Large Scale JavaScript

Code Architecture
The Module Pattern
Memory Management
Performance

Agenda

- Web Sites vs. Web Applications
- JavaScript Short History
- Code Architecture For An App
- Module Pattern
- Memory Management Pitfalls

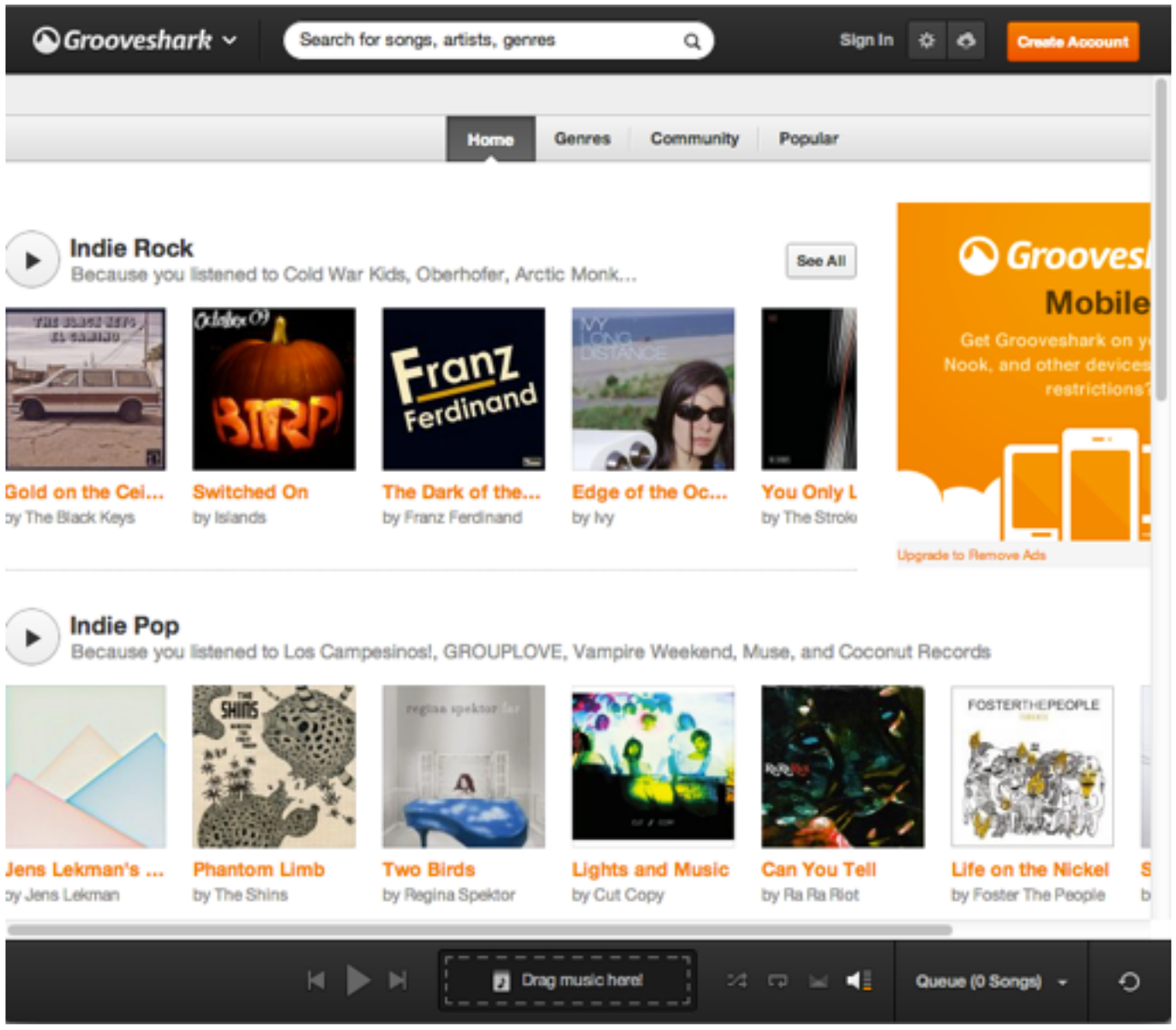
Web Applications Are Not Web Sites

Provide Information

Provide UX

Web Applications Are Not Web Sites

- Applications maintain state
- Applications are more device-specific (fixed sizes and images)
- Applications use device capabilities
- Applications focus on UX



We need a *Different* kind of JavaScript

JavaScript For Web Applications

- Larger code base
- No memory leaks
- Faster and more interactive pages
- Modular And Reusable Code

History

- 1995 Bredan Eich started developing a new language for Netscape Navigator 2.0
- Original name was LiveScript
- Renamed to JavaScript Dec that year
- 1996 MS responded with JScript
- Nov 1996 - Netscape submits JavaScript to ECMA

State Of JavaScript

- Used for frontend web based development for desktop and mobile
- Used for server side development using Node.JS
- Embedded in desktop applications using various JS interpreters
- Used in Windows 8 to write apps

Key Concepts In JavaScript

- It’s interpreter based (though engines are highly optimized)
- It’s loosely typed
- Objects are just hash tables
- Look Ma, OO Without Classes
- Prototypical Inheritance

Q & A

JavaScript Objects

- Objects are containers for data and functions
- They hold key/value pairs
- key is a string (or else it is stringified)
- value can be anything

name	Ynon Perek
website	http://www.ynonperek.com

JavaScript Objects

- Define an object with {...}
- Add key/value pair by assigning to it
- Remove key/value pair with delete
- Print an object with console.dir(...)
- Try not to change them too often

```
var me = {
    name: 'ynon',
    email: 'ynon@ynonperek.com'
};

me.web = 'ynonperek.com';
delete me.web;

var key = 'likes';
me[key] = 'JavaScript';

console.dir( me );
```

Maker Functions

Creating Class-Like Interfaces

Let’s Make Objects

- Create 5 objects to represent 5 different people
- Each should have a name and age keys
- Each should have a method to introduce themselves

Classes are important

- They make it easy to create new objects
- They make it less error prone
- They help JavaScript engines optimize
- They make our code readable

Maker Functions as classes: Take #1

- Works, but not scalable

```
function Person(name, age) {
    this.hello = function() {
        console.log( name );
    };
}

var p = new Person('Jim', 12);
p.hello();
```

Maker Functions as classes: Take #1

- What happens here ?

```
function Person(name, age) {
    this.met = 0;

    this.hello = function() {
        this.met += 1;
        console.log( name + ' Met ' + this.met + ' people' );
    };
}

var p = new Person('Jim', 12);
p.hello();

setTimeout( p.hello, 10 );
```

JavaScript Objects Gotchas

- Methods are bound on **invocation** and not on **declaration**.
- Only closures are bound on **declaration.**
- Solution: Replace `this` with closures.

