# 如何构建一个MVVM框架

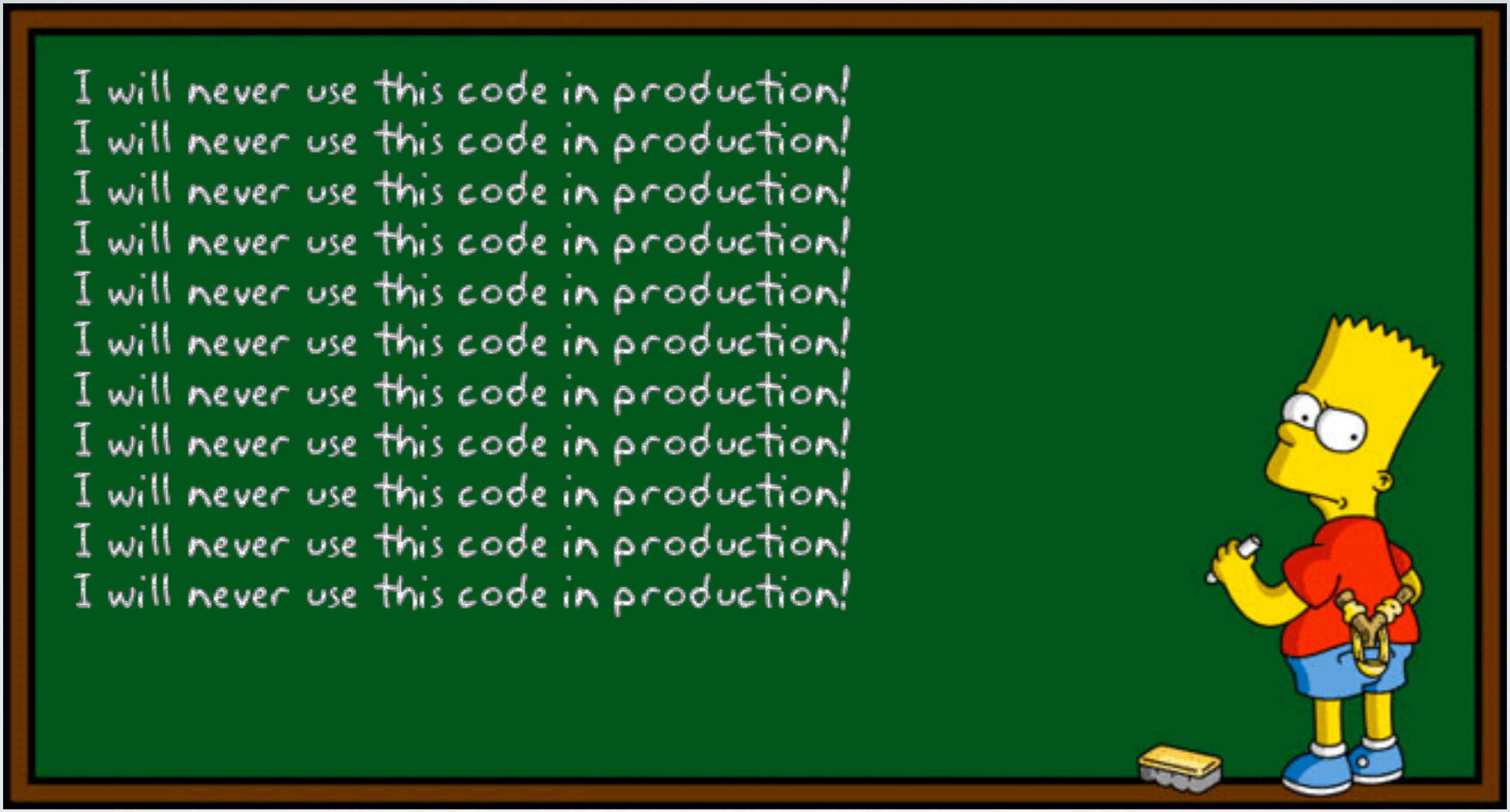
# 自我介绍

目前是饿了么的前端工程师。

5年UI组件开发经验，在前司从事开发一个类似ExtJS的框架。

# SIMPLE MVVM

https://github.com/furybean/simple-mvvm

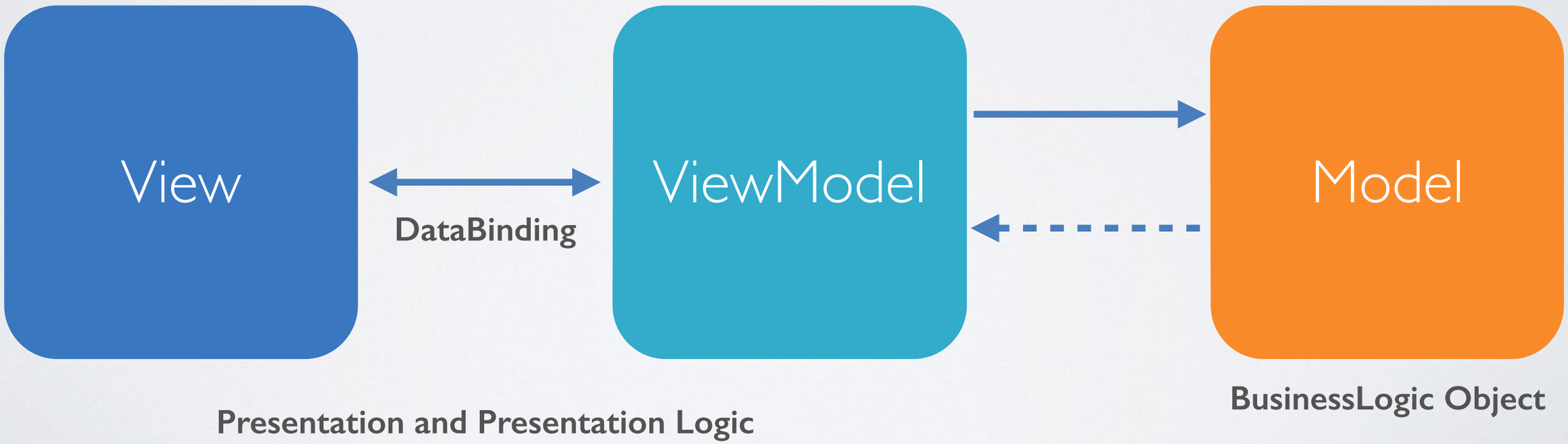

I will never use this code in production!
I will never use this code in production!
I will never use this code in production!
I will never use this code in production!
I will never use this code in production!
I will never use this code in production!
I will never use this code in production!
I will never use this code in production!
I will never use this code in production!
I will never use this code in production!

