# *UNIT - 4*

# *Transport Layer*

# Outline

3.1 INTRODUCTION

3.2 TRANSPORT-LAYER PROTOCOLS

3.3 USER DATAGRAM PROTOCOL

3.4 TRANSMISSION CONTROL PROTOCOL

# INTRODUCTION

- Process-to-Process Communication
- Addressing: Port Numbers
- ICANN Ranges
  - Well-known ports
  - Registered ports
  - Dynamic ports
- Encapsulation and Decapsulation
- Multiplexing and Demultiplexing

# INTRODUCTION

- Flow Control
  - Pushing or Pulling
  - Flow Control at Transport Layer
  - Buffers
- Error Control
  - Sequence Numbers
  - Acknowledgment
- Combination of Flow and Error Control
  - Sliding Window

# INTRODUCTION

- Congestion Control

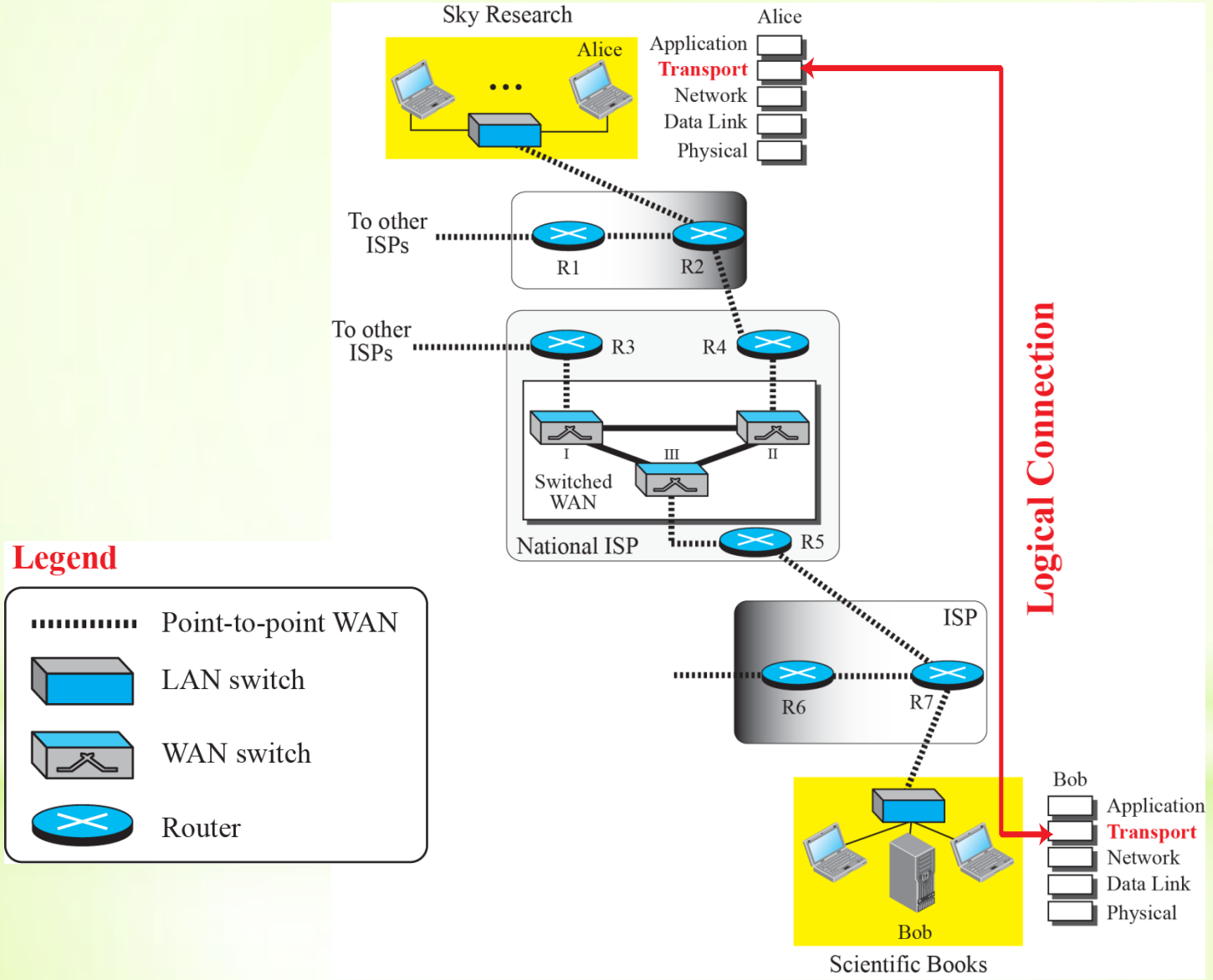
- Connectionless and Connection-Oriented

  - Connectionless Service
  - Connection-Oriented Service
  - Finite State Machine

# INTRODUCTION

- The transport layer provides a process-to-process communication between two application layers.
- Communication is provided using a logical connection, which means that the two Transport layers assume that there is an imaginary direct connection through which they can send and receive messages.
- The transport layer is located between the network layer and the application layer. The transport layer is responsible for providing services to the application layer; it receives services from the network layer.

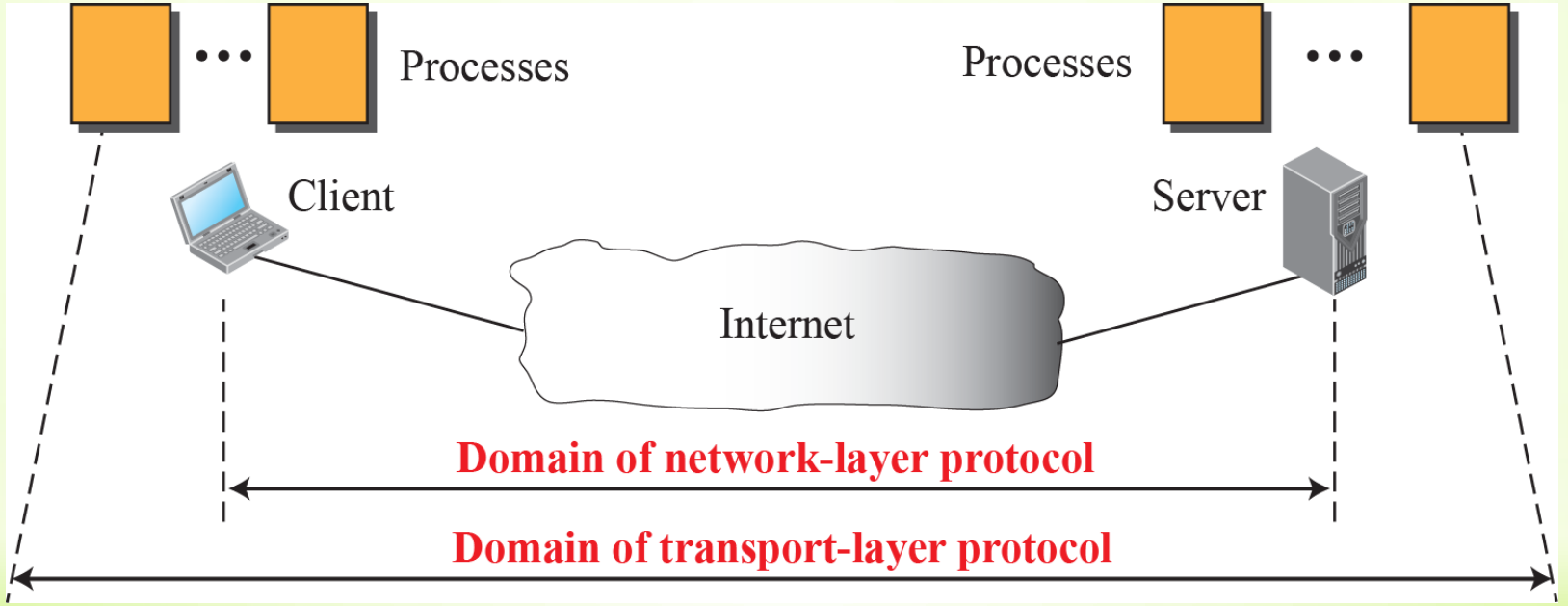
# Logical connection at the transport layer



