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

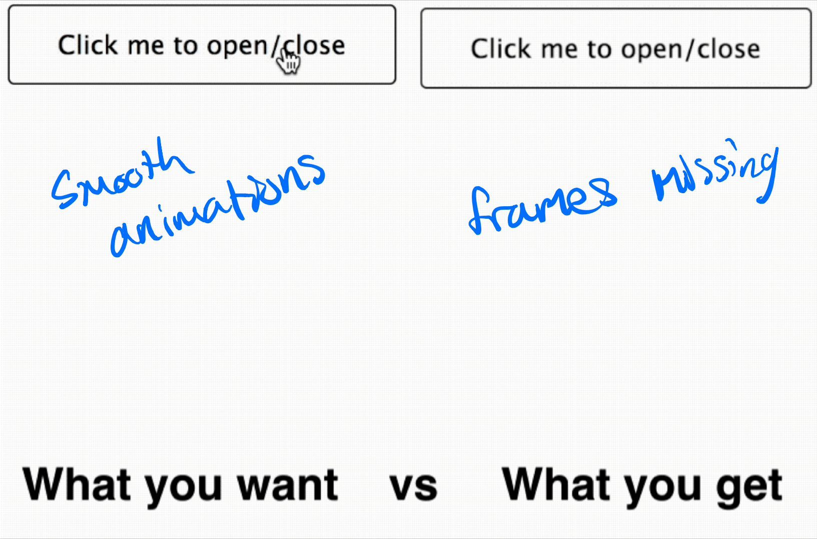
TopHat Questions

Optimizing *Performance* in React

Why do we need to worry about performance?[1]

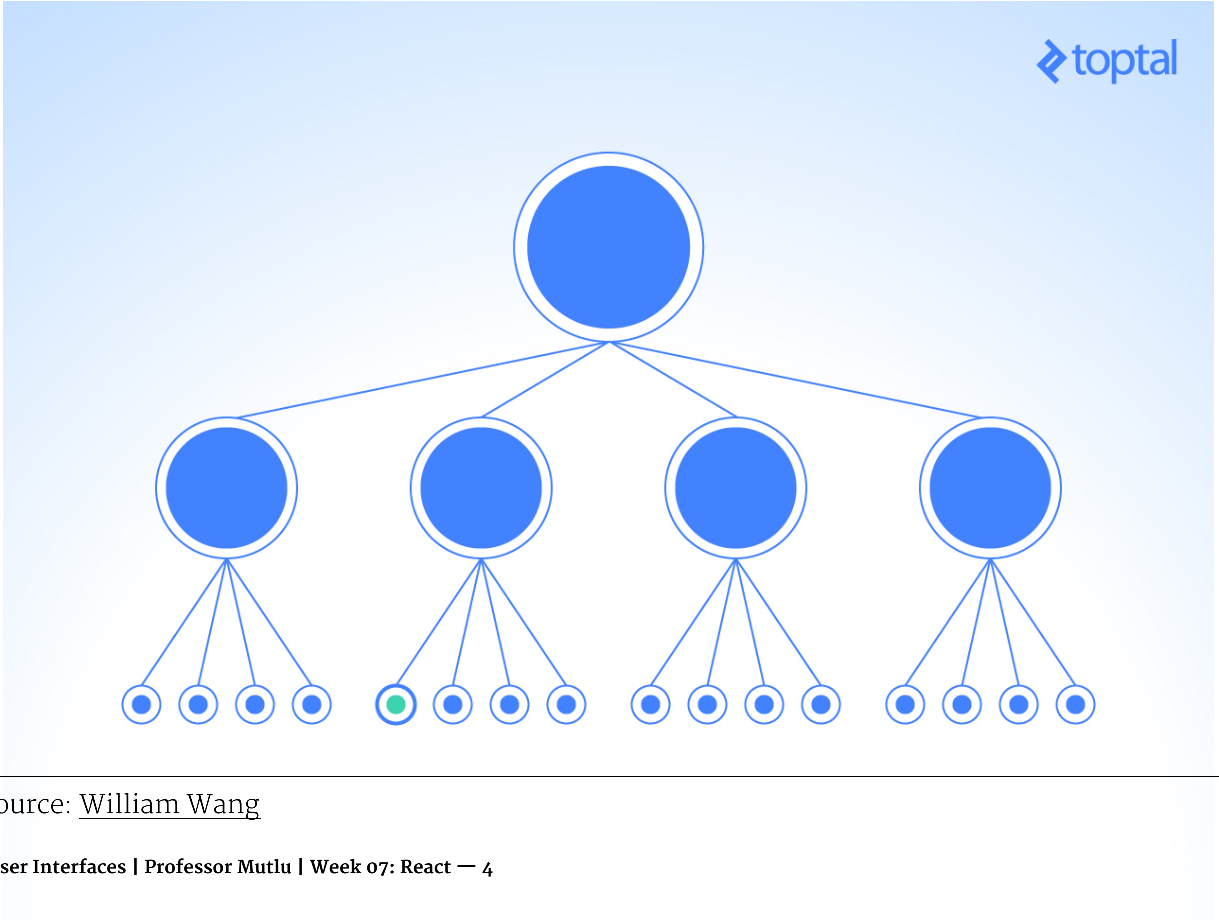
As the complexity of your application scales, performance will necessarily degrade.

