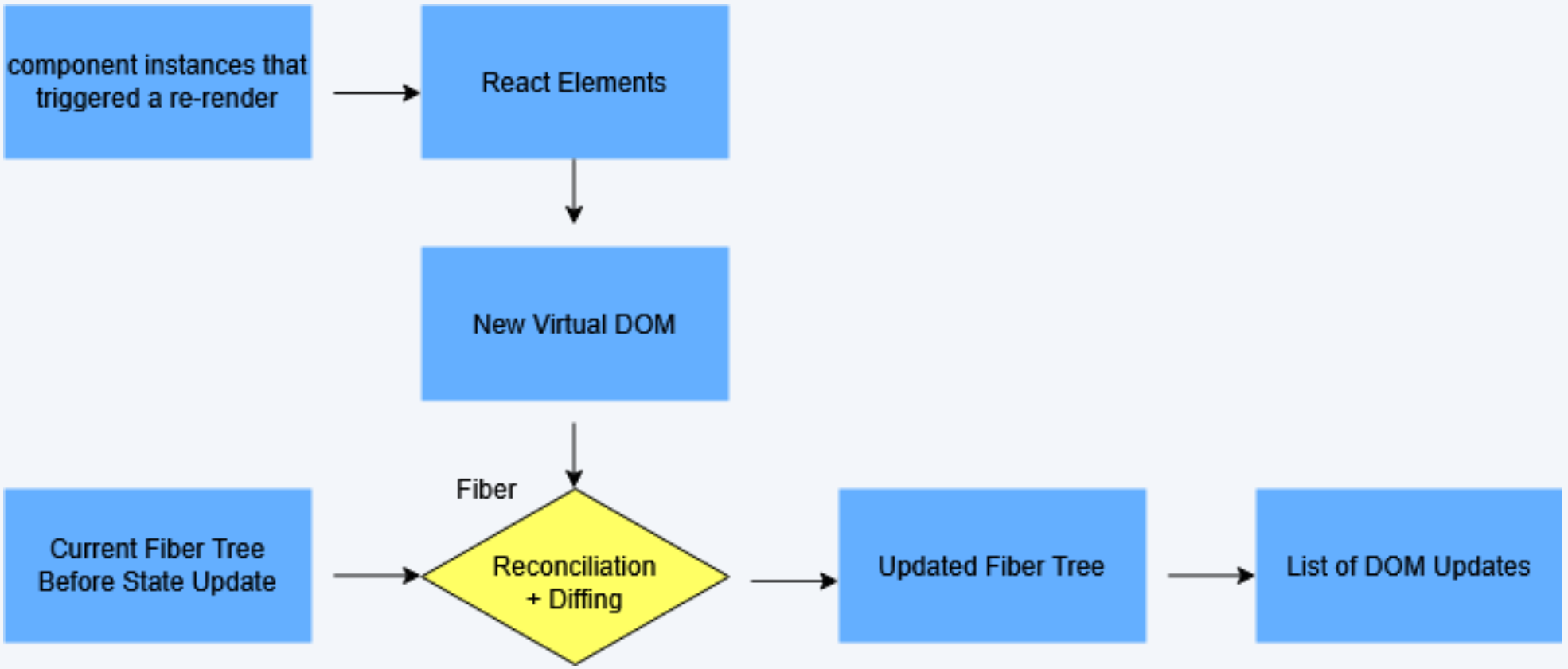
ReactJs

Behind the scences

Render Phase
Virtual DOM
Reconciliation Fiber

Overview of React Rendering Process



Overview of React Rendering Process

React Rendering Overview:

- The process is split into three phases:
    a. Render Phase (internal to React).
    b. Commit Phase (updates the DOM).
    c. Browser Repaint (visual changes on the screen).

High level overview of react application.

- Applications are built using components.
- Components are reused, creating component instances that hold state and props.
- Each component instance produces React elements via React.createElement
- React elements are transformed into DOM elements and displayed as the user interface.

Render Phase Details:

- Triggered by state changes or the initial render.
- React calls component functions to determine how the DOM should be updated.
- Important: The render phase **does not update the DOM** or produce visual changes.
- React's definition of "render" is internal , happens inside the react and different from traditional rendering (displaying elements on the screen).

Misconception
Rendering was used to mean displaying elements on the screen, but technically, it refers to the internal process of calling component functions and determining DOM updates.