# AGENDA

- 介绍MVVM
- MVVM的组成
- 实现Compiler
- 实现ViewModel
- 实现Directive

# MVVM

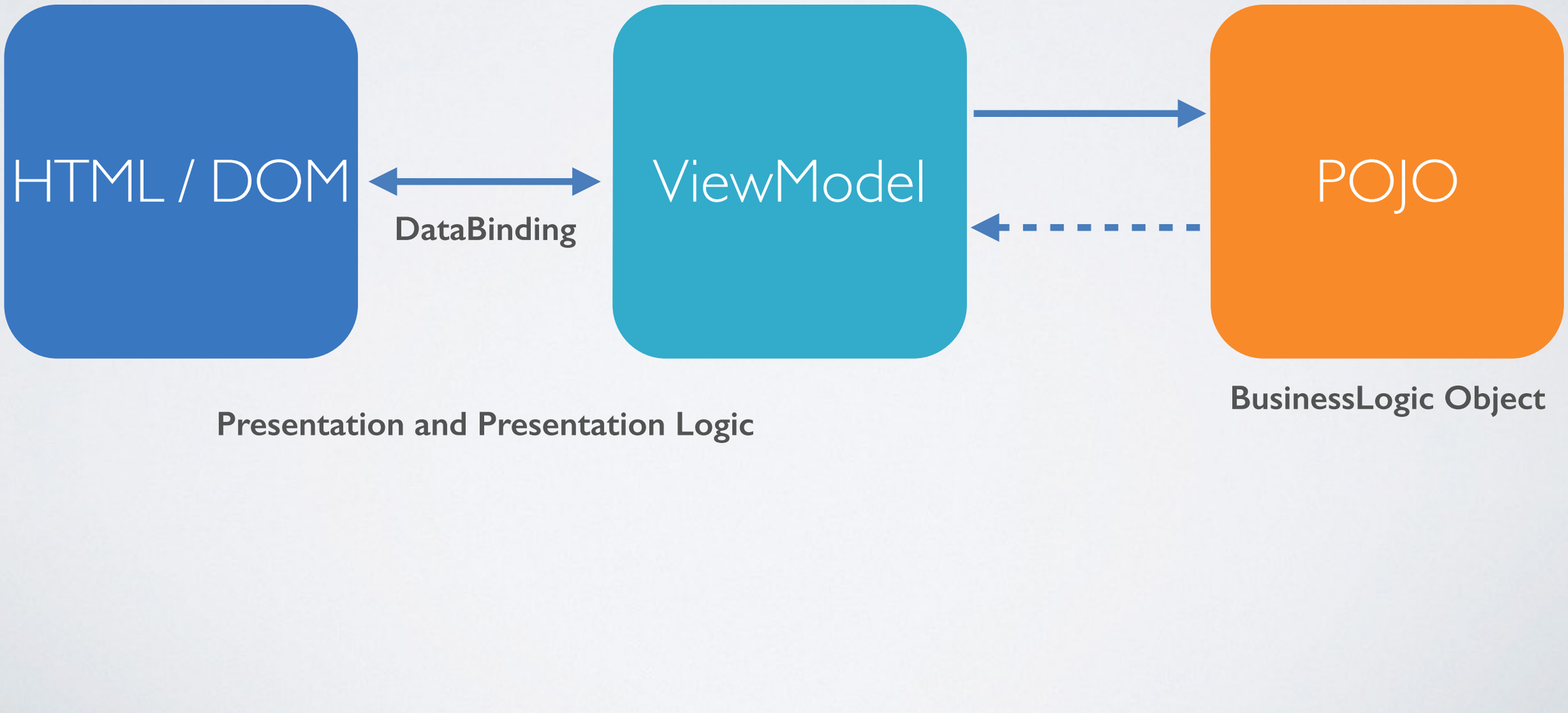
- https://en.wikipedia.org/wiki/Model_View_ViewModel



