COMP3310/6331 – #15

The Web/HTTP

Dr Markus Buchhorn: *markus.buchhorn@anu.edu.au*

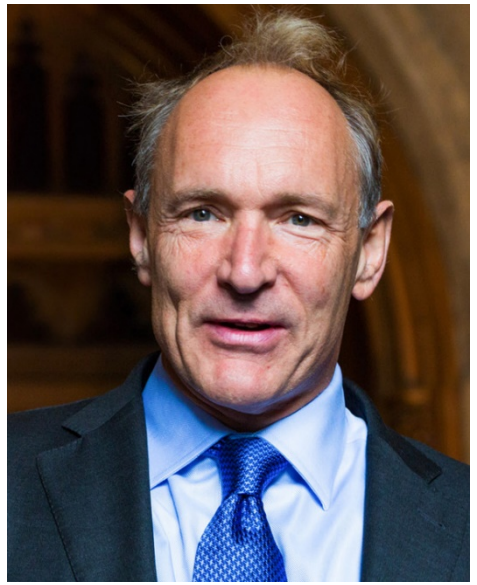
Applications choose their transport

- UDP-based applications:
  - Short messages
  - Simple request/response transactions
  - Light server touch
  - ARQ suffices

- TCP-based applications:
  - Larger content transfers
  - Longer, and more complex, sessions
  - Reliability matters
  - Packaging and presentation becomes important – TCP is a bytestream

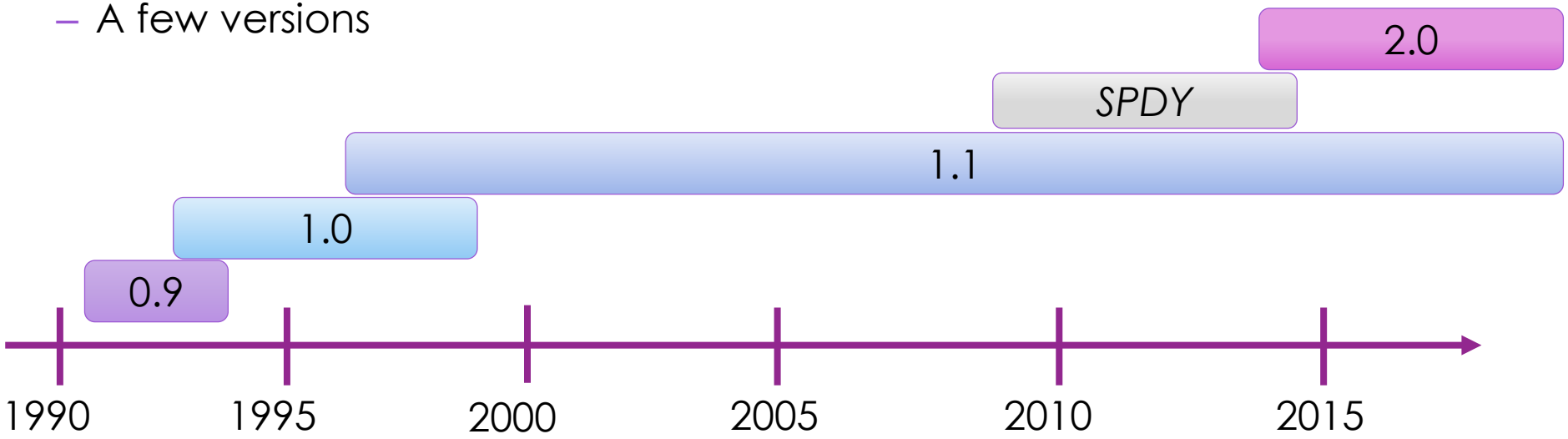
The Web

- Back in the old days… even on the Internet
  - Everything was local
  - Everything was standalone
  - If two things were connected, you made a local copy
- Gopher, WAIS changed that
- The World Wide Web actually changed that
  - Sir Tim Berners-Lee – CS/Eng, at CERN, 1989
  - Core idea – HTML to link "*stuff*"
  - Which needed a protocol – HTTP (IETF)
  - http://info.cern.ch/hypertext/WWW/TheProject.html
  - Now heads up W3C.org (and many other roles)



The Web

- HTTP underpins the web
  - to deliver html and (many) associated content items
- Request(s)/response(s) from multiple resources/sites
  - Port 80, TCP
  - A few versions



Resources

- Aggregating and linking resources need IDENTIFIERS
- Uniform Resource Identifiers (URI)
  - Or is that a Uniform Resource Name (URN)?
  - Or a Uniform Resource Locator (URL)?

