CSC309 Winter 2016 Lecture 10

Larry Zhang

Announcements

- A3 showcase will be held in **DH-2010**, March 29, lecture time
- 6 minutes for each group's presentation, 20 groups
- Order of presentations will generated randomly and posted before the showcase, if you have to leave right after 5:00pm, let me know.
- Everyone will fill in a form to rate other groups' products, there will be small portion of participation marks for this.

Scalability and Performance

First of all

- Performance and Scalability are two different things
    - Performance is more about how fast **a single request** is handled
    - Scalability is about how the performance scales with the size of **user base growing**

Performance

Scalability

Outline

- **How to improve the performance of a single request**
  - Something that can make a difference for you now
  - Mostly about front-end
- **How to scale your web app**
  - Something to worry about when you have 1 million users.
  - More about back-end

Front-end performance tricks

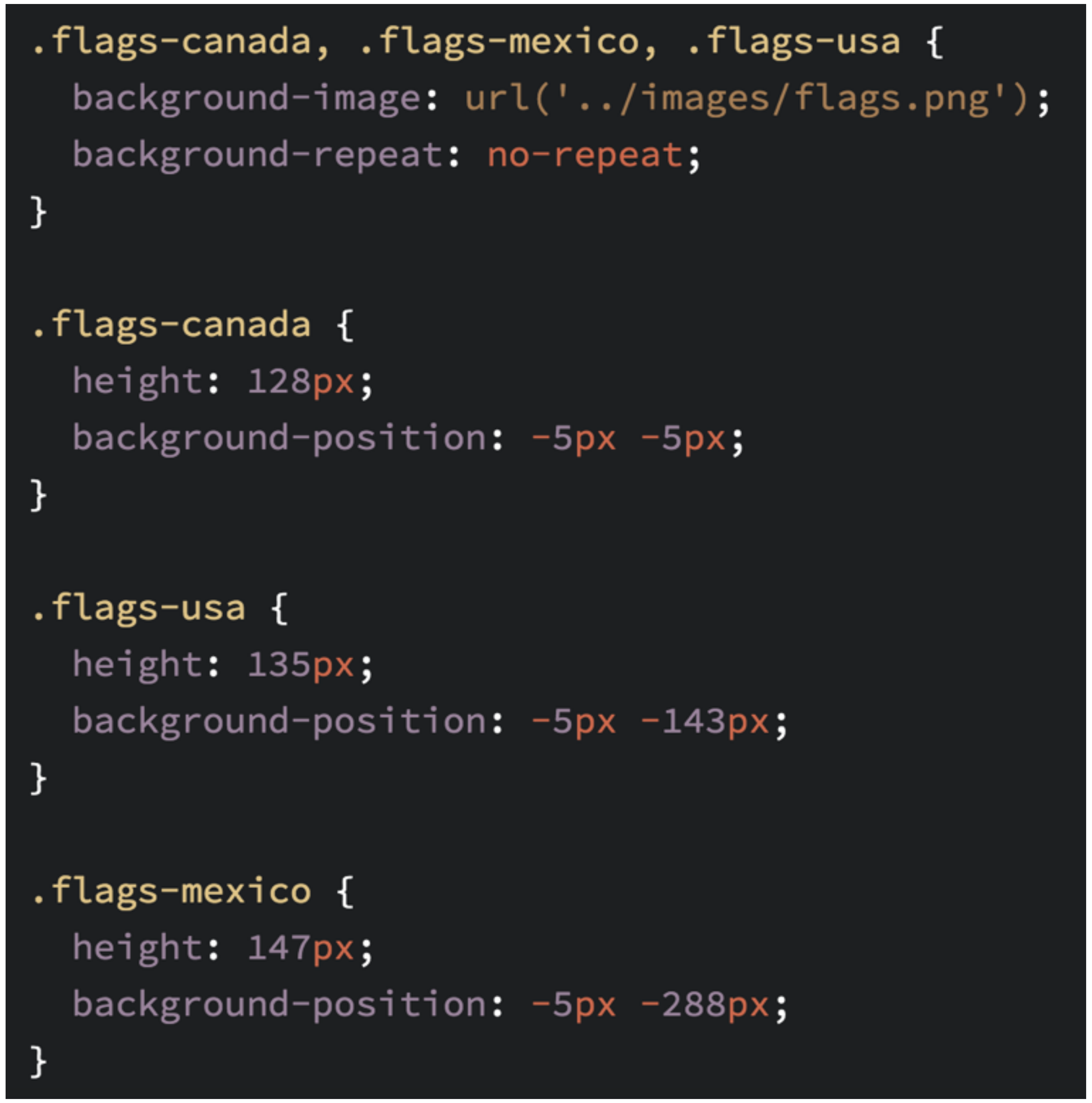
Stylesheets at the top, scripts at the bottom

- **CSS blocks rendering**
  - Nothing is displayed in your browser until CSS is loaded, so need to load CSS right away.
- **JavaScript blocks downloads**
  - Normally browsers try to download as many assets as possible *in parallel*; but JavaScript download blocks all other downloads for good reasons (avoid concurrency issues). So if nothing really need to be blocked, JavaScripts better be downloaded at very last.
- **Browsers are becoming smarter and smarter about how to download things.**

