Vue.js Framework

Internet Engineering

Spring 2018

Pooya Parsa

Professor: Bahador Bakhshi

CE & IT Department, Amirkabir University of Technology

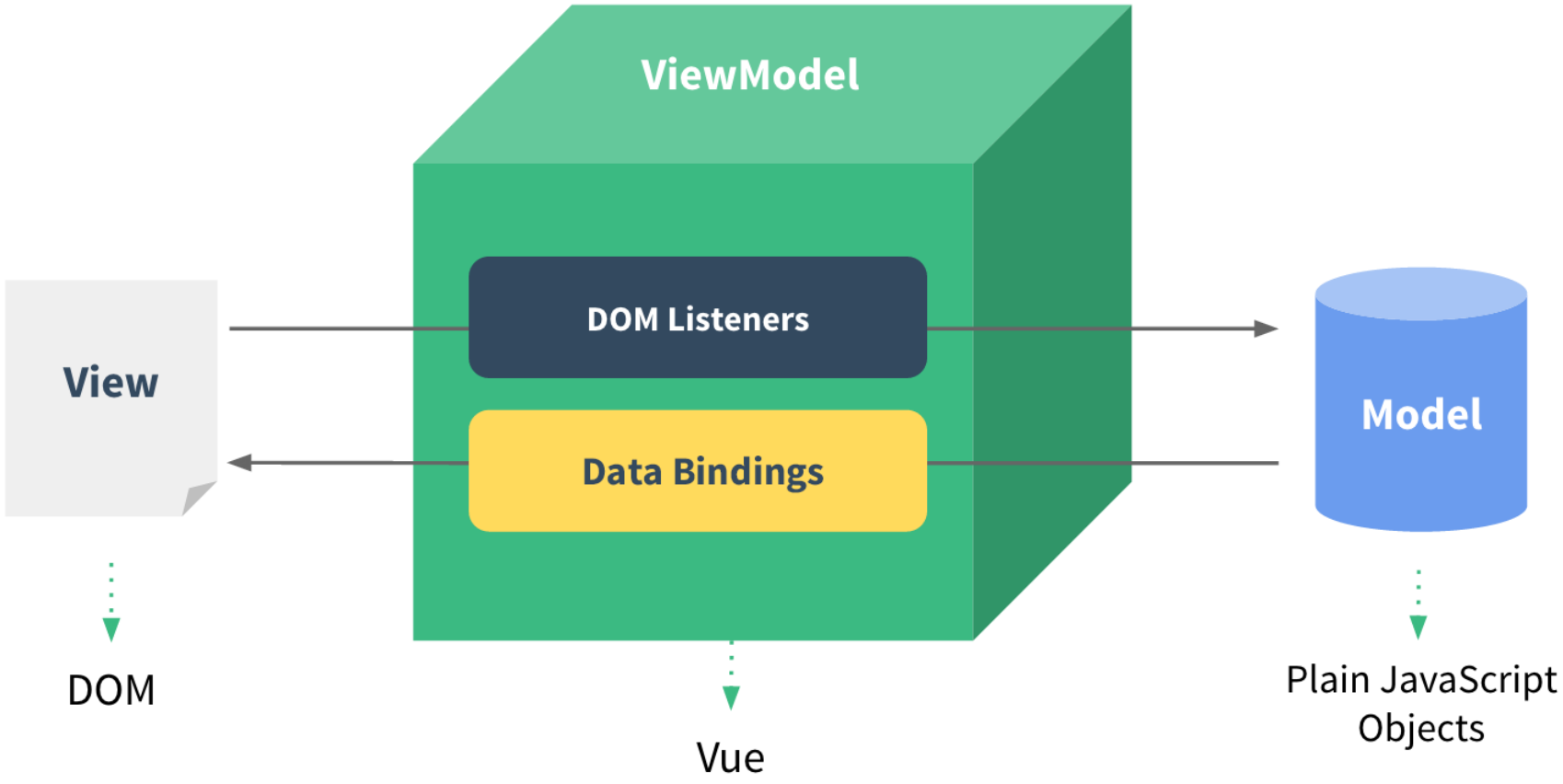
Outline

- Introduction to Vue.js
- The Vue instance
- Declarative Rendering
- Event Listeners & Input handling
- v-if and v-for Directives
- Computed Props
- Components