# Process-to-Process Communication

- The first duty of a transport-layer protocol is to provide process-to-process communication. A process is an application-layer entity (running program) that uses the services of the transport layer.
- A network-layer protocol can deliver the message only to the destination computer. However, this is an incomplete delivery. The message still needs to be handed to the correct process. This is where a transport-layer protocol takes over. A transport-layer protocol is responsible for delivery of the message to the appropriate process.

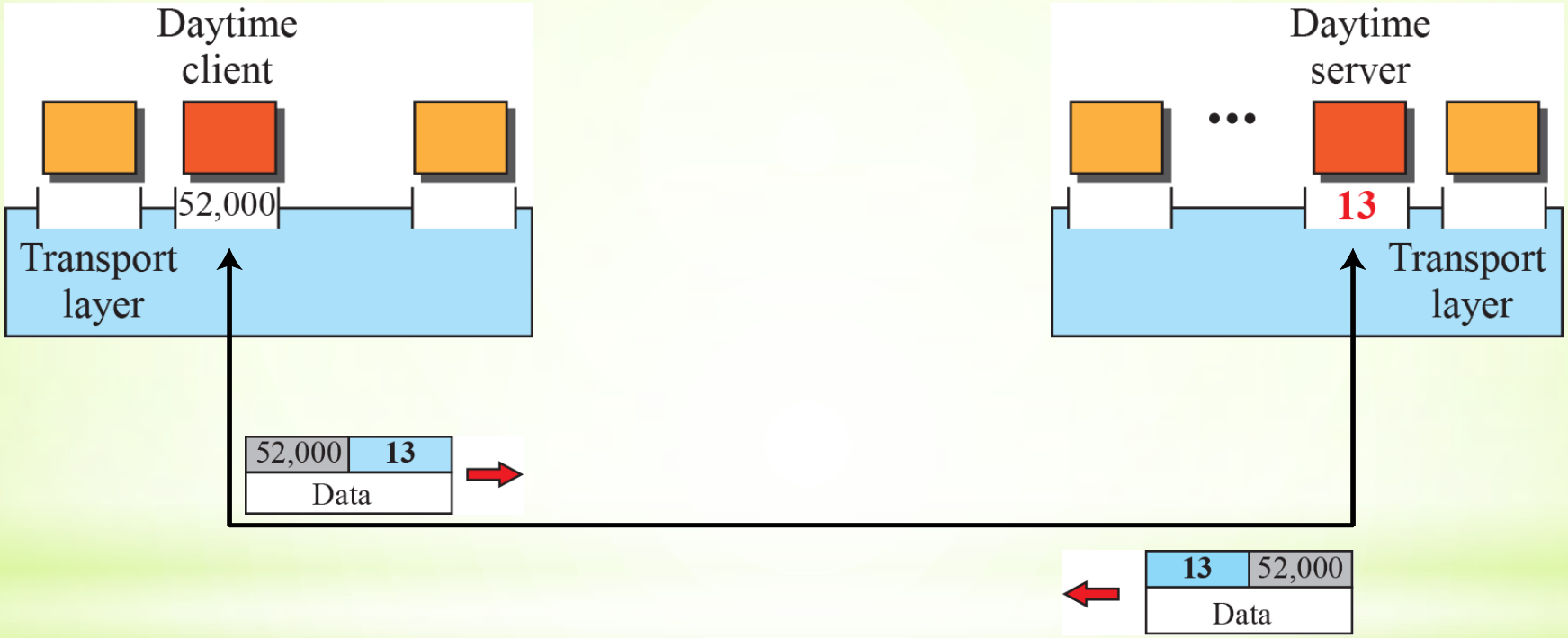
# *Network layer versus transport layer*



# Addressing: Port Numbers

- A process on the local host, called a client, needs services from a process usually on the remote host, called a server.
- The local host and the remote host are defined using IP addresses
- To define the processes, we need second identifiers, called port numbers.
- In the TCP/IP protocol suite, the port numbers are integers between 0 and 65,535 (16 bits).
- ICANN has divided the port numbers into three ranges:
  - Well-known ports. The ports ranging from 0 to 1,023 are assigned and controlled by ICANN. These are the well-known ports.
  - Registered ports. The ports ranging from 1,024 to 49,151 are not assigned or controlled by ICANN. They can only be registered with ICANN to prevent duplication.
  - Dynamic ports. The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used as temporary or private port numbers