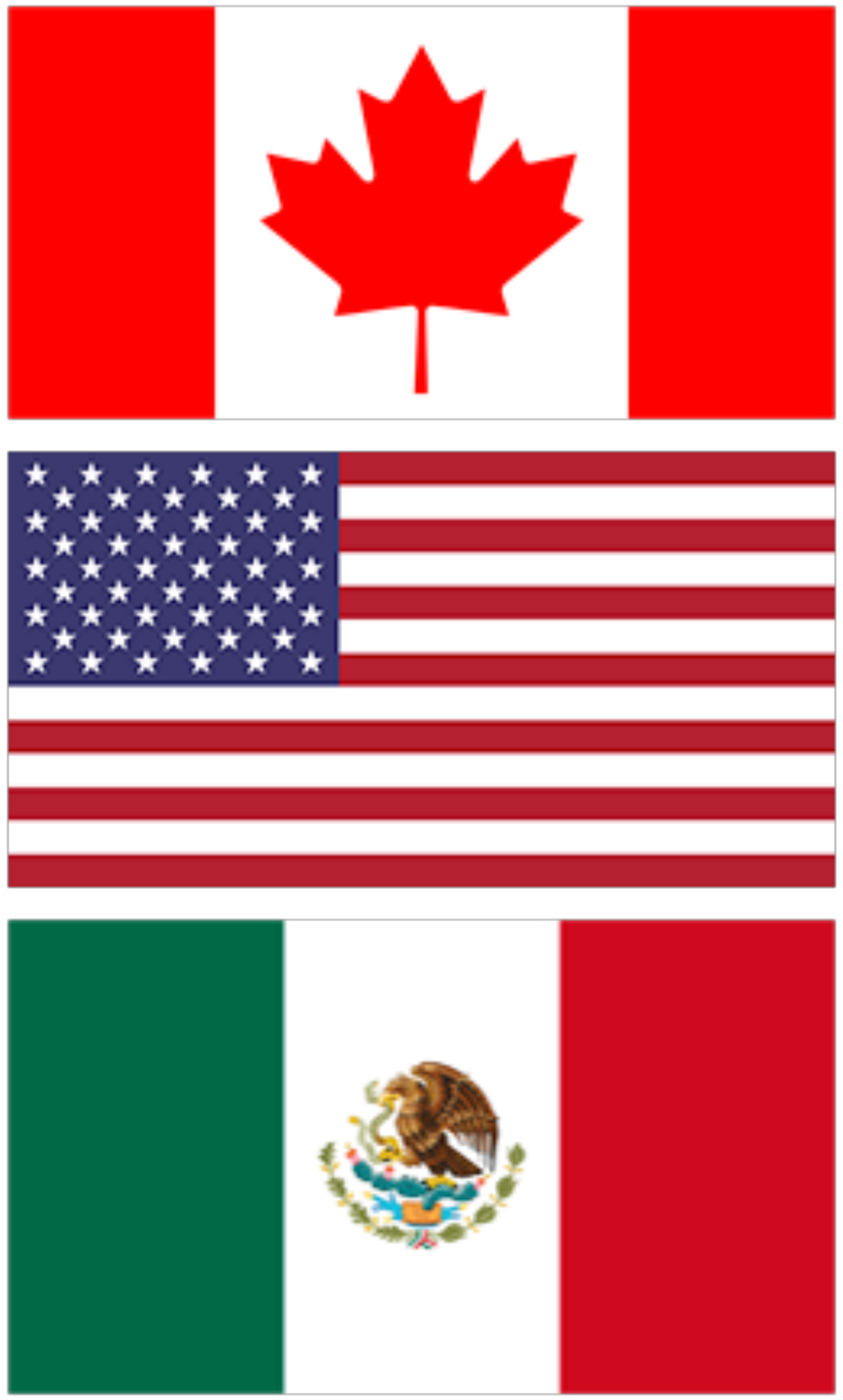
Make fewer requests

- Each HTTP request potential need to do DNS lookups, redirects, 404s, etc, which is costly.
- Make as smaller of HTTP request as possible.
- For example, if you have many small images in your web page, rather than issuing one HTTP request for each image, put all of them into **one big image (sprite)**, download in one single HTTP request, then use CSS positions to get each image. This is called **spriting**.
- https://css-tricks.com/css-sprites/

Spriting example

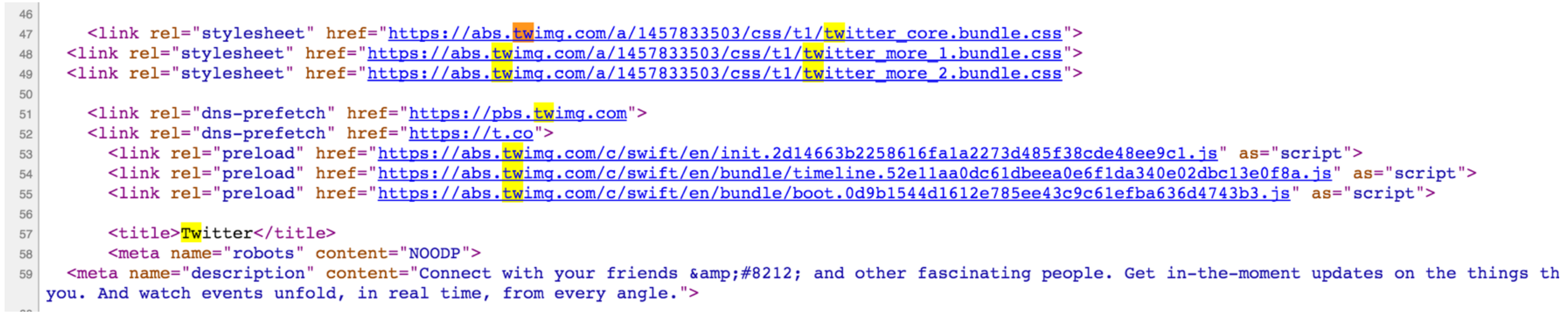


```css
.flags-canada, .flags-mexico, .flags-usa {
  background-image: url('../images/flags.png');
  background-repeat: no-repeat;
}

.flags-canada {
  height: 128px;
  background-position: -5px -5px;
}

.flags-usa {
  height: 135px;
  background-position: -5px -143px;
}

.flags-mexico {
  height: 147px;
  background-position: -5px -288px;
}
```

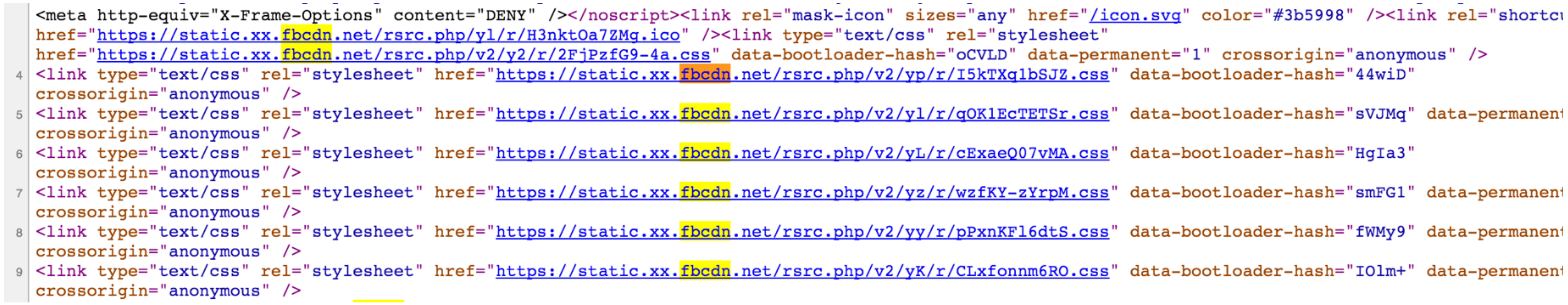


Maximizing Parallelization

- Browsers typical limits the number of parallel download from a domain, e.g., 2 connections.
- In order to get more concurrent download, have a separate domain for separate assets, or even several different domains.
    - e.g., Twitter has abs.twimg.com to server static assets.
    - e.g., Facebook has static.xx.fbcdn.net to server static assets



```
    <link rel="stylesheet" href="https://abs.twimg.com/a/1457833503/css/t1/twitter_core.bundle.css">
  <link rel="stylesheet" href="https://abs.twimg.com/a/1457833503/css/t1/twitter_more_1.bundle.css">
  <link rel="stylesheet" href="https://abs.twimg.com/a/1457833503/css/t1/twitter_more_2.bundle.css">

    <link rel="dns-prefetch" href="https://pbs.twimg.com">
    <link rel="dns-prefetch" href="https://t.co">
      <link rel="preload" href="https://abs.twimg.com/c/swift/en/init.2d14663b2258616fa1a2273d485f38cde48ee9c1.js" as="script">
      <link rel="preload" href="https://abs.twimg.com/c/swift/en/bundle/timeline.52e11aa0dc61dbeea0e6f1da340e02dbc13e0f8a.js" as="script">
      <link rel="preload" href="https://abs.twimg.com/c/swift/en/bundle/boot.0d9b1544d1612e785ee43c9c61efba636d4743b3.js" as="script">

      <title>Twitter</title>
      <meta name="robots" content="NOODP">
  <meta name="description" content="Connect with your friends &amp;#8212; and other fascinating people. Get in-the-moment updates on the things th
you. And watch events unfold, in real time, from every angle.">
```



```
<meta http-equiv="X-Frame-Options" content="DENY" /></noscript><link rel="mask-icon" sizes="any" href="/icon.svg" color="#3b5998" /><link rel="shortcu
href="https://static.xx.fbcdn.net/rsrc.php/yl/r/H3nktOa7ZMg.ico" /><link type="text/css" rel="stylesheet"
href="https://static.xx.fbcdn.net/rsrc.php/v2/y2/r/2FjPzfG9-4a.css" data-bootloader-hash="oCVLD" data-permanent="1" crossorigin="anonymous" />
<link type="text/css" rel="stylesheet" href="https://static.xx.fbcdn.net/rsrc.php/v2/yp/r/I5kTXq1bSJZ.css" data-bootloader-hash="44wiD"
crossorigin="anonymous" />
<link type="text/css" rel="stylesheet" href="https://static.xx.fbcdn.net/rsrc.php/v2/yl/r/qOK1EcTETSr.css" data-bootloader-hash="sVJMq" data-permanen
crossorigin="anonymous" />
<link type="text/css" rel="stylesheet" href="https://static.xx.fbcdn.net/rsrc.php/v2/yL/r/cExaeQ07vMA.css" data-bootloader-hash="HgIa3"
crossorigin="anonymous" />
<link type="text/css" rel="stylesheet" href="https://static.xx.fbcdn.net/rsrc.php/v2/yz/r/wzfKY-zYrpM.css" data-bootloader-hash="smFG1" data-permanen
crossorigin="anonymous" />
<link type="text/css" rel="stylesheet" href="https://static.xx.fbcdn.net/rsrc.php/v2/yy/r/pPxnKFl6dtS.css" data-bootloader-hash="fWMy9" data-permanen
crossorigin="anonymous" />
<link type="text/css" rel="stylesheet" href="https://static.xx.fbcdn.net/rsrc.php/v2/yK/r/CLxfonnm6RO.css" data-bootloader-hash="IOlm+" data-permanen
crossorigin="anonymous" />
```

