CS3103: Computer Networks Practice

TCP Congestion Control

- TCP Revision (For Self Learning)- Upto slide 18
- Congestion Control

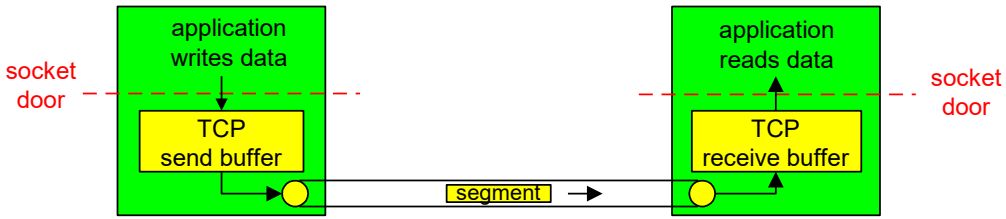
Note: Read slides 1 to 18. And discuss if you have any questions during the Lecture. We will run through these contents quickly during the Lecture.

Dr. Anand Bhojan
COM3-02-49, School of Computing
banand@comp.nus.edu.sg ph: 651-67351

TCP: Overview RFCs: 793,1122, 2018, 5681, 7323, RFC 9293 (Aug 2022)

Revision *Self-Learning*

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- Cumulative ACKs
- send & receive buffers

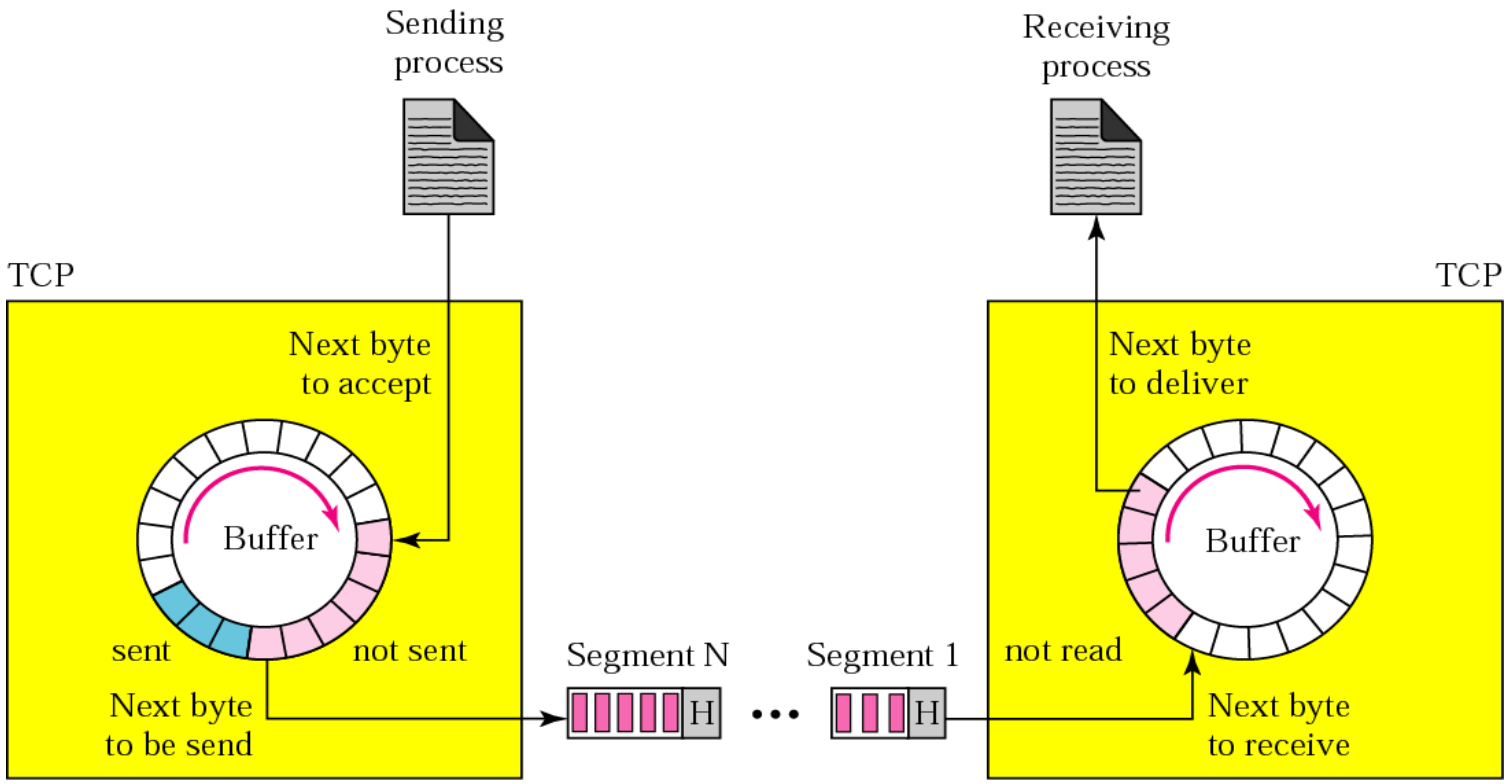


- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

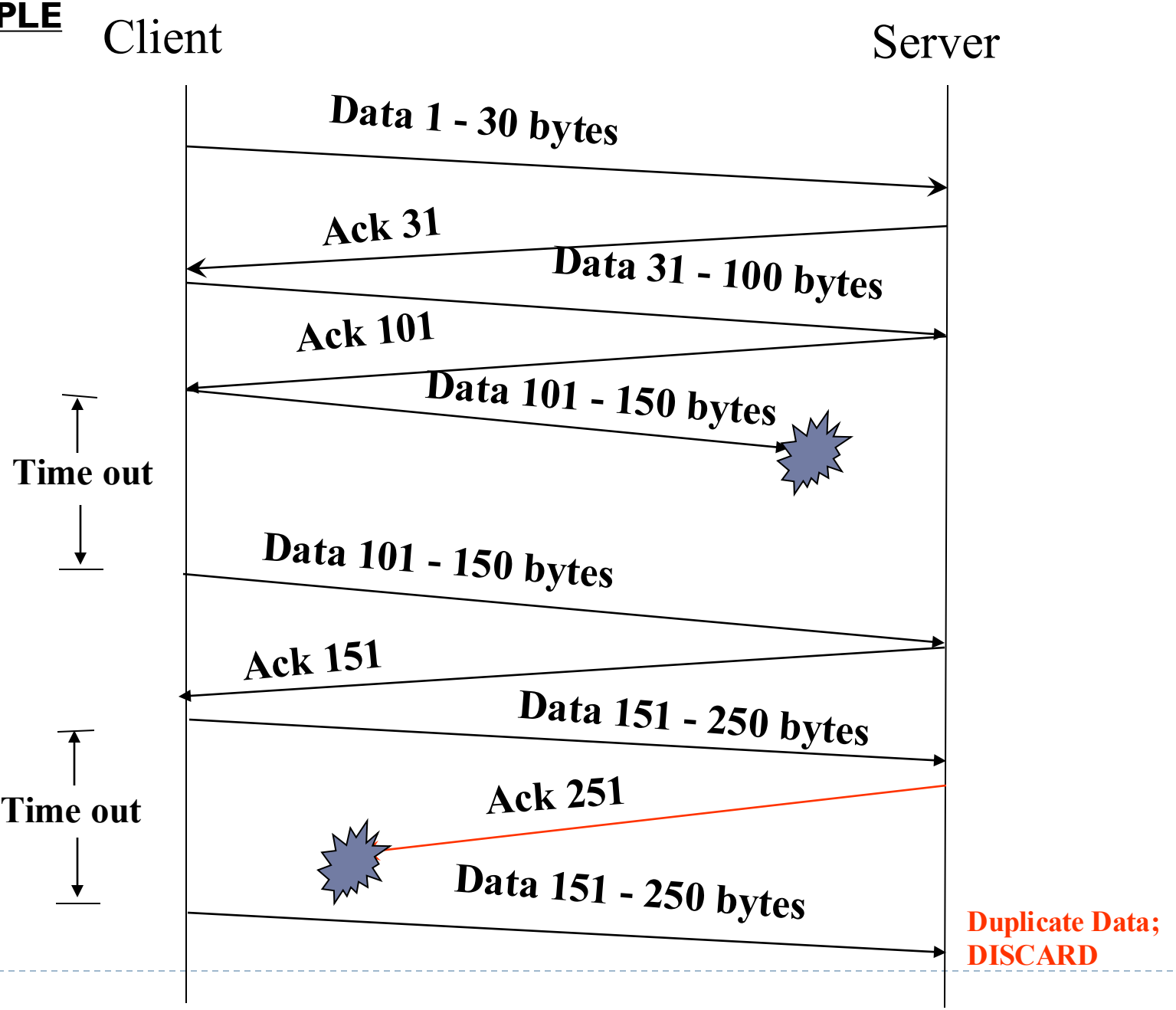
Buffers, Bytes & Segment

Revision Self-Learning

TCP is a byte-stream-oriented protocol

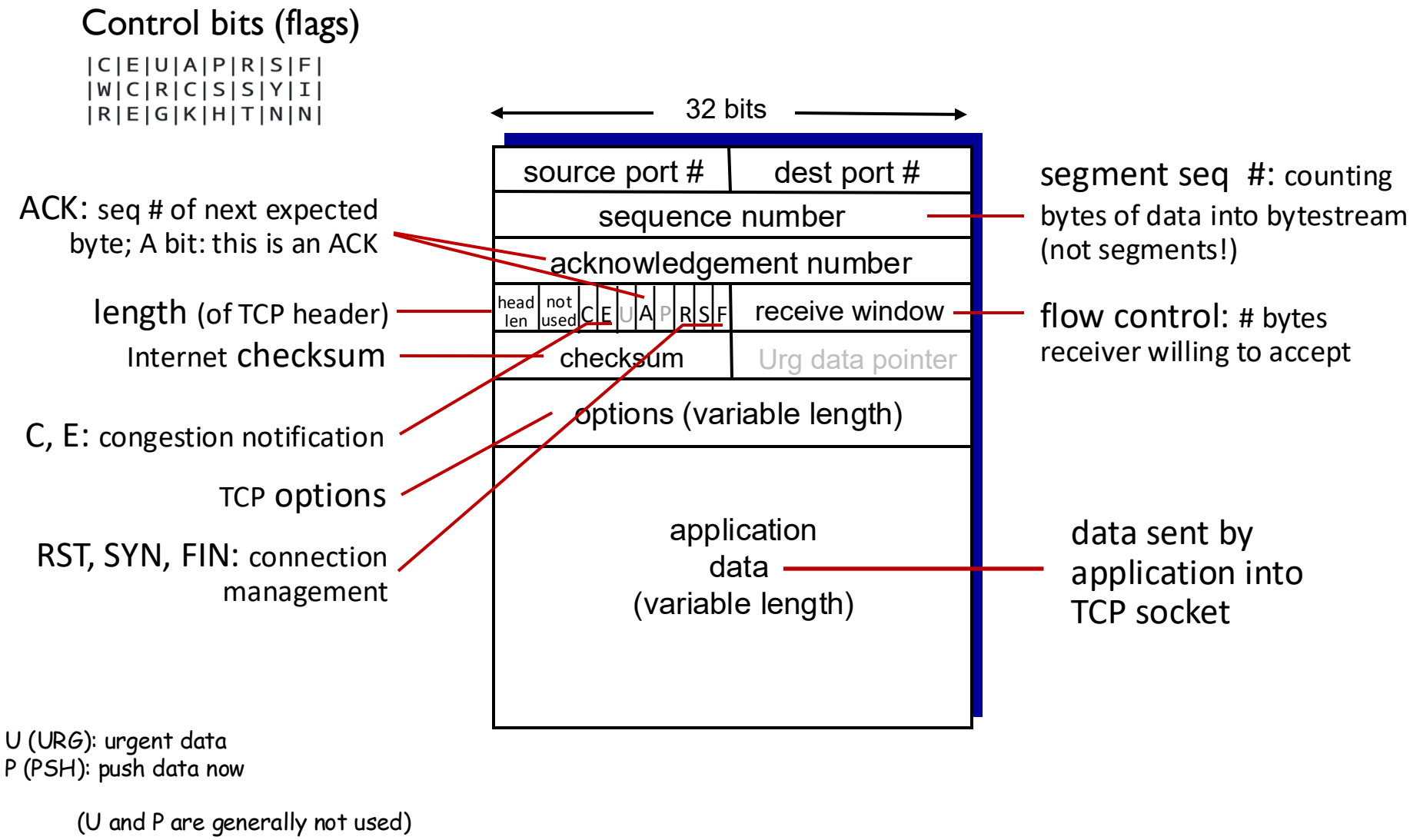


EXAMPLE


Client
Server
Data 1 - 30 bytes
Ack 31
Data 31 - 100 bytes
Ack 101
Data 101 - 150 bytes
Time out
Data 101 - 150 bytes
Ack 151
Data 151 - 250 bytes
Time out
Ack 251
Data 151 - 250 bytes
Duplicate Data;
DISCARD

