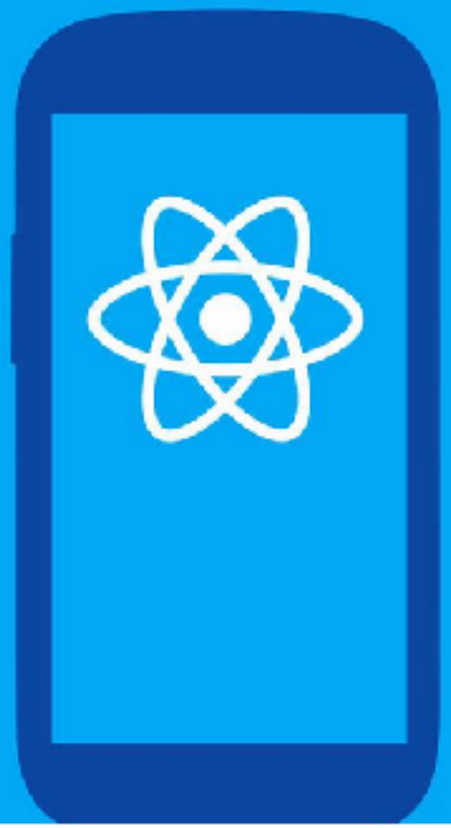
Why React matters


Reese's

ShihChi Huang

- Predictable
- Immutable



September 15 LANGUAGES · OPEN SOURCE · @SCALE · MOBILE · PLATFORM · DEVELOPMENT TOOLS

React Native for Android: How we built the first cross-platform React Native app

Daniel Witte Philipp von Weitershausen

learn once

write anywhere

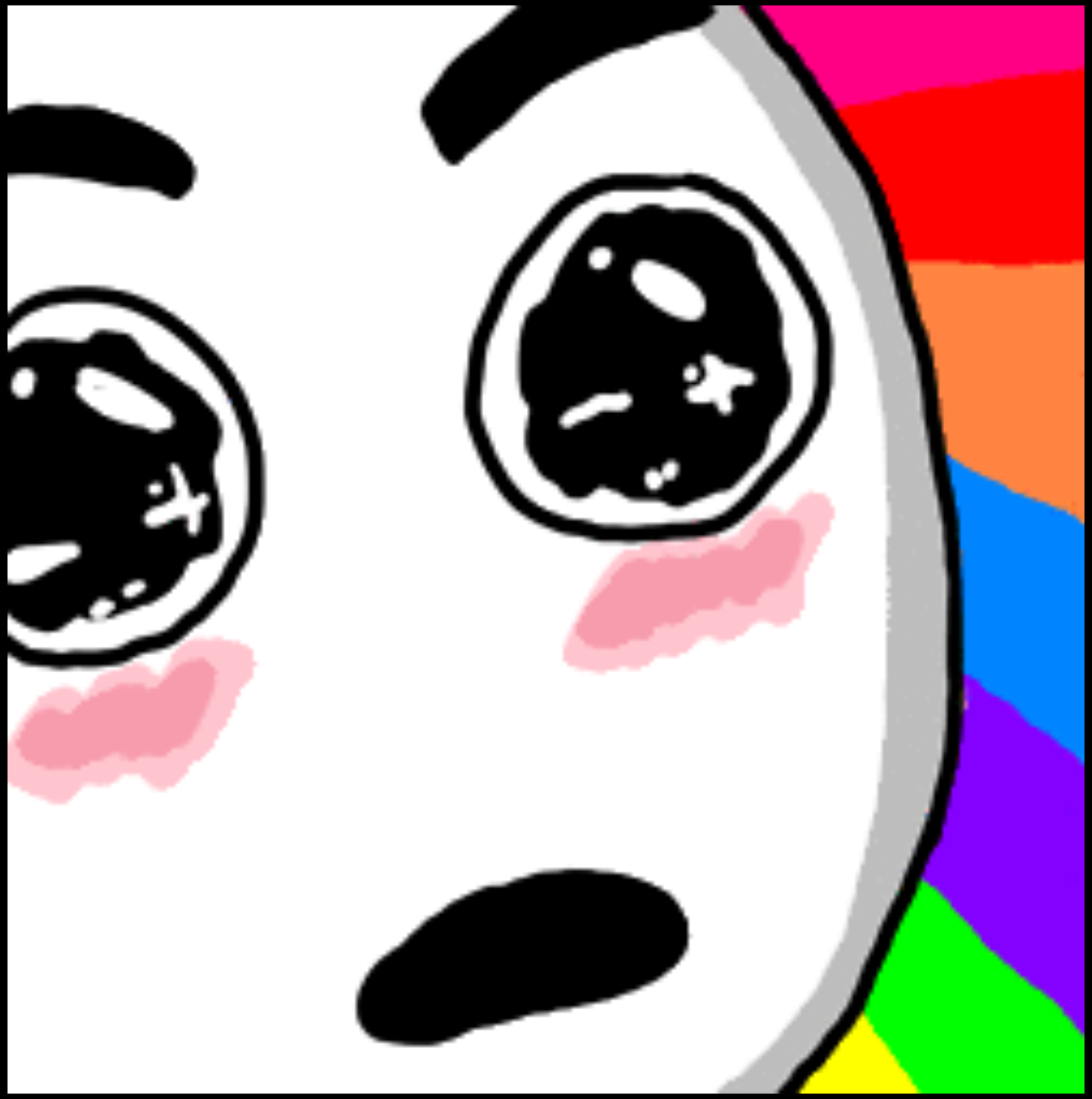
web

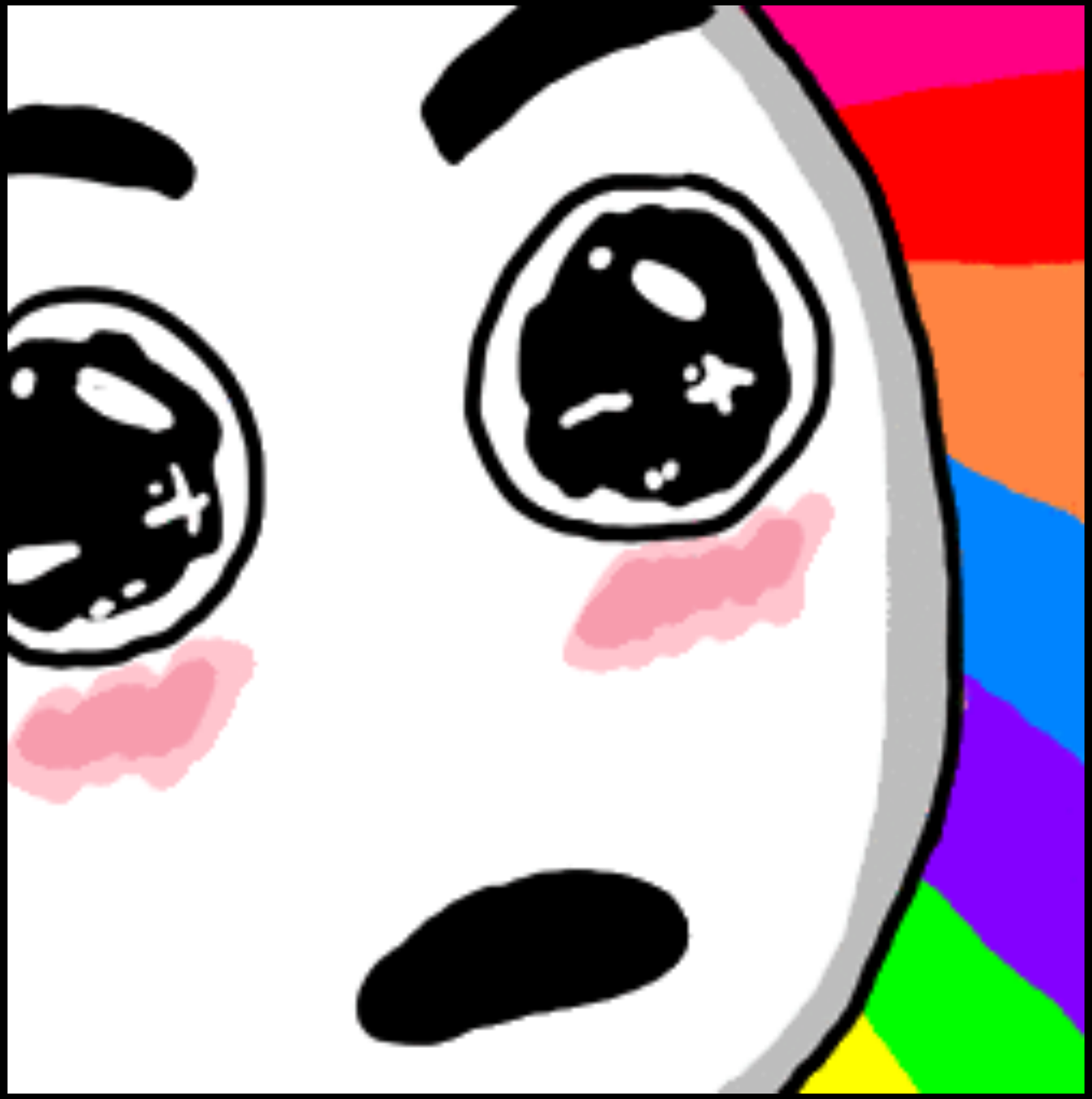
developer

web

iOS / android

developer





What’s React

A javascript library for building UI

A javascript library for building UI

A javascript library for building UI

```
var Greeting = React.createClass({
  render() {
    return (
      <p>{"HELLO " + this.props.name}</p>
    );
  }
});

<Greeting name="everyone" />
// HELLO everyone
```

against best practices

against best practices

worst technology ever*

ridiCOlous

agains tices

worst techn er*

Stop Bugs!

*

WTF

Ben Alman
@cowboy

Follow

Really? Facebook React demo advocating storing HTML in your JavaScript. Which is a HUGE step back in terms of maintainability. #jsconf #wtf

RETWEETS 15 FAVORITES 8

5:21 AM - 30 May 2013

Georgia, USA

min(Time to Find Root Cause)

