

# CS 188

## Scalable Internet Services

Andrew Mutz

November 29, 2016



# Calendar Updates

**Final presentations are Thursday and Friday!**

**Writeups are due Thursday at 4pm, emailed to me:  
[andrew.mutz@cs.ucla.edu](mailto:andrew.mutz@cs.ucla.edu)**

**Thursday's presentations are here, Friday's are in  
Boelter 2760.**



# Calendar Updates

**If you are still load testing, make sure you shut down instances after use.**

**We are limited in number of concurrent servers.**



# Today's Agenda

- Content Delivery Networks
  - Motivation
  - Implementation
- Virtualization
  - Why it is increasingly popular
  - How it works



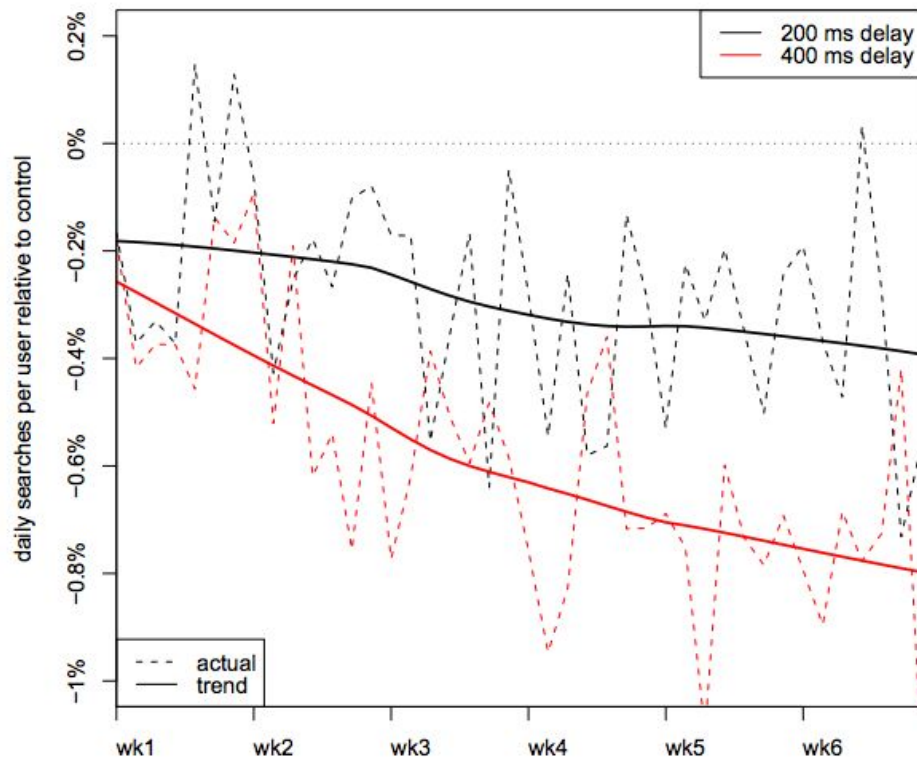
# Motivation

## 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 0.2-0.6%
- Delays reduce user engagement progressively
  - 200ms delay -> 0.22% trending to 0.36% fewer
  - 400ms delay -> 0.44% trending 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
- Effect on pagerank



