CS188
Scalable Internet Services

John Rothfels, 12/8/20

Announcements

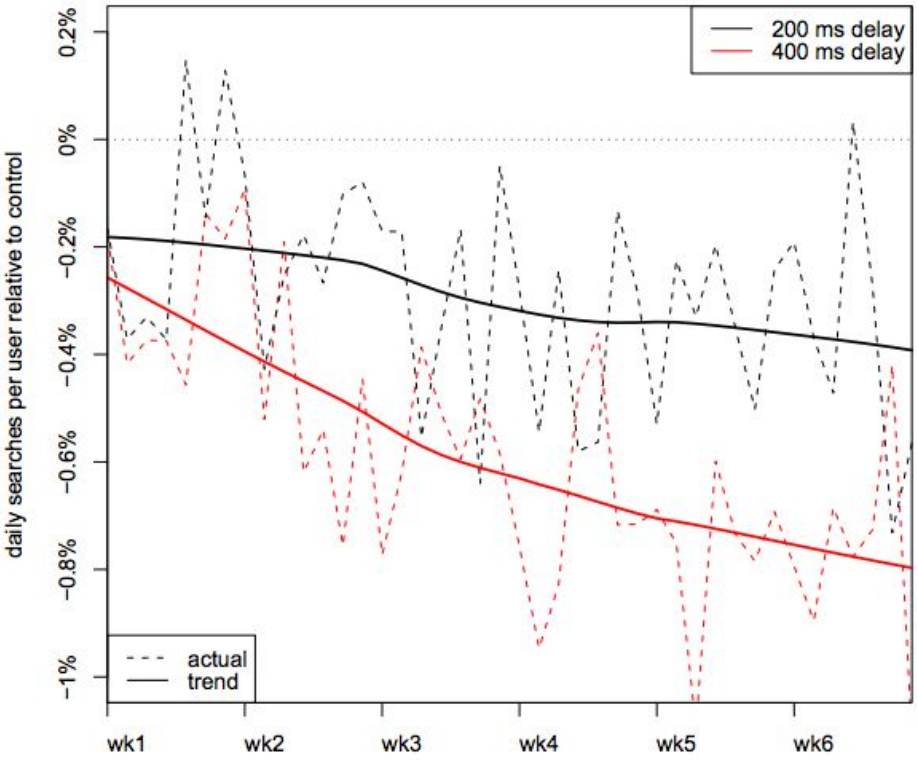
- Final paper due **tomorrow at 11:59pm**
- Limited edition CS188 plant baby contest entries due **tomorrow at 11:59pm**
  - How many plants + musical instruments are in my house?
  - Email your answer to rothfels@cs.ucla.edu
- Final presentations Thursday and Friday
  - Schedule here

Motivation

Speed Matters: Google

- Slowing searches down has a measurable effect on user behavior
  - Slowing 200-400ms decreased the rate of future searches per user by 0.2-0.7%
- Users remember slow performance
  - 200ms delay => 0.22% to 0.36% fewer
  - 400ms delay => 0.44% to 0.74% fewer

Motivation



Users remember poor performance!

- 400ms delay resulted in 0.21% fewer searches for five weeks following the experiment with delays removed

Motivation

Speed matters: Walmart

- Walmart chose a real user monitoring (RUM) approach to measure page load times on walmart.com
  - Used Boomerang.js to measure time between page head and window.onload (all content loaded)
  - Found low page load times were positively correlated with conversion rate

Motivation

Impact of site performance on overall site conversion rate….

Baseline – 1 in 2 site visits had response time > 4 seconds

* Sharp decline in conversion rate as average site load time increases from 1 to 4 seconds

* Overall average site load time is lower for the converted population (3.22 Seconds) than the non-converted population (6.03 Seconds)

Note: Load Time here is the time taken from head of the page to page ready (T_Page)

Motivation

Speed matters: Walmart

- Team set performance goals for particular pages based on measurements
- Improving page load times by 1 second resulted in up to a 2% increase in conversion rates
- Improving page load times by 100ms resulted in as much as 1% revenue increase!

Motivation

⁉️ **What can we do if page load time is poor?**

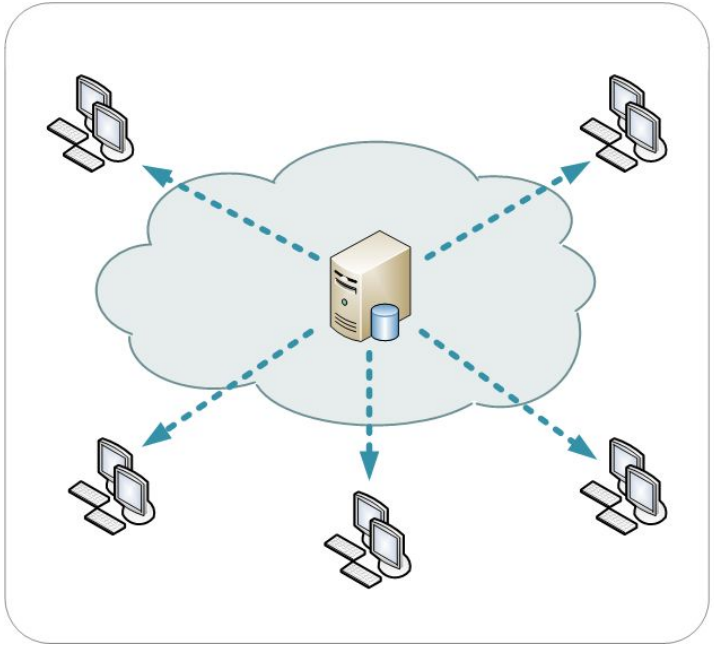
Motivation

⁉️ **What can we do if page load time is poor?**

- Make sure our server isn’t doing anything silly 😅
- Reduce network latency
- Reduce the size of our HTML / JavaScript / CSS / images / …
  - Remove dependencies from package.json
  - https://bundlephobia.com/
- … many other hacks …

Motivation

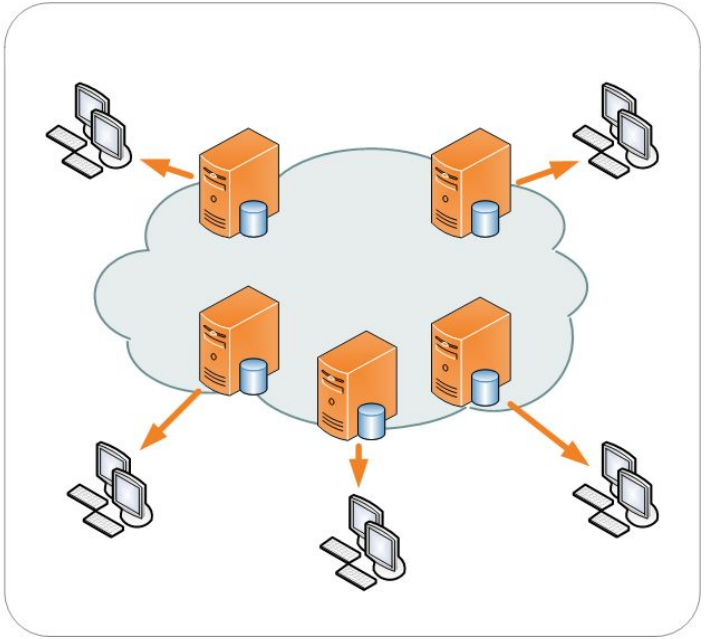
⁉️ **If our users are geographically distributed, how can we reduce latency?**



