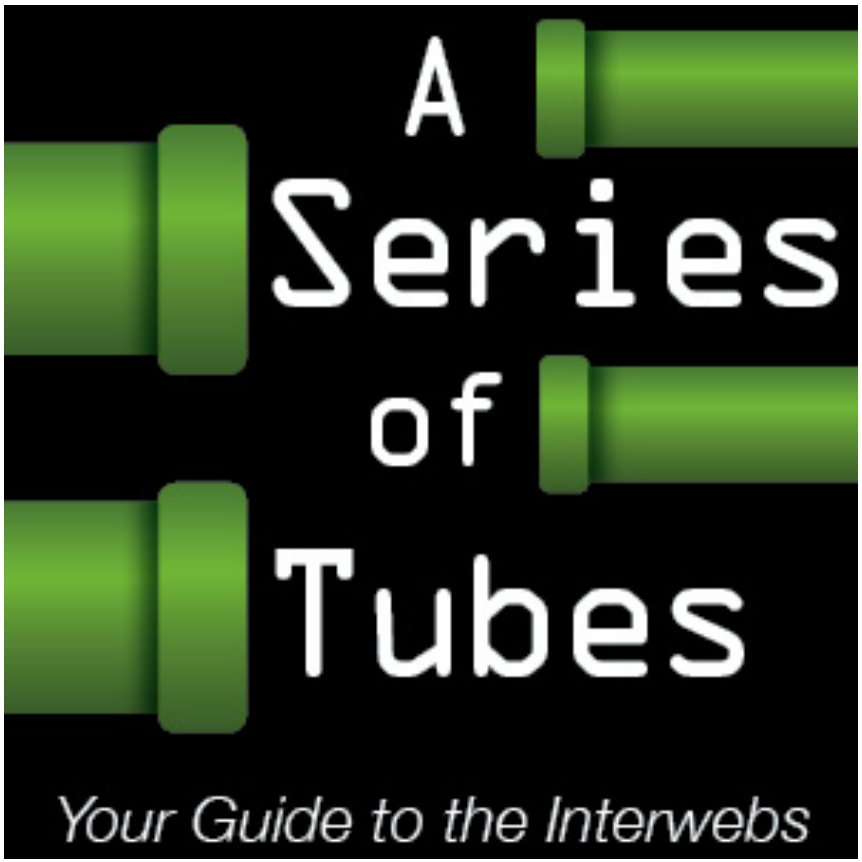
# Static Pages



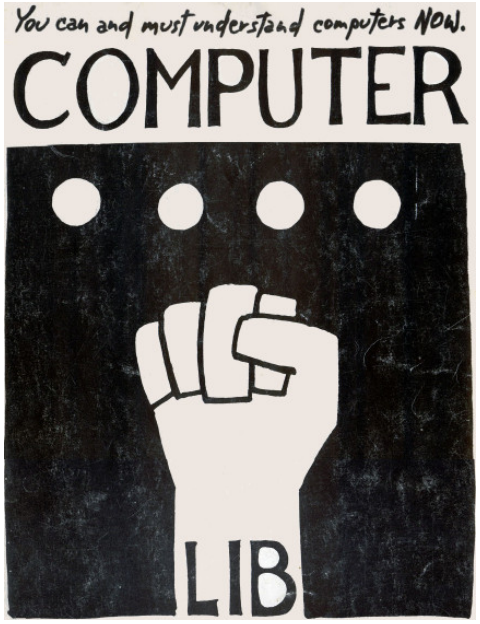

Slides by Andrew DeOrio

# Agenda

- **History**
- Markup languages
  - HTML
  - CSS
- The request response cycle
  - URL encoding
  - HTTP

# Ancient history

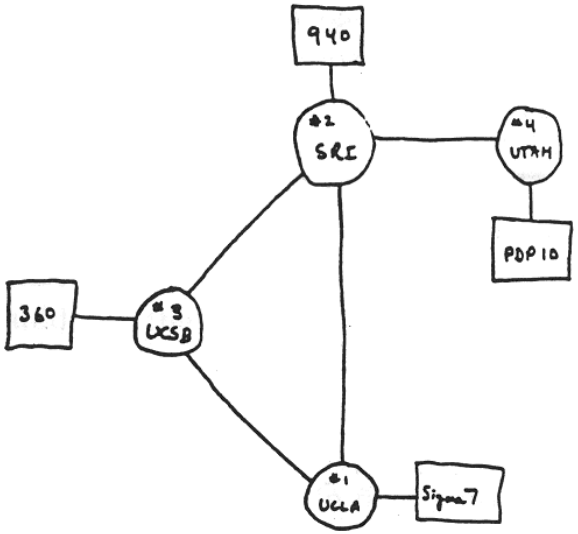
- Pre-history
  - Dewey Decimal system, library science
- 1960: Ted Nelson (23 yrs old) → Xanadu
  - Hypertext vision of WWW
  - Focus on copyright, consistent (bidirectional) links, versioning
  - Much richer than the web
  - Never fully implemented




- 1961 Kleinrock paper on packet switching
  - Contrast with phone lines - circuit switched.

