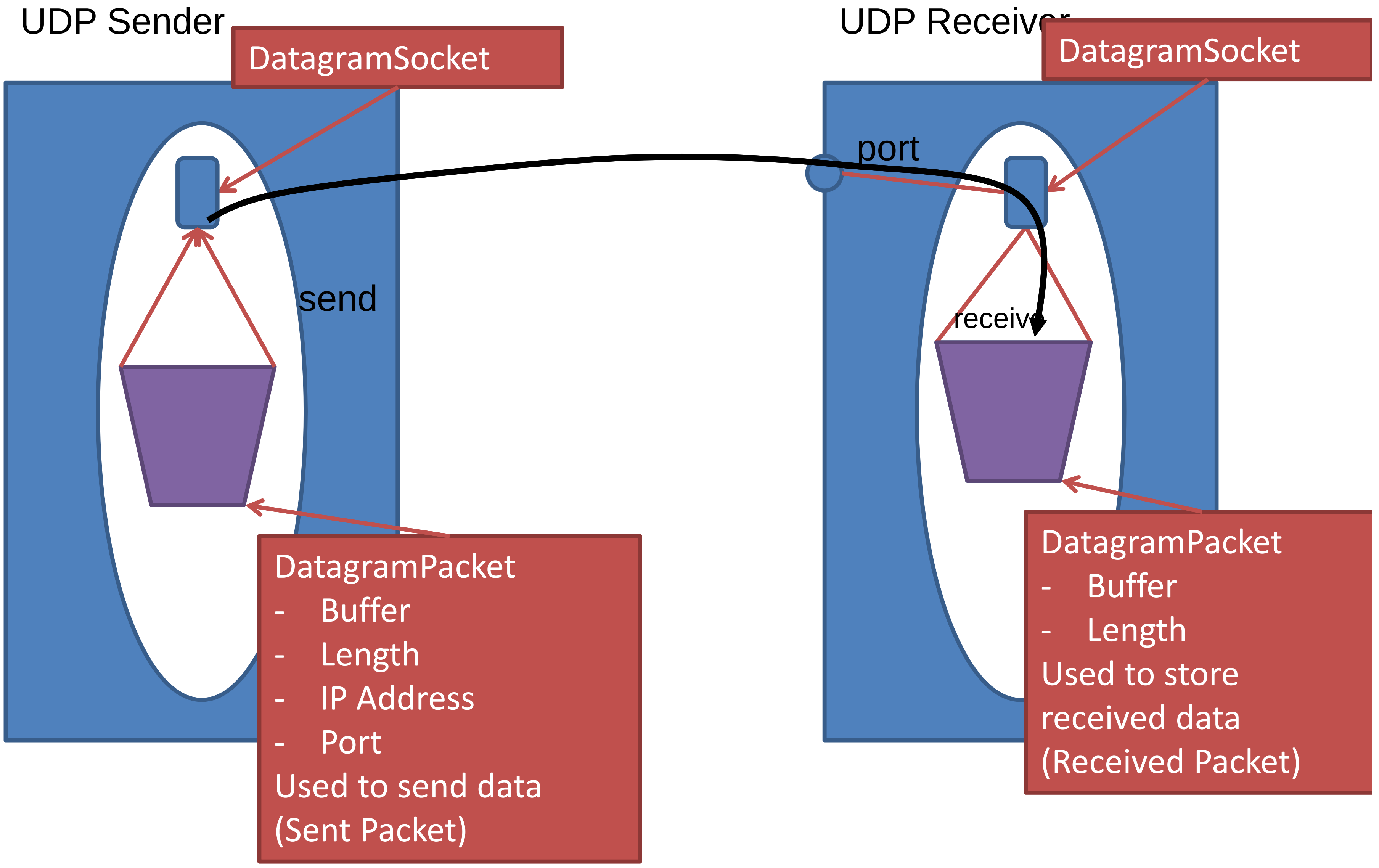
UDP

User Datagram Protocol

How does it work?