# WELL KNOWN MVVM

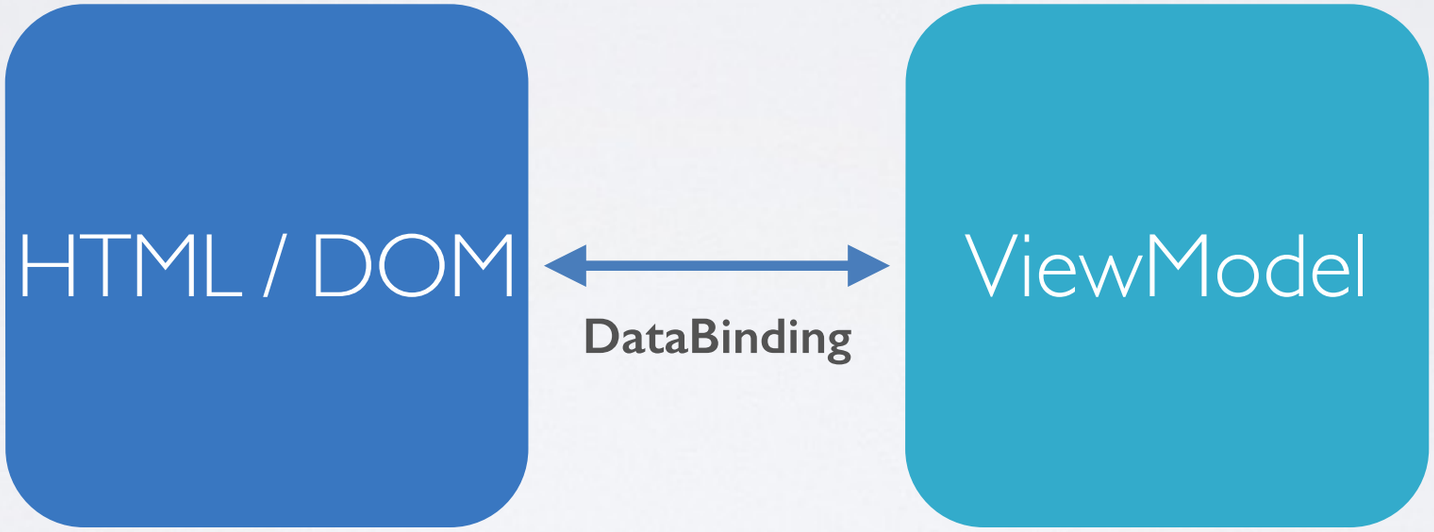
- AngularJs: Controller + HTML
- Ember.js: Controller + Handlebars + Model
- KnockoutJS: Observable + HTML
- Vue.js: ViewModel + HTML

