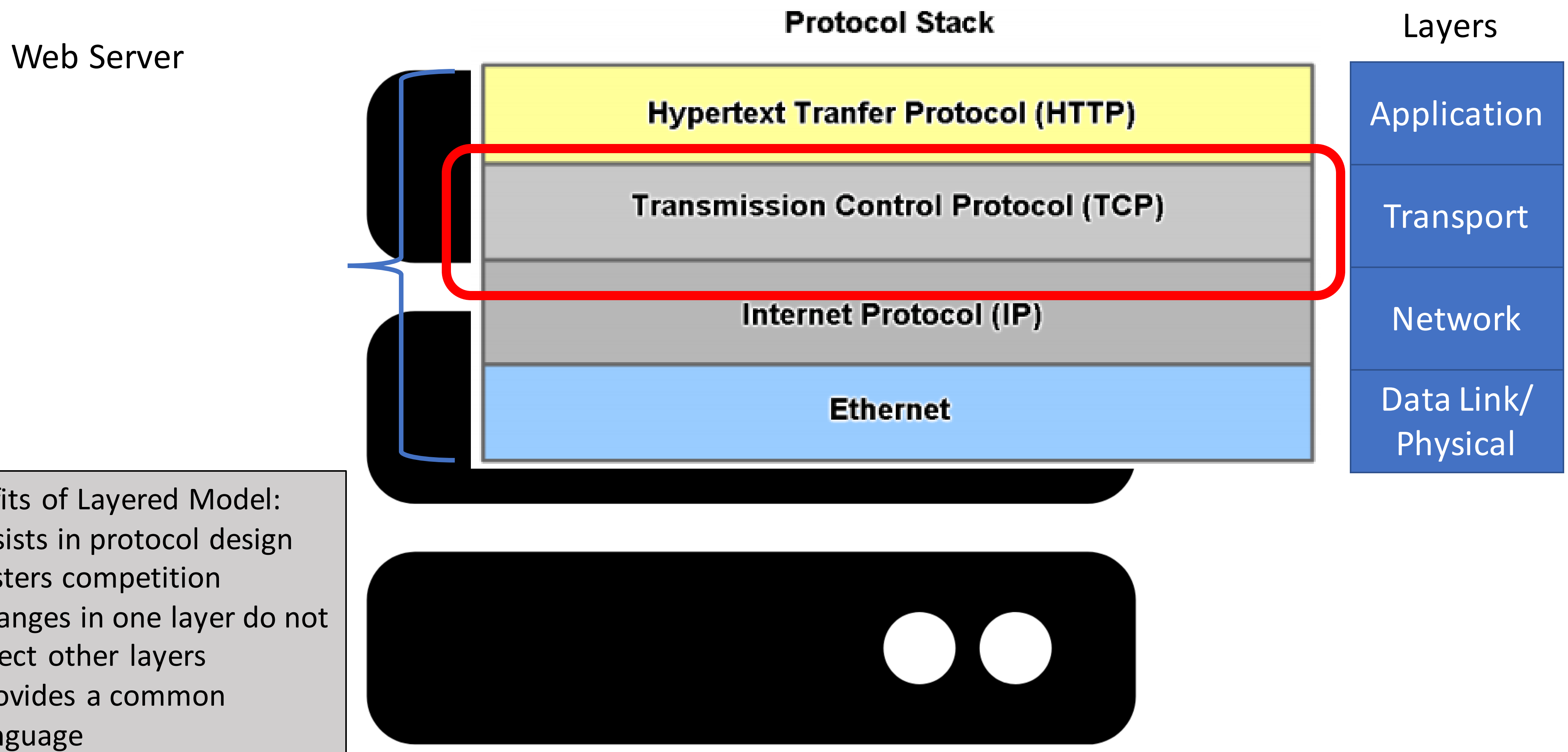
TCP/UDP

Recap: Layered Model Network Comms



Benefits of Layered Model:

- assists in protocol design
- fosters competition
- changes in one layer do not affect other layers
- provides a common language

Ethernet

- Data Link Layer protocol
- Ethernet (IEEE 802.3) is widely used.
- Supported by a variety of physical layer implementations.
- Multi-access (shared medium).

CSMA/CD

- *Carrier Sense Multiple Access with Collision Detection*

- *Carrier Sense* : can tell when another host is transmitting

- *Multiple Access* : many hosts on 1 wire

- *Collision Detection* : can tell when another host transmits at the same time.

WiFi

- Data Link Layer protocol
- IEEE 802.11
- Supported by a variety of physical layer implementations.
- Multi-access (shared medium).

