Google
Developers

A Trip Down Memory Lane with Gmail and Chrome DevTools

Effective Memory Management

Loreena Lee & John McCutchan

Google I/O 13

Performance vs. Memory

My Gmail tab is using a gig of RAM.

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

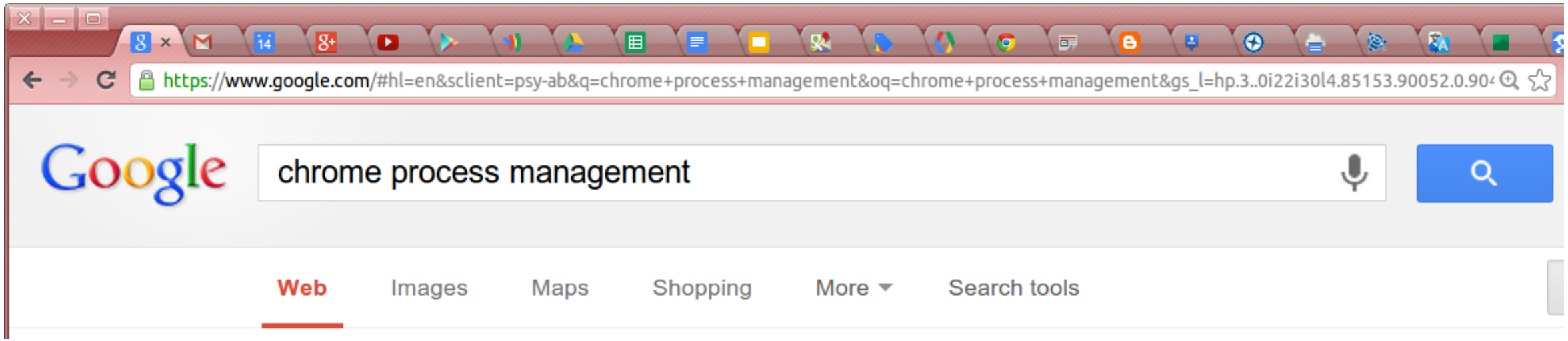
Yeah, but my grandma's Chromebook only has 4GB.

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

Gmail, we have a problem...

- 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

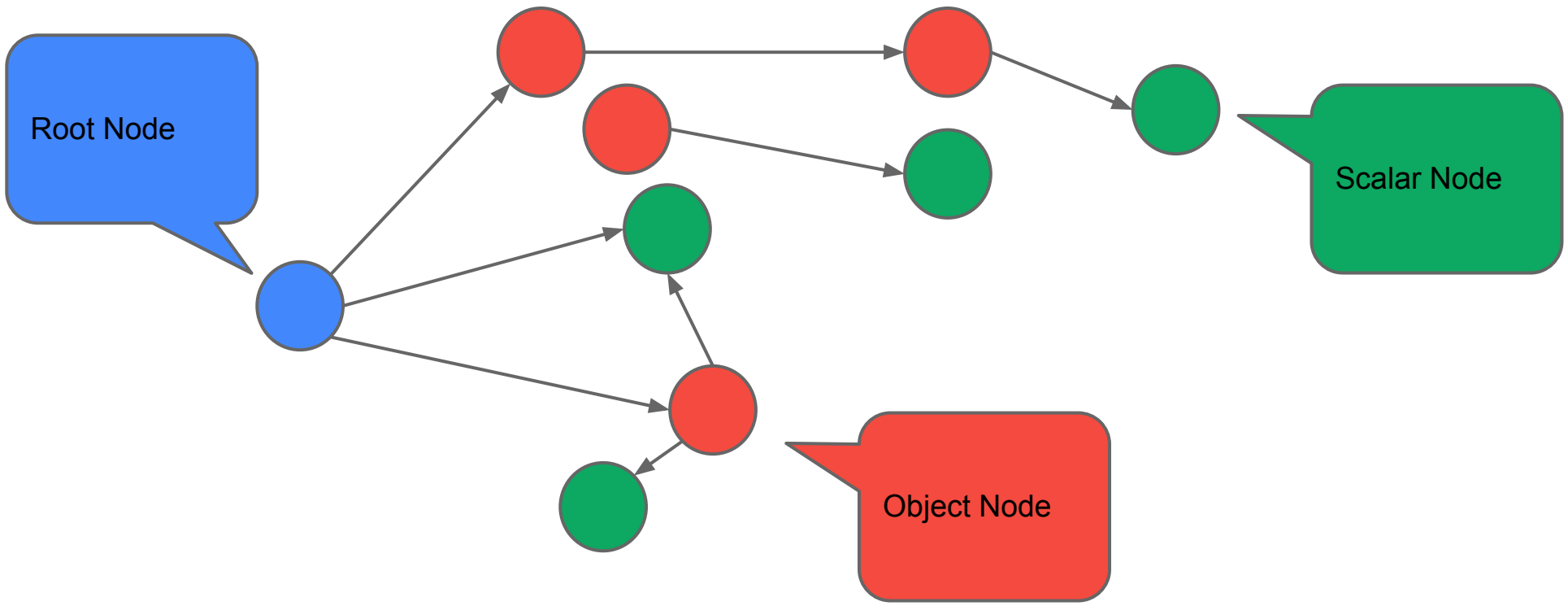
Core Concepts

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

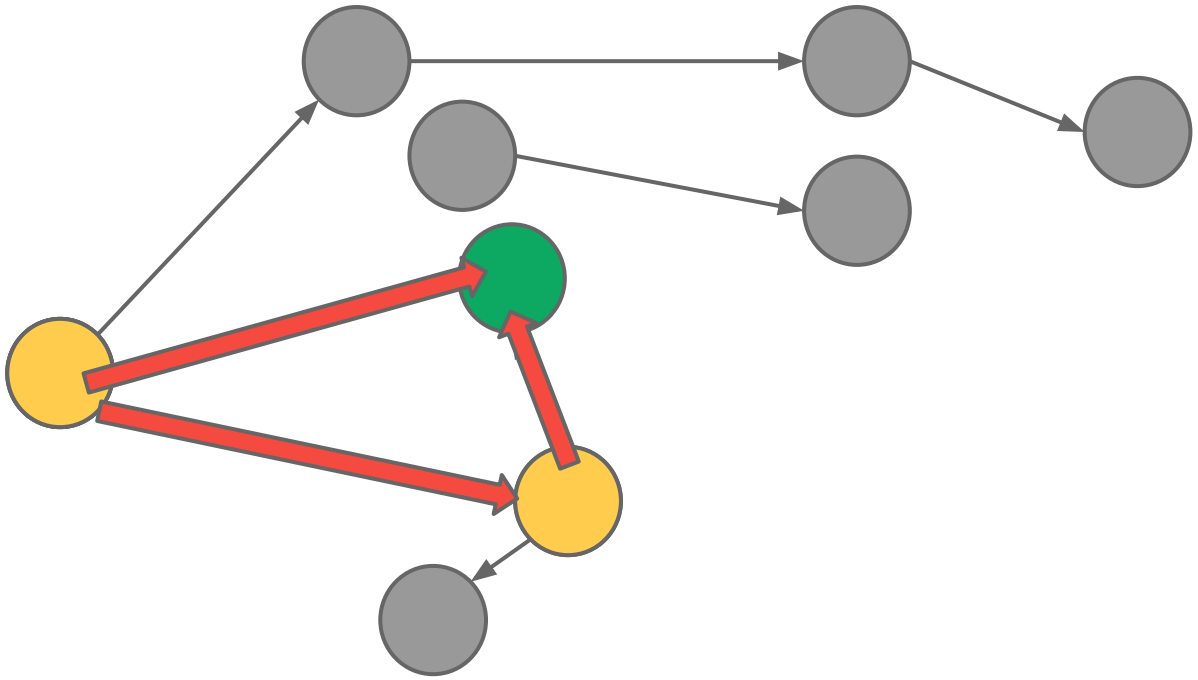
JavaScript value types

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

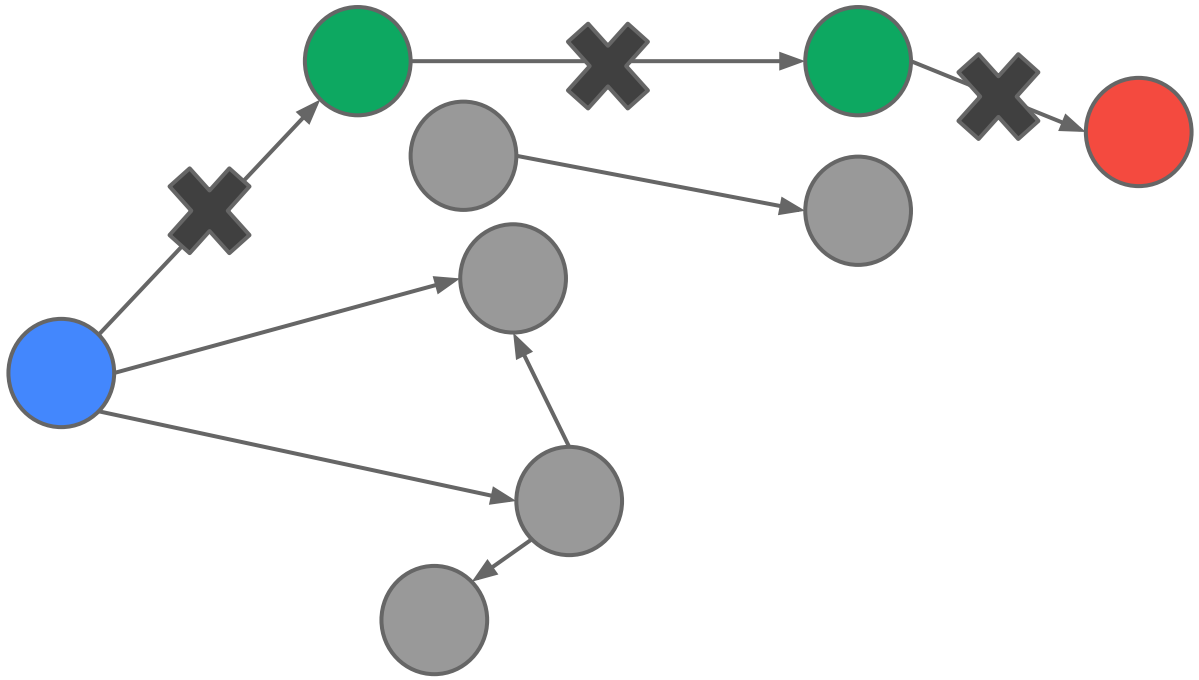
The value graph



A value's retaining path(s)

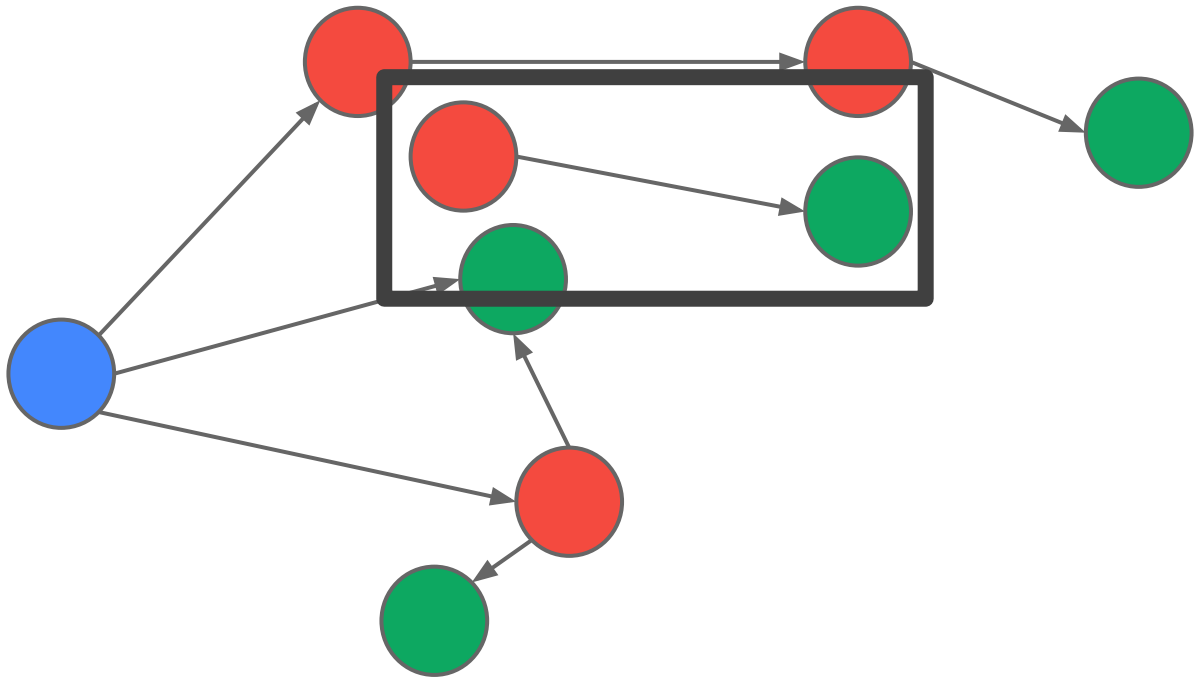


Removing a value from the graph



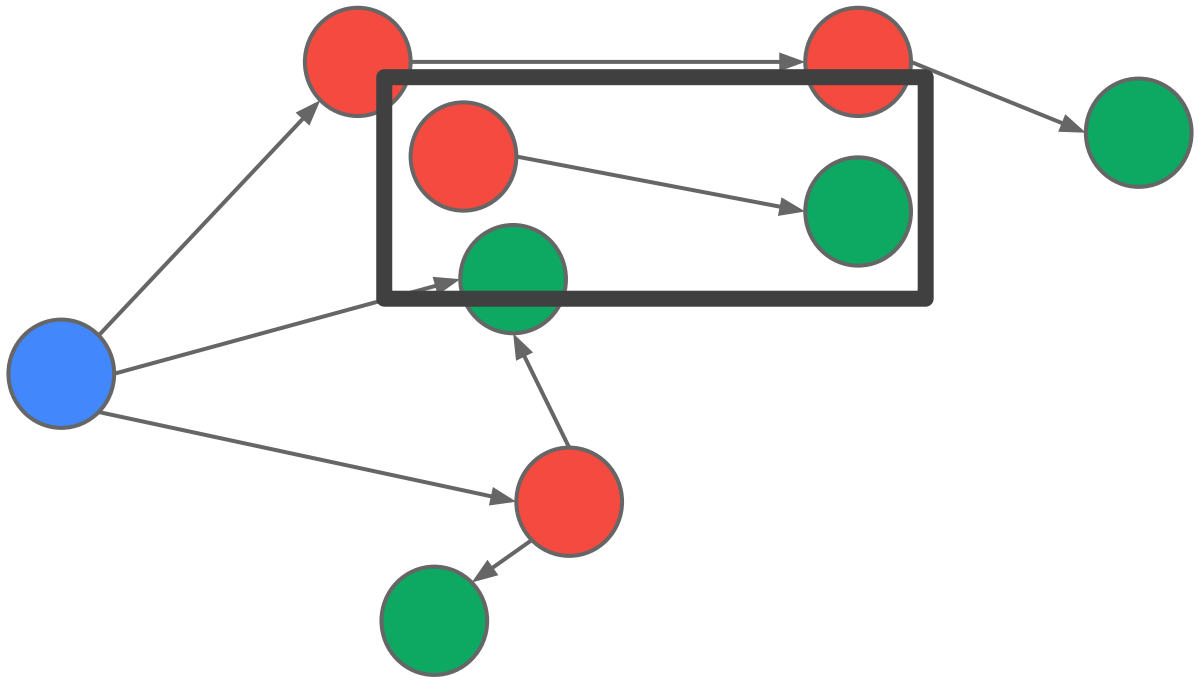
What is garbage?

