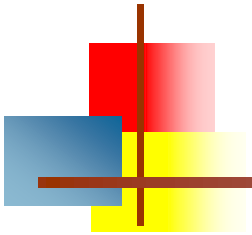
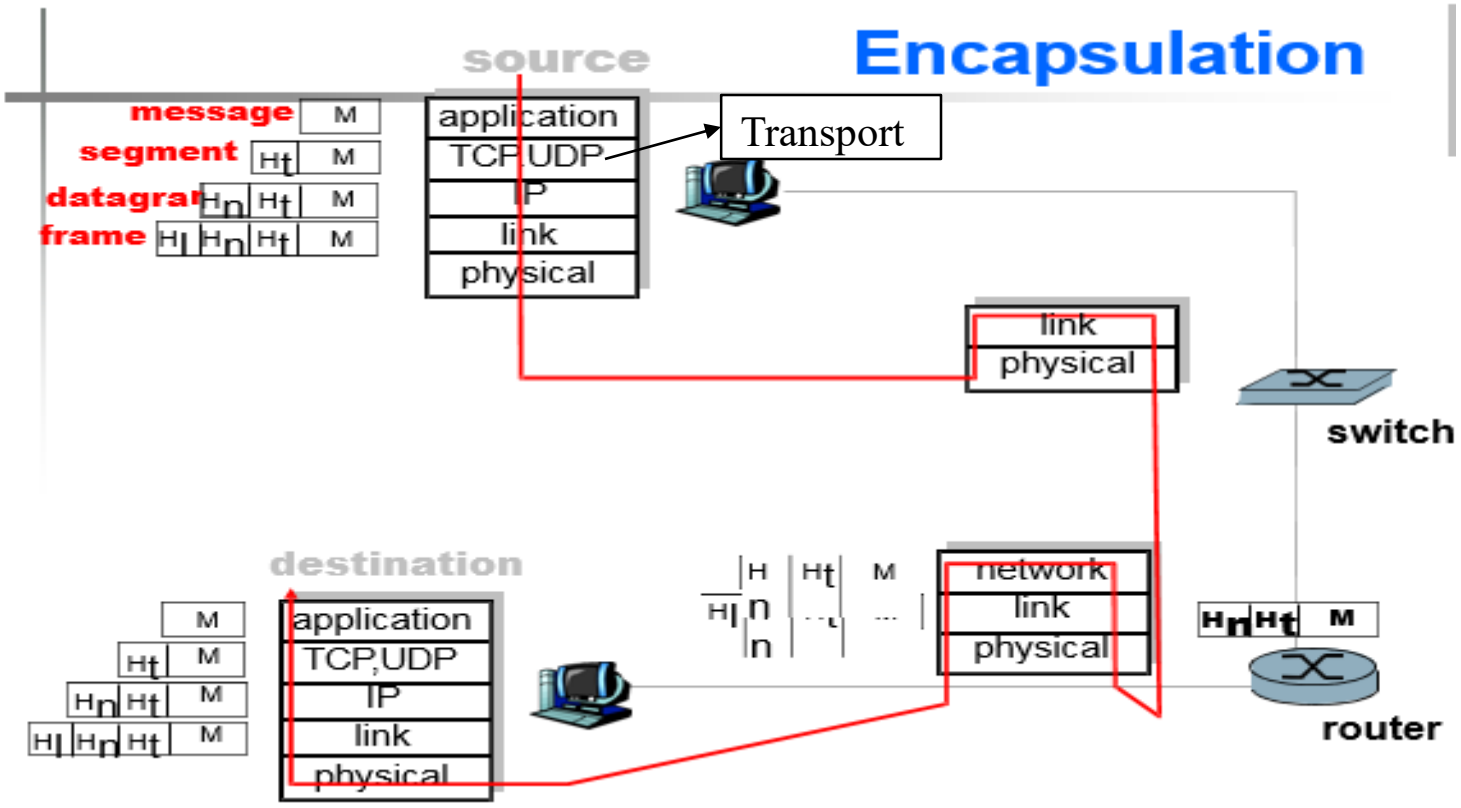