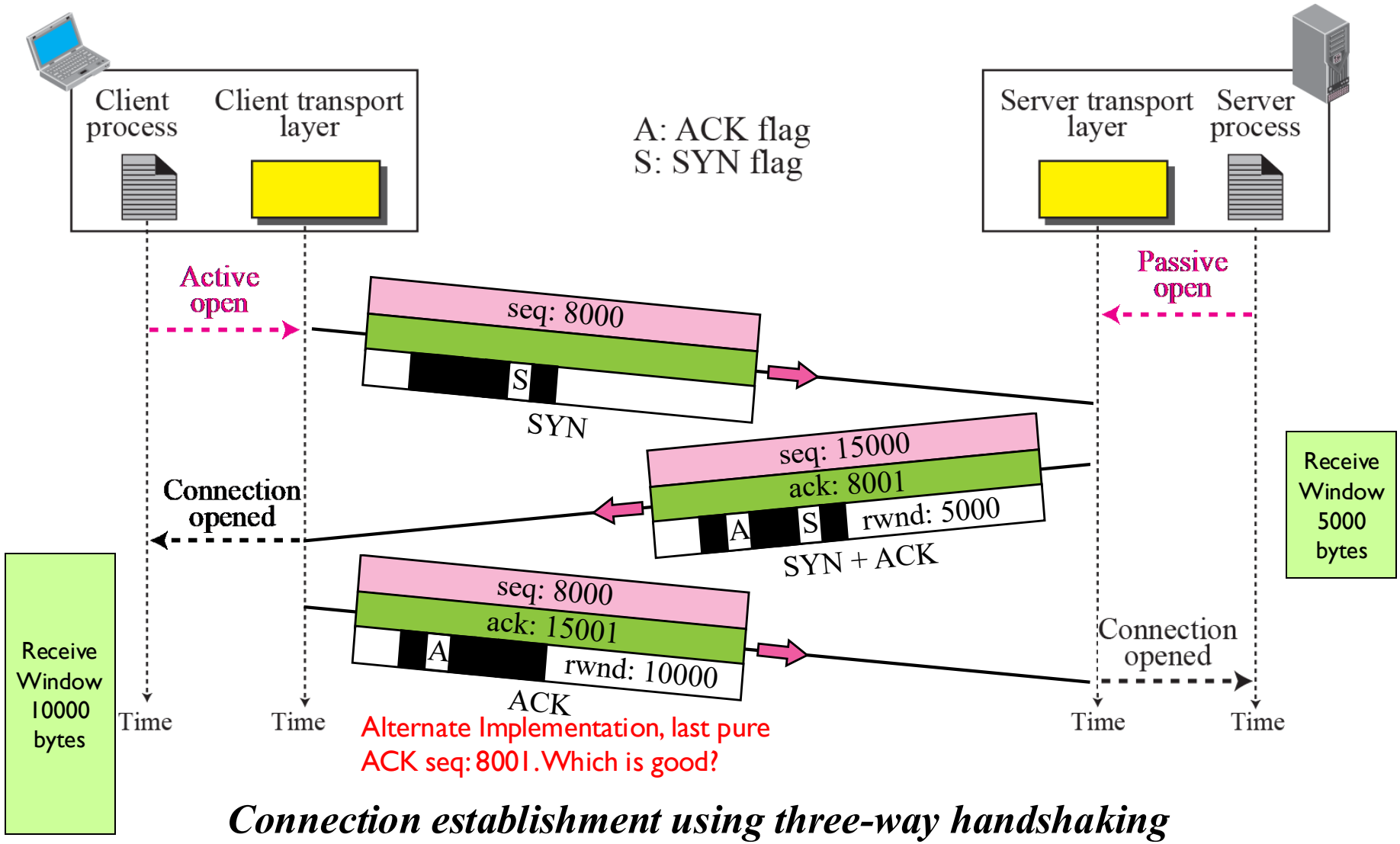
TCP segment structure

Revision Self-Learning



TCP Connection Establishment

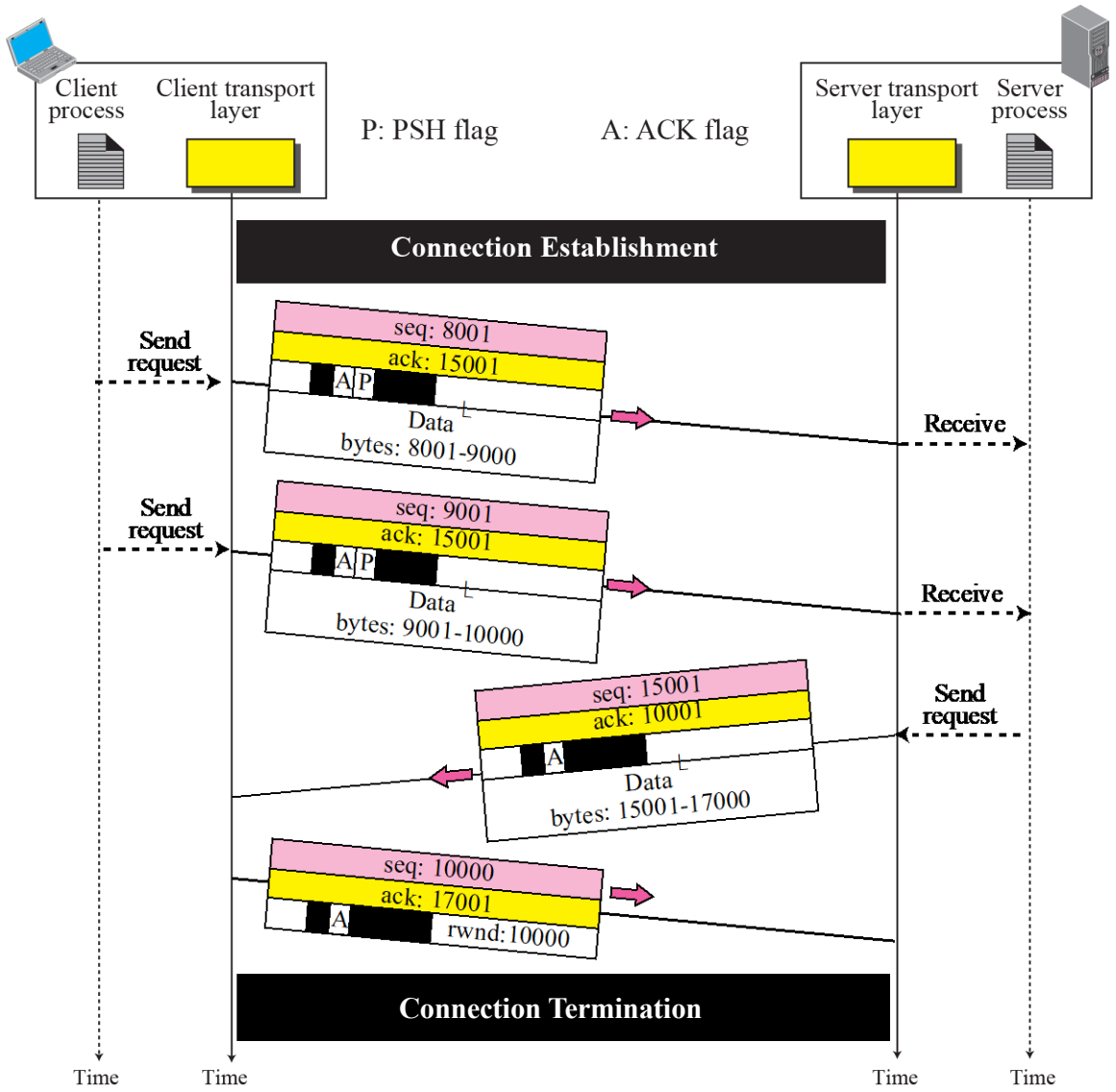
Revision Self-Learning



Connection establishment using three-way handshaking

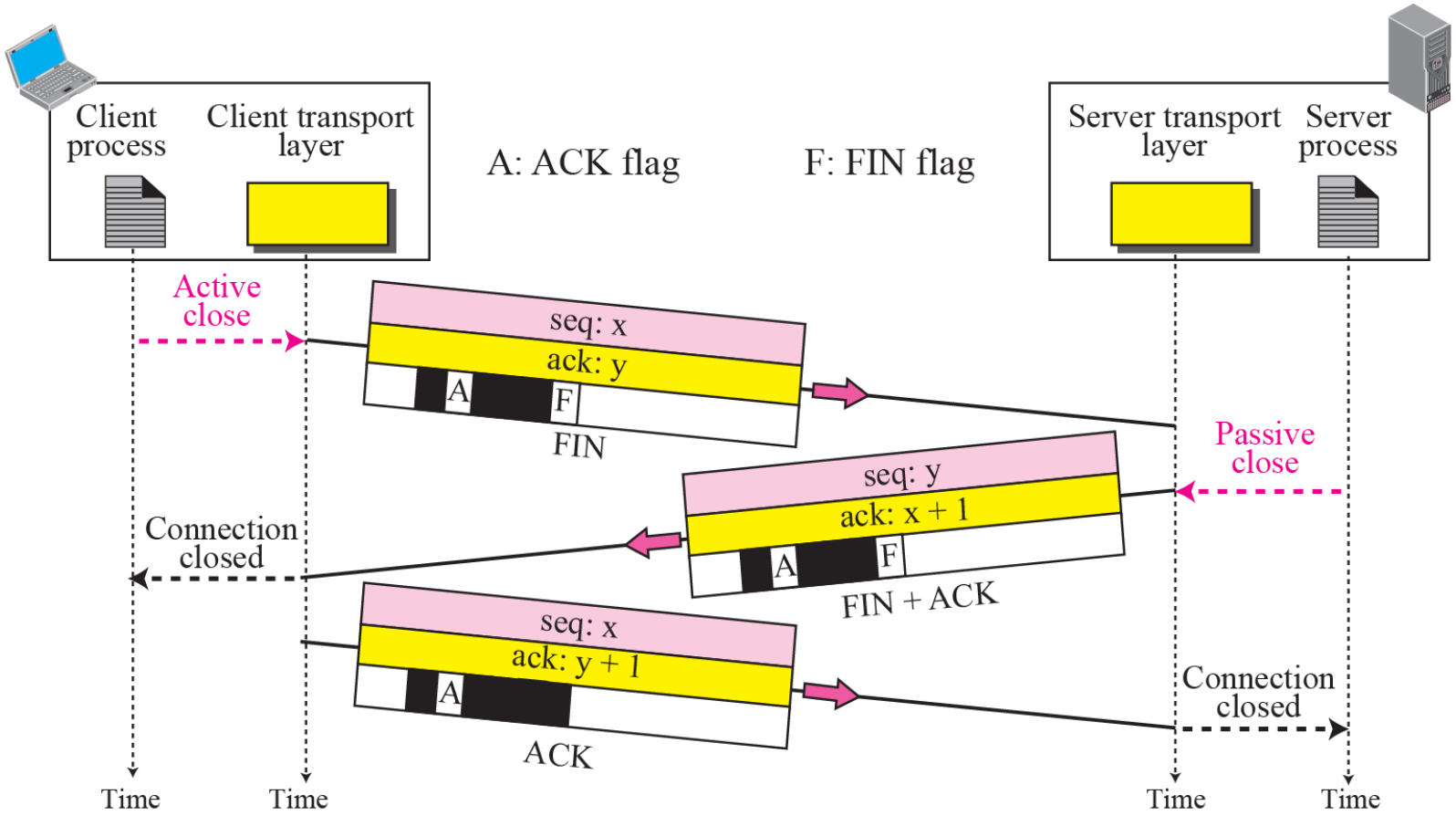
TCP Data Transfer

Revision Self-Learning



TCP Connection Termination

Revision Self-Learning

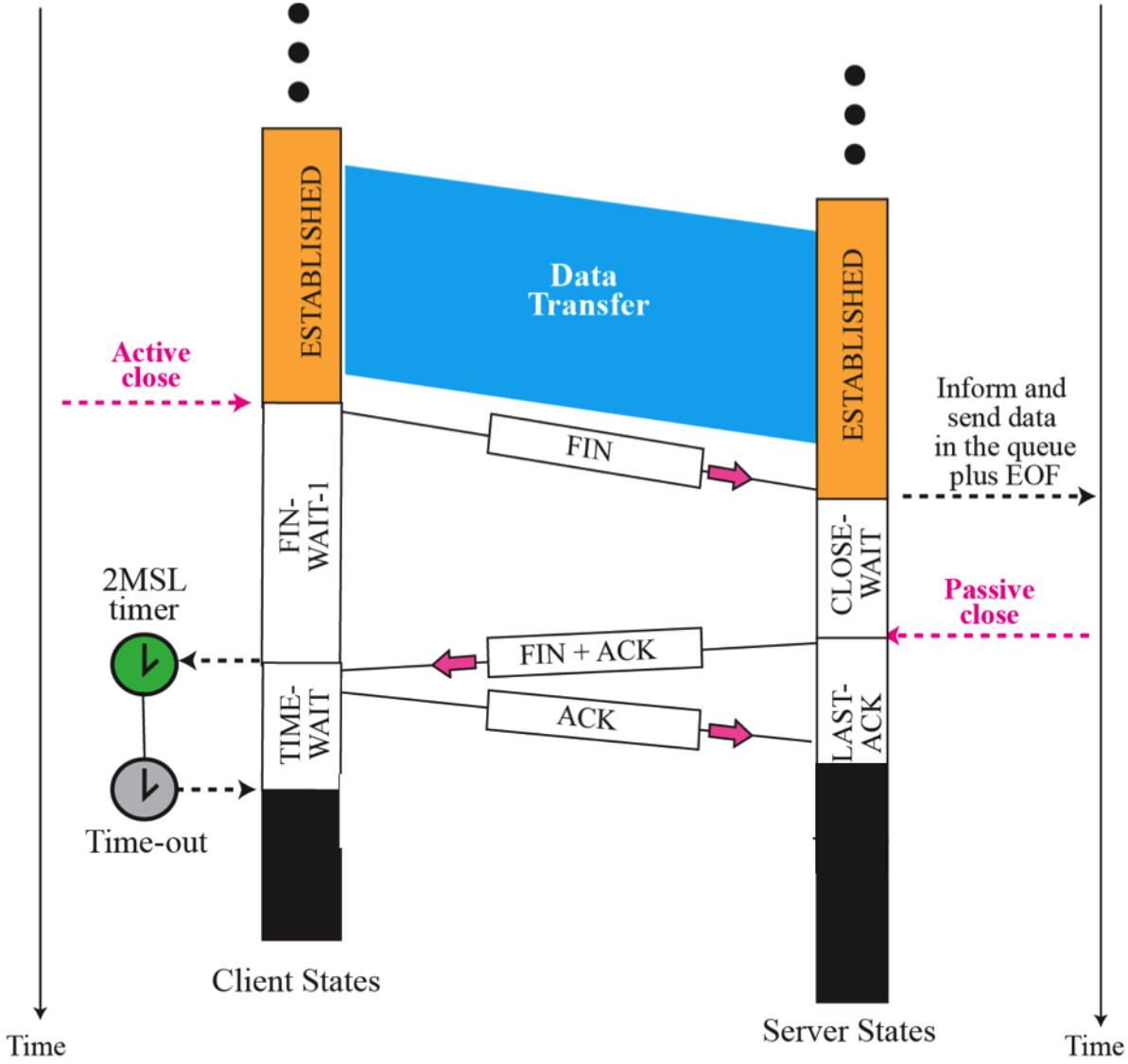


- Each side must independently close its half of the connection
- FIN indicates no more data from that direction.

