

Building User Interfaces

React 4

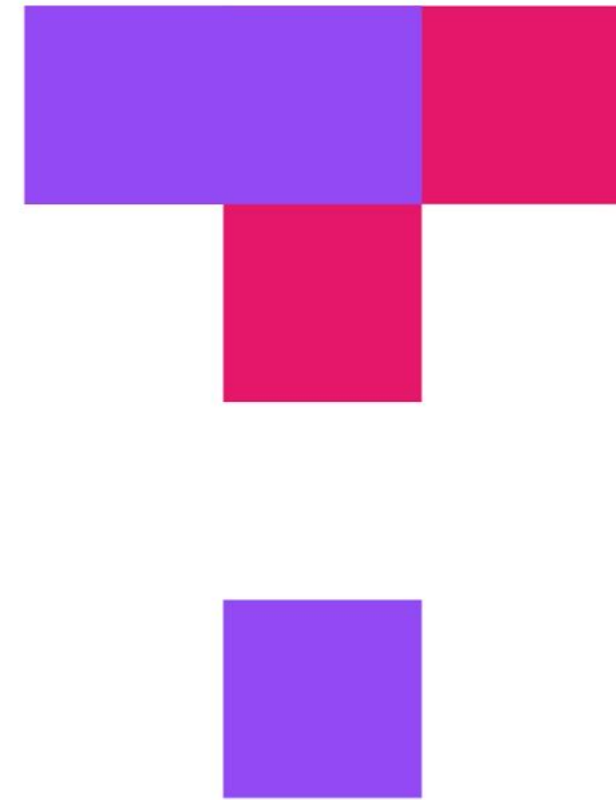
Advanced Concepts

Professor Bilge Mutlu

What we will learn today?

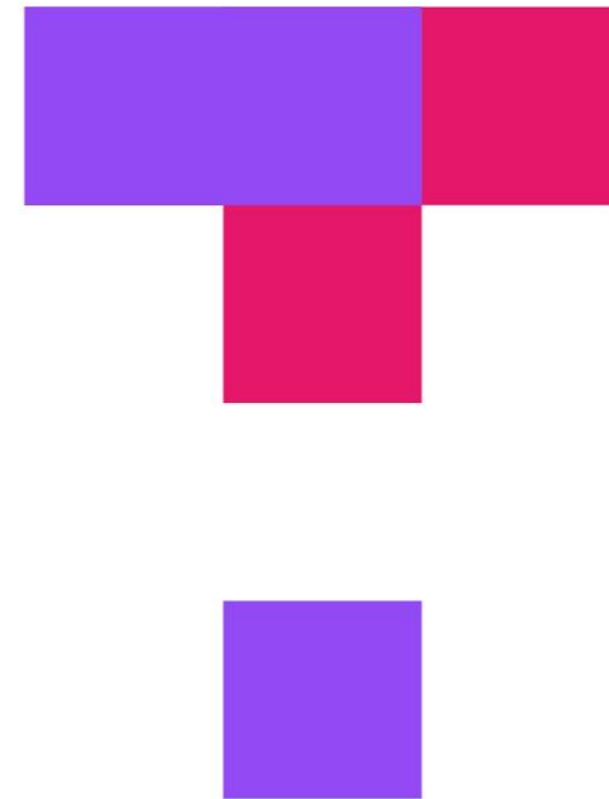
- >> Optimizing performance in React
- >> Advanced asynchronous updating
- >> APIs for advanced interaction