- Stick with URLs here

scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]

URLs - schemes

- ***scheme***:*[//[user[:password]@]host[:port]][/path][?query][#fragment]*
- *https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml*
- 280 of them!
- Some interesting ones:
  - Callto://<phone-number> and Tel://<phone-number>
  - mailto://<email-address>
  - File://<path-to-file-on-my-system>
  - ftp://<some-host>/some-file

- And http:// and https://

URLs – the rest

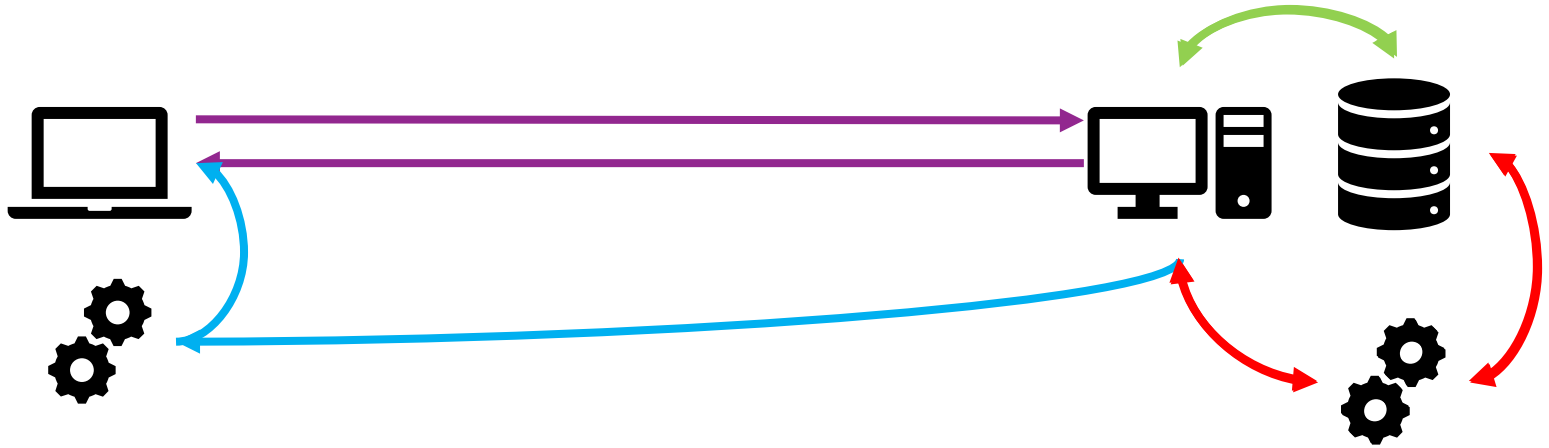
- ***http***://*user:password@host:port [/path][?query][#fragment]*
- You can provide authentication inline. If you want. In plain text…
- **Host** = something you find in the DNS (or an IP address)
- **Port** = if it's not 80, tell me

- **Path** identifies (absolute-path-to) resource on the host
  - #**fragment** goes to a point within that resource
  - *http://en.wikipedia.org/wiki/IEEE_802#See_also*

- **Query** passes information to that resource

The magic of the web

- Static vs dynamic content
- Server-side vs client-side dynamic content



Or all of the above.

