



Google

Developers



A Trip Down Memory Lane with Gmail and Chrome DevTools

Effective Memory Management

Loreena Lee & John McCutchan



Performance vs. Memory

My Gmail tab is using a gig of RAM.

So what? You've got 24GB on your machine!

Yeah, but my grandma's Chromebook only has 4GB.

When it comes down to the age-old *performance vs. memory* tradeoff, developers usually opt for **performance**.



Gmail, we have a problem...



He's Dead, Jim!

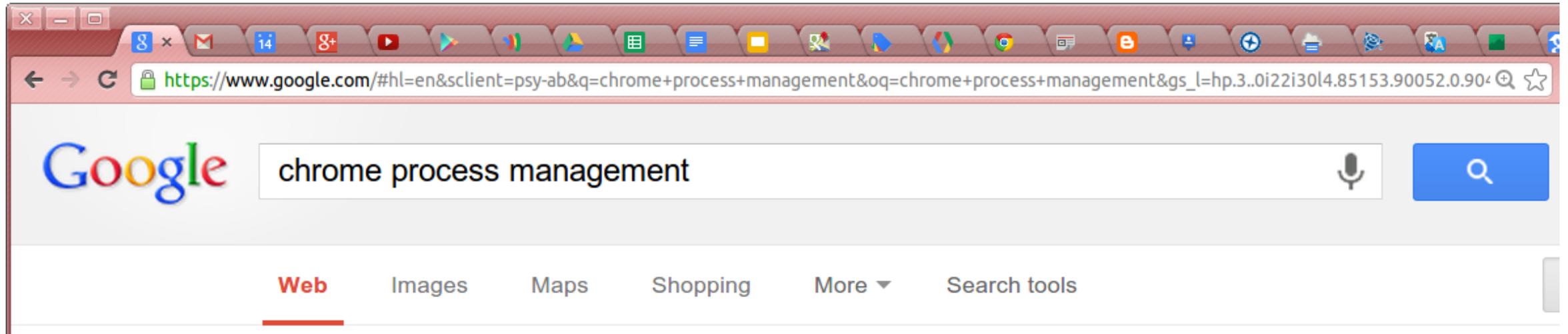
Something caused this webpage to be killed, either because the operating system ran out of memory, or for some other reason. To continue, press Reload or go to another page.

[Learn more](#)

- Lack of actual memory data
- Anecdotes of huge memory footprints
- Uncontrolled memory growth for common actions
- Analysis tools that didn't scale up to Gmail



Not just Gmail's problem



Where do we start?

Before we can tackle this problem...

... we need to go ***back to basics.***





Memory Management Basics

Core Concepts

- What types of values are there?
- How are values organized in memory?
- What is garbage?
- What is a leak?

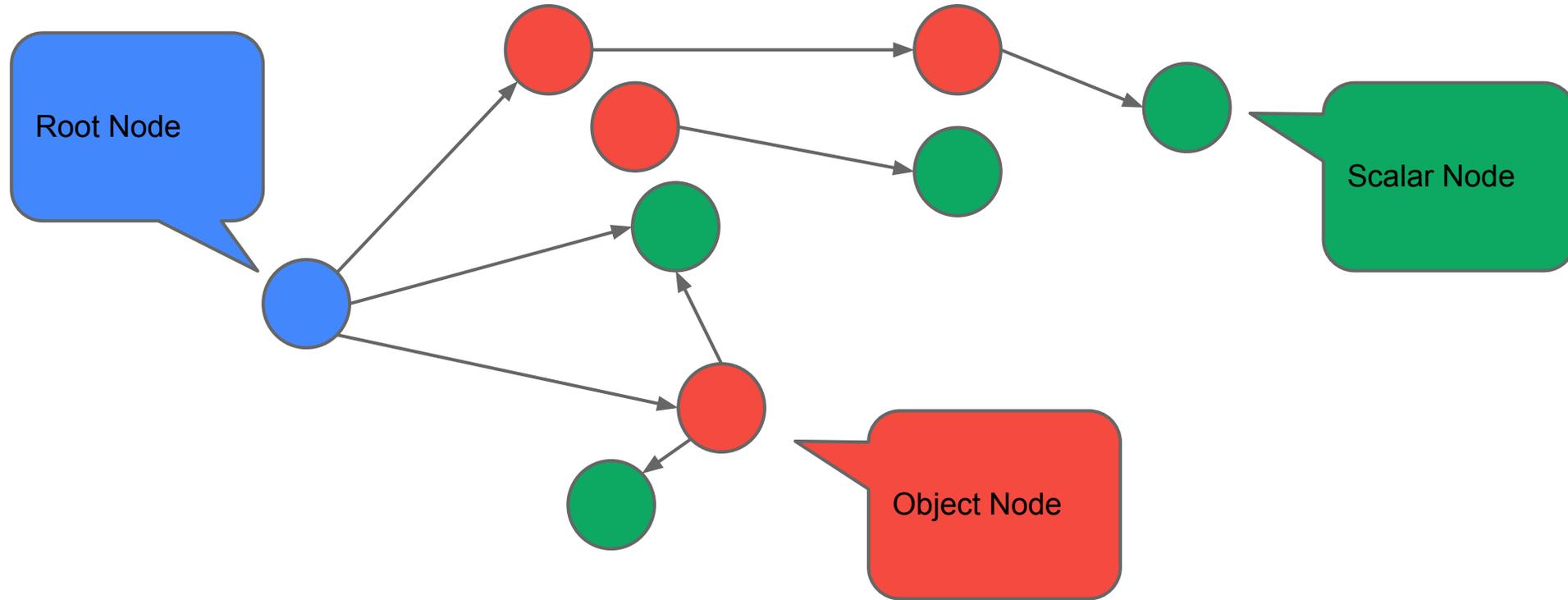


JavaScript value **types**

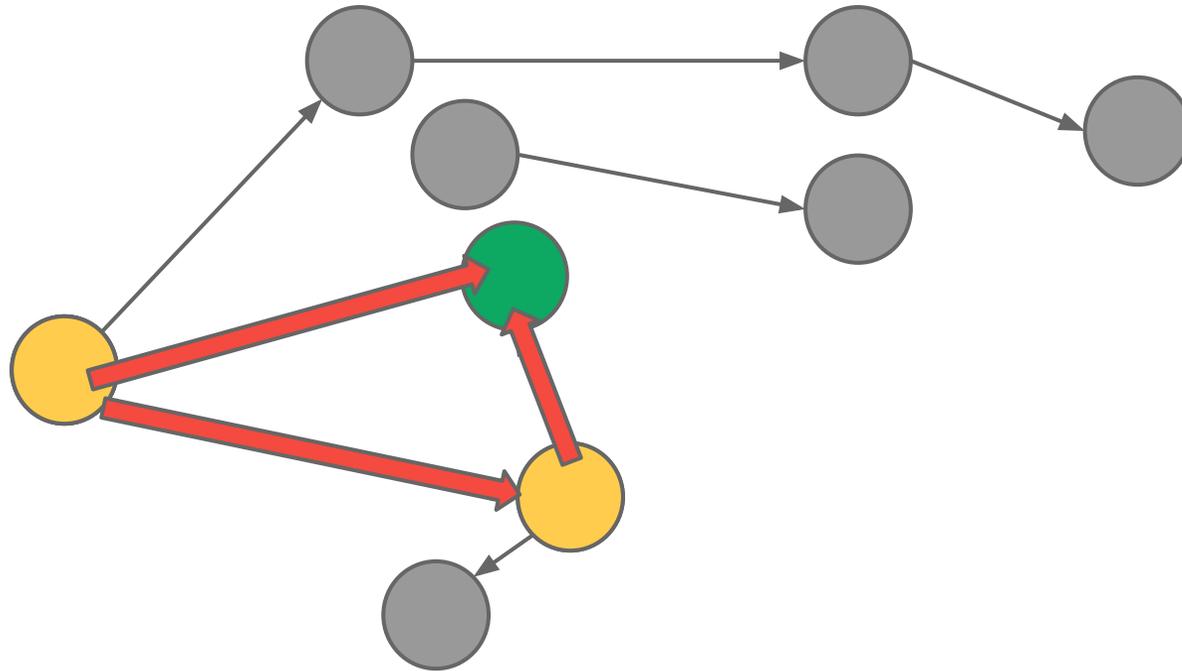
- boolean
 - true or false
- number
 - double precision IEEE 754 number
- string
 - UTF-16 string
- objects
 - associative array
- external objects
 - DOM nodes, image data, ...



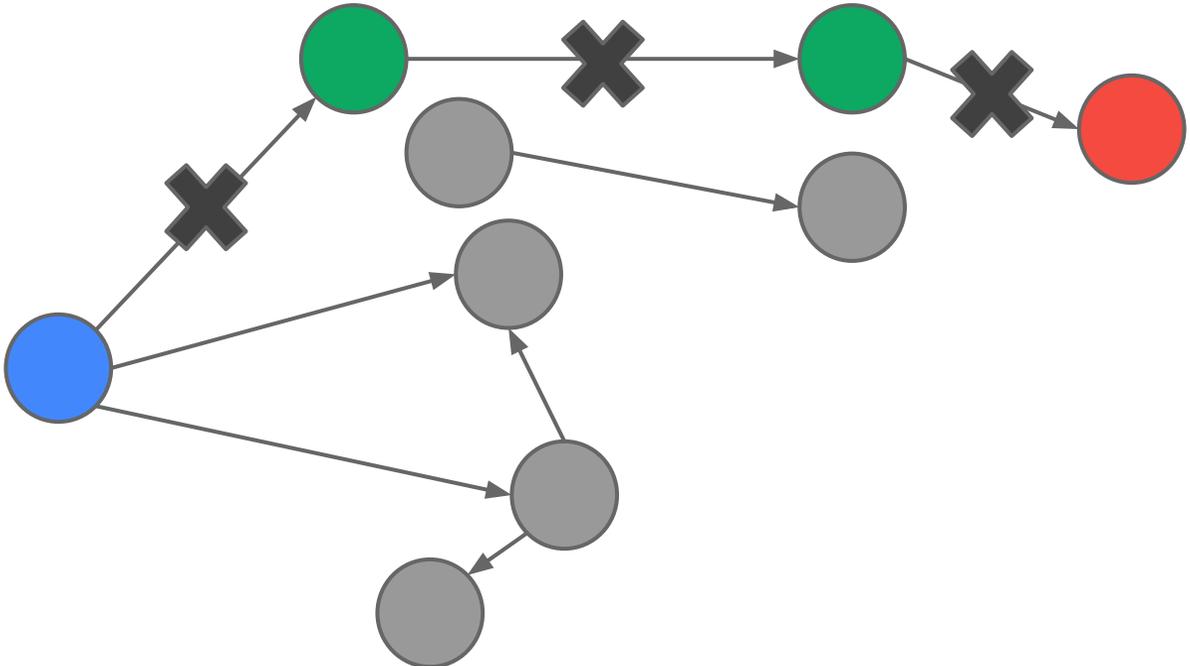
The value graph



A value's retaining path(s)