TopHat Attendance



TOP HAT

TopHat Questions



TOP HAT

Optimizing *Performance* in React

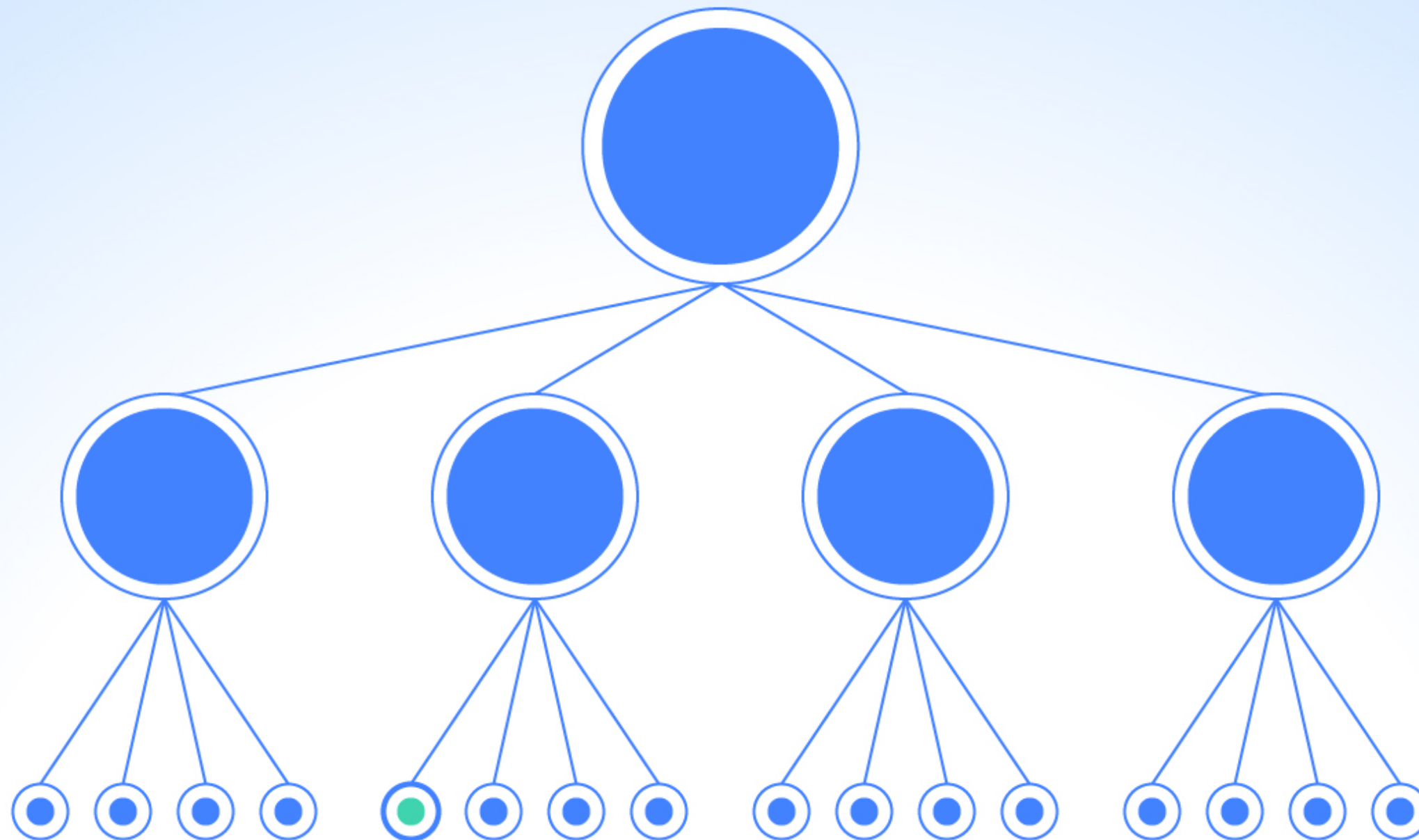
Why do we need to worry about performance?¹

As the complexity of your application scales, performance will necessarily degrade.

