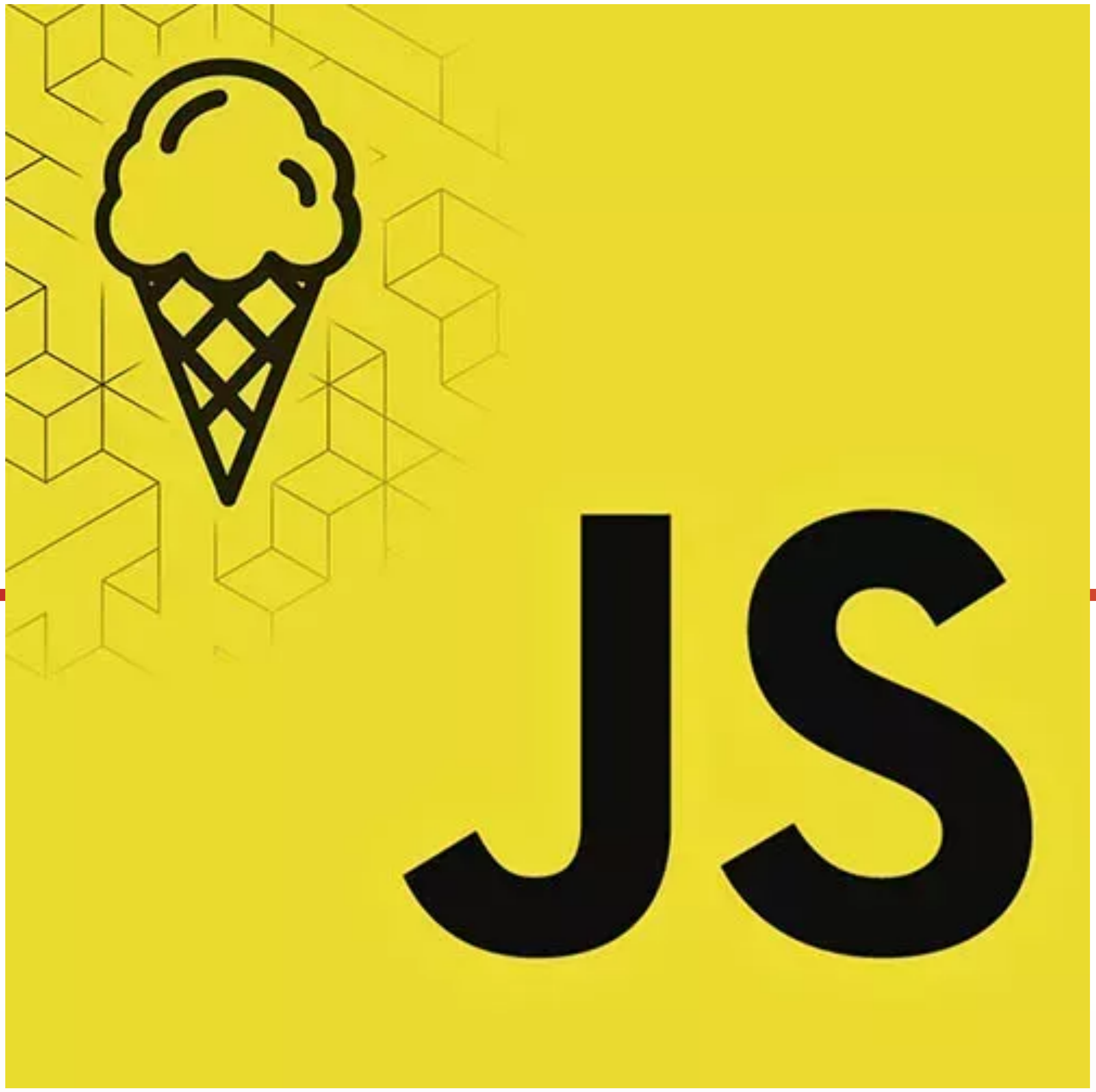


Practical Web App Patterns

with Vanilla JS

Maximiliano Firtman

firt.dev

x.com/firt

linkedin.com/in/firtman

github.com/firtman

About me

Maximiliano Firtman

AJAX
PROGRAMACIÓN PARA CELULARES CON JAVA
DESARROLLOS MÓVILES CON .NET
ASP.NET
Flex 4.5
Visual Studio
Programming the Mobile Web
Programming the Mobile Web
High Performance Mobile
jQuery Mobile: Up and Running
High Performance Mobile Web
移动Web编程
jQuery Mobile
jQuery Mobile
jQuery 모바일
jQuery Mobile: Up and Running
Mobil Web Programlama
Веб-программирование
移动网络程序设计
Programmare per il Web Mobile
Mobile Web 程式設計
WEB MOBILE
O'REILLY
PEARSON

