Integrating JavaScript into Native Applications

Session 615

Mark Hahnenberg
JavaScriptCore engineer

JavaScriptCore.framework

Mac

iOS

C API

?

JavaScriptCore.framework

Mac

iOS

C API	C API
Objective-C API	Objective-C API

Overview

API Goals

Automatic

Safety

Fidelity

Overview

API Goals

Automatic

Safety

Fidelity

Overview

API Goals

Automatic

Safety

Fidelity

Overview

API Goals

Automatic

Safety

Fidelity

What You Will Learn

What You Will Learn

- Objective-C → JavaScript

What You Will Learn

- Objective-C → JavaScript
- JavaScript → Objective-C

What You Will Learn

- Objective-C → JavaScript
- JavaScript → Objective-C
- Memory management

What You Will Learn

- Objective-C → JavaScript
- JavaScript → Objective-C
- Memory management
- Threading

What You Will Learn

- Objective-C → JavaScript
- JavaScript → Objective-C
- Memory management
- Threading
- JavaScriptCore C API

What You Will Learn

- Objective-C → JavaScript
- JavaScript → Objective-C
- Memory management
- Threading
- JavaScriptCore C API
- JavaScriptCore with a WebView

Demo

ColorMyWords application

Objective-C → JavaScript

Evaluating JavaScript Code

```
#import <JavaScriptCore/JavaScriptCore.h>

int main () {
    JSContext *context = [[JSContext alloc] init];

    JSValue *result = [context evaluateScript:@"2 + 2"];
    NSLog(@"2 + 2 = %d", [result toInt32]);
    return 0;
}
```

Evaluating JavaScript Code

```
#import <JavaScriptCore/JavaScriptCore.h>

int main () {
    JSContext *context = [[JSContext alloc] init];

    JSValue *result = [context evaluateScript:@"2 + 2"];

    NSLog(@"2 + 2 = %d", [result toInt32]);
    return 0;
}
```

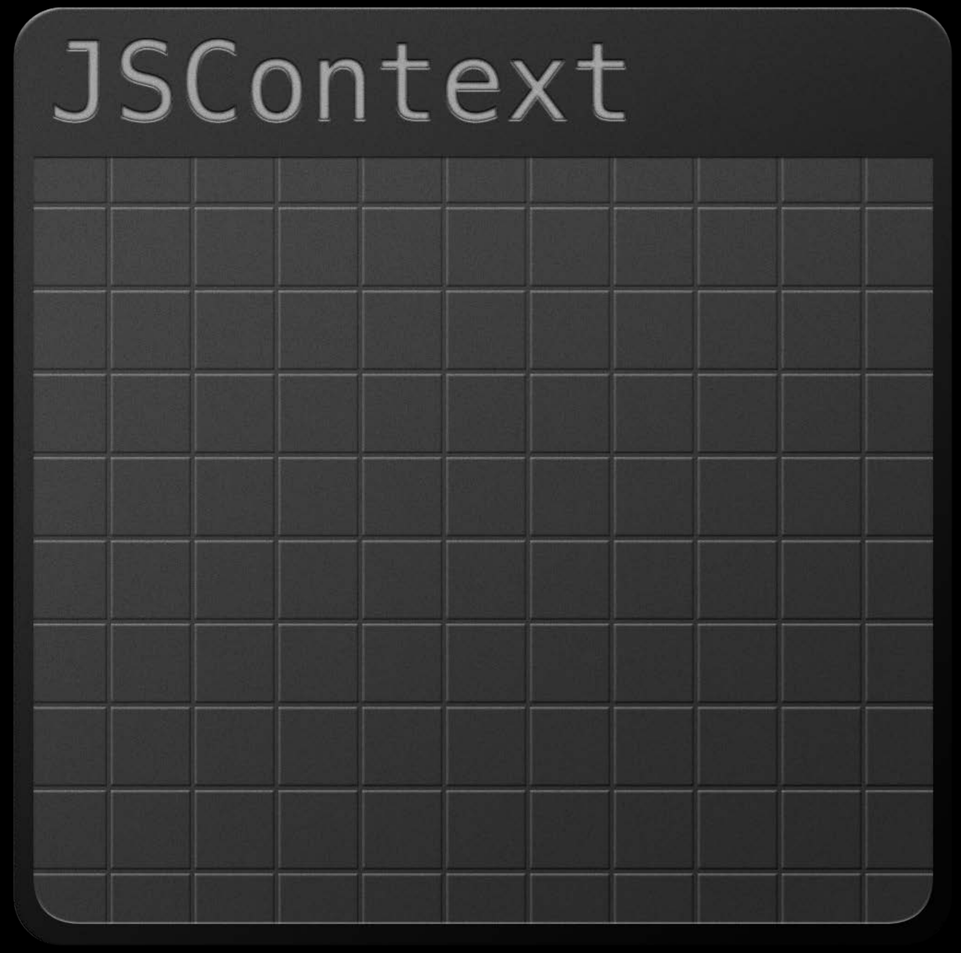
Evaluating JavaScript Code

```
#import <JavaScriptCore/JavaScriptCore.h>

int main () {
    JSContext *context = [[JSContext alloc] init];

    JSValue *result = [context evaluateScript:@"2 + 2"];

    NSLog(@"2 + 2 = %d", [result toInt32]);
    return 0;
}
```

Evaluating JavaScript Code

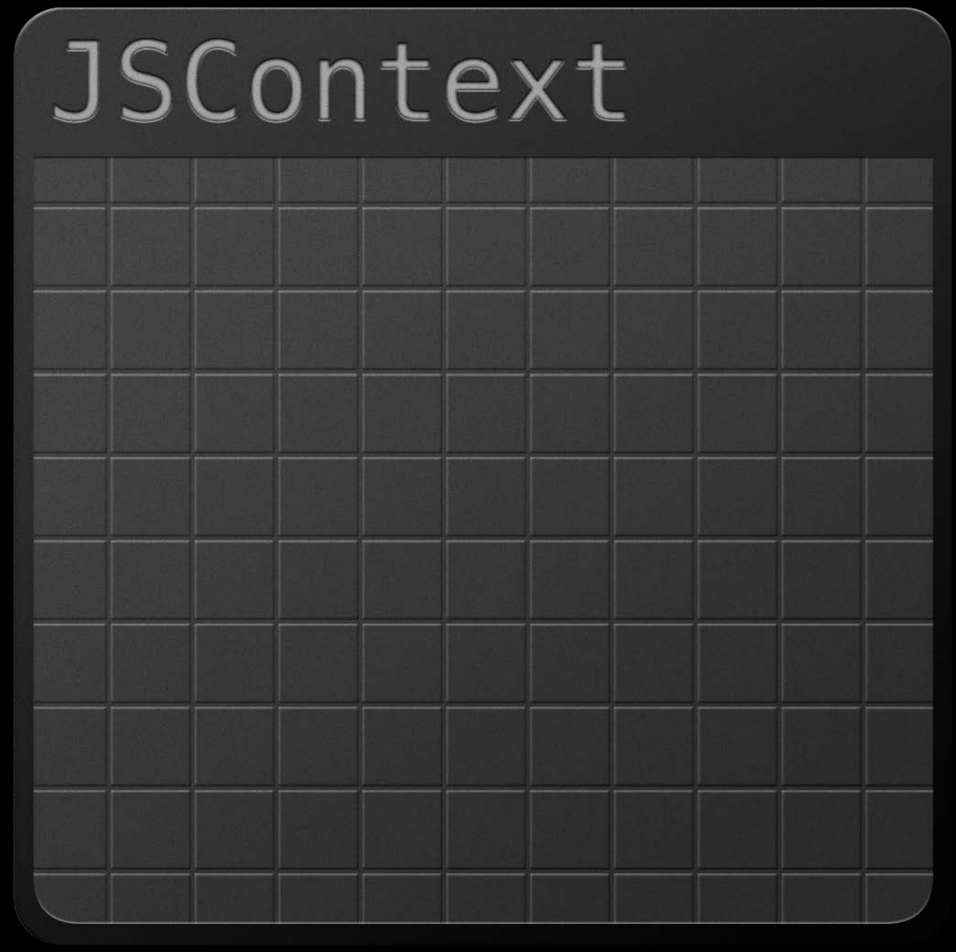
```
#import <JavaScriptCore/JavaScriptCore.h>

int main () {
    JSContext *context = [[JSContext alloc] init];

    JSValue *result = [context evaluateScript:@"2 + 2"];
    NSLog(@"2 + 2 = %d", [result toInt32]);
    return 0;
}
```

