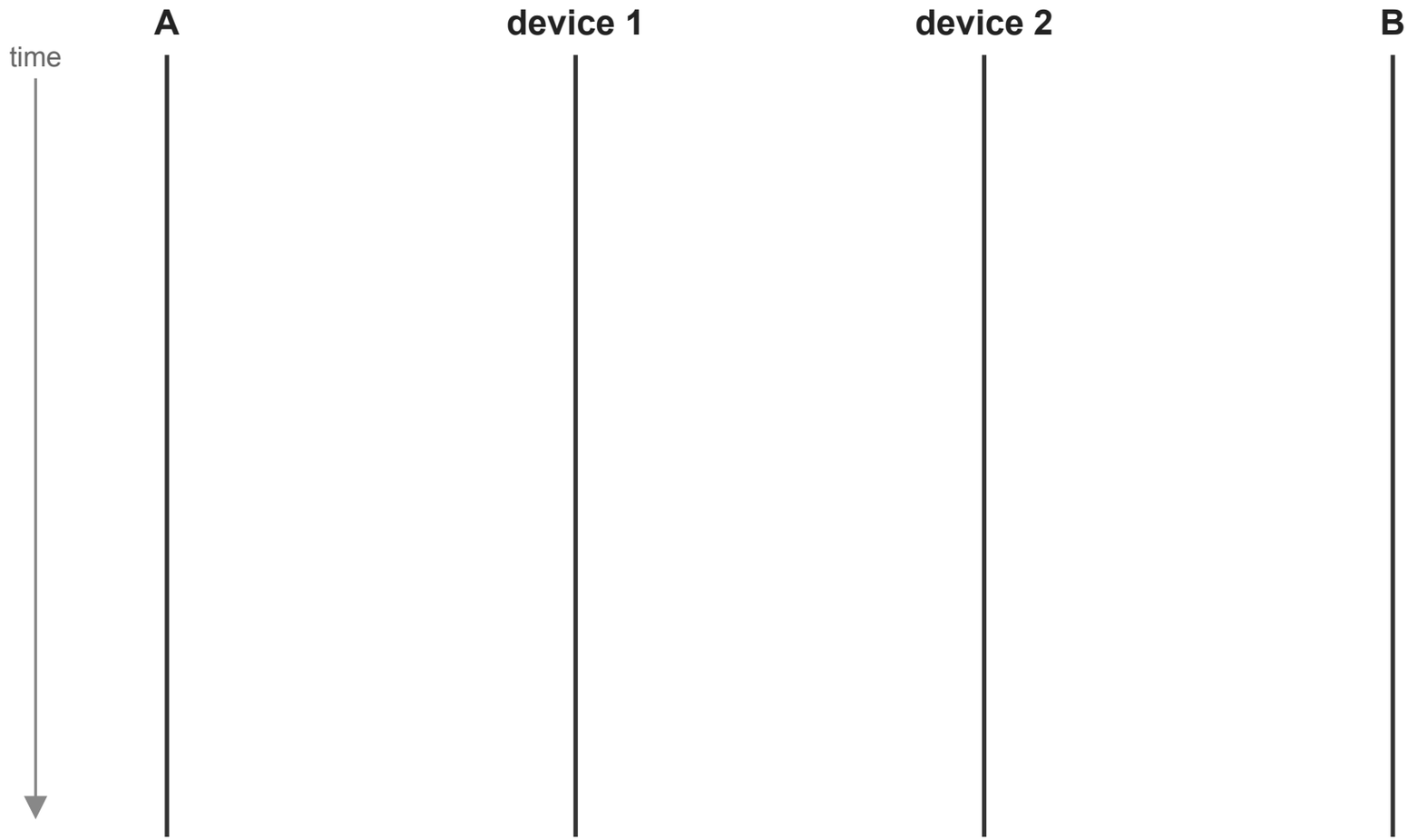
Look up B in the forwarding table: the link to B