MOBILE+WEB DEVELOPER

HTML since 1996

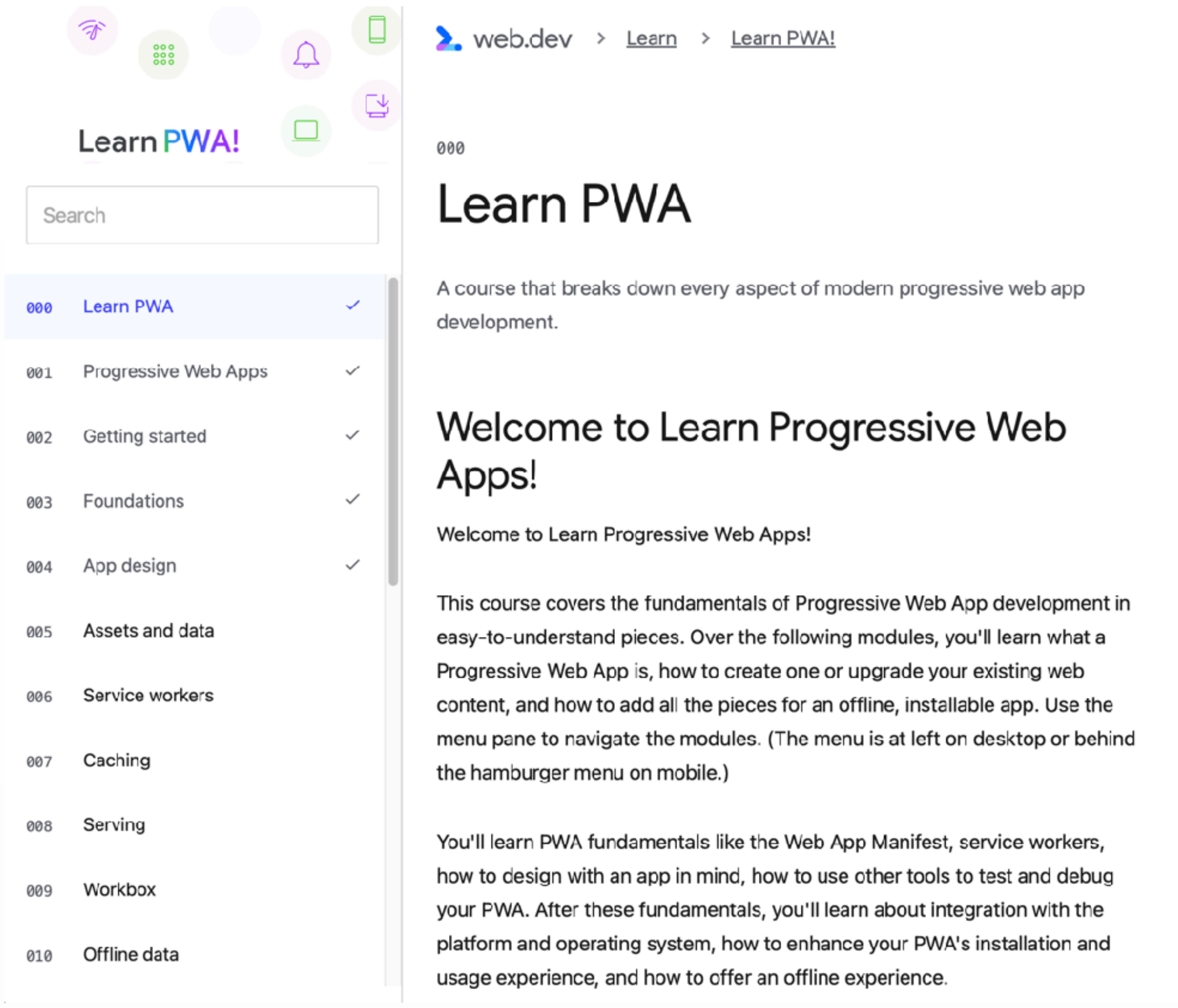
JavaScript since 1998, +150 web apps

AUTHOR

Authored 13 books, +70 courses

About me

Maximiliano Firtman



web.dev > Learn > Learn PWA!

Learn PWA!

Search

000	Learn PWA	✓
001	Progressive Web Apps	✓
002	Getting started	✓
003	Foundations	✓
004	App design	✓
005	Assets and data	
006	Service workers	
007	Caching	
008	Serving	
009	Workbox	
010	Offline data	

000

Learn PWA

A course that breaks down every aspect of modern progressive web app development.

Welcome to Learn Progressive Web Apps!

Welcome to Learn Progressive Web Apps!

This course covers the fundamentals of Progressive Web App development in easy-to-understand pieces. Over the following modules, you'll learn what a Progressive Web App is, how to create one or upgrade your existing web content, and how to add all the pieces for an offline, installable app. Use the menu pane to navigate the modules. (The menu is at left on desktop or behind the hamburger menu on mobile.)

You'll learn PWA fundamentals like the Web App Manifest, service workers, how to design with an app in mind, how to use other tools to test and debug your PWA. After these fundamentals, you'll learn about integration with the platform and operating system, how to enhance your PWA's installation and usage experience, and how to offer an offline experience.

About me

Maximiliano Firtman

FrontendMasters

Frontend Courses

Mobile App Courses

Backend Courses

What we will cover today

Design Patterns

Apply many on three projects

Classic Design Patterns applied in JS

Single Page Applications

Multi Page Applications

Data and State Management

Some other ideas

Pre-requisites

JavaScript experience

Vanilla JavaScript basic concepts

A web browser

A code editor

Workshop Repo

firtman.github.io/webapp-patterns

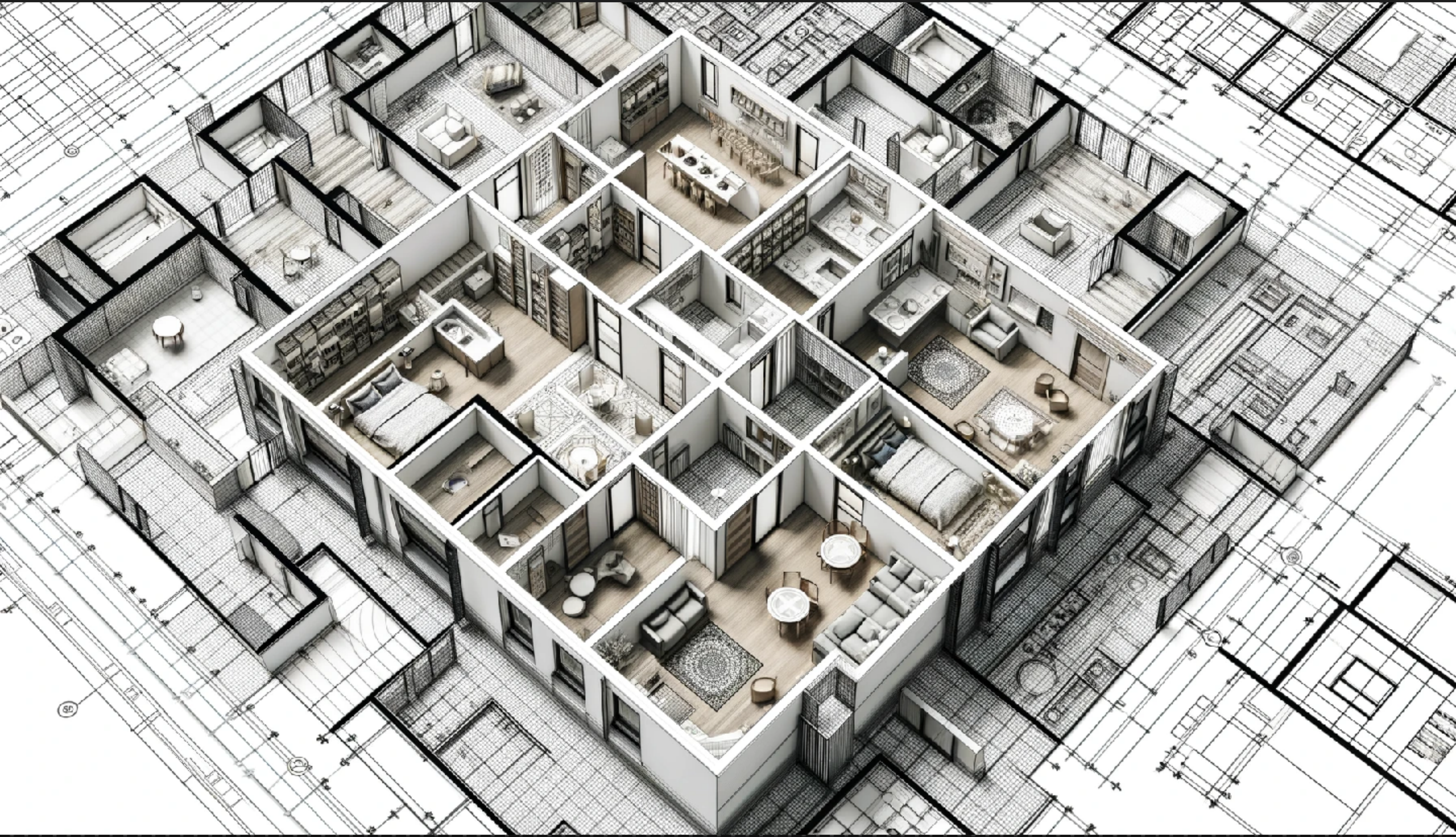
Questions?

