Layered Architecture and Network Protocols

EE450: Introduction to Computer Networks

Professor A. Zahid

Protocols

Human Protocols:

- what's the time?
- I have a question
- Introductions

… specific msgs sent

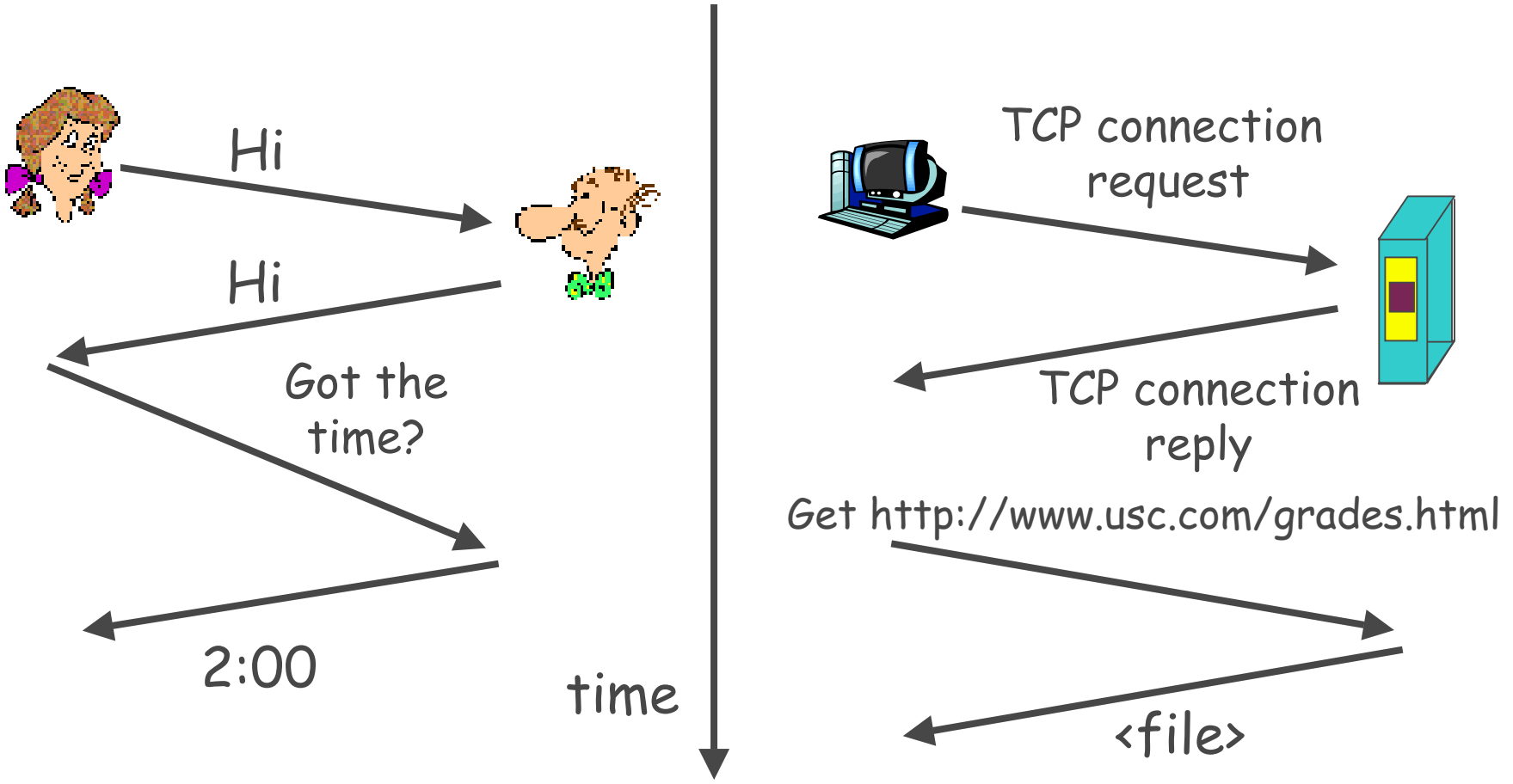
… specific actions taken when msgs received, or other events

Network Protocols:

- Machines rather than humans
- All communication activity in Internet governed by protocols

Protocols define format, order of msgs sent and received among network entities, and actions taken on msg transmission, receipt

Human vs. Network Protocols



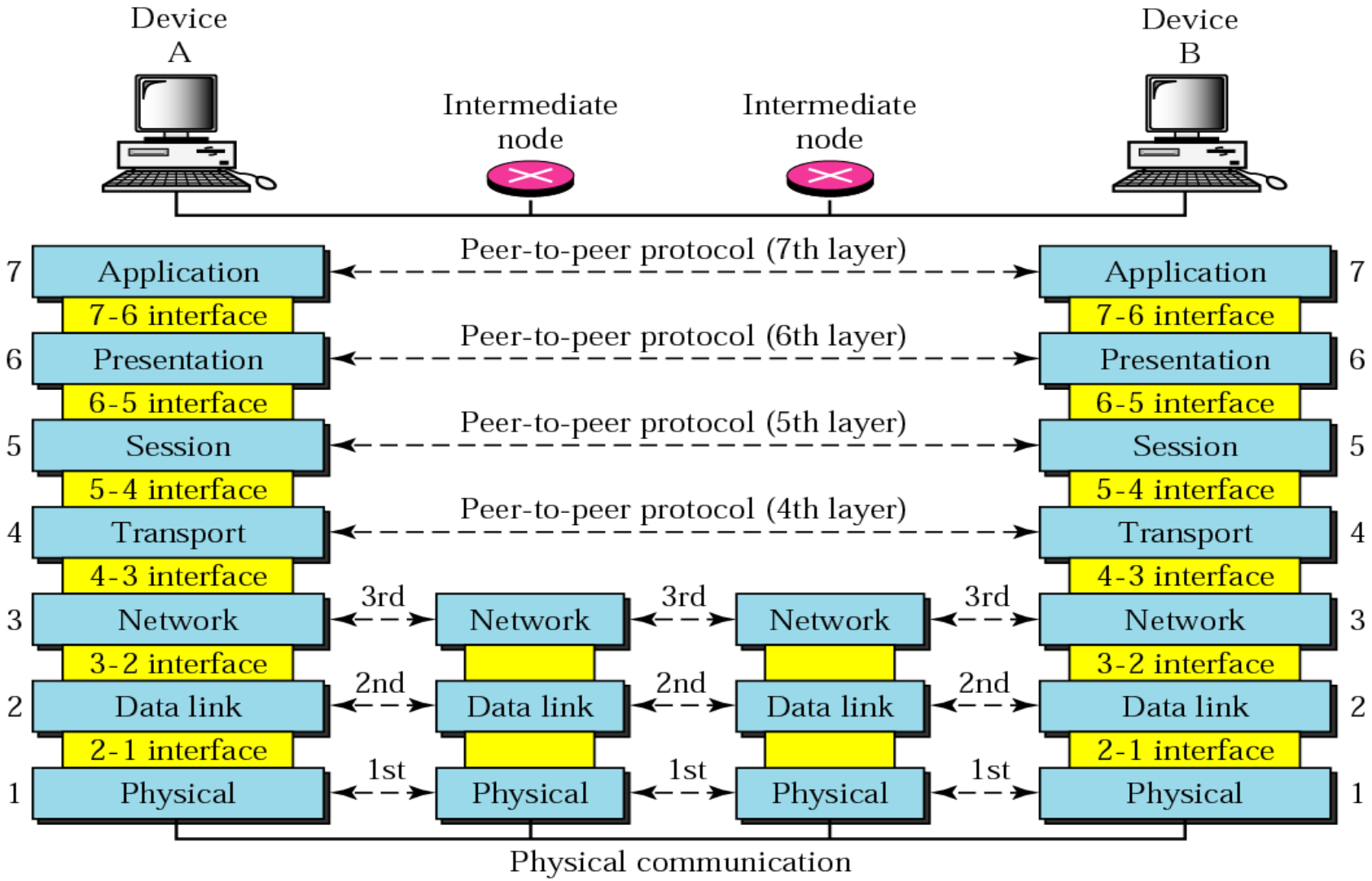
Key Elements of a Protocol

- Syntax
  - Data formats, compression, encryption, etc..
  - Signal levels
- Semantics
  - Control information such as flow & congestion
  - Error detection and control mechanisms
- Timing
  - Speed matching
  - Sequencing
- Fairness

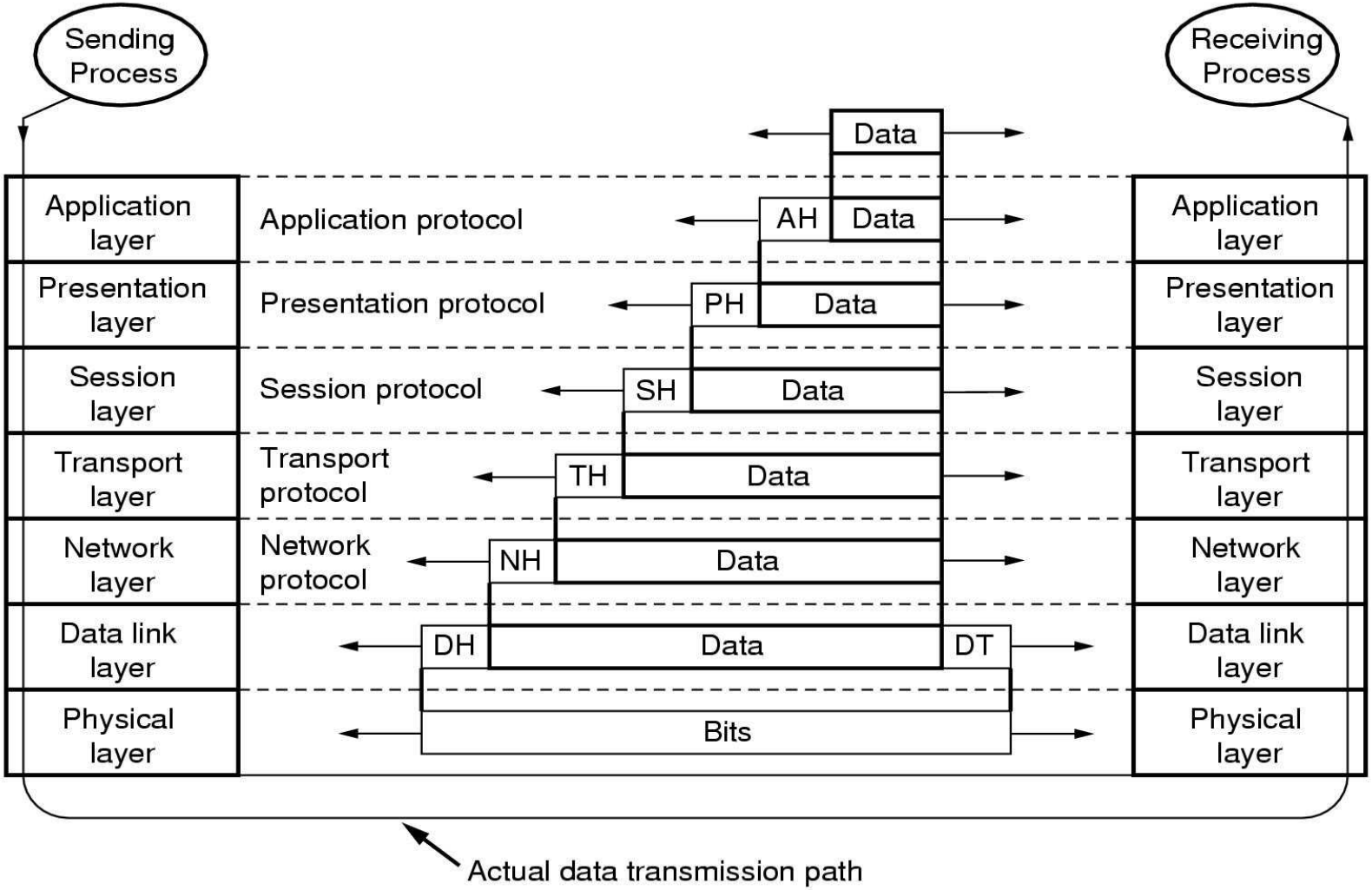
Standards

- Required to allow for interoperability between equipment
- Advantages
  - Ensures a large market for equipment and software
  - Allows products from different vendors to communicate
- Disadvantages
  - Freeze technology

