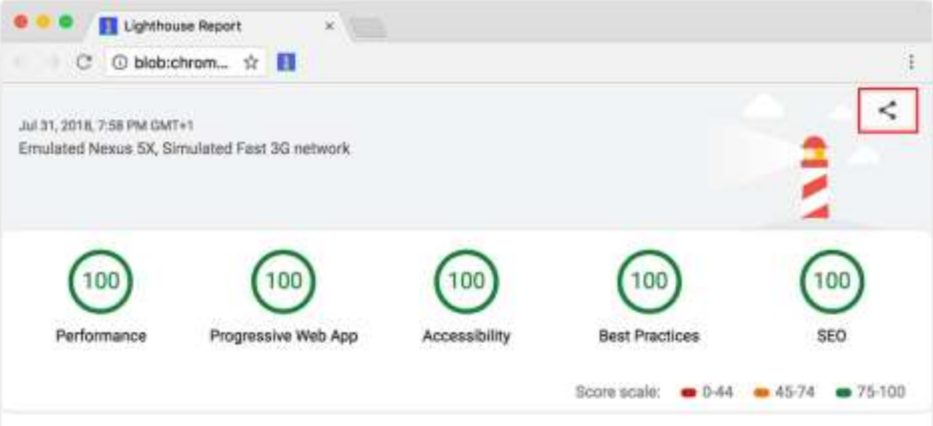
Profiling JavaScript Like a Pro

What users expect



Jul 31, 2018, 7:58 PM GMT+1
Emulated Nexus 5X, Simulated Fast 3G network

100 Performance

100 Progressive Web App

100 Accessibility

100 Best Practices

100 SEO

Score scale: 🟥 0-44 🟧 45-74 🟩 75-100

What users sometimes get

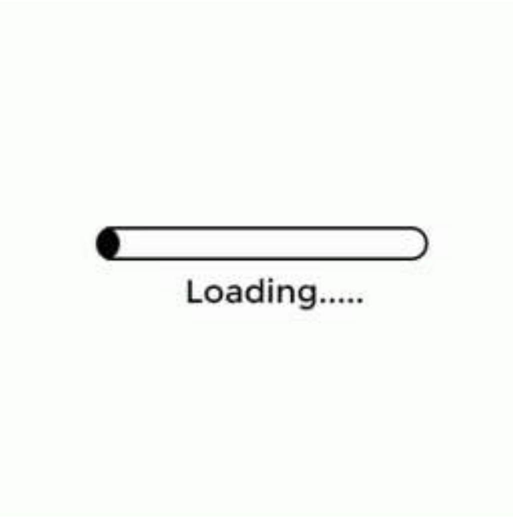
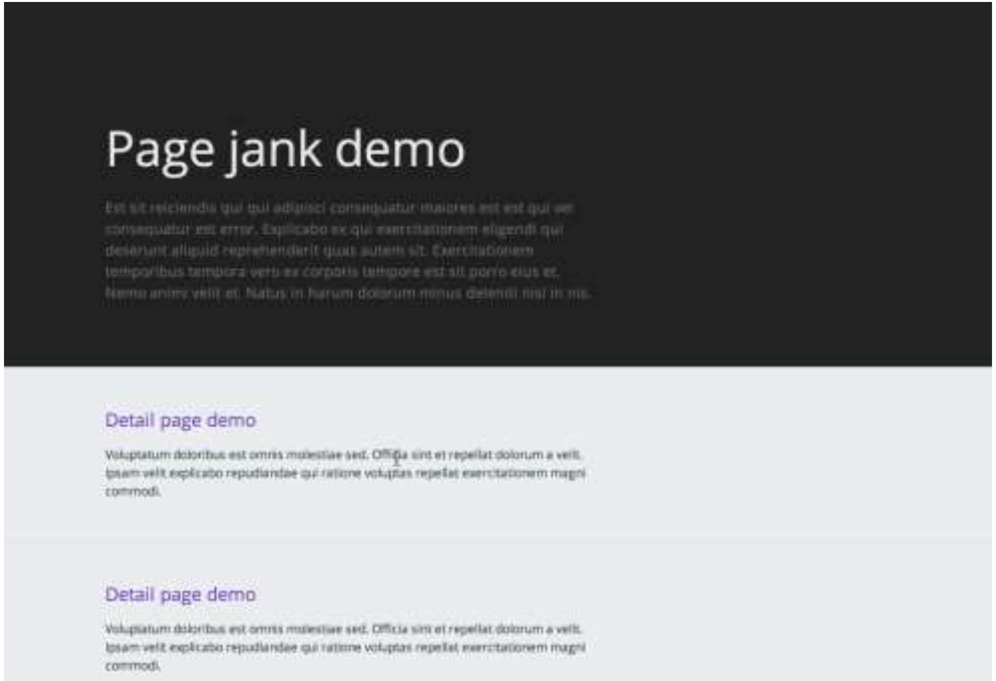


