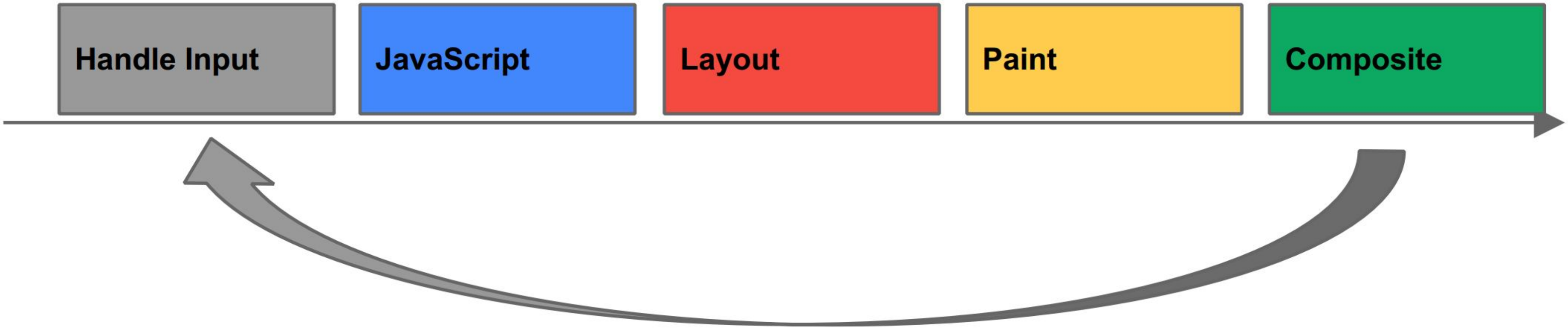
JavaScript Memory Management

Why do we need this?

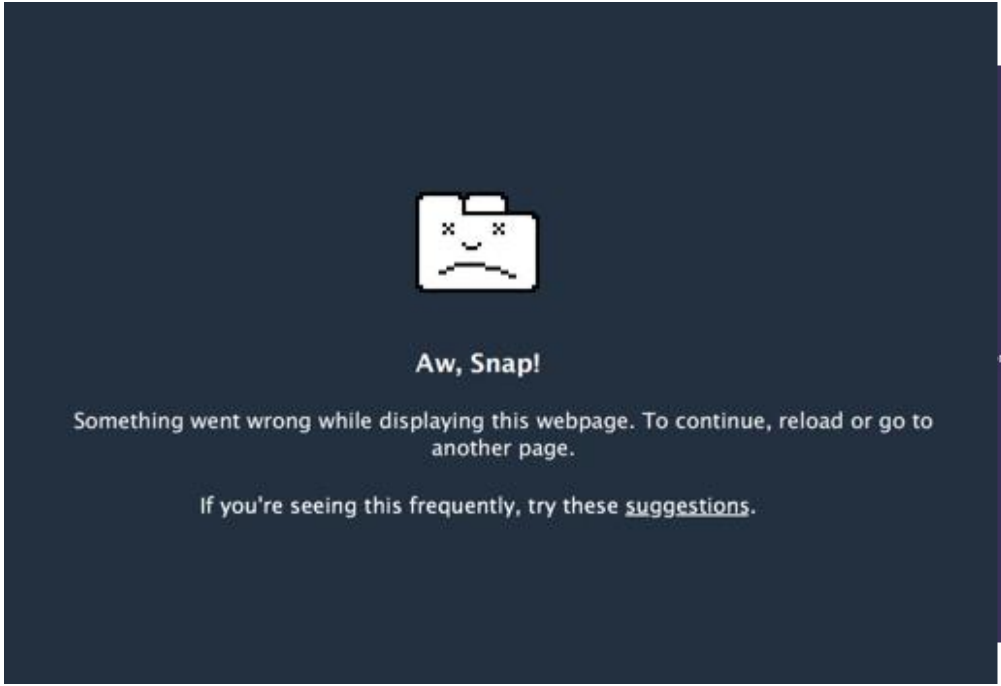
Performance vs. Memory

16ms to do everything.

Workload for a frame:



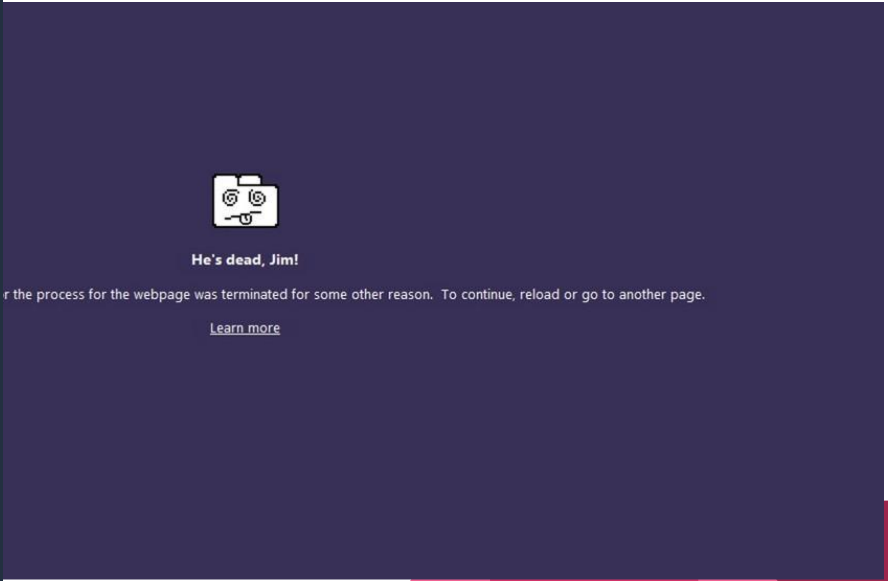
Blow memory & users will be sad.



Aw, Snap!

Something went wrong while displaying this webpage. To continue, reload or go to another page.

If you're seeing this frequently, try these suggestions.



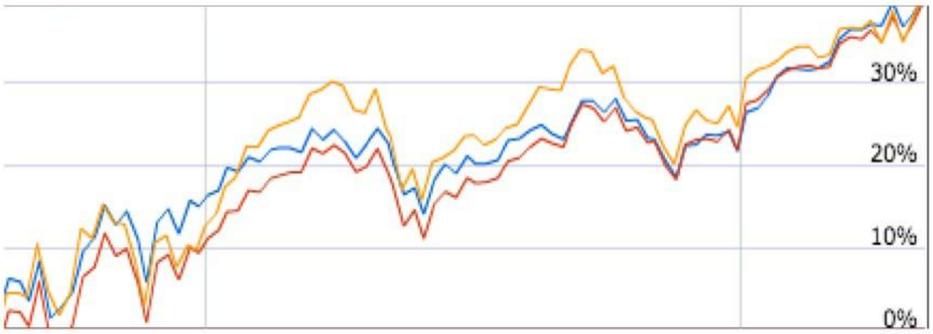
He's dead, Jim!

r the process for the webpage was terminated for some other reason. To continue, reload or go to another page.

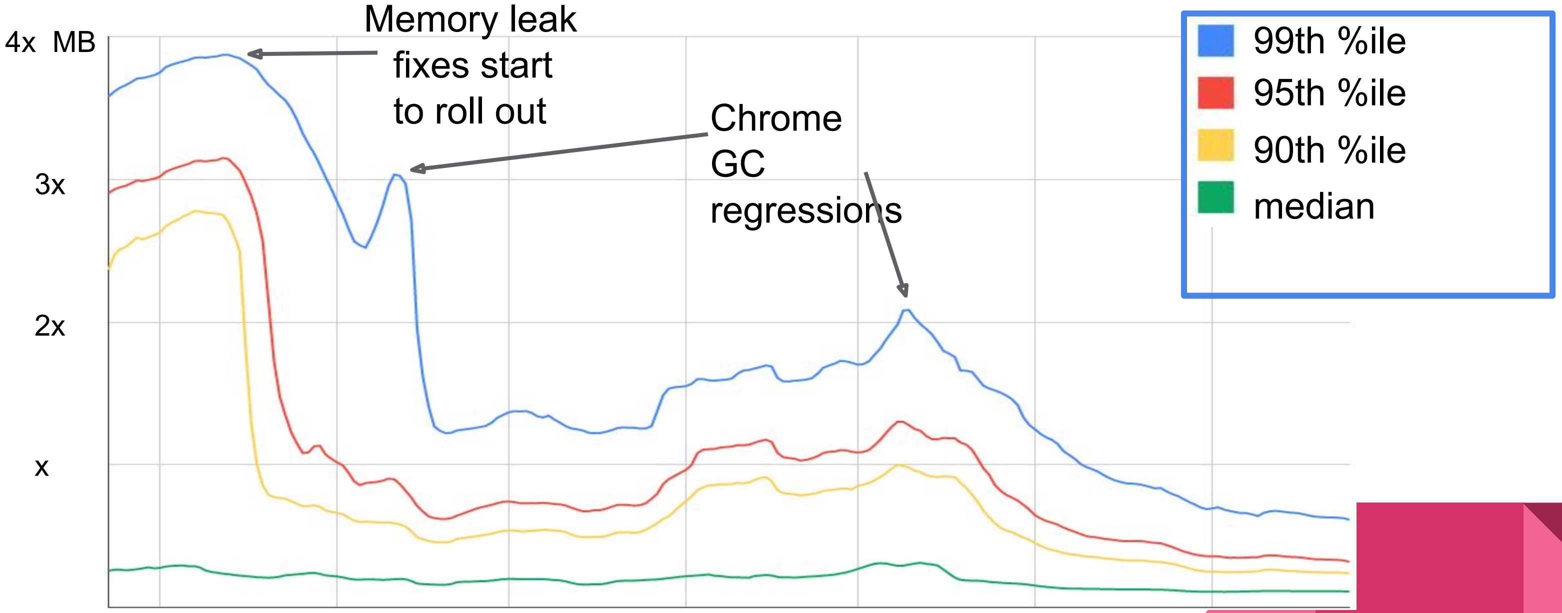
Learn more

Silky smooth apps.

