# HTTP Performance

Abram Hindle
abram.hindle@ualberta.ca
Department of Computing Science
University of Alberta
http://softwareprocess.es/

# Performance on the Web

- **Performance is a non functional requirement referring to how well a web site or service responds.**
- **Performance can be measured in terms**
  - **Requests**
  - **Volume**
  - **Latency**
  - **Bandwidth**
  - **Utilization**

# BEFORE YOU OPTIMIZE

- **Measure something:**
  - Requests, volume, latency, bandwidth, utilization, time
- **Record that number!**
  - You need to compare it against future values
- **Record and track the original settings.**
  - You need to compare to performance with your changes
- **Run your tests more than once.**
  - For real stats, you want more than 10 runs before and after
  - For t-tests and Wilcoxon tests you want 40+

# Google says...

- **Web Performance Best Practices https://developers.google.com/speed/docs/best-practices/rules_intro**
  - **Optimizing caching — keeping your application's data and logic off the network altogether**
  - **Minimizing round-trip times — reducing the number of serial request-response cycles**
  - **Minimizing request overhead — reducing upload size**
  - **Minimizing payload size — reducing the size of responses, downloads, and cached pages**
  - **Optimizing browser rendering — improving the browser's layout of a page**
  - **Optimizing for mobileNew! — tuning a site for the characteristics of mobile networks and mobile devices**
    - **Portions of this page are modifications based on work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.**

# Caching!

- Caching increase locality
- Locality increases bandwidth
- Locality decreases latency
- Levels of cache:
  - CPU
  - Memory
  - Disk
  - Network

# Caching!

```
> GET /static/SoftwareProcess.es.html HTTP/1.1
> User-Agent: curl/7.32.0
> Host: softwareprocess.es
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 07 Apr 2014 03:09:26 GMT
* Server Apache is not blacklisted
< Server: Apache
< Last-Modified: Mon, 07 Apr 2014 03:00:05 GMT
< ETag: "215f-4f66b107fc739"
< Accept-Ranges: bytes
< Content-Length: 8543
< Vary: Accept-Encoding
< Content-Type: text/html; charset=utf-8
<
```

# Caching! Do it Again!

```
> GET /static/SoftwareProcess.es.html HTTP/1.1
> User-Agent: curl/7.32.0
> Host: softwareprocess.es
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 07 Apr 2014 03:10:50 GMT
* Server Apache is not blacklisted
< Server: Apache
< Last-Modified: Mon, 07 Apr 2014 03:00:05 GMT
< ETag: "215f-4f66b107fc739"
< Accept-Ranges: bytes
< Content-Length: 8543
< Vary: Accept-Encoding
< Content-Type: text/html; charset=utf-8
<
```

# Cold GET

SoftwareProcess.es Wiki – Abram Hindle: SoftwareProcess.es – Nightly

SoftwareProcess.es W...

softwareprocess.es/static/SoftwareProcess.es.html

Google

Abram is an assistant professor at the **University of Alberta**, in Edmonton, Alberta, Canada within the **Department of Computing Sciences**. He researches software engineering, mining software repositories, software process recovery and green mining (the study of software change versus software power consumption).

- **A general overview of my Research Record DBLP**
- **Current Version of my Curriculum Vitae**
- **I'd like to thank my numerous collaborators!**

Console | Inspector | Debugger | Style Editor | Profiler | Network

| ✓ | Method | File | Domain | Type | Size | 0 ms / 160 ms / 320 ms / 480 ms / 640 ms / 800 ms |
|---|---|---|---|---|---|---|
| ● | GET | SoftwareProcess.es.html | softwareprocess.es | html | 8.34 KB | → 204 ms |
| ● | GET | static.css | softwareprocess.es | css | 4.08 KB | → 100 ms |
| ● | GET | abram-smiling-profile.png | softwareprocess.es | png | 24.86 KB | → 412 ms |
| ● | GET | web-at-softwareprocess-do... | softwareprocess.es | png | 3.74 KB | → 169 ms |
| ● | GET | whiteback.png | softwareprocess.es | png | 2.21 KB | → 241 ms |
| ● | GET | gears.png | softwareprocess.es | png | 17.66 KB | → 338 ms |
| ● | GET | whitegrad.png | softwareprocess.es | png | 0.37 KB | → 228 ms |

All | HTML | CSS | JS | XHR | Fonts | Images | Media | Flash

7 requests, 61.29 KB, 0.72 s | Clear

SoftwareProcess.es Wiki – Abram Hindle: SoftwareProcess.es – Nightly

SoftwareProcess.es W...