TCP Connection Termination – 2MSL Timer

(MSL - Maximum Segment Lifetime)

Self-Learning



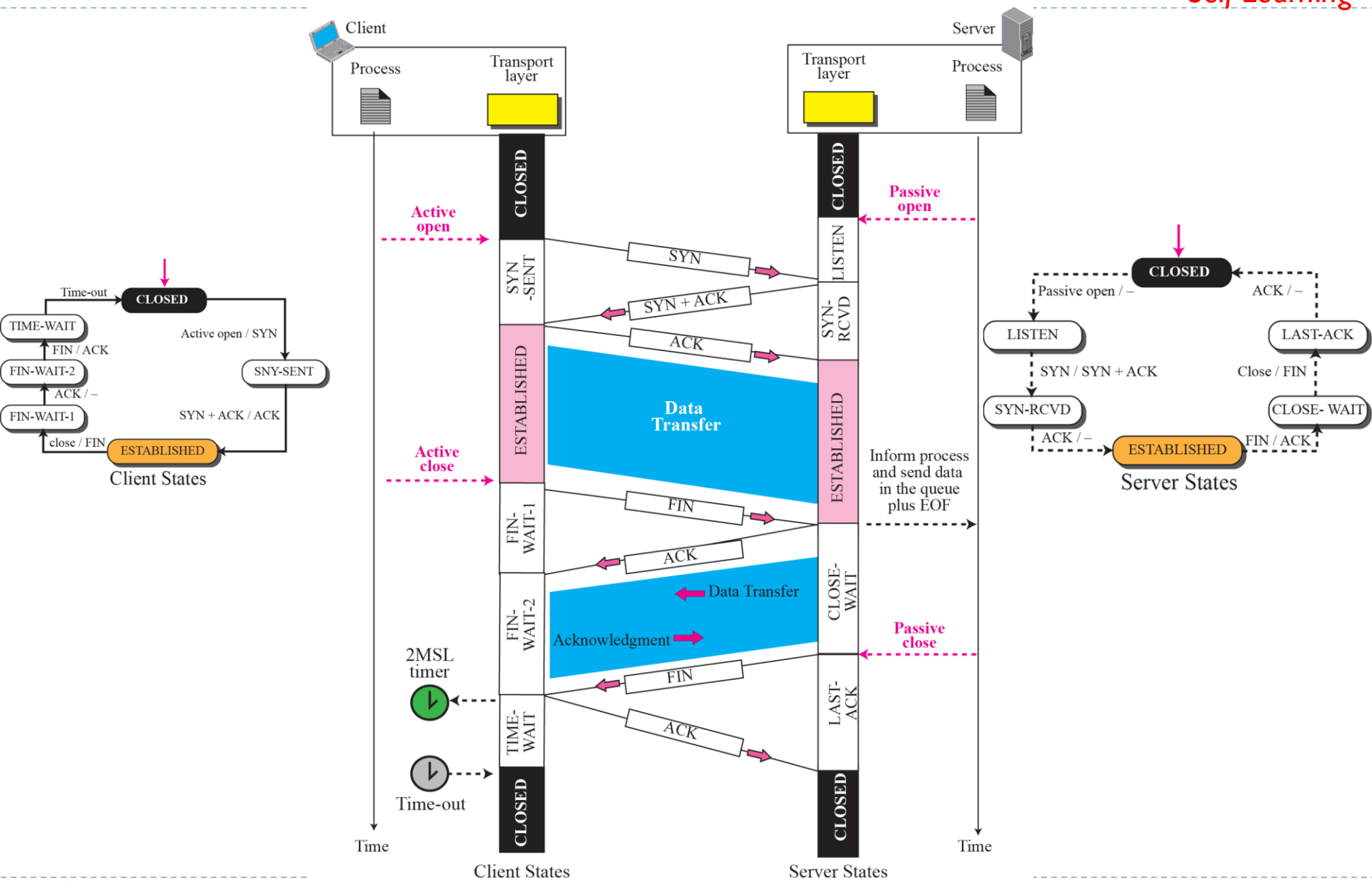
Each side must independently close its half of the connection

