CS 188/219

Scalable Internet Services

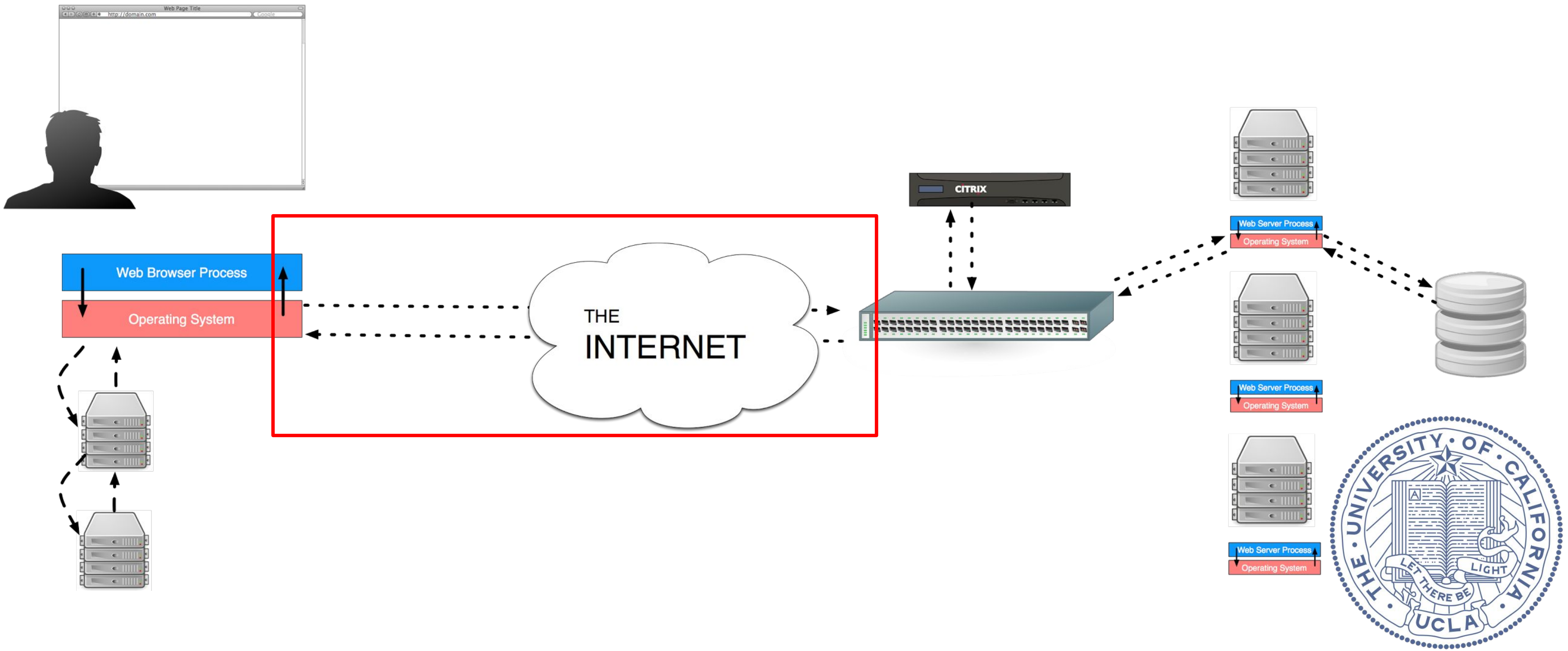
Andrew Mutz

Dec 1, 2015

Agenda

- HTTP/2
- QUIC
- Course Conclusion

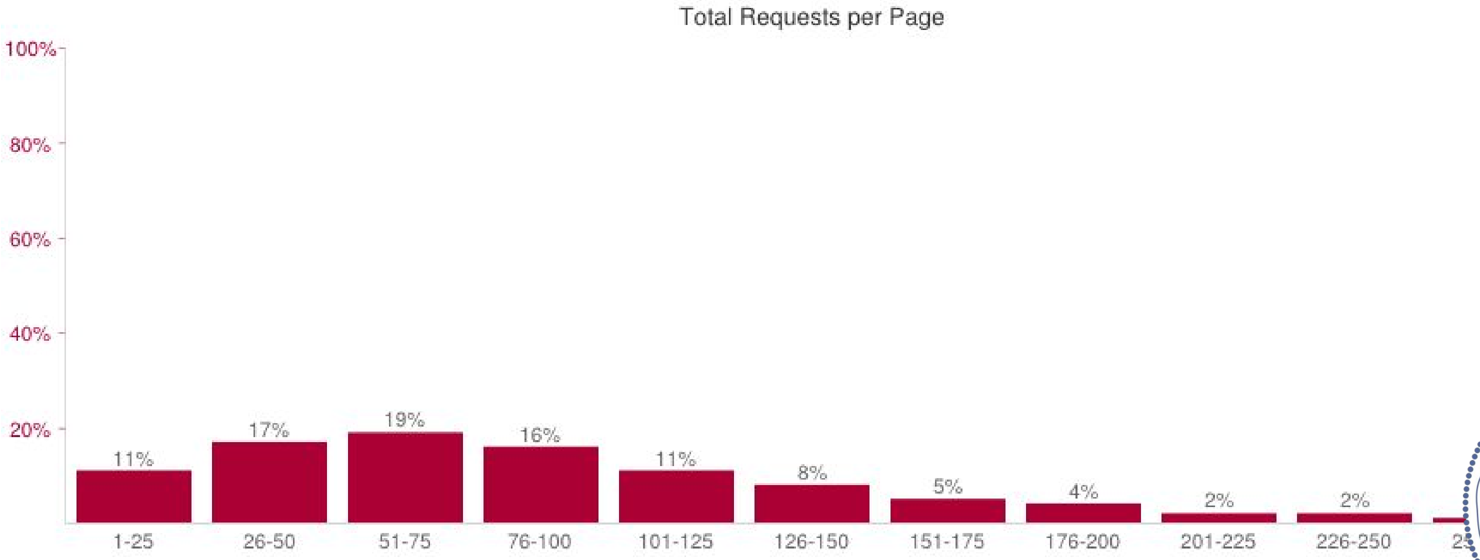
Motivation



Web Page Title
http://domain.com
Web Browser Process
Operating System
THE
INTERNET
CITRIX
Web Server Process
Operating System
Web Server Process
Operating System
Web Server Process
Operating System
THE UNIVERSITY OF CALIFORNIA
LET THERE BE LIGHT
UCLA

Today's Performance Problems

Web pages have many constituent resources



Source: http://httparchive.org/

Today's Performance Problems

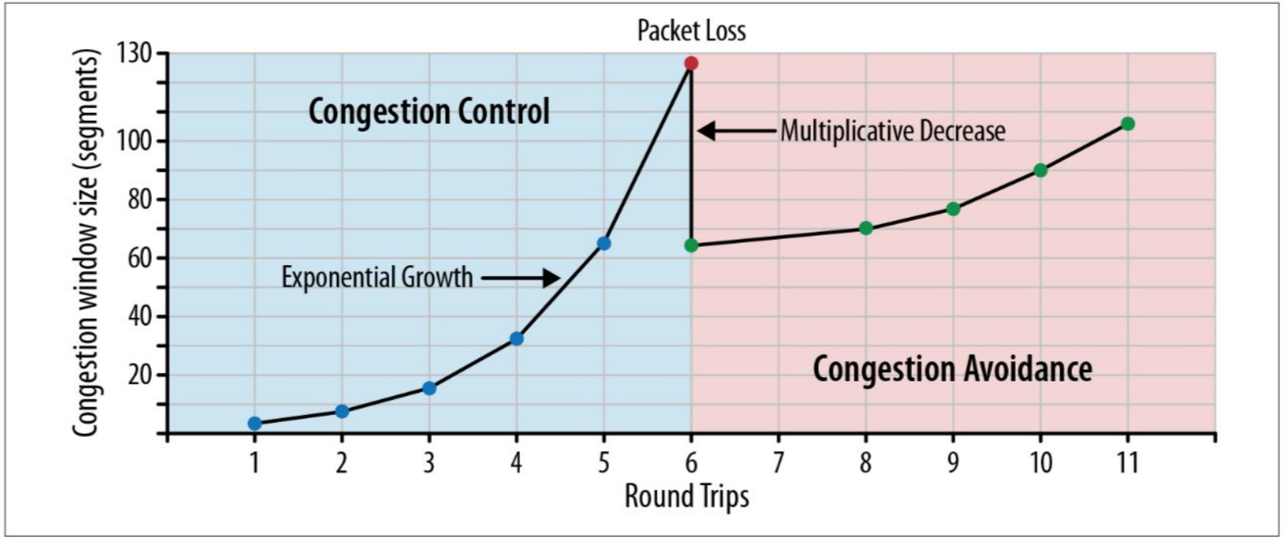
Many requests are needed to present today's web pages.

- CSS
- Javascript
- Images

Establishing many TCP connections to serve all these is very slow. Why?

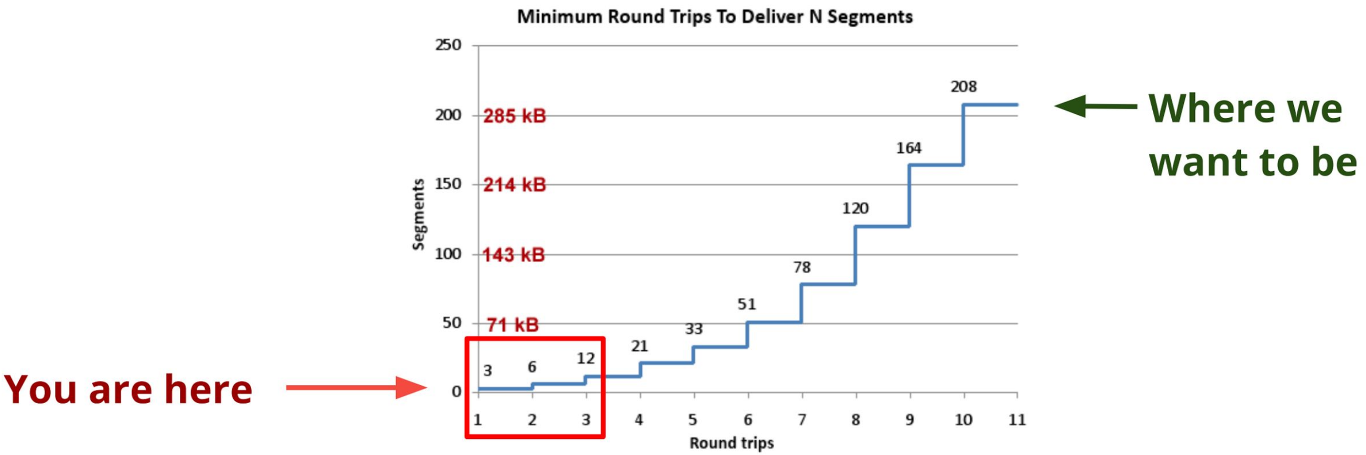
Today's Performance Problems

Establishing many TCP connections to serve all these is very slow.



