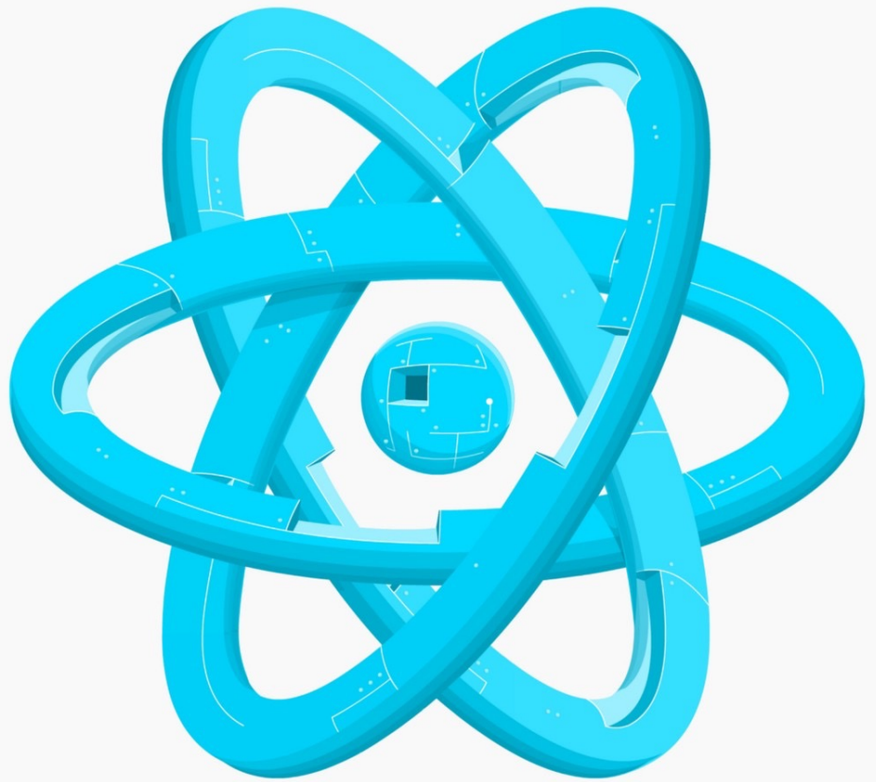


BUILDING WEB APPS USING

React JS

by mostafa fouad

Hello!

You should be familiar with

- Modules
- Using `const` and `let`
- Classes
- Arrow functions
- Template literals
- Object de-structuring

What is React

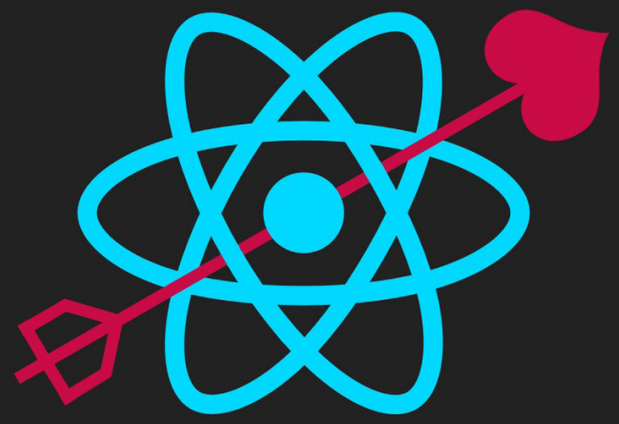
- React is a JS library for building interactive user interfaces.
- Built, open-sourced and maintained by Facebook.
- It's a JS library, not a framework and not a plugin.
- It's the V in MVC.

Components are the future

React builds on the concept of splitting the application into small re-usable components

WHY REACT?

- React is easy to learn
- React uses one-way data flow
- Can be used with any stack (even in non-SPAs)
- No templates or directives
- Reactive updates are dead simple (fully declarative)
- Events behave in a consistent, standards-compliant way in all browsers (even in IE8)



```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
import './index.css';

React.cre
ReactDOM.
registerS
```

createClass function React.createClass<P, S>(spec:
createElement
createFactory
Children
ClipboardEvent
ClipboardEventHandler
CSSPercentage
ChangeTargetHTMLFactory
ChangeTargetHTMLProps
ReactChildren

FIRST EXAMPLE

Hello, React!

Rendering elements

```
ReactDOM.render(element, container);
```

Rendering elements

Valid React elements are:

- Strings
- Components or HTML tags
- Array of Strings, Components or HTML tags

Creating elements

```
React.createElement(name, props, children);
```

Components are like JavaScript functions.

Components

They accept arbitrary inputs (called “props”).

They return React elements describing what should appear on the screen.

How to create a component?

The simplest way to define a component is to write a JavaScript function

```
const Welcome = function(props) {
  return `Hello, ${props.name}!`;
};

ReactDOM.render(
  React.createElement(
    Welcome, { name: 'React' }
  ),
  document.getElementById('root')
);  // Renders 'Hello, React!'
```

</>

JSX

- JSX is a syntax extension to JavaScript
- JSX is used with React to describe what the UI should look like
- JSX elements are compiled to React.createElement() calls

```jsx
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

```jsx
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

Component Props

- Props can be anything
- Props should be immutable (read-only)
- Props can have default values
- Props can be validated at runtime

</>

If props should be immutable...

How can you update the UI?

