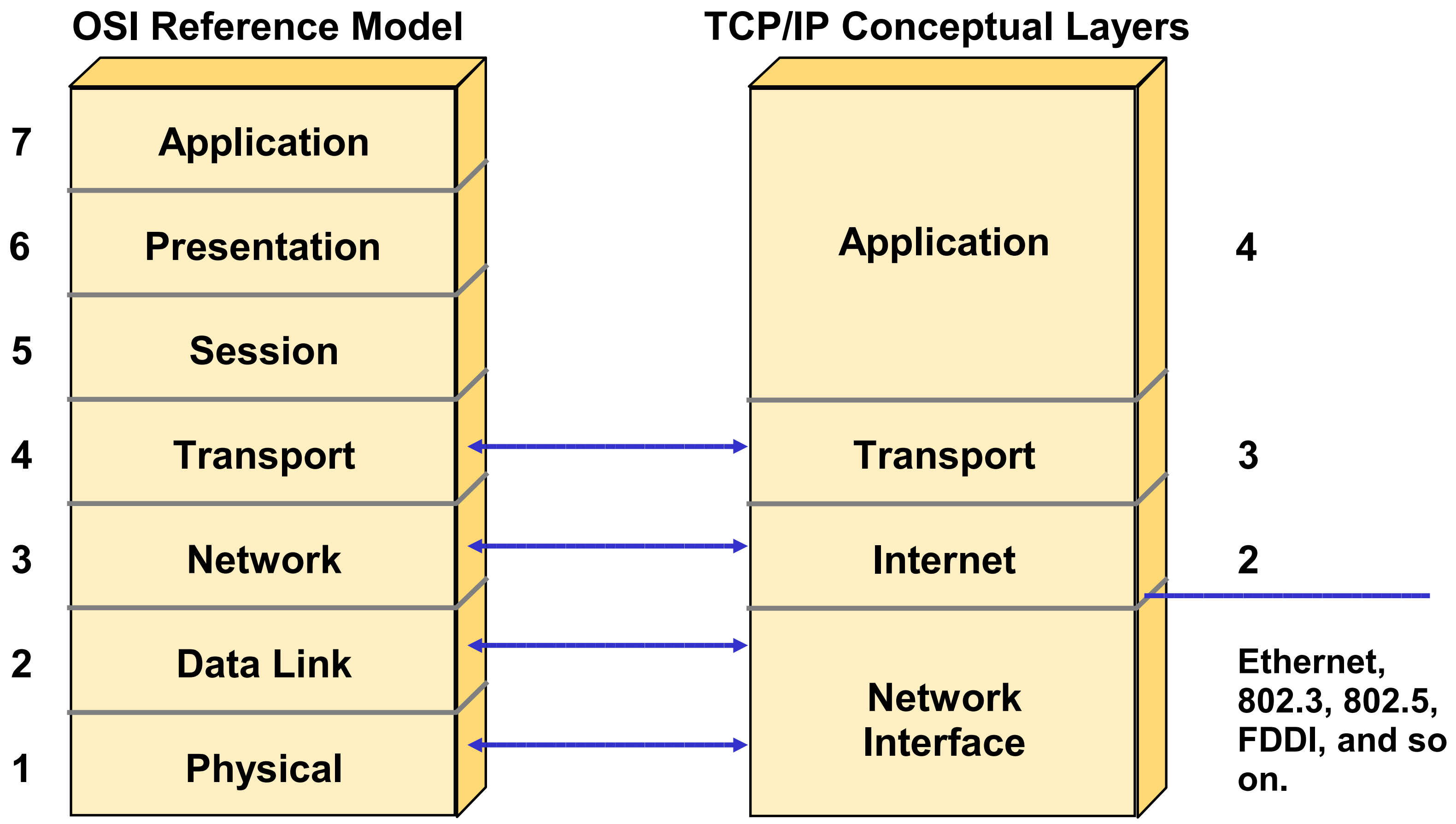
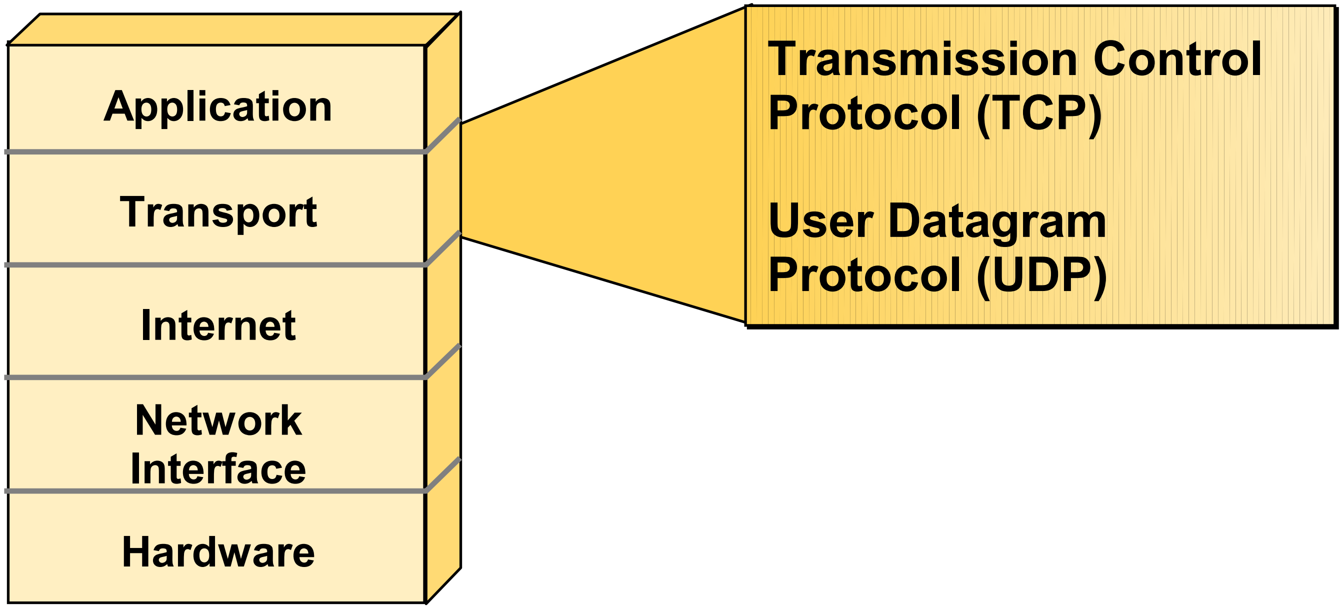
TCP Overview

TCP/IP Protocol Stack



31 January 2014

Transport Layer Overview



Transmission Control Protocol (TCP)

User Datagram Protocol (UDP)

TCP

- Defined in RFC 793
- Reliable
  - Acknowledgments
  - Guarantee of packet delivery
  - Delayed Ack – Piggybacking
  - Reassembly of out of order data
  - Discards duplicates caused by IP
  - Provided end-to-end flow control
  - finite buffer size

TCP

- Connection oriented
  - Segments are dependent
  - Maintain state information of segments
  - Segments can take different routes
  - Segments are delivered in order to the application layer

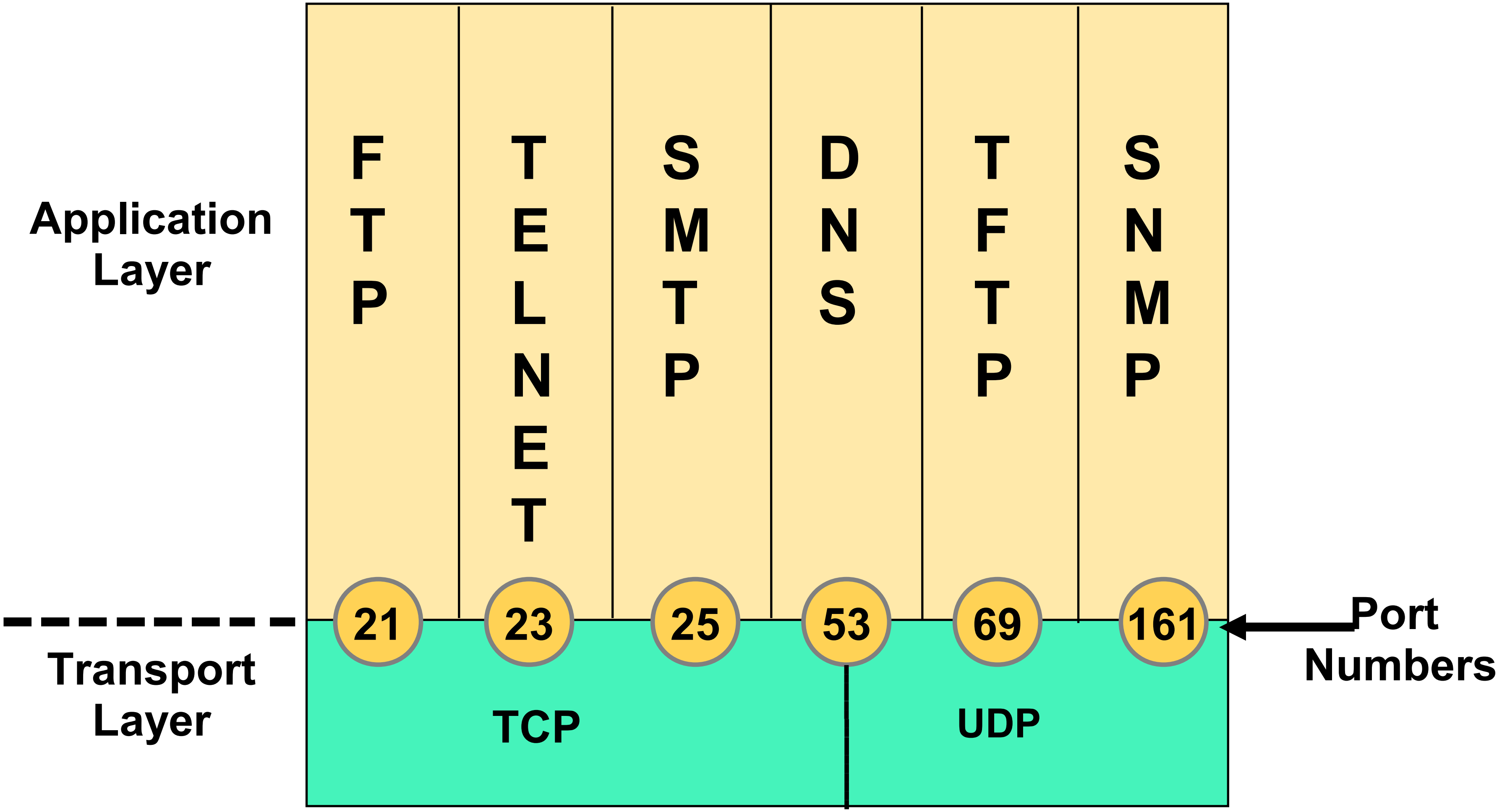
- Full Duplex

TCP Segment Format

# Bits	16	16	32	32	4	6	6
	Source Port	Dest. Port	Sequence Number	Acknowledgement Number	HLEN	Reserved	Code Bits

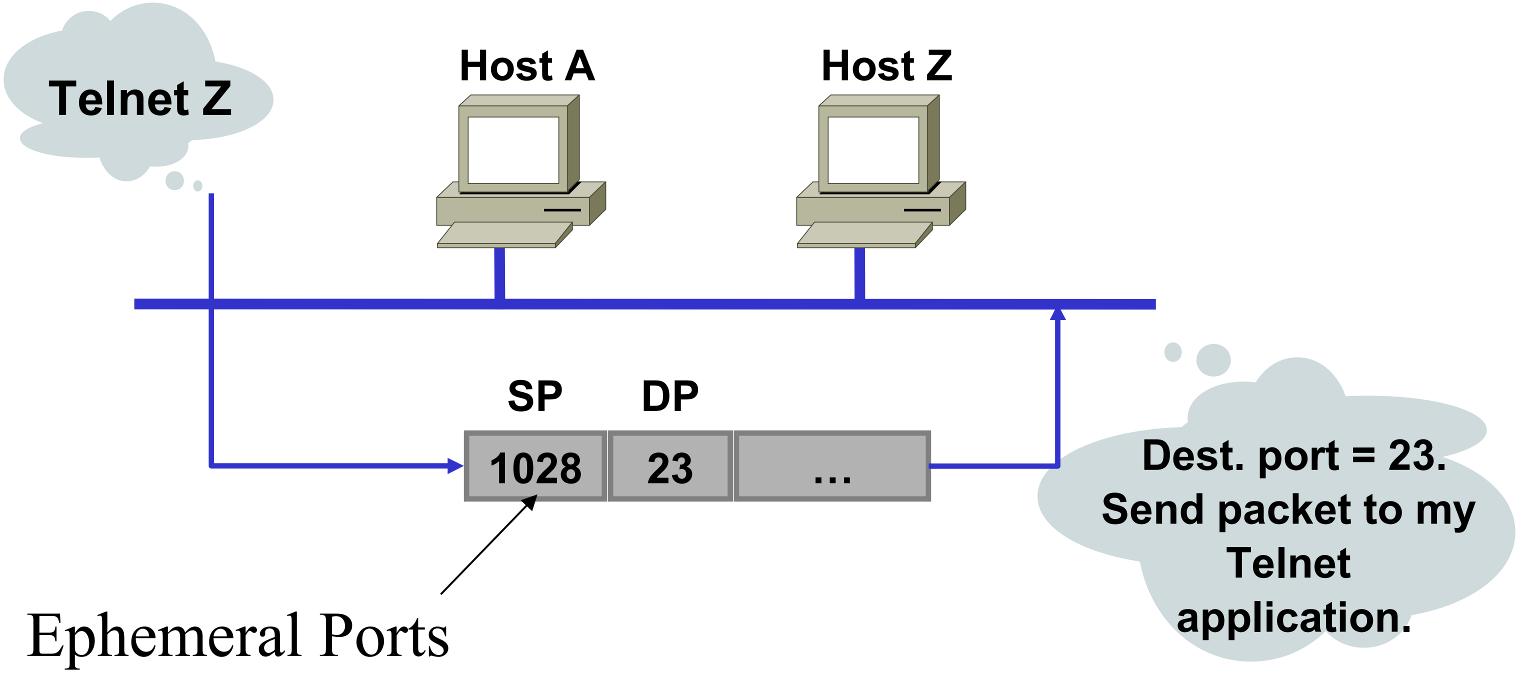
16	16	16	0 or 32	
Window	Check-sum	Urgent Pointer	Options	Data...