Image taken from: https://feld.com/archives/2014/05/stop-slow-lane.html

Which leads to

Image taken from: https://beinspiredchannel.com/frustrated-frustration/

And then they work with the app



Page jank demo

Est sit reiciendis qui qui adipisci consequatur maiores est est qui vel consequatur est error. Explicabo ex qui exercitationem eligendi qui deserunt aliquid reprehenderit quas autem sit. Exercitationem temporibus tempora vero ex corporis tempore est sit porro eius et. Nemo animi velit et. Natus in harum dolorum minus deleniti nisi in nis.

Detail page demo

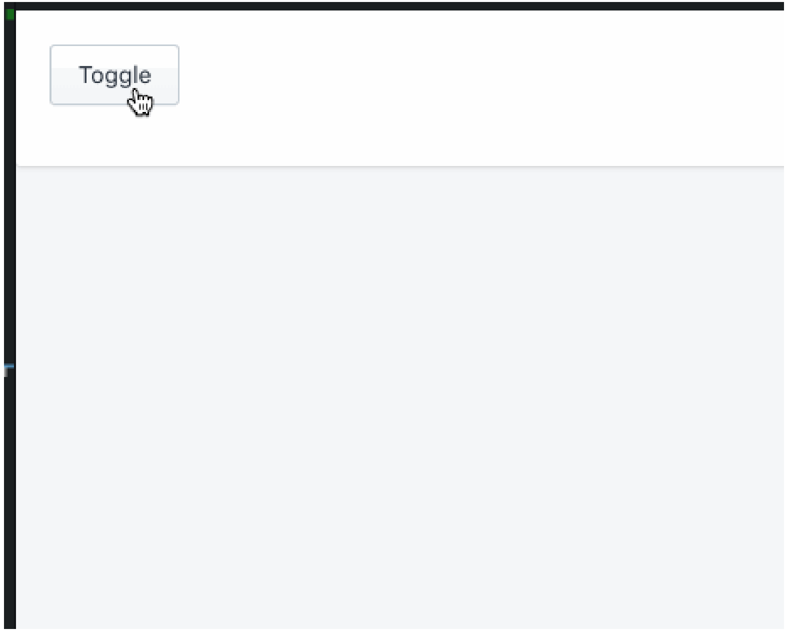
Voluptatum doloribus est omnis molestiae sed. Officia sint et repellat dolorum a velit. Ipsam velit explicabo repudiandae qui ratione voluptas repellat exercitationem magni commodi.

Detail page demo

Voluptatum doloribus est omnis molestiae sed. Officia sint et repellat dolorum a velit. Ipsam velit explicabo repudiandae qui ratione voluptas repellat exercitationem magni commodi.

Image taken from: https://css-tricks.com/wp-content/uploads/2014/08/js-ecc9543f.gif

Image taken from: https://medium.com/myheritage-engineering/how-to-greatly-improve-your-react-app-performance-e70f7cbbb5f6


Toggle

ReactionGIFS.me

Best App Ever!

About Me

- sparXys CEO and senior consultant
- Google Web Technologies GDE & Microsoft MVP
- Pro Single Page Application Development (Apress) co-author

Agenda

- How browsers render pages?
- Profiling JavaScript apps
- Finding JavaScript memory leaks

Understanding Browsers Rendering

Refresh Rates

- Devices refresh their screens 60 times a second = 60fps
- That means that each frame should take 16ms
  - 1 second / 60 = 16.66ms
- In reality a frame takes ~10ms to produce
  - The browsers have management overhead