Anything that presents information can appear on a web page

8 Steps to HTTP happiness

1. Parse URL
2. Resolve DNS
3. Connect to host:port via TCP
4. Make HTTP request
5. Receive content
6. Close TCP connections
7. Unpack content
8. Render

HTTP requests – RFC1945 (HTTP 1.0)

- Request/response, text based, start with the **method**

GET <path> HTTP/1.0	Get the resource at <path>
HEAD <path> HTTP/1.0	Get the headers about the resource at <path>
POST <path> HTTP/1.0	Append my contribution to the resource at <path>

- Requests indicate the protocol version
  - Servers provide backwards compatibility
- Server returns headers, and a body (entity)

Use ‘telnet’ as a client

markus@homemaster:~$ telnet www.google.com 80

GET / HTTP/1.0\n\n

```
HTTP/1.0 302 Found
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Location: http://www.google.com.au/?gfe_rd=cr&dcr=0&ei=192xWr-uObPu8wfUm4noDQ
Content-Length: 272
Date: Wed, 21 Mar 2018 04:21:43 GMT

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-
8"> <TITLE>302 Moved</TITLE></HEAD><BODY> <H1>302 Moved</H1> The document has
moved <A HREF="http://www.google.com.au/?gfe_rd=cr&amp;dcr=0&amp;ei=192xWr-
uObPu8wfUm4noDQ">here</A>. </BODY></HTML>
```

And get it wrong?

markus@homemaster:~$ telnet www.google.com 80

GET / HTTP/3.0\n\n

```
HTTP/1.0 400 Bad Request
Content-Type: text/html; charset=UTF-8
Content-Length: 1555
Date: Wed, 21 Mar 2018 04:25:14 GMT

<!DOCTYPE html> <html lang=en>
<title>Error 400 (Bad Request)!!1</title>
 <style> […]
<b>400.</b> That’s an error.
Your client has issued a malformed or illegal request.
That’s all we know.
```

HTTP/1.0 302 Found

HTTP/1.0 400 Bad Request

HTTP Responses

- Start with 3-digit HTTP codes

Code	Category	Examples
1xx	Informational	*No longer used; could be used*
2xx	Successful	**200 OK**; 201 Created;
3xx	Redirection	301 Moved Permanently; **302 Moved Temporarily**
4xx	Client error	400 Bad request; 403 Forbidden; **404 Not Found**
5xx	Server error	500 Internal Server Error; 503 Service Unavailable

Headers (both directions)

- Provide information about the resource
  - Or additional information about HTTP codes
  - Or other hints about the server/client

Location:
http://www.google.com.au/?gfe_rd=cr&dcr=0&ei=192xWr-uObPu8wfUm4noDQ

https://commbank.com.au

"Function"	Examples
Browser Capabilities (c->s)	*User-Agent, Accept, Accept-Charset, Accept-Encoding, Accept-Language*
Caching (both)	*Date, Last-Modified, Expires, Cache-Control, Etag, If-Modified-Since, If-None-Match*
Browser states (c->s)	*Cookie, Referer, Authorization, Host, Range*
Content delivery (s->c)	*Content-Encoding, -Length, -Type, -Language, -Range, Set-Cookie,* ***Location***

HTTP is stateless

- Every request stands alone
  - Server shouldn't hold state for everything
  - How do I stay logged-in?
- Encode a session key in a URL

- Encode a session key in Cookies!
  - Set by the server, held by the client – and returned whenever "relevant" (same domain).
  - Include various tags/types/flags, and the domain that set them. Sort of.

- Session Cookie – deleted when browser closes
- Persistent Cookie – kept till expiry
- Secure Cookie – only over secure channels
- And many more

Protocol Performance?

