

SCC.306 Internet Applications Engineering

Web Architecture and Performance

Matthew Bradbury and Phil Benachour

m.s.bradbury@lancaster.ac.uk and p.benachour@lancaster.ac.uk

Internet is not the Web

- The internet is broader than the Web
- The Web is one means of information transfer over the internet
 - Typically using a protocol such as Hyper Text Transfer Protocol (HTTP)
- Many other applications make use of the Web
 - Variety of other protocols to facilitate these applications

Part I: Web Architecture



- “We wanted to understand how much the speed of our website affected user engagement, specifically, the quantity of articles read, Using that data we then wanted to quantify the impact on our revenue.”*

Mean % drop in article views between variants and control

Page load time	7 days impact	28 days
1 second slower	-4.9%	-4.6%
2 second slower	-	-5.0%
3 second slower	-7.2%	-7.9%

- Omitted as data did not reach 95% statistical significance

Web Servers

But now they're everywhere

- The primary function of a web server is to **deliver web pages to networked clients**
- A web server is just a piece of software that:
 - Takes a request in HTTP format for some named resource over the network
 - Serves that resource back to the client as payload to an HTTP response (HTML/ MIME formats)
- What's changed since this was originally envisioned?

Web and HTTP

- First, a quick review...
 - **web page** consists of objects
 - object can be HTML file, JPEG image, video file, audio file,...
 - web page consists of **base HTML-file** which includes **several referenced objects**
- each object is addressable by a URL, e.g.,

`www.clevername.com/someDept/pic.gif`

host name

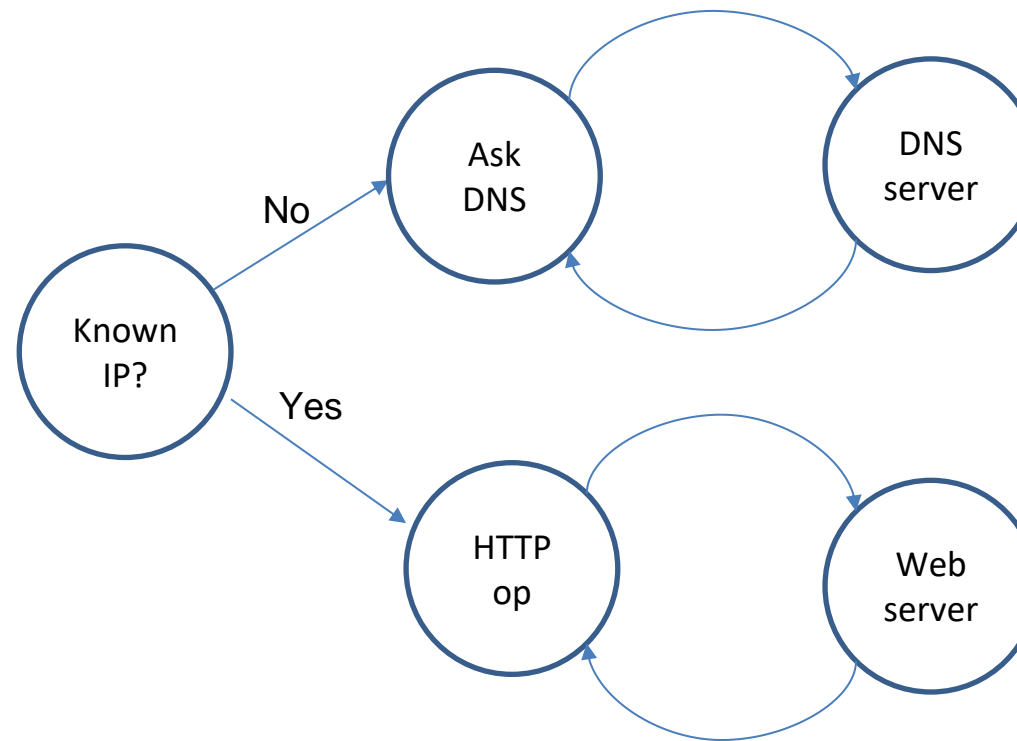
server specific path

But what about DNS?

- Before we can request the page we want, we need the IP (Internet Protocol) address of the server
 - bbc.co.uk has address 212.58.246.78
 - lancaster.ac.uk has address 148.88.2.80
 - Your operating system and network routers use this address to decide how to route the data packets making up your request, hop by hop throughout the network
 - If we don't know the IP, we need a way to find it out. The **domain name system (DNS)** provides a distributed database we can query

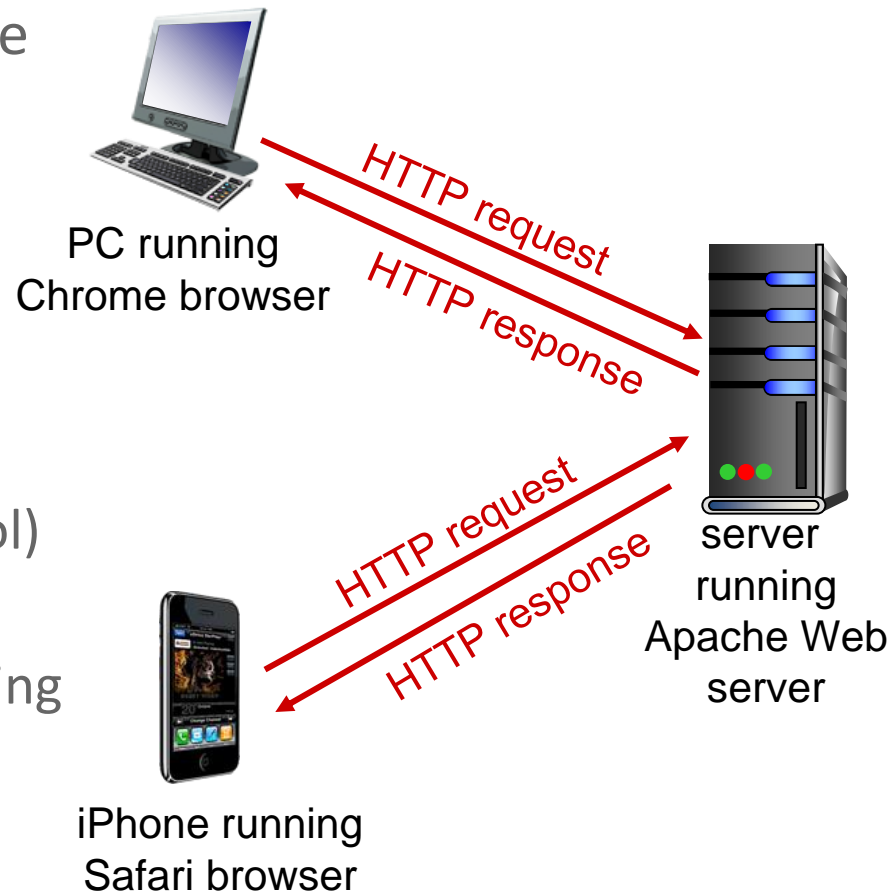
DNS is a network service

- DNS's role, vs. the web server



HTTP request/ response

- For each resource host, resolve IP using DNS
- Make request using HTTP: hypertext transfer protocol
- client/server model
 - **client**: browser that requests, receives, (using HTTP protocol) and “displays” Web objects
 - **server**: Web server sends (using HTTP protocol) objects in response to requests



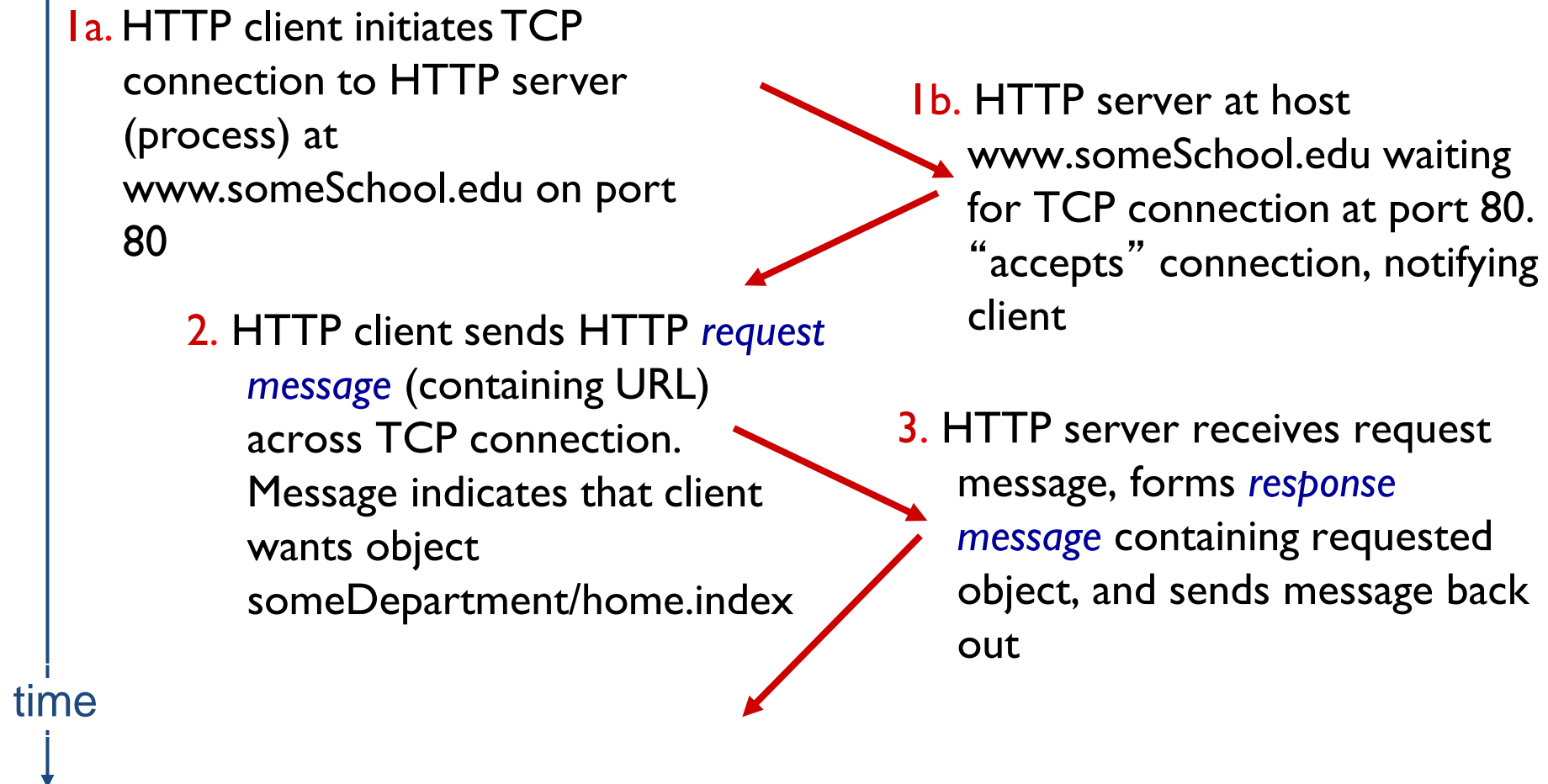
-
- Uses **Transmission Control Protocol (TCP)**:
 - Client initiates TCP connection (creates socket) to server, port 80
 - Server accepts TCP connection from client
 - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
 - TCP connection closed

HTTP in Action

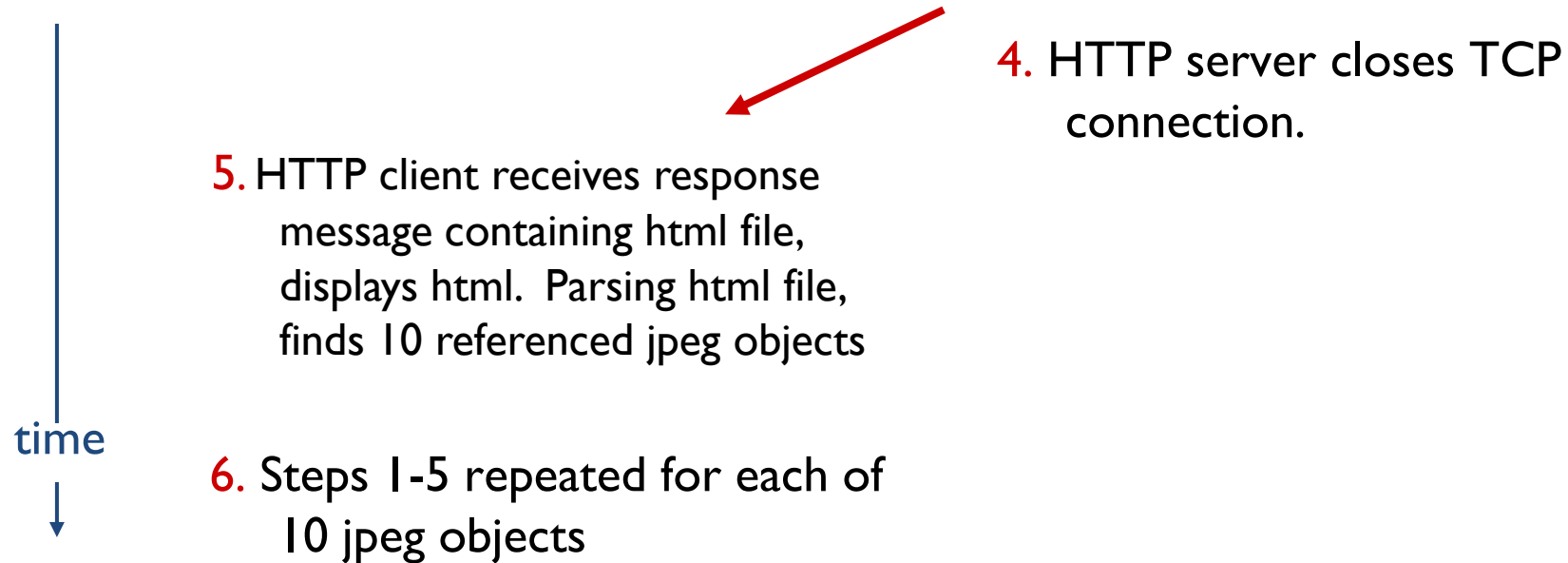
suppose user enters URL:

`www.someSchool.edu/someDepartment/home.index`

(contains text,
references to 10
jpeg images)

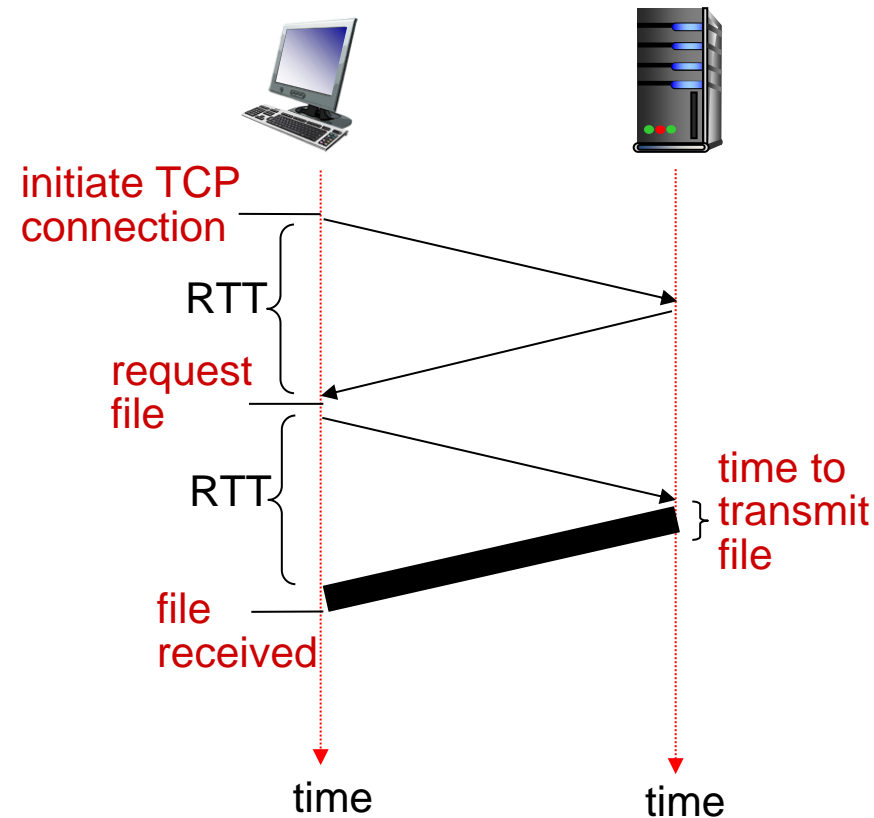


HTTP in Action (cont.)



HTTP response time

- **RTT (definition)**: time for a packet to travel from client to server and back
- **HTTP response time**:
 - one RTT to initiate TCP connection
 - one RTT for HTTP request and first few bytes of HTTP response to return
 - file transmission time
 - non-persistent HTTP response time =
 - $2\text{RTT} + \text{file transmission time}$



Defining Latency

-
- A measure of the time **delay** experienced by a system
 - In a computer network: *the time it takes for a packet of data to get from one designated point to another*

Defining Latency

```
1. broadbent@carlisle: ~ (zsh)
~ ➤ ping -c 10 speedtest.wdc01.softlayer.com
PING speedtest.wdc01.softlayer.com (208.43.102.250): 56 data bytes
64 bytes from 208.43.102.250: icmp_seq=0 ttl=57 time=99.909 ms
64 bytes from 208.43.102.250: icmp_seq=1 ttl=57 time=99.399 ms
64 bytes from 208.43.102.250: icmp_seq=2 ttl=57 time=183.120 ms
64 bytes from 208.43.102.250: icmp_seq=3 ttl=57 time=104.485 ms
64 bytes from 208.43.102.250: icmp_seq=4 ttl=57 time=98.528 ms
64 bytes from 208.43.102.250: icmp_seq=5 ttl=57 time=112.004 ms
64 bytes from 208.43.102.250: icmp_seq=6 ttl=57 time=268.314 ms
64 bytes from 208.43.102.250: icmp_seq=7 ttl=57 time=289.131 ms
64 bytes from 208.43.102.250: icmp_seq=8 ttl=57 time=176.033 ms
64 bytes from 208.43.102.250: icmp_seq=9 ttl=57 time=229.138 ms

--- speedtest.wdc01.softlayer.com ping statistics ---
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 98.528/166.006/289.131/70.757 ms
~ ➤
```

Defining Throughput

-
- **Bandwidth:** the amount of information that can be transmitted over a **network** in a given time
 - Synonymous with: **network throughput**, the amount of data moved successfully from one place to another in a given time period

Defining Throughput

```
mbradbury@inb060000001:~$ iperf -c ping-90ms.online.net
-----
Client connecting to ping-90ms.online.net, TCP port 5001
TCP window size: 85.0 KByte (default)
-----
[  3] local 172.24.72.77 port 50960 connected with 62.210.18.41 port 5001
[ ID] Interval          Transfer    Bandwidth
[  3]  0.0-10.5 sec    22.0 MBytes 17.6 Mbits/sec
```

Measuring Latency and Throughput:

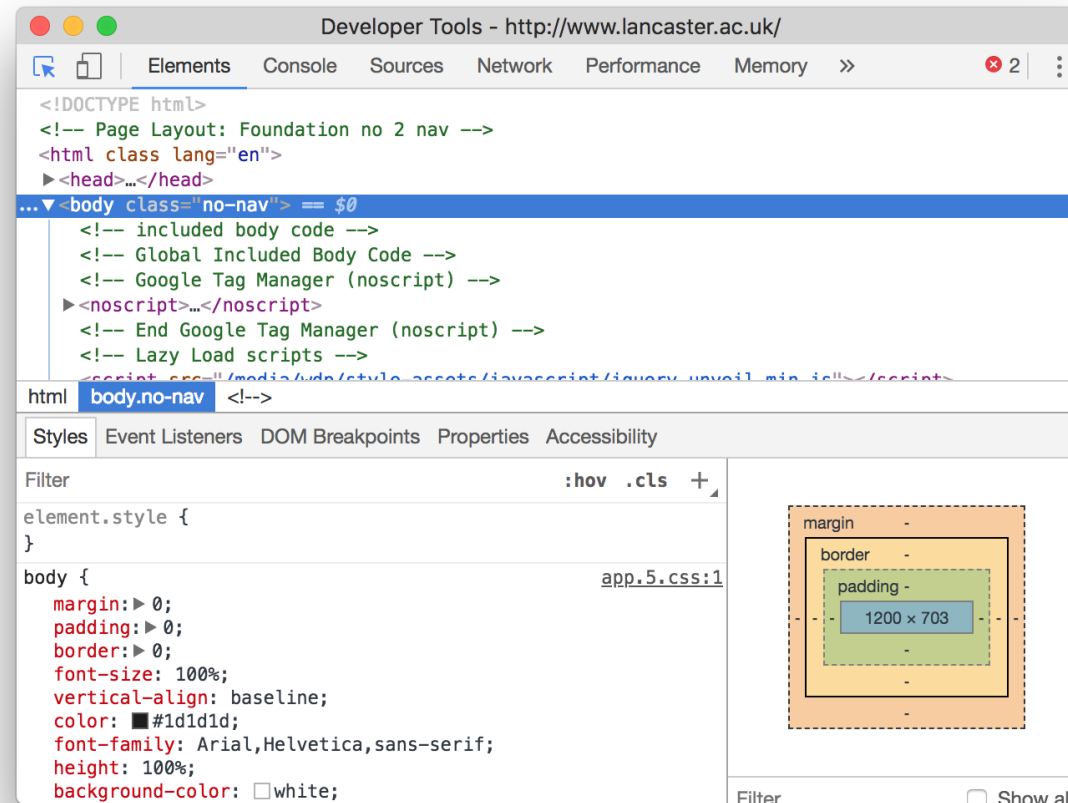
- Ping
 - `ping lancaster.ac.uk`
- iPerf
 - `iperf -c ping-90ms.online.net`

Part II: Front-end Web Performance and Measurement



The Document Object Model (DOM)

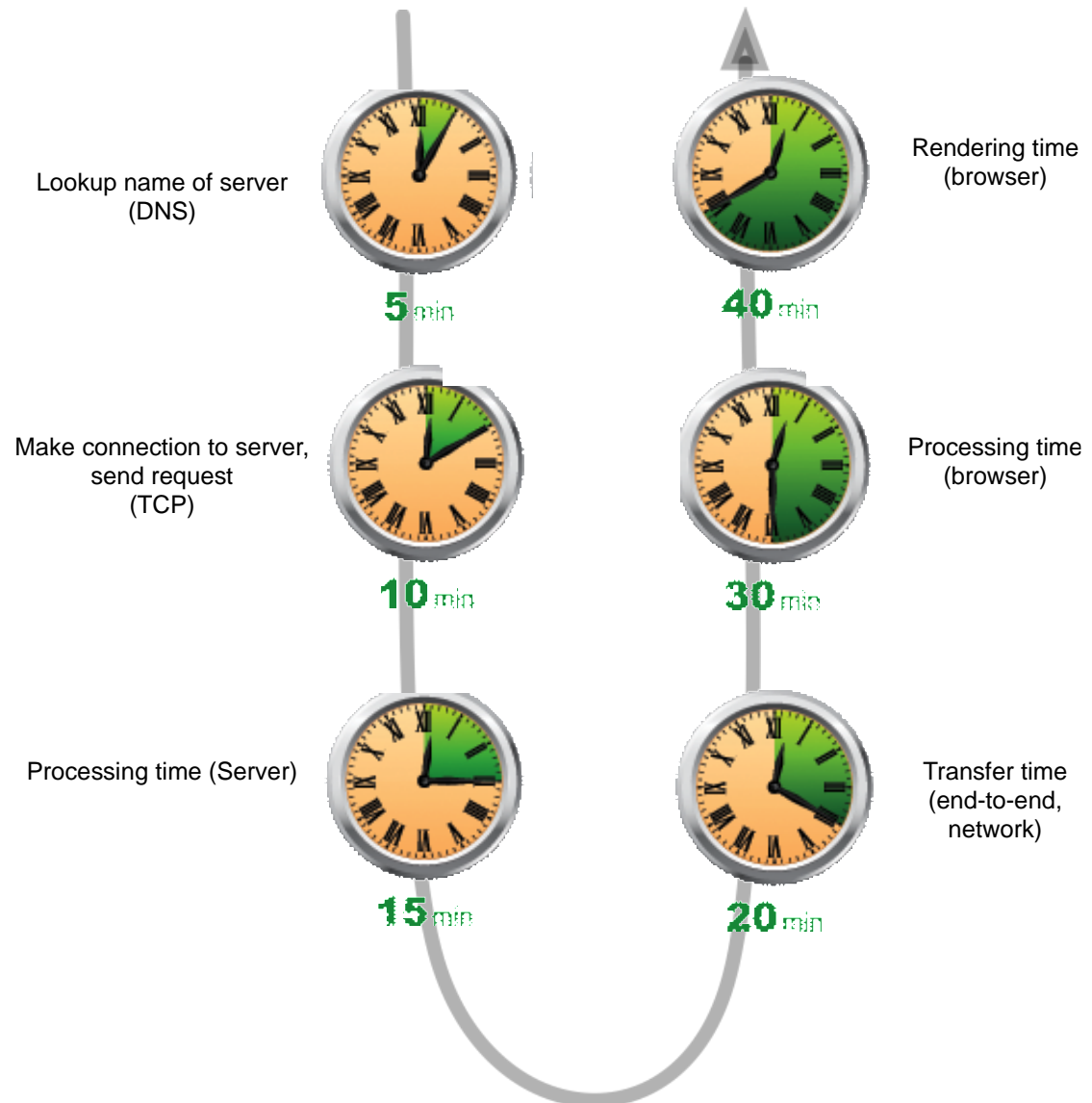
- Way to manipulate the structure & style of HTML
- Represents internals of the page as the browser sees it



Where does the time go?

- Latency (in milliseconds, not minutes 😊), i.e. the time taken to complete each stage before the content can be displayed.

- (Much) more detail:
<http://bit.ly/1uHwuPD>



This means...