Maker Functions as classes: Take #2

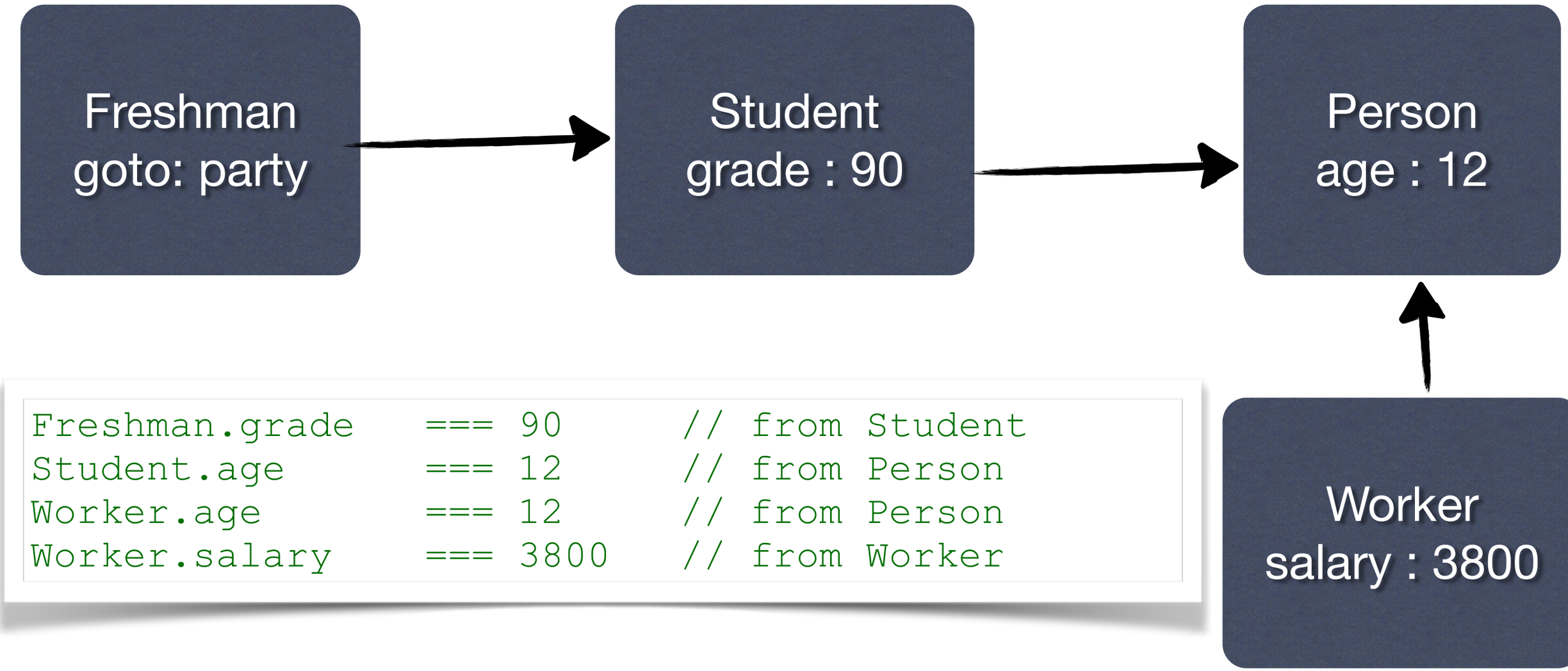
```
function Person(name, age) {
    var self = this;

    self.met = 0;
    self.hello = function() {
        self.met += 1;
        console.log( name + ' Met ' + self.met + ' people' );
    };
}

var p = new Person('Jim', 12);
p.hello();

setTimeout( p.hello, 10 );
```

Q & A

Maker Functions Lab

http://ynonperek.com/course/mobile/javascript-exer.html

Prototype Chain




```
Freshman.grade    === 90      // from Student
Student.age       === 12      // from Person
Worker.age        === 12      // from Person
Worker.salary     === 3800    // from Worker
```

Prototype Chain

- Every JavaScript Object has a prototype
- When the JS interpreter fails to find a property in the object, it searches the prototype
- A prototype can also have its own prototype, that's why we call it a chain
- Define A Prototype by assigning value to the `.prototype` key in the maker function
- Tip: Use prototypes for functions, not for data

Demo: Defining A Prototype without Data

```
var PersonPrototype = {
    hi: function() {
        console.log('Hi ! ');
    }
};
```

```
function Student(age, gradesArray) {
  ...
}

Student.prototype = PersonPrototype;

var me = new Student(20, [90, 80, 70, 80]);
console.log( me.grade() );

// Works from the prototype
me.hi();
```

Prototype and Class Interaction

- Since methods are bound on invocation, inside the prototype `this` usually refers to the child object

- Use `this` to add data to the original object

- Use `this` to read data from the original object

- Never store data on the prototype itself (it'll be shared).

Demo: Prototype and Class Interaction

```
var PersonPrototype = {
  _age: 0,

  hi: function() {
    console.log('Hi ! ' + this._age);
  },

  grow_up: function() {
   this._age += 1;
  }
};
```

```
// Works from the prototype
me.hi();

me.grow_up();

me.hi();

console.log('Me: ');
console.dir( me );

console.log('Prototype: ');
console.dir( PersonPrototype );
```

Prototypes Lab

- Write a Car object. Car should provide `max_speed()` and `drive()` functions.

- Write maker functions for 3 car models (Mitsubishi, BMW and Mazda). Use Car as their prototype. Make sure `max_speed` returns the correct value for each model.

- Add a Race object which takes multiple cars in its maker function and provides a “go” method which prints out which is the fastest.