# 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

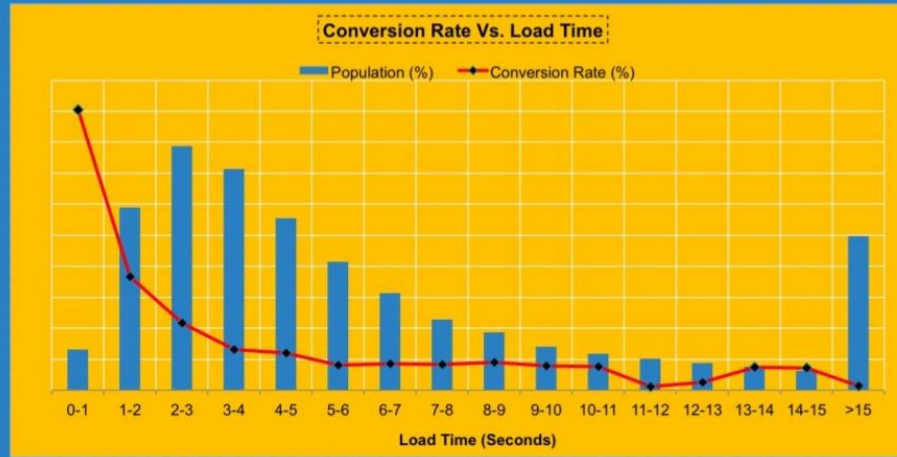


# Motivation

## 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

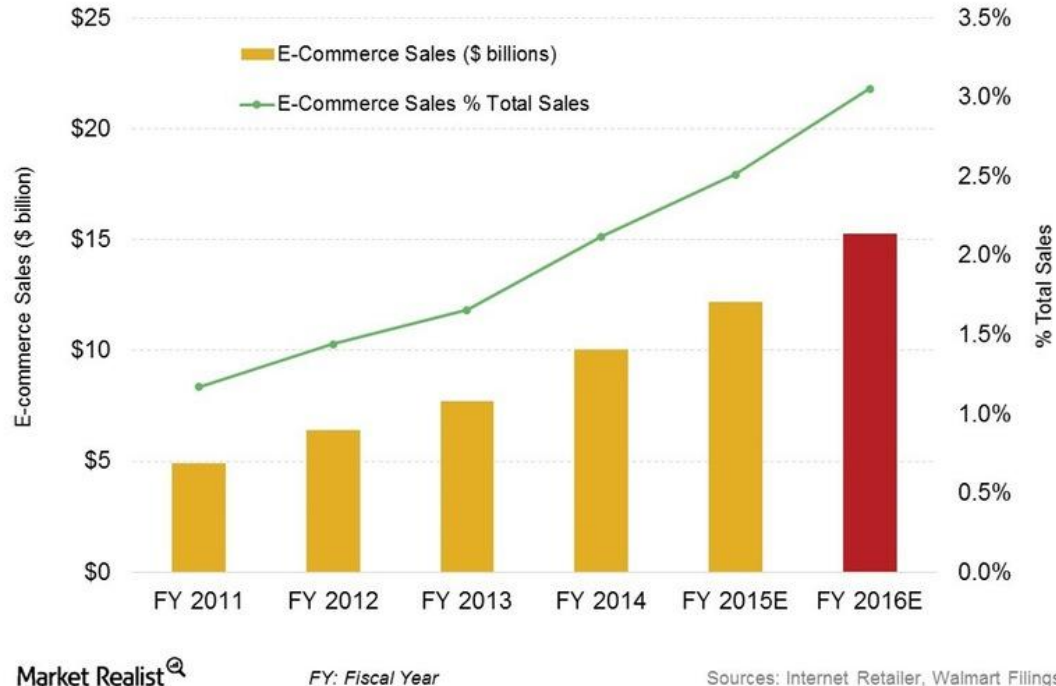
## Speed matters: Walmart

- 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

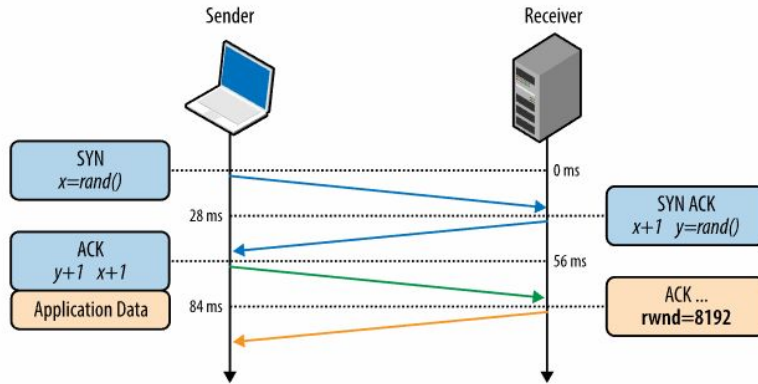


# Motivation

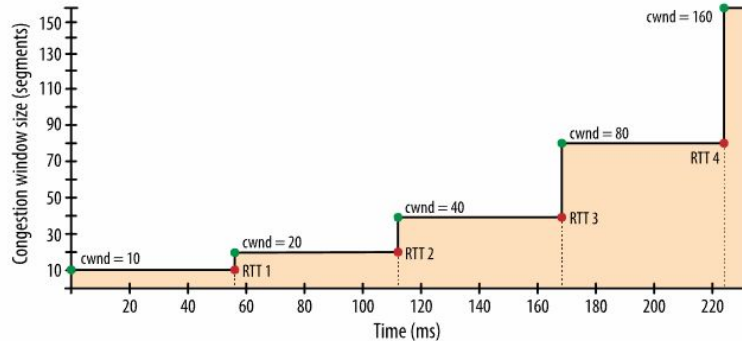
Walmart: E-Commerce An Important Sales Driver



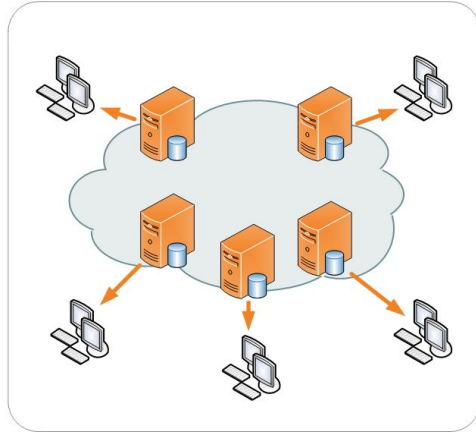
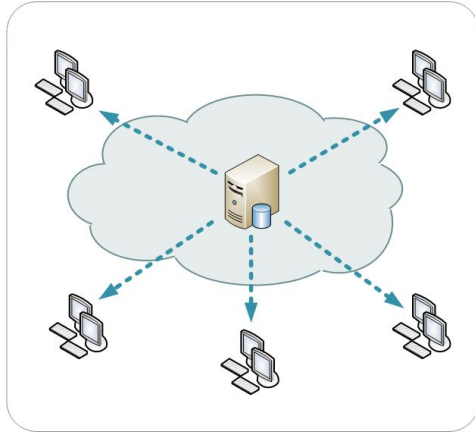
# Motivation



And while page load time is important, network latency continues to hold us back



# Motivation



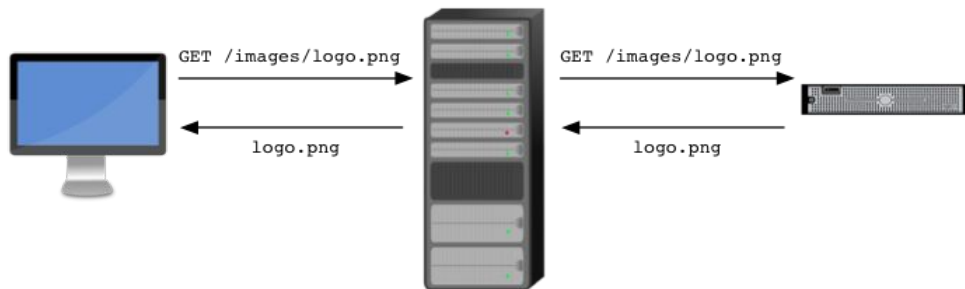
If our users are geographically distributed, how can we reduce latency?

- Bring the content closer to the users

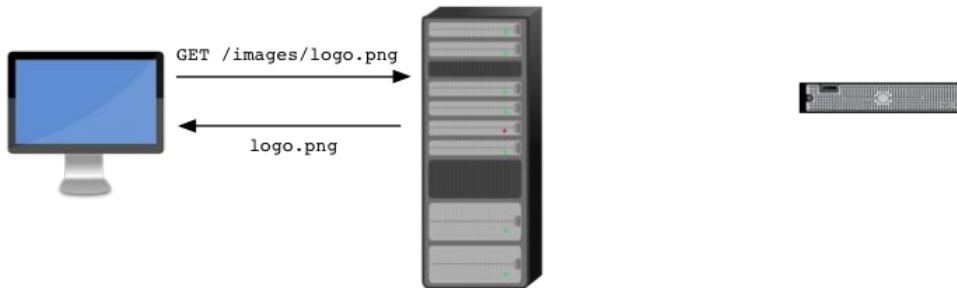


# Implementation

*First client request for logo.png*



*Second client request for logo.png*



Caching mechanism acts like an HTTP proxy.

HTTP headers guide caching decisions



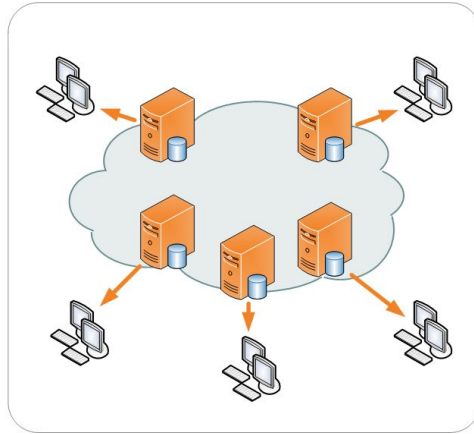
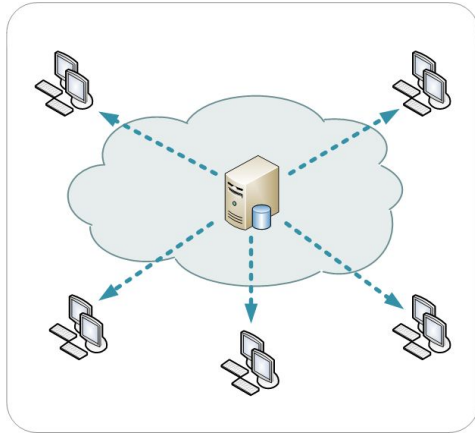
# Implementation

