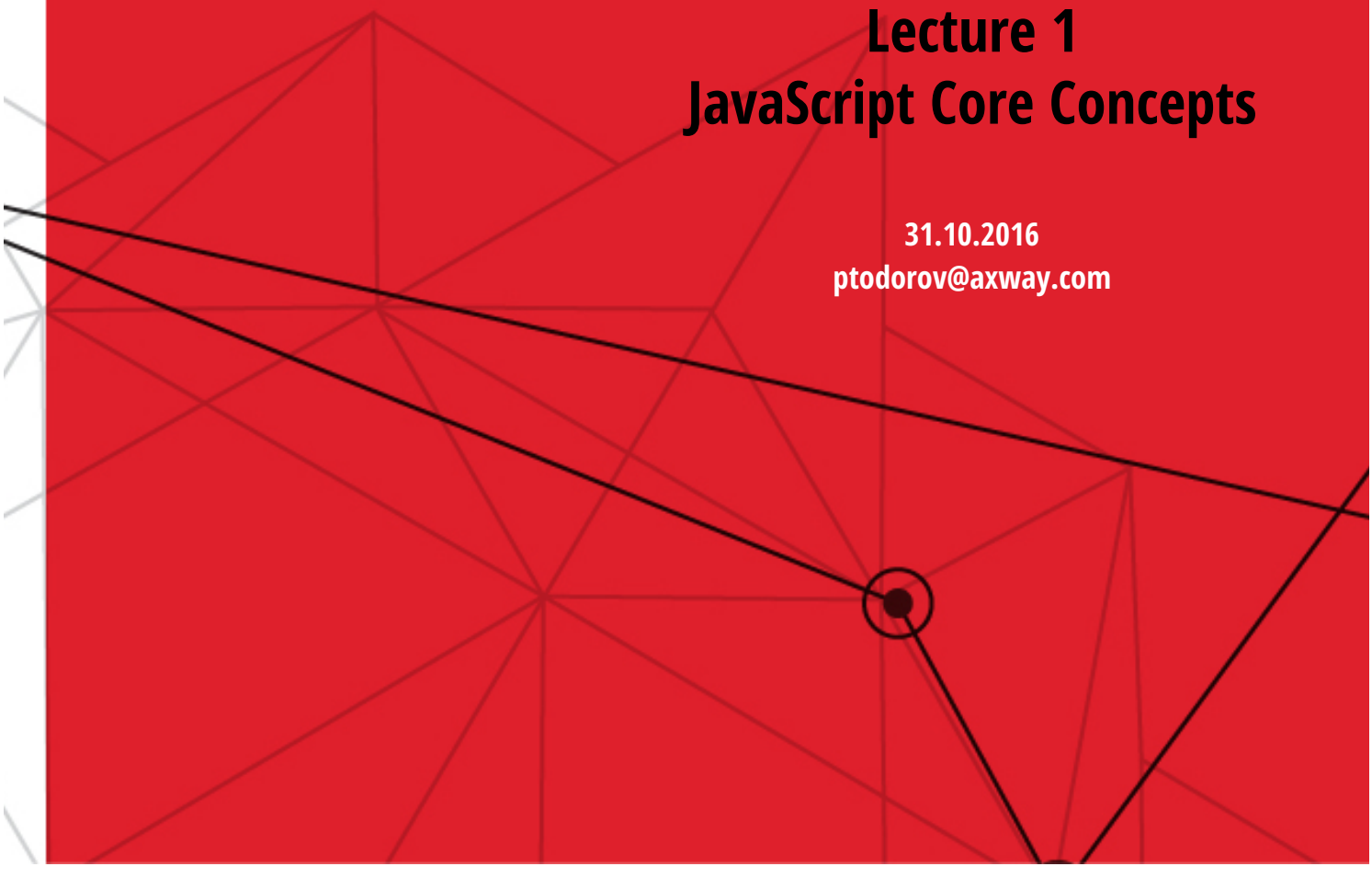
TRAINING

JAVASCRIPT
NODEJS
ARROW

Lecture 1
JavaScript Core Concepts

31.10.2016
ptodorov@axway.com



BEFORE WE START...

- Welcome!
- JavaScript Definition.
- Why We End Up Writing JavaScript?
- Disclaimer
- Training Overview
- Lecture Format
- Environment Setup
- Start...

JavaScript is
1) untyped
2) interpreted
3) dynamic
4) multi-pardigm
programming language
standardized by
ECMAScript specification.

- Wikipedia

WHY WE ALL END UP WRITING JAVASCRIPT?

Because Our Previous Experience With Java/XXX

(this is mostly personal experience)

- Java did NOT force me to do the things the right way
- Java forces me to use OOP paradigm
- Java Web Frameworks does not feel right in the context of SPA (especially component-based like JSF)
- Java feels like too high abstraction which moves you away from DevOps activities (we are not close to the metal)
- There is no such thing like full-stack developer with Java (this is a problem when you work on the full-stack)

Pitch JavaScript - Attempt Number 1 (year 2011)

WHY SPEND TIME ON JAVASCRIPT?

WHY WE ALL END UP WRITING JAVASCRIPT?

Because of the Paradigm Shift in App Development

(Devreach Conference, Sofia 2012)

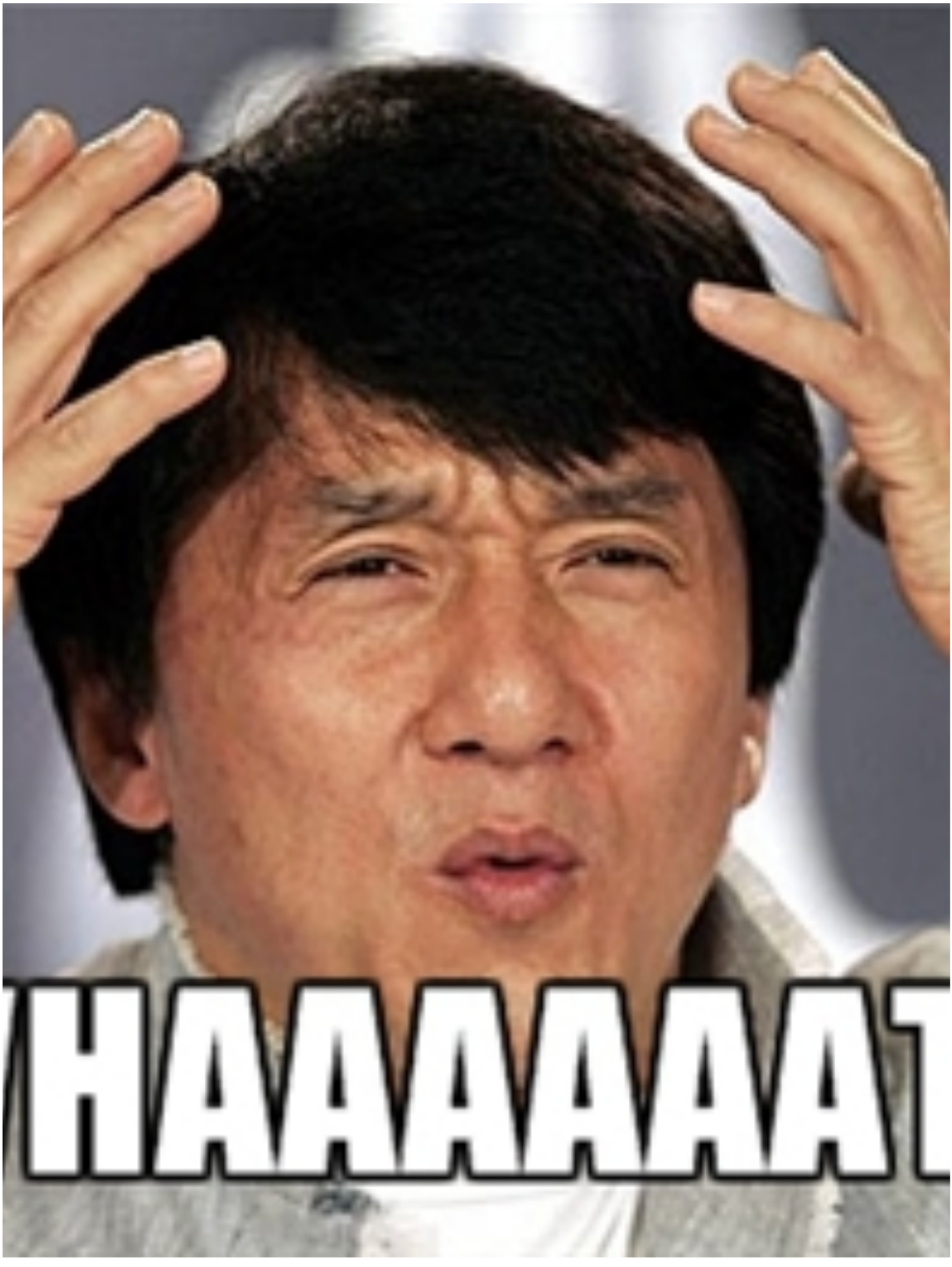
- From Web Sites to Web Applications
- From Web Applications to Mobile Applications
- From Mobile Applications to Cross Platform Mobile Applications (Titanium!)
- From Cross Platform Mobile Application to Cross Platform Desktop Applications

