CS 118 Discussion Week 2:

The Application Layer

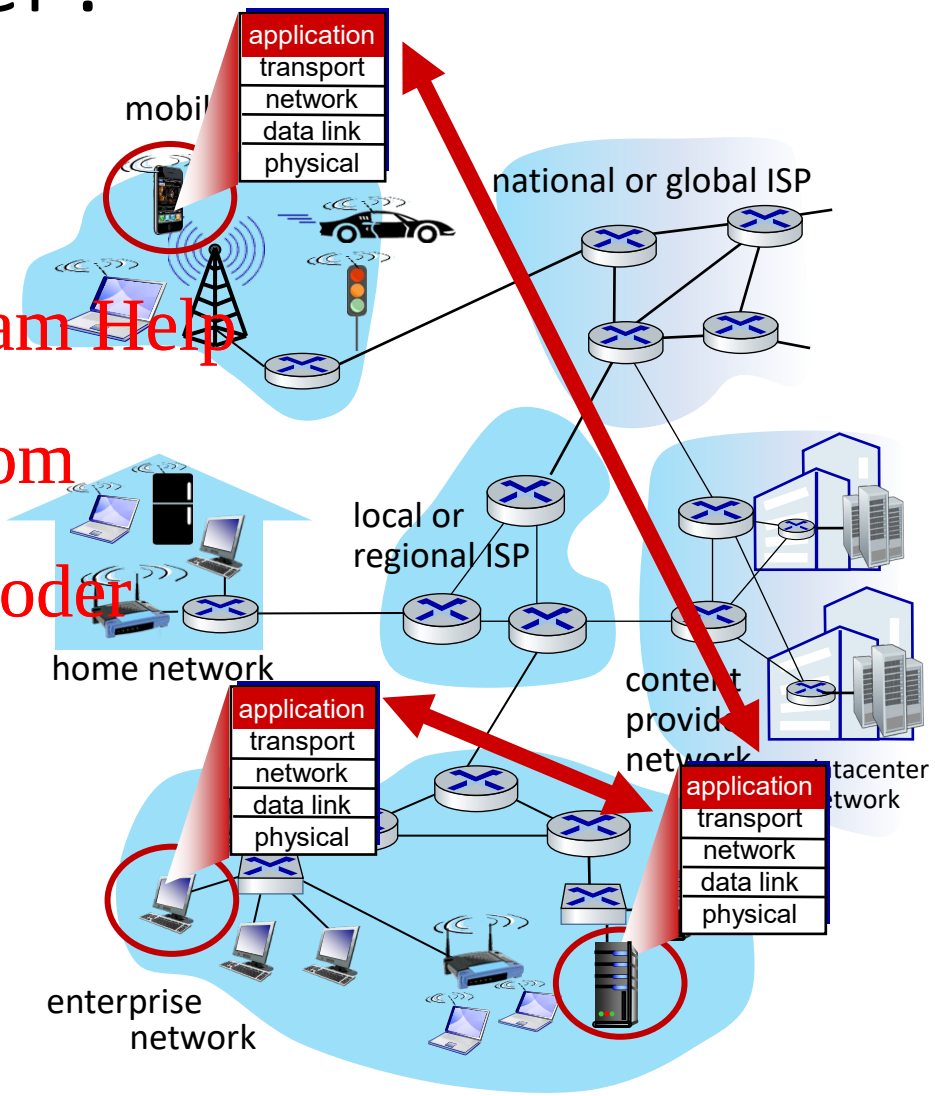
Winter 2021

Overview

- Brief introduction to the Application Layer
- Protocols that live on the Application Layer
- Most Important take-aways:
  - 1: Paradigms
  - 2: Protocols
  - 3: Particular uses *

What is the Application Layer?

- "An application layer is an abstraction layer that specifies the shared protocols and interface methods used by hosts in a communications network. "
- Where essentially everything people usually care about lives.



What lives on the Application Layer?

- The Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- Video streaming
- Content Distribution Networks

What core concepts should you know about it?

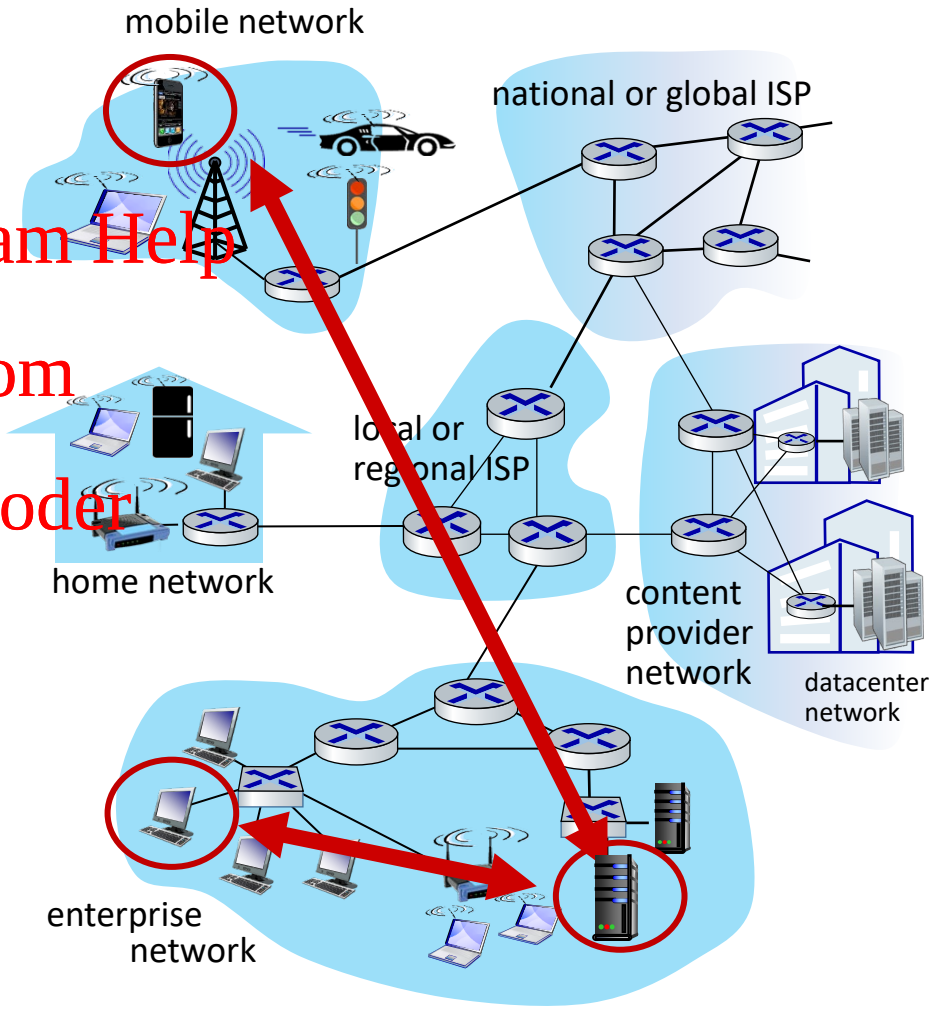
- 1: What paradigms/models get used on the application layer?
  - E.g. transport-layer service models (note: this lives below application layer), client-server paradigm, peer-to-peer paradigm
- 2: What protocols (within those models) are popular and well-used?
  - SMTP, IMAP, DNS, HTTP, etc.
- 3: How can you, personally, interface with it (project 1)?
  - Programming with sockets