Prototype Lab 2

- Given the class Product as described here: http://jsbin.com/ogapuq/1/edit

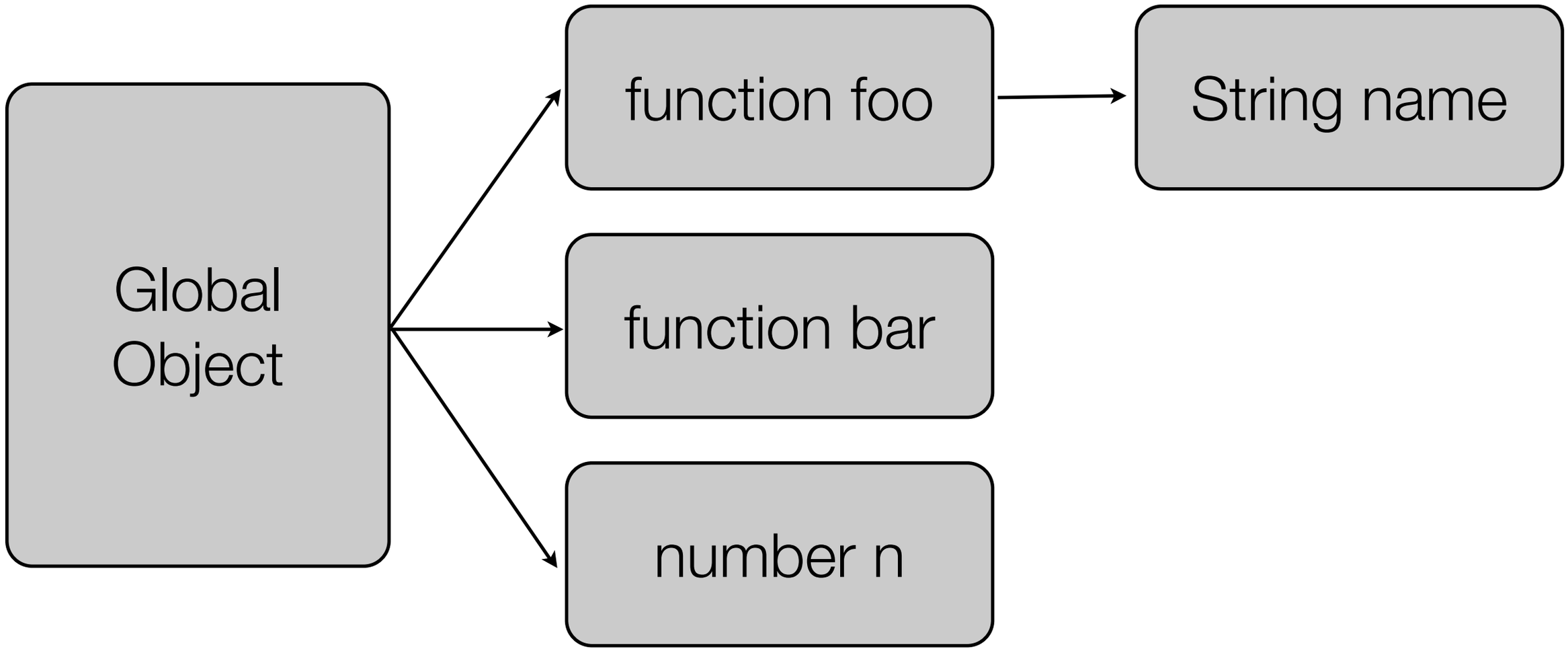
- Write a class Book
  - Ctor should take 3 arguments: name, price and num_pages
  - Book should provide a function price_per_page (returns price / number of pages)
  - Book should provide everything Product provides

Code Architecture For An App

Avoiding Global Objects
Using Namespaces
Using Single Var Statement

JavaScript Bad Choice

- By default, all variables are global
- By global - we mean properties of the global object



Variable Scoping Options

- A Variable can be defined either on the global object, or on a function object
- To define a lexical function variable, use the keyword `var` inside a function

```
function foo() {
    var n = 5;
    var m = 8;
}
```

What's Wrong With Globals

- Bad Programming Style
- Globals make it hard to write and use libraries
- Globals are never garbage-collected

Eliminating Globals Is Hard

```
var main = function() {
    var x = 5;
        y = 7;

    for ( var i=0; i < 10; i++ ) {
        console.log( x + y );
        x += y;
    }
};

main();
```

But It's Possible

```
(function() {
    var x = 5, y = 7;
    for ( var i=0; i < 10; i++ ) {
        console.log( x + y );
        x += y;
    }
}());
```

When Globals Are Inescapable

- Define and use a namespace
- Define classes (Make functions)
- You'll need a global to share data between code written in multiple files
- Demo: Using Globals

What's Variable Hoisting And Why You Should Care

- JavaScript can only create variables as attributes of objects
- In global scope, they're attributes of the global object
- In function scope, they're attributes of the function calling object
- Since variables are **not lexically scopes**, some interesting bugs arise

Hoisting Bug

- Buggy Code:
  http://jsbin.com/iveteb/1/edit

- Fixed Version:
  http://jsbin.com/iveteb/2/edit

Possible Workaround

- Avoid using multiple var statements inside a function
- It won’t prevent the bug, but at least it’ll be easier to spot
- Remember: Only functions have variables

Lab

http://ynonperek.com/course/mobile/javascript-exer.html

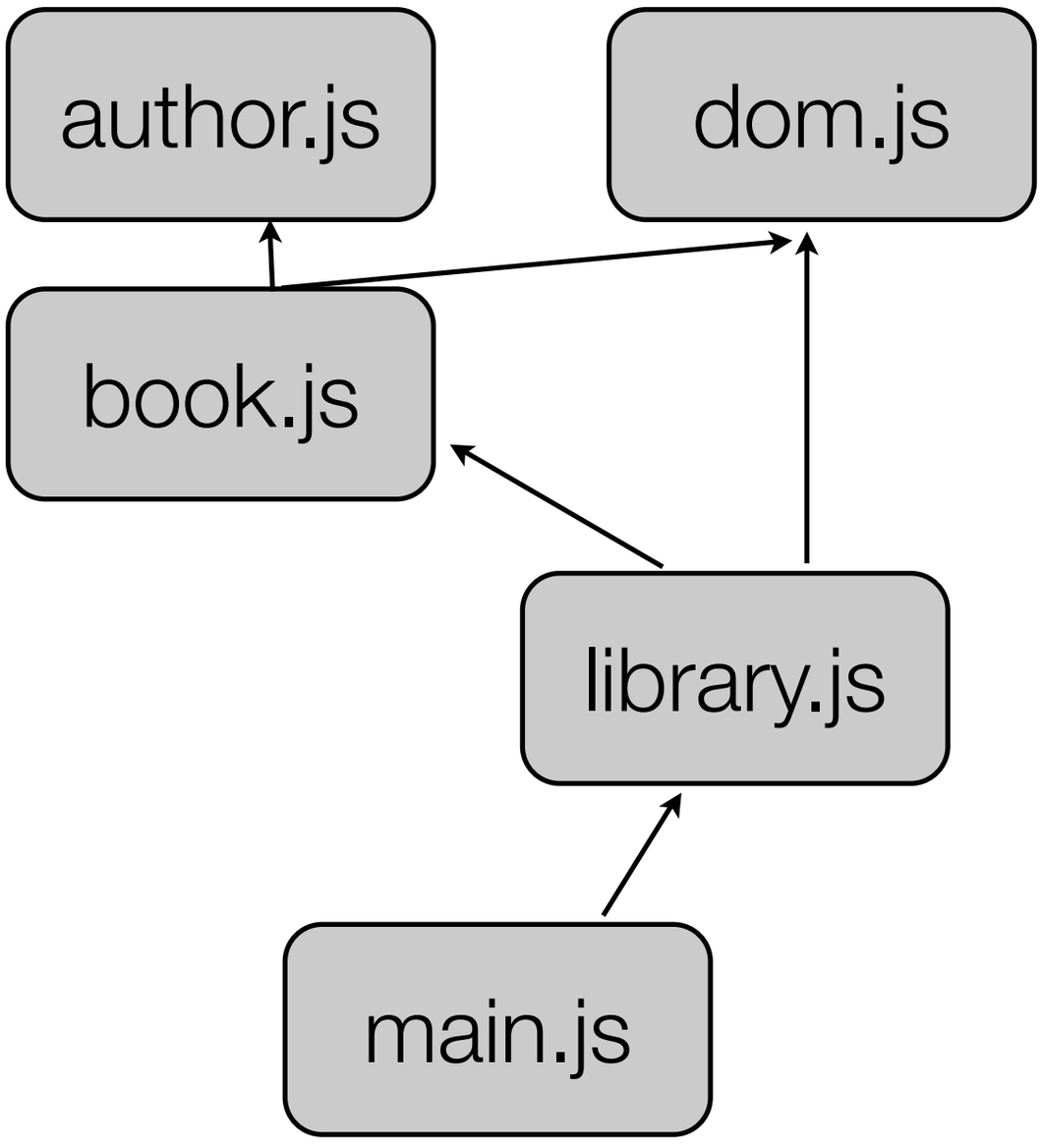
Q & A

Globals and Namespaces

Dynamic Module Loading

What's Wrong With <script>
How: Require.JS To The Rescue
Require.JS Plugins. OMG

Using <script> Tags




```
<script src="dom.js"></script>
<script src="author.js"></script>
<script src="book.js"></script>
<script src="library.js"></script>
<script src="main.js"></script>
```