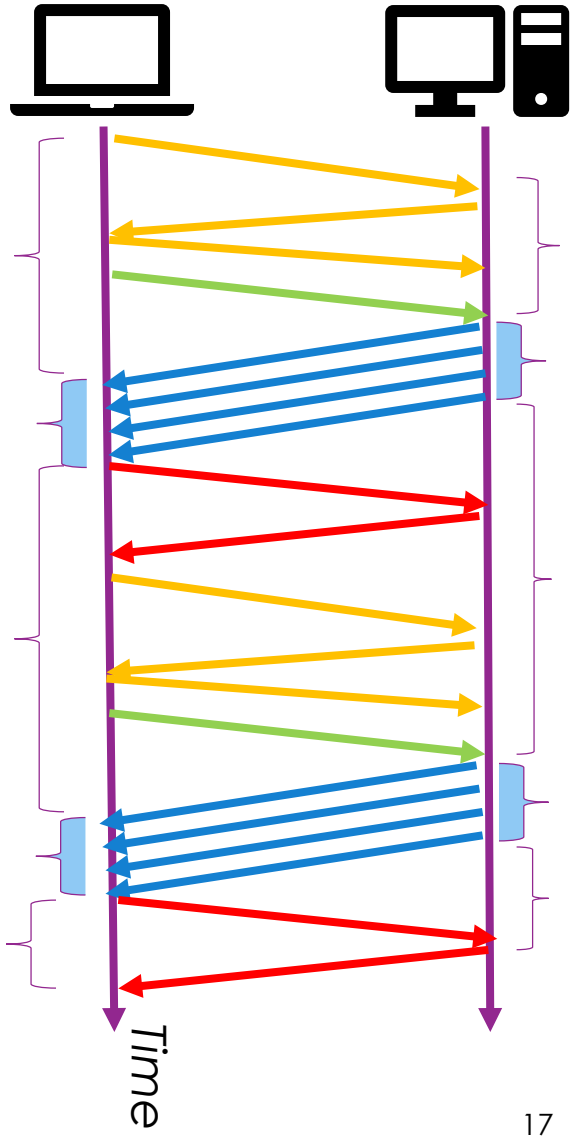
- Measured in www by user experience – *Page Load Times*
  - People get bored. They don't buy from you.

- Depends on:
  - Browser
  - Content structure and complexity, processing
  - *Protocols: HTTP, TCP, IP*
  - *Network path, bandwidth and round-trip-time*

- Typical web page:
  - Core html
  - Plus scripts, css, images, frames/divs, …
    - Each is their own 'object' for GETting

HTTP 1.0

