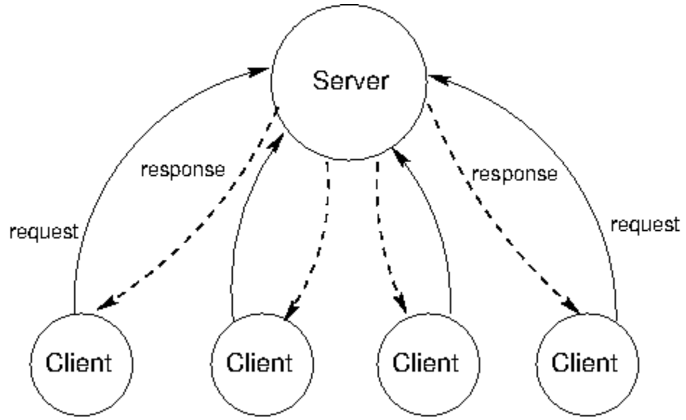
COMMUNICATION

CS435 Distributed Systems

Basit Qureshi PhD, FHEA, SMIEEE, MACM
https://www.drbasit.org/

TOPICS

- Communication in Distributed Systems: An overview
- Networking: A quick Review
- TCP and UDP Sockets



WHY COMMUNICATION

WHY COMMUNICATION

1.Coordination:

- Distributed systems consist of multiple independent components (nodes, servers, or processes)
- Components work together to achieve a common goal.
- Communication facilitates coordination and synchronization among these components.

2.Data Exchange:

- Data needs to be shared among different components in REAL-TIME to perform tasks or make decisions.
- Communication enables the seamless exchange of data between nodes.

3.Fault Tolerance:

- Distributed systems are designed to be resilient to failures, such as node crashes or network issues.
- Communication helps in fault detection, isolation, and recovery.
- Nodes update their status and share information about the health of the system.

WHY COMMUNICATION

4. **Consistency and Replication**
    - Update replicas and synchronize data to ensure that all nodes in the system have a consistent view.

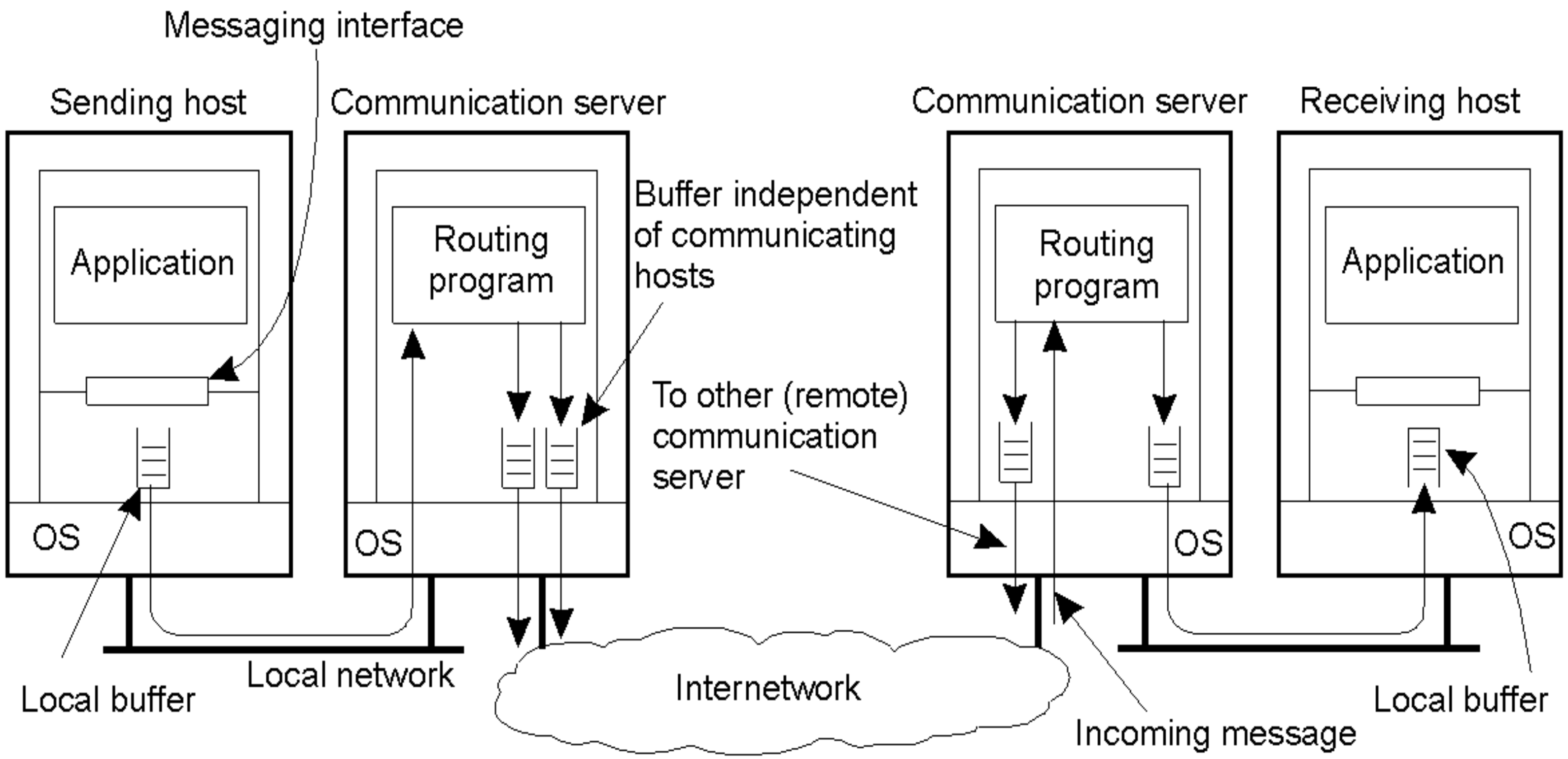
5. **Load Balancing**
    - Tasks can be distributed among nodes to ensure optimal resource utilization and performance.

6. **Scalability:**
    - Horizontal scaling by adding more nodes to the network.
    - Communication enables these new nodes to join the system, share the workload, and contribute to the overall capacity and performance.

7. **Efficiency:**
    - Effective communication can optimize resource usage and reduce unnecessary duplication of efforts.
    - Nodes can collaborate efficiently, sharing resources and information to achieve common goals.

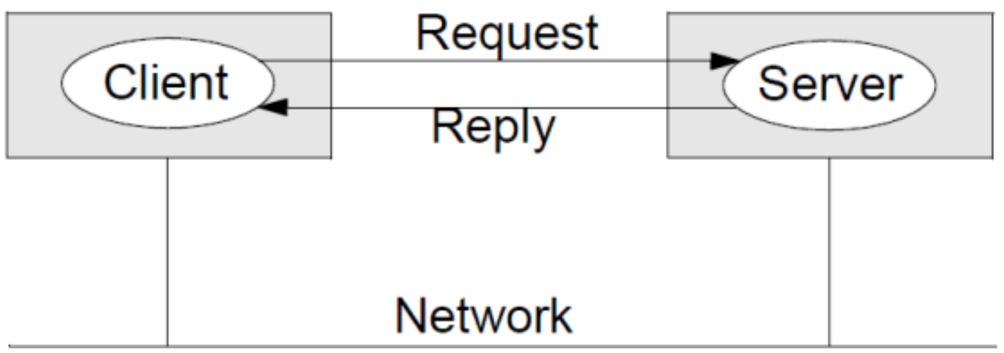
WHY COMMUNICATION

- Communication is an integral aspect of distributed systems, facilitating coordination, data exchange, fault tolerance, consistency, scalability, and adaptability.
- It ensures that the distributed components can work together effectively to achieve the system's *goals* in a *collaborative* and *coordinated* manner.



WHY COMMUNICATION

- The system is structured as a group of processes (objects), called **servers**, that deliver services to **clients**.



The client:

...
send **(request)** to server_reference;

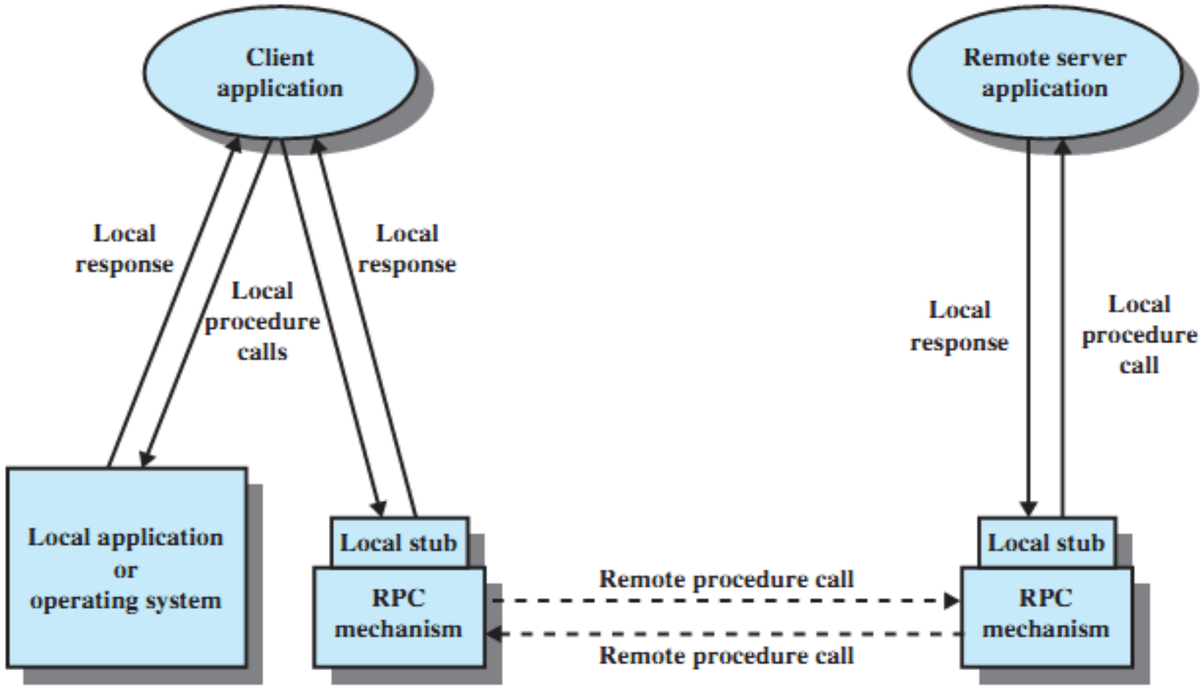
receive **(reply);**
...

The server:

...
receive **(request)** from client-reference;
execute requested operation
send (reply) to client_reference;
...

COMMUNICATION IN DIST SYS

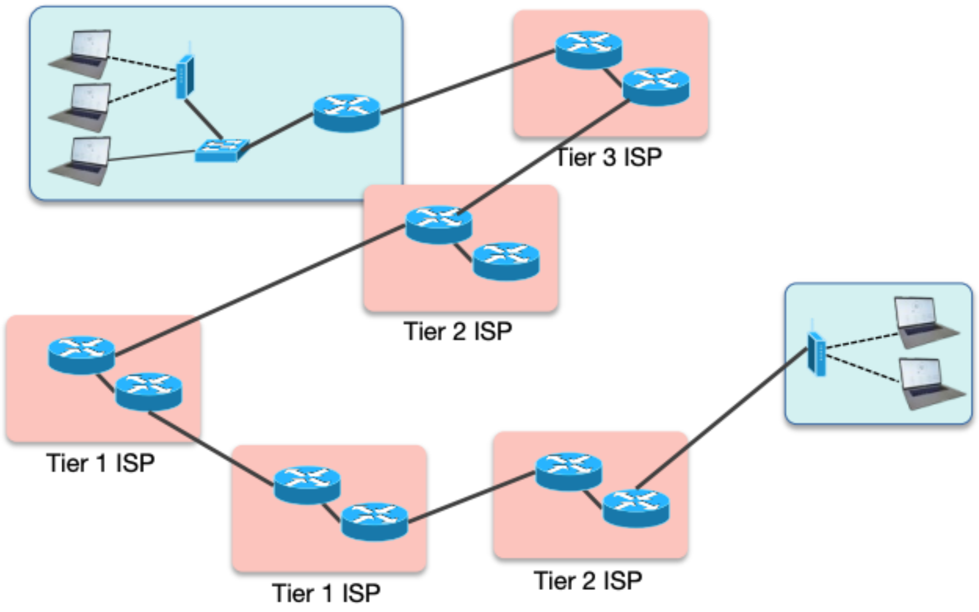
- Communication between distributed objects by means of two models:
  - **Remote Method Invocation (RMI)**
  - **Remote Procedure Call (RPC)**
- RMI, as well as RPC, are implemented on top of request and reply primitives.
- Request and reply are implemented on top of the network protocol (e.g. TCP or UDP in case of the internet)



