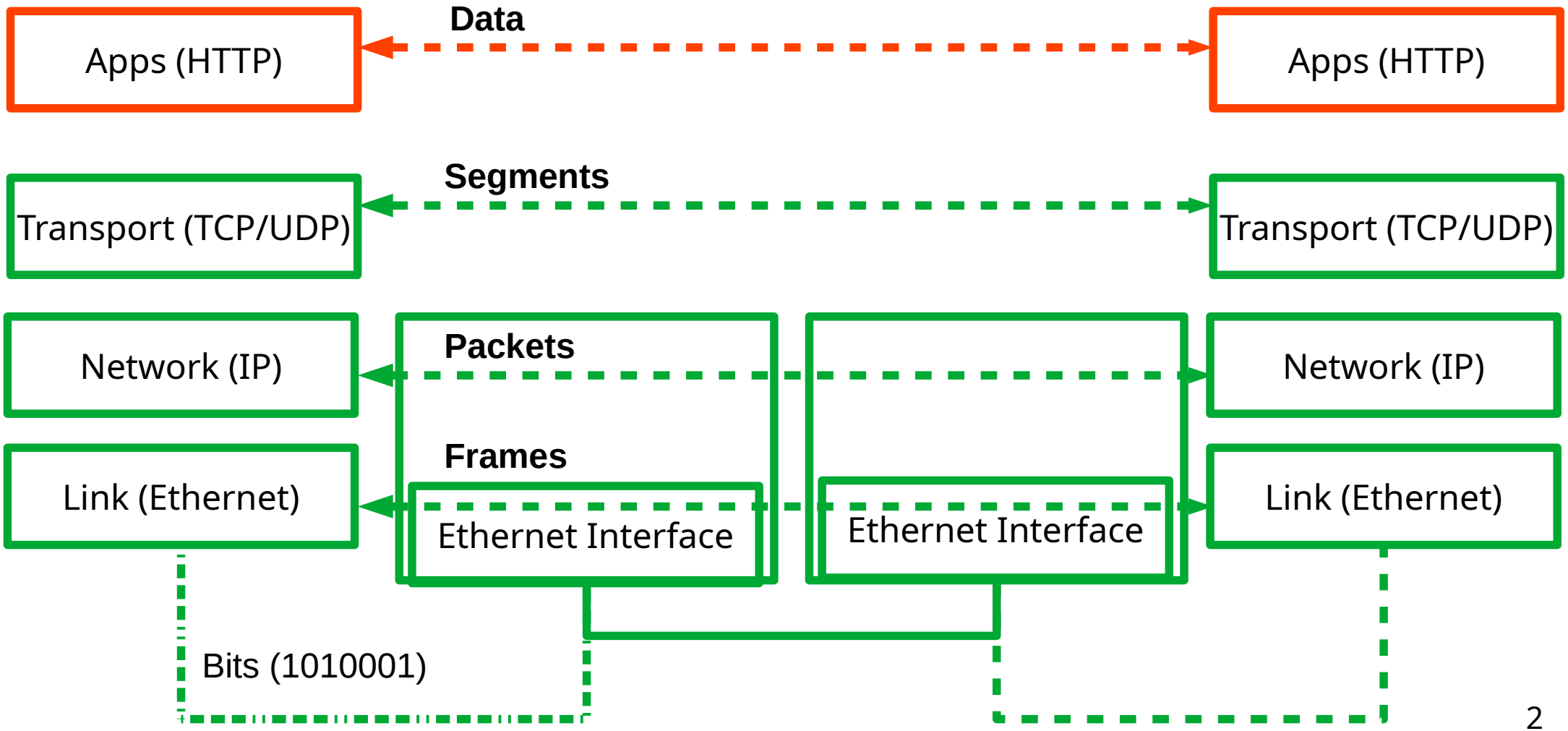
CSC2770 – INTRO TO SYSTEMS AND NETWORKING

Instructor: Susmit Shannigrahi

DNS AND NETWORKED APPLICATIONS

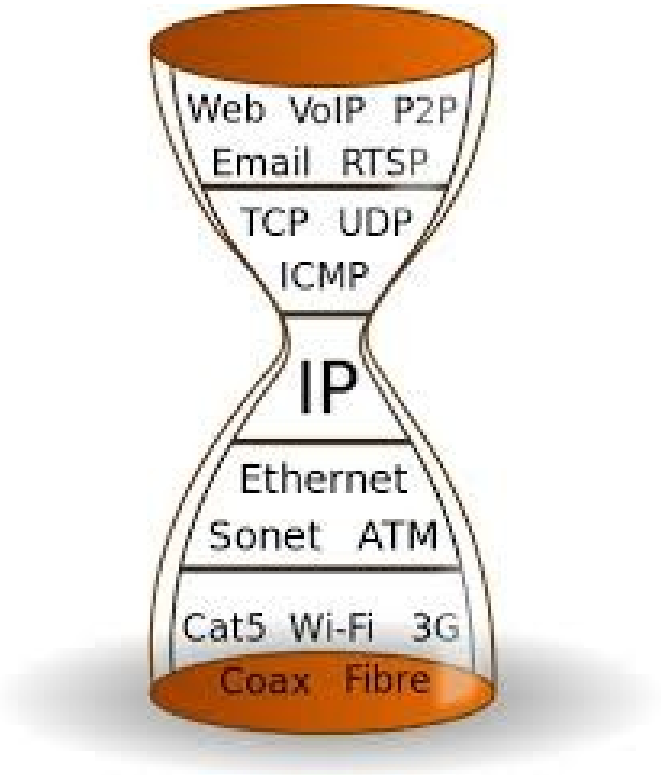
sshannigrahi@tntech.edu

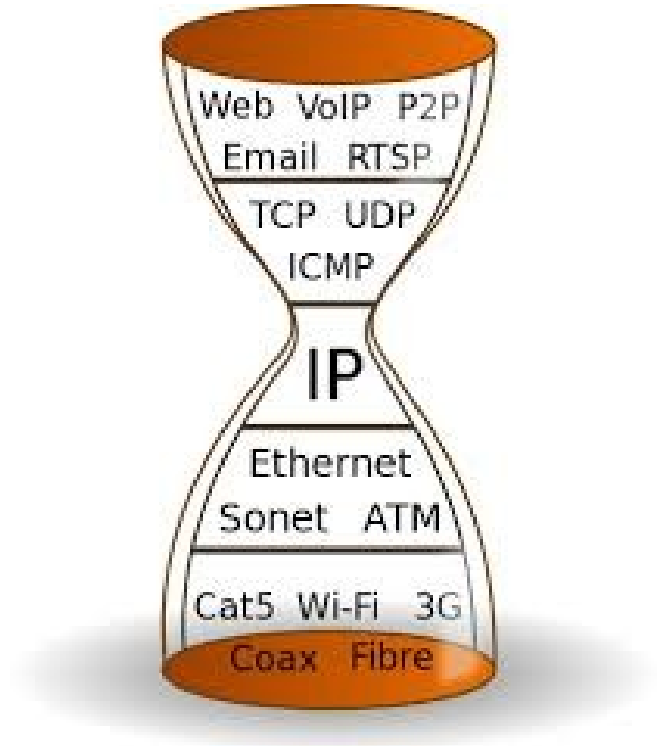
Tennessee TECH


Data
Apps (HTTP)
Apps (HTTP)
Segments
Transport (TCP/UDP)
Transport (TCP/UDP)
Packets
Network (IP)
Network (IP)
Frames
Link (Ethernet)
Ethernet Interface
Ethernet Interface
Link (Ethernet)
Bits (1010001)

