

State Management and Redux

Shan-Hung Wu & DataLab
CS, NTHU

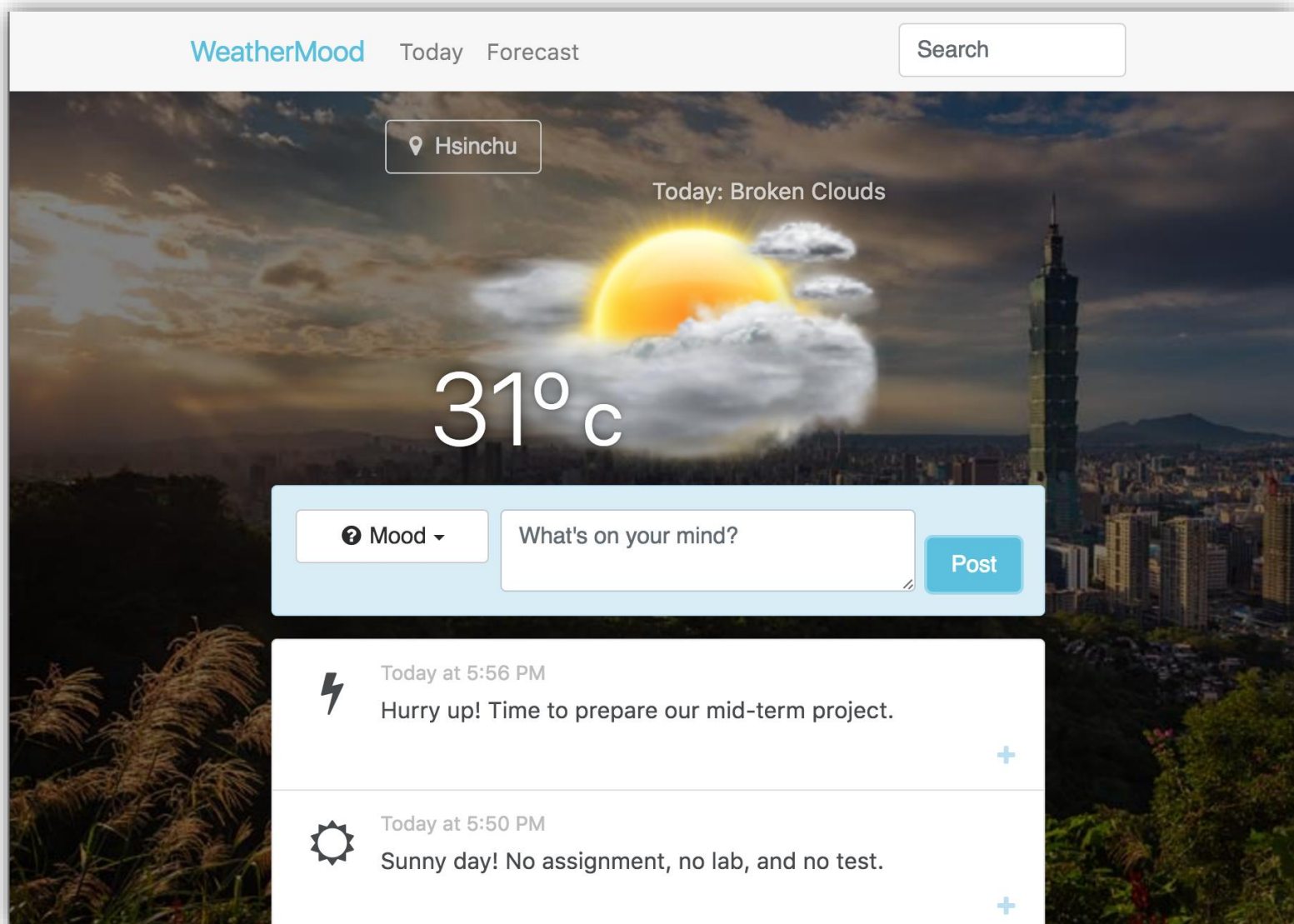
Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- Async Actions and Middleware
- Connecting with React Components
- Remarks

Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- Async Actions and Middleware
- Connecting with React Components
- Remarks