Today's Performance Problems

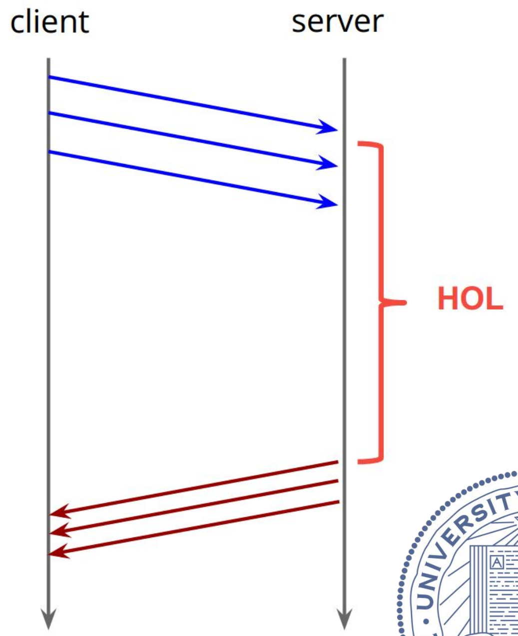
TCP was designed for long-lived flows. HTTP is short and bursty.



Today’s Performance Problems

HTTP Keepalive was introduced to help (and it does), but there are problems.

- We can reuse a TCP socket for multiple HTTP requests, but one heavyweight request can affect all others
- This is called Head-of-line blocking



Today’s Performance Problems

Additionally, if you look at the data that is being sent, there is a lot of repetition.

```
GET / HTTP/1.1
Host: www.etsy.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/536.26.14 (
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
DNT: 1
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: uaid=uaid%3DVdhk5W6sexG-_Y7ZBeQFa3cq7yMQ%26_now%3D1325204464%26_slt%3Ds_L
Connection: keep-alive
```

525 bytes

Today’s Performance Problems

Additionally, if you look at the data that is being sent, there is a lot of repetition.

```
GET /assets/dist/js/etsy.recent-searches.20121001205006.js HTTP/1.1
Host: www.etsy.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/536.26.14 (
Accept: */*
DNT: 1
Referer: http://www.etsy.com/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: autosuggest_split=1; etala=111461200.1476767743.1349274889.1349274889.1349
Connection: keep-alive
```

226 new bytes; 690 total

Today's Performance Problems

Additionally, if you look at the data that is being sent, there is a lot of repetition.

```
GET /assets/dist/js/jquery.appear.20121001205006.js HTTP/1.1
Host: www.etsy.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/536.26.14 (
Accept: */*
DNT: 1
Referer: http://www.etsy.com/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: autosuggest_split=1; etala=111461200.1476767743.1349274889.1349274889.134
Connection: keep-alive
```

14 new bytes; 683 total

Today’s Performance Tricks

Many techniques are used to reduce the number of requests.

- **Why does the asset pipeline exist in Rails?**

Today's Performance Tricks

Many techniques are used to reduce the number of requests.

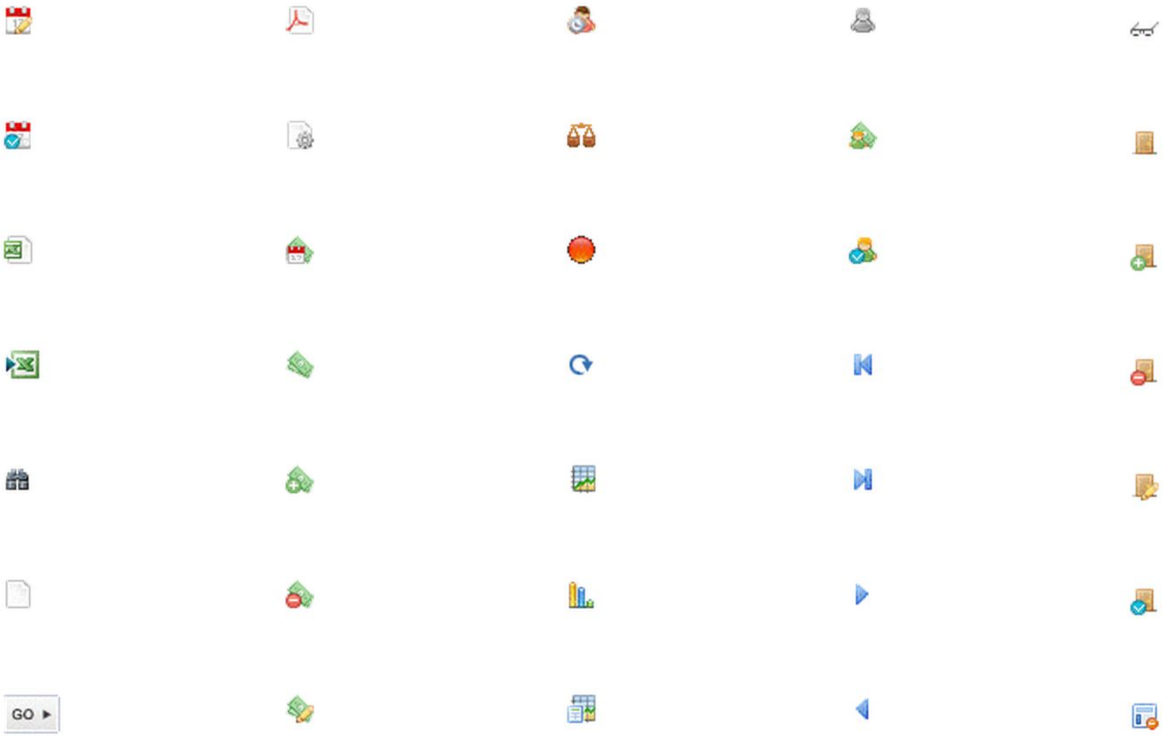
- Why does the asset pipeline exist in Rails?
  - File concatenation.
  - Having many JS, CSS files means having many requests, so we mash them together.

Today’s Performance Tricks

Many techniques are used to reduce the number of requests.

- Image Spriting: the process of putting lots of smaller images in a single image, and then referring to them all using offsets.

Today's Performance Tricks



Today's Performance Tricks

Many techniques are used to reduce the number of requests.

- Image Spriting: the process of putting lots of smaller images in a single image, and then referring to them all using offsets.
  - This spriting is cumbersome to deal with, and is really a hack.

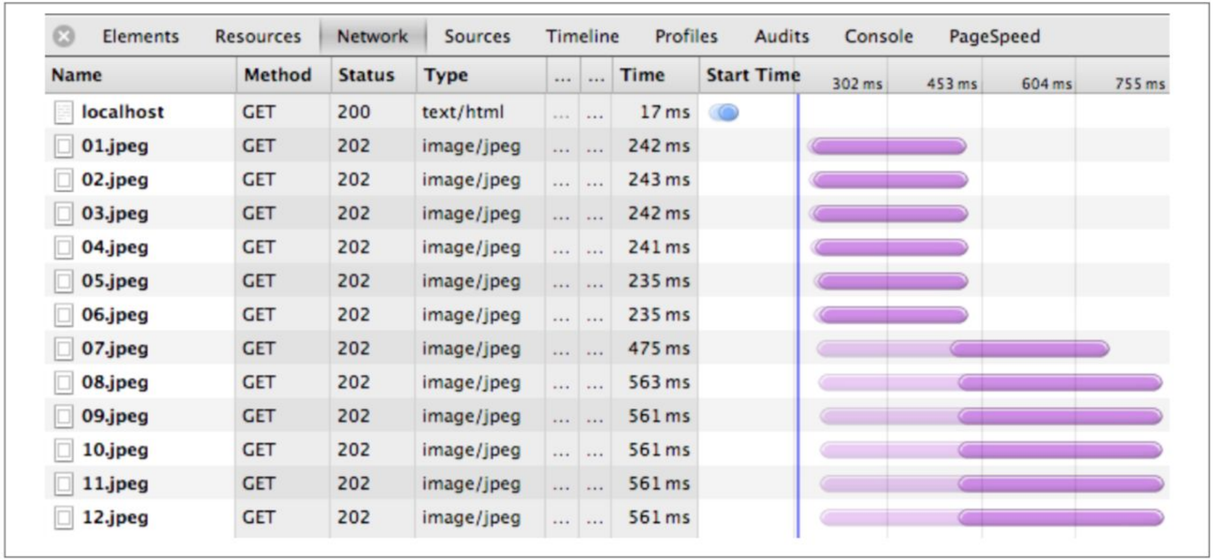
Today's Performance Tricks

Techniques are used to increase the number of parallel requests that browsers can have to a server.

- Most browsers will only open 6 concurrent TCP connections to a single host
  - Why?

Today’s Performance Tricks

This is the result:



Elements Resources Network Sources Timeline Profiles Audits Console PageSpeed

Name	Method	Status	Type	...	...	Time	Start Time 302 ms 453 ms 604 ms 755 ms
localhost	GET	200	text/html	...	...	17 ms	
01.jpeg	GET	202	image/jpeg	...	...	242 ms	
02.jpeg	GET	202	image/jpeg	...	...	243 ms	
03.jpeg	GET	202	image/jpeg	...	...	242 ms	
04.jpeg	GET	202	image/jpeg	...	...	241 ms	
05.jpeg	GET	202	image/jpeg	...	...	235 ms	
06.jpeg	GET	202	image/jpeg	...	...	235 ms	
07.jpeg	GET	202	image/jpeg	...	...	475 ms	
08.jpeg	GET	202	image/jpeg	...	...	563 ms	
09.jpeg	GET	202	image/jpeg	...	...	561 ms	
10.jpeg	GET	202	image/jpeg	...	...	561 ms	
11.jpeg	GET	202	image/jpeg	...	...	561 ms	
12.jpeg	GET	202	image/jpeg	...	...	561 ms	

Today’s Performance Tricks

How do we address this?