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

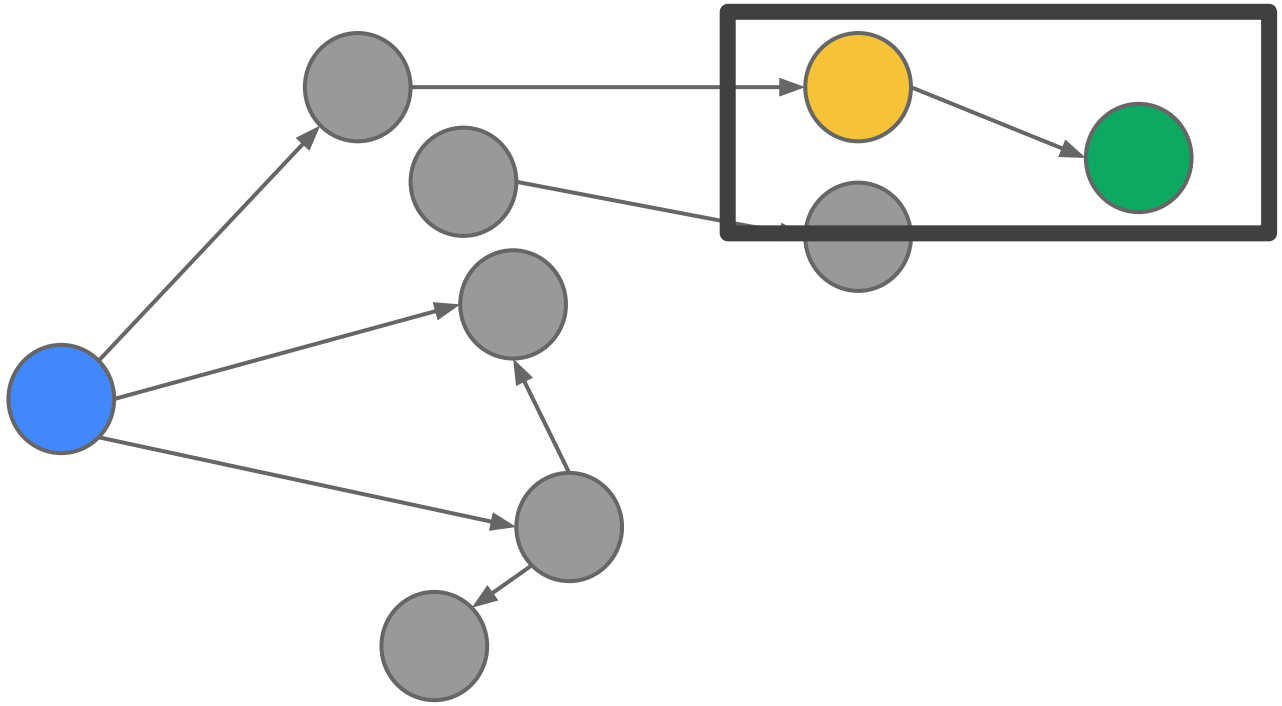


What is garbage collection?

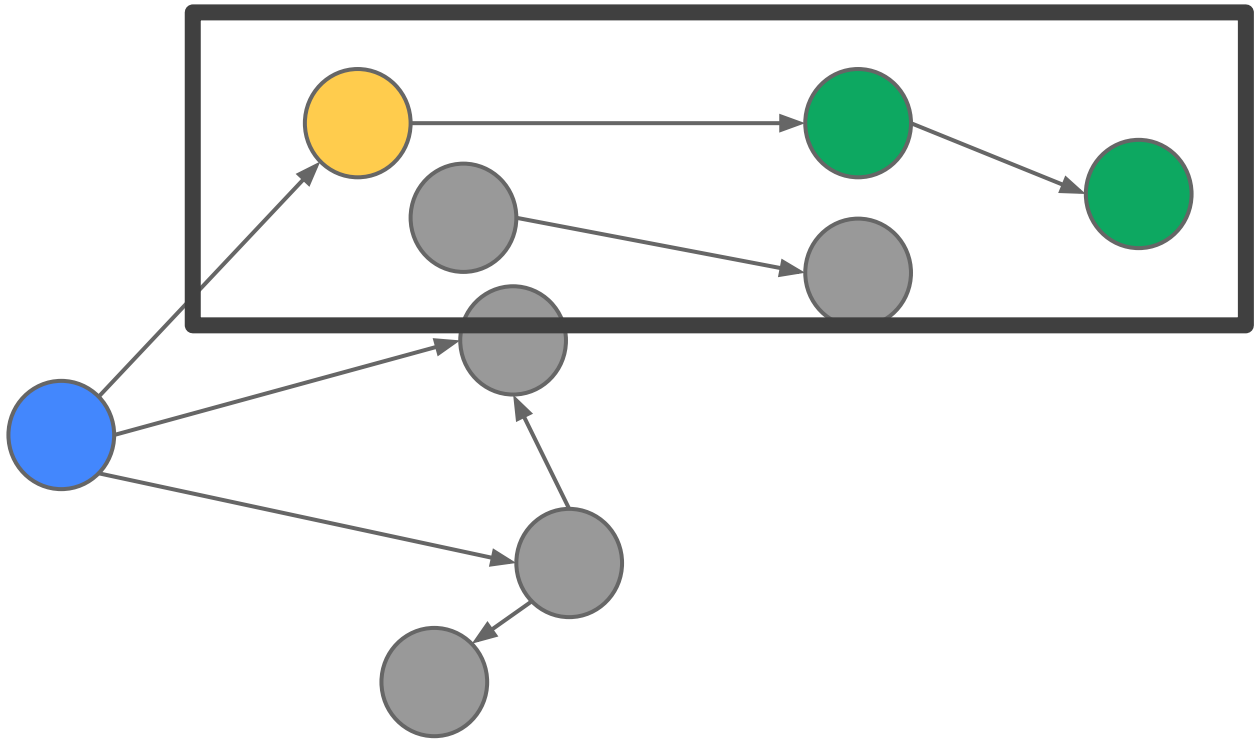
1. Find all live values
2. Return memory used by dead values to system



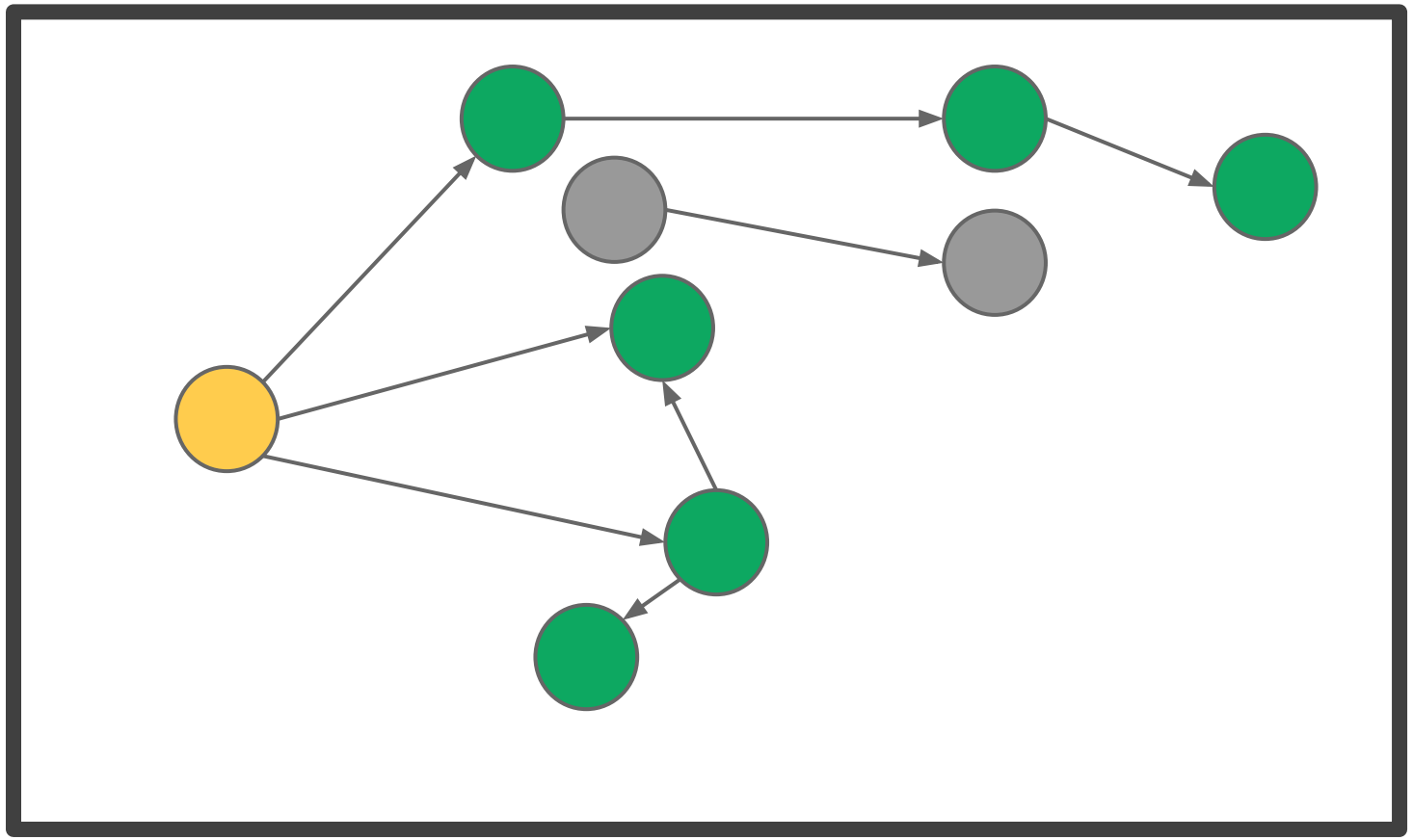
A value's retained size



A value's retained size



A value's retained size



Leaks in JavaScript

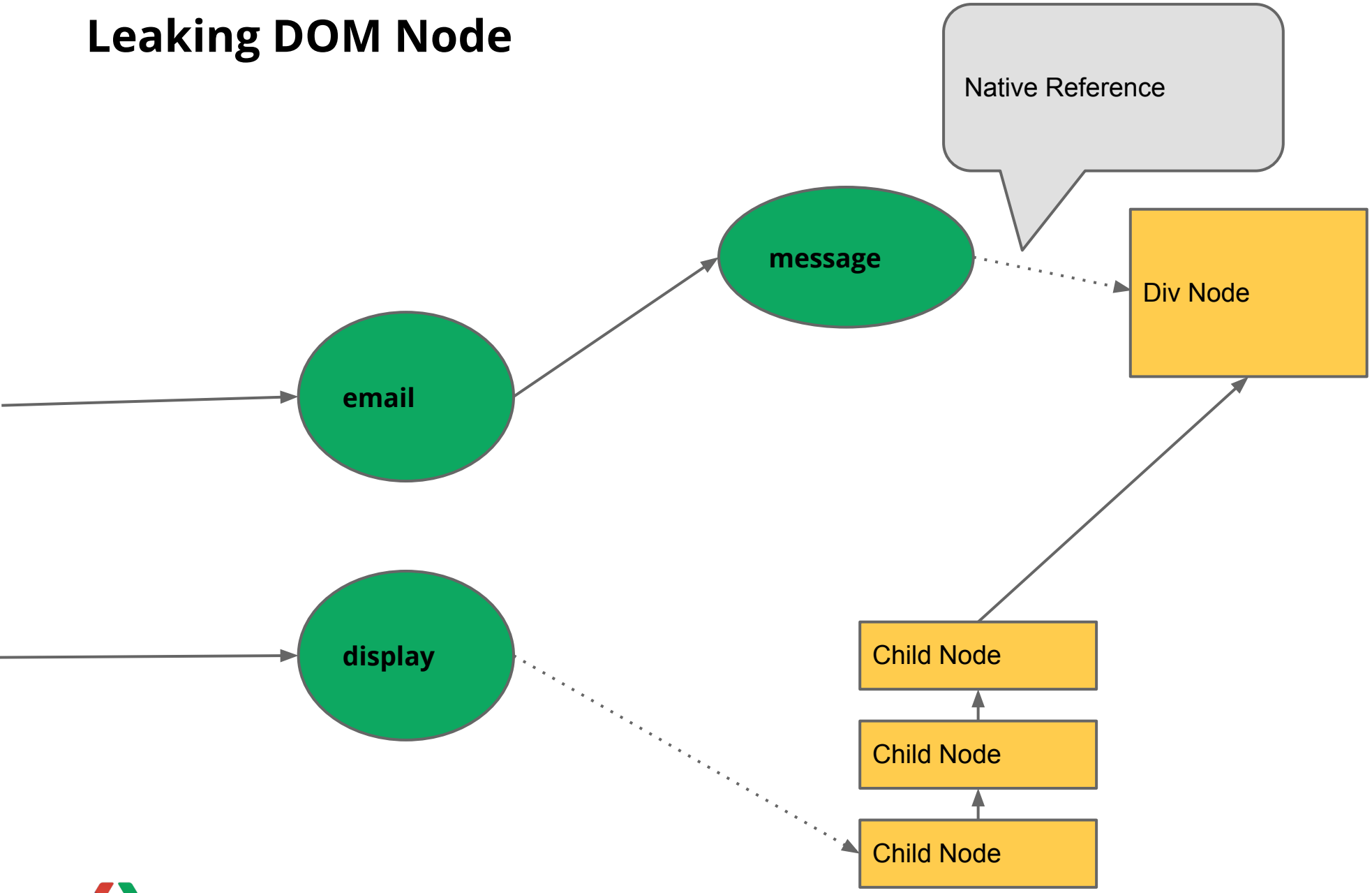
- A value that erroneously still has a retaining path
  - Programmer error