- Each time we fetch a piece of content we incur time penalties (**latency**)
 1. Looking up the domain name (DNS)
 2. Sending and receiving HTTP messages (due to the **latency** / delay and bandwidth & congestion of the network and the **size of the content**)
 3. The performance and load on the server
 4. The complexity of the pages we render and performance of the client

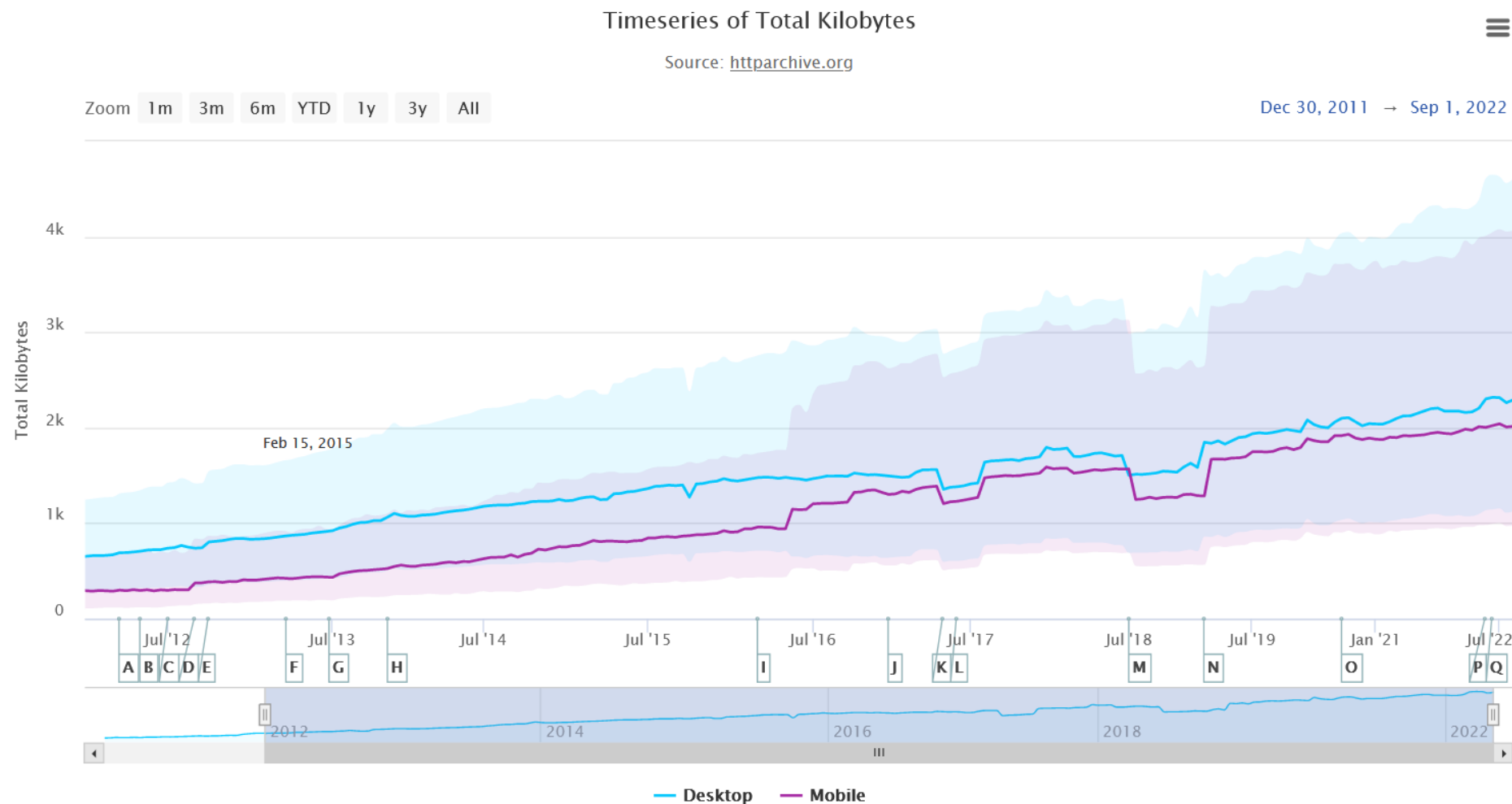
This is all dynamic, and changes over time.

Front-end Performance

- <http://yellowlab.tools>
 - lancaster.ac.uk
 - bbc.co.uk
 - blinkee.com
- <https://developers.google.com/web/tools/chrome-devtools>

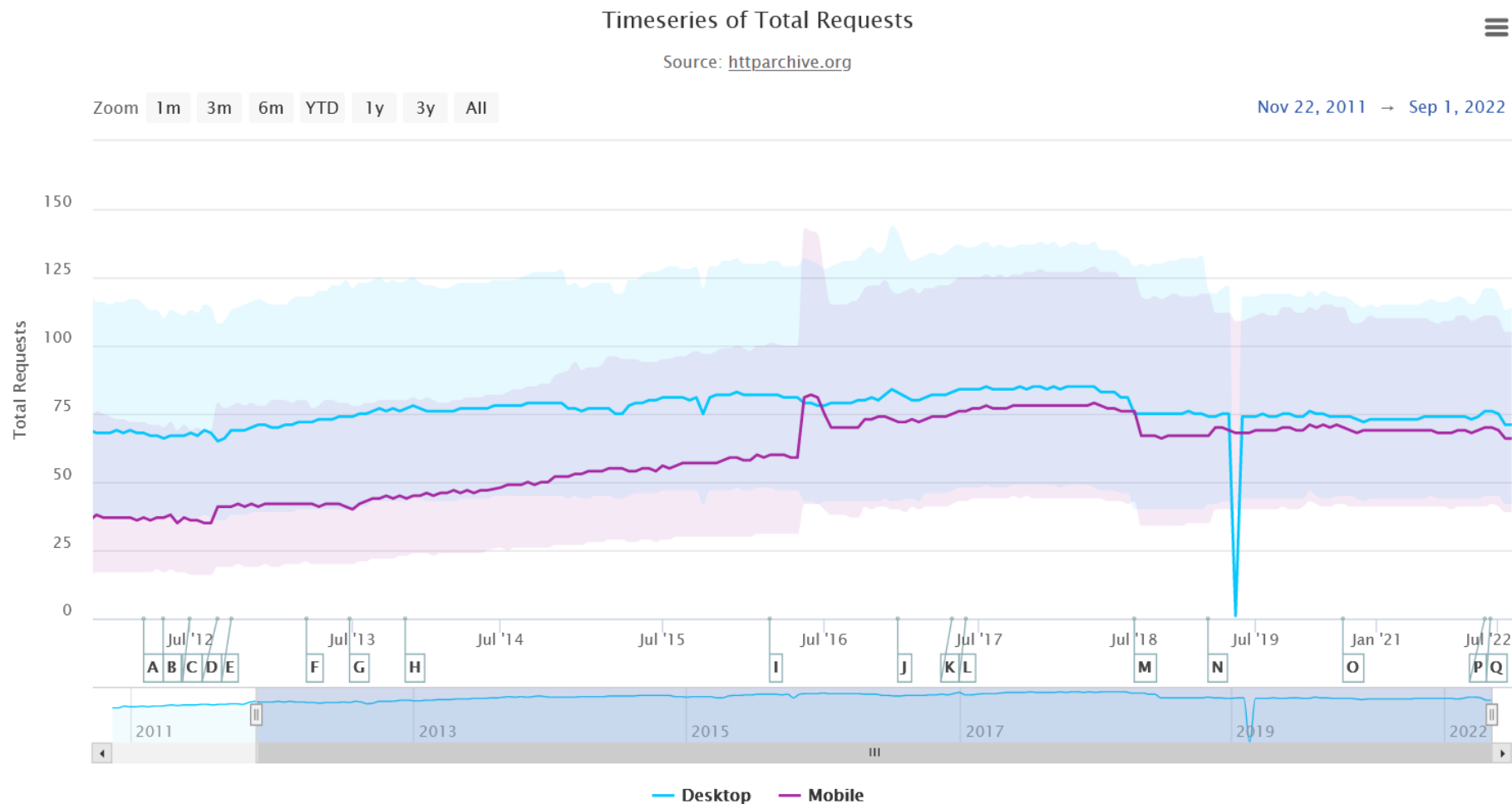
HTTP Archive, state of the Web

<https://httparchive.org/reports/state-of-the-web>



HTTP Archive, state of the Web

<https://httparchive.org/reports/state-of-the-web>



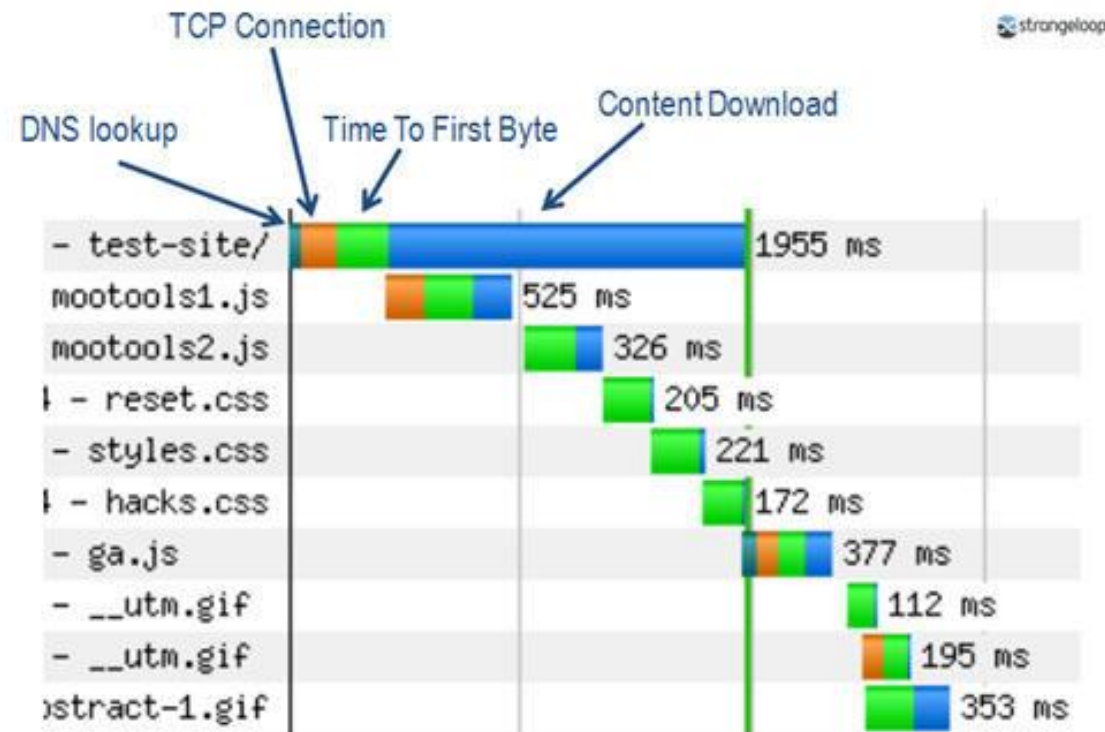
Understanding website performance

Waterfalls 101 – note: green and blue lines



Each web page contains 50-100 objects†

- Each making a round-trip between the browser and the server (request out, processing time, response back)



Definitions

- **Time to first byte** is the time from the request to the server until the first byte of the response is received by the browser
 - **> 100ms** or so, and you have a slow server
- **Start render** is when content begins to display in the user's browser (doesn't mean useful content!)
 - Should be consistent across pages and **less than 2 seconds**
 - *Note: browsers continue to evolve when they draw (e.g. Chrome)*
- **DOMContentLoaded** is when the initial HTML document has been completely loaded and parsed, sufficient to build the DOM (without necessarily waiting for some objects, such as images of known size, to finish loading).
 - DOMContentLoaded = Start Render

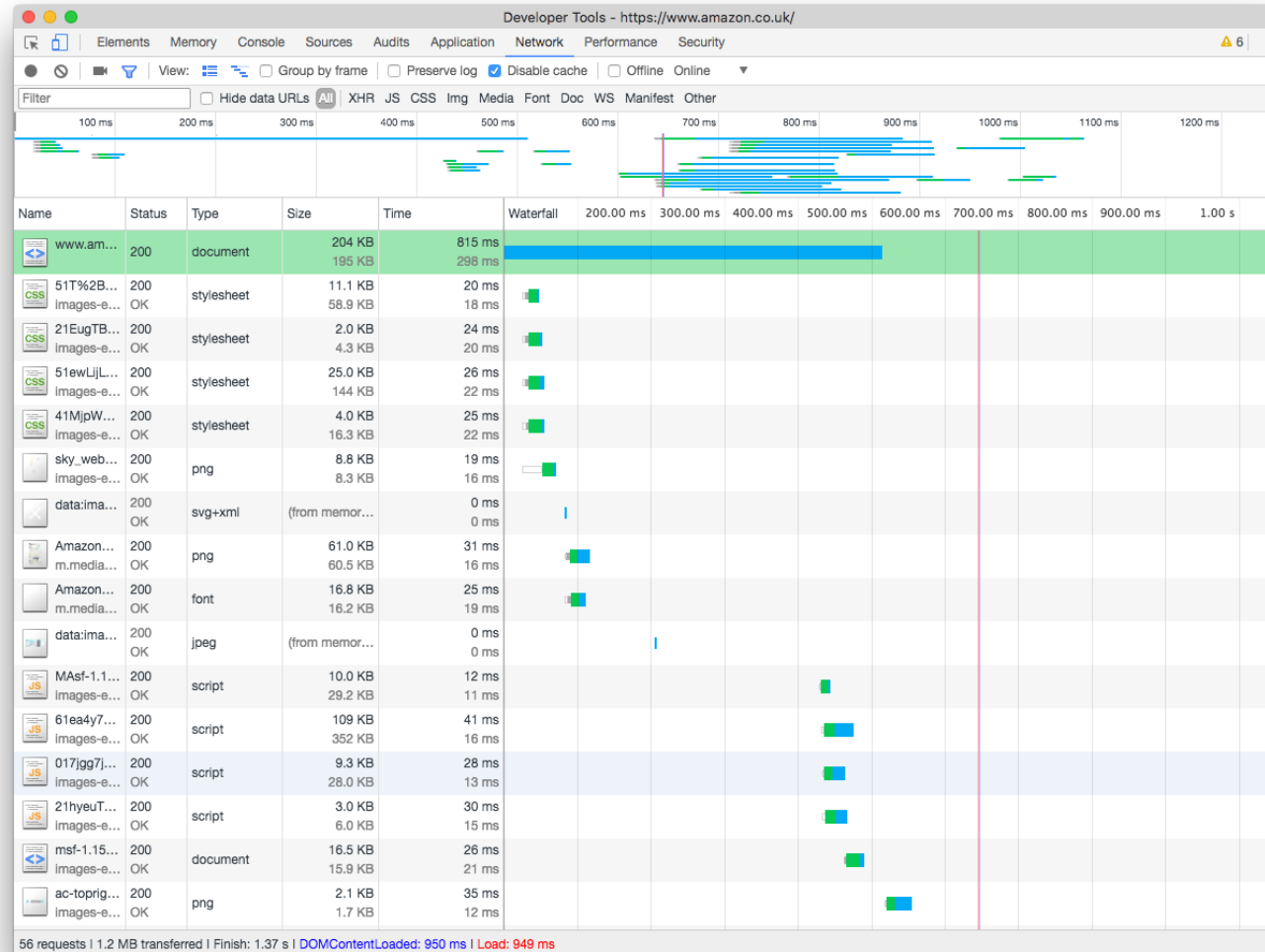
Definitions

- **Load time** *or* **document complete** time *or* **onLoad** time
 - All of the document's resources (i.e. images and CSS files, etc) have been fully loaded
 - Usually a site is interactive before this
 - May be when some key javascripts activate
 - Helps compare coarsely with other sites
 - Content might be *after the fold** (i.e. outside the viewport) but still loading