Clone weathermood/react-post



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

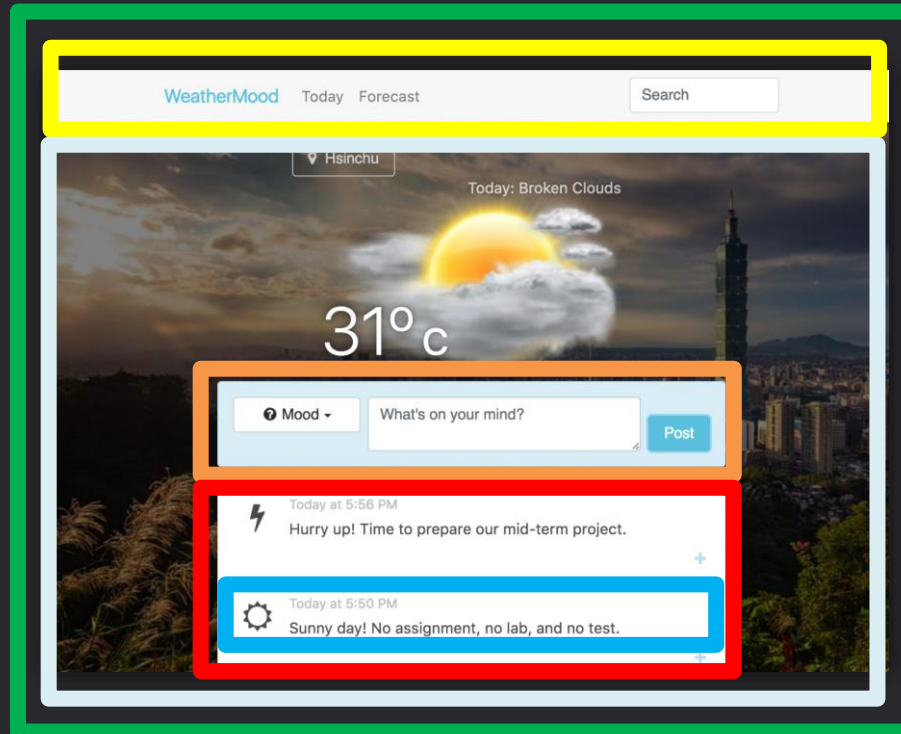
HTML 5 Web Storage

```
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

Main



Navbar

PostForm

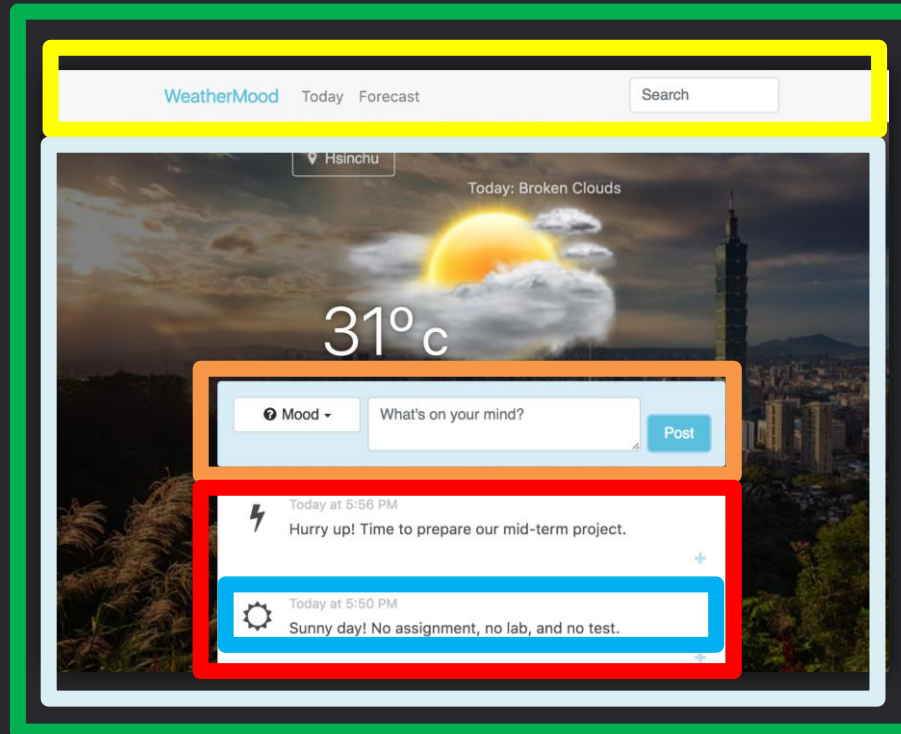
PostList

PostItem

Today

Steps 3 & 4: States

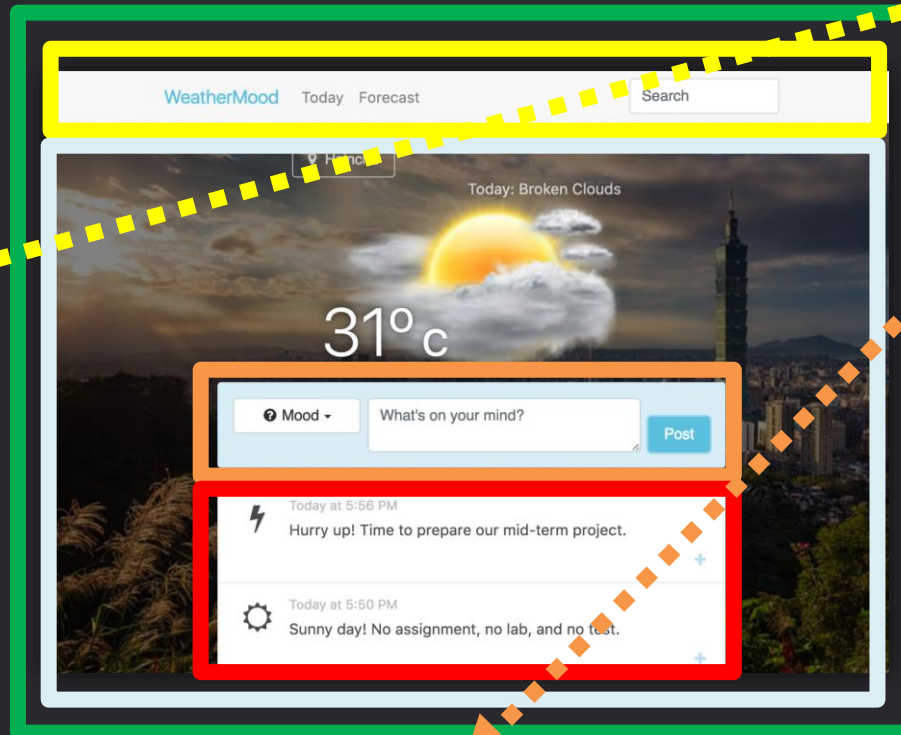
Main {
 searchText
}



Navbar {
 searchText
}
PostForm {
 mood, text
}
PostList {
 posts
}
PostItem {
 votes
}

Step 5: Callbacks

Main {
searchText
}



Navbar {
searchText
}

PostForm {
mood, text
}

PostList {
posts
}

PostItem {
votes
}

Today { posts }

Details

- Search box

- Form validation

- Timestamp



Yesterday at 5:56 PM

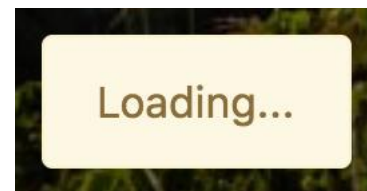
Hurry up! Time to prepare ou

- Tooltips

50 PM

o assignment, no lab, and no test.

- Loading indicators



Outline

- WeatherMood: Posts
- **Why Redux?**
- Actions and Reducers
- Async Actions and Middleware
- Connecting with React Components
- Remarks

React is Declarative in Terms of *States*

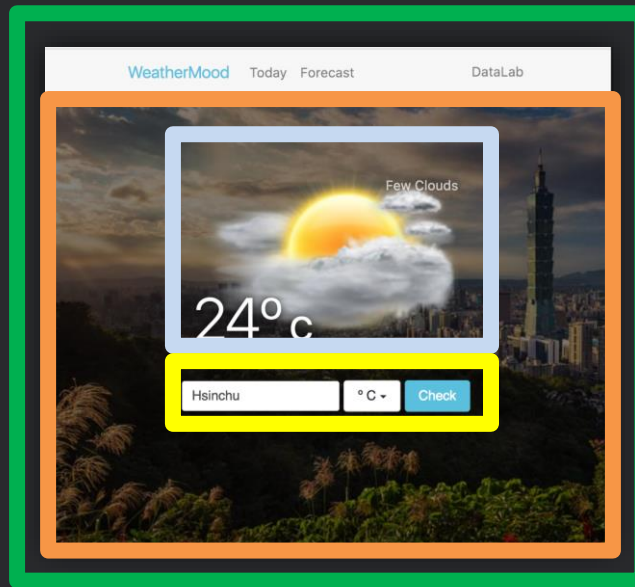
```
render() {  
  return (  
    <h1 className={this.state.toggle}>  
      Hello {this.props.name}  
    </h1>  
  );  
}
```

- Code for “states,” not “changes of states”
 - Virtual DOM tracks changes automatically
- UI = *maps from states* to visual looks
 - Each component is a function of partial states

Limitations I

- States of a component may be controlled outside
 - `Main` and `Today` may be complex and diverse

Main { unit }



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

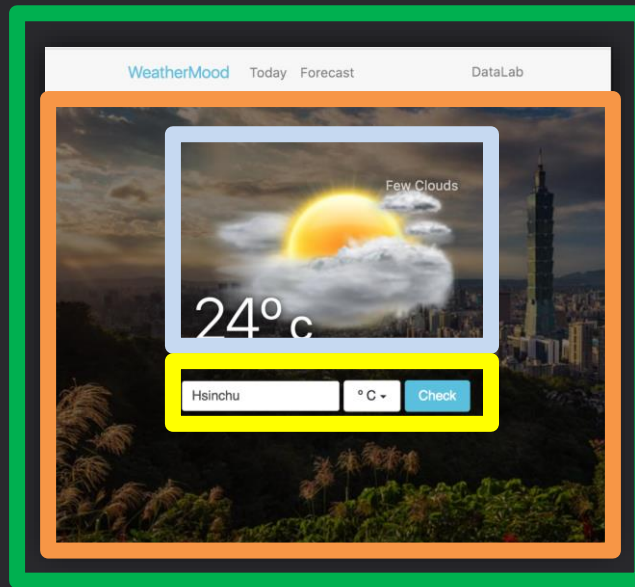
```
WeatherForm {  
  city, unit  
}
```

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

Limitations II

- Cannot move components easily
 - Bad for evolving projects (e.g., startups)

Main { unit }



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