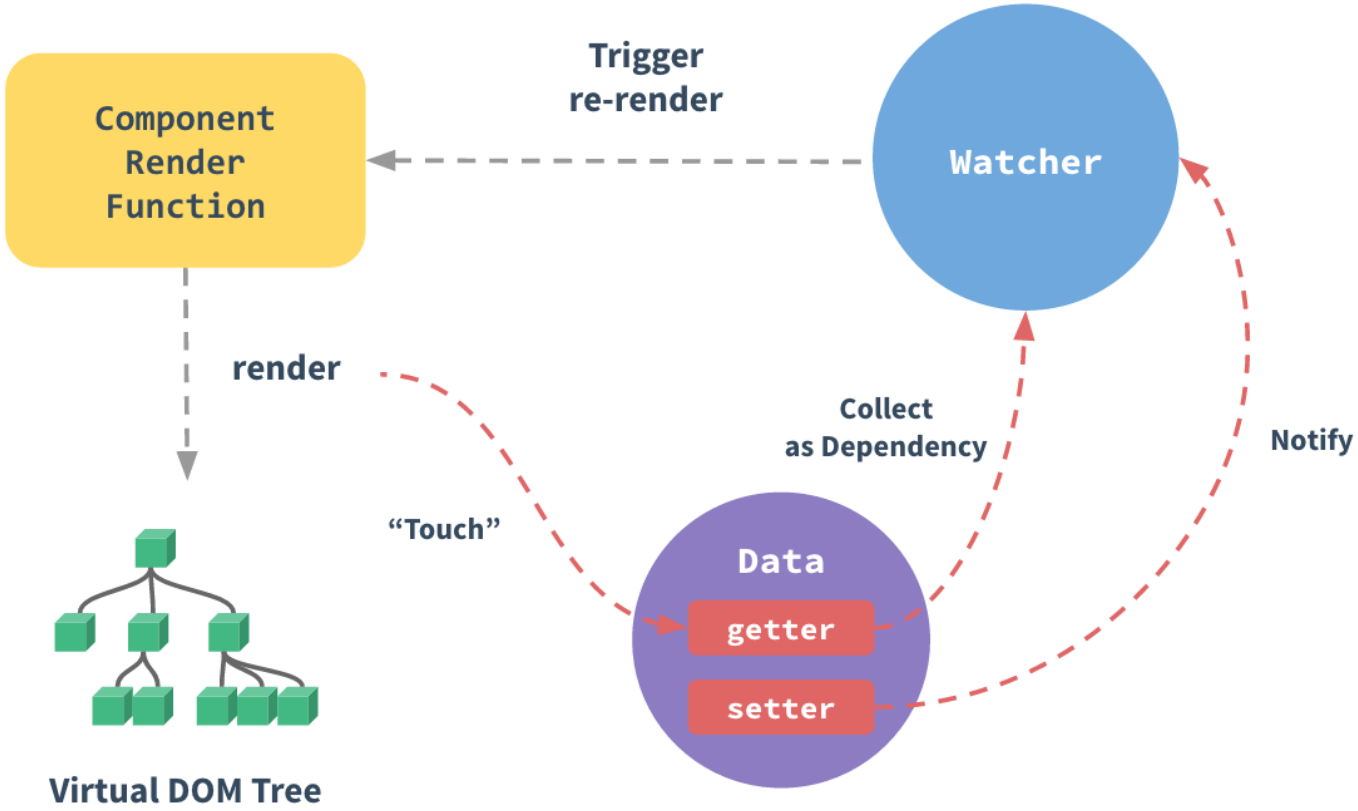
What is Vue.js

- A **progressive framework** for building user interfaces.
- Created by evan you when he was working at Google Creative Labs in 2013.
- Pronounced /vjuː/, like **view!**

