Chapter 12

# Transmission Control Protocol

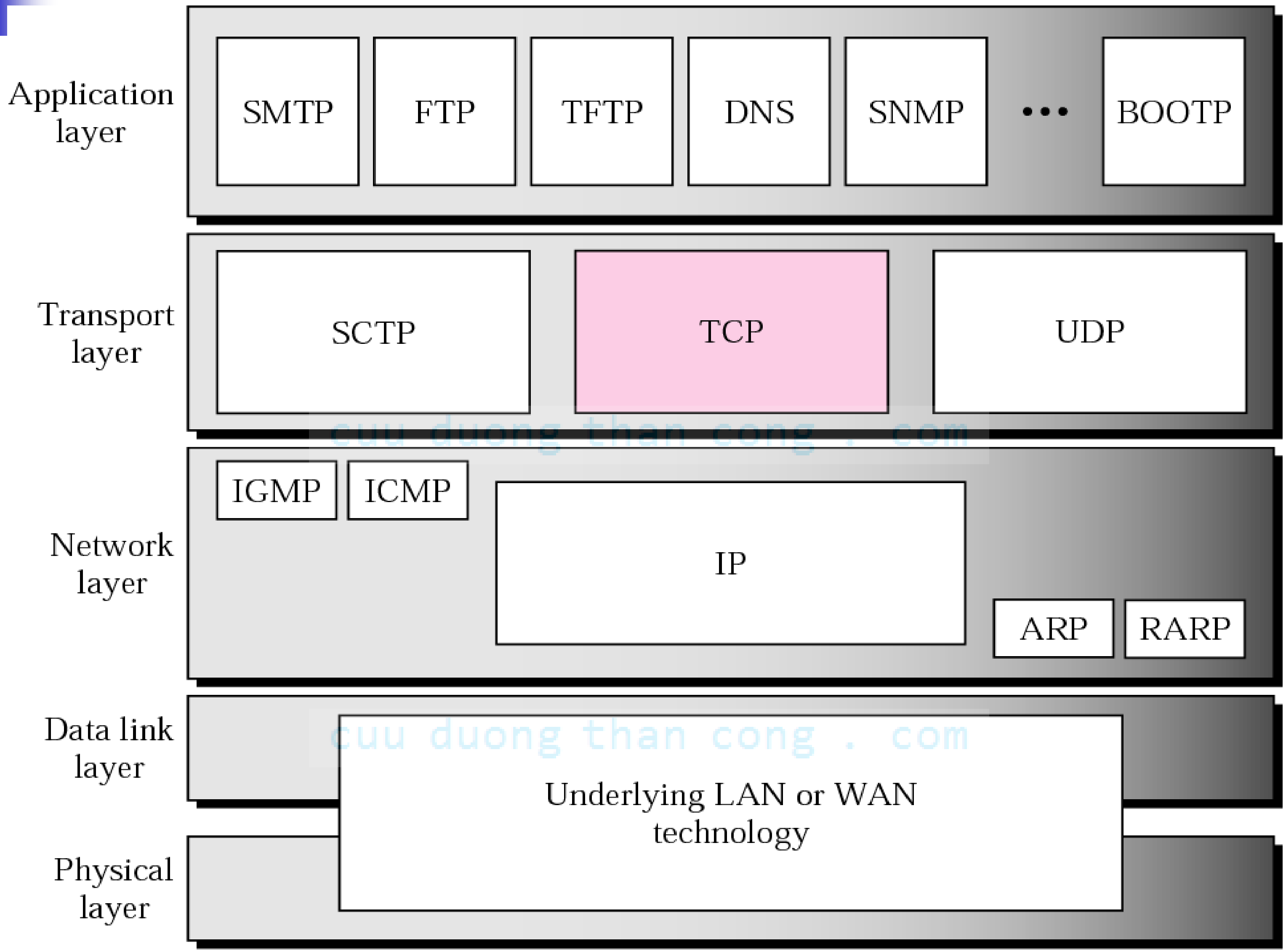
## Objectives



*Upon completion you will be able to:*

- *Be able to name and understand the services offered by TCP*
- *Understand TCP's flow and error control and congestion control*
- *Be familiar with the fields in a TCP segment*
- *Understand the phases in a connection-oriented connection*
- *Understand the TCP transition state diagram*
- *Be able to name and understand the timers used in TCP*
- *Be familiar with the TCP options*

## Figure 12.1 *TCP/IP protocol suite*



# 12.1 TCP SERVICES

*We explain the services offered by TCP to the processes at the application layer.*

*The topics discussed in this section include:*



*Process-to-Process Communication*
*Stream Delivery Service*
*Full-Duplex Communication*
*Connection-Oriented Service*
*Reliable Service*

## *Table 12.1 Well-known ports used by TCP*

| *Port* | *Protocol* | *Description* |
|---|---|---|
| 7 | Echo | Echoes a received datagram back to the sender |
| 9 | Discard | Discards any datagram that is received |
| 11 | Users | Active users |
| 13 | Daytime | Returns the date and the time |
| 17 | Quote | Returns a quote of the day |
| 19 | Chargen | Returns a string of characters |
| 20 | FTP, Data | File Transfer Protocol (data connection) |
| 21 | FTP, Control | File Transfer Protocol (control connection) |
| 23 | TELNET | Terminal Network |
| 25 | SMTP | Simple Mail Transfer Protocol |
| 53 | DNS | Domain Name Server |
| 67 | BOOTP | Bootstrap Protocol |
| 79 | Finger | Finger |
| 80 | HTTP | Hypertext Transfer Protocol |
| 111 | RPC | Remote Procedure Call |

# EXAMPLE 1

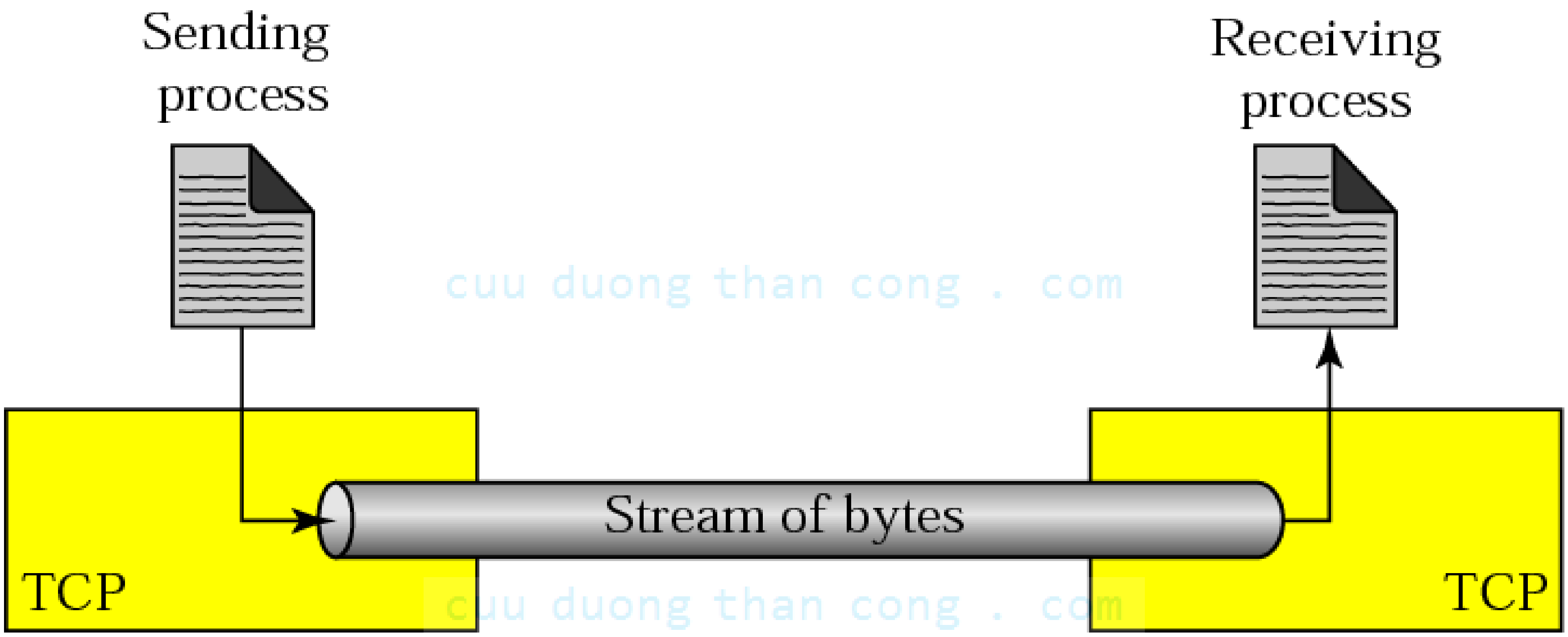
***As we said in Chapter 11, in UNIX, the well-known ports are stored in a file called /etc/services. Each line in this file gives the name of the server and the well-known port number. We can use the grep utility to extract the line corresponding to the desired application. The following shows the ports for FTP.***



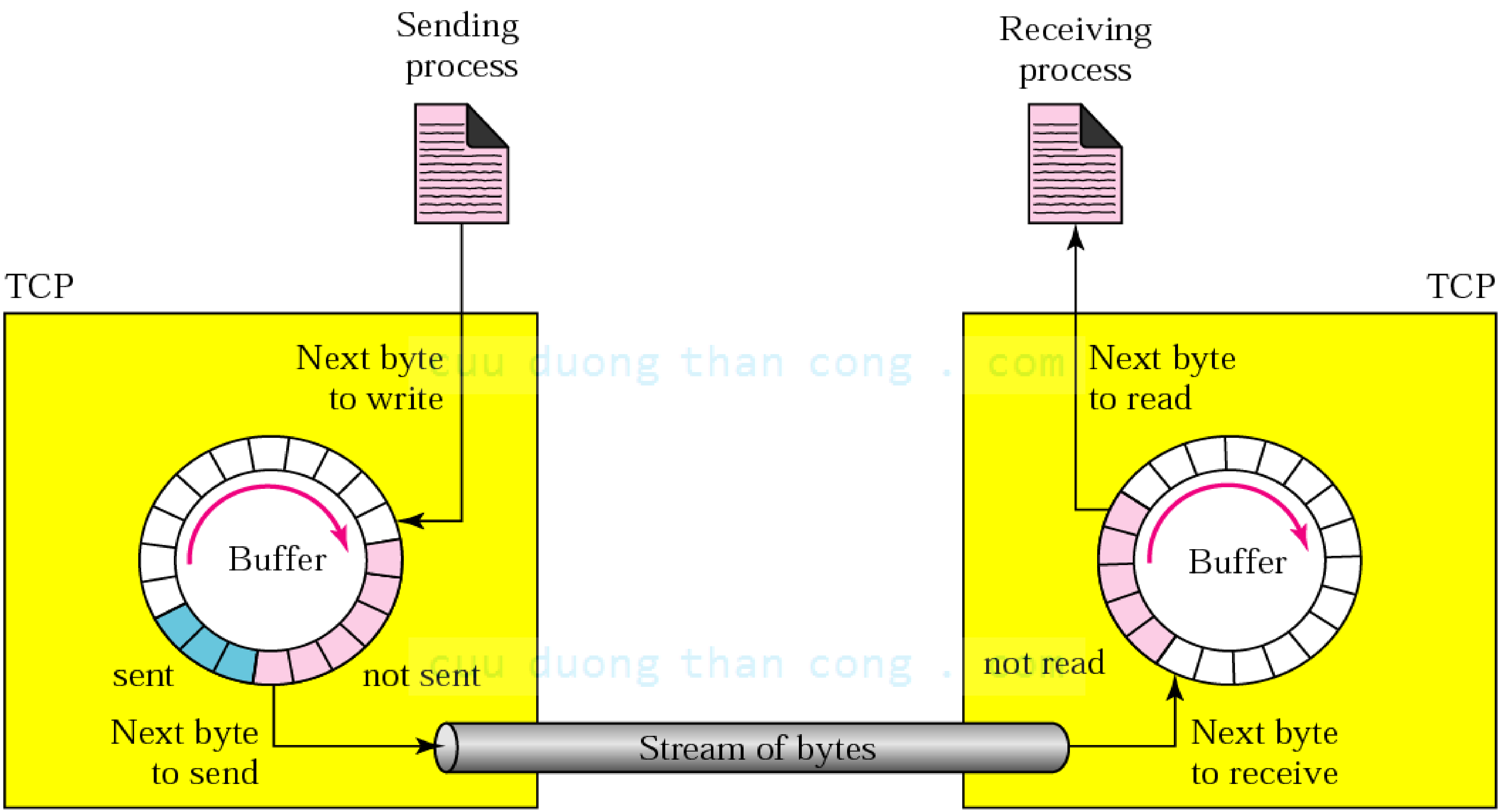
```
$ grep  ftp   /etc/services

ftp-data        20/tcp
ftp-control     21/tcp
```

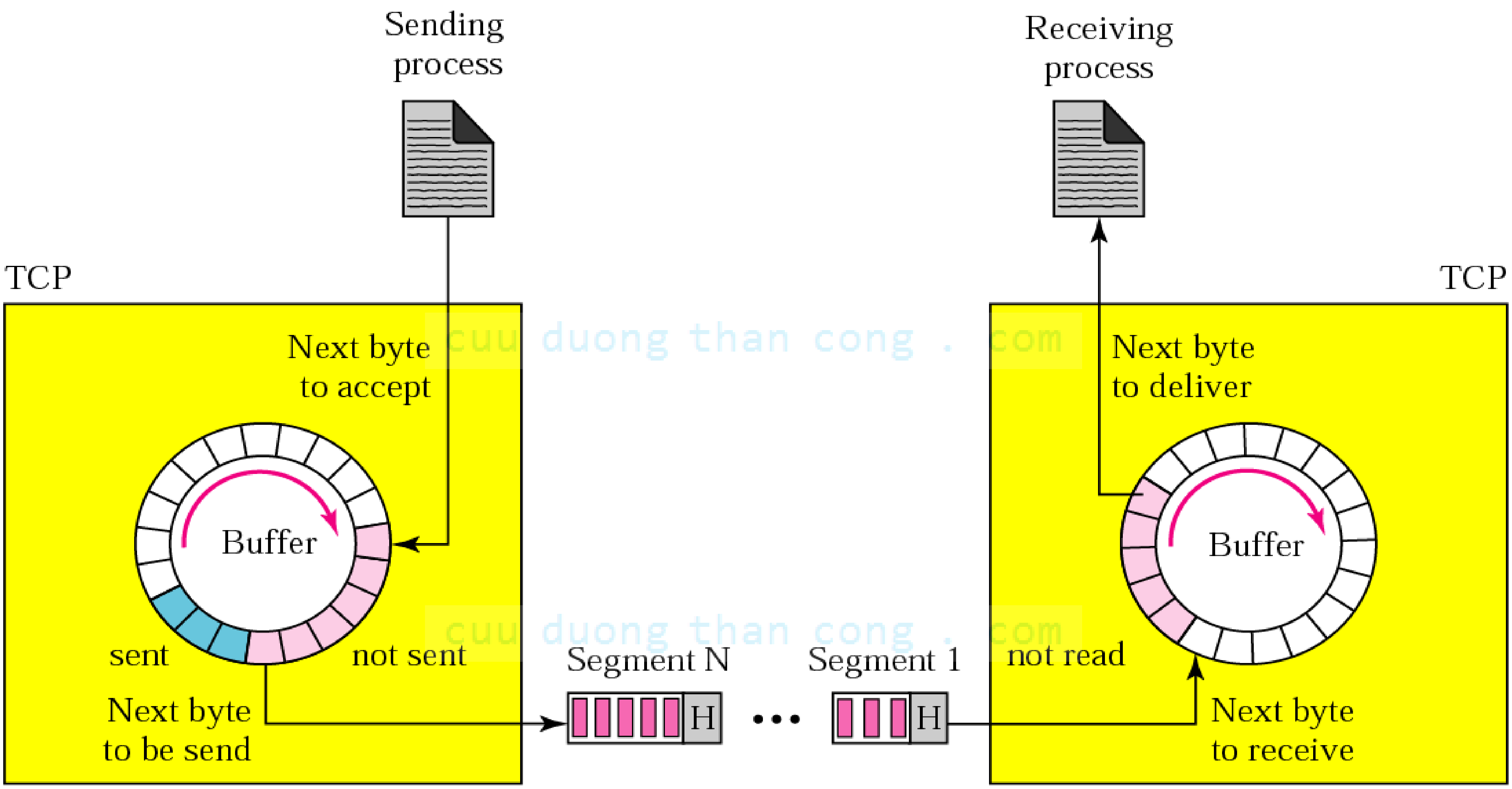
# Figure 12.2 *Stream delivery*



## Figure 12.3 *Sending and receiving buffers*



# Figure 12.4 *TCP segments*



# 12.2 TCP FEATURES

***To provide the services mentioned in the previous section, TCP has several features that are briefly summarized in this section.***



***The topics discussed in this section include:***

***Numbering System***
***Flow Control***
***Error Control***
***Congestion Control***

Note:

***The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with a randomly generated number.***

## EXAMPLE 2

***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 is sent in five segments, each carrying 1000 bytes?***

### *Solution*



***The following shows the sequence number for each segment:***

***Segment 1*** ➞ ***Sequence Number: 10,001 (range: 10,001 to 11,000)***

***Segment 2*** ➞ ***Sequence Number: 11,001 (range: 11,001 to 12,000)***

***Segment 3*** ➞ ***Sequence Number: 12,001 (range: 12,001 to 13,000)***

***Segment 4*** ➞ ***Sequence Number: 13,001 (range: 13,001 to 14,000)***

***Segment 5*** ➞ ***Sequence Number: 14,001 (range: 14,001 to 15,000)***

Note:

***The value in the sequence number field of a segment defines the number of the first data byte contained in that segment.***

Note:

