



RICE<sup>®</sup>

# Web Development

COMP 431 / COMP 531

## Scaling

Scott E Pollack, PhD

April 12, 2016

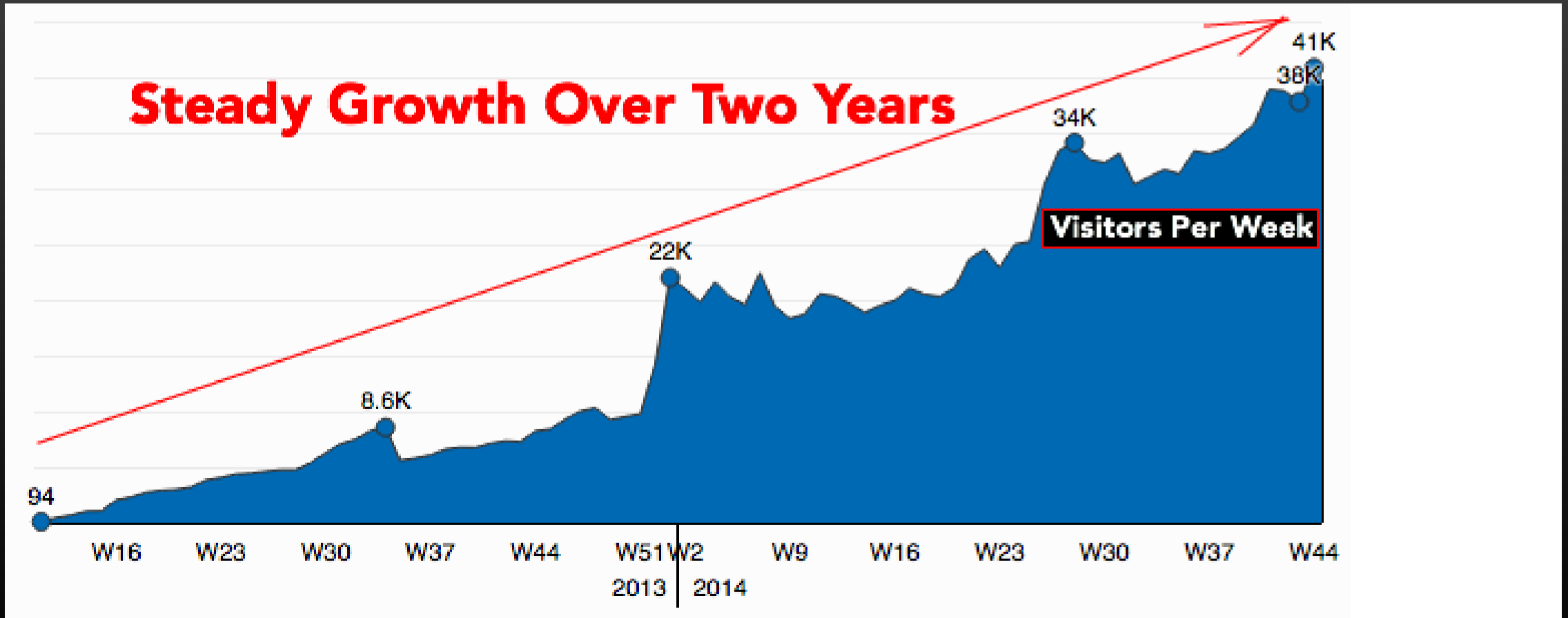
# Part IIb – Back End Development

- Homework Assignment 7 (Integrated Web App)
  - Due TONIGHT 4/12
- COMP 531 Paper and Presentations 4/21
  - Due Thursday 4/21 before class
- Homework Assignment 8 (Final Full Web App)
  - Due Thursday 4/28

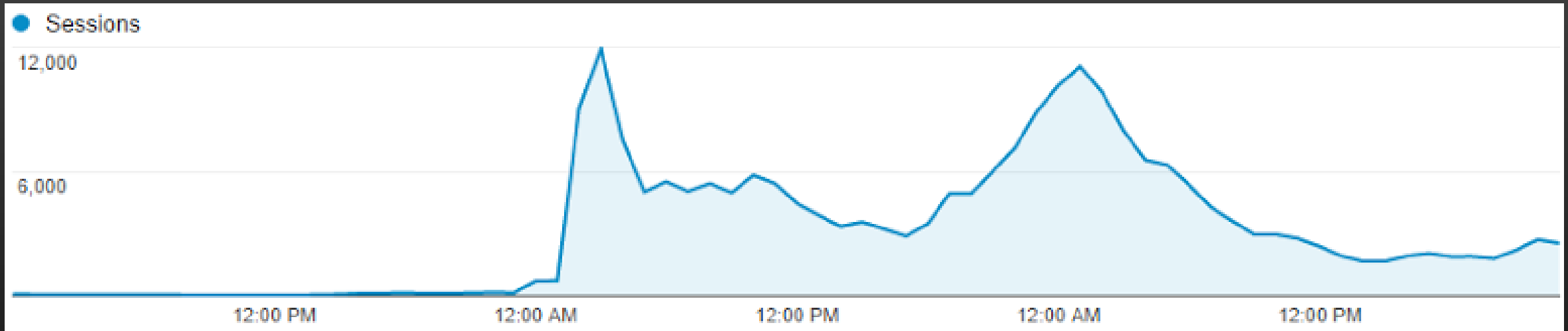


**Scalability** is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth

# What we all want



# Things to fear



"This is [Have I been pwned?](#) (HIBP) going from a fairly constant ~100 sessions an hour to... 12,000 an hour. Almost immediately."