1: What paradigms/models get used on the application layer?

- As we know, certain paradigms or model tend to predominate for a given set of tasks.
- Sending information over connections not fundamentally different.
- Most important differences between the two main types (Client-Server vs Peer-Peer) is who holds the data being transferred.

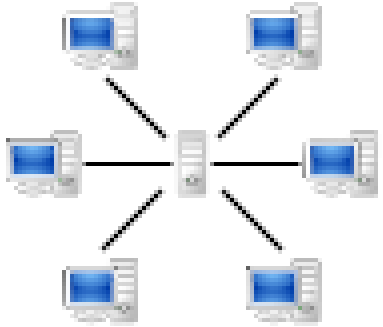
Client-Server Model -- Servers

- Server acts as an always-on host
- Thing of e.g. Google
- Server must have permanent IP address (so you know where to find them)
- Server, in general, must be quite powerful, to facilitate scaling
- Server usually housed in data center



Client-Server -- Clients

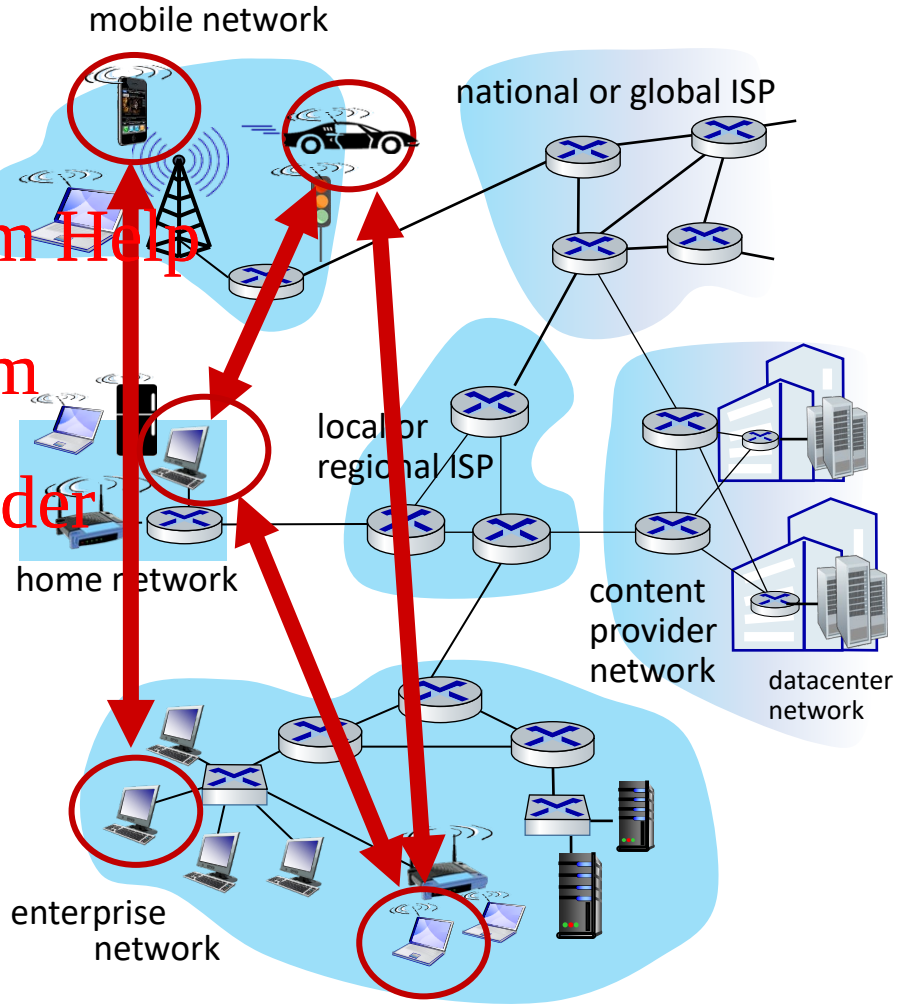
- Example Client: your phone!
- Contact and communicate with server
- May be intermittently connected
- May not have permanent IP address
- Does not communicate with other clients