Motivation

⁉ **If our users are geographically distributed, how can we reduce latency?**

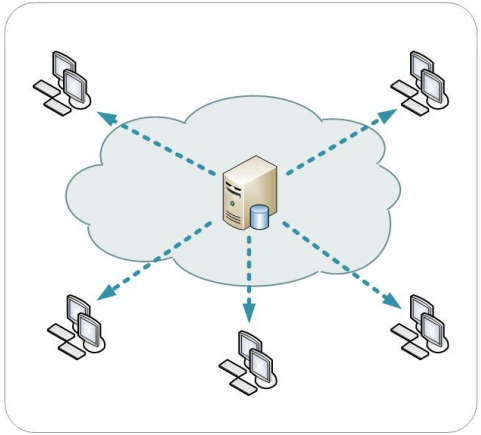
- Bring the content closer to the users

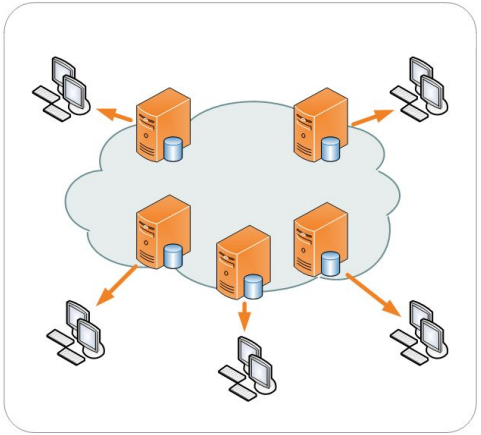


Motivation

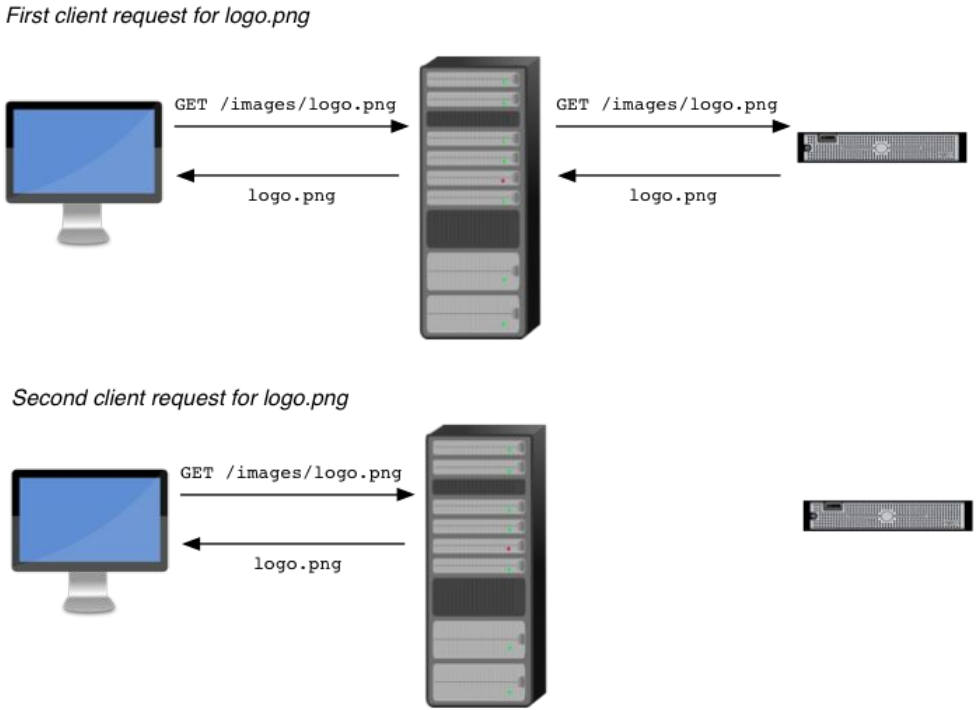
⁉️ **If our users are geographically distributed, how can we reduce latency?**

- So instead of all users contacting the origin app server for content, they will contact servers that are closer to them





CDNs



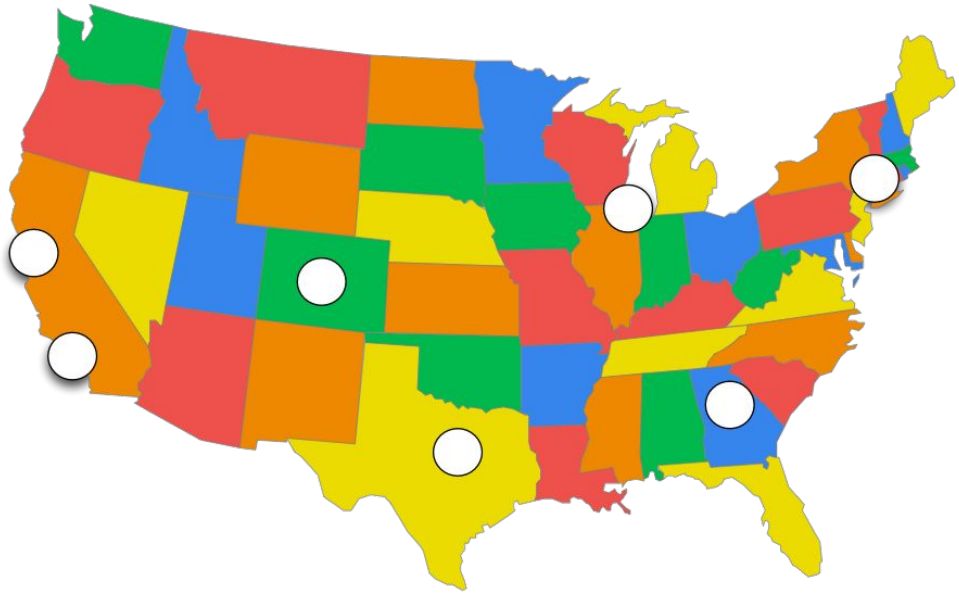
A content delivery network (CDN) is a caching abstraction that acts like an HTTP proxy.

HTTP headers guide caching decisions.

CDNs

- Caches rely on a subset of popular content to maintain hit rate
  - When misses occur, cache interaction is pure overhead
- If content becomes popular very quickly, caches can introduce delay
  - Not in cache implies origin fetch, which ties up resources
  - Many requests for a missing object have to be coalesced to avoid transferring unnecessary load onto the origin
- HTTP Headers
  - `Cache-Control`, `If-Modified-Since`, `etags`

CDNs

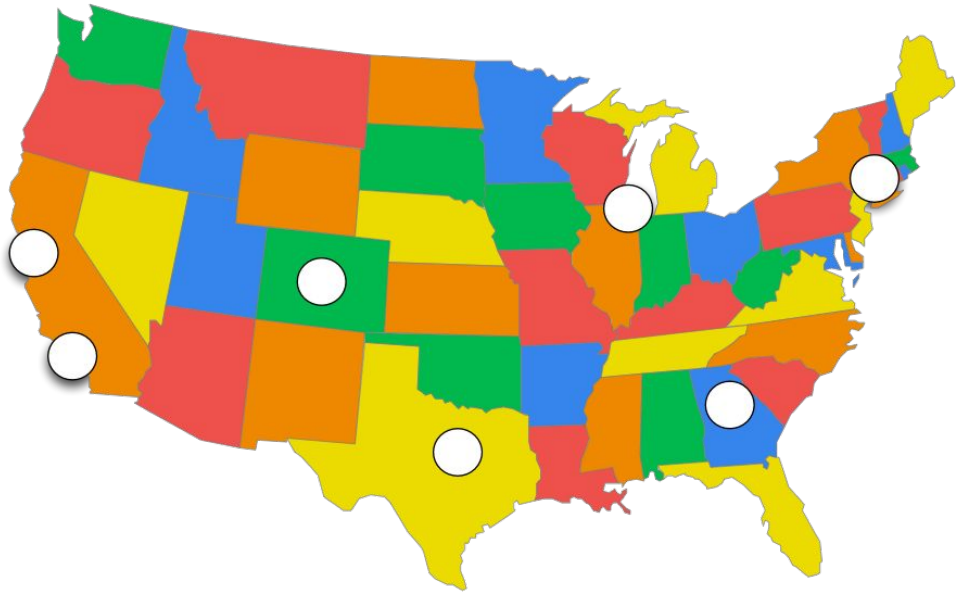


In addition to being a caching abstraction, a CDN is designed to reduce latency by providing many nodes for geographically distributed requests to route to.

Two big questions:

- Which endpoint should we send them to?
- How do we send them there?

CDNs



⁉ How do we send them there?