Pitch JavaScript - Attempt Number 2 (year 2012)

JAVASCRIPT APPLICATIONS


YOU GOTTA BE
KIDDING ME


Really?


HAAAAAAT

WHY WE ALL END UP WRITING JAVASCRIPT?

Because JavaScript has a Business Case

- Large Pool of Developers (web devs)
- Skill Reuse Across Layers (server-side with Node.js)
- Skill Reuse Across Platforms (cross platform compilation e.g. Titanium)
- **Bottom Line: Less Costs / Faster Time to Market**

WHY WE ALL END UP WRITING JAVASCRIPT?

Because of the JavaScript Explosion

- The End of Browser Wars
- Tons of FE Frameworks
- Tons of Languages that Compiles to JavaScript
- **Code Reuse Across the Server and the Client -> UNLOCKED! (Isomorphic/Universal Apps)**
- **Cross Platform Mobile/Desktop Applications -> UNLOCKED! (Hyperloop!)**
- Serverless Architectures where SPA FE Work Directly with Cloud Datasources

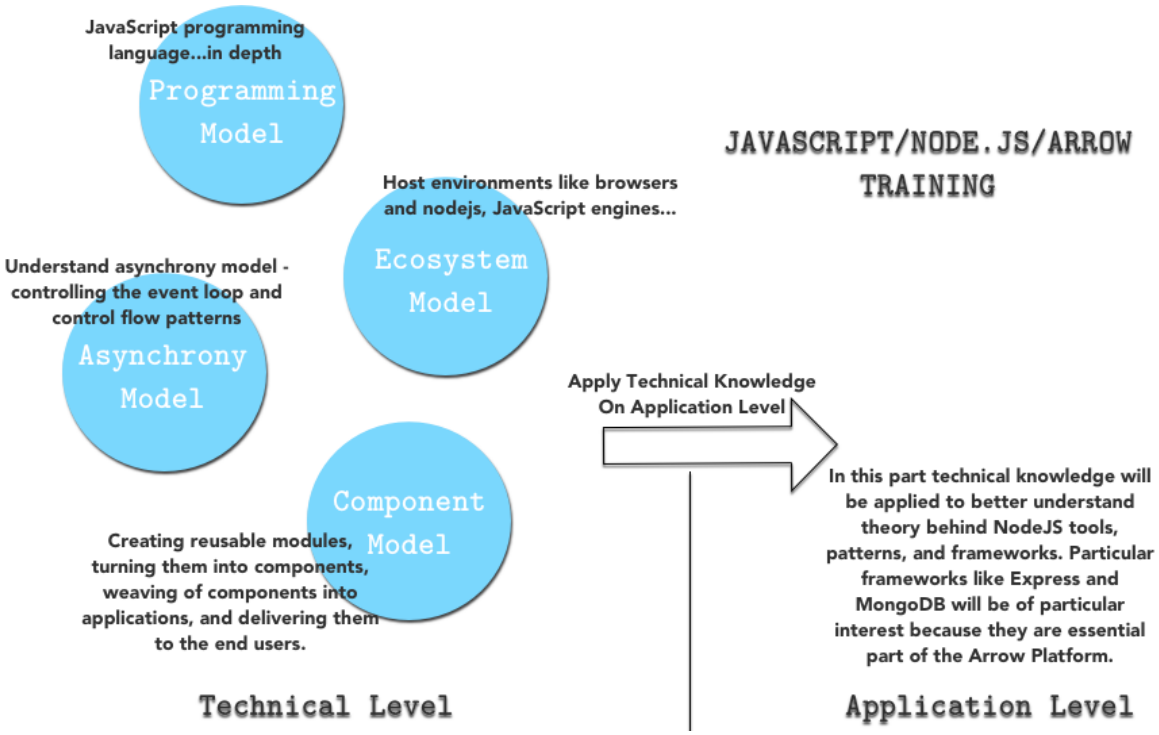
WHY WE ALL END UP WRITING JAVASCRIPT?

Because JavaScript IS REAL Programming Language

- The Standard -> fastly moved from version 3 to 6/7 turning JS into full-blown language
- Node.JS Ecosystem -> fastly moved from 0.10 to 6

Disclaimer

JavaScript/NodeJS is one very good option for I/O heavy business applications as Golang is good system language for cloud infrastructures.


