Lecture 5:

The Transport Layer

Katerina Argyraki, EPFL

application web BitTorrent DNS

transport TCP UDP

network IP

link Ethernet

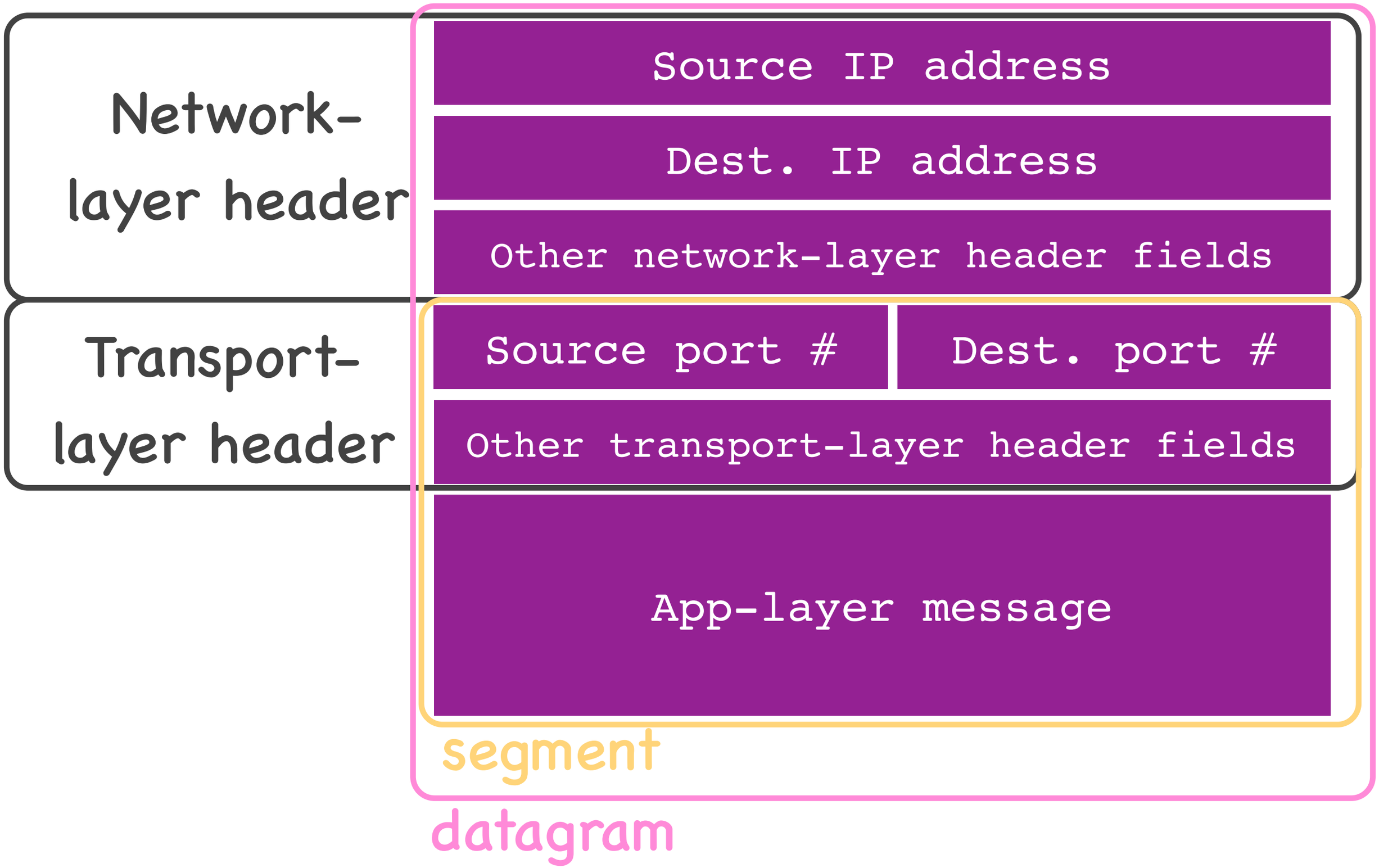
physical

Outline

- Process-to-process communication
  - UDP
  - TCP
- Reliable data delivery
  - Imaginary protocol
  - UDP
  - TCP

Outline

- Process-to-process communication
  - UDP
  - TCP
- Reliable data delivery
  - Imaginary protocol
  - UDP
  - TCP


Network-layer header
Source IP address
Dest. IP address
Other network-layer header fields
Transport-layer header
Source port #
Dest. port #
Other transport-layer header fields
App-layer message
segment
datagram

Process S

```
socket = new socket (UDP type)

socket.bind (IP address: 1.1.1.1, port: 1000)

socket.sendto (message, dest. IP address: 5.5.5.5,
                        dest. port: 5000)

socket.close ( )
```

UDP socket

```
for process S

IP address: 1.1.1.1
port: 1000
```

Process R

```
socket = new socket (UDP type)
socket.bind (IP address: 5.5.5.5, port: 5000)
message = socket.recvfrom (100 bytes)
socket.close ( )
```

UDP socket

```
for process R

IP address: 5.5.5.5
port: 5000
```

Source IP address: 1.1.1.1	
Dest. IP address: 5.5.5.5	
Source port: 1000	Dest. port: 5000
message	

UDP sockets

- Each UDP socket has a unique (IP address, port #) tuple

- A process may use the same UDP socket to communicate with many remote processes

Process S

```
socket = new socket (TCP type)

socket.bind (IP address: 1.1.1.1, port: 1000)

socket.connect (rem. IP address: 5.5.5.5, rem. port: 5000)

socket.send (message)

socket.close ( )
```

TCP socket

```
for process S

IP address:1.1.1.1
port:1000

rem. IP address:5.5.5.5
rem. port:5000
```

Process R

```
socket = new socket (TCP type)
socket.bind (IP address: 5.5.5.5, port: 5000)
socket.listen (for N connections)
connSocket = socket.accept ( )
```

TCP socket

```
for process R

IP address:5.5.5.5
port:5000

listening for N conn.
```

Source IP address: 1.1.1.1	
Dest. IP address: 5.5.5.5	
Source port: 1000	Dest. port: 5000
connection-setup request	

Process R

```
socket = new socket (TCP type)
socket.bind (IP address: 5.5.5.5, port: 5000)
socket.listen (for N connections)
connSocket = socket.accept ( )
message = connSocket.recv (100 bytes)
connSocket.close ( )
```

TCP socket

```
for process R

IP address:5.5.5.5
port:5000

rem. IP address:1.1.1.1
rem. port:1000
```

Source IP address: 1.1.1.1	
Dest. IP address: 5.5.5.5	
Source port: 1000	Dest. port: 5000
message	

TCP sockets

- Listening & connection sockets
- Each connection socket has a unique (local IP, local port, remote IP, remote port) tuple
- A process must use a different TCP connection socket per remote process

Process-to-process communication

- Multiplexing
  - upon receiving a new message from a process, create new packets
  - identify the correct IP addresses and ports
- Demultiplexing
  - many processes running in app layer
  - upon receiving a new packet from the network, identify the correct dest. process

Outline

- Process-to-process communication
  - UDP
  - TCP
- Reliable data delivery
  - Imaginary protocol
  - UDP
  - TCP