JavaScript

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

display.appendChild(email.message);
```

Leaking DOM Node



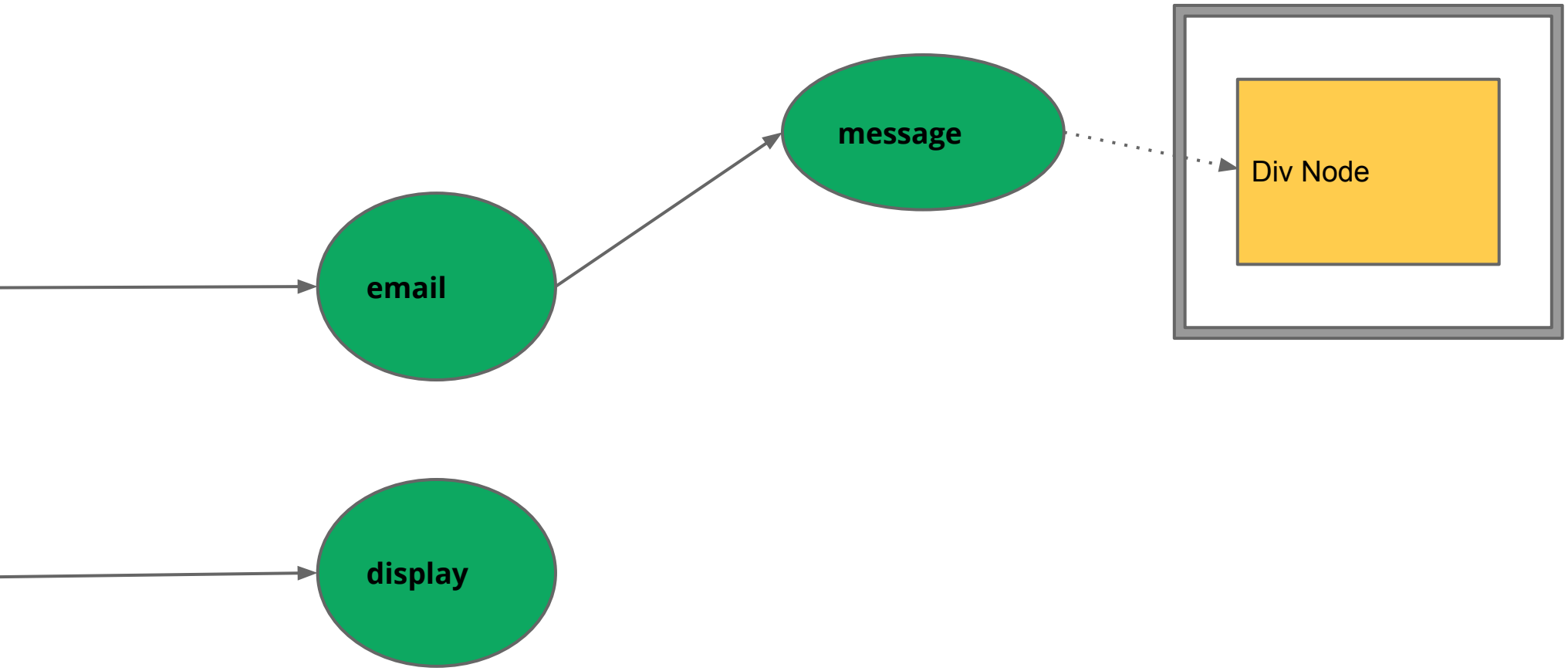
Leaks in JavaScript

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

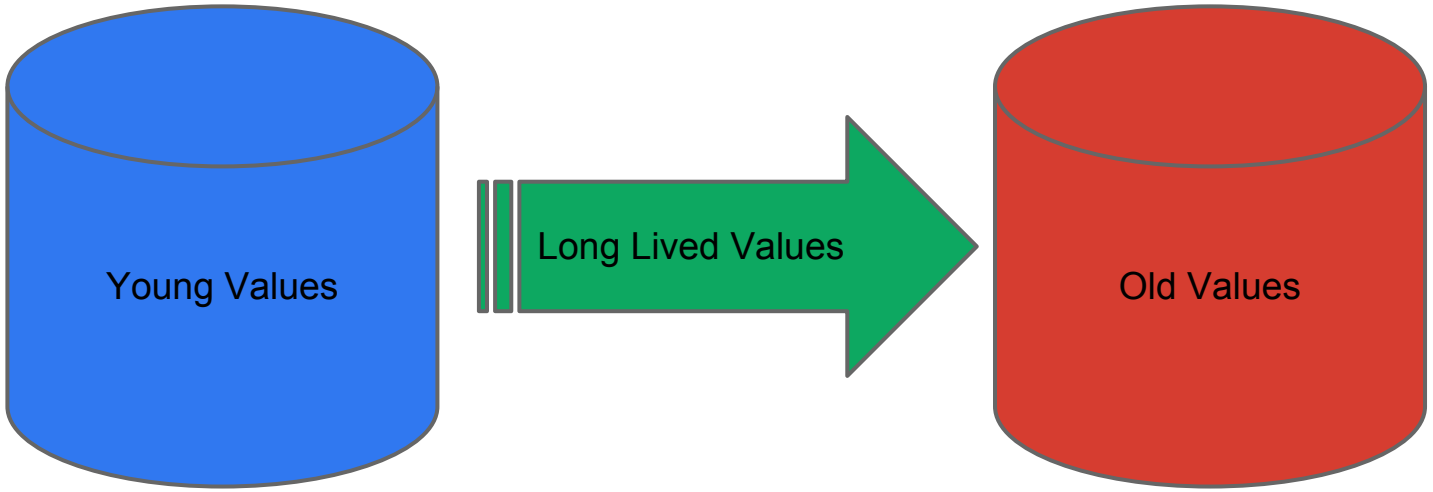
A GC Pause Walkthrough

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

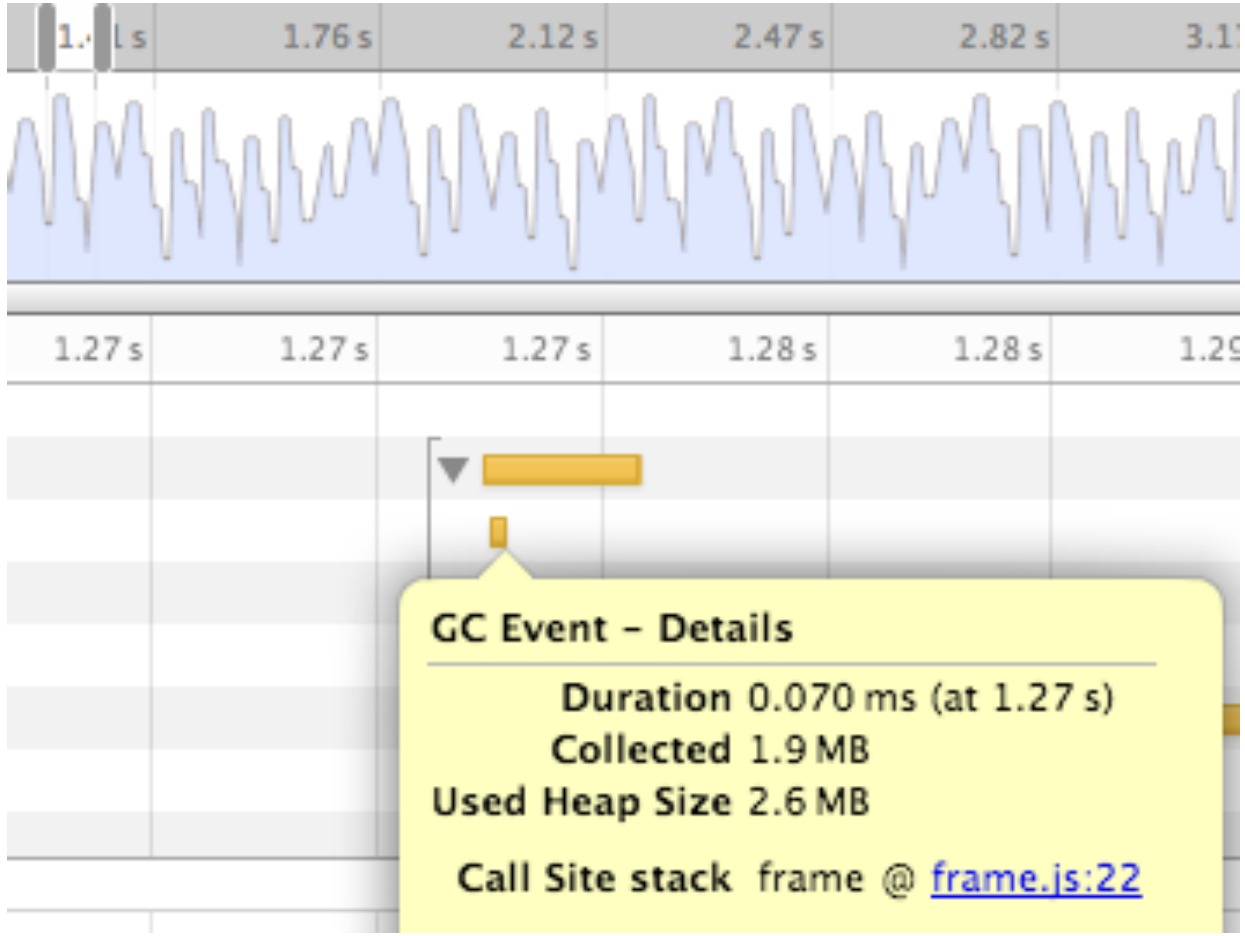
How does V8 manage memory?

- 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



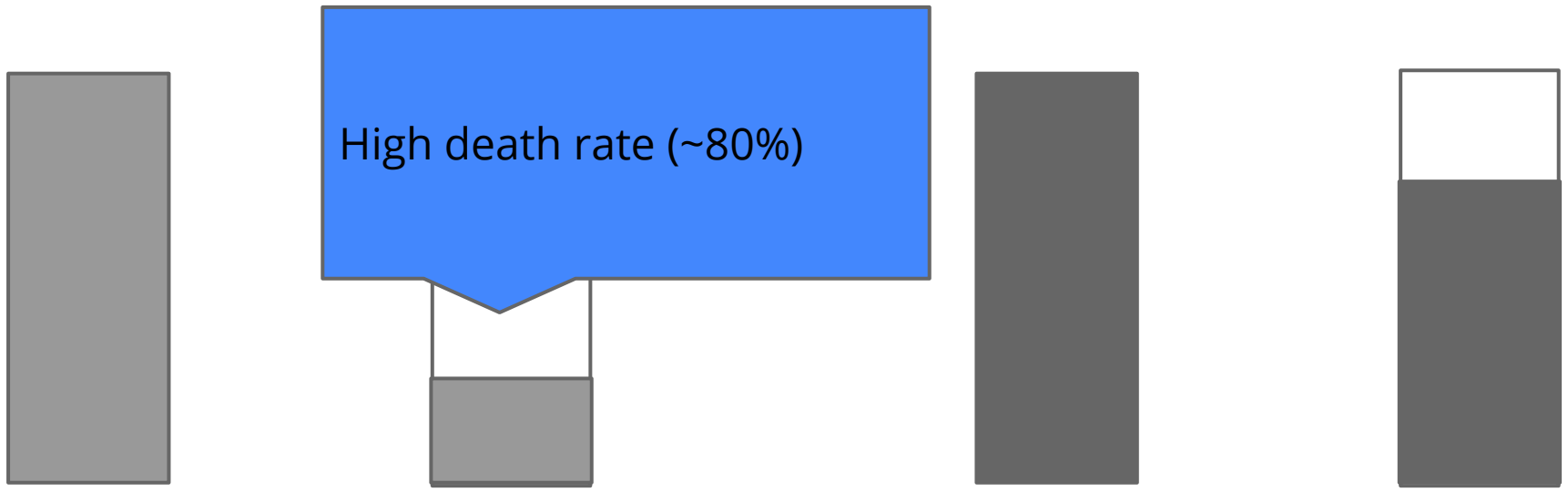
How does V8 manage memory?