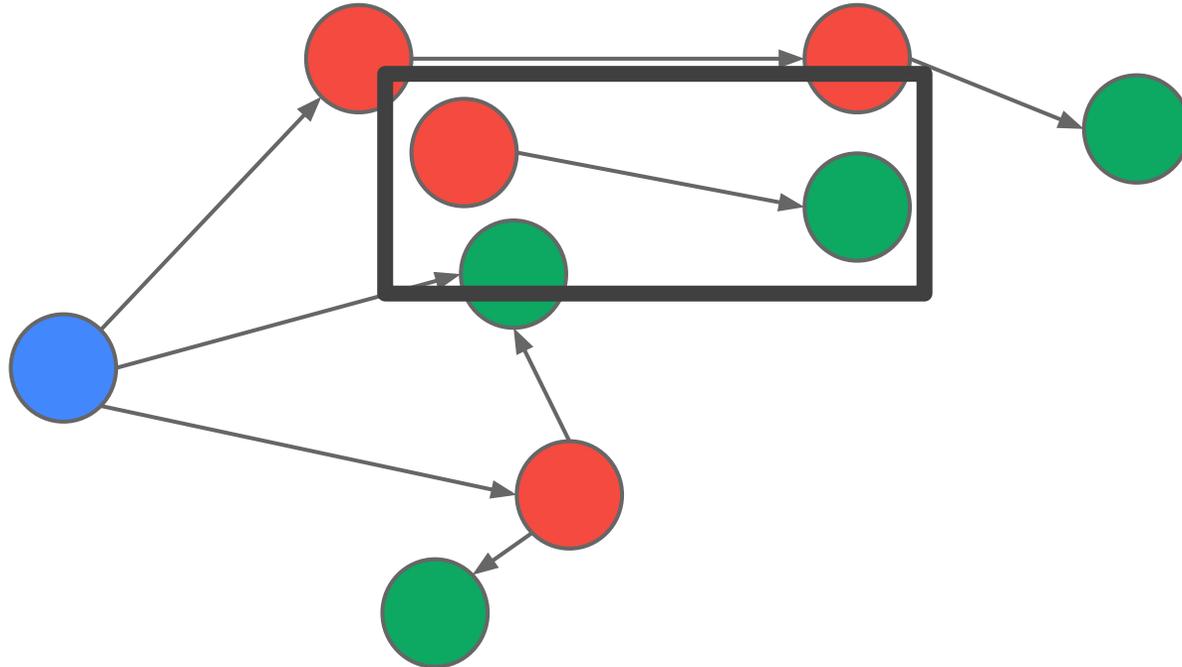


Removing a value from the graph

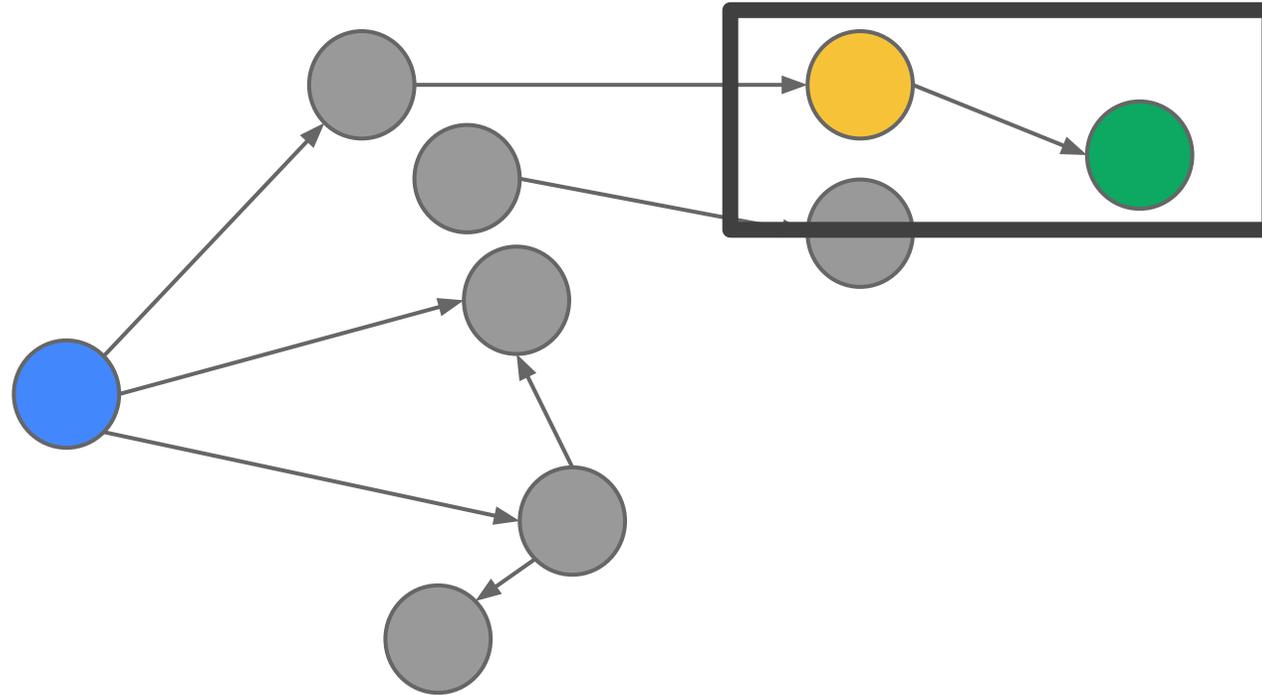


What is **garbage**?

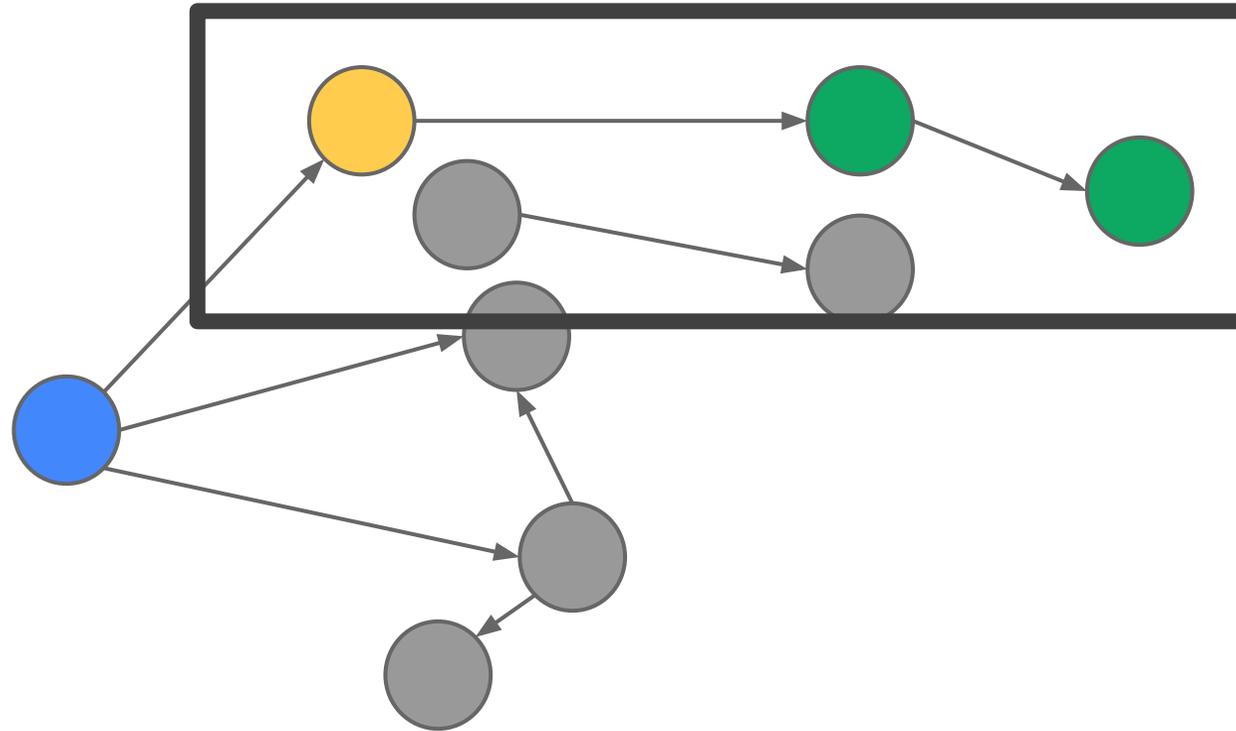
- Garbage: All values which cannot be reached from the root node.



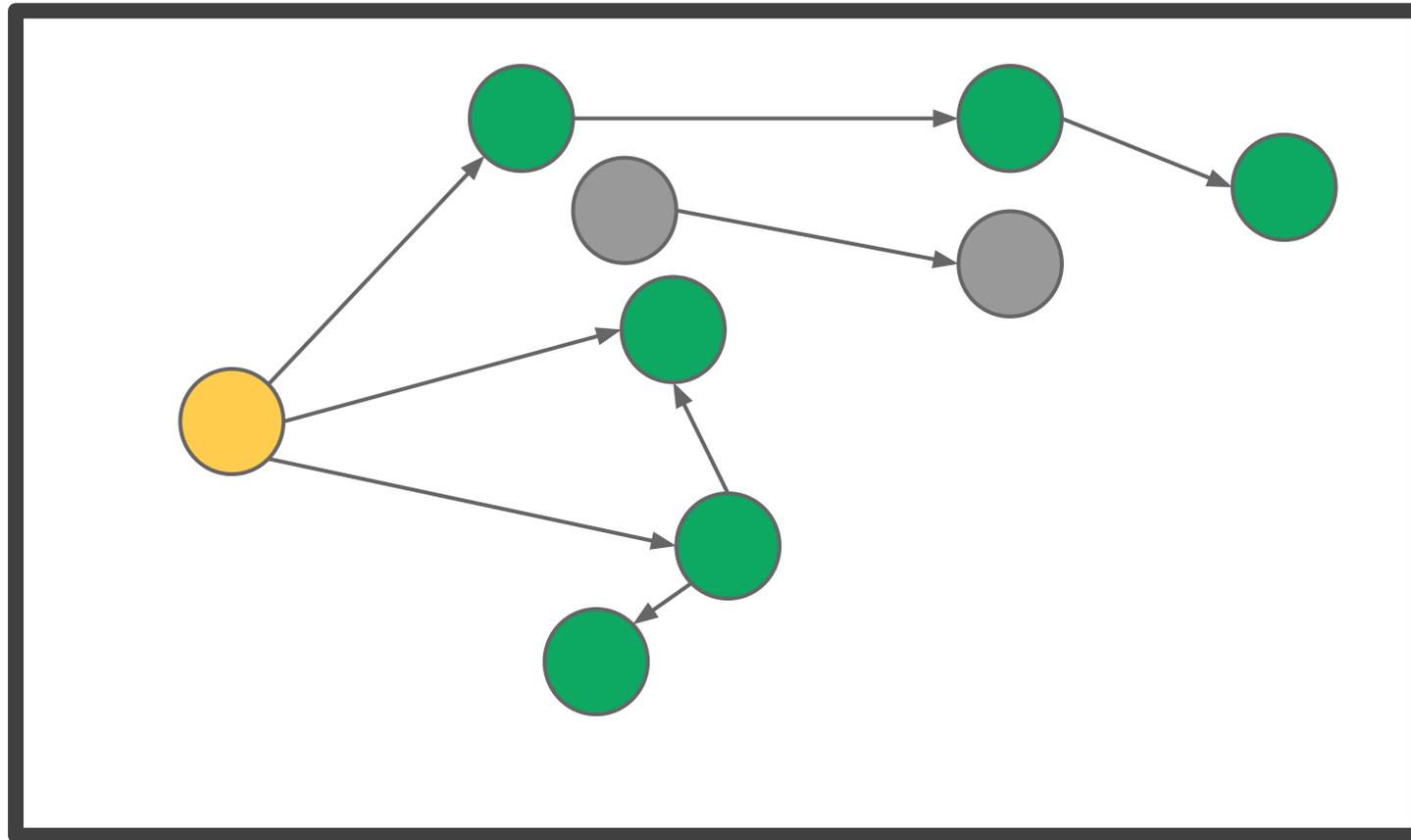
A value's retained size



A value's retained size



A value's retained size



Leaks in JavaScript

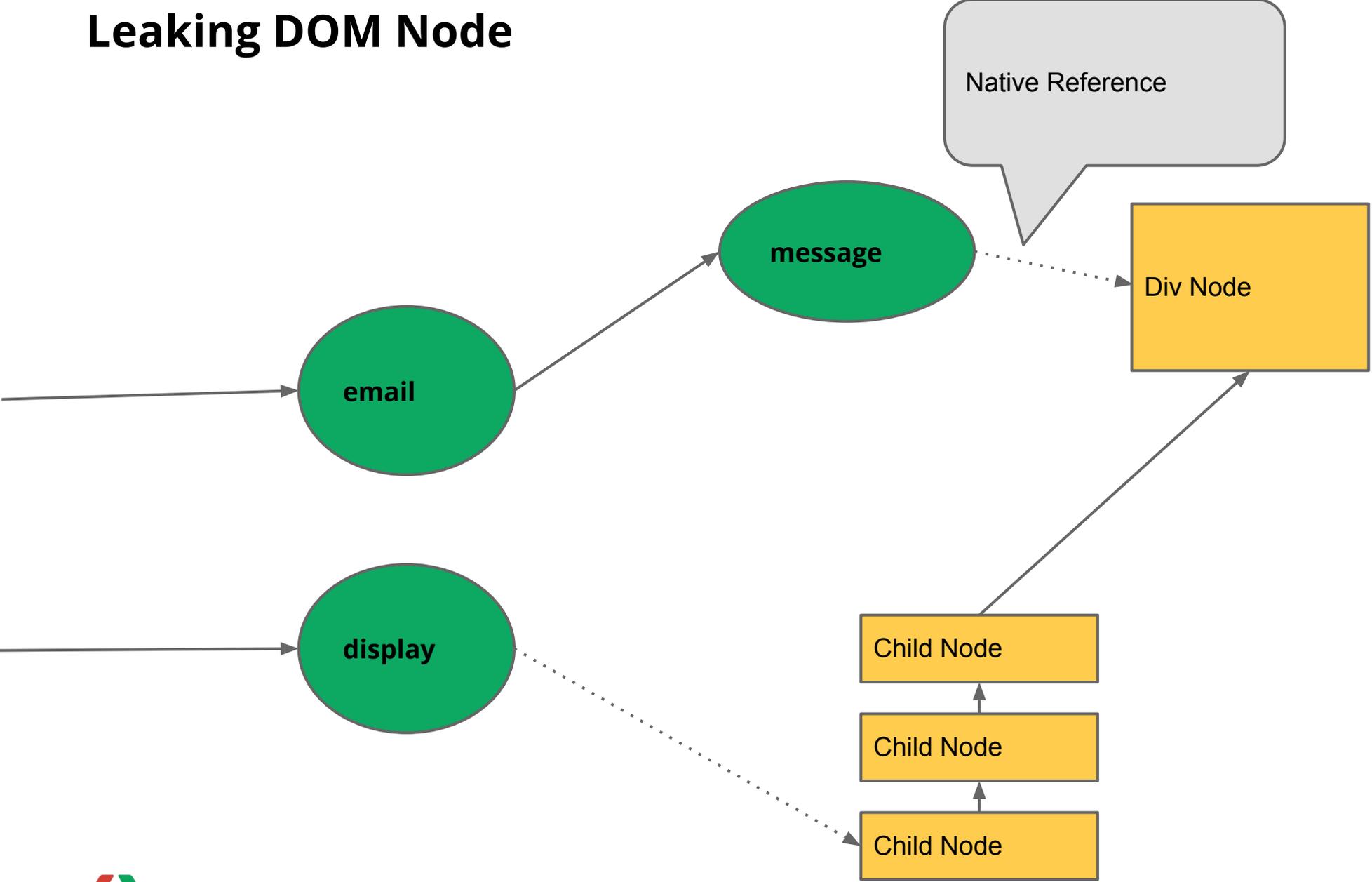
- A **value** that erroneously still has a retaining path
 - Programmer error

JavaScript

```
email.message = document.createElement("div");  
  
display.appendChild(email.message);
```