Alice's computer

Bob's computer

app

transport

network

network

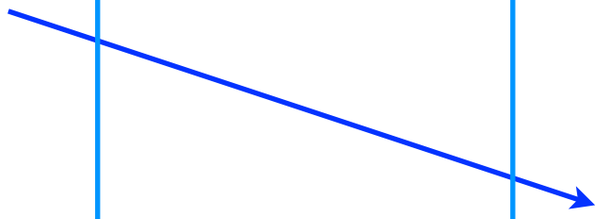
transport

app

Alice's computer

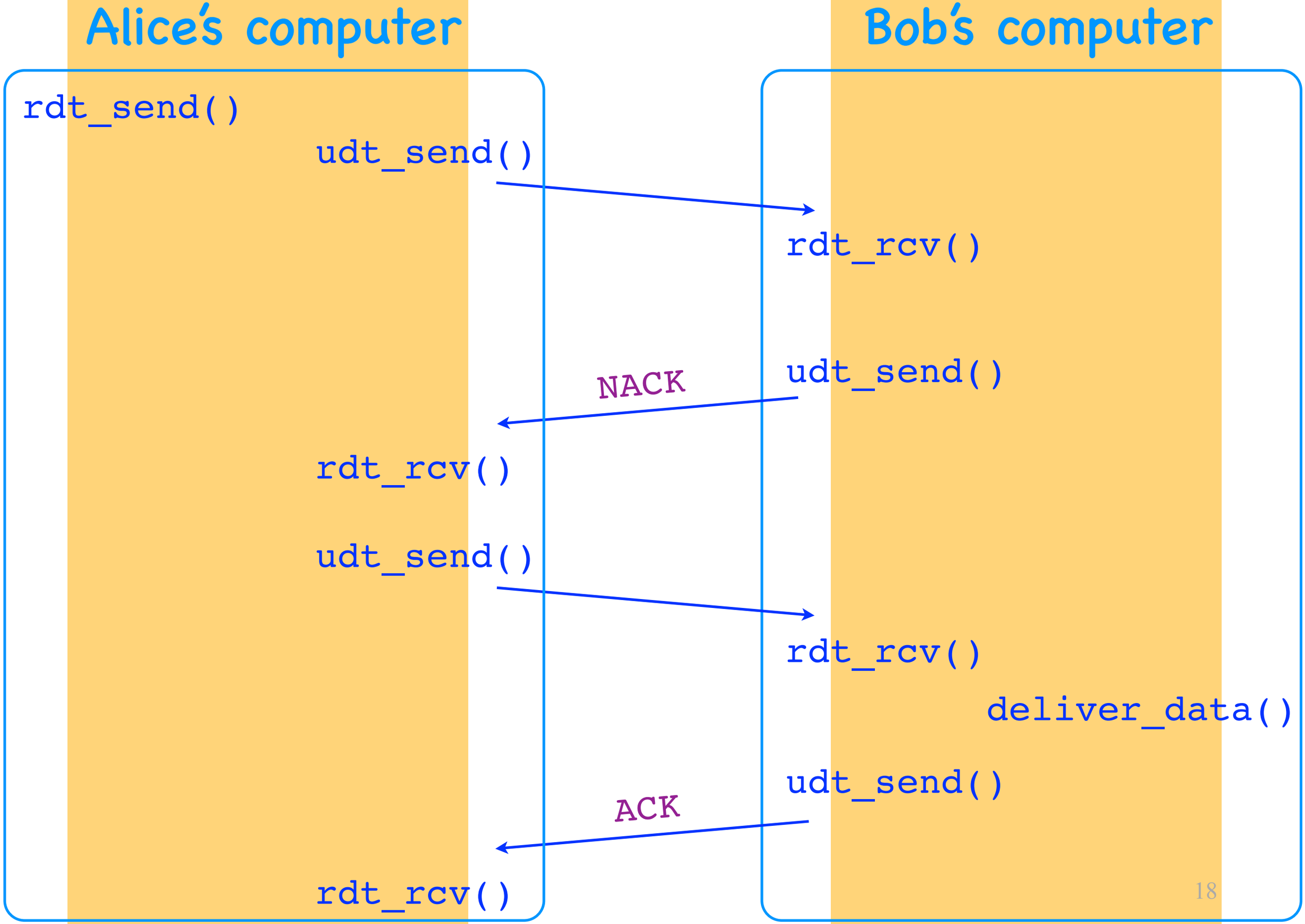
Bob's computer

```
rdt_send()

                udt_send()

                                    rdt_rcv()

                                                deliver_data()
```