FIN indicates no more data from that direction

The common value for MSL is between 30 seconds and 1 minute

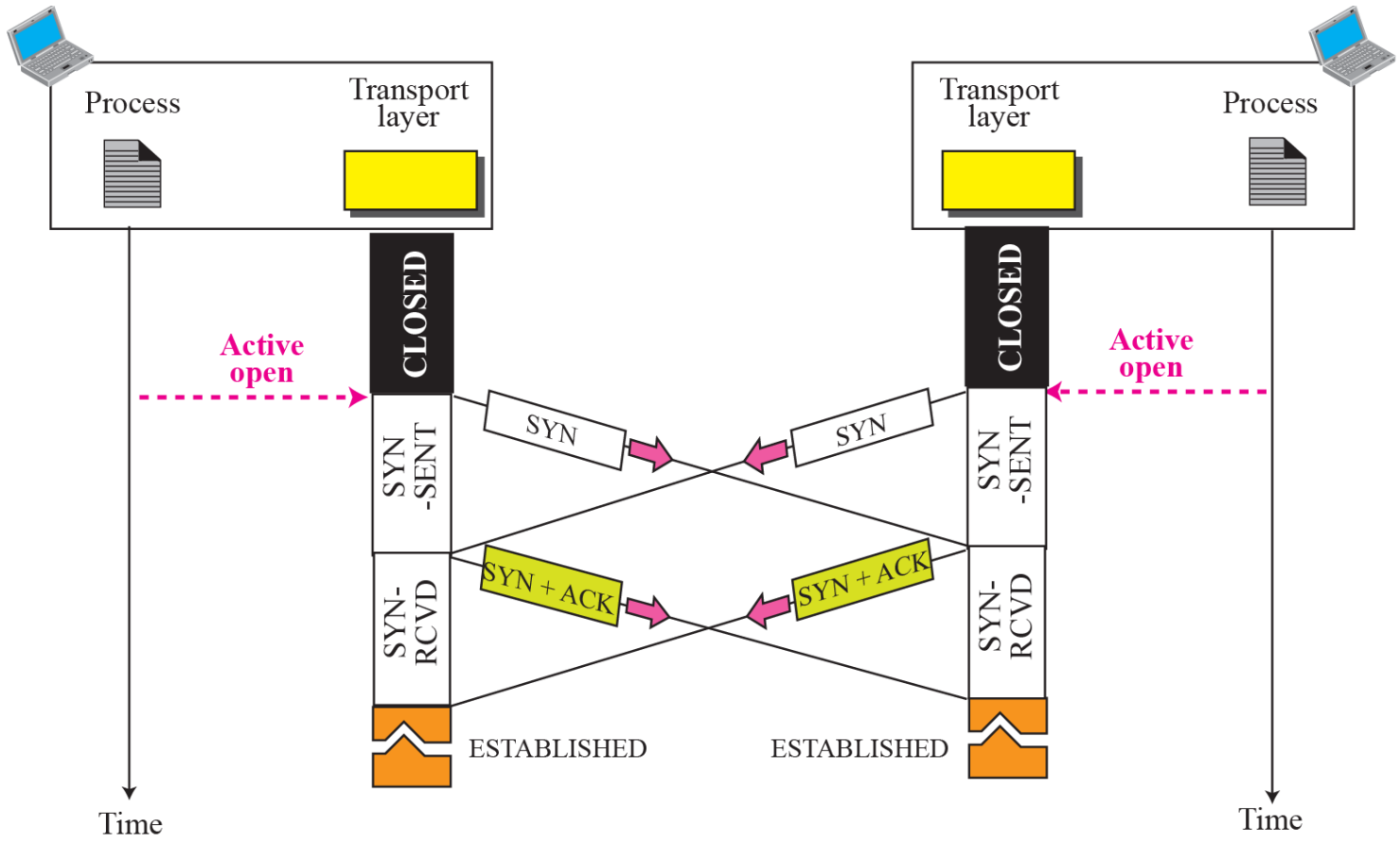
TCP Half-Close

Self-Learning



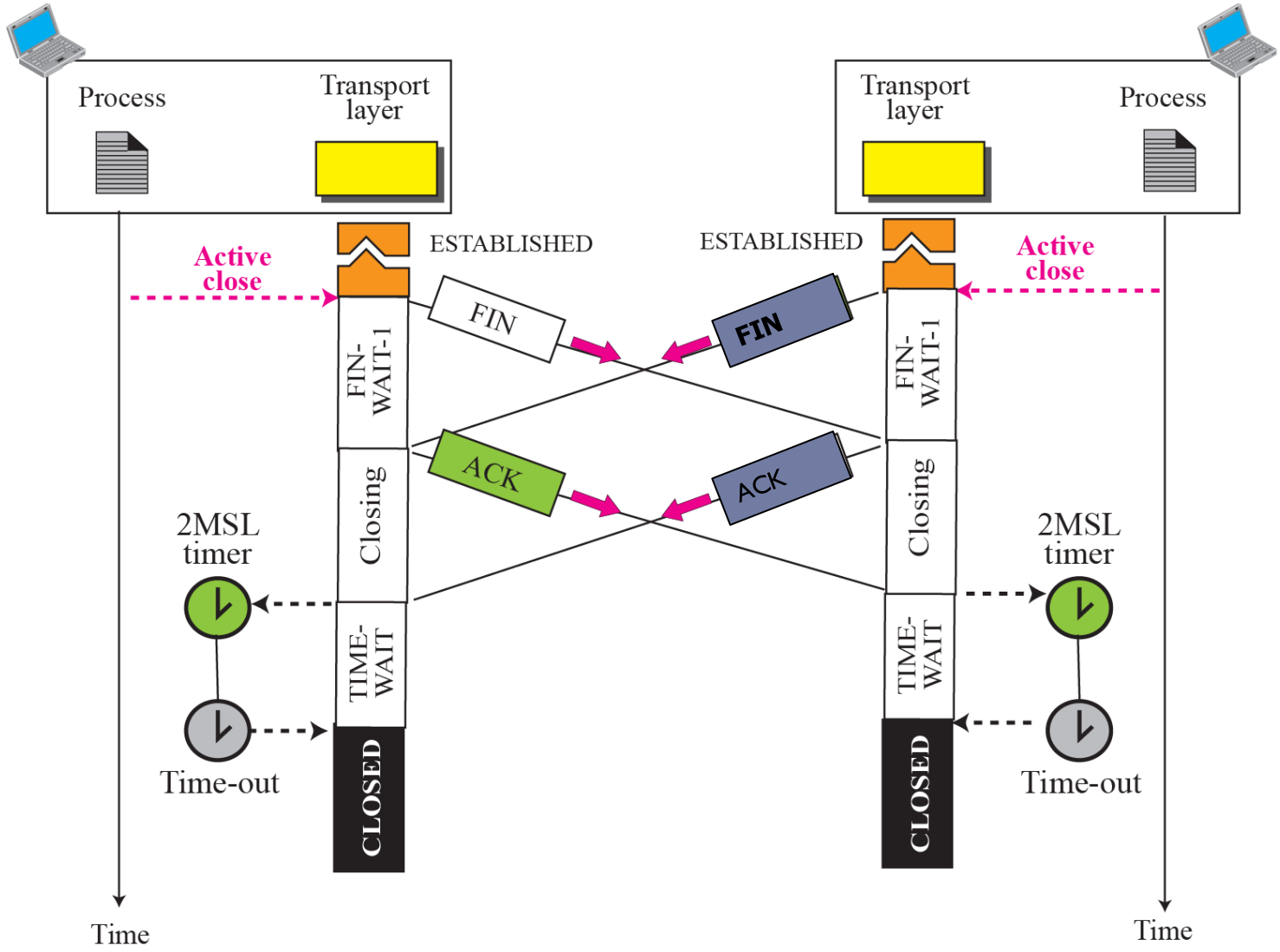
TCP Simultaneous Open

For Your Reference



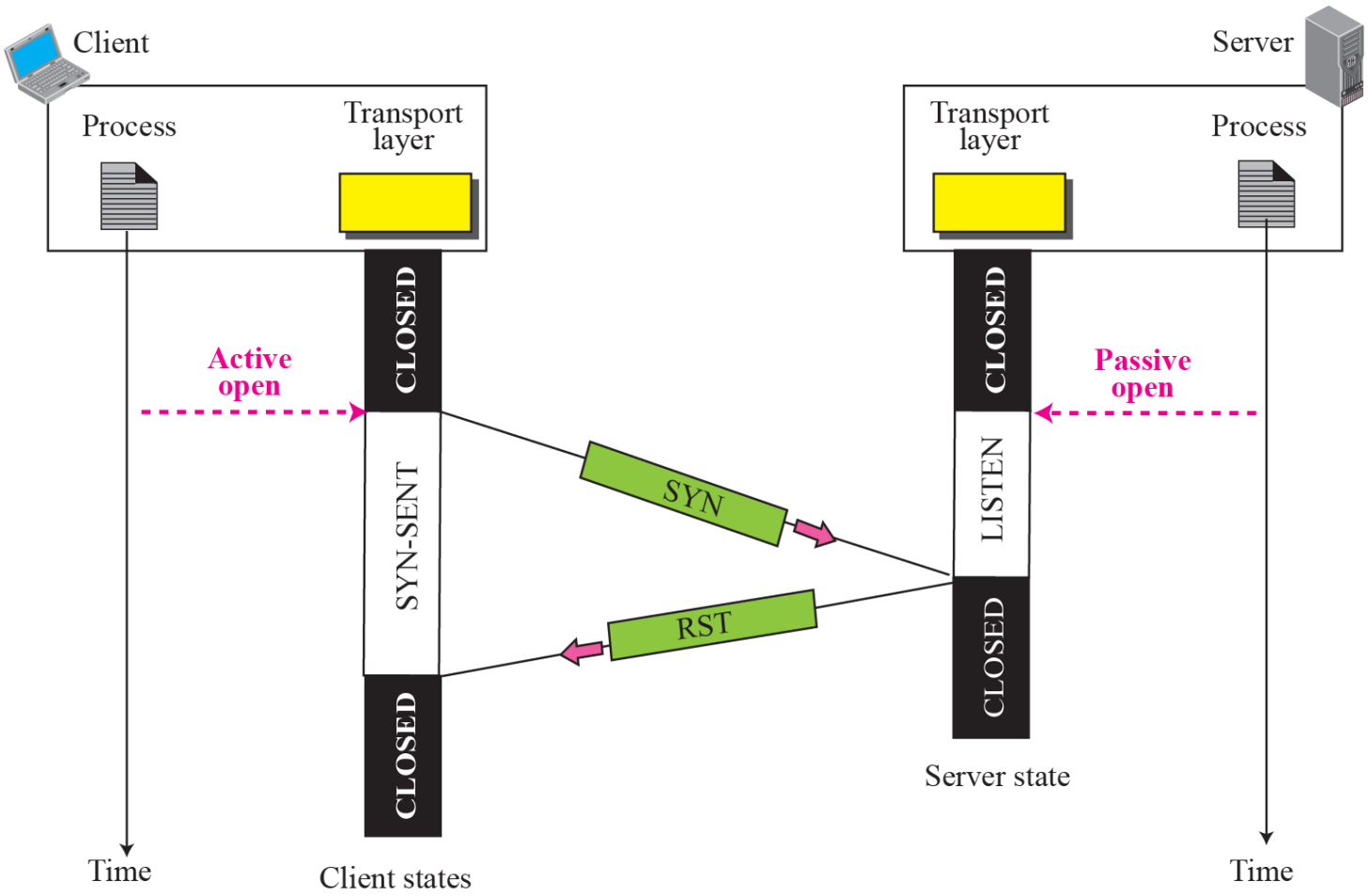
Simultaneous Close

For Your Reference



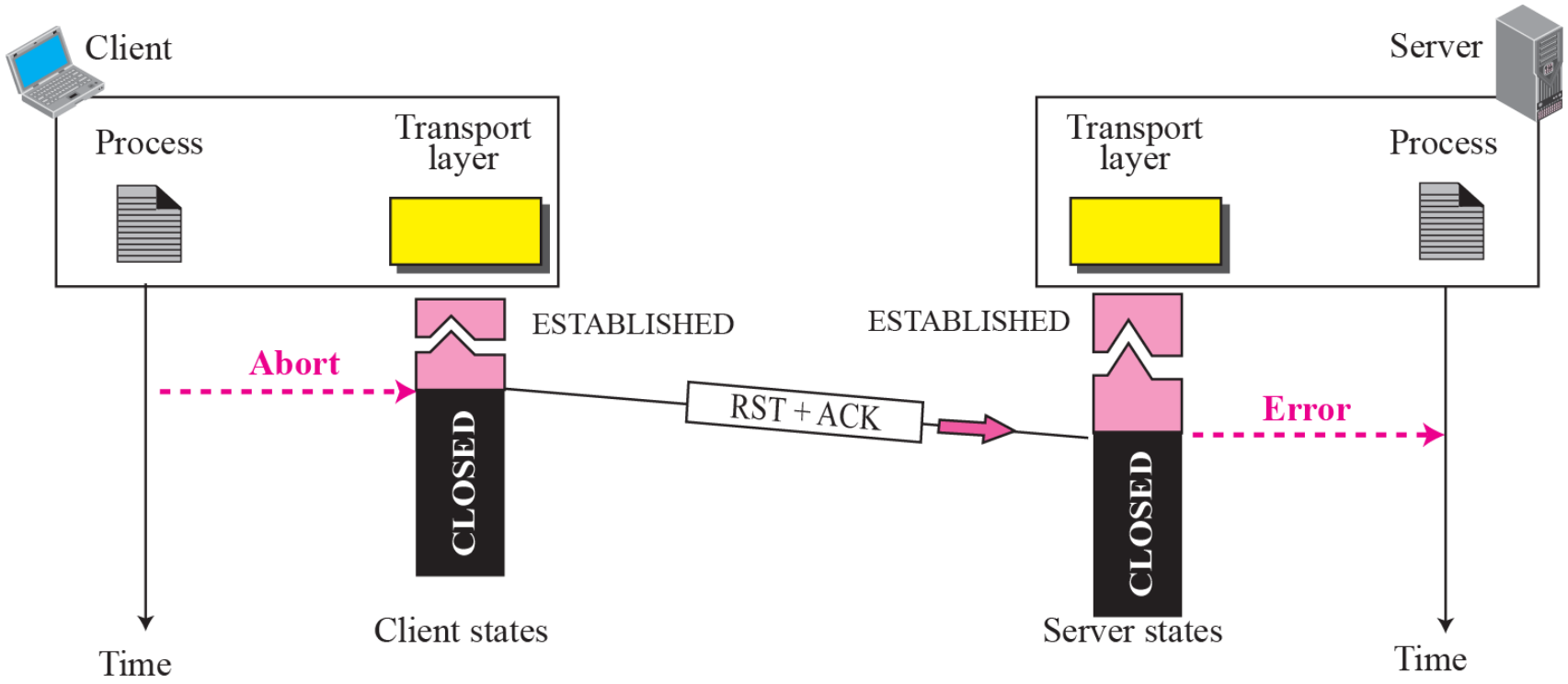
TCP Denying a connection

For Your Reference



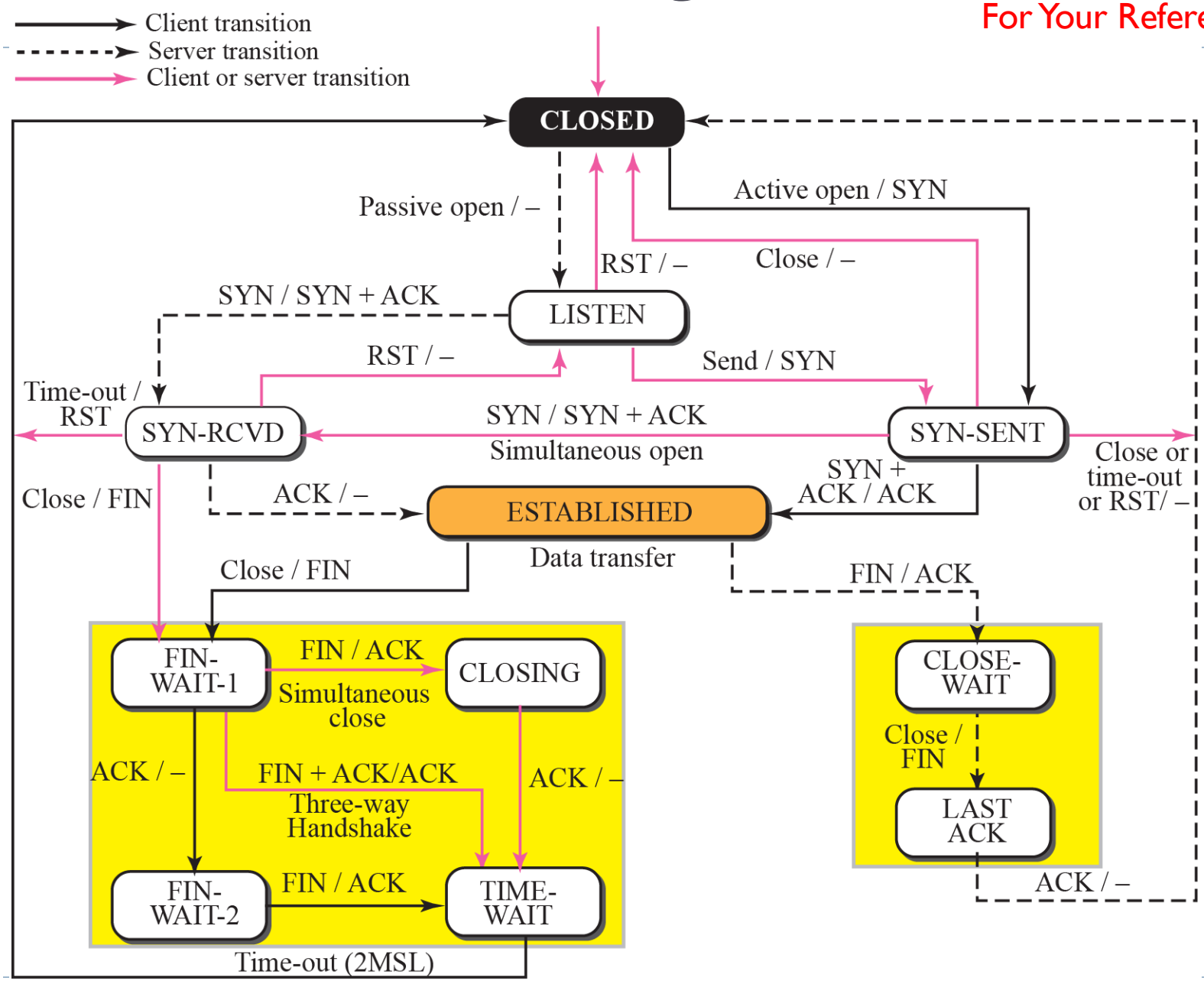
TCP Aborting a connection

For Your Reference



TCP - State transition diagram

For Your Reference

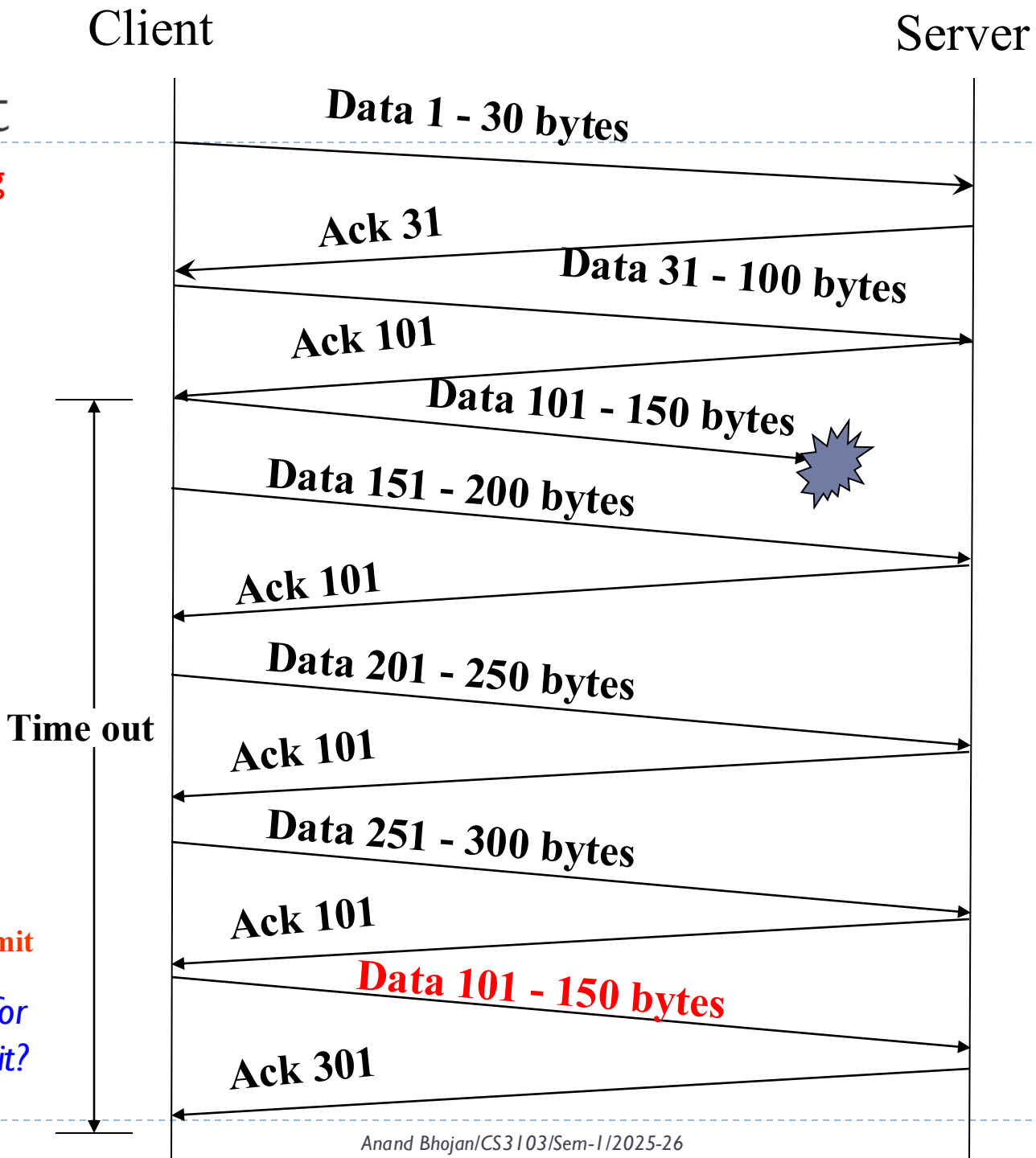


Fast Retransmit

Revision Self-Learning

Sender:Retransmit a packet on receiving 3 Dup-ACKs

Receiver: When a packet arrives out-of-order, receiver sends a duplicate ACK



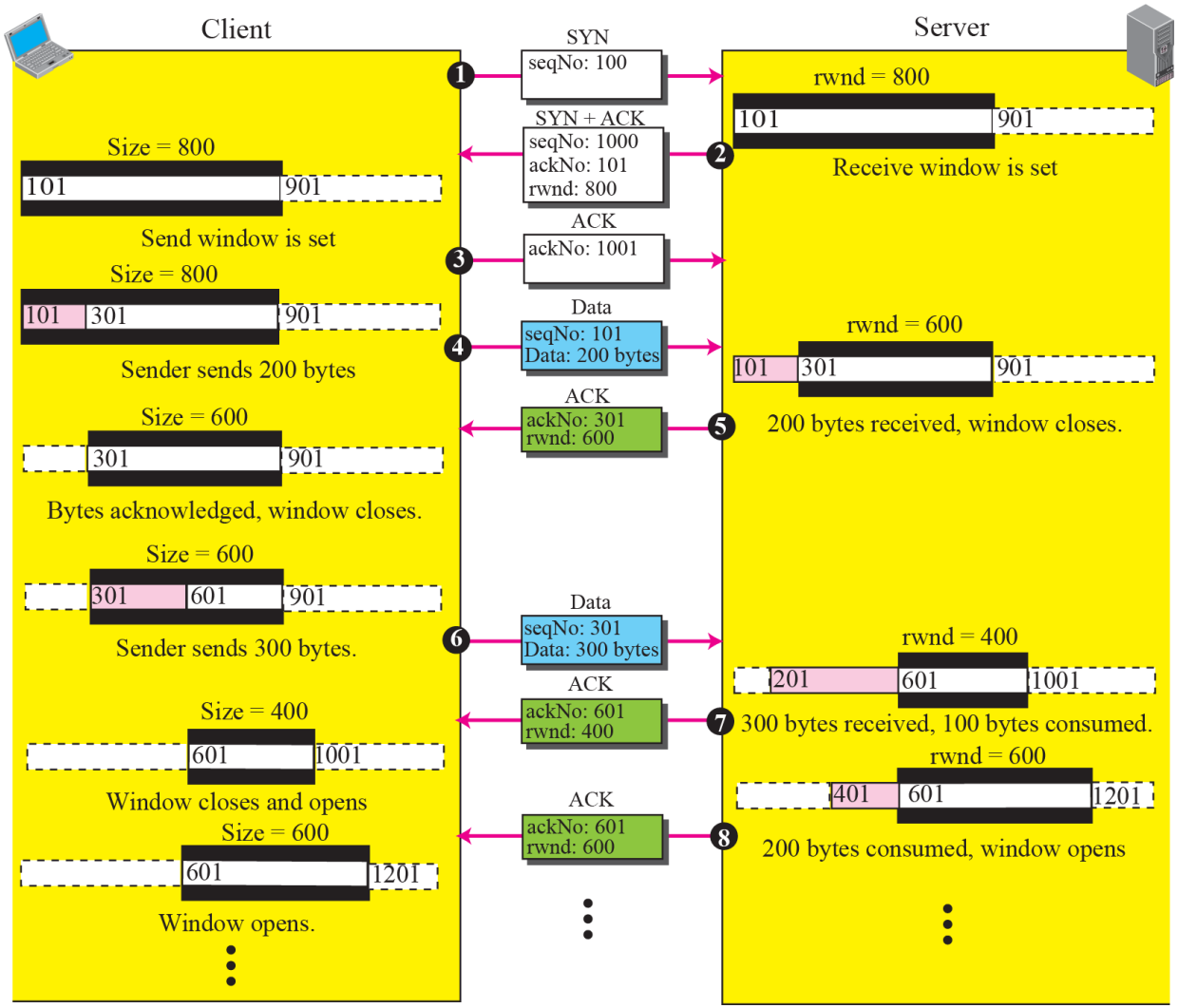
Fast Retransmit

Q:What is the rationale for introducing fast retransmit?

An example of flow control

Revision Self-Learning

SIMILAR TO WHAT IS GG BE TESTED



Window opens based on advertised rwnd size

Window opens as application consumes data

Quiz/Attendance

https://pollev.com/banand

Make sure you LOGIN using your NUSNET ID.

