



BUILDING REACT APPLICATIONS WITH REDUX

YURI TAKHTEYEV

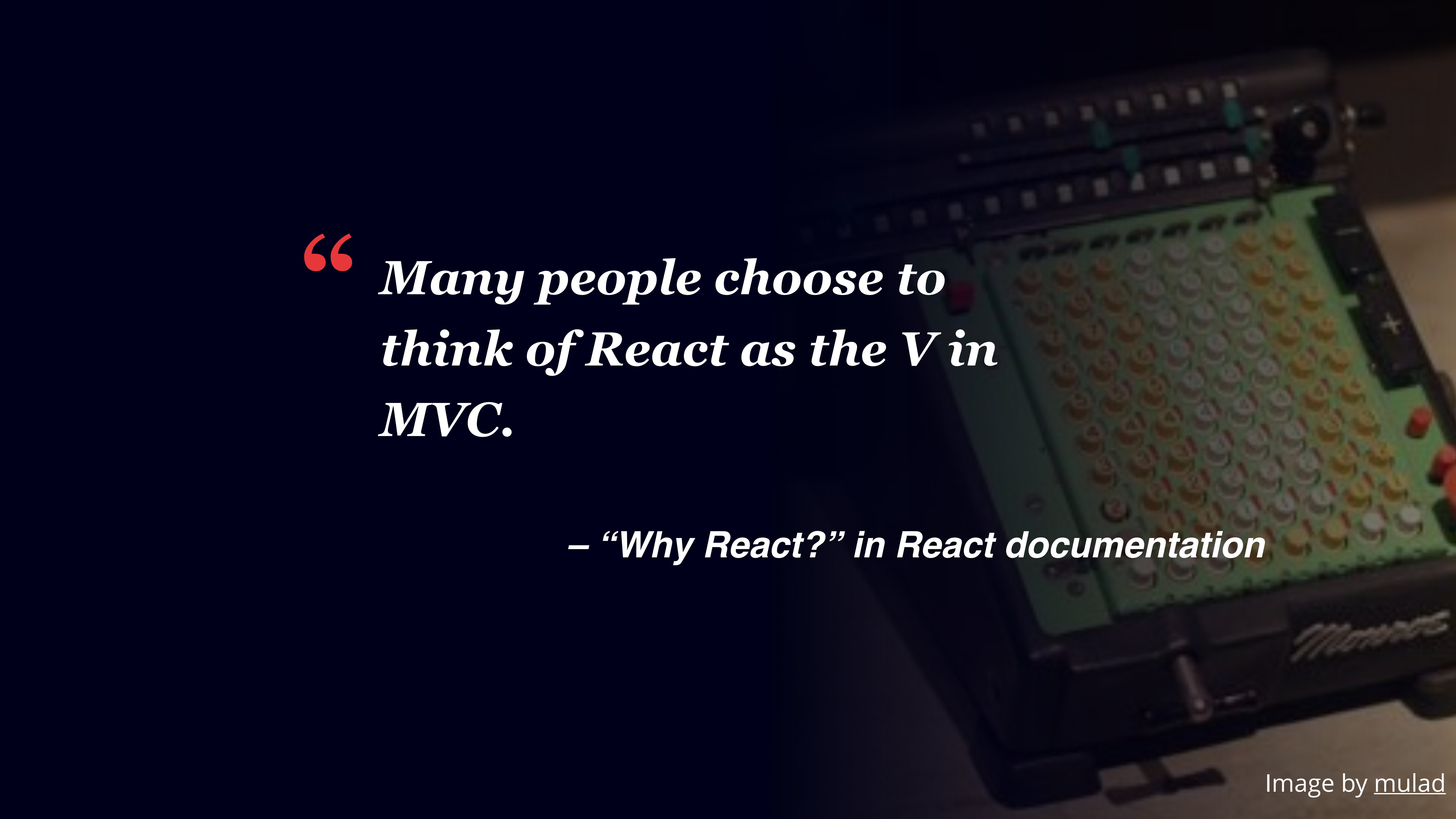
RANGLE.IO
REWRITING THE WEB



SSID: uoft // Spotlight // FITC2016



