CS 188

Scalable Internet Services

Andrew Mutz

October 17, 2019

Today's Agenda

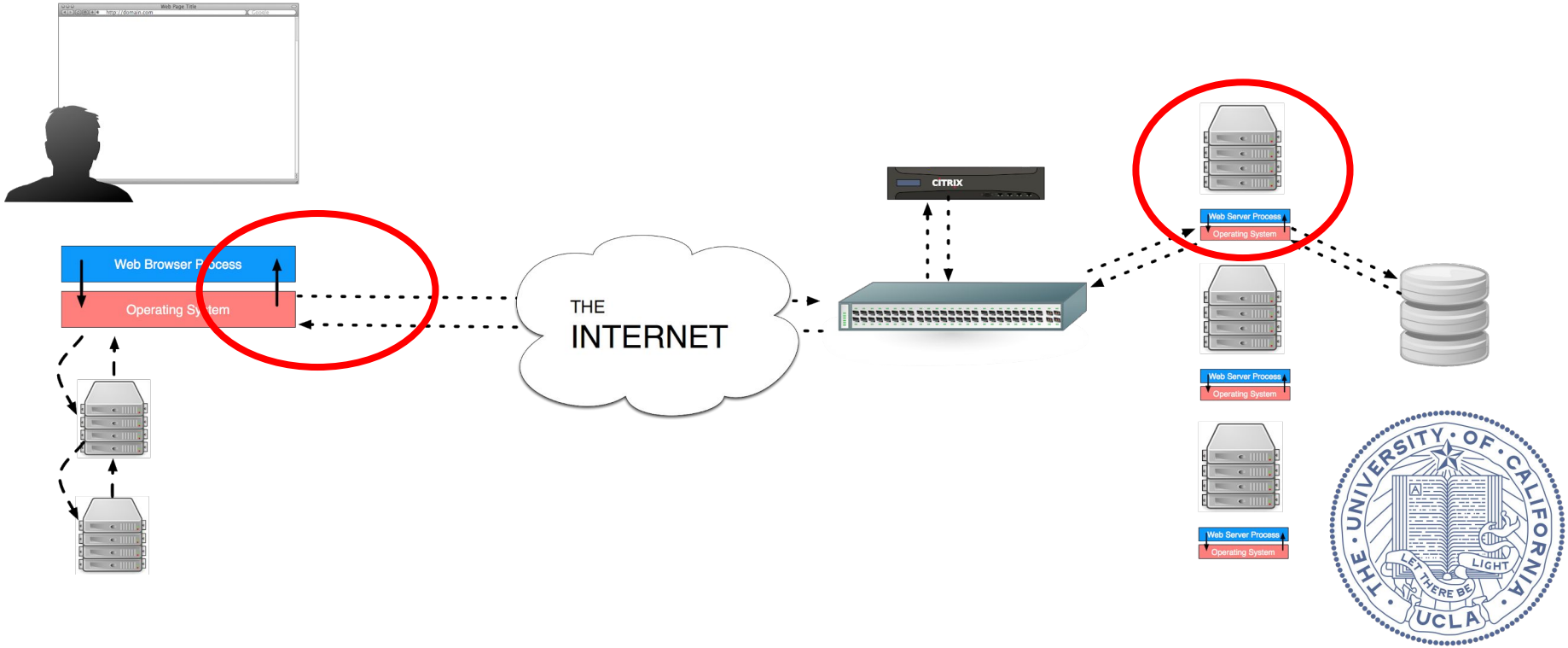
Motivation

Client-side caching

Server-side Caching

For Next Time

Client-side Caching



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

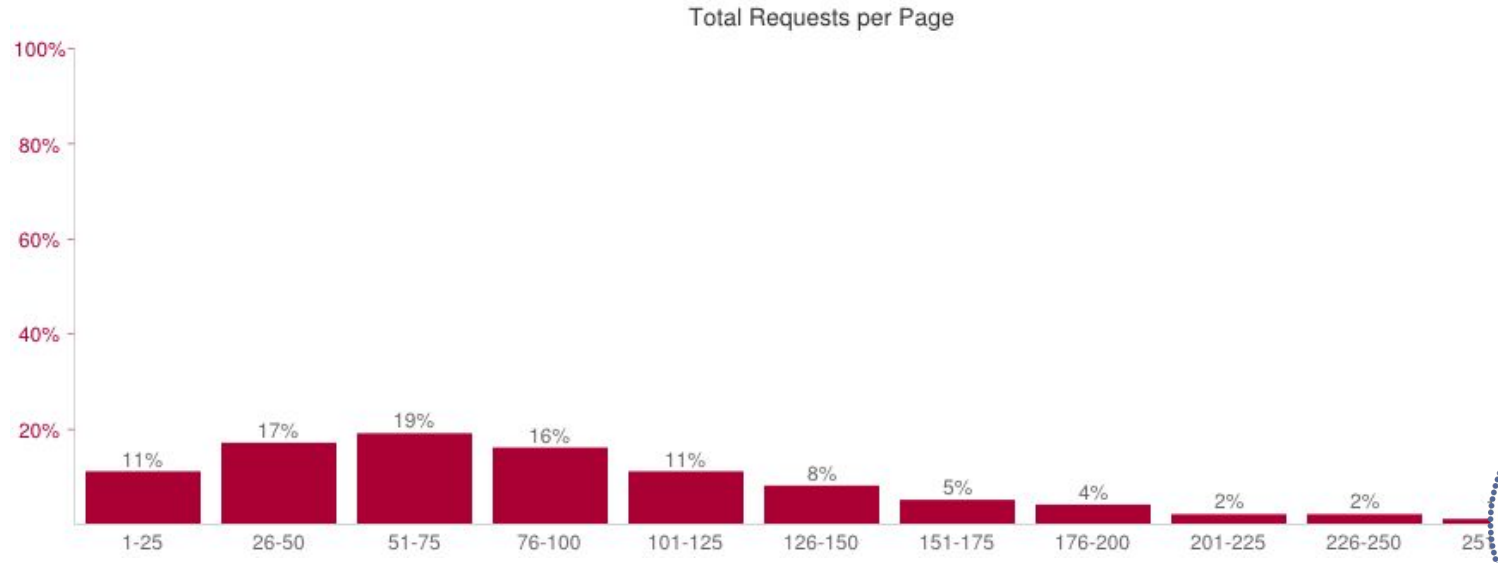
But there are challenges:

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?

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)

Client-side Caching

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

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

`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: 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: 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

`cache-control: max-age=120`

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

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

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

`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”`

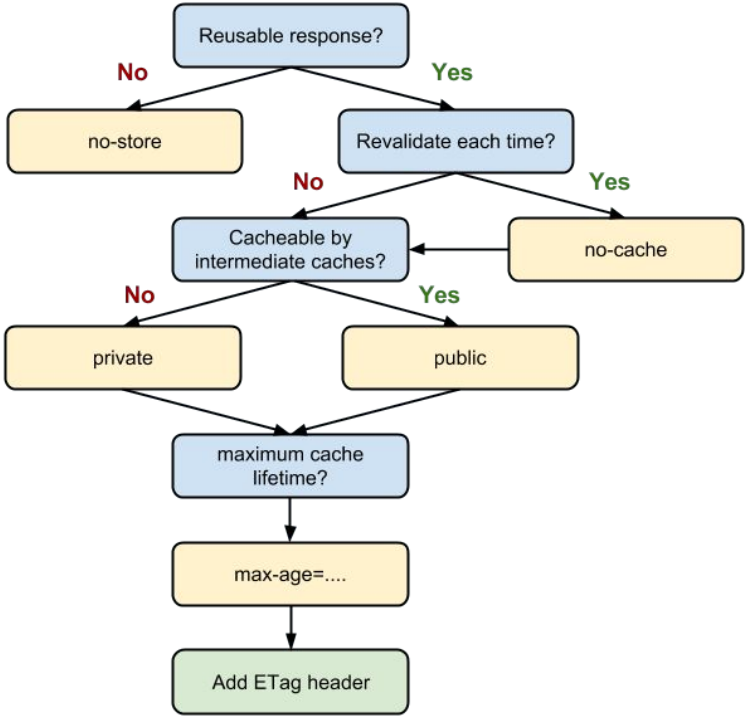
When accompanying a request, this indicates that the client has a cached copy with the associated tag. Multiple etags can be provided.