Disadvantages Of Script Tags

- No Hierarchy in modules
- No optional modules
- Slower code loading (Need to wait for EVERYTHING to load)
- Too many globals

Hello Require.JS

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    <script data-main="scripts/main"  src="http://
cdnjs.cloudflare.com/ajax/libs/require.js/2.1.1/require.min.js"></
script>
</body>
</html>
```

Hello Require.JS - Defining A Module

```
// book.js
define(function() {
    return function(title, author) {
      this.title  = title;
      this.author = author;
      this.year   = new Date().getUTCFullYear();
    };
});
```

Hello Require.JS - Defining A Module

```
// book.js
define(function() {
    return function(title, author) {
      this.title  = title;
      this.author = author;
      this.year   = new Date().getUTCFullYear();
    };
});
```

Note the return value
of the callback define takes

Hello Require.JS - Using A Module

```
// main.js
define(['book'], function(Book) {
  var foo = new Book('JS Rocks', 'Jim');
  console.dir( foo );
});
```

Hello Require.JS - Using A Module

return value of book.js

```
// main.js
define(['book'], function(Book) {
    var foo = new Book('JS Rocks', 'Jim');
    console.dir( foo );
});
```

Define can take an array of dependencies as its first argument. It'll call the callback passing each module as argument

Require and Browser Cache

- By default require instructs the browser to cache all modules
- For development, you can override it by setting urlArgs
- For production, add a constant version value

```
<script>
    require.config({
        urlArgs: "bust=" + new Date().getTime()
    });

</script>
```

What We Got From Require.JS

- No need for global variables or namespaces
- JavaScript files can declare dependencies between each other
- No need to manually set <script> tags order

Defining Modules

- Use define({...}) to define an object that has no dependencies

```
//Inside file my/shirt.js:
define({
    color: "black",
    size: "unisize"
});
```

Defining Modules With Initialization Code

- Use define(function() { ... }) to define a module that has initialization code

- Example:

```
//my/shirt.js now does setup work
//before returning its module definition.
define(function () {
    //Do setup work here

    return {
        color: "black",
        size: "unisize"
    }
});
```

Defining A Module With Dependencies

- Use define([array], function(...) {}) to define a module with dependencies
- All dependencies will be loaded async, and then your callback function is invoked, with all dependencies as arguments

Defining A Module With Dependencies

- Example:

```
//my/shirt.js now has some dependencies, a cart and inventory
//module in the same directory as shirt.js
define(["./cart", "./inventory"], function(cart, inventory) {
        //return an object to define the "my/shirt" module.
        return {
            color: "blue",
            size: "large",
            addToCart: function() {
                inventory.decrement(this);
                cart.add(this);
            }
        }
    }
);
```

Using Maker Functions As Modules

- define's callback's return value can be anything
- Returning a Maker Function from the callback results in a "Class"
- Usage:

```
define(['book'], function(Book) {
  var book = new Book();
});
```

Modules Notes

- Define only one module per file

- Don’t do circular dependencies (but if you must, here’s how to do it right: http://requirejs.org/docs/api.html#circular)

- Modules are loaded by adding <script> tags to head (head.appendChild()).

Require Configuration

```
<script>
    require.config({
        option: value
    });

</script>
```

Supported Options

- baseUrl: Root path to use for all modules lookup. Defaults to the path of the main script, or the location of containing HTML.

- shim: allow using require(...) on old modules by telling require how they work. Example:

```
shim: {
    'backbone': {
        deps: ['underscore', 'jquery'],
        //Once loaded, use the global 'Backbone' as the
        //module value.
        exports: 'Backbone'
    },
```

Supported Options

- waitSeconds: timeout to wait for a module. Defaults to 7.
  If set to 0, waits forever.

- enforceDefine: throws an error if trying to load a script that's not AMD nor defined in a shim

- urlArgs: extra query string data (useful to disable cache)

Demo: Using Vendor Libraries

- Use require config to include vendor libraries from a CDN
- You can even pass an array to provide local fallback

```
requirejs.config({
  paths: {
    jquery: [
      'http://ajax.googleapis.com/ajax/libs/jquery/1.4.4/
jquery.min',
      //If the CDN location fails, load from this location
      'lib/jquery'
    ]
  }
});
```

Handling Errors

- Define a global requirejs.onError function to handle errors
- Function takes two arguments: error object and (optional) modules array

```
requirejs.onError = function(etype, emod) {
    console.log('error');
    console.dir( etype );
    console.dir( emod );
}
```

Require Plugins

- Plugins extend the functionality of require.js
- They allow loading of different file types
- They allow conditional loading of files
- They allow delayed execution

Text Plugin

- Put the file `text.js` in your scripts path
- The plugin uses XHR to get the file
- Great for splitting HTML to smaller files

```
define(['text!test.html'],
function(txt) {
  // now txt is a text string
  // containing the value of test.html
  console.log( txt );
});
```

domReady Plugin

- Wait for a document ready event

- Argument is current document object

- Don’t forget to add domReady.js to your scripts folder

```
require(['domReady!'],
   function (doc) {
      // called when DOM is ready
});
```

i18n Plugin

- Support multiple languages
- Progressive/partial translations
- Change texts without modifying the application

