LECTURE 26

Introduction to React.js and Rendering Approaches

From imperative DOM-poking to describing what the page should look like — and letting React figure out the rest.

CSC336 Web Technologies

OVERVIEW

In This Lecture

1. Why React exists: single-page applications, declarative UI, and components
2. The Virtual DOM and how React's reconciliation process updates the real page
3. Setting up a React project with Vite, and the folder structure
4. JSX: embedding expressions, conditional rendering, and rendering lists with key
5. A conceptual overview of rendering approaches: CSR, SSR, SSG, and hydration

Multi-Page Sites vs. Single-Page Apps

1. Multi-page app (MPA): click a link, the browser throws away the page and requests a brand-new one — even the unchanged navbar re-renders
2. Single-page app (SPA): the browser loads one HTML page + a JS bundle; JavaScript swaps content in and out without a full reload
3. Only the data that changed travels over the network, and only the parts of the page that need to change update — navigation feels instant
4. React is the most widely used SPA library, maintained by Meta and a large open-source community

React Is a Library, Not a Framework

NOTE

React only handles the "view" layer — turning data into UI. Routing and global state management are added separately, unlike a full framework like Angular that bundles everything out of the box.

The Problem With Manual DOM Manipulation

1. A shopping cart: every time an item is added or removed you must remember to update the array, find/add-or-remove the <li>, recalculate the total, and update the total's text

2. Every one of these is a manual, imperative instruction: "go find this element, then change it this way"

3. As an app grows, keeping the DOM in sync with data by hand becomes error-prone — easy to forget a step

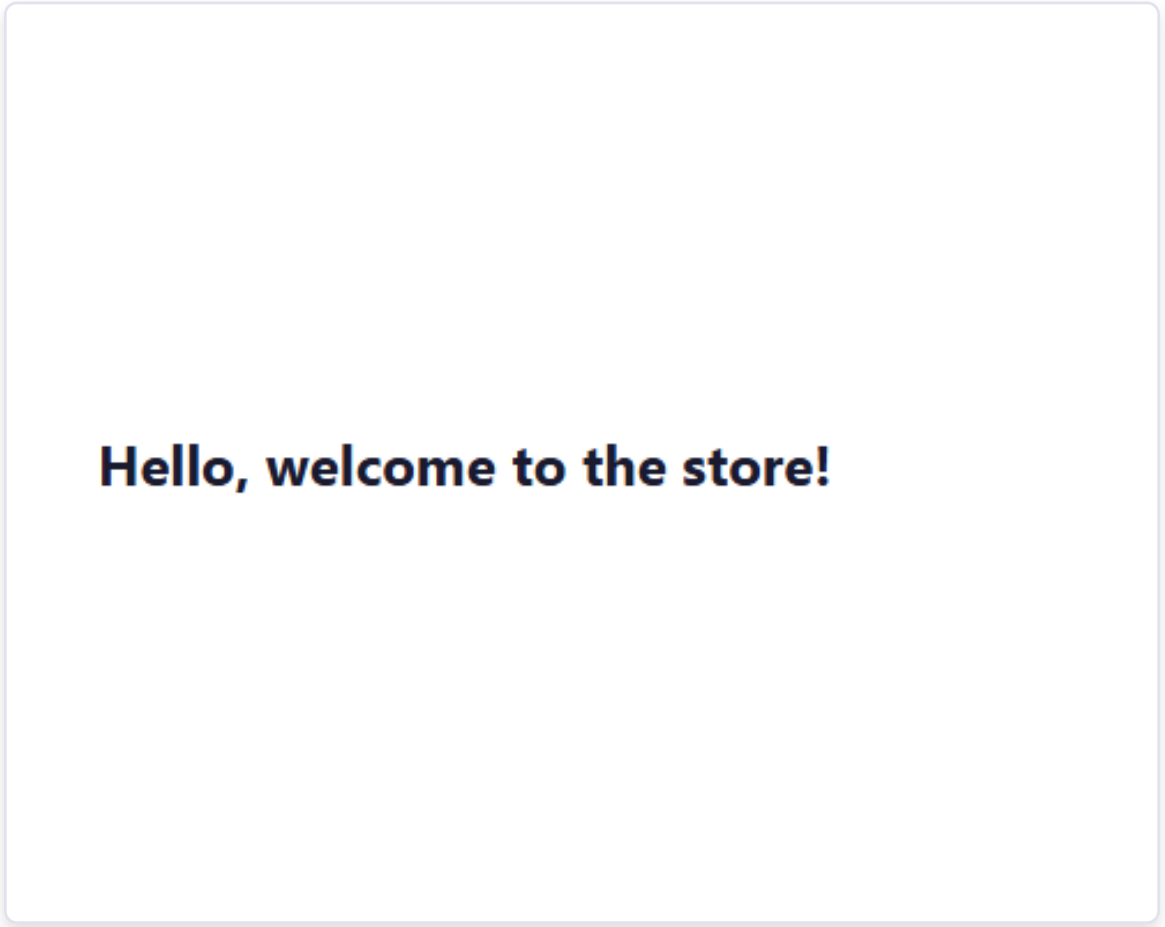
Describe the What, Not the How

```
function CartTotal({ items }) {
  const total = items.reduce(
    (sum, item) => sum + item.price, 0
  );
  return (
    <p>Total: ${total.toFixed(2)}</p>
  );
}
```