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

In Summary...



source: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching

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.

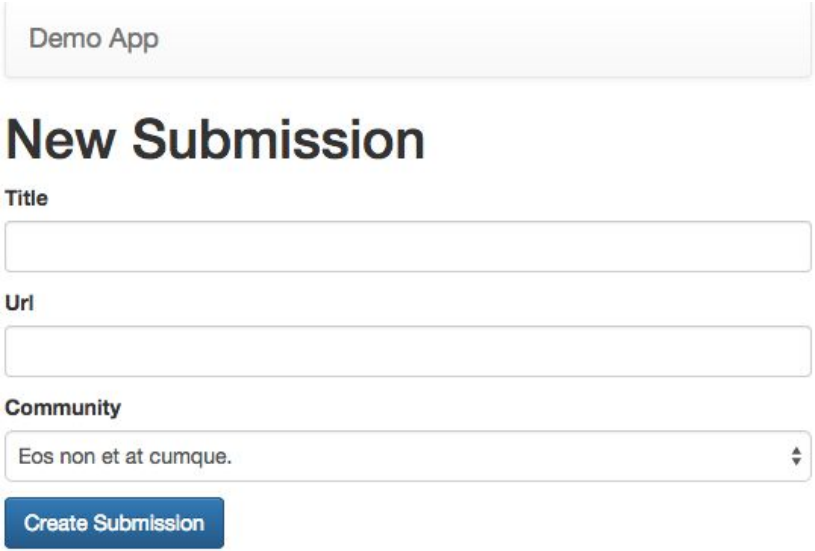
What headers should the response include?

Client-side Caching

We don't want it stored at all:

`Cache-control: no-store`

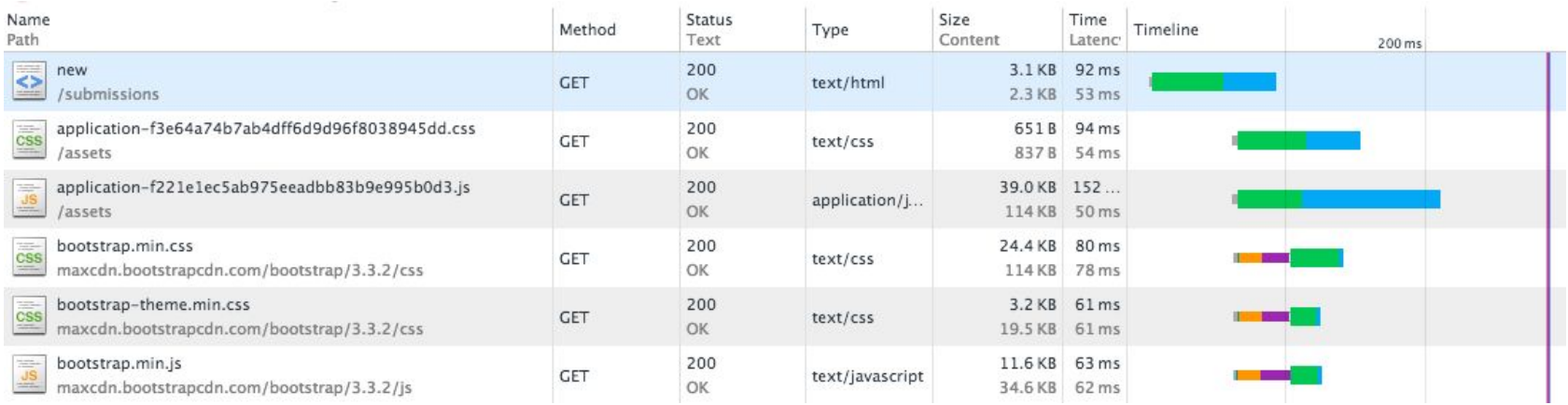
Client-side Caching



Lets try this out on the demo app

Lets implement HTTP caching for this page in the UI.

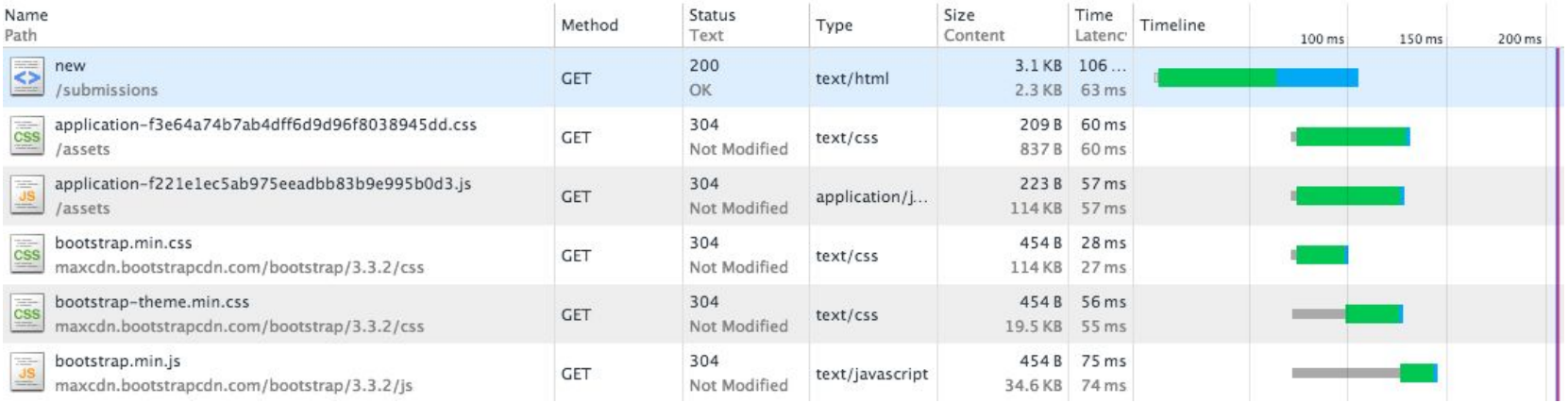
Client-side Caching



Name / Path	Method	Status / Text	Type	Size / Content	Time / Latenc	Timeline (200 ms)
new /submissions	GET	200 OK	text/html	3.1 KB 2.3 KB	92 ms 53 ms	
application-f3e64a74b7ab4dff6d9d96f8038945dd.css /assets	GET	200 OK	text/css	651 B 837 B	94 ms 54 ms	
application-f221e1ec5ab975eeadbb83b9e995b0d3.js /assets	GET	200 OK	application/j...	39.0 KB 114 KB	152 ... 50 ms	
bootstrap.min.css maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css	GET	200 OK	text/css	24.4 KB 114 KB	80 ms 78 ms	
bootstrap-theme.min.css maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css	GET	200 OK	text/css	3.2 KB 19.5 KB	61 ms 61 ms	
bootstrap.min.js maxcdn.bootstrapcdn.com/bootstrap/3.3.2/js	GET	200 OK	text/javascript	11.6 KB 34.6 KB	63 ms 62 ms	