# MVVM FOR WEB

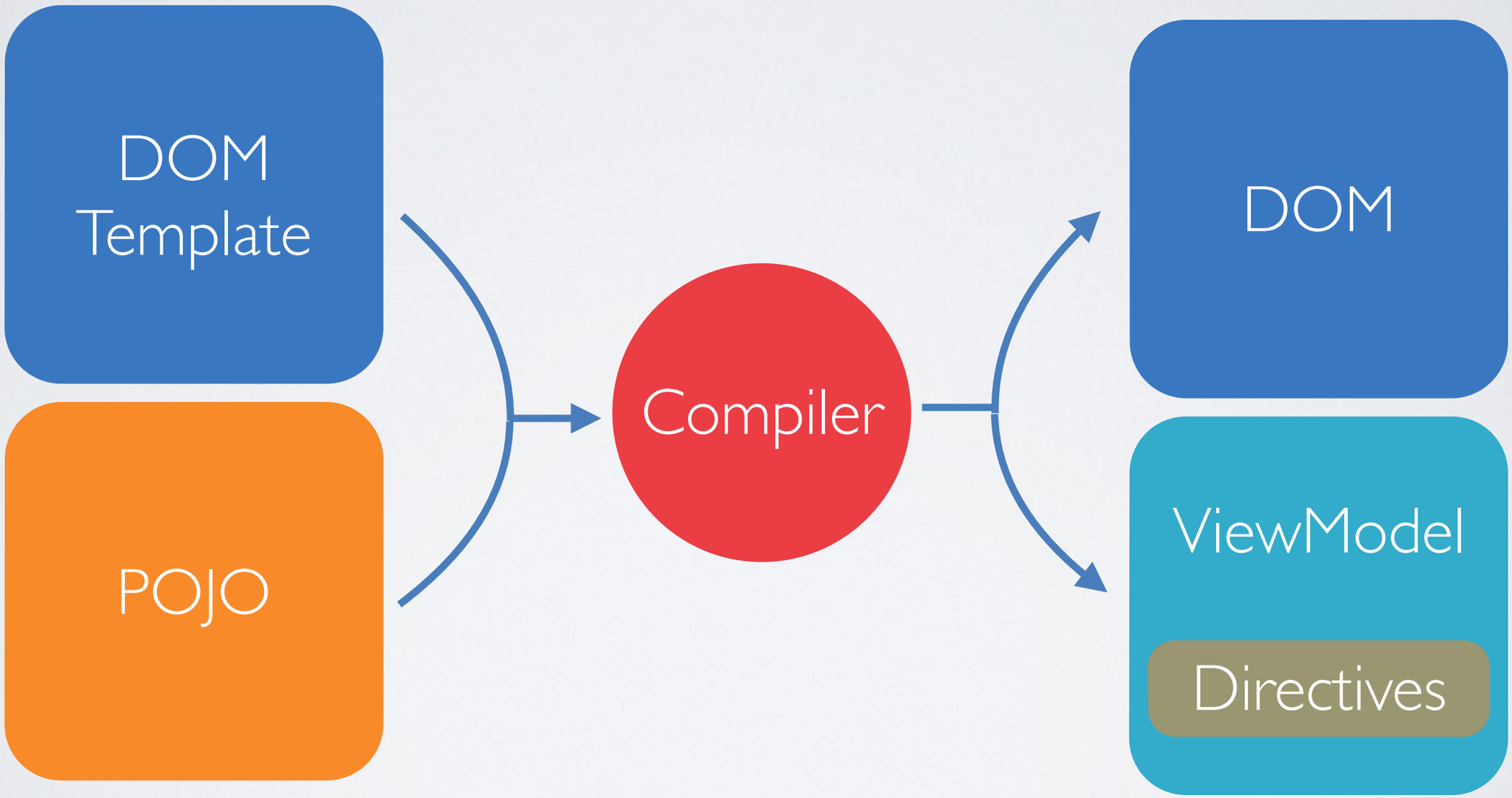

HTML / DOM
DataBinding
ViewModel
POJO
Presentation and Presentation Logic
BusinessLogic Object

# MVVM组成

# PROBLEM



# I/O



# 实现COMPILER

# MAIN

$compile(element, context)

# TODOS

Input your new todo

Add

- [x] ~~first~~ Remove
- [ ] second Remove

# CONTEXT

```
var context = {
  newTodo: "",
  todos: [
    { title: 'first', done: true },
    { title: 'second', done: false }
  ],
  add: function() {
    this.todos.push({ title: this.newTodo, done: false });
    this.newTodo = '';
  },
  remove: function(item) {
    this.todos.splice(this.todos.indexOf(item), 1);
  }
};
```

# HTML

```
<div id="demo">
  <input d-model="newTodo" placeholder="Input your new todo"/>
  <button d-click="add()">Add</button>
  <ul>
    <li d-repeat="item in todos">
      <input type="checkbox" d-model=“item.done"/>
      <span d-class="done:item.done">{{item.title}}</span>
      <a href="Javascript:" d-click="remove(item)">Remove</a>
    </li>
  </ul>
</div>
```

# DOM WALK

```
function walk(node, callback) {
  if (node.nodeType === 1 || node.nodeType === 3) {
    var returnValue = callback(node);
    if (returnValue === false) {
      return;
    }
  }

  if (node.nodeType === 1) {
    var current = node.firstChild;
    while (current) {
      walk(current, callback);
      current = current.nextSibling;
    }
  }
}
```

# SUB CONTEXT

```
<div id="demo">
  <input d-model="newTodo" placeholder="Input your new todo"/>
  <button d-click="add()">Add</button>
  <ul>
    <li d-repeat="item in todos">
      <input type="checkbox" checked="{{item.done}}"/>
      <span d-class="done:item.done">{{item.title}}</span>
      <a href="Javascript:" d-click="remove(item)">Remove</a>
    </li>
  </ul>
</div>
```

context

sub context

# CONTEXT的继承

```
function newContext(context) {
    var empty = function() {};
    empty.prototype = context;

    return new empty();
}
```

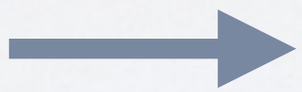
```
Object.create(context);
```

# WALK NODE

1. 节点不存在d-repeat属性，把表达式的值映射到DOM，继续WALK子节点。
2. 节点存在d-repeat属性，结束WALK子节点：
    1. Clone当前节点后把d-repeat属性删除，作为数组元素使用的模板childTemplate。
    2. 对数组中每个元素创建一个Sub Context，执行$compile(childTemplate, subContext)