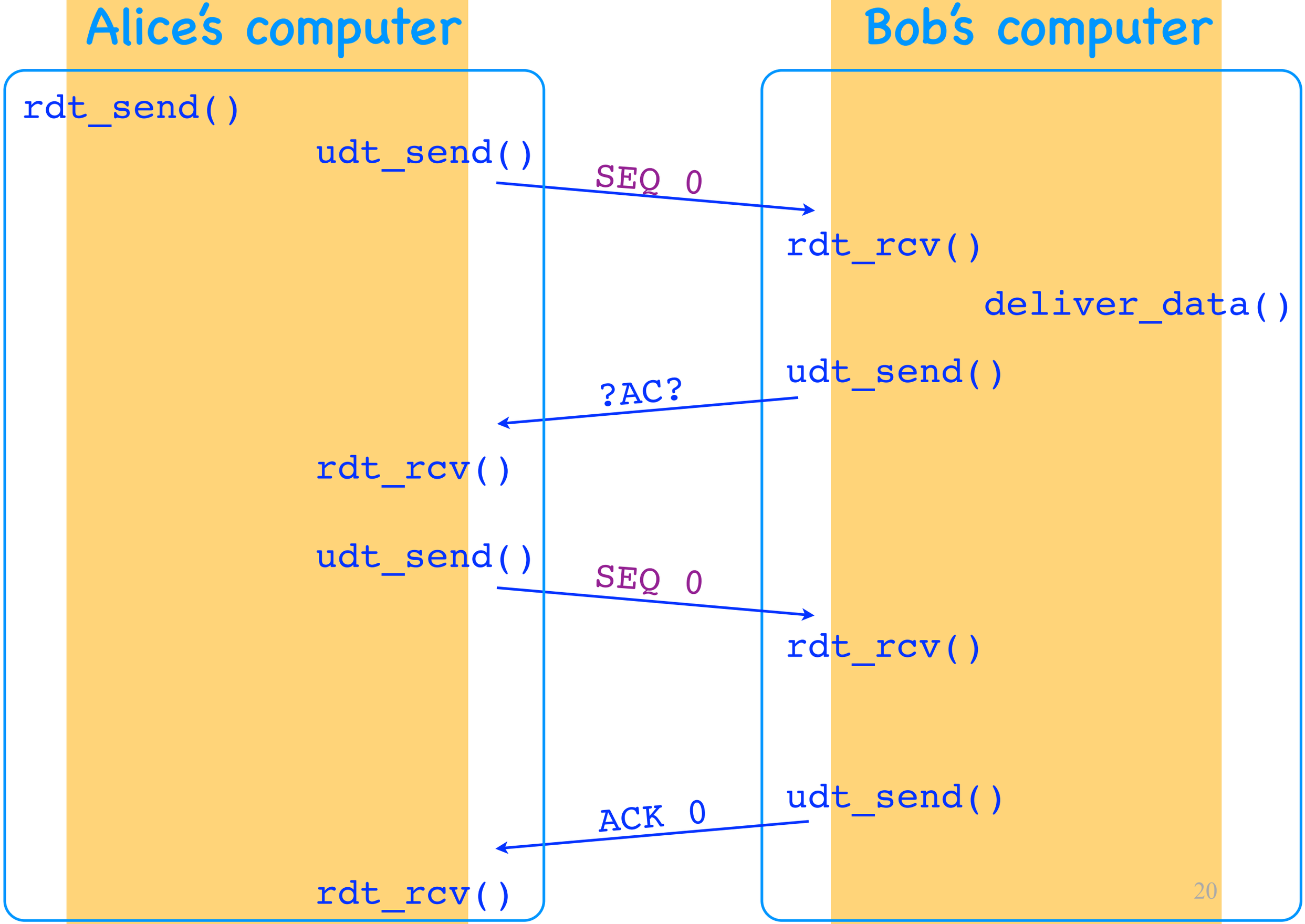
Checksum

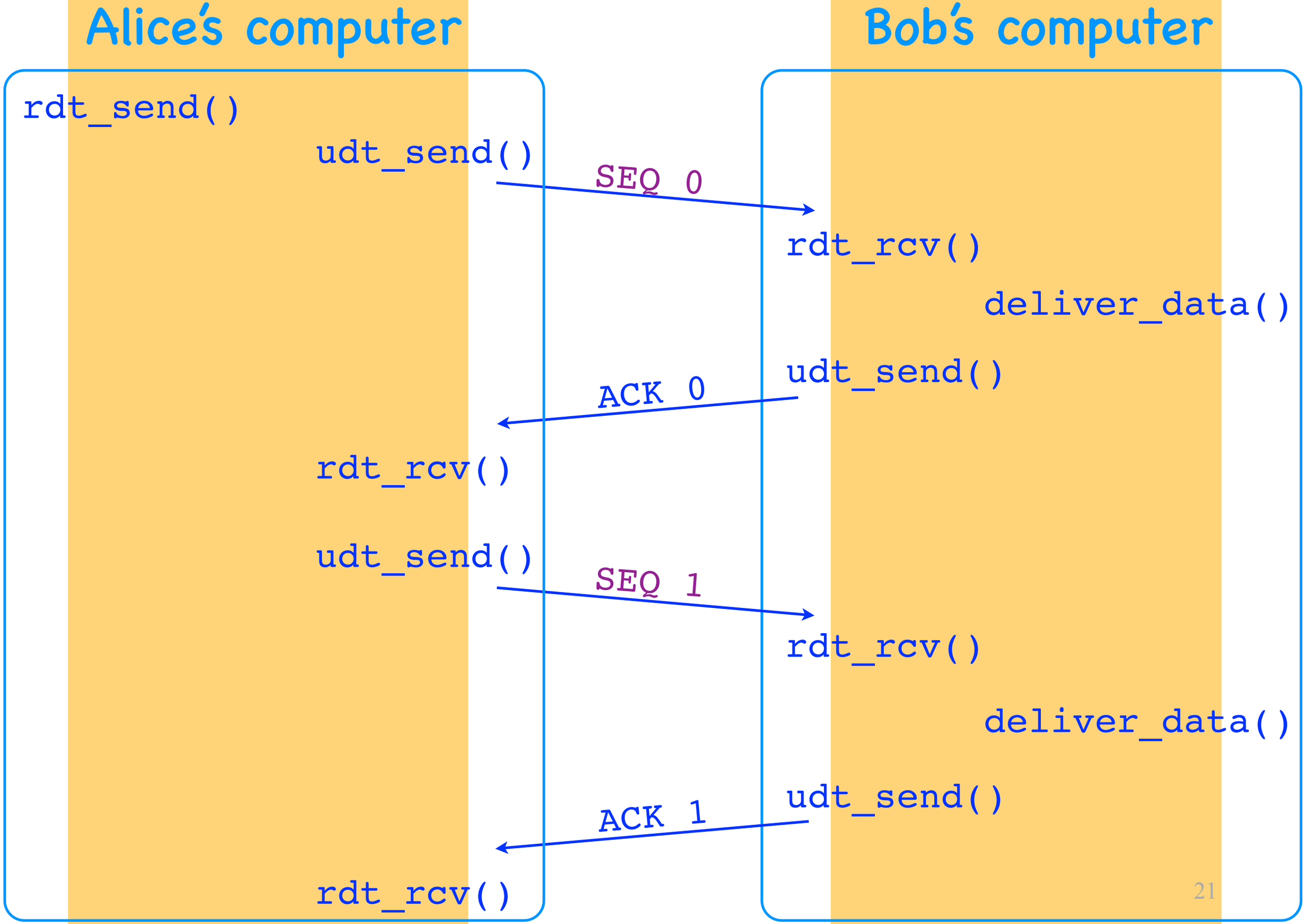
- Redundant information
  - e.g., the binary sum of all data bytes
- Sender adds checksum C to each segment
  - transport-layer header field
- Receiver uses it to detect data corruption
  - receiver recomputes checksum C'
  - if C'!= C, segment was corrupted


Alice's computer
Bob's computer
rdt_send()
udt_send()
rdt_rcv()
NACK
udt_send()
rdt_rcv()
udt_send()
rdt_rcv()
deliver_data()
udt_send()
ACK
rdt_rcv()

Acknowledgment

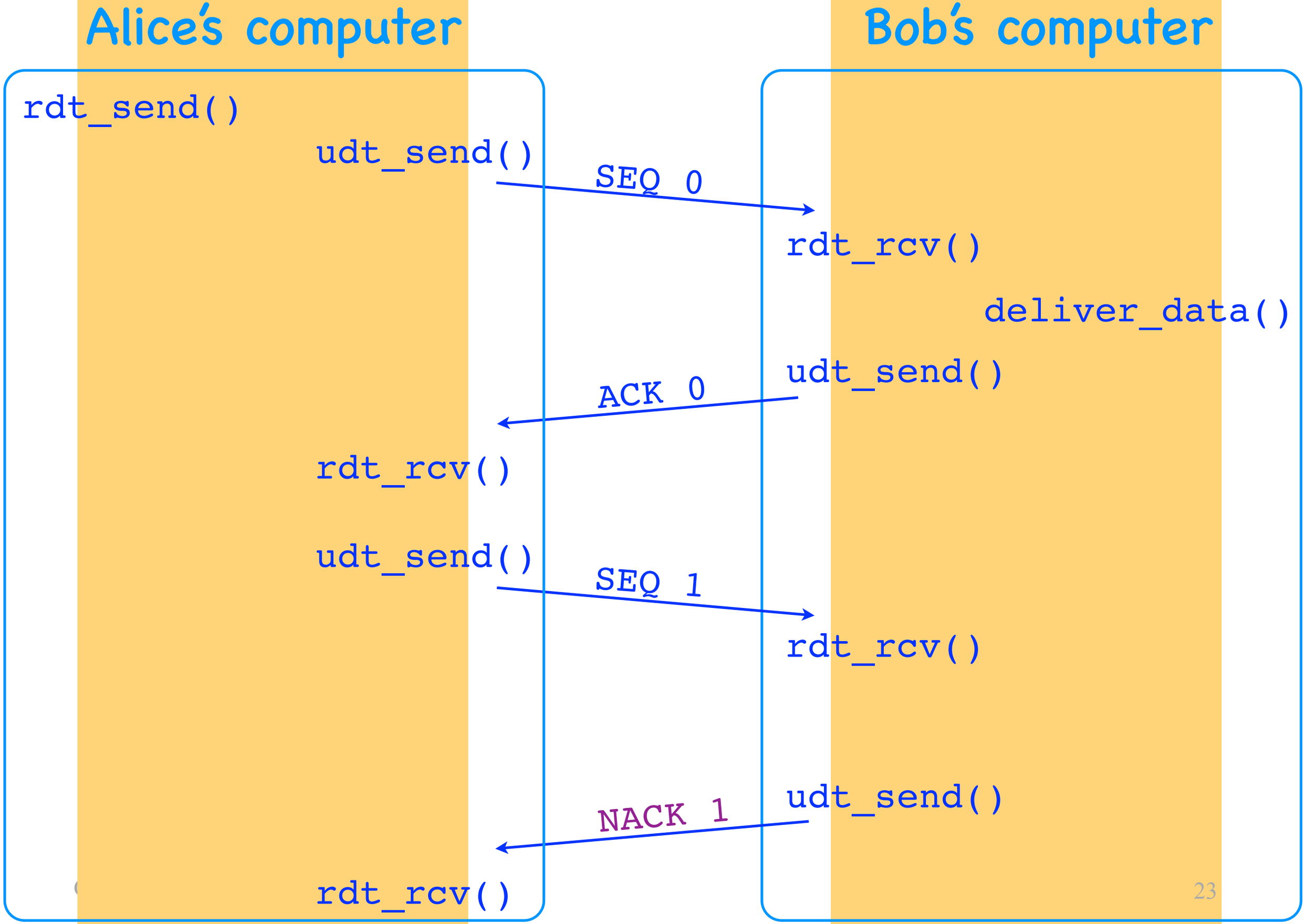
- Feedback from receiver to sender
- Receiver adds ACK to each segment
  - transport-layer header field
- Sender uses it to detect and overcome data corruption
  - if sender gets negative ACK, it retransmits the data