NETWORKING: A QUICK REVIEW

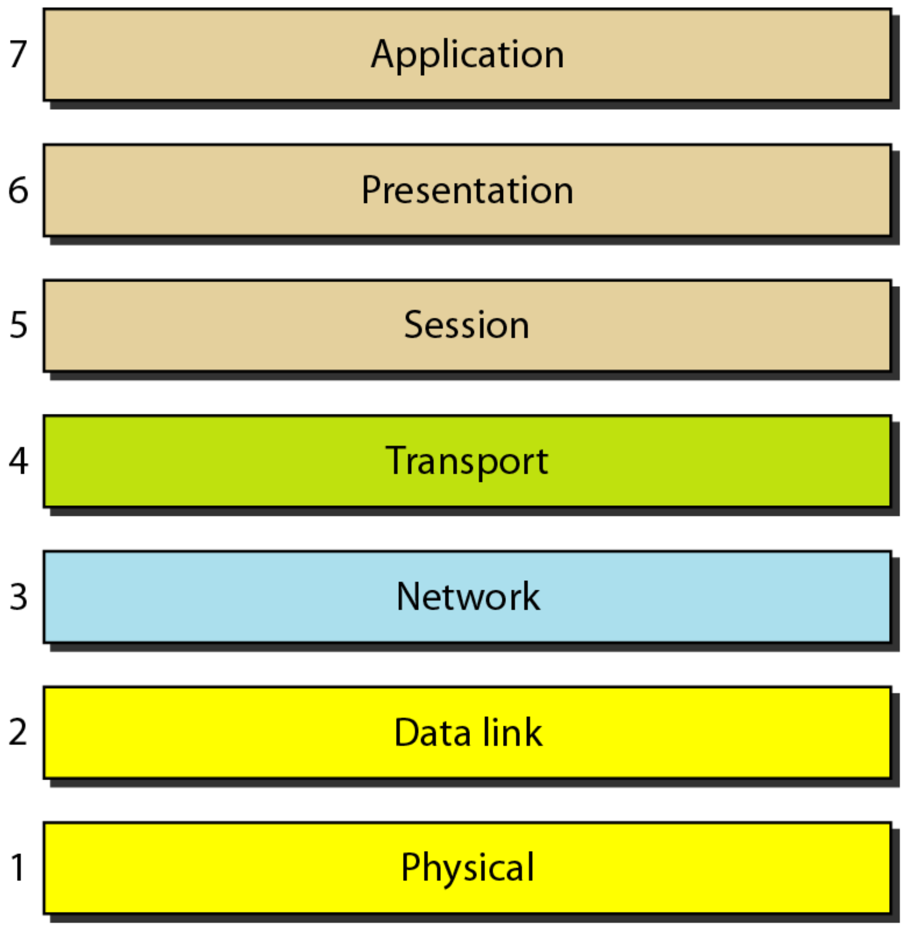
NETWORKING: A QUICK REVIEW

- Data is broken down into tiny packets.
- Packets are sent over the Internet
- The Internet is a interconnect of networking devices (Routers, Switches, Servers, Computers etc)



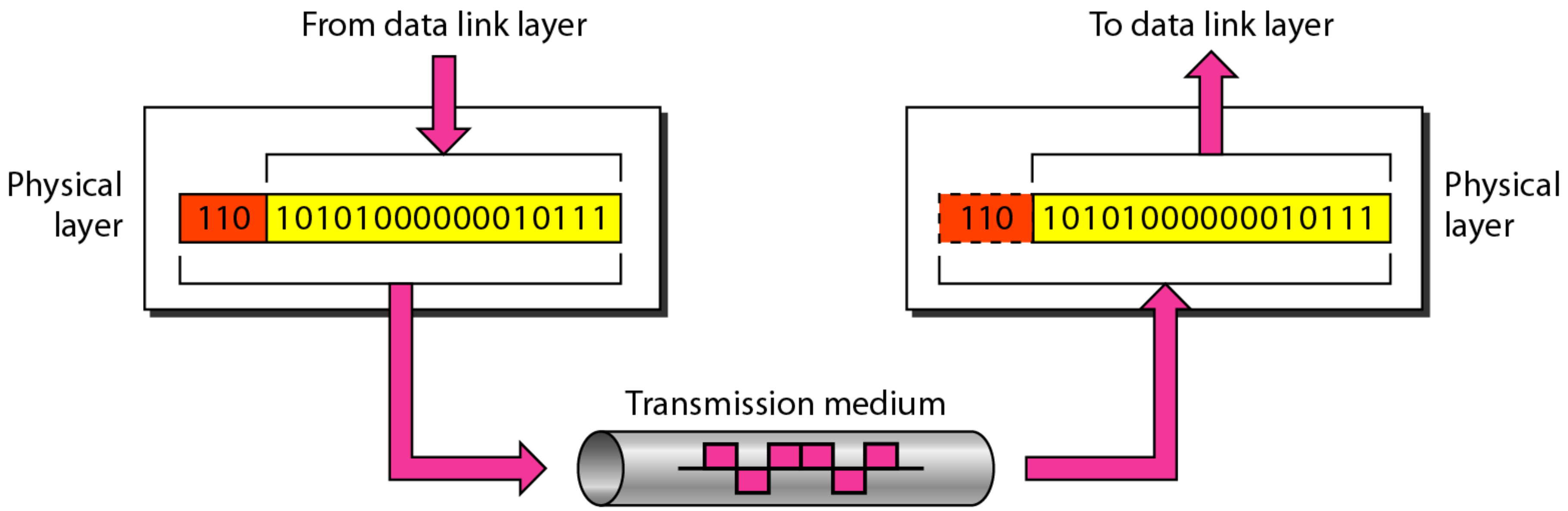
NETWORKING: A QUICK REVIEW

- The ISO Open Systems Interconnection (OSI) model. (1970s)



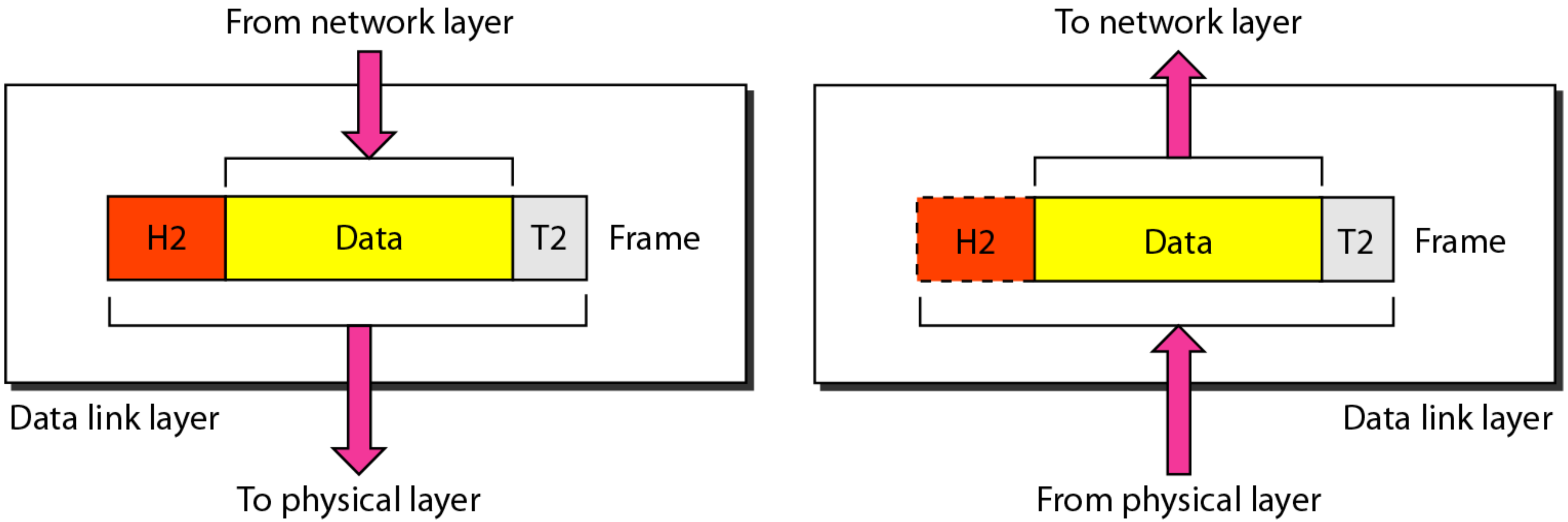
NETWORKING: A QUICK REVIEW

- 1. Physical Layer
- The physical layer is responsible for movements of individual bits from one hop (node) to the next.

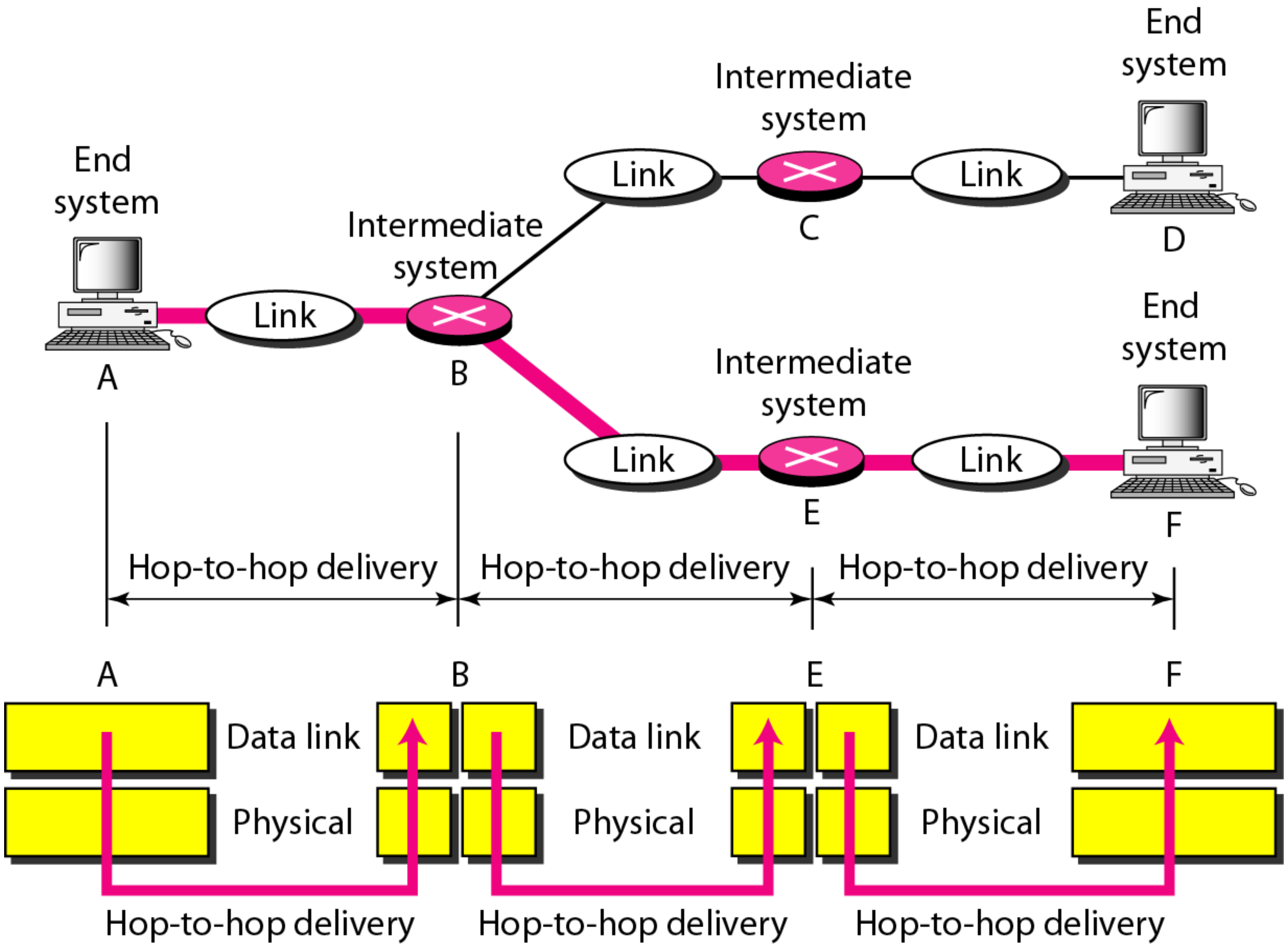


NETWORKING: A QUICK REVIEW

- 2. Data Link Layer
- The data link layer is responsible for moving frames from one hop (node) to the next.