# Interprocess Communication

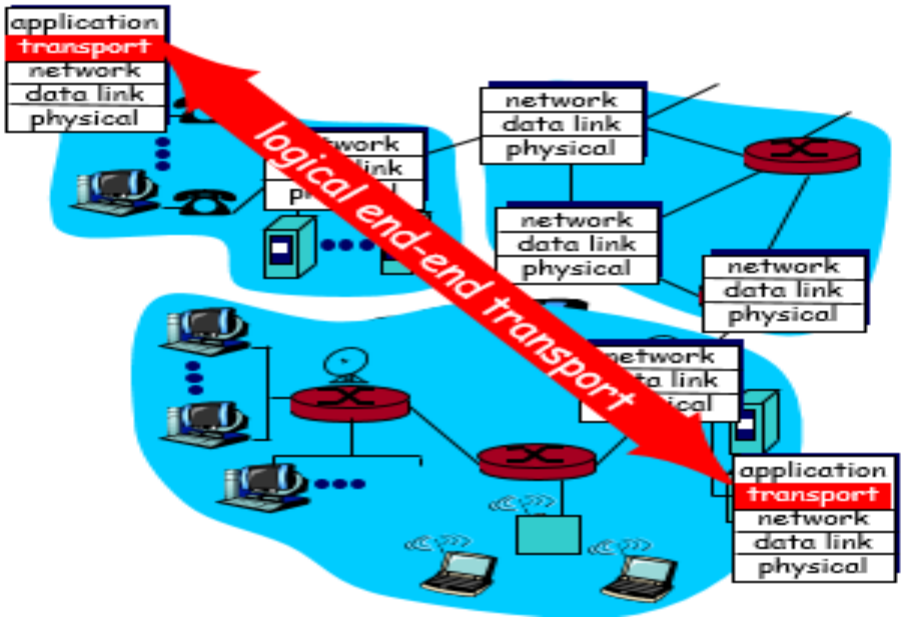
# TCP/IP Model

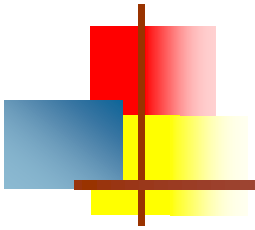
- TCP/IP Model



# Transport Protocols

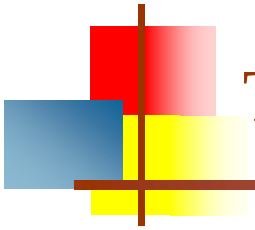
- Provide *logical communication* between application processes running on different hosts
- Run on end hosts
  - Sender: breaks application messages into segments, and passes to network layer
  - Receiver: reassembles segments into messages, passes to application layer
- Multiple transport protocol available to applications
  - Internet: TCP and UDP





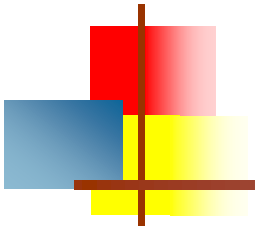
# Internet Transport Protocols

- User Datagram Protocol (UDP)
  - Provides: Unreliable, unordered delivery of segments.
- Transmission Control Protocol (TCP)
  - Provides: Reliable, in-order delivery of segments.
  - TCP includes:
    - Connection set-up (3-way handshake)
    - Flow Control
    - Congestion Control (packets lost)
    - Discarding of corrupted packets
    - Retransmission of lost packets
- ***NOTE:*** Neither protocol provides:
    - Delay guarantees
    - Bandwidth guarantees



# The characteristics of interprocess communication

- Communication: One process sends a message to a destination and another process at the destination receives the message.
- Communication between the sending and receiving processes may be either synchronous (blocking) or asynchronous (non-blocking).
- Queue: A queue is associated with each message destination.
  - Sending processes cause messages to be added to remote queues and receiving processes remove messages from local queues.