```
define(['i18n!nls/texts'], function(texts) {

  document.querySelector('p').innerHTML      = texts.hello;
  document.querySelector('button').innerHTML = texts.clickme;
});
```

Some More Plugins

- image! loads an image
- json! loads a JSON object
- font! loads a web font
- feature! conditional loading of modules

Q & A

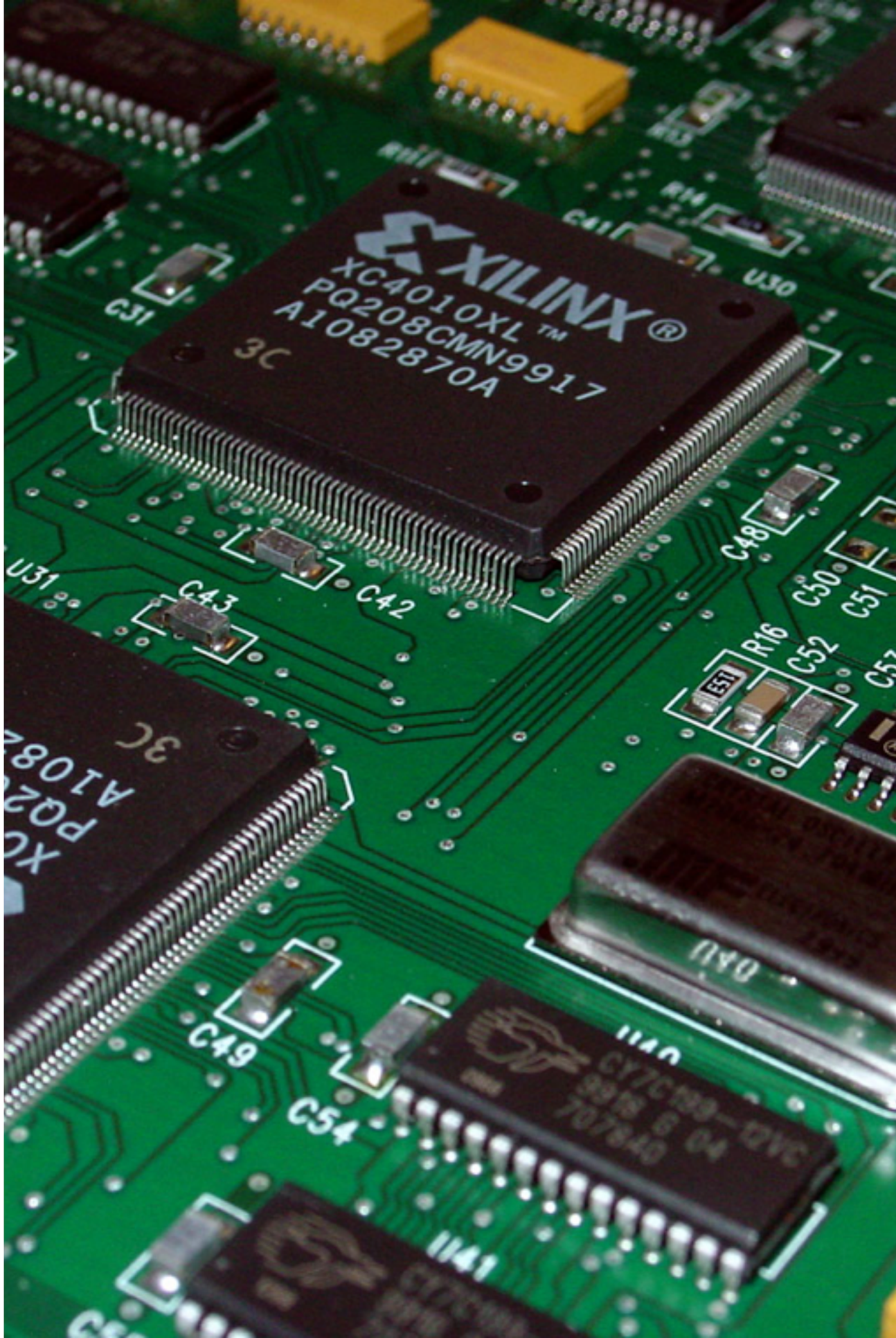
Require.JS

Require.JS Lab

- Write a Quiz App:
    - A Common Question Prototype Module
    - A Module for Multiple Choice Question
    - A Module for scale (1-5) question
    - A Module for the Quiz (has an array of questions)
- Write a quiz with 3 questions and translate it to hebrew

JavaScript Memory Management

Memory Lifecycle
Garbage Collector
Common Leaks



Memory Lifecycle

- All languages are the same:
- (1) Allocate some memory
- (2) Use that memory
- (3) Free that memory
- In JS, #3 is implicit

Memory Is Allocated When You Define Literals

```
var n = 123;
var s = "azerty";

var o = {
  a: 1,
  b: null
};

var a = [1, null, "abra"];

function f(a){
  return a + 2;
}

someElement.addEventListener('click', function(){
  someElement.style.backgroundColor = 'blue';
}, false);
```

Hidden Memory Allocations

```
var d = new Date();
var e = document.createElement('div'); // allocates an DOM
element
```

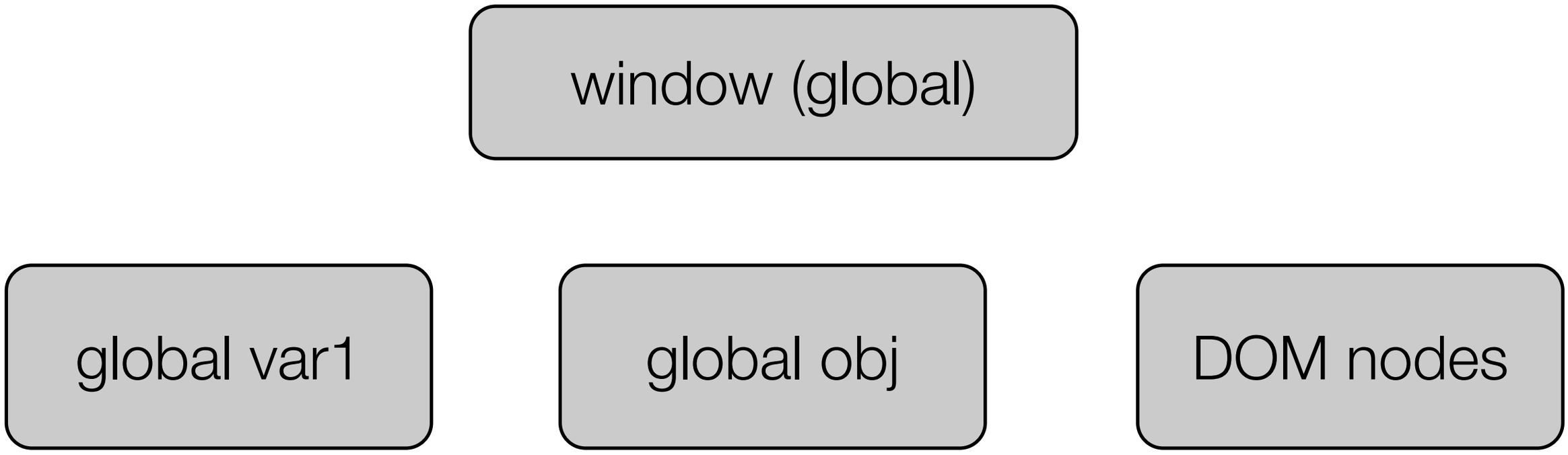
```
var s = "foo";
var s2 = s.substr(0, 3); // s2 is a new string

