IIT Madras
BSc Degree

Copyright and terms of use

"Apps"

What is an "App" anyway?

"An app is computer software, or a program, most commonly a small, specific one used for mobile devices. The term app originally referred to any mobile or desktop application, but as more app stores have emerged to sell mobile apps to smartphone and tablet users, the term has evolved to refer to small programs that can be downloaded and installed all at once."

- *Src: Techopedia*
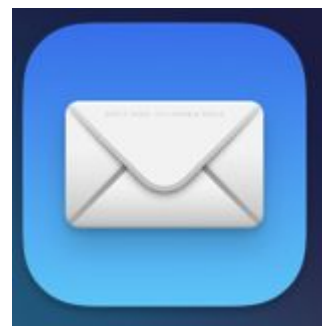
Desktop Apps

- Usually standalone
  - Editors / Word processors
  - Web browser
  - Mail
- Often work offline
  - Local data storage
  - Possible network connection
- Software Dev Kits (SDK)
  - Custom frameworks
  - **OS specific**
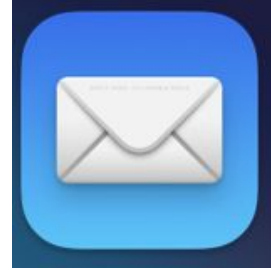
Mobile Apps

- Targeted at mobile platforms: phones/tablets
- Constraints
  - Limited screen space
  - User interaction (touch, audio, camera)
  - Memory / processing
  - Power
- Frameworks
  - OS specific
  - Cross-platform
- Network!
  - Usually network oriented
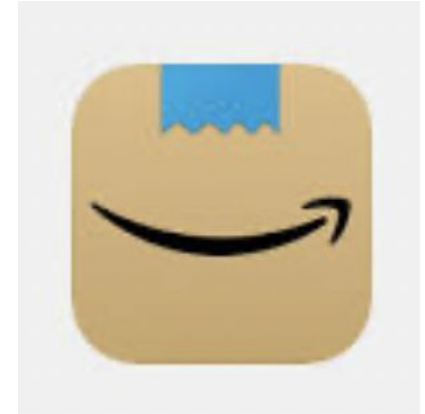
Web Apps

- The **Platform**
- Works across OS, device: create a common base
- Heavily network dependent
  - Workarounds for offline processing

Main focus of this course

Components

- Storage
- Computation
- Presentation

Example: email client

- Storage:
  - Where are the emails stored?
  - How are they stored on the server? File formats etc.
- Compute:
  - Indexing of emails
  - Searching
- Presentation
  - Display list of mails
  - Rendering/display of individual mails

Platforms

- Desktop
- Mobile
- Web-based
- Embedded

Platform features

- Desktop
    - Keyboard, mouse, video
    - Desktop paradigm - folders, files, documents
- Mobile
    - Touchscreen
    - Voice, tilt, camera interfaces
    - Small self-contained apps
- Web-based
    - Datacenter storage - persistent
    - Cloud: access anywhere, multi-device
- Embedded
    - Single function, limited scope

Architectures

- Client-Server
- Distributed - peer-to-peer
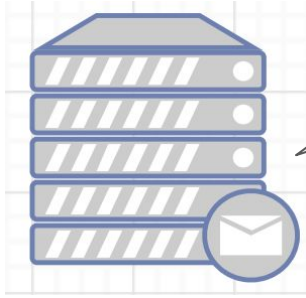
Server:
- Stores data
- Provides data on demand
- May perform computations
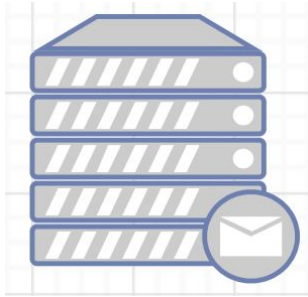
Clients:
- End users
- Request data
- User interaction, display