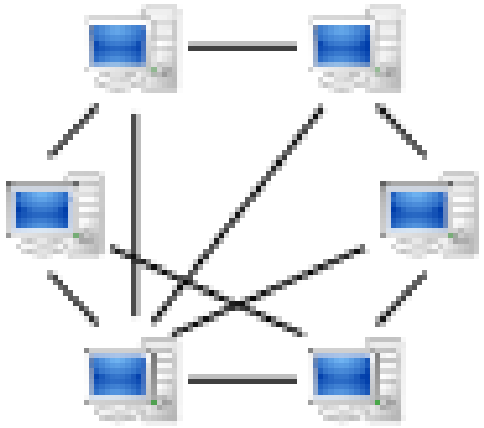
- Examples of Client-Server Protocols:
  - HTTP
  - IMAP

Peer-Peer Model

- No central server that's always on
- Arbitrary nodes communicate directly with one another
- peers request service from other peers, provide service in return to other peers
  - Scales very easily – new peers bring new service capabilities and demands

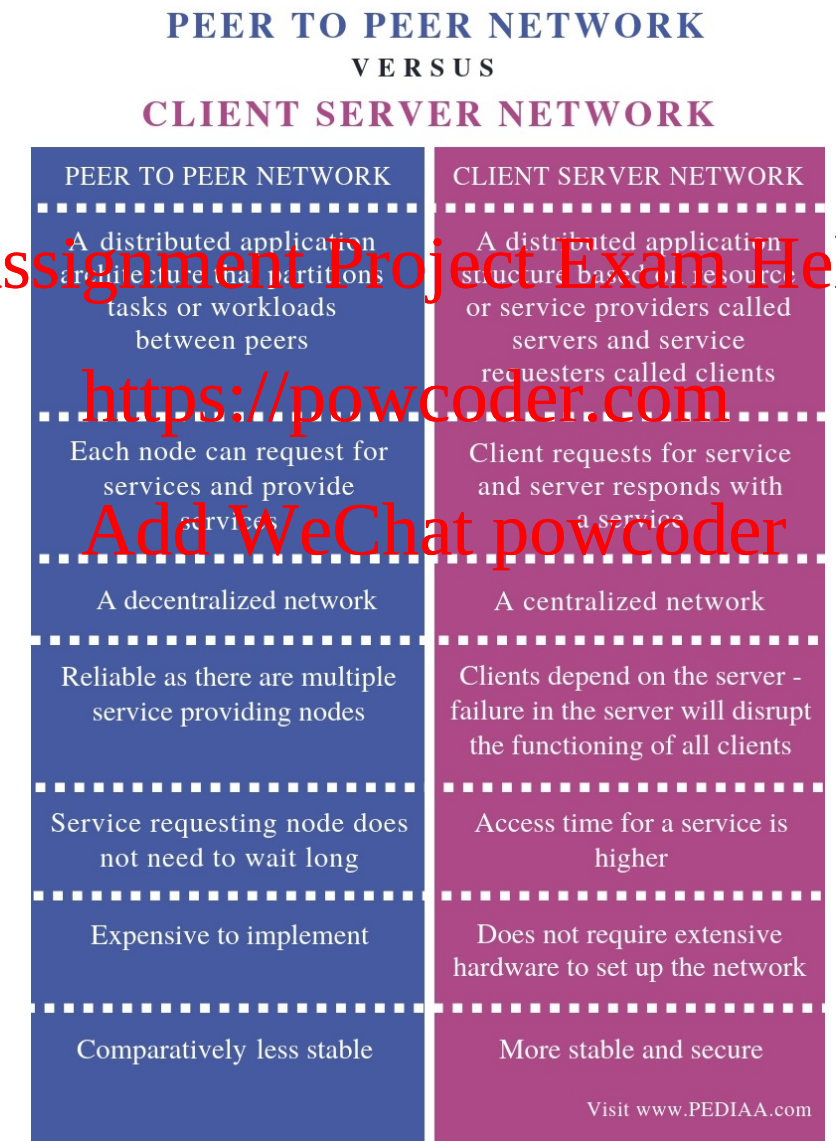


Peer-Peer II



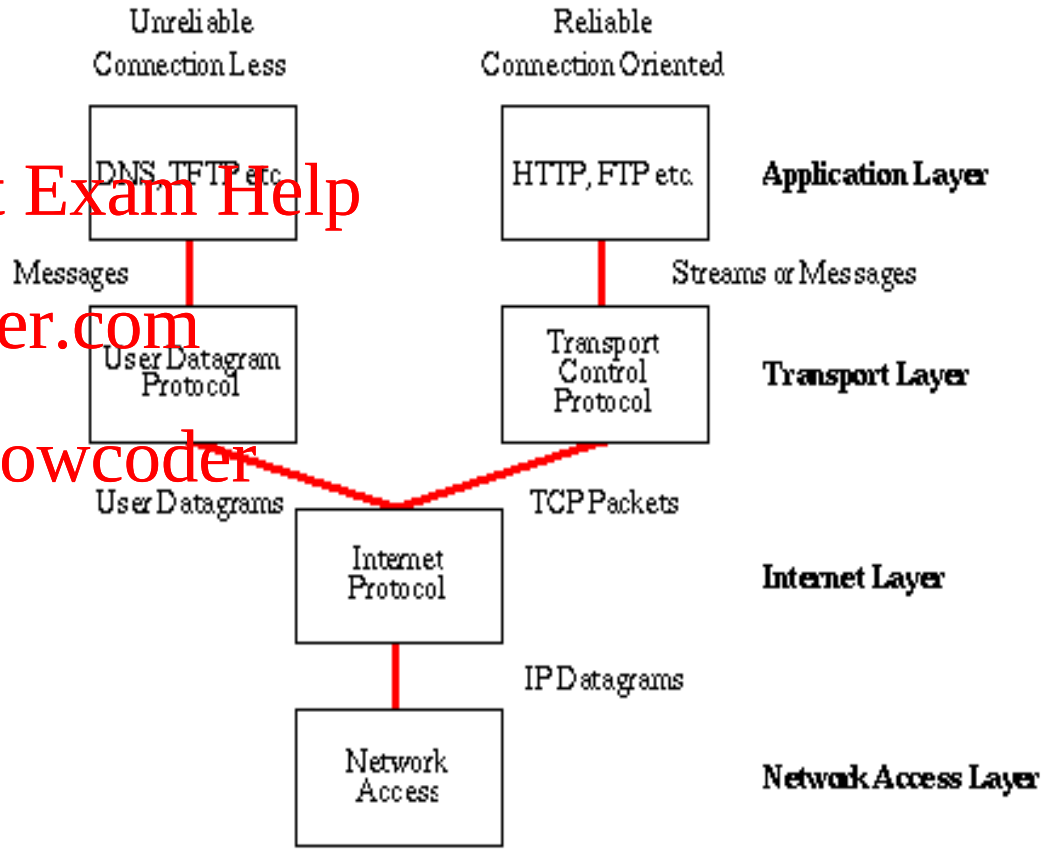
- Peers do *not* need to be always connected, and *may* have a dynamic IP address.
  - This can make management difficult/non-trivial
- Example of Peer-Peer system: Bittorrent/P2P file sharing.
- Discussion questions: Overlap between these two systems? Advantages/Disadvantages between them?

Comparisons