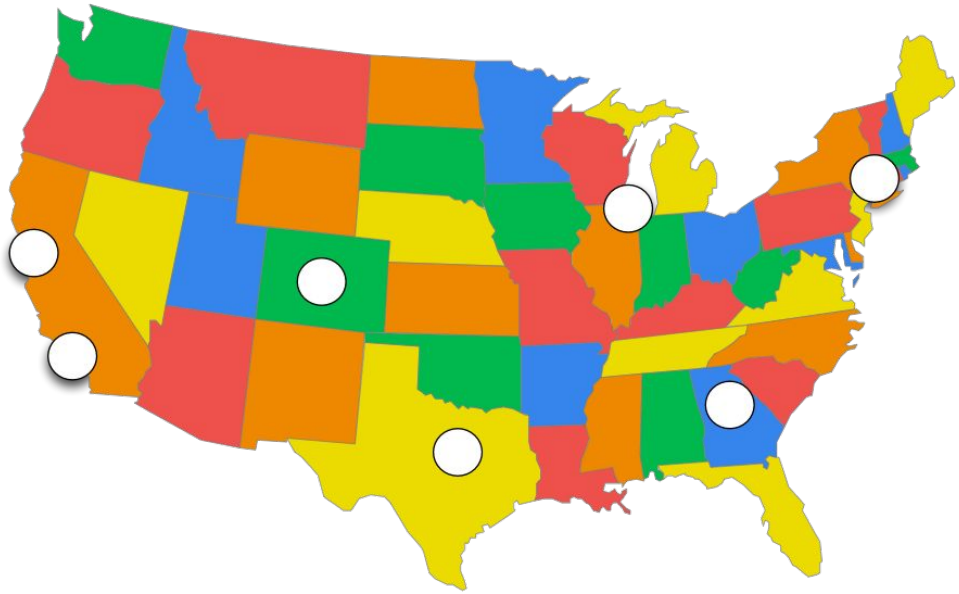
Modify hostname of URLs that you want to cache in order to route to CDN provider:

http://www.example.com/images/logo.png

=>

http://www.example.com.akamai.net/images/logo.png

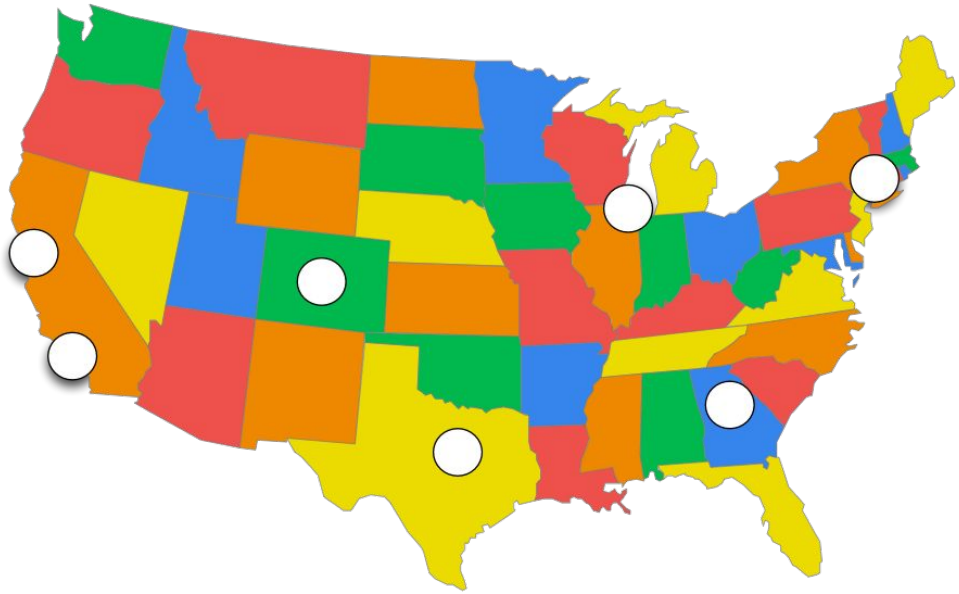
CDNs



⁉ How do we send them there?

CDN provider configures DNS to route these requests to its “Points of Presence (POP)”

CDNs

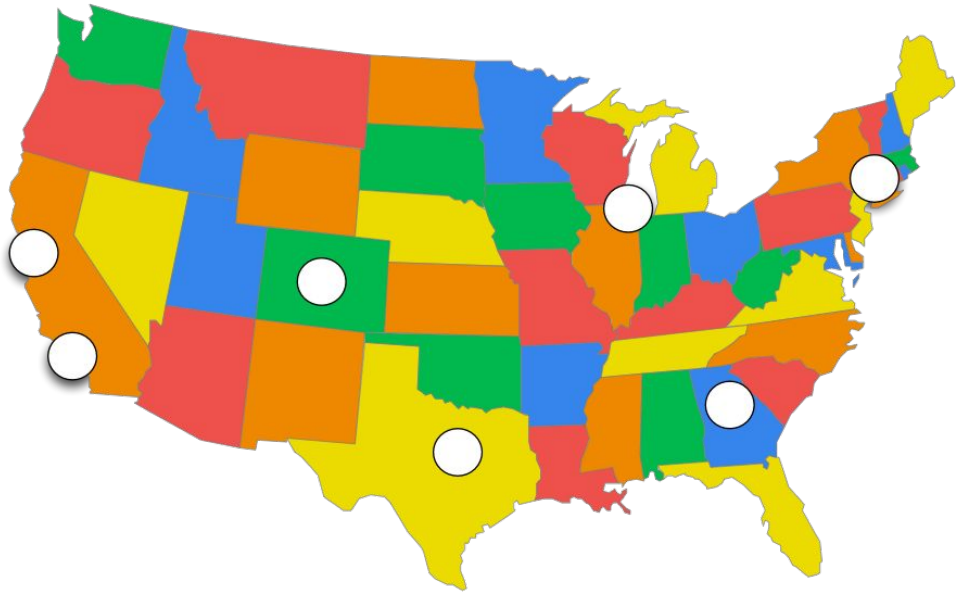


⁉ **How do we send them there?**

www.example.com.akamai.net

DNS provider can choose how to resolve this. Can Resolve to an IP address in Los Angeles for a west coast client, and New York for an east coast client, etc.

CDNs



⁉ How do we choose which POP to send to?

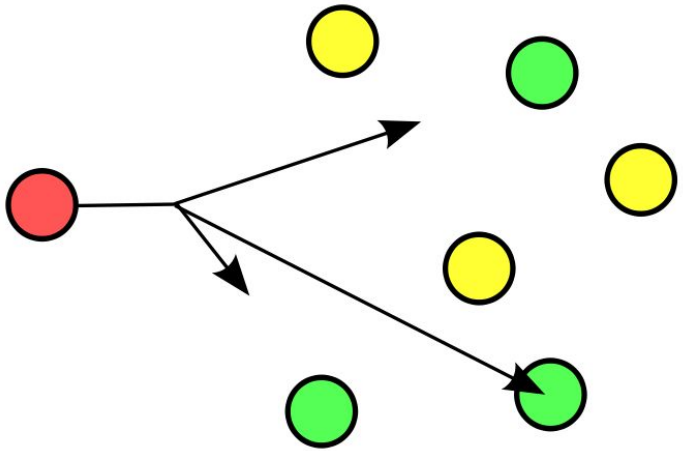
- Anycast
- Geolocation
- Client performance

CDNs: anycast

anycast IP allows multiple hosts on the internet to have the same IP address

BGP rules are set to route to nearest host

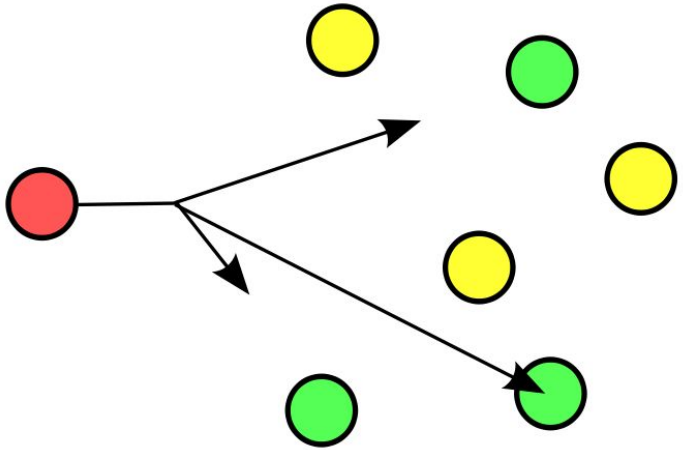
We can have all PoPs have the same IP, and use BGP and anycast to route to nearest PoP



CDNs: anycast

Challenges:

- A sequence of IP packets may end up at different hosts (BGP route changes)
- This will route to closest PoP, but closest might not be best
    - Packet loss, bandwidth could be an issue



CDNs: geolocation

Geolocate the DNS resolver

Send to PoP that is geographically closest to the DNS resolver

Challenges:

Clients don’t always use nearby DNS servers

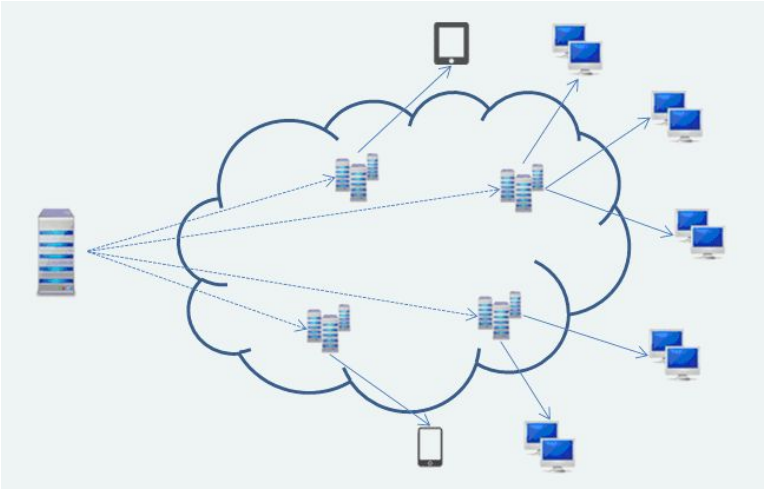
CDNs: client performance measurements

Find a high-traffic web property that you can serve images and JS from

When clients arrive, request an image from a random PoP and record performance

Maintain a mapping of DNS resolver to average client performance

Use this average client performance to make PoP decision at request time



CDNs

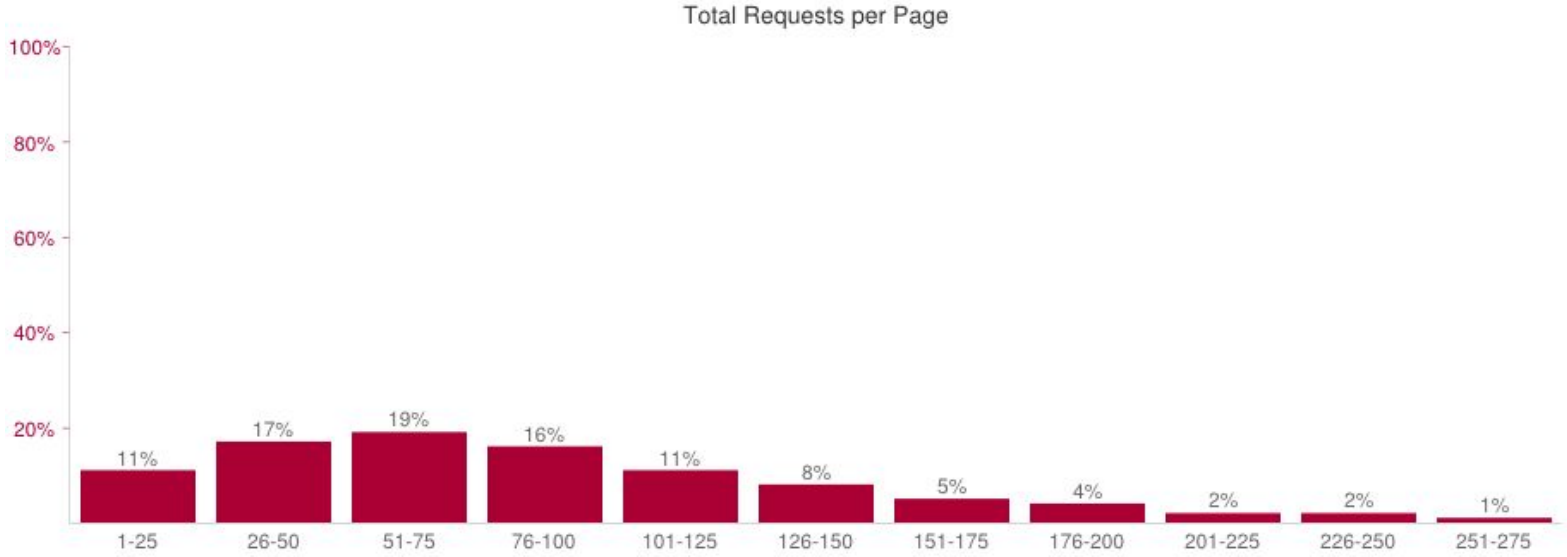
In summary, a CDN is a relatively simple way to speed up the serving of static assets in your web application.

Common providers:

- Akamai
- Amazon (CloudFront)
- CloudFlare

Today’s performance problems

Web pages have many constituent resources



Today’s performance problems

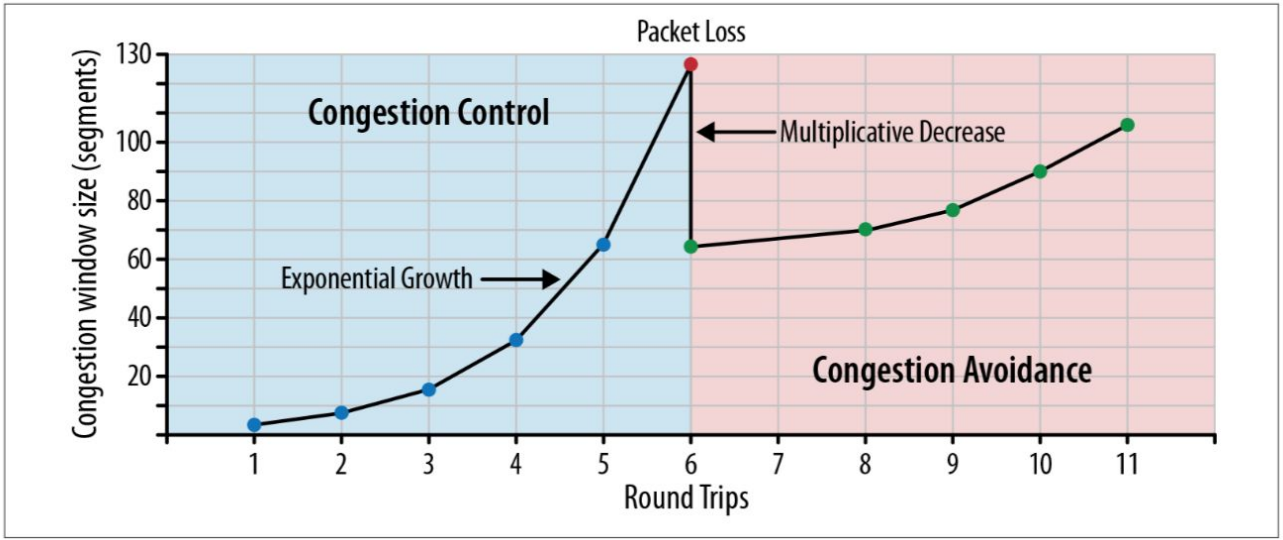
Many requests are needed to present today’s web pages.