OSI Reference Model



Data Transfer in OSI



Layered Communications

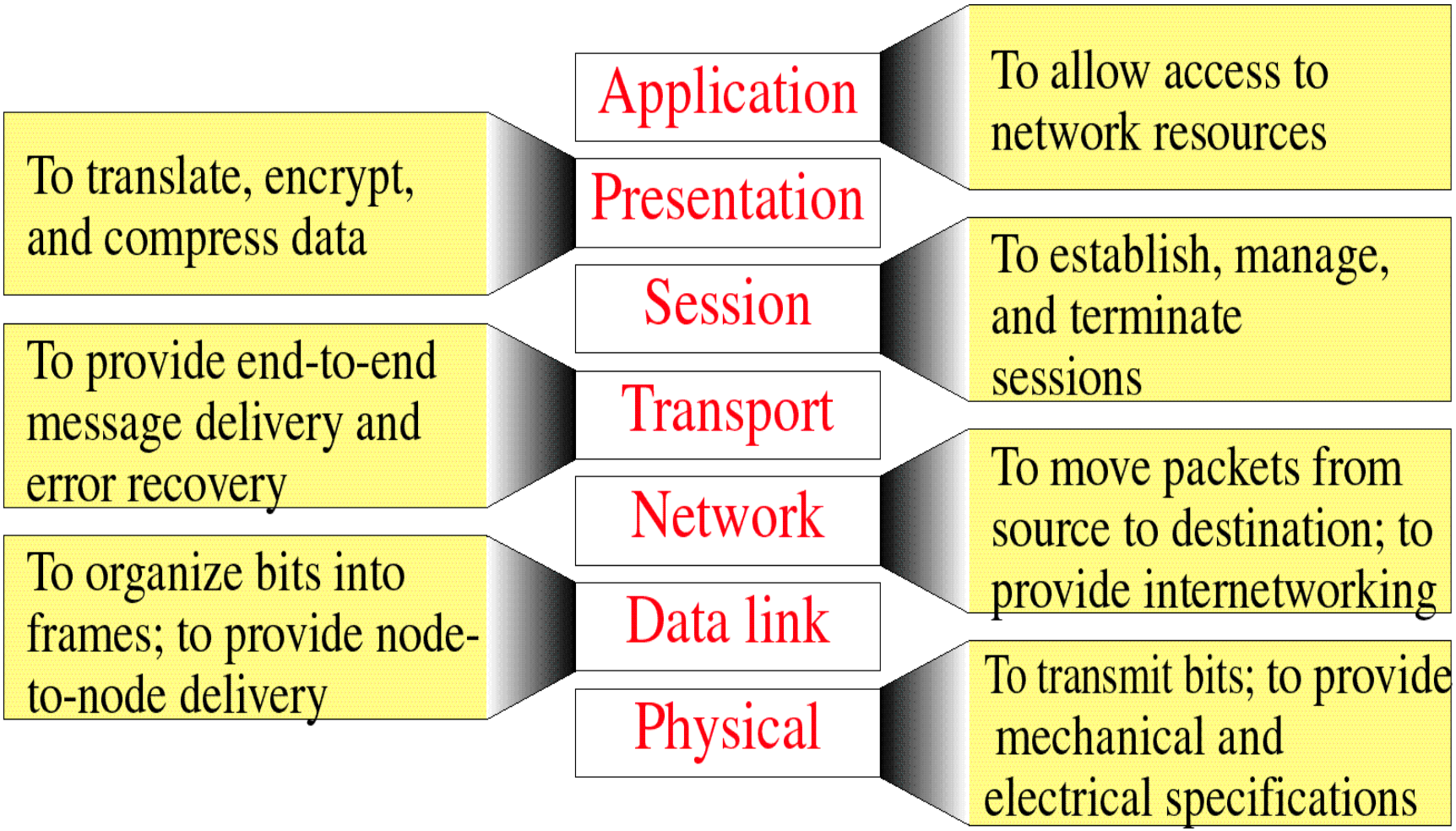
- Peer layers communicate via protocols
- Adjacent layers communicate via service interfaces
- Protocol control information (PCI) is added to Service data unit (SDU) at each layer to form a Protocol data unit (PDU)
  - $(SDU)_N = (PDU)_{N+1}$
  - $(PDU)_N = (SDU)_N + (PCI)_N$

Why Layering?

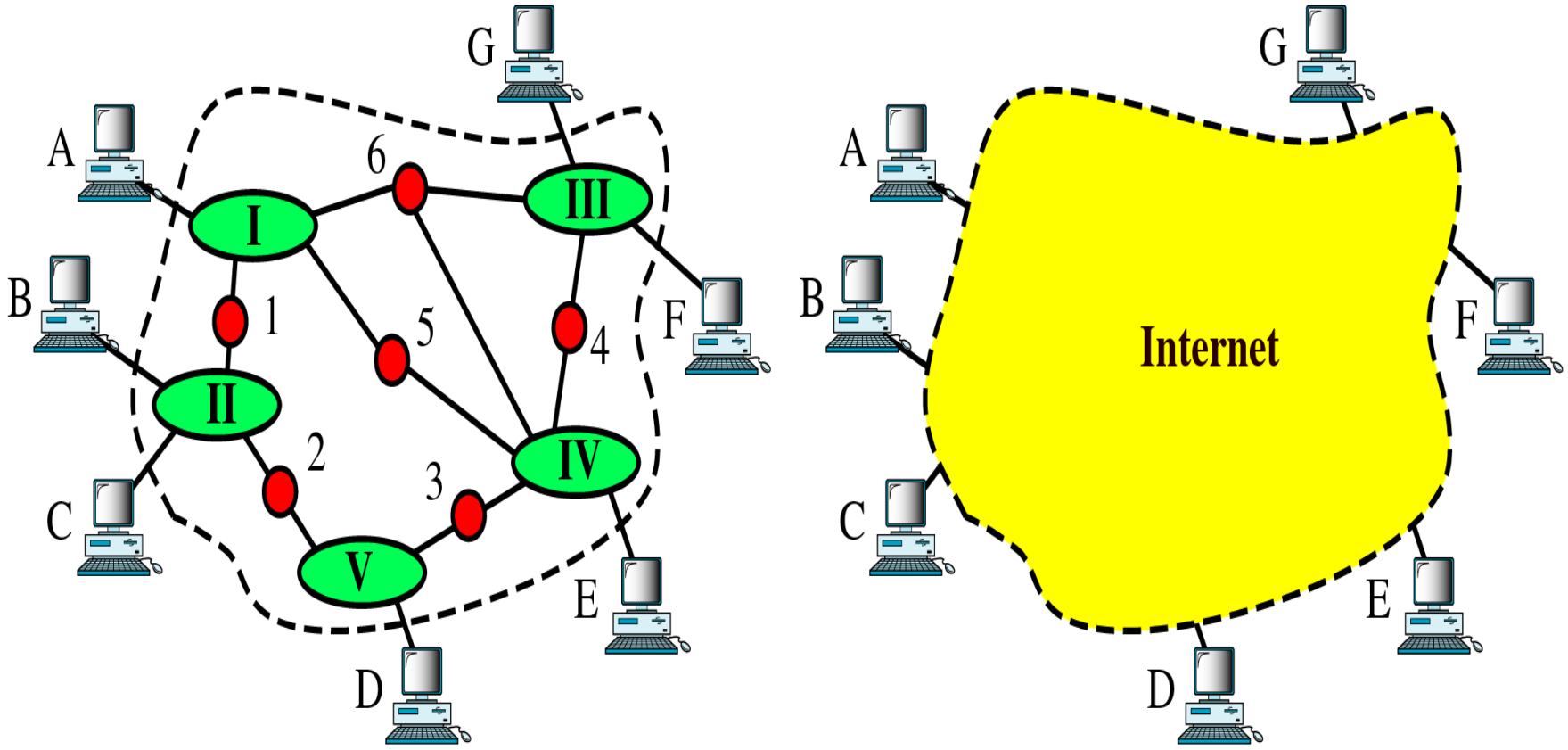
Dealing with complex systems:

- explicit structure allows identification, relationship of complex system's pieces
  - layered reference model for discussion
- Modularization eases maintenance, updating of system
  - change of implementation of layer's service transparent to rest of system
  - change in one layer doesn't affect rest of system

Summary of Layers Functionality

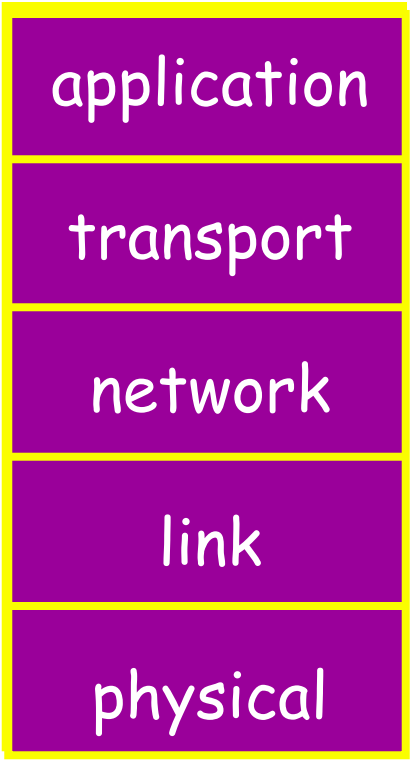


The Internet and TCP/IP



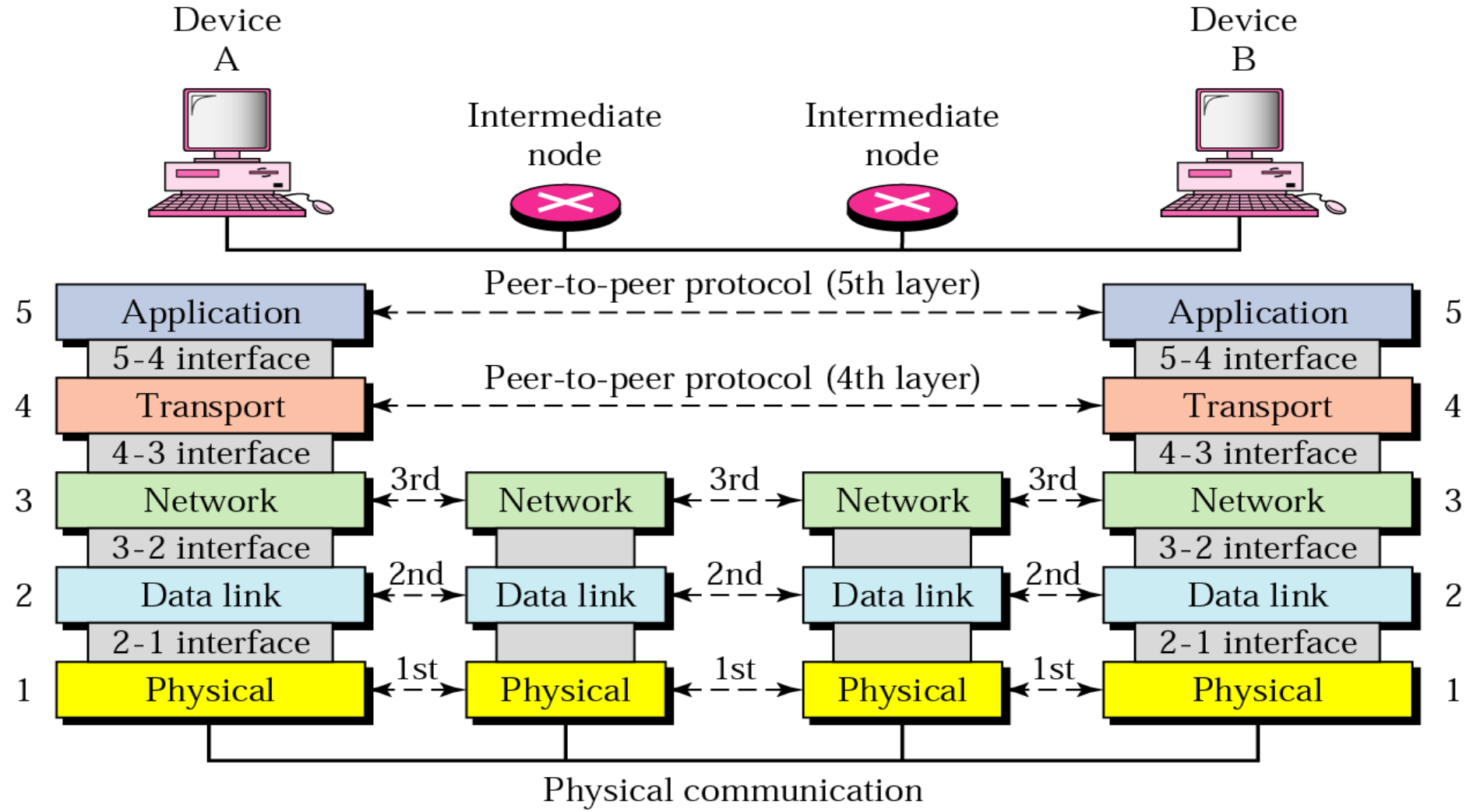
Internet Protocol Stack

- Application: supporting network applications
  - FTP, SMTP, HTTP
- Transport: process-process data transfer
  - TCP, UDP
- Network: routing of datagrams from source to destination
  - IP, routing protocols
- Link: data transfer between neighboring network elements
  - PPP, Ethernet
- Physical: bits "on the wire"