JSContext



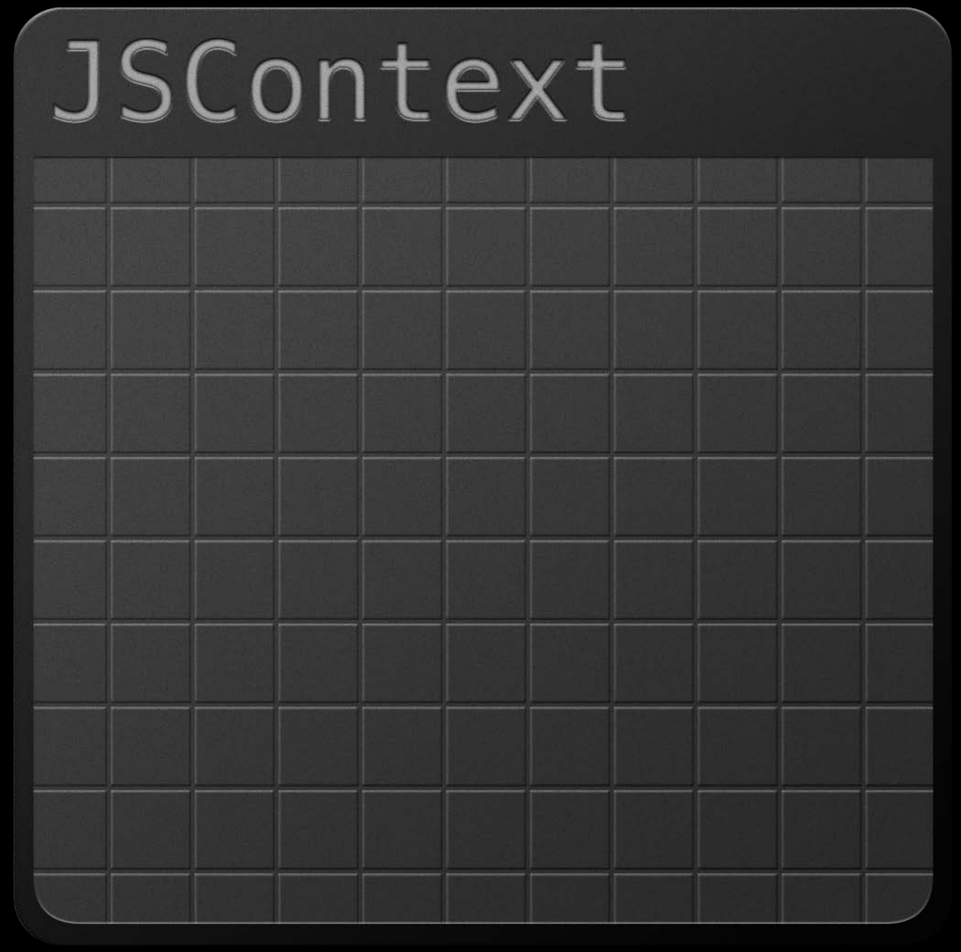
JSContext

- Context for evaluating code

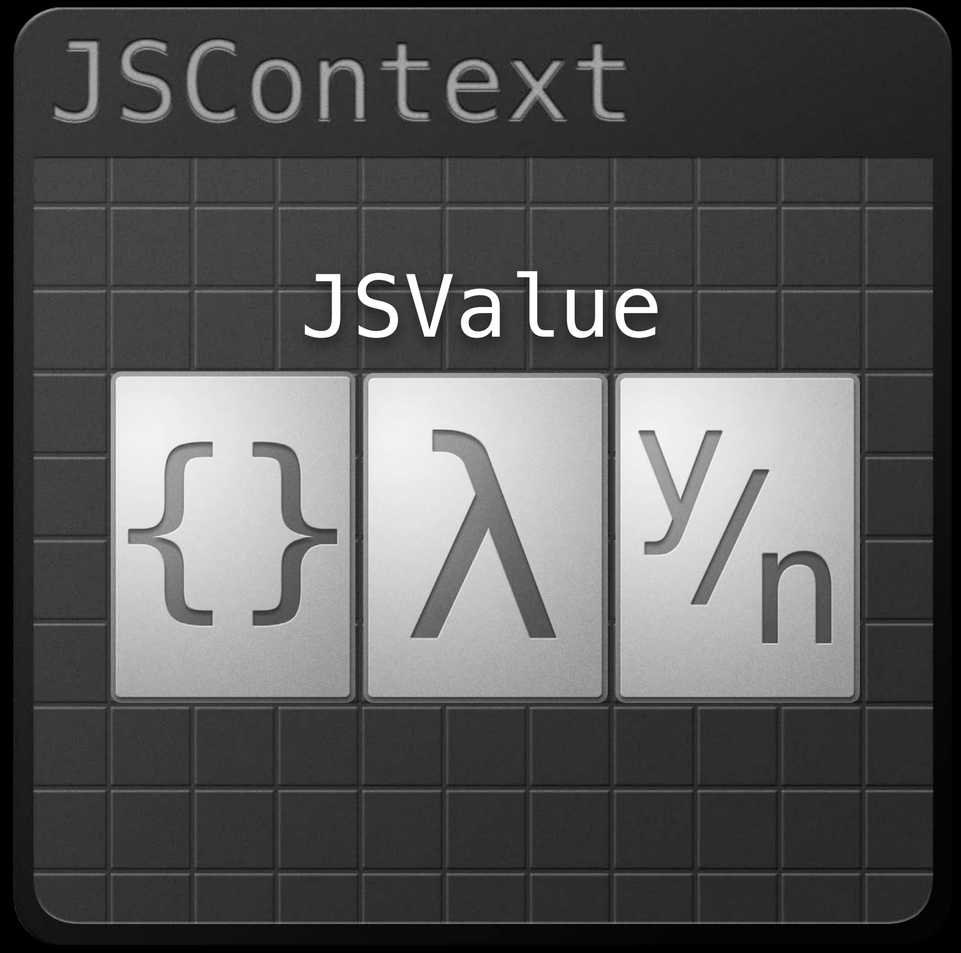


JSContext

- Context for evaluating code
- Global object

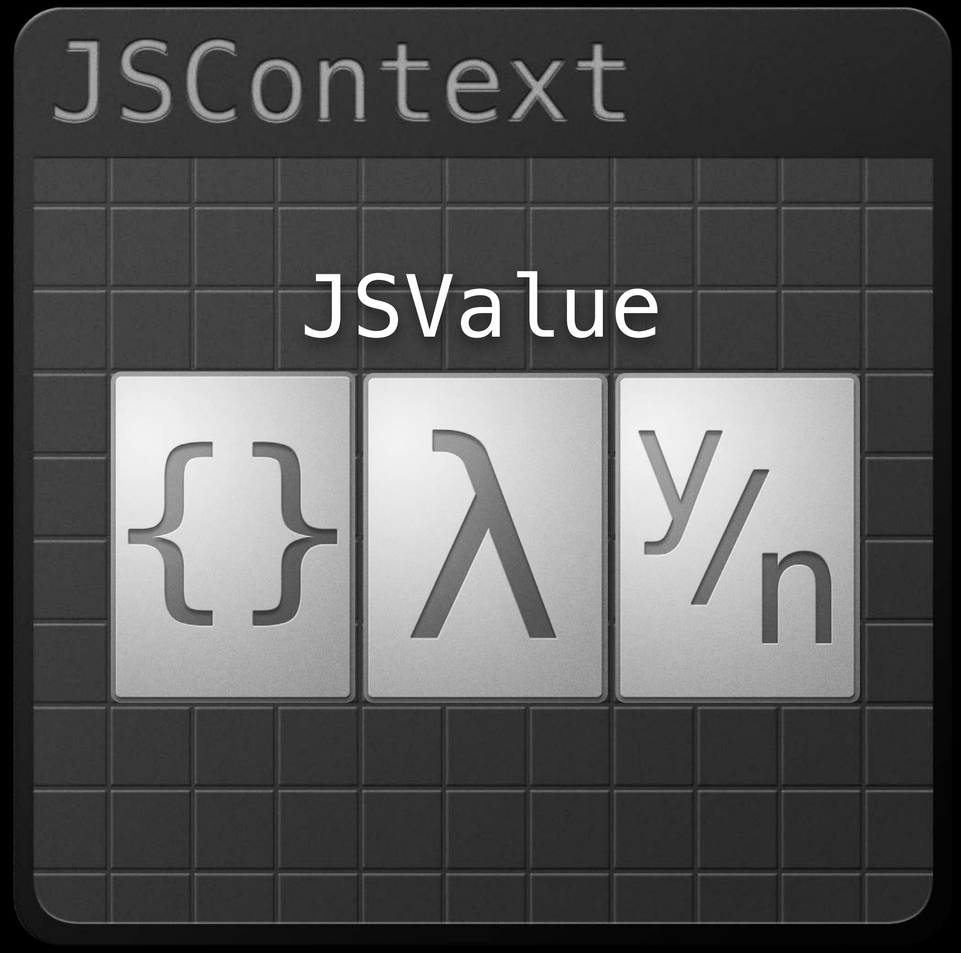


JSValue



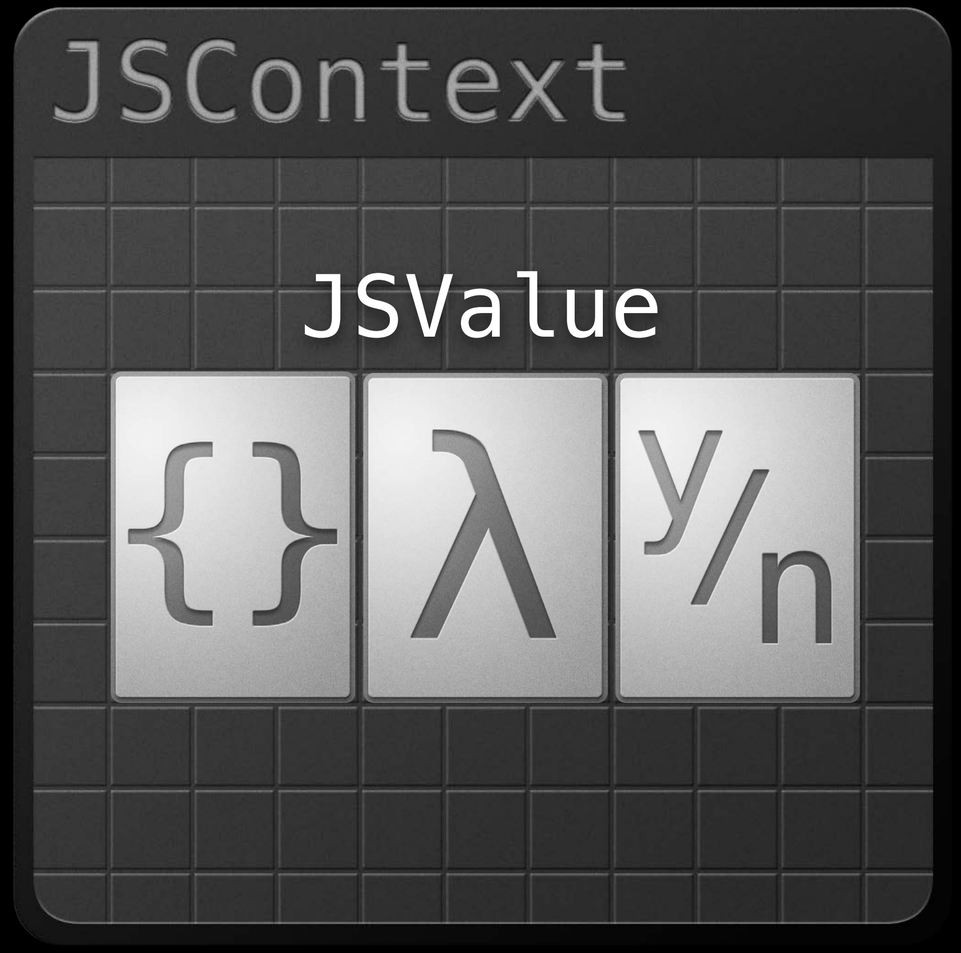
JSValue

- Reference to JavaScript value



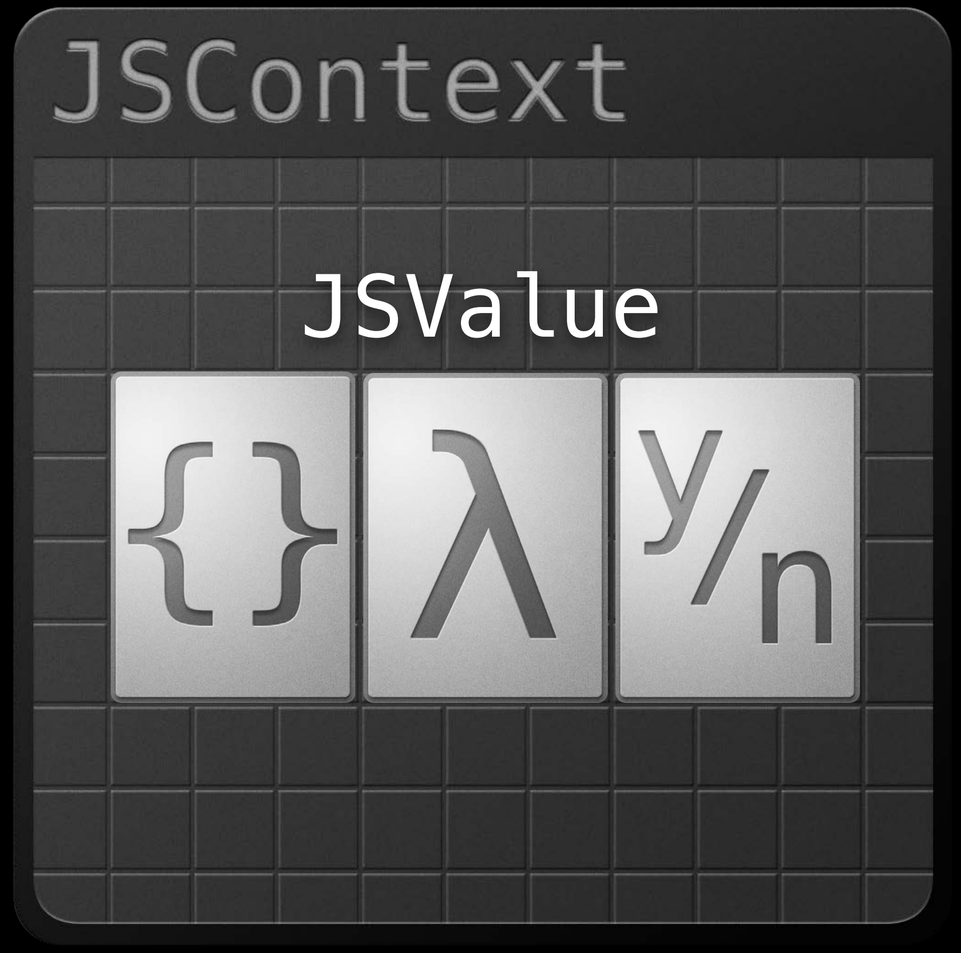
JSValue

- Reference to JavaScript value
  - Strong reference



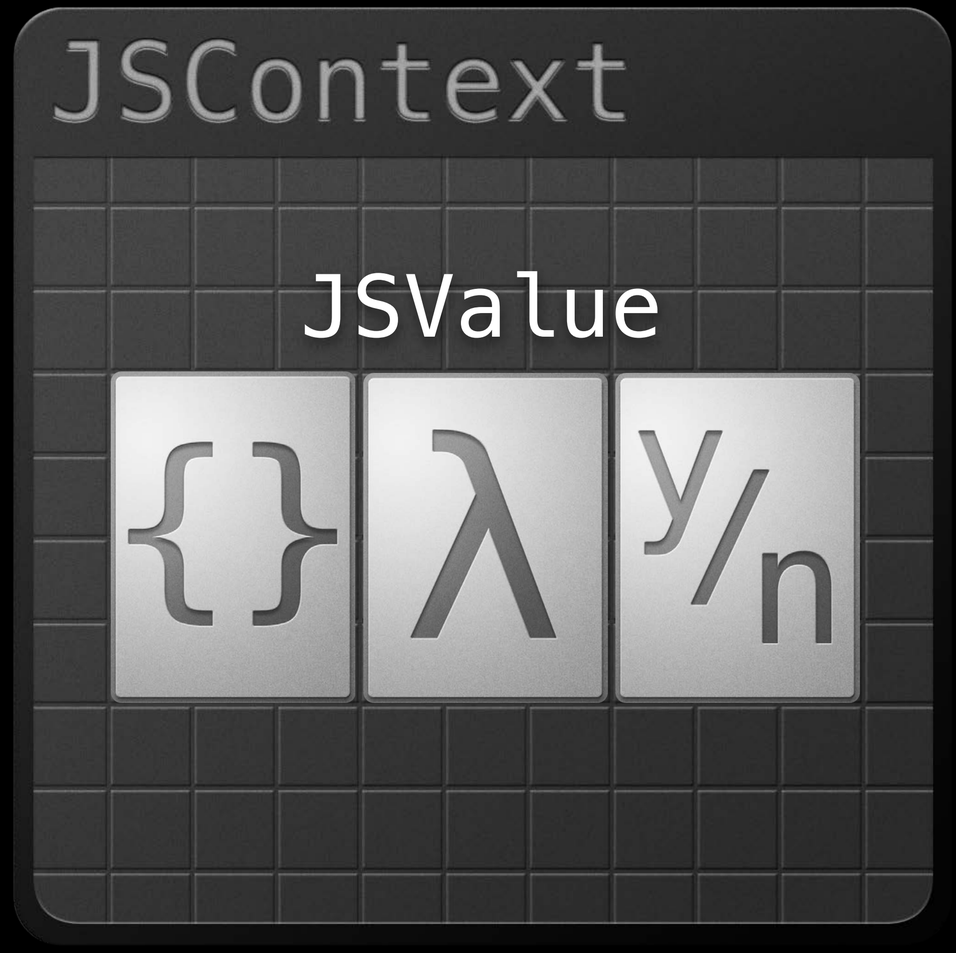
JSValue

- Reference to JavaScript value
  - Strong reference
- Tied to a JSContext



JSValue

- Reference to JavaScript value
  - Strong reference
- Tied to a JSContext
  - Strong reference