- CSS
- Javascript
- Images

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

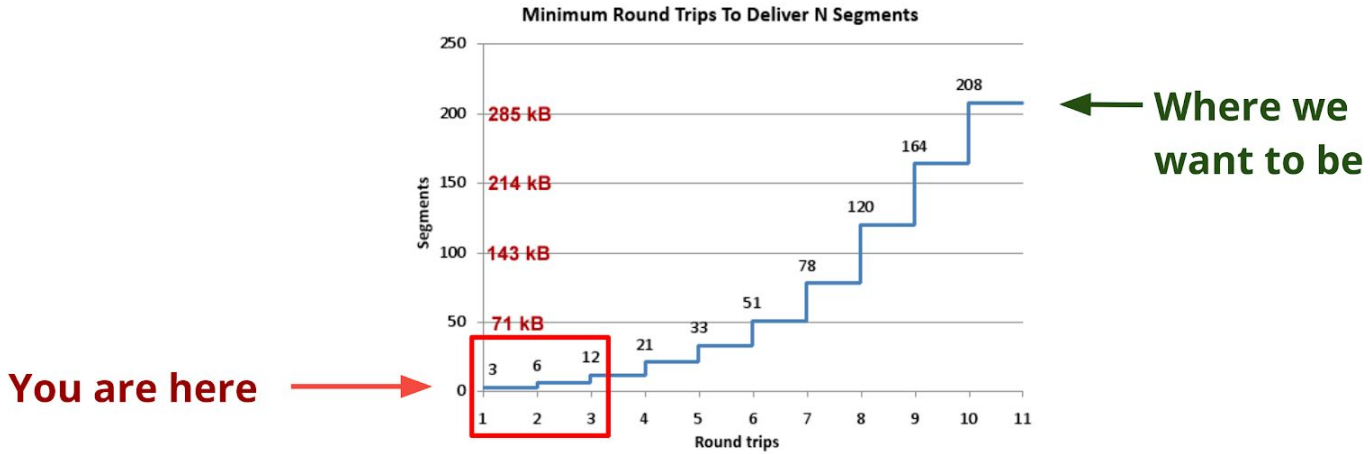
Today's performance problems

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



Today’s performance problems

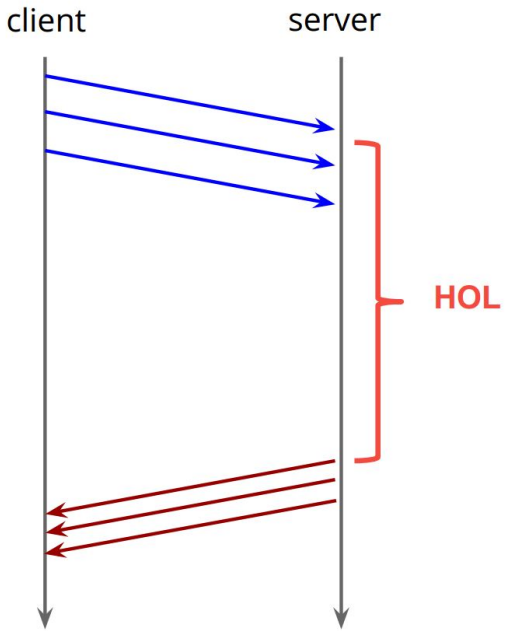
TCP was designed for long-lived flows. HTTP is short and bursty.



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 /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=111461200.1476767743.1349274889.1349274889.134
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.