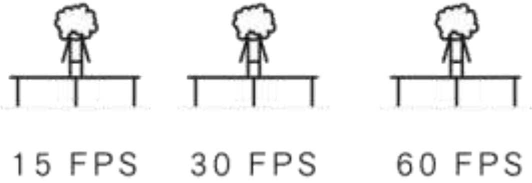
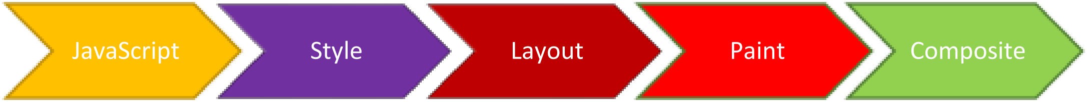


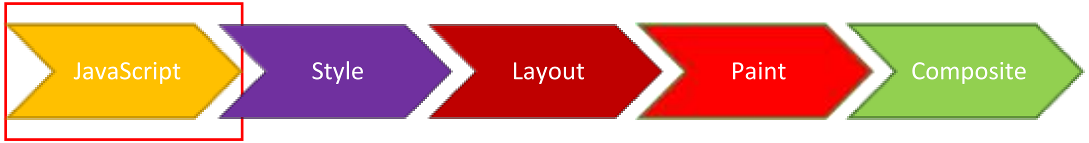
Image taken from:
https://www.reddit.com/r/pcmasterrace/comments/3v26qw/153060_fps_comparison_in_a_very_eli5_way/

Shipping a frame to screen

□ The pixel pipeline:

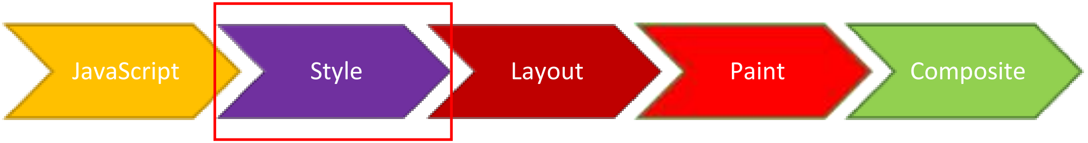


Shipping a frame to screen



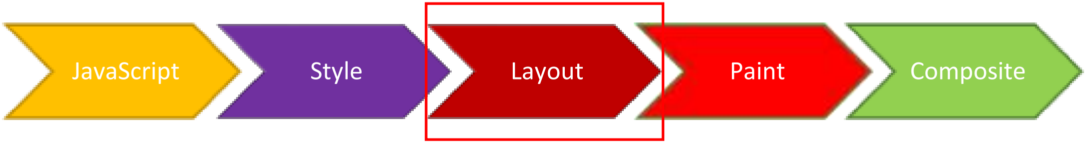
- JavaScript used to handle work that will result in visual changes
- CSS Animations, Transitions, and the Web Animations API are also calculated here

Shipping a frame to screen



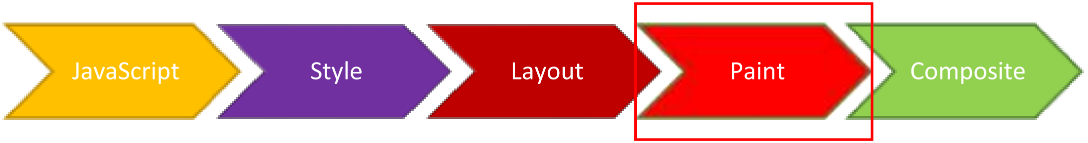
- Browser figures out which CSS rules should be applied to elements based on CSS selectors
- The style is then calculated to each element

Shipping a frame to screen



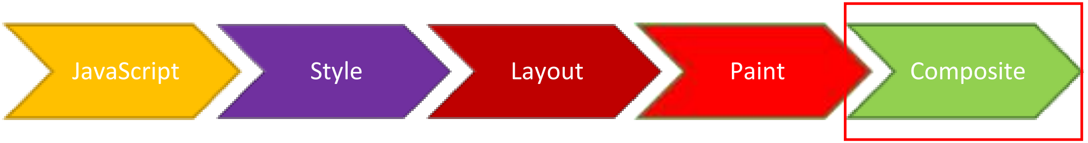
- The browser calculates how much space each element takes up and where it is on screen
- Each element affects other elements in the layout
  - Web layout model

Shipping a frame to screen



- The browser paints all the pixels on screen
- It draws every visual part of an element (text, color, images, and etc.)

