Computer Systems and Networks

ECPE 170 – Jeff Shafer – University of the Pacific

Networking Fundamentals

Lab Schedule

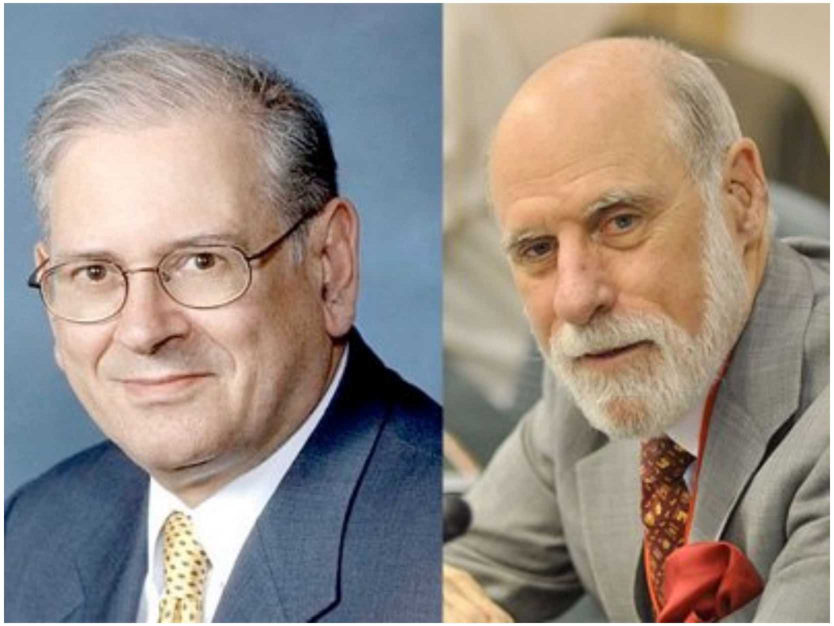
Activities

- **This Week**
  - Python introduction
  - Networking introduction
  - Endianness *(Thursday)*
  - **Lab 8** (HTTP, TCP sockets)

Assignments Due

- **Lab 7**
  - **Due by Mar 16th 5:00am**
- **Lab 8**
  - **Due by Mar 23rd 5:00am**

Persons of the Day: Vint Cerf / Bob Kahn



- Co-designers of TCP/IP protocol suite
  - Enables reliable communication across unreliable network
  - **Foundation of Internet**
- 2004 *ACM Turing Award* winners (shared)
- 2005 *Presidential Medal of Freedom winners* (shared)

Person of the Day: Tim Berners-Lee

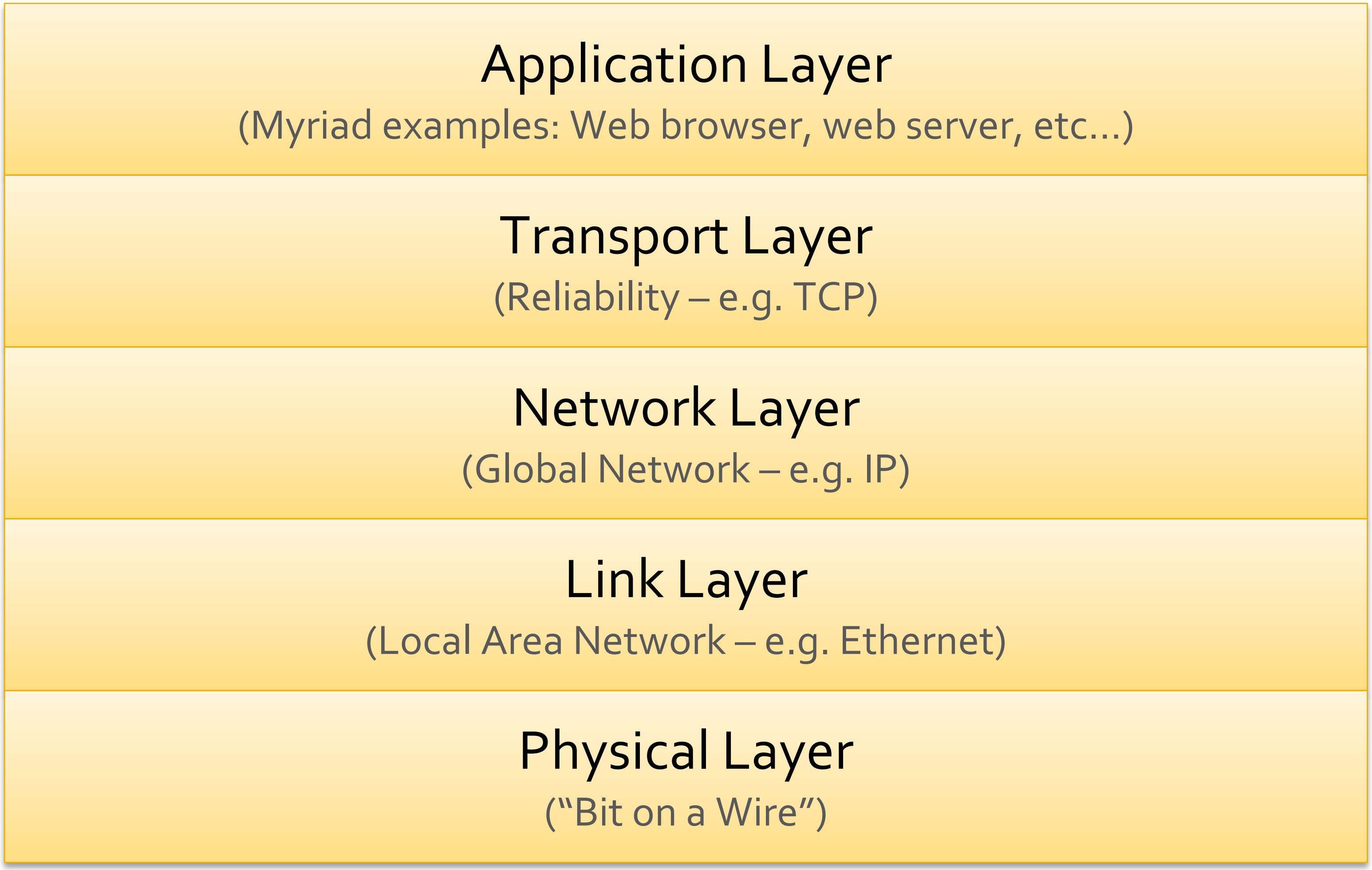
- Inventor of “World Wide Web”
  - First implementation of **HTTP** (HyperText Transfer Protocol) to communicate between client and server
- Knighted by Queen Elizabeth II in 2004