Longer battery life
Smoother interactions
Apps can live longer



GMail’s memory usage (taken over a 10 month period)



Memory leak fixes start to roll out

Chrome GC regressions

> *"Through optimization, we reduced our memory footprint by 80% or more for power-users and 50% for average users."*

Loreena Lee,
GMail

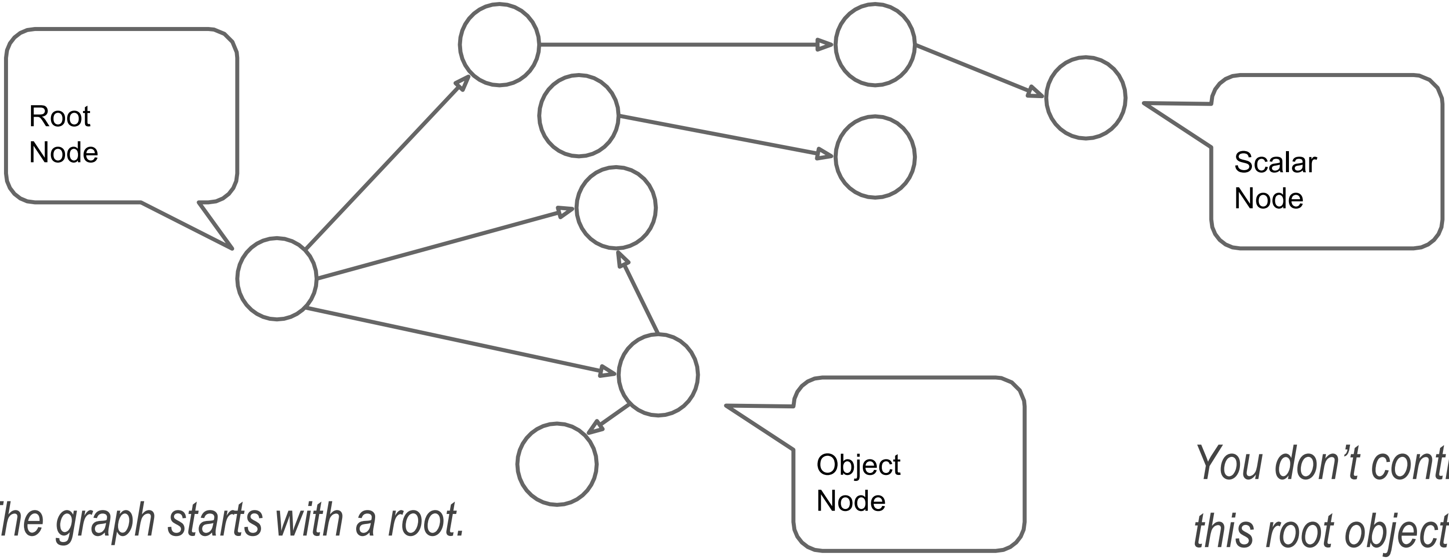
Basics of memory management

Core Concepts

1. Values are organized in a graph
2. Values have retaining path(s)
3. Values have retained size(s)

The value graph

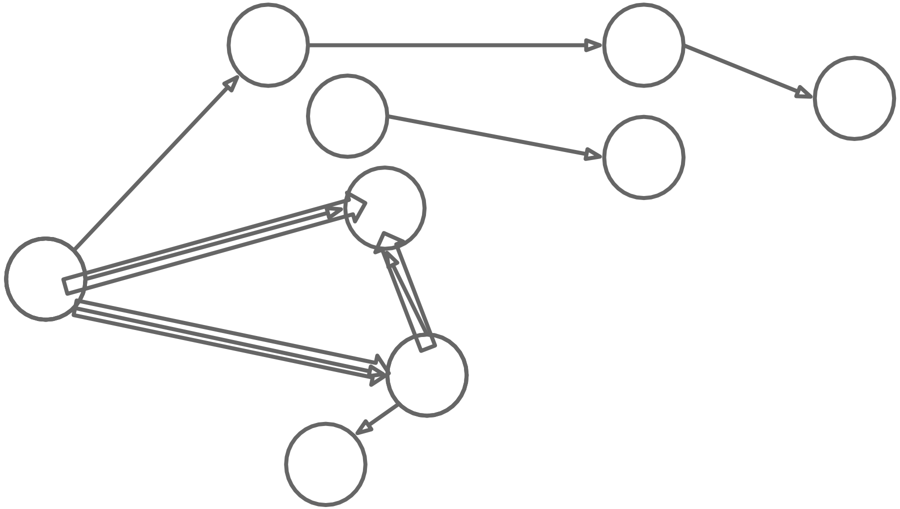
Root could be browser “window” or Global object of a Node module.



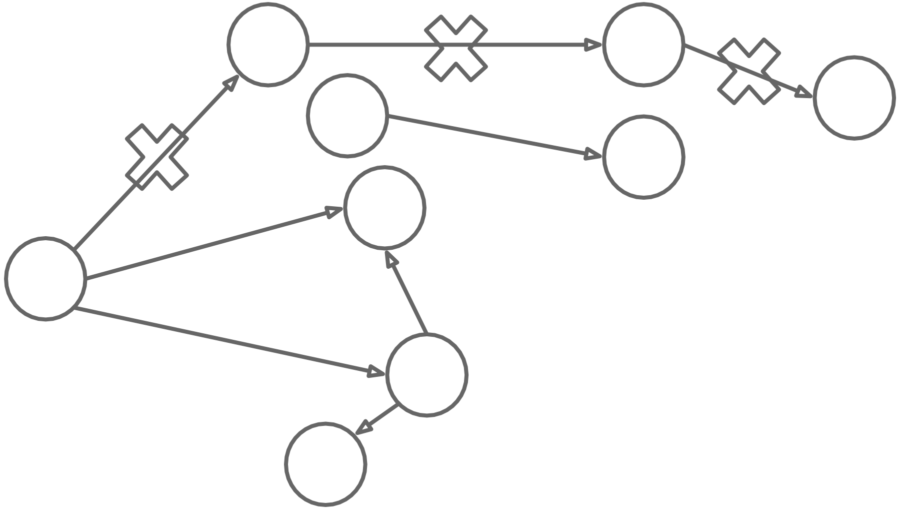
The graph starts with a root.

You don’t control how this root object is GC.

A value's retaining path(s)

