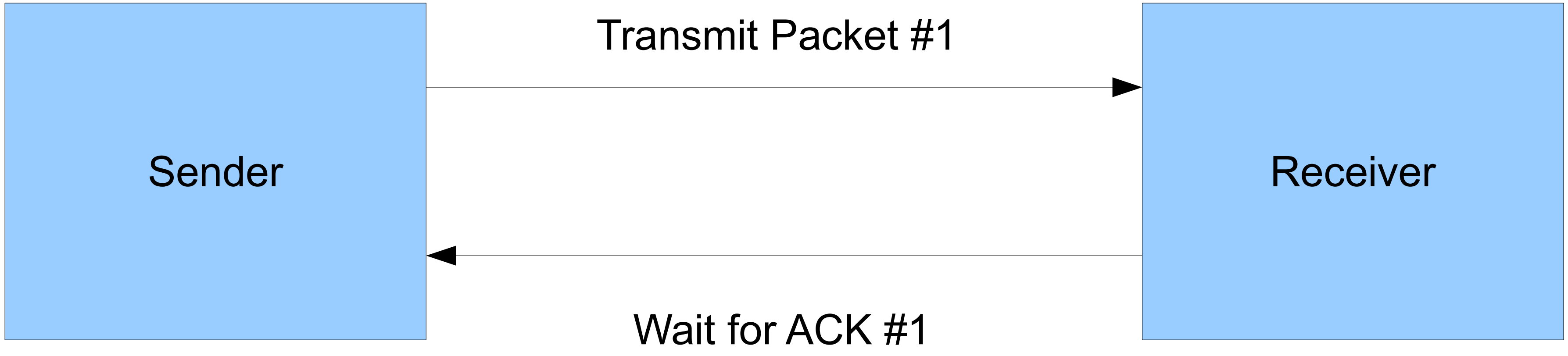
TCP Protocol Details, Part 1

Vermont Technical College
Peter C. Chapin

Introduction: What Not To Do

- IP Network is unreliable
  - Data might never get delivered
  - Data might get delivered multiple times
  - Data might get delivered in the wrong order
- Most applications want reliability
  - **Acknowledge every packet**, retransmit when data is lost.
  - **Use sequence numbers** to recover order and detect duplicates.

Stop and Wait



Protocol transmits, stops and waits…

If no ACK received after a “suitable” time, resend

Receiver uses sequence numbers to ensure duplicates are detected

What’s the Problem?

- Throughput is terrible!
  - Assume: Round Trip Time (RTT) of 50 ms…
  - Assume: Packet contains 1500 octets of data…
  - Then…
    - 1500 octets every 50 ms => 30,000 octets/s
    - *Independent of network bandwidth!!*
  - Stop and Wait is okay for…
    - Small amounts of data with lax latency requirements
    - When the sender and receiver are near each other (small RTT).

Extreme Example

- Long fat pipes…
  - High latency, high bandwidth
  - 10,000 miles at 80% speed of light, 10 Gbps.
    - ~67 ms transit time => ~670 million bits on the the wire.
    - ~56,000 Ethernet II frames!
  - Stop and Wait puts only one frame on the link at a time. *Just ONE!*

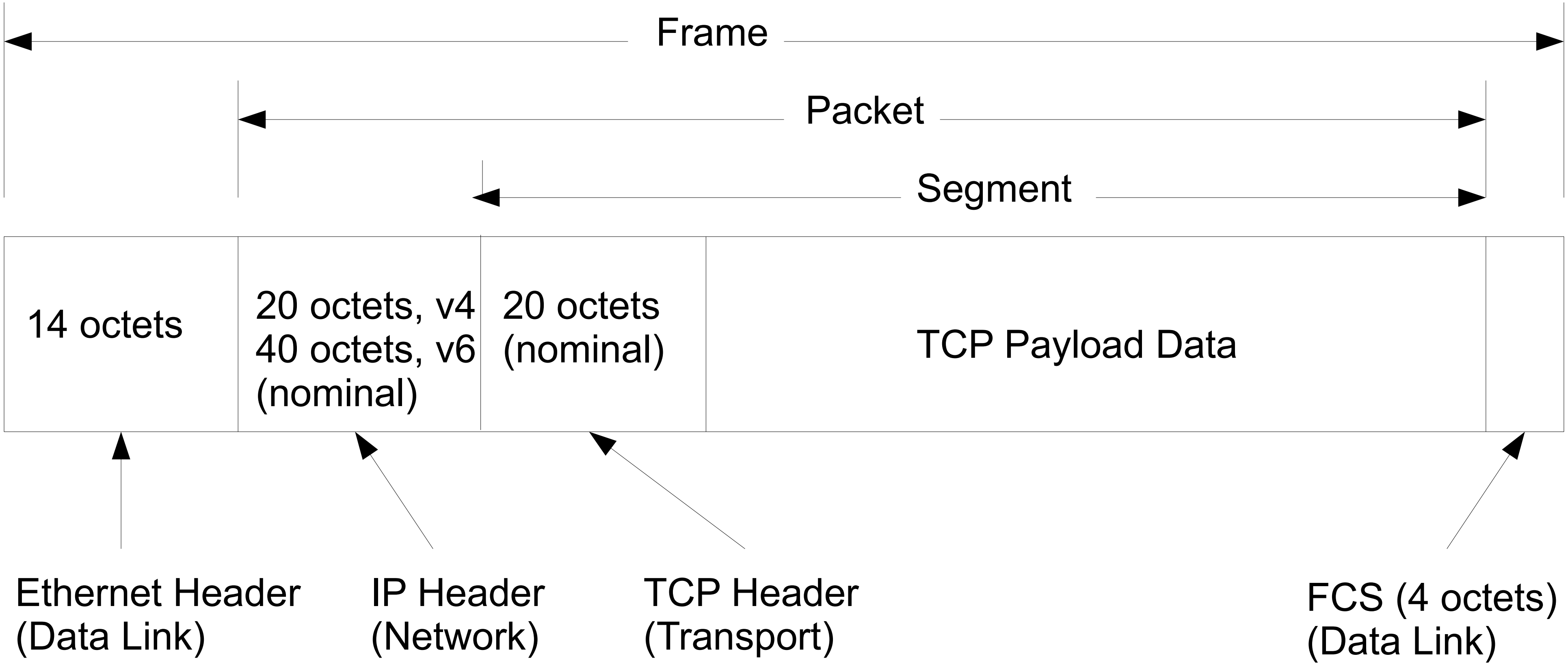
TCP Does Not Do This!

- TCP is highly sophisticated
  - Can transmit speculatively before seeing any ACKs
  - Can adapt transmission rate to account for receiver’s abilities.
  - Can adapt transmission rate to account for network congestion
  - Can dynamically adjust speed to account for changes in network performance or receiver abilities
  - Can do this simultaneously in two directions

But TCP is Not Perfect

- TCP assumes packet loss is due to network congestion.
  - Not always true: wireless networks also loose packets from interference and fading.
  - TCP makes incorrect assumptions in such cases.
  - Doesn’t perform as well across wireless links as it theoretically could.
  - Protocol was designed for a wired world.

Frame Structure



TCP Header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TCP Header Notes

- TCP described in RFC-793 (with updates)
- Note...
  - Source/Destination addresses in IP header.
  - <u>Every</u> octet has a sequence number.
    - Seq # gives number for **first octet** in segment.
    - Ack # gives number for **next octet expected**.
  - Header length (“data offset”) in units of 32 bits.
  - Window size: We will discuss later.
  - Checksum made over “pseudo header” and data.
  - Options typically only occur on initial segments.