***The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive.***

***The acknowledgment number is cumulative.***

# 12.3 SEGMENT

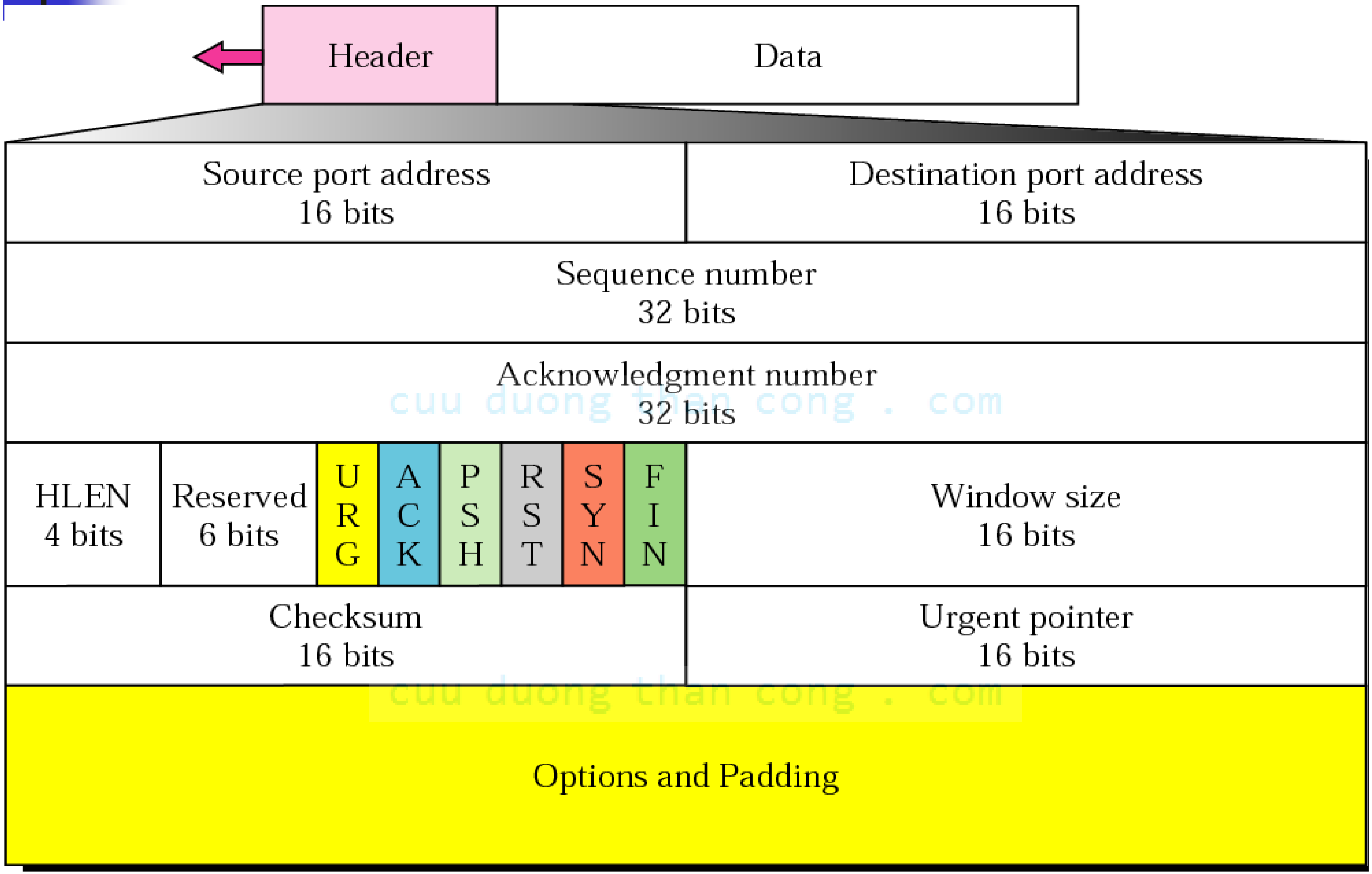
*A packet in TCP is called a segment*



*The topics discussed in this section include:*

*Format*
*Encapsulation*

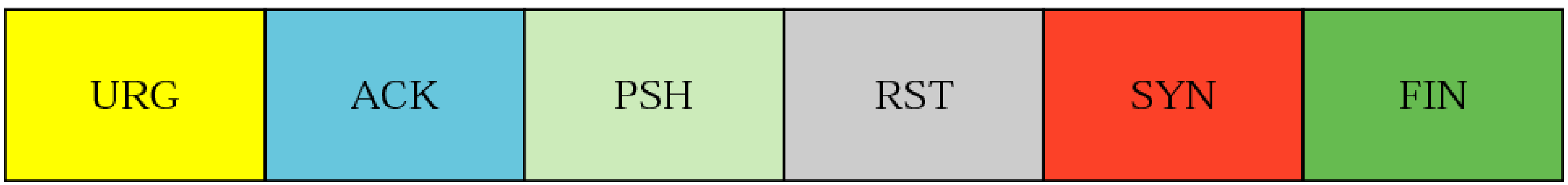
# **Figure 12.5** *TCP segment format*



# **Figure 12.6** *Control field*

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

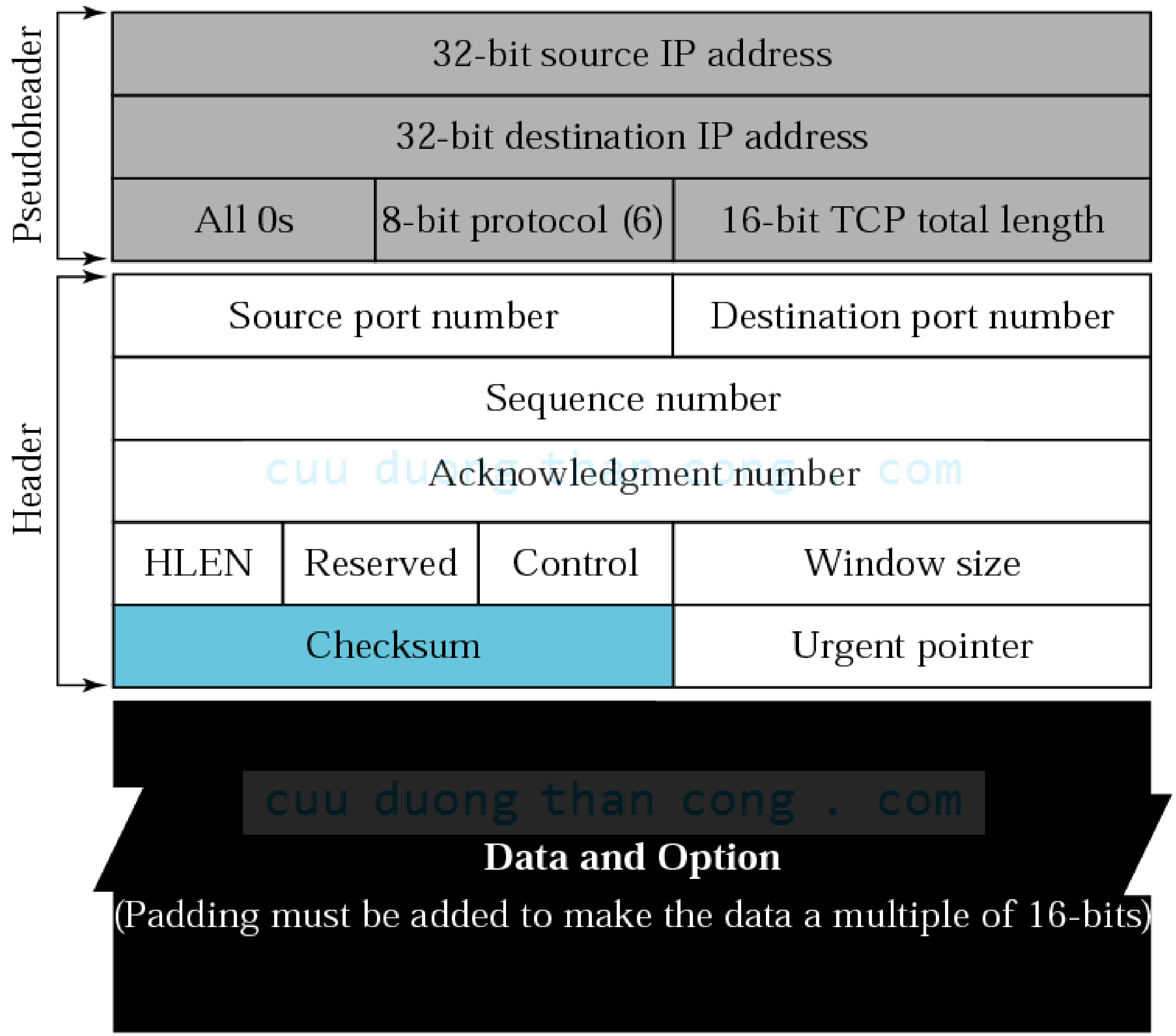




## *Table 12.2 Description of flags in the control field*

| *Flag* | *Description* |
|---|---|
| URG | The value of the urgent pointer field is valid |
| ACK | The value of the acknowledgment field is valid |
| PSH | Push the data |
| RST | The connection must be reset |
| SYN | Synchronize sequence numbers during connection |
| FIN | Terminate the connection |

# Figure 12.7 *Pseudoheader added to the TCP datagram*



Note:

***The inclusion of the checksum in TCP is mandatory.***

## **Figure 12.8** *Encapsulation and decapsulation*



# 12.4 A TCP CONNECTION

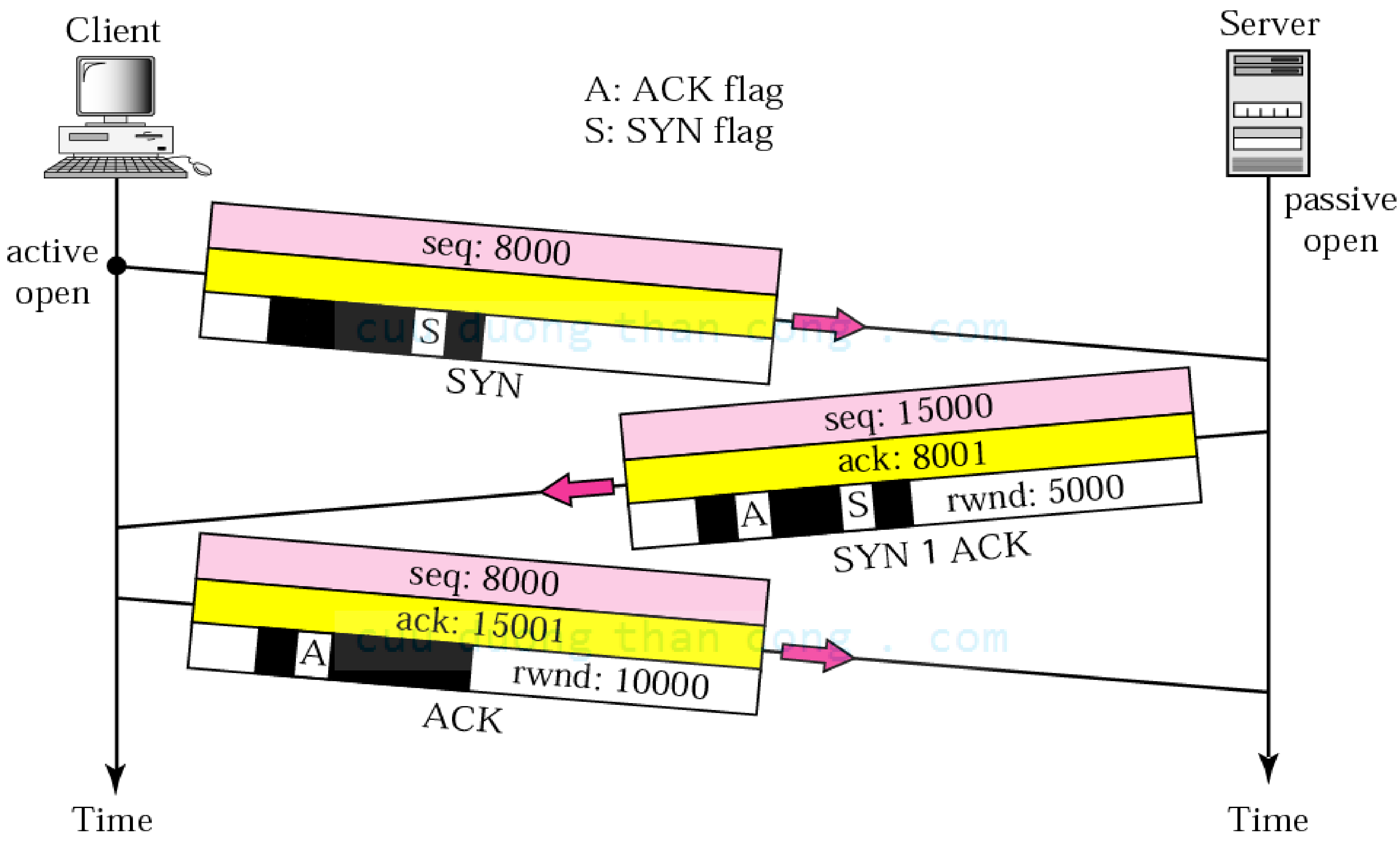
*TCP is connection-oriented. A connection-oriented transport protocol establishes a virtual path between the source and destination. All of the segments belonging to a message are then sent over this virtual path. A connection-oriented transmission requires three phases: connection establishment, data transfer, and connection termination.*



*The topics discussed in this section include:*

*Connection Establishment*
*Data Transfer*
*Connection Termination*
*Connection Reset*

## Figure 12.9 *Connection establishment using three-way handshaking*



Note:

***A SYN segment cannot carry data, but it consumes one sequence number.***

Note:

***A SYN + ACK segment cannot carry data, but does consume one sequence number.***

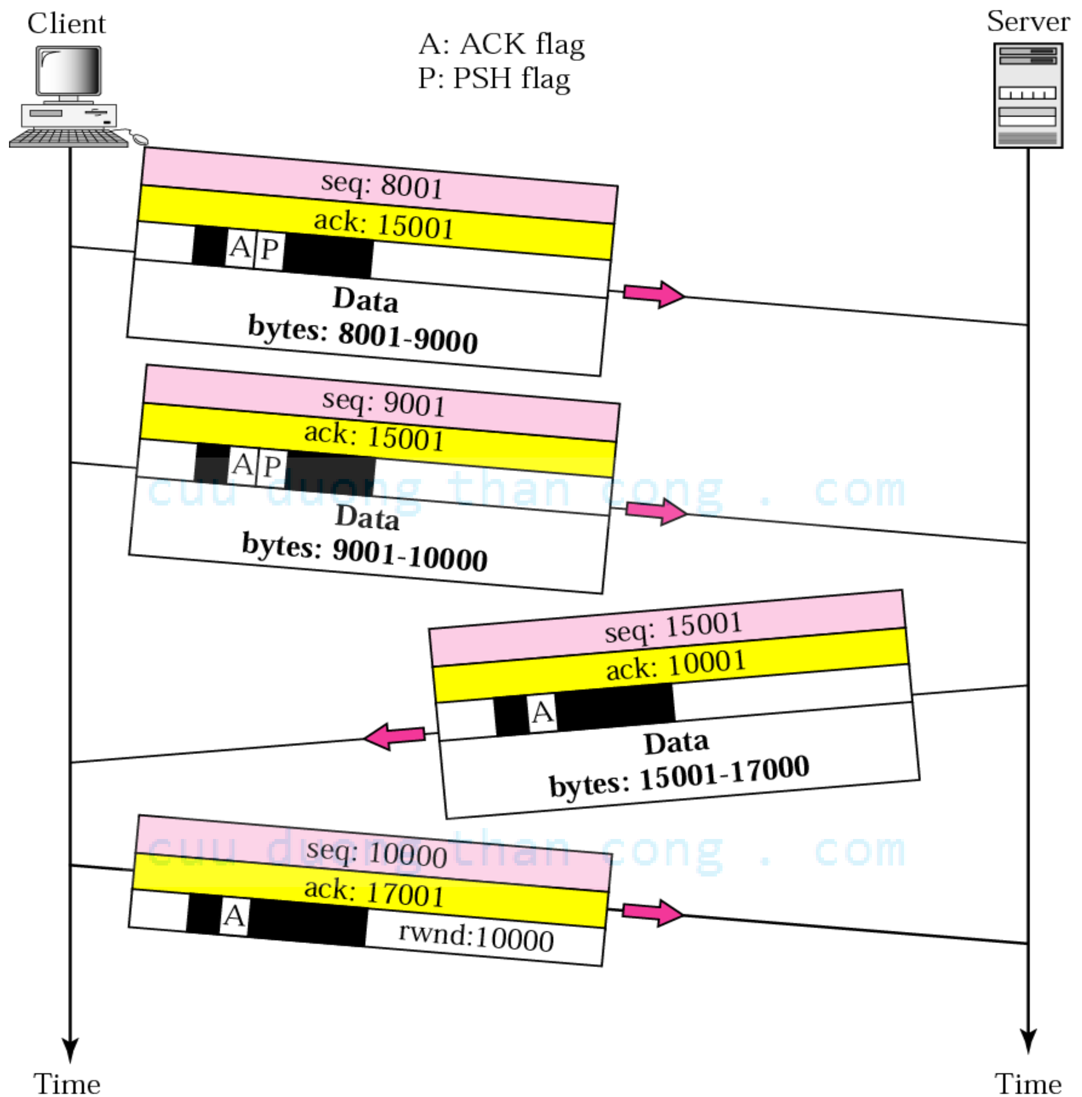
Note:

***An ACK segment, if carrying no data, consumes no sequence number.***



cuu duong than cong . com

## **Figure 12.10** *Data transfer*



Note:

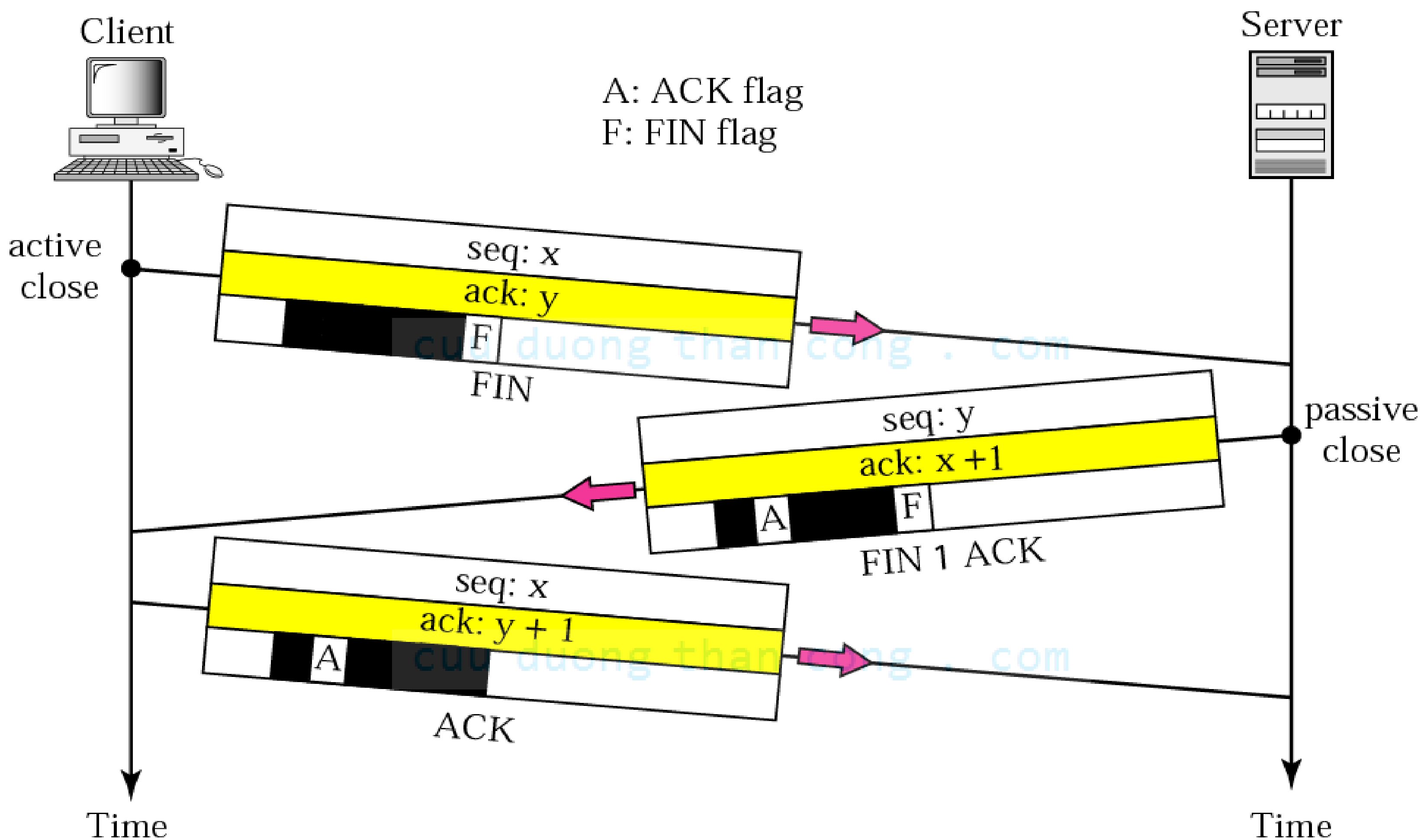
***The FIN segment consumes one sequence number if it does not carry data.***



cuu duong than cong . com

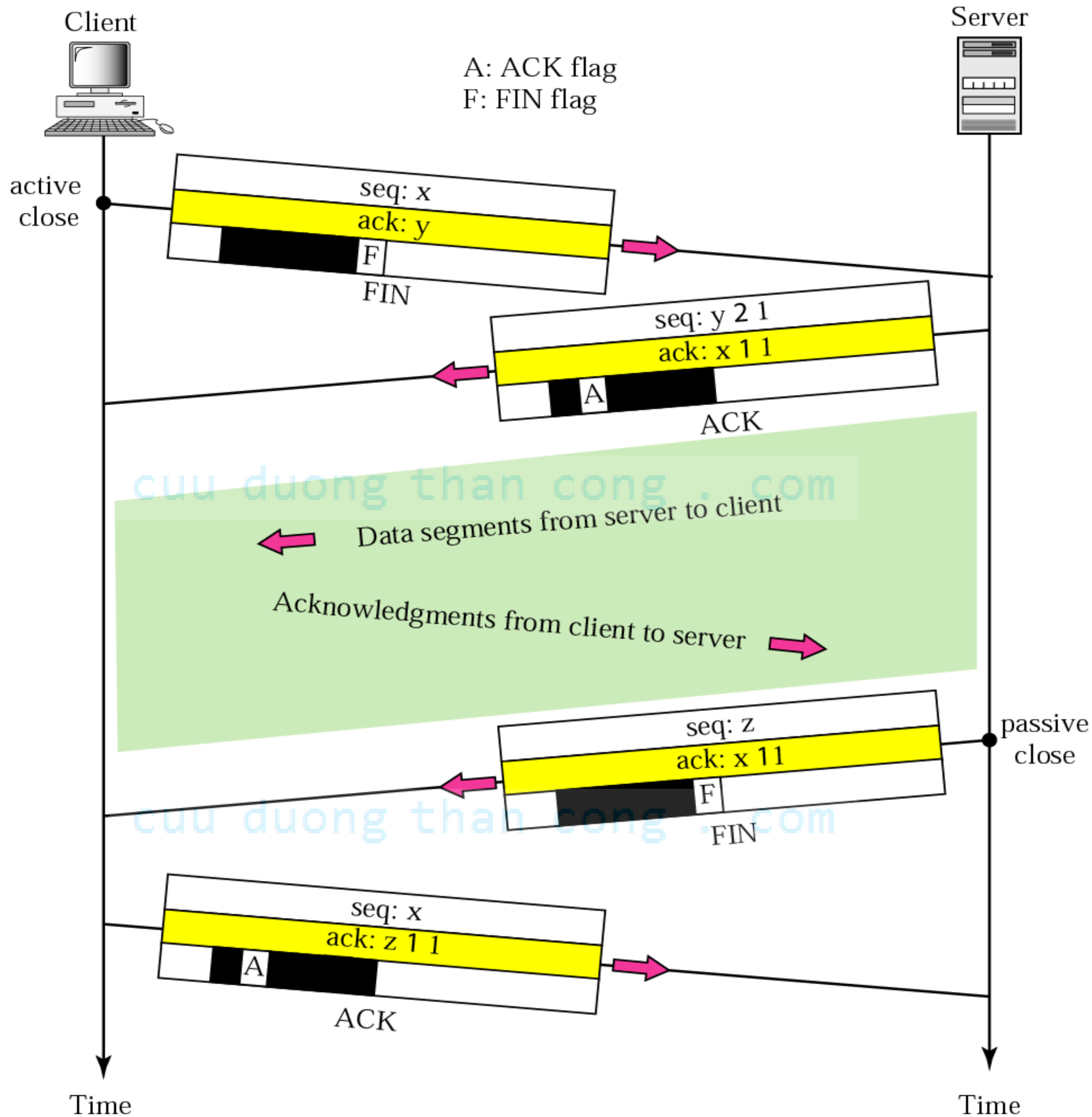
## **Figure 12.11** *Connection termination using three-way handshaking*



**Note:**

***The FIN + ACK segment consumes one sequence number if it does not carry data.***

# Figure 12.12 *Half-close*



# 12.5 STATE TRANSITION DIAGRAM

*To keep track of all the different events happening during connection establishment, connection termination, and data transfer, the TCP software is implemented as a finite state machine. .*



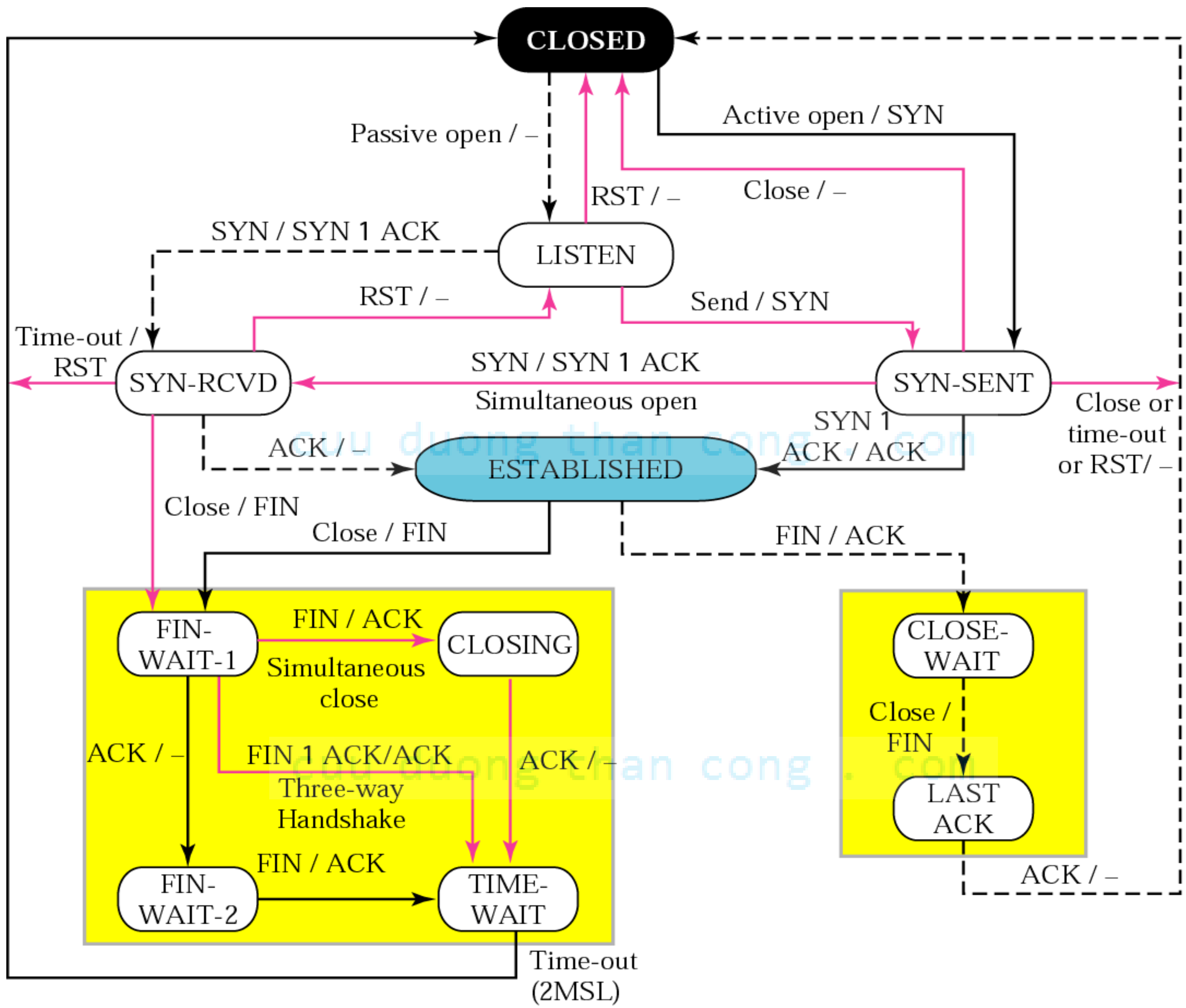
*The topics discussed in this section include:*

