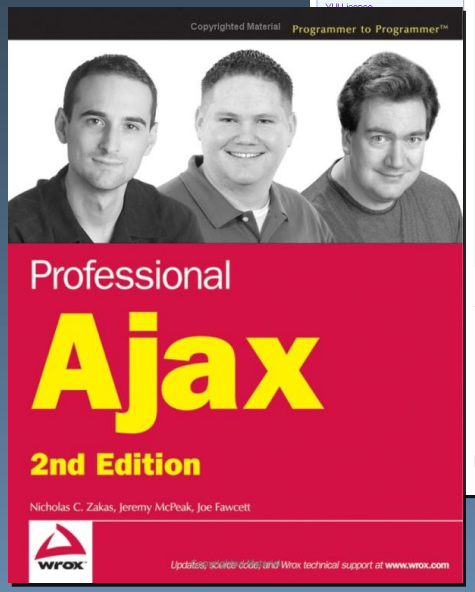
Speed Up Your JavaScript

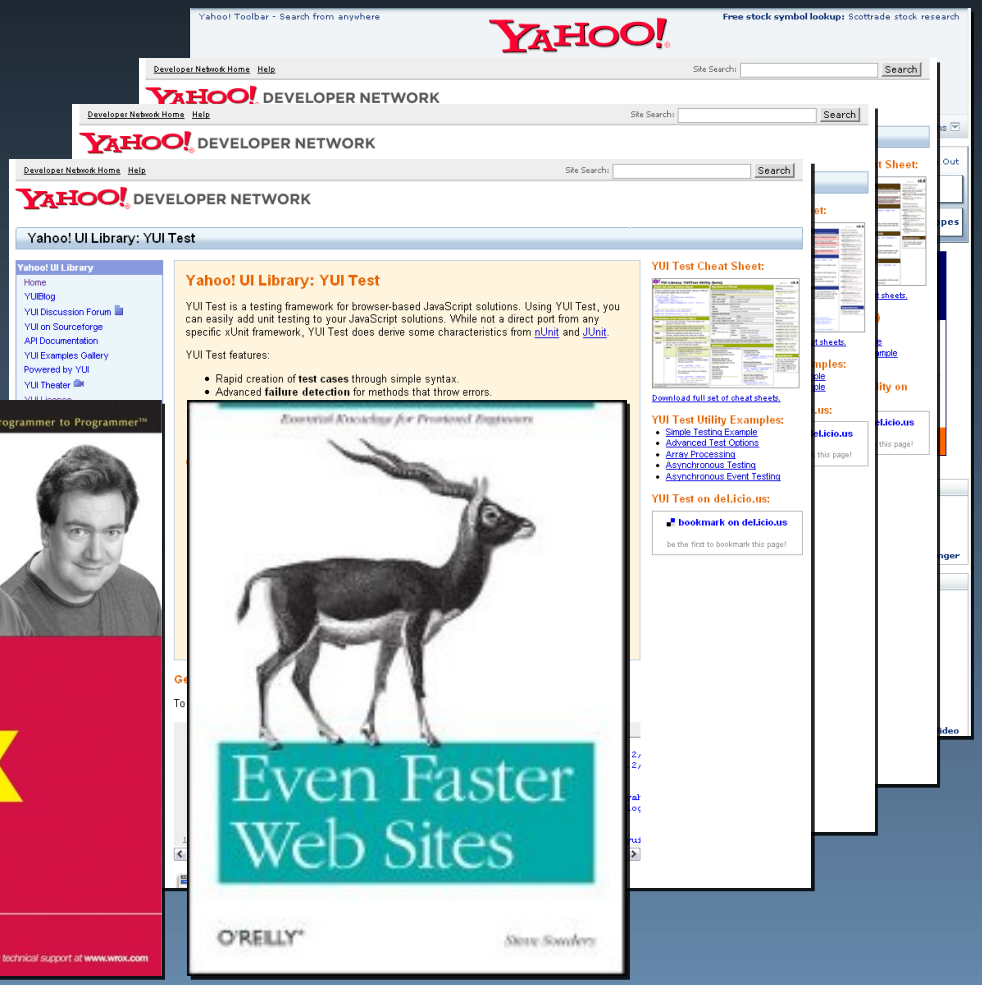
Nicholas C. Zakas
Principal Front End Engineer, Yahoo!

Web E^{x}ponents – June 4, 2009

Who's this guy?

- Principal Front End Engineer, Yahoo! Homepage
- YUI Contributor
- Author





is getting tired of javascript. All it does is slow down page navigation and add complicated layouts and consume zillion resources

12:03 PM May 12th from XMPP Gateway

ultraleetj
Juan Bello

Why slow?

Bad compilation?

No

No compilation!*

* Humor me for now. It'll make this easier.

Browsers won't help your code!!!!

Who will help your code?

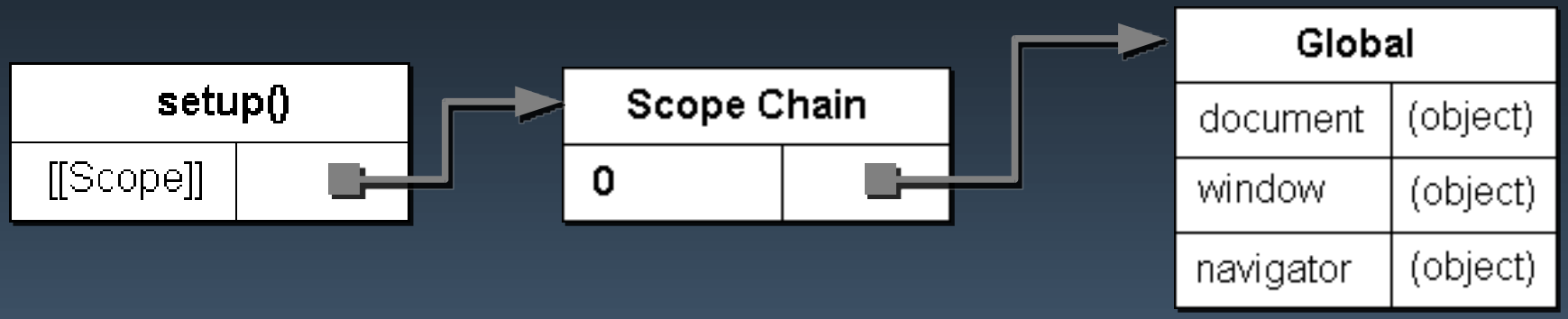

SMOKEY
WOLFE
ONLY YOU

JavaScript Performance Issues

- Scope management
- Data access
- Loops
- DOM

```
function setup(items){

    var divs = document.getElementsByTagName("div");
    var images = document.getElementsByTagName("img");
    var button = document.getElementById("save-btn");

    for (var i=0; i < items.length; i++){
        process(items[i], divs[i]);
    }

    button.addEventListener("click", function(event){
        alert("Saved!");
    }, false);

}
```