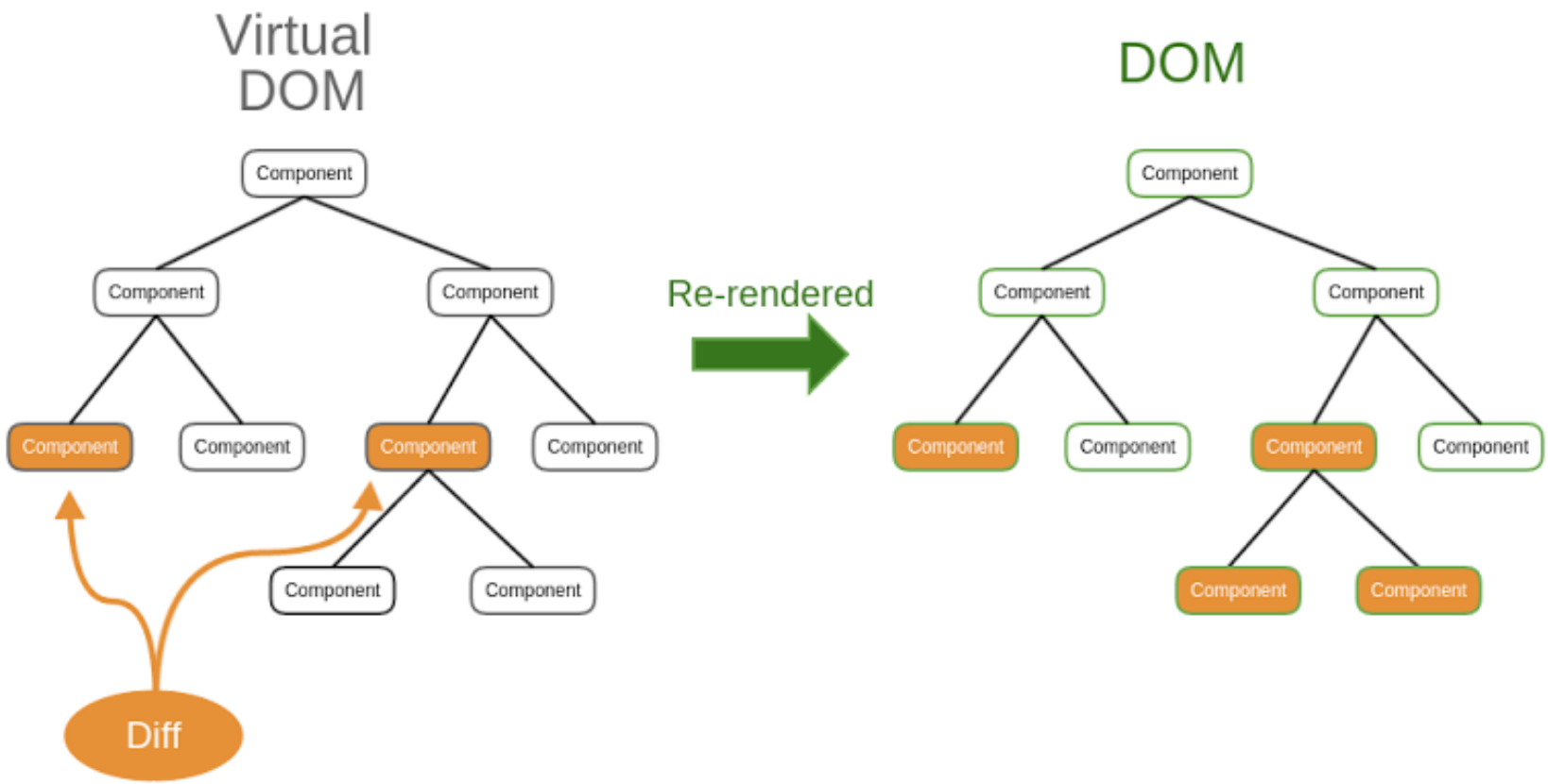
MVVM Architecture Pattern



Reactivity


Trigger
re-render
Component
Render
Function
Watcher
render
Collect
as Dependency
Notify
"Touch"
Data
getter
setter
Virtual DOM Tree

Virtual DOM


Virtual DOM
Component
Component
Component
Component
Component
Component
Component
Component
Component
Diff
Re-rendered
DOM
Component
Component
Component
Component
Component
Component
Component
Component
Component

Outline

- Introduction to Vue.js
- The Vue instance
- Declarative Rendering
- Event Listeners & Input handling
- v-if and v-for Directives
- Computed Props
- Components

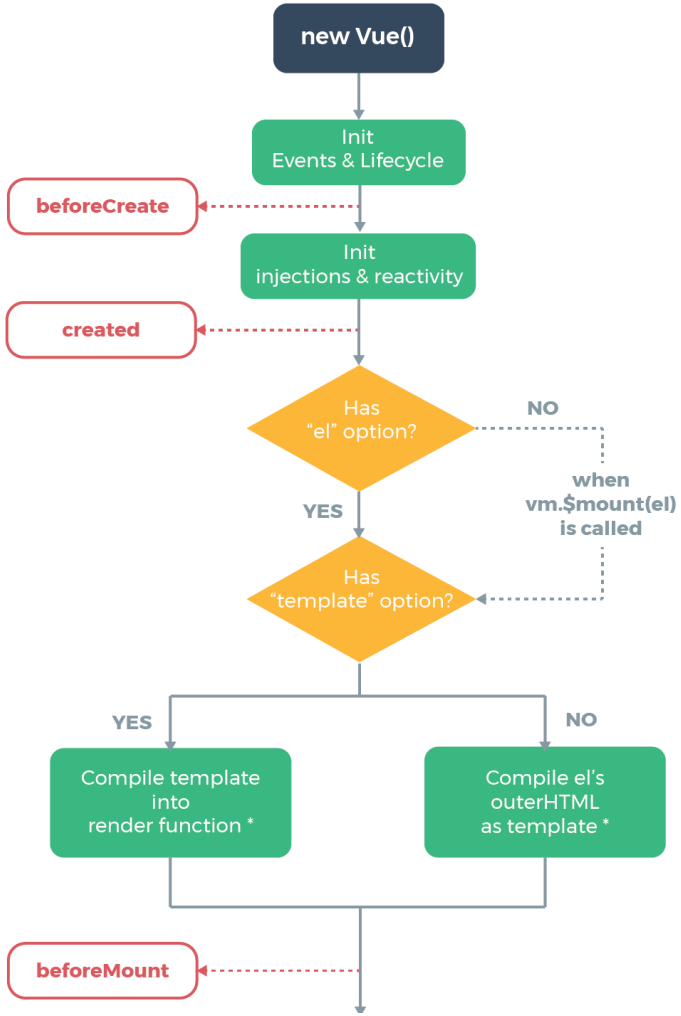
The Vue Instance

- Every Vue application starts by creating a new **Vue instance** with the Vue function:

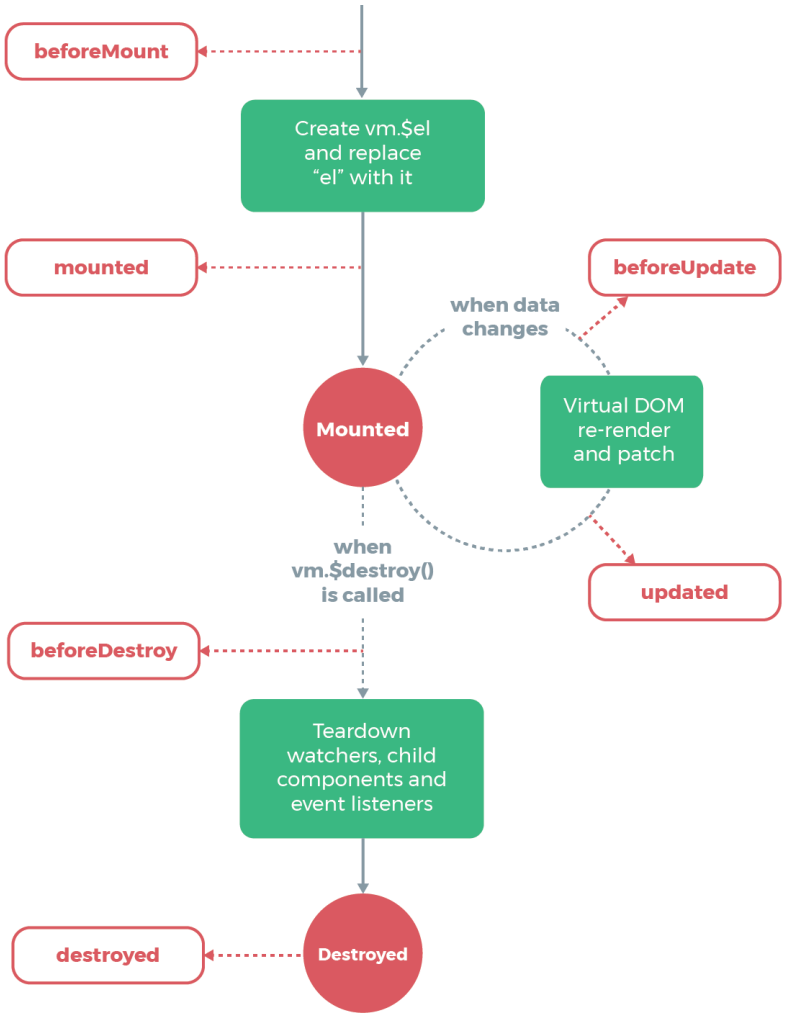
```
<script>
var vm = new Vue({
  // options
})
</script>
```

- As a convention, we often use the variable vm (short for ViewModel) to refer to our Vue instance.
- A Vue application consists of a **root Vue instance** created with new Vue, optionally organized into a tree of nested, reusable components.