# Refresh

softwareprocess.es/static/SoftwareProcess.es.html

Google

## Research Promotion Site of Abram Hindle

Abram is an assistant professor at the **University of Alberta**, in Edmonton, Alberta, Canada within the **Department of Computing Sciences**. He researches software engineering, mining software repositories, software process recovery and green mining (the study of software change versus software power consumption).

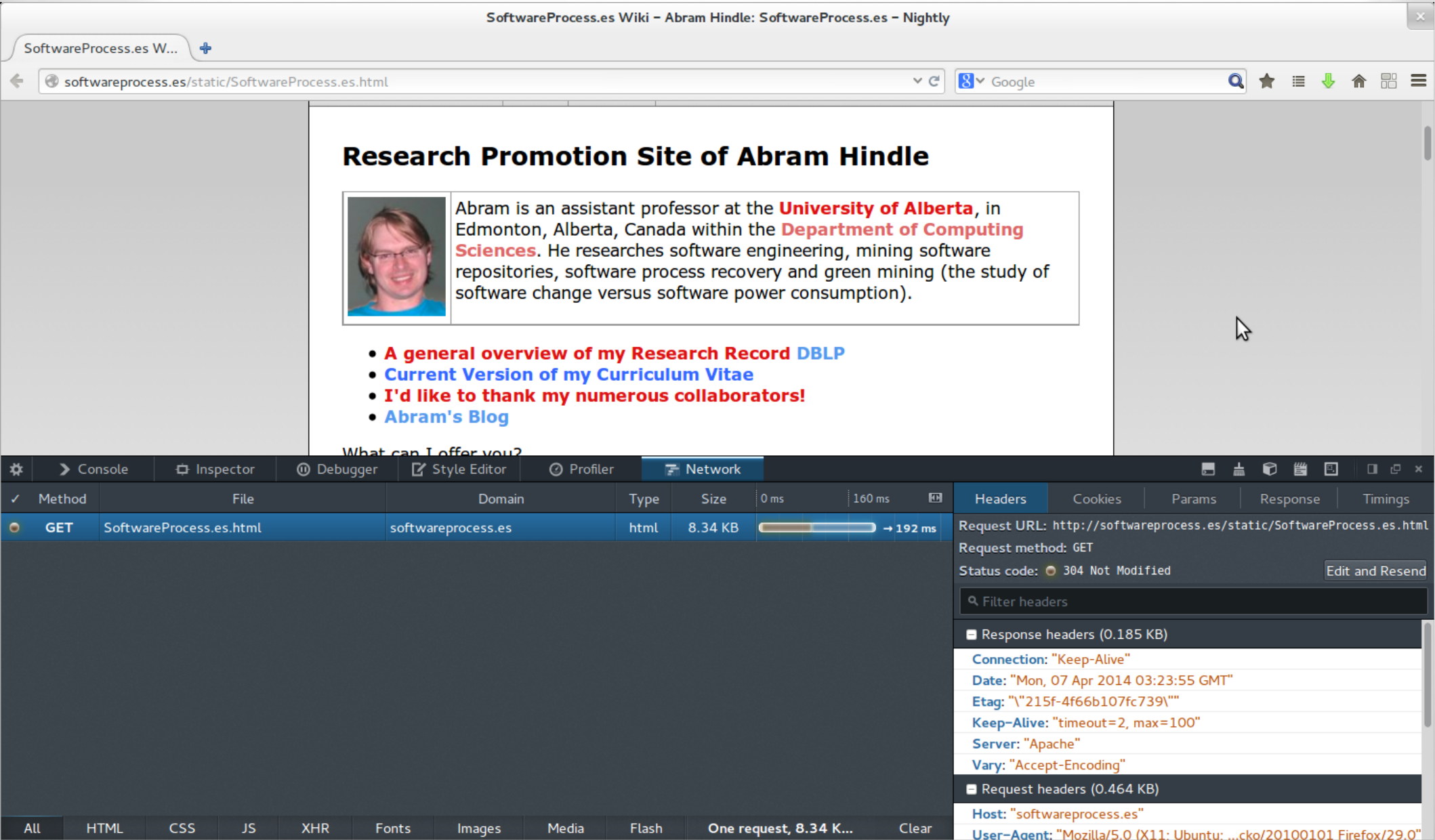
Console | Inspector | Debugger | Style Editor | Profiler | Network