## Web Caching Fundamentals

- 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, range requests



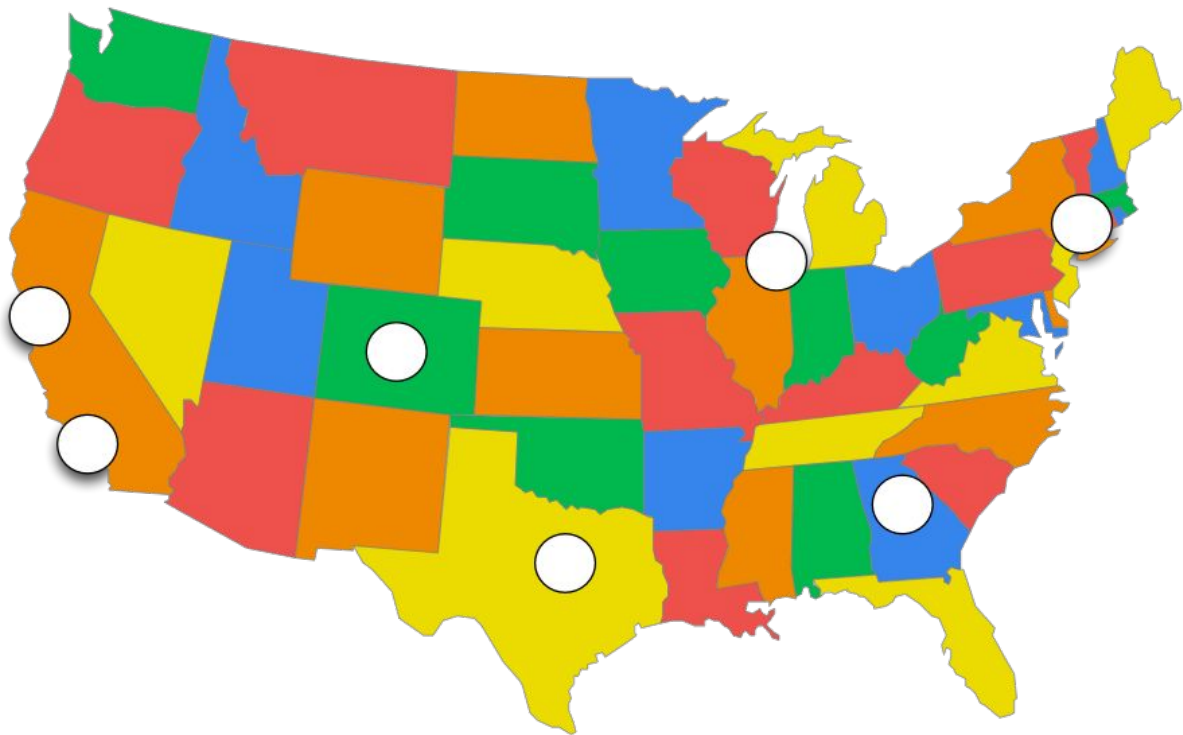
# Motivation



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



# Implementation



Two big questions:

- Which endpoint (“Point of Presence”, PoP) should we send them to?
- How should we send them there?





# Implementation: How

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

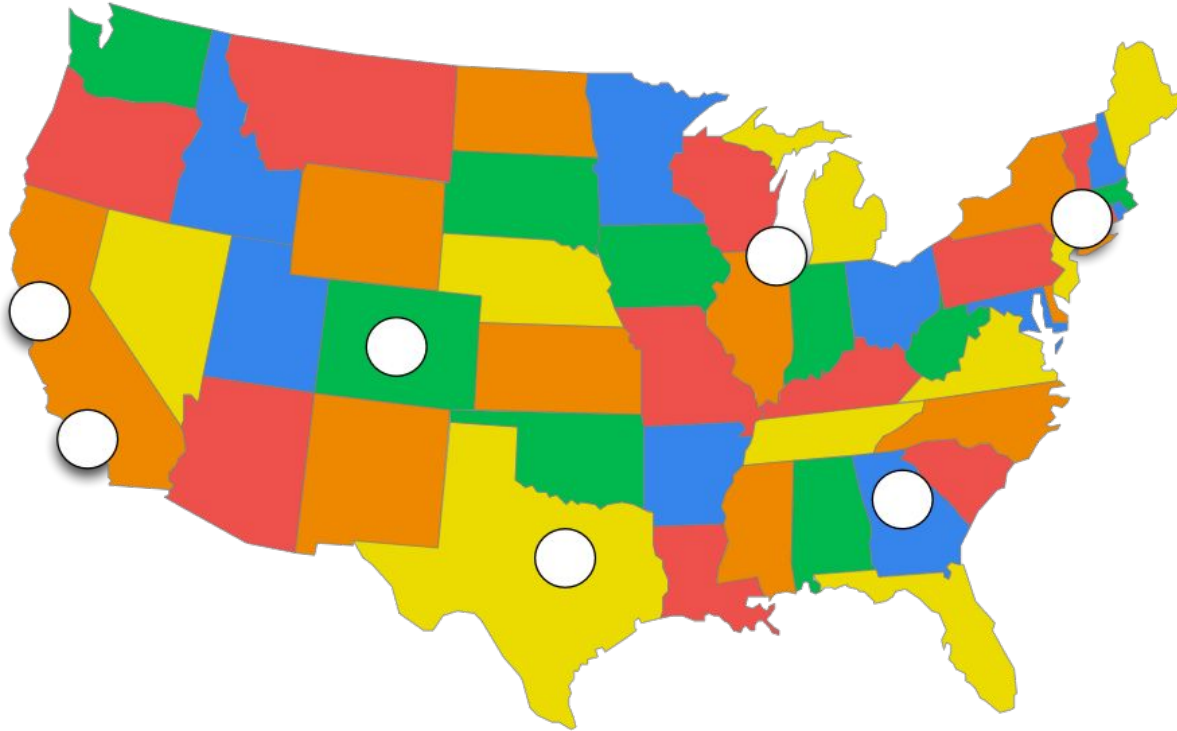
<http://www.example.com/images/logo.png>

Becomes:

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



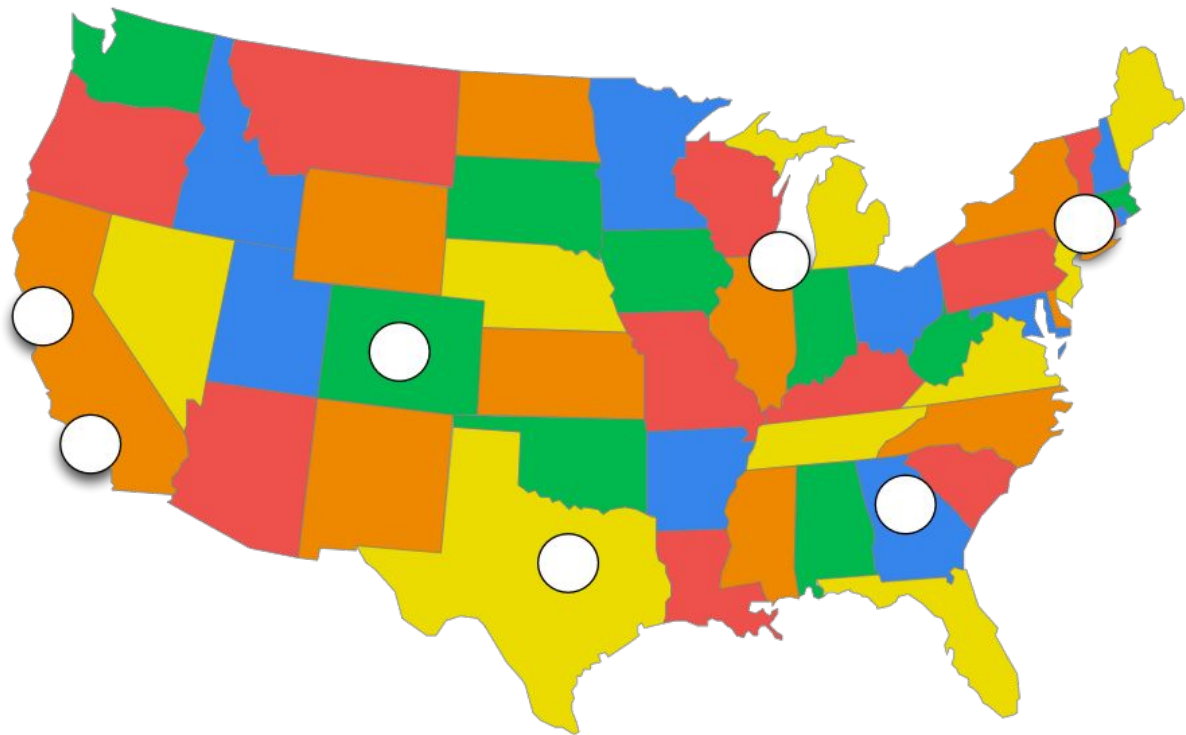
# Implementation: How



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



# Implementation

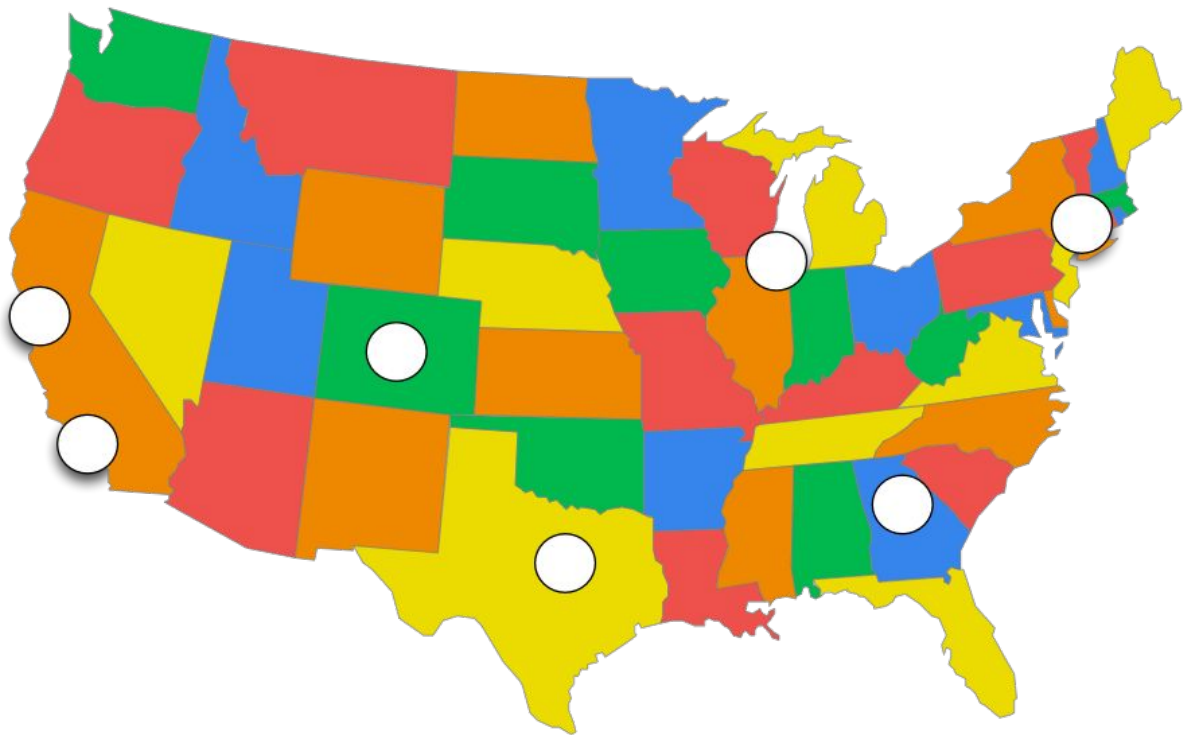


[www.example.com.akamai.net](http://www.example.com.akamai.net)

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.



# Implementation



- Low TTL on DNS means you can be dynamic
  - Resolve load issues on pops or cache servers
  - Take cache servers in and out of service
  - Take POPs in and out of service



# Implementation: Which

How do we choose which PoP to serve the content from?

- Geolocation
- Anycast
- Client performance



# Implementation: Which

- Geolocate the DNS resolver
- Send to PoP that is geographically closest to the DNS resolver
- Works okay, but clients don't always use nearby DNS servers
  - Example?

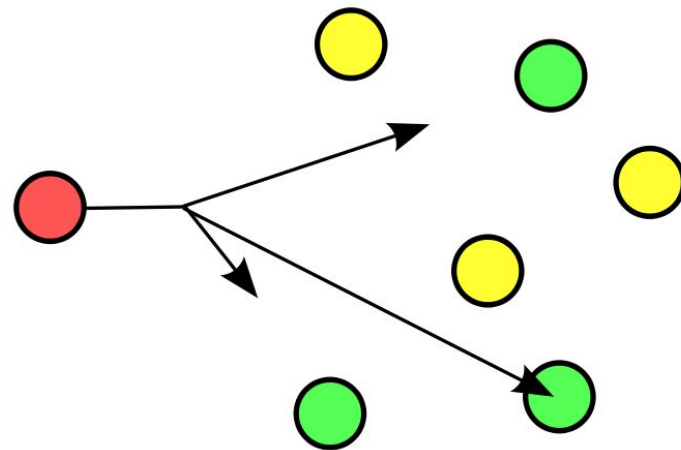


# Implementation: Which

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

BGP rules are set to route to nearest host

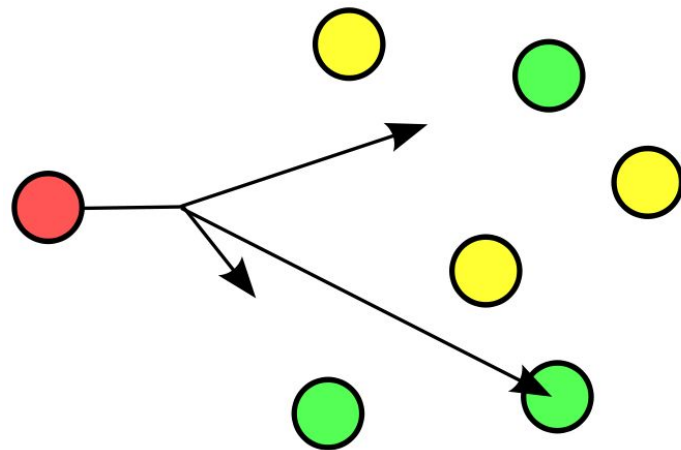
We can have all PoPs have the same IP, and use BGP and anycast to route to nearest PoP



