Relay, Redux, Om/next Oh my!

Managing App State

Jared Forsyth

jaredly
@jaredforsyth

KhanAcademy.org

Managing App State

- What’s the problem?
- Current solutions
- Recommendations

What’s the problem?

- React is the V
- M(data) V(present data) C(manage data)
- Do we just not model or control?

Current Solutions

- React Jun 2013
- Flux May 2014
- Re-frame Dec 2014
- Relay Feb 2015
- Redux May 2015
- Om/next July 2015

The Setup

Users table
id	color
1	'red'

```
const ColorPage = React.createClass({
  render() {
    return <div>
      Color: {this.props.color}
      <button onClick={
        () => this.props.onChangeColor('blue')
      }>
        Change Color to Blue
      </button>
    </div>
  }
})
```

PORC

Plain Old React Components

Setup

- root component has state + update methods

Getting shared state

- passed down from root component

Updating shared state

- root component update methods

PORC

Plain Old React Components

```
getInitialState() {
  return {users: {1: {color: 'red'}}}
```

```
onChangeColor(userId, color) {
  postColorChangeToServer(userId, color);
  this.setState(
```

```
render() {
  return <div>
    <ColorPage
      color={this.state.users[1].color}
      onChangeColor={color => this.onChangeColor(1, color)}
    />
  </div>
}
```

PORC

Plain Old React Components

Pros

- no extra learning / setup

Cons

- no extra help
- the nested data problem
  - parents know child component's needs
- very little convention

Redux

Setup

- “action creators” describe state mutations

  ex: changeColor(userId, color) -> ColorChangeAction

- “reducers” turn (state, action) -> new state

  (state w/ color=red, ColorChangeAction) -> state w/ color=blue

Getting shared state

- a wrapper function that declares data needs

Updating shared state

- the wrapper function also declares update callback functions

Redux

```
1  /* actions.js */
2  export const setColor = (userId, color) => {
3    postColorChangeToServer(userId, color);
4    return {
5      type: Constants.SET_COLOR,
6      userId: userId,
7      color: color
8    }
9  }
10 /* reducers.js */
11 export const rootReducer = (state, {type, userId, color}) => {
12   switch (type) {
13     case Constants.SET_COLOR:
14       return setIn(state, ['users', userId, 'color'], color);
15     default:
16       return state
17   }
18 }
```

← describe mutation

handle mutation

Redux

```
20 import {connect} from 'react-redux';
21 const ColorPageWrapper = connect(
22   (state, props) => ({
23     color: state.users[props.userId].color,
24   }),
25
26   (dispatch, props) => ({
27     onChangeColor: color => (
28       dispatch(actions.setColor(props.userId, color))
29     )
30   }),
31 )(ColorPage)
```

Redux

Benefits

- declarative!
- immutable state
- parents know nothing about child’s data needs
- mutation logic separate from the Views
- super nice for testing

Costs

- creating a new action requires touching 3+ files

Re-frame

A ClojureScript library similar to redux

Setup

> *action creators: same as redux*

> *reducers: same as redux*

> **subscriptions define how to get data from ‘state’**

Getting shared state

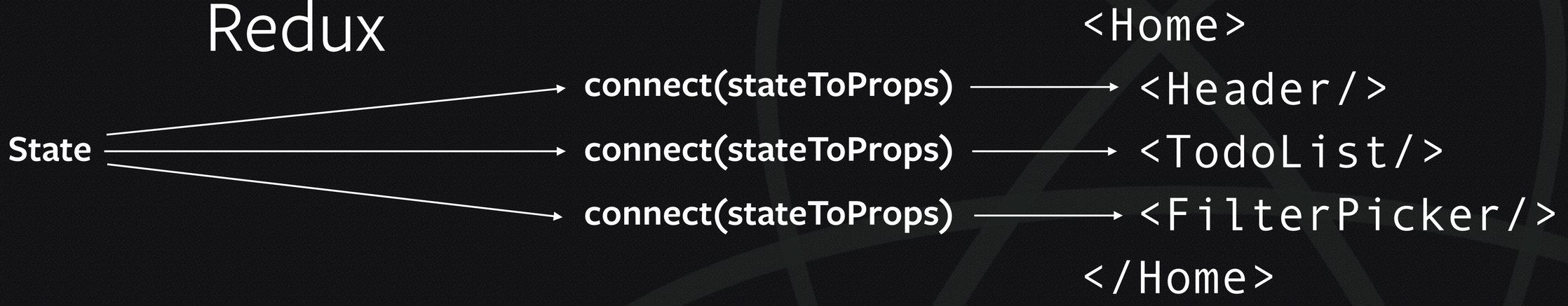
> wrapper component **uses subscriptions**