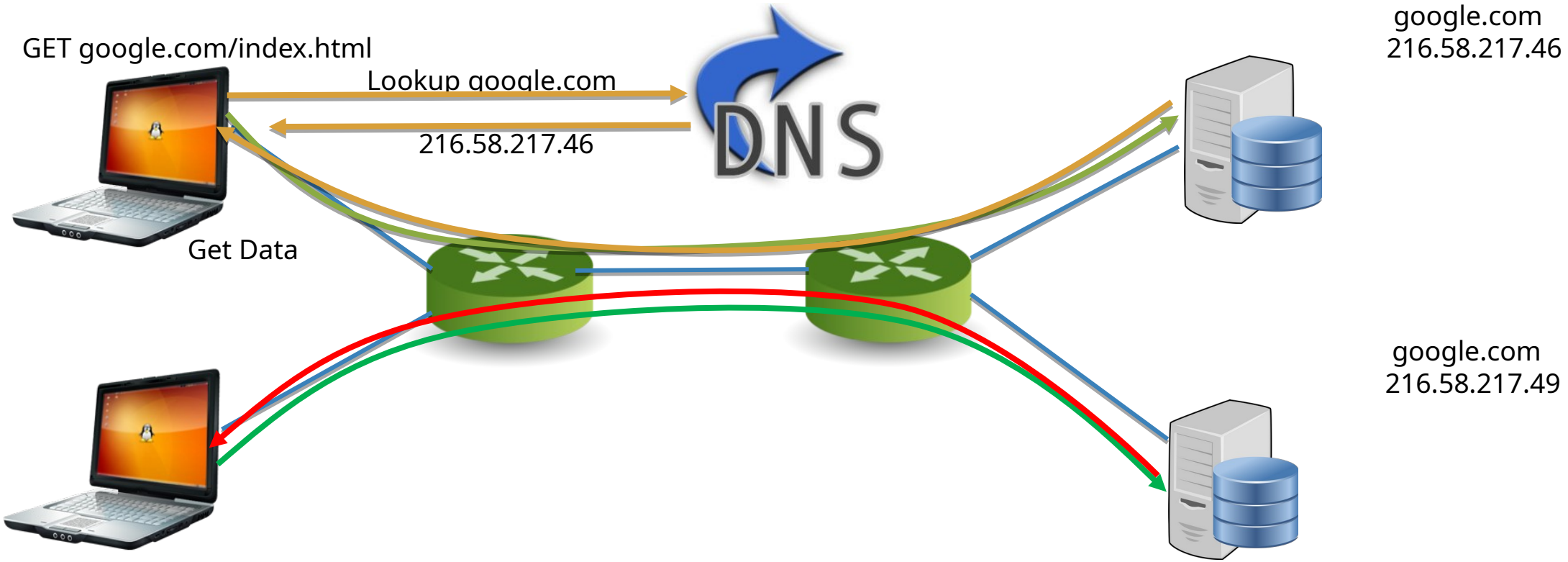
IP Based Communication

youtube.com/catvideo1





IP Based Communication



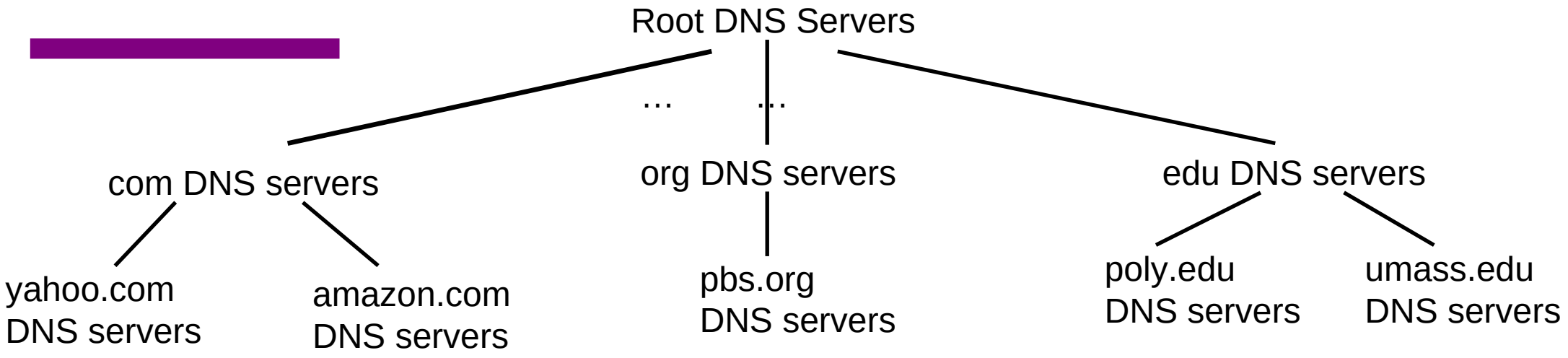
DNS – IP to Name

People: Good with names
Machines: Good with numbers

Ask a person to remember 100s of Ips
- May not work well

DNS maps IP addresses to human readable names.

DNS: a distributed, hierarchical database

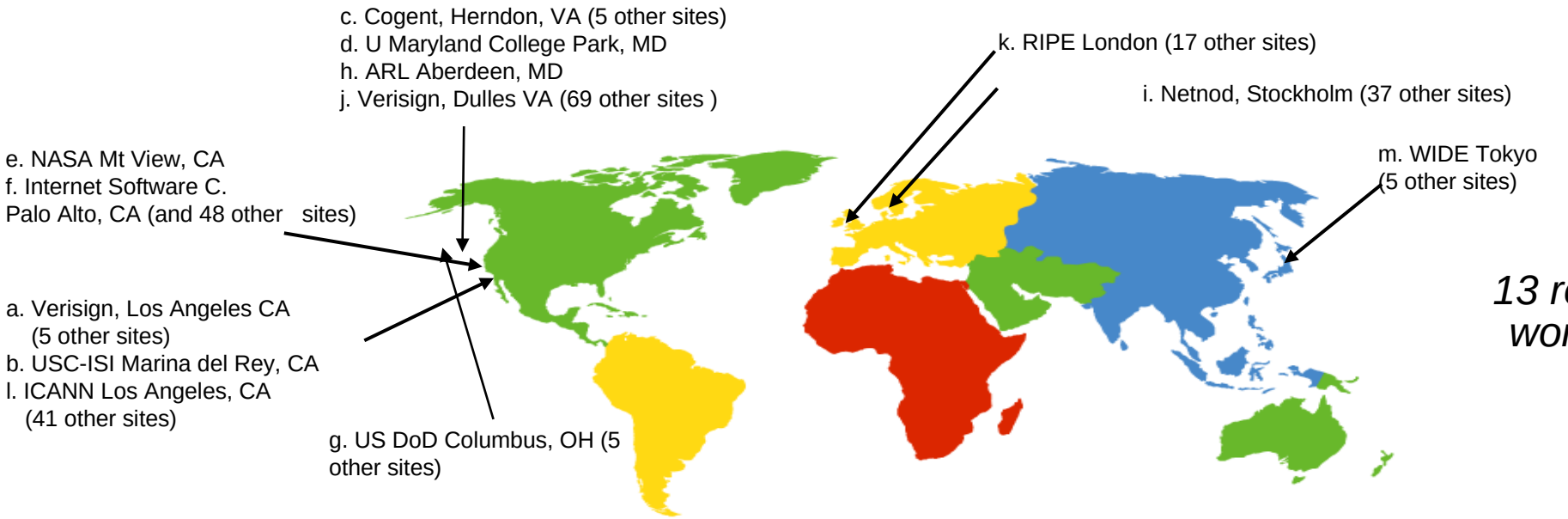


client wants IP for www.amazon.com;

1) client queries root server to find com DNS server
2) client queries .com DNS server to get amazon.com DNS server
3) client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server



13 root name "servers" worldwide

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS name server

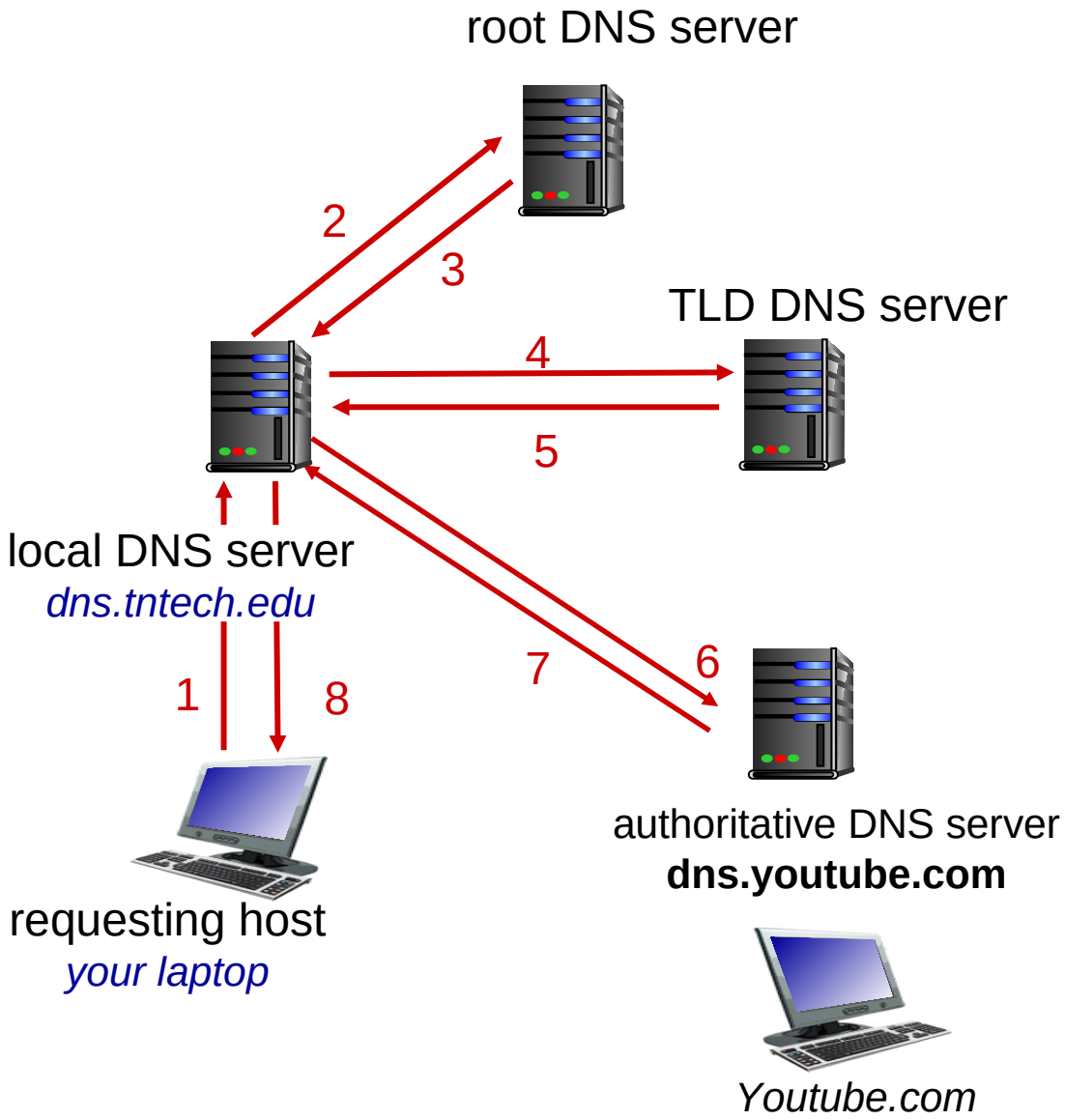
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
  - Served from cache
  - Looked up
  - Attack?

