SOASTA

Philip Tellis
@bluesmoon

Nic Jansma
@nicj

https://github.com/SOASTA/boomerang

http://www.soasta.com/mpulse

http://slideshare.net/nicjansma/measuring-real-user-performance-in-the-browser

Measuring Real User Performance in the Browser

2016-09-20

#velocityconf NY 2016

http://slideshare.net/nicjansma/measuring-real-user-performance-in-the-browser

Agenda

- Real User Measurement
- Browser Performance APIs
- Visual Experience
- Beaconing
- Single Page Apps
- Continuity
- Nixing Noise

Abbé Jean-Antoine Nollet

1700 - 1770

French Clergyman & Budding Electrician

Invented one of the first Electroscopes

(we now call them beacon collectors)

L'Abbé Jean Antoine Nollet — Maurice Quentin de La Tour

Alte Pinakothek, Munich, Germany

In 1746, he conducted the first ever RUM Experiment

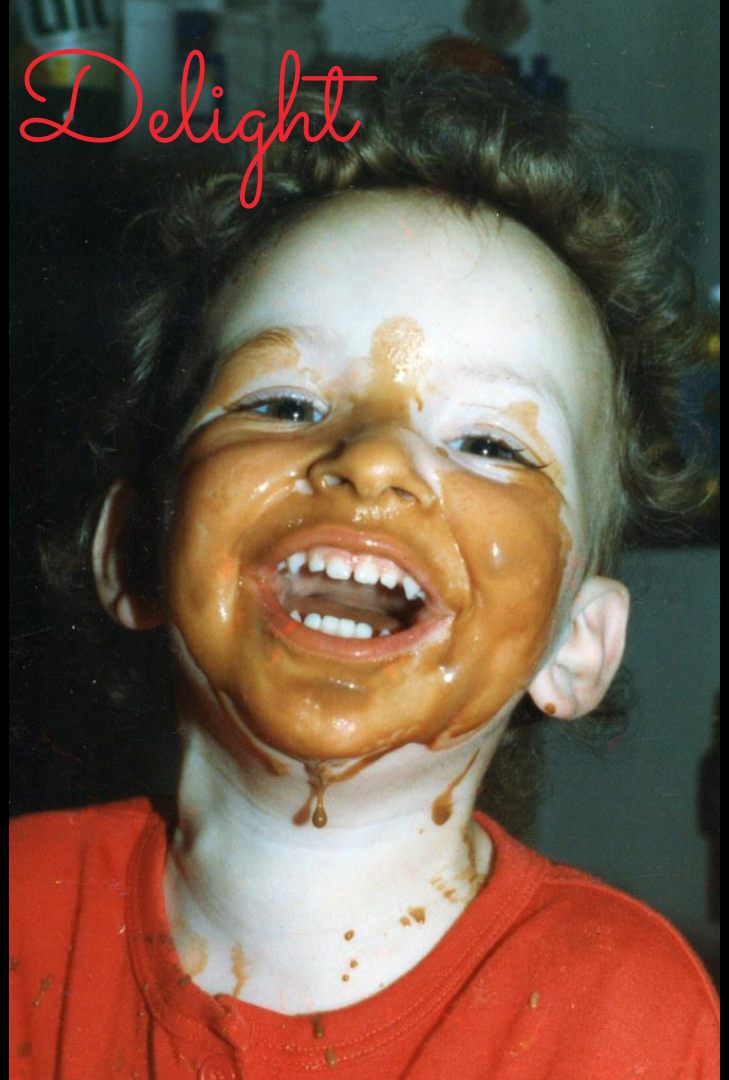
He shot an electric current through 200 monks, and checked how quickly they jumped; thereby measuring the latency of an electric signal with...

Real Users!

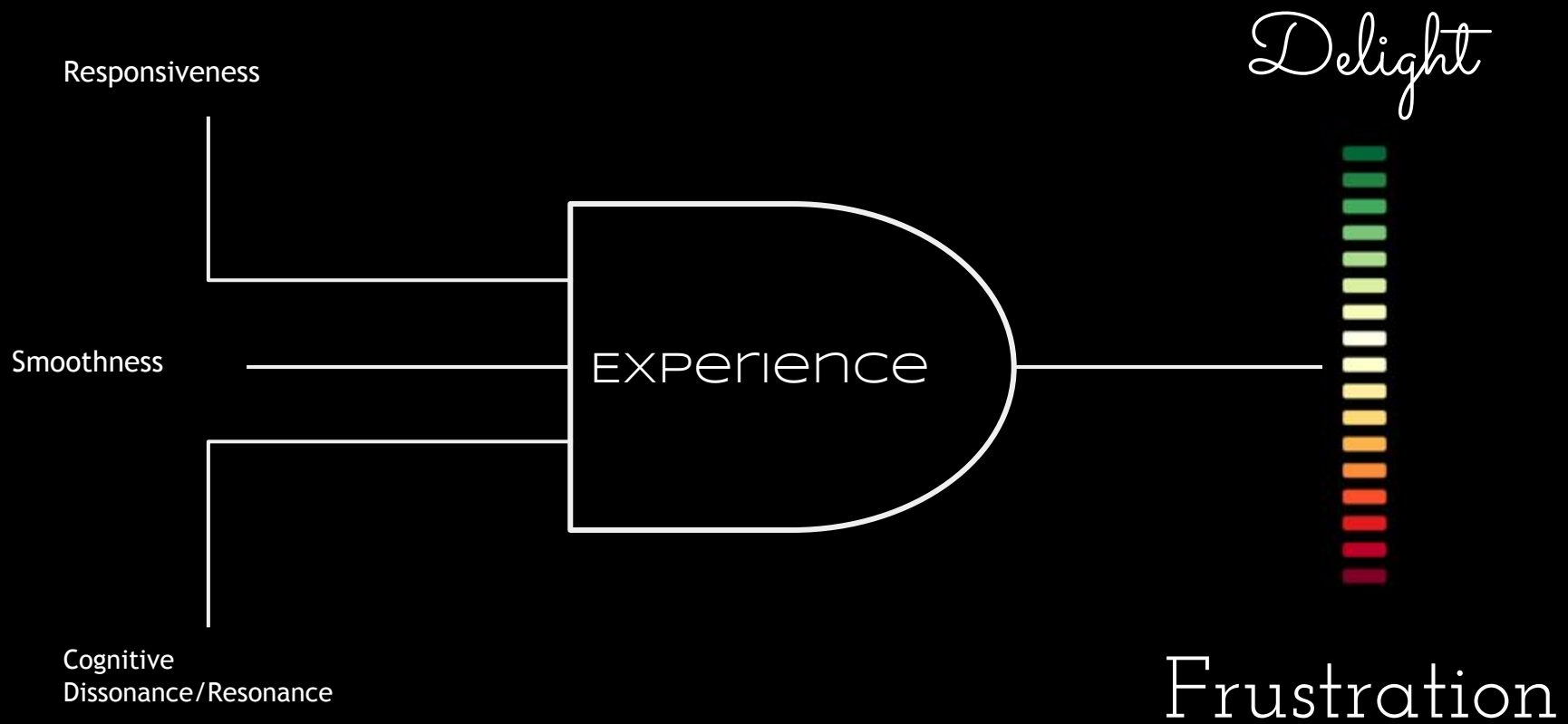
Fortunately, our methods have gotten far less intrusive...

but first...

Why do we care?



Or


Responsiveness
Smoothness
Cognitive
Dissonance/Resonance
experience
Delight
Frustration

ok then...

How do we do it?

Performance aware Browser APIs

- High Resolution Time: Better `Date.now()`
- Navigation Timing (NT): Page load timings
- Performance Timeline: Access NT/RT/UT from one API
- Resource Timing (RT): Resource load timings
- User Timing (UT): Custom site events and measurements
- Page Visibility: Visibility state of the document
- Timing control for script-based animations: `requestAnimationFrame()`
- Efficient Script Yielding: `setImmediate()`
- Resource Hints: dns-prefetch, preconnect, prefetch, prerender
- Preload: Mandatory high-priority fetch for current navigation
- Cooperative Scheduling of Background Tasks: `requestIdleCallback()`
- Beacon: `sendBeacon()`

DOMHighResTimeStamp

High-resolution, monotonically non-decreasing clock

	Date	DOMHighResTimeStamp
Accessed via	`(new Date).getTime()`	`performance.now()`
Start	Unix epoch	`navigationStart`
Monotonically non-decreasing	No	Yes
Affected by user's clock	Yes	No
Resolution	Millisecond	Sub-millisecond
Example	`1420147524606`	`3392.275999998674`

https://w3c.github.io/hr-time/

DOMHighResTimeStamp

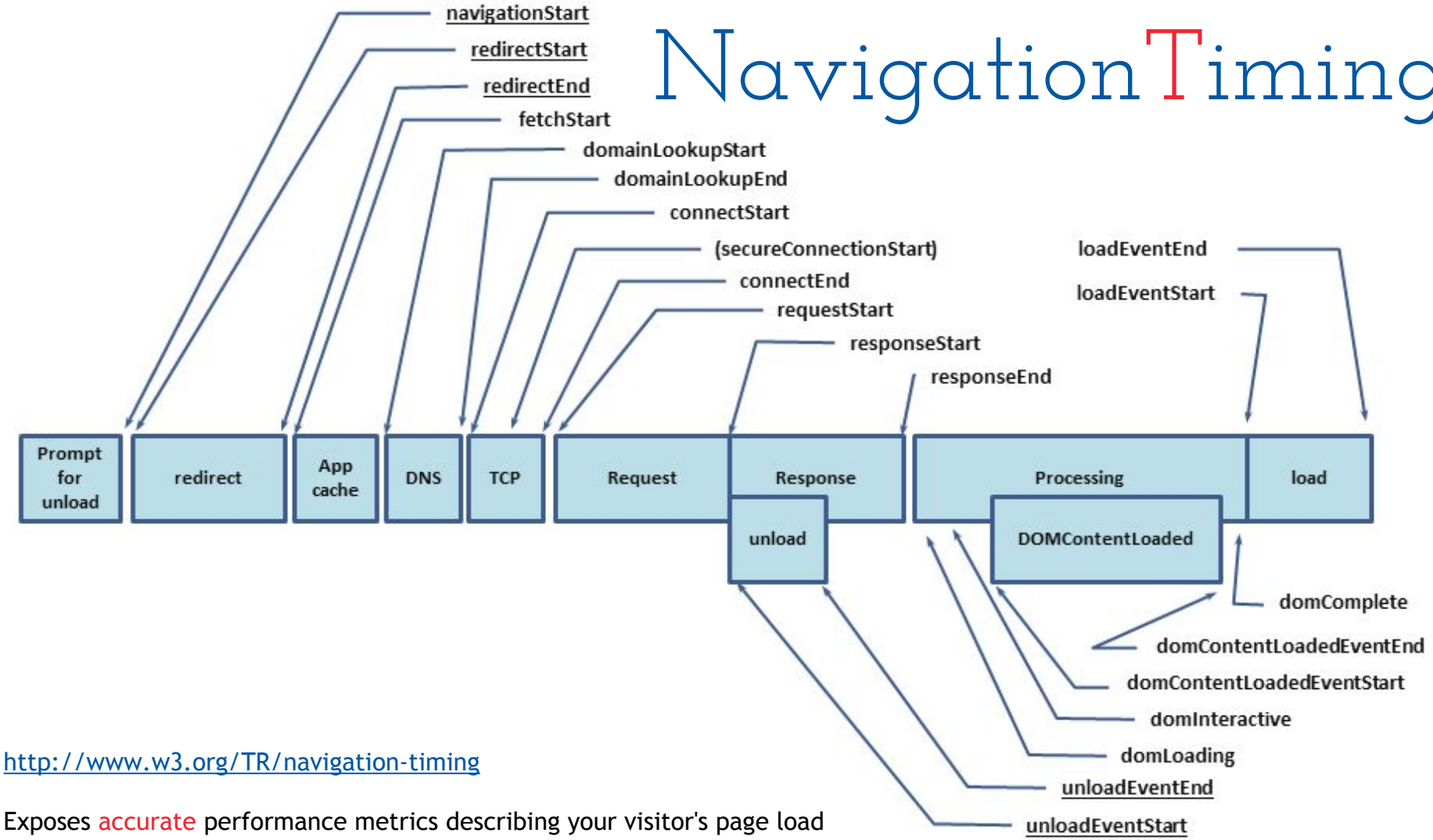
Monotonically non-decreasing

	Date.getTime()	performance.now()
@ Navigation Start	1420147524606	0
@ 100ms	1420147524706	100.0
@ 200ms	1420147524806	200.0
@ 300ms + user's clock moves back 1s	1420147523906	300.0
@ 400ms	1420147524006	400.0

DOMHighResTimeStamp: Usage

```
var myTime = performance.now();

// 8141.84 -> 8.1 seconds after page load
```

NavigationTiming



http://www.w3.org/TR/navigation-timing

Exposes accurate performance metrics describing your visitor's page load

NavigationTiming: Timestamps

`window.performance.timing`

Navigation timestamps:

```
> window.performance.timing

connectEnd: 1473630941946
connectStart: 1473630941946
domComplete: 1473630946441
domContentLoadedEventEnd: 1473630945308
domContentLoadedEventStart: 1473630945294
domInteractive: 1473630945294
domLoading: 1473630943828
domainLookupEnd: 1473630941946
domainLookupStart: 1473630941946
fetchStart: 1473630941946
loadEventEnd: 1473630946441
loadEventStart: 1473630946441
navigationStart: 1473630941871
redirectEnd: 1473630941946
redirectStart: 1473630941871
requestStart: 1473630941951
responseEnd: 1473630943953
responseStart: 1473630943807
secureConnectionStart: 0
unloadEventEnd: 1473630943813
unloadEventStart: 1473630943811
```

NavigationTiming: Characteristics

`window.performance.navigation`

Characteristics of the browser navigation

`window.performance.navigation.type:`

- navigate = 0
- reload = 1
- back / forward = 2

```
> window.performance.navigation
<· ▼PerformanceNavigation {type: 0, redirectCount: 1}
    redirectCount: 1
    type: 0
```

NavigationTiming: Usage

```
function onLoad() {
    if ('performance' in window && 'timing' in window.performance) {
        setTimeout(function() {
            var t = window.performance.timing;
            var ntData = {
                redirect: t.redirectEnd - t.redirectStart,
                dns: t.domainLookupEnd - t.domainLookupStart,
                connect: t.connectEnd - t.connectStart,
                ssl: t.secureConnectionStart ? (t.connectEnd - secureConnectionStart) : 0,
                request: t.responseStart - t.requestStart,
                response: t.responseEnd - t.responseStart,
                dom: t.loadEventStart - t.responseEnd,
                total: t.loadEventEnd - t.navigationStart
            };
        }, 0);
    }
}
```

NavigationTiming: Browser Support

Navigation Timing API - REC

API for accessing timing information related to navigation and elements.

Global	90.73%
unprefixed:	90.65%
U.S.A.	95.63%
unprefixed:	95.58%

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			29						
			49					4.3	
			50					4.4	
8	13	47	51			9.2		4.4.4	
11	14	48	52	9.1	39	9.3	all	51	51
		49	53	10	40				
		50	54	TP	41				
		51	55						

Notes | Known issues (1) | Resources (6) | Feedback

Removed in iOS 8.1 due to poor performance.

NavigationTiming: Integrations

DIY:

- Send this data to your backend for logging
- Show any page's timings via a bookmarklet: kaaes.github.io/timing
- Boomerang: github.com/SOASTA/boomerang
- Boomcatch: cruft.io/posts/introducing-boomcatch
- BoomerangExpress: github.com/andreas-marschke/boomerang-express
- SiteSpeed.io: sitespeed.io
- Piwik: github.com/piwik/piwik

Commercial:

- SOASTA mPulse, Google Analytics Site Speed, New Relic Browser, NeuStar WPM, SpeedCurve, etc...

NavigationTiming: Tips

- Use `fetchStart` instead of `navigationStart` unless you're interested in redirects, tab initialization time, etc.
- `loadEventEnd` will be 0 until after the body's load event has finished (so you can't measure it in the load event)
- We don't have an accurate way to measure the "request time", as "`requestEnd`" is invisible to us (the server sees it)
- Home page scenarios: Timestamps up through `responseEnd` event may be 0 duration because some browsers speculatively pre-fetch home pages (and don't report the correct timings)
- If possible, do any beaconing of the data as soon as possible. Browser `onbeforeunload` isn't 100% reliable for sending data
- Not suitable for Single-Page Apps (we'll cover this later)

Coming Soon to NavigationTiming

- Part of the Performance Timeline: performance.getEntries("navigation")
- Support for `DOMHighResTimeStamp`
- Timing information for `prerender`
- Protocol information: `nextHopProtocol`
- Transfer, encoded body and decoded body sizes

NavigationTiming: chrome.loadTimes()


```
{
   "requestTime"            : 1473093945.032975,
   "startLoadTime"          : 1473093945.129178,
   "commitLoadTime"         : 1473093945.575271,
   "finishDocumentLoadTime" : 1473093946.872513,
   "finishLoadTime"         : 1473093952.281069,
   "firstPaintTime"         : 1473093945.96769,
   "firstPaintAfterLoadTime": 1473093952.316622,
   "navigationType"         : "BackForward",
   "wasFetchedViaSpdy"      : true,
   "wasNpnNegotiated"       : true,
   "npnNegotiatedProtocol"  : "quic/1+spdy/3",
   "wasAlternateProtocolAvailable" : false,
   "connectionInfo"         : "quic/1+spdy/3"
}
```

what did we do before

NavigationTiming?

