THE UNIVERSITY OF SYDNEY

Lecture 7

Transport Layer

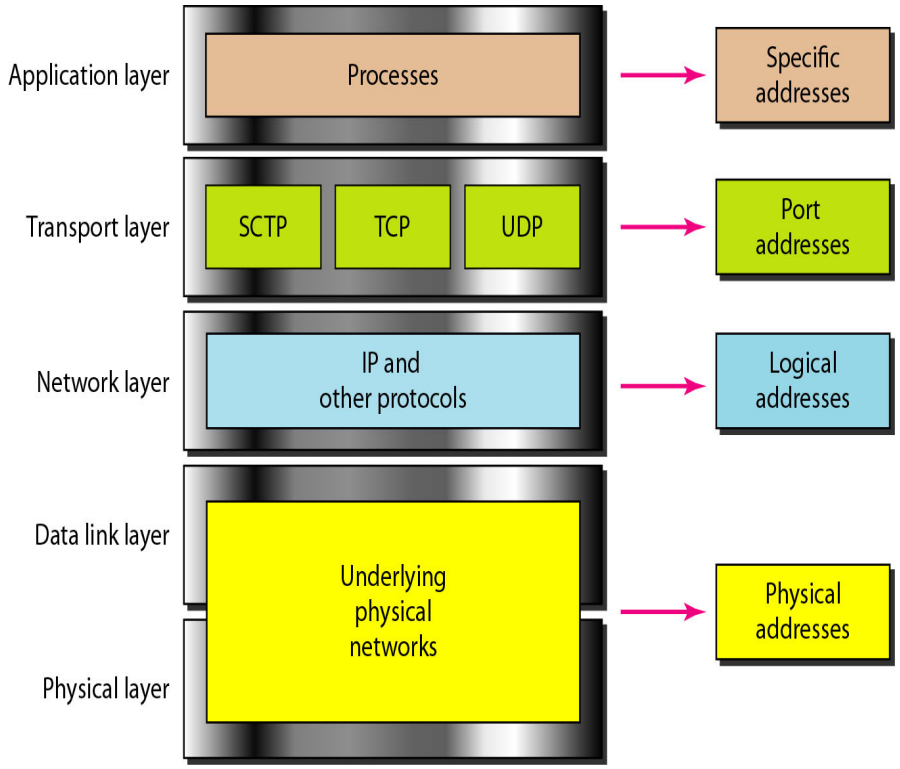
ELEC 3506/9506
Communication Networks

Dr Wibowo Hardjawana

School of Electrical and Information Engineering

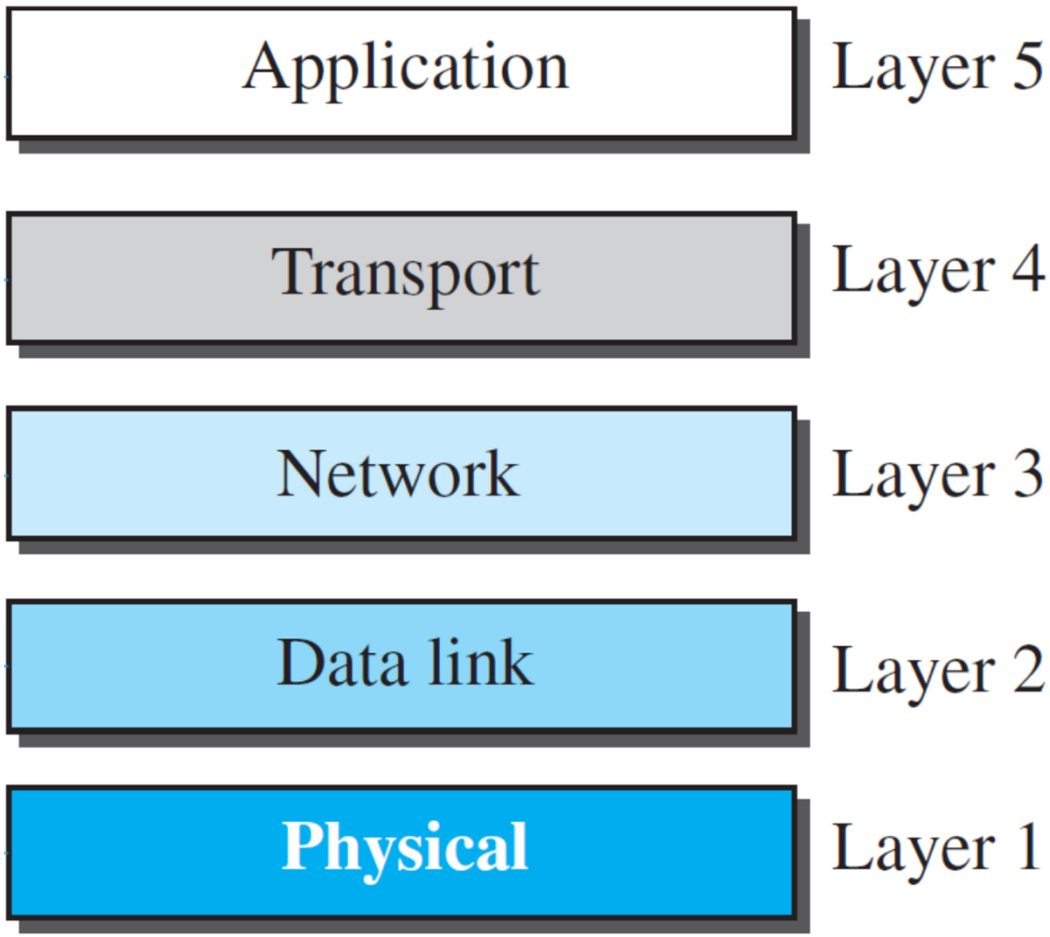
Topics of the day

- General services required from transport layer
  - Process-to-process communication
  - Transport layer addressing
  - Multiplexing and de-multiplexing
  - Error, flow, and congestion control
- Connection-less vs. Connection-oriented services
- Reliable vs. Unreliable
- Two popular transport layer protocols
  - UDP (User Datagram Protocol)
  - TCP (Transmission Control Protocol)



THE UNIVERSITY OF SYDNEY

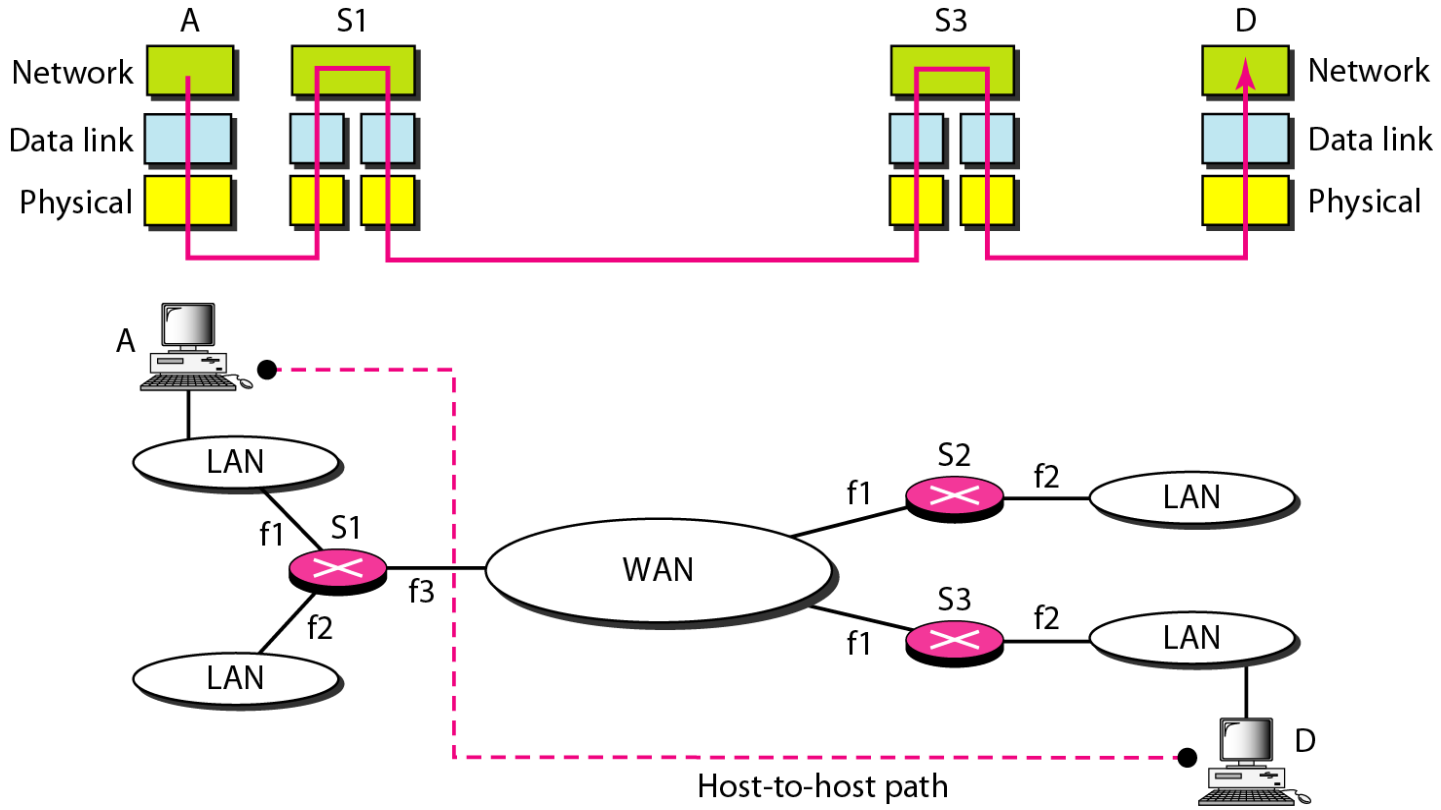
Layers in the TCP/IP Protocol Suite



- **Transport Layer:**
  - Provides services to application layer
  - Receives services from the network layer
  - Heart of the TCP/IP protocol suite

Link Layer vs. Network Layer Delivery

- End-to-End delivery at the **Network Layer**

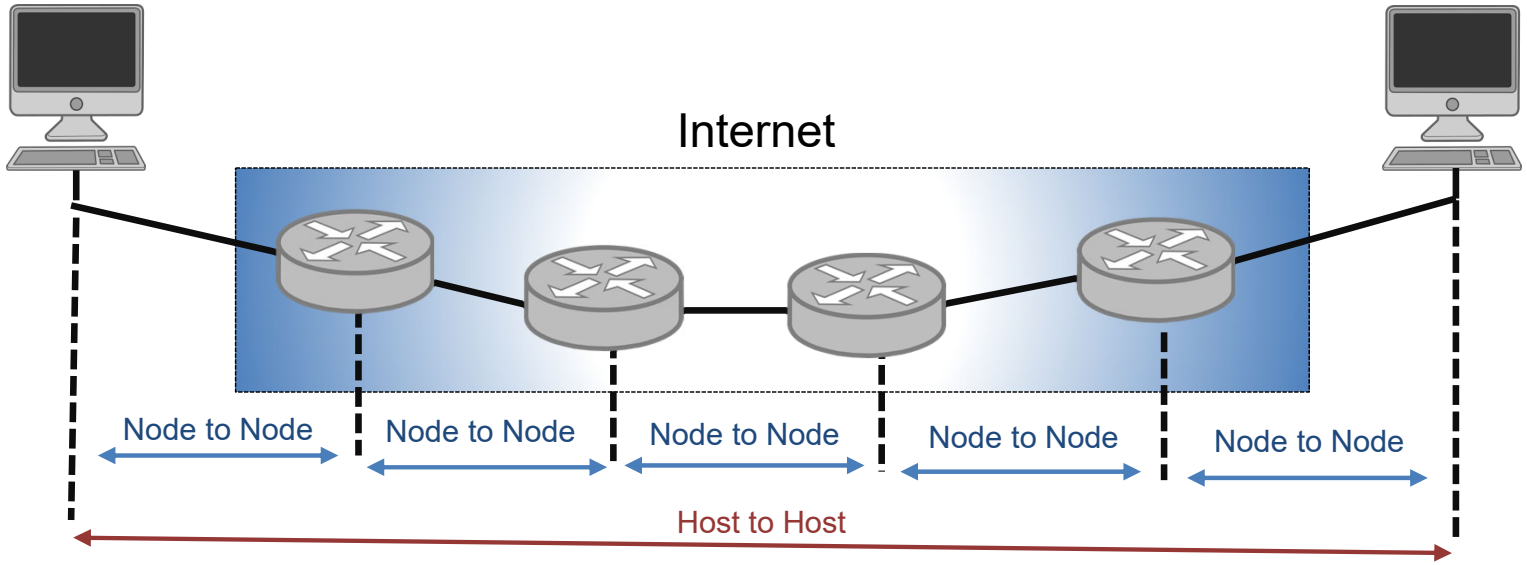


- Hop-to-Hop delivery at the **Data Link Layer**

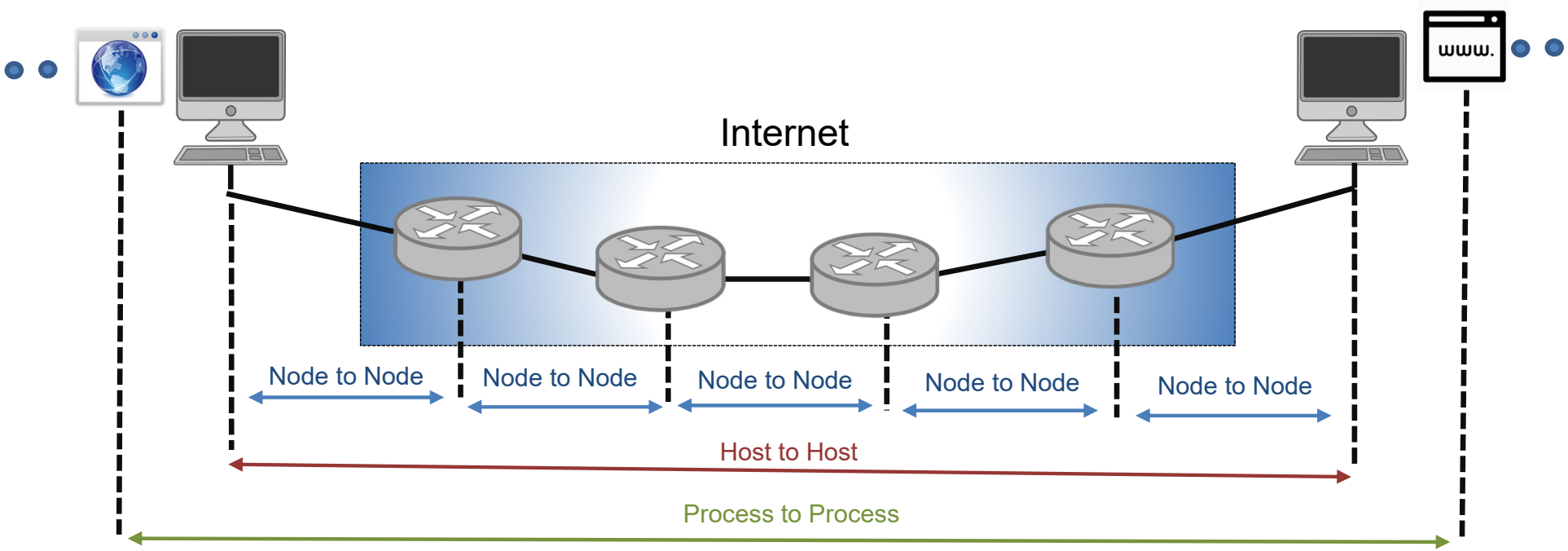
  A Source-to-destination delivery (End-to-End delivery or Host-to-Host delivery) may contain multiple hops

Transport Layer vs Other Layers

- Data Link Layer (Layer 2) – Node to Node Communication (frames transmission) → MAC/NIC address
- Network Layer (Layer 3) – Host-to-Host Communication (datagrams transmission) → IP address

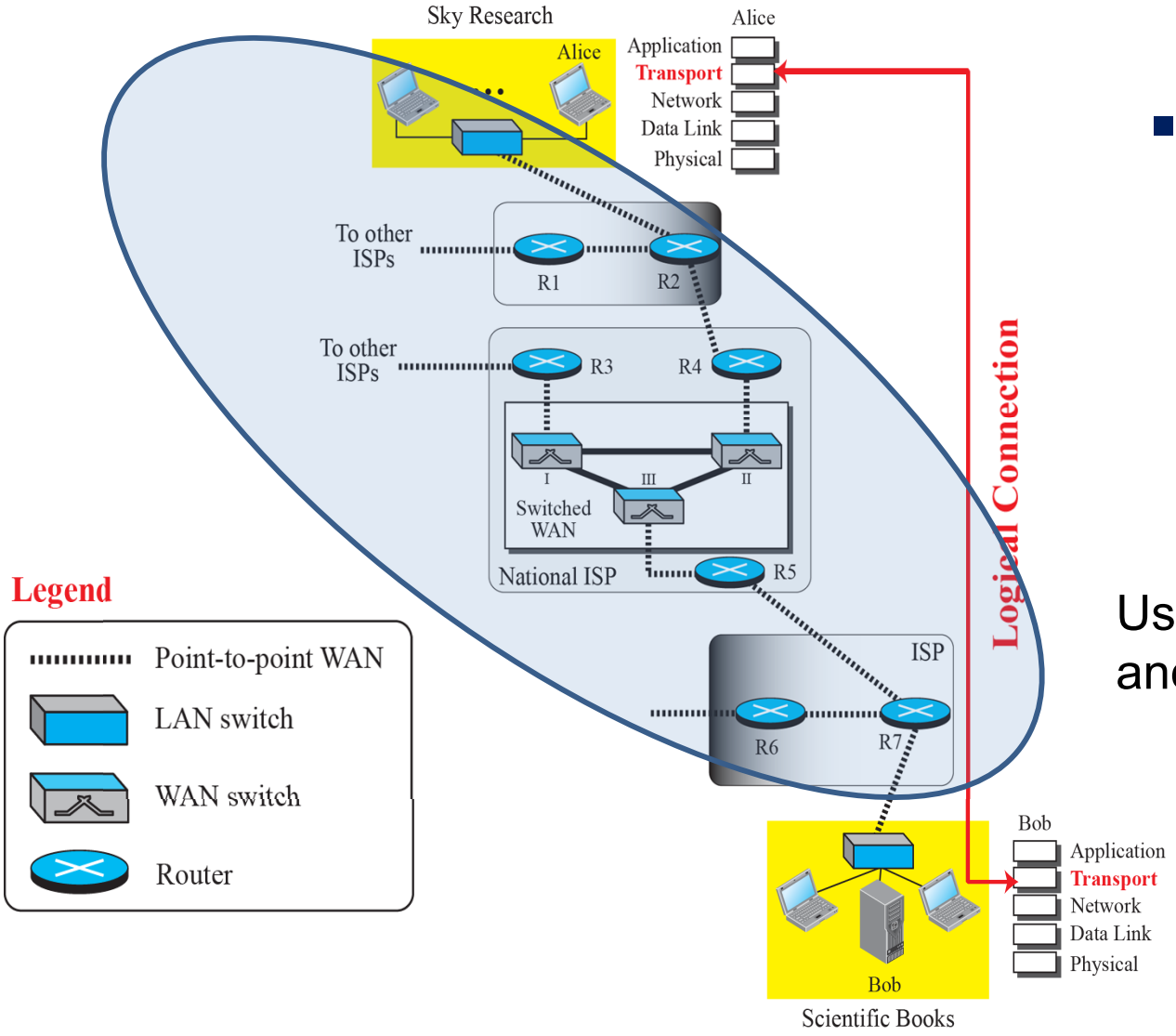


Transport Layer



- **Transport layer**: provides logical communication between applications/**processes** running on different hosts → port number
- A host run multiple processes/applications