DNS name resolution example - Iterative

- host at tntech.edu wants IP address for youtube.com

iterated query:

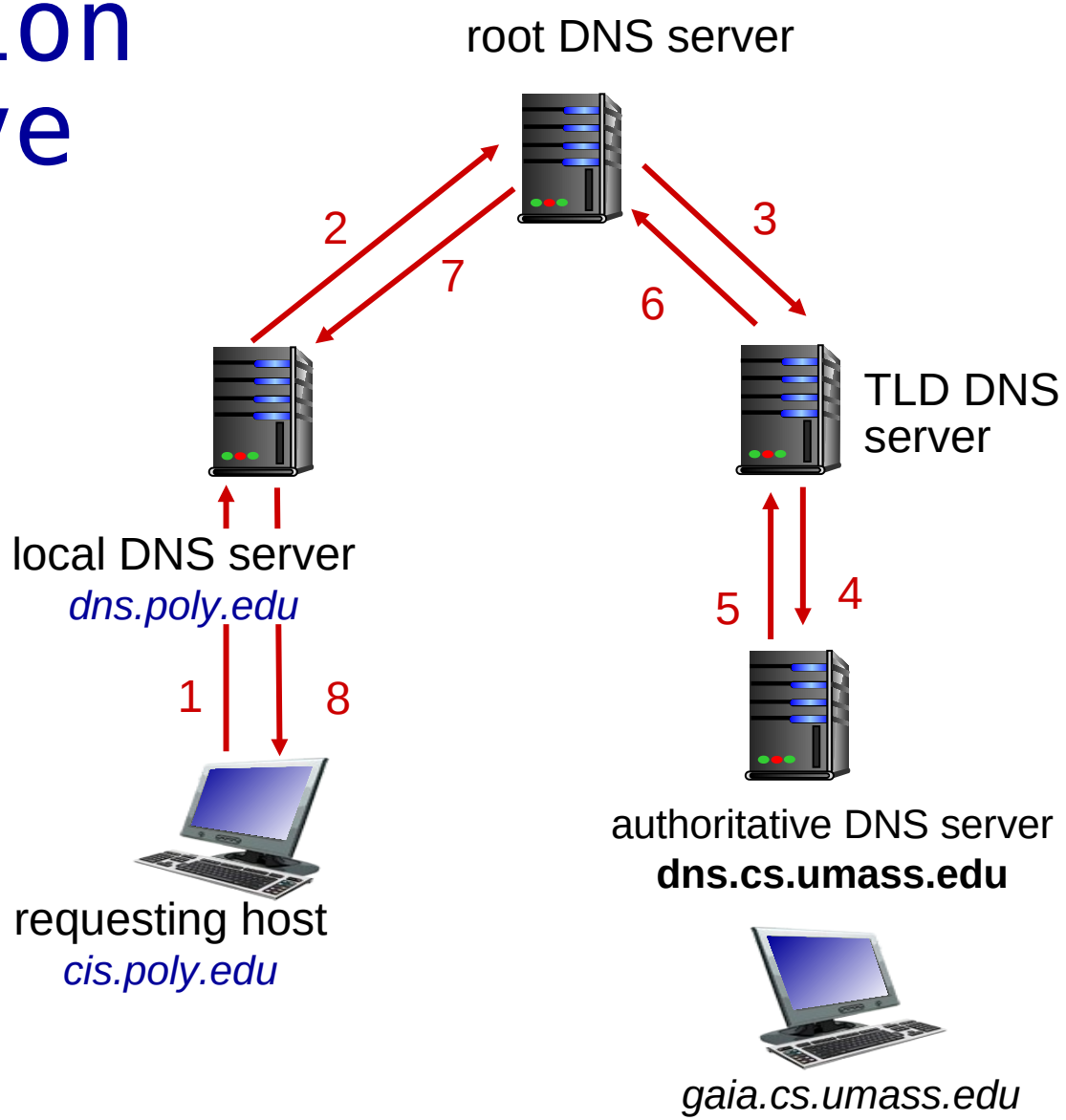
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example- Recursive

recursive query:

- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?

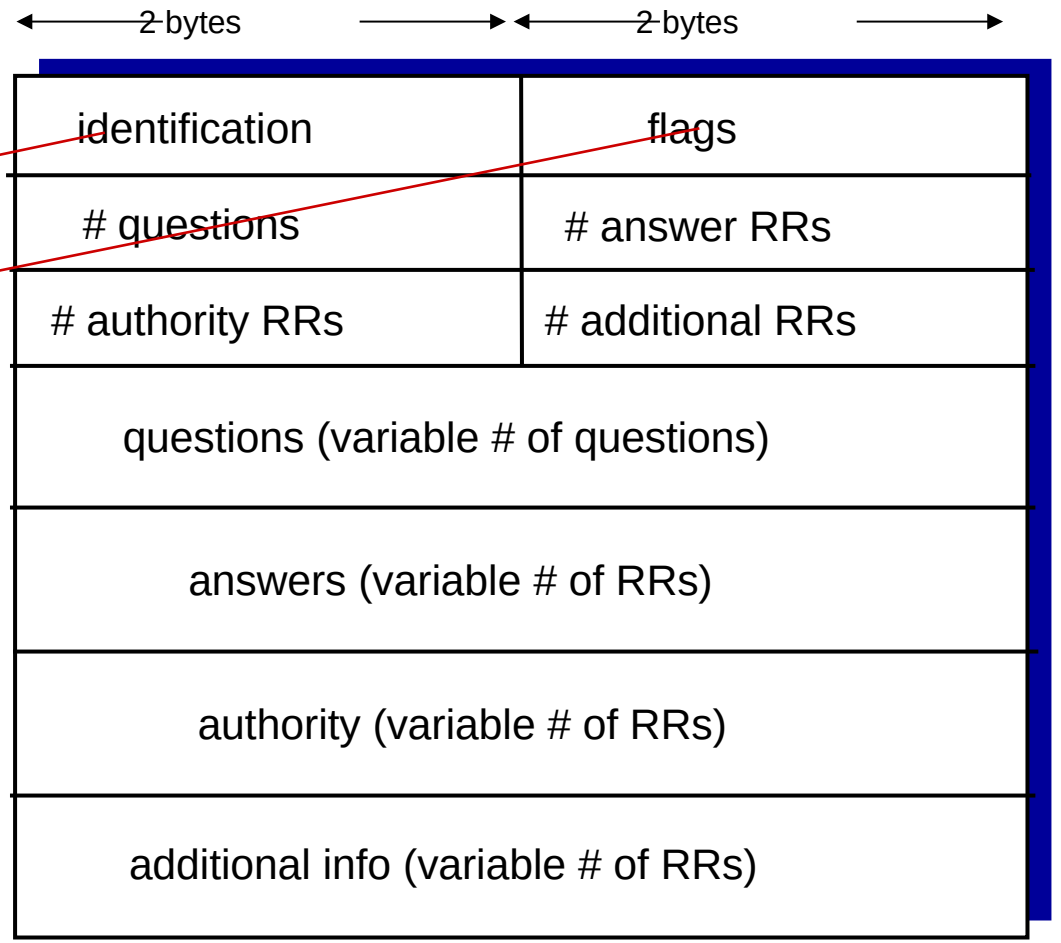


DNS protocol, messages

- *query* and *reply* messages, both with same *message format*

msg header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



Inserting records into DNS

example: new startup “tornadogurard”

- register name tornadoguard.com at *DNS registrar* (godaddy, gandi.net)
  - Tell them the IP of your local DNS server and name
  - registrar inserts two RRs into .com TLD server

Attacking DNS

DDoS attacks

- Bombard root servers with traffic
  - Not successful to date
  - Traffic Filtering
  - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - Potentially more dangerous

Redirect attacks

- Man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus relies to DNS server, which caches

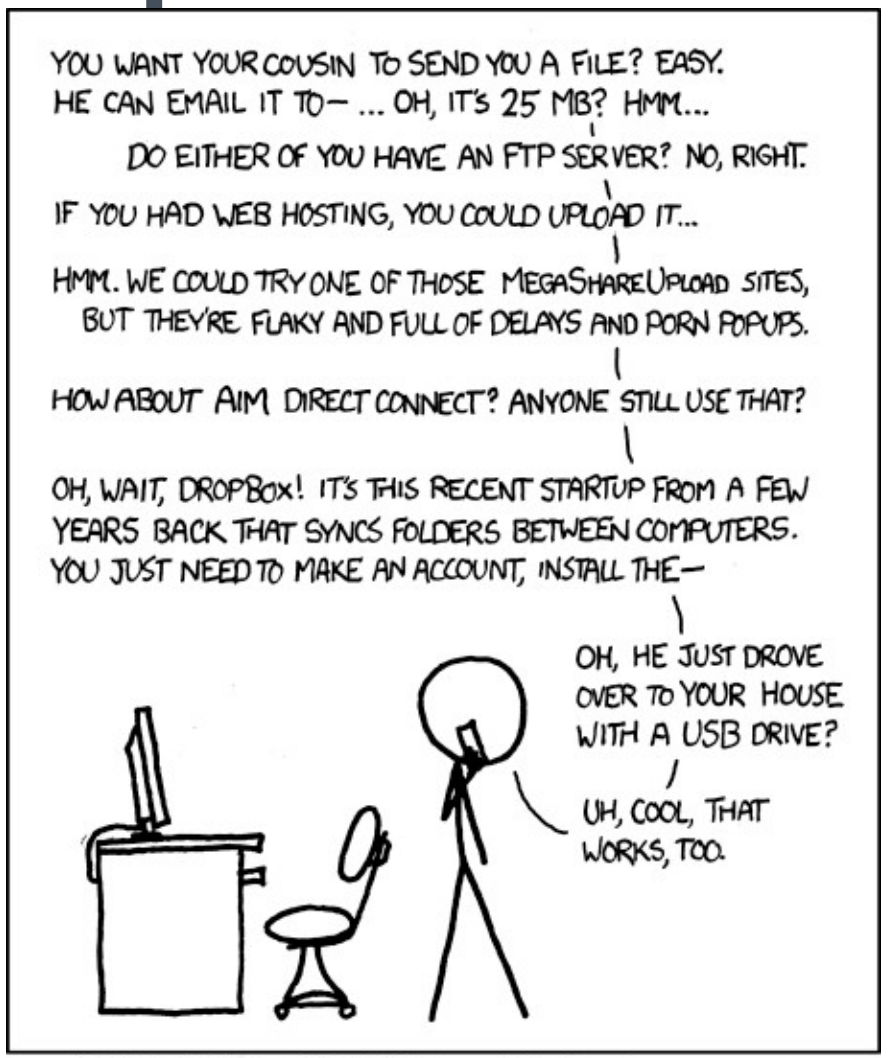
Exploit DNS for DDoS

- Send queries with spoofed source address: target IP
- Requires amplification

Applications – HTTP and P2P

How do you send the cat picture?