# 从CONTEXT取值

```
item.title
```



```
(function() {
  return this.item.title;
}).bind(context);
```

# CODE TO FUNCTION

```
var compileFn = function(body, context) {
  var fn;
  if (body) {
    fn = new Function('return ' + body + ';');
  }
  if (fn && context) {
    return fn.bind(context);
  }
  return fn;
};
```

# PARSE TYPES

- Repeat Expression(d-repeat): item in todos
- Expression(attribute value): newTodo
- Text(text node): {{item.title}}
- Pair(d-class): done: item.done

# EXPRESSION

- newTodo
- item.title
- add()
- remove(item)

# EXPRESSION OUTPUT

- newTodo => this.newTodo
- item.title => this.item.title
- add() => this.add()
- remove(item) => this.remove(this.item)

# OPEN SOURCE PARSER

- esprima： http://esprima.org/

  Esprima is a high performance, standard-compliant ECMAScript parser written in ECMAScript

- jsep: https://github.com/soney/jsep

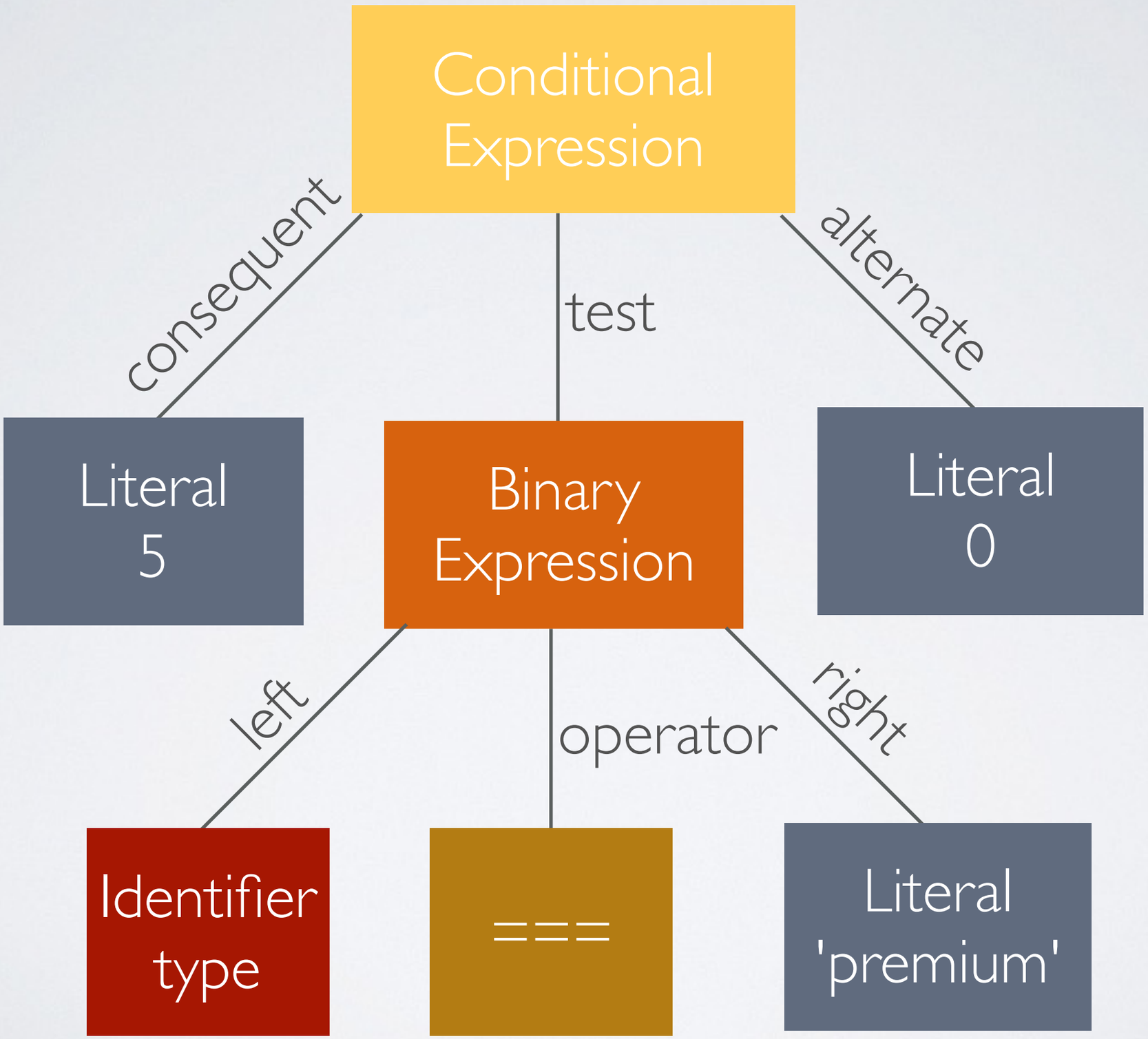
  JavaScript Expression Parser

# JSEP PARSE RESULT

- type === 'premium' ? 5 : 0

```
{
   type: "ConditionalExpression",
   test: {
      type: "BinaryExpression",
      operator: "===",
      left: {
         type: “Identifier", name: "type"
      },
      right: {
         type: “Literal", value: "premium"
      }
   },
   consequent: {
      type: “Literal", value: 5
   },
   alternate: {
      type: “Literal", value: 0
   }
}
```

type === 'premium' ? 5 : 0



# AST TO CODE

```
function astToCode(ast) {
   if (ast.type === 'Literal') {
      return typeof ast.value === 'string' ? '"' + ast.value + '"' : ''
+ ast.value;
   } else if (ast.type === 'ThisExpression') {
      return 'this';
   } else if (ast.type === 'UnaryExpression') {
      return ast.operator + astToCode(ast.argument);