Prior to NavigationTiming

1. Hook into the `beforeUnload`, `unload` and `pagehide` events to set a cookie with the timestamp and url
2. In the `onload` or `pageshow` event, check if the cookie is set and if the url in the cookie matches `document.referrer`
3. If we have a match, calculate the time delta

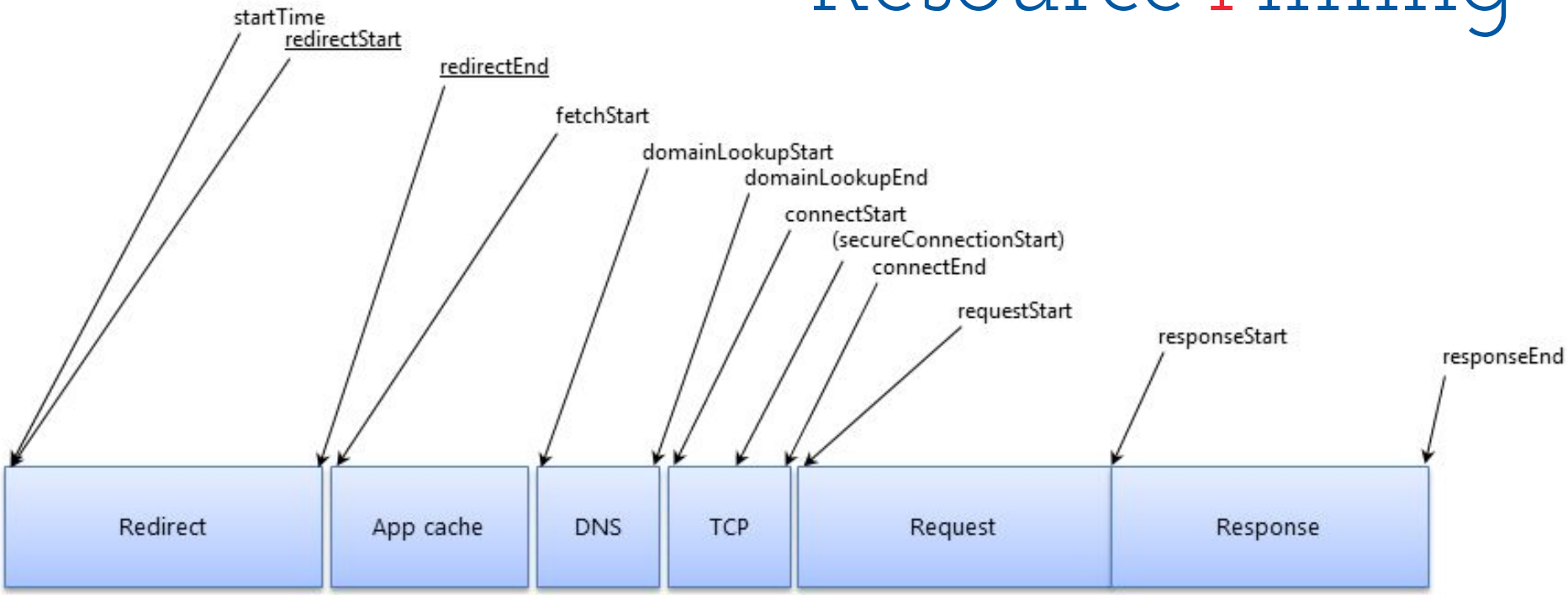
- `beforeUnload` corresponds to `navigationStart`
- `unload` and `pagehide` correspond to `responseStart`
- We also hook into clicks and form submits just in case the user goes off to a new tab

Note: This doesn't help for external referrers!

Surprisingly, this works all the way back to IE5.5

(insofar as we've tested)

ResourceTiming



https://www.w3.org/TR/resource-timing/

Exposes sub-resource performance metrics

ResourceTiming: Inspiration

Developer Tools - http://www.codemash.org/

Elements | Network | Sources | Timeline | Profiles | Resources | Audits | Console | AngularJS

Preserve log | Disable cache

Filter | All | Documents | Stylesheets | Images | Media | Scripts | XHR | Fonts | TextTracks | WebSockets | Other | Hide data URLs

Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency	Timeline (1.00 s, 1.50 s, 2.00 s, 2.50 s)
www.codemash.org	GET	200 OK	text/html	Other	5.5 KB 15.9 KB	939 ms 931 ms	
style.css?ver=2.1.2 /wp-content/themes/codemash	GET	200 OK	text/css	www.codemash.org/:19 Parser	16.4 KB 64.6 KB	407 ms 377 ms	
meteor-slides.css?ver=1.0 /wp-content/plugins/meteor-slides/css	GET	200 OK	text/css	www.codemash.org/:20 Parser	1.8 KB 4.9 KB	309 ms 307 ms	
jquery-migrate.min.js?ver=1.2.1 /wp-includes/js/jquery	GET	200 OK	application/...	www.codemash.org/:22 Parser	3.9 KB 7.0 KB	160 ms 157 ms	
jquery.js?ver=1.11.1 /wp-includes/js/jquery	GET	200 OK	application/...	www.codemash.org/:21 Parser	42.1 KB 93.6 KB	666 ms 579 ms	
jquery.cycle.all.js?ver=4.1 /wp-content/plugins/meteor-slides/js	GET	200 OK	application/...	www.codemash.org/:23 Parser	18.4 KB 52.5 KB	498 ms 441 ms	
jquery.metadata.v2.js?ver=4.1 /wp-content/plugins/meteor-slides/js	GET	200 OK	application/...	www.codemash.org/:24 Parser	2.3 KB 5.1 KB	293 ms 290 ms	
jquery.touchwipe.1.1.1.js?ver=4.1 /wp-content/plugins/meteor-slides/js	GET	200 OK	application/...	www.codemash.org/:25 Parser	1.4 KB 2.2 KB	260 ms 257 ms	
slideshow.js?ver=4.1 /wp-content/plugins/meteor-slides/js	GET	200 OK	application/...	www.codemash.org/:31 Parser	1.3 KB 2.3 KB	338 ms 336 ms	
comment-reply.min.js?ver=4.1 /wp-includes/js	GET	200 OK	application/...	www.codemash.org/:1... Parser	867 B 757 B	391 ms 389 ms	
child-theme-min.js?ver=4.1 /wp-content/themes/codemash/js	GET	200 OK	application/...	www.codemash.org/:2... Parser	776 B 757 B	399 ms 397 ms	
codemash-icon-featured-box.png /wp-content/uploads/2014/07	GET	200 OK	image/png	www.codemash.org/:89 Parser	8.2 KB 8.0 KB	125 ms 120 ms	
megaphone.png /wp-content/uploads/2014/07	GET	200 OK	image/png	www.codemash.org/:1... Parser	4.7 KB 4.5 KB	144 ms 141 ms	
home-widget-1.jpg /wp-content/uploads/2014/07	GET	200 OK	image/jpeg	www.codemash.org/:1... Parser	52.2 KB 52.0 KB	172 ms 134 ms	
160px.QuickenLoans__raster.png /wp-content/uploads/2014/08	GET	200 OK	image/png	www.codemash.org/:1... Parser	7.8 KB 7.6 KB	139 ms 135 ms	
home-widget-2.jpg /wp-content/uploads/2014/07	GET	200 OK	image/jpeg	www.codemash.org/:1... Parser	43.0 KB 42.8 KB	178 ms 143 ms	
up-arrow-button.png /wp-content/themes/codemash/images	GET	200 OK	image/png	www.codemash.org/:1... Parser	743 B 496 B	165 ms 163 ms	

ResourceTiming: History

How it was done in the old days:

```
var start = new Date().getTime();
var image1 = new Image();
var resourceTiming = function() {
    var now = new Date().getTime();
    var latency = now - start;
    alert("End to end resource fetch: " + latency);
};

image1.onload = resourceTiming;
image1.src = 'http://www.w3.org/Icons/w3c_main.png';
```

(not practical for all types of content -- or a regular HTML website)

PerformanceTimeline

Unifying interface to access and retrieve performance metrics

`window.performance:`

- `getEntries()`: Gets all entries in the timeline
- `getEntriesByType(type)`: Gets all entries of the specified type (eg resource, mark, measure)
- `getEntriesByName(name)`: Gets all entries with the specified name (eg URL or mark name)

PerformanceTimeline: Usage

```
> performance.getEntriesByType("resource")
<· ▼Array[150] i
    ▼[0 … 99]
      ▶0: PerformanceResourceTiming
      ▶1: PerformanceResourceTiming
      ▶2: PerformanceResourceTiming
      ▶3: PerformanceResourceTiming
      ▶4: PerformanceResourceTiming
      ▶5: PerformanceResourceTiming
      ▶6: PerformanceResourceTiming
      ▶7: PerformanceResourceTiming
      ▶8: PerformanceResourceTiming
      ▶9: PerformanceResourceTiming
      ▶10: PerformanceResourceTiming
```

ResourceTiming: performance.getEntriesByType("resource")[0]

```
{
    name                    : "http://www.foo.com/foo.png",
    initiatorType           : "img",
    entryType               : "resource",
    startTime               : 566.357000003336,
    workerStart             : 0,
    redirectStart           : 0,
    redirectEnd             : 0,
    fetchStart              : 566.357000003336,
    domainLookupStart       : 566.357000003336,
    domainLookupEnd         : 566.357000003336,
    connectStart            : 566.357000003336,
    secureConnectionStart   : 0,
    connectEnd              : 566.357000003336,
    requestStart            : 568.4959999925923,
    responseStart           : 569.4220000004862,
    responseEnd             : 570.6329999957234,
    duration                : 4.275999992387369
}
```

ResourceTiming: InitiatorType

- `img`
- `link`
- `script`
- `css: url(), @import`
- `xmlhttprequest`
- `image` (SVG)
- `object` (Flash)

localName of the element

ResourceTiming: Buffer

- There is a ResourceTiming buffer (per IFRAME) that stops filling after its size limit is reached (default: 150 entries)
- Listen for the `onResourceTimingBufferFull` event
- `setResourceTimingBufferSize(n)` and `clearResourceTimings()` can be used to modify it
- Do NOT `setResourceTimingBufferSize(99999999)` as this can lead to browser memory growing unbound

ResourceTiming: Compressing

- Each resource is ~ 500 bytes `JSON.stringify()`'d
- HTTP Archive tells us there's 103 HTTP resources on average, per page, with an average URL length of 85 bytes
- That means you could expect around 45 KB of ResourceTiming data per page load
- For comparison, the default TCP Window size allows 15 KB to go through before requiring an ACK, so do the math.
- Compress it: nicj.net/compressing-resourcetiming

ResourceTiming: Compressing

```
{
    connectEnd: 566.357000003336,
    connectStart: 566.357000003336,
    domainLookupEnd: 566.357000003336,
    domainLookupStart: 566.357000003336,
    duration: 4.275999992387369,
    entryType: "resource",
    fetchStart: 566.357000003336,
    initiatorType: "img",
    name: "http://www.foo.com/foo.png",
    redirectEnd: 0,
    redirectStart: 0,
    requestStart: 568.4959999925923,
    responseEnd: 570.6329999957234,
    responseStart: 569.4220000004862,
    secureConnectionStart: 0,
    startTime: 566.357000003336,
    workerStart: 0
}
```

```
{
  "http://": {
    "foo.com/": {
      "js/foo.js": "370,1z,1c",
      "css/foo.css": "48c,5k,14"
    },
    "moo.com/moo.gif": "312,34,56"
  }
}
```

Compresses ResourceTiming data down to 15% of original size

https://github.com/nicjansma/resourcetiming-compression.js

ResourceTiming: Timing-Allow-Origin

- By default to protect the user's privacy, cross-origin resources expose timestamps for only the `fetchStart` and `responseEnd` attributes
- If you have a CDN, or multiple domains, you can allow access to this data from your primary domain
- Use the TAO:
  `Timing-Allow-Origin: origin-list-or-null` OR `*`
- Note: Third-party libraries (ads, analytics, etc) must set this on their servers. 5% do according to HTTP Archive. Google, Facebook, Disqus, mPulse, etc.

  What are the others afraid of?

ResourceTiming: Timing-Allow-Origin

```php
// PHP
<?php
    header('Timing-Allow-Origin: *');
?>

// JBoss
protected void doPost(HttpServletRequest req,
                      HttpServletResponse res) {
    res.setHeader("Timing-Allow-Origin", "*");
}
```

```
// Apache .htaccess
<IfModule mod_headers.c>
    Header set Timing-Allow-Origin "*"
</IfModule>

// nginx
location / {
    add_header 'Timing-Allow-Origin' '*';
}
```

And we can get more creative if we only want to allow specific Origins

ResourceTiming: Blocking Time

- Browsers open a limited number of connections to each unique origin
- If there are more resources than this number, later resources "block"
- ResourceTiming `duration` includes Blocking time!
- So, don't use duration... but this is all you get with cross-origin resources.

```
var waitTime = 0;
if (res.connectEnd && res.connectEnd === res.fetchStart) {
    waitTime = res.requestStart - res.connectEnd;
}
else if (res.domainLookupStart) {
    waitTime = res.domainLookupStart - res.fetchStart;
}
```

ResourceTiming: Cache Hits

- Cached resources will show up along side resources that were fetched from the network
- Due to privacy concerns, no direct indication a resource was fetched from the cache
- In practice, resources with a very short duration are likely cache hits
  - 0 - 2ms → In memory cache
  - 2 - 10ms → Disk cache
  - 10 - 40ms → Cached by Edge Proxy

ResourceTiming: Integrations

DIY:

- Compress + send this data to your backend for logging
- Show any page's resources via a bookmarklet:
  github.com/andydavies/waterfall
- Heatmap bookmarklet / Chrome extension:
  github.com/zeman/perfmap
- Nurun's Performance Bookmarklet:
  github.com/nurun/performance-bookmarklet
- Boomerang supports ResourceTiming:
  github.com/SOASTA/boomerang

Pay:

- SOASTA mPulse, New Relic Browser, SpeedCurve, etc.

ResourceTiming: Browser Support

Resource Timing - CR

Global 71.45%
U.S.A. 69.69%

Method to help web developers to collect complete timing information related to resources on a document.

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49					4.4	
8	13	47	51			9.2		4.4.4	
11	14	48	52	9.1	39	9.3	all	51	51
		49	53	10	40				
		50	54	TP	41				
		51	55						

Notes | Known issues (0) | Resources (6) | Feedback

[1] Can be enabled in Firefox using the dom.enable_resource_timing flag

ResourceTiming: Polyfill

ResourceTiming isn't yet available on iOS, but you can *polyfill* it using `MutationObserver`:

1. Start a `MutationObserver` listening for new nodes with a `src` or `href`
2. Add `load` & `error` event listeners & a `timeout` to deal with cached resources
3. Once the `load` (or `error`) event has fired, you have the total load time for the resource (keep in mind that an error event might also fire on Network Error)

In addition, you'll want to instrument `XMLHttpRequest` (which won't be captured by `MutationObserver`):

1. Proxy the `XMLHttpRequest` object
2. Hook into `.open()` and `.send()` and add `onreadystatechange` listeners

Sample code: github.com/SOASTA/boomerang/blob/master/plugins/auto_xhr.js

Note: This doesn't give you detailed information such as DNS & TCP timings

ResourceTiming: Tips

- Ensure your CDNs and third-party libraries send `Timing-Allow-Origin`
- What isn't included in ResourceTiming:
  - The root HTML page (get this from `window.performance.timing`)
  - HTTP code (privacy concerns)
  - Content that loaded with errors (eg 404s) (browser inconsistencies)
- If you're going to be managing the ResourceTiming buffer, make sure no other scripts are managing it as well
- Each IFRAME will have its own ResourceTiming data, and those resources won't be included in the parent FRAME/document. So you'll need to traverse the document frames to get all resources
- `about:blank`, `javascript:` URLs will be seen in RT data
- You may see browser extensions fetching resources in RT data

ResourceTiming2: Coming Soon

Available in recent Firefox, Chrome:

- `nextHopProtocol`: ALPN Protocol ID (e.g. quic+`http2`)
- `transferSize`: Bytes transferred for HTTP header and response
- `decodedBodySize`: Size of the body after removing any applied content-codings
- `encodedBodySize`: Size of the body after prior to removing any applied content-codings

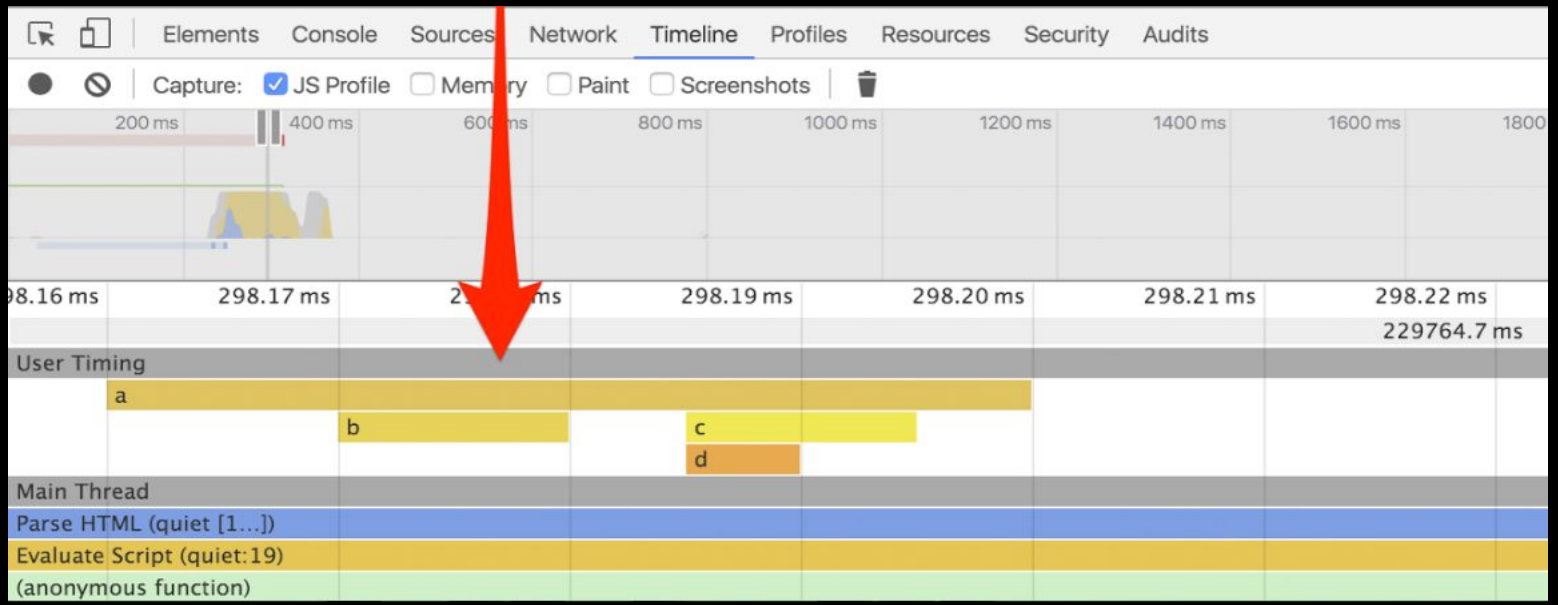
UserTiming

Measuring in-page scripts and other things that don't fire events

UserTiming

https://www.w3.org/TR/user-timing/

Standardized interface to note timestamps ("marks") and durations ("measures")



UserTiming: History

How it was done before:

```
var start = new Date().getTime();
// do stuff
var now = new Date().getTime();
var duration = now - start;
```

UserTiming is a better way of doing this!