CS3103: Computer Networks Practice

TCP Congestion Control

TCP Basics Revision (FYORP)

Congestion Control

Dr. Anand Bhojan
COM3-02-49, School of Computing
banand@comp.nus.edu.sg ph: 651-67351

Principles of Congestion Control

Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- Q: What are the manifestations/effects of congestion?

Principles of Congestion Control

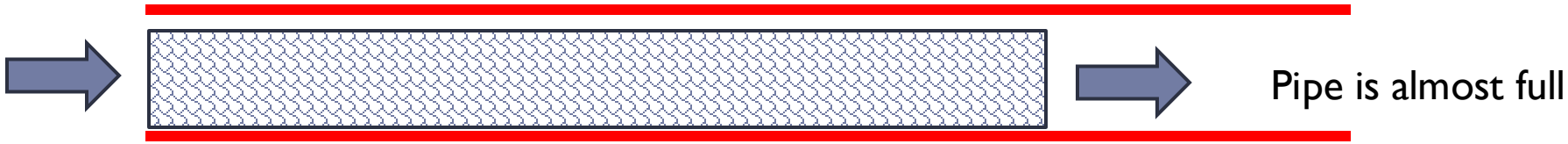
- When the **queues at the routers (switches) overflow** at some level of network load - network is said to be congested
- **Predicting** when congestion is about to happen and then reducing source rate of sending data - congestion avoidance
- Taking **reactive measures** once the congestion happens - congestion control

Broad Classification of Congestion Control Algorithms

A. **Loss based algorithms**
B. Delay based algorithms
C. Network assisted algorithms
   (ECN - Explicit Congestion Notification)

A. Loss based Congestion Control

Goal 1: Efficiently use the available network bandwidth.



Idea:

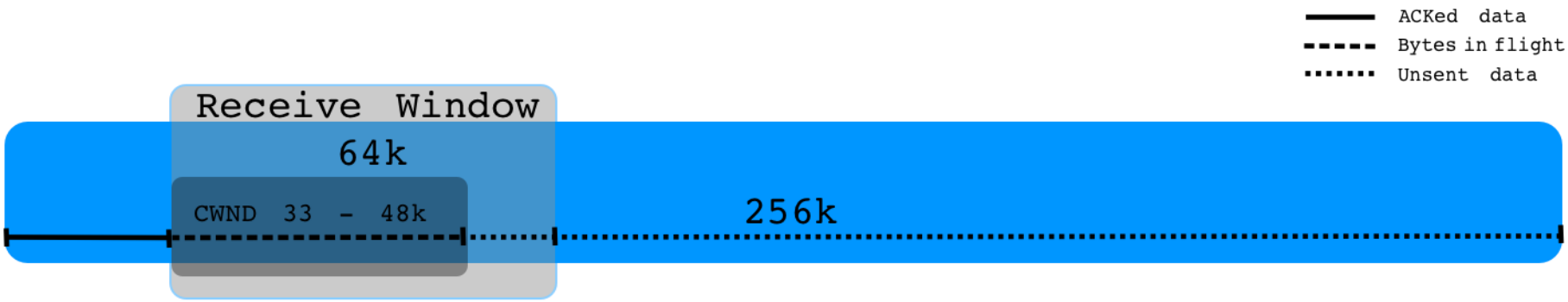
- **Capacity estimation [Capacity Probing]:** determine how much capacity is available in the network and estimate how many packets it can have in transit.
  - **Self clocking:** use the **arrival of an ack** as a signal that one of its packet has been removed from the network (or reached the destination)
    - then insert a new packet into the network.
- Lost packet – indicates **congestion.**
  - Trigger congestion control mechanism

A. Loss based Congestion Control

- Two points on implementation:
  - at each router - appropriate queuing discipline and buffer management
  - at end hosts - **controlled rate of sending packets**

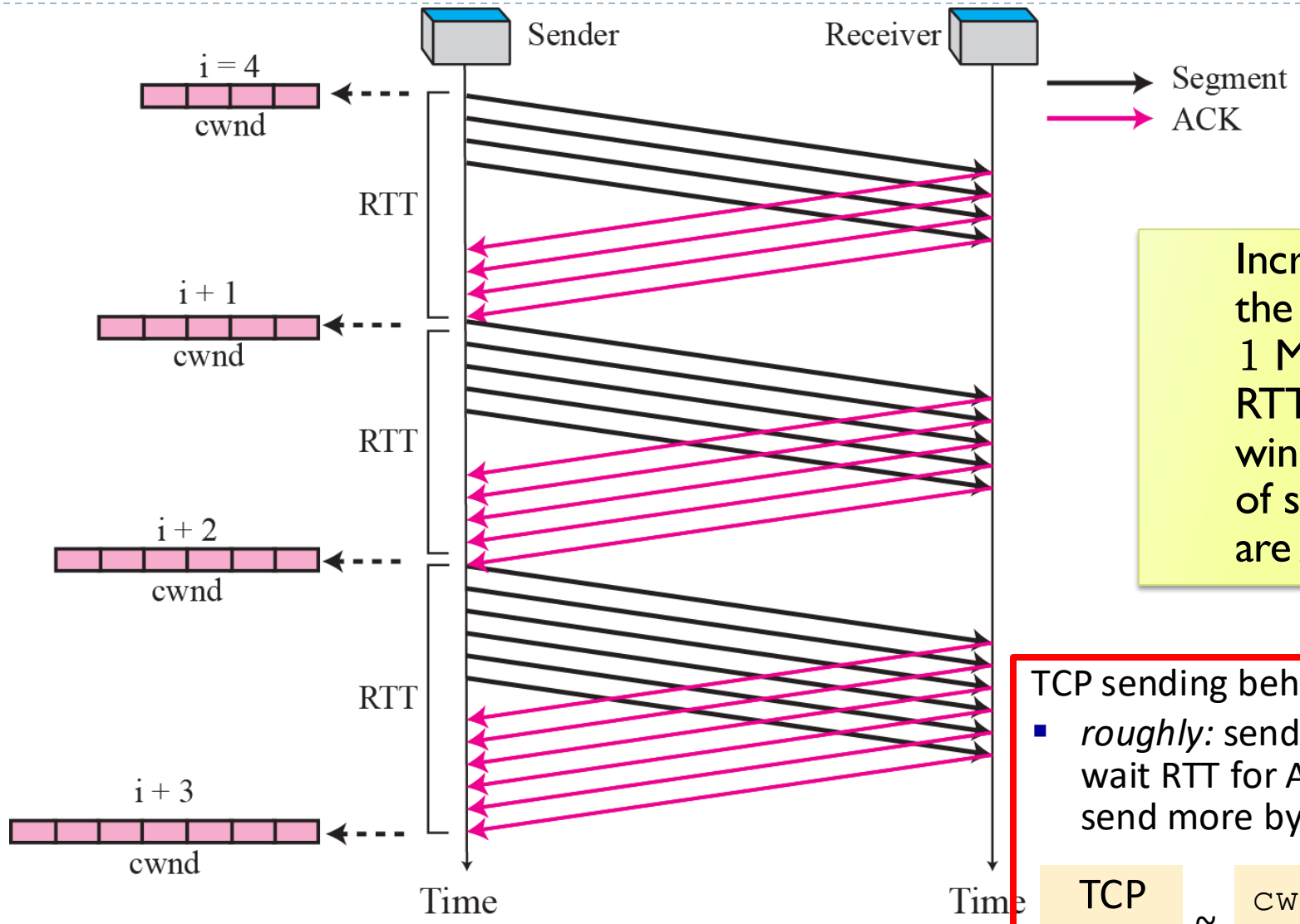
TCP Congestion Control Algorithm

1. **Congestion Avoidance** (Steady State, **Capacity probing**) : behaviour with mild congestion
2. **Slow Start** (**control congestion**): behaviour after serious congestion & also immediately after a TCP session starts
   - Fast Recovery (maintain a higher rate) after congestion
3. New window is introduced: cwnd – Congestion Window (usually stated in MSS)
   - Sender maintains cwnd and rwnd is advertised by the receiver. Sender will send either cwnd amount or data or rwnd amount of data, which ever is smallest.



Capacity Probing (Congestion Avoidance)

ADDITIVE INCREASE ALGORITHM



Increments the cwnd by 1 MSS every RTT (or one windows full of segments are ACKed)

TCP sending behavior:

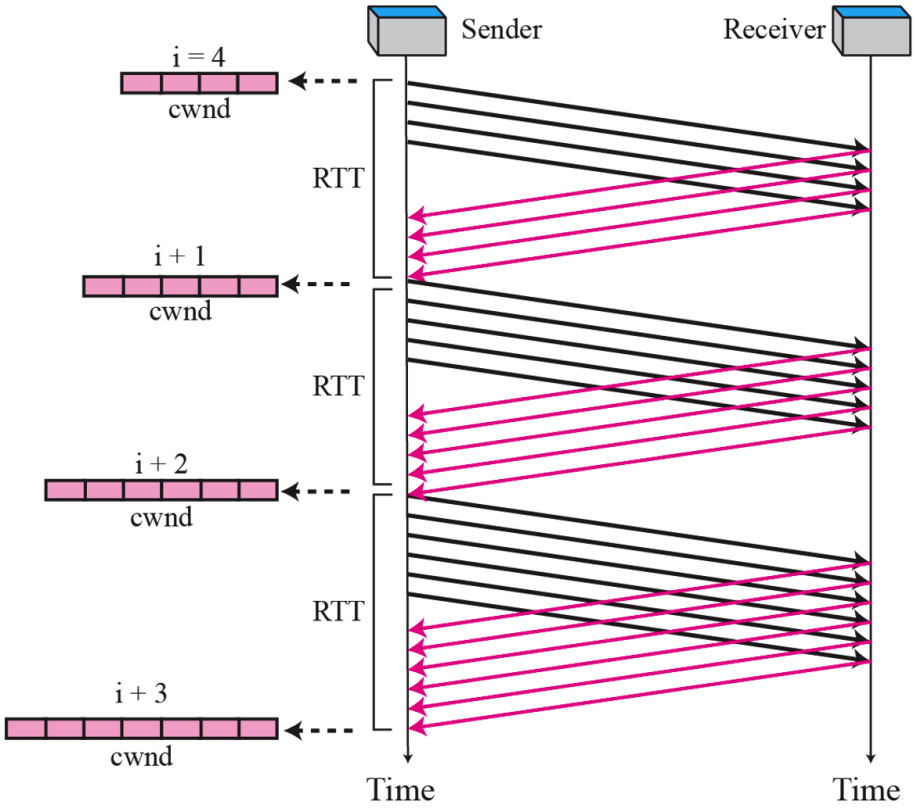
- *roughly:* send `cwnd` bytes, wait RTT for ACKS, then send more bytes

$$\text{TCP rate} \approx \frac{\texttt{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

Capacity Probing (Congestion Avoidance)

ADDITIVE INCREASE ALGORITHM

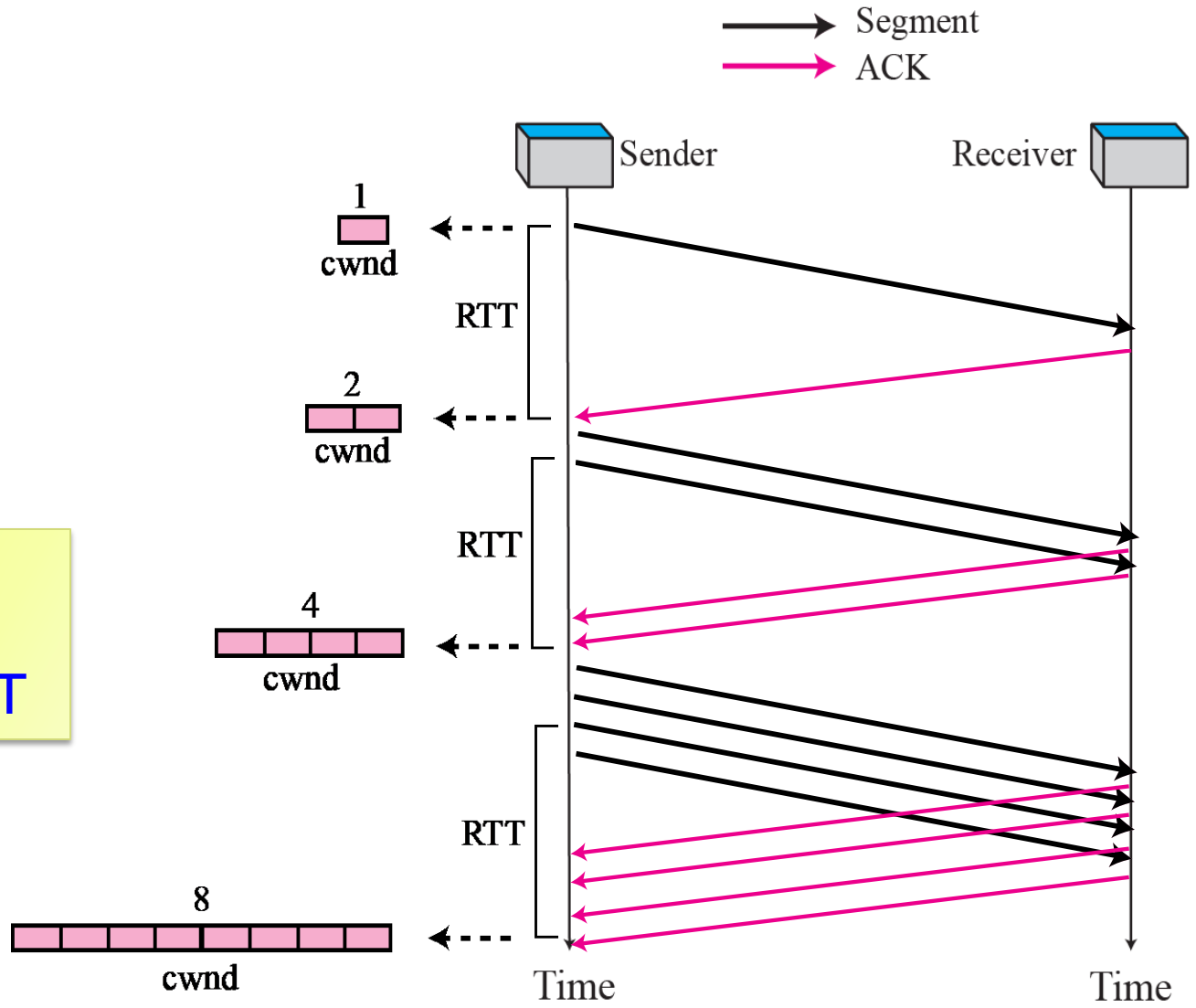
- Additive Increase is fine for capacity probing.
  - But, when to stop?
- When congestion is detected, we can reset the cwnd to small value.
  - How small?
  - What happens to the goal – 'efficient use of nw bandwidth'?
- Need another algorithm!