Leaking DOM Node



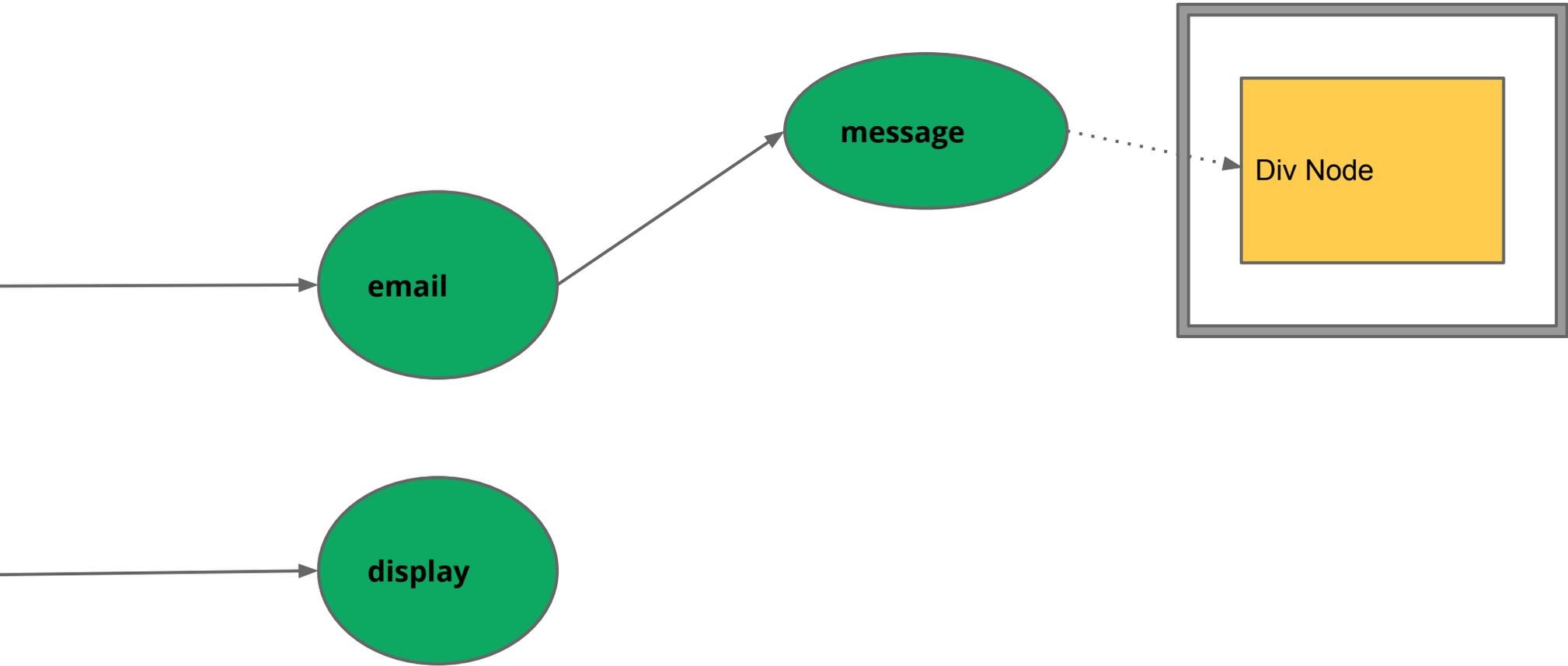
Leaks in JavaScript

JavaScript

```
// ...  
display.removeAllChildren();
```



Leaking DOM Node



Memory Management Basics

- Values are organized in a graph
- Values have retaining path(s)
- Values have retained size(s)





V8 Memory Management

A GC Pause Walkthrough

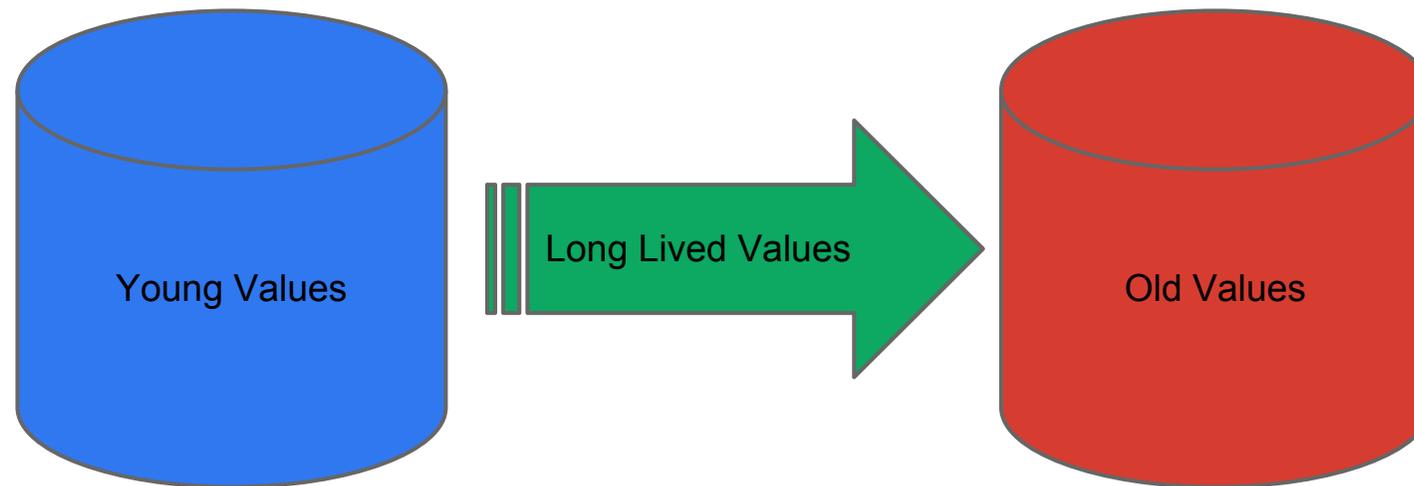
Where is the **cost** in allocating memory?

- Every call to **new** or implicit memory allocation
 - Reserves memory for object
 - Cheap until...
- Memory pool **exhausted**
 - Runtime forced to perform a **garbage collection**
 - Can take milliseconds (!)
- Applications must be careful with object allocation patterns
 - Every allocation brings you closer to a **GC pause**



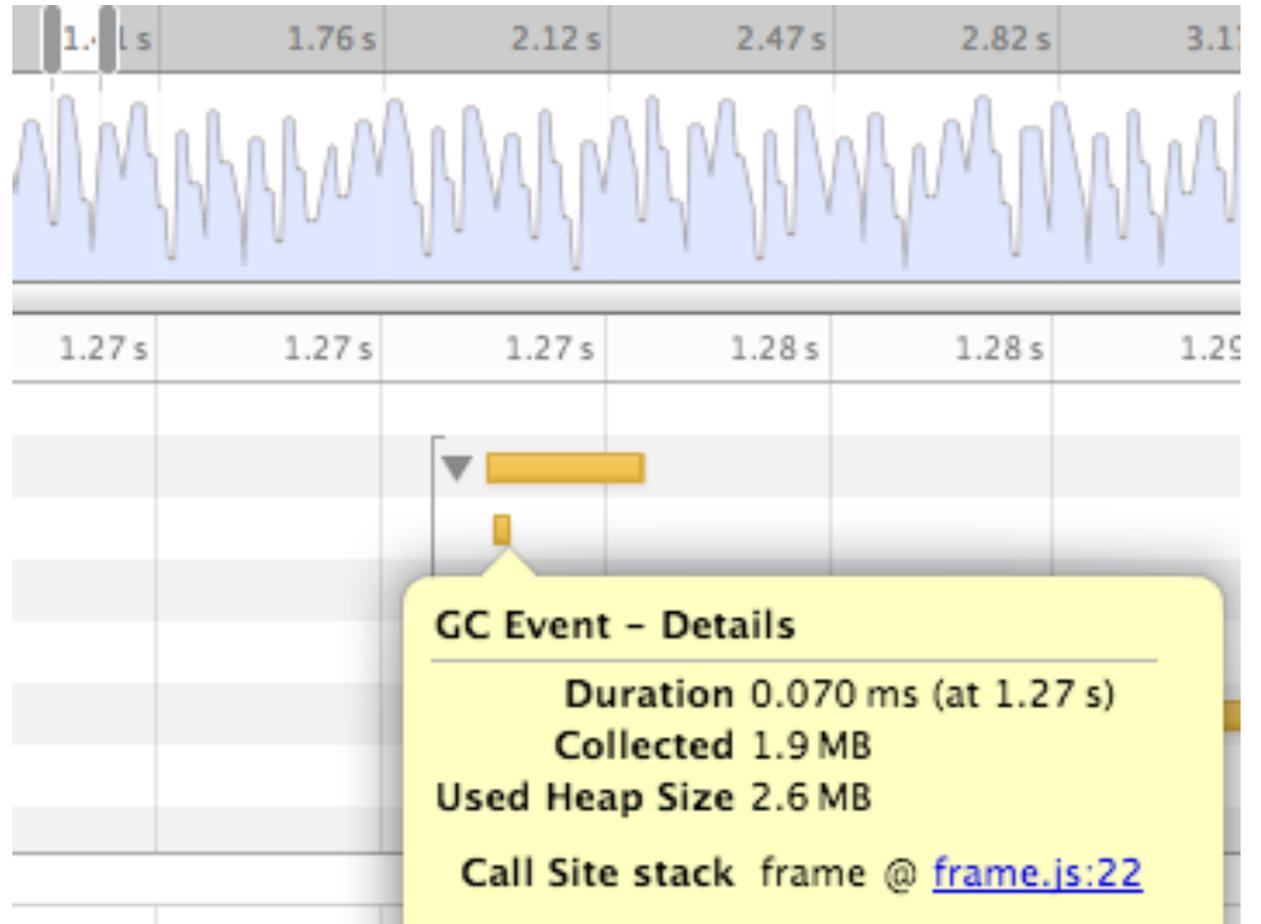
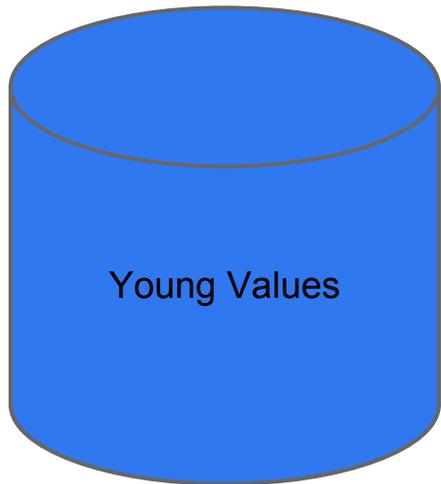
How does V8 manage memory?

- Generational
 - Split values between **young** and **old**
 - Overtime **young** values promoted to **old**



How does V8 manage memory?

- Young Generation
 - Fast allocation
 - Fast collection
 - Frequent collection



How does V8 manage memory?

- Old Generation
 - Fast allocation
 - Slower collection
 - **Infrequently** collected

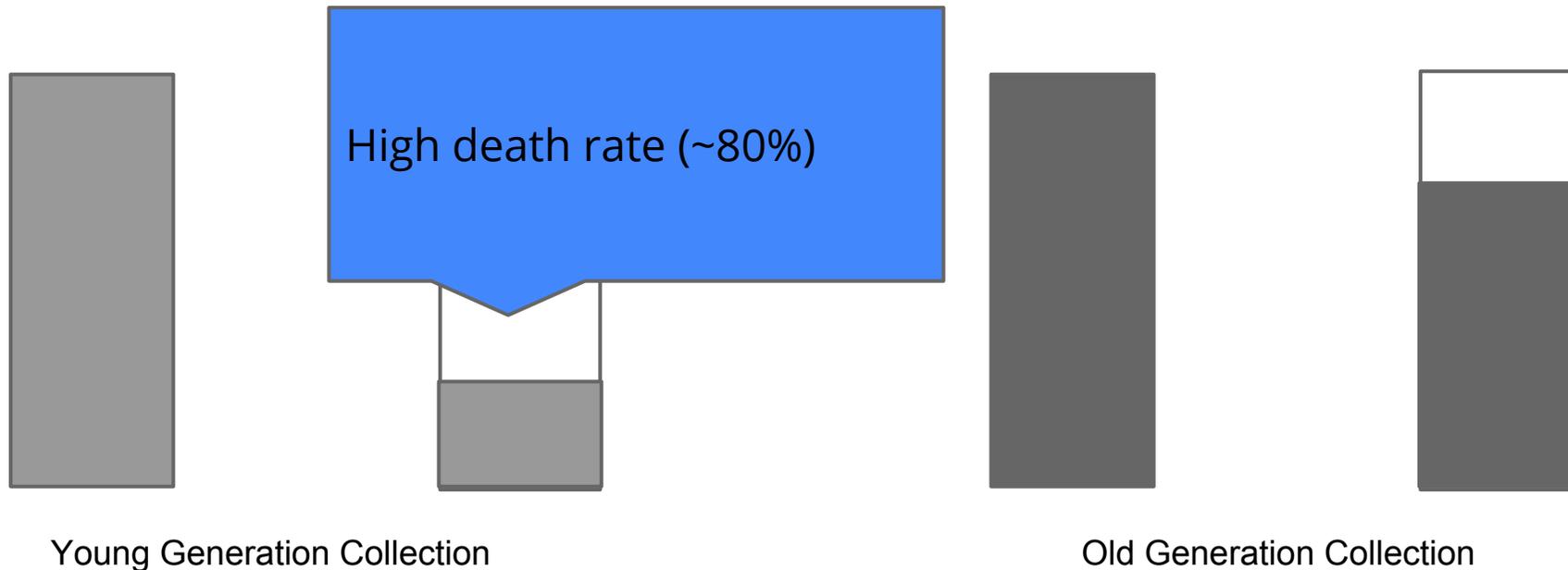


- Parts of collection run concurrently with mutator
 - Incremental Marking
- Mark-sweep
 - Return memory to system
- Mark-compact
 - Move values



How does V8 manage memory?

