FS19 WED1

WEB PERFORMANCE

Prof. Dr. Thomas Bocek
thomas.bocek@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Web Engineering - In the *News*

- **17.04.2019: Bad bots now make up 20 percent of web traffic**
  - Bots, in general, are estimated to make up roughly 37.9 percent of all Internet traffic
  - Amazon is the leading ISP for bad bot traffic origins. In total, 18 percent of bad bot traffic came from the firm's services
  - Almost 50 percent of bad bots use Google Chrome as their user agent
  - 74% are “sophisticated” bots
    - An example of this is mouse mimicry, in which the bot is able to simulate mouse events a genuine visitor may perform on a website domain
  - 53.4 percent of bad bot traffic came from the US

- **19.04.2019: Google AMP lowered our page speed, and there's no choice but to use it**
  - Accelerated Mobile Pages (AMP) Project
    - Website publishing technology by Google as a competitor to Facebook's Instant Articles
    - Goal: load website in less than 1 second
    - Enforcing small assets (CSS max 50 KB) for faster loading, subset of HTML with own additions. Own JavaScript is not allowed. Ad format predefined.
    - Most AMP pages are delivered by Google's AMP cache
  - Controversial: Kill Google AMP before it kills the web
  - “AMP can make your site slower”
  - mean time to first byte for the AMP page is 1005ms, and for non-AMP it’s 989ms
  - page is visually complete for AMP is 2166ms, and for non-AMP it’s 1955ms
  - “AMP isn’t about speed. It’s about control”

Exercises

- **From VSS**
  - ost.ch → nic.ch
  - mhs.ch – who can spot a serious issue?
- **Exercises**
  - Work on your project
  - Next week: testat review in the exercises

mhs.ch
Genau mit uns.

Geschätzte Kundin, geschätzter Kunde

Ihr Ticket - **INC#29113188** - wurde geschlossen.

Falls Sie dieses Ticket erneut öffnen möchten, antworten Sie bitte auf diese Mail.

Sofern wir Ihr Anliegen klären konnten, bitten wir Sie, nach Schliessung des Tickets nicht mehr darauf zu antworten.

Besten Dank für Ihr Verständnis!

Freundliche Grüsse
Ihre mhs internet AG

Ticket ID: INC#29113188
Betreff: Support-Anfrage
Abteilung: Support

Ihre Ticketnummer lautet INC#29113188. Bitte belassen Sie diese Nummer zwecks Rückverfolgung und korrekter Zuweisung bei einer Antwort in der Betreffszeile.

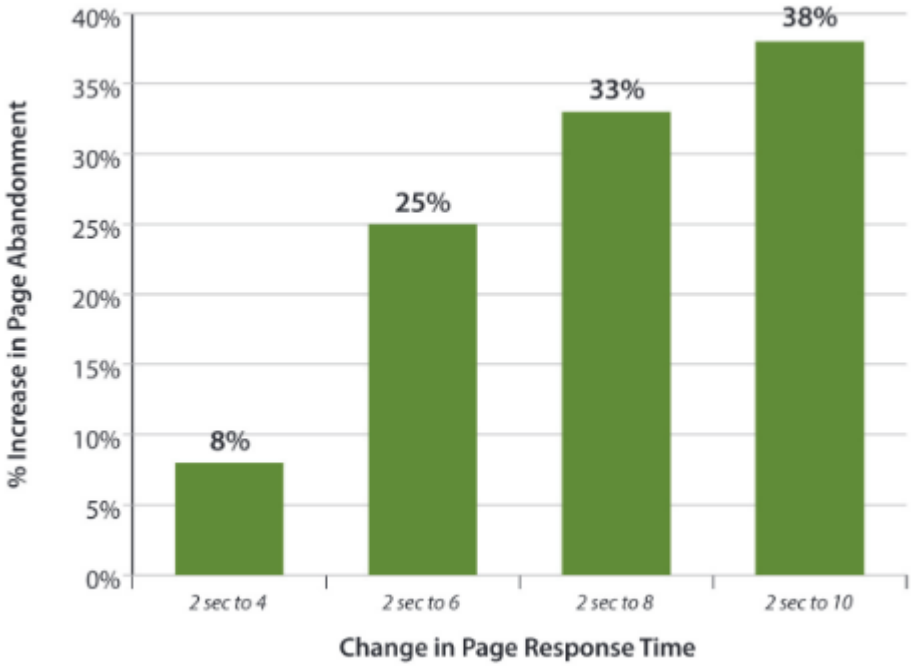
URL: b.socrative.com/student/

Room: HSR

Why Web Performance

- Pinterest 2015
  - Improved mobile web home landing page performance by 60% → Mobile signup conversion rate by 40%
- Pinterest 2016
  - 40% decrease in Pinner wait time → 15% increase in conversion rate to signup
- **“At the BBC we’ve noticed that, for every additional second a page takes to load, 10 per cent of users leave.” [source]**
- **Housing.com, 30% faster page load → 38% more conversions [source]**
- **Many infographics here or here or here or here**
  - “over 25% of visitors will leave a website that doesn’t load in six seconds”

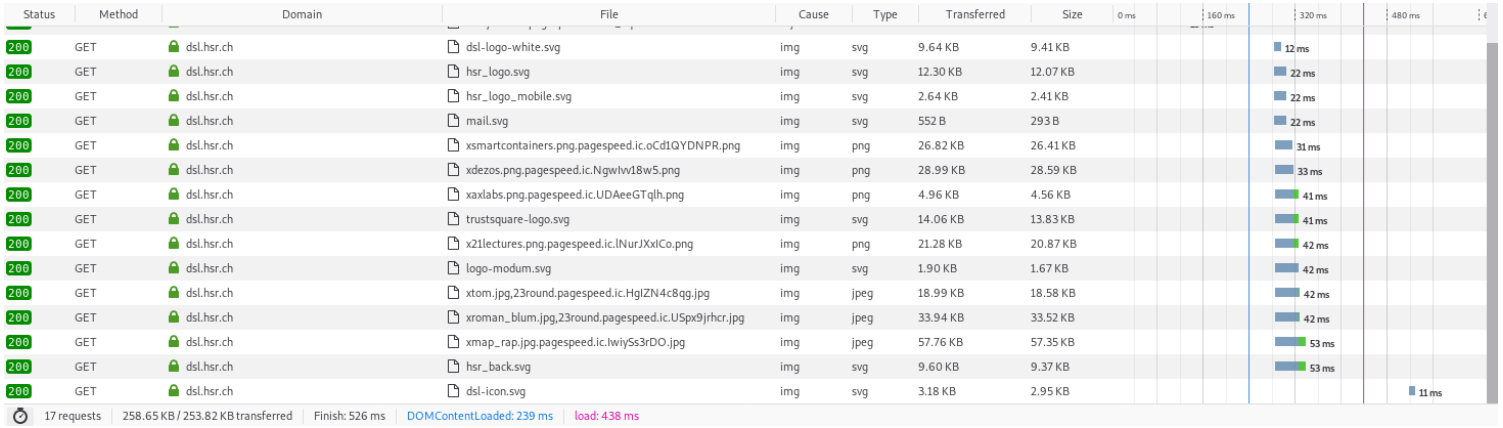
- **Why Performance is Neglected**
  - Lack of ownership
  - Uncertainty where to start
  - Unclear expectations about success



http://www.mcrinc.com/Documents/Newsletters/201110_why_web_performance_matters.pdf