Limitation of separate domains: DNS lookups

- Each new domain that your request (assuming nothing in cache) will involve a DNS lookup, which very time consuming.
- Especially so if this happens inside <script>, since downloading script blocks other parallel download.
- So there is a tradeoff about choosing the right number of separate domains.
- Rule of thumb: if you have ~20 assets, use one subdomain to serve them; if you have ~50 assets, use two subdomains to serve.
- Note: DNS lookup only happens when the **first time** a domain is accessed, then the lookup result will be cached.

DNS Prefetching

```
<head>
    ...
    <link rel="dns-prefetch" href="http://static.fbcdn.com">
    ...
</head>
```

- Do the DNS lookup before your the scripts actually needs the assets.
- Start early, save time.

You can prefetch other things, too

```
<link rel="prefetch" href=“sprite.png”>
<link rel="prefetch" href=“style.css">
```

- If you don’t do prefetch, by default the browser won’t fetch it until it is **really** needed.

Be careful with CSS

- CSS is a major enemy of performance, because it blocks rendering, i.e., CSS is on the critical path of page rendering.
- Nothing is shown on your page until all CSS files are fetched.
- Better get those CSS file as fast as you can
  - Don’t put it on a separate subdomain for static assets, to avoid extra DNS lookup
  - Serve it early
  - Concatenate multiple CSS files into one file, to reduce the number of HTTP requests
  - Gzip and minify it (will talk about it soon)
  - Cache the hell out of it

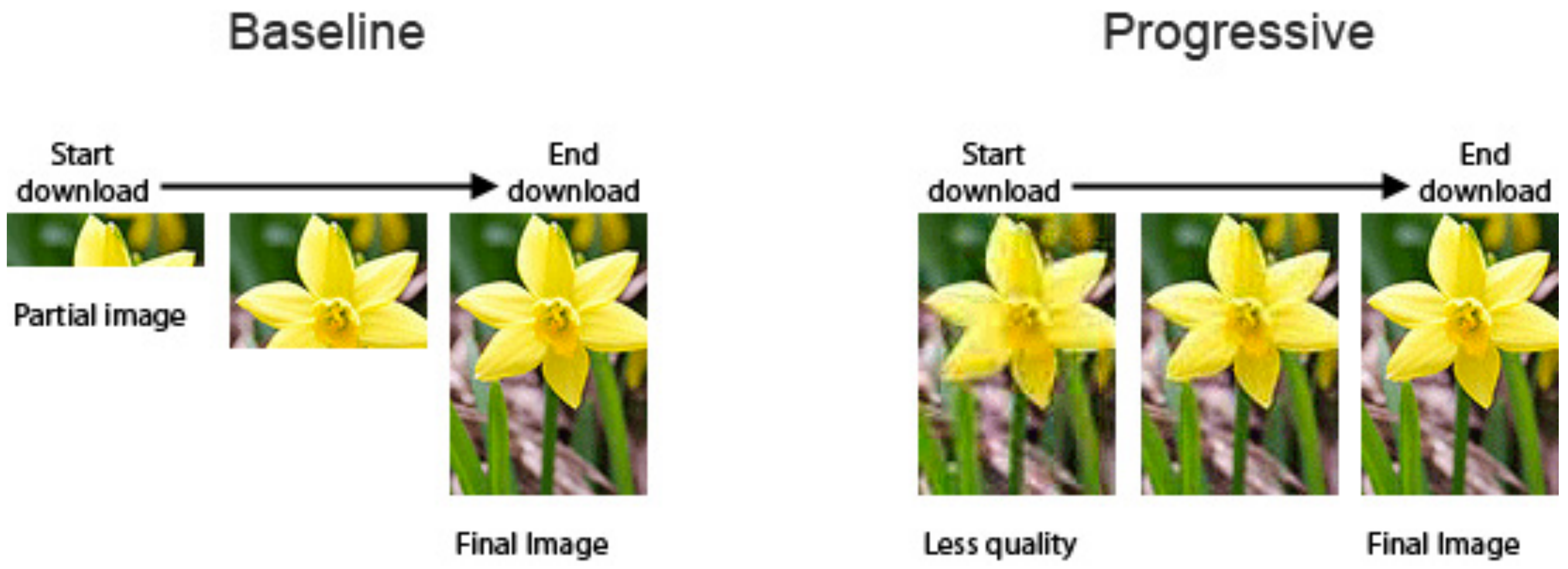
Gzipping and Minifying

- **Minifying**:
  - remove comments and white space in your text assets (like JavaScript files and CSS files);
  - use as-short-as-possible variables names and function names;
  - combine declarations (e.g., rather than **h1 {color:red;} p {color:red}**, just do **h1, p {color:red;}**)
  - https://developers.google.com/speed/docs/insights/MinifyResources#overview
- **Gzipping**: compress it for transmission
  - can give us more than 90% saving, big improvement!

Optimizing images

- Spriting: load one large image over one HTTP request instead of loading several smaller images over several HTTP requests
- Only keep necessary image resolution / sizes, resize the image to how big it is needed; be careful with Retina images (4 times the normal resolution)

Progressive JPG



To enable progressive JPG, check the corresponding options when saving an image in Photoshop (or other image tools).

Overall tip for image optimization

- Avoid using image whenever possible: If it can be done using CSS only, don’t use images.