Creating JSValues

```
+ valueWithBool:(BOOL)value inContext:(JSContext *)context;
+ valueWithDouble:(double)value inContext:(JSContext *)context;
+ valueWithInt32:(int32_t)value inContext:(JSContext *)context;
+ valueWithUInt32:(uint32_t)value inContext:(JSContext *)context;
+ valueWithNullInContext:(JSContext *)context;
+ valueWithUndefinedInContext:(JSContext *)context;
+ valueWithNewObjectInContext:(JSContext *)context;
+ valueWithNewArrayInContext:(JSContext *)context;
+ valueWithNewRegularExpressionFromPattern:
    (NSString *)pattern
    flags:(NSString *)flags
    inContext:(JSContext *)context;
+ valueWithNewErrorFromMessage:
    (NSString *)message
    inContext:(JSContext *)context;
```

Creating JSValues

```
+ valueWithBool:(BOOL)value inContext:(JSContext *)context;
+ valueWithDouble:(double)value inContext:(JSContext *)context;
+ valueWithInt32:(int32_t)value inContext:(JSContext *)context;
+ valueWithUInt32:(uint32_t)value inContext:(JSContext *)context;
+ valueWithNullInContext:(JSContext *)context;
+ valueWithUndefinedInContext:(JSContext *)context;
```

```
+ valueWithNewObjectInContext:(JSContext *)context;
+ valueWithNewArrayInContext:(JSContext *)context;
+ valueWithNewRegularExpressionFromPattern:
    (NSString *)pattern
    flags:(NSString *)flags
    inContext:(JSContext *)context;
+ valueWithNewErrorFromMessage:
    (NSString *)message
    inContext:(JSContext *)context;
```

Creating JSValues

Bridging

```
+ valueWithObject:(id)value inContext:(JSContext *)context;
```

Accessing JavaScript Values

```
- (BOOL)toBool;
- (double)toDouble;
- (int32_t)toInt32;
- (uint32_t)toUInt32;
- (NSNumber *)toNumber;
- (NSString *)toString;
- (NSDate *)toDate;
- (NSArray *)toArray;
- (NSDictionary *)toDictionary;

- (id)toObject;
- (id)toObjectOfClass:(Class)expectedClass;
```

Calling JavaScript Functions

```
var factorial = function(n) {
     if (n < 0)
        return;
     if (n === 0)
        return 1;
     return n * factorial(n - 1)
};
```

Calling JavaScript Functions

```
NSString *factorialScript = loadFactorialScript();
[_context evaluateScript:factorialScript];

JSValue *function = _context[@"factorial"];

JSValue *result = [function callWithArguments:@[@5]];

NSLog(@"factorial(5) = %d", [result toInt32]);
```

Calling JavaScript Functions

```
NSString *factorialScript = loadFactorialScript();
[_context evaluateScript:factorialScript];

JSValue *function = _context[@"factorial"];

JSValue *result = [function callWithArguments:@[@5]];

NSLog(@"factorial(5) = %d", [result toInt32]);
```

Calling JavaScript Functions

```
NSString *factorialScript = loadFactorialScript();
[_context evaluateScript:factorialScript];

JSValue *function = _context[@"factorial"];

JSValue *result = [function callWithArguments:@[@5]];

NSLog(@"factorial(5) = %d", [result toInt32]);
```

Calling JavaScript Functions

```
NSString *factorialScript = loadFactorialScript();
[_context evaluateScript:factorialScript];

JSValue *function = _context[@"factorial"];

JSValue *result = [function callWithArguments:@[@5]];
```

```
NSLog(@"factorial(5) = %d", [result toInt32]);
```

Demo

ColorMyWords recap

JavaScript → Objective-C

JavaScript → Objective-C

JavaScript → Objective-C

- Two ways to interact with Objective-C from JavaScript

JavaScript → Objective-C

- Two ways to interact with Objective-C from JavaScript
  - Blocks

JavaScript → Objective-C

- Two ways to interact with Objective-C from JavaScript
  - Blocks
    - JS functions

JavaScript → Objective-C

- Two ways to interact with Objective-C from JavaScript
  - Blocks
    - JS functions
  - JSExport protocol

JavaScript → Objective-C

- Two ways to interact with Objective-C from JavaScript
  - Blocks
    - JS functions
  - JSExport protocol
    - JS objects

Blocks

Blocks

Blocks

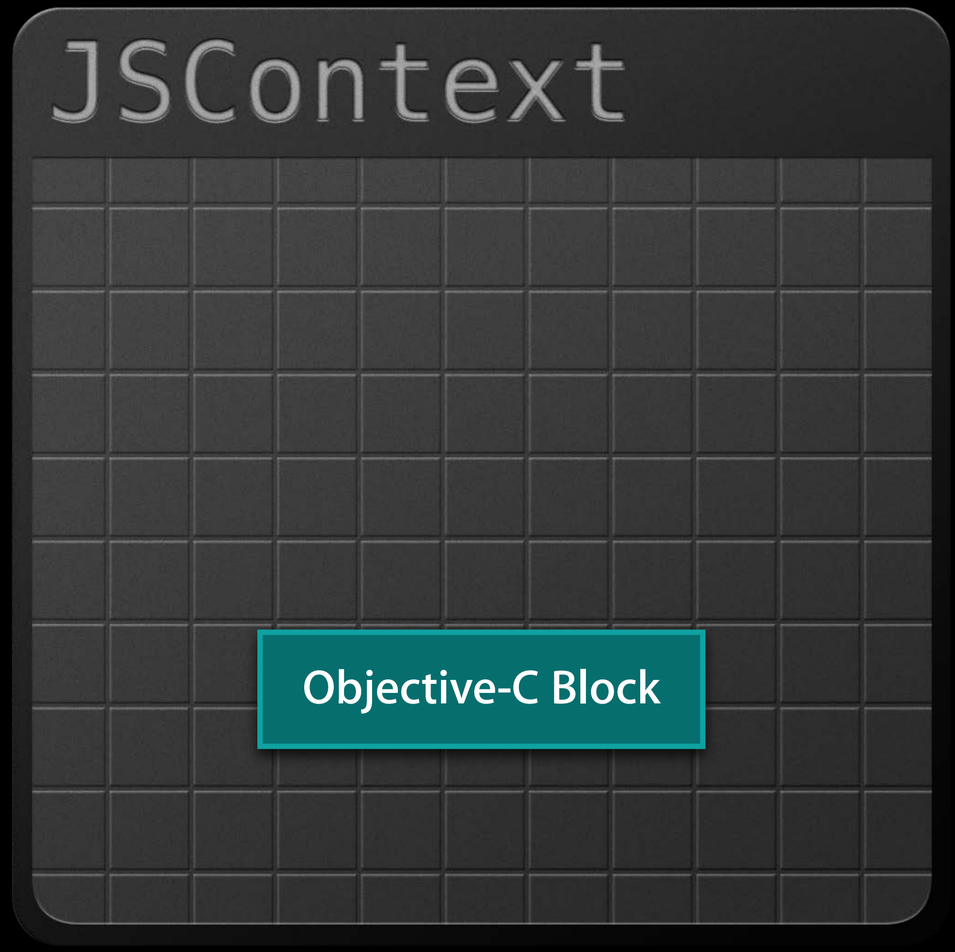
- Easy way to expose Objective-C code to JavaScript

Blocks

- Easy way to expose Objective-C code to JavaScript
- Automatically wraps Objective-C block inside callable JavaScript function

Blocks

```
context[@"makeNSColor"] = ^(NSDictionary *rgb){
    float r = [colors[@"red"] floatValue];
    float g = [colors[@"green"] floatValue];
    float b = [colors[@"blue"] floatValue];
    return [NSColor colorWithRed:(r / 255.0)
        green:(g / 255.0f)
        blue:(b / 255.0f)
        alpha:1.0];
};
```

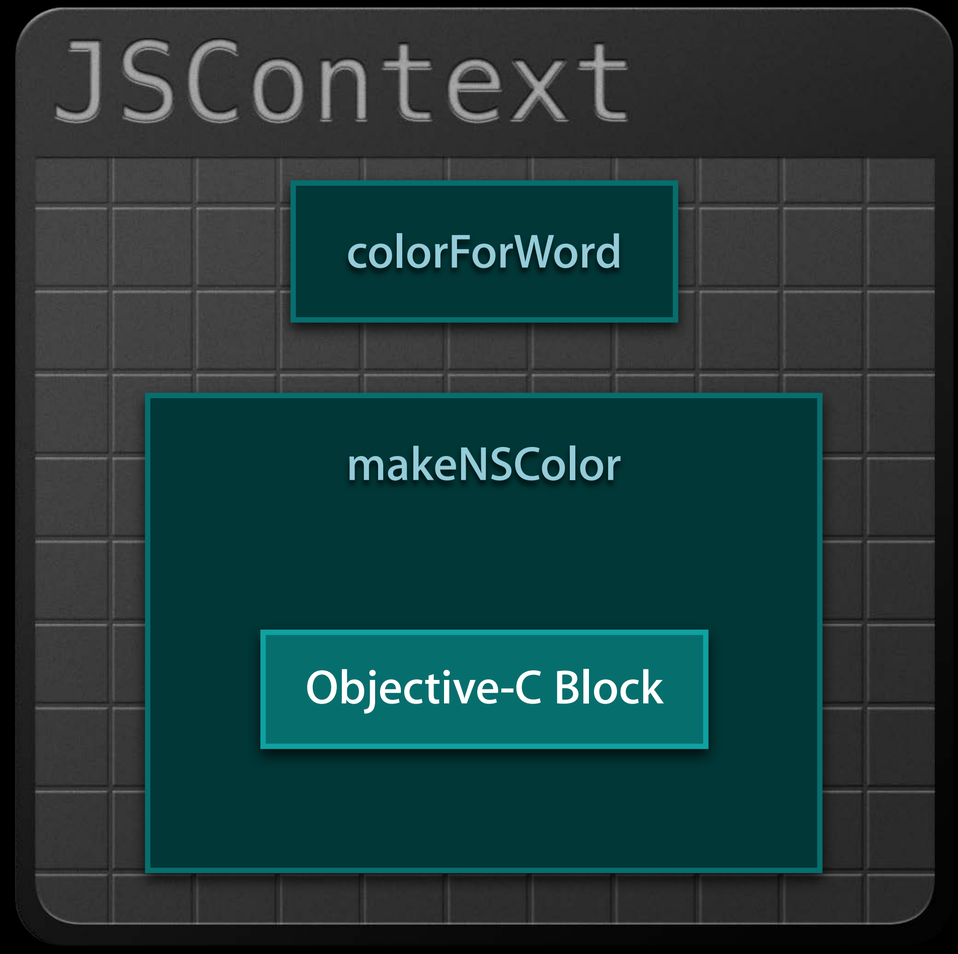


Blocks

```
context[@"makeNSColor"] = ^(NSDictionary *rgb){
    float r = [colors[@"red"] floatValue];
    float g = [colors[@"green"] floatValue];
    float b = [colors[@"blue"] floatValue];
    return [NSColor colorWithRed:(r / 255.0)
        green:(g / 255.0f)
        blue:(b / 255.0f)
        alpha:1.0];
};
```

