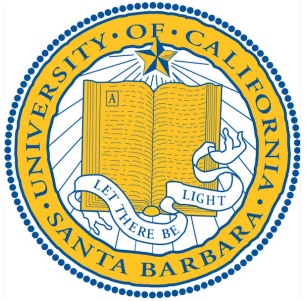
CS 290B

Scalable Internet Services

Andrew Mutz

October 21, 2014



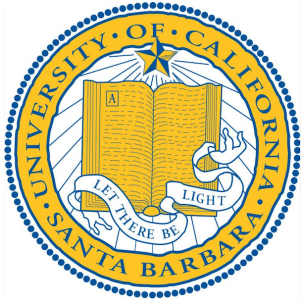
Today's Agenda

Motivation

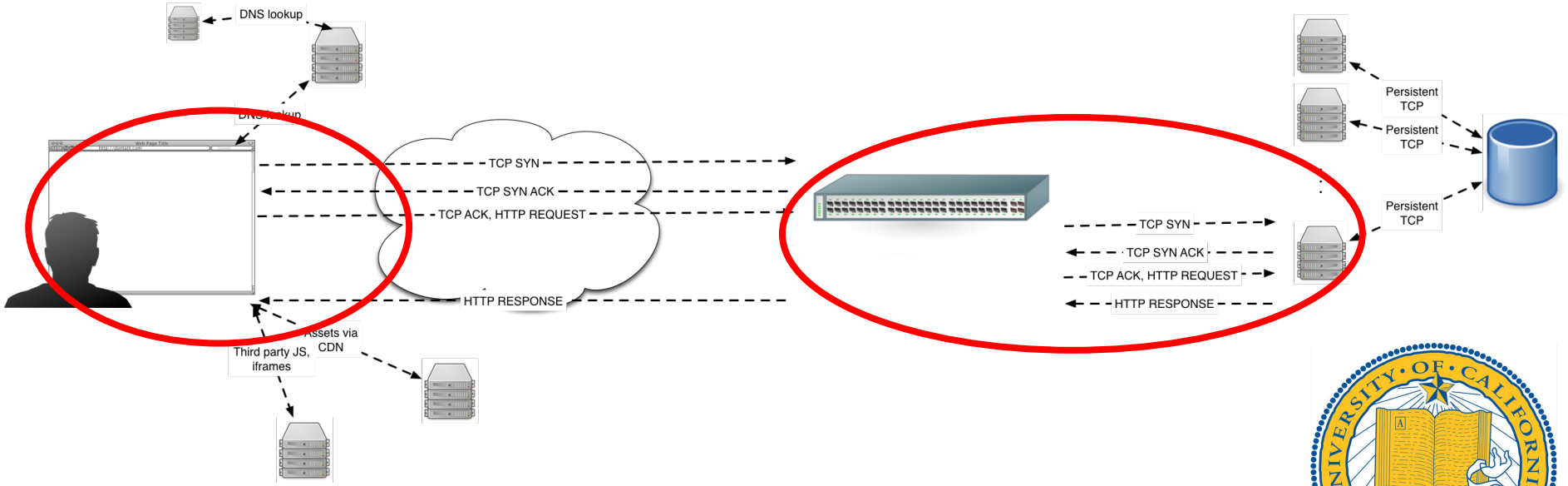
Client-side Caching

Server-side Caching

For Next Time



Motivation

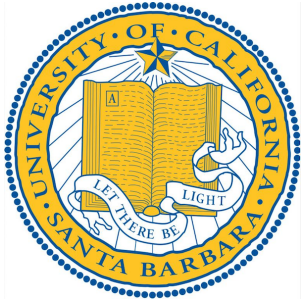

DNS lookup
TCP SYN
TCP SYN ACK
TCP ACK, HTTP REQUEST
HTTP RESPONSE
Assets via CDN
Third party JS, iframes
Persistent TCP
Persistent TCP
Persistent TCP
TCP SYN
TCP SYN ACK
TCP ACK, HTTP REQUEST
HTTP RESPONSE

Motivation

We want our important application data persisted safely in our data center.

And it needs to be regularly read and updated by geographically distributed clients.

And it needs to be fast.

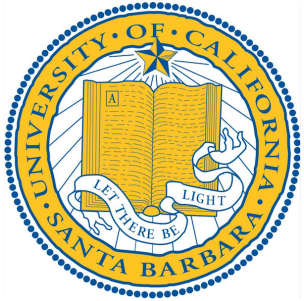


Motivation

Performance Matters!

Delay	User Reaction
0 - 100 ms	Instant
100 - 300 ms	Slight perceptible delay
300 - 1000 ms	Task focus, perceptible delay
1 second+	Mental context switch
10 seconds+	I'll come back later...