- Why is collecting the young generation faster
 - Cost of GC is proportional to the number of live objects



Young Generation In Action



Young Generation In Action

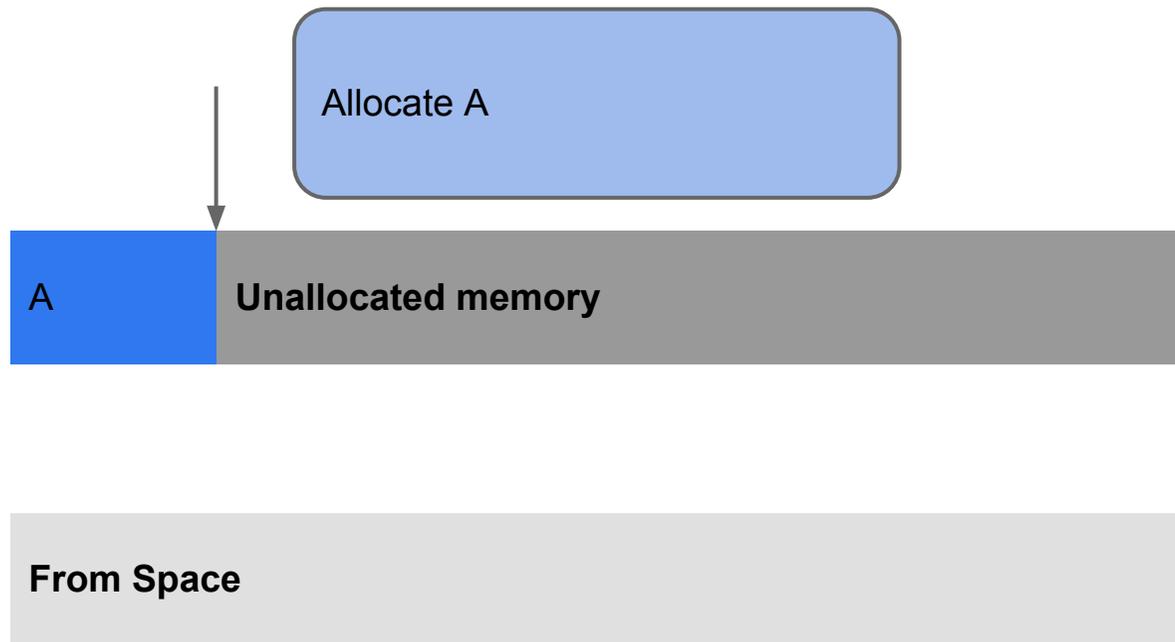


Unallocated memory

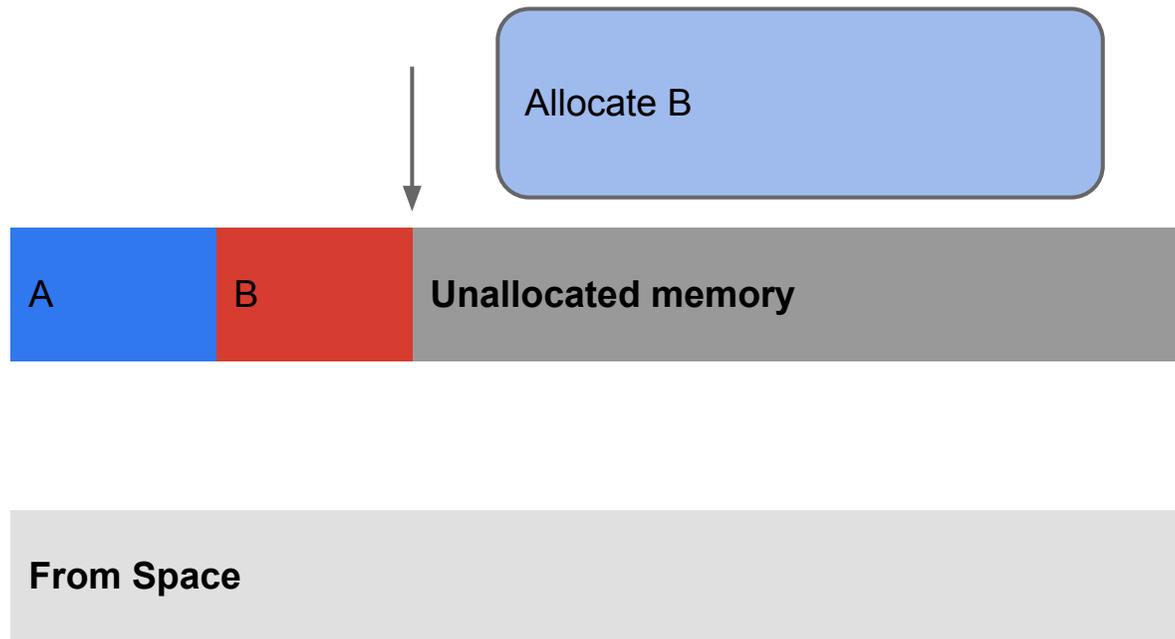
From Space



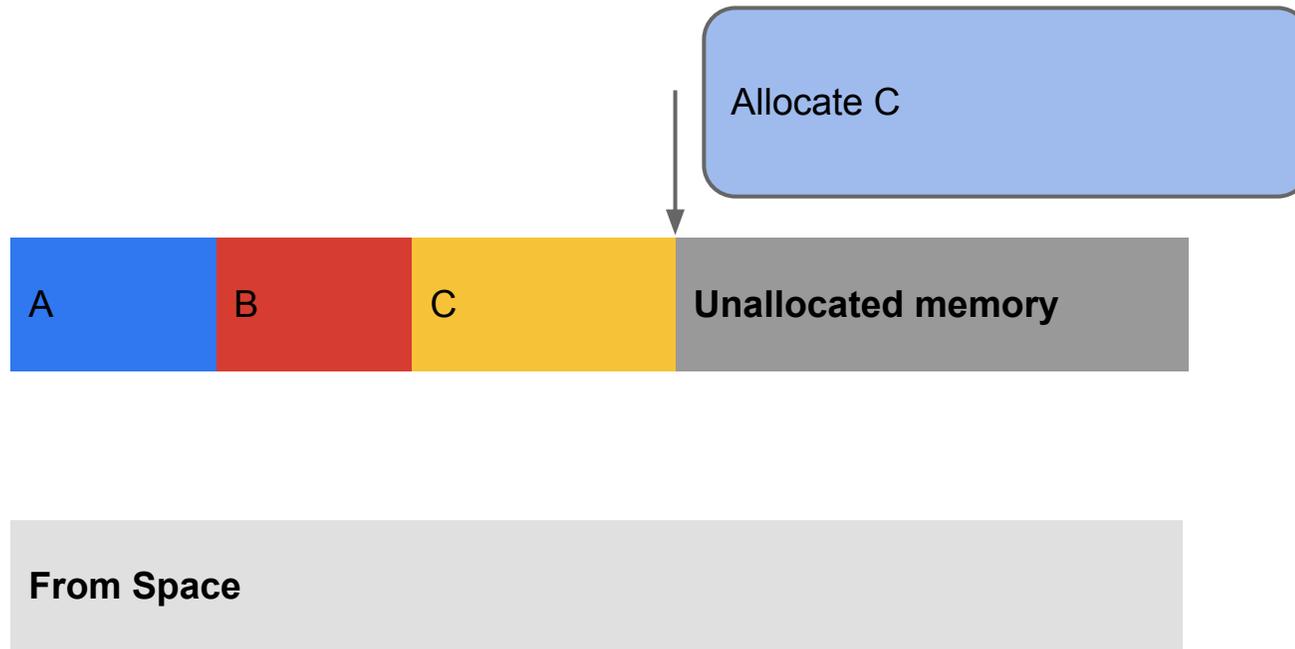
Young Generation In Action



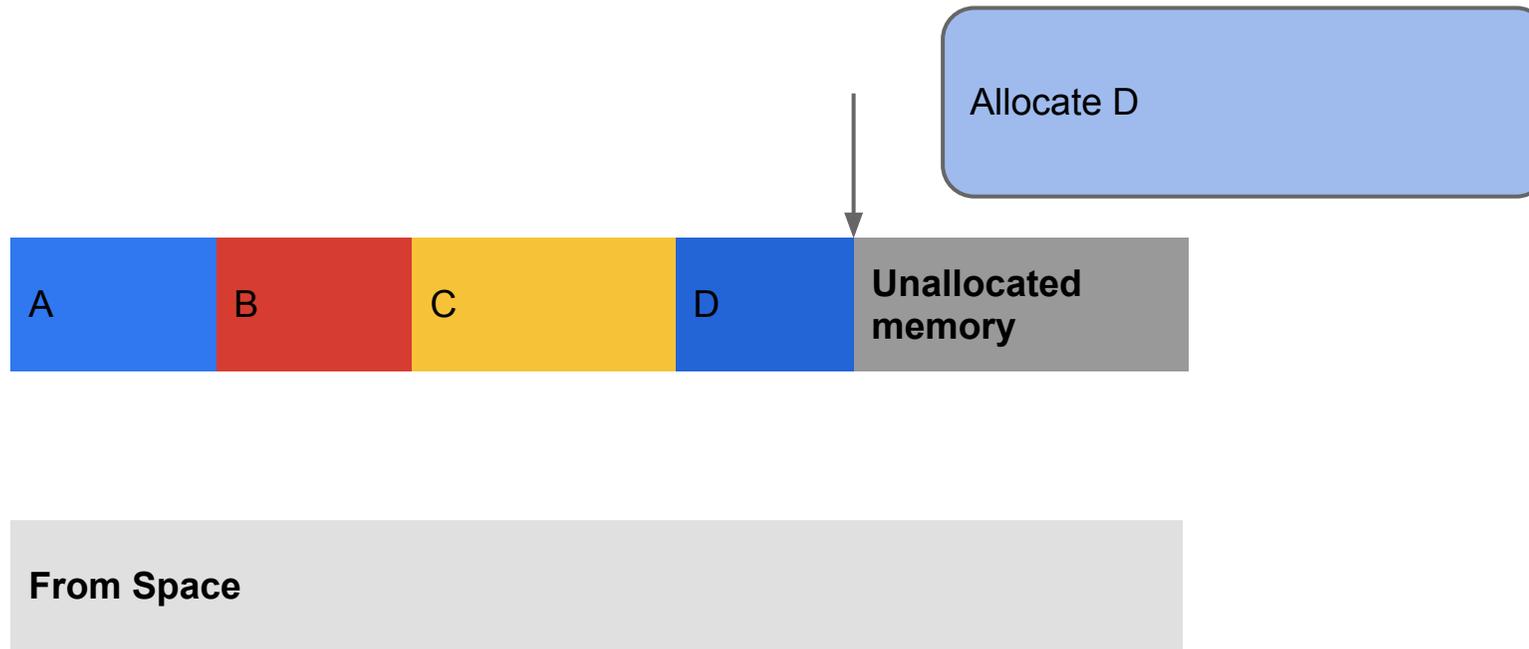
Young Generation In Action



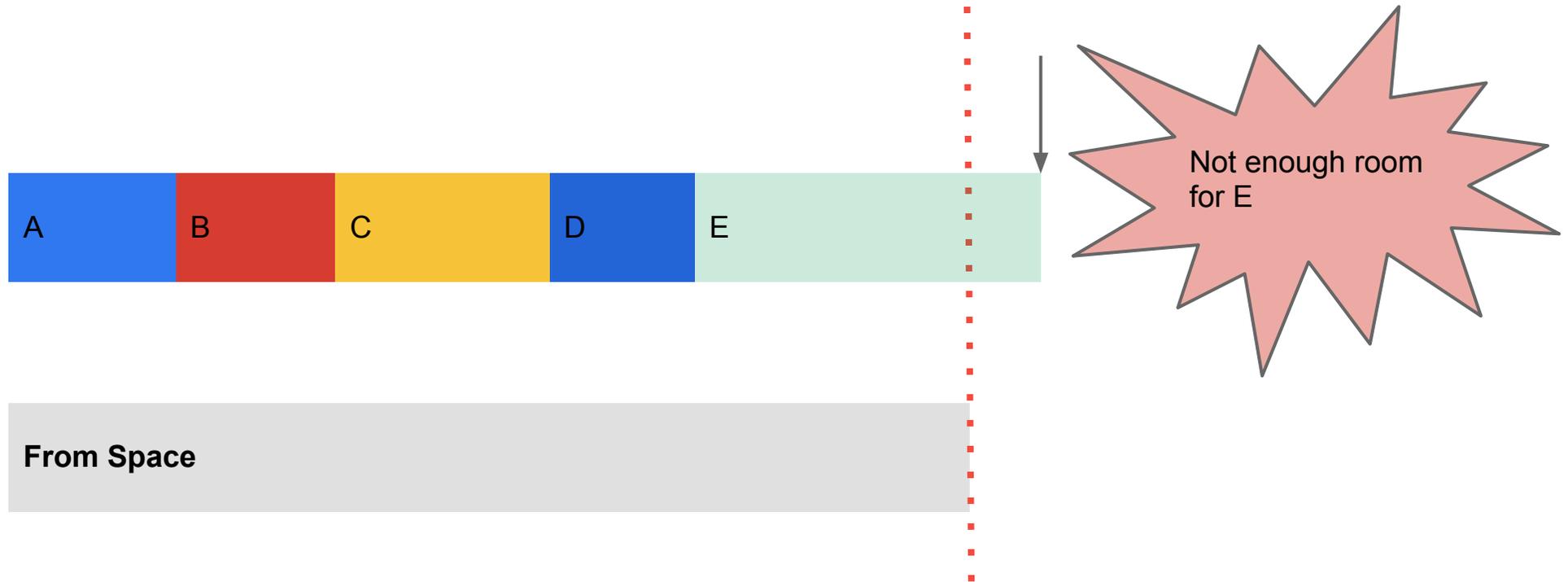
Young Generation In Action



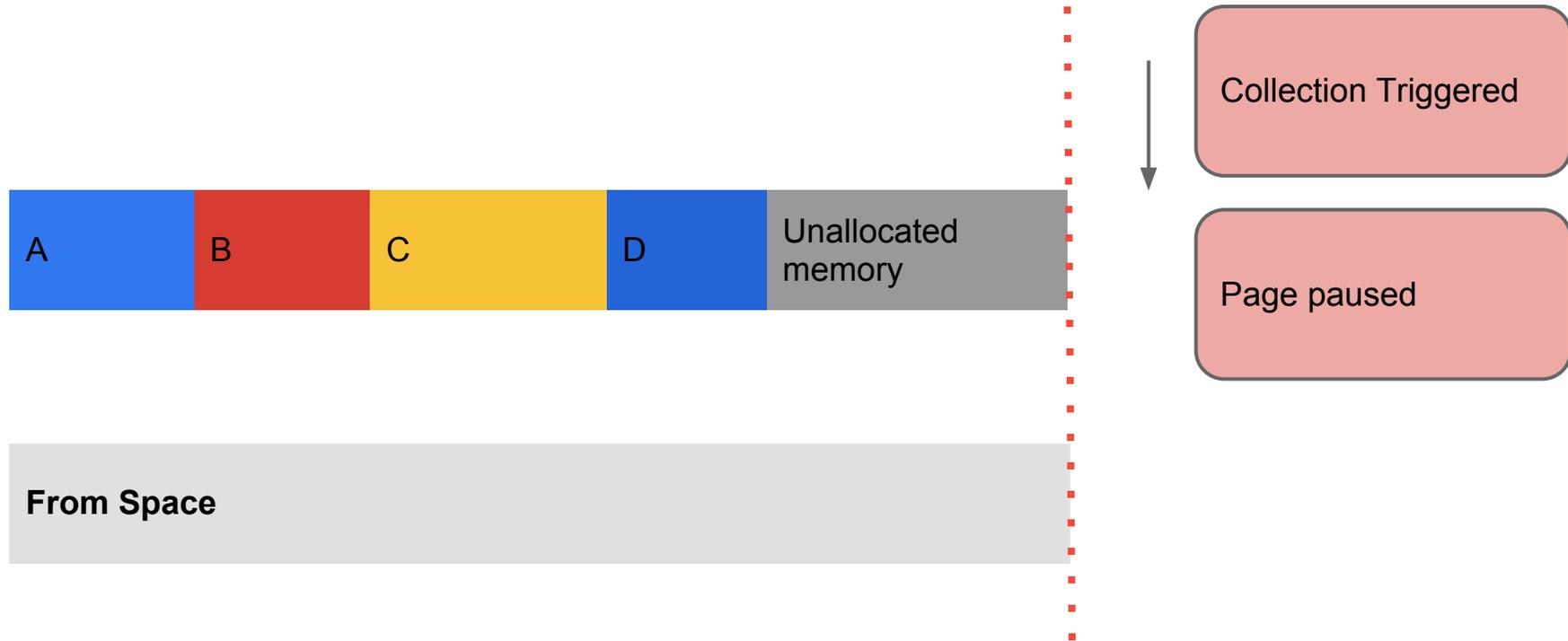