Inside a switch

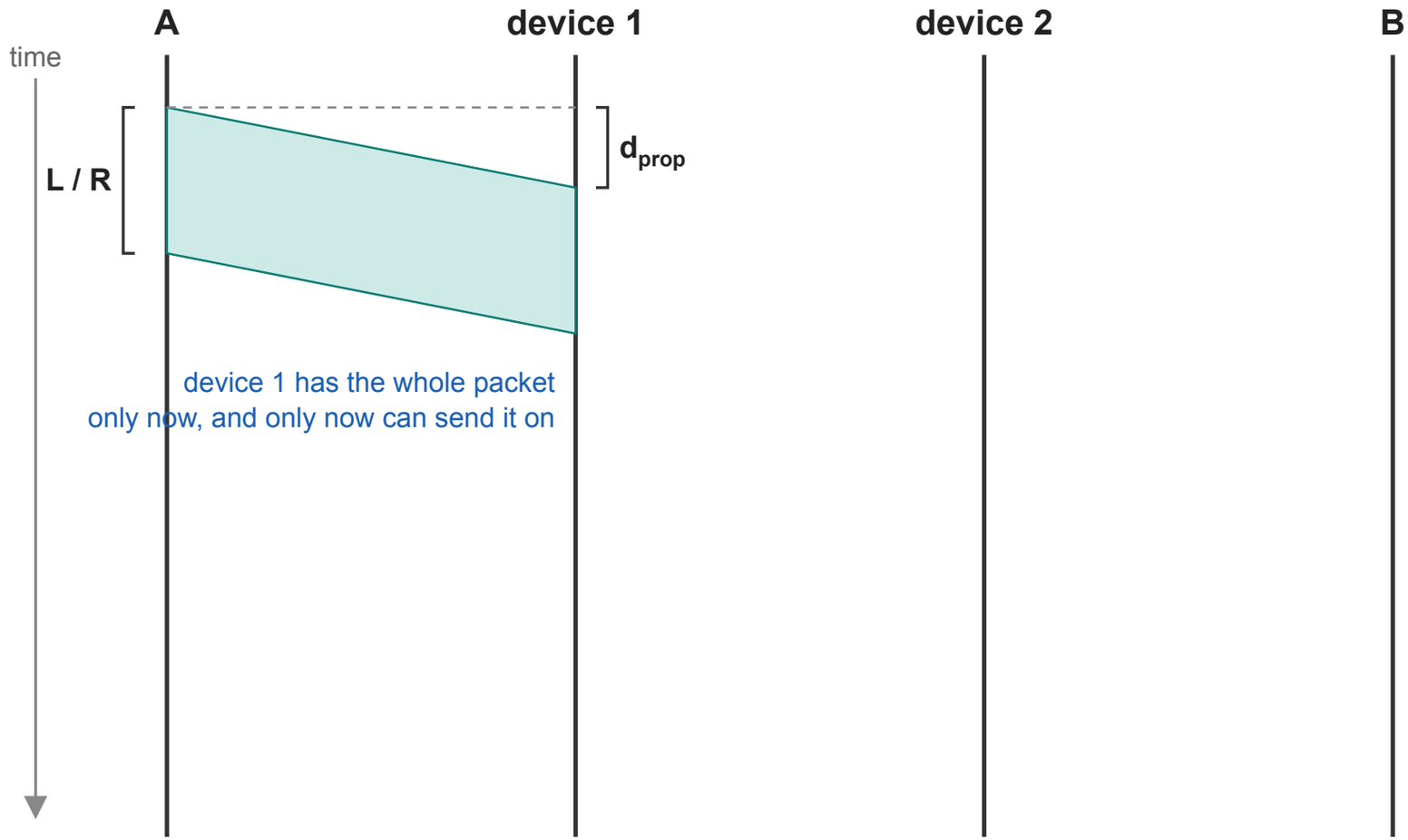


Forward: into that link's queue, and out when the link is free

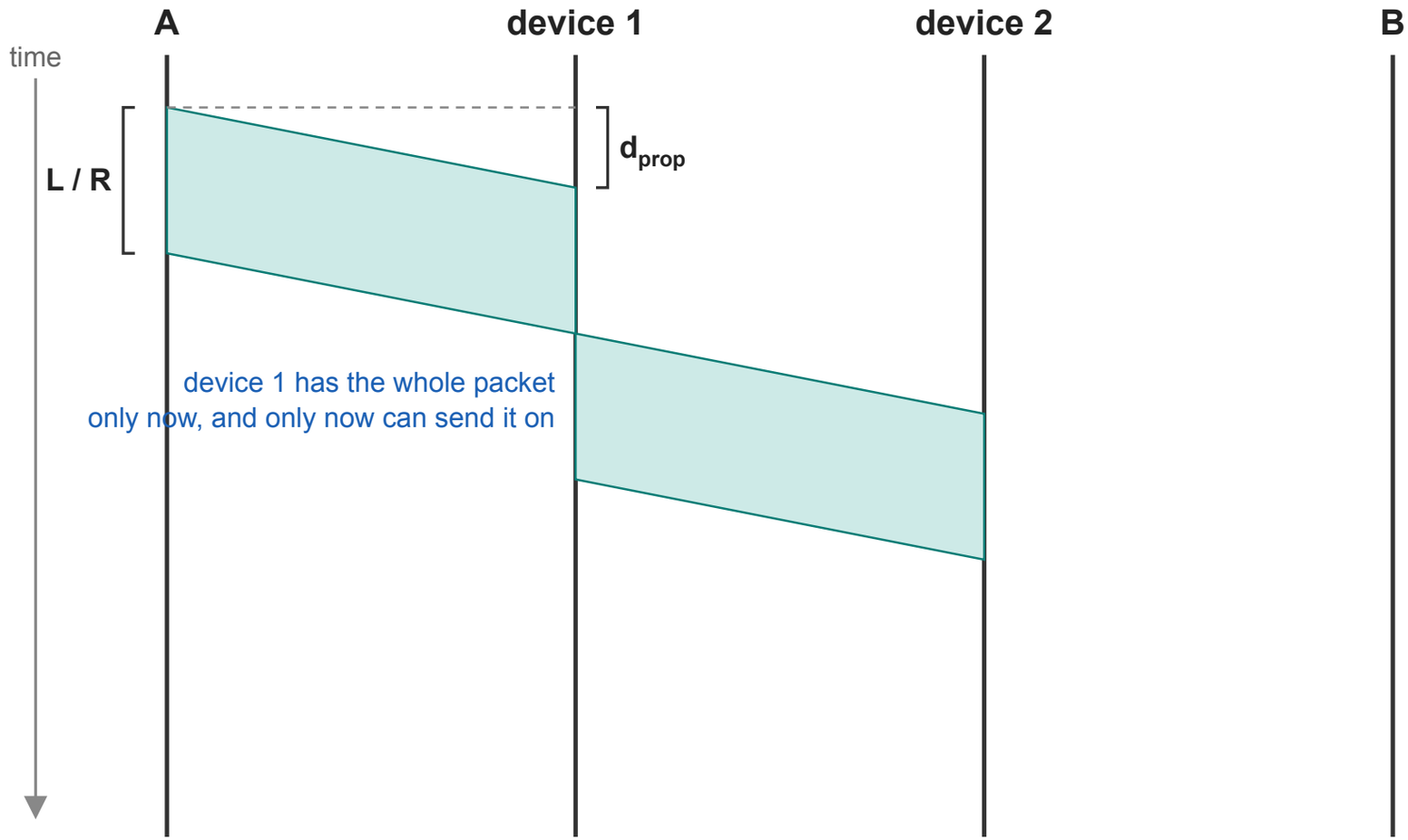
Store and forward



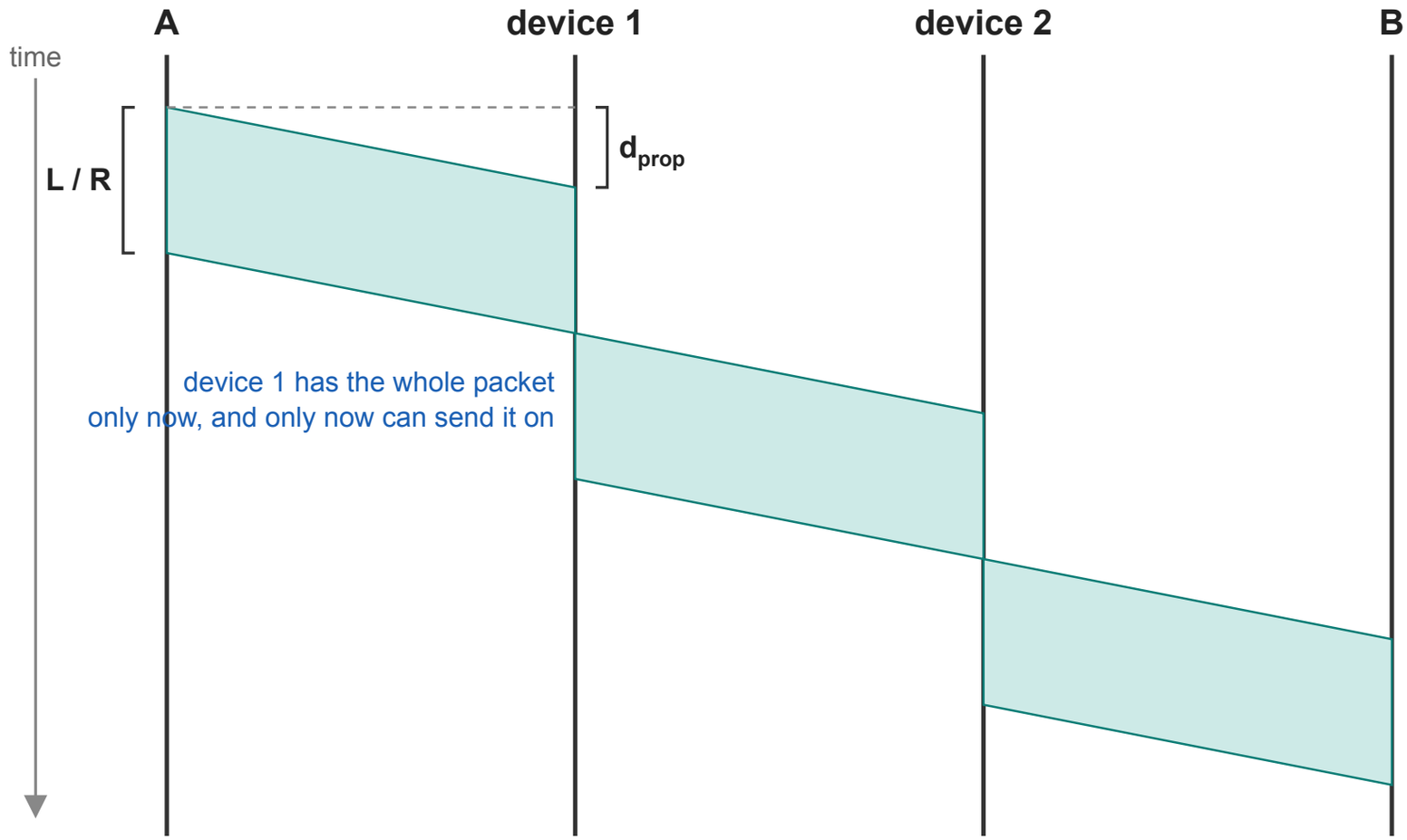
Store and forward



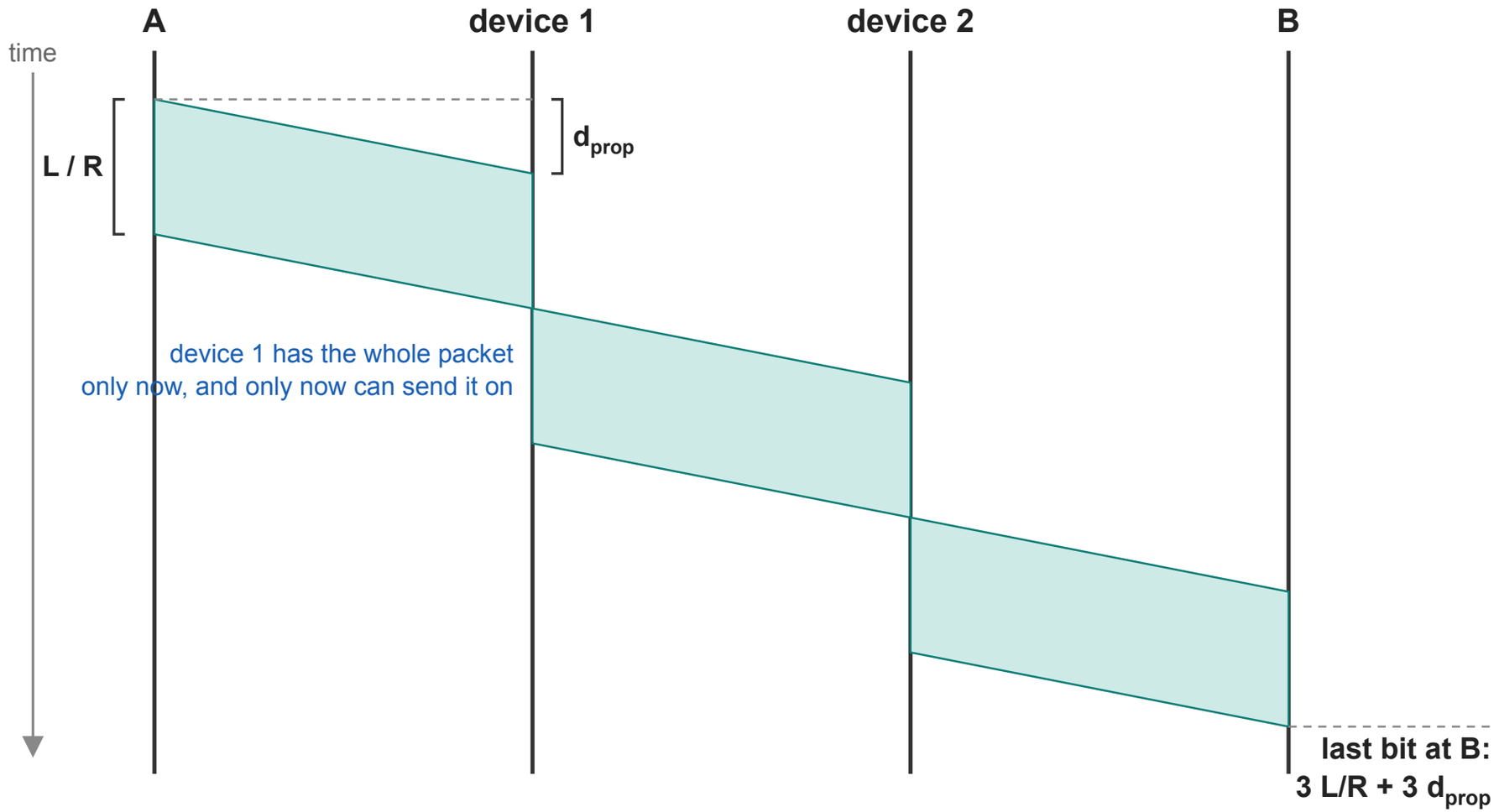
Store and forward



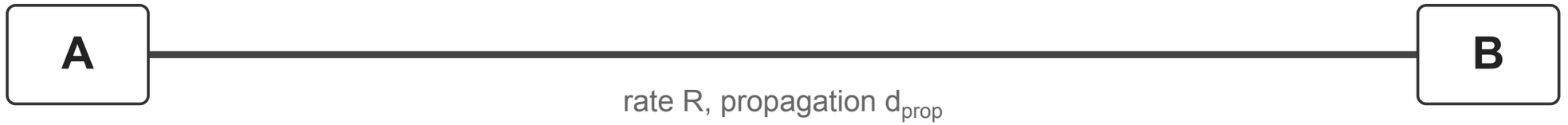
Store and forward



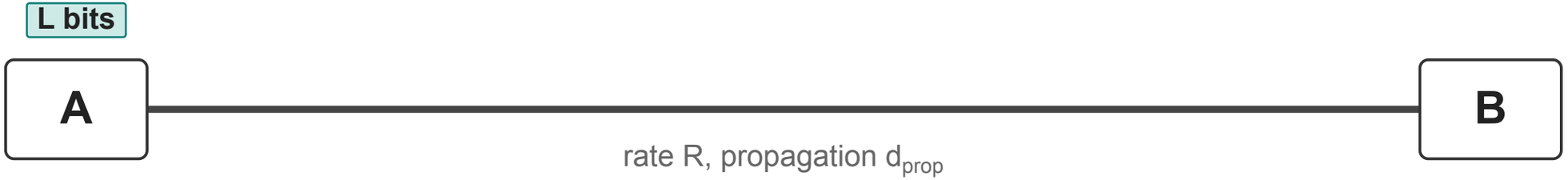
