Transmission Control Protocol (TCP)

Outline

- TCP connections
- Segmentation and acknowledgement
- Flow control
- TCP header

Key features

- Transmission Control Protocol (TCP)
- Operates above IP to offer a number of additional features, and address issues not tackled by IP
- **Reliability**: Data transmitted is checked and re-transmitted where necessary
- **Multiplexing**: Two hosts can have multiple simultaneous 'conversations' without getting confused about which data belongs to which conversation
- **Flow/Congestion Control**: A receiving host/network can only accept and process a given amount of data at once, so the sender must control how fast they send data

Multiplexing

- How does an host running multiple networked applications differentiate between messages?
- TCP conceptually divides the communications a host can be involved with into **ports**
- This allows two hosts to perform several simultaneous communications without getting data confused between them
- A port is identified by a number, used in each message between hosts so that they know which communication the message belongs to
- A few ports are reserved for specific application protocols, e.g.
  - FTP: 20/21
  - SMTP (e-mail): 25
  - HTTP: 80
  - HTTPS: 443

Connections

- In using TCP, hosts must establish a **connection**
  - Data can only be sent during the connection
  - Requires connection setup/teardown
- Why?
  - To exchange 'extra' information between hosts
  - Reliability
  - Resource reservation to ensure quality of service
  - Flow/congestion control
- A TCP connection is one kind of **session**, and many other protocols use sessions

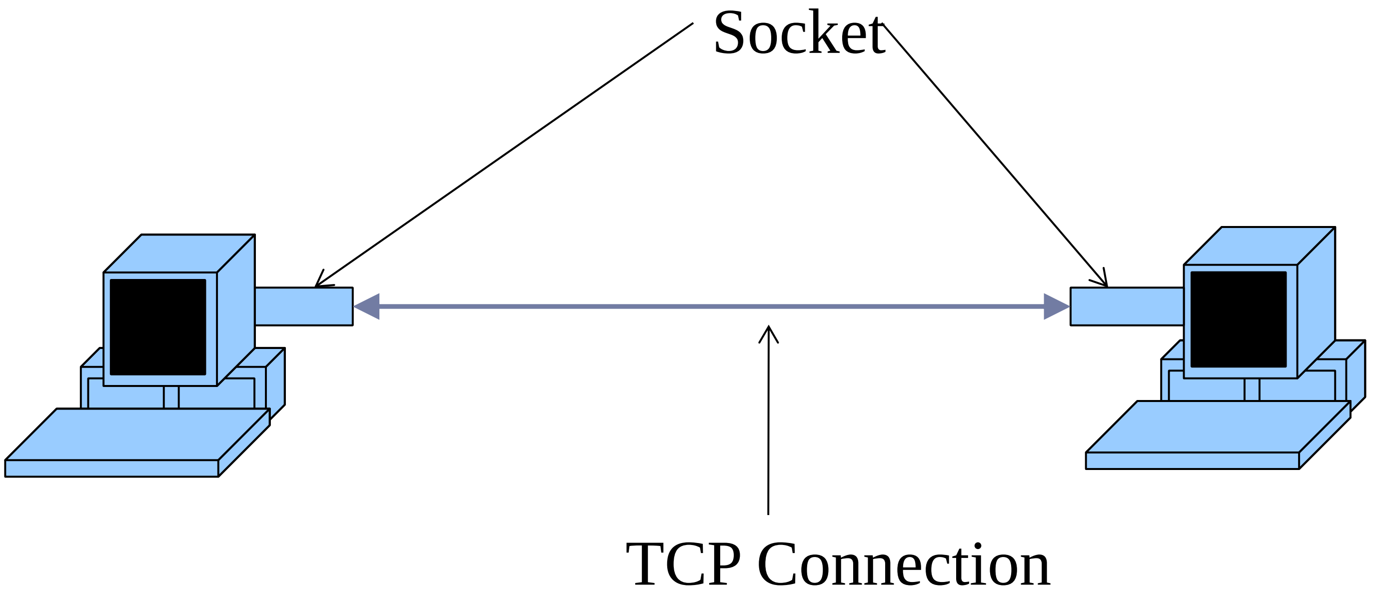
Sockets

- A **socket** is the combination of a host's IP address and a port number

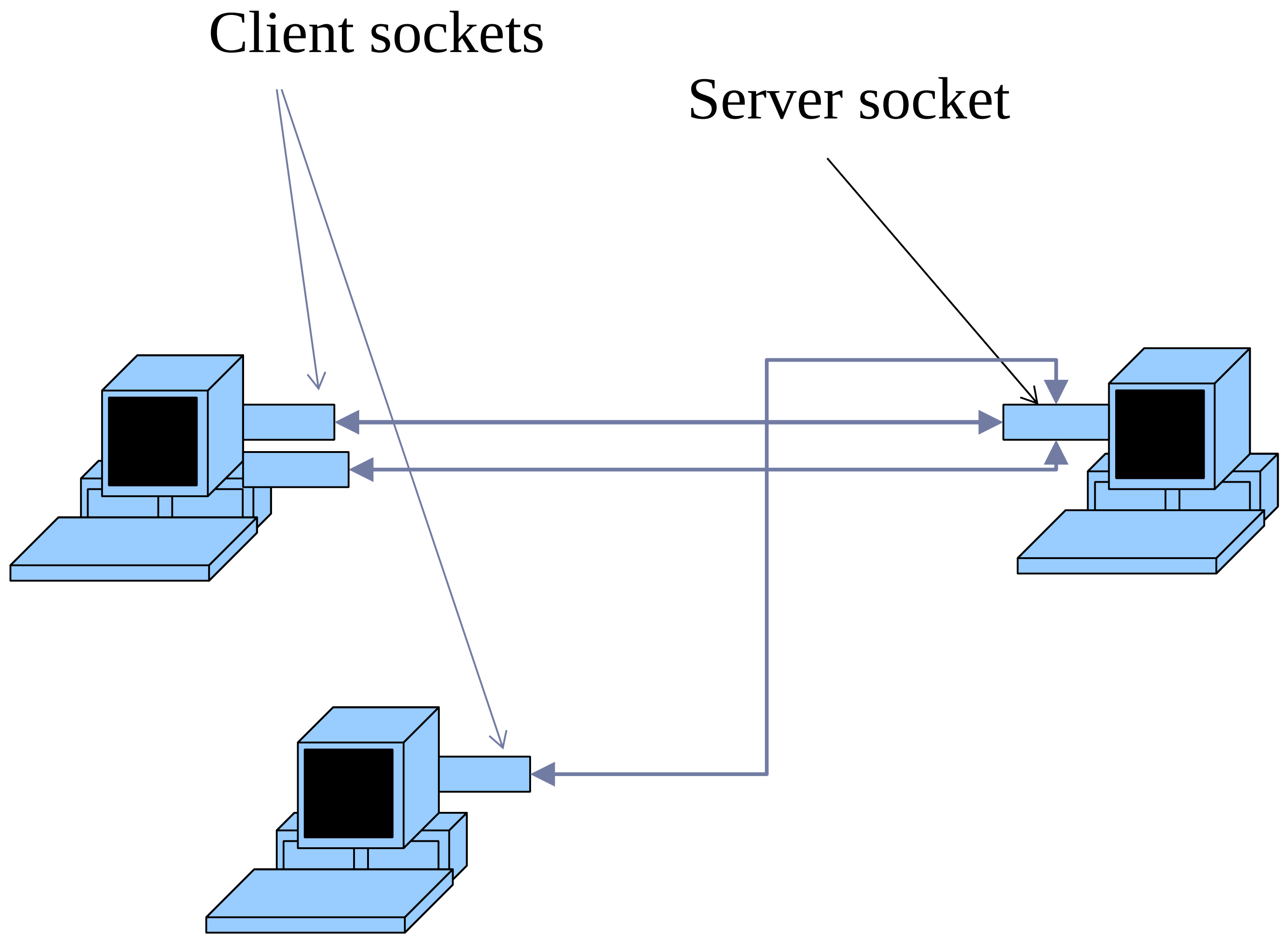
  137.73.9.232:8080

- Every communication in TCP is between two sockets, i.e. two hosts using particular ports.
- Initially, a server will LISTEN on a particular socket.
- Multiple clients can connect to the **same** server socket from different client sockets.
- Once a pair of sockets is connected, data can be sent in both directions between client and server:
  - Connections are **full-duplex**

TCP connections



TCP connections



Client and server

- **Server**: a process that is ready to accept connections on one or more ports
  - The server signals that it is ready to receive connections on a port by informing the TCP software on the host: a Passive OPEN request
  - For example a web server application running on www.google.com
- **Client**: a process that communicates with a server
  - A client signals that it wishes to connect to a server's socket by sending a message to the server: an Active OPEN request
  - For example, a web browser running on your local PC
- Both processes can reside on the same or different hosts.

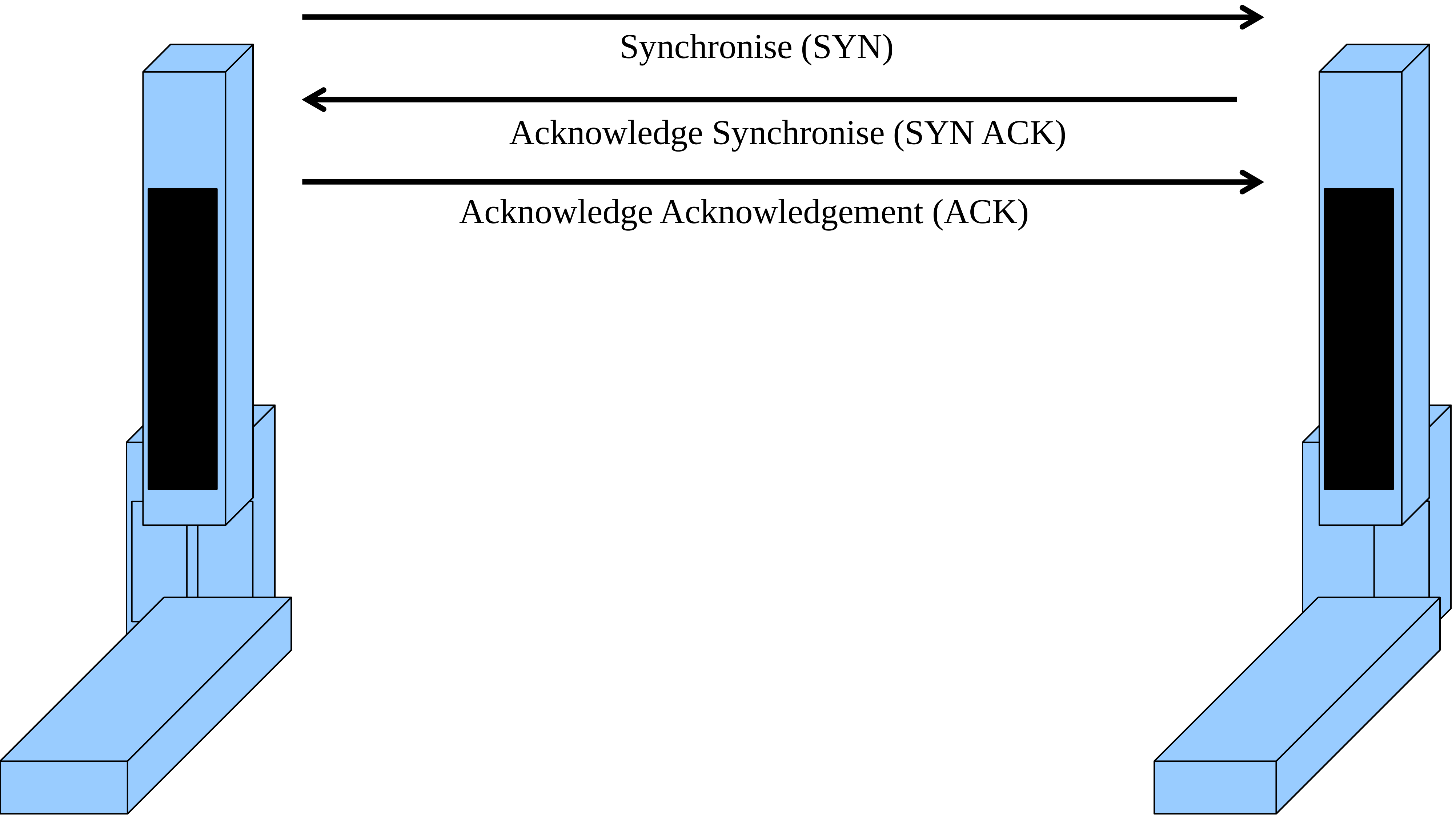
Connection set-up

- A **handshake** is a sequence of messages sent between a client and server to set up the connection before the message data is sent
- The handshake involves three steps:
  - Client sends a synchronise message to the server
  - Server sends a message back acknowledging the synchronise, and giving permission for communication to take place
  - Client sends a message acknowledging the acknowledgement
- The processes can then start sending data to each other.