- 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

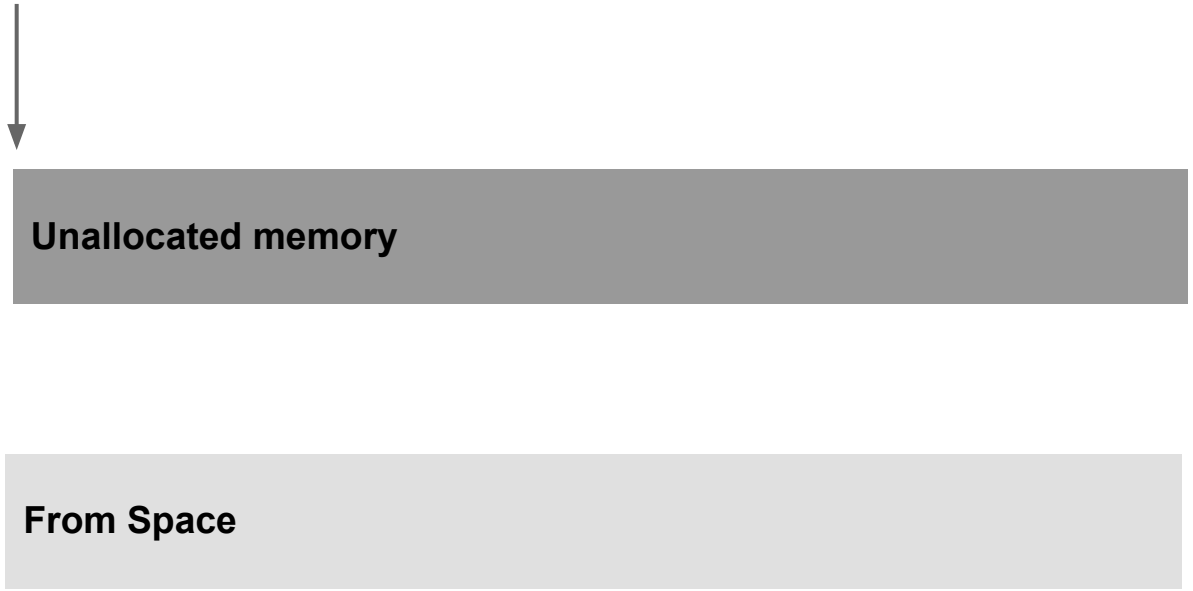
How does V8 manage memory?

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

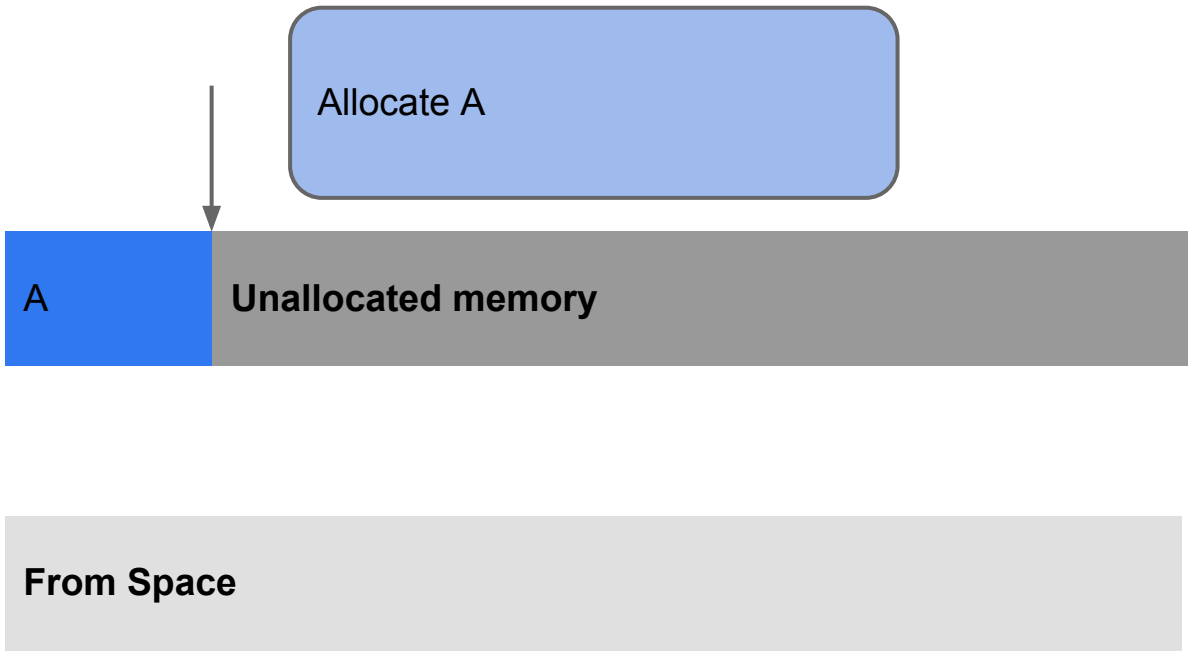


Young Generation In Action

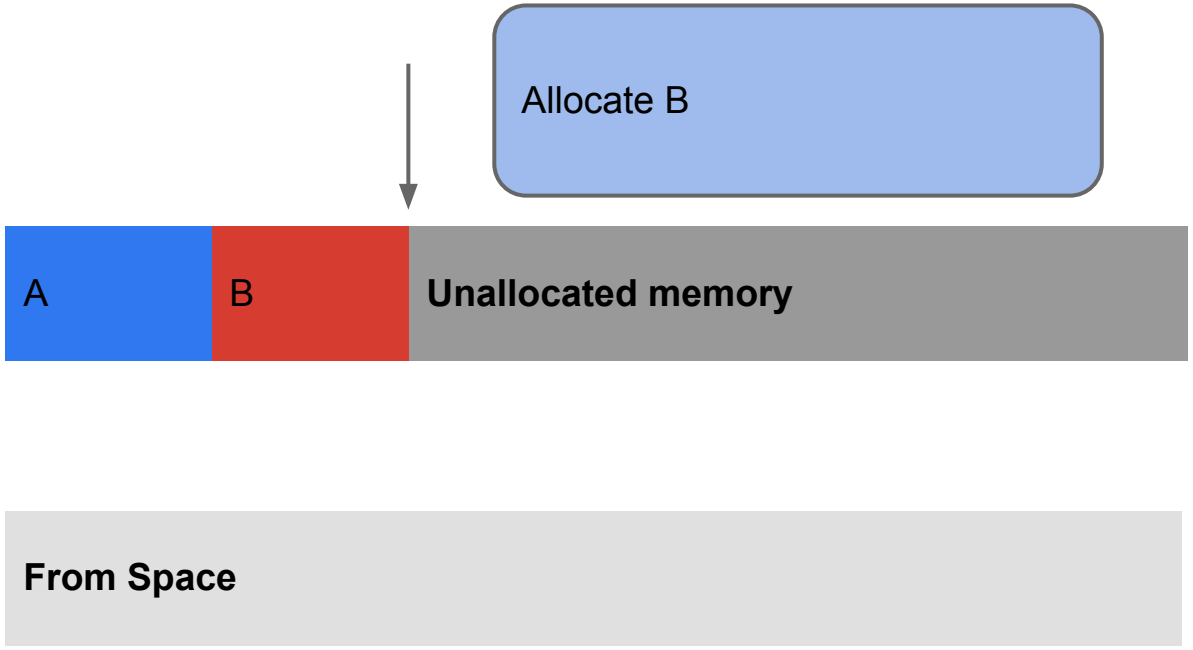
Young Generation In Action



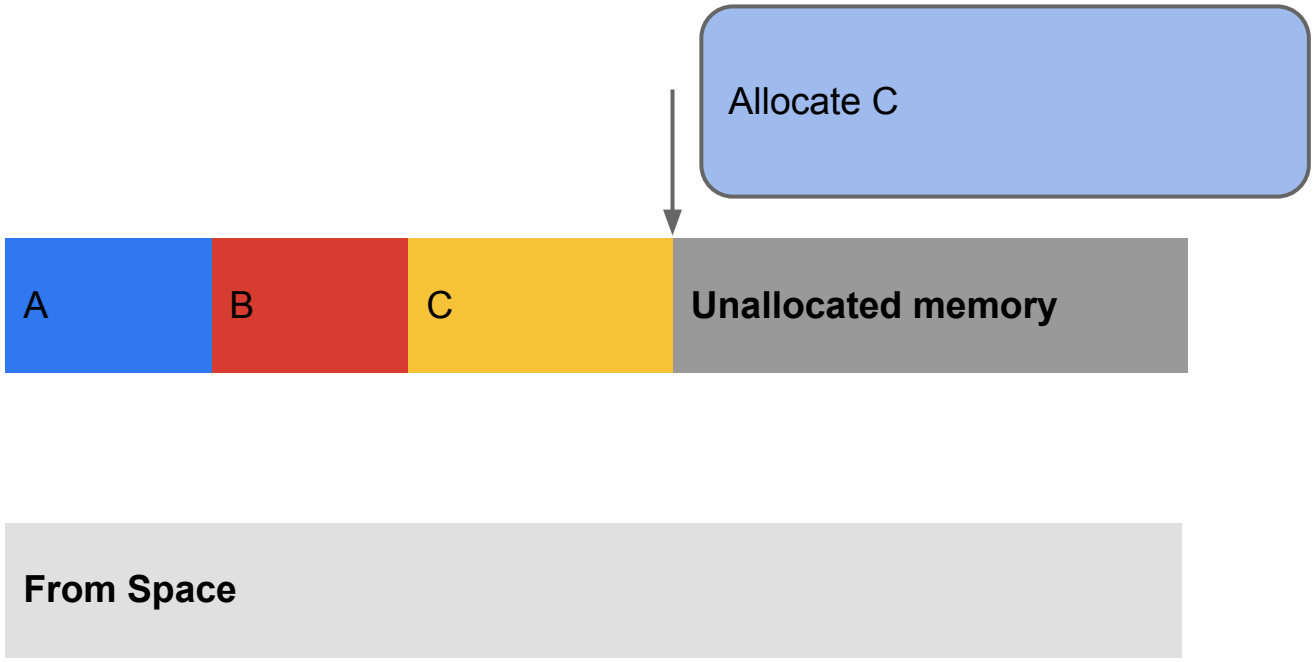
Young Generation In Action



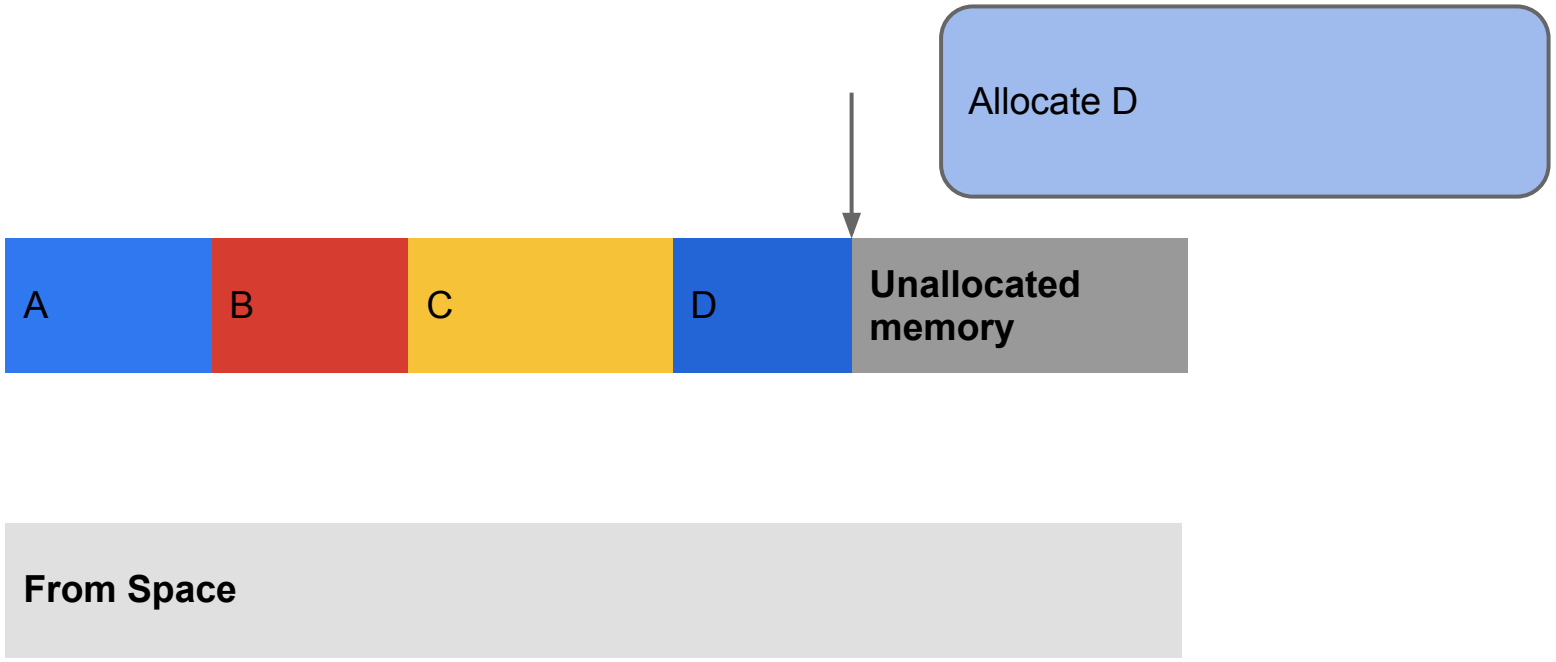
Young Generation In Action



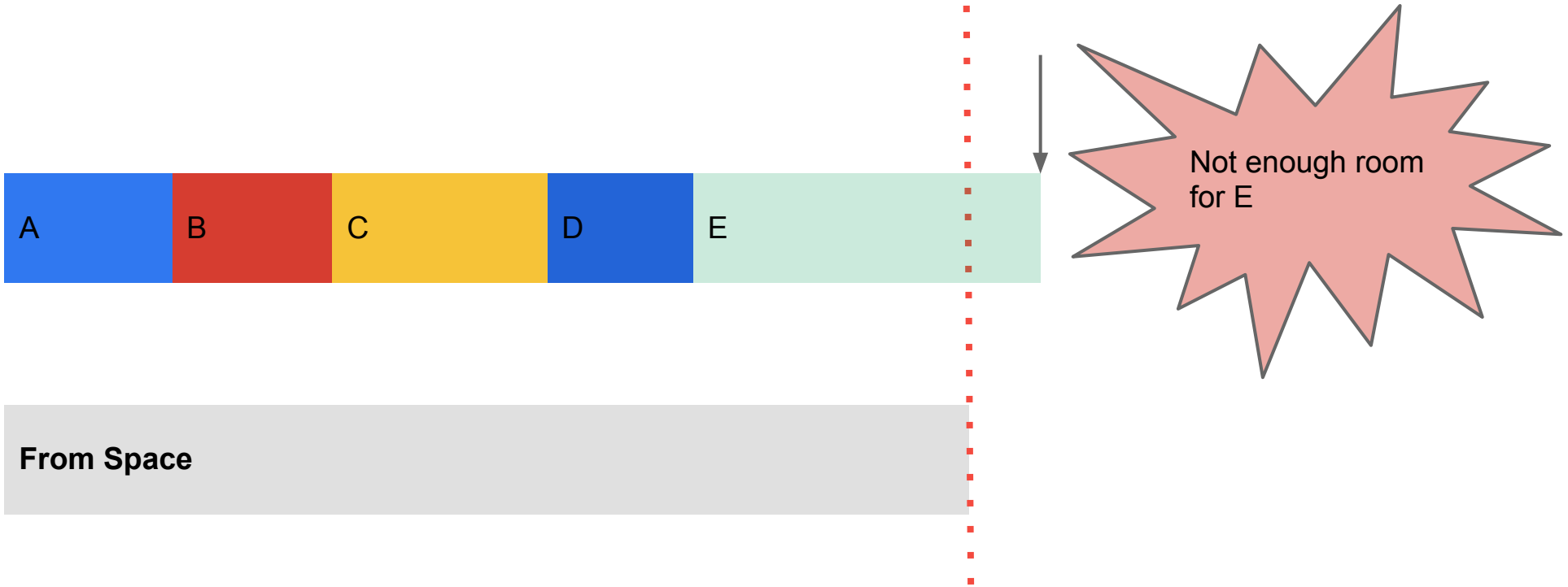
Young Generation In Action



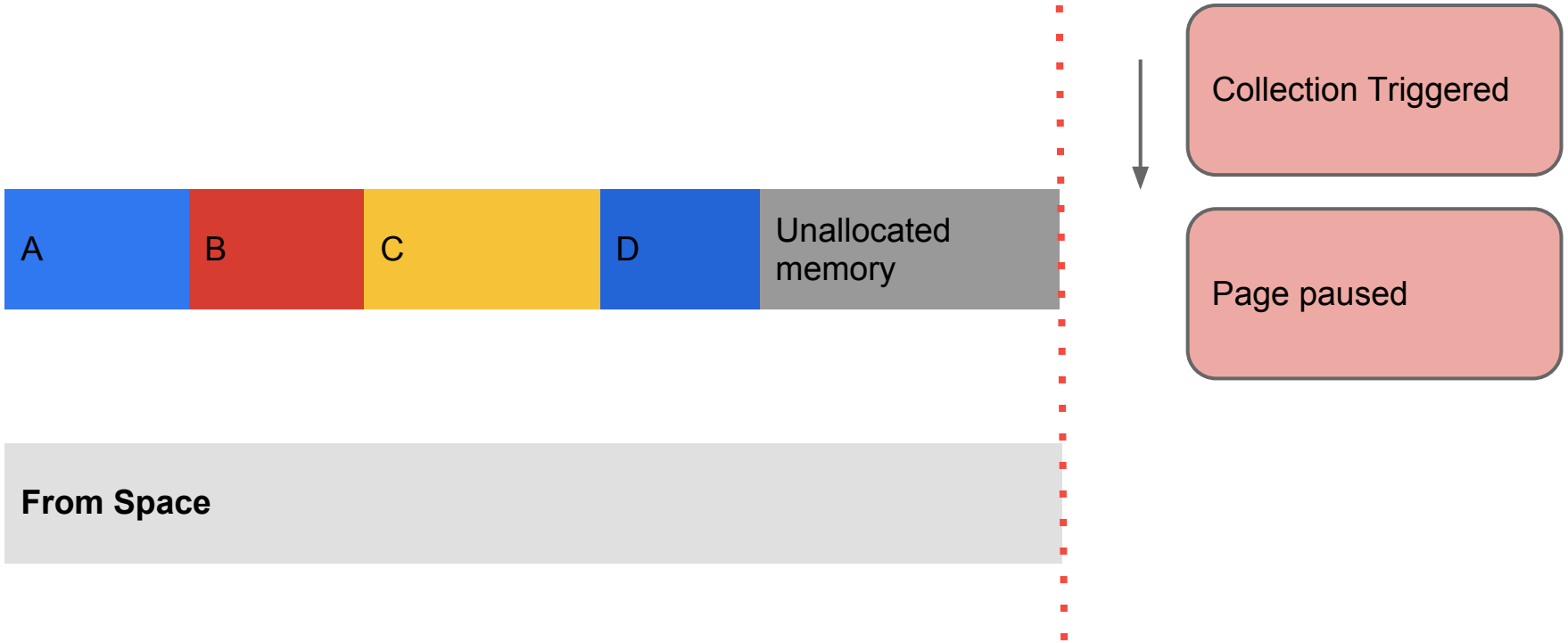