CSMA/CA

- *Carrier Sense Multiple Access with Collision Avoidance*
- *Carrier Sense* : can tell when another host is transmitting
- *Multiple Access* : many hosts on 1 wire
- *Collision Avoidance* : can tell if it's "safe" to transmit.

Physical Addressing

- Every interface has a unique 48 bit address (a.k.a. *hardware address*).
  - Example: **C0:B3:44:17:21:17**
  - The broadcast address is all 1's(Fs).
  - Addresses are assigned to vendors by a central authority.

- Each interface looks at every *frame* and inspects the destination address. If the address does not match the hardware address of the interface (or the broadcast address), the frame is discarded.

Internet Protocol

- IP is the network layer
  - packet delivery service (host-to-host).
  - translation between different data-link protocols

- IP provides connectionless, unreliable delivery of *IP datagrams.*
  - *Connectionless*: each datagram is independent of all others.
  - *Unreliable:* there is no guarantee that datagrams are delivered correctly or even delivered at all.

IP Addresses

- IP addresses are not the same as the underlying data-link (MAC) addresses.

- IP is a network layer - it must be capable of providing communication between hosts on different kinds of networks (different data-link implementations).

- The address must include information about what *network* the receiving host is on. This is what makes routing feasible.

Network and Host IDs

- A Network ID is assigned to an organization by a global authority.
- Host IDs are assigned locally by a system administrator.
- Both the Network ID and the Host ID are used for routing.

Host and Network Addresses

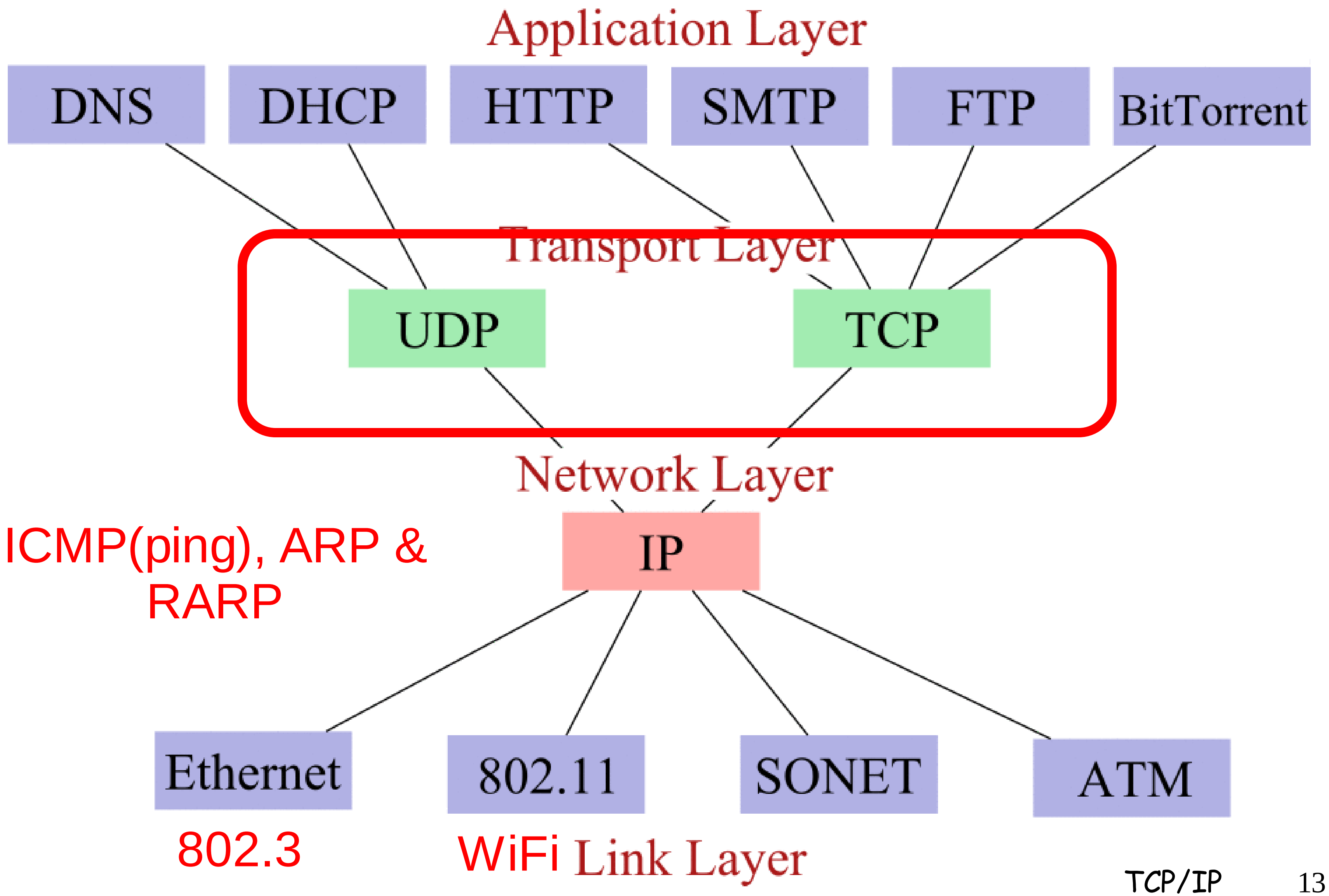
- A single network interface is assigned a single IP address called the *host* address.
- A host may have multiple interfaces, and therefore multiple *host* addresses.
- Hosts that share a network all have the same IP *network* address (the network ID).
- An IP address that has a host ID of all 0s is called a *network address* and refers to an entire network.