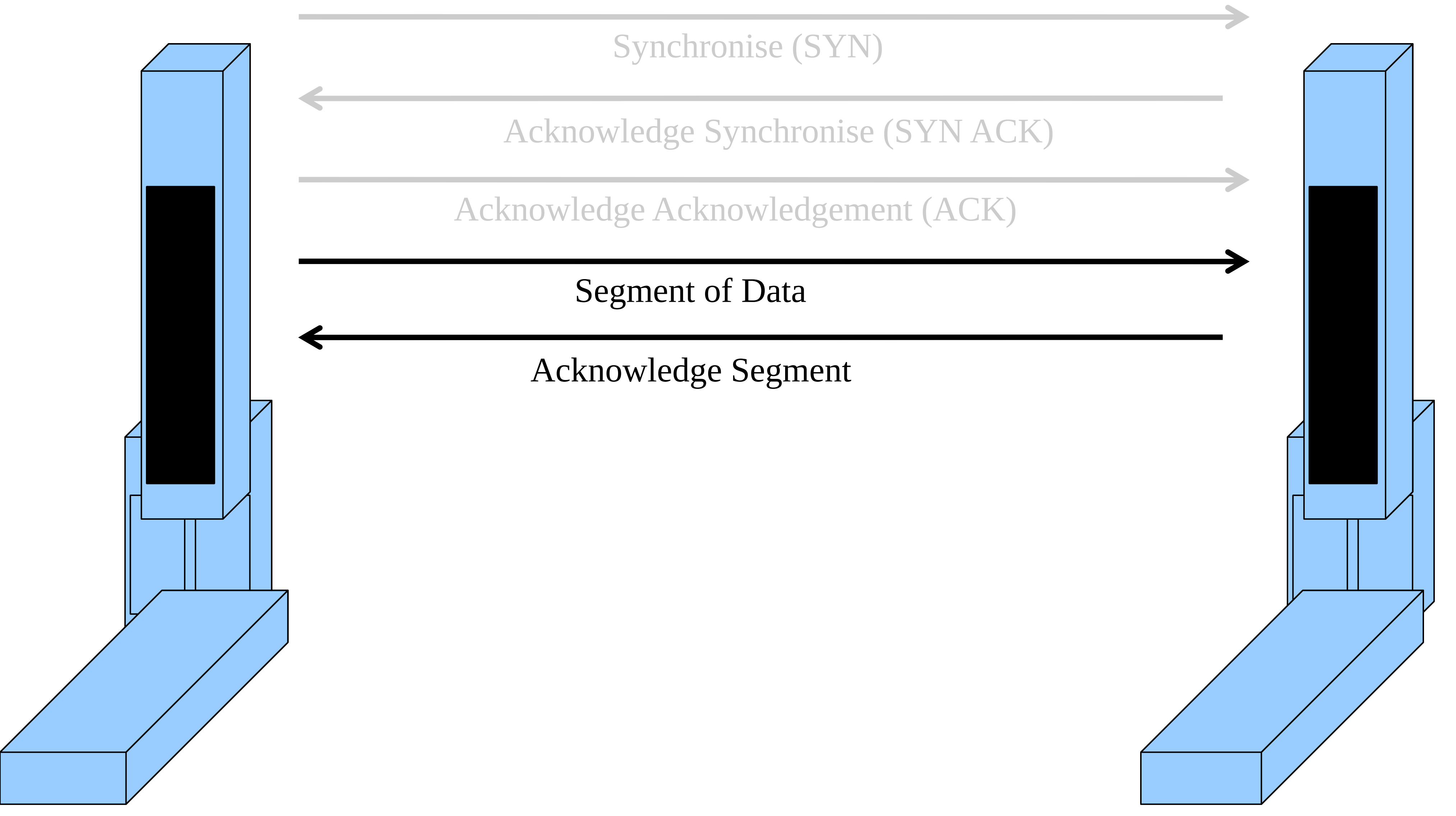
Connection tear-down

- A handshake is also used to close the connection
  - When a sender has finished and wishes to close the connection it sends a finalise message
  - The receiver responds with an acknowledgement of the finalise
  - Finally, the sender responds with an acknowledgement of the acknowledgement
  - Because connections are full-duplex, client and server can close their side of the connection independently.
  - Connections in which only one end-point has closed the connection are in a half-closed state.