Young Generation In Action



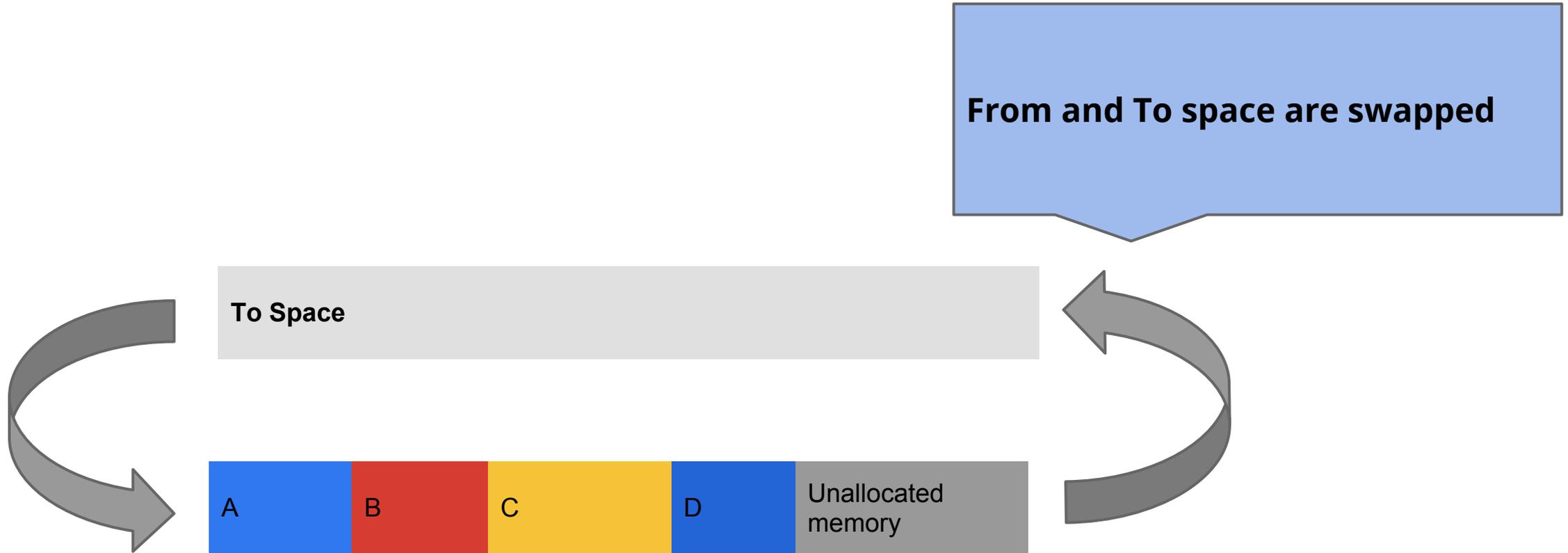
Young Generation In Action



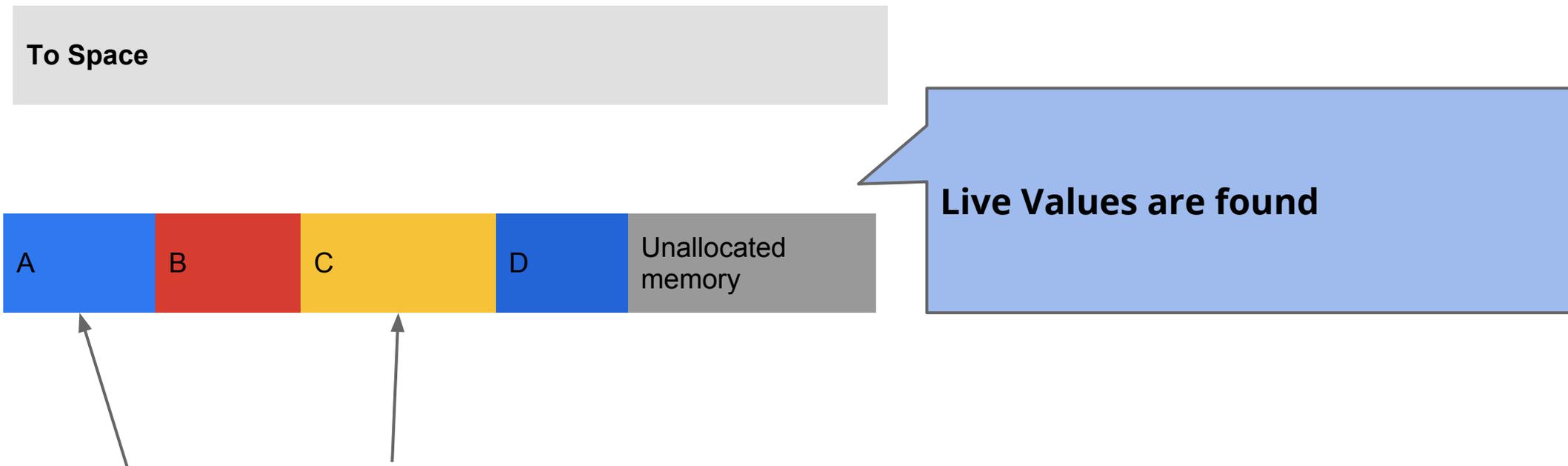
Young Generation In Action



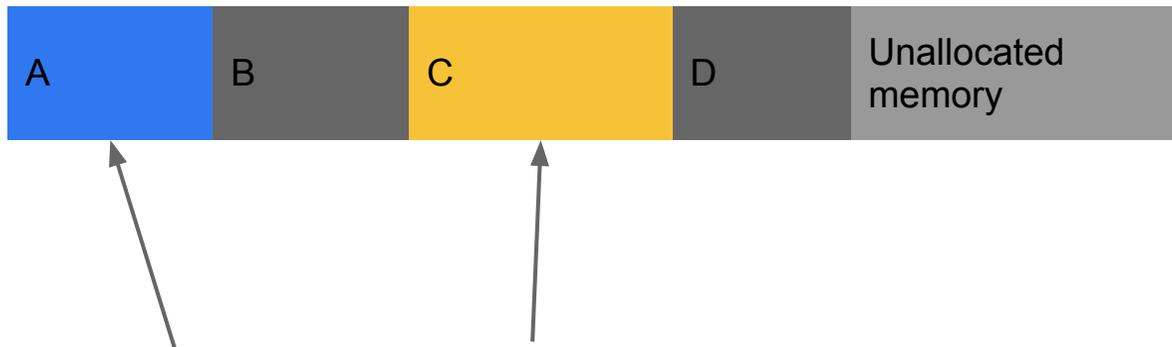
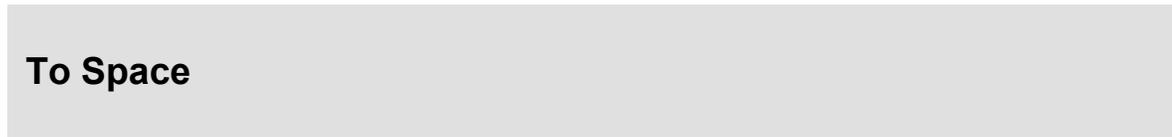
Young Generation In Action



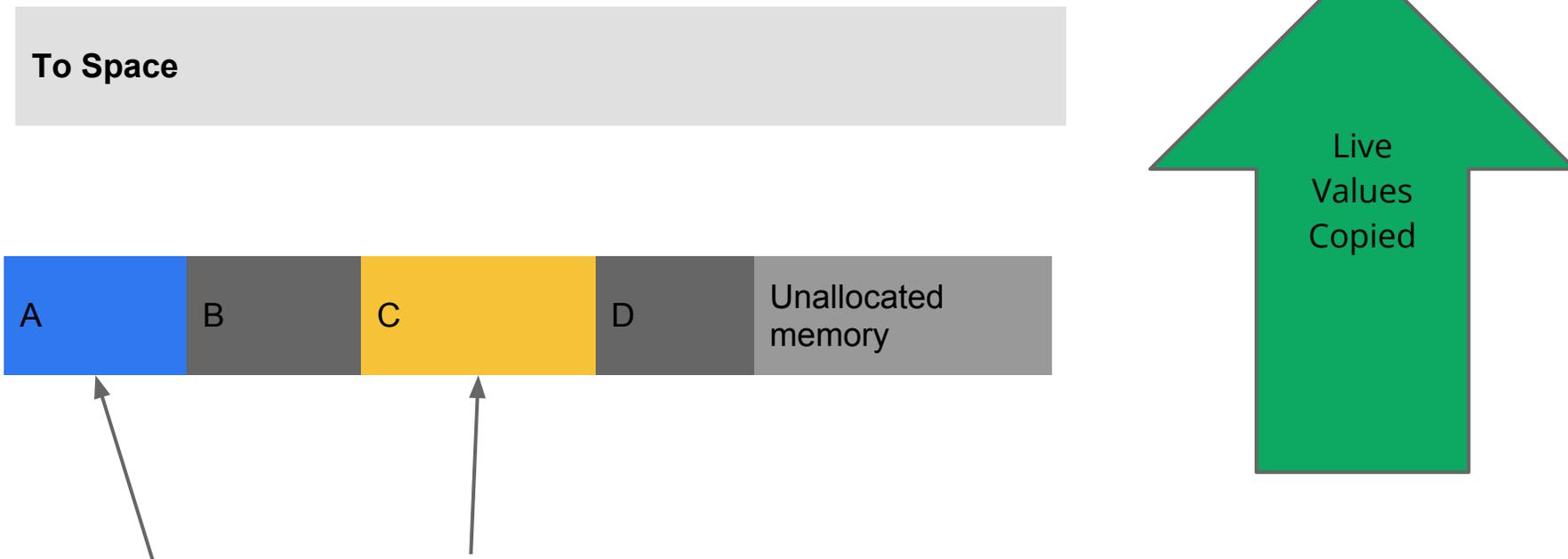
Young Generation In Action



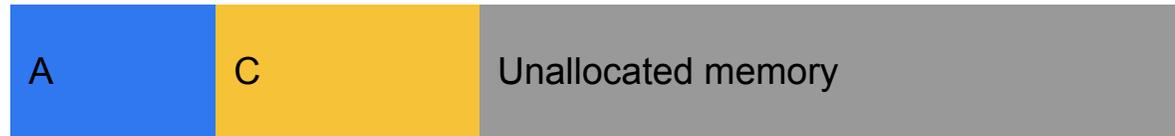
Young Generation In Action



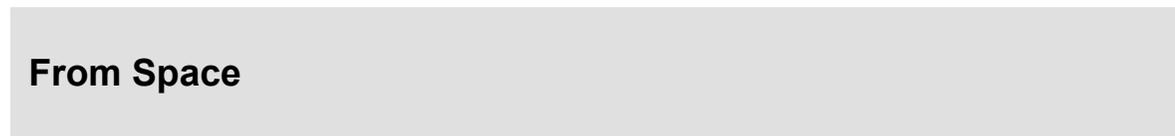
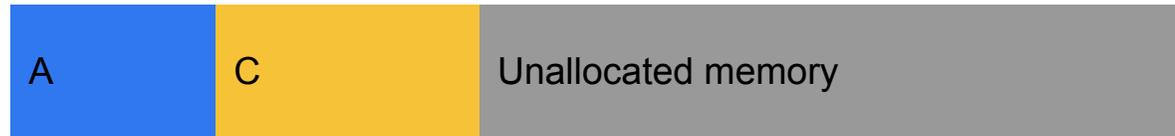
Young Generation In Action