Slow Start *Algorithm*

- Additive increase is appropriate when the network operates near congestion (cwnd is not too small and more data is sent).
- If congestion is detected, cwnd is set to small value to control the data rate. If the cwnd is small (eg. at the start), it takes too long to ramp up a connection from such a small value.
- TCP provides another mechanism to increase cwnd exponentially from a cold start
  - sender sets cwnd to 1 segment
  - with each ACK arrival cwnd is incremented by one segment
  - effectively doubles the cwnd every RTT

Slow start algorithm, exponential increase



doubles the cwnd every RTT

Slow Start algorithm *(cont)*

- Two different situations in which slow start runs
  - **the very beginning of a connection**
    - source has no idea about the network load
    - slow start continues until there is a loss or reaching max receive window size of 64KB (old version). Q: Why 64K?
      - Currently , with TCP window scale option, the receive window size may be increased up to a maximum value of 1,073,725,440 bytes (1 GB). [Since Windows 2000, Since Linux kernel version 2.6.8, August 2004]. Q: How window-scale option is used?
  - **when a loss is detected through timeout**

❖ **cwnd** → 1 MSS (packet loss detected through timeout)

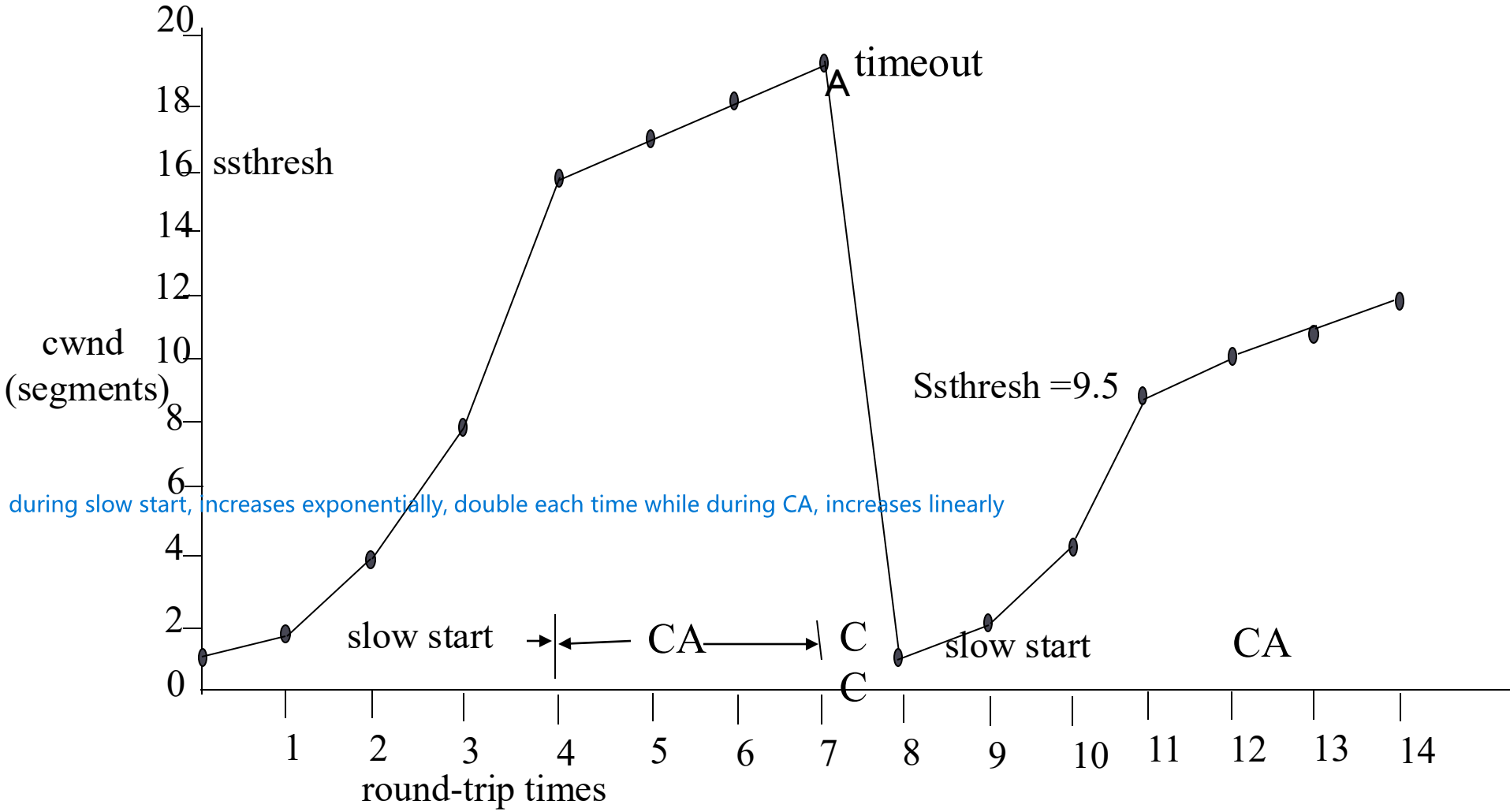
❖ max(2 mss, 1/2 min(cwnd, advertised rwnd)) is saved in **ssthresh** (slow start threshold) - a variable set for each connection (initial value = 64KB, old versions)

❑ Currently, the initial value of **ssthresh** may be set arbitrarily high (e.g., to rwnd or higher).

❑ More appropriate value is determined only after the first packet loss.

Example (slow start algorithm)

AIMD – Additive Increase Multiplicative Decrease



Note:

Most implementations react differently to congestion detection:

- *If detection is by time-out, a new slow start phase starts. (why?)*
- *If detection is by three duplicate ACKs, a new congestion avoidance phase starts. (why?)*

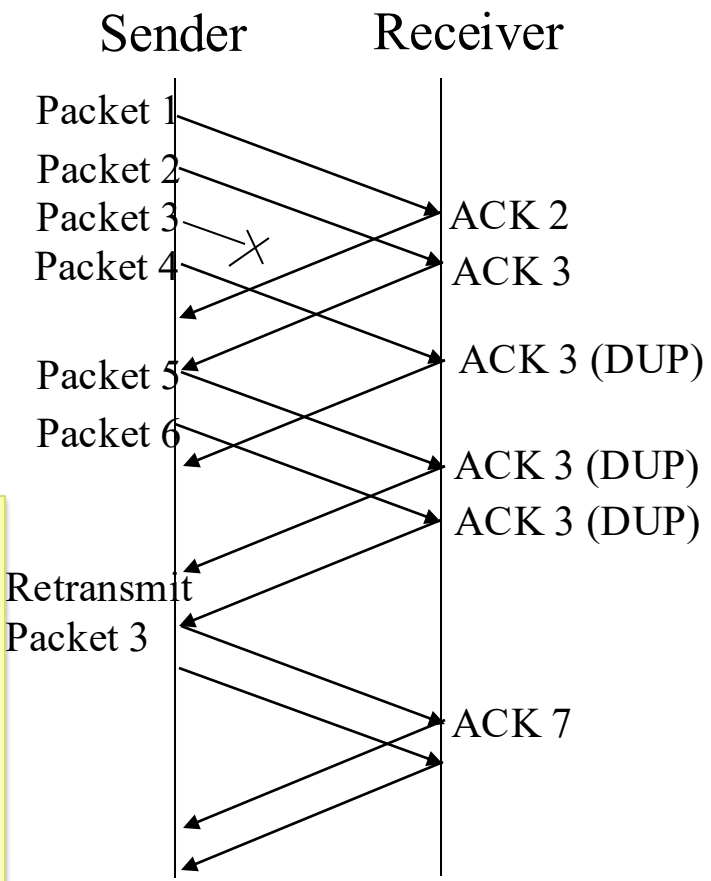
Note: Early version of TCP, "**TCP Tahoe**" do not make this distinction. Always enters slow start when there is a congestion.

Fast Recovery and Fast Retransmit

Fast Retransmit

3 DUPACKs TCP is an early indication of possible packet loss. Sender enters **Fast retransmit state.**

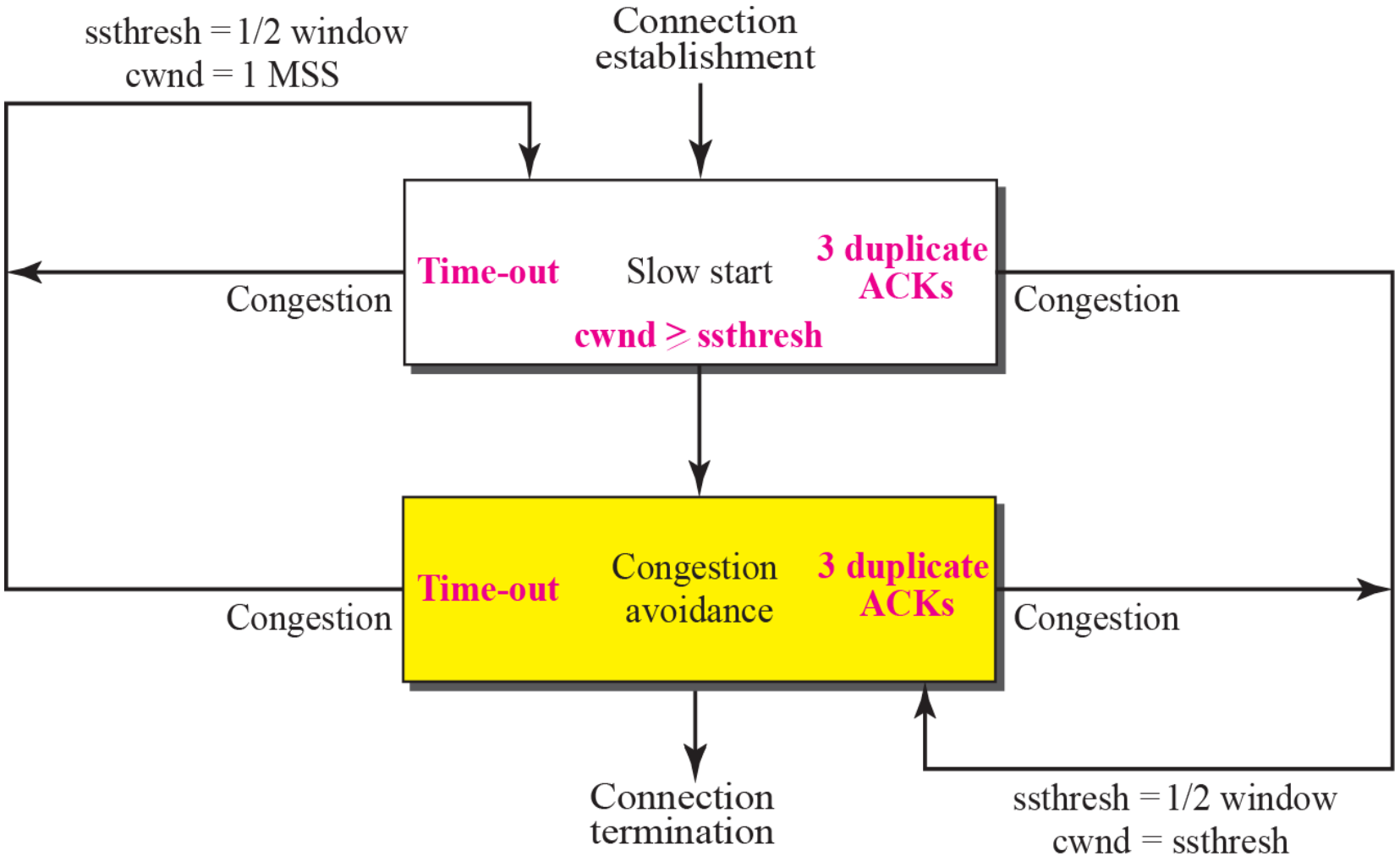
- Sender: Retransmit a packet on receiving 3 DUPACKs
- Receiver: When a packet arrives out-of-order, receiver sends a duplicate ACK



Fast Recovery

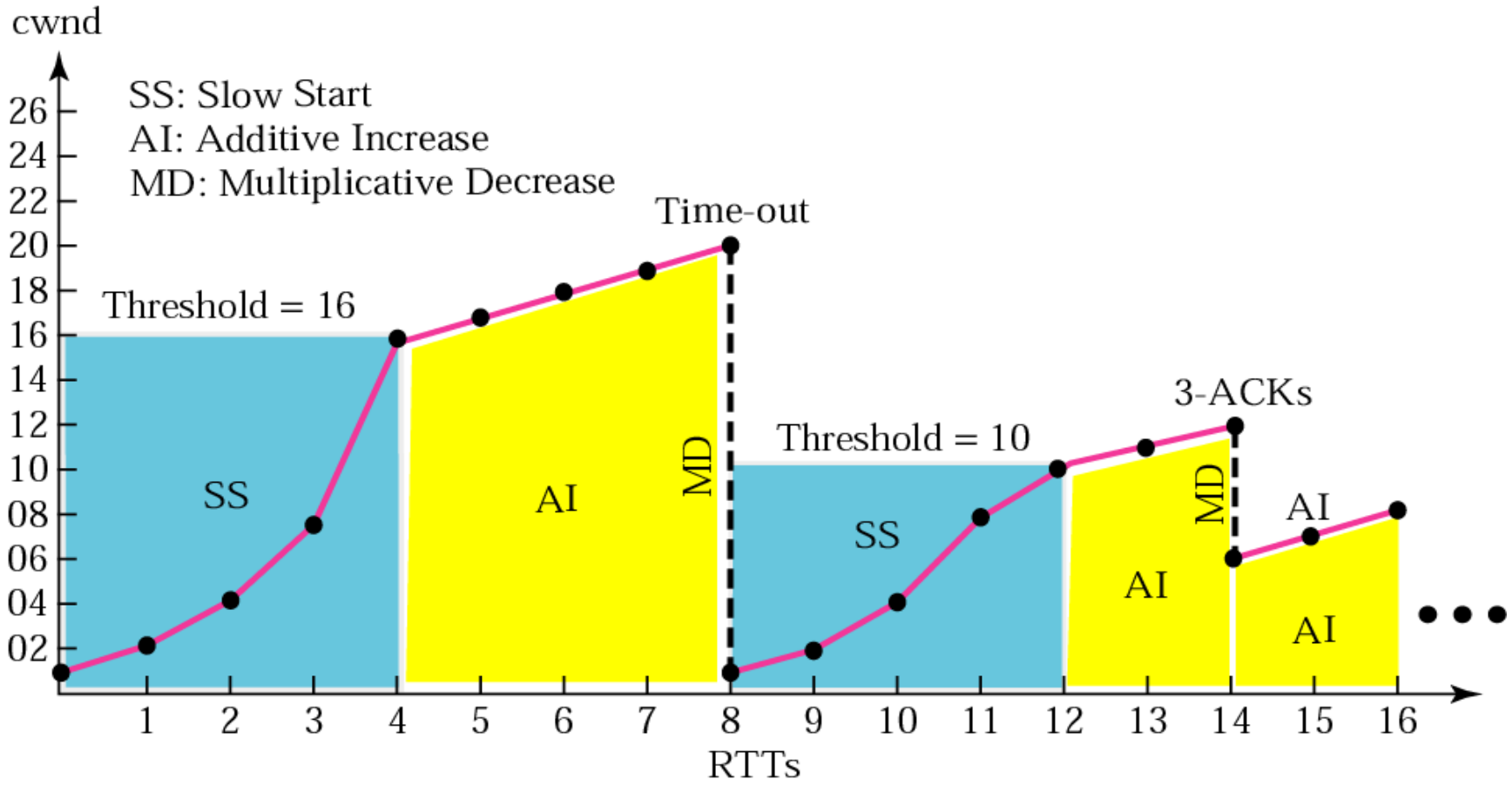
- 3 DupACKs is also considered as signal of light level congestion
- TCP Reno implements **Fast Recovery.**

