VU VRIJE UNIVERSITEIT AMSTERDAM

The Transport Layer

(Part I)

Balakrishnan Chandrasekaran

(This slide deck borrows heavily from lecture materials of Jennifer Rexford, Anja Feldmann, and Bruce M. Maggs.)

Objectives

- Transport layer and its role
- Design Principles

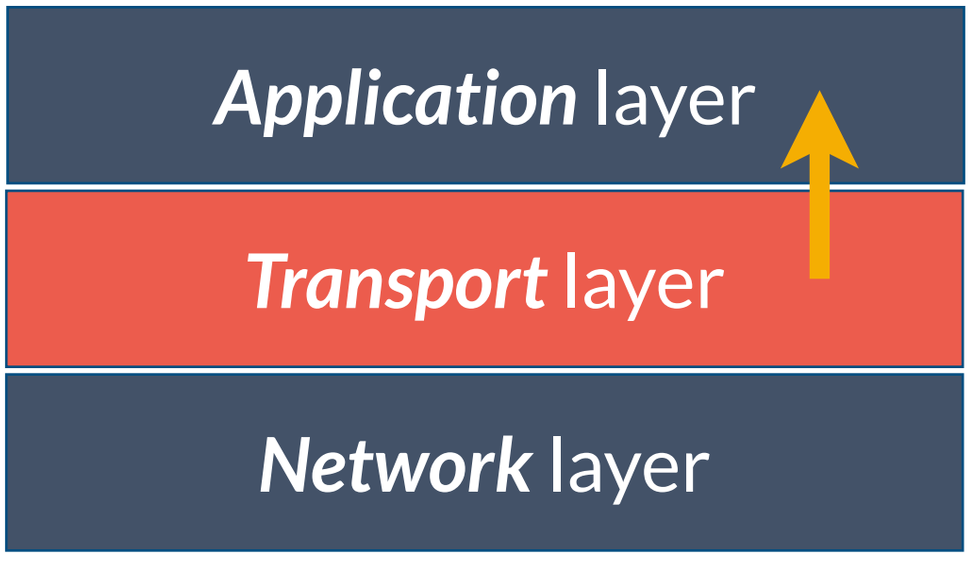
Transport layer

Application layer	Enables applications to access network services *(presentation, session handling, encryption, …)*
Transport layer	End-to-end communication services *(multiplexing, congestion control, flow control, …)*
Network layer	Transport of packets across networks *(routing, forwarding, addressing, …)*
Data link layer	Transfer of frames over a physical link *(framing, error correction, addressing, …)*
Physical layer	Transmission of bits over a channel *(encoding, decoding, transmission, …)*

Transport layer

Application layer	Enables applications to access network services *(presentation, session handling, encryption, …)*
Transport layer	**End-to-end** communication services *(multiplexing, congestion control, flow control, …)*
Network layer	Transport of packets across networks *(routing, forwarding, addressing, …)*
Data link layer	Transfer of frames over a physical link *(framing, error correction, addressing, …)*
Physical layer	Transmission of bits over a channel *(encoding, decoding, transmission, …)*

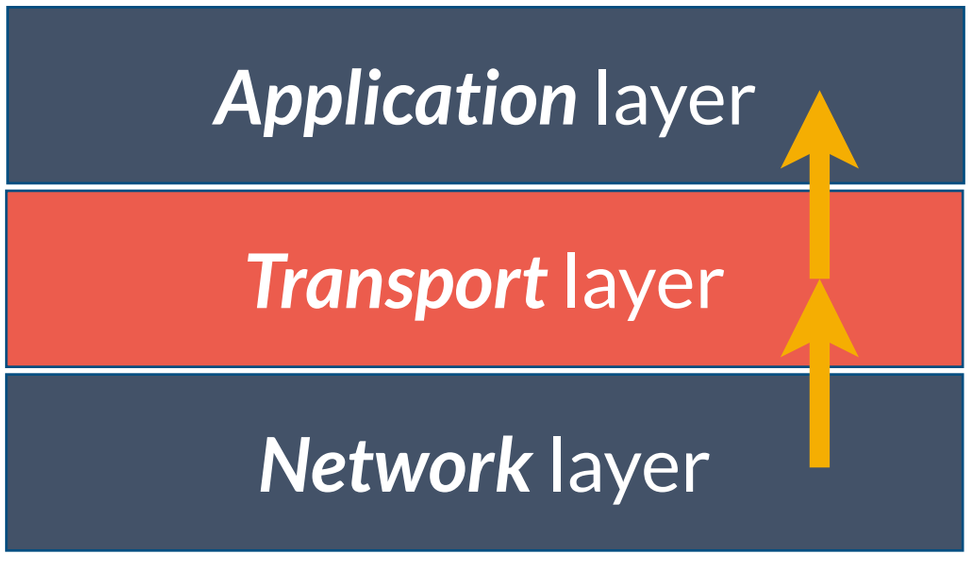
Transport layer



Offers various functionalities to the application

(reliable data transfer, congestion control, flow control, …)

Transport layer



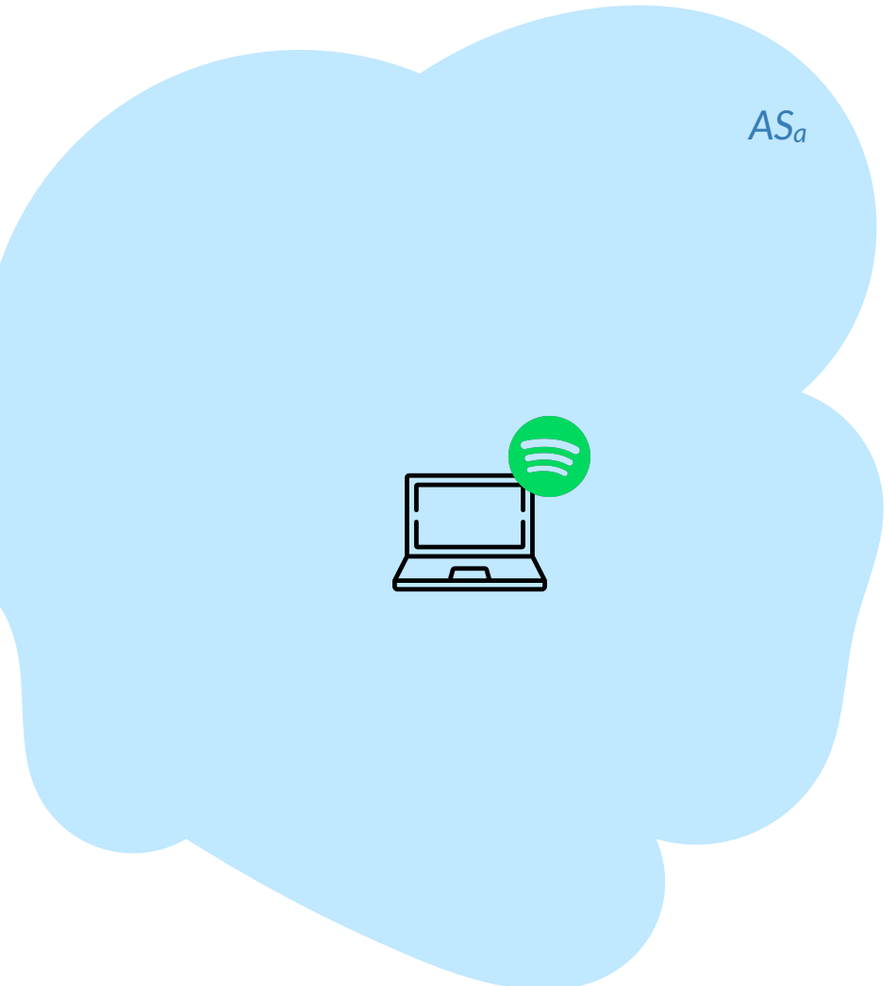
Offers various functionalities to the application

(reliable data transfer, congestion control, flow control, …)

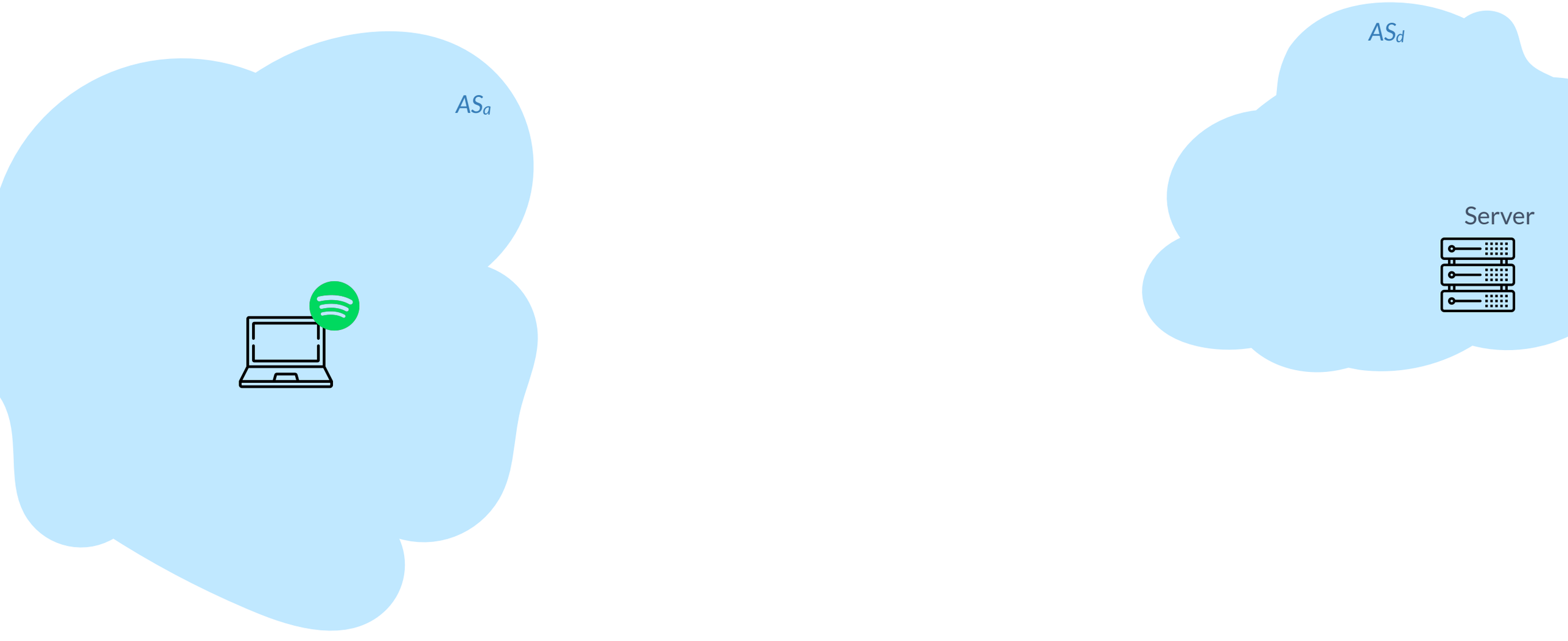
Relies on *services* exposed by the network

(Network layer provides the path for the transport.)

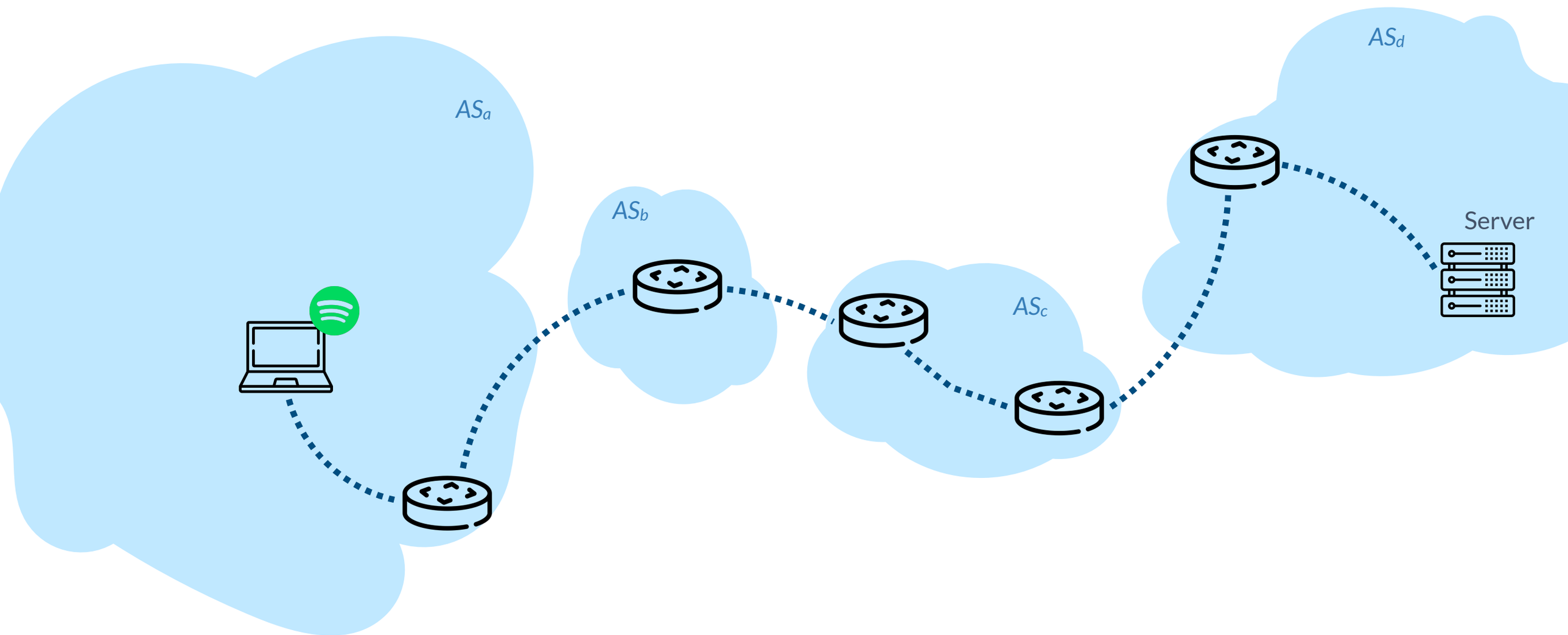
End-to-End protocol



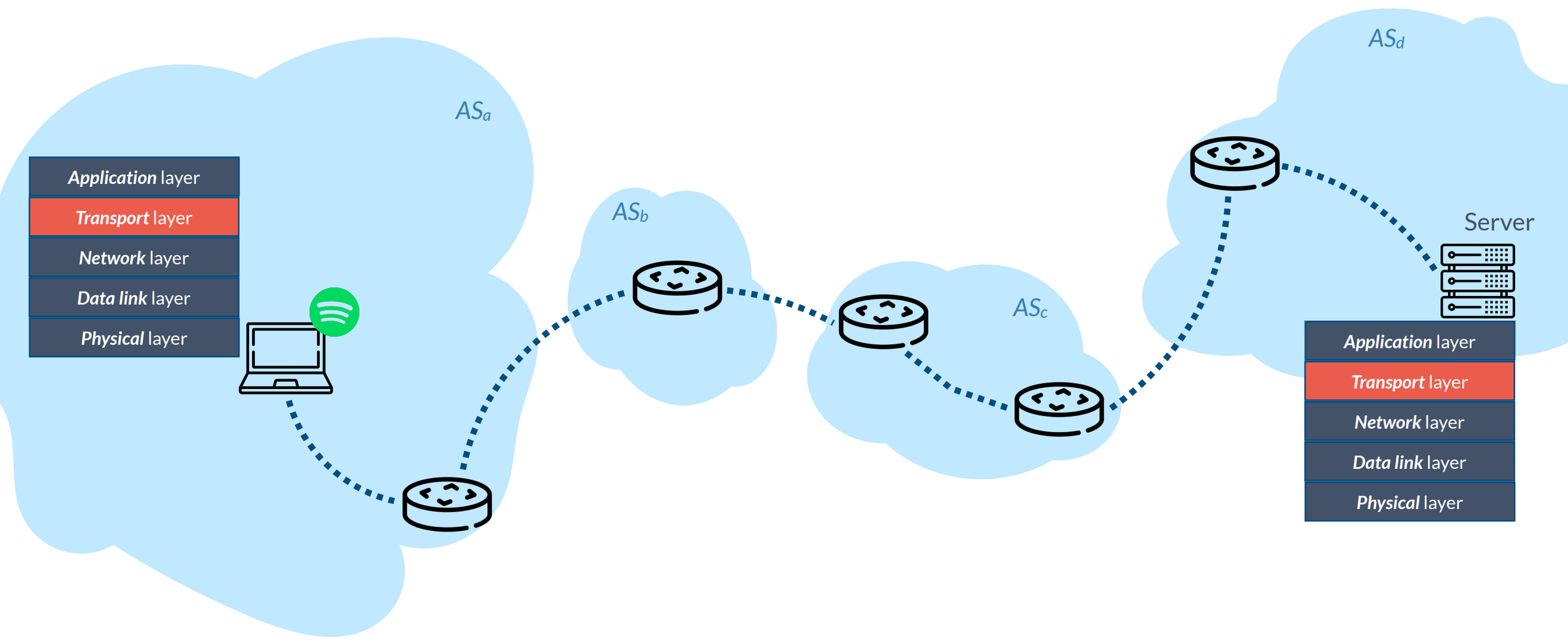
End-to-End protocol



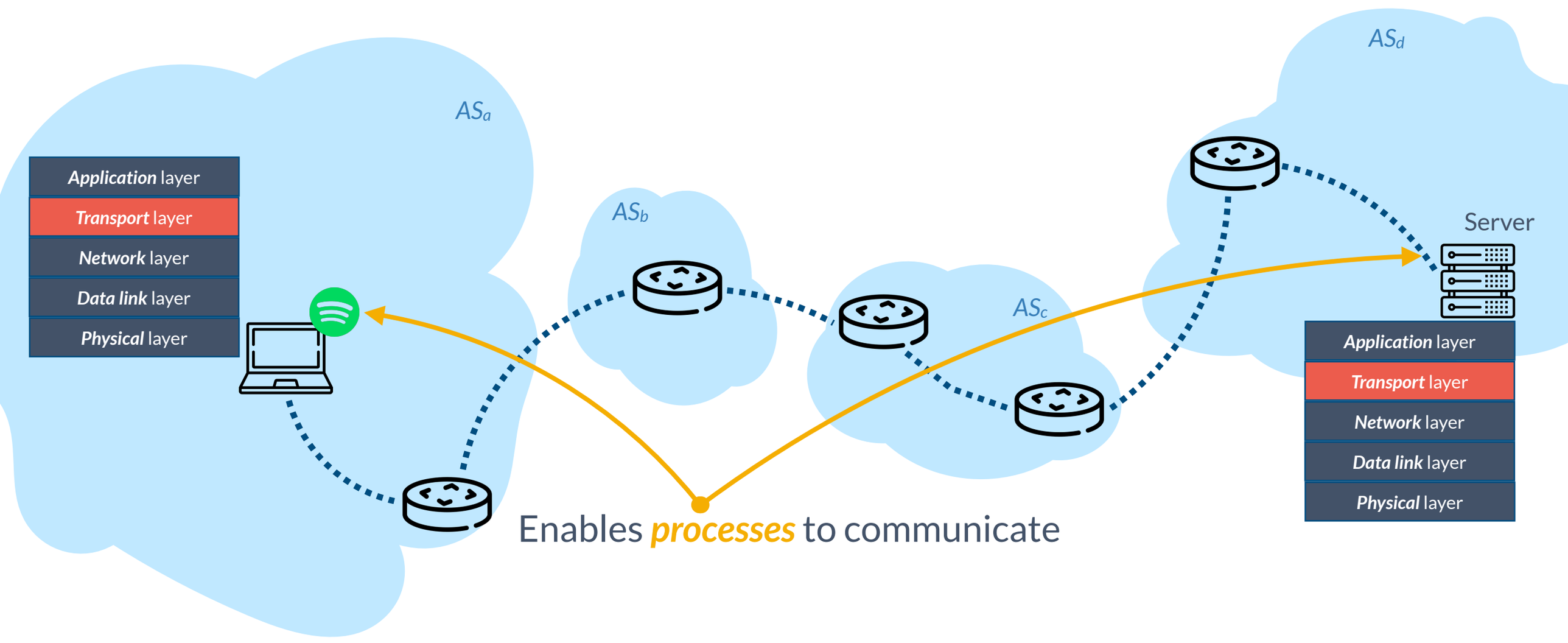
End-to-End protocol



End-to-End protocol

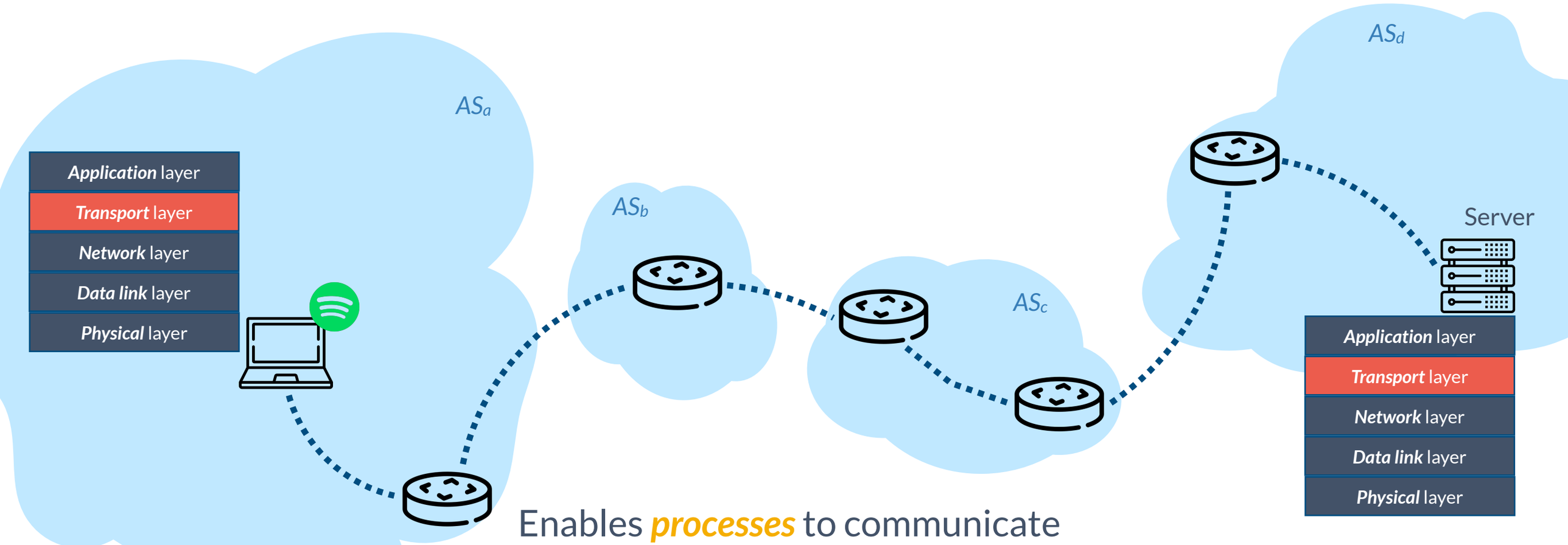


End-to-End protocol



Enables ***processes*** to communicate

End-to-End protocol

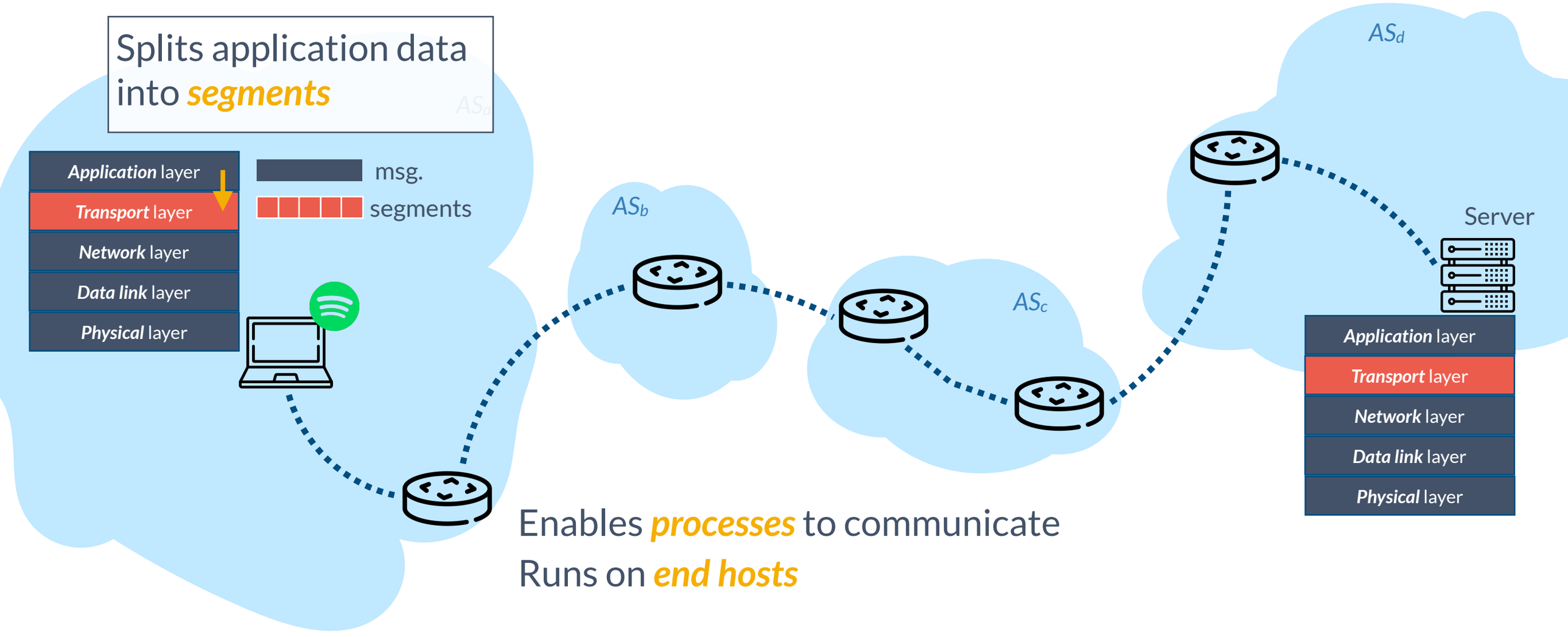


Enables ***processes*** to communicate

Runs on ***end hosts***

End-to-End protocol

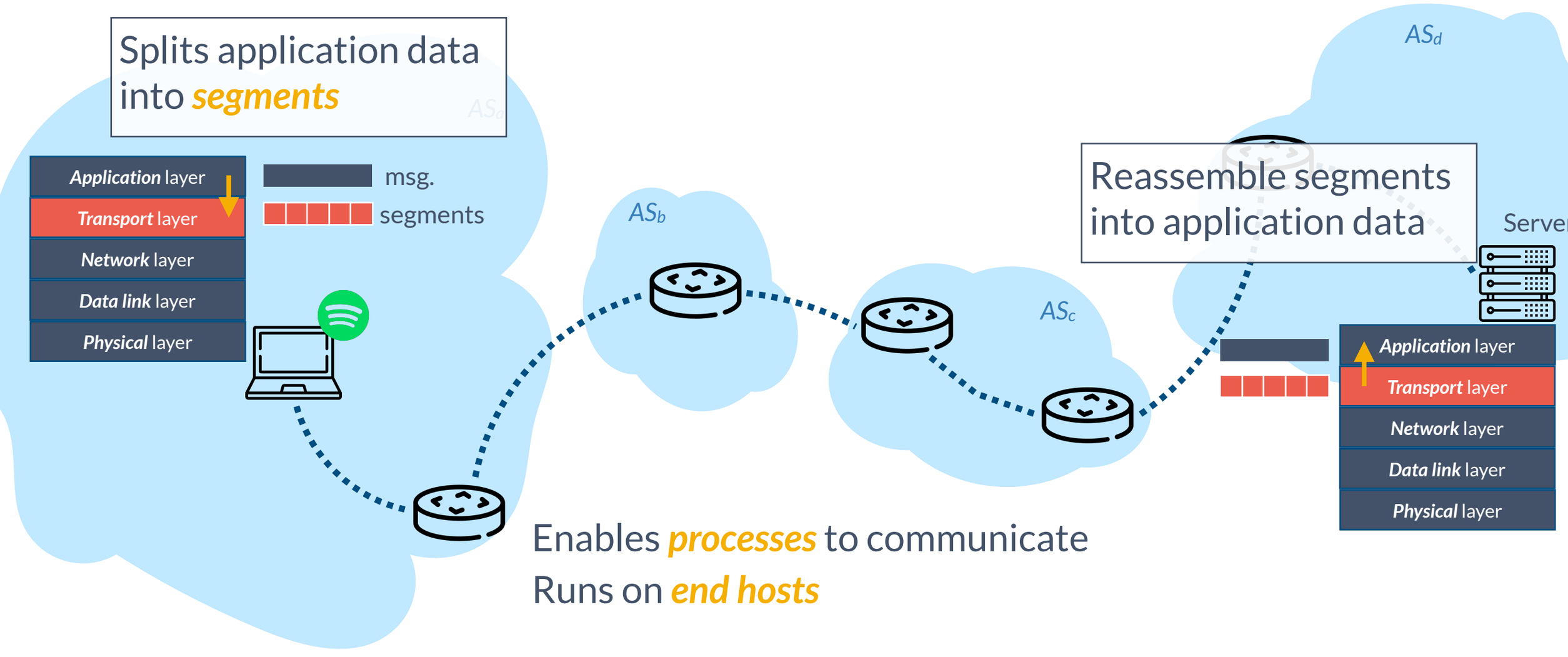
Splits application data into ***segments***