* Jakob Nielsen estimates that, *"...web users spend 80% of their time looking at information above the page fold. Although users do scroll, they allocate only 20% of their attention below the fold."*

Waterfalls a Common Visualisation Technique

Used in this course: Chrome DevTools



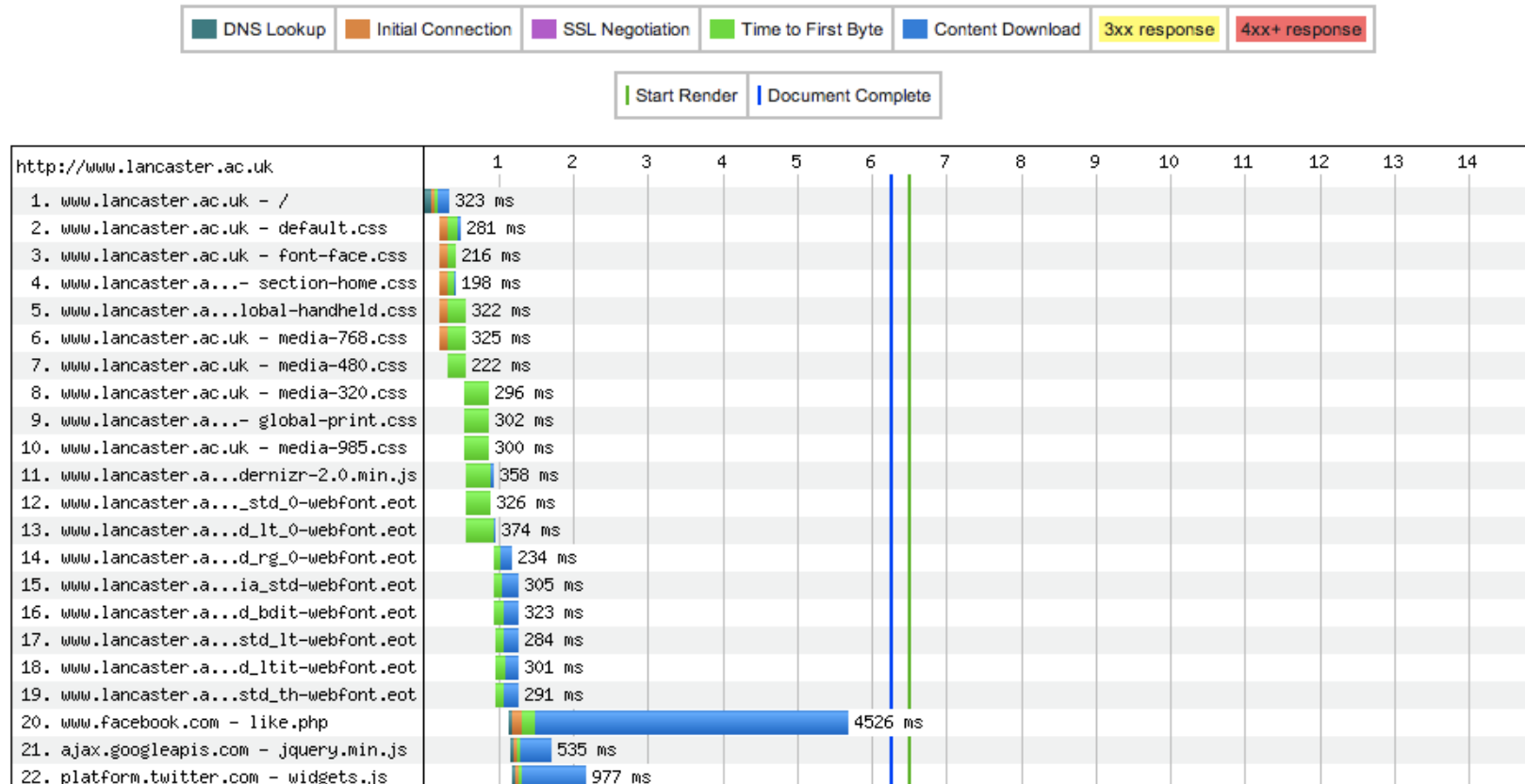
Drilling down on performance problems

So, where **exactly** is the problem?

Example page (with webpagetest.org)

Load Time	First Byte	Start Render	Visually Complete	Speed Index	DOM Elements	Result (error code)	Document Complete			Fully Loaded		
							Time	Requests	Bytes In	Time	Requests	Bytes In
6.242s	0.163s	6.486s	15.540s	12790	548	0	6.242s	88	1,421 KB	15.460s	111	1,694 KB

Waterfall View



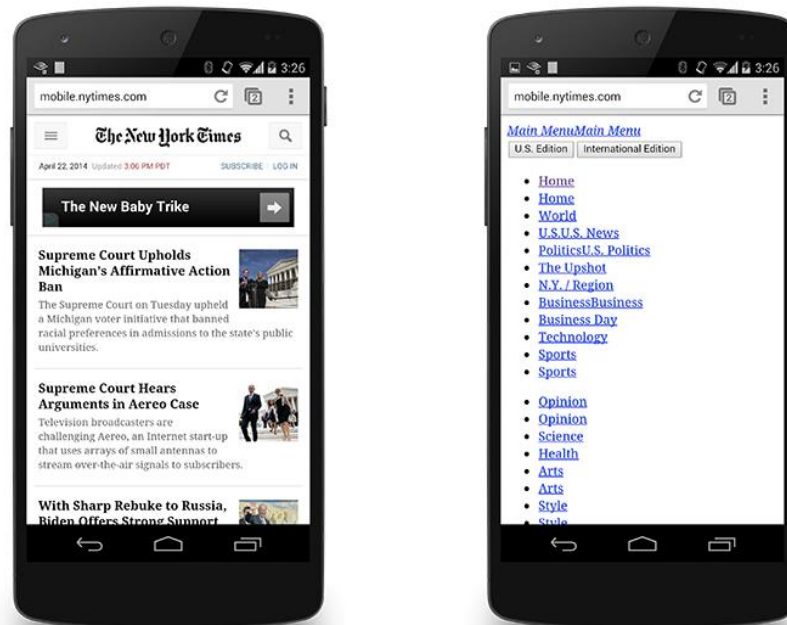
So, a good place to start is to *focus on render time*

Before we render

- The browser must *fetch any linked content* that is *required* to render the page
 - i.e. might effect the Document Object Model (**DOM**)
 - This includes style sheets (**CSS**)
 - And external JavaScript source files (**JS**)
 - External files depend **on the full network round trip** and can block **rendering**
 - By default, CSS is **render blocking** unless marked otherwise
 - Although this is changing: browsers evolving to animate changes

HTML and CSS are “render blocking” resources

- The HTML is obvious, since without the DOM we would not have anything to render
 - But the CSS?
- <https://web.dev/critical-rendering-path-render-blocking-css/>



How can we improve performance?

Tip 1 – Order is important

- Some browsers block rendering until the stylesheet is loaded to avoid redraws
 - CSS goes **in the head** so the DOM can start render more quickly
 - Javascript can often go **in the body** as you don't need the scripts until the page is available

Source Code: [Submit Code »](#)

```
<!DOCTYPE html>
<html>
<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

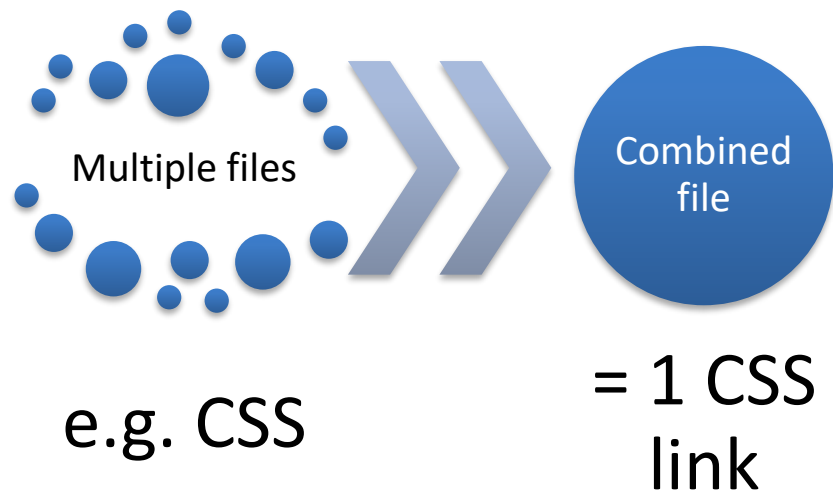
<p><strong>Note:</strong> myFunction is stored in an external file called
"myScript.js".</p>

<script src="myScript.js"> </script>

</body>
</html>
```

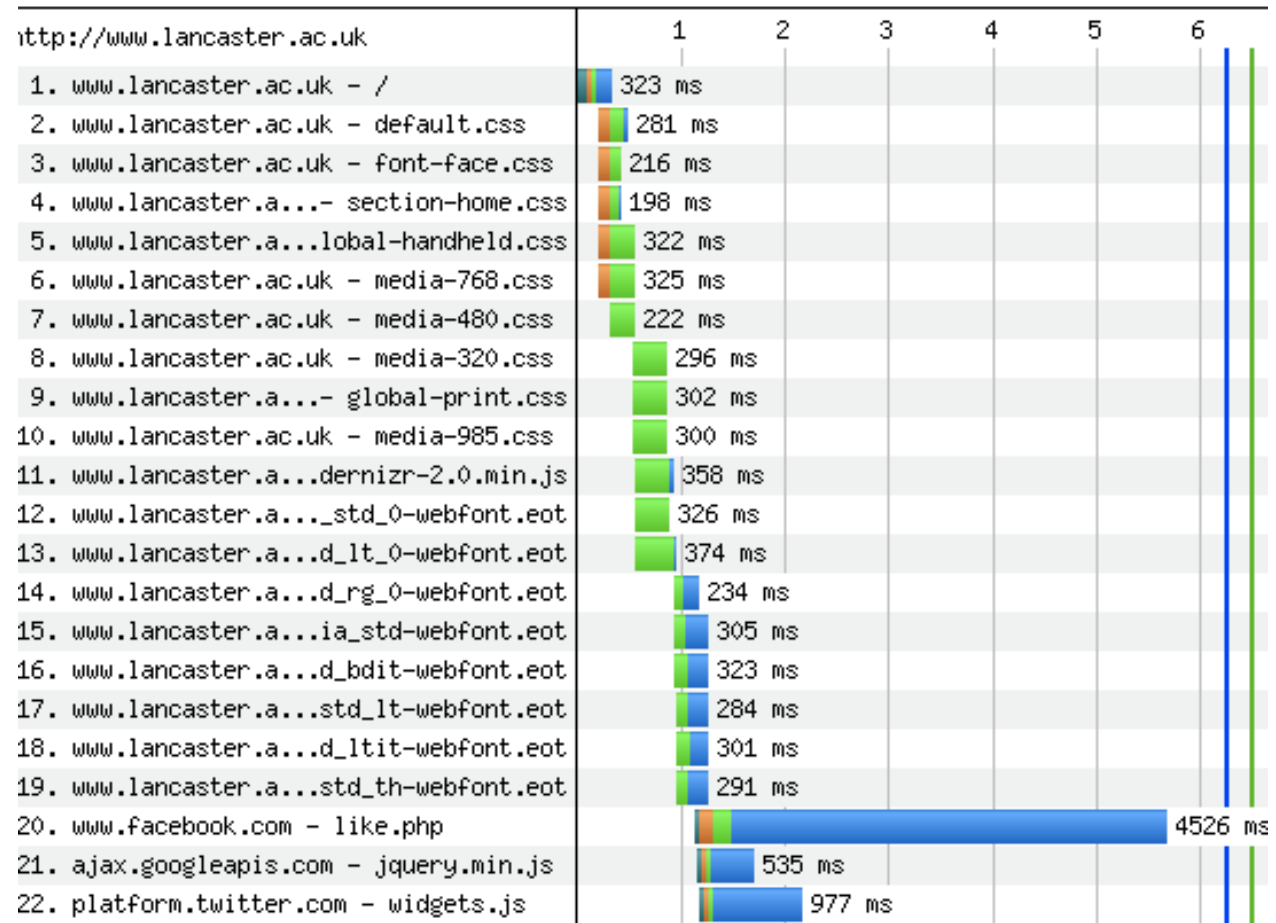
Tip 2 – do less

- A simpler page (simpler DOM) will be faster
 - Reduces number of page elements to load and process
- Also, hard to make TCP go faster, but can **combine files** (CSS, JS)
- Remove duplicates (e.g. redundant JS)!