Logical Connection

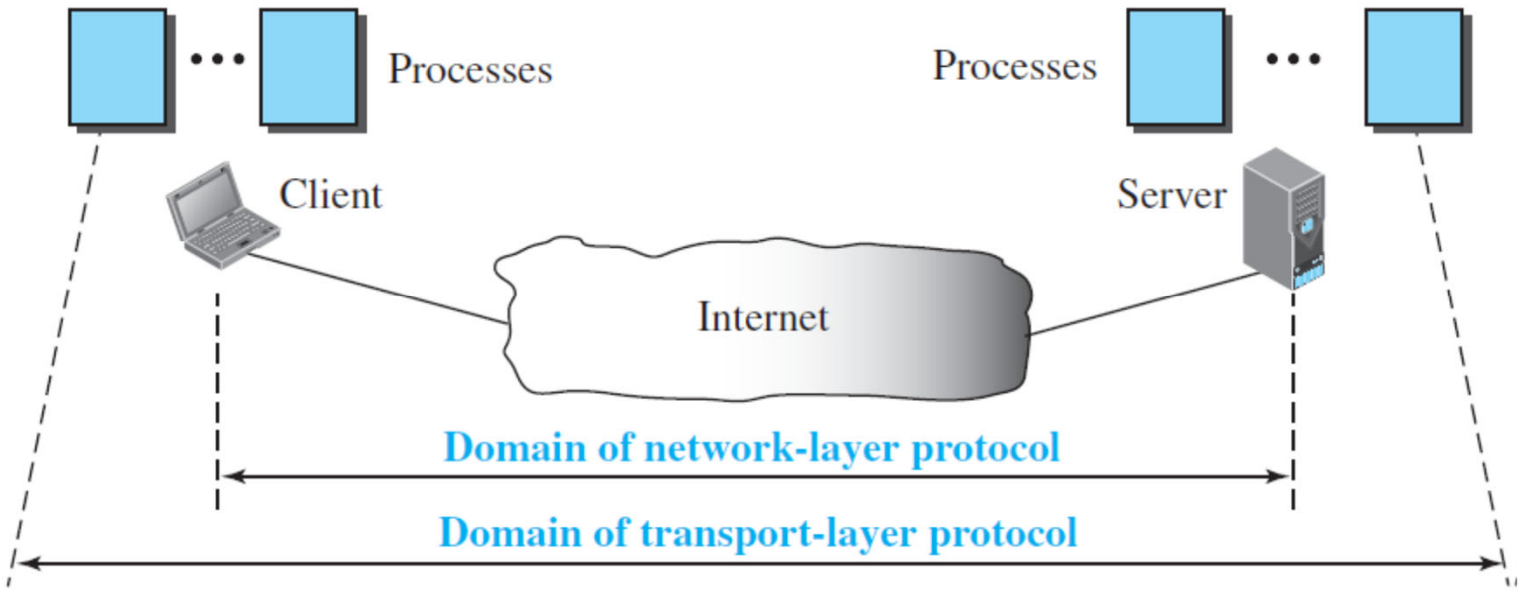


- Imaginary direct connection at the transport layers (Alice and Bob)

Use only PHY, MAC and Network layers

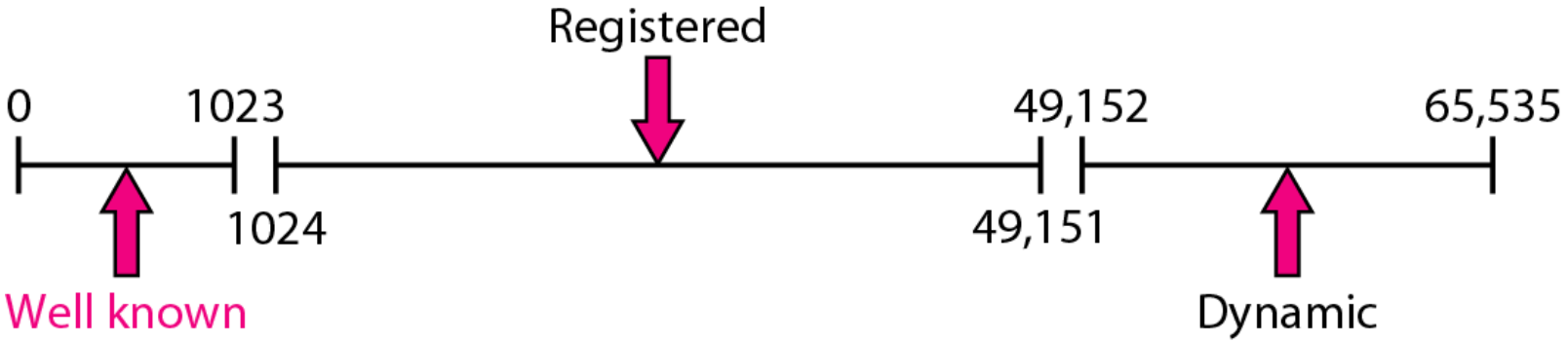
Transport Layer Services

- Provides services to application layer
  - Process-to-process communication
  - Addressing
  - Multiplexing and demultiplexing
  - Error, flow, and congestion control



Transport Layer Addressing

- Client and server may run multiple programs at the same time
- For communication, we must define:
  - Local host and remote host (IP Addresses)
  - Local process and remote process (Port Number)
- **Port number:** 16-bit integer between 0 and 65,535.
- Client port number: ephemeral (short-lived)
- Server port number: well-known
- Source and destination have port numbers

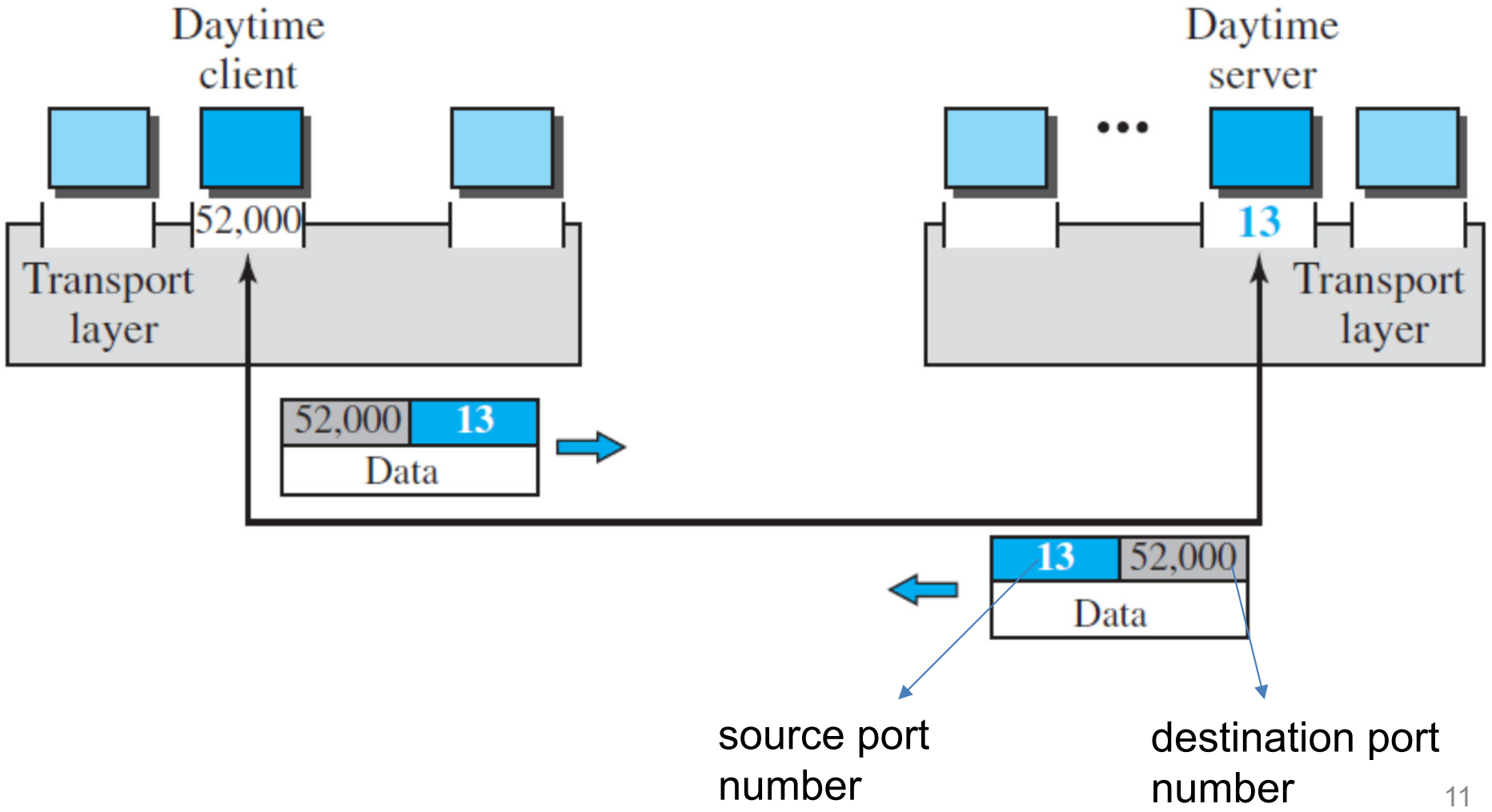


THE UNIVERSITY OF SYDNEY

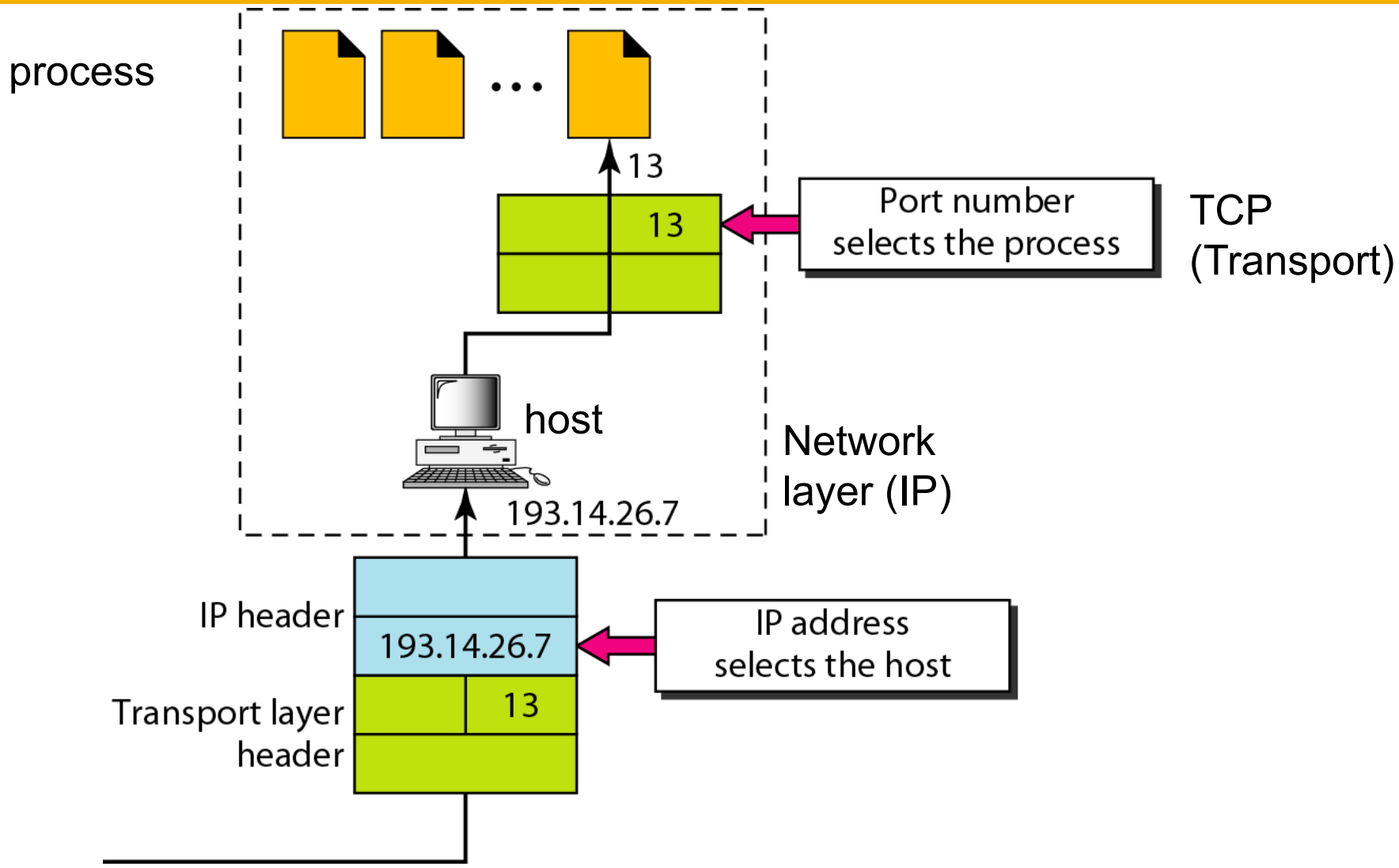
Well Known Port Numbers

Port Number	Description
20	FTP data transfer
21	FTP Control
23	Telnet
25	SMTP
53	DNS
80	HTTP

Port Numbers Example

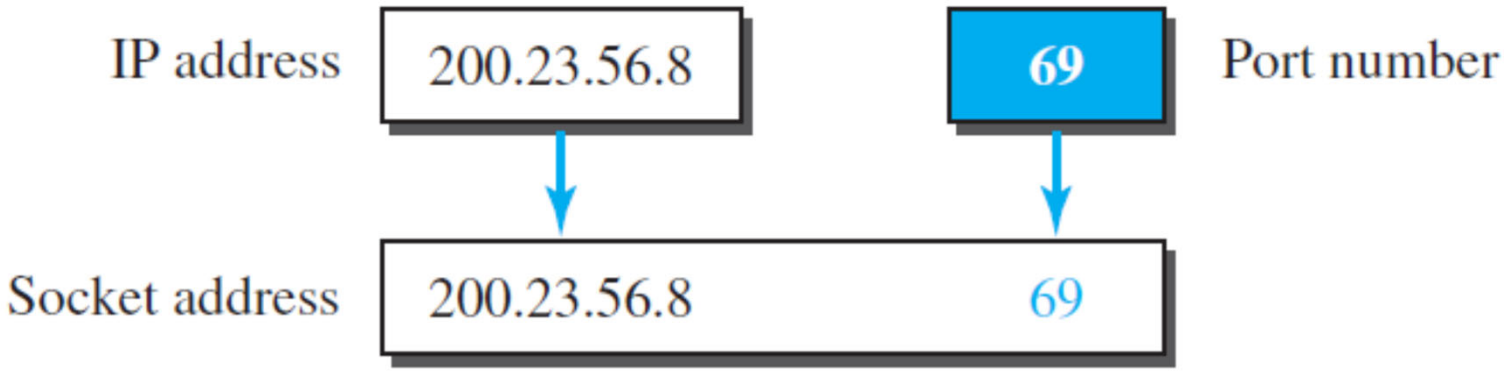


IP Addresses vs. Port Numbers

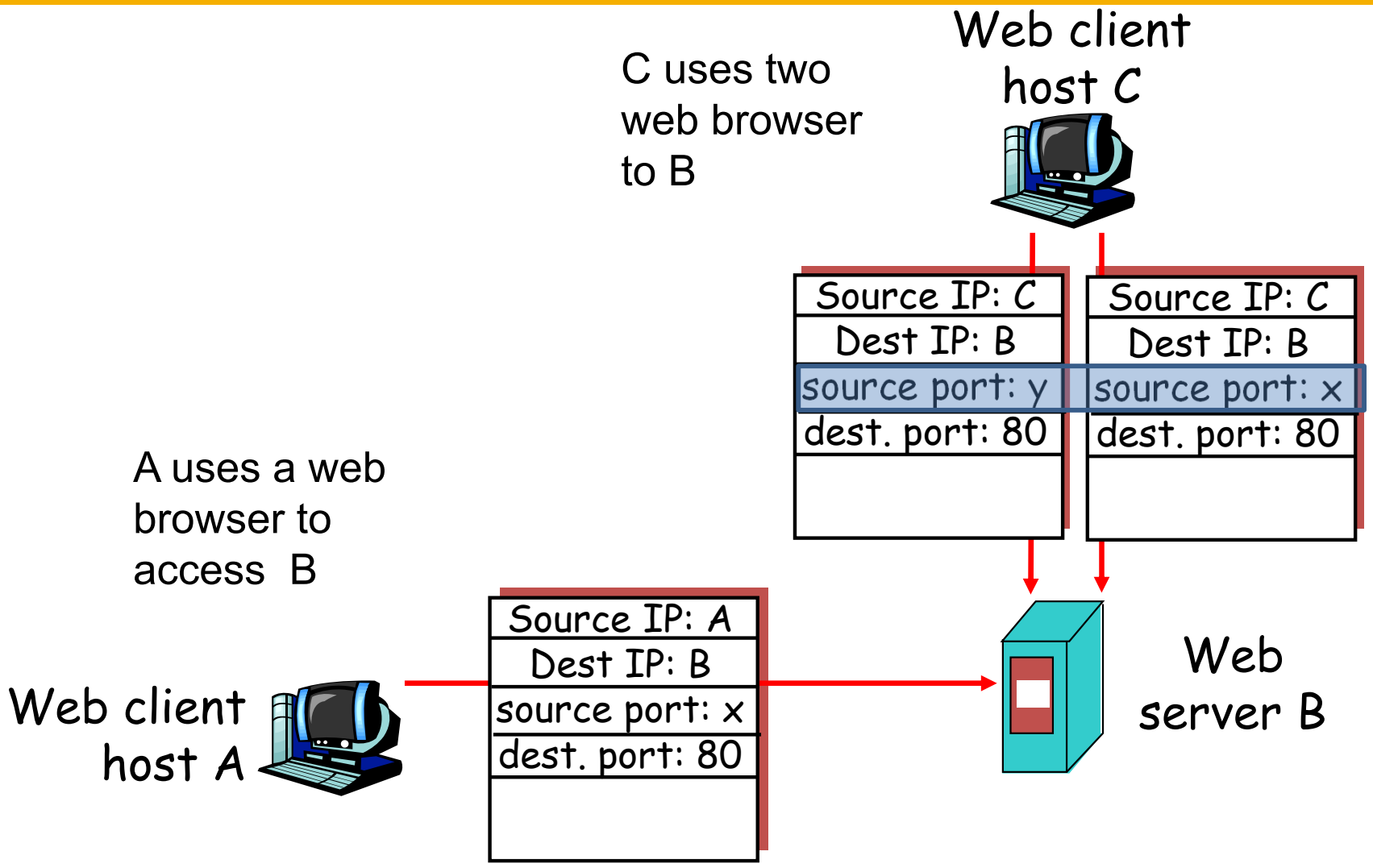


Socket Address

- Socket address: combination of IP address and port number
- Uniquely defines the client/server process
- To use services of the transport layer, we need a pair of socket addresses: client and server socket addresses

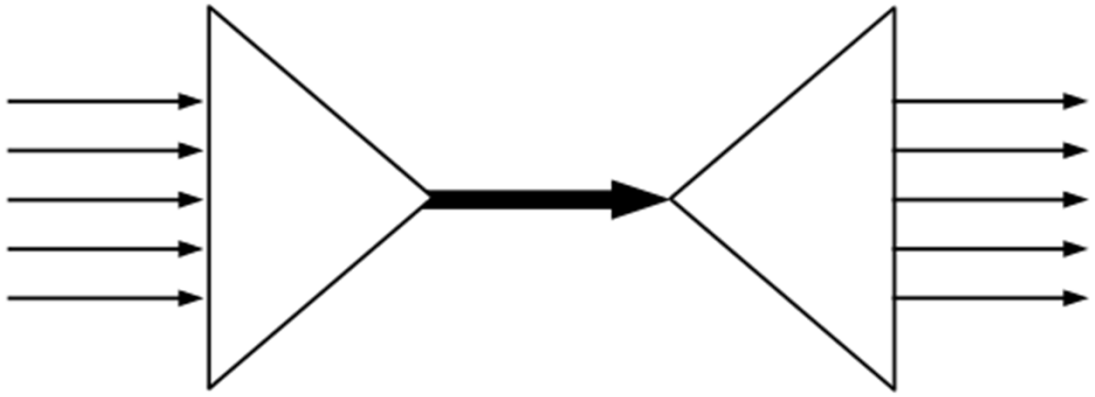


Example


C uses two web browser to B
Web client host C
Source IP: C
Dest IP: B
source port: y
dest. port: 80
Source IP: C
Dest IP: B
source port: x
dest. port: 80
A uses a web browser to access B
Source IP: A
Dest IP: B
source port: x
dest. port: 80
Web client host A
Web server B

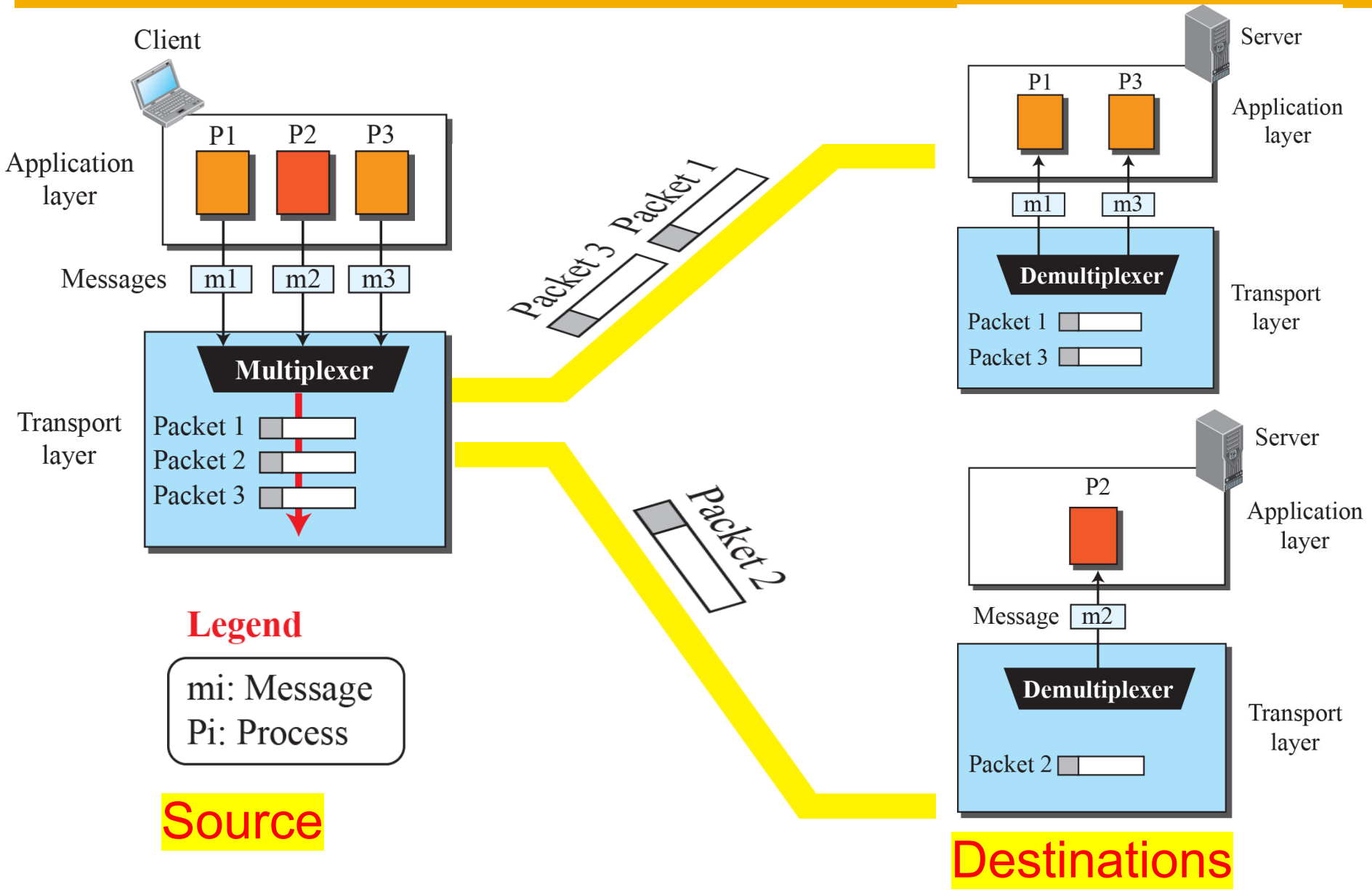
Multiplexing/De-Multiplexing

- Multiplexing (Transport layer at the source): multiple streams/items are combined into one stream to share the medium
- Demultiplexing (Transport layer at the destination): the combined stream is separated to recreate the original streams/items