Vue Lifecycle Diagram



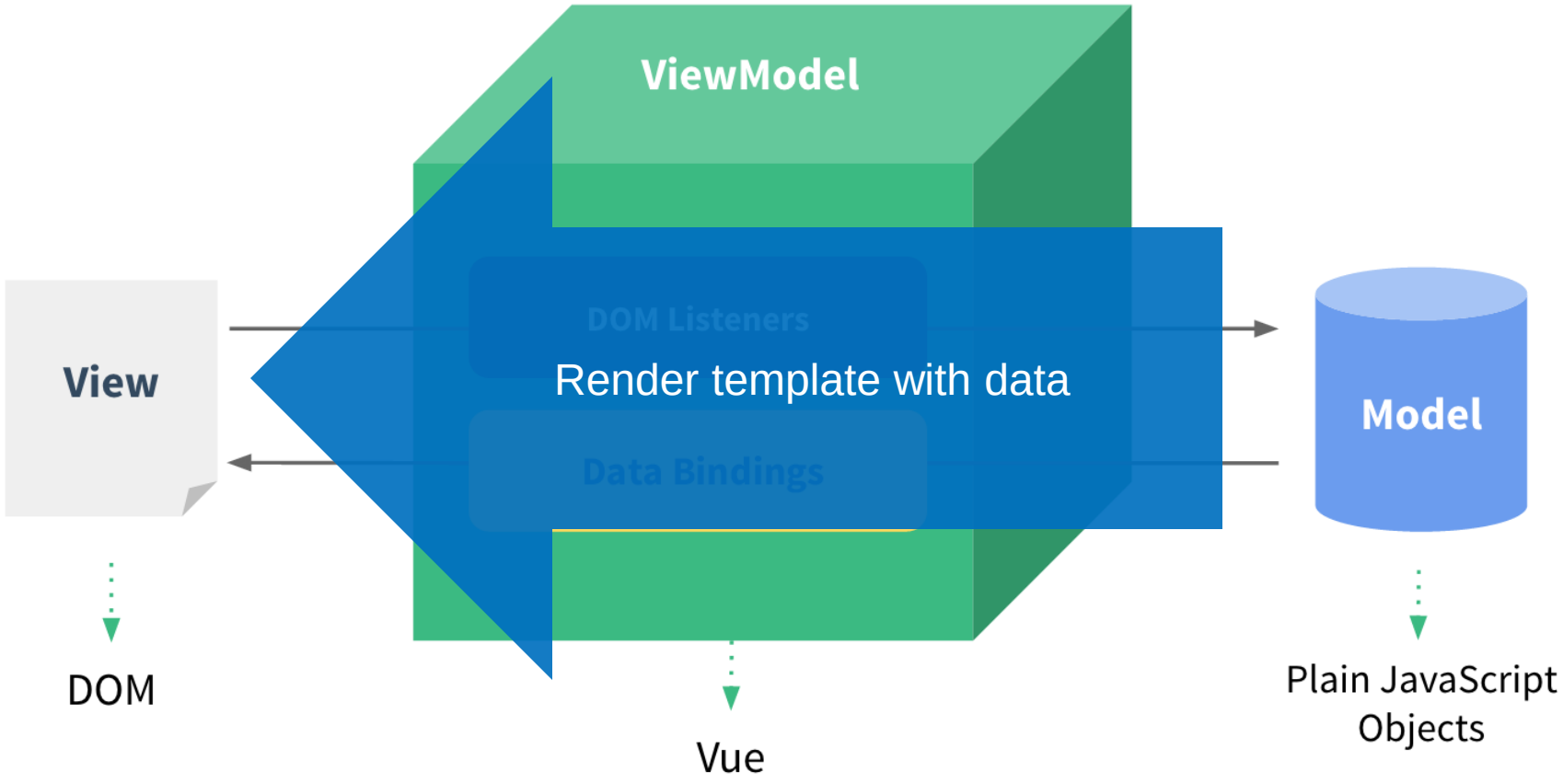
Vue Lifecycle Diagram



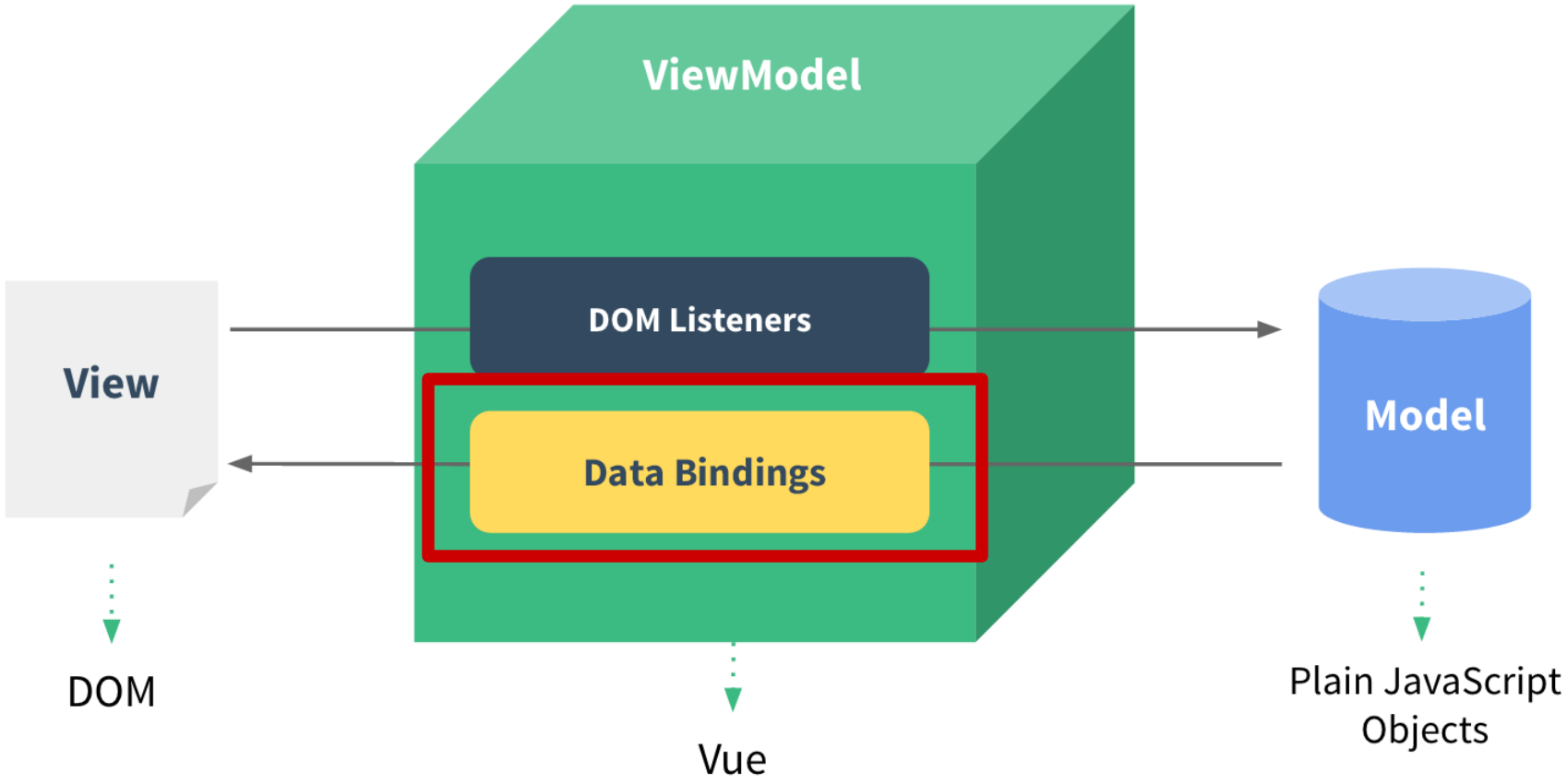
Outline

- Introduction to Vue.js
- The Vue instance
- Declarative Rendering
- Event Listeners & Input handling
- v-if and v-for Directives
- Computed Props
- Components