# Paleolithic era

- 1965 Gordon Moore proposes law
- 1966 Design of ARPAnet




THE ARPA NETWORK

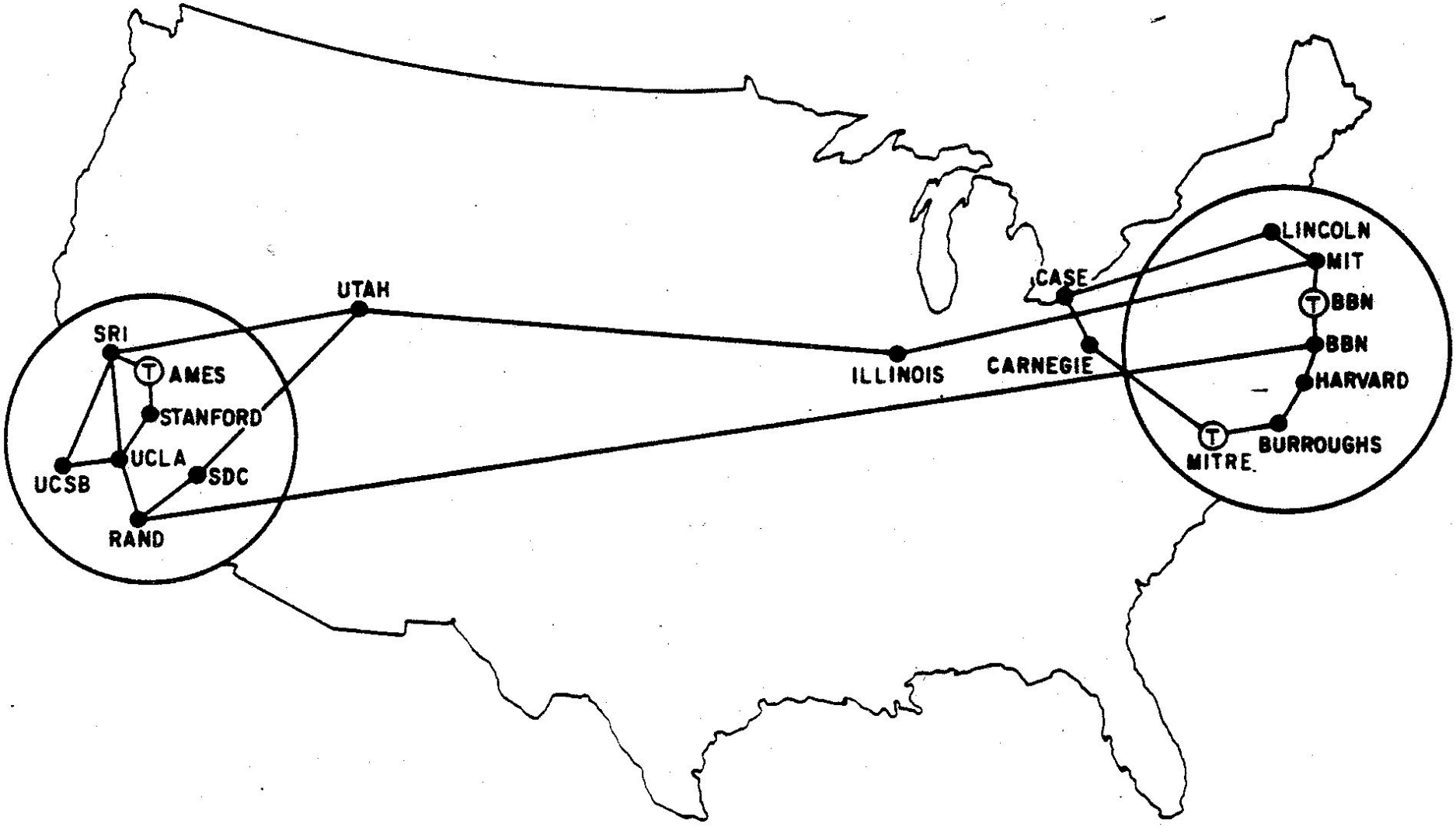
DEC 1969

4 NODES

# Paleolithic era

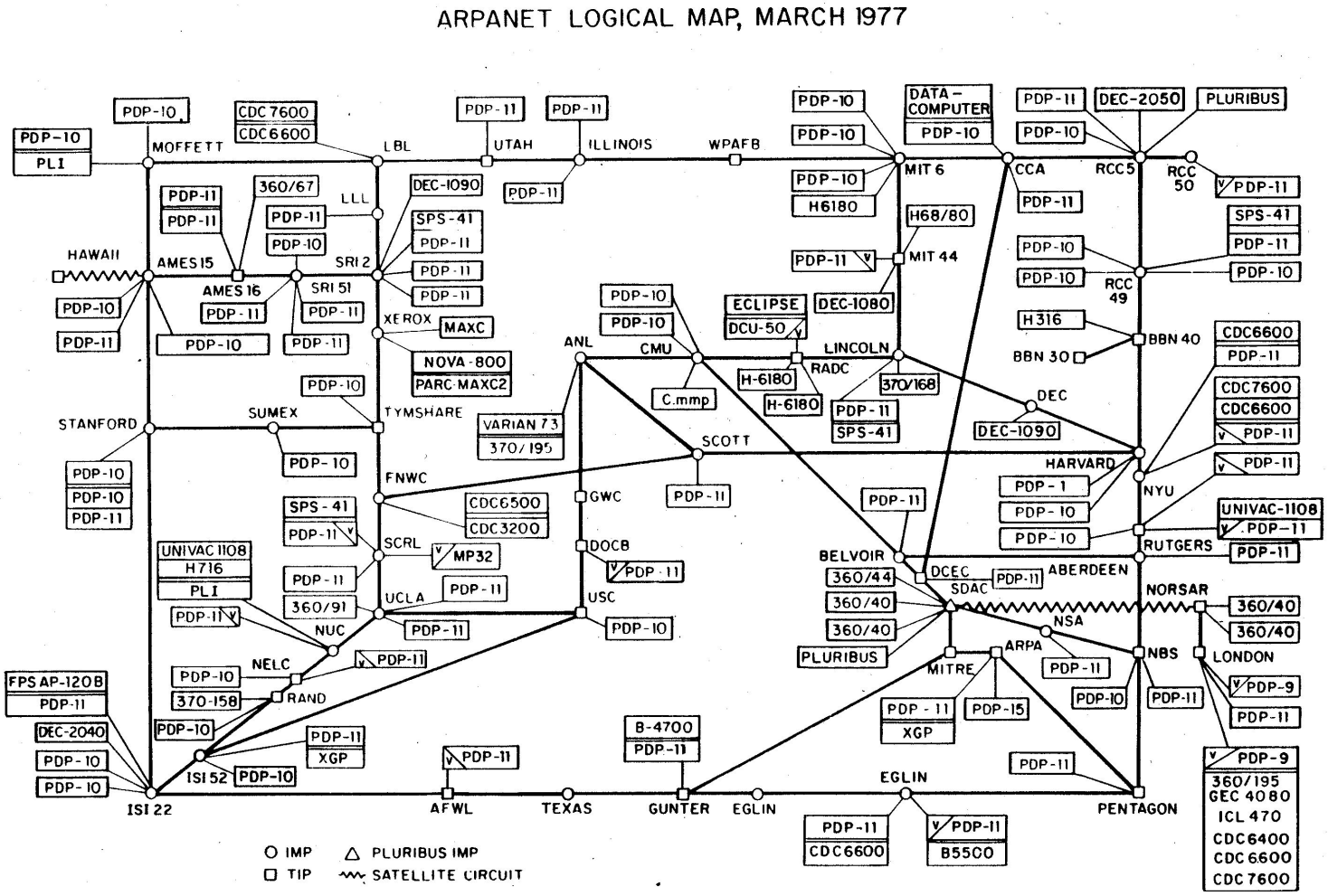
- 1965 Gordon Moore proposes law
- 1966 Design of ARPAnet