Network

Network:
- Connects client to server
- Can be local
- Data pipe - no alterations
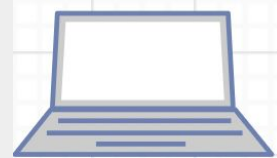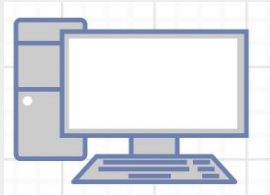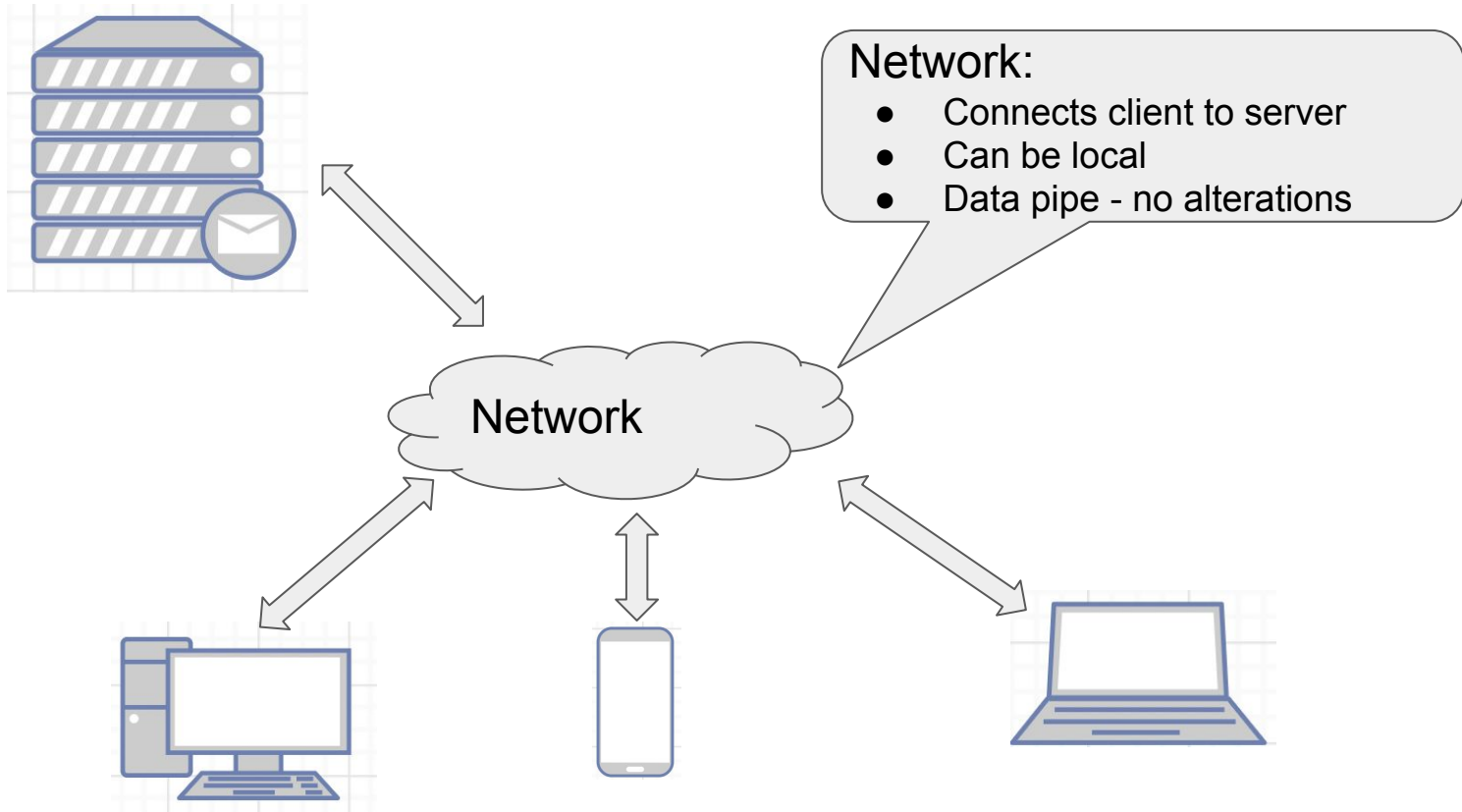