Source: Ilya Grigorik (igvita.com)

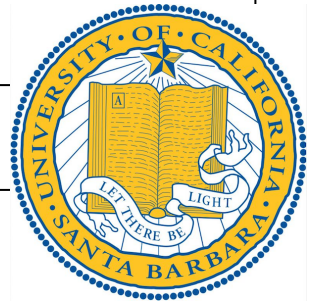


Motivation

But there are challenges:

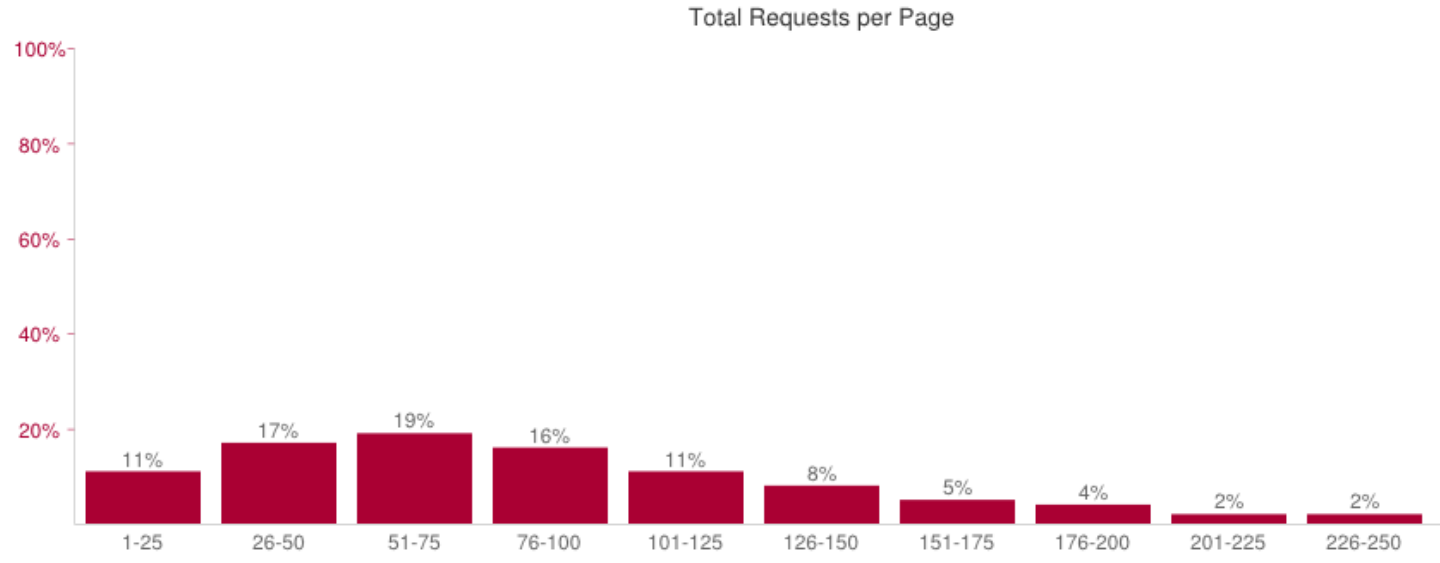
Route	Distance	Time, light in vacuum	Time, light in fiber
NYC to SF	4,148 km	14 ms	21 ms
NYC to London	5,585 km	19 ms	28 ms
NYC to Sydney	15,993 km	53 ms	80 ms
Equator	40,075 km	133 ms	200 ms

Source: High Performance Browser Networking, Ilya Grigorik

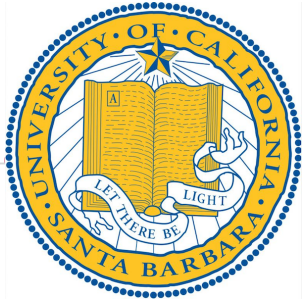


Motivation

A page is more than a single request:



Source: http://httparchive.org/



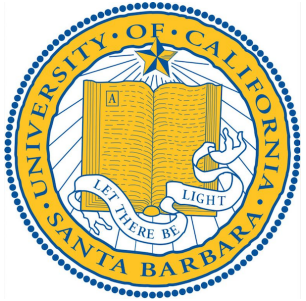
Motivation

The fastest request is the one that never happens!