Class based components

- Class based components are like functional components, only using a class

```
class Welcome extends React.Component {
  render() {
    return `Hello, ${this.props.name}!`;
  }
}

ReactDOM.render(
  React.createElement(
    Welcome, { name: 'React' }
  ),
  document.getElementById('root')
);  // Renders 'Hello, React!'
```

```
class Welcome extends React.Component {
  render() {
    return `Hello, ${this.props.name}!`;
  }
}

ReactDOM.render(
  <Welcome name="React" />,


  document.getElementById('root')
);  // Renders 'Hello, React!'
```

Class based components

- Class based components are like functional components, only using a class
- Class based components have access to a local state

State

- State is like props
- State can be changed to update the UI
- Component state is private and is fully controlled by the component
- State can have initial values

</>

Three things

you should know about the state

1. Never Modify State Directly

```
/* This is wrong! */
this.state.count += 10;

/* This is correct */
this.setState({ count: currentCount + 10 });
```

</>

Three things

you should know about the state

2. State Updates May Be Asynchronous

```
console.log(this.state.count); // 0
this.setState({ count: 100 });
console.log(this.state.count); // still 0
```

</>

Three things

you should know about the state

2. State Updates May Be Asynchronous

```
console.log(this.state.count); // 0

this.setState({ count: 100 }, () => {
  console.log(this.state.count); // 100
});
```

Three things

you should know about the state

3. State Updates are Merged

When you call `setState()`, React merges the object you provide into the current state, it doesn't replace the whole object.

IMPORTANT

State should only be used to store data that affects the visual output of the UI

Rule of thumb

If it is not used in `render()`, then it should not be in the state

</>

Functional Components

vs

Class based Components

Functional vs Class based components

- A functional component is simply a function that receives a single parameter `props` and returns a React element
- Functional components are called that because they are *literally* JavaScript functions
- Functional components are shorter to write

Functional vs Class based components

- Class based components have local state while functional components don't
- Class based components also have access to life cycle hooks while functional components don't

Component life cycle methods

React provides many methods or “hooks” that are called during the life cycle of a component

Component life cycle methods

- constructor(props)
- componentWillMount()
- componentDidMount()
- componentWillReceiveProps(nextProps)
- shouldComponentUpdate(nextProps, nextState)
- componentWillUpdate(nextProps, nextState)
- componentDidUpdate(prevProps, prevState)
- componentWillUnmount()

</>

constructor(props)

The component class constructor is called whenever a new object is created

Do

- Initialise state based on the received props
- Bind functions that will be passed as callbacks

Do not

- Cause any side effects or subscriptions (Ajax calls ... etc)

componentWillMount()

Called right before the component is mounted

Do

- Use setState() if needed, it's free

Do not

- Cause any side effects or subscriptions (Ajax calls ... etc)

componentDidMount()

Called immediately after the component is mounted

Do

- Initialization that requires DOM nodes
- Load data from a remote network
- Setup subscriptions, but don't forget to clean up

Do not

- Use setState() as it will result in an extra render()

componentWillReceiveProps(nextProps)

Called before a mounted component receives new props

Do

- Use to update the state in response to prop changes

Do not

- Cause any side effects or subscriptions (Ajax calls ... etc)

shouldComponentUpdate(nextProps, nextState)

Invoked before rendering when new props or state are being received. Its return value determines whether the component will update or not.

Do

- Enhance performance of a slow component

Do not

- Cause any side effects or subscriptions (Ajax calls ... etc)
- Never use this.setState()

componentWillUpdate(nextProps, nextState)

Invoked just before rendering when new props or state are being received. Not called for the initial render.

Do

- Use to perform preparation before an update occurs

Do not

- Cause any side effects or subscriptions (Ajax calls ... etc)
- Never use this.setState()

componentDidUpdate(prevProps, prevState)

Invoked immediately after updating occurs. Not called for the initial render.

Do

- Operate on the DOM when the component has been updated
- Conditionally load data from a remote network

Do not

- Never use this.setState()

componentWillUnmount()

