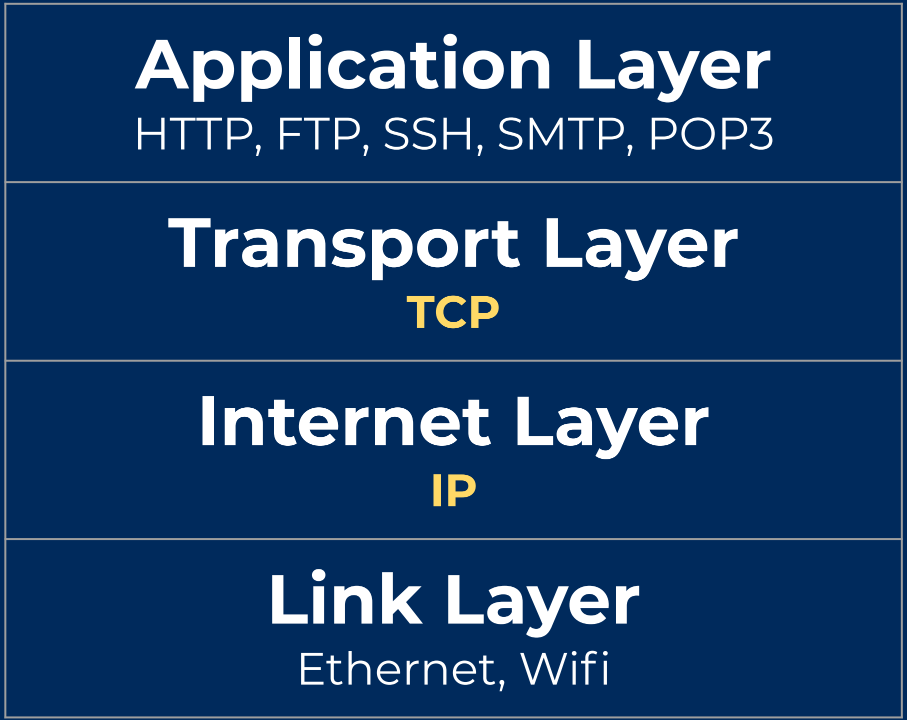
The World Wide Web and HTTP

CSC309

Mark Kazakevich

Last Time



Application Layer

HTTP, FTP, SSH, SMTP, POP3

Transport Layer

TCP

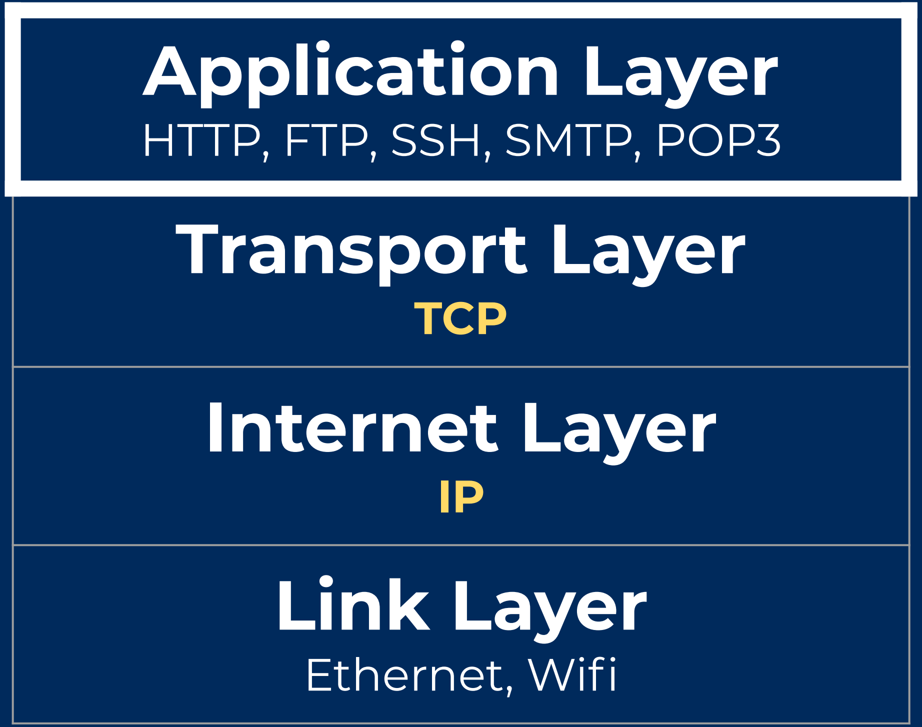
Internet Layer

IP

Link Layer

Ethernet, Wifi

Now



Application Layer

HTTP, FTP, SSH, SMTP, POP3

Transport Layer

TCP

Internet Layer

IP

Link Layer

Ethernet, Wifi

The World Wide Web

- The World Wide Web is **not** the Internet…
  - …but it happens to use the Internet to do its work

What is the web?

