Web Performance and Optimisation

Toby Dussek

Framework Training

June 8-9 2021

Times

- 9:30 start
- 11:00 break
- 12:30 lunch
- 3:00 break
- 4:30 end
- There is an end of course evaluation

Welcome

- What we will cover
  - https://www.frameworktraining.co.uk/courses/coding/web-apps-and-performance-tuning/web-performance-and-optimisation-training-course

Where we are at

- Existing experience
- Ideas, strategies, experience
- Known bottlenecks and work arounds
- Content known to be slow
- Tools, URL and resources

Strategies

- Content Delivery Networks (cdn)
- Minification
- Cacheing
- Lazy Loading
- Use Sprites
- Emojis
- Don’t request things that are not used
  - E.g. only enable third-party metrics that are actually in use

Time and Perception

Users Expect

- Up To One Second load
- Then no perceived lag
- Responses within 100ms
- Animations within 16ms (10-12ms)

Bottlenecks

- 50% of your 1-second page load time budget on mobile is taken up by network latency overhead

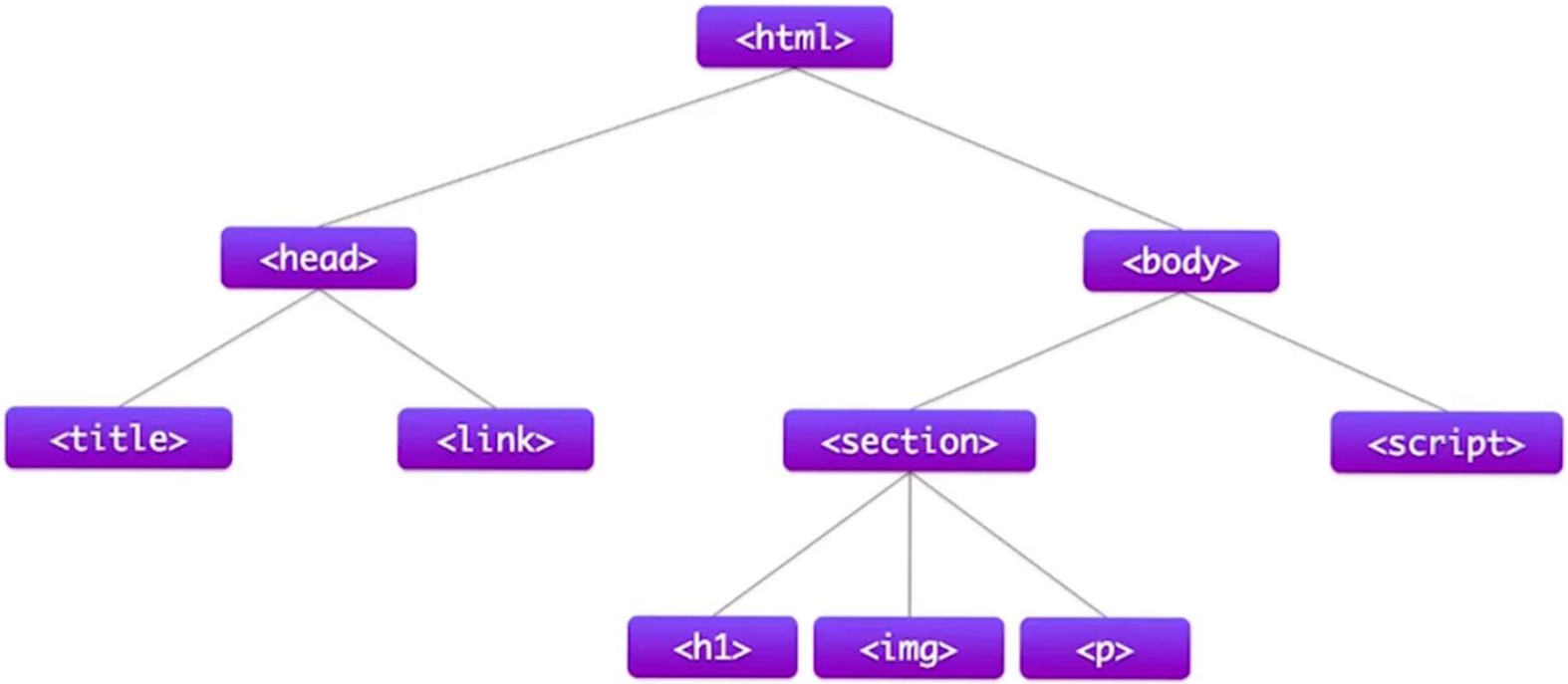
Idle

- Between interactions we have idle time we can use
- 50ms

Device Refresh

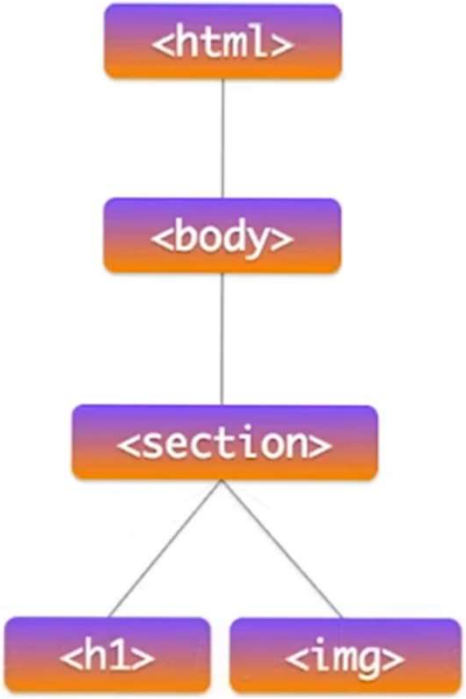
- Typically 60Hz
- Aim for 60fps
- So we have 1/60th of a second per frame
- In milliseconds, that is….
- …less any time for housekeeping
- gc

The DOM is parsed



The Render Tree is created

```
section p {
  display: none;
}
```

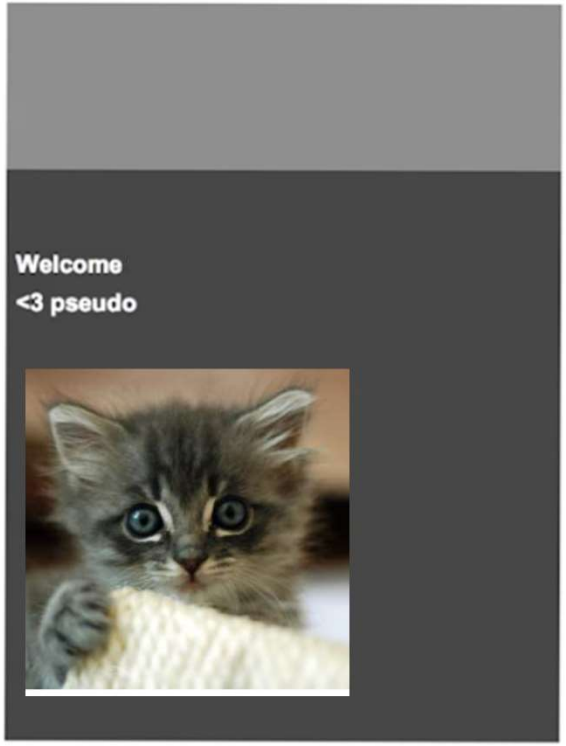


Layout is calculated (Reflow)

```
html, body {
  margin: 0;
  width: 550px;
  height: 730px;
  font-family: Arial;
  background: white;
  color: white;
}

body {
  background: #888;
}

section {
  display: block;
  margin-top: 30%;
  padding-top: 60px;
  left: 0;
  width: 100%;
  height: 70%;
  background: #444444;
}

section h1:after {
  content: '<3 pseudo';
  height: 40px;
  margin-top: 10px;
  display: block;
}

img {
  margin: 30px;
  border-radius: 4px;
  border: 3px solid white;
  box-shadow: 0 2px 2px rgba(0,0,0,0.3);
}
```