Updating shared state

> *callbacks: same as redux*

Subscriptions

Redux



Re-frame

Re-frame

Benefits

- subscriptions separate “deriving view data” from the views
- subscriptions memoize, so less redundant work is done

Costs

- more complexity

Relay

Setup

> graphql schema defined server-side
> mutations defined client + server-side
ex: ColorChangeMutation

Getting shared state

> wrapper component declares own (and child) data needs with GraphQL

Updating shared state

> create mutation, give to Relay

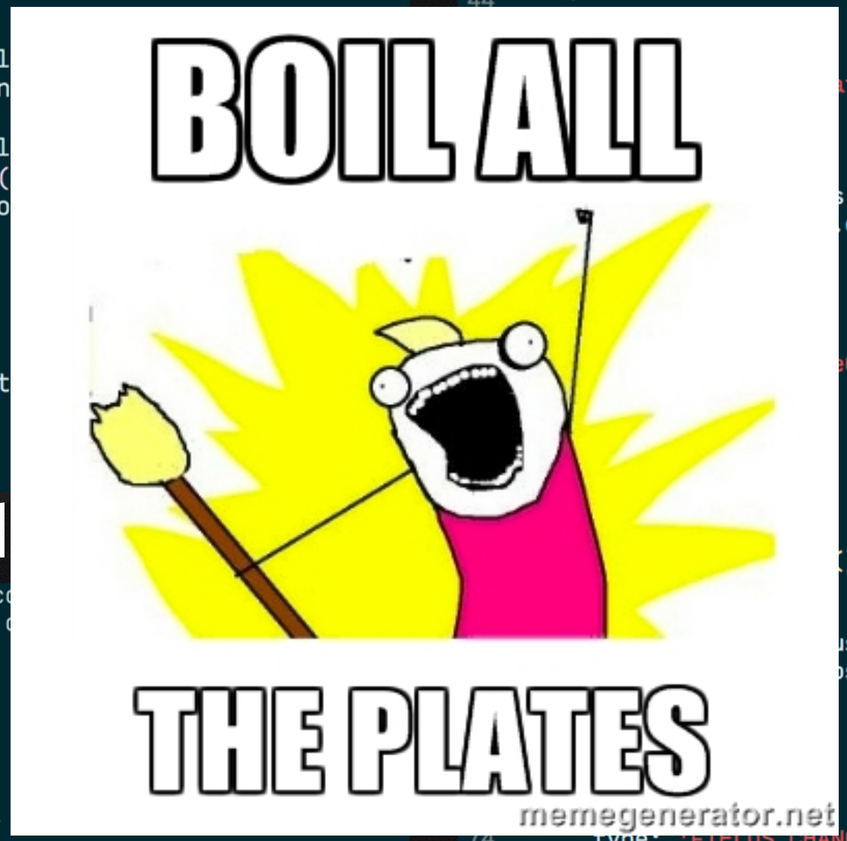
Relay Setup

Server

```
15 const ChangeColorMutation = mutationWithClientMutationId({
16   name: 'ChangeColor',
17   inputFields: {
18     id: {type: new GraphQLNonNul
19     color: {type: new GraphQLNon
20   },
21   mutateAndGetPayload: ({id, col
22     const userId = fromGlobalId(
23     setColorForUser(userId, colo
24     return {userId}
25   },
26   outputFields: {
27     user: {
28       type: User,
29       resolve: ({userId}) => get
30     },
31   },
32 });
```

vs Red

```
 2 export const setColor = (userId, c
 3   postColorChangeToServer(userId,
 4   return {
 5     type: Constants.SET_COLOR,
 6     userId: userId,
 7     color: color
 8   }
 9 }
10 export const rootReducer = (state,
11   switch (type) {
12     case Constants.SET_COLOR:
13       return setIn(state, ['users', userId, 'color'], color);
14     default:
15       return state
16   }
17 }
```



Client

```
38 class ChangeColorMutation extends Relay.Mutation {
39   static fragments = {
40     user: () => Relay.QL`
41       fragment on User {
42         id
43       }
44     `
                                    ation {changeColor}`
                                    s.user.id,
                                    color,
                                    eColorPayload {
                                    () {
                                    user.id,
                                    os.color,
74         type: 'FIELDS_CHANGE',
75         fieldIDs: {
76           user: this.props.user.id
77         }
78       }]
79   }
80 }
```