- One TCP connection for each page resource
- Sequential **request**/response
  - Multiple TCP connections to the one server
  - TCP overhead on **set-up**/**tear-down**
  - Network and endpoints idle for significant periods
    - Only **delivering** for a small fraction of time
- Easy – but slow
- Worse with many small resources
    - *(and TCP throughput has performance limits too…)*



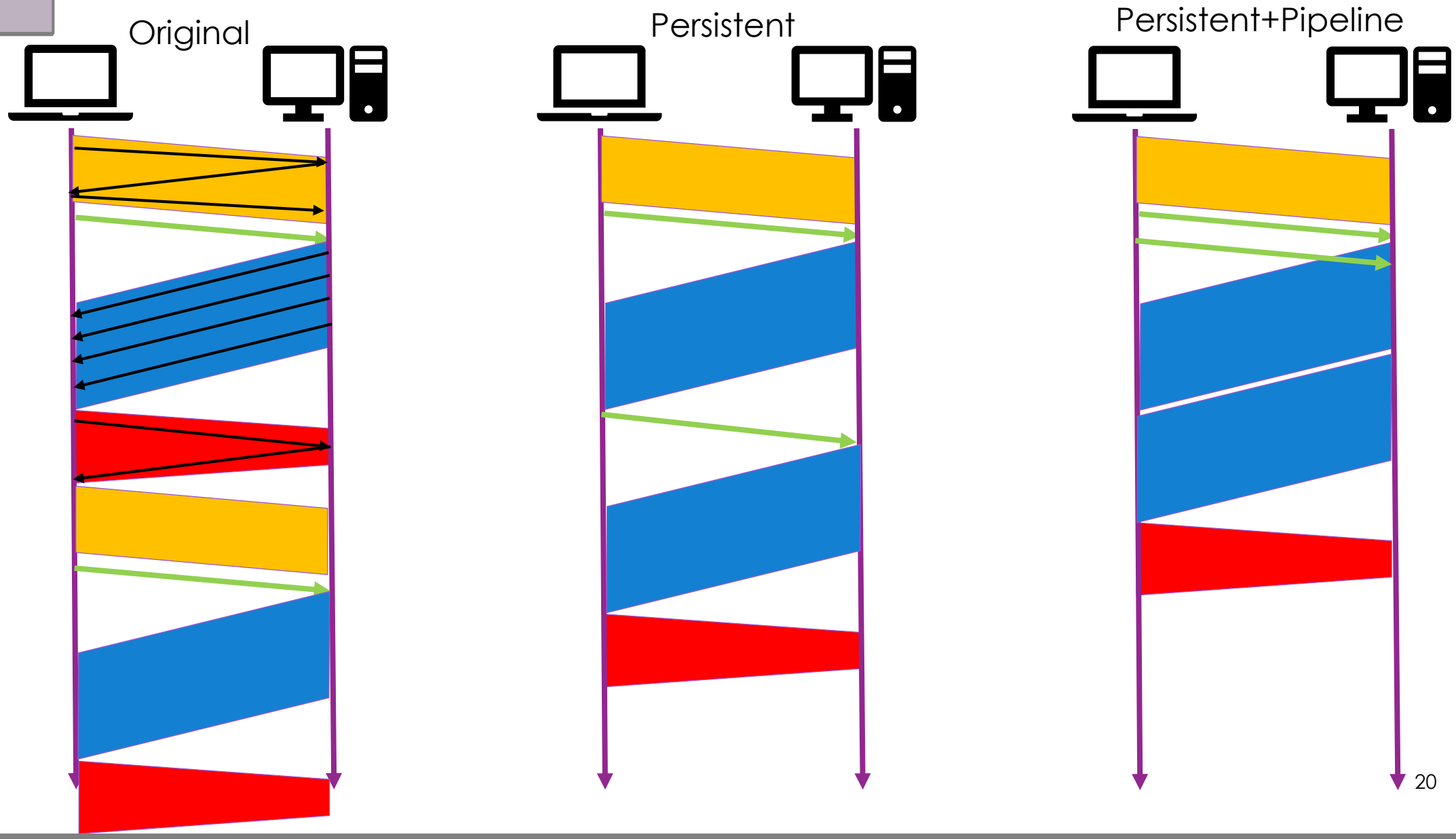
Improvements to “Page Load Times”?

