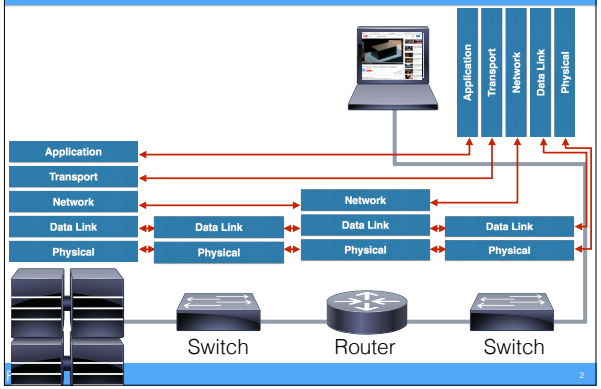
FIT1047 Week 8

Networks: Application Layer

MONASH University

Layers of Abstraction



Recall that devices communicate via different *protocols*, on several *layers* of abstraction. Today we will look at application layer protocols.

Architectures

How do we **split up** the work between the server and the client?

How do we **choose** the right architecture for a particular application?

Application Architectures

Presentation logic
The user interface. Controls the application.

Application / business logic
Defines what the application does.

Data access logic
Defines how the application manages its data.

Data storage
Where the data is kept, e.g. files or data bas

Who does what?

Server-based Architecture

"dumb" terminal

Client sends keystrokes to the server, displays text according to server's instructions.

Problems:
Server can become a bottleneck. Upgrade expensive and "lumpy".

Presentation logic
Application / business logic
Data access logic
Data storage

Oldest architecture developed in the 1960s. Now predominantly used for **accessing UNIX-based servers** through terminal emulators such as **ssh**.
Almost all processing done by server.
Lumpy upgrade: big expensive server must be replaced, need to anticipate future demand.

Client-based Architecture

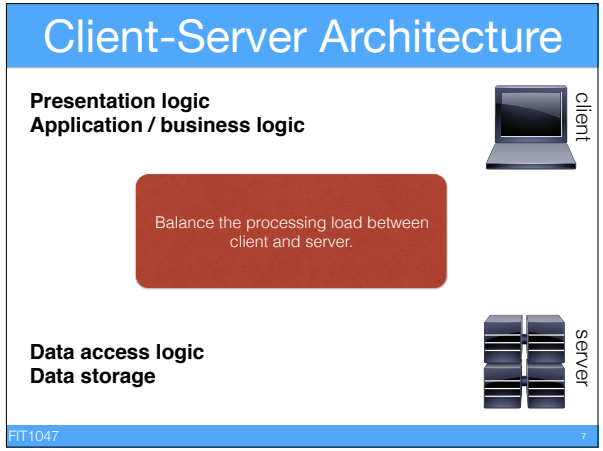
Presentation logic
Application / business logic
Data access logic

All the logic is performed by the client. The server stores the data.

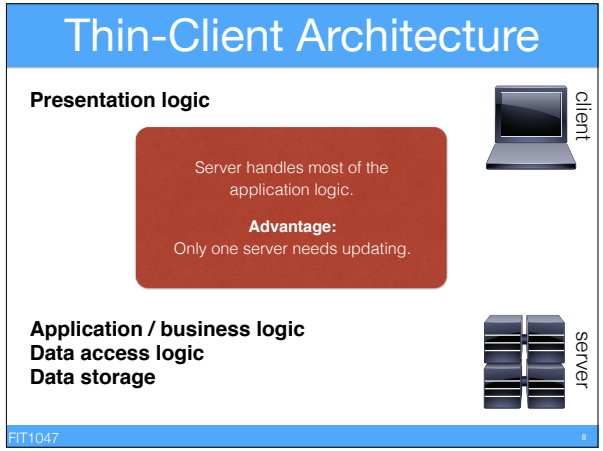
Problems:
All data must travel back and forth between server and client.

Data storage

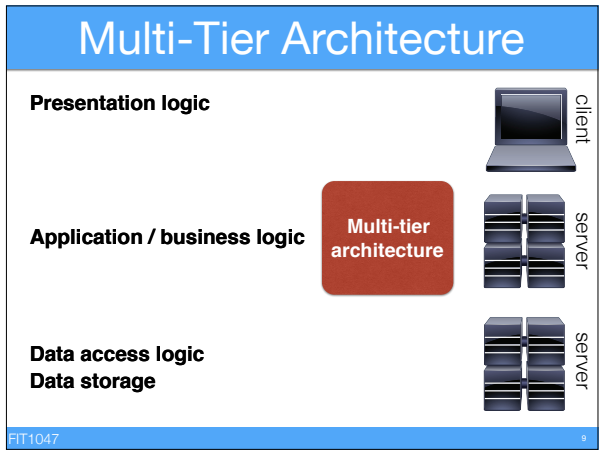
Developed in late 1980s. Example: using a word processing application over a network file system (e.g. Windows file sharing).
Performance problem: E.g. when making a data base query, the **entire data base** must be transferred to the client. Puts stress on the network.