- We want fewer TCP connections, but…
  - we don’t want head-of-line blocking
  - we don’t want to have to jam all our css, js artificially together
  - we don’t want to have to stuff our images in one big file and deal with offsets everywhere
  - we don’t want to have to do DNS tricks to fool the browser

HTTP/2

We address this with HTTP/2

- Started life at Google as SPDY
  - Added to Chrome in 2009
- Pushed towards standardization starting in 2012
- Today supported by Chrome, Firefox, Safari
  - IE is working on it
- Server side support optional in Apache and Nginx

Standard completed Feb 17.

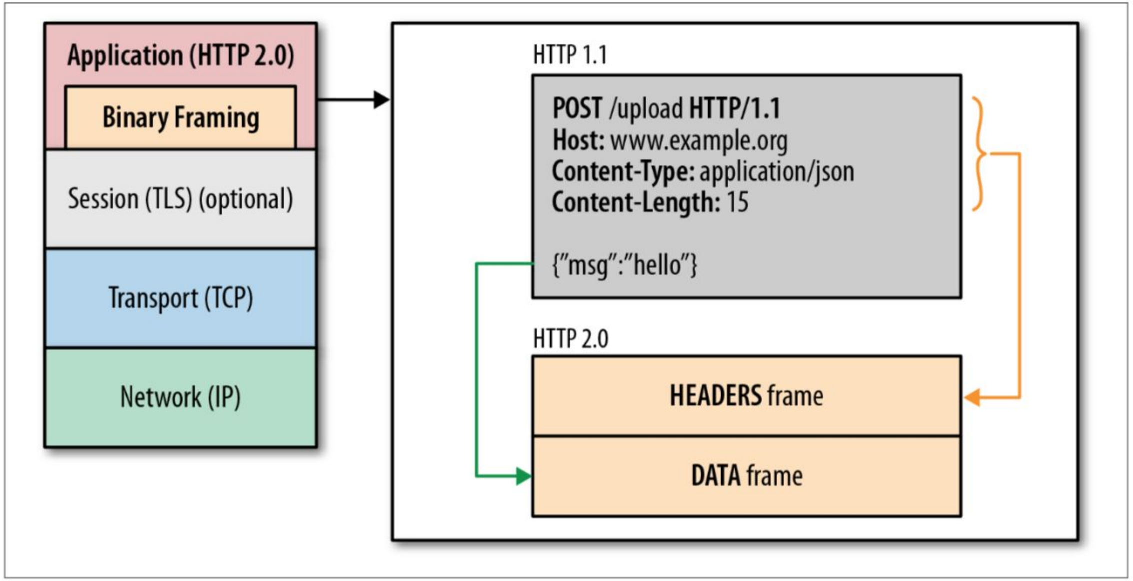
HTTP/2

How does HTTP/2 work?

- One TCP connection, multiplex everything over that
- Header compression
- Server push
- Prioritization

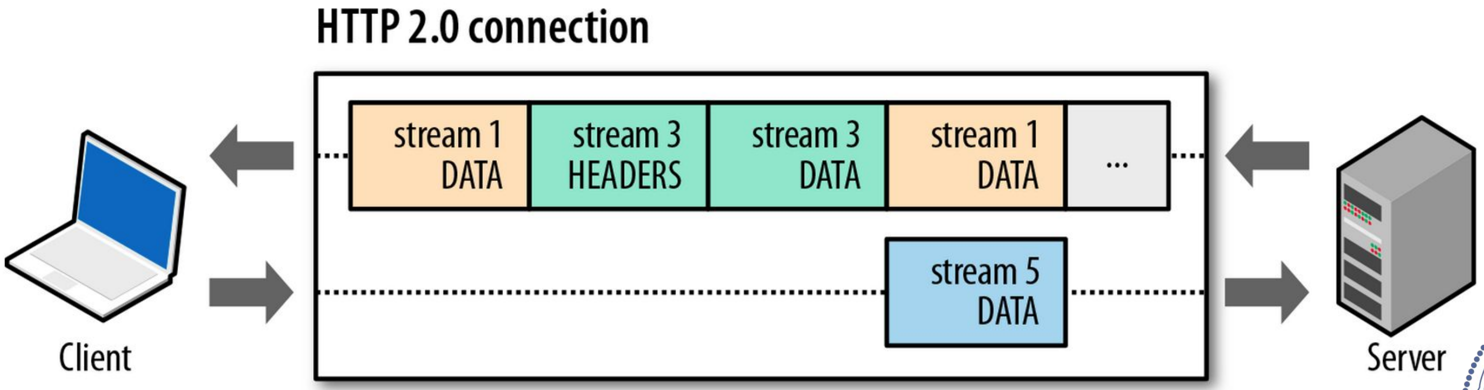
HTTP/2

Single TCP stream: Binary Framed connection



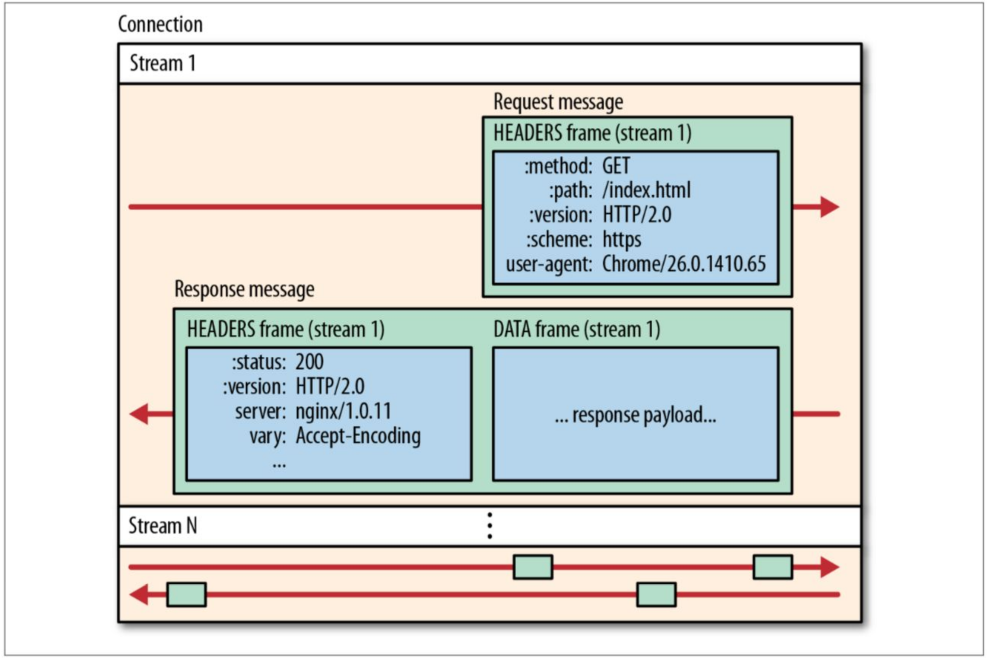
HTTP/2

Single TCP stream: Binary Framed connection



HTTP/2

Single TCP stream: Binary Framed connection

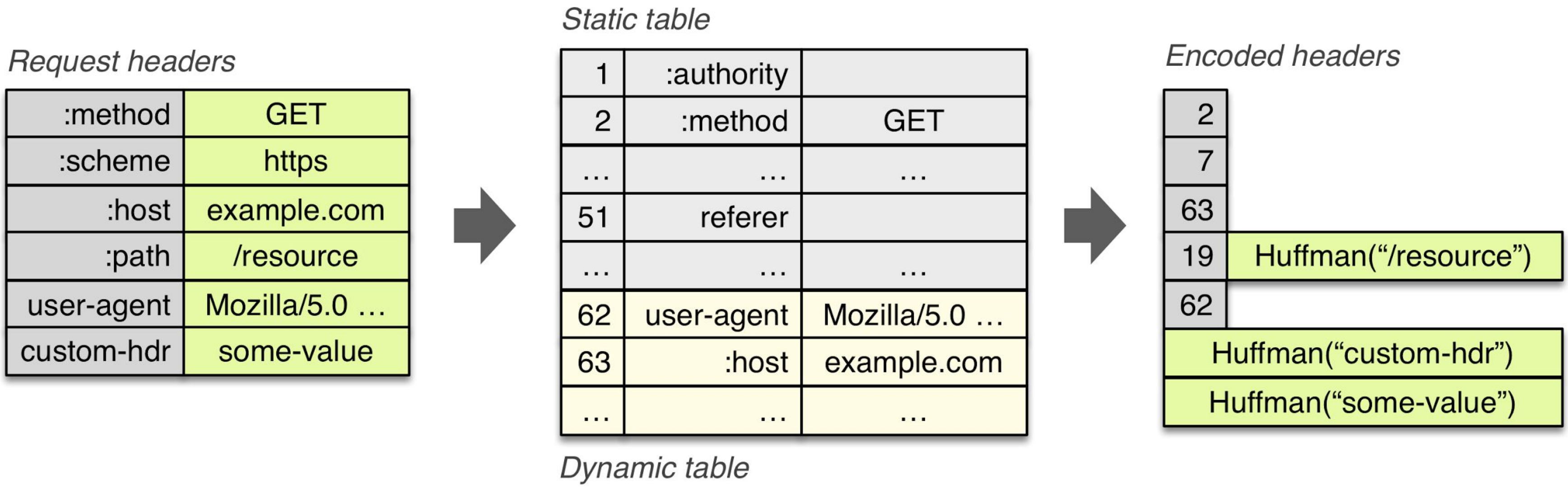


HTTP/2

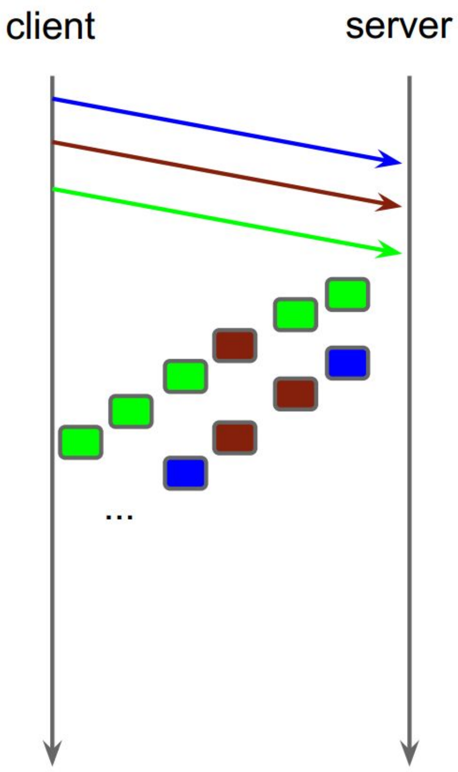
With binary framing, header compression becomes easy.