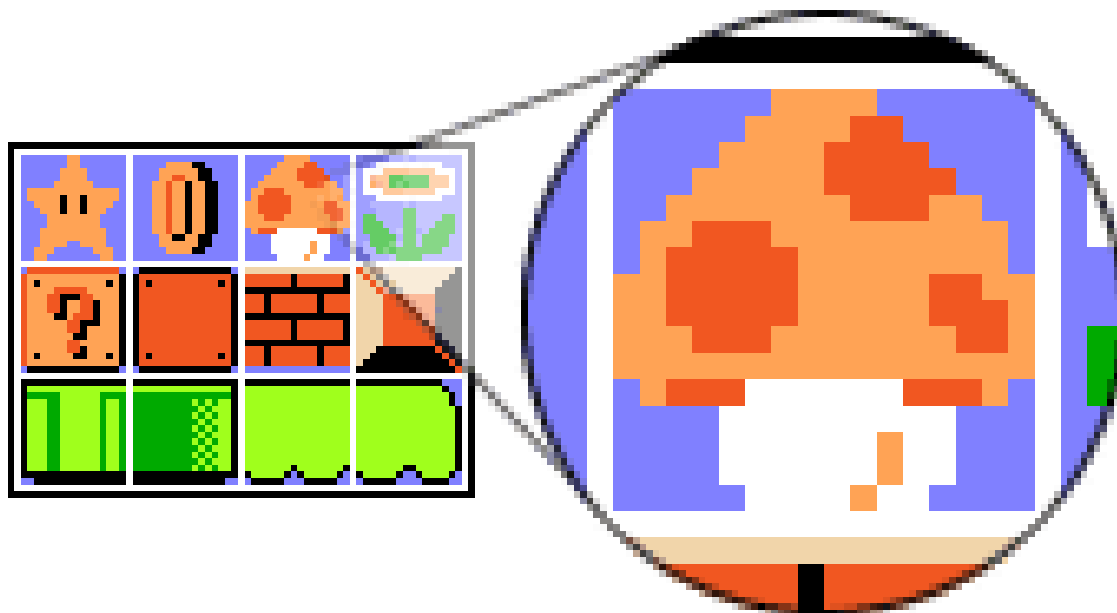
Note large collection of CSS, fonts and scripts blocking rendering

- Until the final stylesheet is loaded... **You want less rows!**



CSS Sprites

- We can combine multiple image files into a larger **image map** (sprite) that is then ‘dissected’ in CSS – you load one larger image, rather than many small ones, amortising overhead (see <http://bit.ly/XqV6dx>)



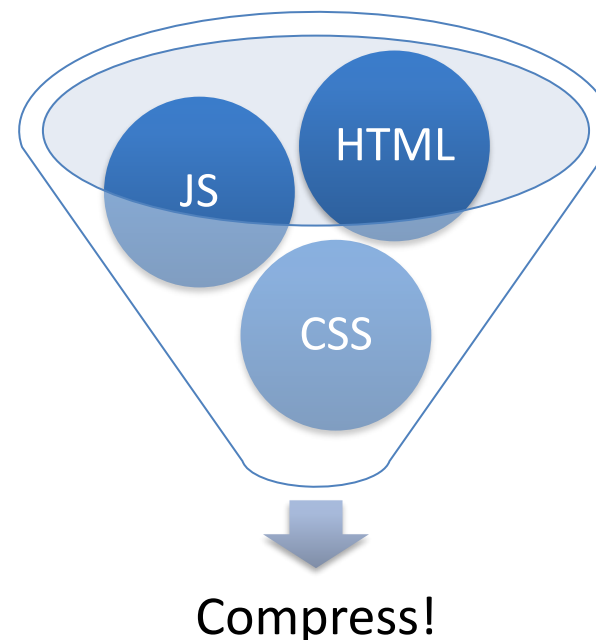
EXAMPLE SPRITES

Amazon example

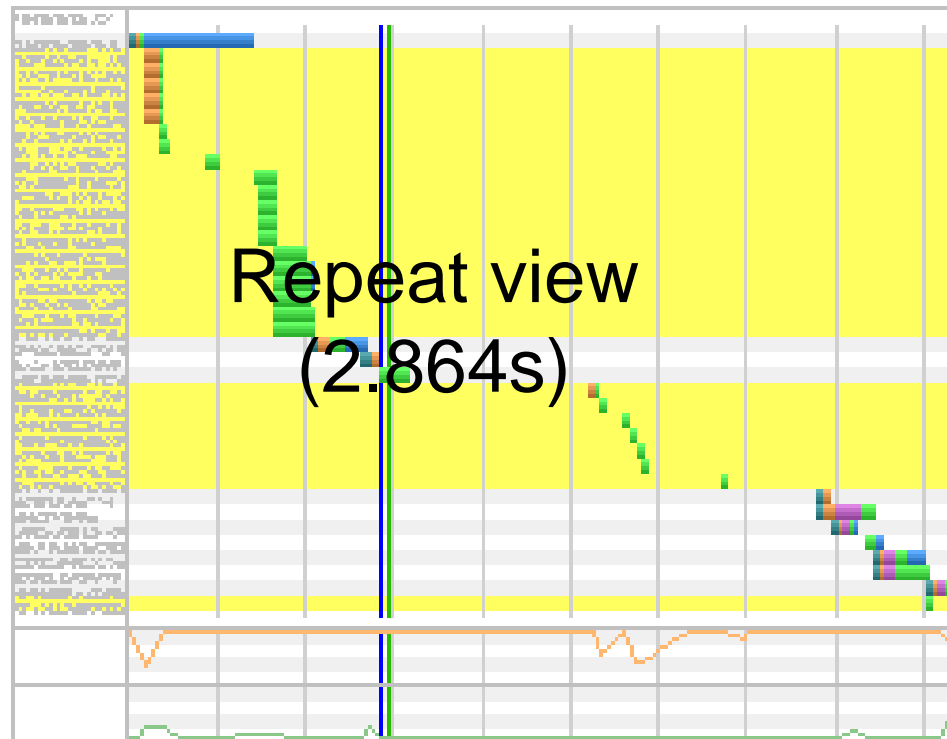
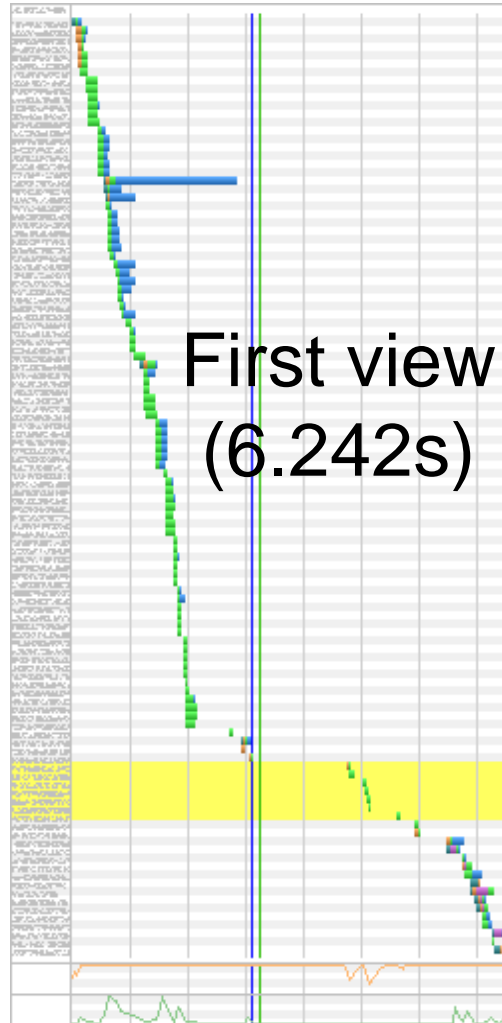


Tip 3 – make things smaller

- Optimise
 - Image sizes/files, make progressive
 - Enable compression (reduces response by 70%, in 90% of browsers!)
 - ‘Minify’ javascript and CSS (e.g. using Google [closure compiler](#)/ [YUI compressor](#) (+use ‘minified’ versions of JS libs)

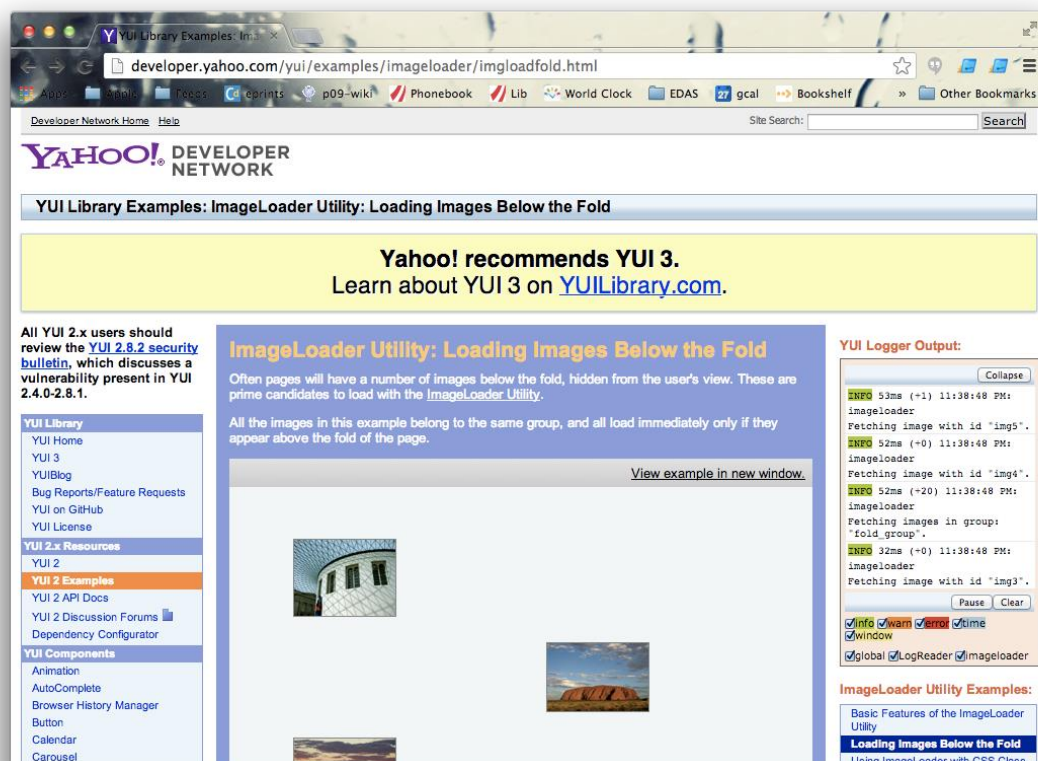


Tip 4 – support caching



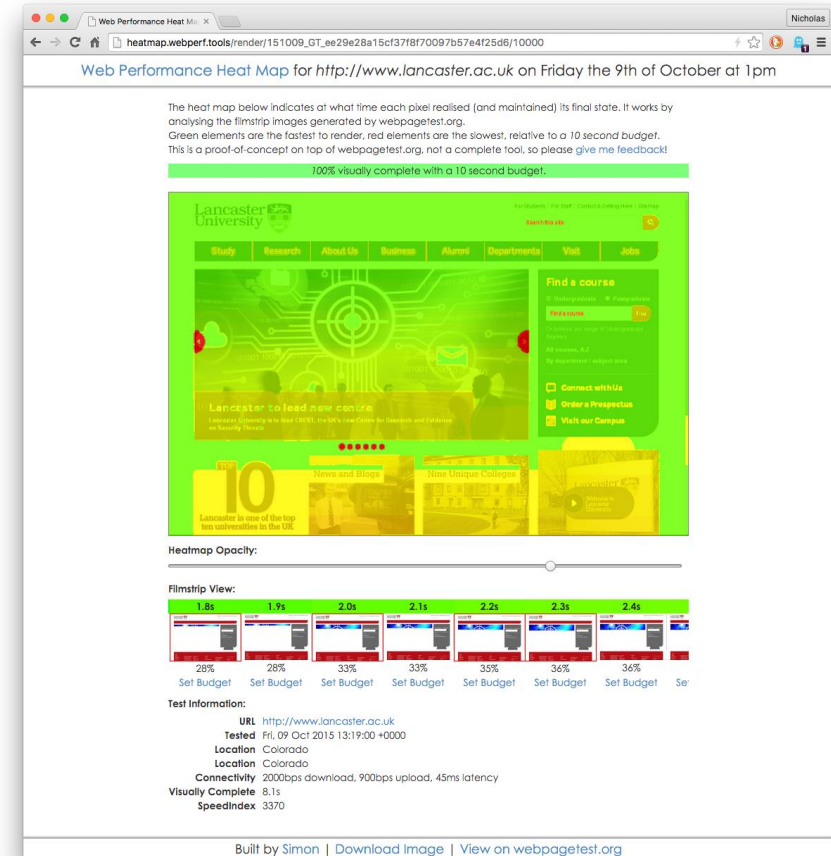
Tip 5 – loading after ‘the fold’

- Delay loading of components not visible to the user until they’re needed, e.g. images using **YUI ImageLoader**.