# Implementation: How

## Challenges with anycast:

- A sequence of IP packets may end up at different hosts (BGP route changes)
  - Because HTTP is short-lived, not a big problem
- This will route to closest PoP, but closest might not be best
  - Packet loss, bandwidth could be issues

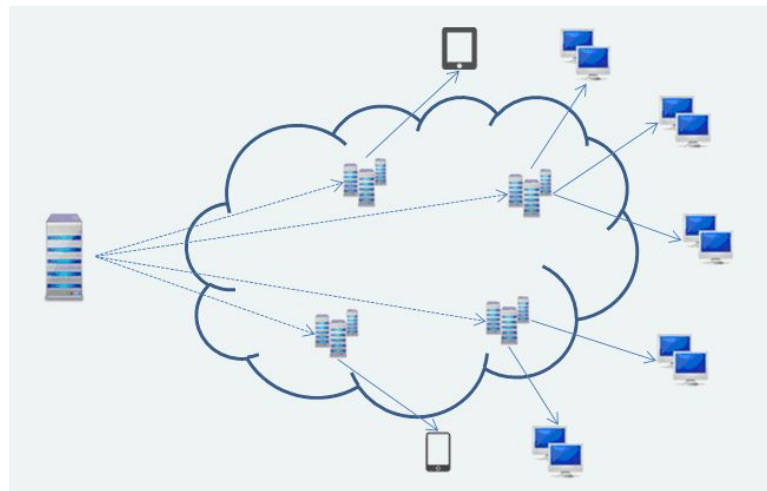




# Implementation: How

## 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



# Implementation: How

## How to use measurements?

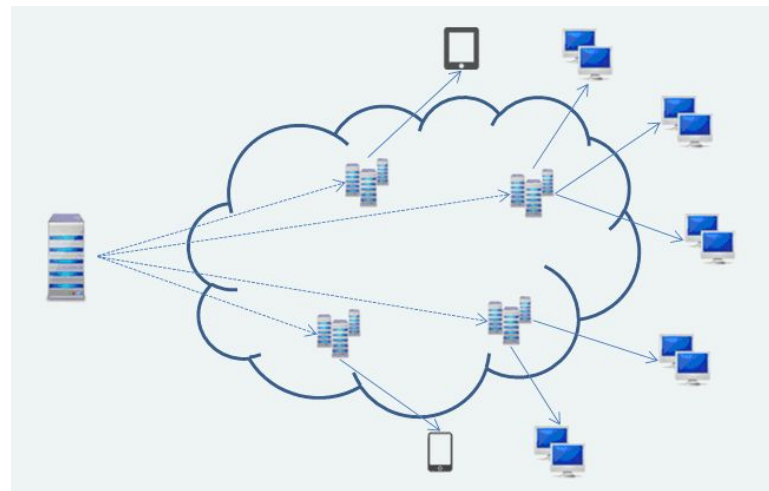
- Over time this becomes a huge dataset
- Want to remember history, but be adaptable
- Simple moving average

$$SMA = \frac{p_M + p_{M-1} + \cdots + p_{M-(n-1)}}{n}$$

- Exponential moving average

$$S_1 = Y_1$$

$$\text{for } t > 1, \quad S_t = \alpha \cdot Y_t + (1 - \alpha) \cdot S_{t-1}$$



# Implementation: How



# Implementation

## 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)
- MaxCDN



# Virtualization

## What is virtualization?

- Virtualization provides software with an execution context that resembles unfettered access to a collection of resources
  - Hardware-level case
  - Operating system-level case
- Like the UNIX process model, but with greater isolation guarantees, plus additional capabilities



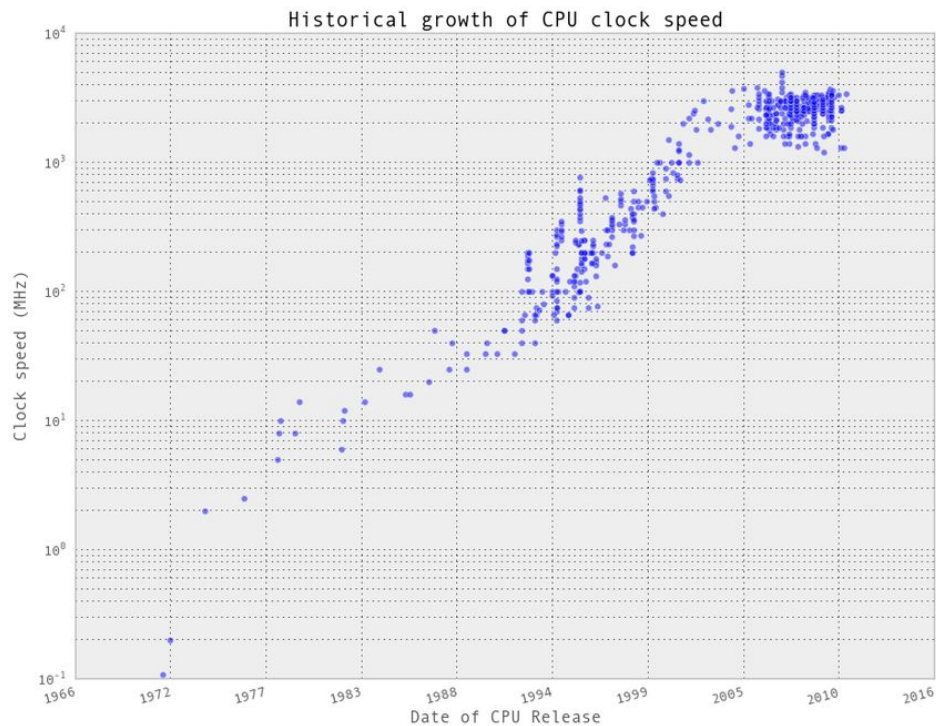
# Virtualization

## Why virtualize?

- Server consolidation
  - Power reduction
    - Even idle servers use about 60% of peak power
  - Flexibility in operating system choice for h/w virt
- Fast provisioning
- Snapshotting\*
- Live process migration\*
- Isolation
- Leverage multicore CPUs