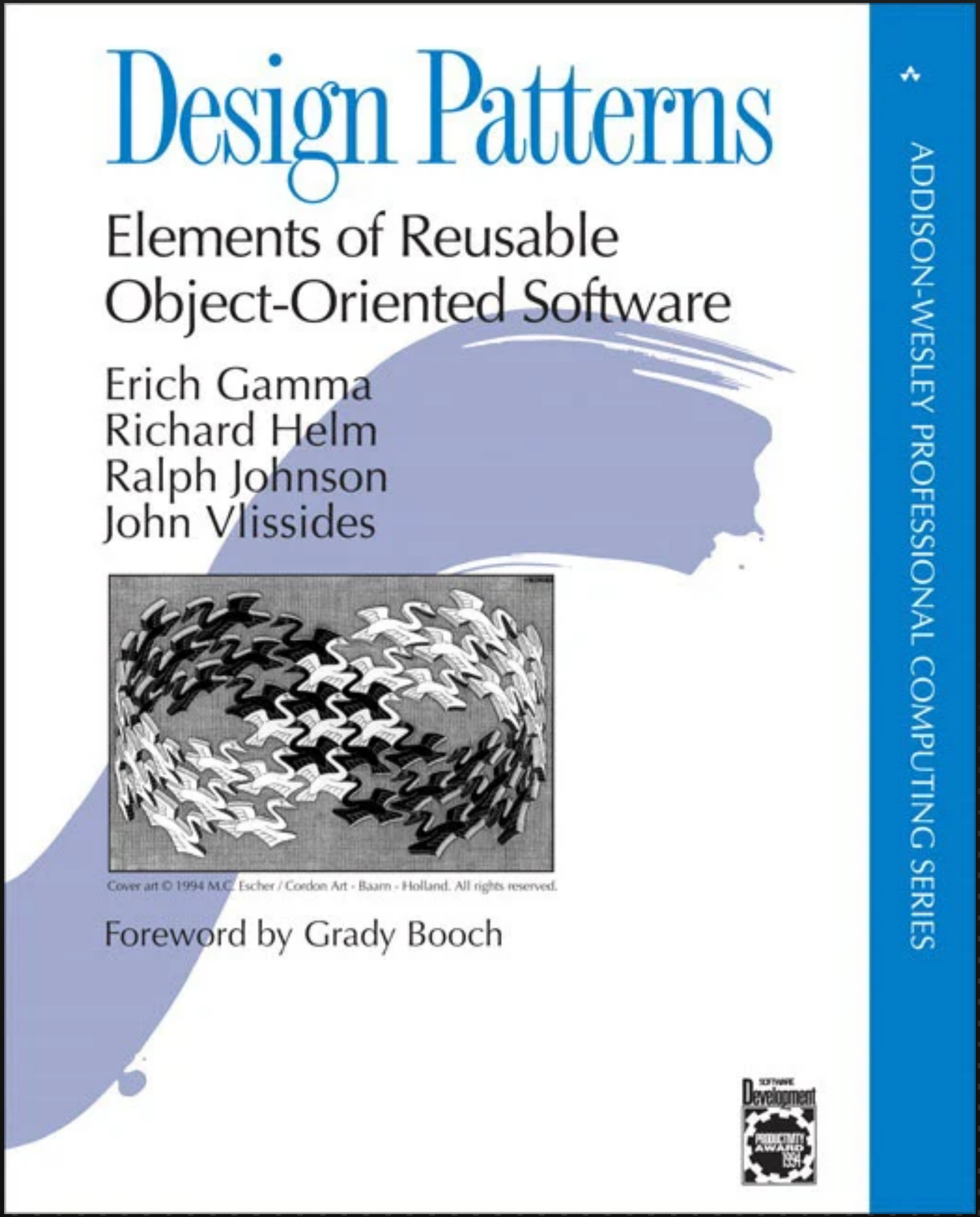
Introduction

Definition

Design Pattern

A design pattern is a reusable template for solving common software design problems, enhancing code readability and efficiency and creating a common vocabulary.





Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Components of a Design Pattern

- Name
- Problem
- Solution
- Context
- Consequences
- Examples

Why is it important for Vanilla JS projects?

- We have complete freedom

- We need to set guidelines to improve:
  - Reusability
  - Scalability
  - Consistency
  - Efficiency
  - Debugging

Idea

Anyone can create a design pattern. It typically starts as a blog post or an article setting a name and explaining the problem and the solution that was already implemented in a real-world example.

Warning

Don't use design patterns just because it sounds cool.

Failures while using design patterns

- Overengineering
- Misapplication
- Inflexibility
- Learning Curve for the team
- Complexity
- Performance Overhead

Definition

Antipattern

Practices that may initially seem beneficial but ultimately lead to poor outcomes. They are typically counterproductive and can introduce issues such as increased complexity, decreased performance, and maintainability problems.

Important

You probably know many design patterns even if you don't recognize them initially as that

Definition

Vanilla JavaScript

The usage of the core language and browser APIs to create web apps without any additional libraries or frameworks added on top

