React Hooks

Shan-Hung Wu & DataLab
CS, NTHU

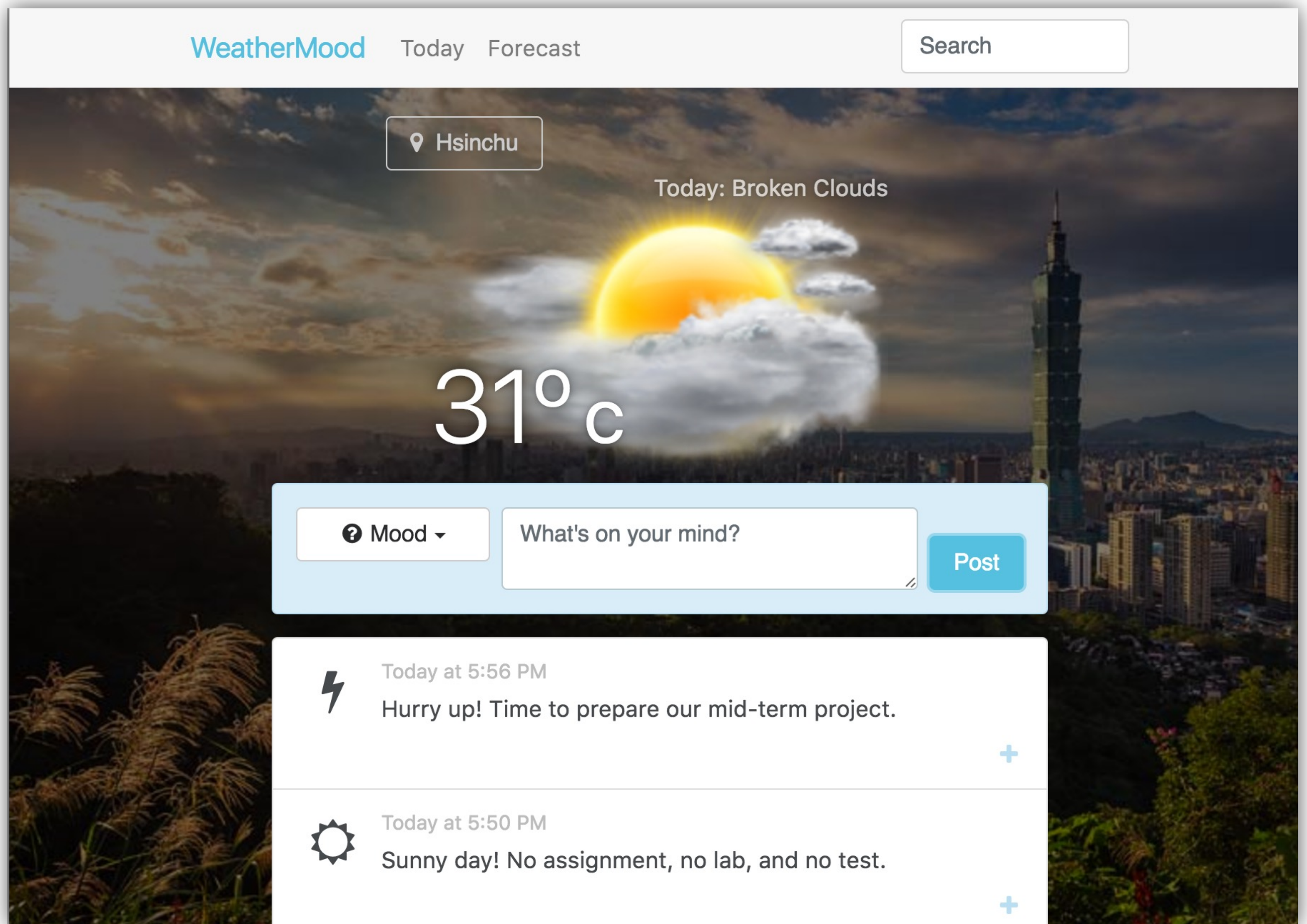
Outline

- WeatherMood: Posts
- Why Hook?
- State Hook
- Effect Hook
- Custom Hook
- Remarks

Outline

- WeatherMood: Posts
- Why Hook?
- State Hook
- Effect Hook
- Custom Hook
- Remarks

Clone `weathermood/react-post`



Today at 5:56 PM

Hurry up! Time to prepare our mid-term project.

Today at 5:50 PM

Sunny day! No assignment, no lab, and no test.