What do these services run on ‘top of’?

- Application level services and protocols make certain assumptions:
  - Data Integrity
    - Reliability?
  - Timing
    - Real time?
  - Throughput
    - Streaming requirements
  - Security
    - E.g. encryption



Free to use, World Wide Web consortium

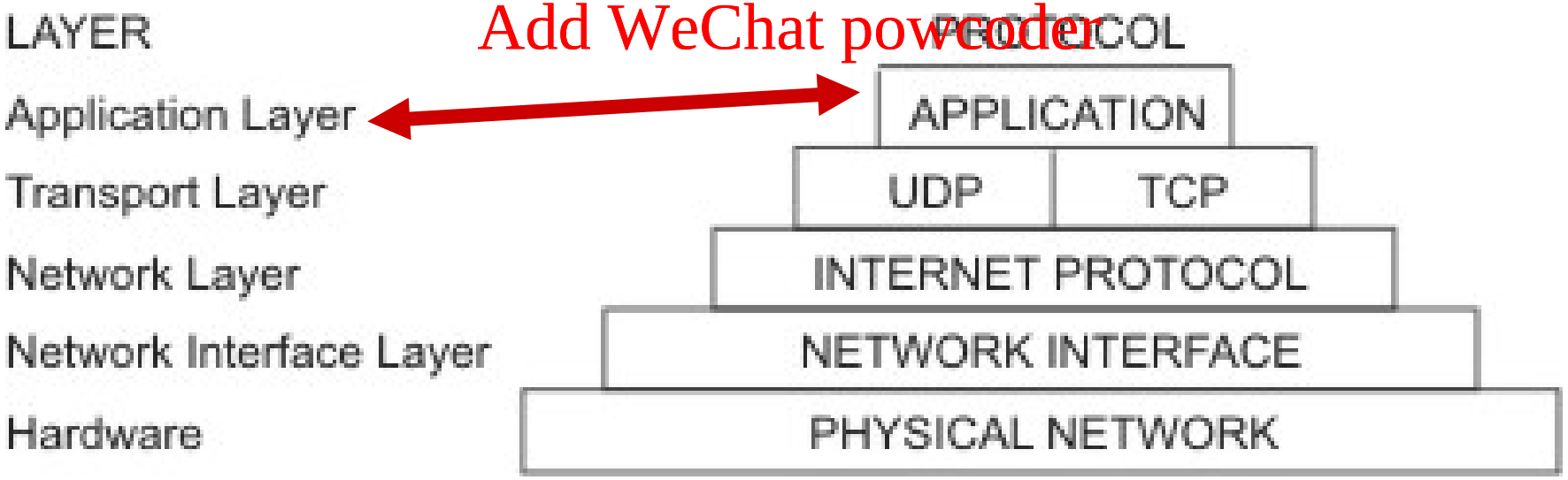
UDP

- The absolute bare-bones
- Provides port-level multiplexing, and absolutely nothing else.
- UDP is unreliable datagram transfer
- Discussion question: Why do UDP?
  - Hint: Think about application protocols

TCP

- Reliable data transfer (accomplished via retransmissions)
- Flow control
- Congestion control
- Provides a reliable *connection*
- However, does not provide: timing, minimum throughput guarantee, security (on its own)
- How do we provide security? TLS,