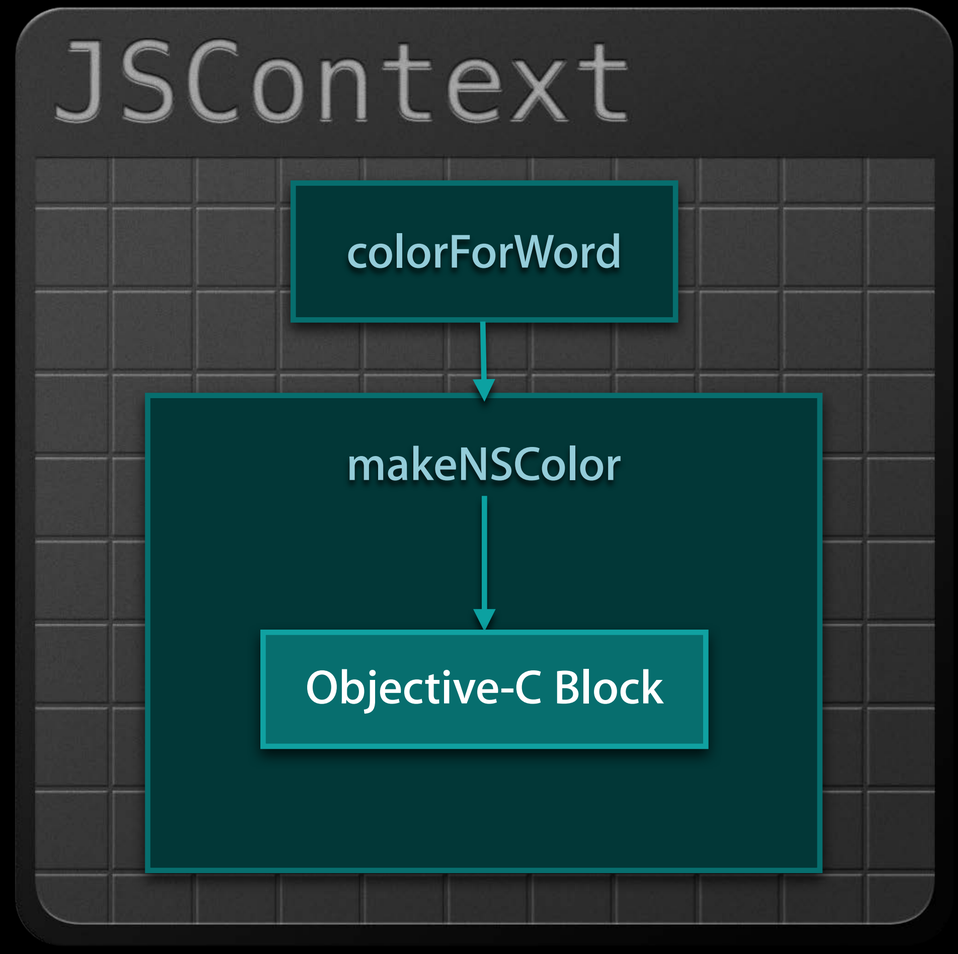
Blocks

```
var colorForWord = function(word) {
    if (!colorMap[word])
        return;
    return makeNSColor(colorMap[word]);
};
```



Blocks

```
var colorForWord = function(word) {
    if (!colorMap[word])
        return;
    return makeNSColor(colorMap[word]);
};
```



Blocks

Caveats

Blocks

Caveats

- Avoid capturing JSValues

Blocks

Caveats

- Avoid capturing JSValues
  - Prefer passing as arguments

Blocks

Caveats

- Avoid capturing JSValues
  - Prefer passing as arguments
- Avoid capturing JSContexts

Blocks

Caveats

- Avoid capturing JSValues
  - Prefer passing as arguments
- Avoid capturing JSContexts
  - Use + [JSContext currentContext]

Blocks

Bad

```
JSContext *context = [[JSContext alloc] init];

context[@"callback"] = ^{
     JSValue *object = [JSValue valueWithNewObjectInContext:context];
     object[@"x"] = 2;
     object[@"y"] = 3;
     return object;
};
```

Blocks

Good

```
JSContext *context = [[JSContext alloc] init];

context[@"callback"] = ^{
     JSValue *object = [JSValue valueWithNewObjectInContext:
        [JSContext currentContext]];
     object[@"x"] = 2;
     object[@"y"] = 3;
     return object;
};
```

JSExport

JSExport

- Easy way for JavaScript to interact with Objective-C objects

JSExport

Objective-C	JavaScript

`@interface MyPoint`
 	
`@property double x;`
`@property double y;`
 	
`- (NSString *)description;`
 	
`+ (MyPoint *)makePointWithX:(double)x`
`y:(double)y;`
 	
`@end`	

JSExport

Objective-C	JavaScript

`@interface MyPoint`
 	
`@property double x;`
`@property double y;`
 	
`- (NSString *)description;`
 	
`+ (MyPoint *)makePointWithX:(double)x`
`y:(double)y;`
 	
`@end`
	`point.x;`
	`point.x = 10;`

JSExport

Objective-C	JavaScript

`@interface MyPoint`
 	
`@property double x;`
`@property double y;`
 	
`- (NSString *)description;`
 	
`+ (MyPoint *)makePointWithX:(double)x`
`y:(double)y;`
 	
`@end`
	`point.x;`
	`point.x = 10;`
	
	`point.description();`

JSExport

Objective-C	JavaScript

`@interface MyPoint`
 	
`@property double x;`
`@property double y;`
 	
`- (NSString *)description;`
 	
`+ (MyPoint *)makePointWithX:(double)x`
`y:(double)y;`
 	
`@end`
	`point.x;`
	`point.x = 10;`
	
	`point.description();`
	
	`MyPoint.makePointWithXY(0, 0);`

JSExport

Objective-C	JavaScript

`@protocol MyPointExports <JSExport>`
 	
`@property double x;`
`@property double y;`
 	
`- (NSString *)description;`
 	
`+ (MyPoint *)makePointWithX:(double)x`
`y:(double)y;`
 	
`@end`
	`point.x;`
	`point.x = 10;`
	
	`point.description();`
	
	`MyPoint.makePointWithXY(0, 0);`

JSExport

```
@interface MyPoint : NSObject <MyPointExports>

- (void)myPrivateMethod;  // Not visible to JavaScript code.

@end
```

```
@implementation MyPoint

...

@end
```

JSExport

JSExport

- Enumeration of methods and properties to export to JavaScript

JSExport

- Enumeration of methods and properties to export to JavaScript
- @property → JavaScript getter/setter

JSExport

- Enumeration of methods and properties to export to JavaScript
- @property → JavaScript getter/setter
- Instance method → JavaScript function

JSExport

- Enumeration of methods and properties to export to JavaScript
- @property → JavaScript getter/setter
- Instance method → JavaScript function
- Class methods → JavaScript functions on global class object

JSExport

```
JSContext *context = [[JSContext alloc] init];
[context evaluateScript:geometryScript];

MyPoint *point1 = [[MyPoint alloc] initWithX:0.0 y:0.0];
MyPoint *point2 = [[MyPoint alloc] initWithX:1.0 y:1.0];

JSValue *function = context[@"euclideanDistance"];
JSValue *result = [function callWithArguments:@[point1, point2]];
```

JSExport

```
JSContext *context = [[JSContext alloc] init];
[context evaluateScript:geometryScript];

MyPoint *point1 = [[MyPoint alloc] initWithX:0.0 y:0.0];
MyPoint *point2 = [[MyPoint alloc] initWithX:1.0 y:1.0];

JSValue *function = context[@"euclideanDistance"];
JSValue *result = [function callWithArguments:@[point1, point2]];
```

JSExport

```
JSContext *context = [[JSContext alloc] init];
[context evaluateScript:geometryScript];

MyPoint *point1 = [[MyPoint alloc] initWithX:0.0 y:0.0];
MyPoint *point2 = [[MyPoint alloc] initWithX:1.0 y:1.0];
```

```
JSValue *function = context[@"euclideanDistance"];
JSValue *result = [function callWithArguments:@[point1, point2]];
```

JSExport

```
context[@"MyPoint"] = [MyPoint class];
```

```
JSValue *function = context[@"midpoint"];
JSValue *jsResult = [function callWithArguments:@[point1, point2]];
MyPoint *midpoint = [jsResult toObject];
```

JSExport

```
context[@"MyPoint"] = [MyPoint class];
```

```
JSValue *function = context[@"midpoint"];
JSValue *jsResult = [function callWithArguments:@[point1, point2]];
MyPoint *midpoint = [jsResult toObject];
```

JSExport

```
// geometry.js

var euclideanDistance = function(p1, p2) {
     var xDelta = p2.x - p1.x;
     var yDelta = p2.y - p1.y;
     return Math.sqrt(xDelta * xDelta + yDelta * yDelta);
};

var midpoint = function(p1, p2) {
     var xDelta = (p2.x - p1.x) / 2;
     var yDelta = (p2.y - p1.y) / 2;
     return MyPoint.makePointWithXY(p1.x + xDelta, p1.y + yDelta);
};
```

Advanced API Topics

Memory Management

Memory Management

Memory Management

- Objective-C uses ARC

Memory Management

- Objective-C uses ARC
- JavaScriptCore uses garbage collection

Memory Management

- Objective-C uses ARC
- JavaScriptCore uses garbage collection
  - All references are strong

Memory Management

- Objective-C uses ARC
- JavaScriptCore uses garbage collection
  - All references are strong
- API memory management is mostly automatic

Memory Management

- Objective-C uses ARC
- JavaScriptCore uses garbage collection
  - All references are strong
- API memory management is mostly automatic
- Two situations that require extra attention:

Memory Management

- Objective-C uses ARC
- JavaScriptCore uses garbage collection
  - All references are strong
- API memory management is mostly automatic
- Two situations that require extra attention:
  - Storing JavaScript values in Objective-C objects

Memory Management

- Objective-C uses ARC
- JavaScriptCore uses garbage collection
  - All references are strong
- API memory management is mostly automatic
- Two situations that require extra attention:
  - Storing JavaScript values in Objective-C objects
  - Adding JavaScript fields to Objective-C objects

Retain Cycles

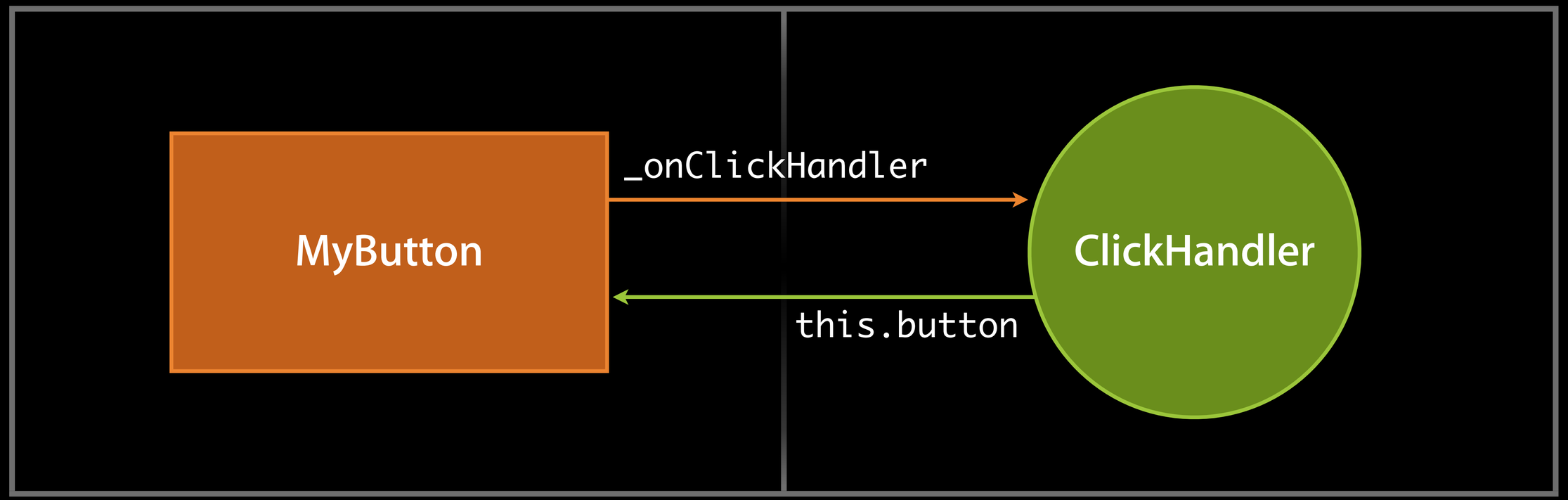
```
function ClickHandler(button, callback) {
     this.button = button;
     this.button.onClickHandler = this;
     this.handleEvent = callback;
};
```