Transport Layer & TCP/IP

- IP is the network layer, so TCP must be the transport layer?

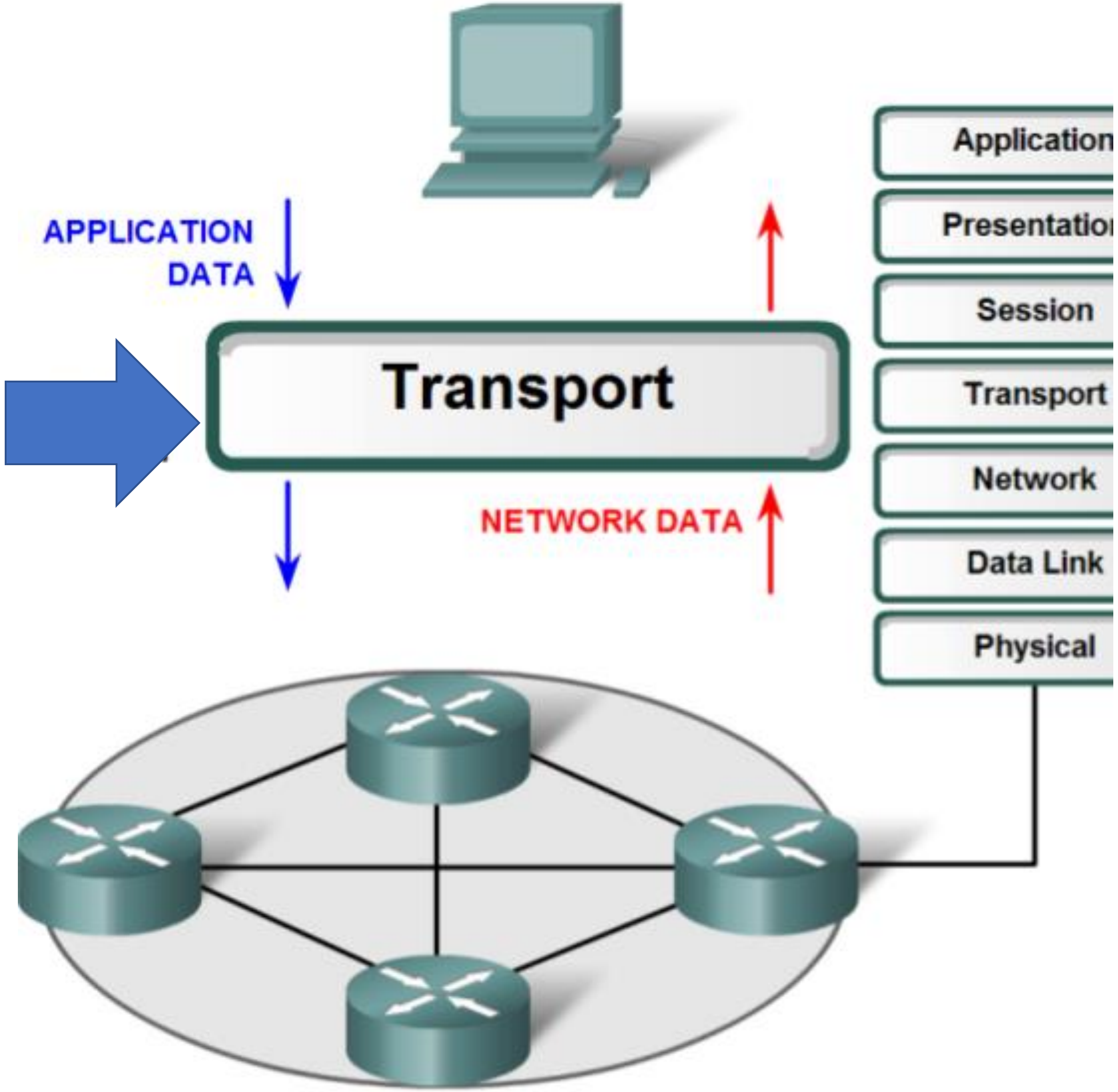
TCP is only part of the TCP/IP transport layer - the other part is UDP (User Datagram Protocol).