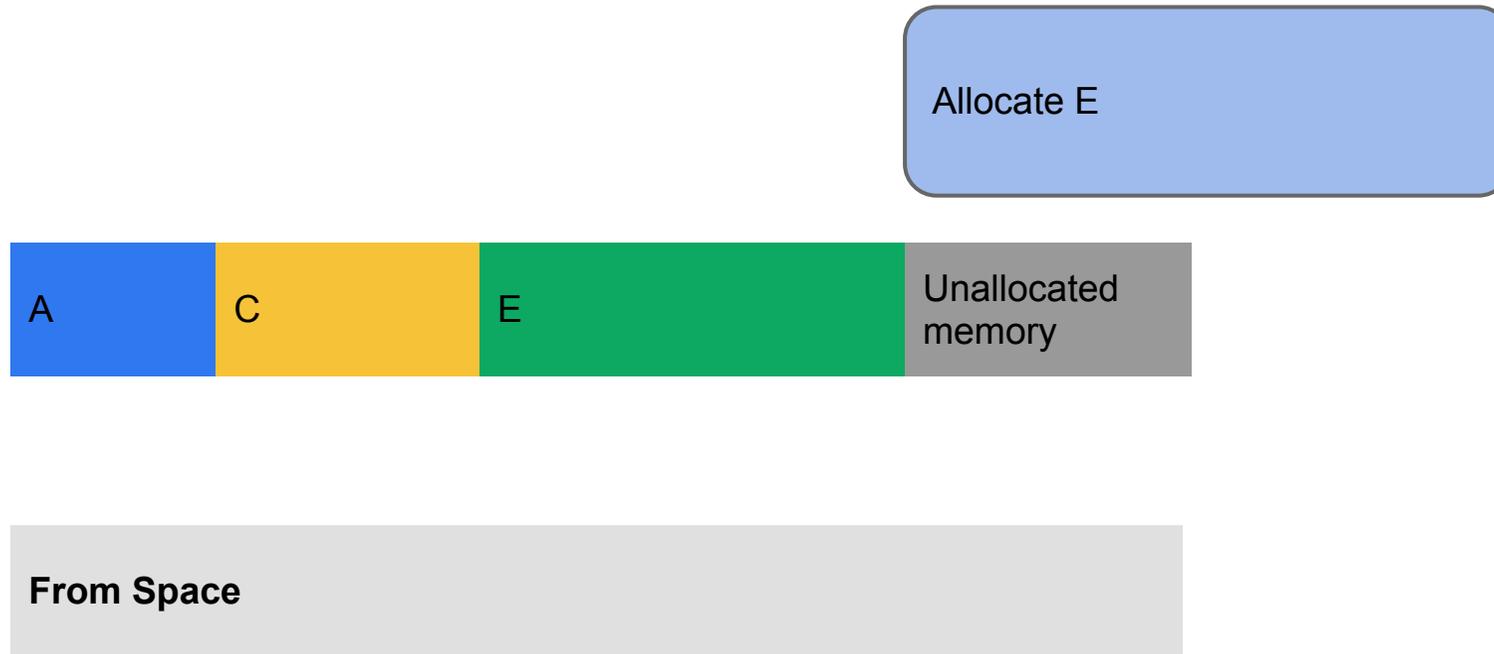
Young Generation In Action



Young Generation In Action



Young Generation In Action



How does V8 manage memory?

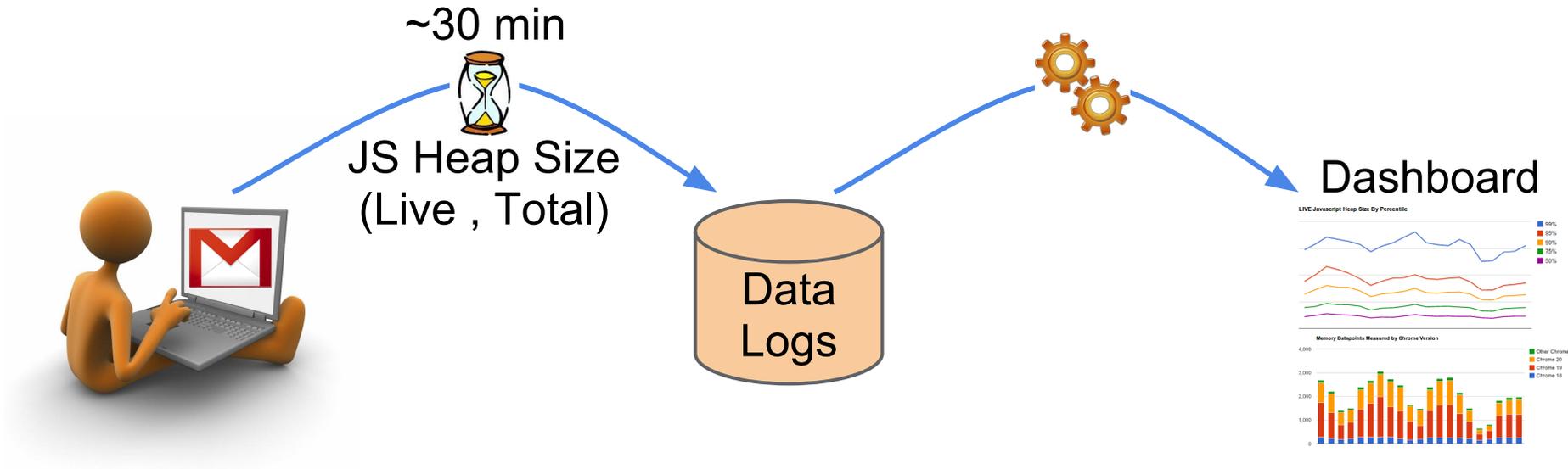
- Each allocation moves you closer to a collection
 - Not always obvious when you are allocating
- Collection pauses your application
 - Higher latency
 - Dropped frames
 - Unhappy users





Tools & Techniques

Collecting field measurements



`window.performance.memory`

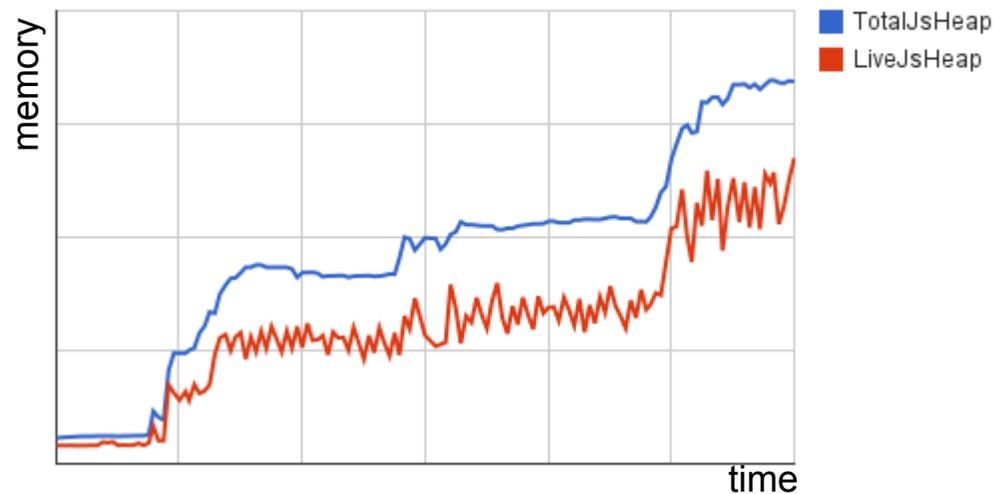
- Enabled by default in Chrome 22
- 3 values returned:
 - **jsHeapSizeLimit** - the amount of memory (in bytes) that the JavaScript heap is limited to
 - **totalJSHeapSize** - the amount of memory (in bytes) that the JavaScript heap has allocated, including free space
 - **usedJSHeapSize** - the amount of memory (in bytes) currently being used



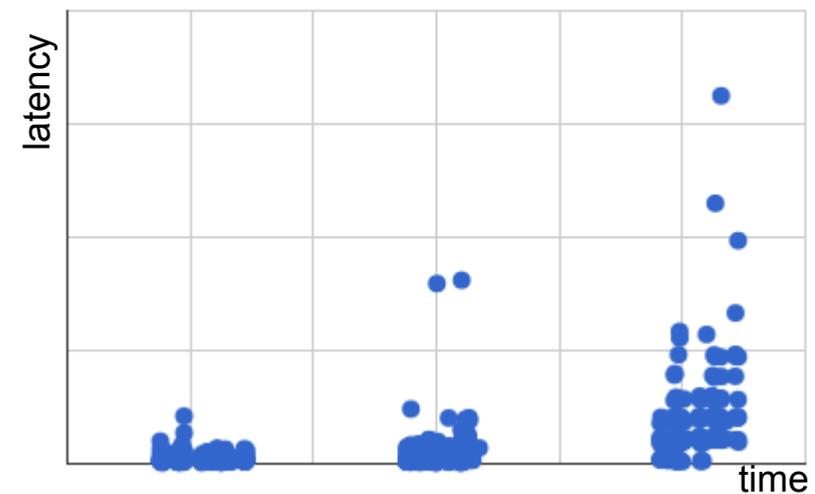
Performance/Memory Tradeoff?

- Common belief:
More Memory == Better Performance
- Reality:

Client Side Memory Footprint



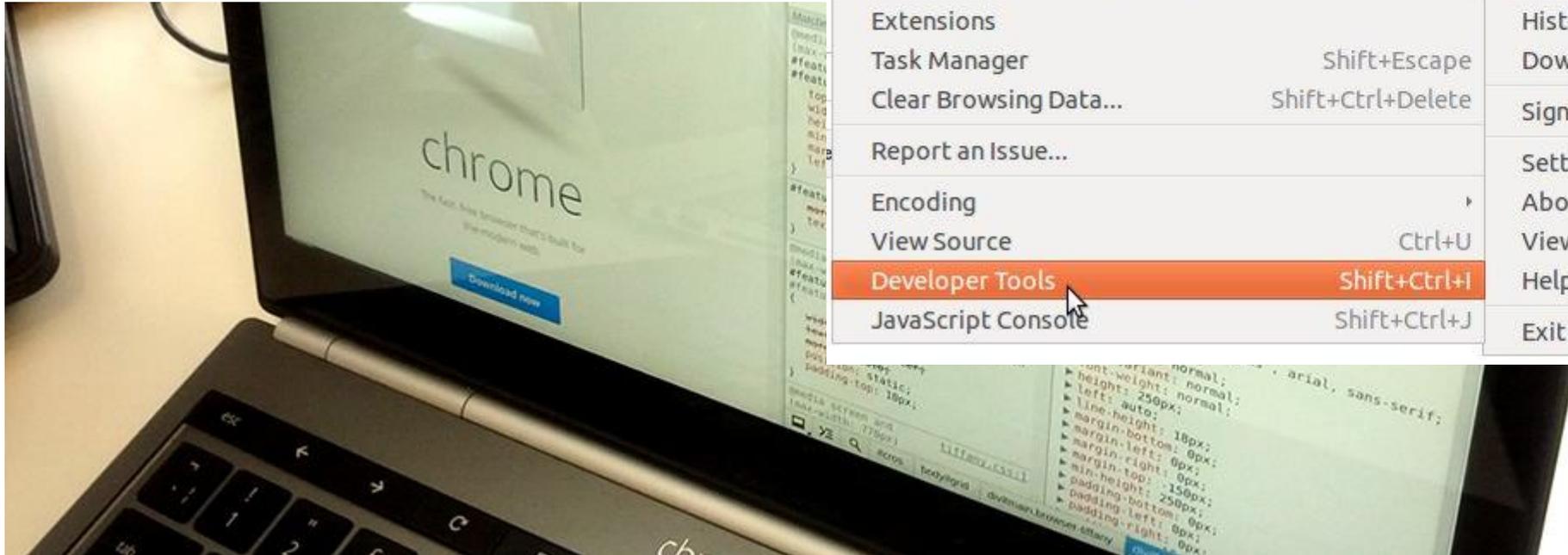
Client-Side Latencies



*Increased memory footprint correlates with **increased latencies and variance***