Declarative Rendering



Data Bindings



Data Bindings

```
<div id="app">
{{ message }}
</div>

<script>
new Vue({
el: '#app',
data: {
  message: 'Hello World!'
}
})
</script>
```

https://codepen.io/pi0/pen/pdKxKZ

Binding attributes

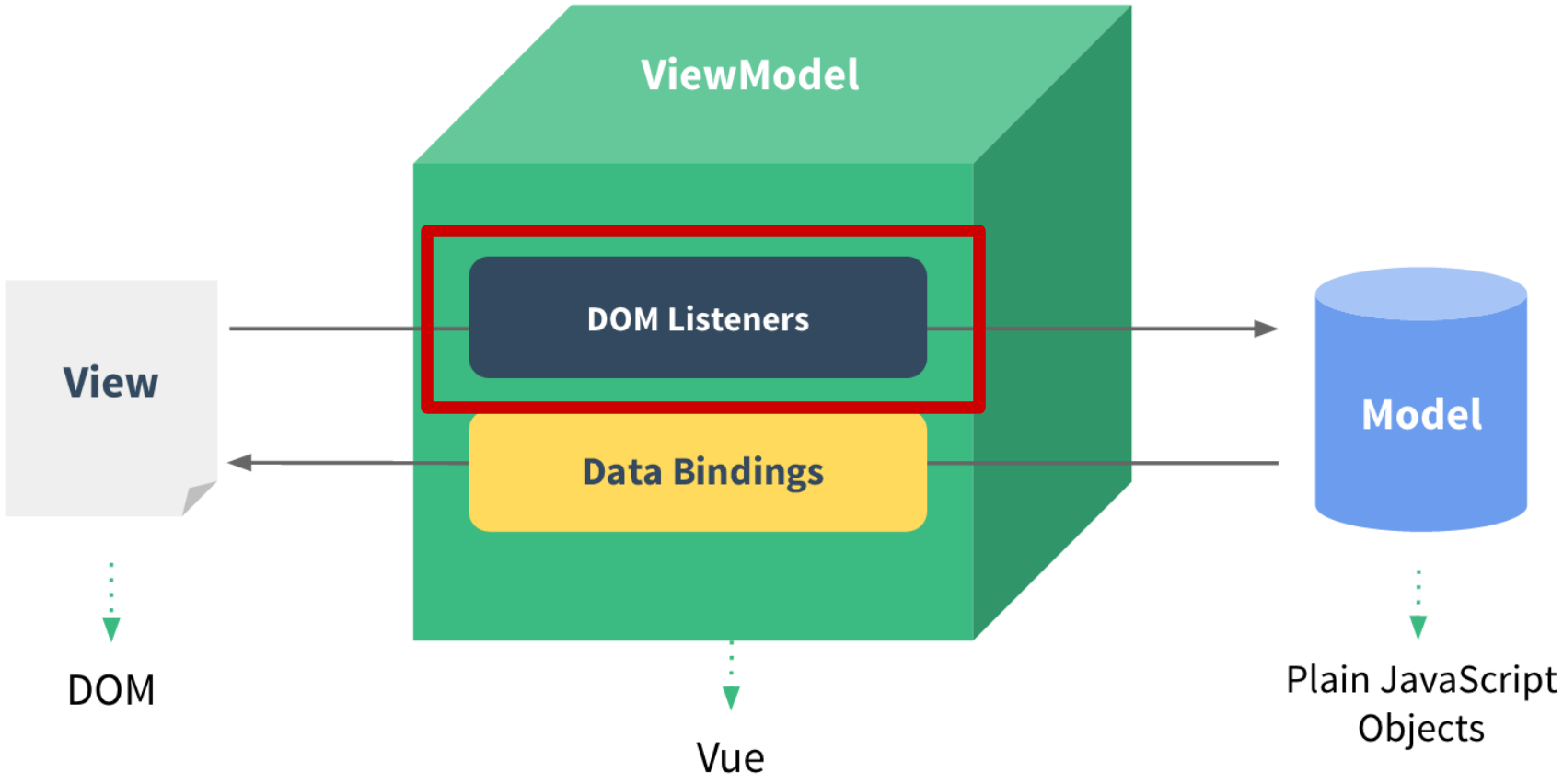
```
<div id="app">
  <img :src="imageSrc" />
</div>

<script>
new Vue({
el: '#app',
data: {
  imageSrc: 'https://lorempixel.com/300/150'
}
})
</script>
```

https://codepen.io/pi0/pen/mqKzgm

Outline

- Introduction to Vue.js
- The Vue instance
- Declarative Rendering
- Event Listeners & Input handling
- v-if and v-for Directives
- Computed Props
- Components

DOM Listeners



DOM Listeners

```
<div id="app">
  <button @click="clicked">Click Me</button>
</div>


<script>
new Vue({
el: '#app',
methods: {
  clicked () {
    alert("HEEY!")
  }
}
})
</script>
```

https://codepen.io/pi0/pen/aVKRPo

Why using listeners in HTML?

- All Vue handler functions and expressions are **strictly** bound to the ViewModel.
- It's easier to **locate the handler** function implementations within your JS code.
- ViewModel code can be **pure logic** and **DOM-free**. This makes it easier to test.
- When a ViewModel is destroyed, all event listeners are **automatically removed**. You don't need to worry about cleaning it up.