Invoked immediately before a component is unmounted and destroyed

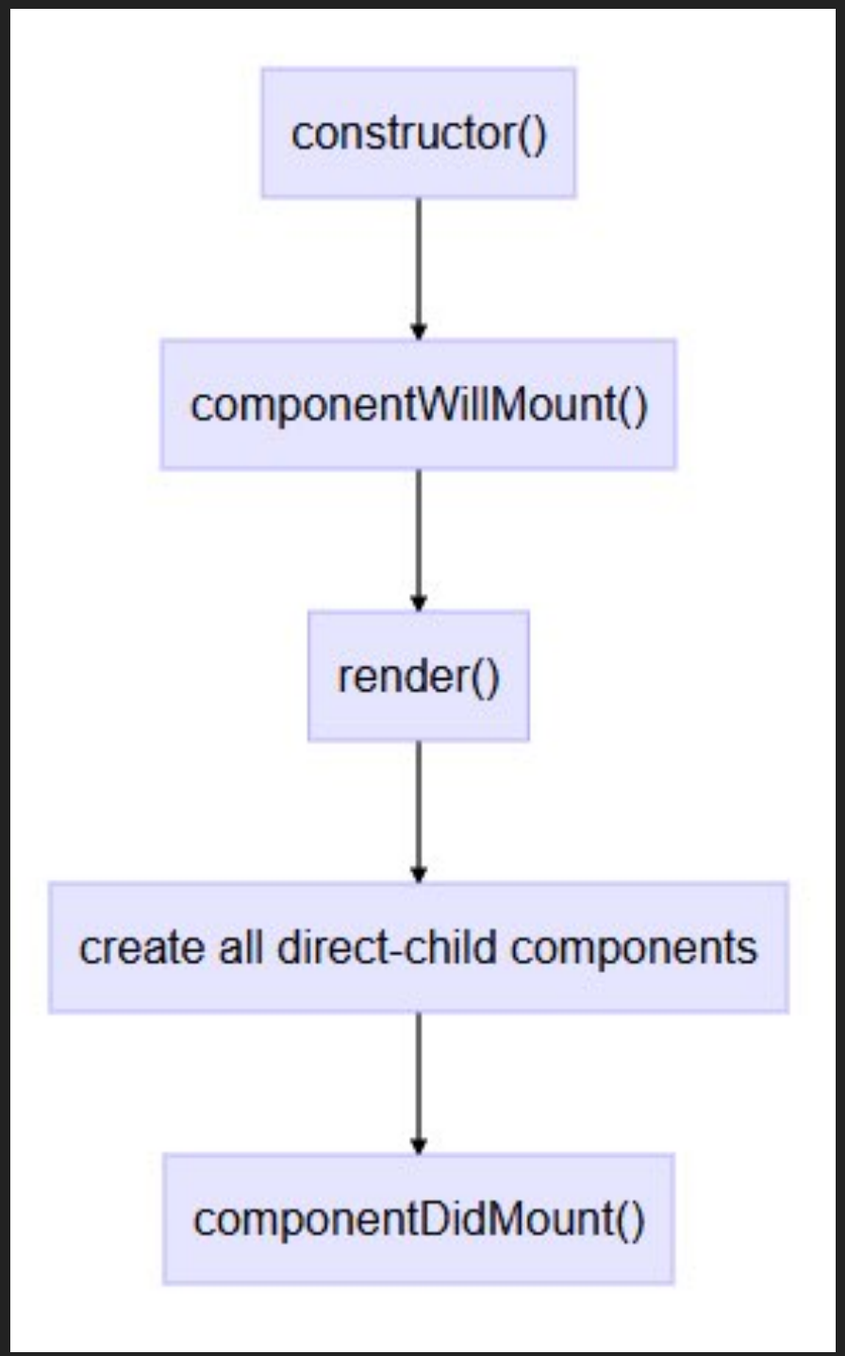
Do

- Perform any necessary cleanup, such as invalidating timers, canceling network requests

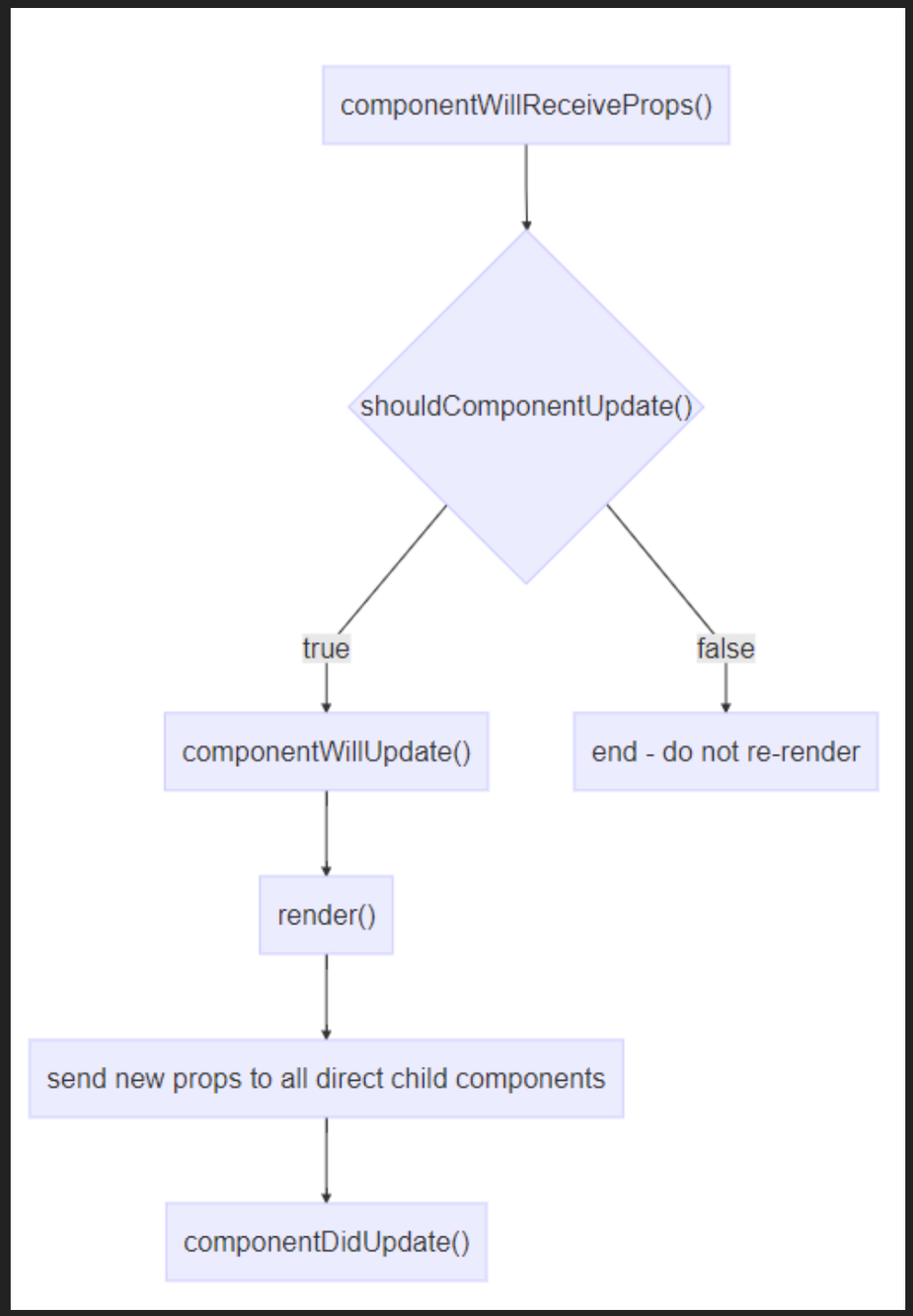
Do not

- Never use this.setState()