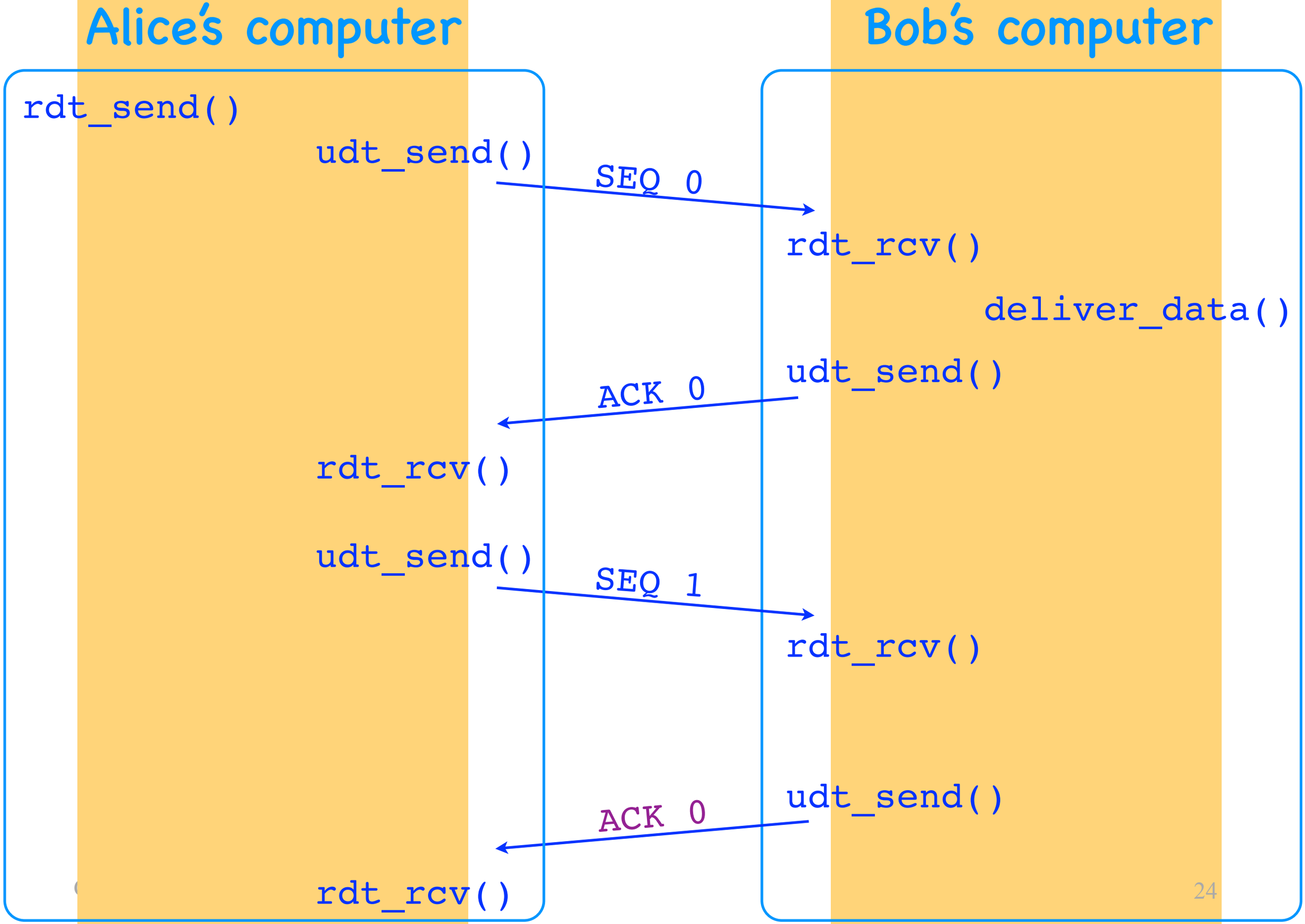

Alice's computer
Bob's computer
rdt_send()
udt_send()
SEQ 0
rdt_rcv()
deliver_data()
udt_send()
?AC?
rdt_rcv()
udt_send()
SEQ 0
rdt_rcv()
udt_send()
ACK 0
rdt_rcv()


Alice's computer
Bob's computer
rdt_send()
udt_send()
SEQ 0
rdt_rcv()
deliver_data()
udt_send()
ACK 0
rdt_rcv()
udt_send()
SEQ 1
rdt_rcv()
deliver_data()
udt_send()
ACK 1
rdt_rcv()

Sequence number

- An identifier for data
- Sender adds SEQ to each segment
  - transport-layer header field
- Receiver uses it to disambiguate data


Alice's computer
Bob's computer
rdt_send()
udt_send()
SEQ 0
rdt_rcv()
deliver_data()
udt_send()
ACK 0
rdt_rcv()
udt_send()
SEQ 1
rdt_rcv()
udt_send()
NACK 1
rdt_rcv()


Alice's computer
Bob's computer
rdt_send()
udt_send()
SEQ 0
rdt_rcv()
deliver_data()
udt_send()
ACK 0
rdt_rcv()
udt_send()
SEQ 1
rdt_rcv()
udt_send()
ACK 0
rdt_rcv()

Alice's computer

Bob's computer

rdt_send()

udt_send()

SEQ 0

timeout

udt_send()

SEQ 0

rdt_rcv()

deliver_data()

udt_send()

ACK 0

rdt_rcv()

Alice's computer

Bob's computer

rdt_send()

udt_send()

SEQ 0

rdt_rcv()

deliver_data()

udt_send()

ACK 0

timeout

udt_send()

SEQ 0

rdt_rcv()

deliver_data()

udt_send()

ACK 0

rdt_rcv()

Alice's computer

Bob's computer

rdt_send()

udt_send()

SEQ 0

rdt_rcv()

deliver_data()

udt_send()

ACK 0

timeout

udt_send()

SEQ 0

rdt_rcv()

rdt_rcv()

udt_send()

udt_send()

SEQ 1

rdt_rcv()

rdt_rcv()