- 1969 First ARPAnet msg, UCLA -> SRI
- 1970 ARPAnet spans country, has 5 nodes
- 1971 ARPAnet has 15 nodes
- 1972 First email programs, FTP spec


LINCOLN
MIT
CASE
UTAH
BBN
BBN
SRI
HARVARD
AMES
ILLINOIS
CARNEGIE
STANFORD
BURROUGHS
UCLA
MITRE
UCSB
SDC
RAND

# The internet ramps up

- 1983 ARPAnet uses TCP/IP; design of DNS; 1000 hosts on ARPAnet



# The internet ramps up

- 1983 ARPAnet uses TCP/IP; design of DNS; 1000 hosts on ARPAnet
- 1985 symbolics.com (computer mfg) is first registered domain name

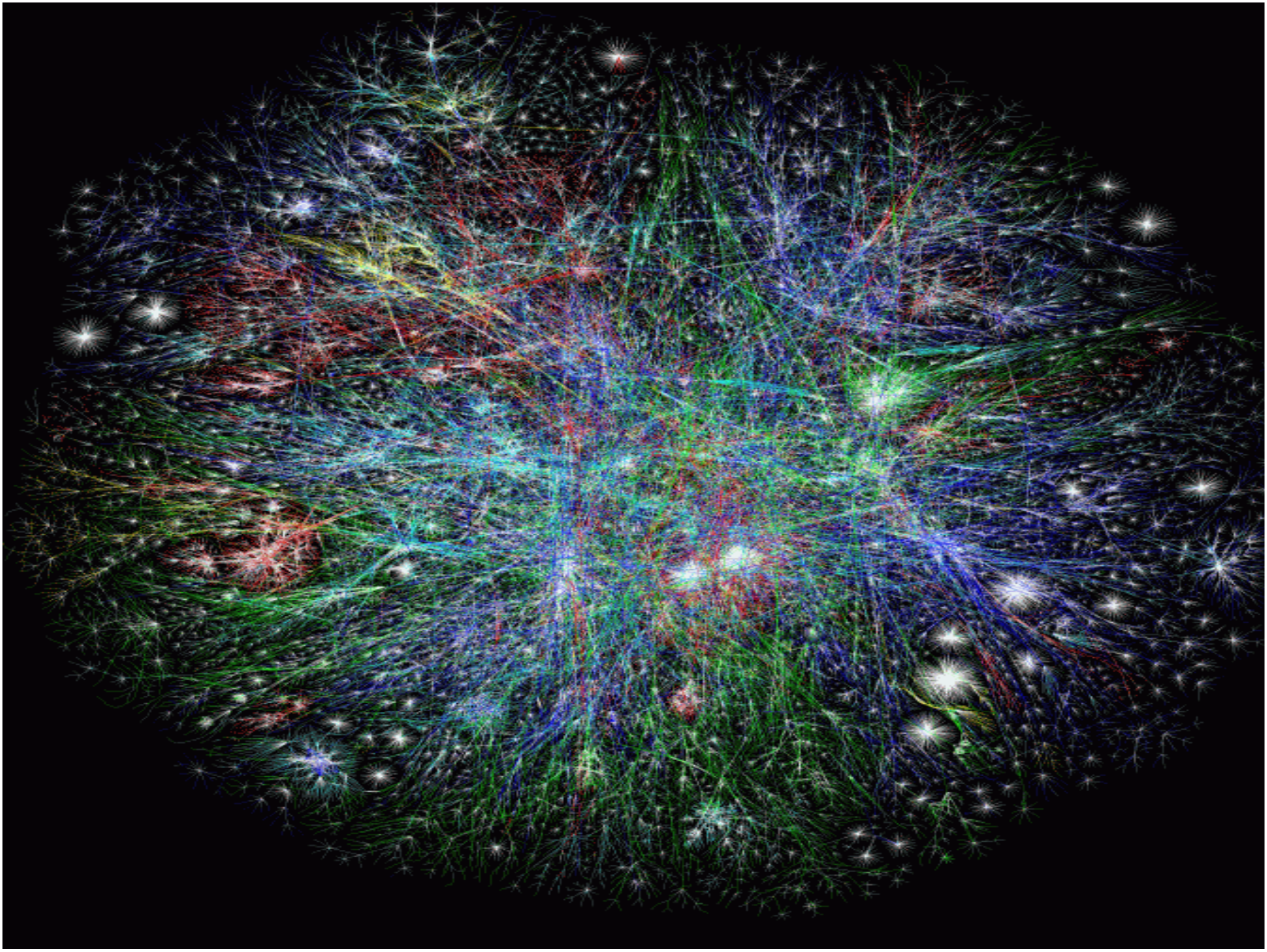
- 1989 100K hosts on Internet
- 1990 Cisco goes public; Tim Berners-Lee creates WWW at CERN; 3M Internet users world-wide

# Modern age

- 1993 WWW Wanderer
  - First crawler
- 1995 Yahoo, Amazon
- 1998 Google & PageRank
- 2003 Skype


# Modern age

- 2004 Facebook founded
- 2006 Twitter founded
- 2010 Instagram founded
- 2019 Facebook has 2.4 B monthly active uses
  - ~30% of humanity
  - https://investor.fb.com/investor-events/