Vanilla JS: You Might Not Need a Framework

Frontend Masters Course

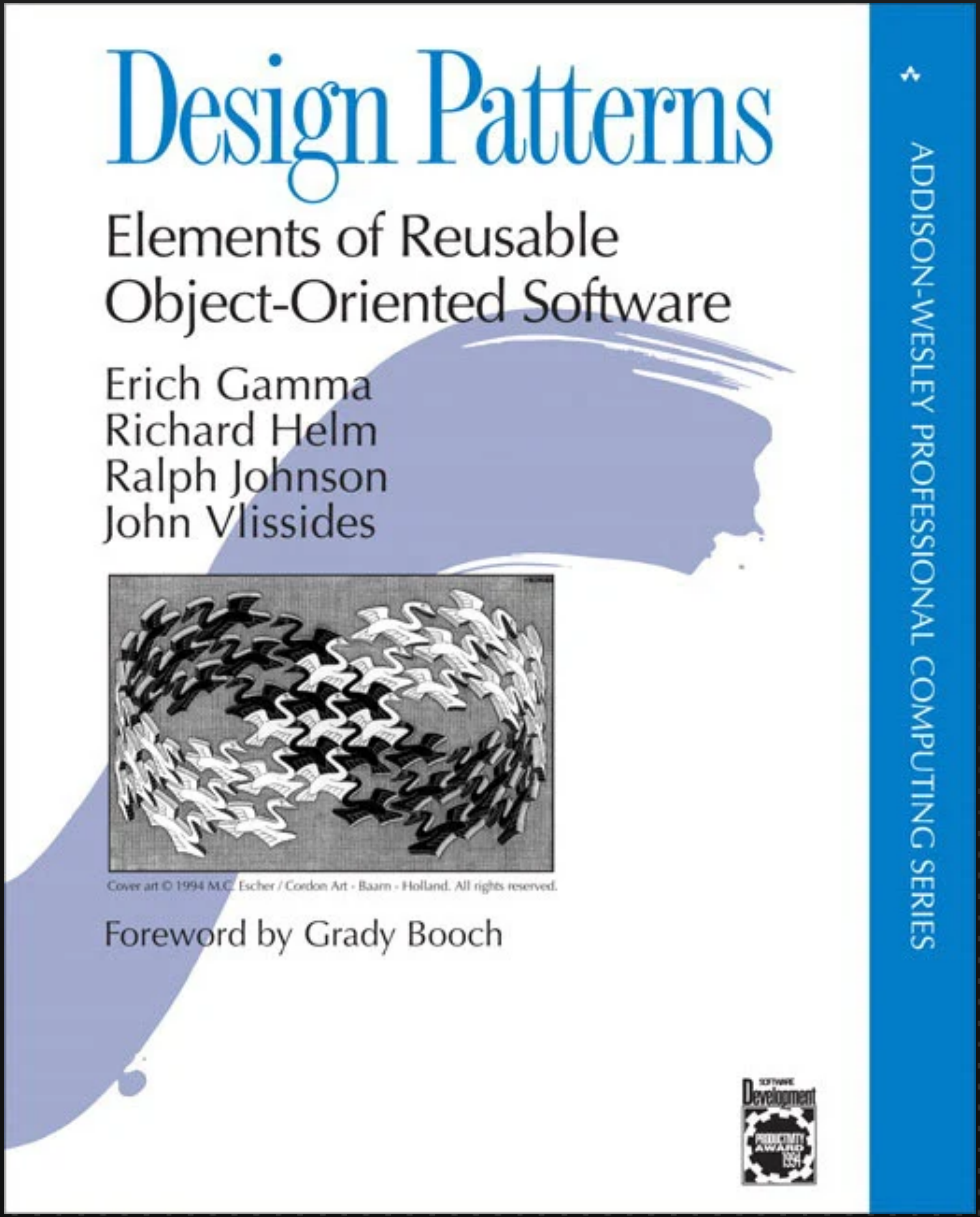
Design Patterns for VanillaJS Web Apps

- Classic
- Web Specific
- Single Page Apps
- Multi Page Apps
- Data and State Management

Warning

We won't cover all the design patterns available.

Classic Patterns in JavaScript



Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Classic Patterns

- Typically around OOP solutions
- They are categorized in
  - Creational
  - Structural
  - Behavioral
- In JavaScript (<= ES5) there were many design patterns that

Important

In JavaScript there are many ways to implement the same design pattern, thanks to the dynamic nature of the language

Warning

Don't implement design patterns as if you are writing Java. Use the ideas and implement them with the power of JavaScript.

Definition

Creational Design Patterns

They aim to solve the problems associated with creating objects in a way that enhances flexibility and reuse of existing code. The primary purpose of creational patterns is to separate the logic of object creation from the rest of the code.

Singleton

- **Problem to Solve:** Ensure that a class has only one instance and provide a global point of access to it.
- **Solution:** Restrict instantiation of the class to one object and provide a method to access this instance.
- Use Cases:
  - Managing a global configuration object.
  - Database connection pooling.
  - Logging service.
  - State management.

Singleton

app.js

```
const Database = {
   open: async () => {}
   sendQuery: async (query) => {}
};
```

Factory

- **Problem to Solve:** Object creation can become complex and may involve multiple steps, conditional logic, or dependencies.
- **Solution:** The factory pattern encapsulates the object creation process within a separate method or class, isolating it from the rest of the application logic.
- Use Cases:
    - UI element creation
    - Different types of notifications
    - Data Parsers

Factory

app.js

```
class PDFReader extends Reader {}
class CSVReader extends Reader {}
class SQLReader extends Reader {}

class Reader {
   static getReader(url) {
       // based on the return type of the URL
       // we return one of the possible readers
   }
}
```

Definition

Structural Design Patterns

Solutions for composing classes and objects to form larger structures while keeping them flexible and efficient. They focus on simplifying relationships between entities to ensure system maintainability and scalability.

Decorator

- **Problem to Solve:** Add additional functionality to objects dynamically without modifying their structure.
- **Solution:** Wrap an object with another object that adds the desired behavior.
- Use Cases:
  - Adding logging, validation, or caching to method calls.
  - Extending user interface components with additional features.
  - Wrapping API responses to format or process data before passing it on.

Decorator

app.js

```
class Button {
    render() {}
}

class DecoratedButton extends Button {
    render() {
        super.render();
        // Decorating code
    }
}
```

Adapter

- **Problem to Solve:** Allow incompatible interfaces to work together.
- **Solution:** Create an adapter that translates one interface into another that a client expects.
- Use Cases:
  - Integrating third-party libraries with different interfaces into your application.
  - Adapting legacy code to work with new systems or APIs.
  - Converting data formats.

Mixins

- **Problem to Solve:** Share functionality between classes without using inheritance.
- **Solution:** Create a class containing methods that can be used by other classes and apply it to multiple classes.
- Use Cases:
  - Integrating third-party libraries with different interfaces into your application.
  - Adapting legacy code to work with new systems or APIs.
  - Converting data formats.