Port Numbers



TCP Port Numbers

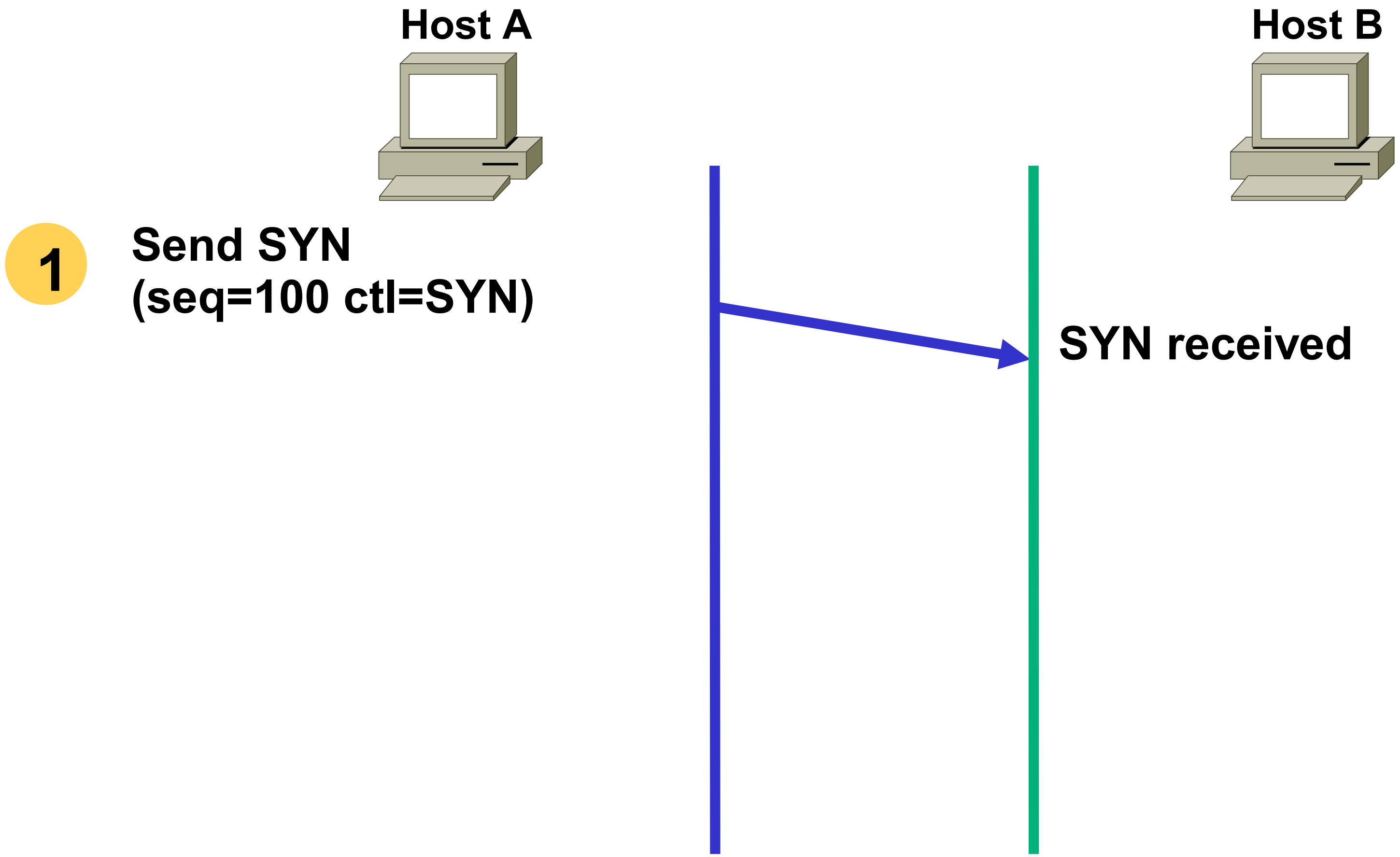
Source Port	Dest. Port	…



Dest. port = 23. Send packet to my Telnet application.

Ephemeral Ports

TCP Handshake/Open Connection



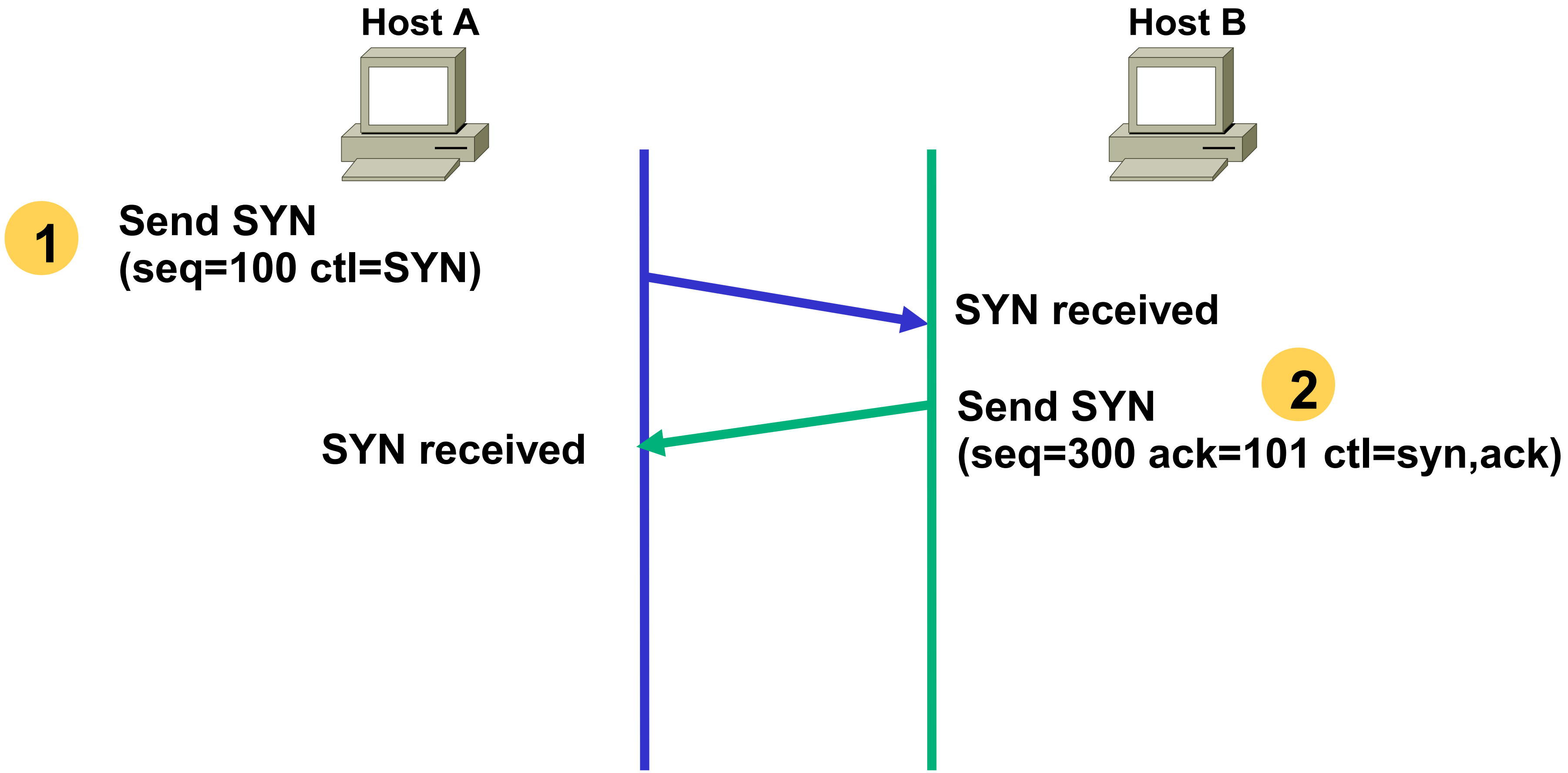
Host A

Host B

1 Send SYN
(seq=100 ctl=SYN)

SYN received

TCP Handshake/Open Connection



Host A

Host B

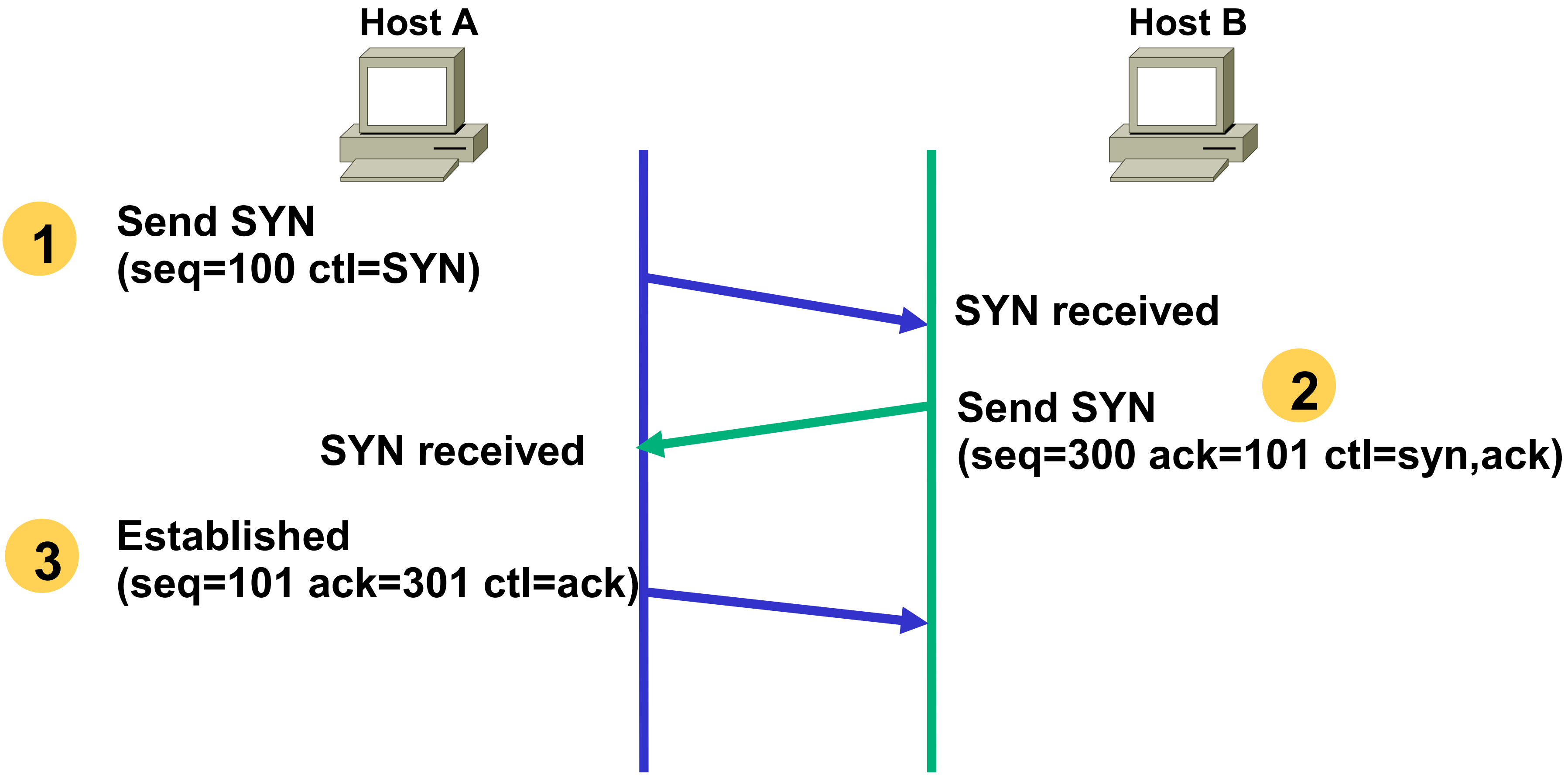
1 Send SYN
(seq=100 ctl=SYN)