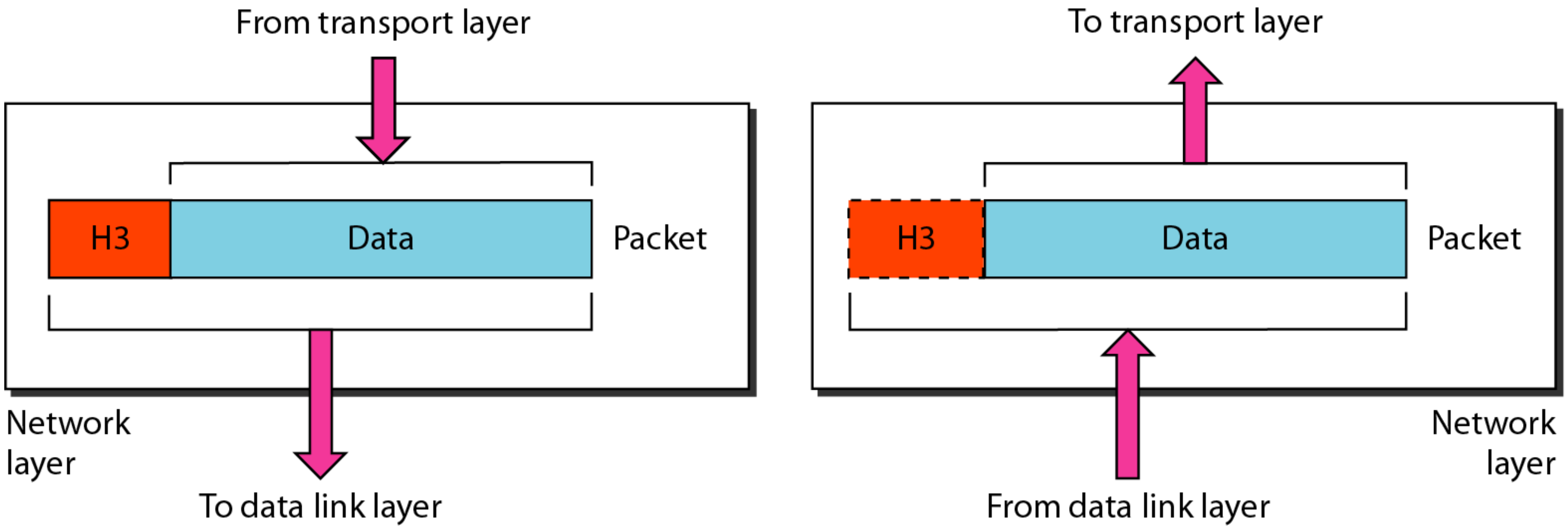


NETWORKING: A QUICK REVIEW

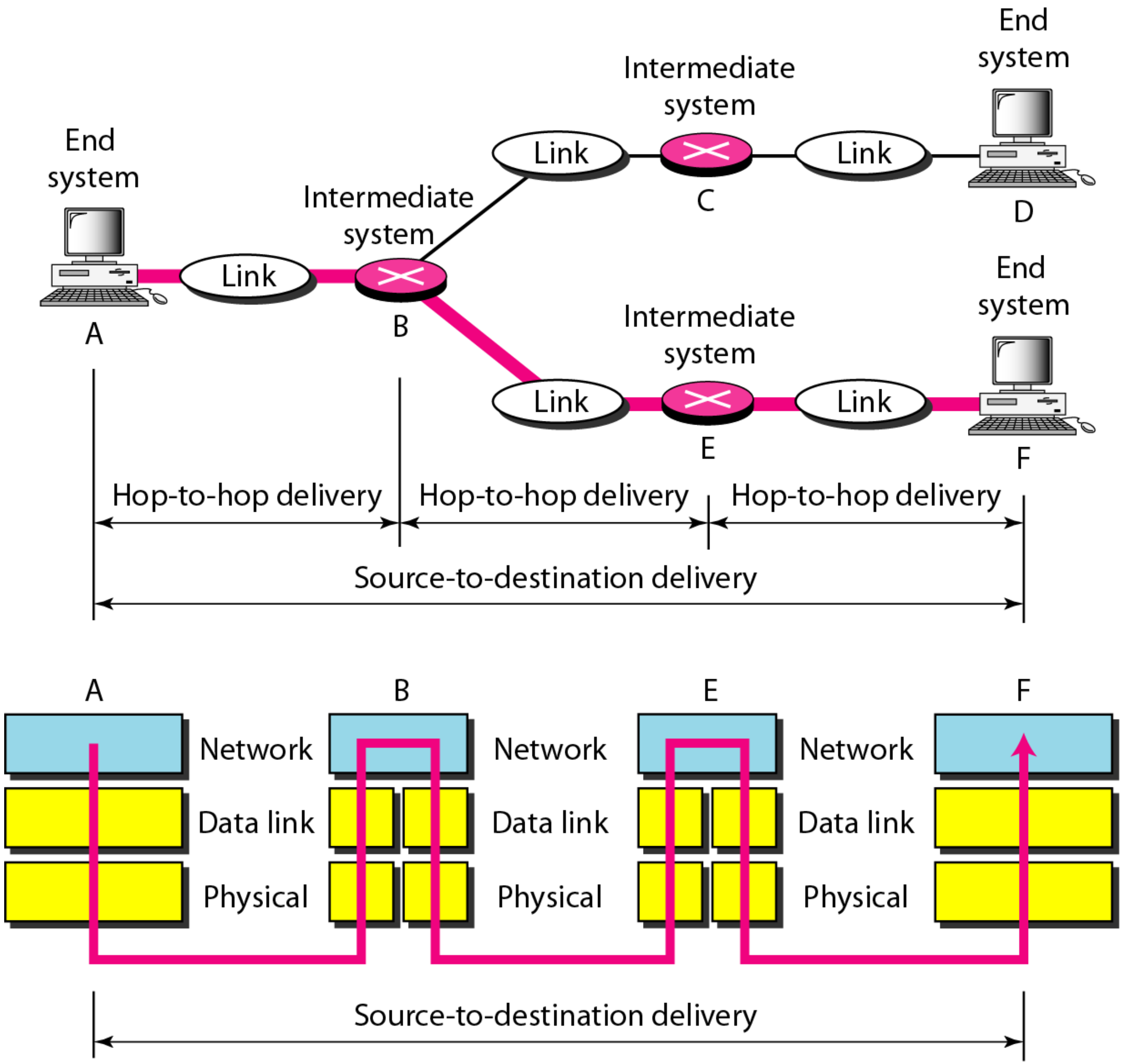


NETWORKING: A QUICK REVIEW

- Network Layer
- The network layer is responsible for the delivery of individual packets from the source host to the destination host.

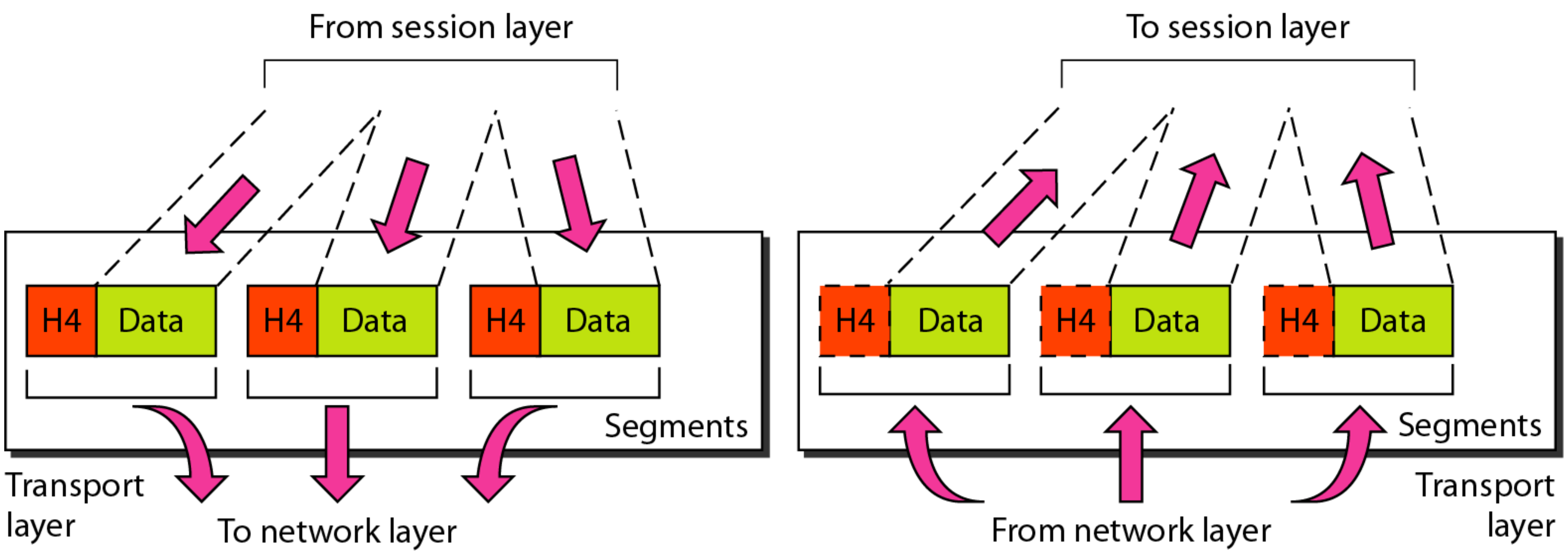


NETWORKING: A QUICK REVIEW

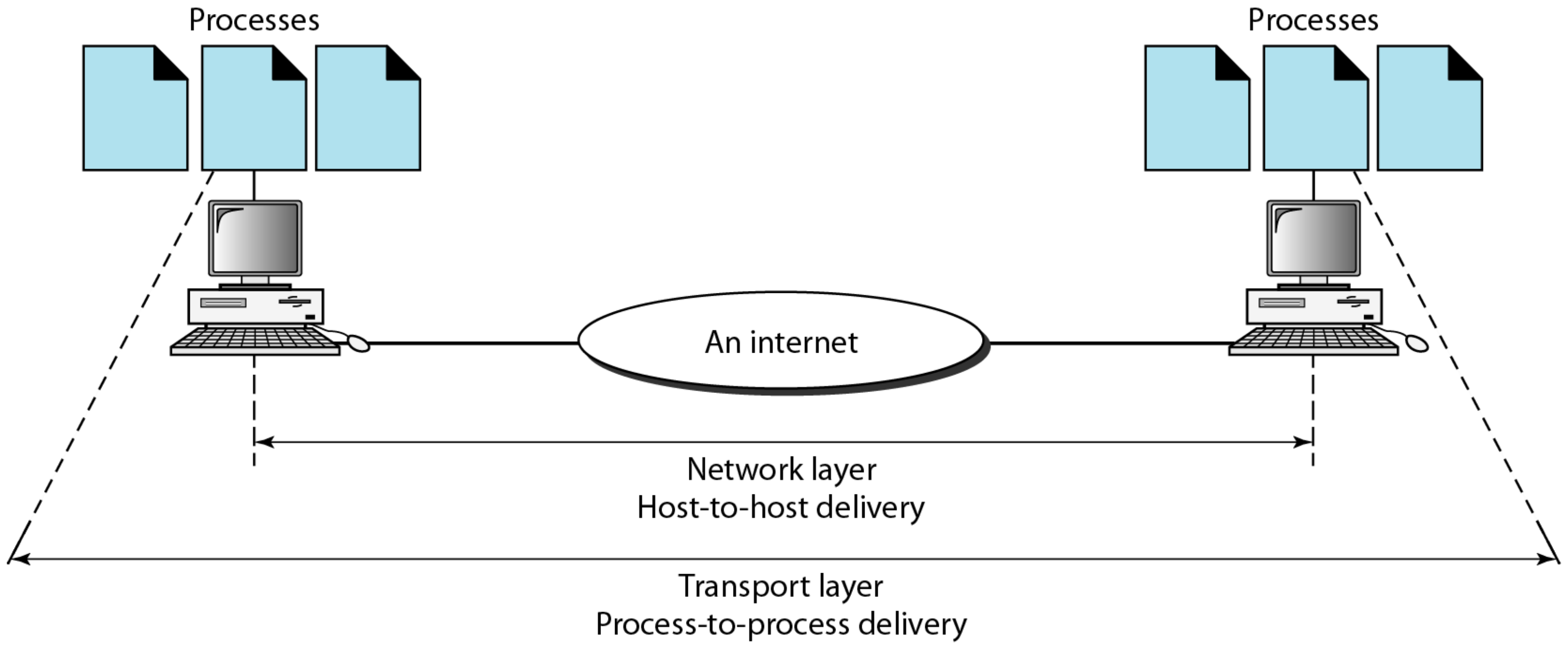


NETWORKING: A QUICK REVIEW

- Transport Layer
- The transport layer is responsible for the delivery of a message from one process to another.