Handling User Input

```
<div id="app">
Your name: <input @input="onInput"></input>

Welcome {{ name }}!
</div>

<script>
new Vue({
el: '#app',
data: {
  name: 'Guest User'
},
methods: {
  onInput(e) {
    this.name = e.target.value
  }
}
})
</script>
```

https://codepen.io/pi0/pen/rYKqEE

Handling User Input (v-model)

```
<div id="app">
Your name: <input v-model="name"></input>

Welcome {{ name }}!
</div>

<script>
new Vue({
el: '#app',
data: {
  name: 'Guest User'
}
})
</script>
```

https://codepen.io/pi0/pen/xPzQZZ

Outline

- Introduction to Vue.js
- The Vue instance
- Declarative Rendering
- Event Listeners & Input handling
- v-if and v-for Directives
- Computed Props
- Components

Conditional rendering (v-if)

```
<div id="app">
<input type="checkbox" id="accept" v-model="accepted">
<label for="accept">I accept terms of use</label>

<p v-if="!accepted">Please accept terms!</p>
<p v-else>Thank you!</p>
</div>

<script>
new Vue({
el: '#app',
data: {
  accepted: false
}
})
</script>
```

https://codepen.io/pi0/pen/LOrXWx

List rendering (v-for)

```
<div id="app">
<ul>
 <li v-for="course in courses">
 <input type="checkbox" v-model="selectedCourses" :value="course" />{{ course }}
</li>
</ul>
{{ selectedCourses }}
</div>

<script>
new Vue({
el: '#app',
data: {
  selectedCourses: [],
  courses: ['IE', 'ML', 'BP', 'AP']
}
})
</script>
```

https://codepen.io/pi0/pen/xPzQLE

Outline

- Introduction to Vue.js
- The Vue instance
- Declarative Rendering
- Event Listeners & Input handling
- v-if and v-for Directives
- Computed Props
- Components

Computed props

- Putting too much logic in your templates can make them bloated and hard to maintain and not declarative.
- Computed props are being **cached**.
- Dependencies will be **auto tracked.**

Computed props

```
<div id="app">
<input v-model="a"> * <input v-model="b"> = {{ result }}
</div>

<script>
new Vue({
el: '#app',
data: {
  a: 10,
  b: 10
},
computed: {
  result() {
    return this.a * this.b
  }
}
})
</script>
```

https://codepen.io/pi0/pen/xPzQYR

Outline

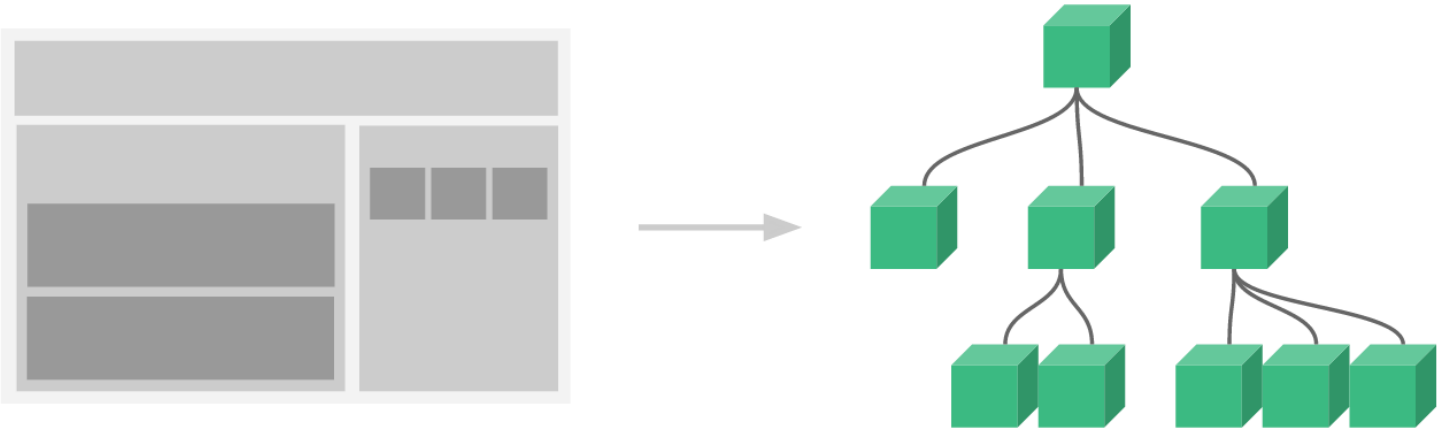
- Introduction to Vue.js
- The Vue instance
- Declarative Rendering
- Event Listeners & Input handling
- v-if and v-for Directives
- Computed Props
- Components

Vue Components

- In Vue, a component is essentially a Vue instance with pre-defined options.
- They help you extend basic HTML elements to encapsulate reusable code.
- Components are one of the most powerful features of Vue.

Composing with Components

- Component are an abstraction that allows us to build large-scale applications composed of small, self-contained, and often reusable components.