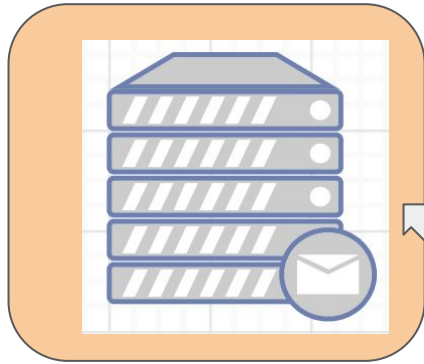
Client-Server Model
- Explicit server(s)
- Explicit client(s)
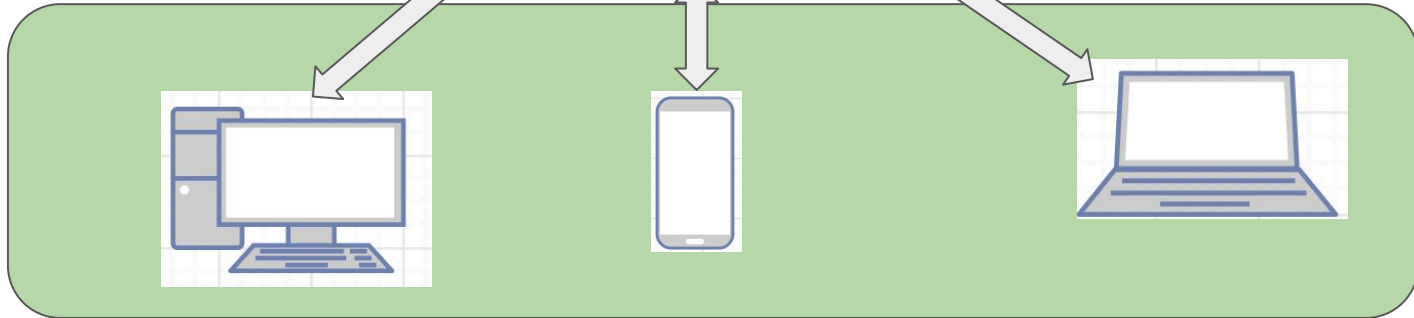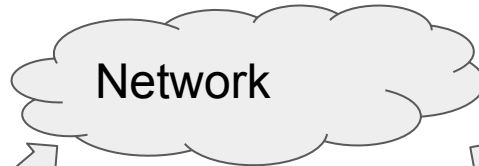
Network