UserTiming: Marks & Measures

- Mark: A timestamp
- Measure: The delta between two timestamps

UserTiming: Usage

Creating:

- `window.performance.mark(name)`
- `window.performance.measure(name, [start], [end])`

Clearing:

- `window.performance.clearMarks([name])`
- `window.performance.clearMeasures([name])`

Querying:

- `window.performance.getEntriesByType("mark")`
- `window.performance.getEntriesByType("measure")`

UserTiming: Mark

```
// mark
performance.mark("start");
performance.mark("end");

performance.mark("another");
performance.mark("another");
performance.mark("another");
```

```
// retrieve
performance.getEntriesByType("mark");

[
    {
        "duration":0,
        "startTime":150384.48100000096,
        "entryType":"mark",
        "name":"start"
    },
    {
        "duration":0,
        "startTime":150600.5250000013,
        "entryType":"mark",
        "name":"end"
    },
    ...
]
```

UserTiming: Measure

```
// measure
performance.mark("start");

// do work
performance.mark("start2");

// measure from "now" to the "start" mark
performance.measure("time to do stuff", "start");

// measure from "start2" to the "start" mark
performance.measure("time from start to start2", "start", "start2");

// retrieval - specific
performance.getEntriesByName("time from start to start2", "measure");
```


```
[
    {
        "duration":4809.890999997151,
        "startTime":145287.66500000347,
        "entryType":"measure",
        "name":"time from start to start2"
    }
]
```

UserTiming: Benefits

- Uses the PerformanceTimeline, so marks and measures are in the PerformanceTimeline along with other events
- Uses `DOMHighResTimestamp` instead of `Date` so timestamps are sub-millisecond, monotonically non-decreasing, etc
- Browsers and third-party tools can find your performance events easily

UserTiming: Polyfill

- It's easy to add a Polyfill that adds UserTiming support to browsers that do not natively support it
- UserTiming is accessed via the PerformanceTimeline, and requires `window.performance.now()` support, so UserTiming.js adds a limited version of these interfaces if the browser does not support them
- github.com/nicjansma/usertiming.js

UserTiming: Compressing

Compresses `performance.getEntriesByName("mark")`:


```
[{"duration":0,"entryType":"mark","name":"mark1","startTime":100.0},
{"duration":0,"entryType":"mark","name":"mark2","startTime":150.0},
{"duration":0,"entryType":"mark","name":"mark3","startTime":500.0},
{"duration":0,"entryType":"mark","name":"measure1","startTime":100.0},
{"duration":100,"entryType":"mark","name":"measure2","startTime":150.0},
{"duration":200,"entryType":"mark","name":"measure3","startTime":500.0}]
```


Down to something more reasonable:


```
~(m~(ark~(1~'2s~2~'5k~3~'8c)~easure~(1~'2s_2s~2~'5k_5k~3~'8c_8c)))
```


nicj.net/compressing-usertiming/

github.com/nicjansma/usertiming-compression.js

UserTiming: Browser Support

User Timing API - REC

Method to help web developers measure the performance of their applications by giving them access to high precision timestamps.

Global	71.35%
U.S.A.	69.6%

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49					4.4	
8	13	47	51			9.2		4.4.4	
11	14	48	52	9.1	39	9.3	all	51	51
		49	53	10	40				
		50	54	TP	41				
		51	55						

Notes | Known issues (0) | Resources (8) | Feedback

No notes

UserTiming: Dev Tools

F12

UI Responsiveness

Edge

Diagnostic session: 14.954 seconds (3.82 seconds selected)

User mark | App lifecycle mark

2.5s | 5s | 7.5s | 10s | 12.5s

CPU utilization (%) — Loading, Scripting, GC, Styling, Rendering, Image decoding

Visual throughput (FPS) — Frames per second

Timeline details — Sort by: Start time — Filter events

Event name	4s
User measure (box cycler)	
DOM event (click)	6.81 ms (0.016 ms)
Layout	1.13 ms (0.46 ms)
Paint	1.68 ms (0.38 ms)
Style calculation	0.21 ms (0.052 ms)
Timer (colorCycle)	3.9 ms (3.16 ms)
Paint	0.85 ms (0.41 ms)
Layout	0.92 ms (0.38 ms)
Timer (colorCycle)	2.53 ms (2 ms)
Paint	6.67 ms (0.79 ms)
Layout	6.01 ms (5.51 ms)
Timer (colorCycle)	15.25 ms (14.02 ms)
Paint	9.37 ms (0.8 ms)
Layout	8.42 ms (6.94 ms)

User measure

Duration (inclusive):	3.82 s
Duration (exclusive):	3.82 s
Start time:	1.54 s
Name:	box cycler
Start mark:	Begin Rotation
End mark:	End Rotation

An app-specific scenario was measured using the performance.measure() method.

UserTiming: Dev Tools

Last User Timing Mark

Load Time	First Byte	Start Render	User Time	Visually Complete	Speed Index	DOM Elements	Result (error code)	Document Complete			Fully Loaded		
								Time	Requests	Bytes In	Time	Requests	Bytes In
2.445s	0.133s	0.440s	3.050s	3.400s	1172	877	0	2.445s	35	705 KB	5.067s	52	844 KB

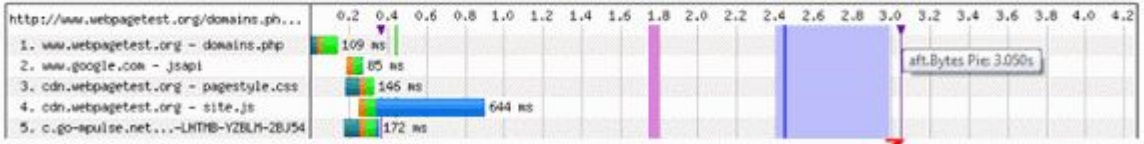
Individual Marks

aft.Site Header	aft.Bytes Pie	aft.Requests Pie	domContentLoaded	loadEvent
0.360s	3.050s	3.050s	1.744s - 1.797s (0.053s)	2.403s - 2.983s (0.580s)

Waterfall View

DNS Lookup | Initial Connection | SSL Negotiation | Time to First Byte | Content Download | 3xx response | 4xx+ response

Start Render | DOM Content Loaded | On Load | Document Complete | User Timings



Individual Marks

Other Useful APIs

Visibility and Painting

PageVisibility

Lets you know when a webpage is *visible* or *in focus*.

`document.visibilityState:`

- `hidden`
  - Browser is minimized
  - Background tab
  - About to unload or traverse session history
  - OS lock screen
- `visible`
- `prerender`
  - Being speculatively pre-rendered
  - Important for analytics!
- `unloaded`

PageVisibility: Usage

```
// query the current state
var state = document.visibilityState;

// listen for state change events
document.addEventListener("visibilitychange", function() {
  if (document.visibilityState === "hidden") {
    // stop doing something
  } else if (document.visibilityState === "hidden") {
    // restart doing something
  }
});
```

PageVisibility: Browser Support

Page Visibility - REC

JavaScript API for determining whether a document is visible on the display

Global	90.77%
unprefixed:	76.77%
U.S.A.	96.59%
unprefixed:	91.29%

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49					4.4	
8	13	47	51			9.2		4.4.4	
11	14	48	52	9.1	39	9.3	all	51	51
		49	53	10	40				
		50	54	TP	41				
		51	55						

Notes | Known issues (1) | Resources (8) | Feedback

No notes

requestAnimationFrame

Tells the browser you wish to run a function prior to the next repaint:

```
var last = performance.now();

function raf(timestamp) {
  var now = performance.now();
  var diff = last - now;

  // update the UI based on the difference in time

  last = now;
  requestAnimationFrame(raf);
}

requestAnimationFrame(raf);
```

More examples when we talk about measuring continuity.

requestAnimationFrame: Browser Support

requestAnimationFrame - LS

API allowing a more efficient way of running script-based animation, compared to traditional methods using timeouts. Also covers support for `cancelAnimationFrame`

Global	90.56%	+ 0.24% =	90.81%
unprefixed:	90.32%		
U.S.A.	96.49%	+ 0.09% =	96.58%
unprefixed:	96.4%		

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49					4.4	
8	13	47	51			9.2		4.4.4	
11	14	48	52	9.1	39	9.3	all	51	51
		49	53	10	40				
		50	54	TP	41				
		51	55						

Notes | Known issues (1) | Resources (6) | Feedback

[1] Partial support refers to lacking `cancelAnimationFrame` support.

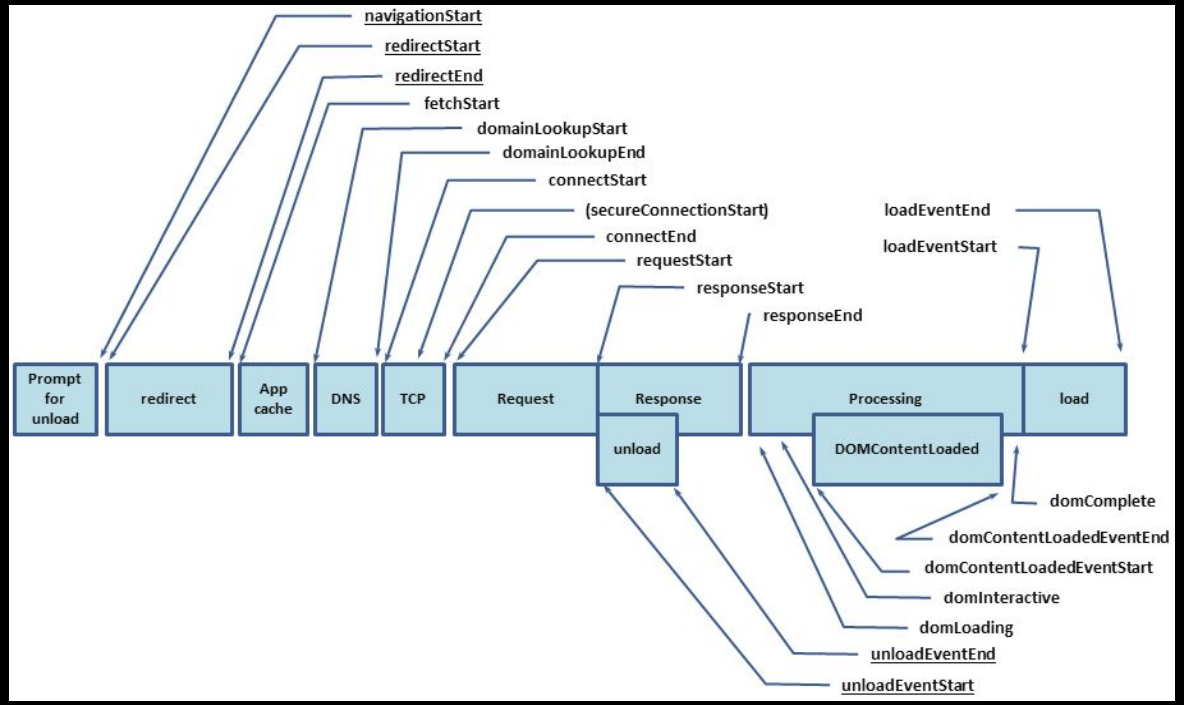
[2] Supports `webkitCancelRequestAnimationFrame` rather than `webkitCancelAnimationFrame.

Page Load Milestones

- First Byte
  - First content was received from the server
  - = `responseStart`
- `onload`
  - Once all content statically included or injected before onload has been fetched
  - = `loadEventStart`
- Fully Loaded
  - Once all static & dynamic content has been fetched
  - No browser event!

Visual Experience

When does the user feel like they can use the app?



Network timings != visual experience

Visual Experience

Milestones:

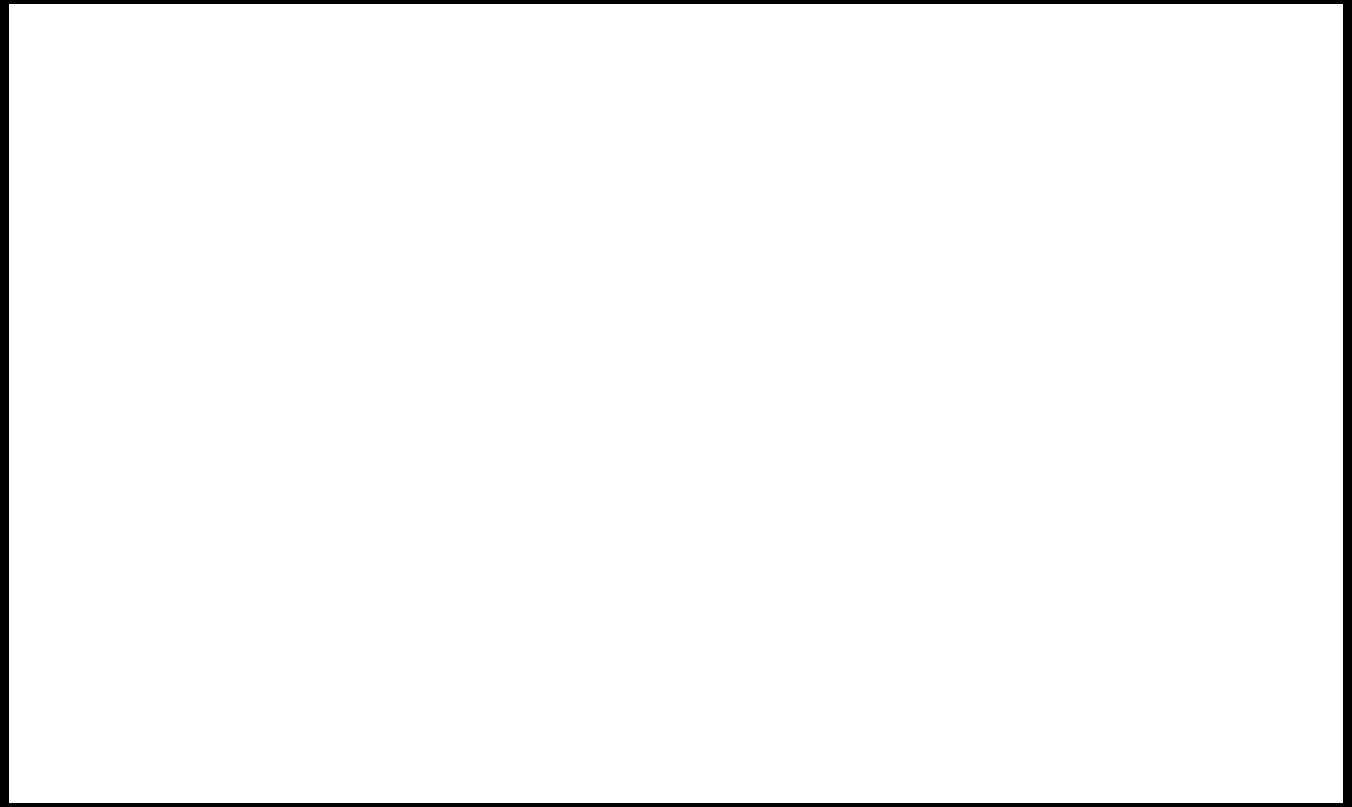
- First Paint
- First Contentful Paint
- First Meaningful Paint
- Visually Complete

Metrics:

- Visual Progress
- Speed Index

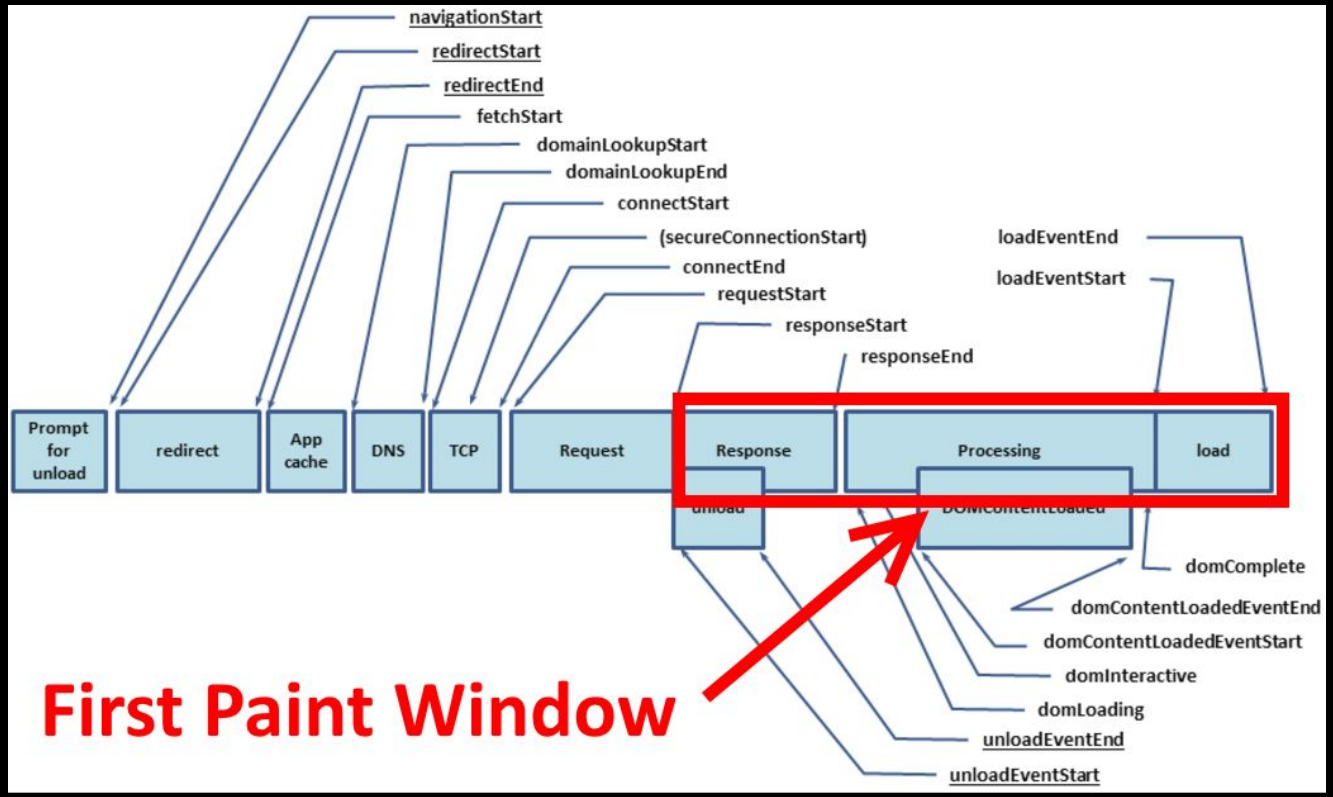
Visual Experience: First Paint

What was the first thing the user saw?



Visual Experience: First Paint

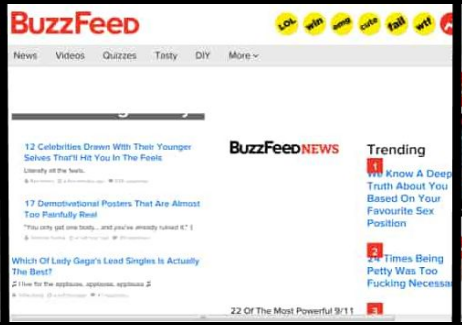
What was the first thing the user saw?

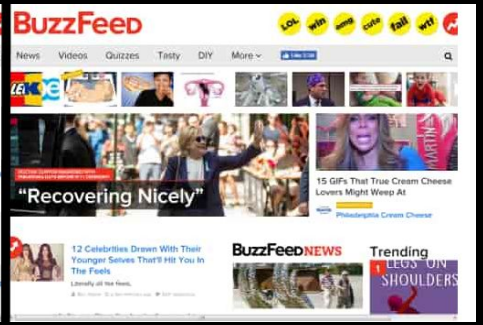


Visual Experience: First Paint

- Not an industry standard metric!
- The first paint of the browser might show zero content (all white)

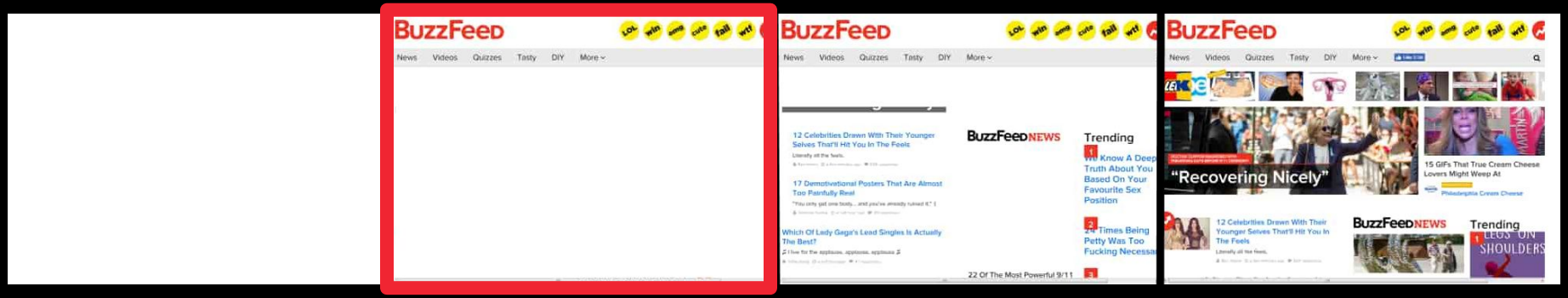
```
// IE 9+ only
window.performance.timing.msFirstPaint
// -> 1473640901

// Chrome only
window.chrome.loadTimes().firstPaintTime;
// -> 1473640917.063874
```





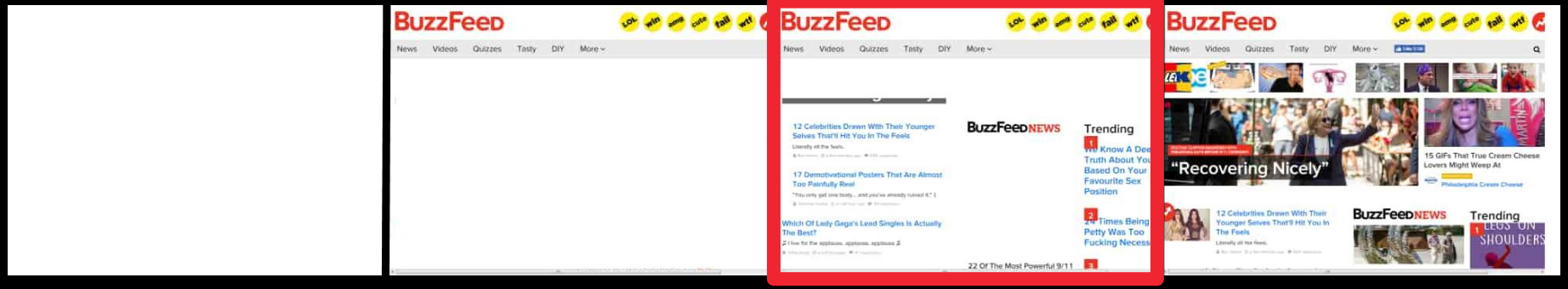
Visual Experience: First Contentful Paint

- First time a "contentful" thing is painted:
  - text
  - image
  - canvas
  - SVG
- Could still be just a minor page element
  - e.g. just a navigation bar



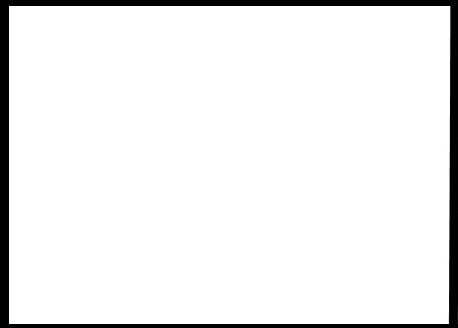
Visual Experience: First Meaningful Paint

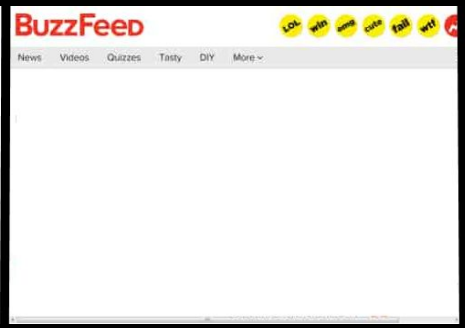
- Page's primary content appears on screen
- Primary content differs for each page
- Definition still being developed, but this could be a heuristic, guided by hints from developers

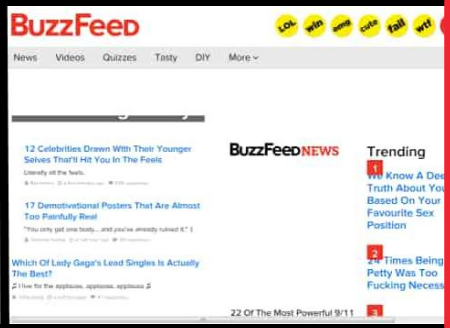


Visual Experience: Visually Complete

- All content has been displayed on the screen
- Might be hard to measure with animations, ads, etc
- Not the same as `onload`! Content can load after `onload`.







Visual Experience

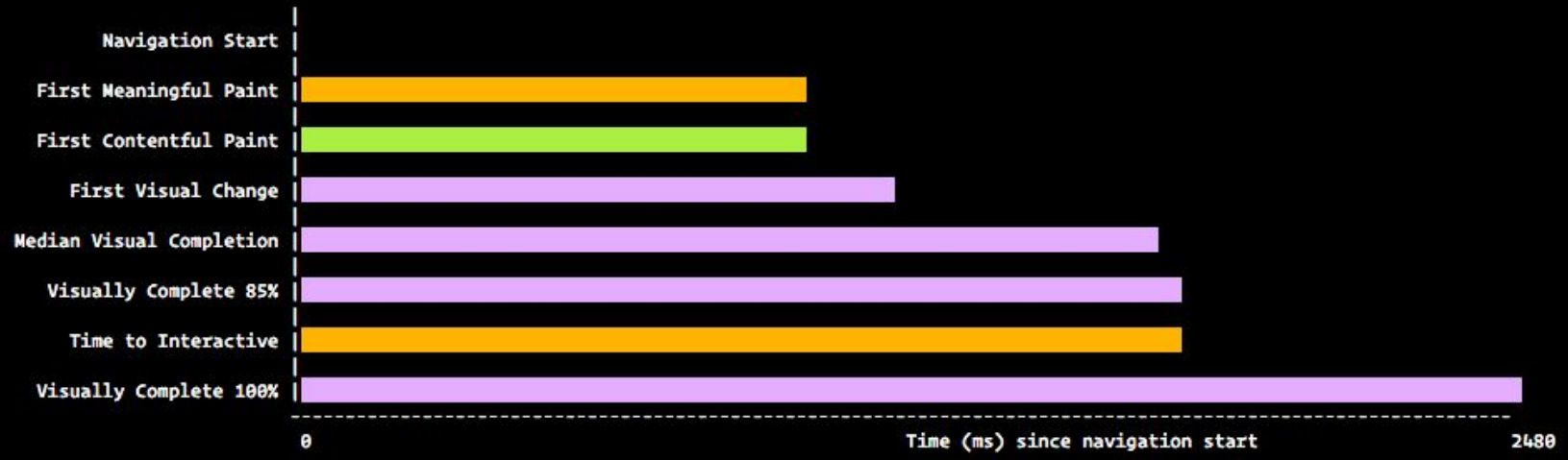
- Besides First Paint, none of these are available in browsers today:
  - First Contentful Paint
  - First Meaningful Paint
  - Visually Complete
- Currently being developed into industry-standard definitions
- Also options:
  - First Non-White (non-background) Paint
  - First Non-Blank Paint

Progressive Web Metrics

Via Paul Irish: github.com/paulirish/pwmetrics

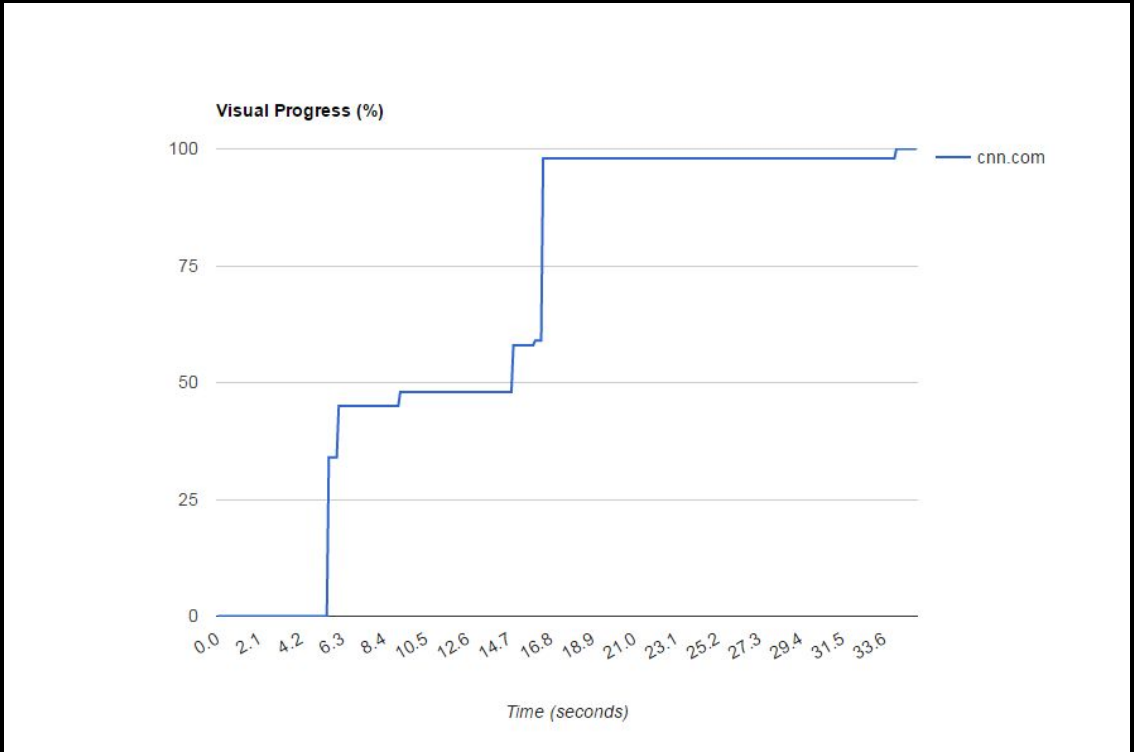
fish

```
~/code/pwa-metrics on (master *)
▸ pwmetrics http://developers.google.com
```



Visual Progress

Percentage of screen drawn over time (relative to last frame)

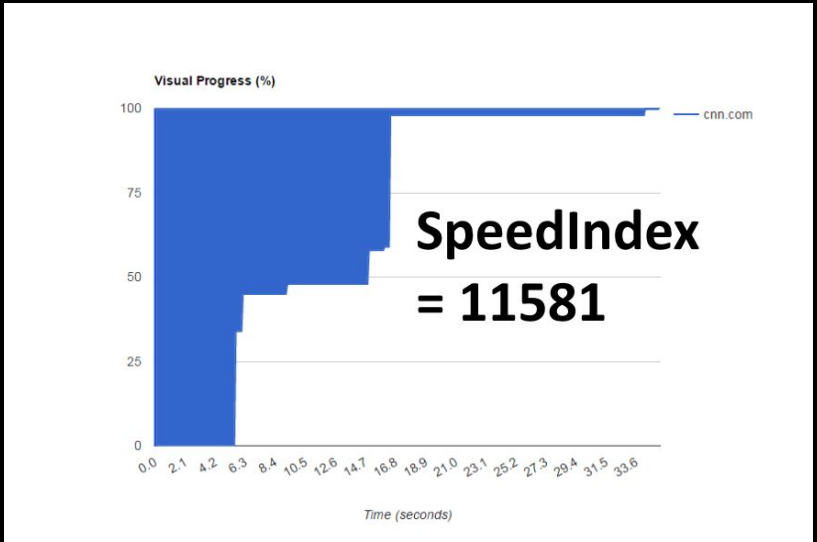


Speed Index

- Average time at which visible parts of the page are displayed
- Expressed in milliseconds
- Area above the curve
- Lower the better

$$\text{Speed Index} = \int_0^{end} 1 - \frac{VC}{100}$$

end = end time in milliseconds
VC = % visually complete



Speed Index

Downsides:

- Not very well understood
- Can be hard to describe (even to techies! let alone marketing)
- Can only be captured accurately in a lab (synthetic testing)

RUM Speed Index

Calculate Speed Index measurements from the field using Resource Timings

- Depends on ResourceTiming support
- Still being developed
  - Needs better support for IFRAMEs, SVGs, etc

github.com/WPO-Foundation/RUM-SpeedIndex

[30 minute break]

http://slideshare.net/nicjansma/measuring-real-user-performance-in-the-browser

Single Page Apps

Single Page Apps (SPAs)

- Run on a single page, dynamically bringing in content as necessary
- Frameworks such as AngularJS, Ember.js, Backbone.js, React, etc.