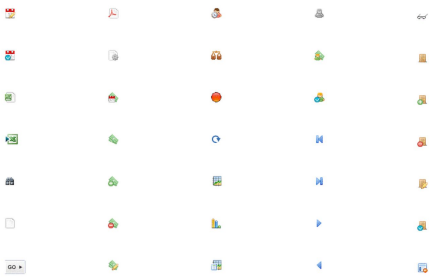
```
GET /assets/dist/js/jquery.appear.20121001205006.js HTTP/1.1
Host: www.etsy.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/536.26.14 (
Accept: */*
DNT: 1
Referer: http://www.etsy.com/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: autosuggest_split=1; etala=111461200.1476767743.1349274889.1349274889.134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

14 new bytes; 683 total

Today’s performance tricks

Many techniques are used to reduce the number of requests:

- GraphQL
- Caching
- Server-side rendering
- File concatenation
  - Having many JS, CSS files means having many requests, so we mash them all together
  - Image Spriting: the process of putting lots of smaller images in a single image, and then referring to them all using offsets.



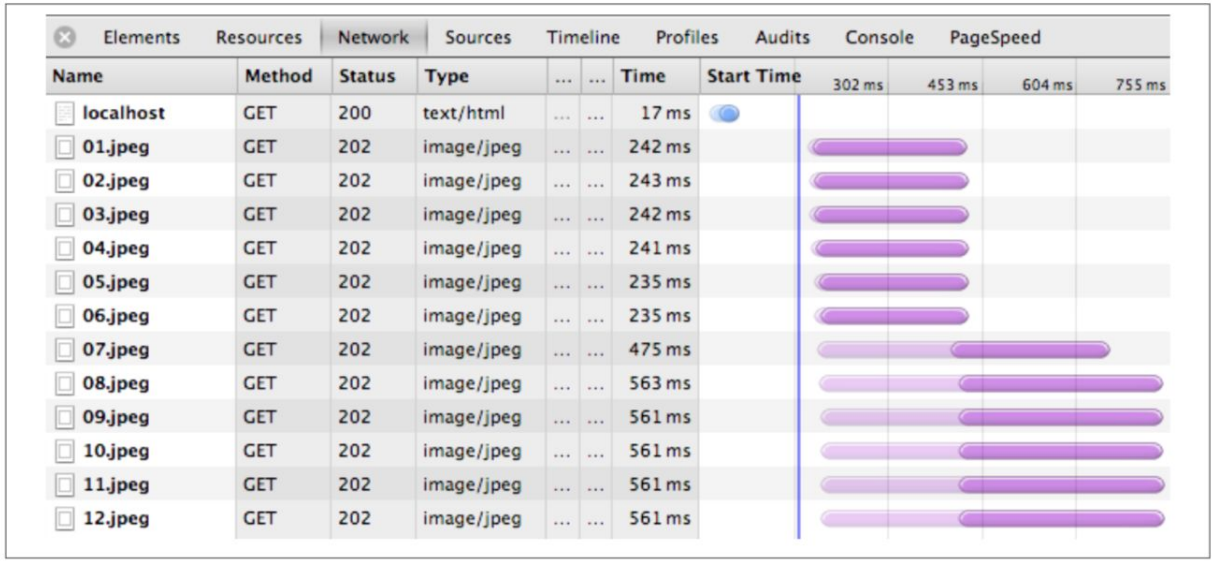
Today’s performance tricks

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:



Name	Method	Status	Type	...	...	Time	Start Time
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 in all major browsers
- Server side support optional in Apache and Nginx

Standard completed in 2015.

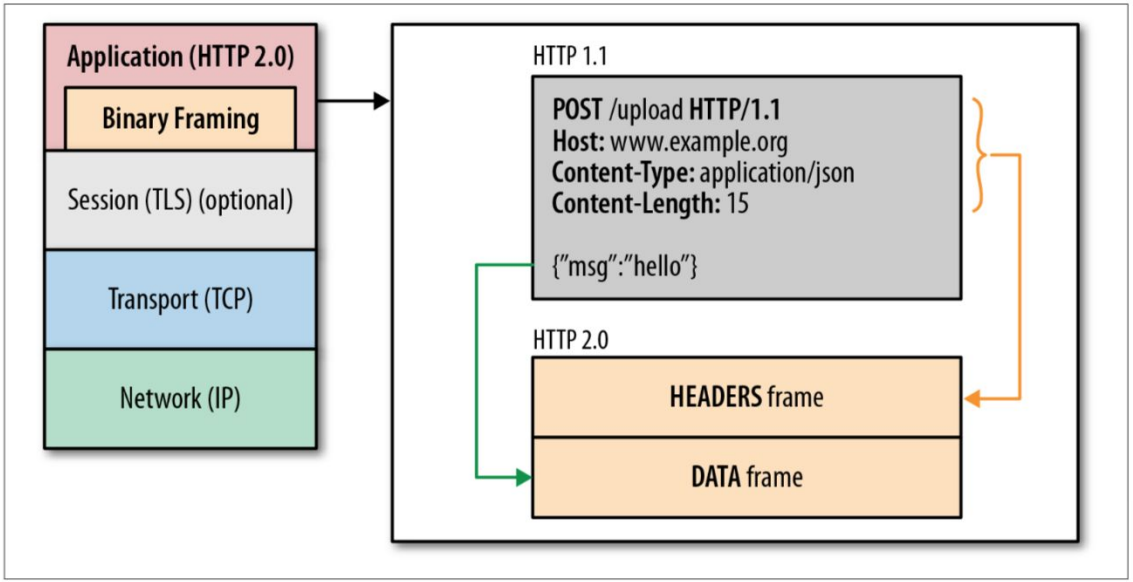
HTTP/2

How does HTTP/2 work?

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

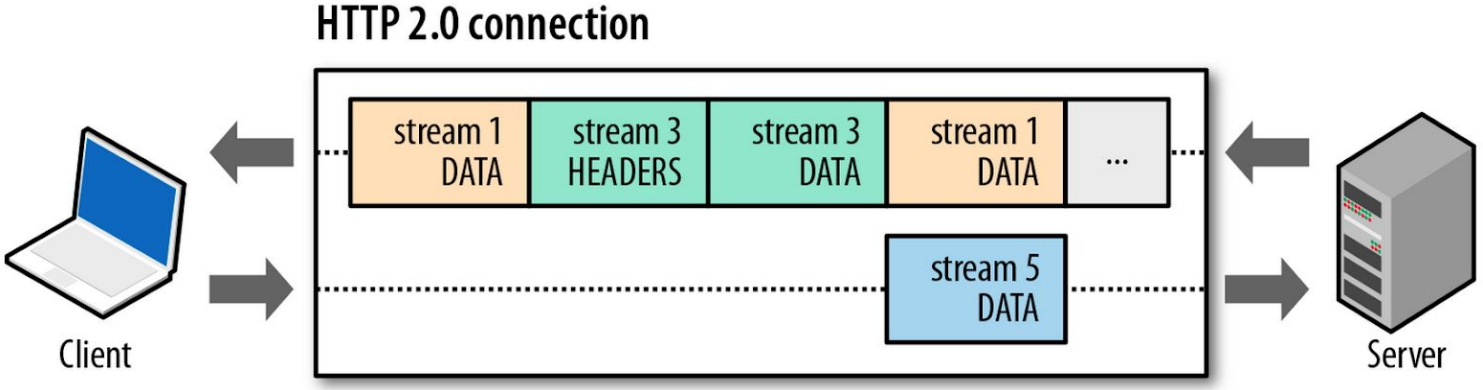
HTTP/2

Single TCP stream: Binary Framed connection



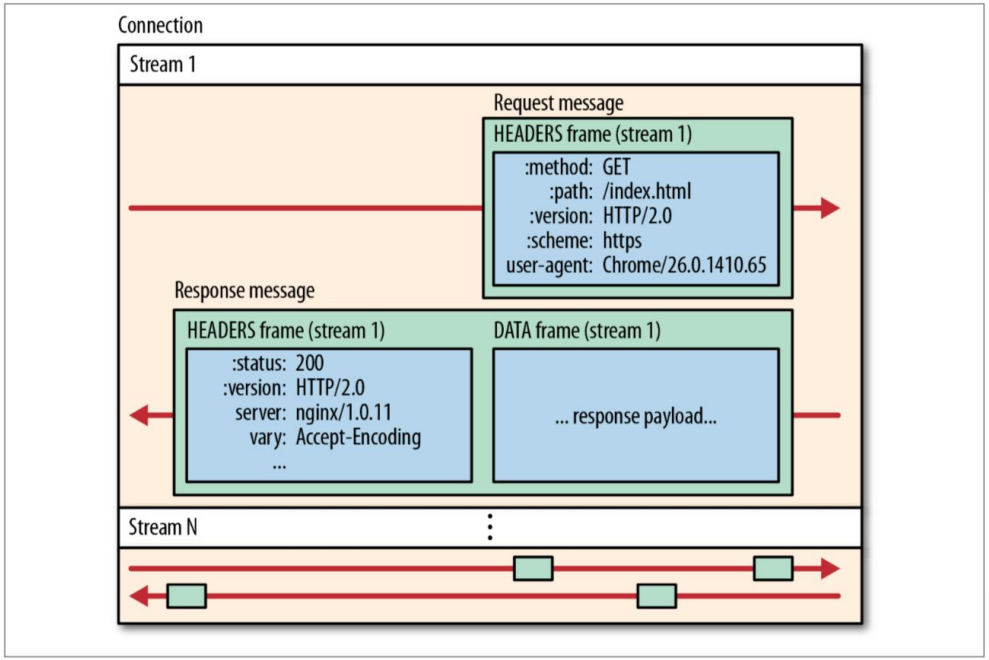
HTTP/2