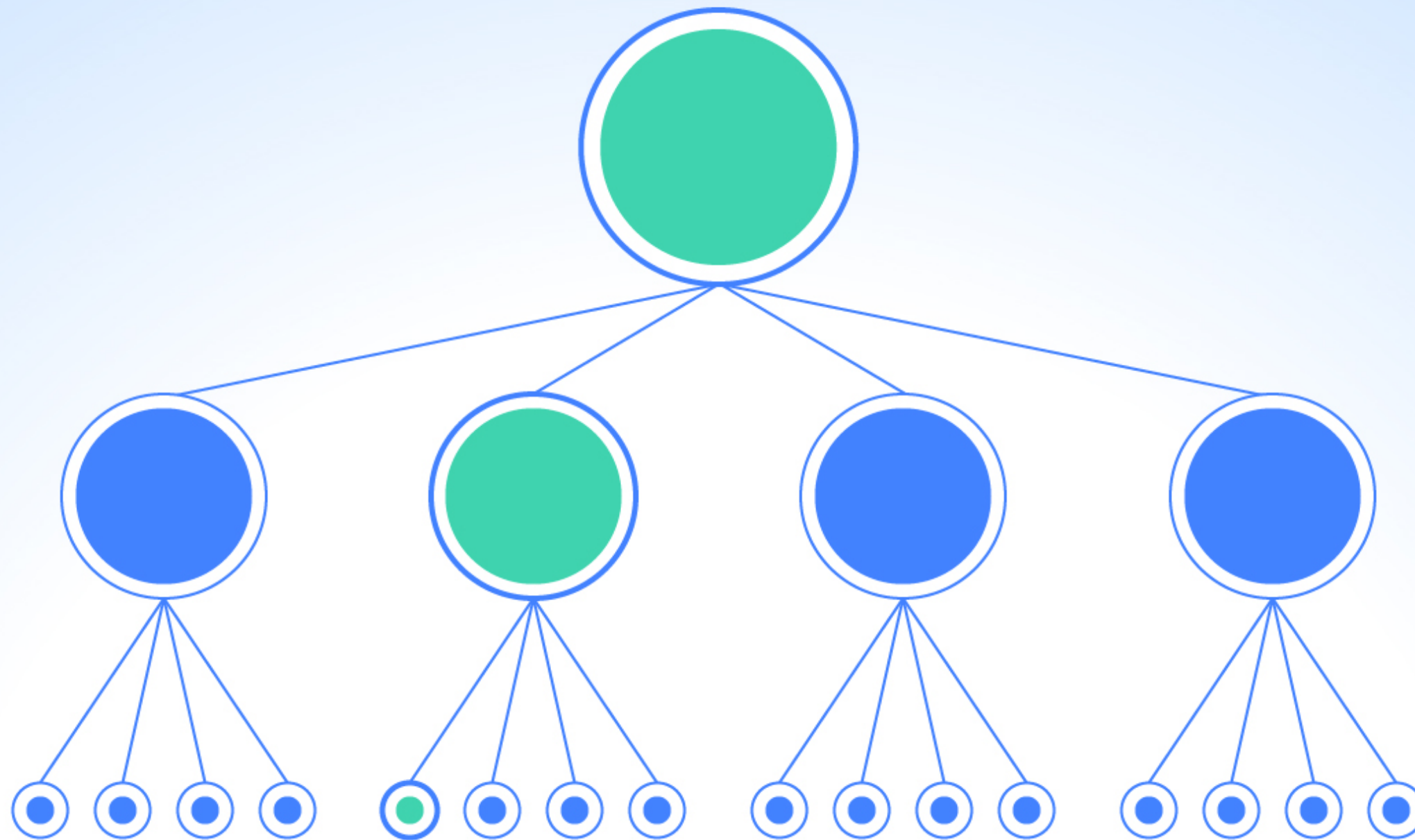
Why? And what do we do about it?