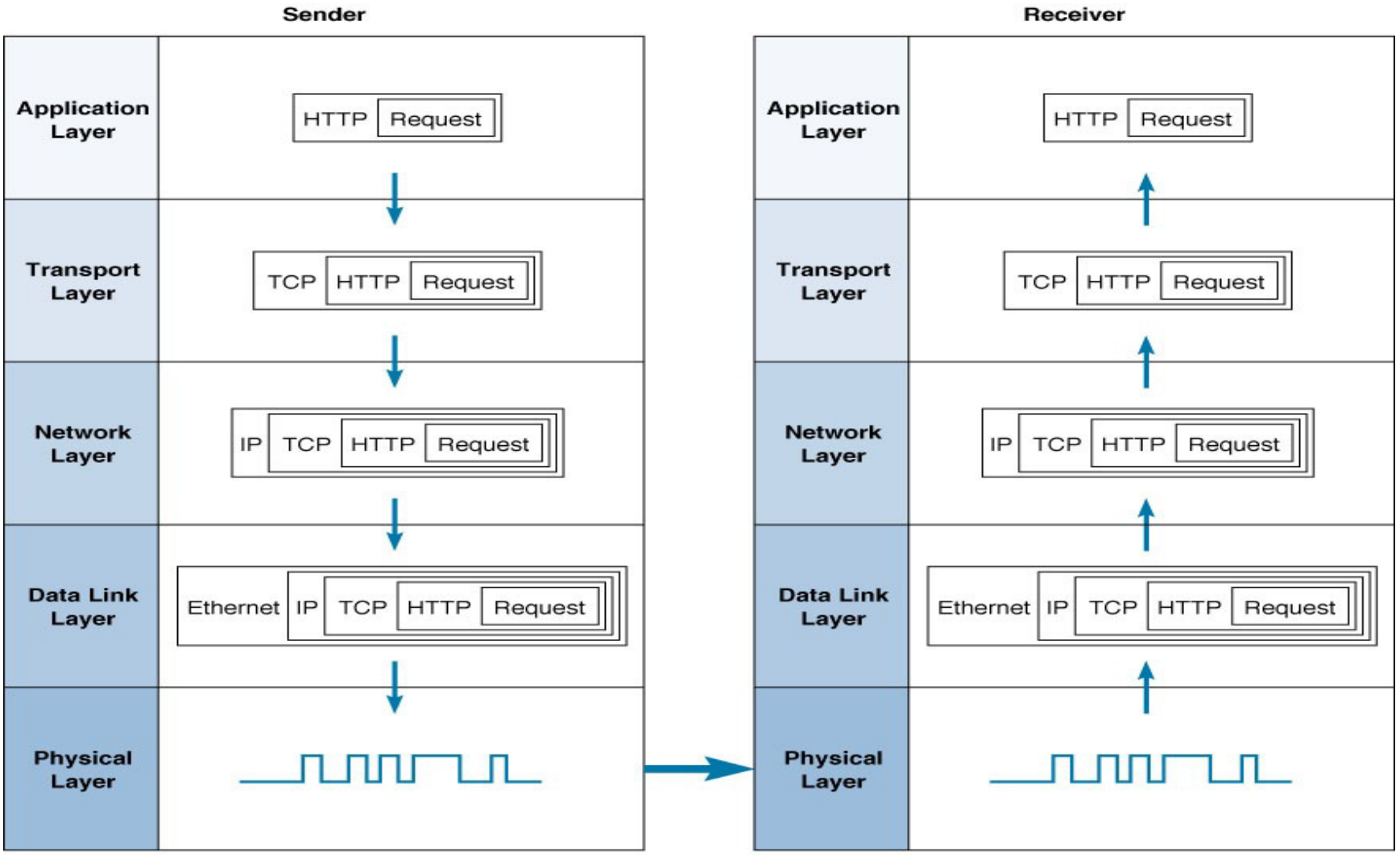


Developed by the US Defence Advanced Research Project Agency (DARPA) for its packet switched network (ARPANET)

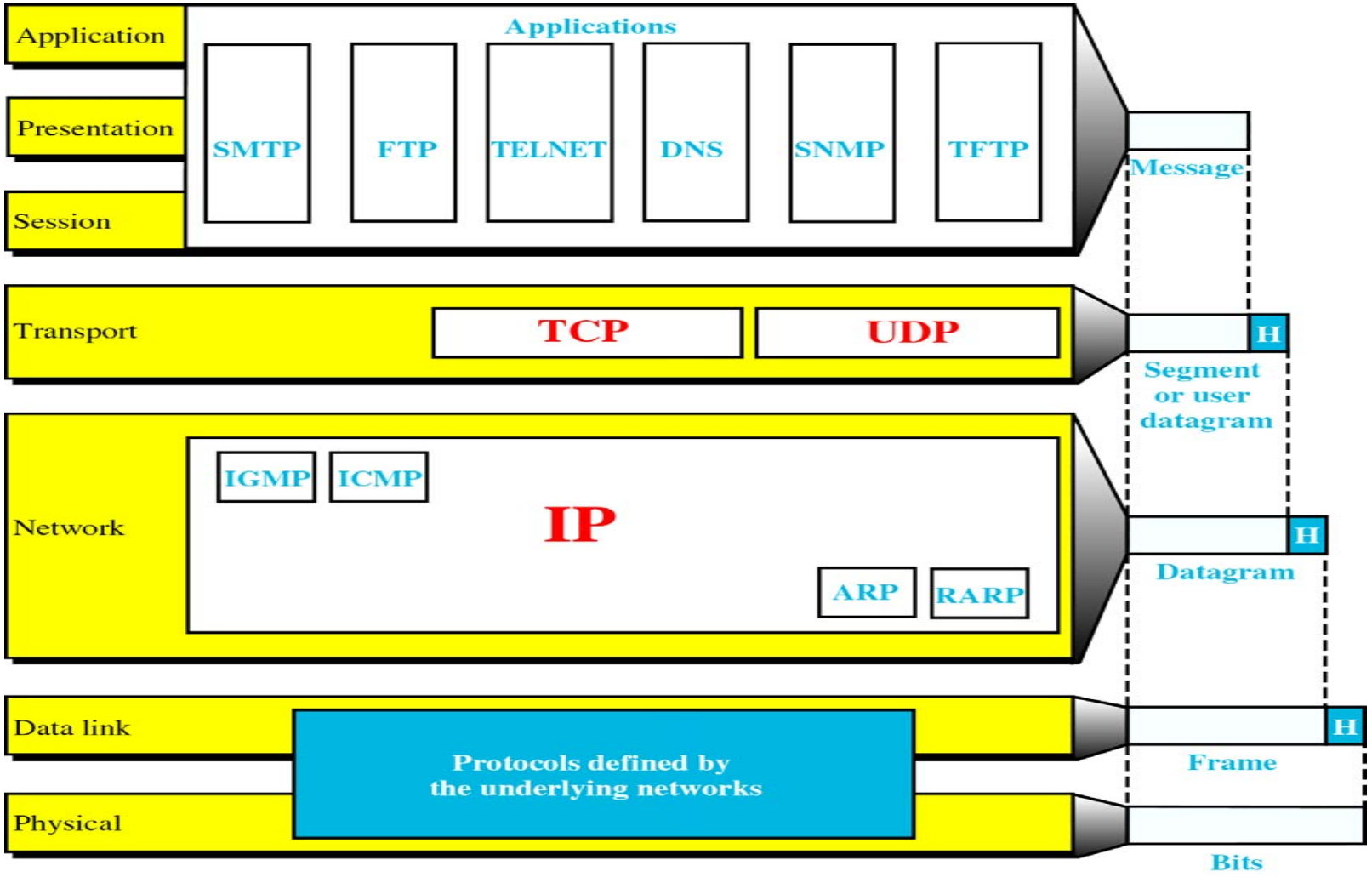
TCP/IP Model


Device A
Device B
Intermediate node
Intermediate node
Peer-to-peer protocol (5th layer)
Peer-to-peer protocol (4th layer)
5
Application
5-4 interface
4
Transport
4-3 interface
3
Network
3-2 interface
2
Data link
2-1 interface
1
Physical
3rd
2nd
1st
Physical communication