Single TCP stream: Binary Framed connection



HTTP/2

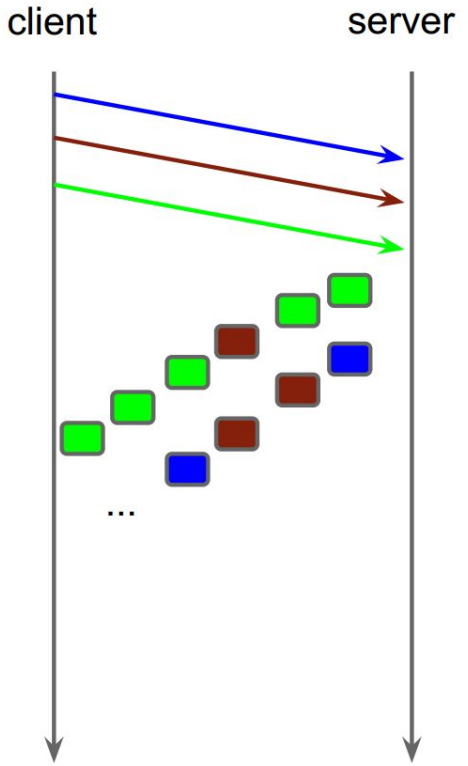
Single TCP stream: Binary Framed connection



HTTP/2

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

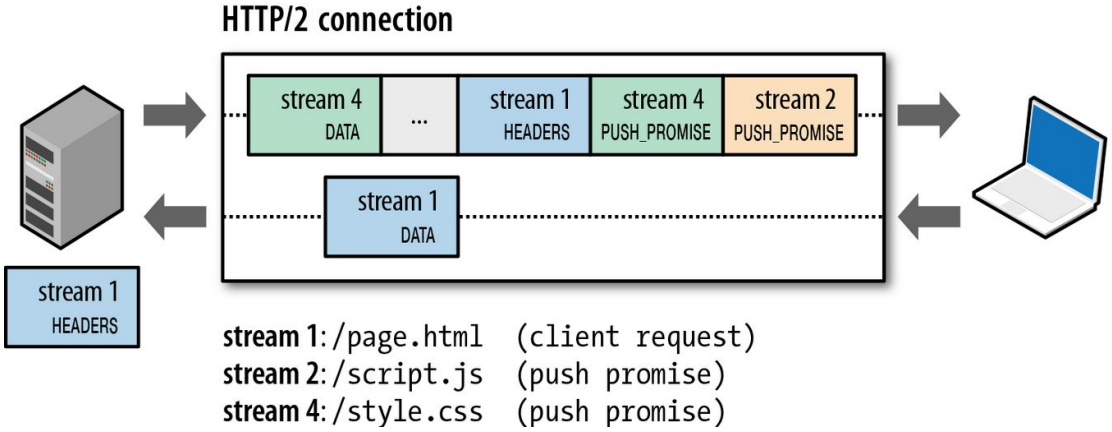
- Since order is no longer mandated, we can implement prioritization
  - DOM highest priority, followed by CSS, Javascript
  - Images lowest

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

HTTP/2

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

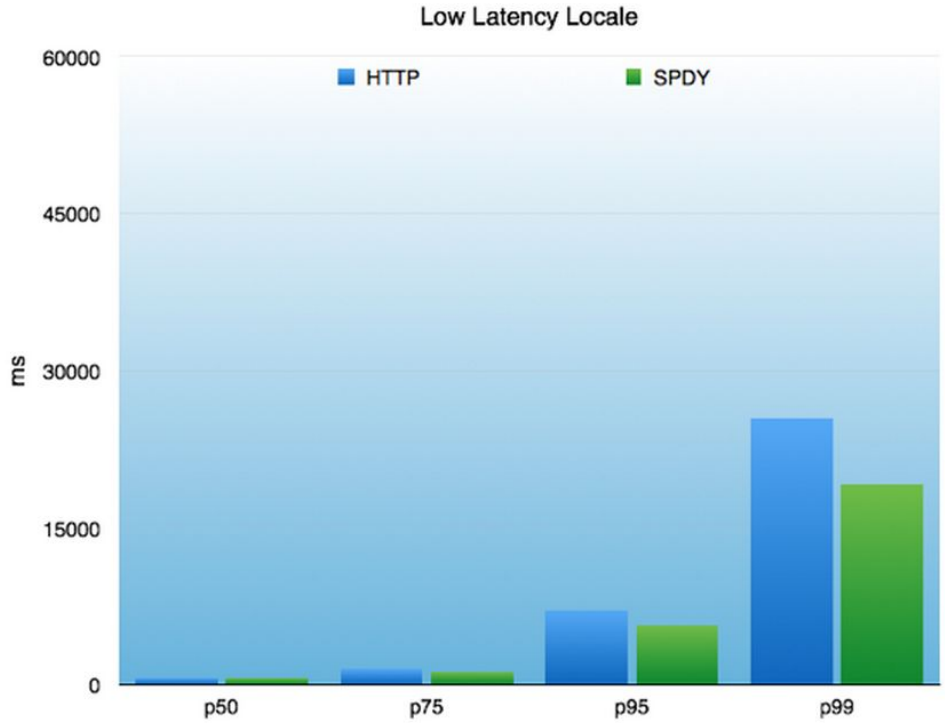
Results: much faster!

	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

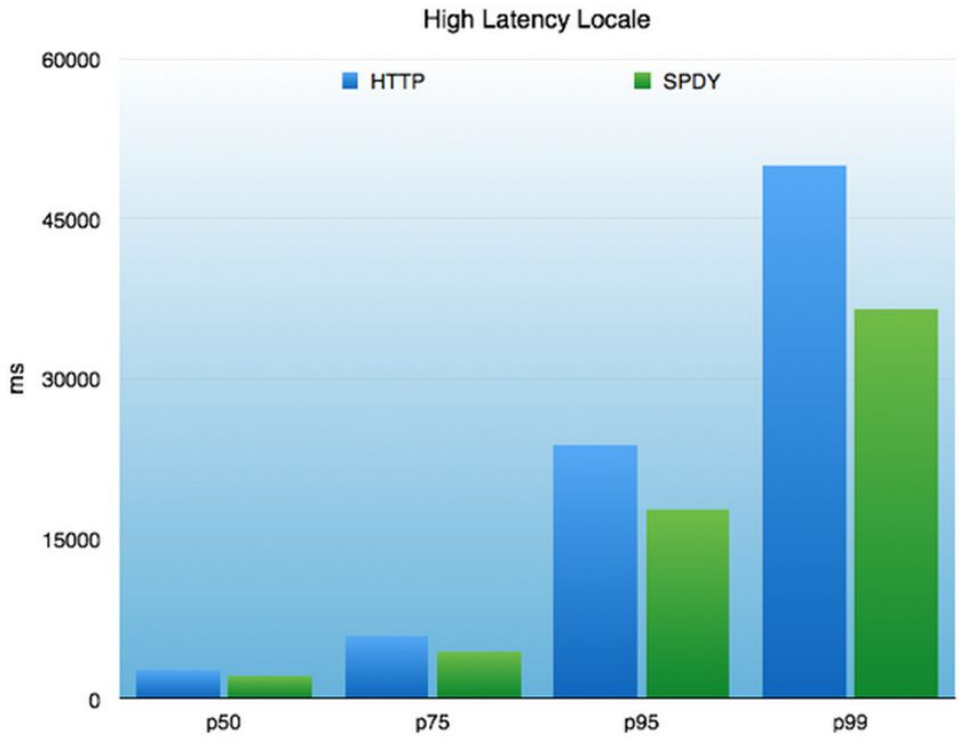
HTTP/2

Results: much faster!



HTTP/2

Results: much faster!



HTTP/2

Demo time:

https://http2.akamai.com/demo

http://www.http2demo.io/

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…

… the world is shut down due to COVID-19…

… 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.

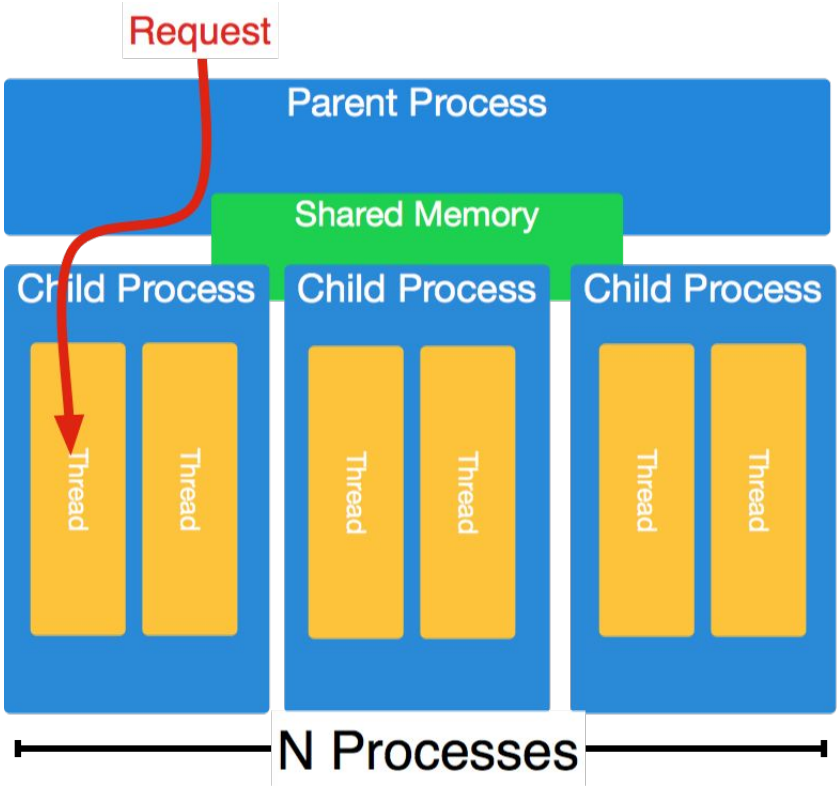
Scaling even relatively simple web applications (🍭) can be very complex!

Course conclusion

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

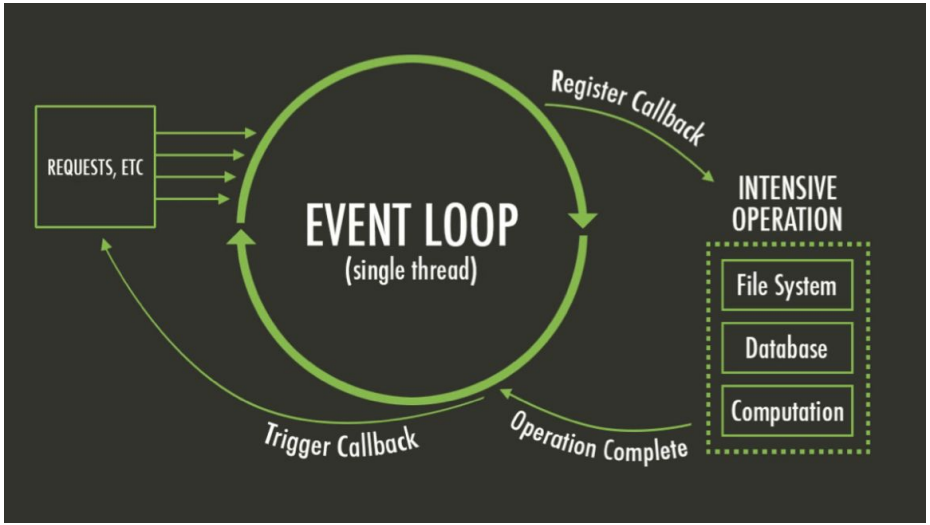
What do you do?

We've covered a lot of ground



HTTP servers, application servers and their design

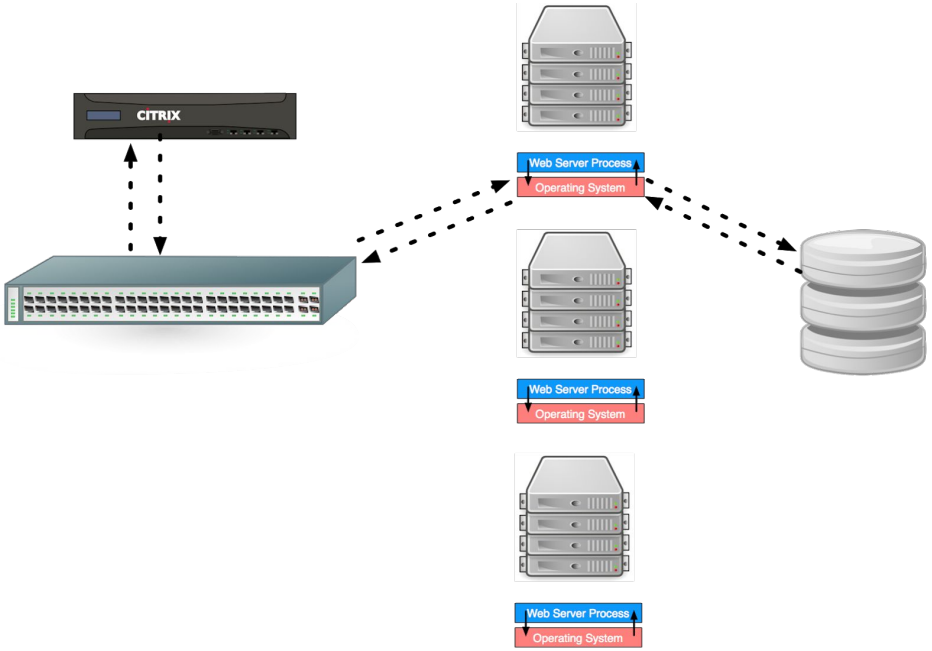
We’ve covered a lot of ground



HTTP servers, application servers and their design