JSManagedValue

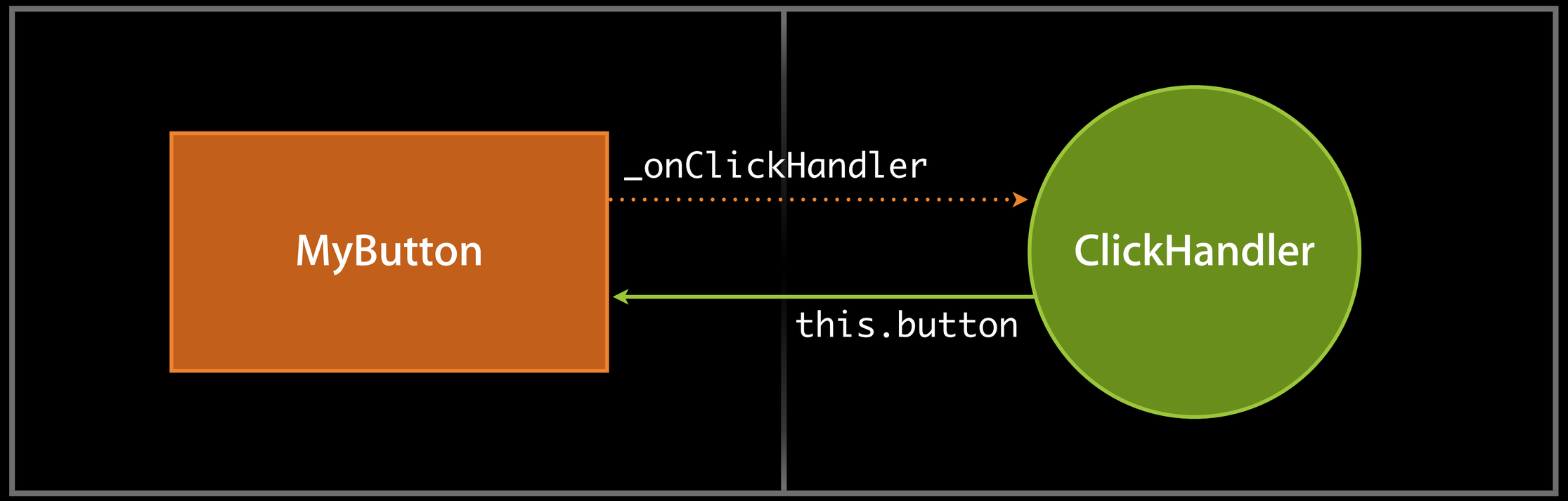
Bad

```
@implementation MyButton

- (void)setOnClickHandler:(JSValue *)handler
{
    _onClickHandler = handler; // Retain cycle
}

@end
```

Retain Cycles



Retain Cycles



JSManagedValue

Good

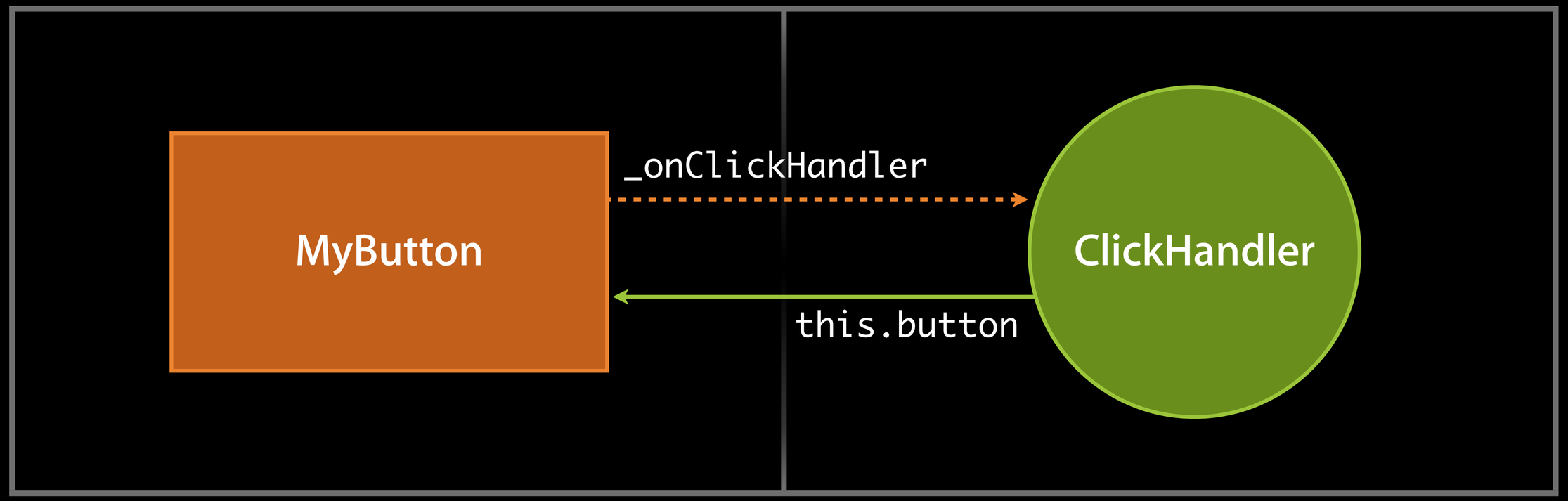
```
@implementation MyButton

- (void)setOnClickHandler:(JSValue *)handler
{
    _onClickHandler = [JSManagedValue managedValueWithValue:handler];
    [_context.virtualMachine addManagedReference:_onClickHandler
                                       withOwner:self]
}

@end
```

JSManagedValue

Good

```
@implementation MyButton

- (void)setOnClickHandler:(JSValue *)handler
{
    _onClickHandler = [JSManagedValue managedValueWithValue:handler];
    [_context.virtualMachine addManagedReference:_onClickHandler
                                       withOwner:self]
}

@end
```

JSManagedValue



Strong

Weak

Garbage collected

Managed References

Managed References

- JSManagedValue by itself is a weak reference to a JavaScript value

Managed References

- JSManagedValue by itself is a weak reference to a JavaScript value
- `-addManagedReference:withOwner:` turns JSManagedValue into a "garbage collected" reference

Managed References

Managed References

- If JavaScript can find the Objective-C owner of a managed reference
  - Reference is kept alive

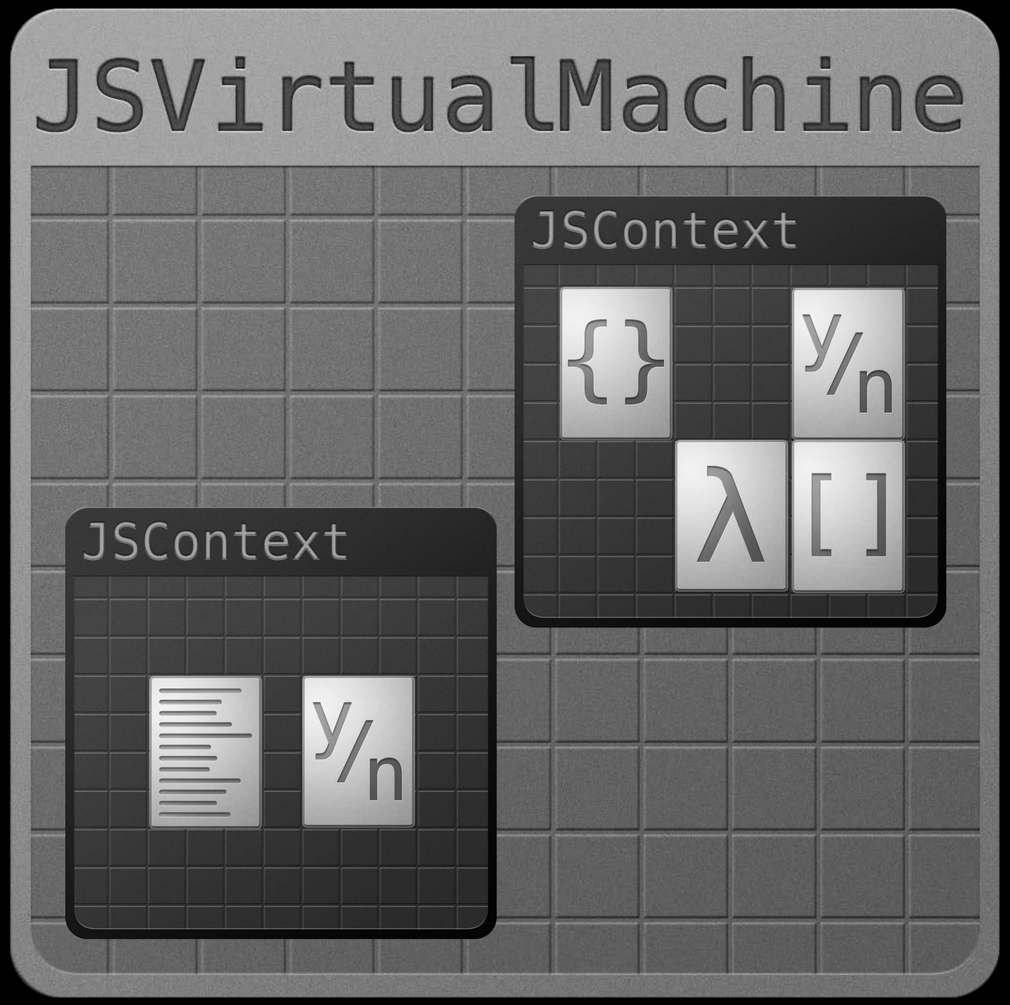
Managed References

- If JavaScript can find the Objective-C owner of a managed reference
  - Reference is kept alive
- Otherwise
  - Reference is released