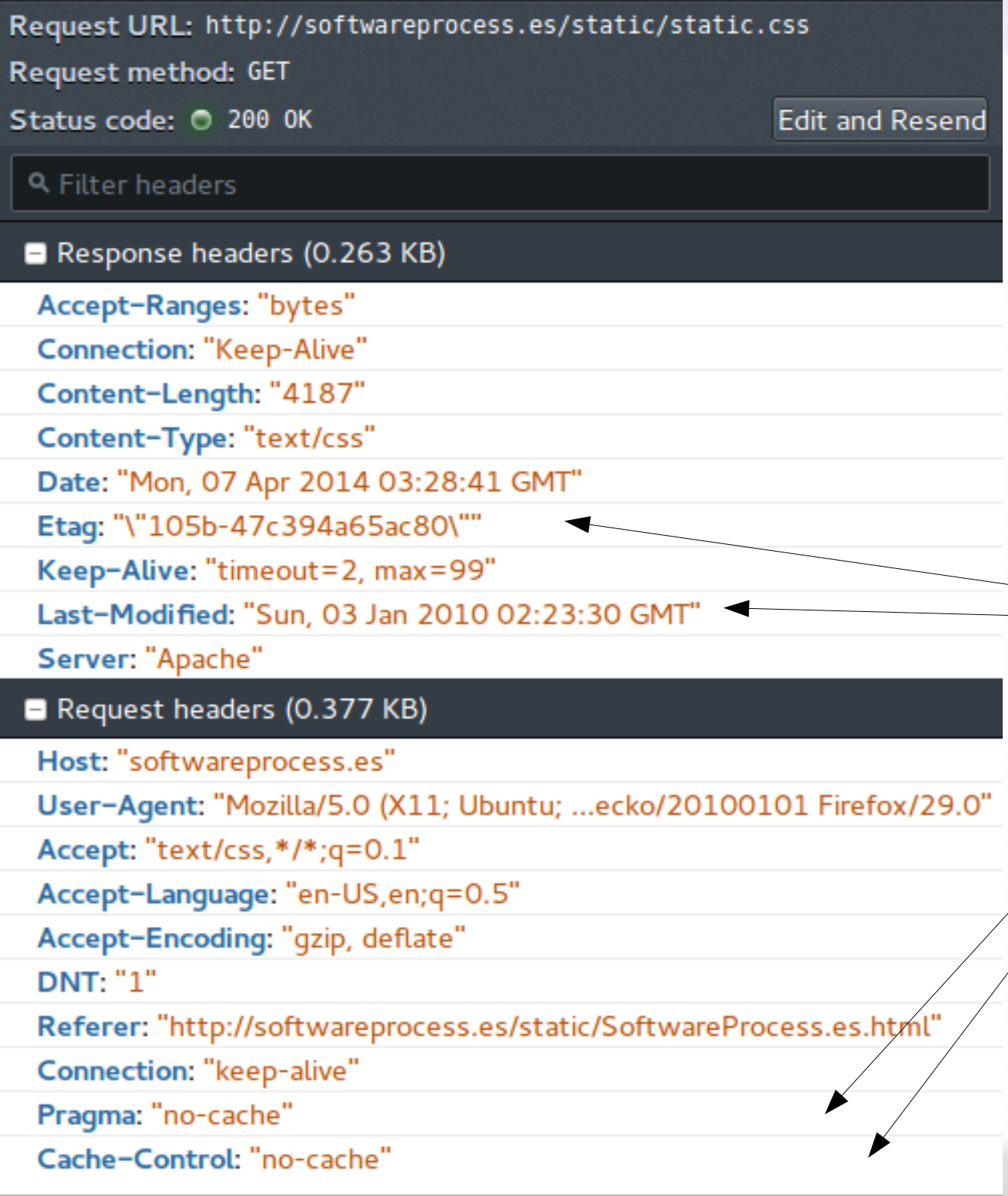
| ✓ | Method | File | Domain | Type | Size | 0 ms / 160 ms / 320 ms / 480 ms |
|---|---|---|---|---|---|---|
| ● | GET | SoftwareProcess.es.html | softwareprocess.es | html | 8.34 KB | → 175 ms |
| ● | GET | static.css | softwareprocess.es | css | 4.08 KB | → 106 ms |
| ● | GET | abram-smiling-profile.png | softwareprocess.es | png | 24.86 KB | → 98 ms |
| ● | GET | web-at-softwareprocess-do... | softwareprocess.es | png | 3.74 KB | → 94 ms |
| ● | GET | whiteback.png | softwareprocess.es | png | 2.21 KB | → 91 ms |
| ● | GET | gears.png | softwareprocess.es | png | 17.66 KB | → 98 ms |
| ● | GET | whitegrad.png | softwareprocess.es | png | 0.37 KB | → 88 ms |