Client Server Model

- Explicit differentiation between clients and servers
- Local systems:
  - both client and server on same machine - local network / communication
  - conceptually still a networked system
- Machine clients
  - eg. Software / antivirus updaters
  - Need not have user interaction
- Variants
  - Multiple servers, single queue, multiple queues, load balancing frontends

Examples:

- Email
- Databases
- WhatsApp / messaging
- Web browsing
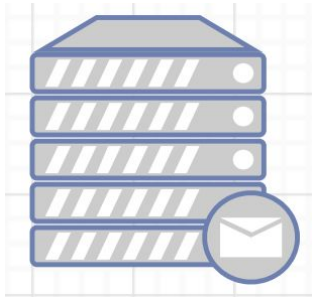
Client?
Server?

Data can flow both ways!

Network
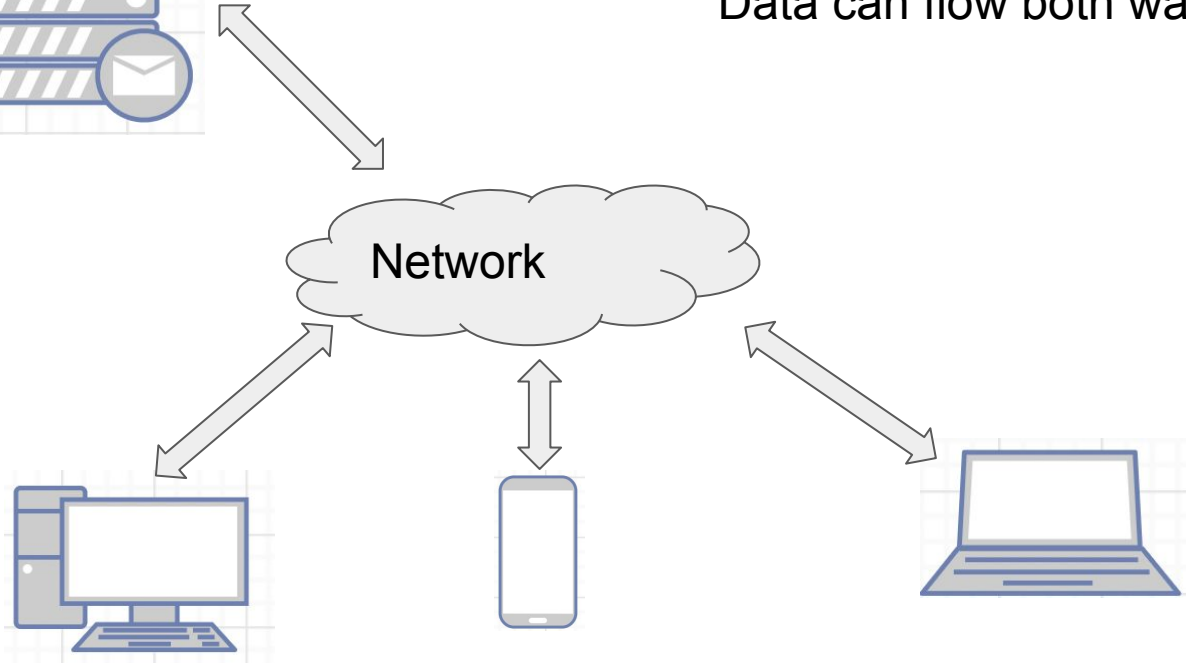
Distributed (Peer-to-Peer) Model

- All peers are considered "equivalent"
  - But some peers may be more equal than others…
- Error tolerance
  - Masters / introducers
  - Election / re-selection of masters on failure
- Shared information

Examples:

- Bittorrent
- Blockchain-based systems
- IPFS, Tahoe (distributed file systems)

Software Architecture Patterns

- MVC
- MVA
- MVP
- HMVC
- MVVM
- . . .

What is a design pattern?

"A general, reusable solution to a commonly occurring problem within a given context in software design." -- *Src: Wikipedia*

- Experienced designers observe "patterns" in code
- Reusing these patterns can make design and development faster
- **Guide** the design and thought process

User: I want to check email
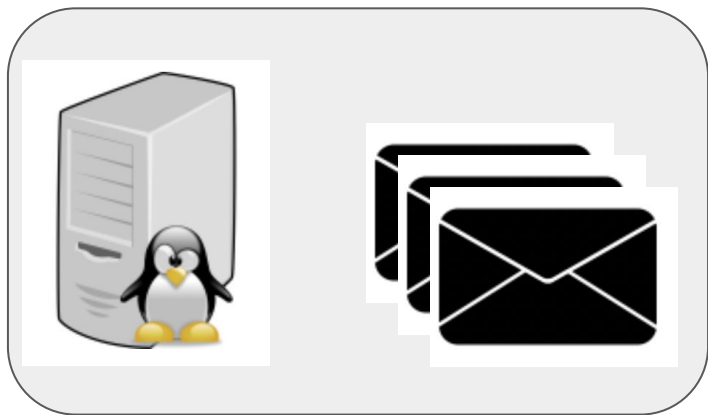
Model: Store emails on server, index, ready to manipulate

View: Display list of emails; Read individual emails
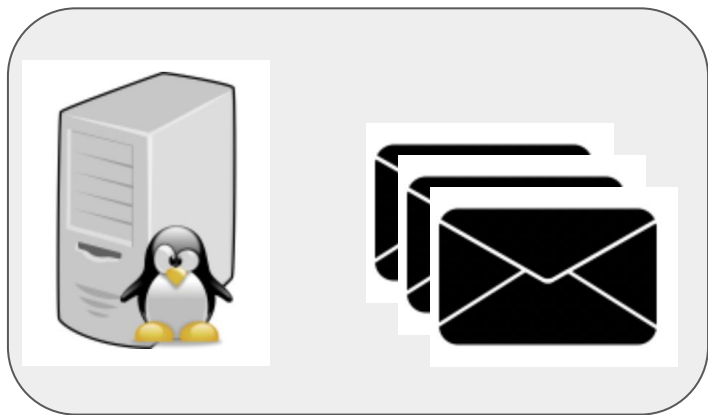
Model: Store emails on server, index, ready to manipulate