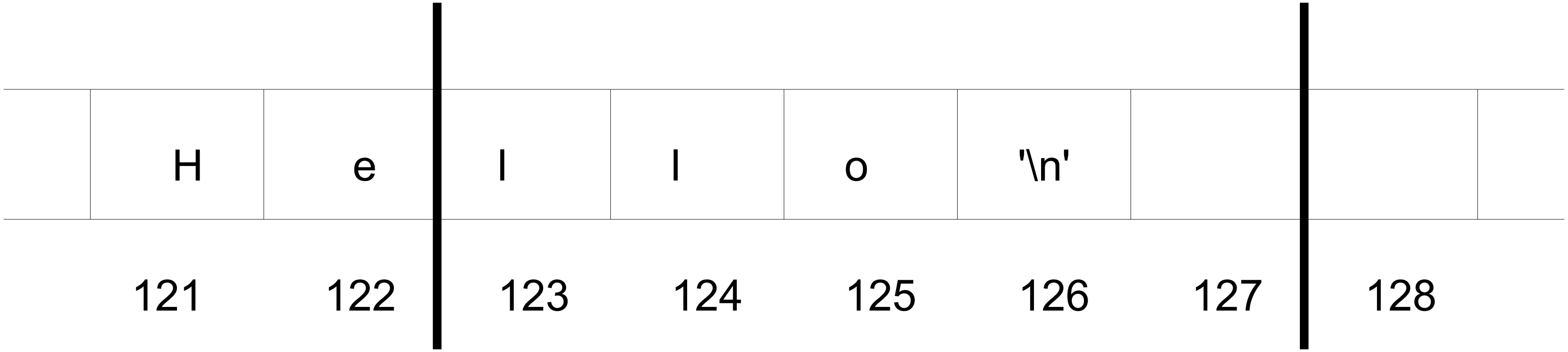
TCP Header Flags

- Several flag bits are defined...
  - `URG`: The value of the Urgent Pointer is valid.
  - `ACK`: The value of the Ack # is valid.
  - `PSH`: The data should be “pushed” to the receiver.
  - `RST`: Reset (end) the TCP connection abruptly.
  - `SYN`: Synchronize (initiate the connection).
  - `FIN`: Close the connection cleanly.

MSS Option

- Most common option is “Maximum Segment Size”
    - Discussed at length in RFC-879
    - Used when connection established. Can only appear in a segment with the SYN flag.
    - Can be different in the different directions.
    - Default 536 bytes (data)
        - Overall packet size 576 bytes (data+TCP+IP).
    - Bigger is better (reduces overhead)
    - Ethernet MSS commonly 1460 bytes
        - Ethernet frame payload 1500 bytes.

Sequence Numbers

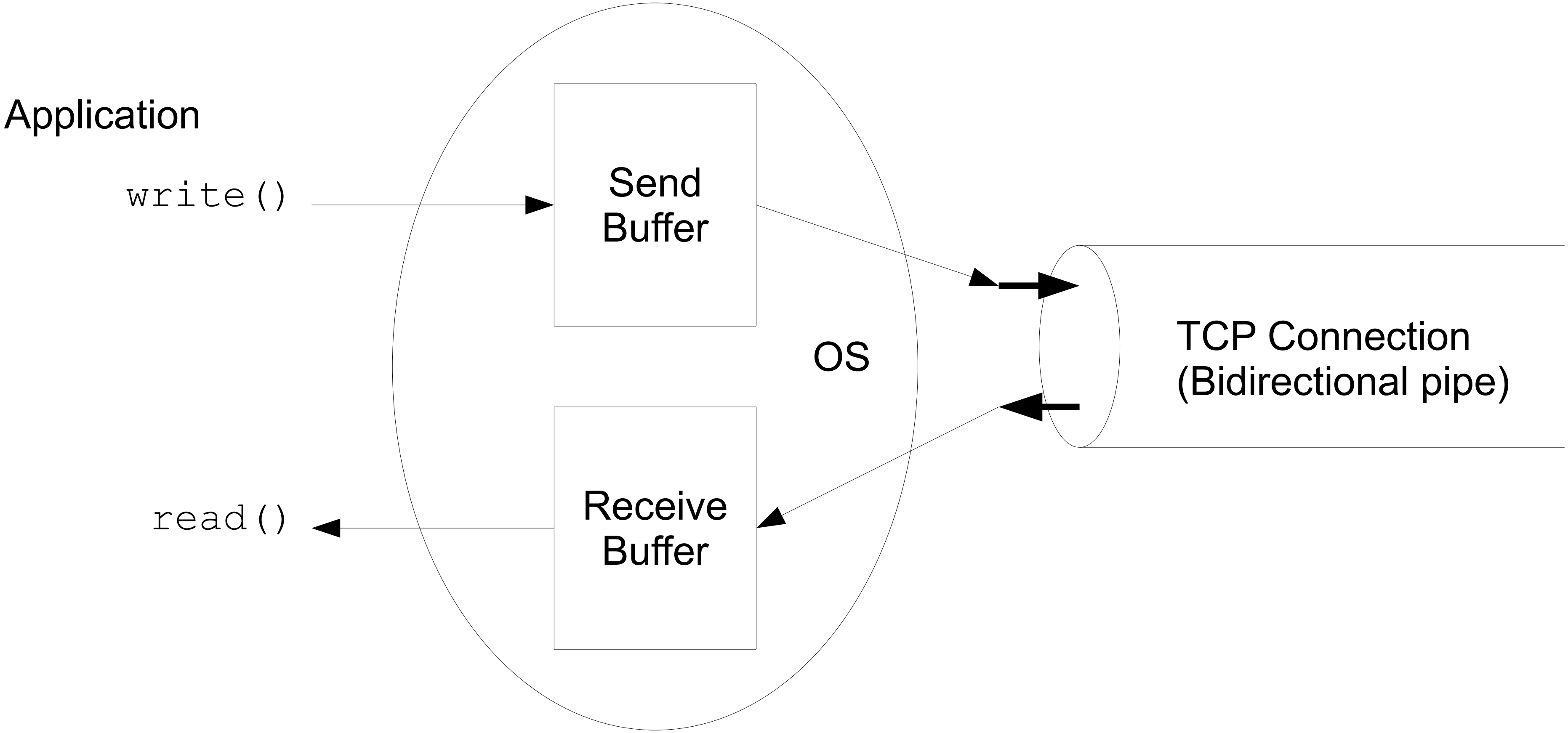


- <u>Every octet</u> has its own number!
- Sequence numbers independent in the two directions.
- *Each segment specifies the sequence number of its first data byte and acknowledges the next sequence number expected from the other side.*
- Segment boundaries are arbitrary.

Stream Oriented

- TCP is a *stream oriented* protocol
  - Data is <u>not</u> broken into records, but instead treated as a continuous stream.
  - TCP breaks data into segments <u>arbitrarily</u>.
  - Applications unaware of segment boundaries.
  - A single call to `write` might...
    - Generate multiple segments
    - Generate only part of a segment
  - A single call to `read` might...
    - Obtain data from multiple segments.
    - Obtain data from only part of a segment