SYN received

2 Send SYN
(seq=300 ack=101 ctl=syn,ack)

SYN received

TCP Handshake/Open Connection



Host A

Host B

1 Send SYN
(seq=100 ctl=SYN)

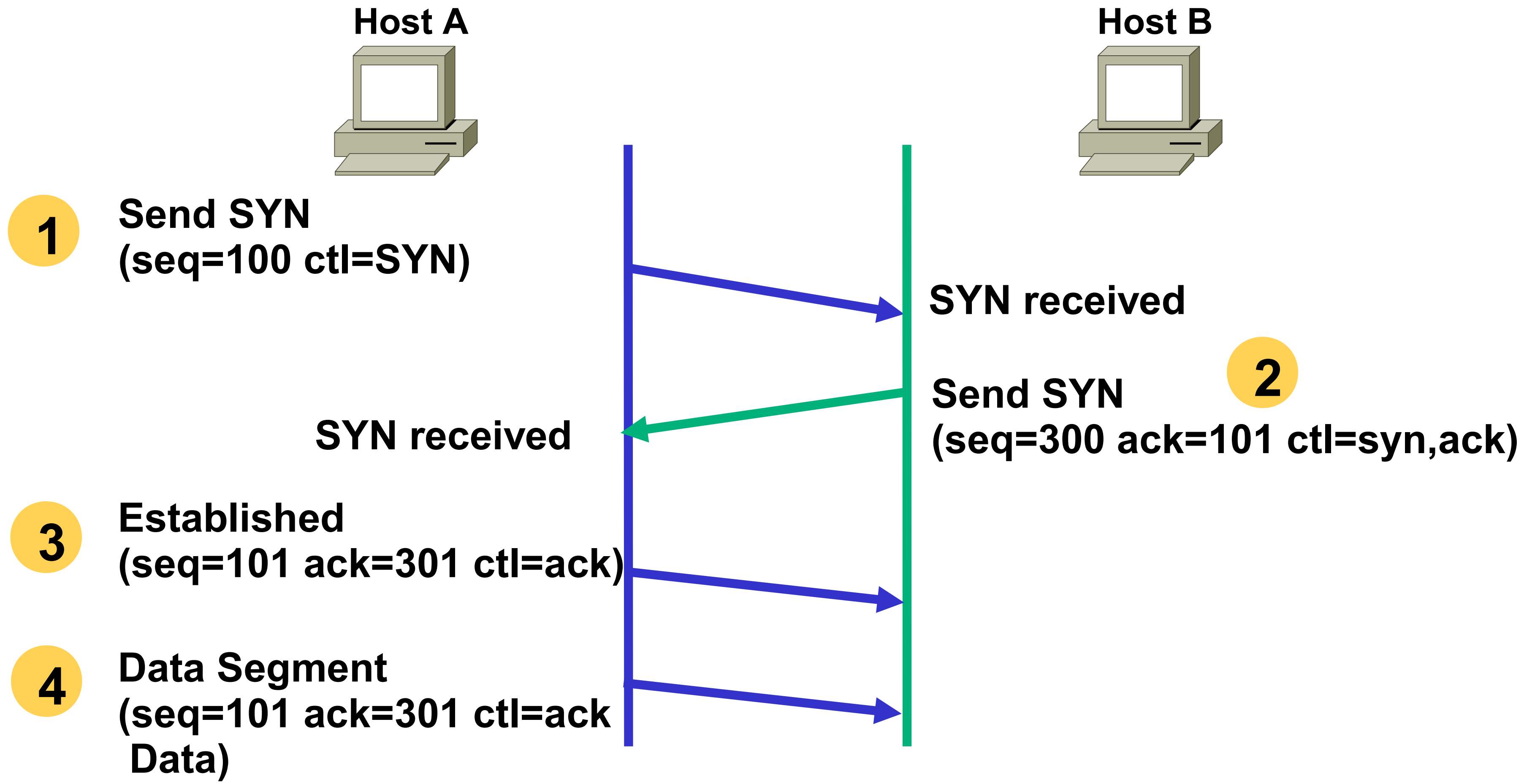
SYN received

2 Send SYN
(seq=300 ack=101 ctl=syn,ack)

SYN received

3 Established
(seq=101 ack=301 ctl=ack)

TCP Handshake/Open Connection



Host A

Host B

1 Send SYN
(seq=100 ctl=SYN)

SYN received

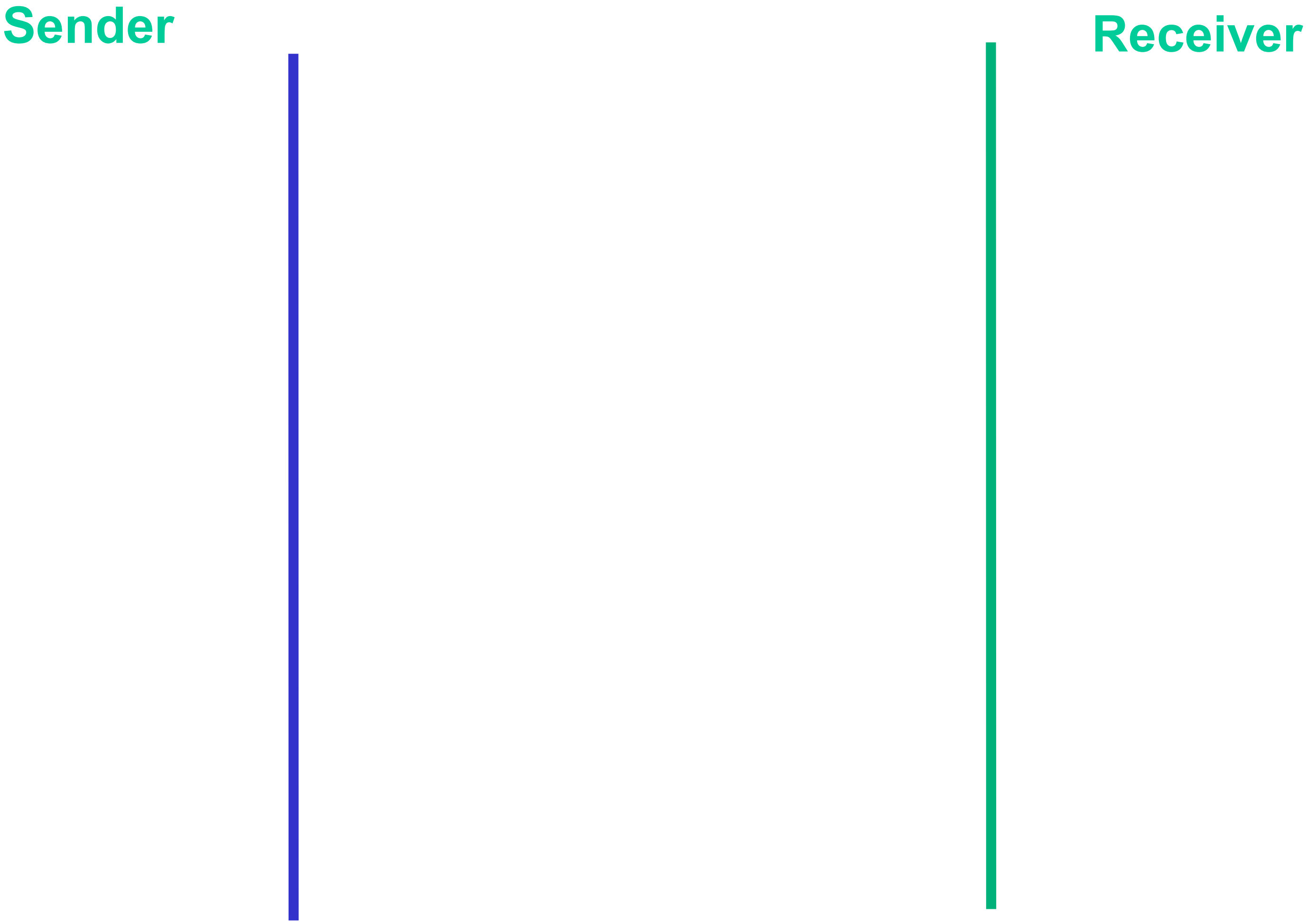
2 Send SYN
(seq=300 ack=101 ctl=syn,ack)

SYN received

3 Established
(seq=101 ack=301 ctl=ack)

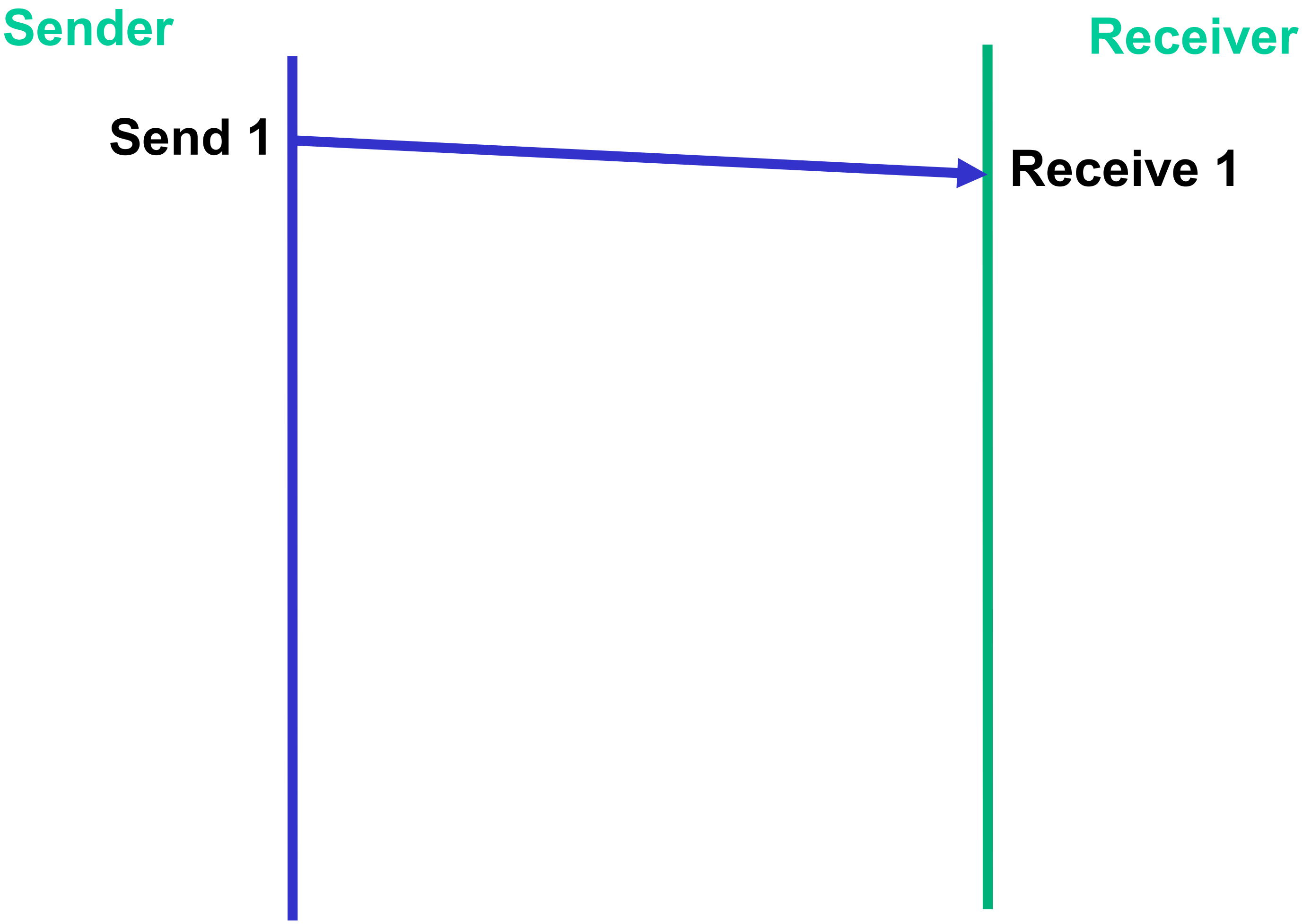
4 Data Segment
(seq=101 ack=301 ctl=ack
Data)