# Port numbers

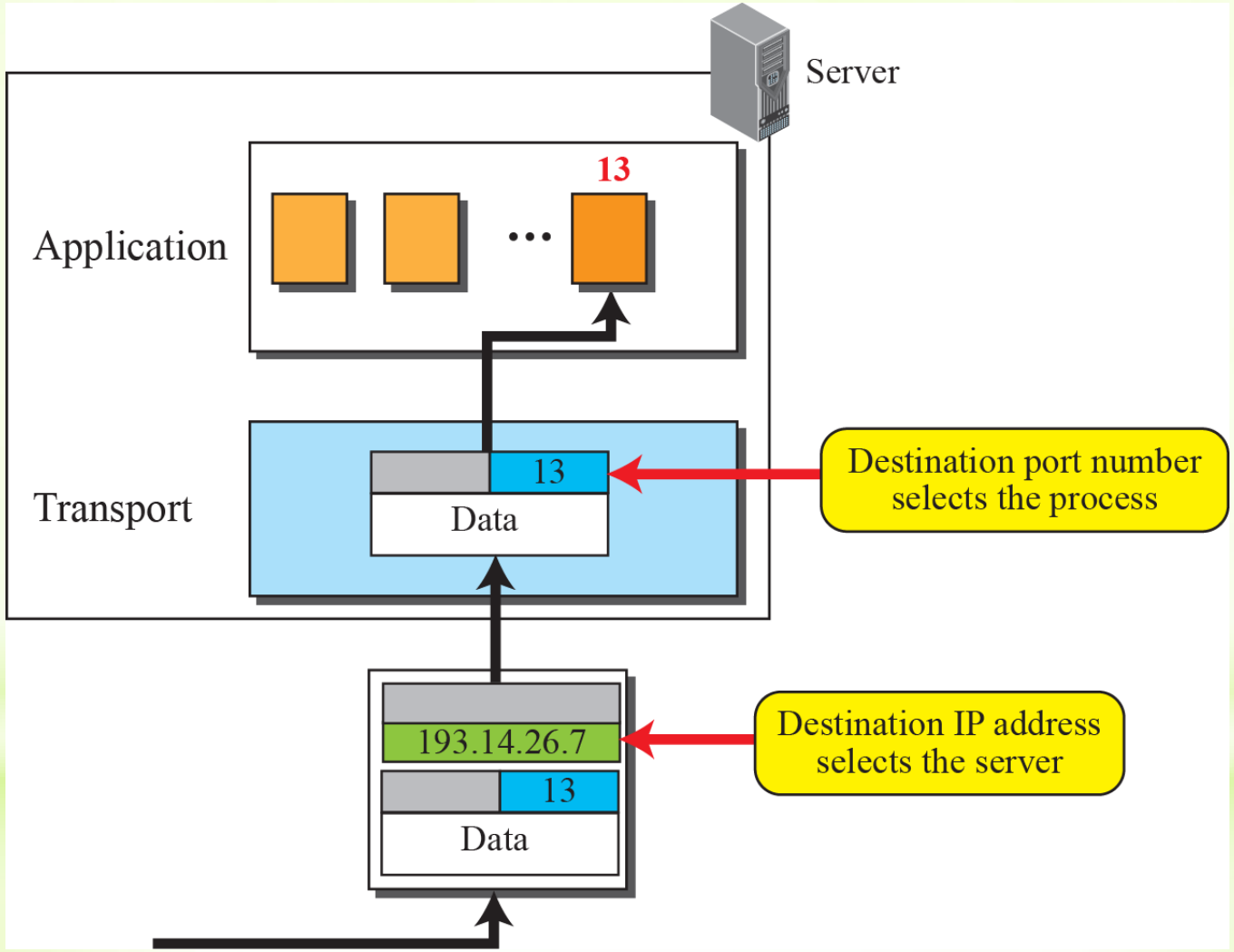


## *ICANN ranges*


The client program defines itself with a port number, called the ephemeral port number.

The server process must also define itself with a port number. This port number, however, cannot be chosen randomly. TCP/IP has decided to use universal port numbers for servers; these are called well-known port numbers(for standardardized Client-server applications)

# *IP addresses versus port numbers*



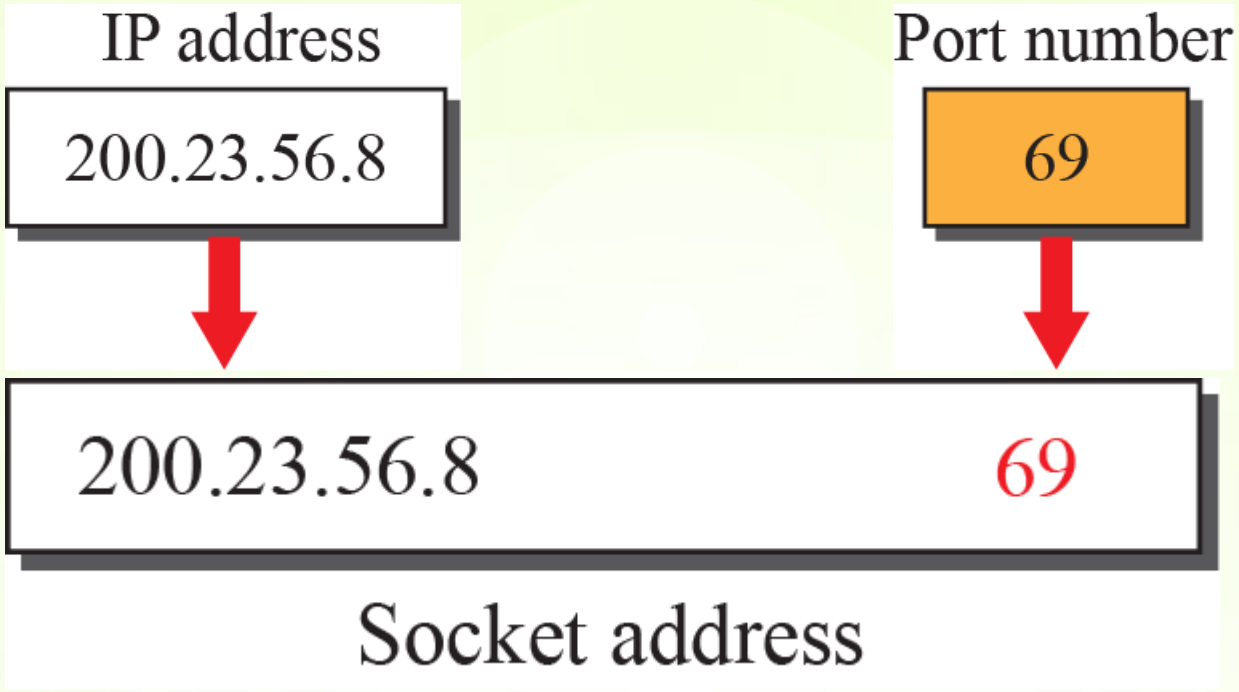
In UNIX, the well-known ports are stored in a file called /etc/services. We can use the *grep* utility to extract the line corresponding to the desired application.

```
$grep     tftp/etc/services
tftp 69/tcp
tftp 69/udp
```

SNMP (see Chapter 9) uses two port numbers (161 and 162), each for a different purpose.

```
$grep     snmp/etc/services
snmp161/tcp#Simple Net Mgmt Proto
snmp161/udp#Simple Net Mgmt Proto
snmptrap162/udp#Traps for SNMP
```

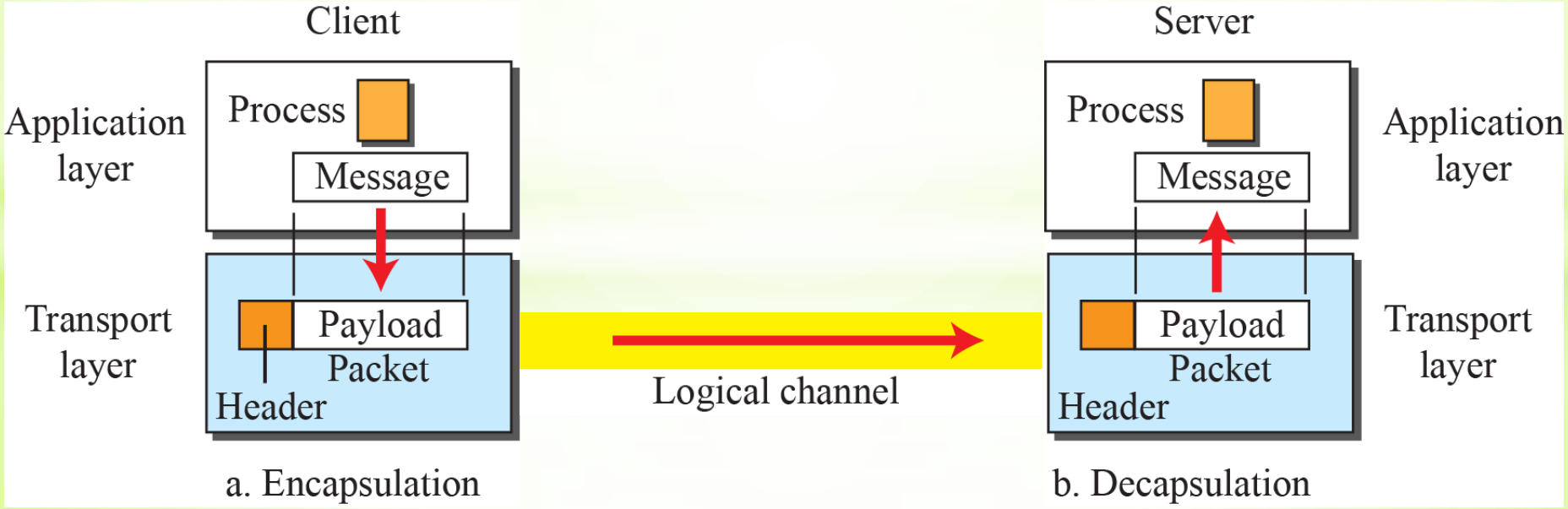
## *Socket address*




- To use services in the Internet, we need a pair of socket addresses: the client socket address and the server socket address at each end, to make a connection.
- The combination of an IP address and a port number is called a socket address