All | HTML | CSS | JS | XHR | Fonts | Images | Media | Flash

7 requests, 61.29 KB, 0.49 s | Clear

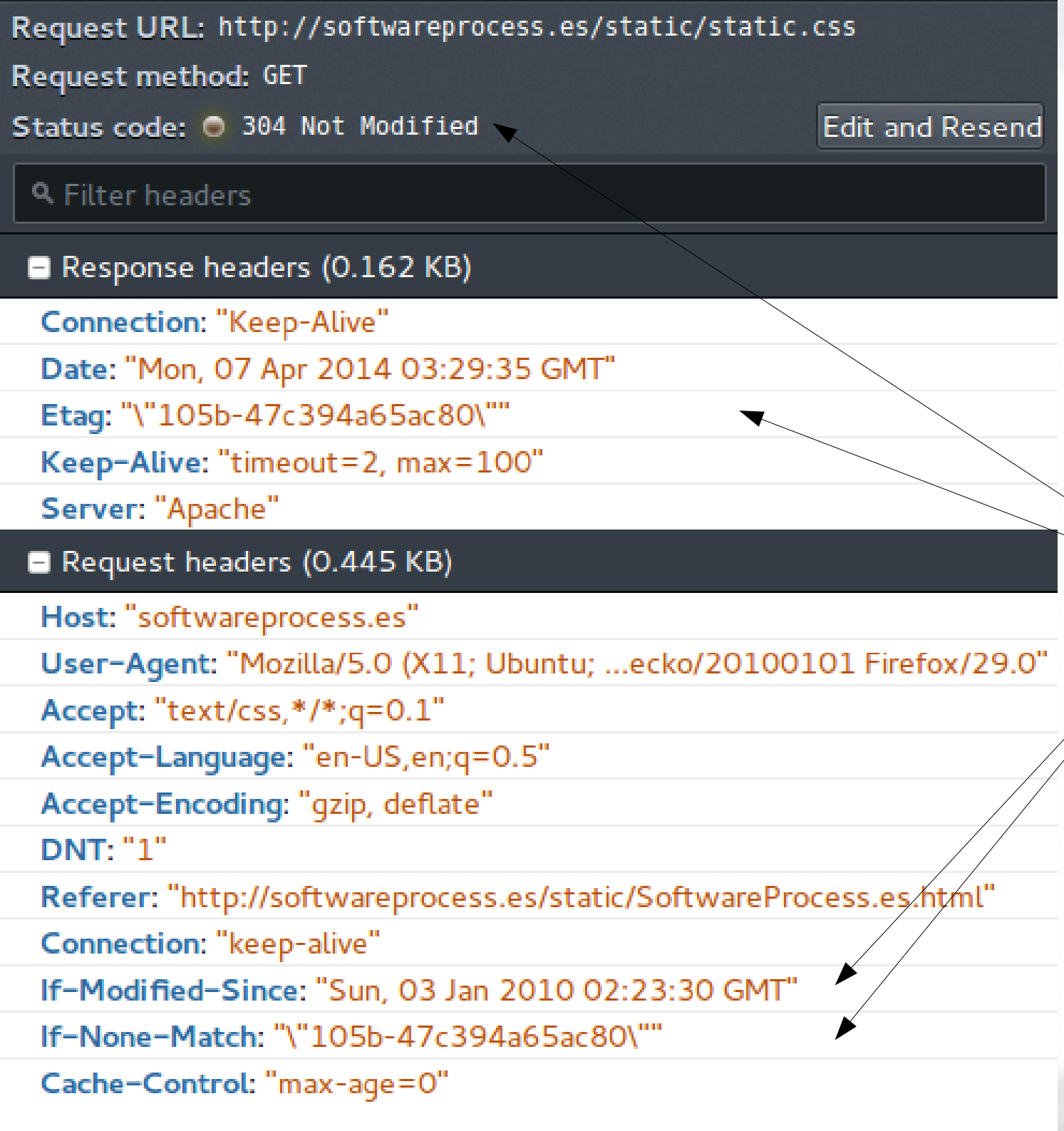
# If I just type in the URL again






# New

Cache Headers




# Refresh

Cache Headers

# User Agent (Browser) Cache

- *Also, if the response does have a Last-Modified time, the heuristic expiration value SHOULD be no more than some fraction of the interval since that time. A typical setting of this fraction might be 10%.* – RFC2616 section 13
- This means if it was modified 10 minutes ago, you should probably hit it up again in a minute.
  - Where as if it was modified 100 days ago, you should get a new version 10 days from now.

# User Agents (Browser) Cache

- Thus it is up to the browser to emit a request
- They do so upon expiry or last modified time heuristic
- Or the user forces a refresh
  - CTRL-SHIFT-R or ctrl-shit click on the refresh button
- In browser cache is the most local and high performance cache!

# Cache-control

- Generally sent by User Agent
- Indicates how they want to handle this request
- It signals proxies and caches how to handle the request

# Cache-Control: no-cache

- You must revalidate
  - We didn't give it a time
- A 304 response is fine
- Forces a request out to the server
- max-age=0 means the same thing

# Cache-Control: no-store

- **Don't store anything**
- **Suggests that the results are not-cacheable and emphemeral.**
- **Will not act as DRM**

# Cache-Control: max-age=

- **Cache-Control: max-age=seconds in a HTTP response tells the Use-Agent the maximum age they should let this resource last**
- Easy to deploy
- Cache-Control: max-age=259200
  - 3 Days
- Benefit: no date math for you!
- Benefit: No date formatting!
- Disadvantage: Have to predict max-age!

# Response Header: Expires

- **Expires tells the Use-Agent after which date they should ask for a new instance of the resource.**
- **Easiest to deploy**
- **Very simple**
- **Causes lots of problems if set wrong!**
- `Expires: Mon, 07 Apr 2014 03:00:05 GMT`

# Request Header: If-Modified-Since

- **Conditional HTTP Request**
- **Return a 304 if not modified since**
- `If-Modified-Since: Mon, 07 Apr 2014 03:00:05 GMT`
  - `Don't send me anything new unless the resource has been modified after that time.`
- `If the response is anything but a 200 OK, return a normal response instead of the 304`
-