Initial page load gets every resource

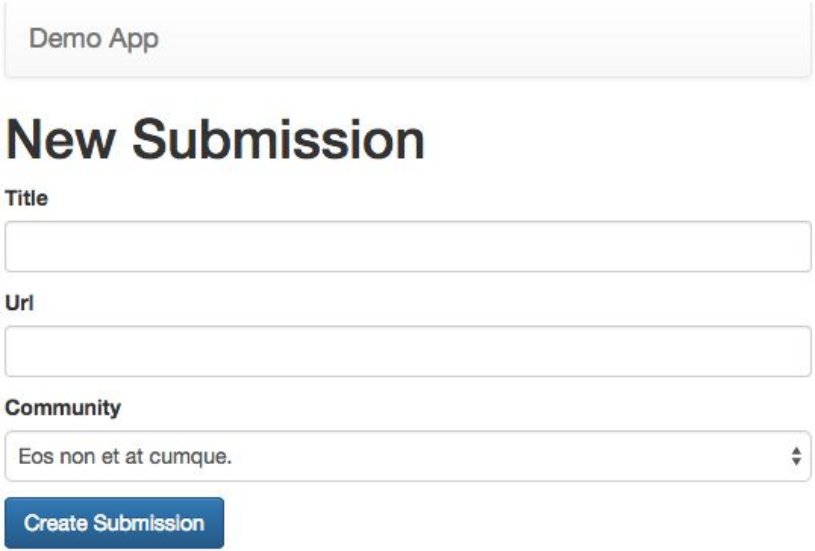
Client-side Caching



Name / Path	Method	Status / Text	Type	Size / Content	Time / Latenc	Timeline (100 ms, 150 ms, 200 ms)
new /submissions	GET	200 OK	text/html	3.1 KB 2.3 KB	106 … 63 ms	
application-f3e64a74b7ab4dff6d9d96f8038945dd.css /assets	GET	304 Not Modified	text/css	209 B 837 B	60 ms 60 ms	
application-f221e1ec5ab975eeadbb83b9e995b0d3.js /assets	GET	304 Not Modified	application/j…	223 B 114 KB	57 ms 57 ms	
bootstrap.min.css maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css	GET	304 Not Modified	text/css	454 B 114 KB	28 ms 27 ms	
bootstrap-theme.min.css maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css	GET	304 Not Modified	text/css	454 B 19.5 KB	56 ms 55 ms	
bootstrap.min.js maxcdn.bootstrapcdn.com/bootstrap/3.3.2/js	GET	304 Not Modified	text/javascript	454 B 34.6 KB	75 ms 74 ms	

Refreshing the page doesn't re-download the assets, but it does redownload /submissions/new

Client-side Caching



When can this page be out of date?

Client-side Caching

```
class SubmissionsController < ApplicationController
  ...

  def new
    @submission = Submission.new
  end

  ...
end
```

Client-side Caching

```
class SubmissionsController < ApplicationController
  ...

  def new
    @submission = Submission.new if stale?(Community.all)
  end

  ...
end
```

Client-side Caching

What is this actually doing?

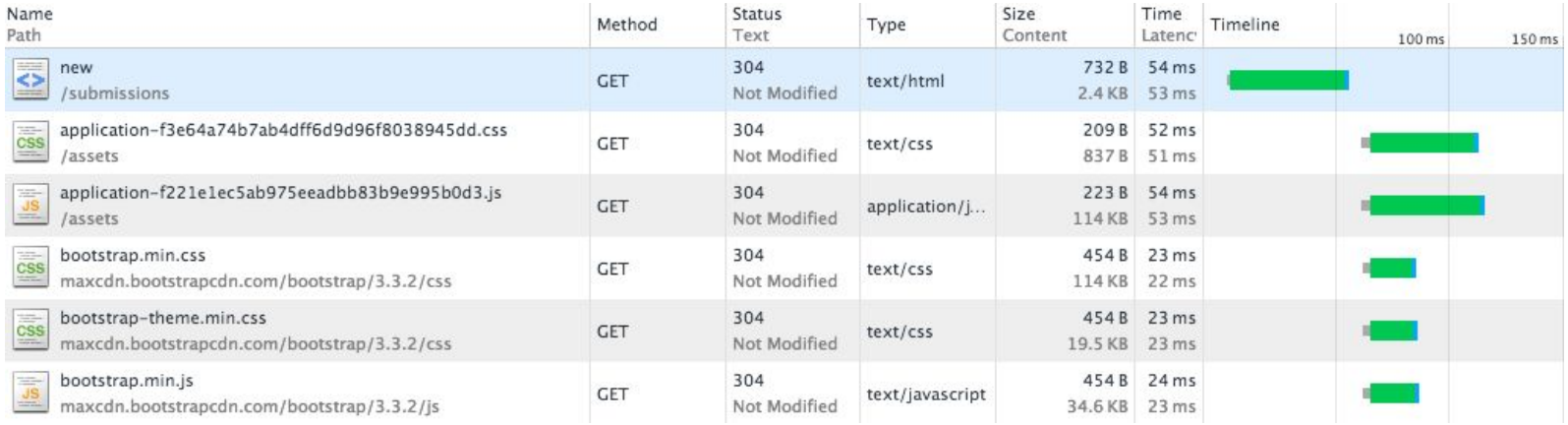
- `if stale?(Community.all)`

Client-side Caching

What is this actually doing?

- `if stale?(Community.all)`
- This tells Rails to base the etag on all communities
- You can think of this as taking the digest of all the concatenated `updated_at` fields of all communities.
- Whenever this controller is invoked, Rails compares the etag presented in the request to the etag that would be generated
  - If they are the same, it returns a 304

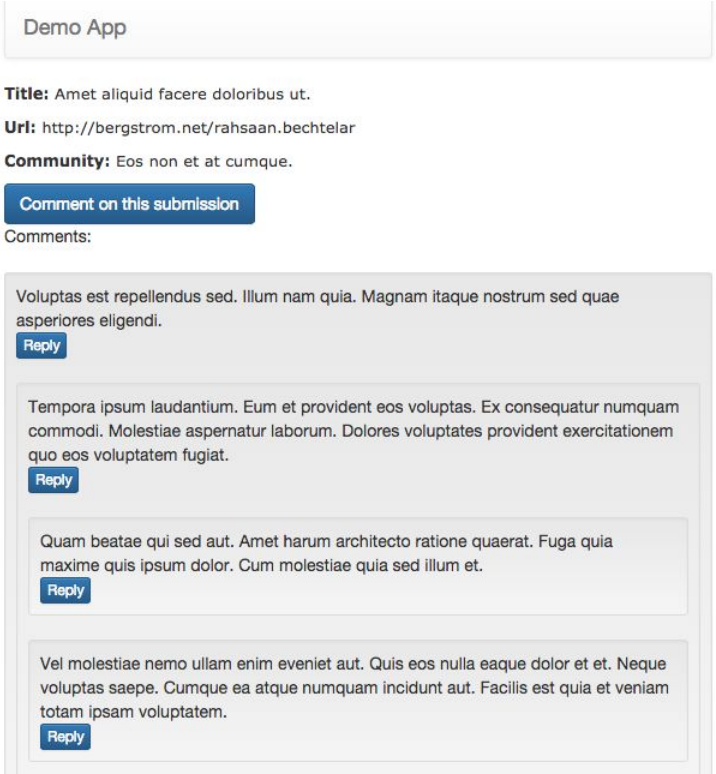
Client-side Caching



Name / Path	Method	Status / Text	Type	Size / Content	Time / Latenc	Timeline (100 ms, 150 ms)
new /submissions	GET	304 Not Modified	text/html	732 B 2.4 KB	54 ms 53 ms	
application-f3e64a74b7ab4dff6d9d96f8038945dd.css /assets	GET	304 Not Modified	text/css	209 B 837 B	52 ms 51 ms	
application-f221e1ec5ab975eeadbb83b9e995b0d3.js /assets	GET	304 Not Modified	application/j...	223 B 114 KB	54 ms 53 ms	
bootstrap.min.css maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css	GET	304 Not Modified	text/css	454 B 114 KB	23 ms 22 ms	
bootstrap-theme.min.css maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css	GET	304 Not Modified	text/css	454 B 19.5 KB	23 ms 23 ms	
bootstrap.min.js maxcdn.bootstrapcdn.com/bootstrap/3.3.2/js	GET	304 Not Modified	text/javascript	454 B 34.6 KB	24 ms 23 ms	

The web console indicates we are successful. Adding a new Community causes a 200 response.

Client-side Caching



Lets try this same technique for this part of the UI.

What can make this page stale?

