Modern web applications

Internet Engineering

Spring 2018

Pooya Parsa

Professor: Bahador Bakhshi

CE & IT Department, Amirkabir University of Technology

Outline

- Modern Web Apps
- Web components Spec

Modern Web Apps

- We still talk about programming as if typing in the code was the hard part. It's not - the hard part is maintaining the code.
- To write maintainable code, we need to keep things simple.

Modern Web Apps

- **Architecture**
  - What (conceptual) parts does our app consist of?
  - How do the different parts communicate with each other?
  - How do they depend on each other?

Modern Web Apps

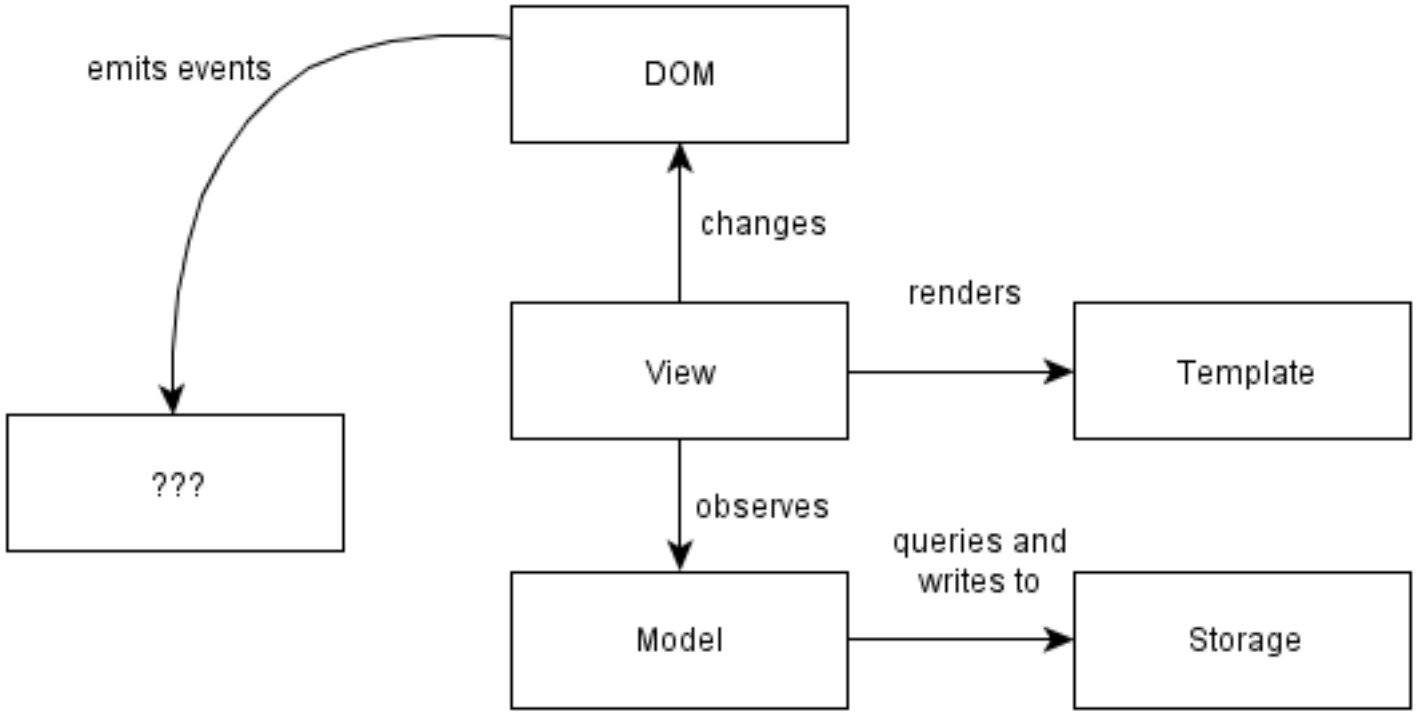
- **Asset packaging**
  - How is our app structured into files and files into logical modules?
  - How are these modules built and loaded into the browser?
  - How can the modules be loaded for unit testing?