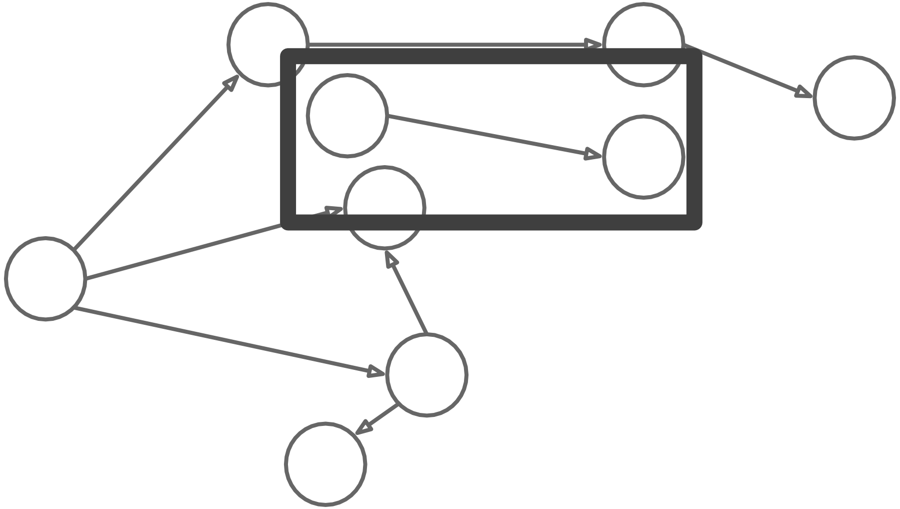


Removing a value from the graph



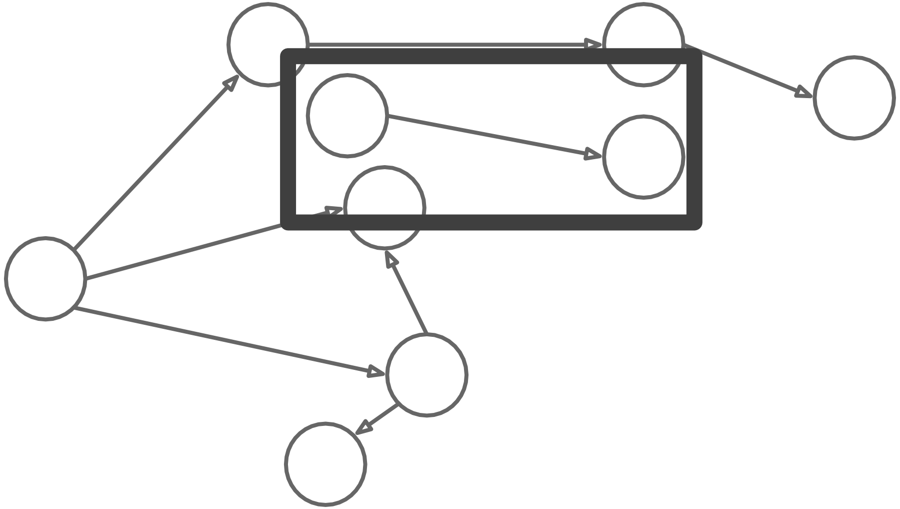
What is garbage?

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

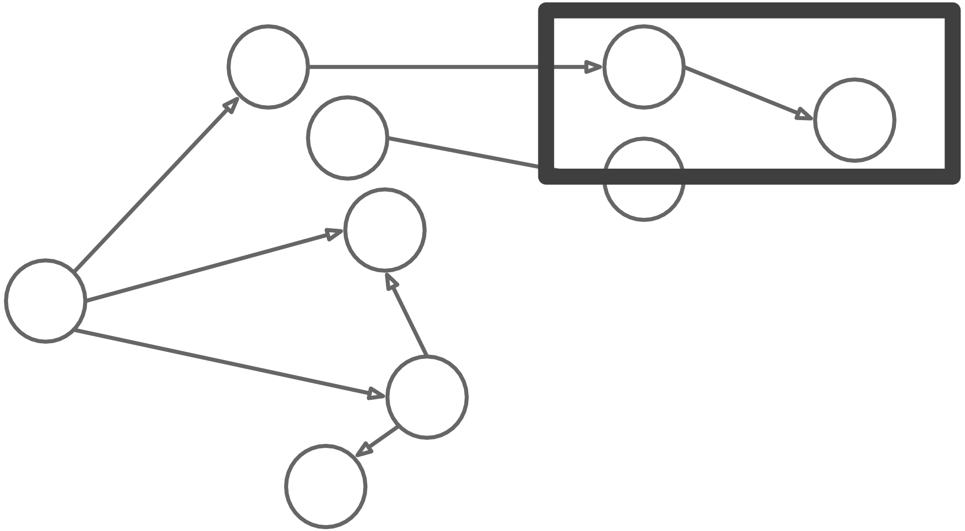


What is garbage collection?

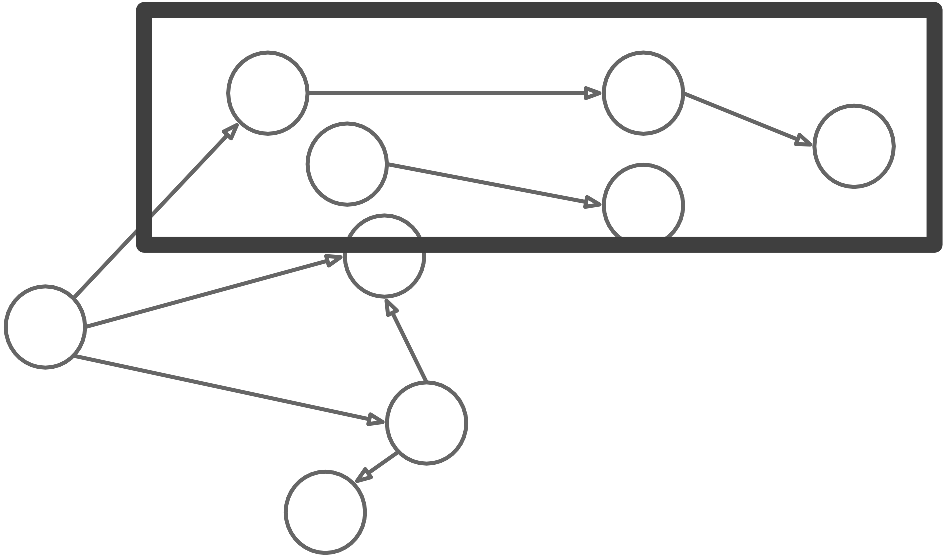
1. Find all live values
2. Return memory used by dead values to system



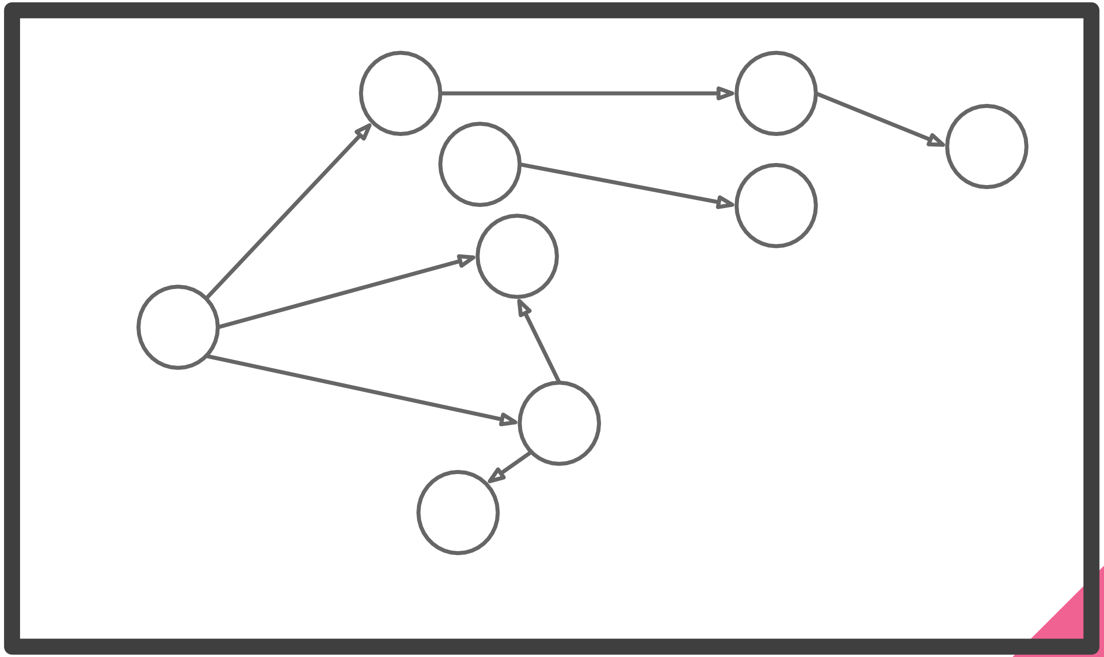
A value's retained size



A value's retained size



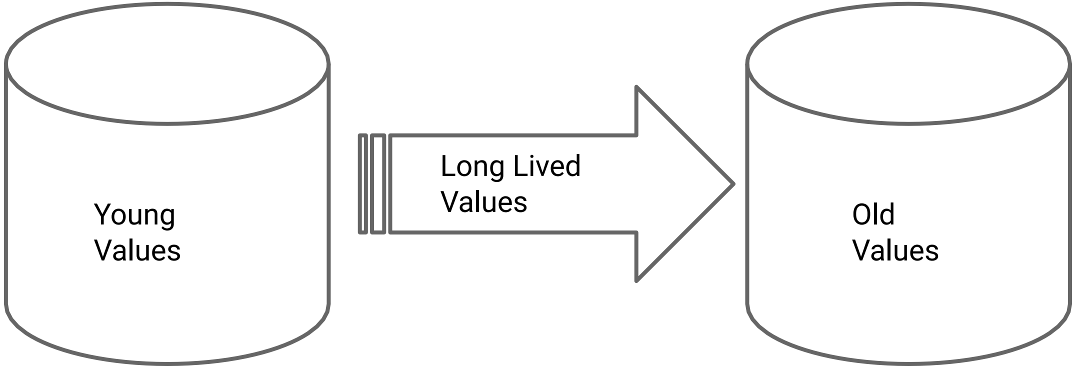
A value's retained size



V8 memory management

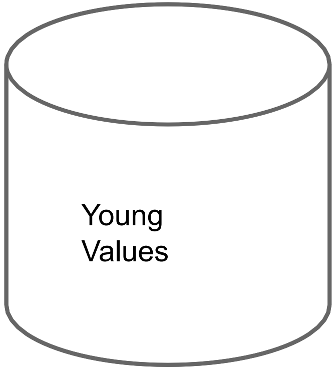
How does V8 manage memory?