Developed in late 1990s. E.g. client software accepts user requests, performs application logic, sends requests to server. Server performs data base access and sends result to client. Client presents the data to the user. Examples: Email, WWW.



Example: web applications (e.g. Google Mail). Classification is not easy: often presentation logic and some application logic is *stored* on server but *executed* by the client (e.g. in JavaScript).



Advantages: Better **load balancing**, distributing the load across several servers. Only high-demand servers require upgrading.
Disadvantages: High load on (internal) network between the servers, difficult to test.

Peer-To-Peer Architecture

Presentation logic
Application / business logic
Data access logic
Data storage

All computers act as both clients and servers.

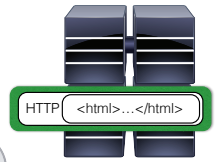
Use local logic to access data stored on another computer.

Presentation logic
Application / business logic
Data access logic
Data storage

Became popular around 1999 (Napster). Today's applications: BitTorrent (file sharing), but also Skype or Bitcoin use P2P principles.

World Wide Web

What is HTTP?
What is HTML?

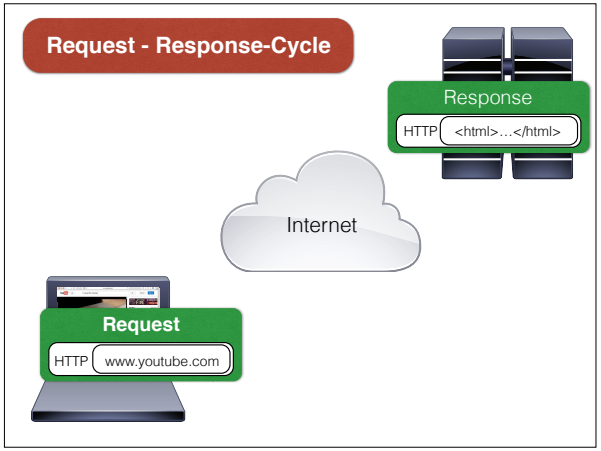


HyperText Transfer Protocol (HTTP)

- Defines how web **browsers** talk to **web servers**
- Invented in 1989 by Tim Berners-Lee at CERN:
  "How will we ever keep track of such a large project?"
- Based on two innovative ideas:
  - **Hypertext:**
    A document containing links to other documents
  - **Uniform Resource Locators (URLs):**
    A standard for identifying links to other documents
- HTTP defines **how** documents are requested and transferred

CERN: European Organisation for Nuclear Research.

http://www.w3.org/History/1989/proposal.html



Cycle is started by clicking on a link or typing a URL into the browser.

Browser generates a (HTTP) request. Response is e.g. a HTML page that contains other *assets* such as images, CSS, JavaScript. Client then requests all these assets in turn.

Basic HTTP session

client:

```
GET /~guidot/test.html HTTP/1.1
Host: www.csse.monash.edu
```

Request line

Req. header

server:

```
HTTP/1.1 200 OK
Date: Thu, 05 Mar 2015 08:30:48 GMT
Server: Apache/1.3.26 (Unix)
Transfer-Encoding: chunked
Content-Type: text/html
```

Response status

Response header

```
<html>
<body>
  <h1>Guido Tack</h1>
  <img src="images/guido3.jpg">
</body>
</html>
```

Response body

Request: command ("method"), URL, protocol version.

Response status: version, status code, reason.

Response header: information about the server.

Response body: the requested document (here HTML page).

Basic HTTP session

```
<html>
<body>
  <h1>Guido Tack</h1>
  <img src="images/guido3.jpg">
</body>
</html>
```

client:
```
GET /~guidot/images/guido3.jpg HTTP/1.1
Host: www.csse.monash.edu
```


Client analyses (“parses”) HTML page, finds link to image URL, requests image.