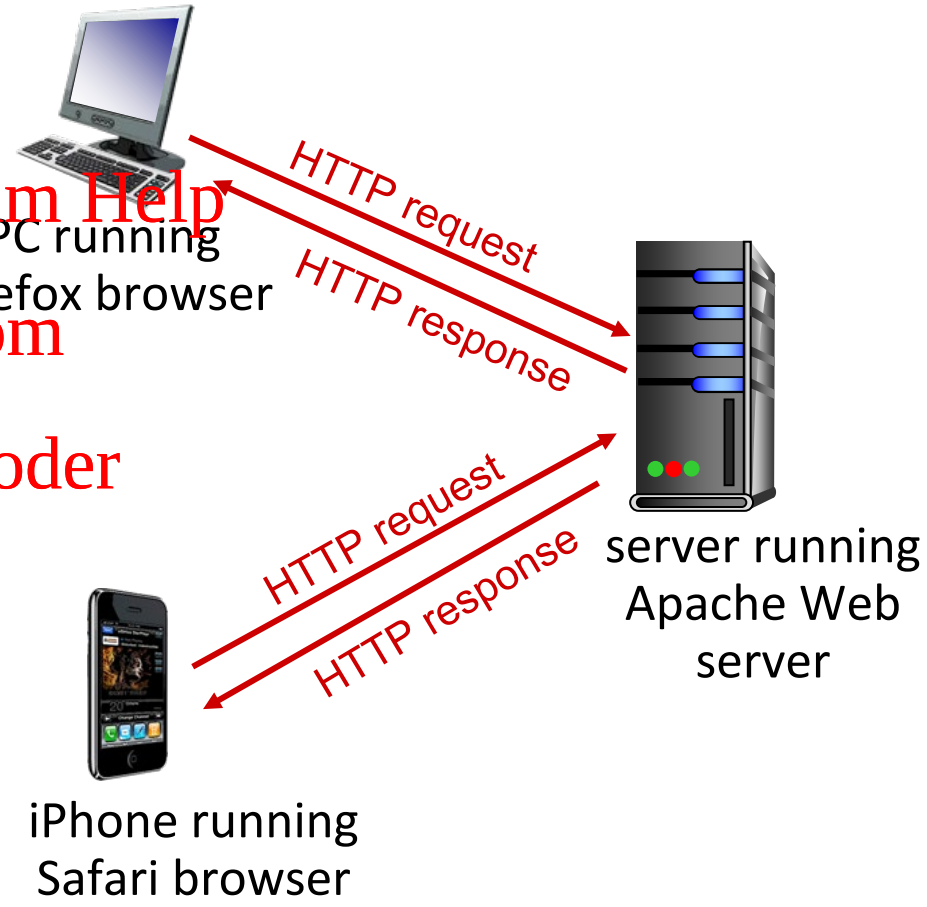
2: What protocols are popular and well-used?



Free to use, IBM

HTTP

- HyperText Transfer Protocol
- What your browser uses (or used to use) to retrieve Web objects
  - Now generally uses secured form, https
- Also what the Server uses to send those objects in response to the request!



HTTP Basics

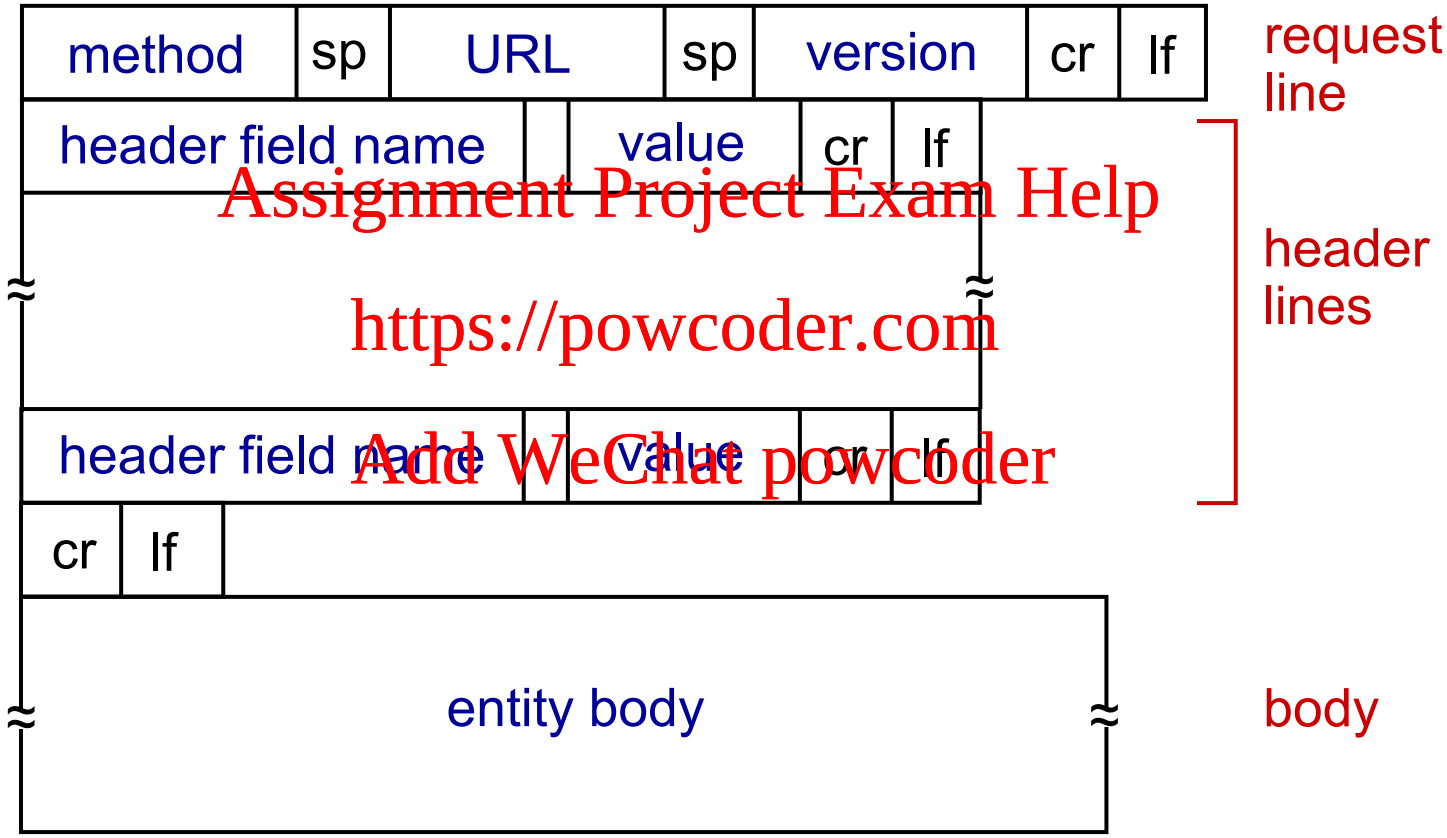
- HTTP uses a *Client-Server* model.
  - Client: Browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - Server: Web server sends (using HTTP protocol) objects in response to requests
- HTTP uses TCP:
  - Client initiates TCP connection (creates socket) to server, port 80
  - Server accepts TCP connection from client
  - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)