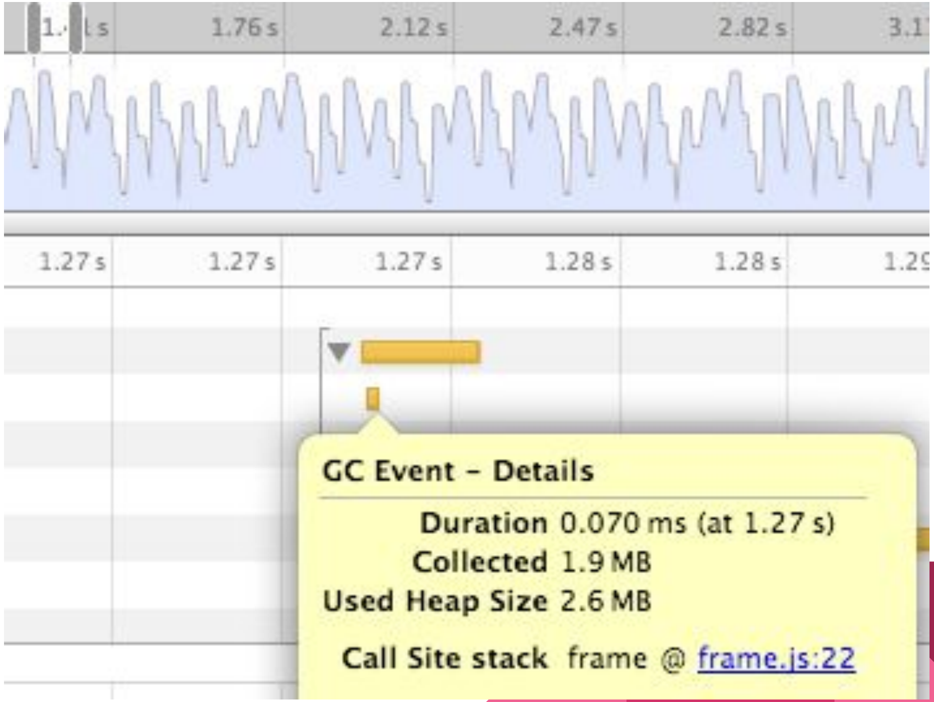
- Generational
  - Split values between young and old
  - Overtime young values promoted to old



How does V8 manage memory?

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





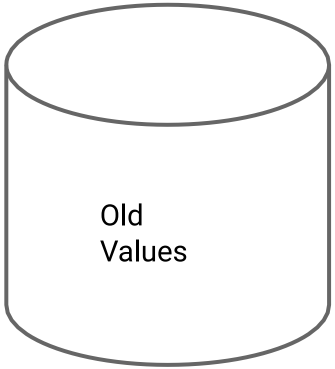
GC Event – Details

Duration 0.070 ms (at 1.27 s)
Collected 1.9 MB
Used Heap Size 2.6 MB

Call Site stack frame @ frame.js:22

How does V8 manage memory?

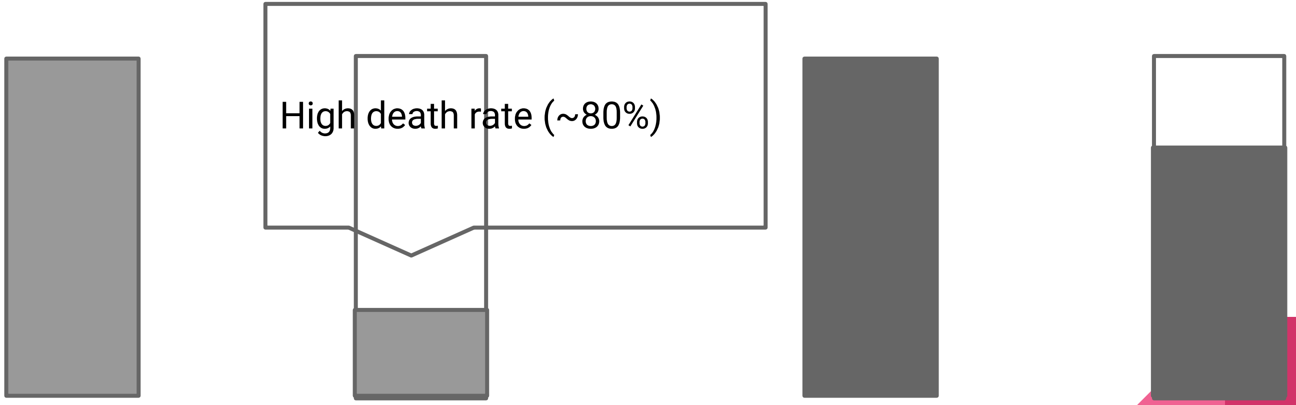
- Old Generation
  - Fast allocation
  - Slower collection
  - **Infrequently** collected



- 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

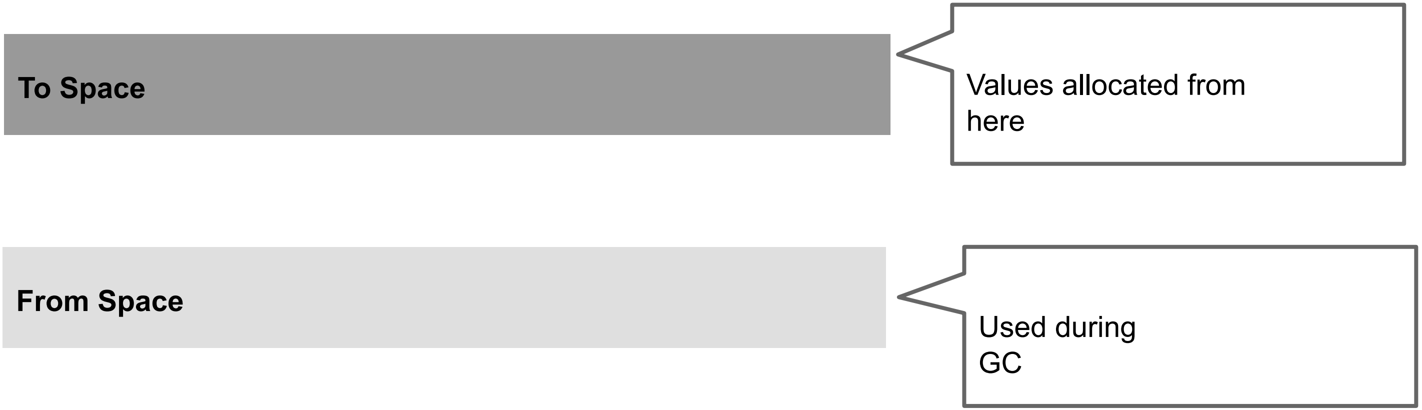


High death rate (~80%)