JavaScript programming language...in depth
Programming Model
JAVASCRIPT/NODE.JS/ARROW TRAINING
Host environments like browsers and nodejs, JavaScript engines...
Ecosystem Model
Understand asynchrony model - controlling the event loop and control flow patterns
Asynchrony Model
Apply Technical Knowledge On Application Level
Component Model
Creating reusable modules, turning them into components, weaving of components into applications, and delivering them to the end users.
In this part technical knowledge will be applied to better understand theory behind NodeJS tools, patterns, and frameworks. Particular frameworks like Express and MongoDB will be of particular interest because they are essential part of the Arrow Platform.
Technical Level
Application Level

LECTURE FORMAT

(C)reate / (R)ead / (U)pdate / (D)elete

- Applications => Use them to manage data (loaded in-memory)
- Applications => Create them with Programming Languages
- Programming Language => Manage in-memory data

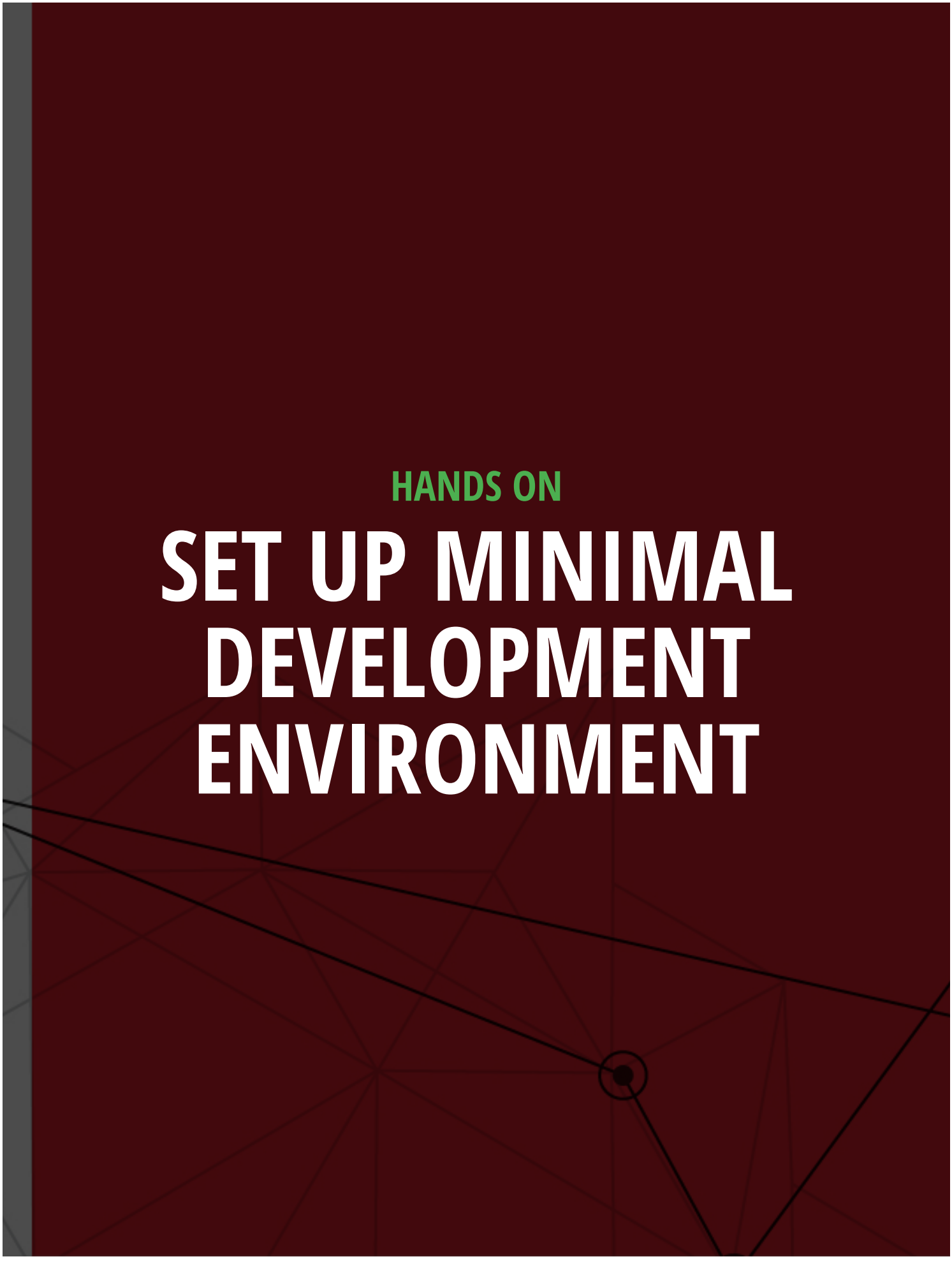
LECTURE FORMAT

Recipe / Code Examples / Takeways

- Set of Recipies will form our communication dictionary
- Each Recipe is bound to Code Examples
- **Each Recipe is bound to Takeways that hopefully will help you write quality code!**

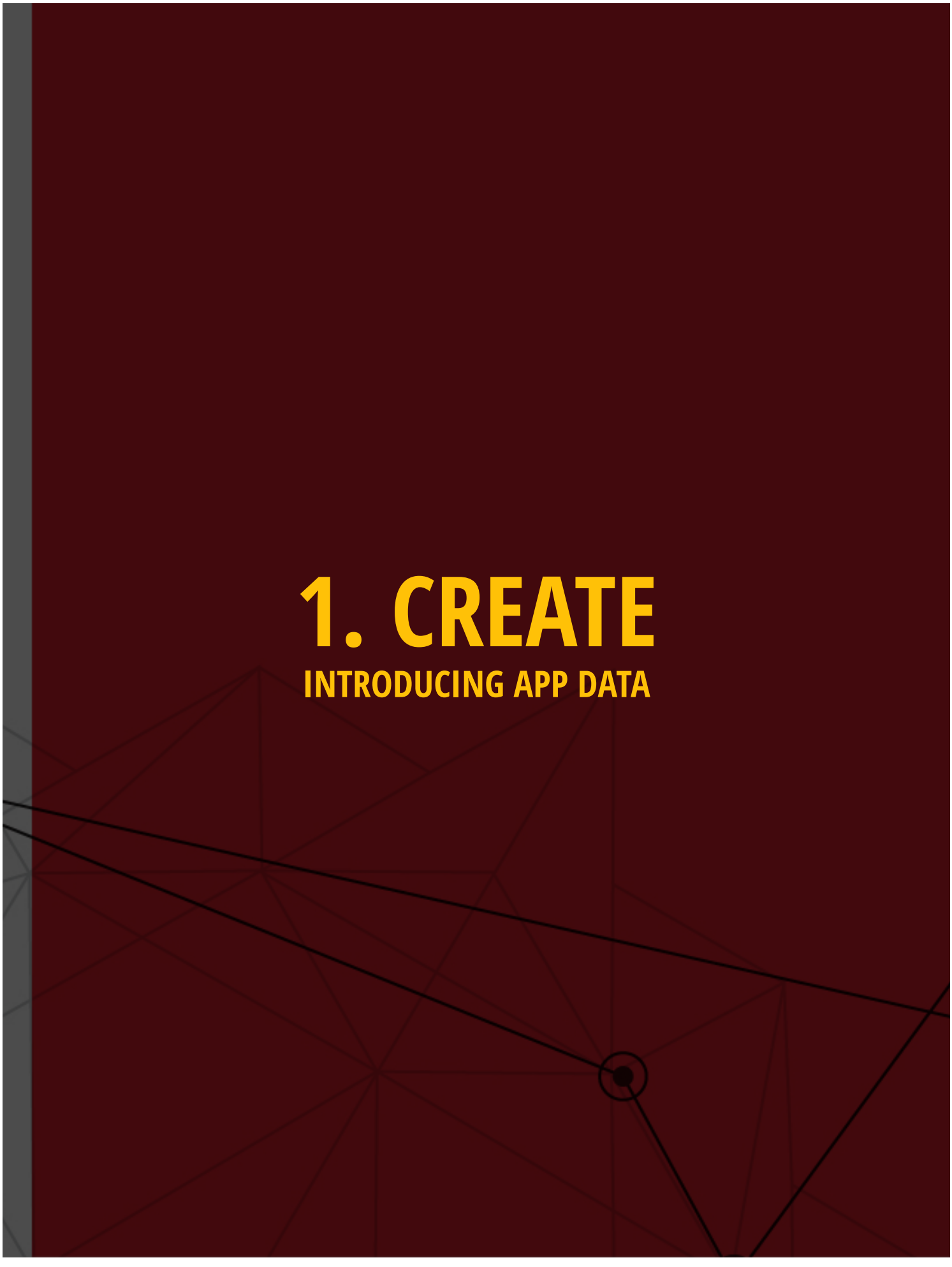
HANDS ON

SET UP MINIMAL DEVELOPMENT ENVIRONMENT



1. CREATE

INTRODUCING APP DATA



1.1 IDENTIFIERS AND DATA BINDING

identifiers-states.js

- Binding is the association of an identifier with value
- Identifier States - NOT EXISTS / NOT BOUND / BOUND
- ReferenceError vs. undefined - make difference between them