Persistent vs Non-Persistent HTTP

Non-Persistent	Persistent
1: TCP connection opened	1: TCP connection opened to a server
2: at most one object sent over TCP connection	2: Multiple objects can be sent over *single* TCP connection between client, and that server
3: TCP connection closed	3: TCP connection closed

For non-persistent connections, downloading multiple objects requires multiple connections.

HTTP Format



HTTP Messages

- Two Types:
- Request
  - POST (send data to server)
  - GET (request data from particular resource)
  - HEAD
  - PUT
- Response
  - **`HTTP/1.1 200 OK`** (protocol, status code, status phrase)

HTTP: New and Improved

- Largely going to gloss over in this discussion section.
- HTTP/2
  - Decreasing delay in multi-object HTTP requests
- HTTP/3
  - Add security, change congestion control, over UDP (read up on QUIC if interested)

SMTP Explanation

- The protocol that (used to be) used for email! (nowadays extended version largely used)
- SMTP is run between the client and server
  - Client: sending mail server
  - Server: receiving mail server
- uses TCP to reliably transfer email message from client (mail server initiating connection) to server, port 25
  - direct transfer: sending server (acting like client) to receiving server
- three phases of transfer
  - SMTP handshaking (greeting)
  - SMTP transfer of messages
  - SMTP closure
- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase

SMTP

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

DNS

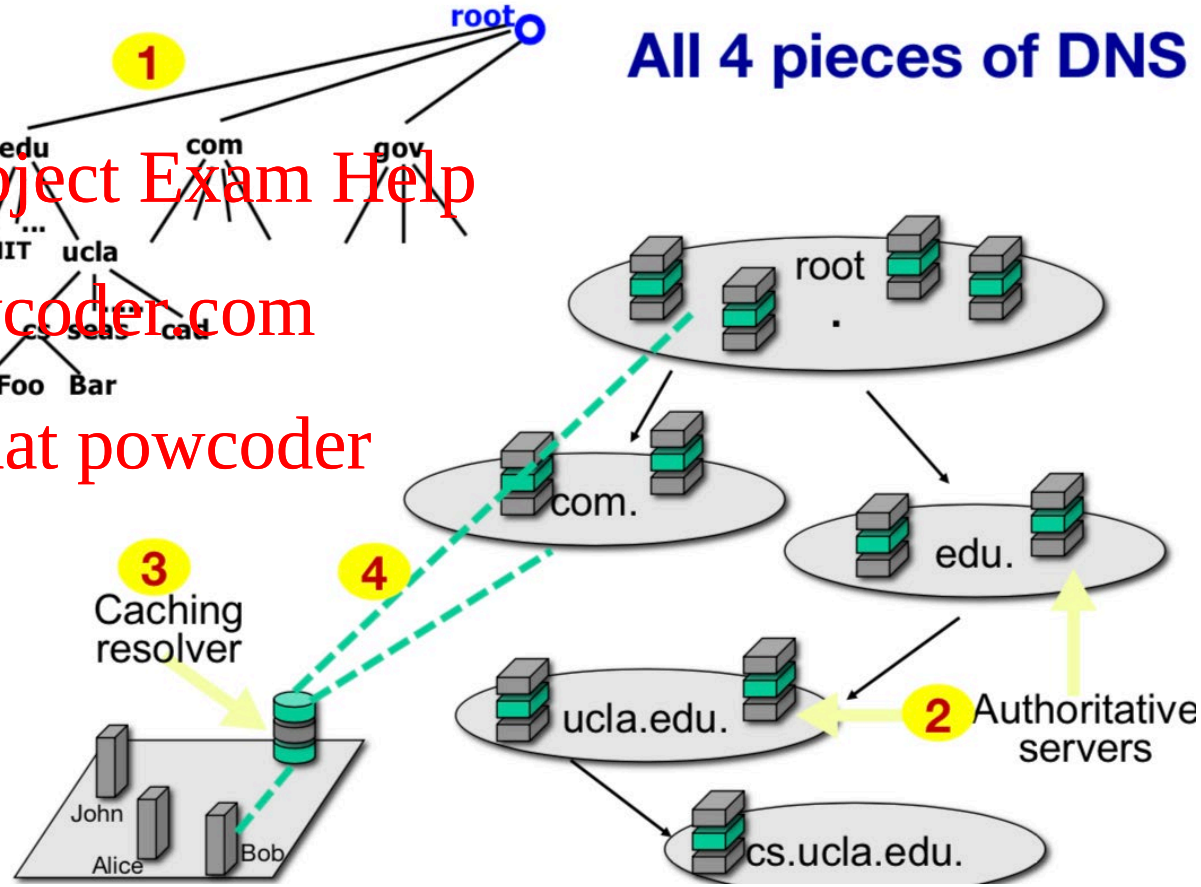
- DNS (Domain Name System) is how we go from ‘amazon.com’ to an IP address we can send actual traffic to.
- Every computer on the Internet uses it
  - e.g. URL to DNS query to IP address to set up TCP conn

1: Defines a hierarchical name space

2: Creates a distributed (federated) database, implemented through a hierarchy of authoritative servers