Young Generation Collection

Old Generation Collection

Young Generation In Action



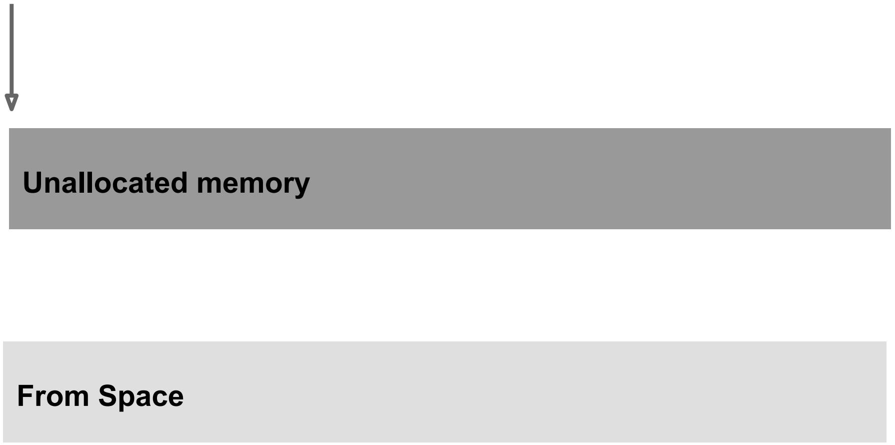
To Space

Values allocated from here

From Space

Used during GC

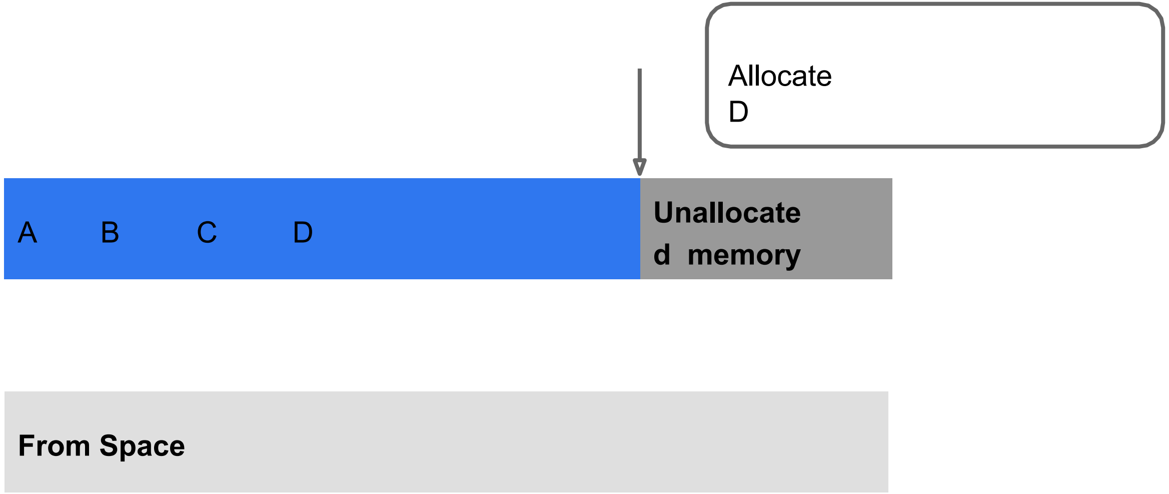
Young Generation In Action



Unallocated memory

From Space

Young Generation In Action



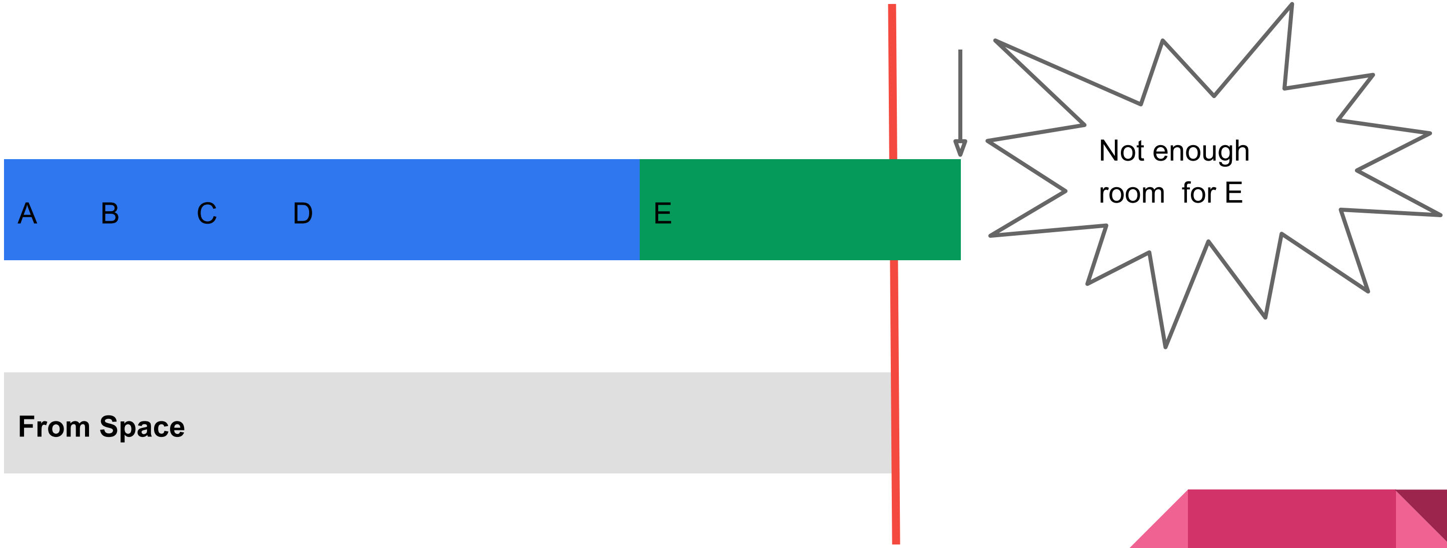
Allocate D

A B C D

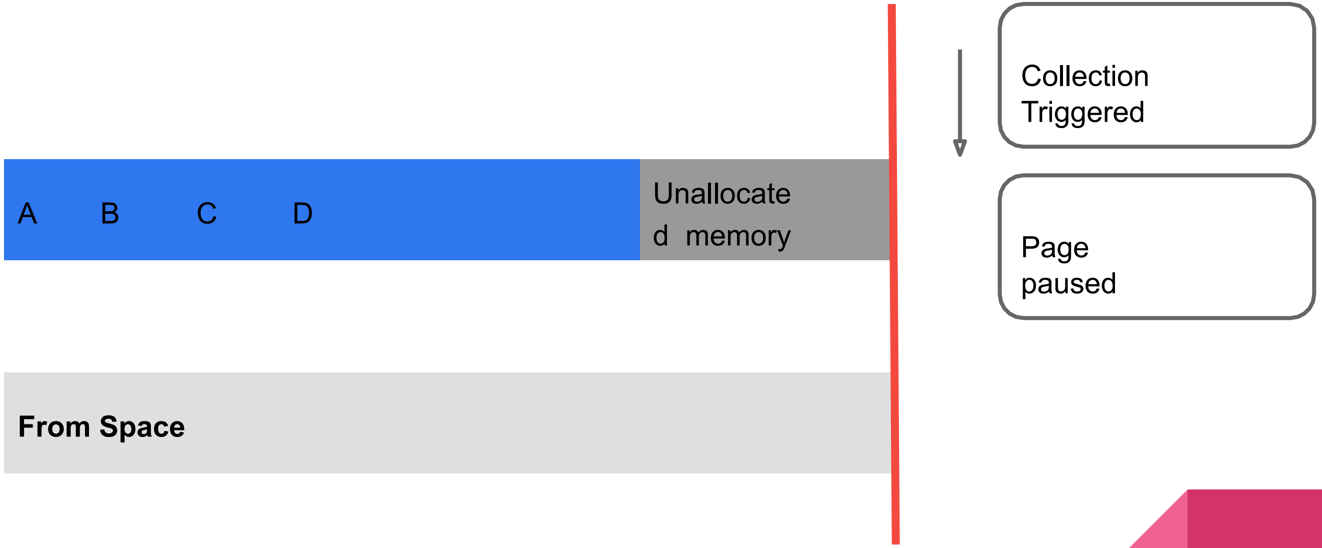
Unallocate d memory

From Space

Young Generation In Action

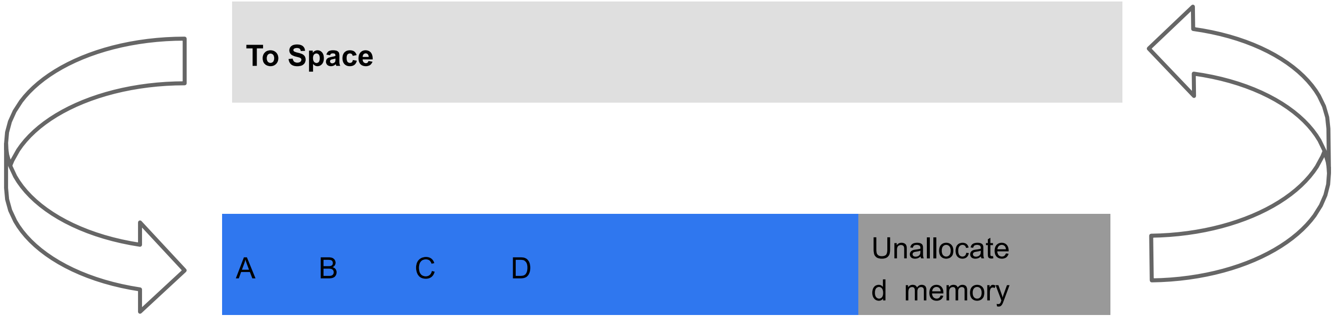


Young Generation In Action

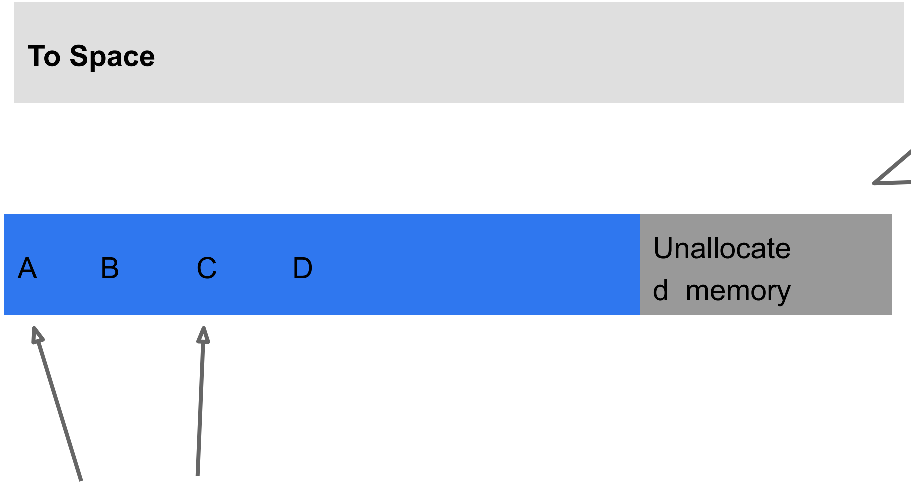


Young Generation In Action