Store and forward



Revisiting delay



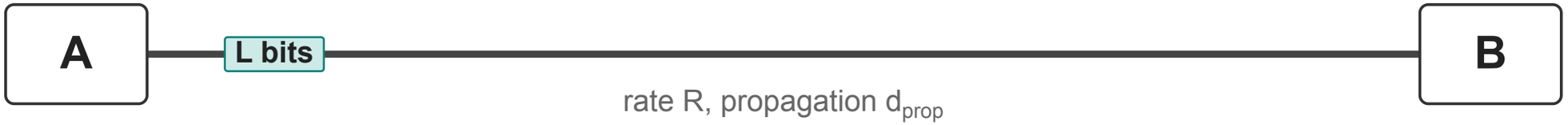
Revisiting delay



Time it takes for the packet to be fully received at B = $d_{proc,A}$ process at A

A prepares the packet, checksum and all: $d_{proc,A}$

Revisiting delay



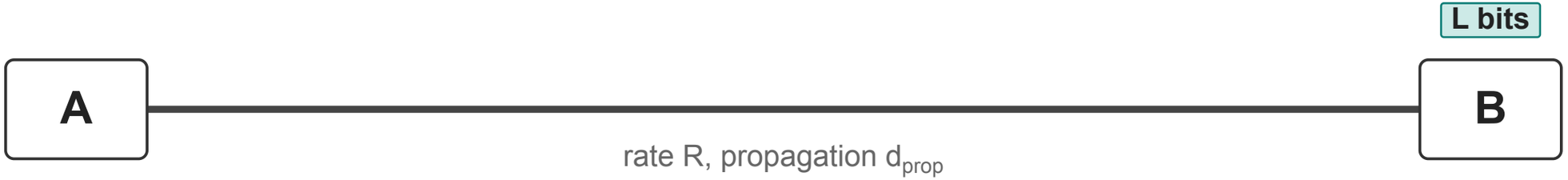
Time it takes for the packet to be fully received at B = $d_{proc,A}$ + L / R