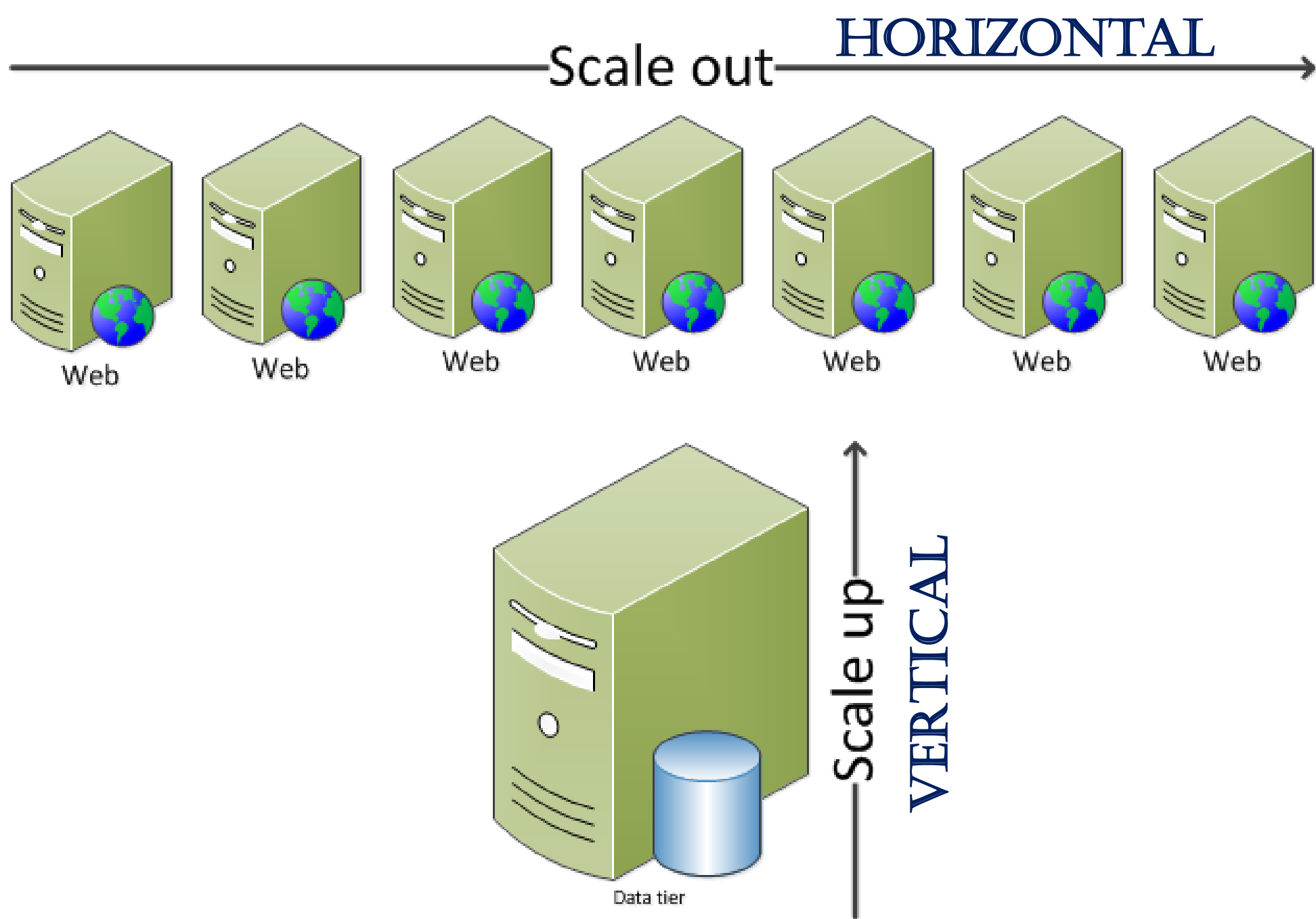
<http://www.troyhunt.com/2014/09/10-things-i-learned-about-rapidly.html>

<http://www.troyhunt.com/2013/12/micro-optimising-web-content-for.html>

# Problems at Scale

# Problems at Scale

- **Network can't handle the load**
  - Your service provider can't send your server all the requests
  - Client requests time out
- **Server can't handle the request load**
  - Client requests time out
  - People are disappointed and look elsewhere
- **Uptime**
  - One box = single point of failure
- **Software Inefficiency**
  - Some bad algorithms slow your response
  - At scale this becomes a problem
- **Limited Compute**
  - Even if efficient we can run out of compute
- **Limited Memory**
  - Store session information in memory
  - Too many sessions out of memory
  - Watch out for memory leaks
- **Disk I/O**
  - Always slow compared to RAM
  - Limit database read/write
  - Databases may still catch fire with overuse and limited throughput





