Lecture 1
Web Basics

EECS 485

January 6, 2009

(some slides due to Dan Weld)

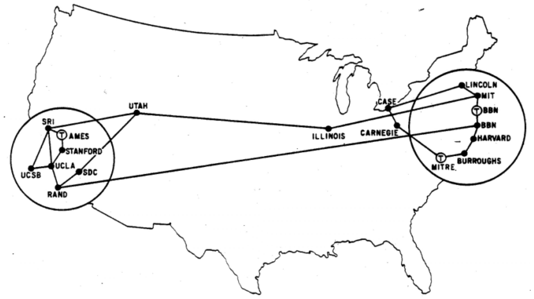
Web Essentials

- A brief history
- Transfer
  - Networking and OS basics
  - HTTP
- Content
  - HTML
  - Encoding
  - Dynamic pages

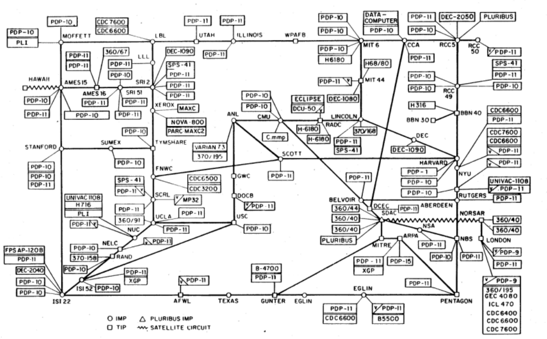
Ancient History

- Pre-history
  - Dewey Decimal system, library science
- 1960: Ted Nelson → Xanadu
  - Hypertext vision of WWW
  - Focus on copyright, consistent (bidirectional) links, versioning
    - Why did it fail?
  - *"Trying to fix HTML is like trying to graft arms and legs onto a hamburger"* -- Ted Nelson

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



ARPANET LOGICAL MAP, MARCH 1977



(PLEASE NOTE THAT WHILE THIS MAP SHOWS THE HOST POPULATION OF THE NETWORK ACCORDING TO THE BEST INFORMATION OBTAINABLE, NO CLAIM CAN BE MADE FOR ITS ACCURACY)

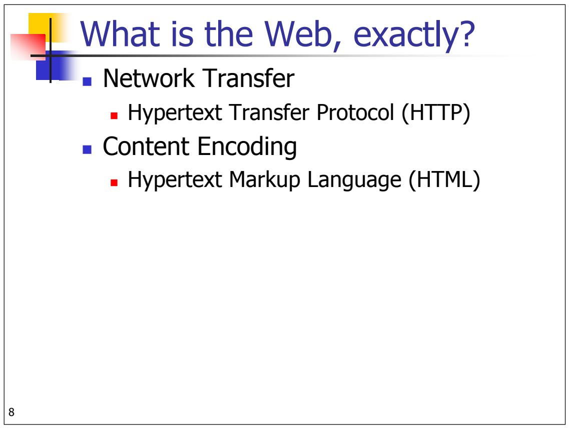
NAMES SHOWN ARE IMP NAMES, NOT (NECESSARILY) HOST NAMES

Internet Ramps Up

- 1983 ARPAnet uses TCP/IP; design of DNS; 1000 hosts on ARPAnet
- 1985 **`symbolic.com`** 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

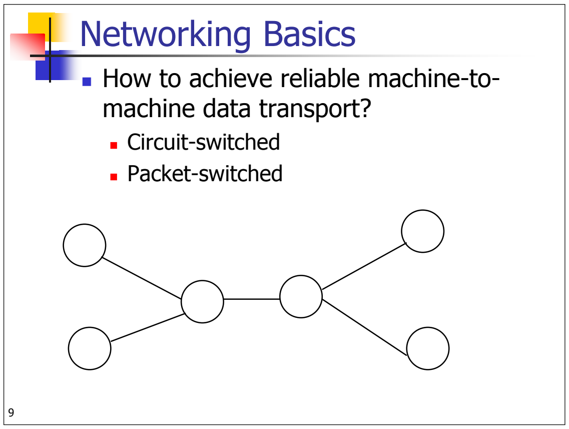
7



What is the Web, exactly?