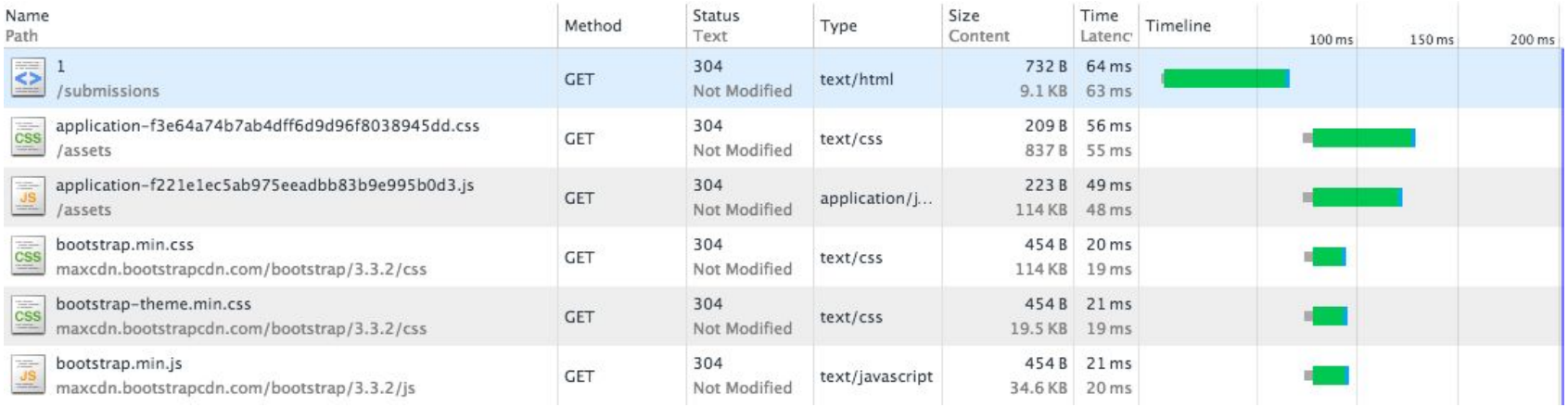
Client-side Caching

```
class SubmissionsController < ApplicationController
  before_action :set_submission, only: [:show, :edit, :update, :destroy]

  …

  def show
  end

  …

  def set_submission
    @submission = Submission.find(params[:id])
  end
end
```

Client-side Caching

```
class SubmissionsController < ApplicationController
  before_action :set_submission, only: [:show, :edit, :update, :destroy]

  …

  def show
   fresh_when([@submission, @submission.community, @submission.comments])
  end

  …

  def set_submission
    @submission = Submission.find(params[:id])
  end
end
```

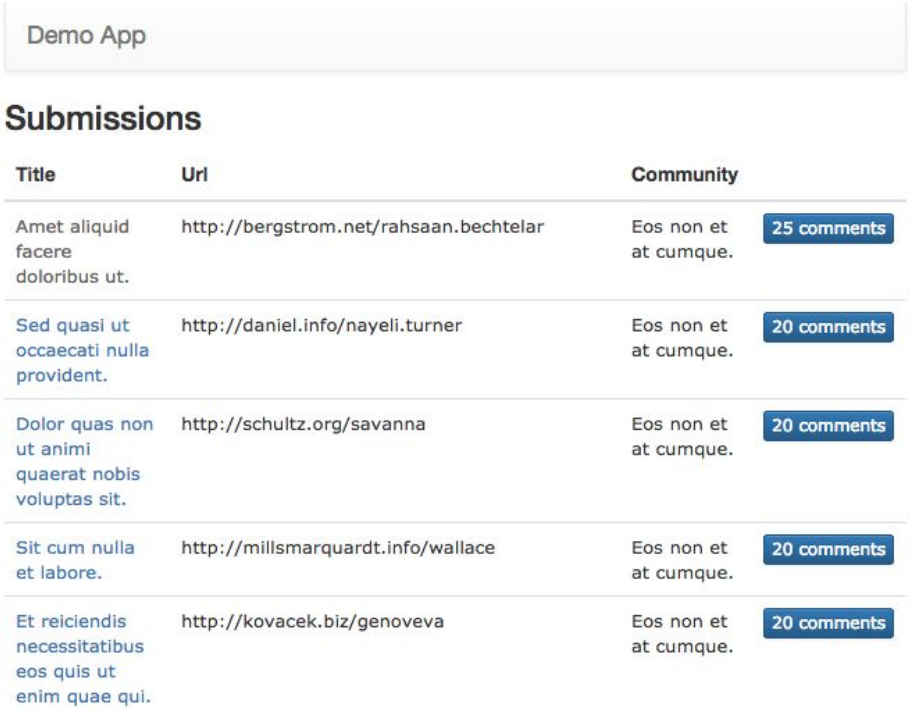
Client-side Caching



Name / Path	Method	Status / Text	Type	Size / Content	Time / Latenc	Timeline (100 ms, 150 ms, 200 ms)
1 /submissions	GET	304 Not Modified	text/html	732 B 9.1 KB	64 ms 63 ms	
application-f3e64a74b7ab4dff6d9d96f8038945dd.css /assets	GET	304 Not Modified	text/css	209 B 837 B	56 ms 55 ms	
application-f221e1ec5ab975eeadbb83b9e995b0d3.js /assets	GET	304 Not Modified	application/j...	223 B 114 KB	49 ms 48 ms	
bootstrap.min.css maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css	GET	304 Not Modified	text/css	454 B 114 KB	20 ms 19 ms	
bootstrap-theme.min.css maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css	GET	304 Not Modified	text/css	454 B 19.5 KB	21 ms 19 ms	
bootstrap.min.js maxcdn.bootstrapcdn.com/bootstrap/3.3.2/js	GET	304 Not Modified	text/javascript	454 B 34.6 KB	21 ms 20 ms	

The web console indicates we are successful. Adding a new Comment or modifying the community causes a 200 response.

Client-side Caching



How about the index page?

What can make this page stale?

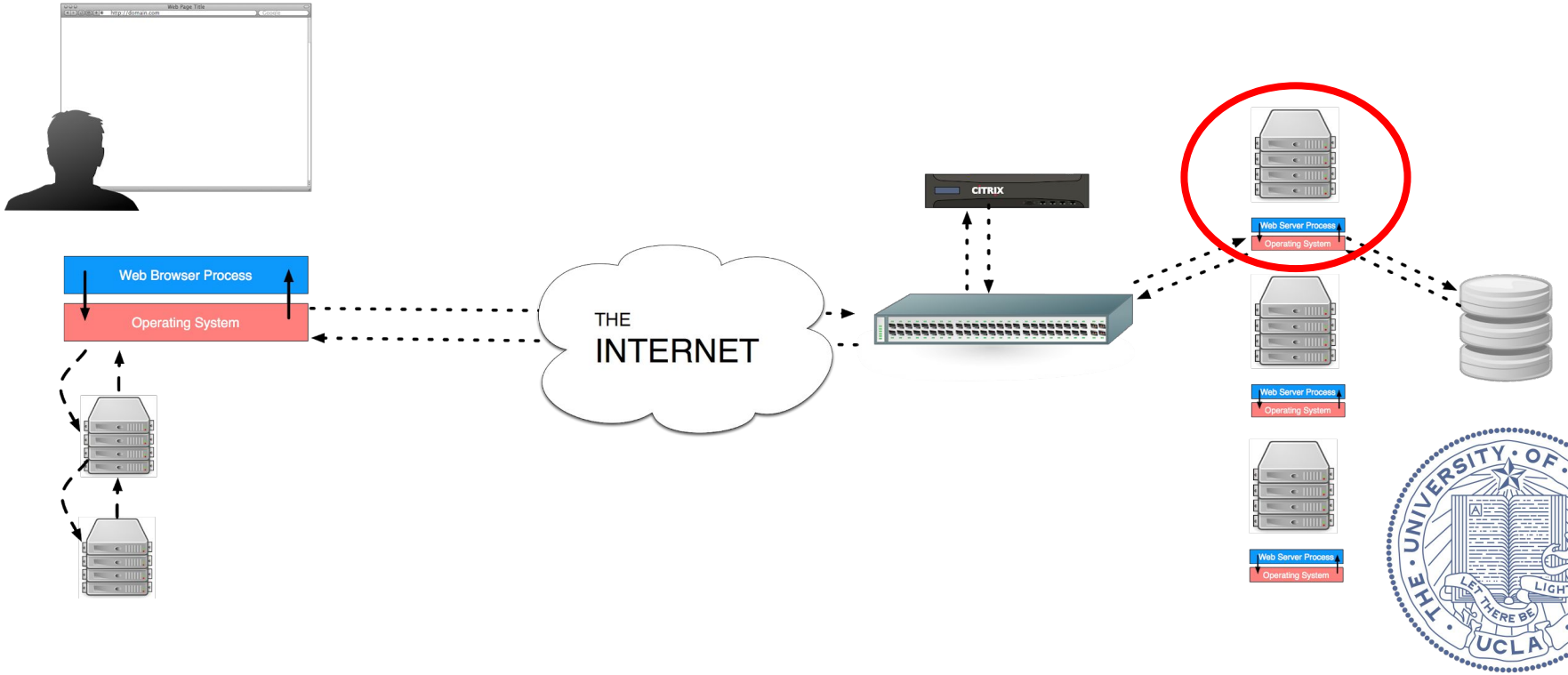
Client-side Caching

```
class SubmissionsController < ApplicationController
  …

  def index
    @submissions = Submission.all
  end

  …

end
```