Multiplexing **De-Multiplexing**

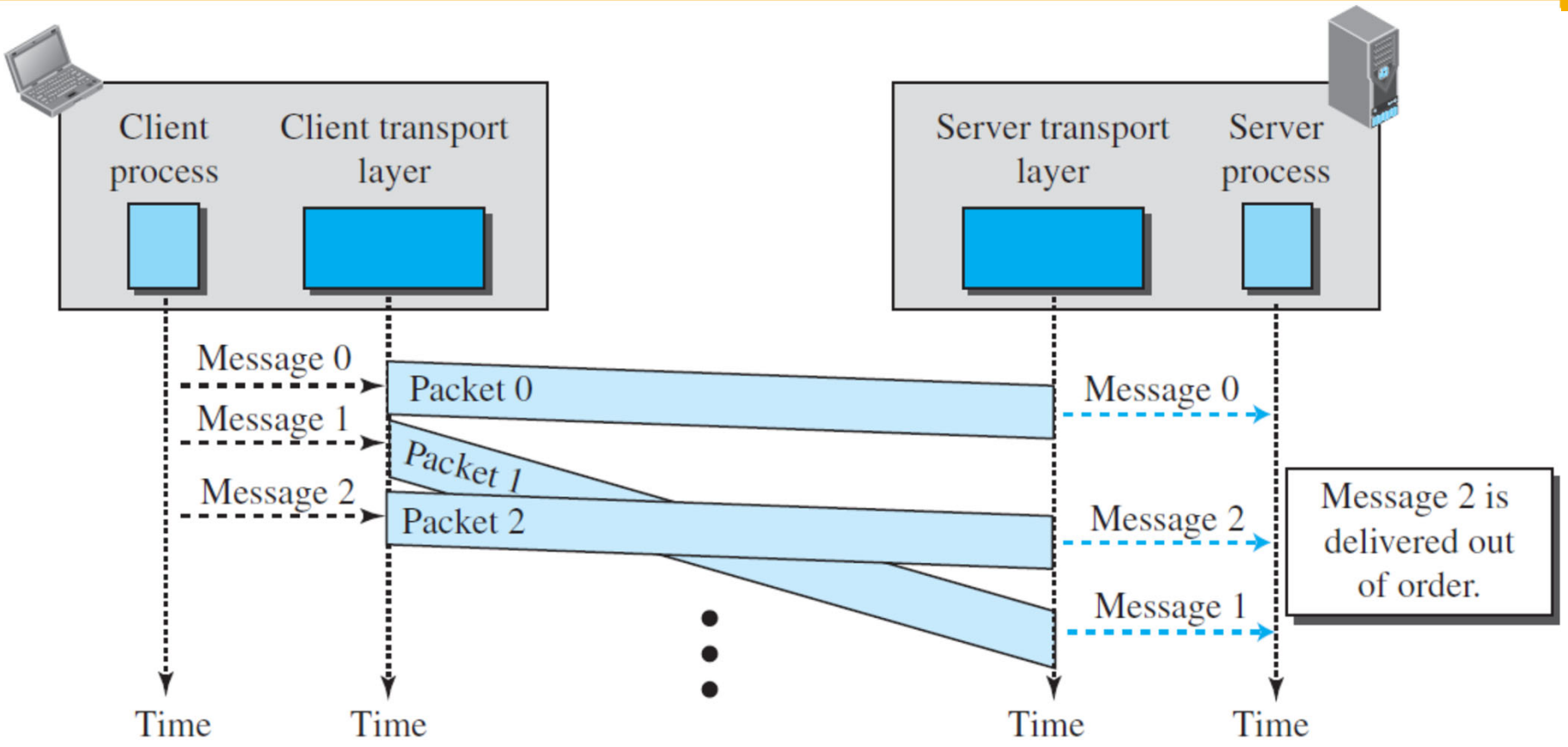
Multiplexing/De-Multiplexing



Connection-less vs. Connection-oriented Service

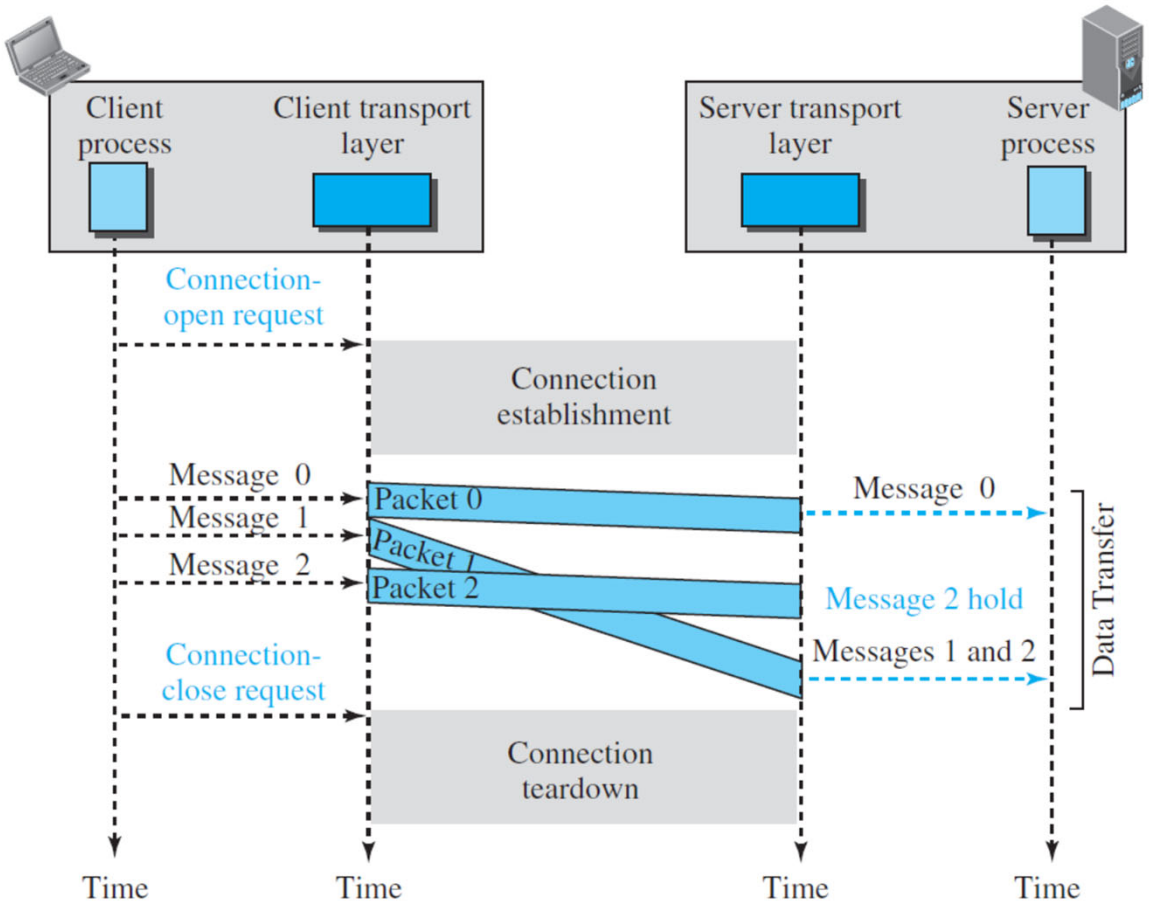
- Connection-less Service: **independency** between packets
  - No need for connection establishment or release
  - Packets are not numbered
  - May be delayed, lost or arrive out of sequence
  - No acknowledgement
  - No flow control, error control, or congestion control can be effectively implemented
- Connection-oriented Service: **dependency** between packets
  - A logical connection is first established between sender and receiver
  - Data transferred /delivered in sequence
  - Connection released

Connection-less Service



- If the three trunks of data belong to the same message, the server may have received out of order message

Connection-oriented Service



- Message can be correctly recovered even when the packets arrive out of order

THE UNIVERSITY OF SYDNEY

Reliable vs. Unreliable

- A reliable transport layer protocol provides
  - Delivery confirmation: acknowledge mechanism
  - Flow control
  - Error control
  - Slower but more complex service
- An unreliable transport layer protocol provides
  - Faster and less complex service
  - No delivery confirmation
  - No flow control
  - No error control

THE UNIVERSITY OF SYDNEY

Ensuring Reliability

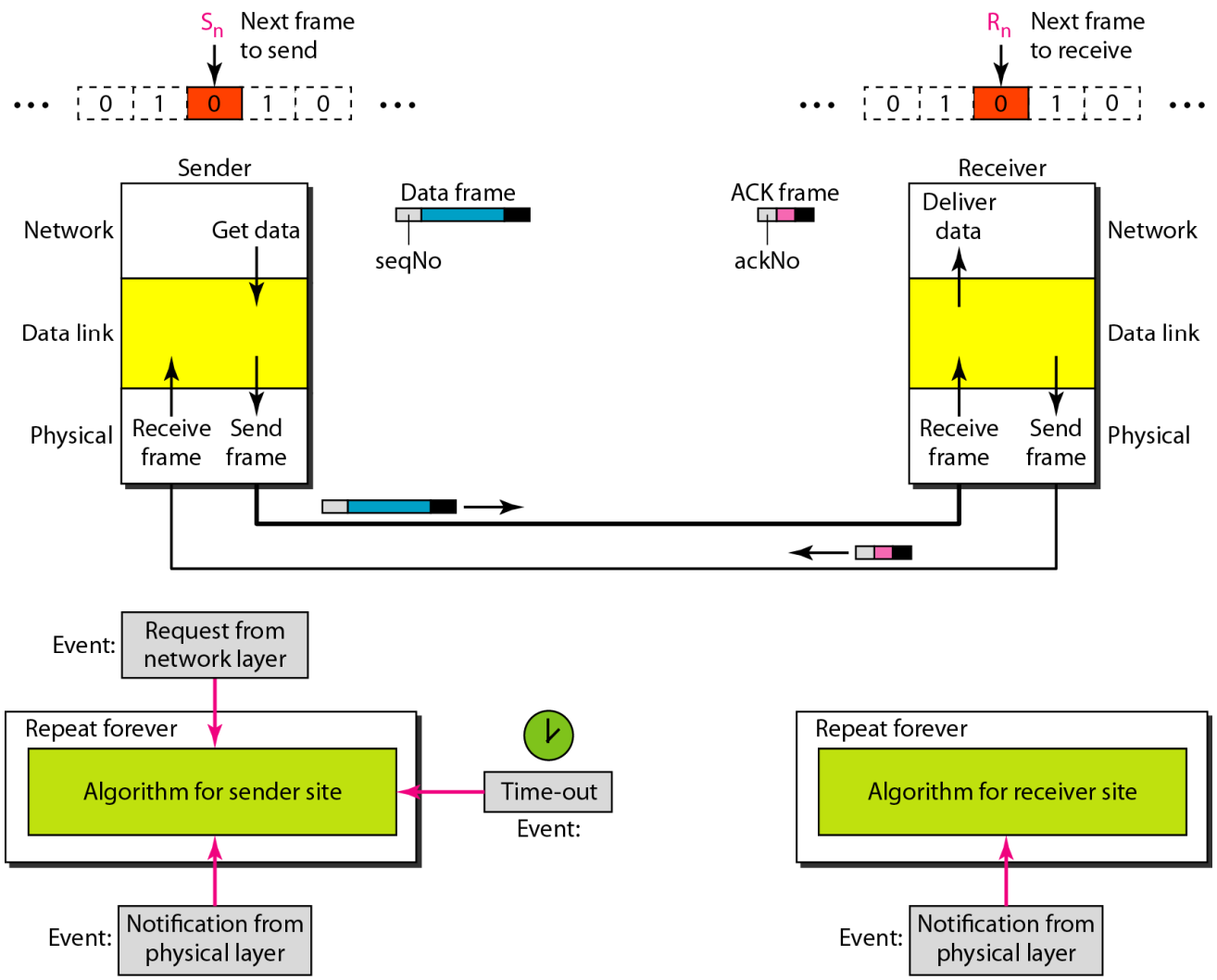
- 3 types of Protocols:
  - Stop-and-wait ARQ (Send one frame at a time)
  - Go-Back-N ARQ (Send several frames at a time)
  - Selective-Repeat ARQ (Send several frames at a time)

THE UNIVERSITY OF SYDNEY

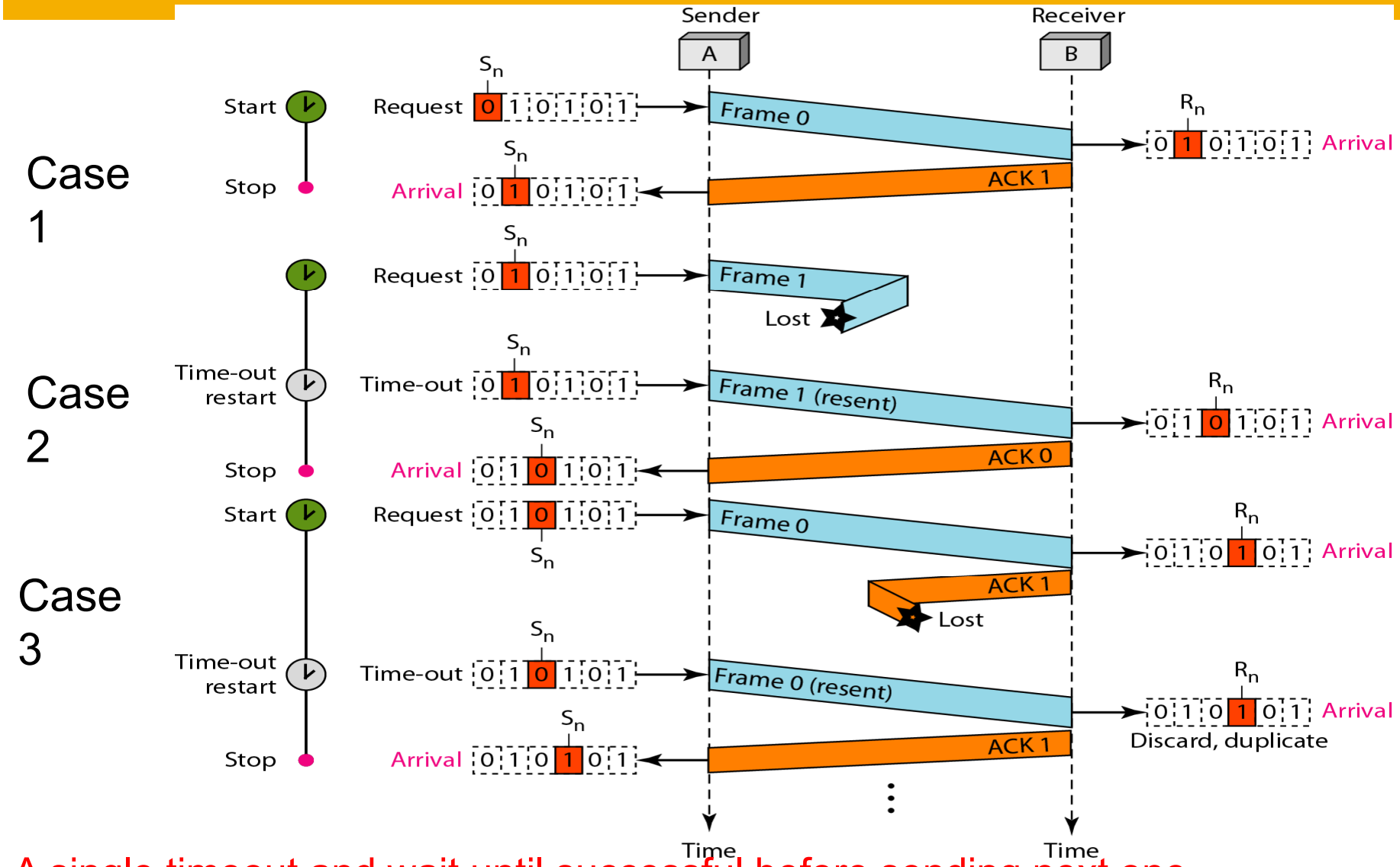
Stop-and-Wait ARQ

- The simplest form of flow control is Stop and Wait ARQ.
- Stop and Wait ARQ works like this:
  - The sending station sends a frame of data and then waits for an acknowledgement from the other station before sending further data
  - The receiving party can stop the flow of data by simply withholding an acknowledgement
  - Source may not send a new frame until the receiver Acknowledges the frame already sent or Timeouts
  - Transmitted frames are numbered by a sequence number based on modulo-2 arithmetic (0,1,0,1,0,1,….)
  - Receiver's ACK frame carries the sequence number of the next frame expected (also based on modulo-2 arithmetic)
- Very inefficient:
  - The sender has to wait for a round trip time (RTT) before another frame can be sent – slow speed
    - RTT: the time that is required for a frame to travel from source to destination and back

Stop-and-Wait ARQ



Stop and Wait ARQ



A single timeout and wait until successful before sending next one

THE UNIVERSITY OF SYDNEY

Go-Back-N ARQ

- To improve the efficiency of transmission, multiple frames must be in transmission while waiting for acknowledgement
- Go-Back-N ARQ can send several frames, before receiving acknowledgements, but keeps copies until acknowledgements arrive
- Frames are sequentially numbered. The sequence numbers are modulo-2^m, where m is the size of the sequence number field in bits.

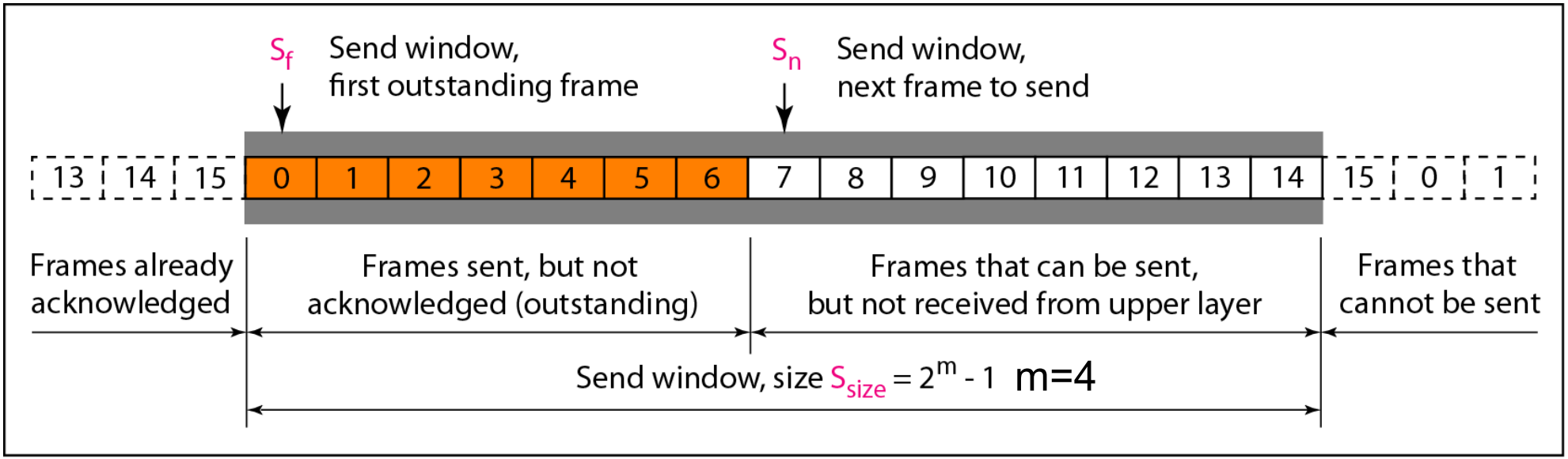
Sliding Window (an abstract concept)

- Defines the range of sequence numbers that is the concern of the sender (send sliding window) and receiver (receive sliding window) .
- Maximum size of sliding window is $2^m - 1$.