- Initially implemented with GZIP, but CRIME attack revealed weaknesses
  - If an attacker can inject data, compressed size can reveal information
- Now uses HPACK
  - More coarse-grained

HTTP/2


Request headers
:method GET
:scheme https
:host example.com
:path /resource
user-agent Mozilla/5.0 …
custom-hdr some-value
Static table
1 :authority
2 :method GET
… … …
51 referer
… … …
62 user-agent Mozilla/5.0 …
63 :host example.com
… … …
Dynamic table
Encoded headers
2
7
63
19 Huffman("/resource")
62
Huffman("custom-hdr")
Huffman("some-value")

HTTP/2



Binary framing means ordering of resources is flexible.

- Handling of many small resources is efficient
- Headers are compressed, so they are lightweight
- Head of line blocking no longer exists
- No TCP setup burden

HTTP/2

Prioritization & Flow Control

- Since order is no longer mandated, we can implement prioritization
  - DOM highest priority, followed by CSS, Javascript
  - Images lowest

WINDOW_UPDATE flag exists to control number of frames “in flight”

HTTP/2

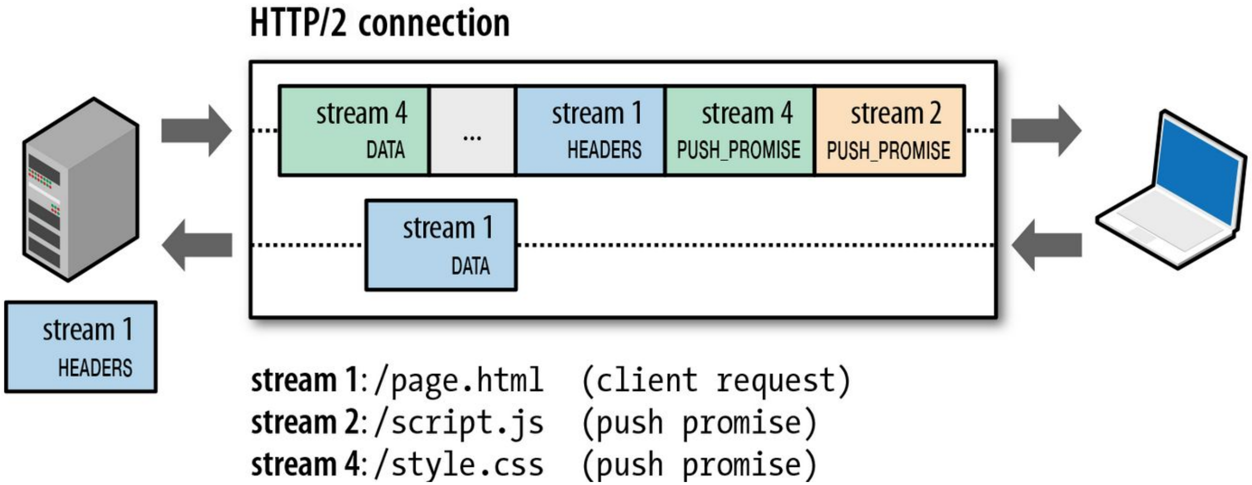
The hacks of the last 20 years can be thrown out:

- Spriting & asset compliation not needed since we can handle many small resources
- Domain sharding irrelevant since all content is delivered over a single connection

HTTP/2

Server push is now possible

- When a resource is requested, the server can proactively send additional resources using PUSH_PROMISE
- Client can indicate it doesn't want the additional content (e.g. it's cached)



HTTP/2

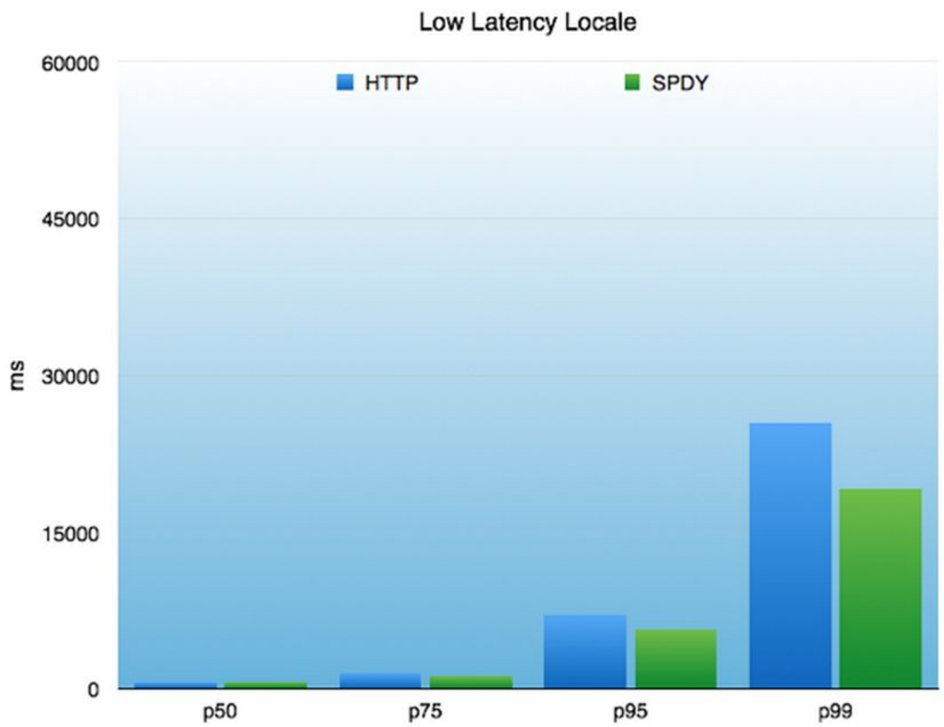
Results: much faster!

	Google News	Google Sites	Google Drive	Google Maps
Median	-43%	-27%	-23%	-24%
5th percentile (fast connections)	-32%	-30%	-15%	-20%
95th percentile (slow connections)	-44%	-33%	-36%	-28%

time from first request byte to onload event in the browser

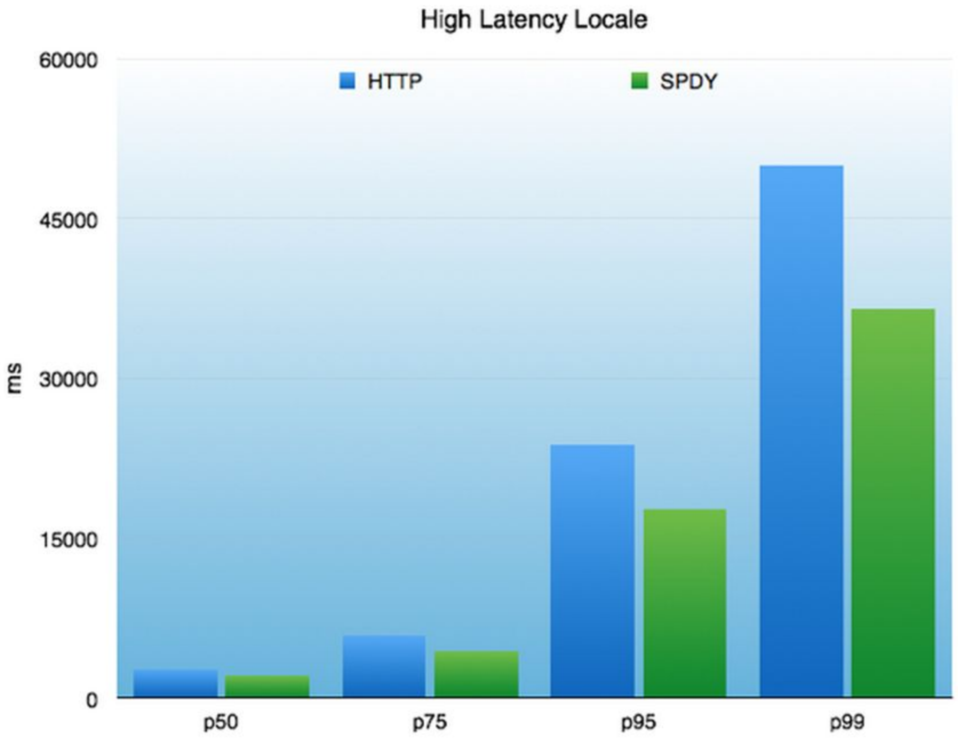
HTTP/2

Results:
much faster!



HTTP/2

Results:
much faster!

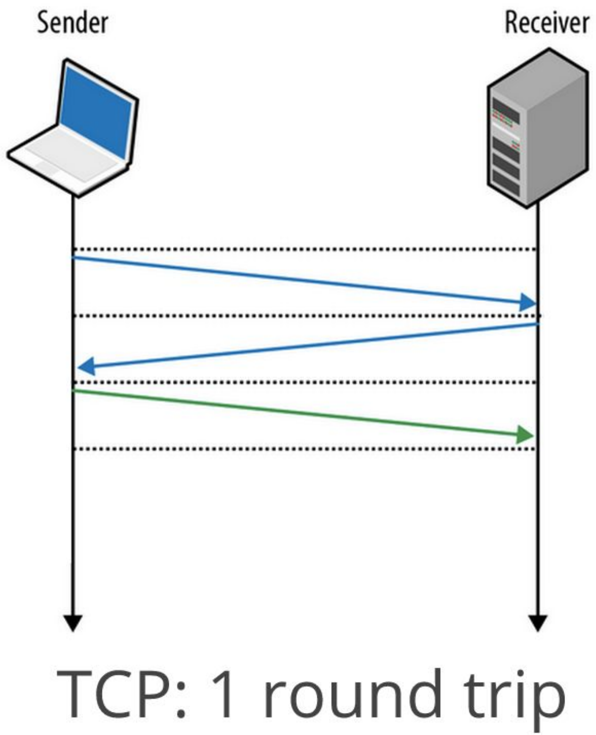


QUIC

HTTP/2 improves HTTP while still using TCP. **QUIC** is an attempt to sidestep TCP in order to sidestep these limitations.