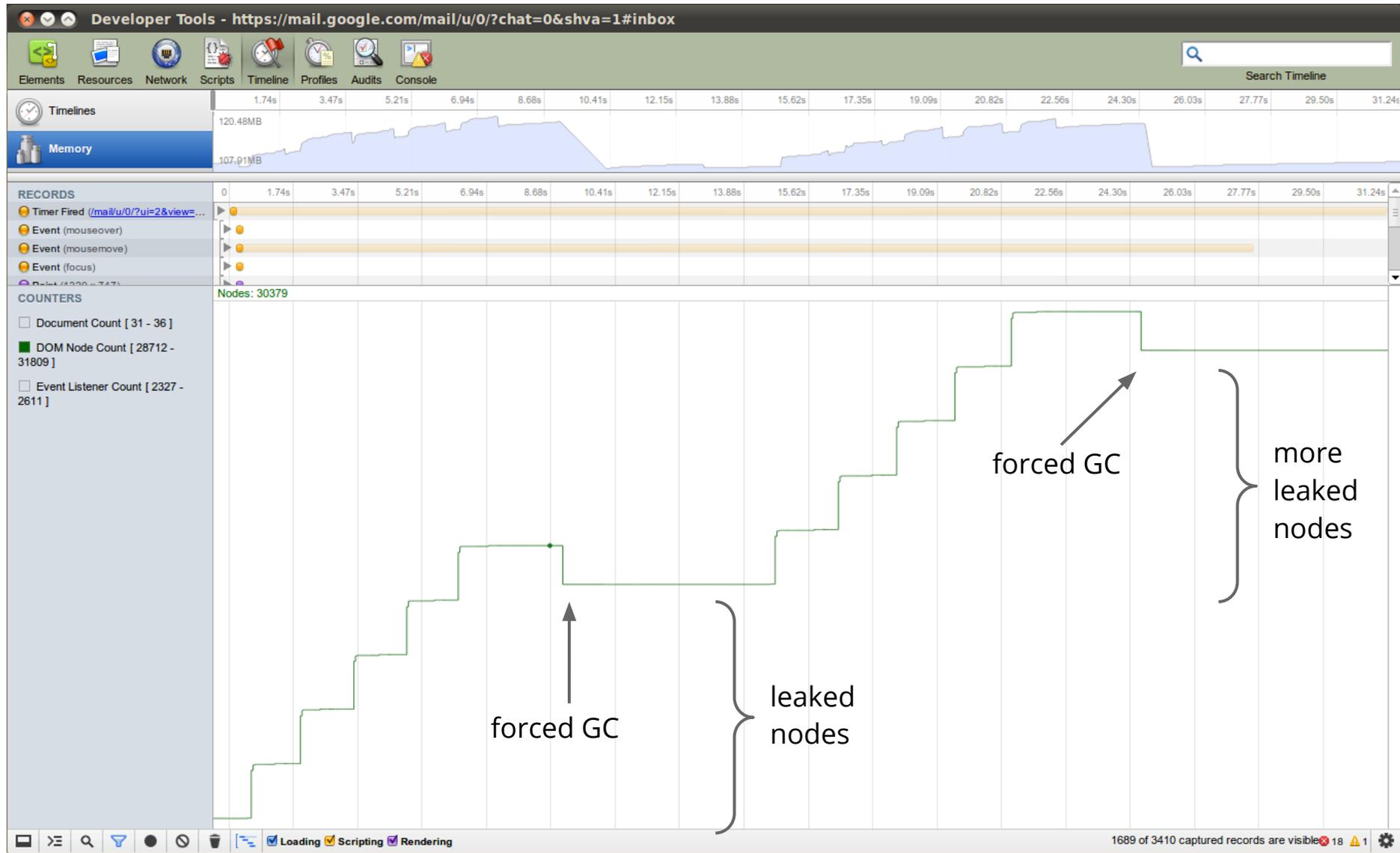
Chrome Developer Tools



Option	Shortcut
New Tab	Ctrl+T
New Window	Ctrl+N
New Incognito Window	Shift+Ctrl+N
Bookmarks	
Edit	Cut Copy Paste
Zoom	- 100% +
Save Page As...	Ctrl+S
Find...	Ctrl+F
Print...	Ctrl+P
Tools	
History	Ctrl+H
Downloads	Ctrl+J
Sign in to Chrome...	
Settings	
About Google Chrome	
View Background Pages (3)	
Help	
Exit	Shift+Ctrl+Q



DevTools Memory Timeline



DevTools Object Tracker



Elements Resources Network Sources Timeline Profiles Audits Console

747 ms | 1.49 s | 2.24 s | 2.99 s | 3.73 s | 4.48 s | 5.23 s | 5.97 s | 6.72 s | 7.47 s | 8.22 s | 8.96 s | 9.71 s | 10.46 s | 11.20 s | 11.95 s

Profiles

HEAP SNAPSHOTS

Snapshot 1
5.2 MB

Class filter

Constructor	Distance	Objects Count	Shallow Size	Retained Size
▼ HTMLDivElement	2	14 0%	552 0%	2 399 160 44%
▶ HTMLDivElement @80699	3		32 0%	32 0%
▶ HTMLDivElement @80863	2		40 0%	40 0%
▶ HTMLDivElement @121131	3		40 0%	399 848 7%
▶ HTMLDivElement @121135	5		40 0%	399 808 7%
▶ HTMLDivElement @181151	3		40 0%	40 0%
▶ HTMLDivElement @181155	2		40 0%	399 808 7%
▶ HTMLDivElement @201165	3		40 0%	40 0%
▶ HTMLDivElement @201169	2		40 0%	399 808 7%
▶ HTMLDivElement @221175	3		40 0%	40 0%
▶ HTMLDivElement @221179	2		40 0%	399 808 7%
▶ HTMLDivElement @241177	3		40 0%	40 0%
▶ HTMLDivElement @241181	2		40 0%	399 808 7%
▶ HTMLDivElement @261183	2		40 0%	40 0%
▶ HTMLDivElement @261187	2		40 0%	399 808 7%
▶ Array	3	6 0%	192 0%	2 398 368 44%
▶ Object	5	3 0%	72 0%	456 0%
▶ CSSStyleDeclaration	5	1 0%	24 0%	344 0%
▶ MouseEvent	5	1 0%	32 0%	184 0%
▶ UIEvent	5	1 0%	32 0%	184 0%

Object's retaining tree

Object	Shallow Size	Retained Size	Distance
▶ __proto__ in HTMLDivElement @121131	40 0%	399 848 7%	3
▶ __proto__ in HTMLDivElement @221175	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @241177	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @201165	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @181151	40 0%	40 0%	3

Summary All objects Selected size: 2.3 MB



DevTools Object Tracker

The screenshot displays the Chrome DevTools Object Tracker interface. At the top, a timeline shows memory allocation events as vertical bars. A red box highlights a specific period on the timeline. Below the timeline, a table lists objects with columns for Constructor, Distance, Objects Count, Shallow Size, and Retained Size. A callout box with an arrow points to the 'Distance' column, stating: "Color-coded bars identify new objects allocated during the timeline".

Constructor	Distance	Objects Count	Shallow Size	Retained Size
HTMLDivElement	2	14 0%	552 0%	2 399 160 44%
▶ HTMLDivElement @80699	3		32 0%	32 0%
▶ HTMLDivElement @80863	2		40 0%	40 0%
▶ HTMLDivElement @121131	3		40 0%	399 848 7%
▶ HTMLDivElement @121135	5		40 0%	399 808 7%
▶ HTMLDivElement @181151	3		40 0%	40 0%
▶ HTMLDivElement @181155	2		40 0%	399 808 7%
▶ HTMLDivElement @201165			40 0%	40 0%
▶ HTMLDivElement @201169			40 0%	399 808 7%
▶ HTMLDivElement @221175			40 0%	40 0%
▶ HTMLDivElement @221179			40 0%	399 808 7%
▶ HTMLDivElement @241177			40 0%	40 0%
▶ HTMLDivElement @241181			40 0%	399 808 7%
▶ HTMLDivElement @261183			40 0%	40 0%
▶ HTMLDivElement @261187			40 0%	399 808 7%
▶ Array		6 0%	192 0%	2 398 368 44%
▶ Object		3 0%	72 0%	456 0%
▶ CSSStyleDeclaration		1 0%	24 0%	344 0%
▶ MouseEvent	5	1 0%	32 0%	184 0%
▶ UIEvent	5	1 0%	32 0%	184 0%

Object's retaining tree

Object	Shallow Size	Retained Size	Distance
▶ __proto__ in HTMLDivElement @121131	40 0%	399 848 7%	3
▶ __proto__ in HTMLDivElement @221175	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @241177	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @201165	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @181151	40 0%	40 0%	3

Summary | All objects | Selected size: 2.3 MB



DevTools Object Tracker

The screenshot displays the Chrome DevTools Object Tracker interface. At the top, a red box highlights the 'Adjustable timeframe selector' which shows a timeline from 747 ms to 11.95 s. Below this, a table lists memory objects. The 'HTMLDivElement' constructor is expanded, showing 14 objects. A callout box with the text 'Adjustable timeframe selector' points to the red box.

Constructor	Distance	Objects Count	Shallow Size	Retained Size
HTMLDivElement	2	14 0%	552 0%	2 399 160 44%
▶ HTMLDivElement @80699	3		32 0%	32 0%
▶ HTMLDivElement @80863	2		40 0%	40 0%
▶ HTMLDivElement @121131	3		40 0%	399 848 7%
▶ HTMLDivElement @121135	5		40 0%	399 808 7%
▶ HTMLDivElement @181151	3		40 0%	40 0%
▶ HTMLDivElement @181155	2		40 0%	399 808 7%
▶ HTMLDivElement @201165			40 0%	40 0%
▶ HTMLDivElement @201169			40 0%	399 808 7%
▶ HTMLDivElement @221175			40 0%	40 0%
▶ HTMLDivElement @221179			40 0%	399 808 7%
▶ HTMLDivElement @241177			40 0%	40 0%
▶ HTMLDivElement @241181			40 0%	399 808 7%
▶ HTMLDivElement @261183	2		40 0%	40 0%
▶ HTMLDivElement @261187	2		40 0%	399 808 7%
▶ Array	3	6 0%	192 0%	2 398 368 44%
▶ Object	5	3 0%	72 0%	456 0%
▶ CSSStyleDeclaration	5	1 0%	24 0%	344 0%
▶ MouseEvent	5	1 0%	32 0%	184 0%
▶ UIEvent	5	1 0%	32 0%	184 0%

Object	Shallow Size	Retained Size	Distance
▶ __proto__ in HTMLDivElement @121131	40 0%	399 848 7%	3
▶ __proto__ in HTMLDivElement @221175	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @241177	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @201165	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @181151	40 0%	40 0%	3



DevTools Object Tracker

The screenshot displays the Chrome DevTools Object Tracker interface. The top navigation bar includes tabs for Elements, Resources, Network, Sources, Timeline, Profiles, Audits, and Console. The main area is divided into a left sidebar and a main table. The sidebar shows 'Profiles' with a 'HEAP SNAPSHOTS' section containing 'Snapshot 1' (5.2 MB). A red box highlights the main table, which lists various objects with columns for Constructor, Distance, Objects Count, Shallow Size, and Retained Size. An arrow points from a text box labeled 'Heap contents' to the table. Below the table, the 'Object's retaining tree' section is visible, showing the prototype chain for selected HTMLDivElement objects.

Constructor	Distance	Objects Count	Shallow Size	Retained Size
HTMLDivElement	2	14 0%	552 0%	2 399 160 44%
▶ HTMLDivElement @80699	3		32 0%	32 0%
▶ HTMLDivElement @80863	2		40 0%	40 0%
▶ HTMLDivElement @121131	3		40 0%	399 848 7%
▶ HTMLDivElement @121135	5		40 0%	399 808 7%
▶ HTMLDivElement @181151	3		40 0%	40 0%
▶ HTMLDivElement @181155	2		40 0%	399 808 7%
▶ HTMLDivElement @201165	3		40 0%	40 0%
▶ HTMLDivElement @201169	2		40 0%	399 808 7%
▶ HTMLDivElement @221175	3		40 0%	40 0%
▶ HTMLDivElement @221179	2		40 0%	399 808 7%
▶ HTMLDivElement @241177	3		40 0%	40 0%
▶ HTMLDivElement @241181	2		40 0%	399 808 7%
▶ HTMLDivElement @261183	2		40 0%	40 0%
▶ HTMLDivElement @261187	2		40 0%	399 808 7%
▶ Array	3	6 0%	192 0%	2 398 368 44%
▶ Object	5	3 0%	72 0%	456 0%
▶ CSSStyleDeclaration	5	1 0%	24 0%	344 0%
▶ MouseEvent	5	1 0%	32 0%	184 0%
▶ UIEvent	5	1 0%	32 0%	184 0%

Object's retaining tree

Object	Shallow Size	Retained Size	Distance
▶ __proto__ in HTMLDivElement @121131	40 0%	399 848 7%	3
▶ __proto__ in HTMLDivElement @221175	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @241177	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @201165	40 0%	40 0%	3
▶ __proto__ in HTMLDivElement @181151	40 0%	40 0%	3

Summary All objects Selected size: 2.3 MB

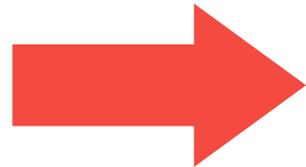




Demo

The Setup

- A simple mail-like app
- Messages are cached for better performance
- Cache size: 5 messages