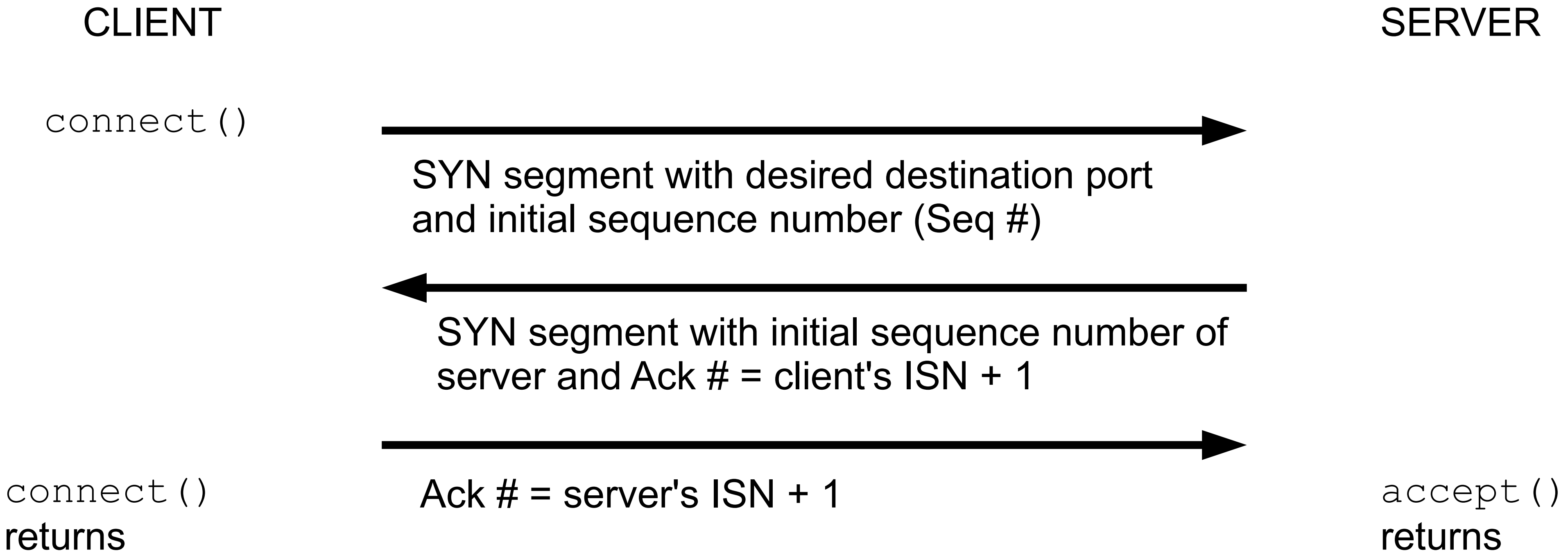
TCP Buffers


Application
write()
read()
Send Buffer
Receive Buffer
OS
TCP Connection (Bidirectional pipe)

TCP Buffers Notes

- `write` normally returns at once.
  - Even before the data has been sent!
- If data arrives it is buffered.
- If receive buffer non-empty, `read` returns at once.
  - Even if data size less than requested amount.
- If receive buffer empty, `read` blocks.
- When connection closed, buffers drain normally.
- Application can terminate before TCP is done!

Establishing TCP Connection



CLIENT

SERVER

`connect()`

SYN segment with desired destination port and initial sequence number (Seq #)

SYN segment with initial sequence number of server and Ack # = client's ISN + 1

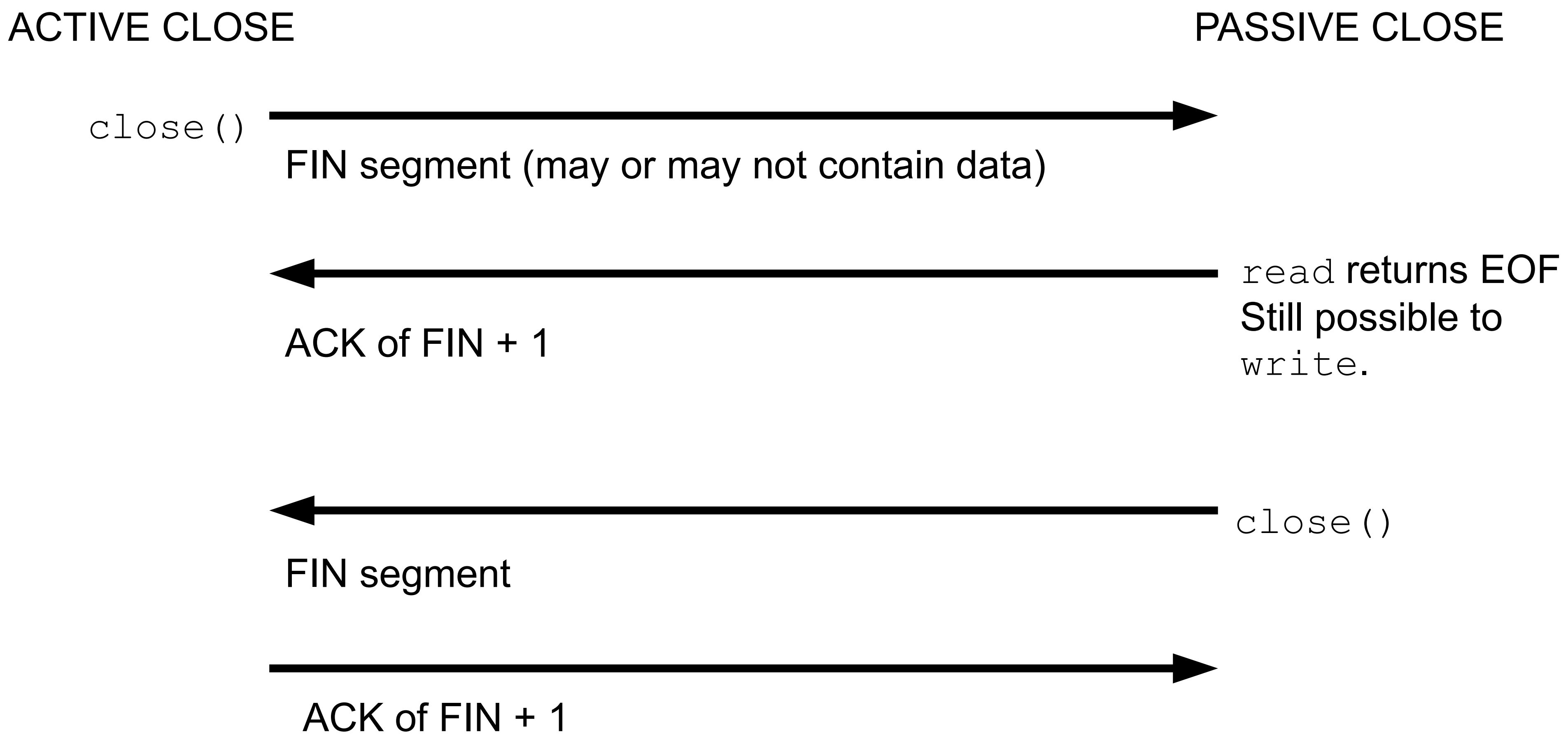
Ack # = server's ISN + 1

`connect()` returns

`accept()` returns

- “Three way handshake” : SYN, SYN/ACK, ACK
- Initial sequence numbers (ISNs) independent and arbitrary
- Connection ESTABLISHED once complete.

Close TCP Connection



ACTIVE CLOSE

PASSIVE CLOSE

`close()`

FIN segment (may or may not contain data)

`read` returns EOF
Still possible to `write`.