Overview of React Rendering Process

Commit Phase Details:

- After the render phase, React updates the DOM in the commit phase.
- New elements are added, existing elements are updated or deleted to reflect the current state.
- This phase is responsible for what is traditionally called "rendering" (visual changes).

Browser Repaint:

- After the DOM is updated, the browser repaints the screen, producing the visual changes users see.
- This step is outside React's control.

Triggering a Render

1. When Renders Are Triggered:
   - Initial Render: When the application runs for the first time.
   - Re-render: Triggered by state updates in one or more component instances.
2. Render Scope:
   - A render is triggered for the entire application, not just the component where the state update occurred.
   - React evaluates the entire component tree but only updates the necessary parts of the DOM.
3. Render Scheduling:
   - Renders are not triggered immediately after state updates.
   - They are scheduled when the JavaScript engine has free time (usually within milliseconds).
   - Multiple state updates in the same function may be batched into a single render.(state update batching more on this in future slides)

Render Phase

- understanding render phase is most important if you need to be in top 1% of react developers.
- Earlier, it was suggested that re-renders replace the entire component view, but this is technically incorrect.
- React does not update the entire HTML DOM for a component instance during re-renders.

1. What Happens in the Render Phase:
    - React goes through the component tree and calls the component functions for instances that triggered a re-render.
    - This process creates updated React elements, which form the virtual DOM.
2. Virtual DOM:
    - The virtual DOM is a tree of React elements created from all component instances.
    - It is a lightweight JavaScript object, making it cheap and fast to create.
    - The term "virtual DOM" is widely used but downplayed in React's official documentation.
3. Re-rendering and Child Components:
    - When a component re-renders, all its child components re-render as well, regardless of whether their props changed.
    - This ensures React captures all potential changes but can lead to unnecessary re-renders in large applications.

Render Phase

Misconceptions About React's Rendering Phase.

Misconception 1: Rendering means updating the screen or the DOM.

- Reality: Rendering is not about the screen or the DOM. It's simply about calling component functions to determine what the UI should look like based on the current state and props.

Misconception 2: During a re-render, React discards the old component view and replaces it with a brand new one.

- Reality: React does not discard the entire component view or update the entire DOM. Instead, it efficiently produce updates for only the parts of the DOM that have changed.

How Renders Are Performed in the Render Phase

- The render phase is where React determines what changes need to be made to the UI based on state and props.
- It involves calling component functions and creating a virtual DOM

Steps in the Render Phase

1. Traversing the Component Tree:
    - React goes through the entire component tree and identifies all components that have triggered a re-render (due to state or prop changes).
2. Calling Component Functions:
    - React calls the corresponding component functions (the code you've written) to generate updated React elements.
3. Creating the Virtual DOM:
    - The updated React elements are combined to form a virtual DOM, which is a lightweight JavaScript object representation of the UI.

Deep Dive into the Virtual DOM (React Element Tree)

What is the Virtual DOM?

- Definition: The virtual DOM is a tree of React elements created from all instances in the component tree.
- How It's Created:
  - On the initial render, React takes the entire component tree and transforms it into a React element tree (virtual DOM).
  - Each component function generates React elements, which together form the virtual DOM.
- Characteristics:
  - It's a lightweight JavaScript object, making it cheap and fast to create and update.
  - It's recreated on every render but is much faster to work with compared to the real DOM.

Why Use the Virtual DOM?

- Efficiency: Updating the real DOM is slow and expensive. The virtual DOM allows React to minimize direct DOM manipulation.
- Optimization: React uses the virtual DOM to calculate the minimal set of changes needed and then applies them to the real DOM in the commit phase.
- Misconception: The term "virtual DOM" is often overhyped. React's official documentation no longer uses this term, but it's still widely used in the community.

Deep Dive into the Virtual DOM (React Element Tree)

Re-rendering in React

- What Happens During a Re-render:
  - When a component's state or props change, React calls its function again and generates a new React element.
  - This new React element is placed in a new virtual DOM.
- Child Components Always Re-render:
  - Whenever a component re-renders, all its child components re-render as well, regardless of whether their props have changed.
  - This happens because React doesn't know beforehand whether an update in a parent component will affect its children.
- Example:
  - If the highest component in the tree (e.g., Component A) re-renders, the entire application will re-render.
  - However, this does not mean the entire DOM is updated. Only the necessary changes are applied.