   } else if (ast.type === 'BinaryExpression' || ast.type ===
'LogicalExpression') {
      return astToCode(ast.left) + ' ' + ast.operator + ' ' +
astToCode(ast.right);
   } else if (ast.type === 'ConditionalExpression') {
      return '(' + astToCode(ast.test) + ' ? (' +
astToCode(ast.consequent) + ') : (' + astToCode(ast.alternate) + '))';
   } else if (ast.type === 'Identifier') {
      return 'this.' + ast.name;
   }
   //...
}
```

# PARSE TEXT(INTERPOLATE)

text{{expression}}text{{expression}}

使用正则会遇到的边界情况：

- {{、}}在字符串或者在表达式{}()[]里

遍历字符串，定义两个变量：inString和level。

# PARSE PAIR

key: expression, key: expression

# 实现VIEWMODEL

# WATCH

- Dirty Check: Angular.js 1.x
- defineProperty: Ember.js / Vue.js
- Object.observe

# OBJECT.OBSERVE

```
var obj = {
   foo: 0
};

Object.observe(obj, function(changes) {
   console.log(changes);
});

obj.foo = 2;

//[{type: 'update', object: { foo: 2 }, name: 'foo', oldValue:
0 }]
```

# 方法

- $watch: Object.observe
- $unwatch
- $extend: Object.create
- $destroy

# 实现DIRECTIVE

# DIRECTIVE TYPE

- attr => setAttribute
- event => addEventListener
- text => innerText / nodeValue
- class => className
- model => value / addEventListener
- repeat => [ Element ]

# DIRECTIVE 属性

- element
- context
- expression
- attr/className/eventName

# DIRECTIVE 方法

- bind
- update
- unbind
- destroy

# DIRECTIVE BIND

```
var directive = this;
if (directive.element && directive.expression && directive.context){
  directive.valueFn = compileExpr(directive.expression,
directive.context);

  var depends = getDepends(directive.expression);
  var context = directive.context;

  depends.forEach(function(depend) {
    context.$watch(depend, directive);
  });

  directive.update();
}
```