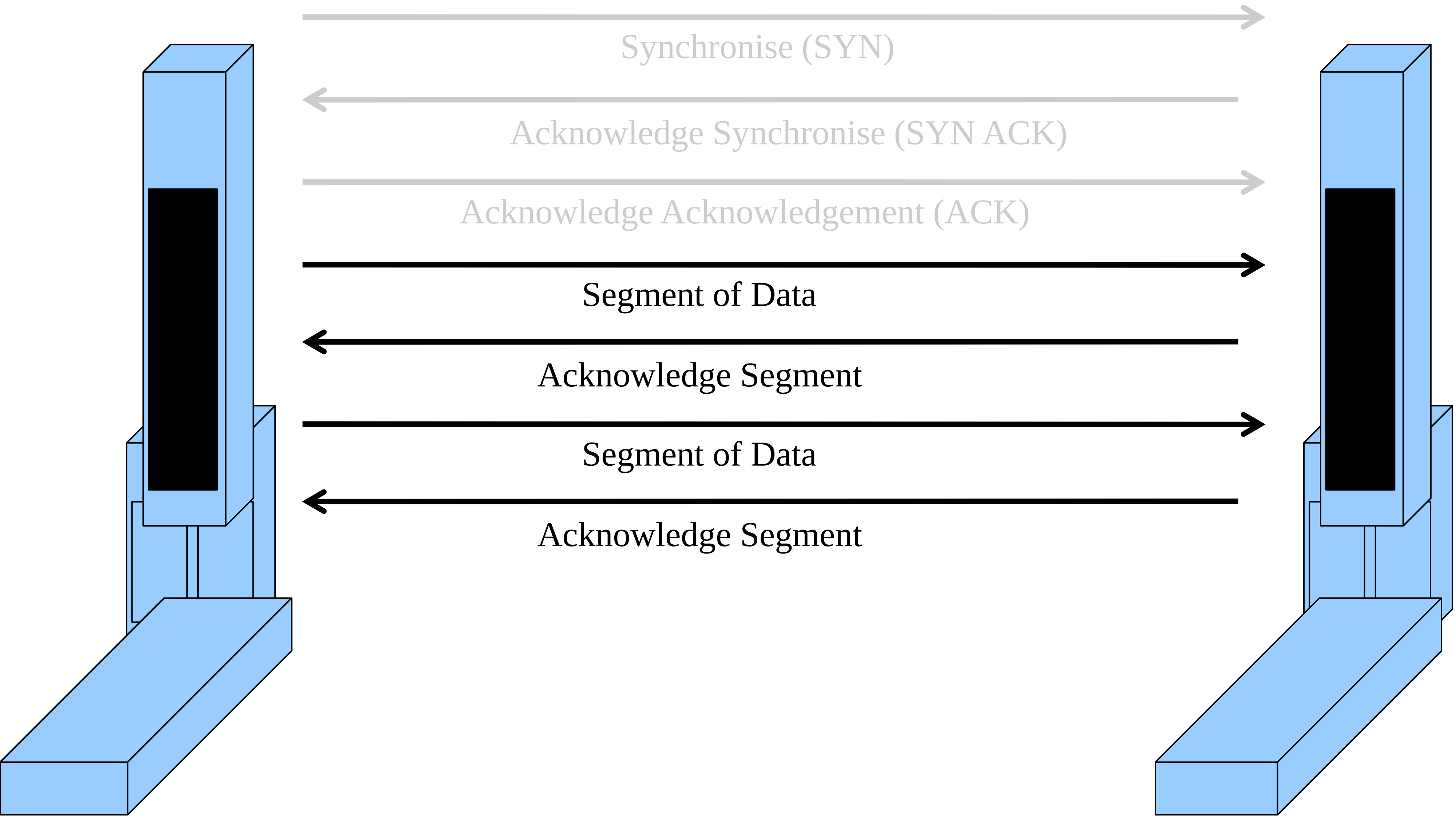
TCP communication sequence



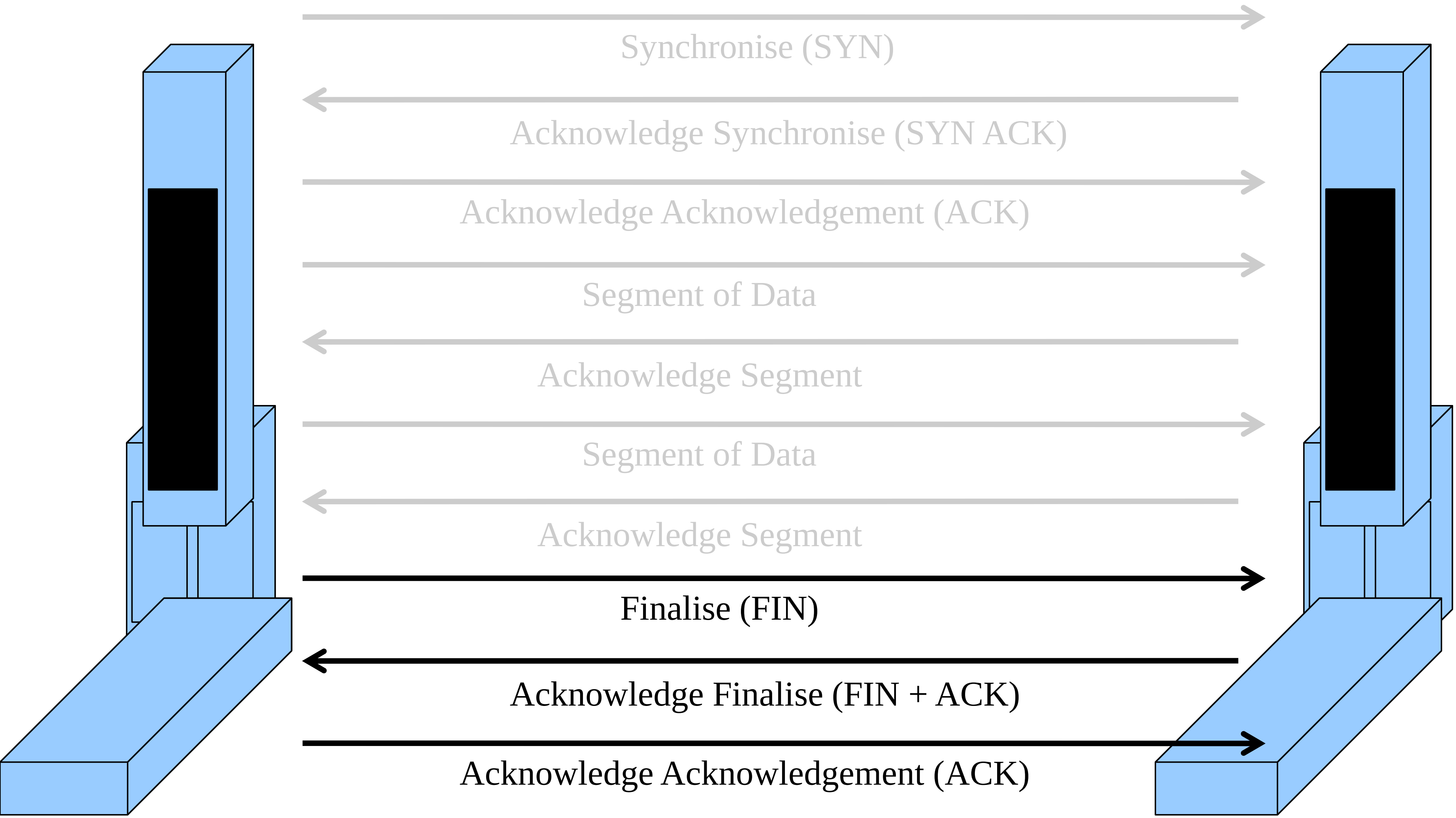
TCP communication sequence



TCP communication sequence



TCP communication sequence



Flow control

- A host can only receive and process data at a certain rate
- At some point its buffers get full and it would have to either overwrite unprocessed data or ignore incoming data
- To resolve this, TCP allows the sender to tell the receiving end how much data it can handle
- The sender will then reduce or increase the rate at which it sends data to match the server

Segmentation

- One part of flow control is to split the message to be sent into separately transmitted **segments**
- This is like fragmentation in IP, but is independent and for different reasons
- In IP, fragments are used to not exceed the limits of the physical networks and network access layer protocols
- In TCP, segments are used to not exceed the limits of the receiving host, flow control, and to aid reliability of communication

Sequence numbers

- Every byte in a message sent from a client to a server has a **sequence number**
- For a given connection and host, there is an **initial sequence number (ISN)** for bytes it sends
- The 1st byte of the message has sequence number ISN+1
- The 2nd byte has sequence number ISN+2
- A segment will contain all data within a range of sequence numbers: ISN+a to ISN+b
- Client communicates its ISN in synchronisation

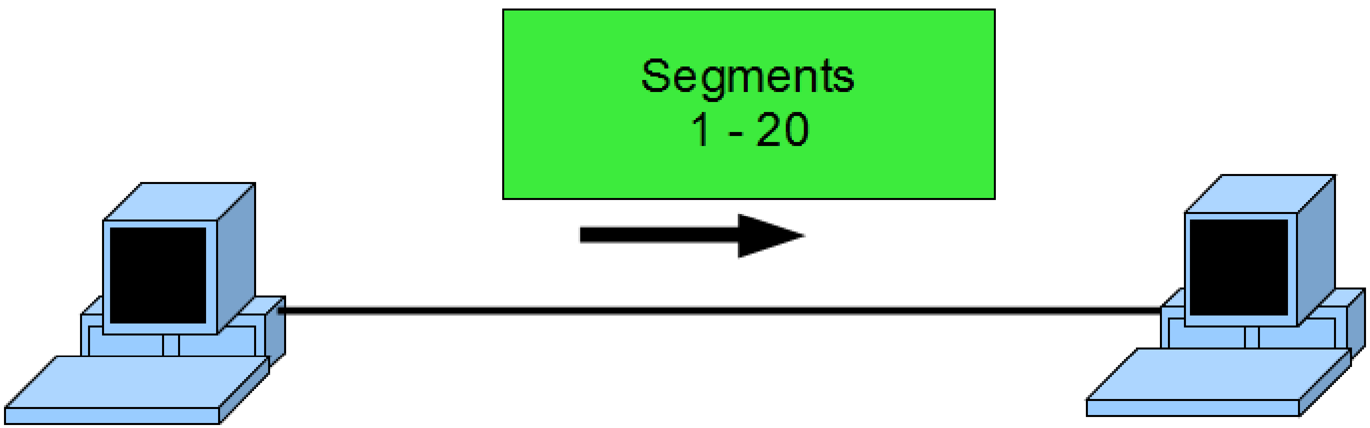
Reliability

- The main method by which TCP provides reliable communication is for the receiver to send back an **acknowledgement** back to the sender for every segment it receives
- A sender (whether client or server) uses acknowledgements to determine whether a segment has been lost
- If it believes a segment is lost, it will re-send it

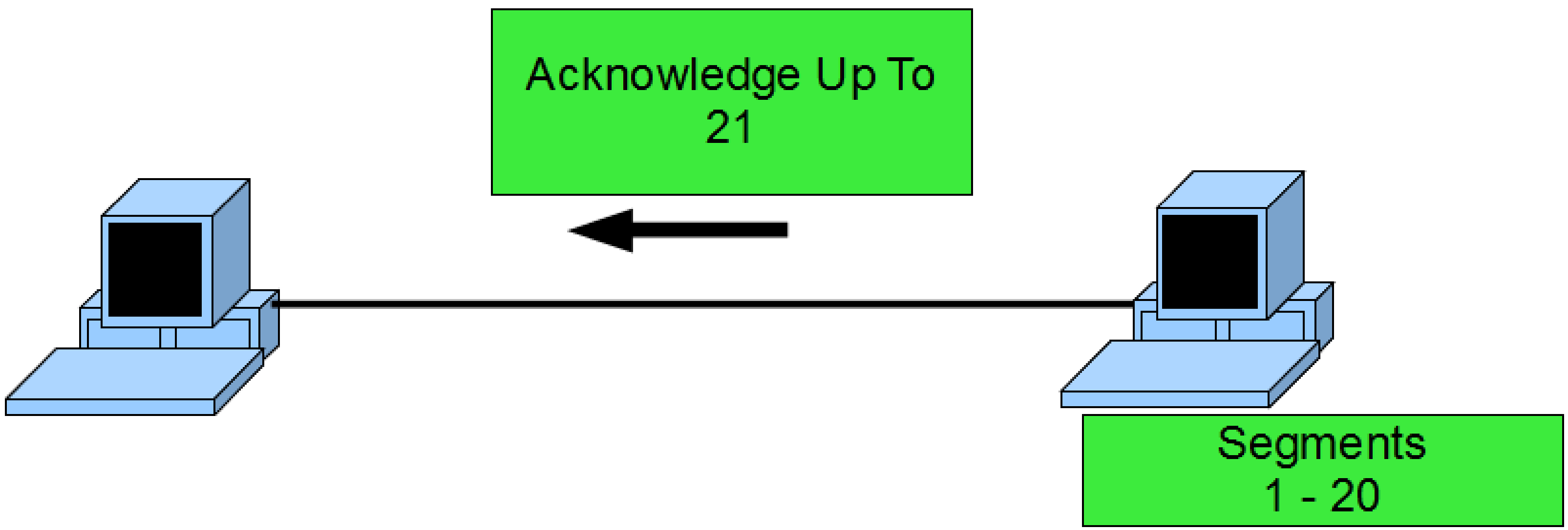
Acknowledging sequences

- An acknowledgement states that the receiver has received all data in the message before a given sequence number
- If the receiving end has received segments with sequence numbers (1 - 20) and (21 - 30), it will send an acknowledgement with sequence number 31
- The segments may be lost
  - Server receives segments (1 - 20) and (50 - 60)
  - Sends an acknowledgement with sequence number 21, as it has all data up to there