- A **global collection** of resources…
- ..which are **identifiable**…
- …and **linked** together.

Let’s discuss these three points

“A **global collection** of resources”

- A **web resource** can be any data we can send through the internet
    - Text, images, video, audio, etc.

- **Global** - want to access these resources no matter where they are in the world

- Where are they stored?
    - On **“web” servers** - Computers with resources that are accessible

“which are **identifiable**”

- We need a way to **get these resources** from their web servers
  - Necessary that we can **locate** where they are in the entire web
  - Need a **consistent** way to identify and access each resource

Uniform Resource Locator (URL)

Uniform Resource Locator (URL)

- Provides us with a way of specifying the location of a web resource
  - A.k.a. a **“web address”**

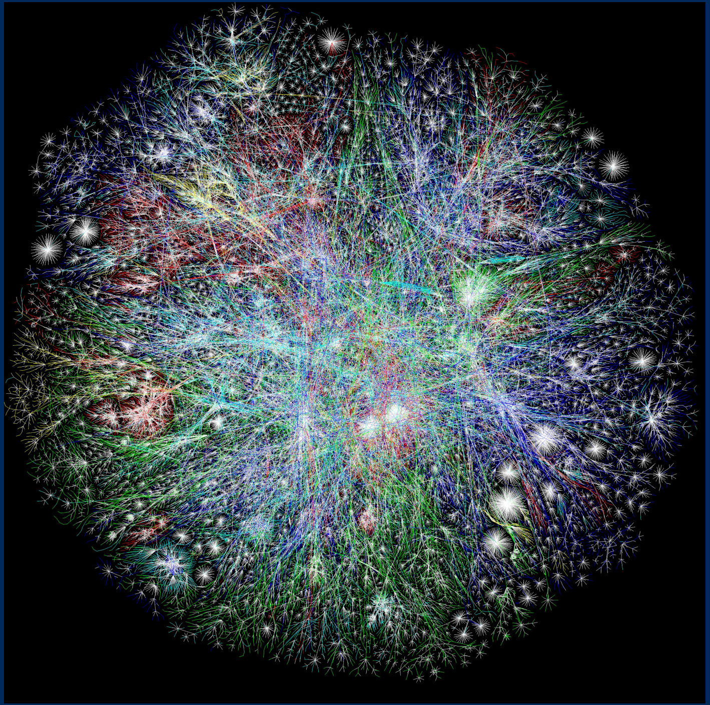
As you’ve seen it before:

`http://www.google.com`

We’ll talk more about how it works

“and **linked** together.”

- The web is...a **web,** after all!
- Resources link to other resources
  - Allows us to easily discover the web
- Those that are similar tend to link to each other