Summary of Front-End Performance

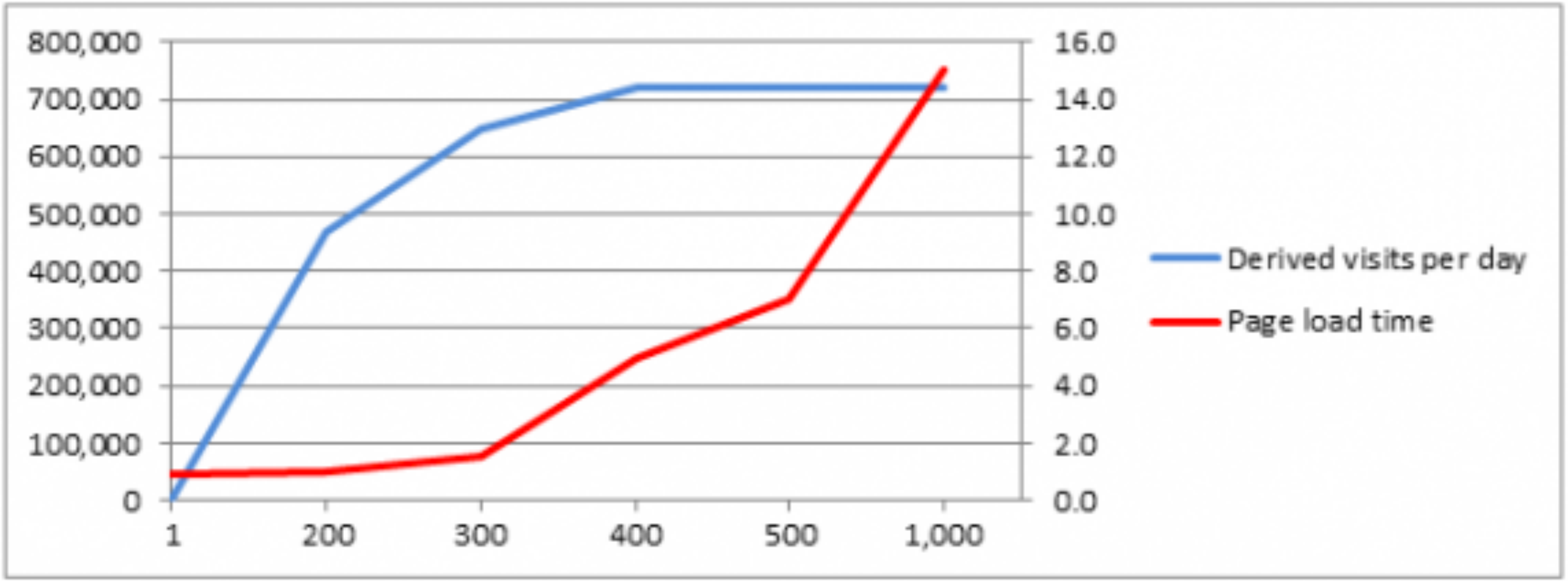
- What matters is the ***perceived*** performance of your website. It is not about the numbers, it is about how fast you website ***feels***.
- Always keep user experience in mind.

Credits: http://csswizardry.com/2013/01/front-end-performance-for-web-designers-and-front-end-developers/

Scalability

how many users we can support without downgrading each user's response time

Application performance drops with more users



Response time (page load speed) vs. concurrent users (& daily visitor equivalent)

Solutions

- First step is to understand the issue, find the bottleneck
- benchmark / load test your system
  - Apache bench: http://httpd.apache.org/docs/2.2/programs/ab.html
  - JMeter: https://jmeter.apache.org/
- Monitor different components
  - CPU, MEMORY, DISK, NETWORK
  - Monitoring tools: http://aarvik.dk/four-linux-server-monitoring-and-management-tools/

Load testing

Purpose

- **Predict** how your application will perform when a large number of users are using your application at the same time

How?

- Simulate **artificial** but **realistic** workloads

Load testing tools

- JMeter: free and written in Java
- Tsung: free and written in Erlang
- Locust.io: free and written in Python

Locust.io example

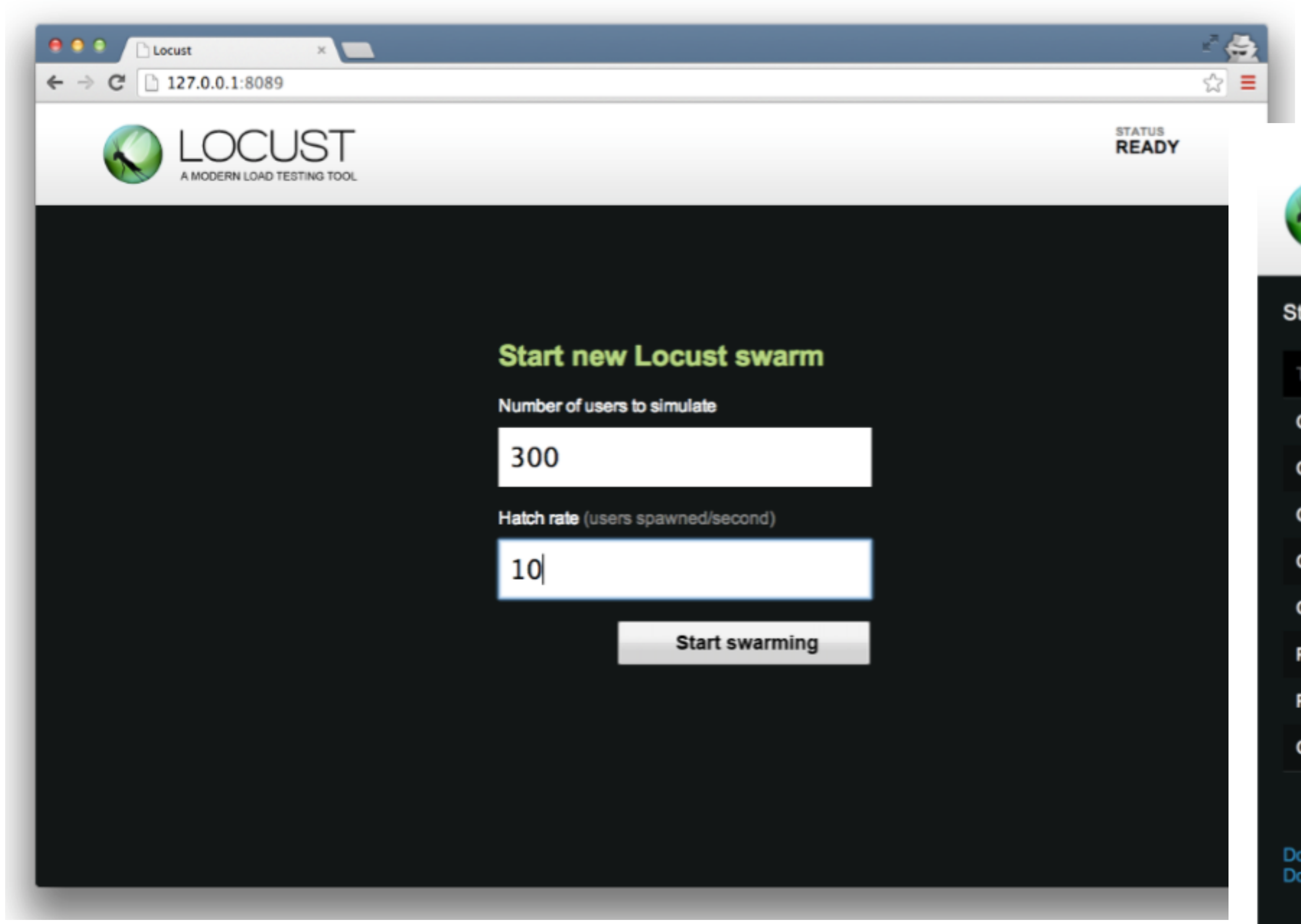
```
class User(HttpLocust):
     task_set = UserTasks
     min_wait = 5000
     max_wait = 15000
```

Explanation:
A user's behaviour will be defined in **UserTasks** class. The user will wait randomly between **5 to 15 secs** before sending a request to the application.

```
class UserTasks(TaskSet):
      @task(2)
      def index(self):

      self.client.get(“/”)

      @task(1)
      def about(self):
      self.client.get(“/
about/”)
```

Explanation:
A user will randomly send a GET request to “/” endpoint and a GET request to “/about/” endpoint. The user will send GET requests to “/” about twice as many times as GET requests to “/about/”.