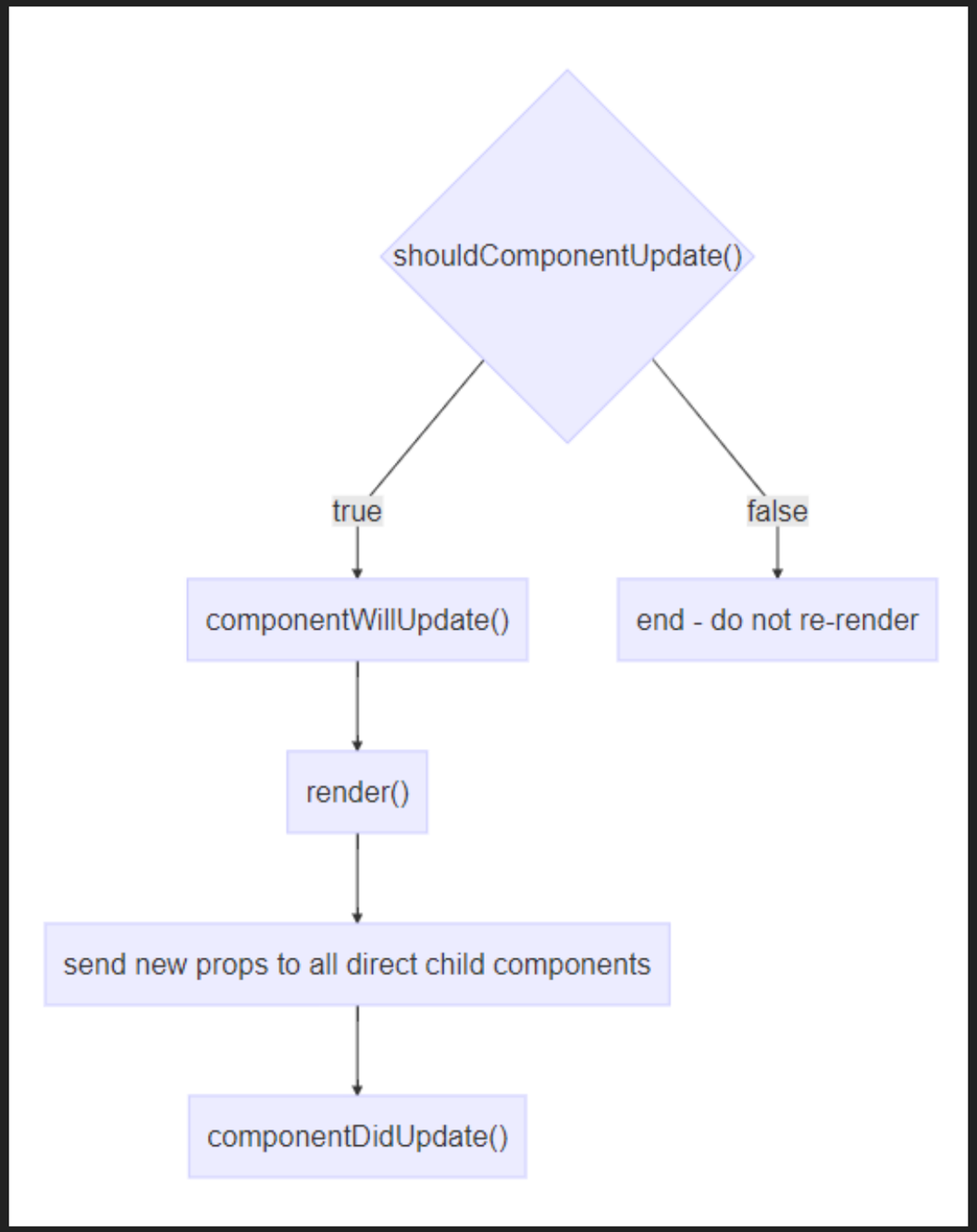
Component creation



New props received



After setState()



Events

- Handling events with React elements is very similar to handling events on DOM elements
- React events are named using camelCase, rather than lowercase

```
<button onClick={this.handleClick}>
  Click me
</button>
```

Events

- Handling events with React elements is very similar to handling events on DOM elements
- React events are named using camelCase, rather than lowercase
- React events are not actually events, they are synthetical events

</>

Be careful

In JavaScript, class methods are not bound by default

If you forget to bind `this.handleClick` and pass it to `onClick`, `this` will be undefined when the function is actually called

</>

Conditional Rendering

Conditional rendering in React works the same way conditions work in JavaScript

Conditional Rendering

Use && operator to conditionally render a component

```
this.state.showBtn && <button>Click me</button>
```

</>

Conditional Rendering

Use a ternary expression to choose between two components

```
isLogged ? <div>Logout</div> : <div>Login</div>
```

</>

Conditional Rendering

Return null to prevent a component from rendering

```
render() { return null; }
```

</>

Rendering a list

```
const numbers = [1, 2, 3, 4, 5];

const listItems = numbers.map((number) => {
  return <li>{number}</li>;
});

ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
);
```

</>

Lists and Keys

- A “key” is a special string attribute you need to include when rendering lists of elements
- Keys help React identify which items have changed, are added, or are removed.
- A key should uniquely identify a list item among its siblings

</>

Lists and Keys

```
const numbers = [1, 2, 3, 4, 5];

const listItems = numbers.map((number) => {
  return <li key={number}>{number}</li>;
});

ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
);
```

IMPORTANT

Don't use the index of the item if the order of the items may change.