View: Display list of emails; Read individual emails

Controller: Sort emails; delete; archive

M-V-C paradigm

- Model:
    - Core data to be stored for the application
    - Databases; indexing for easy searching, manipulation;
- View:
    - User-facing side of application
    - Interfaces for finding information, manipulating
- Controller:
    - "Business logic" - how to manipulate data

Hardly new: origins in Smalltalk language: 1979

M-V-C paradigm

- Model:
  - Core data to be stored for the application
  - Databases; indexing for easy searching, manipulation;
- View:
  - User-facing side of application
  - Interfaces for finding information, manipulating
- Controller:
  - "Business logic" - how to manipulate data

Hardly new: origins in Smalltalk language: 1979

User
uses **controller** ->
to manipulate **model** ->
that updates **view** ->
that user sees

Other design patterns

- Model-View-Adapter
- Model-View-Presenter
- Model-View-Viewmodel
- Hierarchical MVC
- Presentation-Abstraction-Control
- . . .

Each has its uses, but fundamentals are very similar

Focus of this course

- Platform:
  - Web-based
- Architecture:
  - Client-server
- Software architecture:
  - Model-View-Controller

How to build apps (or applications) that are web-based with central servers, and use hypertext markup to control and manipulate display

The "Web"

Why the "Web"

- Platform of choice for this course
- Generic - works across operating systems, hardware architectures
  - Cross-platform operating system?
- Built on sound underlying principles
- Worth understanding
  - Constraints: what can and cannot be done (easily)
  - Costs: storage, network, device sizing, datacenter

Historical background

- Telephone networks ~ 1890+
    - Circuit switched - allow A to talk to B, through complex switching network
    - Physical wires tied up for duration of call even if nothing said

Historical background

- Telephone networks ~ 1890+
  - Circuit switched - allow A to talk to B, through complex switching network
  - Physical wires tied up for duration of call even if nothing said
- Packet switched networks ~ 1960s
  - Wires occupied only when data to be sent - more efficient use
  - Data instead of Voice

Historical background

- Telephone networks ~ 1890+
  - Circuit switched - allow A to talk to B, through complex switching network
  - Physical wires tied up for duration of call even if nothing said
- Packet switched networks ~ 1960s
  - Wires occupied only when data to be sent - more efficient use
  - Data instead of Voice
- ARPANet - 1969
  - Node-to-node network
  - Mostly university driven
- Others:
  - IBM SNA, Digital DECNet, Xerox Ethernet, …

Historical background

- Protocol:
  - How to format packets; place them on wire; headers/checksums;
  - Each network had its own protocol

Historical background

- Protocol:
    - How to format packets; place them on wire; headers/checksums;
    - Each network had its own protocol
- "Inter" network
    - How to communicate between different network protocols?
    - Or replace them with a single "inter"net protocol

Historical background

- Protocol:
  - How to format packets; place them on wire; headers/checksums;
  - Each network had its own protocol
- "Inter" network
  - How to communicate between different network protocols?
  - Or replace them with a single "inter"net protocol
- IP: Internet protocol - 1983
  - Defines headers, packet types, interpretation
  - Can be carried over different underlying networks: ethernet, DECnet, PPP, SLIP
- TCP: Transmission Control Protocol - 1983
  - Establish reliable communications - retry; error control
  - Automatically scale and adjust to network limits

Historical background

- Domain Names ~ 1985
  - Use names instead of IP addresses
  - Easy to remember - .com revolution still in the future

Historical background

- Domain Names ~ 1985
    - Use names instead of IP addresses
    - Easy to remember - .com revolution still in the future
- HyperText ~ 1989+
    - Text documents to be "served"
    - Formatting hints inside document to "link" to other documents - HyperText

Historical background

- Domain Names ~ 1985
  - Use names instead of IP addresses
  - Easy to remember - .com revolution still in the future
- HyperText ~ 1989+
  - Text documents to be "served"
  - Formatting hints inside document to "link" to other documents - HyperText

The World Wide Web

Where are we now?

- Original web limited:
  - static pages
  - complicated executable interfaces
  - limited styling
  - browser compatibility issues

Where are we now?

