MobX & React

React Chicago Meetup
September 2017

Arthur Kay

Senior Software Engineer
IronNet Cybersecurity

JavaScript Mentor
Thinkful

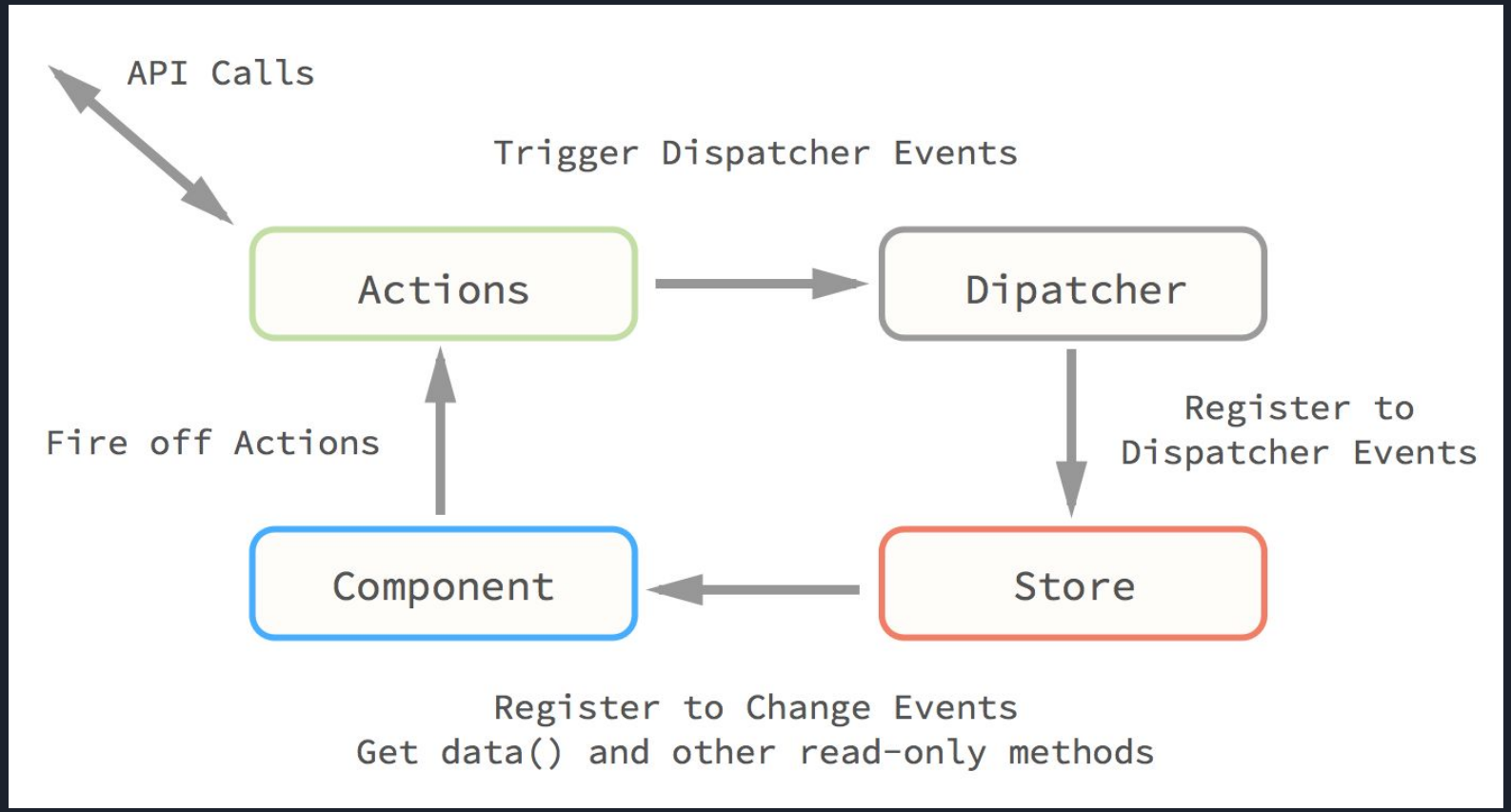
Antioch, IL

www.akaWebDesign.com

@arthurakay

https://mobx.js.org

- Any value can be **observable**
- Any component can be an **observer**
- Components (**observers**) will automatically re-render when **observable values** change



Flux Pattern: Data flows in only one direction.