- Write your own cat picture transfer app
- In an email
- Upload to a webserver and download using FTP
- Upload to dropbox/AWS/Google cloud
- Use a bit-torrent like protocol
- Use a CDN
- And many other ways….



I LIKE HOW WE'VE HAD THE INTERNET FOR DECADES, YET "SENDING FILES" IS SOMETHING EARLY ADOPTERS ARE STILL FIGURING OUT HOW TO DO.

https://xkcd.com/949/

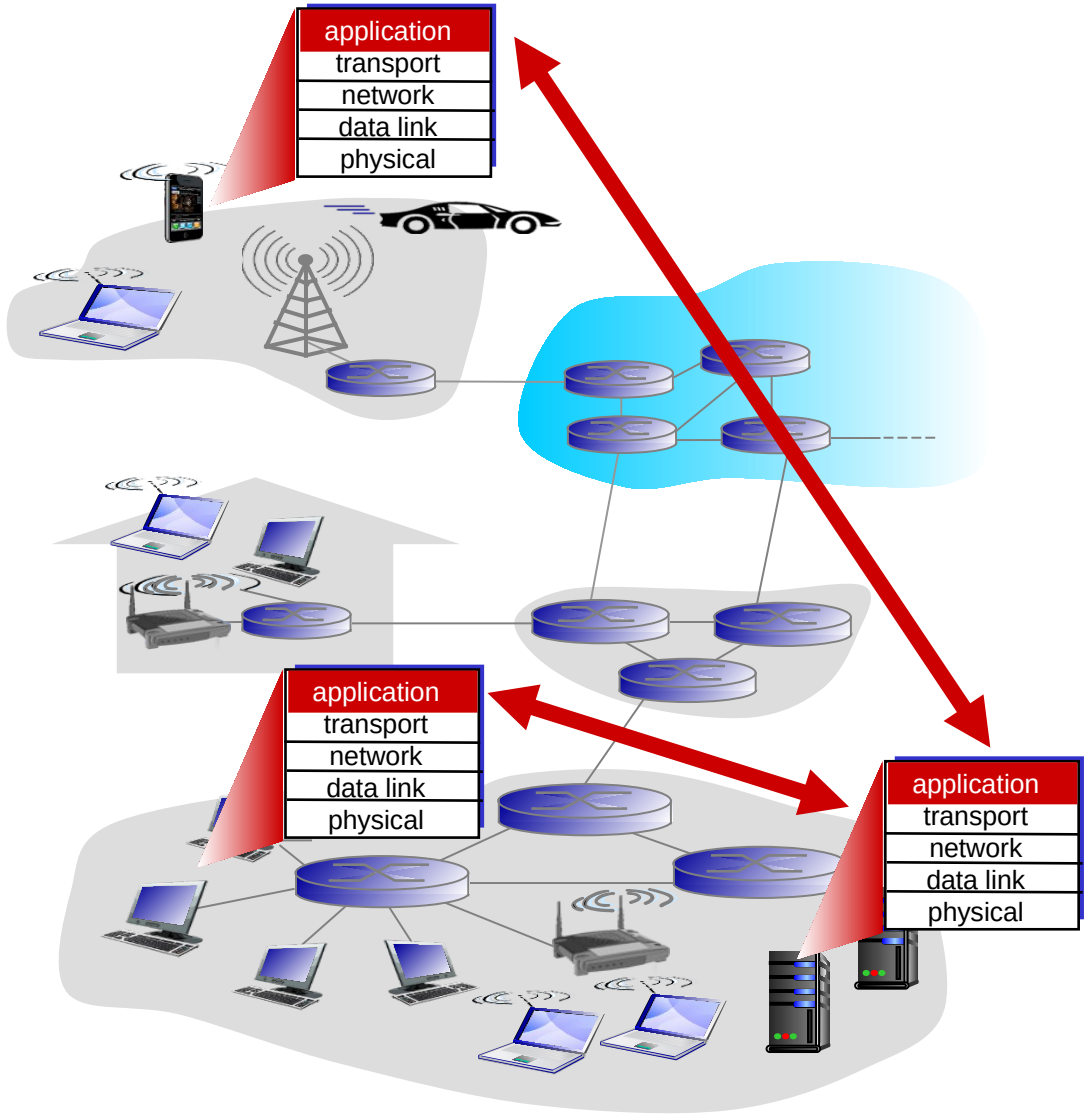
Creating a network app

write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



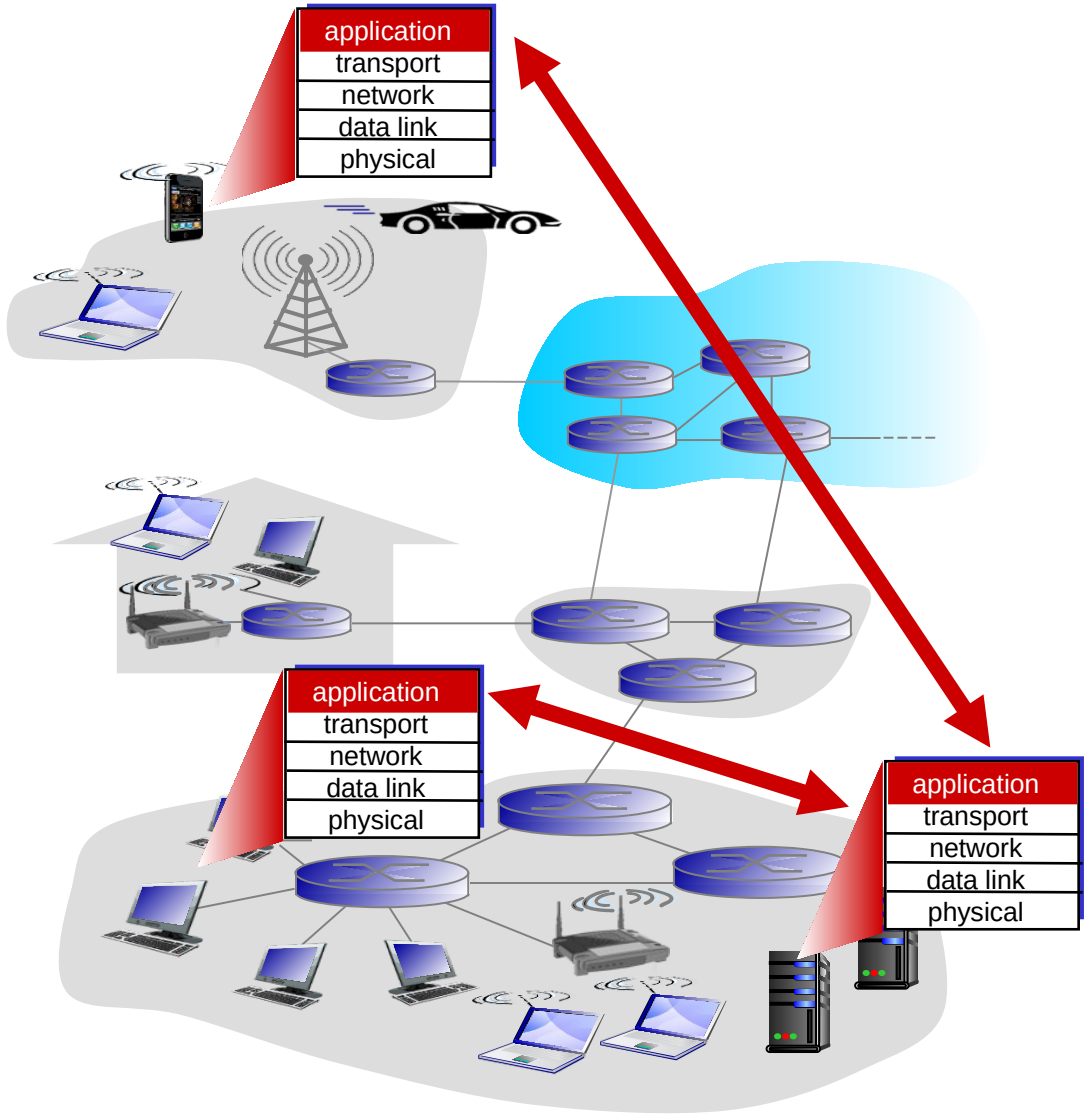
Creating a network app

write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation

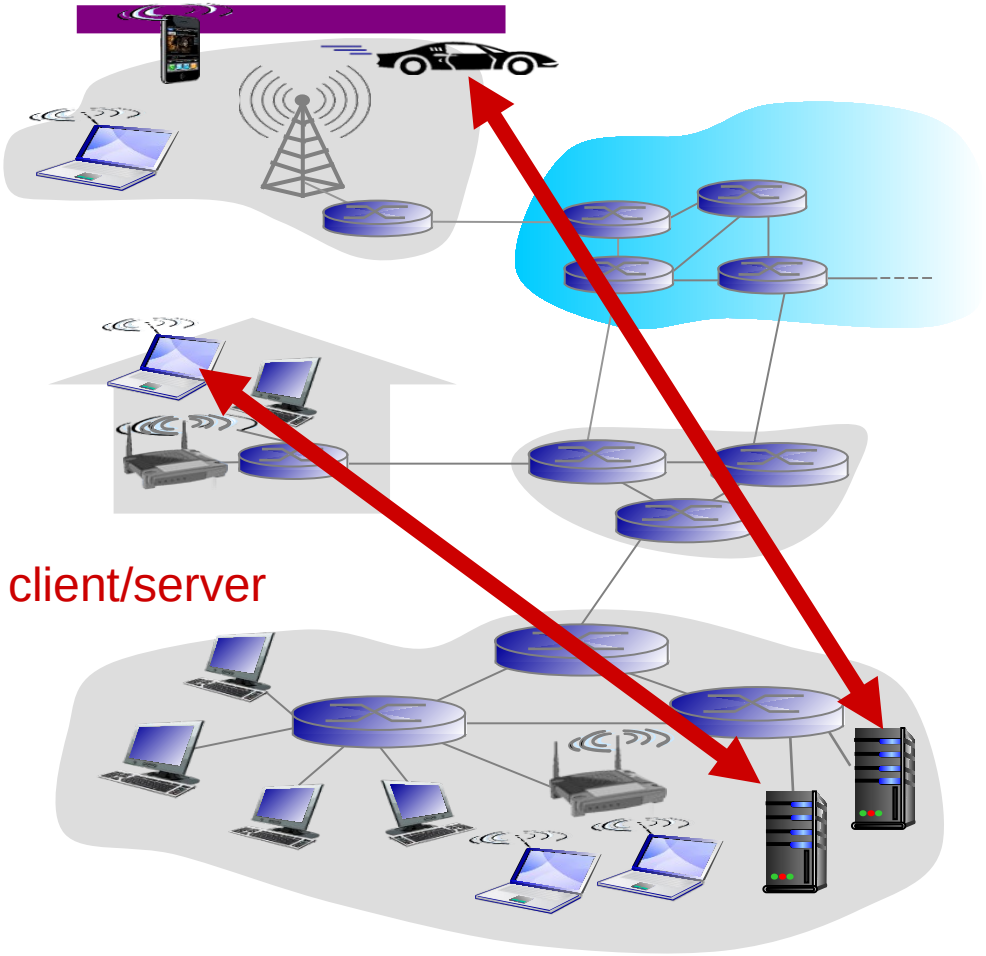


Application architectures

possible structure of applications:

- client-server
- peer-to-peer (P2P)

Client-server architecture



server:

- always-on host
- permanent IP address
- data centers for scaling

clients:

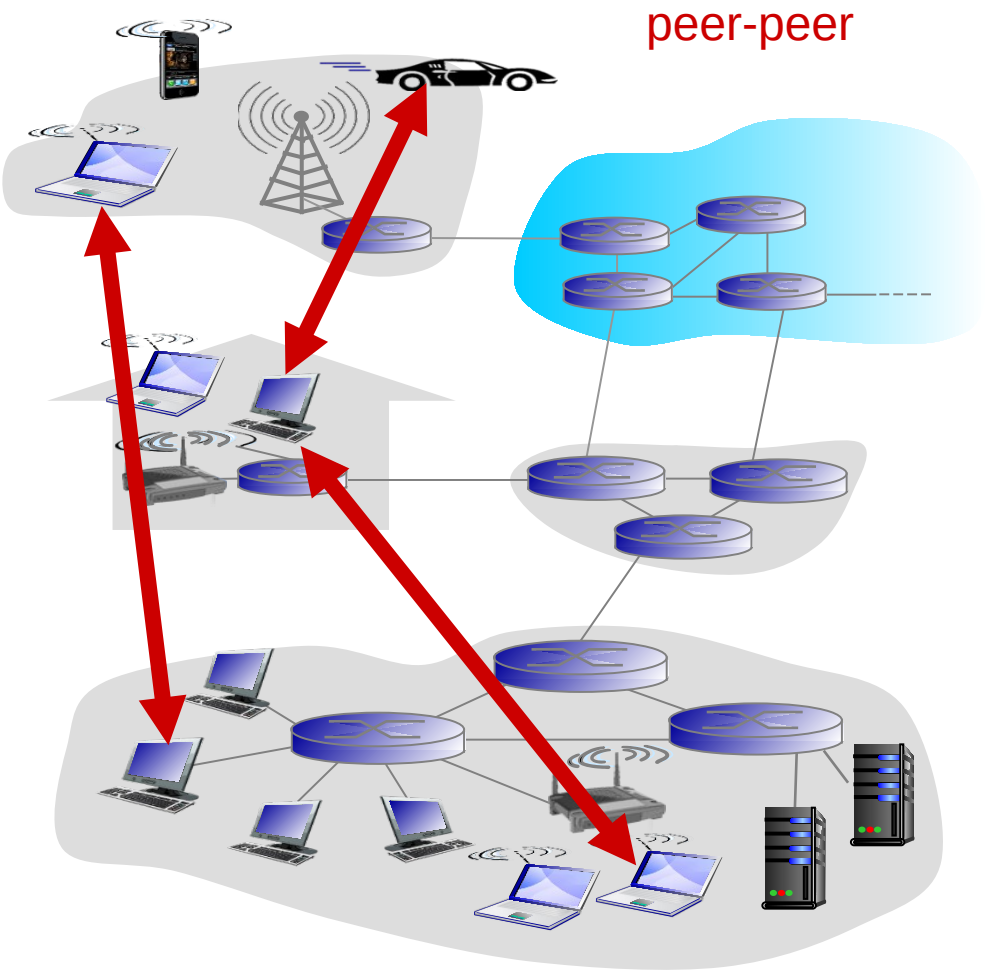
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate

- Services between peers
  - *self scalability*

- peers are intermittently connected and change IP addresses
  - complex management



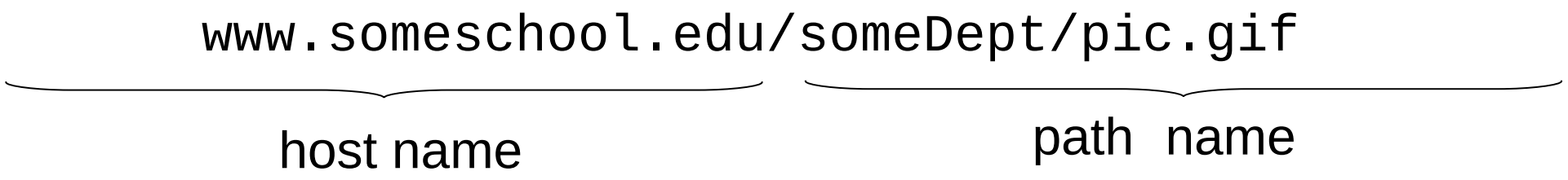
Example of each?

Client server ?

P2P?

Web and HTTP

- *web page* consists of *objects*
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects*
- each object is addressable by a *URL,* e.g.,



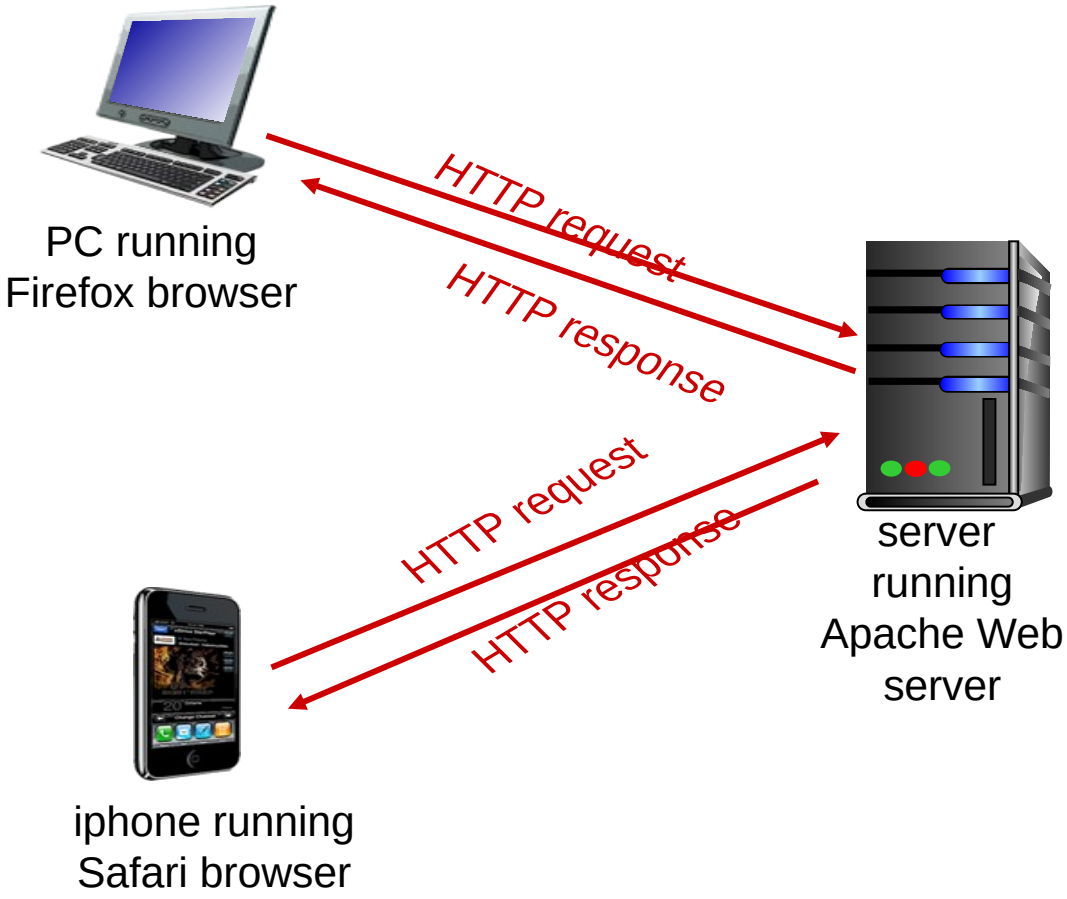
Web vs Internet?

http://info.cern.ch/
http://info.cern.ch/hypertext/WWW/TheProject.html

HTTP overview

HTTP - hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests



HTTP overview (continued)

uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

- server maintains no information about past client requests
- Applications may make it almost "stateful"

HTTP connections (Remember it uses TCP)