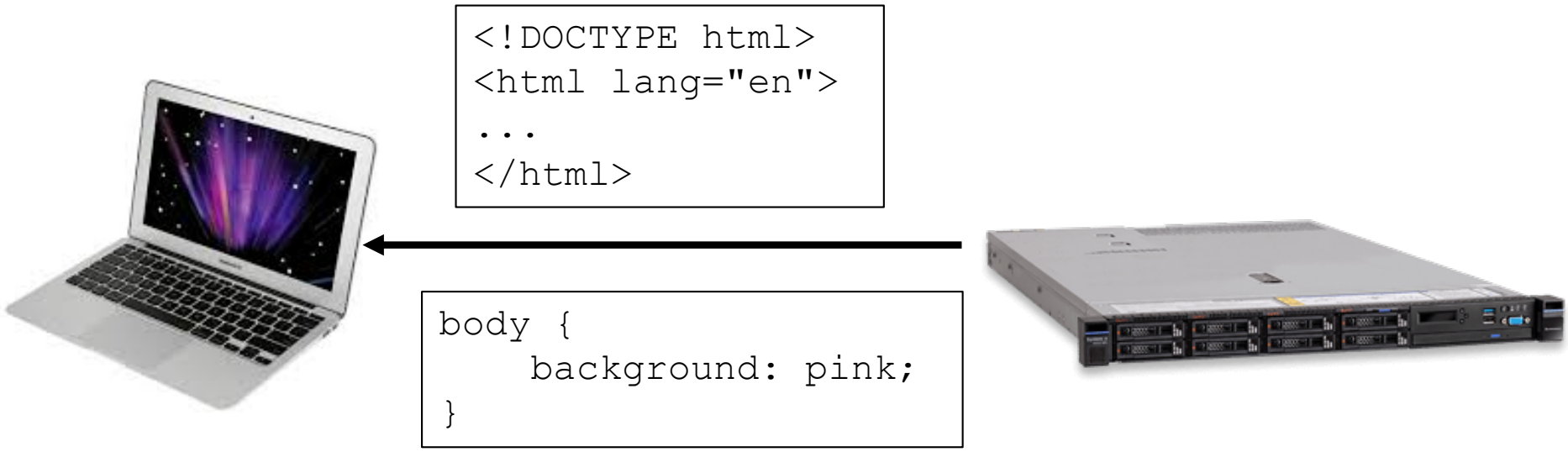


# Agenda

- History
- **Markup languages**
  - **HTML**
  - **CSS**
- The request response cycle
  - URL encoding
  - HTTP

# Review: static pages

- A *static page* is only HTML/CSS
  - No programming language on the server
  - Same content every time the page is loaded



# Review: the request response cycle

- The request response cycle is how two computers communicate with each other on the web

1. A client requests some data
2. A server responds to the request

*client*

*internet*

*server*

# Review: the request response cycle

- A client requests a web page


- A server responds with an HTML file
  - It might create the content on-the-fly

- The client renders the HTML

# Markup language

- How do we improve the presentation of the plain text transmitted over HTTP …

```
Academics+Admissions Research People Industry
News+Awards Events
```

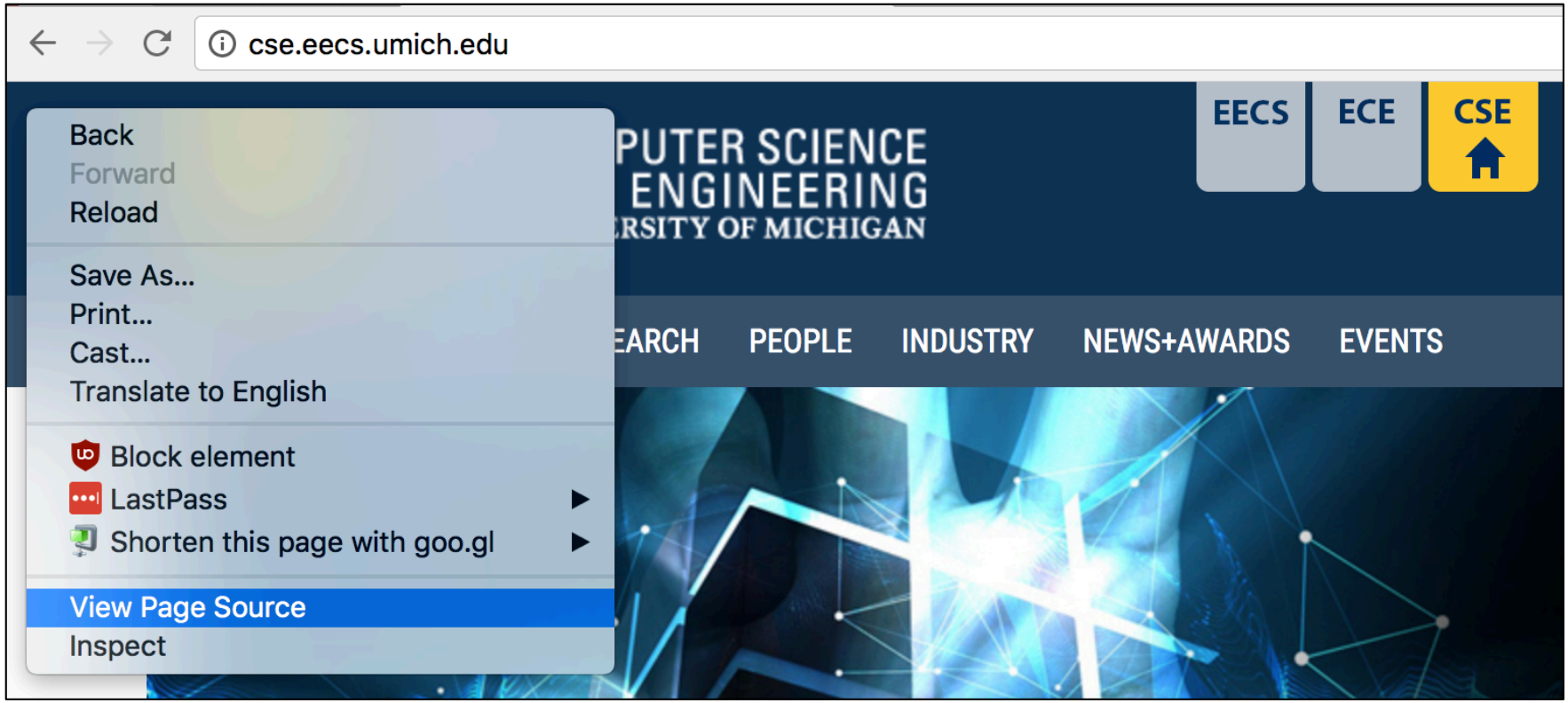
- … so that it looks like this?