From and To space are swapped



Young Generation In Action



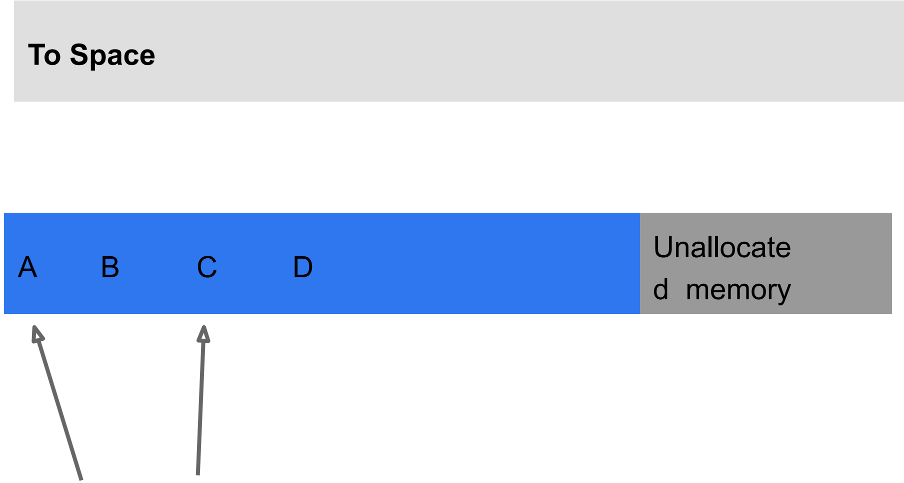
To Space

A B C D

Unallocate d memory

Live Values are found

Young Generation In Action

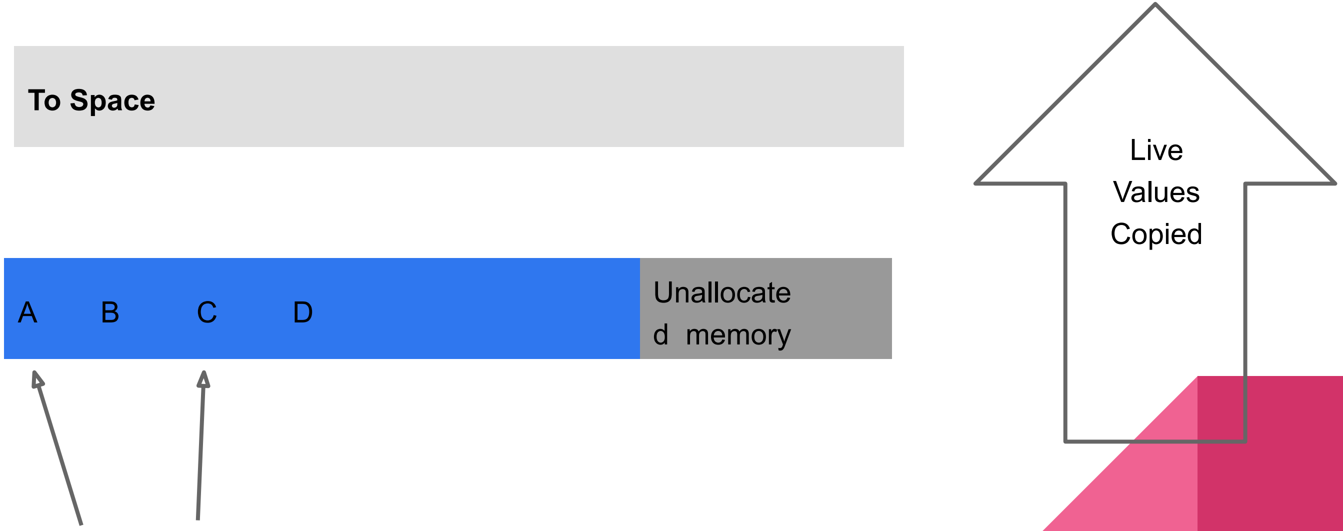


To Space

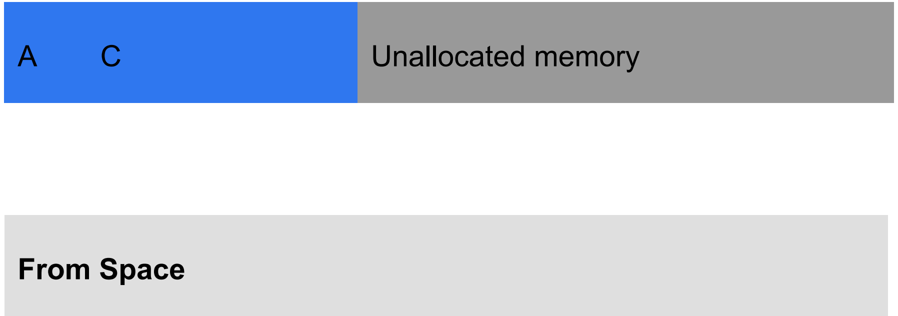
A B C D

Unallocate
d memory

Young Generation In Action



Young Generation In Action

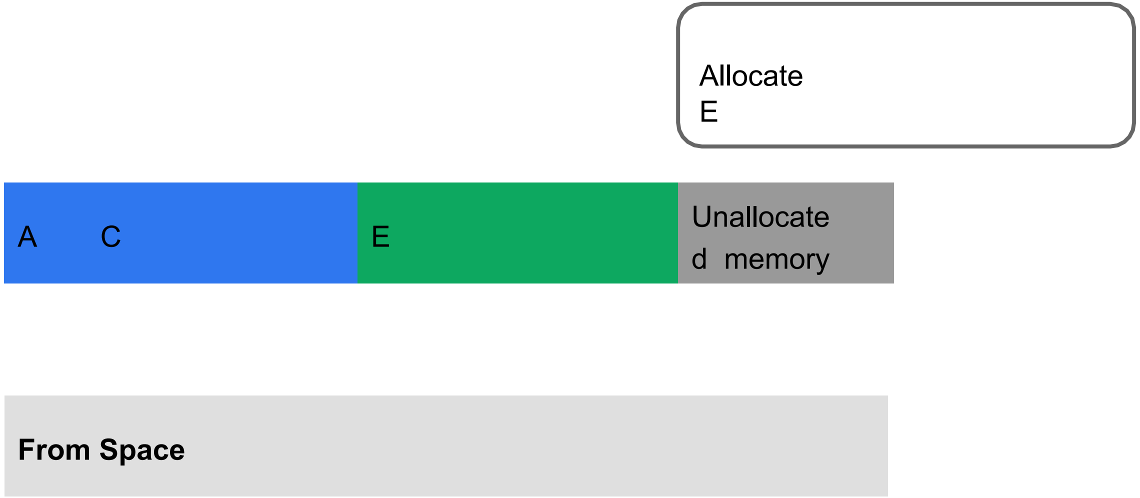


A C

Unallocated memory

From Space

Young Generation In Action



Young Generation In Action

- **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

Remember: Triggering a collection pauses your app.

What is a memory leak?