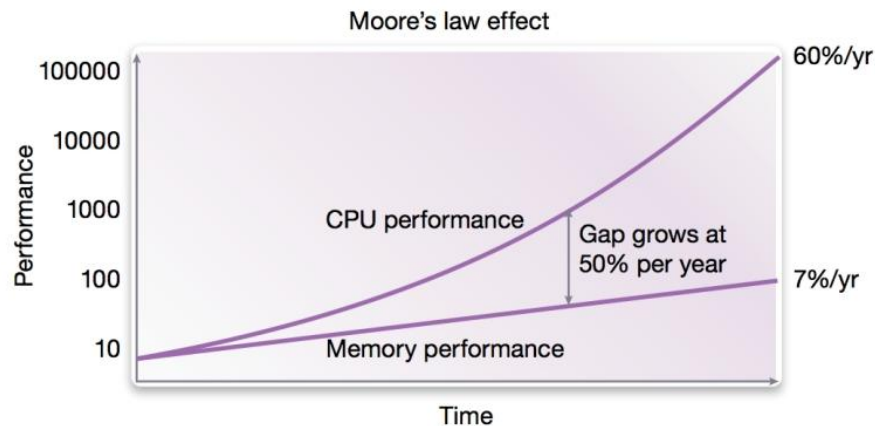
# Virtualization



# Virtualization

## “The memory wall”

- CPU speed growing much faster than memory speed
- Memory-heavy apps hit hardest
- In the limit, though, all apps affected to some degree





# Virtualization

## “The ILP wall”

- Many hardware techniques for parallelizing a single instruction stream
- However, not all instructions can be executed in parallel
- Data dependencies / “hazards”
  - Software dependent

$$x = a + b$$

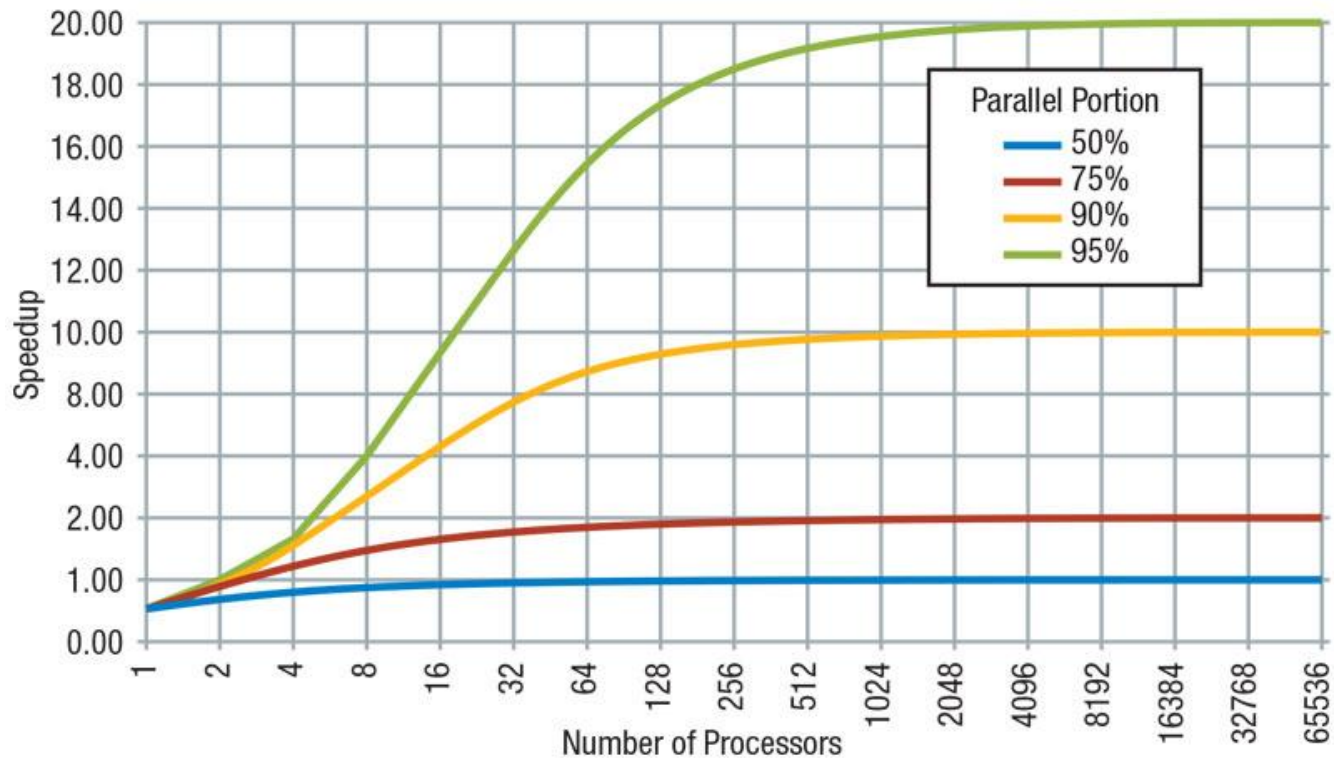
$$y = c + d$$

$$z = x * y$$



# Virtualization

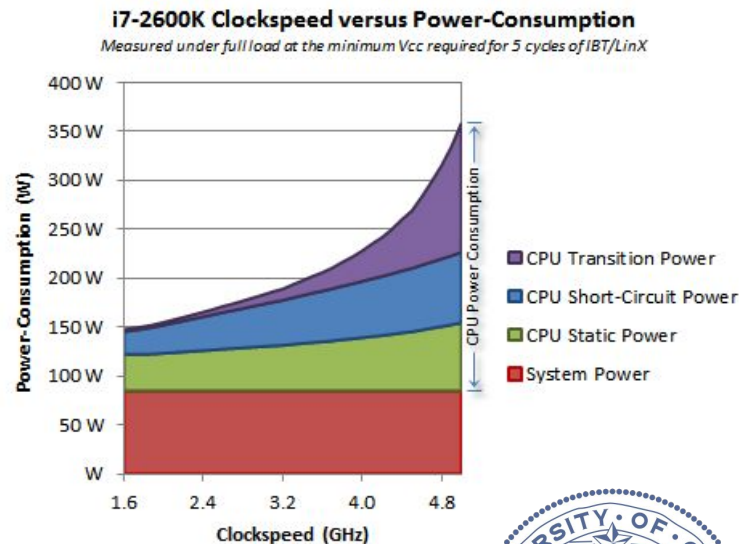
Amdahl's Law



# Virtualization

## “The power wall”

- Superlinear growth in power as a function of frequency
- Increased power budget
- Increased cooling costs
- Not offset by otherwise marginal perf improvement (viz memory, ILP)



# Virtualization

## Virtual machines can exploit multi-core

- Clock speed has levelled out, but Moore's law persists
- Instead of deeper pipelines, spec execution units, we have multiple CPU cores on a single die
  - Many CPUs in one chassis (one power budget)



# Virtualization

## Many types of virtualization

- Hardware Supported
  - VT-X, AMD-V
- Paravirtualization
  - Xen hypervisor
- Operating System
  - Linux LXC, FreeBSD Jails, Docker



# Virtualization on AWS

AWS uses both paravirtualization and hardware virtualization

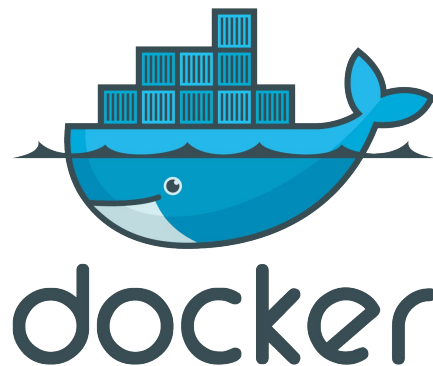
- AMIs are labelled either (PV) or (HVM)
- Allows instances to spin up fast
- PV traditionally faster, while HVM supports any OS