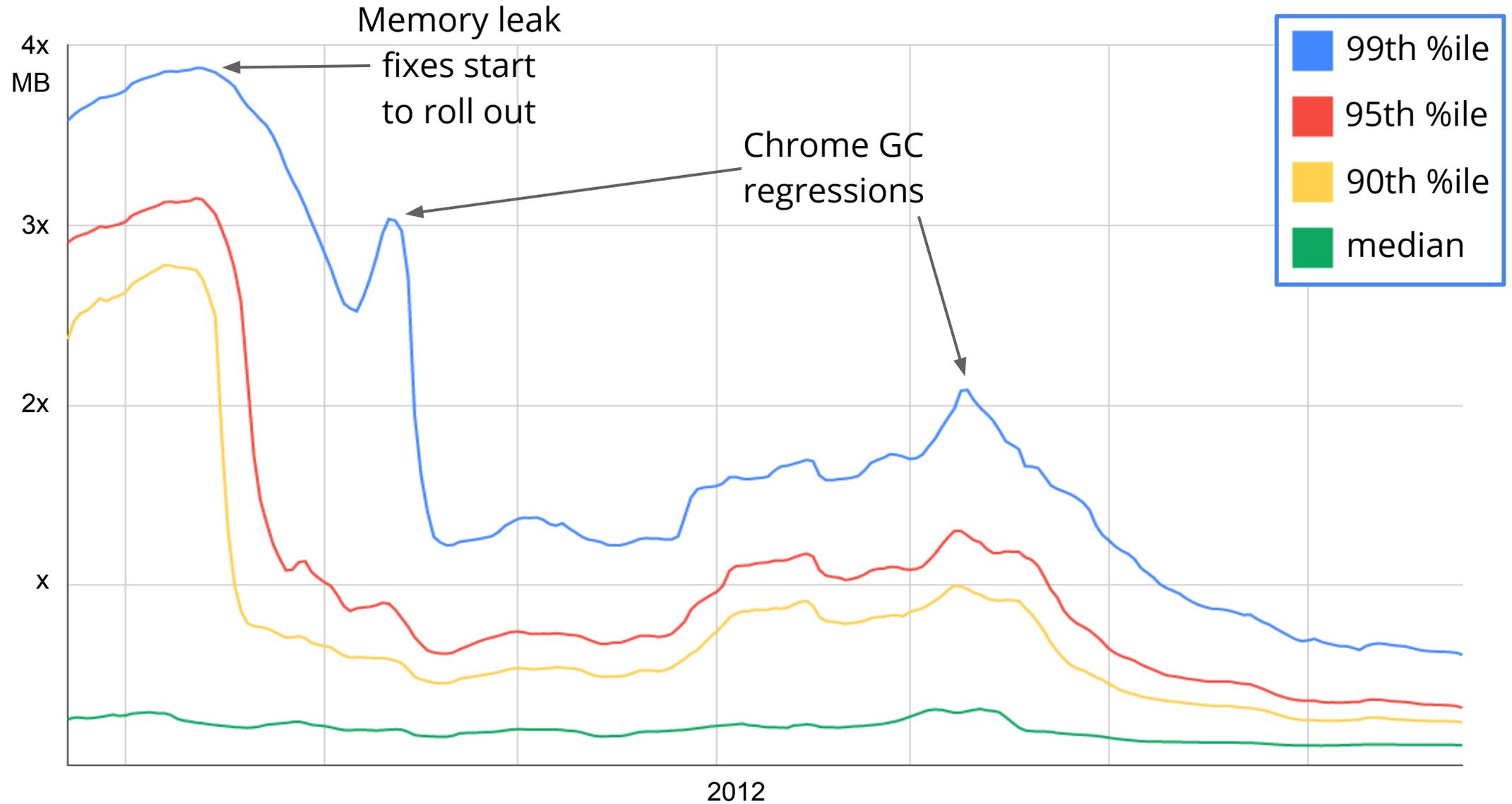
In theory, no more than 5 messages should be resident in memory at any given time...



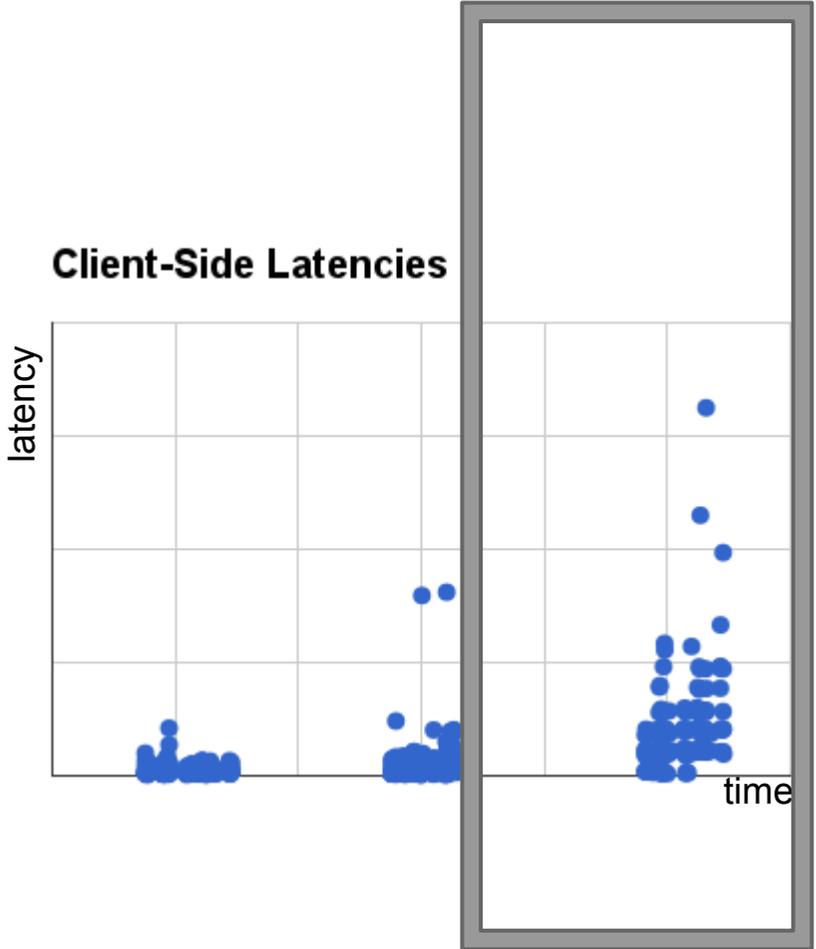
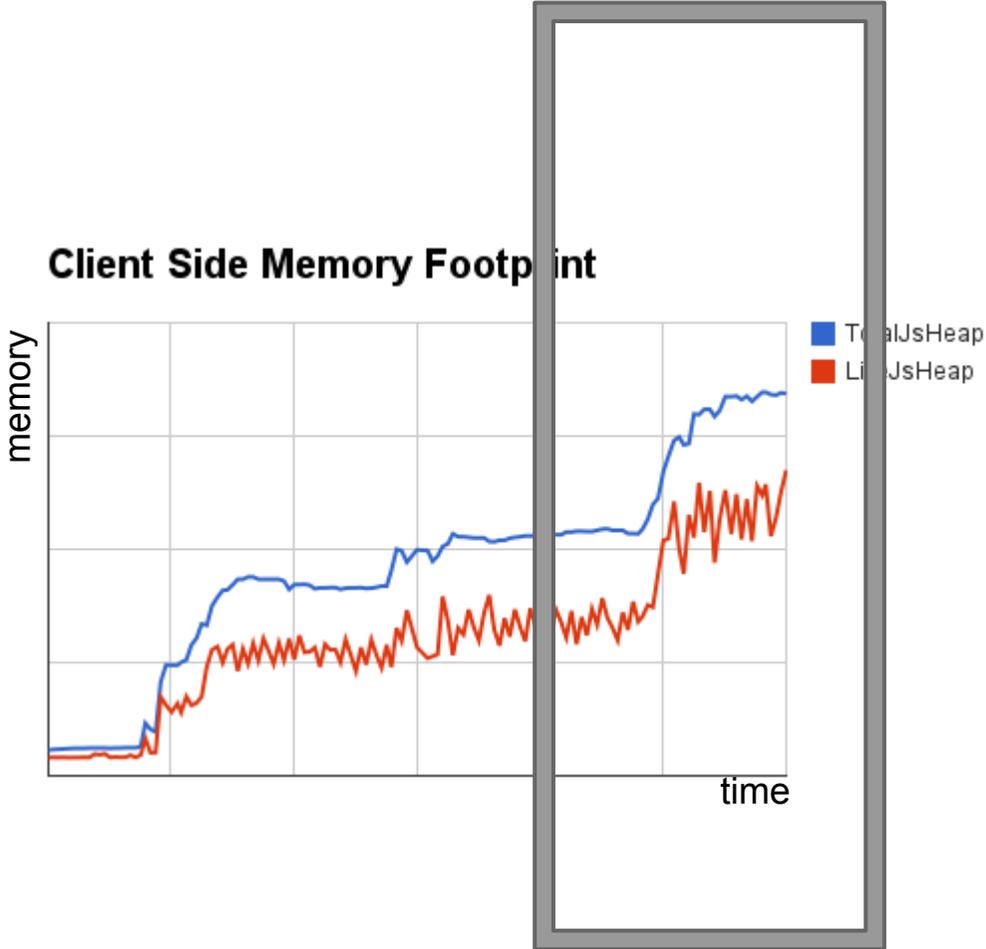


Is it really worth it?

Where are we now?



Call to Action



Call to Action

Ask yourself these questions:

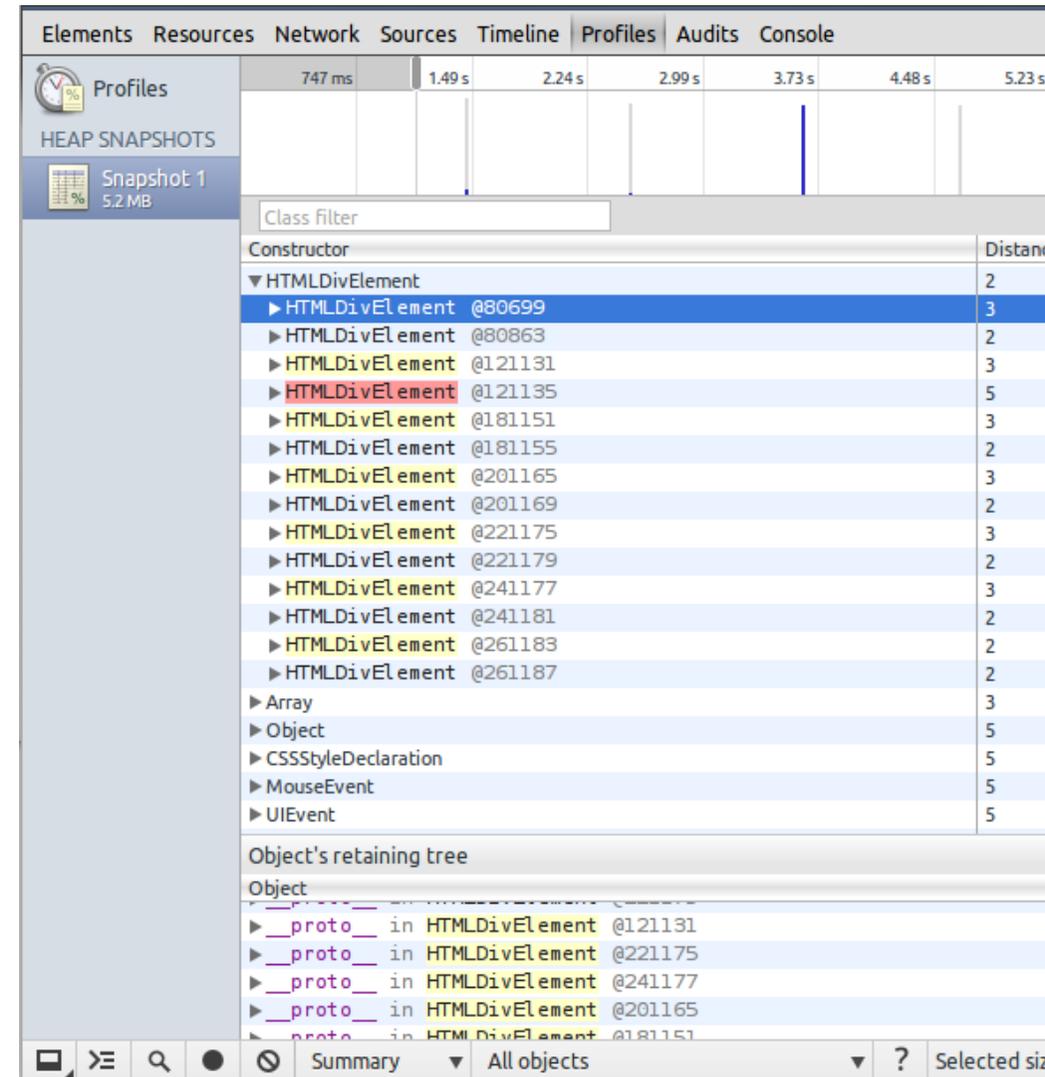
- How much memory is your page using?
- Is your page leak free?
- How frequently are you GCing?



Call to Action

Chrome Developer Tools

- `window.performance.memory`
- Heap Profiler
- Object Tracker
 - Continuous Snapshot Technique



The screenshot shows the Chrome Developer Tools interface, specifically the Profiler and Heap Snapshots panels. The Profiler panel at the top shows a timeline with several snapshots. The Heap Snapshots panel is active, displaying a list of objects. The selected object is HTMLDivElement @80699. The panel also shows the Object's retaining tree.

Constructor	Distance
HTMLDivElement	2
HTMLDivElement @80699	3
HTMLDivElement @80863	2
HTMLDivElement @121131	3
HTMLDivElement @121135	5
HTMLDivElement @181151	3
HTMLDivElement @181155	2
HTMLDivElement @201165	3
HTMLDivElement @201169	2
HTMLDivElement @221175	3
HTMLDivElement @221179	2
HTMLDivElement @241177	3
HTMLDivElement @241181	2
HTMLDivElement @261183	2
HTMLDivElement @261187	2
Array	3
Object	5
CSSStyleDeclaration	5
MouseEvent	5
UIEvent	5

Object's retaining tree

Object
__proto__ in HTMLDivElement @121131
__proto__ in HTMLDivElement @221175
__proto__ in HTMLDivElement @241177
__proto__ in HTMLDivElement @201165
__proto__ in HTMLDivElement @181151



<Thank You!>

Questions!

Be sure to visit Chrome DevRel Office Hours

loreena@google.com
google.com/+LoreenaLee

johnmccutchan@google.com
google.com/+JohnMcCutchan
twitter.com/johnmccutchan



Try the Object Tracker

DEMO

- Source: <http://goo.gl/uI4D4>

ENABLE THE DEVTOOLS OBJECT TRACKER

- Get the latest Chrome Canary
- Go to `about:flags` and enable the Chrome Developer Tools Experiments
- Restart Chrome
- Open DevTools
- Click on Gear > Experiments > Enable heap objects tracking profile type
- Restart DevTools
- The profile panel will now have a 4th snapshot type: Track Allocations