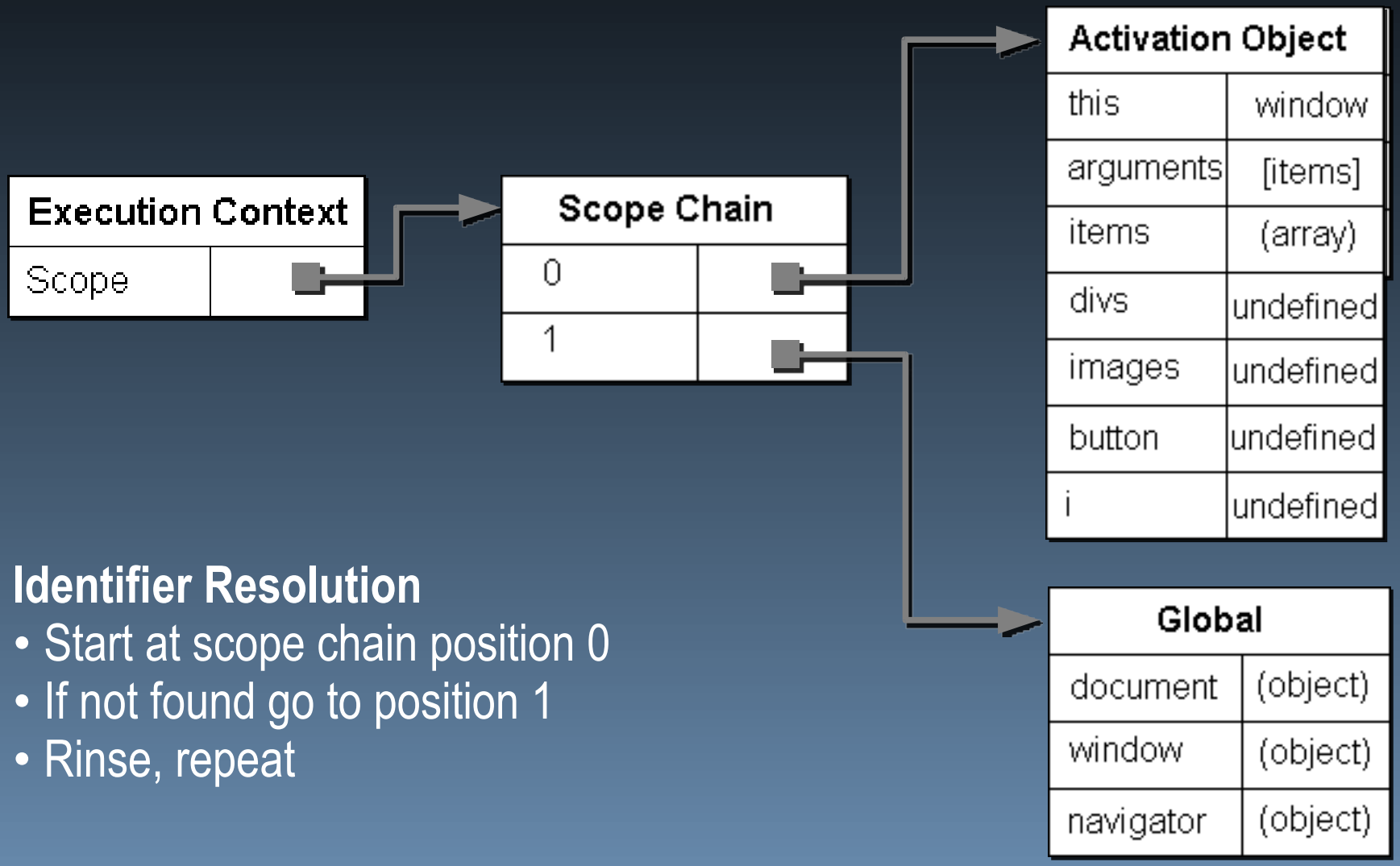
Scope Chains



When a Function Executes

- An execution context is created
- The context's scope chain is initialized with the members of the function's [[Scope]] collection
- An activation object is created containing all local variables
- The activation object is pushed to the front of the context's scope chain

Execution Context



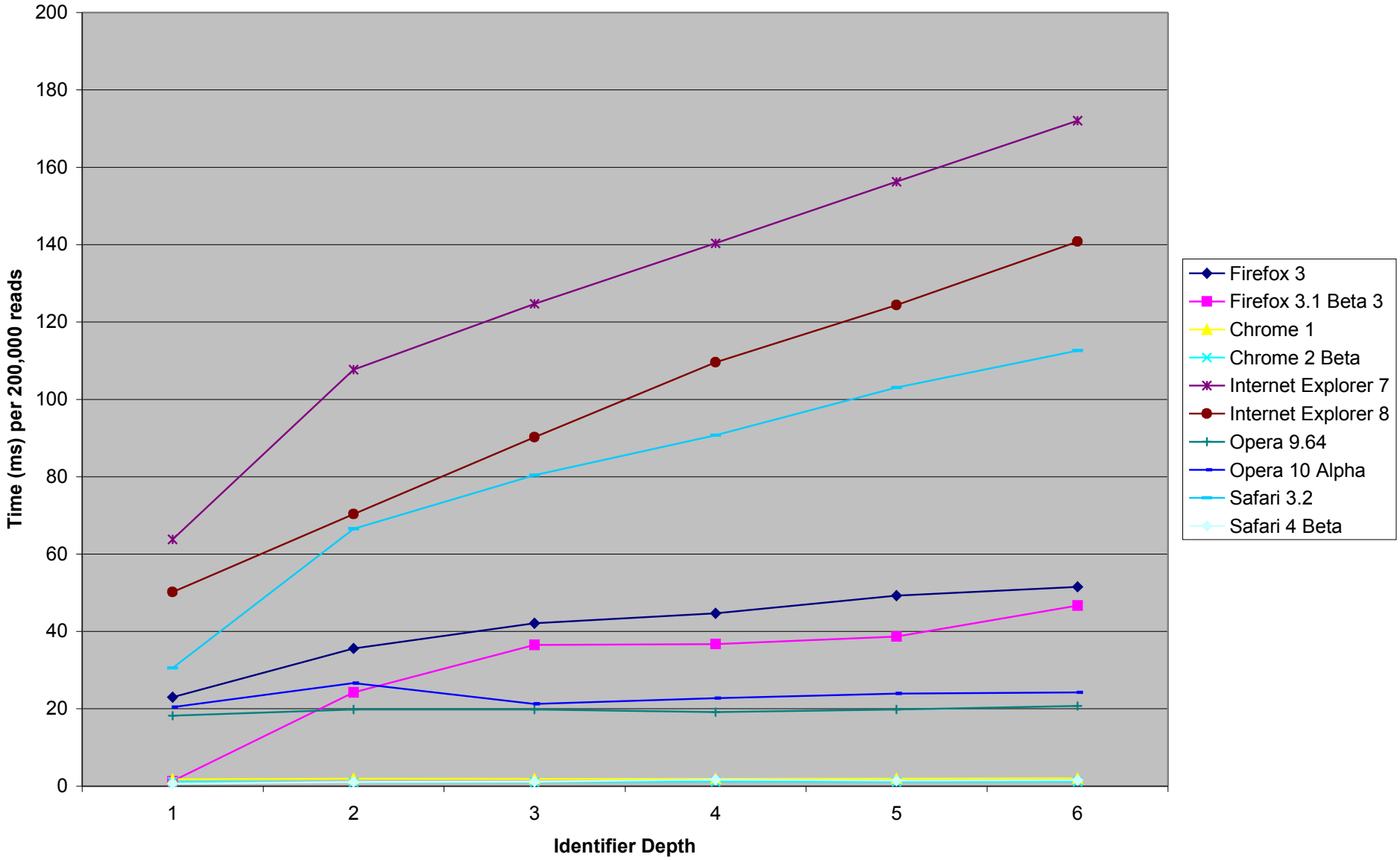
Identifier Resolution

- Start at scope chain position 0
- If not found go to position 1
- Rinse, repeat