$d_{proc,A}$: process at A

+ L / R: transmit

A puts the packet on the link, bit by bit: L / R

Revisiting delay



Time it takes for the packet to be fully received at B = $d_{proc,A}$ + L / R + d_{prop}

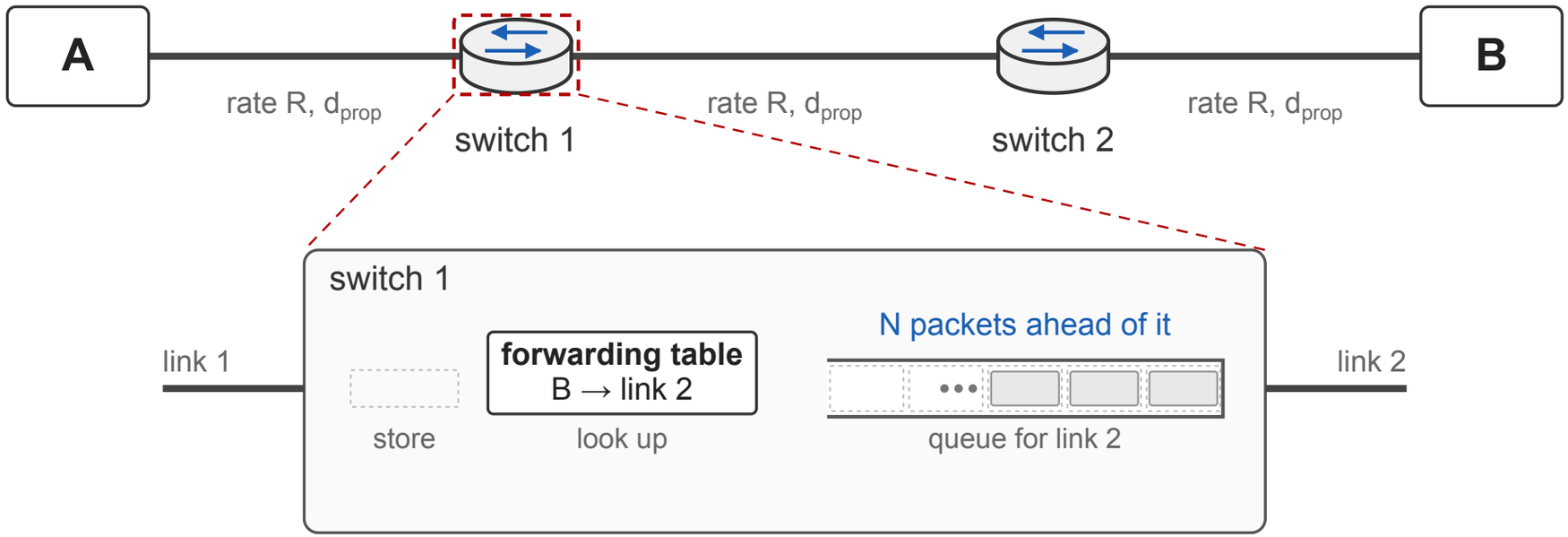
$d_{proc,A}$: process at A

L / R: transmit

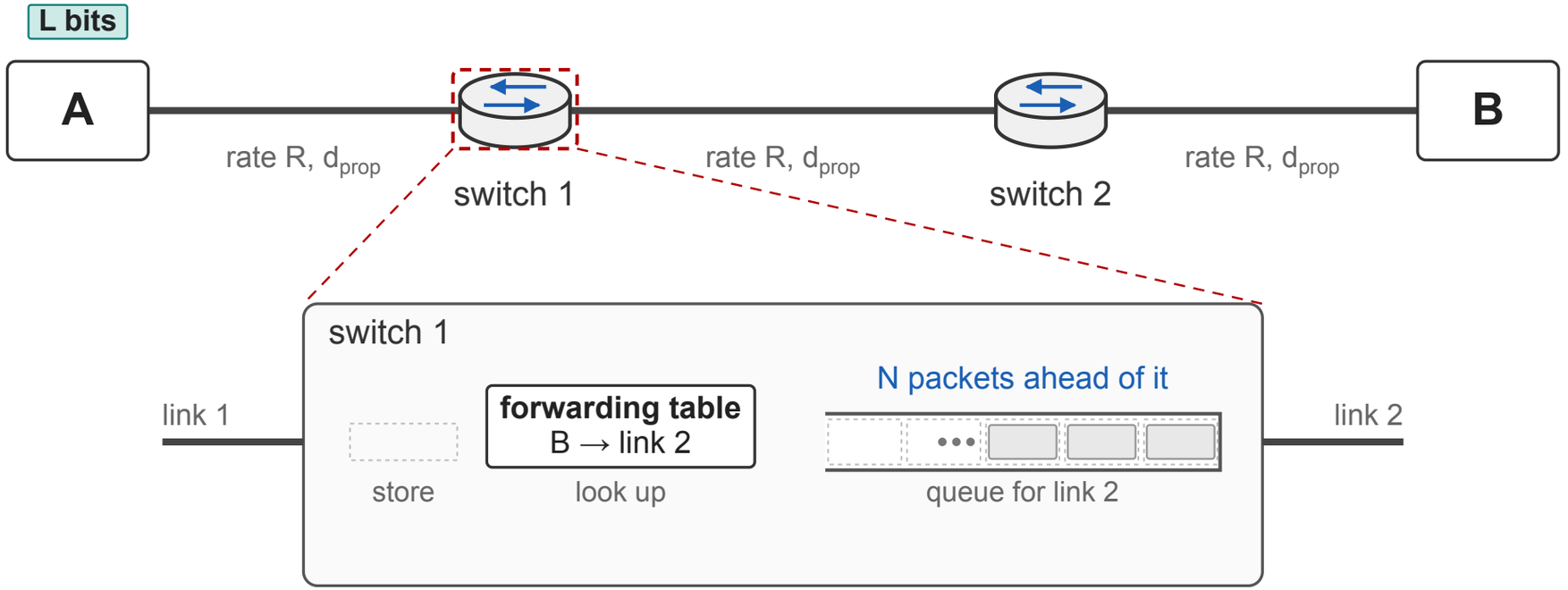
d_{prop}: propagate

The bits travel down the link, and the last one reaches B: d_{prop}

Revisiting delay



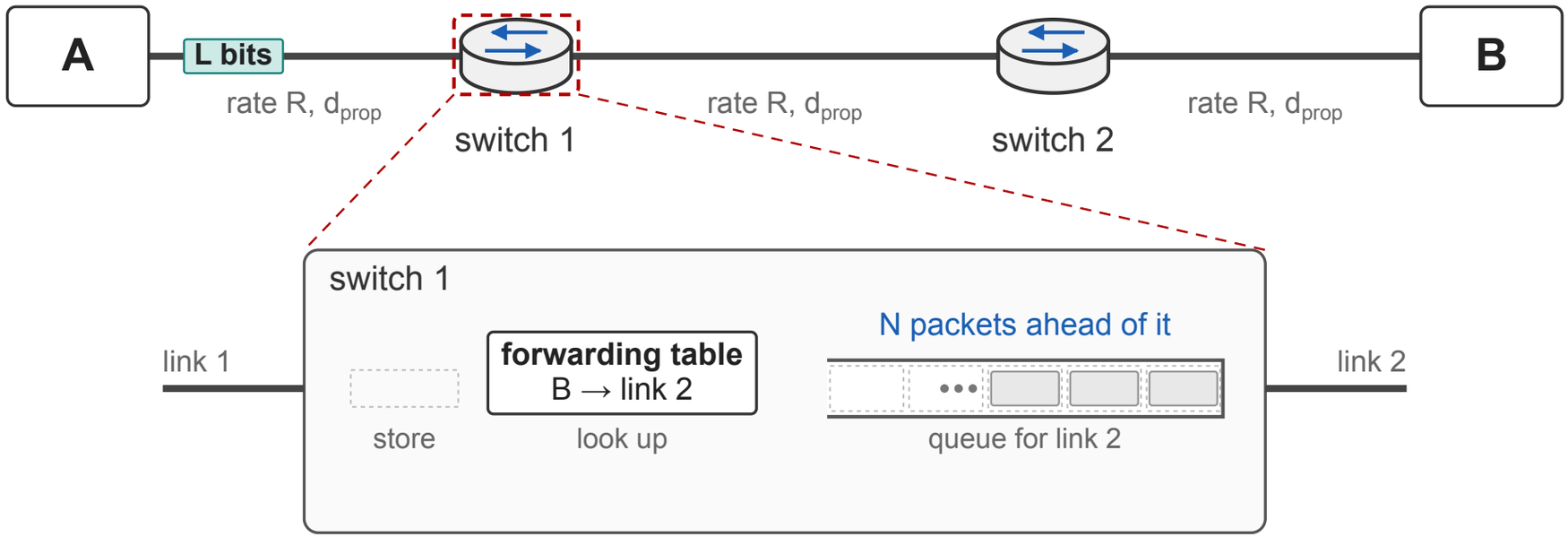