Setup

```
$ npm install --save babel-polyfill \
  moment uuid
```

- Babel Polyfill
  - Use ES6 Promise to simulation asynchronous post fetching
- Moment
  - For displaying date & time
- UUID
  - Generates unique IDs for new posts

API for Posts

```
// in api/posts.js

listPosts(seatchText).then(posts => {
  ...
});
createPost(mood, text).then(post => {
  ... // post.id
});
createVote(id, mood).then(() => {...});
```

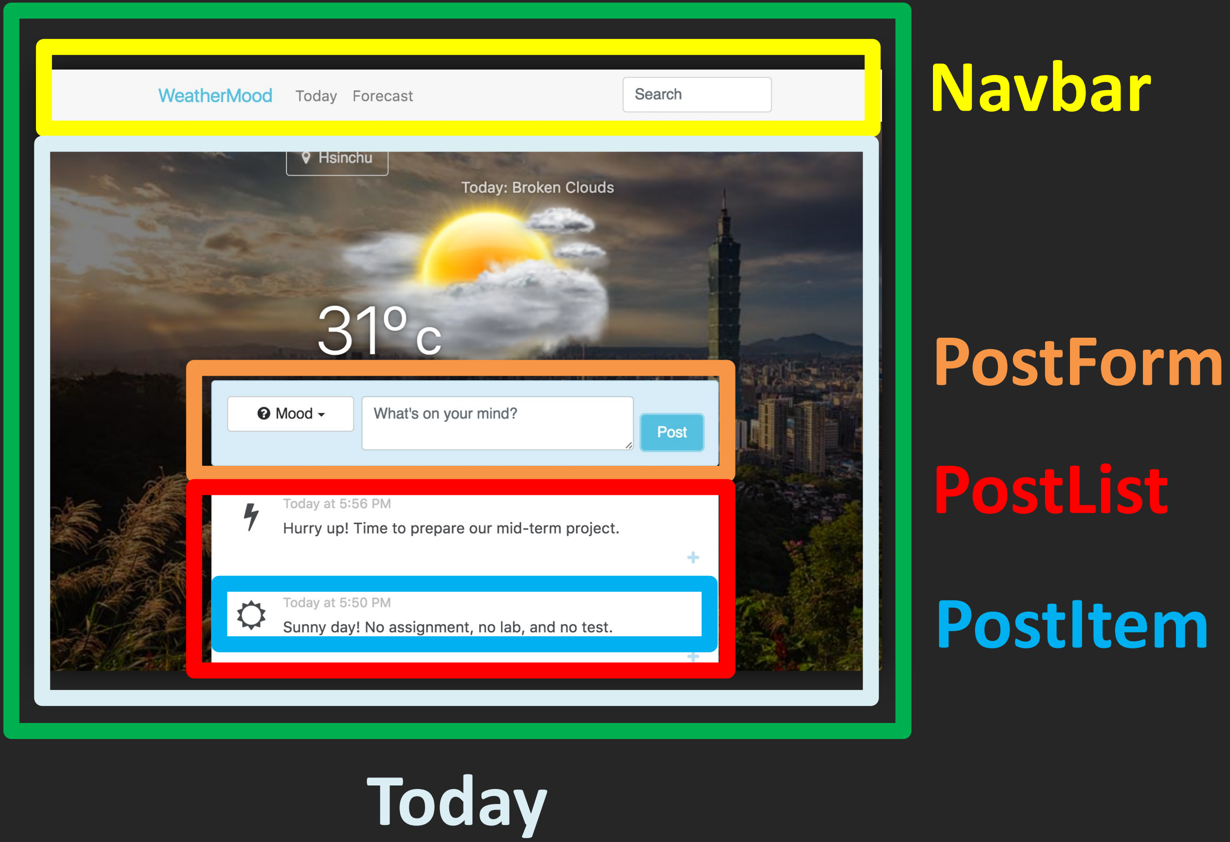
- Asynchronous (ES6 Promise-based)
- Simulated currently