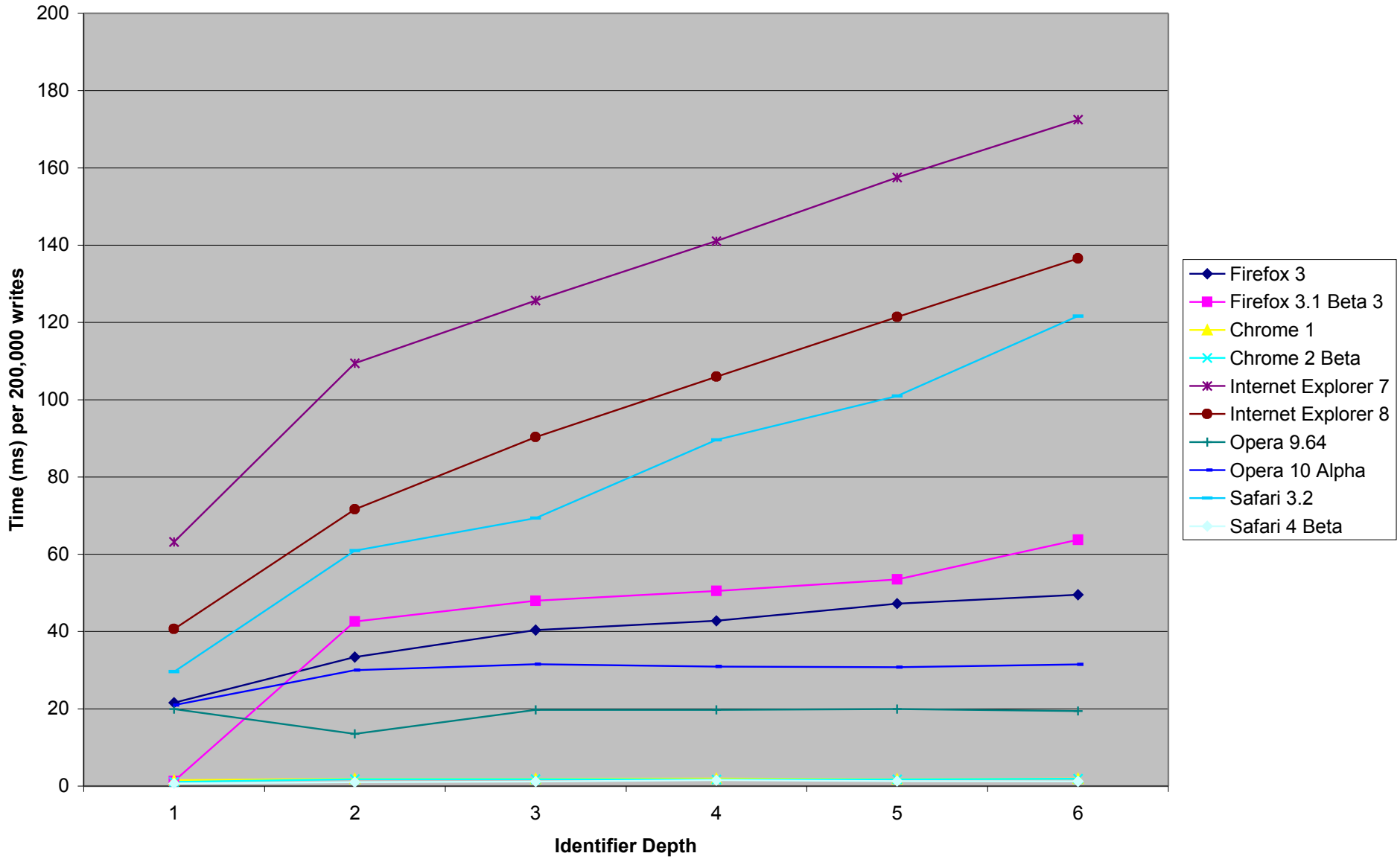
Identifier Resolution

- Local variables = fast!
- The further into the chain, the slower the resolution

Identifier Resolution (Reads)



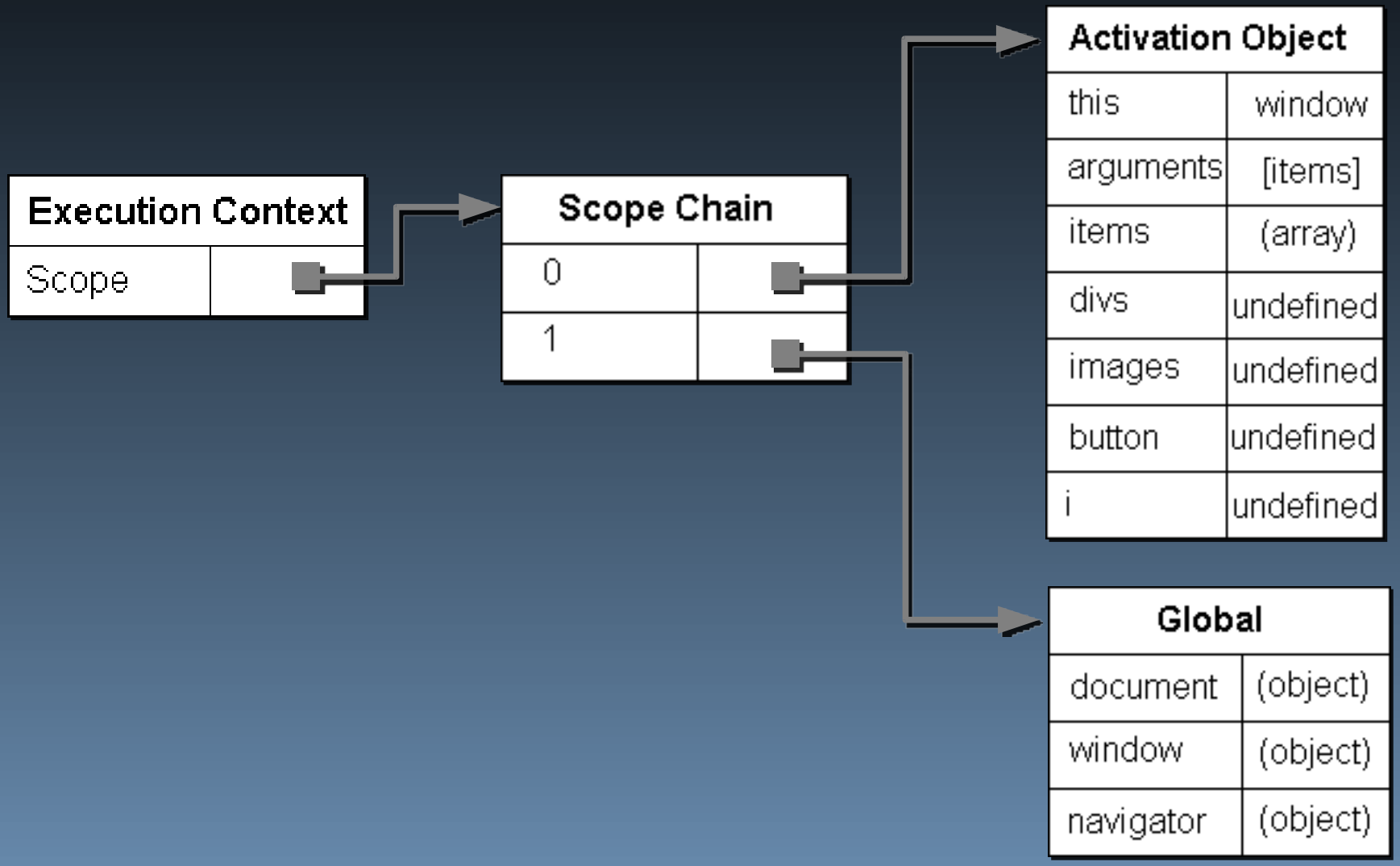
Identifier Resolution (Writes)