Enables ***processes*** to communicate

Runs on ***end hosts***

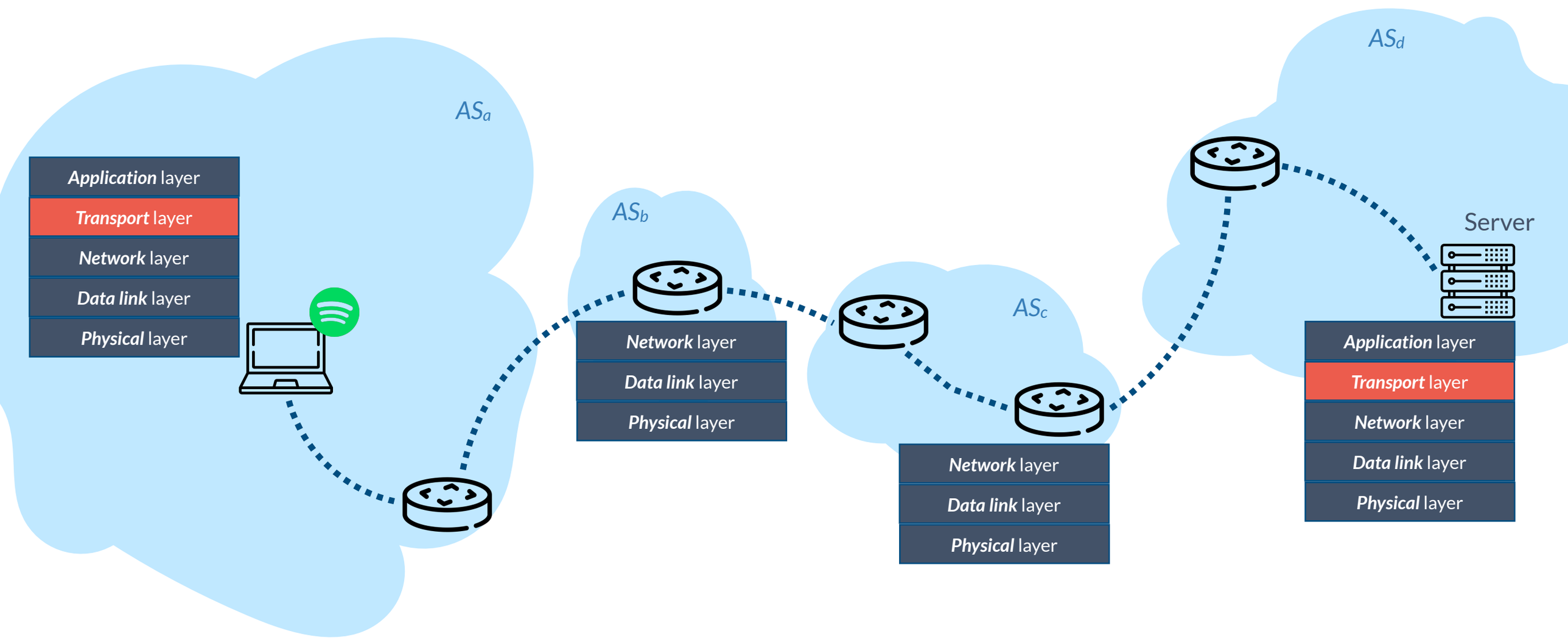
End-to-End protocol



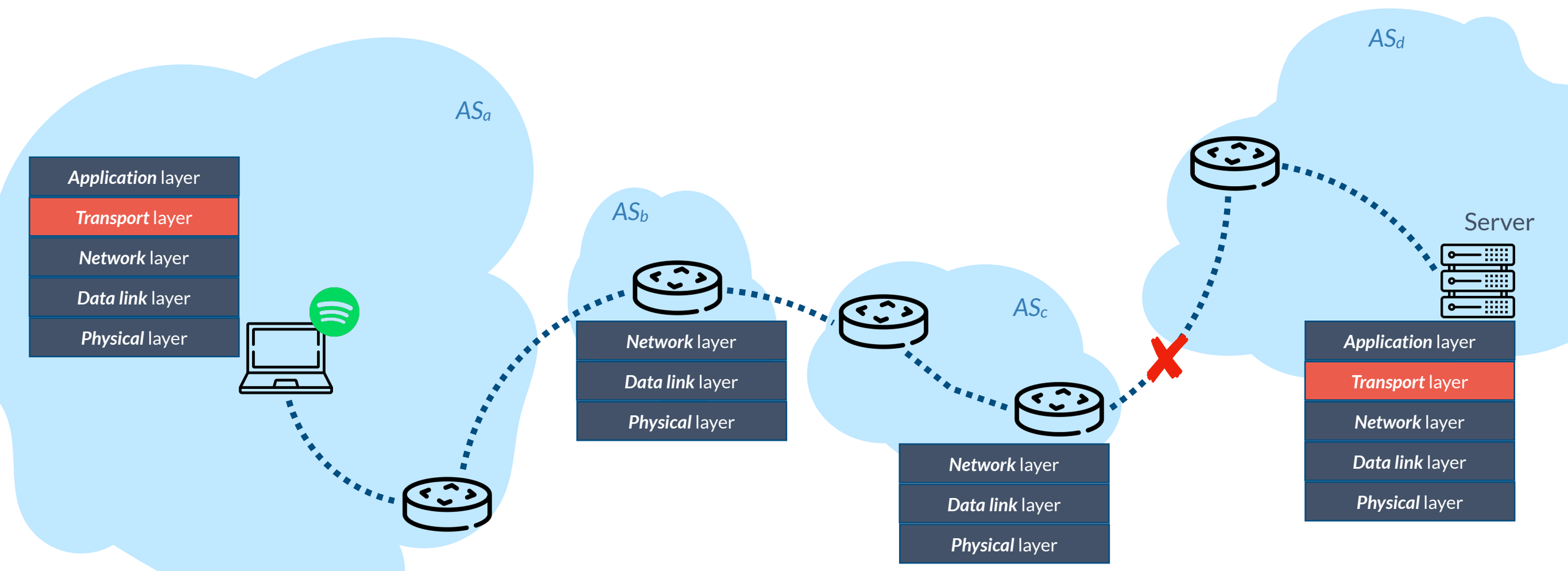
Enables *processes* to communicate

Runs on *end hosts*

End-to-End protocol

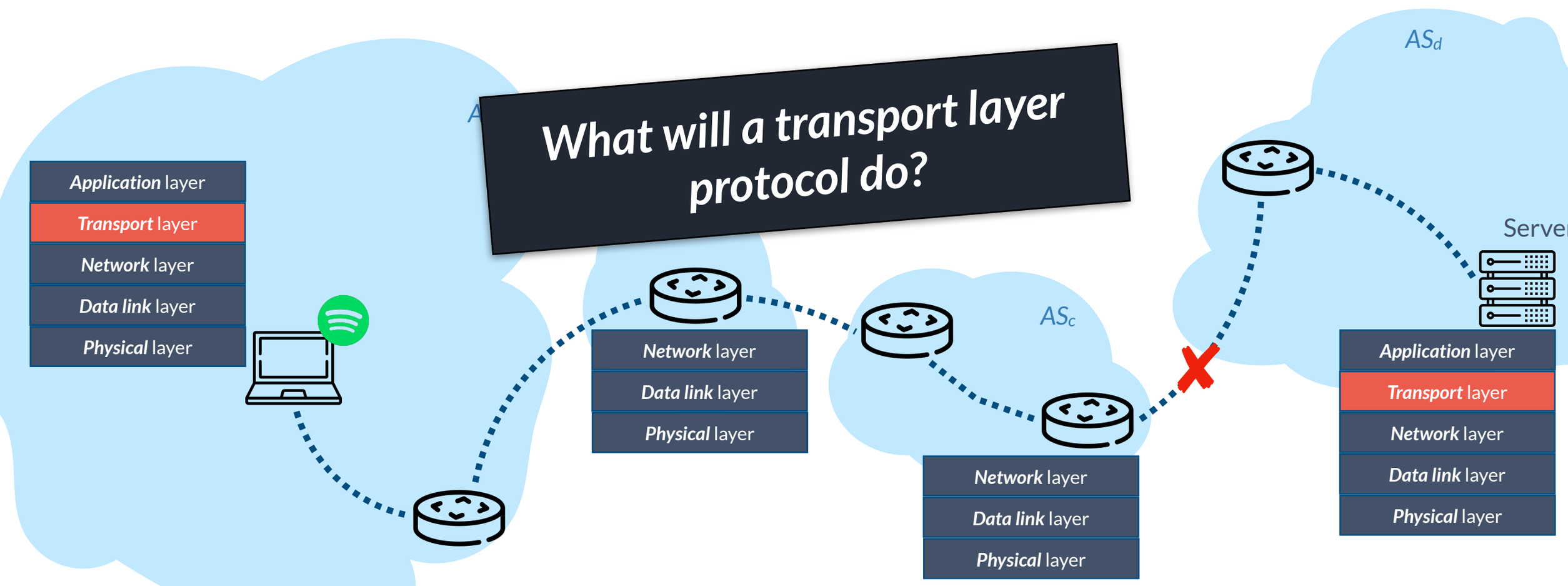


End-to-End protocol



Suppose a link along the network path fails while the processes are exchanging data …

End-to-End protocol



Suppose a link along the network path fails while the processes are exchanging data …

Transport protocols

- *Datagram messaging service*
  - *Connection-less*
  - *Best-effort* delivery; “fire-and-forget”

Transport protocols

- *Datagram messaging service*
  - *Connection-less*
  - *Best-effort* delivery; “fire-and-forget”

User Datagram Protocol (UDP)

VU VRIJE UNIVERSITEIT AMSTERDAM

Transport protocols

- *Datagram messaging service*
  - *Connection-less*
  - *Best-effort* delivery; “fire-and-forget”

User Datagram Protocol (UDP)

- *Reliable, in-order transfer service*
  - *Connection* support
  - *Reliable* delivery
  - *Congestion control* and *flow control*

Transport protocols

- *Datagram messaging service*
  - *Connection-less*
  - *Best-effort* delivery; “fire-and-forget”

User Datagram Protocol (UDP)

- *Reliable, in-order transfer service*
  - *Connection* support
  - *Reliable* delivery
  - *Congestion control* and *flow control*

Transmission Control Protocol (TCP)
Quick UDP Internet Connections (QUIC)

Transport protocols

- *Datagram messaging service*
  - *Connection-less*
  - *Best-effort* delivery; “fire-and-forget”

User Datagram Protocol (UDP)

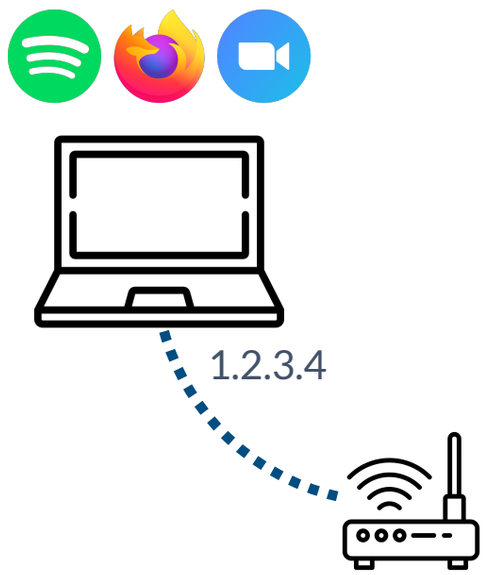
What about latency and bandwidth guarantees?

- *Reliable, in-order*
  - *Connection* support
  - *Reliable* delivery
  - *Congestion control* and *flow control*

Transmission Control Protocol (TCP)
Quick UDP Internet Connections (QUIC)

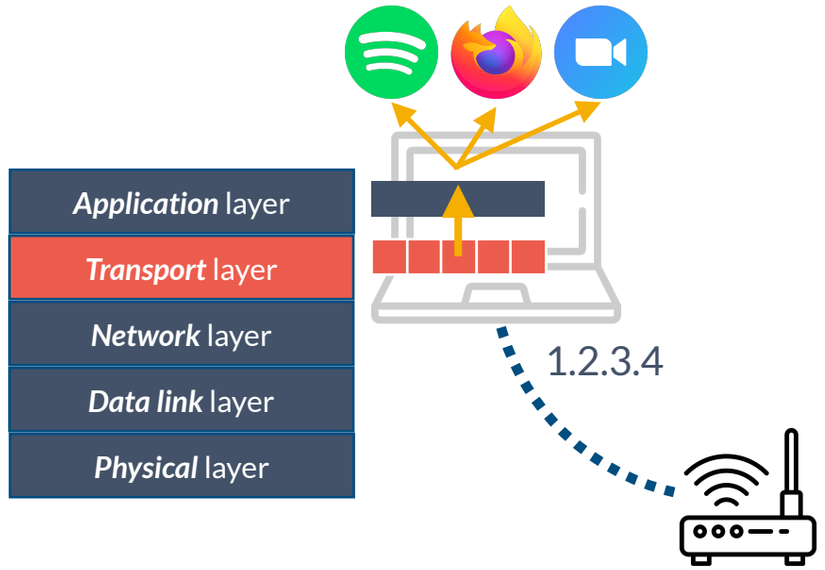
(De)Multiplexing traffic

- All applications running on the end host have the *same* IP address



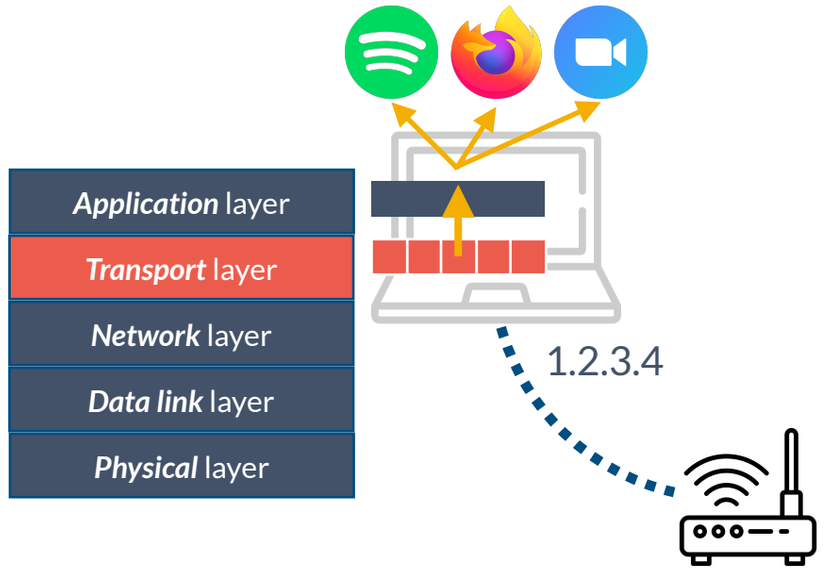
(De)Multiplexing traffic

- All applications running on the end host have the *same* IP address
  - Who should receive reassembled segments?



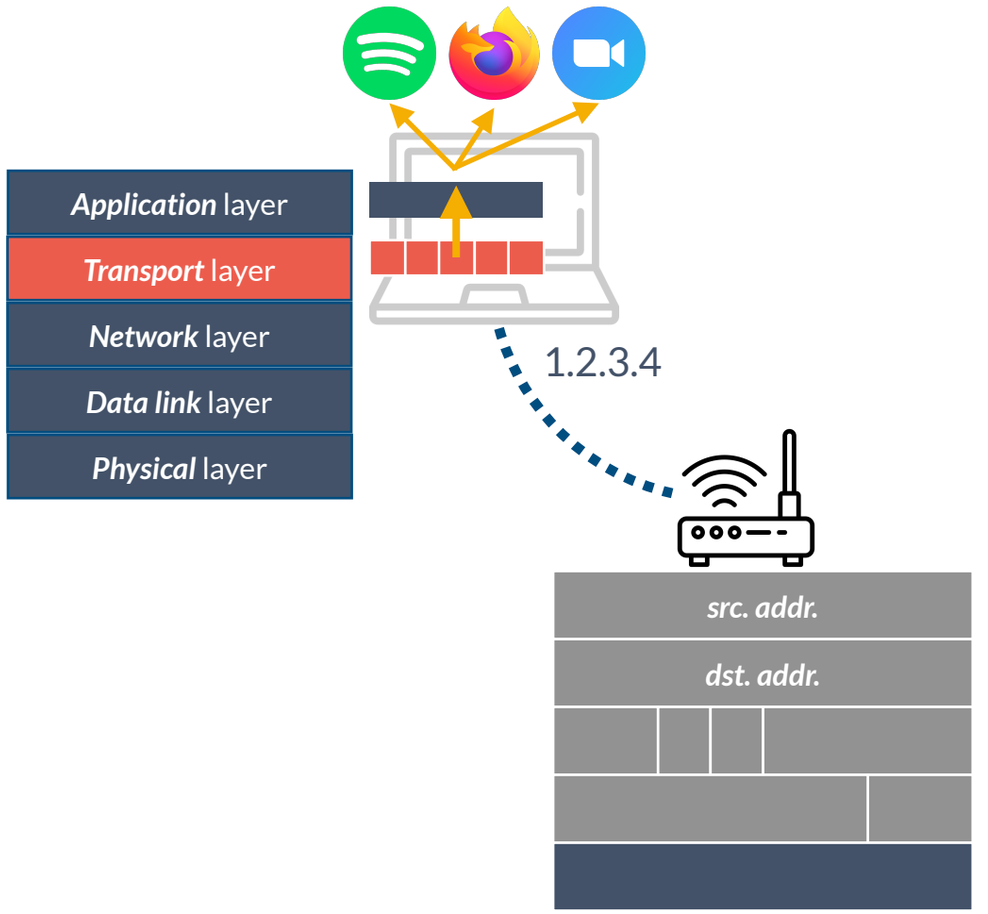
(De)Multiplexing traffic

- All applications running on the end host have the *same* IP address
  - Who should receive reassembled segments?
  - Transport layer needs more than IP addresses to identify *end points*



(De)Multiplexing traffic

- All applications running on the end host have the *same* IP address
  - Who should receive reassembled segments?
  - Transport layer needs more than IP addresses to identify *end points*



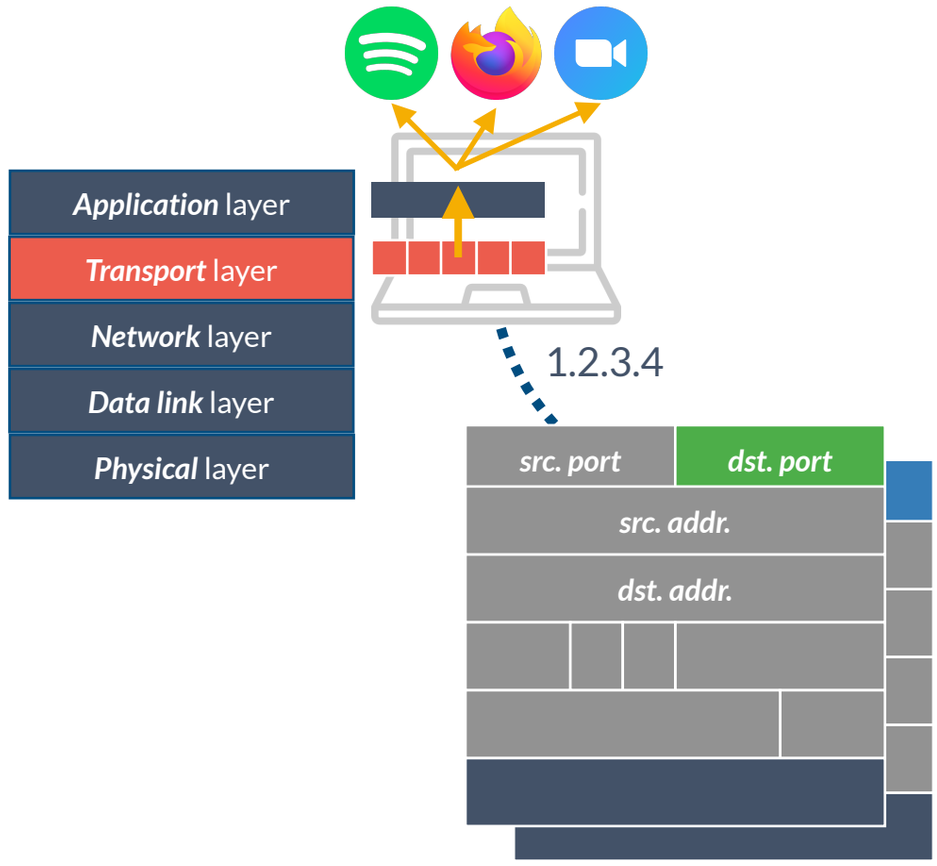
- Host receives IP *datagrams*, each of which
  - has a *source* and *destination* IP address
  - carries *one* transport *segment*

(De)Multiplexing traffic

- All applications running on the end host have the *same* IP address
  - Who should receive reassembled segments?
  - Transport layer needs more than IP addresses to identify *end points*

- Host receives IP *datagrams*, each of which
  - has a *source* and *destination* IP address
  - carries *one* transport *segment*

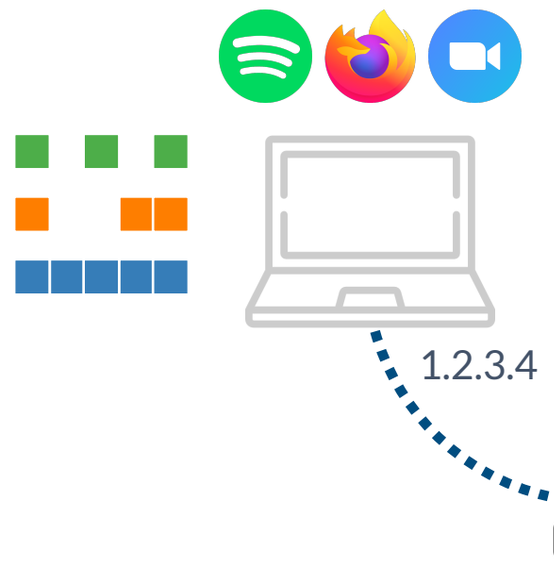
- Each *segment* has a *source* and *destination port* number



IP addresses and port numbers determine which application gets which segments.

(De)Multiplexing traffic

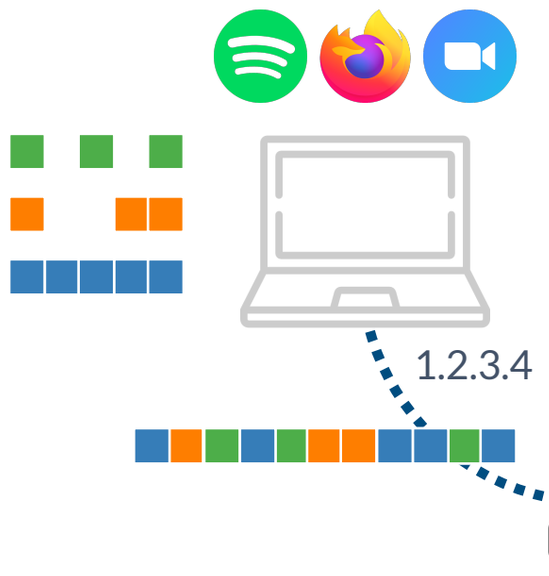
- Different applications may generate different traffic workloads
  - Browsing: typically *bursty*



(De)Multiplexing traffic

- Different applications may generate different traffic workloads
  - Browsing: typically *bursty*

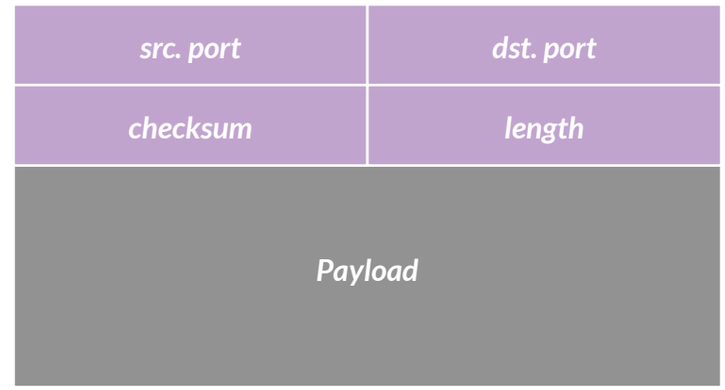
- *Statistical multiplexing*
  - Maximize *utilization* by multiplexing different applications' traffic



Unreliable message delivery

- *User Datagram Protocol (UDP)*
  - Source and destination ports
  - Checksum for error checking, if required

- Simple protocol
  - *Best-effort* delivery



UDP: useless or useful?

VU VRIJE UNIVERSITEIT AMSTERDAM

UDP: useless or useful?

- *No delays*
  - No connection setup
  - *Fine-grained* control: send *whenever* you want to

UDP: useless or useful?

- *No delays*
  - No connection setup
  - *Fine-grained* control: send *whenever* you want to

- *No state*
  - No “continuity” between different segments; scales easily

UDP: useless or useful?

- *No delays*
  - No connection setup
  - *Fine-grained* control: send *whenever* you want to

- *No state*
  - No “continuity” between different segments; scales easily

- *Minimal overhead*
  - Only *8 bytes*!

UDP: useless or useful?

- *No delays*
  - No connection setup
  - *Fine-grained* control: send *whenever* you want to

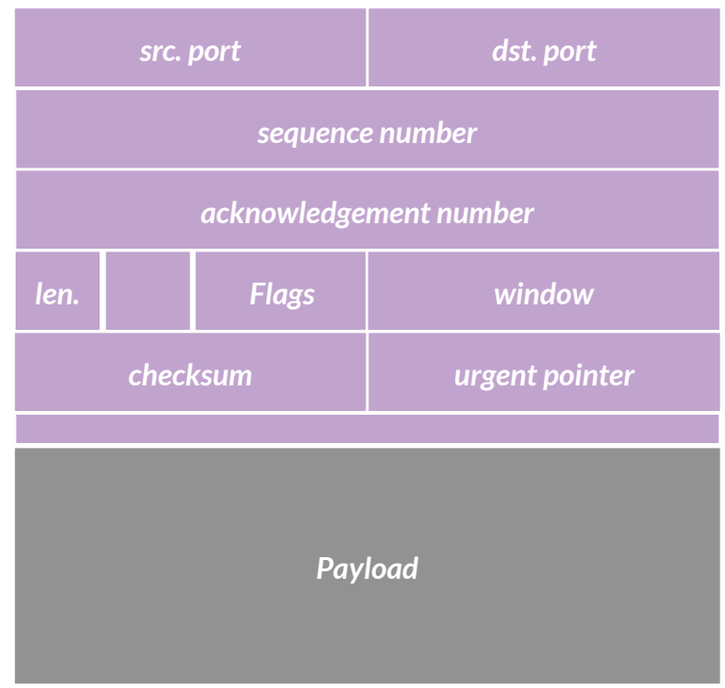
- *No state*
  - No “continuity” between different segments; scales easily

- *Minimal overhead*
  - Only *8 bytes*!