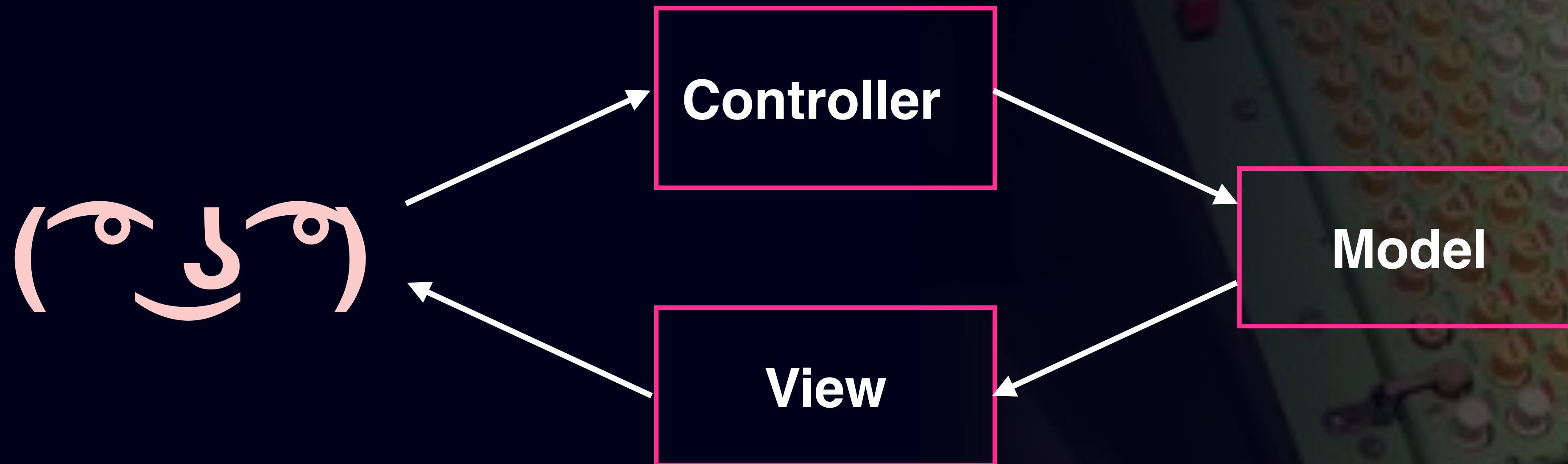
@FITC #FITCSPOTLIGHT



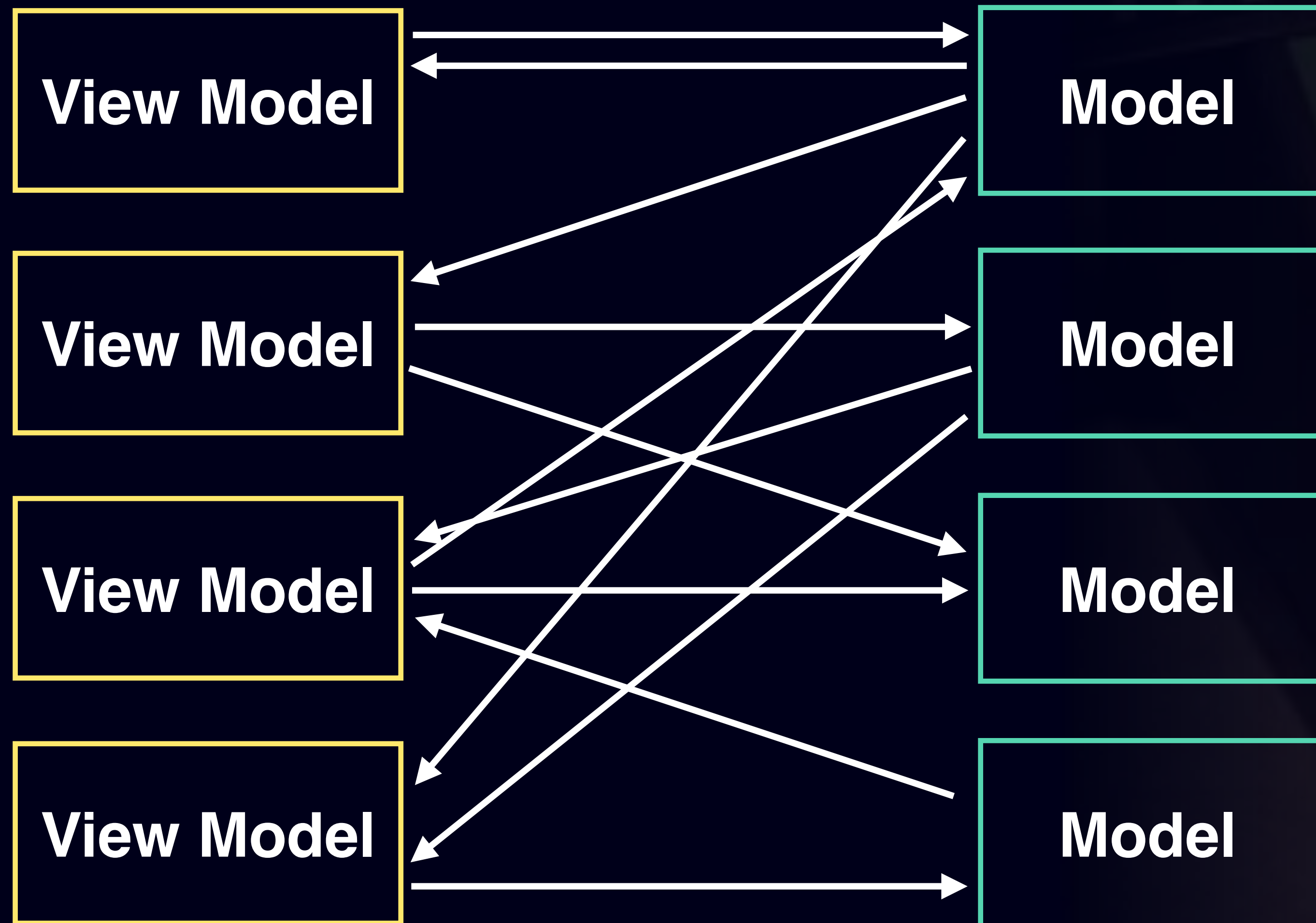
“Many people choose to think of React as the V in MVC.