- Heavy emphasis on reducing round trips

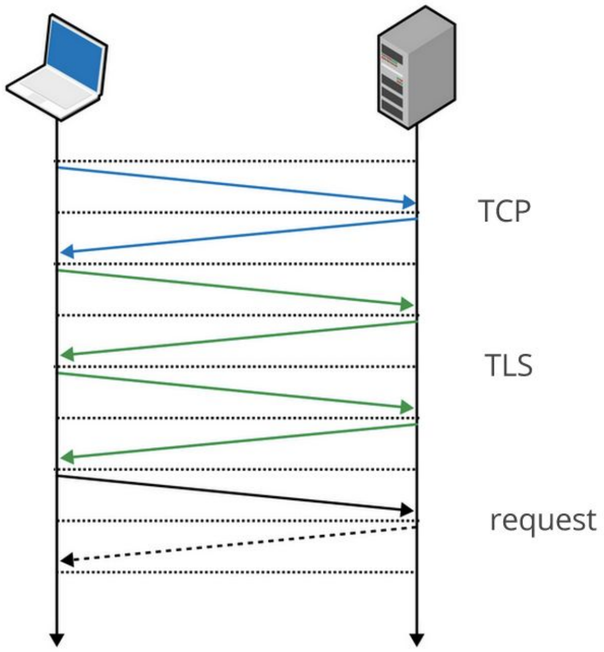
QUIC - Limitations of TCP



Connection Establishment:

- TCP connections require a full round trip before any application data can be sent

QUIC - Limitations of TCP

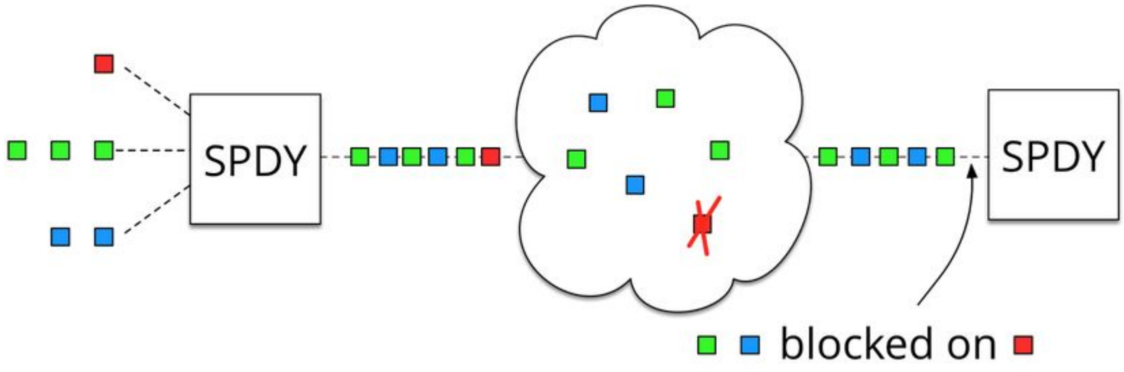


TCP + TLS: 3 round trips

Connection Establishment:

- HTTPS requires three full round trips before any application data can be sent

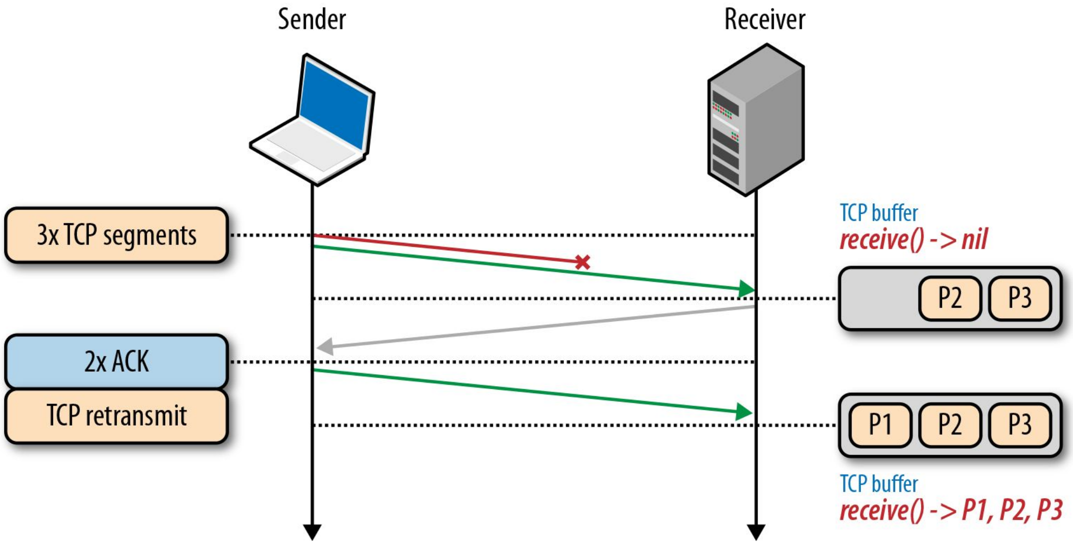
QUIC - Limitations of TCP



Multiplexing:

- We have many independent HTTP/2 requests on a single TCP connection.
- Losing one packet can block unrelated requests

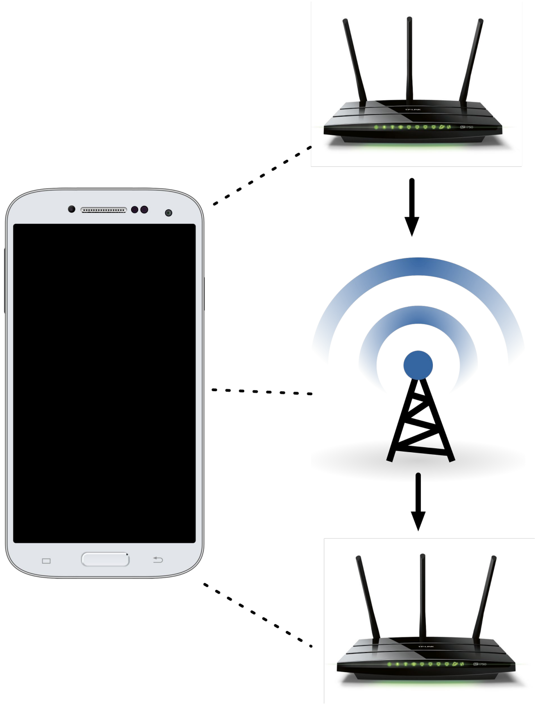
QUIC - Limitations of TCP



Retransmissions:

- Losing packets means retransmissions.
- In high latency networks, this is slow

QUIC - Limitations of TCP



Mobile users:

- A TCP connection is defined by its source address/port combination.
- Mobile networks have higher latency (100-400ms additional RTT)
- If I've got a TCP connection with a good window size open, and I move from wifi to cell, I need to reconnect.

QUIC

Given the limitations of TCP, can we build a better HTTP/2 on top of UDP instead?

- QUIC is an attempt to do this.
- Primary goal is to reduce latency
- You are currently using it if you are using Chrome to talk to Google servers.
    - To inspect, go to chrome://net-internals/

QUIC

Initial connection

- Client sends random 64-bit CID
- Server replies with certificate and cookie
- Client responds with CID, cookie, proposed encrypted session key, algorithm
- Client can immediately send requests.

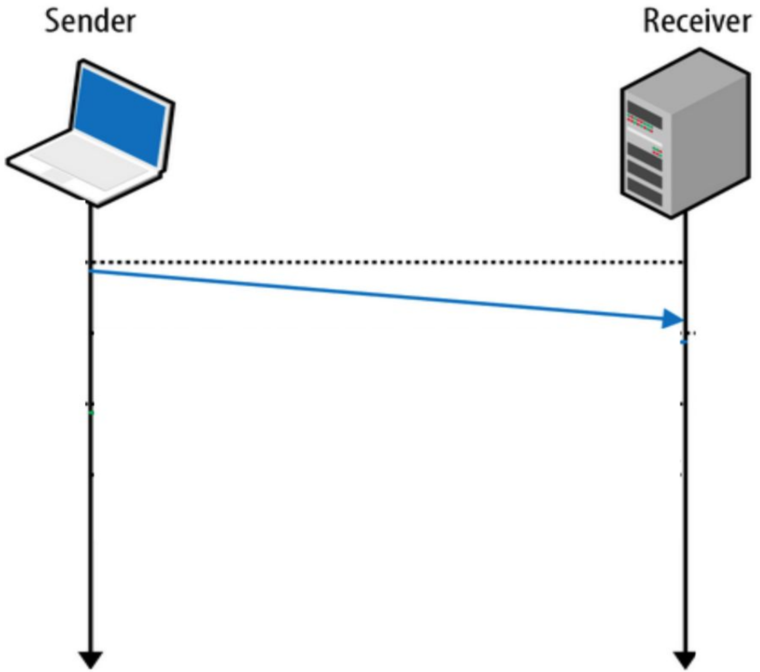
QUIC

Subsequent requests

- Client sends CID, cookie, proposed encrypted session key, algorithm
- Client can immediately send requests.
- Assumes server cert hasn't changed

Engineered to avoid DOS traffic amplification attacks (IP spoofing)

QUIC

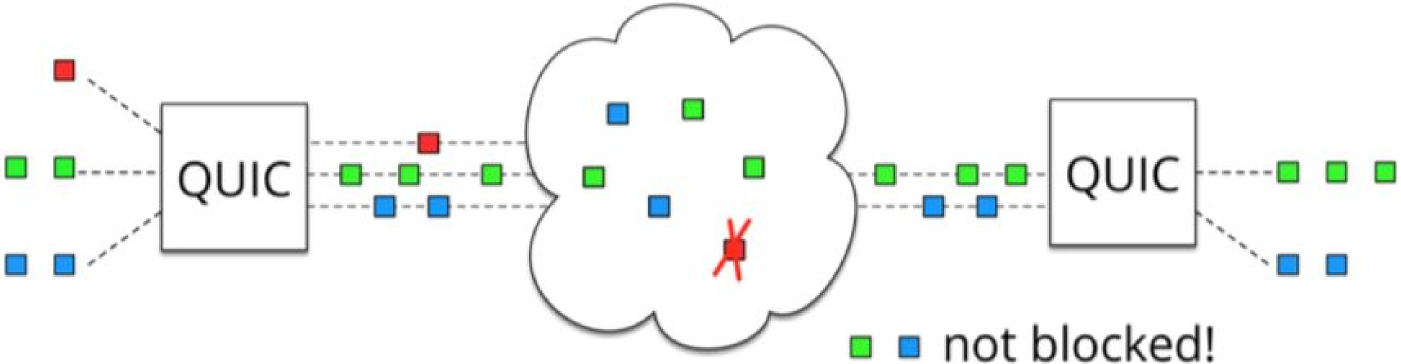


For the common case, initial roundtrips can be completely avoided.