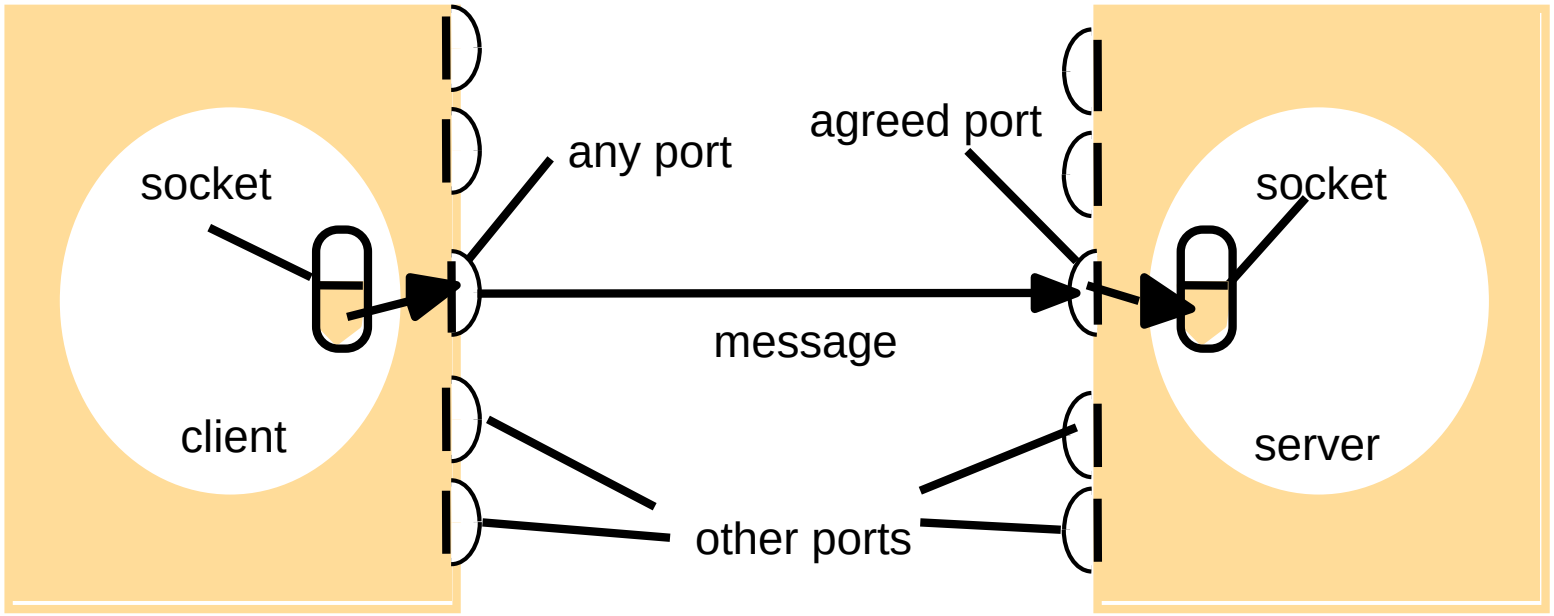
# The characteristics of interprocess communication

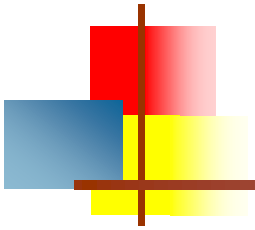
- Messages are sent to (Internet address, local port) pairs.
  - A port has exactly one receiver but can have many senders.
  - Processes may use multiple ports to receive messages.
  - Any process that knows the number of a port can send a message to it.
  - Each computer has a large number ($2^{16}$) of possible port numbers for use by local processes for receiving messages.
  - Process cannot share ports with other processes on the same computer.
- Validity: A point-to-point message service is reliable if messages are guaranteed to be delivered despite packet drop
- Integrity: Messages must arrive uncorrupted and without duplication
- Ordering: messages be delivered in sender order

# Sockets

- Interprocess communication consists of transmitting a message between a socket in one process and a socket in another process.

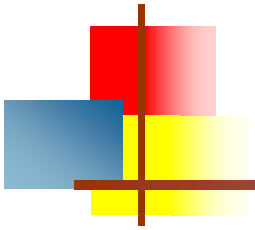
- Processes may use the same socket for sending and receiving messages.





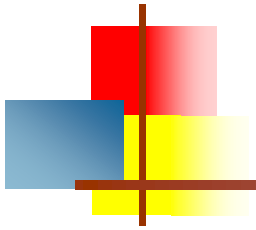
# UDP datagram communication

- Datagram sent by UDP is transmitted from a sending process to a receiving process without acknowledgement or retries.
- If a failure occurs, the message may not arrive.
- To send or receive messages a process must first create a socket bound to an Internet address of the local host and a local port.
- A client binds its socket to any free local port.
- Server will bind its socket to a known server port.
- The receive method returns the Internet address and port of the sender, in addition to the message, allowing the recipient to send a reply.