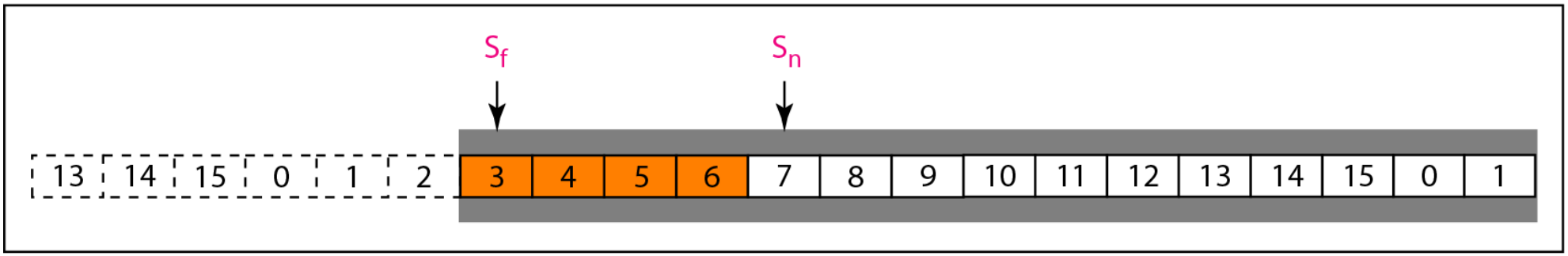
Send Window

- Window divides the possible sequence numbers to four regions.
- The send window's size and location defined by three variables: S_f, S_n, and S_{size}
- The send window can slide one or more slots when a valid acknowledgement arrives (acknowledgements are cumulative)

Send Window for Go-Back-N ARQ



a. Send window before sliding



b. Send window after sliding

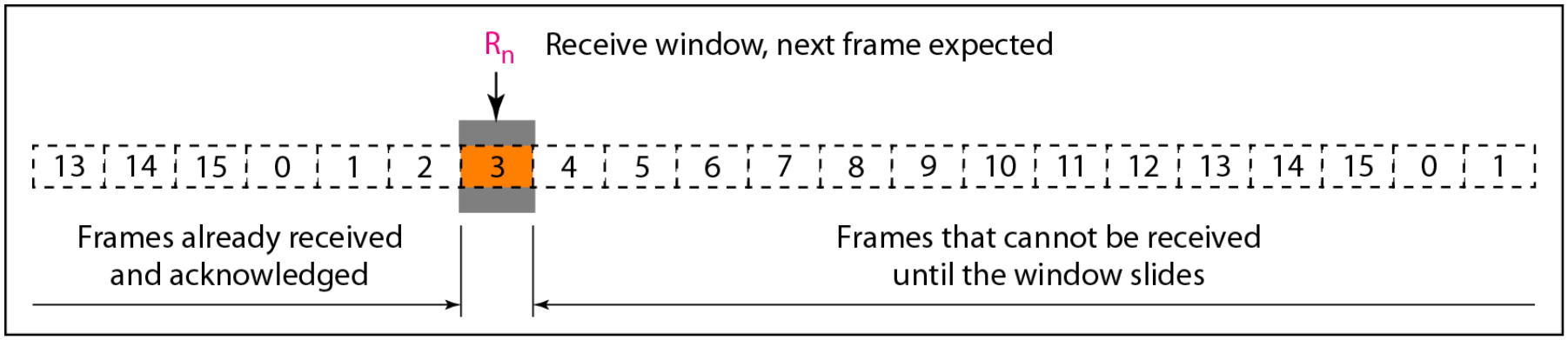
THE UNIVERSITY OF SYDNEY

Receive Window for Go-Back-N ARQ

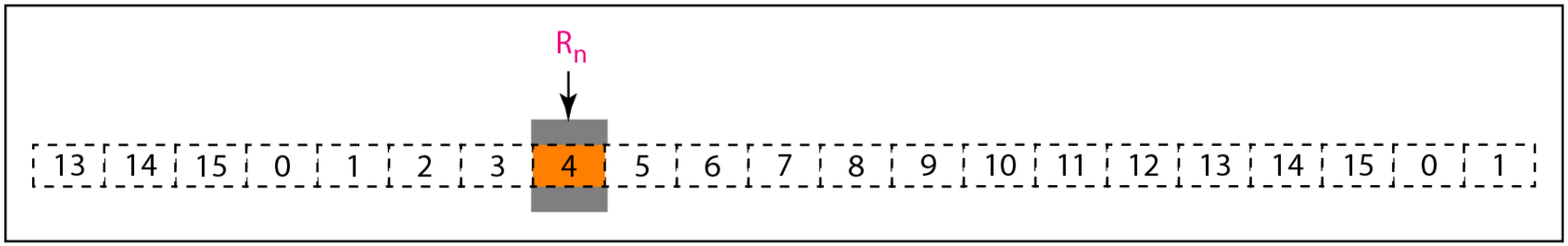
Receive Window

- Makes sure that correct frames are received and correct acknowledgements are sent.
- The size of receive window is always 1 with a single variable R_n.
- Window slides when a correct frame has arrived.
- Sliding occurs one slide at a time.
- Only a frame with a sequence number matching the value R_n is accepted and acknowledged.
- Any frames arriving out of order are discarded until it receives the one it expects.
- Receiver is silent for damaged and out of order frames.
- When receiver is silent, the first unacknowledged frame's timer expires first.
- Then the sender goes back and resends all frames beginning with the one with the expired timer.
- Hence called Go-Back-N.

Receive Window for Go-Back-N ARQ

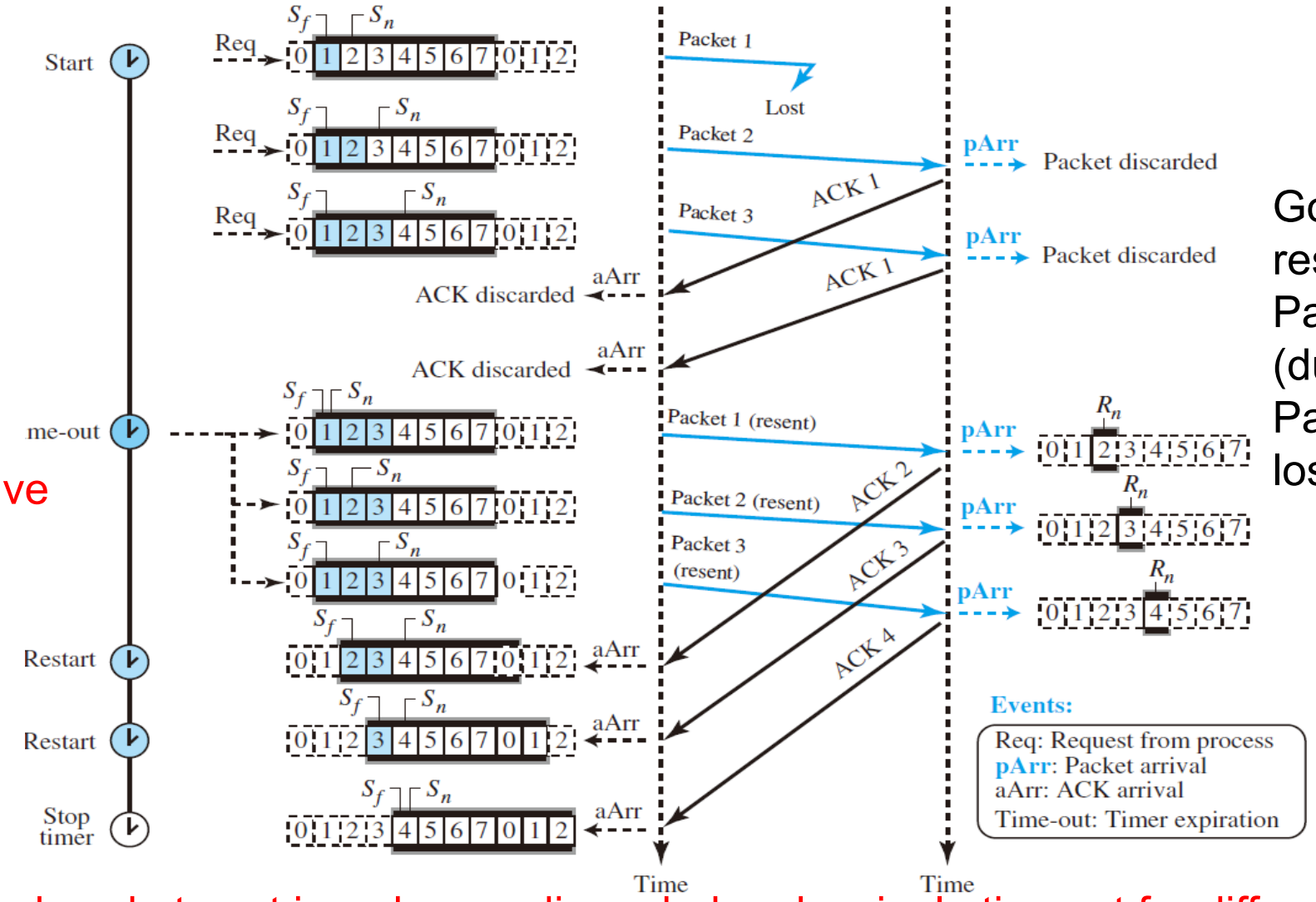


a. Receive window



b. Window after sliding

Example of Go-Back-N



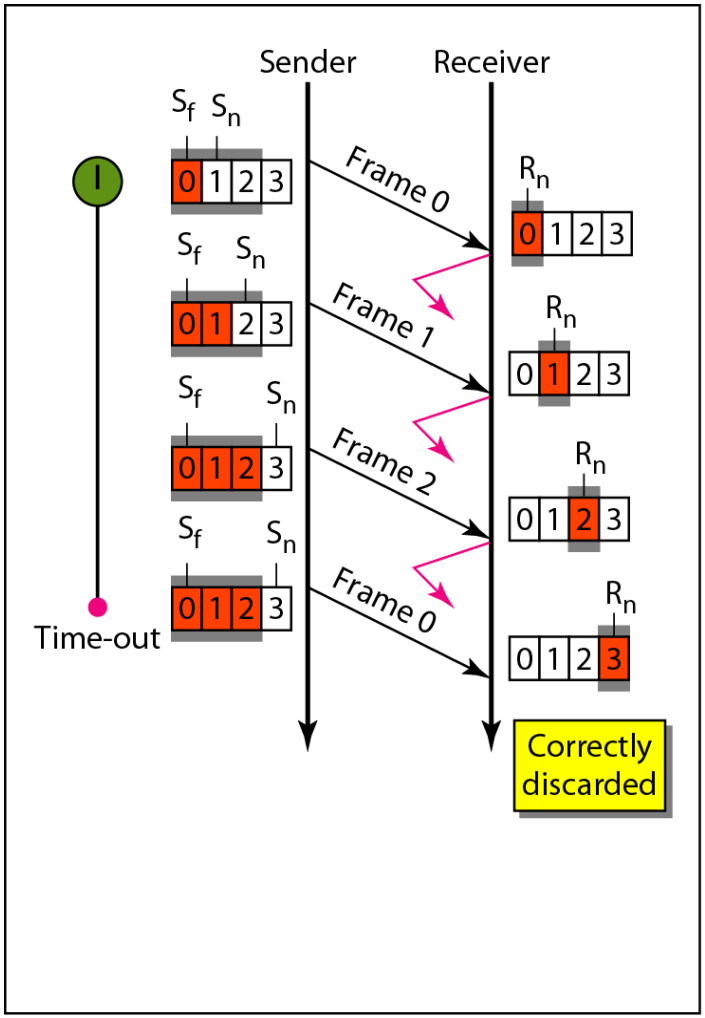
Go back and resend Packets 1,2,3 (due to Packet 1 is lost)

Cumulative ACK

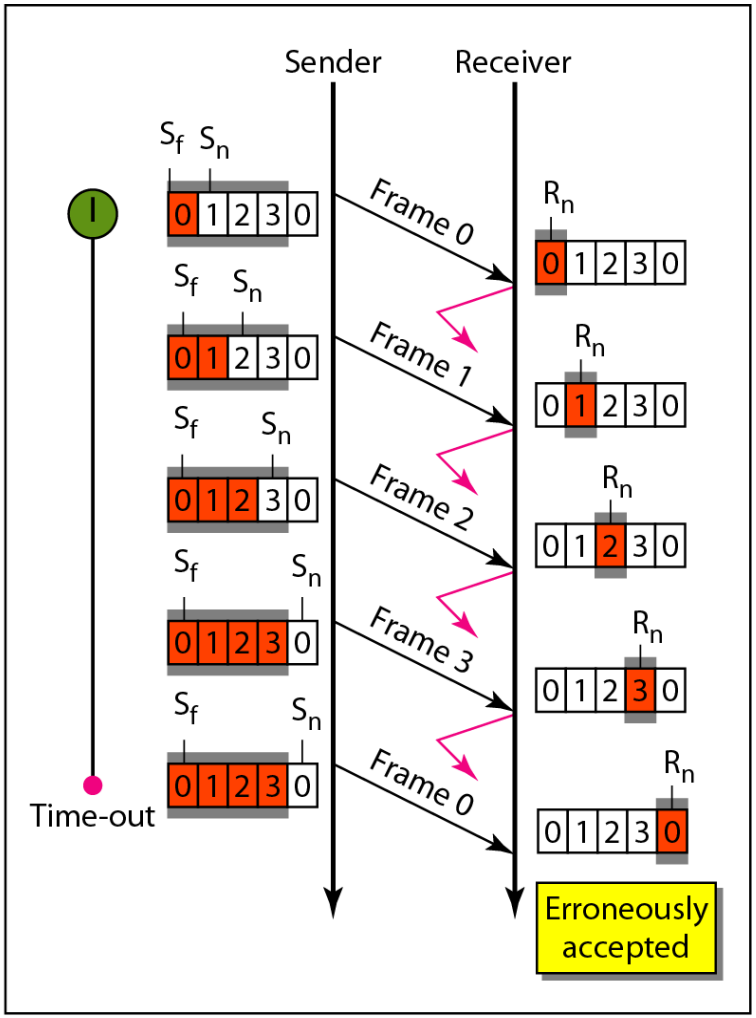
Received packets not in order are discarded and a single timeout for different connections

Send Window Size

- The size of send window < 2^m (m is the size of the sequence number field in bits)
- Why? Lets assume m = 2, window size of 2^m - 1 = 3



a. Window size < 2^m

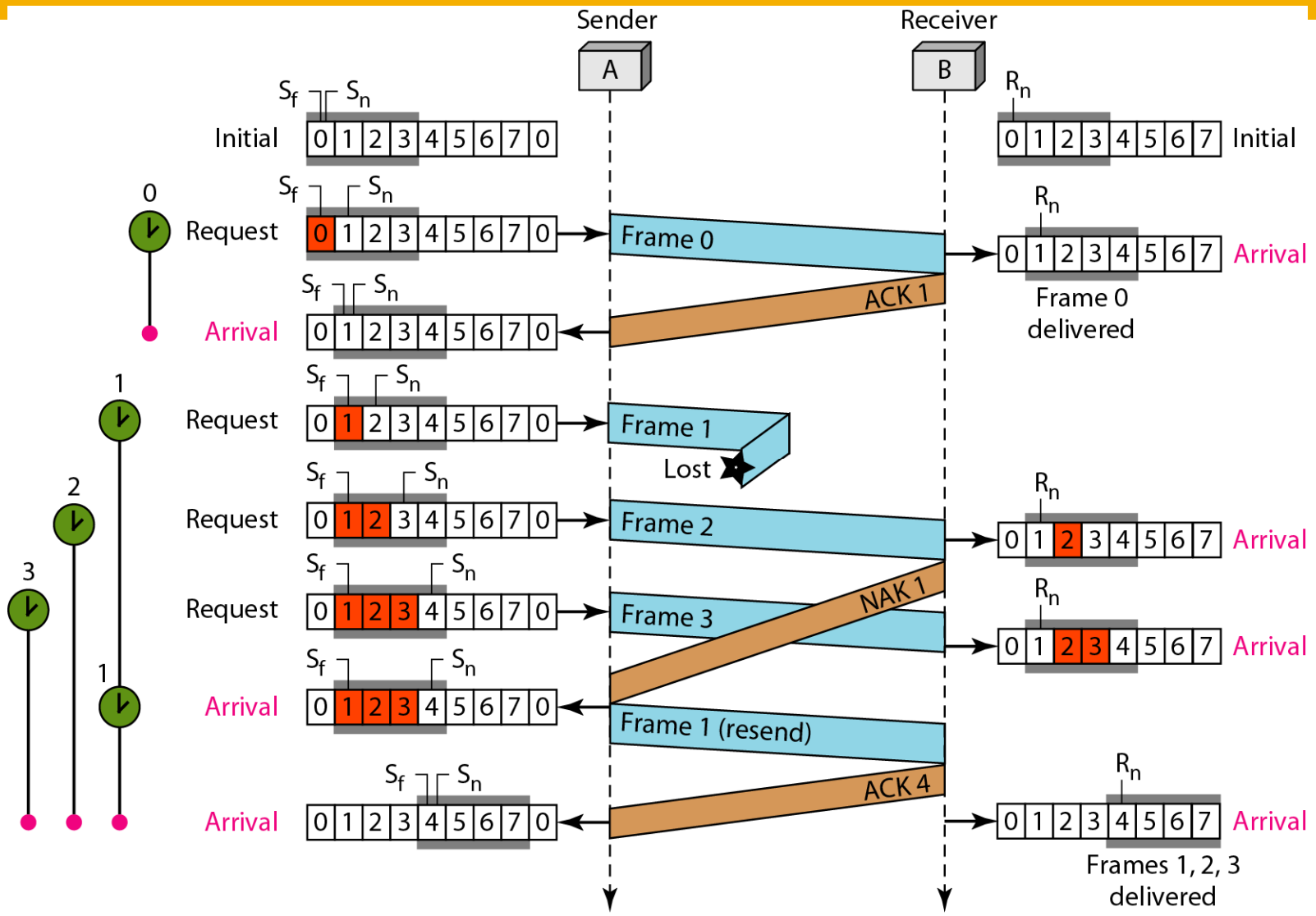


b. Window size = 2^m

Selective Repeat ARQ

- Go-Back-N is very inefficient for noisy links.
- Why resend N frames (3 while only packet 1 is in error in the previous example) when just 1 frame is damaged?
- If a damaged frame is received, a NAK is sent to inform that it has not received the frame it expected.
- One main difference is the number of timers (each frame sent or resent has a timer)

Selective Repeat



Received packets not in order are not discarded and separate timeouts for different connections

Transport Layer Protocols

- User Datagram Protocol (UDP) (aka, unreliable datagram protocol)
  - Connectionless
  - Unreliable
  - Adds no value to services of IP except process-to-process communication
- Transmission Control Protocol (TCP)
  - Connection oriented
  - Reliable transmission concept
  - In-order delivery
  - Congestion control
  - Flow control and error control

UDP: User Datagram Protocol

- Connectionless
  - No handshaking between UDP peers
  - Each UDP segment handled independently
- "Best effort" service, UDP segments may be
  - Lost
  - Delivered out of order

- Advantages of UDP:
  - **Small delay**: no need for connection establishment
  - **Simple**: no connection state at sender, receiver
  - **Low overhead**: small segment header

