

# REACT SO FAR

- React function-based components
- state-per-component from `useState` hook
- passing state as props
- altering state in children via callback props
- per-render/init effects from `useEffect` hooks
- changing css classes via state/props for non-structural visual changes

# COMPLEX STATE

`useState` is normally fine

- What if you have multiple state flags that could change at the same time?
- What if your new state is based on the previous state

Answer: `useReducer` hook

# WHAT IS A REDUCER?

- A **state**
  - usually an object
- A **reducer** function
  - a **pure function** (output only based on input)
  - takes the previous state
  - takes an "action" ( a name + data )
  - returns a **new state**
- A **dispatcher** function
  - called with the action name + data
  - calls the reducer
  - replaces the previous state with the new state

# EXAMPLE: APPROPRIATE TIME FOR REDUCER

Imagine you have a **user profile** with these values:

- username
- user actual name
- avatar image url
- theme ('dark' or 'light')
- last active time

You could track each of these with a `useState` value and update them as needed

...But that can be tedious

# EXAMPLE: REDUCER STATE

```
{  
  username: 'bao',  
  actualName: 'Wu Bao',  
  avatar: 'https://examplecat.com/cat.png',  
  theme: 'light',  
  lastActive: 1585797861760, // Date.now() - ms since Epoch  
}
```

# WHAT ACTIONS DO WE HAVE?

This COULD be changing each field individually

- But some actions might be grouped

Example:

- Changing theme
- Changing username/avatar
- Changing actual name/birthday
- Updating lastActive
  - Plus, all the OTHER actions should update this too

# ACTIONS AS CODE:

```
{  
  type: 'changeTheme',  
  theme: 'dark', // or 'light'  
}
```

```
{  
  type: 'updatePersonalInfo',  
  info: {  
    actualName: 'Xing Ming',  
    birthday: '2000-01-01',  
  }  
}
```

```
{  
  type: 'updateLastActive'  
}
```

# EXAMPLE REDUCER:

A **reducer** function

- takes the previous state
- takes an "action" ( a name + data )
- returns a **new state**

```
const reducer = (state, action) => {  
  switch(action.type) {  
    case 'changeTheme':  
      return { ...state, theme: action.theme };  
    case 'updatePersonalInfo':  
      return { ...state, ...action.info };  
    case 'updateLastActive':  
      return { ...state, lastActive: Date.now() };  
    default:  
      return state;  
  }  
};
```



# ONE WAY TO ALWAYS UPDATE LASTACTIVE

```
const reducer = (state, action) => {  
  state = { ...state, lastActive: Date.now() };  
  switch(action.type) {  
    case 'changeTheme':  
      return { ...state, theme: action.theme };  
    case 'updatePersonalInfo':  
      return { ...state, ...action.info };  
    case 'updateLastActive':  
      return state;  
    default:  
      return state;  
  }  
};
```

# USEREDUCER HOOK

```
useReducer(reducer, initialArg);
```

- `initialArg` is the initial state
- returns `[ state, dispatch ]`
  - `state` is the current state
  - `dispatch` is the `dispatcher` function

Updates the state (and triggers any re-renders):

- `dispatch({ type: 'changeTheme', theme: 'dark' });`
- You can pass `dispatch` as a prop to descendants
- They can dispatch actions without other callbacks

# REACT EXAMPLE

Assume `initState` and `reducer` are imported:

```
const App = () => {
  const [state, dispatch] = useReducer(reducer, initState);
  const changeTheme= (e) => dispatch({
    type: 'changeTheme',
    theme: e.target.value
  });
  return (
    <div className={state.theme}>
      <select value={state.theme} onChange={changeTheme}/>
        <option value="light">Light</option>
        <option value="dark">Dark</option>
      </select>
    </div>
  );
};
```

# WHEN TO USEREDUCER?

`useState` is **not wrong**

use `useReducer` when you have:

- multiple pieces of state that need to update simultaneously
- complex state changing logic
  - such as state changing based on state
- state-changing logic that you want
  - to reuse
  - to have testable outside of components