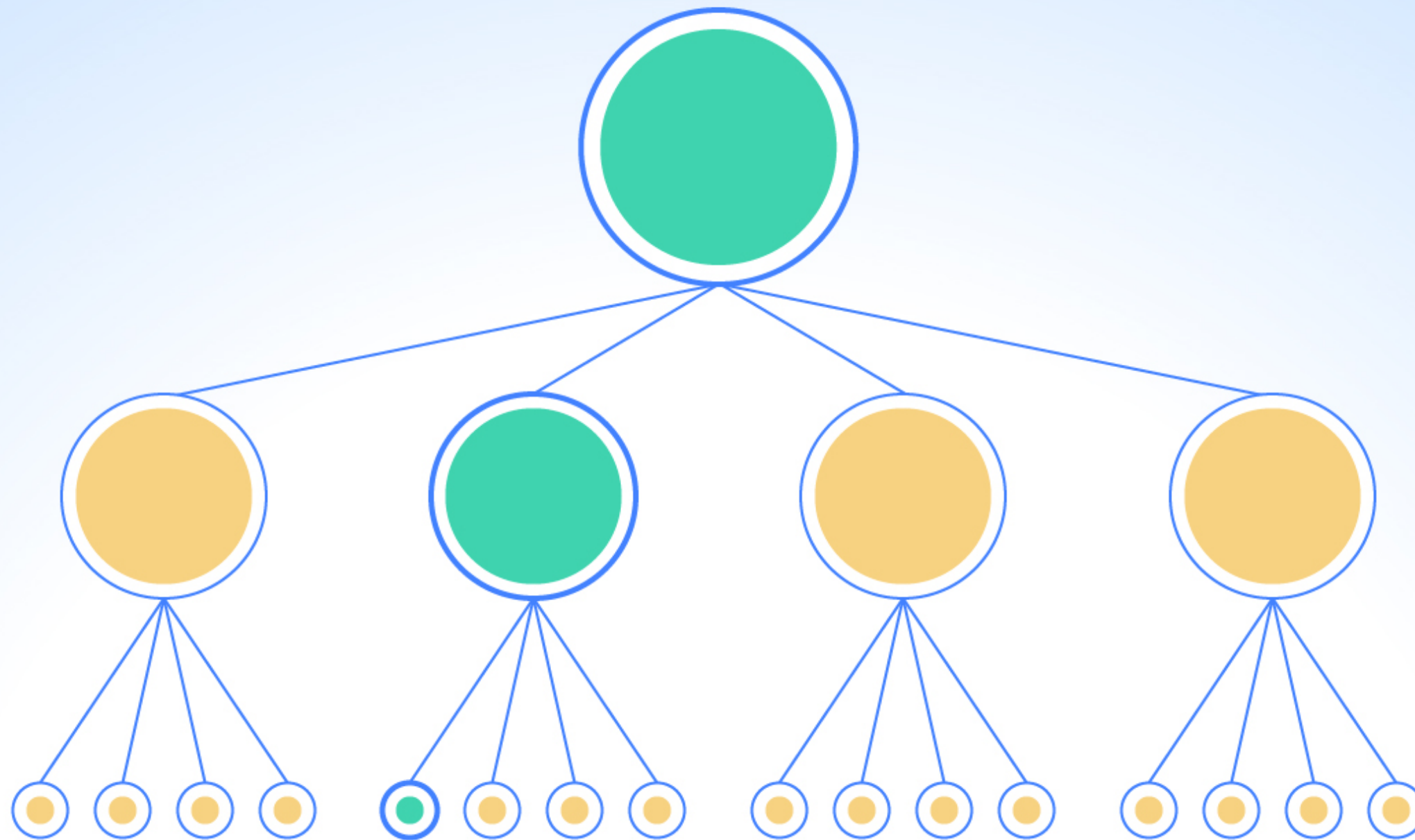
¹Image Source: [Noam Elboim](#)



² Image Source: [William Wang](#)



² Image Source: [William Wang](#)



² Image Source: [William Wang](#)

Why does React do that?

That's how React works!

We discussed in React 1 that the diffing within Virtual DOM—*reconciliation*—is what makes it fast, but when things are scaled up, continuous diffing and updating affects performance.

How do we know that?

Performance tools: React provides a powerful library, `react-addons-perf`,³ for taking performance measurements.

```
import Perf from 'react-addons-perf';
```

```
Perf.start()
```

```
// Our app
```

```
Perf.stop()
```

³ReactJS.org: [Performance tools](#)

Useful Perf methods

- >> `Perf.printInclusive()` prints overall time taken.
- >> `Perf.printExclusive()` prints time minus mounting.
- >> `Perf.printWasted()` prints time *wasted* on components that didn't actually render anything.
- >> `Perf.printOperations()` prints all DOM manipulations.
- >> `Perf.getLastMeasurements()` prints the measurement from the last Perf session.