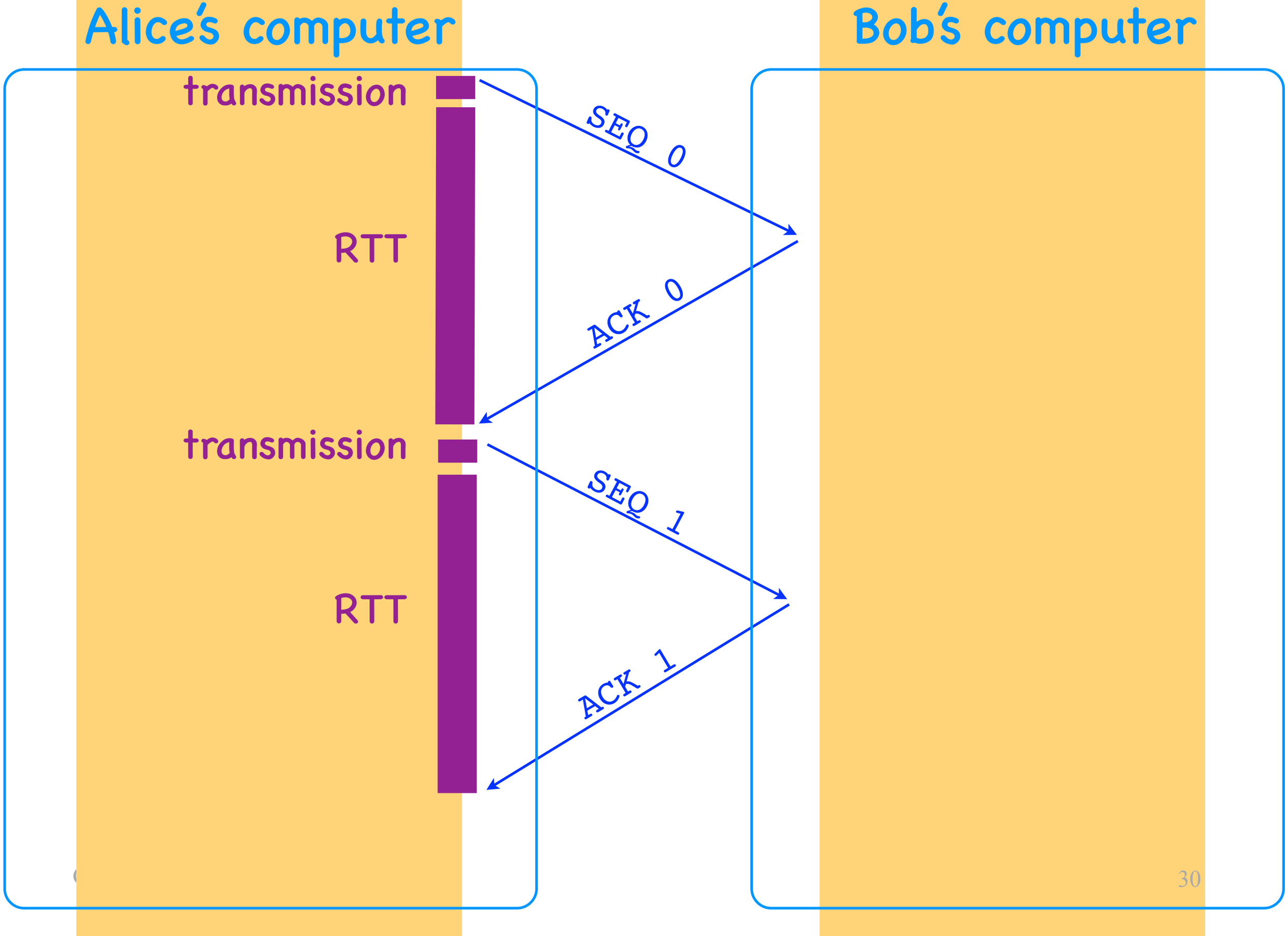
deliver_data()

Timeout

- No arrival of an expected ACK
  - a segment was lost or delayed
  - the ACK for a segment was lost or delayed
- Sender uses it to overcome data loss
  - if the sender times out, it retransmits

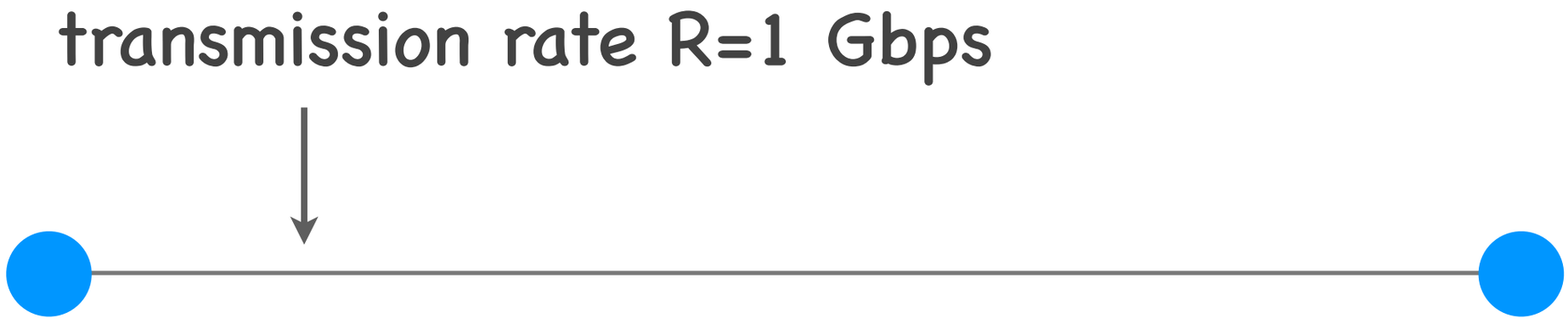
Basic elements

- Checksums
  - detect data corruption
- ACKs + retransmissions + SEQs
  - overcome data corruption
- Timeouts + ACKs + retransmissions + SEQs
  - overcome data loss


Alice's computer
Bob's computer
transmission
RTT
SEQ 0
ACK 0
transmission
RTT
SEQ 1
ACK 1