*Scenarios*

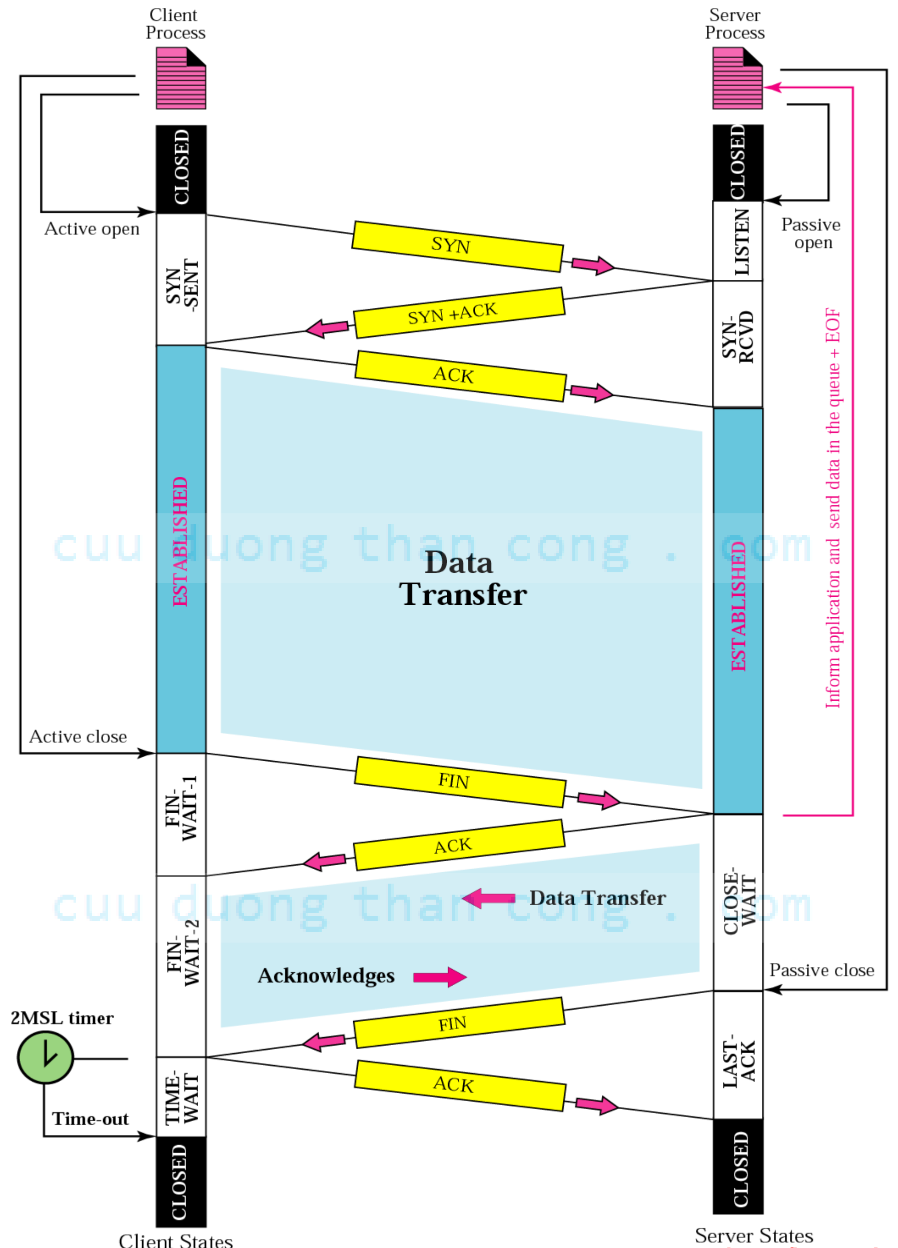
## *Table 12.3 States for TCP*

| *State* | *Description* |
|---|---|
| **CLOSED** | There is no connection |
| **LISTEN** | Passive open received; waiting for SYN |
| **SYN-SENT** | SYN sent; waiting for ACK |
| **SYN-RCVD** | SYN+ACK sent; waiting for ACK |
| **ESTABLISHED** | Connection established; data transfer in progress |
| **FIN-WAIT-1** | First FIN sent; waiting for ACK |
| **FIN-WAIT-2** | ACK to first FIN received; waiting for second FIN |
| **CLOSE-WAIT** | First FIN received, ACK sent; waiting for application to close |
| **TIME-WAIT** | Second FIN received, ACK sent; waiting for 2MSL time-out |
| **LAST-ACK** | Second FIN sent; waiting for ACK |
| **CLOSING** | Both sides have decided to close simultaneously |

# Figure 12.13 *State transition diagram*



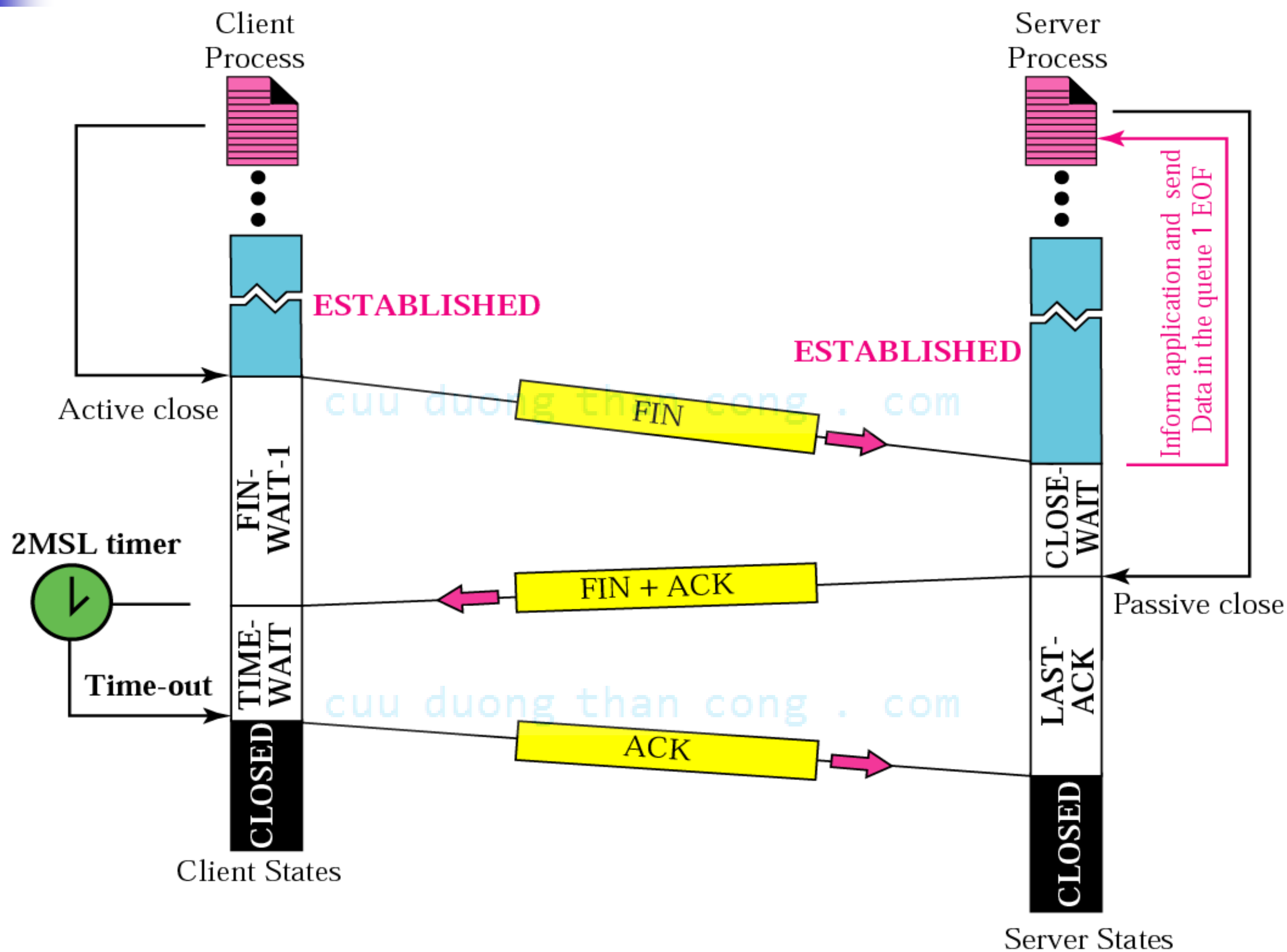
# Figure 12.14 *Common scenario*



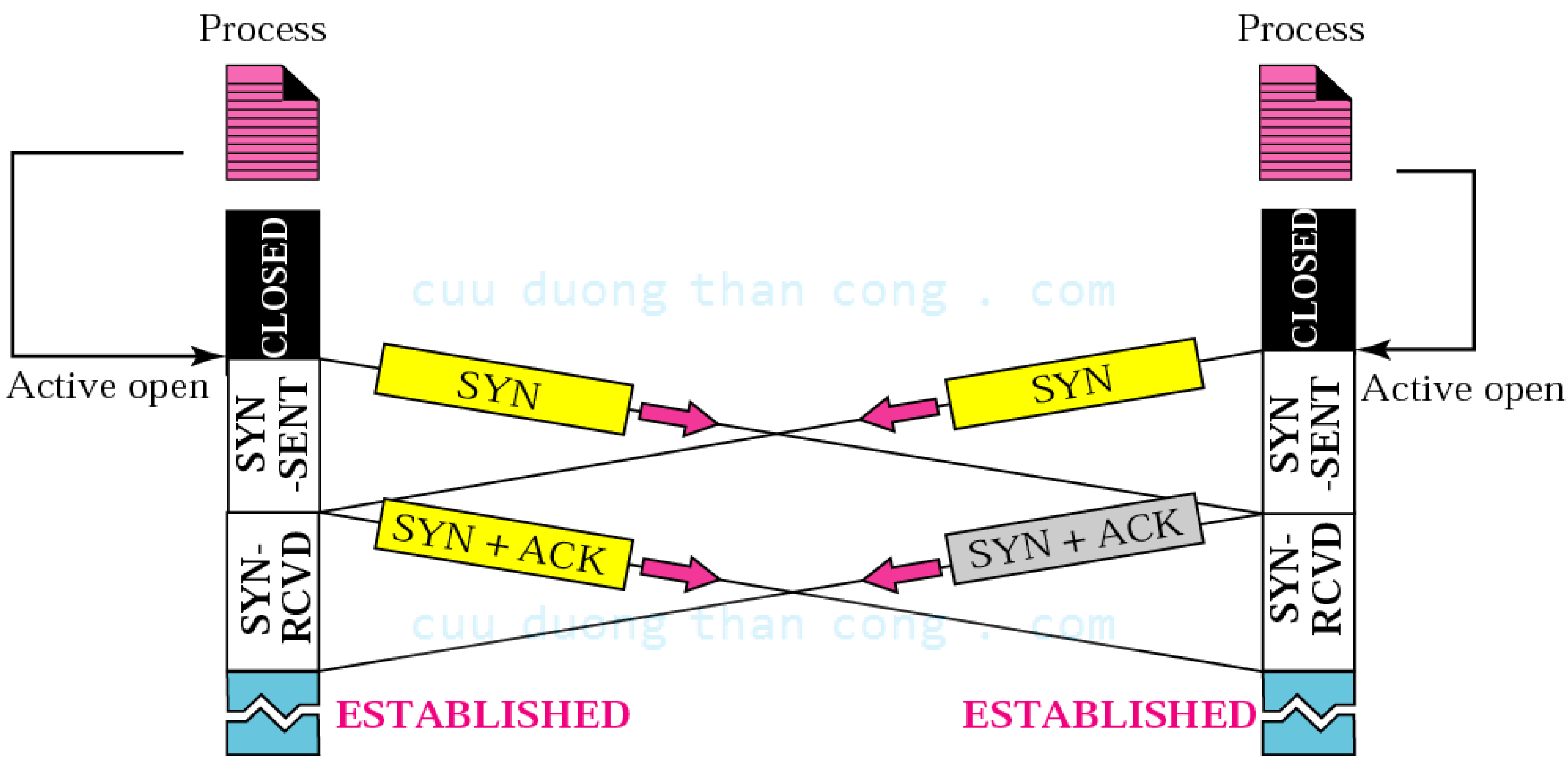
Note:

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

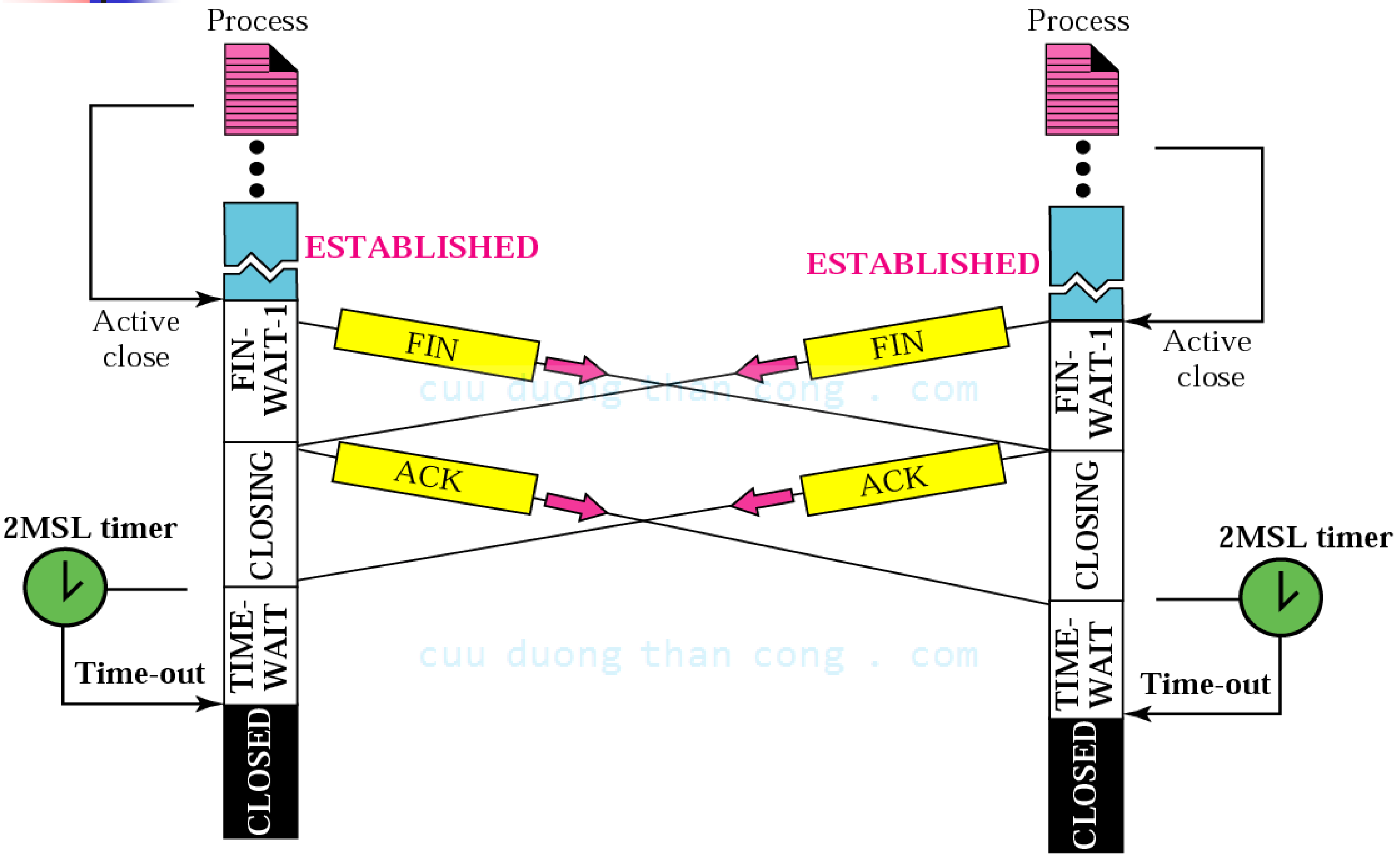
# Figure 12.15 *Three-way handshake*



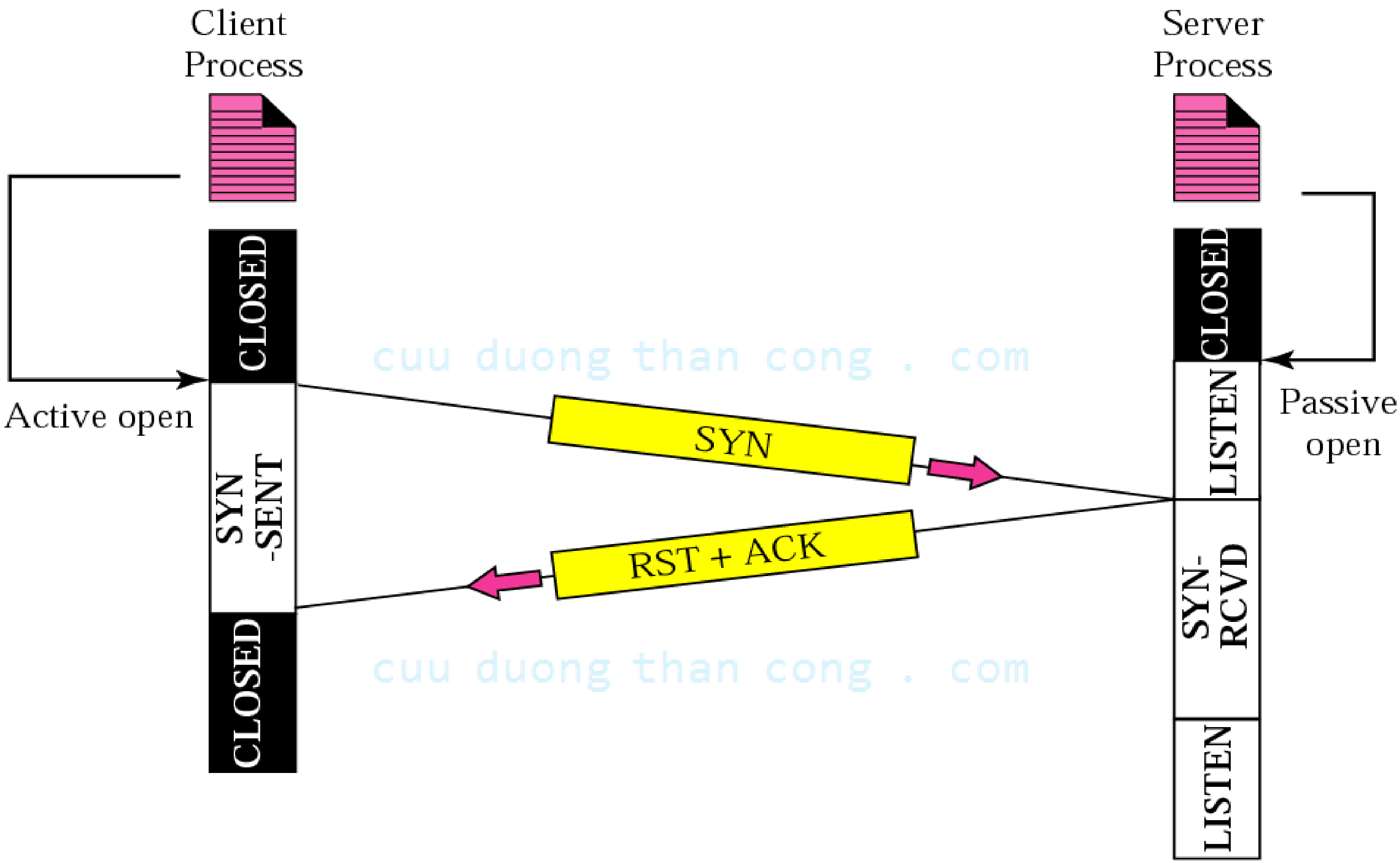
# Figure 12.16 *Simultaneous open*



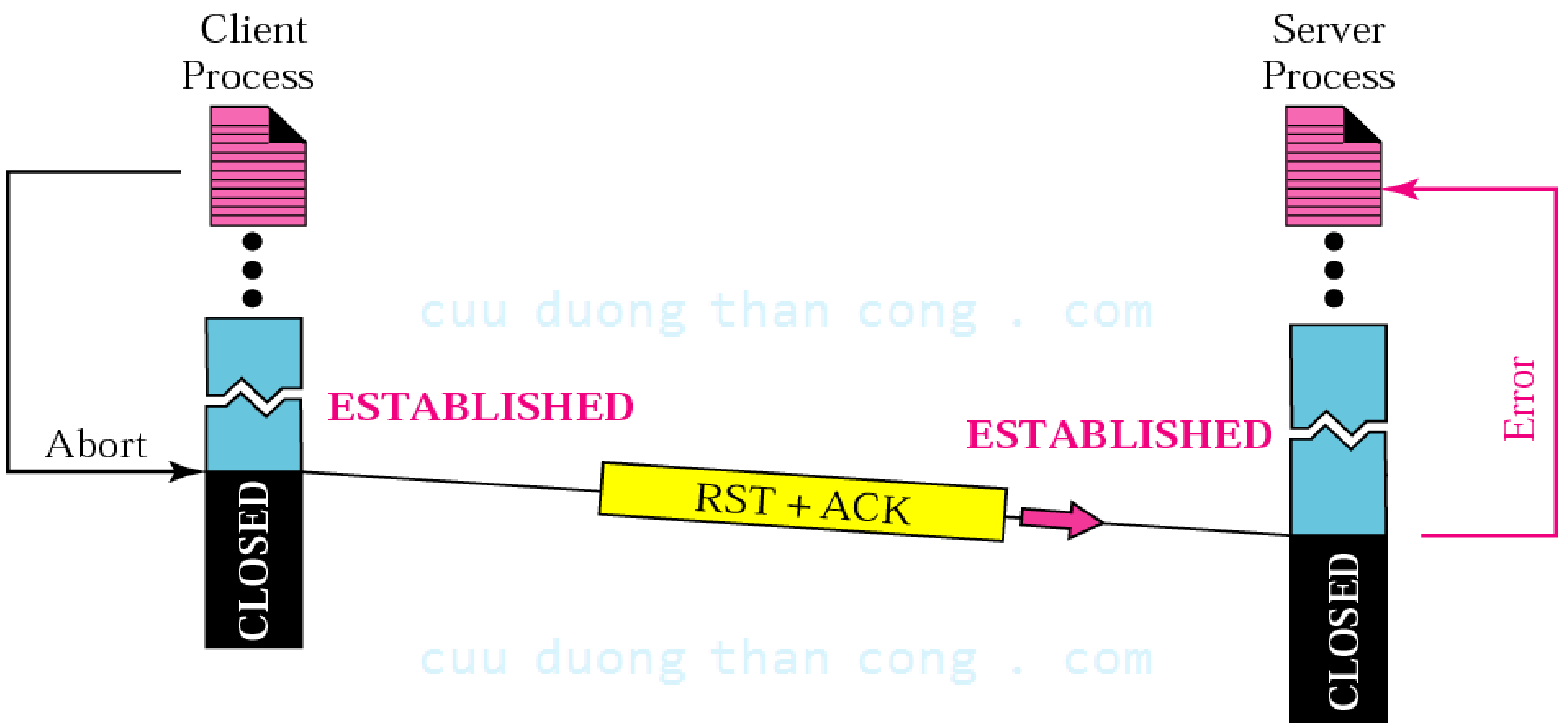
# Figure 12.17 *Simultaneous close*



# Figure 12.18 *Denying a connection*



# Figure 12.19 *Aborting a connection*



# 12.6 FLOW CONTROL

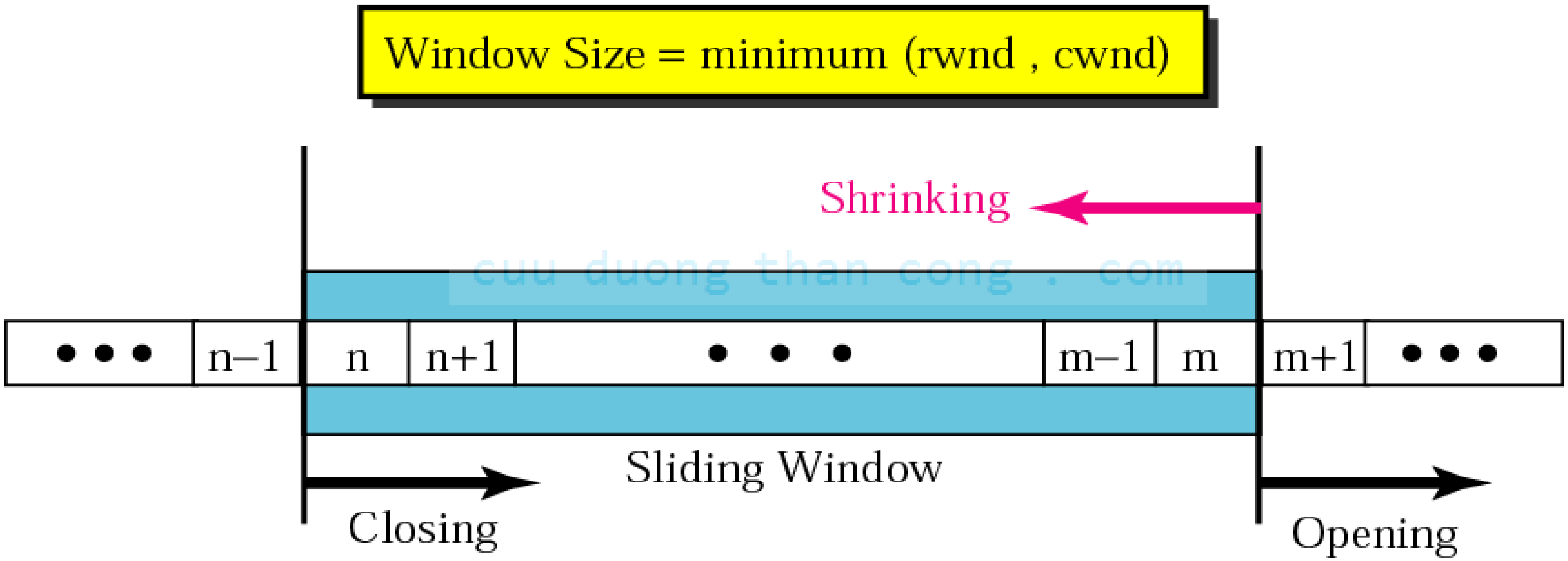
***Flow control regulates the amount of data a source can send before receiving an acknowledgment from the destination. TCP defines a window that is imposed on the buffer of data delivered from the application program.***



*The topics discussed in this section include:*

***Sliding Window Protocol***
***Silly Window Syndrome***

# Figure 12.20 *Sliding window*



Note:

***A sliding window is used to make transmission more efficient as well as to control the flow of data so that the destination does not become overwhelmed with data.***

***TCP's sliding windows are byte oriented.***

## Example 3

***What is the value of the receiver window (rwnd) for host A if the receiver, host B, has a buffer size of 5,000 bytes and 1,000 bytes of received and unprocessed data?***



### *Solution*

***The value of rwnd = 5,000 − 1,000 = 4,000. Host B can receive only 4,000 bytes of data before overflowing its buffer. Host B advertises this value in its next segment to A.***

## Example 4

***What is the size of the window for host A if the value of rwnd is 3,000 bytes and the value of cwnd is 3,500 bytes?***



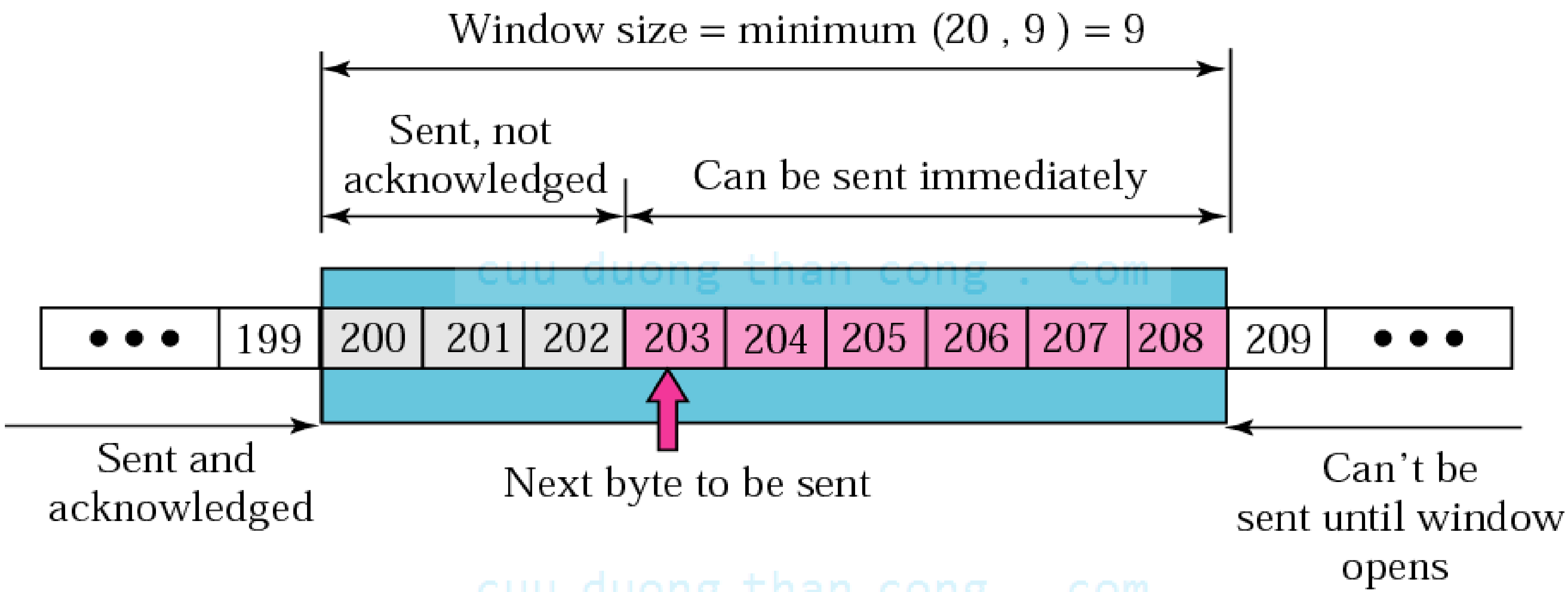
***Solution***

***The size of the window is the smaller of rwnd and cwnd, which is 3,000 bytes.***

# EXAMPLE 5

***Figure 12.21 shows an unrealistic example of a sliding window. The sender has sent bytes up to 202. We assume that cwnd is 20 (in reality this value is thousands of bytes). The receiver has sent an acknowledgment number of 200 with an rwnd of 9 bytes (in reality this value is thousands of bytes). The size of the sender window is the minimum of rwnd and cwnd or 9 bytes. Bytes 200 to 202 are sent, but not acknowledged. Bytes 203 to 208 can be sent without worrying about acknowledgment. Bytes 209 and above cannot be sent.***

# Figure 12.21 *Example 5*



## EXAMPLE 6

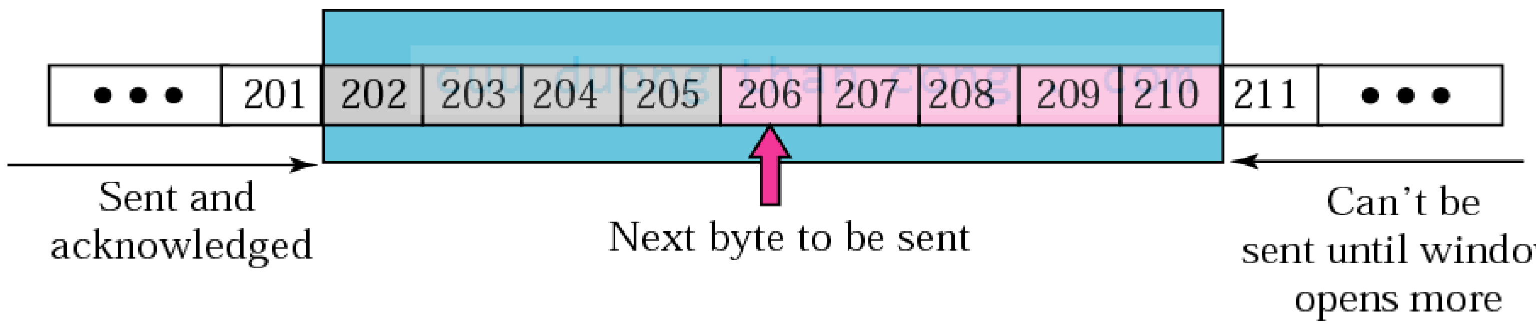
*In Figure 12.21 the server receives a packet with an acknowledgment value of 202 and an rwnd of 9. The host has already sent bytes 203, 204, and 205. The value of cwnd is still 20. Show the new window.*



### *Solution*

*Figure 12.22 shows the new window. Note that this is a case in which the window closes from the left and opens from the right by an equal number of bytes; the size of the window has not been changed. The acknowledgment value, 202, declares that bytes 200 and 201 have been received and the sender needs not worry about them; the window can slide over them.*

## **Figure 12.22** *Example 6*



# Example 7

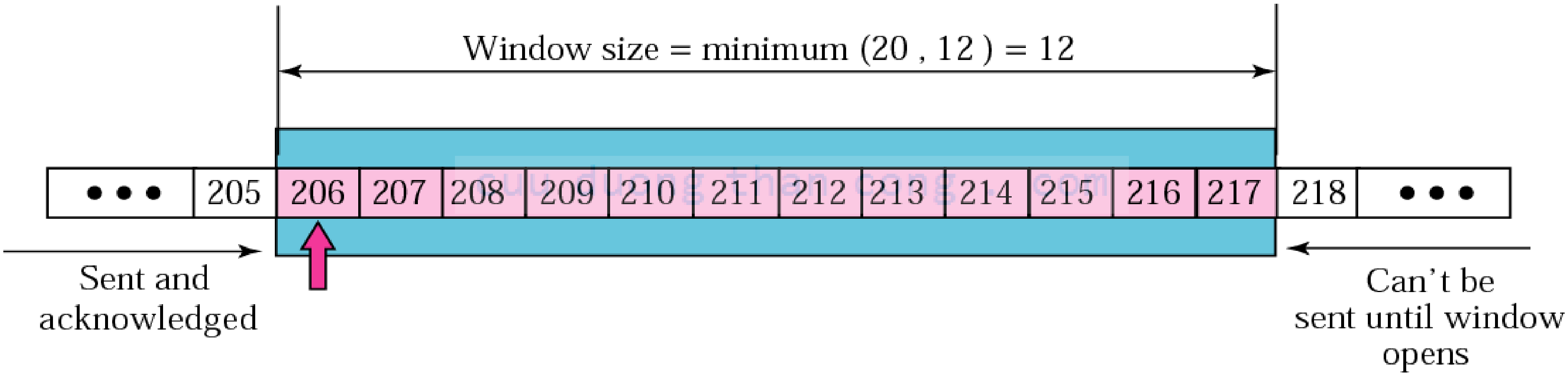
***In Figure 12.22 the sender receives a packet with an acknowledgment value of 206 and an rwnd of 12. The host has not sent any new bytes. The value of cwnd is still 20. Show the new window.***



## Solution

***The value of rwnd is less than cwnd, so the size of the window is 12. Figure 12.23 shows the new window. Note that the window has been opened from the right by 7 and closed from the left by 4; the size of the window has increased.***

# Figure 12.23 *Example 7*



# EXAMPLE 8

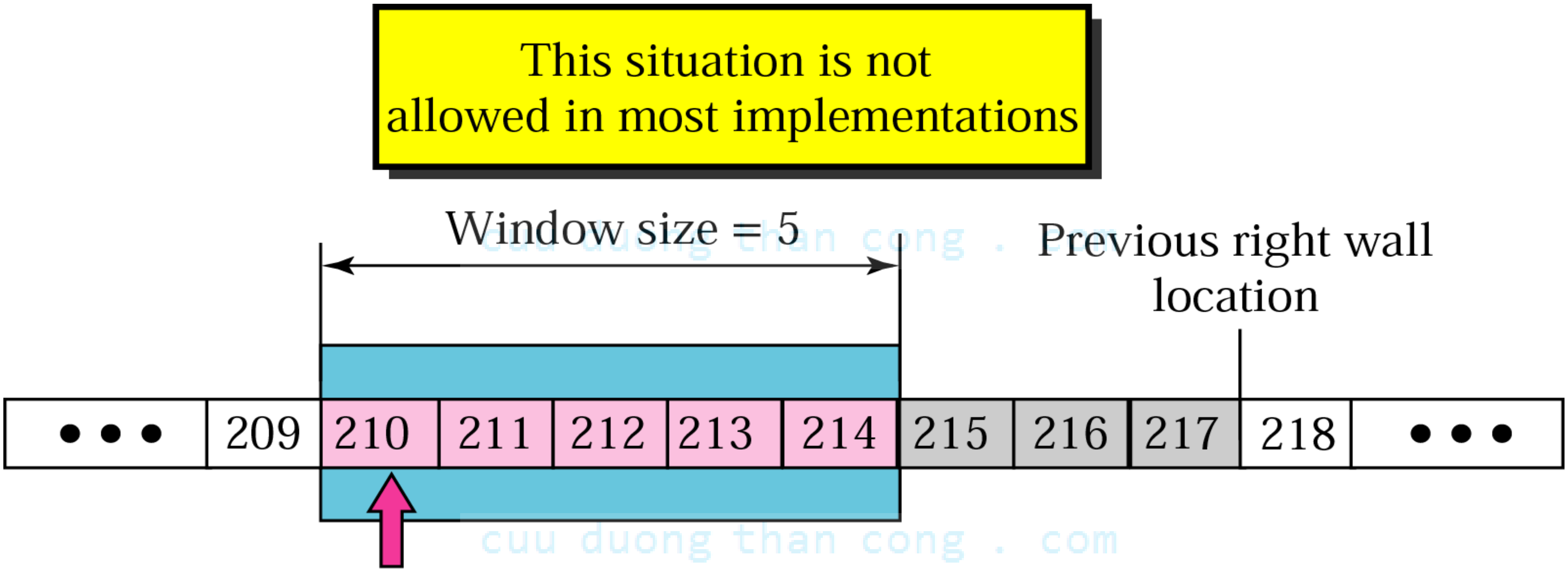
***In Figure 12.23 the host receives a packet with an acknowledgment value of 210 and an rwnd of 5. The host has sent bytes 206, 207, 208, and 209. The value of cwnd is still 20. Show the new window.***



## *Solution*

***The value of rwnd is less than cwnd, so the size of the window is 5. Figure 12.24 shows the situation. Note that this is a case not allowed by most implementations. Although the sender has not sent bytes 215 to 217, the receiver does not know this.***

# Figure 12.24 *Example 8*



# EXAMPLE 9

*How can the receiver avoid shrinking the window in the previous example?*

**Solution**

*The receiver needs to keep track of the last acknowledgment number and the last rwnd. If we add the acknowledgment number to rwnd we get the byte number following the right wall. If we want to prevent the right wall from moving to the left (shrinking), we must always have the following relationship.*



$$\text{new ack} + \text{new rwnd} \geq \text{last ack} + \text{last rwnd}$$

or

$$\text{new rwnd} \geq (\text{last ack} + \text{last rwnd}) - \text{new ack}$$

Note:

***To avoid shrinking the sender window, the receiver must wait until more space is available in its buffer.***

Note:

## *Some points about TCP's sliding windows:*

- ***The size of the window is the lesser of rwnd and cwnd.***
- ***The source does not have to send a full window's worth of data.*** 
- ***The window can be opened or closed by the receiver, but should not be shrunk.***
- ***The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.***



- ***The receiver can temporarily shut down the window; the sender, however, can always send a segment of one byte after the window is shut down.***

# 12.7 ERROR CONTROL

***TCP provides reliability using error control, which detects corrupted, lost, out-of-order, and duplicated segments. Error control in TCP is achieved through the use of the checksum, acknowledgment, and time-out.***

***The topics discussed in this section include:***

***Checksum***
***Acknowledgment***
***Acknowledgment Type***
***Retransmission***
***Out-of-Order Segments***
***Some Scenarios***

Note:

***ACK segments do not consume sequence numbers and are not acknowledged.***



cuu duong than cong . com

Note:

***In modern implementations, a retransmission occurs if the retransmission timer expires or three duplicate ACK segments have arrived.***

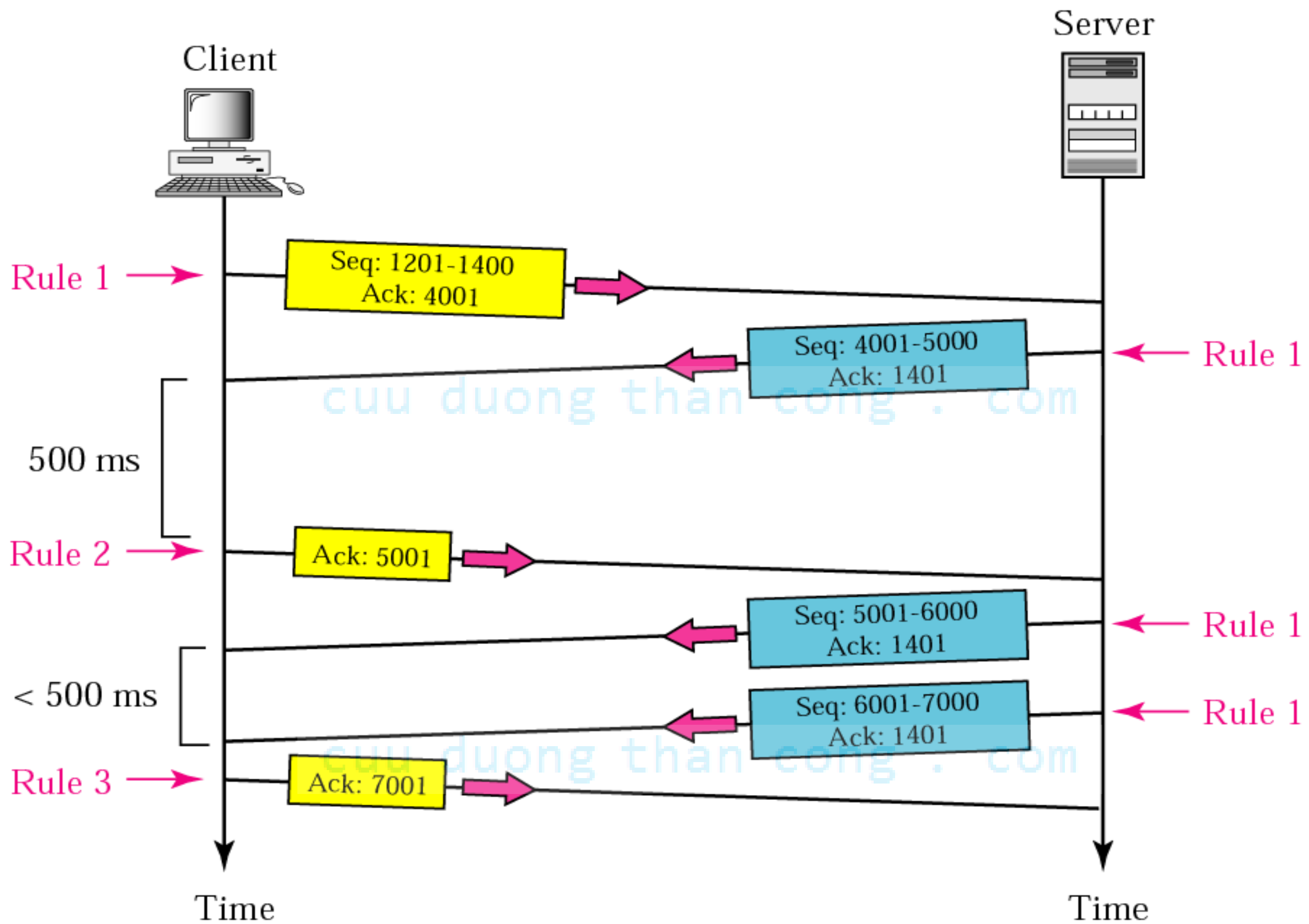
Note:

***No retransmission timer is set for an ACK segment.***

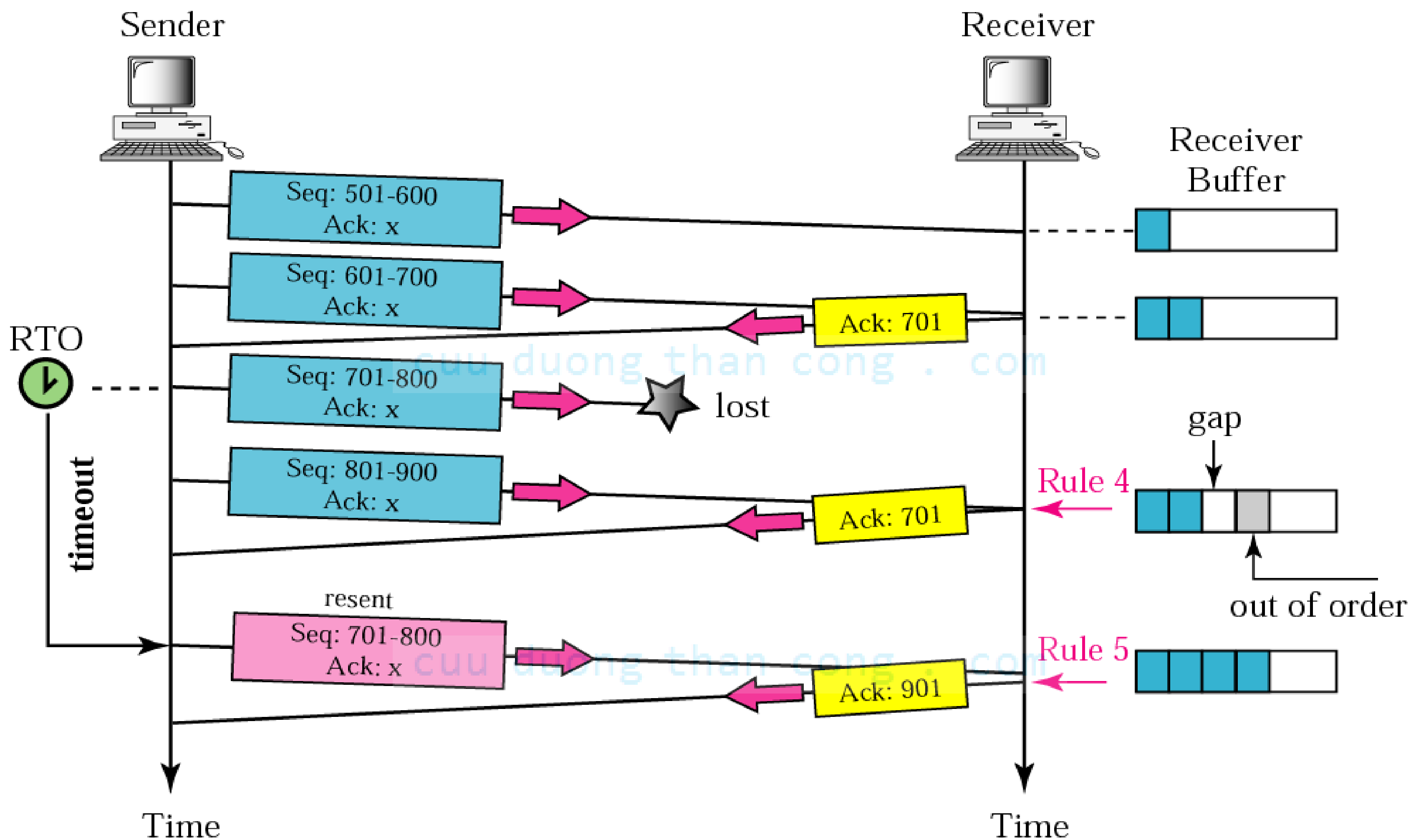
Note:

***Data may arrive out of order and be temporarily stored by the receiving TCP, but TCP guarantees that no out-of-order segment is delivered to the process.***

# Figure 12.25 *Normal operation*



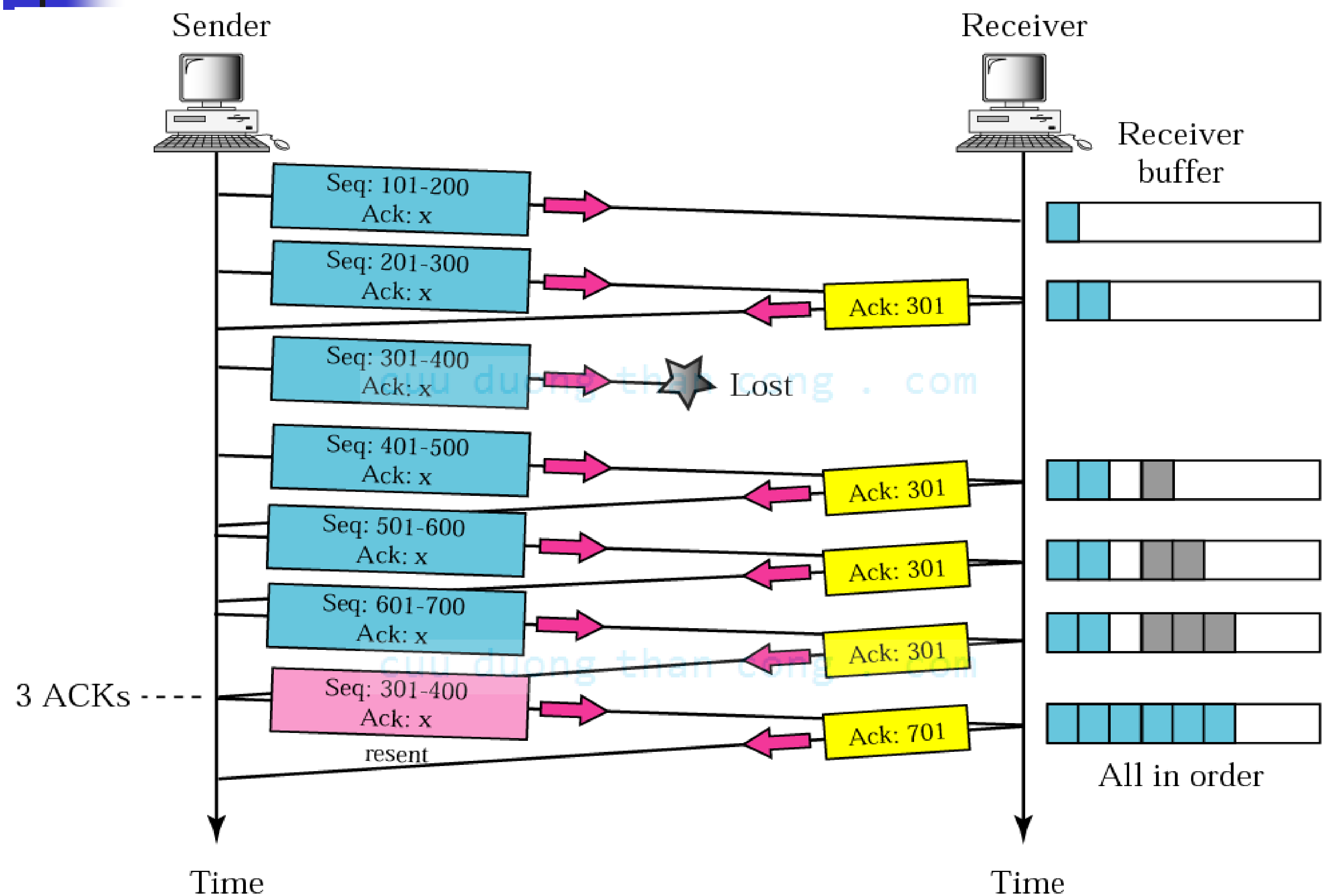
# Figure 12.26 *Lost segment*



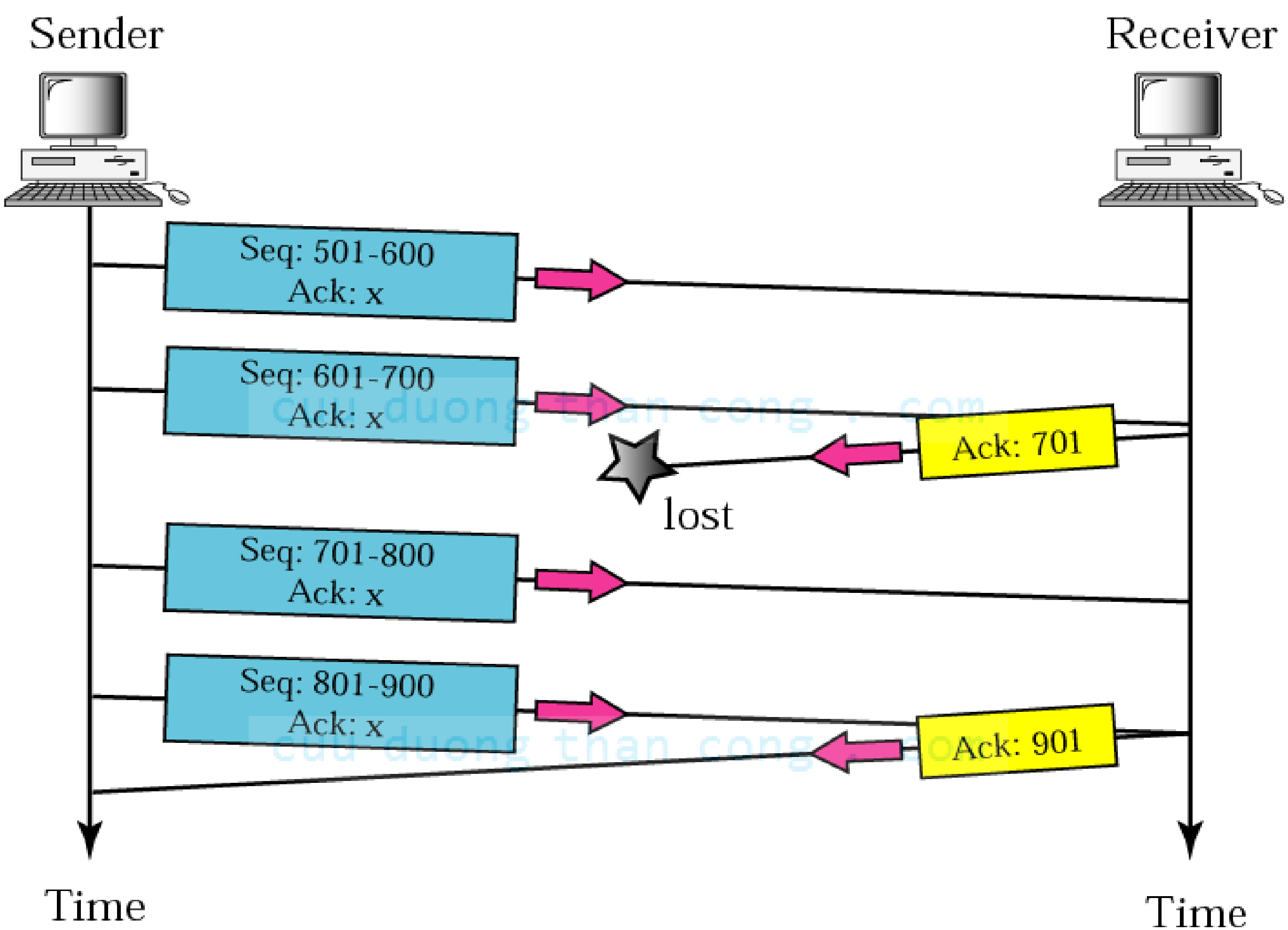
Note:

***The receiver TCP delivers only ordered data to the process.***

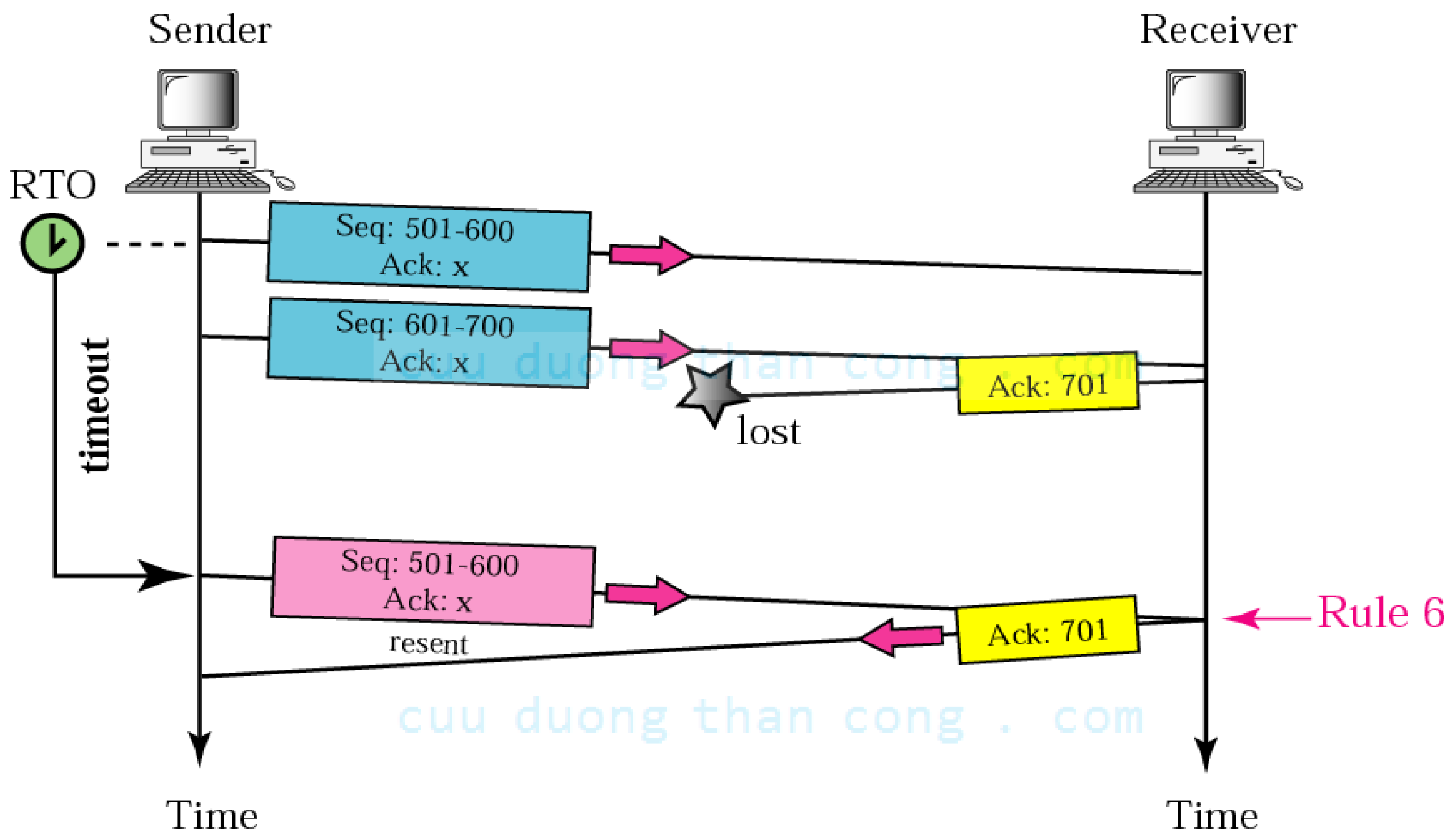
# Figure 12.27 *Fast retransmission*



## Figure 12.28 *Lost acknowledgment*



# Figure 12.29 *Lost acknowledgment corrected by resending a segment*



Note:

***Lost acknowledgments may create deadlock if they are not properly handled.***

# 12.8 CONGESTION CONTROL

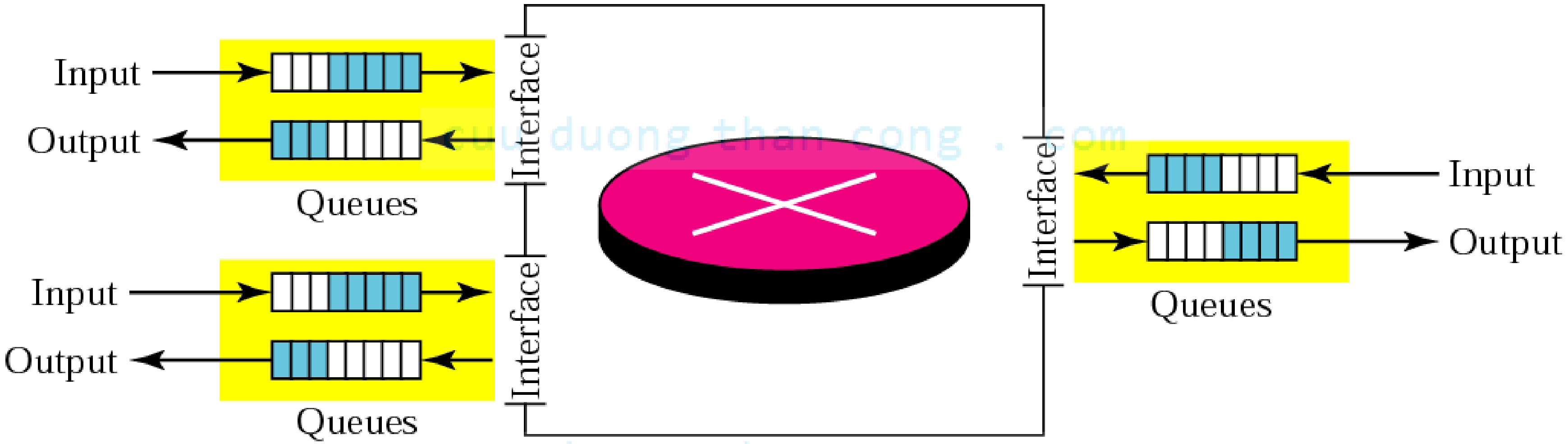
***Congestion control refers to the mechanisms and techniques to keep the load below the capacity.***



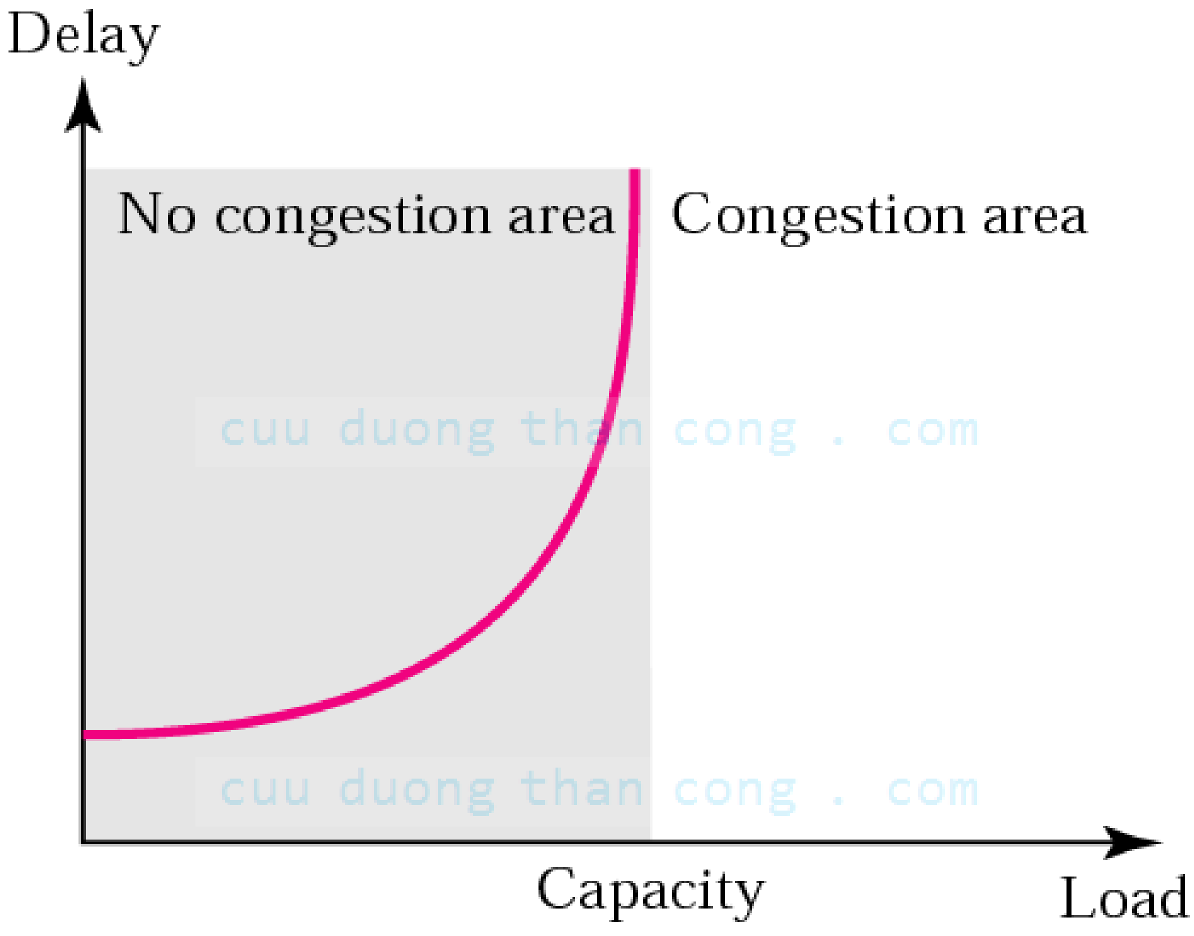
***The topics discussed in this section include:***