TCP Reno congestion policy summary



TCP Reno - Congestion example

AIMD – Additive Increase Multiplicative Decrease



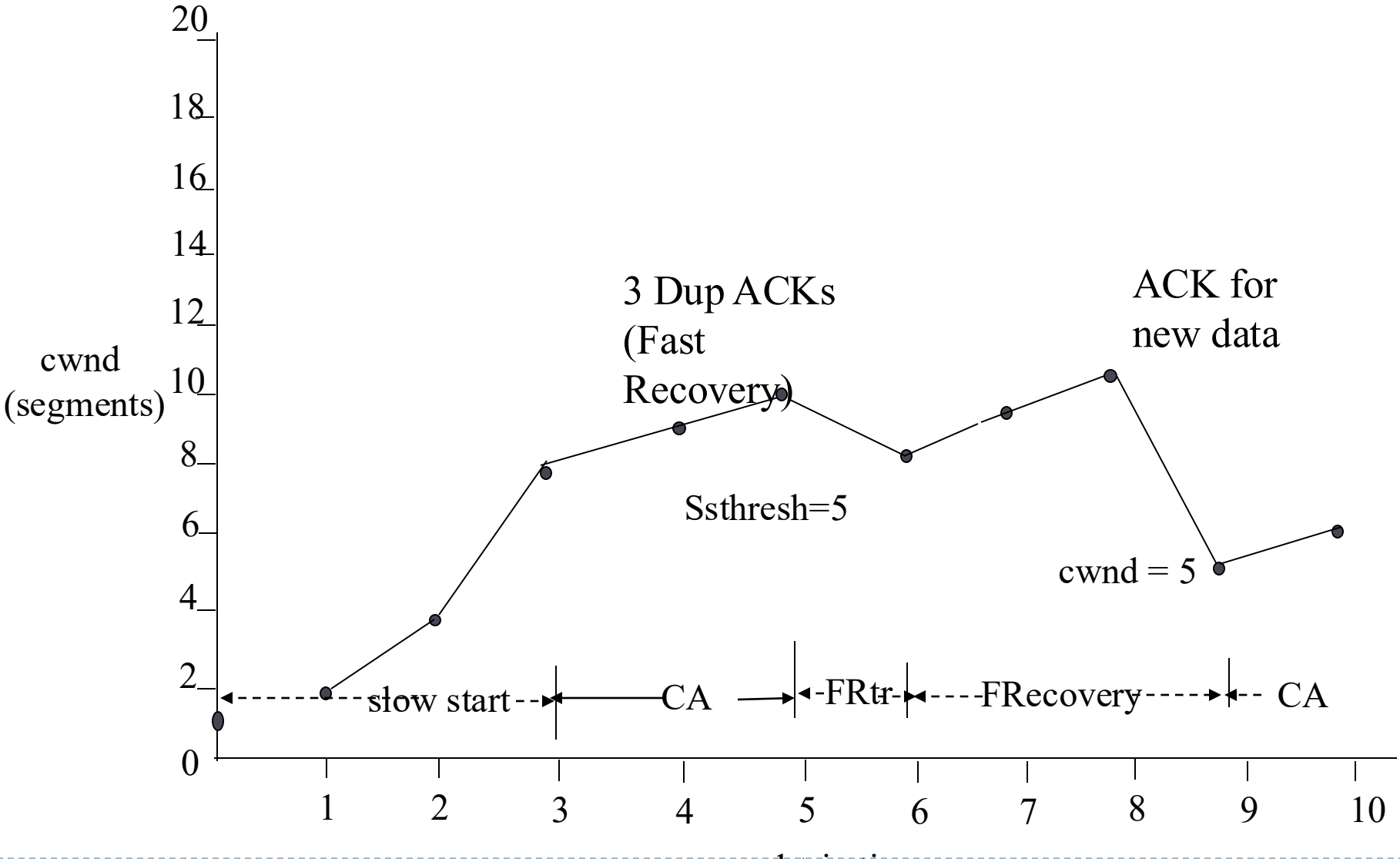
Fast Retransmit & Recovery Algorithm in **TCP New Reno – Default upto Linux v 2.6.7**

Fast Retransmit and Fast Recovery are implemented together in TCP Reno as follows (RFC 2581):

1. When the 3rd DUP ACK is received, set
   ssthreah = max (*cwnd*/2, 2 × MSS)
2. Retransmit the lost segment & set
   cwnd = *ssthresh* + 3 × MSS
3. For each additional DUP ACK received,
   cwnd = *cwnd* + MSS
4. Transmit a segment, if allowed by new value of *cwnd* and receiver's advertised window
5. When next ACK arrives that acks new data, set
   cwnd = *ssthresh* (value in Step 1.) [ends fastRecovery Phase]

In Steps 2, the **congestion window is artificially inflated** by the number of segments that have left (removed from the) the network.

Example– Fast Recovery & Fast Retransmit

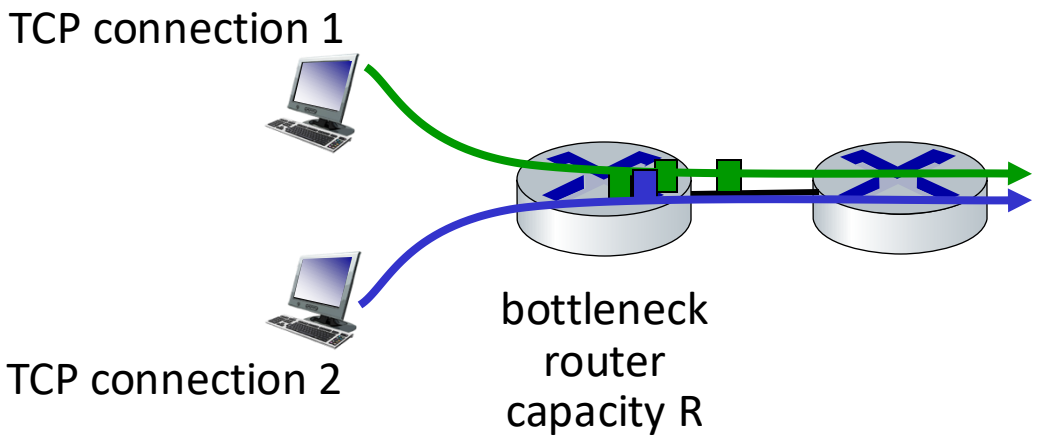


TCP fairness (Multiple TCP sessions)

Goal 1: Efficiently use the available network bandwidth.

Goal 2: Fair share of the bandwidth among multiple senders

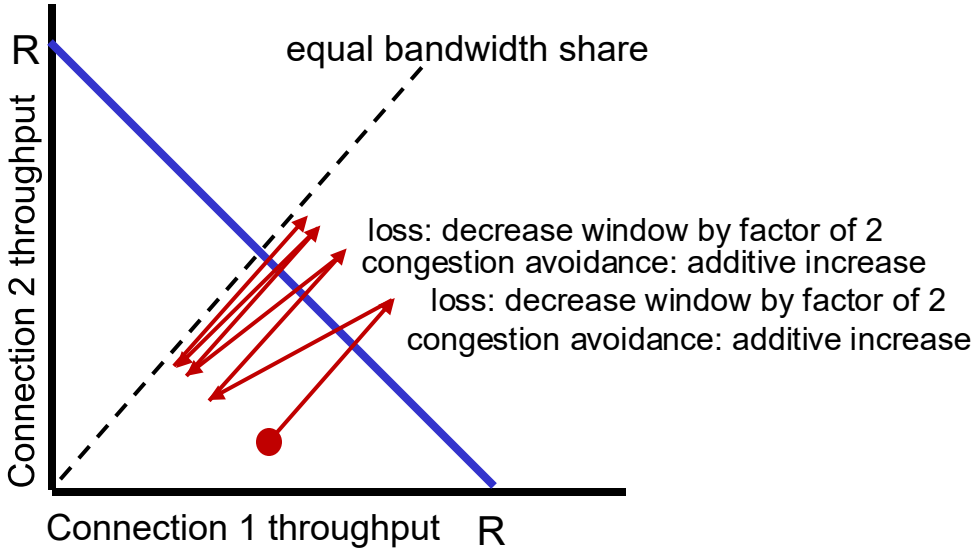
Fairness goal: if *K* TCP sessions share same bottleneck link of bandwidth *R*, each should have average rate of *R/K*



Q: is TCP Fair?

Example: two competing TCP sessions:

- additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally



Is TCP fair?

A: Yes, under idealized assumptions:

- same RTT
- fixed number of sessions only in congestion avoidance

Fairness: must all network apps be “fair”?

Fairness and UDP

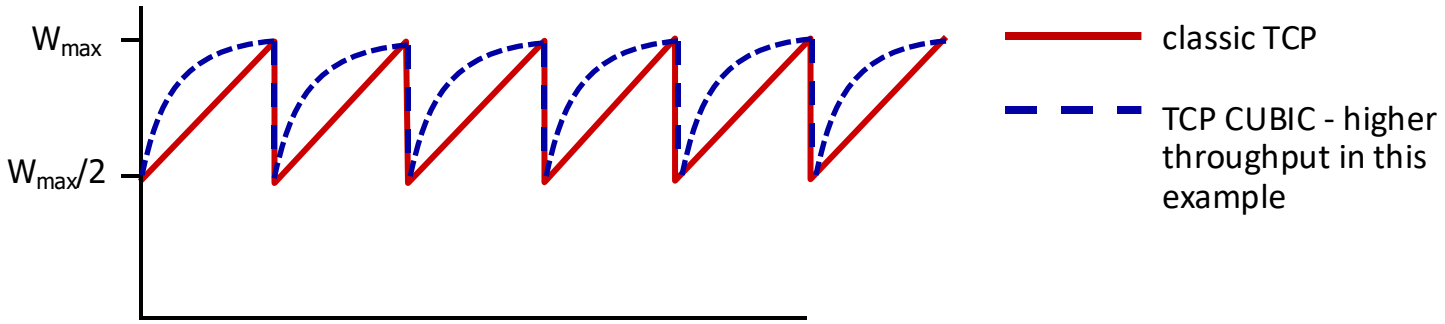
- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss
- there is no “Internet police” policing use of congestion control

Fairness, parallel TCP connections

- application can open *multiple* parallel connections between two hosts
- web browsers do this , e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2

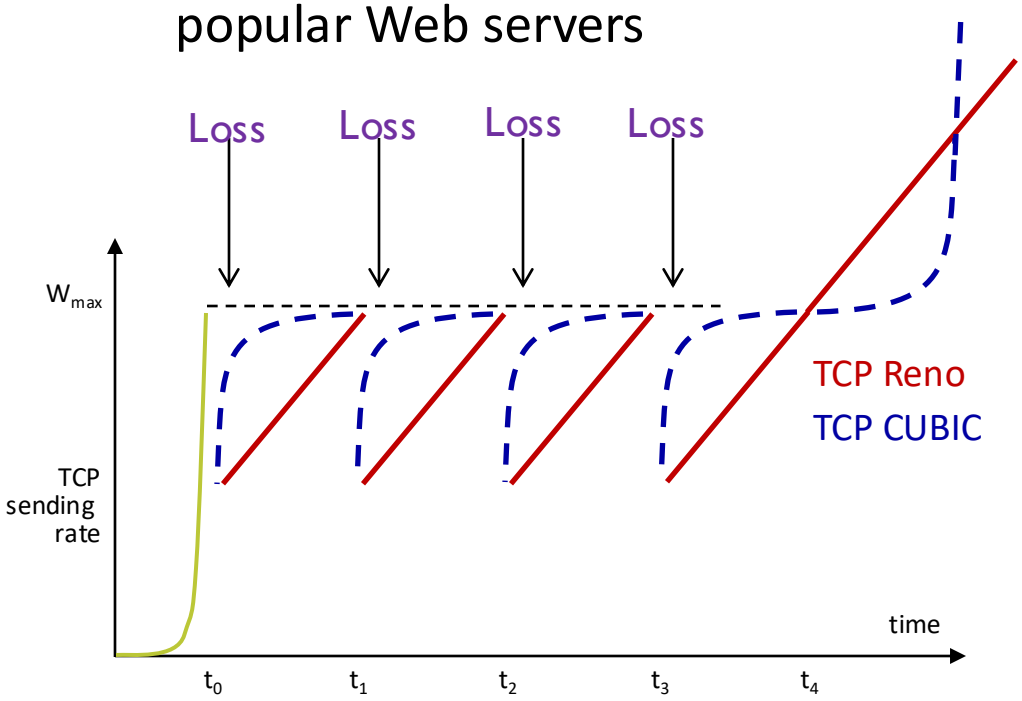
TCP CUBIC – (current default algorithm in Linux)

- Is there a better way than AIMD to "probe" for usable bandwidth?
- Insight/intuition:
  - W_{max}: sending rate (window size) at which congestion loss was detected
  - congestion state of bottleneck link probably (?) hasn't changed much
  - after cutting rate/window in half on loss, initially ramp up to W_{max} *faster*, but then approach W_{max} more *slowly*