Relay

```
82 const ColorPageWrapper = React.createClass({
83   render() {
84     return <ColorPage
85       color={this.props.user.color}
86       onChangeColor={color => {
87         Relay.Store.commitUpdate(new ChangeColorMutation({
88           user: this.props.user,
89           color: color
90         }));
91       }}
92     />
93   }
94 });
95
96 const ColorPageRelayWrapper =
97 Relay.createContainer(ColorPageWrapper, {
98   fragments: {
99     user: () => Relay.QL`
100       fragment on User {
101         color
102         ${ChangeColorMutation.getFragment('user')}
103       }
104     `
105   },
106 });
```

Relay

Benefits

- even more declarative
- no custom getter logic
- tight server integration
  - intelligent caching
  - optimistic updates
  - minimal fetching
  - static query validation
- graph-data oriented

Costs

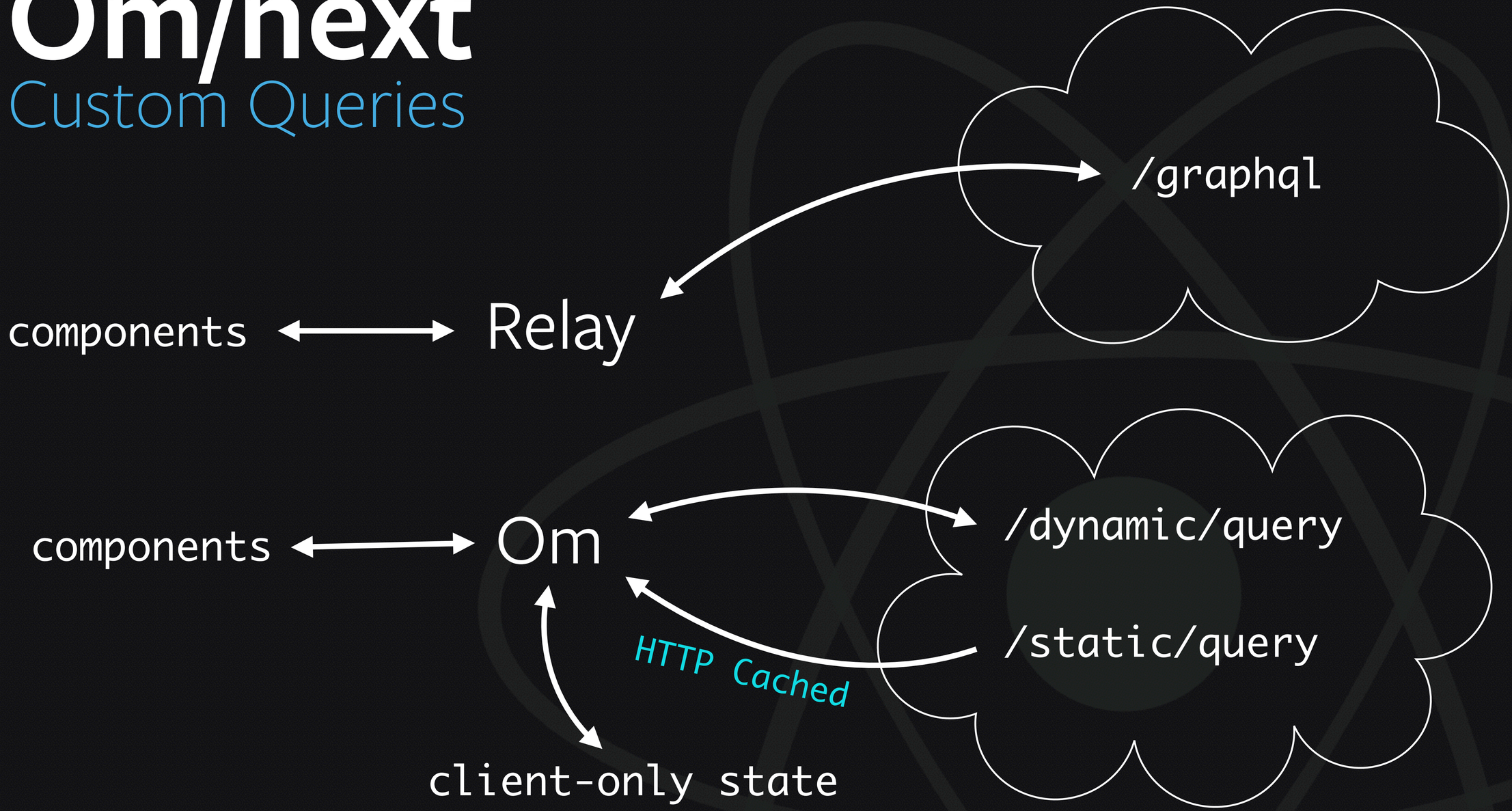
- you need a GraphQL backend
- doesn’t handle client-only state
- tons more complexity
- less flexible
- parents need some knowledge of children’s data needs