Layout is Rasterized (Paint into Layers)

1. save
2. translate
3. drawRectangle
4. drawRectangle
5. drawRectangle
6. drawText
7. drawText
8. save
9. clipPath
10. drawRoundedRectangle
11. restore
12. drawPath
13. save
14. clipRoundedRectangle
15. drawBitmap
16. restore
17. translate
18. restore

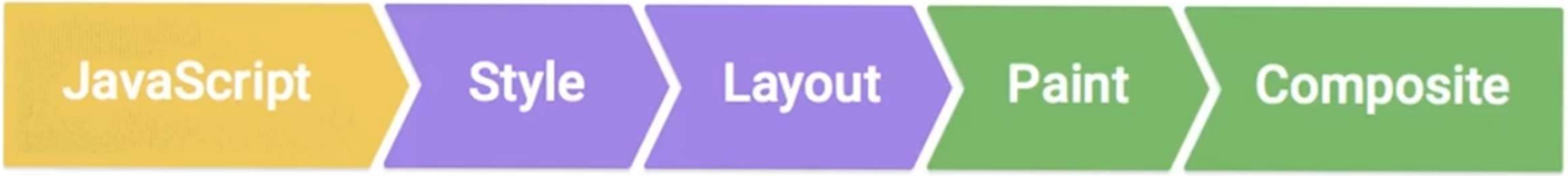


Layout Paint Demo

- http://udacity.github.io/60fps/lesson1/layoutPaint/index.html

Development

Typical Frame Flows



- Javascript or CSS initiates the frame changes
- All steps could fire
- Layout is skipped if no geometry changes
- Layout and Paint are skipped if only need to Composite
- See https://csstriggers.com/

Eliminate Unnecessary Content

- Measure the performance of each asset (inc. 3rd party assets)
- Determine if they are providing value for their performance

Optimize Assets

- CSS
- Fonts
- Images
  - WebP, avif
- Audio
- Video
- ‘above the fold’

CSS

- Give browser a clue that an element is going to be changed, e.g.

```
.someElement {
    will-change:transform
}
```

- Browser will make a layer in advance, to handle this for us
  - Use with transform, left, top, width, height or any visual property
- Pointless for color changes etc.

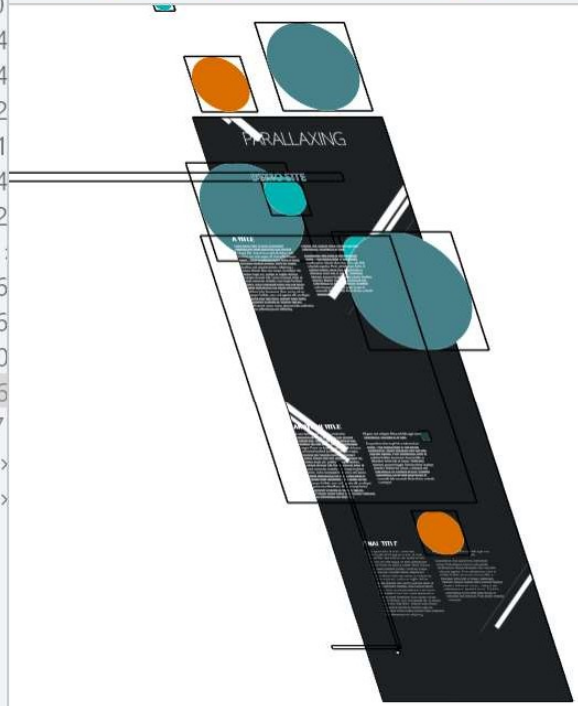
Slow stuff | Slow stuff | Parallax Demo - rAF + DOM + W | Will-Change Quiz | JS Bin | Parallax Demo - DOM + Absolute

html5rocks.com/static/demos/parallax/demo-1a/demo.html

Elements | Console | Sources | Network | Performance | Memory | Layers

Slow scroll rects | Paints

#6(0 × 0)
#document(960
img#blob-1(454
img#blob-2(284
img#blob-3(202
img#blob-4(101
img#blob-5(504
img#blob-6(202
img#blob-7(50
img#blob-8(226
img#blob-9(606
h2#demosite(30
h2#demosite(96
#document(347
#document(17
#document(17



FINAL TITLE

Lorem ipsum dolor sit amet, co
adipiscing elit. Morbi eget puru
volutpat felis. Sed et lectus nisl
Vivamus non ante augue, sit a
magna. Donec eu dapibus sap
vel massa hendrerit porttitor. M
faucibus sed vulputate lacinia,
massa. Mauris vitae sem nequ
Nullam turpis est, porttitor et sa
volutpat sit amet velit. Donec e
amet commodo molestie, nunc
tortor, varius consequat massa
Morbi eleifend risus sed mauris
ultrices tortor fermentum. Nunc
tempor facilisis, nunc orci eges
purus eros vitae libero. Aliquan
hendrerit id lobortis at, faucibus
Praesent ornare massa placera
eu pellentesque est adipiscing.

Details

Size	960 × 1374 (at 0, 0)
Compositing Reasons	Secondary layer, home for a group of squashable content.
Memory estimate	5.3 MB
Slow scroll regions	

19:22 04/06/2021

Optimize Data

- Use short-codes
- Choose efficient data formats
- Remove comments
- Reduce headers
- Compress (after minification)
  - All modern browsers support GZIP compression
  - Server must be configured to enable GZIP compression

Example

- Original (200 characters)
  - # Below is a secret message, which consists of a set of headers in
  - # key-value format followed by a newline and the encrypted message.
  - format: secret-cipher
  - date: 08/05/21
  - AAAZZBBBBEEEMMM EEETTTAAA
  - Color:blue
- Reduced (56 characters)
  - format: secret-cipher
  - date: 08/05/21
  - 3A2Z4B3E3M 3E3T3A
  - Color:#0000ff or color:#00f

Slow stuff | Slow stuff | Google

google.co.uk/?gws_rd=ssl

About Store Gmail Images Sign in

Google

Google Search I'm Feeling Lucky

United Kingdom

Carbon neutral since 2007

Advertising Business How Search works

Privacy Terms Settings

Elements Console Sources Network Performance Memory Application

Preserve log Disable cache No throttling

Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other

Has blocked cookies Blocked Requests

10000 ms 20000 ms 30000 ms 40000 ms 50000 ms 60000 ms 70000 ms 80000 ms 90000

Name	Status	Type	Initiator	Size	Time	Waterfall
?gws_rd=ssl	200	document	Other	37.1 kB	1.09 s	