Index as a key

```
<ul>
  <li>first</li>
  <li>second</li>
</ul>
```

```
<ul>
  <li>first</li>
  <li>second</li>
  <li>third</li>
</ul>
```

Only the third item has been created

Index as a key

```
<ul>
  <li>Maadi</li>
  <li>Nasr City</li>
</ul>
```

```
<ul>
  <li>October</li>
  <li>Maadi</li>
  <li>Nasr City</li>
</ul>
```

All items have been re-rendered

Forms

- HTML form elements in React are slightly different from other elements

- HTML form elements naturally keep some internal state

</>

Refs and the DOM

React allows accessing DOM elements through refs

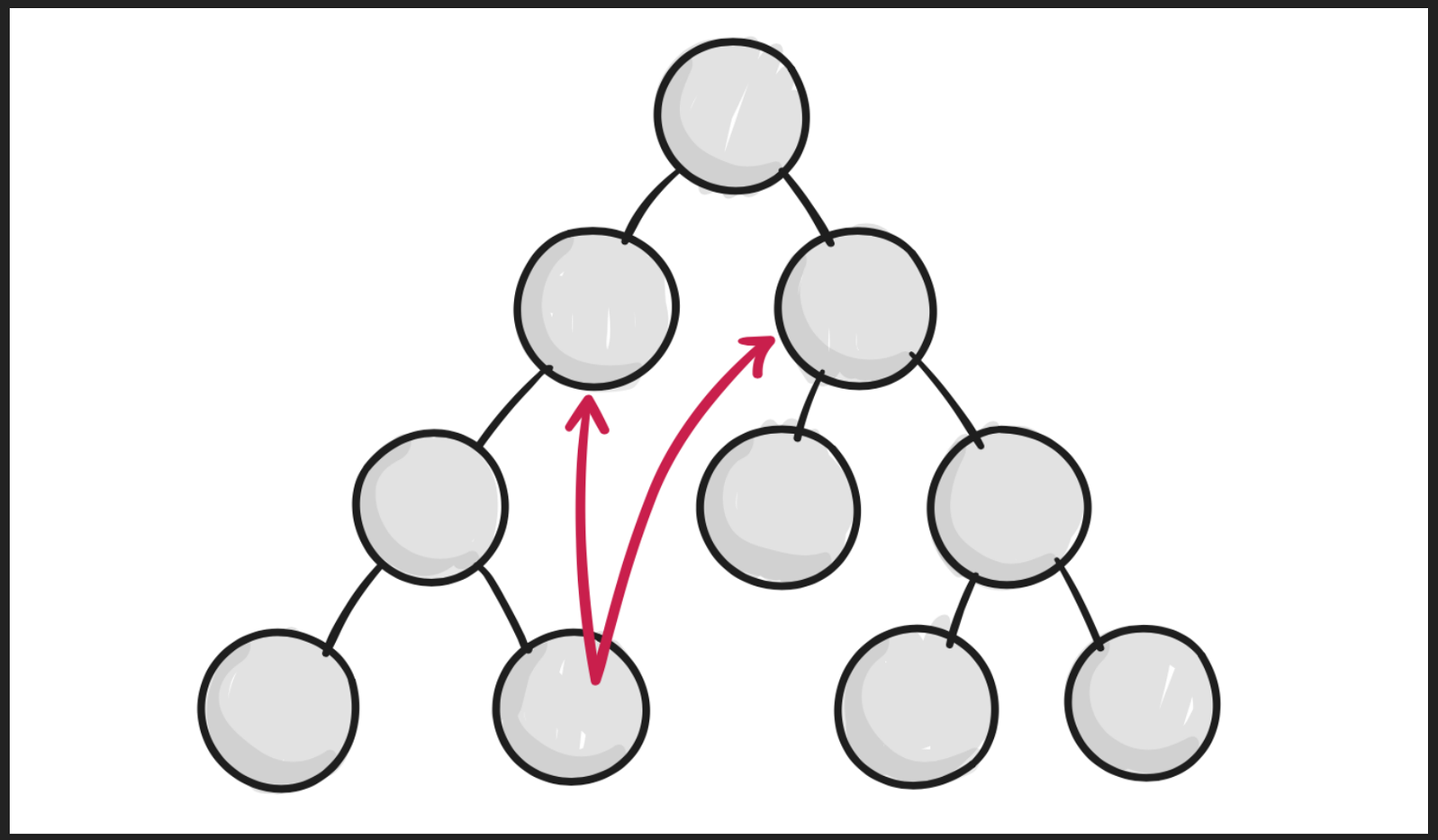
```
<input ref={input => { this.textInput = input; }} />
```