ACK of FIN + 1

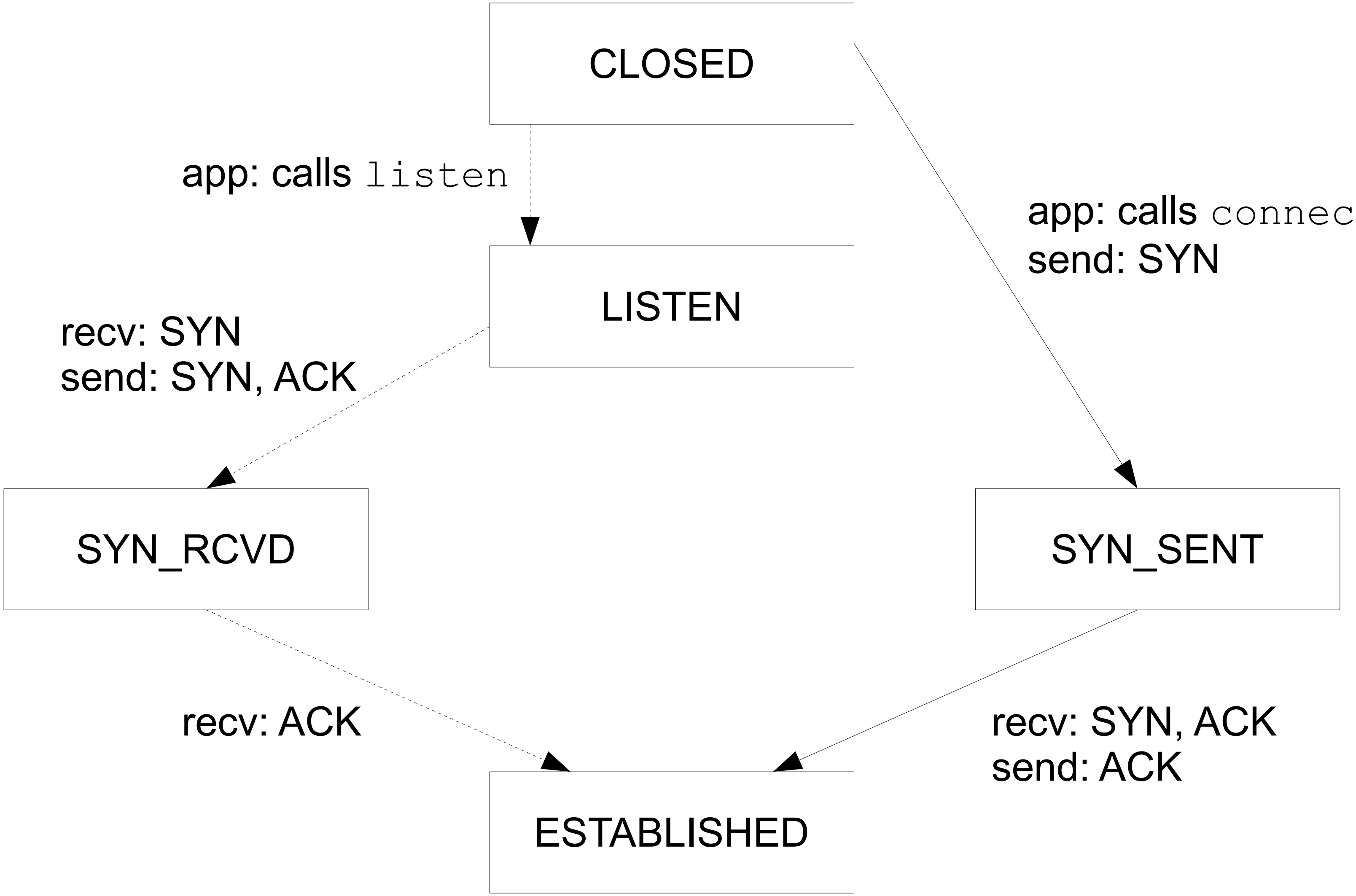
`close()`

FIN segment

ACK of FIN + 1

- Middle two segments might get combined
  - For example: If application closes very quickly after `read` returns EOF.