var a = ["ouais ouais", "nan nan"];
var a2 = ["generation", "nan nan"];
var a3 = a.concat(a2);
```

Releasing Memory

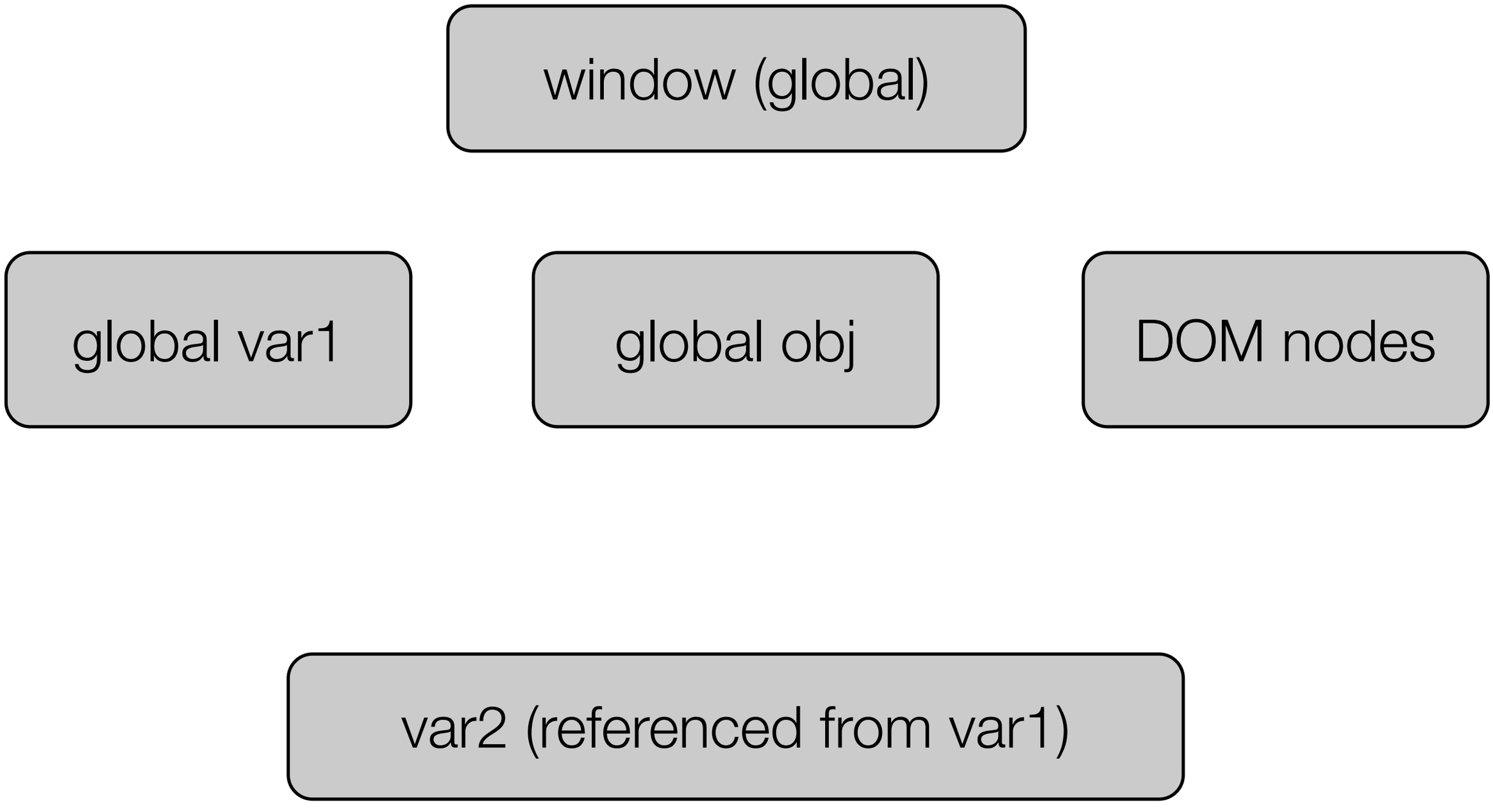
- JavaScript uses Mark-And-Sweep garbage collection algorithm
- It starts from known memory (global object)
- Follows all references
- In the end, clear the unreachable

Objects Graph

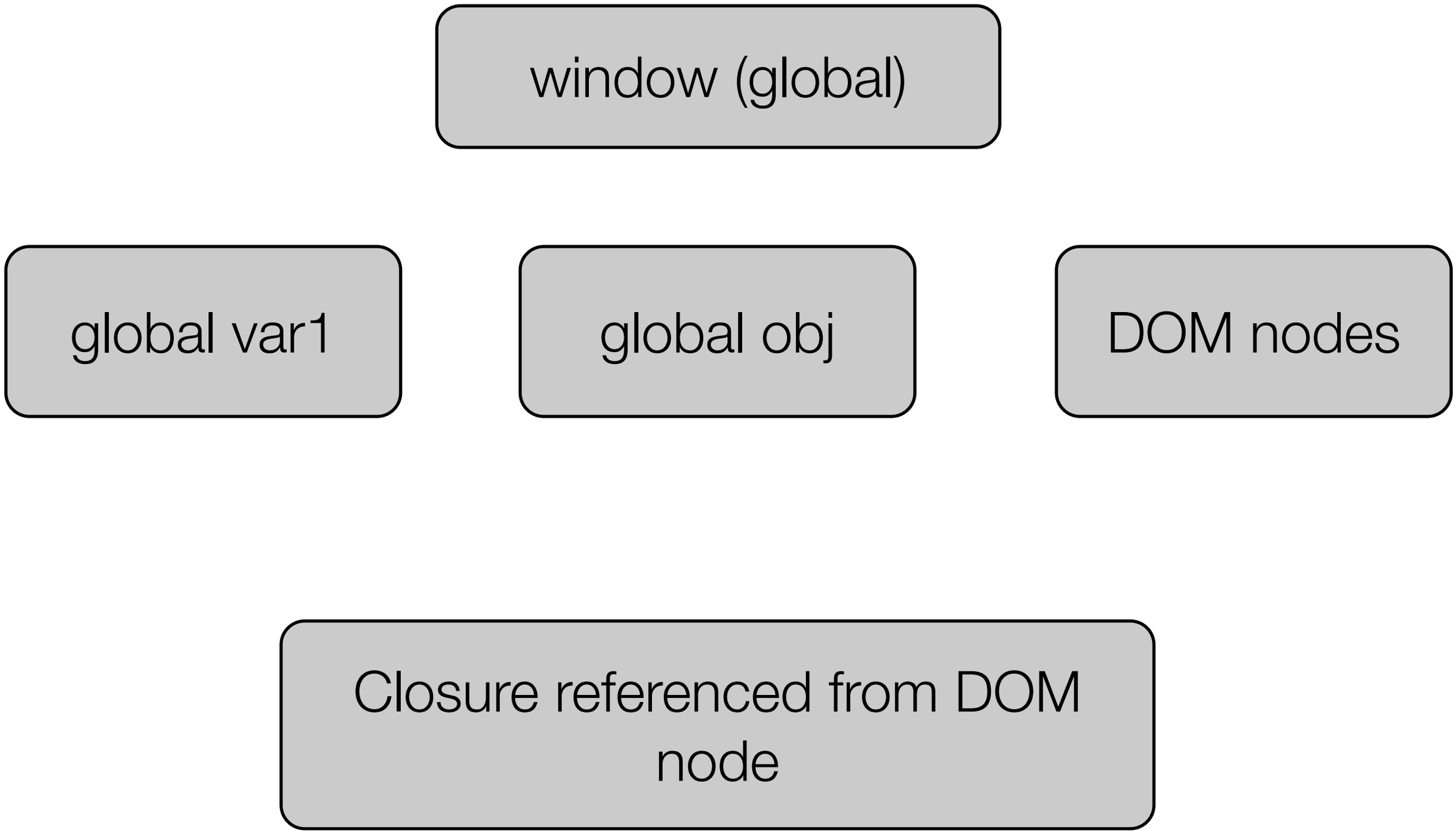
Objects Graph



Objects Graph



Objects Graph



Cleaning Up Memory

An object is released when:

garbage collector runs

AND

that object is unreachable

Good Release Candidates

- Function scope variables (lexicals) defined with var
- Detached DOM elements that nobody needs anymore
- Deleted object properties

Cleaning Up Memory

- Avoid global variables
    - A global is NEVER freed
    - A lexical is freed when out-of-scope
- Limit cache sizes
- Don’t use old IE (6-7) - but if you do, use jQuery

Common Leak: Hanging Detached Nodes