Why? And what do we do about it?



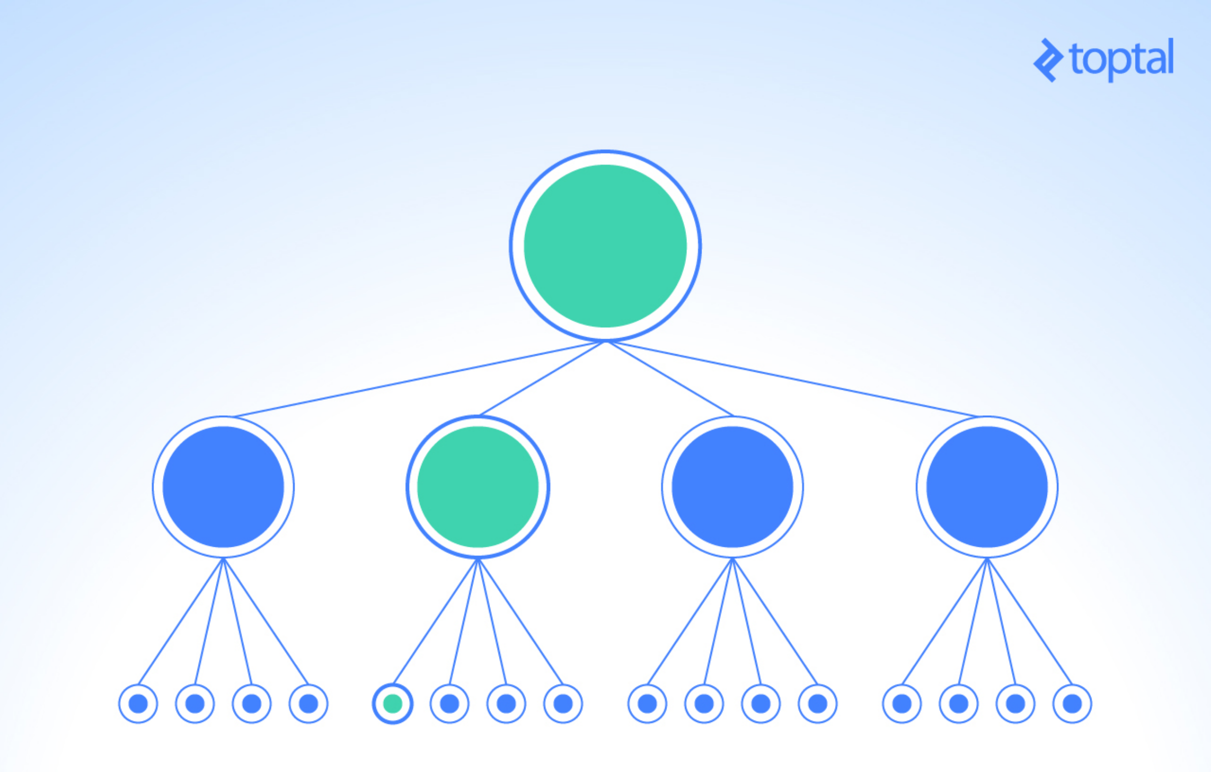
[1] Image Source: Noam Elboim

[2]