TCP Simple Acknowledgment



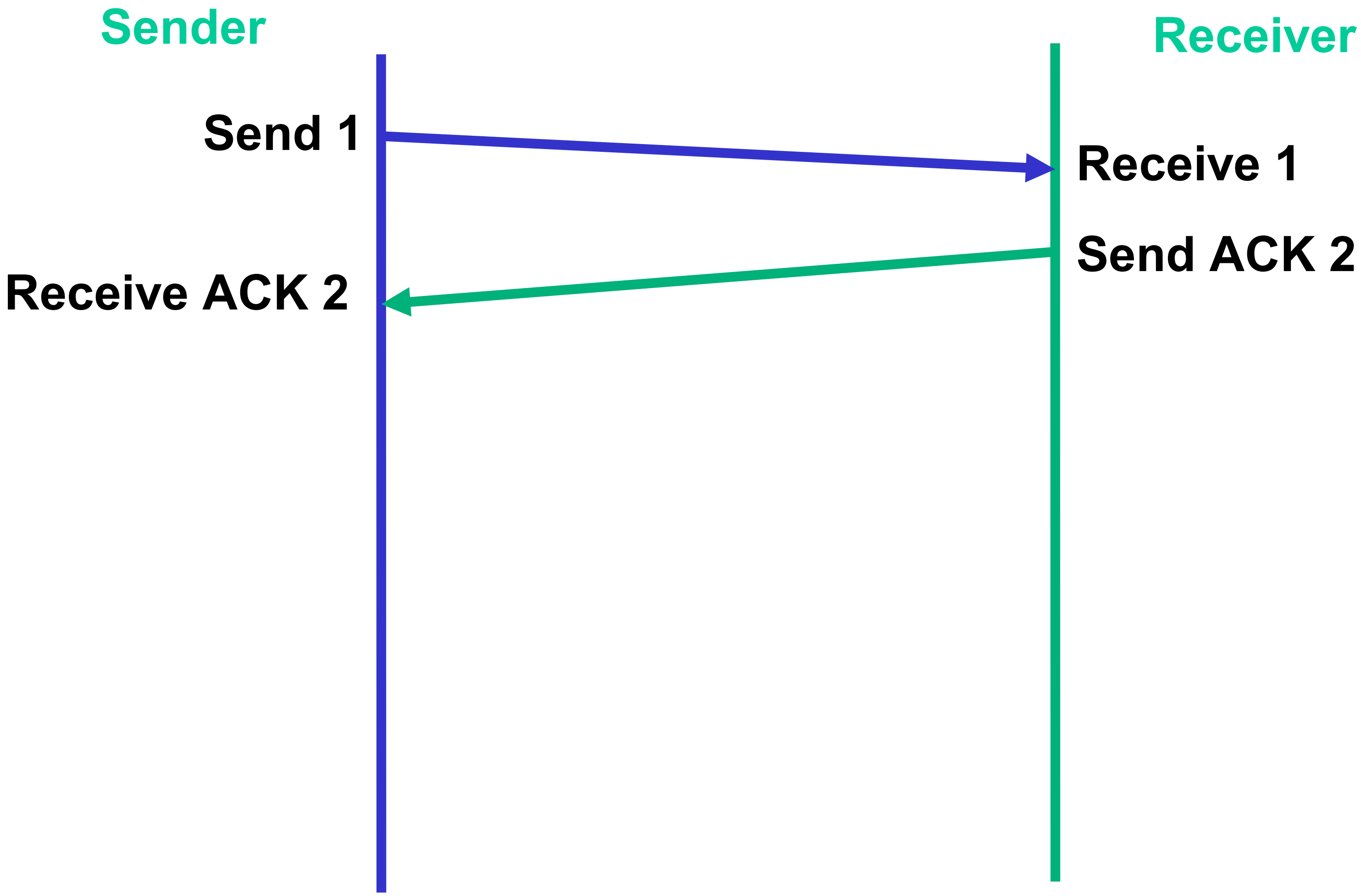
- Window size = 1

TCP Simple Acknowledgment



- Window size = 1

TCP Simple Acknowledgment



Sender

Receiver

Send 1

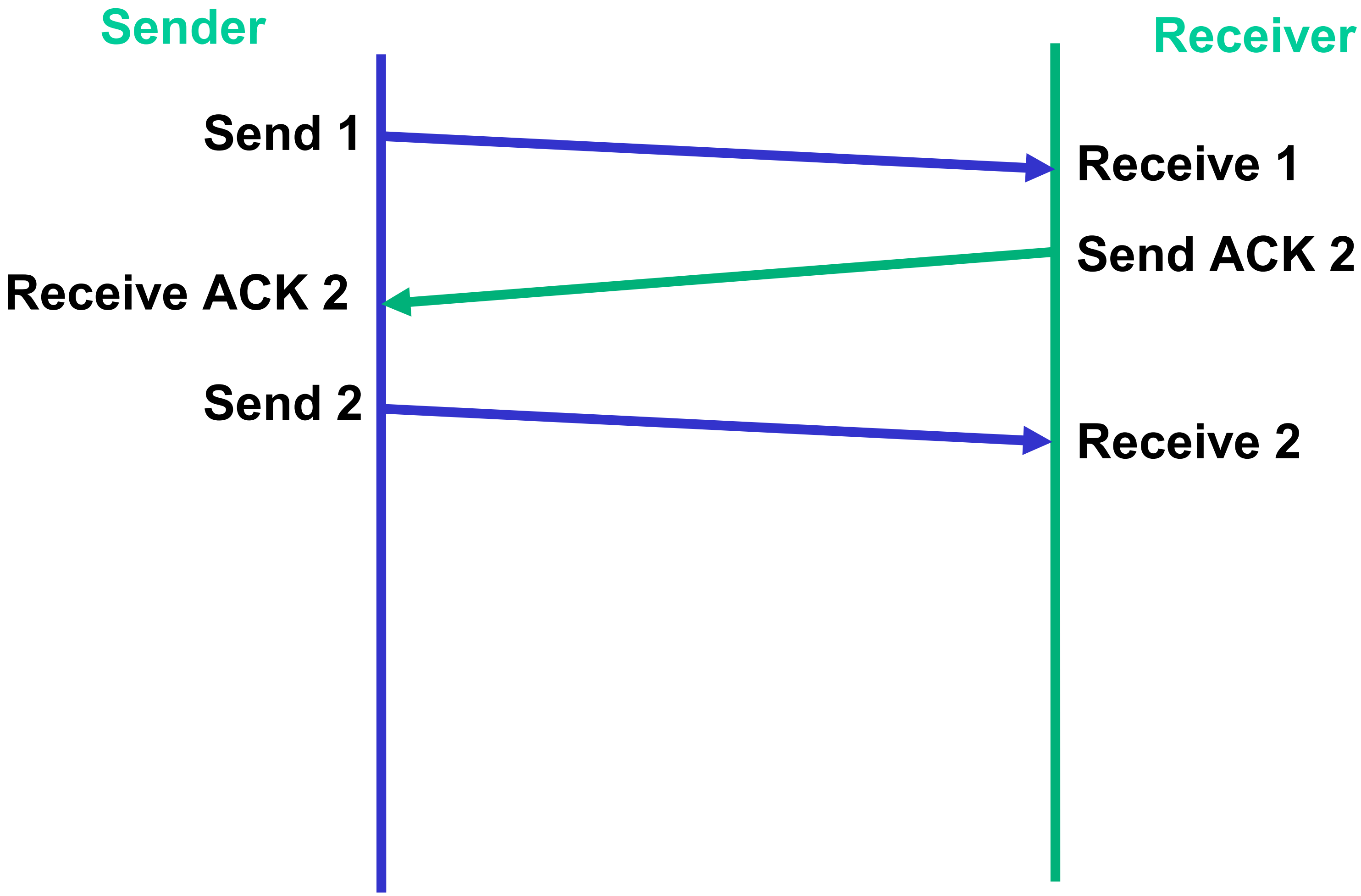
Receive 1

Send ACK 2

Receive ACK 2

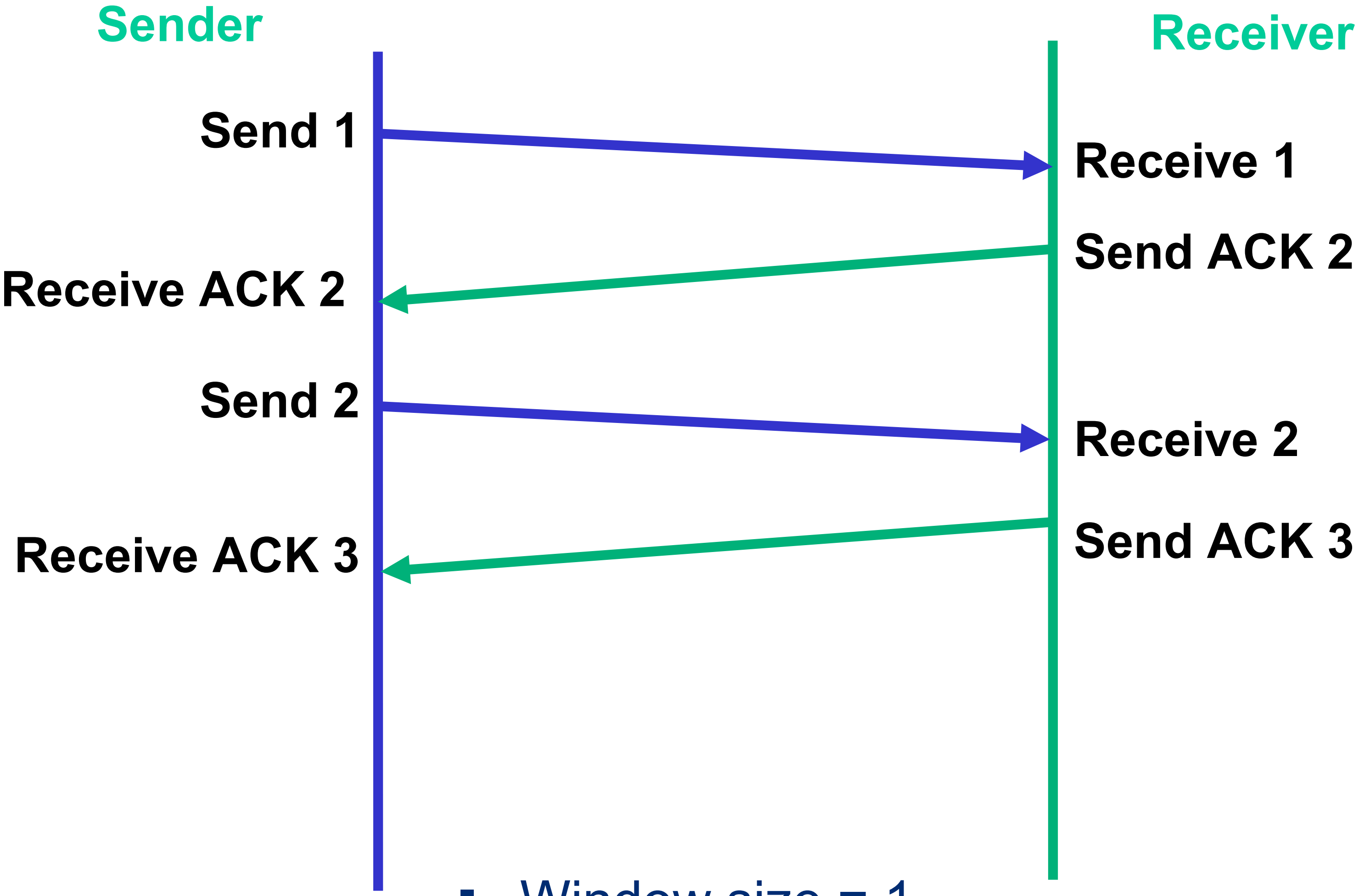
- Window size = 1

TCP Simple Acknowledgment



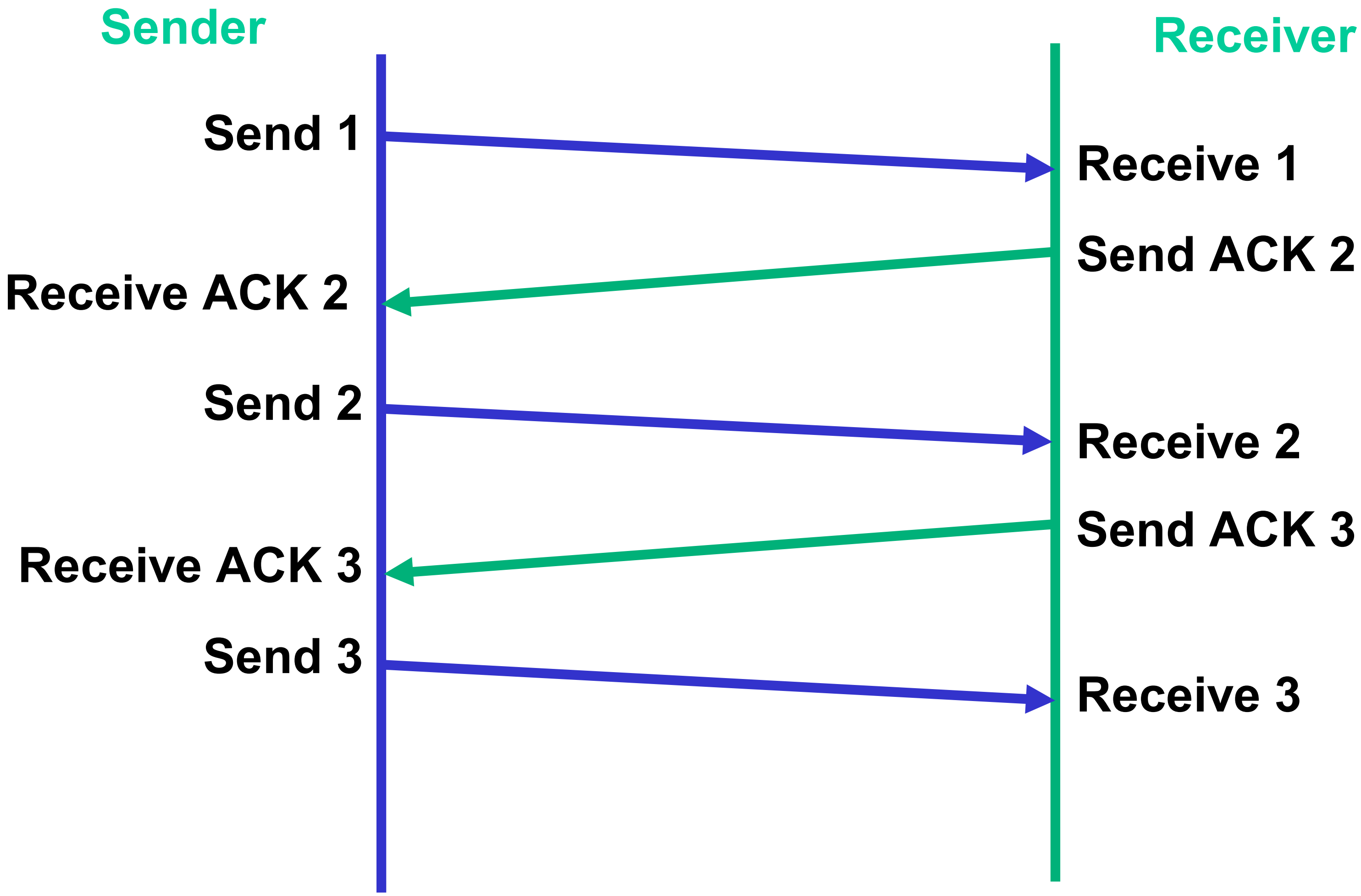
- Window size = 1

TCP Simple Acknowledgment



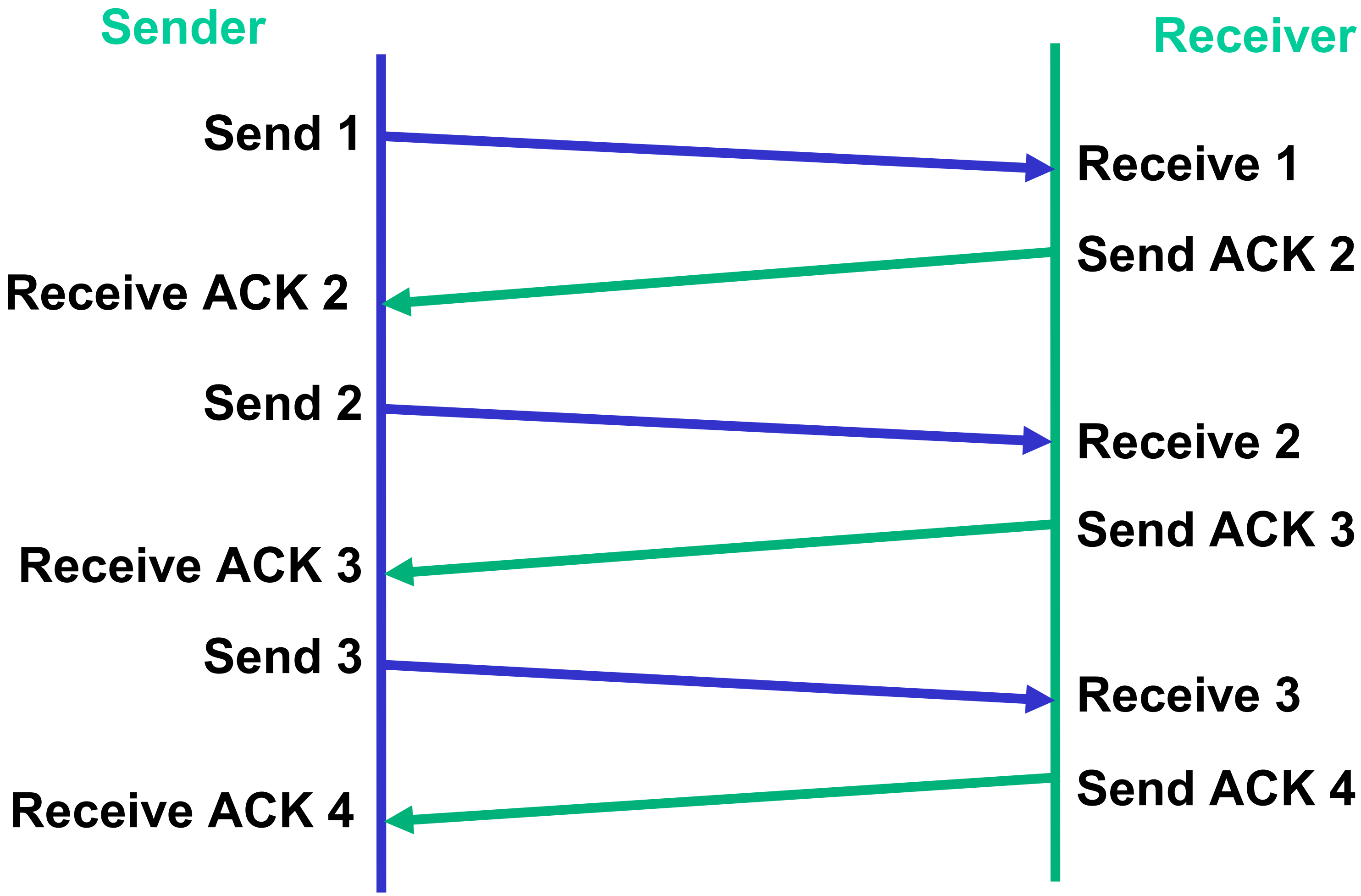
- Window size = 1

TCP Simple Acknowledgment



- Window size = 1

TCP Simple Acknowledgment



Sender

Receiver

Send 1

Receive 1

Send ACK 2

Receive ACK 2

Send 2

Receive 2

Send ACK 3

Receive ACK 3

Send 3

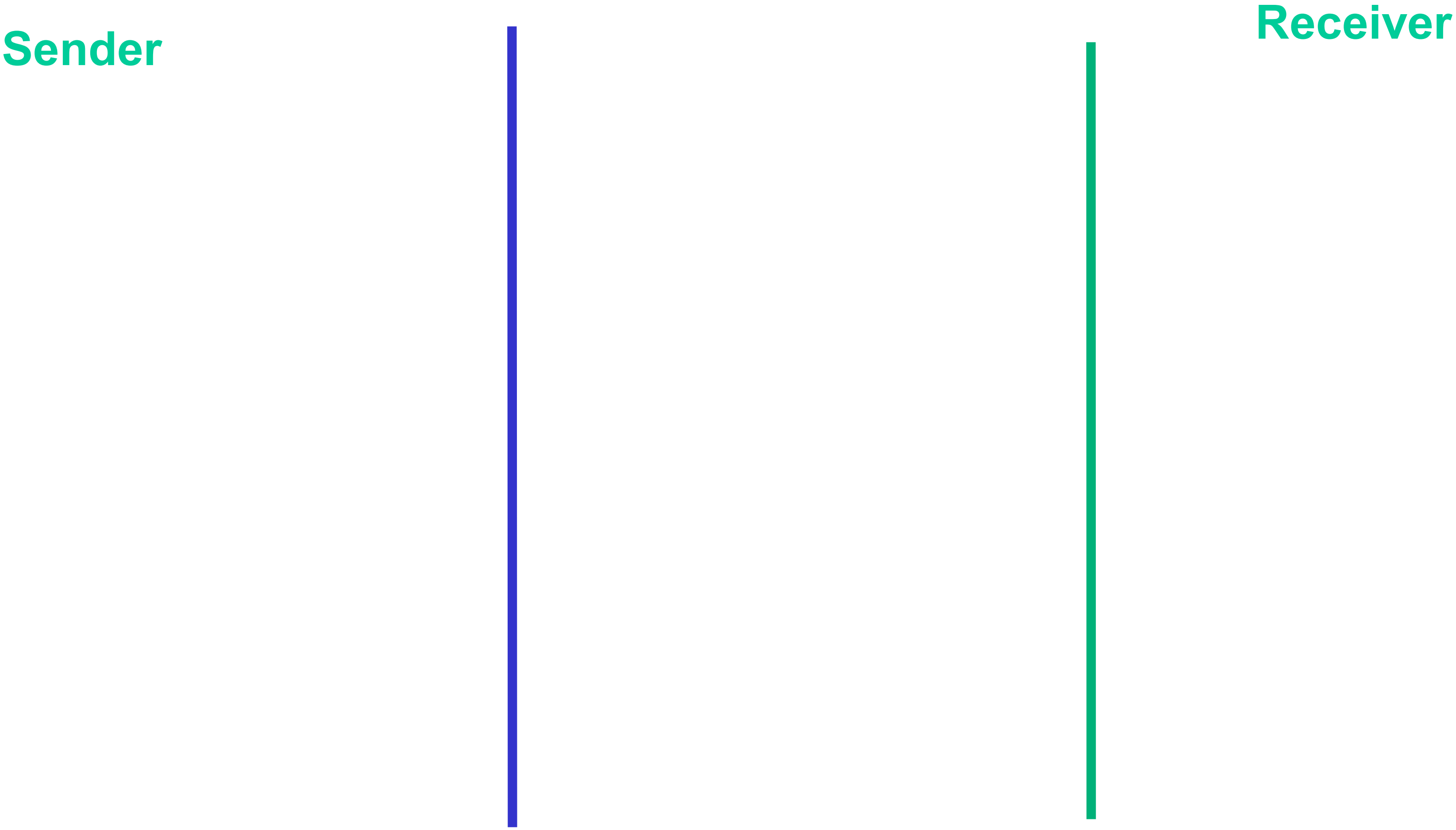
Receive 3

Send ACK 4

Receive ACK 4