# *Encapsulation and decapsulation*

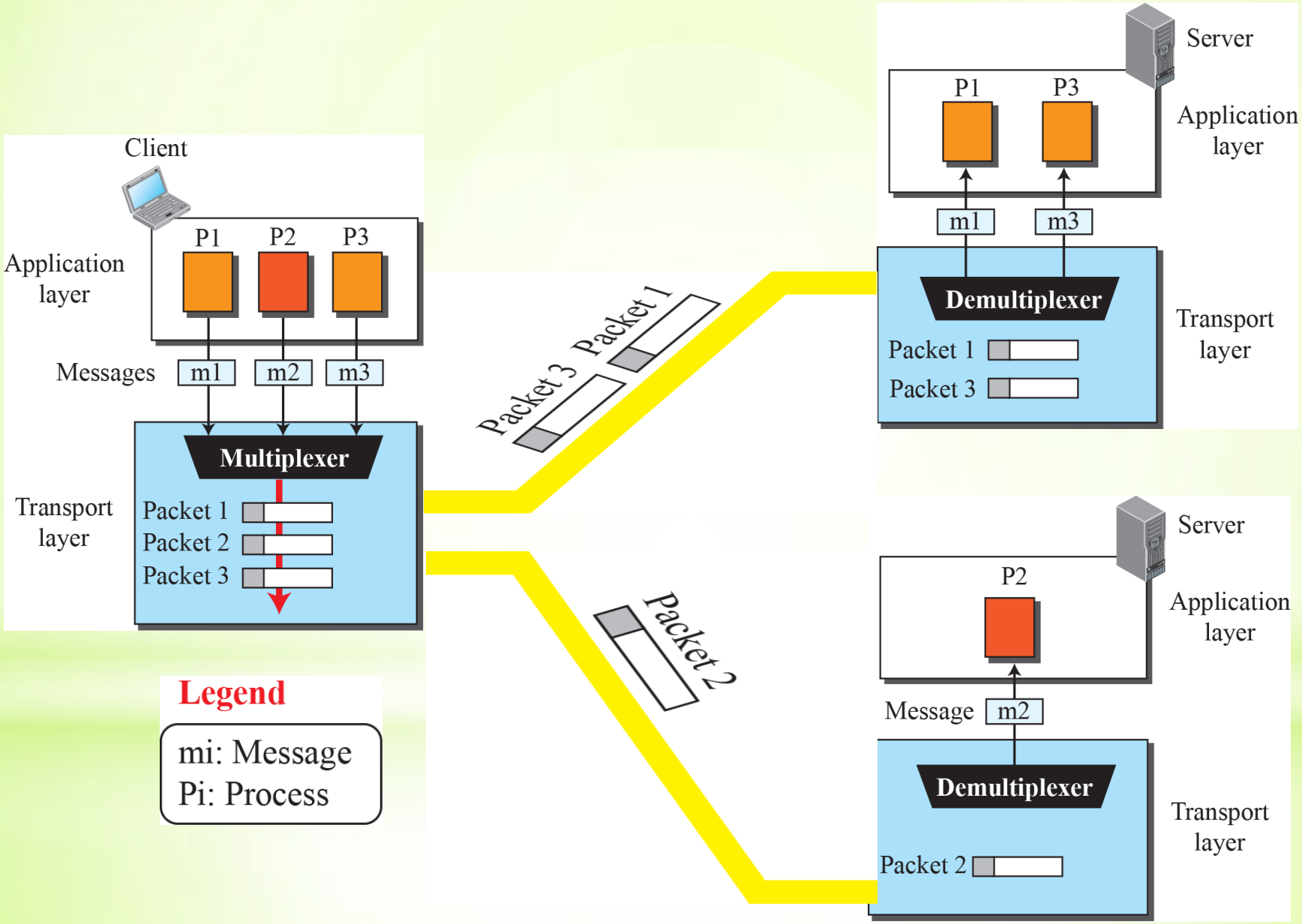
- Encapsulation happens at the sender site. The transport layer receives the data and adds the transport-layer header(i.e. port address, and other information). The packets at the transport layers in the Internet are called **user datagrams, segments, or packets**, depending on what transport-layer protocol we use.
- Decapsulation happens at the receiver site. When the message arrives at the destination transport layer, the header is dropped and the transport layer delivers the message to the process running at the application layer. The sender socket address is passed to the process in case it needs to respond to the message received.



# *Multiplexing and demultiplexing*

- Whenever an entity accepts items from more than one source, this is referred to as multiplexing (many to one); whenever an entity delivers items to more than one source, this is referred to as demultiplexing (one to many).
- The transport layer at the source performs multiplexing; the transport layer at the destination performs demultiplexing.
- The transport layer at the client site accepts three messages from the three processes and creates three packets. It acts as a multiplexer.
- When they arrive at the server, the transport layer does the job of a demultiplexer and distributes the messages to two different processes.

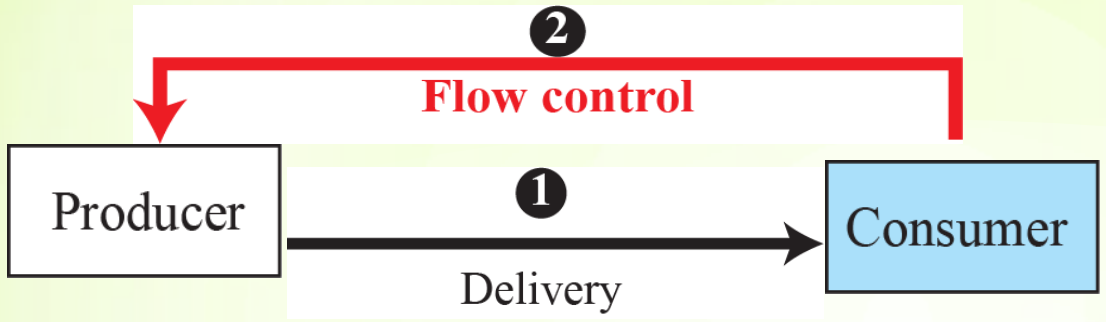
# Multiplexing and demultiplexing



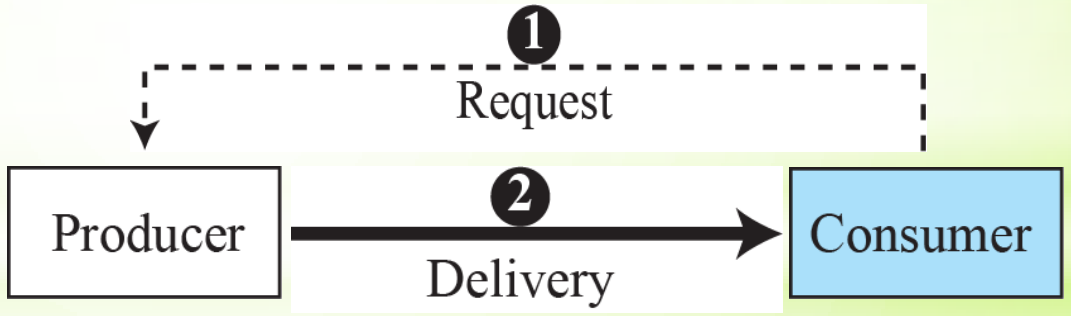
# *Flow Control*

- In communication at the transport layer, we are dealing with four entities: sender process, sender transport layer, receiver transport layer, and receiver process.
- The sending process at the application layer is only a producer. It produces message chunks and pushes them to the transport layer. The sending transport layer has a double role: it is both a consumer and a producer. It consumes the messages pushed by the producer. It encapsulates the messages in packets and pushes them to the receiving transport layer.
- The receiving transport layer also has a double role, it is the consumer for the packets received from the sender and the producer that decapsulates the messages and delivers them to the application layer. The last delivery, however, is normally a pulling delivery; the transport layer waits until the application-layer process asks for messages.