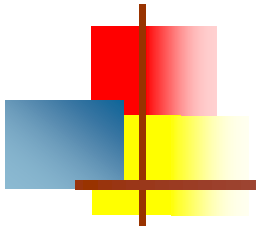
# UDP datagram communication

- Message size: The receiving process needs to specify an array of bytes of a particular size in which to receive a message.
- Blocking: Sockets normally provide non-blocking sends and blocking receives for datagram communication
- The send operation returns when it has handed the message to the underlying UDP and IP protocols.
- Queue: On arrival, the message is placed in a queue for the socket that is bound to the destination port.
- The message can be collected from the queue by an outstanding or future invocation of receive on that socket.
- Messages are discarded at the destination if no process already has a socket bound to the destination port.



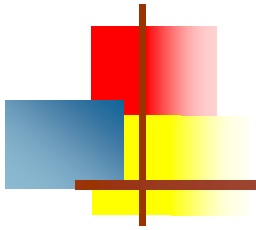
# UDP datagram communication

- Timeouts: Use to avoid  waiting forever.
- UDP uses a checksum to ensure that the message is not corrupted.
- UDP datagrams suffer from the following failures:
  - Omission failures
  - Ordering
- A reliable delivery service may be constructed using acknowledgements which is not available in UDP.



# Java API for Internet addresses

- Java provides a class, InetAddress, that represents Internet addresses. Users of this class refer to computers by Domain Name System (DNS) hostnames.
- For example, to get an object representing the Internet address of the host whose DNS name is bruno.dcs.qmul.ac.uk, use:
  - InetAddress aComputer = InetAddress.getByName("bruno.dcs.qmul.ac.uk");
- This method can throw an UnknownHostException.
- The interface does not depend on the number of bytes needed to represent Internet addresses



# Java Classes for UDP datagrams

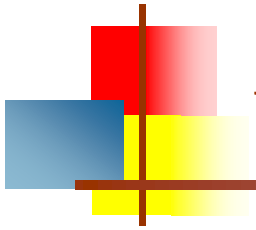
- DatagramPacket: This class provides an instance that contains the message, length, IP and port
  - The message can be retrieved with the method getData.
- DatagramSocket: This class supports sockets for sending and receiving UDP datagrams. Methods include:
  - The constructor accepts a port. An exception(SocketException) is thrown if the chosen port is already in use or is reserved
  - send and receive
  - setSoTimeout

# UDP client sends a message to the server and gets a reply

```
import java.net.*;
import java.io.*;
public class UDPClient{
    public static void main(String args[]){
                // args give message contents and server hostname
                DatagramSocket aSocket = null;
                  try {
                                aSocket = new DatagramSocket();
                                byte [] m = args[0].getBytes();
                                InetAddress aHost = InetAddress.getByName(args[1]);
                                int serverPort = 6789;
                                DatagramPacket request = new DatagramPacket(m,  m.length(), aHost, serverPort);
                                aSocket.send(request);
                                byte[] buffer = new byte[1000];
                                DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                                aSocket.receive(reply);
                                System.out.println("Reply: " + new String(reply.getData()));
                  }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
                  }catch (IOException e){System.out.println("IO: " + e.getMessage());}
                }finally {if(aSocket != null) aSocket.close();}
    }
}
```

# UDP server repeatedly receives a request and sends it back to the client

```
import java.net.*;
import java.io.*;
public class UDPServer{
        public static void main(String args[]){
        DatagramSocket aSocket = null;
          try{
                aSocket = new DatagramSocket(6789);
                byte[] buffer = new byte[1000];
                while(true){
                  DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                  aSocket.receive(request);
                  DatagramPacket reply = new DatagramPacket(request.getData(),
                          request.getLength(), request.getAddress(), request.getPort());
                  aSocket.send(reply);
                }
           }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
          }catch (IOException e) {System.out.println("IO: " + e.getMessage());}
        }finally {if(aSocket != null) aSocket.close();}
   }
}
```



# Why Would Anyone Use UDP?

- Finer control over what data is sent and when
  - As soon as an application process writes into the socket
  - … UDP will package the data and send the packet
- No delay for connection establishment
  - UDP just blasts away without any formal preliminaries
  - … which avoids introducing any unnecessary delays
- No connection state
  - No allocation of buffers, parameters, sequence #s, etc.
  - … making it easier to handle many active clients at once
- Small packet header overhead
  - UDP header is only eight-bytes long

# Popular Applications That Use UDP

- Multimedia streaming
  - Retransmitting lost/corrupted packets is not worthwhile
  - By the time the packet is retransmitted, it's too late
  - E.g., telephone calls, video conferencing, gaming
- Simple query protocols like Domain Name System
  - Overhead of connection establishment is overkill
  - Easier to have application retransmit if needed