Client-side Caching

```
class SubmissionsController < ApplicationController
  …

  def index
    if stale?([Submission.all, Community.all, Comments.all])
      @submissions = Submission.all
    end
  end
  …

end
```

Server-side Caching



Motivation

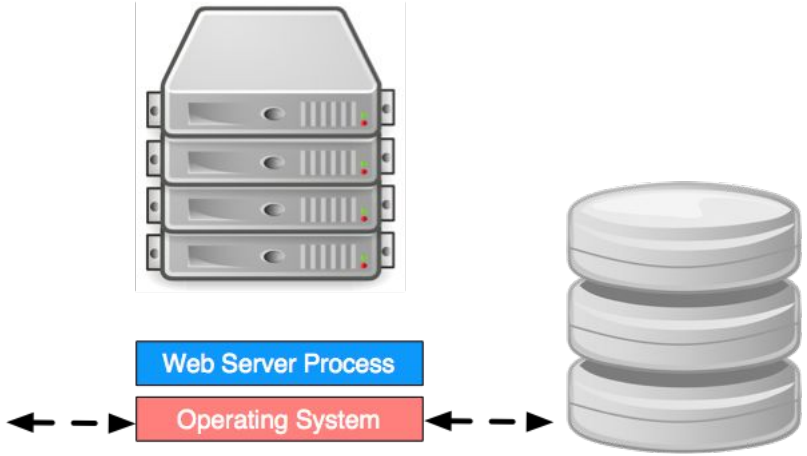
After today you should understand

- Why server side caching exists
- What options you have when using server side caching
- How to use this in your projects

Server-side Caching

We have a web server process that is repeatedly responding to HTTP requests from a variety of clients.

Responding to each request requires computation and I/O to be performed, and this can be expensive.

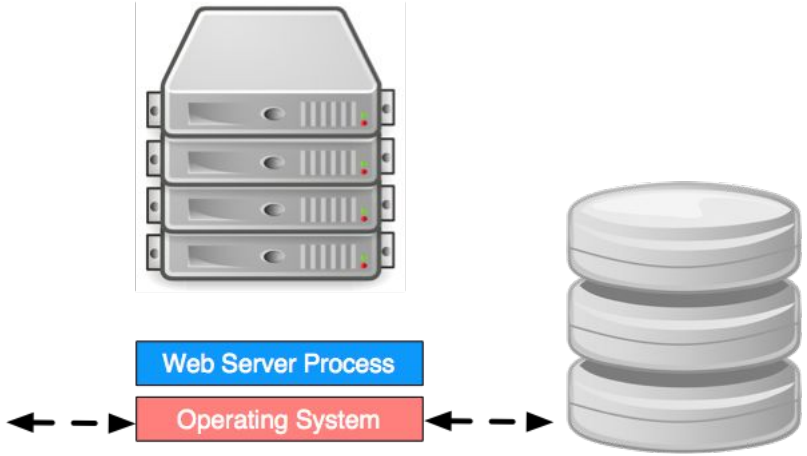


Server-side Caching

In practice, there is a great deal of similarity between responses.

With client-side caching we looked at optimizing scenarios where repeated responses are identical.

In this lecture we will look at optimizing scenarios where repeated responses are not identical, but are similar.

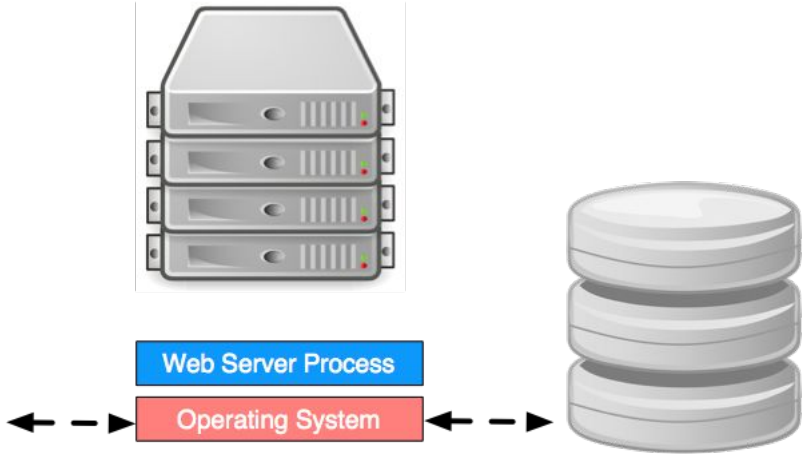


Server-side Caching

There are many parts of a response that are similar.

There are many steps to creating a response that are repeated.

What can you think of?



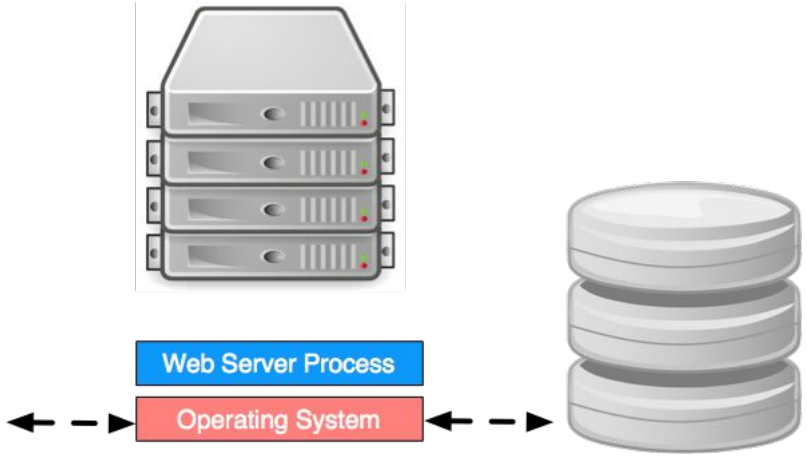
Server-side Caching

There are many parts of a response that are similar.

There are many steps to creating a response that are repeated.

What can you think of?

- View Fragments
- Rarely modified ORM objects
- Any summarized data that is expensive to compute



Server-side Caching

View Fragments: Similarity between pages

```
<html><head><meta name="referrer" content="origin"><link rel="stylesheet" type="text/css" href="news.css?Dc4WHhDIHIntiL2Oh45C">

        <link rel="shortcut icon" href="favicon.ico">

          <link rel="alternate" type="application/rss+xml" title="RSS" href="rss">

        <script type="text/javascript">

          function hide(id) { var el = document.getElementById(id); if (el) { el.style.visibility = 'hidden'; } }

          function vote(node) { var v = node.id.split(/_/); var item = v[1]; hide('up_'   + item); hide('down_' + item); var ping = new Image(); ping.src = node.href;
return false;}

        </script><title>Hacker News</title></head><body><center><table id="hnmain" op="news" border="0" cellpadding="0" cellspacing="0" width="85%" bgcolor="#f6f6ef">

        <tr><td bgcolor="#ff6600"><table border="0" cellpadding="0" cellspacing="0" width="100%" style="padding:2px"><tr><td style="width:18px;padding-right:4px"><a
href="http://www.ycombinator.com"><img src="y18.gif" width="18" height="18" style="border:1px #ffffff solid;"></a></td>

                  <td style="line-height:12pt; height:10px;"><span class="pagetop">

                              <b><a href="news">Hacker News</a></b><img src="s.gif" height="1" width="10"><a href="newest">new</a> | <a href="newcomments">comments</a> | <a
href="show">show</a> | <a href="ask">ask</a> | <a href="jobs">jobs</a> | <a href="submit">submit</a></span></td><td style="text-align:right;padding-right:4px;"><span
class="pagetop">

                              <a href="login?goto=news">login</a></span></td></tr></table></td></tr><tr style="height:10px"></tr><tr><td><table border="0" cellpadding="0"
cellspacing="0">

            <tr><td align="right" valign="top" class="title"><span class="rank">1.</span></td>

            <td><center><a id="up_9403571" href="vote?for=9403571&amp;dir=up&amp;goto=news"><div class="votearrow" title="upvote"></div></a></center></td><td
class="title"><span class="deadmark"></span><a href="http://fossdroid.com/">Fossdroid.com: Free and open source Android applications</a><span class="sitebit comhead">
(fossdroid.com)</span></td></tr><tr><td colspan="2"></td><td class="subtext">

        <span class="score" id="score_9403571">164 points</span> by <a href="user?id=SnaKeZ">SnaKeZ</a> <a href="item?id=9403571">4 hours ago</a>                | <a
href="item?id=9403571">38 comments</a></td></tr>

      <tr class="spacer" style="height:5px"></tr><tr>.....
```