Computer Networks

Disclaimer

- **These topics take an entire semester of COMP 177 (Computer Networking) to explore!**
- A few days *(most of which is lab time)* is only sufficient for the briefest of overviews…

Network Model



Application Layer

(Myriad examples: Web browser, web server, etc…)

Transport Layer

(Reliability – e.g. TCP)

Network Layer

(Global Network – e.g. IP)

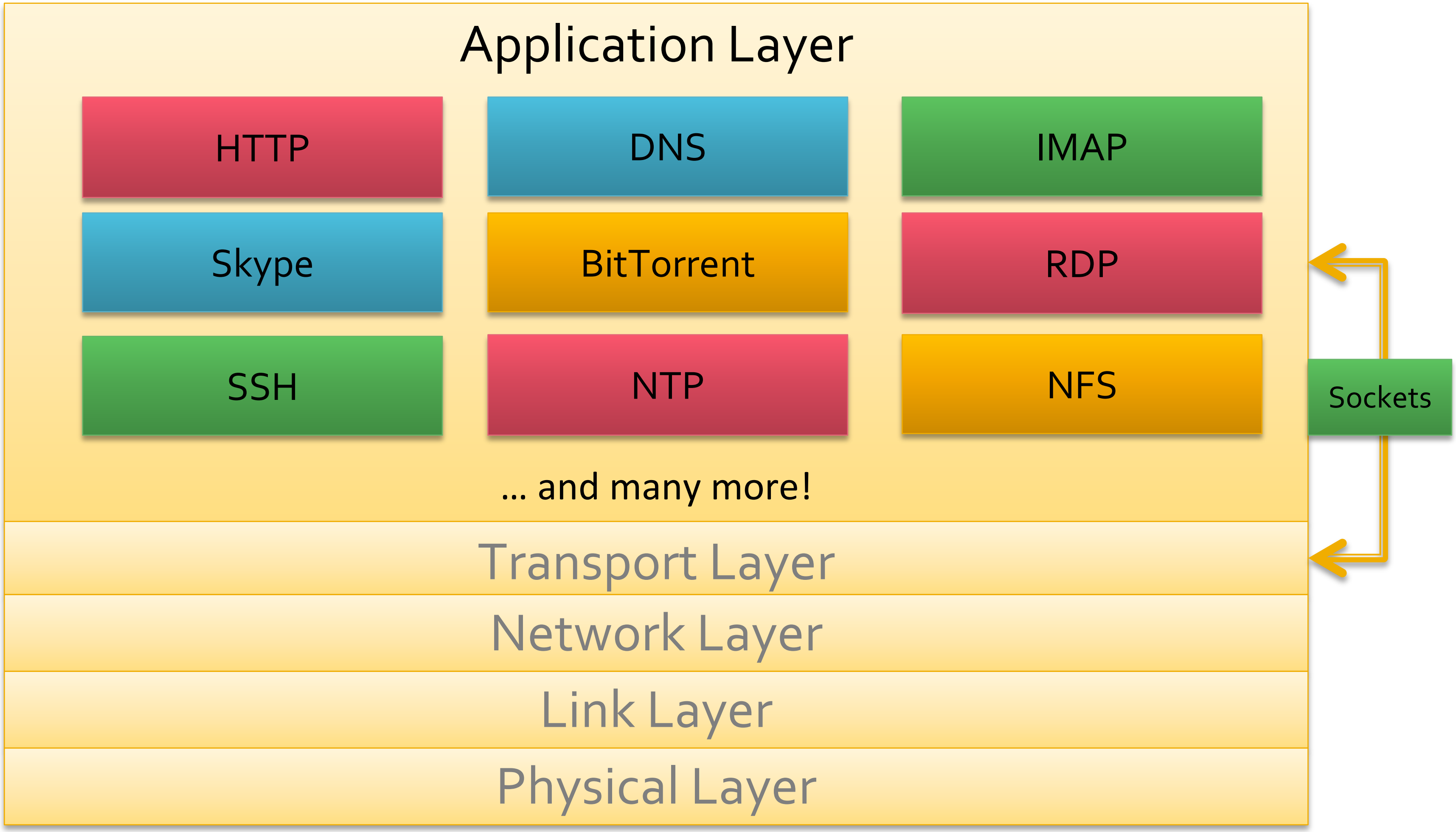
Link Layer

(Local Area Network – e.g. Ethernet)