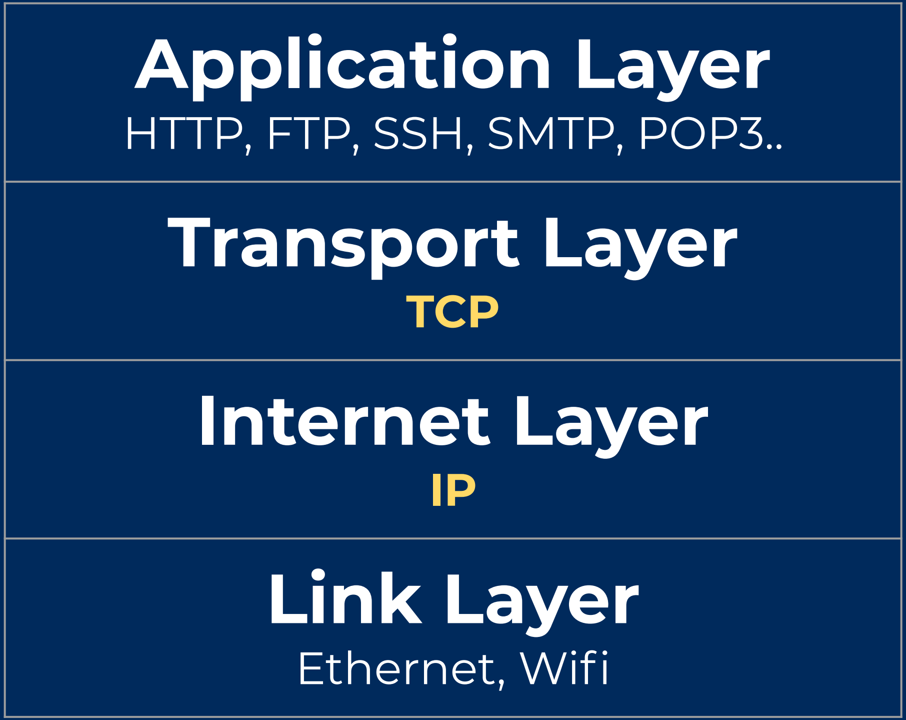
So that’s the World Wide Web

A **global collection** of resources which are **identifiable** and **linked** together.

Now…how do we put this vision into practise?

The Internet.

The World Wide Web works over the Internet



Application Layer

HTTP, FTP, SSH, SMTP, POP3..

Transport Layer

TCP

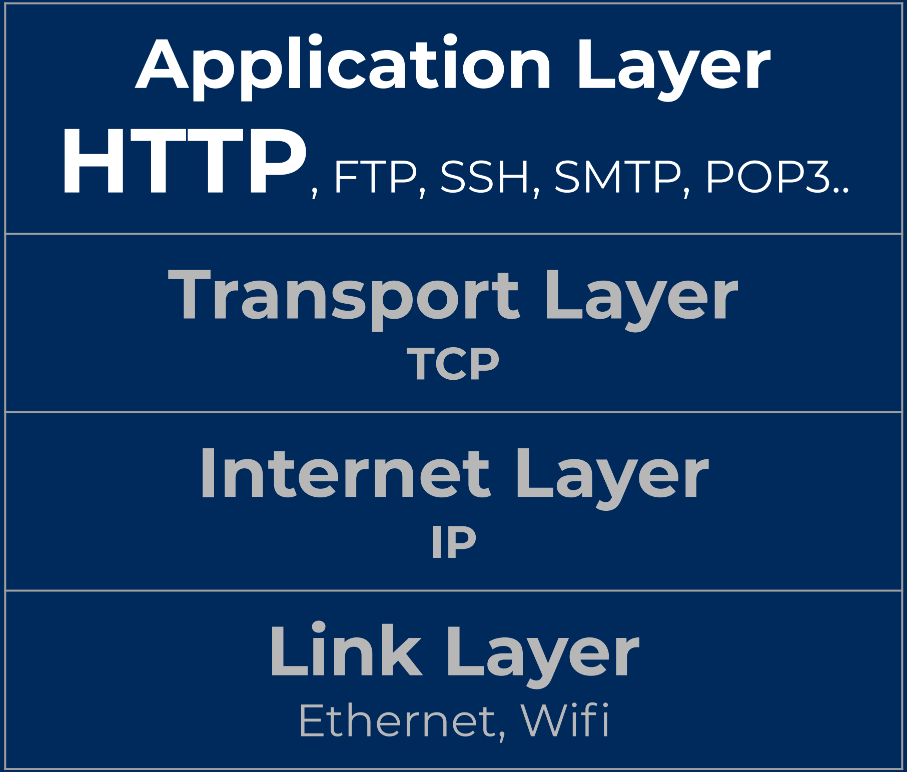
Internet Layer

IP

Link Layer

Ethernet, Wifi

Of particular importance...



Application Layer

HTTP, FTP, SSH, SMTP, POP3..

Transport Layer

TCP

Internet Layer

IP

Link Layer

Ethernet, Wifi

HyperText Transfer Protocol (HTTP)

- The protocol of the web

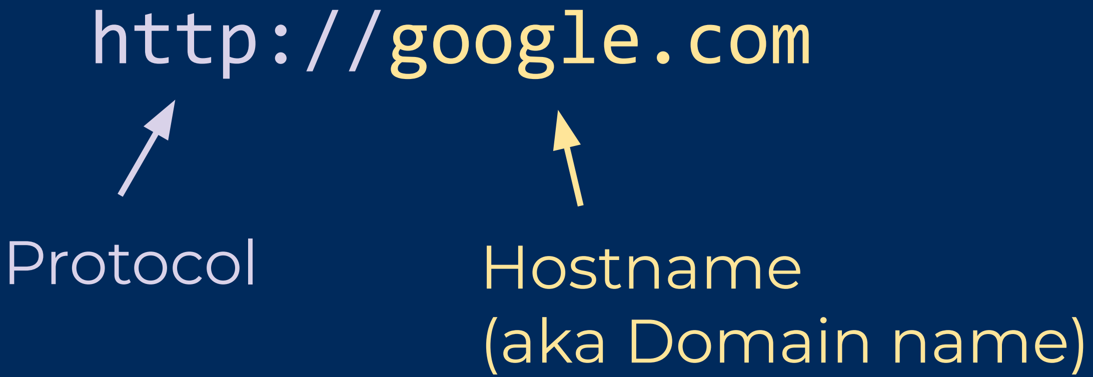
- Gives the client and server a mutual language at the application layer