37.1 kB transferred over network, resource size: 118 kB

1 / 20 requests | 37.1 kB / 37.5 kB transferred | 118 kB / 1.4 MB resources | Finish: 1.3 min | DOMContentLoaded: 1.14 s | Load: 1.14

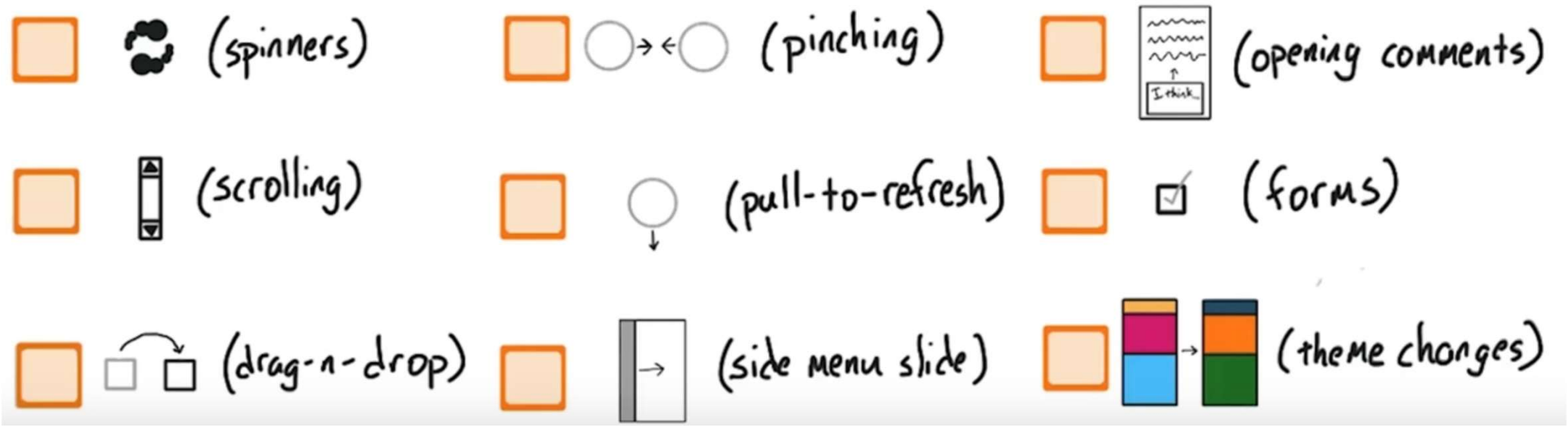
17:30 04/06/2021

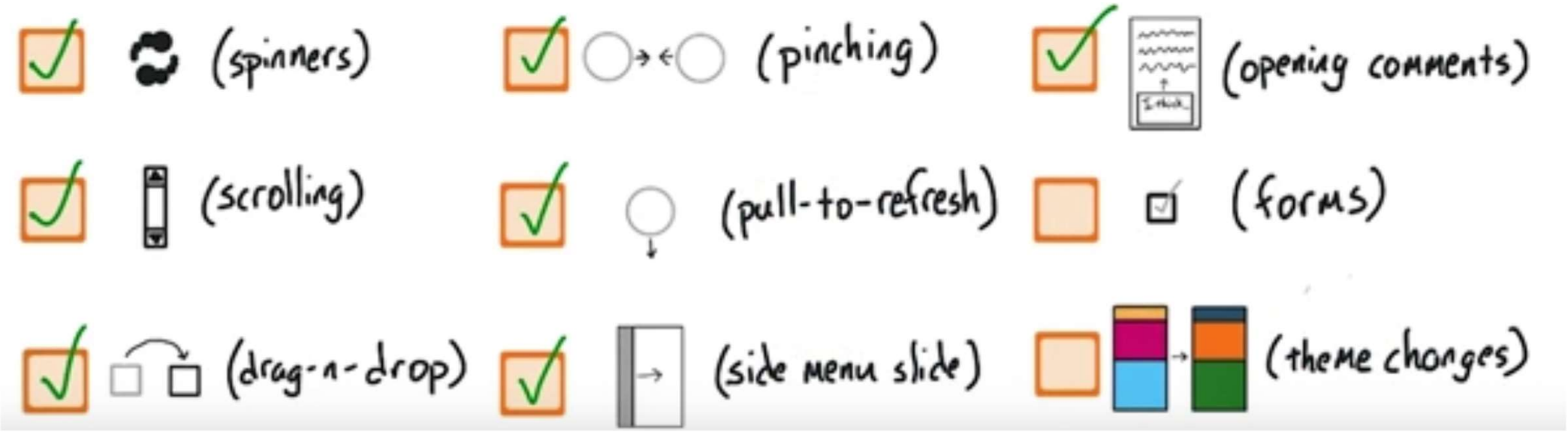
Optimize Animations

- First-Last-Invert-Play
- Use ‘this’ scope
- Use transform and opacity
- Use requestAnimationFrame

```
function animate() {

    // Do something

    requestAnimationFrame(animate);
}

requestAnimationFrame(animate);
```

Which Interactions are Expensive?




(spinners)
(pinching)
(opening comments)
(scrolling)
(pull-to-refresh)
(forms)
(drag-n-drop)
(side menu slide)
(theme changes)

When…

	Response	Animation	Idle	Load
Threshold	**100ms**	**10ms**	**50ms chunks**	**1000ms**
Asset load / parse	**Avoid**	**Avoid**	*Unknown*	400ms
JS: Parse	**Avoid**	**Avoid**	*Unknown*	30ms
JS: Execute	15ms	3ms	*Unknown*	60ms
JS: GC	**Avoid**	**Avoid**	*Unknown*	20ms
Blink: Style Calcs	10ms	1ms	*Unknown*	25ms
Blink: Layout	15ms	3ms	*Unknown*	90ms
Blink: Layer Management	10ms	2ms	*Unknown*	10ms
Blink: Paint	5ms	**Avoid**	*Unknown*	20ms
Compositor: Rasterize	30ms	**Avoid**	*Unknown*	100ms
Compositor: Image Decode	**Avoid**	**Avoid**	*Unknown*	180ms
Compositor: Image Resize	**Avoid**	**Avoid**	*Unknown*	55ms
Composite	10ms	2ms	*Unknown*	10ms

https://speakerdeck.com/paullewis/making-a-silky-smooth-web

Async and Defer

- Async downloads in the background without blocking

```
// load example.js without interrupting your webpage's rendering
<script src="example_async.js" async></script>
```

- Defer downloads after the rest of the page has loaded

```
// load example.js after the page has finished loading
<script src="example_defer.js" defer></script>
```