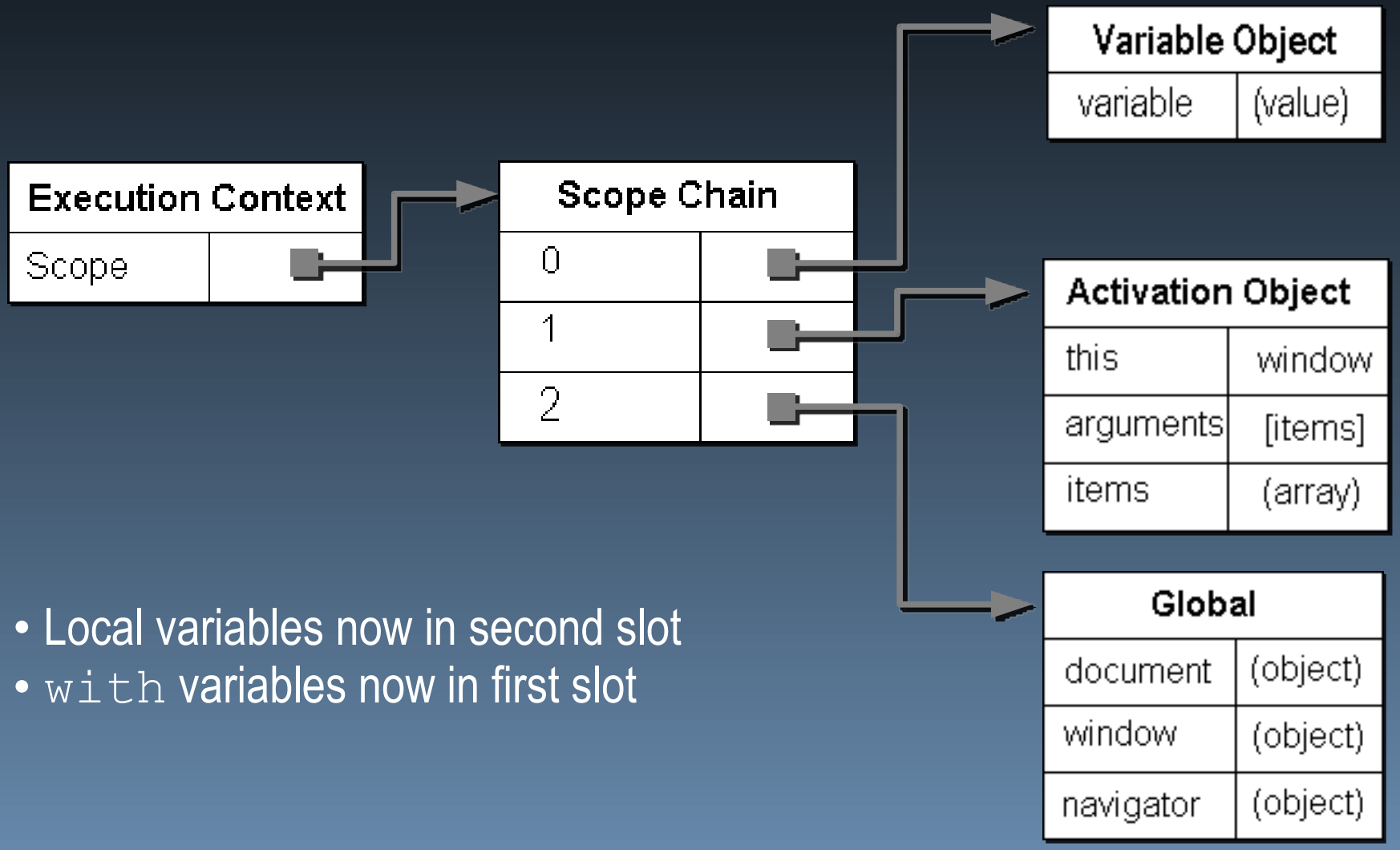
Scope Chain Augmentation

- The `with` statement
- The `catch` clause of `try-catch`
- Both add an object to the front of the scope chain