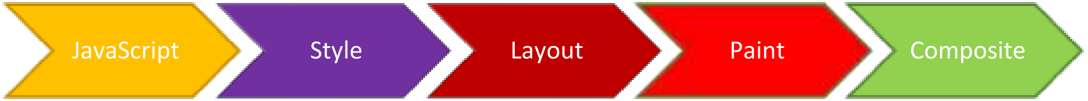
Shipping a frame to screen



- The browser draws the elements according to their layer
  - If elements overlap each other
- Happens mostly on the machine GPU
  - Therefore this step is fast

Reflows

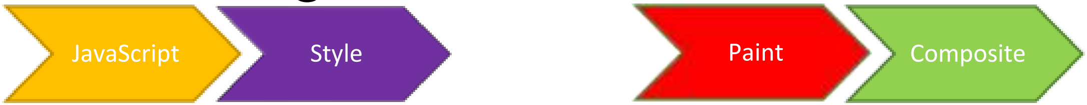
- Reflow might occur whenever a visual change requires a change in the layout of the page
  - Examples: browser resize, DOM manipulation and etc.



- All the flow of the pixel pipeline will run again

Repaints

- Repaint occurs when a visual change doesn’t require recalculation of the whole layout
  - Examples: element visibility change, changes in text color or background colors and etc.



JavaScript

Style

Paint

Composite

- All the flow of the pixel pipeline except layout will run again

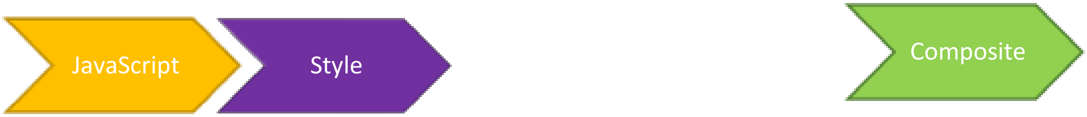
Reflows/Repaints

Try to minimize them as much as possible

Changes without Reflow/Repaint

- JavaScript or CSS changes that don’t affect neither layout or paint

- The flow of the pixel pipeline will run again without layout and paint:



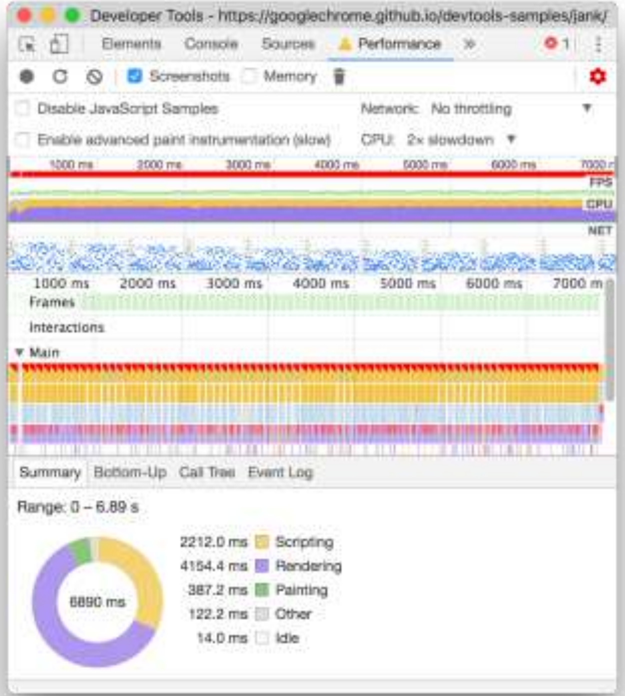
RAIL Model

Image taken from: https://developers.google.com/web/fundamentals/performance/rail

- User-centric performance model
  - Response: process events in under 50ms
  - Animation: produce a frame in 10ms
  - Idle: maximize idle time
  - Load: deliver content and become interactive in under 5 seconds