# Virtualization: Docker

Docker is an increasingly common technology for OS-level virtualization

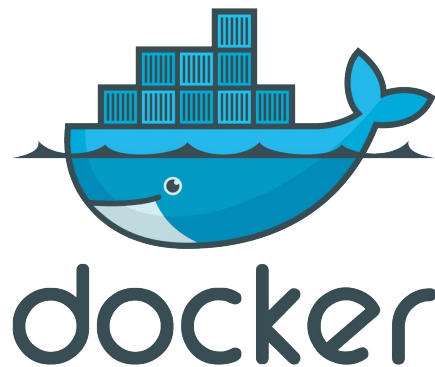
- Uses LXC: Namespaces and Cgroups for virtualization
- Uses Dockerfiles to extend and modify container images
- Has a globally accessible repository of images (Dockerhub)
- You have been using it on Cloud9



# LXC: Namespaces + Control Groups

## Namespaces

- Limit the scope of addressable objects
- Inherited by subprocesses
- Nestable
- Can construct mappings between enclosing context
  - Allows uid 0 inside a container
  - Allows multiple containers to all listen on “port 80”

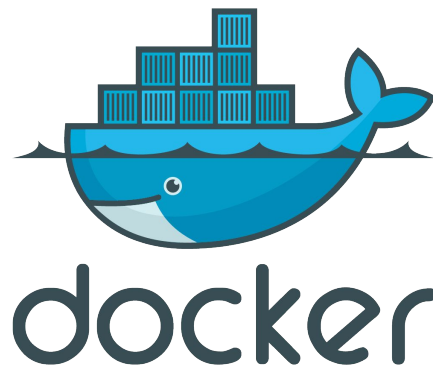




# LXC: Namespaces + Control Groups

## Control groups (cgroups)

- Limit, account for, isolate resource usage of a collection of processes
- Examples
  - CPU: pin a container's processes to a subset of CPU cores
  - Devices: allow/deny access to specific devices
  - Memory: Limit memory use by tasks in a cgroup
  - I/O: proportional weight, throttle by bps or iops
  - Network: limit/reserve bandwidth of a container



# Dockerfiles

```
FROM ruby:2.3
```

```
apt-get install -y postgresql-client
```

```
WORKDIR /usr/src/app
```

```
COPY Gemfile* ./
```

```
RUN bundle install
```

```
COPY . .
```

```
EXPOSE 3000
```

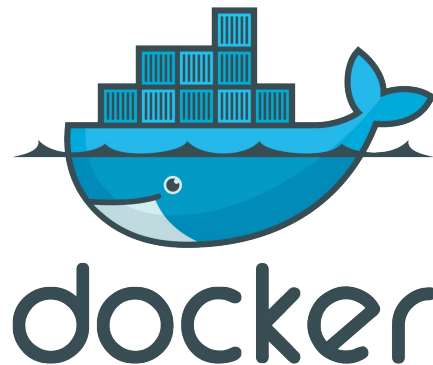
```
CMD ["rails", "server", "-b", "0.0.0.0"]
```

---

```
% docker build -t my_image .
```

```
% docker run my_image
```

```
% docker run -it my_image /bin/bash
```

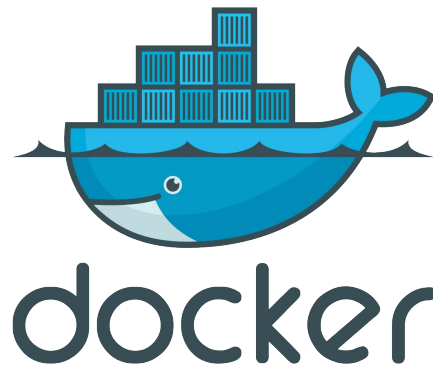


# Docker-compose.yml

```
version: '2'
services:
  web:
    build: .
    ports:
      - "3000:80"
  db:
    image: postgres
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
```

---

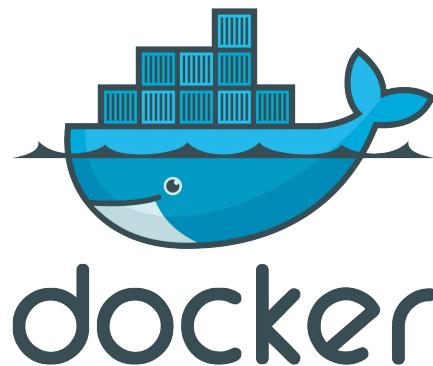
```
% docker-compose up
% docker-compose down
% docker-compose run web
```



# Docker orchestration

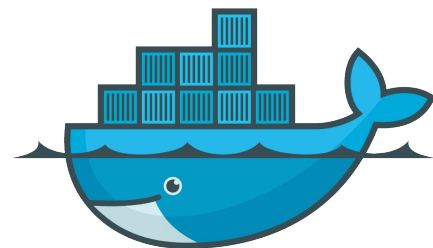
Docker as deployable units is even more powerful across a fleet of machines

- Needs tooling to connect different containers to each other, known as “orchestration”
  - Docker Swarm
  - Kubernetes
  - CoreOS Fleet
  - Mesosphere Marathon



# Docker conclusion

In summary, docker gives you great tooling for building your application using reusable building blocks.



docker

When combined with orchestration, allows complex distributed systems to be in source control.



# 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

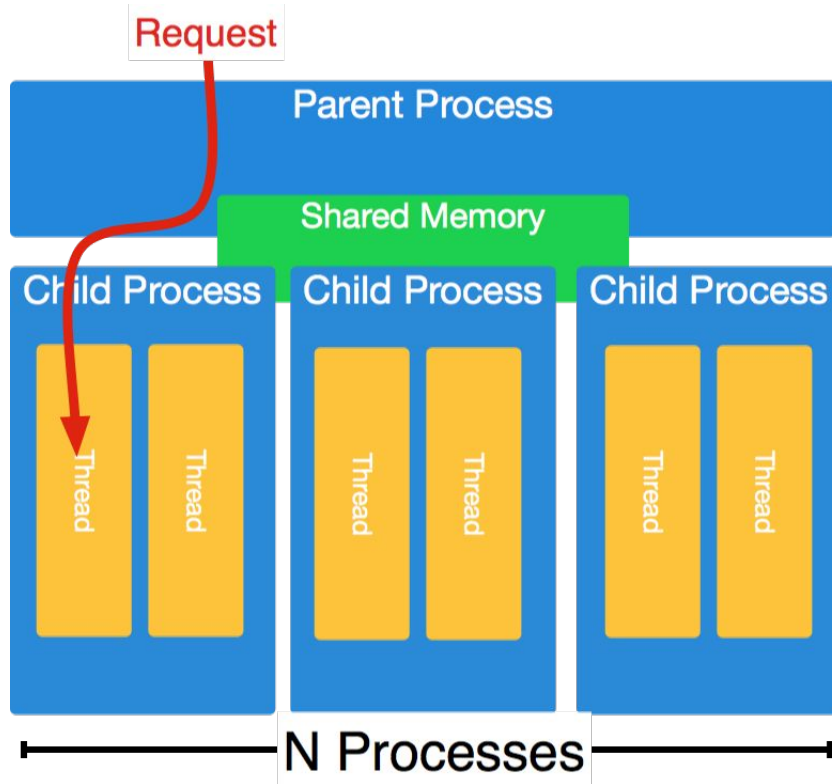
You've got a web application that is becoming increasingly popular and performance is degrading.