Total: $4.50

A Component Is Just a Function

```
function Greeting() {
  return (
    <h1>
      Hello, welcome
      to the store!
    </h1>
  );
}
```



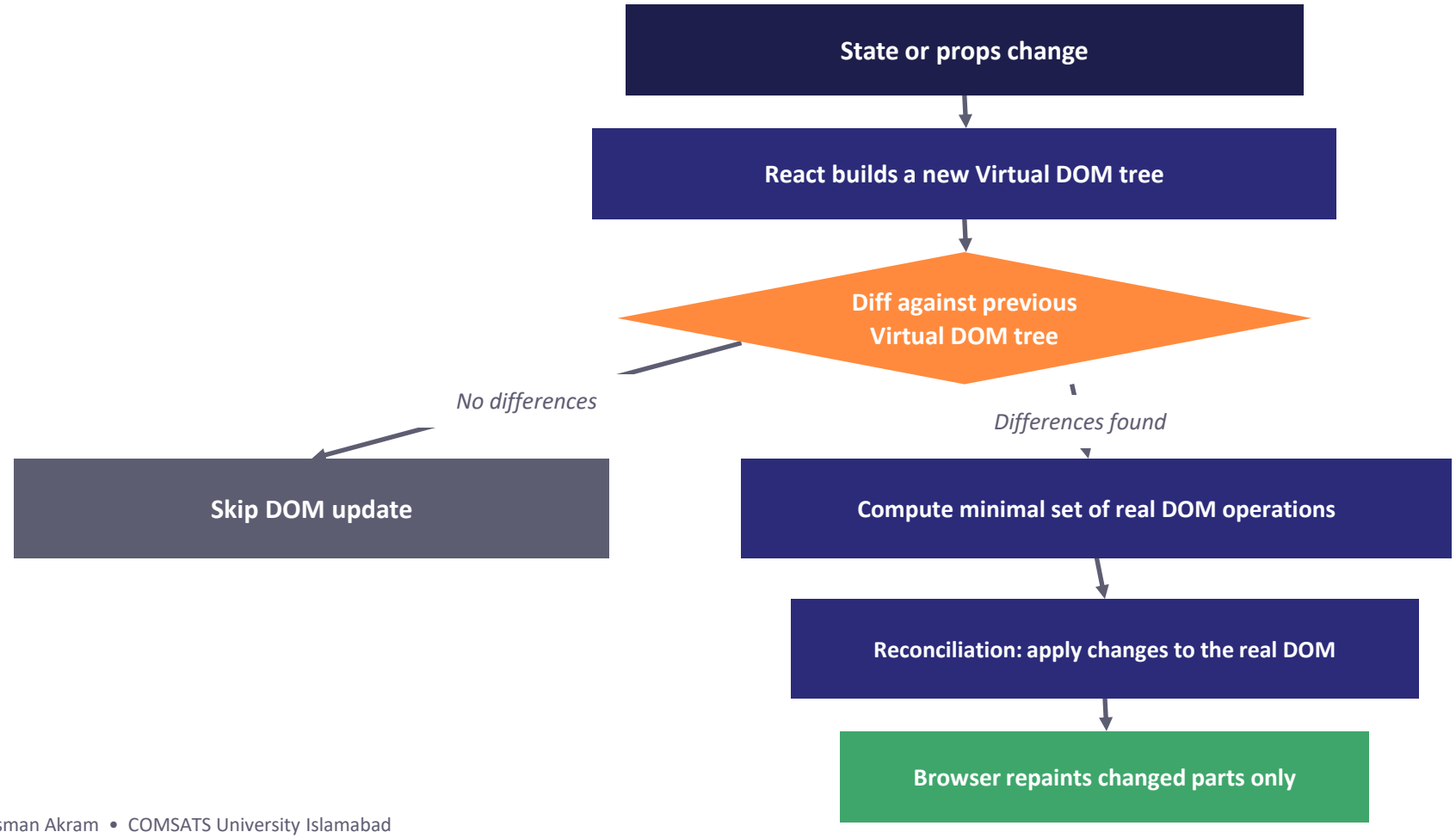
Components Combine Like Building Blocks

1. Components are small, self-contained, reusable pieces of UI, each responsible for one part of the page
2. A Navbar, ProductList, ProductCard, and Footer component can all be assembled inside a single App component
3. Mirrors how you already think about HTML — a page made of sections — but each section is reusable, testable JavaScript
4. Components, props, and composition are studied in depth in Lecture 27

Why React Avoids Touching the Real DOM

1. Direct changes to the real DOM are relatively expensive — layout recalculation, repainting, and more
2. The Virtual DOM (VDOM) is a lightweight, in-memory JavaScript object tree that mirrors your UI's structure
3. When data changes, React builds a new VDOM tree, diffs it against the previous one, and computes the smallest set of real DOM changes needed
4. Comparing plain JS objects in memory is much faster than repeatedly touching the real DOM

Reconciliation, Step by Step



You Rarely Think About This Directly

TIP

As a React developer, you almost never interact with the Virtual DOM yourself. You write components that describe the UI for the current data, and reconciliation handles the rest.

Vite: The Modern Build Tool

1. A build tool takes your JSX and modern JavaScript and bundles it into files browsers can run, plus a local dev server with fast reloading

2. Vite ("veet") is the modern standard — fast because it uses native ES modules during development instead of bundling the whole app on every change

3. Create React App (CRA) was the old official way to start a project — no longer actively maintained, noticeably slower than Vite

Creating a New Project

```
npm create vite@latest my-react-app -- --template react
cd my-react-app
npm install
npm run dev
```

Starts a local dev server (usually http://localhost:5173) with hot module replacement (HMR) — saving a file updates the browser instantly, without a full page reload.