We are ready to profile

DevTools Performance Tab



Developer Tools - https://googlechrome.github.io/devtools-samples/jank/

☑ Screenshots □ Memory

□ Disable JavaScript Samples — Network: No throttling

□ Enable advanced paint instrumentation (slow) — CPU: 2x slowdown

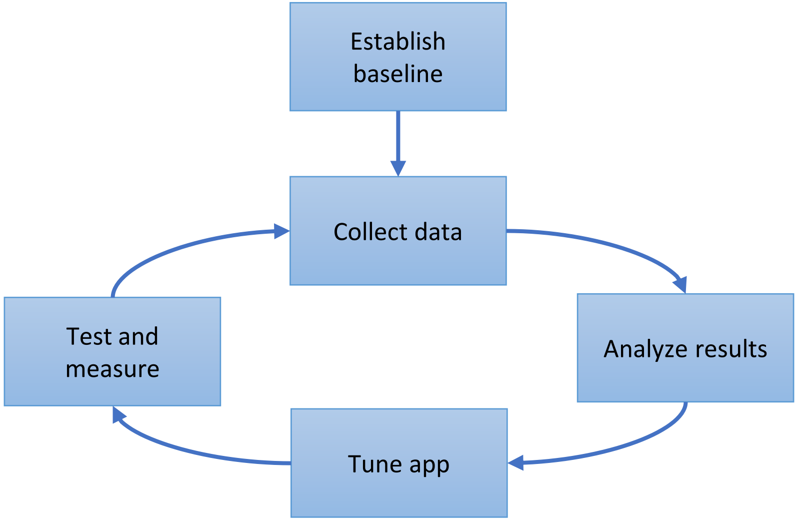
Range: 0 – 6.89 s

2212.0 ms	🟨	Scripting
4154.4 ms	🟪	Rendering
387.2 ms	🟩	Painting
122.2 ms	⬜	Other
14.0 ms	□	Idle

Demo

Chrome DevTools Performance Tab

Profiling Process



Demo

Profiling JavaScript using Chrome DevTools

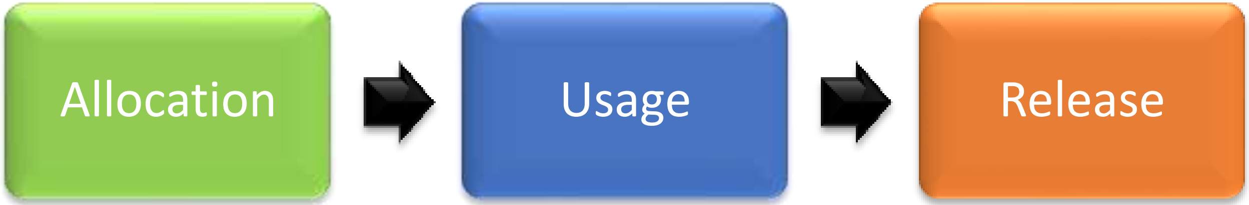
TM & © 2005 The Jim Henson Company

Story Time

A-HA!

“Hey! Your application has a memory leak.”

Memory Lifetime



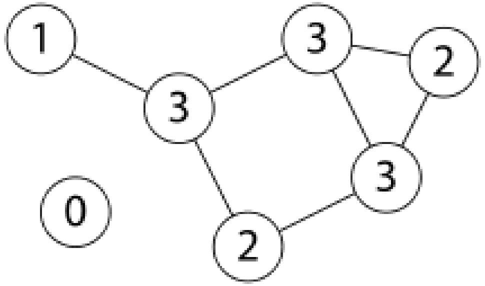
Allocation

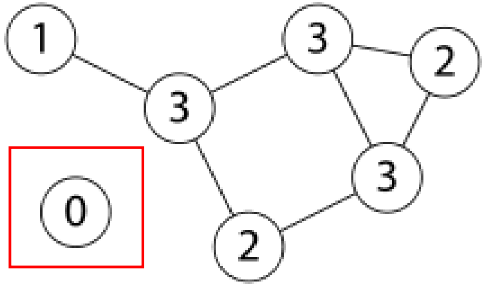
Usage

Release

Memory

- Can be represented as a connected graph
- The graph starts with a root
    - Node number 1 in the diagram





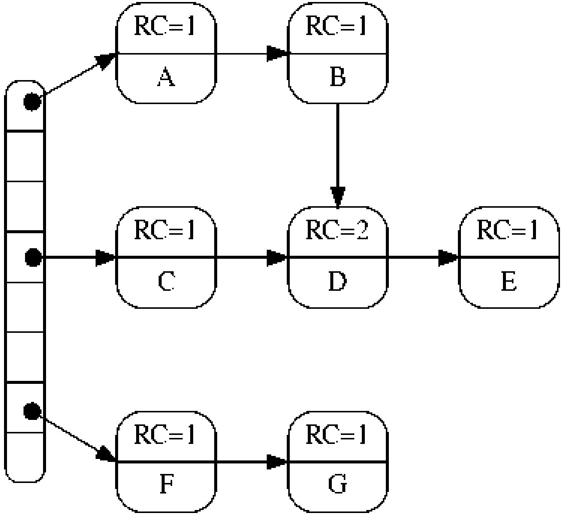
What about unreferenced memory?

