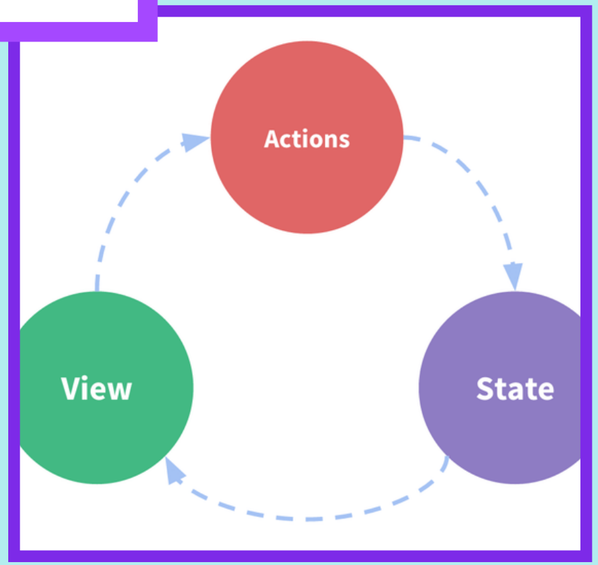
Introduction to State Management

Presented by
Cian O'Leary



What We'll Talk About

- What is State
- What is State Management
- Why is it Important
- Example
- Redux
- Discussion on Local Storage

What to Consider

There is no "correct" answer only Tradeoffs

Things to consider include:

- Performance
- Upfront development complexity
- Performance
- Ongoing Development Cost
- System Awareness need to be productive

What is State?

Events are at the centre of all interactivity.

Events happen and the application needs to React
Let's say a user clicks a button to collapse a sidebar. Whether the sidebar is open/closed is a piece of state

The sum of this state is the state of the application
State can be stored in many places but it constitutes the state of the application, which in turn defines how the application looks and what further events can trigger

State Management

A method by which application data is

- Stored
- Distributed to Components
- Altered by Actions/Events

React Component State

Component State

React Provides multiple ways to store state in a component

Functional Component

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Class Component

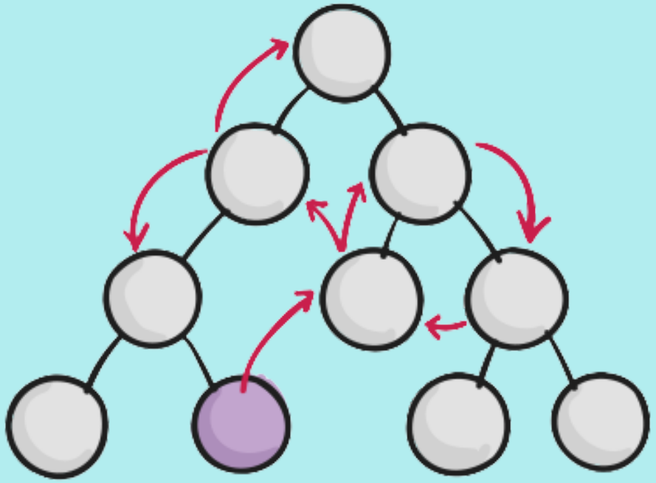
```
class Example extends React.Component {
  state = {
    count: 0
  };

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </div>
    );
  }
}
```

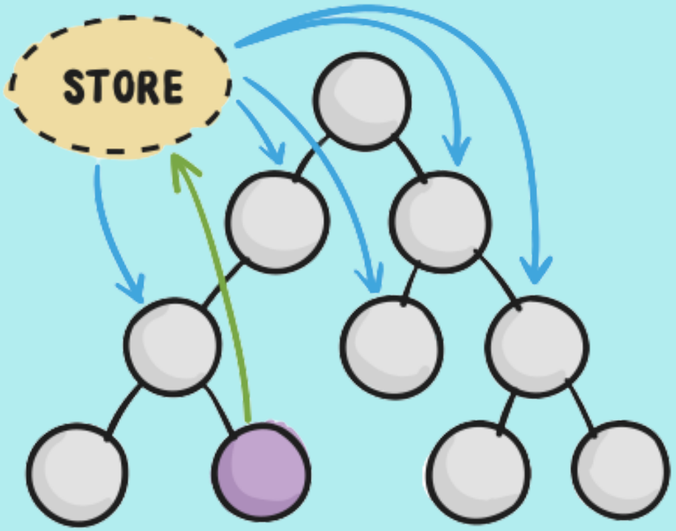
Reference: https://reactjs.org/docs/state-and-lifecycle.html