Definitions

- Hard Navigation: The first page load, which will include all static HTML, JavaScript, CSS, the SPA framework itself (e.g. `angular.js`), plus showing the initial route
- Soft Navigation: Any subsequent route (address bar) change
- Any URL might be loaded via either hard or soft navigation

SPAs...

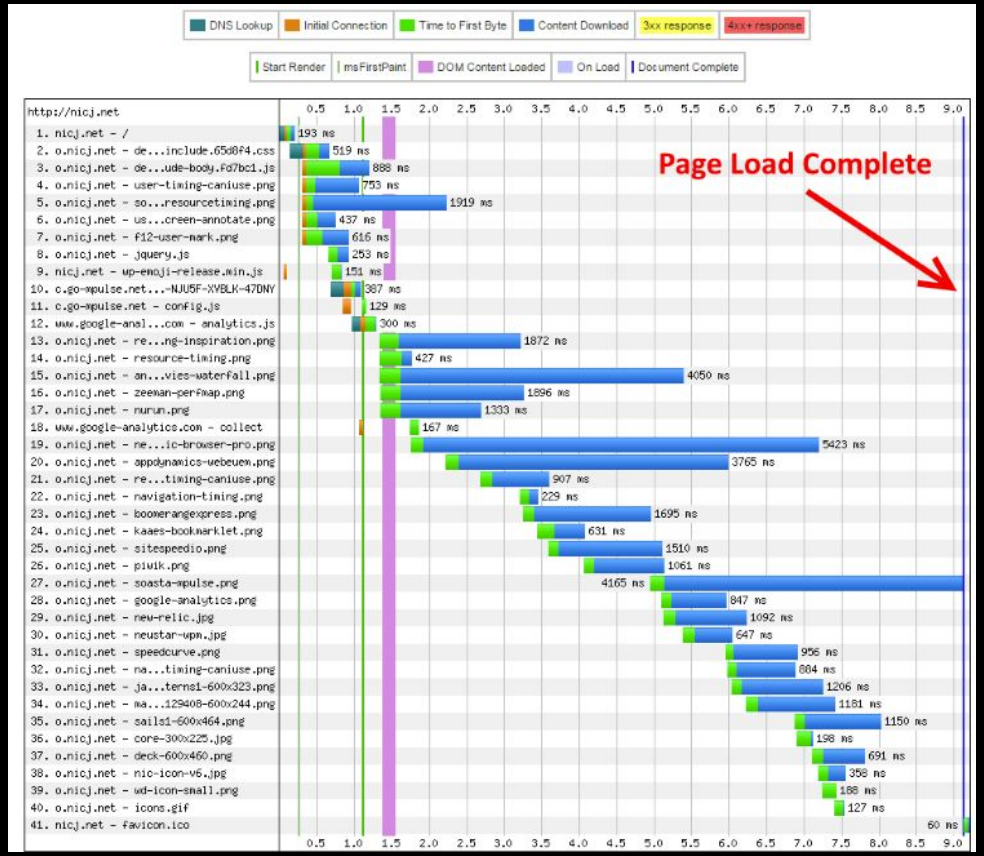
3 Challenges

Challenge 1: The onload Event No Longer Matters

Traditional Websites:

- On navigation, the browser begins downloading all of the JavaScript, CSS, images and other static resources
- Once all static resources are fetched, the body's `onload` event will fire
- This is traditionally what websites consider as page load complete

Challenge 1: The onload Event No Longer Matters



Challenge 1: The onload Event No Longer Matters

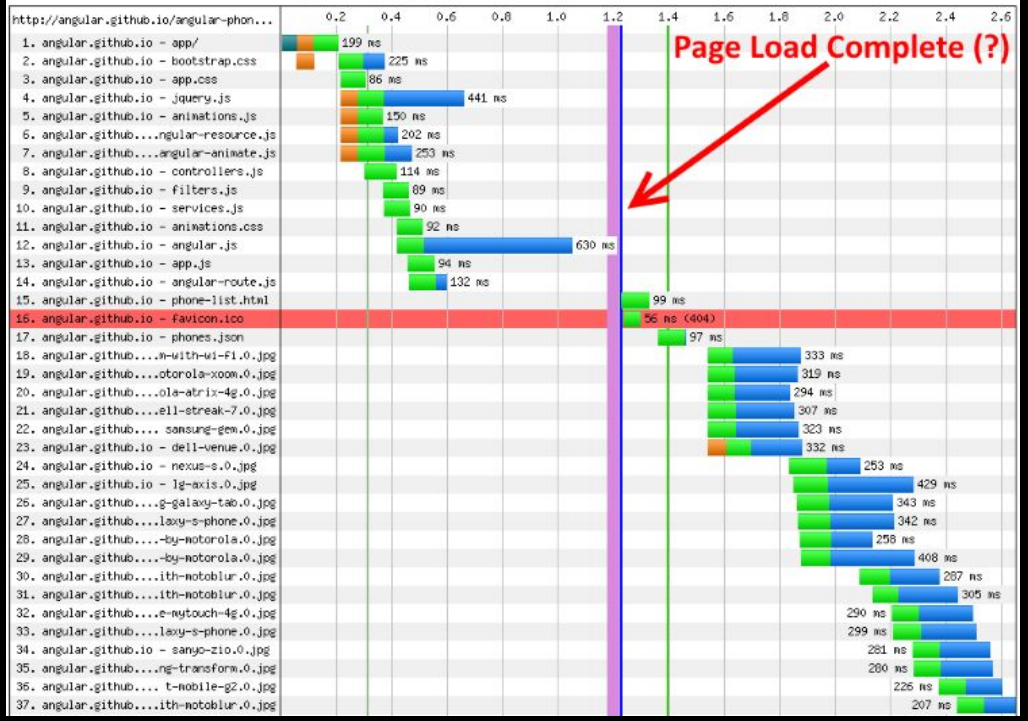
Single Page Apps:

- Load all static content like a traditional website
- The frameworks' code will also be fetched (e.g. `angular.js`)
- (the `onload` event fires here)
- Once the SPA framework is loaded, it starts looking at routes, fetching views and data
- All of this content is fetched after the `onload` event

Challenge 1: The onload Event No Longer Matters

Load Time	First Byte	Start Render	Visually Complete	Speed Index	DOM Elements	Result (error code)	Document Complete			Fully Loaded		
							Time	Requests	Bytes In	Time	Requests	Bytes In
1.225s	0.204s	1.395s	1.900s	1794	155	99999	1.225s	14	432 KB	2.646s	37	1,001 KB

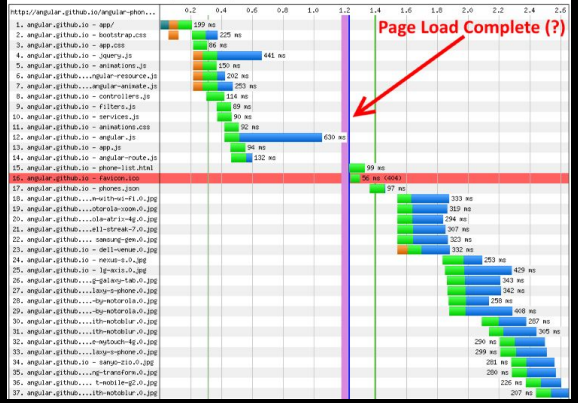
RUM First Paint	domContentLoaded	loadEvent
0.310s	1.180s - 1.219s (0.039s)	1.222s - 1.222s (0.000s)



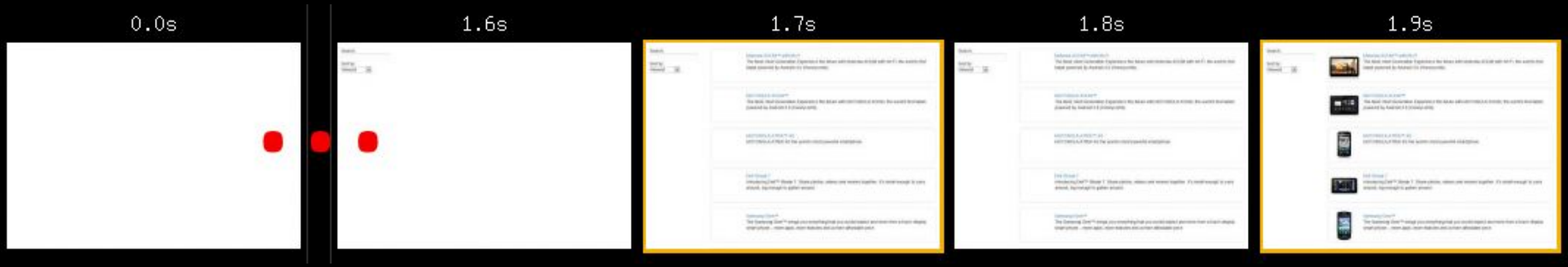
Challenge 1: The onload Event No Longer Matters

Load Time	First Byte	Start Render	Visually Complete	Speed Index	DOM Elements	Result (error code)	Document Complete			Fully Loaded		
							Time	Requests	Bytes In	Time	Requests	Bytes In
1.225s	0.204s	1.395s	1.900s	1794	155	99999	1.225s	14	432 KB	2.646s	37	1,001 KB

RUM First Paint	domContentLoaded	loadEvent
0.310s	1.180s - 1.219s (0.039s)	1.222s - 1.222s (0.000s)



- Browser fires `onload` at 1.225 seconds
- All visual resources (`.jpgs`) aren't complete until after 1.7 seconds
- Filmstrip confirms nothing is shown until around 1.7 seconds
- `onload` fired 0.5 seconds too early!



Challenge 2: Soft Navs are not Real Navigations

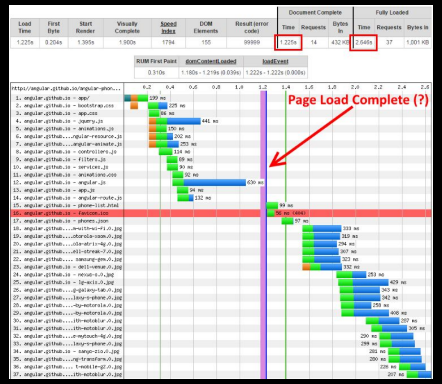
- This is great for performance
- The browser is no longer re-rendering the same header, footer or common components
- The browser is no longer re-parsing the same HTML, JavaScript and CSS

Bad for measuring:

- Browser events (`readyState`, `onload`) and metrics (NavigationTiming) are all geared toward a single load event
- Won't run again until the next time it loads on a full navigation

Challenge 3: Browser Won't Tell You When All Downloads Have Completed

- The browser fires `onload` only once
- The `onload` event helps us know when all static content was fetched
- In a soft navigation scenario, the browser does not fire the `onload` event again, so we don't know when its content was fetched



Challenge 3: Browser Won't Tell You When All Downloads Have Completed

SPA soft navigations may fetch:

- Templates
- Images
- CSS
- JavaScript
- XHRs
- Videos

How to Measure SPAs

We need to figure out at what point the navigation started (the start event), through when we consider the navigation complete (the end event).

How to Measure SPAs: Start Event

Hard navigations:

- Same as a traditional app - `navigationStart`

Soft navigations:

- We need to figure out when the user's view is going to significantly change
- The browser history is changing
- SPA framework routing events can give us an indicator that the view will be changing
- Other important events that might indicate a view change are a user click, or an XHR that triggers DOM changes

How to Measure SPAs: Start Event

SPA frameworks fire routing events when the view is changing:

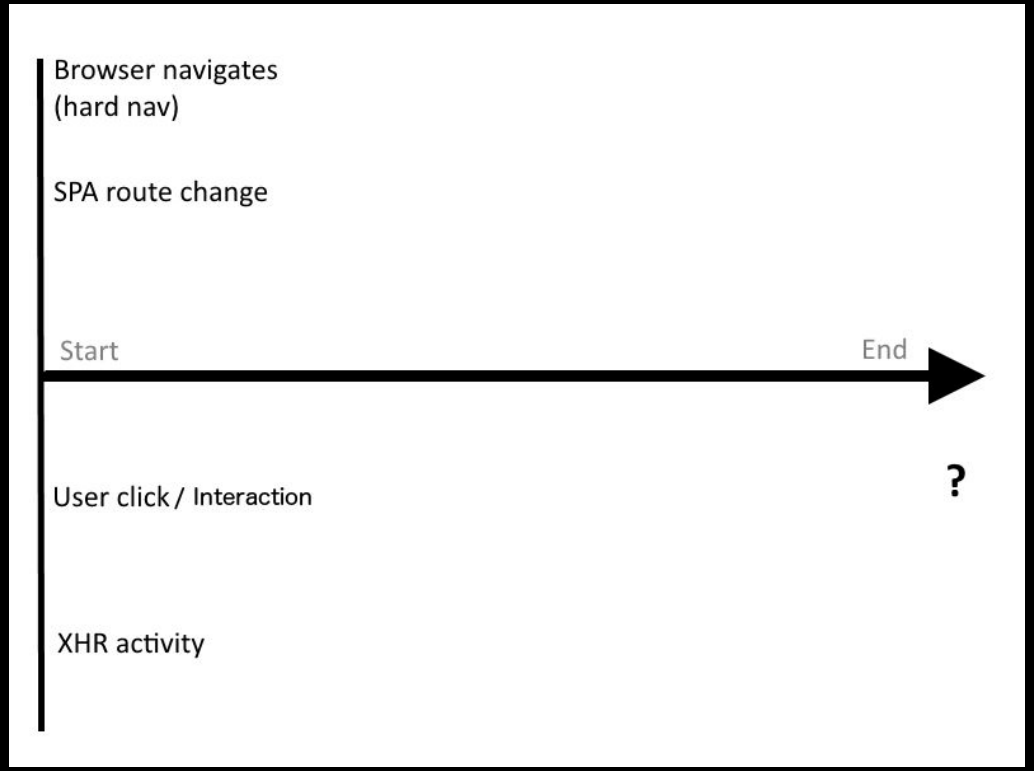
- AngularJS: `$rootScope.$on("$routeChangeStart")`
- Ember.js: `beforeModel` or `willTransition`
- Backbone.js: `router.on("route")`

How to Measure SPAs: Start Event

Clicks & XHRs:

- To determine if a user click or XHR is really triggering a navigation, we can listen to what happens next
- If there was a lot of subsequent network activity, we can keep on listening for more events
- If history (address bar) changed, we can consider the event the start of a navigation
- If the DOM was updated significantly, we can consider the event the start of a navigation
- If nothing else happened, it was probably just an insignificant interaction

How to Measure SPAs: Start Event



How to Measure SPAs: End Event

When do we consider the SPA navigation complete?

- When all networking activity has completed
- When the UI is visually complete (above-the-fold)
- When the user can interact with the page

Remember: `onload` doesn't work:

- Only tracks static resources
- SPA frameworks dynamically load other content
- `onload` doesn't fire for Soft Navs

How to Measure SPAs: End Event

Let's make our own SPA complete event:

- Similar to the body `onload` event, let's wait for all network activity to complete
- This means we will have to intercept both implicit resource fetches (e.g. from new DOM elements) as well as programmatic (e.g. XHR) resource fetches

How to Measure SPAs: Monitoring XHRs

- XHRs are used to fetch HTML, templates, JSON, XML, data and other assets
- We should monitor to see if any XHRs are occurring
- The `XMLHttpRequest` object can be proxied
- Intercept the `.open()` and `.send()` methods to know when an XHR is starting
- Listen to `onReadyStateChange` events to know when it's complete

github.com/lognormal/boomerang/blob/master/plugins/auto_xhr.js

How to Measure SPAs: Monitoring DOM Fetches

- XHR is the main way to fetch resources via JavaScript
- What about Images, JavaScript, CSS and other HTML elements that trigger resource fetches?
- We can't proxy the Image object as that only works if you create a `new Image()` in JavaScript
- If only we could listen for DOM changes...

MutationObserver

developer.mozilla.org/en-US/docs/Web/API/MutationObserver:

- MutationObserver provides developers a way to react to changes in a DOM
- `observe()` for specific events
- Get a callback when mutations for those events occur

How to Measure SPAs: Monitoring DOM Fetches

- Start listening when an XHR, click, route change or other interesting navigation-like event starts
- Use MutationObserver to listen for DOM mutations
- Attach `load` and `error` event handlers and set timeouts on any IMG, SCRIPT, LINK or FRAME
- If an interesting element starts fetching keep the navigation "open" until it completes
- After the last element's resource has been fetched, wait a few milliseconds to see if it kicked off anything else
- If not, the navigation completed when the last element's resource was fetched

github.com/lognormal/boomerang/blob/master/plugins/auto_xhr.js

How to Measure SPAs: Monitoring DOM Fetches

What's interesting?