- Network Transfer
  - Hypertext Transfer Protocol (HTTP)
- Content Encoding
  - Hypertext Markup Language (HTML)

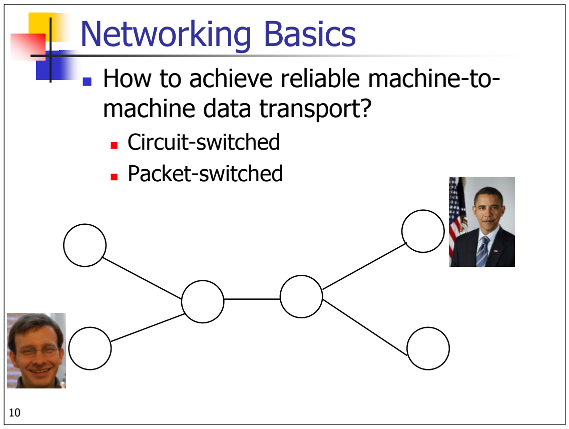
8



Networking Basics

- How to achieve reliable machine-to-machine data transport?
  - Circuit-switched
  - Packet-switched

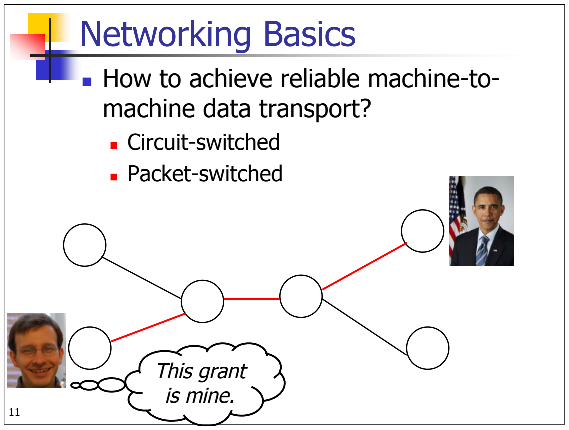
9



Networking Basics

- How to achieve reliable machine-to-machine data transport?
  - Circuit-switched
  - Packet-switched

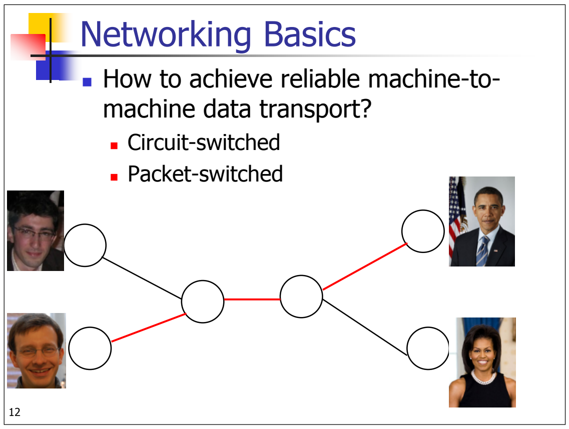
10



Networking Basics

- How to achieve reliable machine-to-machine data transport?
  - Circuit-switched
  - Packet-switched