State Managment

With React State



With Redux
(State Management Library)



Component that fired an event

To Recap

Component Level (Local State)
Contained to a Component or Small Related Group

Examples

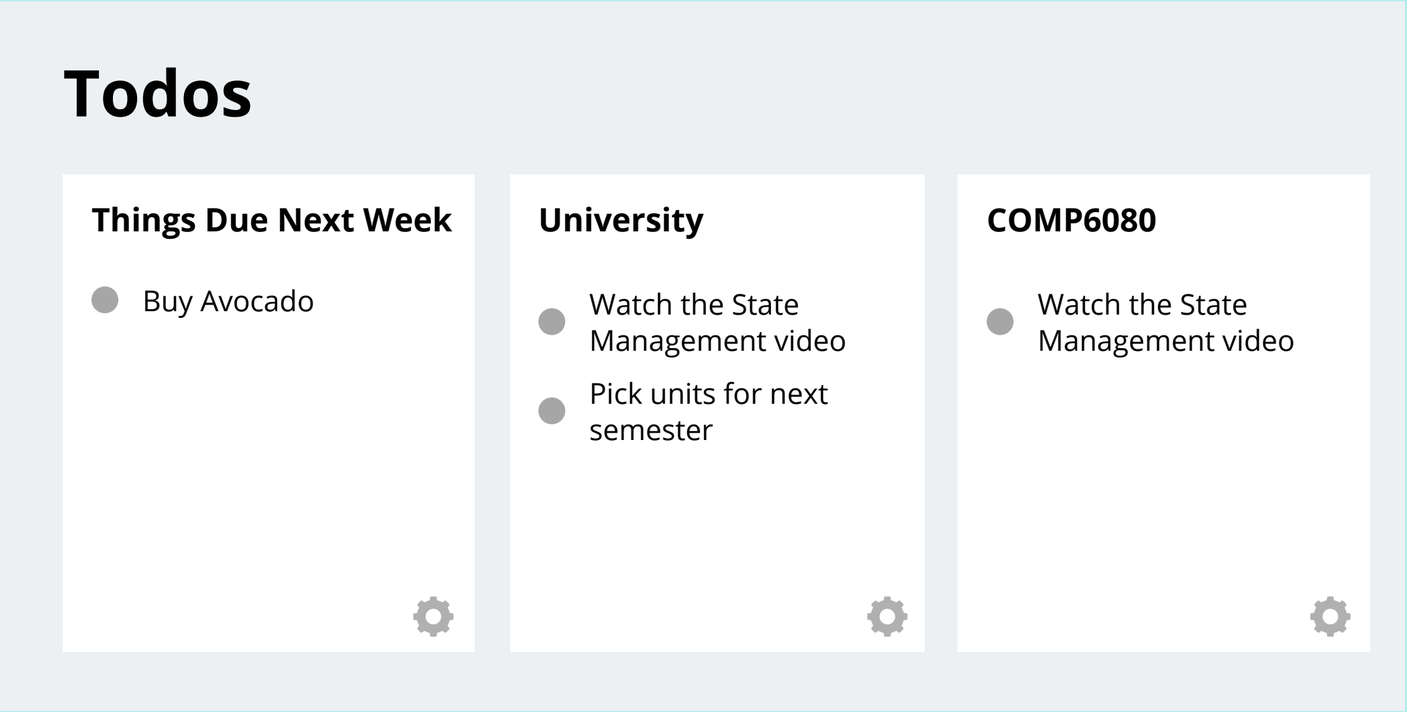
- Text in an input that hasn't been submitted
- What item in a list the user focused on
- Is a dropdown open