– “Why React?” in React documentation

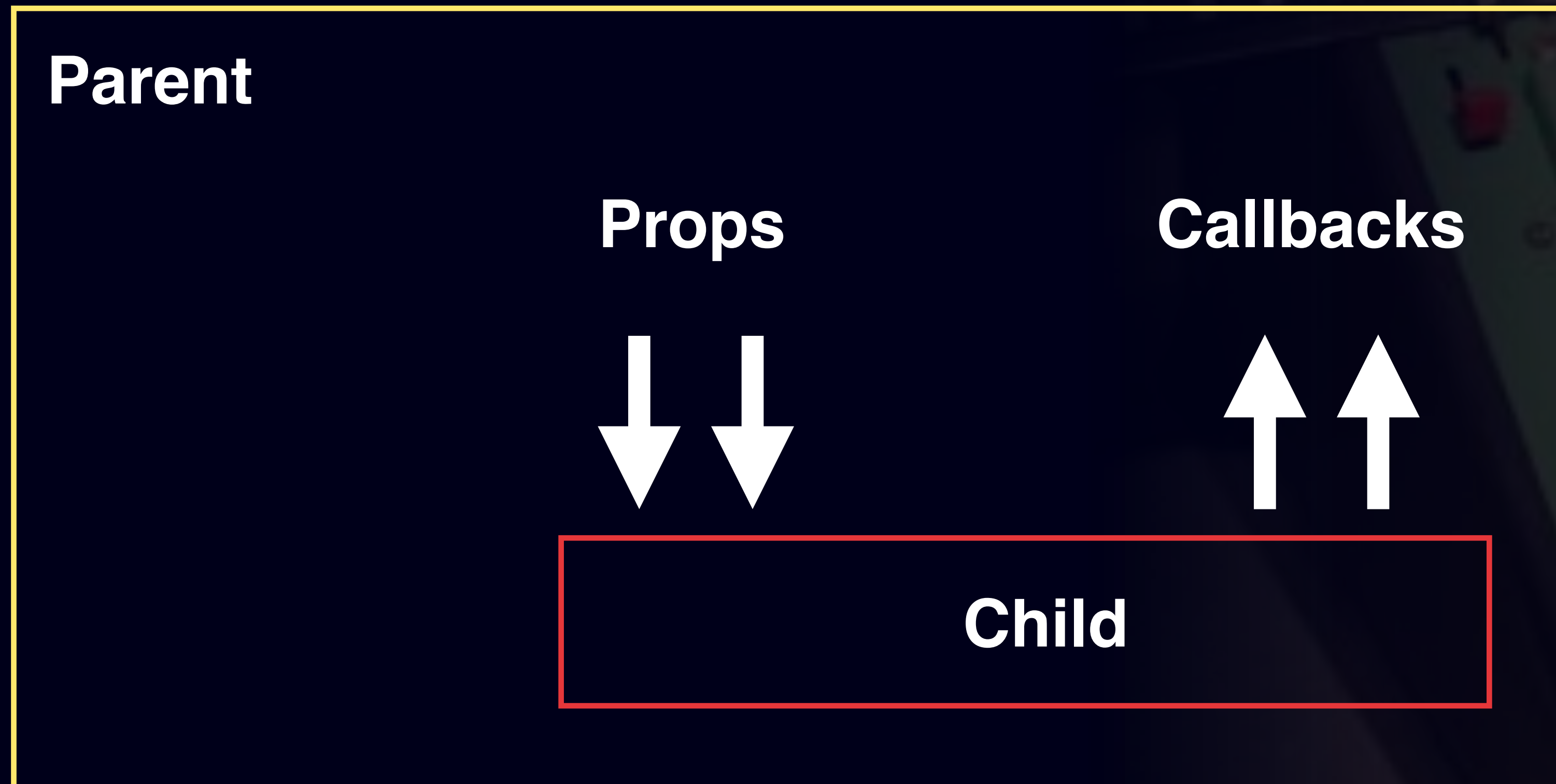
MVC in Theory



MVC in Practice



Isolated (“Dumb”) Components



☞ Maximize your use of “dumb” components.

Advantages of Isolated Components

- Helps with code reuse.
- Helps with testing.
- Makes it easy to reason about your code.

Sources of Those Benefits

- Dumb components are isolation from the rest of the application.
- Dumb components are mostly stateless.

Trouble with State

- A lot of us were raised on OOP. Objects are stateful.
- The effect of calling a method depends on the arguments and the object's state.