QUIC

Packet loss affects only the resource lost

- No head-of-line blocking



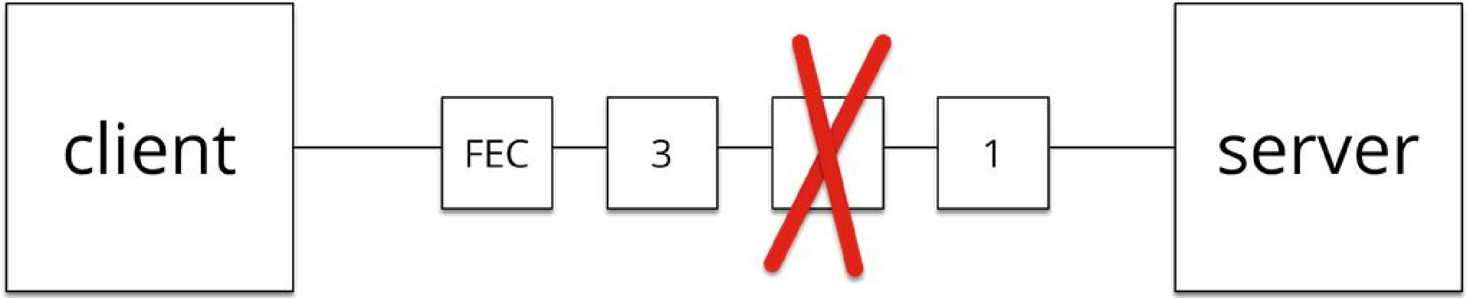
QUIC

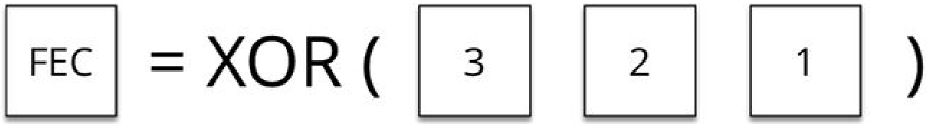
Traditional TCP handles packet loss through retransmission.

Quic can handle packet loss without retransmission. How?

QUIC

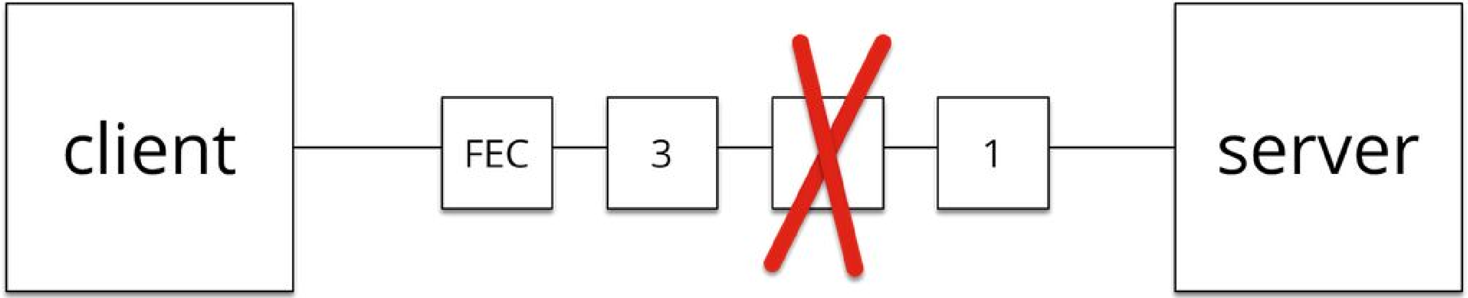
Forward Error Correction

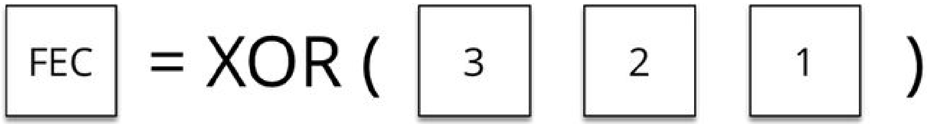




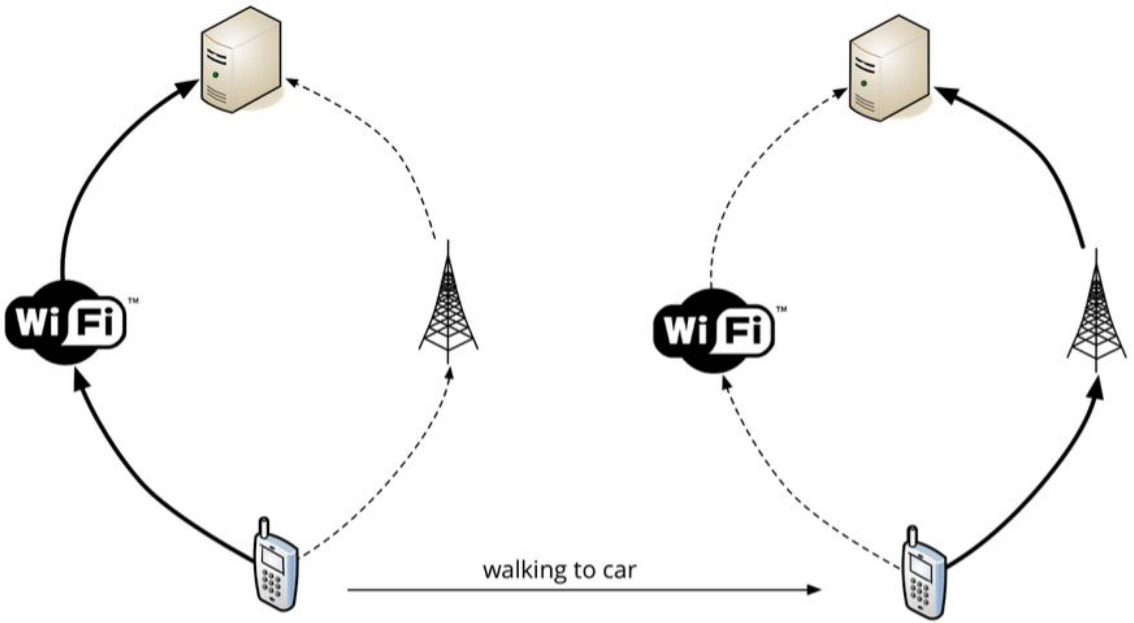
QUIC

Note: we are trading bandwidth for latency.





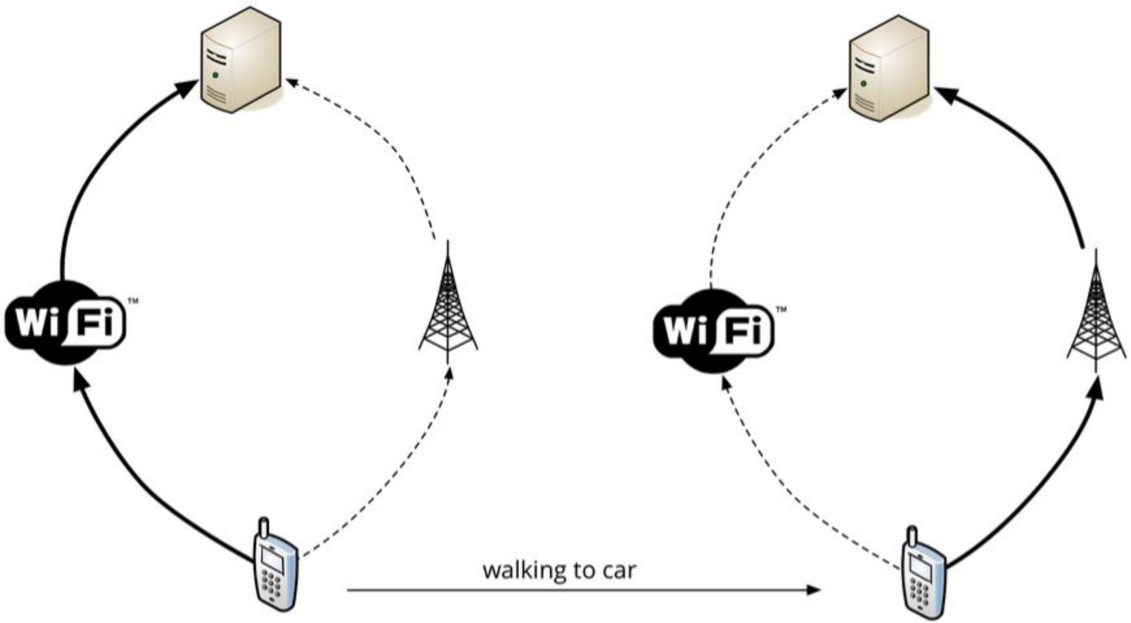
QUIC



Parking lot problem

- Web doesn’t handle changing IPs very well
- Mobile users are increasingly common

QUIC



Parking lot problem

- QUIC doesn't expect IP to stay static
- Connection defined by CID

QUIC

Results:

- 75% of requests avoid handshake
- Google search saw 3% reduction in mean page load time.
- For slowest 1% of users, can reduce page load time by a full second.
- YouTube users report 30% fewer rebuffers.
- Performs best under poor network conditions

Course Conclusion

Let’s say...