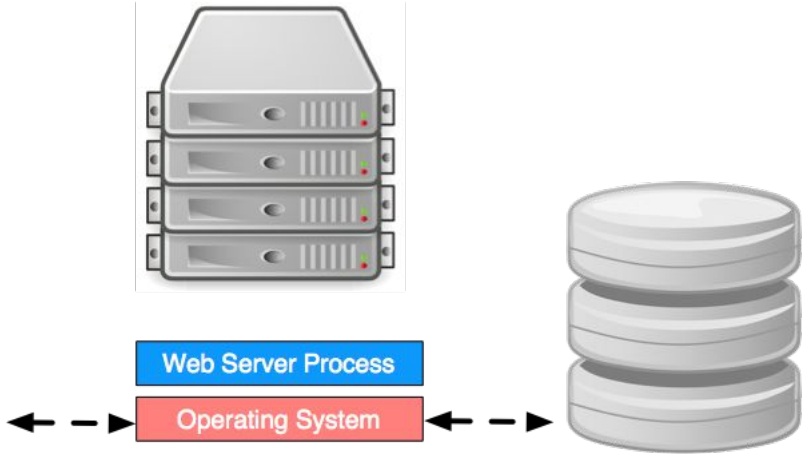
Server-side Caching

Rarely modified ORM objects?

- User permissions
- Configuration options
- Any database-backed data that changes rarely

Summarized data that is difficult to compute

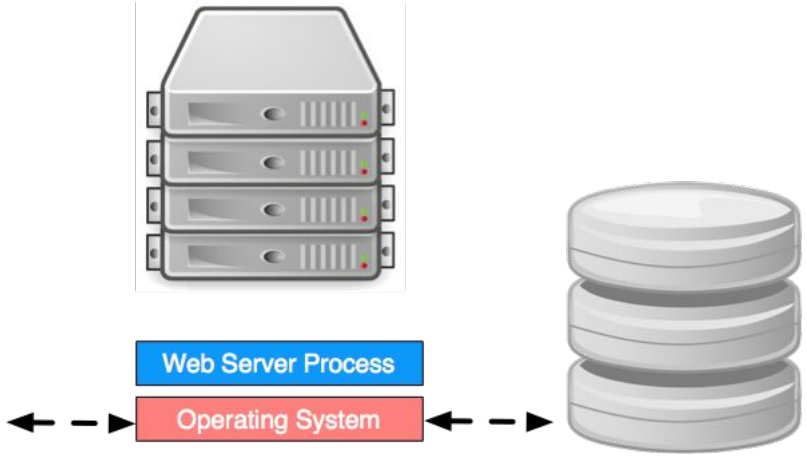
- Any particularly heavyweight SQL query
- Example: Total account balance on Mint.com



Server-side Caching

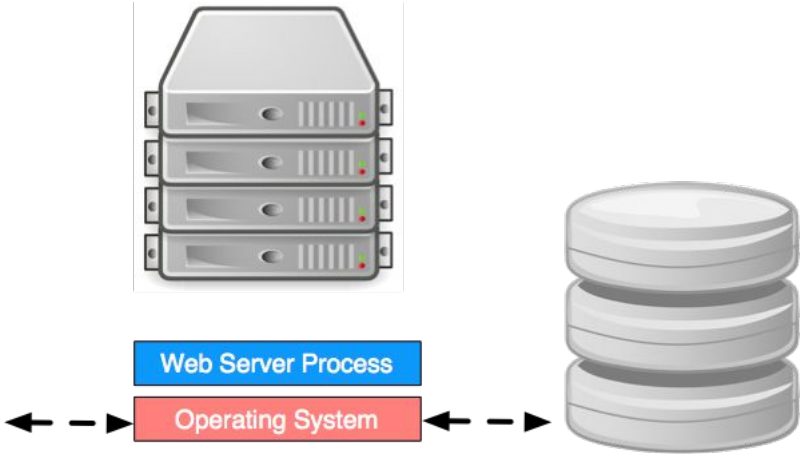
Some of these things are expensive to materialize

- View fragments are produced by extensive string manipulation.
  - Ruby optimizes humans over CPU
- The database can be a bottleneck in our current architecture
- Some SQL queries are necessarily heavyweight



Server-side Caching

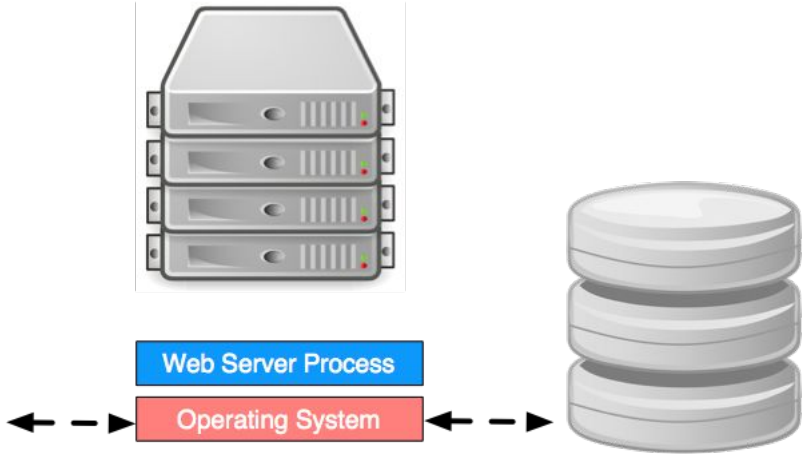
So if we want to keep previously computed results around between requests, how should we do it? Where should we put it?



Server-side Caching

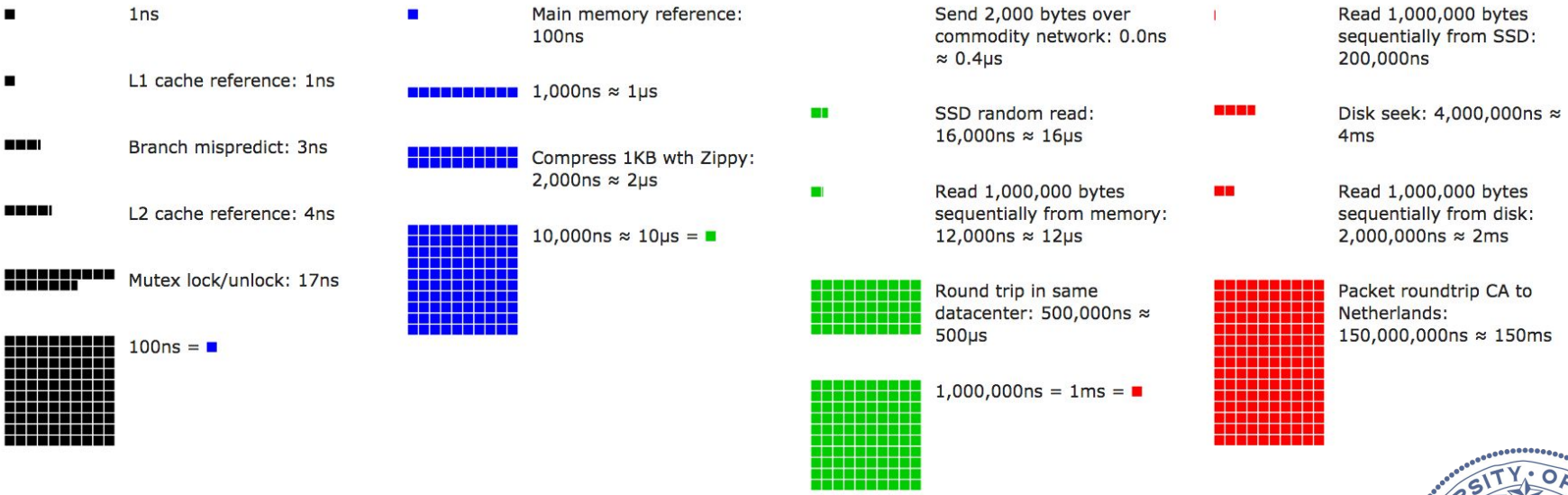
So if we want to keep previously computed results around between requests, how should we do it? Where should we put it?