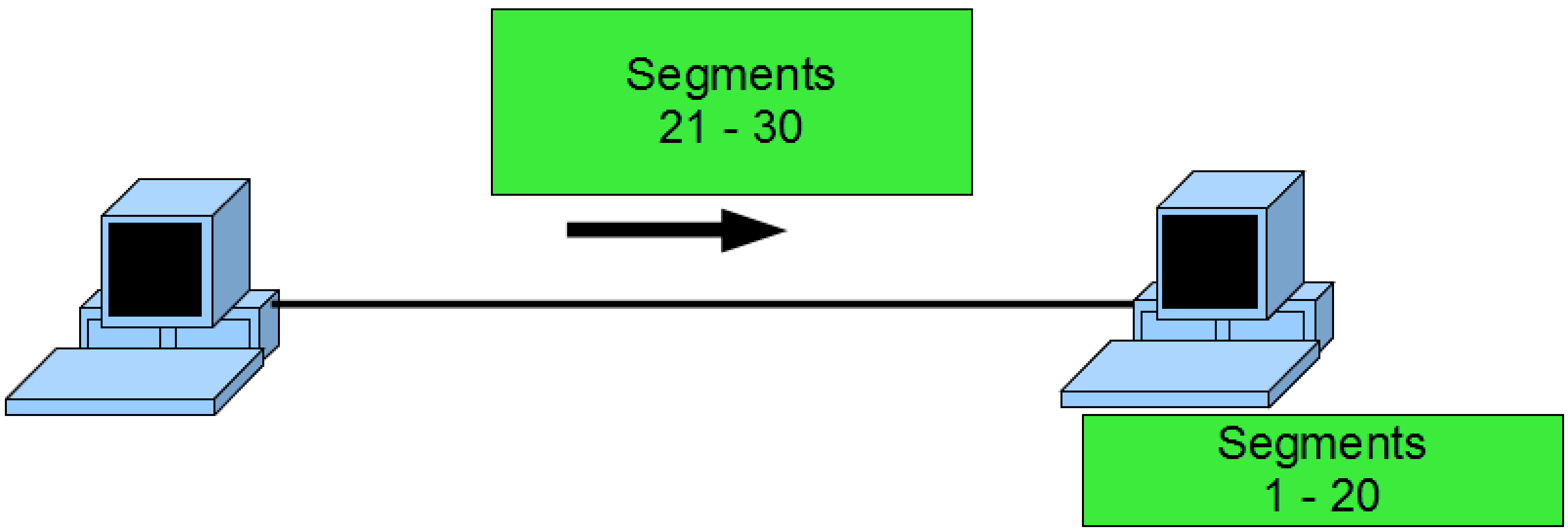
Acknowledgements



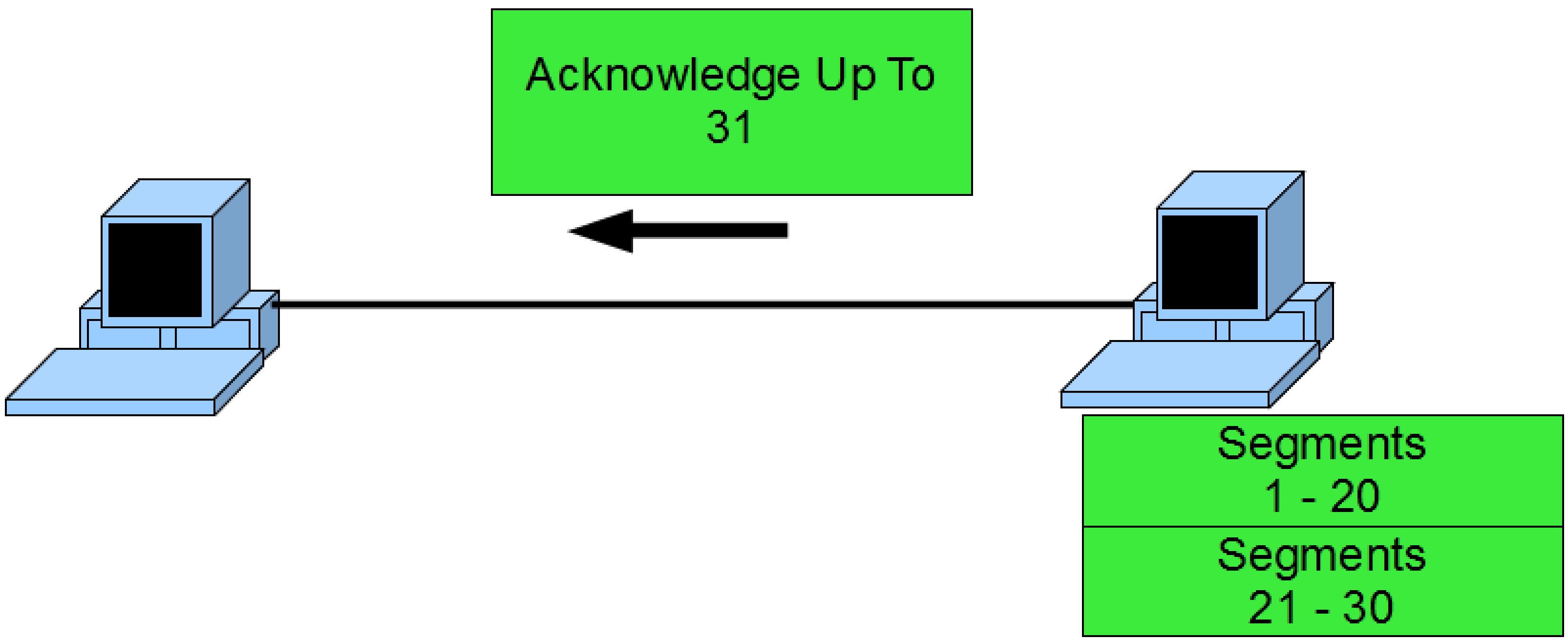
Acknowledgements



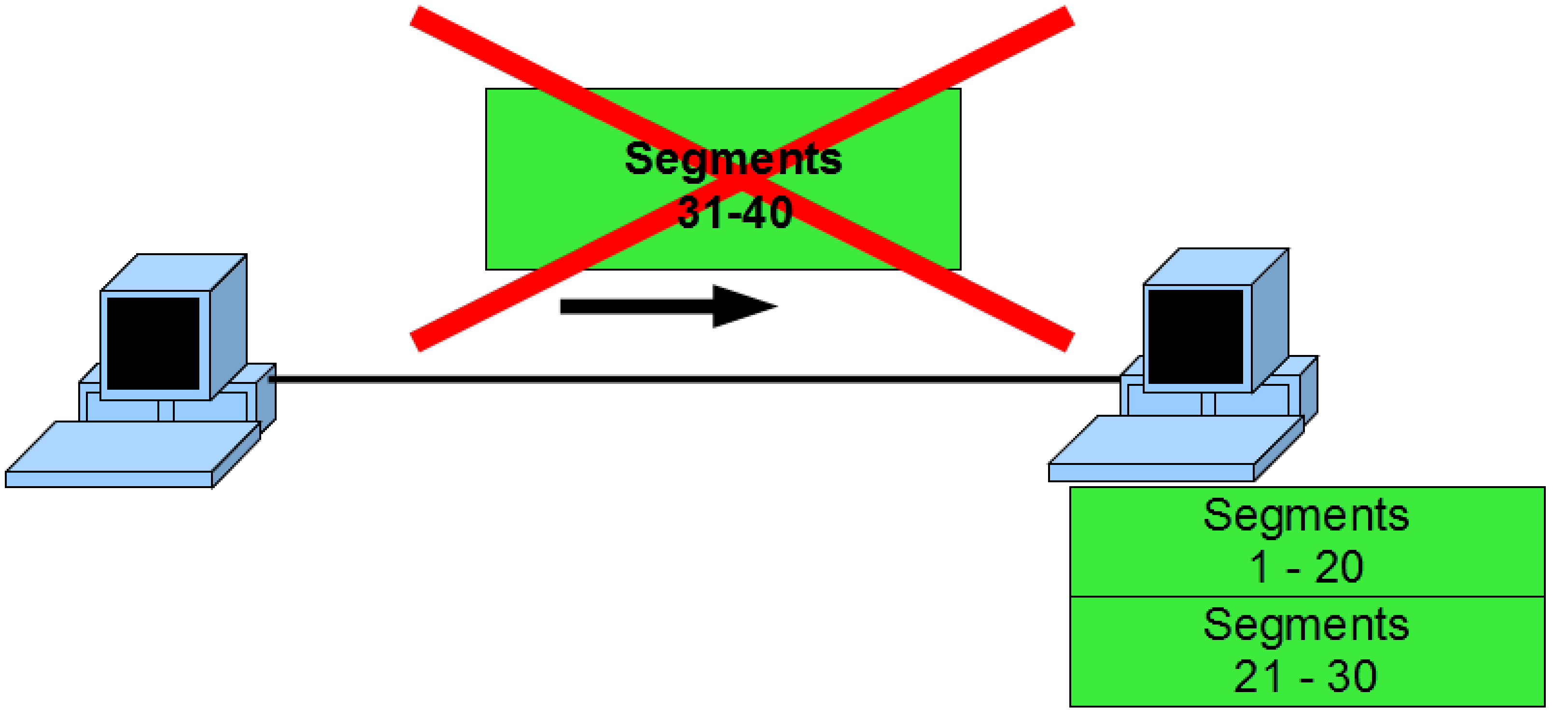
Acknowledgements



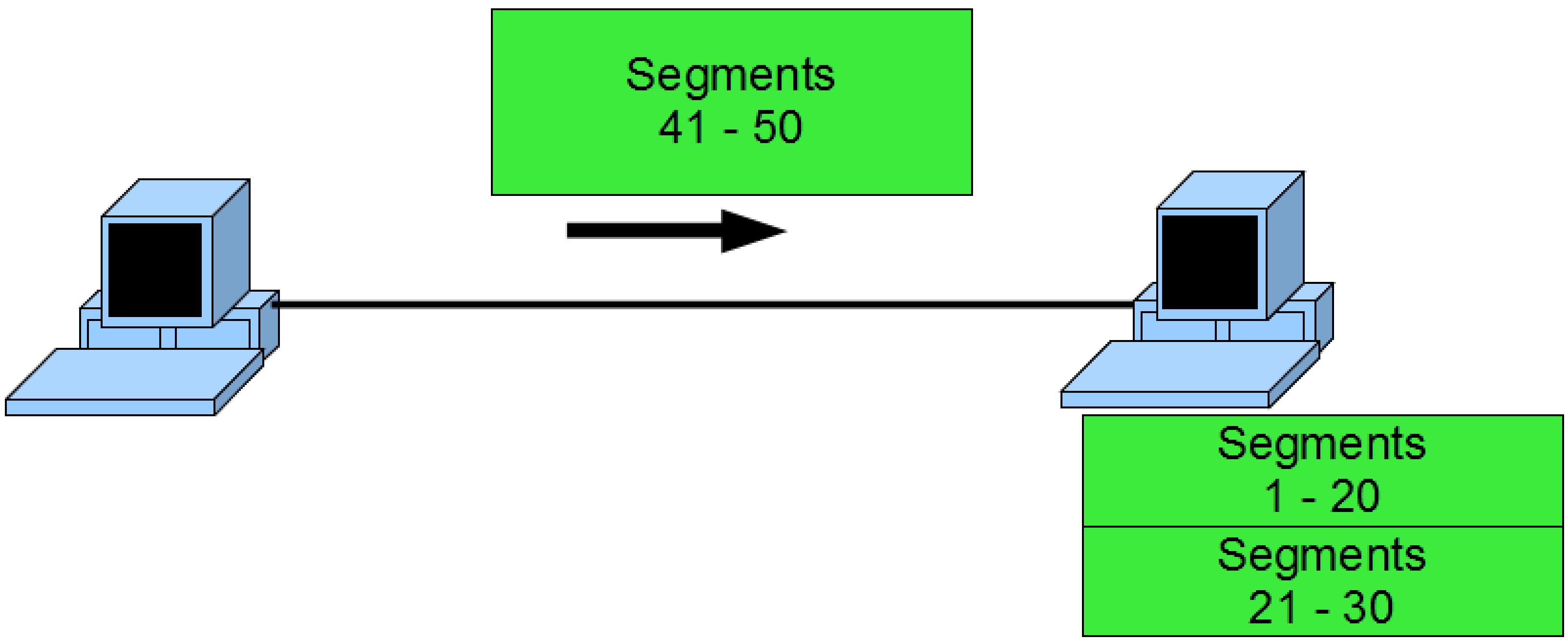
Acknowledgements



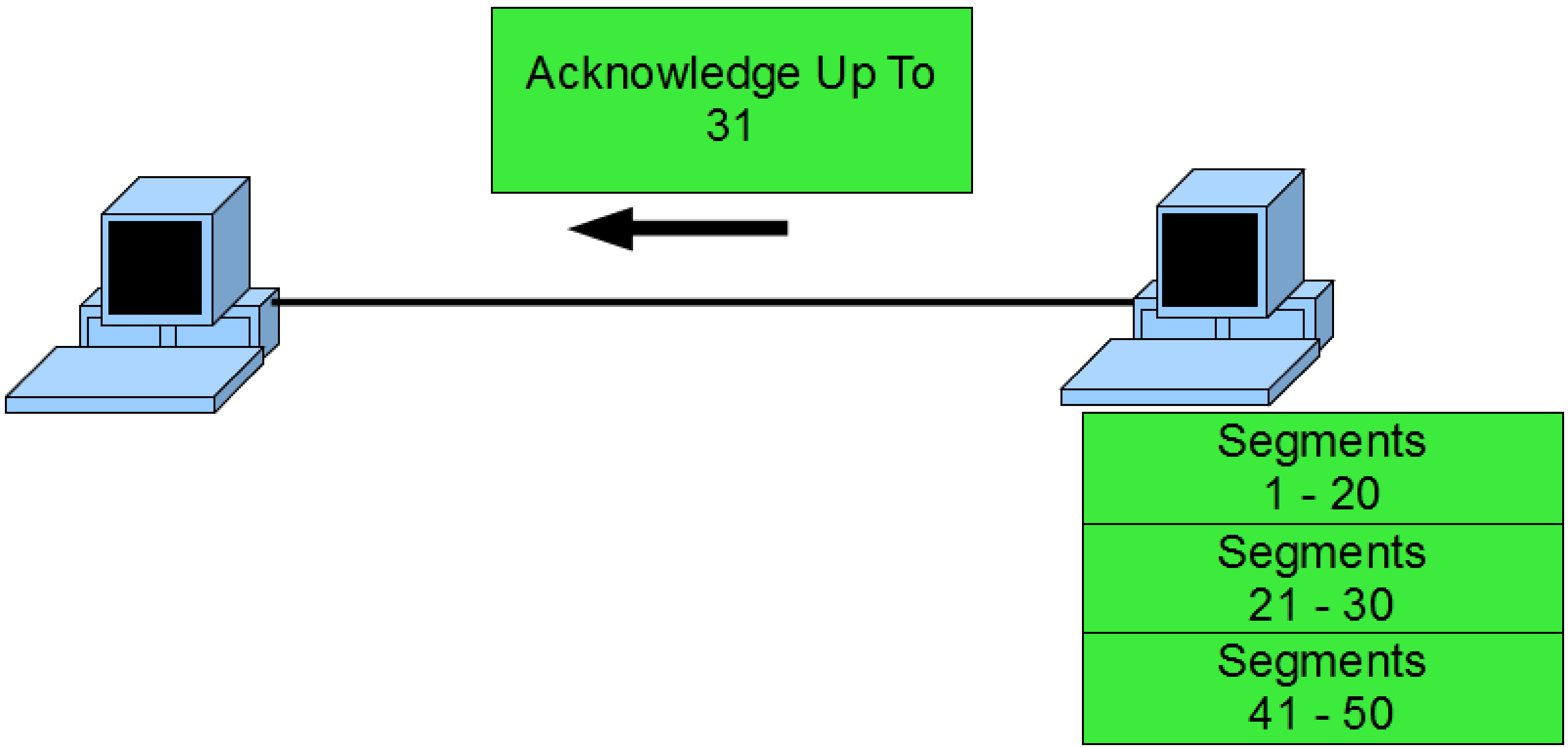
Acknowledgements



Acknowledgements



Acknowledgements



Resending strategies

- When will a client know to resend a segment?
- **Time-out**: A client may resend a segment if it has not received an acknowledgement for it after a given period of time after sending the segment
- **Repetition**: A client may resend a segment, X, if it receives acknowledgements suggesting other segments are being received but X has not, e.g.
  - If the server receives segments