- Window size = 1

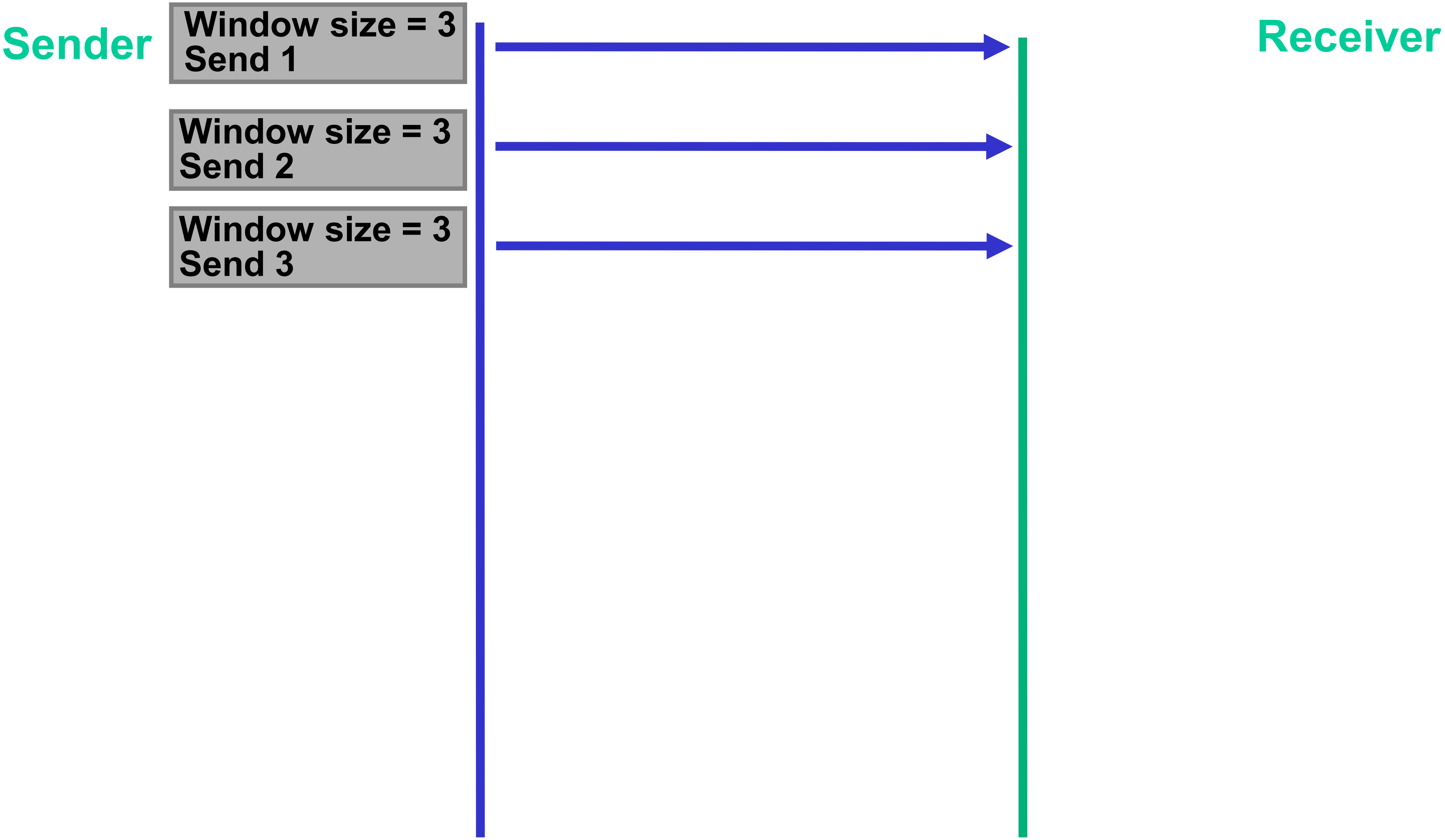
TCP Windowing



Sender

Receiver

TCP Windowing



Sender

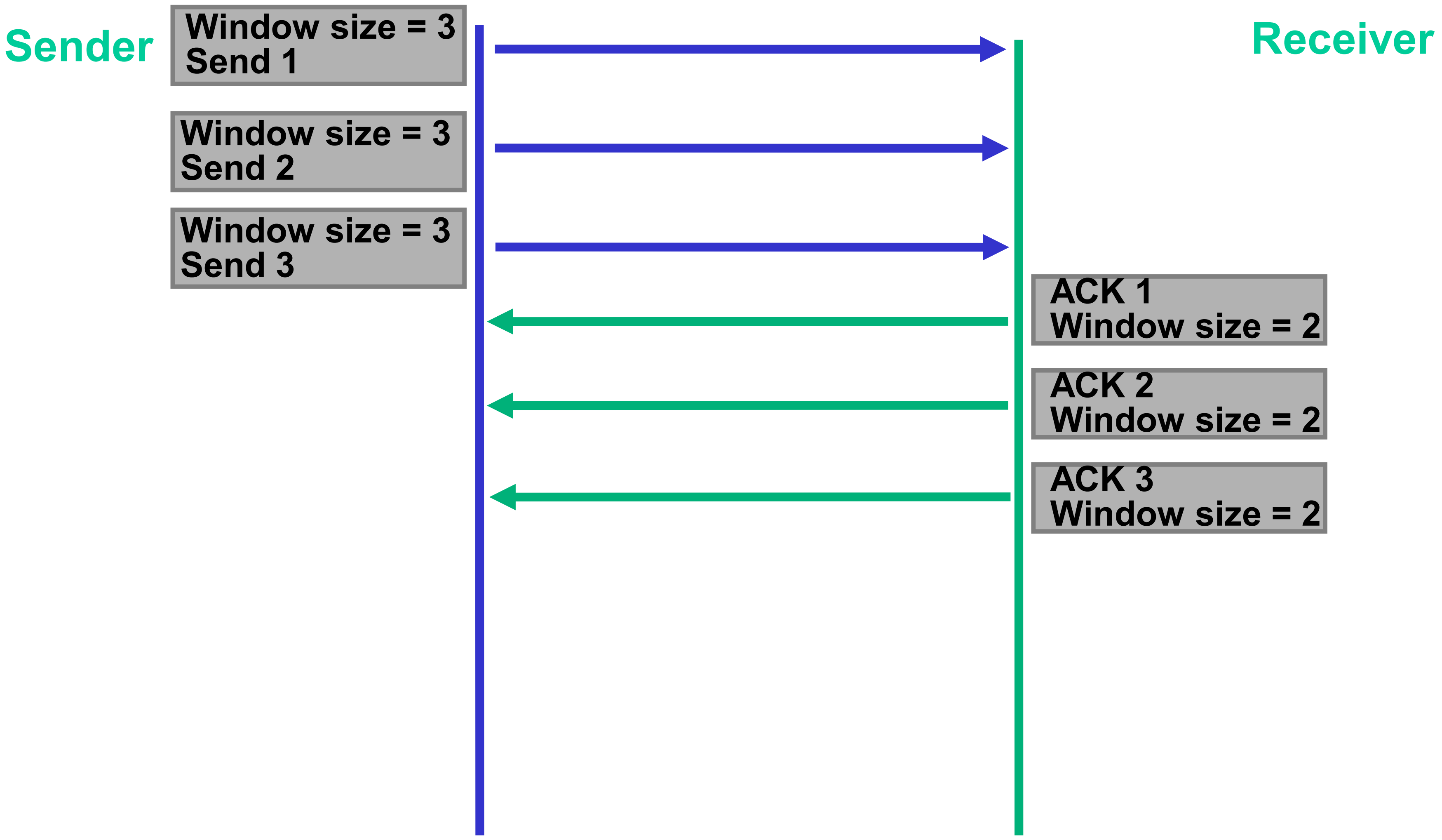
Window size = 3
Send 1

Window size = 3
Send 2

Window size = 3
Send 3

Receiver

TCP Windowing



Sender

Receiver

Window size = 3
Send 1

Window size = 3
Send 2

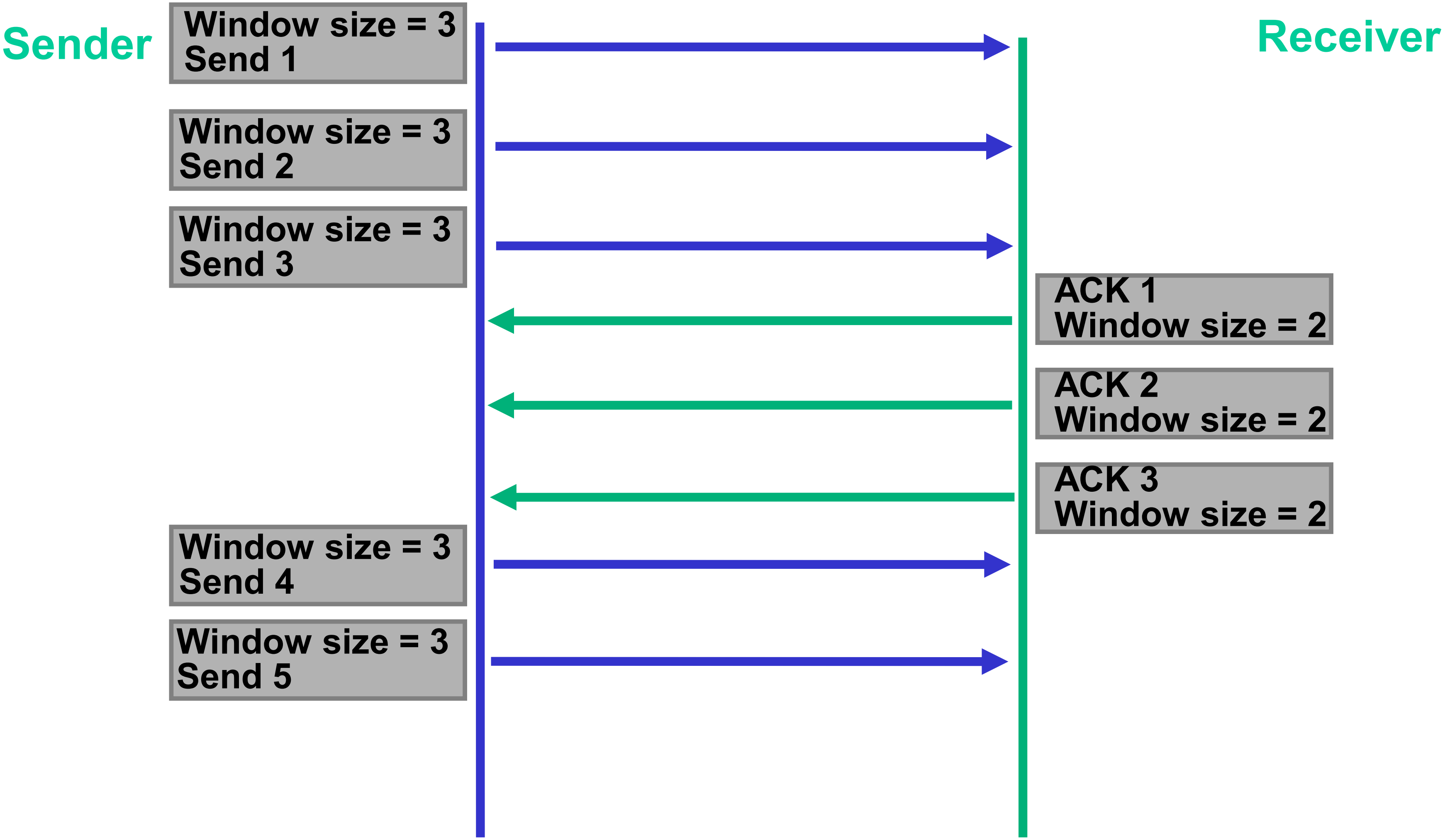
Window size = 3
Send 3

ACK 1
Window size = 2

ACK 2
Window size = 2

ACK 3
Window size = 2

TCP Windowing



Sender

Window size = 3
Send 1

Window size = 3
Send 2

Window size = 3
Send 3

Receiver

ACK 1
Window size = 2

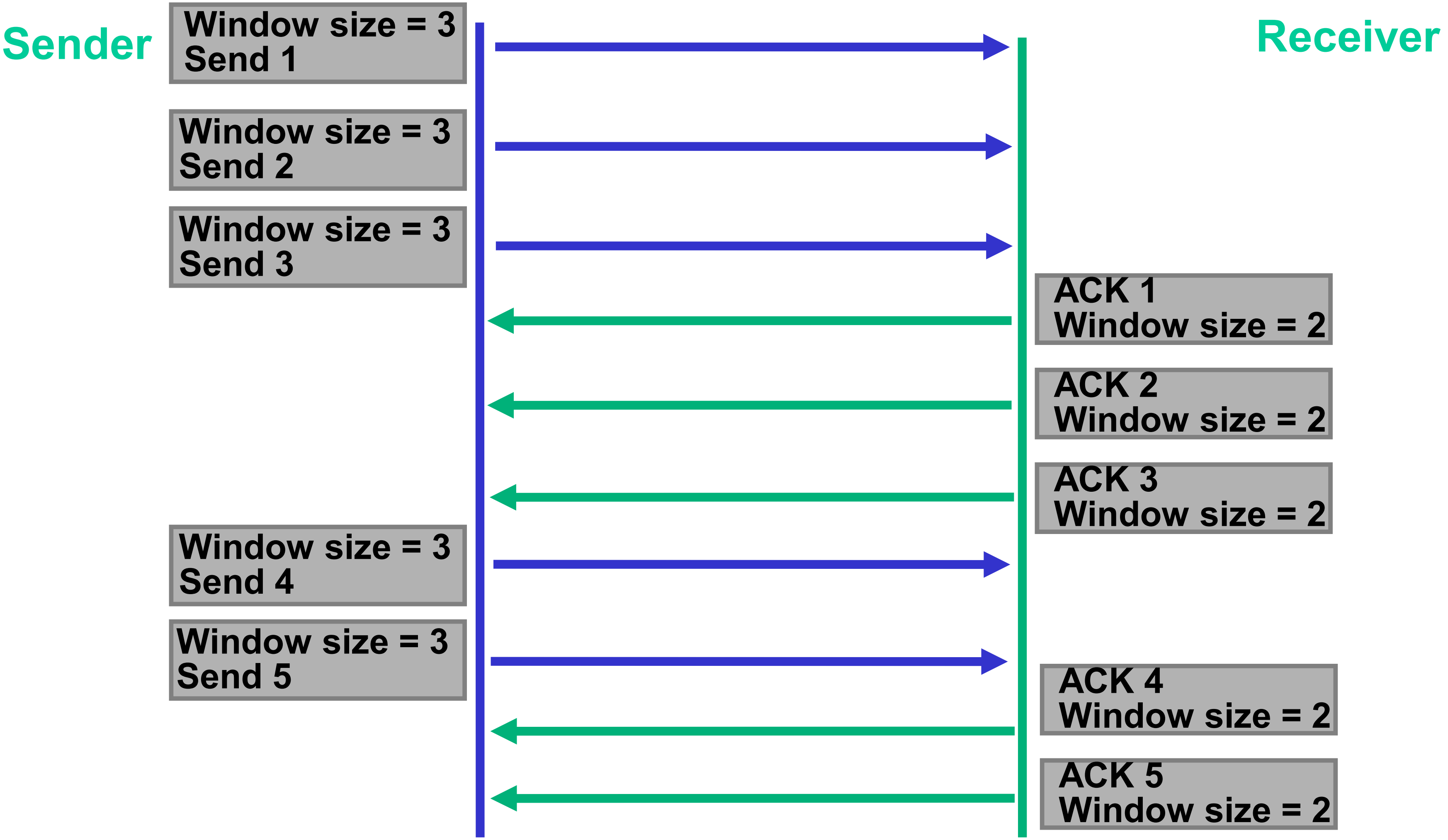
ACK 2
Window size = 2

ACK 3
Window size = 2

Window size = 3
Send 4

Window size = 3
Send 5

TCP Windowing



Sender

Receiver

Window size = 3
