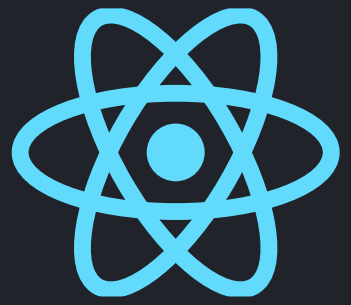


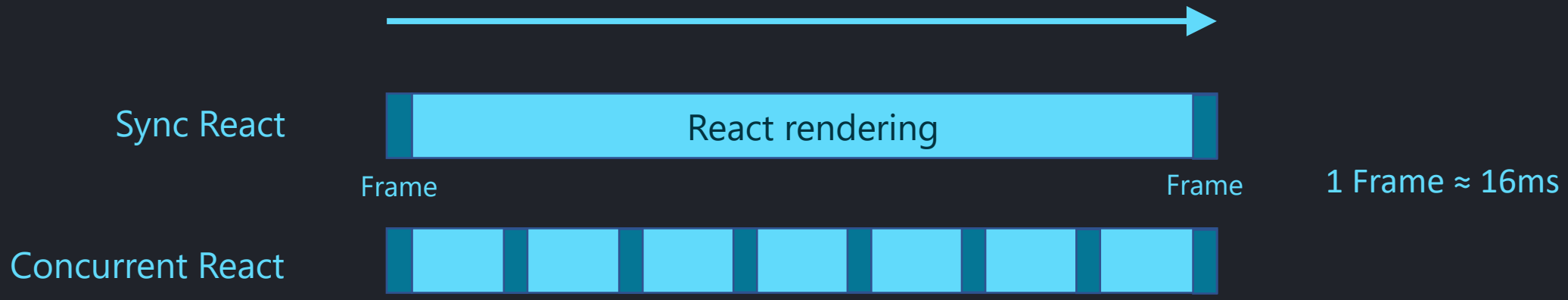
React 18

Demo | Github | LinkedIn

Isha Pachkawade

Concurrent React

- Before React 18, rendering was a single, uninterrupted, synchronous transaction. Once rendering started, it couldn't be interrupted.
- With concurrent rendering, React can interrupt, pause, resume, or abandon a render. This allows React to respond to the user interaction quickly even if it is in the middle of a heavy rendering task.
- This unlocks new possibilities to improve both real and perceived performance of apps.



Migration

- `ReactDOM.render` has been deprecated but not removed. Concurrent features will be enabled only when `ReactDOM.createRoot` is used.

```
/* Until react 17 */
import ReactDOM from 'react-dom';
import { App } from './App';

ReactDOM.render(<App />, document.getElementById('root'));
```

```
/* React 18 onwards */
import ReactDOM from 'react-dom/client';
import { App } from './App';

const rootElement = document.getElementById('root');
const root = ReactDOM.createRoot(rootElement!);
root.render(<App />);
```

Automatic Batching

- Grouping/Batching multiple state updates into a single re-render produces better performance.
- Updates are said to be batched in a function if the Component re-renders only once at the end of the function and applies all state updates at once instead of re-rendering after every statement which updates state.
- Until React 17, updates only inside React event handlers were batched.
- React 18 onwards updates inside of `promises`, `setTimeout`, `native event handlers` or any other event are also batched by default.
- Opting out of automatic batching can be done using `flushSync`.
- Demo: https://isha-11.github.io/react-18-features/#/batching

Transition

- Transition is a new concept in React to distinguish between urgent and non-urgent updates.
  - **Urgent updates** – direct user interactions like typing, clicking, pressing and so on.
  - **Transition (non-urgent) updates** – transitions of the UI from one view to another.

- By default all updates are urgent. Updates marked as transition are handled as non-urgent and will be interrupted if more urgent updates like clicks or key presses come in.

- React provides two ways to mark updates as transition (non-urgent):
  - `useTransition` - a hook to start transitions, including a value to track the pending state.
  - `useDeferredValue` - a hook which accepts a value and returns a new copy of the value that will defer till urgent updates are completed.

- Demo:
  - https://isha-11.github.io/react-18-features/#/transition
  - https://isha-11.github.io/react-18-features/#/deferred-value

Transition: useTransition

```
const App = () => {
  const [urgentQuery, setUrgentQuery] = useState('');
  const [nonUrgentQuery, setNonUrgentQuery] = useState('');
  const [isPending, startTransition] = useTransition();

  const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
    setUrgentQuery(e.target.value);
    startTransition(() => setNonUrgentQuery(e.target.value));
  };

  return (
    <div>
      <input type='text' value={urgentQuery} onChange={handleChange} />
      {isPending ? <div>Loading...</div> : <Result query={nonUrgentQuery} />}
    </div>
  );
};
```

Transition: useDeferredValue

```
const App = () => {
  const [query, setQuery] = useState('');
  const deferredQuery = useDeferredValue(query);
  const isPending = query !== deferredQuery;

  const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
    setQuery(e.target.value);
  };

  return (
    <div>
      <input type='text' value={query} onChange={handleChange} />
      {isPending ? <div>Loading...</div> : <Result query={deferredQuery} />}
    </div>
  );
};
```

Suspense Recap

- Suspense adds the ability to have a component notify React at render time that it's waiting for asynchronous data - this is called suspending.
- The component throws a promise which is caught by `<Suspense/>`, awaiting its resolution.
- While waiting, React will decline to render the pending state and `<Suspense/>` renders a fallback component.
- When the promise is settled, the component is rendered again and fallback is removed.

Suspense + Transition

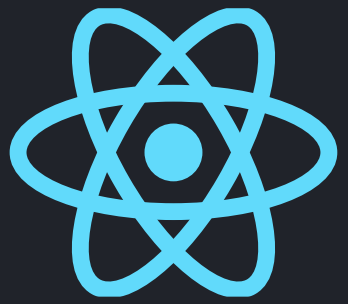
- Until React 17, every time the component suspended, Suspense used to show a fallback.
- However, from the user's perspective, this can be disorienting if there are frequent switches between fallback and content.
- In particular, it is sometimes better to show the “old” UI while the “new” UI is being prepared.
- From React 18, if component suspends during a transition (non-urgent update), React will prevent already-visible content from being replaced by a fallback.
- React will keep the old UI in place and interactive, and will switch to showing new content when it is ready i.e. when the transition update is completed.
- Demo: https://isha-11.github.io/react-18-features/#/suspense

Strict Mode

- In future, React may add a feature that allows to add and remove sections of the UI while preserving state.
- To do this, React will support remounting trees using the same component state used before unmounting.
- This feature requires components to be resilient to effects being mounted and destroyed multiple times.
- Most effects will work without any changes, but if some effects do not properly clean up subscriptions in the destroy callback, or implicitly assume they are only mounted or destroyed once – those would cause errors.
- This new check will automatically unmount and remount every component, whenever a component mounts for the first time, restoring the previous state on the second mount.

Summary

- We can adopt React 18 features gradually.
- Concurrent features are opt-in. Will only be enabled when `ReactDOM.render` is changed to `ReactDOM.createRoot`.
- State updates will be batched by default.
- State updates will remain urgent unless marked as transition.
- StrictMode behaviour has been modified wherein on the first component mount, React will simulate mounting-unmounting-remounting of effects. This is to prepare for a future release.



Presentation Unmounted

Demo | Github | LinkedIn