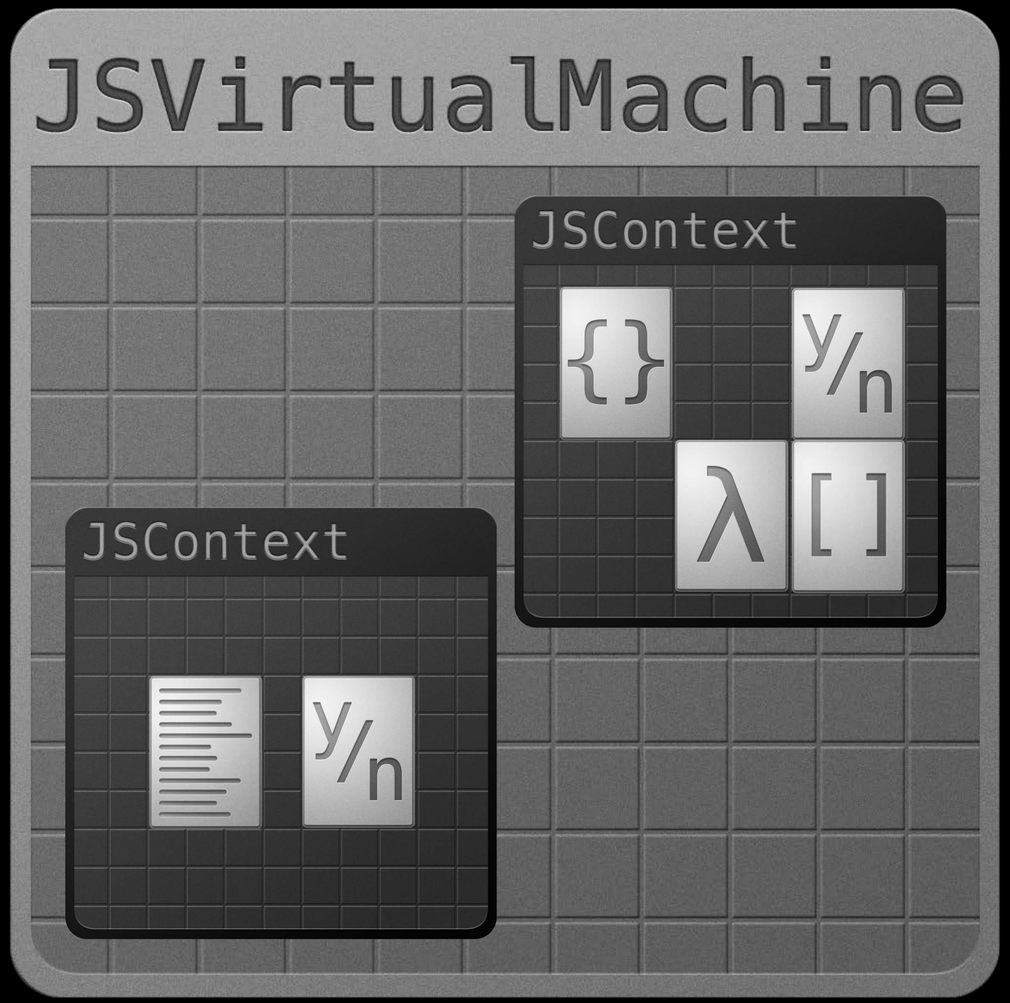
Threading

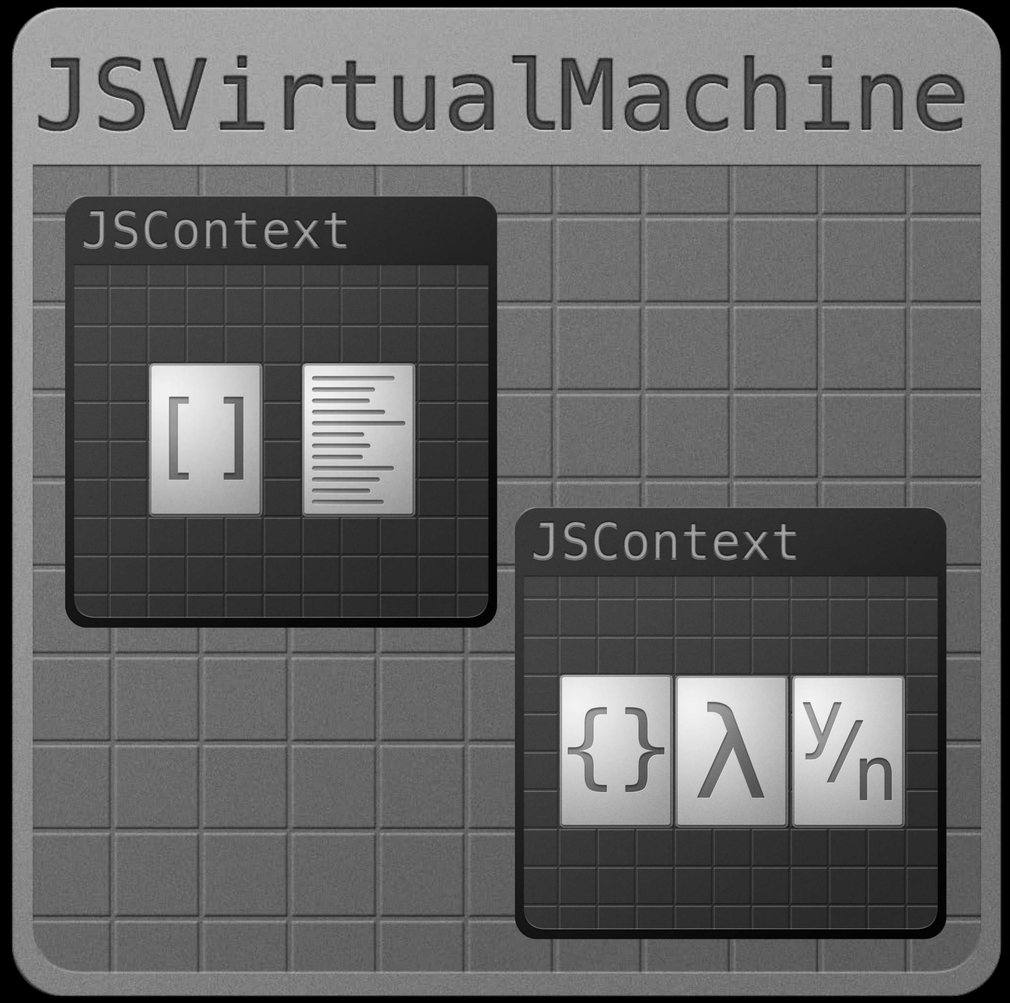
JSVirtualMachine

JSVirtualMachine

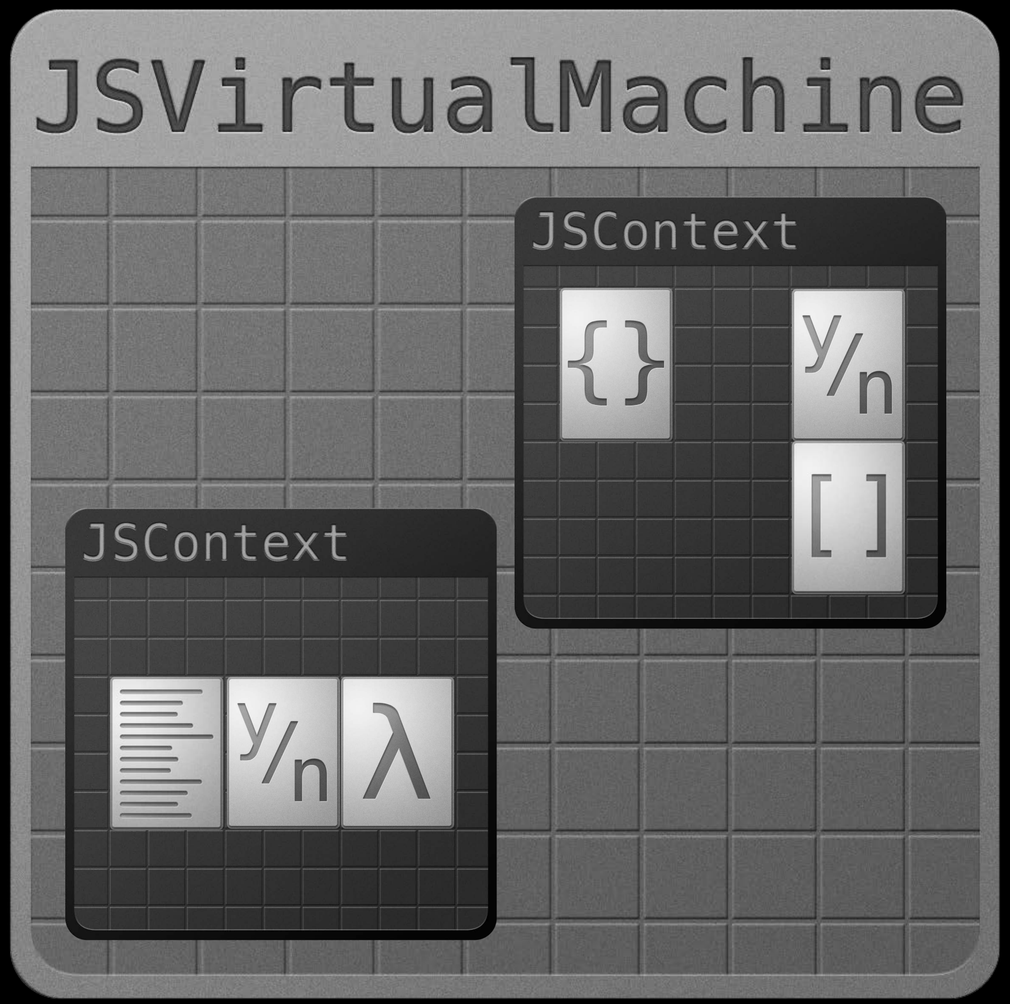


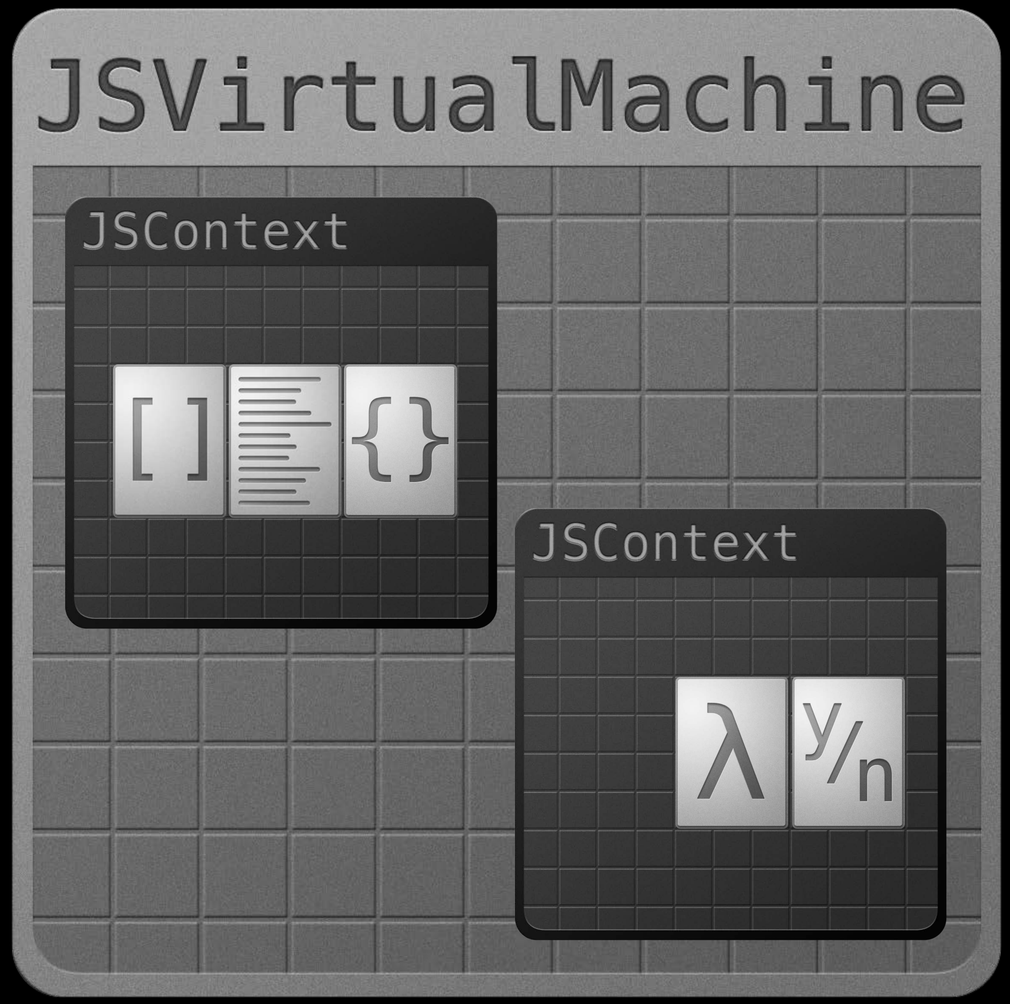
JSVirtualMachine



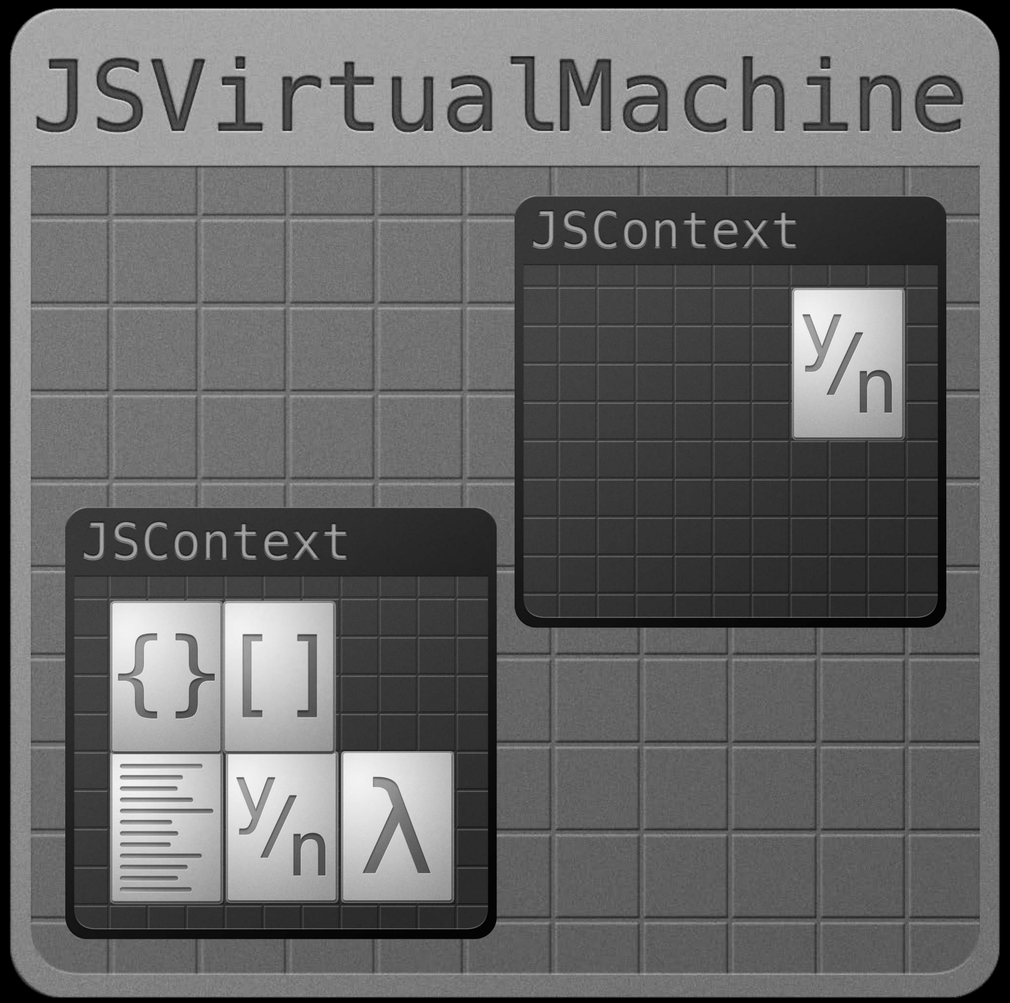


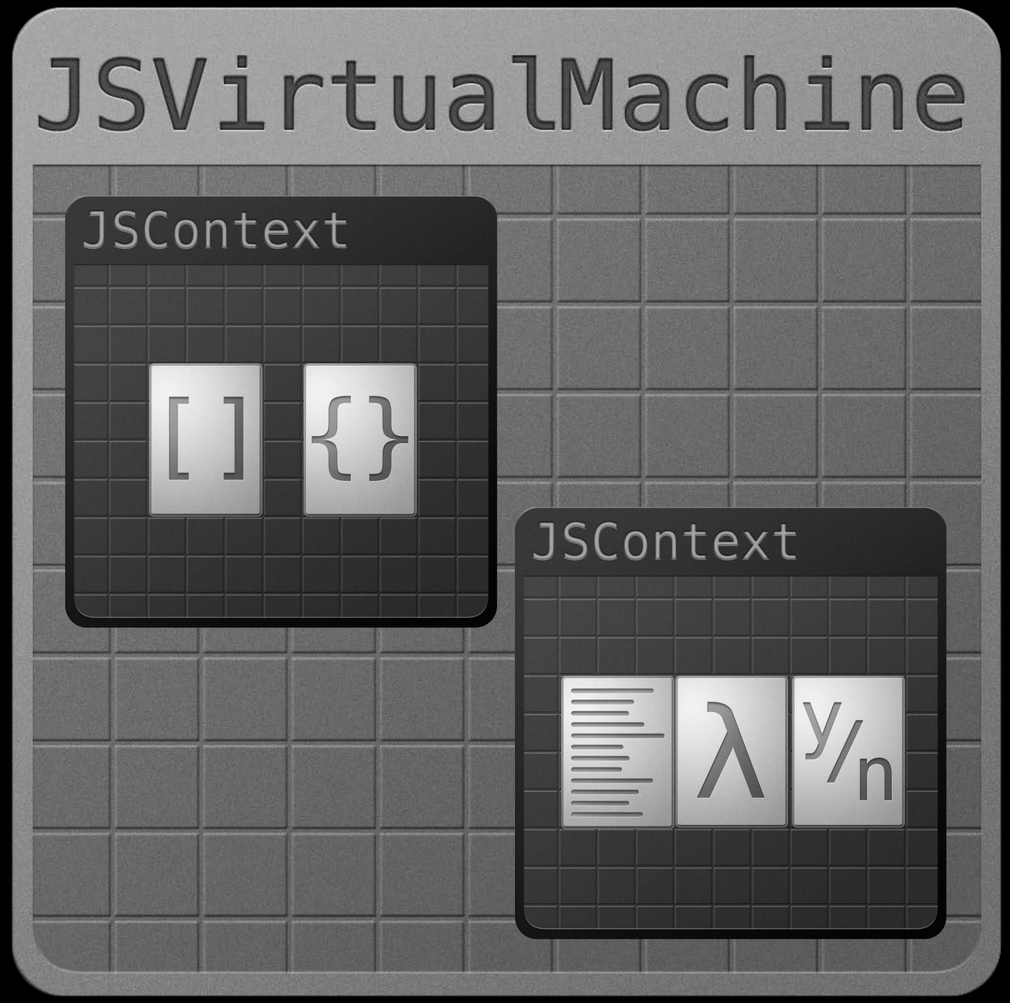
JSVirtualMachine



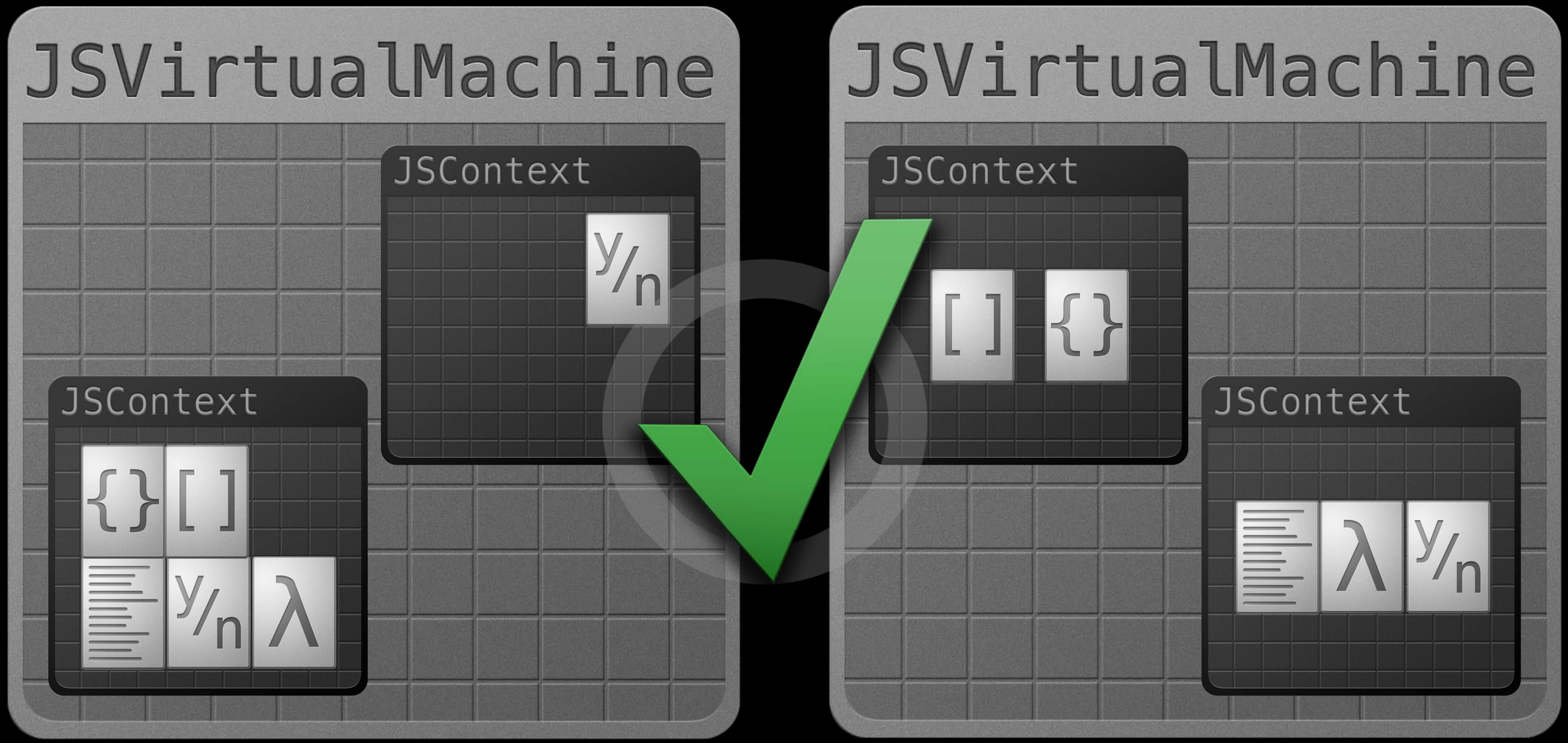


JSVirtualMachine

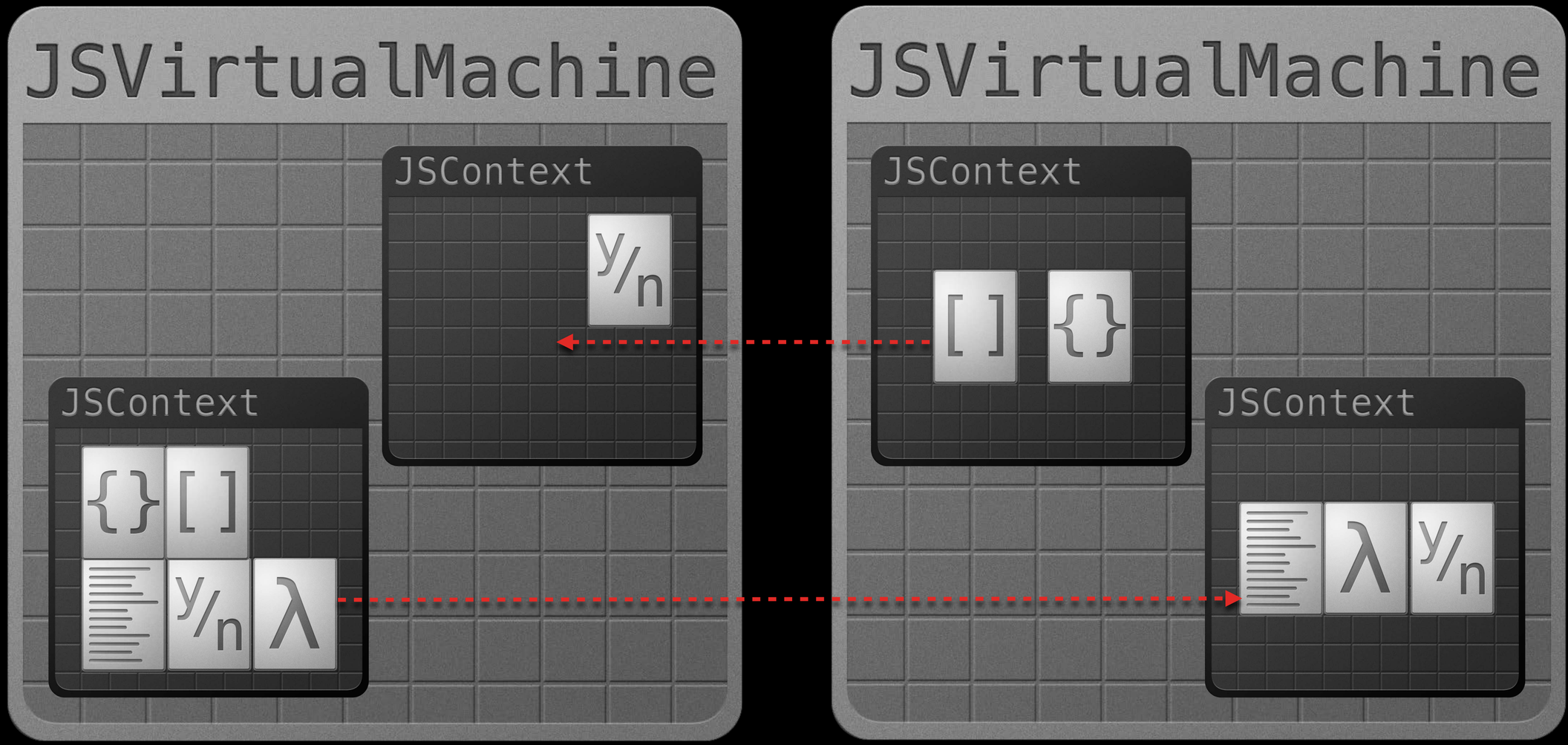




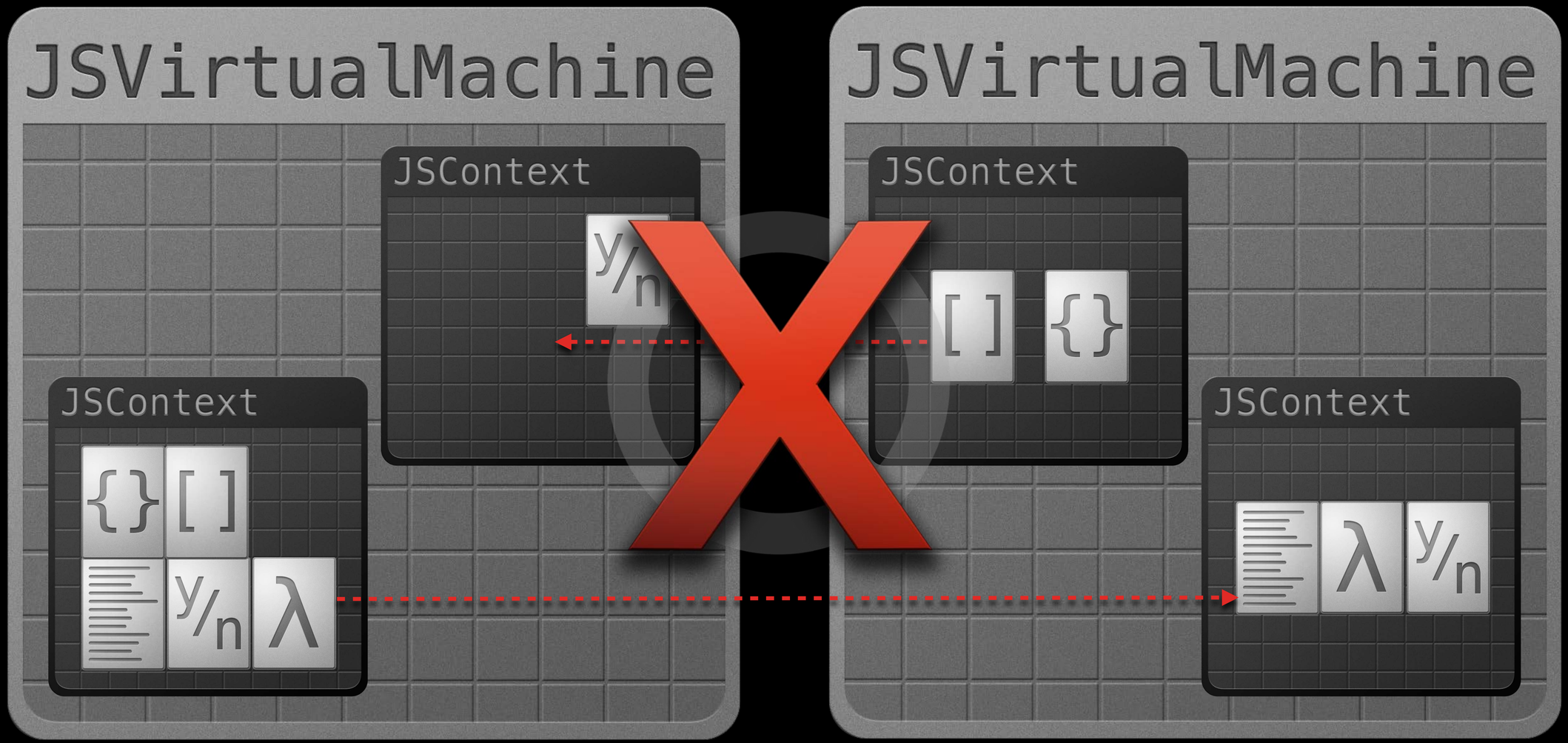
JSVirtualMachine



JSVirtualMachine



JSVirtualMachine



Threading

Threading

- API is thread safe

Threading

- API is thread safe
- Locking granularity is JSVirtualMachine

Threading

- API is thread safe
- Locking granularity is JSVirtualMachine
  - Use separate JSVirtualMachines for concurrency/parallelism

Interfacing with the JavaScriptCore C API

Interfacing with the C API

- JSValue ↔ JSValueRef
- JSContext ↔ JSGlobalContextRef

JSContext ↔ JSGlobalContextRef

```
JSGlobalContextRef ctx = ...
JSContext *context = [JSContext contextWithJSGlobalContextRef:ctx];

JSContext *context = ...
JSGlobalContextRef ctx = [context JSGlobalContextRef];
```

JSValue ↔ JSValueRef

```
JSValueRef valueRef = ...
JSValue *value = [JSValue valueWithJSValueRef:valueRef inContext:context];

JSValue *value = ...