- Very painful to test.
- Very hard to understand.
- Aims to mimic the real world.
- But why replicate the unnecessary complexity?

Alternative: Pure Functions

- The output of a pure function depends only on inputs.
- Pure function has no side-effects.
- Much easier to understand.
- Much easier to test.
- Very easy to build through composition.

Truly Stateless Components

```
function HelloMessage(props) {  
  return <div>Hello {props.name}</div>;  
}
```


Even Better

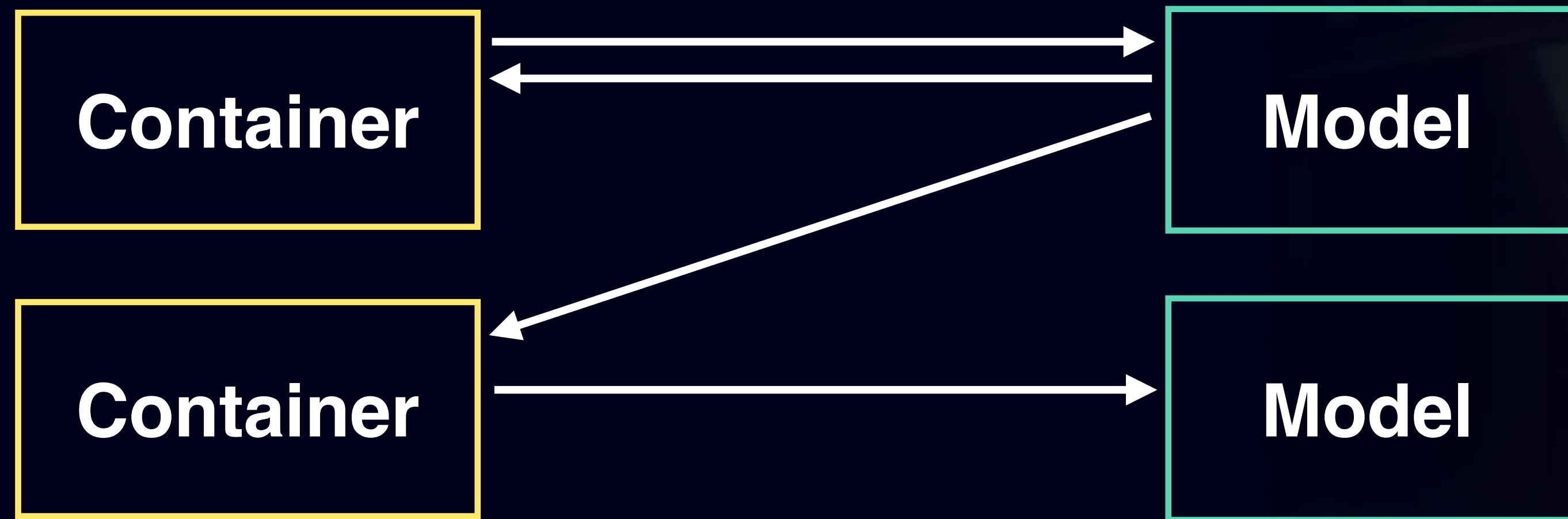
```
const HelloMessage = (props) => <div>Hello {props.name}</div>;
```