...I want to find a home to live in.

...I am lost in a foreign city.

...I want to go on a date.

... what do I do?

Course Conclusion

Every day, billions of people use the same suite of technologies to solve these problems: internet services.

As these services get increasingly popular, they need to continue to function.

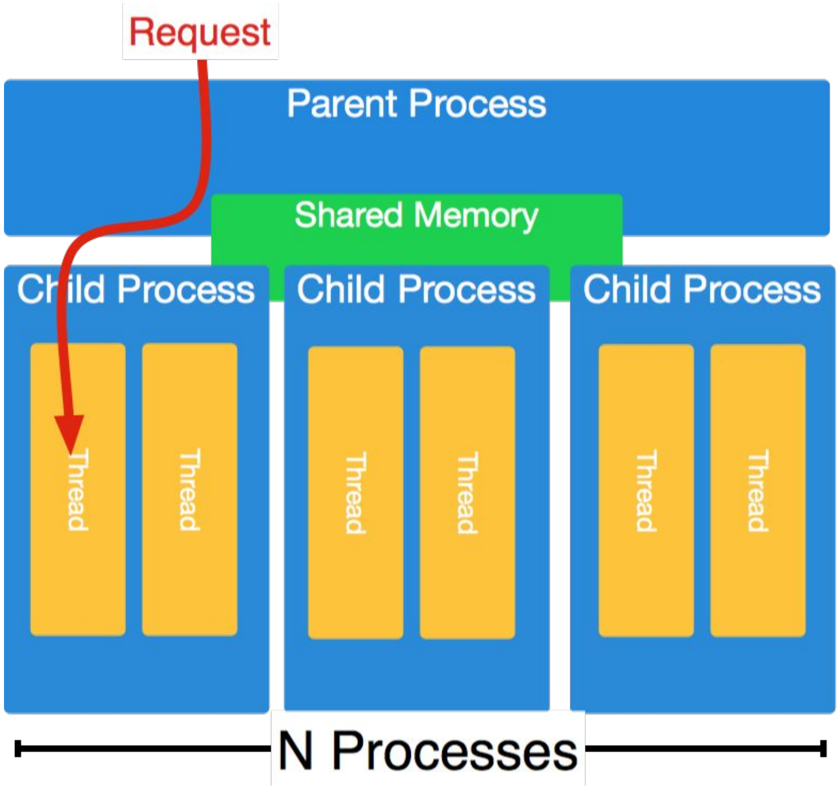
Scaling even relatively simple web applications (Twitter) can be very complex.

Course Conclusion

You've got a web application that is becoming increasingly popular and performance is degrading.

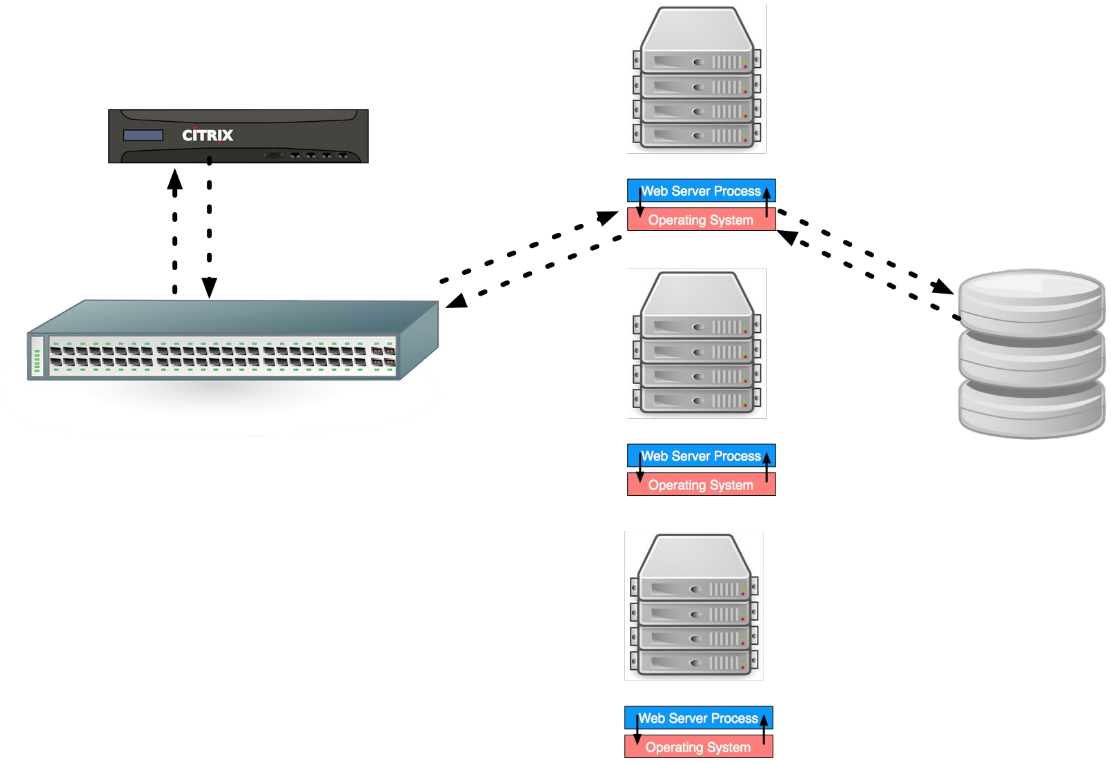
What do you do?

We've covered a lot of ground



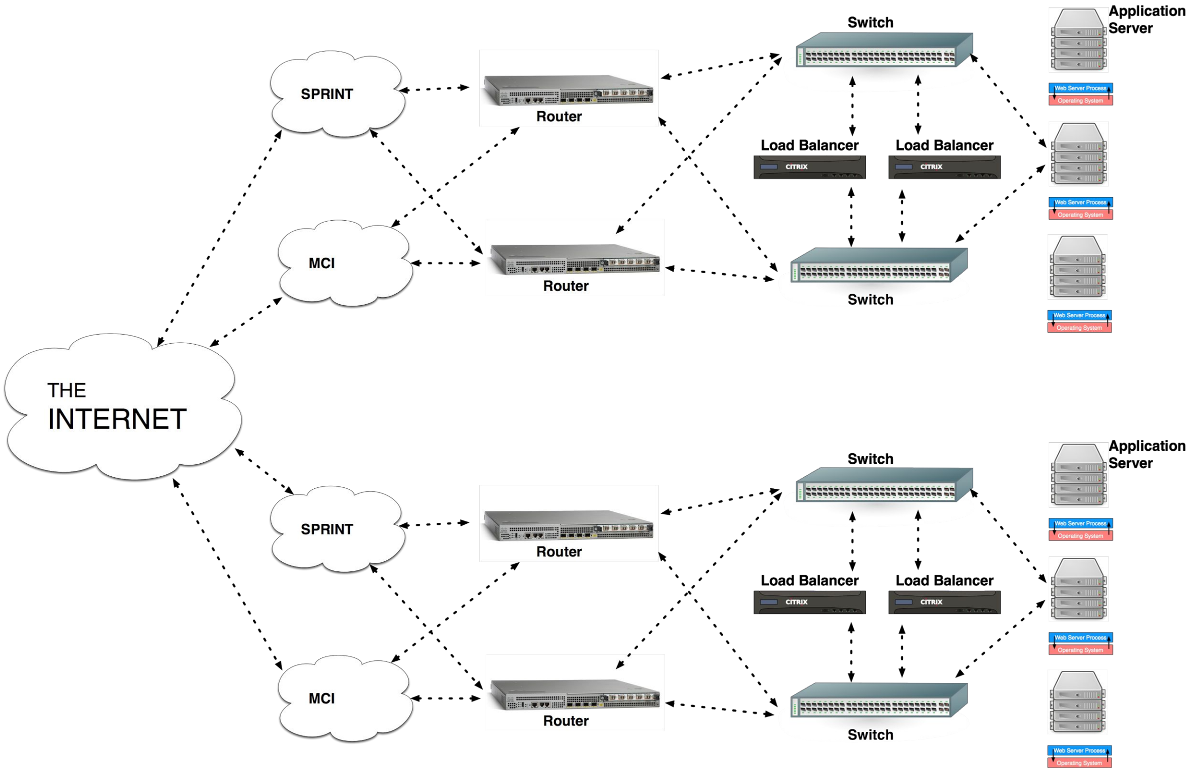
HTTP Servers, Application servers and their design