Physical Layer

(“Bit on a Wire”)

Application Layer



Application Layer

- The **application layer** programmer can make many (fantastic) assumptions about the network
  - The network is reliable
    - Messages are not lost
    - Messages are received in the order they are sent
  - The network can transfer data of infinite length (you can send as much data as desired)
  - You can deliver messages directly to a specific application on a specific computer anywhere on the planet
- The lower layers (transport, network, link, …) do all the heavy-lifting to make these assumptions true

Client-Server Architecture

Server

- Always-on host
- Always has a known IP address
- Lots of bandwidth
- **Server process**: process that waits to be contacted

Client

- Communicate with server
- May be intermittently connected
- May have dynamic IP addresses
- Do not communicate directly with each other
- **Client process**: process that initiates communication

Why Do We Have Sockets?

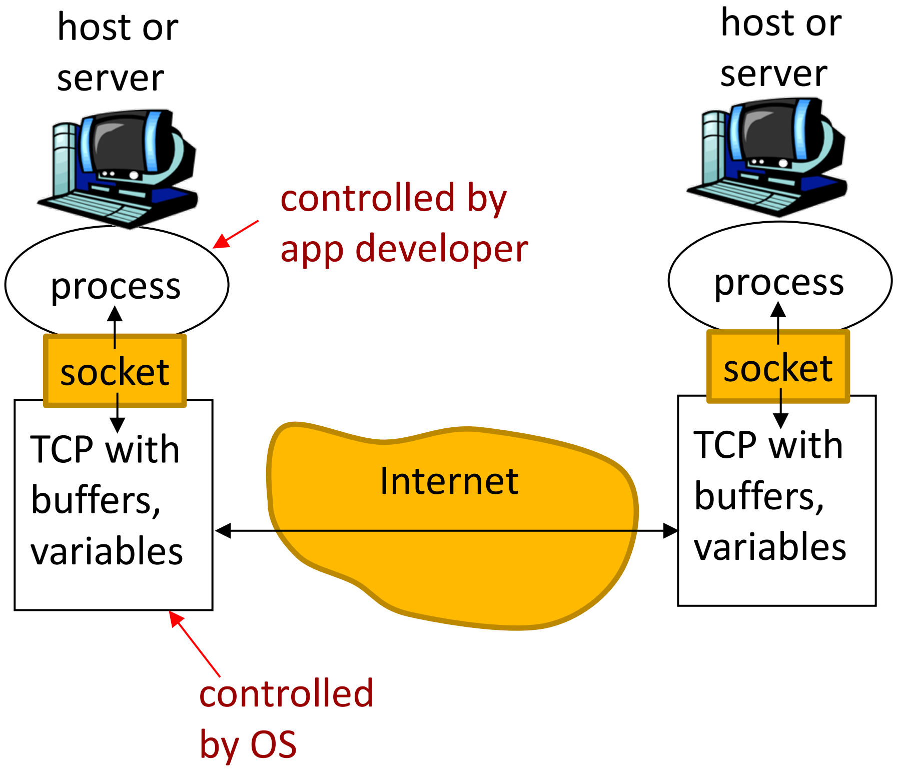
- Challenge – **Inter-process communication**
- A **process** is an independent program running on a host
  - Separate memory space
- How do processes communicate with other processes
  - On the same host?
  - On different hosts?
- Send **messages** between each other

What is a Socket?

- An interface between process (application) and network
  - The application creates a socket
  - The socket *type* dictates the style of communication
    - Reliable vs. best effort
    - Connection-oriented vs. connectionless
- Once configured the application can
  - Pass data to the socket for network transmission
  - Receive data from the socket (transmitted through the network by some other host)

What is a Socket?

- Process sends/receives messages to/from its socket
- Socket analogous to door
  - Sending process shoves message out door
  - Transport infrastructure on other side of door carries message to socket at receiving process
  - **Imagine you are just writing to a file…**
- API allow customization of socket
  - Choose transport protocol
  - Choose parameters of protocol

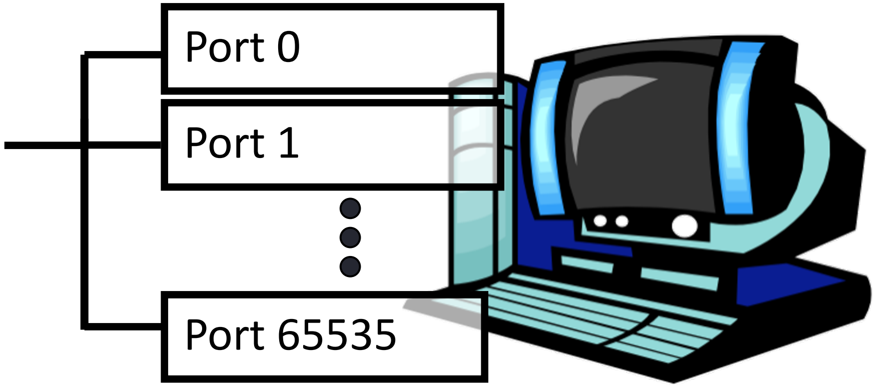

host or server
process
socket
TCP with buffers, variables
controlled by app developer
Internet
controlled by OS
host or server
process
socket
TCP with buffers, variables