A Freshly Created Project's Folder Structure

```
my-react-app/
├── index.html            # the single HTML page the app is injected into
├── package.json           # project metadata and dependencies
├── vite.config.js         # Vite's configuration file
├── public/                 # static assets copied as-is
└── src/
    ├── main.jsx            # entry point: mounts <App /> into index.html
    ├── App.jsx              # the root component
    ├── App.css
    └── index.css
```

main.jsx: Mounting the App

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import App from "./App.jsx";

createRoot(document.getElementById("root")).render(
  <StrictMode>
    <App />
  </StrictMode>
);
```

index.html contains one empty container, . Everything your users see is ultimately rendered inside that single div — the essence of a single-page application.

What Is JSX?

1. JSX (JavaScript XML): a syntax extension letting you write HTML-like markup directly inside JavaScript
2. Browsers cannot run JSX directly — Vite transforms it into React.createElement(...) calls before the code reaches the browser
3. You write JSX because it reads much closer to the HTML you already know, while still being plain JavaScript underneath

What JSX Compiles To

```
const element = <h1>Hello, world!</h1>;

// Vite/Babel transforms the line above into approximately:
const element2 = React.createElement("h1", null, "Hello, world!");
```

Embedding Expressions With { }

```
function UserGreeting({ name }) {
  const hour = new Date().getHours();
  const timeOfDay = hour < 12
    ? "morning" : "afternoon";
  return (
    <p>
      Good {timeOfDay},
      {name.toUpperCase()}!
    </p>
  );
}
```

Good afternoon, AYESHA!

JSX Rules to Remember

WARNING

A component must return a single root element (or a Fragment). Use className instead of class. Every tag must be closed, including self-closing tags like <img />. Curly braces { } can only hold expressions, not statements like if or for.

Expressions That Evaluate to JSX