- Just keep it in memory between requests?
- Store it on the filesystem?
- Store it in memory on another machine?



All of these are reasonable options. Lets look into each in more depth.

Server-side Caching



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

Server-side Caching

What can we conclude from these numbers?

- Storing in memory and reading later is fast:
  - Random reads from memory will be 0.1µs, reading 1MB will be 12µs
- Storing on disk is slow *without SSD*:
  - Disk seek is 4000µs, subsequent sequential read of 1MB is 2000µs
- Storing on disk *with SSD* is much more reasonable:
  - Random read is 16µs, sequential read of 1MB will be 200µs
- Storing on another machine is reasonable:
  - Round trip within datacenter is 500µs.

Server-side Caching

Summary

- In memory: tens of µs
- On SSD: hundreds of µs
- On Disk: thousands of µs
- And if it's on a remote machine, add hundreds of µs.

Conclusion

- Always use SSD
- Memory > local SSD > Remote?

Server-side Caching

It's not that simple. Why?

Server-side Caching

It's not that simple.

What effect on the cache hit rate does each of these designs have?

- In memory: Cache per process
- On local SSD: Cache per machine
- On (single) remote machine: Cache per cluster

Server-side Caching

Conclusion:

- **In memory**: highest performance, lowest hit rate
- **On SSD**: lower performance, higher hit rate
- **On remote cache server**: lowest performance, highest hit rate

There is no silver bullet. How will each of these affect system performance:

- Number of processes per machine?
- Concurrency model of Application Server: threads vs. processes?
- Number of machines per cluster?

Memcached

Memcached is a commonly used implementation of a remote cache server

- Keeps a cache in memory
- Accepts TCP connections and returns lookup requests
- Distributed key-value store
  - Keys can be up to 250 bytes, values can be up to 1MB
  - Can scale horizontally
- When it runs out of space, it uses a simple LRU mechanism to make more space
- Lightweight features, everything is constant time.
- Originally developed at LiveJournal

Another commonly used tool is Redis

Rails Caching

The good news for your projects is that Rails has great support for server-side caching.

Rails emphasizes three types of caching:

- HTTP caching
- Fragment caching
- Low level caching

We covered HTTP caching in the last lecture, so today we will talk about fragment caching and 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.

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/"`

Rails Caching

ActiveSupport::Cache::MemcacheStore

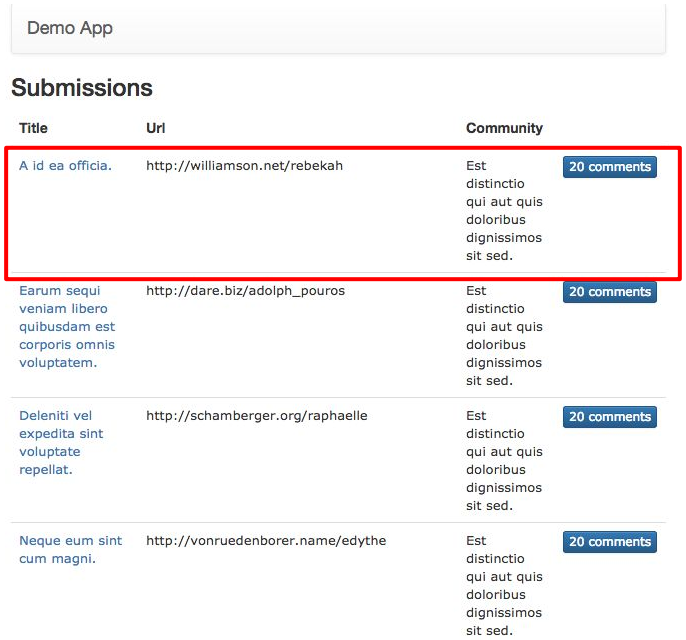
- Cached data is stored in memory on another machine.
- Can configure the location of the server in Rails environment:
  - `config.cache_store = :mem_cache_store, "cache-1.example.com"`

Rails Caching - Fragment Caching

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

Let’s take a look at the demo app...

Rails Caching - Fragment Caching



We can cache each line of this markup.

Regardless of anything else that changes on the page, we can rerender this if it stays fresh

Rails Caching - Fragment Caching

```
    <tbody>
      <% @submissions.each do |submission| %>
        <tr>
          <td><%= link_to(submission.title, submission.url) %></td>
          <td><%= submission.url %></td>
          <td><%= submission.community.name %></td>
          <td><%= link_to "#{submission.comments.size} comments", submission, class: 'btn
btn-primary btn-xs'  %></td>
        </tr>
      <% end %>
    </tbody>
  </table>
```

Rails Caching - Fragment Caching

```
    <tbody>
      <% @submissions.each do |submission| %>
        <% cache(cache_key_for_submission_row(submission)) do %>
          <tr>
            <td><%= link_to(submission.title, submission.url) %></td>
            <td><%= submission.url %></td>
            <td><%= submission.community.name %></td>
            <td><%= link_to "#{submission.comments.size} comments", submission, class: 'btn
btn-primary btn-xs'  %></td>
          </tr>
        <% end %>
      <% end %>
    </tbody>
  </table>
```

Rails Caching - Fragment Caching

How should we choose a cache key?

```
module SubmissionsHelper
  def cache_key_for_submission_row(submission)
    "submission-#{submission.id}"
  end
end
```

What are the weaknesses with the above approach?

Rails Caching - Fragment Caching

How should we choose a cache key?

```
module SubmissionsHelper
  def cache_key_for_submission_row(submission)
    "submission-#{submission.id}"
  end
end
```

What are the weaknesses with the above approach?

- Invalidation will be annoying: clear out the cache on possible action causing staleness?

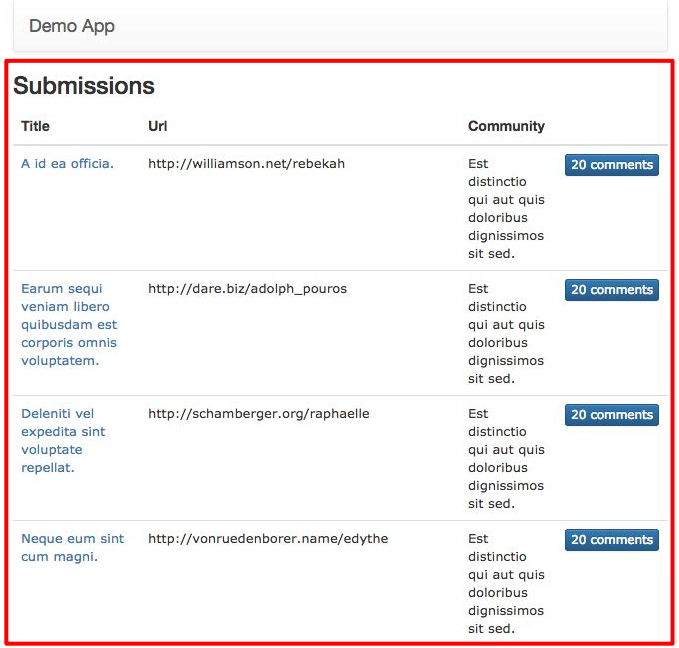
Rails Caching - Fragment Caching

Instead, let's make the key change whenever the data gets stale.

```
module SubmissionsHelper
  def cache_key_for_submission_row(submission)
    "submission-#{submission.id}-#{submission.updated_at}-#{submission.comments.count}"
  end
end
```

There is no action needed to invalidate the cache: the cache key changes.

Rails Caching - Fragment Caching



If we step back and look at this page, we can observe that the whole table is expensive to compute and stays fresh for awhile.