HTTP

- Global collection of resources
    - **All machines** can use HTTP through applications - global reach

HTTP

- Identified through URLs



`http://google.com`

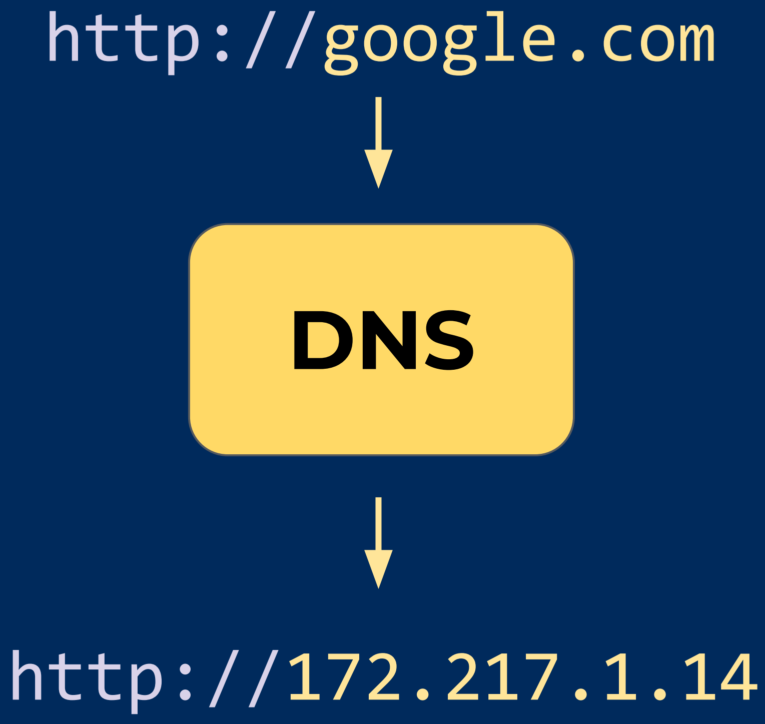
Protocol

Hostname
(aka Domain name)

- *Note: URLs are not unique to HTTP; they are used in other protocols as well*

HTTP URLs

- Hostnames translated to IP addresses by the Domain Name System (DNS)
- IP address can change, name can stay the name



http://google.com

DNS

http://172.217.1.14

URLs point to resources

This URL gives us one resource, a **web page**:

```
http://google.com
```

Most websites however, have more than one resource

Can access them by extending URL as needed:

```
http://google.com/location/of/resource
```

How do we **use** HTTP?

- **Accessing resources** through URLs doesn't always mean downloading something

- Think about how we use the web day-to-day
  - Download things
  - Upload things
  - Change things
    - I.e. Update our credit card info
  - Delete things
    - Embarrassing pictures

How do we **use** HTTP?

- We want to be able to ask a web server to do all of these things

- So let’s see how HTTP makes that happen

- HTTP works by **request-response**
    - Request from client
    - Response from server

- Request and response originate from Application Layer on both sides

HTTP Request includes…

- **URL**
  - To get to the resource on the server we want
- HTTP **Method**
  - To tell the server what we want to do with that resource
- Request **Headers** and **Body**
  - Give the server additional information about our request

HTTP Methods

- HTTP Methods are **verbs** that are used to label the actions we *expect* a server to take

Verb	Expected Server Action
GET	Retrieve a resource
POST	Create a resource
PATCH	Update a resource
DELETE	Delete a resource

- Technically speaking, server doesn’t have 100% obligation to do these expected actions, but they are pretty well followed standards.

- We’ll talk more about specific standards in the course.

Example: `GET` Request

- Let’s say we wanted to access a course website homepage:

`www.teach.cs.toronto.edu/~csc148h/winter/index.html`

Since we are **retrieving** a resource (a web page), we use the **GET** method. The request looks like:

```
GET /~csc148h/winter/index.html HTTP/1.1
Host: www.teach.cs.toronto.edu
```

Example: GET Request

```
GET /~csc148h/winter/index.html HTTP/1.1
Host: www.teach.cs.toronto.edu
```

HTTP method: `GET`

Resource: `/~csc148h/winter/index.html`

Host: `www.teach.cs.toronto.edu`

What does the response look like?