Persist

- let r = ()=>{}

HTTP 1 2 and 3

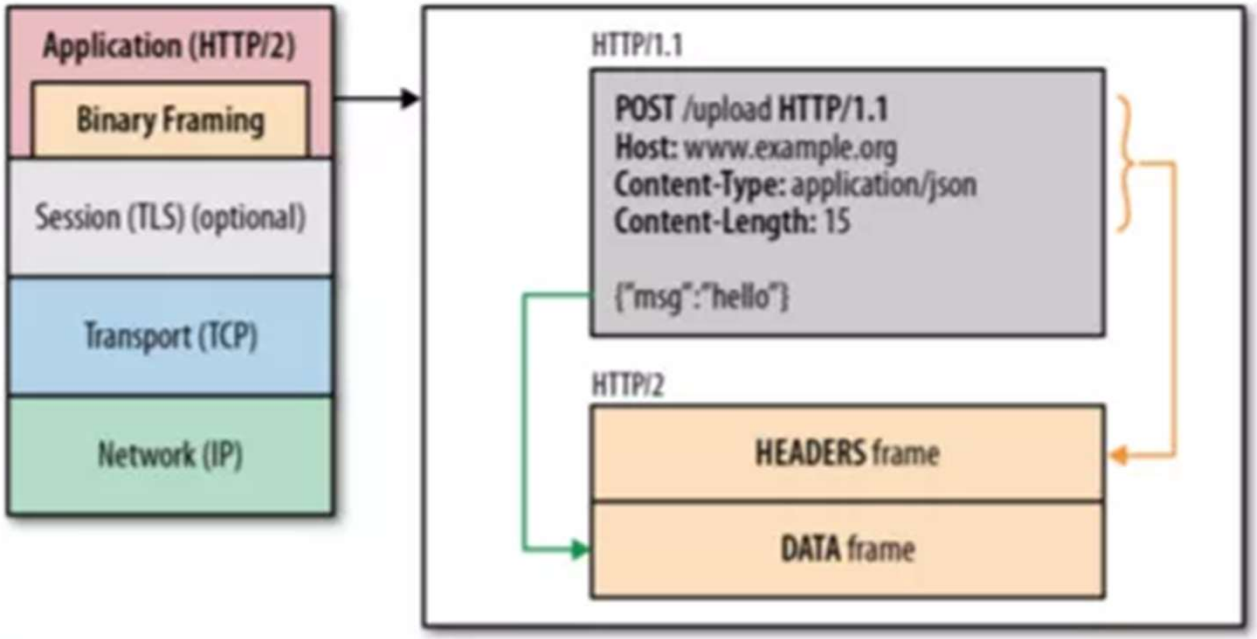
- HTTP/1
  - Without which we wouldn't have the current internet
  - Standard URIs, headers, methods, protocols and status codes
- HTTP/2 https://en.wikipedia.org/wiki/HTTP/2
  - Supported in all major browsers
  - Encryption is not mandated but browsers insist on https
  - Decrease latency via header compression, server push, pipelining and multiplexing requests
  - Deliver expected content before browser requests it (e.g. css)
- HTTP/3 https://en.wikipedia.org/wiki/HTTP/3
  - Chrome and Firefox support as of May 2021
  - Native multiplexing: lost packets only impact streams where data was lost

HTTP/1.1 superseded by HTTP/2

- HTTP/1.1
  - Pages use many TCP connections each with its own client-server request
  - No header compression means potentially large HTTP request headers
  - Content is text not binary
  - Domain sharding and concatenation strategies try to address these problems
- HTTP/2
  - Content is binary not text
  - Multiple requests and responses are sent at the same time (Multiplexing)
    - Client can use just one connection per origin
  - Headers are compressed
  - Server Push avoids delays by pushing responses the client may need to cache

HTTP/2 https://www.keycdn.com/blog/keycdn-http2-support

1. **One TCP connection**
2. **Request → Stream**
   - Streams are multiplexed
   - Streams are prioritized
3. **Binary framing layer**
   - Prioritization
   - Flow control
   - Server push
4. **Header compression (HPACK)**



Chrome Developer Tools

https://developer.chrome.com/docs/devtools/evaluate-performance/reference/

Recommended

- Close other apps
- Use Incognito mode
- Look for causes not symptoms
- Measure then optimize

Chrome Dev Tools Performance Tutorial

https://developer.chrome.com/docs/devtools/speed/get-started/

- https://glitch.com/edit/#!/tony
- Remix Project and show in it's own tab
- Open Dev tools (undocked)
- Run Lighthouse audit
  - Notice bundle.js header has no content-encoding
  - Follow tutorial to enable server compression
  - Reload and now content-encoding is gzip
  - Alter src/model to specify 'small' instead of 'big' images
  - Eliminate render-blocking resources (in this case, un-used code)
  - Do less on the main thread (in this case, use production mode)

Relative-Sized Images

- https://developers.google.com/web/fundamentals/design-and-ux/responsive/images#relative_sized_images
- Instead of fixed image sizes, specify the size of each image with a width descriptor
- The browser automatically calculates the pixel density and chooses which image to download

```
<img src="lighthouse-200.jpg" sizes="50vw"
     srcset="lighthouse-100.jpg 100w, lighthouse-200.jpg 200w,
             lighthouse-400.jpg 400w, lighthouse-800.jpg 800w,
             lighthouse-1000.jpg 1000w, lighthouse-1400.jpg 1400w,
             lighthouse-1800.jpg 1800w" alt="a lighthouse">
```

Chrome Dev Tools Example

- https://googlechrome.github.io/devtools-samples/jank/
- Keep adding until slow
- Compare optimized version
- See Performance Tab

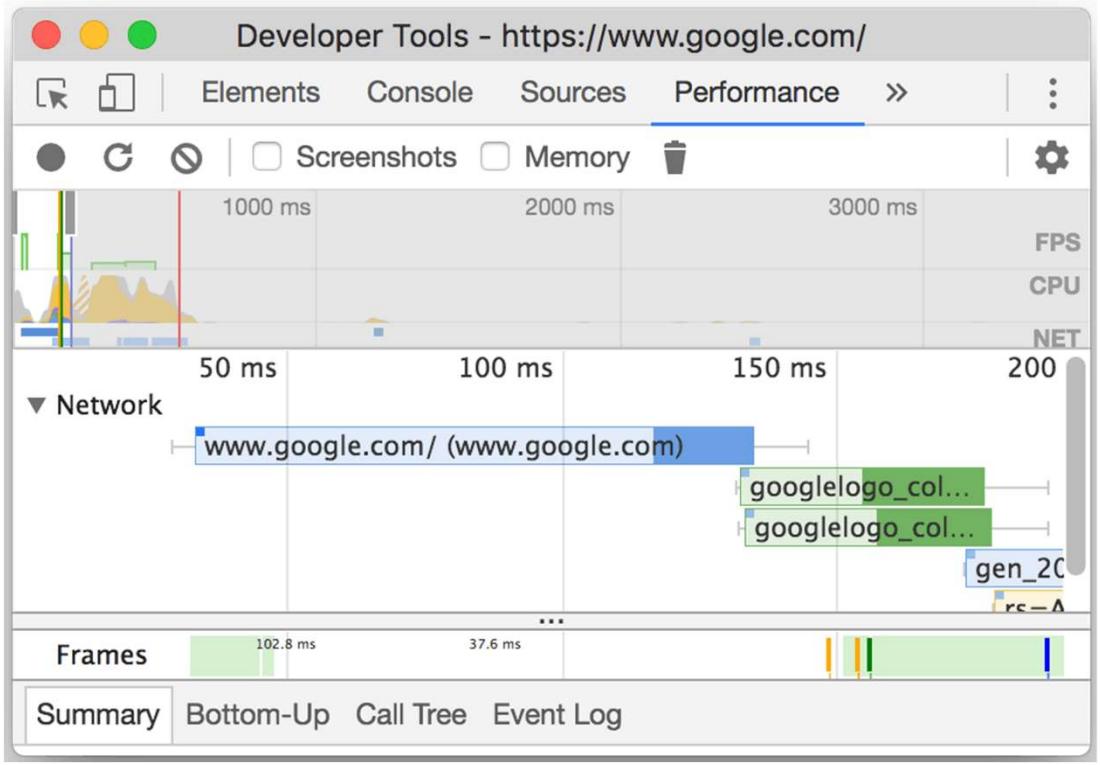
Chrome Dev Tools Performance Reference

- https://developer.chrome.com/docs/devtools/evaluate-performance/reference
- Main
  - the main thread
- Call Tree
  - Root activities that cause the most work
- Bottom-Up
  - Activities where the most time was directly spent
- Event Log
  - Activities in the order in which they occurred