    (1-20), (30-40), (41-44), (45-50)
  - then it will send an acknowledgement with sequence number 21 for each segment received: 4 times acknowledgement up to 21
  - The repetition suggests to the client that the segment starting 21 is lost and needs resending

Maximum segment size

- A **maximum segment size** (MSS) for a connection is maximum amount of data allowed in one segment transmitted to a particular socket.

- It is specified by each connection end-point during synchronisation handshake
- Note that the client and server can have different MSS for the same connection.

Window size

- Because of the limits on the processes receiving data, segments should often be smaller than the end-point's maximum segment size
- The amount of data that the receiving end can process is called the **window size**
- No segment sent by the client should be larger than the window size, as data is then lost
- The receiver can adjust the window size during communication if it can accept more or less data, through a message to the sender

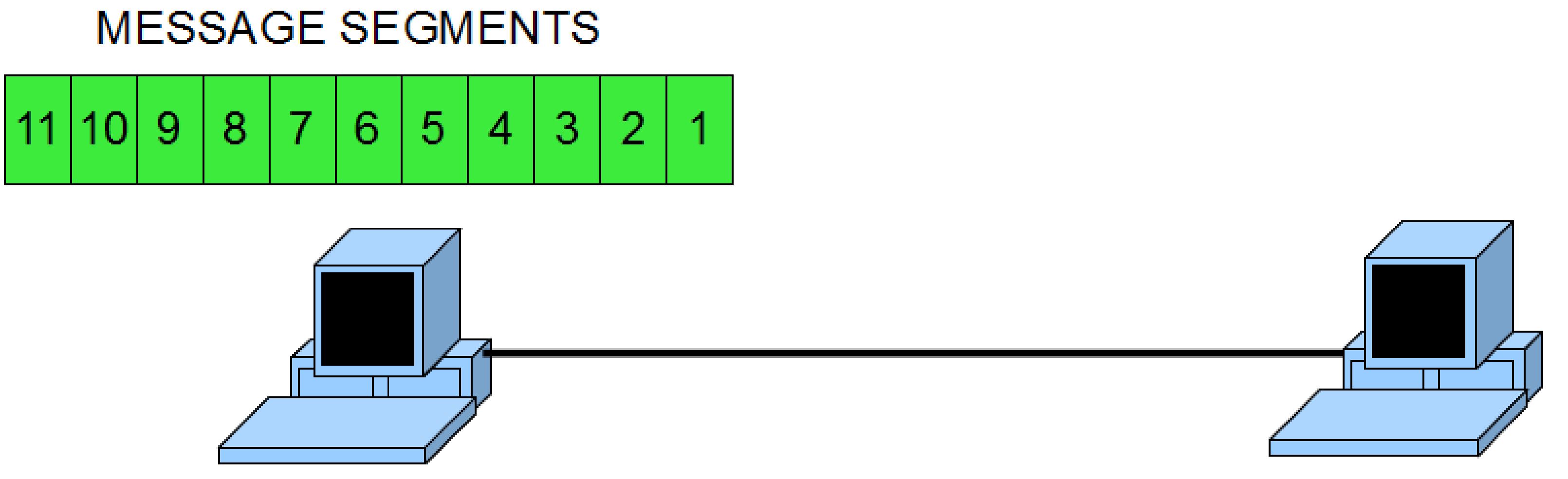
Segment sizing problem

- However, it is more complex than that...

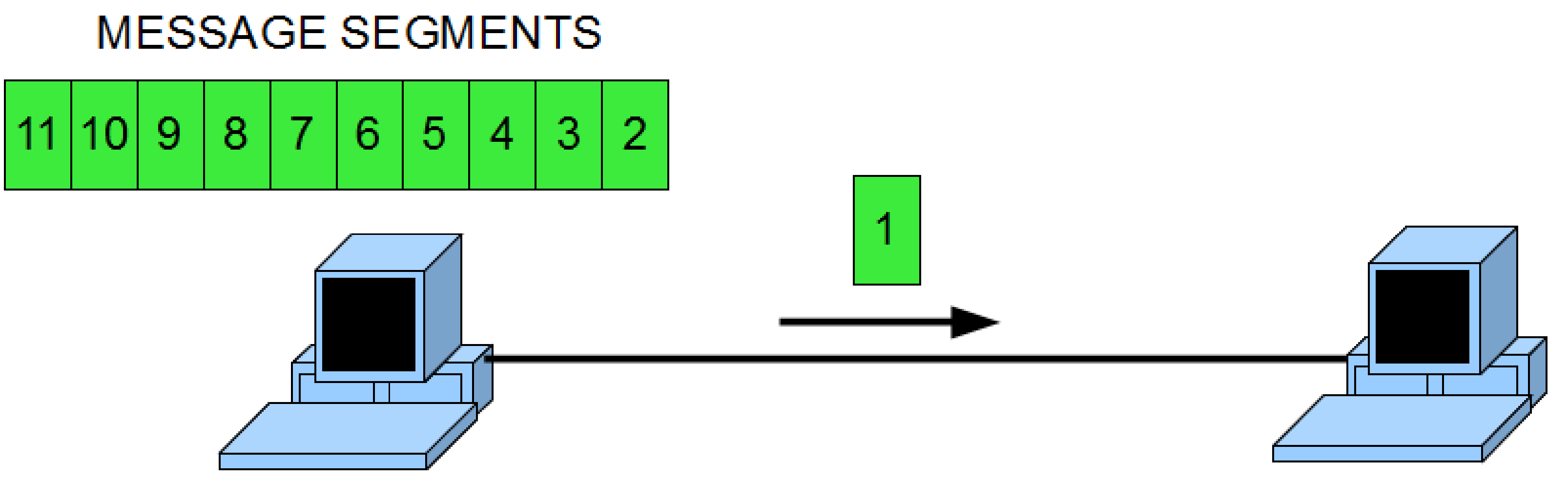
- The amount of data a process can accept depends on how fast it is processing the already received data
- The sender can only be sure that data which has been acknowledged has actually been received