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

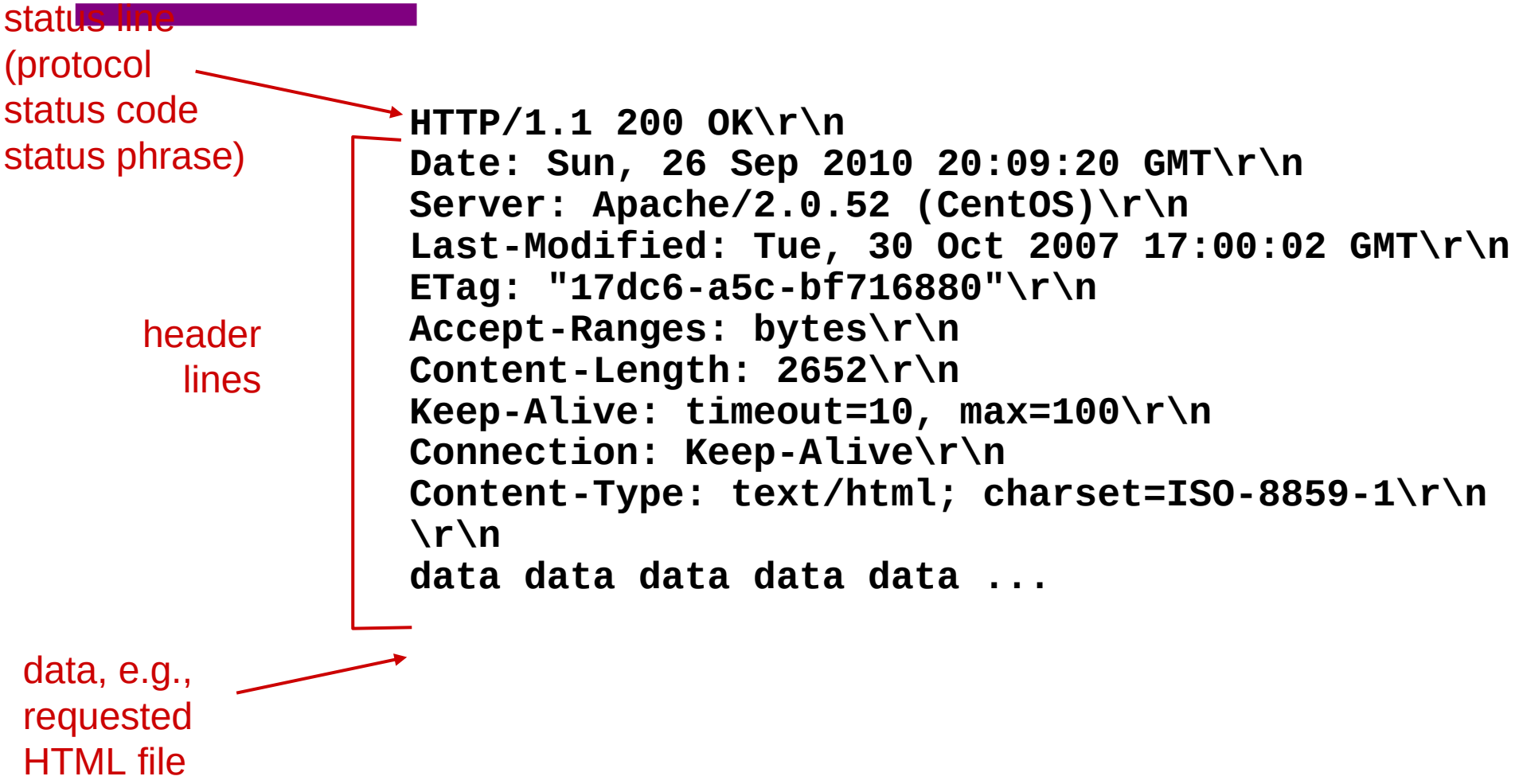
persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

HTTP response message



HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK
  - request succeeded, requested object later in this msg

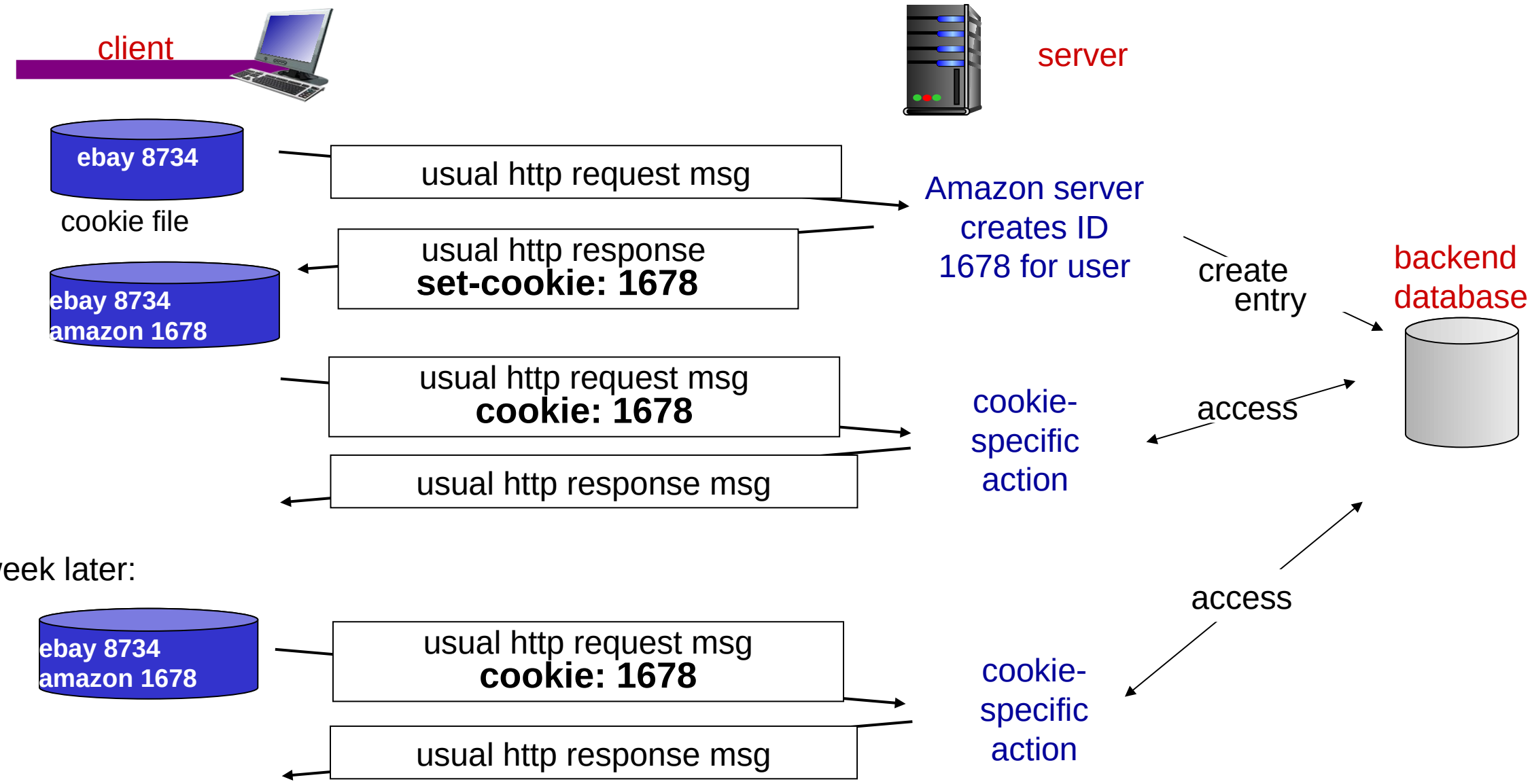
301 Moved Permanently
  - requested object moved, new location specified later in this msg (Location:)

400 Bad Request
  - request msg not understood by server

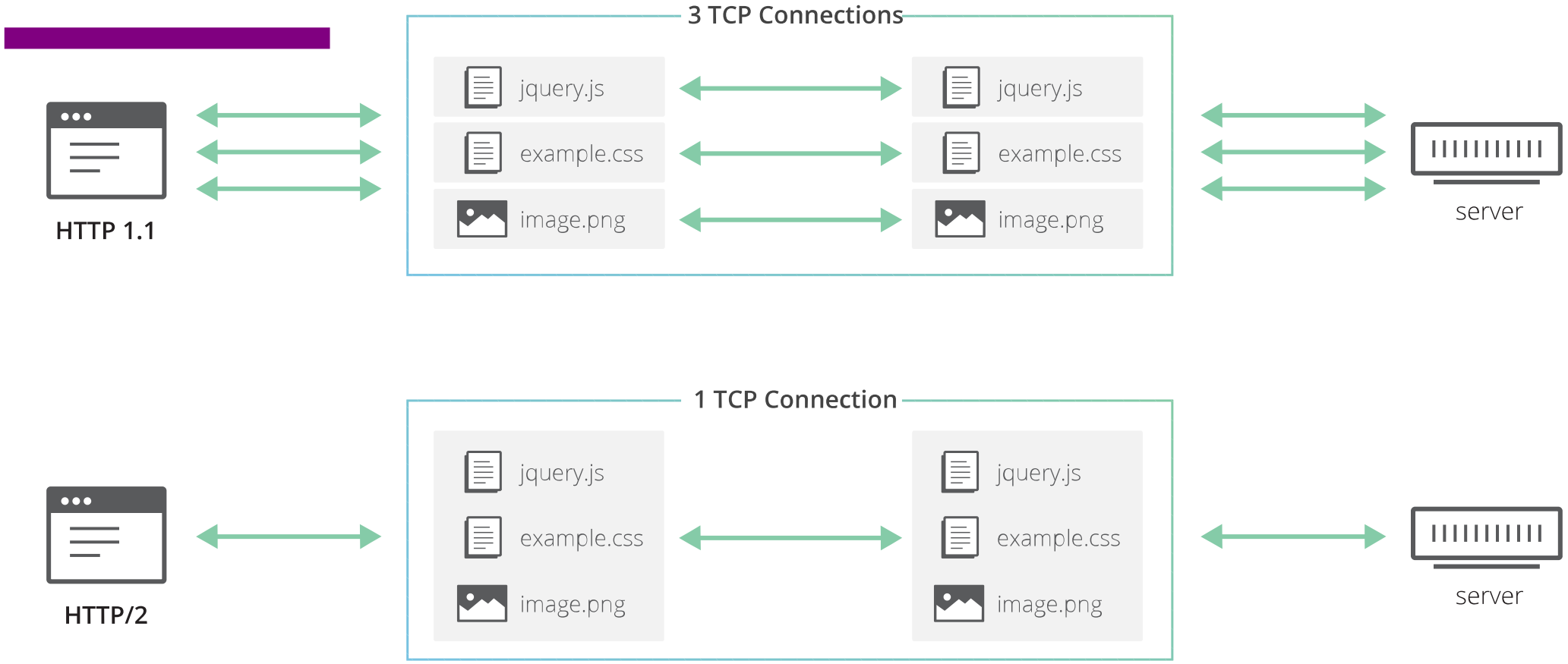
404 Not Found
  - requested document not found on this server