Chrome Dev Tools: Network Requests

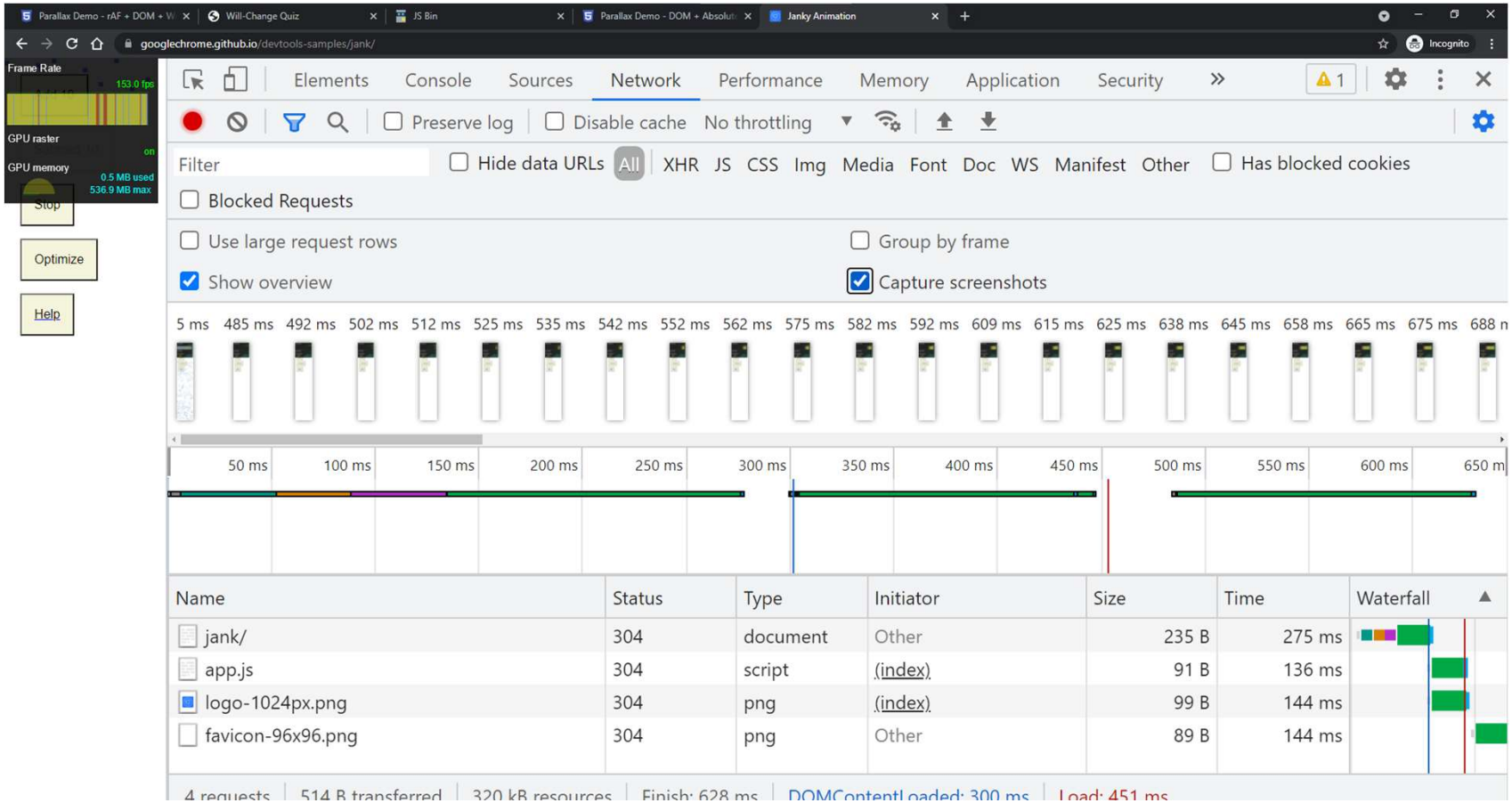
- Left line
  - everything up to Connection Start
  - (everything before Request Sent)
- Light portion
  - Request Sent and Waiting (TTFB)
- Dark portion
  - Content Download
- Right line
  - Time waiting for main thread