Leaks in JavaScript

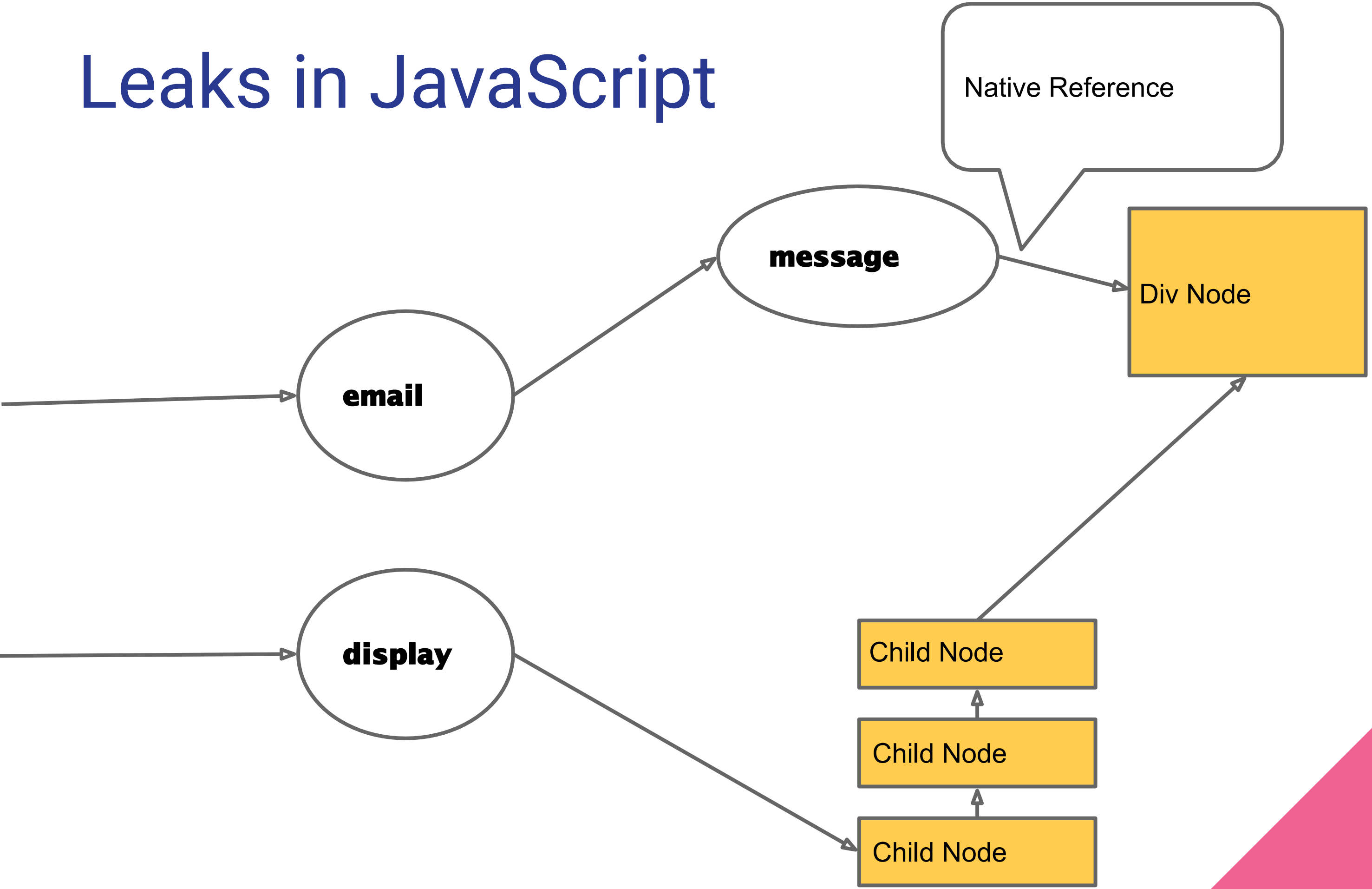
- A value that erroneously still has a retaining path
  - Programmer error

JavaScript

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

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

Leaks in JavaScript



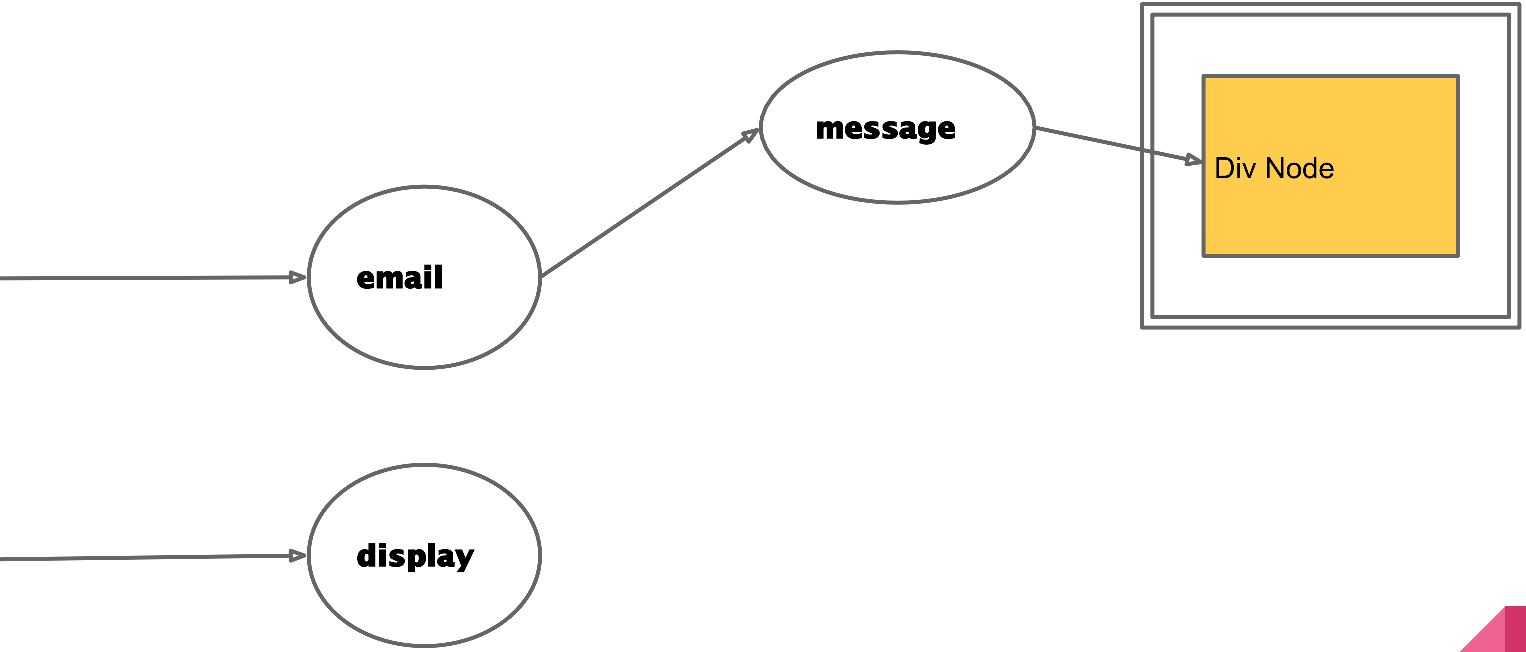
Leaks in JavaScript

JavaScript

```
// ...

display.removeAllChildren();
```

Leaks in JavaScript



Closures

Closures can be a source of memory leaks too. Understand what references are retained in the closure.

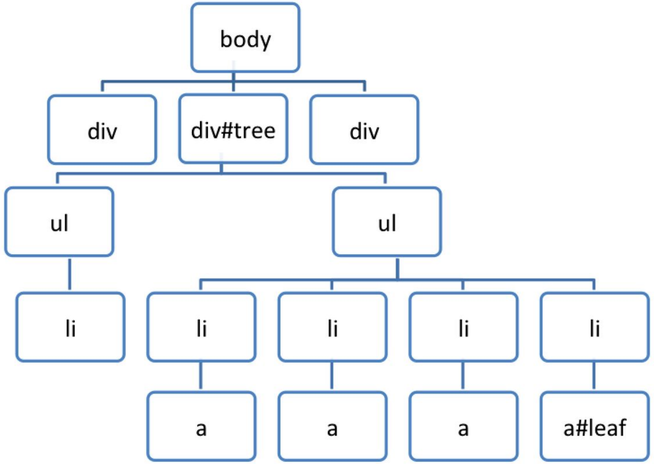
```
var theThing = null;

var replaceThing = function() {
    var originalThing = theThing;

    var unused = function() {
        if (originalThing)
            console.log("hi");
    };

    theThing = {
        longStr: new Array(1000000).join('*'),
        someMethod: function() {
            console.log(someMessage);
        }
    };
};
setInterval(replaceThing, 1000);
```

DOM Leaks.



```
var select = document.querySelector;

var treeRef = select("#tree");
var leafRef = select("#leaf");
var body = select("body");

body.removeChild(treeRef);
//#tree can't be GC yet due to treeRef

//let’s fix that:
treeRef = null;
//#tree can't be GC yet, due to
//indirect reference from leafRef

leafRef = null;
//NOW can be #tree GC
```

Timers

Timers are a common source of memory leaks.

Anything you’re repetitively doing in a timer should ensure it isn’t maintaining refs to DOM objects that could accumulate leaks if they can be GC’d.

```
for (var i = 0; i < 90000; i++) {
    var buggyObject = {
        callAgain: function() {
            var ref = this;
            var val = setTimeout(function() {
                ref.callAgain();
            }, 90000);
        }
    }

    buggyObject.callAgain();
    buggyObject = null;
```

ES6 WeakMaps

```
var Person = (function() {

  var privateData = {}, // strong reference  privateId = 0;

    function Person(name) {
      Object.defineProperty(this, "_id", {
        value: privateId++
      });

      privateData[this._id] = {
        name: name
      };
    }

  Person.prototype.getName = function() {
    return privateData[this._id].name;
  };

  return Person;
}());
```

```
var Person = (function() {
  var privateData = new WeakMap();

  function Person(name) {
    privateData.set(this, {
      name: name
    });
  }
  Person.prototype.getName = function() {
    return privateData.get(this).name;
  };
  return Person;
}());
```

V8’s Hidden Classes

Take care with the delete keyword
“o” becomes a SLOW object.

It’s better to set “o” to “null”.