Perf.printInclusive() and Perf.printWasted() output:⁴

ReactPerf.js:32

| (index) | Owner > Component | Inclusive render time (ms) | Instance count | Render count |
|---------|------------------------------|----------------------------|----------------|--------------|
| 0 | "App > RecipesContainer" | 21.49 | 1 | 1 |
| 1 | "RecipesContainer > Route" | 20.58 | 2 | 2 |
| 2 | "Route > recipeList" | 20.51 | 1 | 1 |
| 3 | "recipeList > recipeShow" | 12.42 | 1 | 1 |
| 4 | "recipeShow > AddToPlanner" | 6.31 | 1 | 1 |
| 5 | "AddToPlanner > t" | 4.86 | 1 | 1 |
| 6 | "t > t" | 0.59 | 1 | 1 |
| 7 | "recipeList > Link" | 0.42 | 6 | 6 |
| 8 | "RecipesContainer > Planner" | 0.27 | 1 | 1 |
| 9 | "recipeList > recipeSearch" | 0.1 | 1 | 1 |
| 10 | "recipeList > Route" | 0 | 1 | 1 |

► Array(11)

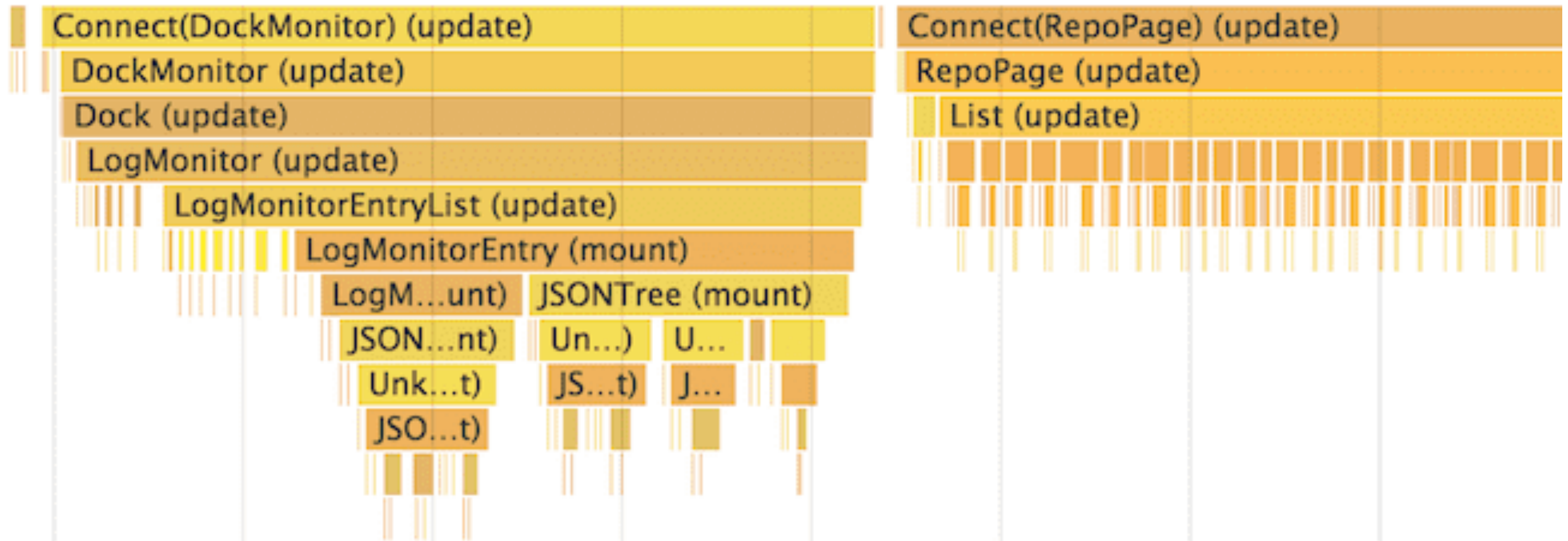
ReactPerf.js:32

| (index) | Owner > Component | Inclusive wasted time (ms) | Instance count | Render count |
|---------|------------------------------|----------------------------|----------------|--------------|
| 0 | "recipeList > Link" | 0.42 | 6 | 6 |
| 1 | "RecipesContainer > Planner" | 0.27 | 1 | 1 |
| 2 | "recipeList > recipeSearch" | 0.1 | 1 | 1 |
| 3 | "RecipesContainer > Route" | 0 | 1 | 1 |
| 4 | "recipeList > Route" | 0 | 1 | 1 |