Why React Re-renders Everything by Default

- Safety First: React prefers to play it safe and re-render all child components to ensure no updates are missed.
- Performance Impact:
  - Recreating the virtual DOM is not a big problem in small or medium-sized applications because it's just a JavaScript object.
  - However, in large applications, unnecessary re-renders can impact performance, which is why tools like React.memo, useMemo, and useCallback are used to optimize re-renders.

What is Reconciliation and Why do we need it ?

High Level overview of Reconciliation

- Reconciliation is basically deciding exactly which DOM elements need to be inserted, deleted or updated in order to reflect the latest state changes.
- New virtual DOM that was created after the state update will get reconciled with the current so-called Fiber tree as it exists before the state update.
- Now this reconciliation is done in React's reconciler which is called Fiber.
- Results of this reconciliation process is gonna be an updated Fiber tree, so a tree that will eventually be used to write to the DOM.

Why Not Update the Entire DOM?

- Writing to the actual DOM is expensive and slow.
- On state changes, usually only a small portion of the DOM needs to be updated.
- Reusing the existing DOM is more efficient than recreating it from scratch.
- Example: In a complex app like Udemy.com, clicking a button to show a modal only updates the modal's DOM elements, leaving the rest unchanged.

What is Reconciliation?

- Definition: The process of determining what changed between the current and new Virtual DOM.
- Goal: Decide which DOM elements need to be inserted, deleted, or updated to reflect the latest state changes.
- Output: A list of DOM operations necessary to update the current DOM.

What is Reconciliation and Why do we need it ?

Reconciler (Fiber Reconciler)

- The engine of React, responsible for reconciliation.
- Determines what changes are needed between the old and new Virtual DOM.
- Allows developers to describe the UI based on state without directly manipulating the DOM.
- The current reconciler in React is called Fiber .The Fiber reconciler enables concurrency and prioritization of updates.

How React Ensures Efficiency

- React minimizes DOM updates by comparing the Virtual DOM with the Fiber tree.
- Uses a diffing algorithm to identify changes.
- Prioritizes high-priority updates (e.g., user interactions) over low-priority ones.
- Outputs an updated Fiber tree, which is used to write changes to the DOM.

Reconciler Fiber

- During the initial render of the application Fiber takes the entire React element tree and based on it builds yet another tree which is the Fiber tree.
- The Fiber tree is a special internal tree where for each component instance and DOM element in the app, there is one so-called Fiber.
- special about this tree is that unlike React elements in the virtual DOM, Fibers are not recreated on every render.
- Unlike React elements in the Virtual DOM, Fibers are not recreated on every render.The Fiber tree is a mutable data structure that persists across renders.

What Makes Fibers Special?

- Fibers are not destroyed after each render; they are mutated during future reconciliation steps.
- Fibers store Current component state and props, Side effects (e.g., lifecycle methods, hooks). A list of used hooks.
- The actual state and props of any component instance are stored inside the corresponding Fiber.

Fiber as a Unit of Work

- Each Fiber contains a queue of work to do, such as Updating state, Updating refs, Running registered side effects, Performing DOM updates.
- This is why a Fiber is defined as a unit of work

Reconciler Fiber

Fiber Tree Structure

- The Fiber tree is arranged differently than the React element tree (Virtual DOM).
- Instead of a normal parent-child relationship, fibers are arranged in form of structure linked list.
- The linked list structure makes it easier for React to process the work associated with each Fiber.It allows React to efficiently traverse and update the tree during reconciliation.
- The Fiber tree includes not only React components but also regular DOM elements (e.g., h3, button).
- Both the React element tree and the Fiber tree are complete representation of the entire DOM structure.

Asynchronous Rendering in Fiber

- Fibers are units of work, and the Fiber reconciler can perform work asynchronously.
- The rendering process can be Split into chunks, Prioritized (some tasks are more important than other, Paused, resumed, or discarded if no longer valid.
- Concurrent Features: Enables modern React features like Suspense and transitions (introduced in React 18)
- Non-Blocking: Prevents long renders from blocking the browser's JavaScript engine.
- Improved Performance: Especially beneficial in large applications where long renders can cause performance issues.
- Asynchronous rendering happens automatically and is invisible to developers.
- The render phase does not produce any visible output to the DOM, allowing React to pause and resume work without affecting the UI.