Other Tools

- <http://heatmap.webperf.tools>
- red means slow
- <http://yellowlab.tools>
– analyse performance bottlenecks
- <http://lab.speedcurve.com>
– visualise performance trends



Pointers for Getting Started

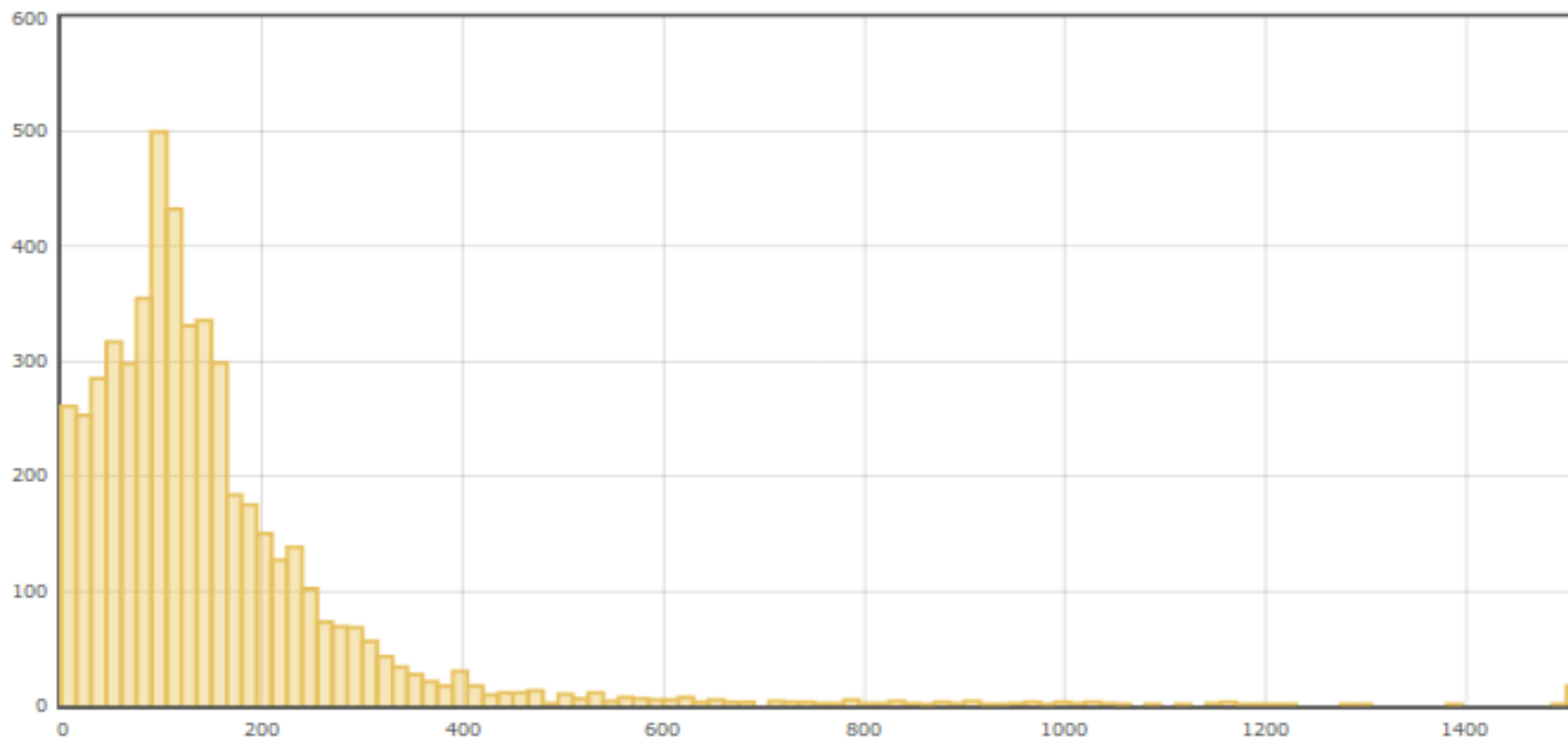
Intuition and Statistics

- Be warned:
 - Intuition about statistics can be very bad...
 - Don't base your measurements on:
 - **just one run** (**sampling error**)
 - from one location (typically local!)
 - Include measures of variability
 - Standard deviation/error
 - Confidence intervals
- Draw a histogram of responses



Intuition and Statistics

Graph:

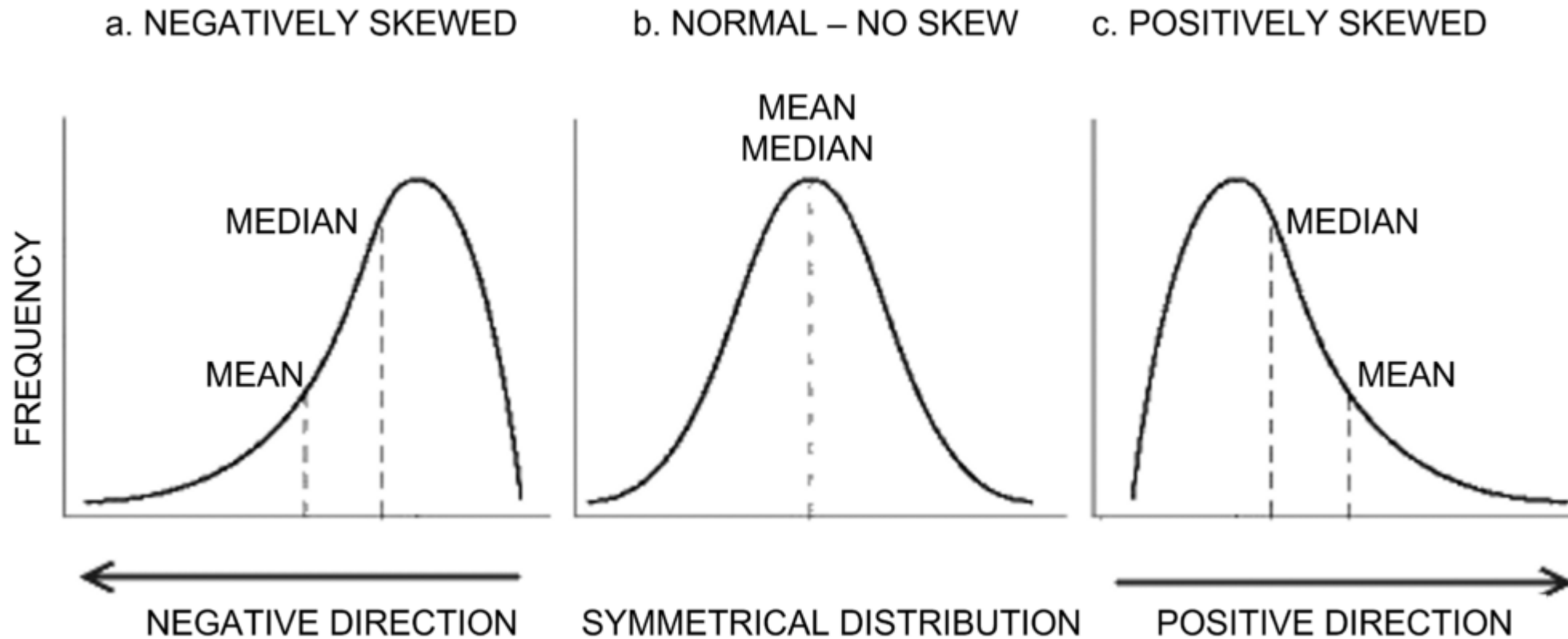


(The X axis is the server response in MS, and the Y axis is number of hits that fell into that response bracket)

Which descriptive statistic should you use?

-
- Typically you will consider using the average (or mean) and standard deviation
 - For some distributions the median will be a better choice
 - When a distribution has a long tail, the median can be better
 - Multiple ways to measure variability for median (e.g., interquartile range)
 - When evaluating performance, distributions typically have a long right tail

Differences between mean and median



Part III: Back-end Web Performance and Scaling



What is scale?

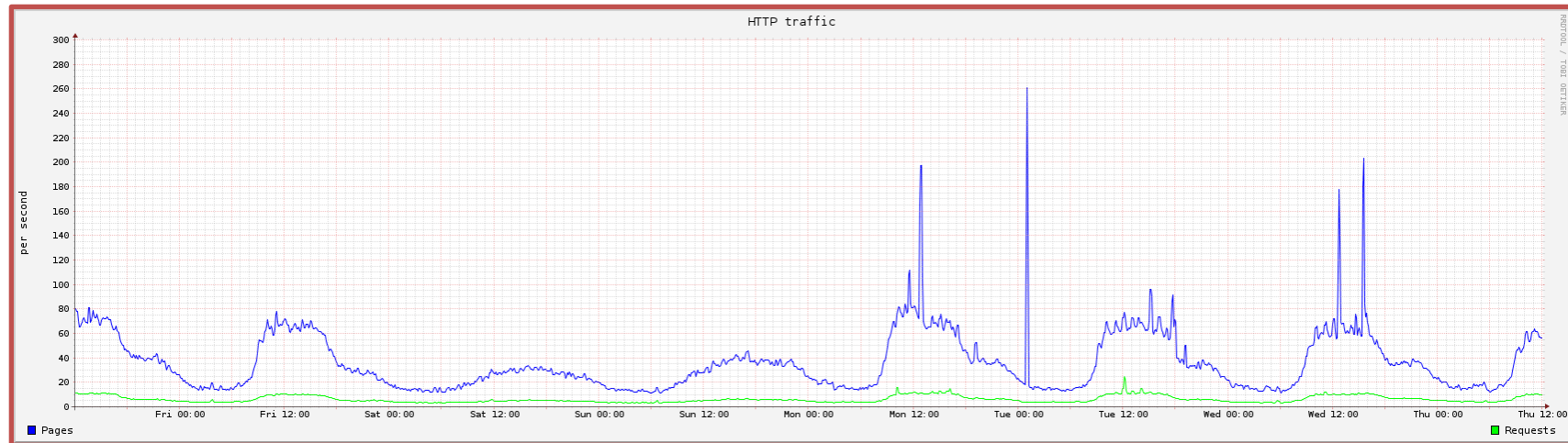
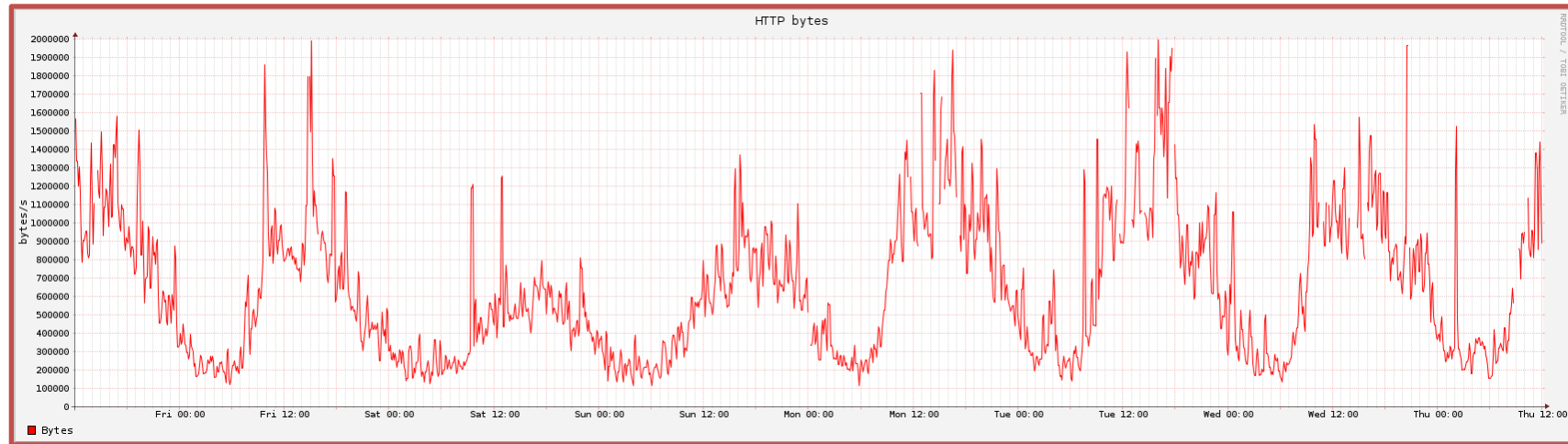
-
- The ability handle an increasing number of (possibly concurrent) requests
 - While maintaining performance targets
 - Low latency (short **time to first byte**)
 - High throughput (short **content download**)

Why is scale important?

-
- We've already seen the impact that page load times can have on performance
 - But what influence does the infrastructure have on this?
 - Consider the amount of people that now have access to the Internet
 - It is inherently global!
 - How do we build around this popularity?
 - Handling load *well* is key

Load changes over time!

Part of the reason that we take multiple measures (recall: *sampling bias*)



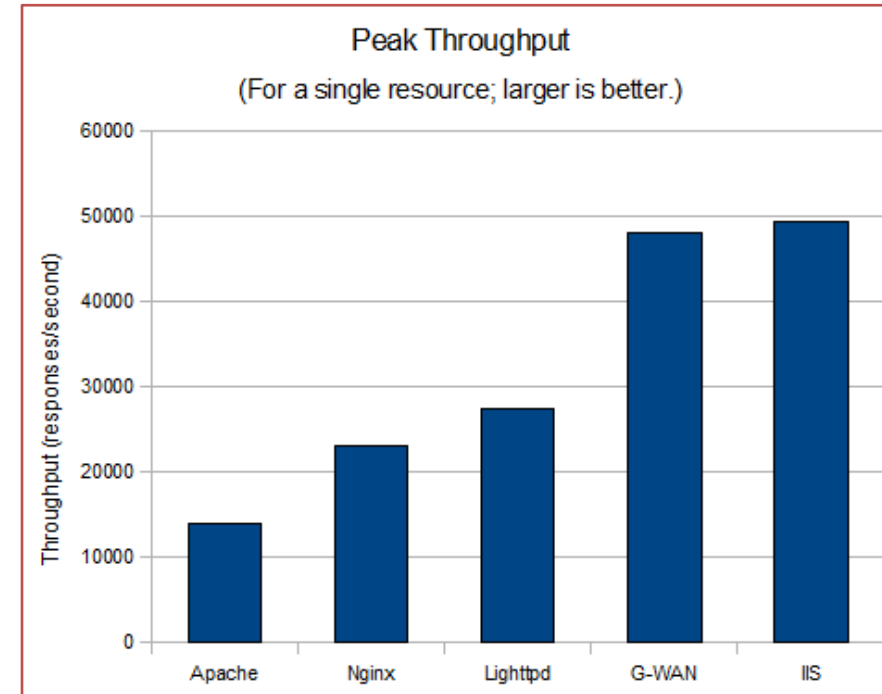
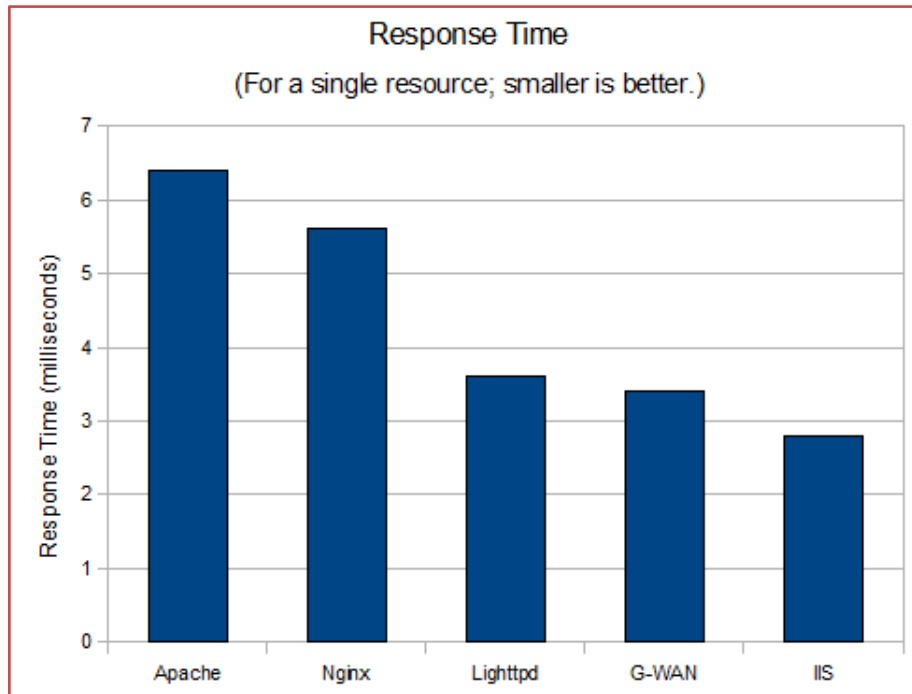
Metrics that matter