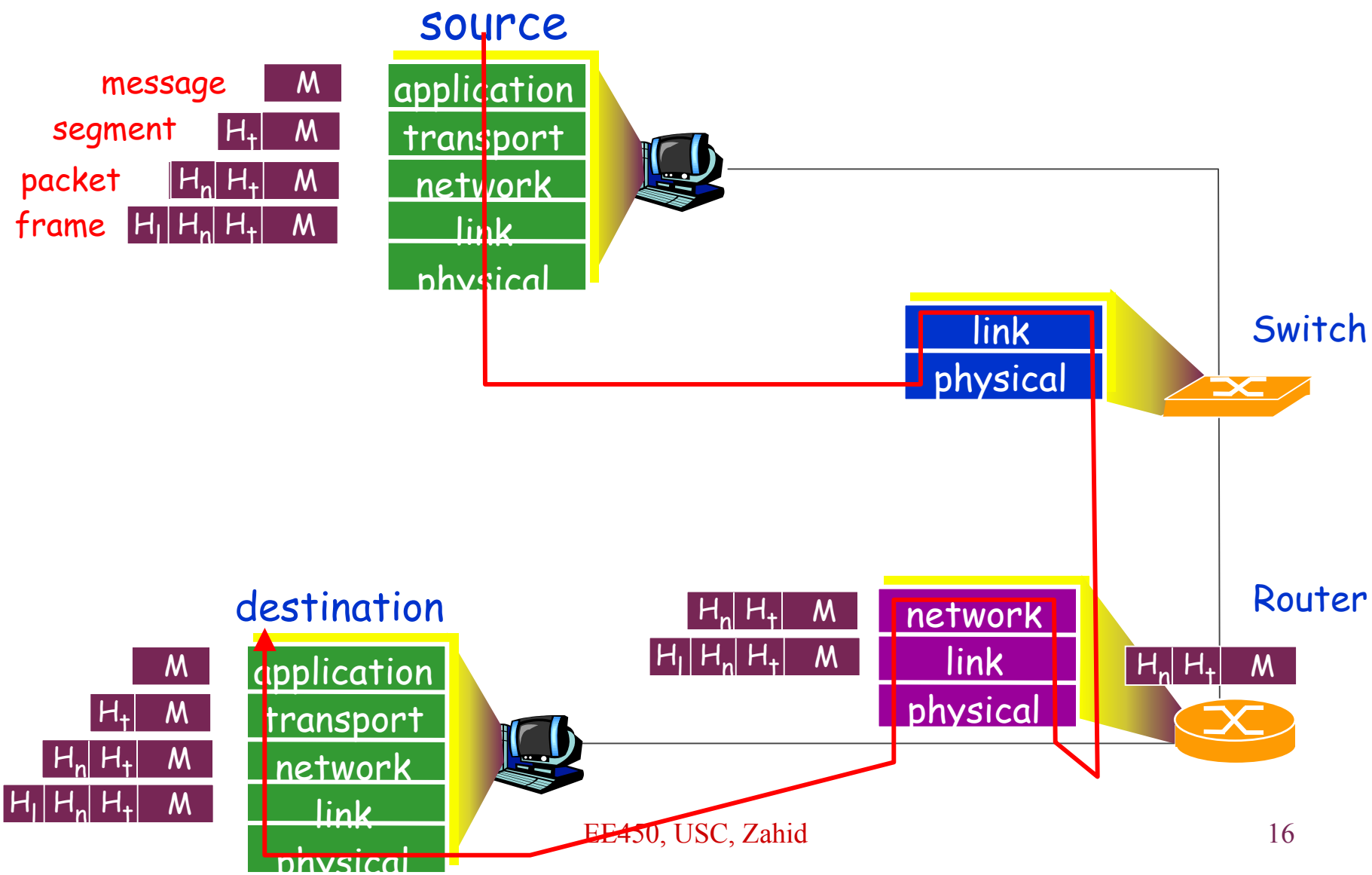
Message Transmission using TCP/IP



TCP/IP vs. OSI Models



Encapsulation



Communications between Networked Applications

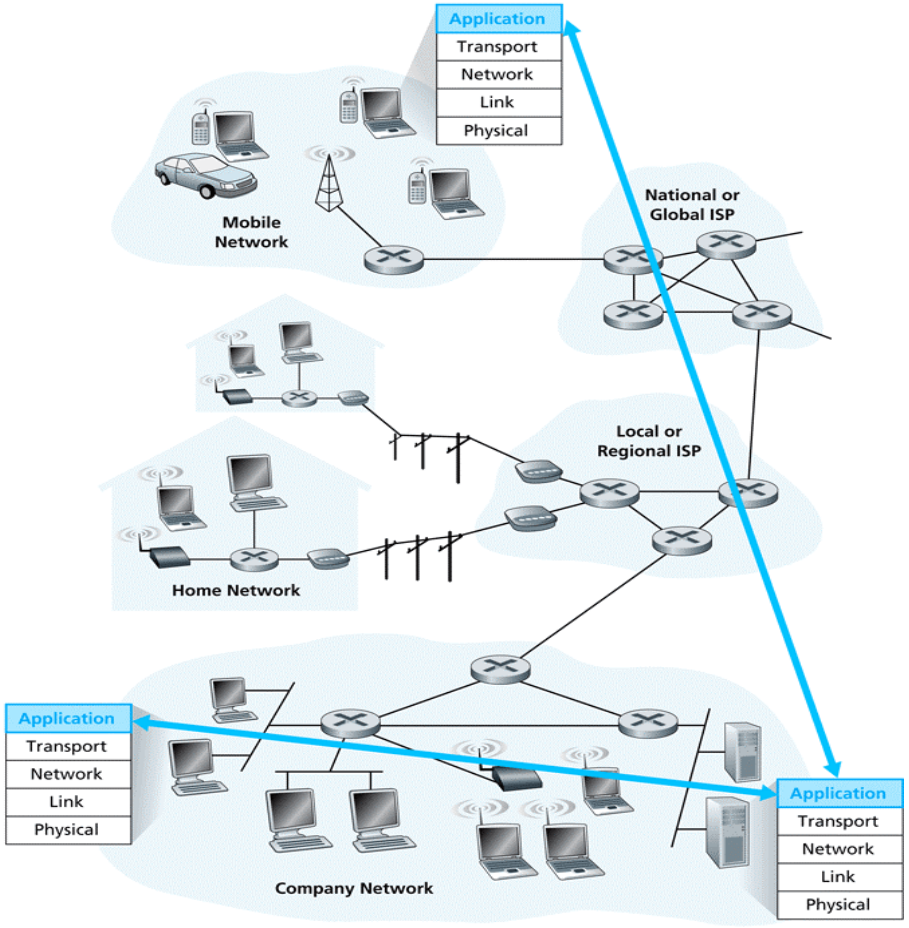


Figure 2.1 ♦ Communication for a network application takes place between end systems at the application layer.

Communications between Transport Layers

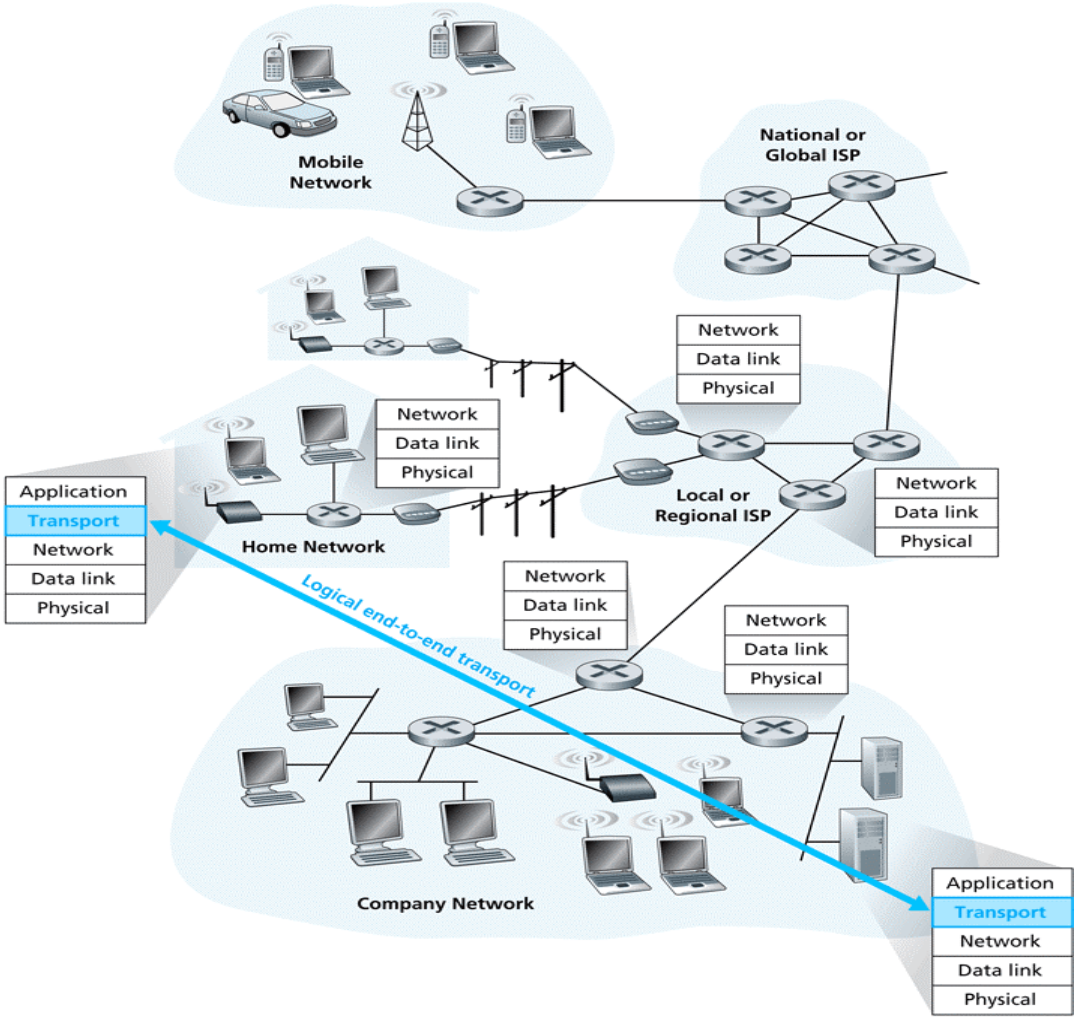


Figure 3.1 ♦ The transport layer provides logical rather than physical communication between application processes.