Addressing Processes

- To receive messages, each process on a host must have an **identifier**
  - IP addresses are unique
  - **Is this sufficient?**
- No, there can thousands of processes running on a single machine (with one IP address)
- Identifier must include
  - IP address
  - **and** port number (example: 80 for web)

Ports

- Each host has 65,536 ports
- Some ports are *reserved for specific apps*
  - FTP (20, 21), Telnet (23), HTTP (80), etc…
- Outgoing ports (on clients) can be dynamically assigned by OS in upper region (above 49,152) – called **ephemeral ports**
- See http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers



Socket Usage: Client Program

- Basic socket functions for **connection-oriented (TCP) <u>clients</u>**

1.	`socket()`	create the socket descriptor
2.	`connect()`	connect to the remote server
3.	`send()`,`recv()`	communicate with the server
4.	`close()`	end communication by closing socket descriptor

Application-Layer Protocol

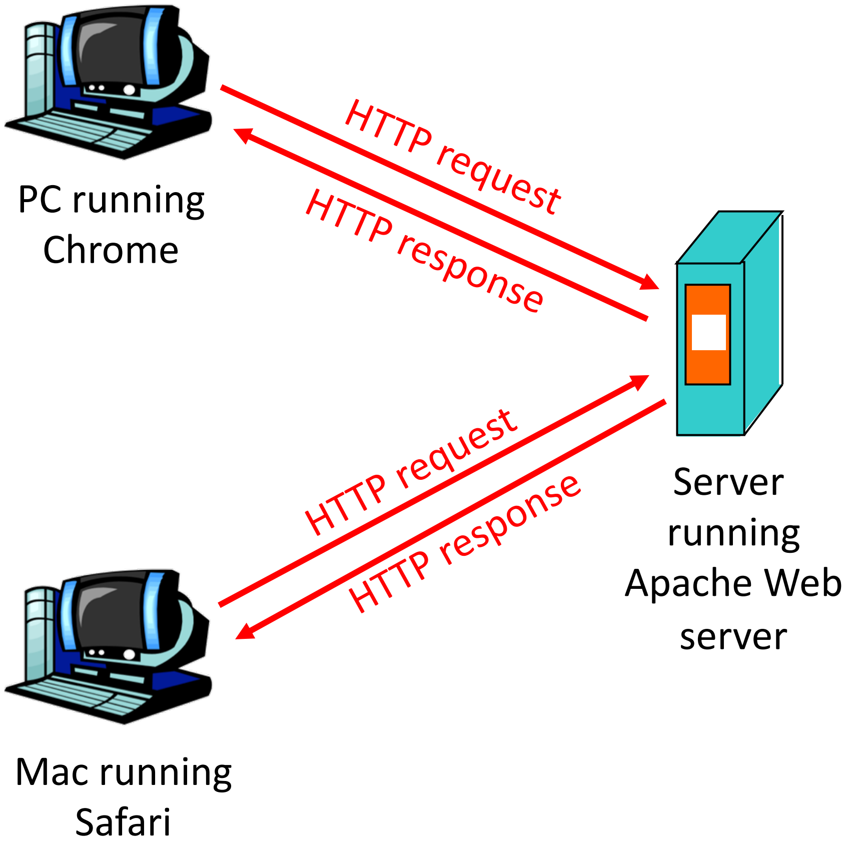
- Sockets just allow us to send raw messages between processes on different hosts
  - Transport service takes care of moving the data
- **What** exactly is sent is up to the application
  - An **application-layer** protocol
  - HTTP, NTP, IMAP, SFTP, Skype, etc…

Application-Layer Protocol

- Both the client and server speaking the protocol must agree on
  - **Types of messages exchanged**
    - e.g., request, response
  - **Message syntax**
    - What fields are in messages
    - How fields are delineated
  - **Message semantics**
    - Meaning of information in fields
  - Rules for **when** and **how** processes send and respond to messages

Hypertext Transfer Protocol Overview

- **HTTP** is the *application layer protocol* for the web
- It is how the client and server communicate
- Client/server model
  - **Client**: browser that requests, receives, "displays" Web objects
  - **Server**: Web server sends objects in response to requests