TCP State Diagram, Part 1



CLOSED

app: calls `listen`

LISTEN

app: calls `connect`
send: SYN

recv: SYN
send: SYN, ACK

SYN_RCVD

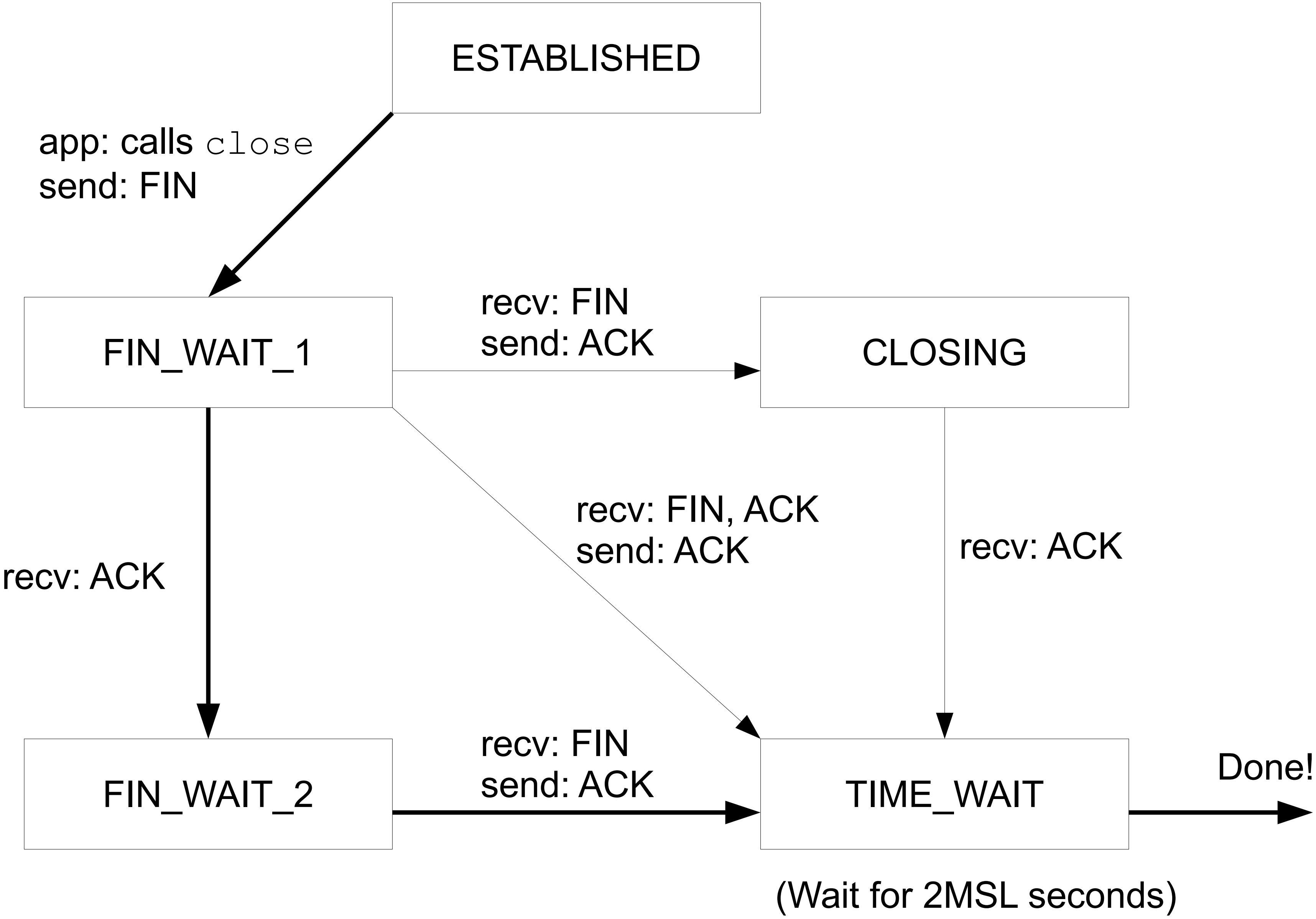
SYN_SENT

recv: ACK

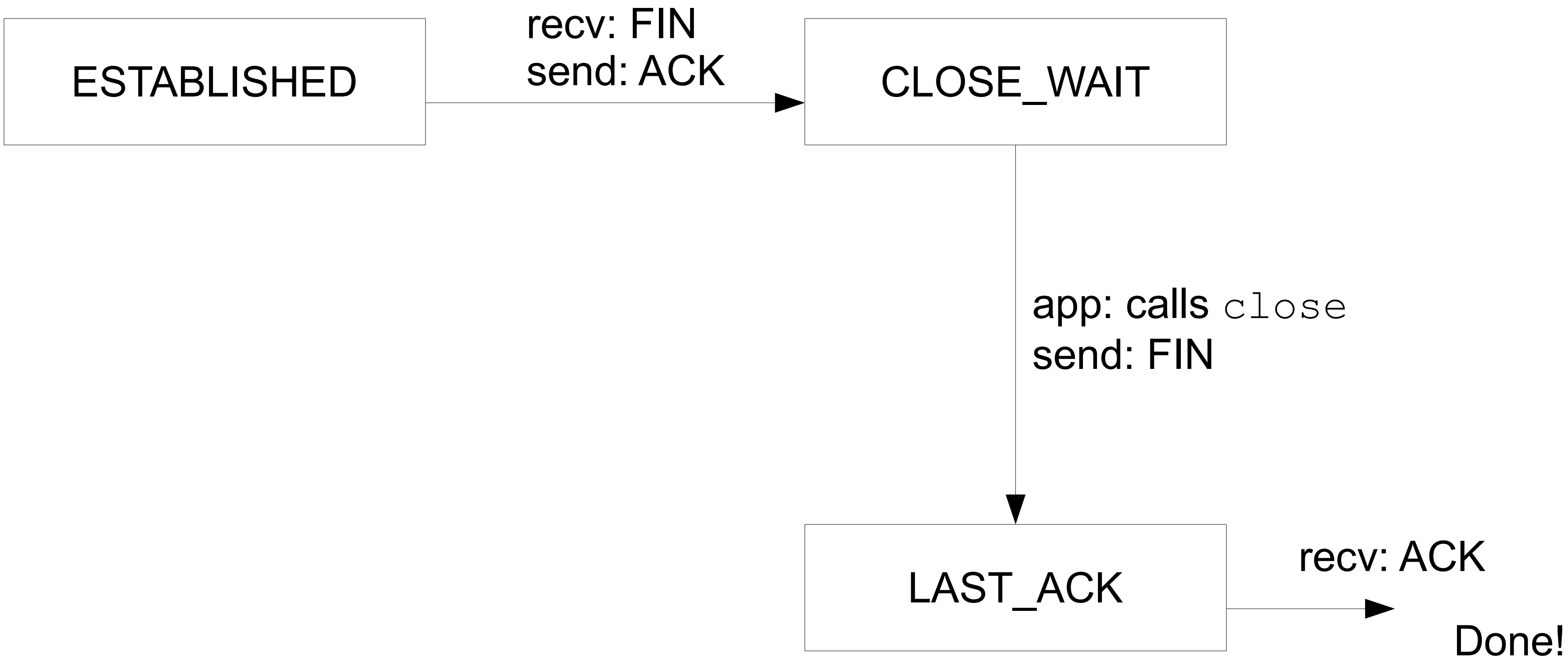
recv: SYN, ACK
send: ACK

ESTABLISHED

TCP State Diagram, Part 2



TCP State Diagram, Part 3



ESTABLISHED

recv: FIN
send: ACK

CLOSE_WAIT

app: calls `close`
send: FIN

LAST_ACK

recv: ACK

Done!

Tools

- On Unix use `netstat` to view connection state
    - `netstat -a` shows “all” connections (including listening sockets).
    - `netstat -A inet6` shows connections in the “inet6” address family (TCP running on Ipv6).
    - See man page for more details.
- On Windows TCPView is a GUI netstat tool
    - http://www.sysinternals.com/

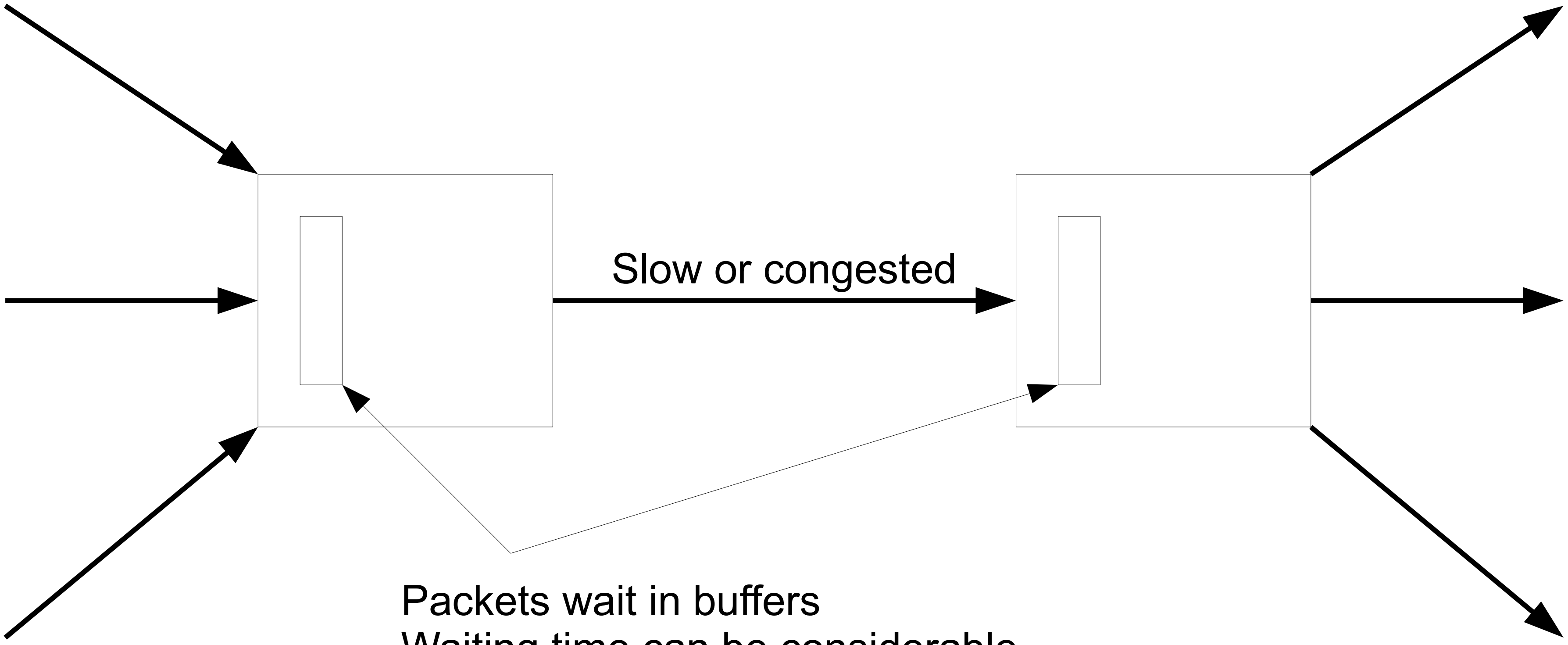
Stop-And-Wait

- Simple protocol for transferring data.
  - Send one segment.
  - Wait for acknowledgement.
- Easy to implement, but has disadvantages:
  - Only one segment on the network.
    - Inefficient use of network bandwidth.
  - Sender must wait for `2*TT` (where `TT` is the transit time across the network).
    - Could be many milliseconds... or even seconds!
    - Lots of waiting; slow data transfer.

Transit Time

- Finite speed of light (2.998x10^8 m/s)
  - Time is required to move bits (on the order of 50 ms to go 10,000 miles).
  - Speed on cables is actually less.
    - “Velocity factor” on typical cables might be 0.80.
    - Due to dielectric material used as insulation and cable geometry.
- Also router delays.
  - This is usually the biggest factor.

Router Delay



Slow or congested

Packets wait in buffers
Waiting time can be considerable

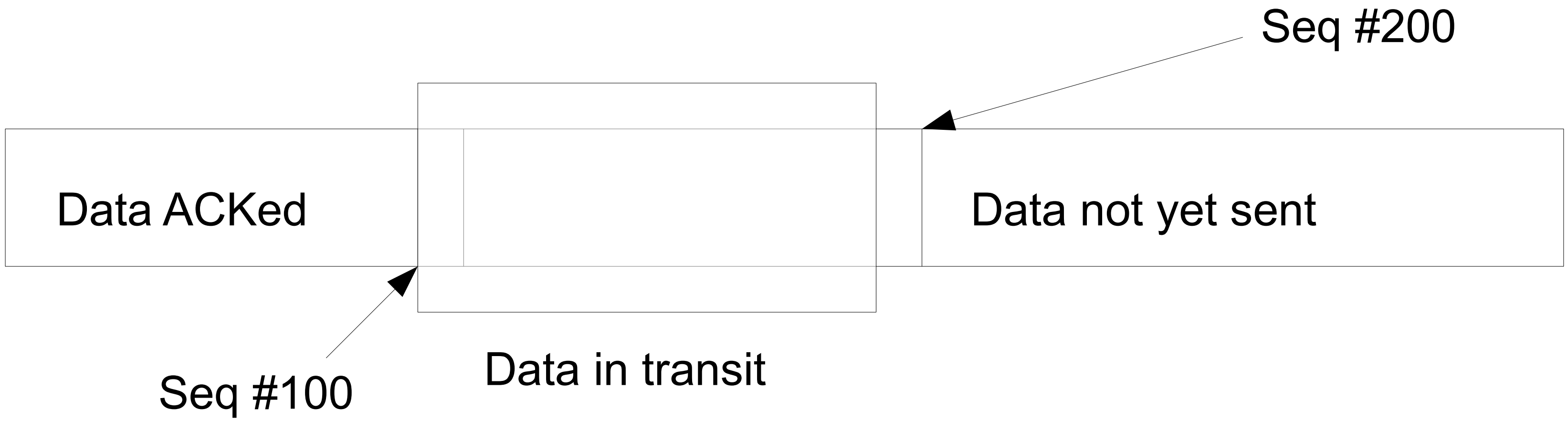
Stop-And-Wait Computation

- Assume...
  - Transit time = 50 ms (one way) or 100 ms round.
  - Each packet contains 1000 bytes.
  - Transfer rate = 1000 bytes/0.1s = **10,000 bytes/s**.
- Stop-And-Wait must wait for the ACK
  - Spends most of its time waiting.
- Transfer rate is independent of bandwidth!
  - Calculation above is valid as long as bandwidth is sufficient.