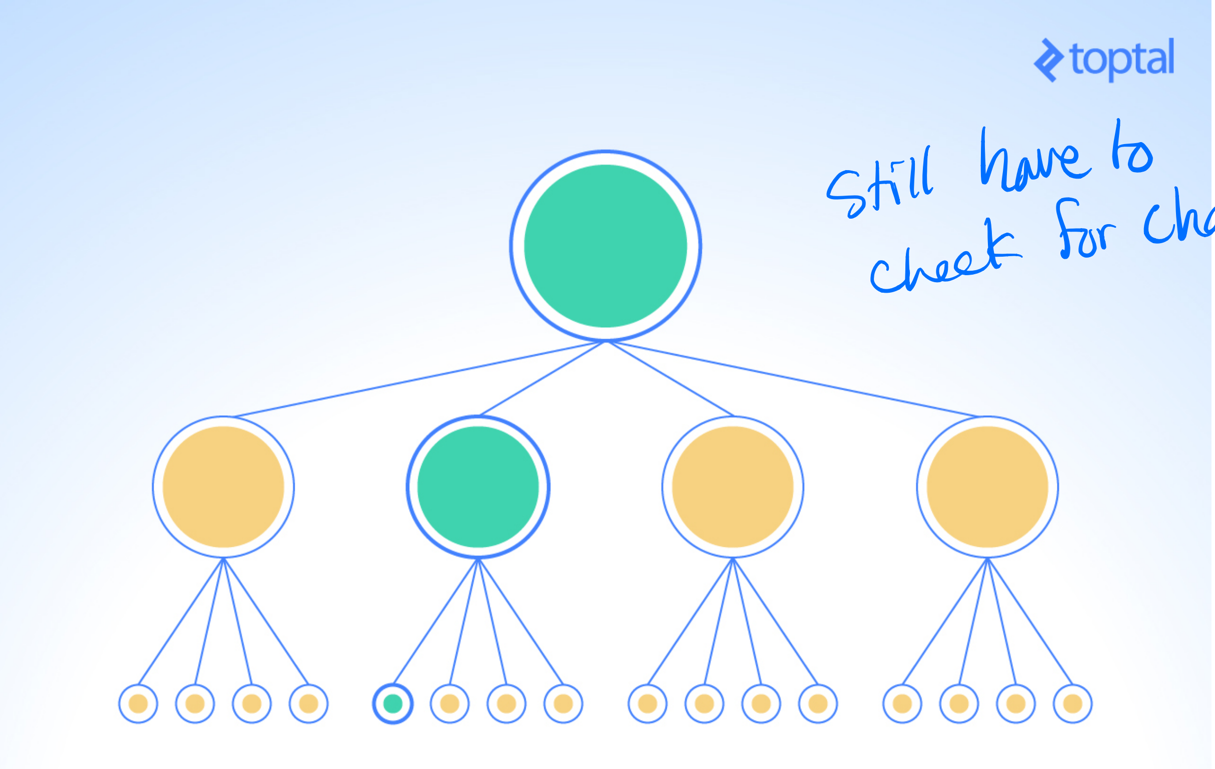
[2] Image Source: William Wang

[2]



[2] Image Source: William Wang

[2]