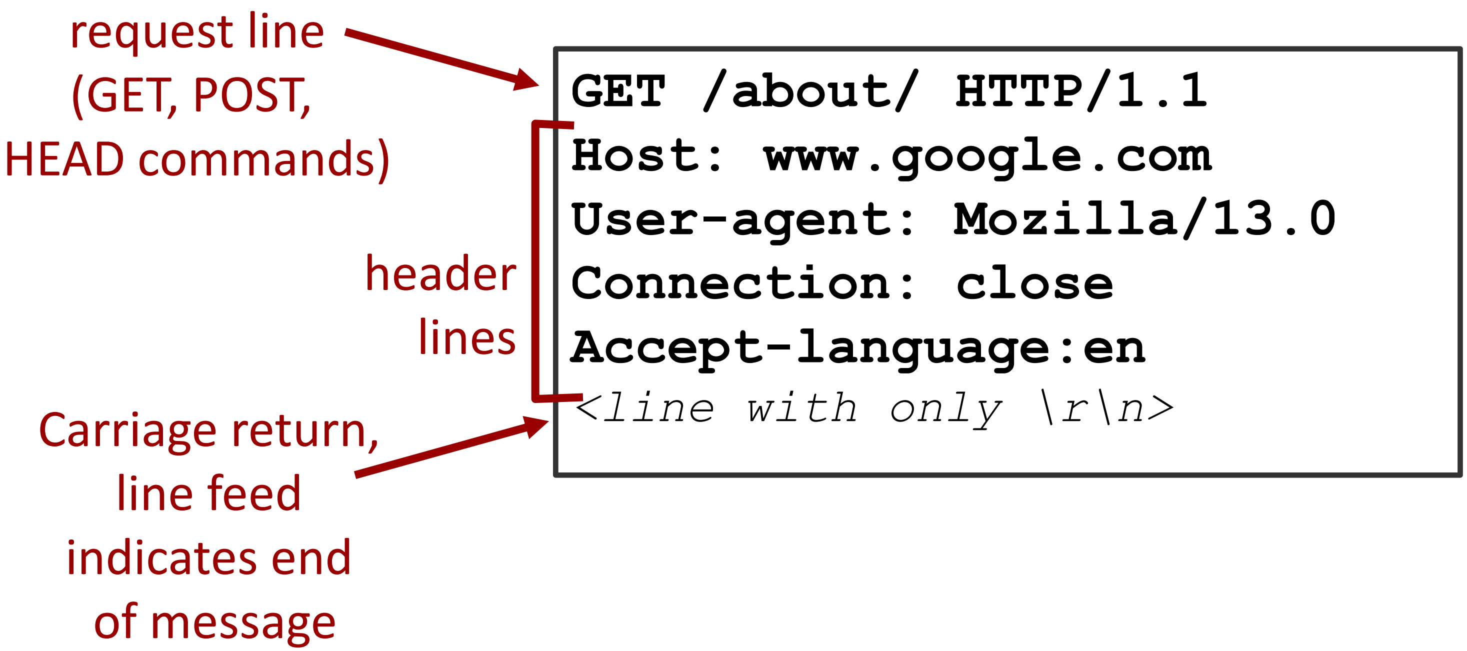


Web and HTTP

- Web **page** consists of base HTML file and (potentially) many referenced **objects**
  - HTML file, PNG image, Flash video, …
- Each object is addressable by a **URL**
- Example URL:

$$\underbrace{\texttt{www.somecompany.com}}_{\text{host name}}\underbrace{\texttt{/someDept/image.png}}_{\text{path name}}$$

HTTP Request Message (Client->Server)



request line (GET, POST, HEAD commands)