► Array(5)

⁴Image Source: [Daniel Park](#)

We can also visualize the performance of all components:⁵ ⁶



⁵[An advanced guide to profiling performance using Chrome Devtools](#)

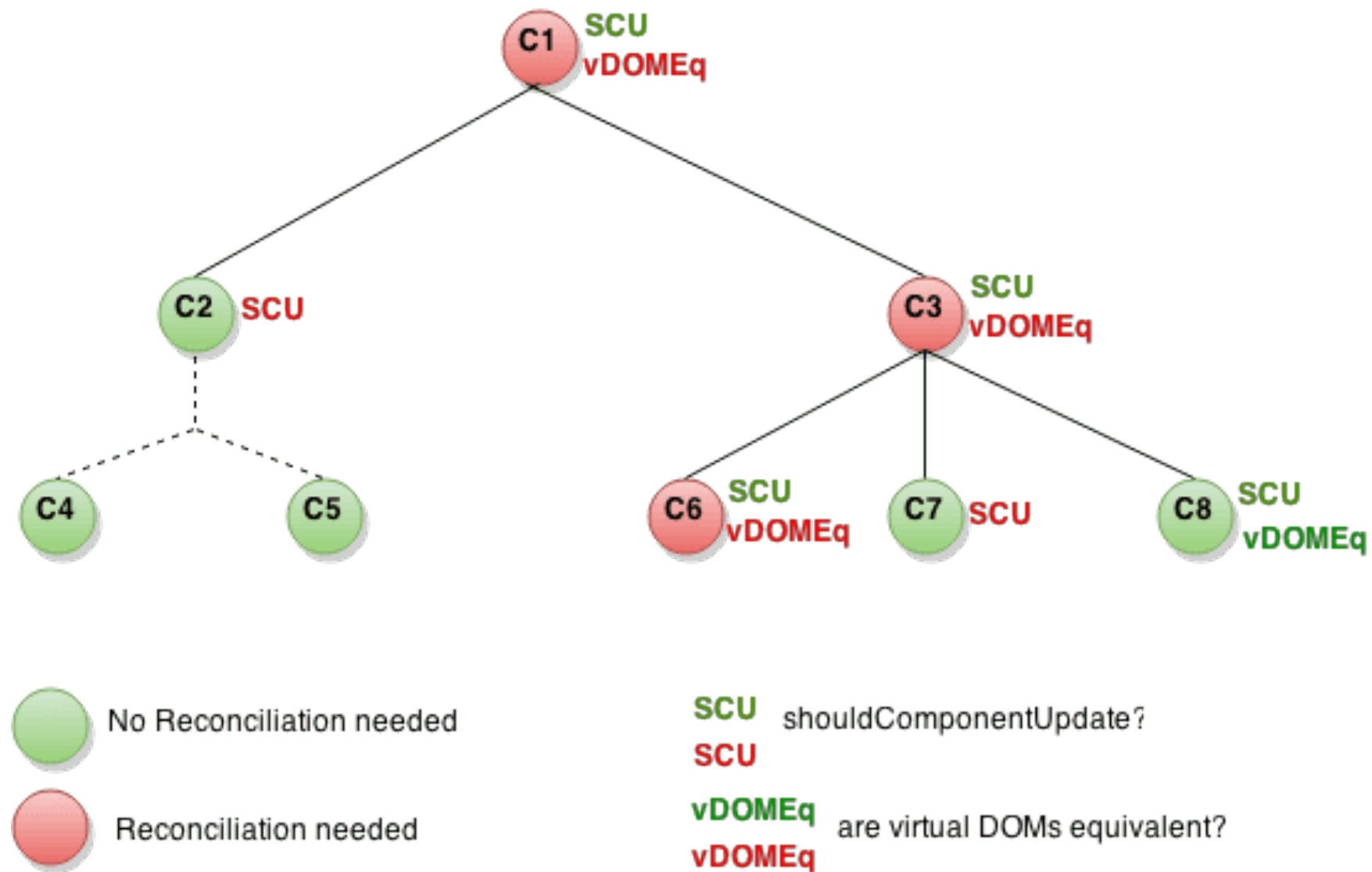
⁶[Image source](#)

How to eliminate time wasted?