Inside of Global Function



Inside of `with`/`catch` Statement



- Local variables now in second slot
- `with` variables now in first slot

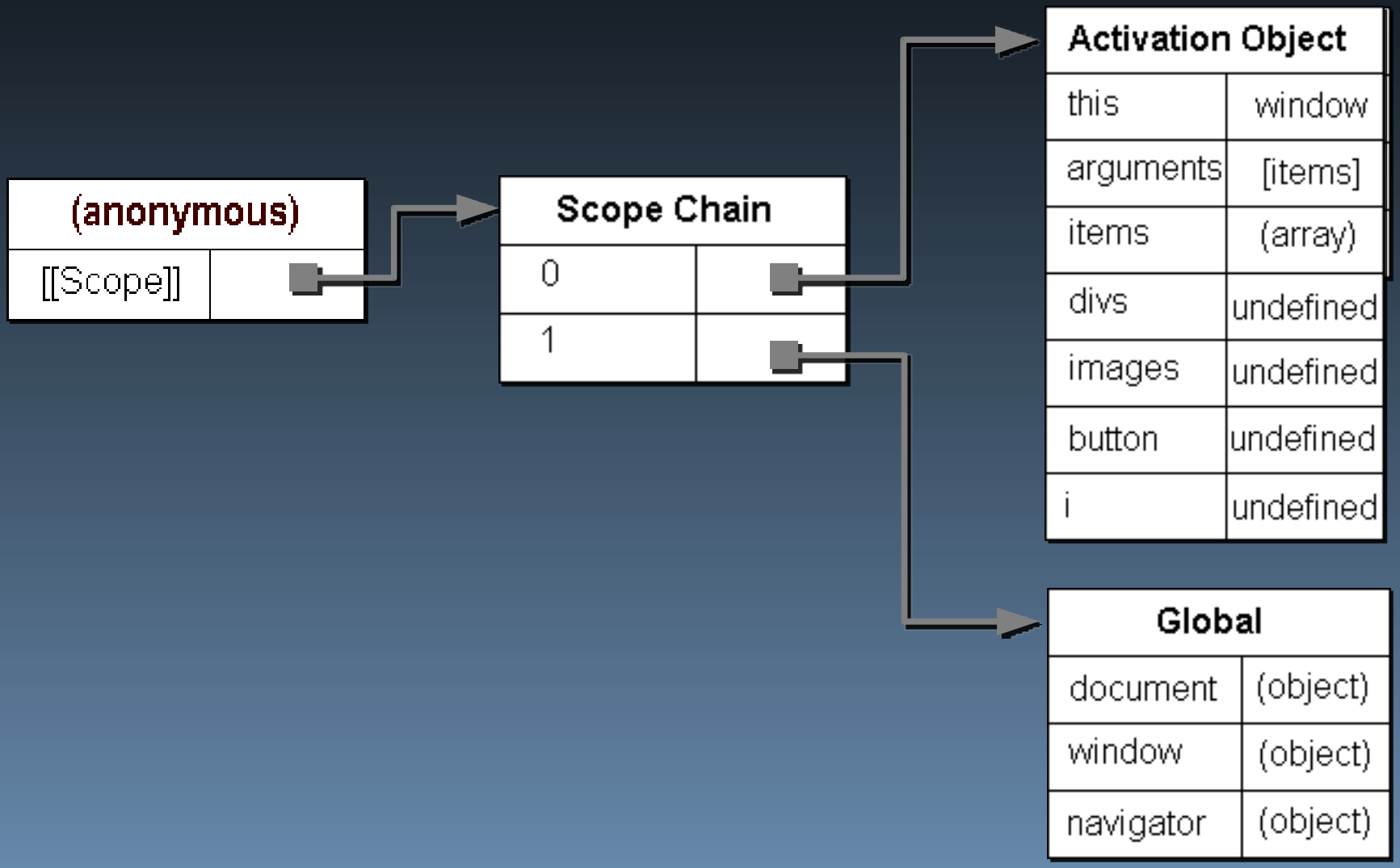

“with statement considered harmful”
-Douglas Crockford

Closures

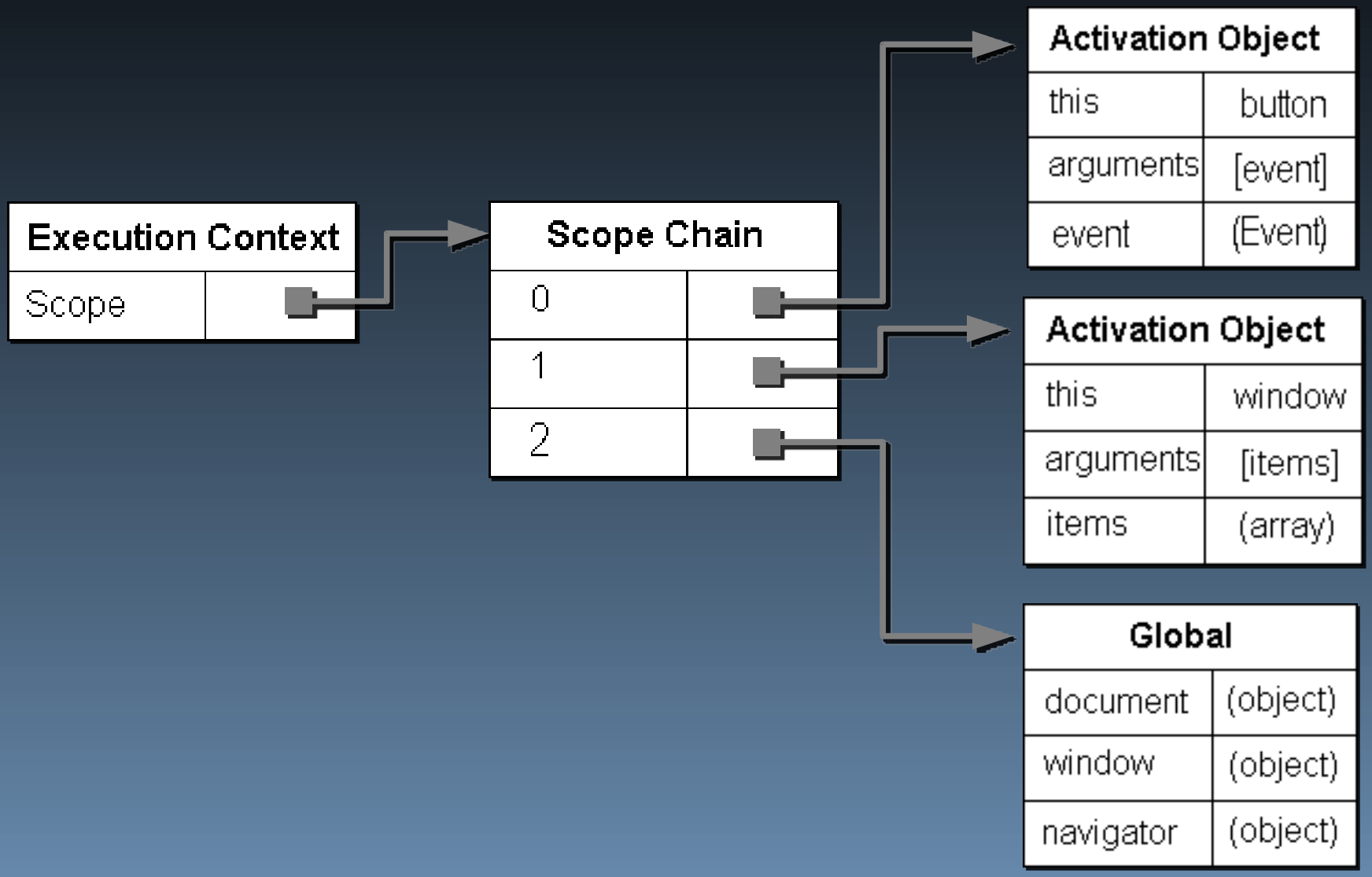
- The [[Scope]] property of closures begins with two objects
- Calling the closure means three objects in the scope chain (minimum)

```
function setup(items){

    var divs = document.getElementsByTagName("div");
    var images = document.getElementsByTagName("img");
    var button = document.getElementById("save-btn");

    for (var i=0; i < items.length; i++){
        process(items[i], divs[i]);
    }

    button.addEventListener("click", function(event){
        alert("Saved!");
    }, false);

}
```

Closures


(anonymous)
[[Scope]]
Scope Chain
0
1
Activation Object
this
window
arguments
[items]
items
(array)
divs
undefined
images
undefined
button
undefined
i
undefined
Global
document
(object)
window
(object)
navigator
(object)

Inside of Closure



Recommendations

- Store out-of-scope variables in local variables
  - Especially global variables
- Avoid the `with` statement
  - Adds another object to the scope chain, so local function variables are now one step away
  - Use local variables instead
- Be careful with `try-catch`
  - The `catch` clause also augments the scope chain
- Use closures sparingly
- Don't forget `var` when declaring variables

```
function setup(items){

    var doc = document;
    var divs = doc.getElementsByTagName("div");
    var images = doc.getElementsByTagName("img");
    var button = doc.getElementById("save-btn");

    for (var i=0; i < items.length; i++){
        process(items[i], divs[i]);
    }

    button.addEventListener("click", function(event){
        alert("Saved!");
    }, false);

}
```