Locust.io Web Interface



Locust

127.0.0.1:8089

LOCUST
A MODERN LOAD TESTING TOOL

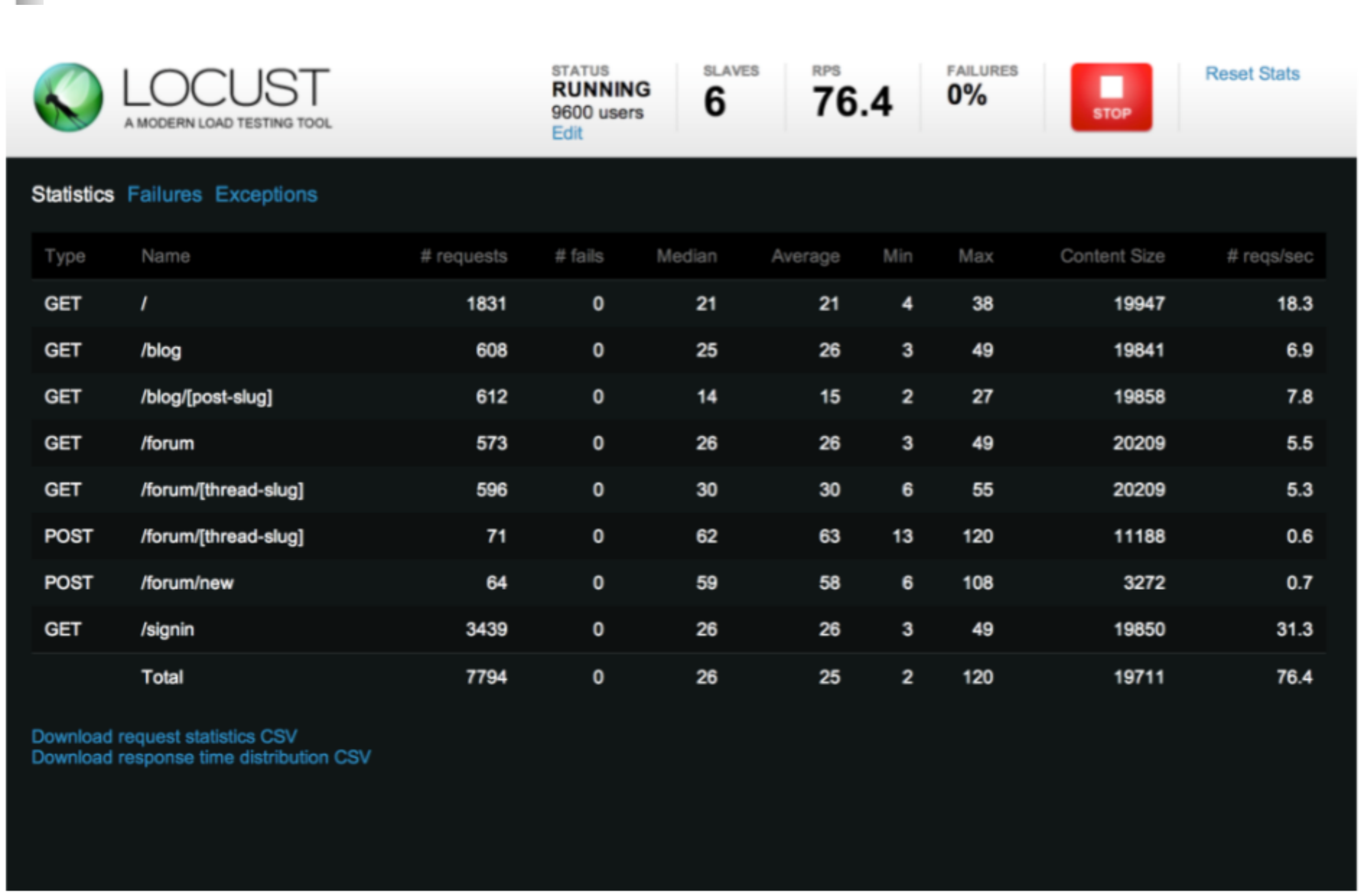
STATUS
READY

Start new Locust swarm

Number of users to simulate: 300

Hatch rate (users spawned/second): 10

Start swarming



LOCUST
A MODERN LOAD TESTING TOOL

STATUS: RUNNING, 9600 users, Edit
SLAVES: 6
RPS: 76.4
FAILURES: 0%
STOP
Reset Stats

Statistics Failures Exceptions

Type	Name	# requests	# fails	Median	Average	Min	Max	Content Size	# reqs/sec
GET	/	1831	0	21	21	4	38	19947	18.3
GET	/blog	608	0	25	26	3	49	19841	6.9
GET	/blog/[post-slug]	612	0	14	15	2	27	19858	7.8
GET	/forum	573	0	26	26	3	49	20209	5.5
GET	/forum/[thread-slug]	596	0	30	30	6	55	20209	5.3
POST	/forum/[thread-slug]	71	0	62	63	13	120	11188	0.6
POST	/forum/new	64	0	59	58	6	108	3272	0.7
GET	/signin	3439	0	26	26	3	49	19850	31.3
	Total	7794	0	26	25	2	120	19711	76.4

Download request statistics CSV
Download response time distribution CSV

Solution #1: Improve Hardware

- Get more RAM
- Use RAID
- Use SSD
- Spend $

This solution is OK …

- If you have tons of money to spend
- Buying a bigger box is quicker (ish) than redesigning software
- Running out of PostgreSQL performance?
  - spend months redesigning distributed database
  - or just buy a ton of RAM

Solution #2: Application Optimization

- All the front-end optimization that we have mentioned
- At the back-end minimize number requests to database and file system
- Optimize algorithms, better data structures
- Consider time/space trade-offs, pre-compute and store useful information to speed up response time

Solution #3: Apache Optimization

- Optimize PHP
  - Use Native PHP functions, use single quotes, use ===, choose proper string functions, pass by reference, disable debugging messages, use caching, use JSON, etc.
  - http://www.thegeekstuff.com/2014/04/optimize-php-code/

Solution #3: Apache Optimization

- Caching
  - mod_mem_cache, caching data in memory to speed up. (http://httpd.apache.org/docs/2.2/mod/mod_cache.html)
  - squid caching proxy: http://www.squid-cache.org/
  - varnish cache: https://www.varnish-cache.org/about

Solution #3: Apache Optimization

- Load balancing: When you have multiple server instances running, keep their load balanced
- mod_proxy_balancer: Apache’s load balancing module
    - http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html