Garbage Collector

- Looks out for unreachable objects, which are removed from the memory
- Known algorithms:
  - Reference-counting garbage collection
  - Mark-and-sweep

Reference-counting Garbage Collection

- An object is said to be "garbage", or collectible if there are zero references pointing to it



Mark-and-sweep Algorithm

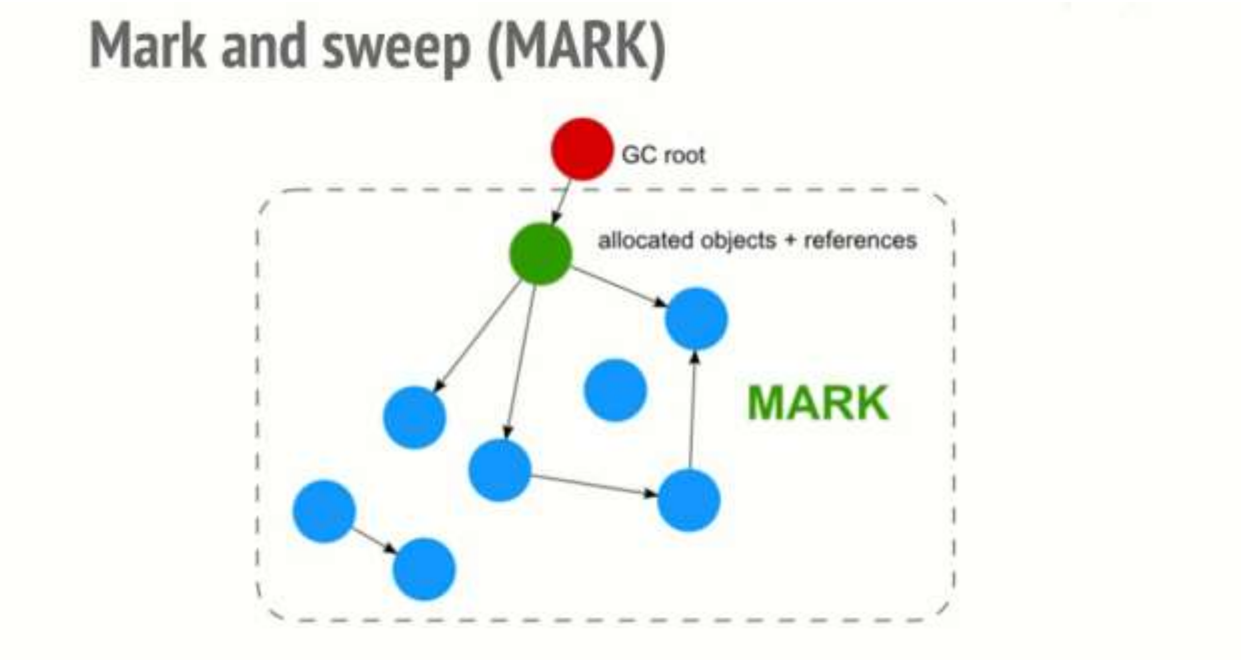
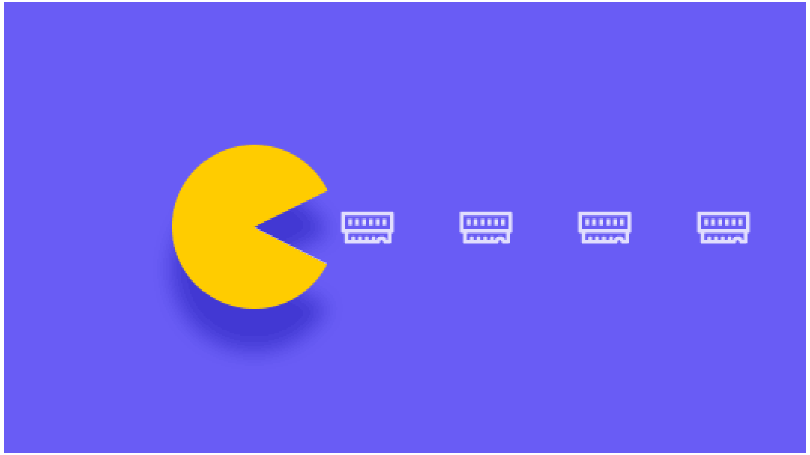