Revisiting delay



Time it takes for the packet to be fully received at B = $d_{proc,A}$ process at A

A prepares the packet, checksum and all: $d_{proc,A}$

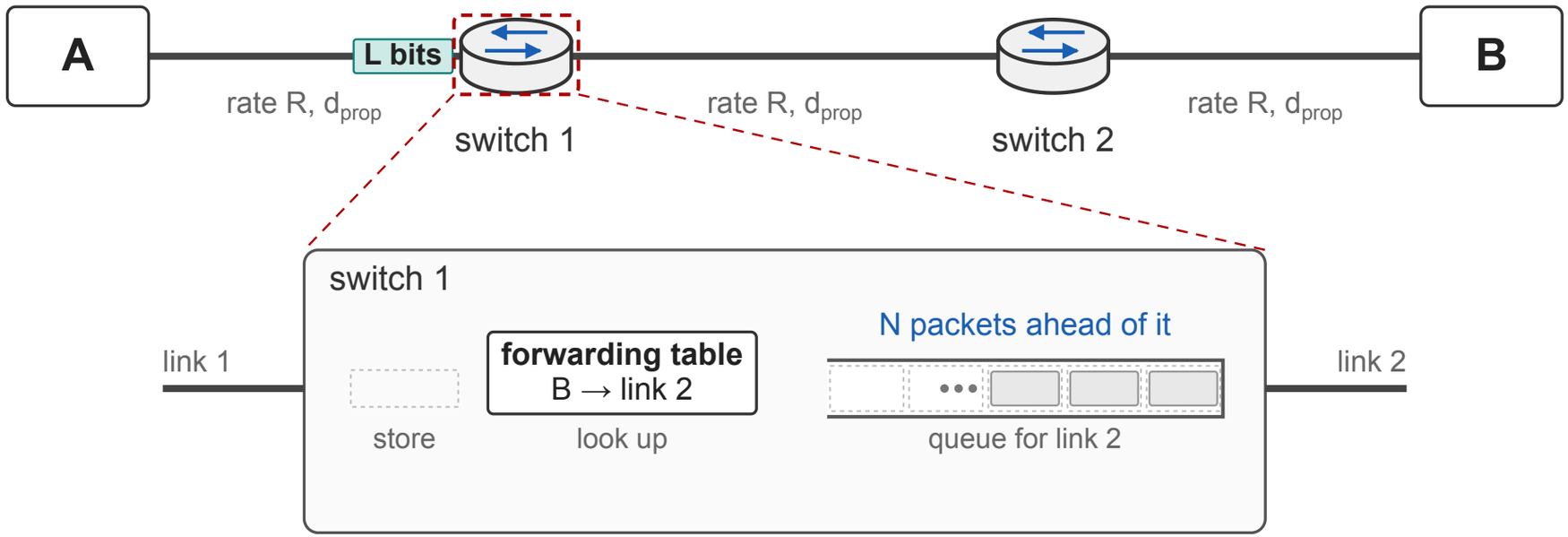
Revisiting delay



Time it takes for the packet to be fully received at B = $d_{proc,A}$ (process at A) + L / R (transmit)

A puts the packet on link 1, bit by bit: L / R

Revisiting delay



Time it takes for the packet to be fully received at B = $d_{proc,A}$ + L / R + d_{prop}