Latency vs Bandwidth

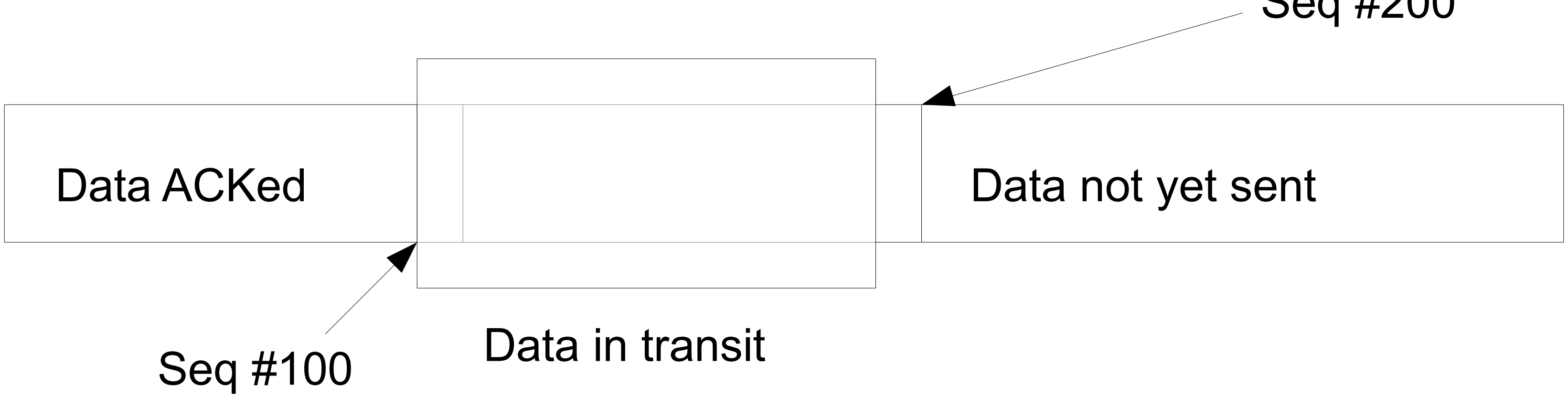
- Be aware that “latency” is different than “bandwidth.”
  - Latency:
    - How long does it take for the first bit transmitted to reach the destination?
    - Delays due to the speed of light and router buffering, etc.
  - Bandwidth:
    - How many bits/s can be transmitted?
- High latency, high bandwidth connections...
  - “Long fat pipes.”

TCP Window



- Data is sent to fill a “window”
  - TCP speculatively sends without ACKs
  - Fills the network with data
    - Much more efficient than stop-and-wait
  - By the time the window is transmitted, the first ACKs arrive (we hope).
  - Each ACK moves the window forward.

TCP Window



Seq #200

Data ACKed

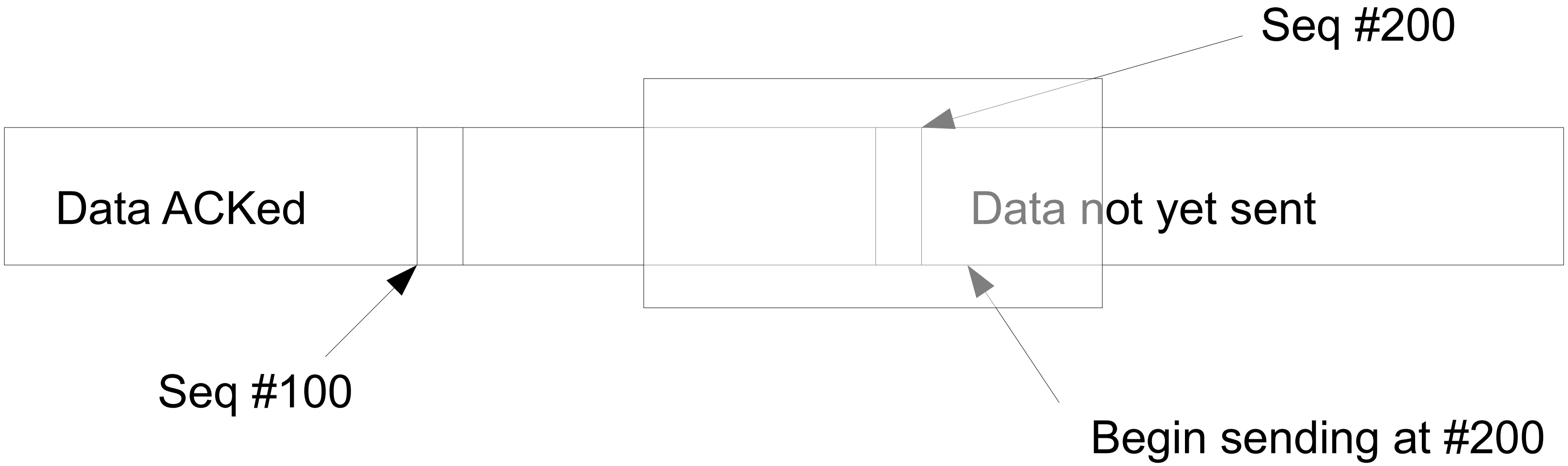
Data not yet sent

Seq #100

Data in transit

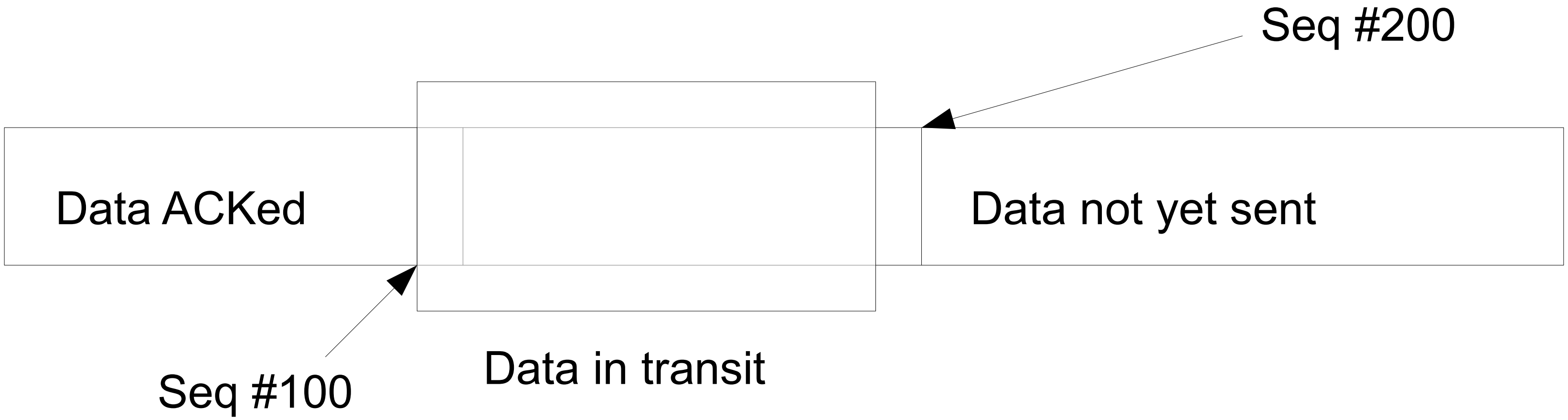
Now: segment arrives: ACK# 150; window 100

TCP Window



- ACK: “I've received <u>everything</u> below the ACK number.”
- Window moves as ACK advances.
  - Exposed data can now be sent until window is filled again.
  - Try to keep window-size bytes of data in flight at all times.