➡ Use stateless components whenever you can.

Where Does Business Logic Go?

- Each component gets props from its parent.
- Each component transmits actions to the parent.
- Turtles all the way ~~down~~ up?
- Top-level components need to be a bit smarter!
- We'll call them "container components".

Linking Containers with Models



Nevermind...

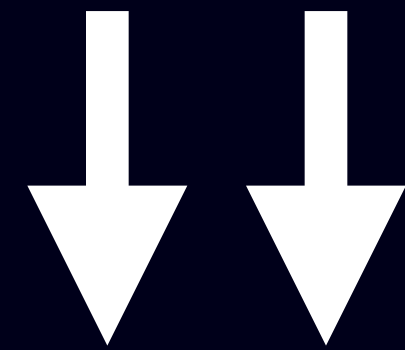
Should We Use the Container's State?

- We could, but so much for separation of concerns...

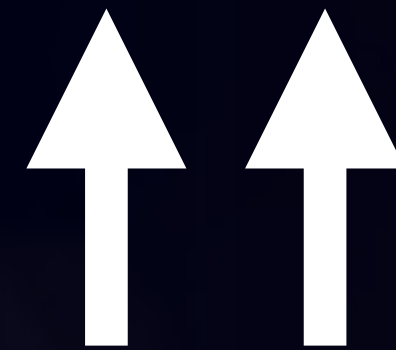
What If We Did It the React Way?

???

Props



Callbacks

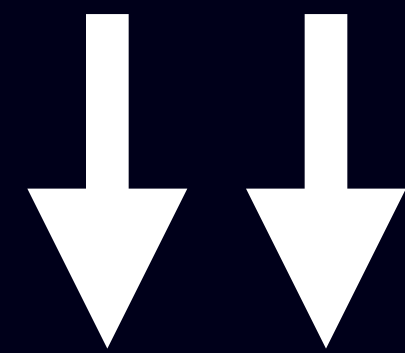


Container Component

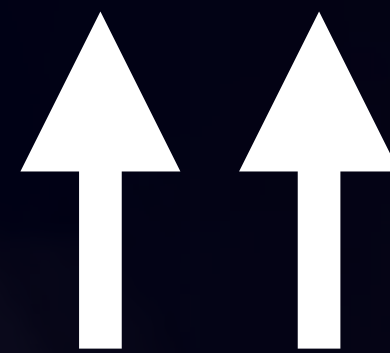
What If We Did It the React Way?