The Internet Hourglass



Transport Layer Role

- Prepares application data for transport over a network
- Processes network data for use by apps



Transport Layer Purpose

- Segments data and reassebles segements into various communication streams as follows:
  - Tracks individual communication between apps on source and destination hosts
    - A host can have multiple simultanious networked apps (e.g. browser, Skype etc.
  - Segements data and manages each piece
    - The Transport layer segments data from the Application layer
    - Each piece of application data requires headers to indicate to which communication it is associated with
  - Reassembles segments into application data
    - At a receiving host, individual pieces of data must also be reconstructed into a complete data stream that is useful to the Application layer.
  - Identifies different applications
    - Transport layer must be able to identify target apps: Uses Port Numbers

Choosing Transport Layer Protocols

- Application developers choose based on the nature of the app

- IP Telephony
- Video Streaming
- Frequent Sensor data

Requirements:
- Fast
- Low overhead
- No need for acknowedgements
- No need to resend lost data
- Delivers data as it arrives.

- SMPT/POP (email)
- HTTP (web page)
- Command data

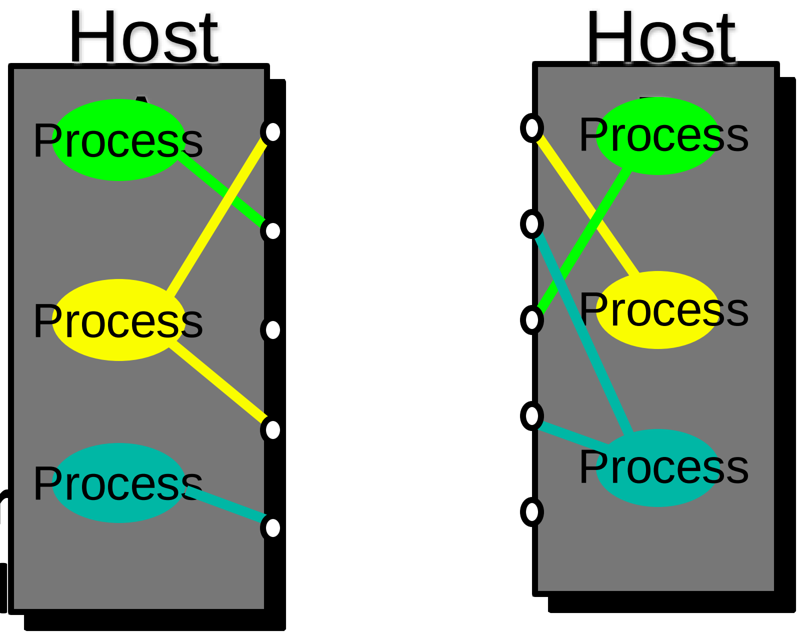
Requirements:
- Reliable
- Acknowedge data
- Resend lost data
- Delivers data in order sent.

UDP - User Datagram Protocol

- UDP is a transport protocol
  - communication between processes
- UDP uses IP to deliver datagrams to the right host.
- UDP uses ***ports*** to provide communication services to individual apps/processes.

Ports

- TCP/IP uses an abstract destination point called a protocol port.
- Ports are identified by a positive integer.
    - **Port number & IP address allow any process/application in any computer on Internet to be uniquely identified**
- Operating systems provide a mechanism that apps & processes use to specify a port.



Ports & UDP

- Source/destination port: port numbers identify sending & receiving processes
- Ports can be static or dynamic
  - Static (< 1024) assigned centrally, known as well known ports
  - Dynamic (can be used by any computer application program to communicate with any other application program, 49152-65535 for Windows)
- Message length in bytes includes the UDP header and data

UDP

- Datagram Delivery
- Connectionless
- Unreliable
- Minimal

UDP Datagram Format

Source Port	Destination Port
Length	Checksum
Data	

TCP
Transmission Control Protocol

- TCP is an alternative transport layer protocol supported by TCP/IP.

