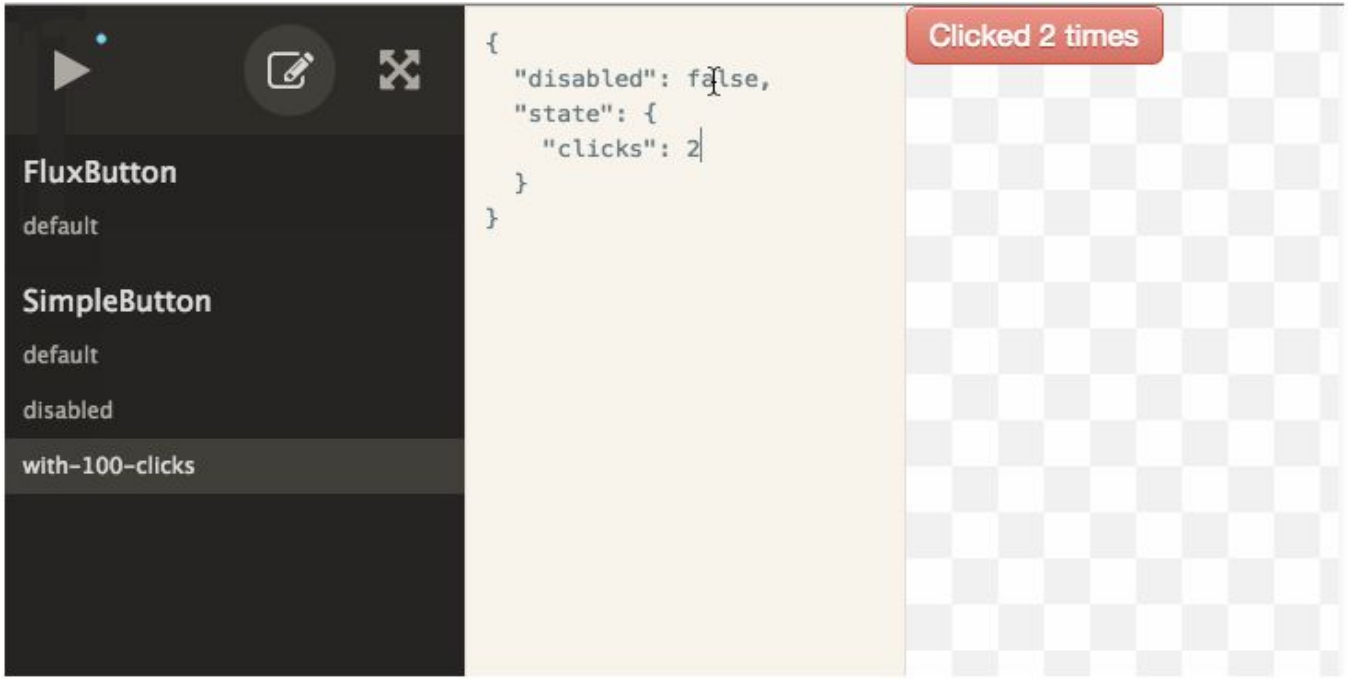
The future of React Cosmos

New look, features and more

What is Cosmos?

- DX tool for designing React components
- Render components in isolation
- Edit state and props with live feedback

Then...


FluxButton
default
SimpleButton
default
disabled
with-100-clicks
{
"disabled": false,
"state": {
"clicks": 2
}
}
Clicked 2 times

...and now


stles100
ButtonBearers
fibonacci
StatelessButton
with-100-clicks
{
"count": 10,
"state": {
"children": {
"bearer0": {↔},
"bearer1": {
"children": {
"button": {
"clicks": 1
}
}
},
"bearer2": {
"children": {
"button": {↔}
}
},
"bearer3": {
"children": {↔}
},
"bearer4": {↔},
"bearer5": {↔},
"bearer6": {
"children": {
"button": {
"clicks": 8
}
}
I bear a simple button
Click and let click
I bear a simple button
Clicked once
I bear a simple button
Clicked once
I bear a simple button
Clicked 2 times
I bear a simple button
Clicked 3 times
I bear a simple button
Clicked 5 times
I bear a simple button
Clicked 8 times
I bear a simple button
Clicked 13 times
I bear a simple button
Clicked 21 times
I bear a simple button
Clicked 34 times

New features

- Scroll to selected fixture
- Filter components & fixtures
- Resizable editor
- Enhance editor

DEMO!