## *Pushing or pulling*

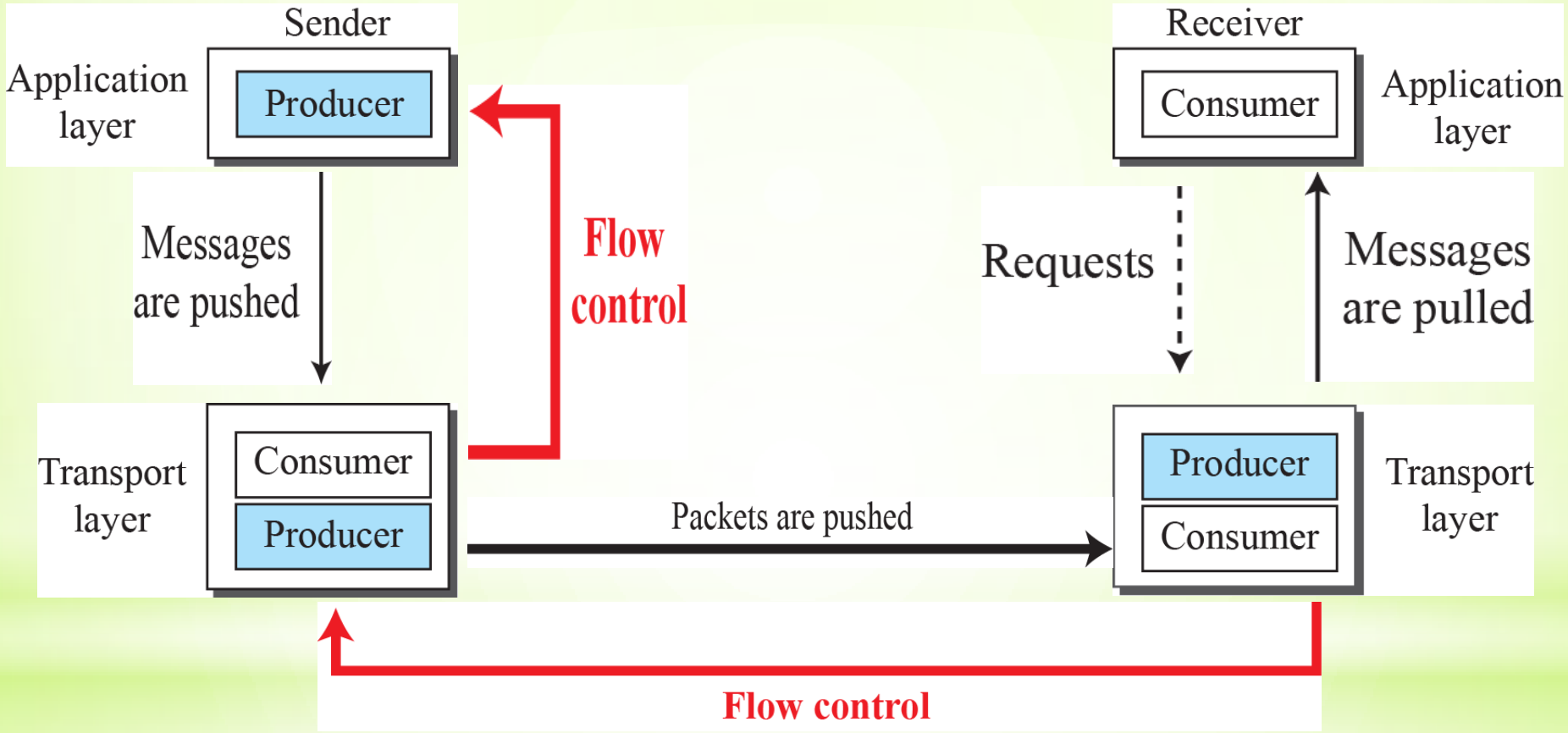



a. Pushing




b. Pulling

# *Flow control at the transport layer*



# Buffers

- Although flow control can be implemented in several ways, one of the solutions is normally to use two buffers: one at the sending transport layer and the other at the receiving transport layer.
- A buffer is a set of memory locations that can hold packets at the sender and receiver.
- When the buffer of the sending transport layer is full, it informs the application layer to stop passing chunks of messages; when there are some vacancies, it informs the application layer that it can pass message chunks again.
- When the buffer of the receiving transport layer is full, it informs the sending transport layer to stop sending packets. When there are some vacancies, it informs the sending transport layer that it can send packets again.

# Error Control

- In the Internet, since the underlying network layer (IP) is unreliable, we need to make the transport layer reliable if the application requires reliability.
- Reliability can be achieved to add error control services to the transport layer. Error control at the transport layer is responsible for:
  1. Detecting and discarding corrupted packets.
  2. Keeping track of lost and discarded packets and resending them.
  3. Recognizing duplicate packets and discarding them.
  4. Buffering out-of-order packets until the missing packets arrive.

## SEQUENCE NUMBER

- To perform error control, the packets are numbered. We can add a field to the transport-layer packet to hold the sequence number of the packet. Packets are numbered sequentially. If the header of the packet allows m bits for the sequence number, the sequence numbers range from 0 to $2^m - 1$.

## ACKNOWLEDGMENT NUMBER

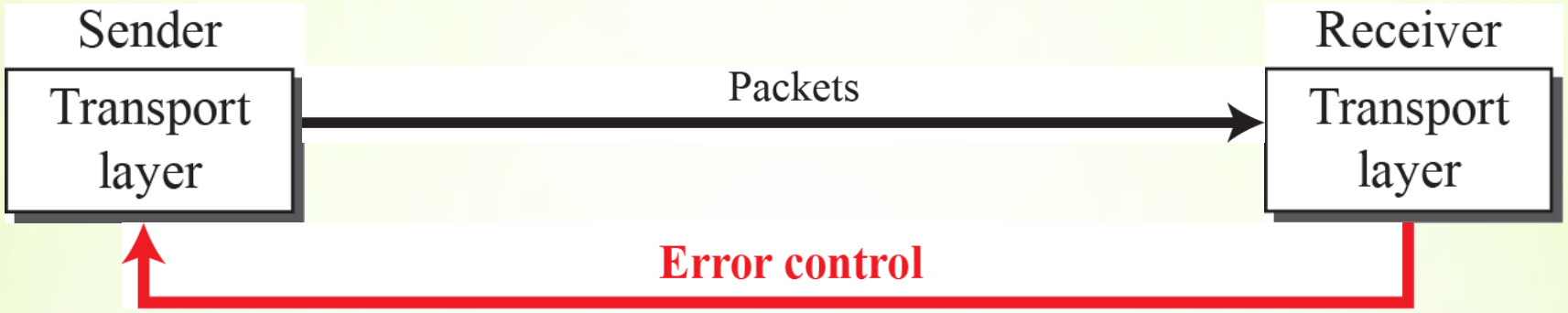
- We can use both positive and negative signals as error control
- The receiver side can send an acknowledgment (ACK) for each of a collection of packets that have arrived safe and sound. The receiver can simply discard the corrupted packets.