Solution #4: Use other web servers than Apache

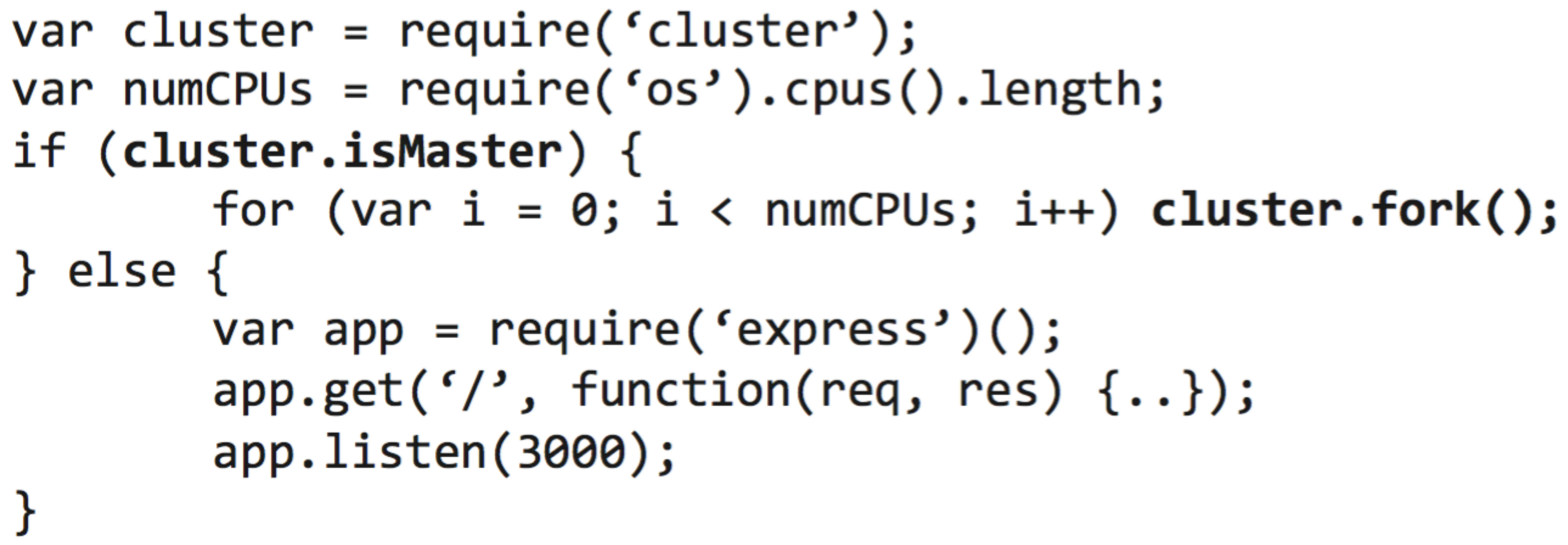
- Node.js: generally have better performance than Apache
- Nginx: https://www.nginx.com/resources/wiki/#

More about Node.js

- As we mentioned before, Node.js is single-threaded, i.e., each Node.js server only uses one core
- On a multi-core machine, we want to run one Node.js process per core.
- How to effectively manage the multiple Node.js processes?

Cluster Module of Node.js

- Make it easier to handle mange multiple Node.js servers.
- https://nodejs.org/api/cluster.html



```
var cluster = require(‘cluster’);
var numCPUs = require(‘os’).cpus().length;
if (cluster.isMaster) {
        for (var i = 0; i < numCPUs; i++) cluster.fork();
} else {
        var app = require(‘express’)();
        app.get(‘/’, function(req, res) {..});
        app.listen(3000);
}
```

Caching in Node.js with **Redis**

- http://redis.io/
- An in-memory data structure store, for caching

```
var redis = require('redis');
var client = redis.createClient();
app.get('/expensive', function(req, res) {
          client.get('result', function(err, val) {
                    if (val) return res.send(val);
                    else {
                              var result = 0;
                              for(var i=0; i<10000; i++) {..}
                              client.set('result', result);
                    }
          });
});
```

Solution #4: Database optimizations

- Create indexes
- optimize queries
- denormalize data
- Memcached: http://memcached.org/
- NoSQL DBs: MongoDB, Apache Cassandra, etc.
    - Sharding and replication work together for distributed database to work.

Overall, the goal of scalability

- The system is organize in such that, if there is more work to handle, there will be more workers.
- Ideally, we have linear scalability, i.e., n times the workers gives n times the performance.
- This ideally would only happen if the work can be fully parallelized.
- But in reality some tasks are sequential rather than parallelizable

Sequential vs Parallelizable tasks

- 1 baker takes 1 hour to bake 1 cake.
- 10 bakers take 1 hour to bake 10 cakes.
- 10 bakers take 1/10 hour to bake 1 cake?

Platforms for scalable computing

- where they make tasks as paralleizable as possible
- Google MapReduce
- Hadoop
- Apache Spark
- Cloud computing infrastructure
  - Amazon AWS
  - Google Compute Engine

Other issues

- Security is a big thing, you’ll learn it or have learned it from other courses.

Going Live

Now you're capable of making something that people can use.
What are the next steps before becoming the next Facebook?

Hosting

- Get a domain name
  - Namecheap, Dreamhost, Godaddy, etc.
- Host it on some cloud computing platform
  - AWS, Google Compute Engine, Rackspace, etc.

Refine your service

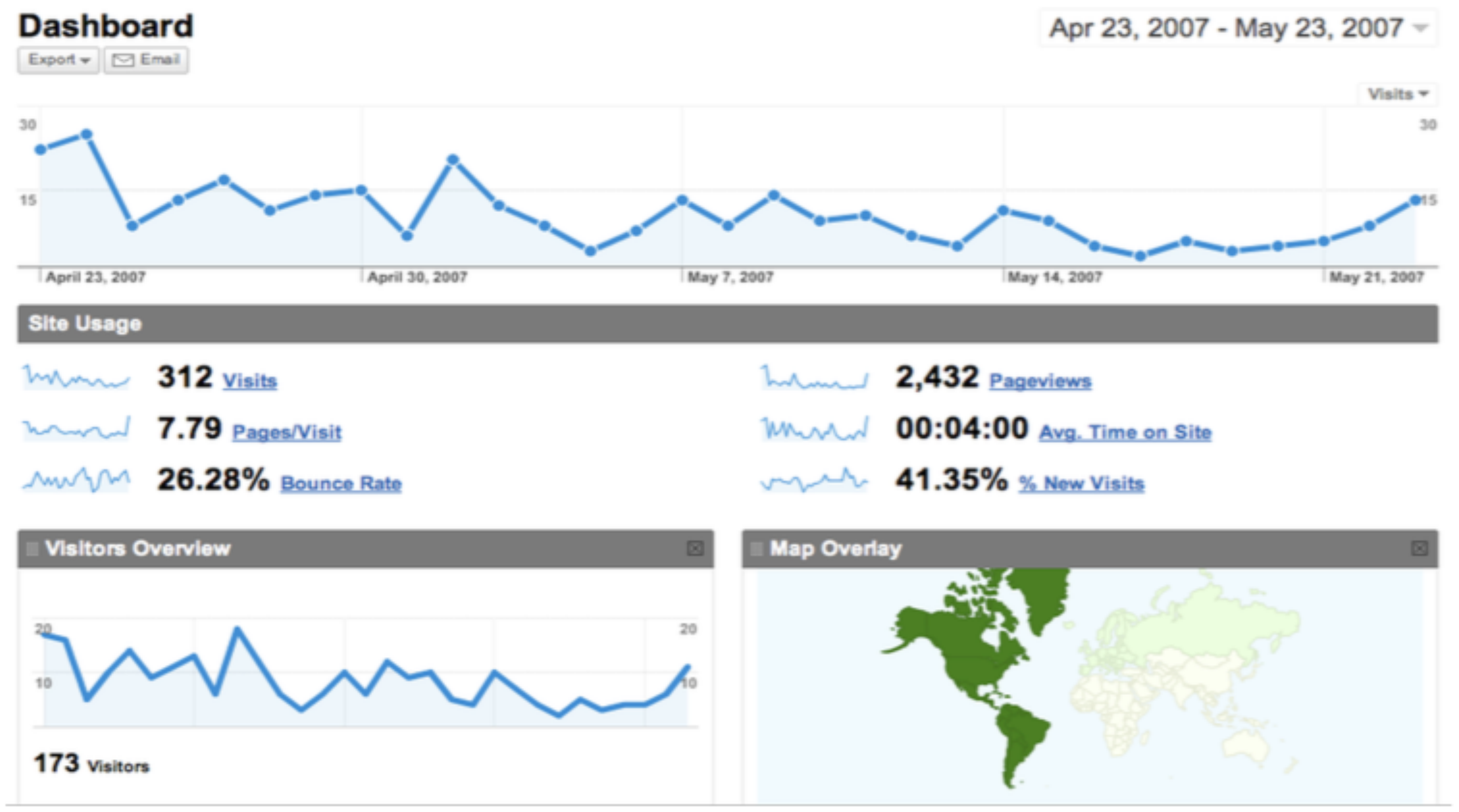
- Understand user (by talking to them for example)
- Improve your apps usability
- Look and feel
- Core features
- API framework
- Tough decisions to be made

Privacy

- Be upfront with the user
- Specify privacy policy in a “Term” page
    - nobody reads it, but it is still necessary

Track your website

- Use Google Analytics
- Gives refined insights about your website's traffic



312 Visits
2,432 Pageviews
7.79 Pages/Visit
00:04:00 Avg. Time on Site
26.28% Bounce Rate
41.35% % New Visits
173 Visitors

Revenue model

- Make some money with advertisement (maybe)
- Google AdSense
    - Automatically displayed relevant ads on your site
    - Paid per click, ~$0.10~$3.00
- Many e-commerce sites offers commission (5-10%) for referral
    - Amazon, Bestbuy, etc.