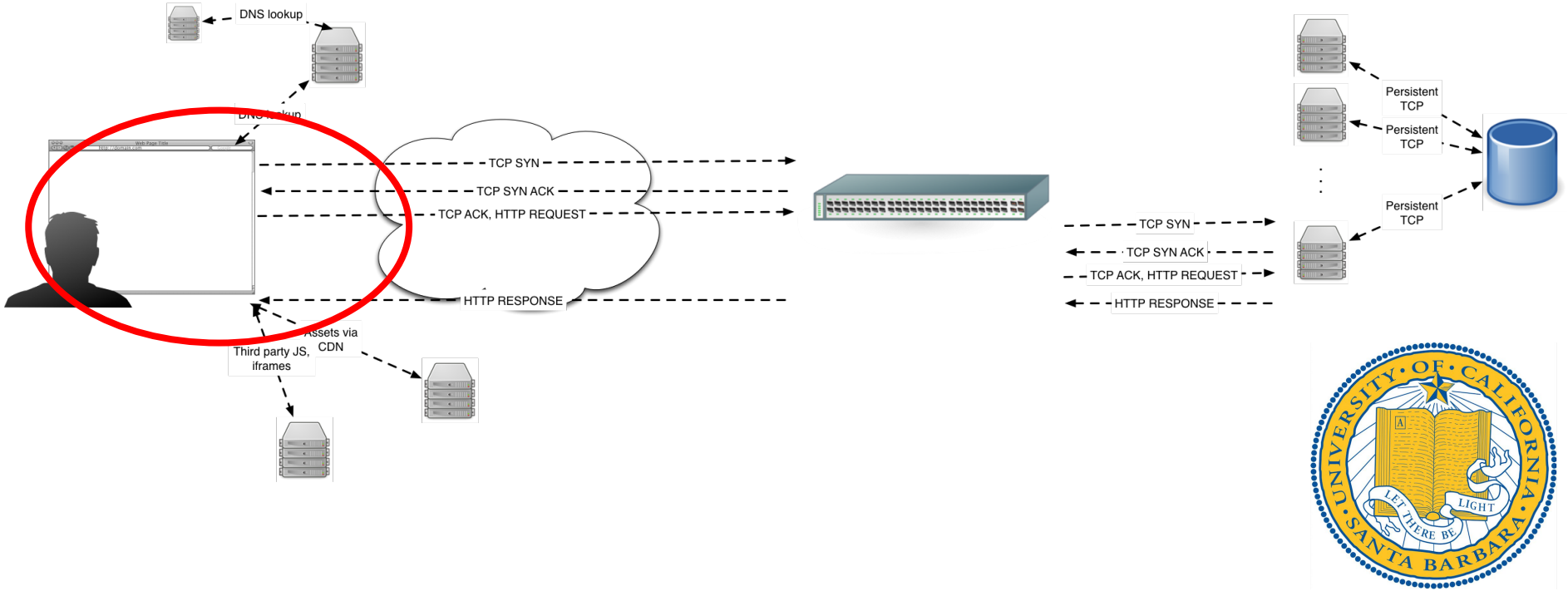
Cache: a component that transparently stores data so that future requests for that data can be served faster.

Where to introduce caching?

- Inside the browser
- In front of the server (CDNs, etc.)
- Inside the application server
- Inside the database (query cache)



Motivation

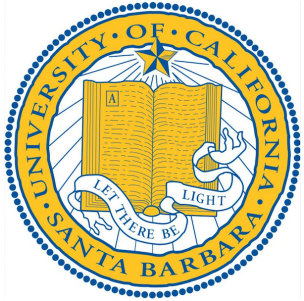

DNS lookup
TCP SYN
TCP SYN ACK
TCP ACK, HTTP REQUEST
HTTP RESPONSE
Assets via CDN
Third party JS, iframes
Persistent TCP
Persistent TCP
Persistent TCP
TCP SYN
TCP SYN ACK
TCP ACK, HTTP REQUEST
HTTP RESPONSE

Client-side Caching

How does the browser cache data? How does it know when it can safely present previously seen data as current?

The building blocks are all HTTP headers:

- `etag`
- `cache-control`
  - `max-age`
  - `no-cache`
  - `no-store`
  - `public | private`
- `if-modified-since`
- `if-none-match`



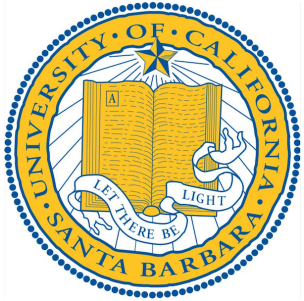
Client-side Caching

`etag: “5bf444d26f9f1c74”`

When accompanying a response, the browser will keep this “entity tag” along with saved copies of the resource.

When requesting the same resource in the future, this tag can be presented to indicate the version it had previously seen.

This isn’t necessarily a digest of the resource that was served up, but can be thought of as such.