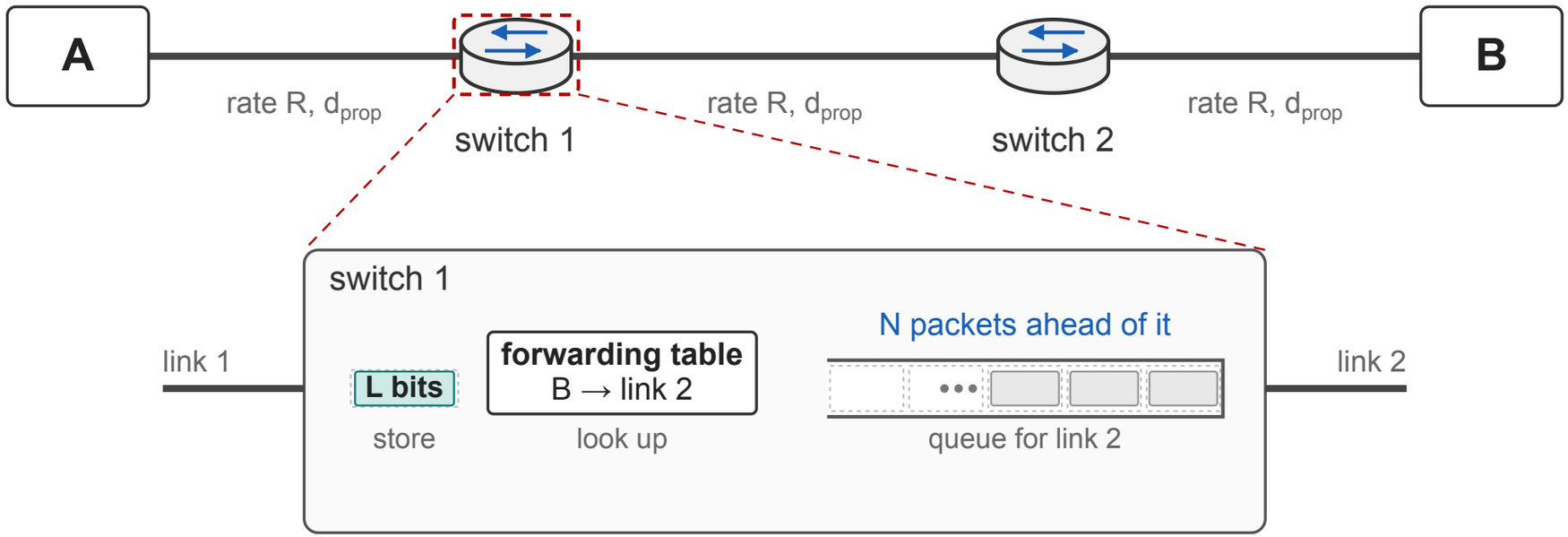
$d_{proc,A}$: process at A

L / R: transmit

d_{prop}: propagate

The bits travel down link 1: d_{prop}

Revisiting delay

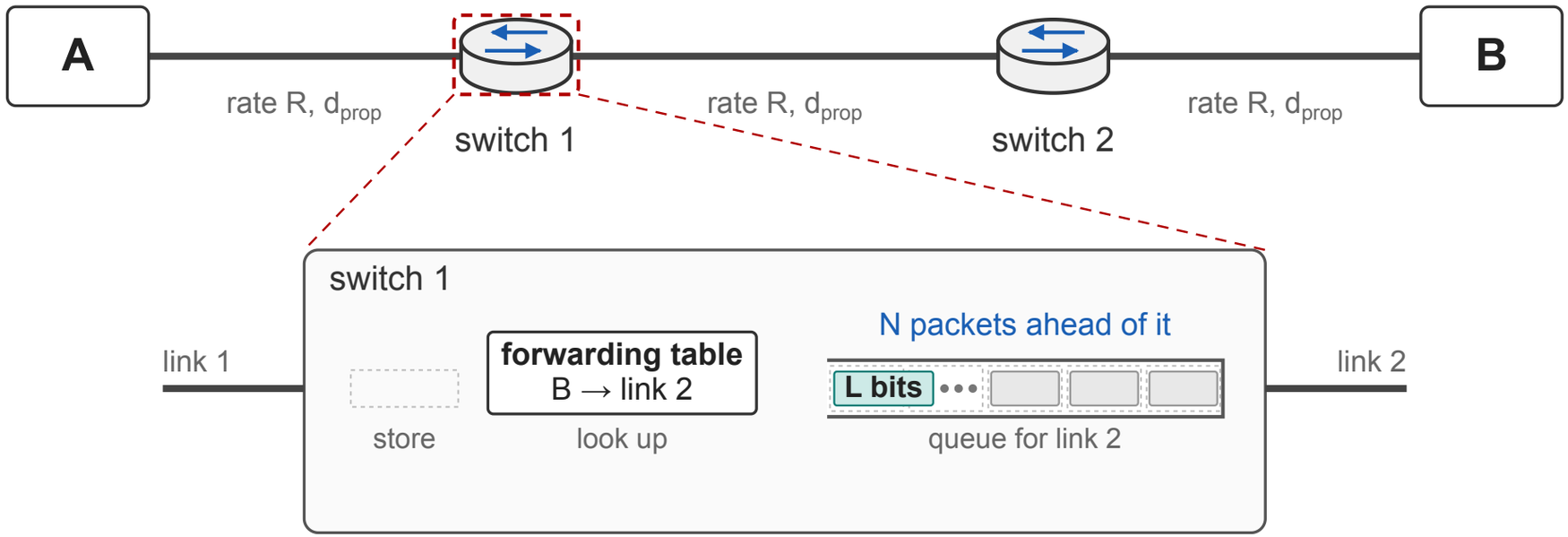


Time it takes for the packet to be fully received at B = $d_{proc,A}$ (process at A) + L / R (transmit) + d_{prop} (propagate)

+ d_{proc} (process)

Switch 1 waits for the whole packet, checks it, looks up B: processing, d_{proc}

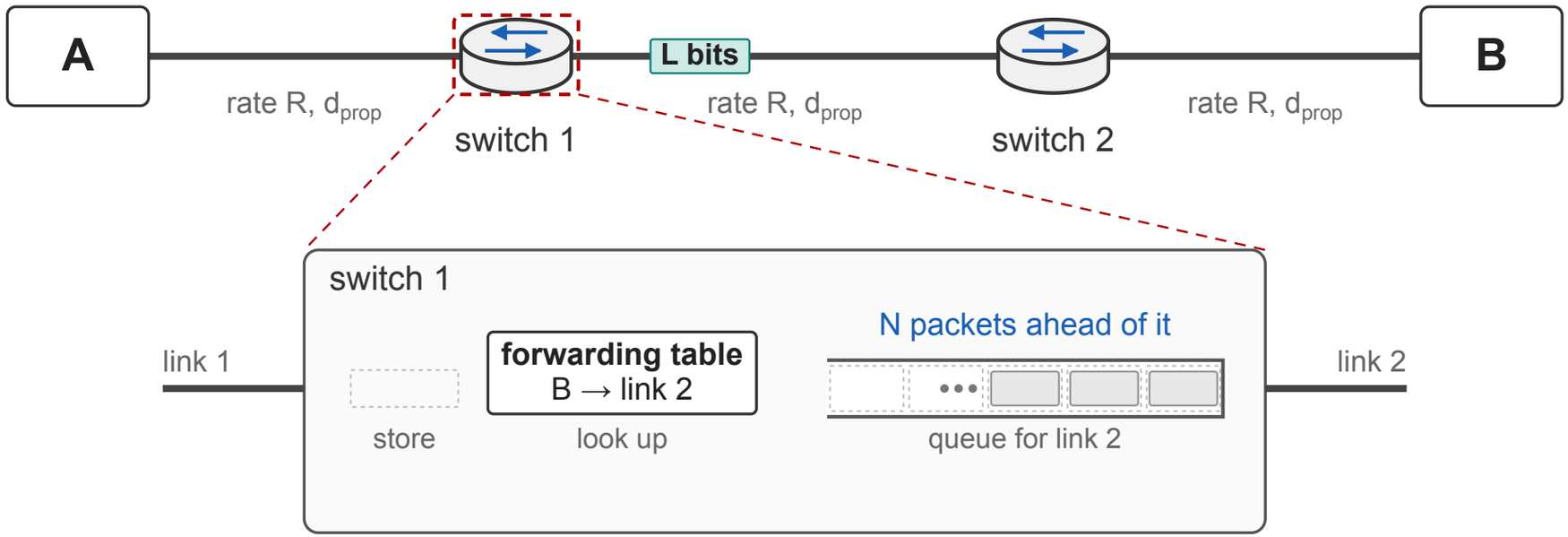
Revisiting delay



Time it takes for the packet to be fully received at B = $d_{proc,A}$ process at A + L / R transmit + d_{prop} propagate + d_{proc} process + **N · L / R** **queuing delay**

N packets of, say, L bits are ahead of it in the queue for link 2, each taking L / R to transmit: it waits N · L / R, the queuing delay

Revisiting delay



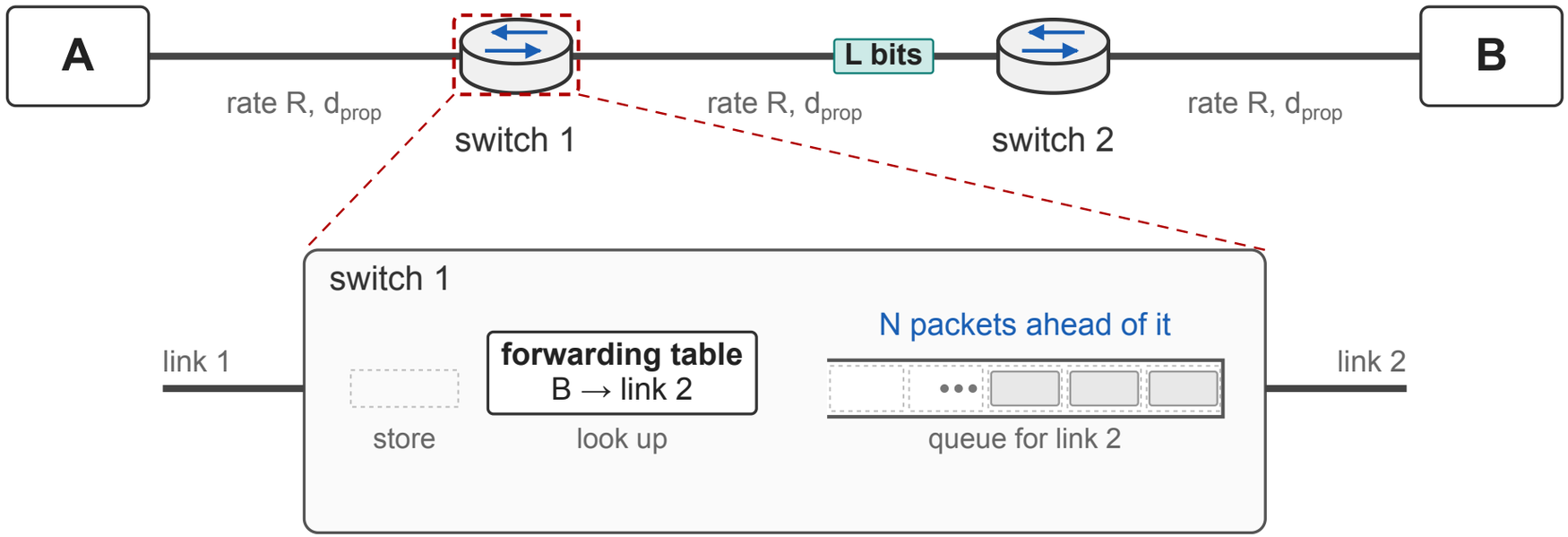
Time it takes for the packet to be fully received at B =