***Network Performance***
***Congestion Control Mechanisms***
***Congestion Control in TCP***

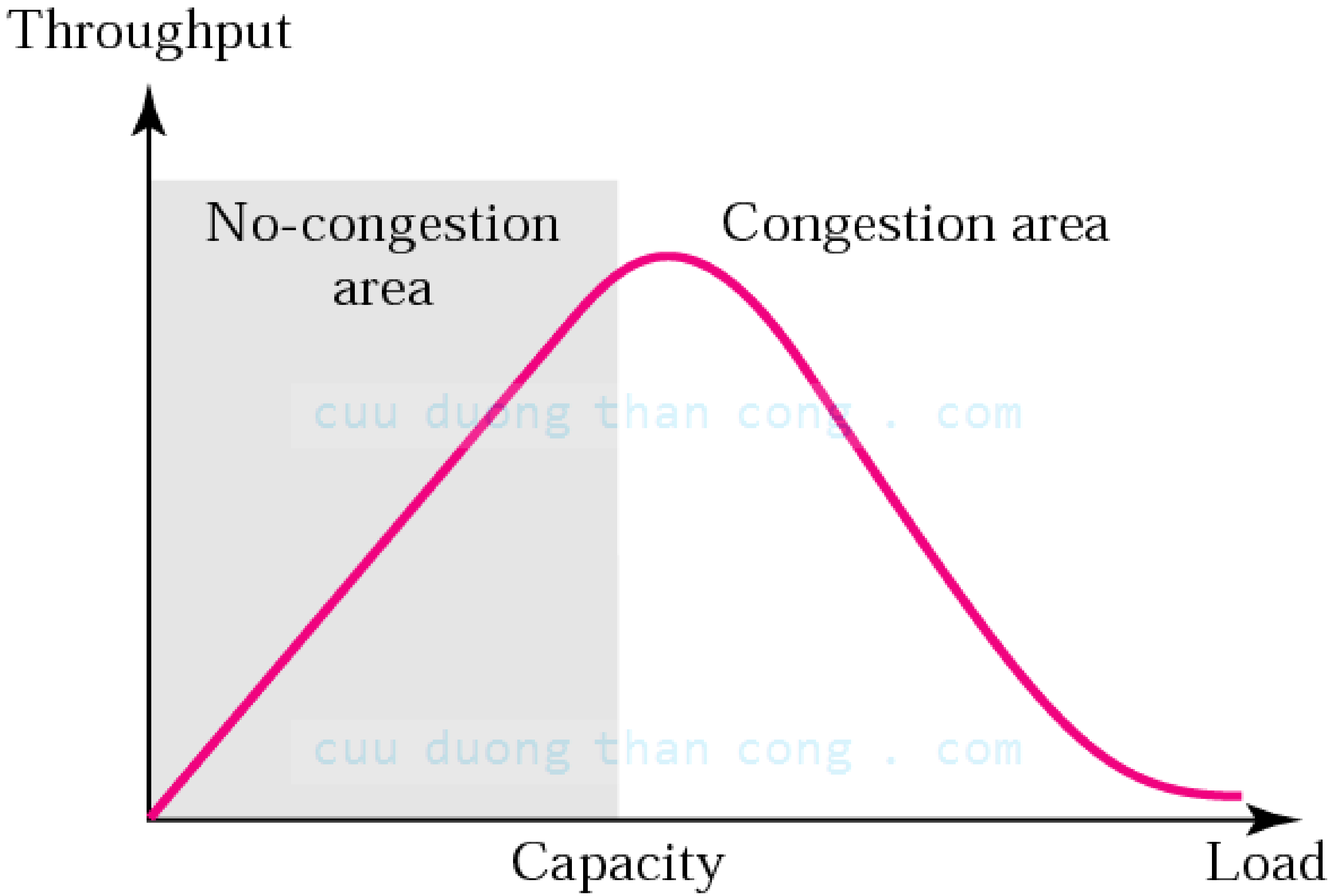
## **Figure 12.30** *Router queues*



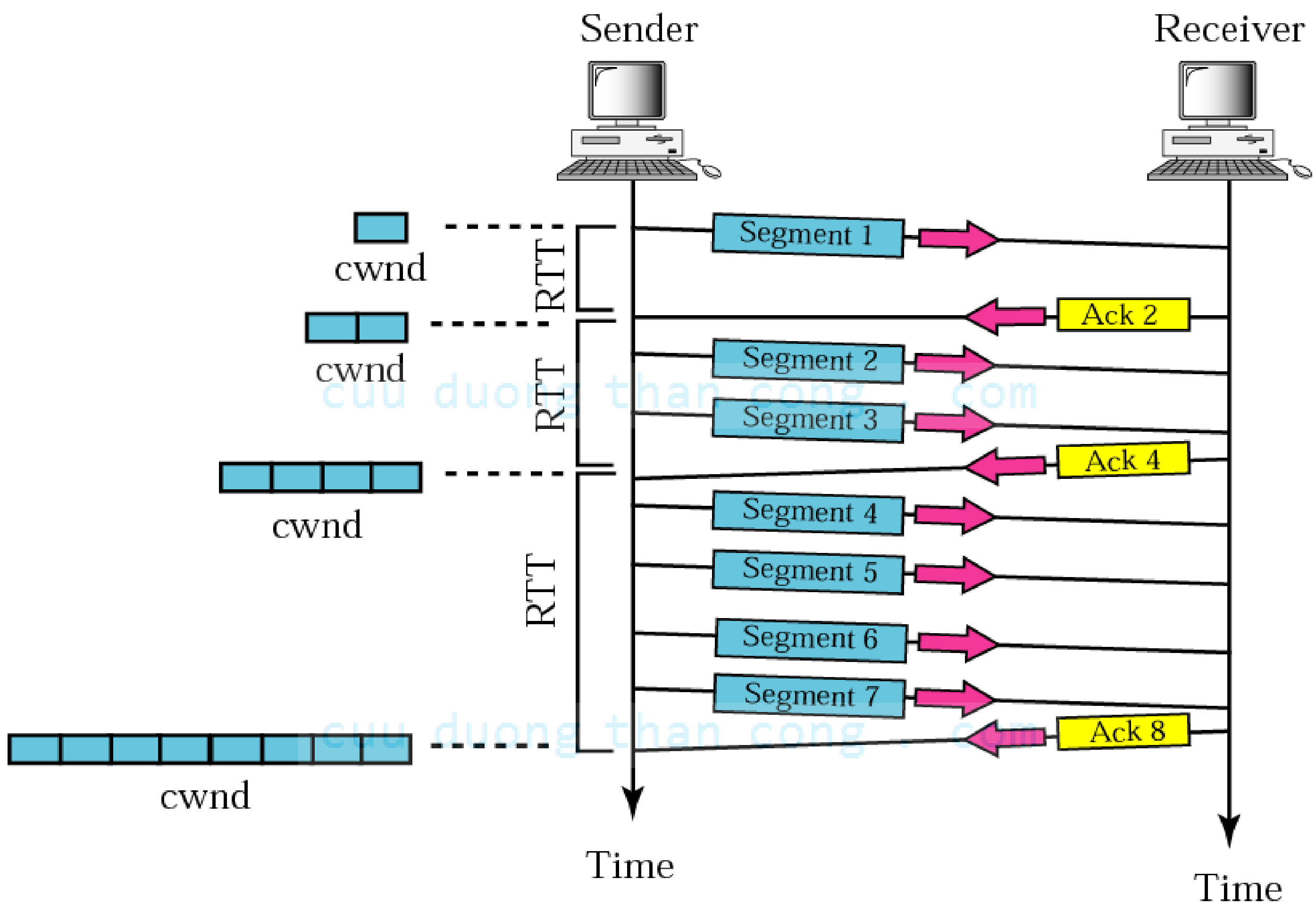
## **Figure 12.31** *Packet delay and network load*



## **Figure 12.32** *Throughput versus network load*



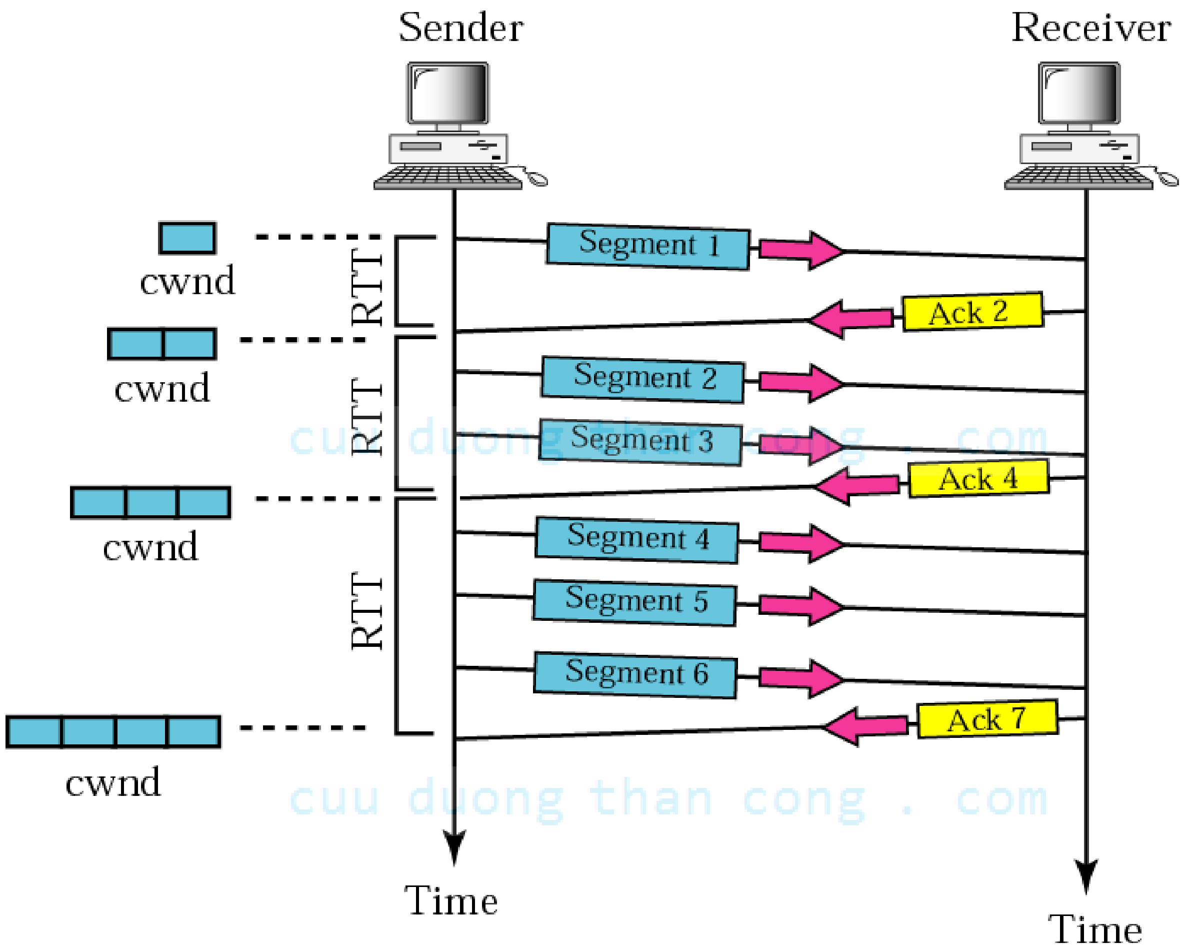
## Figure 12.33 *Slow start, exponential increase*



Note:

***In the slow start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.***

# **Figure 12.34** *Congestion avoidance, additive increase*



Note:

***In the congestion avoidance algorithm the size of the congestion window increases additively until congestion is detected.***

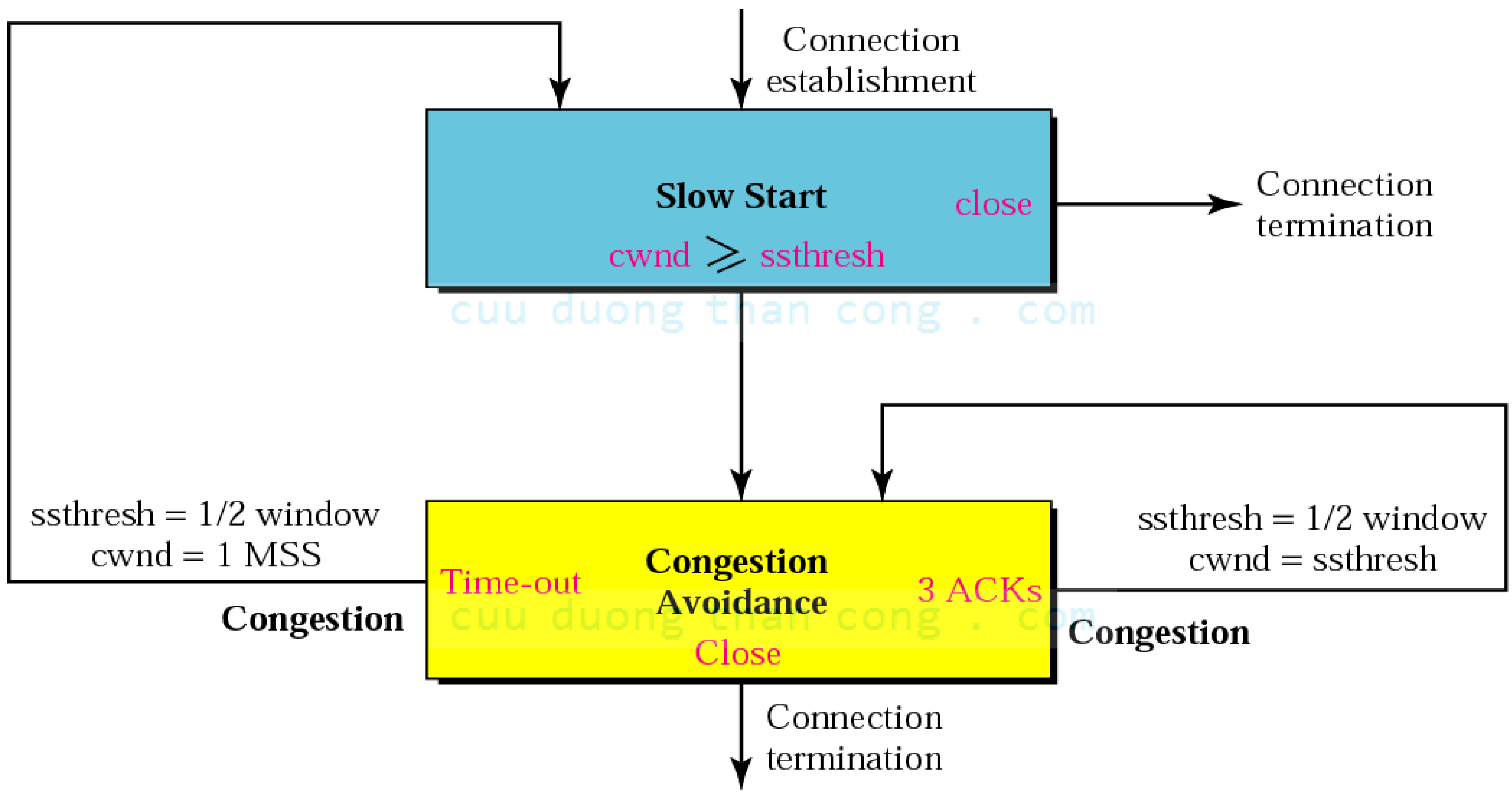
Note:

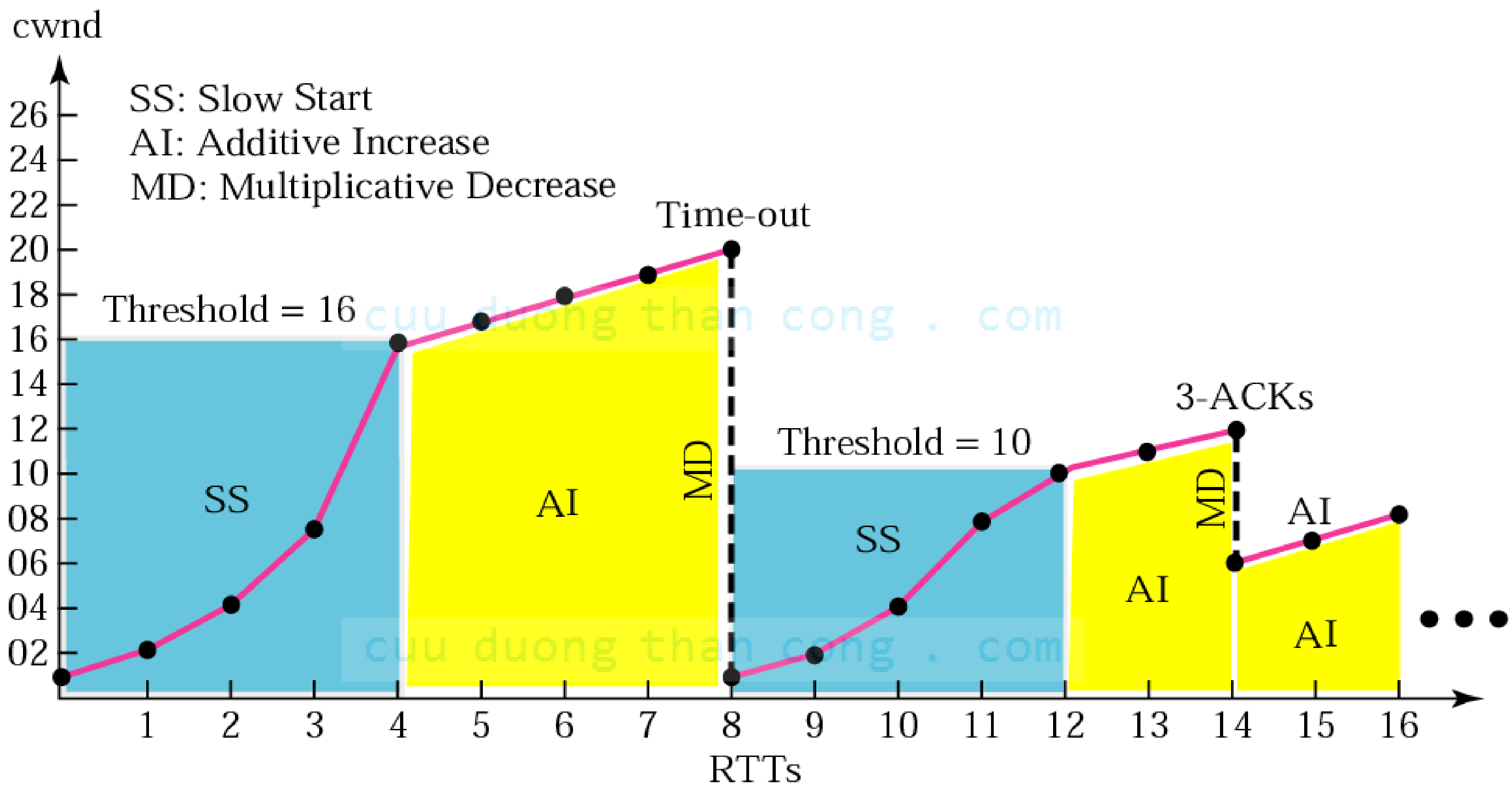
***Most implementations react differently to congestion detection:***



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

# Figure 12.35 *TCP congestion policy summary*




cwnd
SS: Slow Start
AI: Additive Increase
MD: Multiplicative Decrease
Time-out
Threshold = 16
Threshold = 10
3-ACKs
SS
AI
MD
SS
AI
MD
AI
AI
26
24
22
20
18
16
14
12
10
08
06
04
02
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
RTTs



# 12.9 TCP TIMERS

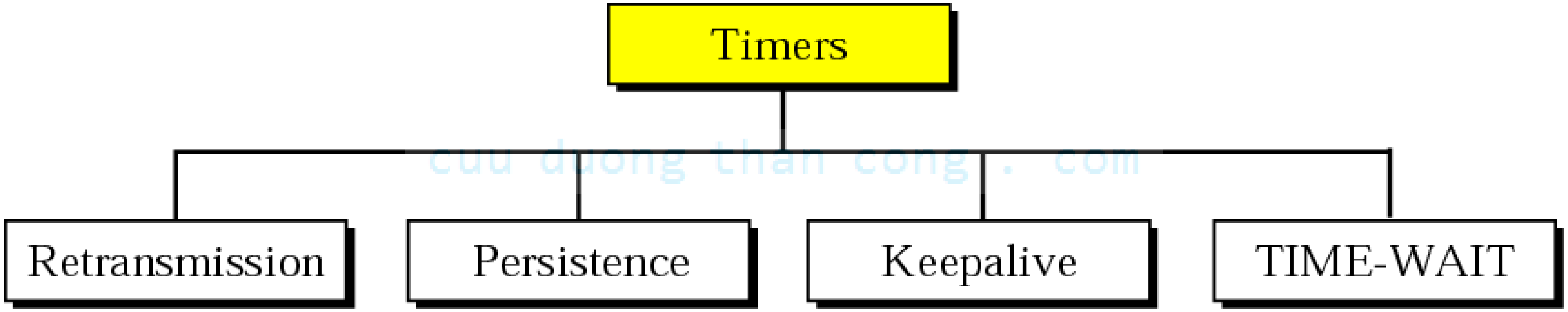
***To perform its operation smoothly, most TCP implementations use at least four timers.***



***The topics discussed in this section include:***

***Retransmission Timer***
***Persistence Timer***
***Keepalive Timer***
***TIME-WAIT Timer***

## **Figure 12.37** *TCP timers*



Note:

***In TCP, there can be only be one RTT measurement in progress at any time.***

# EXAMPLE 10

***Let us give a hypothetical example. Figure 12.38 shows part of a connection. The figure shows the connection establishment and part of the data transfer phases.***

***1. When the SYN segment is sent, there is no value for $RTT_M$, $RTT_S$, or $RTT_D$. The value of RTO is set to 6.00 seconds. The following shows the value of these variables at this moment:***



| | |
|---|---|
| $RTT_M = 1.5$ | $RTT_S = 1.5$ |
| $RTT_D = 1.5 / 2 = 0.75$ | $RTO = 1.5 + 4 . 0.75 = 4.5$ |



***2. When the SYN+ACK segment arrives, $RTT_M$ is measured and is equal to 1.5 seconds. The next slide shows the values of these variables:***

# Example 10 (continued)

$RTT_M = 1.5$ $RTT_S = 1.5$

$RTT_D = 1.5 / 2 = 0.75$ $RTO = 1.5 + 4 \cdot 0.75 = 4.5$

**3.** ***When the first data segment is sent, a new RTT measurement starts. Note that the sender does not start an RTT measurement when it sends the ACK segment, because it does not consume a sequence number and there is no time-out. No RTT measurement starts for the second data segment because a measurement is already in progress.***