Alice's computer

Bob's computer



packet size L = 1000 bytes

transmission delay = L/R = 8 usec

propagation delay = 15 msec

Alice's computer

Bob's computer

0.008 msec

30 msec

SEQ 0

ACK 0

Busy for $\frac{0.008}{30.008}$

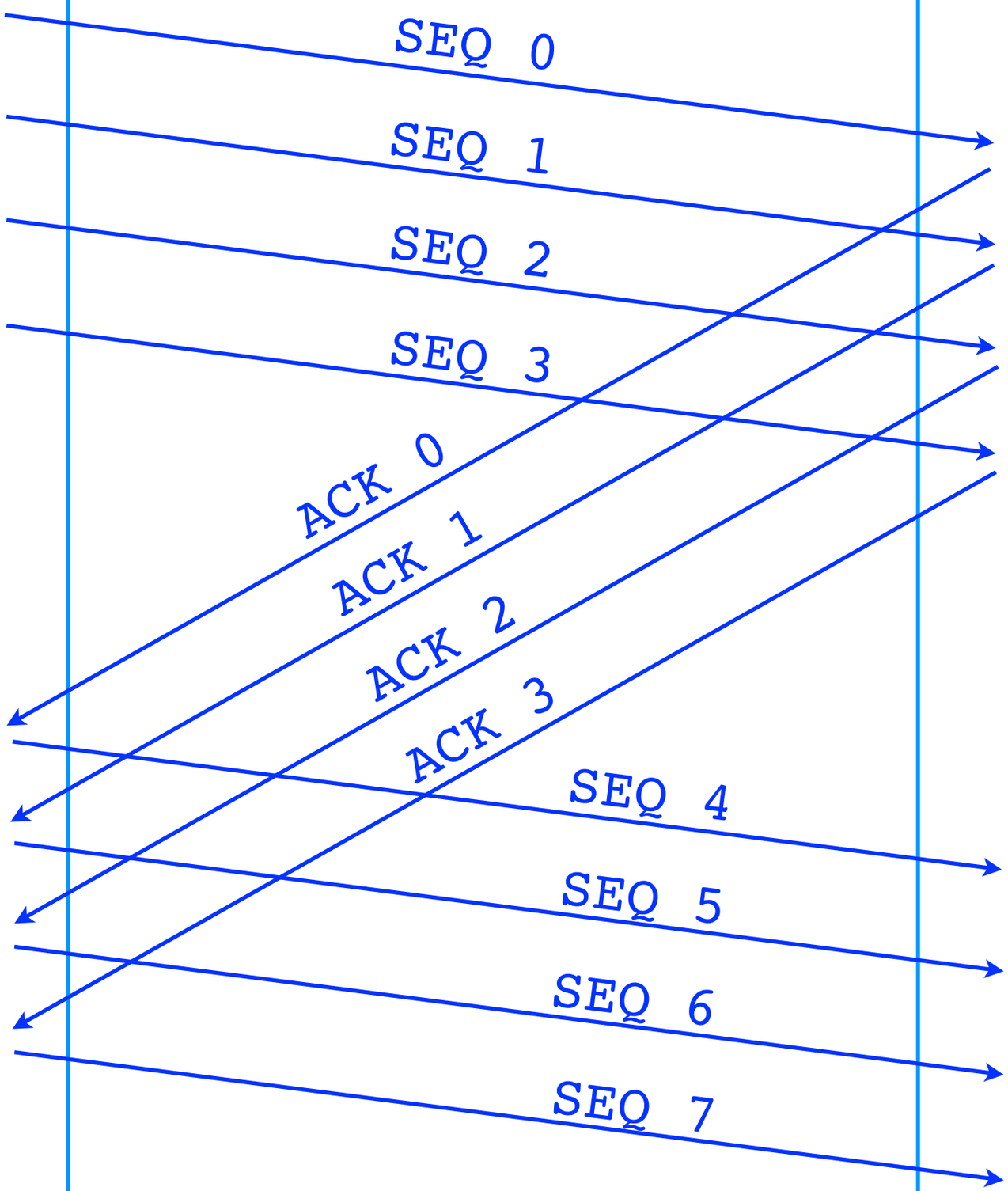
= 0.00027

sender/channel utilization

Alice's computer

Bob's computer

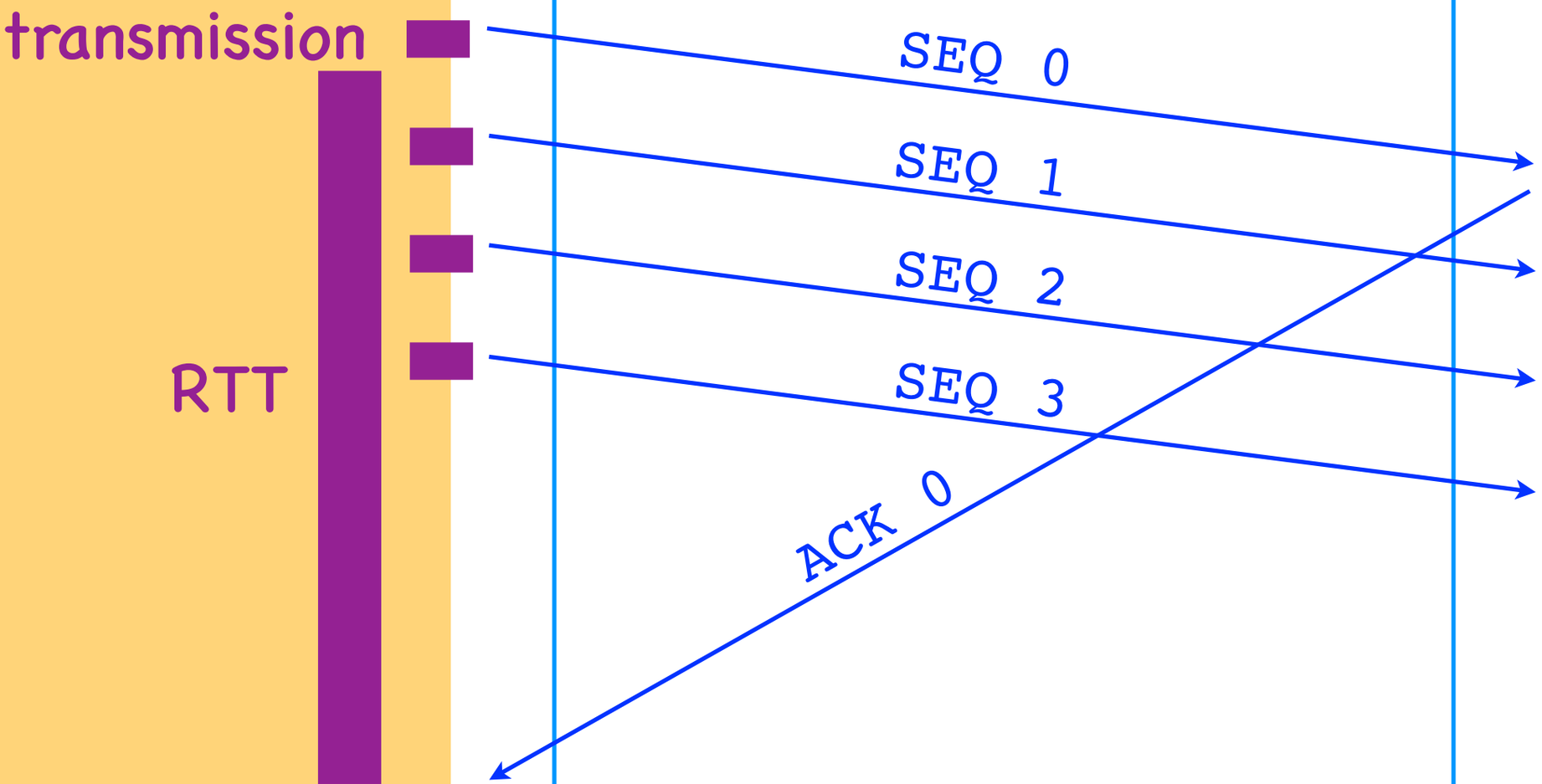
0
1
2
3
4
5
6
7