Application Level (Global State)
Data required by multiple unrelated components

Examples

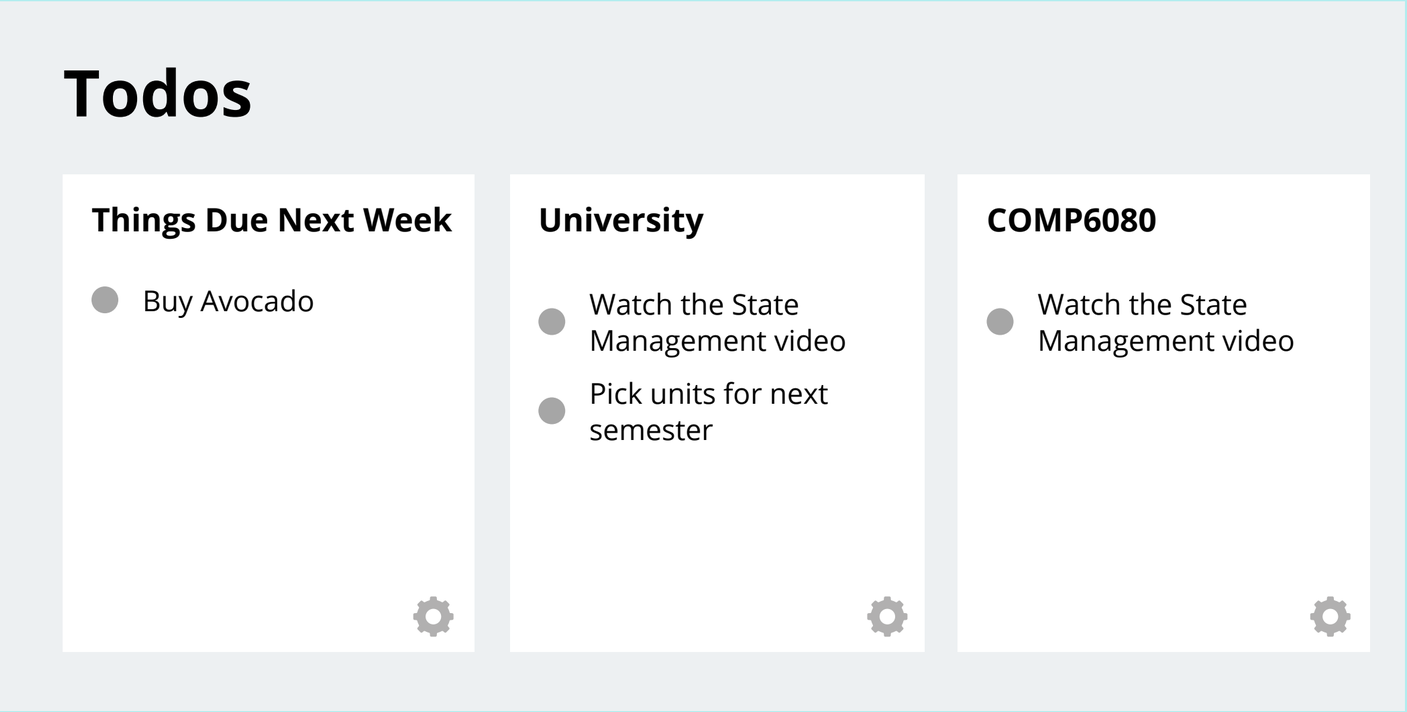
- Who is logged in if any
- Theme settings (Light / Dark Mode)
- What page the user is on