We’ve covered a lot of ground



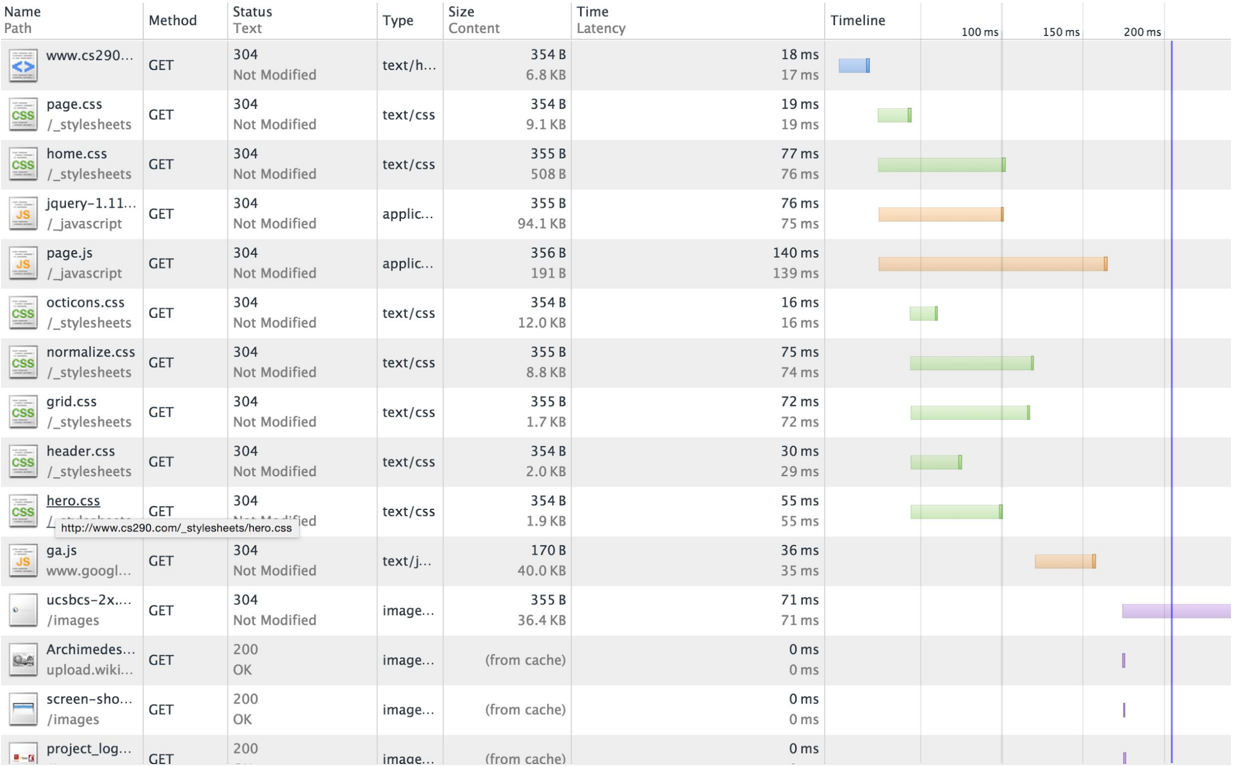
The use of load balancing in achieving horizontal scaling

We’ve covered a lot of ground



The architecture of a high availability, share nothing web application

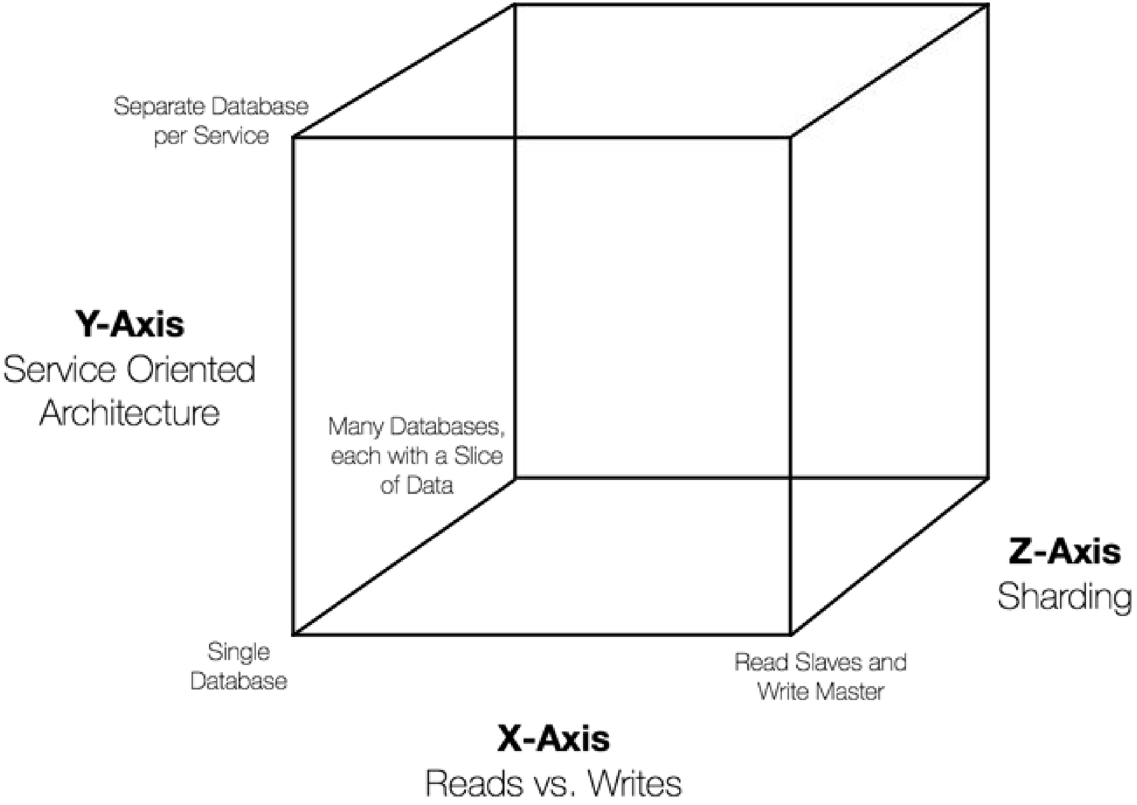
We’ve covered a lot of ground



Name Path	Method	Status Text	Type	Size Content	Time Latency	Timeline 100 ms 150 ms 200 ms
www.cs290…	GET	304 Not Modified	text/h…	354 B 6.8 KB	18 ms 17 ms	
page.css /_stylesheets	GET	304 Not Modified	text/css	354 B 9.1 KB	19 ms 19 ms	
home.css /_stylesheets	GET	304 Not Modified	text/css	355 B 508 B	77 ms 76 ms	
jquery-1.11… /_javascript	GET	304 Not Modified	applic…	355 B 94.1 KB	76 ms 75 ms	
page.js /_javascript	GET	304 Not Modified	applic…	356 B 191 B	140 ms 139 ms	
octicons.css /_stylesheets	GET	304 Not Modified	text/css	354 B 12.0 KB	16 ms 16 ms	
normalize.css /_stylesheets	GET	304 Not Modified	text/css	355 B 8.8 KB	75 ms 74 ms	
grid.css /_stylesheets	GET	304 Not Modified	text/css	355 B 1.7 KB	72 ms 72 ms	
header.css /_stylesheets	GET	304 Not Modified	text/css	354 B 2.0 KB	30 ms 29 ms	
hero.css http://www.cs290.com/_stylesheets/hero.css	GET	304 …ied	text/css	354 B 1.9 KB	55 ms 55 ms	
ga.js www.googl…	GET	304 Not Modified	text/j…	170 B 40.0 KB	36 ms 35 ms	
ucsbcs-2x.… /images	GET	304 Not Modified	image…	355 B 36.4 KB	71 ms 71 ms	
Archimedes… upload.wiki…	GET	200 OK	image…	(from cache)	0 ms 0 ms	
screen-sho… /images	GET	200 OK	image…	(from cache)	0 ms 0 ms	
project_log…	GET	200	image…	(from cache)	0 ms	

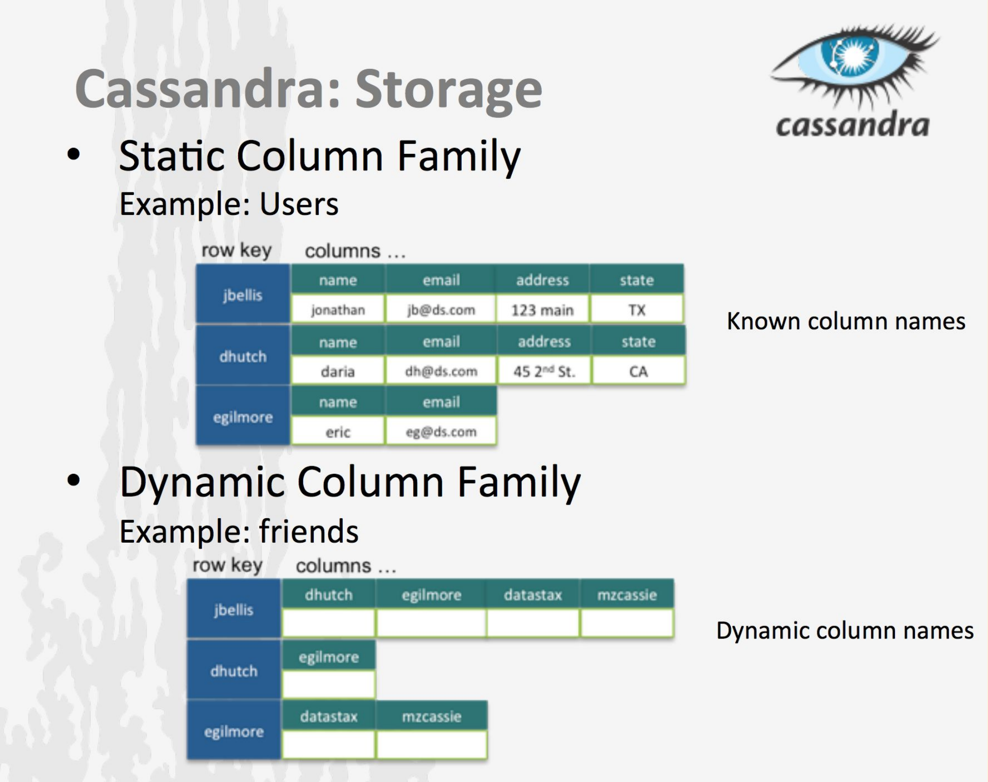
All about caching, both on the client and the server

We've covered a lot of ground



Relational databases and how to scale them

We’ve covered a lot of ground



Scaling beyond relational databases: NoSQL stores

We’ve covered a lot of ground

The use and design of Content Distribution Networks

We’ve covered a lot of ground

In addition to the question of scaling we have looked at:

- Basics of web security
- Client side MVC
- Asm.js and Emscripten
- The future: HTTP 2.0 & QUIC

We’ve covered a lot of ground

We’ve gained a lot of experience with modern web application technology

What I hope...

If you are headed for academia, I hope this window into industry helps your future research

If you are headed for industry, I hope the skills you've gained can help you get a job

If you ever want to start a company, I hope this course has given you all the tools you need to build a scalable internet service.

What I hope...

And I hope you've had fun!

- Please remember to fill out EIP on my.ucla.edu
- See you all on Thursday for presentations (here, usual time).
- Please send writeups by 4pm Thursday to andrew.mutz@cs.ucla.edu