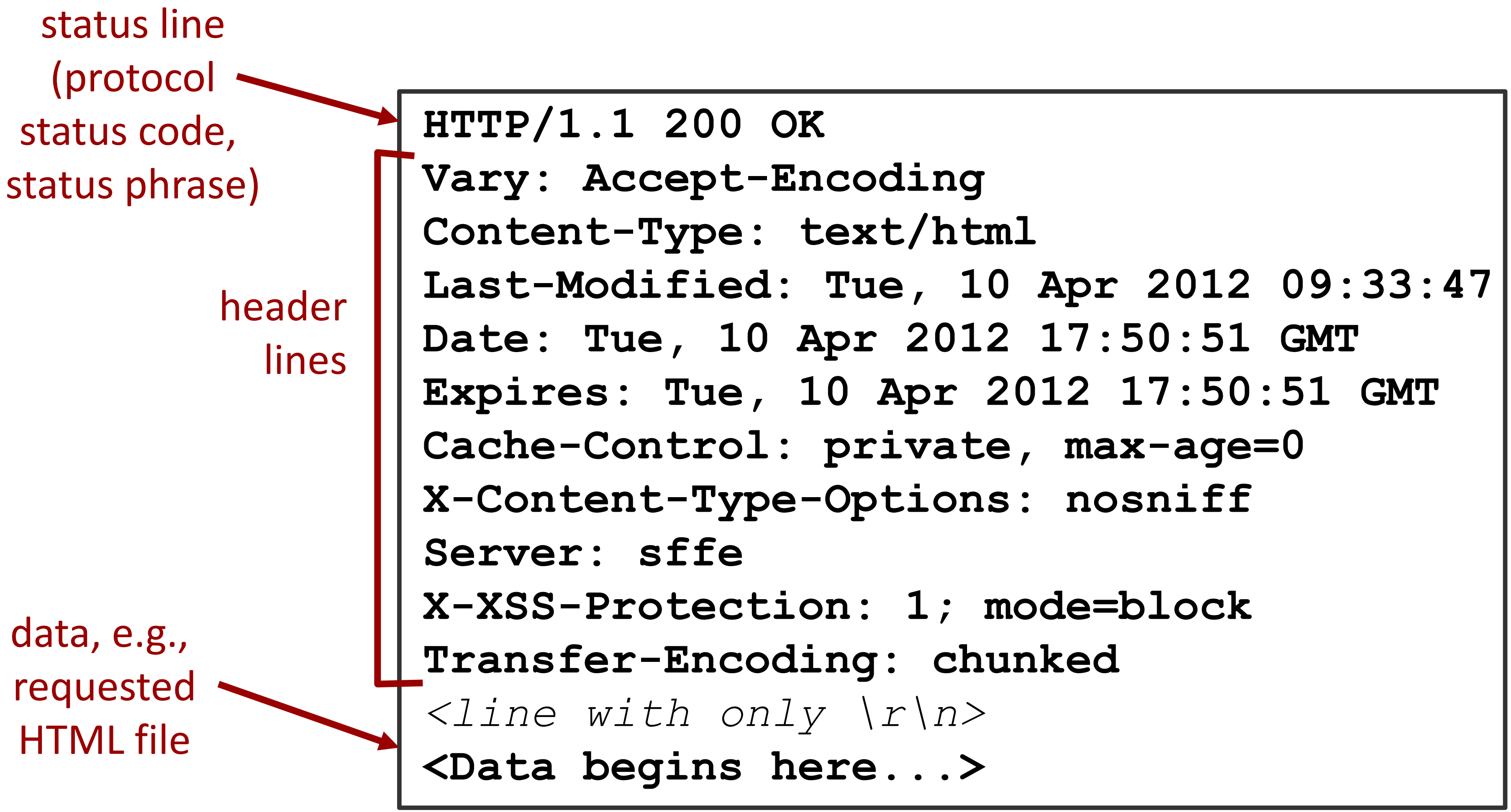
header lines

Carriage return, line feed indicates end of message

```
GET /about/ HTTP/1.1
Host: www.google.com
User-agent: Mozilla/13.0
Connection: close
Accept-language:en
<line with only \r\n>
```

HTTP is a text-based protocol. The client sends ASCII bytes in the request, and the server responds with ASCII bytes in the reply.

HTTP Response Message (Server -> Client)