By avoiding reconciliation, i.e., only rendering when there is actually an update, using `shouldComponentUpdate()`.

Definition: For components that implement `shouldComponentUpdate()`, React will only render if it returns `true`.

```
function shouldComponentUpdate(nextProps, nextState) {  
    return true;  
}
```



⁷[Image source](#)

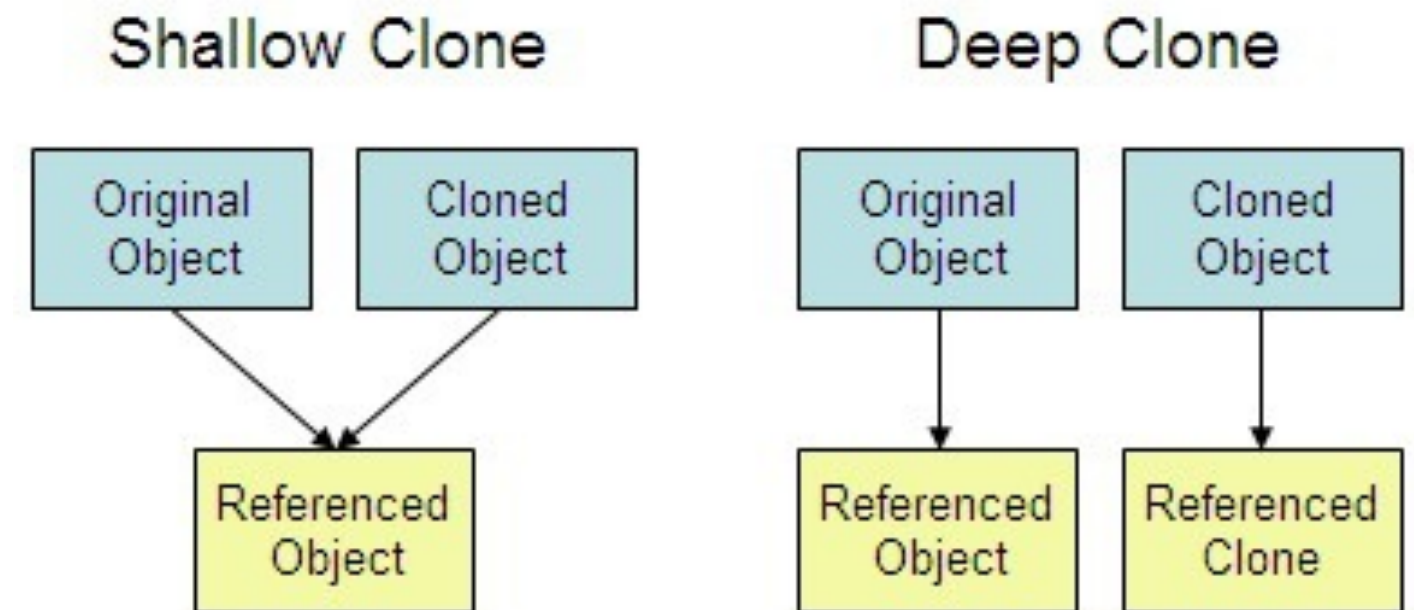
An example of *shallow* comparison to determine whether the component should update:

```
shouldComponentUpdate(nextProps, nextState) {  
    return this.props.color !== nextProps.color;  
}
```

Detour: Shallow vs. Deep Comparison⁸

Shallow Comparison: When each property in a pair of objects are compared using *strict* equality, e.g., using `===`.

Deep Comparison: When the properties of two objects are recursively compared, e.g., using Lodash `isEqual()`.



⁸[Image source](#)

React.PureComponent

React provides a component called `PureComponent` that implements `shouldComponentUpdate()` and only diffs and updates when it returns `true`.