- Original web limited:
  - static pages
  - complicated executable interfaces
  - limited styling
  - browser compatibility issues
- Web 2.0 ~ 2004+
  - dynamic pages - generate on the fly
  - HTTP as a transport mechanism - binary data; serialized objects
  - client side computation and rendering
  - platform agnostic operating system

Digging Deeper

Or - how does the web work?

- What is a server?
- What is a transport protocol?
- Impact on performance:
  - Network
  - Server compute resources
  - Storage requirements
  - Client compute resources

Web Server

- Any old computer with a network connection
- Software:
  - Listen for incoming network connections on a fixed port
  - Respond in specific ways
  - Opening network connections, ports etc already known to OS
- Protocol:
  - What should client ask server
  - How should server respond to client
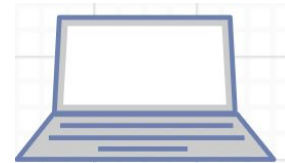
HTTP

- HyperText:
  - Regular text document
  - Contains "codes" inside that indicate special functions - how to "link" to other documents
- HyperText Transfer Protocol
  - Largely text based: client sends requests, server responds with hypertext document

Simplest server

```
while true; do
    echo -e "HTTP/1.1 200 OK\n\n $(date)" |
        nc -l localhost 1500;
done
```

General web server

- Listen on a fixed port
- On incoming request, run some code and return a result
  - Standard headers to be sent as part of result
  - Output can be text or other format - MIME

Typical request

```
GET / HTTP/1.1
Host: localhost:1500
User-Agent: curl/7.64.1
Accept: */*
```

Viewing responses with curl

- curl, wget etc - simple command line utilities
- Can perform full HTTP requests
- Verbose output includes all headers
- Very useful for debugging!

```
curl -v http://localhost:1500
```

[curl simple web server](#)

```
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 1500 (#0)
> GET / HTTP/1.1
> Host: localhost:1500
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
* no chunk, no close, no size. Assume close to signal end
<
 Thu Jun 17 08:14:55 IST 2021
```

[curl simple web server](#)

```
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 1500 (#0)
> GET / HTTP/1.1
> Host: localhost:1500
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
* no chunk, no close, no size. Assume close to signal end
<
 Thu Jun 17 08:14:55 IST 2021
```

[curl simple web server](#)

```
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 1500 (#0)
> GET / HTTP/1.1
> Host: localhost:1500
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
* no chunk, no close, no size. Assume close to signal end
<
 Thu Jun 17 08:14:55 IST 2021
```

Protocol

- What is a protocol
- Examples for HTTP

Protocol

- Both sides agree on how to talk
  - Walk into a room, greet each other, shake hands, sit down, chat about weather…
- Server expects "requests"
  - Nature of request
  - Nature of client
  - Types of results client can deal with
- Client expects "responses"
  - Ask server for something
  - Convey what you can accept
  - Read result and process

HyperText Transfer Protocol: HTTP

- Primarily text based:
- Requests specified as "GET", "POST", "PUT" etc.
  - Headers can be used to convey acceptable response types, languages, encoding
  - Which host to connect to (if multiple hosts on single server)
- Response headers
  - convey message type, data
  - cache information
  - status codes: 404 not found...

Use cases

- GET: simple requests, search queries etc.
- POST: more complex form data, large text blocks, file uploads
- PUT/DELETE/…
    - Rarely used in Web 1.0
    - Extensively used in Web 2.0
    - Basis of most APIs - REST, CRUD

Another web server

```
python -m http.server
```

- Serve files from local folder
- Understands basic HTTP requests
- Gives more detailed headers and responses

[GET /](#)

```
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8000 (#0)
> GET / HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.64.1
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Server: SimpleHTTP/0.6 Python/3.9.2
< Date: Thu, 17 Jun 2021 03:11:15 GMT
< Content-type: text/html
< Content-Length: 31
< Last-Modified: Thu, 17 Jun 2021 03:10:18 GMT
<
<h1>Hello world</h1>
Hi there!
* Closing connection 0
```