Basic HTTP session

client:
```
GET /~guidot/images/guido3.jpg HTTP/1.1
Host: www.csse.monash.edu
```

server:
```
HTTP/1.1 200 OK
Date: Thu, 05 Mar 2015 08:31:23 GMT
Server: Apache/1.3.26 (Unix)
Last-Modified: Tue, 20 Nov 2012 03:29:22 GMT
Accept-Ranges: bytes
Content-Length: 15681
Content-Type: image/jpeg

JFIFHH@ICC_PROFILE0appl mntrRGB
…
```


HTTP Methods

- `GET`:
  Retrieve specified URL from server
- `HEAD`:
  Retrieve **only header** for specified URL
- `POST`:
  Add data specified in request body to specified URL

  E.g. add a message to a web forum, or an item to a shopping cart. Also retrieves document.
- Other methods (`PUT`, `DELETE`, `OPTIONS`…) less common

Full HTTP Request

```
POST /~guidot/test_form.php HTTP/1.1
Host: www.csse.monash.edu
Connection: keep-alive
Accept: text/html,application/xhtml+xml
User-Agent: Mozilla/5.0 (Macintosh)
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US;en
Content-Type: application/x-www-form-urlencoded
Content-Length: 26

search=data+communications
```

Request line

Request header

Request body

Request header and body are optional.
HTTP 1.1 requires at least the `Host` header.

Full HTTP Response

```
HTTP/1.1 200 OK
Date: Thu, 05 Mar 2015 08:30:48 GMT
Server: Apache/1.3.26 (Unix)
Transfer-Encoding: chunked
Content-Type: text/html

<html>
<body>
  <h1>Guido Tack</h1>
  <img src="images/guido3.jpg">
</body>
</html>
```

Response status

Response header

Response body

Response header and body are optional.

HTTP is stateless

- Each request is an **independent transaction**
- Example: shopping cart
  - *Client* browses online store
  - *Server* responds with web pages
  - *Client* puts item into shopping cart
  - *Client* continues browsing
  - How does the server keep track of **which client** put **which items** into the shopping cart?
    (This is called the **state** of the **session**)

A **session** is the sequence of transactions between a user and a web application.

Adding state to HTTP

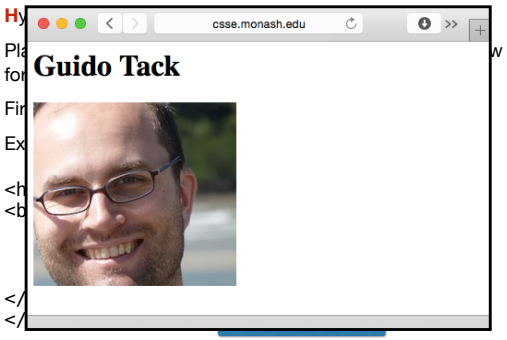
- Two approaches:
  - Client sends **session identifier** with every request, e.g. encoded in URL or as data in POST
  - Server sets a **cookie,** client transmits the cookie with **every future request**

Cookies can contain arbitrary data. Most often they contain a session identifier (random number that identifies a user). Some companies use cookies to **track users.**