HTML 5 Web Storage

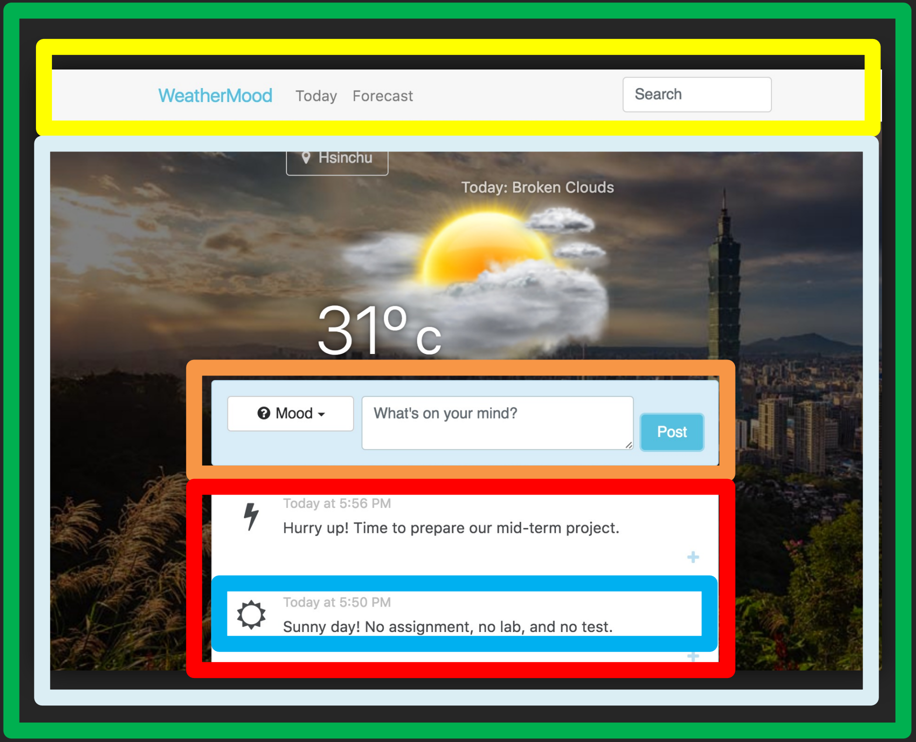
```
localStorage.setItem('key', 'value');
let v = localStorage.getItem('key');
localStorage.removeItem('key');
```

- Specific to domain ***and protocol***
- >5MB
- Values must be ***strings***
  - Use `JSON.stringify()` and `JSON.parse()` for objects
- `sessionStorage` is similar, except data gone when `window` closed

Steps 1 & 2: Components & Props



Steps 3 & 4: States



Today at 5:56 PM
Hurry up! Time to prepare our mid-term project.

Today at 5:50 PM
Sunny day! No assignment, no lab, and no test.

Main {
  searchText
}

Today { posts }

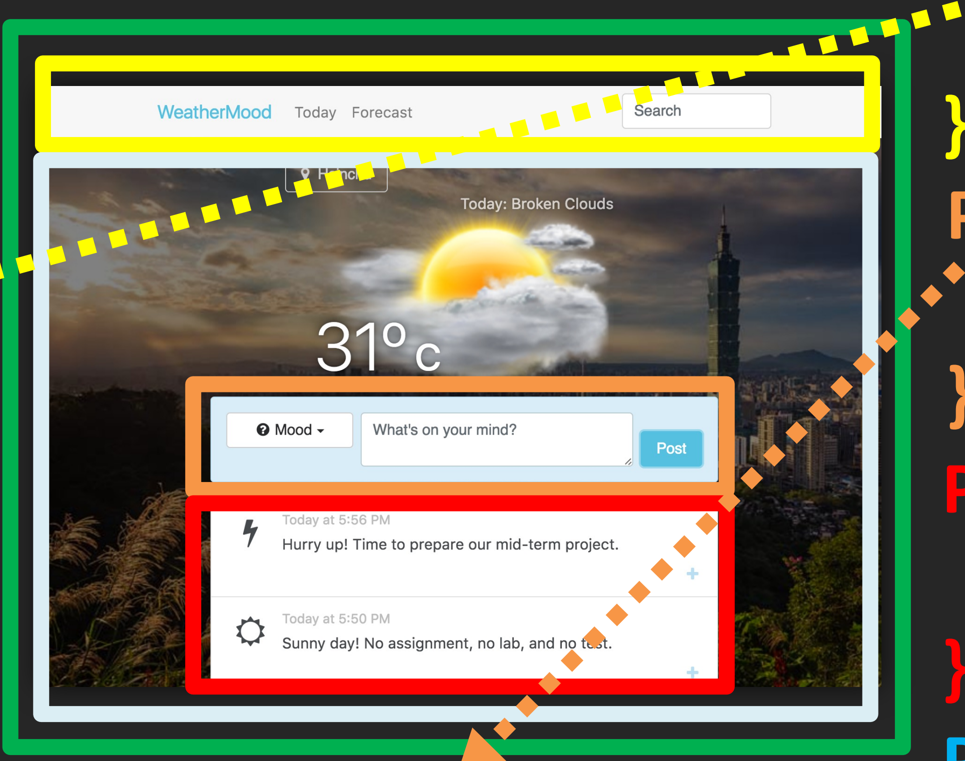
Navbar {
  ~~searchText~~
}

PostForm {
  ~~mood~~, ~~text~~
}

PostList {
  ~~posts~~
}

PostItem {
  ~~votes~~
}

Step 5: Callbacks



Today at 5:56 PM
Hurry up! Time to prepare our mid-term project.