UDP: Typical Applications

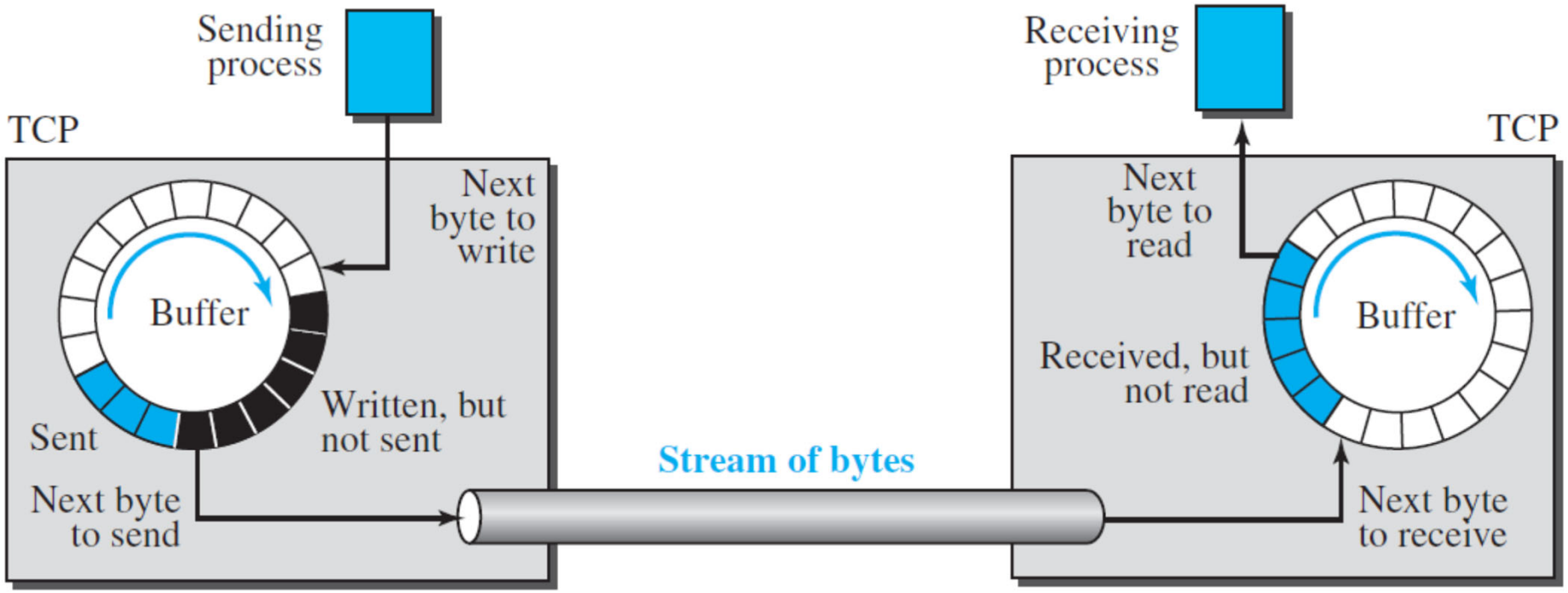
- Processes that require simple request-response communication with little concern for flow and error control
- Processes with internal flow-control and error-control mechanisms
- Management processes such as SNMP (Simple Network Management Protocol)
- Interactive real-time applications that cannot tolerate uneven delay between sections of a received message
- Multicasting
- ……

THE UNIVERSITY OF SYDNEY

TCP: Transmission Control Protocol

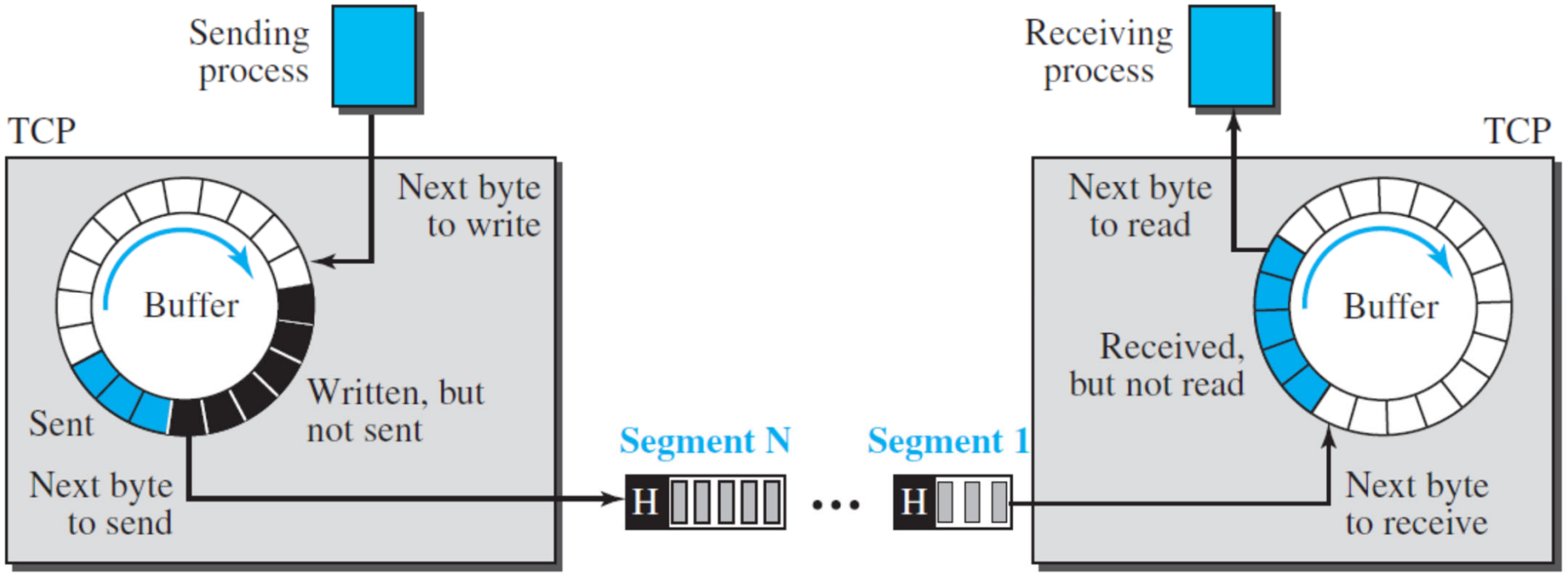
- Connection-oriented reliable communications
- Sending and receiving buffers
- Segmentation of data : grouping a number of bytes together into packets (segment)
- Sequencing and numbering of segments and acknowledgements
- Flow Control – Manage data buffering and coordinate traffic
- Error Control
- Congestion Control – Determines the congestion of the network

Sending and Receiving Buffers



- Why are buffers needed?
- Sender buffer: three chambers (sent, written but not sent, empty that can be used to receive from the process)
- Receiver buffer: two chambers (received but not read, empty that can be used to receive from sender)

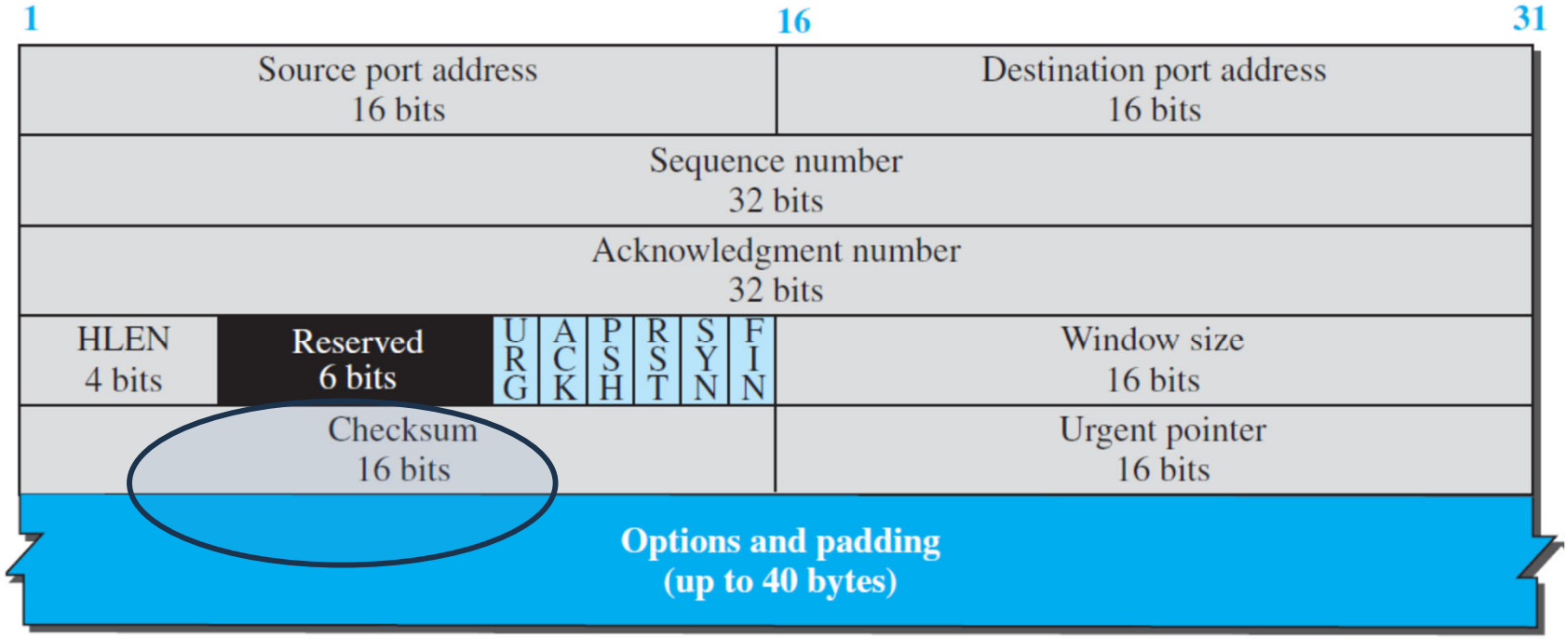
TCP Segments



- Grouping a number of bytes together into packets (segment)
- Add header to each segment for transmission **control**
- Segments may have different sizes
- **Buffer is used to store TCP segments**

TCP Segment Format

a. Segment



b. Header

Checksum provides error detection, compulsory for TCP

TCP Numbering System

- TCP keeps track of the segments via **sequence number** and **acknowledgment number**, both are byte numbers
- All data bytes are numbered (byte number) with 32 bits
- Numbering is independent in each direction and does not necessarily start 0.
- **Sequence number:**
  - The byte number of the first data byte contained in that segment
  - The sequence number of the first segment is the ISN (initial sequence number), which is a random number
- **Acknowledgement number**:
  - The number of the next byte a party expects to receive

THE UNIVERSITY OF SYDNEY

Sequence Number: Example

Example: Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 10001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?

Solution:

- **Segment 1:** Sequence Number: 10001; Range: 10001 to 11000
- **Segment 2:** Sequence Number: 11001; Range: 11001 to 12000
- **Segment 3:** Sequence Number: 12001; Range: 12001 to 13000
- **Segment 4:** Sequence Number: 13001; Range: 13001 to 14000
- **Segment 5:** Sequence Number: 14001; Range: 14001 to 15000

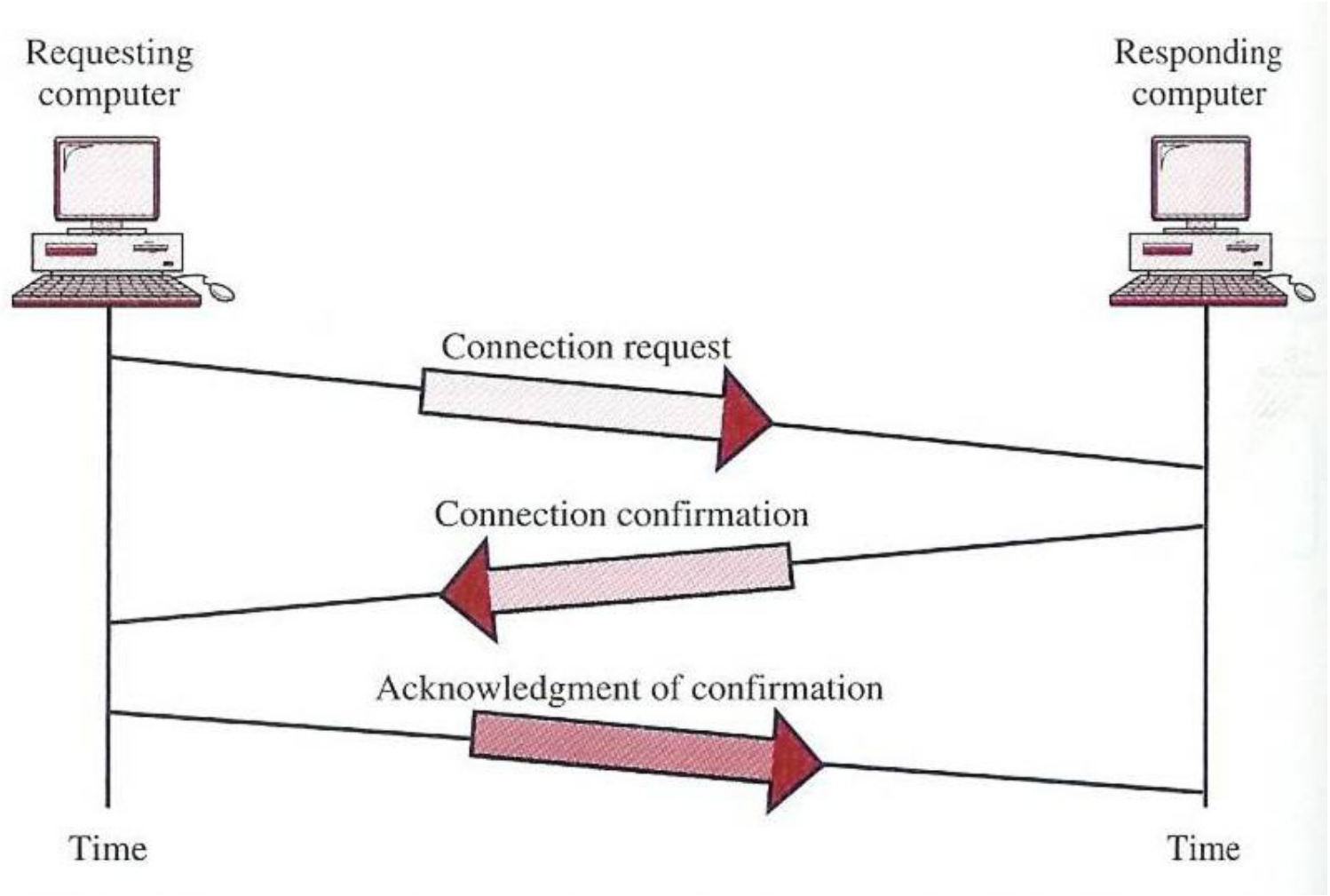
Sequence number on TCP Segment Format Header

THE UNIVERSITY OF SYDNEY

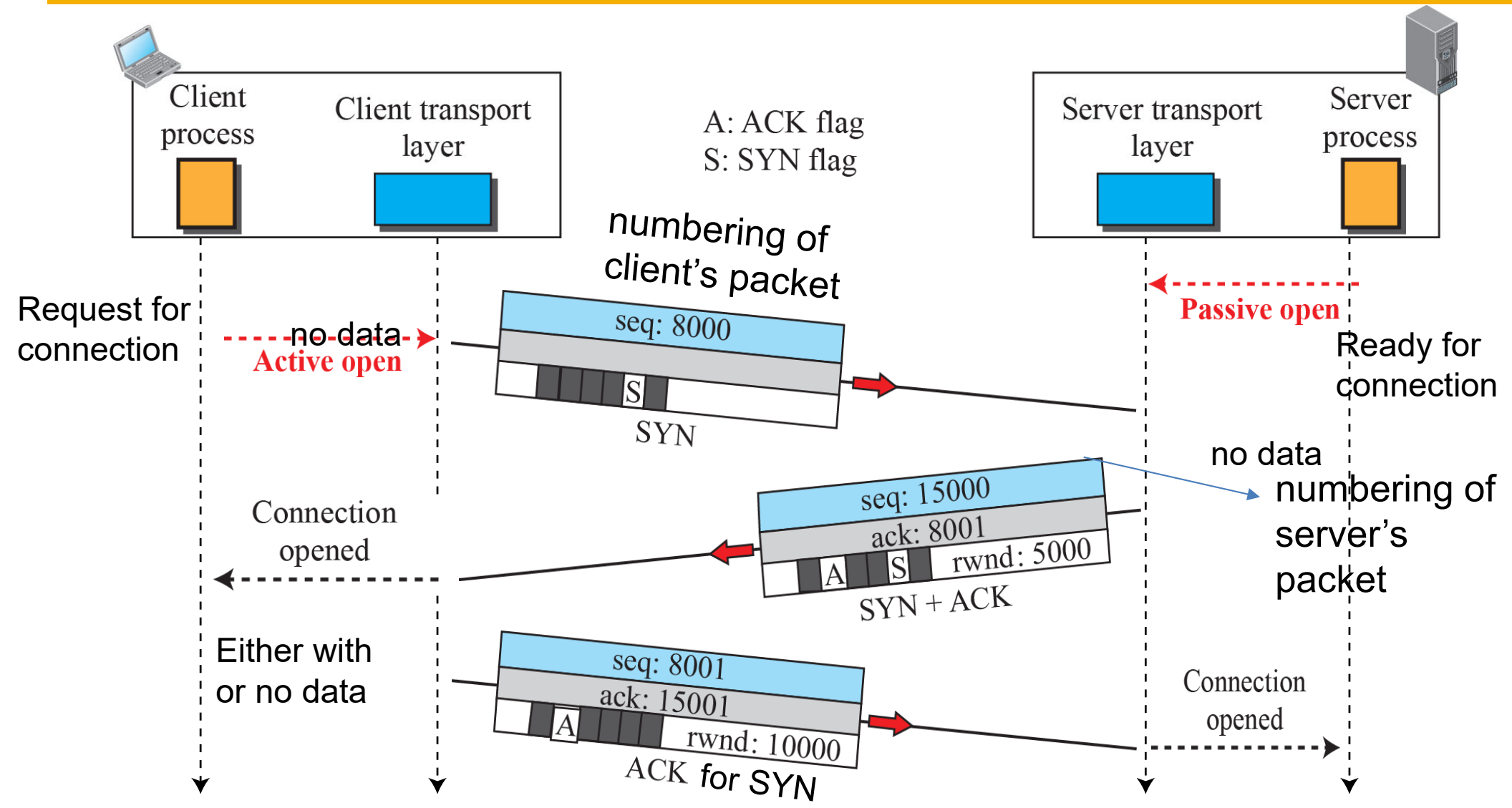
TCP Connection

- How can we transmit the TCP segments in the buffer?
- Connection-oriented transmission requires three phases:
  - Connection establishment
  - Data transfer
  - Connection termination
- The connection establishment in TCP is called **three-way handshaking**.
- Once a connection is established, data may flow back and forth until the connection is closed.