## TIMERS

- The sender can detect lost packets if it uses a timer.
- When a packet is sent, the sender starts a timer. If an ACK does not arrive before the timer expires, the sender resends the packet.

Duplicate packets can be silently discarded by the receiver. Out-of-order packets can be either discarded (to be treated as lost packets by the sender), or stored until the missing ones arrives.

## *Error control at the transport layer*



# COMBINATION OF FLOW AND ERROR CONTROL

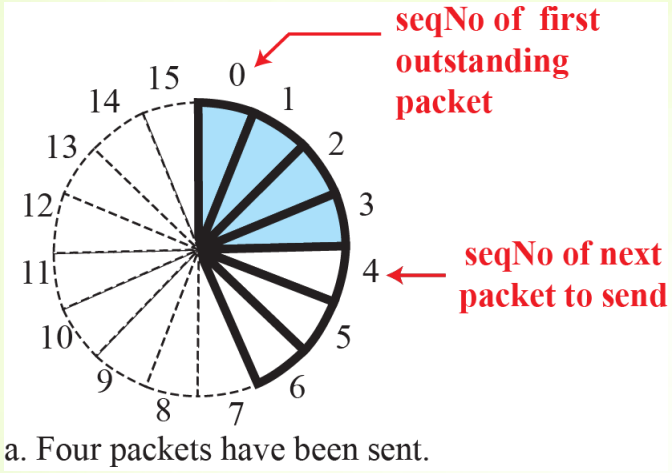
- Flow control requires the use of two buffers, one at the sender site and the other at the receiver site.
- Error control requires the use of sequence and acknowledgment numbers by both sides.
- These two requirements can be combined if we use two numbered buffers, one at the sender, one at the receiver.

- At the sender, when a packet is prepared to be sent, we use the number of the next free location, x, in the buffer as the sequence number of the packet. When the packet is sent, a copy is stored at memory location x, awaiting the acknowledgment from the other end. When an acknowledgment related to a sent packet arrives, the packet is purged and the memory location becomes free.
- At the receiver, when a packet with sequence number y arrives, it is stored at the memory location y until the application layer is ready to receive it. An acknowledgment can be sent to announce the arrival of packet y.

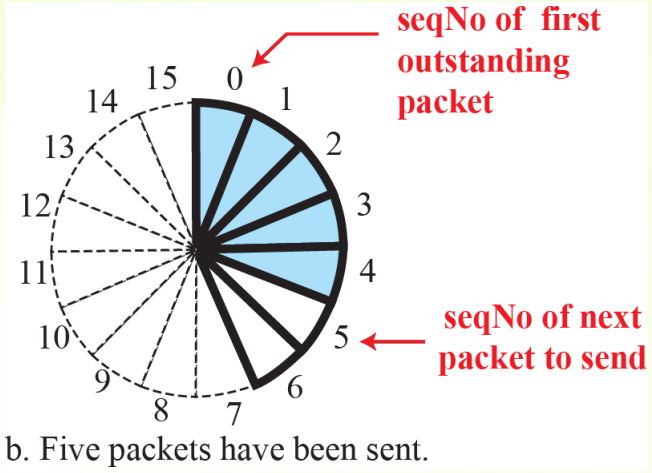
# SLIDING WINDOW

- Since the sequence numbers used modulo $2^m$, a circle can represent the sequence numbers from 0 to $2^m - 1$
- The buffer is represented as a set of slices, called the sliding window, that occupies part of the circle at any time.
- At the sender site, when a packet is sent, the corresponding slice is marked. When all the slices are marked, it means that the buffer is full and no further messages can be accepted from the application layer.
- When an acknowledgment arrives, the corresponding slice is unmarked.
- If some consecutive slices from the beginning of the window are unmarked, the window slides over the range of the corresponding sequence numbers to allow more free slices at the end of the window.
- Most protocols show the sliding window using linear representation.

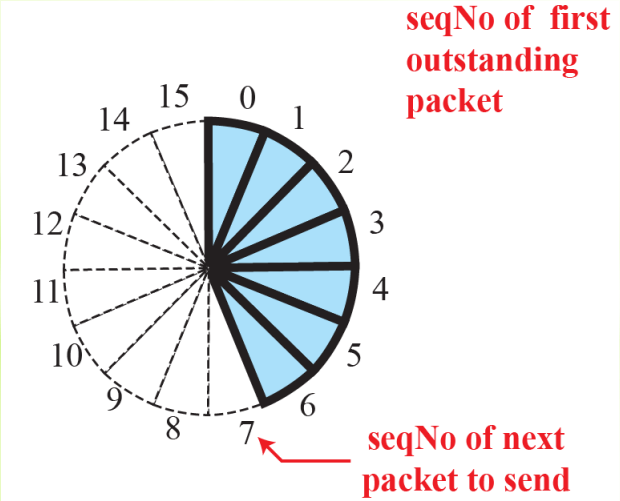
# Sliding window in circular format



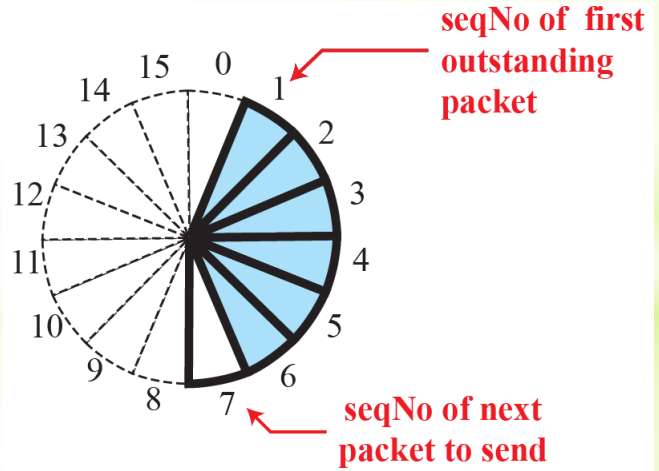

a. Four packets have been sent.




b. Five packets have been sent.

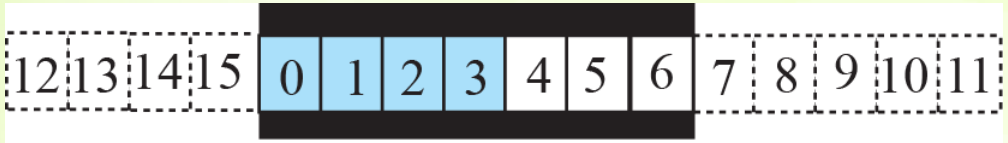



c. Seven packets have been sent;
window is full.



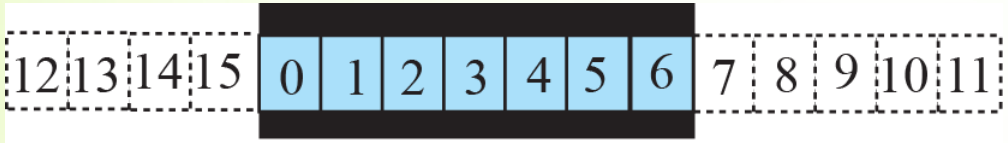

d. Packet 0 has been acknowledged;
window slides.