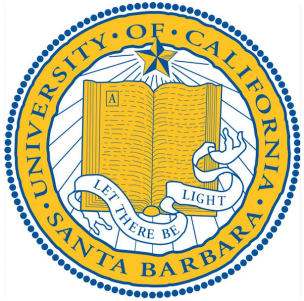


Client-side Caching

`cache-control: no-cache`

When accompanying a response, the browser (or intermediate proxy) is instructed to revalidate before reusing it.

Without this, the browser can use recently seen versions safely.

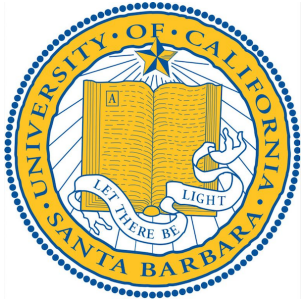


Client-side Caching

`cache-control: max-age=120`

When accompanying a response, the browser (or intermediate proxy) should consider this copy fresh until the specified number of seconds has passed.

The more modern version of the `expires` and `date` headers.

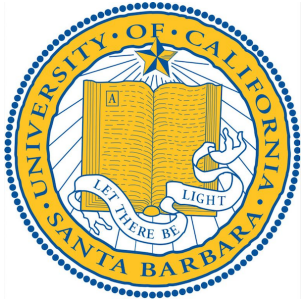


Client-side Caching

`cache-control: no-store`

When accompanying a response, the browser (or intermediate proxy) is instructed to not reuse this data under any circumstances.

This can also used for sensitive information.



Client-side Caching

`cache-control: private`

When accompanying a response, the browser (or intermediate proxy) is instructed that the data is specific to the requesting user.

Intermediate proxies should discard such data, but a single-user browser can reuse it.

The opposite of this is `cache-control: public`

Client-side Caching

`if-modified-since: Sun, 19 Oct 2014 19:43:31`

When accompanying a request, this indicates that the client already has a copy that was fresh as of the specified date.

If the server's copy is newer than the specified date, it will be served to the client.

If the server's copy hasn't changed since the specified date, the server will return `304 (not modified)`.

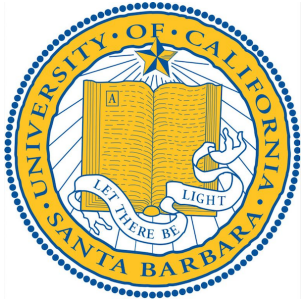
Client-side Caching

`if-none-match: “5bf444d26f9f1c74”`

When accompanying a request, this indicates that the client has a cached copy with the associated tag. Multiple etags can be provided.

If the server’s current version has one of the etags listed, the server will return `304 (not modified)` with the etag of the current resource included.

If the server’s version has a non-matching etag, then the result will be returned as normal.

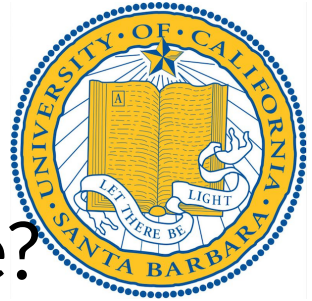


Client-side Caching

Let’s pull this together and apply what we’ve seen.

Let’s say we are serving up some javascript that won’t change over the next day, but does have some user-specific code in it.

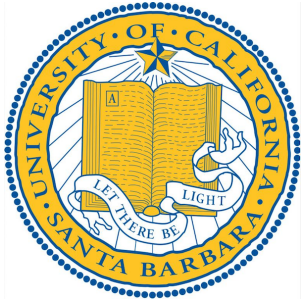
What headers should the response include?



Client-side Caching

We want it reusable, but private:

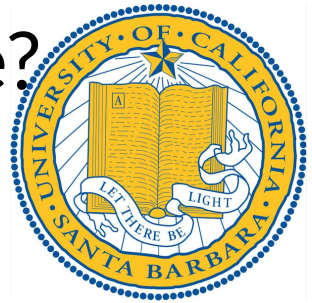
```
Cache-control: private, max-age=86400
```



Client-side Caching

Let's say we are serving up an image that may be changing in the future, and we never want a stale version shown. The image is not specific to the requestor.

What headers should the response include?



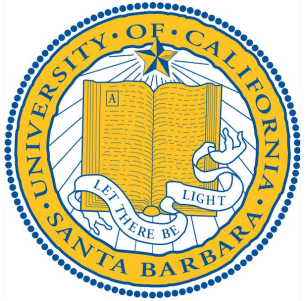
Client-side Caching

We want it reusable with revalidation and public:

```
Cache-control: public, no-cache
ETag: “4d7a6ca05b5df656”
```

Clients will request the resource with:

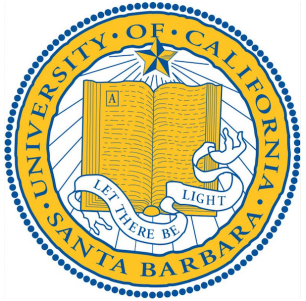
```
if-none-match: “4d7a6ca05b5df656”
```



Client-side Caching

Let’s say we are serving up an image with the user’s social security and credit card numbers.