# Response Header: Last-Modified

- *The Last-Modified entity-header field value is often used as a cache validator. In simple terms, a cache entry is considered to be valid if the entity has not been modified since the Last-Modified value.* – HTTP RFC 2616 Fielding et al. http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.3.1

# Response Header: Last-Modified

- **Last-Modified is a date that the resource was last modified**
- **Used for simple caching**
- **Requires the HTTP server to respond**

# HTTP ETag

- **What if you cannot guess or estimate the time that content will be safe?**
- **What if content updates all the time, unpredictably?**
- **What if content updates but all changes aren't that important:**
  - **E.g. your age does increase every second but maybe it isn't important to caching to have your age updated per each second?**

# HTTP ETag

- How do you make it?
- If strong (exact content) just use a hash like SHA1
- If weak then hash some content you think is relevant and prefix with W/”etagvalue” to indicate it is a weak hash
- If hashing is pointless make an etag of actual values in plaintext
- Keep it short

# ETags: Entity Tags!

- HTTP Response Header
- Contains a name or tag indicating the content or revision of a resource.
  - Is not date related
  - Tends to be content related
  - Can be any value
  - Can use any hash

# ETags: If-None-Match

- **HTTP Request Header that makes the request conditional.**
- **If any of the provided e-tags match send us back a 304 status code, otherwise send us the resource!**
- **If-None-Match: “someetag”**
- **If-None-Match: “*” // rely on the date stuff, not etags**
- **If-None-Match: “etag1”, “etag2”**

# ETags: If-Match

- HTTP Request Header that makes the request conditional.
- Used in updates to ensure the wrong version is not being updates (like a revision id)
- Provide “*” or an ETag
  - “*” means anything (e.g. you probably have it or it might not exist, but you're just checking)
- If-Match: “someetag”
- If-Match: “*”
- If-Match: “etag1”, “etag2”

# Dangers of the Etag

- Cookies part II
  - Etags allow for vendors (advertisers) to finger print your client because your client will send the etags back.
  - If you deny cookies, you tend to send etags.
  - AOL, Spotify, GigaOm, Etsy, KISSmetrics sued over undeletable tracking cookies http://www.extremetech.com/internet/91966-aol-spotify-gigaom-etsy-kissmetrics-sued-over-undeletable-tracking-cookies
  - Ayenson, Mika, et al. "Flash cookies and privacy II: Now with HTML5 and ETag respawning." World Wide Web Internet And Web Information Systems (2011). ftp://peramides.com.ar/SSRN-id1898390.pdf

# Dangers of the Etag

- Too much information
  - Some Etags contain irrelevant information!
  - What if the browser reboots and the Etags are lost?
    - If the browser/user-agent had a timing guarantee this wouldn't be a problem

# Performance, the Cross Cutting Concern

- **Performance is a cross cutting concern because it interacts with other functionality:**
  - Security
    - Lack of encryption means global proxying
    - Authentication can limit caching
    - Authentication can imply state
  - State
    - State can limit caching
    - State can limit layering

# Round Trips

- DNS Lookups, Connections, HTTP transactions
- Async is fast: Just send it! Who cares when it arrives
- Rountrips are synchronous and slow: We must wait for a response!
  - Avoid HTTP Redirects that aren't cacheable
    - Rewrite Server Side
  - Avoid too many HTTP hosts
  - Can you piggyback?
  - CSS Sprites are often recommended to reduce number of image requests
    - Avoid CSS imports

# Round Trip Tricks

- **Use multiple static content hostnames:**
  - **Take a hit in DNS lookup**
  - **But improve parallel download performance**
  - **Static hosts should not be dynamic and have stable IPs**

# Reduce Request Size

- **Giant Cookies – NOOOOO**
- **Giant URIs -- NOOOO**
- **Too many headers? NOOOO**
- **Remember all that networking we went over?**
  - **Try to fit within the MTU!**

# Avoid Dynamism and Cookies for Static Content

- For static content, do GETs to get it
  - For static content avoid dynamism and cookies
  - Cookies imply state and can mess up caching
- Use seperate domains for static content to avoid statefulness

# Minimize Resource Size

- Images – too big
- Javascript – minify (I dislike this one)
- GZIP Encoding!
- Sound – too big
- Video – too big
  - You can fake Sound and Video in JS!