Redux and MobX are both implementations of the Flux pattern.


Yeah, well, you know, that's just, like,
YOUR OPINION, MAN
quickmeme.com

Differences: **MobX** vs **Redux**

MobX

- Multi-Store
- Observable data
- Mutable state
- Nested state

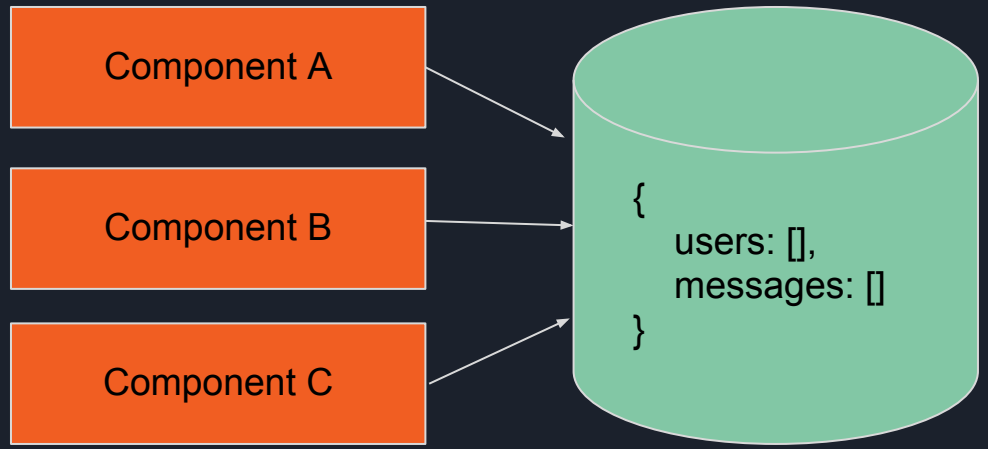
- Automatic

Redux

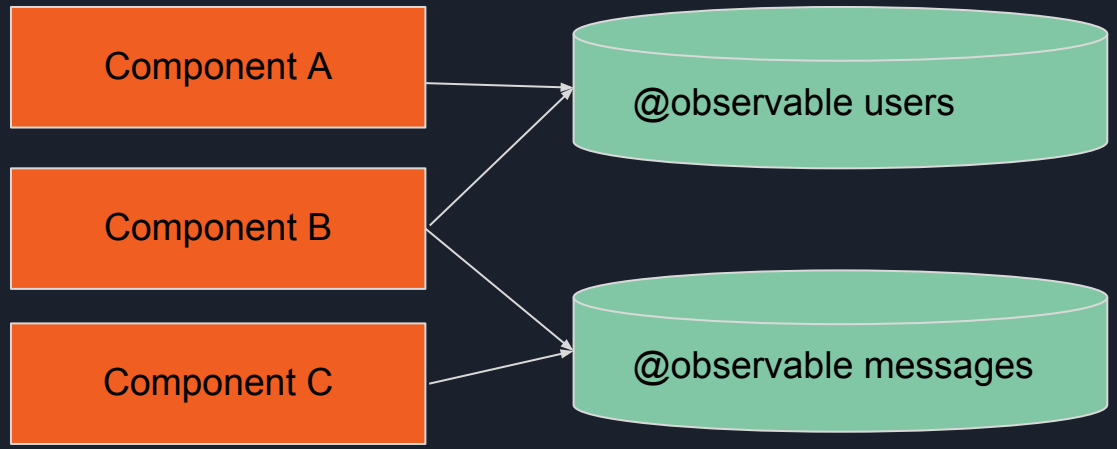
- Single Store
- Plain objects
- Immutable state
- Normalized state