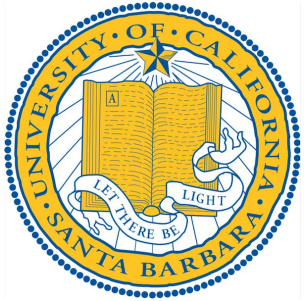
What headers should the response include?



Client-side Caching

We want it reusable, but private:

`Cache-control: private, no-store`



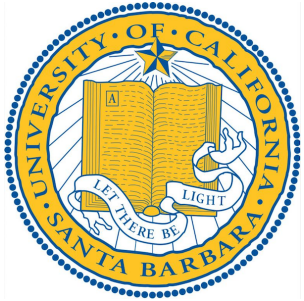
Today's Agenda

Motivation

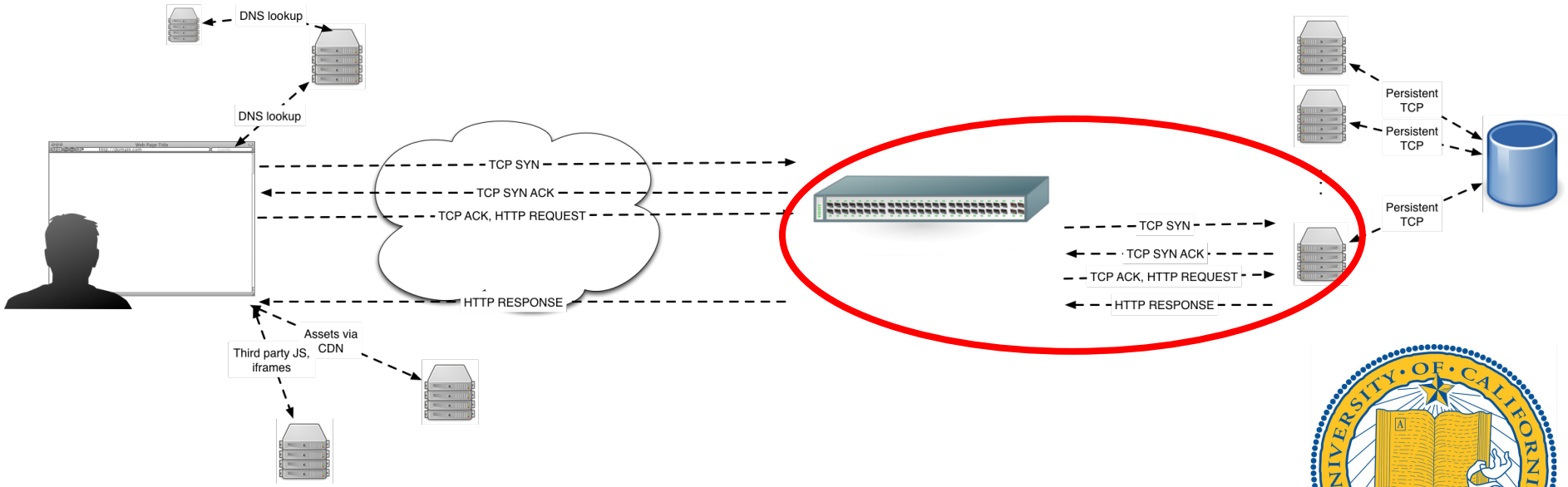
Client-side Caching

Server-side Caching

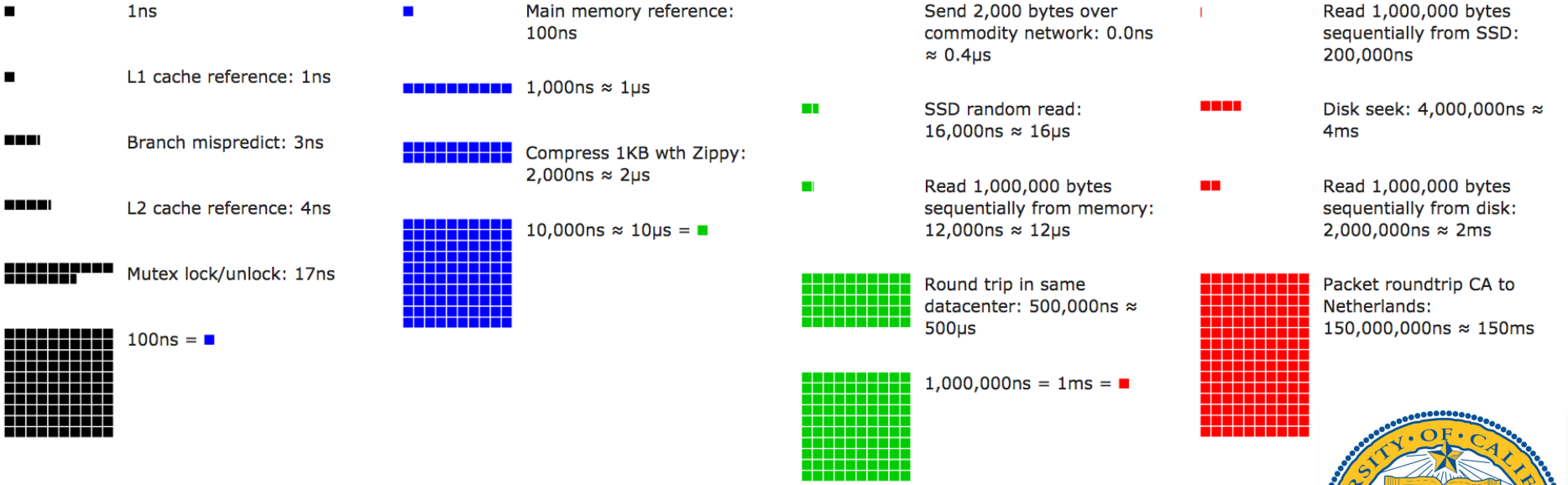
For Next Time



Server-side Caching



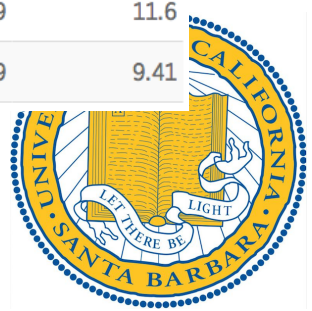
Server-side Caching



Source: http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Server-side Caching

Category	Segment	% Time	Avg calls (per txn)	Avg time (ms)
Database	SQL - SELECT	15.9	18.7	65.3
View	occupancies/_sidebar_show.html.erb Partial	11.8	0.989	48.5
Controller	OccupanciesController#show	9.4	1.0	38.8
View	occupancies/_occupancy_financials_show.html.erb Partial	5.4	0.989	22
View	occupancies/_occupancy_status_show.html.erb Partial	4.3	0.989	17.5
View	occupancies/show.html.erb Template	3.8	0.989	15.5
View	occupancies/_occupancy_monthly_charges_show.html.erb Partial	2.8	0.989	11.6