- Internal and cached resources may not fetch anything, so you have to inspect elements first
- IMG elements that haven't already been fetched (`naturalWidth==0`), have external URLs (e.g. not `data-uri:`) and that we haven't seen before
- `SCRIPT` elements that have a `src` set
- `IFRAMEs` elements that don't have `javascript:` or `about:` protocols
- `LINK` elements that have a `href` set

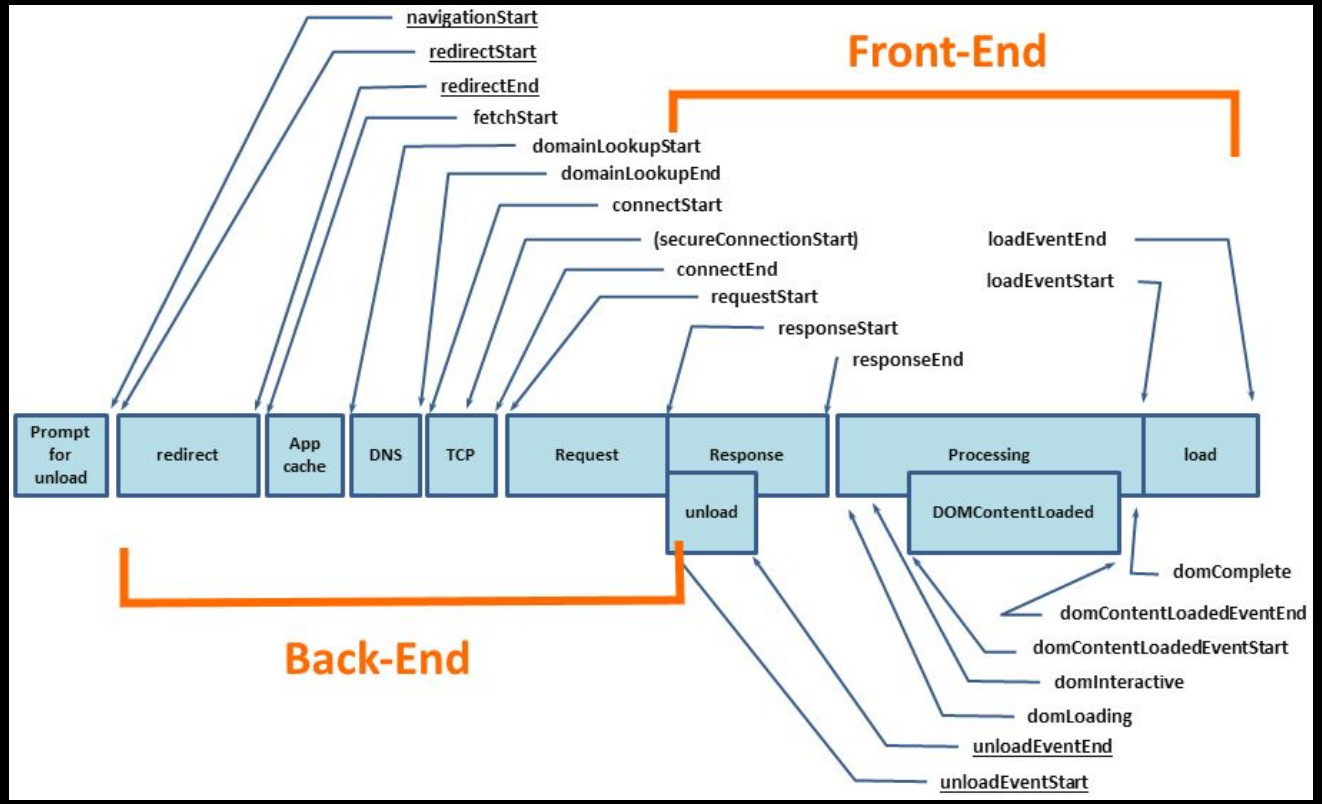
How to Measure SPAs: Monitoring DOM Fetches

Why not ResourceTiming?

- ResourceTiming events are only added to the buffer after they complete
- In order to extend the SPA navigation end time, we have to know if any resource fetches are outstanding
- We can use ResourceTiming later to supplement the data we get from XHR+MO

How to Measure SPAs: Front-End vs. Back-End

Traditional apps:



How to Measure SPAs: Front-End vs. Back-End

Traditional apps:

- Back-End: HTML fetch start to HTML response start
- Front-End: Total Time - Back-End

Single Page Apps:

- Back-End: Any time slice with an `XHR` or `SCRIPT` outstanding
  - Since these are most likely critical path resources
- Front-End: Total Time - Back-End

Accelerated Mobile Pages

AMP: Accelerated Mobile Pages

What is AMP?

- A way to build web pages for improved performance
- Restricts what you can put in your site to achieve this

Components:

- AMP HTML: Similar to HTML5 with restrictions
- AMP JavaScript: JavaScript library you include
- Google AMP Cache: Free CDN

Restrictions

- Cannot include any first- or third-party JavaScript

AMP: RUM

```
<amp-pixel src="http://...">
```

- GET query URL
- Substitution variables to gather metrics:
  - Document info (URL, Canonical URL, Title, Referer)
  - NavigationTiming (TCP, DNS, SSL, Page Load, etc)
  - Navigation Type and Redirect Count
  - Persisted Client ID
  - Total Engaged Time
  - Screen/Viewport dimensions

Example:

```
<amp-pixel src="http://myserver.com/beacon?u=AMPDOC_URL&t=PAGE_LOAD_TIME">
```

AMP: RUM

`<amp-analytics>`

- AMP extension
- Built in vendor configs (> 25)
  - Easy to configure
  - Predefined list of metrics is sent to vendor

Continuity

- We measure everything up to navigation complete (page load or SPA nav)
- We measure whether users bounce or convert

But

- The bulk of user interaction and experience happens after navigation has completed

Which continuous variables can we measure and how?

Developer Tools

Elements Console Sources Network Timeline Resources Security Audits Profiles HTTPS Everywhere EditThisCookie Adblock Plus

Capture: JS Profile Memory Paint Screenshots

500 ms 1000 ms 1500 ms 2000 ms 2500 ms 3000 ms 3500 ms 4000 ms 4500 ms

FPS CPU NET HEAP 28.8 MB – 71.8 MB

600 ms 800 ms 1000 ms 1200 ms 1400 ms 1600 ms 1800 ms 2000 ms 2200 ms 2400 ms 2600 ms 2800 ms 3000 ms 3200 ms 3400 ms 3600 ms 3800 ms 4000 ms 4200 ms 4400 ms 4600 ms 4800 ms

Interactions: 431.0 ms, 185.8 ms, 295.4 ms, 342.6 ms, 163.9 ms, 165.7 ms

Main

Par...]) P...) Evalua...js:1) E... C... (an...on) (... re...js re...re init

JS Heap [28.8 MB – 71.8 MB] Documents [13 – 36] Nodes [2 342 – 5 064] Listeners [95 – 391] GPU Memory

Summary Bottom-Up Call Tree Event Log

Range: 385 ms – 4.81 s

168.2 ms	Loading
2136.1 ms	Scripting
450.1 ms	Rendering
41.0 ms	Painting
554.2 ms	Other
1075.0 ms	Idle

Total: 4.42 s

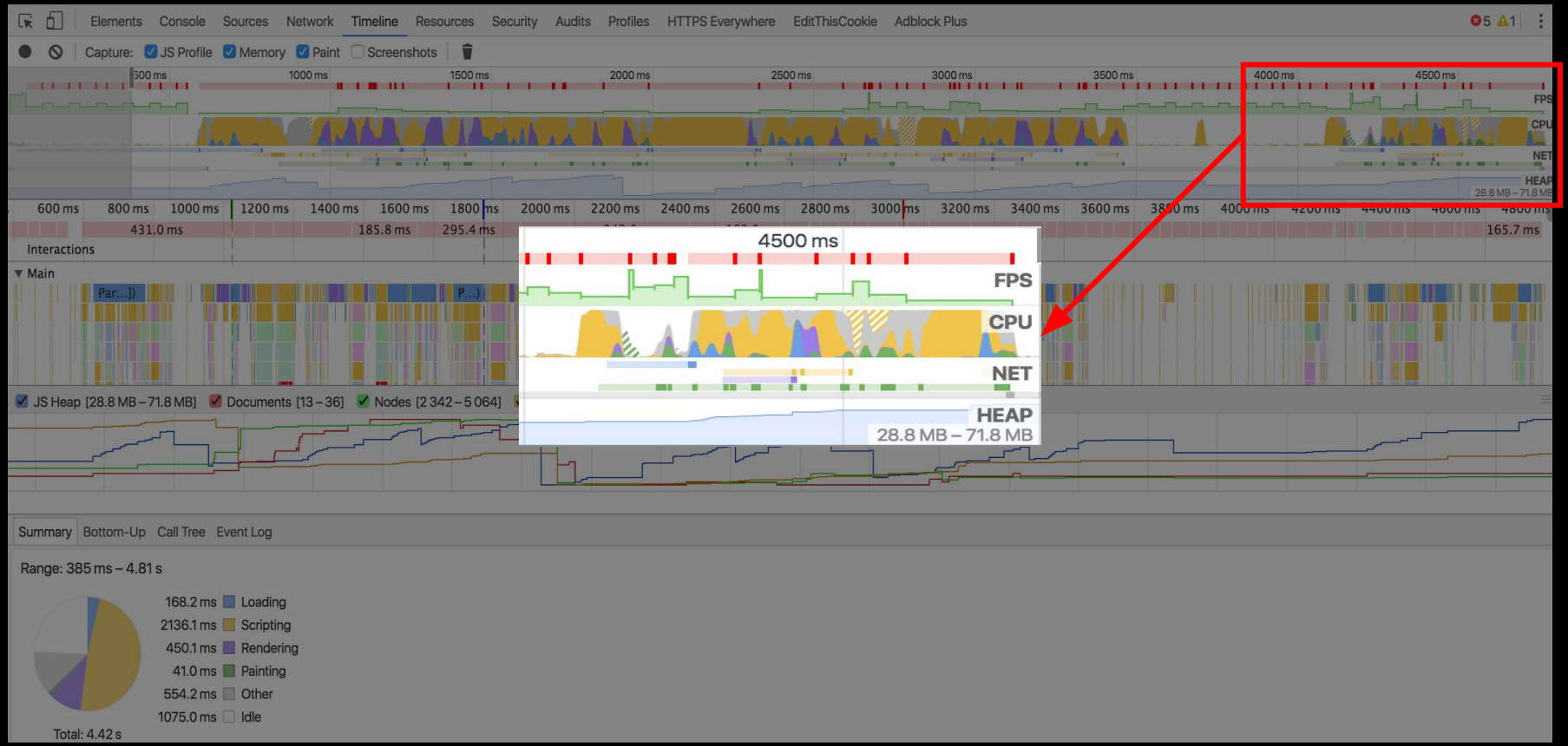
Developer Tools

"The fact that something is possible to measure, and may even be highly desirable and useful to expose to developers, does not mean that it can be exposed as runtime JavaScript API in the browser, due to various privacy and security constraints"

– Performance APIs, Security and Privacy

https://w3c.github.io/perf-security-privacy/

Continuity Metrics



FPS - Frames Per Second

- `requestAnimationFrame(callback)`
- Callback is run before the next paint


```
// total frames seen this second
var frames = 0;

function measureFps() {
  frames++;

  // request a callback before the next frame
  window.requestAnimationFrame(measureFps);
}

// start measuring
window.requestAnimationFrame(measureFps);

// report on frame rate (FPS) once a second
setInterval(function() {
  console.log("FPS: " + frames);
  frames = 0;
}, 1000);
```

FPS - Long Frames

Frames > 16.6 ms lead to < 60 FPS

```
var lastFrame = performance.now();
var longFrames = 0;

function measureFps() {
  var now = performance.now();

  // calculate how long this frame took
  if (now - lastFrame >= 18) { longFrames++; }

  lastFrame = now;

  window.requestAnimationFrame(measureFps);
}
window.requestAnimationFrame(measureFps);

// report on long frames once a second
setInterval(function() {
  console.log("Long frames: " + longFrames);
  longFrames = 0;
}, 1000);
```

FPS - Video

HTML5 VIDEO metrics (Chrome/FF)

```
var latestFrame = 0;
var latestReportedFrame = 0;

setInterval(function() {
  // find the first VIDEO element on the page
  var vids = document.getElementsByTagName("video");
  if (vids && vids.length) {
    var vid = vids[0];
    if (vid.webkitDecodedFrameCount || vid.mozPaintedFrames) {
      latestFrame = vid.webkitDecodedFrameCount || vid.mozPaintedFrames;
    }
  }

  console.log("Video FPS: "
    + Math.max(latestFrame - latestReportedFrame, 0));

  // reset count
  latestReportedFrame = latestFrame;
}, 1000);
```

CPU - Page Busy

- Browser doesn't expose CPU metrics directly
- Detect Busy by running a function at a regular interval
- See if the callback runs at the time we expect
- If the callback was delayed, the page was Busy
- Busy can be caused by other JavaScript, layout, render, etc

CPU - Page Busy

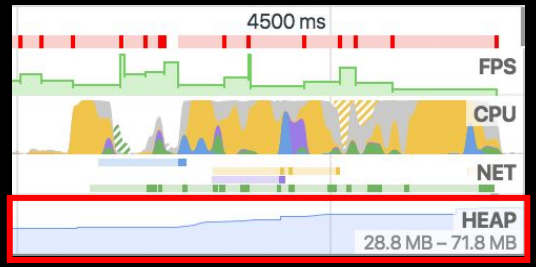
```
setInterval(function() {
  var now = performance.now();
  var delta = now - last;
  last = now;

  // if we are more than 2x the poll
  // + deviation, we missed one period completely
  while (delta > ((POLLING_INTERVAL * 2)
    + ALLOWED_DEVIATION_MS)) {
    total++;
    late++;
    delta -= POLLING_INTERVAL; // adjust, try again
  }

  total++;

  if (delta > (POLLING_INTERVAL + ALLOWED_DEVIATION_MS)) {
    late++;
  }
}, POLLING_INTERVAL);
```

NET - Resources

- ResourceTiming
- Bytes available in ResourceTiming2

```
var resources =
  window.performance.getEntriesByType("resource");

// number of resources fetched
var resourceCount = resources.length;

// number of bytes
var bytesOverWire = 0;
resources.forEach(function(res) {
  bytesOverWire +=
    res.transferSize ? res.transferSize : 0;
});

console.log("Resources: " + resourceCount
  + " " + bytesOverWire + "b");
```

HEAP - Memory Usage



- Non-standard (Chrome only)
- Reduced precision to avoid privacy concerns

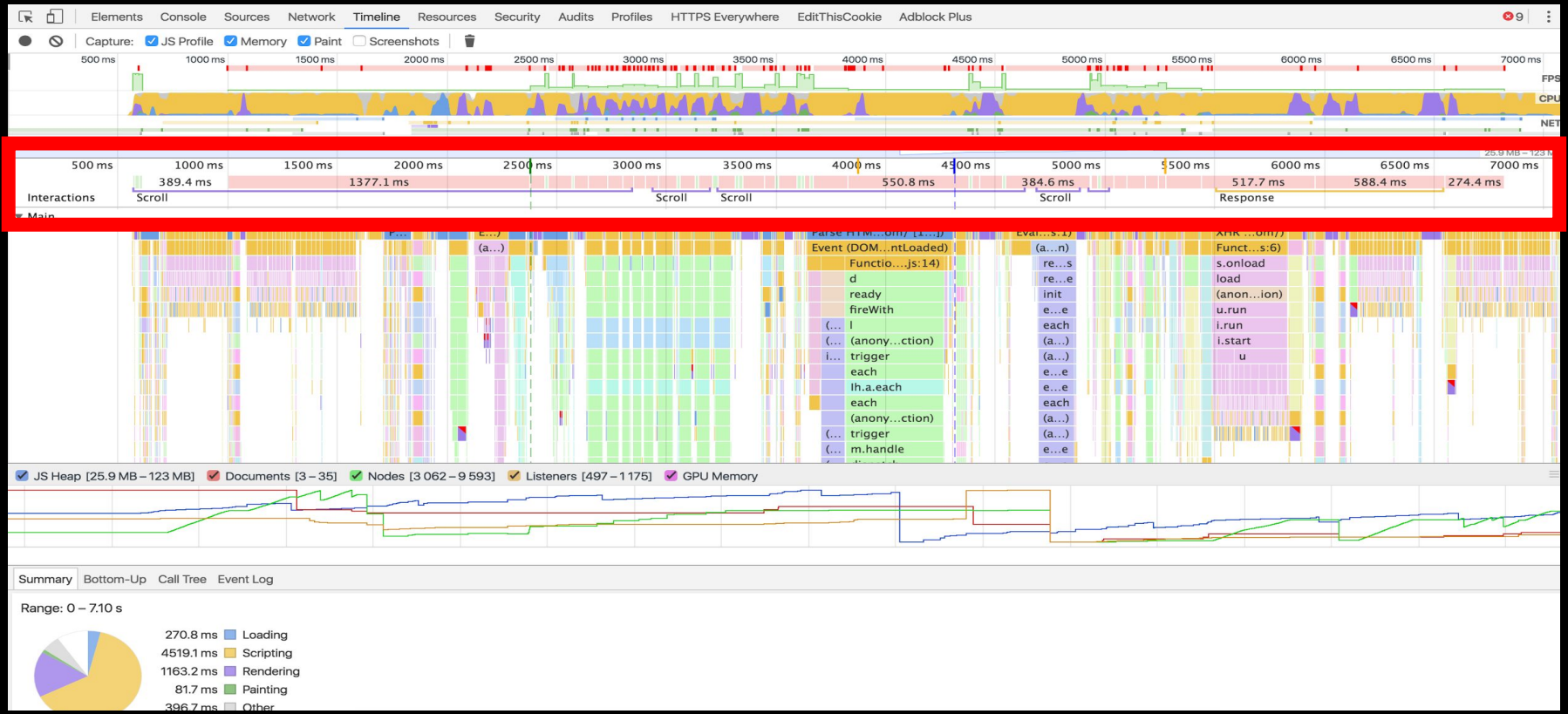
```
// report on JS object memory once a second
setInterval(function() {
  var mem = window.performance
    && window.performance.memory
    && window.performance.memory.usedJSHeapSize;

  console.log("Memory usage: " + mem);
}, 1000);
```