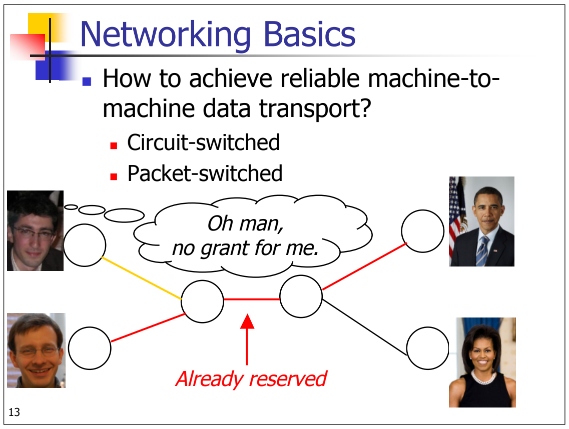
11



Networking Basics

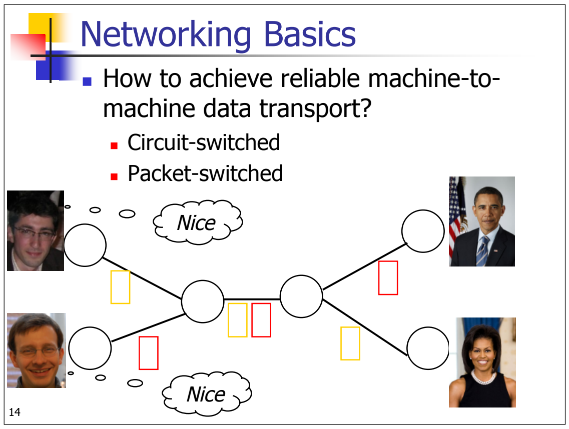
- How to achieve reliable machine-to-machine data transport?
  - Circuit-switched
  - Packet-switched

12



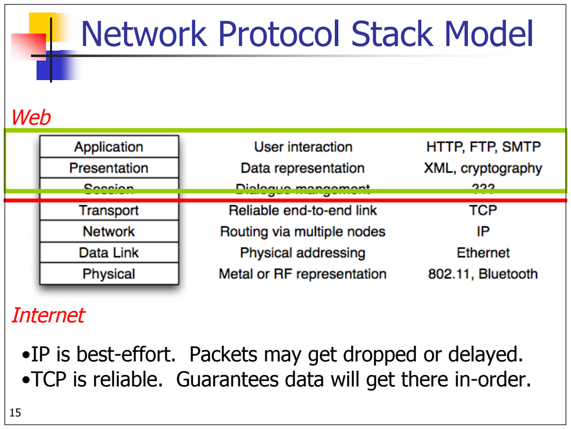
Networking Basics

- How to achieve reliable machine-to-machine data transport?
  - Circuit-switched
  - Packet-switched



Networking Basics

- How to achieve reliable machine-to-machine data transport?
  - Circuit-switched
  - Packet-switched