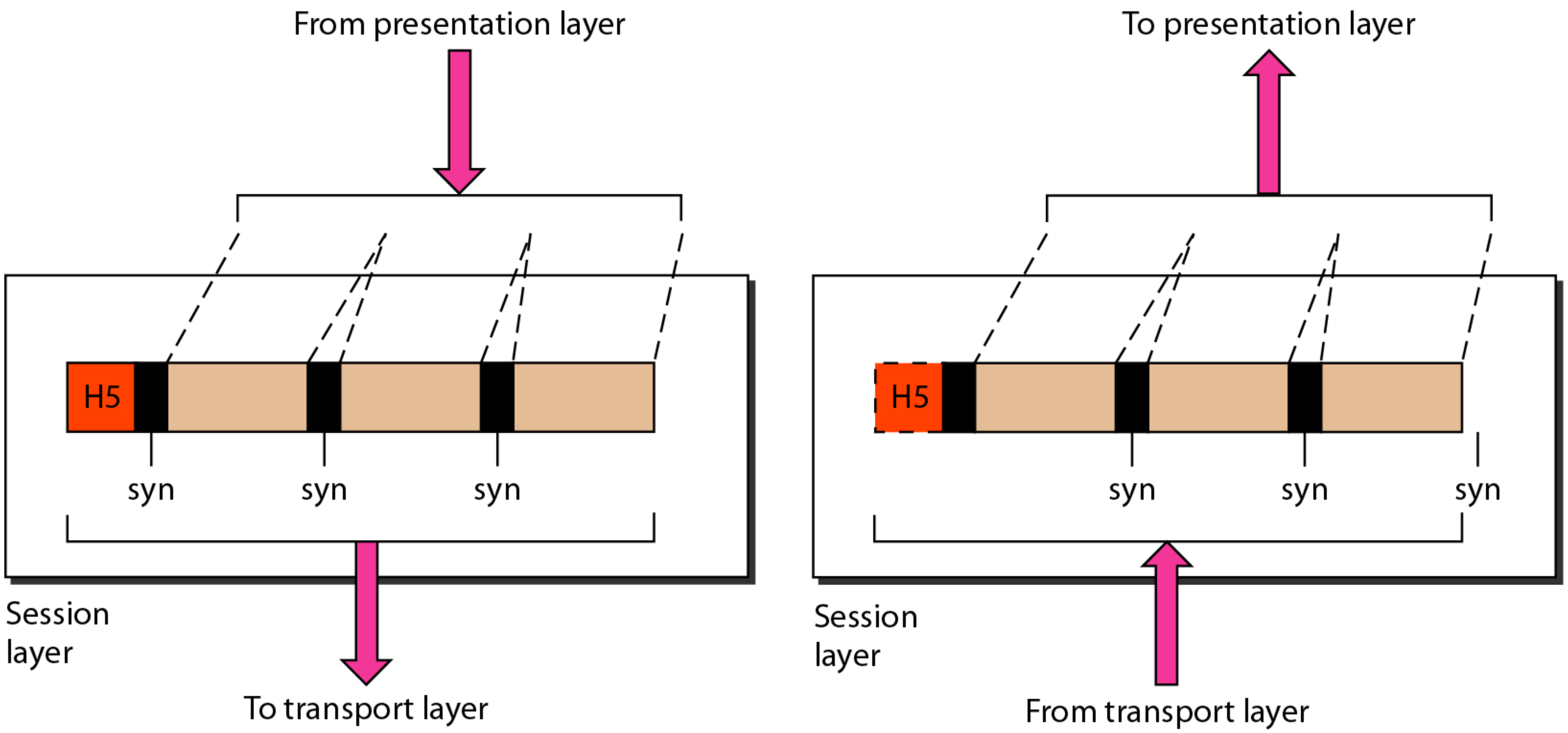


NETWORKING: A QUICK REVIEW



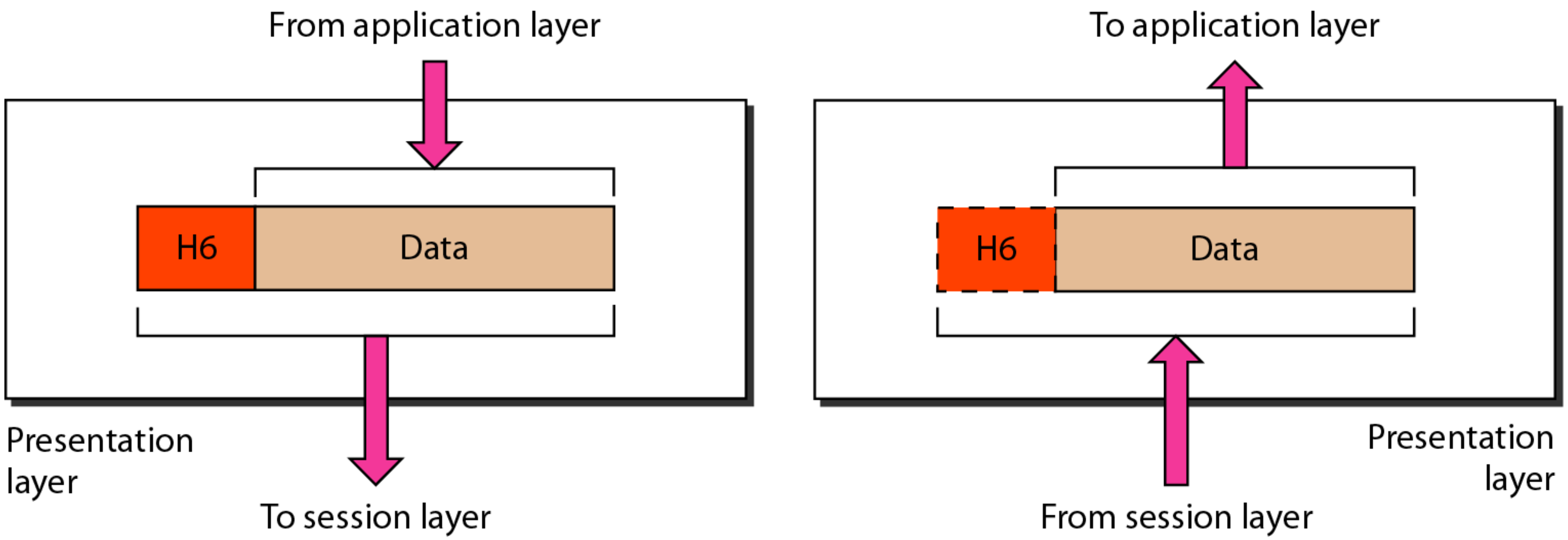
NETWORKING: A QUICK REVIEW

- Transport Layer
- The session layer is responsible for dialog control and synchronization.



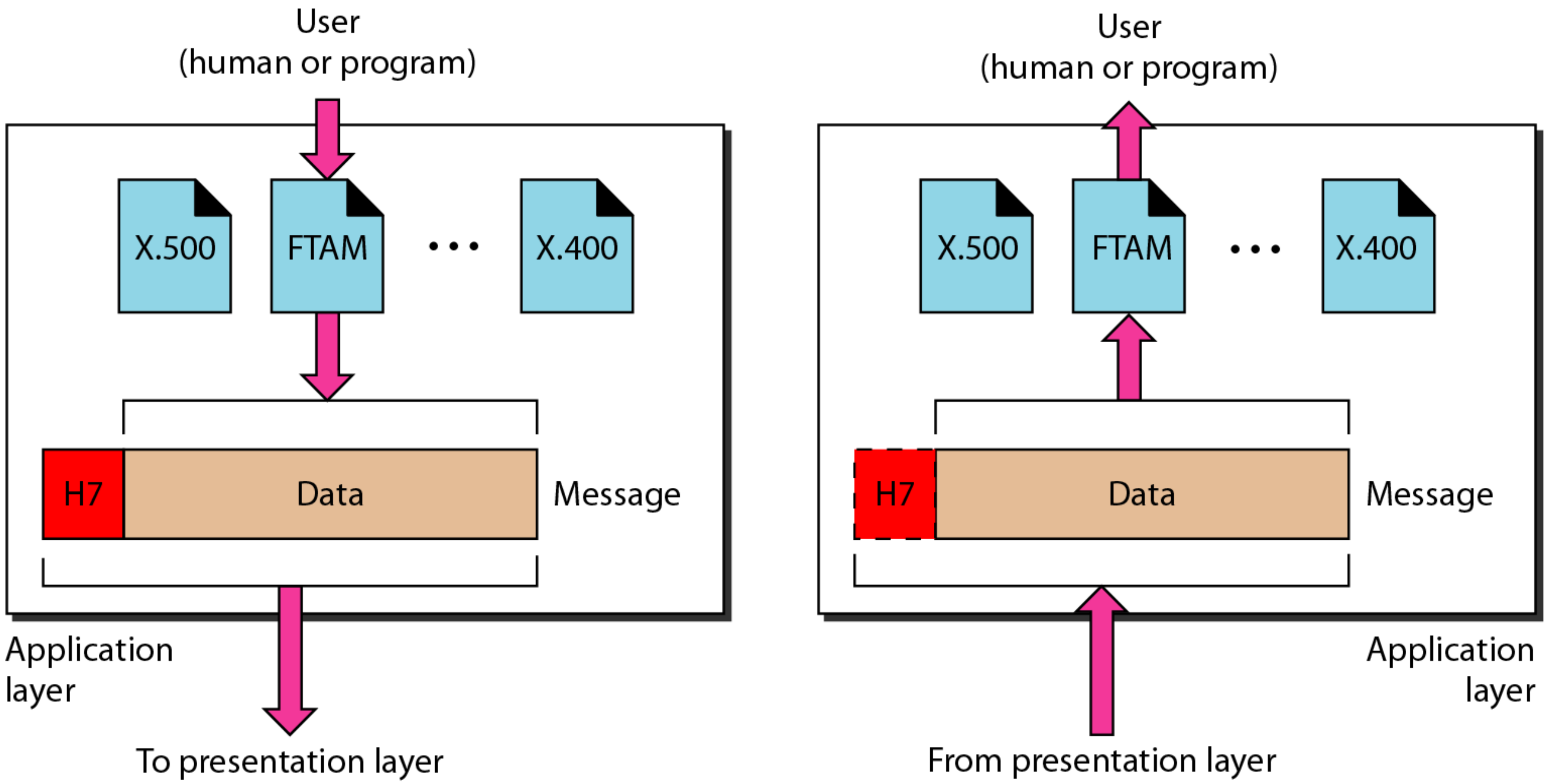
NETWORKING: A QUICK REVIEW

- Presentation Layer
- The presentation layer is responsible for translation, compression, and encryption.



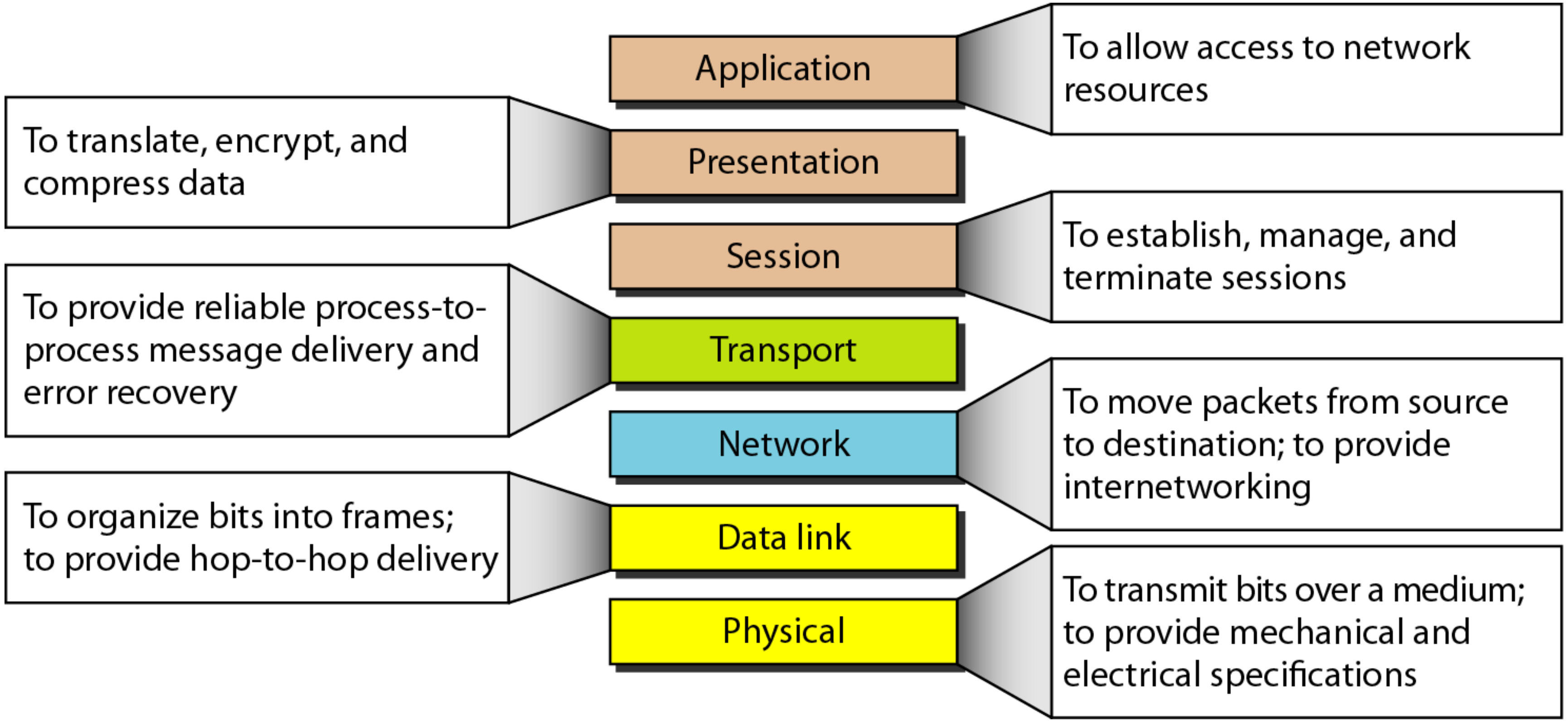
NETWORKING: A QUICK REVIEW

- Application Layer
- The application layer is responsible for providing services to the user.