Modern Web Apps

- **Run-time state**
  - When loaded into the browser, what parts of the app are in memory?
  - How do we perform transitions between states and gain visibility into the current state for troubleshooting?

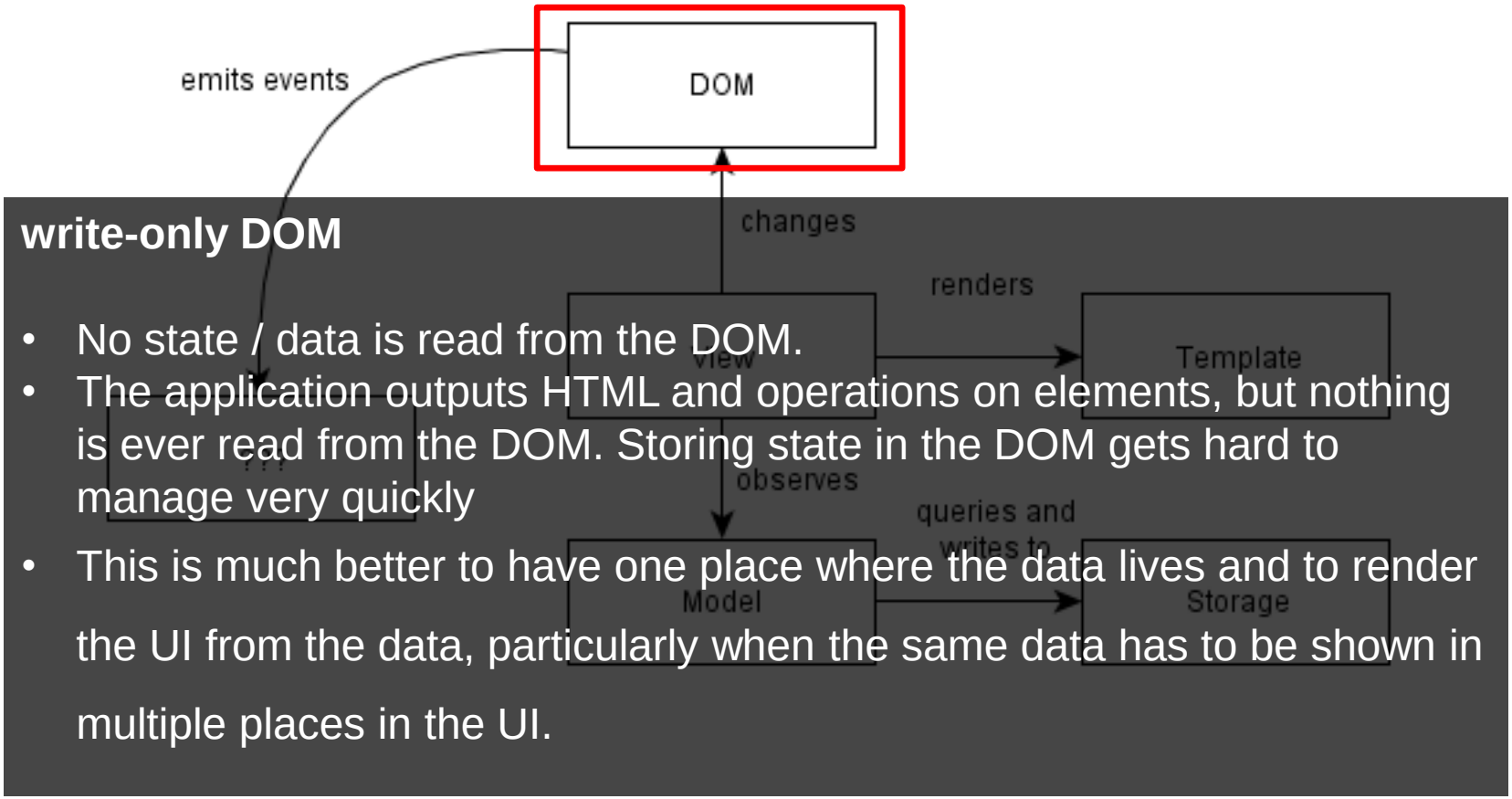
Modern Web Apps

- A modern web application architecture



Modern Web Apps

- A modern web application architecture

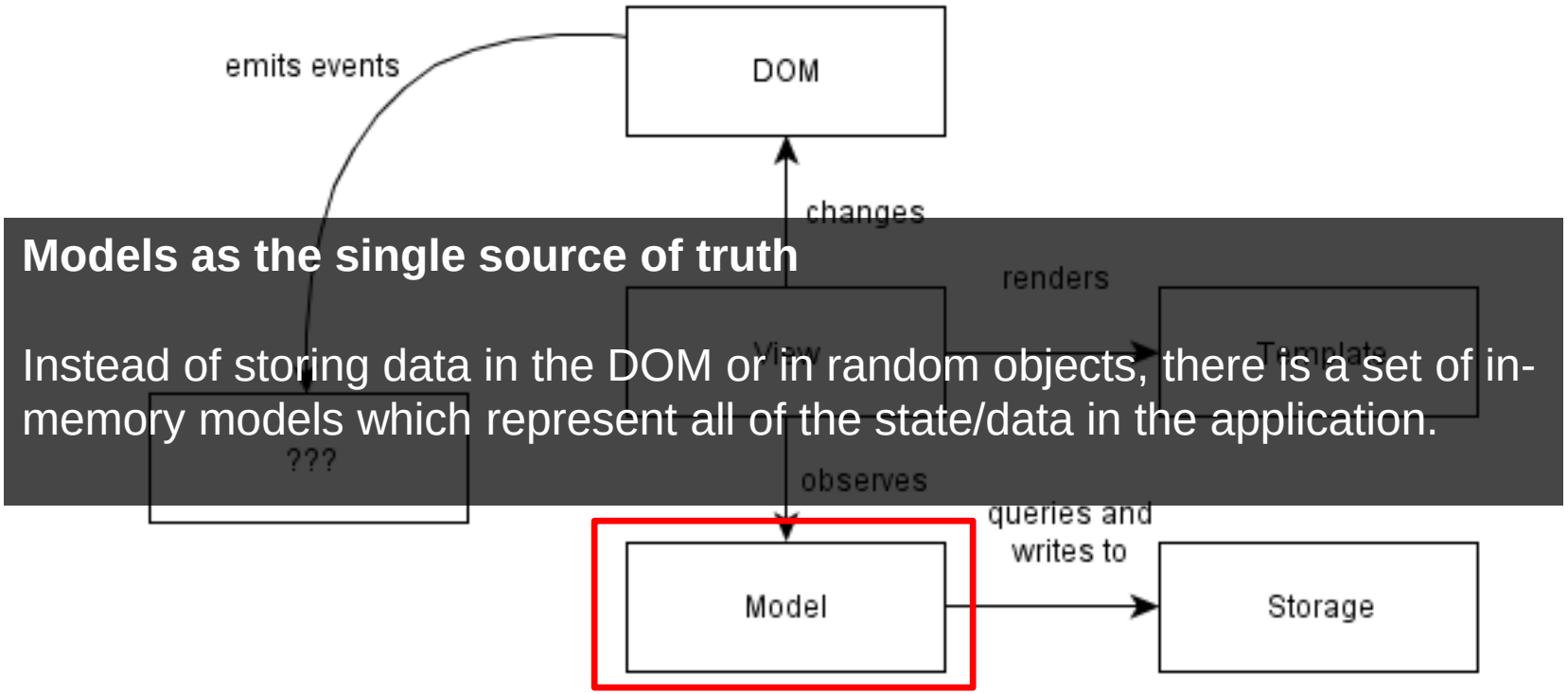


write-only DOM

- No state / data is read from the DOM.
- The application outputs HTML and operations on elements, but nothing is ever read from the DOM. Storing state in the DOM gets hard to manage very quickly
- This is much better to have one place where the data lives and to render the UI from the data, particularly when the same data has to be shown in multiple places in the UI.