JavaScript Performance Issues

- Scope management
- Data access
- Loops
- DOM

Places to Access Data

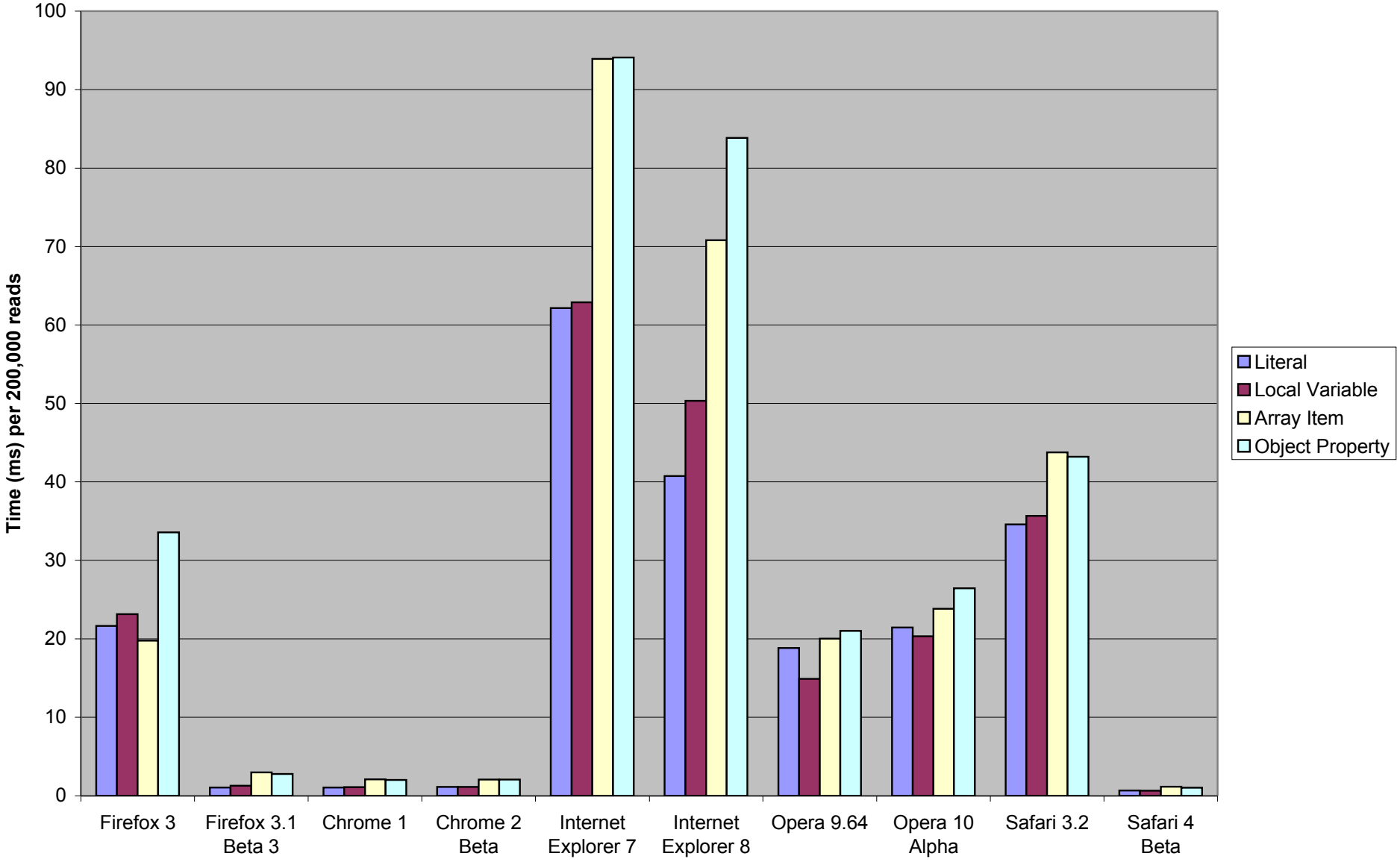
- Literal value
- Variable
- Object property
- Array item

```
1   //literal
2   var name = "Nicholas";
3
4   //variable
5   var name2 = name;
6
7   //object property
8   var name3 = object.name;
9
10  //array item
11  var name4 = items[0];
```

Data Access Performance

- Accessing data from a literal or a local variable is fastest
  - The difference between literal and local variable is negligible in most cases
- Accessing data from an object property or array item is more expensive
  - Which is more expensive depends on the browser