# Solutions?

- **Network can't handle the load**
  - Use another provider
  - Content Delivery Network
- **Server can't handle the request load**
  - Content Delivery Network
  - Scale Horizontally: Add servers
- **Uptime**
  - Scale Horizontally: Add servers
- **Software Inefficiency**
  - Performance test algos and rewrite
- **Limited Compute**
  - Scale Vertically: Get beefier machines
  - Scale Horizontally: Distribute load
- **Limited Memory**
  - Scale Vertically: Get beefier machines
  - Scale Horizontally: Distribute load
  - Refactor memory usage
  - Memory profile and rewrite for leaks
- **Disk I/O**
  - Cache, cache, cache!
  - Horizontally scale databases
  - Cache, cache, cache!

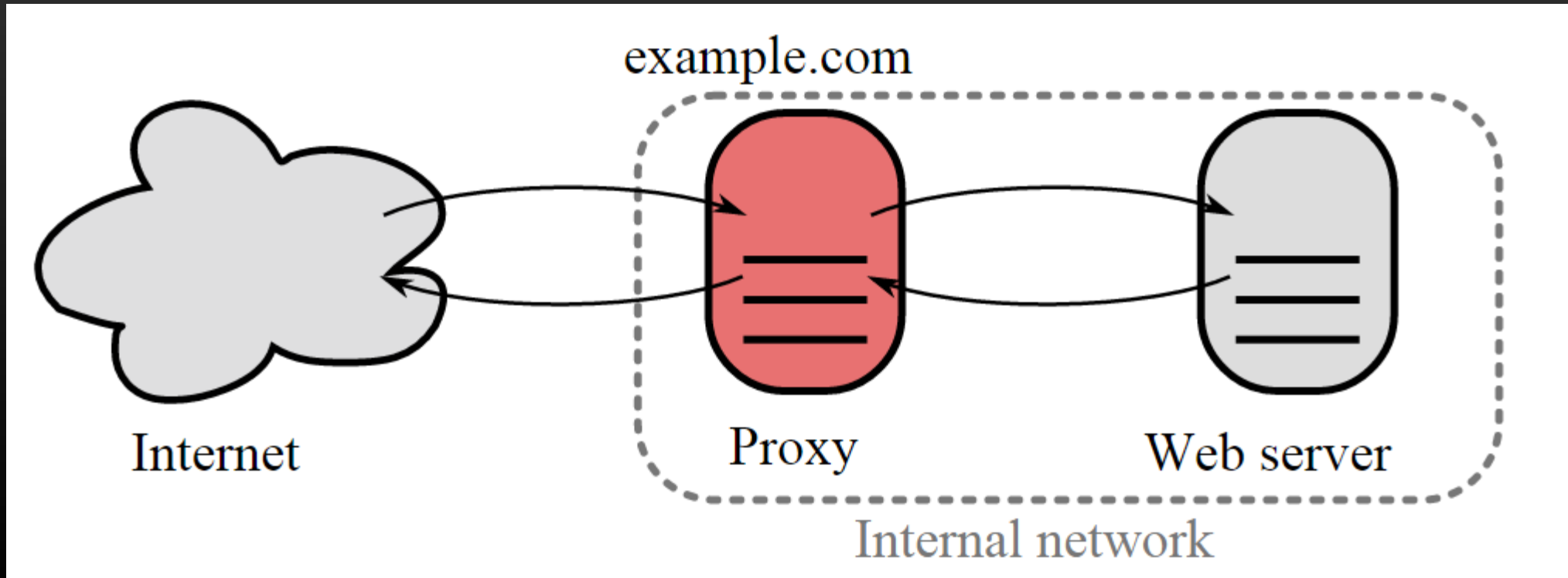
# Use a Content Delivery Network!

- Public CDN for public files
- For static files use a CDN too!
  - Amazon CloudFront
  - Azure CDN
  - Akamai
  - CloudFare
  - ...