JSValueRef valueRef = [value JSValueRef];
```

Demo

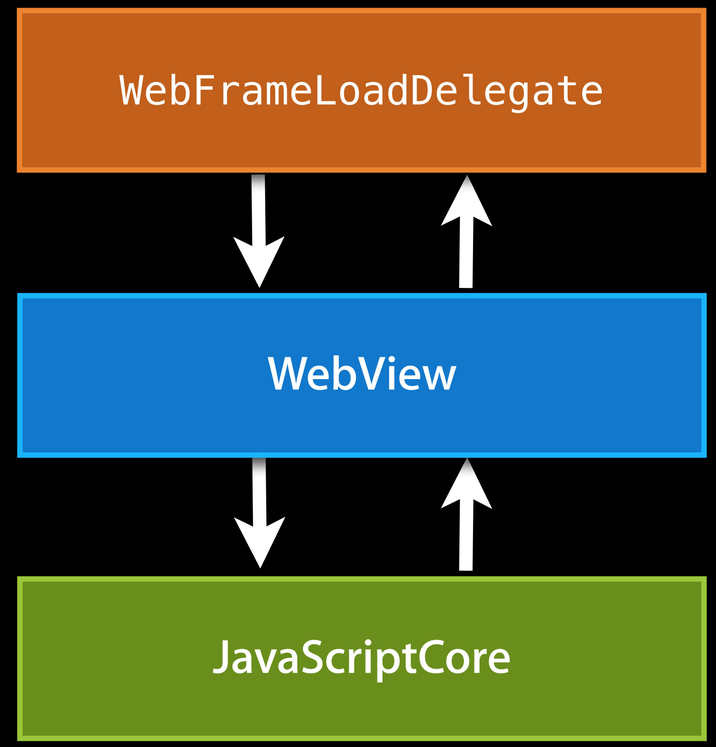
ColorMyCode

WebKit WebView

WebKit WebView (Mac)

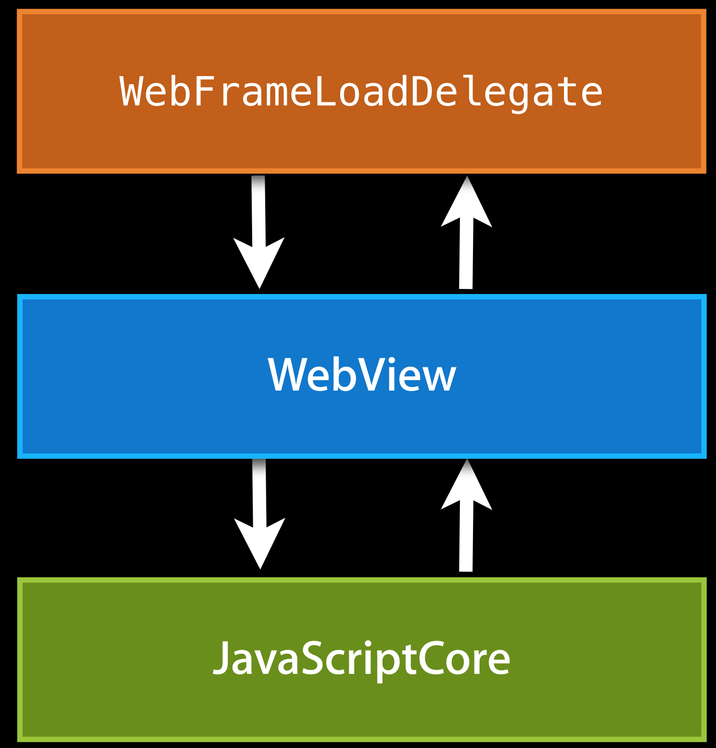
WebKit WebView (Mac)

- `-webView:didCreateJavaScriptContext:forFrame:`



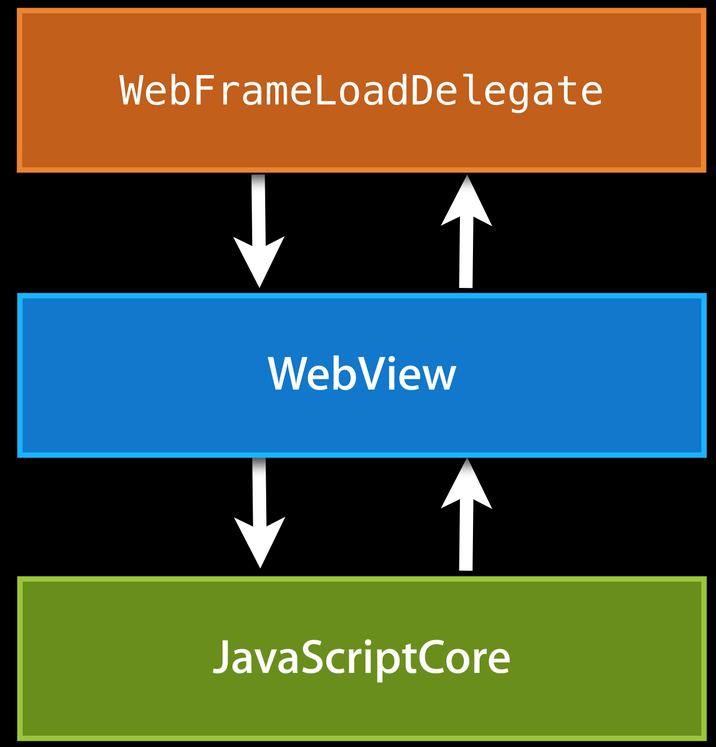
WebKit WebView (Mac)

- `-webView:didCreateJavaScriptContext:forFrame:`
- Install custom objects using context argument



WebKit WebView (Mac)

- `-webView:didCreateJavaScriptContext:forFrame:`
- Install custom objects using context argument
- Replaces old callbacks gracefully



WebKit WebView (Mac)

- `-webView:didCreateJavaScriptContext:forFrame:`
- Install custom objects using context argument
- Replaces old callbacks gracefully
- iTunes store uses WebView

WebKit WebView (Mac)

```
@implementation MyFrameLoadDelegate

- (void)webView:(WebView *)webView didCreateJavaScriptContext
(JSContext *)context forFrame:(WebFrame *)frame
{
    MyConsole *console = [[MyConsole alloc] init];
    context[@"myConsole"] = console;
}

@end
```

WebKit WebView (Mac)

```
<html>
     <head>
        <script>
        var onClickHandler = function() {
           myConsole.log("clicked!");
        };
        </script>
     </head>
     <body>
        <button onclick="onClickHandler()">Click Me!</button>