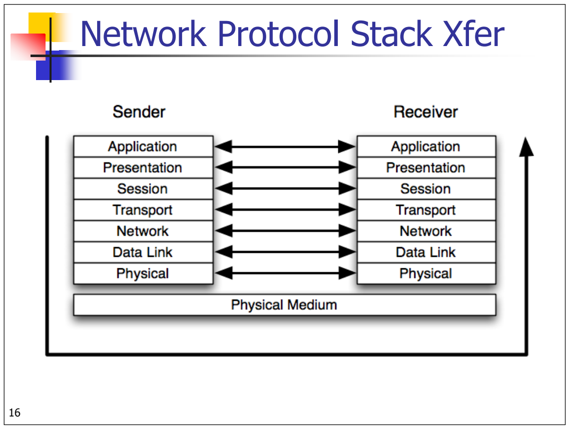
Network Protocol Stack Model

Web

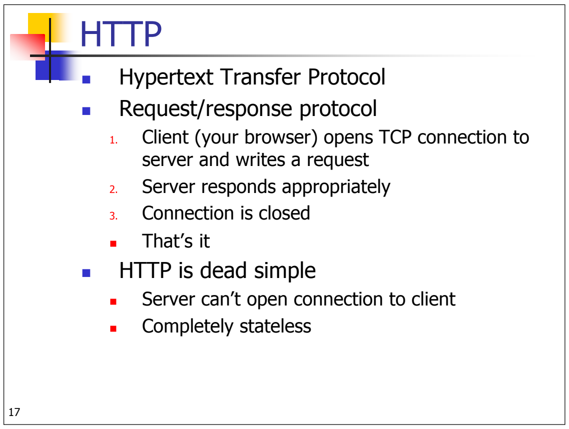
Application	User interaction	HTTP, FTP, SMTP
Presentation	Data representation	XML, cryptography
Session	Dialogue mangement	???
Transport	Reliable end-to-end link	TCP
Network	Routing via multiple nodes	IP
Data Link	Physical addressing	Ethernet
Physical	Metal or RF representation	802.11, Bluetooth

Internet

•IP is best-effort. Packets may get dropped or delayed.
•TCP is reliable. Guarantees data will get there in-order.

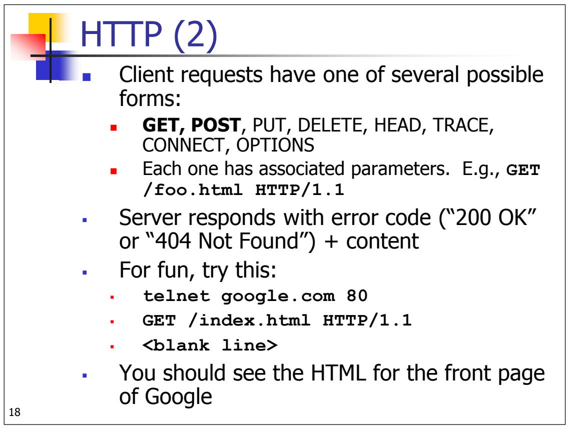


Network Protocol Stack Xfer



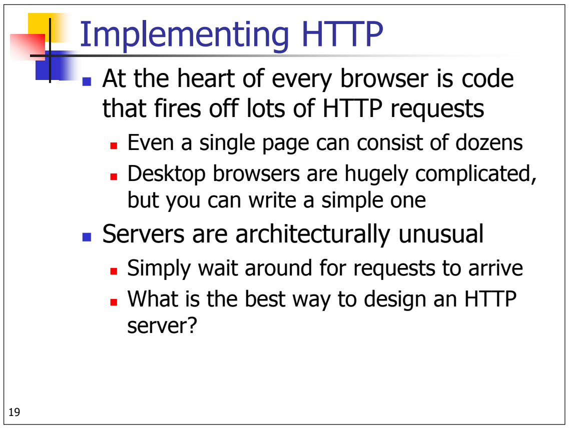
HTTP

- Hypertext Transfer Protocol
- Request/response protocol
  1. Client (your browser) opens TCP connection to server and writes a request
  2. Server responds appropriately
  3. Connection is closed
  - That's it
- HTTP is dead simple
  - Server can't open connection to client
  - Completely stateless



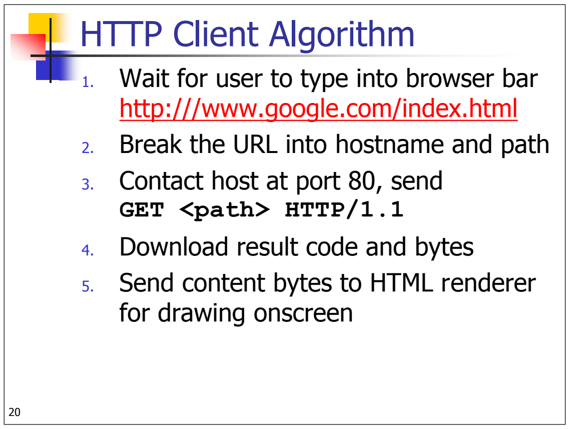
HTTP (2)