# HyperText Markup Language: HTML

- HyperText Markup Language
- Invented by Tim Berners-Lee in 1990
- Set of tags for rendering page

# Example

- See the HTML source with “View Source” in your web browser



# Example


- Add **tags** as "mark up" to text
- Document still "primarily" text

```
<html>
  <head></head>
  <body>
    <nav>
    <ul>
      <li><a href="">Academics+Admissions</a></li>
      <li><a href="">Research</a></li>
      <li><a href="">People</a></li>
      <li><a href="">Industry</a></li>
      <li><a href="">News+Awards</a></li>
      <li><a href="/eecs/etc/events/cseevents.html">Events</a></li>
    </ul>
    </nav>
  </body>
</html>
```

# Markup language vs. programming language

- Markup language: data presentation
  - How does it look?
- Programming language: data transformation
  - How does it work?
- Markup language examples:
  - HTML, used for web pages
  - XML, used to communicate data between applications.  Less popular today.
  - Markdown, used by GitHub
  - troff and nroff, used for man pages
  - TEX, used for publishing
- Programming language examples:
  - C/C++, Python, JavaScript, Bash (shell scripting)

# Hypertext

- Text with embedded links to other documents.
- Anchor tag

```
<a href="/eecs/etc/events/cseevents.html">
   Events
</a>
```

# Escape strings in HTML

- Some characters have special meaning in an application context
  - Example: “<“ in HTML
- What if you want to communicate such a character as an ordinary character?
  - Example: “I <3 chickens”
- Need an escape string
  - Example: “I **<**3 chickens”

# Escape strings in URLS

- Some characters have special meaning in an application context
  - Example: “ “ (a space) in a URL
- What if you want to communicate such a character as an ordinary character?
  - Example: `hello world.jpg`
- Need an escape string
  - Example: `hello%20world.jpg`

# XML

- eXtensible Markup Language
- No standard set of tags!!
- Define tags and use them
- Tags form a hierarchy of objects
  - Each open tag has a matching close
  - Must balance, like parentheses
  - Can build a tree of document objects
    - Document Object Model (DOM)

# XHTML

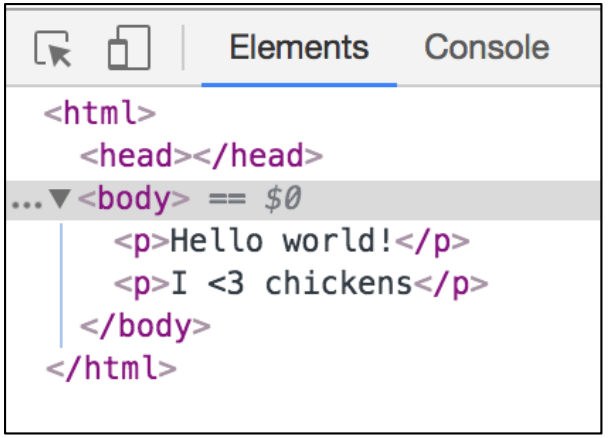
- HTML permits unbalanced tags.
  - Example: the paragraph tag has no closing </p> tag
    ```
    <p>This is an HTML paragraph.
    ```
- XHTML is a dialect of HTML that meets XML rules
  - Proper containment
- XHTML is still HTML, so it is understood by browsers
  - Example:
    ```
    <p>This is an XHTML paragraph.</p>
    ```

# Document Object Model (DOM)

- HTML tags form a tree

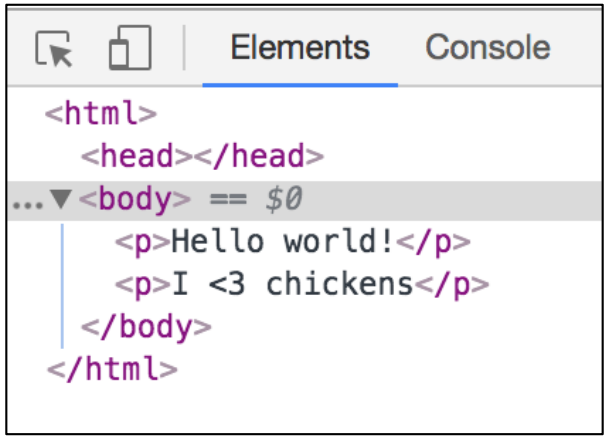
```
<html>
  <head></head>
  <body>
    <p>Hello world!</p>
    <p>I &lt;3 chickens</p>
  </body>
</html>
```


- This tree is called the Document Object Model (DOM)
- Inspect the DOM with
  - Chrome developer tools
  - Firefox developer tools



# Document Object Model (DOM)

- The DOM is a data structure built from the HTML
- In the DOM, everything is a **node**
  - All HTML elements are element nodes
  - Text inside HTML elements are text nodes



# HTML5

- HTML5 merges all that has happened over years related to HTML:
  - XHTML
  - Browser-specific extensions of HTML
- Finalized and published by W3C in 2014
- Check your HTML using a website
  - `https://validator.w3.org/nu/?doc=URL_GOES_HERE`
- Check your HTML at the CLI

```
$ pip install html5validator
$ html5validator --root YOUR_HTML_DIR/
```

# Content, presentation, layout

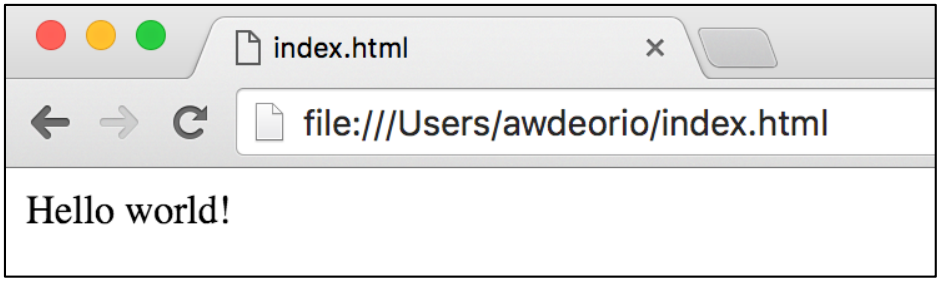
- HTML has tags for all.
  - `<h1>` is content
  - `<b>` is presentation
- Good to separate these.
- Content is data; presentation is words, tables, organization; layout is visual.
- Use HTML for content and presentation, add CSS for layout.