Young Generation In Action



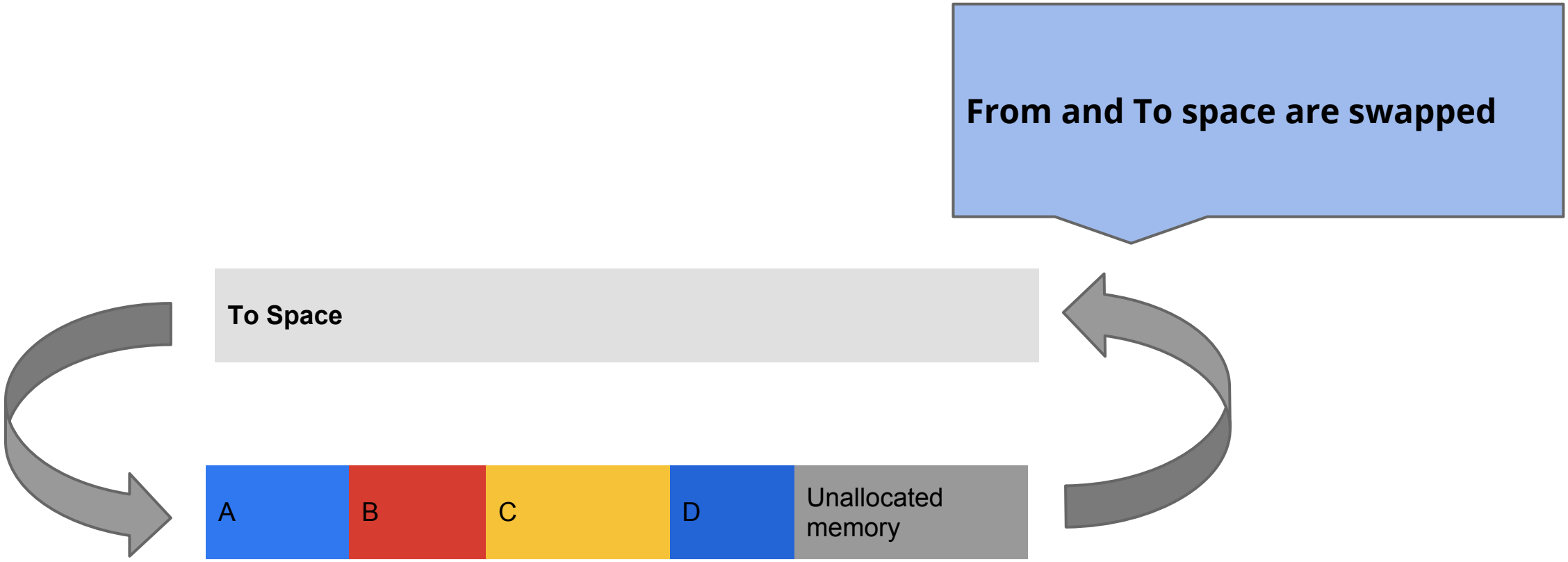
Young Generation In Action



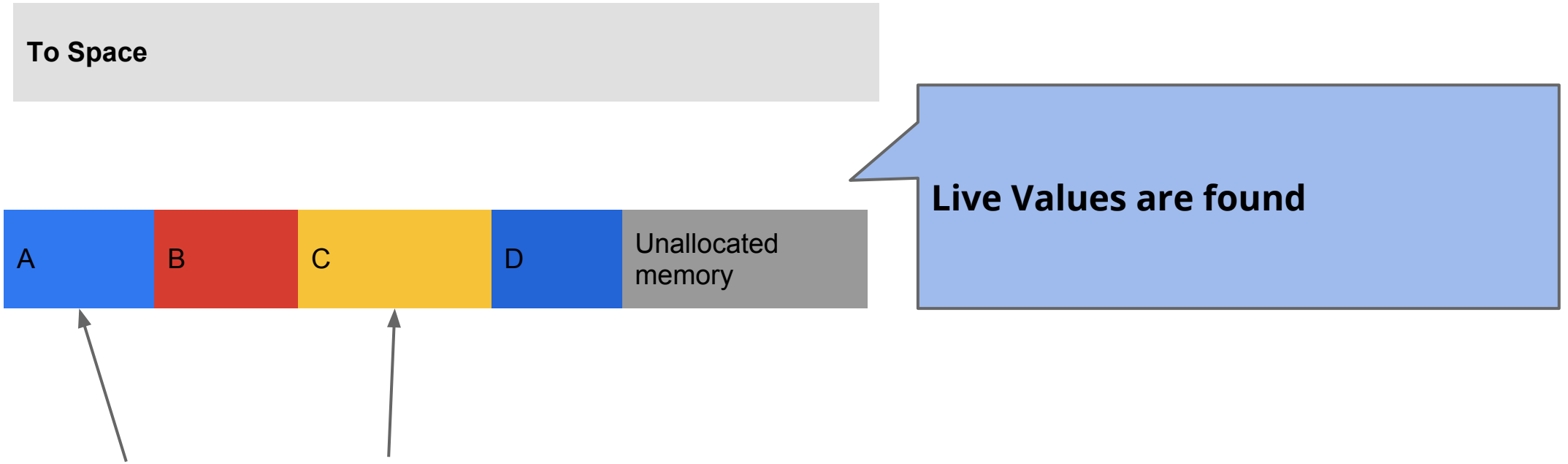
Young Generation In Action



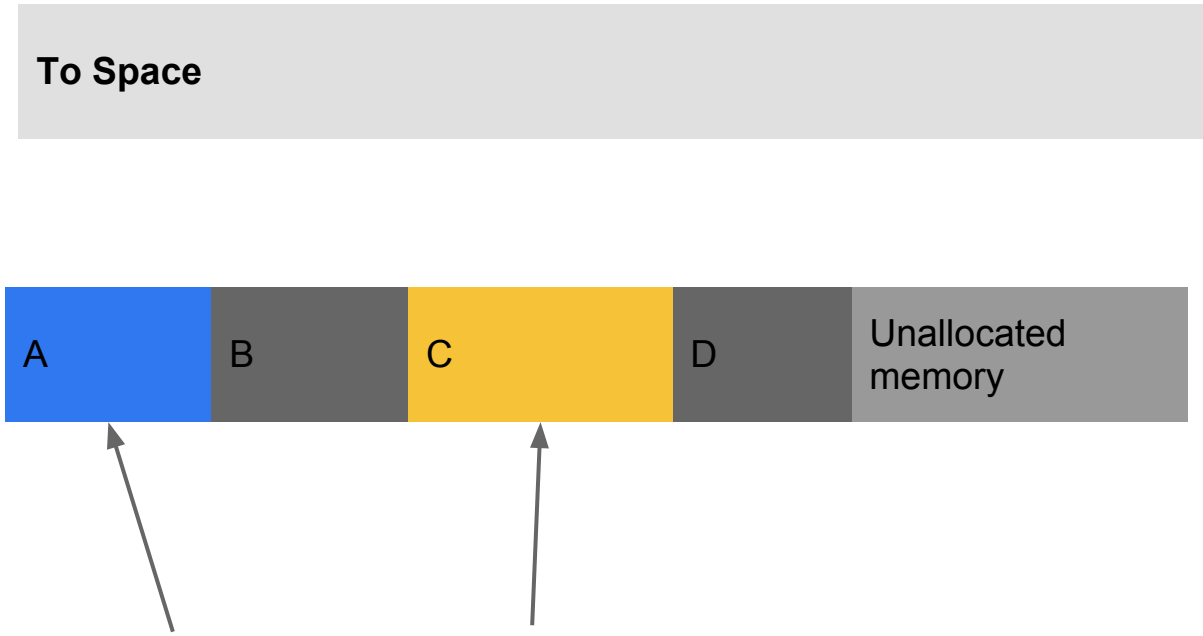
Young Generation In Action



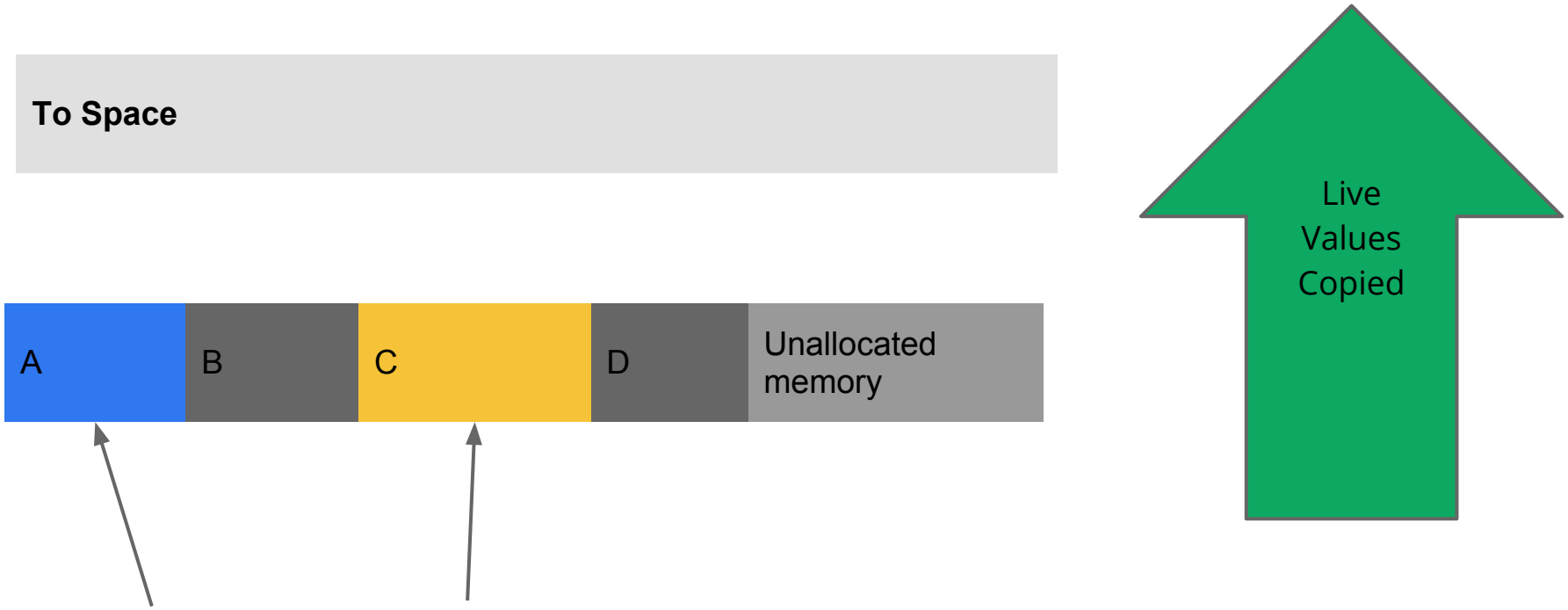
Young Generation In Action



Young Generation In Action



Young Generation In Action

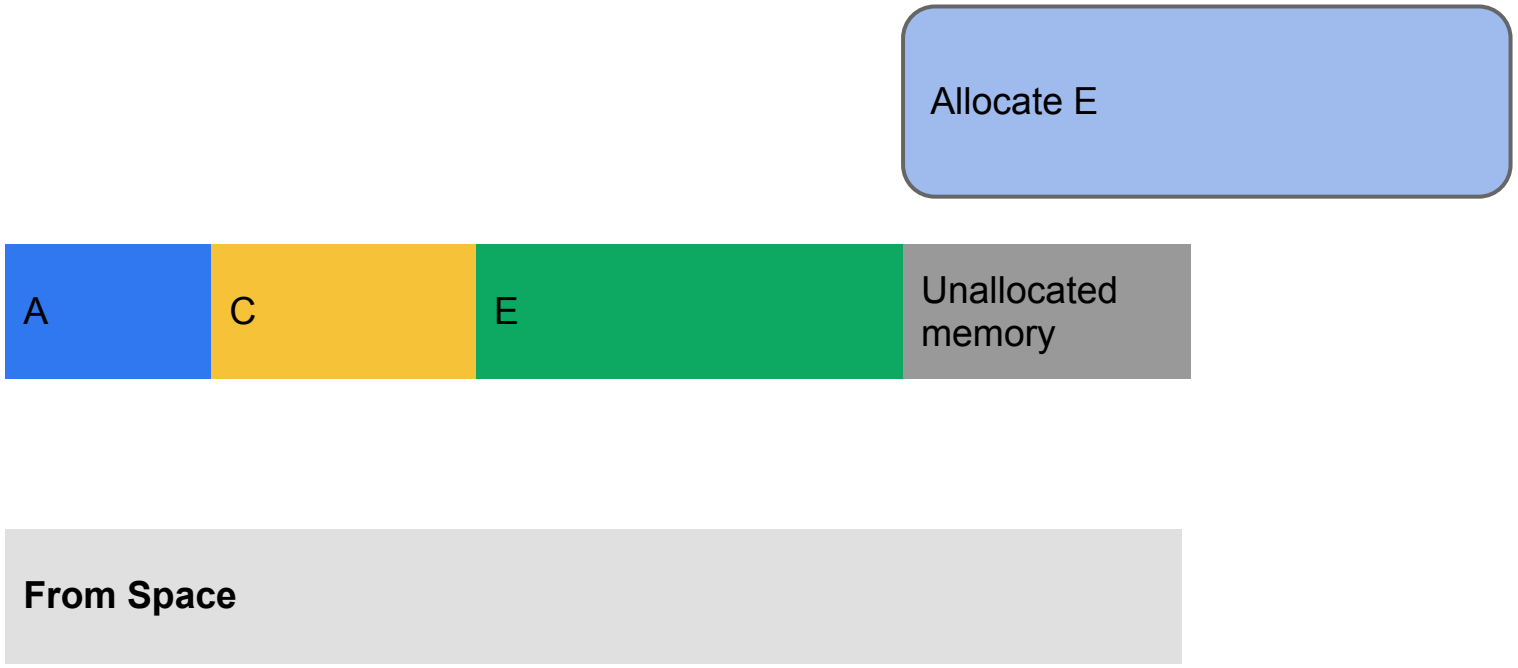


Young Generation In Action

Young Generation In Action

From Space

Young Generation In Action



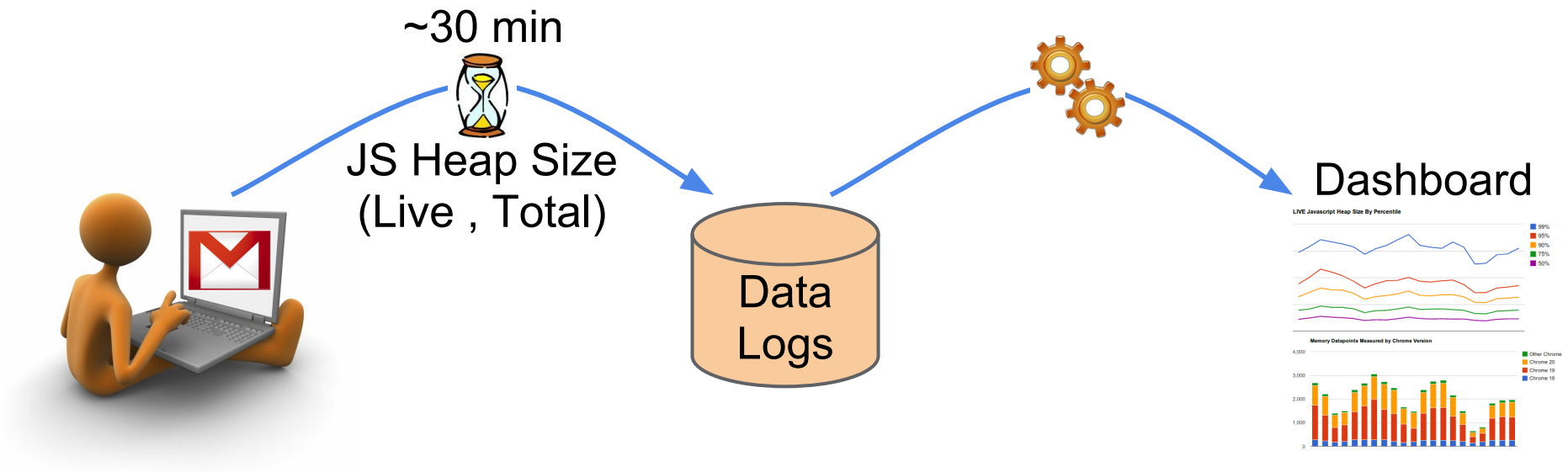