Concrete Example




```
{
 todos: [
  {
   message: "Pick units for next semester",
   tags: [
    "University"
   ],
   due: "2021-10-22T09:30:00"
  },
  {
   message: "Watch the State Management video",
   tags: [
    "University",
    "COMP6080"
   ],
   due: "2021-10-20T09:30:00"
  },
  {
   message: "Buy Avocado",
   tags: [
    "Personal"
   ],
   due: "2021-10-18T15:45:00"
  }
 ]
}
```

Concrete Example




```
{
 todos: [
  {
   message: "Pick units for next semester",
   tags: [
    "University"
   ],
   due: "2021-10-22T09:30:00"
  },
  {
   message: "Watch the State Management video",
   tags: [
    "University",
    "COMP6080"
   ],
   due: "2021-10-20T09:30:00"
  },
  {
   message: "Buy Avocado",
   tags: [
    "Personal"
   ],
   due: "2021-10-18T15:45:00"
  }
 ]
}
```

State Managment Libraries

Today I will be talking about Redux

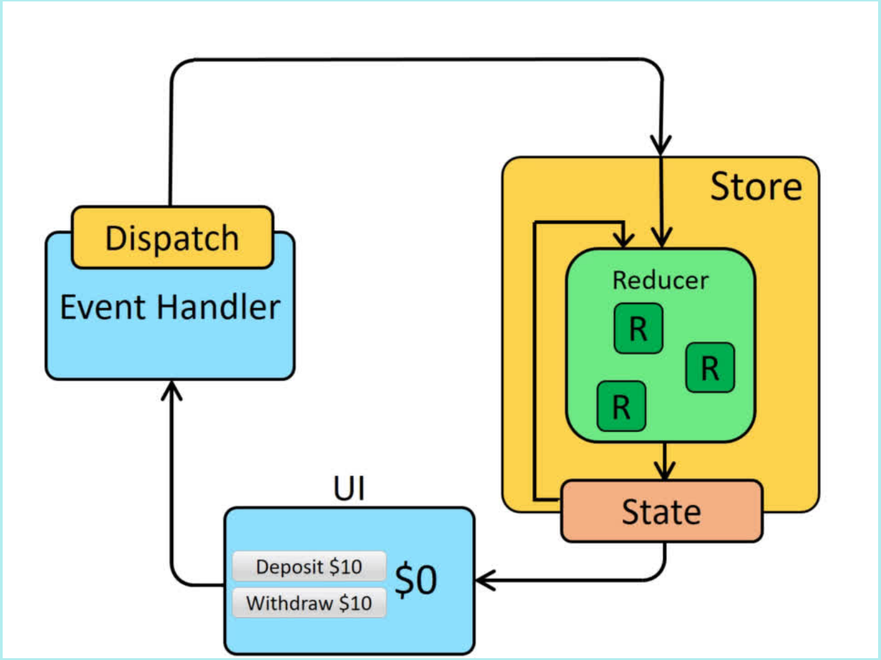
Some other libraries (Vaguely ordered by popularity)

- MobX
- React Query
- XState
- Flux
- Recoil
- Akita

Thankfully most are self-explanatory once you know one and developers are not expected to know more than one or two but rather understand the underlying concepts

Redux

Redux is for storing the global application state in a centralised store and modifying it in predictable ways.



Actions - Plain javascript objects which describe the event or change

Store - the application state at a given point in time.

Reducers - functions that take the store and the action as parameters and return a new state after that action
(state, action) => newState