Establishing Connections

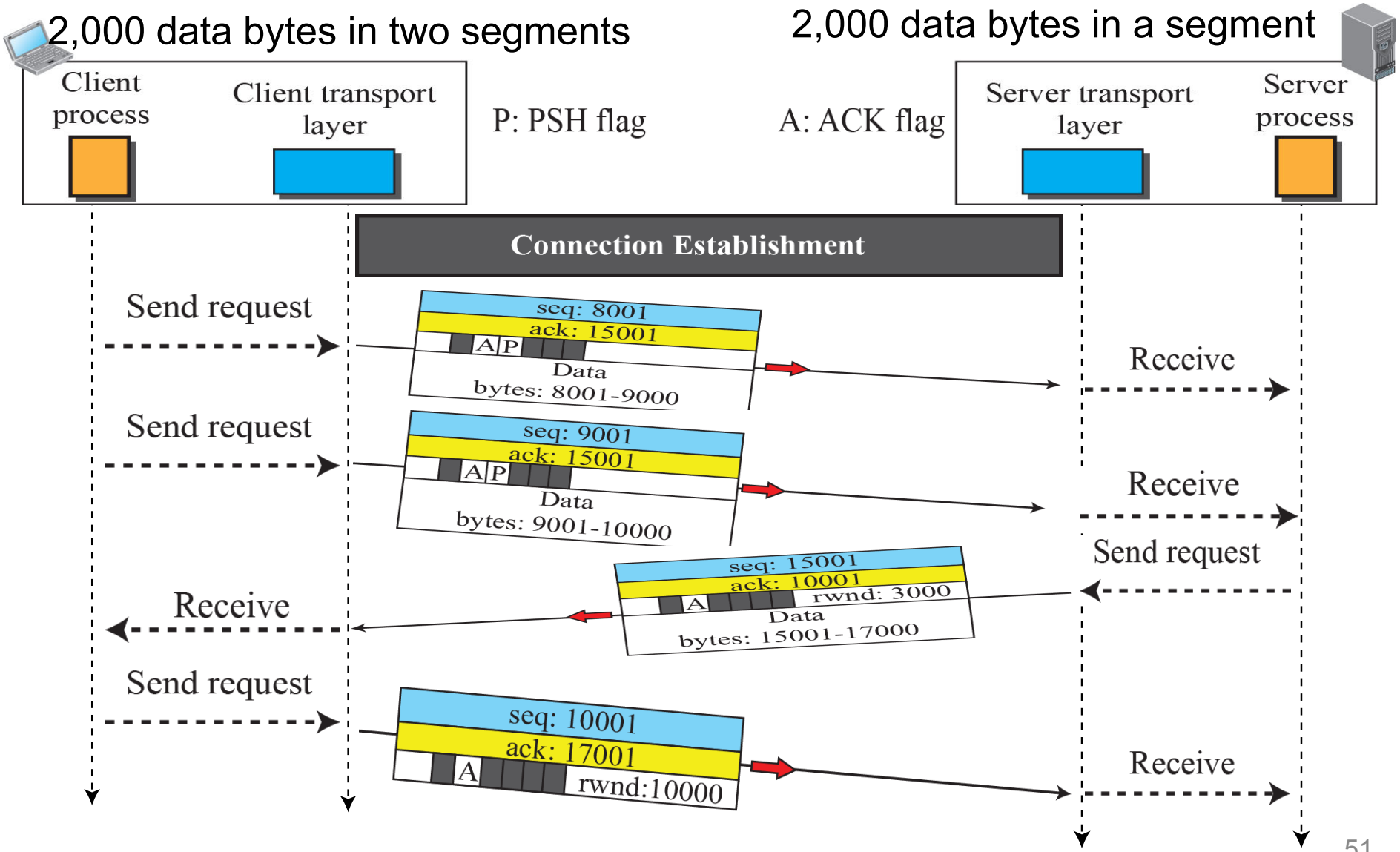


Establishing Connections – three-way handshake



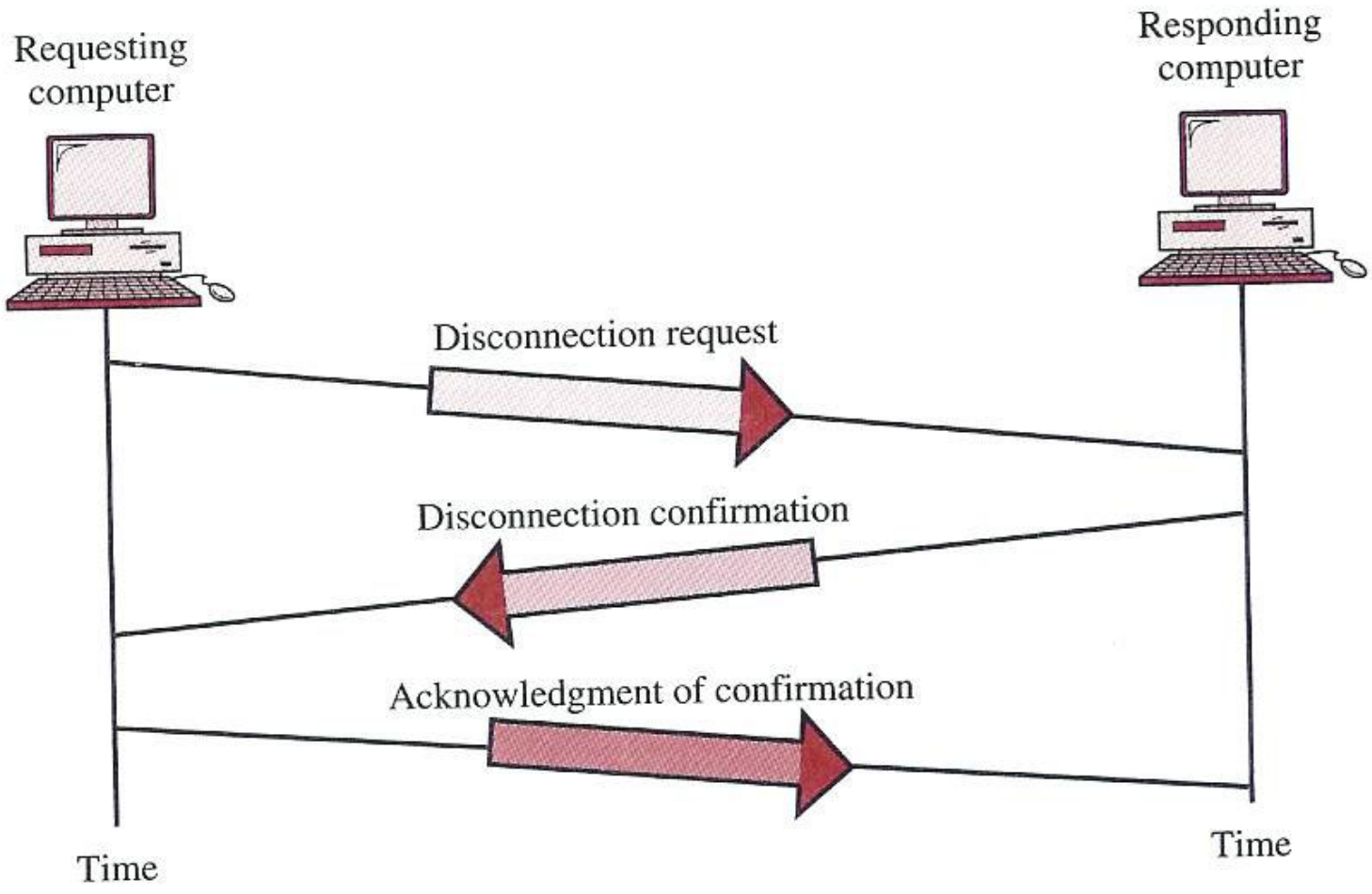
rwnd: received window of the client/server for flow control (to be explained later)

Data Transfer

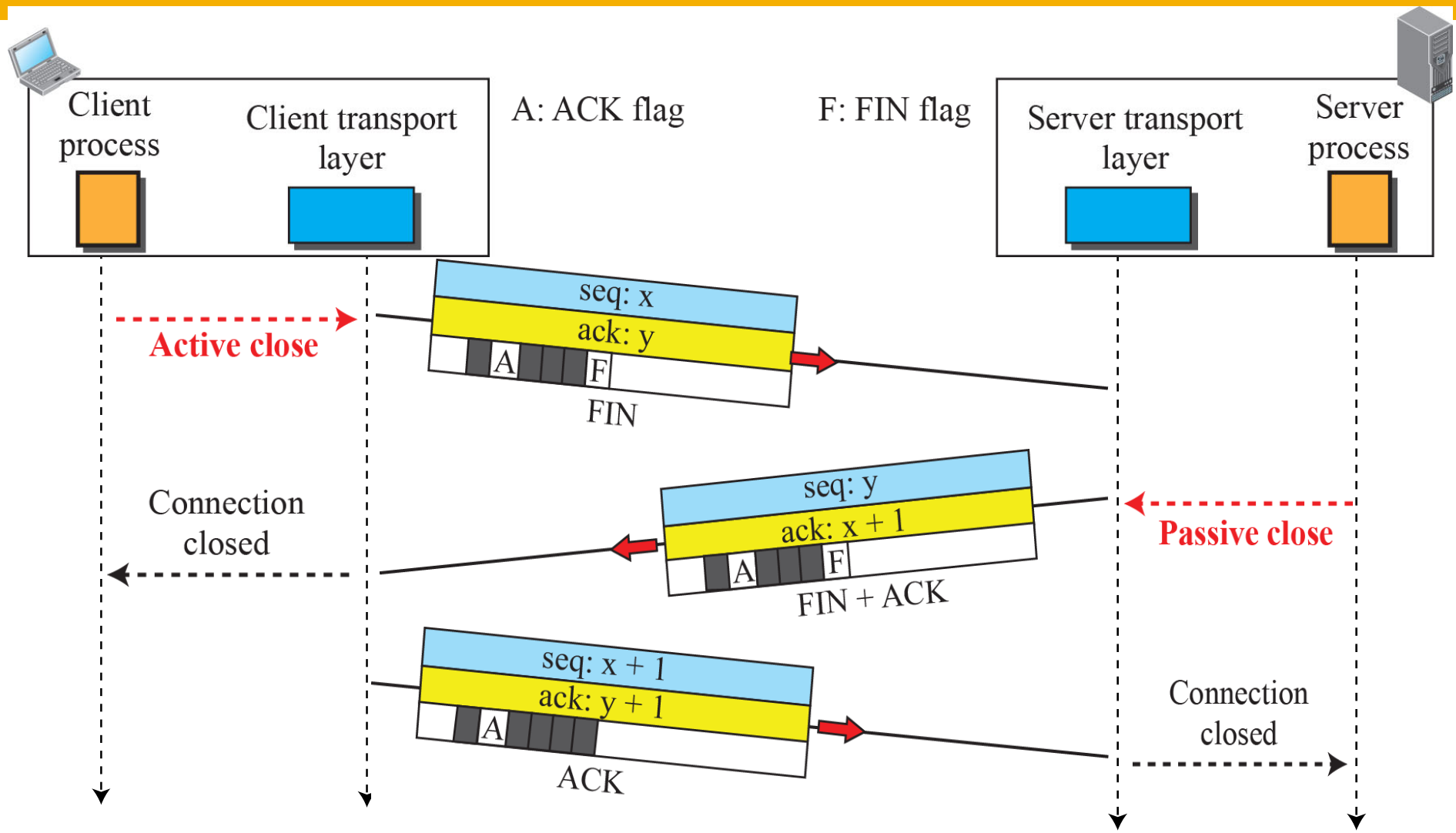


PSH flag indicates data transmissions from client

Terminating Connections – three-way handshake



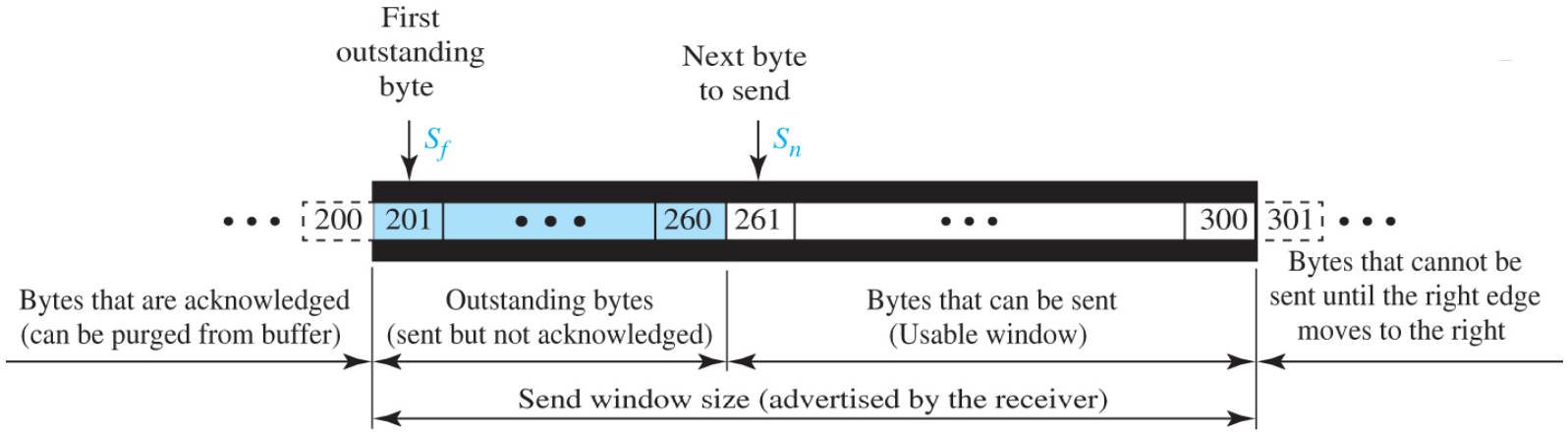
Terminating Connections – three-way handshake



FIN flag is sent together with the last data segment

TCP Flow Control - Window

- Flow control: balances the rate a sender transmits sequences with the rate the receiver can use the sequences
- Flow control by managing Transmission Window size at the sender and receiver

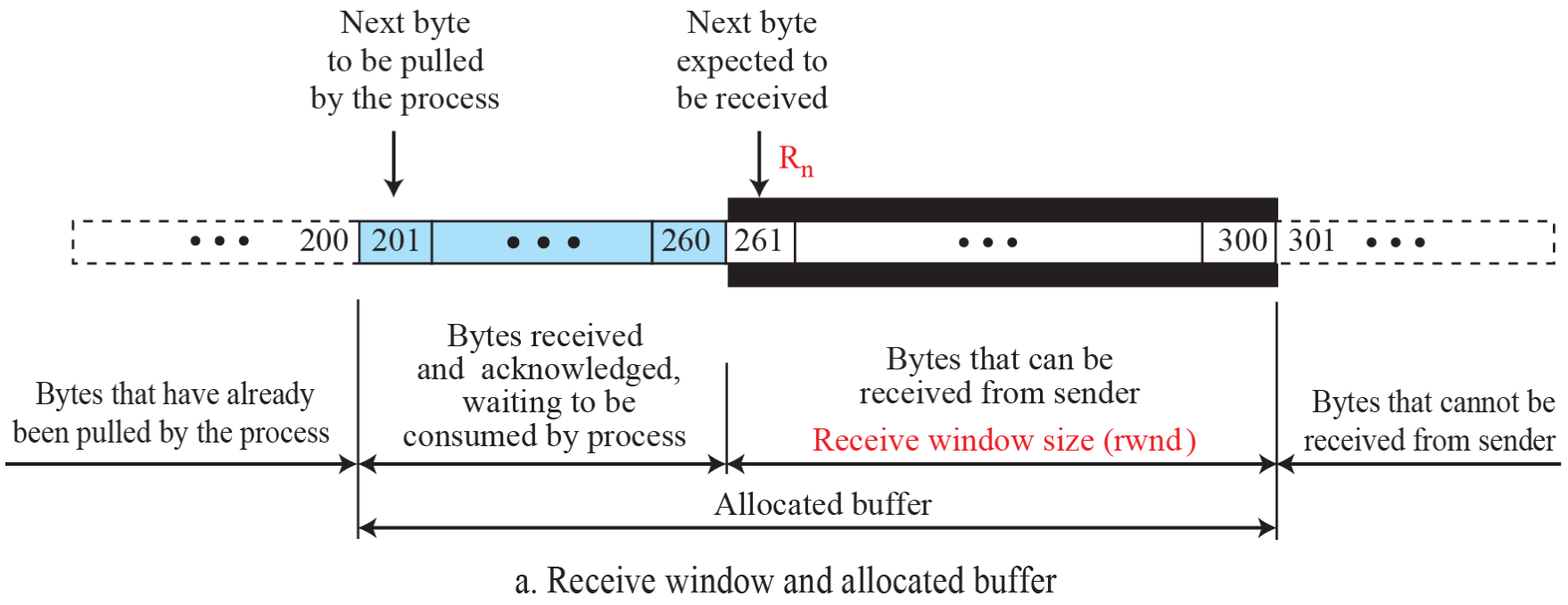


a. Send window

b. Opening, closing, and shrinking send window

TCP Flow Control - Window

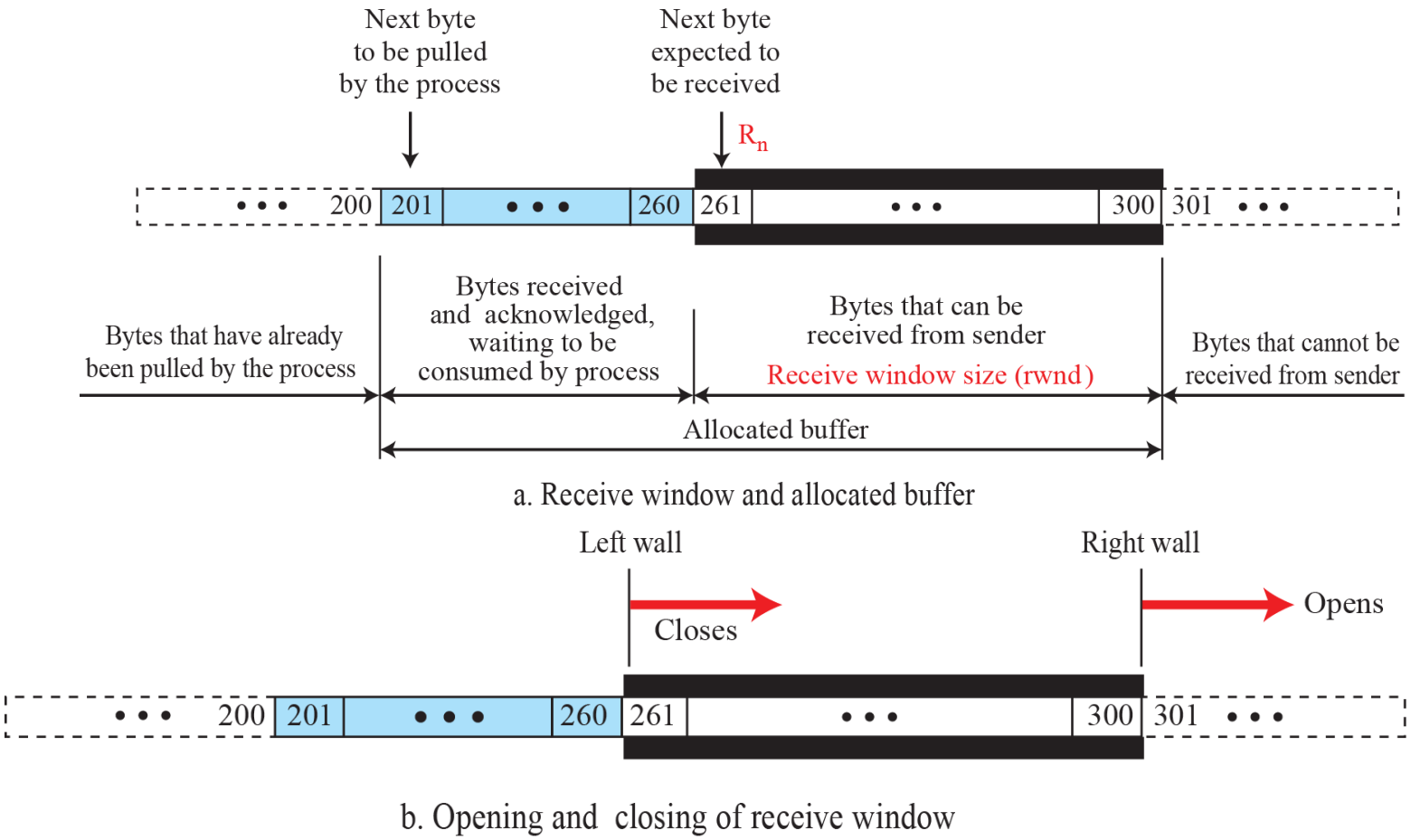
- The receiver controls the amount of data that can be transmitted via the **receive window size** (***rwnd***) . Sender obeys this



a. Receive window and allocated buffer

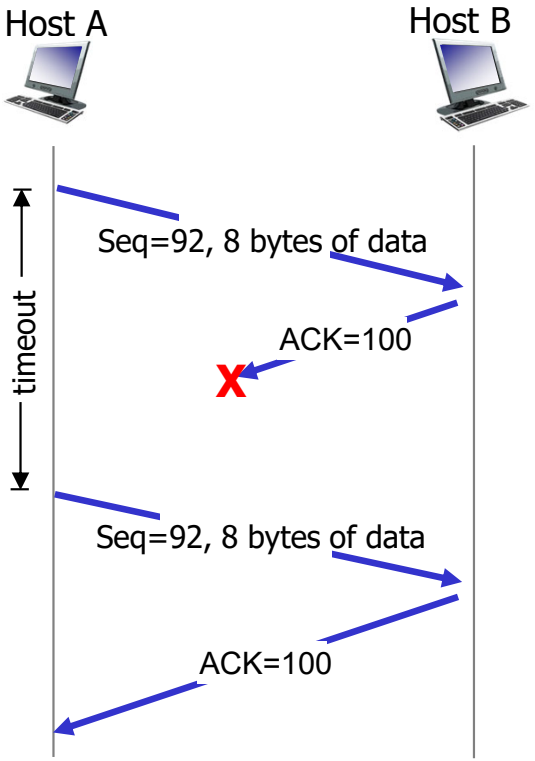
rwnd = buffer size – number of waiting bytes to be pulled

TCP Flow Control - Window

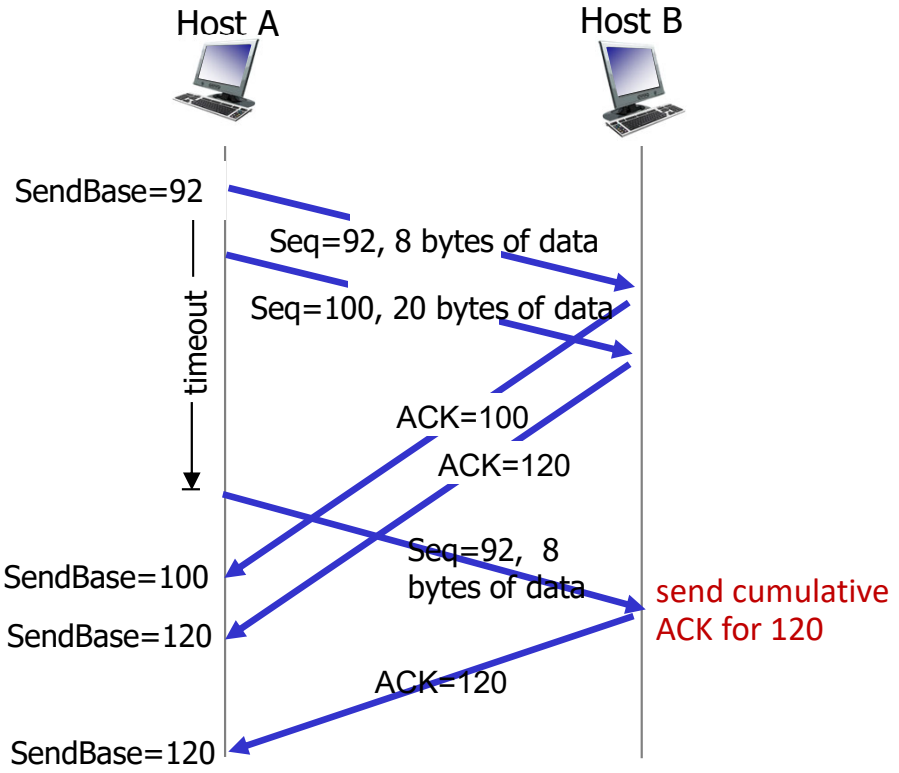


- Receive window getting smaller when more bytes arrive from the sender
- It opens (become bigger) when more bytes are pulled by the process