ReactDOM polyfill (I)

- Tiny wrapper for ReactDOM

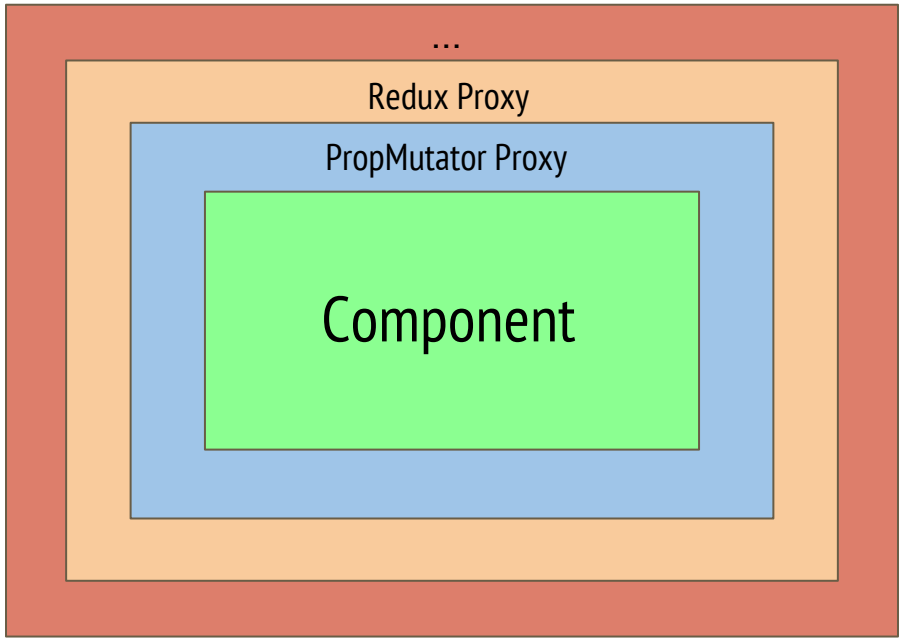
- Provides full React 0.12 - 15 compatibility through a common interface

- Makes your project available to large codebases that use old React versions

ReactDOM polyfill (II)

```
module.exports = (React) => {
  const version = parseFloat(React.version);

  if (version >= 0.14) {
    // Let bundlers (e.g. webpack) know react-dom won't always be there
    try {
      return require('react-dom');
    } catch (e) {
      return null;
    }
  } else {
    const { render, unmountComponentAtNode } = React;
    return {
      findDOMNode: (reactElement) => (
        typeof reactElement.getDOMNode === 'function' ?
          reactElement.getDOMNode() : reactElement
      ),
      render,
      unmountComponentAtNode,
    };
  }
};
```

Introducing proxies



A simple proxy example

```
1  const PropMutatorProxy = React.createClass({
2    render() {
3      return React.cloneElement(this.props.children,
4        Object.assign({}, this.props.children.props, { myProp: true }));
5    },
6  });
```

The first proxy (I)

Background problem: Cosmos couldn't render components that used state container frameworks (Flux/Redux).

Solution: *Redux proxy*

Passes a fake store to the component via context.

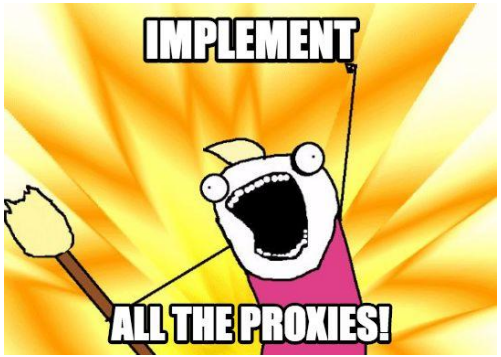
Store state is defined in each fixture, so different components/fixtures can be provided with different stores.

The first proxy (II)

```
class ReduxProxy extends React.Component {
  constructor(fixture) {
    super();
    this.store = createStore((state) => state, fixture[storeKey]);
  }

  getChildContext() {
    return {
      store: this.store,
    };
  }

  render() {
    const { type, props, ref } = this.props.children;

    return (
      <div>
        {React.createElement(type, _.assign({}, _.omit(props, storeKey), { ref }))}
      </div>
    );
  }
}
```

Proxies are the future

- Split non-mandatory Cosmos logic to proxies
- Add more proxies to help users: Resize, Drag, Fixed size, ...
- Make Cosmos easier to integrate and setup
- The dream: 1 command to run Cosmos



Help and feedback always welcome!

github.com/skidding/react-cosmos

github.com/bogdanjsx

Q & A

Thank you!