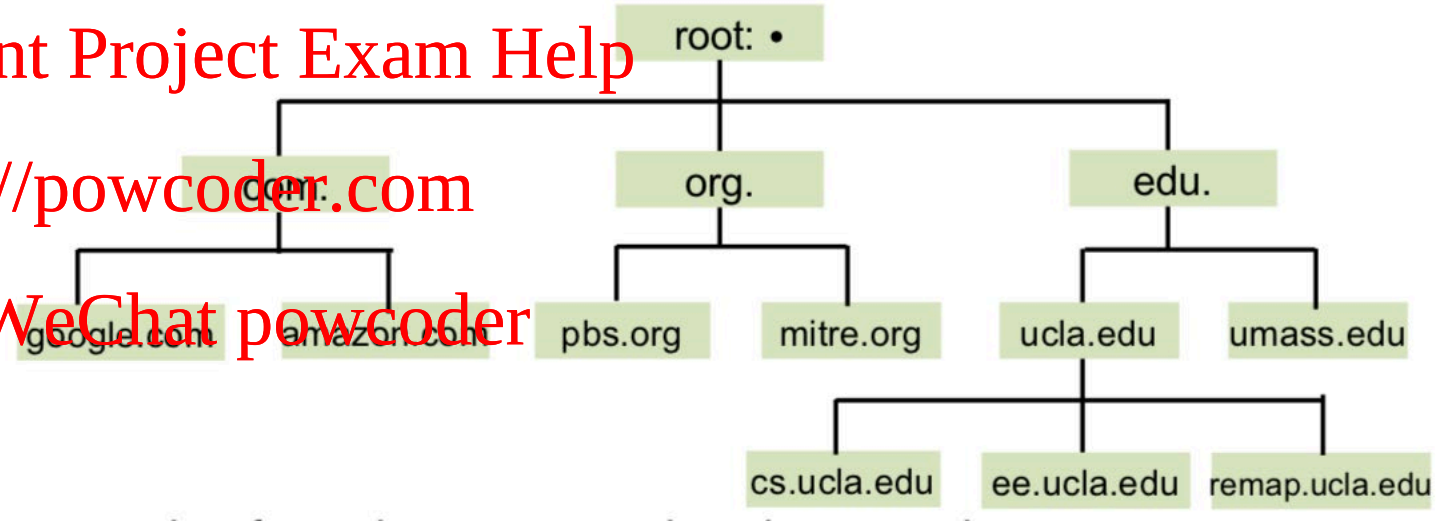
3. Local DNS servers (also called caching resolver) look up the database

4. Local caching resolvers query authoritative servers using DNS query protocol



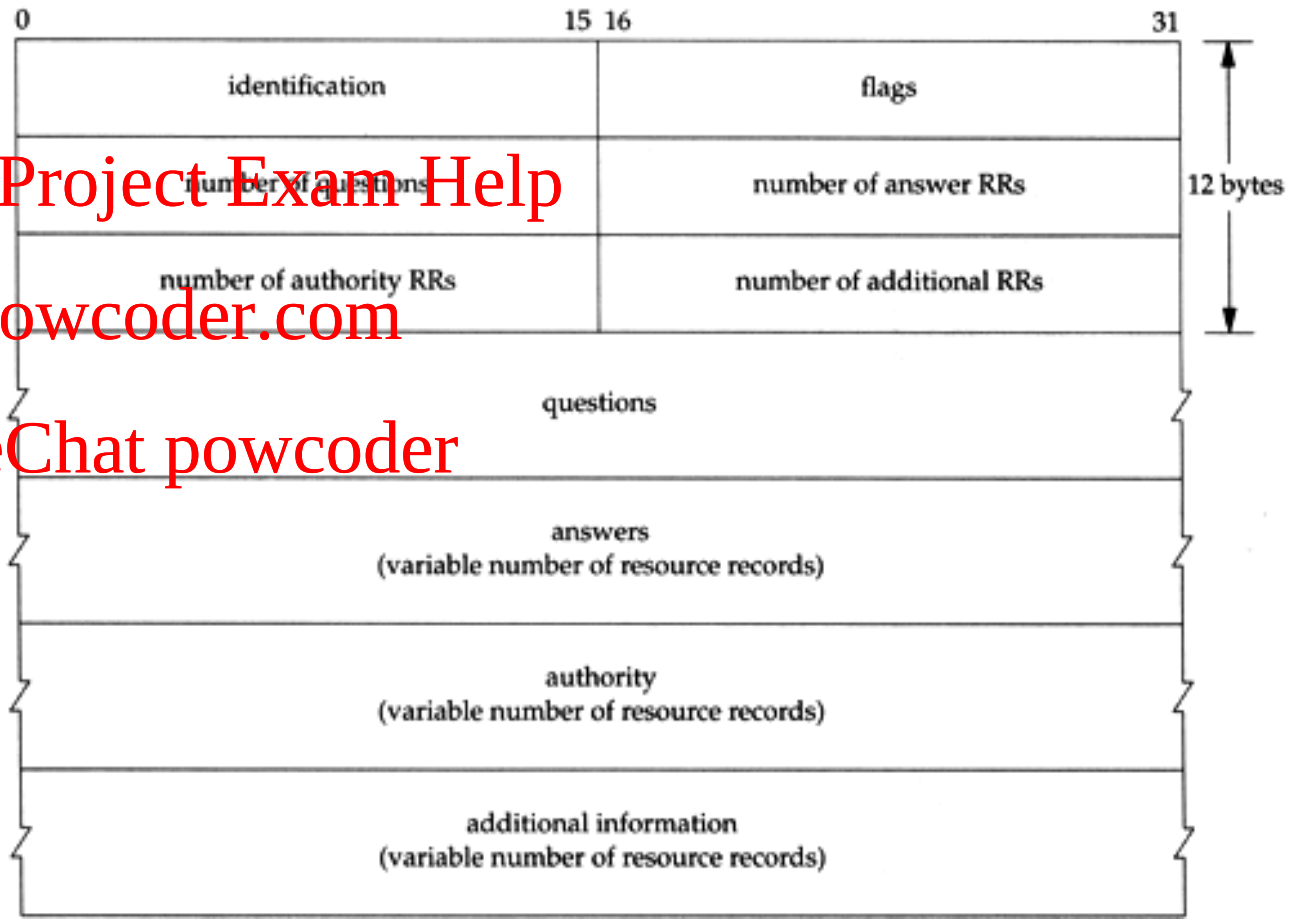
How do we define this namespace?