status line (protocol status code, status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK
Vary: Accept-Encoding
Content-Type: text/html
Last-Modified: Tue, 10 Apr 2012 09:33:47
Date: Tue, 10 Apr 2012 17:50:51 GMT
Expires: Tue, 10 Apr 2012 17:50:51 GMT
Cache-Control: private, max-age=0
X-Content-Type-Options: nosniff
Server: sffe
X-XSS-Protection: 1; mode=block
Transfer-Encoding: chunked
<line with only \r\n>
<Data begins here...>
```

HTTP Response Status Codes

A few examples out of many!

200 OK

- Request succeeded, requested object later in this message

301 Moved Permanently

- Requested object moved, new location specified later in this message (Location:)

400 Bad Request

- Request message not understood by server

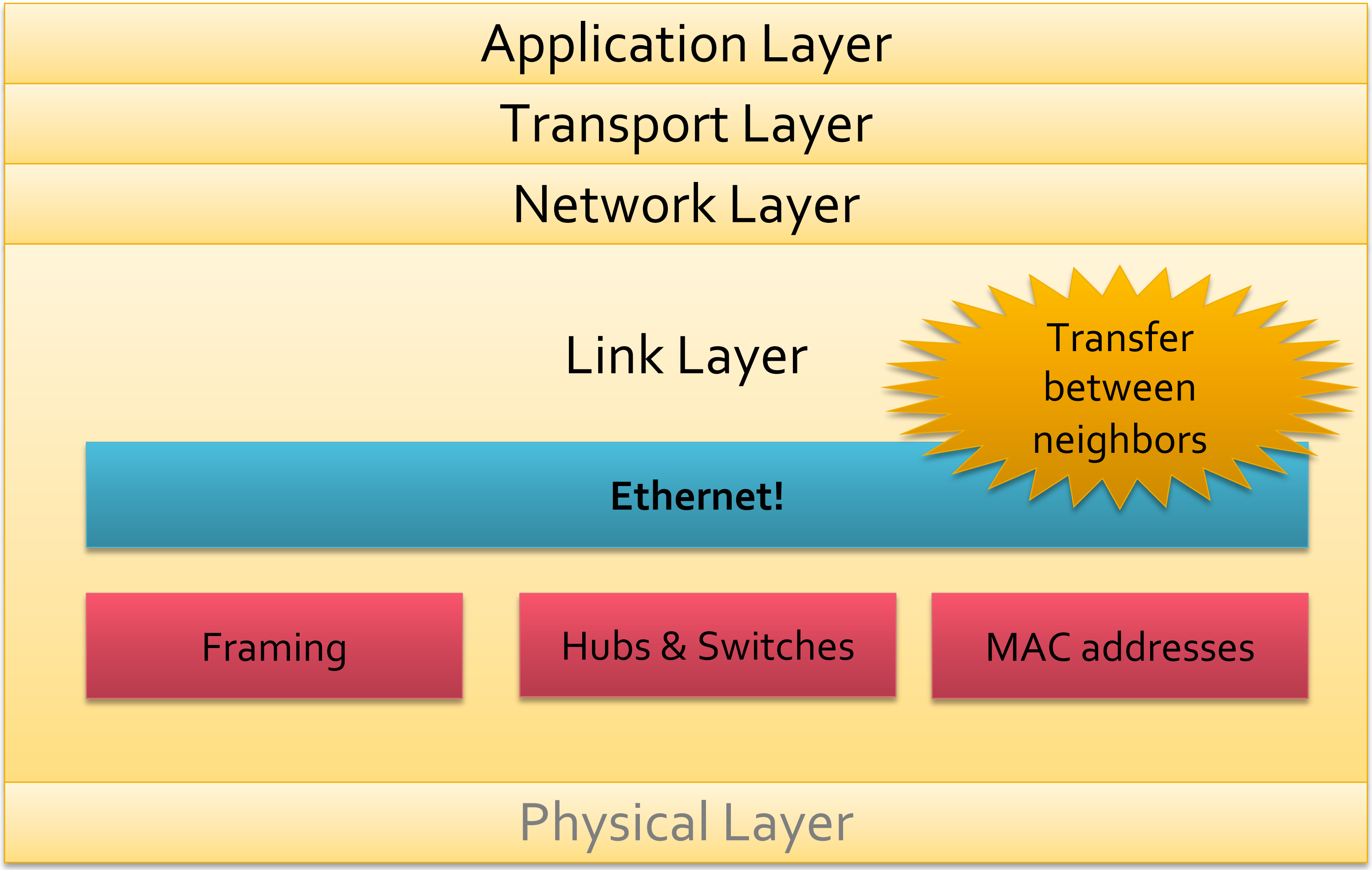
404 Not Found

- Requested document not found on this server

505 HTTP Version Not Supported

Other Layers

Link Layer



Application Layer

Transport Layer

Network Layer

Link Layer

Transfer between neighbors

Ethernet!

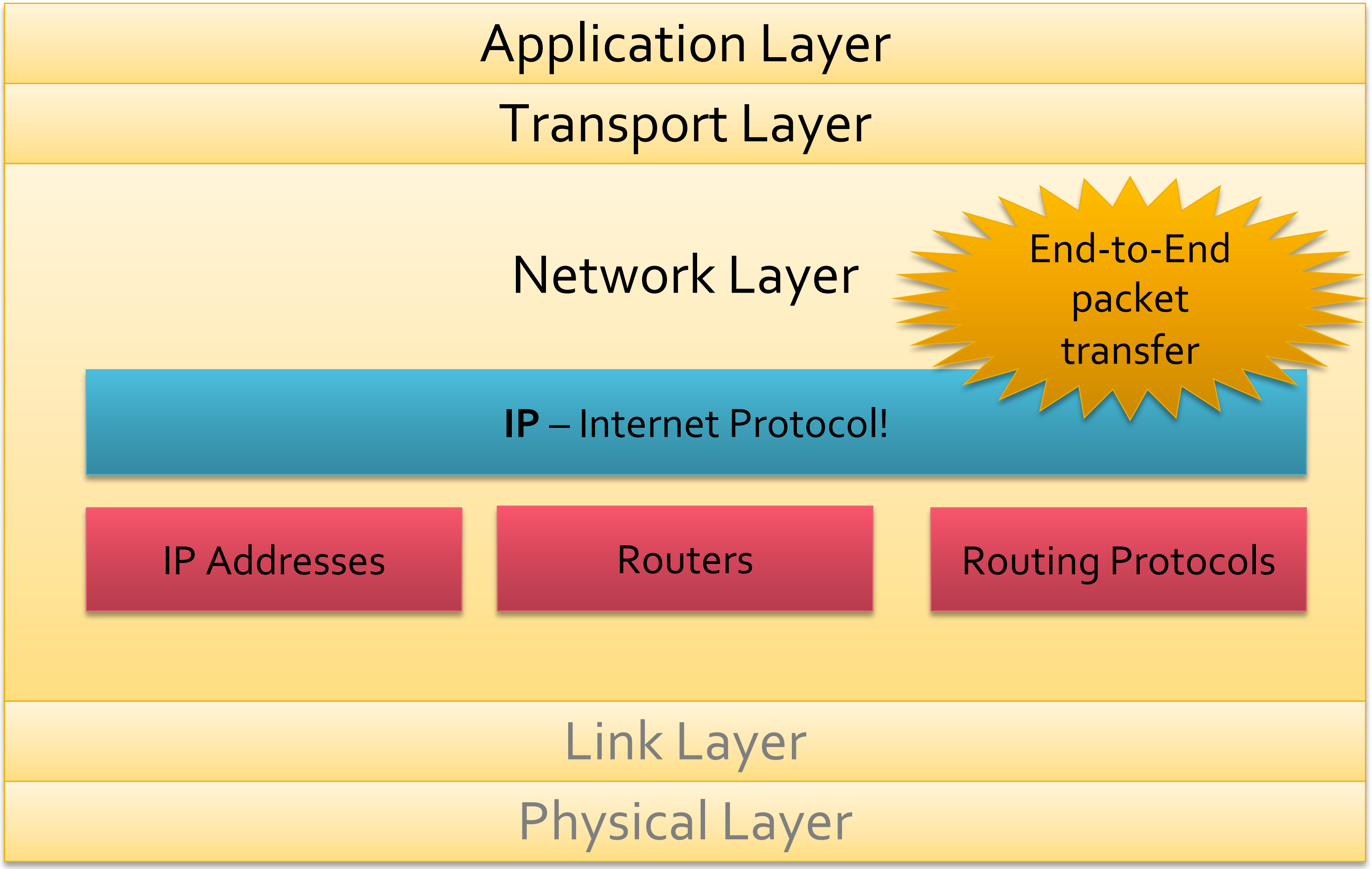
Framing

Hubs & Switches

MAC addresses

Physical Layer

Network Layer



Application Layer

Transport Layer

Network Layer

End-to-End packet transfer