Reduce the size of your page

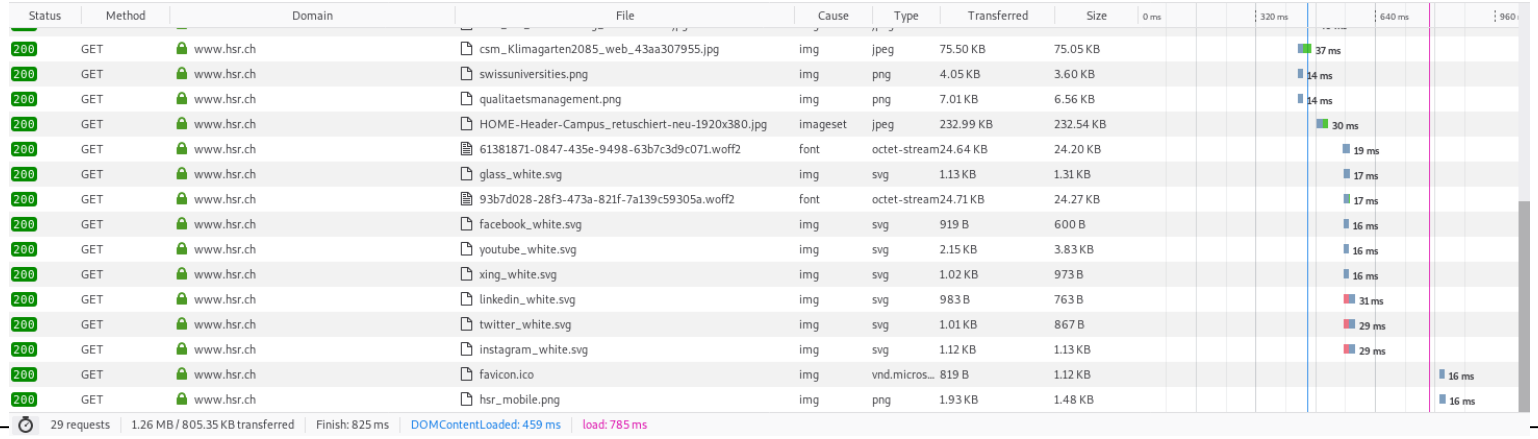
- **Measure, measure, measure**
  - Simple measurements a good start
  - https://hsr.ch vs. https://dsl.hsr.ch
  - 29 requests vs. 17 requests
  - 805KB vs. 254KB
  - ~825ms vs. ~548ms



Status	Method	Domain	File	Cause	Type	Transferred	Size
200	GET	dsl.hsr.ch	dsl-logo-white.svg	img	svg	9.64 KB	9.41 KB
200	GET	dsl.hsr.ch	hsr_logo.svg	img	svg	12.30 KB	12.07 KB
200	GET	dsl.hsr.ch	hsr_logo_mobile.svg	img	svg	2.64 KB	2.41 KB
200	GET	dsl.hsr.ch	mail.svg	img	svg	552 B	293 B
200	GET	dsl.hsr.ch	xsmartcontainers.png.pagespeed.ic.oCd1QYDNPR.png	img	png	26.82 KB	26.41 KB
200	GET	dsl.hsr.ch	xdezos.png.pagespeed.ic.NgwIvv18w5.png	img	png	28.99 KB	28.59 KB
200	GET	dsl.hsr.ch	xaxlabs.png.pagespeed.ic.UDAeeGTqlh.png	img	png	4.96 KB	4.56 KB
200	GET	dsl.hsr.ch	trustsquare-logo.svg	img	svg	14.06 KB	13.83 KB
200	GET	dsl.hsr.ch	x21lectures.png.pagespeed.ic.INurJXxICo.png	img	png	21.28 KB	20.87 KB
200	GET	dsl.hsr.ch	logo-modum.svg	img	svg	1.90 KB	1.67 KB
200	GET	dsl.hsr.ch	xtom.jpg,23round.pagespeed.ic.HgIZN4c8qg.jpg	img	jpeg	18.99 KB	18.58 KB
200	GET	dsl.hsr.ch	xroman_blum.jpg,23round.pagespeed.ic.USpx9jrhcr.jpg	img	jpeg	33.94 KB	33.52 KB
200	GET	dsl.hsr.ch	xmap_rap.jpg.pagespeed.ic.IwiySs3rDO.jpg	img	jpeg	57.76 KB	57.35 KB
200	GET	dsl.hsr.ch	hsr_back.svg	img	svg	9.60 KB	9.37 KB
200	GET	dsl.hsr.ch	dsl-icon.svg	img	svg	3.18 KB	2.95 KB

17 requests | 258.65 KB / 253.82 KB transferred | Finish: 526 ms | DOMContentLoaded: 239 ms | load: 438 ms

- **SVG minification, from inkscape: https://dsl.hsr.ch/lect/wed1/hsr_back_1.svg**
  - 19.1KB, lots of meta information, nicely formatted
  - 1:1, minified, 9.6KB
  - Loss of precision (digits in the fractional part) ~9.6KB (attention!)
  - hsr_logo.svg ~12.3KB, precision 1 7.7KB (mobile 2.5KB to 0.3KB)



Status	Method	Domain	File	Cause	Type	Transferred	Size
200	GET	www.hsr.ch	csm_Klimagarten2085_web_43aa307955.jpg	img	jpeg	75.50 KB	75.05 KB
200	GET	www.hsr.ch	swissuniversities.png	img	png	4.05 KB	3.60 KB
200	GET	www.hsr.ch	qualitaetsmanagement.png	img	png	7.01 KB	6.56 KB
200	GET	www.hsr.ch	HOME-Header-Campus_retuschiert-neu-1920x380.jpg	imageset	jpeg	232.99 KB	232.54 KB
200	GET	www.hsr.ch	61381871-0847-435e-9498-63b7c3d9c071.woff2	font	octet-stream	24.64 KB	24.20 KB
200	GET	www.hsr.ch	glass_white.svg	img	svg	1.13 KB	1.31 KB
200	GET	www.hsr.ch	93b7d028-28f3-473a-821f-7a139c59305a.woff2	font	octet-stream	24.71 KB	24.27 KB
200	GET	www.hsr.ch	facebook_white.svg	img	svg	919 B	600 B
200	GET	www.hsr.ch	youtube_white.svg	img	svg	2.15 KB	3.83 KB
200	GET	www.hsr.ch	xing_white.svg	img	svg	1.02 KB	973 B
200	GET	www.hsr.ch	linkedin_white.svg	img	svg	983 B	763 B
200	GET	www.hsr.ch	twitter_white.svg	img	svg	1.01 KB	867 B
200	GET	www.hsr.ch	instagram_white.svg	img	svg	1.12 KB	1.13 KB
200	GET	www.hsr.ch	favicon.ico	img	vnd.micros...	819 B	1.12 KB
200	GET	www.hsr.ch	hsr_mobile.png	img	png	1.93 KB	1.48 KB

29 requests | 1.26 MB / 805.35 KB transferred | Finish: 825 ms | DOMContentLoaded: 459 ms | load: 785 ms

Compression: gzip

- **Reduce size by compression (lossless)**
- **Almost all browsers support gzip**
  - Accept-Encoding: gzip, deflate, br
- **For gzip: zopfli (100%), for br: brotli (90%)**
  - Zopfli 5% smaller than gzip, needs 80 times longer to compress (not suitable for on-the-fly)
- **Brotli vs zopfli**
  - 14% smaller than gzip for JavaScript
  - 21% smaller than gzip for HTML
  - 17% smaller than gzip for CSS

```
2475 Jan  9 14:36 dsl-logo-white.svg.br
2614 May 12 14:12 dsl-logo-white.svg.gz
```

  - brotli -Zkf *.svg → dsl.hsr.ch page: 223 KB

- **Configure Apache/Nginx**

```
location ~ \.svg$ {
    brotli_static on;
}
```

- **Google Logo, smaller than 305 bytes?**

URL: b.socrative.com/student/

Room: HSR

Images

- **Vector vs. Raster images**
  - Vector for images with geometric shapes
  - Raster images, for photos, landscape etc.
  - Vector images, resolution independent
- **Resolution**
  - Raster: the higher the resolution, the higher the size
- **Lossless (png,gif) vs lossy (jpg)**
  - Lossless: recreate the original file exactly
  - Lossy: eliminate "unnecessary" bits, those that you ideally don’t see
- **Reduce number of colors, e.g. dsl-logo-whitebg.png, from 23KB to 13KB**
  - pngquant, reduce from 32bit to 8bit, lossy

Compression levels

Quality 10; 9KB

Quality 20; 13KB

Quality 40; 20KB

JPG encoders

- **MozJPEG vs. Guetzli**
  - Better than default
- **Scaled down, quality 100: 153KB**
  - Guetzli: 29KB (quality 85), slow (not progessive)
  - MozJPG: 41KB (quality 80), fast
- **dsl.hsr.ch: recompress images**
  - map_rap.jpg: 87KB to 49KB
  - roman_blum.jpg: 41KB to 24KB
  - tom.jpg: 25KB to 16KB
  - Total page size: 201KB
- **Recompress logos**
  - Total page size: 129KB
- **BPG (comparison) – unsupported ☹, Patent issues**