# Minimize Number of Resources

- 1 or 0 CSS Files
- 1 or 0 Javascript Files
- 1 or 0 Images (CSS Sprites)
- 1 or 0 HTML Files
  - You could generate a page in JS and take no hit.
- 1 giant page has the problem if it is dynamic, but if it is 1 giant page that does dynamic things you can cache that page and never have to get HTML/JS/CSS again.

# Optimize Rendering

- Recommendations:
  - CSS at the top
  - Javascript at the bottom
  - Content in the middle
- Give appropriate sizes and hints
  - The layouter is quite expensive, give some hints and it will go to town.

# Defer Javascript

- **Nasty trick:**
  - **Encode javascript as a string and eval it later when you need it.**
  - **Avoids heavy page loading and JS parsing with the browser is working hard.**
  - **CPU driven**

# Content Delivery Networks

- **These are global networks of content providers that can mirror your content.**
- **Excellent latency and locality**
- **Great for static stuff**
- **$$$**

# GZIP It!

- **Turn on GZIP encoding to make things smallers**
  - **Watch out! This affects latency**
  - **Improves bandwidth**
  - **Balancing act**
  - **Generally recommended**

# Fix DNS

- Use A and AAAA records instead of CNAMEs
- Have your authoritative name server give the results
- Allow for caching of DNS requests to avoid excessive lookups
- Provide many A records in 1 response
- Reduce number of hosts on 1 page
- Some recommend using CNAME to allow multiple connections (so it depends!)

# Avoid Indirection

- Redirections
- Imports
- Dynamic choosing of content
- Javascript downloading of content

# Avoid POST

- POST is not idempotent
- POST is not cacheable
- POST is dynamic
- POST smashes all the performance infrastructure in a fine powder and blows it into your face.

# Check for Errors

- 403s, 404s, 410s, etc. Are all slow
  - Often the server works harder to serve an error than it does to serve real content.
  - Errors cause exceptions, exceptions cause latency and pain and reporting.
  - Errors cause logging – more IO
  - Errors are worthless requests hogging up resources

# Encoding!

- Sending a JSON encoding is not always appropriate in terms of size.
- Sending an audio stream as ASCII text? Bad idea. Use an appropriate format.
- Do you need lossy or losseless?
- Do you need XML? JSON? CSV? Binary?

# Repeat yourself or don't repeat yourself

- **Cache matters if the cache can avoid you repeating yourself then go for it.**
- **But sometimes denormalizing data and providing duplicate information avoids more requests.**

# Async over Sync

- Asynchronous means that you can do other things while something is occurring.
- Synchronous is blocking which implies latency.
- Synchronous means round trips
- Async means parallelizable
- Synchronous means serialized

# Javascript Includes...

- Some people like to include common libraries from the library homepage.
  - Benefit: Someone else has done this so the user has cached it.
  - Disadvantage: What if they get hacked
  - Disadvantage: What if they are slower than you?
  - Disadvantage: What if you lose locality?

# Resources

- RFC 2616 Section 13
  http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html
- RFC 2616 Section 14
  http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html
- Web Performance Best Practices
  https://developers.google.com/speed/docs/best-practices/rules_intro

# Resources

- HTTP Caching
  https://developers.google.com/speed/articles/caching
- Best Practices for Speeding Up Your Web Site
  http://developer.yahoo.com/performance/rules.html
- Browser Cache Matters
  http://yuiblog.com/blog/2007/01/04/performance-research-part-2/

# Resources

- Chapter 10 — Improving Web Services Performance
  http://msdn.microsoft.com/en-us/library/ff647786.aspx
- 19 Tuning Web Services
  http://docs.oracle.com/cd/E24329_01/web.1211/e24390/webservicestune.htm