```
<button>Click Me</button>
<div>10</div>
<script>
    var btn = document.querySelector('button');
    var counter = 10;
    var div = document.querySelector('div');

    btn.addEventListener('click', function() {

        if ( counter < 0 ) return;

        counter -= 1;
        div.innerHTML = counter;
        if ( counter == 0 ) {
            div.parentElement.removeChild(div);
        }
    });
</script>
```

Fix By Using Lexicals

```
<button>Click Me</button>
<div>10</div>
<script>
    var btn = document.querySelector('button');
    var counter = 10;

    btn.addEventListener('click', function() {
        var div = document.querySelector('div');

        if ( counter < 0 ) return;

        counter -= 1;
        div.innerHTML = counter;
        if ( counter == 0 ) {
            div.parentElement.removeChild(div);
        }
    });
</script>
```

Common Leak: Unlimited Cache Size

```
var cache = {};

var fib = function(n) {
  if ( ! cache[n] ) {
    if ( n < 2 ) {
      cache[n] = 1
    } else {
      cache[n] = fib(n-1) + fib(n-2);
    }
  }
  return cache[n];
};
```

Fix: Smart Caching

- Define Cache() class to take size limit on ctor (or use default)
- Cache.add should check if has enough space, or remove old elements
- Bonus: play with keep/remove strategies to reach maximum performance

```
var cache = new Cache(10);

var fib = function(n) {
  if ( ! cache[n] ) {
    if ( n < 2 ) {
      cache.add(n, 1 );
    } else {
      cache.add(n, fib(n-1) + fib(n-2) );
    }
  }
  return cache.get(n);
};
```

Common Error: Keeping Multiple Function Copies

- Each function takes space in memory
- A constructor that puts functions directly on the object makes the object larger
- Many objects that share the same function are a waste

```
var Person = function() {
  var self = this;

  self.hi = function() {
    console.log('Hello World!');
  };

  self.grow_up = function() {
    self.age += 1;
  };

  self.age = 0;
};
```

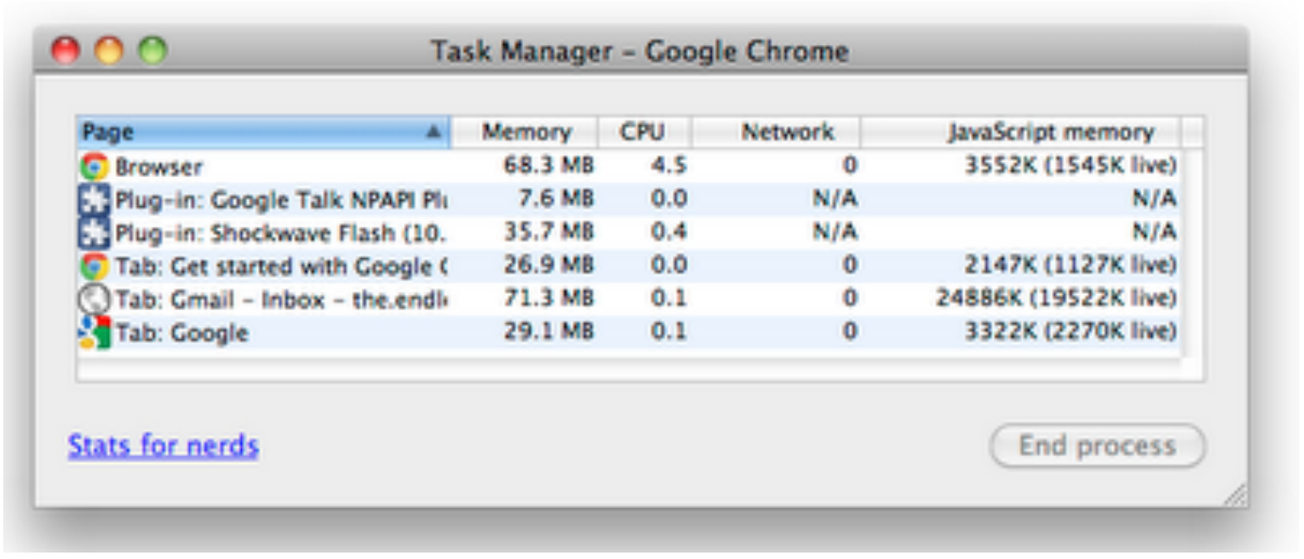
Fix: Use Prototypes

- Functions are defined in prototypes
- Data is defined in object
- Use bind when you need to use a function as an event handler
- Note: Can hurt readability

```
var PersonActions = {
  hi: function() {
    console.log('Hello World!');
  },
  grow_up: function() {
    this.age += 1;
  }
};