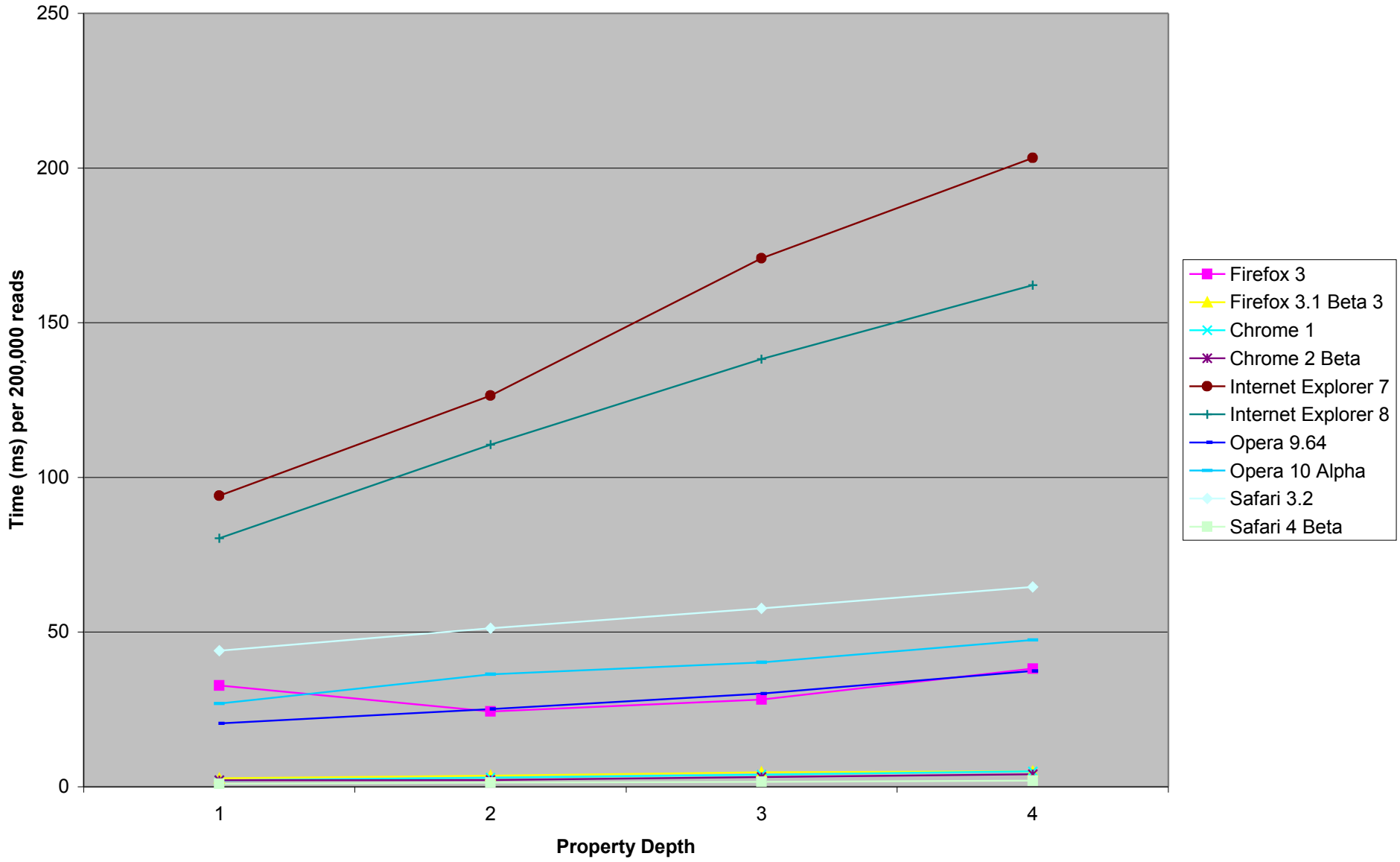
Data Access


Time (ms) per 200,000 reads
100
90
80
70
60
50
40
30
20
10
0
Firefox 3
Firefox 3.1 Beta 3
Chrome 1
Chrome 2 Beta
Internet Explorer 7
Internet Explorer 8
Opera 9.64
Opera 10 Alpha
Safari 3.2
Safari 4 Beta
Literal
Local Variable
Array Item
Object Property

Property Depth

- `object.name` < `object.name.name`
- The deeper the property, the longer it takes to retrieve

Property Depth



Property Notation

- Difference between `object.name` and `object["name"]`?
  - Generally no
  - Exception: Dot notation is faster in Safari

Recommendations

- Store these in a local variable:
  - Any object property accessed more than once
  - Any array item accessed more than once
- Minimize deep object property/array item lookup

```
function process(data){
    if (data.count > 0){
        for (var i=0; i < data.count; i++){
            processData(data.item[i]);
        }
    }
}
```

```
function process(data){
    var count = data.count,
        item  = data.item;
    if (count > 0){
        for (var i=0; i < count; i++){
            processData(item[i]);
        }
    }
}
```

JavaScript Performance Issues

- Scope management
- Data Access
- Loops
- DOM

Loops

- ECMA-262, 3rd Edition:
  - `for`
  - `for-in`
  - `do-while`
  - `while`
- ECMA-357, 2nd Edition:
  - `for each`

```
//for loop
for (var i=0; i < values.length; i++){
    process(values[i]);
}

//do-while loop
var j=0;
do {
    process(values[j++]);
} while (j < values.length);

//while loop
var k=0;
while (k < values.length){
    process(values[k++]);
}
```

Which loop?

It doesn't matter!

What Does Matter?

- Amount of work done per iteration
  - Includes terminal condition evaluation and incrementing/decrementing
- Number of iterations
- These don't vary by loop type

Fixing Loops

- Decrease amount of work per iteration
- Decrease number of iterations

```
//for loop
for (var i=0; i < values.length; i++){
    process(values[i]);
}

//do-while loop
var j=0;
do {
    process(values[j++]);
} while (j < values.length);

//while loop
var k=0;
while (k < values.length) {
    process(values[k++]);
}
```

```
//for loop
for (var i=0; i < values.length; i++) {
    process(values[i]);
}

//do-while loop
var j=0;
do {
    process(values[j++]);
} while (j < values.length);

//while loop
var k=0;
while (k < values.length){
    process(values[k++]);
}
```

```
//for loop
for (var i=0; i < values.length; i++){
    process(values[i]);
}

//do-while loop
var j=0;
do {
    process(values[j++]);
} while (j < values.length);

//while loop
var k=0;
while (k < values.length){
    process(values[k++]);
}
```

Easy Fixes

- Eliminate object property/array item lookups

```
var len = values.length;

//for loop
for (var i=0; i < len; i++){
    process(values[i]);
}

//do-while loop
var j=0;
do {
    process(values[j++]);
} while (j < len);

//while loop
var k=0;
while (k < len) {
    process(values[k++]);
}
```

Easy Fixes

- Eliminate object property/array item lookups
- Combine control condition and control variable change
  - Work avoidance!

```
var len = values.length;

//for loop
for (var i=0; i < len; i++){
    process(values[i]);
}

//do-while loop
var j=0;
do {
    process(values[j++]);
} while (j < len);

//while loop
var k=0;
while (k < len){
    process(values[k++]);
}
```

Two evaluations:
j < len
j < len == true

```
var len = values.length;

//for loop
for (var i=len; i--; {
    process(values[i]);
}

//do-while loop
var j = len - 1
do {
    process(values[j]);
} while (j--)

//while loop
var k = len;
while (k--)
    process(values[k]);
}
```

One evaluation
j-- == true

-50%

Easy Fixes

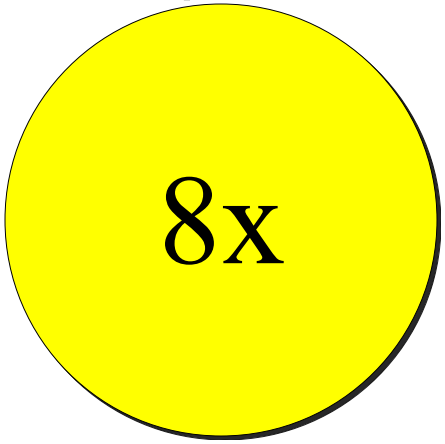
- Eliminate object property/array item lookups
- Combine control condition and control variable change
  - Work avoidance!

Things to Avoid for Speed

- ECMA-262, 3rd Edition:
  - `for-in`
- ECMA-357, 2nd Edition:
  - `for each`
- ECMA-262, 5th Edition:
  - `array.forEach()`
- Function-based iteration:
  - `jQuery.each()`
  - `Y.each()`
  - `$each`
  - `Enumerable.each()`

```
values.forEach(function(value, index, array){
    process(value)
});
```

- Introduces additional function
- Function requires execution (execution context created, destroyed)
- Function also creates additional object in scope chain



JavaScript Performance Issues

- Scope management
- Data Access
- Loops
- DOM


DOM


HTMLCollection

`HTMLCollection` Objects

- `document.images`, `document.forms`, etc.
- `getElementsByTagName()`
- `getElementsByClassName()`

2.3. Miscellaneous Object Definitions

Interface *HTMLCollection*

An `HTMLCollection` is a list of nodes. An individual node may be accessed by either ordinal index or the node's `name` or `id` attributes.

Note: Collections in the HTML DOM are assumed to be *live* meaning that they are automatically updated when the underlying document is changed.

IDL Definition

Note: Collections in the HTML DOM are assumed to be *live* meaning that they are automatically updated when the underlying document is changed.

`length`

This attribute specifies the length or *size* of the list.

Methods

`item`

This method retrieves a node specified by ordinal index. Nodes are numbered in tree order (depth-first traversal order).

Parameters

`index` The index of the node to be fetched. The index origin is 0.

Return Value

The `Node` at the corresponding position upon success. A value of `null` is returned if the index is out of range.

This method raises no exceptions.

`namedItem`

This method retrieves a `Node` using a name. It first searches for a `Node` with a matching `id` attribute. If it doesn't find one, it then searches for a `Node` with a matching `name` attribute, but only on those elements that are allowed a name attribute.