```
WeatherForm {  
  city, unit  
}
```

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

Limitations III

- States are hard to track
 - Spread among multiple components
- Mixture of concerns
 - Code that maintain states
 - Rendering logics
- State changes are implicit
 - Where did such a state come from?

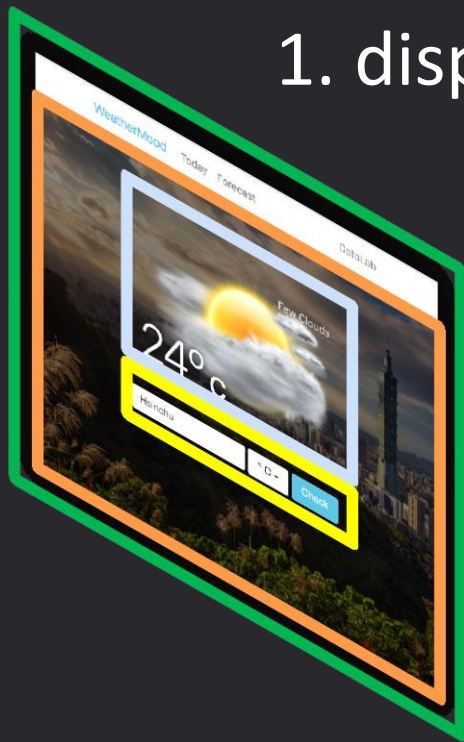


Redux

- A state management framework
 - Restricts how you write state management code
- Not tied to, but works well with React

React (UI)