$d_{proc,A}$ process at A + L / R transmit + d_{prop} propagate

+ d_{proc} process + N · L / R queuing delay + L / R transmit

Its turn: switch 1 puts it on link 2: L / R

Revisiting delay



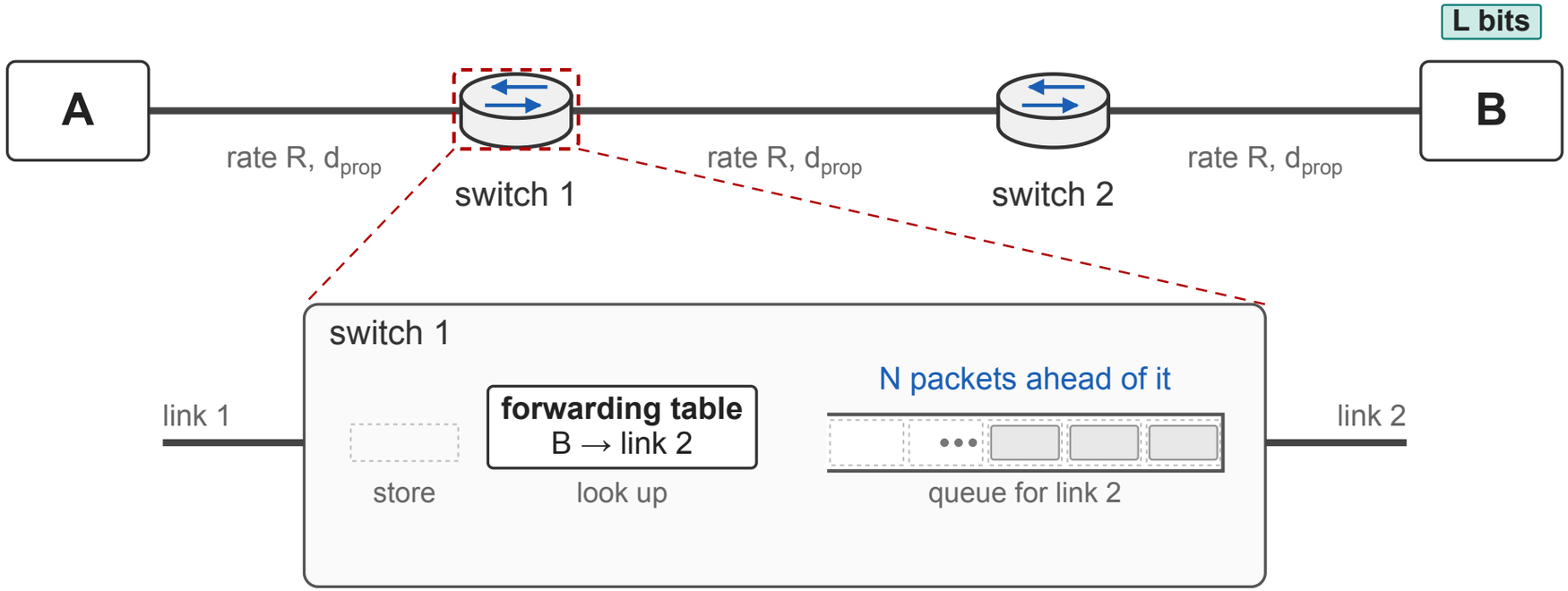
Time it takes for the packet to be fully received at B =

$d_{proc,A}$ process at A + L / R transmit + d_{prop} propagate

+ d_{proc} process + N · L / R queuing delay + L / R transmit + d_{prop} propagate

Down link 2: d_{prop}

Revisiting delay

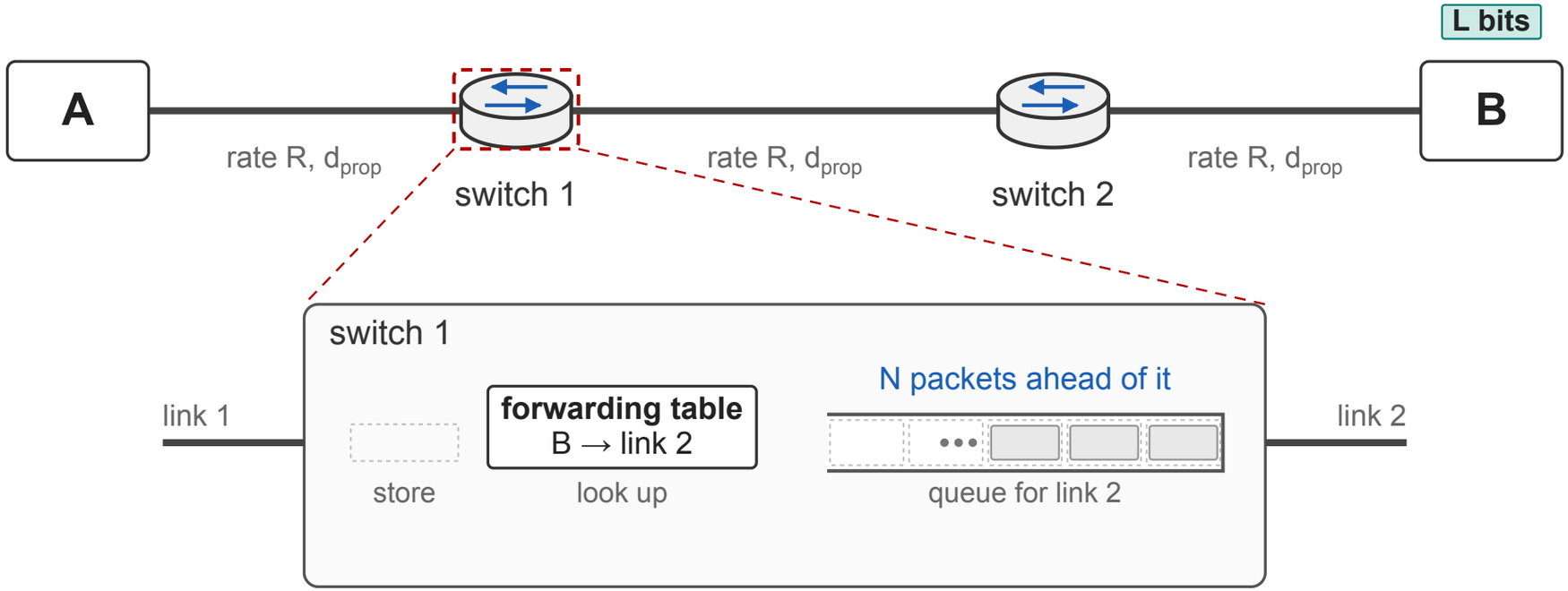


Time it takes for the packet to be fully received at B = $d_{proc,A}$ (process at A) + L / R (transmit) + d_{prop} (propagate)