View	notes/_note_show.html.erb Partial	2.3	1.29	9.41



Server-side Caching

From this data we can see that the network inside our datacenter is fast.

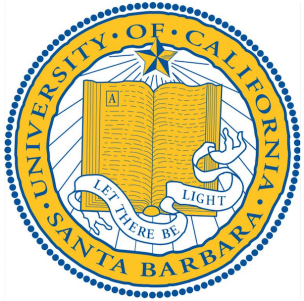
- Roundtrip inside the datacenter is about ~500ns
- Main memory access is ~100ns.

Reads from SSDs are faster, disk is slower

- 0.016ms to 0.2ms for SSD reads
- 4ms to 6ms for magnetic disk reads

Ruby/Rails is slow.

- Some partials are taking tens of milliseconds to render.



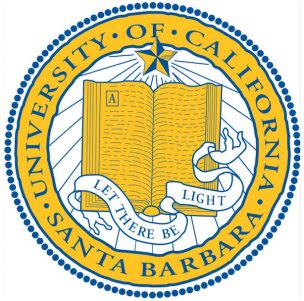
Server-side Caching

Basic math reveals opportunities for caching:

- If an entire page can be reused, storing it on disk and retrieving it when needed can bring big wins
  - 5ms* vs. hundreds of milliseconds

- If only part of a page can be reused, it still may make sense to cache it on disk
  - 5ms* vs. tens of milliseconds

- Instead of saving on disk, traversing the network and storing in memory can be faster*.
  - What are other advantages to non-local caching?



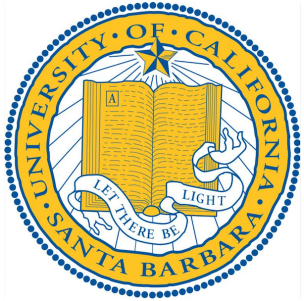
Rails Caching

Caching has changed significantly in Rails 4.