- HTML: Blue
- CSS: Purple
- JS: Yellow
- Images: Green



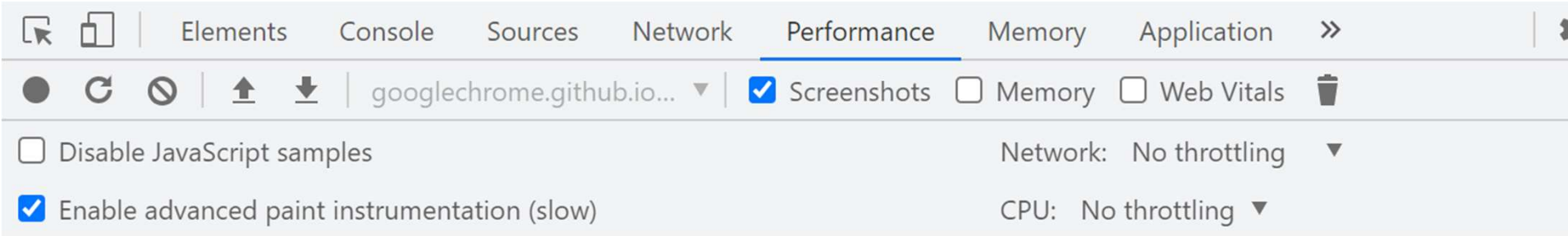
Network filmstrip

- The network filmstrip grabs screenshots while loading

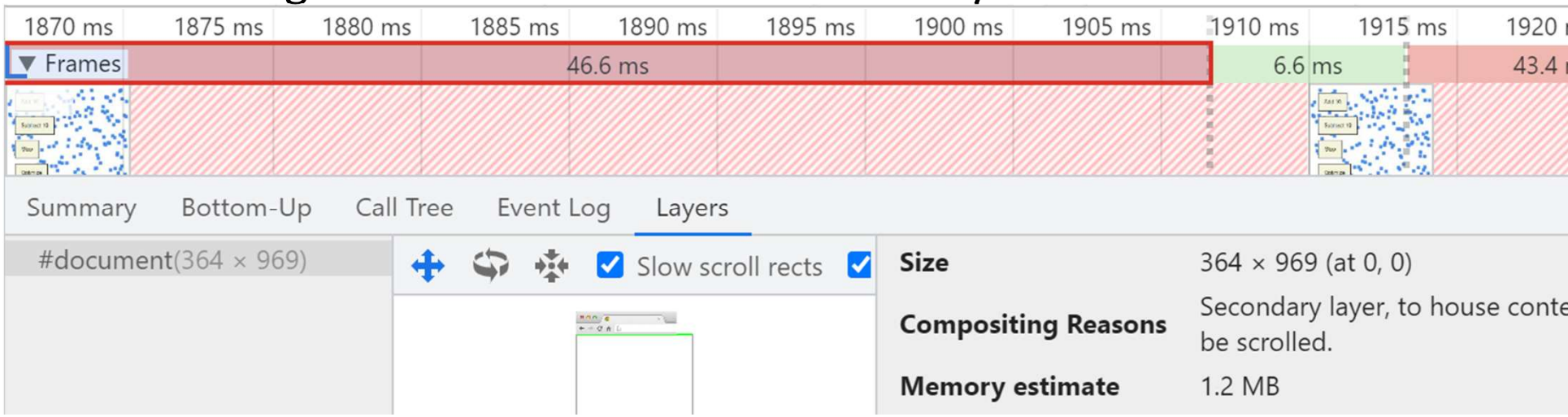


Chrome Dev Tools: View layers information

- Enable advanced paint instrumentation before recording



- Then selecting a frame will show that frames layers in a tab



Parallax Demo - rAF + DOM + W | Will-Change Quiz | JS Bin | Parallax Demo - DOM + Absolute | Janky Animation

googlechrome.github.io/devtools-samples/jank/

Frame Rate 39.0 fps
GPU raster
GPU memory 2.3 MB used 536.9 MB max

Stop
Optimize
Help

Elements | Console | Sources | Network | Performance | Memory | Application | Security | Layers

#4(0 × 0)
#document(206 × 969

Slow scroll rects | Paints

Details

Size	206 × 969 (at 0, 0)
Compositing Reasons	Secondary layer, to house co
Memory estimate	798 kB
Paint count	4864

Console | Performance monitor | Rendering | Animations

100% | 25% | 10%

Waiting for anim

0

Animations
Changes
Coverage
Developer Resources
Issues
JavaScript Profiler
Layers
Media
Memory Inspector
Network conditions
Network request blocking
Performance monitor
Quick source
Rendering
Search
Security
Sensors
WebAudio
WebAuthn
What's New

Dock side

Hide console drawer Esc
Search Ctrl + Shift + F
Run command Ctrl + Shift + P
Open File Ctrl + P
More tools
Shortcuts
Help

Parallax Demo - rAF + DOM + W | Will-Change Quiz | JS Bin | Parallax Demo - DOM + Absolute | Janky Animation

googlechrome.github.io/devtools-samples/jank/

Frame Rate 186.0 fps

GPU raster on

GPU memory 0.9 MB used 536.9 MB max

Start

Optimize

Help

Elements | Console | Sources | Network | Performance | Memory | Application | Security

googlechrome.github.io... | Screenshots | Memory | Web Vitals

500 ms | 1000 ms | 1500 ms | 2000 ms | 2500 ms | 3000 ms

FPS

CPU

NET

1870 ms | 1875 ms | 1880 ms | 1885 ms | 1890 ms | 1895 ms | 1900 ms | 1905 ms | 1910 ms | 1915 ms | 1920 ms

Main — https://googlechrome.github.io/devtools-samples/jank/

Task | Task

Animation Frame Fired | Updat... Tree | P...t | Animation Frame Fired

Summary | Bottom-Up | Call Tree | Event Log | Paint Profiler

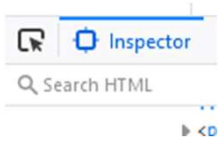
1.13 ms

- drawImageRect(1024, 1024, false, 6050, 0, 0, 1024, 1024, 348, 256, 359, 267, paint)
- drawImageRect(1024, 1024, false, 6050, 0, 0, 1024, 1024, 349, 172, 360, 183, paint)
- drawImageRect(1024, 1024, false, 6050, 0, 0, 1024, 1024, 351, 85, 361, 96, paint)
- drawImageRect(1024, 1024, false, 6050, 0, 0, 1024, 1024, 352, 40, 363, 51, paint)
- drawRect(23, 23, 110, 77, paint)
- drawRect(23, 23, 110, 25, paint)
- drawRect(23, 75, 110, 77, paint)
- drawPath(path, paint)
- drawPath(path, paint)

Total blocking time: 7.77ms (estimated) Learn more

14:59 06/06/2021

HTML and the DOM

- HTML is the initial page hierarchy
- DOM is the same, until it is altered by script
- Inspect DOM elements with right-click or 

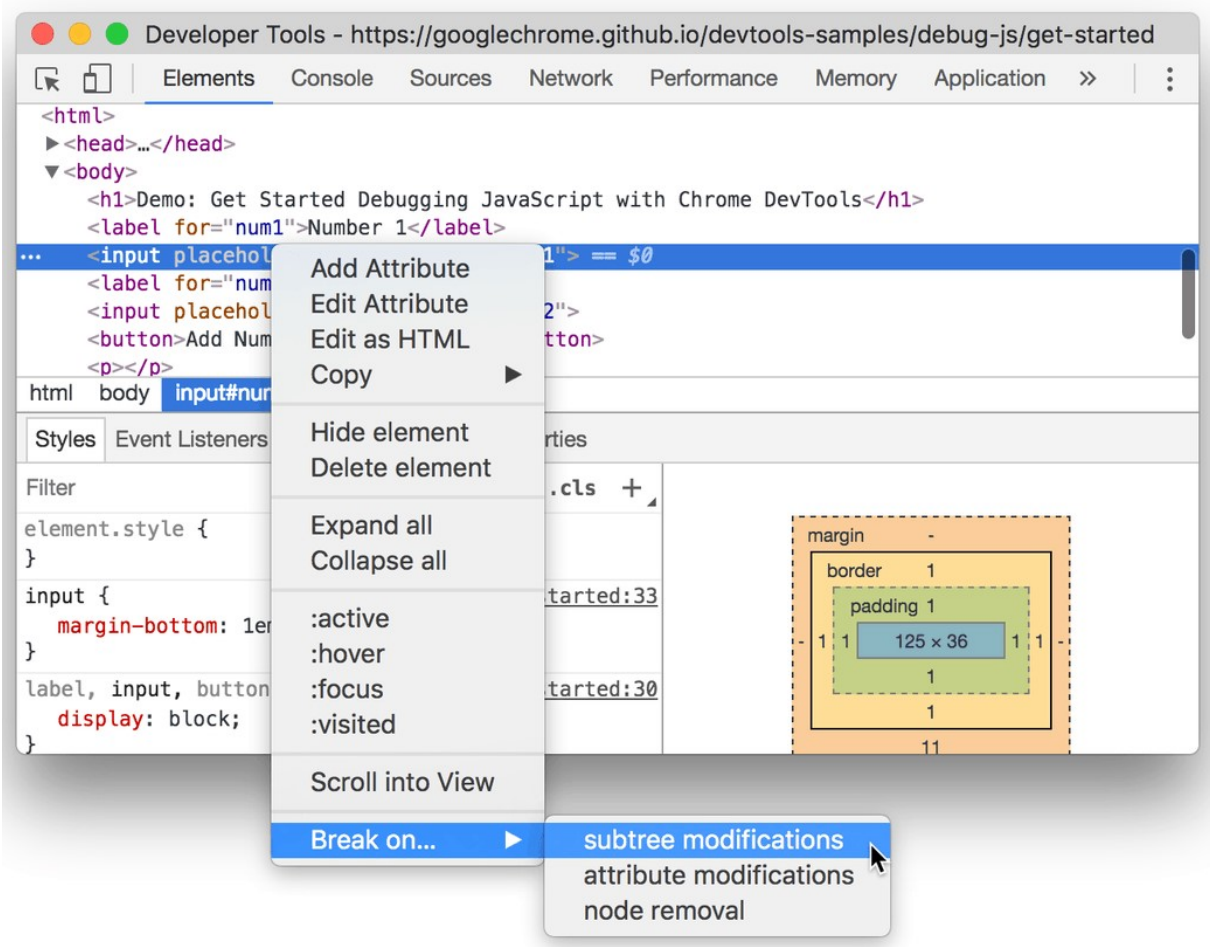
- Then:
  - Navigate with arrows keys
  - Double-click to edit elements, attributes or values in the DOM
  - Drag to re-order elements
  - Press ‘H’ to hide an inspected node
  - Delete nodes