How does V8 manage memory?

- 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

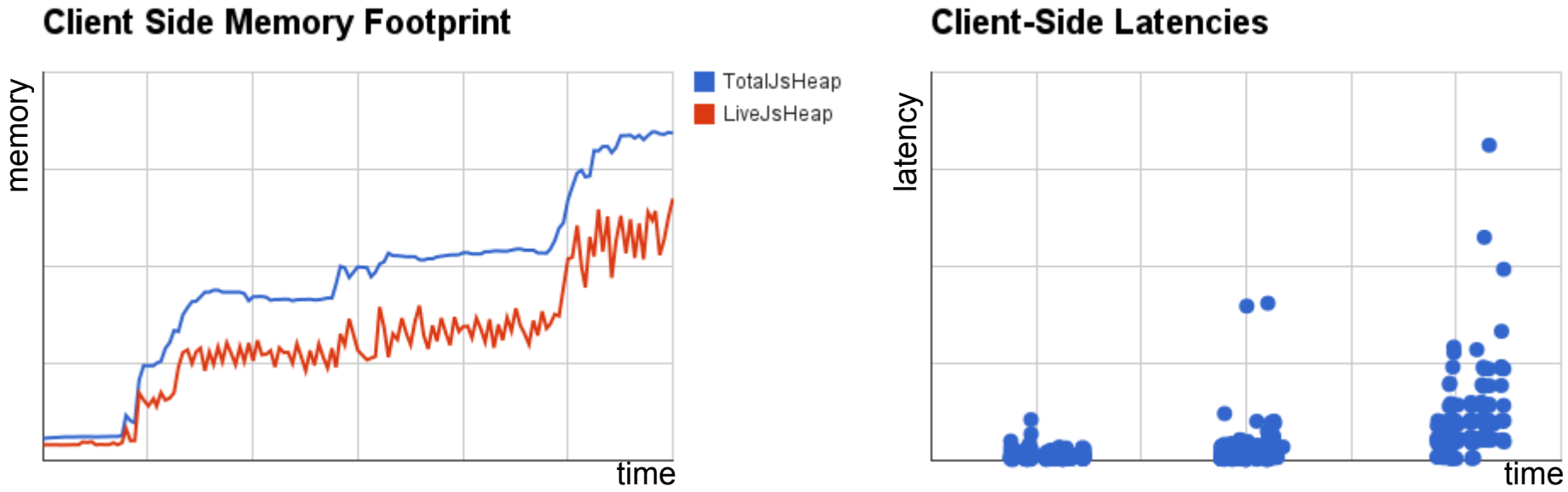
- Enabled by default in Chrome 22
- 3 values returned:
  - **jsHeapSizeLimit** - the amount of memory (in bytes) that the JavaScript heap is limited to
  - **totalJSHeapSize** - the amount of memory (in bytes) that the JavaScript heap has allocated, including free space
  - **usedJSHeapSize** - the amount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

Performance/Memory Tradeoff?

- Common belief:

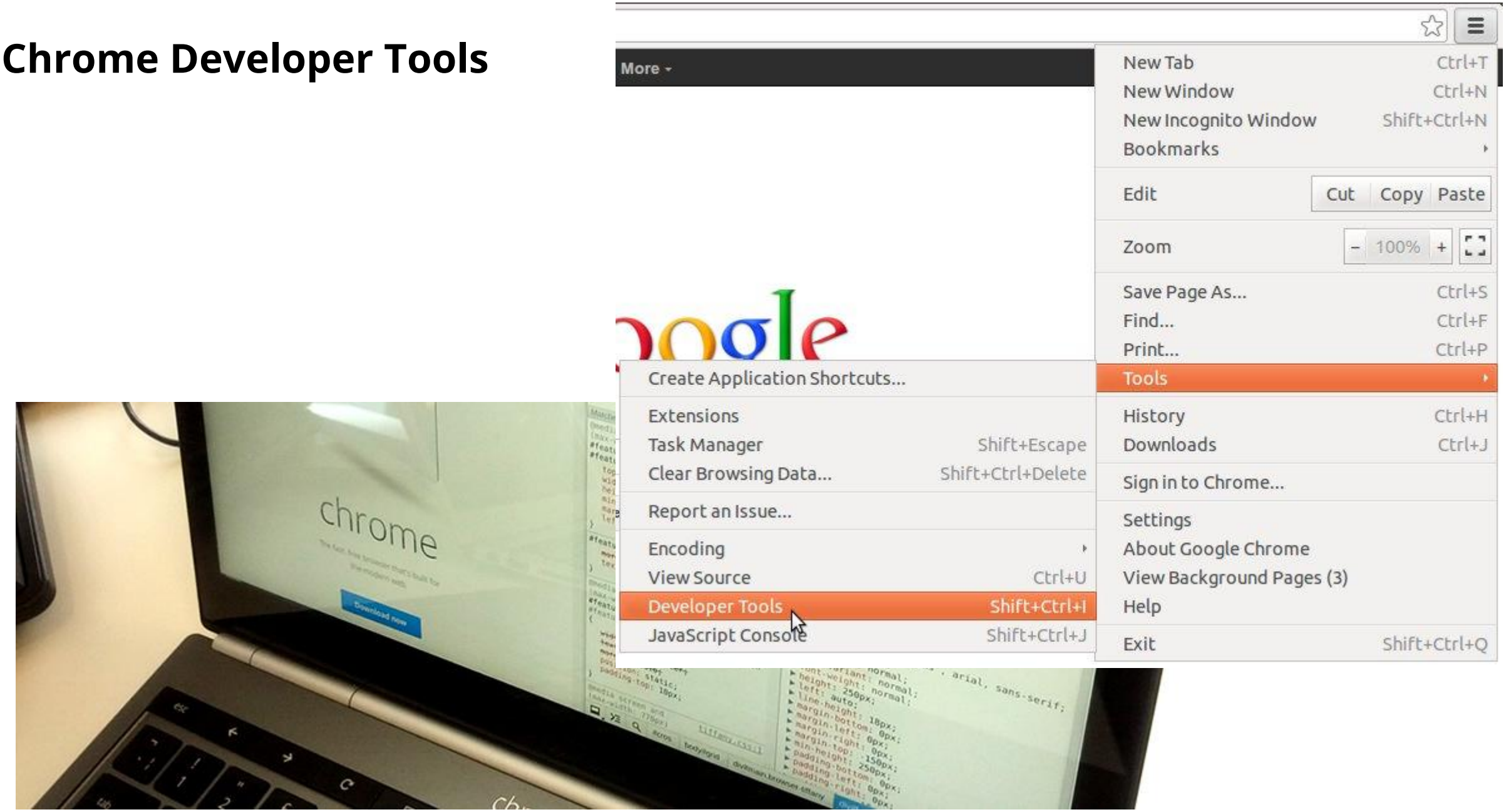