505 HTTP Version Not Supported

Cookies: keeping “state”

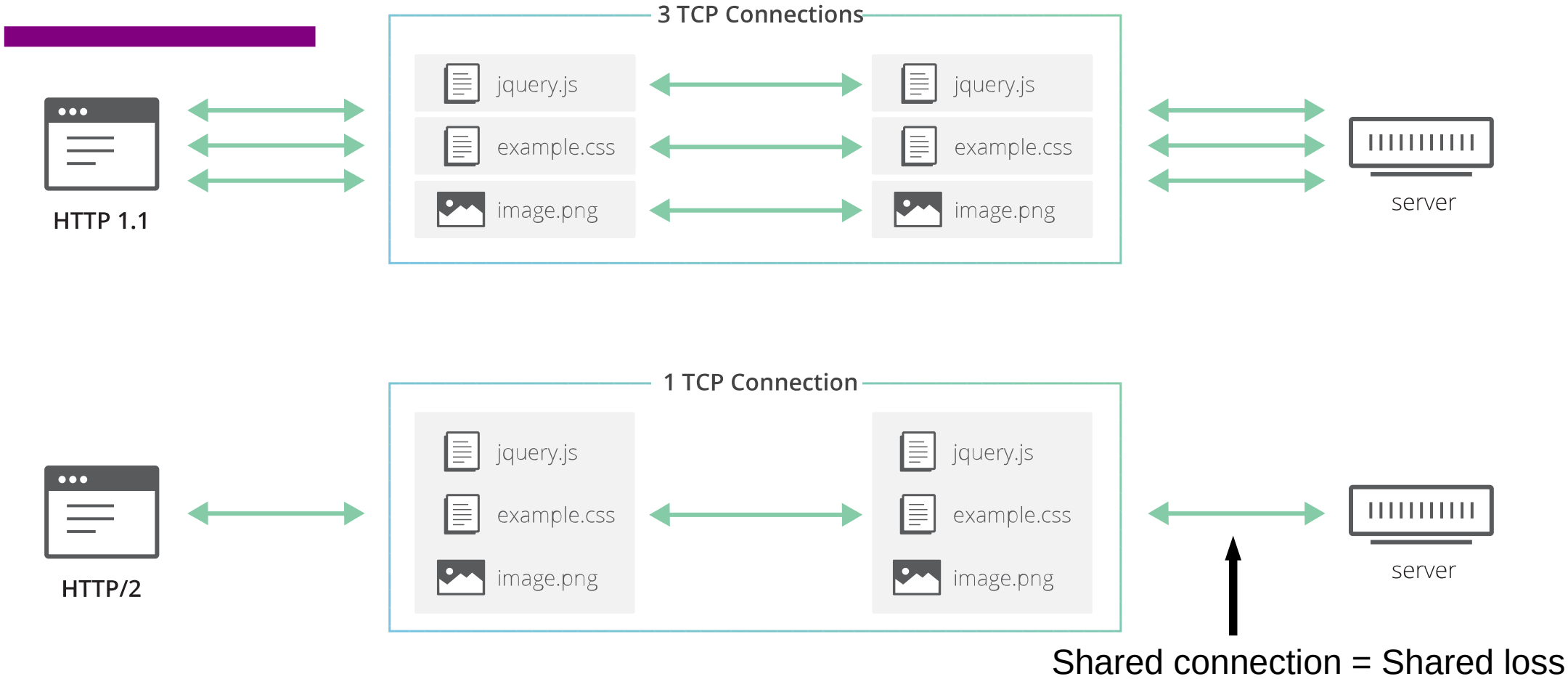


HTTP 1 vs 2



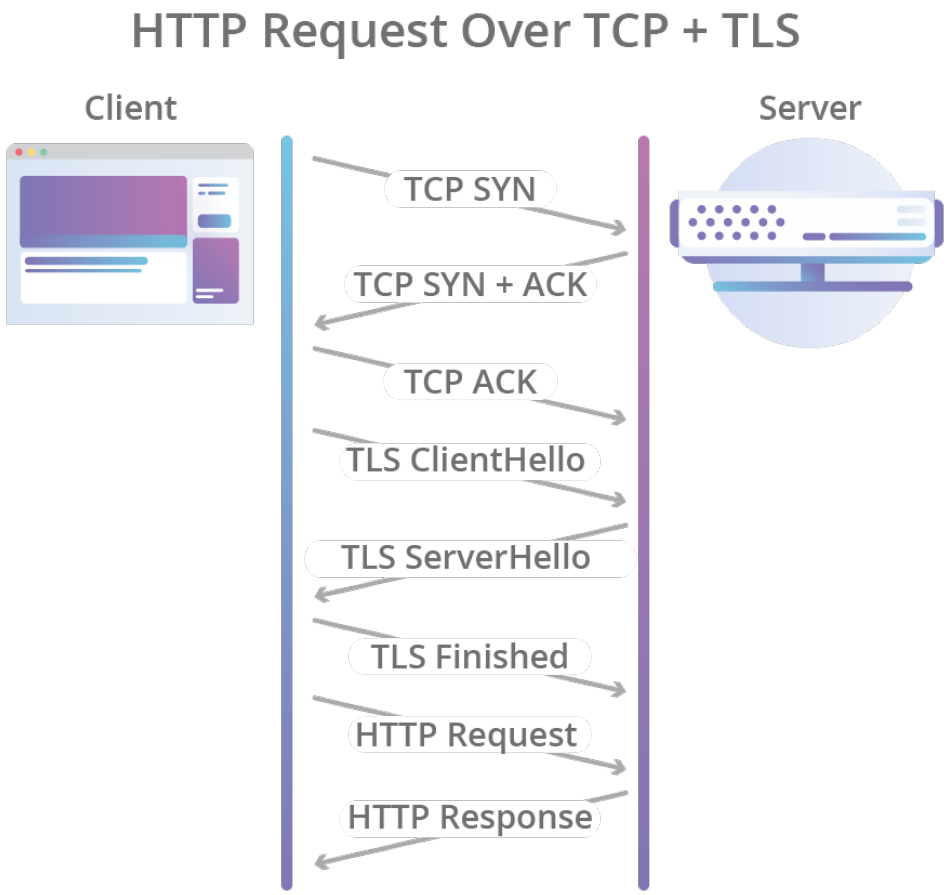
https://blog.cloudflare.com/the-road-to-quic/