Rails previously had mechanisms for caching entire pages and actions. As of Rails 4, these have been moved to a separate library called `actionpack-page_caching`.

Rails now emphasizes three types of caching:

- HTTP caching
- Fragment caching
- Low level caching



Rails Caching

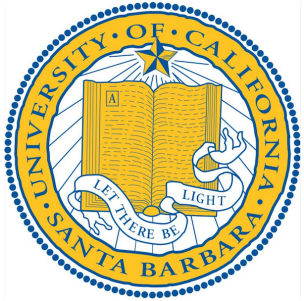
By default, caching is disabled in development and test, and enabled in production

- If you want to use it in development mode, add this to your environment:

```
config.action_controller.perform_caching = true
```

Rails can be configured to store cached data in a few different places:

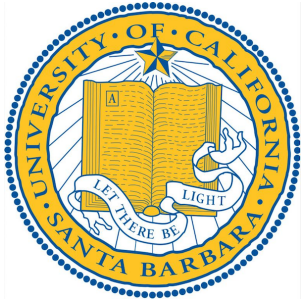
- In memory
- Local file system
- Remote in-memory store



Rails Caching

ActiveSupport::Cache::MemoryStore

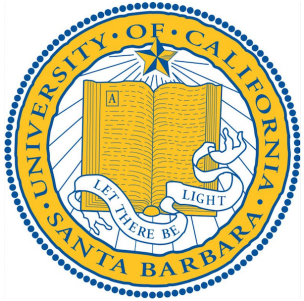
- Cached data is stored in memory, in the same address space as the ruby process and is retained between requests.
- Defaults to 32 megs, but is configurable.
- Strengths?
- Weaknesses?



Rails Caching

ActiveSupport::Cache::MemoryStore

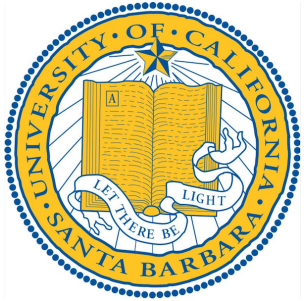
- Cached data is stored in memory, in the same address space as the ruby process and is retained between requests.
- Defaults to 32 megs, but is configurable.
- Strengths:
  - Local memory is fast
- Weaknesses:
  - We are likely running many Rails processes on a single machine, and these processes can’t use each others memory cache.



Rails Caching

ActiveSupport::Cache::FileStore

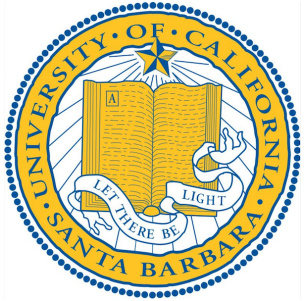
- Cached data is stored on the local file system.
- Can configure the location of the storage in Rails environment:
  - `config.cache_store = :file_store, "/path/to/cache/directory"`
- Strengths?
- Weaknesses?



Rails Caching

ActiveSupport::Cache::FileStore

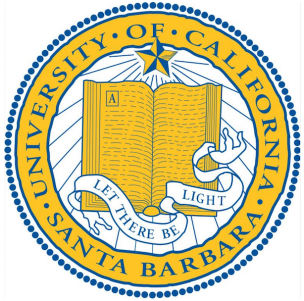
- Cached data is stored on the local file system.
- Can configure the location of the storage in Rails environment:
  - `config.cache_store = :file_store, "/path/to/cache/directory"`
- Strengths
  - A pool of processes on the same machine can now share the same cache.
- Weaknesses
  - Disk isn't all that fast
  - If our production deployment has many machines, they can't share caches.



Rails Caching

ActiveSupport::Cache::MemcacheStore

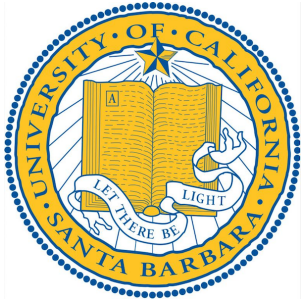
- Cached data is stored in memory on another machine.
- Can configure the location of the server in Rails environment:
  - `config.cache_store = :mem_cache_store, "cache-1.example.com"`
- Strengths
  - A pool of processes across machines can now share the same cache.
  - As we've seen, intra-datacenter network round trip and memory access can give us ~1ms responses.
- Weaknesses
  - System complexity



Rails Caching - HTTP Caching

HTTP Caching: Rails makes this easy

- By default, Rails sets the etag to an MD5 digest of the content body.
  - If subsequent requests generate the same body, a 304 is sent instead of the body.
  - This is network efficient and easy, but what's still inefficient?