- Example applications
  - *DNS, Voice over IP (VOIP), Video conferencing, …*

Reliable data transfer

- *Transmission Control Protocol (TCP)*
  - Source and destination ports
  - Sequence and acknowledgement numbers
  - At least 20-bytes of header
- Rich protocol
  - *Connection-oriented*
  - *Stream-of-bytes* service
  - *Reliable*, *in-order* delivery
  - *Flow* control
  - *Congestion* control

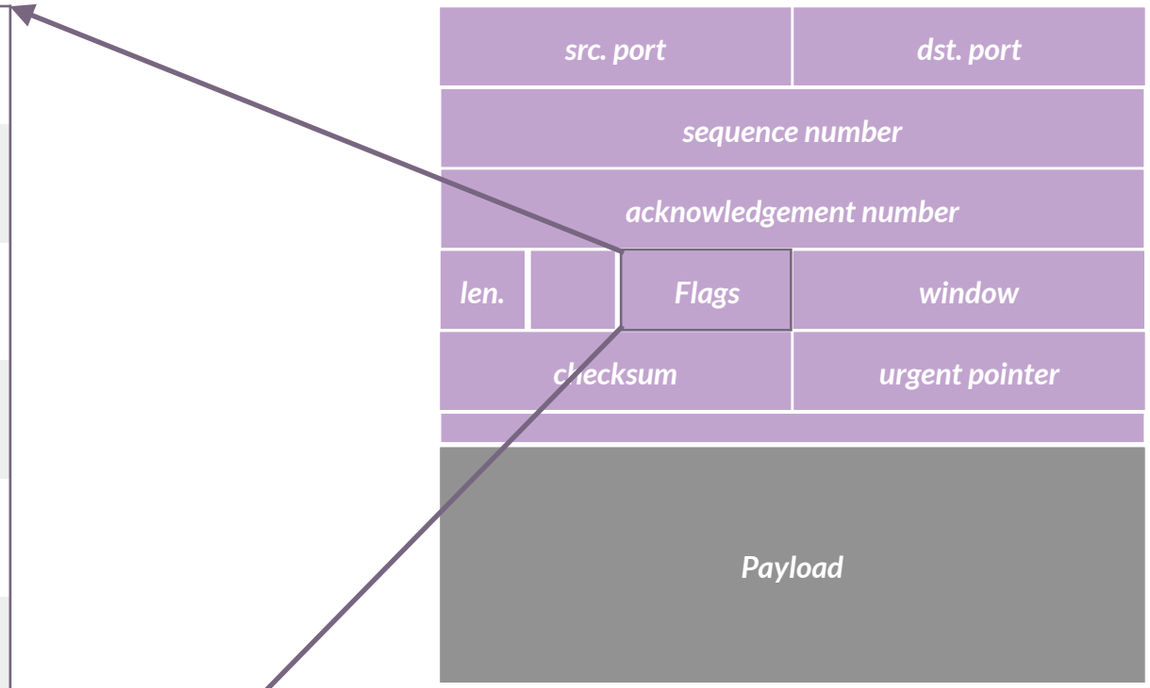


Reliable data transfer challenges

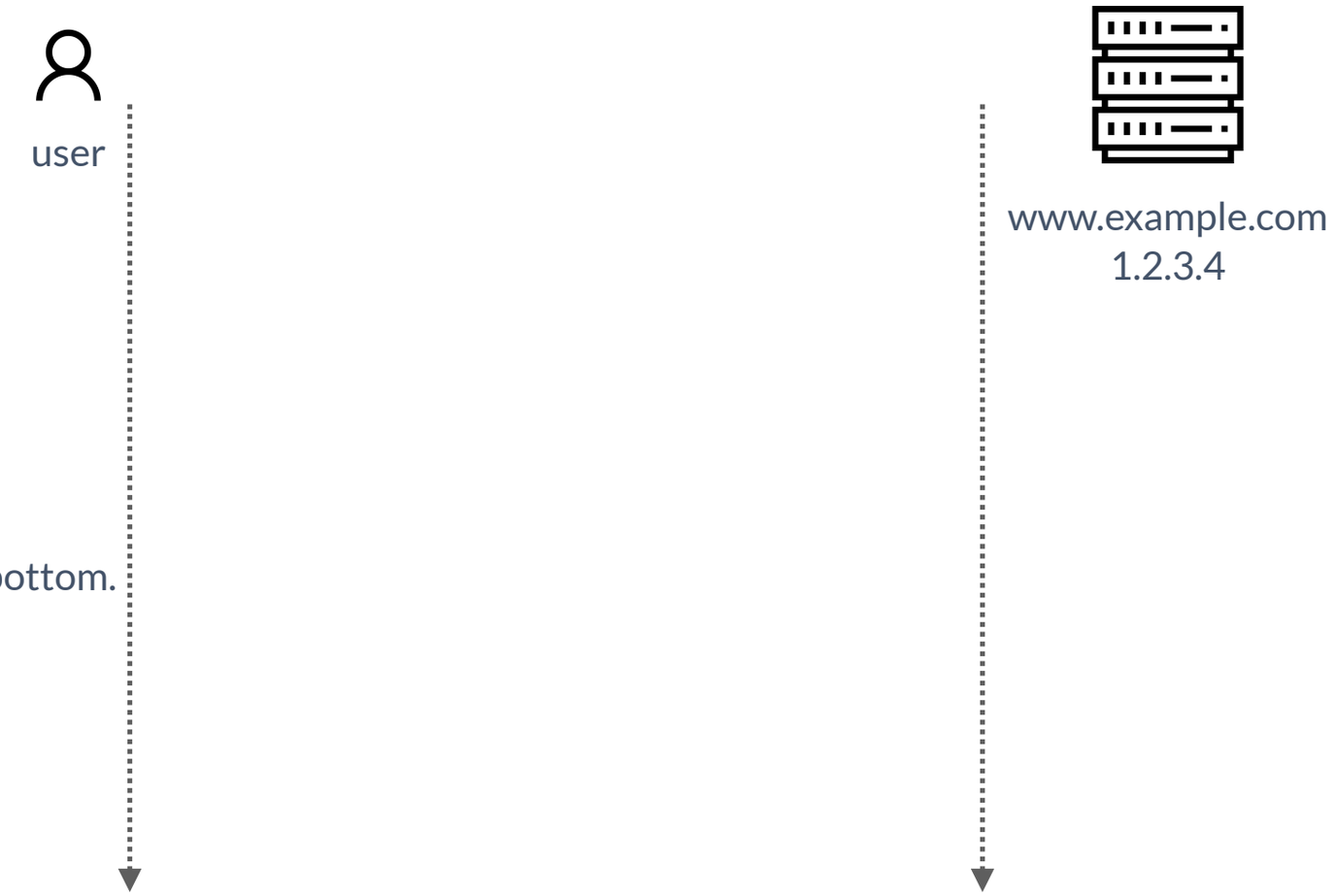
VU VRIJE UNIVERSITEIT AMSTERDAM

TCP “Handshake”

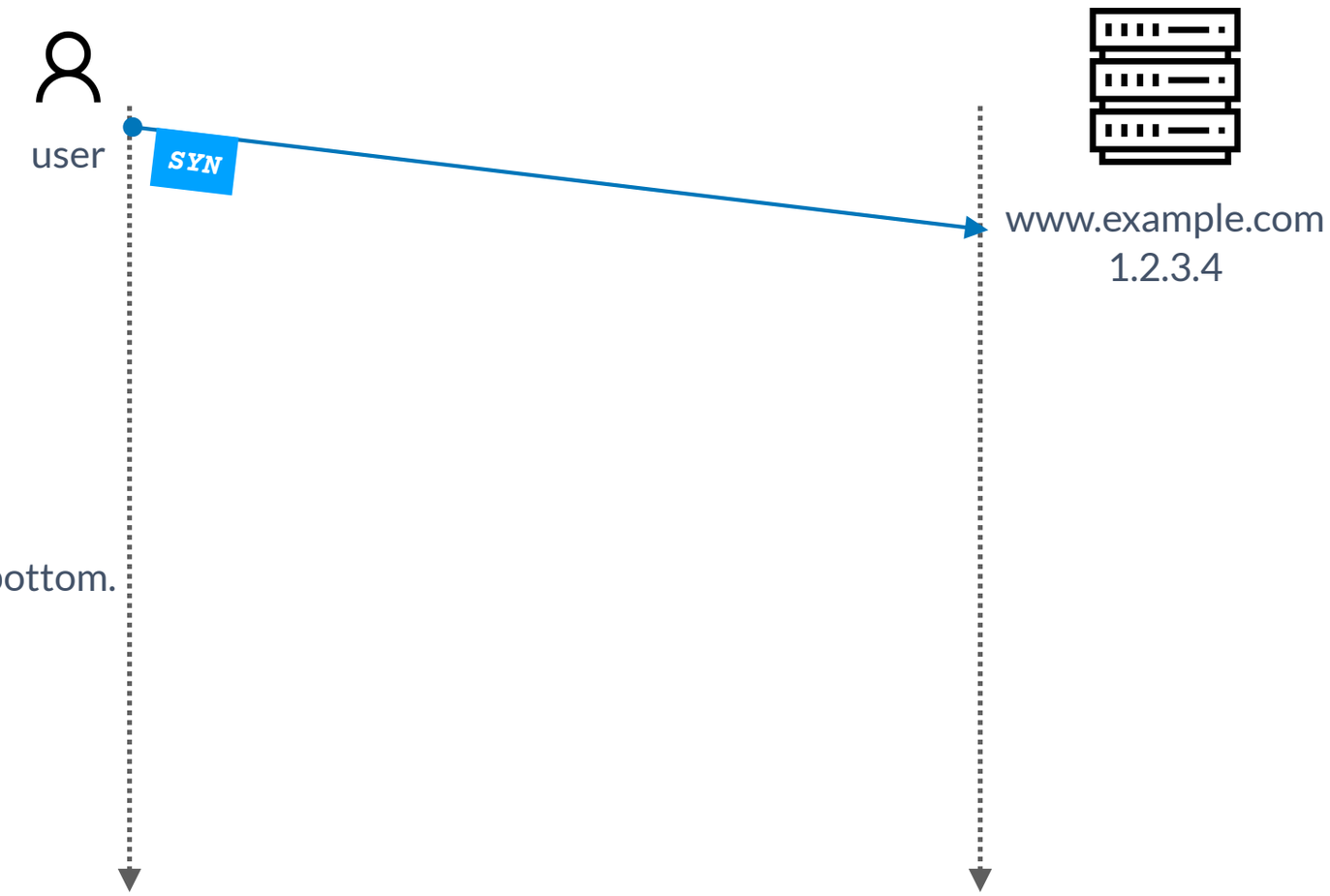
CWR	Congestion window reduced
ECE	ECN Echo
URG	Urgent pointer is significant
ACK	Acknowledgement *(All segments after the initial SYN will have it set)*
PSH	Push buffered data (on the wire)
RST	Reset connection
SYN	Synchronize sequence numbers
FIN	Last packet from sender



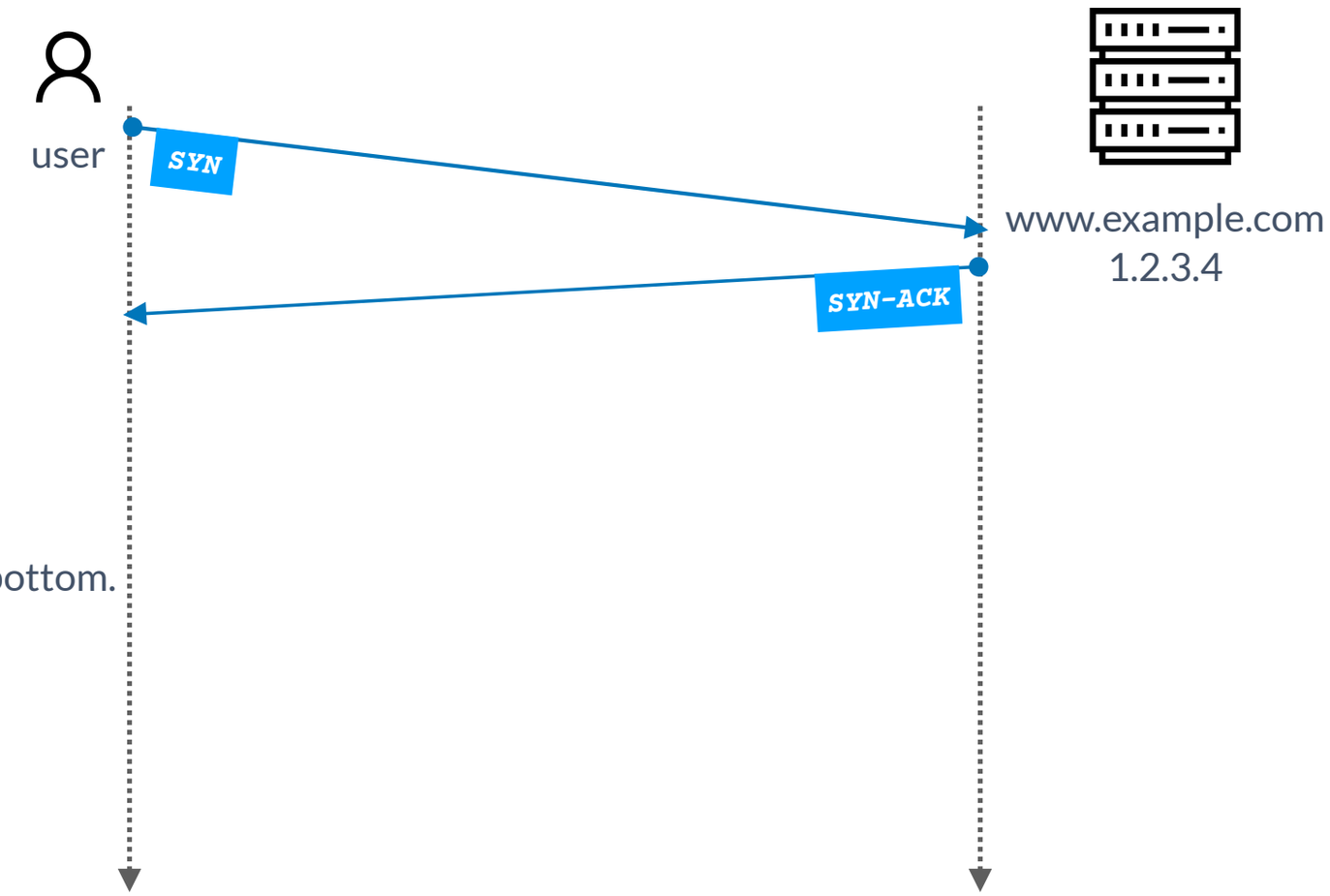
TCP “Handshake”



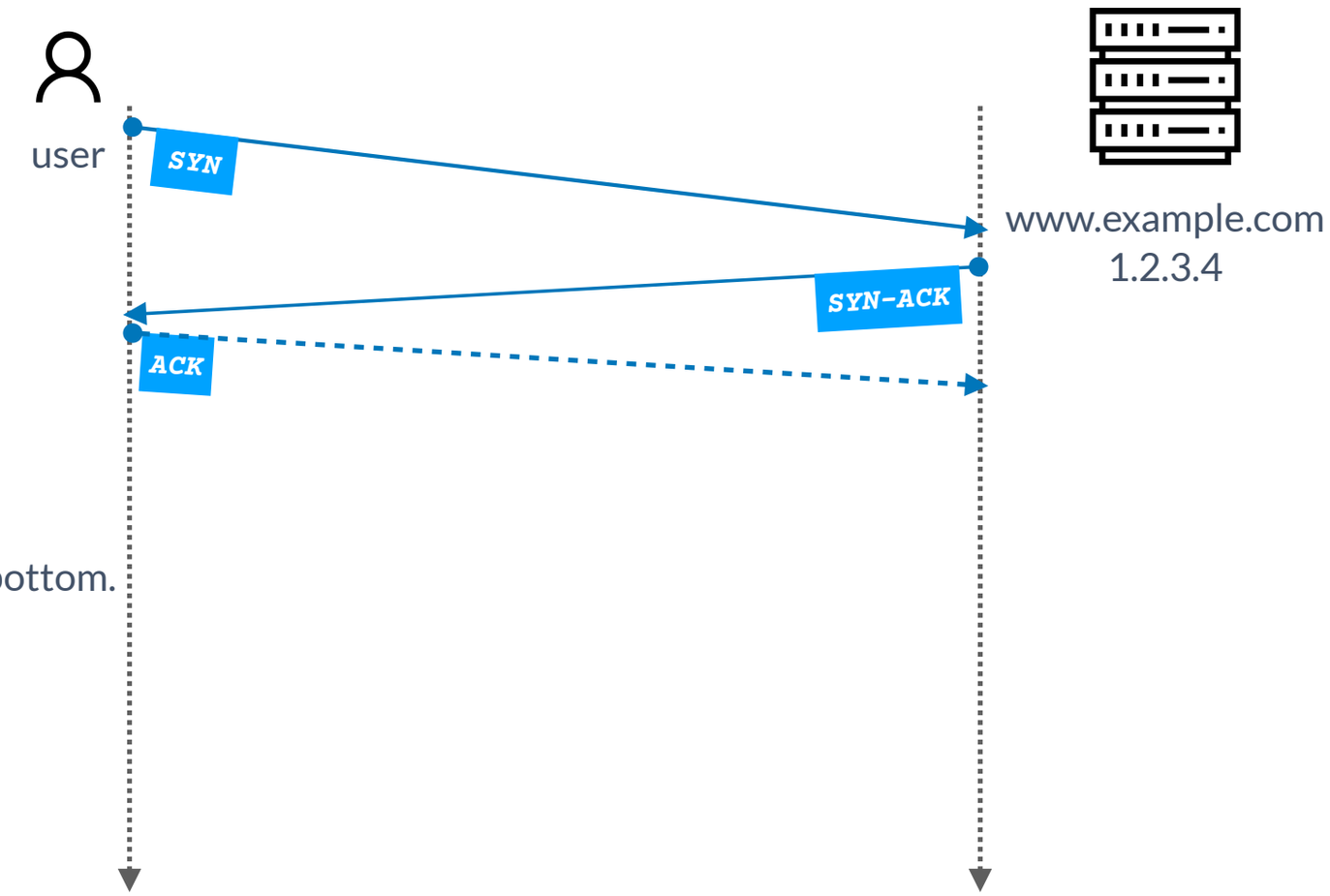
TCP “Handshake”



TCP “Handshake”

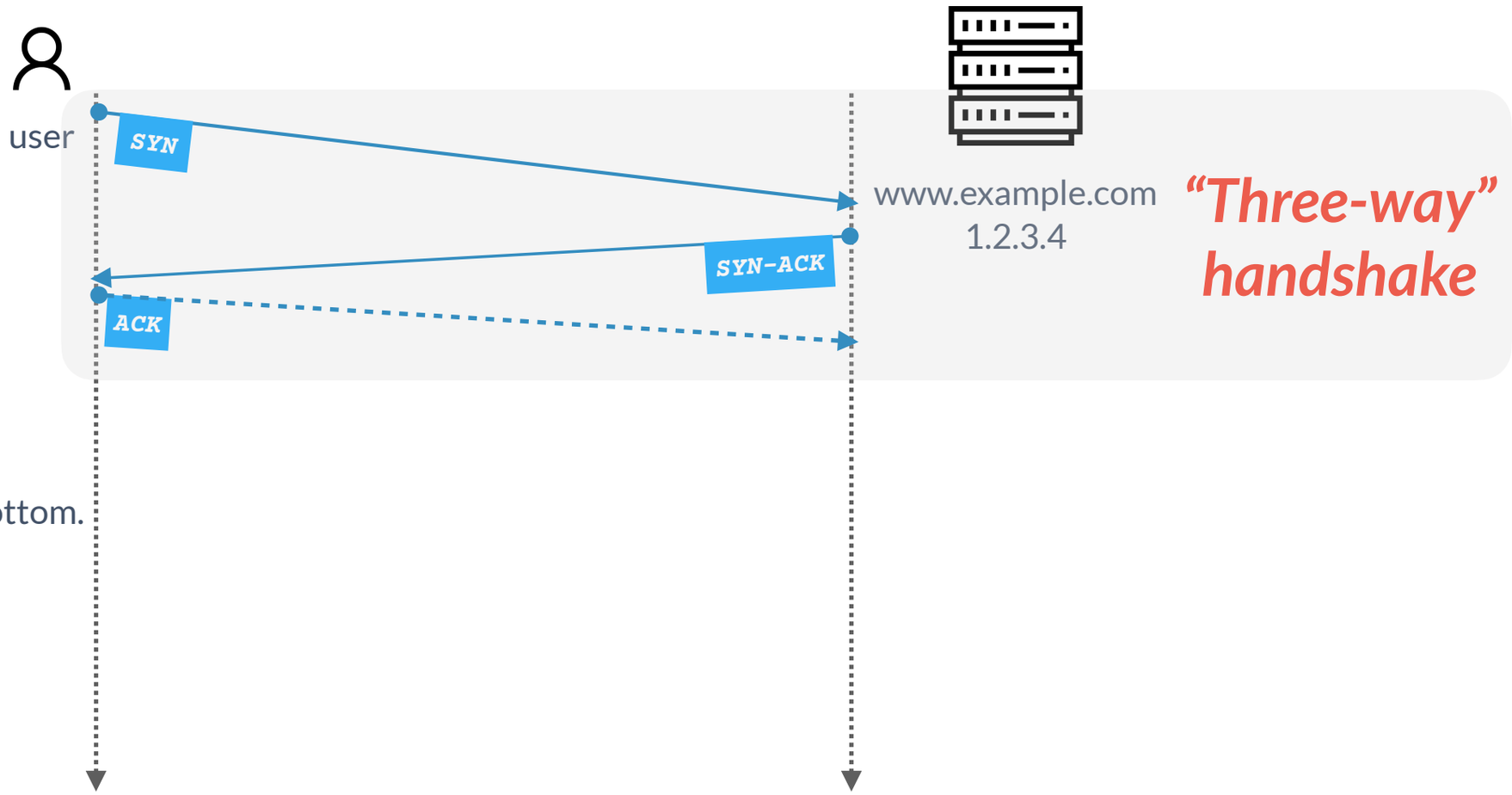


TCP “Handshake”



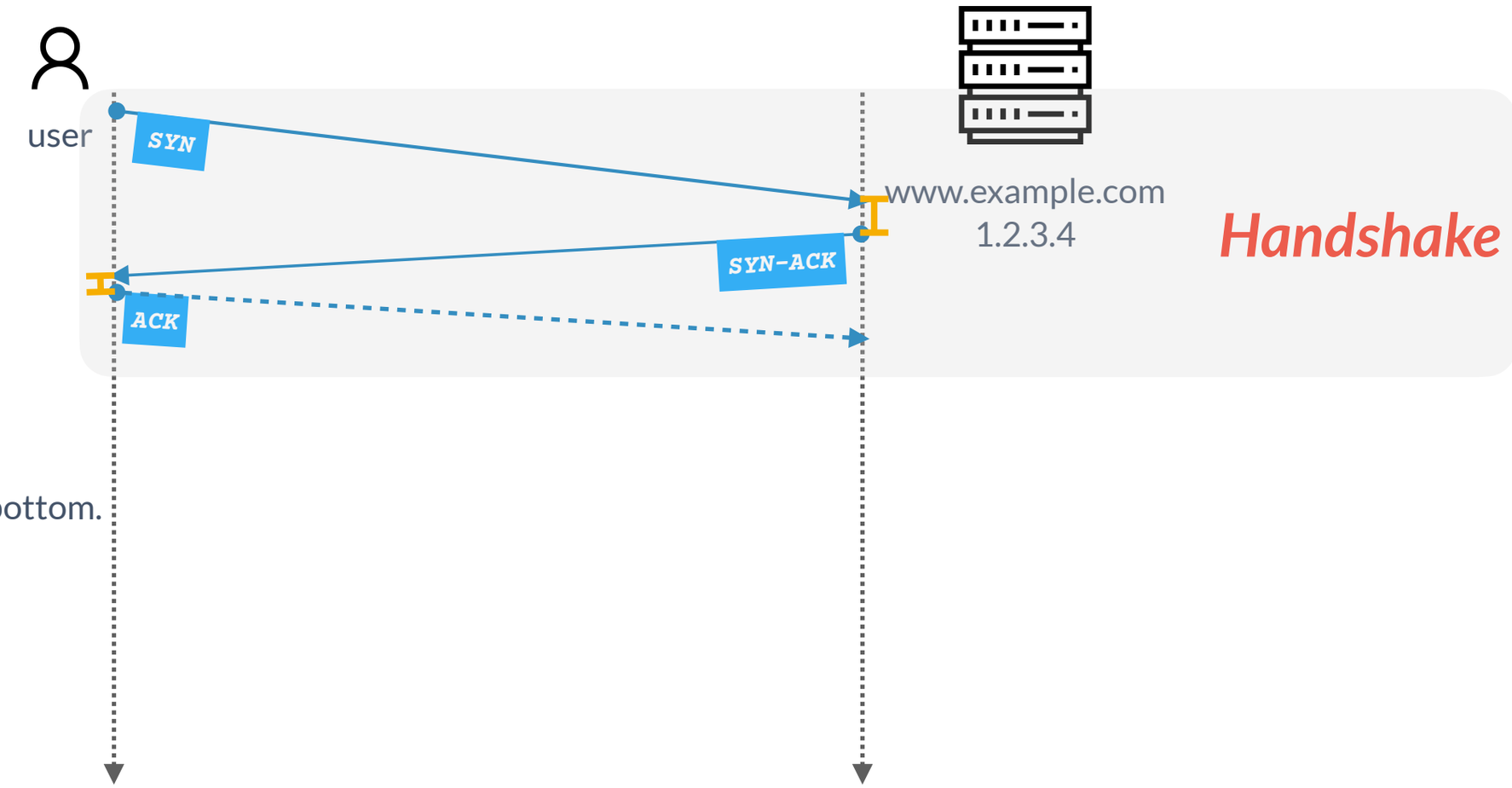
Time moves from top to bottom.

TCP “Handshake”



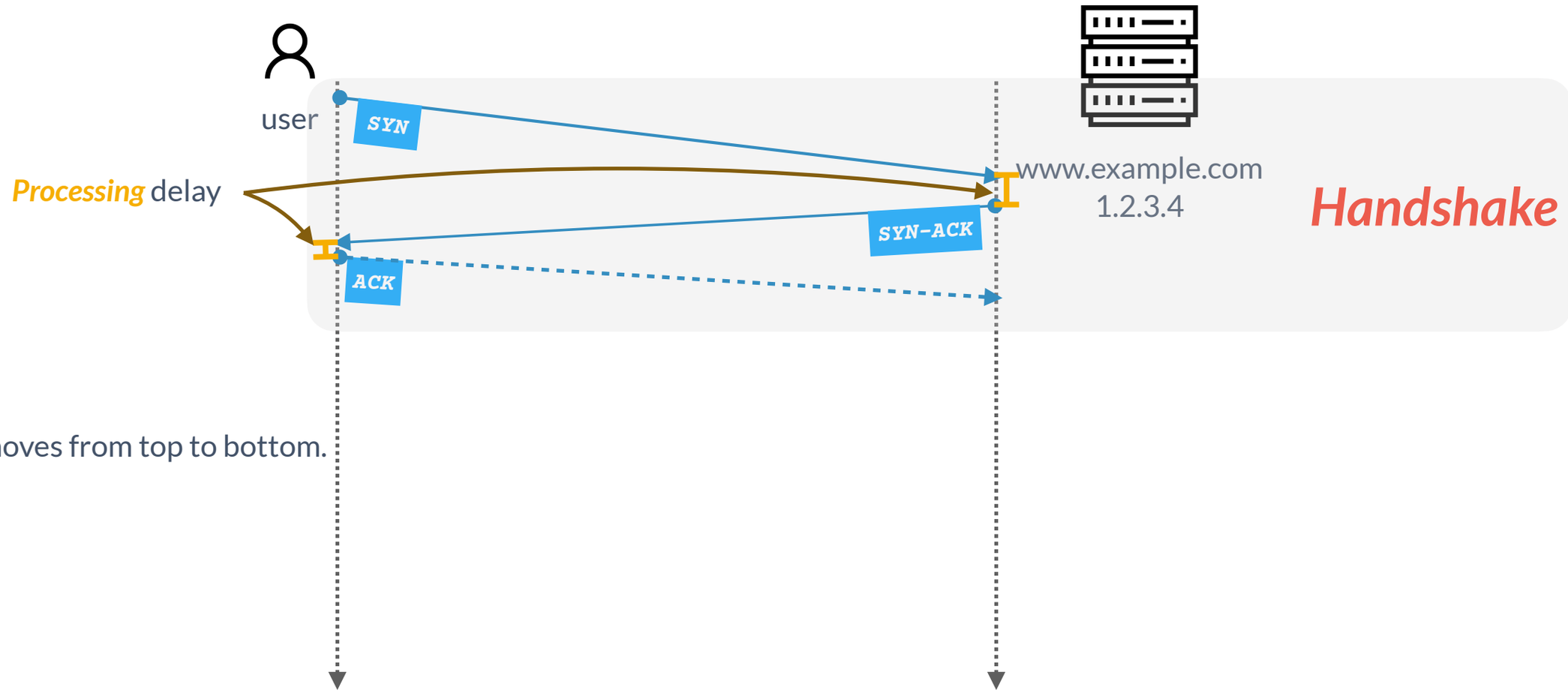
Time moves from top to bottom.

TCP “Handshake”

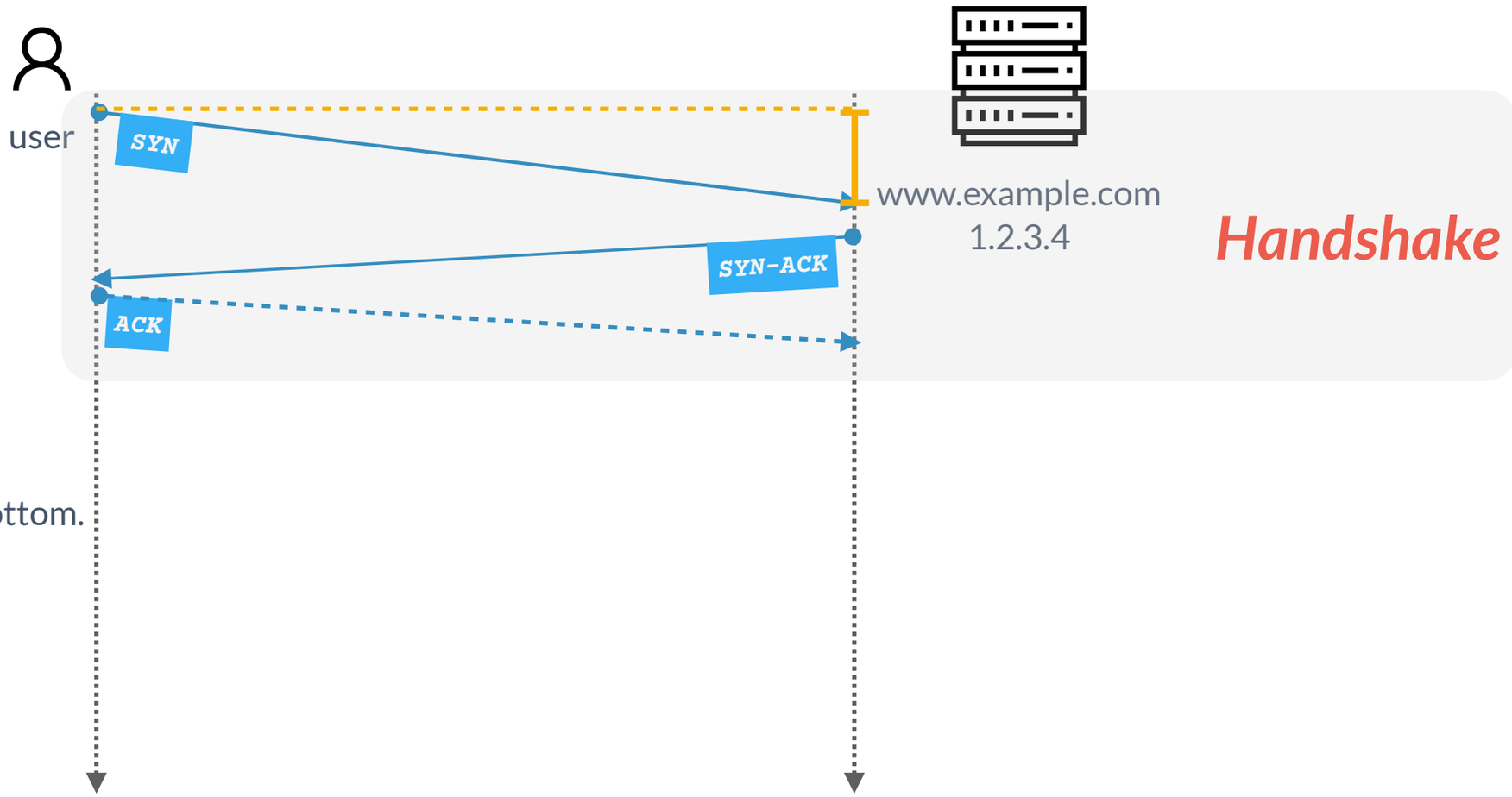


Time moves from top to bottom.