1.2 THREE WAYS TO CREATE IDENTIFIERS

identifiers-creation.js

- Variable Declaration
- Function Declaration
- Function Formal Parameters

1.3 IDENTIFIERS CREATION ORDER (HOISTING)

identifiers-creation-order.js

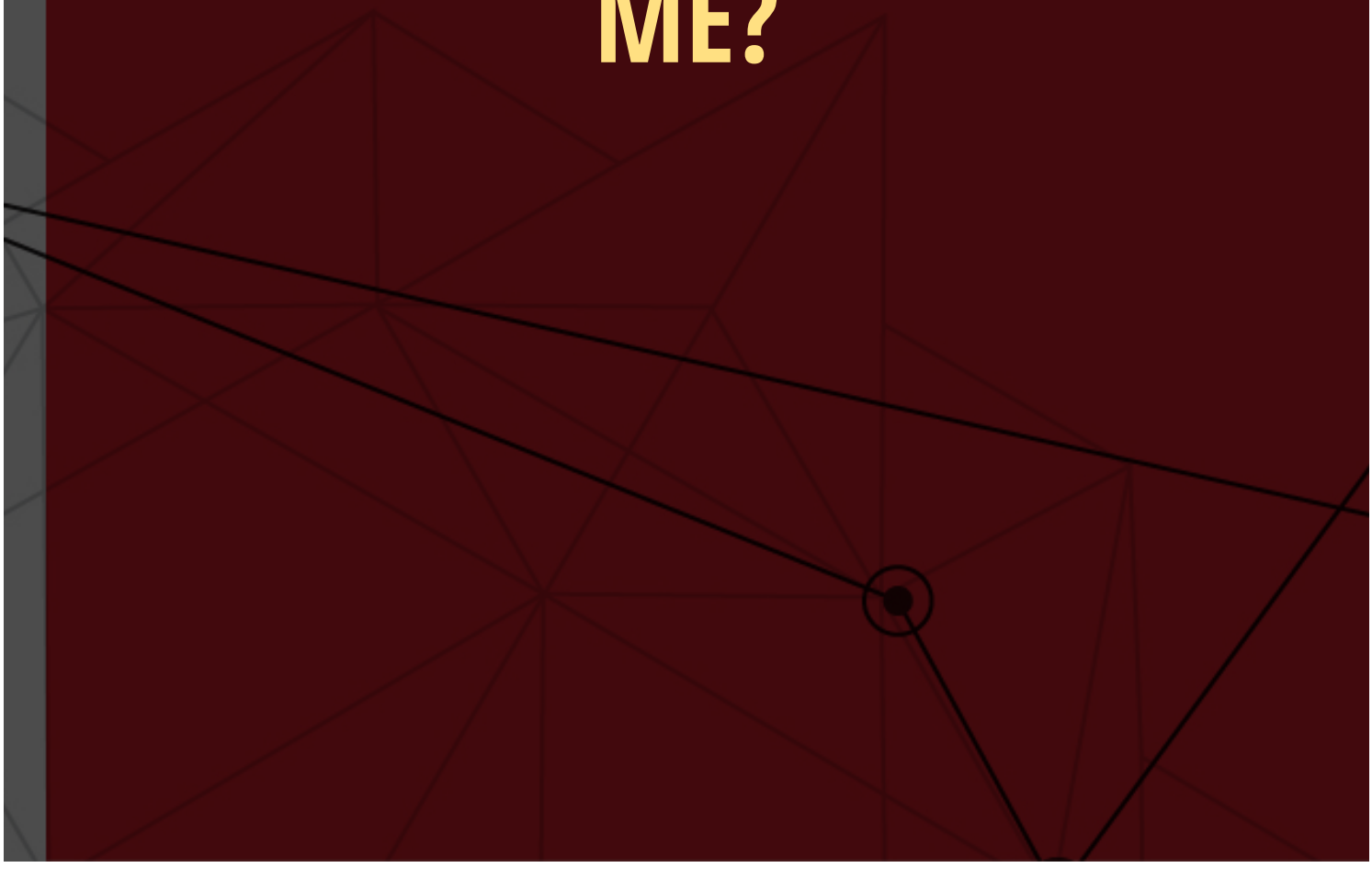
- Be aware about Function Declarations vs. Function Expressions
- Be aware about Global and Local Environments (back to this in Read Section)
- Be aware about Environment Creation vs. Environment Execution

1.4 LANGUAGE DEFINED IDENTIFIERS

identifiers-language-defined.js

- JS engine understand some built-in identifiers - Object, Array, this, arguments etc.
- Hijacking Identifier / Monkey Patching / Polyfill are good examples of why thinking in terms of identifers could be helpful

ARE YOU STILL WITH ME?



1.5 PRIMITIVE DATA TYPES

data-types-primitives.js

- 5 primitive data types
- JS is dynamic so it apply automatic type conversion - coercion
- Coercion - during arithmetic operations / comaprison with ==
- Evaluation order (from left to right) matters in case of coercion
- Comparison with === does not use coercion
- Primitives are compared by value
- Comparison and if statements -> careful with 7 falsy values

1.6 REFERENCE DATA TYPES

data-types-reference.js

- To represent complex data JS use reference types
- Reference Types - Object, Array, Function, RegExp etc.

1.7 TYPEOF VS. INSTANCEOF

data-types-reference.js

- "typeof" for Primitive Types is not working for null
- "typeof" for Reference Types works only for Object and Function
- Use "instanceof" instead of "typeof" to check the concrete Reference type

1.8 AUTOBOXING

autoboxing.js

- primitive types are autoboxed -> wrapped with corresponding Reference types - Number, String, Boolean
- do not be surprised that we could call method via identifier that points to primitive type
- manual usage of these is not recommended "e.g. var b = new Boolean(false) ... b is true"...should use vauleOf