     </body>
</html>
```

Summary

What We've Covered

- Objective-C → JavaScript
- JavaScript → Objective-C
- Interfacing with the JavaScriptCore C API
- Reference counting vs. Garbage collection
- Threading
- Custom objects in WebKit WebViews

Call to Action

- Convert one bit of old code to use the new Objective-C API
- Add a small snippet of JavaScript to your app

More Information

John Geleynse
Director, Technology Evangelism
geleynse@apple.com

Documentation
JavaScriptCore.framework Headers
https://developer.apple.com/library/mac/#documentation/Carbon/Reference/WebKit_JavaScriptCore_Ref/

Apple Developer Forums
http://devforums.apple.com

Related Sessions

Getting to Know Web Inspector	Russian Hill Tuesday 10:15AM	
Getting the Most Out of Web Inspector	Russian Hill Tuesday 11:30AM	
Power and Performance: Optimizing Your Website for Great Battery Life and Responsive Scrolling	Russian Hill Wednesday 9:00AM	
Preparing and Presenting Media for Accessibility	Nob Hill Wednesday 10:15AM	
Implementing OS X Push Notifications for Websites	Marina Friday 9:00AM	
Integrating JavaScript into Native Apps	Marina Friday 10:15AM	

Labs

Integrating Web Technologies into Native Apps Lab	Media Lab A Friday 11:30 AM	
Safari and Web Tools Lab	Media Lab A Friday 2:00 PM	

WWDC2013