- Client requests have one of several possible forms:
  - **GET, POST**, PUT, DELETE, HEAD, TRACE, CONNECT, OPTIONS
  - Each one has associated parameters. E.g., **`GET /foo.html HTTP/1.1`**
- Server responds with error code ("200 OK" or "404 Not Found") + content
- For fun, try this:
  - **`telnet google.com 80`**
  - **`GET /index.html HTTP/1.1`**
  - **`<blank line>`**
- You should see the HTML for the front page of Google



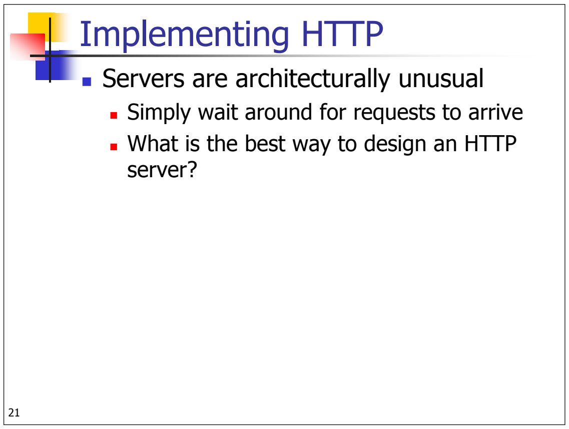
Implementing HTTP

- At the heart of every browser is code that fires off lots of HTTP requests
  - Even a single page can consist of dozens
  - Desktop browsers are hugely complicated, but you can write a simple one
- Servers are architecturally unusual
  - Simply wait around for requests to arrive
  - What is the best way to design an HTTP server?



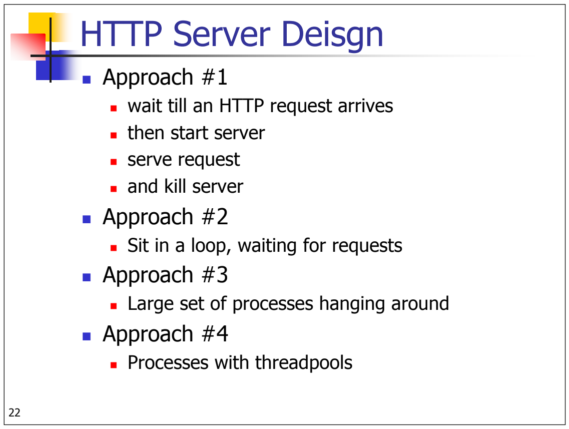
HTTP Client Algorithm

1. Wait for user to type into browser bar http:///www.google.com/index.html
2. Break the URL into hostname and path
3. Contact host at port 80, send **`GET <path> HTTP/1.1`**
4. Download result code and bytes
5. Send content bytes to HTML renderer for drawing onscreen



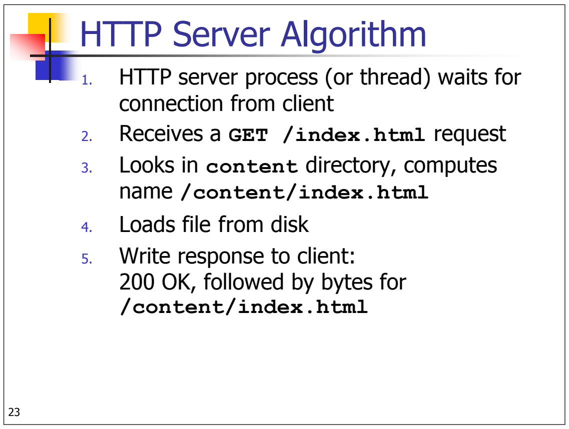
Implementing HTTP

- Servers are architecturally unusual
  - Simply wait around for requests to arrive
  - What is the best way to design an HTTP server?