Image taken from: https://blog.sessionstack.com/how-javascript-works-memory-management-how-to-handle-4-common-memory-leaks-3f28b94cfbec

Memory Leaks

- Memory that isn't required by an app, but isn't returned to the pool of free memory



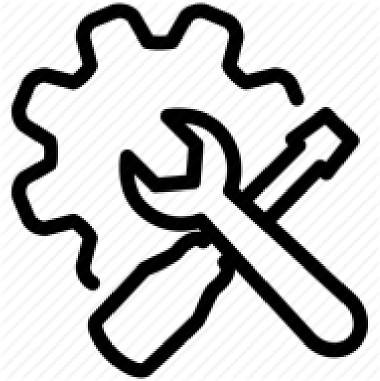
Memory Leaks in JavaScript?

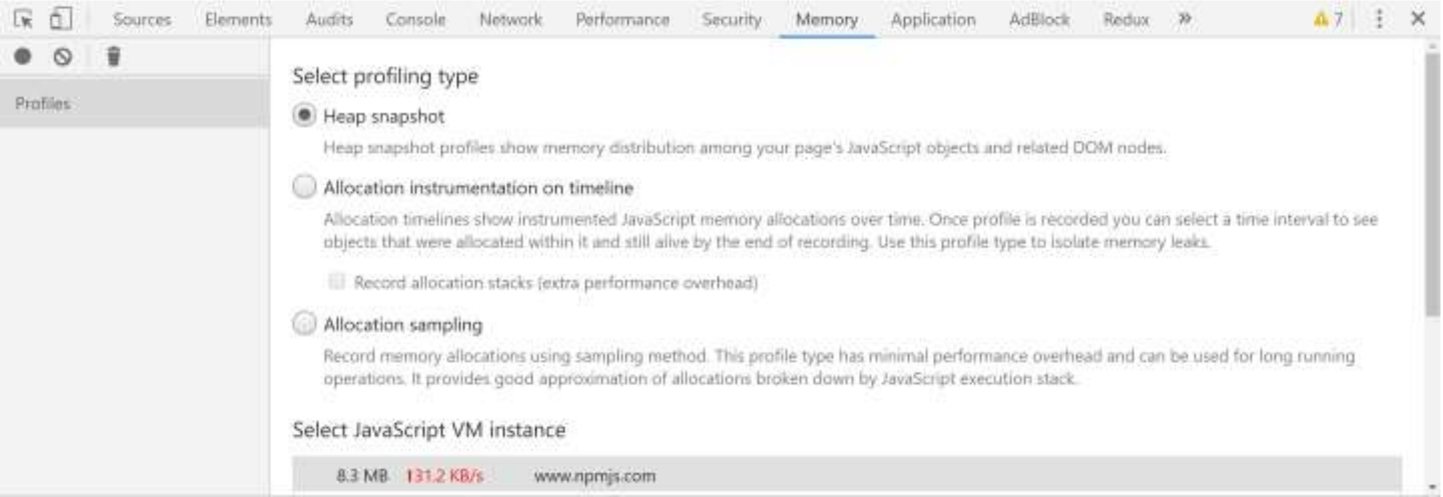
Common JS Memory Leak Pitfalls

- Accidental global variables
- Forgotten timers or callbacks
- Closures
- Out of DOM references

Detect Memory Leaks in the Browser

- Using Browser DevTools
- Using window.performance object





Select profiling type

● Heap snapshot

Heap snapshot profiles show memory distribution among your page's JavaScript objects and related DOM nodes.

○ Allocation instrumentation on timeline

Allocation timelines show instrumented JavaScript memory allocations over time. Once profile is recorded you can select a time interval to see objects that were allocated within it and still alive by the end of recording. Use this profile type to isolate memory leaks.