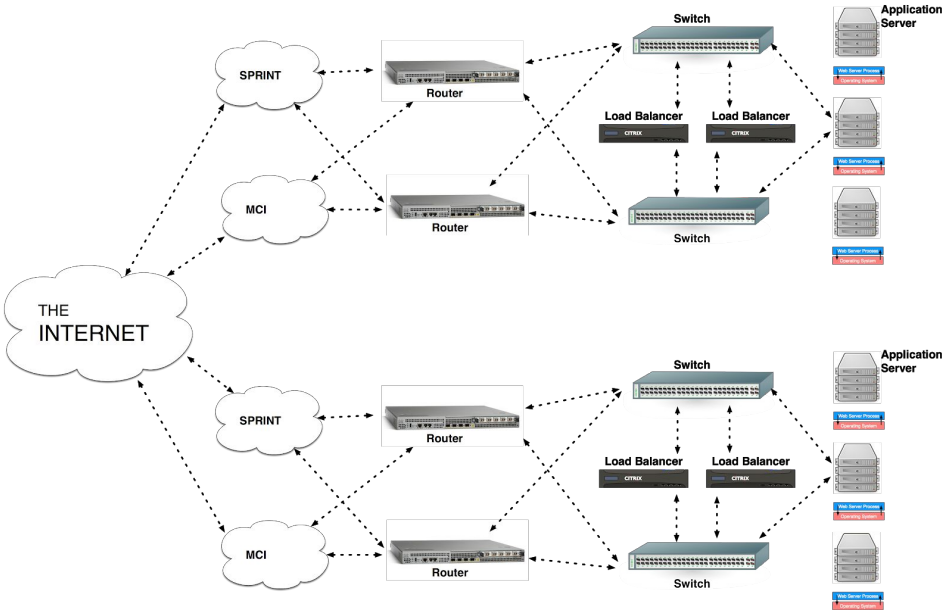
Programming with JavaScript, Node.js

We’ve covered a lot of ground



The use of load balancing in achieving horizontal scaling

We’ve covered a lot of ground

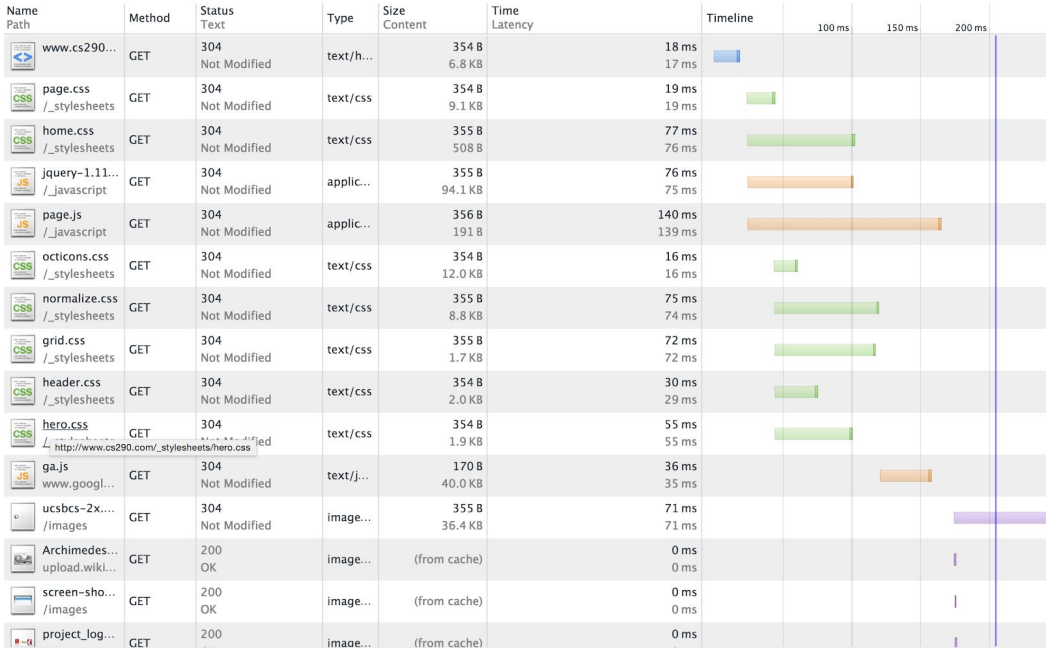


The architecture of a high availability, share nothing web application

We’ve covered a lot of ground

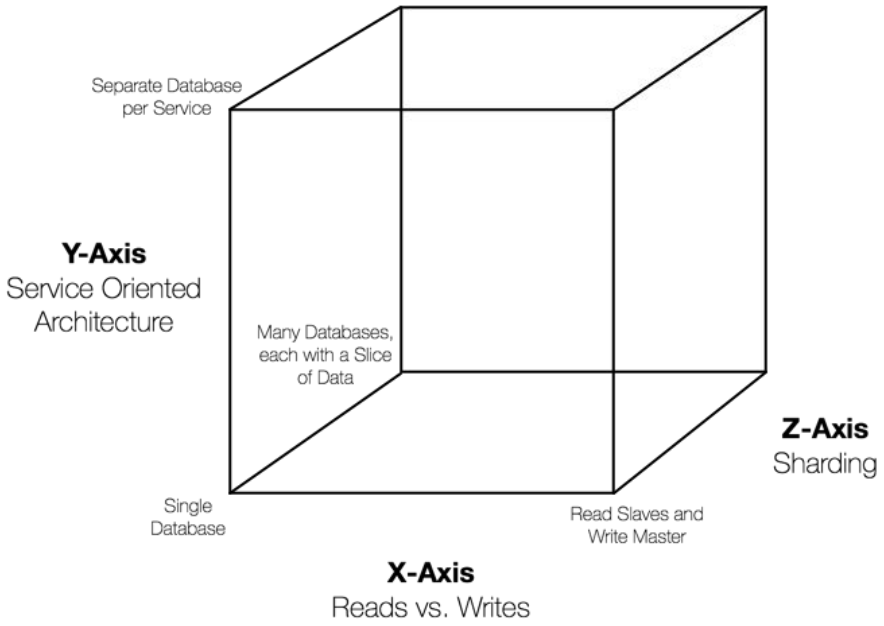
Designing an API

We’ve covered a lot of ground



All about caching, both on the client and the server

We've covered a lot of ground



Relational databases and how to scale them

Using ORMs to represent database models in code

... or not

We've covered a lot of ground

Cassandra: Storage

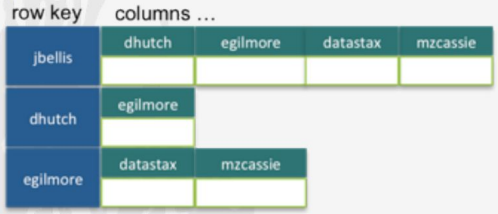
- Static Column Family

  Example: Users

Known column names

- Dynamic Column Family

  Example: friends



Dynamic column names

Scaling beyond relational databases: NoSQL stores

We’ve covered a lot of ground

How to load test your application and observe what happened

What I hope...

- If you're 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! And keep your users happy!
- If you ever want to start a company, I hope this course has given you 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/Friday for presentations
  - Make sure your whole team is present during your time window
- Please send papers to rothfels@cs.ucla.edu by **Wednesday @ 11:59pm**

Thank you!