Parameters

Infinite Loop!

```
var divs = document.getElementsByTagName("div");

for (var i=0; i < divs.length; i++){
    var div = document.createElement("div");
    document.body.appendChild(div);
}
```

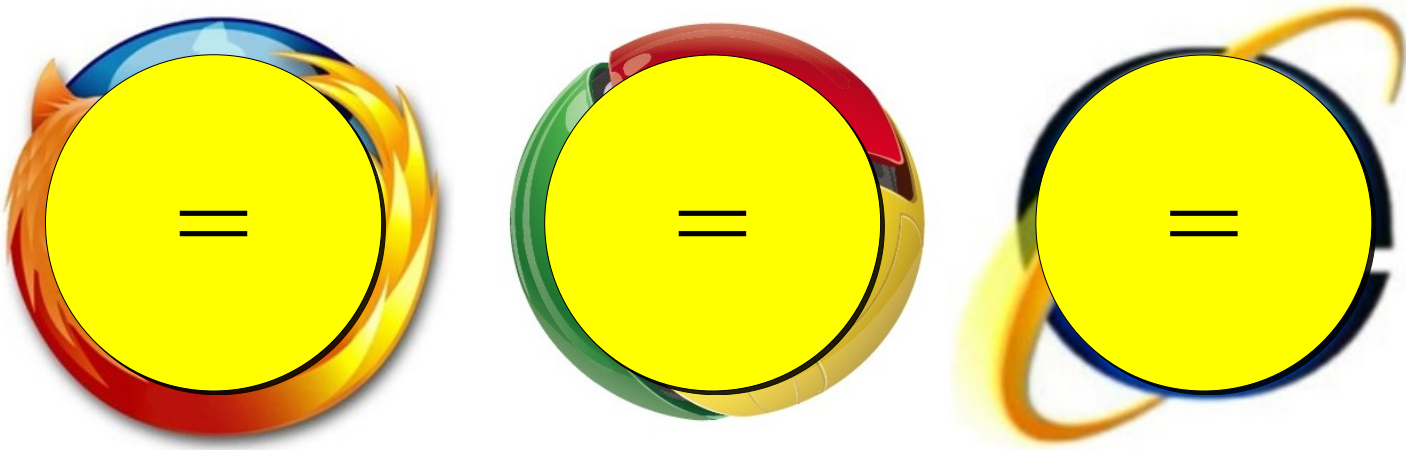
`HTMLCollection` Objects

- Look like arrays, but aren't
  - Bracket notation
  - `length` property
- Represent the results of a specific query
- The query is re-run each time the object is accessed
  - Include accessing `length` and specific items
  - Much slower than accessing the same on arrays
  - Exceptions: Opera, Safari

```
var items = [{}, {}, {}, {}, {}, {}, {}, {}];

for (var i=0; i < items.length  i++){
}
```

```
var divs = document.getElementsByTagName("div");

for (var i=0; i < divs.length; i++){
}
```

```
var items = [{}, {}, {}, {}, {}, {}, {}, {}];

for (var i=0, len=items.length; i < len; i++){
}


var divs = document.getElementsByTagName("div");

for (var i=0, len=divs.length; i < len; i++){
}
```



`HTMLCollection` Objects

- Minimize property access
  - Store length, items in local variables if used frequently
- If you need to access items in order frequently, copy into a regular array

```
function array(items){
    try {
        return Array.prototype.slice.call(items);
    } catch (ex){
        var i      = 0,
            len    = items.length,
            result = Array(len);

        while (i < len){
            result[i] = items[i];
            i++;
        }
    }

    return result;
}
```

Reflow

Reflow is the process by which the geometry of the layout engine's formatting objects are computed.

- Chris Waterson, Mozilla

When Reflow?

- Initial page load
- Browser window resize
- DOM nodes added or removed
- Layout styles applied
- Layout information retrieved

```
var list = document.getElementById("list");

for (var i=0; i < 10; i++){
    var item = document.createElement("li");
    item.innerHTML = "Option #" + (i+1);
    list.appendChild(item);
}
```

Reflow!

`DocumentFragment`

- A document-like object
- Not visually represented
- Considered a child of the document from which it was created
- When passed to `addChild()`, appends all of its children rather than itself

```
var list = document.getElementById("list");
var fragment = document.createDocumentFragment();

for (var i=0; i < 10; i++){
    var item = document.createElement("li");
    item.innerHTML = "Option #" + (i+1);
    fragment.appendChild(item);
}

list.appendChild(fragment);
```

No reflow!

Reflow!

When Reflow?

- Initial page load
- Browser window resize
- DOM nodes added or removed
- Layout styles applied
- Layout information retrieved

```
element.style.height = "100px";
element.style.display = "block";
element.style.fontSize = "130%";
```

Reflow!

Reflow!

Reflow!

What to do?

- Minimize changes on `style` property
- Define CSS class with all changes and just change `className` property

```
.active {
    height: 100px;
    display: block;
    font-size: 130%;
}

element.className = "active";
```

Reflow!

When Reflow?

- Initial page load
- Browser window resize
- DOM nodes added or removed
- Layout styles applied
- Layout information retrieved
  - Only if reflow is cached

```
var width = element.offsetWidth;
var scrollLeft = element.scrollLeft;
var display = window.getComputedStyle(div, '').
                getPropertyValue("display");
```

Reflow?

Reflow?

Reflow?

What to do?

- Minimize access to layout information
- If a value is used more than once, store in local variable

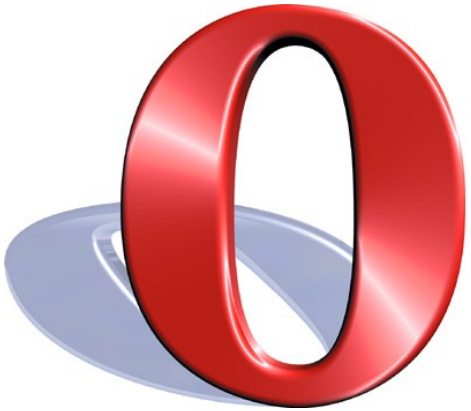
Speed Up Your DOM

- Be careful using `HTMLCollection` objects
- Perform DOM manipulations off the document
- Change CSS classes, not CSS styles
- Be careful when accessing layout information


PLEASE DO
NOT TOUCH.
TOUCHING
CAN HARM
THE ART.

Will it be like this forever?

No



Browsers With Optimizing Engines

- Chrome (V8)
- Safari 4+ (Nitro)
- Firefox 3.5+ (TraceMonkey)
- Opera 10? 11? (Carakan)

All use native code generation and JIT compiling to achieve faster JavaScript execution.


Hang in there!

Summary

- Mind your scope
- Local variables are your friends
- Function execution comes at a cost
- Keep loops small
- Avoid doing work whenever possible
- Minimize DOM interaction
- Use a good browser and encourage others to do the same


Questions?

Etcetera

- My blog: www.nczonline.net
- My email: nzakas@yahoo-inc.com
- Twitter: @slicknet

Creative Commons Images Used

- http://www.flickr.com/photos/blackbutterfly/3051019058/
- http://www.flickr.com/photos/23816315@N07/2528296337/
- http://www.flickr.com/photos/37287477@N00/515178157/
- http://www.flickr.com/photos/ottoman42/455242/
- http://www.flickr.com/photos/crumbs/2702429363/
- http://flickr.com/photos/oberazzi/318947873/