Rails Caching - Fragment Caching

```
<h3>Submissions</h3>
<table class="table">
  <thead>
    <tr>
      <th>Title</th>
      <th>Url</th>
      <th>Community</th>
      <th colspan="3"></th>
    </tr>
  </thead>

  <tbody>
    <% @submissions.each do |submission| %>
      <% cache(cache_key_for_submission_row(submission)) do %>
        <tr>
          <td><%= link_to(submission.title, submission.url) %></td>
          <td><%= submission.url %></td>
          <td><%= submission.community.name %></td>
          <td><%= link_to "#{submission.comments.size} comments", submission, class: 'btn btn-primary btn-xs'  %></td>
        </tr>
      <% end %>
    <% end %>
  </tbody>
</table>

<br>
<%= link_to 'New Submission', new_submission_path, class: 'btn btn-primary' %>
<%= link_to 'New Community', new_community_path, class: 'btn btn-primary' %>
```

Rails Caching - Fragment Caching

```
<% cache(cache_key_for_submission_table) do %>
  <h3>Submissions</h3>
  <table class="table">
    <thead>
      <tr>
        <th>Title</th>
        <th>Url</th>
        <th>Community</th>
        <th colspan="3"></th>
      </tr>
    </thead>

    <tbody>
      <% @submissions.each do |submission| %>
        <% cache(cache_key_for_submission_row(submission)) do %>
          <tr>
            <td><%= link_to(submission.title, submission.url) %></td>
            <td><%= submission.url %></td>
            <td><%= submission.community.name %></td>
            <td><%= link_to "#{submission.comments.size} comments", submission, class: 'btn btn-primary btn-xs'  %></td>
          </tr>
        <% end %>
      <% end %>
    </tbody>
  </table>

  <br>
  <%= link_to 'New Submission', new_submission_path, class: 'btn btn-primary' %>
  <%= link_to 'New Community', new_community_path, class: 'btn btn-primary' %>
<% end %>
```

Rails Caching - Fragment Caching

```
module SubmissionsHelper
  def cache_key_for_submission_row(submission)
    "submission-#{submission.id}-#{submission.updated_at}-#{submission.comments.count}"
  end
  def cache_key_for_submission_table
    "submission-table-#{Submission.maximum(:updated_at)}-#{Comment.maximum(:updated_at)}"
  end
end
```

This technique of nesting cache fragments is known as “Russian Doll” caching.

Rails Caching - Low-level Caching

You can use the same mechanisms to cache anything:

```
class Product < ActiveRecord::Base
  def competing_price
    Rails.cache.fetch("#{cache_key}/competing_price", expires_in: 12.hours) do
      Competitor::API.find_price(id)
    end
  end
end
```

Rails Caching - Low-level Caching

Let’s compare the demo app’s performance!

For these tests I will compare the performance of the branch master, with the branch “server_side_caching”, which implements the caching shown in the previous slides.

Master intentionally includes no optimizations to the way we interact with the database.

We will use the default (memory) caching.

Rails Caching

We'll use an M3-medium instance with the usual workload.

When we test we will have 12 phases, of 60 seconds each:

Phase	1	2	3	4	5	6	7	8	9	10	11	12
Users/sec	1	1.5	2	4	6	10	16	20	25	35	45	55

Deployed using nginx & passenger.

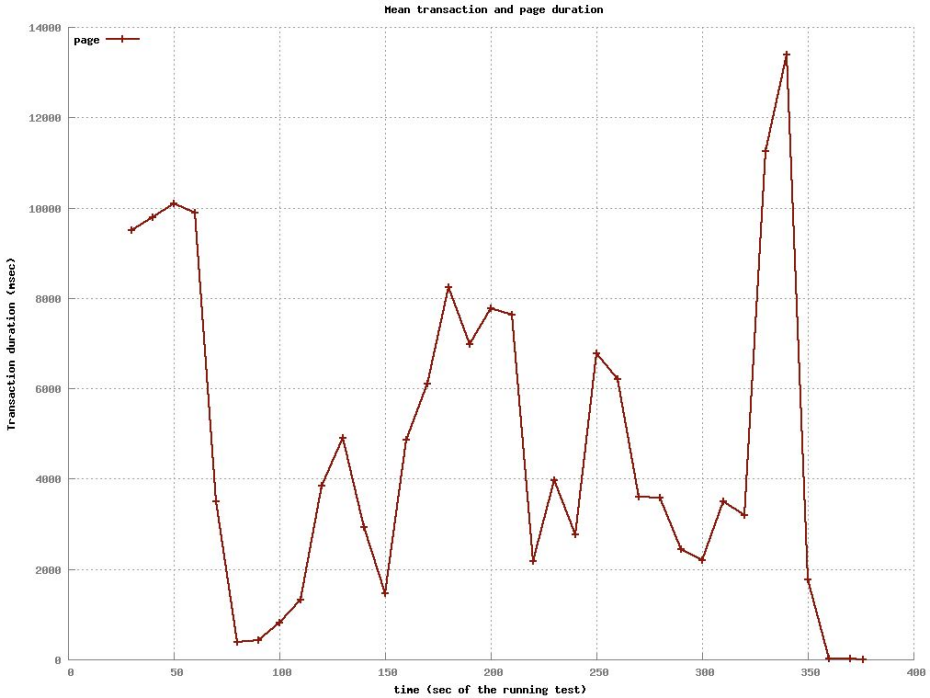
Rails Caching

We will use the same testing script from before...

1. Going to the homepage
2. Waiting for up to 2 seconds
3. Requesting a form to create a new community
4. Waiting for up to 2 seconds
5. Submitting the new community
6. Requesting a form to create a new link submission
7. Waiting for up to 2 seconds
8. Submitting the new link
9. Waiting for up to 2 seconds
10. Delete the link
11. Waiting for up to 2 seconds
12. Delete the community

Rails Caching

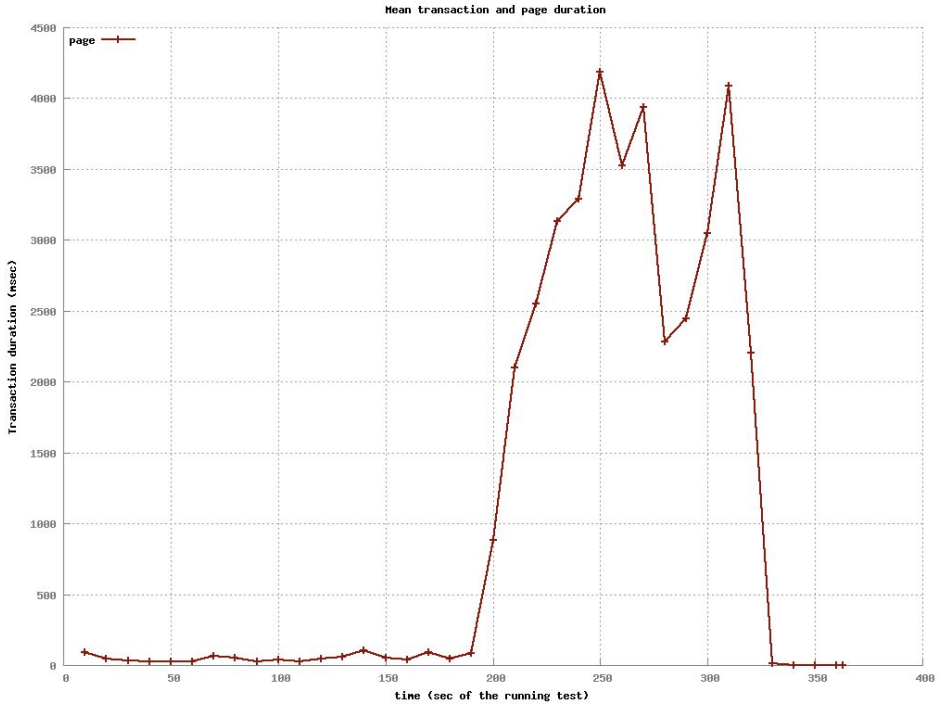
Performance on `master`:



From the start, the application can't handle a single user arriving each second

Rails Caching

Performance on `server_side_caching`:



With the server-side caching implemented, the server can handle up to two new users a second easily.

For Next Time...

Continue to work on sprint 1 stories. We will demo your progress at tomorrow's lab.

We will give out AWS access at tomorrow's lab