TCP Window



Seq #200

Data ACKed

Data not yet sent

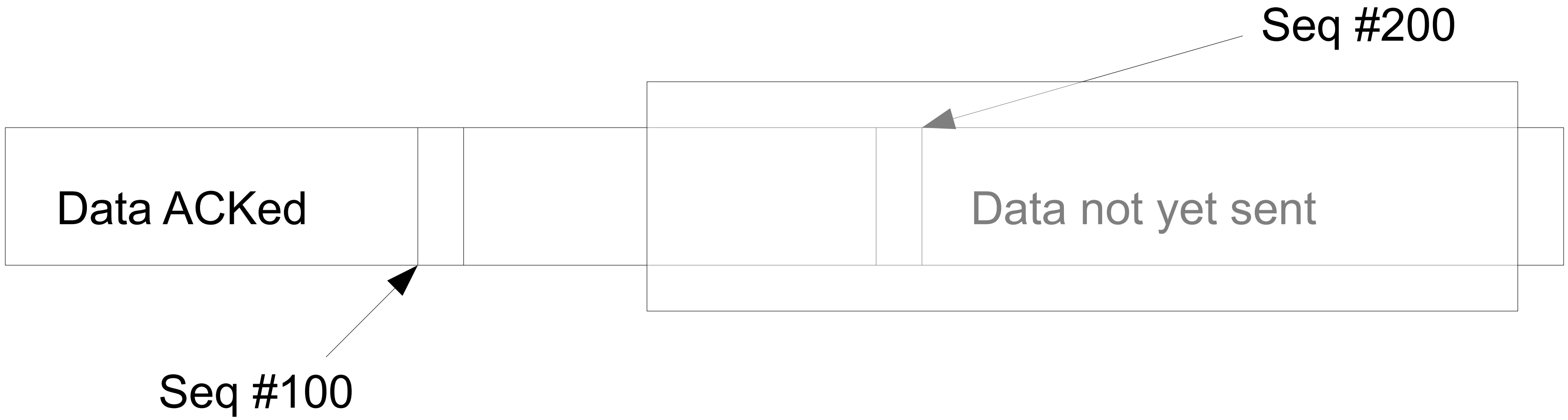
Seq #100

Data in transit

Now: segment arrives: ACK# 150, window = 200

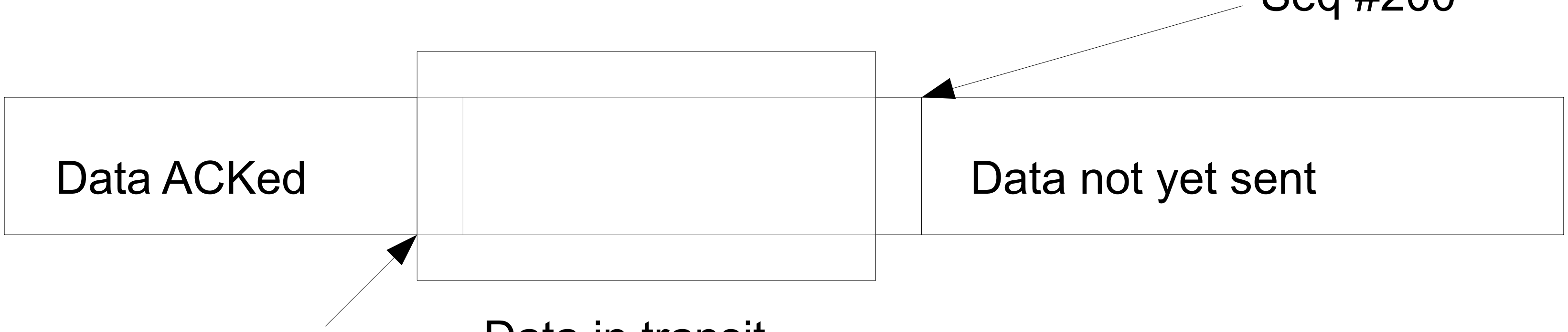
Window size can change!

TCP Window



- Receiver modulates window size to reflect receive buffer size.
  - If the receiver has only a small buffer: small window.
  - As receiver consumes data, buffer empties. Window opens.
- Sender never sends more than receiver can handle!

TCP Window


Seq #200

Data ACKed

Data not yet sent

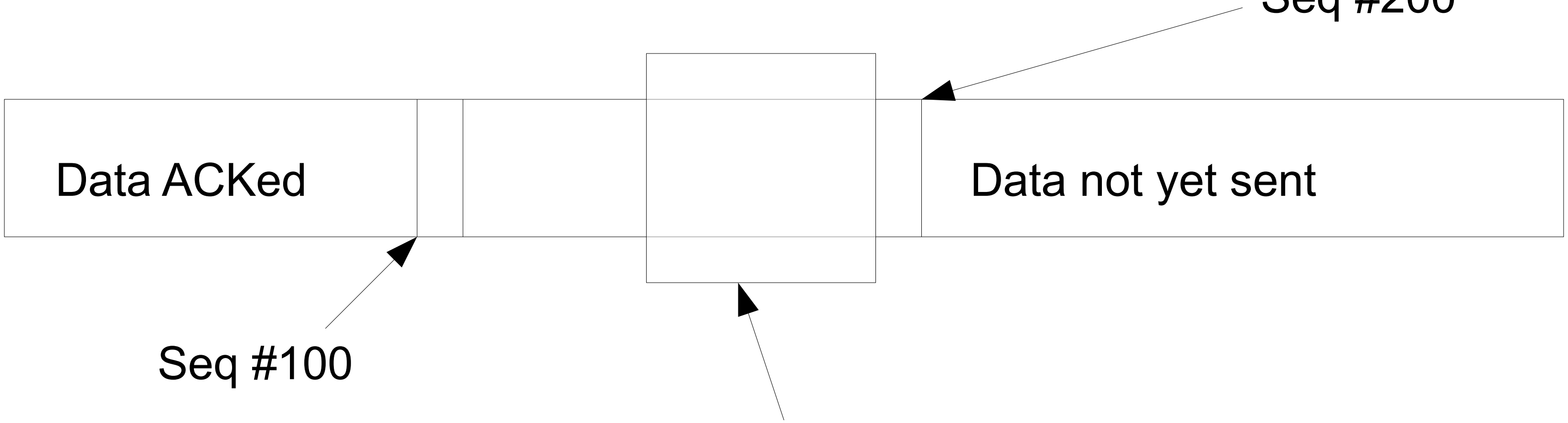
Seq #100

Data in transit

Now: segment arrives: ACK# 150, window = 50

Zero size means RX buffer full

TCP Window



Seq #200

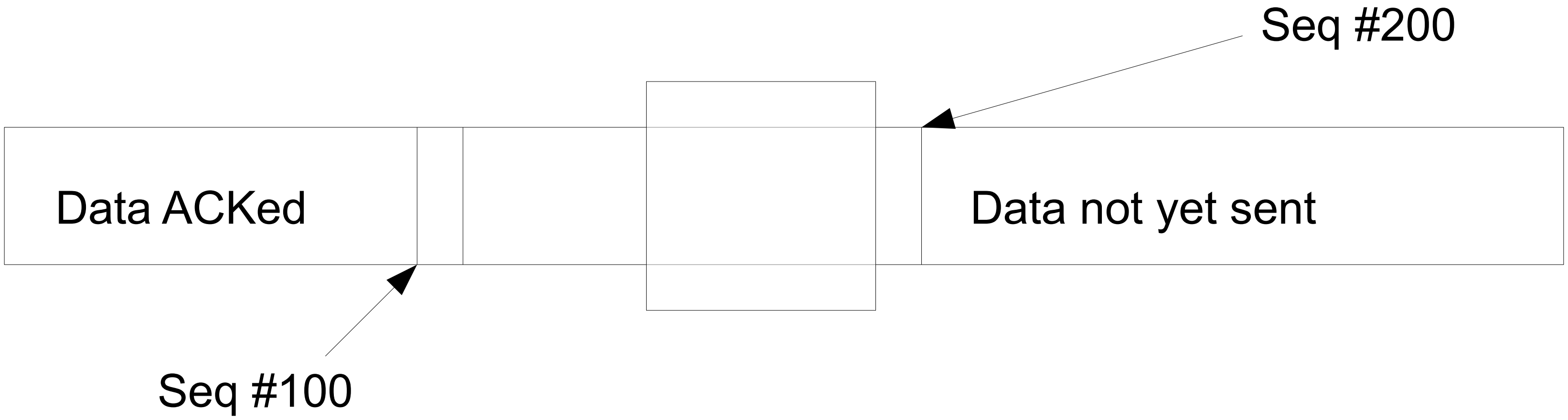
Data ACKed

Data not yet sent

Seq #100

Window doesn't expose new data Can't send.