## *Sliding window in linear format*

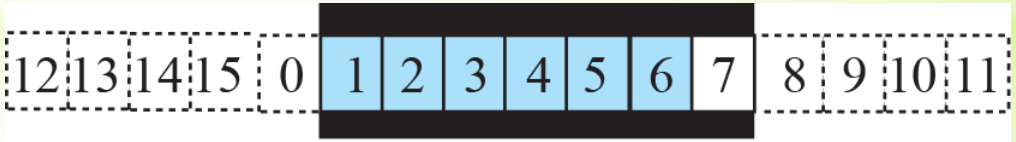



a. Four packets have been sent.


b. Five packets have been sent.




c. Seven packets have been sent;
window is full.




d. Packet 0 has been acknowledged;
window slides.

# Congestion Control

- An important issue in a packet-switched network, such as the Internet, is congestion.
- Congestion in a network may occur if the load on the network—the number of packets sent to the network—is greater than the capacity of the network—the number of packets a network can handle.
- Congestion control refers to the mechanisms and techniques that control the congestion and keep the load below the capacity.
- Congestion at the transport layer is actually the result of congestion at the network layer, which manifests itself at the transport layer
- TCP, assuming that there is no congestion control at the network layer, implements its own congestion control mechanism

# Connectionless and Connection-Oriented Services

- A transport-layer protocol, like a network-layer protocol, can provide two types of services: connectionless and connection-oriented.

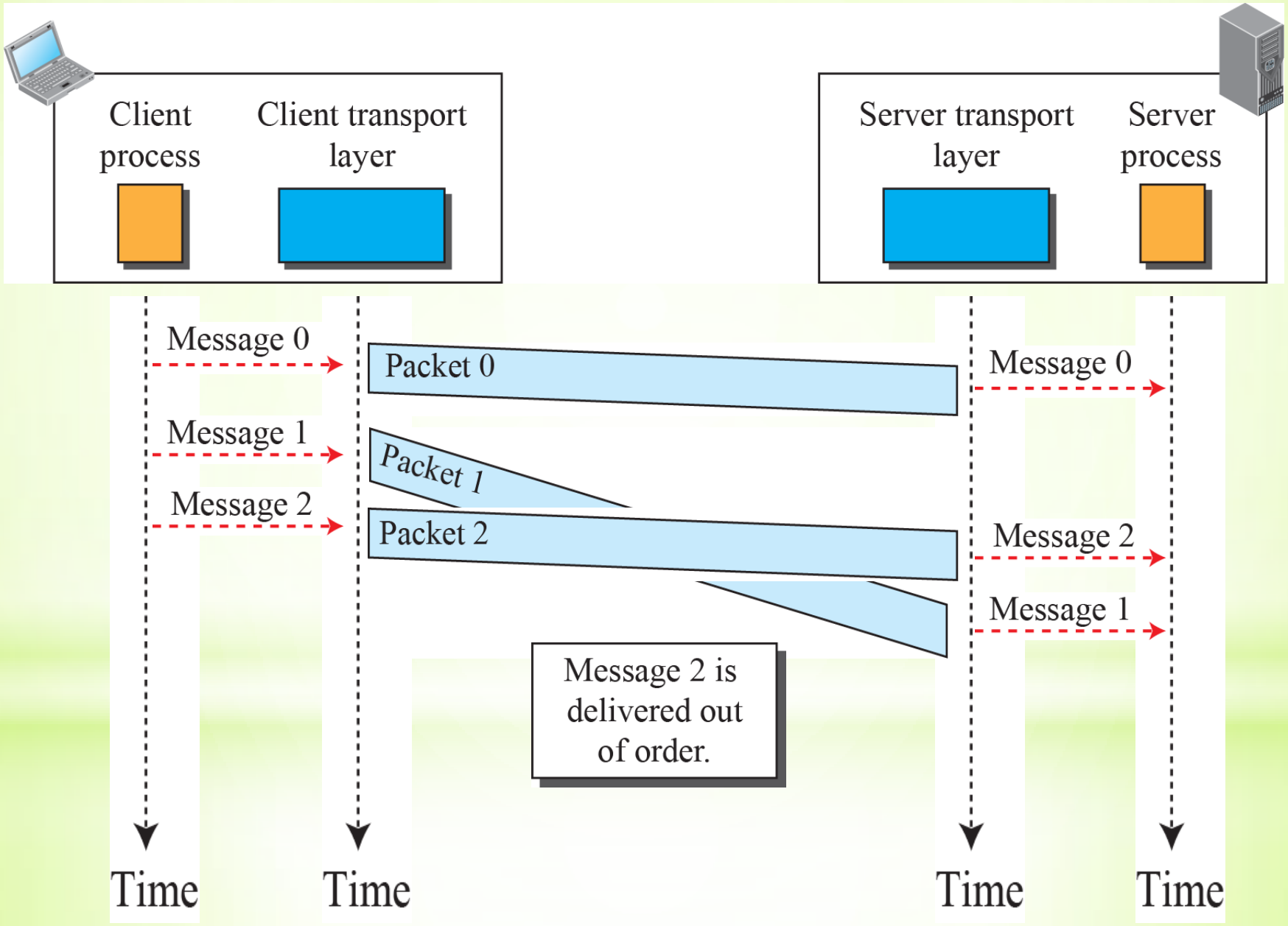
- At the transport layer, we are not concerned about the physical paths of packets (we assume a logical connection between two transport layers).

- Connectionless service at the transport layer means independency between packets; connection-oriented means dependency.

# CONNECTIONLESS SERVICE

- In a connectionless service, the source process (application program) needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one.
- The transport layer treats each chunk as a single unit without any relation between the chunks. When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it.
- Problems such as lost packets, out of order delivery exists in such mechanism.
- We can say that no flow control, error control, or congestion control can be effectively implemented in a connectionless service.
- A well known real time protocol used for Connectionless service is UDP.