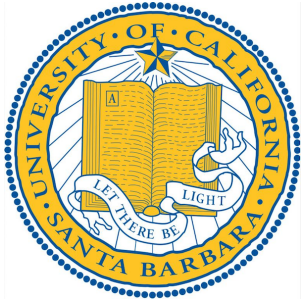


Rails Caching - HTTP Caching

We're still generating the response each time.

Solution:

```
class ProductsController < ApplicationController
  def show
    @product = Product.find(params[:id])
    fresh_when last_modified: @product.published_at.utc,
               etag: @product
  end
end
```



Rails Caching - Fragment Caching

Whole-page caching is great when possible, but for dynamic web applications, it frequently isn't.

Fragment caching caches a portion of a rendered view for reuse on future requests.

For example, we may have a page listing recent orders (dynamic) alongside a list of products (less dynamic)

Rails Caching - Fragment Caching

```
<% Order.find_recent.each do |o| %>
  <%= o.buyer.name %> bought <%= o.product.name %>
<% end %>

<% cache ‘all_available_products’ do %>
  All available products:
  <% Product.all.each do |p| %>
    <%= link_to p.name, product_url(p) %>
  <% end %>
<% end %>
```

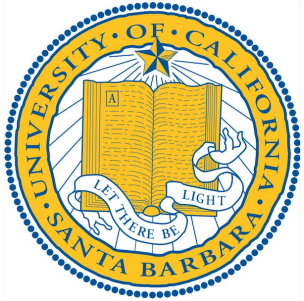
Rails Caching - Fragment Caching

When the products **do** change, the fragment must be expired using

```
expire_fragment('all_available_products')
```

Manually managing cache expiration can be tricky and lead to bugs.

Can you think of a better way?

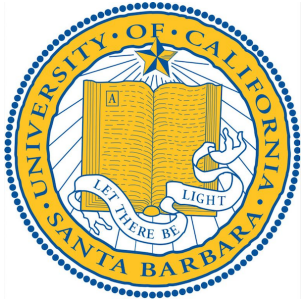


Rails Caching - Fragment Caching

```
module ProductsHelper
  def cache_key_for_products
    count          = Product.count
    max_updated_at = Product.maximum(:updated_at).utc.to_s, :number)
    "products/all-#{count}-#{max_updated_at}"
  end
end
```

…

```
<% cache(cache_key_for_products) do %>
  All available products:
    <% Product.all.each do |p| %>
    <%= link_to p.name, product_url(p) %>
  <% end %>
<% end %>
```

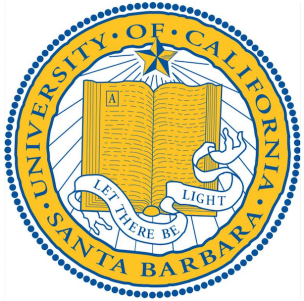


Rails Caching - Fragment Caching

Another option is to use the model itself as the cache key:

```
<% Product.all.each do |p| %>
  <% cache(p) do %>
    <%= link_to p.name, product_url(p) %>
  <% end %>
<% end %>
```

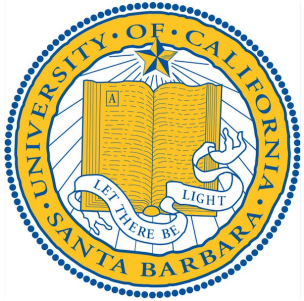
Which will generate a cache key that includes the updated_at timestamp (something like "`products/23-20130109142513`").



Rails Caching - Fragment Caching

We can also nest our fragment caching, for a style referred to as “Russian Doll Caching”:

```
<% cache(cache_key_for_products) do %>
  All available products:
  <% Product.all.each do |p| %>
    <% cache(p) do %>
    <%= link_to p.name, product_url(p) %>
    <% end %>
  <% end %>
<% end %>
```

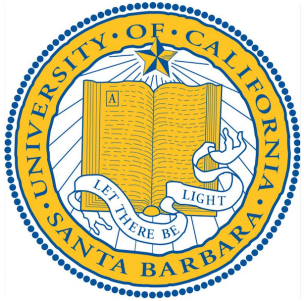


Rails Caching - Manual Caching

Manual Caching is also supported, for fine-grained control:

```
#product.rb

class Product
  def competing_price
    Rails.cache.fetch("/product/#{id}-#{updated_at}/comp_price",
      expires_in => 12.hours) do
    Competitor::API.find_price(id)
    end
  end
end
```



For Next Time...

Be in a group for tomorrow's lab. We will be kicking off projects.

Citrix data center tour Nov 6!

By tomorrow at noon, put on Piazza:

- Team name
- For each team member
  - Name, Github account name, Email