Battery

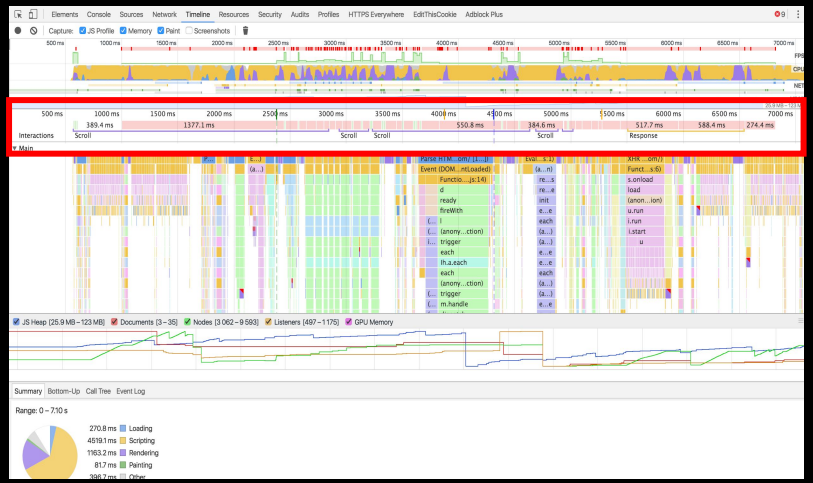
- Monitor your visitor's battery state
- Reduce work on low battery

```
setInterval(function() {
  navigator.getBattery().then(function(batt) {
    console.log(batt.level);
  });
}, 1000);
```

Interactions



Interactions - User Input



- scroll
- mousemove
- click
- keydown

Interactions - Visibility

Window's visibility state

```
document.addEventListener("visibilitychange", function() {
    console.log(document.hidden ? "hidden" : "visible");
}, false);
```

Also look at the `IntersectionObserver`

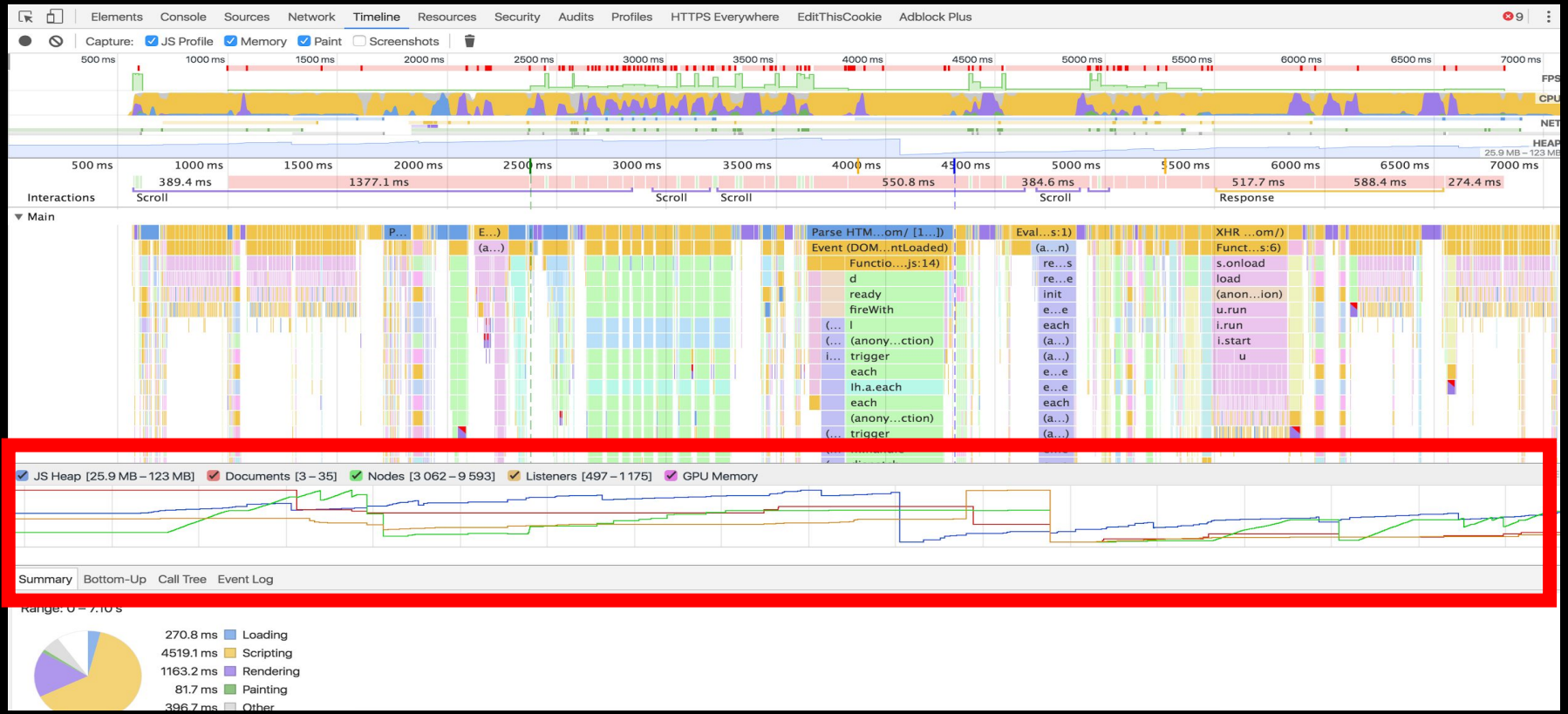
https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API

Interactions - Orientation

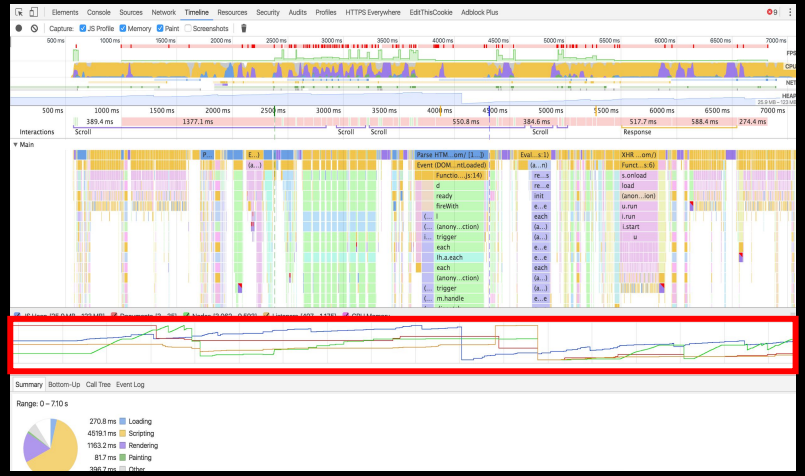
How the device is being held

```
window.addEventListener("orientationchange", function() {
    console.log("orientation: " + screen.orientation.angle);
});
```

Size Metrics



Size - Nodes



- HTML size (bytes)
- Overall Node count
- IFRAME, IMG, SCRIPT, etc., node count

Size - DOM Changes

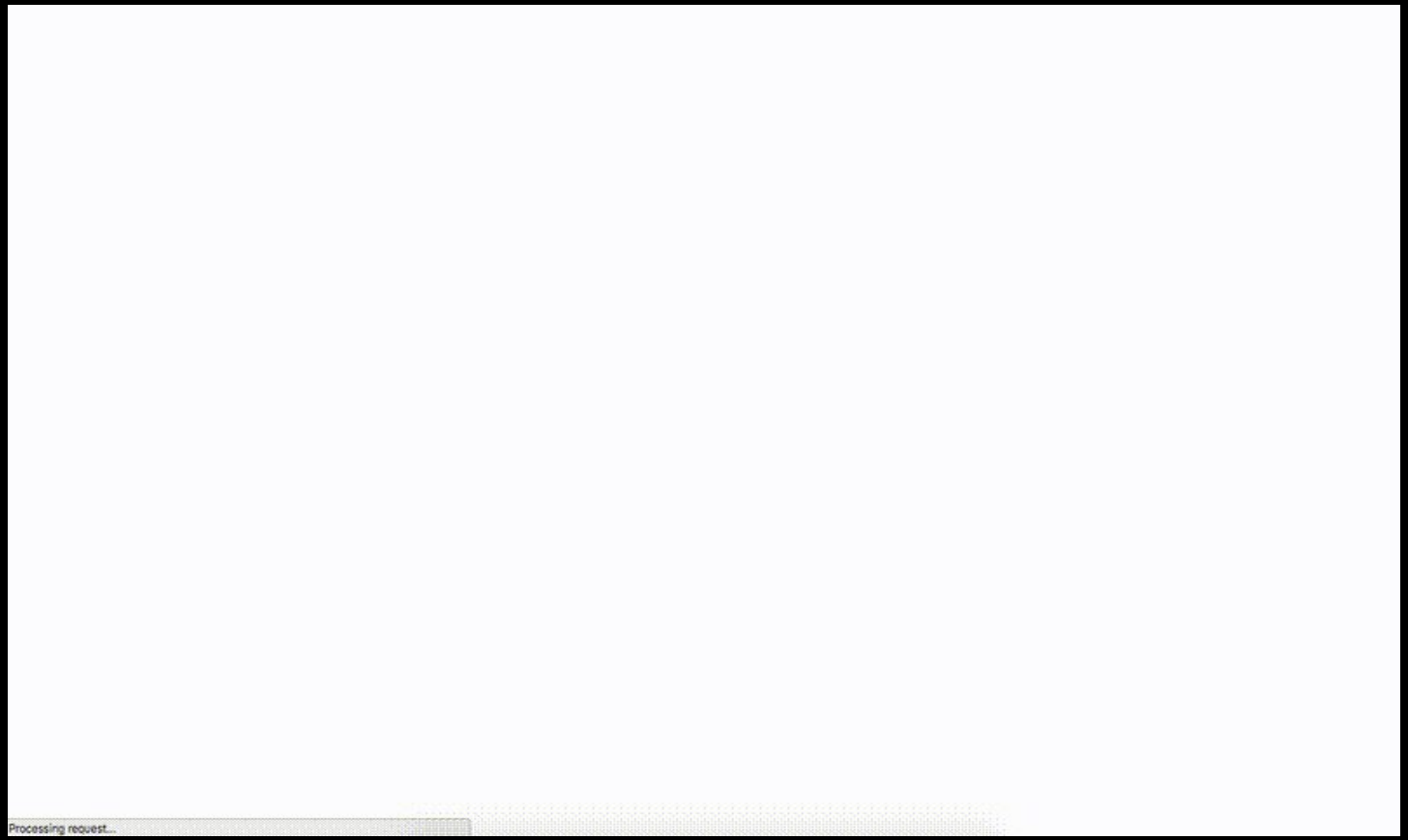
MutationObserver == change over time

```
var d = document;
var mutationCount = 0;
var domLength =
  d.getElementsByTagName("*").length;

// create an observer instance
var observer = new MutationObserver(function(mutations) {
  mutations.forEach(function(mutation) {
    if (mutation.type !== "childList") { return; }
    for (var i = 0; i < mutation.addedNodes.length; i++) {
      var node = mutation.addedNodes[i];
      mutationCount++;
      mutationCount += node.getElementsByTagName ?
        node.getElementsByTagName("*").length : 0;
    }
  });
});

// configure the observer
observer.observe(d, { childList: true, subtree: true });
```

Errors

```
var errorCount = 0;

window.onerror = function () {
  errorCount++;
}

setInterval(function() {
  console.log("Errors: " + errorCount);
  errorCount = 0;
}, 1000);
```

Demo

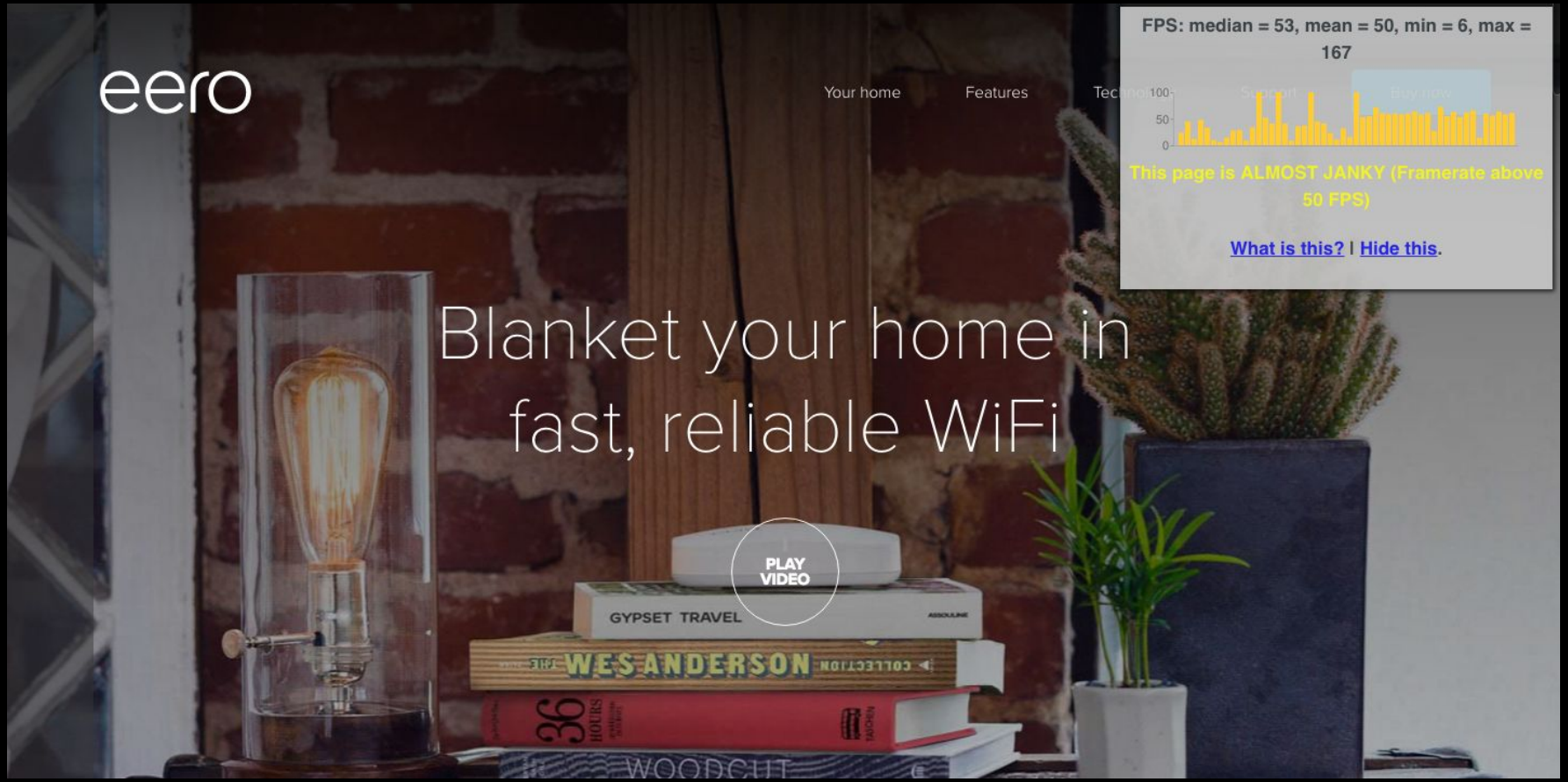


github.com/SOASTA/measuring-continuity

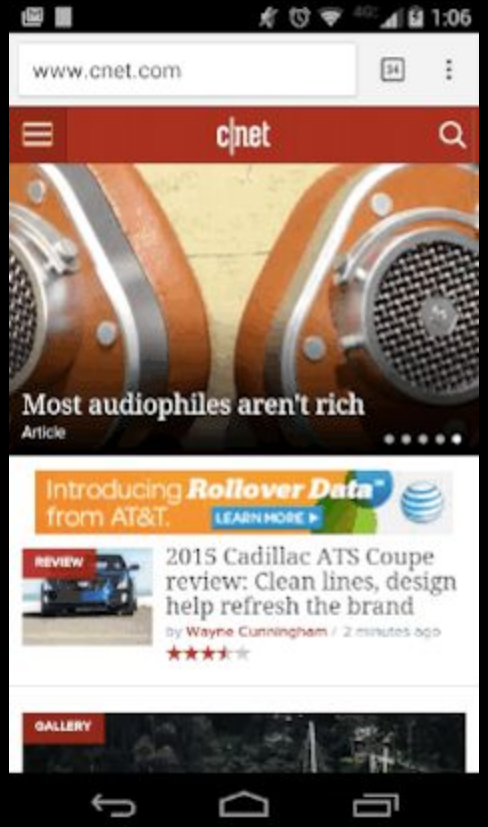
So what?

- Raw data != useful metrics
- Let’s measure the user experience
  - Smoothness
  - Responsiveness
  - Reliability
  - Emotion

Smoothness - FPS during scroll



Smoothness - FPS after interaction



Responsiveness

- How long it takes for the site to respond to input?
  - `requestAnimationFrame` to detect next paint
  - MutationObserver to detect DOM changes
- UserTiming to monitor your own code
- SPA instrumentation via boomerang
- Strive to give feedback within 100 milliseconds of a user interaction!