Network Layer Communications

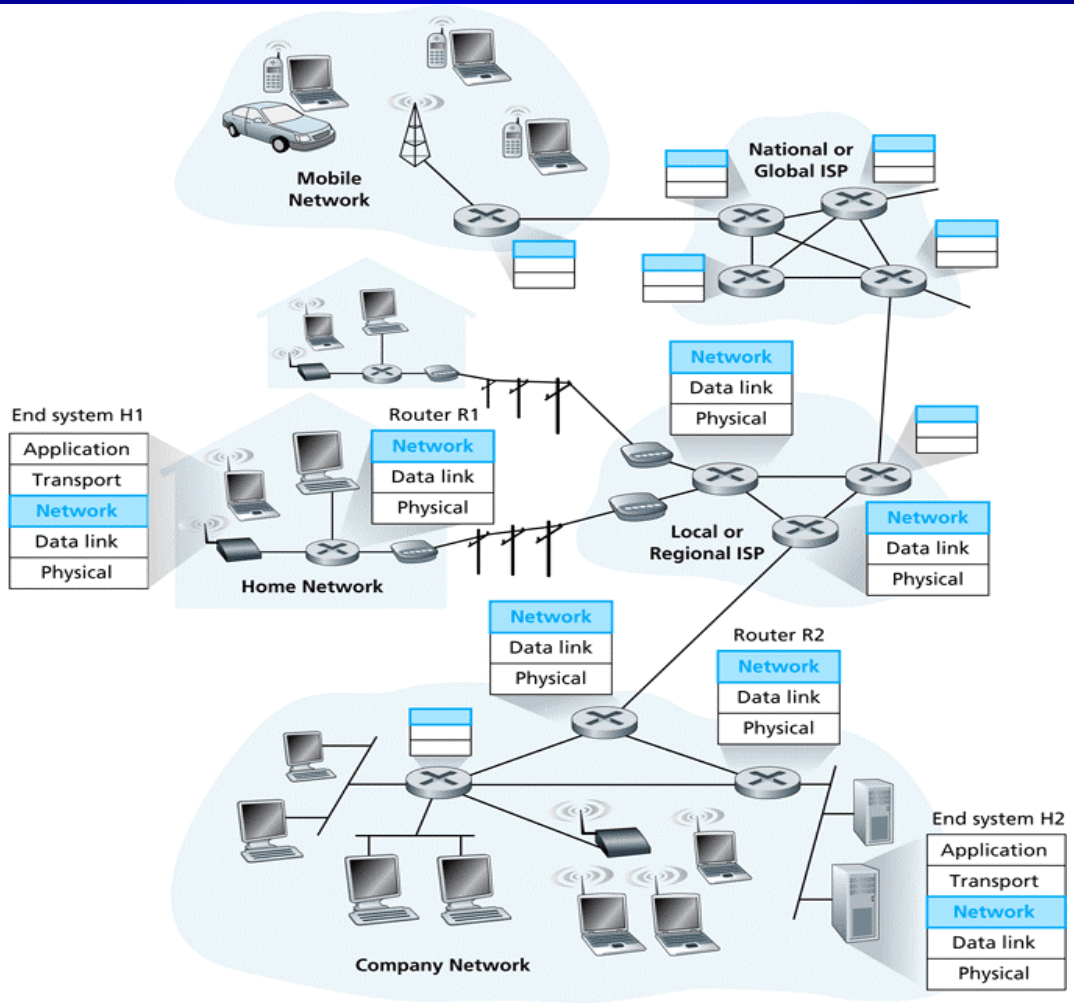


Figure 4.1 ♦ The network layer

Link Layer Communications

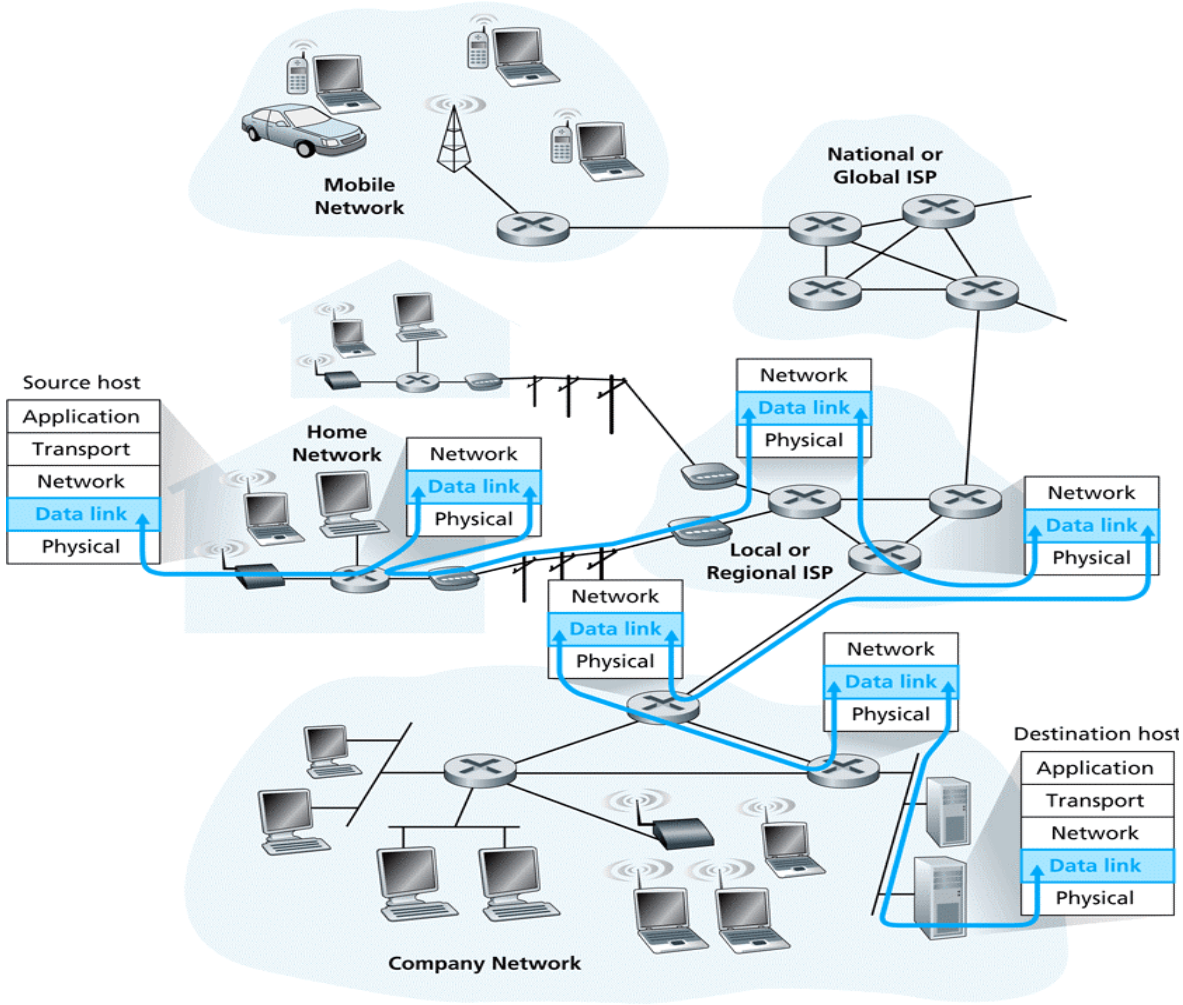
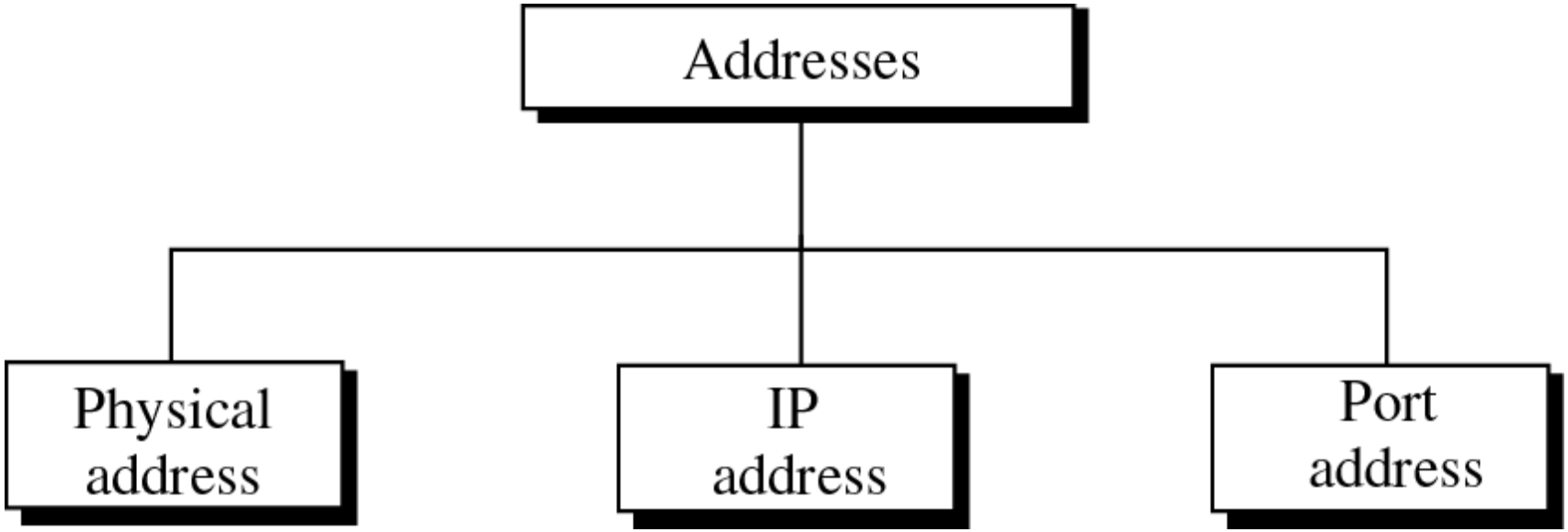
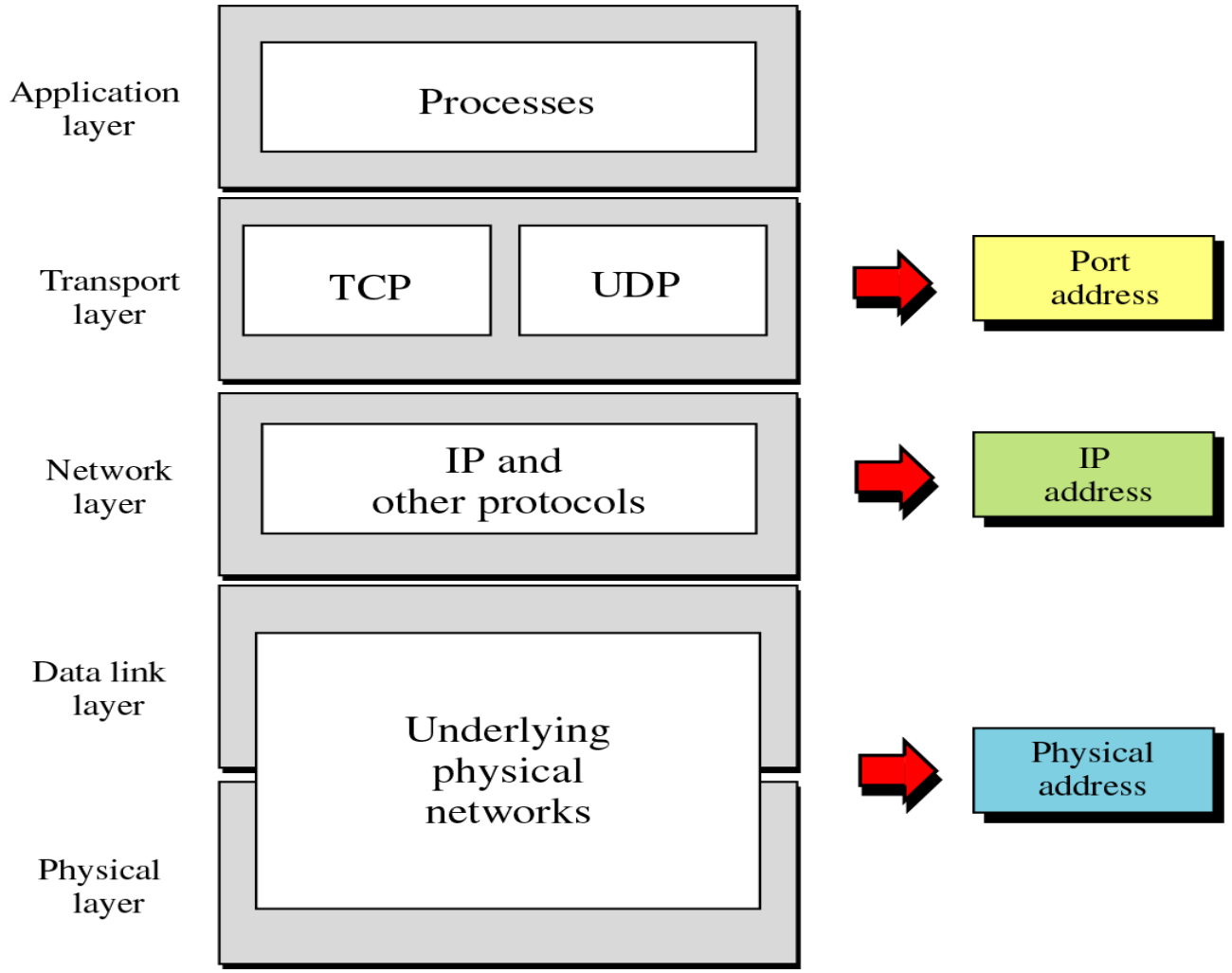


Figure 5.1 ♦ The link layer

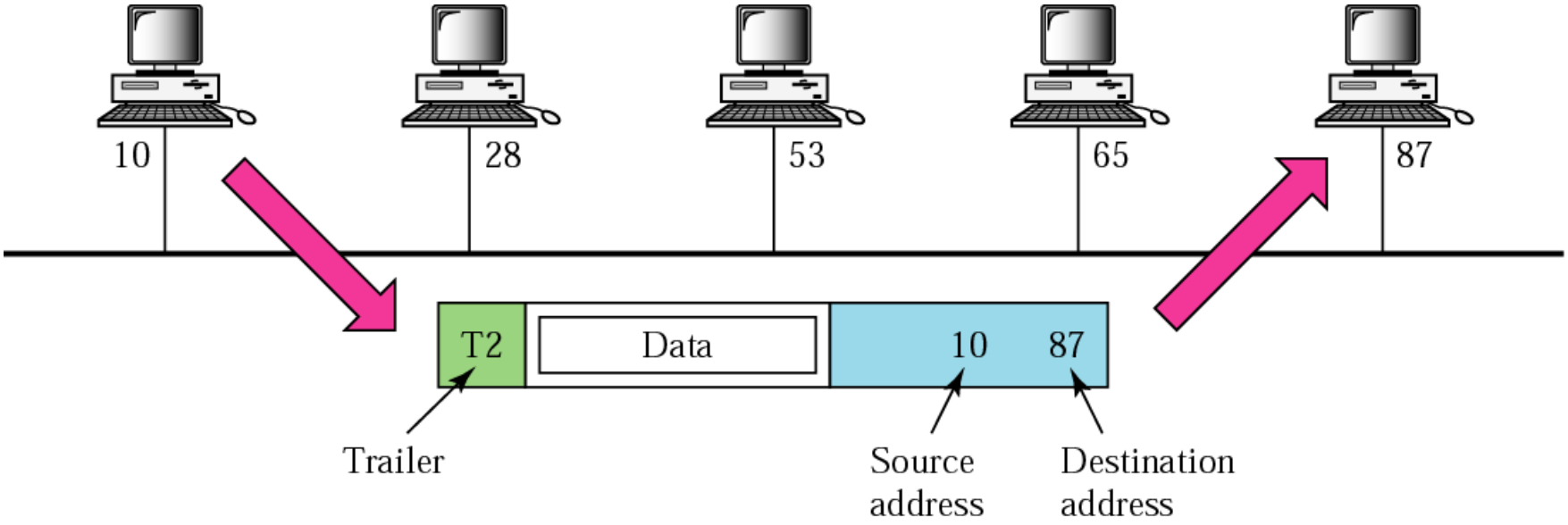
Addressing in TCP/IP



TCP/IP Layers and Addresses



Link Layer (MAC) Addresses

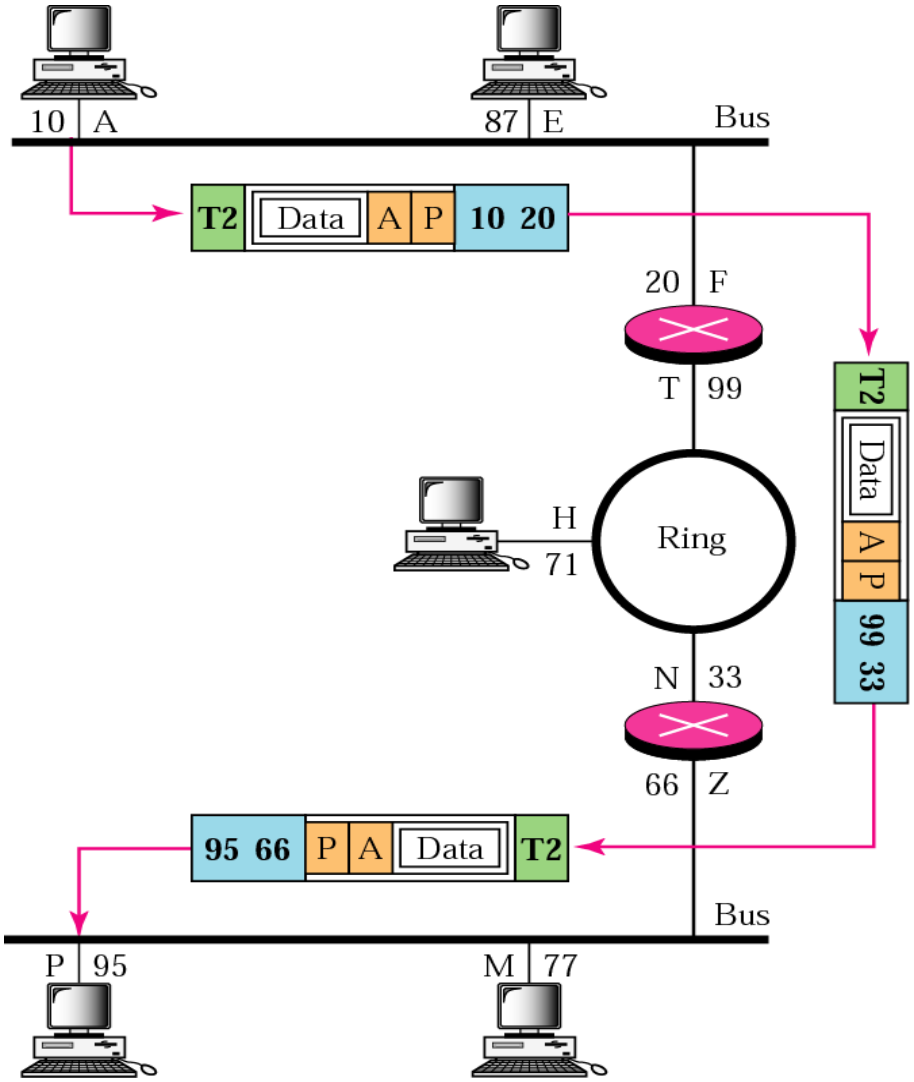


Most local area networks use a 48-bit (6 Bytes) physical address written as 12 hexadecimal digits, with every 2 Bytes separated by a hyphen for example