TCP “Handshake”

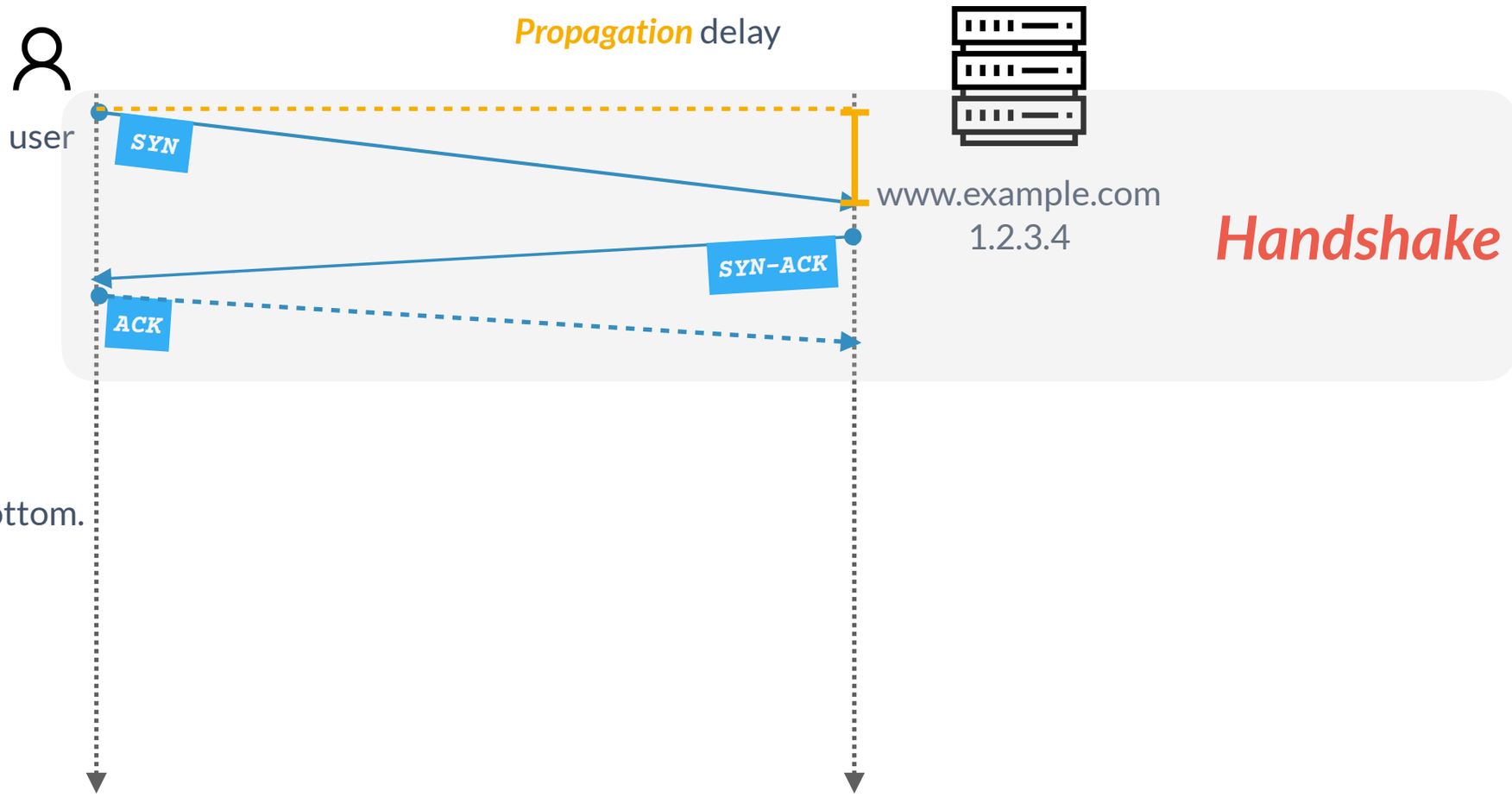


TCP “Handshake”

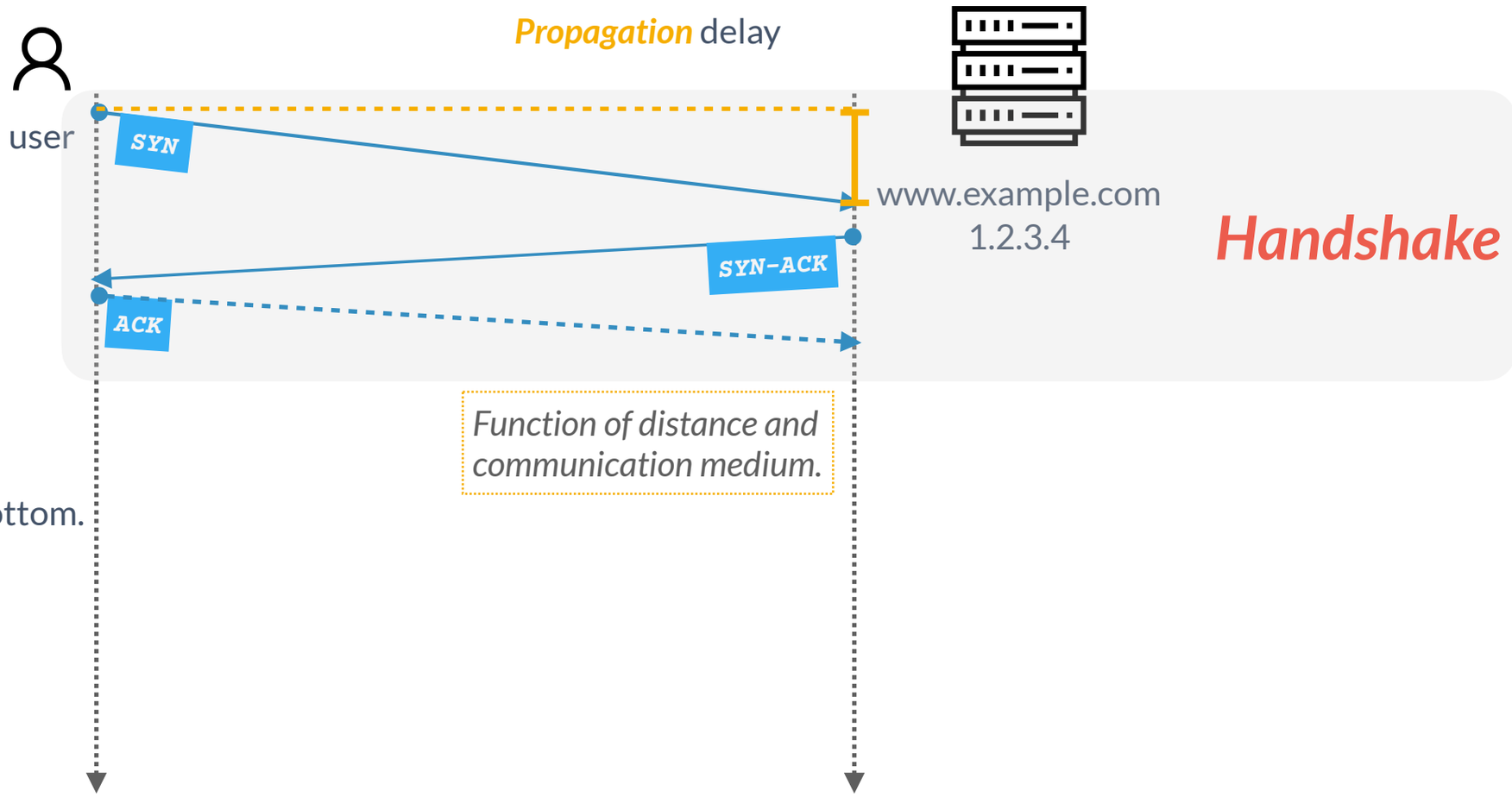


Time moves from top to bottom.

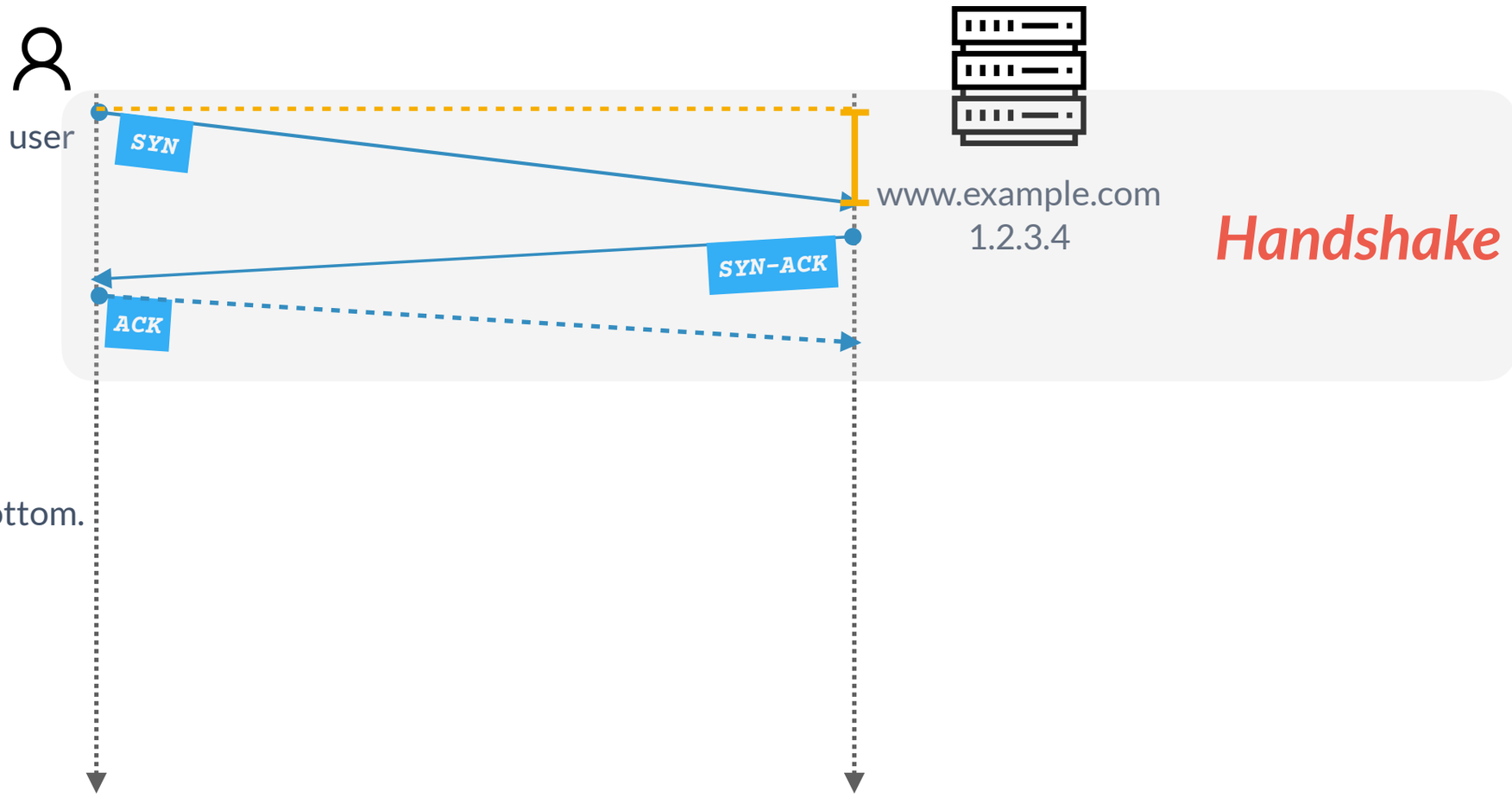
TCP “Handshake”



TCP “Handshake”

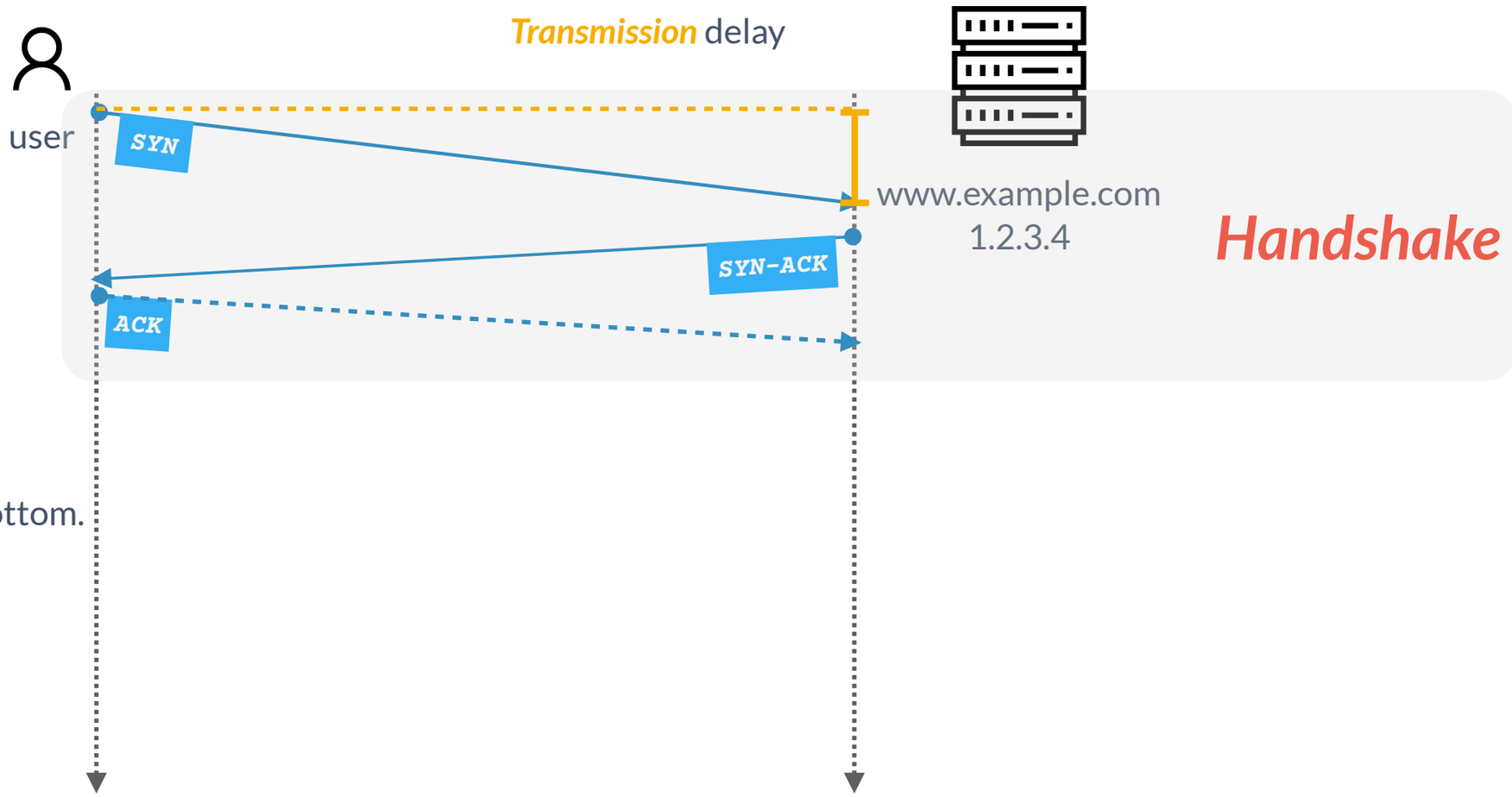


TCP “Handshake”



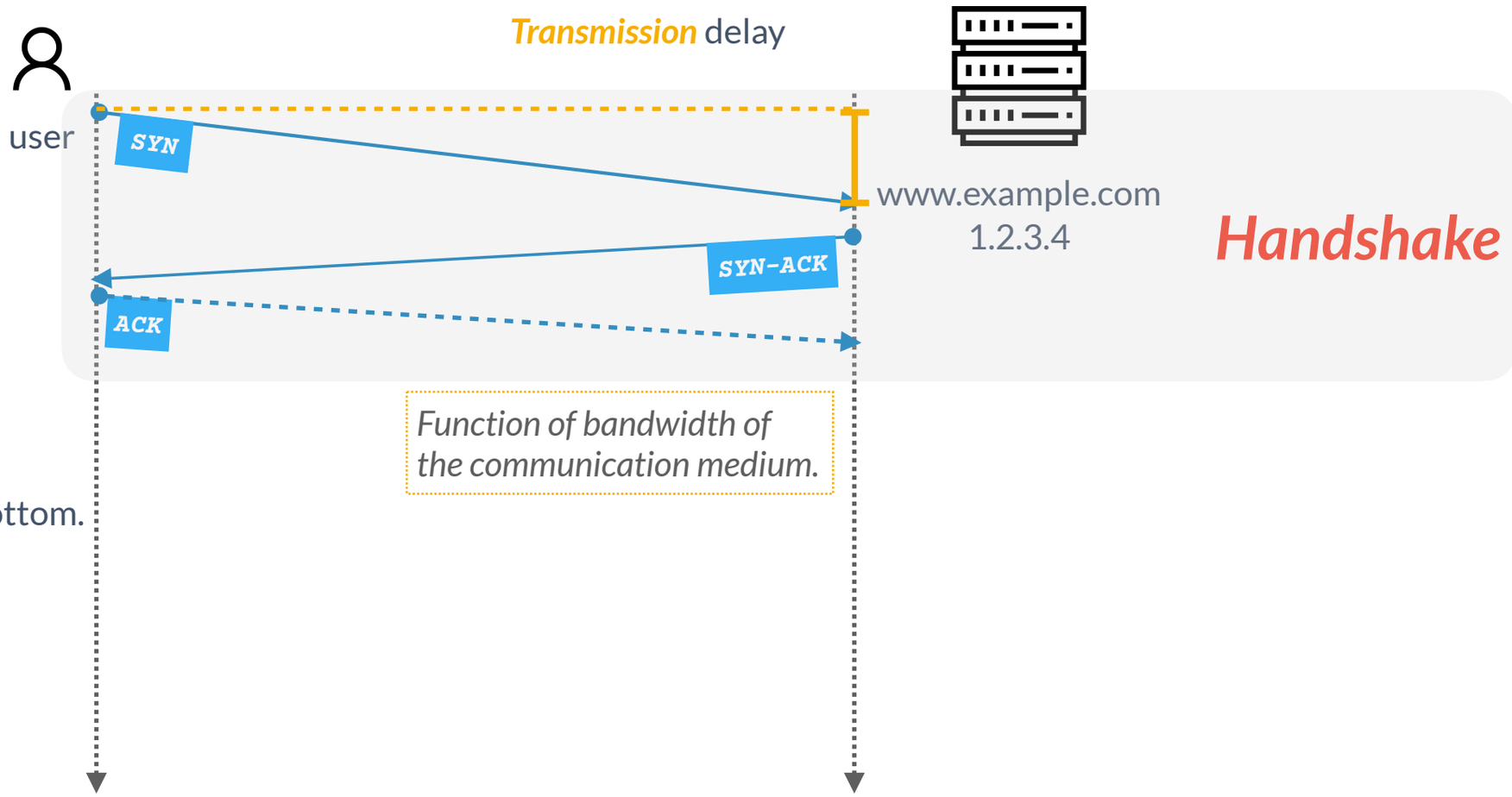
Time moves from top to bottom.

TCP “Handshake”

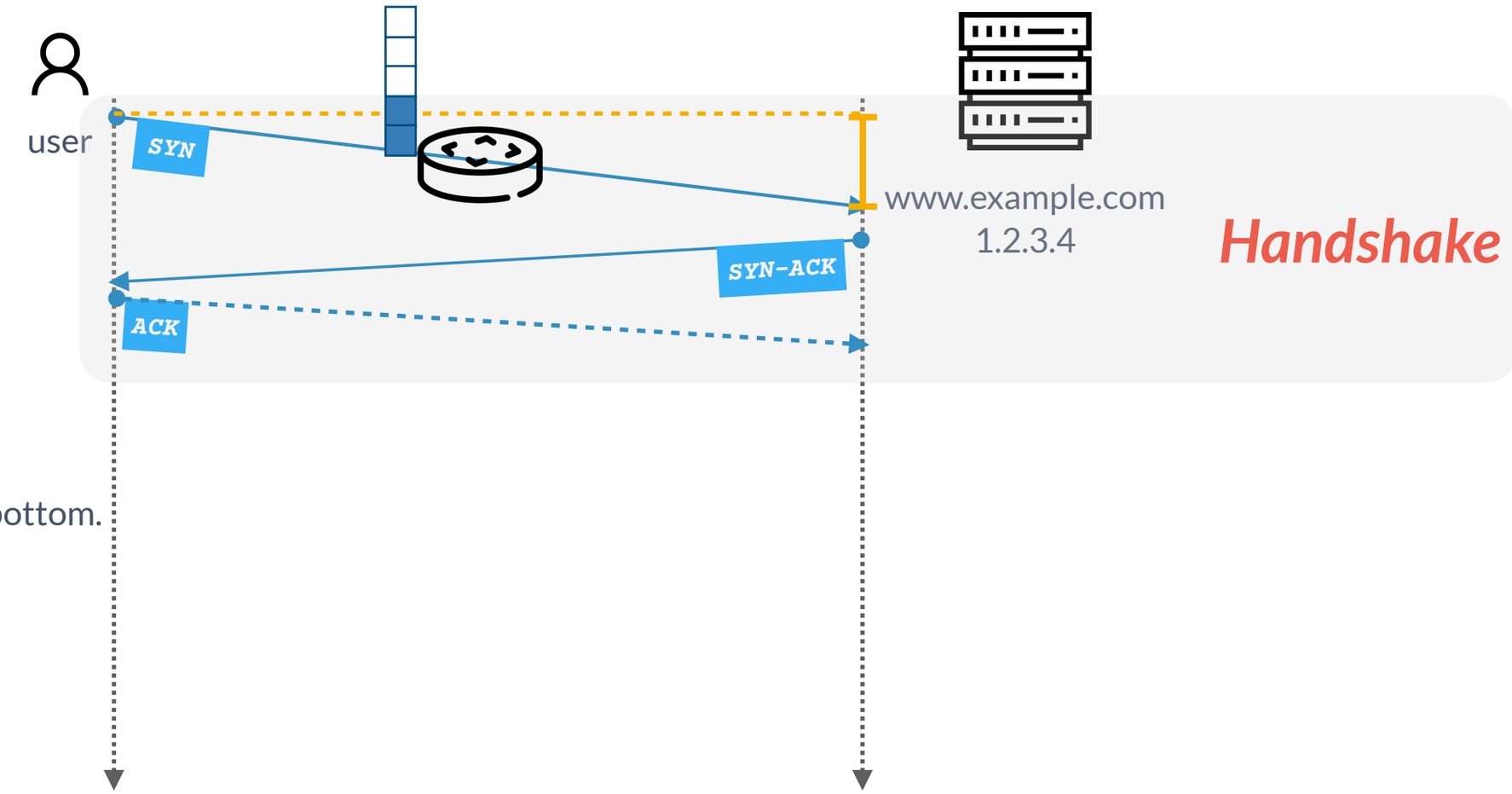


Time moves from top to bottom.

TCP “Handshake”

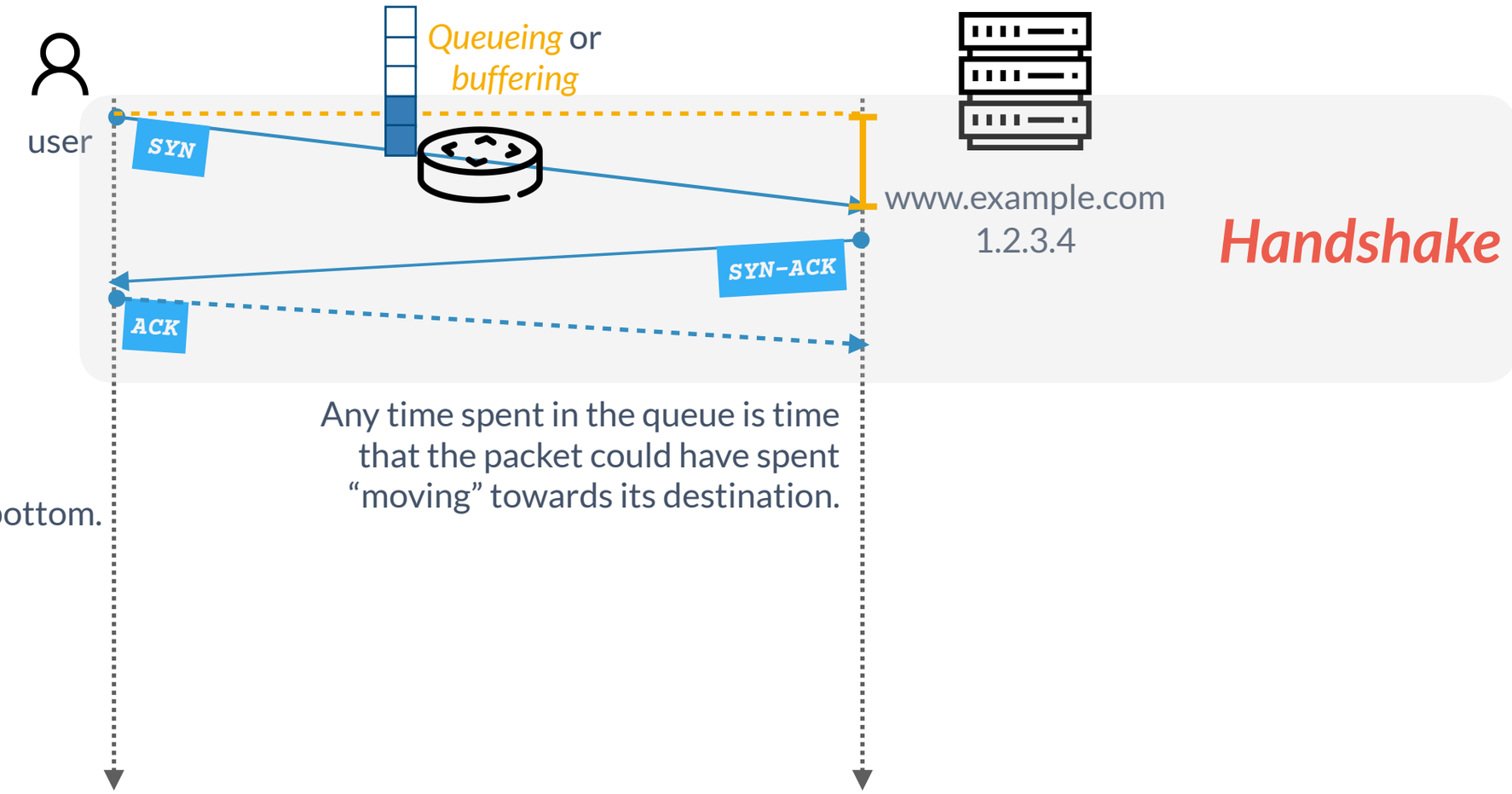


TCP “Handshake”

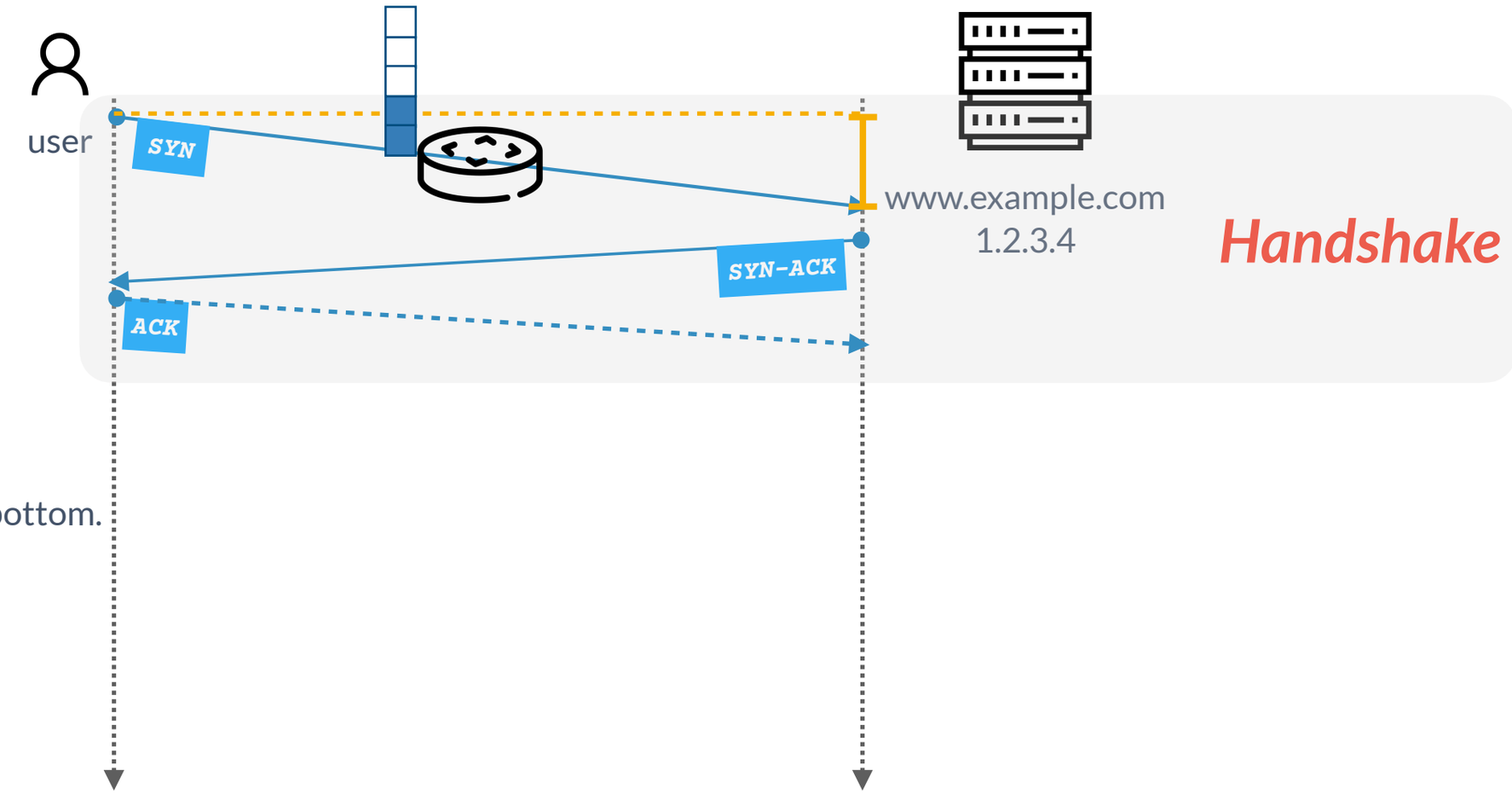


Time moves from top to bottom.