**What do you do?**





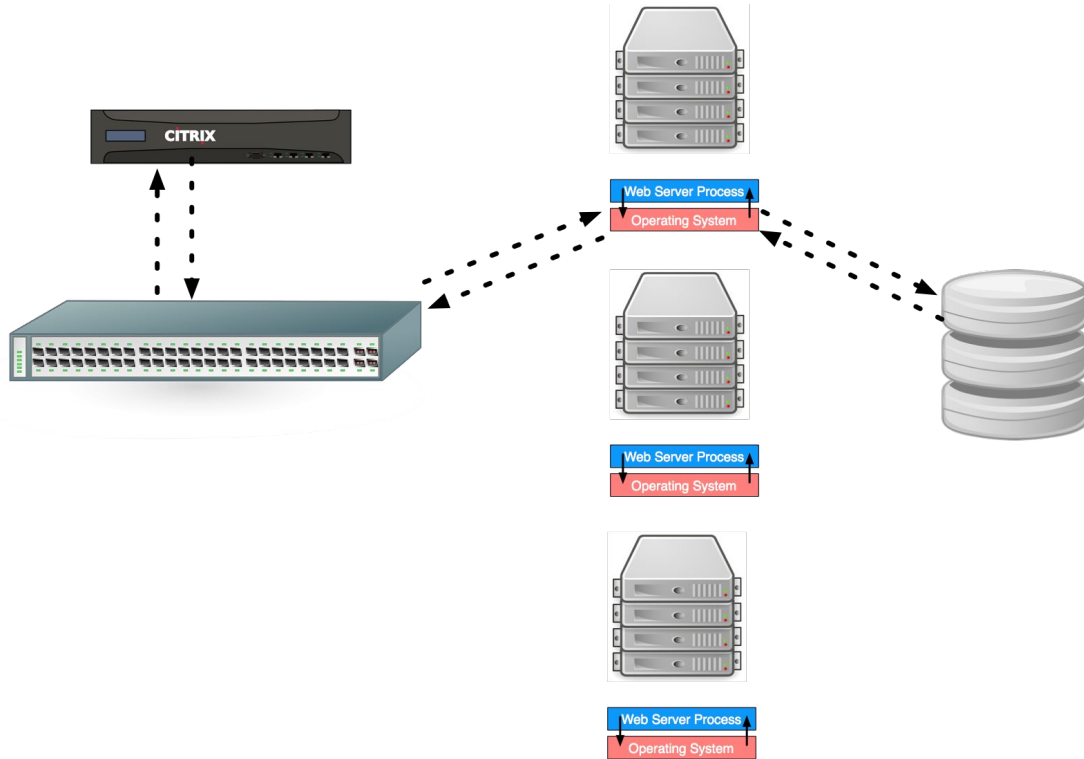
# We've covered a lot of ground



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



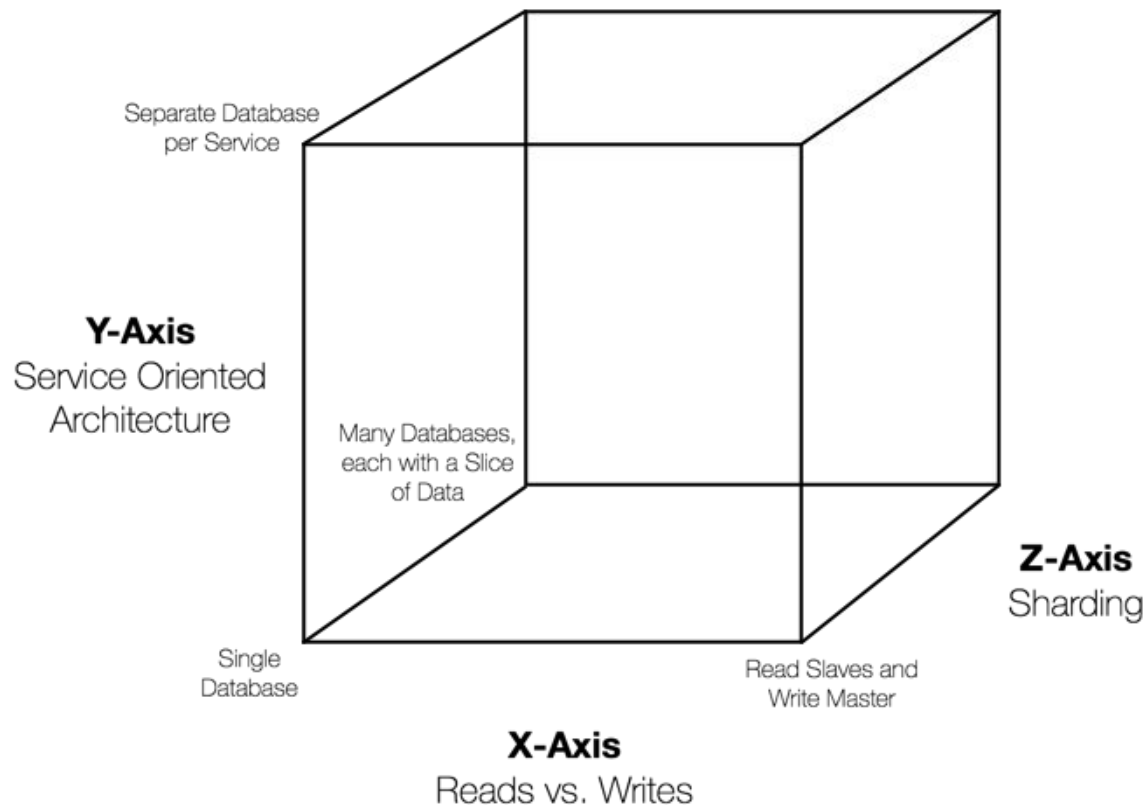
# We've covered a lot of ground

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 /_stylesheets	GET	304 Not Modified	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 OK	image...	(from cache)	0 ms 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

## Cassandra: Storage

- Static Column Family

Example: Users

row key	columns ...			
jbellis	name	email	address	state
	jonathan	jb@ds.com	123 main	TX
dhutch	name	email	address	state
	daria	dh@ds.com	45 2 <sup>nd</sup> St.	CA
egilmore	name	email		
	eric	eg@ds.com		



Known column names

- Dynamic Column Family

Example: friends

row key	columns ...			
jbellis	dhutch	egilmore	datastax	mzcassie
dhutch	egilmore			
egilmore	datastax	mzcassie		

Dynamic column names

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
- HTTP 2.0 & QUIC
- CDNs and Virtualization





# 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](https://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](mailto:andrew.mutz@cs.ucla.edu)