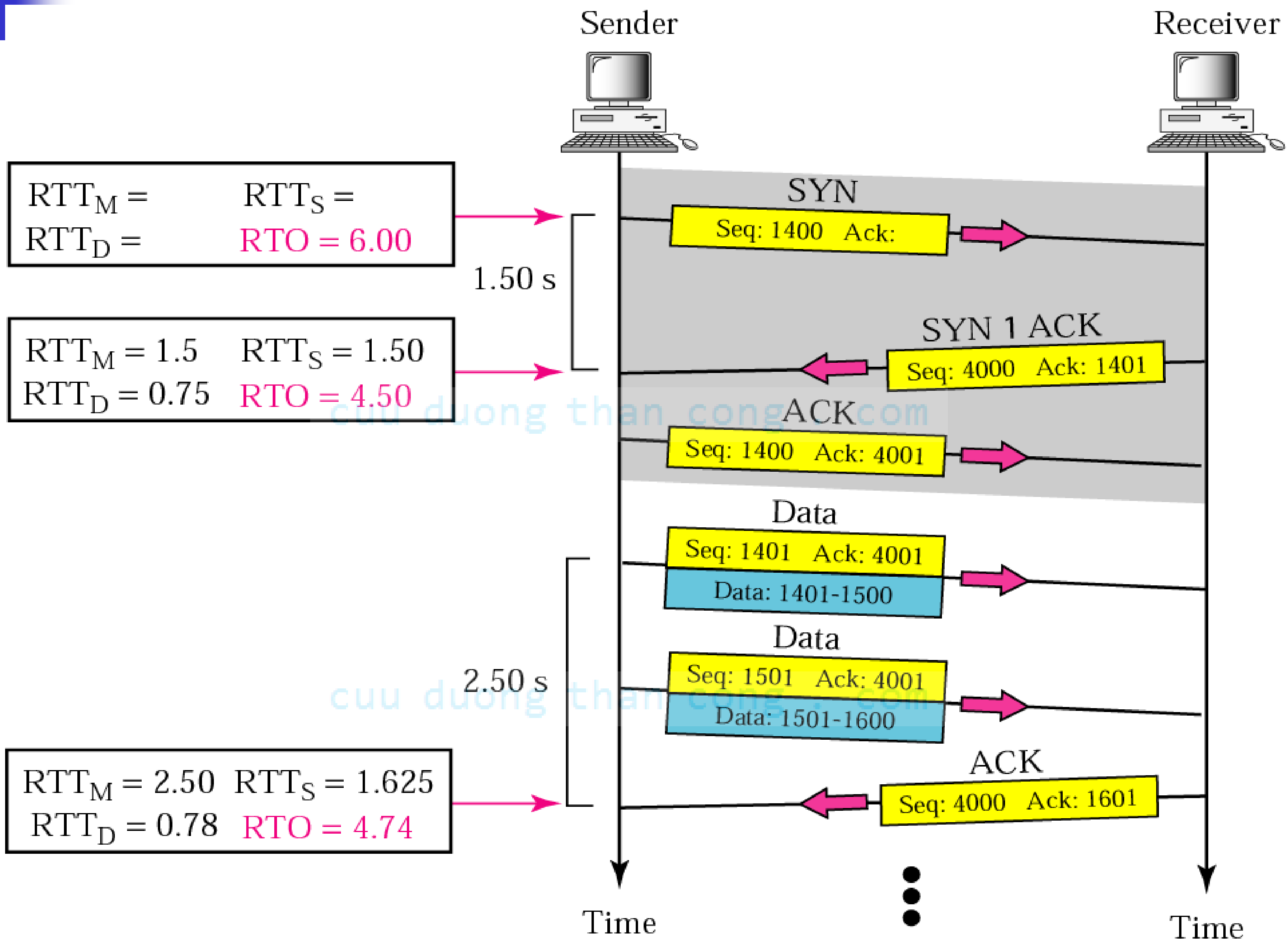
$RTT_M = 2.5$

$RTT_S = 7/8\ (1.5) + 1/8\ (2.5) = 1.625$

$RTT_D = 3/4\ (7.5) + 1/4\ |1.625 - 2.5| = 0.78$

$RTO = 1.625 + 4\ (0.78) = 4.74$

## **Figure 12.38** *Example 10*



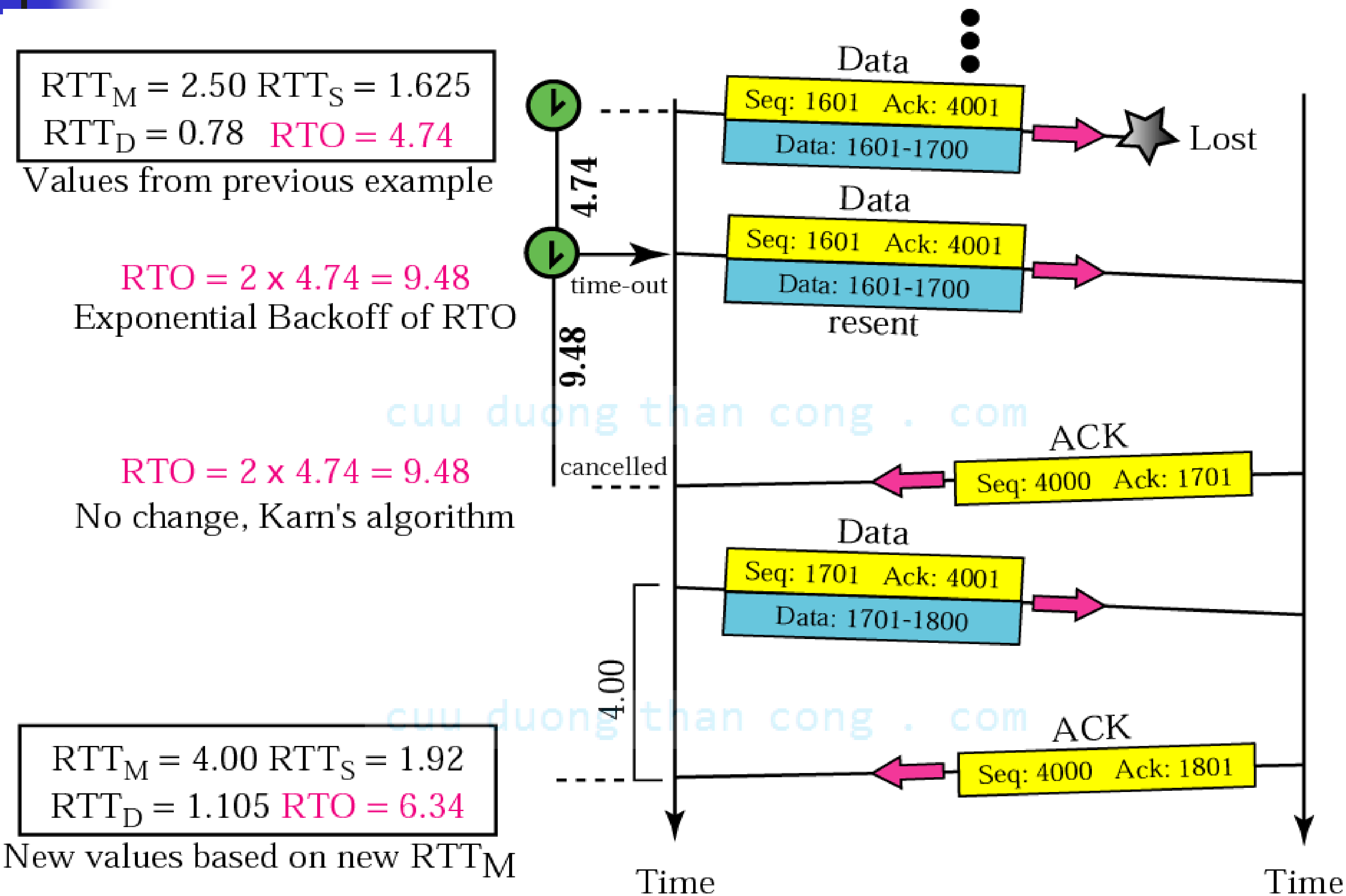
Note:

***TCP does not consider the RTT of a retransmitted segment in its calculation of a new RTO.***

# EXAMPLE 11

***Figure 12.39 is a continuation of the previous example. There is retransmission and Karn's algorithm is applied. The first segment in the figure is sent, but lost. The RTO timer expires after 4.74 seconds. The segment is retransmitted and the timer is set to 9.48, twice the previous value of RTO. This time an ACK is received before the time-out. We wait until we send a new segment and receive the ACK for it before recalculating the RTO (Karn's algorithm).***

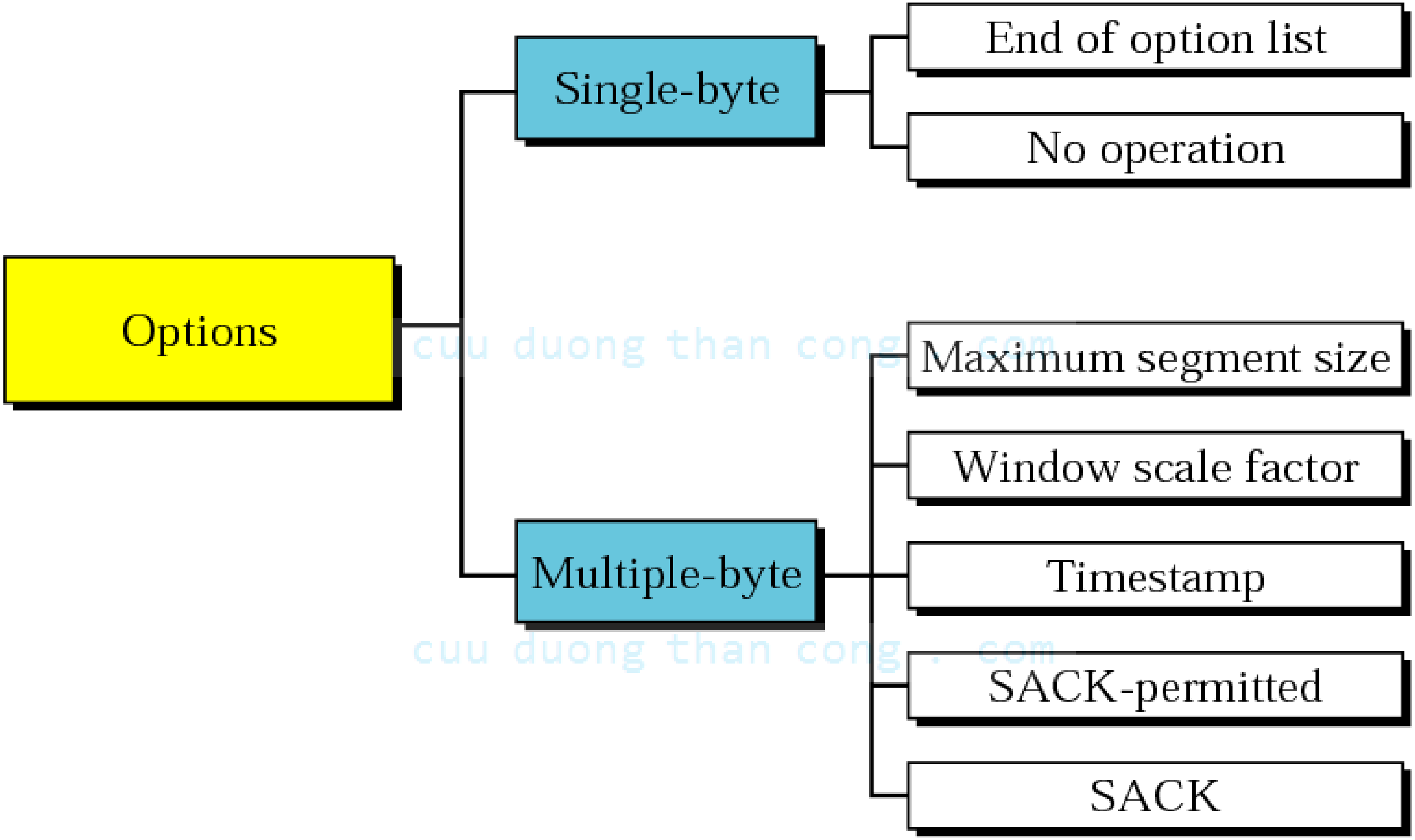
## **Figure 12.39** *Example 11*



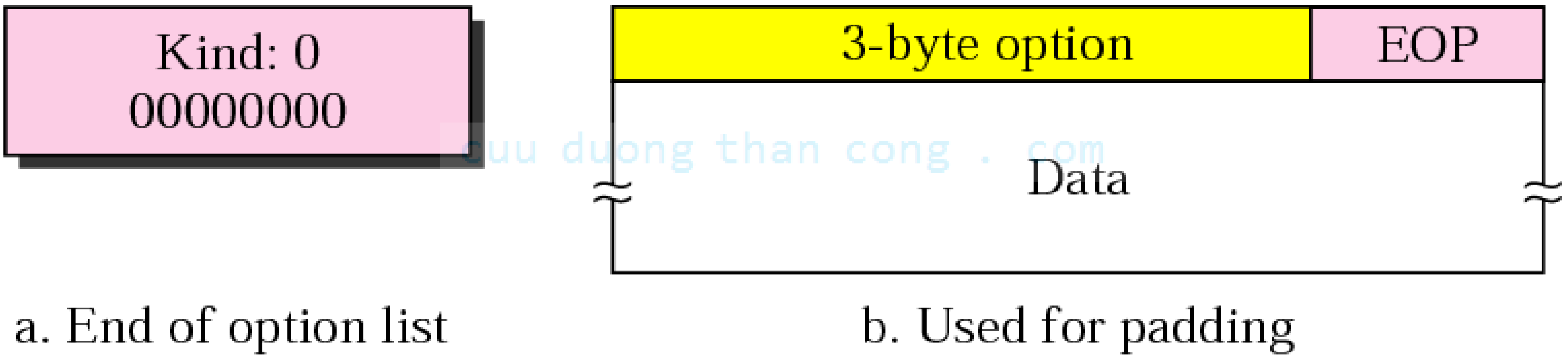
# 12.10 OPTIONS

***The TCP header can have up to 40 bytes of optional information. Options convey additional information to the destination or align other options.***

# Figure 12.40 *Options*



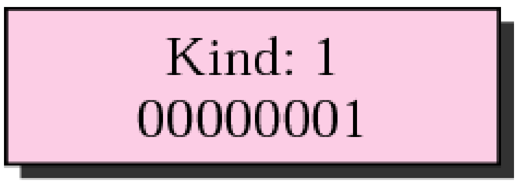
## **Figure 12.41** *End-of-option option*



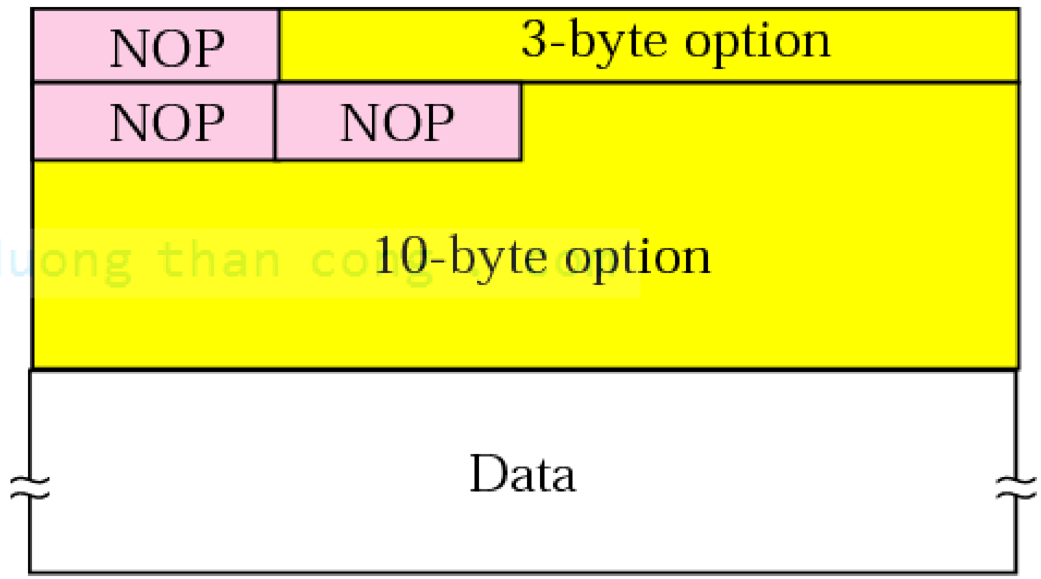
Note:

***EOP can be used only once.***

# Figure 12.42 *No-operation option*




a. No operation option




b. Used to align beginning of an option

Note:

***NOP can be used more than once.***

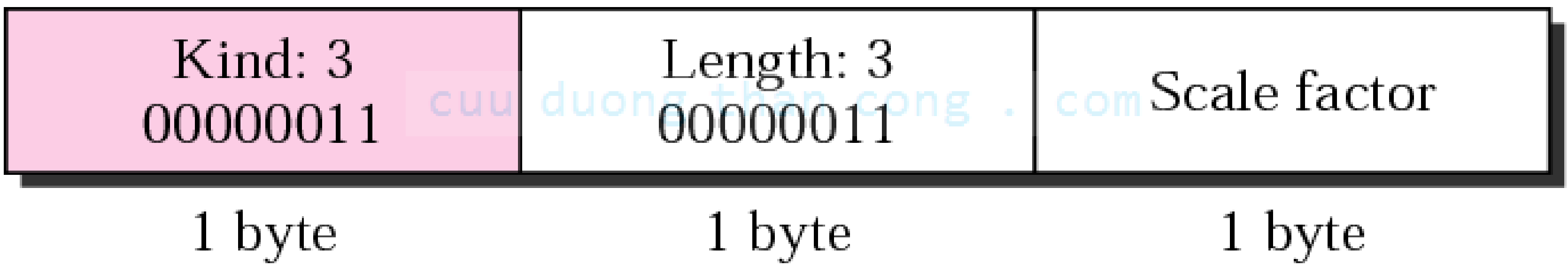
# Figure 12.43 *Maximum-segment-size option*

| Kind: 2<br>00000010 | Length: 4<br>00000100 | Maximum segment size |
|---|---|---|
| 1 byte | 1 byte | 2 bytes |

Note:

***The value of MSS is determined during connection establishment and does not change during the connection.***

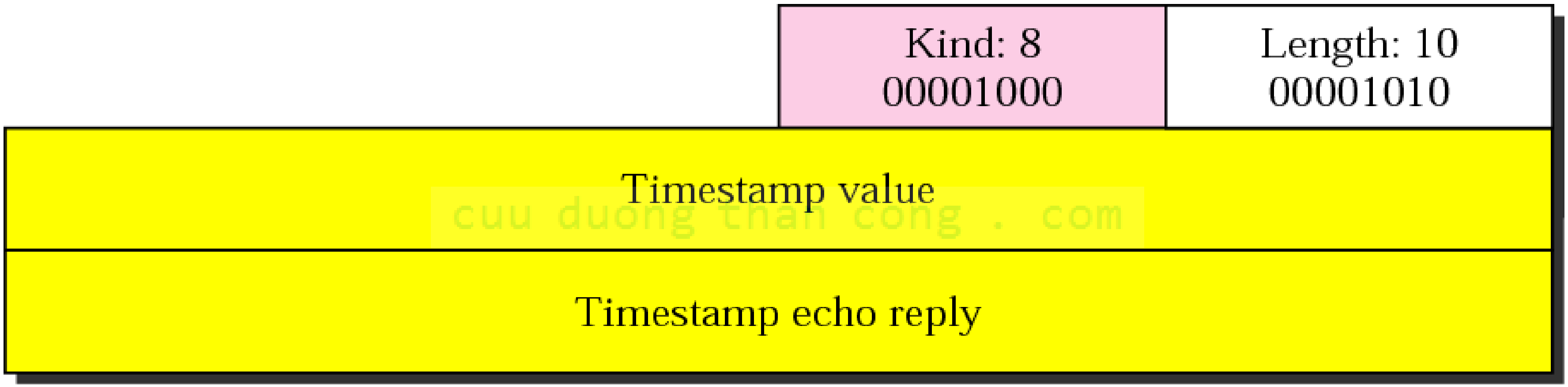
# **Figure 12.44** *Window-scale-factor option*



Note:

***The value of the window scale factor can be determined only during connection establishment; it does not change during the connection.***

## **Figure 12.45** *Timestamp option*



Note:

***One application of the timestamp option is the calculation of round trip time (RTT).***



cuu duong than cong . com

# EXAMPLE 12

***Figure 12.46 shows an example that calculates the round-trip time for one end. Everything must be flipped if we want to calculate the RTT for the other end.***

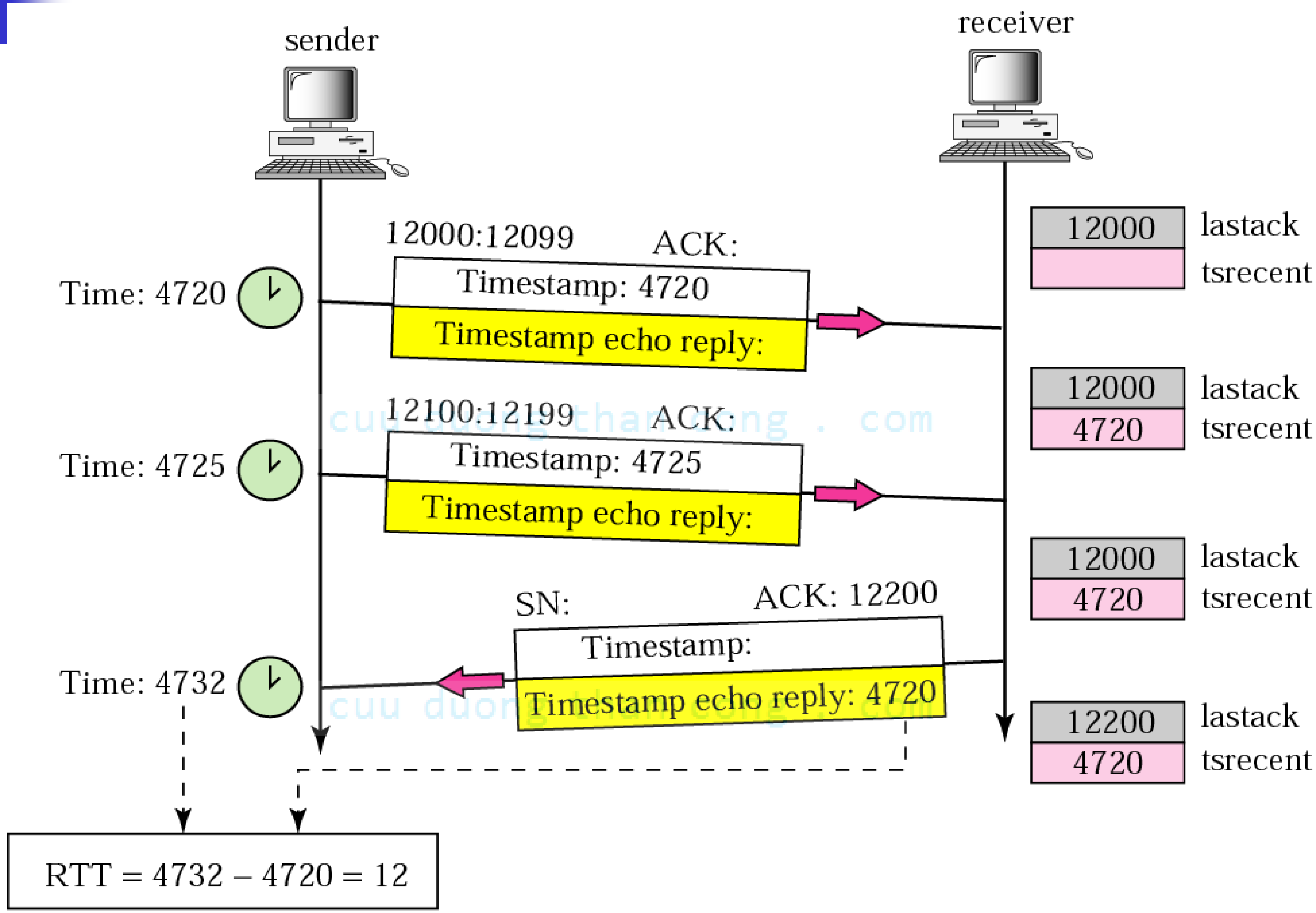
***The sender simply inserts the value of the clock (for example, the number of seconds past from midnight) in the timestamp field for the first and second segment. When an acknowledgment comes (the third segment), the value of the clock is checked and the value of the echo reply field is subtracted from the current time. RTT is 12 s in this scenario.***