Person.prototype = PersonActions;
```

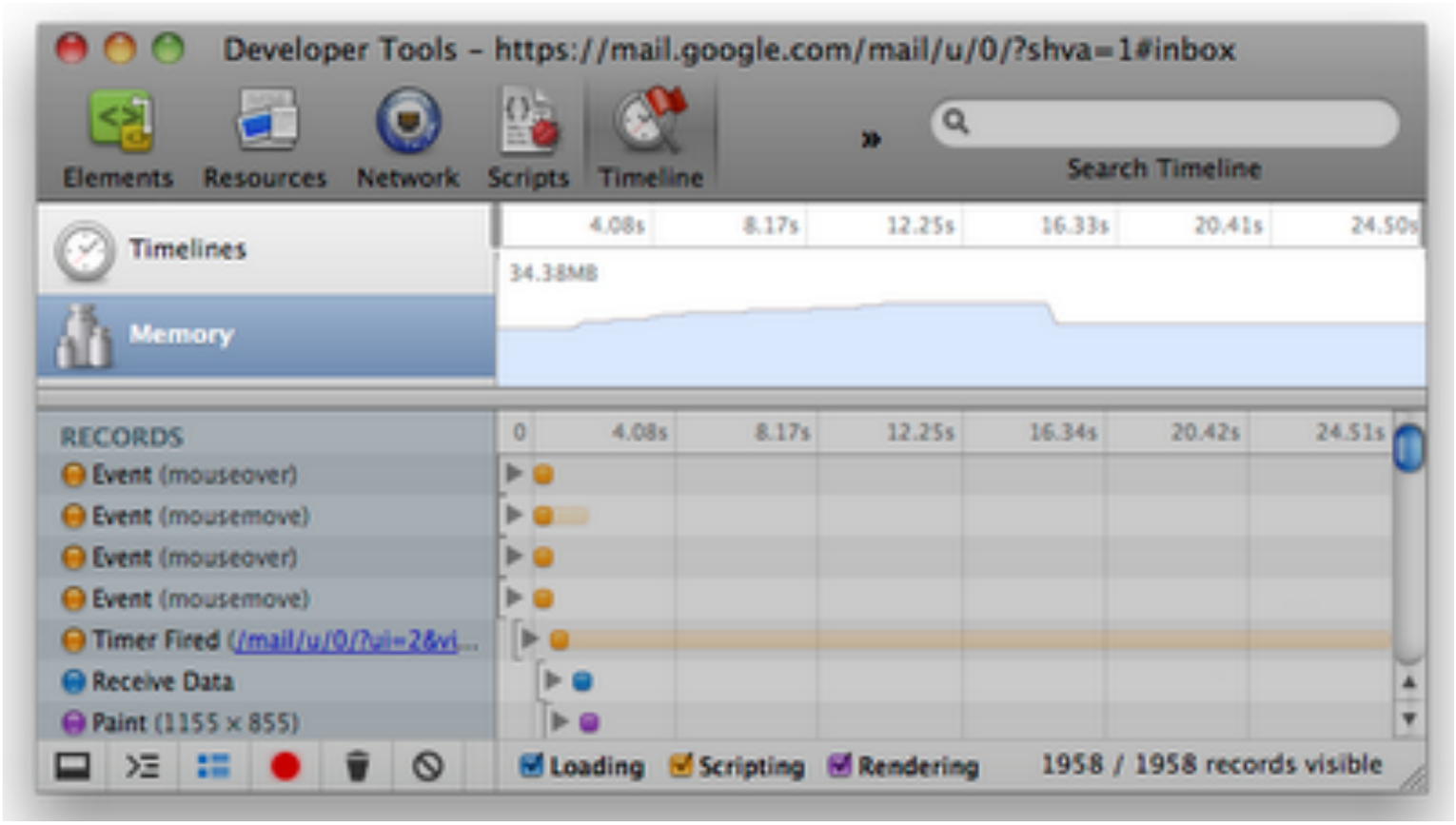
Tools For Memory Management

- Chrome Task Manager, available from:
- Tools->Task Manager
- If memory tab is hidden, turn it on via context menu



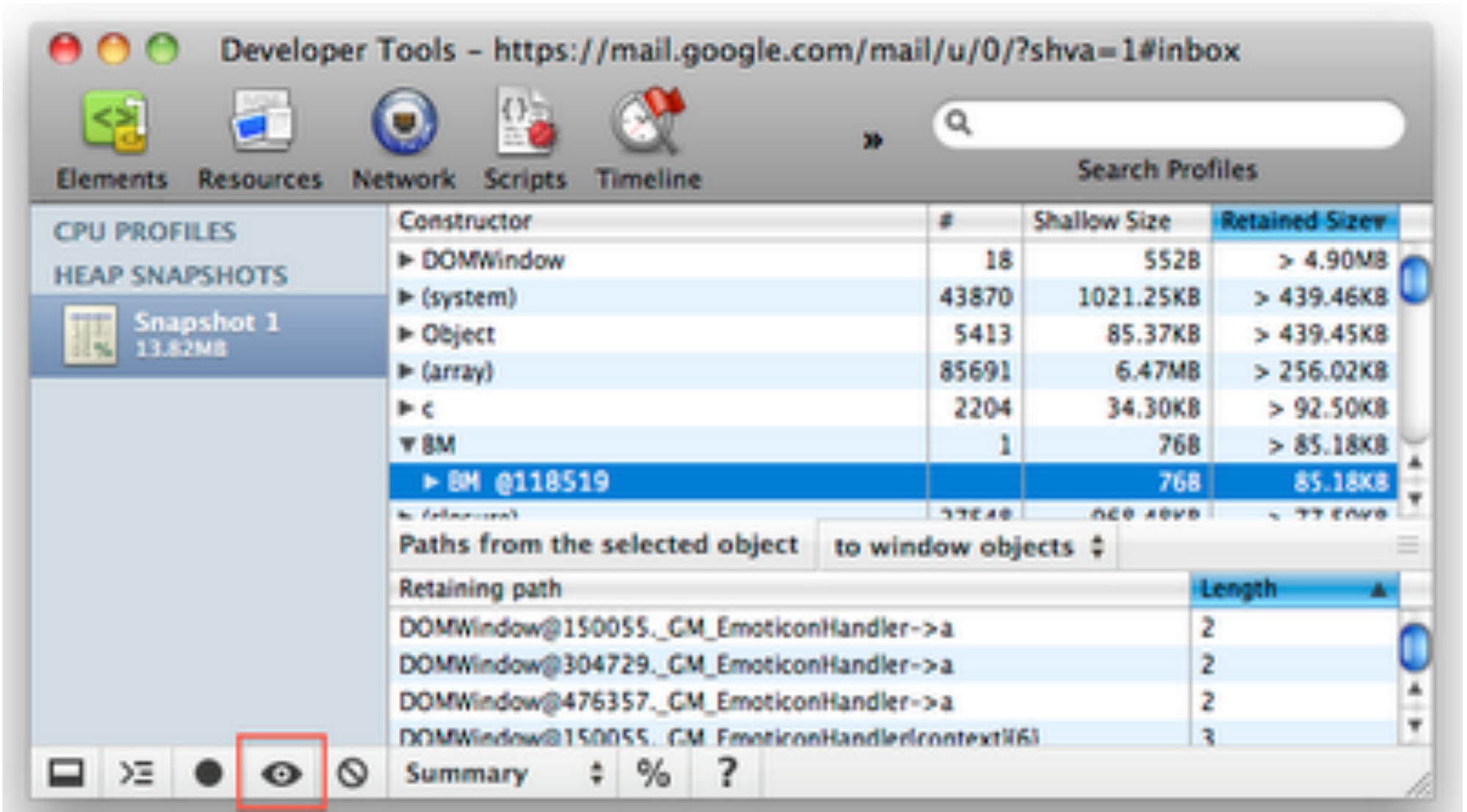
Tools For Memory Management

- Chrome Memory Timeline
- Available from Developer Tools



Tools For Memory Management

- Chrome Heap Profiler
- Available from Developer Tools



Lab

- There’s a memory leak in the following code: https://gist.github.com/ynonp/4742321

- And this one: https://gist.github.com/ynonp/4749795

- What is leaking ?

- What’s causing the leak ?

- Fix It !

Q & A

Memory Management

Thanks For Joining

- Slides By:
- Ynon Perek
- ynon@ynonperek.com
- http://ynonperek.com

- Free photos from:
- 123rf.com
- http://morguefile.com