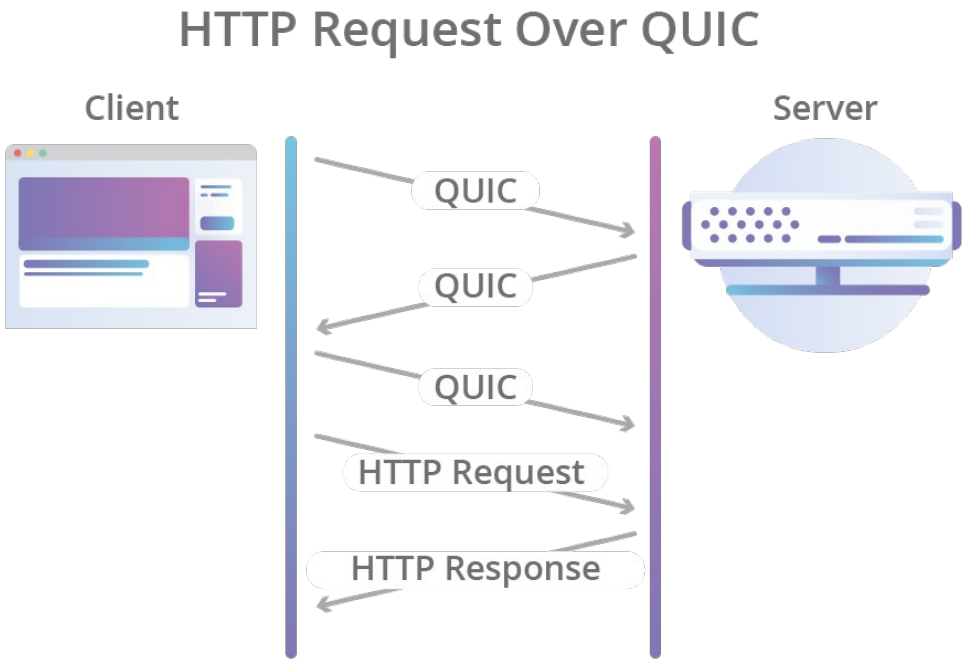
HTTP 2 Head-of-the-line Blocking



https://blog.cloudflare.com/the-road-to-quic/

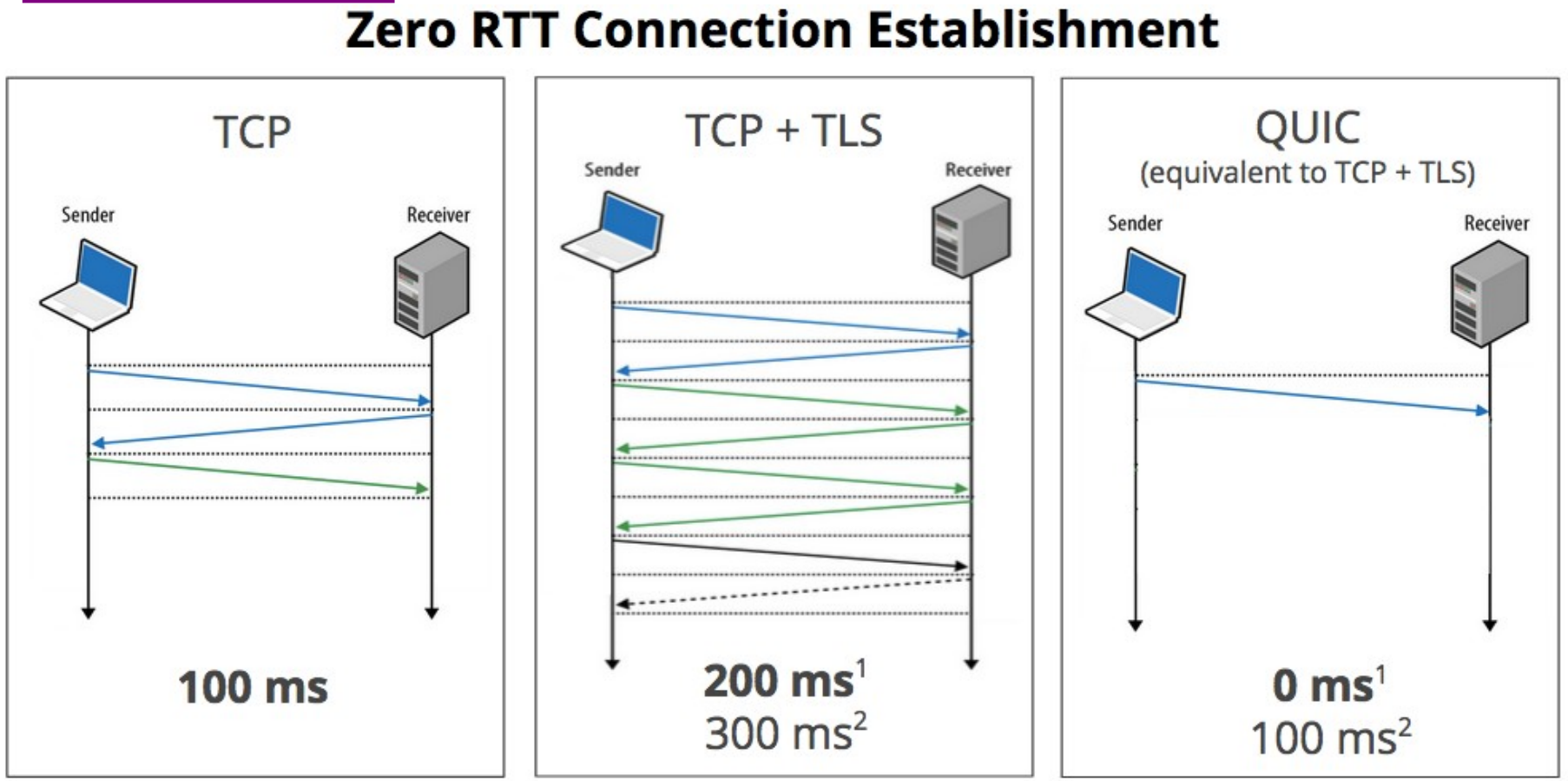
QUIC





https://blog.cloudflare.com/the-road-to-quic/

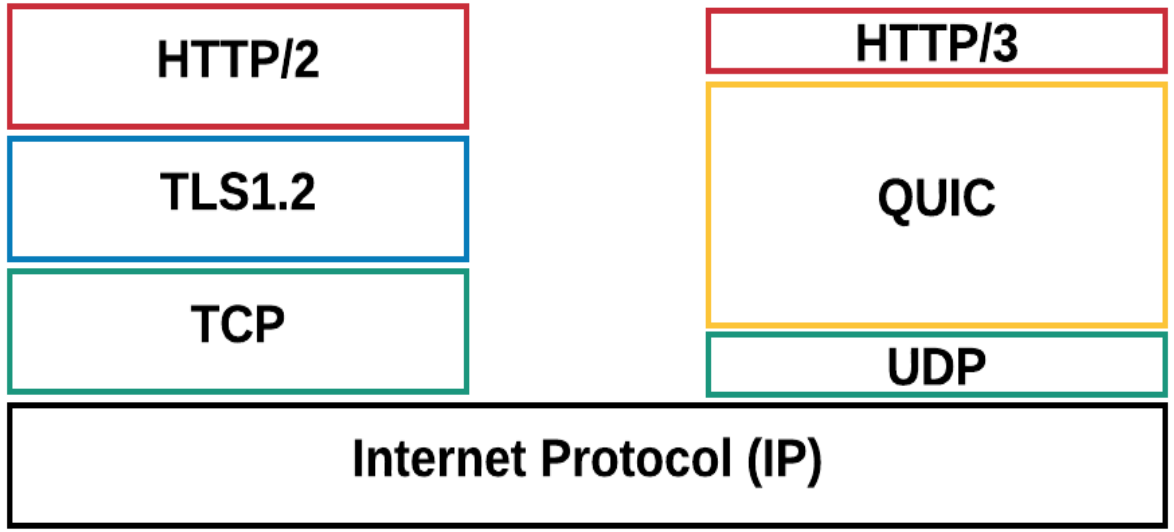
QUIC is Quick(er)



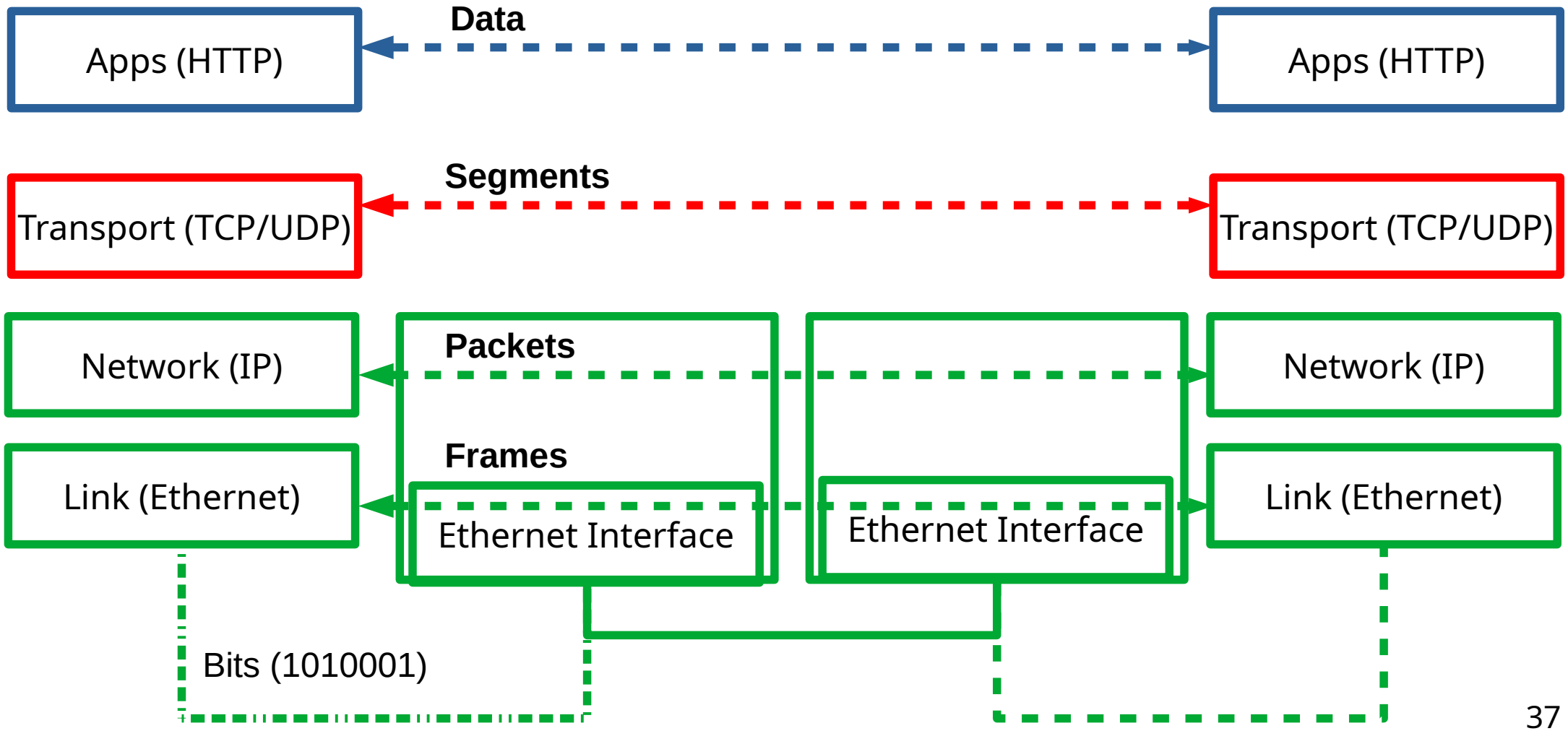
https://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html

HTTP 2/TCP vs HTTP 3/QUIC

1. Faster connection establishment
2. No HoL blocking
3. Multiplexing connections with ability to differentiate
4. Connection migration



https://eng.uber.com/employing-quic-protocol/


Data
Apps (HTTP)
Apps (HTTP)
Segments
Transport (TCP/UDP)
Transport (TCP/UDP)
Packets
Network (IP)
Network (IP)
Frames
Link (Ethernet)
Ethernet Interface
Ethernet Interface
Link (Ethernet)
Bits (1010001)