Mixins

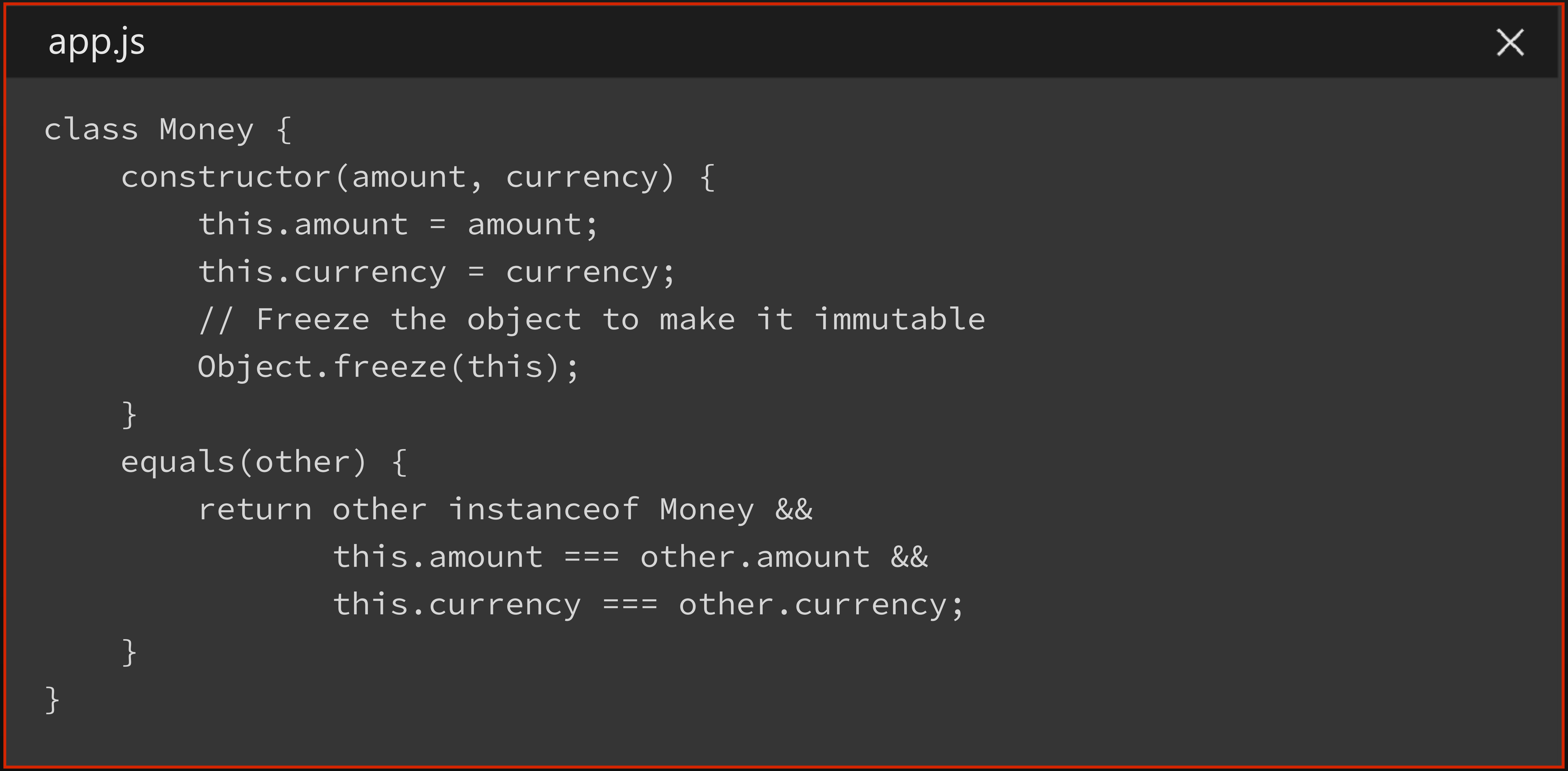
app.js

```javascript
let sayHiMixin = {
  sayHi() { alert(`Hello ${this.name}`); }
};

class User {
  name
}

Object.assign(User.prototype, sayHiMixin);
```

Value Object

- **Problem to Solve:** Represent a value that is immutable and distinct from other objects based on its properties rather than its identity.
- **Solution:** Create a class where instances are considered equal if all their properties are equal and ensure the object is immutable.
- Use Cases:
  - Representing complex data types like money, coordinates, or dates.

Value Object


app.js

```
class Money {
    constructor(amount, currency) {
        this.amount = amount;
        this.currency = currency;
        // Freeze the object to make it immutable
        Object.freeze(this);
    }
    equals(other) {
        return other instanceof Money &&
               this.amount === other.amount &&
               this.currency === other.currency;
    }
}
```

Definition

Behavioral Design Patterns

Deal with object interaction and responsibility distribution. They focus on how objects communicate and cooperate, ensuring that the system is flexible and easy to extend.

Observer

- **Problem to Solve:** Allow an object (subject) to notify other objects (observers) about changes in its state without requiring them to be tightly coupled.
- **Solution:** Define a subject that maintains a list of observers and notifies them of any state changes, typically by calling one of their methods.
- Use Cases:
  - Event handlers
  - Real-time notifications
  - UI updates

Observer

app.js

```
class Subject {
    observers = new Set();
    addObserver(observer) { this.observers.add(observer); }
    removeObserver(observer) { this.observers.delete(observer); }

    notifyObservers(message) {
        this.observers.forEach(observer => observer(message));
    }
}

// Usage
subject1.addObserver(message => console.log("Event fired"));
```

Template Method

- **Problem to Solve:** Define the skeleton of an algorithm that will change on different implementations.
- **Solution:** Create a class with a template method that outlines the algorithm and make subclasses to override specific steps of the algorithm.
- Use Cases:
  - Data Processing
  - Form Validation