TCP “Handshake”

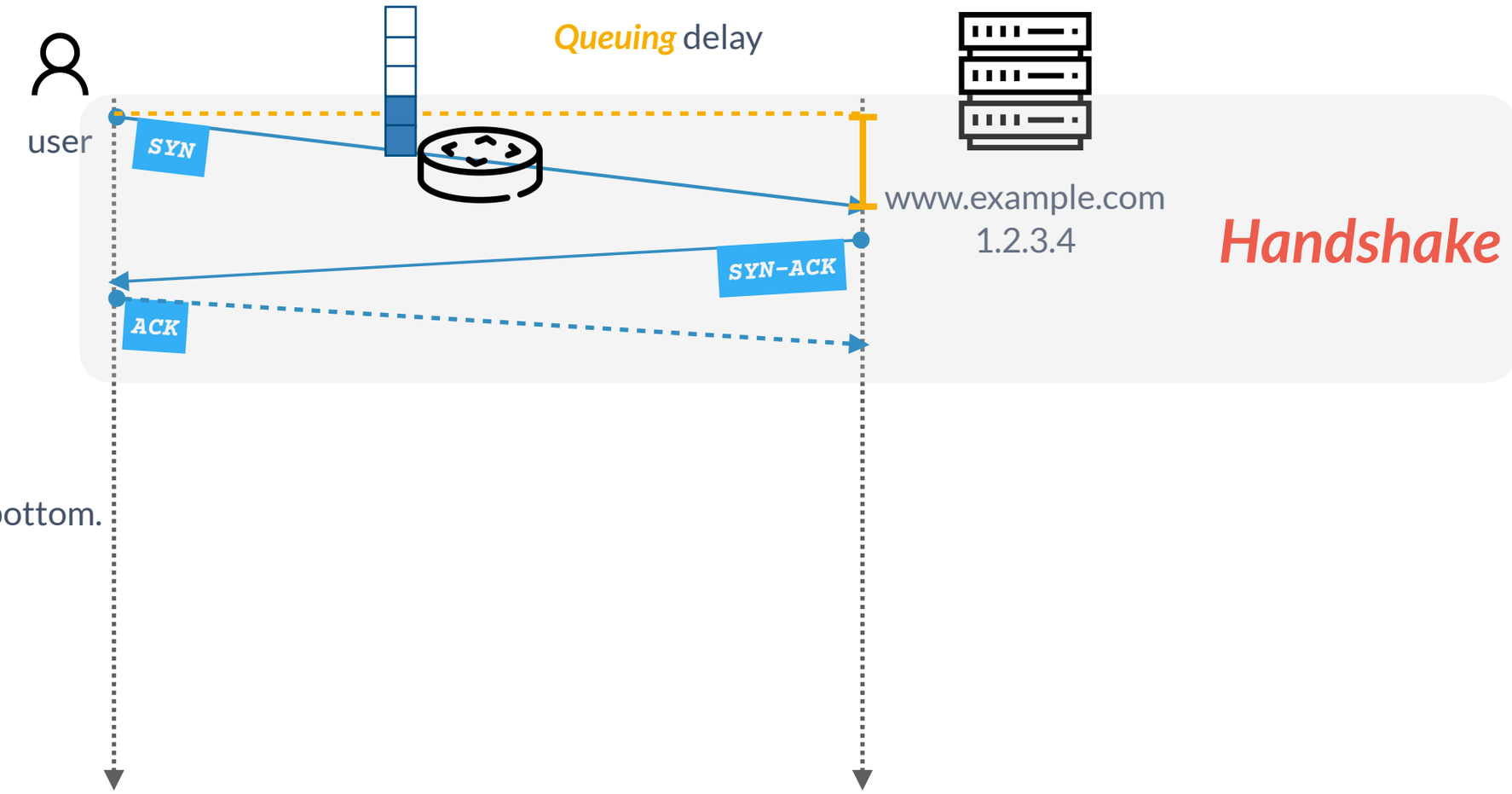


TCP “Handshake”



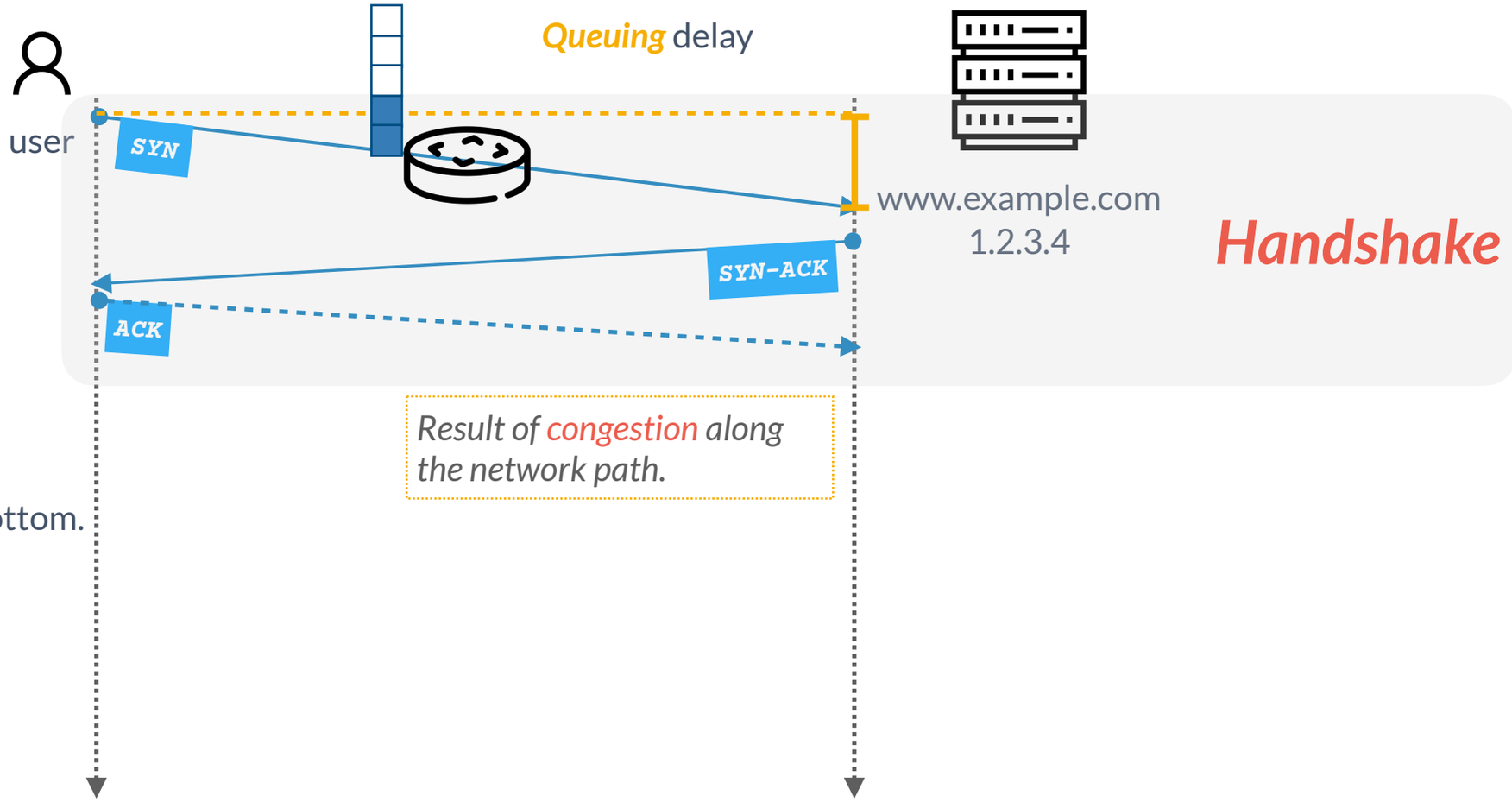
Time moves from top to bottom.

TCP “Handshake”



Time moves from top to bottom.

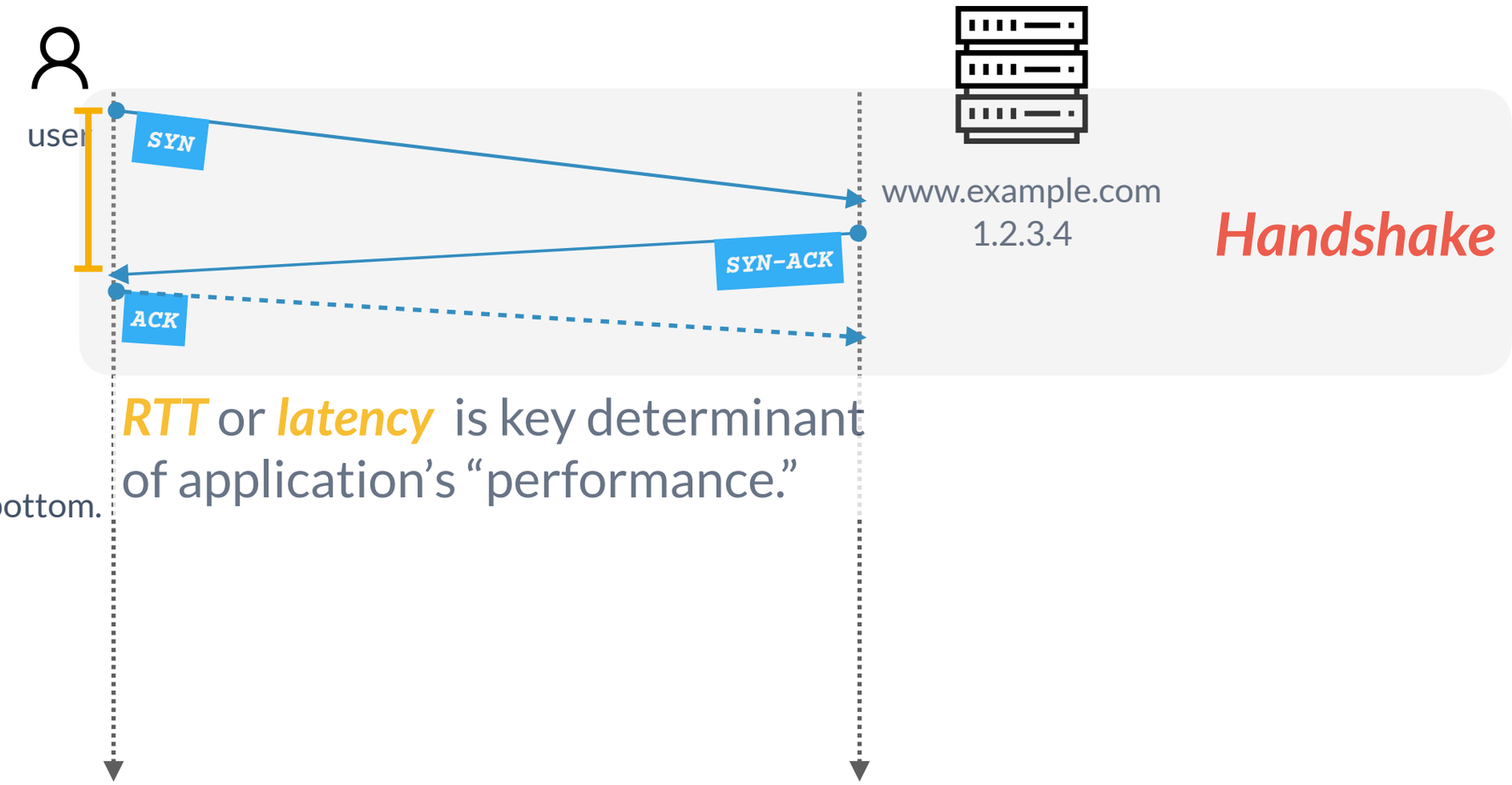
TCP “Handshake”



Time moves from top to bottom.

TCP “Handshake”

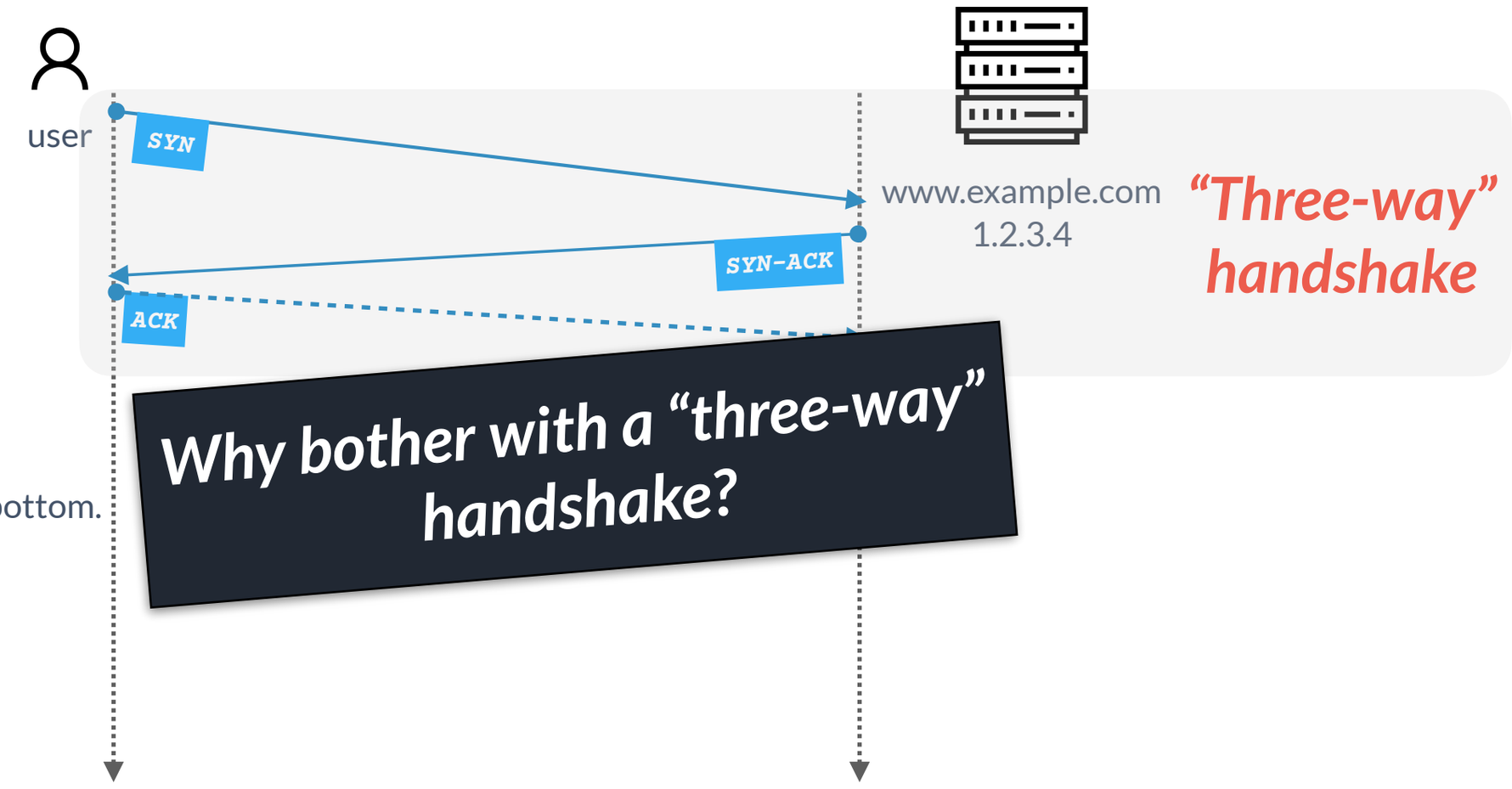
Round-trip time (RTT)



Time moves from top to bottom.

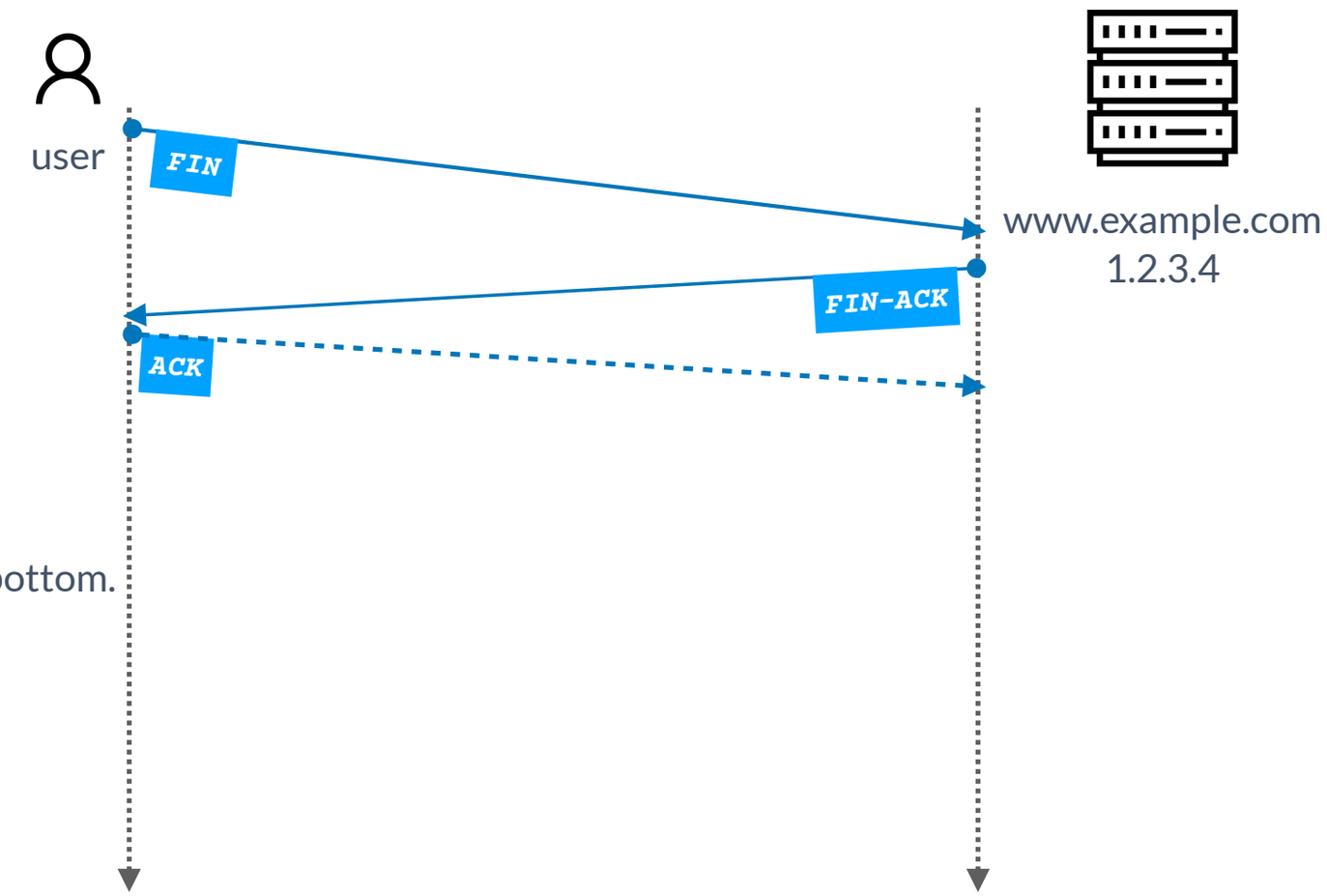
RTT or ***latency*** is key determinant of application’s “performance.”

TCP “Handshake”



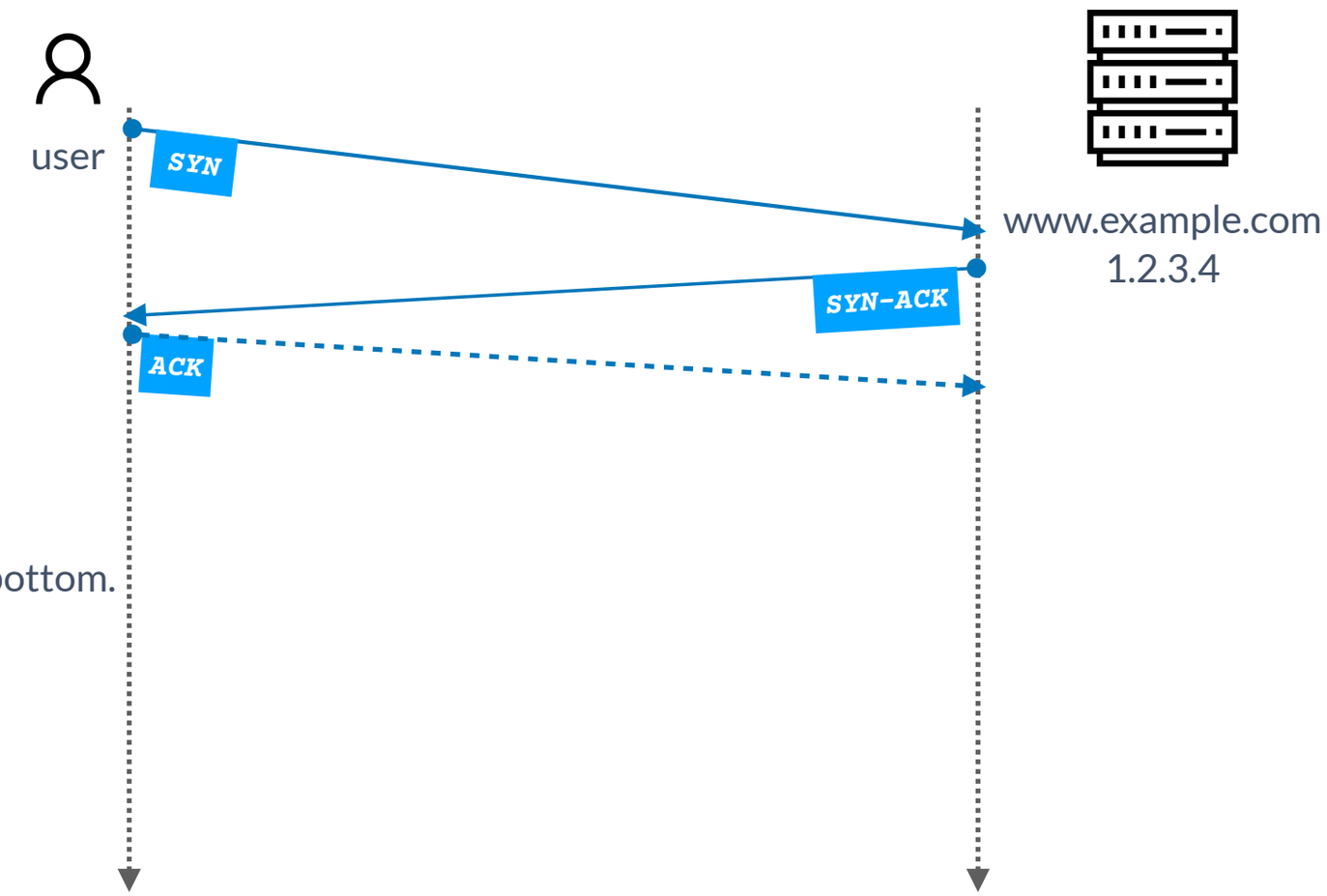
Time moves from top to bottom.

Connection “tear down”



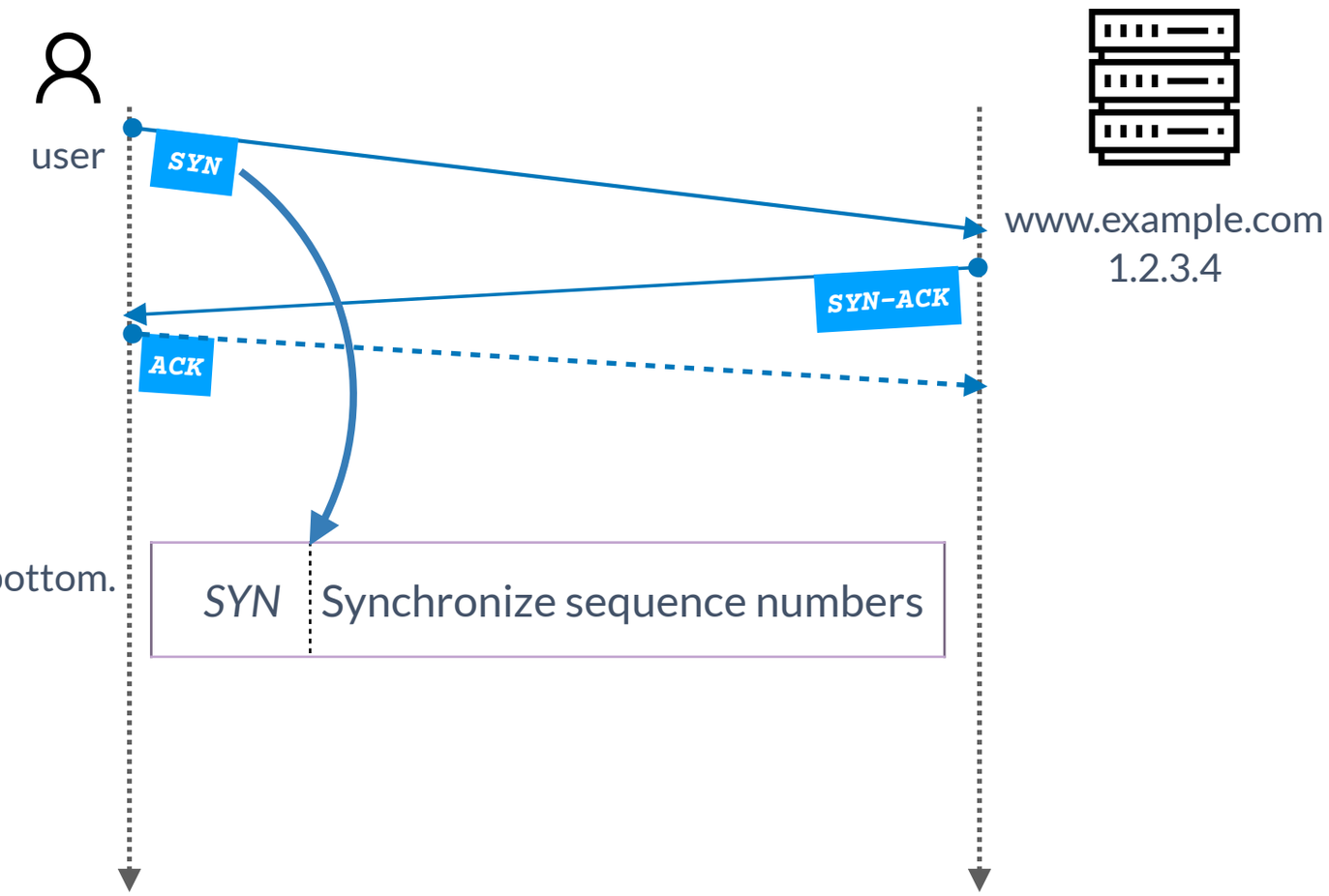
Time moves from top to bottom.