- **JPG vs. WebP Image Formats**
  - WebP supported on most browsers
  - “average WebP file size is 25%-34% smaller compared to JPEG file size” [source]
  - JPG: compatibility

File Name	Original Size	Guetzli	Lossy	WebP
test-1.jpg	712 KB	67 KB in 38.30 s	83 KB in 2 s	53 KB in 3 s
test-2.jpg	1.7 MB	231 KB in 1 m 21 s	238 KB in 4 s	254 KB in 3 s
test-3.jpg	2.2 MB	346 KB in 2 m 30 s	416 KB in 4 s	344 KB in 4 s
test-4.jpg	4.6 MB	478 KB in 4 m 46 s	499 KB in 5 s	322 KB in 5 s

https://www.keycdn.com/blog/benchmarking-guetzli

JPG encoders

- **WebP better than PNG?**
  - Not when reducing colors, can reduce also in WebP
  - Zopfli can reduce 5-10% on average PNG
  - Switching to WebP? iOS → not supported

PNG and WebP

Size	Reference	Optimized	
	libvips	Zopfli	WebP
96×54 px	4,5 kB	3,8 kB	2,8 kB
192×108 px	12,8 kB	10,5 kB	7,1 kB
320×180 px	27,9 kB	22,9 kB	15,0 kB
384×216 px	37,9 kB	31,2 kB	20,3 kB
544×306 px	68,7 kB	57,2 kB	37,5 kB
640×360 px	89,7 kB	74,6 kB	48,9 kB
768×432 px	123,5 kB	103,6 kB	67,5 kB
1088×612 px	209,5 kB	171,9 kB	114,6 kB

- **Guetzli better with small images, larger images, WebP**
- **«Better than JPEG» [source]**
  - Guezli creates many candidate output JPEG, selects best

JPEG and WebP

Size	Reference		Optimized	
	JPEG 100	JPEG 88	Guetzli	WebP
96×54 px	8,4 kB	2,3 kB	1,7 kB	2,0 kB
192×108 px	26,5 kB	6,1 kB	5,2 kB	4,9 kB
320×180 px	63,0 kB	13,6 kB	11,9 kB	10,4 kB
384×216 px	85,1 kB	18,1 kB	16,0 kB	13,8 kB
544×306 px	155,6 kB	32,3 kB	28,8 kB	23,7 kB
640×360 px	201,2 kB	41,5 kB	37,0 kB	30,2 kB
768×432 px	274,0 kB	56,0 kB	49,7 kB	28,2 kB
1088×612 px	491,6 kB	99,0 kB	89,2 kB	47,5 kB

https://www.ctrl.blog/entry/webp-vs-guetzli-zopfli.html

URL: b.socrative.com/student/

Room: HSR

HTML5/progressive JPEG

- **Progressive JPEG**
  - See content faster

https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/automating-image-optimization/

- **HTML5: srcset**
  - Responsive images, desktop vs mobile
  - Let the browser decide which image
- **srcset: filename space width, demo**

- **Sizes can be defined for media conditions**
- **low-res/high-res: `images/500x250.png 2x`**

```
<img
  class="srcset-demo-img"
  alt=""
  src="images/2000x1000.png"

  srcset="
    images/500x250.png   500w,
    images/1000x500.png  1000w,
    images/1500x750.png  1500w,
    images/2000x1000.png 2000w,
    images/3000x1500.png 3000w">
```

- **Support new formats**

```
<picture>
  <source type="image/svg+xml" srcset="pyramid.svg">
  <source type="image/webp" srcset="pyramid.webp">
  <img src="pyramid.png" alt="regular pyramid ">
</picture>
```

Images

- **Animated GIF vs. Video**
  - Use video, example: 14MB to 0.8MB
- **Image optimization checklist**
  - Prefer vector formats
  - Minify and compress SVG assets
  - Pick best raster image format
  - Experiment with optimal quality settings for raster formats
  - Remove unnecessary image metadata
  - Serve scaled images
  - Automate, automate, automate

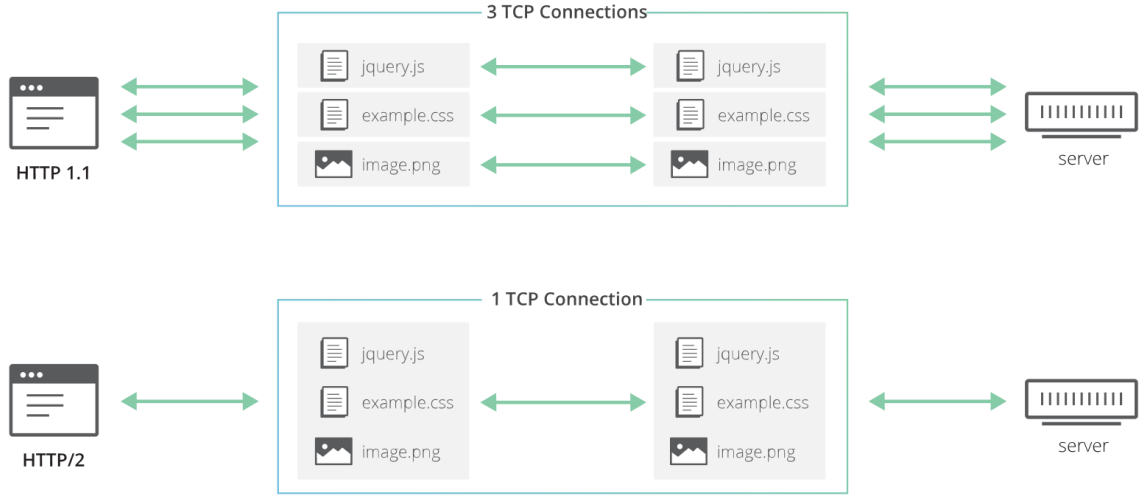
“run your JPEGs through MozJPEG (q=80 or lower is fine for web content) and consider Progressive JPEG support, PNGs through pngquant and SVGs through SVGO. Explicitly strip out metadata (--strip for pngquant) to avoid bloat. Instead of crazy huge animated GIFs, deliver H.264 videos (or WebM for Chrome, Firefox and Opera)! If you can't at least use Giflossy. If you can spare the extra CPU cycles, need higher-than-web-average quality and are okay with slow encode times: try Guetzli.” [source]