Redux (State Store)



1. dispatch(action)



2. reduce(action)

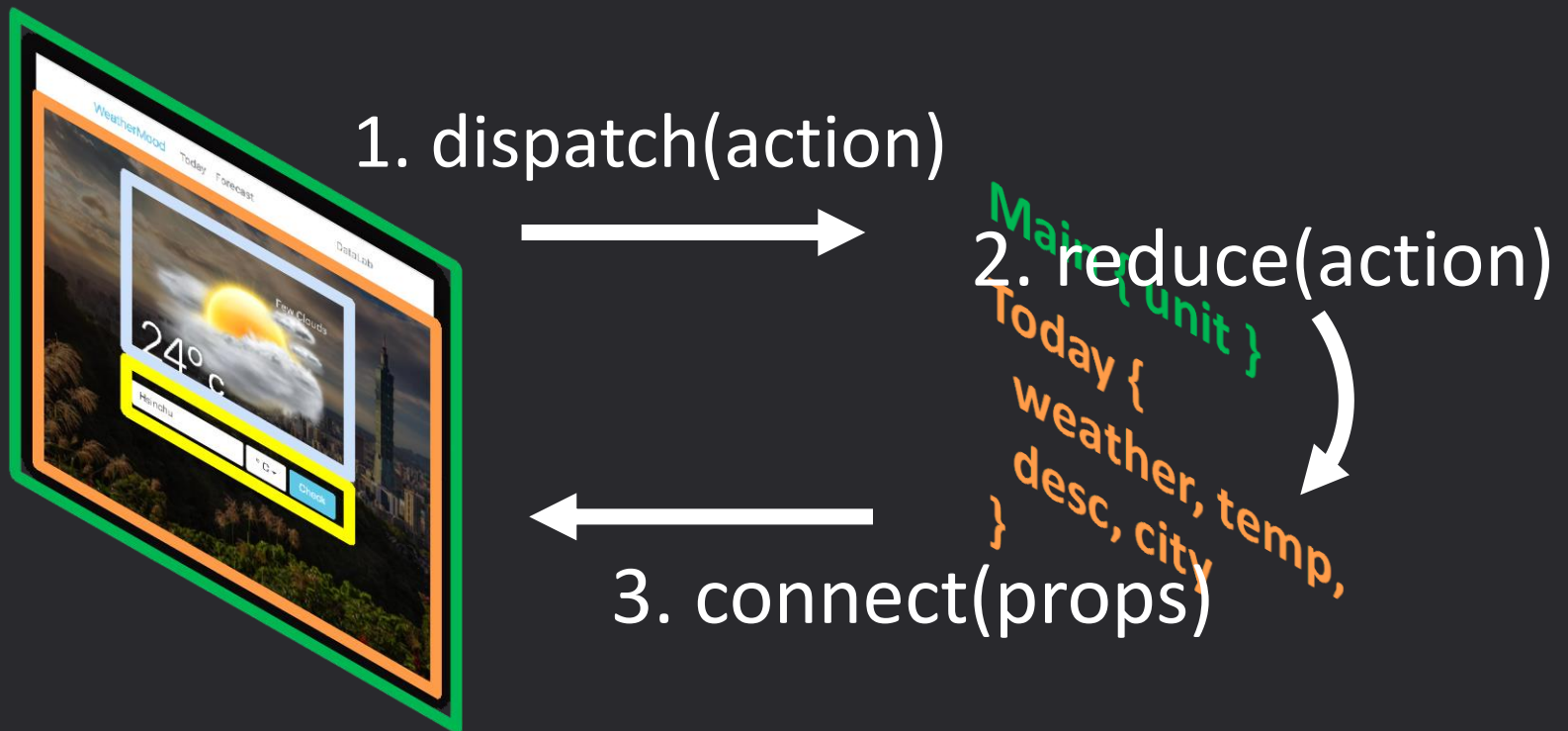
Main { unit }
Today {
weather, temp,
desc, city
}



3. connect(props)

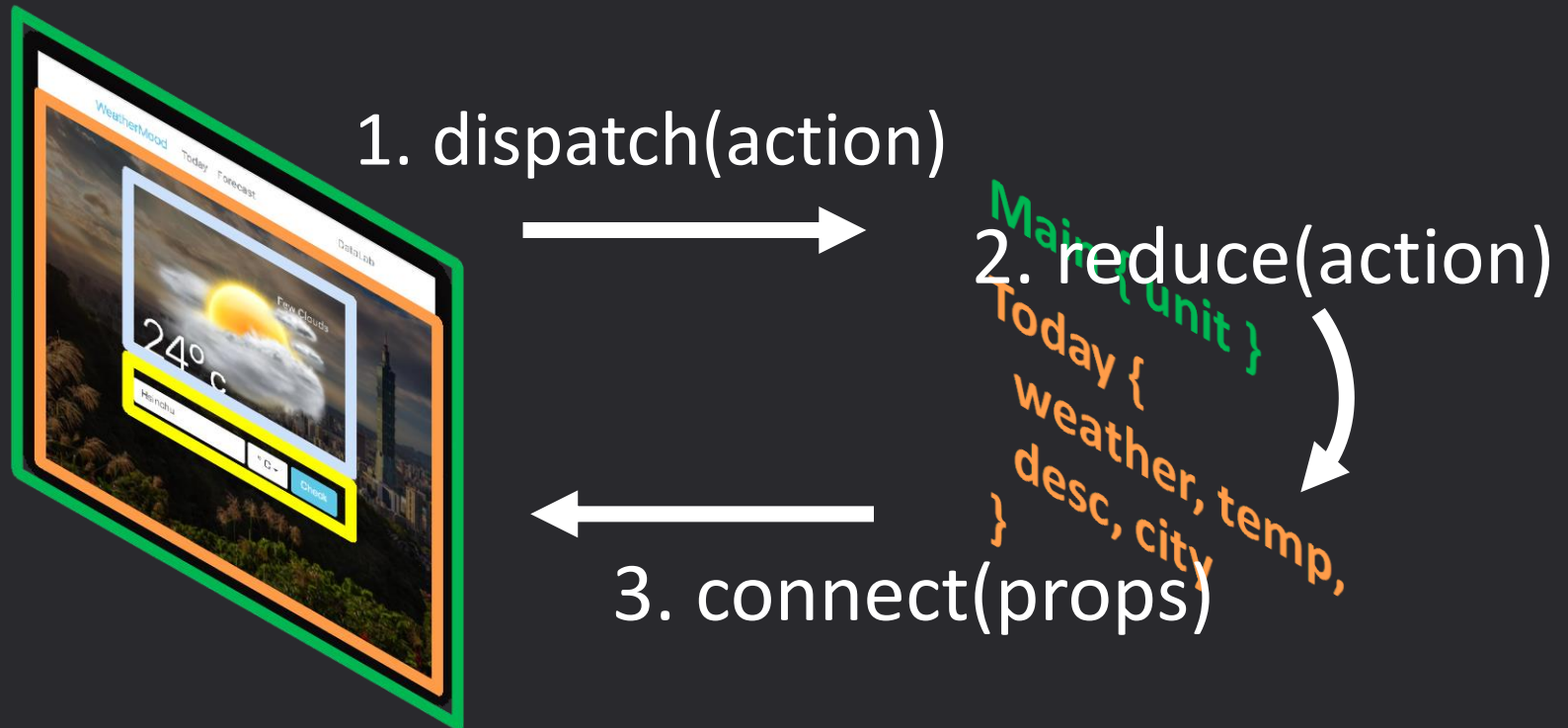
Advantages I

- Separation of concerns
 - Rendering logic vs. state management



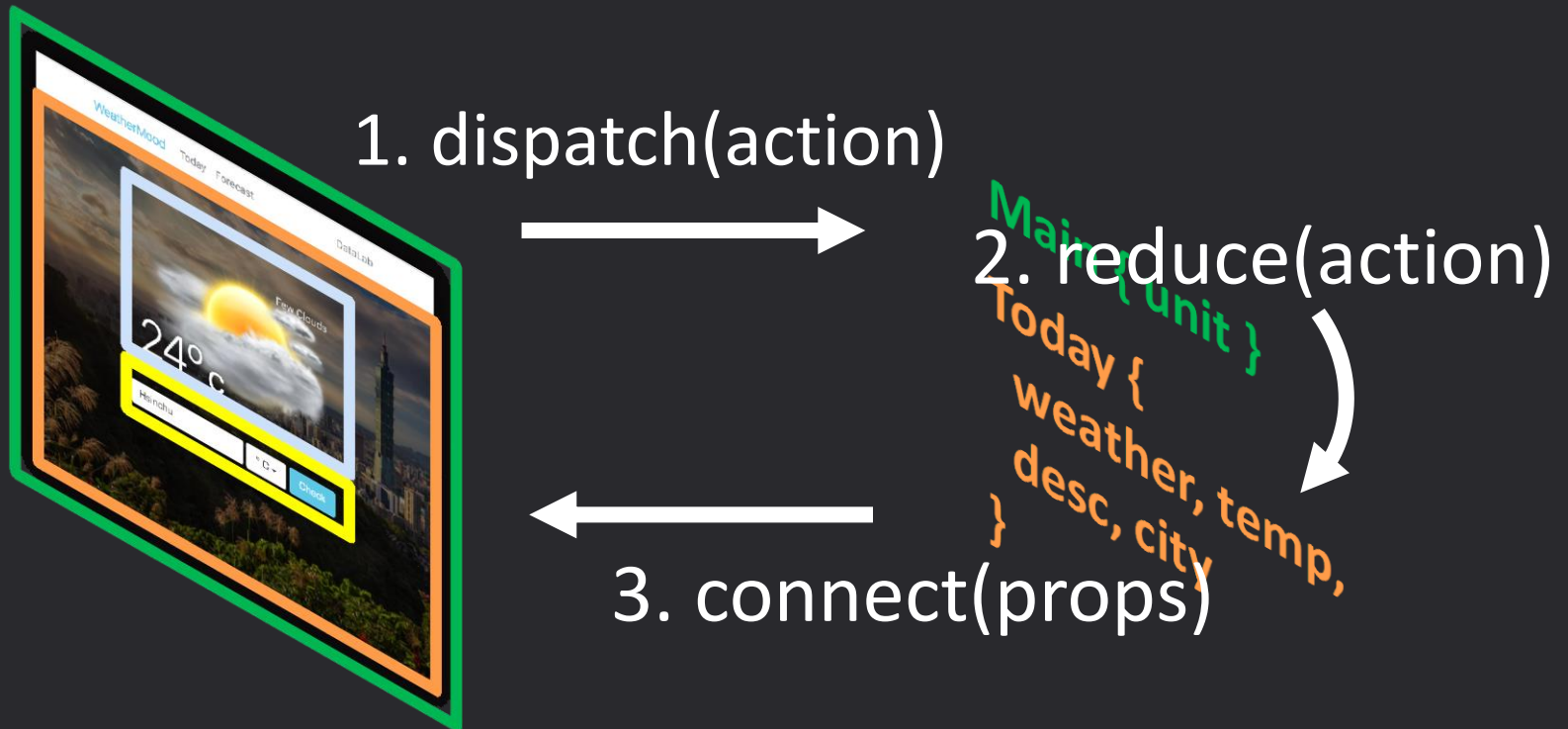
Advantages II

- Unidirectional (top-down) data flow in React
 - Loosely coupled components; UI easy to change



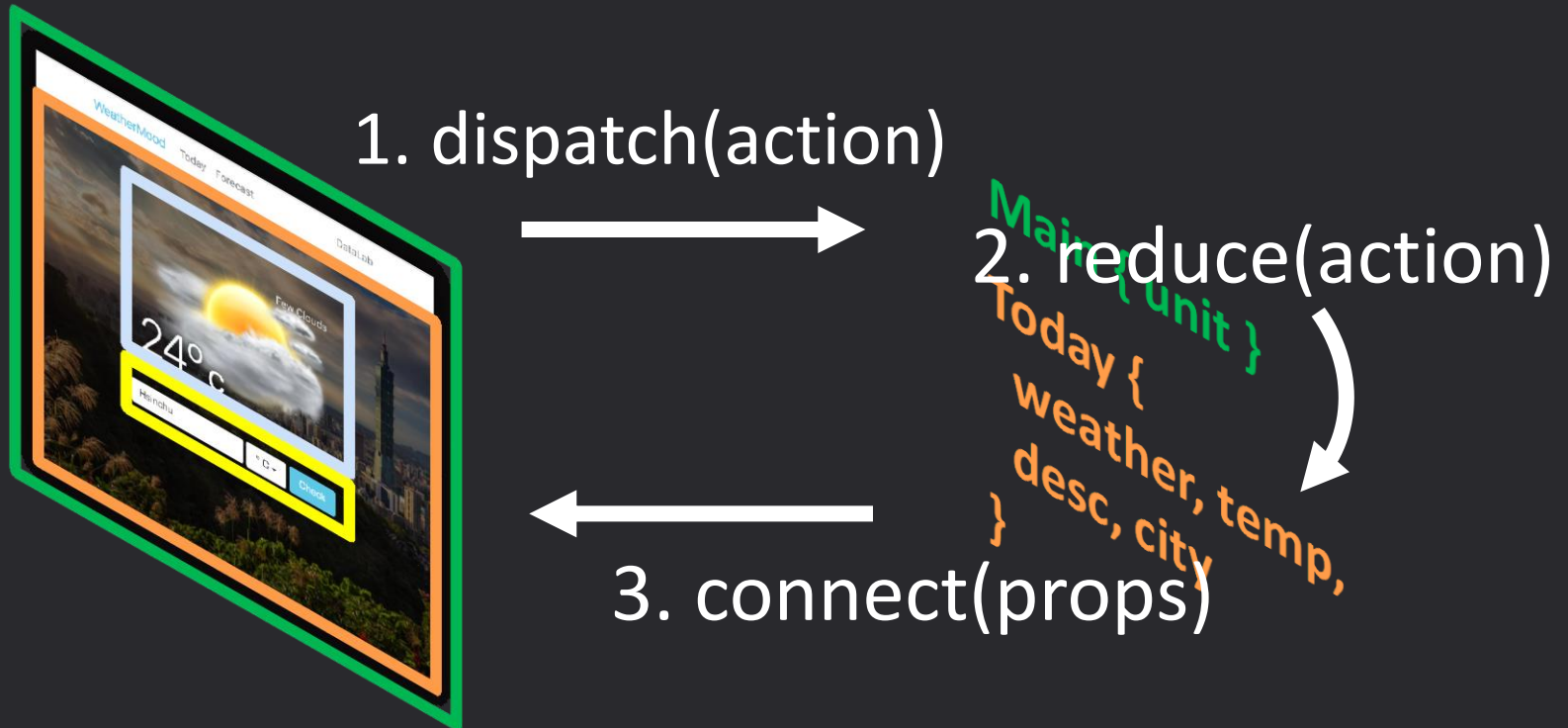
Advantages III