1.9 DATA STRUCTURES

ANY DATA STRUCTURE IS REPRESENTED VIA ARRAY OR OBJECT

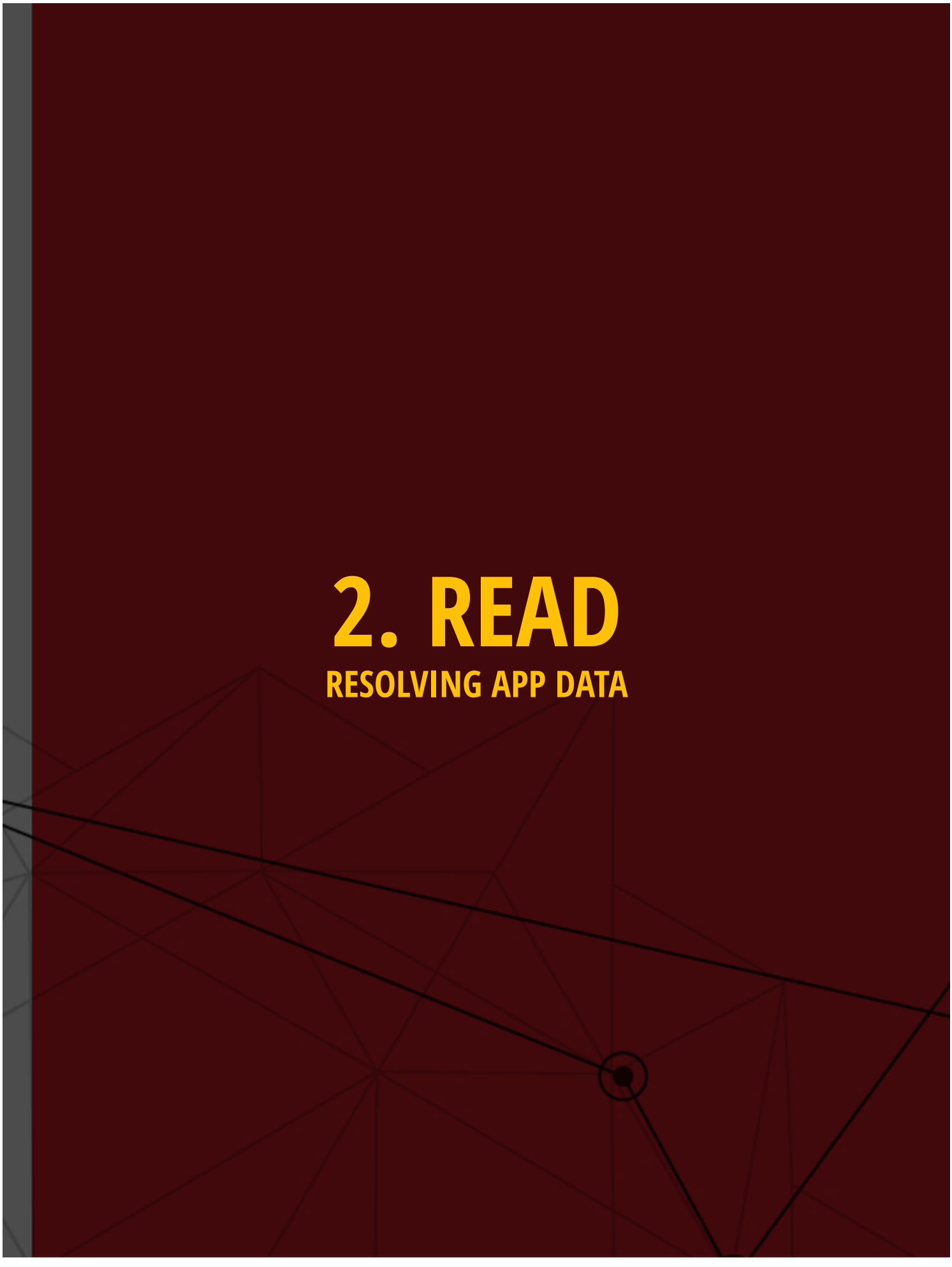
Itsy Bitsy Data Structures

DataStructures and Algorithms with JavaScript

JavaScript Algorithms (github repo with code examples)

2. READ

RESOLVING APP DATA



2.1 ENVIRONMENT - DEFINITION AND TYPES

- Environment = Set of Identificators with Their Corresponding Values
- Global Environment - only one; we have this by default
- Local Environment - many; JS engine create these during program execution

QUESTION

WHEN JAVASCRIPT ENGINE CREATE LOCAL ENVIRONMENT?

ANSWER

JAVASCRIPT ENGINE CREATES LOCAL ENVIRONMENT ON EACH FUNCTION INVOCATION

2.2 STACK OF ENVIRONMENTS

- To understand how data is resolved means to understand the stack of environments
- Lets see next diagram...

```
var a = 5; //global env
// we see only a here

function foo() {
    var a = 6; //foo env
    var b = 10; //foo env
    // we see a and b here
    // a is now 6

    function bar() {
        var c = 15; //bar env
        // we see a (=6), b, and c here
    }

    bar(); // function activation triggers
           // bar environment creation

}
// we see only a here

foo(); // function activation triggers
       // foo environment creation
```

JAVA SCRIPT PROGRAM

=

STACK OF ENVIRONMENTS

2.3 RULES OF DATA VISIBILITY

SCOPE CHAIN RESOLUTION (scope-chain.js)

- Data is visible in environment where defined (no block scope in ES5 or less)
- Data defined in previous environments is visible in newer environments
- If particular environment override data the updated value is visible in the newer environments

QUIZ 1

```
debug(p); // ?

var p = 5;

debug(p); // ?

var p = 6;

debug(p); // ?
```

QUIZ 2

```
debug(typeof x); // ?

var x = 10;

debug(x); // ?

x = 20;

function x() {};

debug(x); //?
```

QUIZ 3

```
function test() {

    bar(); // ?

    foo(); // ?

    var foo = function () {
        debug("foo");
    };

    function bar() {
        debug("bar");
    }
}

test();
```

FUNCTIONS ARE ESSENTIAL!

LETS TALK ABOUT FUNCTIONS...

QUIZ 4

```
var x = 10;

function foo() {
    var x = 20;
    var y = 30;

    var foo = function () {
        console.log(x);
        console.log(y);
    };

    var bar = new Function('console.log(x); console.log(y);');

    foo(); // ?
    bar(); // ?

}
```