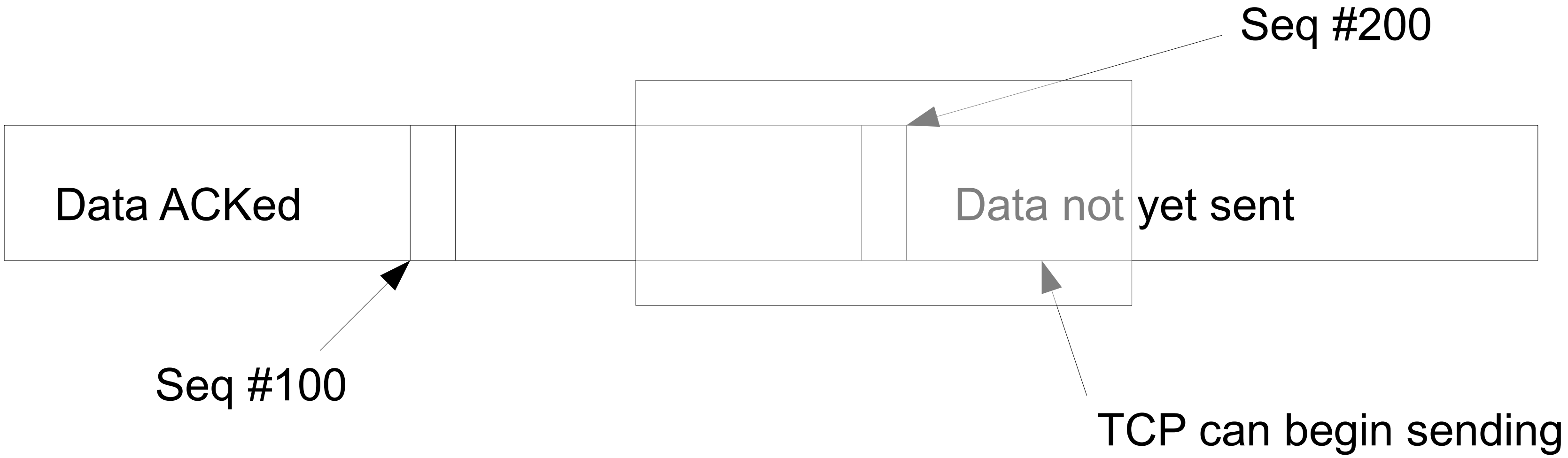
TCP Window



Now: segment arrives: ACK #150, window = 100

Segment ACKs nothing new.
Just used for window size update.

TCP Window



Seq #200

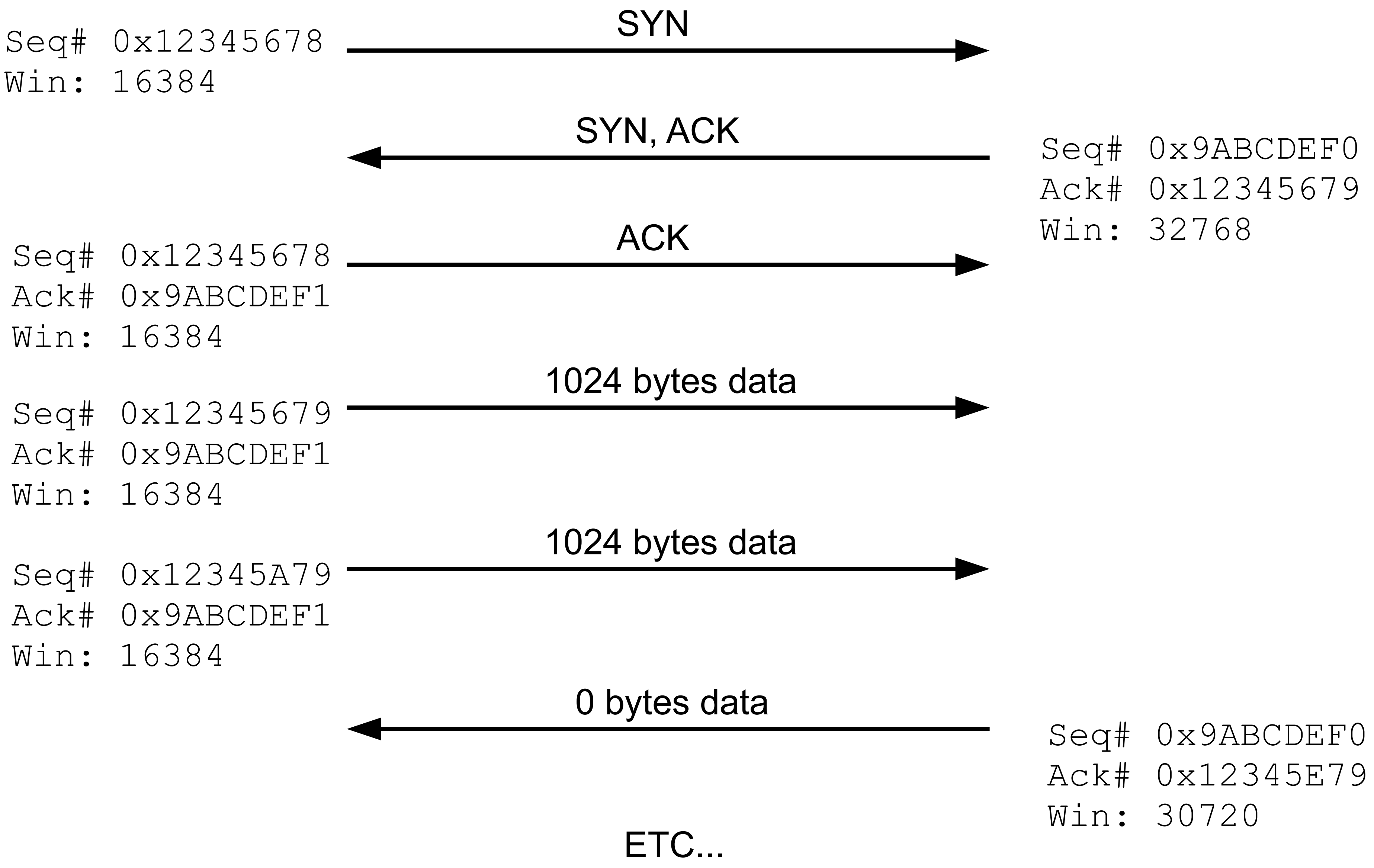
Data ACKed

Data not yet sent

Seq #100

TCP can begin sending

Example



Seq# 0x12345678
Win: 16384

SYN

SYN, ACK

Seq# 0x9ABCDEF0
Ack# 0x12345679
Win: 32768

ACK

Seq# 0x12345678
Ack# 0x9ABCDEF1
Win: 16384

1024 bytes data

Seq# 0x12345679
Ack# 0x9ABCDEF1
Win: 16384

1024 bytes data

Seq# 0x12345A79
Ack# 0x9ABCDEF1
Win: 16384

0 bytes data

Seq# 0x9ABCDEF0
Ack# 0x12345E79
Win: 30720

ETC...