- TCP provides:
  - Connection-oriented
  - Reliable
  - Full-duplex
  - Byte-Stream

Connection-Oriented

- *Connection oriented* means that a virtual connection is established before any user data is transferred.

- If the connection cannot be established, the user program is notified (finds out).

- If the connection is ever interrupted, the user program(s) is finds out there is a problem.

Reliable

- *Reliable* means that every transmission of data is acknowledged by the receiver.
- Reliable does not mean that things don't go wrong, it means that we find out when things go wrong.
- If the sender does not receive acknowledgement within a specified amount of time, the sender retransmits the data.

Byte Stream

- *Stream* means that the connection is treated as a stream of bytes.

- The user application does not need to package data in individual datagrams (as with UDP).

Buffering

- TCP is responsible for buffering data and determining when it is time to send a datagram.

- It is possible for an application to tell TCP to send the data it has buffered without waiting for a buffer to fill up.

Full Duplex

- TCP provides transfer in both directions (over a single virtual connection).
- To the application program these appear as 2 unrelated data streams, although TCP can piggyback control and data communication by providing control information (such as an ACK) along with user data.

TCP Ports

- Interprocess communication via TCP is achieved with the use of ports (just like UDP).

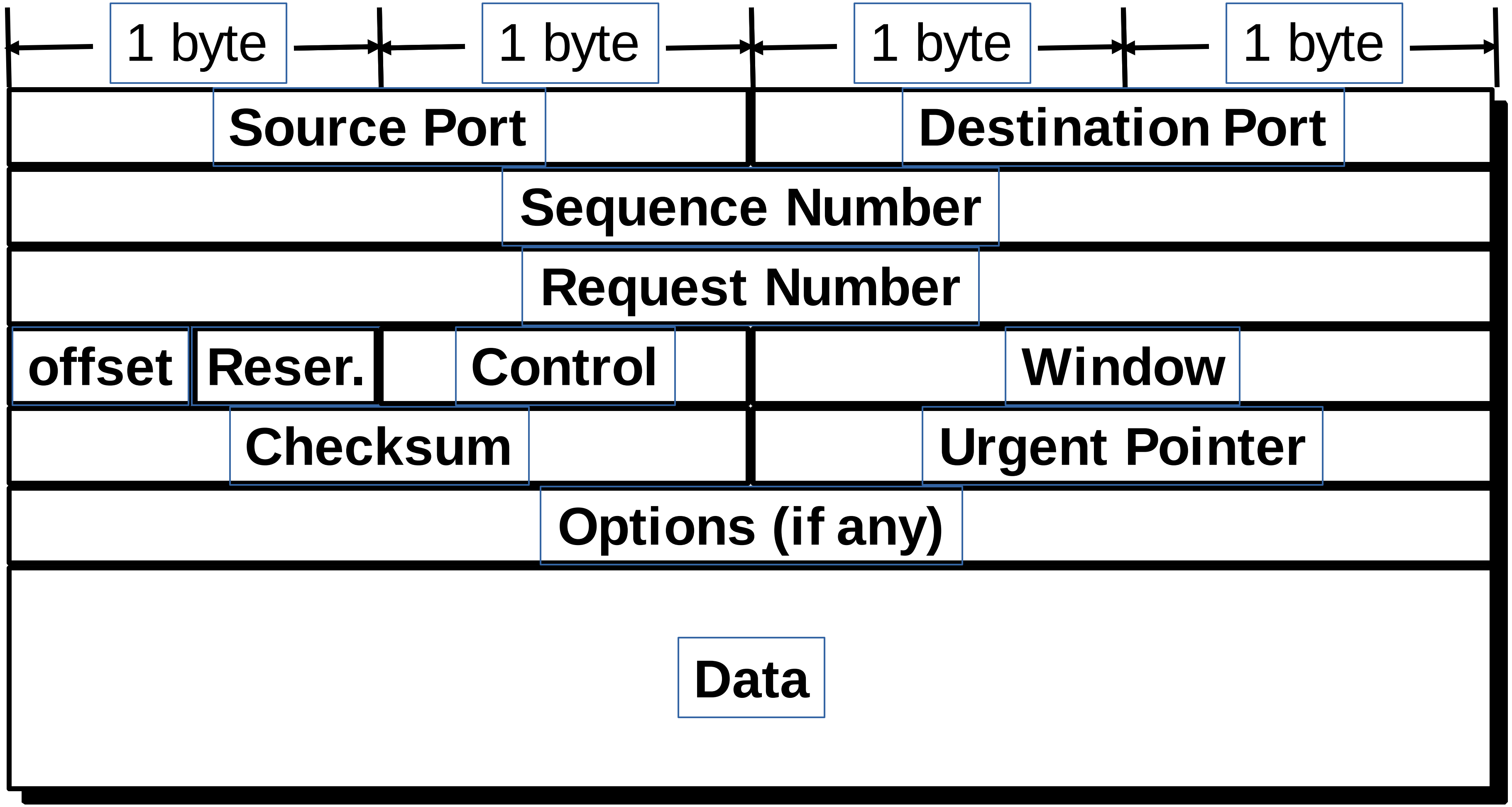
- UDP ports have no relation to TCP ports (different name spaces).