Modern Web Apps

- A modern web application architecture



Models as the single source of truth

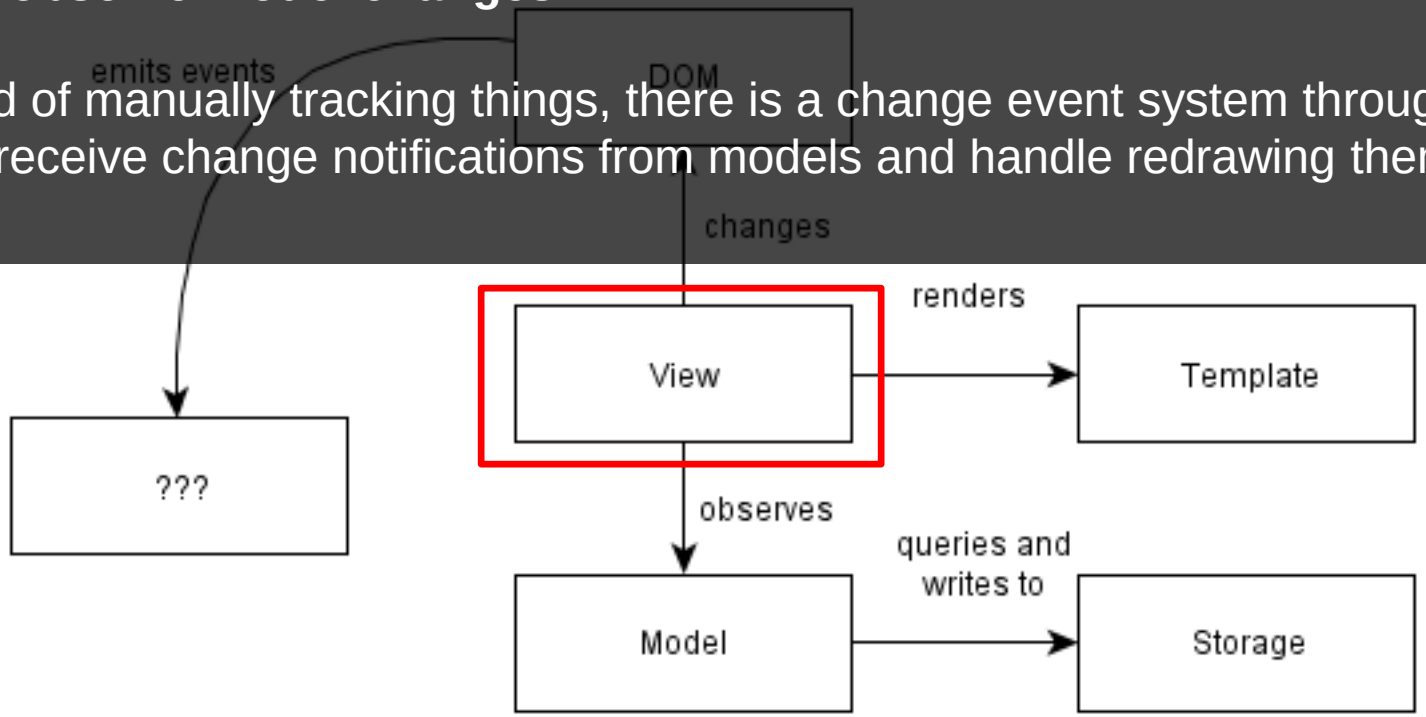
Instead of storing data in the DOM or in random objects, there is a set of in-memory models which represent all of the state/data in the application.

Modern Web Apps

- A modern web application architecture

Views observe model changes.