Let’s see it in **Postman**

- An app/browser extension that lets you easily make HTTP requests

- Nice GUI for seeing responses to requests

- Save requests and change settings on the fly

So the response has..

- **Response code**
  - Gives us standard indicator of the overall status of the response
- **Headers**
  - Give information about the response
- **Body**
  - The content of the resource, if available

Important: The web server decides what the URL does

Just because it looks like a path to some file in a filesystem, doesn’t mean it actually looks like that on the server.

```
http://google.com/path/to/resource
```

The server decides what accessing this URL does.

More on this when we get to server-side programming

HTTP: Linking together

- So what about the **web**?
    - We want our resources to be linked together somehow

- **HyperText** Transfer Protocol
    - Text/resources with “**hyperlinks**” - links to other resources
    - Similar resources are often linked together
    - This is what gives us the feeling of a **connected** web

TCP: How does this look one layer down?

- Remember that a web server listens for a request

- That means there needs to be a **process** on the server that is listening

- Issue: what if there are multiple processes that want to listen for connections?

Ports

- Every process on a computer that uses the internet is assigned a port
    - TCP or UDP port

- Server process that listens for **HTTP** requests usually uses port **80**

TCP with Ports

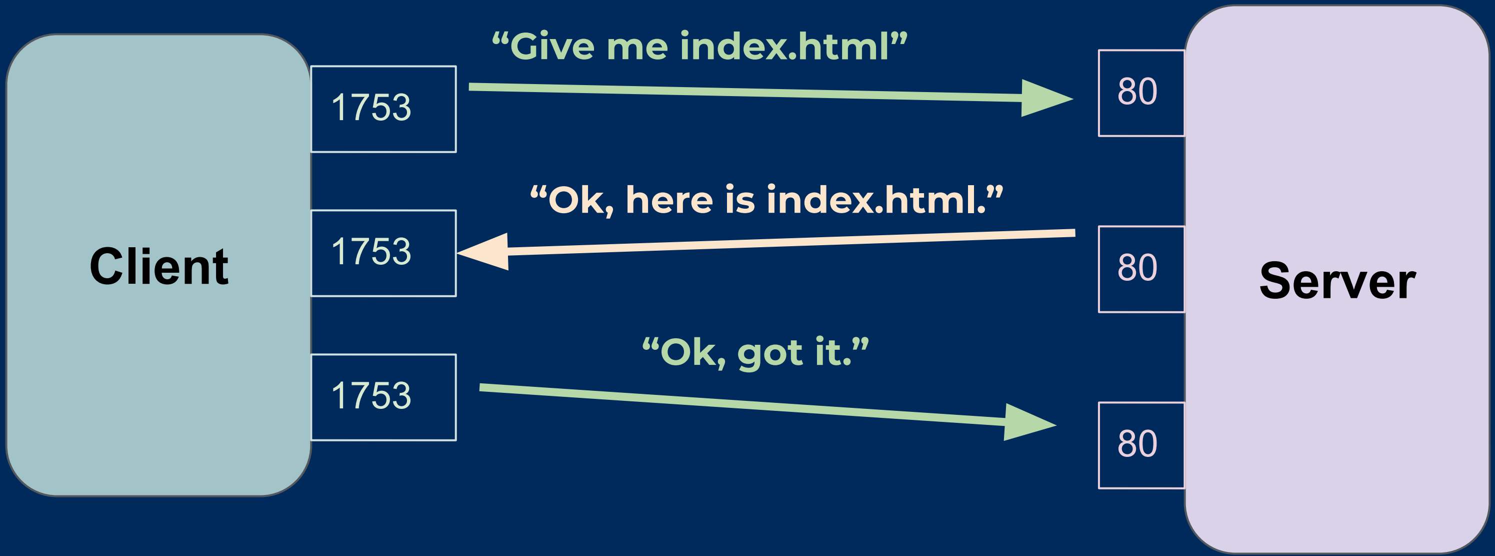
Application

Transport

Internet

Link

Assume server is listening on usual HTTP port (80), and client process talking through port 1753 (randomly assigned)



HTTP: “Stateless” protocol

- Each request is *independent*,
  - Server doesn’t need to keep track of previous requests
  - Doesn’t care how many are sent at once

- This simplifies the protocol

- Illusion of state (e.g. knowing which pages the user browsed) can still occur, but is not part of the protocol