TCP Segments

- The chunk of data that TCP asks IP to deliver is called a *TCP segment*.

- Each segment contains:
  - data bytes from the byte stream
  - control information that identifies the data bytes

TCP Segment Format



Addressing in TCP/IP

- Each TCP/IP address includes:
  - Internet Address
  - Protocol (UDP or TCP)
  - Port Number

Remember: TCP/IP is a *protocol suite* that includes IP, TCP and UDP

TCP vs. UDP

Q: Which protocol is better ?

A: It depends on the application.

TCP provides a connection-oriented, reliable, byte stream service (lots of overhead).

UDP offers minimal datagram delivery service (as little overhead as possible).

TCP Process

- When a client requests a connection, it sends a “SYN” segment (a special TCP segment) to the server port.
- SYN stands for *synchronize*. The SYN message includes the client’s ISN.
- ISN is Initial Sequence Number.

TCP Process 2

- Every TCP segment includes a *Sequence Number* that refers to the first byte of *data* included in the segment.

- Every TCP segment includes a *Request Number* (*Acknowledgement Number*) that indicates the byte number of the next data that is expected to be received.
  - All bytes up through this number have already been received.

TCP Process 3

- There are a bunch of control flags:
  - URG: urgent data included.
  - ACK: this segment is (among other things) an acknowledgement.
  - RST: error - abort the session.
  - SYN: synchronize Sequence Numbers (setup)
  - FIN: polite connection termination.

TCP Process 4

- MSS: Maximum segment size (A TCP option)

- Window: Every ACK includes a Window field that tells the sender how many bytes it can send before the receiver will have to toss it away (due to fixed buffer size).

TCP Connection Creation

- Programming details later - for now we are concerned with the actual communication.

- A *server* accepts a connection.
  - Must be looking for new connections!

- A *client* requests a connection.
  - Must *know* where the server is!

Client Starts

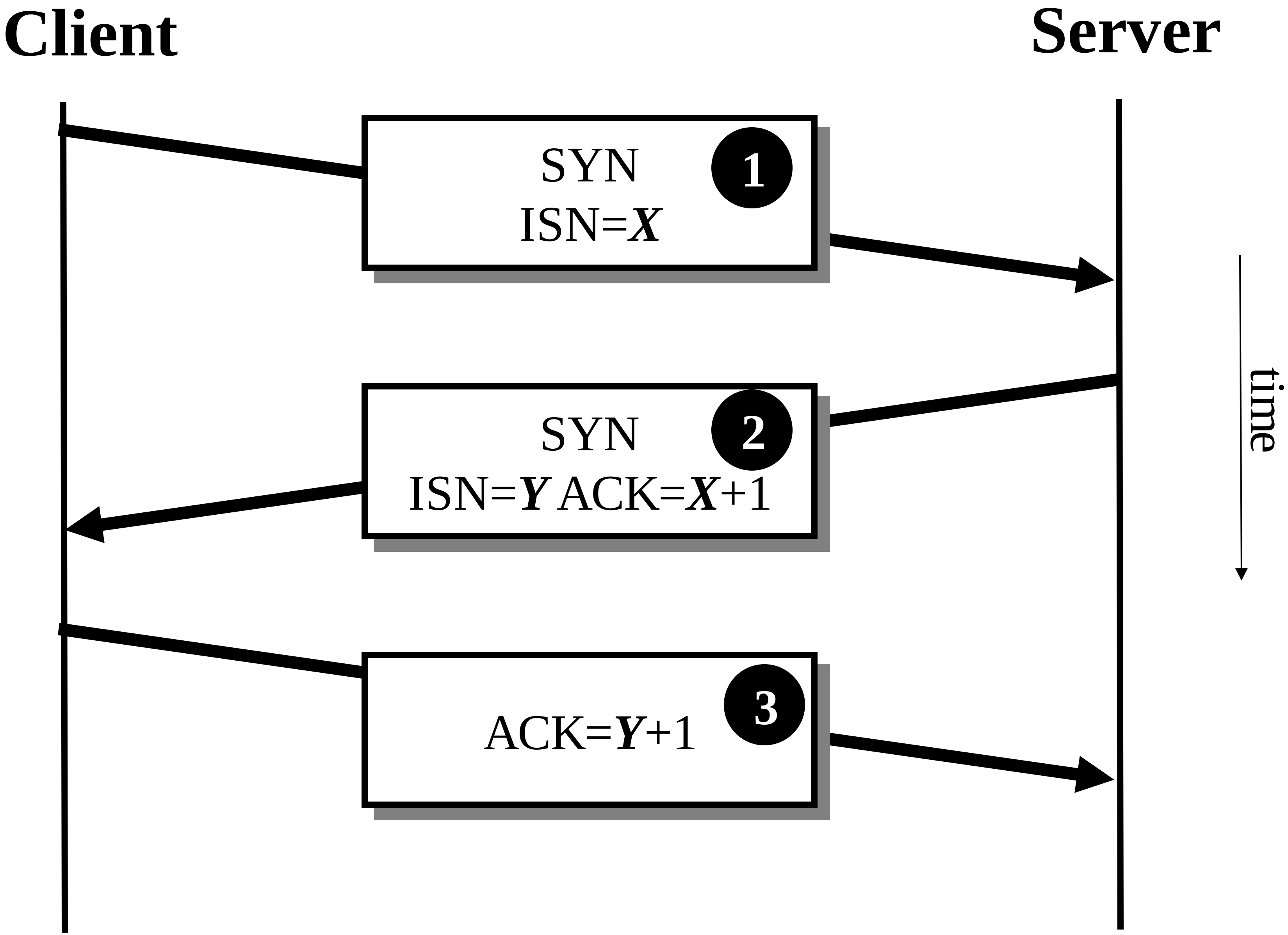
- A client starts by sending a SYN segment with the following information:
    - Client's ISN (generated pseudo-randomly)
    - Maximum Receive Window for client.
    - Optionally (but usually) MSS (largest datagram accepted).
    - No payload! (Only TCP headers)