Today at 5:50 PM
Sunny day! No assignment, no lab, and no test.

Navbar {
~~searchText~~
}

PostForm {
~~mood~~, ~~text~~
}

PostList {
~~posts~~
}

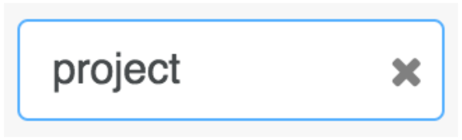
PostItem {
~~votes~~
}

Main {
searchText
}

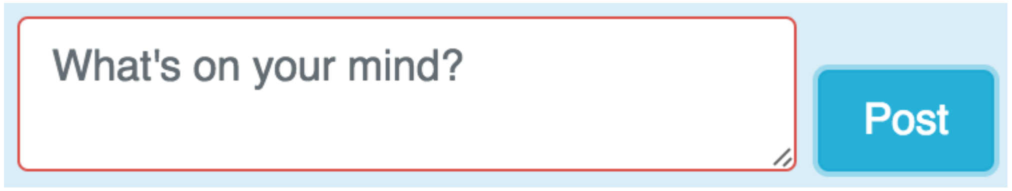
Today { posts }

Details

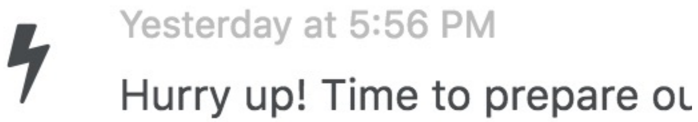
- Search box



- Form validation

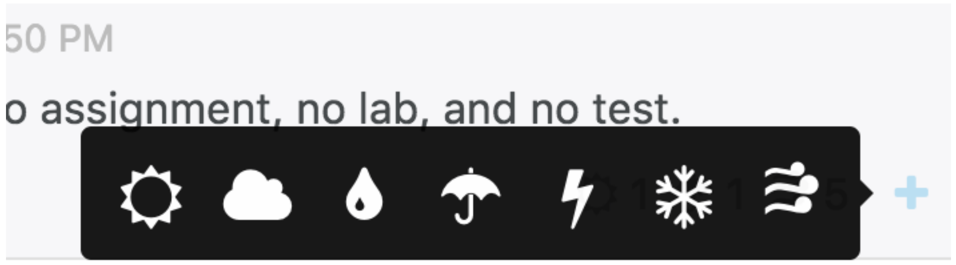


- Timestamp


Yesterday at 5:56 PM

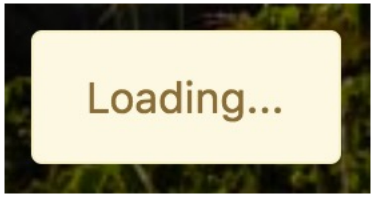
Hurry up! Time to prepare ou

- Tooltips


50 PM

o assignment, no lab, and no test.

- Loading indicators



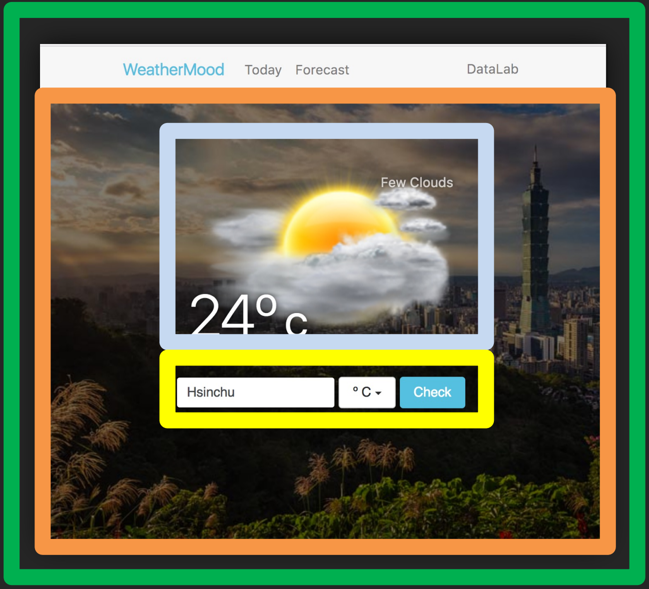
Outline

- WeatherMood: Posts
- Why Hook?
- State Hook
- Effect Hook
- Custom Hook
- Remarks

Limitation 1: Opaque States

- States of a component may be controlled outside
  - Changes are hard to track and debug
  - Hard to change component hierarchy