```
function LoginStatus({ isLoggedIn }) {
  if (isLoggedIn) return <p>Welcome back!</p>;
  return <p>Please log in.</p>;
}

function Notification({ count }) {
  return (
    <div>
      {count > 0 && <span className="badge">{count} new</span>}
      {count === 0 ? <span>No notifications</span> : null}
    </div>
  );
}
```

count > 0 && ... renders the span only when truthy, otherwise nothing. The ternary chooses between two different pieces of UI.

Notification, Two States

COUNT = 3

COUNT = 0

No notifications

.map() and the key Prop

```
function ProductList({ products }) {
  return (
    <ul>
      {products.map((p) => (
        <li key={p.id}>
          {p.name} — ${p.price}
        </li>
      ))}
    </ul>
  );
}
```

- Notebook — $3.5
- Pen — $1
- Highlighter — $2.25

Don't Use the Array Index as a Key

WARNING

It's tempting to write key={index} — fine for static lists that never reorder. But if the list CAN change, using the index can mix up which DOM element belongs to which data (subtle bugs like inputs showing the wrong value). Prefer a stable, unique identifier from your data, such as a database id.

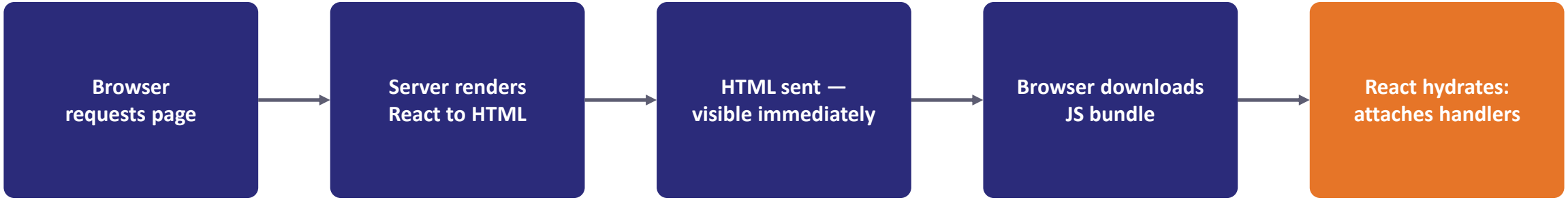
CSR, SSR, SSG, and Hydration

Approach	When HTML Is Generated	Typical Use Case
CSR	In the browser, after JS downloads and runs	Dashboards, apps behind a login
SSR	On the server, for each request	Fast first load, SEO-friendly content
SSG	At build time, before deployment	Blogs, documentation, marketing pages
Hydration	A step after SSR/SSG delivers HTML	Making pre-rendered HTML interactive

CSR, SSR, and SSG in Plain Terms

1. CSR: what a plain Vite + React app does by default — first paint can be slow, but navigation is fast once loaded
2. SSR: runs React components on the server (Node.js) per request, producing HTML the browser can show immediately
3. SSG: produces full HTML once, at build time — served instantly from a CDN, no per-visitor server computation

SSR + Hydration, in Sequence



Hydration connects SSR/SSG HTML to React on the client: the HTML appears instantly, but has no event listeners until React "wakes it up" by attaching them, without re-creating any DOM nodes.

This Course Focuses on CSR

NOTE

In this course you will build a client-side-rendered SPA with Vite, talking to the Express REST API from Lecture 25. SSR, SSG, and frameworks like Next.js are covered in depth in the Advanced Web Technologies course.

KEY TAKEAWAYS

Lecture 26 in Six Points

1. React builds single-page applications using a declarative style: describe what the UI should look like, instead of manually updating the DOM.
2. UIs are built from components — reusable functions that return JSX.
3. The Virtual DOM is an in-memory copy of the UI tree; React diffs it against the previous version and applies only minimal changes — reconciliation.
4. Vite is the modern tool for creating and running React projects, replacing the older Create React App.
5. JSX lets you write HTML-like syntax in JavaScript; use { } for expressions, className instead of class, and a stable, unique key for list items.
6. CSR, SSR, and SSG are different strategies for turning components into HTML; hydration makes server-delivered HTML interactive in the browser.