- Adjust content to suit client
  - Small screen, small images, Large screen, large images
- Add caching
  - Avoid getting the same thing multiple times
- Change http
  - Be smarter with its connections

Smarter (http) connections

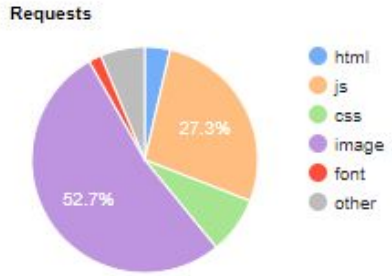
- **Parallelism**
  - Instead of one http GET, just do 8+ at the same time…!
    - No server change needed
    - Take advantage of idle bandwidth
    - *Creates bursts of CPU/NIC load, traffic and loss*
- **Persistence** – sequential requests (HTTP 1.1)
  - Open one TCP connection
  - And use it for multiple requests in order
- **Persistence** – and **pipelining**
  - Make all your requests at once
  - Responses come back in order

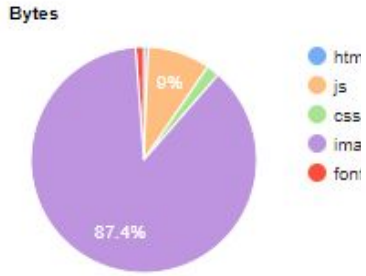
Comparison

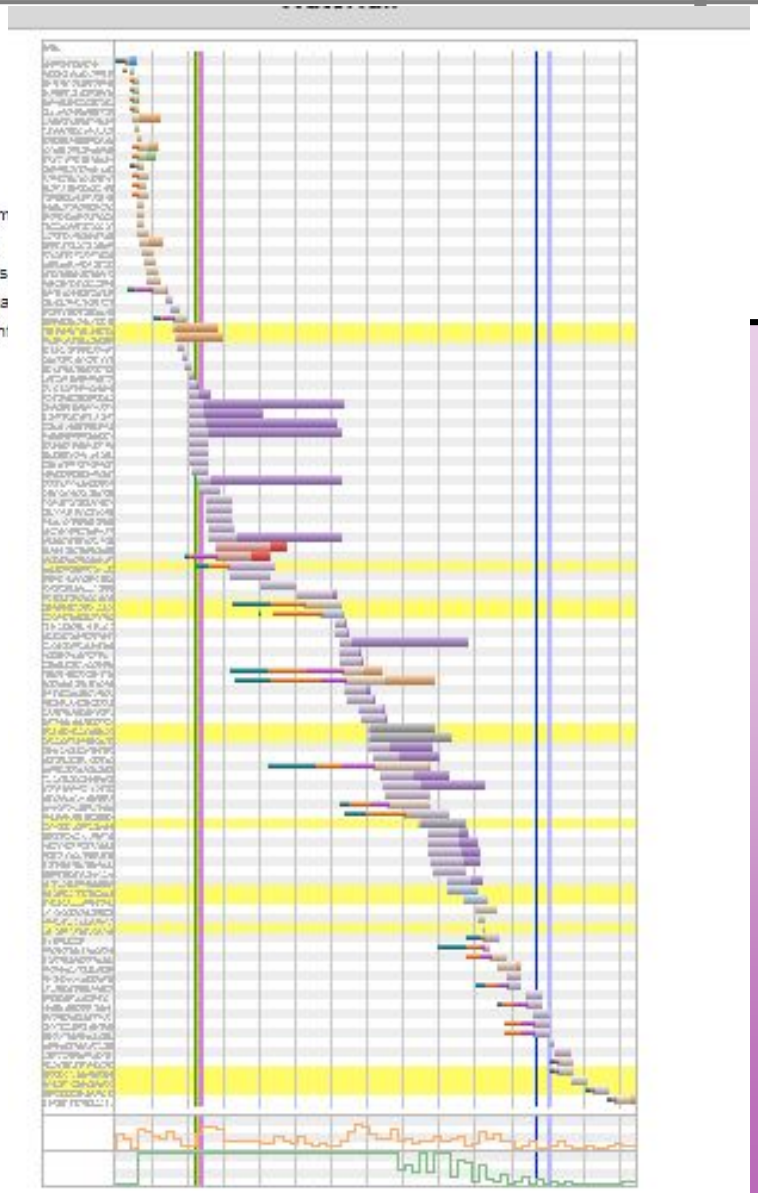


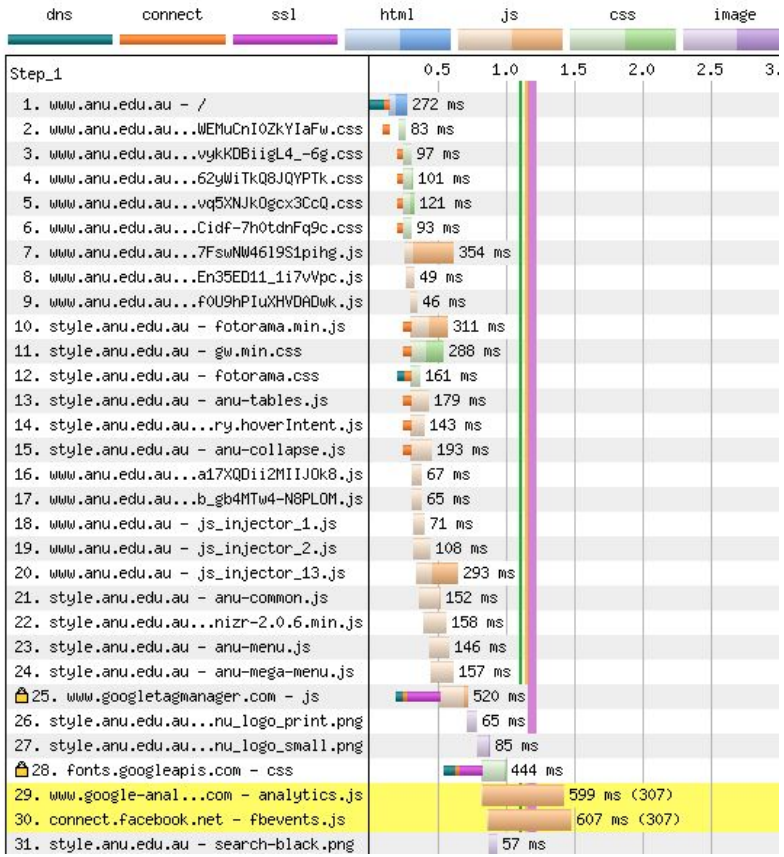
Real world performance

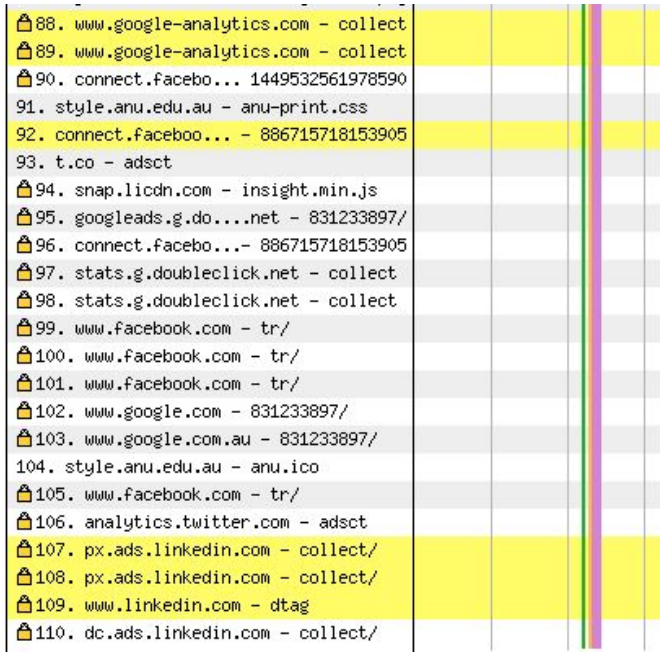
Webpagetest.org