  More Memory == Better Performance

- Reality:

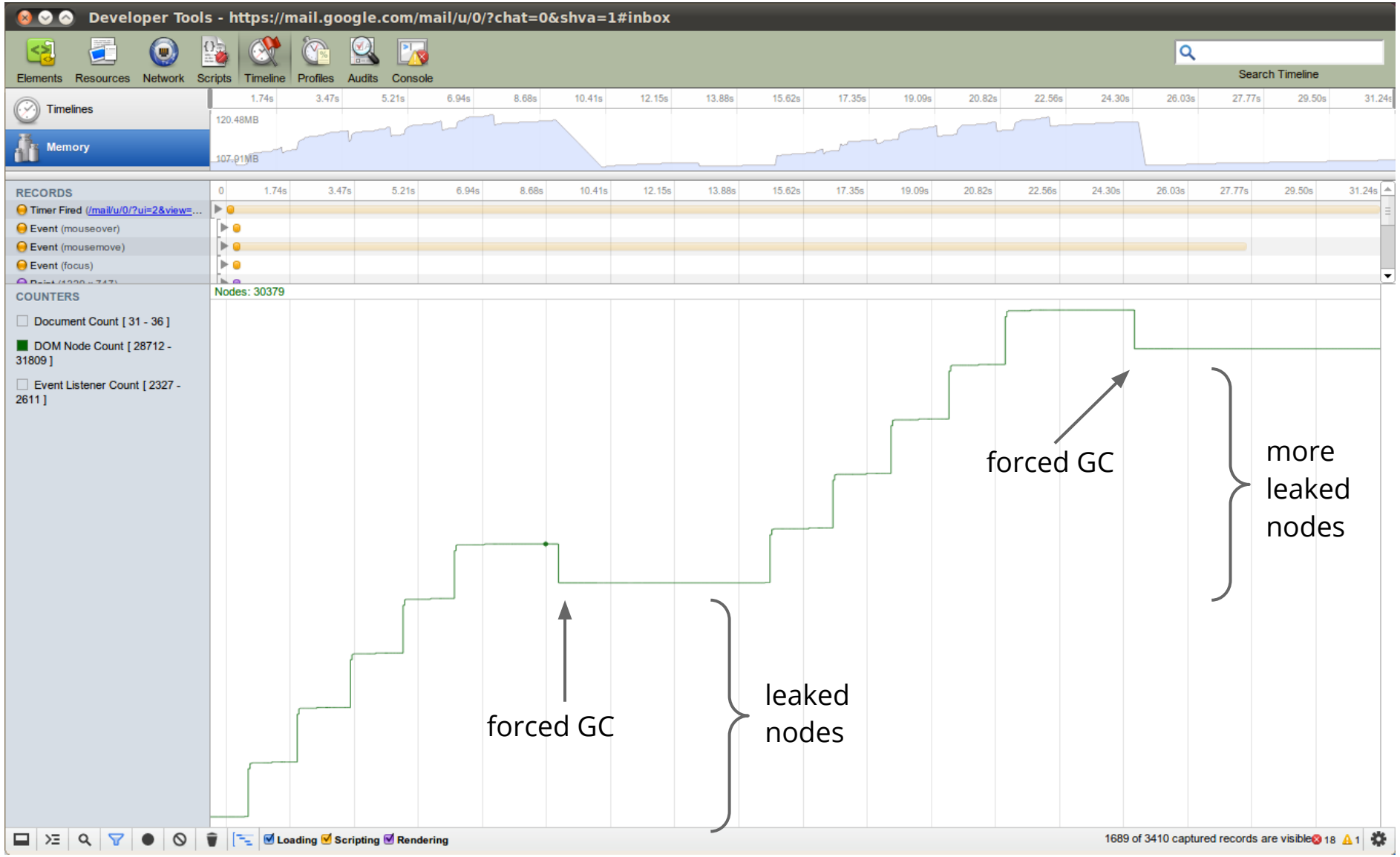


Increased memory footprint correlates with ***increased latencies and variance***

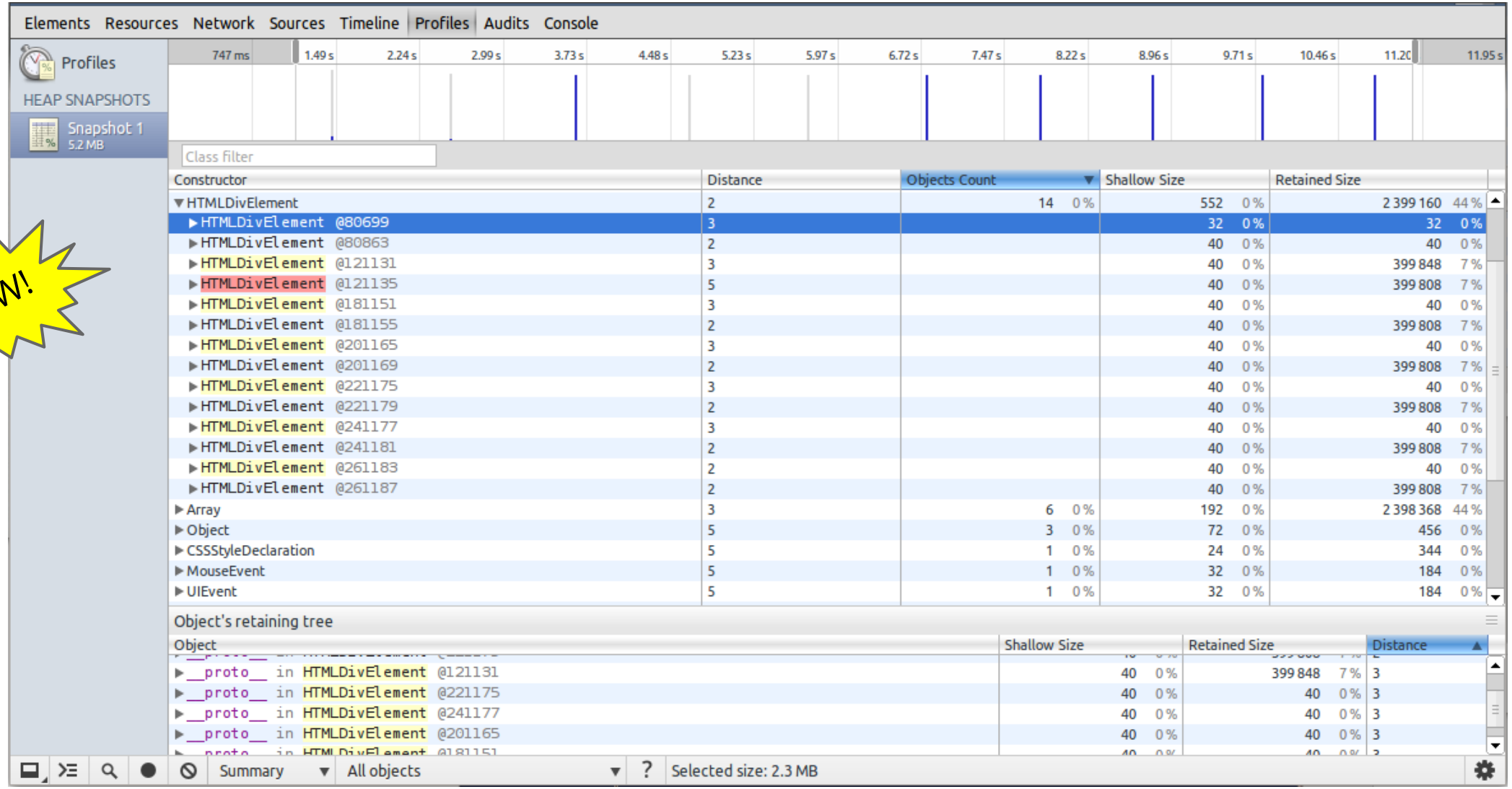
Chrome Developer Tools



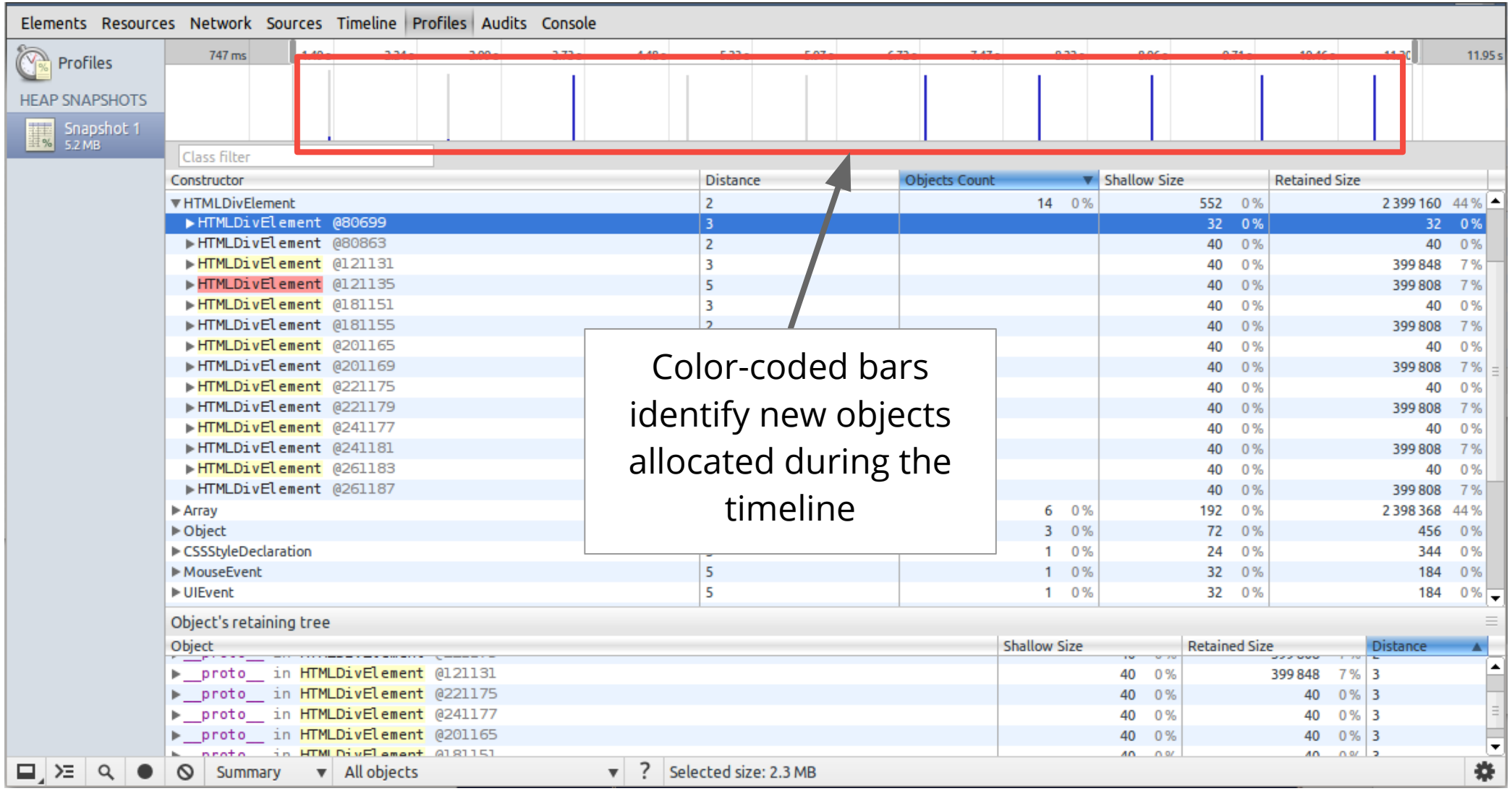
DevTools Memory Timeline



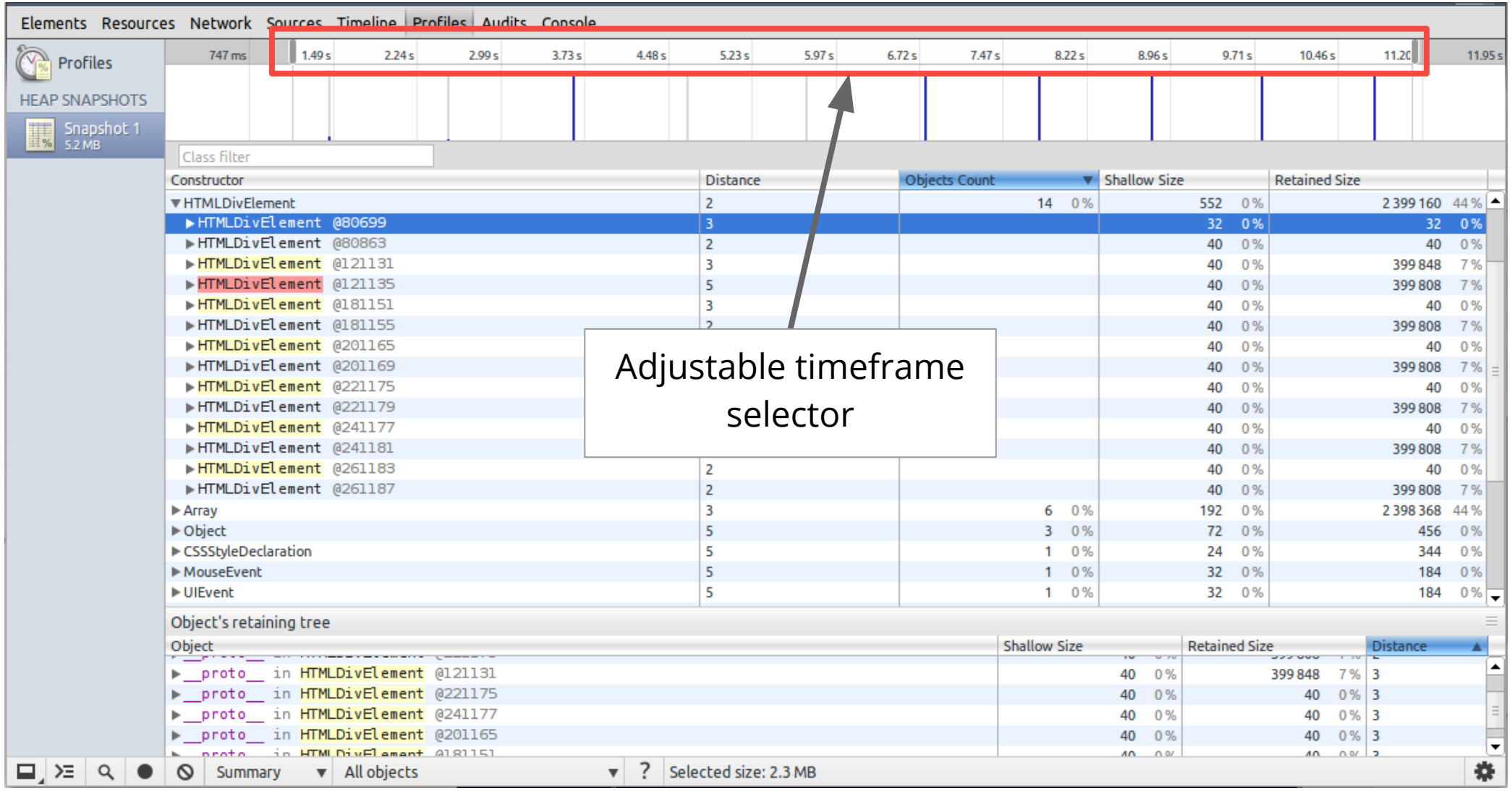
DevTools Object Tracker



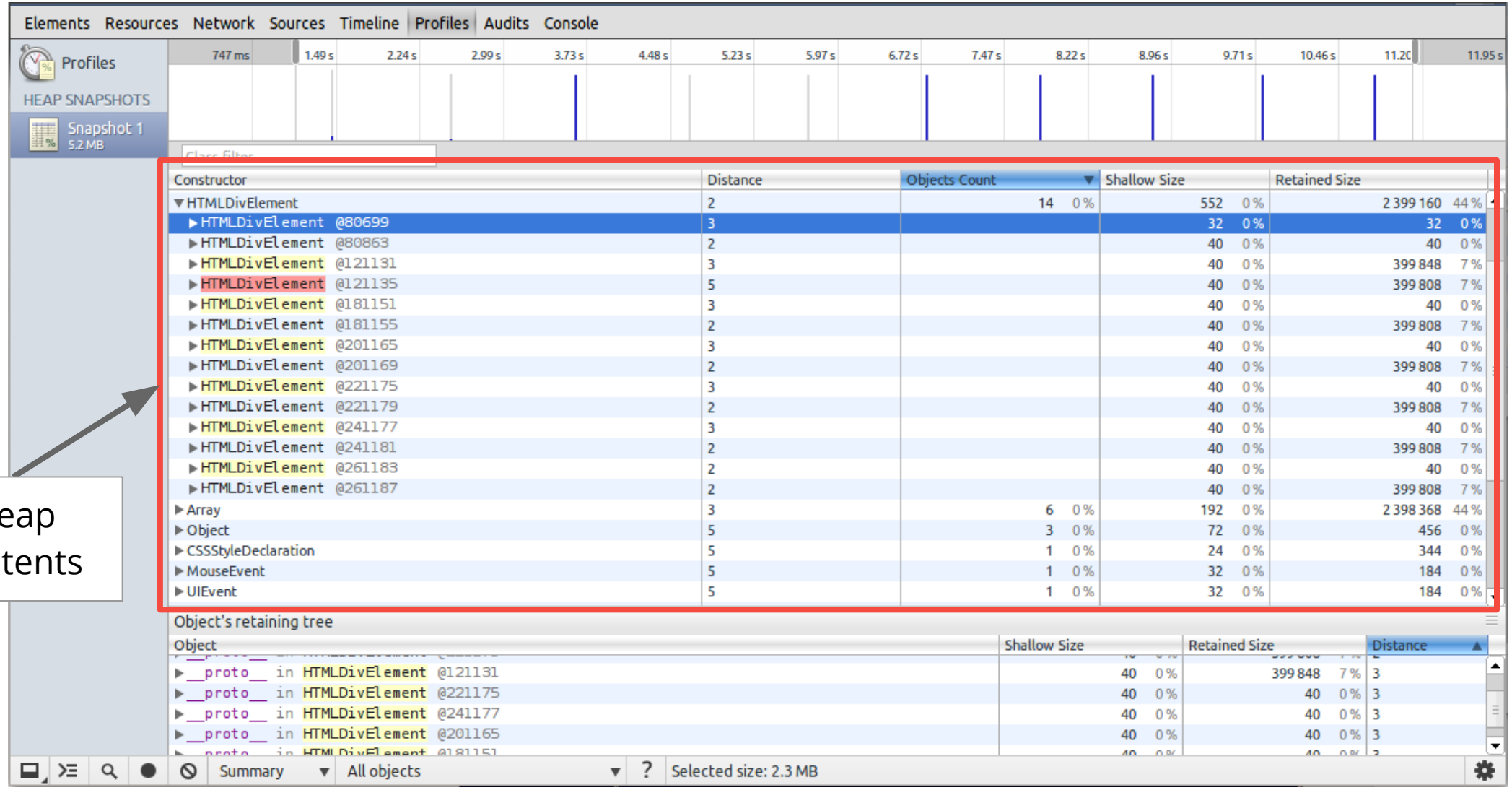
DevTools Object Tracker



DevTools Object Tracker



DevTools Object Tracker

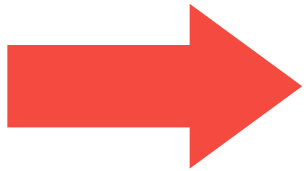