Sever Response

- When a waiting server sees a new connection request, the server sends back a SYN segment with:
  - Server's ISN (generated pseudo-randomly)
  - Request Number is Client ISN+1
  - Maximum Receive Window for server.
  - Optionally (but usually) MSS
  - No payload! (Only TCP headers)

Finally

- When the Server's SYN is received, the client sends back an ACK with:
    - Request Number is Server's ISN+1


Client
Server
SYN
ISN=X
1
SYN
ISN=Y ACK=X+1
2
ACK=Y+1
3
time

TCP Data and ACK

- Once the connection is established, data can be sent.
- Each data segment includes a sequence number identifying the first byte in the segment.
- Each segment (data or empty) includes a request number indicating what data has been received.

TCP Buffers

- The TCP layer doesn't know when the application will ask for any received data.
  - TCP buffers incoming data so it's ready when we ask for it.
- Both the client and server allocate buffers to hold incoming and outgoing data
  - The TCP layer does this.
- Both the client and server announce with every ACK how much buffer space remains (the Window field in a TCP segment).

Send Buffers

- The application gives the TCP layer some data to send.
- The data is put in a send buffer, where it stays until the data is ACK'd.
  - it has to stay, as it might need to be sent again!
- The TCP layer won't accept data from the application unless (or until) there is buffer space.

ACKs

- A receiver doesn't have to ACK every segment (it can ACK many segments with a single ACK segment).
- Each ACK can also contain outgoing data (piggybacking).
- If a sender doesn't get an ACK after some time limit (MSL) it resends the data.

TCP Segment Order

- Most TCP implementations will accept out-of-order segments (if there is room in the buffer).
- Once the missing segments arrive, a single ACK can be sent for the whole thing.
- Remember: IP delivers TCP segments, and IP in not reliable - IP datagrams can be lost or arrive out of order.