- Single source of the “truth”
 - States easy to inspect



Advantages IV

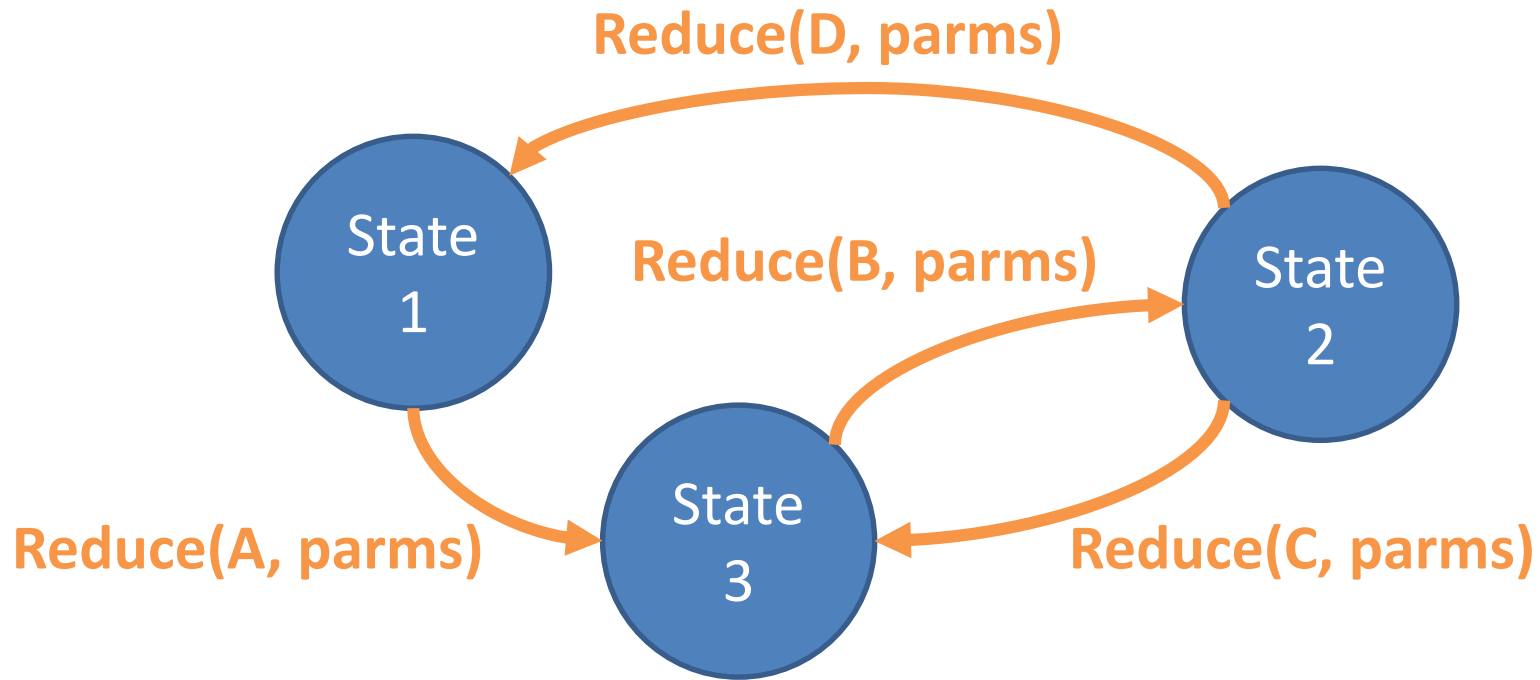
- Explicit actions
 - State changes revertable; easy to debug



Outline

- WeatherMood: Posts
- Why Redux?
- **Actions and Reducers**
- Async Actions and Middleware
- Connecting with React Components
- Remarks

Redux Store Is a State Machine



- State transitions must be ***deterministic***
- I.e., same (prev state, action, parms), same next state

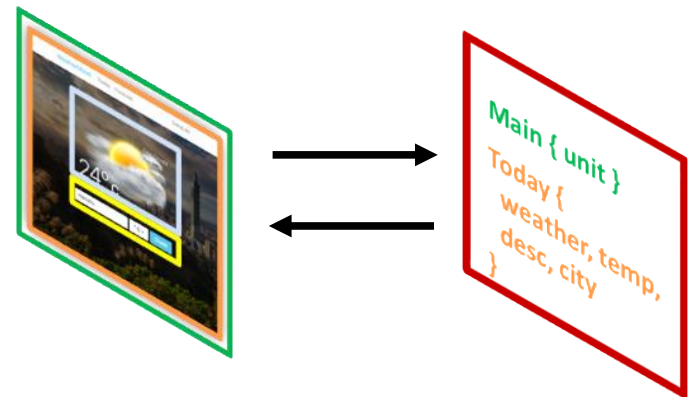