Responsiveness

```
document.addEventListener("click", function(e) {
  var start = performance.now();
  requestAnimationFrame(function() {
     var delta = performance.now() - start;
     console.log("Click responsiveness: " + delta);
  });
}, false);
```

Responsiveness: Long Task API

- https://github.com/spanicker/longtasks
- Call a callback whenever a task takes too long to complete

Reliability

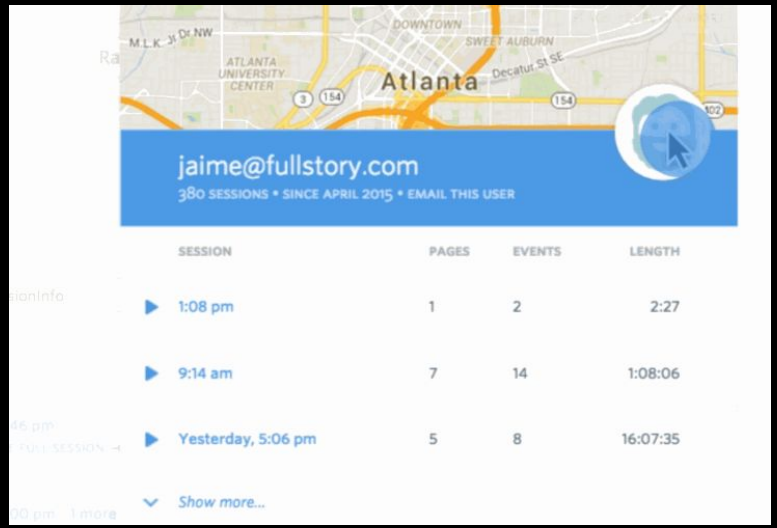
- JavaScript errors
- Leaks:
  - JavaScript memory usage over time
  - DOM size increase over time

Tracking Emotion

Rage Clicks

Rage clicks are series of clicks in which your users are pummeling their mouse buttons in frustration. It's like punching your site in the face, usually because it's not doing what the user wants or expects it to.

– Caitlin Brett, FullStory



blog.fullstory.com/moar-magic-announcing-rage-error-and-dead-clicks-1f19e50a1421

Rage Clicks

```
var same = 0, x = 0, y = 0, targ = null;

document.addEventListener("click", function(e) {
  var nX = e.clientX; var nY = e.clientY;

  // calculate number of pixels moved
  var pixels = Math.round(
    Math.sqrt(Math.pow(y - nY, 2) +
    Math.pow(x - nX, 2)));

  if (targ == e.target || pixels <= 10) {
    same++;
  } else {
    same = 0;
  }

  console.log("Same area clicked: " + same);

  x = nX; y = nY; targ = e.target;
}, false);
```

Dead Clicks

- Clicking without any meaningful visual (DOM) change
- Might happen during (or right after) page load due to delayed JavaScript

Dead Clicks

Missed Clicks

user clicks near an element, but misses it

Mouse Movement

"People who are angry are more likely to use the mouse in a jerky and sudden, but surprisingly slow fashion.

People who feel frustrated, confused or sad are less precise in their mouse movements and move it at different speeds."

– Inferring Negative Emotion from Mouse Cursor Movements

Martin Hibbeln, Jeffrey L. Jenkins, Christoph Schneider, Joseph S. Valacich, and Markus Weinmann

Trusting Your Data

Avoiding the Observer Effect

- JavaScript is Single Threaded (per domain)
- Unless the browser is idle, anything you do in JavaScript will slow down some other JavaScript
- So how do we measure performance without affecting performance?

Avoiding the Observer Effect

Use the IFrame Loader Technique to load measurement code outside the critical path

OR

Load measurement code after the `onload` event
(but then you can't measure things that happen before `onload`)

Avoiding the Observer Effect

- Do as little as possible in event handlers, eg, read a timestamp or save state to a variable
- Do more expensive processing of this data via a `requestIdleCallback` that runs when the browser is idle

requestIdleCallback is only available on Chrome and Opera at the moment, so use a shim for other browsers

- requestIdleCallback API Spec: MDN://docs/Web/API/Window/requestIdleCallback
- Complete, spec compliant SHIM: github://aFarkas/requestIdleCallback
- Minimal Google SHIM (not entirely compliant): github://github/requestIdleCallback

Avoiding the Observer Effect

- The `unload` event is a bit problematic because nothing can be deferred from it, so KiSS*, like writing to `localStorage`
- Anything else should be run deferred from an `onBeforeUnload` handler (which is non-standard and not supported everywhere)
- Also stay away from the `scroll` event or debounce/throttle

https://css-tricks.com/debouncing-throttling-explained-examples/

KiSS: Keep it Short & Simple

and whatever you do...

Never Instrument

Flash!

Beaconing

the mechanics of getting performance data back to you

How to Beacon

There are several methods for sending ("beaconing") data:

- Image beacon
- Form beacon
- XMLHttpRequest beacon
- Beacon API

Image Beacon

Create an `<IMG>` element from JavaScript with your query string data

```
<script>
var img = new Image();
img.src = "http://myserver.com/beacon?pageLoad=100&dns=30";
// your data has been sent!
</script>
```

Pros:

- Easy and lightweight!
- 100% browser support

Hint: Return a `204 No Content` HTTP response

Image Beacon

Cons:

- Must put data on the query string (no POST)
- URL length (payload) limitation:
  - Windows IE 6-8: 2083 b
  - Windows Chrome, Safari, Opera, Firefox, IE9+: >100 kb
  - Mac: Chrome, Opera, Firefox, Safari: >100 kb
  - Android Chrome, Opera: 81,500 b
  - iOS Safari, Chrome: >100 kb
  - Proxies? Beware

Image Beacon: URL Limit

Need to update your server config config for >8 KB URLs:

```
// Apache
// https://httpd.apache.org/docs/2.2/mod/core.html#limitrequestline
// Default: LimitRequestLine 8190
LimitRequestLine 16380

// nginx
// https://nginx.org/en/docs/http/ngx_http_core_module.html#large_client_header_buffers
// Default: large_client_header_buffers 4 8k;
large_client_header_buffers 4 16k;

// JBoss
// https://docs.jboss.org/jbossweb/2.1.x/config/http.html
// Default: maxHttpHeaderSize="8192"
<Connector ... maxHttpHeaderSize="16384"/>
```

Form Beacon

Create a <FORM> element, POST it to a hidden IFRAME

```
var iframe, name = "beacon-" + Math.random();
try {
  // IE <= 8
  iframe = document.createElement('<iframe name="' + name + '">');
} catch (ignore) {
  // everything else
  iframe = document.createElement("iframe");
}

form.action = "https://my-server.com/beacon";
form.target = iframe.name = iframe.id = name;
iframe.style.display = form.style.display = "none";
iframe.src = "javascript:false";

remove(iframe.id);
remove(form.id);


document.body.appendChild(iframe);
document.body.appendChild(form);

try { form.submit(); }
catch (ignore) {}
```

```
function remove(id) {
  var el = document.getElementById(id);
  if (el) {
    el.parentNode.removeChild(el);
  }
}

// cleanup
setTimeout(function() { remove(iframe.id); }, 10000);
```

https://github.com/SOASTA/boomerang/blob/master/boomerang.js#L710-L761

Form Beacon

Server Implementation:

- IE 10 *can* hang if given a 200 response. For best compat:
  - `204 No Content`
  - `Content-Length: 0`
  - `Content-Type: image/gif`
  - `X-XSS-Protection: 0`
- `Timing-Allow-Origin: *`
  - To be able to capture ResourceTiming data
- `Access-Control-Allow-Origin: [*|domain]`
  - If sent from another origin

Form Beacon

Pros:

- POST data > 2,083 bytes (works in IE <= 8)

Cons:

- Complex JavaScript
- Less efficient than an Image beacon
- Lots of potential for browser bugs and incompatibilities! We've seen browser hangs, beacons opening in new windows, beacon URL in the status bar, etc. Use the boomerang.js code to avoid these.

XMLHttpRequest Beacon

Create an `XMLHttpRequest` object

```
<script>
var xhr = new XMLHttpRequest();
xhr.open("GET", "https://my-server.com/beacon?pageLoad=100", true);
xhr.send();
</script>
```

Pros:

- Easy and relatively lightweight!
- GET and POST support
- Large payload support

Hint: Return a `204 No Content` HTTP response

XMLHttpRequest Beacon

Server Implementation:

- For best performance:
  - `204 No Content`
  - `Content-Length: 0`
- `Timing-Allow-Origin: *`
  - To be able to capture ResourceTiming data
- `Access-Control-Allow-Origin: [*|domain]`
  - If sent from another origin

XMLHttpRequest Beacon

Cons:

- Not supported in IE 8/9. Requires `XDomainRequest`
  - GET/POST only
  - No custom headers
  - Only `text/plain` requests
  - No authentication or cookies
  - Restricted to same scheme as host page

Beacon API

How do we guarantee an beacon is sent when the user is leaving the page?

```
window.addEventListener('unload', logData, false);

function logData() {
    var client = new XMLHttpRequest();
    client.open("POST", "/log", false); // third parameter indicates sync xhr. :(
    client.setRequestHeader("Content-Type", "text/plain;charset=UTF-8");
    client.send(analyticsData);
}
```

This is bad because it is synchronous and blocks the browser UI.

Async XHRs and Images can be cancelled in unload.

`onbeforeunload` is not supported by Safari.

Beacon API

Beacon API requests are:

- Prioritized to avoid competition with other UI and higher-priority network requests
- Optimized on mobile devices (may be coalesced)
- Guaranteed to be initiated before page is unloaded

Beacon API: Usage

Simple API:

```
window.addEventListener("visibilitychange", logData, false);

function logData() {
    if (document.visiblityState === "hidden") {
        navigator.sendBeacon("/log", analyticsData);
    }
}
```

Note: Use `visibilitychange` event as `unload` will not fire whenever a page is `hidden` and the process is terminated.

Beacon API: Browser Support

Beacon API - WD

Global 61.45%

Allows data to be sent asynchronously to a server with `navigator.sendBeacon`, even after a page was closed. Useful for posting analytics data the moment a user was finished using the page.

Current aligned | Usage relative | Show all

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
			49						
			51			9.2		4.4	
8	13	47	52			9.3		4.4.4	
11	14	48	53	9.1	39	10	all	52	51
		49	54	10	40				
		50	55	TP	41				
		51	56						

Notes | Known issues (0) | Resources (4) | Feedback

No notes

How to Beacon

We recommend:

- Use `sendBeacon()` if available
- If not, use `Image` beacon if < 2,000 bytes
- If not, use `XMLHttpRequest` if available and > 2,000 bytes
- If not, consider using FORM beacons or just shrug and move on

When to Beacon

Depends on your use-case:

- As soon as you can to be the most reliable

- For general analytics:
  - As soon as you load if you're not waiting for perf metrics

- For performance analytics:
  - After `onload` or SPA complete to gather all relevant performance metrics

- For continuous metrics or session length:
  - On `pagehide` if supported
  - On `beforeunload` if supported
  - On `unload` as a last resort (avoid using sync XHR)

Nixing Noise

getting rid of obviously absurd data

Getting Rid of Noise

- Look at simple things like data type & range of all the data you collect
- Don't trust client timestamps, only trust deltas
- Check for a reasonable rate of data input per client
- Validate that your data collector isn't CSRFed
- Segregate known bot traffic (well behaved bots)

Some common bogus things we see

- All timer values are set to 9999999999
- Client timestamps are more than a day, year or 30+ years in the past
- Requests that do not change over time
- Requests that are very regularly spaced
- Client makes exactly the same high valued purchase repeatedly
- Page Load time is negative or more than a week long
- Measured bandwidth in Terabits/second

Getting Rid of Noise

- We also use statistical methods to identify Outliers
- Use MAD or Tukey's method to identify data that is outside the expected range
- Nelson Rules to check for sufficient randomness
- *3e-Smoothing* to compare actual values with expected values
- Don't throw away outliers, analyze them separately

MAD: Median Absolute Deviation: wikipedia://Median_absolute_deviation

John Tukey's fences: datapigtechnologies.com/.../highlighting-outliers-...-with-the-tukey-method/

Nelson Rules: wikipedia://Nelson_rules

Offline First

measuring without network connectivity

Offline First

- ResourceTiming includes `workerStart` that tells us when a ServiceWorker that intercepted a request started
- Our measurement code should also run as a ServiceWorker, queuing up beacons while offline…
- But how do we distinguish these queued beacons from forged beacons without an unexpired anti-CSRF token?

This is something we're still experimenting with, so we don't have any concrete recommendations, but we invite you to join the experiment.

W3C WebPerf Working Group

www.w3.org/2010/webperf

Founded in 2010 to give developers the ability to assess and understand performance characteristics of their web apps:

"The mission of the Web Performance Working Group is to provide methods to measure aspects of application performance of user agent features and APIs"

Microsoft, Google, Mozilla, Opera, Facebook, Netflix, Akamai, SOASTA, etc

API Reference

- Hi Res Timer
  https://w3c.github.io/hr-time/
- Navigation Timing
  http://www.w3.org/TR/navigation-timing
  developer.mozilla.org/.../Web/API/Navigation_timing_API
- Resource Timing
  https://www.w3.org/TR/resource-timing/
  developer.mozilla.org/.../Web/API/Resource_Timing_API
- Resource Timing Compressor
  http://nicj.net/compressing-resourcetiming
- User Timing
  https://www.w3.org/TR/user-timing/
  developer.mozilla.org/.../Web/API/User_Timing_API
- User Timing Polyfill
  https://github.com/nicjansma/usertiming.js
- User Timing Compressor
  http://nicj.net/compressing-usertiming/
- Page Visibility
  https://w3c.github.io/page-visibility/
  developer.mozilla.org/.../Web/API/Page_Visibility_API
- Service Workers
  https://www.w3.org/TR/service-workers/
  developer.mozilla.org/.../Web/API/Service_Worker_API
- requestAnimationFrame
  https://www.w3.org/TR/animation-timing/
  developer.mozilla.org/.../Web/API/window/requestAnimationFrame
- requestIdleCallback
  https://www.w3.org/TR/requestidlecallback/
  developer.mozilla.org/.../Web/API/Window/requestIdleCallback
- Spec compliant requestIdleCallback SHIM
  github://aFarkas/requestIdleCallback
- Minimal requestIdleCallback SHIM
  github://github/requestIdleCallback
- Mutation Observer
  developer.mozilla.org/.../Web/API/MutationObserver
- Performance Observer
  https://www.w3.org/TR/performance-timeline-2/
  developer.mozilla.org/.../Web/API/PerformanceObserver
- Intersection Observer
  https://wicg.github.io/IntersectionObserver/
  developer.mozilla.org/.../Web/API/Intersection_Observer_API
- Beacon API
  https://w3c.github.io/beacon/
  developer.mozilla.org/.../Web/API/Navigator/sendBeacon
- W3C Web Performance Working Group
  http://www.w3.org/2010/webperf

Further Reading

- Boomerang
  https://github.com/SOASTA/boomerang
- Andy Davies' Waterfall Bookmarklet
  https://github.com/andydavies/waterfall
- Mark Zeman's Heatmap
  https://github.com/zeman/perfmap
- Progressive Web Metrics
  https://github.com/paulirish/pwmetrics
- Accelerated Mobile Pages
  https://www.ampproject.org/
- The IFrame Loader Technique
  http://www.lognormal.com/blog/2012/12/12/the-script-loader-pattern/
- Affectiva Emotion Analyzer
  https://github.com/affectiva/youtube-demo
- MAD: Median Absolute Deviation
  https://en.wikipedia.org/wiki/Median_absolute_deviation
- John Tukey's fences
  datapigtechnologies.com/.../highlighting-outliers-...-with-the-tukey-method/
- Nelson Rules
  https://en.wikipedia.org/wiki/Nelson_rules
- Exponential Smoothing
  grisha.org/blog/2016/01/29/triple-exponential-smoothing-forecasting/
- Rage Clicking
  http://blog.fullstory.com/2015/12/reducing-ux-rage-with-fullstorys-rage-clicks/
- Inferring Emotion from Mouse Movements
  telegraph://technology/.../Websites-could-read-emotions-by-...-move-your-mouse.html
- Scroll Behaviour
  http://blog.chartbeat.com/2013/08/12/scroll-behavior-across-the-web/
- WebGazer: Eye tracking in JavaScript
  http://webgazer.cs.brown.edu/
- What JavaScript knows about you
  http://webkay.robinlinus.com/
- Video Metrics
  https://wiki.whatwg.org/wiki/Video_Metrics
- The Runtime Performance Checklist
  http://calendar.perfplanet.com/2013/the-runtime-performance-checklist/
- Jank Meter
  https://webperf.ninja/2015/jank-meter/
- RAIL Performance Audit of SFGate.com
  https://docs.google.com/document/d/1K-mKOqiUiSjqZTEscBLjtjd6E67oiK8H2ztOiq5tiqk
- Debouncing and Throttling Events
  https://css-tricks.com/debouncing-throttling-explained-examples/

Photo Credits

Angel Delight *by* Auntie P
https://www.flickr.com/photos/auntiep/360764980/

Frustrated *by* Kevin Lawver
https://www.flickr.com/photos/kplawver/1903240219/

Thank You

http://slideshare.net/nicjansma/measuring-real-user-performance-in-the-browser

SOASTA

Philip Tellis
@bluesmoon

Nic Jansma
@nicj

https://github.com/SOASTA/boomerang

http://www.soasta.com/mpulse

http://slideshare.net/nicjansma/measuring-real-user-performance-in-the-browser