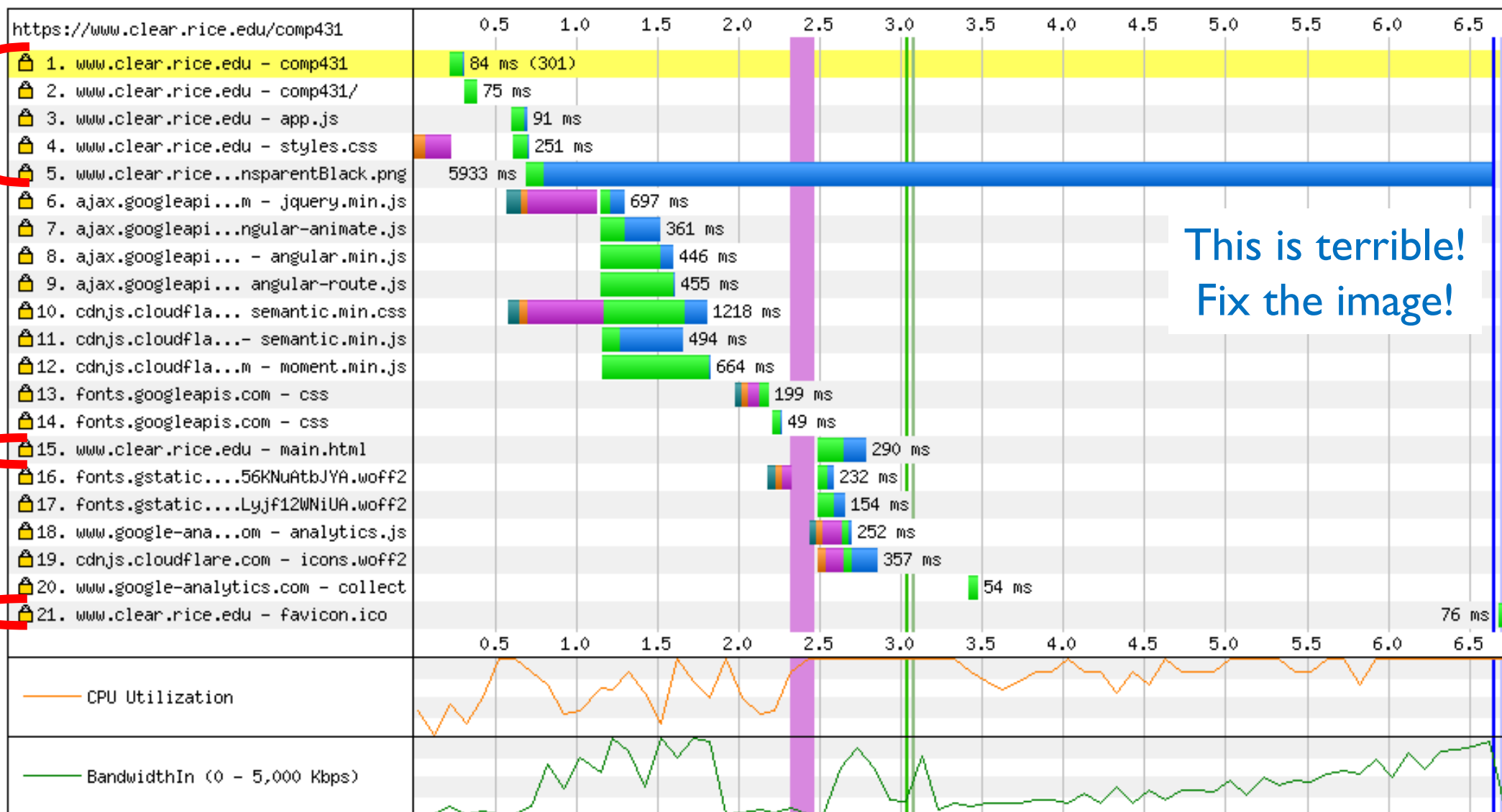
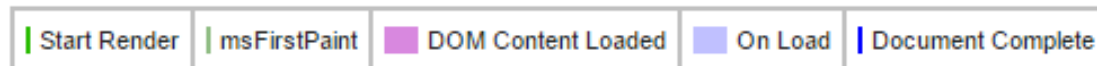


# Web/HTTP Accelerators

- A caching layer that sits between the client and our server
- Be sure to set expiry dates for cache items
- Of the Top 10K sites in the web, around a tenth use Varnish



# Waterfall View

































This is terrible!  
Fix the image!