```

// action generator
export function setWeather(code, temp) {
  return { // action and parms
    type: '@WEATHER/SET_WEATHER',
    code,
    temp
  };
}

// reducer
export function weather(state = {...}, action) {
  switch (action.type) {
    case '@WEATHER/SET_WEATHER':
      return {
        ...state,
        code: action.code,
        temp: action.temp
      };
    default:
      return state;
  }
}

```

Actions & Reducers



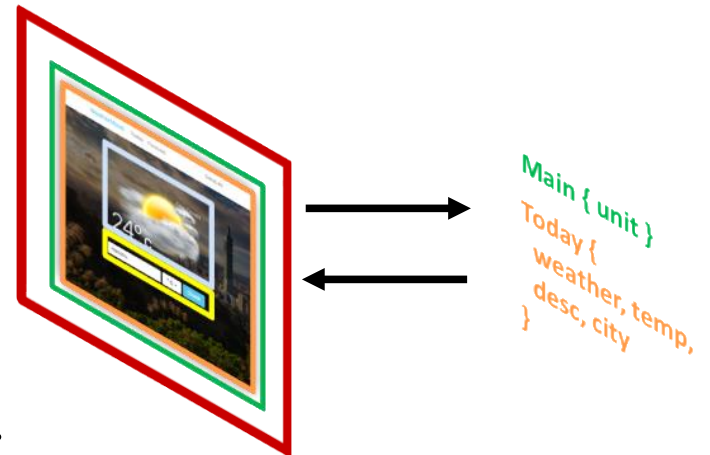
Using Redux Store

```
// in UI
import {createStore} from 'redux';
import {setWeather, weather} from ...;

const store = createStore(weather);
```

```
// in Component1
store.subscribe(() => {
  console.log(store.getState());
});
```

```
// in Component2
store.dispatch(setWeather(800, 21));
```



Reducers Must Be *Pure* Functions

- To ensure deterministic state transitions
- Pure functions?
- Same input, same output
 - No `Math.random()` nor `Date.now()`
- No side effect
 - Cannot update variables outside
 - Cannot mutate input
 - Cannot make API calls
- Synchronous

Splitting Reducers