on www.anu.edu.au











More performance: Caching

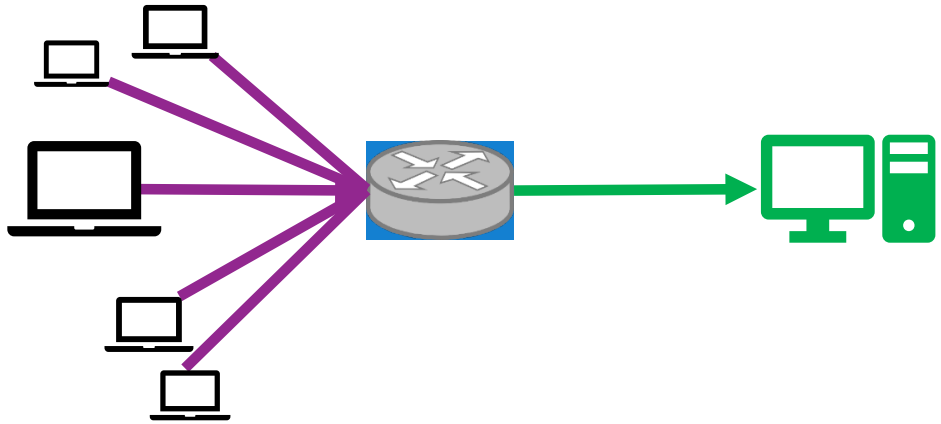
- In the browser
  - Don't download what you grabbed earlier
  - Populated on demand
- Along the path
  - Same idea, bigger and better and SHARED – win for you and your ISP/org
  - Proxy caches – on your behalf
- Content Distribution Networks (CDNs)
  - Replicate the site somewhere closer to you
  - And then act like a cache

The art of caching

- How do you know cache is good?
  - **Expires** header (HTTP 1.0)
    - Should…
  - **Last-modified** header (HTTP 1.0)
    - If you have it – take a guess
    - If you don't have it – ask for it (HEAD method)
  - **E**(ntity)**Tag** (HTTP 1.1)
    - Like a checksum, a small HEAD request
- '**conditional GET**' (HTTP 1.0)
  - Header: If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match
- '**partial GET**' (HTTP 1.1)
  - **Range** header - only part of the entity is transferred