Template Method

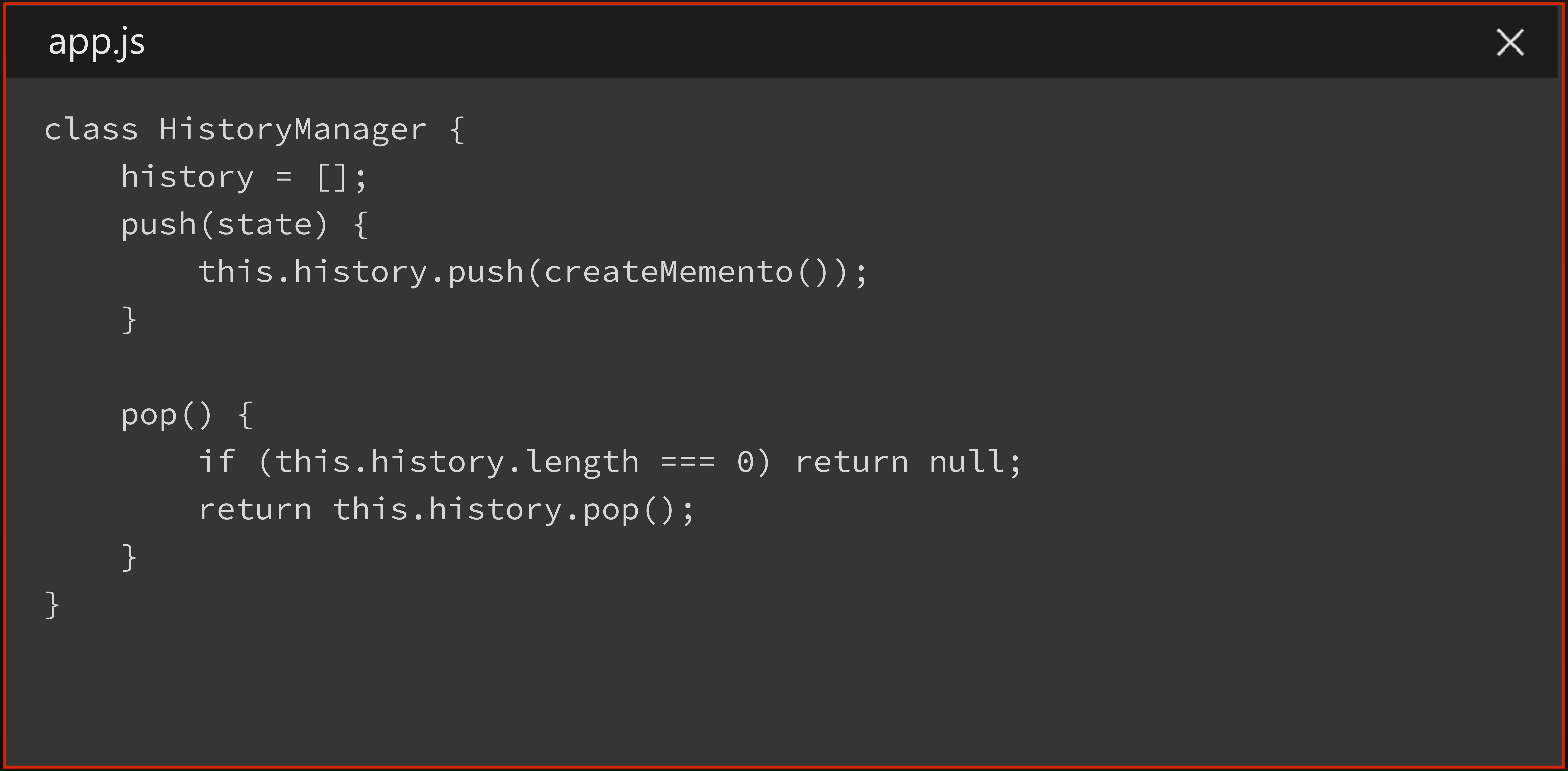
app.js

```
class DataProcessor {
    process() {
        this.loadData();
        this.processData();
        this.saveData();
    }
}

class JSONDataProcessor extends DataProcessor {
    loadData() { /* code */ }
    processData() { /* code */ }
    saveData() { /* code */ }
}
```

Memento

- **Problem to Solve:** Capture and externalize an object's internal state so that it can be restored later, without violating encapsulation.
- **Solution:** Create anobject that stores the state of the original object and provide methods to save and restore the state.
- Use Cases:
  - Undo/Redo functionality
  - Saving a game or app session
  - Time-travel debugging

Memento


app.js

```
class HistoryManager {
    history = [];
    push(state) {
        this.history.push(createMemento());
    }

    pop() {
        if (this.history.length === 0) return null;
        return this.history.pop();
    }
}
```

Command

- **Problem to Solve:** How to avoid hard-wiring a request from its invoker.
- **Solution: create an** object that is used to encapsulate all information needed to perform an action or trigger an event at a later time
- Use Cases:
  - Manage the actions of your app (such as Add, Delete, print, save, load)

LAB

Todo Masters

- Simple Todo app with Vanilla JS
- The code works but it has several problems
- What if we want to:
  - Save the list locally?
  - Add keyboard shortcuts?
  - Make it more complex in the future?
  - Create an undo action?

Let's decouple the project using design patterns with a JavaScript twist.

Lab time

Workshop Repo

firtman.github.io/webapp-patterns

Single Page Application Patterns

Definition

Single Page Application (SPA)

Type of web application that interacts with the user by dynamically rewriting the current web page with new data from the web server, instead of loading entire new pages.

Lazy Load

- **Problem to Solve:** Loading too many JavaScript files when the app loads lead to performance and memory usage problems.
- **Solution:** Use Dynamic Imports from ECMAScript to load modules when needed.
- Use Cases:
  - Load web components when you need them
  - Load routes in SPA when you access them for the first time

View Transitions

- **Problem to Solve:** When changing between routes, there are no transitions as in most apps
- **Solution**: Use the new View Transitions API.
- Use Cases:
  - Animate page change
  - Morph elements between pages

HTML Templates with Interpolation

- **Problem to Solve:** When using templates for Web Components, you can't express in the HTML the bindings you want.
- **Solution:** Use a trick using with ES string templates that will let us interpolate with dynamic data from the HTML.
- Use Cases:
  - Define in the HTML the bindings for the data

Routing Metadata

- **Problem to Solve:** When working with SPA, web page metadata, such as title, SEO data and other information stays static not matter the current URL.
- **Solution:** Update the metadata dynamically when the route changes.
- Use Cases:
  - Adapt the theme-color
  - Change the title
  - Update the favicon based on the current page

LAB

Coffee Masters

- SPA for a Coffee Store
- We will see some patterns implemented: modularization with Web Components
- Check the course **VanillaJS: You Might Not Need a Framework** for more info
- We will implement new patterns:
  - Lazy Load
  - View Transitions for SPA
  - HTML Templates
  - Routing Metadata

Lab time

Multiple Page Application Patterns

Definition

Multiple Page Application (MPA)

Traditional web application architecture where each page of the application is served separately using a new request from the browser to the server.

View Transitions for MPA

- **Problem to Solve:** When changing pages, users can see a white flash between page loads
- **Solution:** Use the View Transitions API for cross-documents.
- Use Cases:
  - Make MPAs feel like SPAs
  - Morph one element from one HTML to another element in the next HTML

Prefetch

- **Problem to Solve:** When the user wants to navigate to a new page, there is a performance penalty
- **Solution:** Use different techniques to prefetch the next possible page, including using the Cache Storage with Service Workers or the Speculation Rules API.
- Use Cases:
    - Prefetch or pre-render the most probable next page on every HTML

HTML Templates for MPA

- **Problem to Solve:** Every new page navigation downloads a whole HTML including the header, footer and navigation again.
- **Solution:** Use service workers to download partial HTML files when you navigate to a new page and marge them with a master page template client-side.
- Use Cases:
  - Improve Performance for MPAs

LAB

Cooking Masters

- MPA for a Recipe website
- We will implement new patterns:
  - * View Transitions for MPA
  - * Prefetch

Lab time

Data and State Management Patterns

Promisify Data

- **Problem to Solve:** Data management tends to change in the future, and when working with static hardcoded data is difficult to move later to an async call.
- **Solution:** Use Promises to deliver all data, including sync data by resolving the Promise statically.
- Use Cases:
  - Hardcoded data
  - Access to sync APIs, such as Local Storage