Stream of Bytes Service

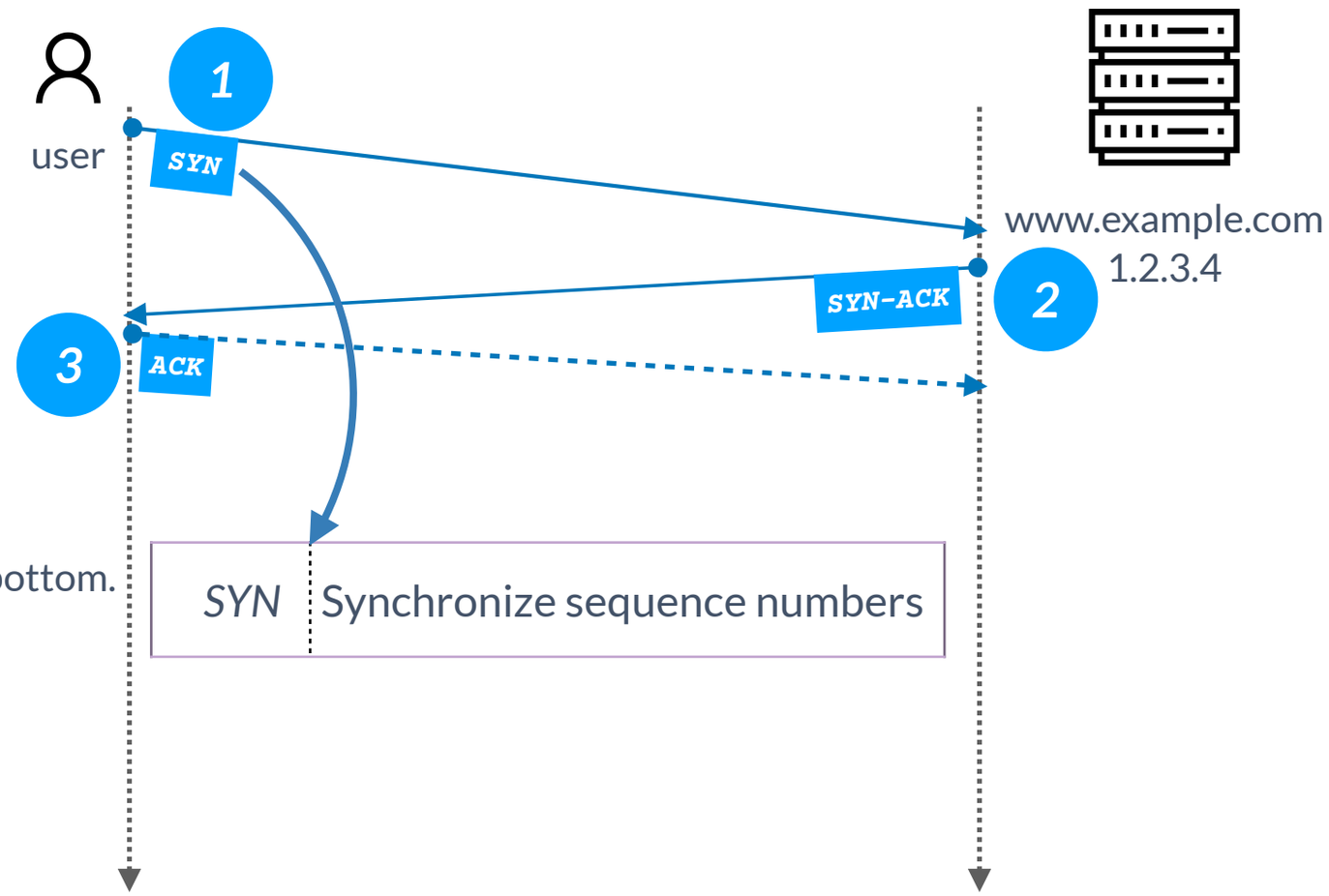


Time moves from top to bottom.

Stream of Bytes Service

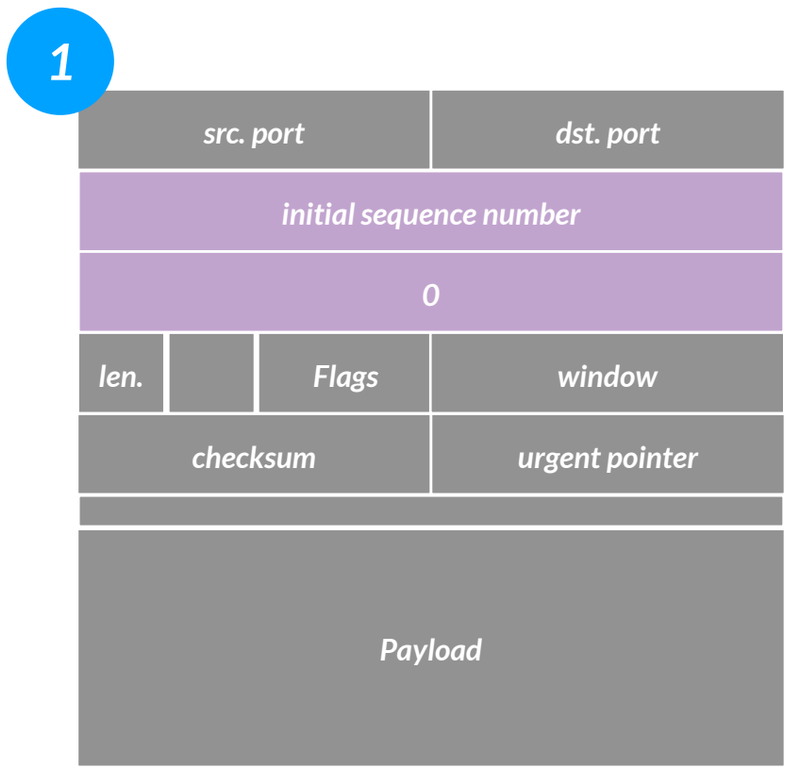


Stream of Bytes Service



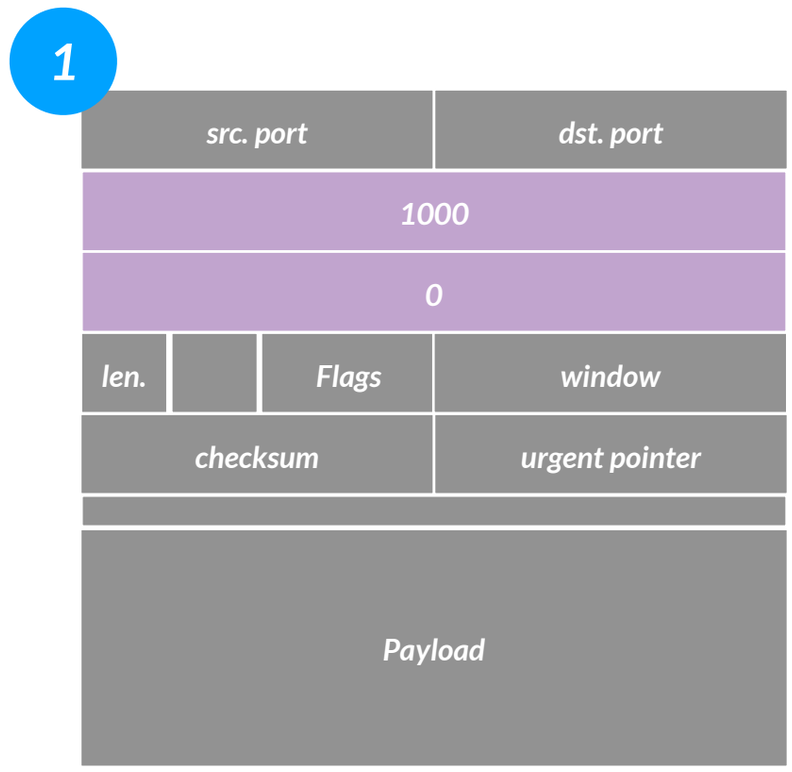
Stream of Bytes Service

- SYN
  - Sender's *initial sequence (SEQ)* number



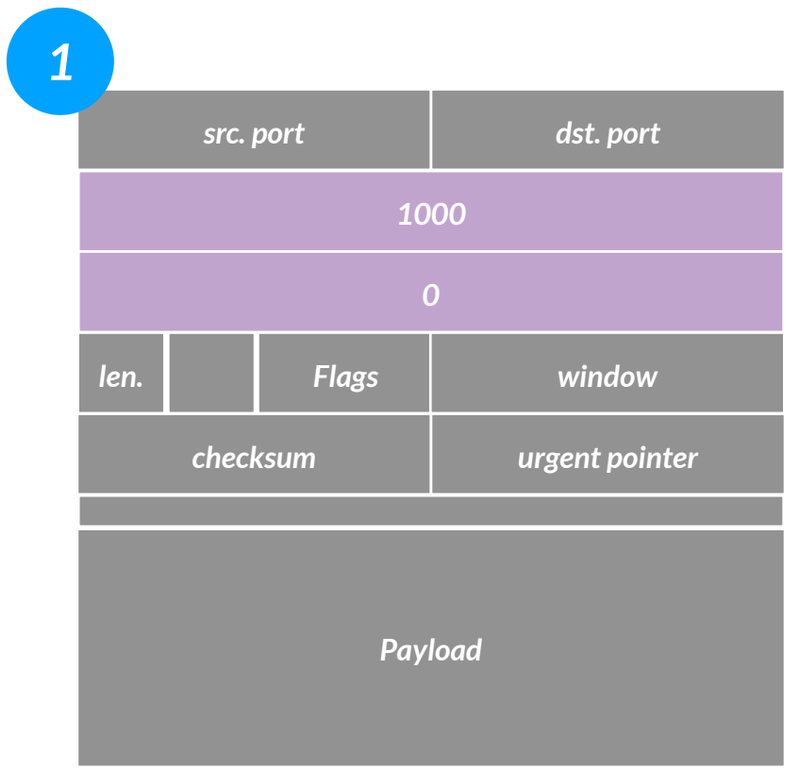
Stream of Bytes Service

- SYN
  - Sender's *initial* SEQ number
  - Pick a *random* number



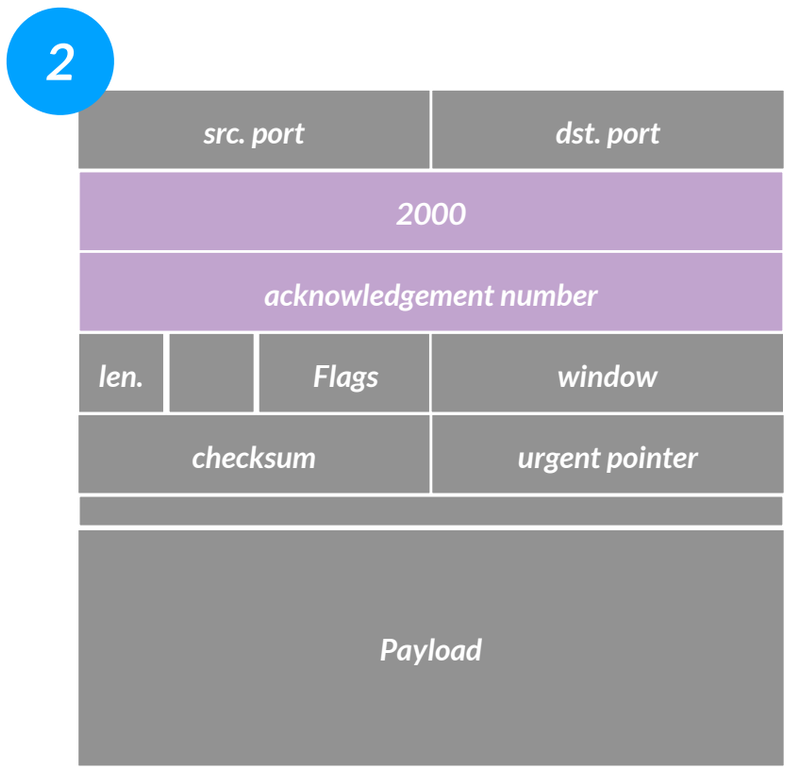
Stream of Bytes Service

- SYN
  - Sender's *initial* SEQ number
  - Pick a *random* number
  - Set *acknowledgement (ACK)* number to *zero*



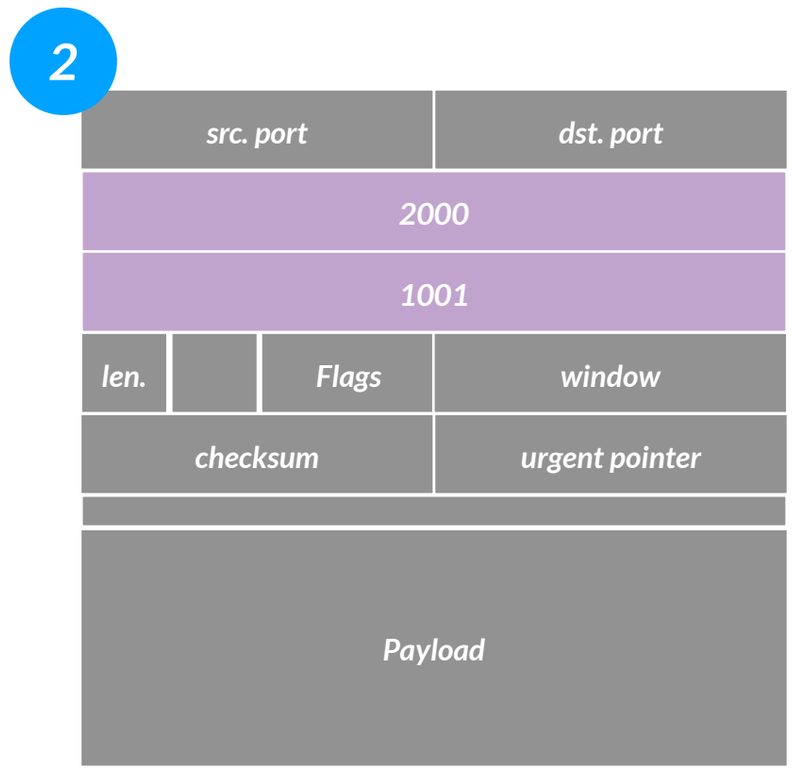
Stream of Bytes Service

- SYN-ACK
  - Receiver's *random* initial SEQ number



Stream of Bytes Service

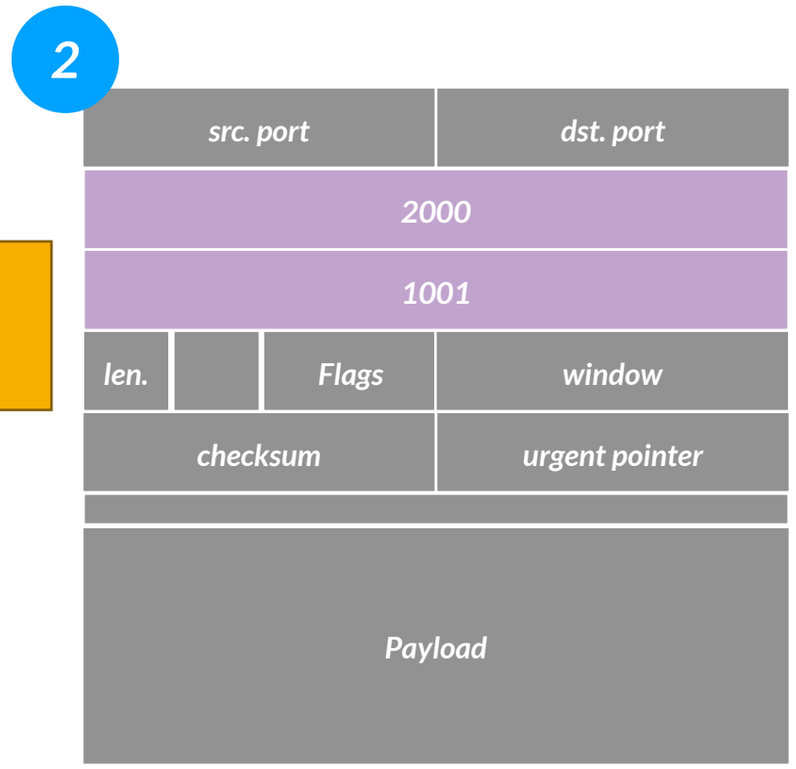
- SYN-ACK
  - Receiver's *random* initial SEQ number
  - Set ACK number to *Sender's SEQ number + 1*



Stream of Bytes Service

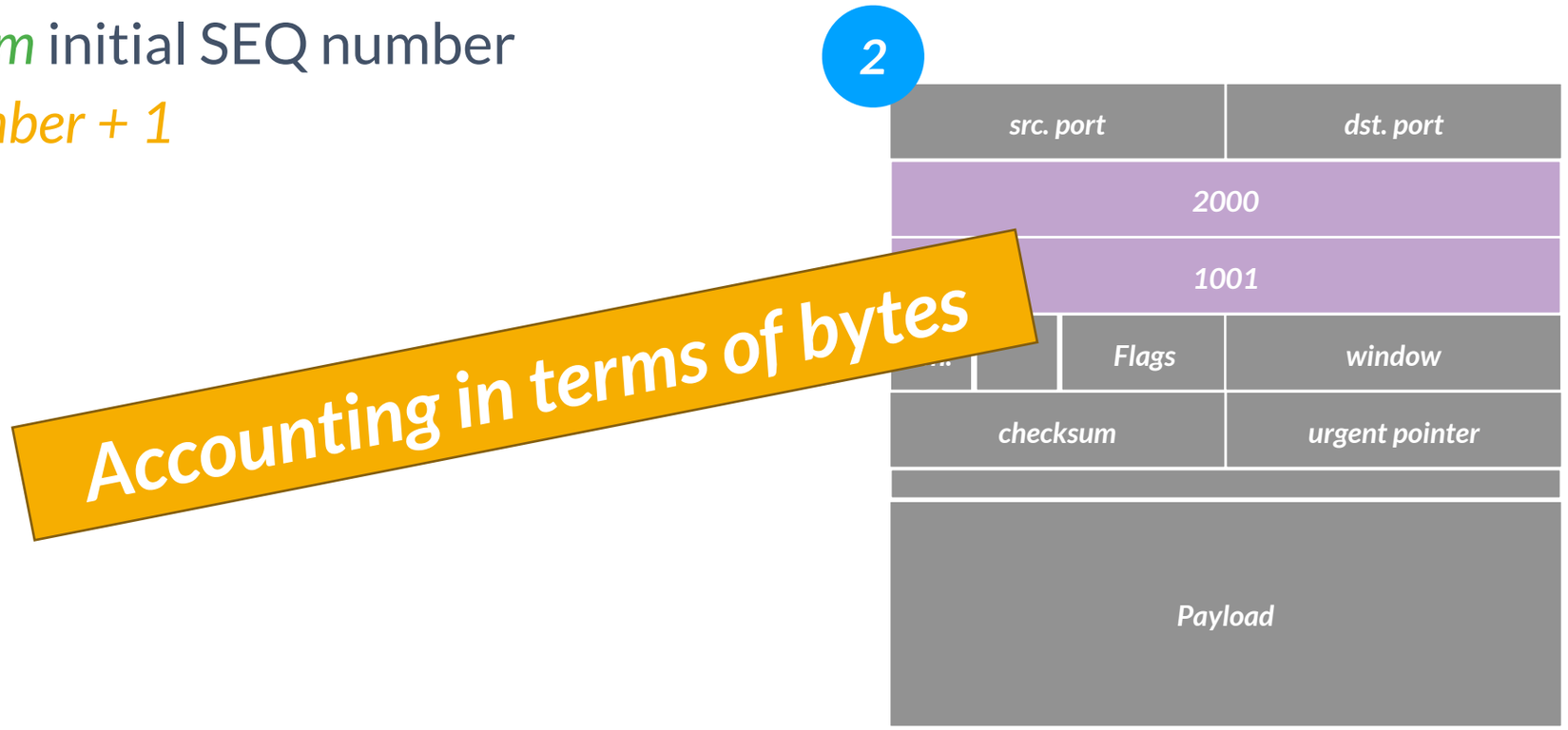
- SYN-ACK
  - Receiver's *random* initial SEQ number
  - *Sender's SEQ number + 1*

Indicating the next byte that the receiver expects to receive



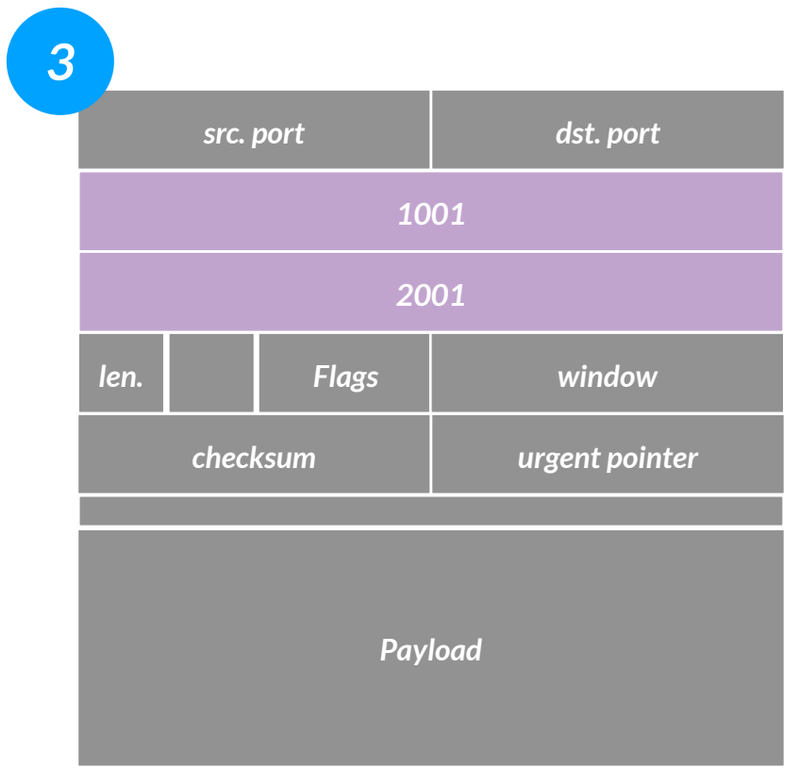
Stream of Bytes Service

- SYN-ACK
  - Receiver's *random* initial SEQ number
  - *Sender's SEQ number + 1*



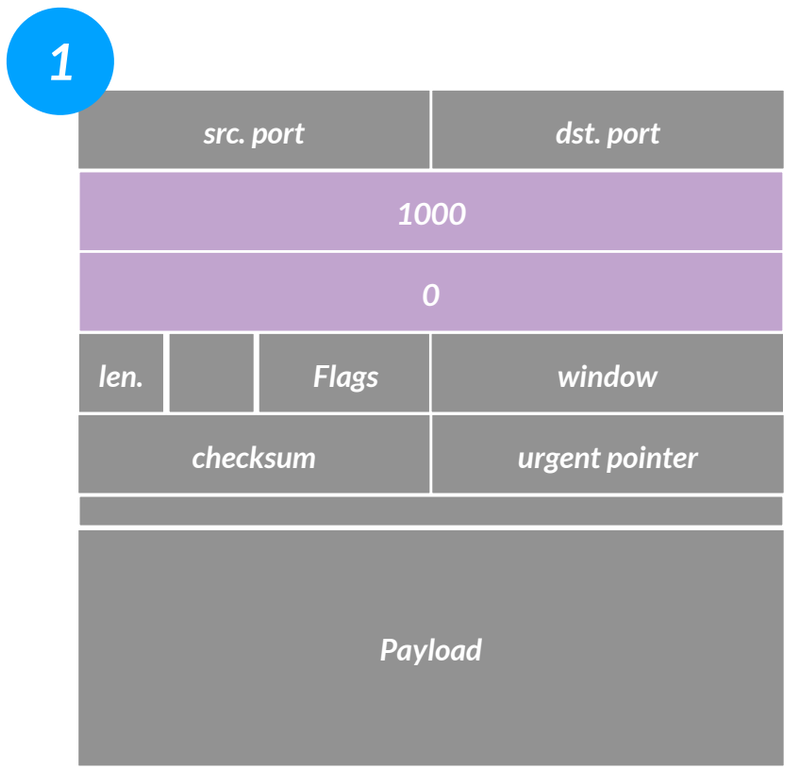
Stream of Bytes Service

- ACK
  - Next SEQ number is the *starting byte* of the sequence being sent
  - Next ACK number is *receiver's SEQ number + 1*



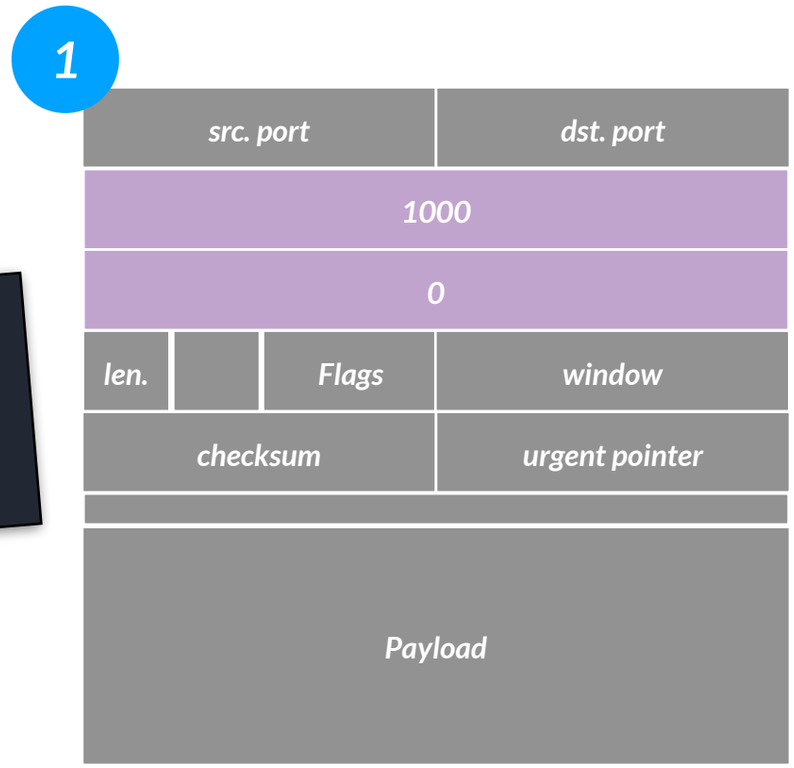
Stream of Bytes Service

- SYN
  - Sender's *initial* SEQ number
  - Pick a *random* number
  - Set *acknowledgement (ACK)* number to *zero*
- SYN-ACK
  - Receiver's *random* initial SEQ number
  - Set ACK number to *Sender's SEQ number + 1*



Stream of Bytes Service

- SYN
  - Sender's *initial* SEQ number
  - Pick a *random* number
  - Set *acknowledgement (ACK)* number to *zero*

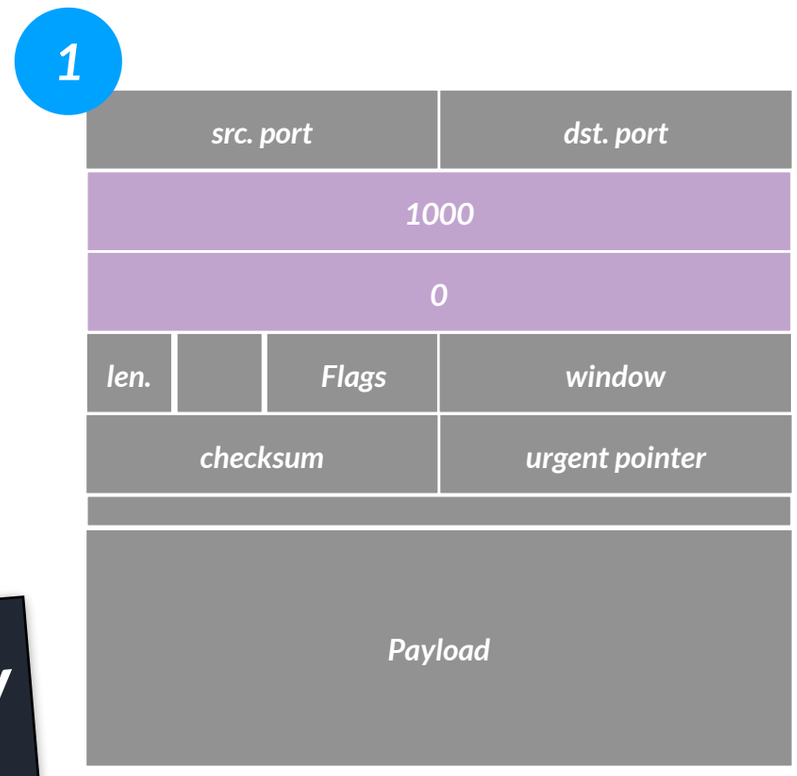


- SYN-ACK
  - Receiver's *random* initial S
  - Set ACK number to *Sende* *number + 1*

Why *random*?

Stream of Bytes Service

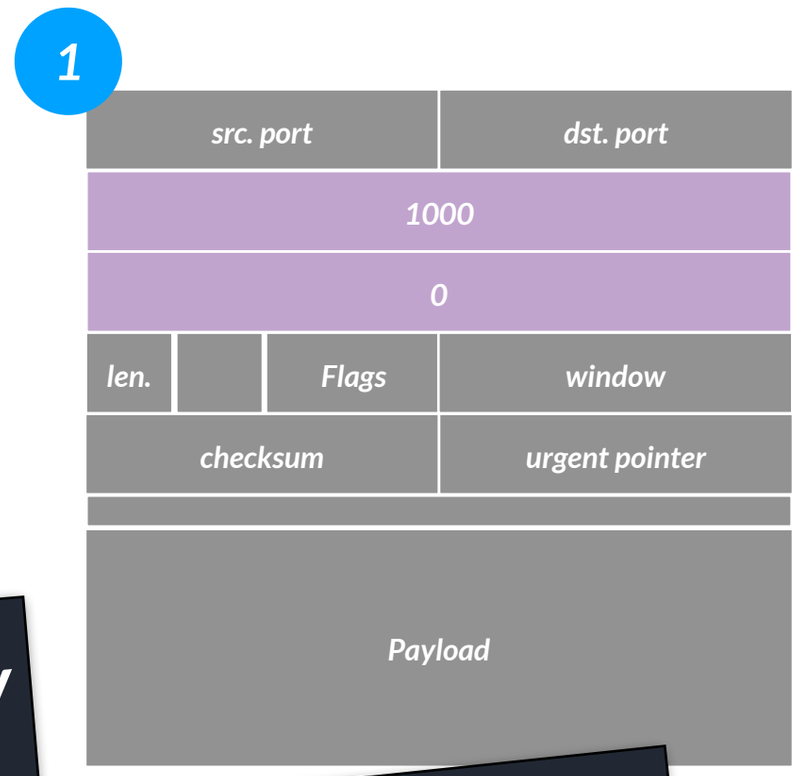
- SYN
  - Sender's *initial* SEQ number
  - Pick a *random* number
  - Set *acknowledgement (ACK)* number to *zero*
- SYN-ACK
  - Receiver's *random* initial SEQ number
  - Set ACK number to *Sender's SEQ number + 1*



Did sender send any payload?