State Container

Props



Callbacks

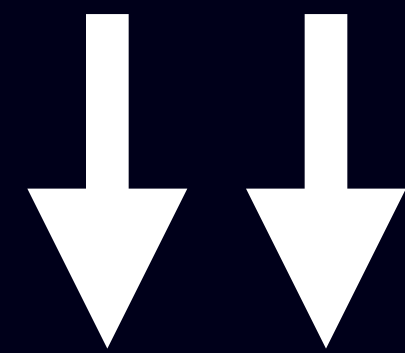


Container Component

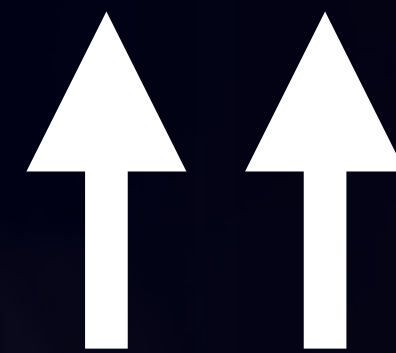
Flux Architecture

State Container

Change events

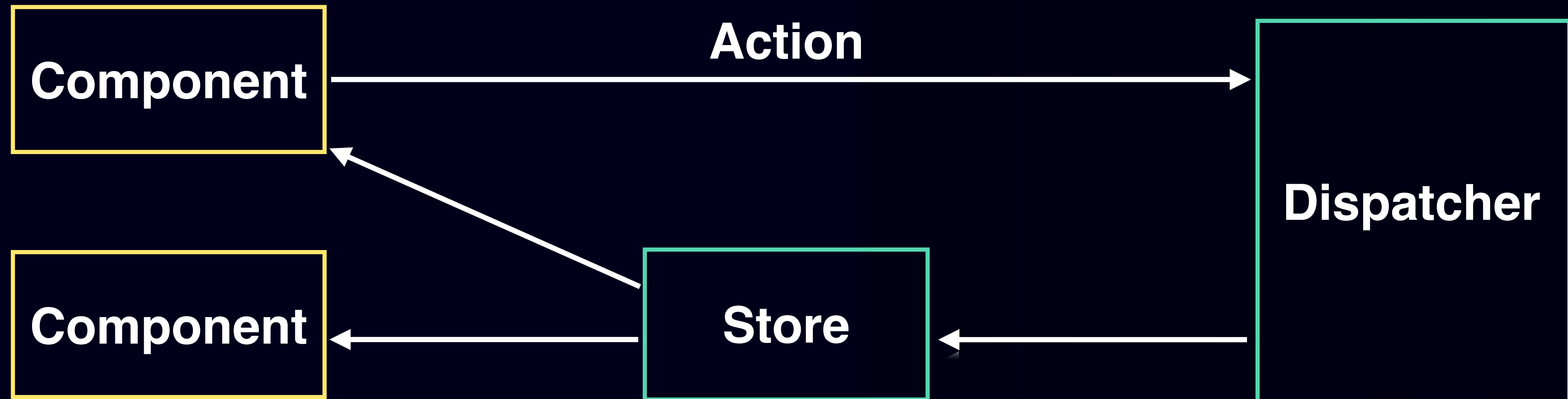


Actions

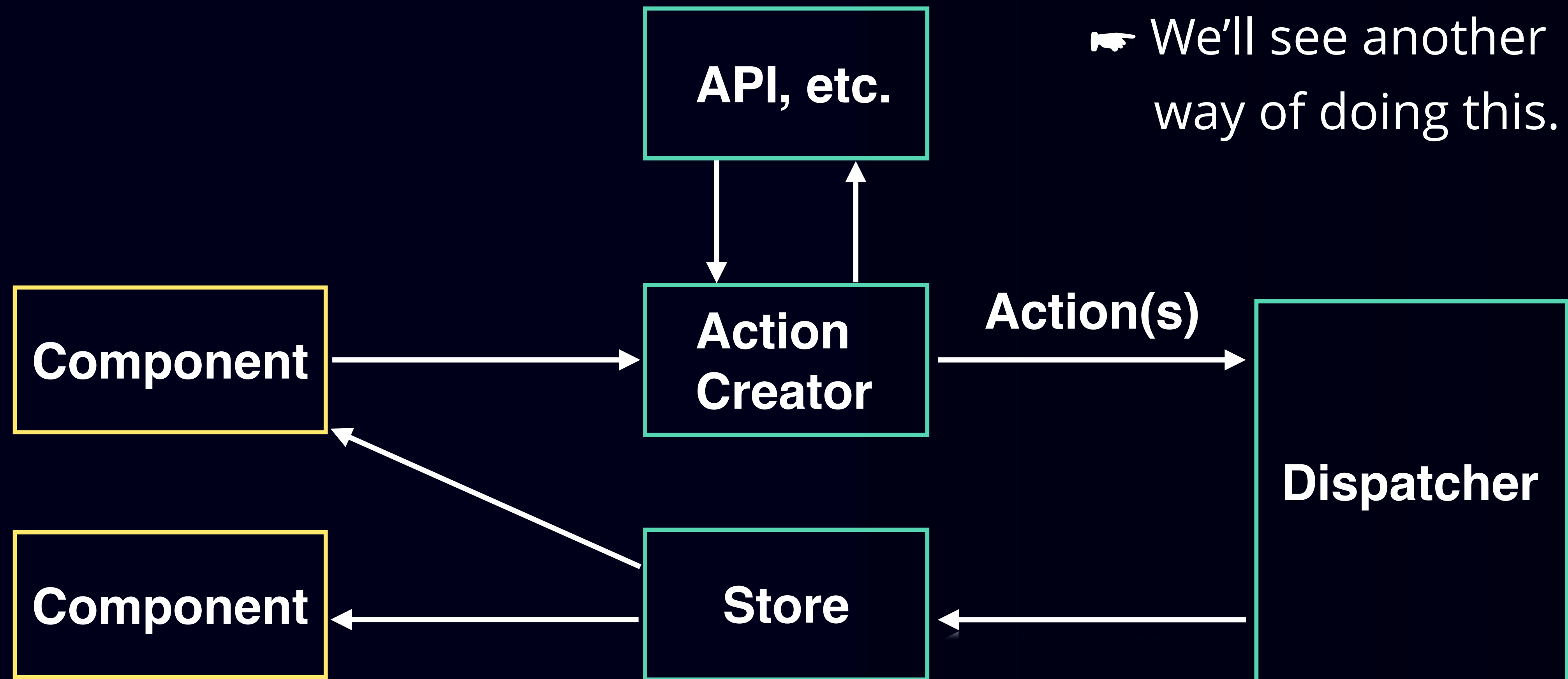


Container Component

Handling Data in the State Container



Handling Data in the State Container



Redux

A Better State Container

- <https://github.com/rackt/redux>
- Pretty much the dominant way of doing Flux with React today.
- Key concepts: a single store + state “reducers”.
 - Can a single store be a good idea?
 - What the heck is a “reducer”?

Reducer Functions

- **Basic** `reduce()`:

```
function addOneValue (value, state) {  
    return state + value;  
}
```

```
[1, 2, 3, 4, 5].reduce(addOneValue, 0);
```

- **Running through the items:**

1: 0 => 1

2: 1 => 3

3: 3 => 6

4: 6 => 10

5: 10 => 15

Action Reducers in Redux

- Take an action and a state, return a new state.
- Your app's state is a reduction over the actions.
- Each reducer operates on a subset of the global state.
- Simple reducers are combined into complex ones.