Main { unit }



WeatherDisplay {
~~temp~~, ~~unit~~
~~weather~~, ~~desc~~
}

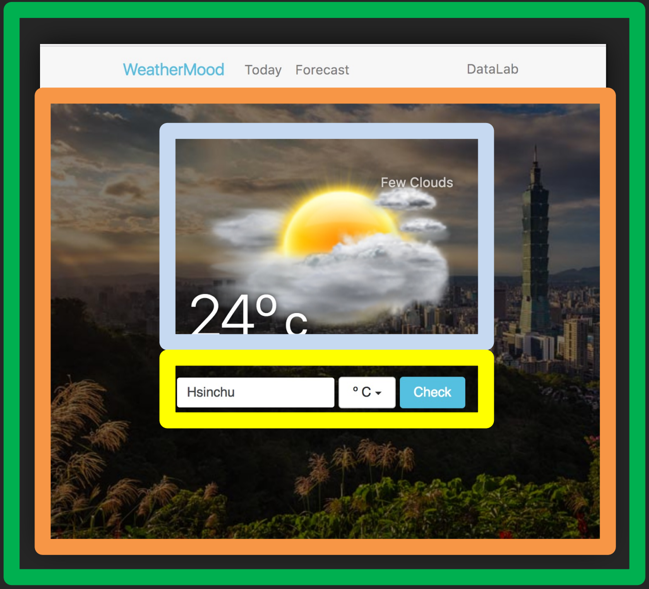
WeatherForm {
~~city~~, ~~unit~~
}

Today { weather, temp, desc, city }

Limitation 2: Mixture of Concerns

- Mixed concerns in lifecycle handlers
  - UI logic, state management, (async) side effects, etc.

Main { unit }



WeatherDisplay {
~~temp~~, ~~unit~~
~~weather~~, ~~desc~~
}

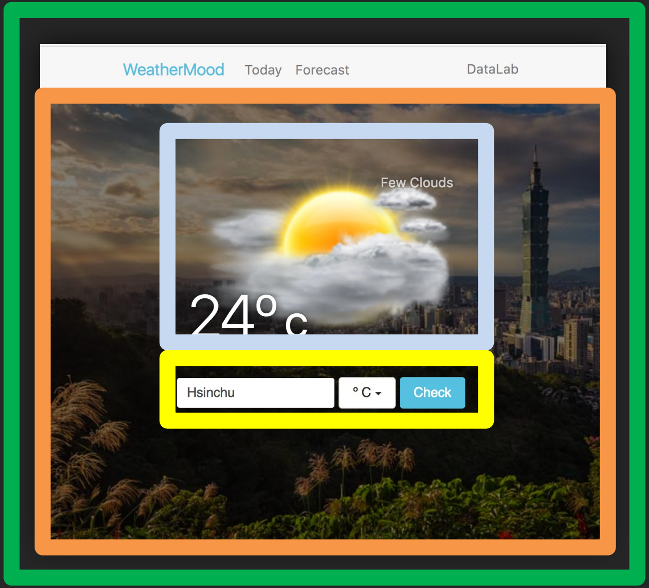
WeatherForm {
~~city~~, ~~unit~~
}

Today { weather, temp, desc, city }

Limitation 3: Non-reusable Stateful Logic

- Stateful logic cannot be shared and reused between components
  - E.g., “fetch data, render children if 200, else show error”

Main { unit }

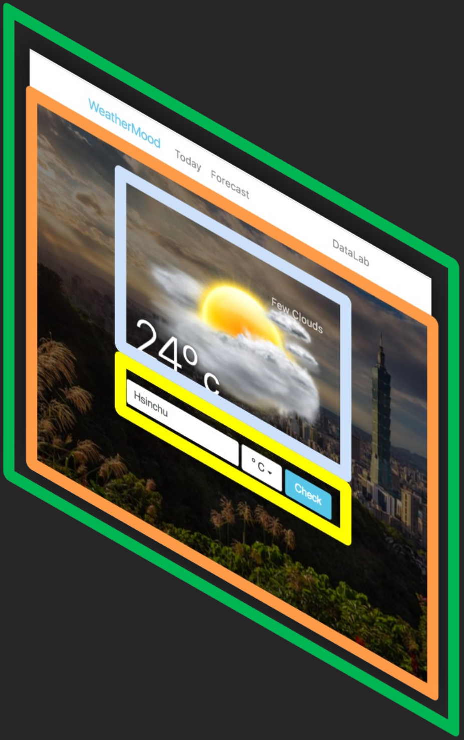


WeatherDisplay {
~~temp~~, ~~unit~~
~~weather~~, ~~desc~~
}

WeatherForm {
~~city~~, ~~unit~~
}

Today { weather, temp, desc, city }

React (UI) Hook (Stateful Logic)



1. use

2. stateful logic

```
Main { unit }
Today {
  weather, temp,
  desc, city
}
```

3. results
(state or side effect)

How?