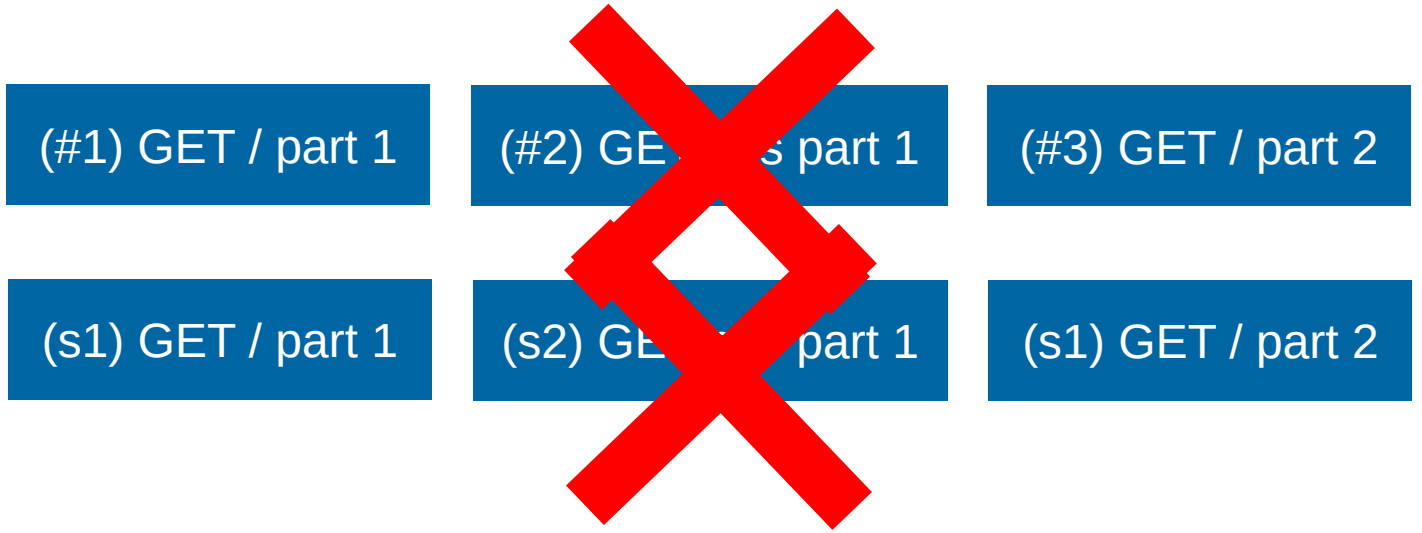
URL: b.socrative.com/student/

Room: HSR

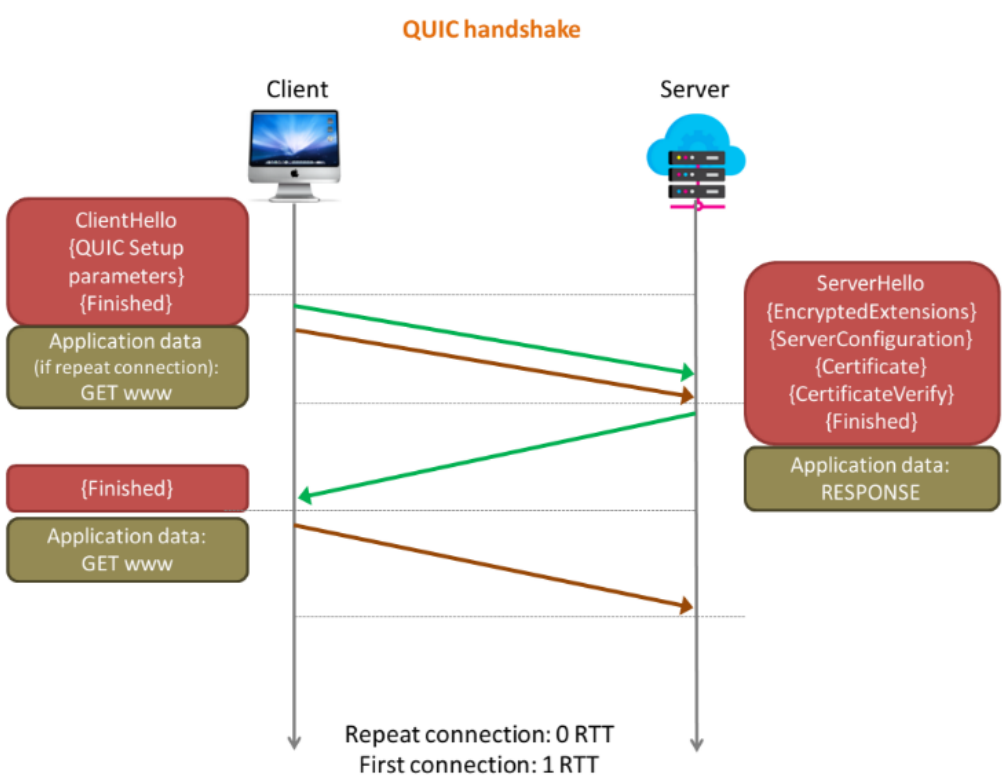
HTTP/3 vs. HTTP/2 vs. HTTP/1

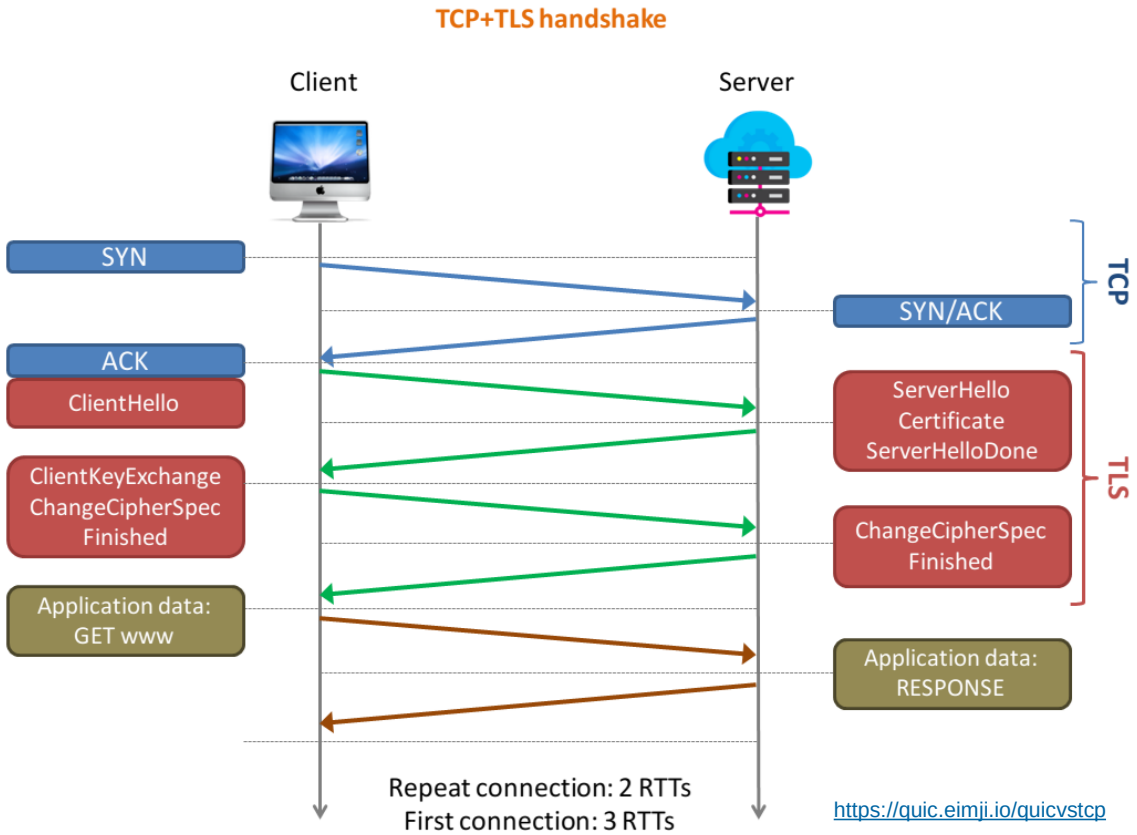
- **Head-of-line blocking**
  - HTTP/1 → HTTP/2
    - Initial handshake
    - Initial congestion window ramp up
    - Servers limit concurrent connections
  - One packet loss, TCP needs to be ordered, everything stops
  - QUIC (HTTP/3) can multiplex requests: one stream does not affect others
- **QUIC also reduces RTT**
  - From TCP/TLS 3 RTT to 1 RTT
  - TLS also help reducing 1 RTT