Creating an Action

```
const UPDATE_NOTE_CONTENT = 'App/UPDATE_NOTE_CONTENT';

function updateNoteContent(noteId, newContent) {
  return {
    type: UPDATE_NOTE_CONTENT,
    payload: {
      noteId: noteId,
      newContent: newContent
    }
  }
}
```


Creating an Action, more ES6

```
const UPDATE_NOTE_CONTENT = 'App/UPDATE_NOTE_CONTENT';

const updateNoteContent = (noteId, newContent) => ({
  type: UPDATE_NOTE_CONTENT,
  payload: {noteId, newContent}
});
```


A Reducer

```
export const notes = createReducer({
  [UPDATE_NOTE_CONTENT]: (state, action) => state.setIn(
    ['byId', action.payload.noteId, 'html'],
    action.payload.newContent
  ),
  [SOME_OTHER_ACTION]: (state, action) => {
    // do something else or just
    return state;
  }),
});
```

A Reducer, more ES5ish

```
export const notes = createReducer({
  [UPDATE_NOTE_CONTENT]: function (state, action) {
    return state.setIn(
      ['byId', action.payload.noteId, 'html'],
      action.payload.newContent
    );
  },
  [SOME_OTHER_ACTION]: function (state, action) {
    // do something else or just
    return state;
  },
});
```


Why Is This a Good Idea?