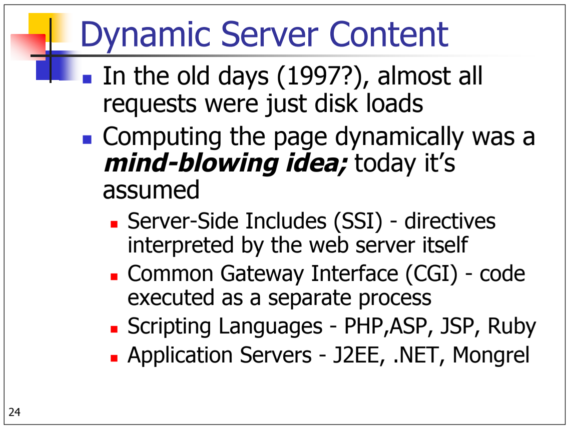
HTTP Server Deisgn

- Approach #1
  - wait till an HTTP request arrives
  - then start server
  - serve request
  - and kill server
- Approach #2
  - Sit in a loop, waiting for requests
- Approach #3
  - Large set of processes hanging around
- Approach #4
  - Processes with threadpools



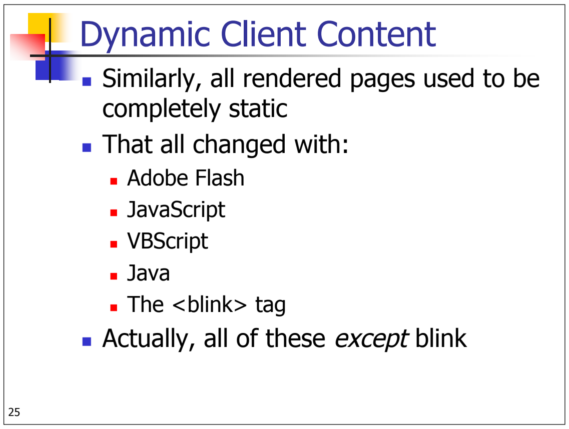
HTTP Server Algorithm

1. HTTP server process (or thread) waits for connection from client
2. Receives a **`GET /index.html`** request
3. Looks in **`content`** directory, computes name **`/content/index.html`**
4. Loads file from disk
5. Write response to client:
   200 OK, followed by bytes for
   `/content/index.html`



Dynamic Server Content

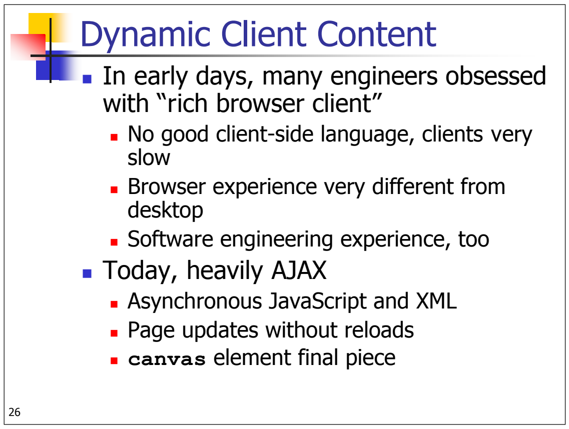
- In the old days (1997?), almost all requests were just disk loads
- Computing the page dynamically was a ***mind-blowing idea;*** today it's assumed
  - Server-Side Includes (SSI) - directives interpreted by the web server itself
  - Common Gateway Interface (CGI) - code executed as a separate process
  - Scripting Languages - PHP,ASP, JSP, Ruby
  - Application Servers - J2EE, .NET, Mongrel



Dynamic Client Content

- Similarly, all rendered pages used to be completely static
- That all changed with:
  - Adobe Flash
  - JavaScript
  - VBScript
  - Java
  - The <blink> tag
- Actually, all of these *except* blink

25



Dynamic Client Content

- In early days, many engineers obsessed with "rich browser client"
  - No good client-side language, clients very slow
  - Browser experience very different from desktop
  - Software engineering experience, too
- Today, heavily AJAX
  - Asynchronous JavaScript and XML
  - Page updates without reloads
  - **`canvas`** element final piece

26