UDP communication

- UDP and the Datagram method
  - In a nutshell
    - Datagram sent from one process to a receiving process
    - First the sender (the *client*) and receiver (the *server*) must bind to a socket
      - » Client could bind to any port
      - » Server binds to its specified broadcast port for receiving messages
    - The client then sends its message to the server address including in the message the senders address (required for the response)
    - The server receives and processes
    - The server then sends a response to the client address and port

UDP communication

- More details
  - The *send* method is non-blocking (asynchronous)
    - thus they are free once send has taken place
  - The *receive* method uses blocking (synchronous), although other threads can be used to perform other work.
  - Receive blocking can use time-outs to limit the length of the block.
    - although defining ‘good’ values for timeouts is hard
  - Received messages are stored in the bound socket’s queue.
  - Receive inspects the bound socket for messages
  - Received messages could come from anywhere

UDP communication

- Uses of UDP and the Datagram method
  - Useful where omission failures are tolerable
    - i.e. naming services
  - Useful in reducing communication overheads that exist in guaranteed delivery methods

UDP communication

- UDP and the Datagram method
  - Method example:
    - aSocket.send(*request*)
    - aSocket.recieve(*reply*)
  - where both *request* and *reply* are DatagramPackets
  - Other methods:
    - setSoTimeout
    - connect

API & IP: UDP communication

- In Java
  - A **DatagramPacket** class contains:

The message	Length of message	Internet address	Port

  - i.e.

3432 543 4531	13	145.25.123.871	589

In Java a DatagramPacket is constructed:

```
myPacket = new DatagramPacket(m,args[0].length(), aHost, serverPort);
```

Note: the DatagramPacket contains the host address (aHost) and the host port (serverPort)

API & IP: UDP communication

- In Java

In Java a DatagramPacket is sent and received thus:

```
aSocket.send(myPacket);
aSocket.recieve(myPacket);
```

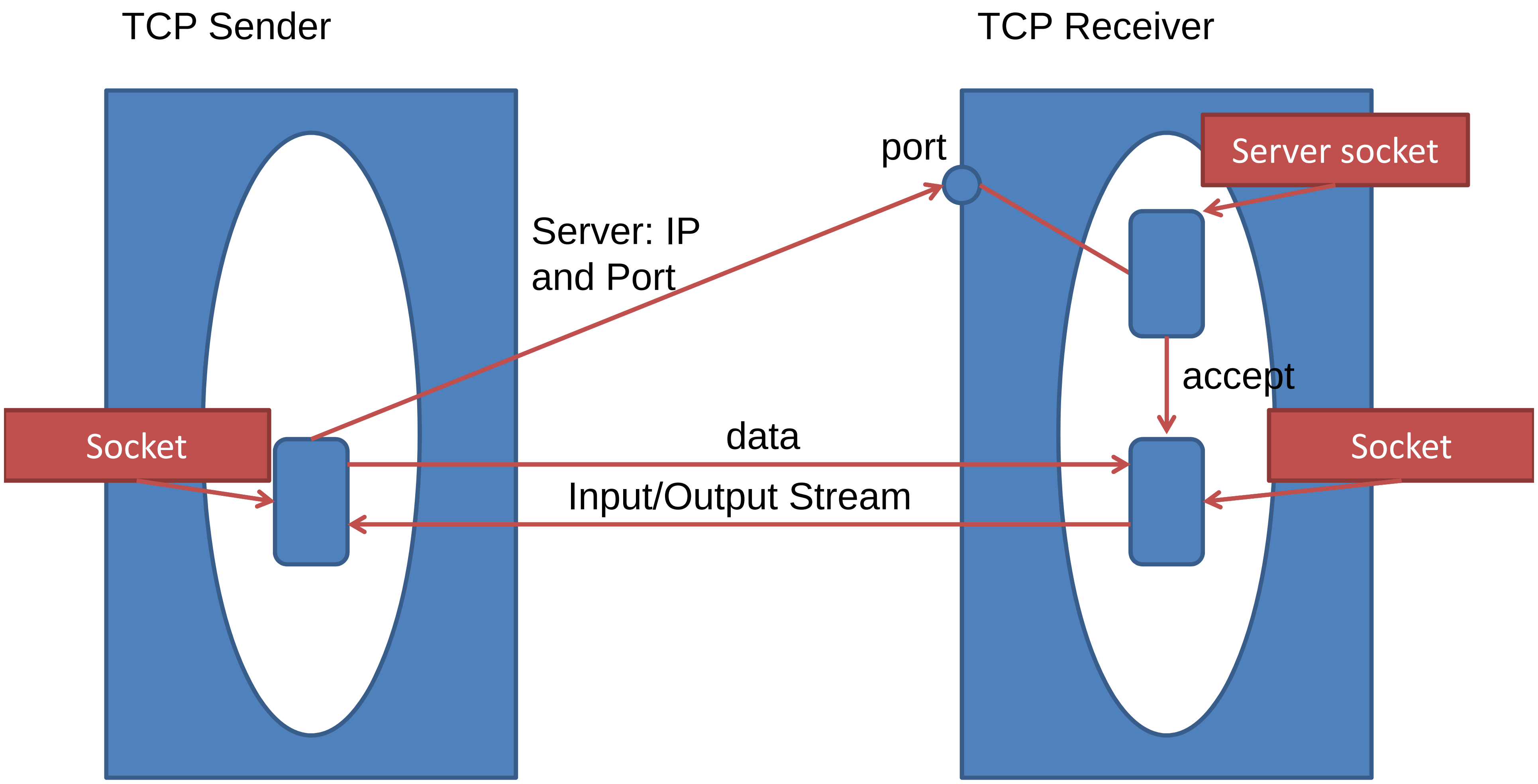
Note: aSocket is an instantiated instance of the Java DatagramSocket class.