# CSS – try this

- HTML for content

```
<!-- index.html -->
<html>
<head>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
<body><p>Hello world!</p></body>
</html>
```

- Load the page in your web browser

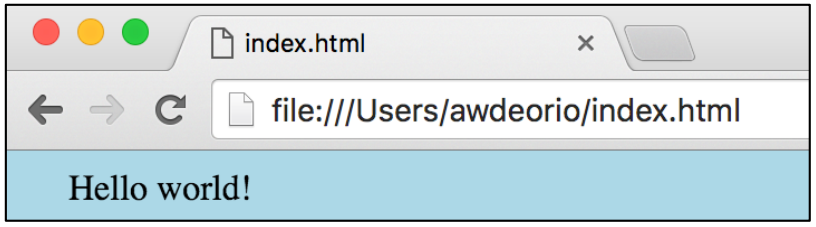


# CSS – try this

- CSS for layout

```
/* style.css */
body {
    background-color: lightblue;
}
p {
    margin-left: 20px;
}
```

- Now reload index.html



# Agenda

- History
- Markup languages
  - HTML
  - CSS
- **The request response cycle**
  - **URL encoding**
  - **HTTP**

# A closer look at the request response cycle

- Command line web browser

```
$ lynx cse.eecs.umich.edu
```

- Download web page

```
$ wget http://cse.eecs.umich.edu/
$ 2017-09-12 15:58:44 (25.8 MB/s) -
‘index.html’ saved [31745]
```

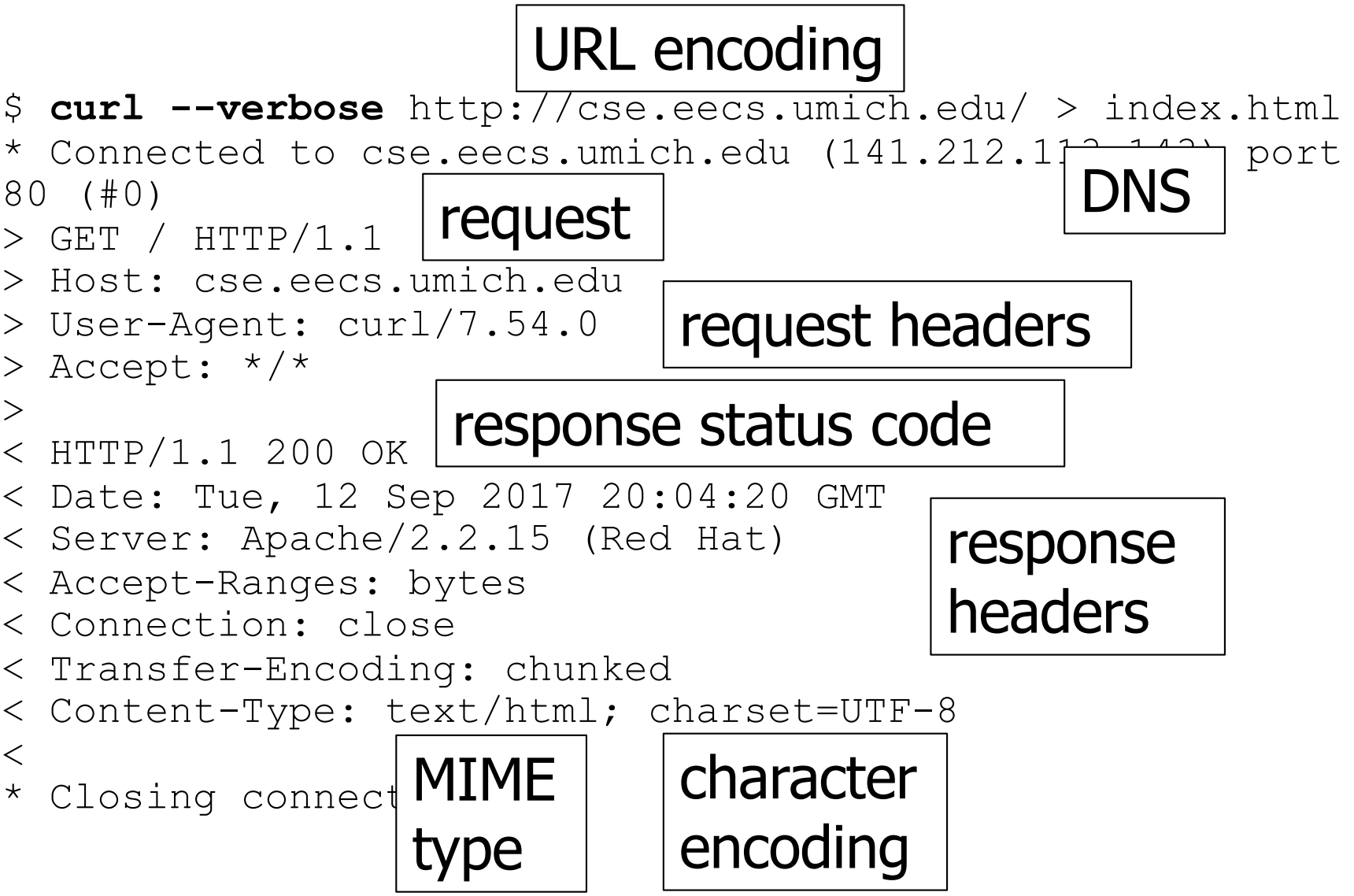
- Dump web page to terminal

```
$ curl http://cse.eecs.umich.edu/
<html lang="en">
  ...
</html>
```

# Details of the request response cycle

```
$ curl --verbose http://cse.eecs.umich.edu/ > index.html
* Connected to cse.eecs.umich.edu (141.212.113.143) port
80 (#0)
> GET / HTTP/1.1
> Host: cse.eecs.umich.edu
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 12 Sep 2017 20:04:20 GMT
< Server: Apache/2.2.15 (Red Hat)
< Accept-Ranges: bytes
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
<
* Closing connection 0
```

# Details of the request response cycle



# URL encoding

```
$ curl --verbose http://cse.eecs.umich.edu/
```

```
protocol://server:port/path?query#fragment
```

- URLs have several parts
  - Protocol
  - Server
  - Port
  - Path
  - Query
  - Fragment