Class vs. Function Components

- Class:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

- Function:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- Use:

```
const root = ReactDOM.createRoot(
                    document.getElementById('root'));
const element = <Welcome name="Sara" />;
root.render(element);
```

Why Function Components?

- Less code
- Faster and smaller; no ES6 transpilation
- No `this` and bindings

- No states
- No lifecycle (e.g., `componentDidMount()`)
- To be replaced by hooks

State Hook

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

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

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

- Name state and its setter function by convention

Multiple State Hooks

```
function ExampleWithManyStates() {
  // Declare multiple state variables!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{
                                  text: 'Learn Hooks’
                              }]);
  // ...
}
```

- One hook for each state

Effect Hook

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

function Example() {
  const [count, setCount] = useState(0);
  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

- Your ***effect function*** is called after React flushes changes to the DOM

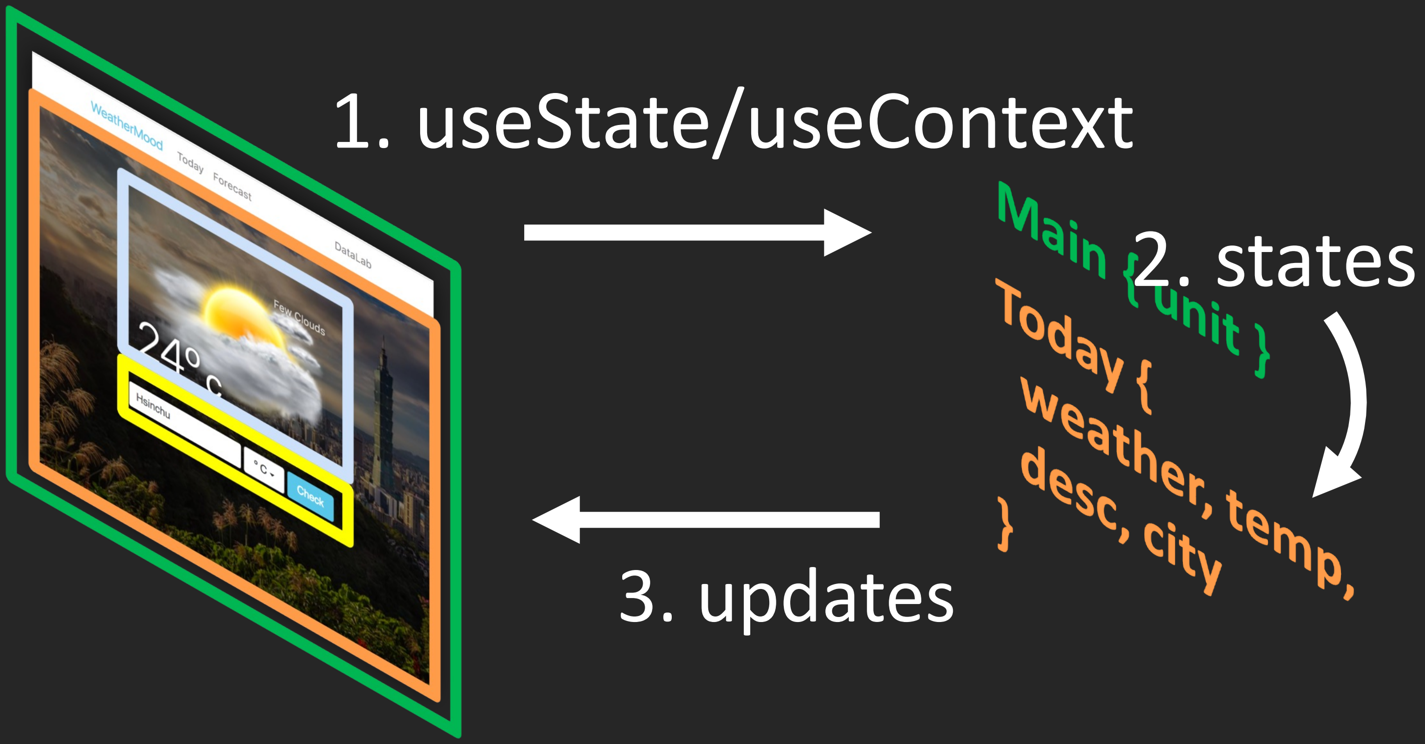
Effect Hook for Unmount Callback

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

function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  useEffect(() => {
    ChatAPI.subscribeToFriendStatus(props.friend.id,
                                    handleStatusChange);
    // similar to componentWillUnmount
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id,
                                    handleStatusChange);
    };
  });
  if (isOnline === null) return 'Loading...';
  return isOnline ? 'Online' : 'Offline';
}
```

Advantage 1: Clearer States

- States are always "behind" components
  - No state management outside (by ancestors)
  - Easy to change component hierarchy



Advantage 2: Separation of Concerns