- Reducers are pure functions – easy to understand, easy to test.
- Reducers are synchronous.
- Data logic is 100% separated from view logic.
- You can still have modularity by combining reducers.
- New opportunities for tools.

How Do We Avoid Mutating State?

- Reducers are supposed to not change the old state. But how do we keep them honest?
- One option is to clone the object every time.
- “Immutable.js” is the library to use.

```
var newData = data.setIn(  
  [ 'foo', 'bar', 'baz' ],  
  42  
);
```

- This is a key building block for stateless architecture.

The Reducer Example Again

```
export const notes = createReducer({
  [UPDATE_NOTE_CONTENT]: (state, action) => state.setIn(
    ['byId', action.payload.noteId, 'html'],
    action.payload.newContent
  ),
  [SOME_OTHER_ACTION]: (state, action) => {
    // do something else or just
    return state;
  }),
});
```

Connecting Container Components

```
import { connect } from 'react-redux';  
  
export default connect(  
  mapStateToProps,  
  mapDispatchToProps,  
) (MainPage);
```


React-Redux Mapping Functions

```
function mapStateToProps(state) {  
  const notesAsJS = state.notes.toJS();  
  return {  
    notes: notesAsJS.ordered.map((id) => notesAsJS.byId[id])  
  };  
}
```

```
function mapDispatchToProps(dispatch) {  
  return {  
    onContentChange: (noteId, newContent) =>  
      dispatch(noteActions.updateNoteContent(  
        noteId, newContent)),  
  };  
}
```

Handling Asynchronous Actions

- Option 1: “Smart” actions creators.
- Option 2: Middleware.
- Option 2': React Sagas.

Smart Action Creators

- Call an action creator to initiate a request.
- The action creator emits an action to inform everyone that a request was made.
- The action creator makes a request.
- If the request succeeds, the action creator emits an action to inform everyone that a request was completed. (Same for failure and timeout.)
- The details of interacting with the server should be pushed into a module.

An Example, Part 1

```
function makeFetchNotesAction(status, data) {  
  return {  
    type: FETCH_NOTES,  
    payload: {  
      status: status,  
      data: data  
    }  
  };  
}
```


An Example, Part 1

```
export function fetchNotes() {  
  return dispatch => {  
    dispatch(makeFetchNotesAction('request'));  
    return fetchData()  
      .then(json => dispatch(makeFetchNotesAction('success', json)))  
      .catch(error => dispatch(makeFetchNotesAction('error', error)));  
  };  
}
```

Alternatively: Redux Middleware

- **A middleware is a function that allows you to capture actions and do something before they get to the reducers.**
- **This can include things like API calls.**
- **More generally, the sky is the limit.**
- **Perhaps a little too generic for many cases.**

Alternative: Redux Sagas

- **A middleware that works with generators.**
- **Awesome, if you understand generators.**
- **Can simplify complex workflows.**
- **Very new.**

Redux Sagas Example

```
function* fetchNotesForStack(action) {  
  const stackId = action.payload.stackId;  
  yield put(getNotes.request());  
  const {data, error} = yield dataApi.getNotesForStack(stackId);  
  if (!error) {  
    yield put(getNotes.success(organizeNotes(data.notes)));  
  } else {  
    yield put(getNotes.failure(error));  
  }  
}  
  
function* watchFetchNotesForStack() {  
  yield* takeEvery(SELECT_STACK, fetchNotesForStack);  
}
```


Testing Is Easy

```
describe('on ADD_NOTE', () => {  
  it('should create a new note', () => {  
    const getNoteCount = () => initialState.toJS().ordered.length;  
    const previousCount = getNoteCount();  
    const newState = notesReducer(initialState, {type: ADD_NOTE});  
    assert.strictEqual(previousCount + 1, getNoteCount());  
    const newNoteId = newState.toJS().ordered[0];  
    const newNoteHtml = newState.getIn(['byId', newNoteId, 'html']);  
    assert.strictEqual(newNoteHtml, 'No content');  
  });  
});
```

Caveats

- Its addictive.
- You won't be happy using anything else.
- Your friends might not understand your obsession.

THANK YOU!



Yuri Takhteyev

CTO, Rangle.io



@qaramazov



rangle

RANGLE.IO

REWRITING THE WEB



Some rights reserved - Creative Commons 2.0 by-sa