Proxies

- Somebody else does the work for you
  - Hide network internals, protect clients, …
- **Proxying cache** – or – **Caching Proxy**



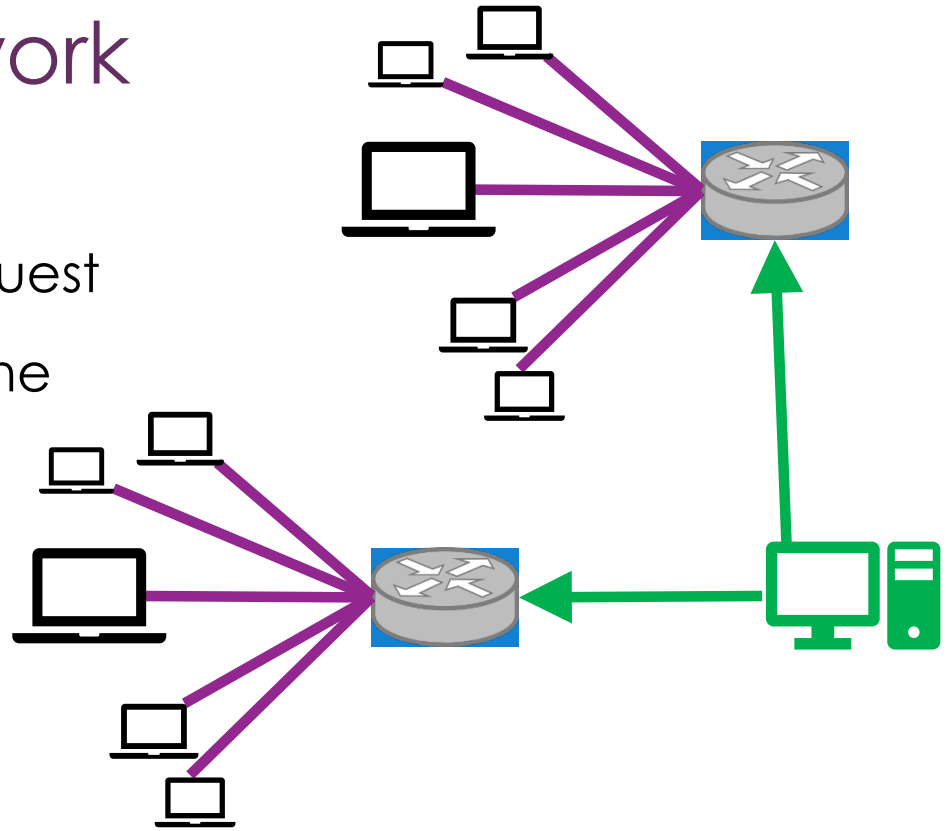
- Put cache out further on the network – and share it
  - Win: Performance
  - Win: Network traffic reduction
  - Win: Security checking
  - Win: Organisation Access Policies!!

Downsides:

- Not for secured content
- Not for dynamic content
- Gets filled with lots of ‘fluff’

Content Distribution Network

- Invert the picture:
- **Push** content to caches **before** the request
- Use DNS to send clients to nearest cache
  - *Html encodes cache locations*
- Take traffic load off popular sites
  - And the ISPs that host them
  - Win:Win:Win
- Akamai (~1996) pioneered this
  - It's a commercial service (benefits clients)
  - They see a lot of network behaviours (benefits Akamai)



Ever faster/better

- HTTP 2.0
  - Better pipelining of requests
  - Client can prioritise server responses
  - Header compression
  - Server push
    - "You'll probably want this too"
  - *Slowly appearing, some contentious elements – expect to see HTTP 2.1?*

- Server-side application-level improvements?
  - Some Apache modules rewrite/repackage your page (and code) on the fly…

HTTP as a ‘transport’ protocol?

- It carries real-time audio/video!?!
  - Various web-conferencing apps – vs RTP, RTSP, …
- SOAP and REST
  - *Simple Object Access Protocol*
  - *Representational State Transfer*
  - Remote Procedure Calls (RPC) over HTTP
- Used as a firewall-traversal-protocol
  - Everything (else) gets blocked!
  - So let’s use HTTP…