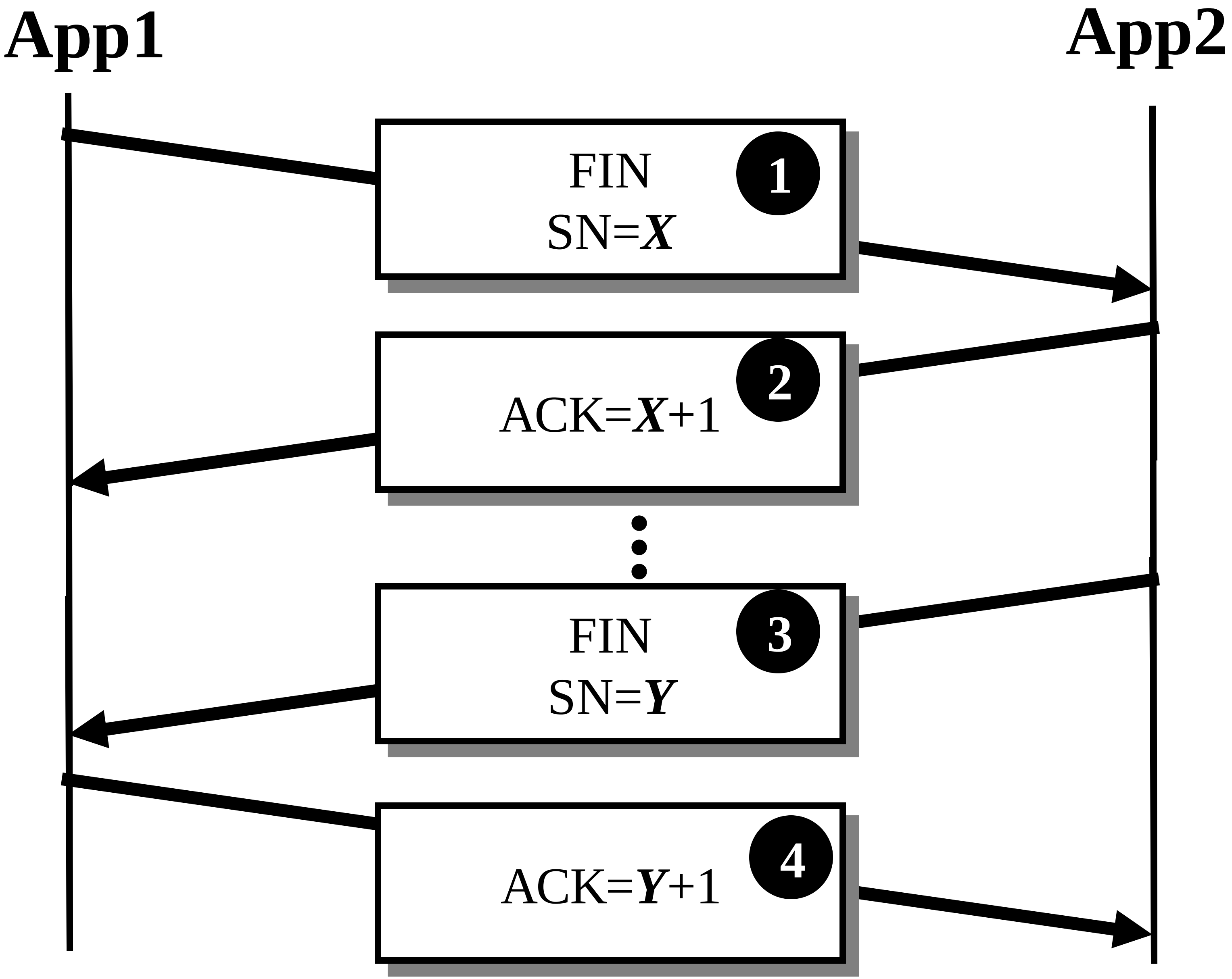
Termination

- The TCP layer can send a RST segment that terminates a connection if something is wrong.

- Usually the application tells TCP to terminate the connection politely with a FIN segment.

FIN

- Either end of the connection can initiate termination.
- A FIN is sent, which means the application is done sending data.
- The FIN is ACK’d.
- The other end must now send a FIN.
- That FIN must be ACK’d.


App1
App2
FIN
SN=X
1
2
ACK=X+1
3
FIN
SN=Y
4
ACK=Y+1

TCP Termination

1. App1: “I have no more data for you”.

2. App2: “OK, I understand you are done sending.”

dramatic pause…

3. App2: “OK - Now I’m also done sending data”.

4. App1: “Roger, Over and Out, Goodbye, Astalavista Baby, Adios, It’s been real ...”

camera fades to black ...

TCP TIME_WAIT

- Once a TCP connection has been terminated (the last ACK sent) there is some unfinished business:
  - What if the ACK is lost? The last FIN will be resent and it must be ACK'd.
  - What if there are lost or duplicated segments that finally reach the destination after a long delay?
- TCP hangs out for a while to handle these situations.

QI

- Why is a 3-way handshake necessary?
    - HINTS: TCP is a reliable service, IP delivers each TCP segment, IP is not reliable.

- Who sends the first FIN - the server or the client?

- Once the connection is established, what is the difference between the operation of the server's TCP layer and the client's TCP layer?