Promisify Data

app.js

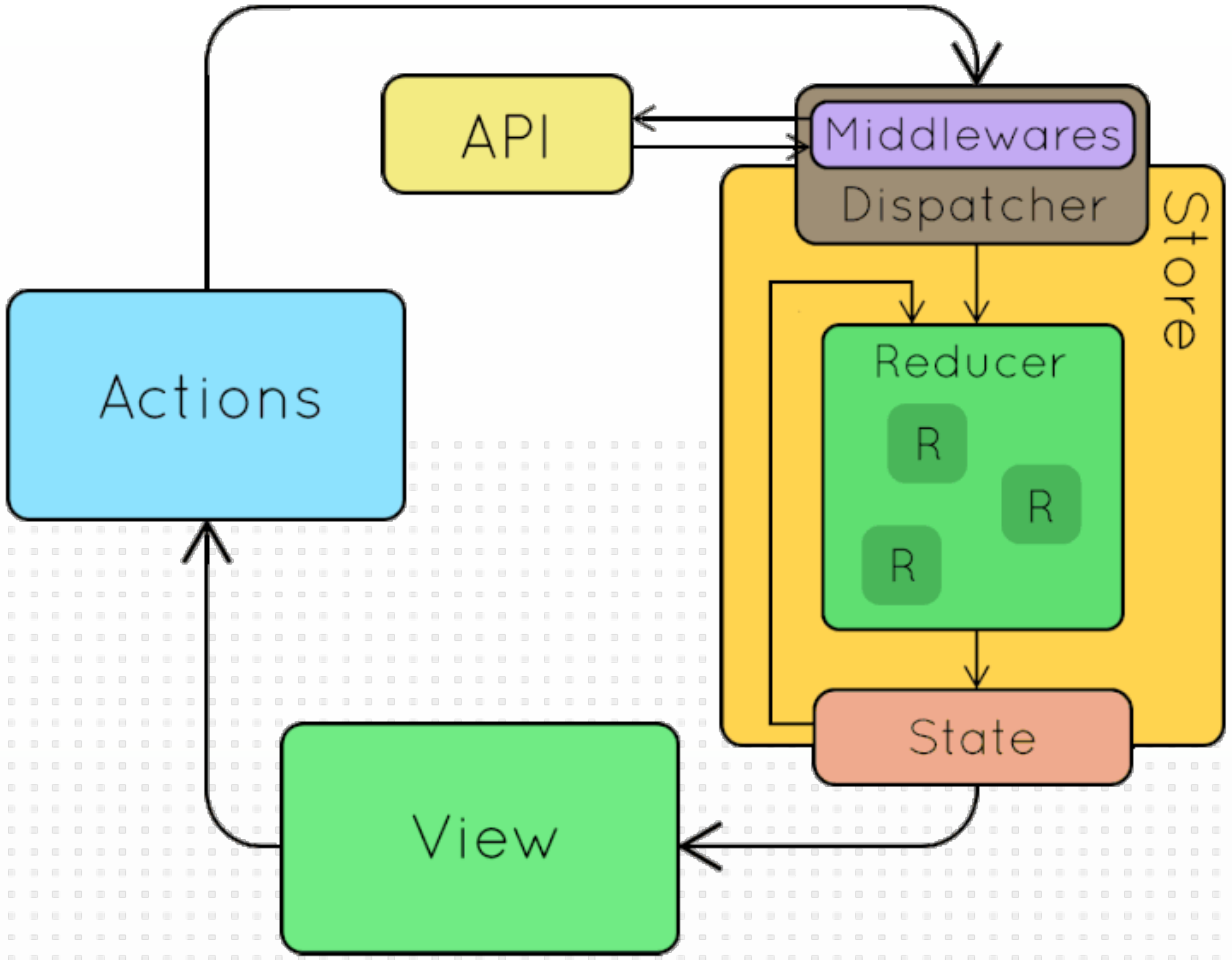
```
// Old version
function getImportantData() {
   return data;
}

// New version
function getImportantData() {
   return Promise.resolve(data);
}
```

Flux

- **Problem to Solve:** In large scale applications, managing the state of the app becomes too complex and unpredictable.
- **Solution:** Use unidirectional data flow, simplifying the architecture and predictability of state changes.
- Use Cases:
  - Data Storage
  - Form intense applications
  - E-commerce
  - CMSs

Redux animation

Based on Flux



Lazy Sync

- **Problem to Solve:** Syncing data to the server takes time and it's not always possible
- **Solution:** Make all the sync to the server asynchronously and detached from the UI.
- Use Cases:
  - Save data and analytics
  - Downloading news
  - Updating app's components in the background

Idea

For more information on this topic, check the course JavaScript in the Background at Frontend Masters

Proxy

- **Problem to Solve**: We don't have always control on the access to an object, including to detect when some value changes.
- **Solution**: Use a Proxy object instead of the object directly.
- Use Cases:
  - Reactive Programming
  - Adding a security layer
  - Logging all access to important objects

Idea

For more information on this topic, check the course Vanilla JS You Might Not Need a Framework at Frontend Masters

Middleware

- **Problem to Solve:** Handling tasks that affect multiple parts of the application, like logging, security checks, error handling, authentication is difficult.
- **Solution**: insert layers of processing between the initial request and the final response, like going through a pipeline.
- Use Cases:
  - API access
  - Database access

More Advanced Ideas and Patterns

Web App Classic Patterns

- Progressive Web Apps
- Responsive Web Design
- Mobile First
- Offline First

Idea

For more information on this topic, check the course Progressive Web Apps at Frontend Masters