Instead of manually tracking things, there is a change event system through which views receive change notifications from models and handle redrawing themselves.



Single-page applications

- Allow us to offer a more-native-app-like experience to the user.
- Rich interactions with multiple components on a page.
- Server-side rendering is hard to implement for all the intermediate states.

Minimizing DOM dependent-code

- Any code that depends on the DOM needs to be tested for cross-browser compatibility
- Incompatibilities are in the DOM implementations, not in the JavaScript implementations
  - So it makes sense to minimize and isolate DOM-dependent code.

Outline

- Modern Web Apps
- Web components Spec

Web Components Spec

- Shadow DOM
- Custom Elements
- HTML Templates
- HTML Imports

Shadow DOM

- **W3** standard spec
- *Ability of the browser to include a subtree of DOM elements*
- Encapsulates the implementation.
- **Scoped** styles to the web platform
  - Bundle CSS with markup
- Self-contained **components** in **vanilla JavaScript**.

Shadow DOM (Example)

```
<h1>Light DOM</h1>
<div id="app"></div>

<script>
// Select #app element
const app = document.querySelector('#app')

// Attach a shadow root to element
const shadowRoot = app.attachShadow({mode: 'open'})

// Insert scoped style and content
shadowRoot.innerHTML = `
<style>* { color: blue }</style>
<h1>Shadow DOM</h1>
`
</script>
```

Light DOM

Shadow DOM

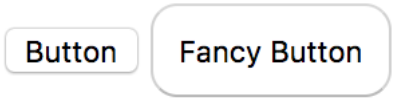
https://codepen.io/pi0/pen/WXdKYj

Custom Elements

- Custom elements give us a new tool for **defining new HTML tags** in the browser
- Brings a web standards-based way to create **reusable components** using nothing more than vanilla JS/HTML/CSS
- The API is the foundation of **web components**

Custom Elements (Example)

```
<button>Button</button>
<fancy-button>Fancy Button</fancy-button>
<script>
class FancyButton extends HTMLElement {
  constructor () {
    super()
    // Attach a shadow root to <fancy-button>
    const shadowRoot = this.attachShadow({mode: 'open'})
    shadowRoot.innerHTML = `
      <!-- styles are scoped to fancy-h2! -->
      <style>
      button {
        padding: 10px;
        border-radius: 10px;
      }
      </style>
      <button><slot></slot></button>
    `
  }
}
// Define <fancy-button> custom element
customElements.define('fancy-button', FancyButton)
</script>
```



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

Custom Elements (Example)

Example - extending `<button>` :

```
// See https://html.spec.whatwg.org/multipage/indices.html#element-interfaces
// for the list of other DOM interfaces.
class FancyButton extends HTMLButtonElement {
  constructor() {
    super(); // always call super() first in the ctor.
    this.addEventListener('click', e => this.drawRipple(e.offsetX, e.offsetY));
  }

  // Material design ripple animation.
  drawRipple(x, y) {
    let div = document.createElement('div');
    div.classList.add('ripple');
    this.appendChild(div);
    div.style.top = `${y - div.clientHeight/2}px`;
    div.style.left = `${x - div.clientWidth/2}px`;
    div.style.backgroundColor = 'currentColor';
    div.classList.add('run');
    div.addEventListener('transitionend', e => div.remove());
  }
}

customElements.define('fancy-button', FancyButton, {extends: 'button'});
```

Custom Elements

- Cross-browser (web standard) for creating and extending reusable components.
- Requires no library or framework to get started. Vanilla JS/HTML FTW!
- Provides a familiar programming model. It's just DOM/CSS/HTML.
- Works well with other new web platform features (Shadow DOM, <template>, CSS custom properties, etc.)
- Tightly integrated with the browser's DevTools.
- Leverage existing accessibility features.

References

- http://singlepageappbook.com (2013 - Mikito Takada)
- https://developers.google.com/web/fundamentals/web-components/customelements
- https://developers.google.com/web/fundamentals/web-components/shadowdom