Access nodes in the Console

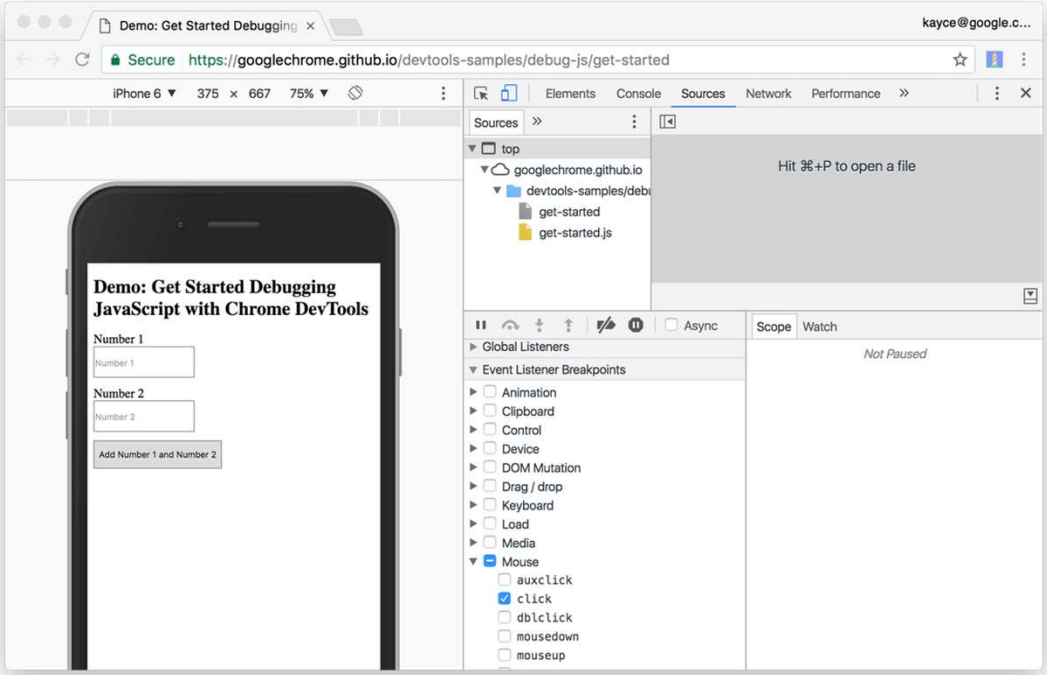
- Inspected node shows == $0
  - You can reference this node in the Console with the variable $0
- <esc> opens console in a drawer
- Right-click a node to store as global variable (type a variable name)
  - Careful – can be tricky to locate this menu option

Break on DOM changes

- You can break when JavaScript modifies the DOM



Break on Event



https://developer.chrome.com/docs/devtools/javascript/

Optimize website speed

- Establish a baseline
- Understand your report
- https://developer.chrome.com/docs/devtools/

Other Links

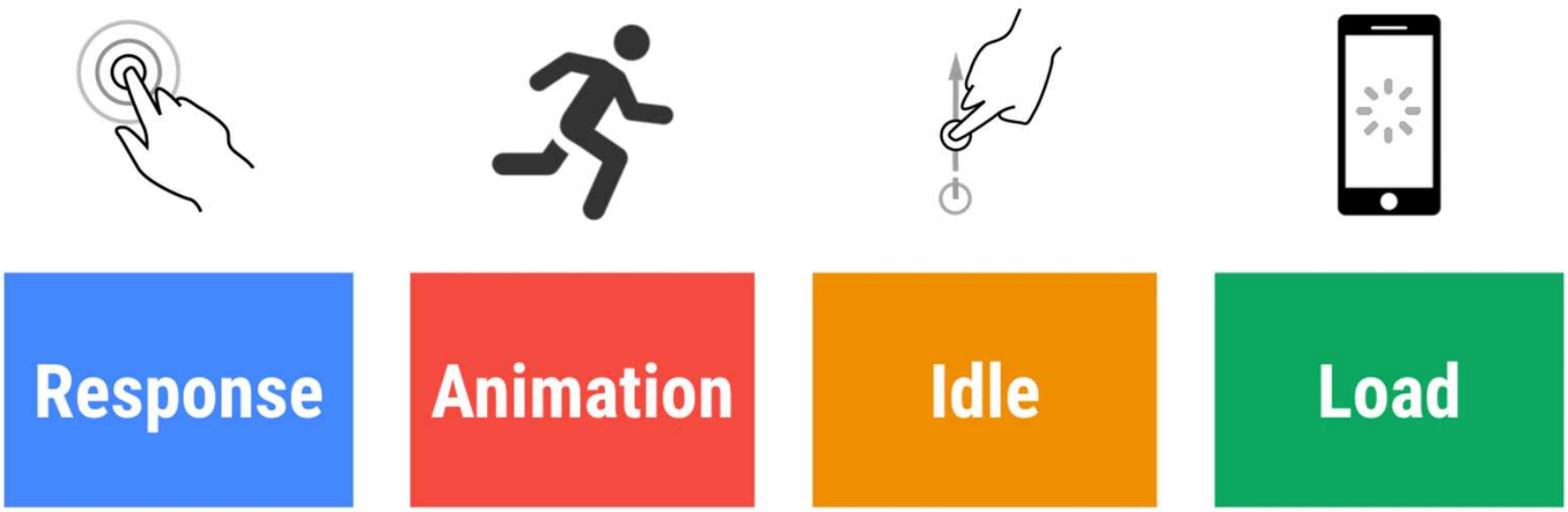
- https://www.keycdn.com/blog/chrome-devtools
  - Very good brief vids
- https://developers.google.com/web/tools/lighthouse/
  - Official project lighthouse docs