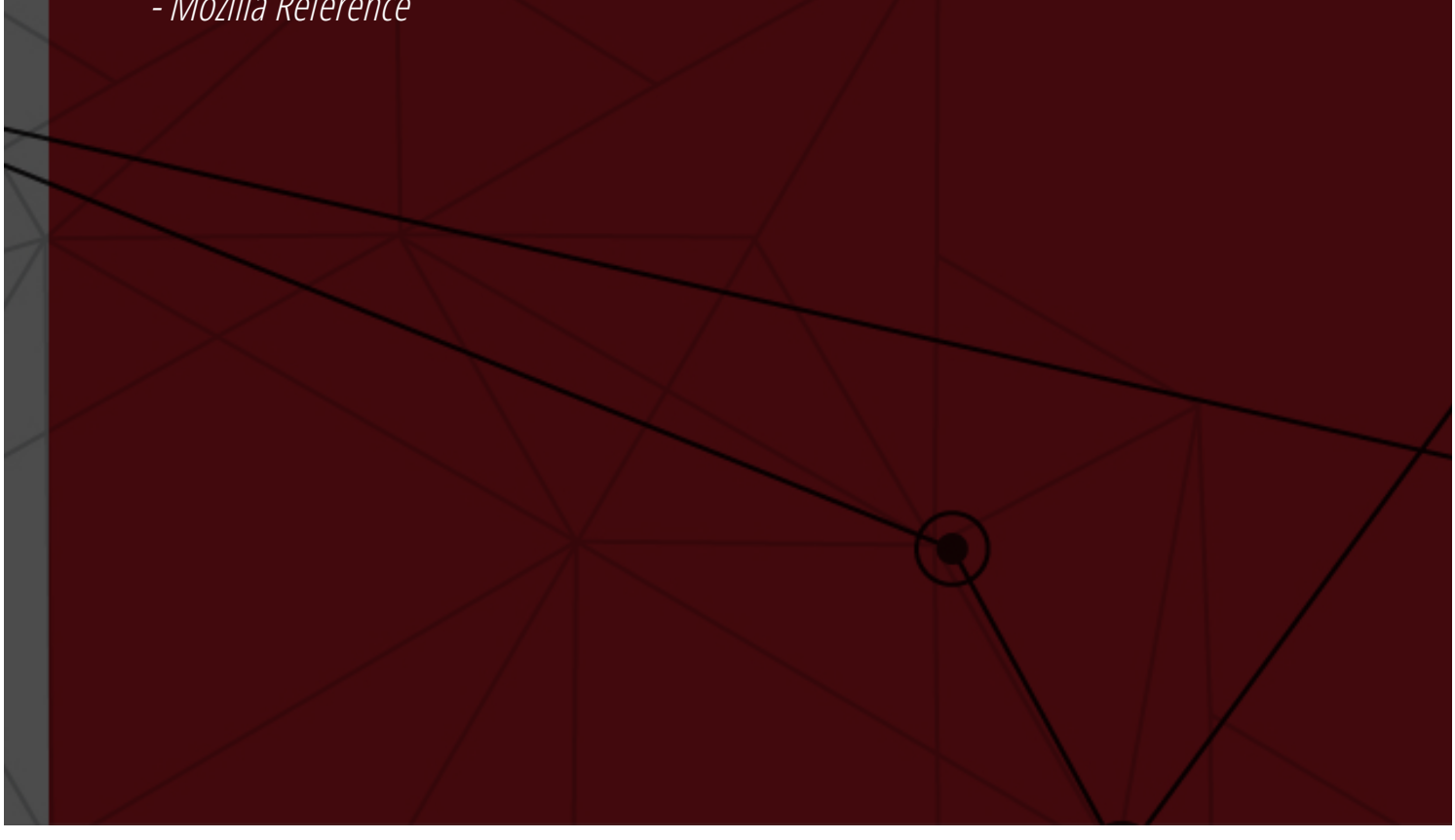
2.4 FUNCTION CONSTRUCTOR

function-constructor.js

- Function constructor does not behave like function declaration and function expression
- Make difference between lexical scope and global only scope

Functions created with the Function constructor do not create closures to their creation contexts. They are always created in the global scope.

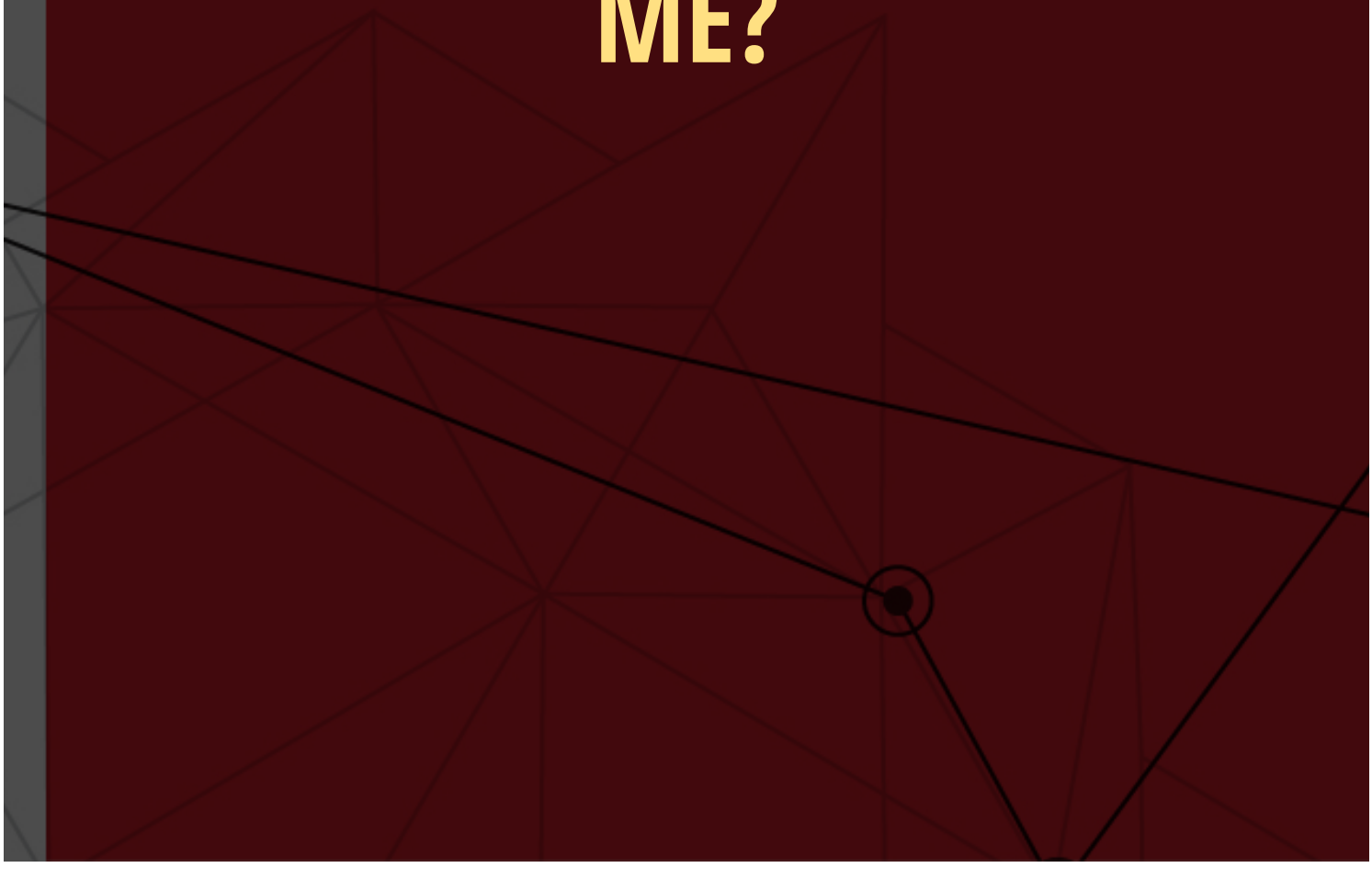
- Mozilla Reference



DATA RESOLUTION TAKEWAYS

- Think in terms of identifers ... do they exist, are they defined, from which environment they are resolved
- Know the lifecycle phases - Environment Creation / Environment Execution
- Make difference between functions created with declaration / expression / constructor
- **Bottom Line: If you want predicatble data in your environments/scope chain use Function Declarations**

ARE YOU STILL WITH ME?