HTTP/3 vs. HTTP/2 vs. HTTP/1





Minification

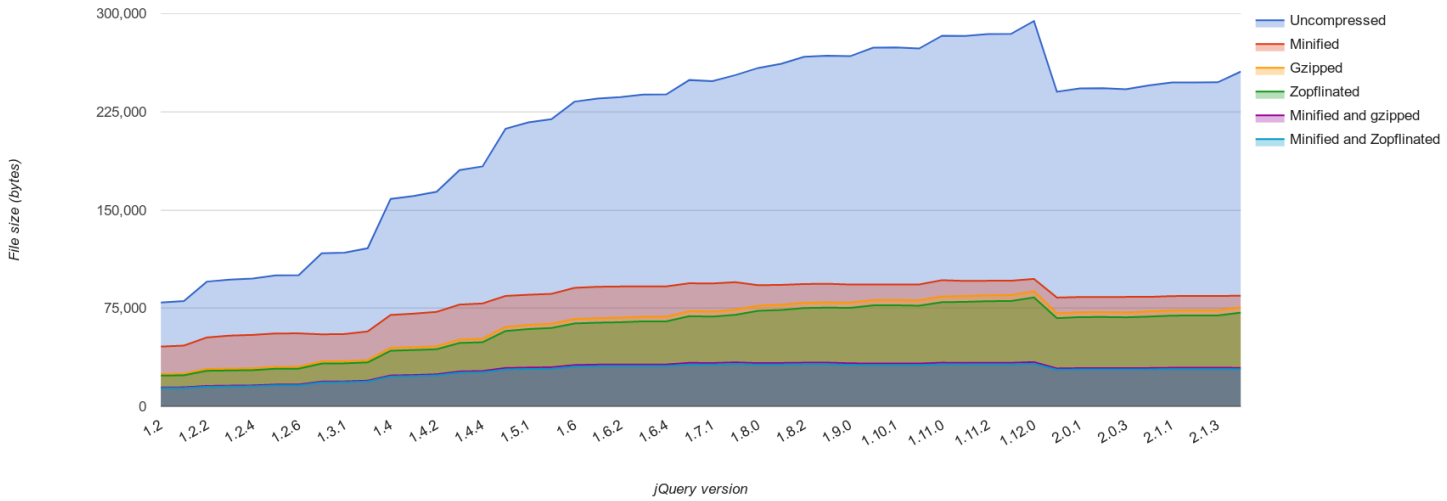
- **HTML, CSS, JS, SVG minification, bundling**

```
var array = [];
for (var i = 0; i < 20; i++) {
  array[i] = i;
}
```

```
for(var a=[i=0];++i<20;a[i]=i);
```

- **Quicker download times, reduced bandwidth consumption**
- **Reduced number of HTTP requests when combining many JavaScript files into one compressed file**
- **Comments and whitespace not needed for JavaScript execution, names can be shortened**

- **Minify and gzip/zopfli**
  - dsl.hsr.ch uses pagespeed, minifies on the fly with ngxpagespeed
- **Lazy loading: defers loading of non-critical resources**
  - E.g., on scrolling



https://mathiasbynens.be/demo/jquery-size

Minification

- **Example: https://github.com/tbocek/WED3-SPA/tree/master/components**
  - Mode: 'development', run: webpack
    - 406’869 Mai 19 18:19 dist/app.js
    - 96’298 Mai 19 18:22 app.js.gz
  - Mode: ‘production’, run: webpack
    - 126’698 Mai 19 18:27 app.js
    - 44’791 Mai 19 18:31 app.js.gz
- **Minification**
  - Many minification libraries
    - Check your tools to squeeze every bit out of your bundle
    - 110KB vs120KB – does it matter?
- **CSS minification – comparison**
  - Webpack – cssnano
  - crass – slow, but does ordering

```
b, c, a {
    third: rgba(255, 255, 255, 0.9);
    second: abc;
    first: 50%;
}
```

```
a, b, c {
    first: 50%;
    second: abc;
    third: hsla(0, 0%, 100%, 0.9);
}
```

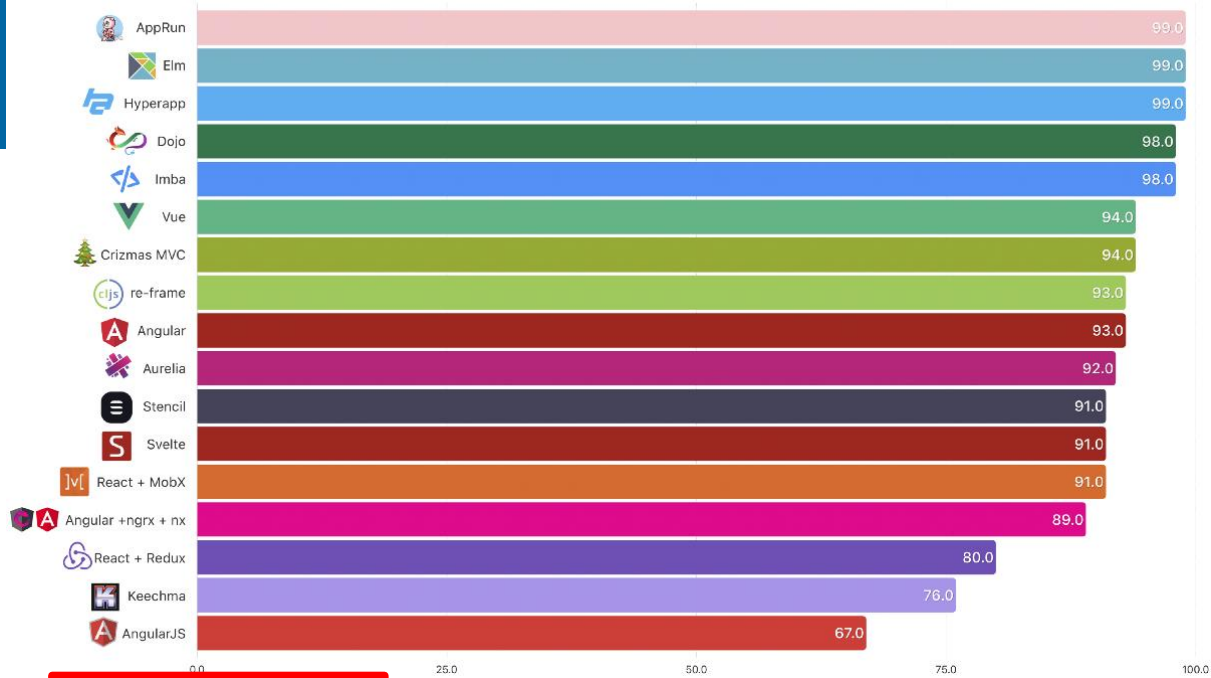
  - Much better for compression – gzip, zopfli
- **Test go-based minification: https://github.com/tdewolff/minify/releases**
  - 5772 for 960.css, 120’753 bootstrap, 47ms

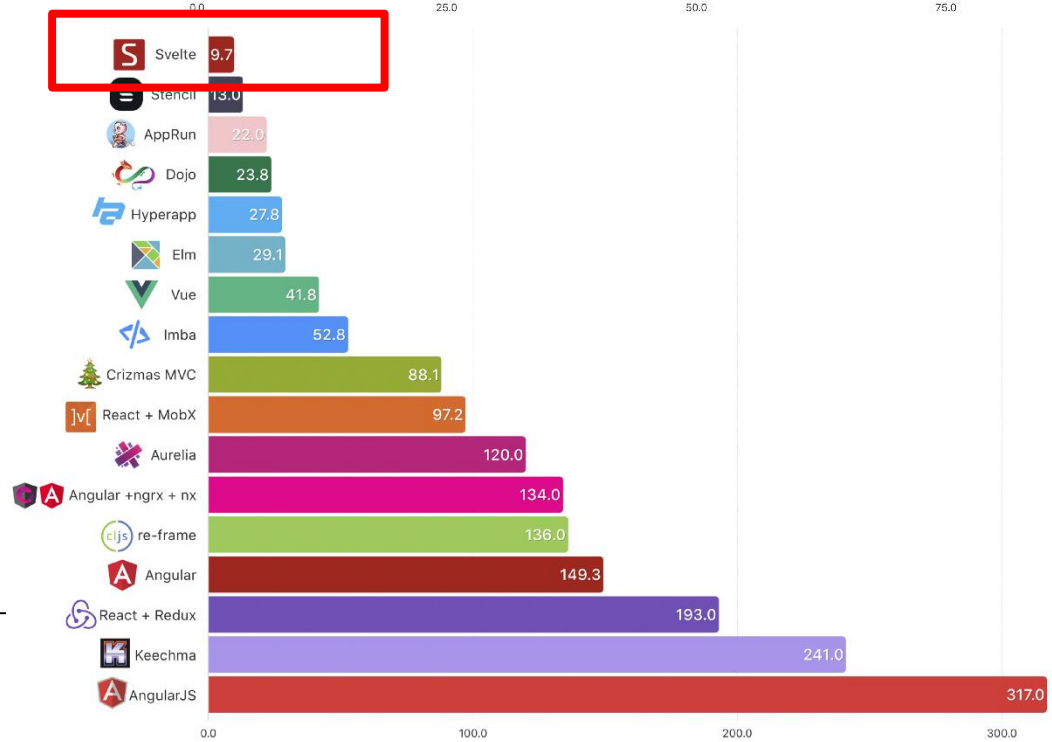
URL: b.socrative.com/student/

Room: HSR

Svelte 3

- **WED3: in the news - 08.04.2019: A Real World Comparison of Front-End Frameworks with Benchmarks (2019 update)**
  - Performance according to Lighthouse Scoring Guide
  - Size and LoC comparison
- **Svelte with 9.7KB LoC. Impressive!**
- **“Svelte is a radical new approach to building user interfaces. Whereas traditional frameworks like React and Vue do the bulk of their work in the browser, Svelte shifts that work into a compile step that happens when you build your app.”**
- **By choosing the right framework, you can save bandwidth, but you loose flexibility**
  - Not the only factor: popularity, community