IP – Internet Protocol!

IP Addresses

Routers

Routing Protocols

Link Layer

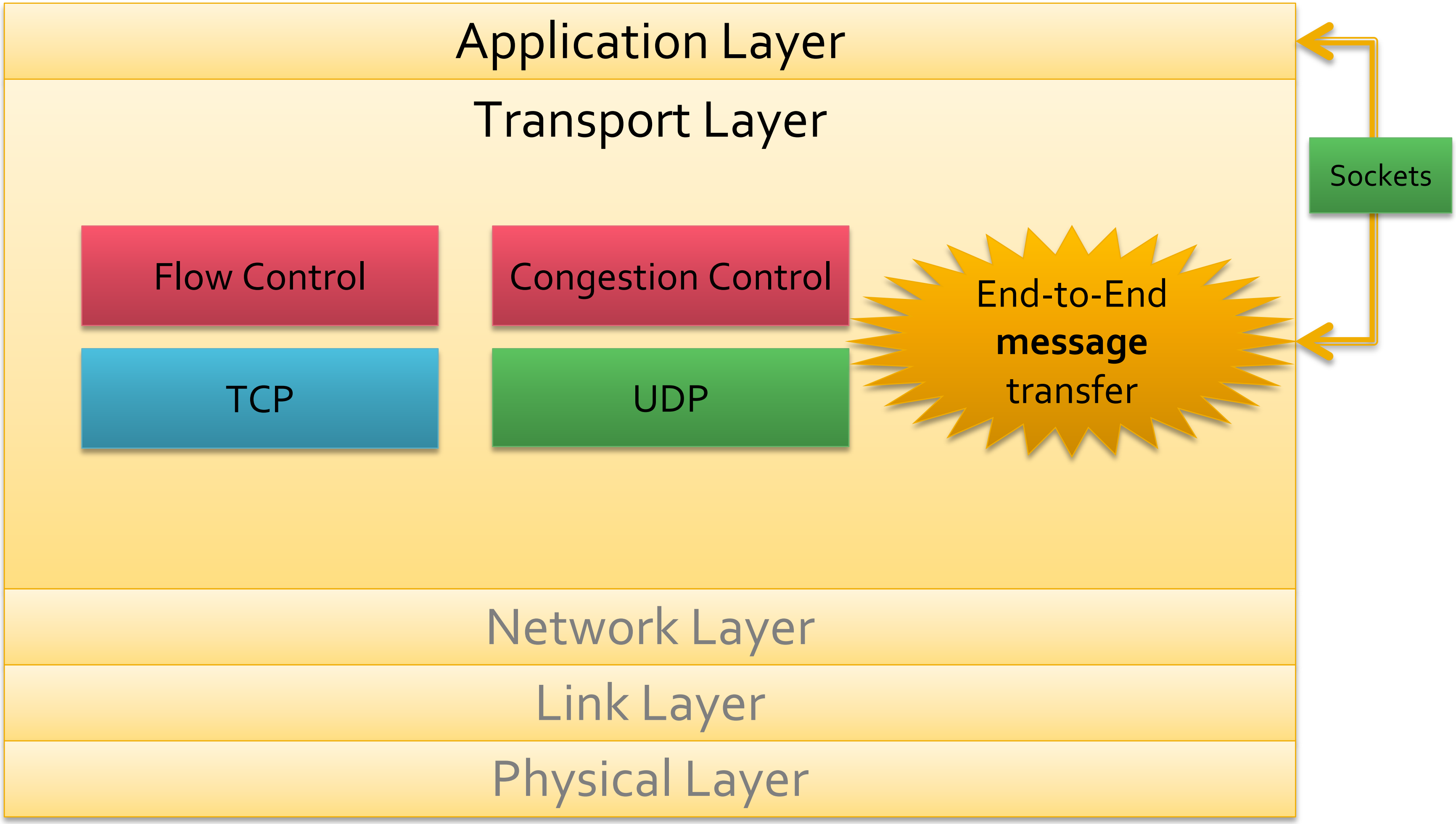
Physical Layer

IP Properties

- **Datagram**
  - Each packet is **individually routed**
  - Packets may be **fragmented** or **duplicated** by underlying networks
- **Connectionless**
  - No guarantee of delivery in sequence
- **Unreliable**
  - No guarantee of delivery
  - No guarantee of integrity of data
- **Best effort**
  - Only drop packets when necessary
  - No time guarantee for delivery

Ethernet networks provide the same "guarantees"

Transport Layer



Transport Layer

- **Link Layer**
  - **IP**: Un-reliable, order not guaranteed, delivery of **individual messages**

- **Transport Layer**
  (*Choose between these with your sockets*)
  - **UDP**: Un-reliable, order not guaranteed, delivery of individual messages
  - **TCP**: Reliable, in-order delivery of data **stream**
    - TCP is built on top of IP!

Demos

Demos

1. Walkthrough of `client.py` and `server.py` demo programs
2. Impersonate web browser via Telnet (HTTP request to http://neverssl.com/)
3. Run `display.py` with example image
4. Monitor `display.py` with *Wireshark* and examine packet trace