Om/next

Like Relay
+ ClojureScript
- GraphQL server
+ client-only state
+ custom queries

Om/next

Custom Queries



Om/next

Benefits

- custom query resolution
- both client & remote state fully supported
- fairly server agnostic
- very flexible

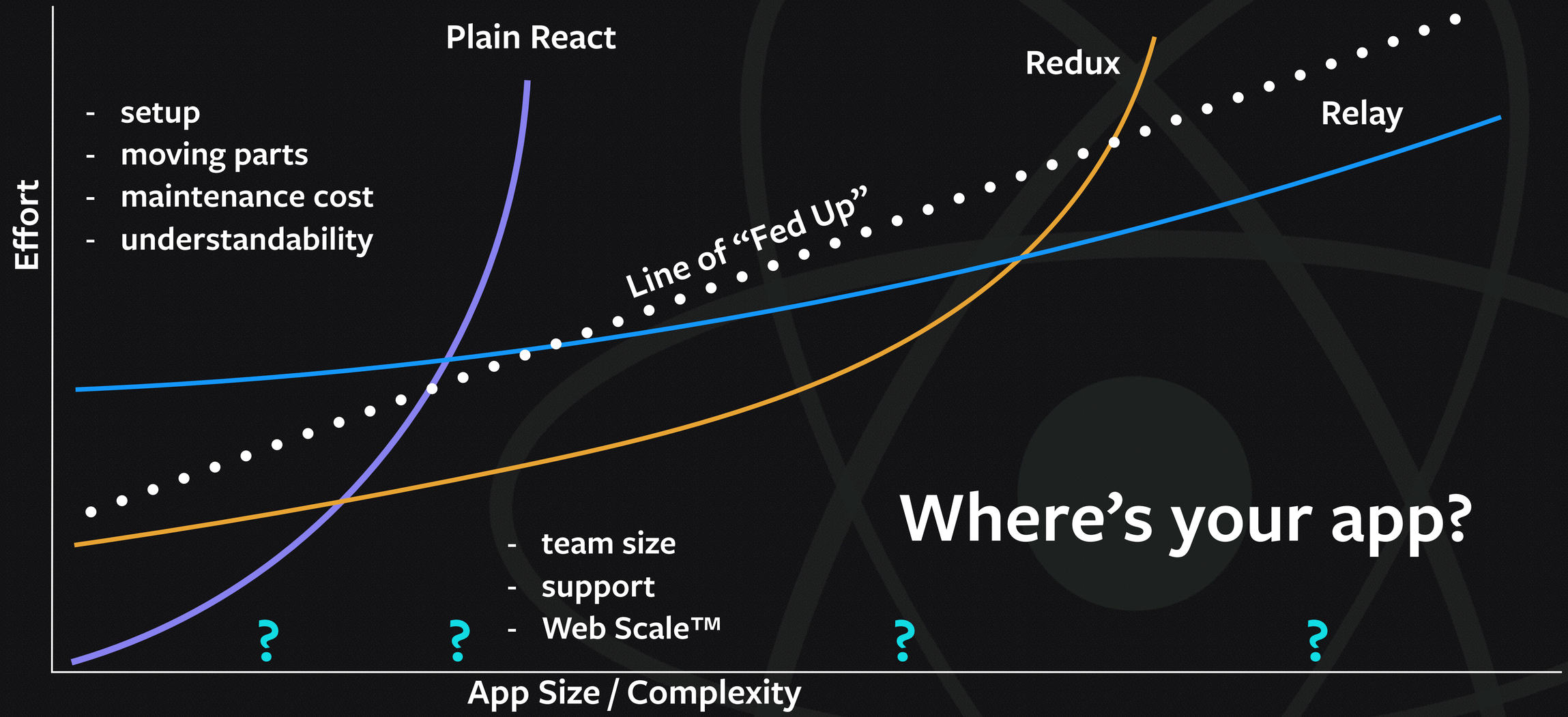
Costs

- less optimized than Relay
- still under development, various loose ends
- clojurescript :)

Recommendations

- have conventions
- use pure components
- examine tradeoffs

Recommendations



Note: not to scale. margin of error is large and not shown. axes might be logarithmic.

FIN

github.com/jaredly/reactconf

jaredly

@jaredforsyth