Vue.js vs Svelte – Looks similar - Ticker.vue vs Ticker.svelte

```
<style>
    /* nice background color */
    body {
        background-color:#f0f0f0;
    }
</style>

<script>
    export default {
        props: ['symbol'],
        data: function () {
            return {
                coinmarketcapData: {}
            }
        },
        created: function () {
            fetch('https://api.coinmarketcap.com/v1/ticker/?limit=10')
                .then(response => response.json())
                .then(body => {
                    this.coinmarketcapData = body.reduce(function(map, obj) {
                        map[obj.symbol] = obj;
                        return map;
                    }, {});
                });
        },
        computed: {
            price: function() {
                if (typeof this.coinmarketcapData[this.symbol] !== 'undefined') {
                    return this.coinmarketcapData[this.symbol].name + " " +
                           this.coinmarketcapData[this.symbol].price_usd;
                }
            }
        }
    }
</script>

<template>
    <p>{{price}} USD</p>
</template>
```

```
<style>
    /* nice background color */
    body {
        background-color:#f0f0f0;
    }
</style>

<script>
    import { onMount } from 'svelte';
    export let symbol;
    let coinmarketcapData = {};
    let price;

    onMount(async () => {
        const res = await fetch('https://api.coinmarketcap.com/v1/ticker/?limit=10');
        let body = await res.json();
        coinmarketcapData = body.reduce(function(map, obj) {
            map[obj.symbol] = obj;
                return map;
            }, {});
        price = calcPrice();
    });

    function calcPrice() {
        if (typeof coinmarketcapData[symbol] !== 'undefined') {
            return coinmarketcapData[symbol].name + " " + coinmarketcapData[symbol].price_usd;
        }
    }
</script>

<p>{price} USD</p>
```

- Source: WED3-SPA

Vue.js vs Svelte – Significantly smaller

- Vue.js
  - Mode: ‘production’, run: webpack
  - 126’698 Mai 19 18:27 app.js
  - 44’791 Mai 19 18:31 app.js.gz
- Svelte
  - rollup -c --environment INCLUDE_DEPS,BUILD:production
  - Only small functionality, comparison not fair
  - 3’183 Mai 20 16:05 bundle.js
  - 1’505 Mai 20 16:06 bundle.js.gz
  - 1’386 Mai 20 16:06 bundle.js.br

- **No recent performance measurements**

Name	vanillajs-keyed	vue-v2.5.16-keyed	svelte-v2.9.7-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed
create rows Duration for creating 1000 rows after the page loaded.	126.7 ± 4.2 (1.0)	182.1 ± 7.6 (1.4)	220.4 ± 4.9 (1.7)	185.2 ± 10.2 (1.5)	180.5 ± 7.3 (1.4)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	134.6 ± 2.3 (1.0)	158.8 ± 2.7 (1.2)	231.6 ± 3.3 (1.7)	161.2 ± 2.7 (1.2)	157.3 ± 2.0 (1.2)
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	65.1 ± 2.6 (1.0)	156.4 ± 9.8 (2.4)	75.1 ± 3.8 (1.2)	68.8 ± 3.7 (1.1)	81.9 ± 2.7 (1.3)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	11.0 ± 2.3 (1.0)	10.6 ± 2.0 (1.0)	10.7 ± 1.0 (1.0)	7.9 ± 4.3 (1.0)	10.3 ± 2.1 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	17.5 ± 6.7 (1.0)	20.0 ± 2.9 (1.1)	20.4 ± 4.4 (1.2)	105.8 ± 1.8 (6.1)	106.5 ± 1.9 (6.1)
remove row Duration to remove a row. (with 5 warmup iterations).	46.1 ± 0.9 (1.0)	54.2 ± 2.2 (1.2)	48.2 ± 1.0 (1.0)	47.1 ± 3.0 (1.0)	49.6 ± 0.8 (1.1)
create many rows Duration to create 10,000 rows	1,229.1 ± 39.7 (1.0)	1,603.2 ± 34.8 (1.3)	2,376.0 ± 40.7 (1.9)	1,693.9 ± 70.1 (1.4)	1,935.4 ± 33.6 (1.6)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	205.6 ± 4.0 (1.0)	342.5 ± 6.0 (1.7)	354.4 ± 11.8 (1.7)	243.3 ± 6.3 (1.2)	268.6 ± 6.9 (1.3)
clear rows Duration to clear the table filled with 10.000 rows.	131.4 ± 3.9 (1.0)	191.9 ± 6.1 (1.5)	183.5 ± 4.1 (1.4)	263.9 ± 3.0 (2.0)	175.4 ± 4.1 (1.3)
slowdown geometric mean	**1.00**	**1.37**	**1.39**	**1.50**	**1.50**

Name	vanillajs-keyed	vue-v2.5.16-keyed	svelte-v2.9.7-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	1,877.7 ± 0.3 (1.0)	2,252.7 ± 0.2 (1.2)	1,877.7 ± 0.7 (1.0)	3,278.5 ± 4.0 (1.7)	2,477.6 ± 0.6 (1.3)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	16.0 ± 0.0 (1.0)	55.1 ± 1.5 (3.4)	16.0 ± 0.0 (1.0)	235.2 ± 8.5 (14.7)	65.6 ± 2.3 (4.1)
main thread work cost total amount of time spent doing work on the main thread. Includes style/layout/etc.	310.1 ± 70.1 (1.1)	420.6 ± 67.5 (1.5)	277.7 ± 4.5 (1.0)	679.3 ± 5.3 (2.4)	466.0 ± 3.9 (1.7)
total byte weight network transfer cost (post-compression) of all the resources loaded into the page.	150,302.0 ± 0.0 (1.0)	215,445.0 ± 0.0 (1.4)	150,230.0 ± 0.0 (1.0)	365,497.0 ± 0.0 (2.4)	251,915.0 ± 0.0 (1.7)

Name	vanillajs-keyed	vue-v2.5.16-keyed	svelte-v2.9.7-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed
ready memory Memory usage after page load.	2.0 ± 0.0 (1.0)	2.7 ± 0.2 (1.4)	2.2 ± 0.1 (1.1)	6.0 ± 0.0 (3.1)	2.8 ± 0.2 (1.4)
run memory Memory usage after adding 1000 rows.	2.3 ± 0.1 (1.0)	7.1 ± 0.0 (3.1)	3.6 ± 0.0 (1.5)	9.8 ± 0.0 (4.2)	6.7 ± 0.0 (2.9)
update each 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	2.6 ± 0.1 (1.0)	7.2 ± 0.0 (2.8)	3.9 ± 0.0 (1.5)	9.8 ± 0.0 (3.8)	7.6 ± 0.0 (3.0)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	2.8 ± 0.1 (1.0)	7.2 ± 0.0 (2.6)	3.9 ± 0.0 (1.4)	10.1 ± 0.0 (3.6)	7.9 ± 0.0 (2.9)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	2.5 ± 0.0 (1.0)	3.0 ± 0.0 (1.2)	2.6 ± 0.0 (1.0)	6.4 ± 0.0 (2.5)	3.8 ± 0.0 (1.5)