When to Use Refs

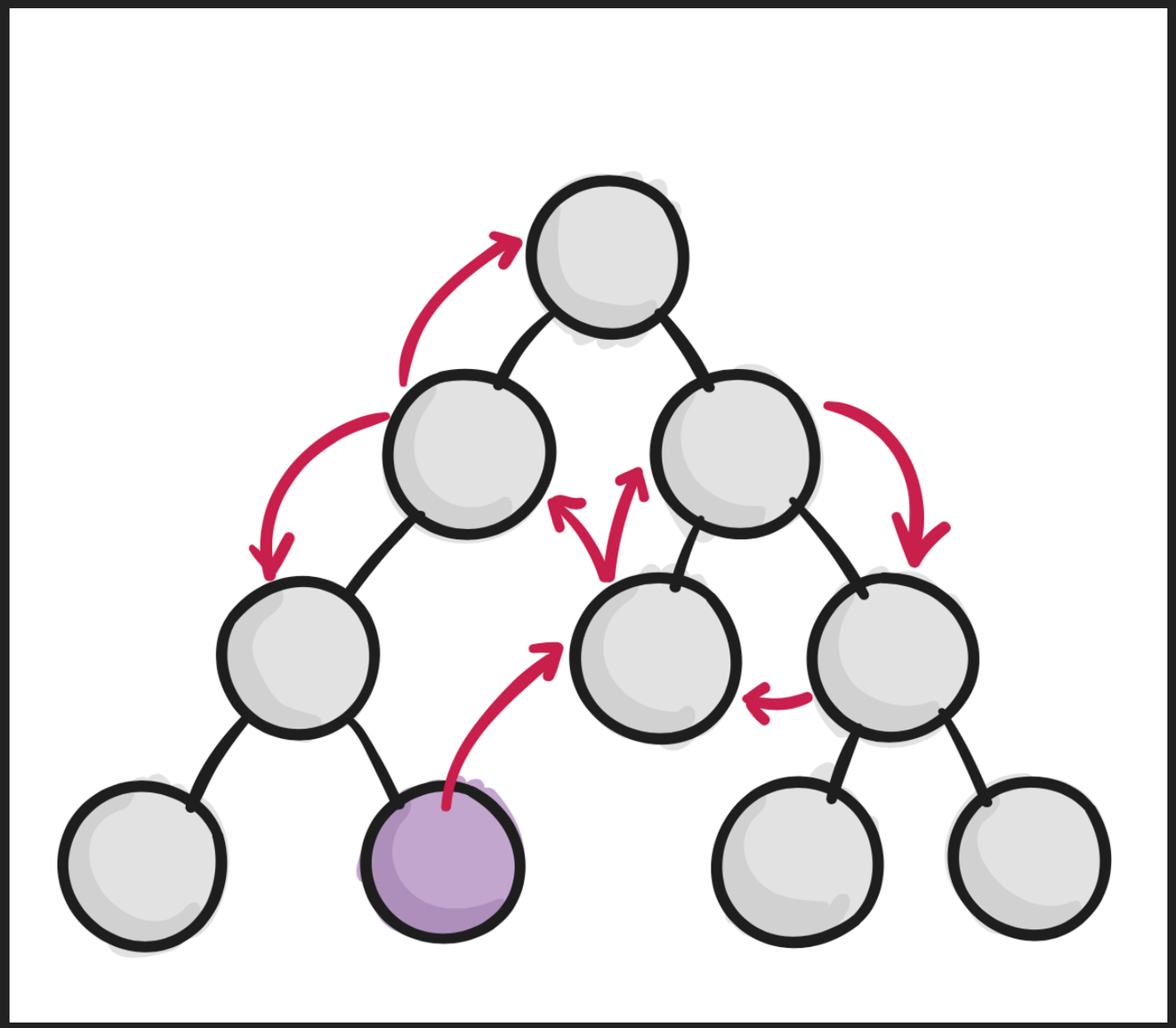
- Managing focus, text selection, or media playback.
- Triggering imperative animations.
- Integrating with third-party DOM libraries.