- Manual

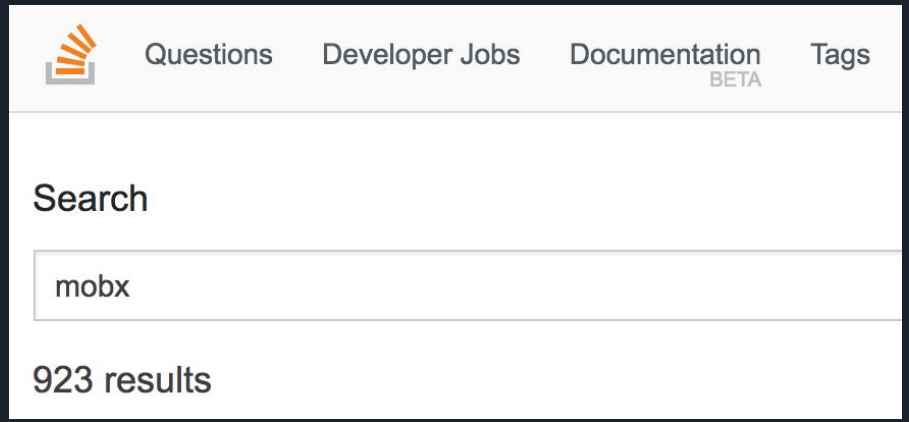
Redux: Single Store

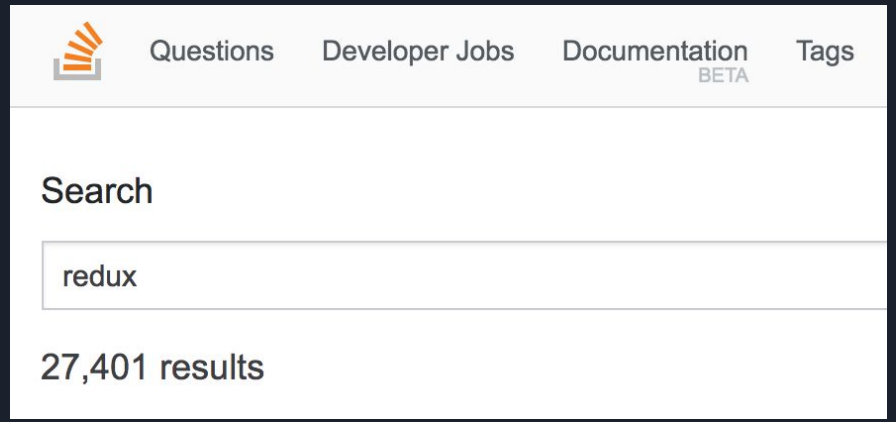


MobX: Multi-Store

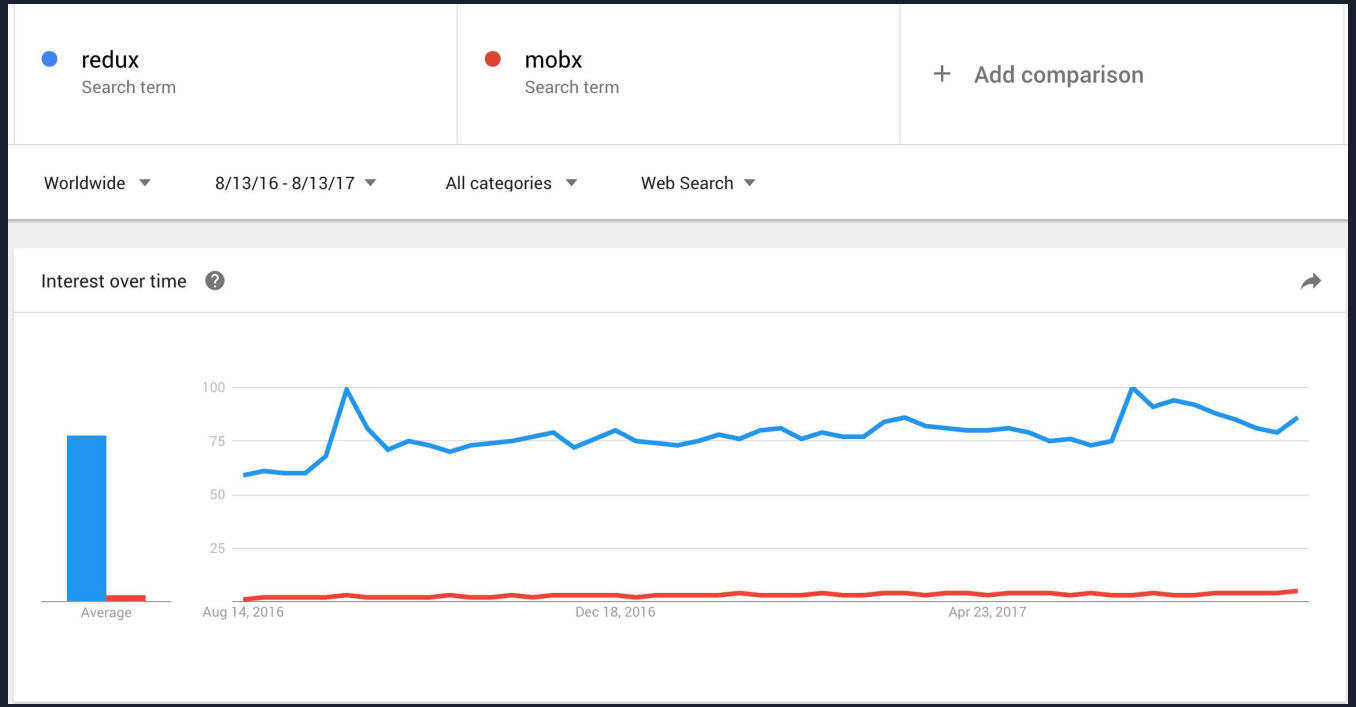


MobX vs Redux





MobX vs React



Redux

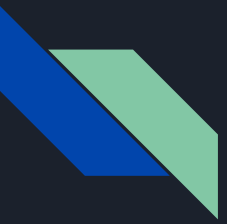
```
import React from 'react';
import { bindActionCreators } from 'redux';
import { connect } from 'react-redux';
import * as actions from '../../actions';
import TodoApp from './presenter';

function mapStateToProps(state) {
    const { todos, visibilityFilter } = state;
    return {
        todos,
        visibilityFilter
    }
}

function mapDispatchToProps(dispatch) {
    return {
        addTodo: bindActionCreators(actions.addTodo, dispatch),
        removeTodo: bindActionCreators(actions.removeTodo, dispa
        completeTodo: bindActionCreators(actions.completeTodo, c
        changeVisibilityFilter: bindActionCreators(actions.chang
    }
}

export default connect(mapStateToProps, mapDispatchToProps)(Todo
```

Over and over again...

MobX

```
class Todo {
    id = Math.random();
    @observable title = "";
    @observable finished = false;
}
```

```
class TodoList {
    @observable todos = [];
    @computed get unfinishedTodoCount() {
        return this.todos.filter(todo => !todo.finished).length;
    }
}
```



MobX

```
@observer
class TodoListView extends Component {
    render() {
        return <div>
            <ul>
                {this.props.todoList.todos.map(todo =>
                    <TodoView todo={todo} key={todo.id} />
                )}
            </ul>