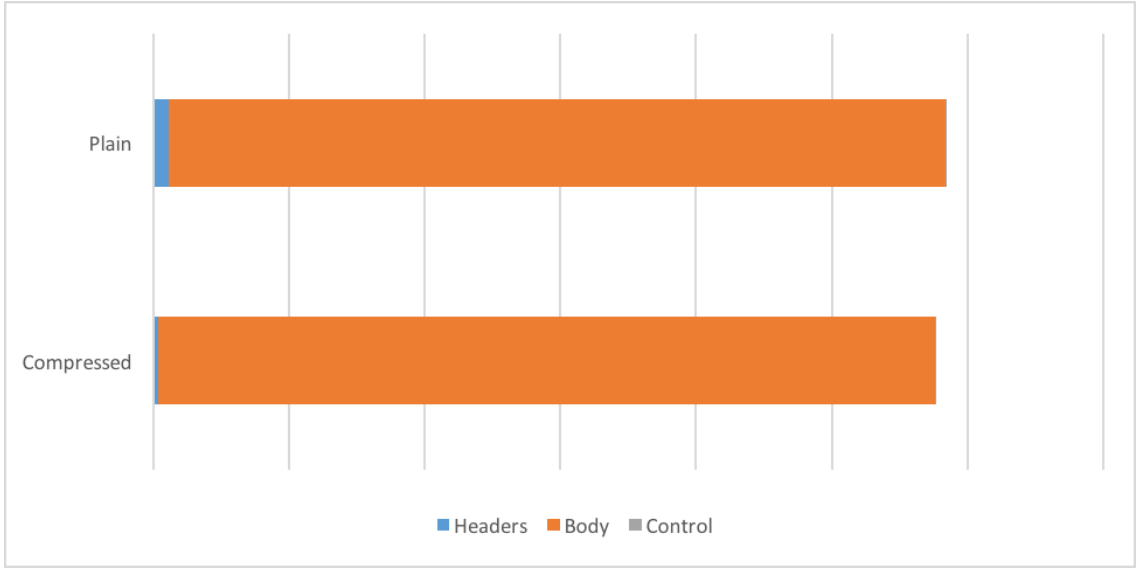
- **Performance**
  - Svelte performance looks ok in v2
  - Benchmark: e.g., “Duration to create 10,000 rows”
- important not to mix "benchmark" and "real life“
- Paginating 1k rows is not terribly realistic, it would be bad UI, list of 10k rows is not real-life
- Bootup time: 16ms vs 235ms
  - Main thread work cost: better than plain JavaScript
- Memory: close to plain JavaScript

URL: b.socrative.com/student/

Room: HSR

HTTP/2 Header Compression

- **HPACK header compression in HTTP/2**
  - Stateful compression, SPDY uses gzip
  - Reference other headers, static table, Huffmann Code
    - Dynamic table: initially empty, updated based on exchanged values within a particular connection
- **Image: ~20KB, with header compression, you save ~100 bytes. Does it matter?**
  - Only header send, you send 1 frame anyway, unless multiplexed
- **Cloudflare: “1.4% savings in the total egress HTTP/2 traffic”**



https://blog.cloudflare.com/hpack-the-silent-killer-feature-of-http-2/

HTTP/2 Server Push

- **Traditional method**

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>hello world</h1>
  <img src="example.png">
</body>
</html>
```

- **GET /**
  - 2 subsequent requests (HTTP overhead):
  - GET /style.css
  - GET /example.png

- **Improvement of the traditional method**

```
<link rel="preload" href="/styles.css" as="style">
<link rel="preload" href="/example.png" as="image">
```

  - Only works if the previous site also had preload, does not work for the initial site

- **Server push**
  - Nginx:

```
location / {
      root   /usr/share/nginx/html;
      index  index.html index.htm;
      http2_push /style.css;
      http2_push /example.png;
   }
```

  - For each request to index.html, style.css and example.png is pushed to the client
  - + No request from the browser

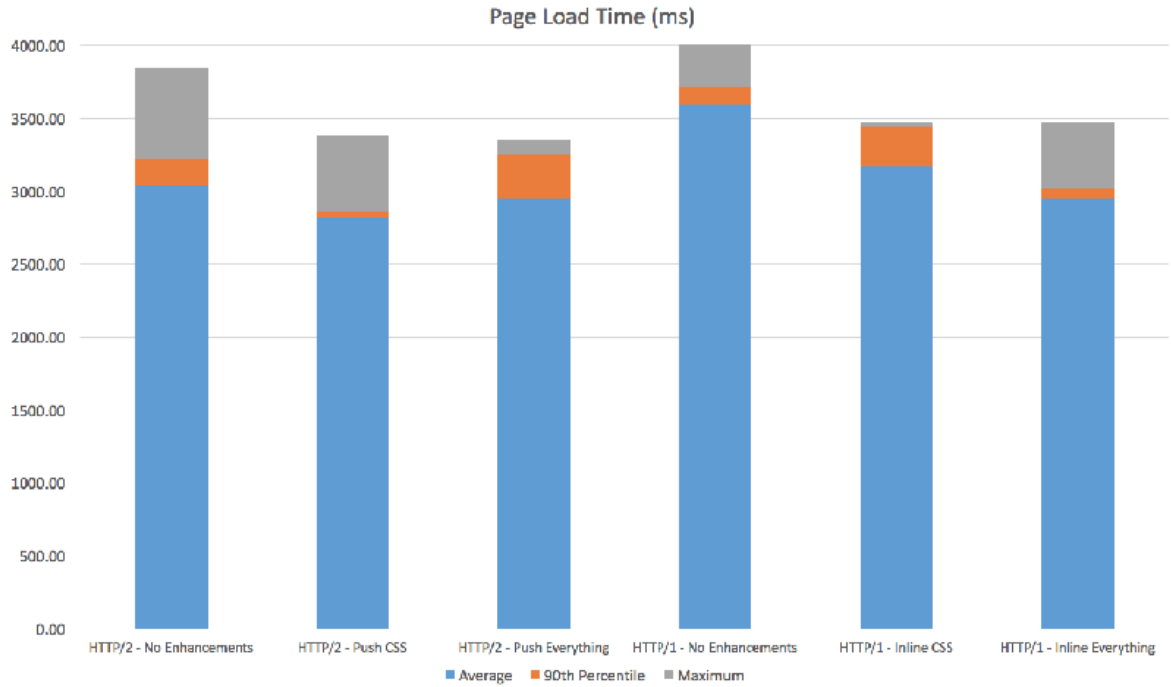
HTTP/2 Server Push

- **Server push**
  - Need to change nginx config
  - Always pushed – no caching?
- **Back-end**
  - User Link header, configure nginx

```
Link: </styles.css>; rel=preload; as=style
Link: </styles.css>; rel=preload; as=style,
</example.png>; rel=preload; as=image
```