HTML

- Hy
- Pla
  for
  w to
- Fir
- Ex

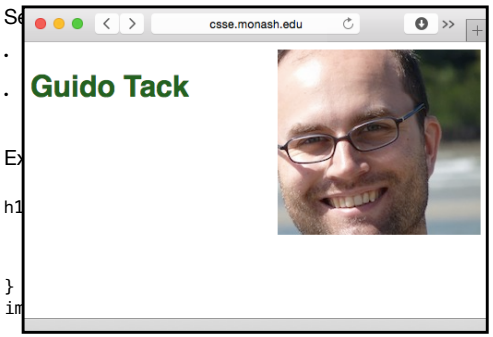
```
<h
<b
</
</
```



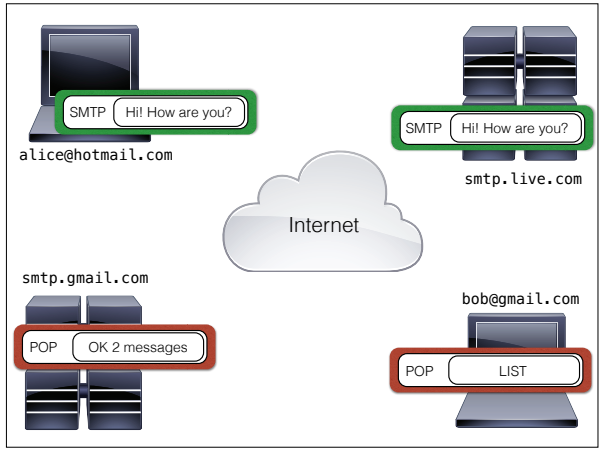
HTML + CSS

- Se
  -
  .
- Ex

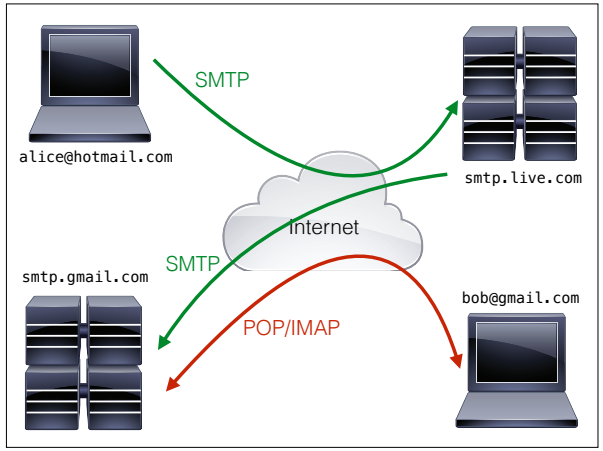
```
h1

}
im
```



Electronic Mail



Traditional two-tier setup: Alice's client sends email to mail server via SMTP, server sends email to Bob's mail server via SMTP. Mail **stays in Bob's inbox** until he checks for new mail. Bob's client fetches email from server via POP or IMAP.



This is the picture of Alice sending email to Bob. The other direction works the same (Bob sends via SMTP, Alice reads mail via POP or IMAP)

Email Protocols

- Simple Mail Transfer Protocol **(SMTP)**
  - Handles transfer of text messages between email client and mail server, and between mail servers
- Post Office Protocol **(POP)**
  - Messages are downloaded onto client and deleted from server
- Internet Message Access Protocol **(IMAP)**
  - Messages remain on server
  - Multiple clients can be connected simultaneously to same mailbox

Example SMTP Session

```
220 MyMailServer ESMTP Exim 4.82 Ubuntu Sat, 07 Mar 2015 20:37:24 +1100
HELO my.laptop
250 MyMailServer Hello laptop [192.168.1.5]
MAIL FROM:<alice@mymail.com>
250 OK
RCPT TO:<guido.tack@monash.edu>
250 Accepted
DATA
354 Enter message, ending with "." on a line by itself
From: "Alice" <alice@mymail.com>
To: "Guido Tack" <guido.tack@monash.edu>
Date: Mon, 09 Mar 2015 19:24:00 +1000
Subject: test message

Hi Guido!
This is just a test.

Cheers,
Alice
.
250 OK id=1YUBBf-0003Ch-M1
QUIT
221 MyMailServer closing connection
```

Header

Body

Black: server

Red: client

MIME

- **M**ulti-Purpose **I**nternet **M**ail **E**xtensions
- Remember: SMTP handles **plain text** email
- How do you attach other file types?
  - MIME specifies an **encoding**
  - Supports **character sets** (e.g. Unicode) to send emails with non-latin characters
  - Supports **non-text attachments**
  - Supports **multi-part** message bodies