Note that any child of `PureComponent` must be a `PureComponent`.

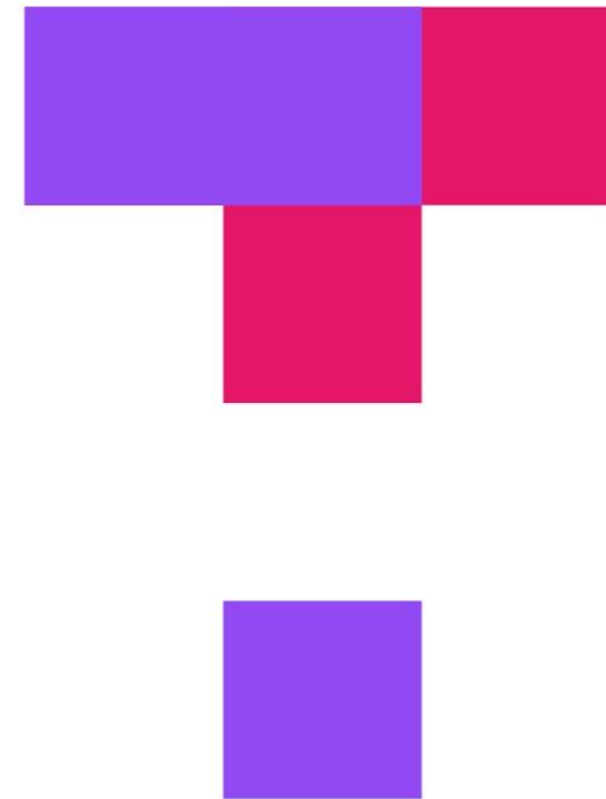
Other Ways of Optimizing Performance

- >> Not mutating objects
- >> Using immutable data structures
- >> Using the production build of React
- >> Many more,...

Further Reading on React Performance

- >> [21 Performance Optimization for React Apps](#)
- >> [Efficient React Components: A Guide to Optimizing React Performance](#)
- >> [ReactJS.org: Optimizing Performance](#)

TopHat Questions



TOP HAT

Advanced Asynchronous Updating

Getting data within `componentDidMount()`

Ideally, we want to interact with the server in the following way.
What would happen here?

```
componentDidMount() {  
  const res = fetch('https://example.com')  
  const something = res.json()  
  this.setState({something})  
}
```


But we end up following up `fetch()` with a series of `then()`s.

```
componentDidMount() {  
  fetch('https://example.com')  
    .then((res) => res.json())  
    .then((something) => this.setState({something}))  
}
```

`then()` allows us to program asynchronously (by allowing `componentDidMount()` to wait for the Promise to be resolved). Although, this syntax can be unintuitive and not readable.

Programming asynchronously using `async/await`

`async/await` provides syntax to program asynchronously in an intuitive and clean way.

Usage:

- >> `async function()` denotes that the `function()` will work asynchronously.
- >> `await expression` enables the program to wait for `expression` to be resolved.

Example:⁹

```
async componentDidMount() {  
  const res = await fetch('https://example.com')  
  const something = await res.json()  
  this.setState({something})  
}
```

⁹[See in CodePen](#)

async Functions¹⁰

Any function can be asynchronous and use `async`. Useful where the function has to wait for another process.

```
async addTag(name) {  
    if(this.state.tags.indexOf(name) === -1) {  
        await this.setState({tags: [...this.state.tags, name]});  
        this.setCourses();  
    }  
}
```

¹⁰ See [example in CodePen](#) (line 70)

APIs for advanced interaction

Interaction Libraries

- >> [react-beautiful-dnd: Examples](#)
- >> [react-smooth-dnd: Demo](#)
- >> [React DnD: Examples](#)

Component Libraries

- >> Material UI
- >> Material Kit React: Demo
- >> Rebass
- >> Grommet
- >> React Desktop : Demo

Managing Data

>> React Virtualized: Demo

What did we learn today?

- >> Optimizing performance in React
- >> Advanced asynchronous updating
- >> APIs for advanced interaction

Assignment Q & A

Midterm Q & A