GET /serve.sh

```
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8000 (#0)
> GET /serve.sh HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.64.1
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Server: SimpleHTTP/0.6 Python/3.9.2
< Date: Thu, 17 Jun 2021 03:11:36 GMT
< Content-type: application/x-sh
< Content-Length: 95
< Last-Modified: Wed, 16 Jun 2021 17:30:25 GMT
<
#!/bin/bash

while true; do echo -e "HTTP/1.1 200 OK\n\n $(date)" | nc -l localhost 1500; done
* Closing connection 0
```

Performance

- How fast can a site be?
- What limits performance?
- Basic observations

Latency

- Speed of light: 3e8 m/s in vacuum, ~ 2e8 m/s on cable
  - ~ 5ns / m => ~ 5ms for 1000km

Latency

- Speed of light: 3e8 m/s in vacuum, ~ 2e8 m/s on cable
  - ~ 5ns / m => ~ 5ms for 1000km
- Data center 2000km away
  - One way - request - ~ 10ms
  - Round-trip 20ms

Latency

- Speed of light: 3e8 m/s in vacuum, ~ 2e8 m/s on cable
  - ~ 5ns / m => ~ 5ms for 1000km
- Data center 2000km away
  - One way - request - ~ 10ms
  - Round-trip 20ms
- Max 50 requests / second

Response size

- Response = 1KB of text (headers, HTML, CSS, JS)

Response size

- Response = 1KB of text (headers, HTML, CSS, JS)
- Network connection = 100 Mbps
  - ~ 10 MBytes/s

Response size

- Response = 1KB of text (headers, HTML, CSS, JS)
- Network connection = 100 Mbps
  - ~ 10 MBytes/s
- ~ 10,000 requests / second

```
* TCP_NODELAY set
* Connected to www.google.com (142.250.77.164) port 80 (#0)
> GET / HTTP/1.1
> Host: www.google.com
> User-Agent: curl/7.64.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 17 Jun 2021 03:17:49 GMT
< Expires: -1
< Cache-Control: private, max-age=0
< Content-Type: text/html; charset=ISO-8859-1
< P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
< Server: gws
< X-XSS-Protection: 0
< X-Frame-Options: SAMEORIGIN
< Set-Cookie: 1P_JAR=2021-06-17-03; expires=Sat, 17-Jul-2021 03:17:49 GMT; path=/; domain=.google.com; Secure
< Set-Cookie:
NID=217=ofdnZ7Yfq3cyvrbHPN3MSQvk4MaWpSp-MMS6NRbuAxaD4mj_4Brf6vWQSQY7eCIletUo78OxfhpomyizbaEmf2noWfhwoGy759xq4
TMhcQC7im0JMsBRff2uQL_gREsRs7tjH2ZSt662imXk-1AGZ8y-TJBwmPu2DWDHotZqP4; expires=Fri, 17-Dec-2021 03:17:49 GMT;
path=/; domain=.google.com; HttpOnly
< Accept-Ranges: none
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
<
{ [13448 bytes data]
100 15226    0 15226    0     0   144k        0 --:--:-- --:--:-- --:--:--  144k
* Connection #0 to host www.google.com left intact
* Closing connection 0
```

Google response

- Headers - ~ 100B
- Content: 144kB
- Approx 60,000 requests per second (maybe)

Google response

- Headers - ~ 100B
- Content: 144kB
- Approx 60,000 requests per second (maybe)

~ 80 Gbps bandwidth

Memory - YouTube

- One python HTTP server process: ~ 6MB (measured)
- Multiple parallel requests: multiple processes
  - eg. YouTube will have long running server processes
- 2016 Presidential debate in US:
  - > 2 Million concurrent viewers

Memory - YouTube

- One python HTTP server process: ~ 6MB (measured)
- Multiple parallel requests: multiple processes
  - eg. YouTube will have long running server processes
- 2016 Presidential debate in US:
  - > 2 Million concurrent viewers

2 M x 6 MB => 12 TB RAM

Storage - Google

- Index 100s of billions of web pages (funny cat videos?)
- Cross-reference, pagerank
- Total index size estimate:
  - 100,000,000 Gigabytes => 100 Petabytes
- Storage?
- Retrieval

Summary

- The Web is a useful device/OS agnostic platform for apps
- Built on HTTP for transport, HTML and related tech for presentation
- Servers trivial at most basic
- Scaling requires careful design