"07-01-02-01-2C-4B"

a node with MAC address 10 sends a frame to a node with MAC address 87. The two nodes are connected by a link. At the data link level this frame contains MAC addresses in the header. These are the only addresses needed. The header contains other information needed @ this level. The trailer contains extra bits needed for error detection

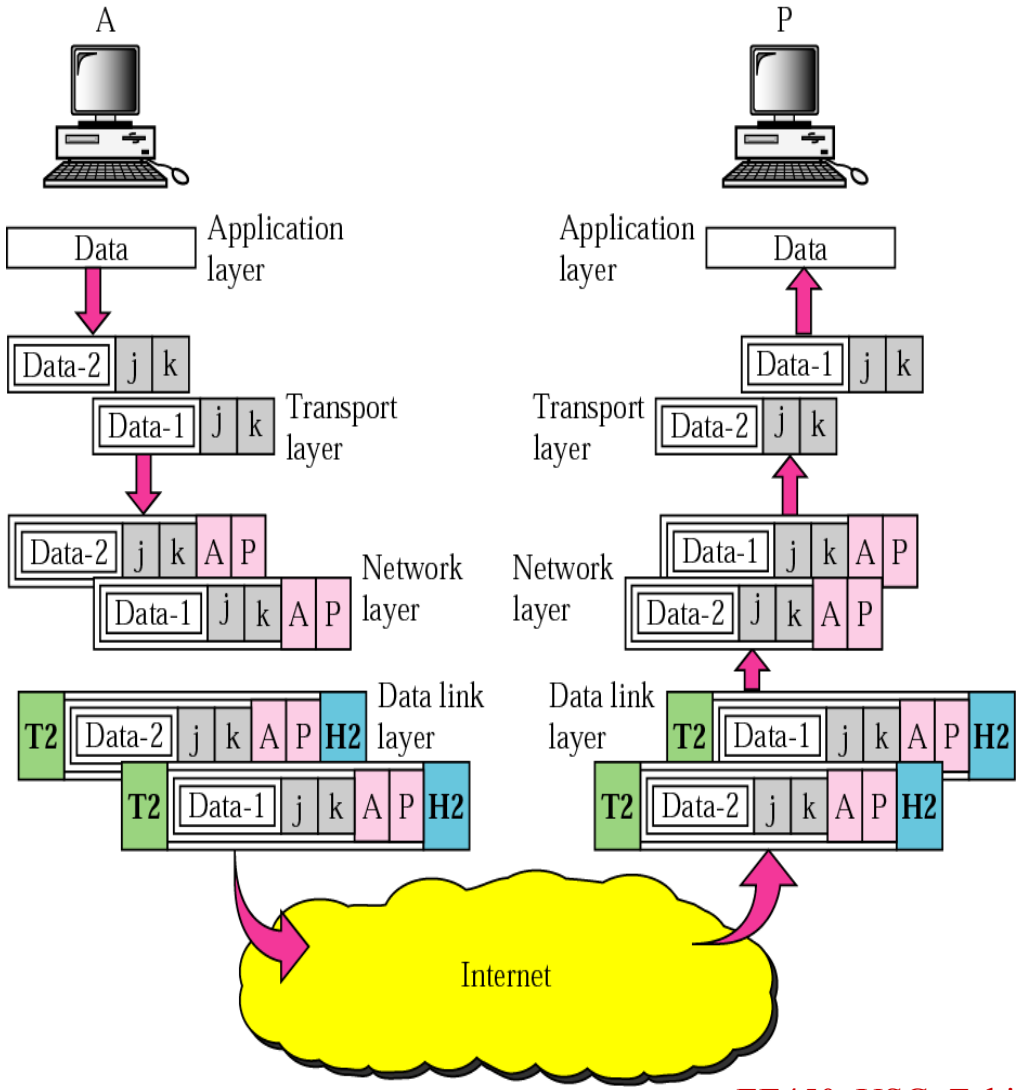
Internetwork Communications



An Internet address (in IPv4) is 32 bits in length written as four decimal numbers, with each number representing 1 Byte. The numbers are separated by a dot. For example 128.125.75.9

A node with an IP address A and MAC address 10, located on one LAN, to a node with an IP address P and MAC address 95, located on another LAN. Because the two devices are located on different networks, we cannot use MAC addresses only; the MAC addresses only have local significance. What we need here are universal addresses that can pass through the LAN boundaries. The IP address has this characteristic.

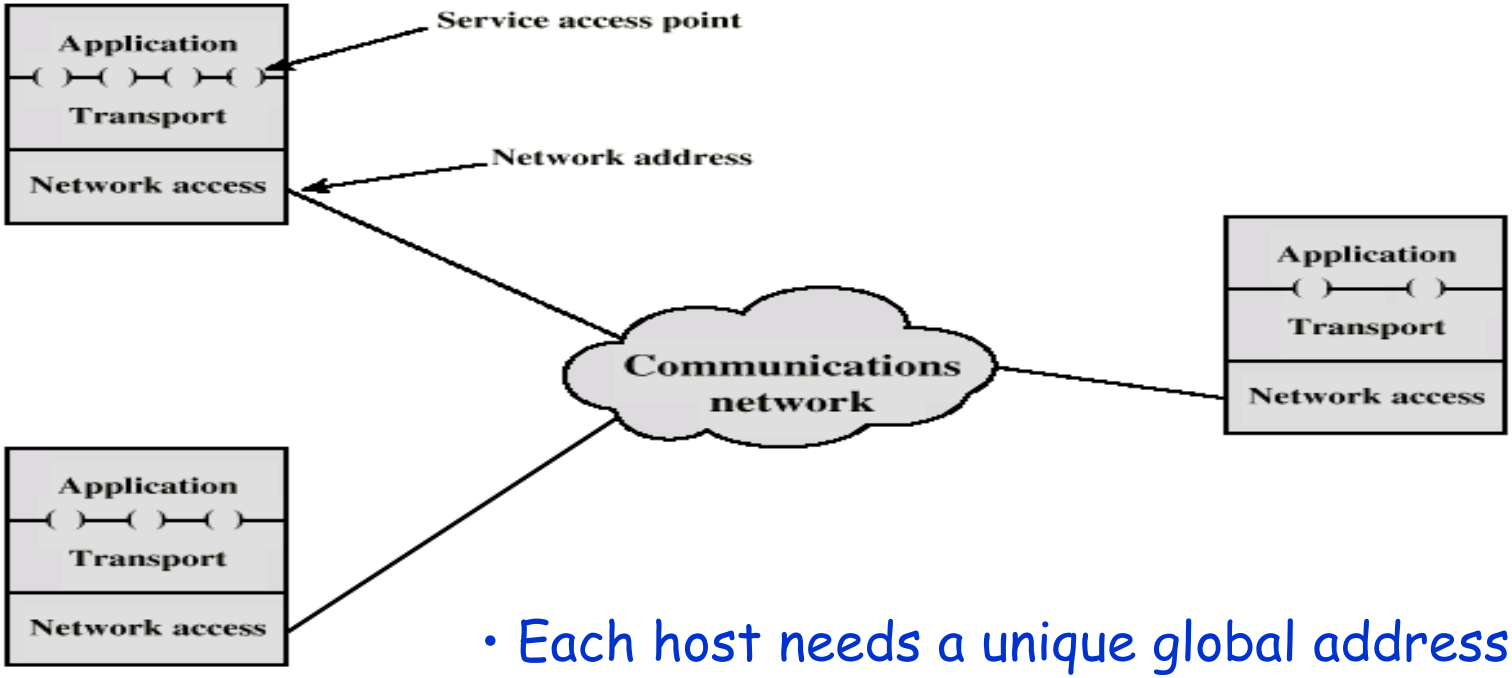
Port Addresses



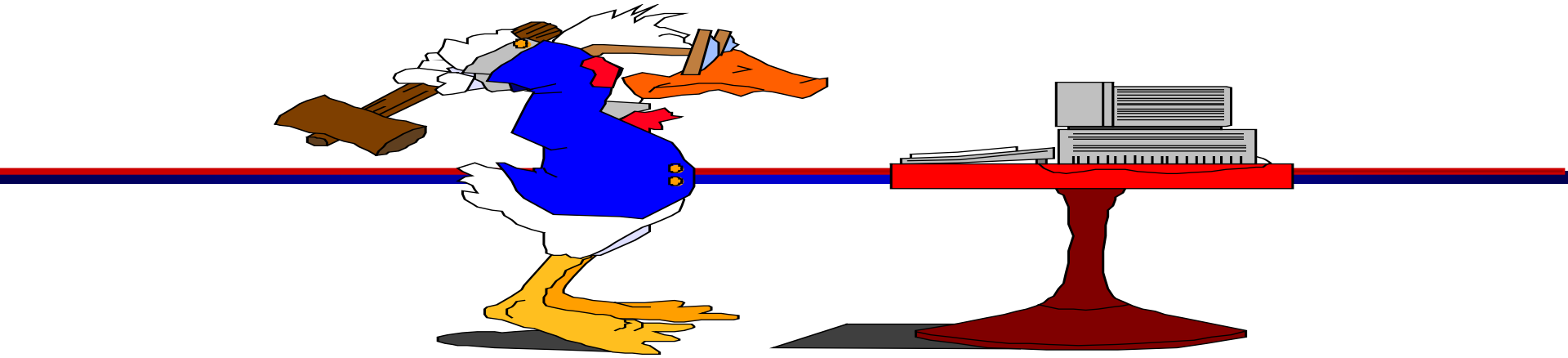
A port address is a 16-bit address represented by one decimal number for example 750

Data coming from the upper layers have port addresses j and k (j is the address of the sending process, and k is the address of the receiving process). Data are split into two Packets, each retaining the port addresses (j and k). Then in the network layer, IP addresses (A and P) are added to each packet.

Layered Architecture and Networks



- Each host needs a unique global address referred to as the IP address
- Each application on a multitasking computer needs a unique address, referred to as the Port number, within the computer



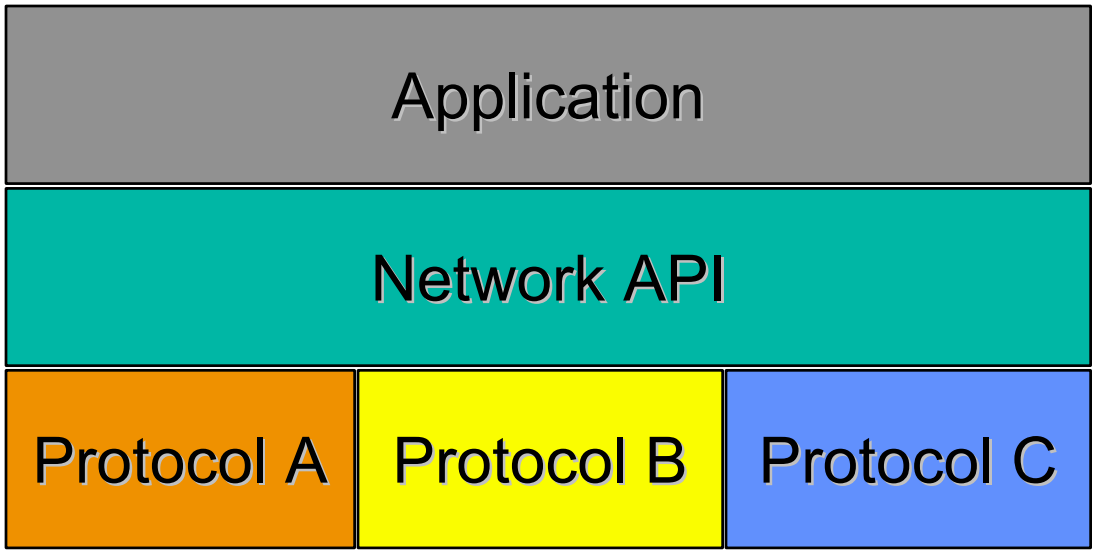
Overview of Network Programming: Sockets