            Tasks left: {this.props.todoList.unfinishedTodoCount}
        </div>
    }
}

const TodoView = observer(({todo}) =>
    <li>
        <input
            type="checkbox"
            checked={todo.finished}
            onClick={() => todo.finished = !todo.finished}
        />{todo.title}
    </li>
)
```

MobX: Computed values!

Events invoke actions. Actions are the only thing that modify state and may have other side effects.

State is observable and minimally defined. Should not contain redundant or derivable data. Can be a graph, contain classes, arrays, refs, etc.

Computed values are values that can be derived from the state using a pure function. Will be updated automatically by MobX and optimized away if not in use.

Reactions are like computed values and react to state changes. But they produce a side effect instead of a value, like updating the UI.

```
@action onClick = () => {
  this.props.todo.done = true;
}
```

```
@observable todos = [{
  title: "learn MobX",
  done: false
}]
```

```
@computed get completedTodos() {
  return this.todos.filter(
    todo => todo.done
  )
}
```

```
const Todos = observer({ todos } =>
  <ul>
    todos.map(todo => <TodoView ... />
  </ul>
)
```

MobX: Computed values!

```
class User {
    @observable firstName = “John”;
    @observable lastName = “Doe”;

    @computed get fullName() = {
        return this.firstName + “ “ + this.lastName;
    };
}
```

60%

Of the time

Live Demo

Works every time

https://github.com/arthurakay/chicago-react-meetup-mobx

Arthur Kay

www.akaWebDesign.com

@arthurakay

Questions?

Thank you!