-
- Number of **requests served per second** (depends on type of request and **work/request!**)
 - **Latency (response time)** for each new connection or request (in **ms**)
 - **Throughput in bytes per second** (depends on file size, cached or not cached content, available network bandwidth...)

$$\text{Response time} \approx \frac{\text{concurrent sessions}}{\text{requests} \times \text{second}^{-1}}$$

Response time is inversely proportional to requests per second

We sometimes have to provision our system to trade these off. If we spend too much time beyond steady state, then we need to *do something about it!*



And not all web servers are the same

Web Performance, 'The fastest web server', 16.11.2011. <https://www.webperformance.com/load-testing-tools/blog/2011/11/what-is-the-fastest-webserver/>

$$\textit{Requests/sec} \approx \frac{\textit{connections}}{\textit{time}}$$

Requests per second is page requests / time

This can be gleaned from the logs.

$$\text{Bytes/sec} \approx \frac{\sum \text{size of requests}}{\text{time}}$$

Bytes per second is the sum of the size of each request / time

Again, this can be gleaned from the logs.

Optimise *before* scaling

Reducing server load (by far the easiest/cheapest option!)



Buying a bigger box is quick.
Redesigning software is not.

Spend months
federating your
database, or buy
loads of RAM? 😊

Vertical Scaling == bigger/ upgrades

More processors, more memory, gets exotic & expensive

Has the nice property that nothing drastically changes

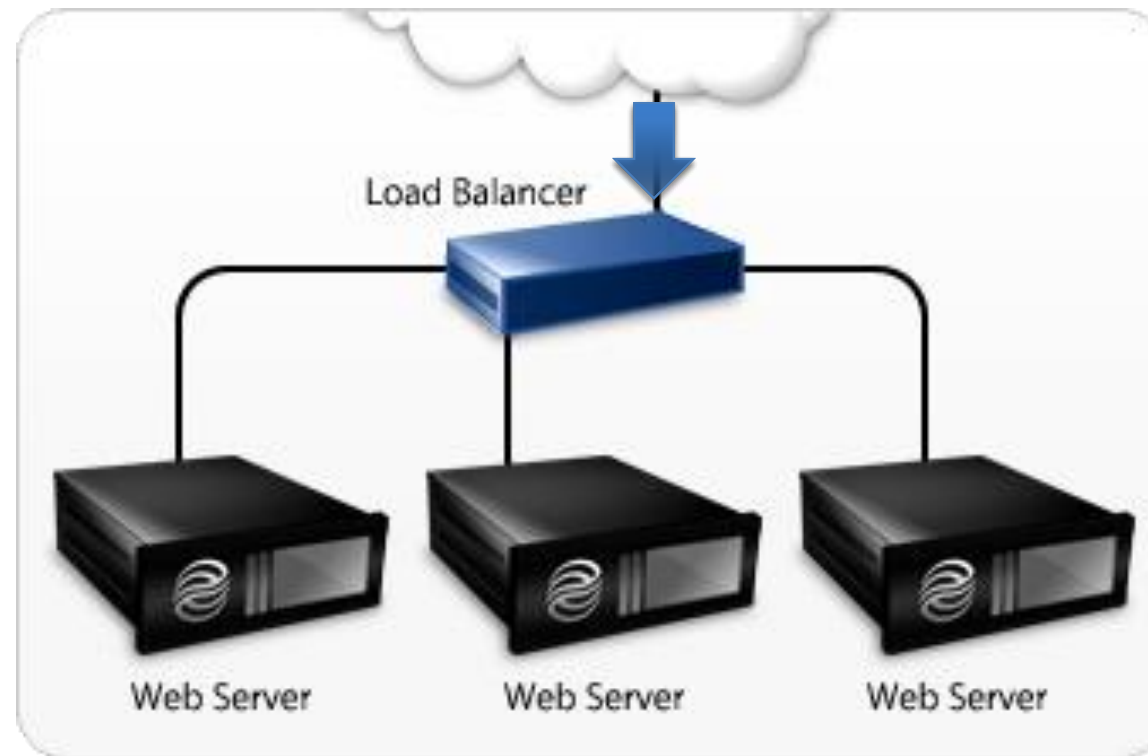


Horizontal Scaling == more hardware

But typically simpler and lower cost (the hardware). This will complicate all but the simplest of scenarios!

Load balancers

- Distribute the load evenly across two or more servers
 - High performance/cost hardware: Load Balancer
 - Rewrites headers at Layer 4/adjusts DNS
 - TCP connection between client and server (end-to-end)
 - Software ([perlbal](#), [nginx](#)) – ‘reverse proxy’
 - Echoes HTTP requests to particular servers (layer 7)
 - TCP connection between client and balancer



EXAMPLE OF A LOAD BALANCED HOSTING ENVIRONMENT

<http://www.liquidweb.com/>

Replication

All servers store the same set of content. Read-only things, 'static resources', scale well.

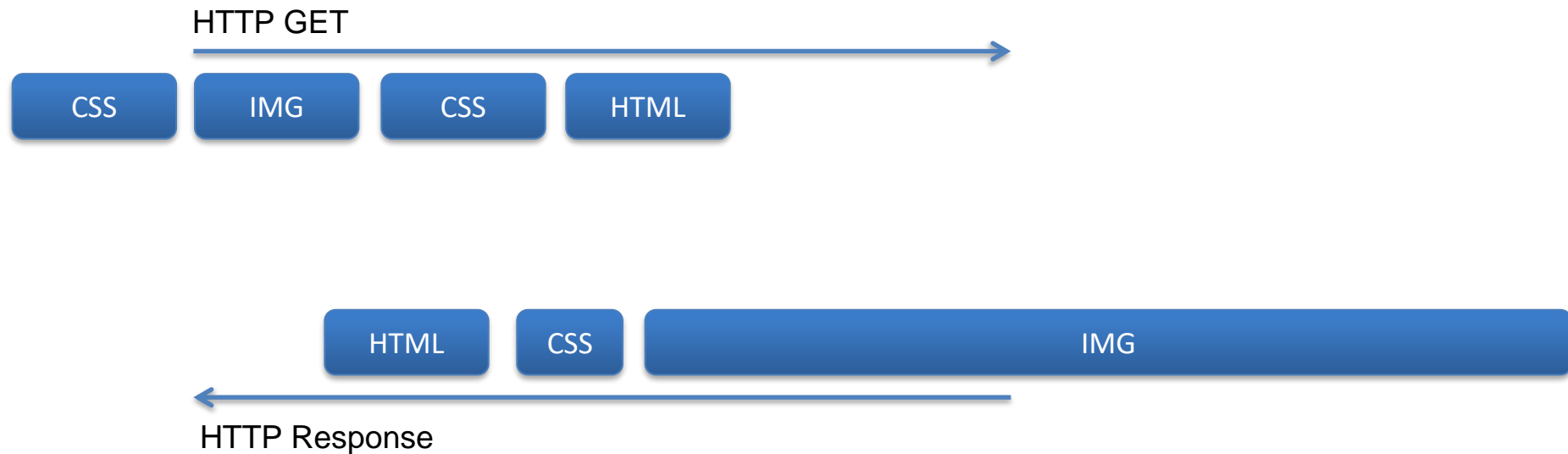
What replicates well?

- Things that don't change very often
 - Media assets (images, movies, sounds)
- But also self-contained tasks:
 - Image conversion
 - Audio transcoding
 - Video transcoding
 - Web crawling
 - Compute

Session tracking and state

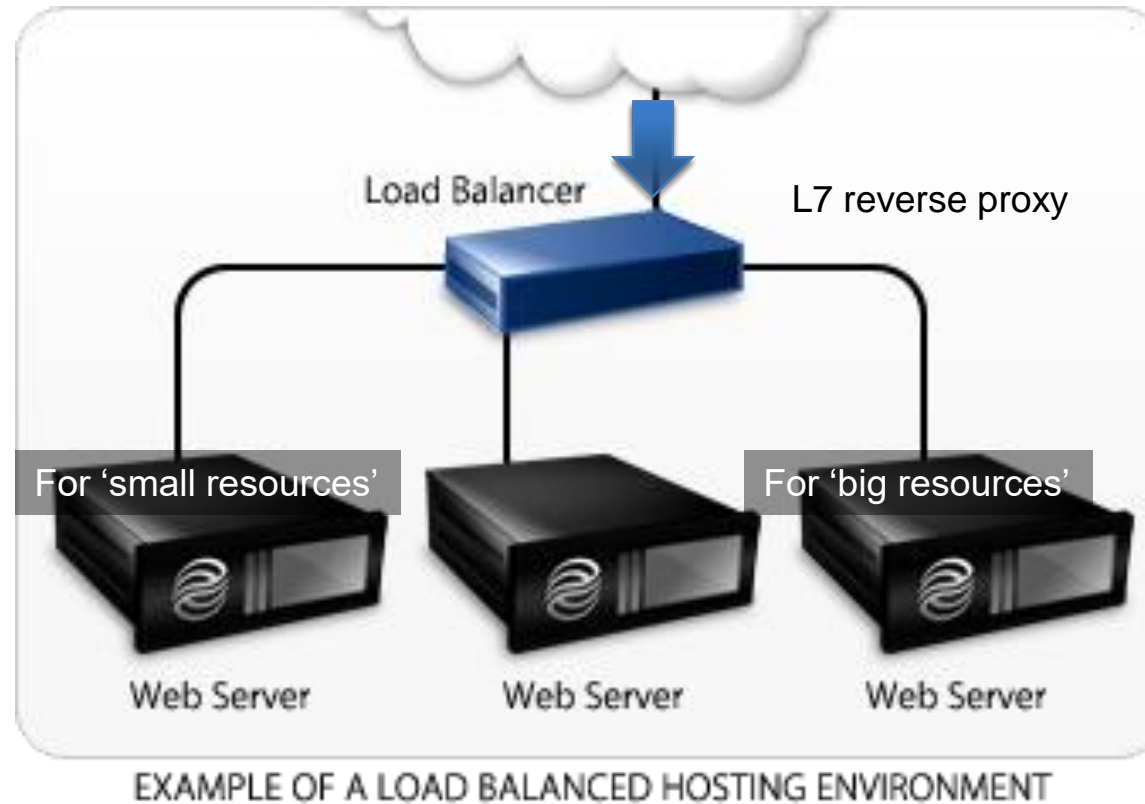
- Fly in the ointment of scaling: personalisation
 - User sessions, e.g. state of interaction, shopping basket etc.
 - Sessions are state that is built up between the client and the server
 - Load balancers have ‘affinity’ to certain servers, but *this defeats the object!*
- Also: User generated content
 - OK, some of it may go viral
 - But most of it is highly specialised and interesting only to a small group of people
 - Is it worth caching?