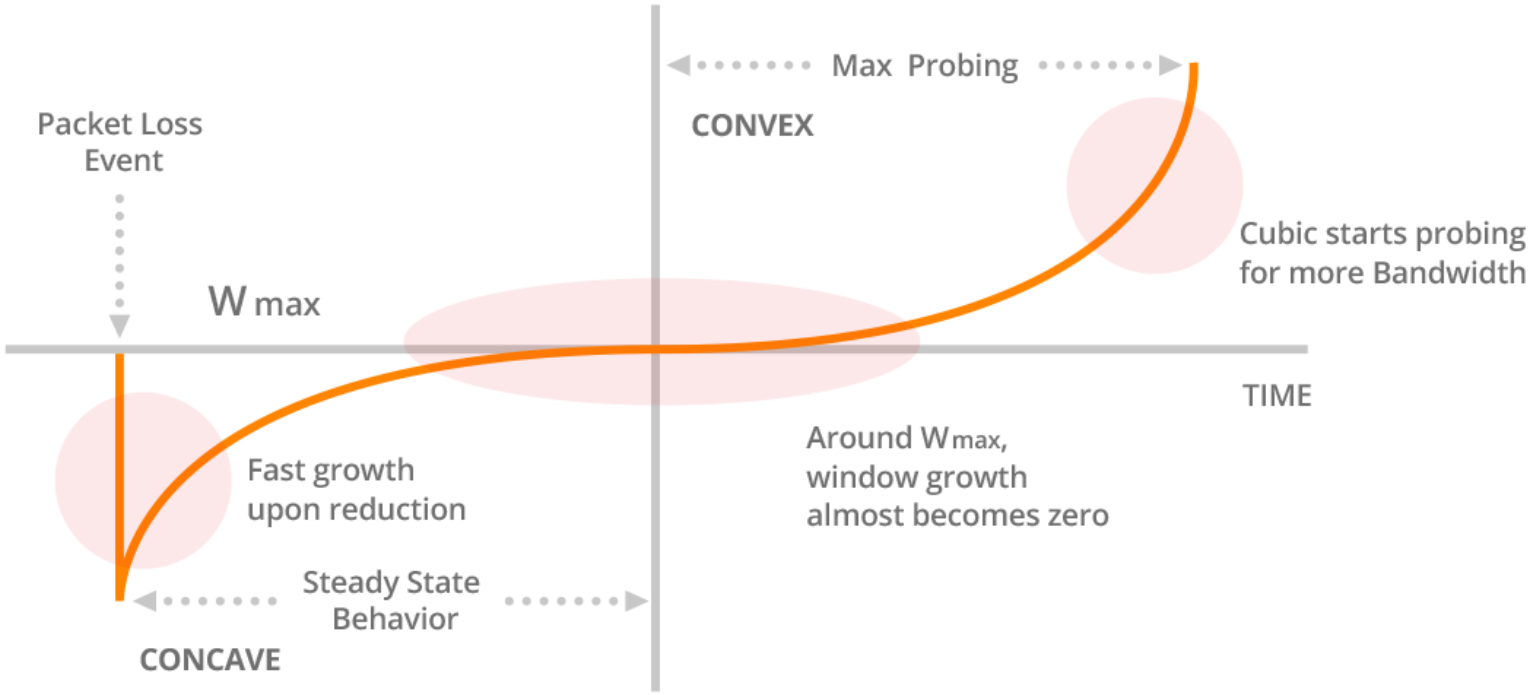


TCP CUBIC

- K: point in time when TCP window size will reach W_{max}
  - K itself is tuneable
- increase W as a function of the *cube* of the distance between current time and K
  - larger increases when further away from K
  - smaller increases (cautious) when nearer K
- TCP CUBIC default in Linux, most popular TCP for popular Web servers



TCP CUBIC


Max Probing
CONVEX
Packet Loss Event
W max
Cubic starts probing for more Bandwidth
TIME
Around W max, window growth almost becomes zero
Fast growth upon reduction
Steady State Behavior
CONCAVE

Broad Classification of Congestion Control Algorithms

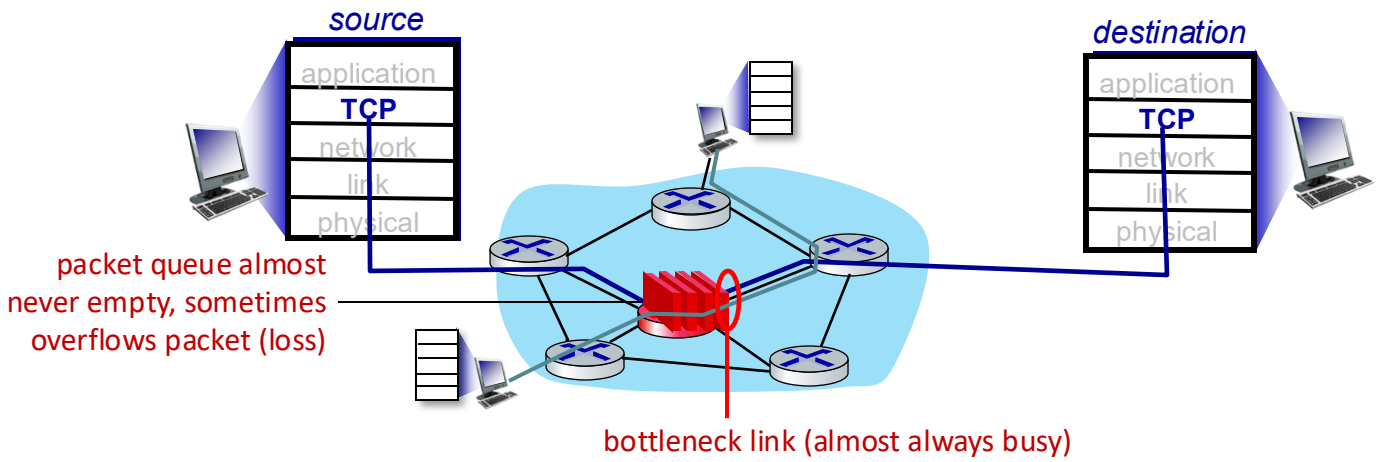
A. Loss based algorithms
B. **Delay based algorithms**
C. Network assisted algorithms
(ECN - Explicit Congestion Notification)

B. Delay based Congestion Control

- Basic Idea
  - Normal RTT means transmitting at the capacity.
  - Longer RTT means there is a congestion.
  - Smaller RTT means the bandwidth is underutilised
- Eg. BBR (*Bottleneck Bandwidth and Round-trip propagation time*) by Google. Available in Linux

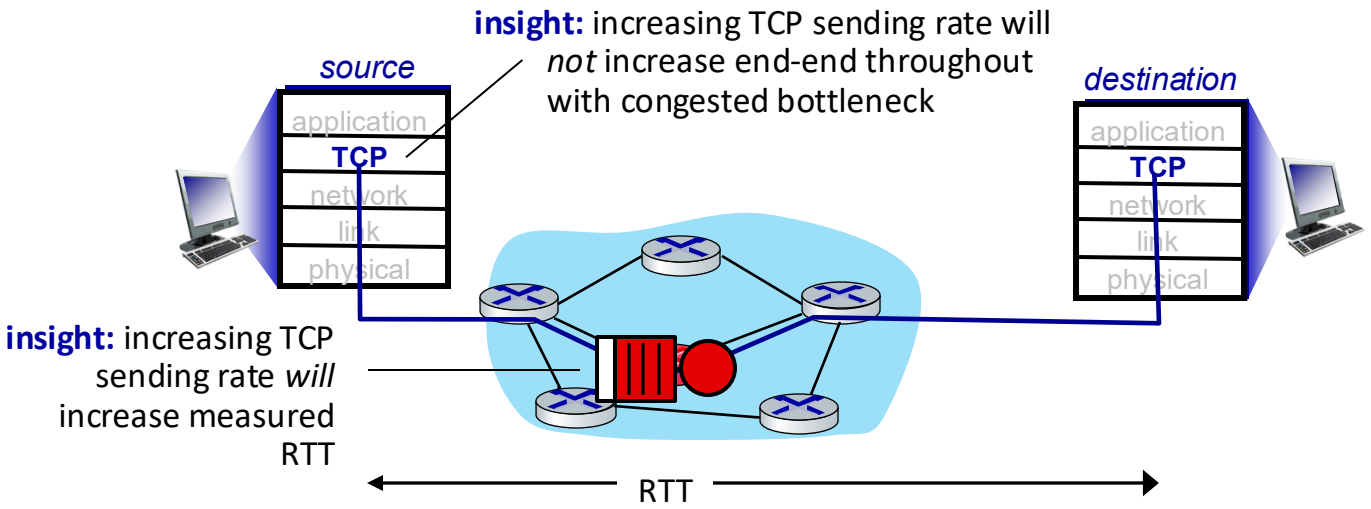
TCP and the congested "bottleneck link"

- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*



TCP and the congested "bottleneck link"

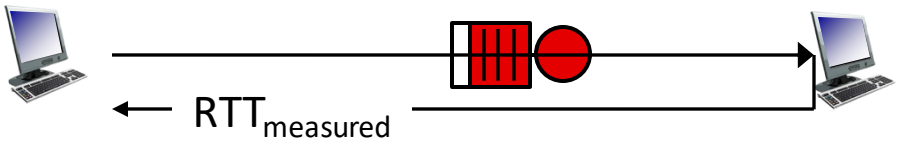
- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*
- understanding congestion: useful to focus on congested bottleneck link



Goal: *"keep the end-end pipe just full, but not fuller"*

Delay-based TCP congestion control

Keeping sender-to-receiver pipe "just full enough, but no fuller": keep bottleneck link busy transmitting, but avoid high delays/buffering



Delay-based approach:

- RTT_{min} - minimum observed RTT (uncongested path)
- uncongested throughput with congestion window `cwnd` is cwnd/RTT_{min}

if measured throughput *"very close" to uncongested throughput*
increase `cwnd` linearly /* since path not congested */
else if measured throughput *"far below" uncongested throughout*
decrease `cwnd` linearly /* since path is congested */

$$\textbf{measured throughput} = \frac{\text{\# bytes sent in last RTT interval}}{RTT_{measured}}$$

Delay-based TCP congestion control

- congestion control without inducing/forcing loss
- maximizing throughout ("keeping the just pipe full… ") while keeping delay low ("…but not fuller")
- a number of deployed TCPs take a delay-based approach
  - BBR deployed on Google's (internal) backbone network

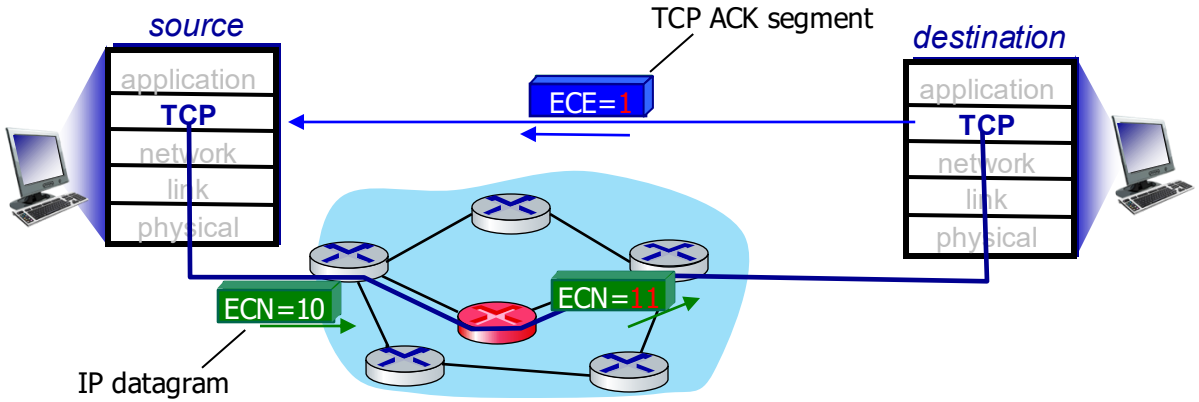
Broad Classification of Congestion Control Algorithms

A. Loss based algorithms
B. Delay based algorithms
C. **Network assisted algorithms**
(ECN - Explicit Congestion Notification)

Network assisted algorithm
(using ECN - Explicit congestion notification)

TCP deployments often implement *network-assisted* congestion control:

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
  - *policy* to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets ECE bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)



IP ToS (flags)

0	1	2	3	4	5	6	7
DSCP field						ECN field	

TCP Control bits (flags)

C	E	U	A	P	R	S	F
W	C	R	C	S	S	Y	I
R	E	G	K	H	T	N	N

ECE – Explicit Congestion ECHO

TCP Congestion Control

- TCP State Transitions
- Congestion Control
- LABs :
  - Lab 9: TCP and Its Performance

Question to Ponder: Compare the performance of CUBIC and BBR.

Current standards:

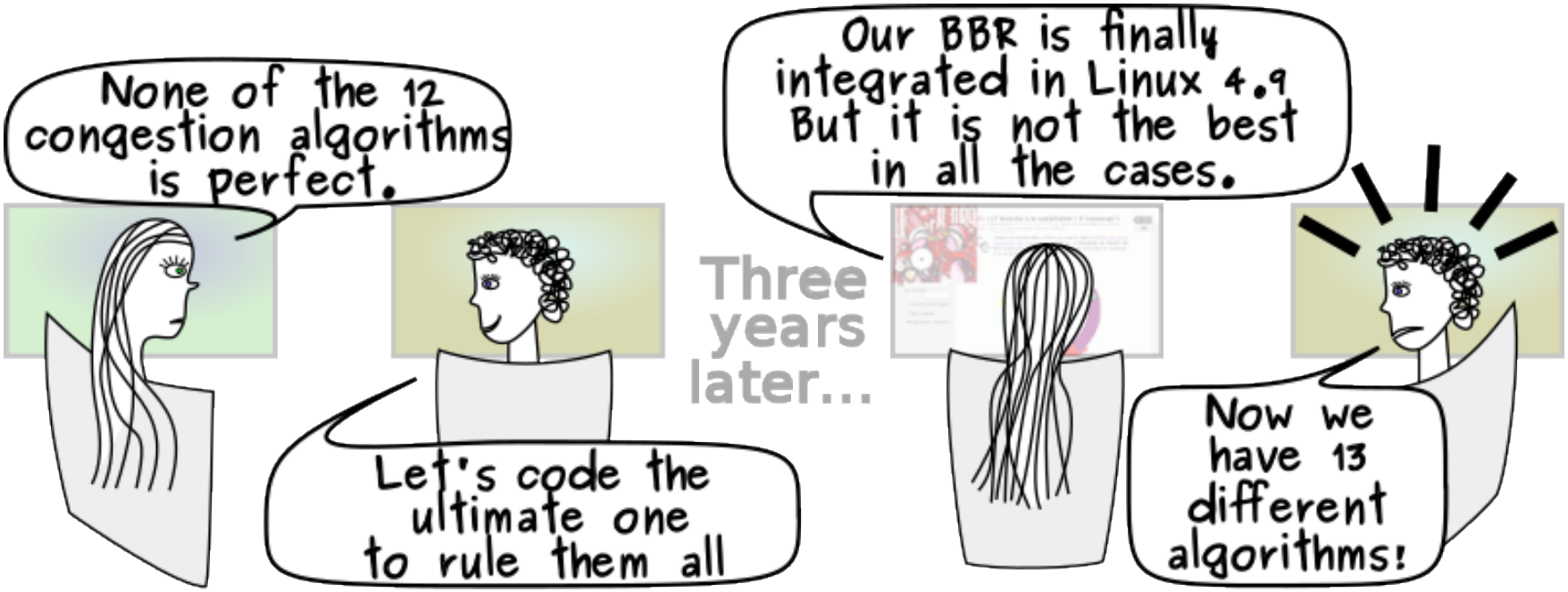
Current Default Version in Linux Kernels

- From V 2.6.8 – TCP BIC
- From V 2.6.19 – TCP CUBIC
  - BIC, CUBIC have advantages with LFNs
- Recenly linux version 4.9 also added Google's BBR, developed in 2016. [BBR is model based, while the previous versions are loss based]. It is NOT default algo in Linux.

Current Default Version in Windows:

- Compound-TCP

TCP Congestion Control



THE END