2.5 CLOSURES

Definition

> **Combination of code block and data... If you understand the scope chain, the question on understanding closures in ECMAScript will disappear by itself.**

- DMITRY SOSHNIKOV

2.5 CLOSURES

In General Theory

- FUNARG: FUNCTION USED AS FUNCTION ARGUMENT
- HIGHER ORDER FUNCTION: THE ONE THAT TAKES FUNARG
- FUNCTION VALUED FUNCTIONS: FUNCTIONS THAT RETURN OTHER FUNCTIONS
- FIRST CLASS FUNCTIONS: CAN BE PASSED AS ARGUMENTS AND RETURNED FROM OTHER FUNCTIONS

2.5 CLOSURES

In ECMAScript (closure.js)

- Functions are data
- Functions can be passed as parameters and returned by other functions
- Anonymous Function could be used as Funarg

QUIZ

```
var firstClosure;
var secondClosure;

function foo() {
  var x = 1;

  firstClosure = function () { return ++x; };
  secondClosure = function () { return --x; };

  x = 2;
  console.log(firstClosure()); // ?
}
foo();
console.log(firstClosure()); // ?
console.log(secondClosure()); // ?
```

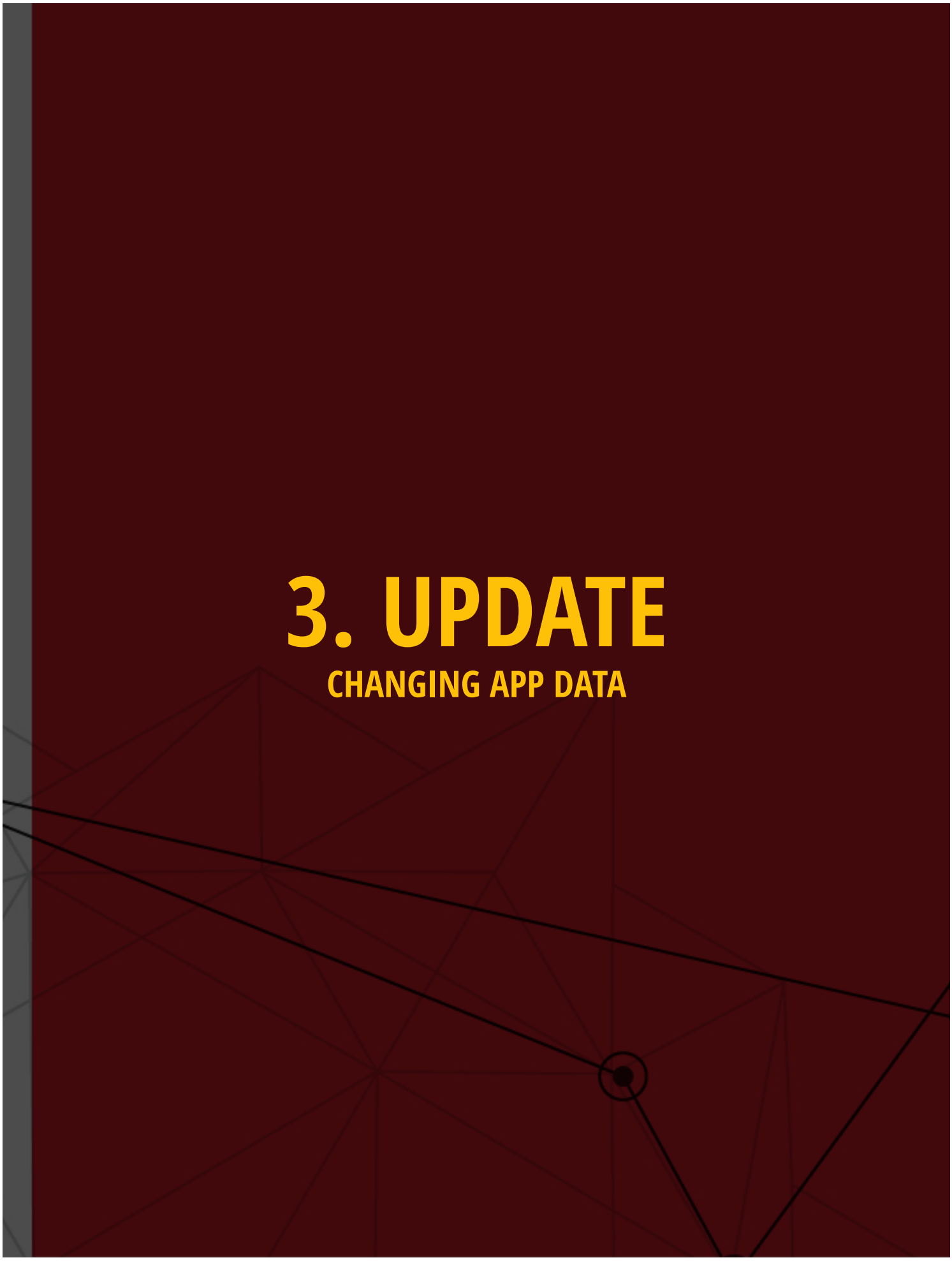
2.5 CLOSURES IN ECMASCRIPT

TAKEWAYS

- All functions are first-class
- All functions are closures except function created with Function constructor
- Closures from the same environment share the same scope
- FOCUS ON: functions that have function arguments that use free variables
- FOCUS ON: functions that return functions

3. UPDATE

CHANGING APP DATA



CHANGE DATA TAKEWAYS IN ONE SLIDE

binding-rebinding-mutation.js

- Make difference between binding / rebinding / mutation
- Make difference between call by value / call by reference / call by sharing
- The more immutable data the better

4. DELETE

REMOVING APP DATA

> **Many modern language engines such as Chrome's V8 JavaScript engine dynamically manage memory for running applications so that developers don't need to worry about it themselves. The engine periodically passes over the memory allocated to the application, determines which data is no longer needed, and clears it out to free up room. This process is known as garbage collection.**