Stream of Bytes Service

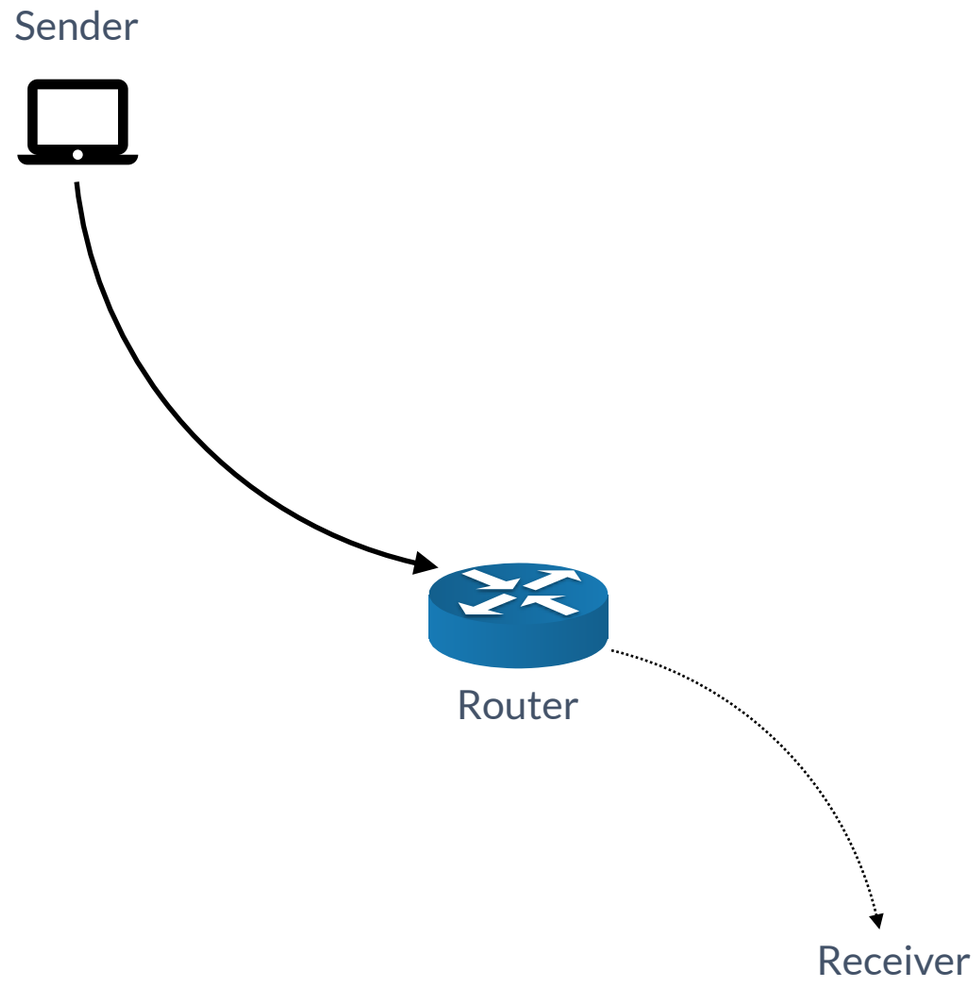
- SYN
  - Sender's *initial* SEQ number
  - Pick a *random* number
  - Set *acknowledgement (ACK)* number to *zero*
- SYN-ACK
  - Receiver's *random* initial SEQ number
  - Set ACK number to *Sender's SEQ number + 1*



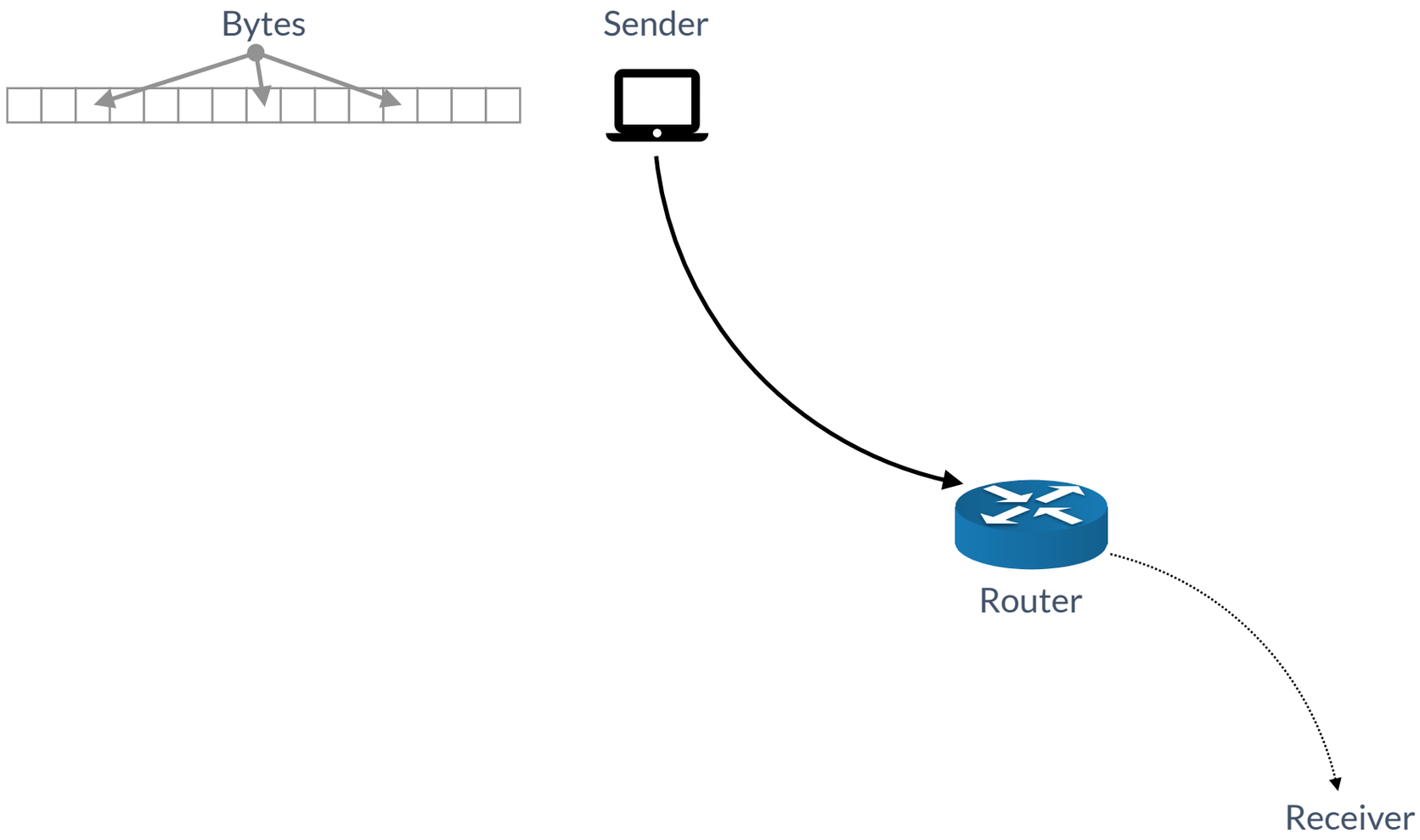
Did sender send any payload?

Can the sender send payload in the initial SYN packet?

Sliding Window

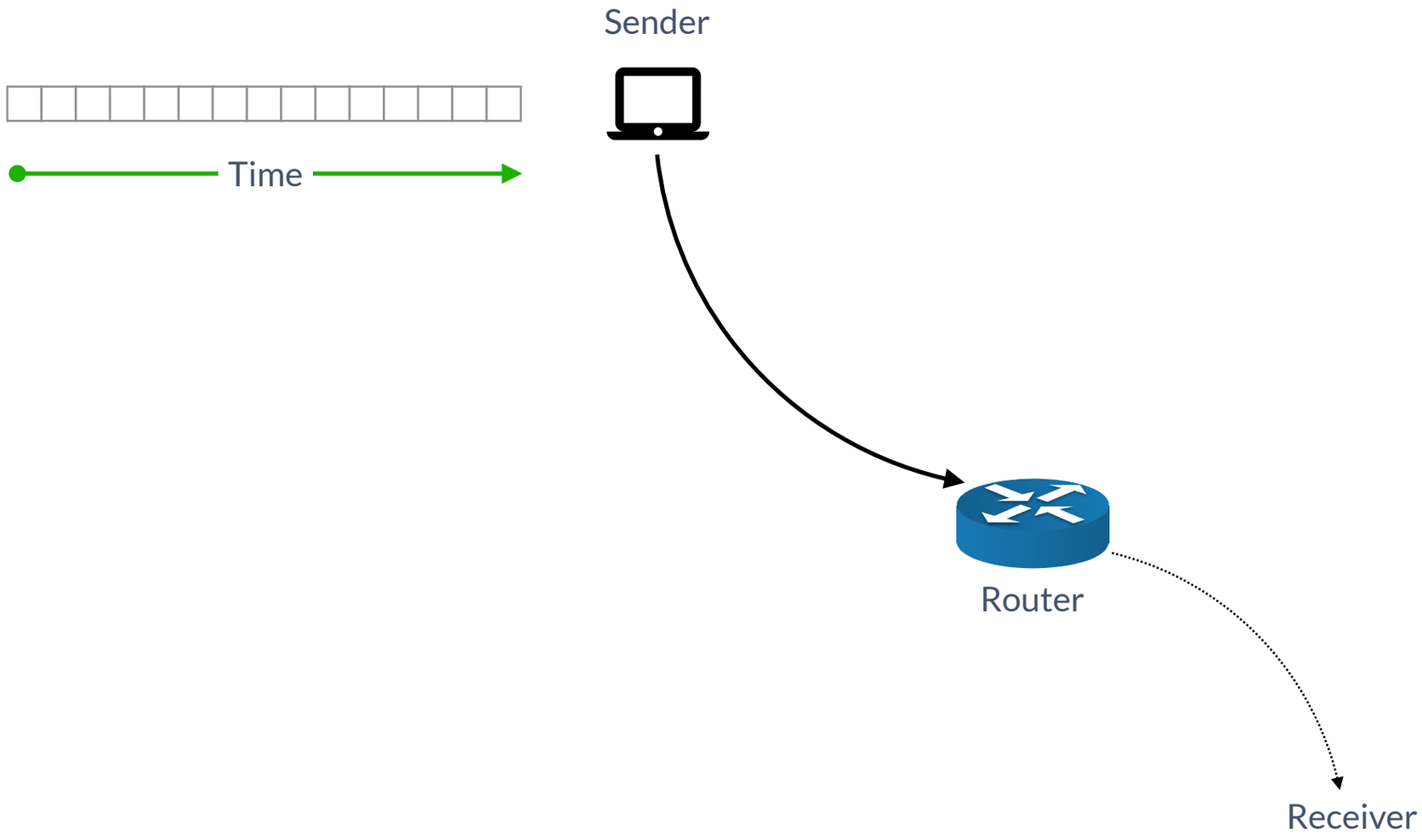


Sliding Window

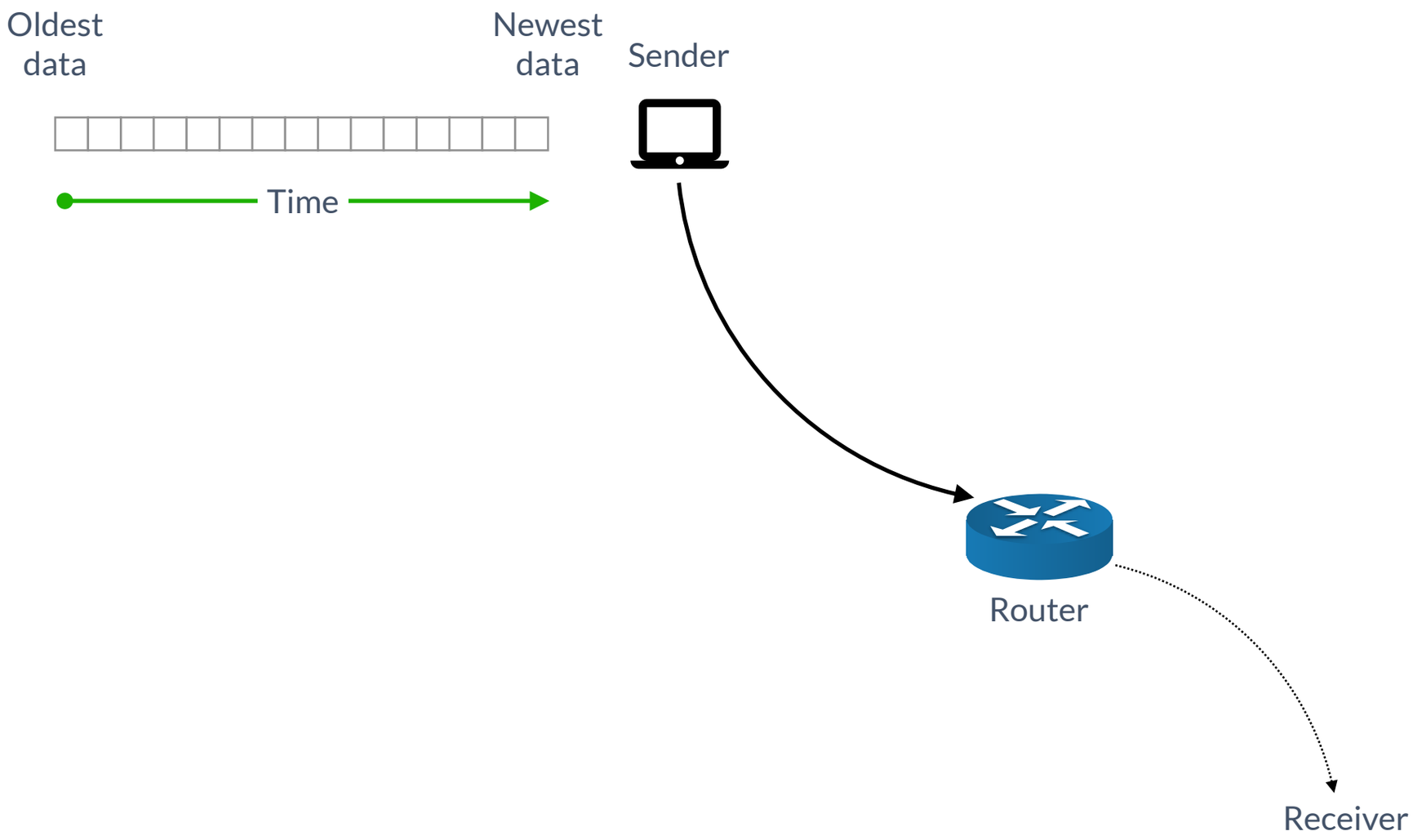


Window limits are in terms of bytes, but we will assume segments or packets for simplifying the explanation.