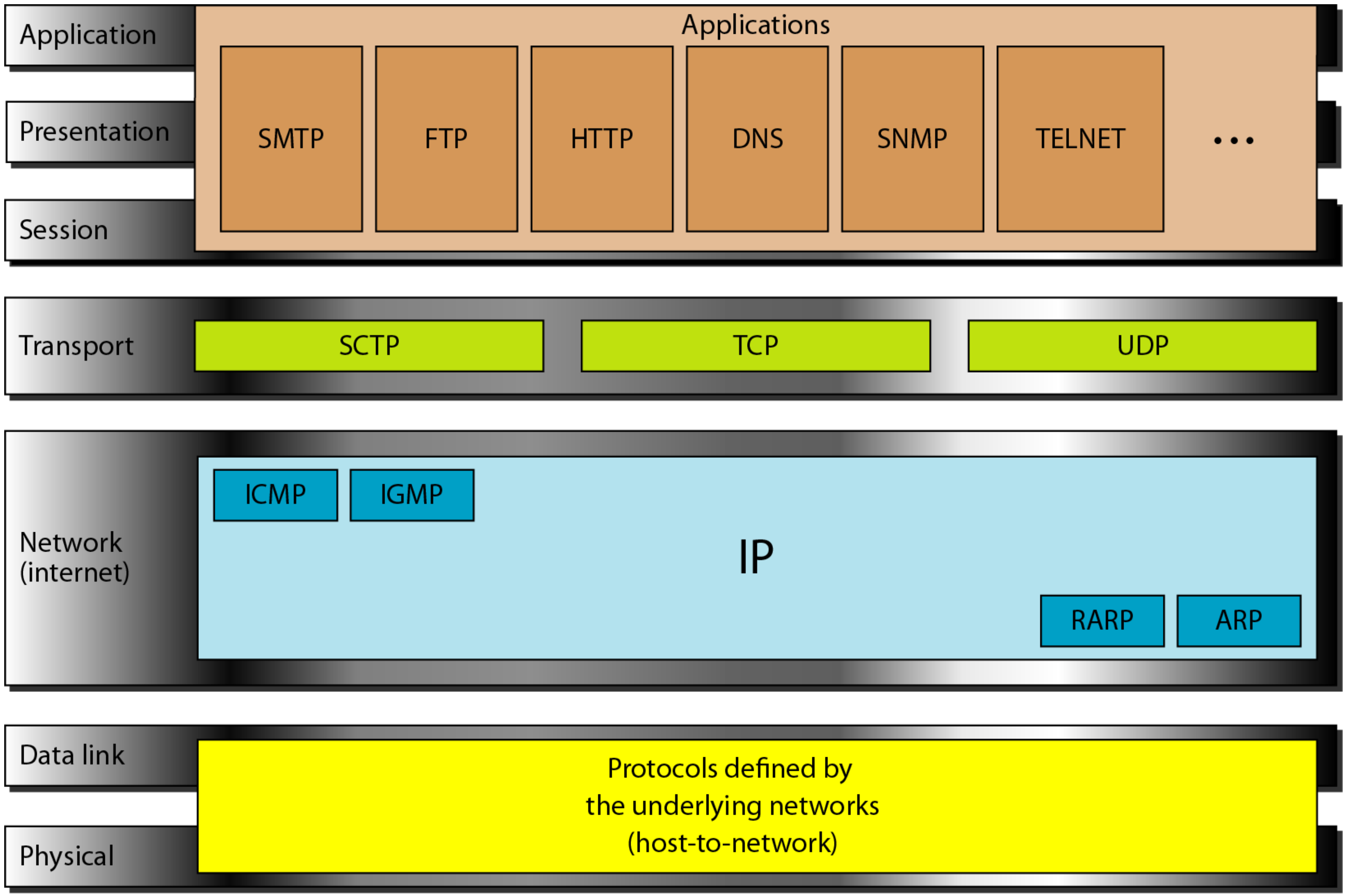
NETWORKING: A QUICK REVIEW

- Summary of OSI Model



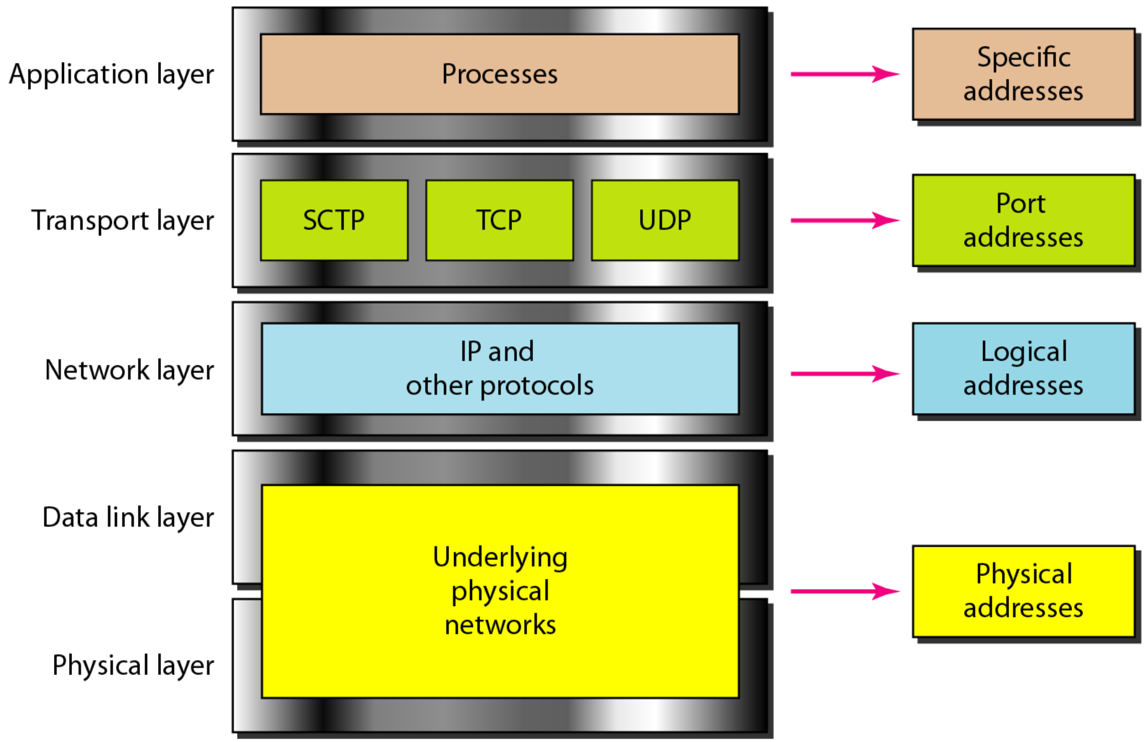
NETWORKING: A QUICK REVIEW

- The original **TCP/IP** protocol suite was defined as having four layers:
  - host-to-network,
  - internet,
  - transport, and
  - application.



NETWORKING: A QUICK REVIEW

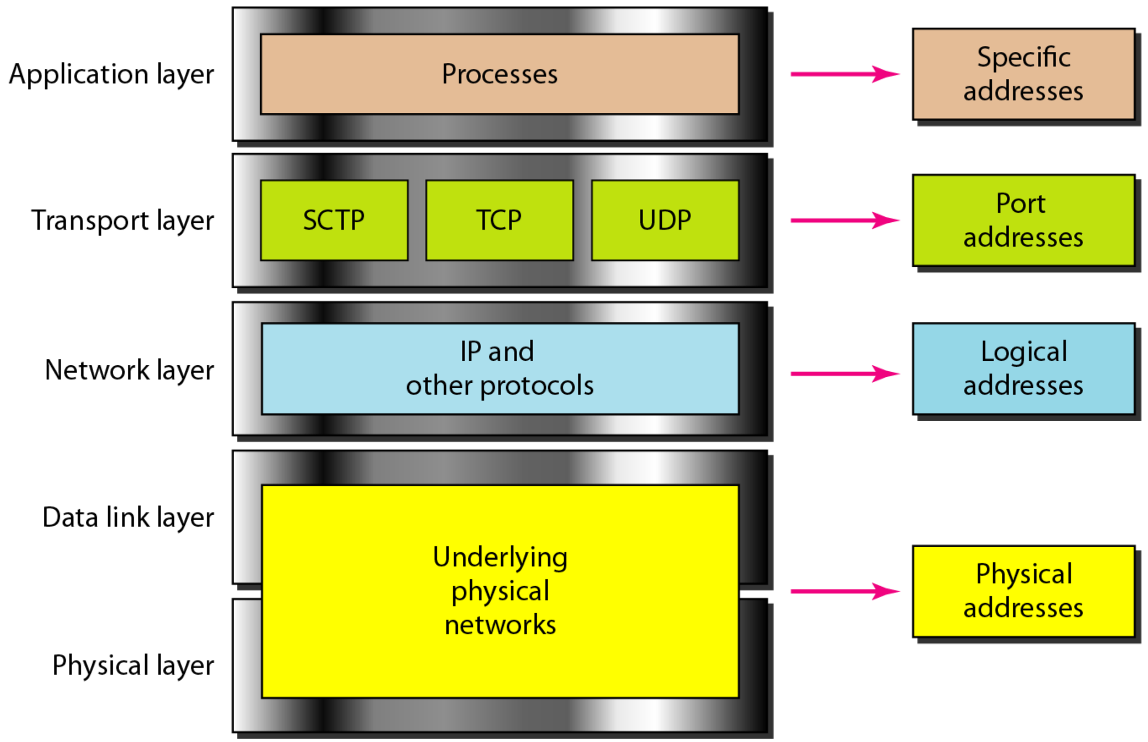
- IP (Internet Protocol) is responsible for transporting packets between computers.
  - Enables applications to communicate with each other by providing logical communication channels so that related messages can be *abstracted as a single stream at an application*.



- Each network endpoint has a unique IP address
  - IPv4: 32-bit address www.psu.edu.sa = 128.6.46.88
  - IPv6: 128-bit address www.google.com = 2607:f8b0:4004:811::2004
- Data is broken into packets
  - Source & destination IP addresses
  - Header checksum
  - Data IP gives us machine-to-machine communication

NETWORKING: A QUICK REVIEW

- TCP (Transmission Control Protocol) provides reliable byte stream (connection-oriented) service.
  - Ensures that packets arrive at the application in order and lost or corrupt packets are retransmitted.
  - Keeps track of the destination so the application can have the illusion of a *connected data stream*.

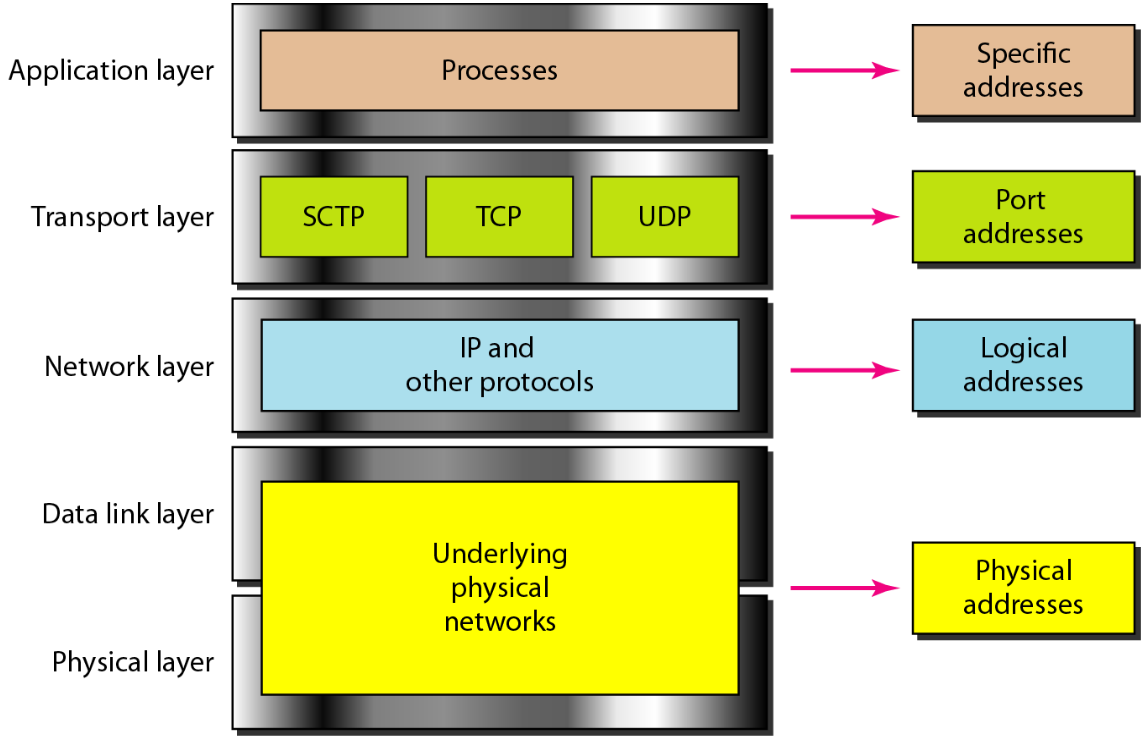


NETWORKING: A QUICK REVIEW

- TCP (Transmission Control Protocol) upside vs downside
- Upsides
  - In-order, reliable byte streams
  - Congestion control (plays nice in sharing the network), flow control (avoids queue overflow)
- Downsides
  - Storing & managing state in the operating system
    - Sequence numbers, Buffering out-of-order data, Acknowledgments
    - Significant kernel memory use when lots of connections
  - Congestion control
    - Slows down transmission but doesn't always accurately reflect network congestion (based on packet loss)
  - Recovery
    - All state is lost if a system goes down – connections will need to be re-established
  - Increased latency
    - Data may not be immediately transmitted or presented to the receiving app

NETWORKING: A QUICK REVIEW

- UDP (User Datagram Protocol) provides datagram (connectionless) service.
  - While UDP drops packets with corrupted data, it does not ensure in-order delivery or reliable delivery.