Send 1

Window size = 3
Send 2

Window size = 3
Send 3

ACK 1
Window size = 2

ACK 2
Window size = 2

ACK 3
Window size = 2

Window size = 3
Send 4

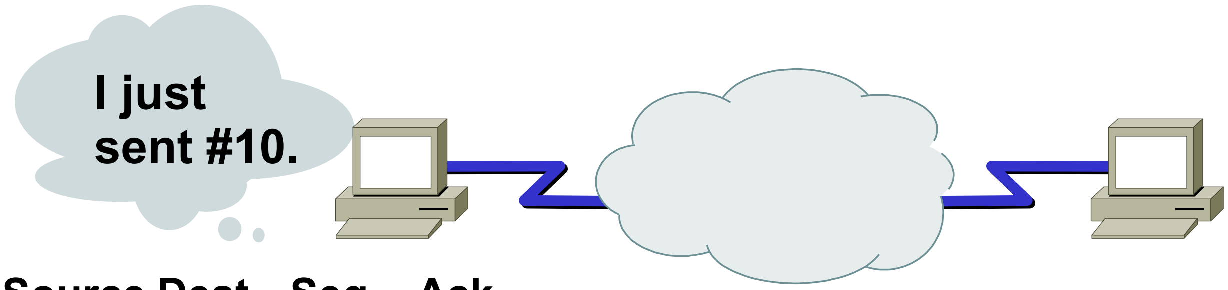
Window size = 3
Send 5

ACK 4
Window size = 2

ACK 5
Window size = 2

TCP Sequence and Ack Numbers

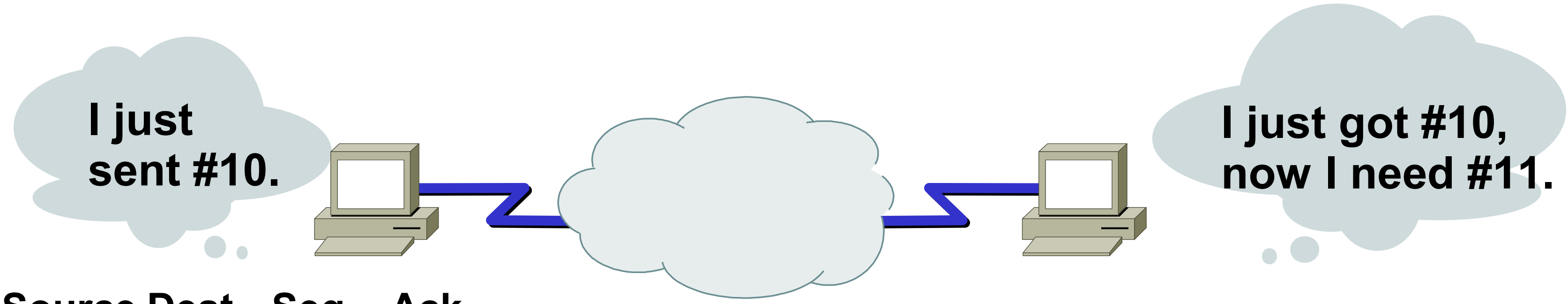
Source Port	Dest. Port	Sequence #	Acknowledgement #	…



Source	Dest.	Seq.	Ack.	
1028	23	10	1	…

TCP Sequence and Ack Numbers

Source Port	Dest. Port	Sequence #	Acknowledgement #	…



Source	Dest.	Seq.	Ack.	
1028	23	10	1	…

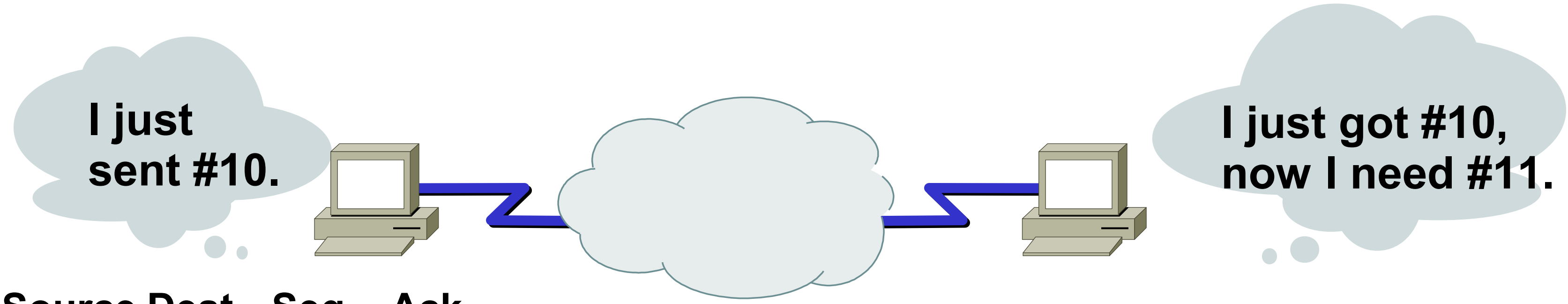
→

Source	Dest.	Seq.	Ack.	
23	1028	1	11	…

←

TCP Sequence and Ack Numbers

Source Port	Dest. Port	Sequence #	Acknowledgement #	…



I just sent #10.

I just got #10, now I need #11.