Components for large scale apps

- An (imaginary) example of what an app's template might look like with components:

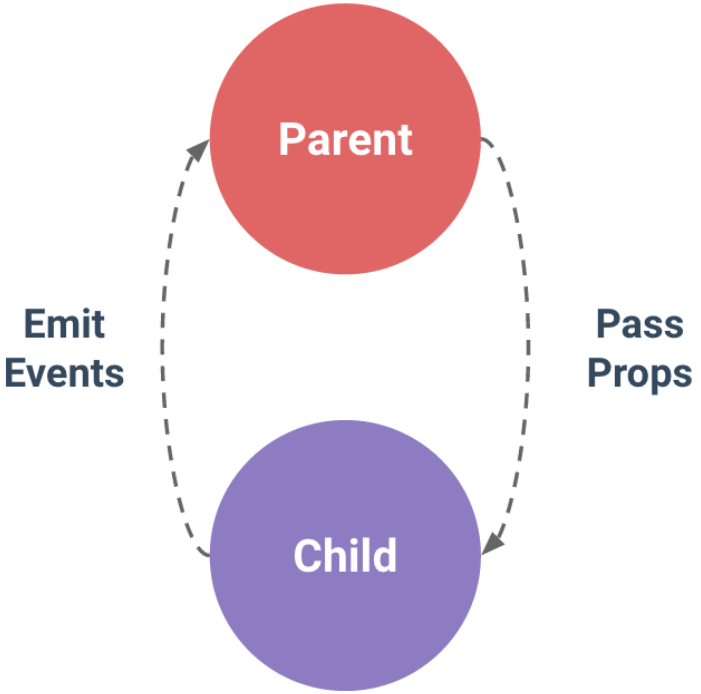
```html
<div id="app">
  <app-nav></app-nav>
  <app-view>
    <app-sidebar></app-sidebar>
    <app-content></app-content>
  </app-view>
</div>
```

Composing Components

- Components are meant to be used together, most commonly in parent-child relationships.
- In Vue, the parent-child component relationship can be summarized as **props down, events up**.
- This prevents child components from accidentally mutating the parent's state, which can make your app's data flow harder to understand.

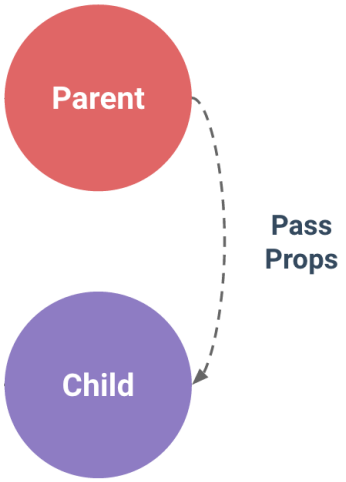
One-Way Data Flow

- Parent passes data down to the child via **props**, and the child sends messages to the parent via **events.**



Component Props

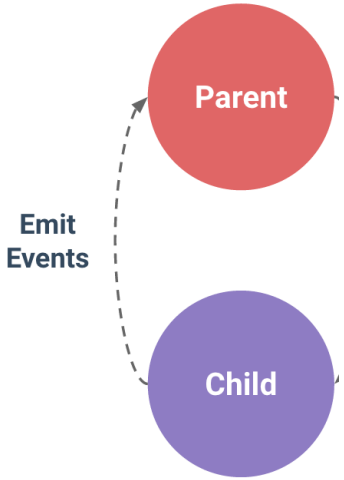
- Every component instance has its own **isolated scope.**
- Data can be **passed down** to child components using **props**.
- A prop is a custom attribute for passing information from parent components.




```html
<child message="hello!"></child>
```

Custom Events

- Every Vue instance implements an **events interface**, which means it can:
  - Trigger an event using $emit(eventName)
  - Listen to an event using
    - $on(eventName)
    - v-on:eventName
    - @eventName




```html
<div id="counter-event-example">
  <p>{{ total }}</p>
  <button-counter v-on:increment="incrementTotal"></button-counter>
  <button-counter v-on:increment="incrementTotal"></button-counter>
</div>
```

Declare a component

- In Vue, a component is essentially a Vue instance with pre-defined options:

```js
// Define a new component called todo-item
Vue.component('todo-item', {
  template: '<li>This is a todo</li>'
})
```

```js
Vue.component('todo-item', {
  // The todo-item component now accepts a
  // "prop", which is like a custom attribute.
  // This prop is called todo.
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})
```

Declare a component

- In more complex projects or when your frontend is entirely driven by JavaScript, these disadvantages become apparent:
- **Global definitions**
  - Force unique names for every component
- **String templates**
  - Lack syntax highlighting and require ugly slashes for multiline HTML
- **No CSS support**
  - While HTML and JavaScript are modularized into components, CSS is conspicuously left out

Single File Components

- In modern UI development, we have found that instead of dividing the codebase into three huge layers (HTML/CSS/JS) that interweave with one another, it makes much more sense to divide them into loosely-coupled components and compose them.
- This is possible using **single-file components** with a .vue extension.

Single File Components

Hello.vue

```vue
<template>
  <p>{{ greeting }} World!</p>
</template>

<script>
module.exports = {
  data: function () {
    return {
      greeting: 'Hello'
    }
  }
}
</script>

<style scoped>
p {
  font-size: 2em;
  text-align: center;
}
</style>
```

Line 21, Column 1 Spaces: 2 Vue Component

References

- http://singlepageappbook.com (2013 - Mikito Takada)
- The Majesty of Vue.js 2 (2017- Alex Kyriakidis, Kostas Maniatis and Evan You)
- https://developers.google.com/web/fundamentals/web-components/customelements
- https://developers.google.com/web/fundamentals/web-components/shadowdom