Port numbers in both TCP and UDP are used to allow the operating system to direct the data to the appropriate application (or, more precisely, to the communication endpoint, or **socket**, that is associated with the communication stream).

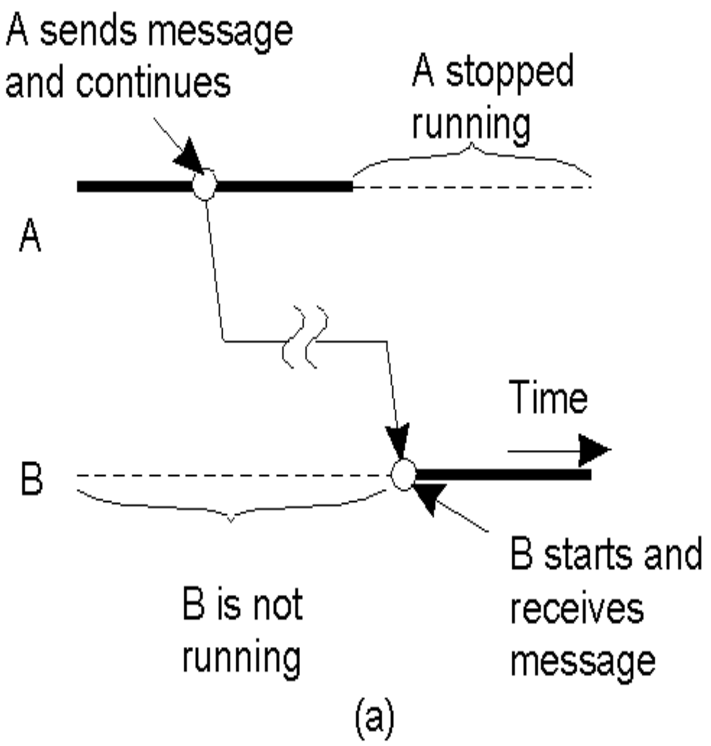
NETWORKING: A QUICK REVIEW

- UDP (User Datagram Protocol) upside vs downside
- Upsides
  - Fewer kernel resources
  - No connection setup overhead
  - useful data can be sent with 1st packet
  - Received data immediately sent & delivered to the application
  - No delay in sending messages
  - No state recovery
  - traffic can be easily redirected to a standby system
- Downsides
  - Delivery & message order not guaranteed
  - Usually perfect on local area networks; less reliable on wide area networks

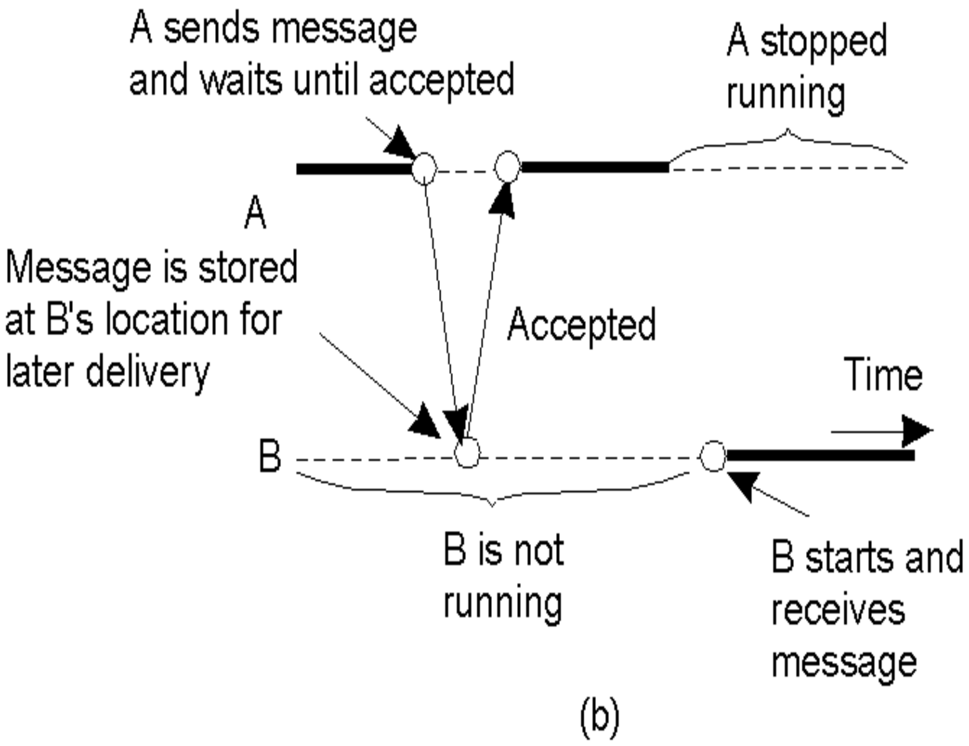
COMMUNICATION TERMINOLOGY

- *Persistent Communications:*
  - Once sent, the “sender” can stop executing. The “receiver” need not be operational at this time – the communications system **buffers** the message as required (until it can be delivered).
- *Transient Communications:*
  - The message is only stored as long as the “sender” and “receiver” are executing. If problems occur, the message is simply **discarded** …
- *Asynchronous Communications:*
  - A sender **continues** with other work immediately upon sending a message to the receiver.
- *Synchronous Communications:*
  - A sender **blocks**, **waiting** for a reply from the receiver before doing any other work. (This tends to be the default model for RPC/RMI technologies).

COMMUNICATION TERMINOLOGY

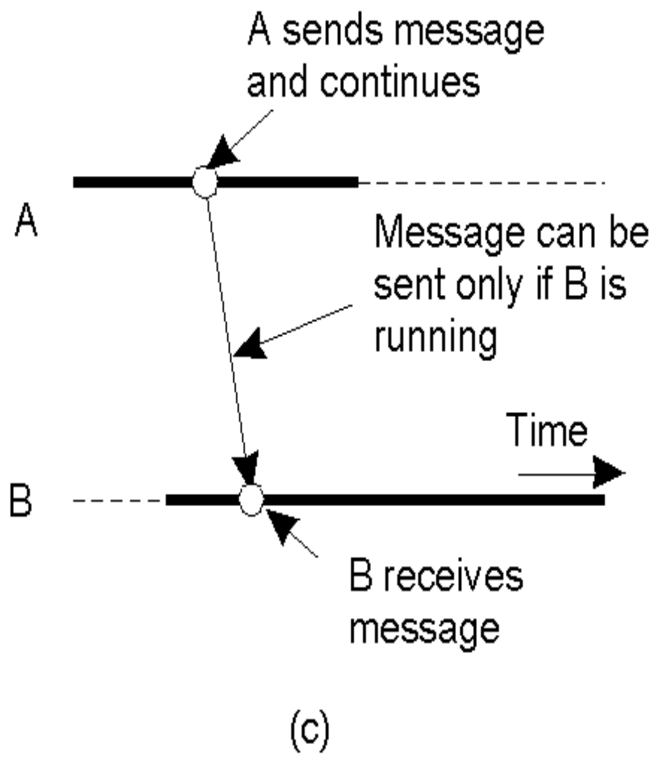


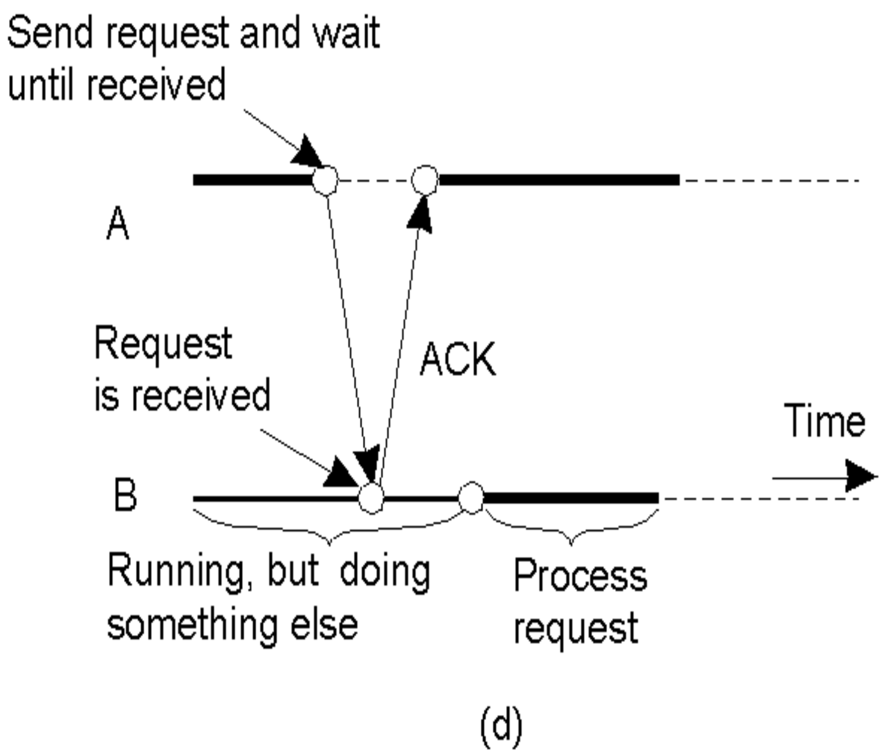
a) Persistent asynchronous communication.



b) Persistent synchronous communication.

COMMUNICATION TERMINOLOGY

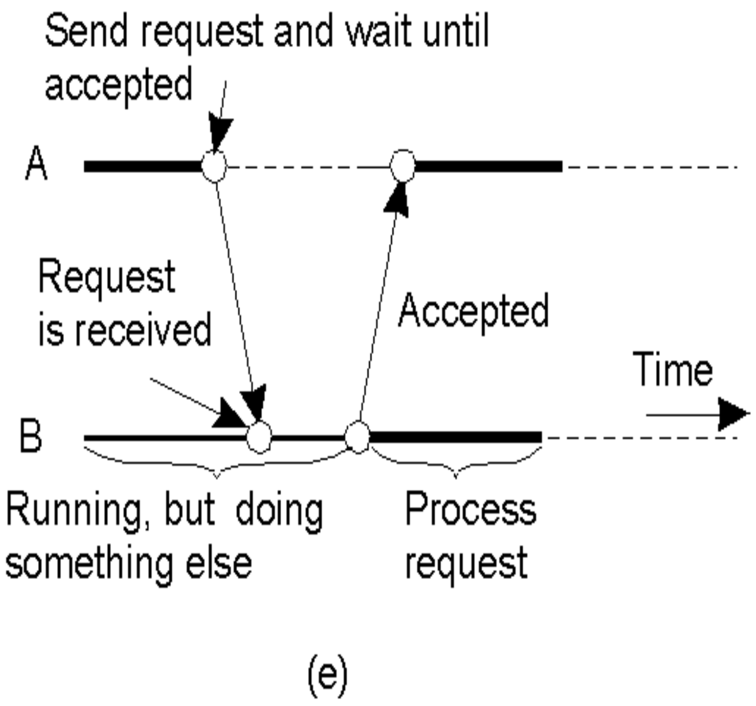


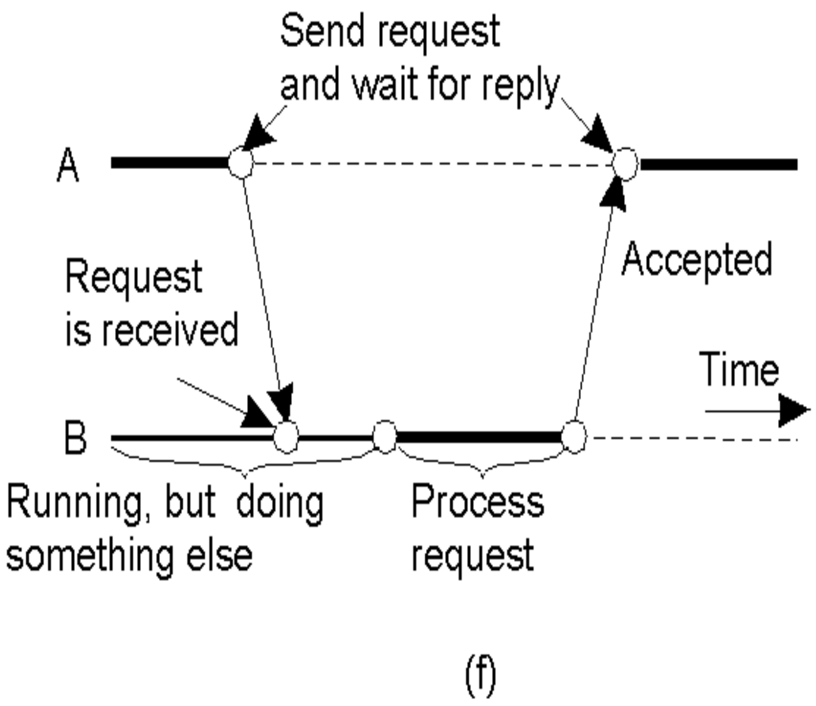


c) Transient asynchronous communication.

d) Receipt-based transient synchronous communication.