EE450: Introduction to Computer Networks

Professor A. Zahid

Application Programming Interface

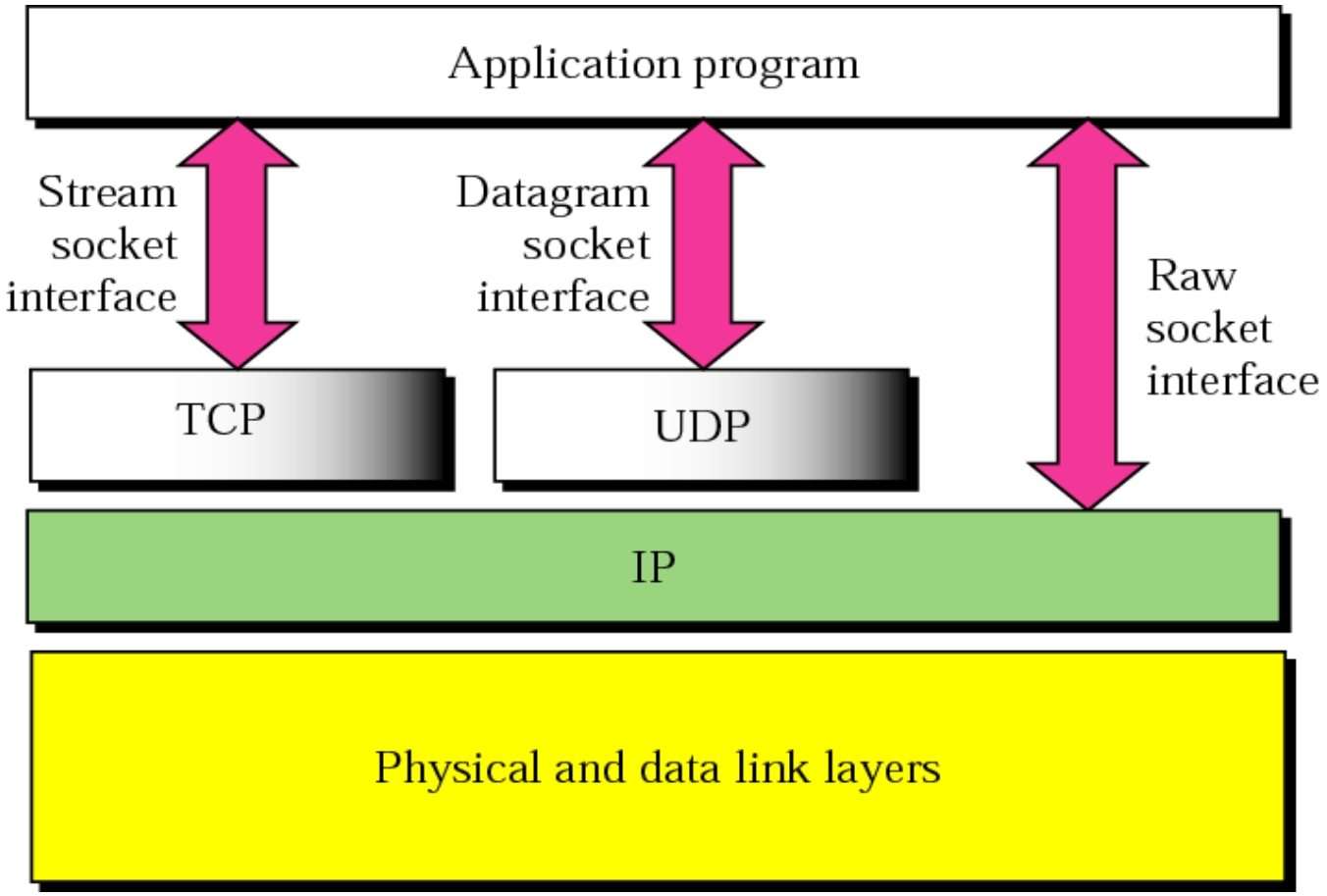
- The services provided by the operating system that provide the interface between application and the TCP/IP Protocol Suite.



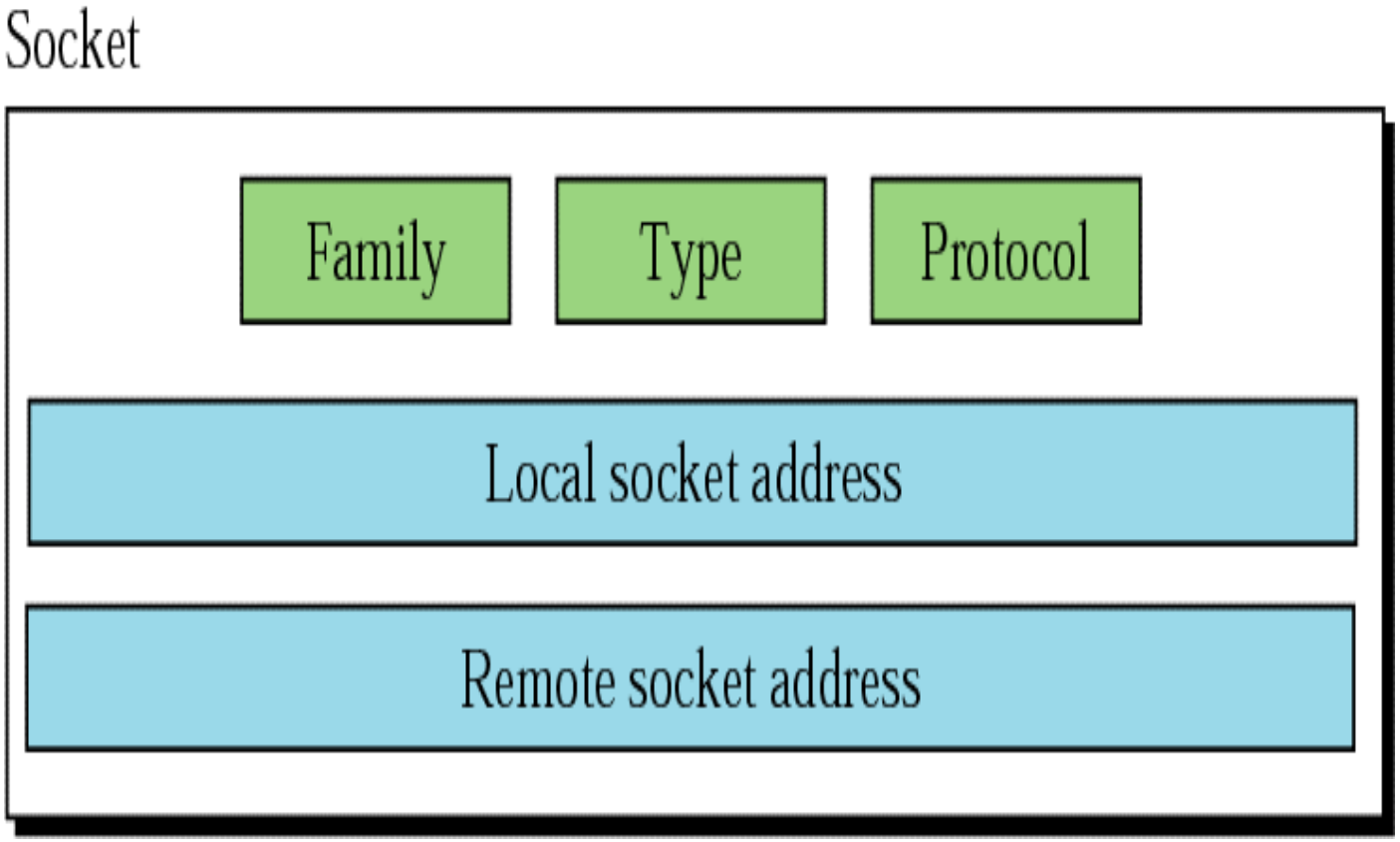
API: Sockets

- TCP/IP does not include an API definition.
- There are a variety of APIs for use with TCP/IP:
  - UNIX Sockets
  - Winsock (Windows Sockets)
- A socket is an abstract representation of a communication endpoint.
- A socket allows the application to "plug in" to the network and communicate with other applications
- A socket is uniquely identified by the IP address, Port number and the underlying transport layer protocol

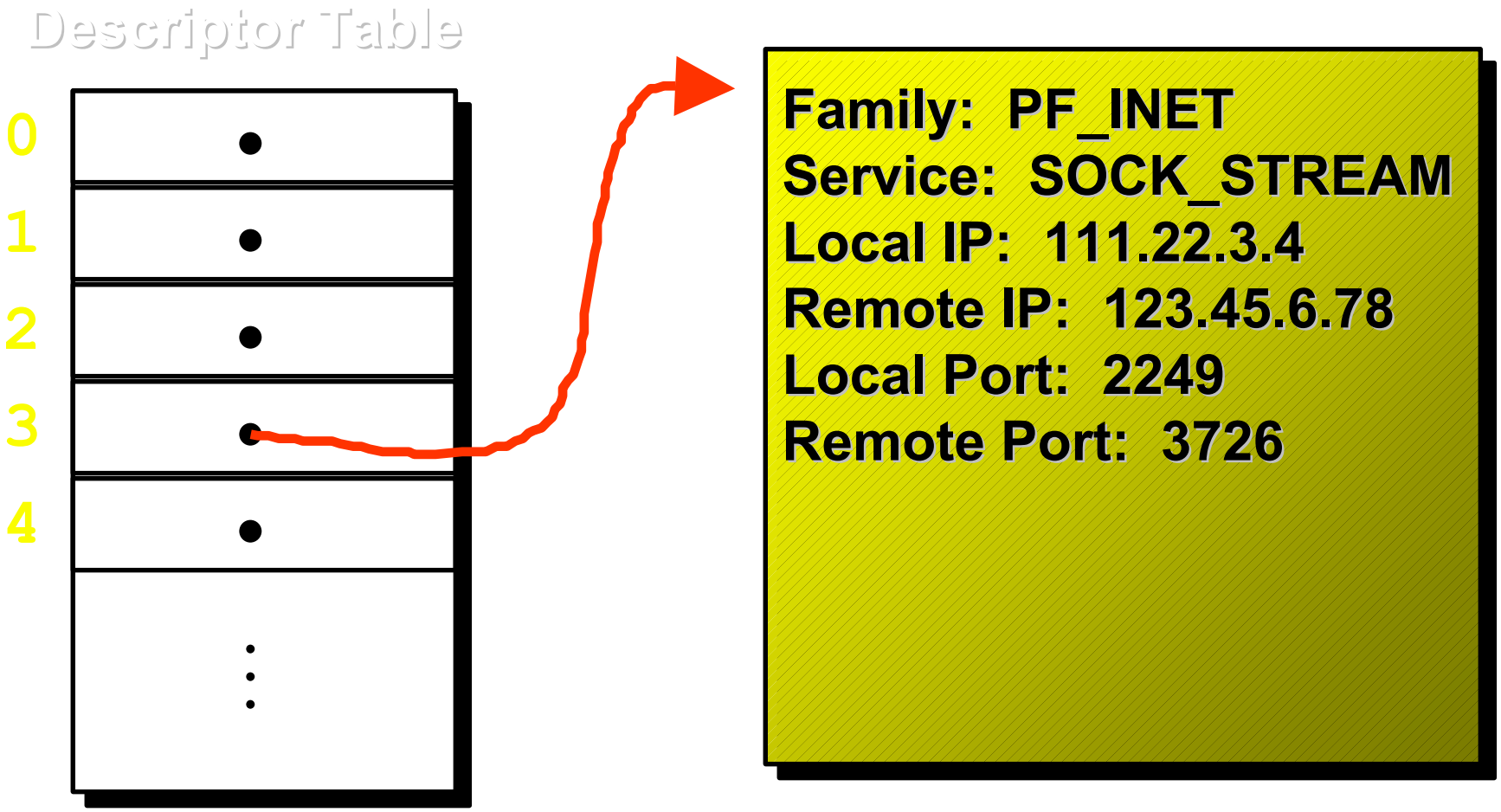
Socket Types



Socket Structure



Socket Descriptor Data Structure



Basic Sockets API

- Clients and Servers differ in some aspects of their use of API and they are the same in other aspects
- Both client and server programs begin by asking the NOS to "create" a socket. The function to accomplish that is Socket ( )
  - int socket (int protocol family, int type, int protocol)
  - Protocol family: PF_INET
  - Type: SOCK_STREAM, SOCK_DGRAM
  - Protocol: TCP, UDP
- Return value of Socket ( ) is a non-negative integer called Socket Descriptor (or –1 if errors)

TCP Server

- Create a TCP socket using Socket ( )
- Assign a port number to the socket using Bind ( )
  - int bind (int socket, local address, address length)
  - Local address is the IP address and the port number of the server. It is this address that the client and the server need to agree on to communicate. Neither one need to know the client address.
  - Address length is length of address structure
  - If successful, Bind ( ) returns a 0, otherwise it returns -1
  - If successful, the socket at server side is "associated " to the local IP address and the local port number
- Listen to connections from clients using Listen ( )
  - int listen (int socket, int queue limit)
  - Queue limit is an upper bound on # of clients that can be waiting
  - If successful, Listen ( ) returns a 0, otherwise it returns -1

TCP Server (Continued)