# 14 Rules for Faster-Loading Web Sites

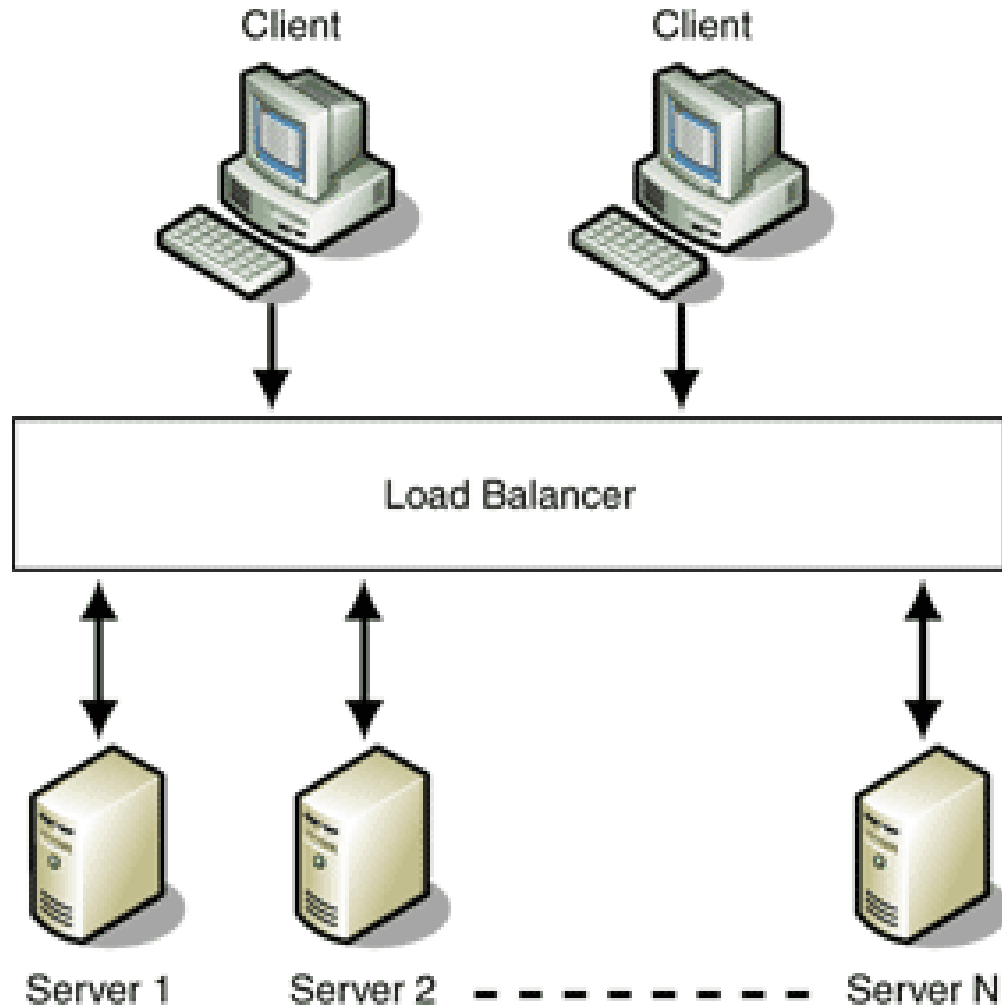
Steve Souders

1. Make Fewer HTTP Requests
2. Use a CDN
3. Add an Expires Header
  - *Makes content cacheable*
4. Gzip Components
5. Put Stylesheets at the Top
  - *Prevent unstyled flash of content*
6. Put Scripts at the Bottom
  - *Prevent delayed downloads*
- ~~7. Avoid CSS Expressions~~
  - *Internet Explorer only*
8. Make JavaScript and CSS External
9. Reduce DNS Lookups
10. Minify JavaScript
11. Avoid Redirects
12. Remove Duplicate Scripts
13. Configure ETags
14. Make AJAX Cacheable

# Parallel Downloading

Name	Sta...	Domain	Size	Time	Timeline – Start Time	400.00 ms	6
 TabApp	301	www.clear.rice.edu	288 B	67 ms			
 TabApp/	200	www.clear.rice.edu	2.3 KB	32 ms			
 jquery.min.js	200	ajax.googleapis.com	28.9 KB	62 ms			
 bootstrap.min.css	200	maxcdn.bootstrapcdn.com	25.3 KB	230 ms			
 bootstrap.min.js	200	maxcdn.bootstrapcdn.com	12.0 KB	259 ms			
 angular.js	200	cdnjs.cloudflare.com	260 KB	329 ms			
 angular-route.min.js	200	cdnjs.cloudflare.com	2.5 KB	242 ms			
 angular-resource.min.js	200	cdnjs.cloudflare.com	2.2 KB	240 ms			
 app.js	200	www.clear.rice.edu	1.3 KB	96 ms			
 srv.js	200	www.clear.rice.edu	1.0 KB	97 ms			
 simpleControllers.js	200	www.clear.rice.edu	867 B	91 ms			
 serviceControllers.js	200	www.clear.rice.edu	1.0 KB	124 ms			
 httpController.js	200	www.clear.rice.edu	2.3 KB	129 ms			
 resourceController.js	200	www.clear.rice.edu	2.9 KB	126 ms			
 directiveController.js	200	www.clear.rice.edu	947 B	138 ms			

# Horizontal Scaling



$$(\text{server throughput}) = \frac{(\text{site throughput})}{N}$$

DNS points to load balancer

Individual servers have arbitrary addresses, but are registered with LB

Dyno Scaling :

ON

When Dyno Scaling is on your app will be scaled to match the current Recommendation.

Dyno Range :

6

30

The low end is the fewest number of dynos you think your app can safely run on.

The high end is the largest number of dynos you are willing to pay for. We won't scale beyond this, we promise.

Expected response time:

350

This is about how long (in milliseconds) your app usually takes to respond, on average. We will use this to determine if more dynos might help so if you are overly optimistic you may end up with more dynos than you need.

Sample window:

8

A sample window value of 1 will react to smaller changes much more frequently, while a value of 10 reacts to traffic more smoothly. Be warned, for apps with larger slug size it is not advisable to scale up and down frequently. A setting of 5-6 is good start for most websites.