□ Record allocation stacks (extra performance overhead)

○ Allocation sampling

Record memory allocations using sampling method. This profile type has minimal performance overhead and can be used for long running operations. It provides good approximation of allocations broken down by JavaScript execution stack.

Select JavaScript VM instance

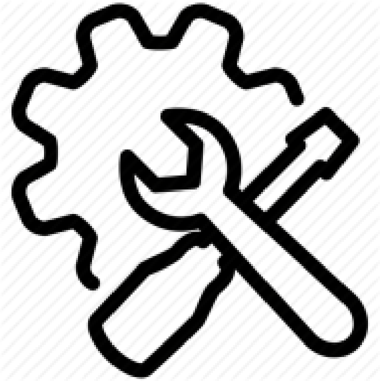
8.3 MB ↑31.2 KB/s www.npmjs.com

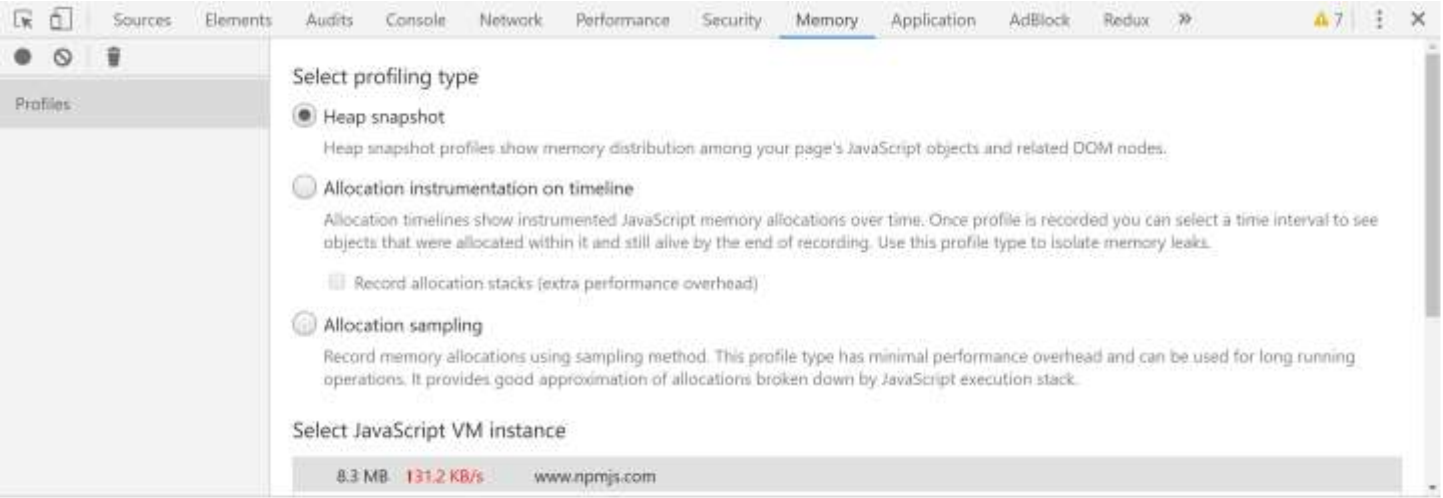
Demo

Detecting a Memory Leak in the Front-end

Detect Memory Leaks in Node.js

- Using Chrome DevTools
  - Run node in –inspect mode
  - Use Memory tab
- Using node-memwatch
- Using Heapdump





Select profiling type

● Heap snapshot

Heap snapshot profiles show memory distribution among your page's JavaScript objects and related DOM nodes.

○ Allocation instrumentation on timeline

Allocation timelines show instrumented JavaScript memory allocations over time. Once profile is recorded you can select a time interval to see objects that were allocated within it and still alive by the end of recording. Use this profile type to isolate memory leaks.

□ Record allocation stacks (extra performance overhead)

○ Allocation sampling

Record memory allocations using sampling method. This profile type has minimal performance overhead and can be used for long running operations. It provides good approximation of allocations broken down by JavaScript execution stack.

Select JavaScript VM instance

8.3 MB ↑31.2 KB/s www.npmjs.com

Demo

Detecting a Memory Leak – in the Backend

Performance problems will happen
Don’t wait that your users will complain!

Monitor Your Production

Thank you

Follow me on Twitter: @gilfink