Still have to check for changes

[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()
```

← start the monitor

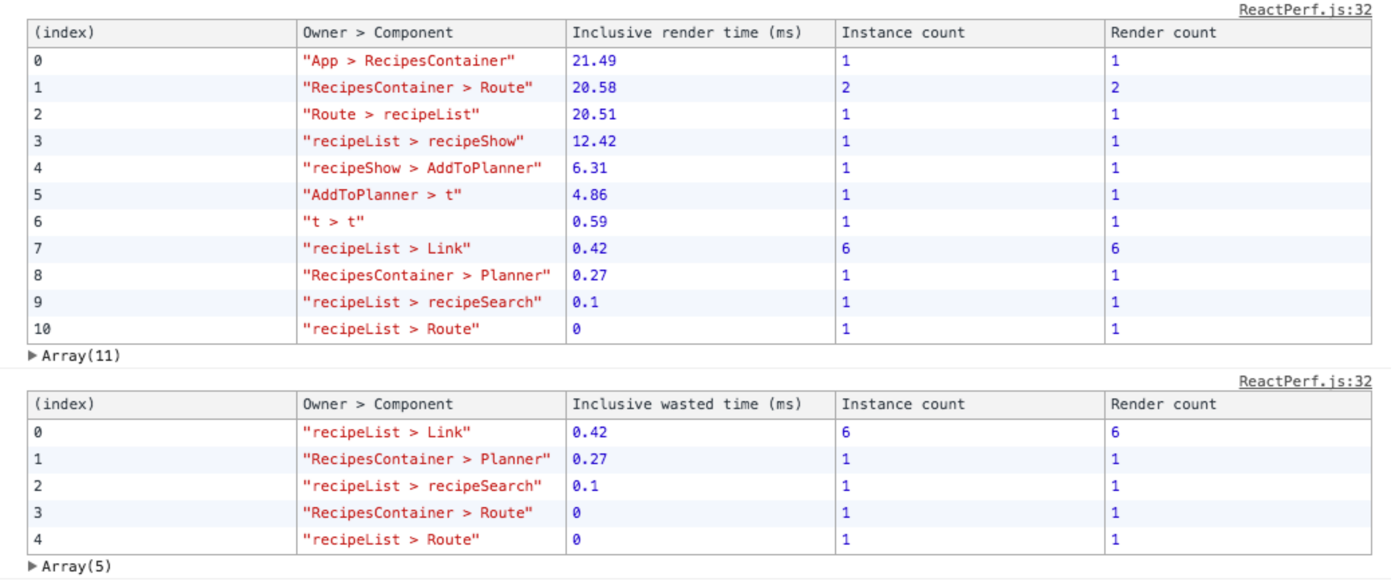
← stop the monitor

[3] ReactJS.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)

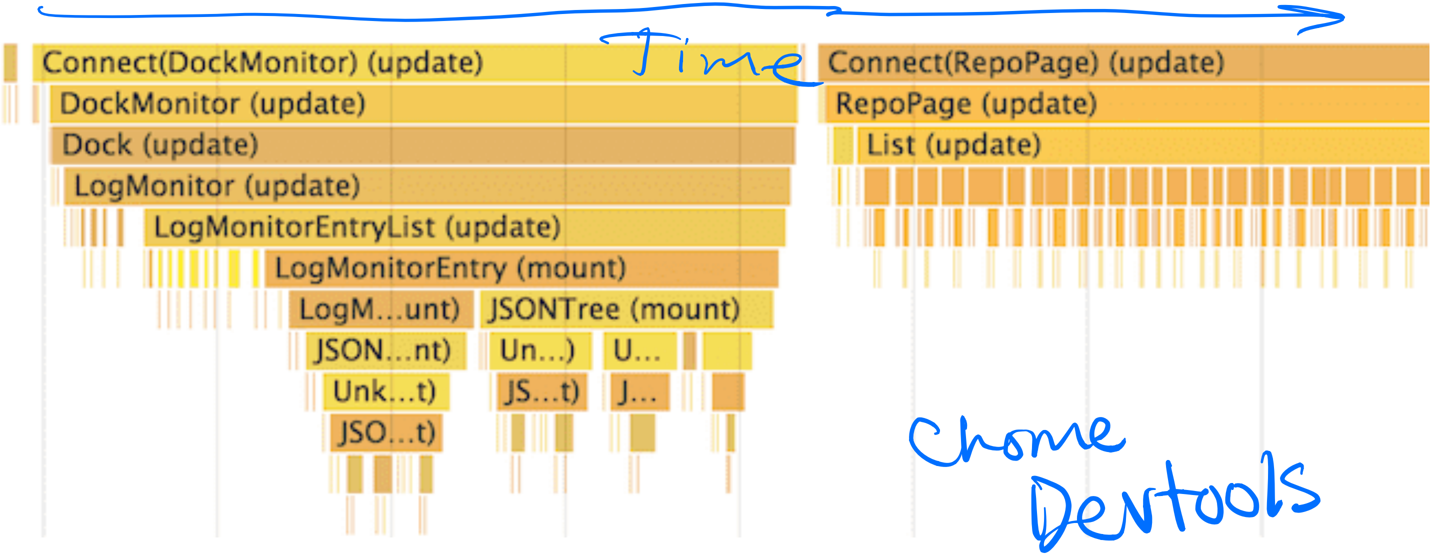
ReactPerf.js:32

(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)

[4] Image Source: Daniel Park

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



Time

Chome Devtools

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

[6] Image source

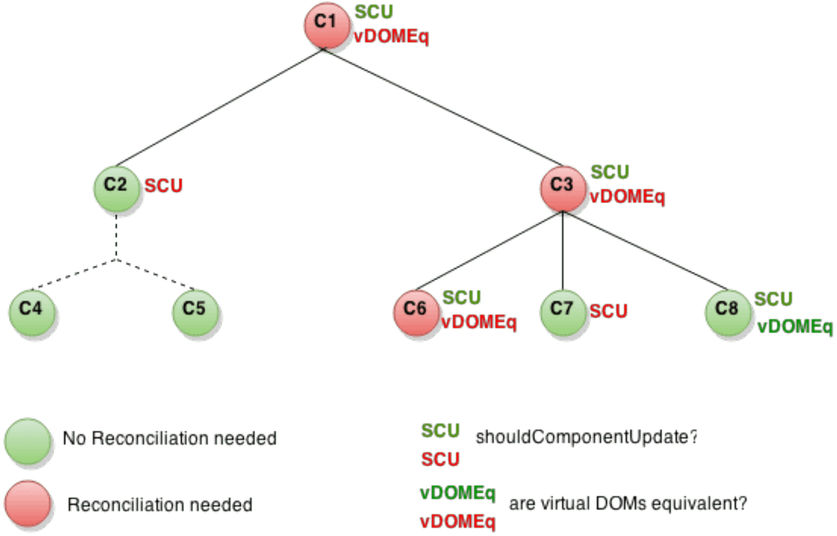
How to eliminate time wasted?

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]



[7] Image source

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

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

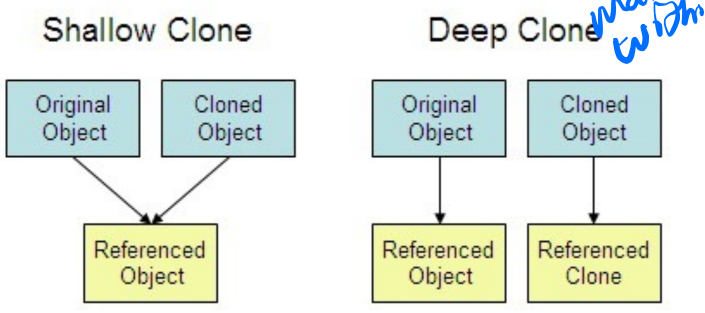
Detour: Shallow vs. Deep Comparison[8]

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

if === is true, the object must be the same. if false, it must be different, even if a child matches within

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

recursively compares the properties



[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

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.

expressive, but can get hard to understand

Programming asynchronously using `async/await`

newer syntax

`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})
}
```

async

[9] See in CodePen

async Functions[10]

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

Drag and Drop Libraries

Component Libraries

- Material UI
- Material Kit React: Demo

} Google

- Rebass

} Small

- Grommet

} used by lots of web apps

- React Desktop : Demo

} behaves like desktop applications

Managing Data

- React Virtualized: Demo

stress testing

What did we learn today?

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

Assignment Q & A

Midterm Q & A