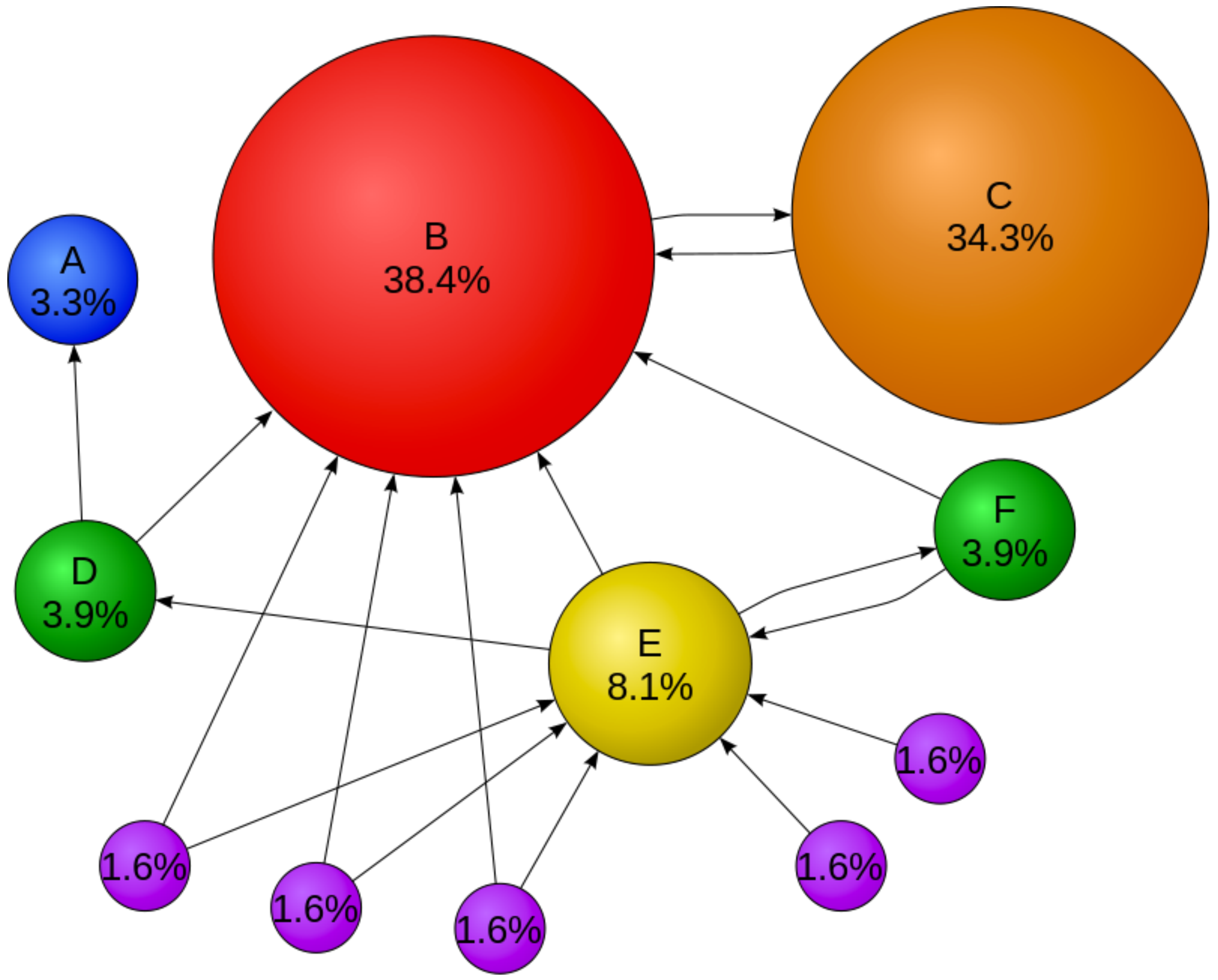
Advertise your website

- Use Google AdWords
    - you pay Google for advertising your website

Search Engine Optimization (SEO)

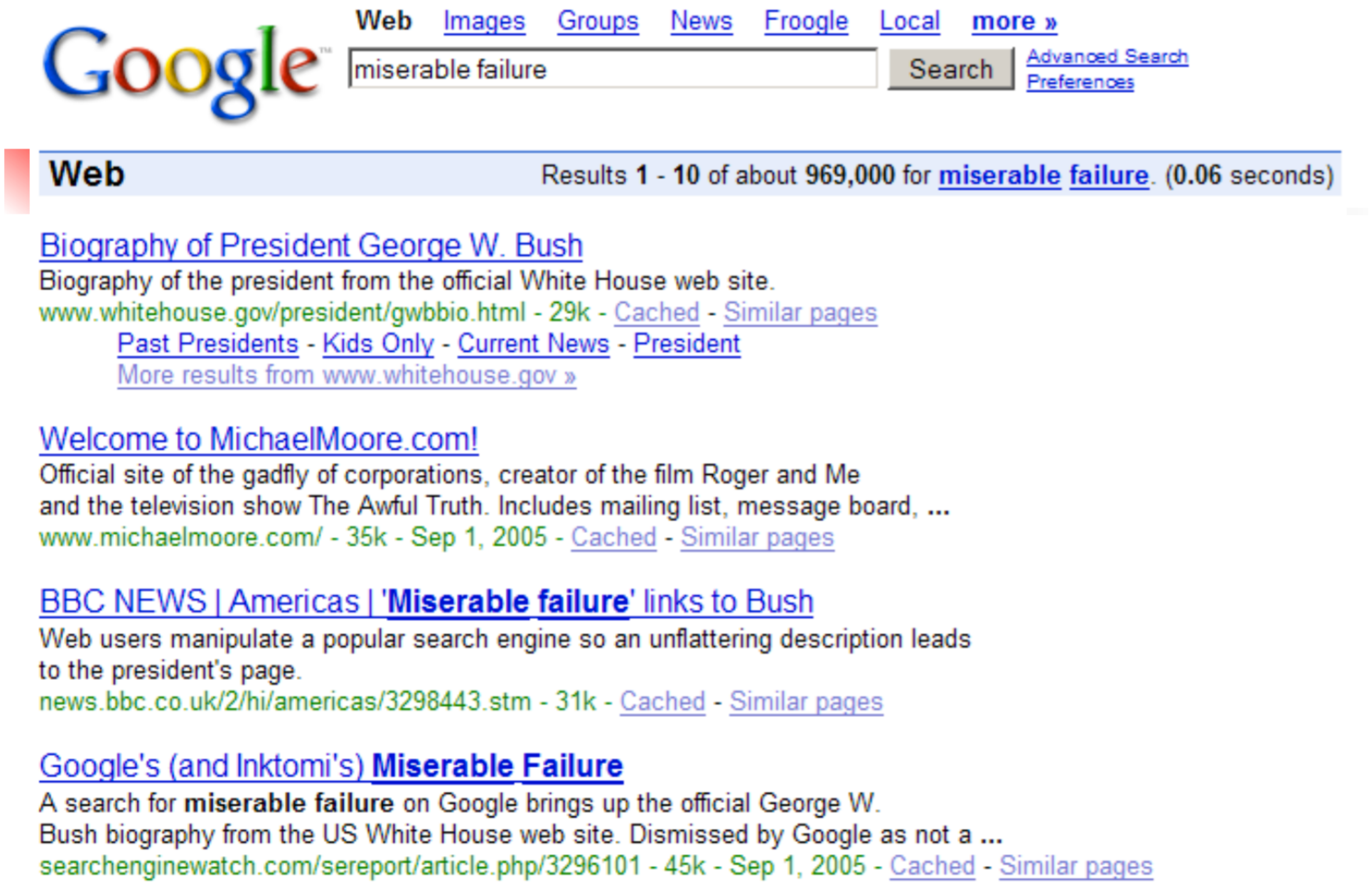
- Include proper <meta>
- Use a descriptive <title>
- Learn about how Google's ranking algorithm works
  - https://en.wikipedia.org/wiki/PageRank
  - https://en.wikipedia.org/wiki/Nofollow
  - https://en.wikipedia.org/wiki/Search_engine_optimization