## EXAMPLE 12 (CONTINUED)

*The receiver's function is more involved. It keeps track of the last acknowledgment sent (12000). When the first segment arrives, it contains the bytes 12000 to 12099. The first byte is the same as the value of lastack. It then copies the timestamp value (4720) into the tsrecent variable. The value of lastack is still 12000 (no new acknowledgment has been sent). When the second segment arrives, since none of the byte numbers in this segment include the value of lastack, the value of the timestamp field is ignored. When the receiver decides to send an accumulative acknowledgment with acknowledgment 12200, it changes the value of lastack to 12200 and inserts the value of tsrecent in the echo reply field. The value of tsrecent will not change until it isreplaced by a new segment that carries byte 12200 (next segment).*

## EXAMPLE 12 (CONTINUED)

***Note that as the example shows, the RTT calculated is the time difference between sending the first segment and receiving the third segment. This is actually the meaning of RTT: the time difference between a packet sent and the acknowledgment received. The third segment carries the acknowledgment for the first and second segments.***

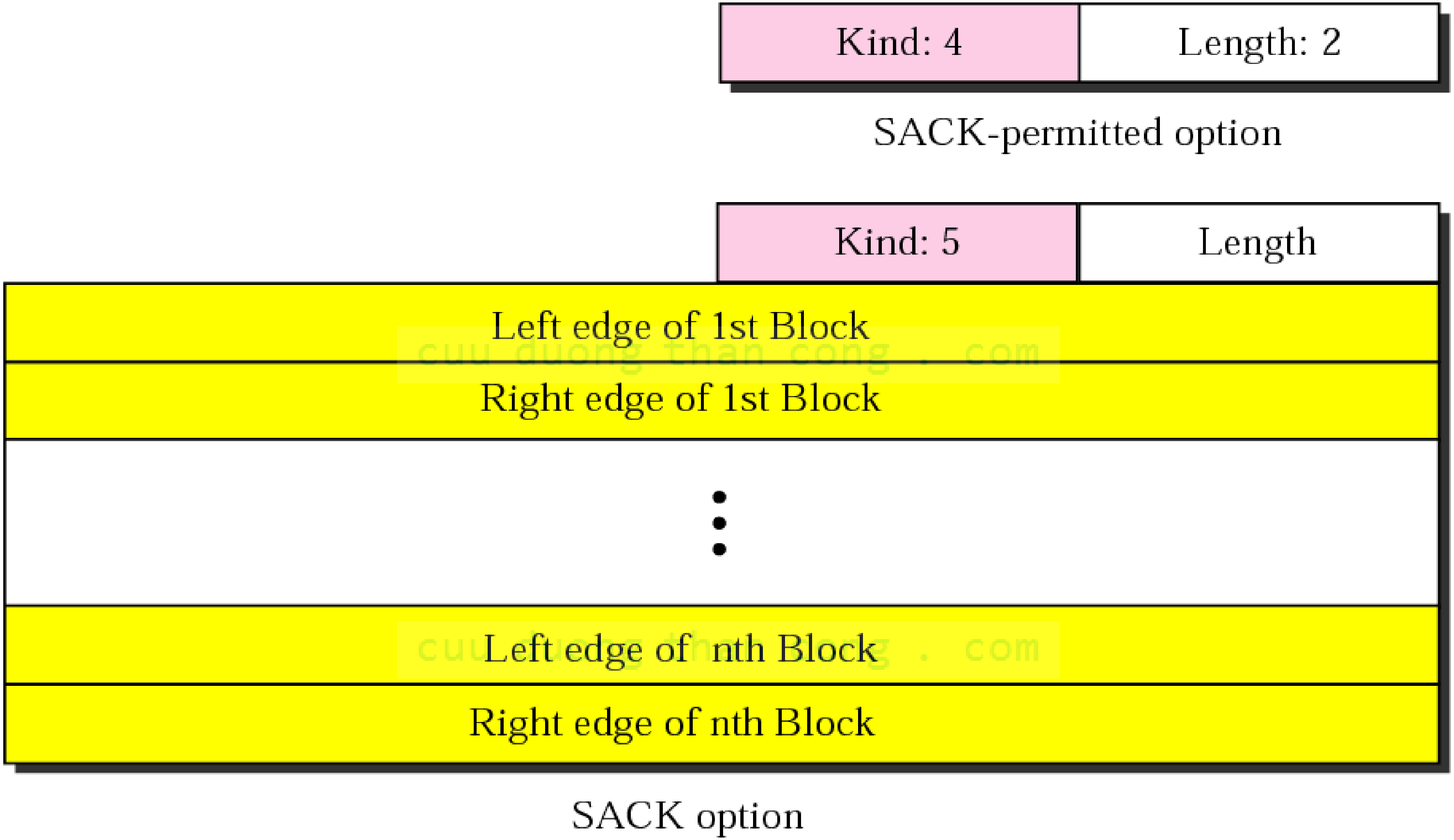


sender
receiver
12000:12099
ACK:
Timestamp: 4720
Timestamp echo reply:
Time: 4720
12000
lastack
tsrecent
12100:12199
ACK:
Timestamp: 4725
Timestamp echo reply:
Time: 4725
12000
4720
lastack
tsrecent
SN:
ACK: 12200
Timestamp:
Timestamp echo reply: 4720
Time: 4732
12000
4720
lastack
tsrecent
12200
4720
lastack
tsrecent
RTT = 4732 – 4720 = 12

Note:

***The timestamp option can also be used for PAWS.***

# Figure 12.47 *SACK*

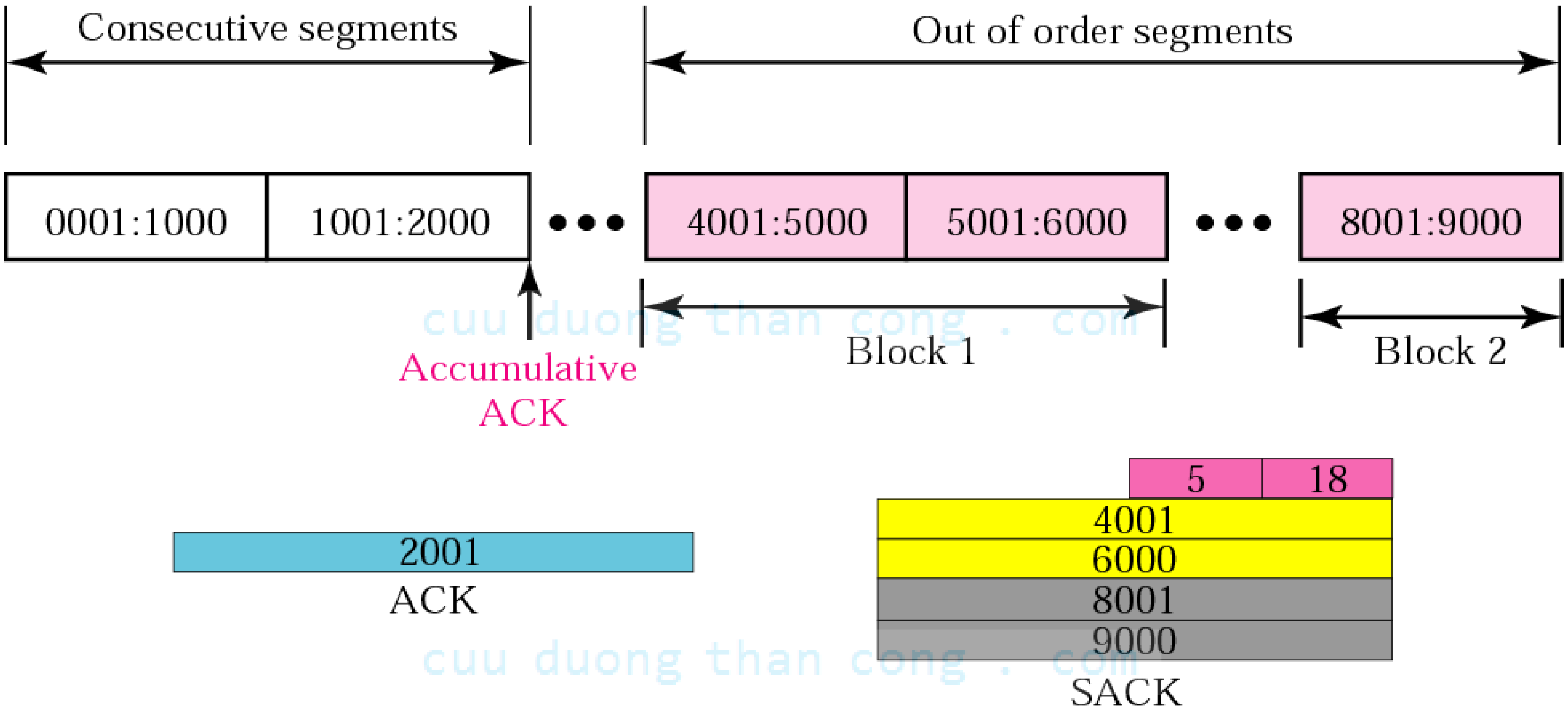


# Example 13

***Let us see how the SACK option is used to list out-of-order blocks. In Figure 12.48 an end has received five segments of data.***

***The first and second segments are in consecutive order. An accumulative acknowledgment can be sent to report the reception of these two segments. Segments 3, 4, and 5, however, are out of order with a gap between the second and third and a gap between the fourth and the fifth. An ACK and a SACK together can easily clear the situation for the sender. The value of ACK is2001, which means that the sender need not worry about bytes 1 to 2000. The SACK has two blocks. The first block announces that bytes 4001 to 6000 have arrived out of order. The second block shows that bytes 8001 to 9000 have also arrived out of order. This means that bytes 2001 to 4000 and bytes 6001 to 8000 are lost or discarded. The sender can resend only these bytes.***

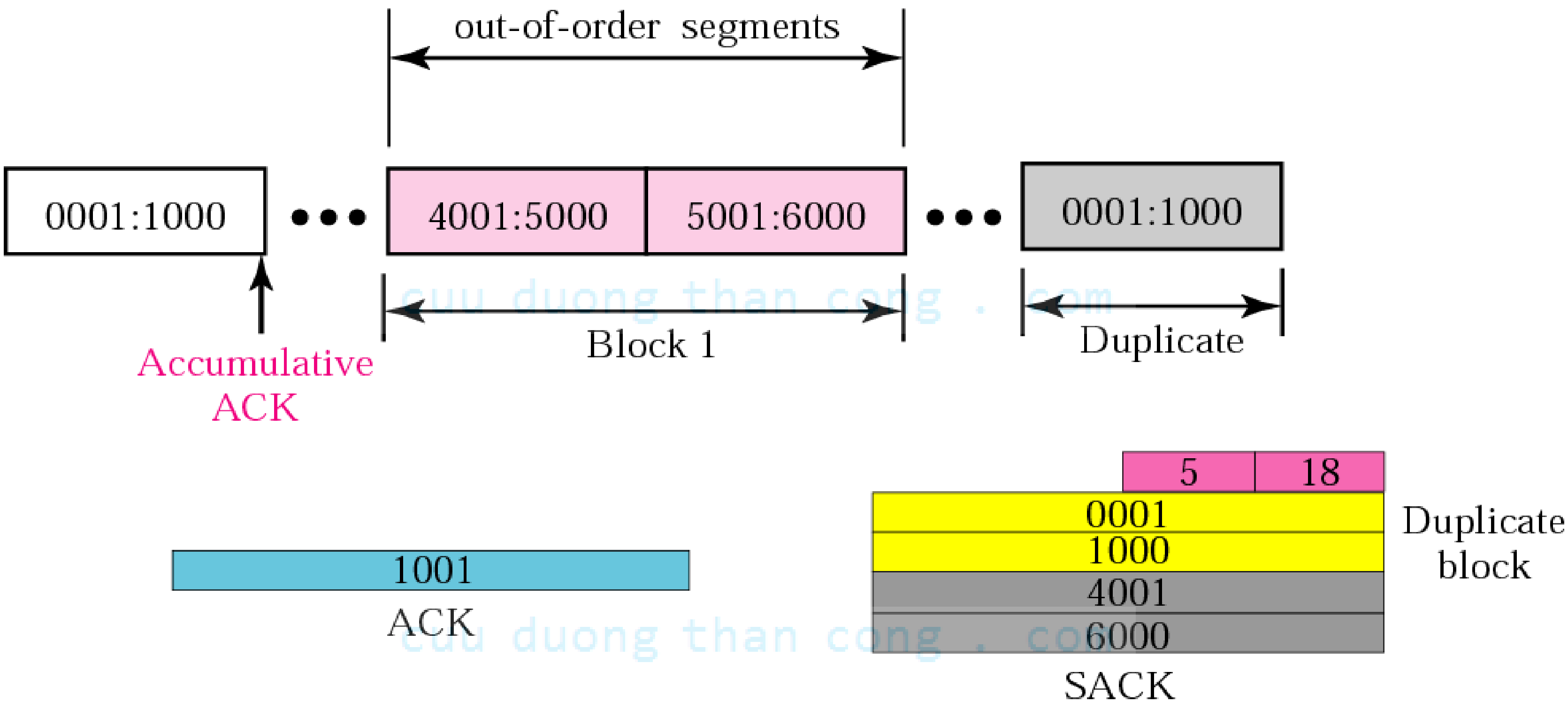
## **Figure 12.48** *Example 13*



# EXAMPLE 14

***The example in Figure 12.49 shows how a duplicate segment can be detected with a combination of ACK and SACK. In this case, we have some out-of-order segments (in one block) and one duplicate segment. To show both out-of-order and duplicate data, SACK uses the first block, in this case, to show the duplicate data and other blocks to show out-of-order data. Note that only the first block can be used for duplicate data. The natural question is how the sender, when it receives these ACK and SACK values knows that the first block is for duplicate data (compare this example with the previous example). The answer is that the bytes in the first block are already acknowledged in the ACK field; therefore, this block must be a duplicate.***

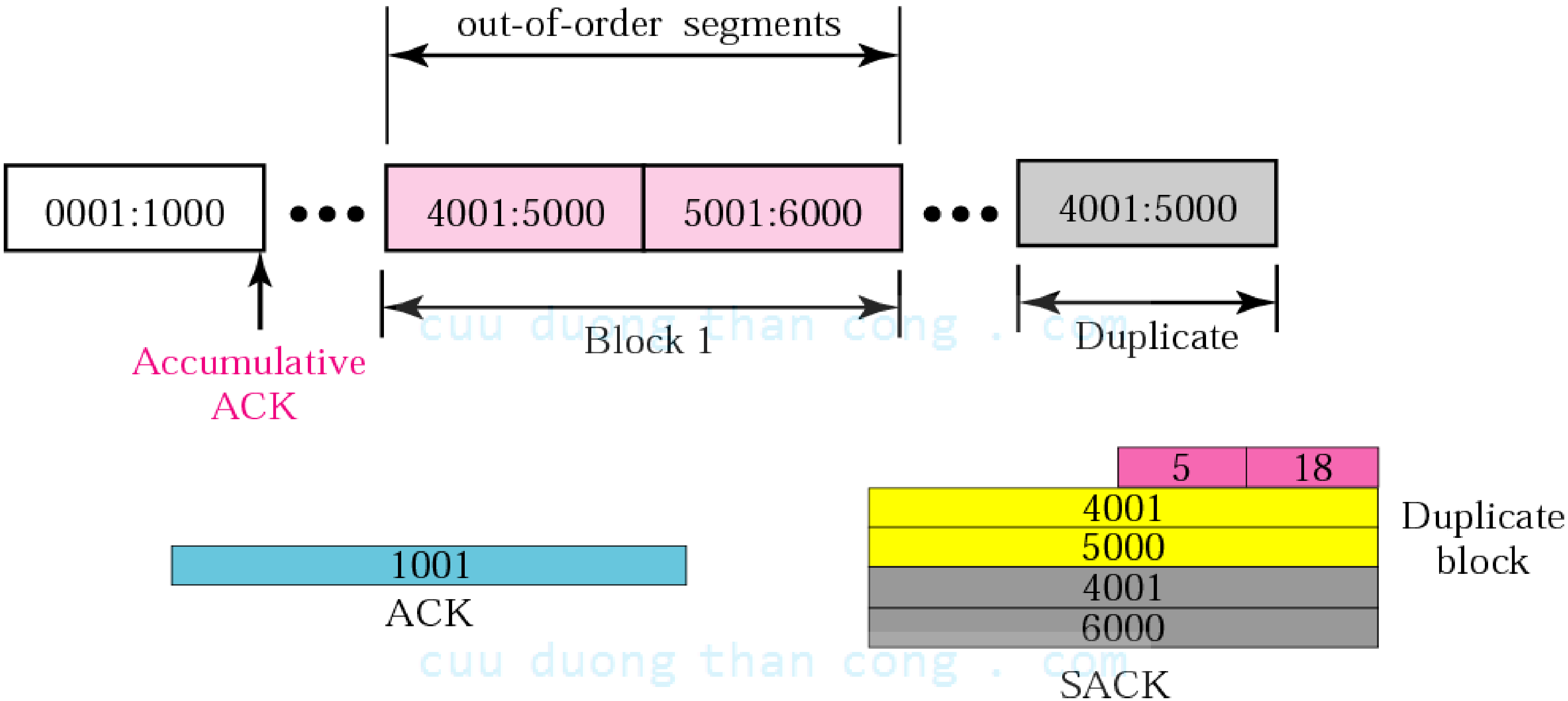
## Figure 12.49 *Example 14*



## Example 15

***The example in Figure 12.50 shows what happens if one of the segments in the out-of-order section is also duplicated. In this example, one of the segments (4001:5000) is duplicated. The SACK option announces this duplicate data first and then the out-of-order block. This time, however, the duplicated block is not yet acknowledged by ACK, but because it is part of the out-of-order block (4001:5000 is part of 4001:6000), it is understood by the sender that it defines the duplicate data.***

# Figure 12.50 *Example 15*



# 12.11 TCP PACKAGE

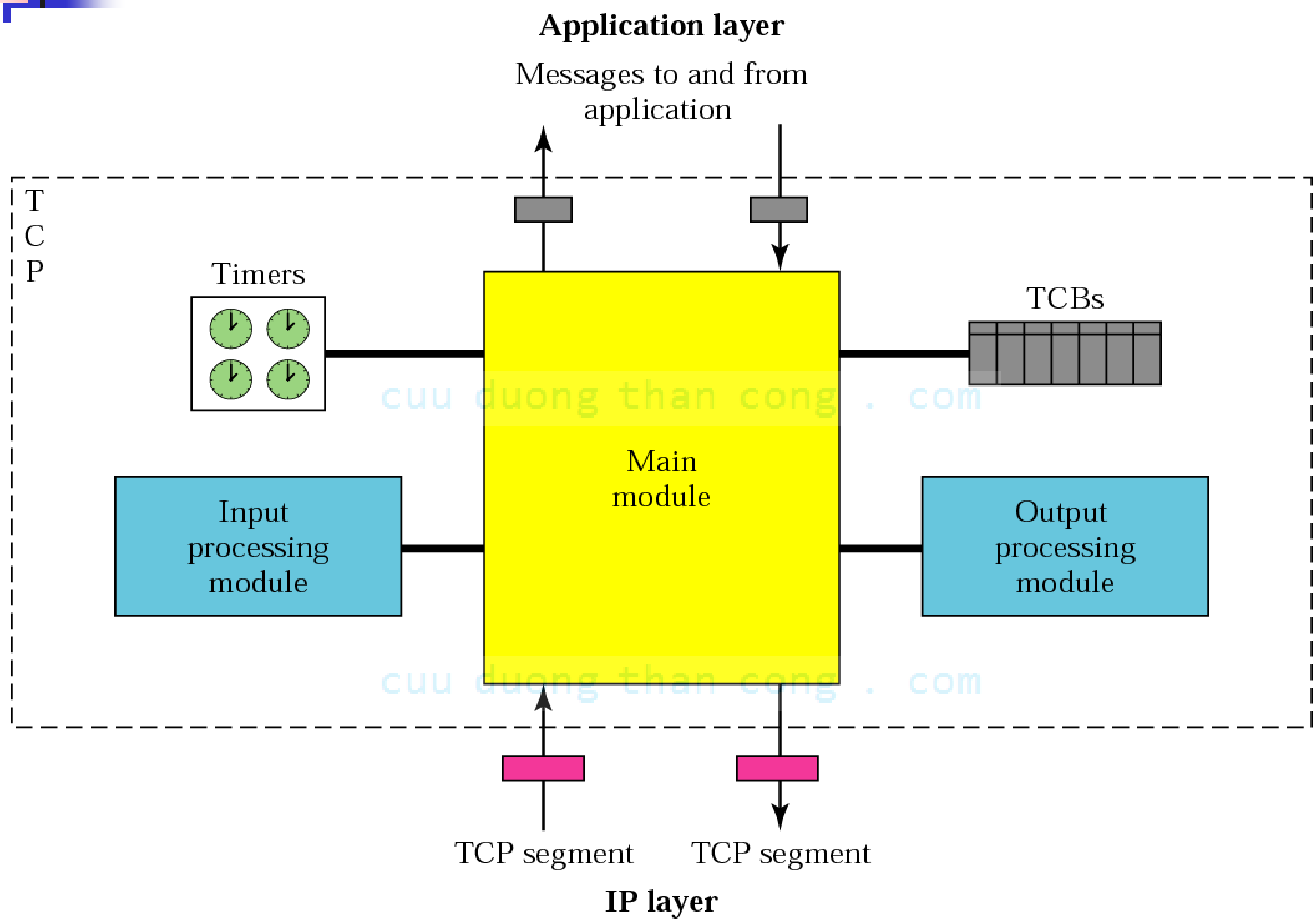
***We present a simplified, bare-bones TCP package to simulate the heart of TCP. The package involves tables called transmission control blocks, a set of timers, and three software modules.***



***The topics discussed in this section include:***

***Transmission Control Blocks (TCBs)***
***Timers***
***Main Module***
***Input Processing Module***
***Output Processing Module***

## Figure 12.51 *TCP package*



## Figure 12.52 *TCBs*