Flow control

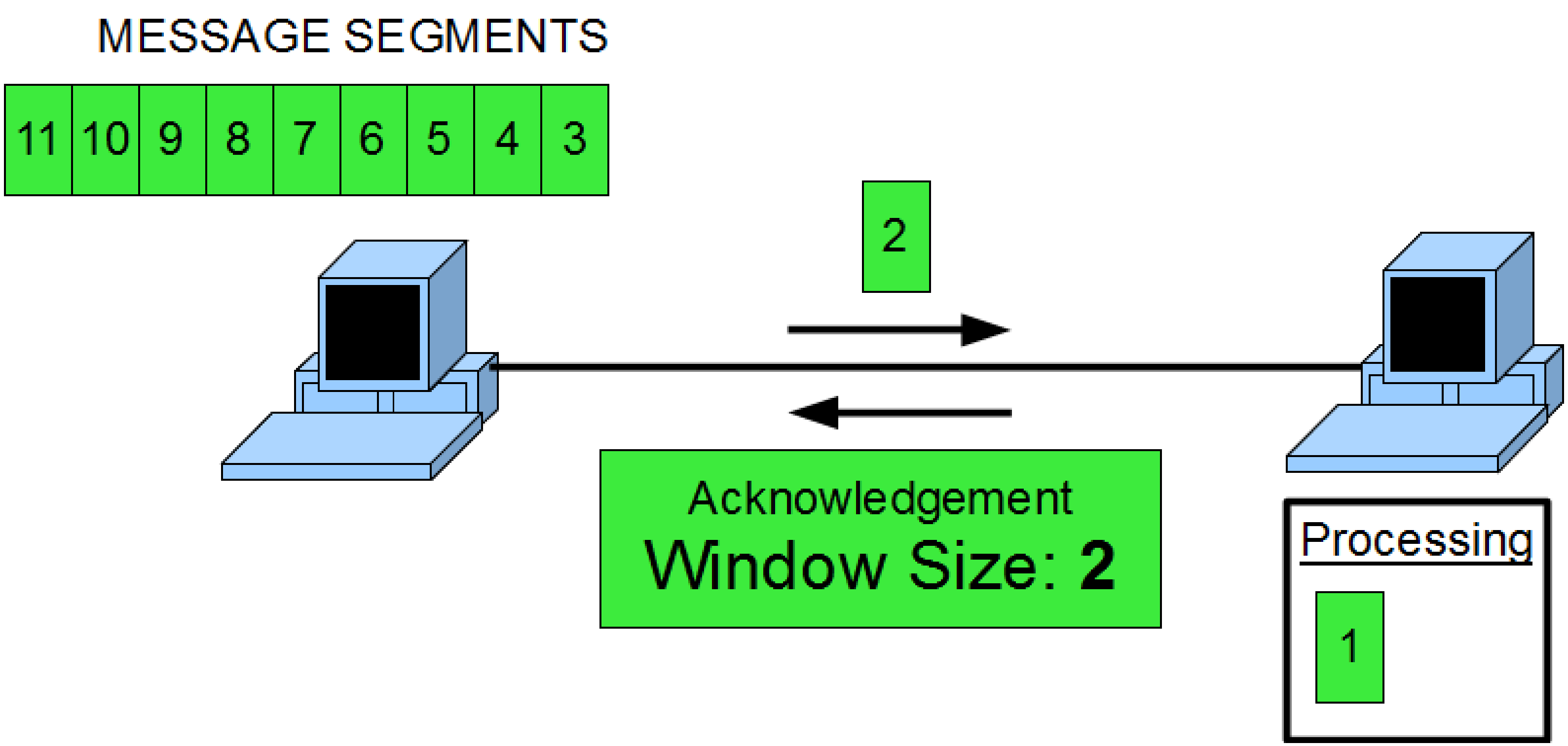
Flow control



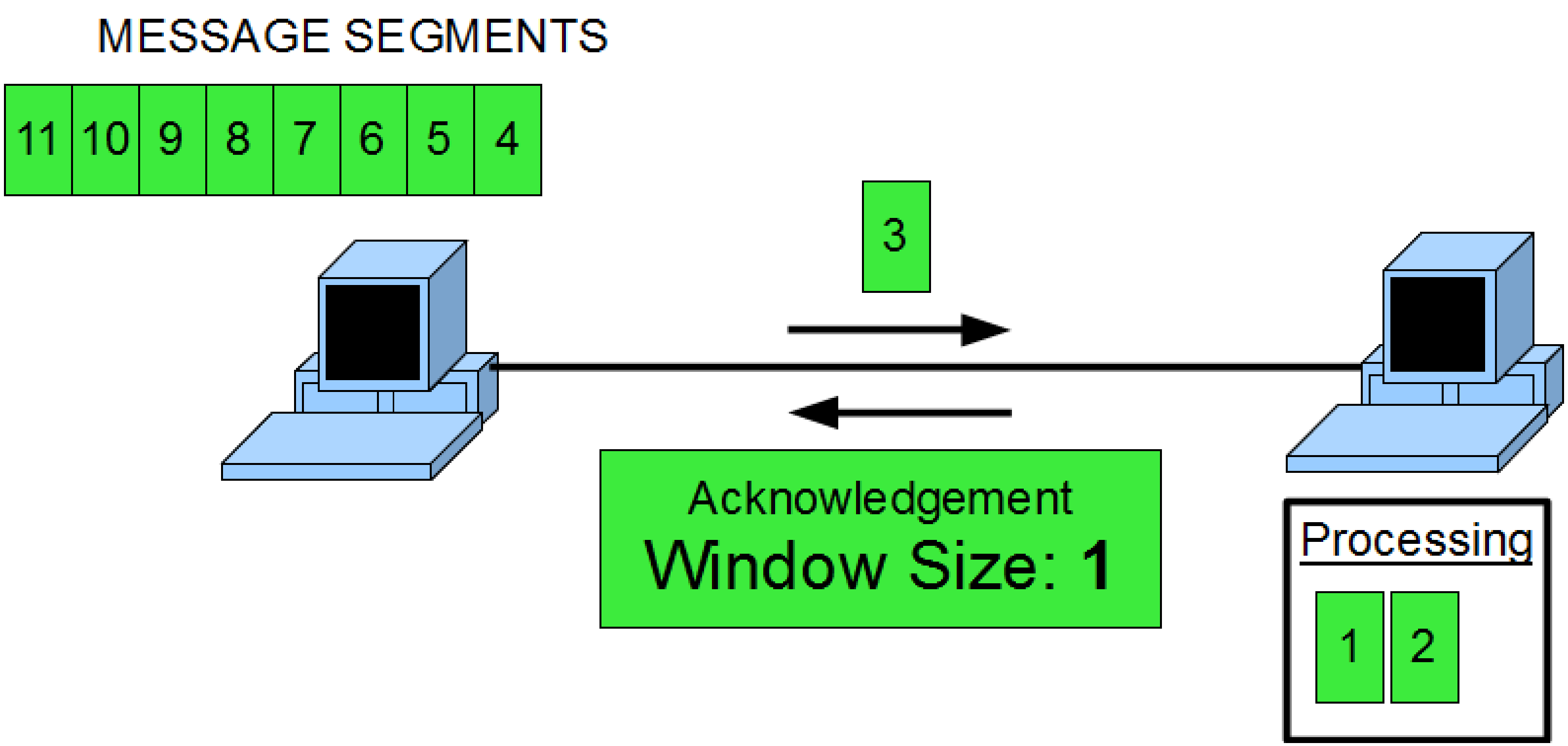
Flow control



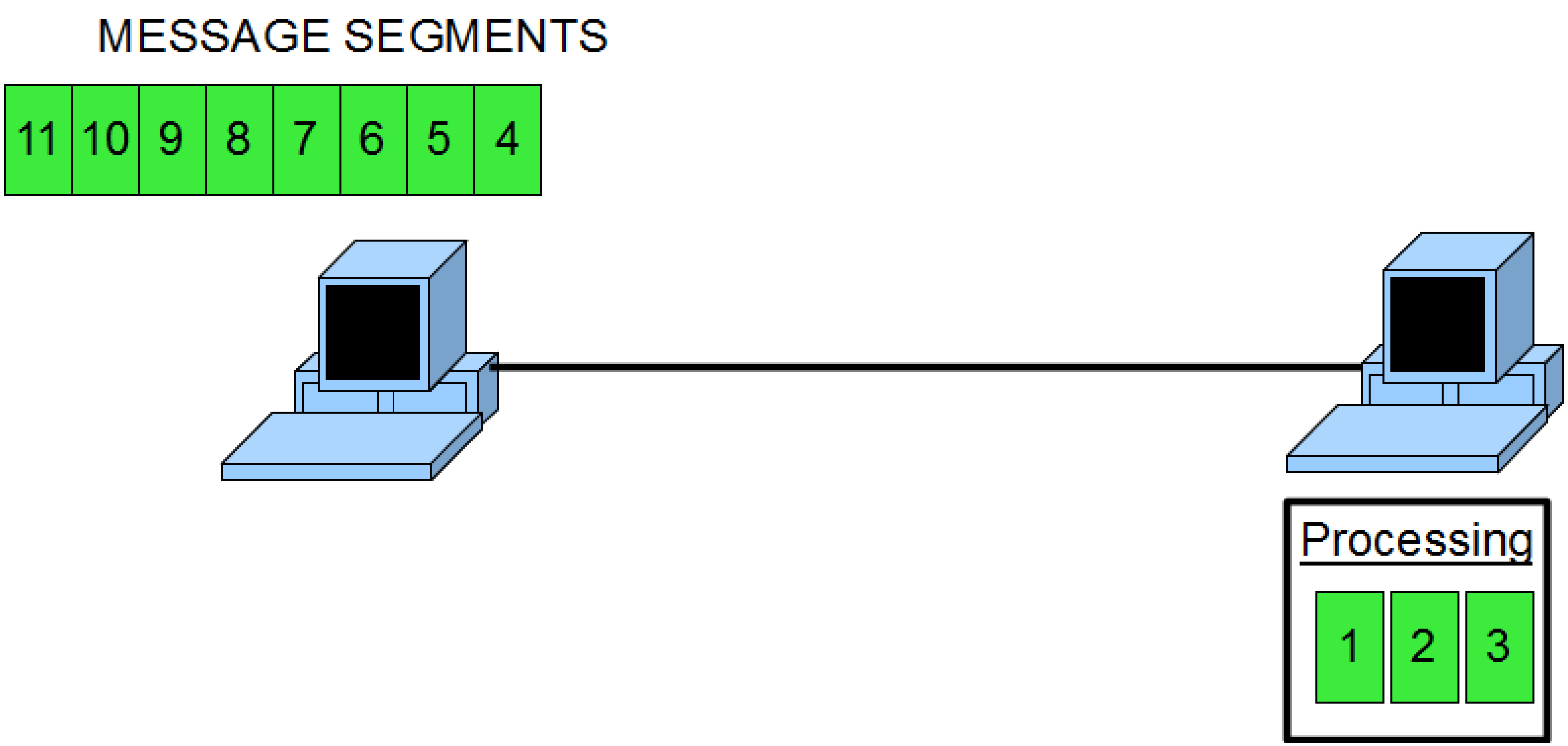
Flow control



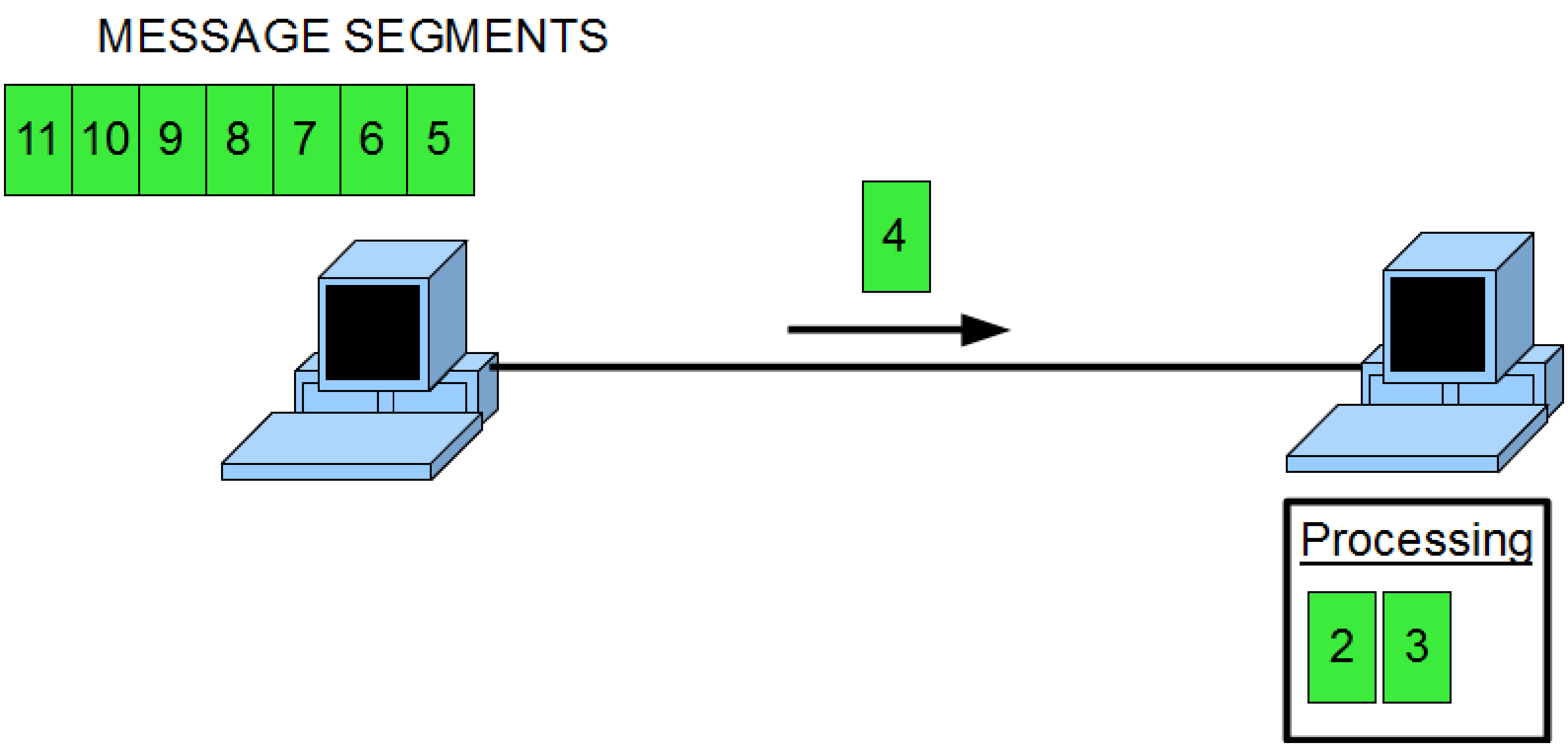
Flow control



Flow control



Flow control

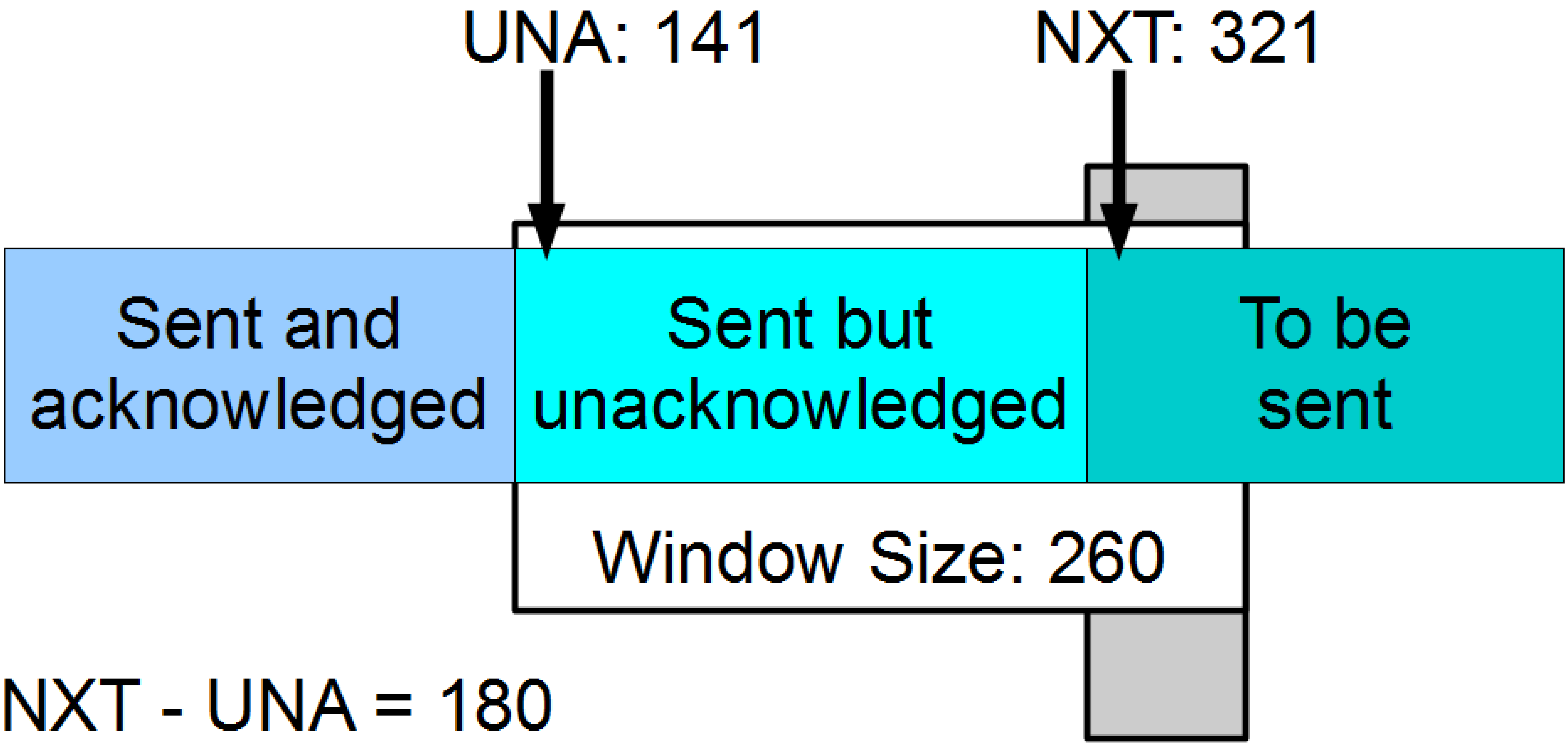


Segment sizing algorithm

- The sender keeps track of three variables:
  - Sequence number of the first byte sent but not yet acknowledged (UNA)
  - Sequence number of next byte to be sent (NXT)
  - The window size (WND)
- The **usable window** is the range of bytes that have not yet been sent, but the client believes the server is ready to receive
- The usable window size is calculated:

$$UNA + WND - NXT$$

Usable send window



NXT - UNA = 180
WND - 180 = 80
UNA + WND - NXT = **80**

Usable Window =
UNA + WND - NXT

Silly Window Syndrome

- If the receiver adjusts the window size to be too small, bandwidth usage becomes very inefficient
- This is because lots of very small segments will be sent and there will be an acknowledgement for each
- This is called silly window syndrome

Nagle’s algorithm

- Both sender and reciever can help tackle problem
- Sender’s strategy: not send further data until either:
  - All sent has been acknowledged, or
  - The data to be sent reaches the MSS

- The receiver’s strategy is to, as it becomes able to accept more data, not tell the client about this larger window size until it reaches either:
  - MSS
  - Half the server's maximum buffer size

Push

- Because of the flow control strategies, there is a delay between the client having data available to send and the client's TCP software sending it
- In some cases, this is unacceptable, and a client can request that all the data ready to send should be sent
- This is called a PUSH request