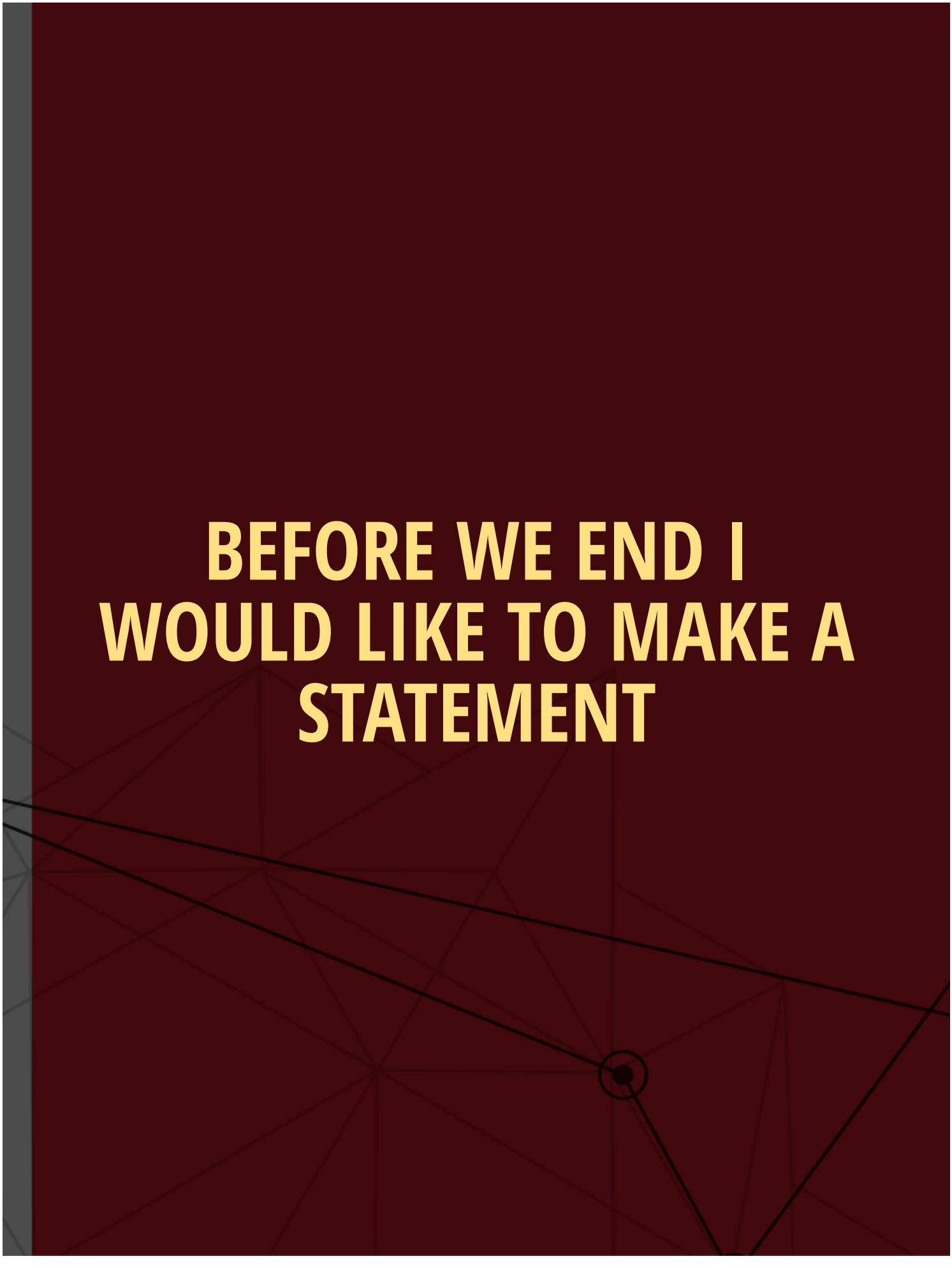
- V8 JavaScript Engine Blog

DELETE DATA TAKEWAYS IN ONE SLIDE

data-deletion.js

- JavaScript engines use garbage collection
- Manuall deletion could be made with 'delete' operator
- Developer can delete only properties with 'DontDelete' attibute set to false but no identifiers

BEFORE WE END I WOULD LIKE TO MAKE A STATEMENT



1. We could perfectly write programs with the paradigms we already learned. So our programs could stick only to -> functions / scope chain resolution / objects as key-value data stores.

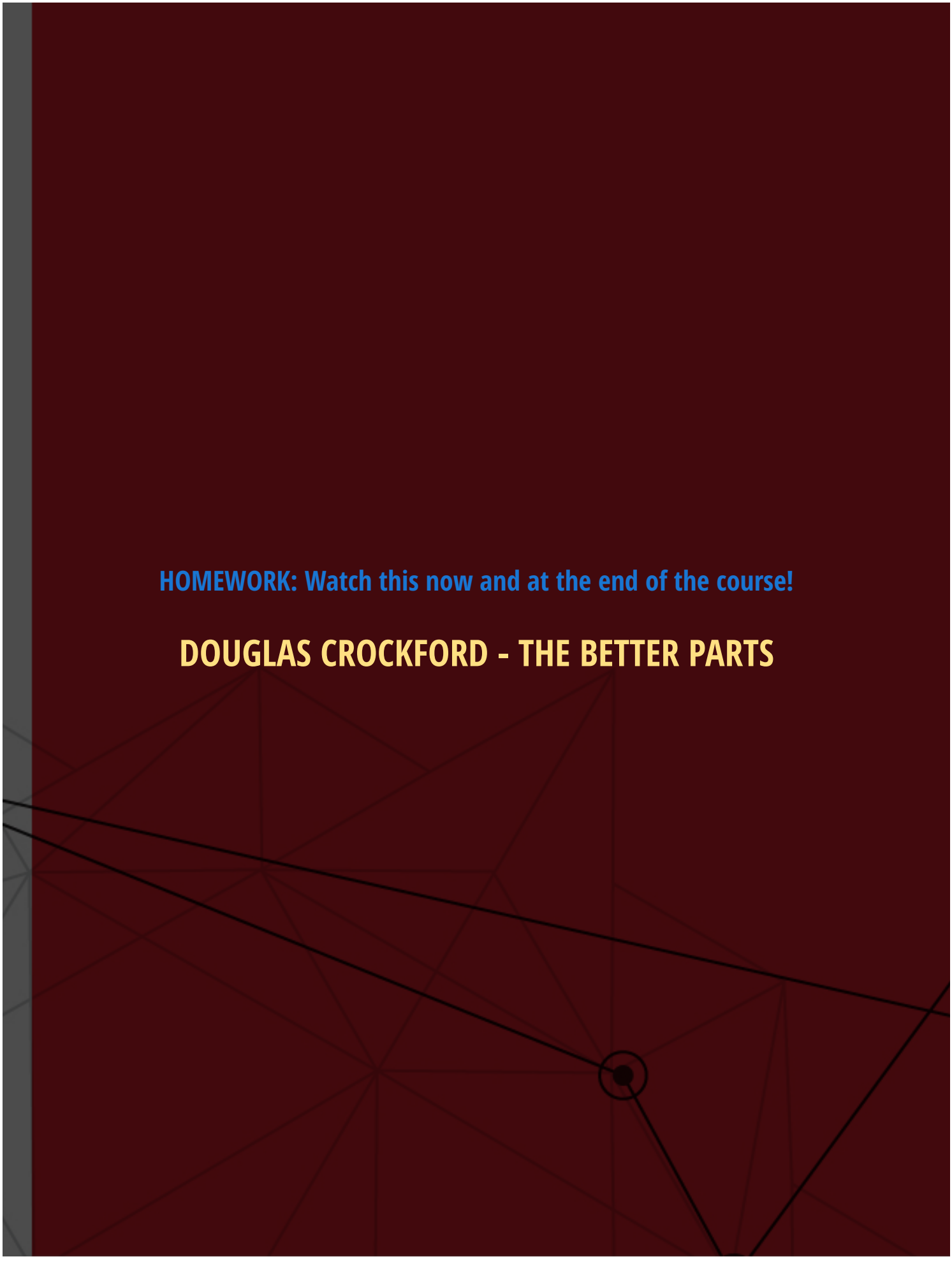
2. This is great because our programs will not only be simpler and less error prone but will also fit in the context of current trends - doing functional programming with immutable data structures.

3. As you will see in the forthcoming lectures adding OOP into the mix will add a whole new set of complexity having to deal with - prototype chain / this context and its binding / various OOP patterns that do not naturally fit to JavaScript language (which is mitigated with newest standards ES6/7).

Bottom Line: The added value by this additional abstraction layer is so little that it does not justify the additional complexity which you must cope with.

HOMEWORK: Watch this now and at the end of the course!

DOUGLAS CROCKFORD - THE BETTER PARTS



WANT TO LEARN MORE?

1. KEEP CALM
2. GRAB THE SOURCE CODE
3. START CODING

THANK YOU!

ptodorov@axway.com