Source	Dest.	Seq.	Ack.	
1028	23	10	1	…

→

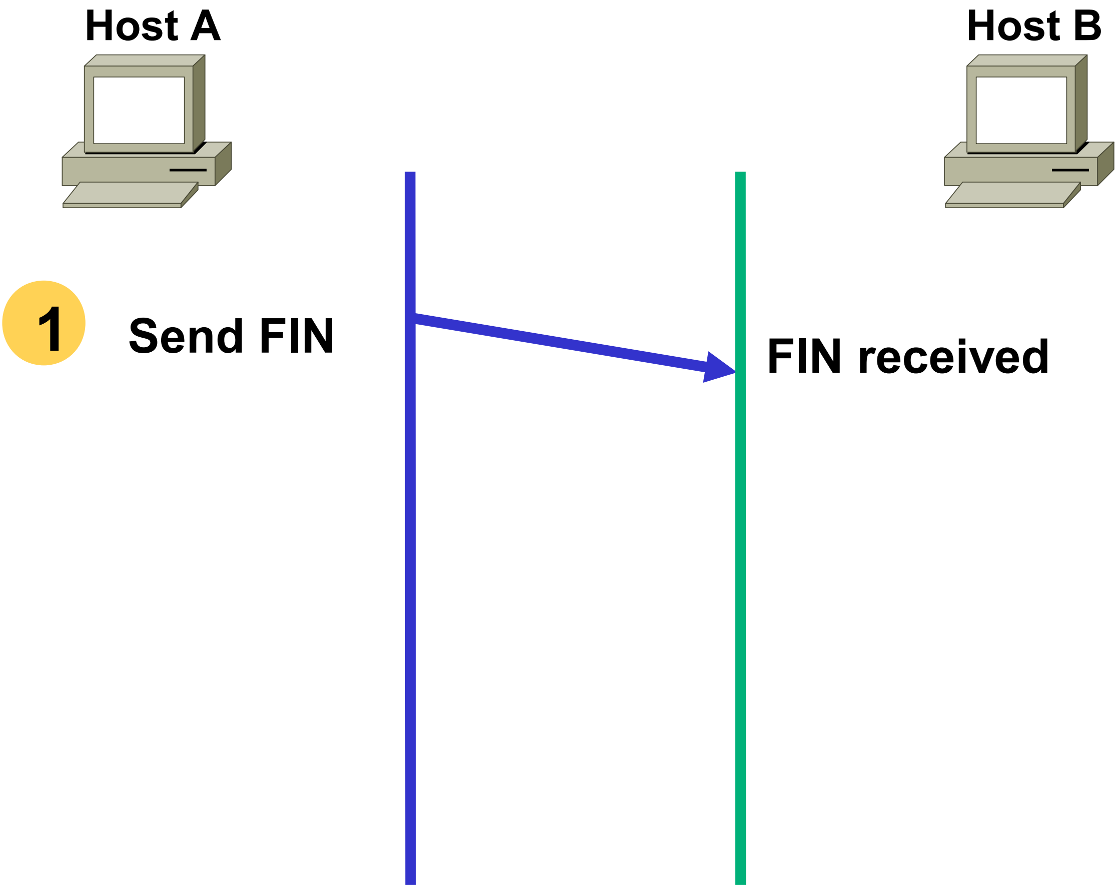
Source	Dest.	Seq.	Ack.	
23	1028	1	11	…

←

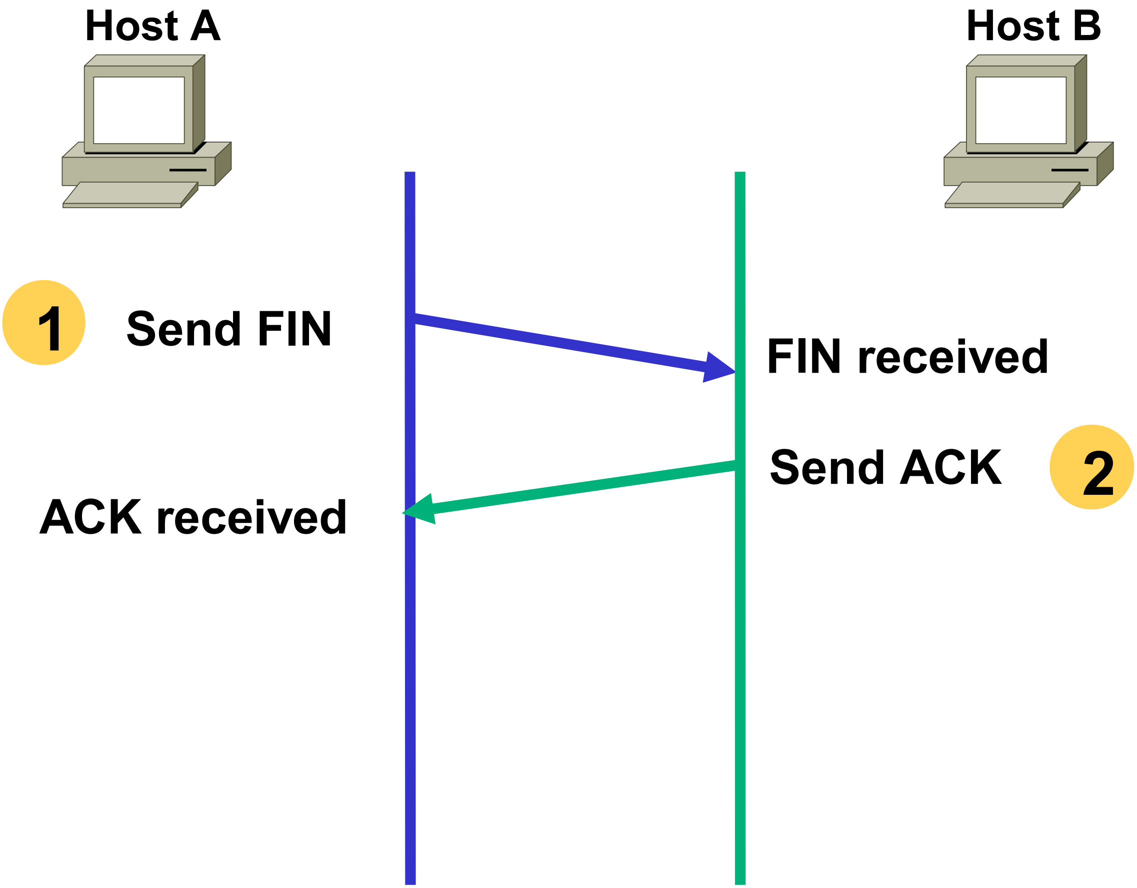
Source	Dest.	Seq.	Ack.	
1028	23	11	2	…

→

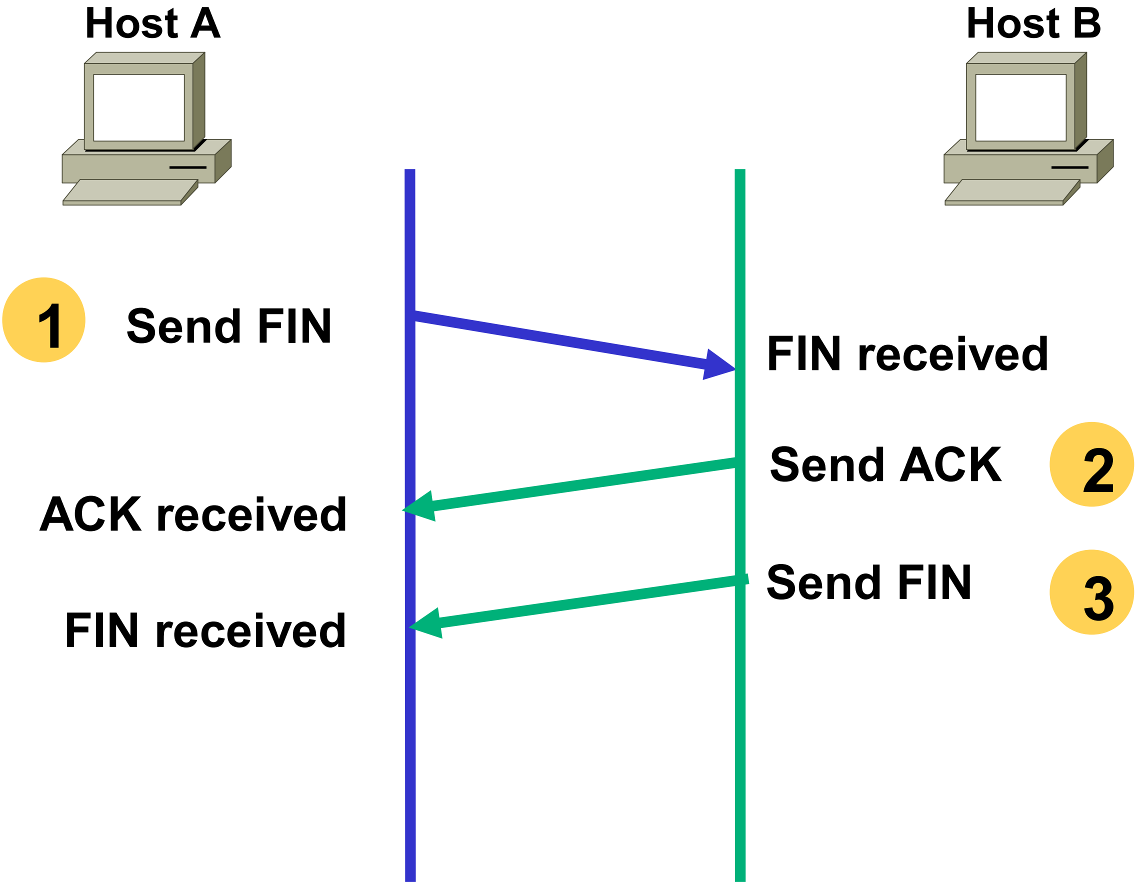
TCP Connection Closure



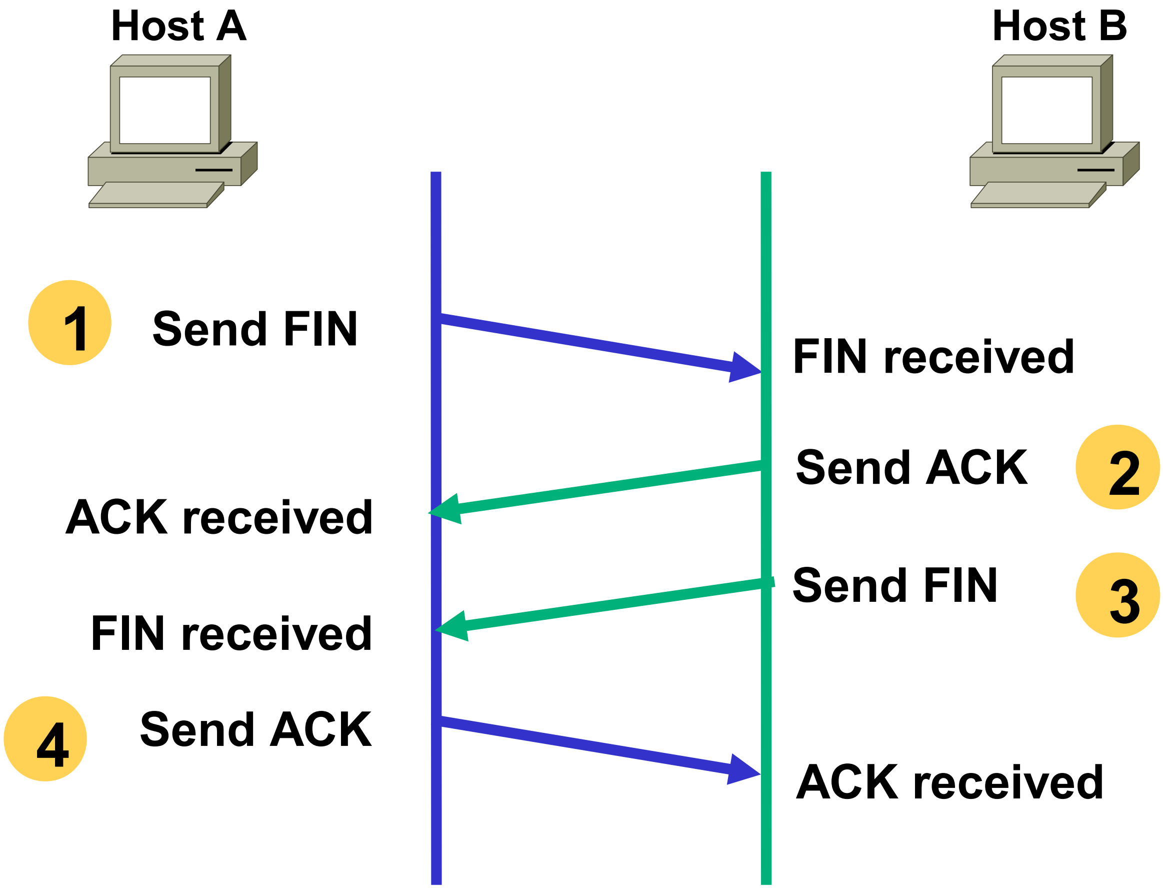
TCP Connection Closure



TCP Connection Closure



TCP Connection Closure



TCP Connection Closure

- Since TCP is full-duplex, connection must be shut down from both sides independently
  - it takes 4 segments to close the connection completely
- *Active close* – initiation of first FIN request
- *Passive close* – initiation of second FIN request