Dyno increase rate:

3

The higher the number the more often dynos will be added.

Dyno decrease rate:

3

The higher the number the more often dynos will be removed.

# Scaling on Heroku

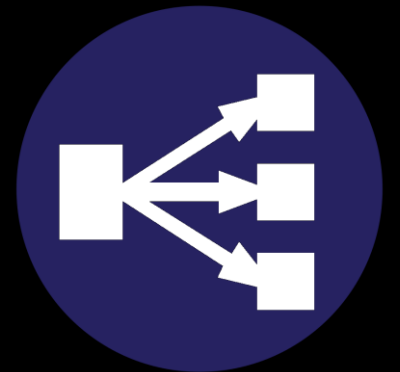
## *addon Adept Scale*

Save Settings



# Load Balancing

- Distribute requests across multiple servers
- Round-robin DNS
  - Multiple IPs associated in DNS
  - Any query to a DNS server will delegate to a chosen IP
  - Subsequent query will return the next IP in list
- Load Balancing Switch
  - Hardware solution
  - Could sniff cookies for specific IP redirection
- Load Balancing Server (haproxy)
  - Software solution
  - E.g., Nginx proxies requests to app server
  - Could sniff headers (cookie, origin, etc) for specific IP redirection



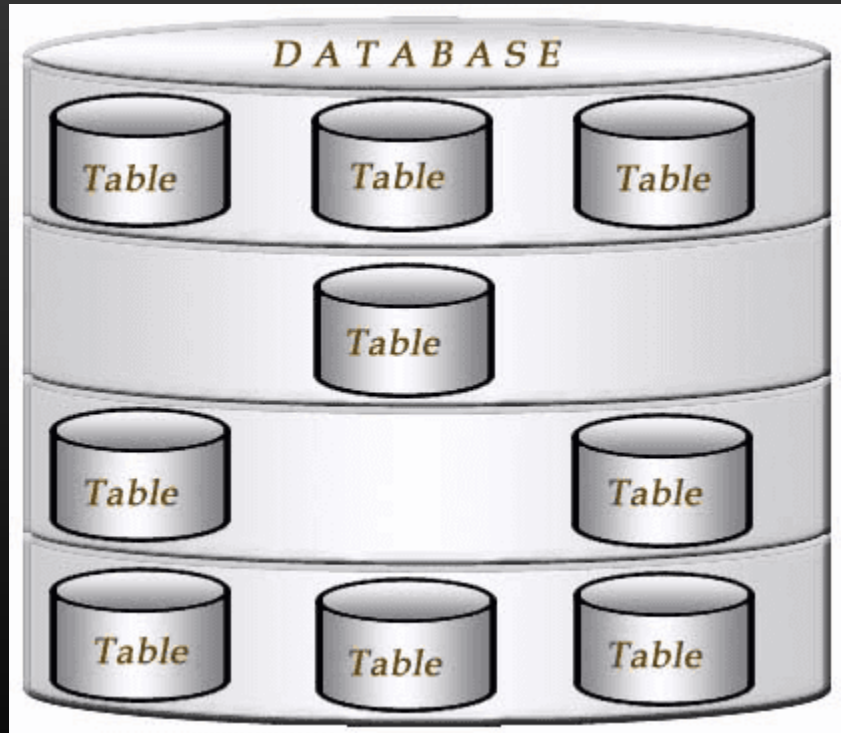
# Session Management?

- Stateless servers
  - Sessions are stored in cookie on frontend
  - Then we don't have session management
  - Cookies are limited in size
  - LocalStorage?
- Otherwise?
- We keep data on server, but then either
  - Need LB to route calls to correct server
  - Or query server to server to get data
- OR any additional data we query from database
  - ... cache database queries!