```
export function code(state = -1, action) {
  switch (action.type) {
    case '@CODE/SET_CODE':
      return action.code;
    default:
      return state;
  }
}

export function temp(state = 0, action) {
  switch (action.type) {
    case '@TEMP/SET_TEMP':
      return action.temp;
    default:
      return state;
  }
}
```

- One reducer for independent “state group”

```
const store = createStore((state, action) => ({ // wrapper
  code: code(state.code, action),
  temp: temp(state.temp, action)
})));
```

Simplification

```
const store = createStore((state, action) => ({  
  code: code(state.code, action),  
  temp: temp(state.temp, action)  
}));
```

// same as

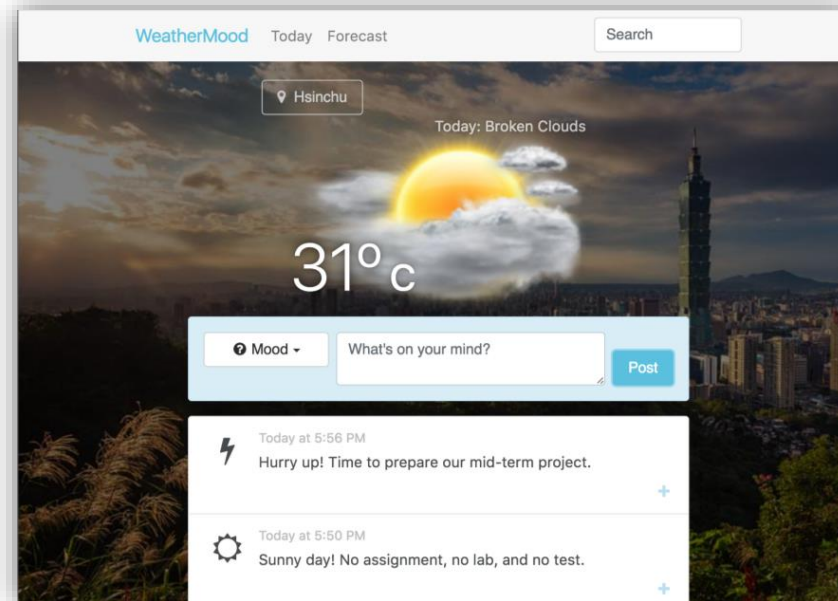
```
import {combineReducers} from 'redux';
```

```
const store = createStore(combineReducers({  
  code,  
  temp  
}));
```

Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- **Async Actions and Middleware**
- Connecting with React Components
- Remarks

weathermood/redux-weather



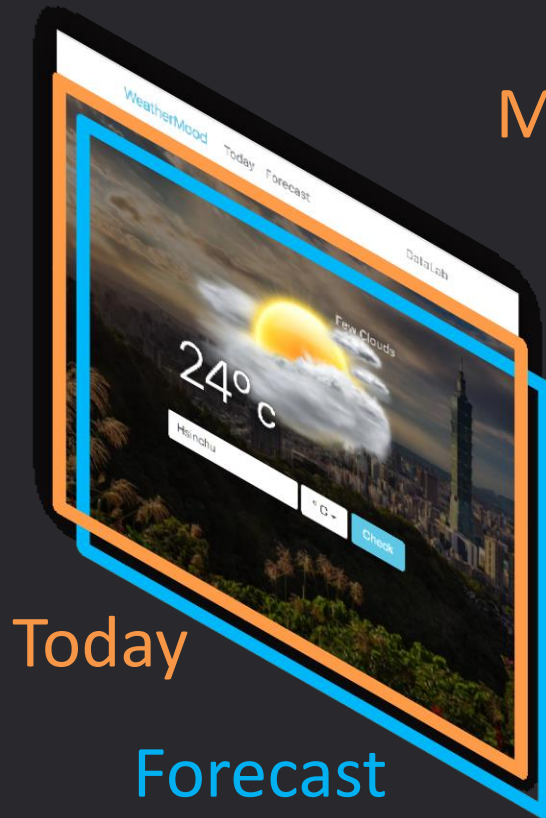
- Looks the same as react-post
- But weather components (Today, Forecast, etc.) use Redux to manage states

How to Design Reducers?

1. Identify independent “state groups”
 - E.g., weather+forecast vs. posts

How to Design Reducers?

2. Come out lifted state hierarchy as in react
3. Move states of each component to a reducer



Move with
Today

Move with
Forecast

```
@weather {  
  city, code, temp, desc  
}  
@unit { unit }  
@forecast { city, list }
```

Async Actions

- For fetching weather, forecast, posts, etc.
- But reducers must be pure
 - No API call, synchronous
- How?
 1. Break async action into *sequence of steps*
 - State transition for each step is deterministic
 2. Dispatch steps in UI following the sequence

```
// action generators
export function startGetWeather() {
  return {type: '@WEATHER/START_GET_WEATHER'};
}
export function endGetWeather(code, temp) {
  return {
    type: '@WEATHER/END_GET_WEATHER',
    code,
    temp
  };
}
// reducers (pure)
...

// in UI
store.dispatch(startGetWeather());
const {code, temp} = ... // AJAX callback
store.dispatch(endGetWeather(code, temp));
```

Problems?

State management in UI again

```
$ npm install --save redux-thunk
```

```
// high-order action generator
export function getWeather() {
  return (dispatch, state) => {
    dispatch(startGetWeather());
    const {code, temp} = ... // AJAX callback
    dispatch(endGetWeather(code, temp));
  };
}
```

Dispatching Action Sequences

```
// in UI
import {compose, applyMiddleware} from 'redux';
import thunkMiddleware from 'redux-thunk';