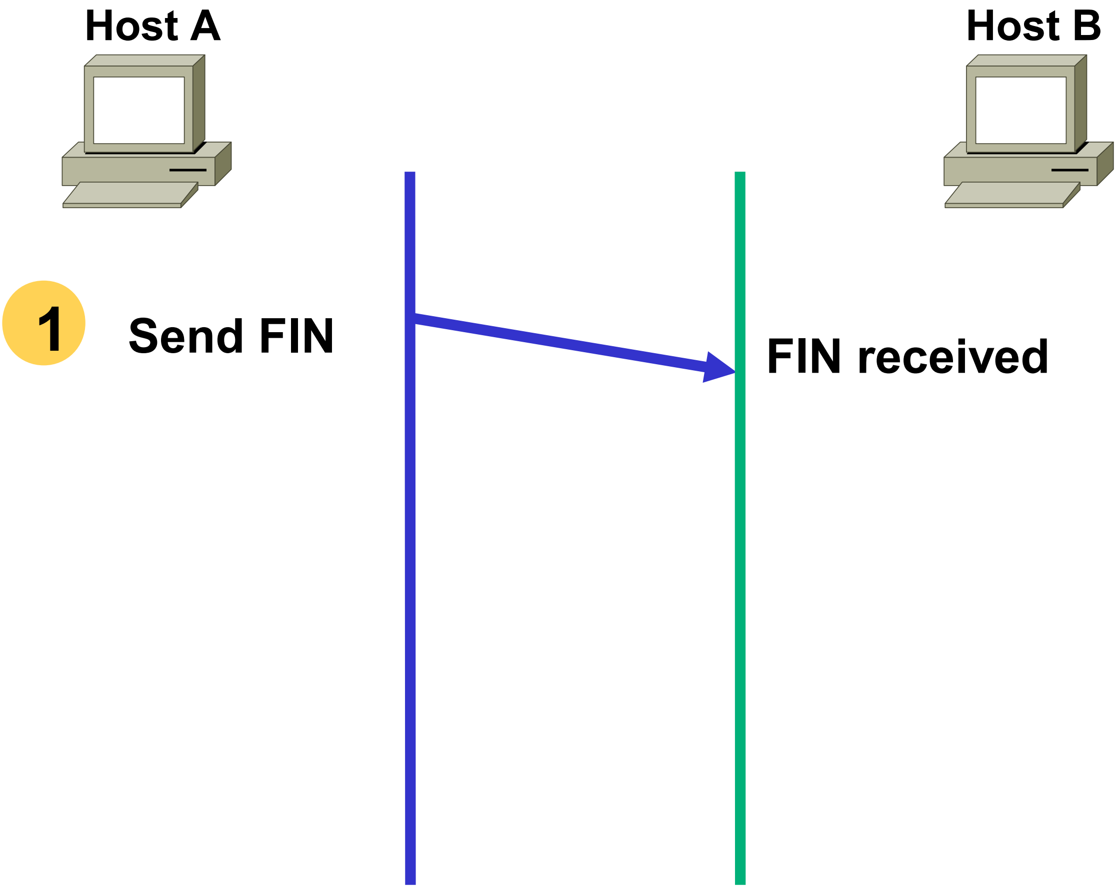
Connection closure sequence

- Application closes the session
- TCP sends FIN to the server
- Server TCP sends ACK to client
- Server TCP informs application
- Server application closes the session
- Server TCP sends FIN
- Client TCP sends ACK to server

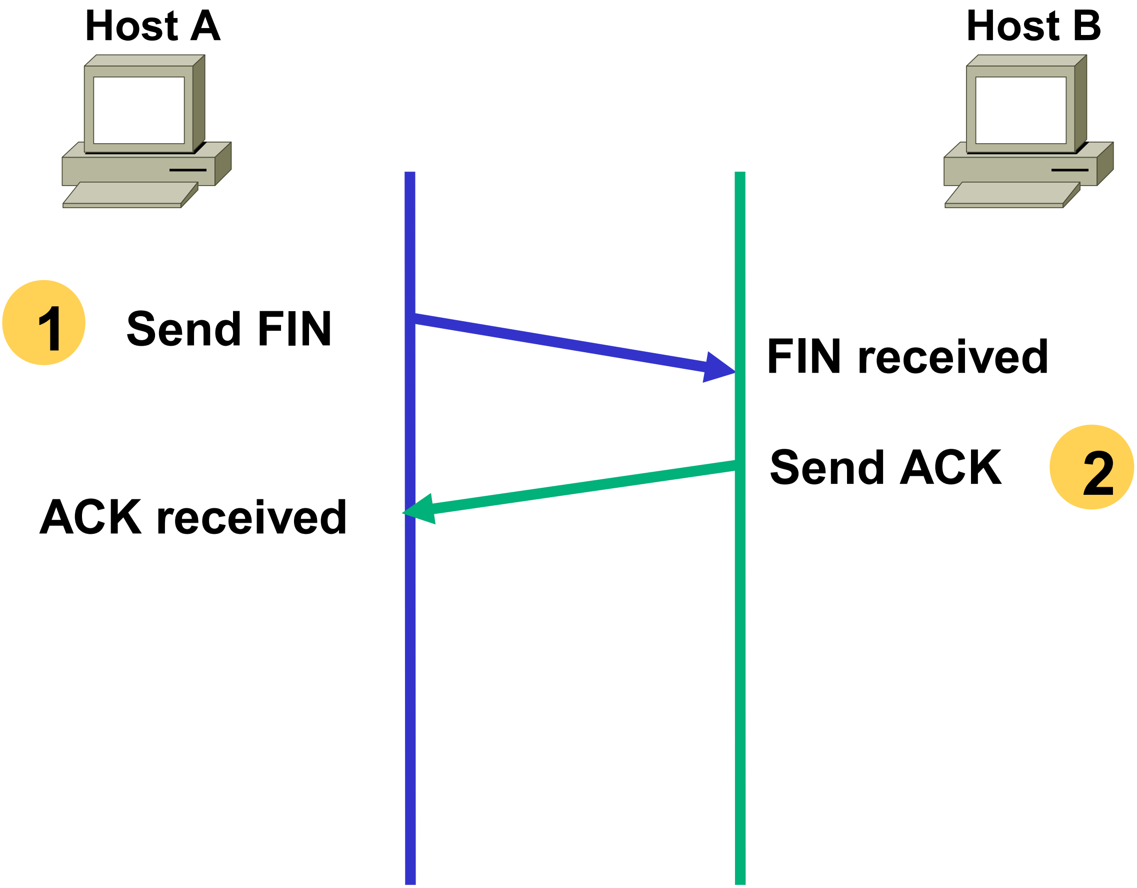
TCP Connection Closure

- ***Half-close*** is also possible
  - Only one side discontinue transmission
  - Most of the present applications do not use *half-close*

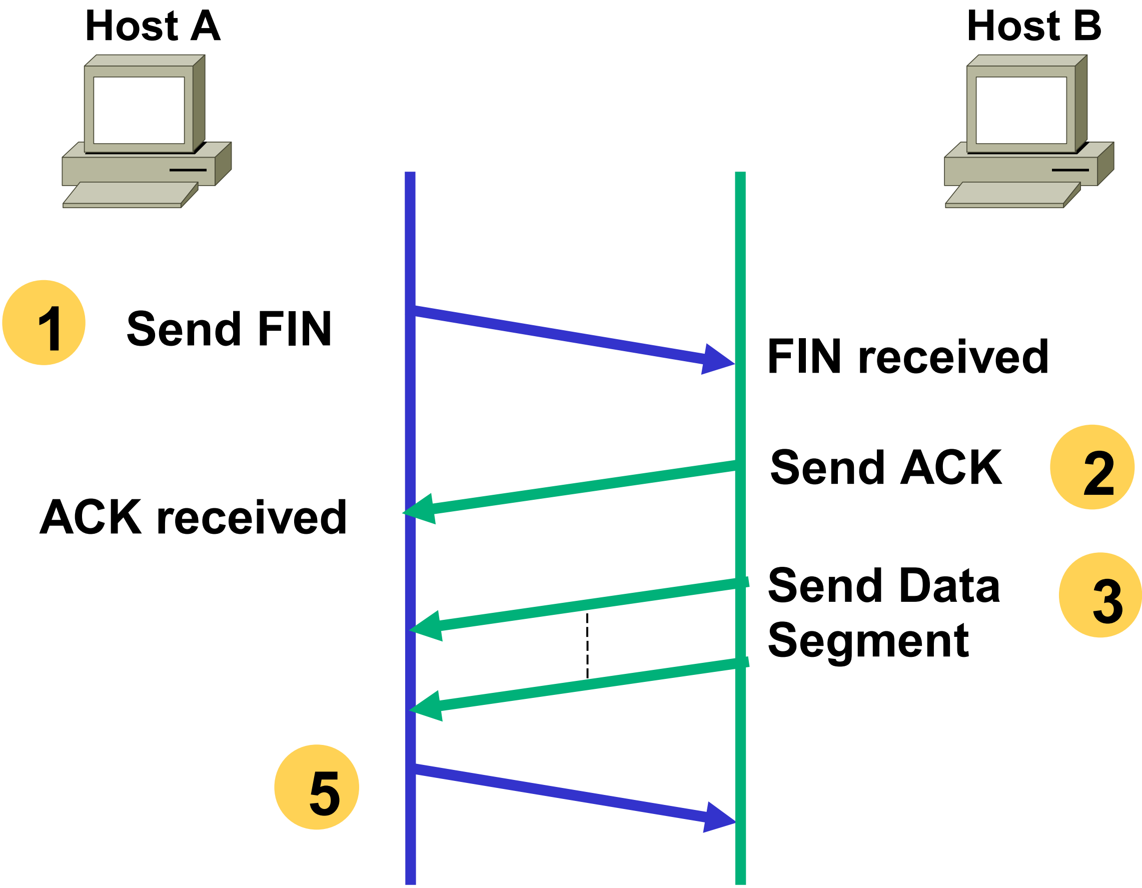
Half-Close



Half Close



Half Close


Host A
Host B
1
Send FIN
FIN received
Send ACK
2
ACK received
Send Data
Segment
3
5

Timeout during Connection Setup

- When initiator does not receive SYN-ACK
  - Resends SYN after some time
  - First retry after 6 sec
  - Second retry after 24 sec
  - Stops retrying after 75 seconds (Unix system)
  - These times are implementation dependent

MSS

- Maximum Segment Size
  - Exchanged with the initial SYN packets as an optional field
  - MSS does not appear in other packets
  - If MSS value is not received default value assumed is 536 bytes
  - 536+20IPhdr+20TCPHdr = 576byte IP datagram

MSS

- Larger the MSS, better it is for the protocol efficiency, provided it is not fragmented
- MSS may be at the most

  = MTU – 20 – 20 byte

  - For Ethernet = 1500-20-20 = 1460
- If destination is non-local, MSS normally defaults to 536
- MSS is configurable value

MSS

- When two sides announce different MSS, they normally settle down to the lower value.
  - This is not mandatory
  - Avoids fragmentations
  - Not necessarily eliminate fragmentations if intermediate links have even lower MTU
  - Use MTU discovery mechanism to avoid this

Non-delivery of TCP Segments

- Non delivery is indicated by non-receipt of ACK at sender side
- This may be caused by
  - Loss of packet
    - ❖Due to congestion
    - ❖Due to error in header
    - ❖Due to error in data
  - Loss of ACK
  - Delayed delivery by IP

Non-delivery of TCP Segments

- Non-receipt of ACK is decided by
  - Retransmission timer
  - Receipt of duplicate ack
- TCP assumes that the non-delivery is because of congestion
- Reduces the window size when the packet is declared as undelivered
  - Not an efficient way if loss of packet is due to a transmission error

Types of Connection Closure

- *Orderly release* – graceful shutdown
  - When closure initiated by applications
  - No loss of data
  - Using *FIN* segment
- *Abortive release*
  - Abrupt termination
  - Using reset (*RST*)

Variation in Connection Open and Close

- Simultaneous Opening
  - Both sides send SYN
  - Both sides respond with SYN-ACK
- Simultaneous Closure
  - Both sides send FIN
  - Both sides send ACK

RST

- Generated on receipt of an ***incorrect*** TCP segment
  - Packet does not belong to the referenced connection, determined by
    - ❖IP
    - ❖Port number
    - ❖Sequence number
- Generated on receipt of connection request to an nonexistent port
- Generated by application when it aborts the application

RST

- At sender side, any queued data is thrown away
- At the receiver side, APIs used should be able to inform application about the abortive release

PSH

- PUSH Flag
- Indicates to the receiver to send the data to the application without further delay
- Used in the interactive applications or during interactive operations
- Also used when last portion of the data is sent by sender stack

Half-open Connections

- One side abruptly terminates the session
- May be caused by
  - System crash
  - Machine powered off without graceful shutdown
- Server will not know the closure and will be in wait state
- Security risk

Interactive Data Flow

- ***Tinygrams***
  - Small data flow during interactive applications
  - Example: ***Rlogin***
- Nagle Algorithm (RFC 896)
  - TCP connection can have only one outstanding small segment that is not yet acknowledged
  - Small data is collected by TCP and sent together when *Ack* is received for previous small segment
  - Might cause problem when ASCII escape character is involved (special function keys)

Interactive Data Flow

- Repacketisation
  - Sending retransmission and next segment in the same segment
- TCP protocol spoofing
  - Used when delay is high, bandwidth is sufficent
  - Improves user experience and application performance
  - Used in VSATs

Sliding Window

- Start small – ***slow start***
- Grow exponentially
- Bound by upper limit of window size
- Reduces window size when encounter segment loss
- Increases window size again
  - Offered window size
  - Usable window size

Sliding Window

- *Slow Start*
  - *Rate of transmission depends on rate of receipt of acknowledgments*
  - *A flow control imposed by sender based on its assessment of congestion in the network*

Congestion Avoidance

- Indication of loss of packet
  - Timeout
  - Duplicate ACK
- On receiving duplicate ACK reduce usable window size to half
- If congestion is indicated by timeout, reduce usable window size to one, initiate slow start

Fast Retransmit

- When three or more duplicate ACK received, retransmit the un-ACKed packets without waiting for timeout of retransmission timer

Reading Assignment

- Congestion Avoidance
- Slow Start
- Fast Retransmit
- Fast Recovery

Congestion Avoidance

- Indication of loss of packet
  - Timeout
  - Duplicate ACK
- Two variables
  - ***cwnd*** (congestion window)
  - ***ssthresh***(slow start threshold)
- On congestion, ***ssthresh*** = ***cwnd***/2
- If congestion is indicated by timeout, ***cwnd*** is set to one, slow start

Thank you!