PageRank



Google Bombs

- Before Google got smart, they keyworded mostly by anchor text
- Bloggers linked to whitehouse.gov/president with anchor text "miserable failure"
- Googling "miserable failure" showed whitehouse.gov/president as the first link
- Google has since fixed this
- http://en.wikipedia.org/wiki/Google_bomb



Web Results **1** - **10** of about **969,000** for **miserable failure**. (**0.06** seconds)

Biography of President George W. Bush
Biography of the president from the official White House web site.
www.whitehouse.gov/president/gwbbio.html - 29k - Cached - Similar pages
Past Presidents - Kids Only - Current News - President
More results from www.whitehouse.gov »

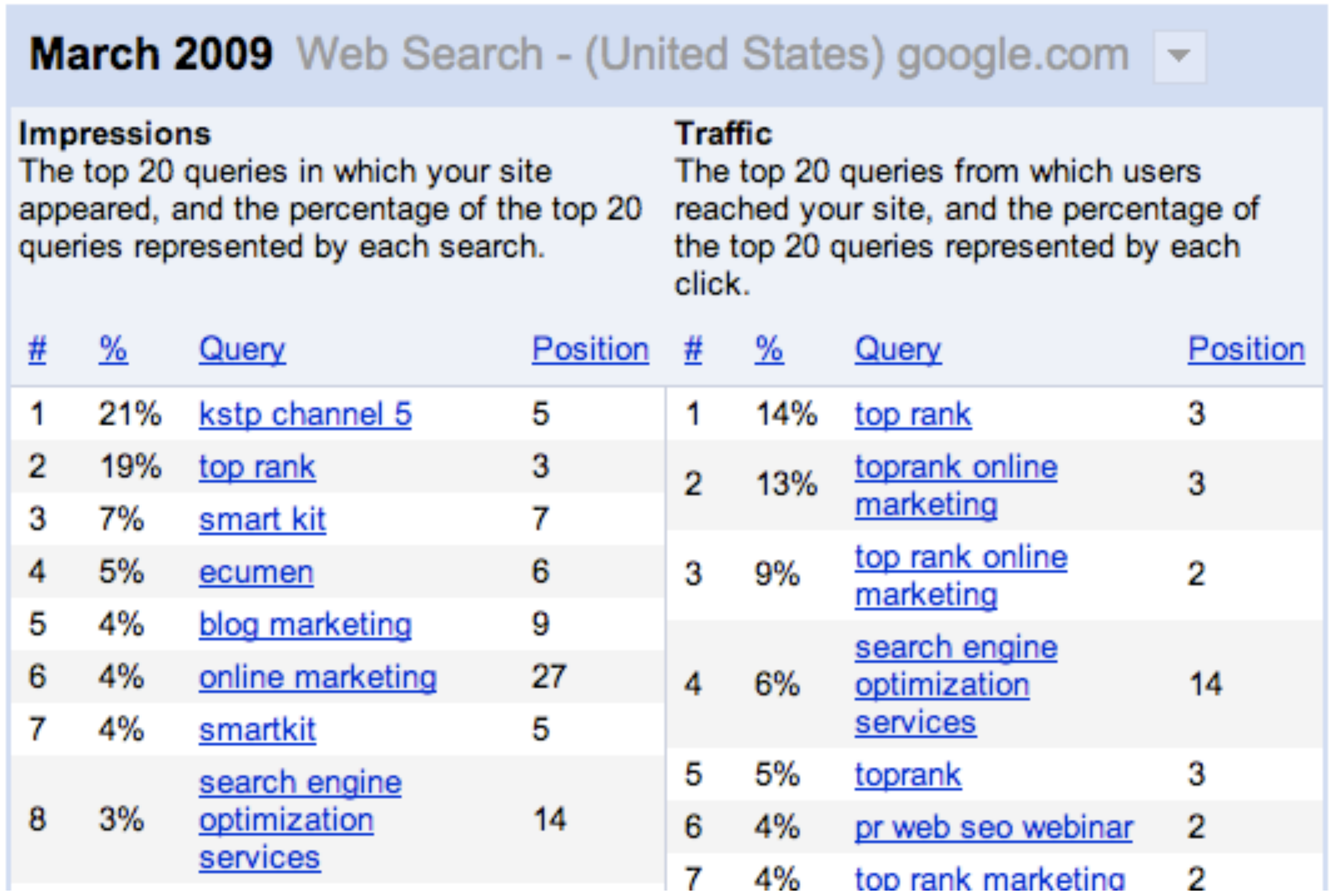
Welcome to MichaelMoore.com!
Official site of the gadfly of corporations, creator of the film Roger and Me and the television show The Awful Truth. Includes mailing list, message board, ...
www.michaelmoore.com/ - 35k - Sep 1, 2005 - Cached - Similar pages

BBC NEWS | Americas | '**Miserable failure**' links to Bush
Web users manipulate a popular search engine so an unflattering description leads to the president's page.
news.bbc.co.uk/2/hi/americas/3298443.stm - 31k - Cached - Similar pages

Google's (and Inktomi's) **Miserable Failure**
A search for **miserable failure** on Google brings up the official George W. Bush biography from the US White House web site. Dismissed by Google as not a ...
searchenginewatch.com/sereport/article.php/3296101 - 45k - Sep 1, 2005 - Cached - Similar pages

A useful SEO tool

- Google WebMaster Tools
  - https://www.google.com/webmasters/
  - Information about your site in Google's search engine



March 2009 Web Search - (United States) google.com

Impressions
The top 20 queries in which your site appeared, and the percentage of the top 20 queries represented by each search.

#	%	Query	Position
1	21%	kstp channel 5	5
2	19%	top rank	3
3	7%	smart kit	7
4	5%	ecumen	6
5	4%	blog marketing	9
6	4%	online marketing	27
7	4%	smartkit	5
8	3%	search engine optimization services	14

Traffic
The top 20 queries from which users reached your site, and the percentage of the top 20 queries represented by each click.

#	%	Query	Position
1	14%	top rank	3
2	13%	toprank online marketing	3
3	9%	top rank online marketing	2
4	6%	search engine optimization services	14
5	5%	toprank	3
6	4%	pr web seo webinar	2
7	4%	top rank marketing	2

Legal stuff

- Intellectual Property
- Copyright
- Trademark
- Patent
- Protect yourself

then, be lucky.

Next week

- Final Exam Preview