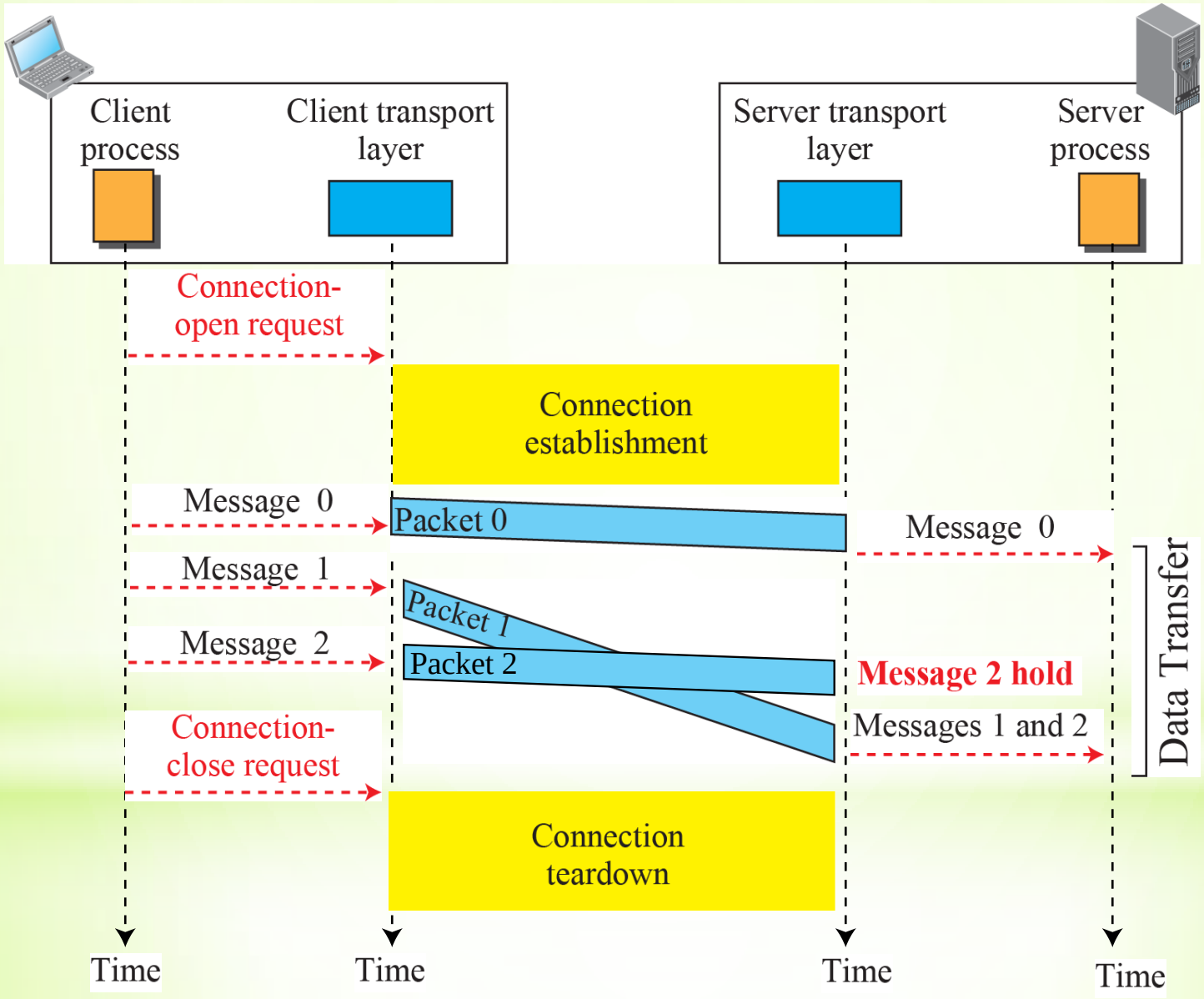
## *Connectionless service*



# CONNECTION-ORIENTED SERVICE

- In a connection-oriented service, the client and the server first need to establish a logical connection between themselves.
- The data exchange can only happen after the connection establishment.
- One completed the data exchange the connection is closed
- We can implement flow control, error control, and congestion control in a connection oriented protocol.
- A well known real time protocol used for Connection oriented service is TCP.

# Connection-oriented service



# FINITE STATE MACHINE

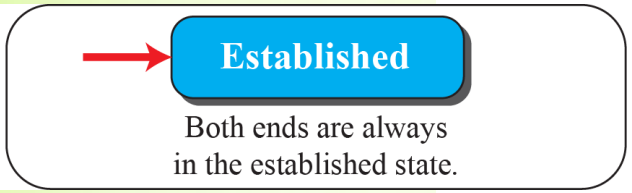
- The behavior of a transport-layer protocol, both when it provides a connectionless and when it provides a connection-oriented protocol, can be better shown as a **finite state machine (FSM).**

- In a connectionless service mechanism, there is only one state i.e. both ends are always in the established state.

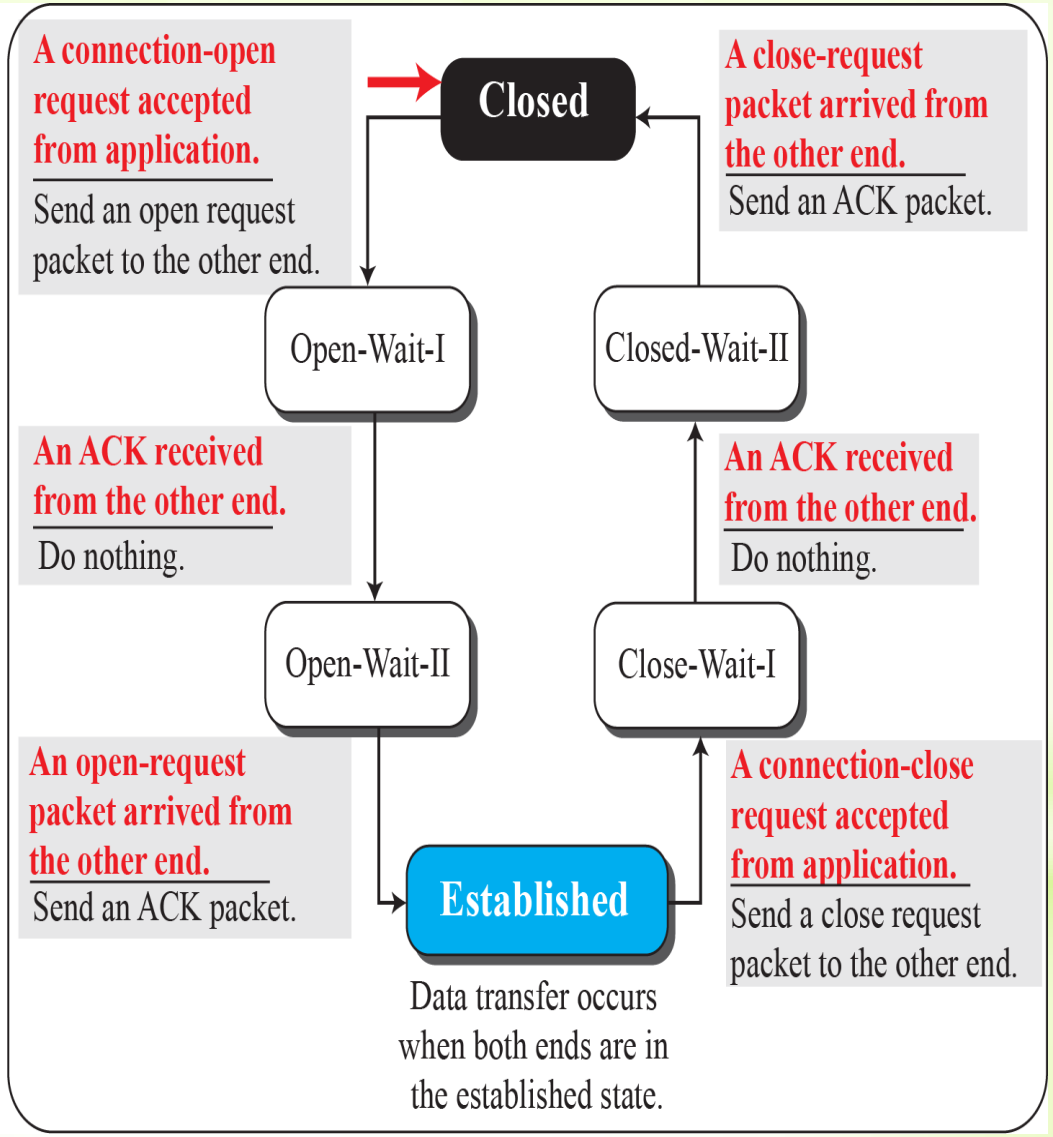
- In a connection oriented mechanism, there are totally six states:
  - Open-wait-I
  - Open-wait-II
  - Established
  - Close-wait-I
  - Close-wait-II
  - Closed

# *Connectionless and connection-oriented service represented as FSMs*

FSM for connectionless transport layer




FSM for connection-oriented transport layer




**Note:**
The colored arrow shows the starting state.