# EXPRESSION DEPENDS

- 从AST提取，只关注两种类型：

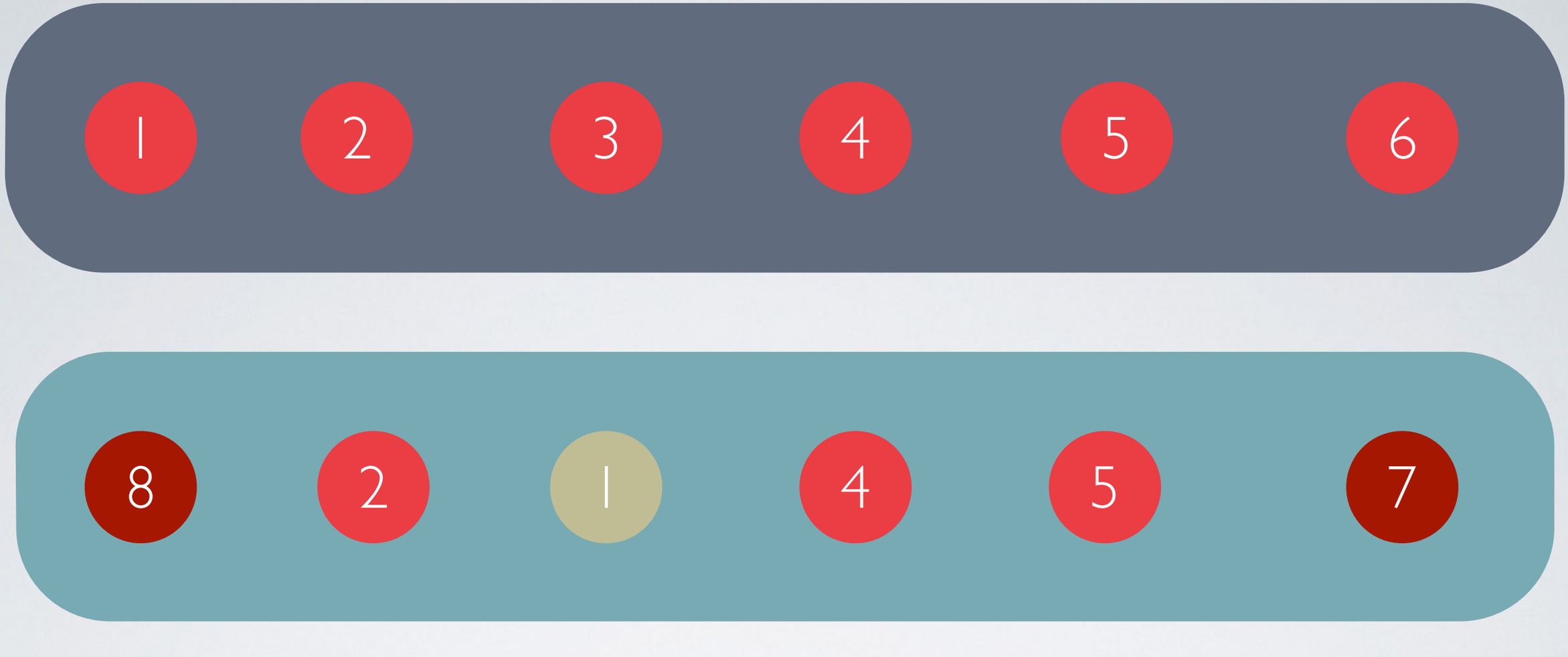
- Identifier: newTodo

- MemberExpression: item.done

# REPEAT DIRECTIVE

item in todos track by id

- track by的作用是对Array中的元素进行hash
- Array的diff就简化成了有序集合的diff




- 遍历集合A，如果元素在集合A中存在，集合B中不存在，则该元素被删除
- 遍历集合B，如果元素在集合B中存在，集合A中不存在，则该元素为新增

  如果元素在集合B中存在，集合A中存在，但是元素的前一个元素不同，则该元素被移动

# REVIEW

- DOM TREE WALK
- AST WALK
- ViewModel => EVENT EMITTER
- ARRAY DIFF => SET DIFF

# THANKS