# Scale Storage

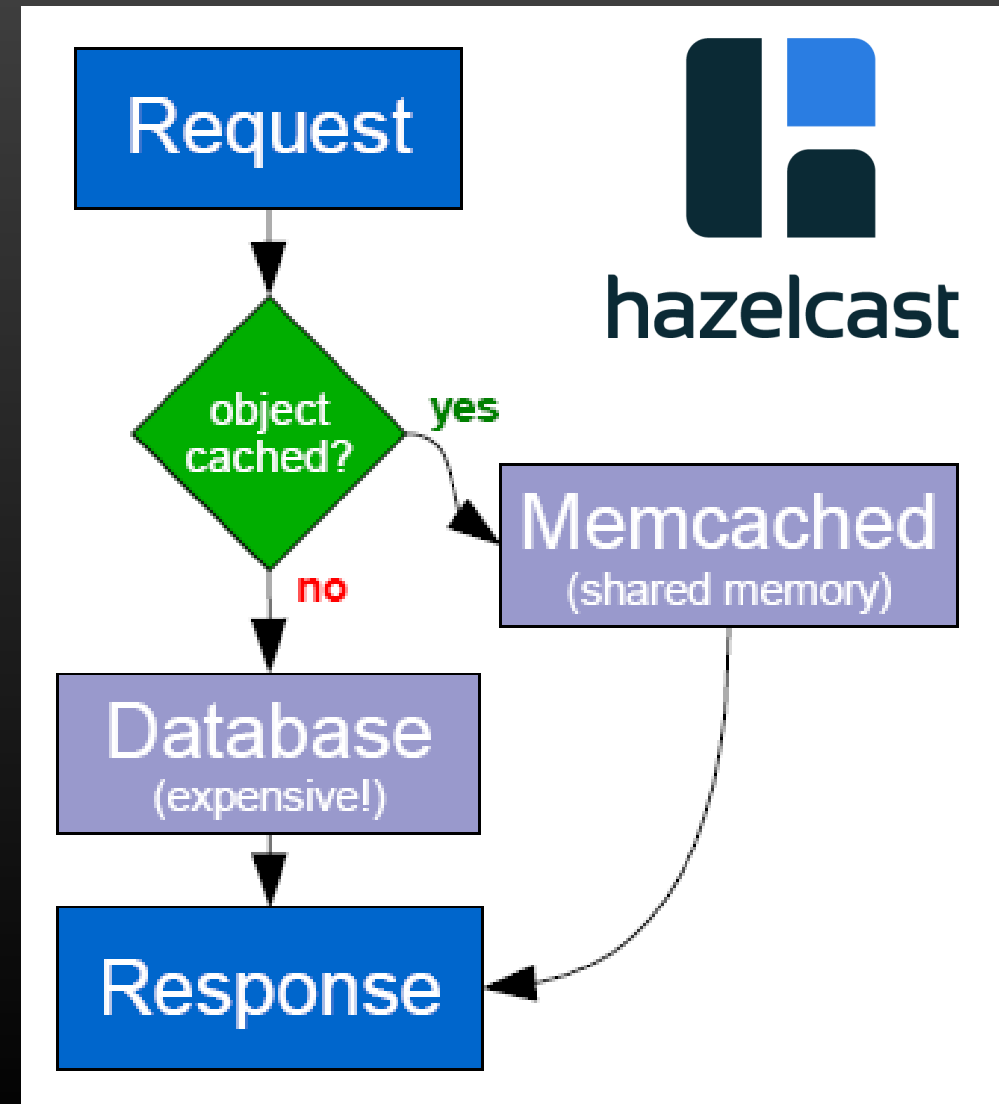
- One write master, many reader slaves
- Shard the database