HTTP 1.1 Pipelines requests to the same server in order



The response to the 2nd CSS request is 'blocked'

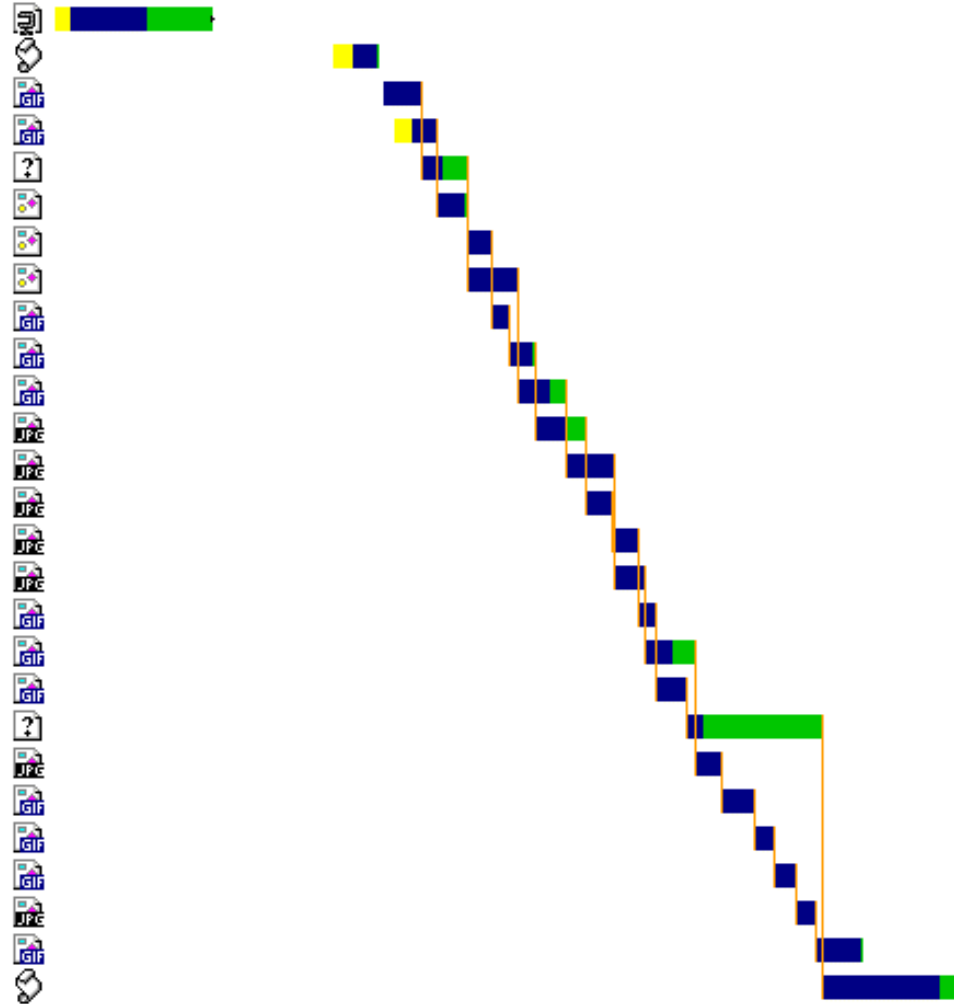
Requests are serviced in order... so the small requests have to wait



<http://www.liquidweb.com/>

Distribution

Split requests across groups of servers. Serve the essentials for rendering the page quickly. Let the large media items fetch more slowly from other servers.



Domain Sharding

<http://www.stevesouders.com/>

Different server name = different connection! YouTube uses i1.ytimg.com, i2.ytimg.com, i3.ytimg.com, and i4.ytimg.com. Live Search uses ts1.images.live.com, ts2.images.live.com, ts3.images.live.com, and ts4.images.live.com...

HTTP/2

- Multiplexed streams
- HTTP header compression
- Servers can *push* content

<https://caniuse.com/http2>

HTTP/2 protocol - OTHER

Global usage 96.95% + 0.51% = 97.46%

Networking protocol for low-latency transport of content over the web. Originally started out from the SPDY protocol, now standardized as HTTP version 2.

Chrome	Edge	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS	Samsung Internet	Opera Mini	Opera Mobile	UC Browser for Android	Android Browser	Firefox for Android
4 - 40		3.1 - 8	2 - 35	9 - 27									
41 - 50	12 - 18	9 - 10.1	36 - 52	28 - 37			3.2 - 8.4	4					
51 - 105	79 - 105	11 - 15.6	53 - 104	38 - 90	5.5 - 10		9 - 15.6	5 - 17.0		10 - 12.1		2.1 - 4.4.4	
106	106	16.0	105	91	11	105	16.0	18.0	all	64	13.4	105	104
107 - 109		16.1 - TP	106 - 107				16.1						

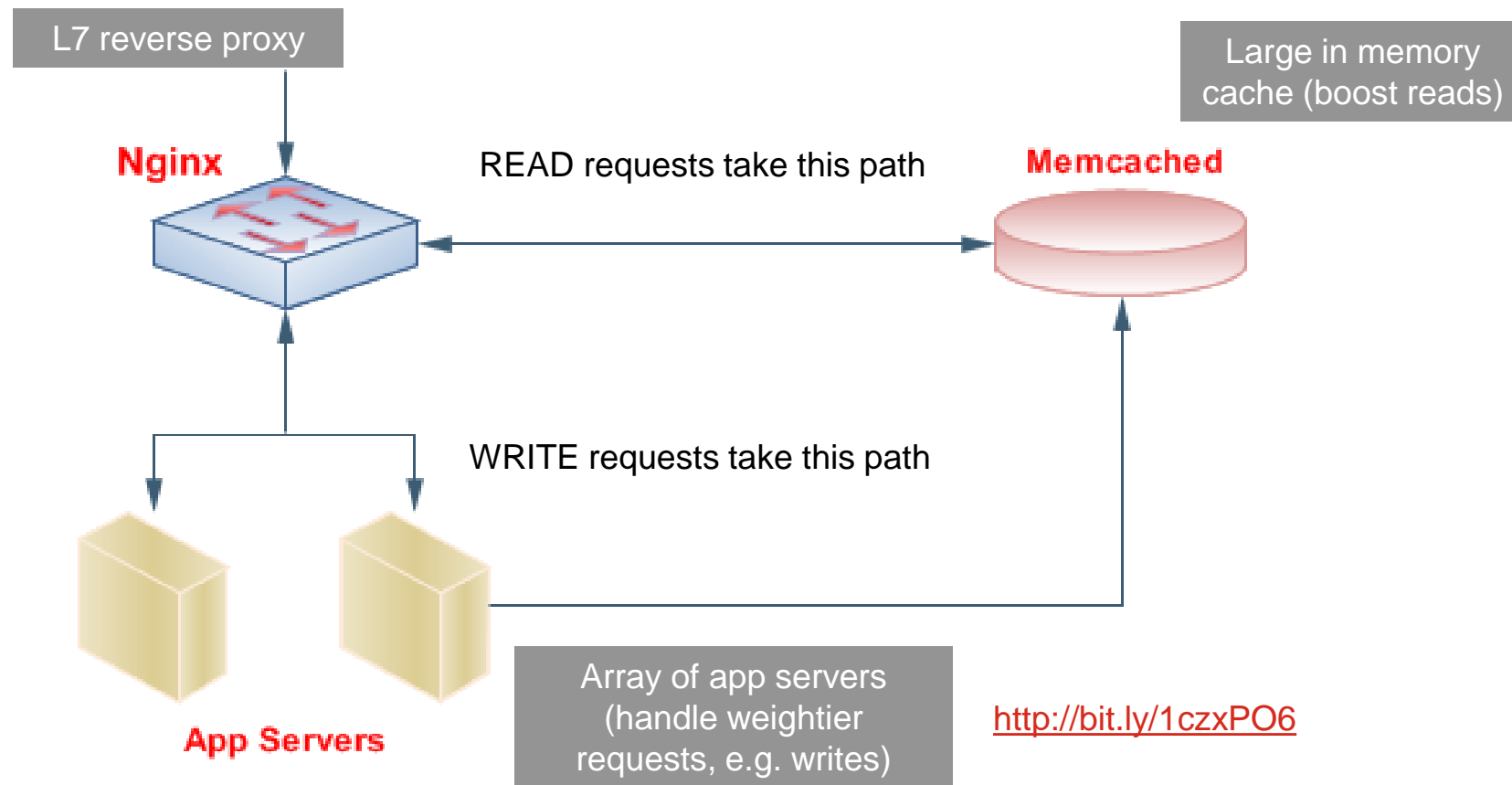
Where is the bottleneck?

How can we avoid it?

Caching Exploits Locality

-
- Move content (web pages, videos, images, etc.) closer to the user
 - Fewer hops == less latency (lower transport and in network cost)
 - Hopefully less narrow pipes between you/them
 - Or less congestion (e.g. the transatlantic link)

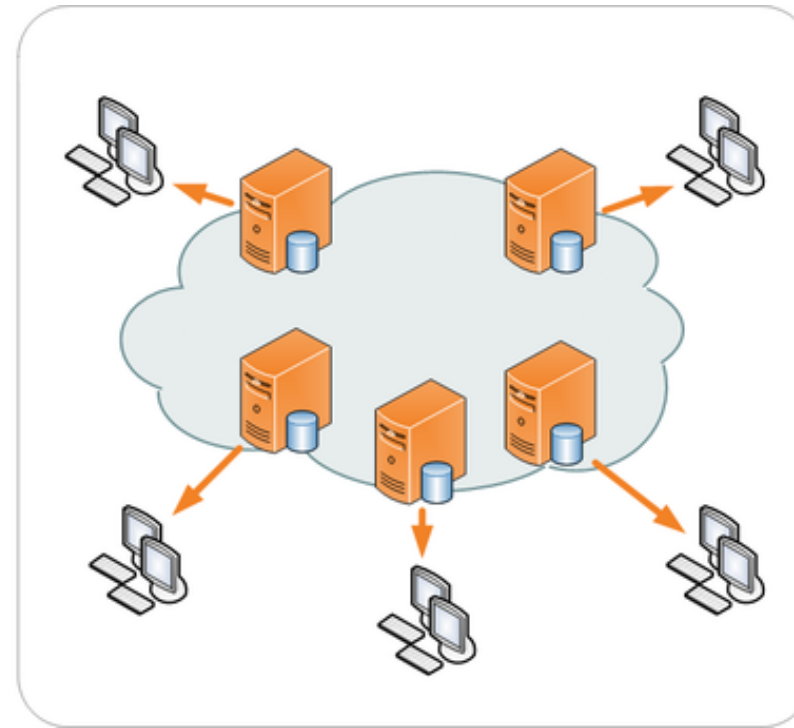
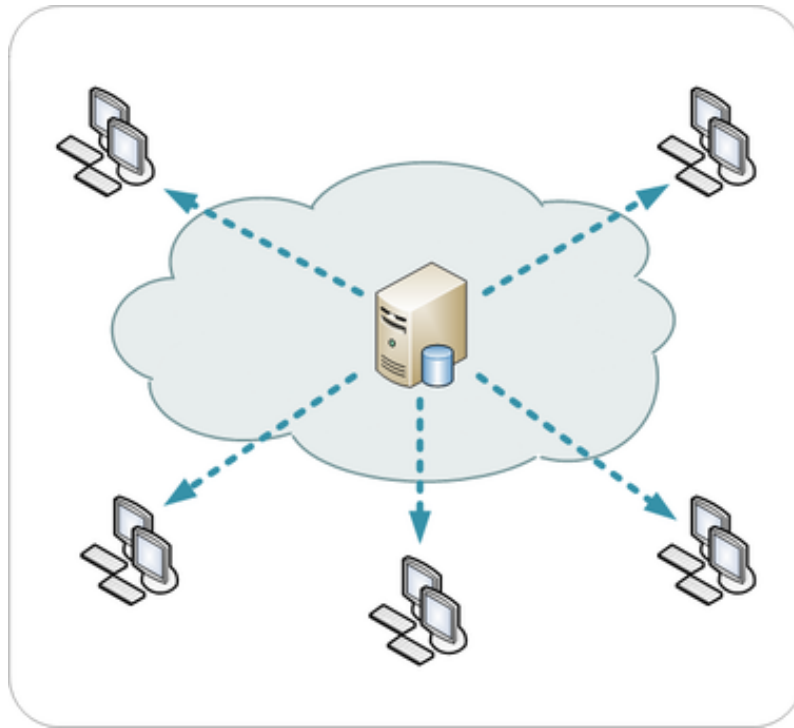
 - Caching can be done:
 - In-browser
 - In-network
 - At the server



(At the server) Caching

Optimising read performance of infrequently updated or static content helps a lot!

Content Delivery Networks (CDNs)



Infrastructure as a Service (IaaS)

- It's hardware, only in software
 - Machines are 'virtual machine instances'
 - Can be provisioned for various roles (frontend, backend, memcached, compute intensive); anything really!
 - Can be re-provisioned (vertical scaling)
 - Can be replicated (horizontal scaling)
 - Can be created and destroyed (almost) instantly
 - Can be connected however you want
- **Inherently flexible and scalable!**
- These are the services that allow start-ups to scale massively, without requiring their own datacentre

Platform as a Service (PaaS)

- Deploy **services instead of servers** (e.g. Google App/Engine)
 - Standalone and replicated data stores (SQL and NoSQL)
 - Arbitrary web applications (e.g. in Python, Java or Go)
 - Store data (with backup)
 - Compute
 - Content Delivery Network!

Compute

Storage

Networking

Big Data

Services

Management

Advantages of PaaS

-
- Resilience to hardware and network failures (99.999% reliability)
 - A run-time proportion of CPU, memory, storage, network IO (specifiable configurability)
 - On-demand scalability ('elasticity')
 - Lower cost – and pay per use...
 - Global availability

Disadvantages of PaaS

-
- Latency is no longer guaranteed; my machines could be anywhere around the globe!
 - Interoperability issues (moving cloud providers?)
 - Hidden (on) costs (hidden supplier charges or even suppliers!) – transnational borders (e.g. US)
 - Data recovery (can they?)

Lecture Summary

- Performance bottlenecks (at the client)
 - Latency and where it comes from
 - How lots of files, resources and when they're loaded delays rendering
 - Tips for avoiding blocking and load delays, inc. reducing connections, exploiting caching and compression
- Performance bottlenecks (at the server)
 - Where do scaling related bottlenecks come from
 - Metrics for detecting scale problems
 - What types of scaling up can we do
 - Strategies for optimising web architectures

Further reading

- <http://developer.yahoo.com/performance/rules.html>
 - 35 best practices for optimising the performance of your website
- <http://yslow.org/user-guide/>
 - 23 performance rules (also open-source site analyser, c.f. Google's PageSpeed)
- <https://web.dev/critical-rendering-path/>.
 - Google's 'critical rendering path' tutorial