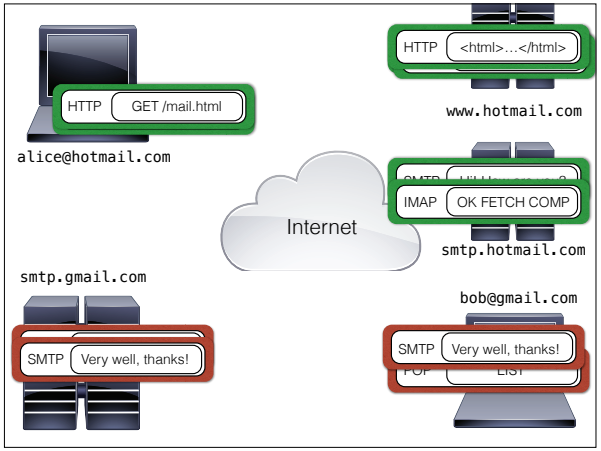
MIME Example

```
  Apple-Mail=_DC544A01-B885-421C-B475-2DBBCF2DEE37
Content-Transfer-Encoding: base64
Content-Disposition: inline;
   filename=guido3.jpg
Content-Type: image/jpeg;
   name="guido3.jpg"
Content-Id: <B073584A-EAF2-4A30-9ACC-1368C9C2E846@iiNet>

/9j/4AAQSkZJRgABAQEASABIAAD/4gVASUNDX1BST0ZJTEUAAQEAAAUwYXBwbAIgAABtbnRyUkdC
IFhZWiAH2QACABkACwAaAAthY3NwQVBQTAAAAABhcHBsAAAAAAAAAAAAAAAAAAAAAAAA9tYAAQAA
AADTLWFwcGwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAtk
c2NtAAABCAAAAvJkZXNjAAAD/AAAAG9nWFlaAAAEbAAAABR3dHB0AAAEgAAAABRyWFlaAAAElAAA
ABRiWFlaAAAEqAAAABRyVFJDAAAEvAAAAA5jcHJ0AAAEzAAADhjaGFkAAAFBAAAACxnVFJDAAAE
vAAAAA5iVFJDAAAEvAAAAA5tbHVjAAAAAAAAABEAAAAMZW5VUwAAACYAAAJ+ZXNFUwAAACYAAAGC
ZGFESwAAAC4AAAHqZGVERQAAACwAAAGoZmlGSQAAACgAAADcZnJGVQAAACgAAAEqaXRJVAAAACgA
AAJWbmxOTAAAACgAAAIYbmJOTwAAACYAAAEEcHRCUgAAACYAAAGCc3ZTRQAAACYAAAEEamFKUAAA
ABoAAAFSa29LUgAAABYAAJAemhUVwAAABYAAAFsemhDTgAAABYAAAHUcnVSVQAAACIAAAKkcGxQ
TAAAACwAAALGAFkAbABlAGkAbgBlAG4AIABSAEcAQgAtAHAAcgBvAGYAaQBpAGwAaQBHAGUAbgBl
AHIAaQBzAGsAIABSAEcAQgAtAHAAcgBvAGYAaQBsAFAAcgBvAGYAaQBsACAARwBpAG4A6QByAGkA
cQB1AGUAIABSAFYAQk4AgiwAIABSAEcAQgAgMNcw7TDVMKEwpDDrkBp1KAAgAFIARwBCACCCcl9p
Y8+P8ABQAGUAcgBmAGkAbAAgAFIARwBCACAARwBlAG4A6QByAGkAYwBvAEEAbABsAGcAZQBtAGUA
aQBuAGUAcwAgAFIARwBCAC0AUAByAG8AZgBpAGxmbpAaACAAUgBHAEIAIGPPj/Blh072AEcAZQBu
AGUAcgBlAGwAIABSAEcAQgAtAGIAZQBzAGsAcgBpAHYAZQBsAHMAZQBBAGwAZwBlAG0AZQBlAG4A
IABSAEcAQgAtAHAAcgBvAGYAaQBlAGzHfLwYACAAUgBHAEIAINUEuFzTDMd8AFAAcgBvAGYAaQBs
AG8AIABSAEcAQgAgAEcAZQBuAGUAcgBpAGMAbwBHAGUAbgBlAHIAaQBiACAAUgBHAEIAIABQAHIA
```

Represent image data as plain text!

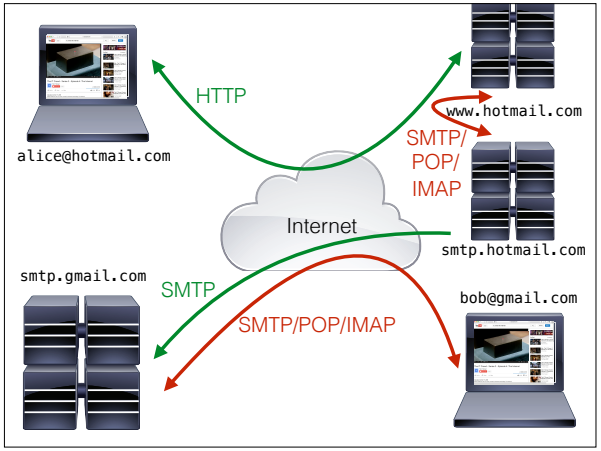
Two-tier vs Three-tier mail

- **Two-tier:**
  - Client-server architecture
  - Client implements application logic, talks to server using SMTP and POP/IMAP
- **Three-tier:**
  - Thin client accesses **web application**
  - Server handles application logic
  - Client accesses server using HTTP



Web Mail is a **three-tier thin client-server** architecture.

The web server presents an email **web application**. It forwards messages via SMTP to a mail server and retrieves email via IMAP. Because all systems use **open standards**, both sides can use two- or three-tier architectures.



Hotmail may not use SMTP&IMAP internally (e.g. they could directly access a data base instead). But transmission from smtp.hotmail.com to smtp.gmail.com **must** use SMTP.

Summary

- **Application architecture** determines how client and server share the work load (server-based, client-based, client-server, thin-client-server, peer-to-peer)
- **WWW**: based on HTML hypertext and URLs, communicates using HTTP
- **Email**: SMTP, POP, IMAP, thin-client web mail
- Important: **standard protocols enable interoperability**