TCP Flow Control – Example



lost ACK scenario

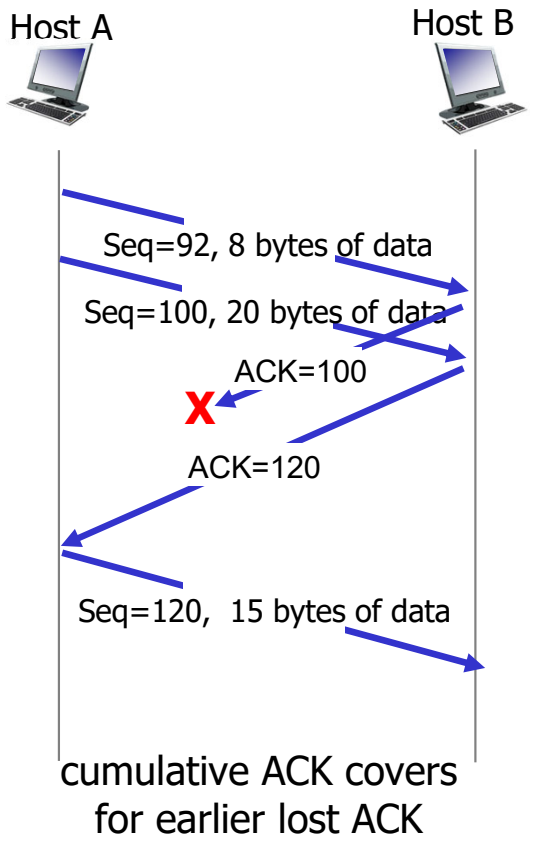


premature timeout

TCP: retransmission scenarios

THE UNIVERSITY OF SYDNEY

TCP Flow Control – Example



Resemblance to Go-Back-N

TCP: retransmission scenarios

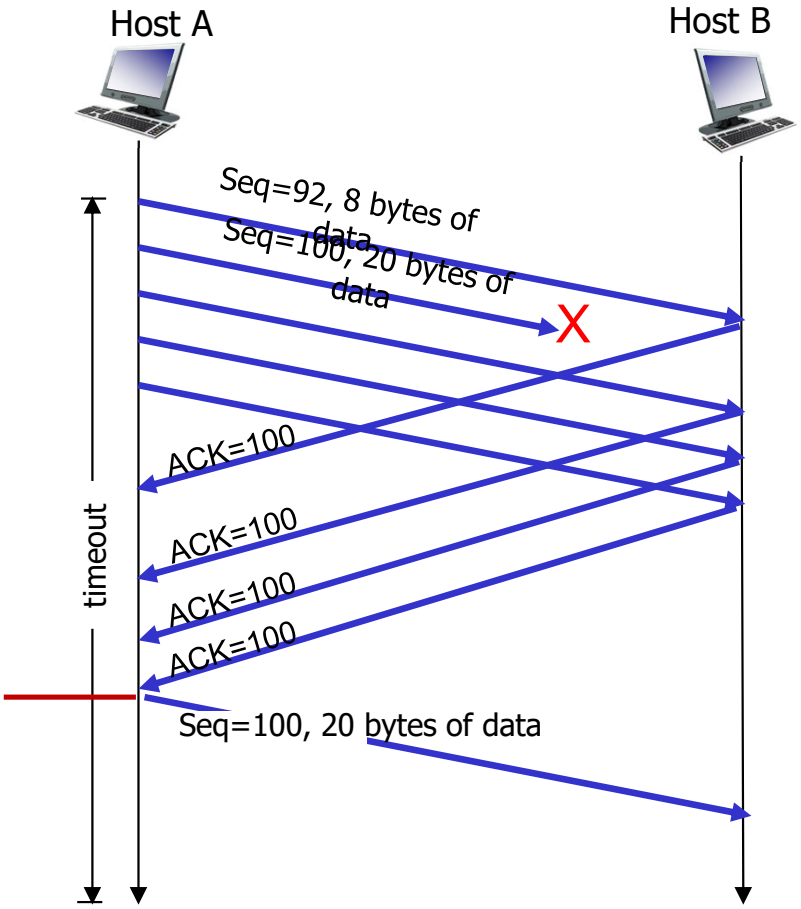
TCP Flow Control – Example

TCP fast retransmit

if sender receives 3 additional ACKs for same data ("triple duplicate ACKs"), resend unACKed segment with smallest seq #

- likely that unACKed segment lost, so don't wait for timeout

Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!



TCP fast retransmit implies selective repeat

THE UNIVERSITY OF SYDNEY

TCP Error Control

- Error control includes mechanisms for detecting and correcting
  - Corrupted segments
  - Lost segments
  - Out-of-order segments
  - Duplicated segments

- This is done with the use of
  - Checksum (embedded in TCP segment)
  - acknowledgements and timeouts

Checksum Example (16 bit header)

4	5	0	28
49.153		0	0
4	17	0	
10.12.14.5			
12.6.7.9			

4, 5, and 0	4	5	0	0
28	0	0	1	C
49.153	C	0	0	1
0 and 0	0	0	0	0
4 and 17	0	4	1	1
0	0	0	0	0
10.12	0	A	0	C
14.5	0	E	0	5
12.6	0	C	0	6
7.9	0	7	0	9
Sum	1	3	4	4	E
Wrapped sum		3	4	4	F
Checksum		**C**	**B**	**B**	**0**

Hexadecimal representation for each 4 bits 0-15 → 0-F

The new checksum, CBB0, is inserted in the checksum field

Wrapped F=E+1

Set 1 to 0 and 0 to 1

Checksum Example (16 bit header)

4	5	0	28
49.153		0	0
4	17	0	
10.12.14.5			
12.6.7.9			

2

4, 5, and 0	4	5	0	0
28	0	0	1	C
49.153	C	0	0	1
0 and 0	0	0	0	0
4 and 17	0	4	1	1
0	0	0	0	0
10.12	0	A	0	C
14.5	0	E	0	5
12.6	0	C	0	6
7.9	0	7	0	9
Sum	1	3	4	4	E
Wrapped sum		3	4	4	F
Checksum		**C**	**B**	**B**	**0**

Hexadecimal representation for each 4 bits 0-15 → 0-F

0+12+1+1+12+5+6+9=46

0x2^7 + 0x2^6 + 1x2^5 + 0x2^4 + 14 = 46

2 + E

The new checksum, CBB0, is inserted in the checksum field

Wrapped F=E+1

Set 1 to 0 and 0 to 1

Checksum Example (16 bit header)

4	5	0	28
49.153		0	0
4	17	0	
10.12.14.5			
12.6.7.9			

4, 5, and 0	→	4	5	0	0
28	→	0	0	1	C
49.153	→	C	0	0	1
0 and 0	→	0	0	0	0
4 and 17	→	0	4	1	1
0	→	0	0	0	0
10.12	→	0	A	0	C
14.5	→	0	E	0	5
12.6	→	0	C	0	6
7.9	→	0	7	0	9
Sum	→	1	3	4	4	E
Wrapped sum	→		3	4	4	F
Checksum	→		**C**	**B**	**B**	**0**

0 2

Hexadecimal representation for each 4 bits 0-15 → 0-F

2+1+1=4

0x2^5 + 0x2^4 + 4 = 4

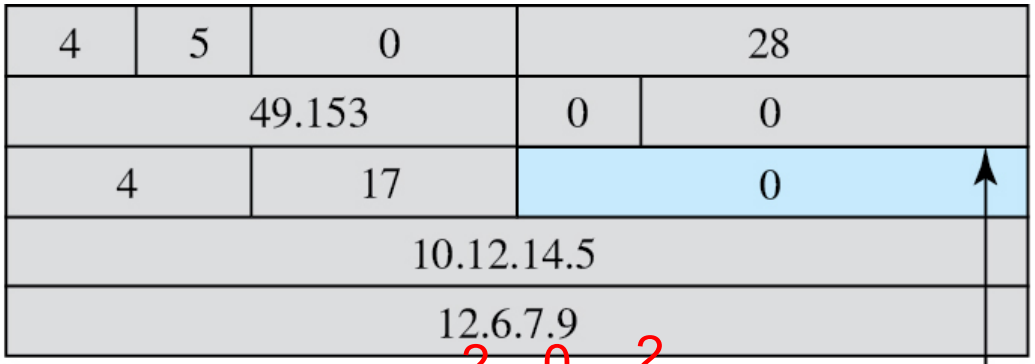
0 + 4

The new checksum, CBB0, is inserted in the checksum field

Wrapped F=E+1

Set 1 to 0 and 0 to 1

Checksum Example (16 bit header)



		2	0	2	
4, 5, and 0	→	4	5	0	0
28	→	0	0	1	C
49.153	→	C	0	0	1
0 and 0	→	0	0	0	0
4 and 17	→	0	4	1	1
0	→	0	0	0	0
10.12	→	0	A	0	C
14.5	→	0	E	0	5
12.6	→	0	C	0	6
7.9	→	0	7	0	9
Sum	→	1	3	4	4	E
Wrapped sum	→		3	4	4	F
Checksum	→		**C**	**B**	**B**	**0**

Hexadecimal representation for each 4 bits 0-15 → 0-F

5+4+10+14+12+7=52

1x2^5 + 1x2^4 + 4 = 52

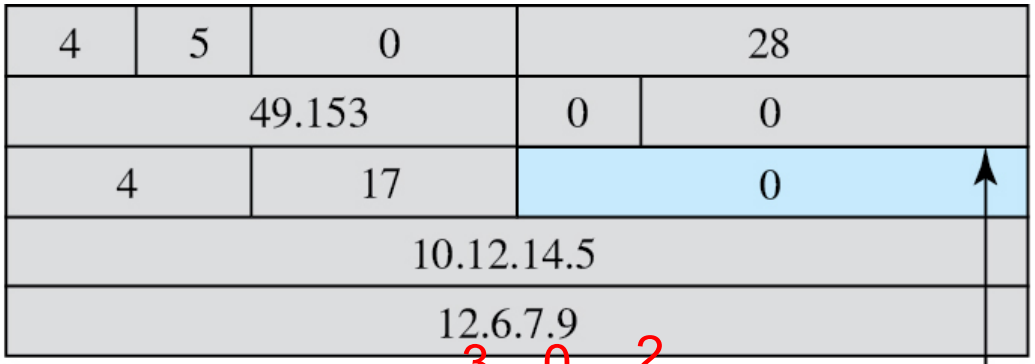
3 + 4

The new checksum, CBB0, is inserted in the checksum field

Wrapped F=E+1

Set 1 to 0 and 0 to 1

Checksum Example (16 bit header)



		3	0	2	
4, 5, and 0	→	4	5	0	0
28	→	0	0	1	C
49.153	→	C	0	0	1
0 and 0	→	0	0	0	0
4 and 17	→	0	4	1	1
0	→	0	0	0	0
10.12	→	0	A	0	C
14.5	→	0	E	0	5
12.6	→	0	C	0	6
7.9	→	0	7	0	9
Sum	→	1	3	4	4	E
Wrapped sum	→		3	4	4	F
Checksum	→		**C**	**B**	**B**	**0**

Hexadecimal representation for each 4 bits 0-15 → 0-F

3+4+12=19

0x2^5 + 1x2^4 + 3 = 19

1 + 3

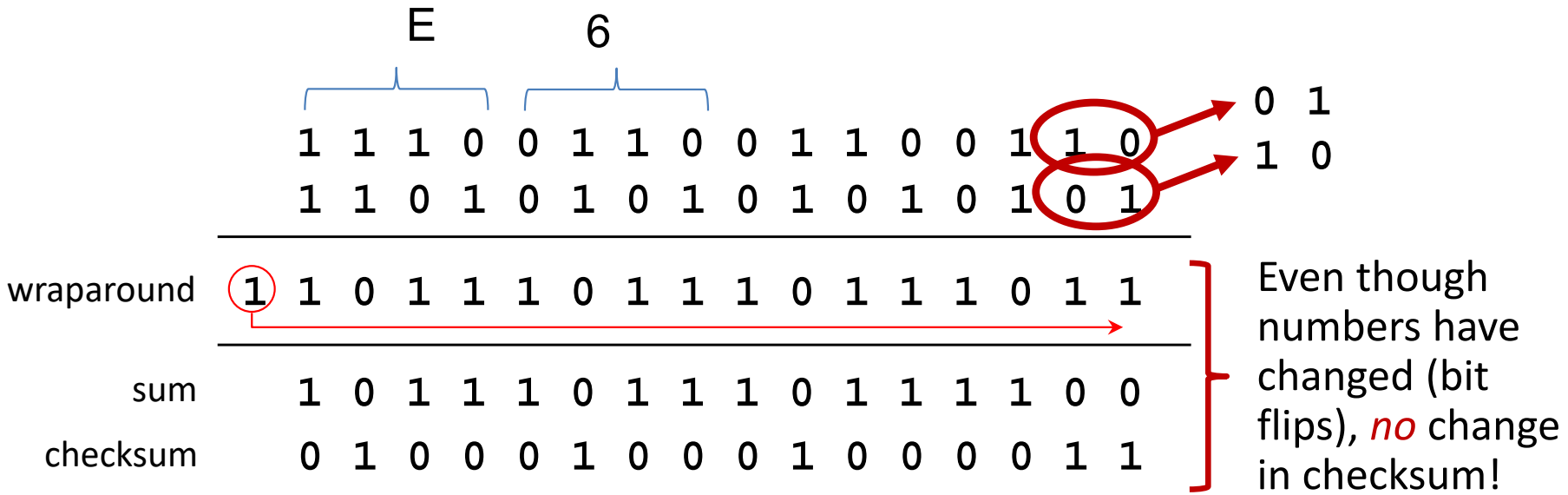
The new checksum, CBB0, is inserted in the checksum field

Wrapped F=E+1

Set 1 to 0 and 0 to 1

Weak Protection

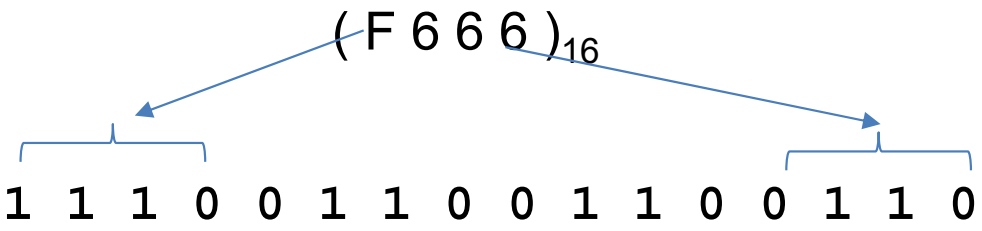
Example: add two 16-bit integers of two headers E666 and D555 for a **checksum** that is received incorrectly



Note: You can calculate in either hexadecimal or bits (see tutorial for more practice)

Hexadecimal ⟶ Bits ⟶ Decimal

Hexadecimal



Bits

Hexadecimal ⟶ Decimal $15x16^3 + 6x16^2 + 6x16^1 + 6x16^0$

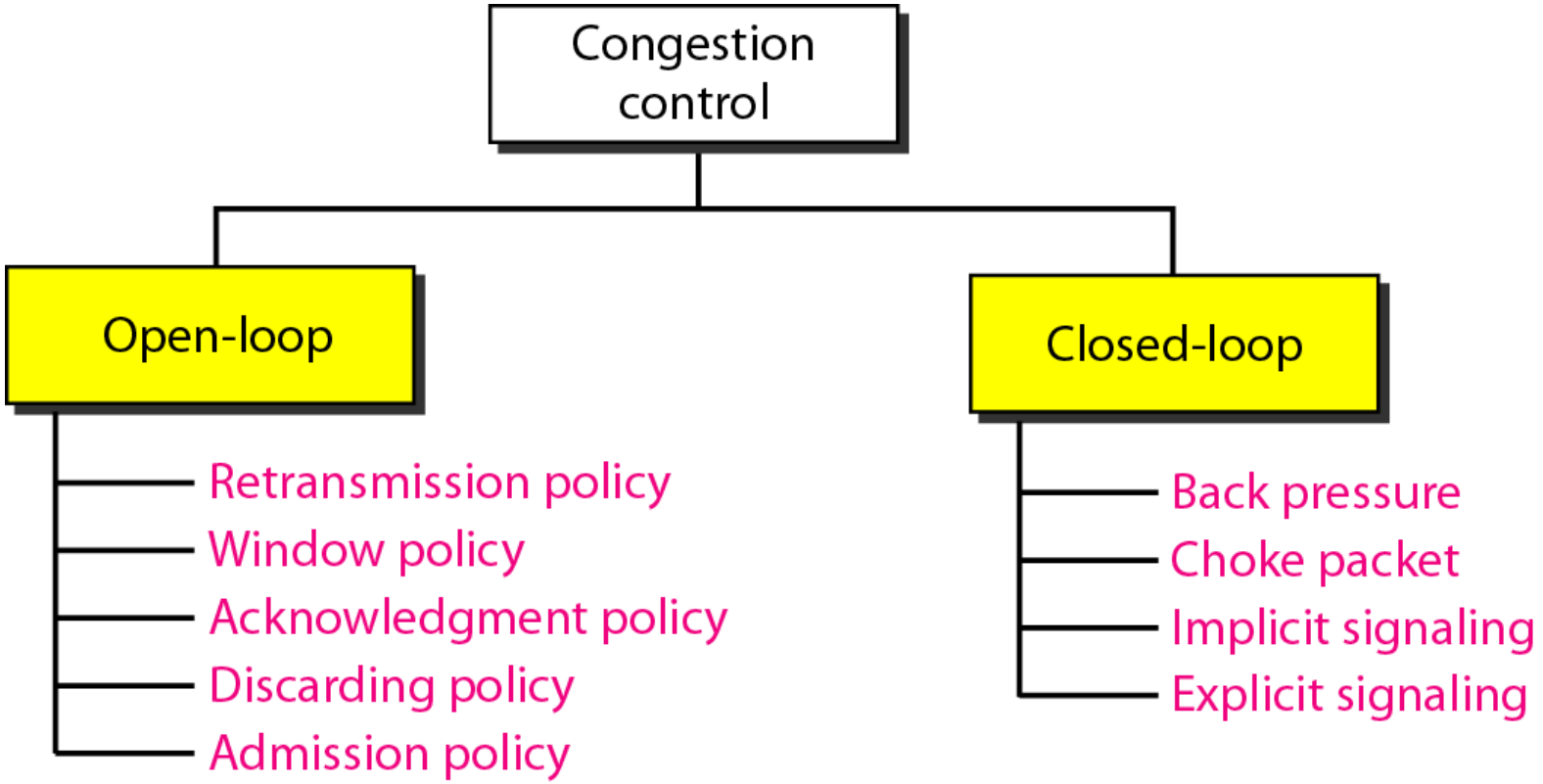
Bits ⟶ Decimal $1x2^{15} + 1x2^{14} + 1x2^{13} + 1x2^{10} + \ldots 1x2^1$

Congestion Control

- Flow control versus congestion control?
- Flow control guarantees that the receive window is never overflowed (no end congestion), but the intermediate buffers may still be overflowed (congestion).
- Congestion control – the mechanisms and techniques to control congestion and keep the load below the capacity
- Congestion window (*cwnd*): the value is determined by the congestion situation in the network
- *cwnd* and *rwnd* together define the size of send window

Actual window size= minimum (cwnd, rwnd)

Congestion Control Categories



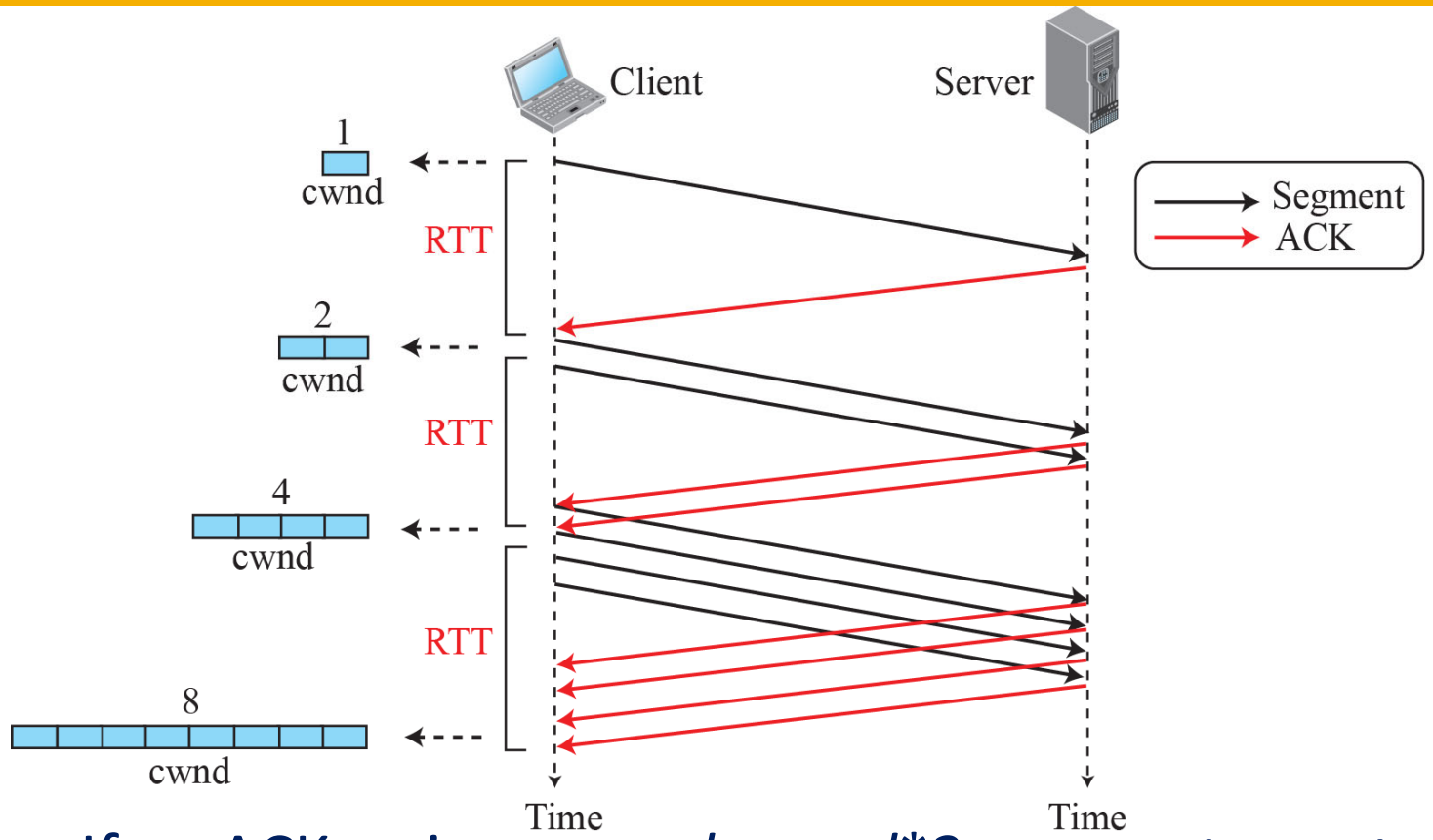
- **Open loop**: prevent congestion before it happens
- **Closed loop**: alleviate congestion after it happens