window size N = 4

Alice's computer

Bob's computer



Busy for

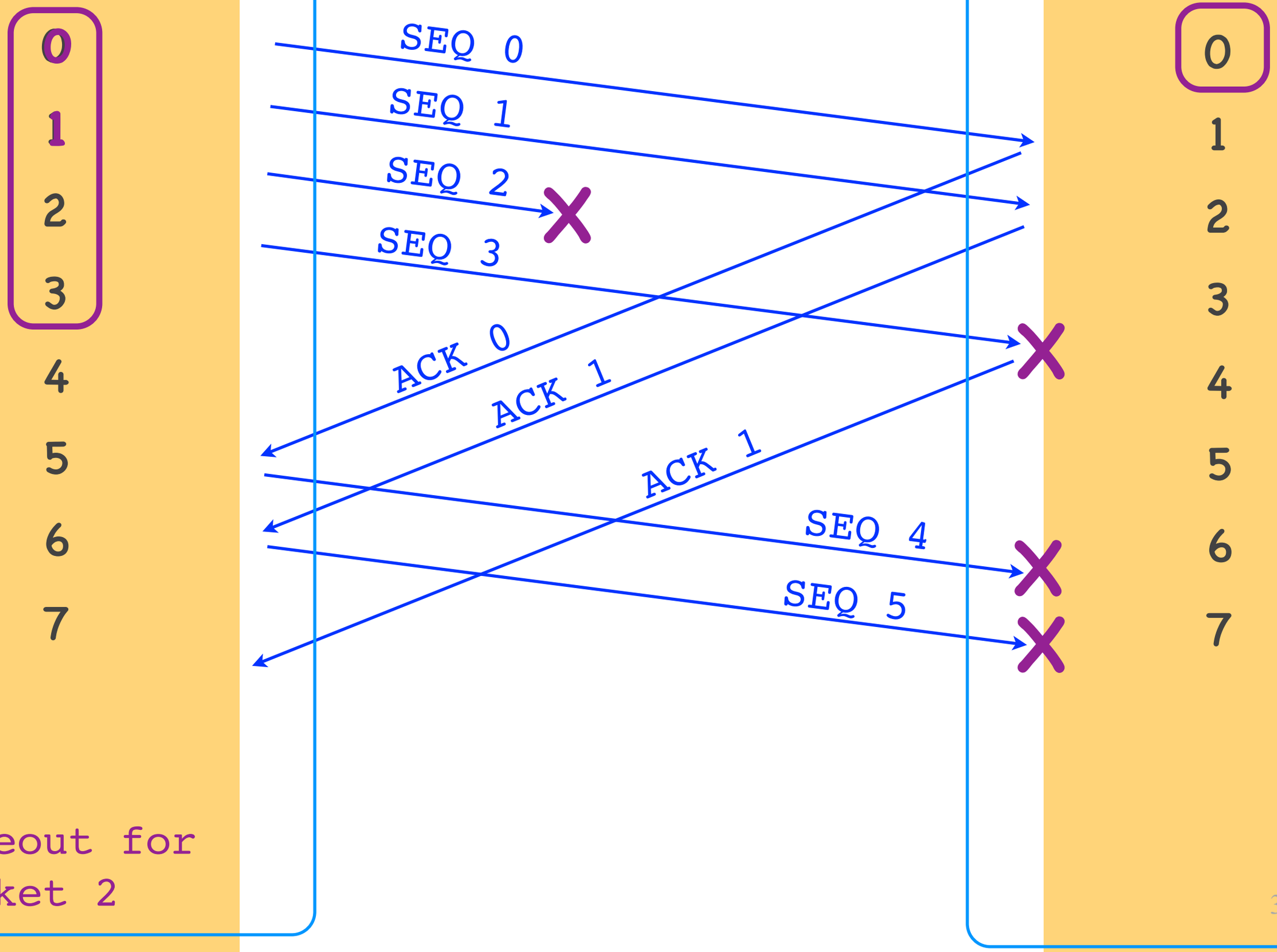
$$\frac{\text{trans. } N}{\text{trans. + RTT}}$$

Sender utilization

- Stop and wait: poor sender utilization
  - the sender does nothing while waiting for the receiver's ACK or the timeout
- Pipelining: better utilization
  - the sender sends up to N un-ACKed segments
  - N = sliding window size