# URL encoding: protocol

**protocol**://server:port/path?query#fragment

- protocol tells the server what protocol to use. In other words, what "language" to speak
- Example: unencrypted http

```
$ curl --verbose http://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu
(141.212.113.143) port 80 (#0)
```

- Example: encrypted https

```
$ curl --verbose https://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu
(141.212.113.143) port 443 (#0)
* TLS 1.2 connection using
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
* Server certificate: www.cse.umich.edu
```

# URL encoding: server

`protocol://`**`server`**`:port/path?query#fragment`

- `server` helps locate the machine we want to talk to
- Example

```
$ curl --verbose http://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu
(141.212.113.143) port 80 (#0)
```

- DNS lookup translates server name into an IP address

```
$ host cse.eecs.umich.edu
cse.eecs.umich.edu has address 141.212.113.143
```

# URL encoding: port

`protocol://server:`**`port`**`/path?query#fragment`

- `port` is used to identify a specific service
- One host machine, several servers
  - Example: 80 is typically HTTP, 443 for HTTPS
- Check if a port is open

```
$ nc -v -z cse.eecs.umich.edu 80
cse.eecs.umich.edu [141.212.113.143] 80 (http)
open
$ nc -v -z cse.eecs.umich.edu 443
cse.eecs.umich.edu [141.212.113.143] 443 (https)
open
```

- See what ports are open

# URL encoding: port

`protocol://server:`**`port`**`/path?query#fragment`

- See what ports are open

```
$ nmap cse.eecs.umich.edu
PORT     STATE    SERVICE
22/tcp   open     ssh
80/tcp   open     http
443/tcp  open     https
5666/tcp open     nrpe
```

- WARNING: it's "not nice" to port scan!
  - Some websites might blacklist you for doing this

# URL encoding: path

`protocol://server:port`**`/path`**`?query#fragment`

- `path` is a file name relative to the server root
- Default is `/index.html`
  `$ curl http://cse.eecs.umich.edu`
  is the same as
  `$ curl http://cse.eecs.umich.edu/index.html`
- A different path loads a different page
  `$ curl http://cse.eecs.umich.edu/eecs/faculty/csefaculty.html`

# Absolute vs. relative paths

- Absolute path
  - Starts with a slash (/)
  - Examples:
    ```
    <img src="/uploads/9887e06812ef434d291e4936417d125cd594b38a.jpg">
    <a href="/u/awdeorio/">awdeorio</a>
    ```
- Relative path
  - Computed starting “present directory”.  AKA, the path of the current page
  - Example:
    ```
    <img src="9887e06812ef434d291e4936417d125cd594b38a.jpg">
    <a href="awdeorio/">awdeorio</a>
    ```
- Prefer absolute path for automatically generated pages
  - Including P1
- Prefer to automatically generate links
  - Remember this on P2

# URL encoding: query

`protocol://server:port/path`**`?query`**`#fragment`

- `query` string is a general-purpose set of parameters that the server (or specified resource on server) can use as it pleases

- http://cse.eecs.umich.edu/eecs/etc/fac/CSEfaculty.html
- http://cse.eecs.umich.edu/eecs/etc/fac/CSEfaculty.html?match=All
- http://cse.eecs.umich.edu/eecs/etc/fac/CSEfaculty.html?match=Lecturer
- http://cse.eecs.umich.edu/eecs/etc/fac/CSEfaculty.html?match=Tenure

# URL encoding: fragment

`protocol://server:port/path?query`**`#fragment`**

- `fragment` is identified at the client, ignored by server
- Example: navigate directly to the section labeled "Linking"
  `http://en.wikipedia.org/wiki/World_Wide_Web`**`#Linking`**

# Details of the request response cycle

```
$ curl --verbose http://cse.eecs.umich.edu/ > index.html
* Connected to cse.eecs.umich.edu (141.212.113.143) port
80 (#0)
> GET / HTTP/1.1
> Host: cse.eecs.umich.edu
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 12 Sep 2017 20:04:20 GMT
< Server: Apache/2.2.15 (Red Hat)
< Accept-Ranges: bytes
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
<
* Closing connection 0
```

# HTTP

- Hypertext Transfer Protocol
- Request/response protocol
  - Client (your browser) opens connection to server and writes a request
  - Server responds appropriately
  - Connection is closed
  - That's it
- Server can't open connection to client
- Completely stateless
  - Each request is treated as brand new
  - No state => no history

# HTTP manual example

```
$ telnet cse.eecs.umich.edu 80
Trying 141.212.113.143...
Connected to cse.eecs.umich.edu.
Escape character is '^]'.
GET /index.html HTTP/1.0
```

request

```
HTTP/1.1 200 OK
...
<!doctype html>
...
</html>
```

response

# HTTP request methods

- Request method indicates server action
- GET: request a resource
  - Example: load a page
- HEAD: identical to GET, but without response body
  - Example: see if page has changed
- POST: send data to server
  - Example: web form

- Others we will cover later in the REST API lecture

# HTTP request headers

- Headers accompany request
- Most are optional

```
$ curl --verbose http://cse.eecs.umich.edu/ >
index.html
* Connected to cse.eecs.umich.edu
(141.212.113.143) port 80 (#0)
> GET / HTTP/1.1
> Host: cse.eecs.umich.edu
> User-Agent: curl/7.54.0
> Accept: */*
```

- `Host` distinguishes between DNS names sharing a single IP address
  - Required as of HTTP/1.1
- `User-Agent:` which browser is making the request
- `Accept:` which content ("file") types the client will accept

# User agent

- When a browser visits a page, it identifies itself with a `User-agent` string
  - For example, check yours out:
  - http://www.whatismyuseragent.net/
- Example from Google Chrome:
  - `Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.103 Safari/537.36`
  - Previously used to indicate compatibility with the Mozilla rendering engine
  - During the "browser wars", some web sites would only send advanced features to some user agents

# HTTP status code

- Response starts with a status code
  - 1XX: Informational
  - 2XX: Successful
  - 3XX: Redirection Error
  - 4XX: Server Error
- `$ curl --verbose http://cse.eecs.umich.edu/`
  `> GET / HTTP/1.1`
  `< HTTP/1.1` **`200 OK`**
- `$ curl --verbose http://cse.eecs.umich.edu/`**`asdf`**
  `> GET /asdf HTTP/1.1`
  `< HTTP/1.1` **`404 Not Found`**