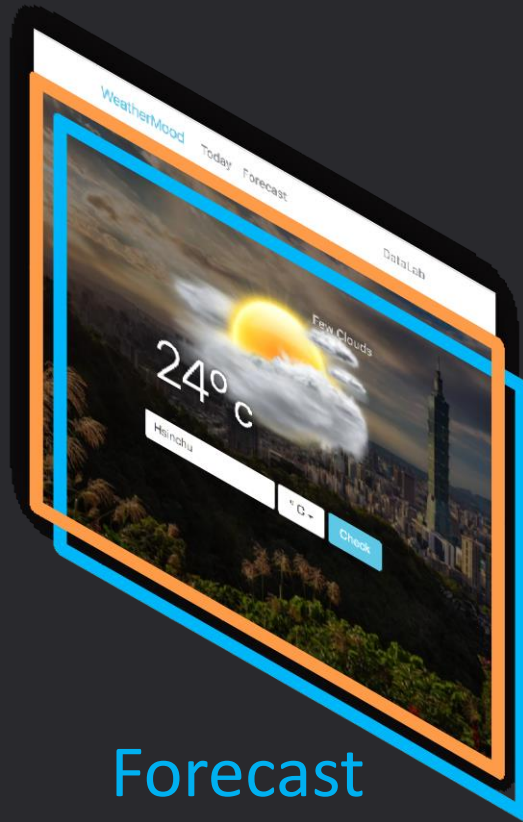
const store = createStore(combineReducers({
  ...
}), compose(applyMiddleware(thunkMiddleware)));

store.dispatch(getWeather());
```

Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- Async Actions and Middleware
- **Connecting with React Components**
- Remarks

Today



Forecast

How?

```
@weather {  
  city, code, temp, desc  
}  
@unit { unit }  
@forecast { city, list }
```

Tedious Way

1. Create `store` in `Main`, then pass it down to all descendants
 - Lots of repeating props in JSX
2. In each component, call `store.subscribe()` and `dispatch()`
 - No `this.state` and `setState()`
 - Instead, use `this.forceUpdate()` and determine when to re-render


```
$ npm install --save react-redux
```

React-Redux

```
// in Main.jsx
import {Provider} from 'react-redux';
render() {
  return (
    <Provider store={...}>...</Provider>
  );
}
```

```
// in Today.jsx
import {connect} from 'react-redux';
class Today extends React.Component {
  ... // has this.props.dispatch
}
export default connect(state => ({ // state to props
  ..state.weather,
  unit: state.unit
})) (Today);
```

- Only props in components

Outline

- WeatherMood: Posts
- Why Redux?
- Actions and Reducers
- Async Actions and Middleware
- Connecting with React Components
- **Remarks**

Remarks I

- Separation of concerns
- Components can be moved easily



Remarks II

- States easy to inspect
- Explicit actions + deterministic state transition = time travel

The image shows the Redux DevTools Chrome extension page. At the top, it says "Redux DevTools" offered by remotedev.io, with a 5-star rating from 226 reviews and 178,159 users. Below this are tabs for OVERVIEW, REVIEWS, SUPPORT, and RELATED. The main content area displays a preview of the Redux DevTools interface, which includes a state inspector, a log monitor, and a timeline. The state inspector shows a state object with a 'todos' array. The log monitor shows a sequence of actions, including '@@@INIT', 'COMPLETE_TODO', and 'ADD_TODO'. The timeline shows a sequence of events, including '@@@INIT', 'COMPLETE_TODO', and 'ADD_TODO'. On the right side of the page, there is a section titled "Compatible with your device" and a description of the extension: "Redux DevTools for debugging application's state changes. The extension provides power-ups for your Redux development workflow. Apart from Redux, it can be used with any other architectures which handle the state. It's an opensource project. See the official repository for more details: <https://github.com/zalmoxisus/redux-devtools-extension>". At the bottom right, there are links for "Website" and "Report Abuse", and a section for "Additional Information" with the version "2.14.3".

Redux DevTools
offered by remotedev.io
★★★★★ (226) | [Developer Tools](#) | 178,159 users

OVERVIEW REVIEWS SUPPORT RELATED

Compatible with your device

Redux DevTools for debugging application's state changes.

The extension provides power-ups for your Redux development workflow. Apart from Redux, it can be used with any other architectures which handle the state.

It's an opensource project. See the official repository for more details:
<https://github.com/zalmoxisus/redux-devtools-extension>

[Website](#)
[Report Abuse](#)

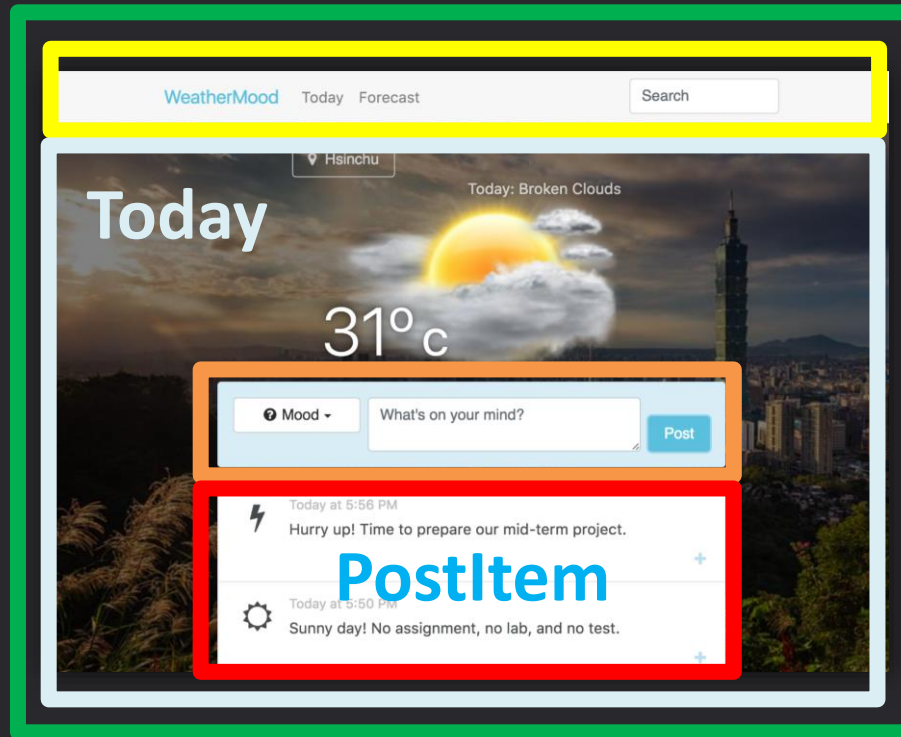
Additional Information
Version: 2.14.3

Readings

- [Advanced Redux walkthrough](#) (optional)
 - Async actions & flow
 - Middlewares
 - Usage with React Router
 - More examples

Assignment: Post Components + Redux

Main



Navbar

PostForm

PostList

Requirements

- Specify reducers and action types (with @'s) in README
- Setup store to allow time travel using Redux DevTools