State management in React

Direct communication between components is bad



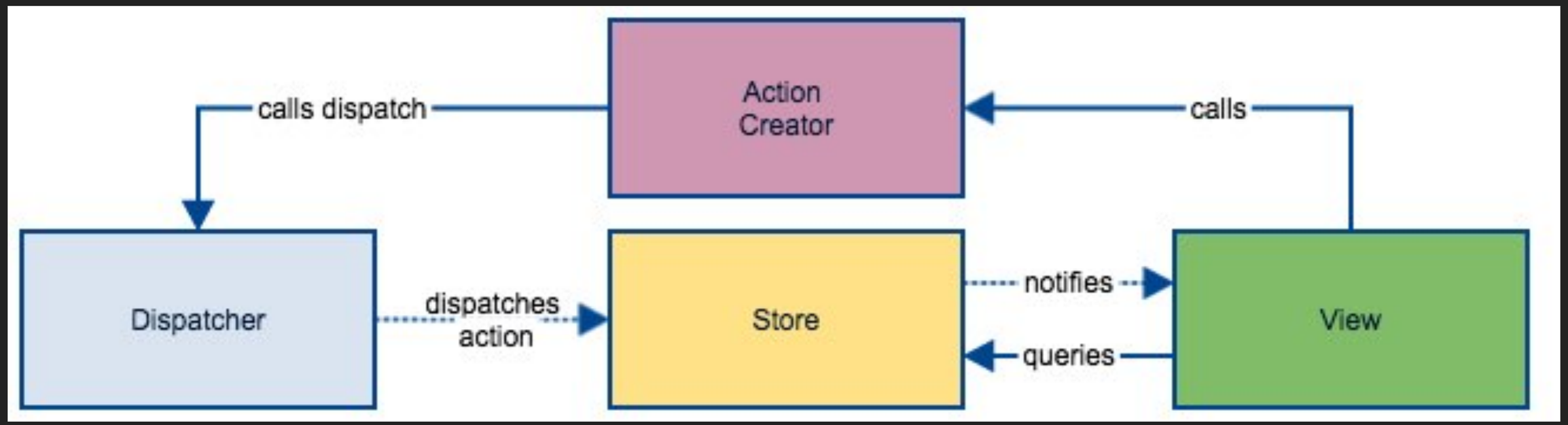
And it gets even worse…



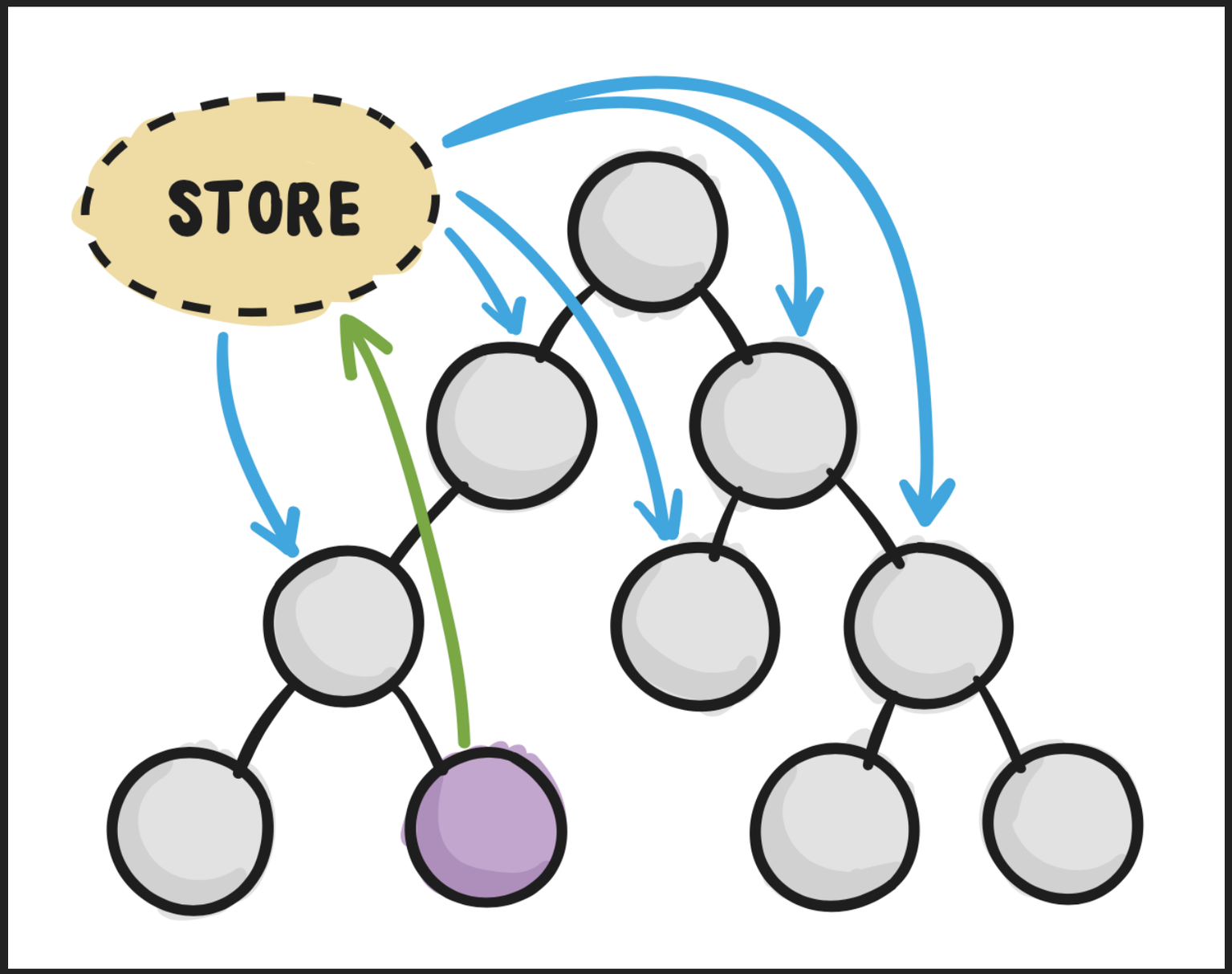
Lift the state up

When several components need to reflect the same changing data, move the shared state up to their closest common ancestor