Alice's computer

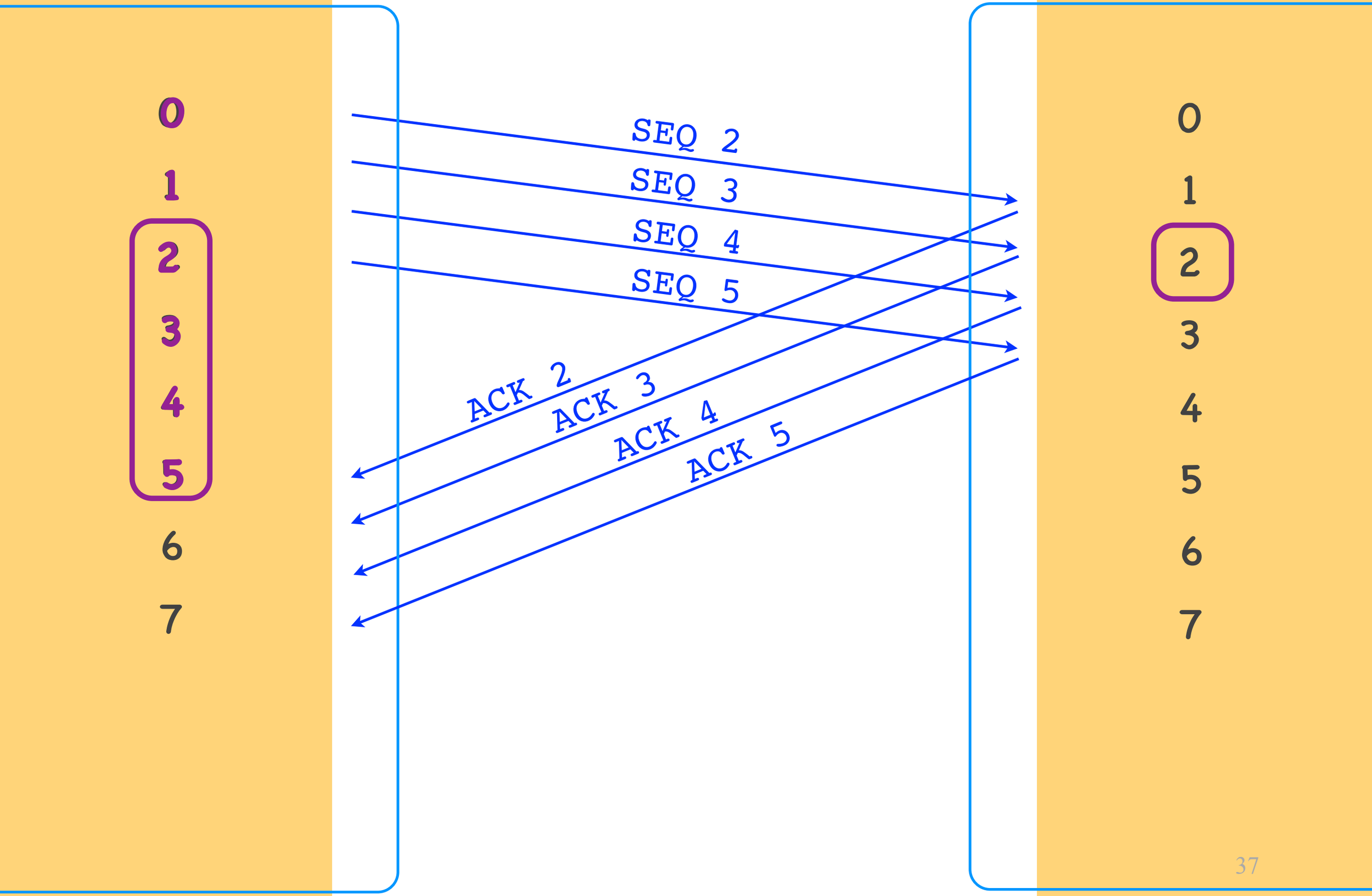
Bob's computer



timeout for packet 2

Alice's computer

Bob's computer

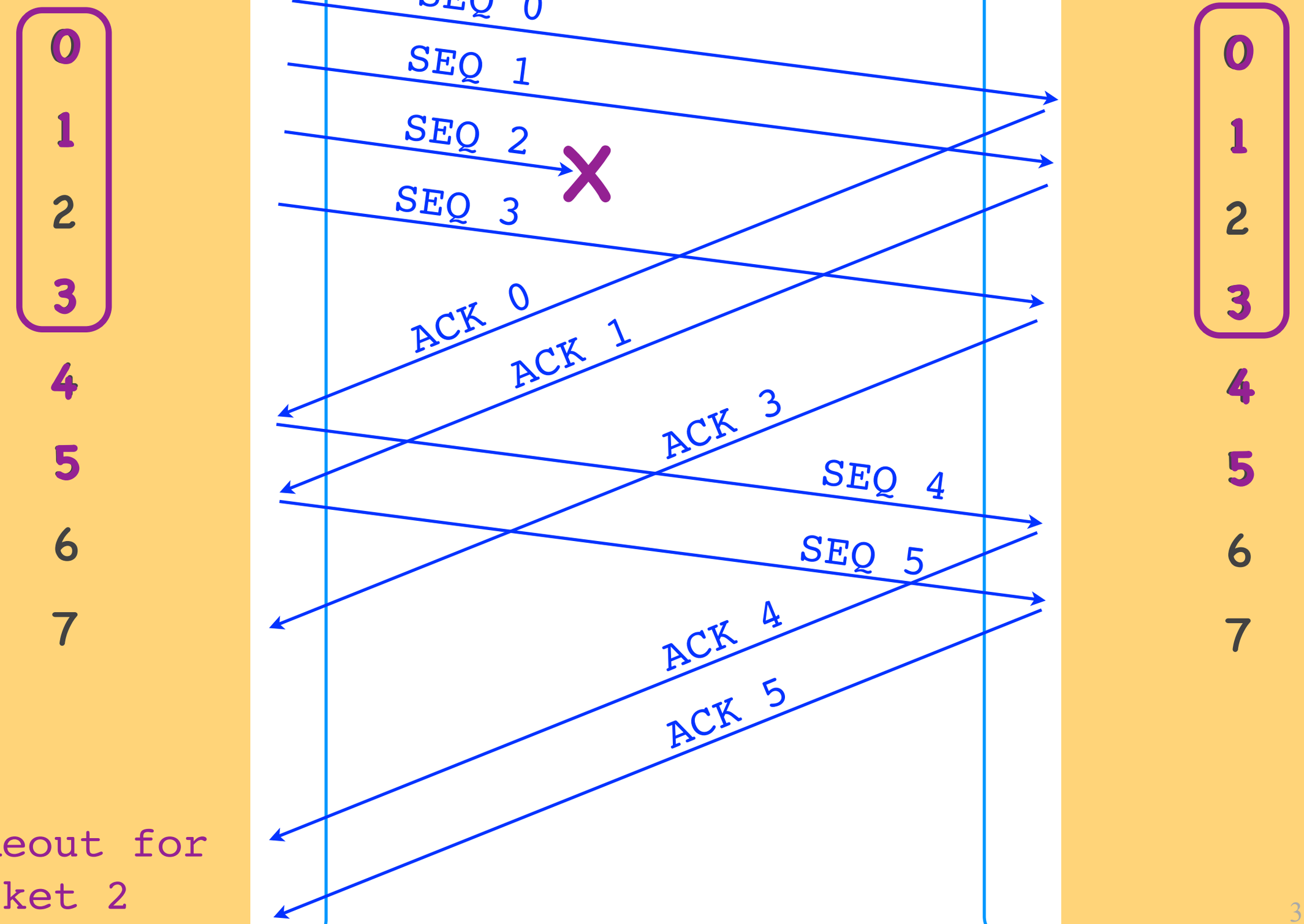


Go-back-N

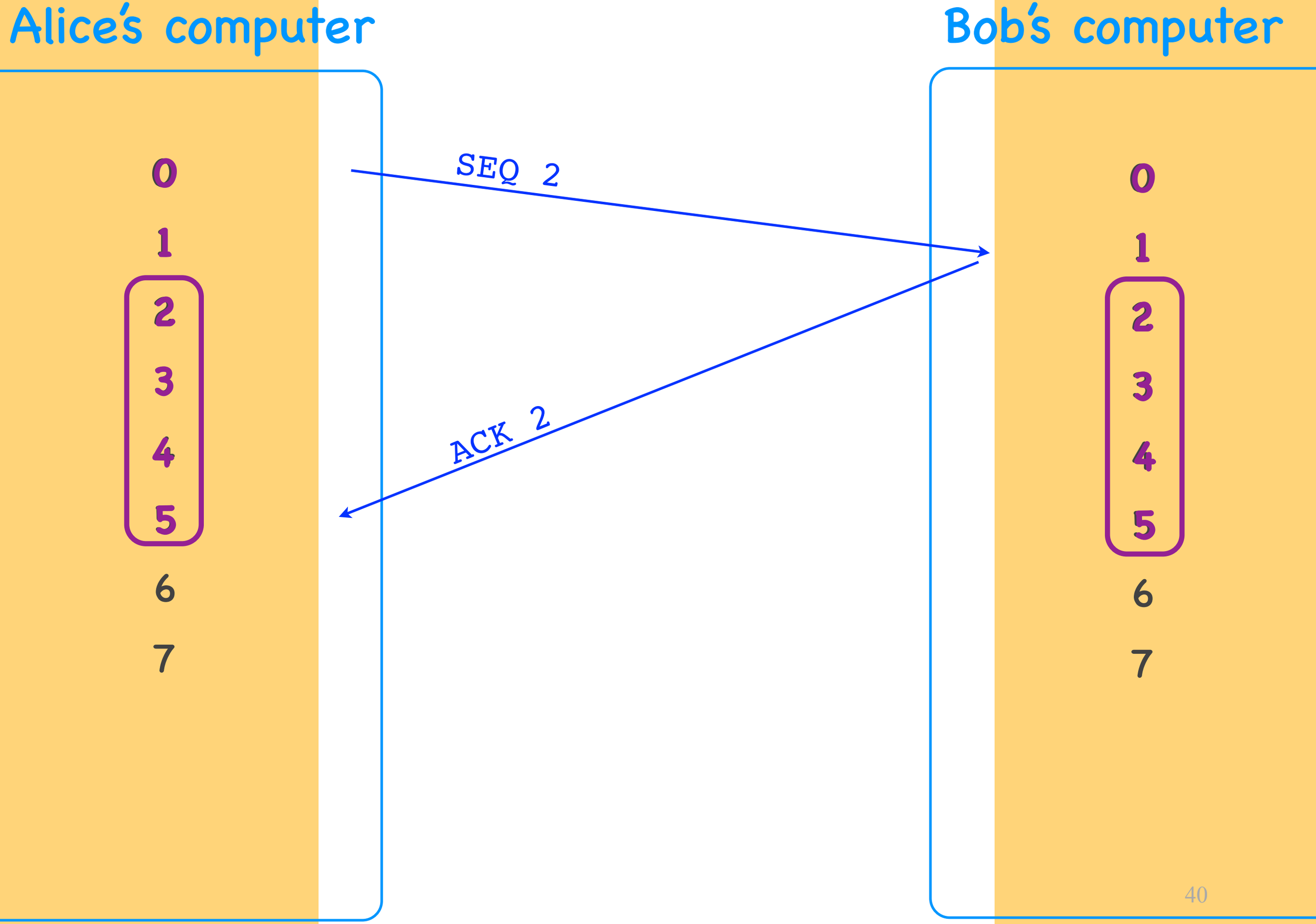
- The receiver accepts no out-of—order segments
- ACKs are cumulative
  - an ACK for segment 10 indicates that all segments until and including 10 have been received
- When the sender retransmits, it retransmits all the un-ACK-ed segments

Alice's computer

Bob's computer



timeout for packet 2


Alice's computer
Bob's computer
0
1
2
3
4
5
6
7
SEQ 2
ACK 2
0
1
2
3
4
5
6
7

Selective Repeat

- The receiver accepts N-1 out-of-order segments
- ACKs are selective
  - an ACK for segment 10 indicates that segment 10 has been received
- When the sender retransmits, it retransmits only one segment