Interested readers are referred to the course text (pages 138 and 139) for more detailed implementations of UDP client and servers

TCP

Transmission Control Protocol

How does it work?



API & IP: TCP communication

- API to TCP uses a stream abstraction
  - A stream of bytes

- The stream abstraction hides the following:
  - Message sizes
  - Lost messages
  - Flow control
  - Message duplication and ordering
  - Destinations

API & IP: TCP streaming

- How does it work? In a nutshell
  - Client requests a stream
  - Client creates stream socket to any port
  - Client sends a *connect* request to a server at the server specific port
  - The server is bound to a designated server port and listens for *connect* messages from clients
  - Once a connect message is accepted the server uses a new socket for the stream
  - The client is informed of the socket
  - The client then connects to the dedicated socket

API & IP: TCP streaming

- How does it work? More detailed
  - The server is bound to a designated server port and listens for *connect* messages from clients
  - A client requests a stream and creates a socket to any port to send its request from
  - The client then sends a *connect* request to a server at the server specific port
  - Once a connect message is accepted the server creates a another socket for the stream
  - The client is informed of the new socket
  - The server send information via the new socket
  - The client and server communicate using input and output streams through their sockets.

API & IP: TCP streaming

- Dealing with failures
  - TCP does not provide reliable communication as it can fail
    - The sender stops sending if too many dropped or fail checksums or the network fails
    - NOTE: The cause of the failure cannot be known by each process (i.e. network failure or process failure)

API & IP: TCP streaming

- Example uses of TCP
    - HTTP
    - FTP
    - Telnet
    - SMTP
- Frequently using reserved port numbers

API & IP: TCP communication

- In Java
  - Both the server and client have two classes:
    - **DataInputStream** and a **DataOutputStream**
  - A **Socket** class is used to retrieve each stream
    - **aClientSocket.getInputStream()**
    - **sClientSocket.getOutputStream()**
  - The streams can be read or write to using
    - **outputStream.writeUTF(data)**
    - **inputStream.readUTF(data)**

Each stream is a string of bits representing in binary primitive data types.

Other classes include:
ServerSocket (used for the server to listen for client stream request)