```
function FriendStatusWithCounter(props) {
  const [count, setCount] = useState(0);
  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  const [isOnline, setIsOnline] = useState(null);
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  useEffect(() => {
    ChatAPI.subscribeToFriendStatus(props.friend.id,
                                    handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id,
                                          handleStatusChange);
    };
  });
  // ...
```

Advantage 3: Reusable Stateful Logic

- Define a custom hook:

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

function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  useEffect(() => {
    ChatAPI.subscribeToFriendStatus(friendID,
                                    handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID,
                                          handleStatusChange);
    };
  });
  return isOnline;
}
```

Advantage 3: Reusable Stateful Logic

- Use a custom hook:

```
function FriendStatus(props) {
  const isOnline = useFriendStatus(props.friends.id1);

  if (isOnline === null) return 'Loading...';
  return isOnline ? 'Online' : 'Offline';
}

function FriendListItem(props) {
  const isOnline = useFriendStatus(props.friends.id2);

  return (
    <li style={{ color: isOnline ? 'green' : 'black' }}>
      {props.friend.name}
    </li>
  );
}
```

- The two `isOnline` are independent and can have different values

Outline

- WeatherMood: Posts
- Why Hook?
- State Hook
- Effect Hook
- Custom Hook
- Remarks

Function Component with State

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

function Example() {
  // Declare a new state variable, which we 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 with State

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.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>
    );
  }
}
```

Why not `creatState`?

- In a function component, `useState` is called ***whenever the component is re-rendered***
- New state is created only during the first render

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

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

  return (
    <div> … </div>
  );
}
```

Array Destructuring

```
const [count, setCount] = useState(0);

// equals to

var stateVariable = useState(0);
var count = stateVariable[0];
var setCount = stateVariable[1];
```

- Name by convention

Outline

- WeatherMood: Posts
- Why Hook?
- State Hook
- Effect Hook
- Custom Hook
- Remarks

Function Component with Side Effect

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

function Example() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

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

Class Component with Side Effect

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }
  componentDidMount() {
    document.title = `You clicked ${this.state.count} times`;
  }
  componentDidUpdate() {
    document.title = `You clicked ${this.state.count} times`;
  }
  render() {
    return (
      <div> … </div>
    );
  }
}
```

Effect Function: Access

```
useEffect(() => {
  document.title = `You clicked ${count} times`;
});
```

- Your effect/clean-up function can access props and states directly
  - through JavaScript ***closures***
  - instead of React-specific APIs

Effect Function: Calling Time

```
function Example() {
  const [count, setCount] = useState(0);
  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}> … </button>
    </div>
  );
}
```

- Your effect/clean-up function is run ***whenever the component mounts/updates/unmounts***
  - To update `document.title` in case `count` changes

Performance Optimization

- What if effect function should only run when a condition is met?
  - E.g., change of `props.friend.id`

```
useEffect(() => {
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  ChatAPI.subscribeToFriendStatus(props.friend.id,
                                  handleStatusChange);
  // similar to componentWillUnmount()
  return () => {
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id,
                                  handleStatusChange);
  };
});
// Re-subscribe after every re-render
```

Performance Optimization

- What if effect function should only run when a condition is met?
  - E.g., change of `props.friend.id`

```
useEffect(() => {
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  ChatAPI.subscribeToFriendStatus(props.friend.id,
                                  handleStatusChange);
  // similar to componentWillUnmount()
  return () => {
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id,
                                  handleStatusChange);
  };
}, [props.friend.id]);
// Re-subscribe after every re-render
```

Performance Optimization

- Make sure the array
  - includes all values from the component scope (such as props and state) that change over time and are used by the effect
  - include nothing `[]` if you want to run an effect clean-up function only once on mount/unmount

```
useEffect(() => {
  … // effect function
}, []);
// Run effect function just once
```

Outline

- WeatherMood: Posts
- Why Hook?
- State Hook
- Effect Hook
- Custom Hook
- Remarks

Custom Hook

- Define:

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

function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);
  function handleStatusChange(status) {
    setIsOnline(status.isOnline);
  }
  useEffect(() => {
    ChatAPI.subscribeToFriendStatus(friendID,
                                    handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID,
                                          handleStatusChange);
    };
  }, [friendID]);
  return isOnline;
}
```

- ***Clean-up function*** also runs whenever component updates (before effect function)