Heap contents

Demo

The Setup

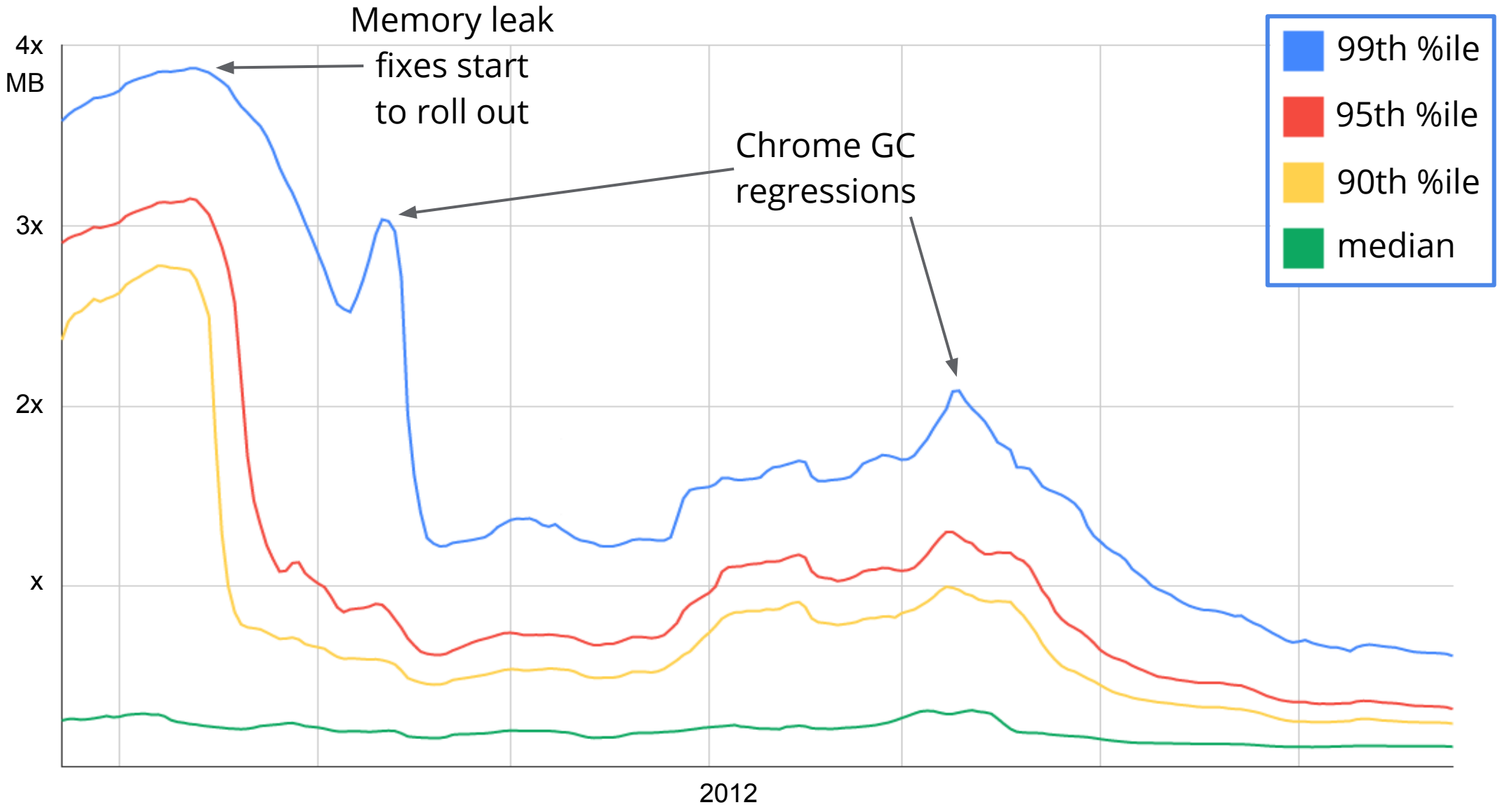
- A simple mail-like app
- Messages are cached for better performance
- Cache size: 5 messages



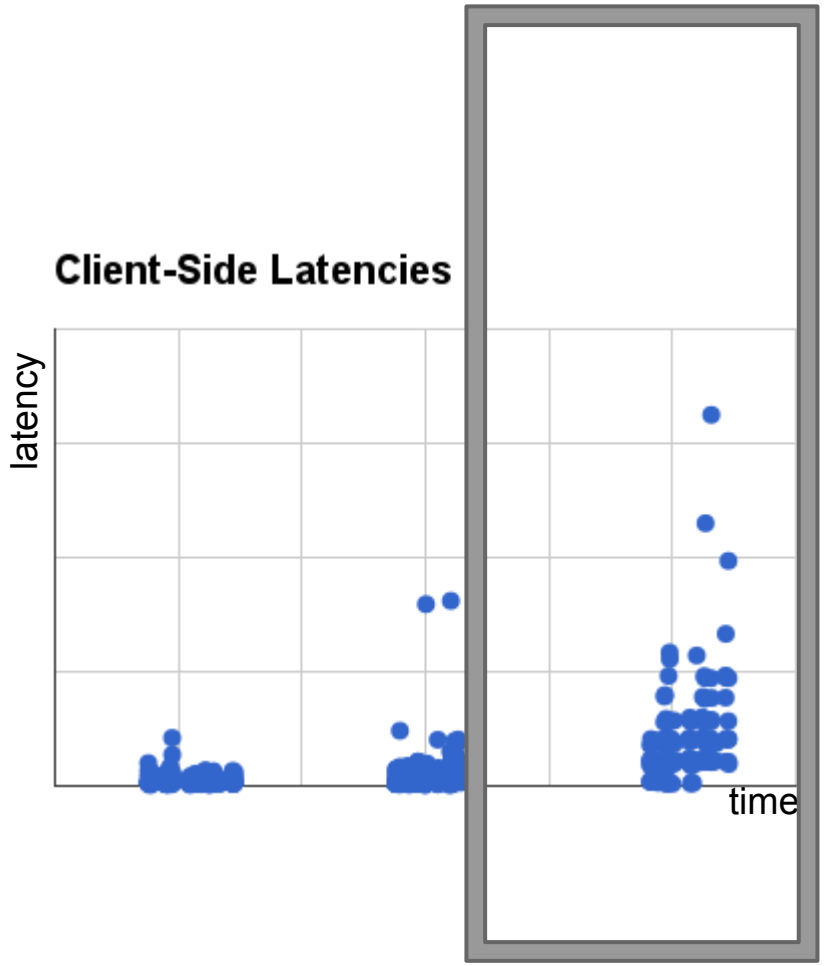
In theory, no more than 5 messages should be resident in memory at any given time...

Is it really worth it?

Where are we now?



Call to Action



Call to Action

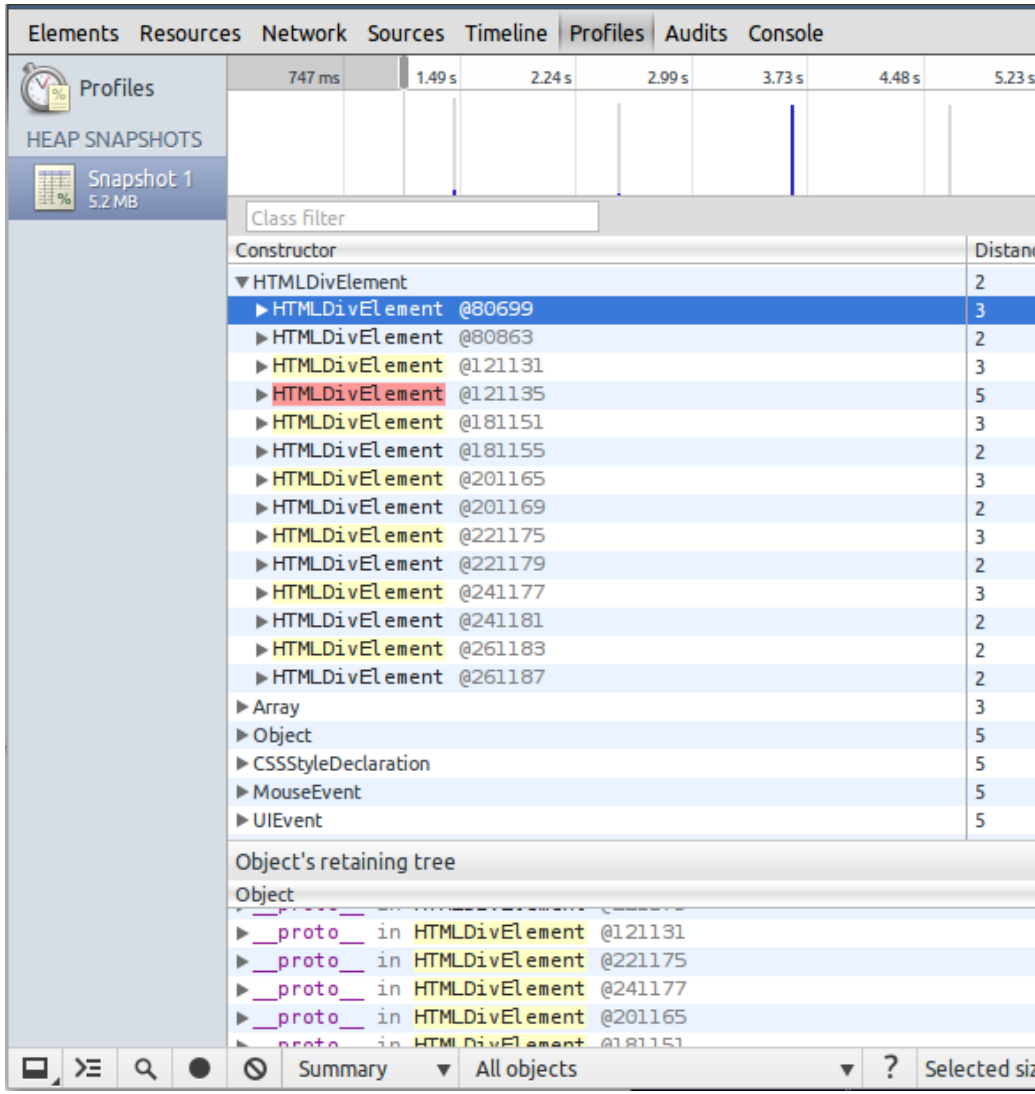
Ask yourself these questions:

- 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



<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