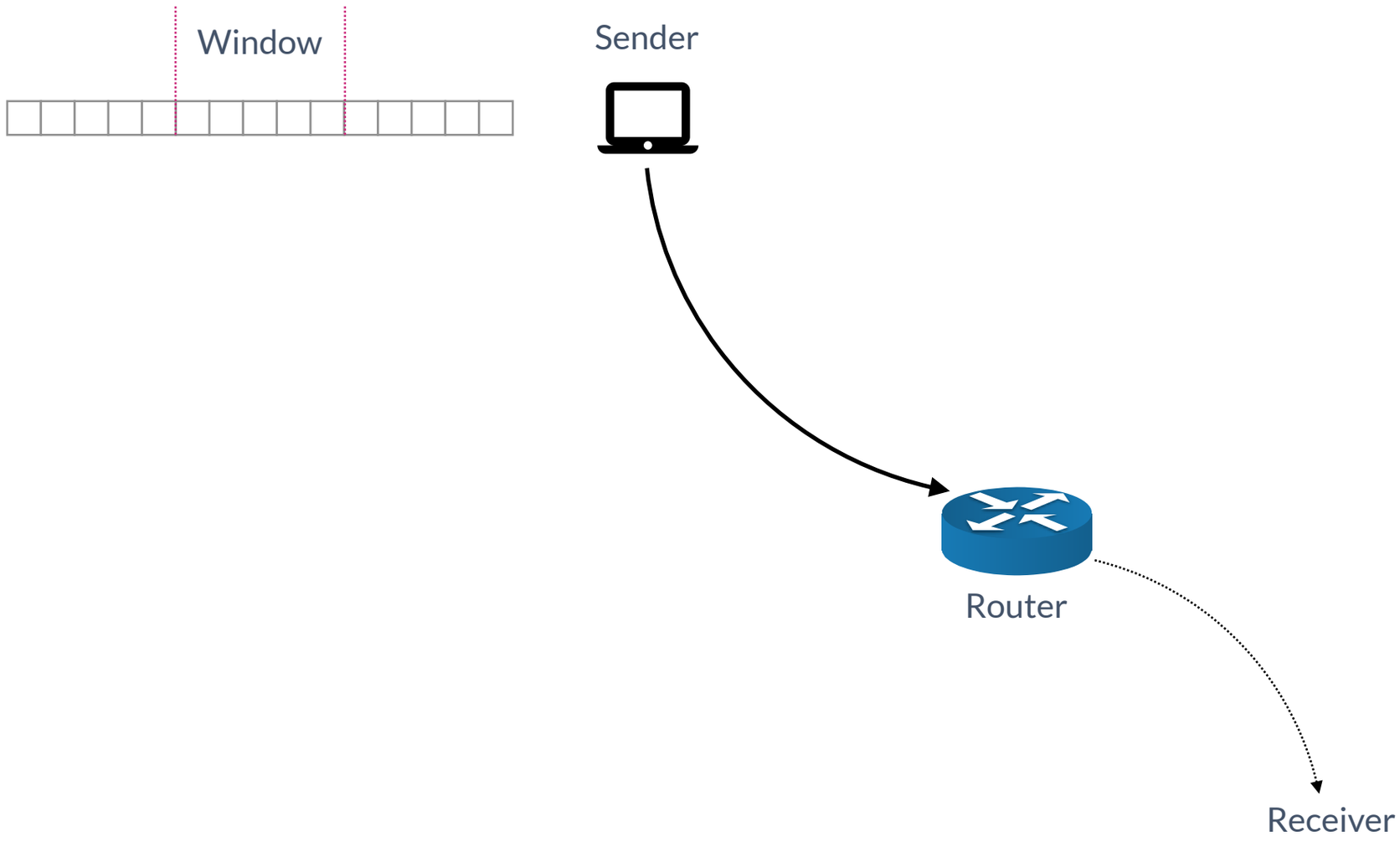
Sliding Window



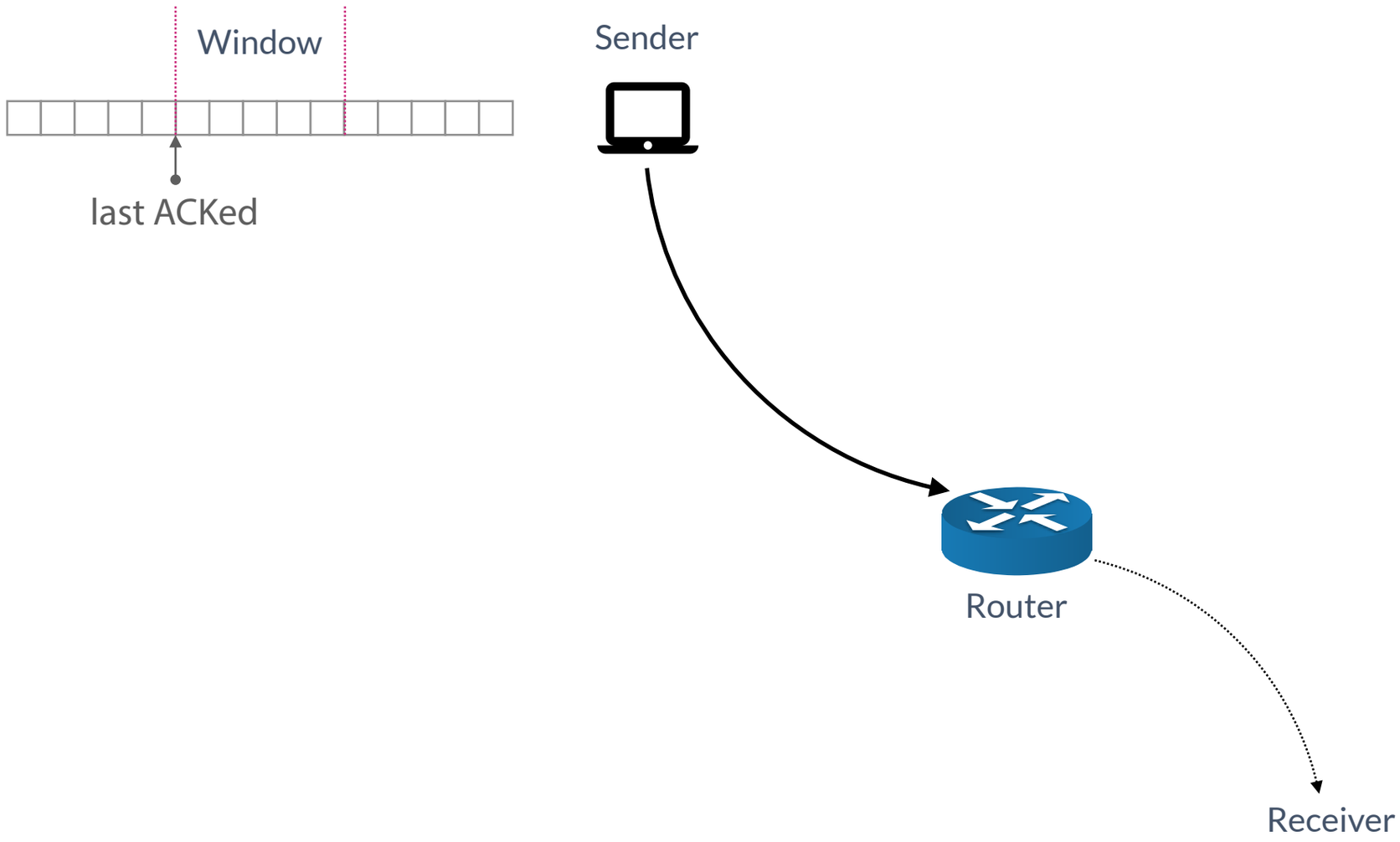
Sliding Window



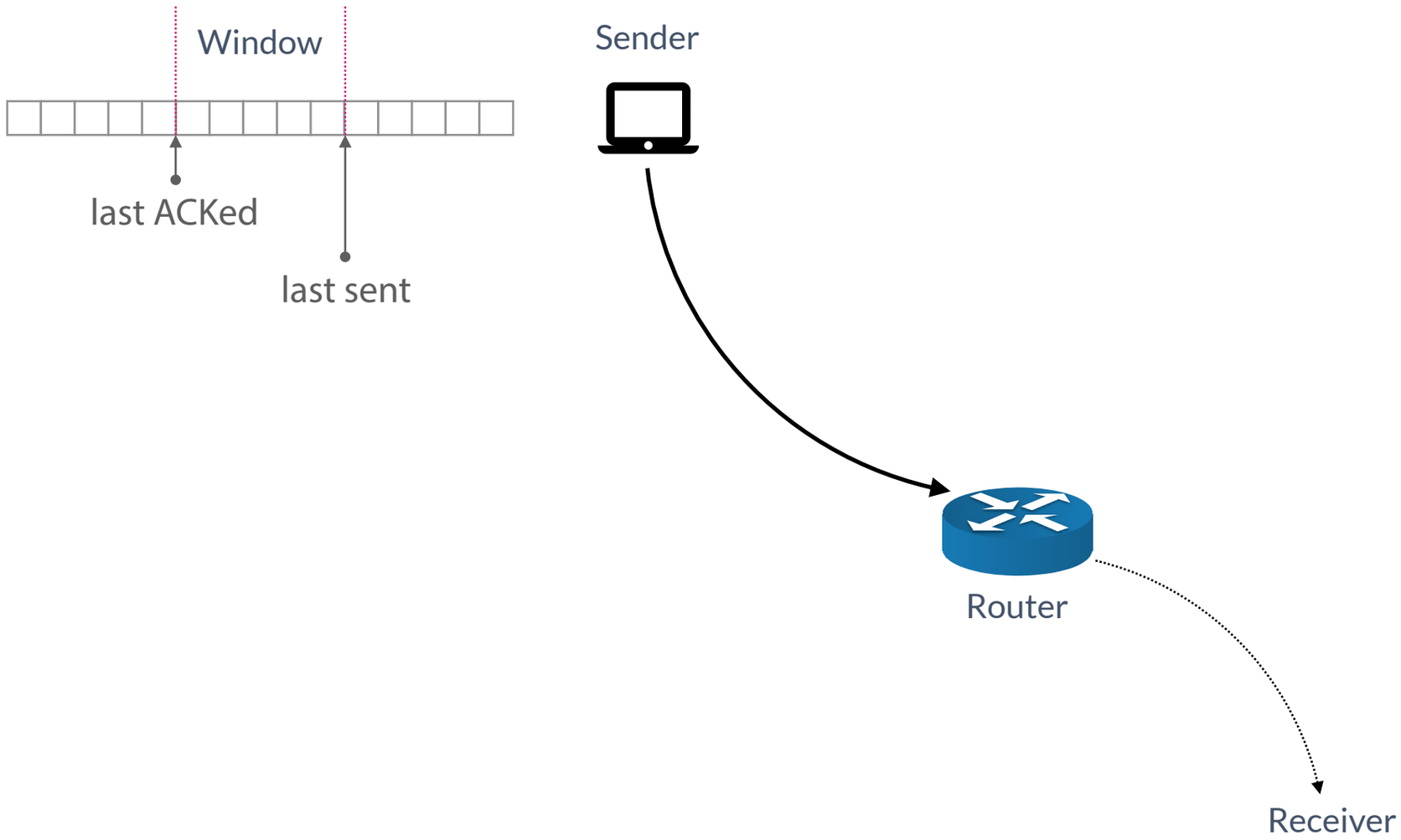
Sliding Window



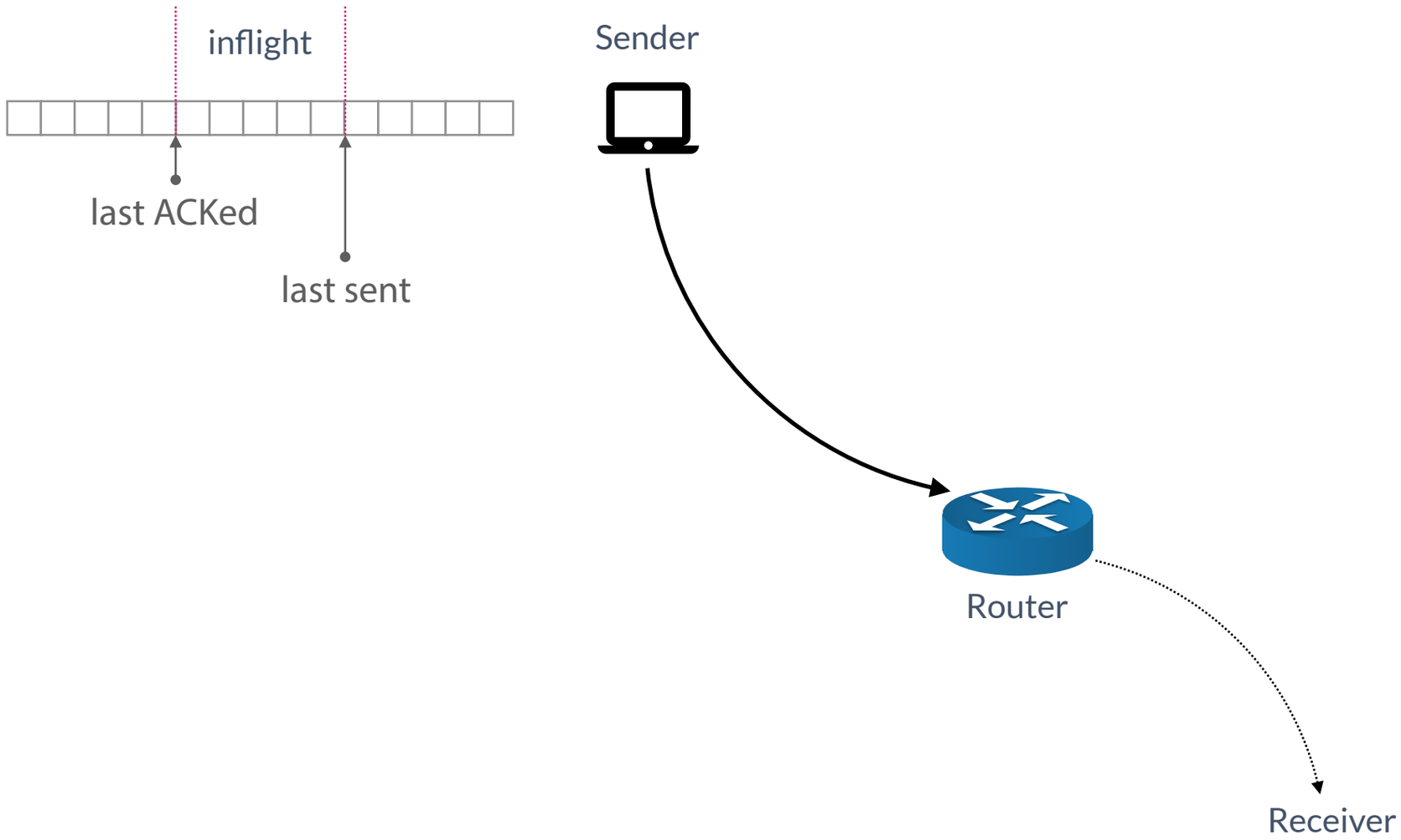
Sliding Window



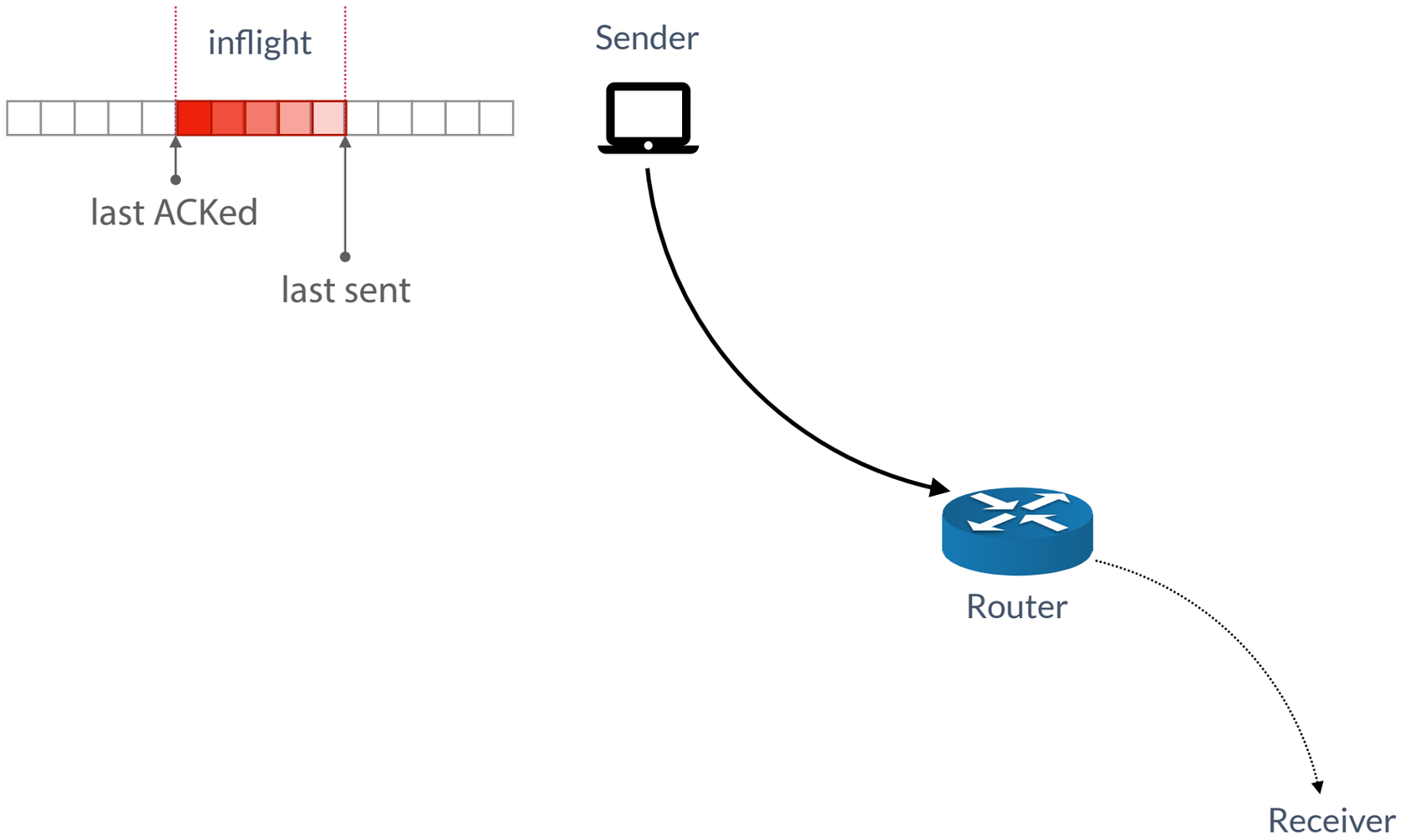
Sliding Window



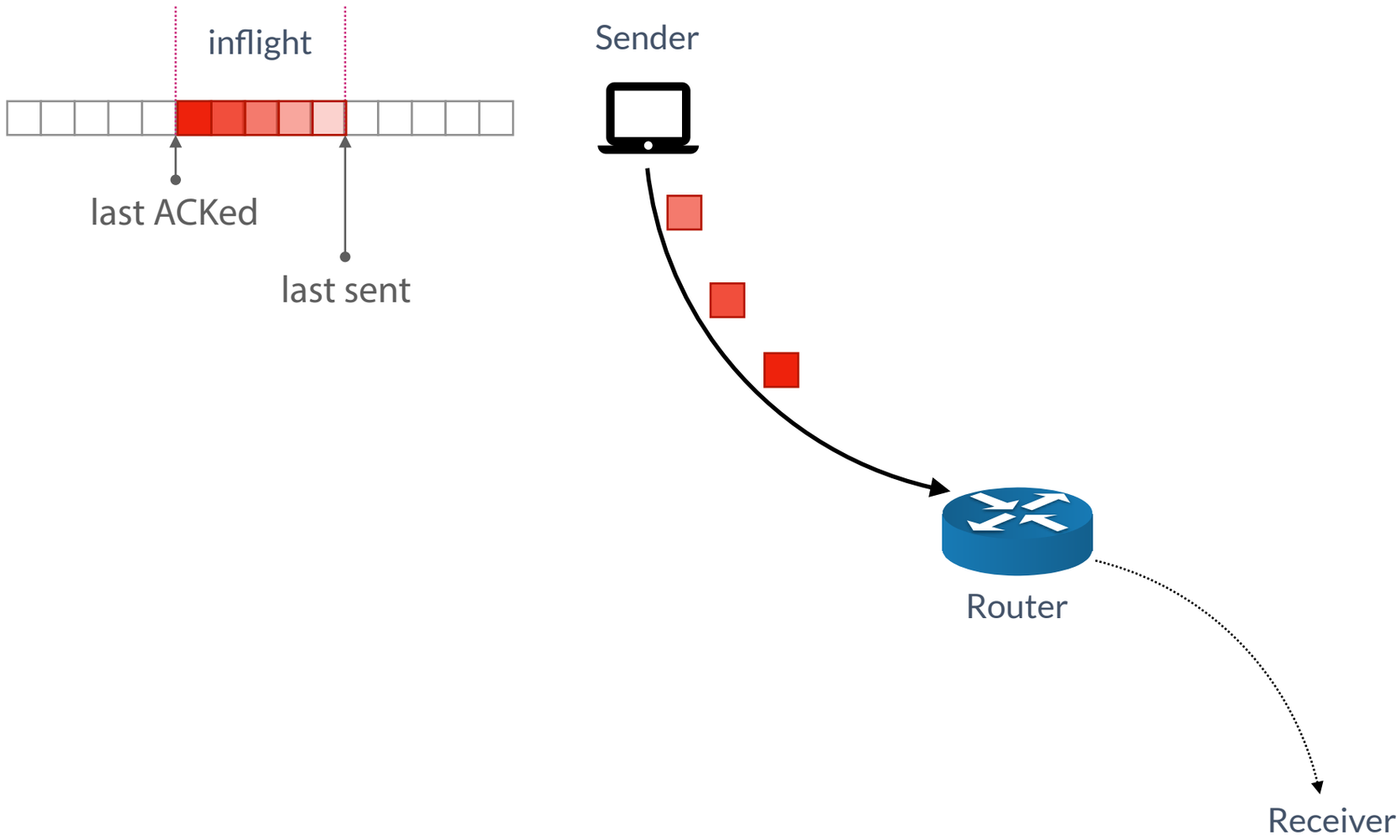
Sliding Window



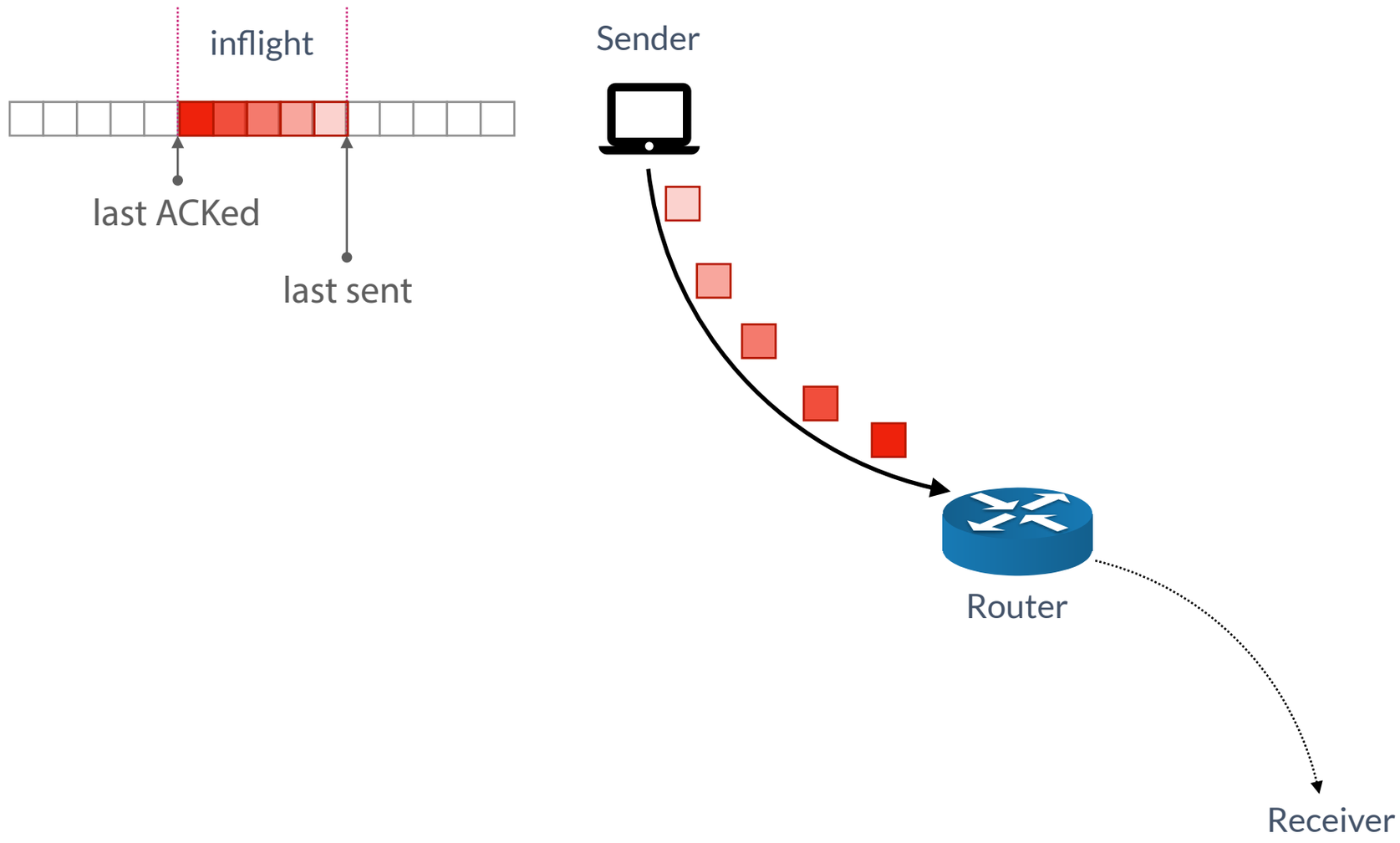
Sliding Window



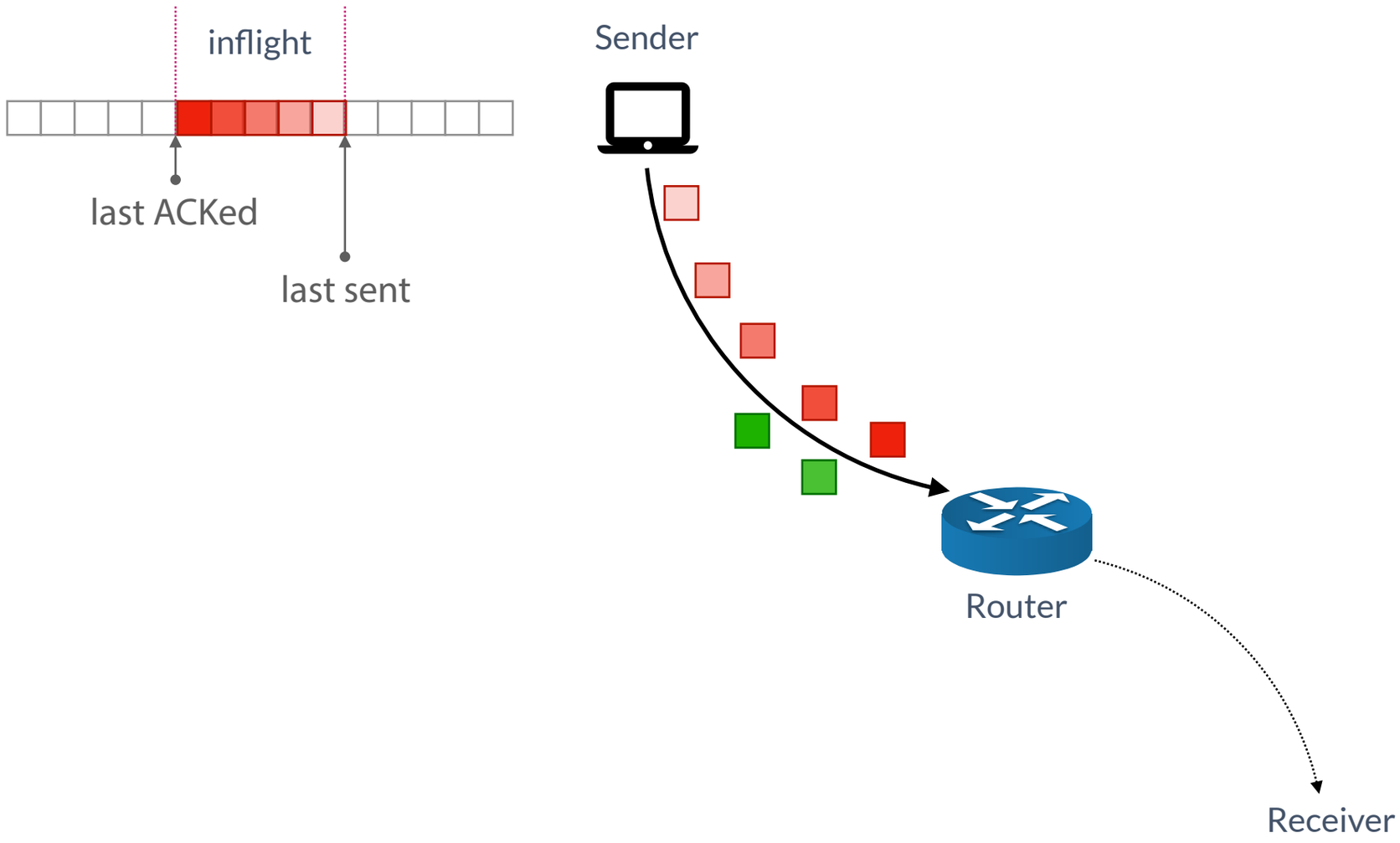
Sliding Window



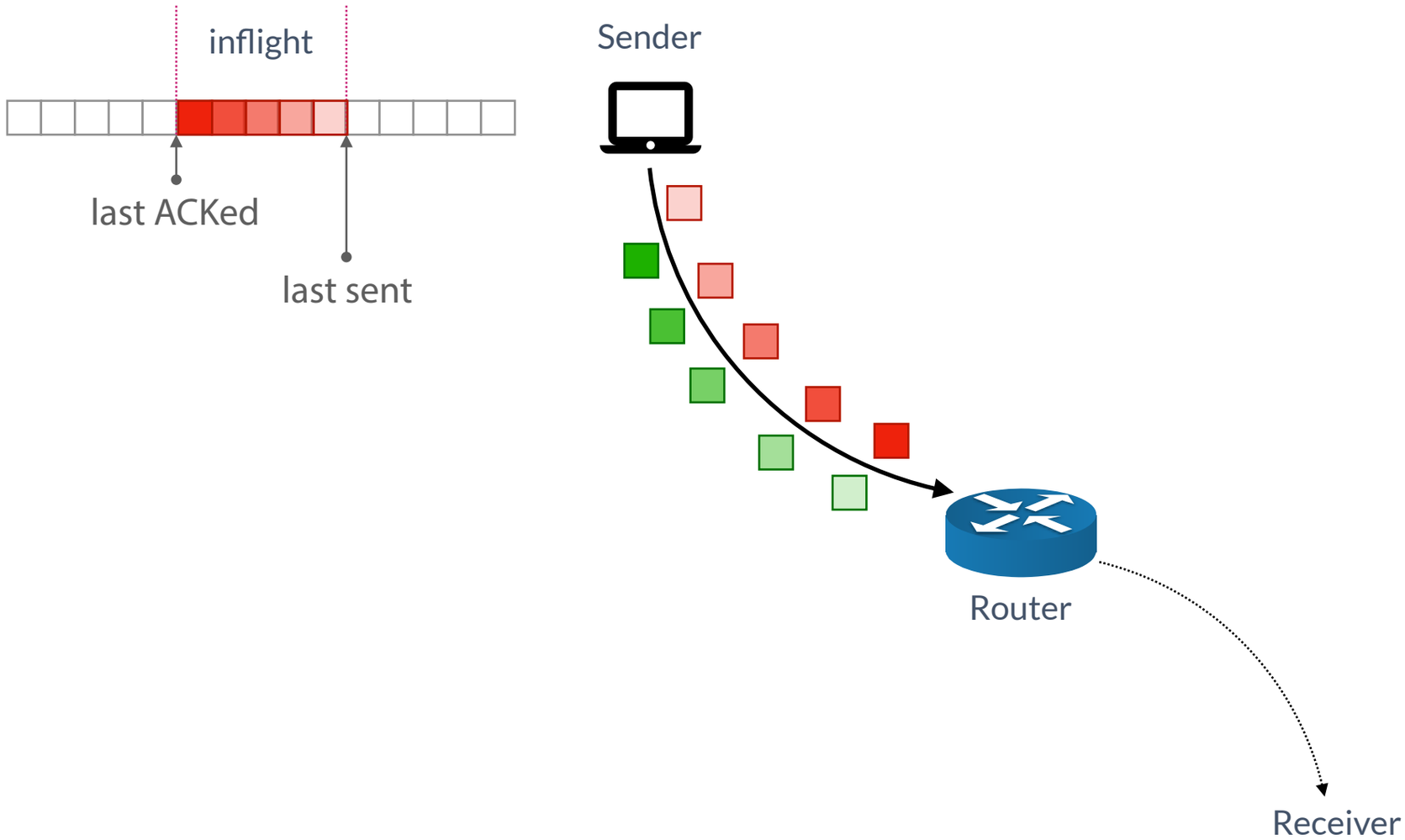
Sliding Window



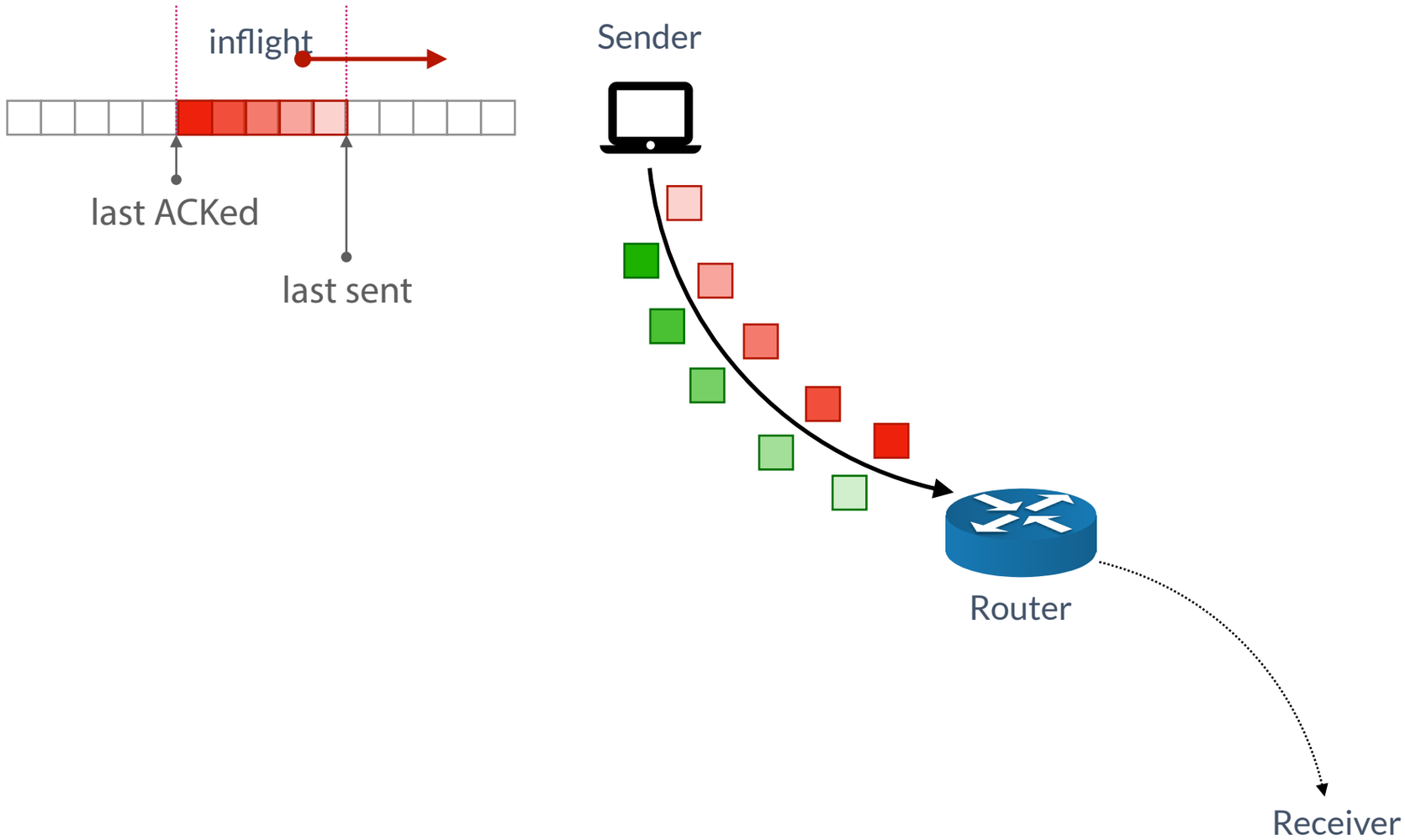
Sliding Window



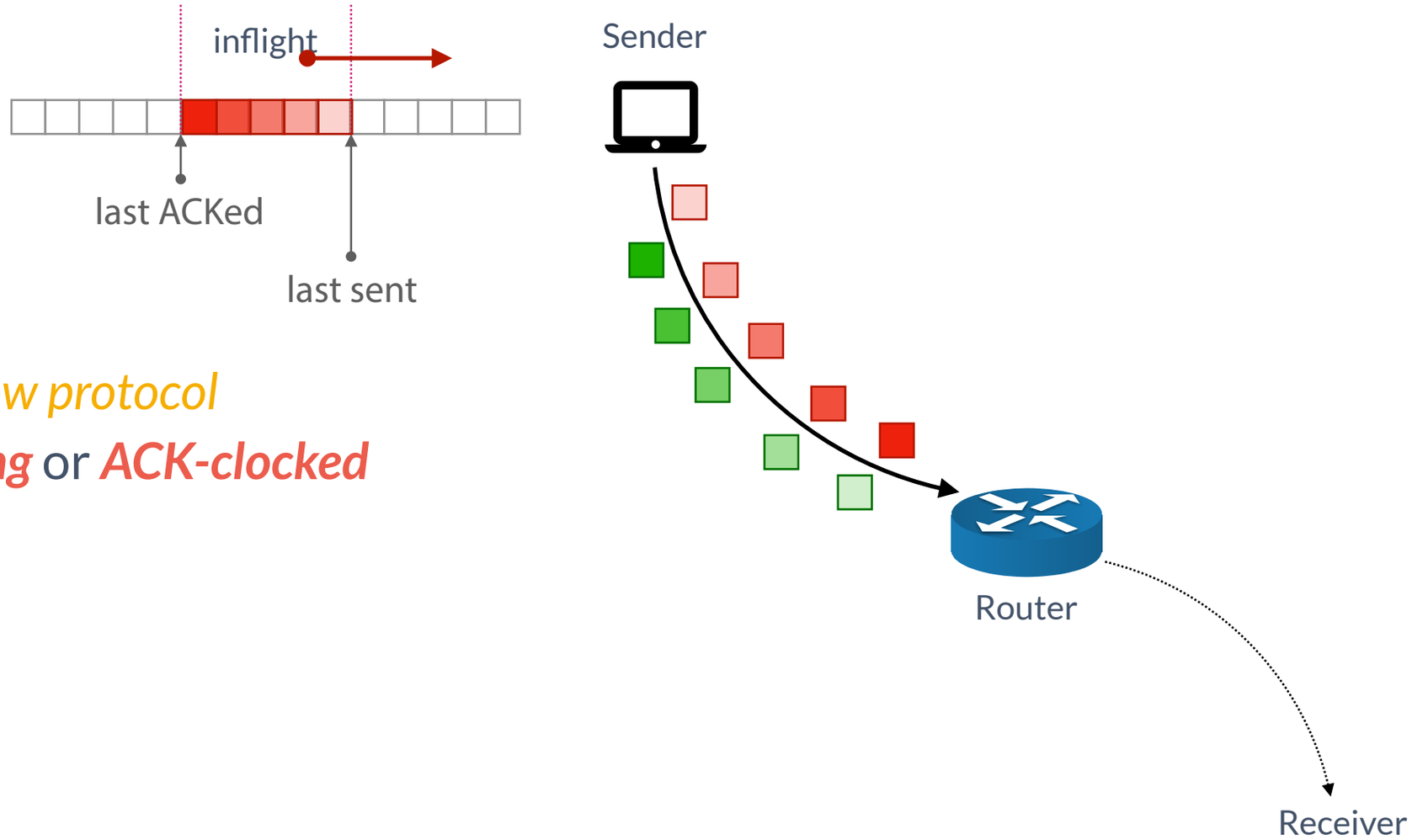
Sliding Window



Sliding Window



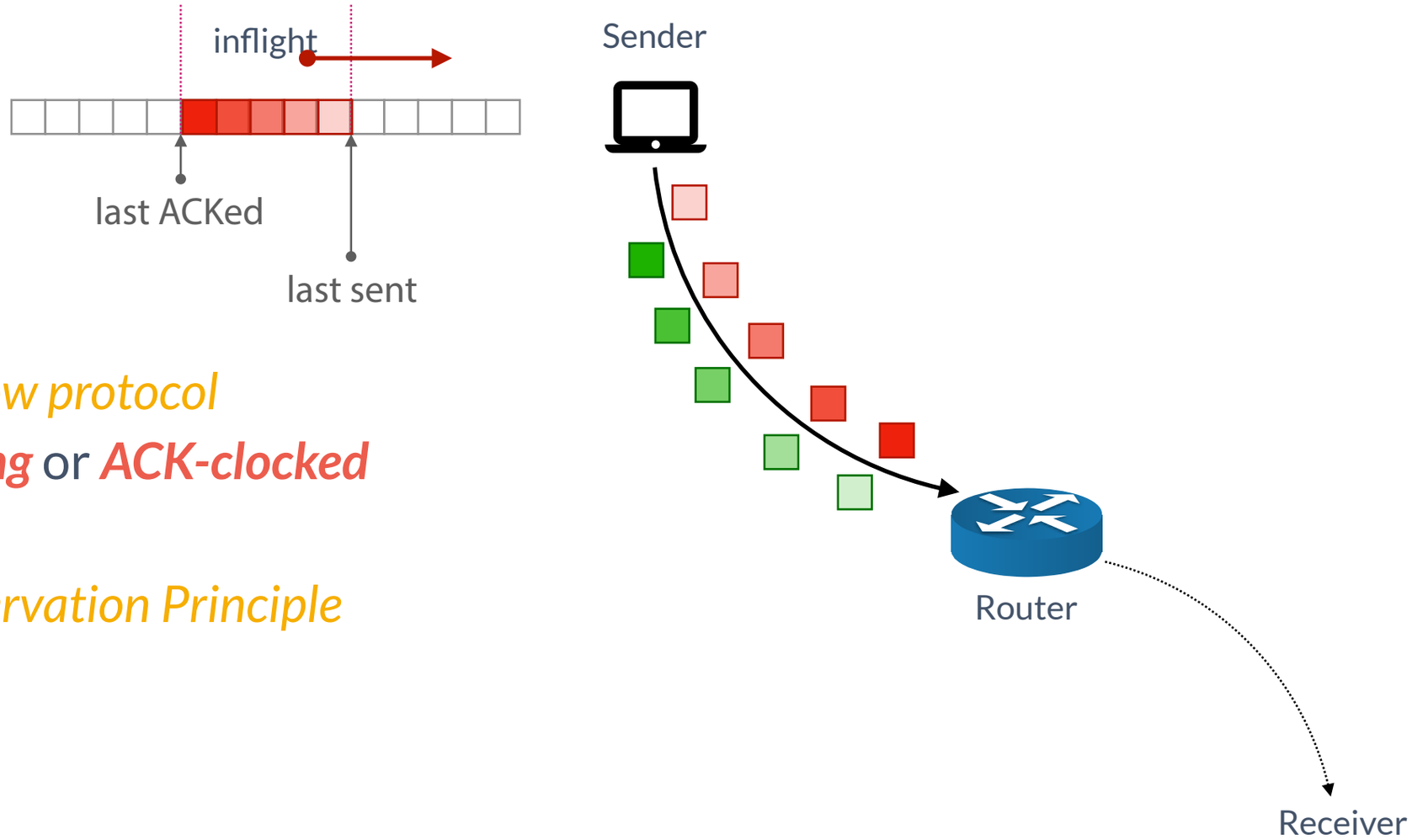
Sliding Window



Sliding window protocol

Self-clocking or ***ACK-clocked***

Sliding Window



Sliding window protocol

Self-clocking or ***ACK-clocked***

Packet Conservation Principle