```
server {
    listen 443 ssl http2;
    # ...
    root /var/www/html;
    location = / {
        proxy_pass http://upstream;
        http2_push_preload on;
    }
}
```

- **Still caching issue: use cookies**
  - Only use server push if the user never used the site before



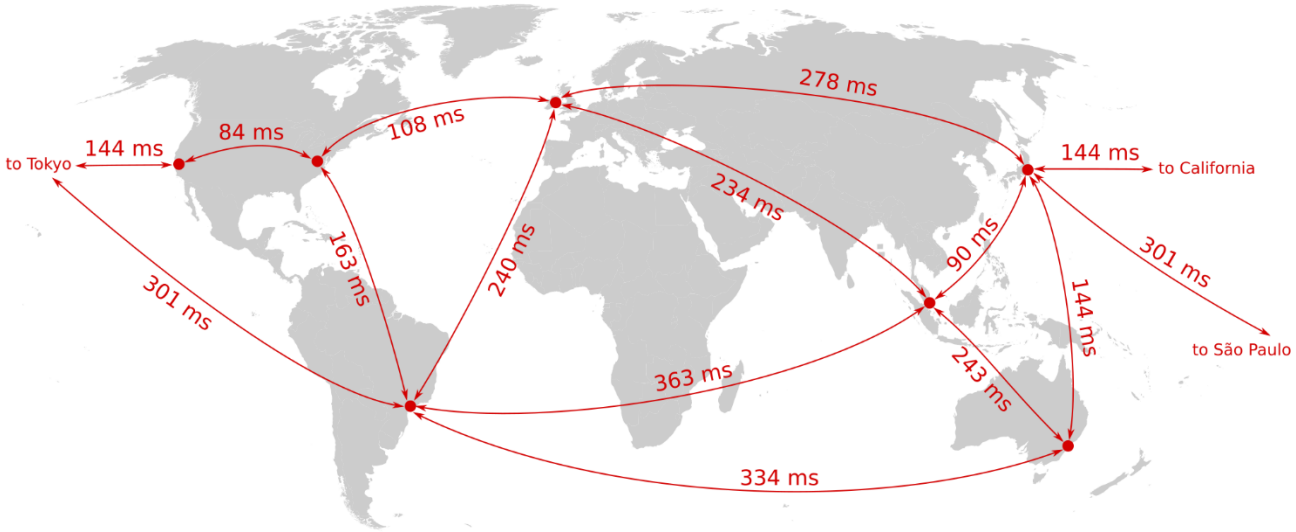
https://www.tutorialdocs.com/article/http2-server-push.html

Caching

- **“A cache is the single most important technology in keeping sites scalable and fast” [source]**
- **Browser caches not shared / local cache / content distribution/delivery network (CDN)**
  - One million requests, one million browser caches to fill
    - But good for repeated requests for static assets, e.g., css or js files
  - Local cache
    - Cache static content, don’t use app server
  - Varnish or Nginx for static content caching?
    - Nginx (if you use it anyway, stick with it), Varnish (more configurable), or both

https://www.inkandswitch.com/local-first.html

- CDN: content that can be served to users without the need to access your servers
- CDN: geographically distributed



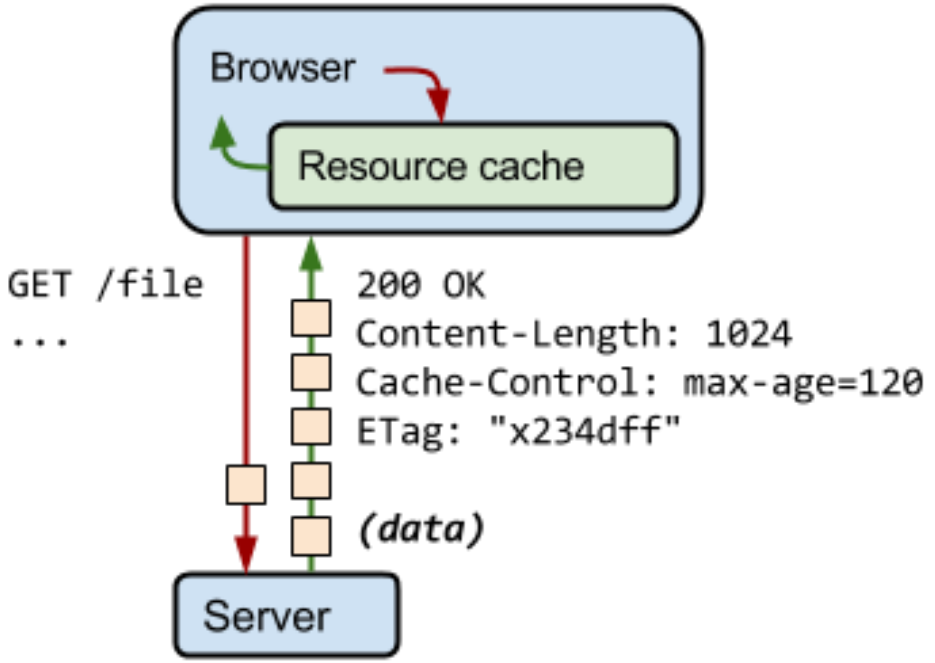
- Pay as you go: Cloudflare, Amazon CloudFront

URL: b.socrative.com/student/

Room: HSR

ETag – Browser Cache

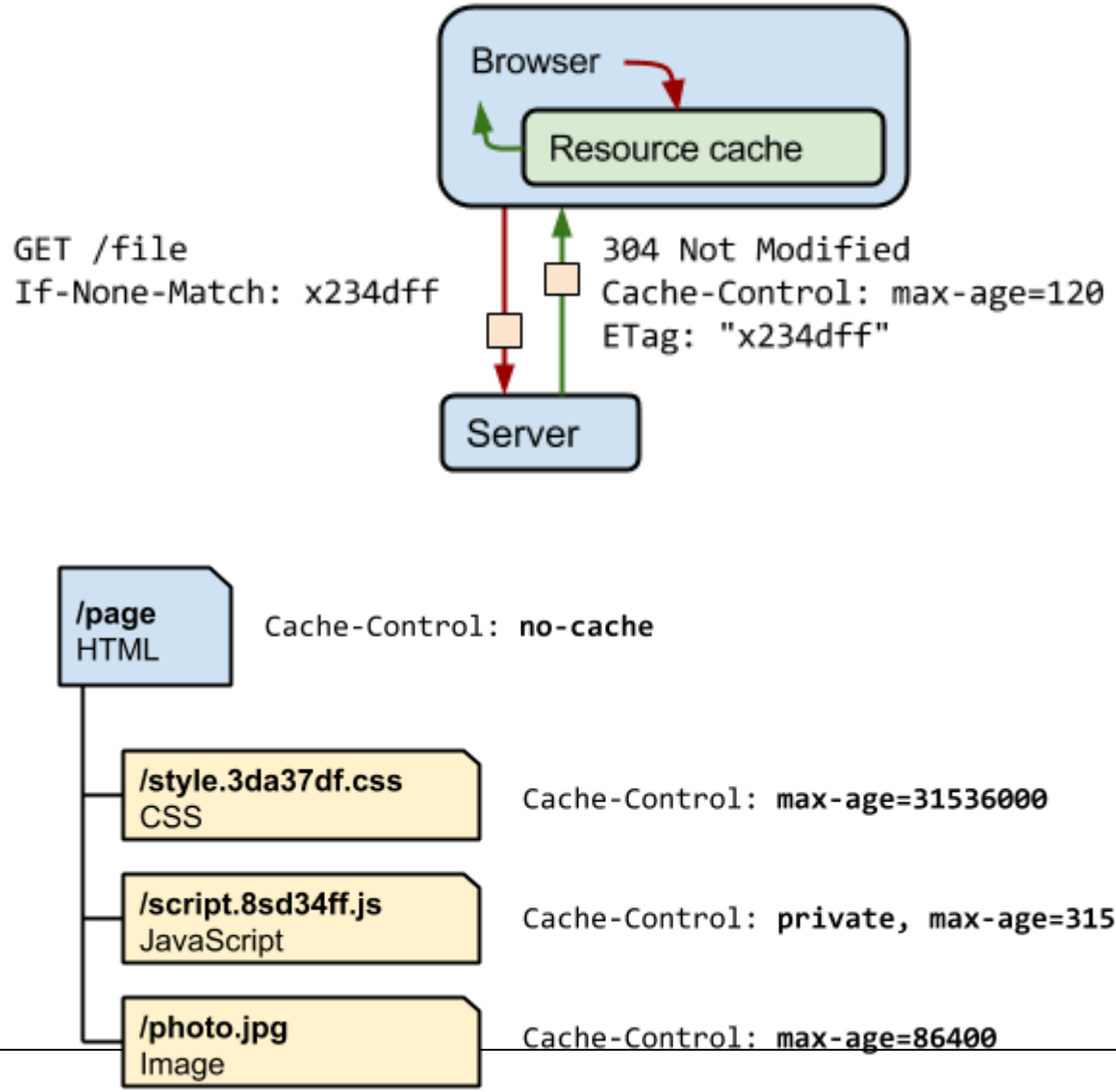
- **Browser cache: ETag**
  - Request: 200 Response with ETag: “x234dff”
  - Browser stores /file with key x234dff
  - Cache-Control: max-age=120
    - maximum time in seconds that the fetched response is allowed to be reused - not going to send request
    - Cache-Control: max-age=2592000, stale-while-revalidate=86400
      - Deliver stale assets, update assets in the background. E.g., for css files.
  - Attention with inlineing header images, caching may be better!



[source]

ETag – Browser Cache

- **After 120 seconds**
  - Browser checks the local cache and finds the previous response expired (>120s)
  - Request If-None-Match: “x234dff”
  - If the asset was not modified
    - 304 Not Modified
    - Asset is not transferred
  - Asset is now in local cache for 120s
    - After that 304 if not modified
    - Or the modified asset is sent
  - "no-cache" and "no-store“
    - No-cache: never use local cache, always check with ETag if cache is valid
    - No-store: never store any data
  - Unique URLs
    - Long caching time



[source]

URL: b.socrative.com/student/

Room: HSR

Questions?

Prof. Dr. Thomas Bocek
thomas.bocek@hsr.ch