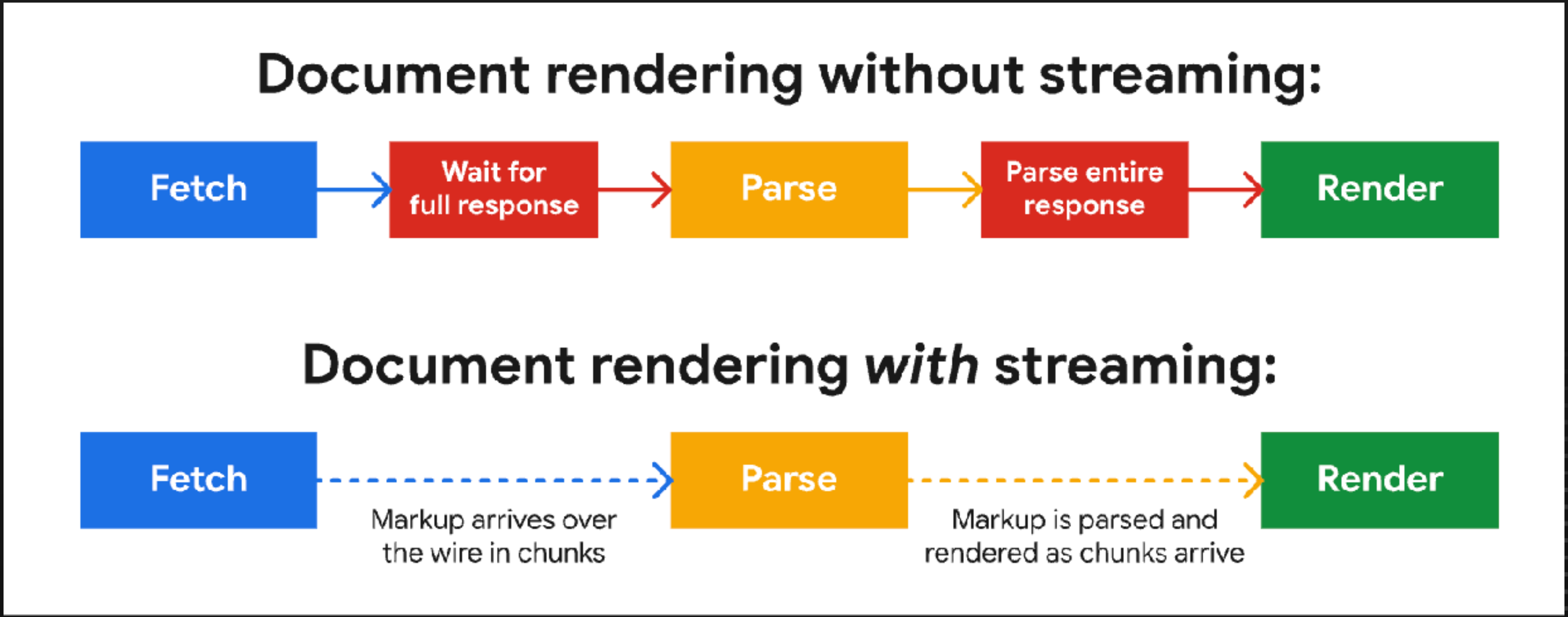
Progressive Enhancement

- **Problem to Solve:** Not every platform supports all the APIs that we want to use.
- **Solution:** Start by offering a solution that works everywhere and add layers on top of that only if the platform supports the API.
- Use Cases:
  - Access hardware and platform APIs
  - Offline support
  - Accessibility

HTML Streaming

- **Problem to Solve:** On large pages, the browser doesn't render the page or data until all the response was sent and downloaded.
- **Solution:** Use Streams and Service Workers to render the HTML response partially in chunks as soon as they are received.
- Use Cases:
  - Improve performance on initial page load
  - Render data ATF initially

Streams with Service Workers


Document rendering without streaming:
Fetch
Wait for full response
Parse
Parse entire response
Render
Document rendering *with* streaming:
Fetch
Parse
Render
Markup arrives over the wire in chunks
Markup is parsed and rendered as chunks arrive

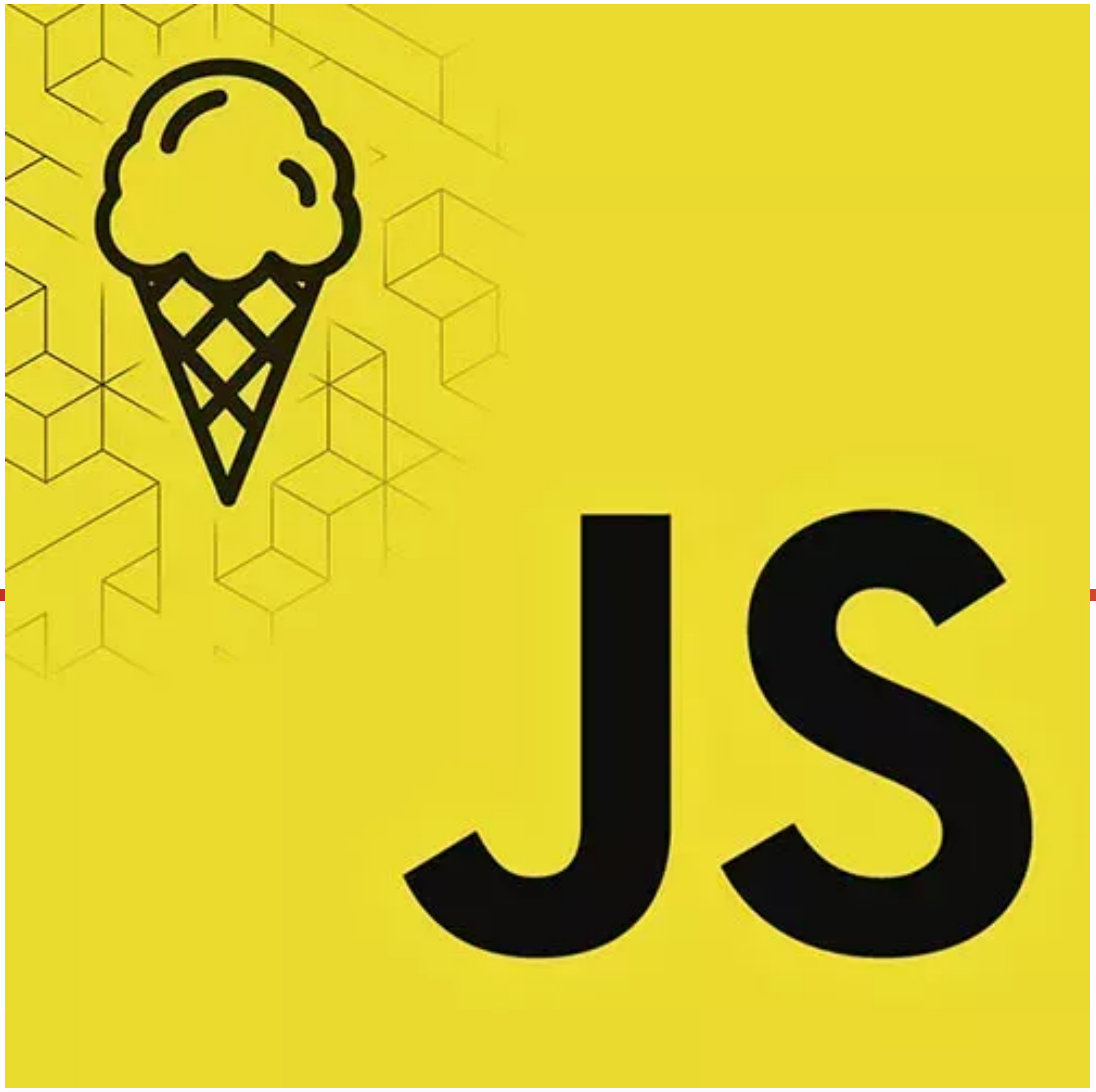
Virtual DOM

- **Problem to Solve:** Working with the DOM directly is expensive
- **Solution:** Create a virtual DOM in memory, work with it and synchronize it with the real DOM once it's a good time for it.
- Use Cases:
  - Complex user interfaces with lot of elements
  - Large lists with re-order and CRUD operations

Recap

Recap

- What's a design pattern
- Classic Design Patterns in JavaScript
- Patterns for SPA
- Patterns for MPA
- Patterns for Data and State Management
- Other ideas



Thanks!

Maximiliano Firtman

firt.dev

x.com/firt

linkedin.com/in/firtman

github.com/firtman