- The sender informs the receiver that the data being sent has been pushed
- In practice, in modern TCP/IP implementations, the PUSH flag is always set.

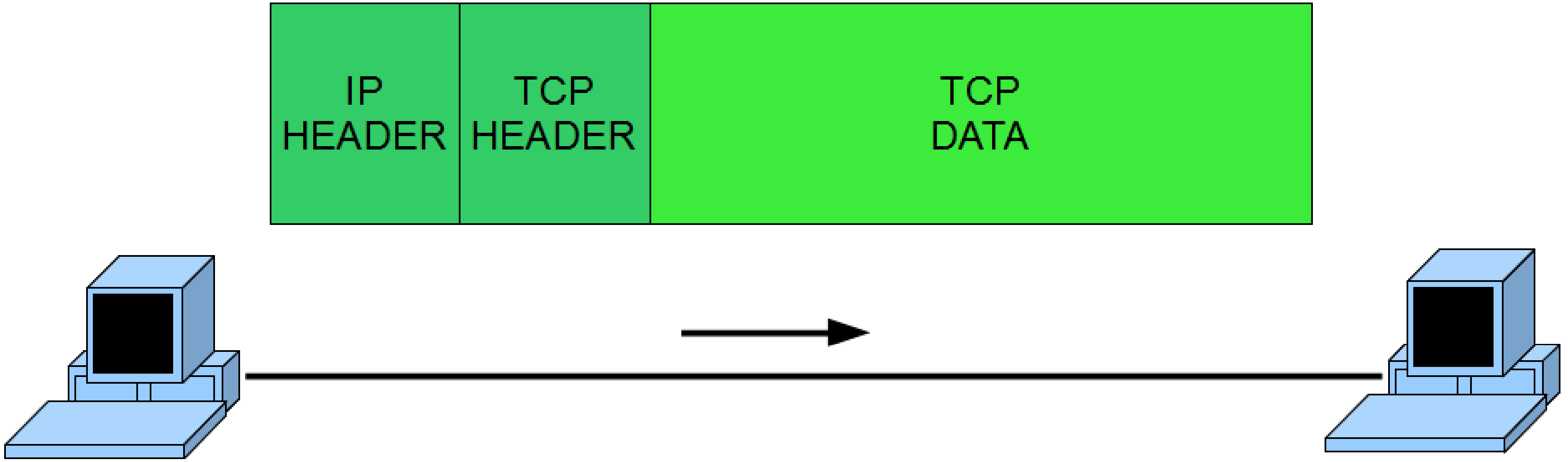
Urgent

- Sometimes data within a segment requires immediate action.
- The client can mark such data in a segment as urgent.
- For example CTRL-C in a remote shell session over TCP.

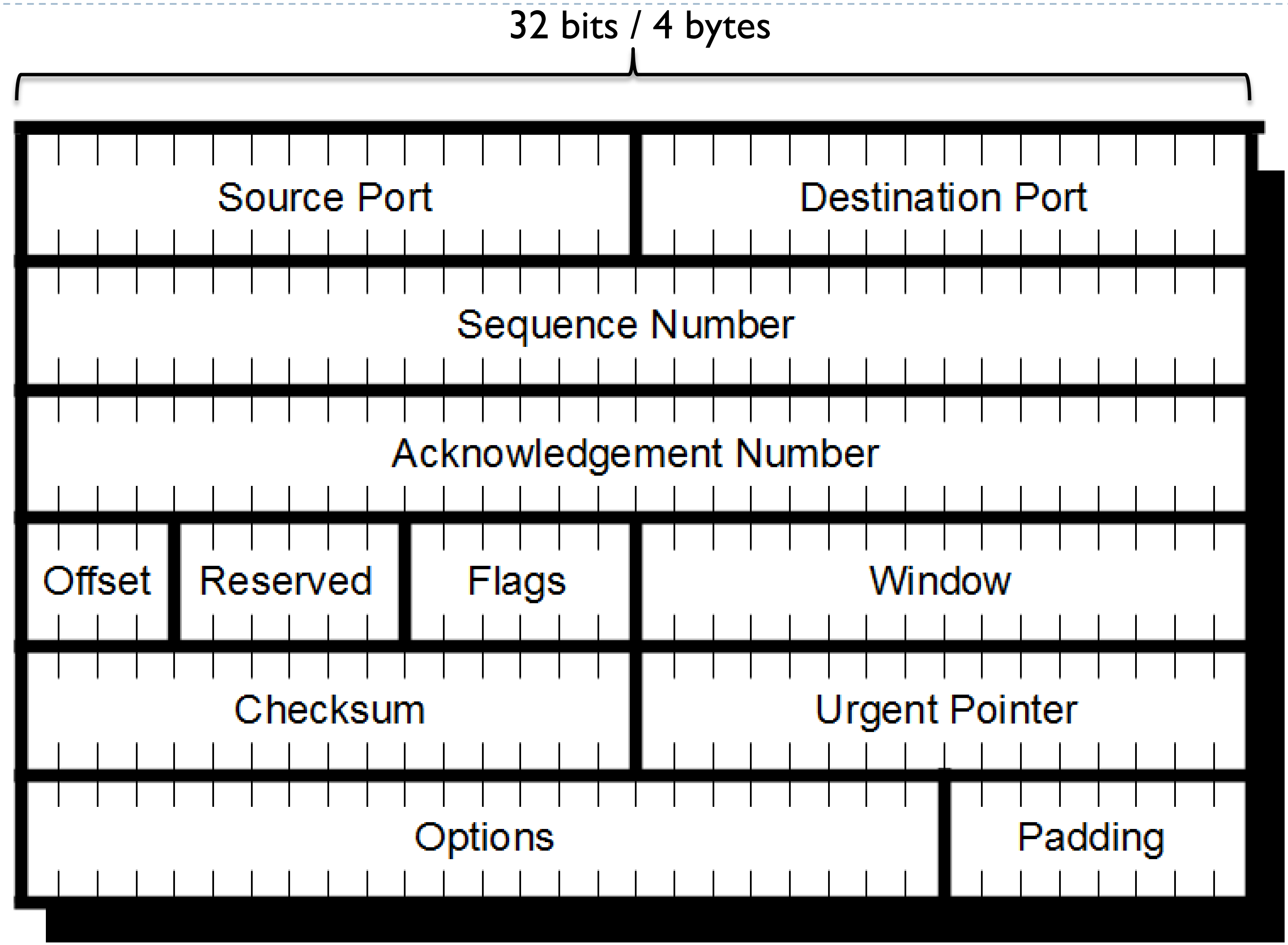
Reset

- If a host crashes or there are severe transmission problems, one host in a connection may lose its knowledge of that connection
- This means that it will not be expecting the data sent from the other host
- If a host receives unexpected TCP data it will respond with a **reset** message to stop the connection and resolve the problem
- Reset also used if a client tries to communicate with a server port that is not open

Combining protocols



TCP header


32 bits / 4 bytes
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Offset
Reserved
Flags
Window
Checksum
Urgent Pointer
Options
Padding

Source and Destination Port fields

- Source Port: the port of the segment sender
- Destination Port: the port of the receiver
- 16 bits each

- Combines with IP address in IP header to make socket address, e.g.
  - IPv4 Header Source Address: 137.73.9.232
  - TCP Port Address: 8080

Sequence Number field

- Sequence Number represents different values depending on the type of message
- 32 bits

- If in a synchronise message setting up the connection, it is the Initial Sequence Number (ISN)
- Otherwise, this is the sequence number of the start of the segment

Acknowledgement Number field

- Acknowledgement Number is used in acknowledgement messages
- 32 bits

- This field contains the sequence number before which the server has all the message data, e.g.