Only when the last reference to an object is removed does that object get
eligible for collection.

```
var o = {   x: “y”  };
delete o.x;
o.x; //      undefined

var o = {   x: “y”  };
o = null;
o.x; //      TypeError
```

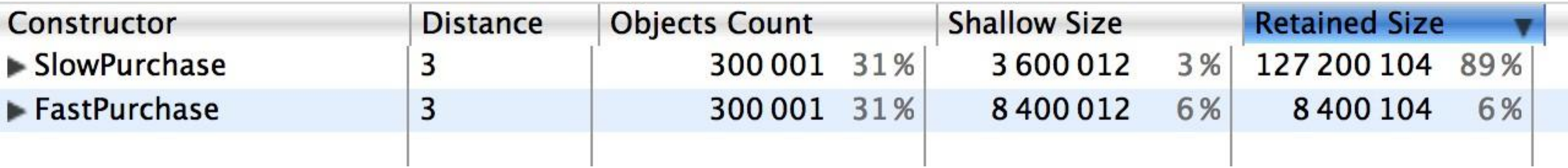
Fast object

```
function FastPurchase(units, price) {
    this.units = units;
    this.price = price;
    this.total = 0;
    this.x = 1;
}
var fast = new FastPurchase(
```

Slow object

```
function SlowPurchase(units, price) {
    this.units = units;
    this.price = price;
    this.total = 0;
    this.x = 1;
}
var slow = new SlowPurchase(3, 25);
//x property is useless
//so I delete it
delete slow.x;
```

Reality: “Slow” uses 15 times more memory



Constructor	Distance	Objects Count	Shallow Size	Retained Size ▼
▶ SlowPurchase	3	300 001 31 %	3 600 012 3 %	127 200 104 89 %
▶ FastPurchase	3	300 001 31 %	8 400 012 6 %	8 400 104 6 %

Object Pools & Static Allocation

Different approach for managing memory

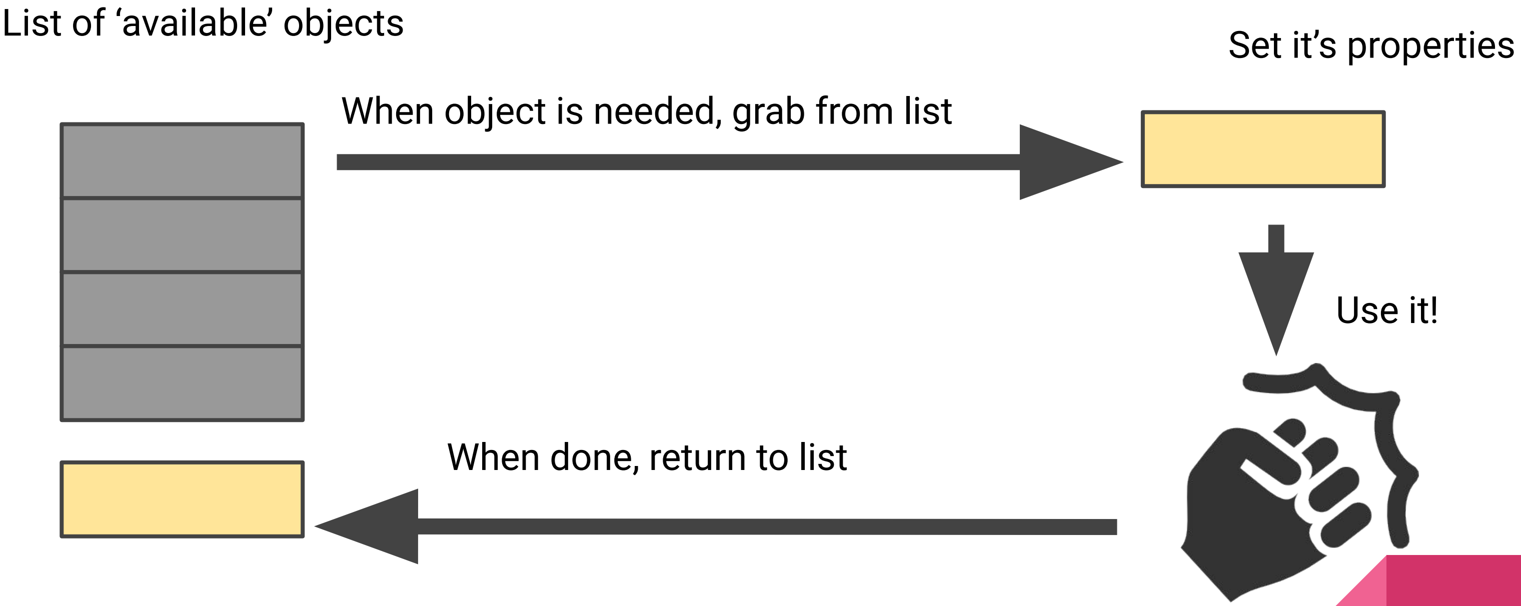
emscripten

Pre-allocates large memory heap

Manages all memory ops to that heap manually.

Yay performance!

Object Pools



List of ‘available’ objects

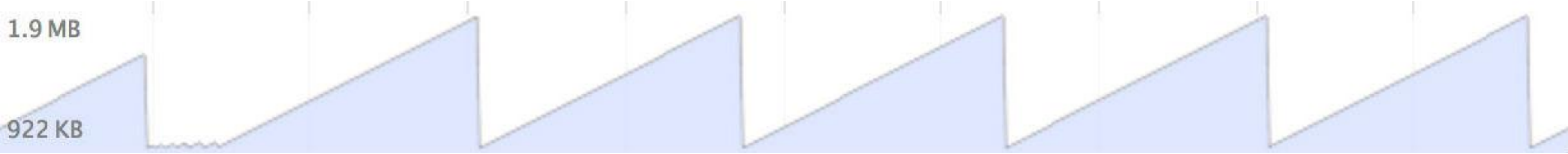
When object is needed, grab from list

Set it’s properties

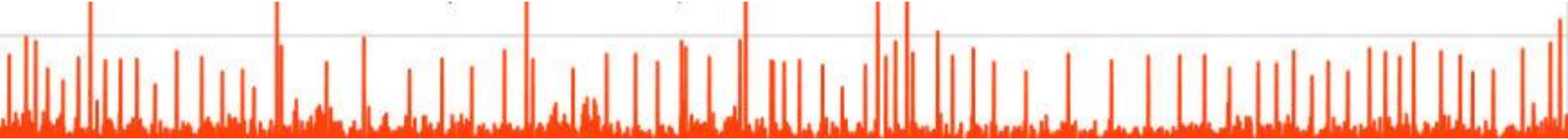
Use it!

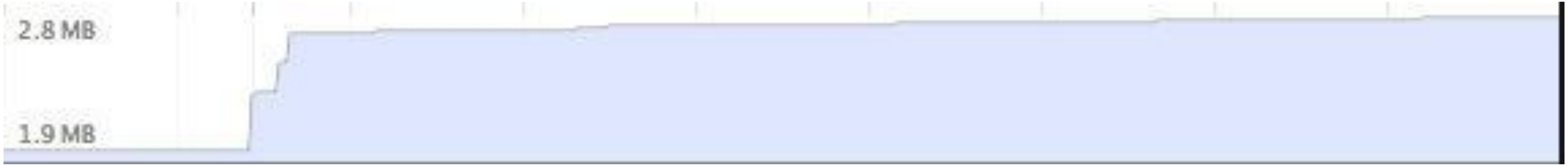
When done, return to list

GC Events



Milliseconds per frame




2.8 MB
1.9 MB

Success.

Memory Checklist

Memory Checklist

- Is my app using too much memory?

Timeline memory view and Chrome task manager can help you identify if you're using too much memory. Memory view can track the number of live DOM nodes, documents and JS event listeners in the inspected render process.

Memory Checklist

- Is my app using too much memory?
- Is my app free of memory leaks?

The Object Allocation Tracker can help you narrow down leaks by looking at JS object allocation in real-time. You can also use the heap profiler to take JS heap snapshots, analyze memory graphs and compare snapshots to discover what objects are not being cleaned up by garbage collection.

Memory Checklist

- Is my app using too much memory?
- Is my app free of memory leaks?
- How frequently is my app forcing garbage collection?

If you are GCing frequently, you may be allocating too frequently. The Chrome Timeline memory view can help you identify pauses of interest.

Design first.

Code from the design.

Then profile the result.

Optimize at the right time.

> *“Premature optimization is the root of all evil.”*

Donald Knuth

Good rules to follow

- Avoid long-lasting refs to DOM elements you no longer need
- Avoid circular object references

- Use appropriate scope

- Unbind event listeners that aren’t needed anymore
- Manage local cache of data. Use an aging mechanism to get rid of old objects.

Resources

- developers.google.com/web/tools/chrome-devtools/memory-problems
- developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management
- www.html5rocks.com/en/features/performance
- thlorenz.com/v8-perf/
- auth0.com/blog/four-types-of-leaks-in-your-javascript-code-and-how-to-get-rid-of-them/

Thank you!