COMMUNICATION TERMINOLOGY





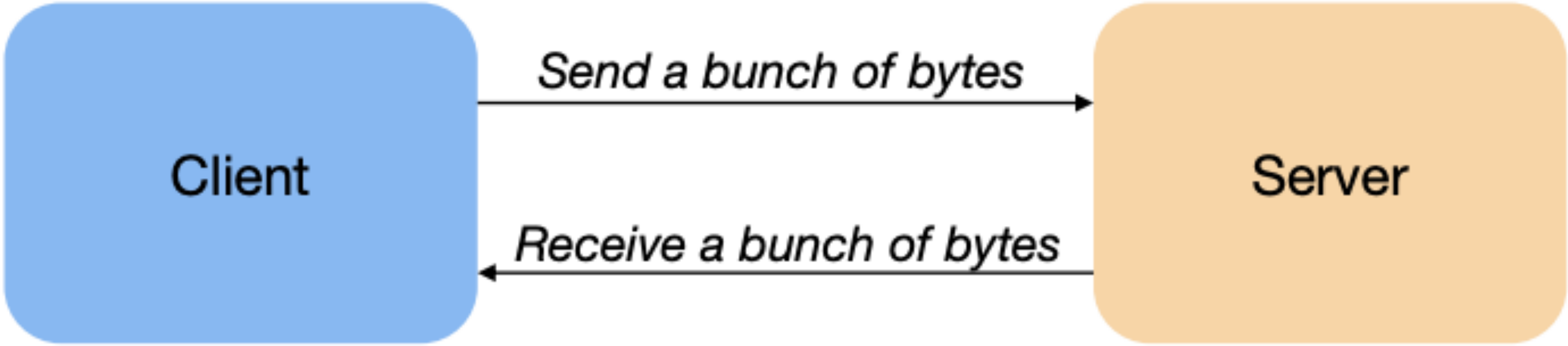
e) Delivery-based transient synchronous communication at message delivery.

f) Response-based transient synchronous communication.

SOCKETS

SOCKETS

- A socket refers to a software endpoint that establishes communication between two processes on a network.
- Sockets enable processes running on different devices to communicate with each other by providing a standard interface for sending and receiving data.
- Sockets are a fundamental concept in network programming and are widely used for building Distributed systems applications.



JAVA SOCKET PROGRAMMING

JAVA SOCKETS PROGRAMMING

- The package java.net provides support for sockets programming (and more).

- Typically you import everything defined in this package with:

```
import java.net.*;
```

SOCKET CLASS

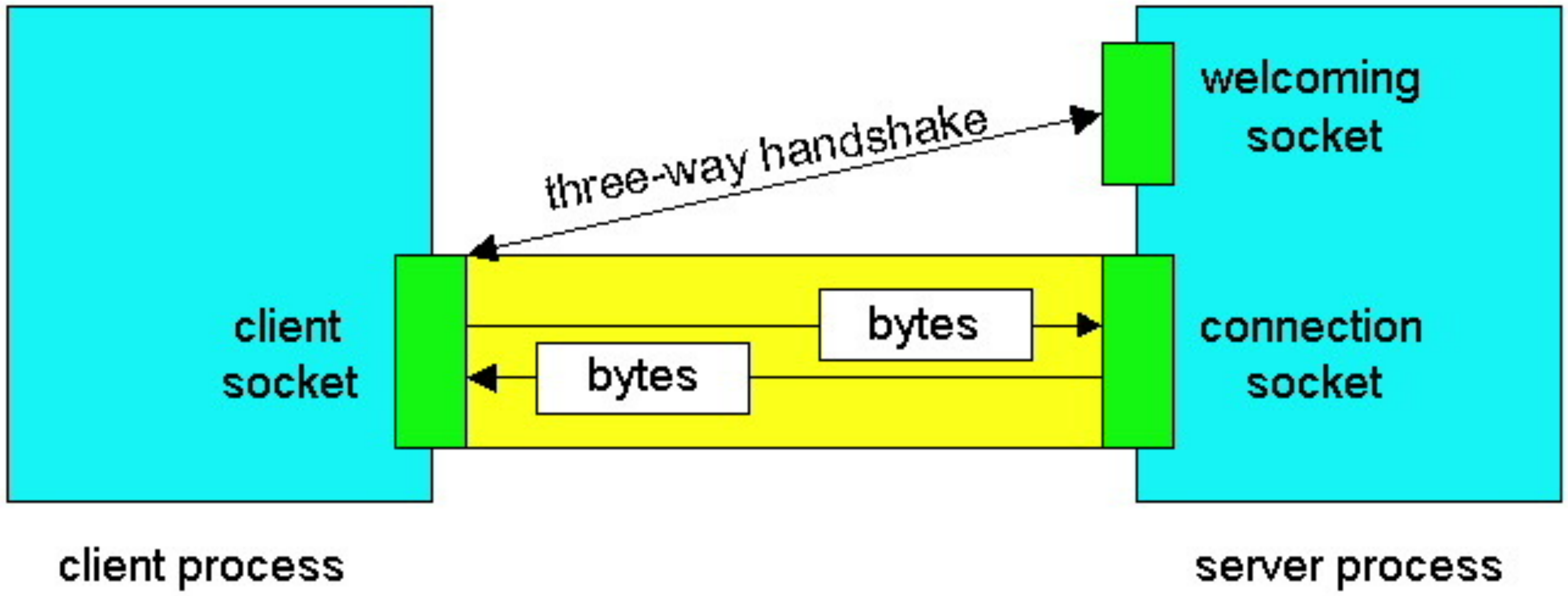
- Corresponds to active TCP sockets only!
  - client sockets
  - socket returned by accept();
- Passive sockets are supported by a different class:
  - ServerSocket
- UDP sockets are supported by
  - DatagramSocket

JAVA TCP SOCKETS

- java.net.Socket
  - Implements client sockets (also called just “sockets”).
  - An endpoint for communication between two machines.
  - Constructor and Methods
    - Socket(String host, int port): Creates a stream socket and connects it to the specified port number on the named host.
    - InputStream getInputStream()
    - OutputStream getOutputStream()
    - close()

- java.net.ServerSocket
  - Implements server sockets.
  - Waits for requests to come in over the network.
  - Performs some operation based on the request.
  - Constructor and Methods
    - ServerSocket(int port)
    - Socket Accept(): Listens for a connection to be made to this socket and accepts it. This method blocks until a connection is made.

SOCKETS



Client socket, welcoming socket (passive) and connection socket (active)

SOCKET CONSTRUCTORS

- Constructor creates a TCP connection to a named TCP server.
  - There are a number of constructors:

```
Socket(InetAddress server, int port);

Socket(InetAddress server, int port,
       InetAddress local, int localport);

Socket(String hostname, int port);
```

INPUTSTREAM

```
// reads some number of bytes and
// puts in buffer array b
int read(byte[] b);

// reads up to len bytes
int read(byte[] b, int off, int len);
```

Both methods can throw **`IOException`**.

Both return –1 on EOF.

OUTPUTSTREAM

```
// writes b.length bytes
void write(byte[] b);

// writes len bytes starting
// at offset off
void write(byte[] b, int off, int len);
```

Both methods can throw **`IOException`**.

SERVERSOCKET CLASS (TCP PASSIVE SOCKET)

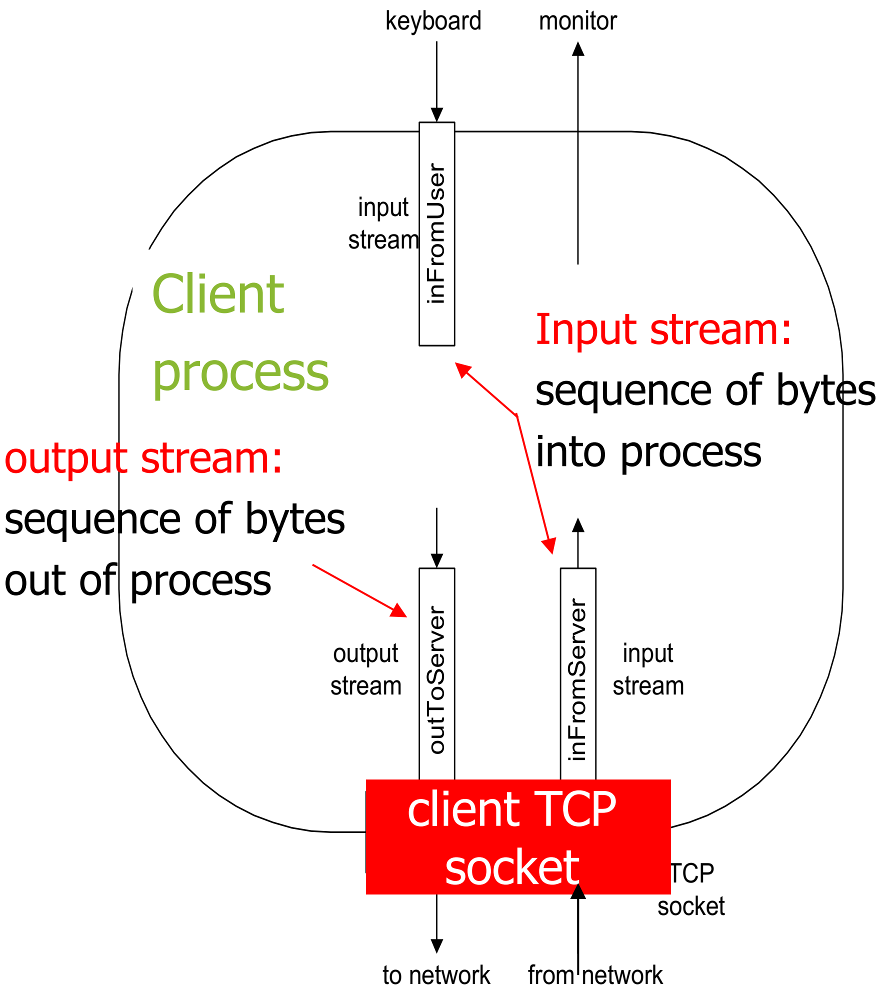
- Constructors:

```
ServerSocket(int port);

ServerSocket(int port, int backlog);

ServerSocket(int port, int backlog, InetAddress bindAddr);
```

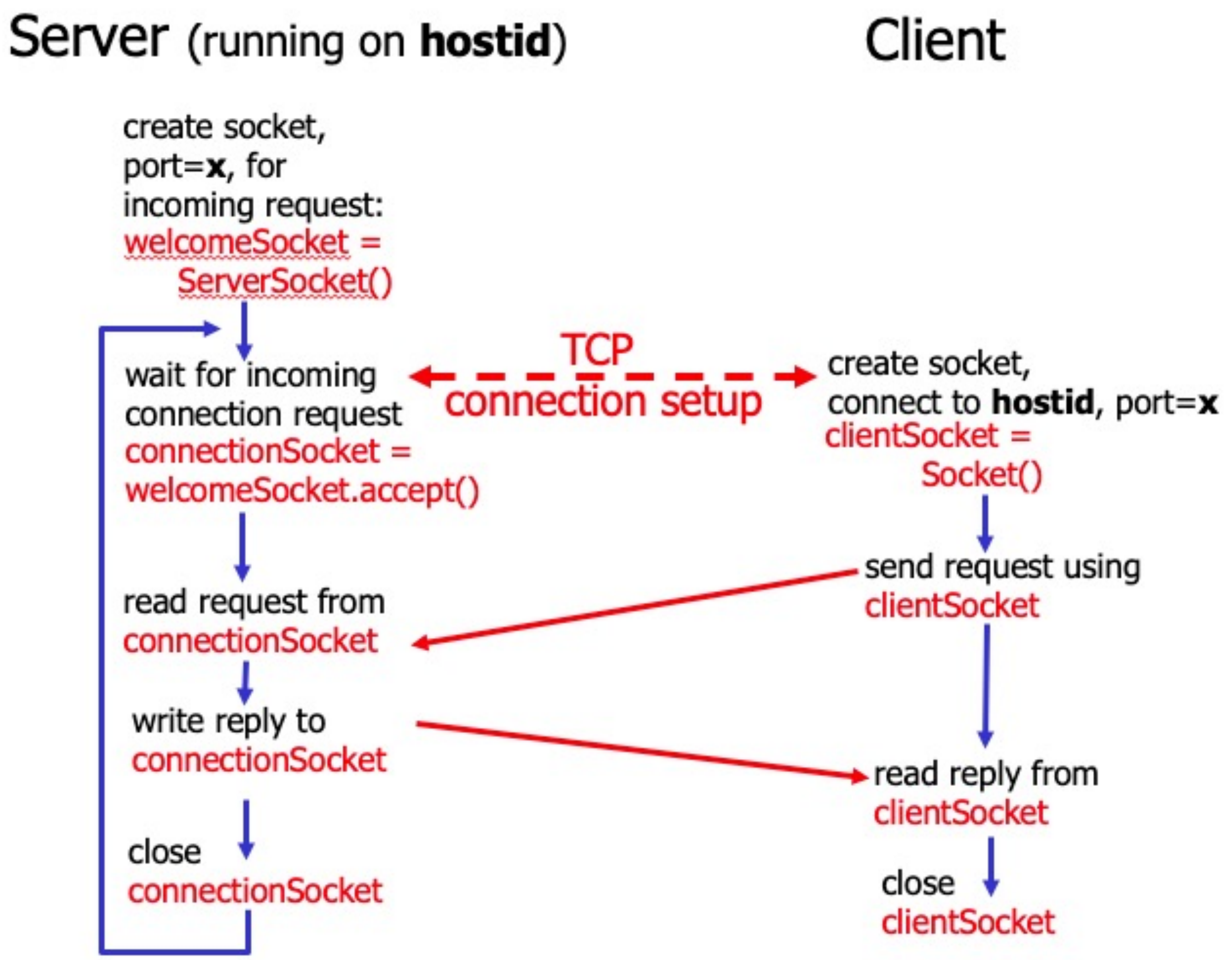
SOCKET PROGRAMMING WITH TCP

Example client-server app:

- client reads line from standard input (**inFromUser** stream) , sends to server via socket (**outToServer** stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (**inFromServer** stream)



CLIENT/SERVER SOCKET INTERACTION: TCP



TCPCLIENT.JAVA

```
package sockets;
import java.io.*;
import java.net.*;
import java.util.Scanner;
class TCPClient {
    public static void main(String [] args) throws Exception{
        String Str;
        String modifiedSentence;
        Scanner In = new Scanner(System.in);
        Socket cs = new Socket("hostname", 6789);
        DataOutputStream outToServer = new DataOutputStream(cs.getOutputStream());
        InputStreamReader StreamIn = new InputStreamReader(cs.getInputStream());
        BufferedReader inFromServer = new BufferedReader(StreamIn);
        Str = In.nextLine();
        //send Str to server
        outToServer.writeBytes(Str + '\n');
        //Listen to server response
        modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: " + modifiedSentence);
        cs.close();
    }
}
```

TCPSERVER.JAVA

```
package sockets;
import java.io.*;
import java.net.*;
public class TCPServer {
    public static void main(String [] args) throws Exception {
        String Str;
        String UpperCase;
        ServerSocket s = new ServerSocket(6789);
        while(true) {
            Socket cs = s.accept();
            InputStreamReader StreamIn = new InputStreamReader(cs.getInputStream());
            BufferedReader in = new BufferedReader(StreamIn);
            DataOutputStream outToClient = new DataOutputStream(cs.getOutputStream());
            //read input stream into Str
            Str = in.readLine();
            UpperCase = Str.toUpperCase() + '\n';
            //write string to stream
            outToClient.writeBytes(UpperCase);
        }
    }
}
```

UDP SOCKETS

- DatagramSocket class
- DatagramPacket class needed to specify the payload
  - incoming or outgoing

SOCKET PROGRAMMING WITH UDP

- UDP
  - Connectionless and unreliable service.
  - There isn’t an initial handshaking phase.
  - Doesn’t have a pipe.
  - transmitted data may be received out of order, or lost

- Socket Programming with UDP
  - No need for a welcoming socket.
  - No streams are attached to the sockets.
  - the sending hosts creates “packets” by attaching the IP destination address and port number to each batch of bytes.
  - The receiving process must unravel to received packet to obtain the packet’s information bytes.

JAVA UDP SOCKETS

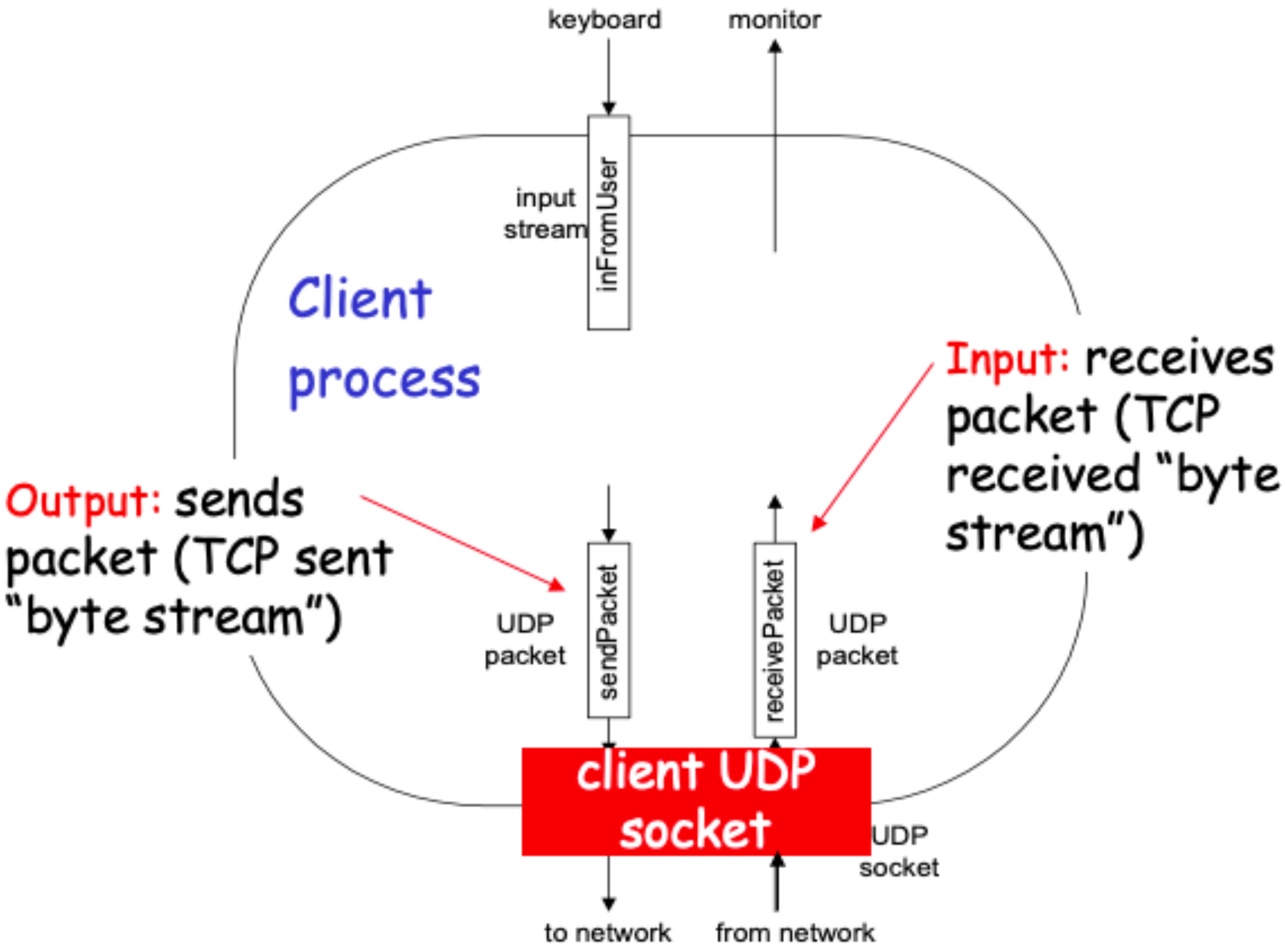
- In Package java.net
  - java.net.DatagramSocket
    - A socket for sending and receiving datagram packets.
    - Constructor and Methods
      - DatagramSocket(int port): Constructs a datagram socket and binds it to the specified port on the local host machine.
      - void receive(DatagramPacket p)
      - void send(DatagramPacket p)
      - void close()

DATAGRAMSOCKET CONSTRUCTORS

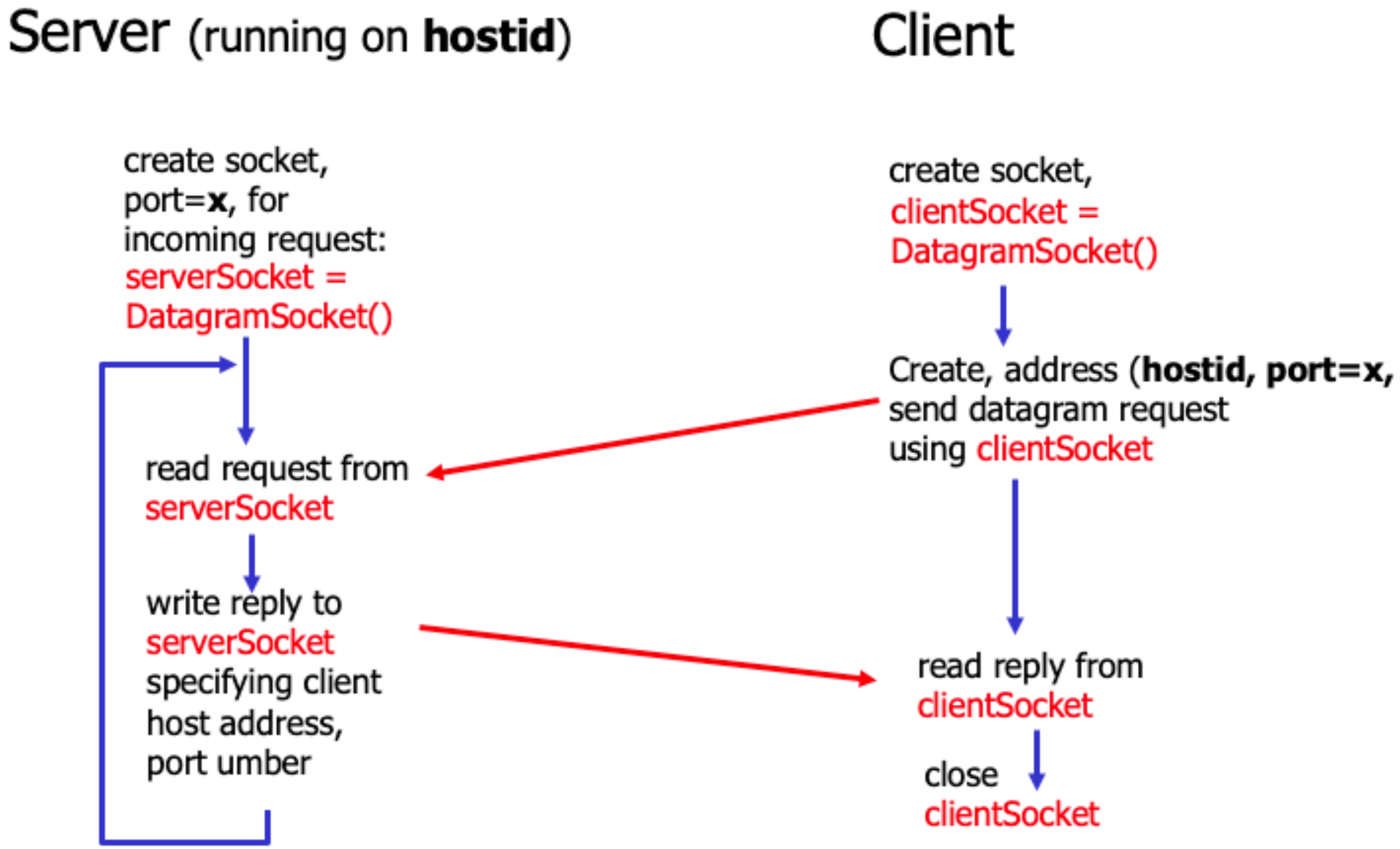
```
DatagramSocket();

DatagramSocket(int port);

DatagramSocket(int port, InetAddress a);
```

All can throw SocketException or SecurityException

EXAMPLE: JAVA CLIENT (UDP)



CLIENT/SERVER SOCKET INTERACTION: UDP



UDPCLIENT.JAVA

```
import java.io.*;
import java.net.*;

  class UDPClient {
      public static void main(String args[]) throws Exception
      {

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        DatagramSocket clientSocket = new DatagramSocket();

        InetAddress IPAddress =
  InetAddress.getByName("hostname");

        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();

        sendData = sentence.getBytes();
```

UDPCLIENT.JAVA

```
    DatagramPacket sendPacket =
         new DatagramPacket(sendData, sendData.length,
IPAddress, 9876);

  clientSocket.send(sendPacket);

  DatagramPacket receivePacket =
         new DatagramPacket(receiveData, receiveData.length);

  clientSocket.receive(receivePacket);

  String modifiedSentence =
         new String(receivePacket.getData());

  System.out.println("FROM SERVER:" + modifiedSentence);

  clientSocket.close();
   }
}
```

UDPSERVER.JAVA

```
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception
      {

        DatagramSocket serverSocket = new
   DatagramSocket(9876);

        byte[] receiveData = new byte[1024];
        byte[] sendData  = new byte[1024];

        while(true)
          {

            DatagramPacket receivePacket =
               new DatagramPacket(receiveData, receiveData.length);

            serverSocket.receive(receivePacket);

            String sentence = new String(receivePacket.getData());
```

SUMMARY

- In distributed systems, multiple components interact across networks, requiring effective communication to **coordinate** their actions and ensure coherent behavior.
- Communication facilitates **fault detection and recovery mechanisms**. Nodes need to exchange information to detect failures, redistribute tasks, and maintain system resilience.
- Communication enables the **synchronization** of data and state across distributed nodes, ensuring that all components have consistent views of the system, critical for maintaining integrity and correctness.
- Network protocols are essential for communication
- Sockets enable TCP/UDP communication
- RPC/MPI facilitate reliable dist sys communication (next lecture)