$+ d_{proc}$ (process) $+ N \cdot L / R$ (queuing delay) $+ L / R$ (transmit) $+ d_{prop}$ (propagate) $+ d_{proc} + N' \cdot L / R + L / R + d_{prop}$ (switch 2 and link 3)

The same again at switch 2 and on link 3

Revisiting delay



Time it takes for the packet to be fully received at B = $d_{proc,A}$ (process at A) + L / R (transmit) + d_{prop} (propagate)

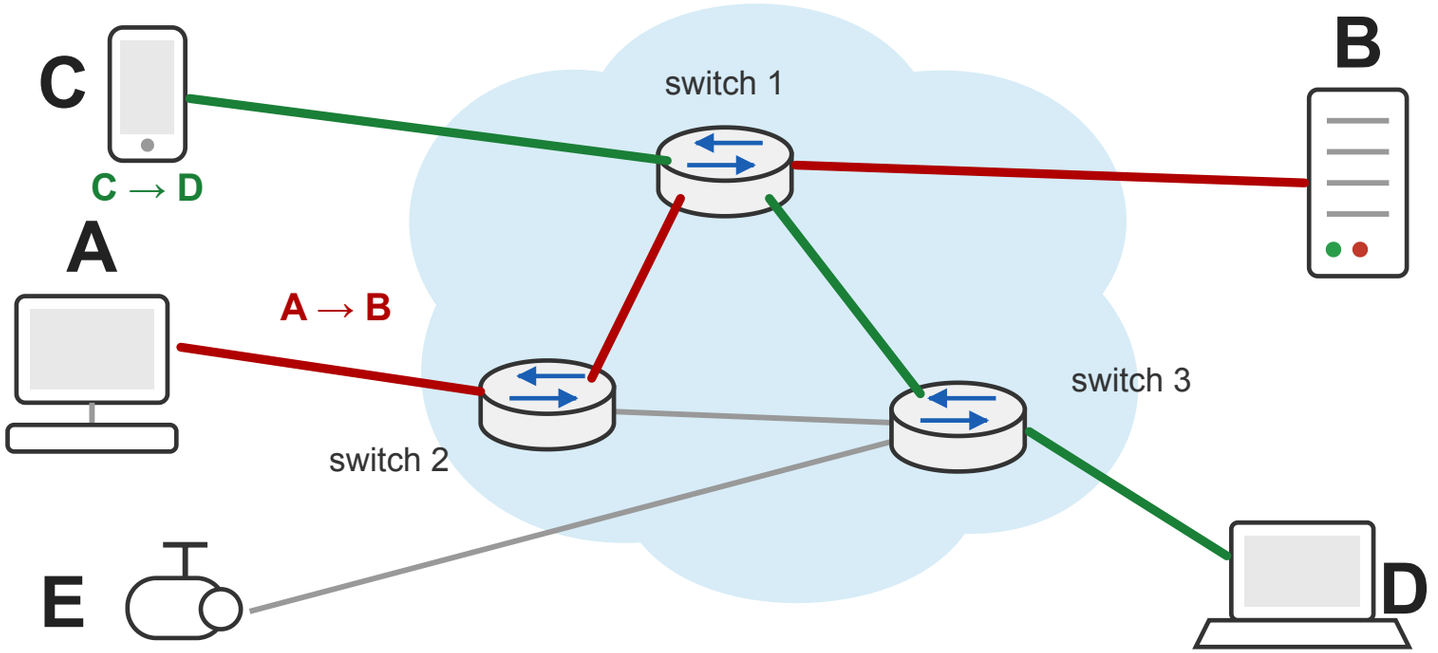
$+ d_{proc}$ (process) $+ N \cdot L / R$ (queuing delay) $+ L / R$ (transmit) $+ d_{prop}$ (propagate) $+ d_{proc} + N' \cdot L / R + L / R + d_{prop}$ (switch 2 and link 3)

On every device + outgoing link on the path: processing + queuing + transmission + propagation.
To get the end-to-end delay, sum this up across the path.

Today, we discussed …

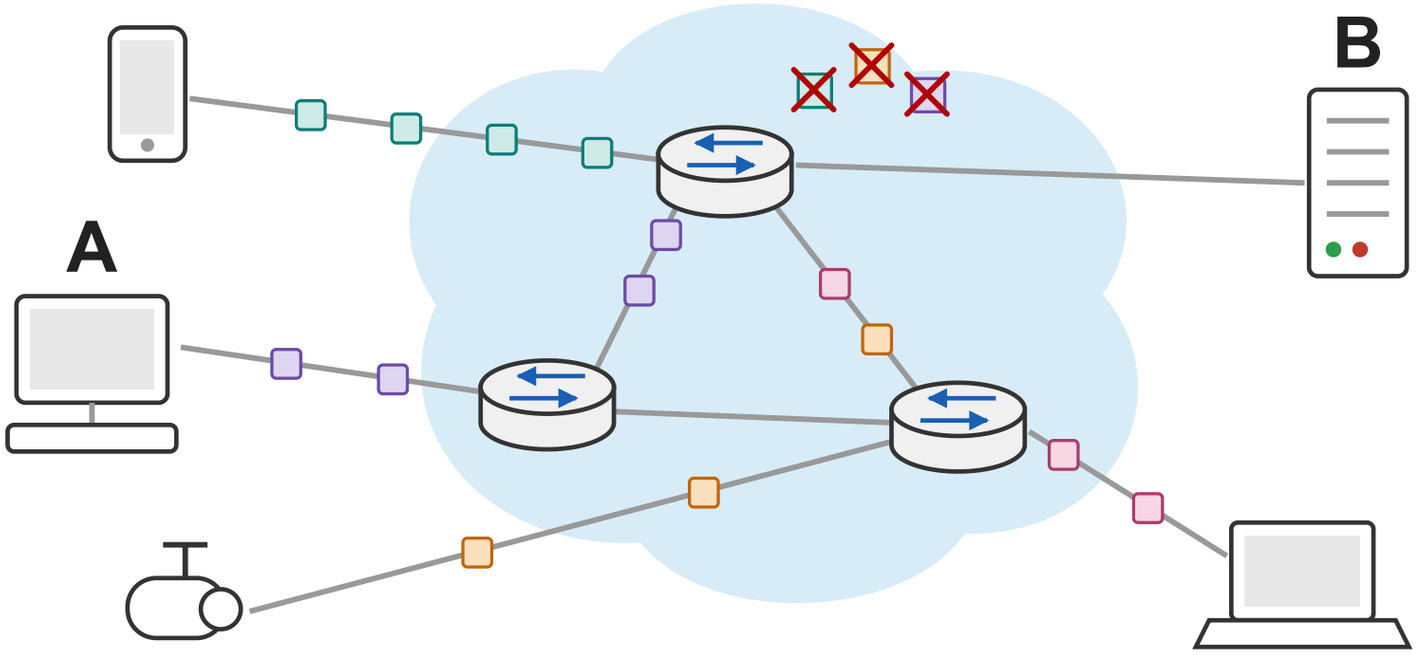
- Networks are a set of switches and links:
  - connected in a graph
  - shared among many endpoints
  - handle concurrent data transfers
- Packet switching vs. circuit switching
- Inside of a switch in a packet-switched network
- Computing end-to-end delay in the presence of such network switches

Problem 1: Finding paths



- The forwarding table in each switch should be populated such that all endpoints are reachable from each other, even under link and switch failures

Problem 2: Congestion control



- In a packet switched network, how can we ensure endpoints can utilize available network capacity without creating congestion?