Code Examples

Store

```
// Initial Store
{
  count: 0,
}
```

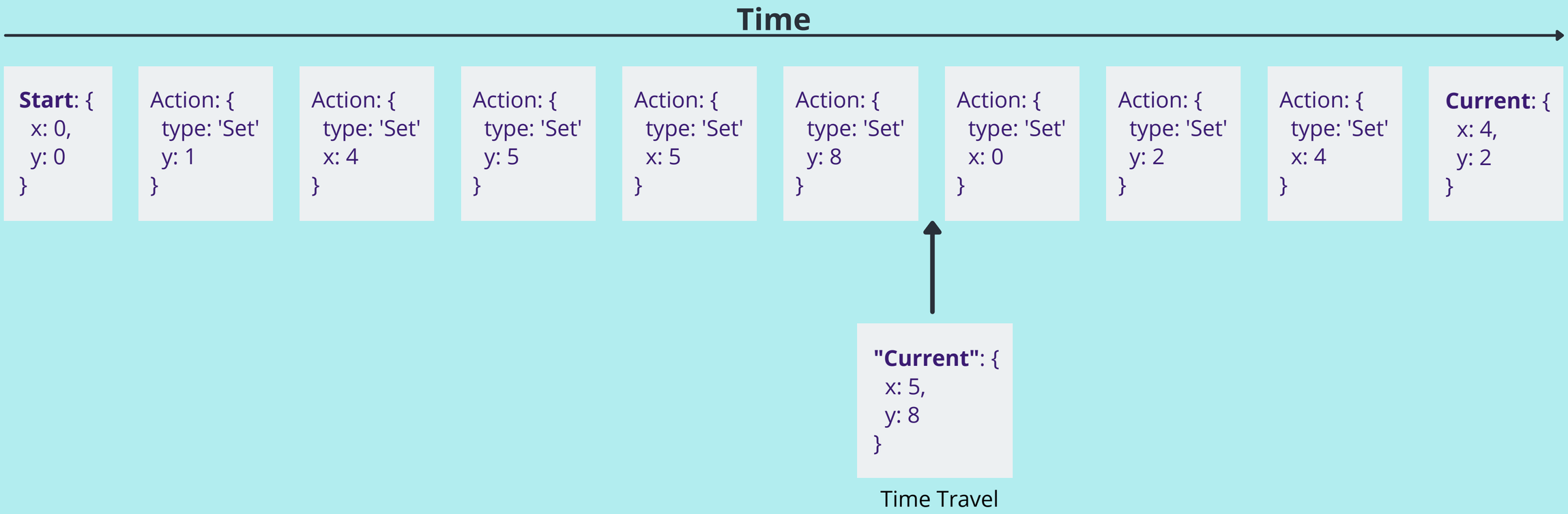
Action

```
// In this case, the action
would be an object of the shape
{
  type: 'increment',
  payload: 10,
}
```

Reducer

```
function appReducer(state, action) {
  // The reducer normally looks at the action type field to decide what happens
  // Traditionally `type` is the only required property of every action
  switch (action.type) {
    // Do something here based on the different types of actions
    case 'increment': {
      return {
        // Return all the other store data if any
        ...state,
        // Payload be the number 10 in the example
        count: state.count + action.payload,
      }
    }
    default:
      // If this reducer doesn't recognise the action type or doesn't
      // care about this specific action, return the existing state unchanged
      return state
  }
}
```

Time Travel


Time
Start: {
x: 0,
y: 0
}
Action: {
type: 'Set'
y: 1
}
Action: {
type: 'Set'
x: 4
}
Action: {
type: 'Set'
y: 5
}
Action: {
type: 'Set'
x: 5
}
Action: {
type: 'Set'
y: 8
}
Action: {
type: 'Set'
x: 0
}
Action: {
type: 'Set'
y: 2
}
Action: {
type: 'Set'
x: 4
}
Current: {
x: 4,
y: 2
}
"Current": {
x: 5,
y: 8
}
Time Travel

Should this be in Redux

Some general guides to help with the question are:

- Do other parts of the application care about this data?
- Do you need to be able to create further derived data based on this original data?
- Is the same data being used to drive multiple components?
- Is there value to you in being able to restore this state to a given point in time (i.e., time travel debugging)?
- Do you want to cache or persist the data?

If you answered yes to any of these, the answer is maybe; if you responded yes to multiple or all, the answer is almost certainly yes.

Redux Logo

What about Local Storage?

Thanks!
See you next time!

More Reading / Ref: https://redux.js.org/