THE UNIVERSITY OF SYDNEY

Congestion Control in TCP

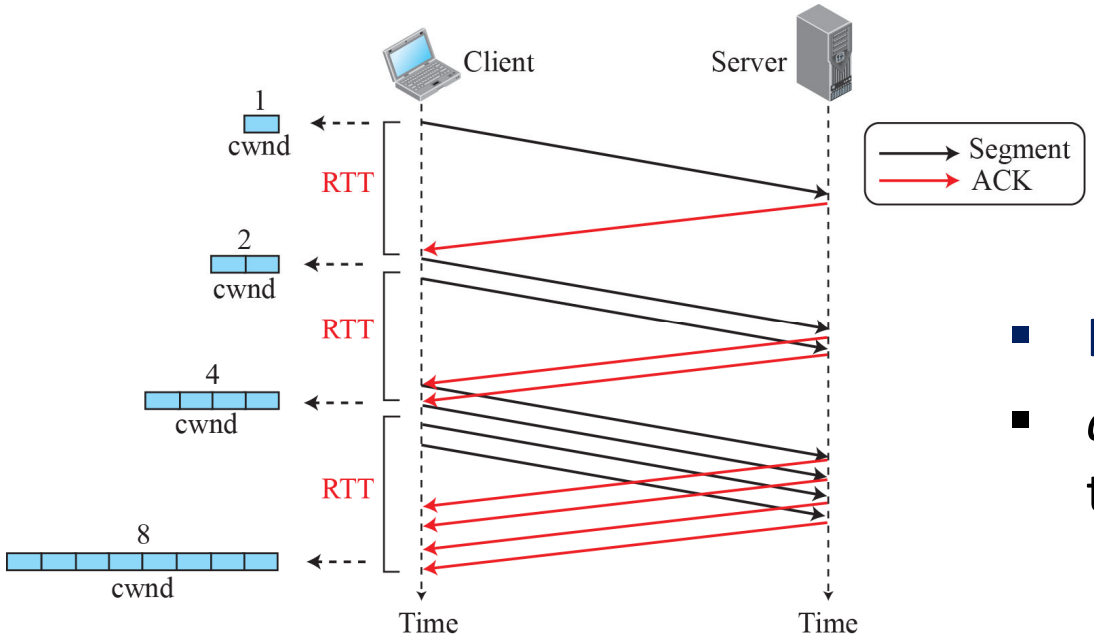
- TCP's general policy for handling congestion is based on three phases:
  - Slow start – The size of the congestion window *(cwnd)* starts with one maximum segment size (MSS) or any other number, but it increases by one MSS each time an ACK arrives until the *cwnd* in bytes reaches the slow-start threshold (*ssthresh*) → increasing data rate
  - Congestion avoidance – Next the size of the *cwnd* increases additively until congestion is detected.
  - Congestion detection – Sender goes back to Slow start phase or Congestion avoidance phase based on how the congestion is detected.

Slow start, exponential increase



- If an ACK arrives, *cwnd=cwnd**2 segments are transmitted
- Cwnd increases when ACK for all segments are received
- *cwnd* increases exponentially in terms of round-trip times (RTT)
- Increase exponentially until threshold

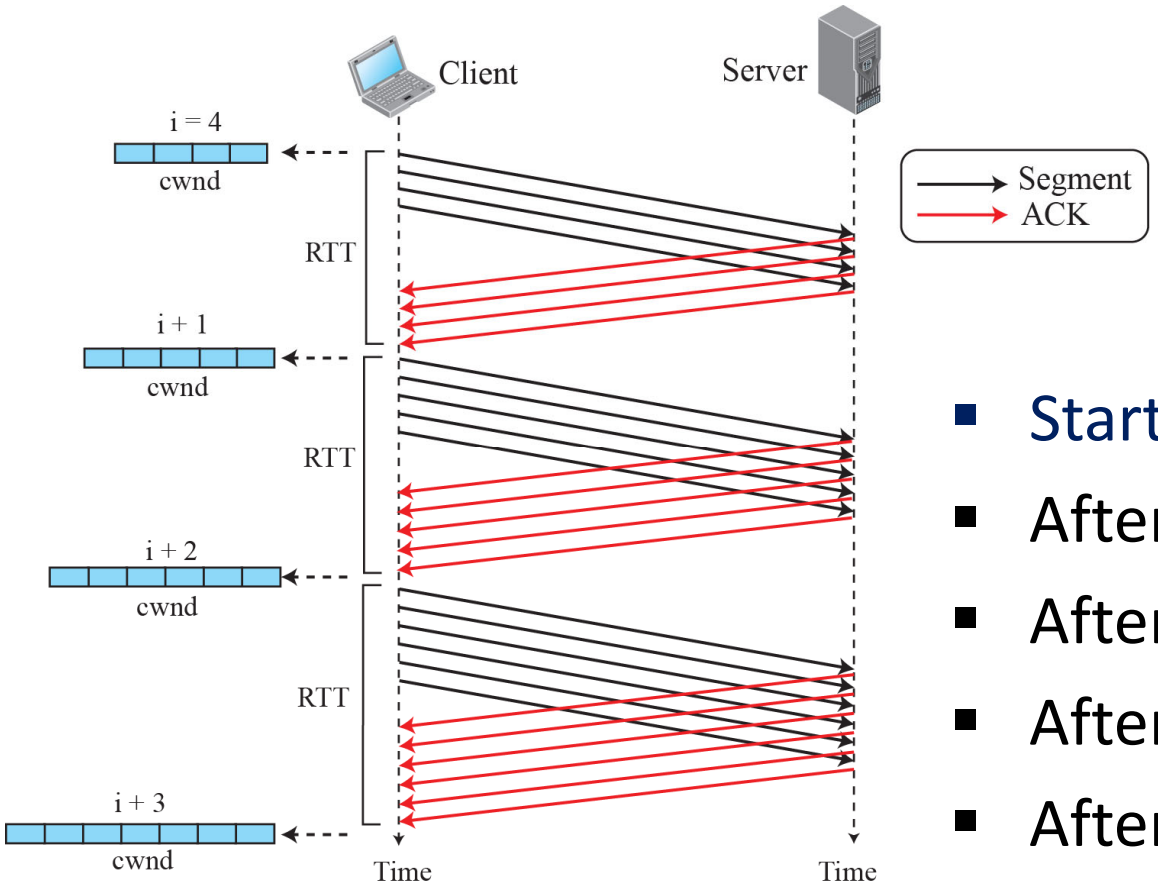
Slow start, exponential increase



- If an ACK arrives, *cwnd=cwnd*2*
- *cwnd* increases exponentially in terms of round-trip times (RTT)

- Start: *cwnd*=1→ 2^0
- After 1 RTT: *cwnd=cwnd*+1=2→ 2^1
- After 2 RTT: *cwnd=cwnd*+2=4→ 2^2
- After 3 RTT: *cwnd=cwnd*+4=8→ 2^3
- After n RTT: *cwnd=cwnd*+ 2^{n-1} → 2^n

Cong. avoidance, additive increase



- Start: *cwnd=i*
- After 1 RTT: *cwnd=i*+1
- After 2 RTT: *cwnd=i*+2
- After 3 RTT: *cwnd=i*+3
- After n RTT: *cwnd=i*+n

- If an ACK arrives, *cwnd=cwnd*+ n where n is as defined before
- *cwnd* increases linearly in terms of round-trip times (RTT)

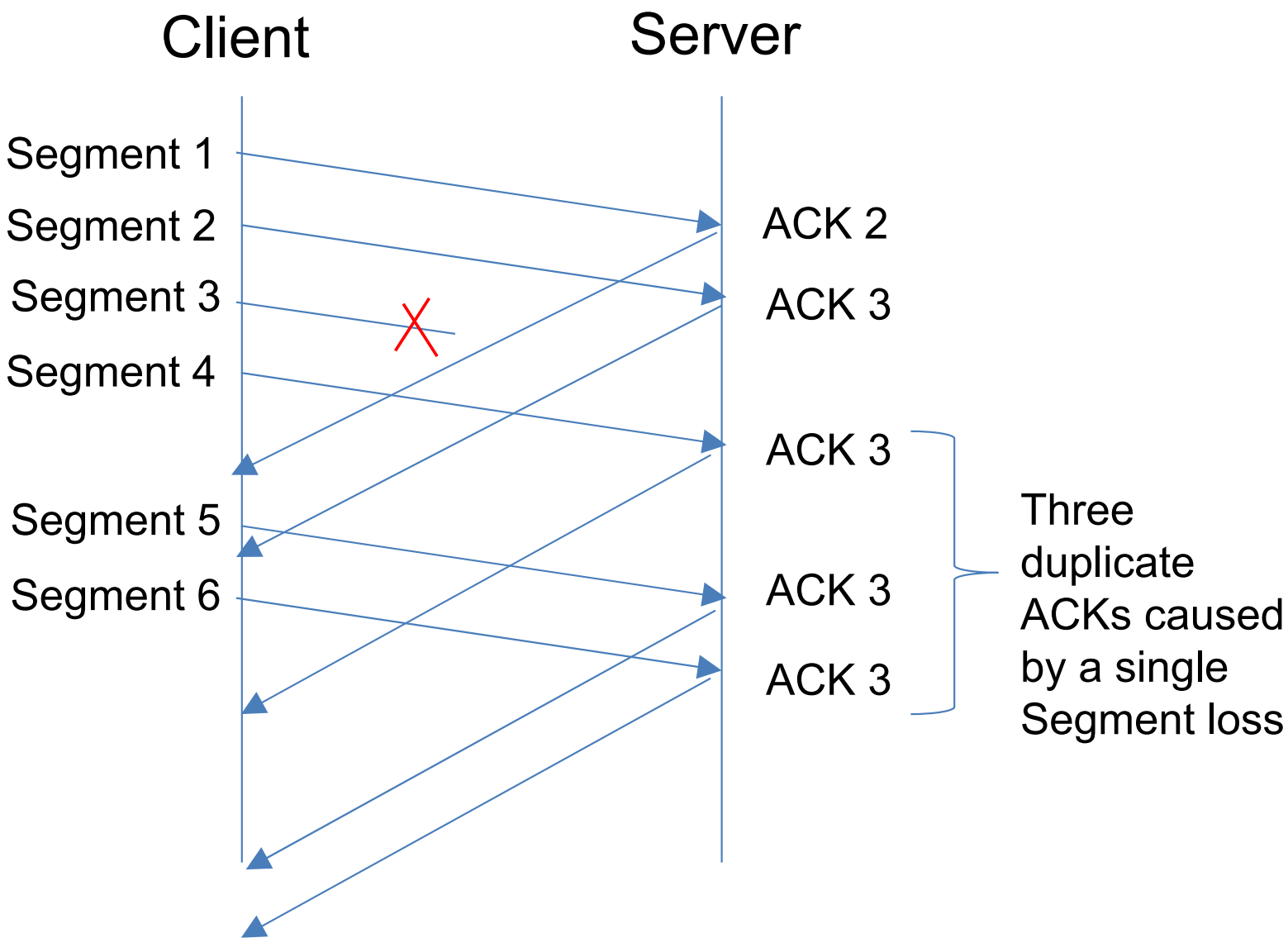
Congestion detections

- **Congestion detected by time-out (no ACKs received): Stronger possibility of a congestion and TCP acts strongly**
  - Sets the threshold to one-half the current window size
  - Sets the *cwnd* to 1 MSS (or any number if start value is not 1)
  - Starts the slow-start phase again

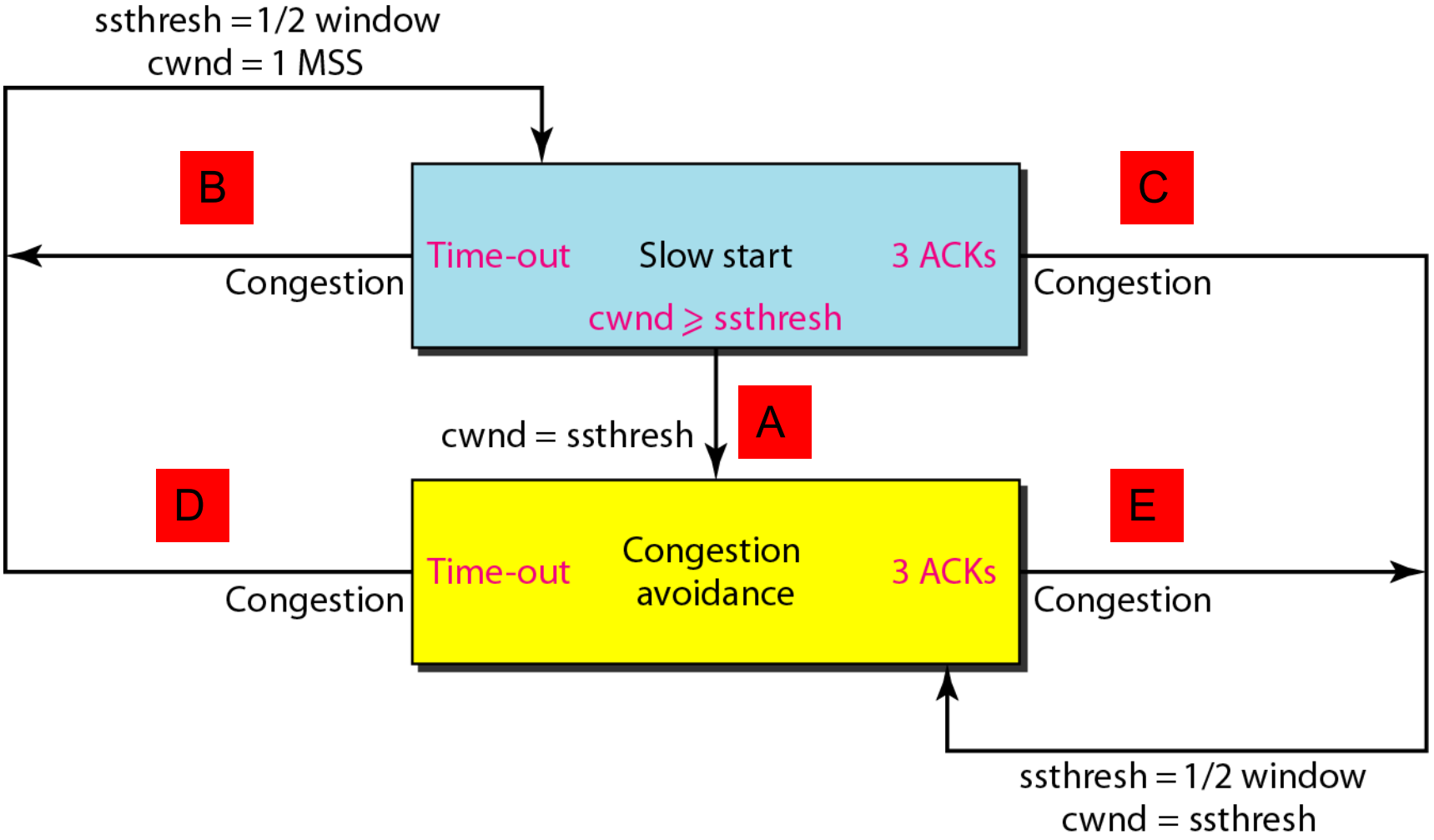
- **Congestion detected by three duplicate ACKs (some ACK is missing): Weaker possibility of a congestion and TCP acts weakly**
  - Sets the threshold to one-half the current window size
  - Sets the *cwnd* to the value of the threshold
  - Starts congestion avoidance phase

Recall ACK informed sender the next expected sequence number

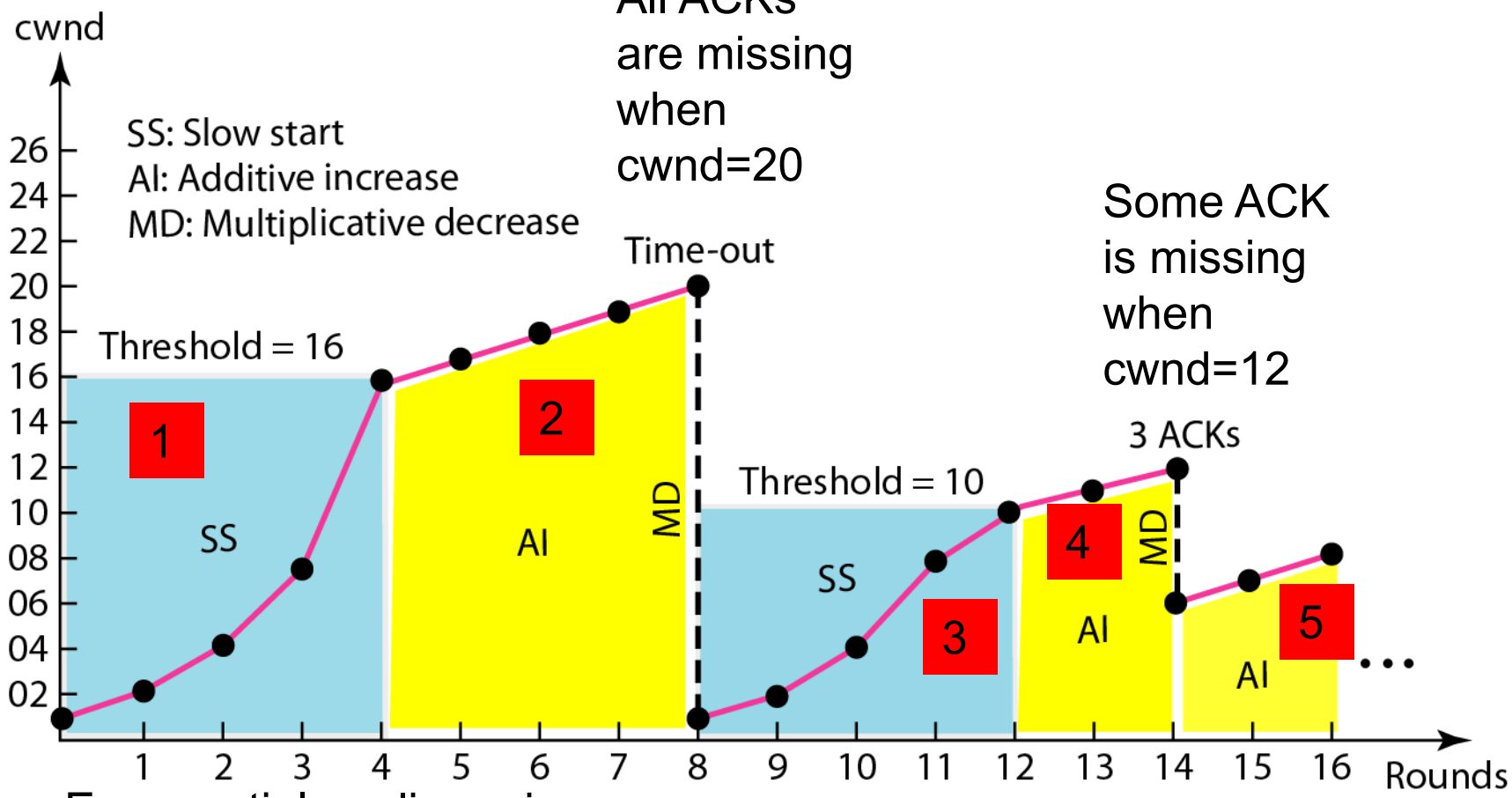
Three Duplicate ACKs



TCP Congestion Policy Logic Summary



Congestion Control Example



Exponential increase (slow start window)

linear increase (congestion avoidance window)

No ACK is received ssthresh is 20/2

congestion avoidance window with sshtresh=12/2

Recommended Reading

- Behrouz A. Forouzan, Data Communications and Networking with TCP/IP Protocol Suite, 6th ed., 2022, Chapters 9

- J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach, 8th ed., 2022, Chapter 3