Interacting with Other Hooks

```
const friendList = [{ id: 1, name: 'Phoebe’ }, … ];
function ChatRecipientPicker() {
  const [recipientID, setRecipientID] = useState(1);
  const isRecipientOnline = useFriendStatus(recipientID);
  return (
    <div>
      <Circle color={isRecipientOnline ? 'green' : 'red'} />
      <select value={recipientID}
        onChange={e => setRecipientID(Number(e.target.value))}
      >
        {friendList.map(friend => (
          <option key={friend.id} value={friend.id}>
            {friend.name}
          </option>
        ))}
      </select>
    </div>
  );
}
```

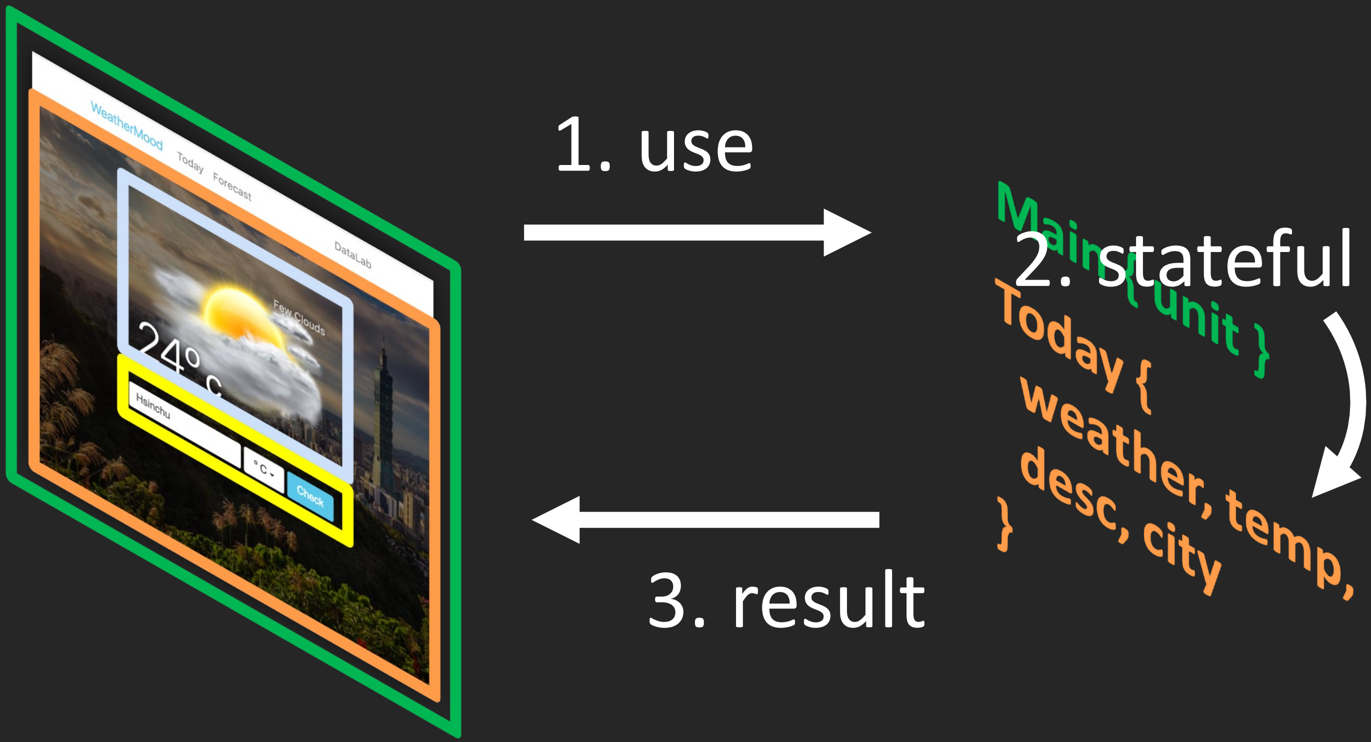
- If `recipientID` changes, `useFriendStatus` will unsubscribe from the previously selected friend and then subscribe to the new one

Outline

- WeatherMood: Posts
- Why Hook?
- State Hook
- Effect Hook
- Custom Hook
- Remarks

Advantages of Hooks

- Clearer state management; more composable UI hierarchy
- Separation of concerns
- Reusable stateful logic



Rules of Thumb when Using Hooks

- Only call Hooks at the top level in function components
  - Don’t call Hooks inside loops, conditions, or nested functions
- Only call Hooks from React function components or custom hooks
  - Don’t call Hooks from regular JavaScript functions
- Use linter plugin to enforce these rules

Assigned Reading

- React hook tutorial
  - For people already familiar with React Class Components
- API doc
  - useContext
    - Share states between multiple components
  - useReducer
    - Useful when your state has multiple sub-values that need to be updated together in a consistent manner
- Common custom hooks
  - E.g., data fetch, stateful mutations, etc.