Performance

- https://developer.chrome.com/docs/devtools/evaluate-performance/
- https://developers.google.com/web/fundamentals/performance/rendering

The RAIL model

https://web.dev/rail/



Dev Tools connected to Android

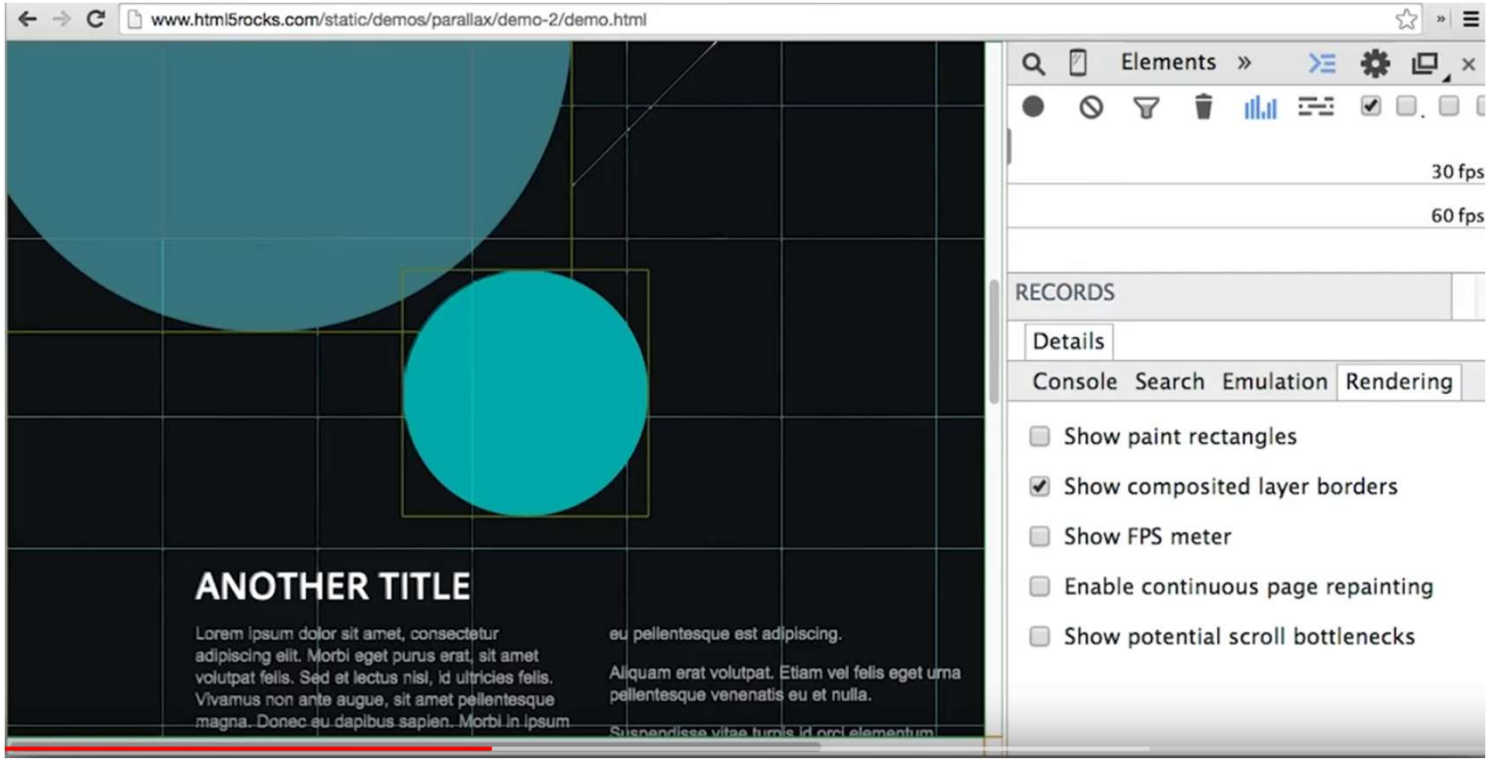
- https://classroom.udacity.com/courses/ud860/lessons/4146058564/concepts/4222787740923 (free sign-up)
- Android device in Developer mode
- Connect USB
- chrome:inspect (on laptop)
- Open tabs them inspect and screen-cast

Avoid Micro-Optimizing

- Modern browsers run JIT (or AOT) and will optimize code
- Animate with requestAnimationFrame
- Maybe also look at the ECMAScript run loop, setImmediate, set…
- Use JavaScript profiler only when you know there is long running JS
- Use CSS instead of JavaScript
- Compare add/remove nodes with hiding them

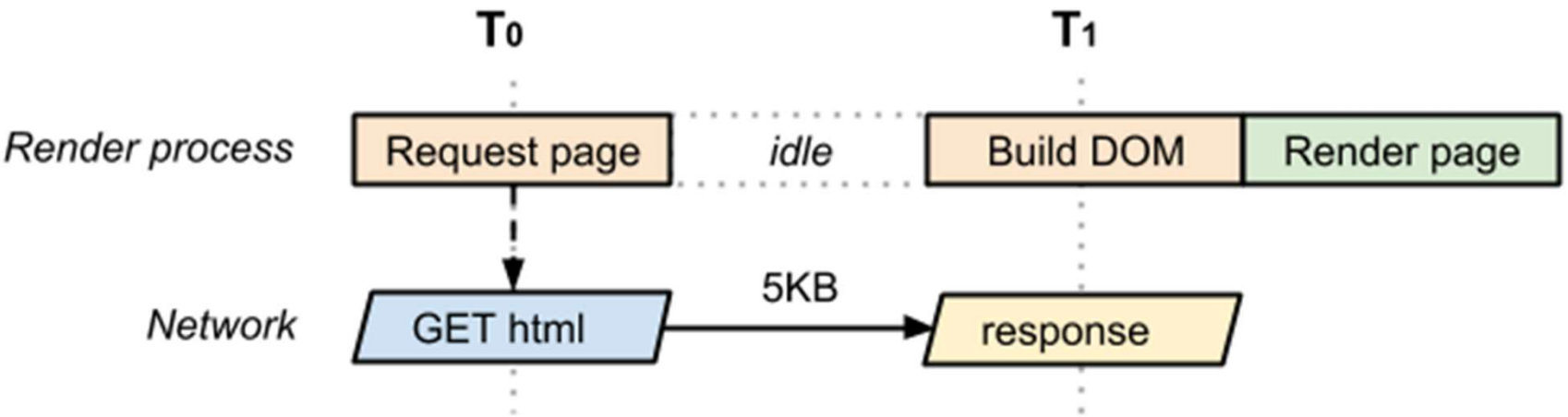
See Layers

https://classroom.udacity.com/courses/ud860/lessons/4129068601/concepts/41395386460923



Critical Path

- See https://developers.google.com/web/fundamentals/performance/critical-rendering-path/analyzing-crp.html



End of Course Evaluation

- Link to the course evaluation: https://frameworktraining.typeform.com/to/AC7BzIRT