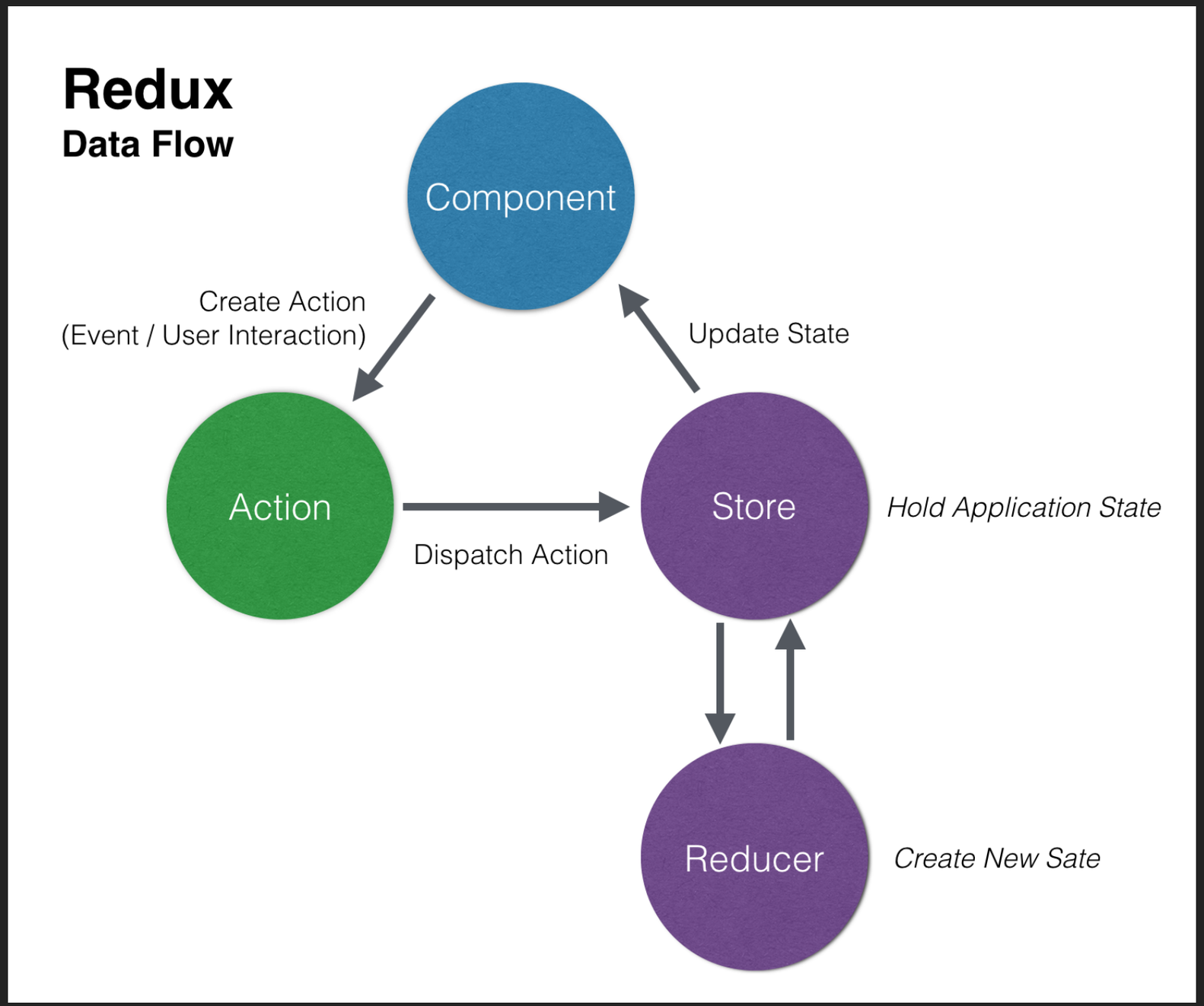
Flux


calls dispatch
Action
Creator
calls
Dispatcher
dispatches
action
Store
notifies
queries
View

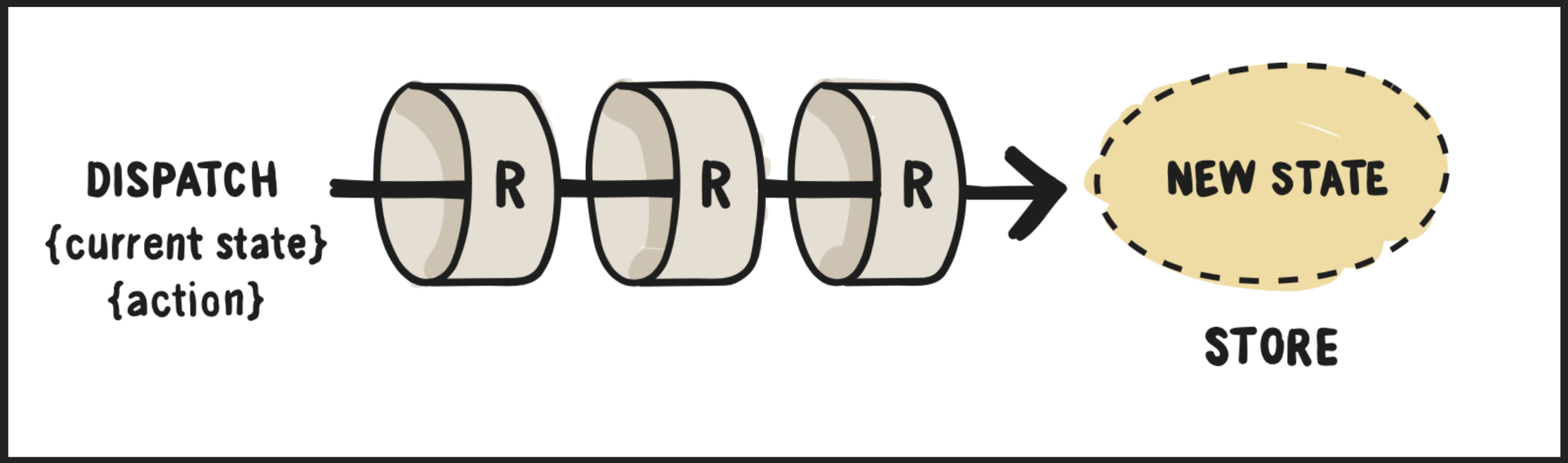
Redux



Redux


Redux
Data Flow
Component
Create Action
(Event / User Interaction)
Update State
Action
Dispatch Action
Store
Hold Application State
Reducer
Create New Sate

Redux



Speedux

Testing React components

Tips and best practices

Thanks