- Received: 0-10, 10-20 Ack number: 21
- Received: 0-10, 20-30 Ack number: 11

Data Offset field

- Data Offset: The length of the TCP header in 32-bit words
- 4 bits: 0 - 15

- Indicates where the data starts within the TCP message
- Minimum value: 5

Reserved field

- Reserved: Not currently used, reserved for future uses
- 6 bits
- All 0s

Urgent Flag field

- URG Flag: Marks that this message contains urgent data

- Urgent data is normally pushed to ensure it is sent immediately

Acknowledgement Flag field

- ACK Flag: Marks that this is an acknowledgement
- The acknowledgement may be of:
  - A segment being received
  - A synchronisation message, in establishing connection
  - An acknowledgement message, in establishing or terminating a connection
  - A finalisation message, in terminating a connection

Push Flag field

- PSH Flag: Marks that this data was pushed

- A client can request that all the data ready to send should be sent
- This is called a PUSH request
- The PSH flag indicates to the server that this has occurred

Reset Flag field

- RST Flag: Marks that this is a reset message

- If a host receives unexpected TCP data it will respond with a reset message to stop the connection and resolve the problem
- Reset is also used if a client tries to communicate with a server port that is not open

Synchronise Flag field

- SYN Flag: Marks that this is a synchronise message or acknowledgement of a synchronise

- The first two messages in a TCP handshake have this flag set
  - From client to server: I wish to connect
  - From server to client: I accept the connection

Finalise Flag field

- FIN Flag: Marks that this is a finalise message or acknowledgement of a finalise
- The first two messages in a TCP finalisation have this flag set
  - From client to server: I wish to end the connection
  - From server to client: I accept the end of the connection

Window field

- Window: The current acceptable window size
- 16 bits: 0 - 65,535

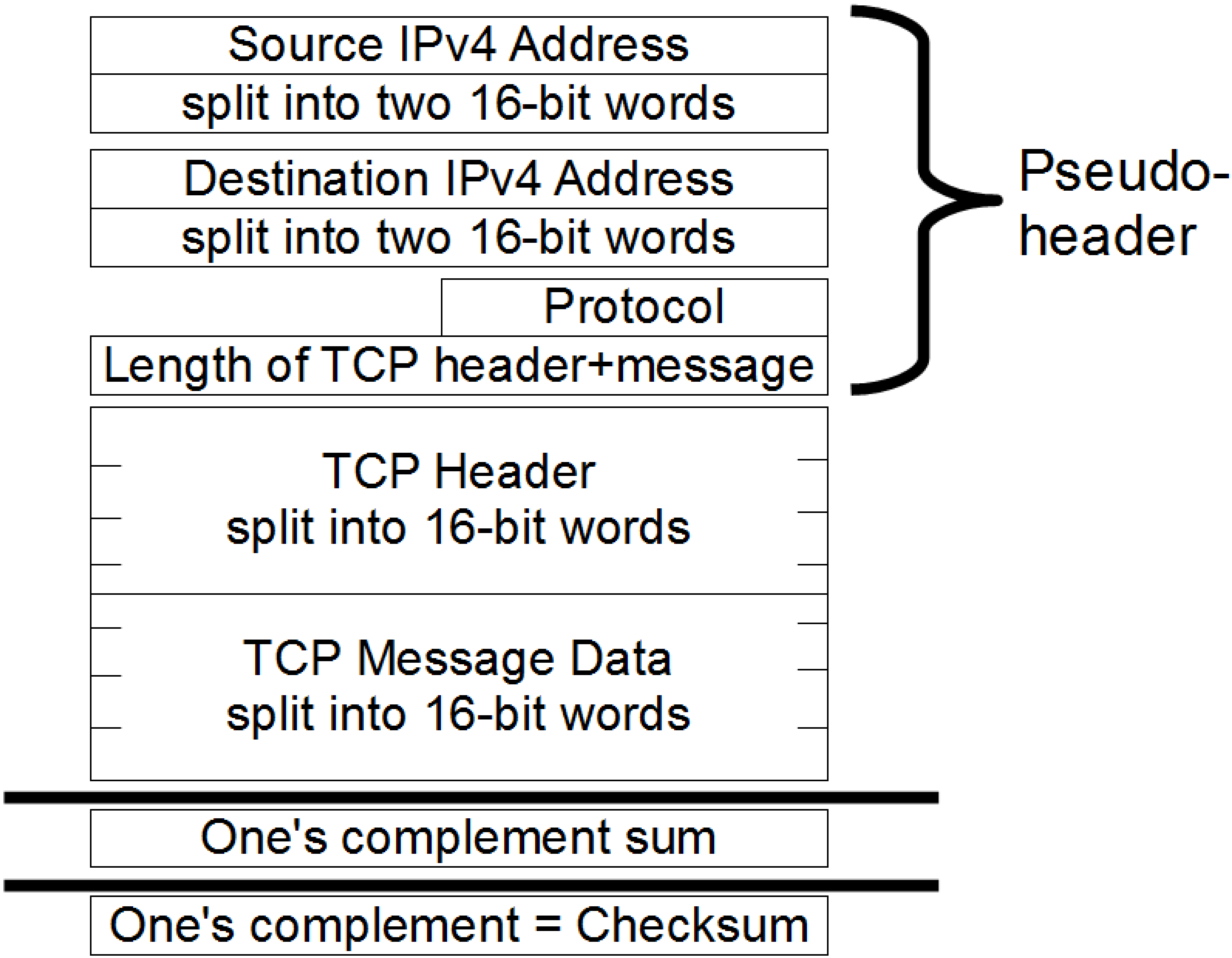
- The server uses this field to adjust the window size during communication if it can accept more or less data
- It is sent in its acknowledgement messages to the client

Checksum field

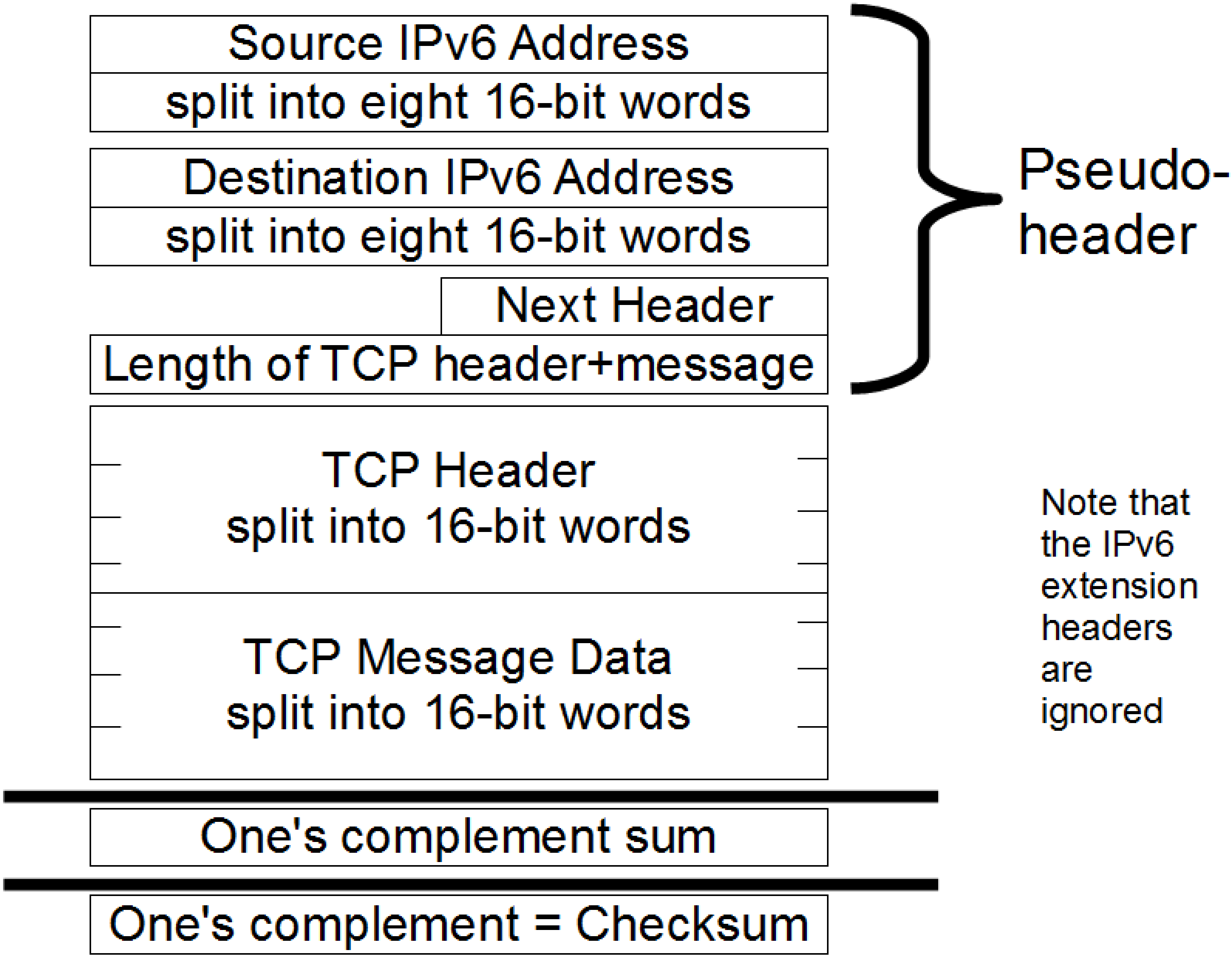
- Checksum: A checksum over the segment, used to check for corruption
- 16 bits

- As with the IPv4 checksum, this is a one's complement of a one's complement sum of 16-bit words
- However, the TCP checksum includes more:
  - The TCP header (taking checksum field as 0)
  - The message data
  - The IP addresses: source and destination
  - The Protocol / Next Header field (from IP header)
  - The length of the TCP message (header + data)

Checksum with IPv4



Checksum with IPv6



Note that the IPv6 extension headers are ignored

Urgent Pointer field

- Urgent Pointer: Position of where the urgent data ends inside the segment
- 16 bits

- Given as a sequence number
- Used when URG Flag is set to Yes (1)
- The message data may mix urgent data with non-urgent data
- The pointer indicates the first byte after the urgent data

Options field

- Options: Various TCP options
- Varies in length

- An important TCP option is the Maximum Segment Size (16 bits)
- This is used for the server to specify the largest segment size it is willing to accept
- It may only be used in synchronisation, i.e. SYN flag is set to Yes (1)
- The server can declare the MSS in the message acknowledging synchronisation

TCP and IP

- TCP is used over IP
- This means that the data using TCP becomes the data content of IP datagrams
- IP deals with:
  - Addressing hosts
  - Fragmentation to ensure the networks can transmit the data
- TCP can somewhat ignore what IP does and just pass it data to send to a given address

RFCs

TCP	793
TCP and IP	879
Silly Window Syndrome	813
Maximum Segment Size	879
Strategy for Silly Window Syndrome	896
Other issues relevant to TCP	2001

- http://www.tcpipguide.com

Outline

- TCP connections
- Segmentation and acknowledgement
- Flow control
- TCP header