# Memory Caching: ~~memcached~~

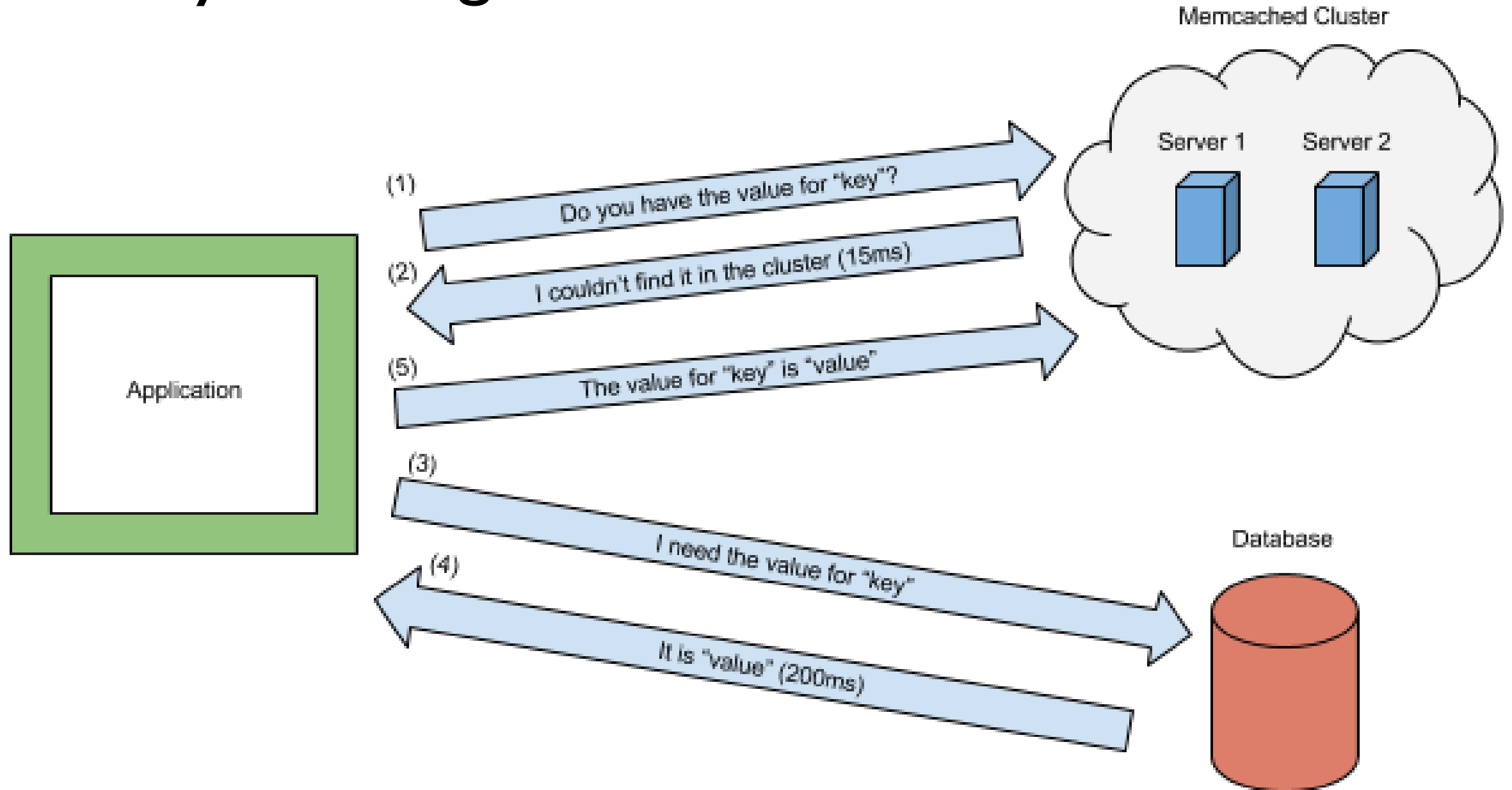


- Key-value store
- Least Recently Used cache
- Reduce database load
- Run on separate host for resource balancing
- Shared among app servers
- Distribute load by running multiple instances



# Memory Caching

## Memcached Usage

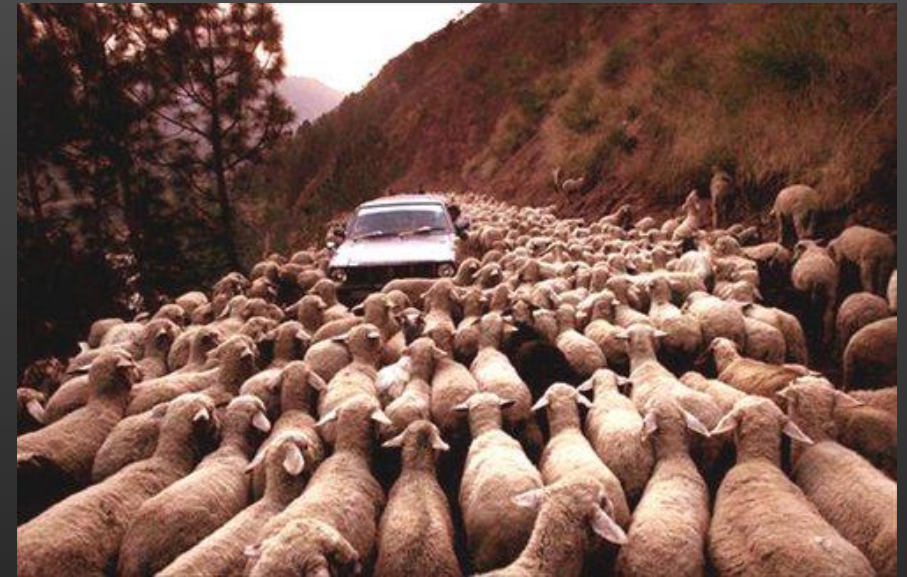




# Gotcha!

- Consider situation where cache is missing value.
- Multiple requests come in for that key
  - **Cache stampede!**
- All requests miss the cache miss and head towards the data store
  - Queries **dogpile** on database and nobody happy
  - Multiple queries for same content
  - **Congestion collapse** may occur
- Contention in concurrent writes to cache

*Solutions? Locking and prefetching*



## An aside...

- Big data is big
- **Really big!**
- An unstructured document store is difficult to scale
  - Think about searching for a document among millions or trillions!
- But RDBMS don't scale so well either
  - And certainly are not as extensible (think about changing the schema)
- Columnar based stores have advantages here





# What is Apache Cassandra

- Masterless Architecture with read/write anywhere design
- Continuous Availability with no single point of failure
- Multi-Data Center and Zone support
- Flexible data model for unstructured, semi-structured and structured data
- Linear scalable performance with online expansion (scale-out and scale-up)
- Security with integrated authentication
- Operationally simple
- CQL - Cassandra Query Language





## Single Region, Multiple Availability Zone



# In-Class Exercise: Redis caching store

- Move the in-memory session map to a Redis store
  - heroku addons:create heroku-redis:hobby-dev
  - heroku plugins:install heroku-redis
  - npm install redis --save

```
var redis = requires('redis').createClient(process.env.REDIS_URL)

redis.hmset(sid, userObj)

redis.hgetall(sid, function(err, userObj) {
  console.log(sid + ' mapped to ' + userObj)
})
```

heroku config | grep REDIS

- heroku redis:cli --confirm <APPNAME>  
redis> keys \*  
redis> flushall

**Turnin auth.js with your redis session caching**  
**COMP431-S16:inclass-24**