- The socket that has been bounded to a port and marked for listening is never actually used for sending and receiving. The socket (known as the welcoming socket or the "parent" socket.
- "Child" sockets are created for each client. It is this socket that is actually used for sending and receiving
- Server gets a socket for an incoming client connection by calling Accept ( )
  - int accept (int socket, client address, address length)
  - If successful, accept ( ) returns a descriptor for the new socket, otherwise it returns -1

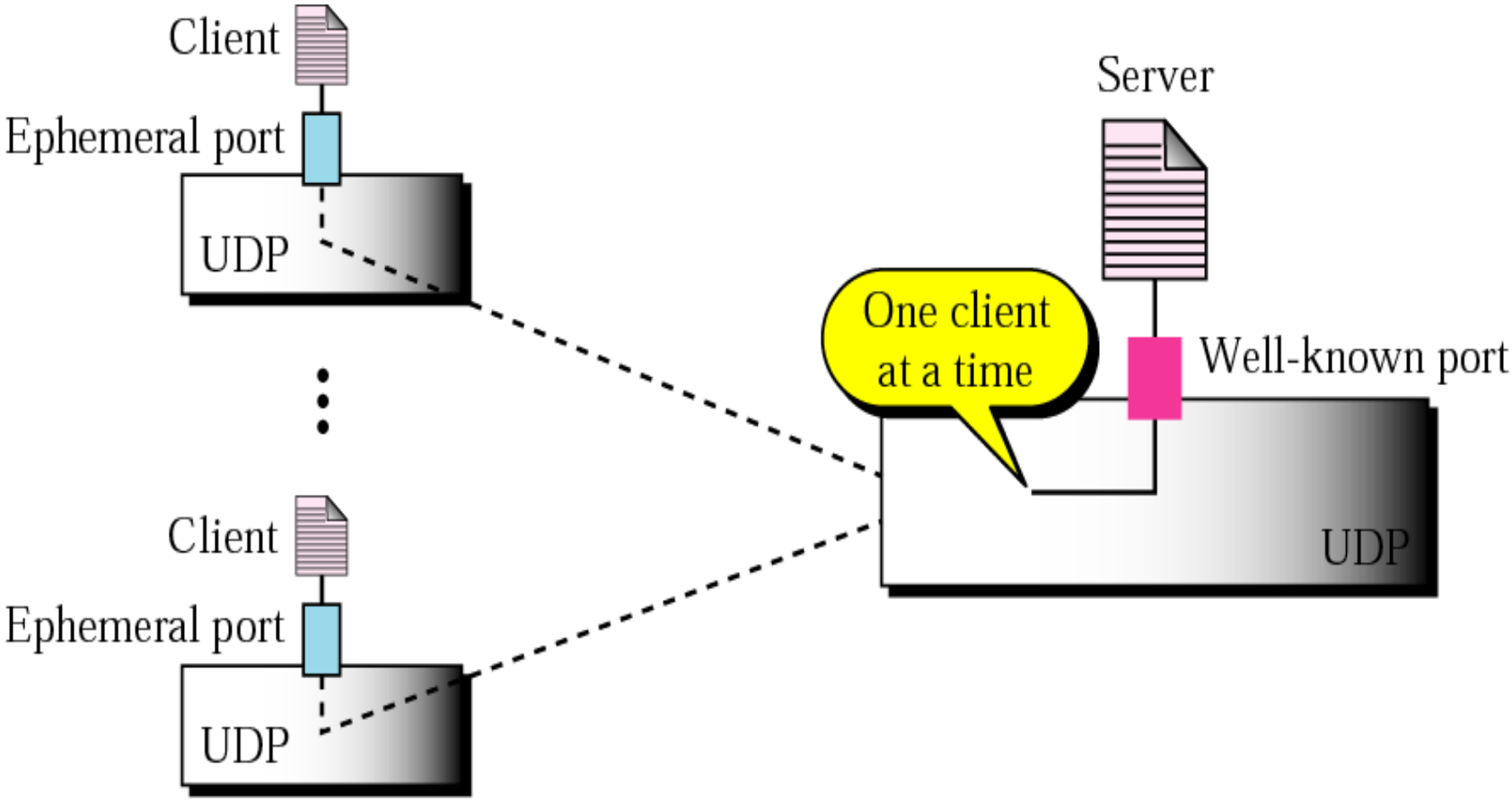
TCP Server (Continued)

- Once the child socket is created, communications (send and receive) can take place
  - int send (int socket, message, message length)
  - int recv (int socket, recv buffer, buffer length)
- When the application is done with the socket, it needs to close it (The child socket, not the parent socket which is passively open to welcome new clients). This is done by calling Close ( )
  - int close (int socket)

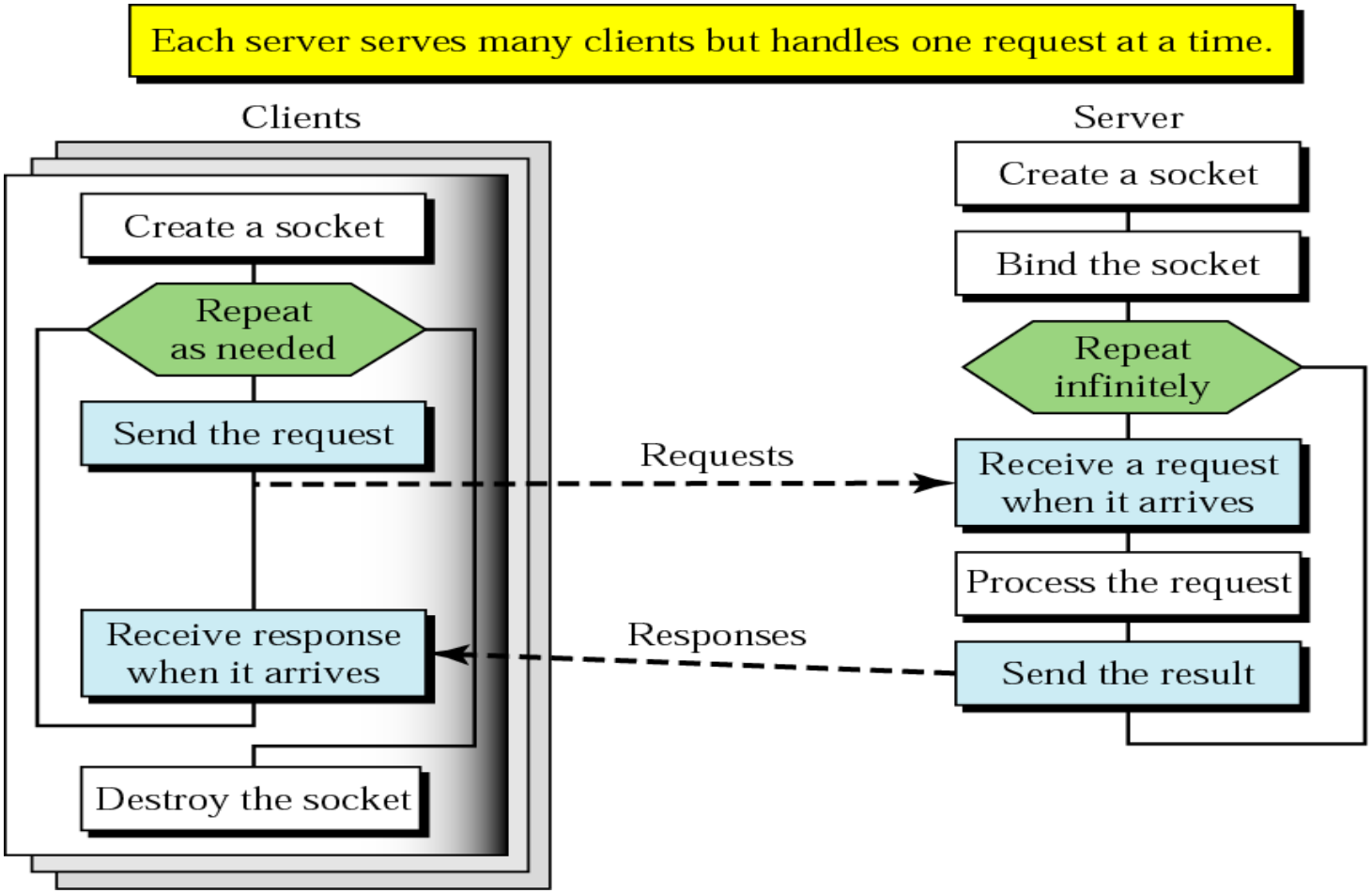
TCP Client

- Create a TCP socket using Socket ( )
- Establish a connection to the server using Connect ( )
  - int connect (int socket, foreign address, address length)
  - Foreign address is the address of the server and the port number of the server (The well-known port number)
- Communications using Send and Recv
  - int send (int socket, message, message length)
  - int recv (int socket, recv buffer, buffer length)
- Close the socket using Close ( )
  - int close (int socket)

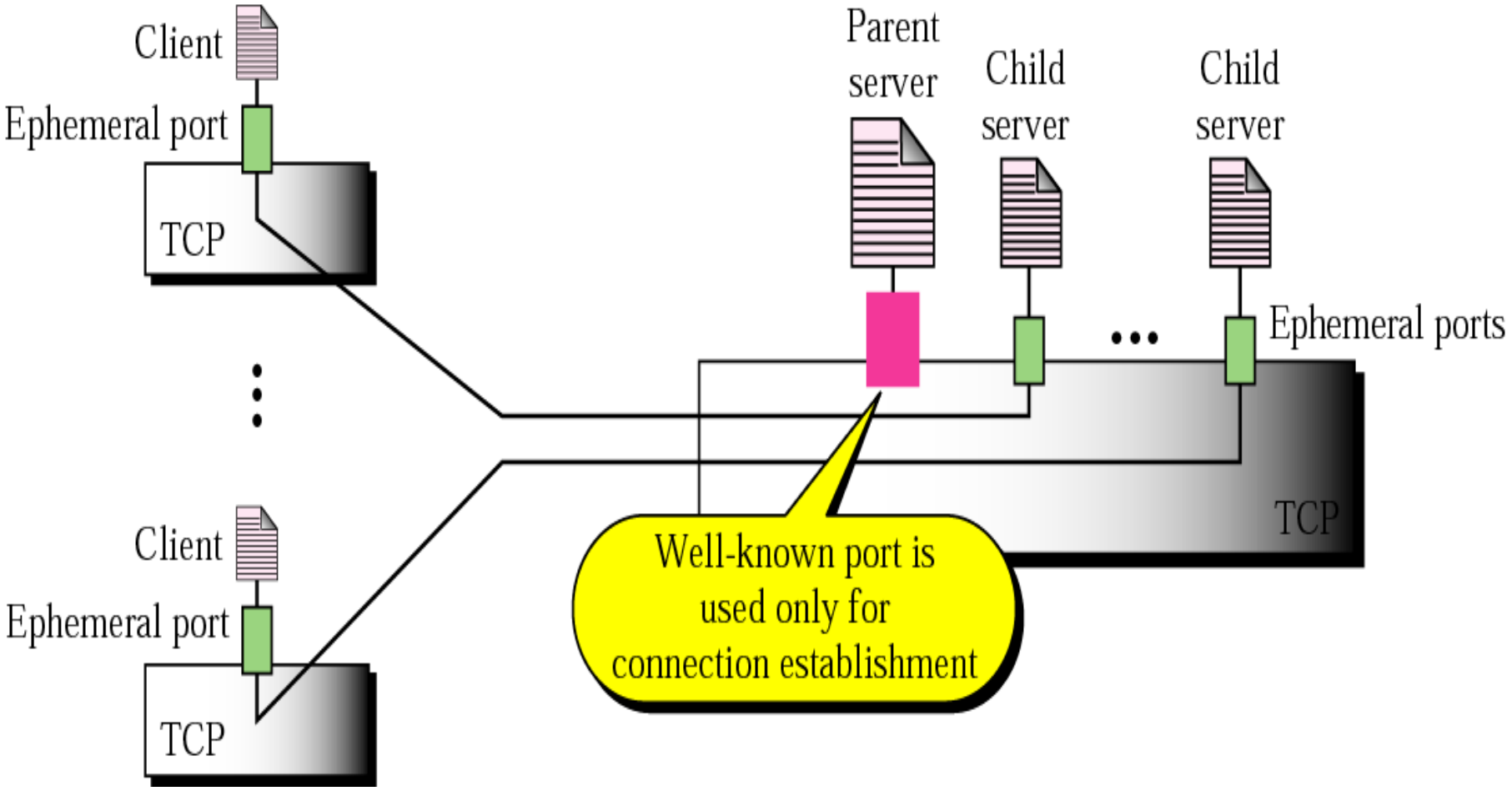
Connection-less Iterative Server



Flow Chart: Socket Interface for Connection-Less Iterative Server



Connection-oriented Concurrent Server



Flow Chart: Socket Interface for Connection-Oriented Concurrent Server