- Starting from the root, growing downwards
- Each leaf node is a DNS name (e.g. amazon.com)
- Each other node is a domain.
- Note that this hierarchy is independent of the topological connectivity.



DNS Message Format

- Any particular DNS query or response broken up as such
- Contains questions (e.g. ‘what IP address does x.y.z correspond to’) in the questions section.
- Response messages with contain RRs (resource records) that answer these queries.



3: How can you, personally, interface with the application layer?

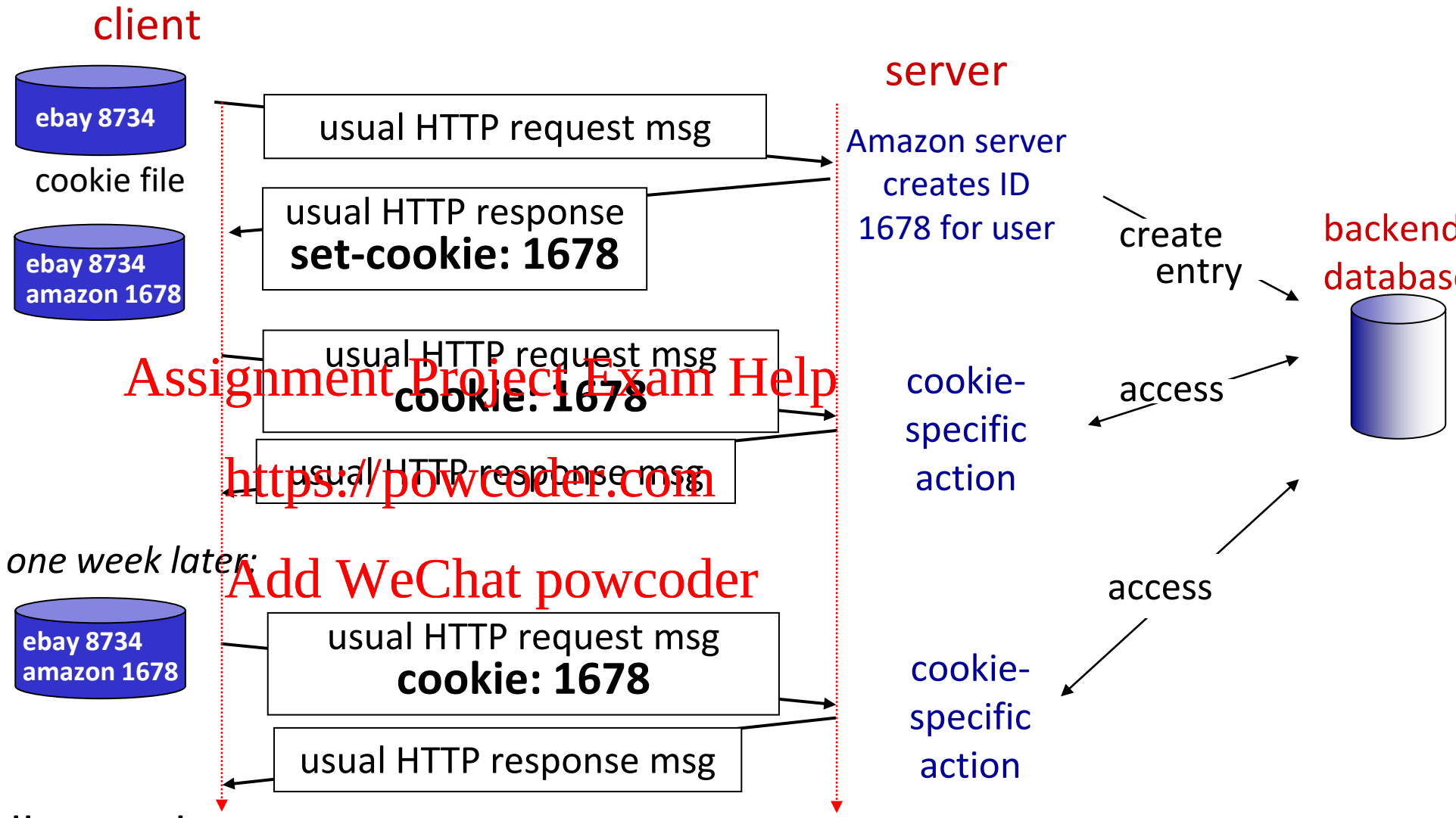
Photo by Neven Krcmarek on Unsplash

- Socket programming!
  - What's a socket?
    - An abstraction of a FIFO pipe (where the pipe is the network connection).
  - UDP: Unreliable datagram service
  - TCP: reliable, byte stream-oriented service
- Example Application:
  - Client reads a line of characters from terminal and sends to server
  - Server reads the lines, creates a response from them, sends back to client.
  - Client receives and displays this responses

Extra Time: How do we speed all of this up?

- Better Hardware
- Caching!
  - Web caching
  - CDNs

State



- HTTP is naturally stateless
- How do we then do multi-stage transactions?
- Cookies!