# HTTP response headers

- Headers accompany a response
- Most are optional

```
$ curl --verbose http://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu
> GET / HTTP/1.1
...
< HTTP/1.1 200 OK
< Date: Tue, 12 Sep 2017 20:04:20 GMT
< Server: Apache/2.2.15 (Red Hat)
< Accept-Ranges: bytes
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
```

# HTTP content type

- Content type describes the "file" type and encoding

```
$ curl --verbose http://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu
> GET / HTTP/1.1
...
< HTTP/1.1 200 OK
...
< Content-Type: text/html; charset=UTF-8
```

# HTTP content type

- Content type describes the "file" type and encoding

```
$ curl --verbose
http://cse.eecs.umich.edu/eecs/images/CSE-Logo-
Mobile.png > CSE-Logo-Mobile.png
* Connected to cse.eecs.umich.edu
> GET /eecs/images/CSE-Logo-Mobile.png HTTP/1.1
...
< HTTP/1.1 200 OK
< Content-Type: image/png
```

COMPUTER SCIENCE & ENGINEERING

UNIVERSITY OF MICHIGAN

# MIME Types

`Content-Type: **text/html;** charset=UTF-8`

- MIME: Multipurpose Internet Mail Extensions
- Way to identify files
- Browser can open or display content correctly
- **<type>/<subtype>**

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Complete_list_of_MIME_types

# Character encodings

`Content-Type: text/html;` **`charset=UTF-8`**

- Character encoding maps bits to symbols
- ASCII was the first
  - 128 different alphanumeric characters
  - `char` in C/C++
- ISO-8859-1 was the default character set for HTML 4
  - 256 codes
  - HTML5 also supported UTF8
- UTF-8 AKA Unicode covers almost all of the characters and symbols in the world

# Character encodings

- Check an HTML file for its reported encoding:

```
<!DOCTYPE html>
  <head>
    <meta charset="UTF-8">
    ...
```

- Check any file for its encoding on the file system

```
$ file index.html
index.html: HTML document text, ASCII text
$ file --mime index.html
index.html: text/html; charset=us-ascii
```

- UTF8 is a superset of ASCII
- UTF-8 is identical to ASCII for the values from 0 to 127

# Character encodings and Python

- Strings in Python2 are ASCII encoded (default)
- Strings in Python3 are Unicode encoded (default)

- Much of the web is Unicode
  - Put another way: lots of the HTML out there is UTF-8 encoded
  - Why?  Because the web is global!

- Unicode support was one reason for Python 3

# HEAD request

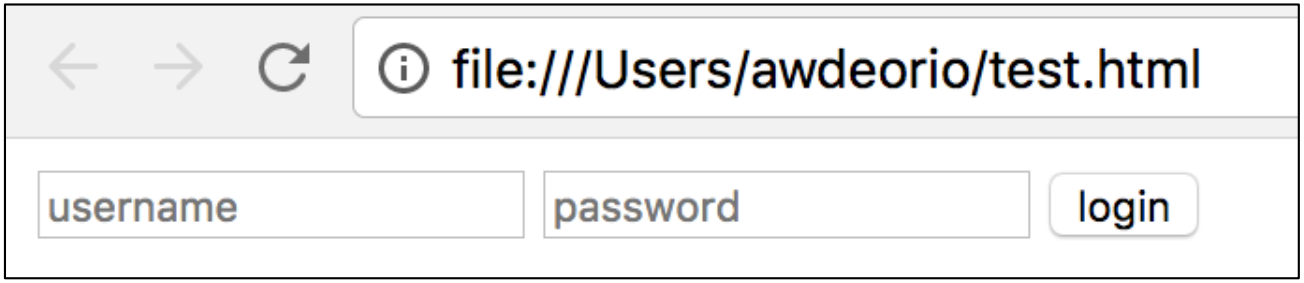
- HEAD requests are useful for checking if a page has changed
- Good for caching stuff that doesn't change much and updating it
- Need a page that doesn't change much for this example

```
$ curl --head --verbose
http://cse.eecs.umich.edu/eecs/images/CSE-Logo-
Mobile.png > CSE-Logo-Mobile.png
* Connected to cse.eecs.umich.edu
> GET /eecs/images/CSE-Logo-Mobile.png HTTP/1.1
...
< HTTP/1.1 200 OK
< Content-Type: image/png
...
< Last-Modified: Thu, 28 May 2015 13:40:55 GMT
```

# POST request

- POST request sends data from the client to the server
- Commonly used with HTML forms

```
<html>
<body>
  <form action="" method="post" enctype="multipart/form-data">
    <input type="text" name="username" placeholder="username"/>
    <input type="password" name="password" placeholder="password"/>
    <input type="submit" value="login"/>
  </form>
</body>
</html>
```



# POST request

- No POST requests needed in project 1
- Example from project 2

```
$ curl \
    --form 'username=awdeorio' \
    --form 'password=password' \
    --form 'submit=login' \
    localhost:8000/accounts/login/
```

# HTTP versions

- See the HTTP version in the request and response

```
 $ curl --verbose http://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu
> GET / HTTP/1.1
...
< HTTP/1.1 200 OK
```

- Three versions:
  - HTTP/1.0 (old)
  - **HTTP/1.1 (common)**
  - HTTP/2 (new)

# HTTP/1.0 .vs HTTP/1.1

- How many TCP connections?

```
<html>
<body>
  <p>Block M</p>
  <img src="http://cse.eecs.umich.edu/eecs/images/CSE2017-hi-rev.png">
  <img src="http://cse.eecs.umich.edu/eecs/images/CSE2017-rev.png">
</body>
</html>
```

- HTTP/1.0: 3
  - Load HTML, first `img`, second `img`
- HTTP/1.1: 1
  - Reuse one HTTP connection

# HTTP/2

- HTTP 2 supported by ~20% of the web
  - As of November 2017
- Methods, status codes, etc. same as HTTP/1.1
- One new feature: server push
  - Server supplies data it knows a web browser will need to render a web page, without waiting for the browser to examine the first response.
  - Example: images from previous slide