Credits: vjeux@

MVC

UNIX Philosophy

UNIX Philosophy

do one thing and do it well

UNIX Philosophy

do one thing and do it well

write programs to work together

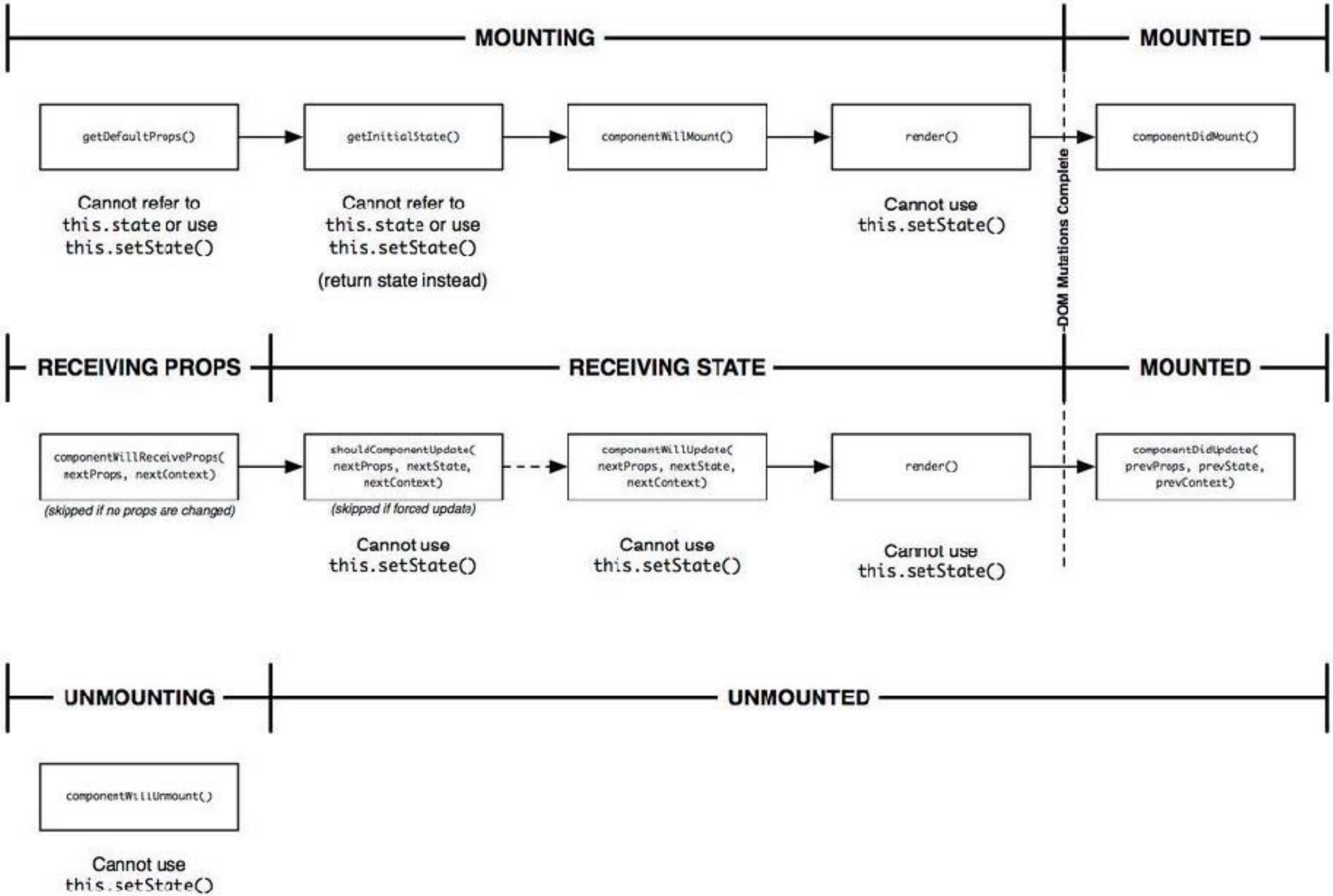
UNIX Philosophy

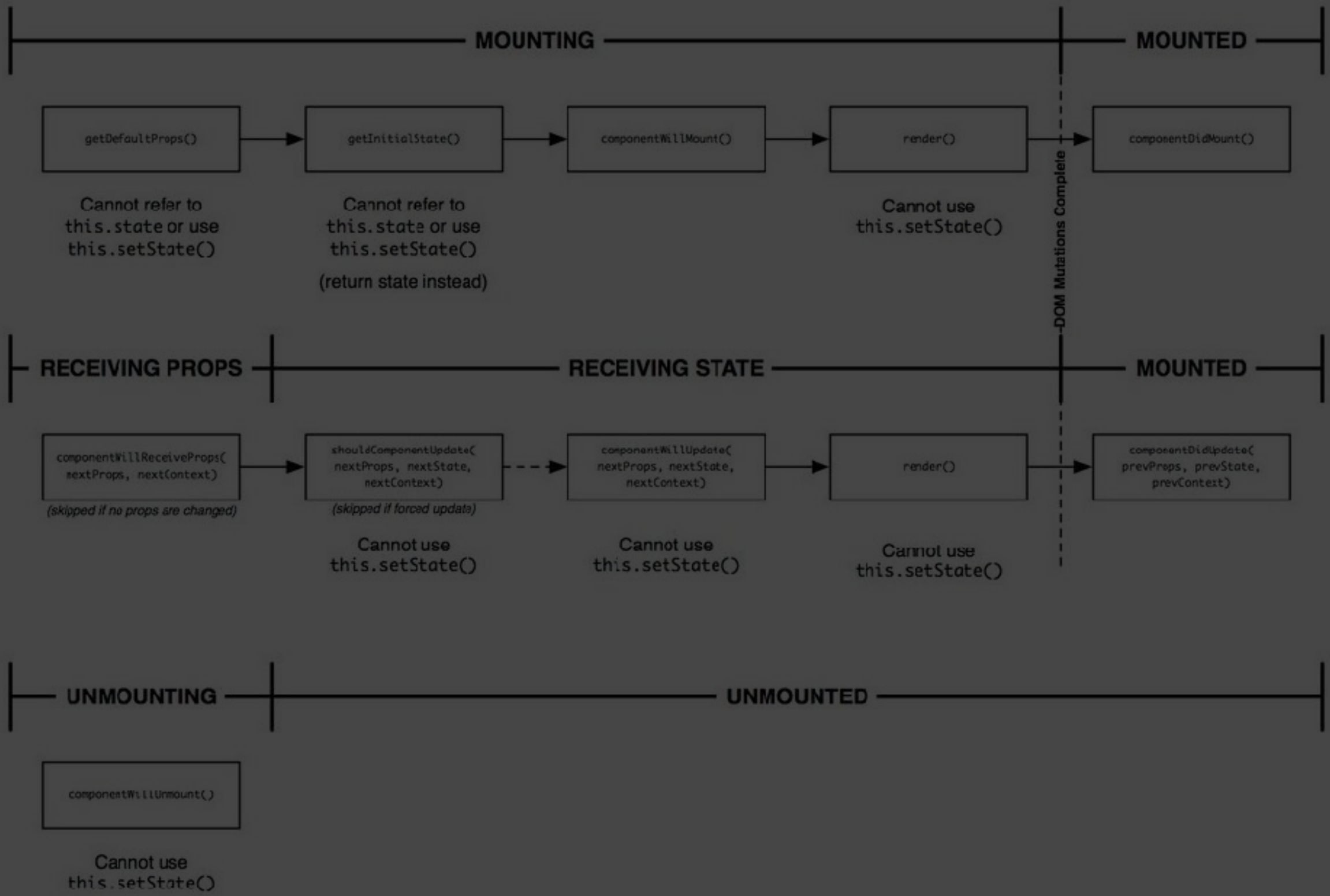
do one thing and do it well

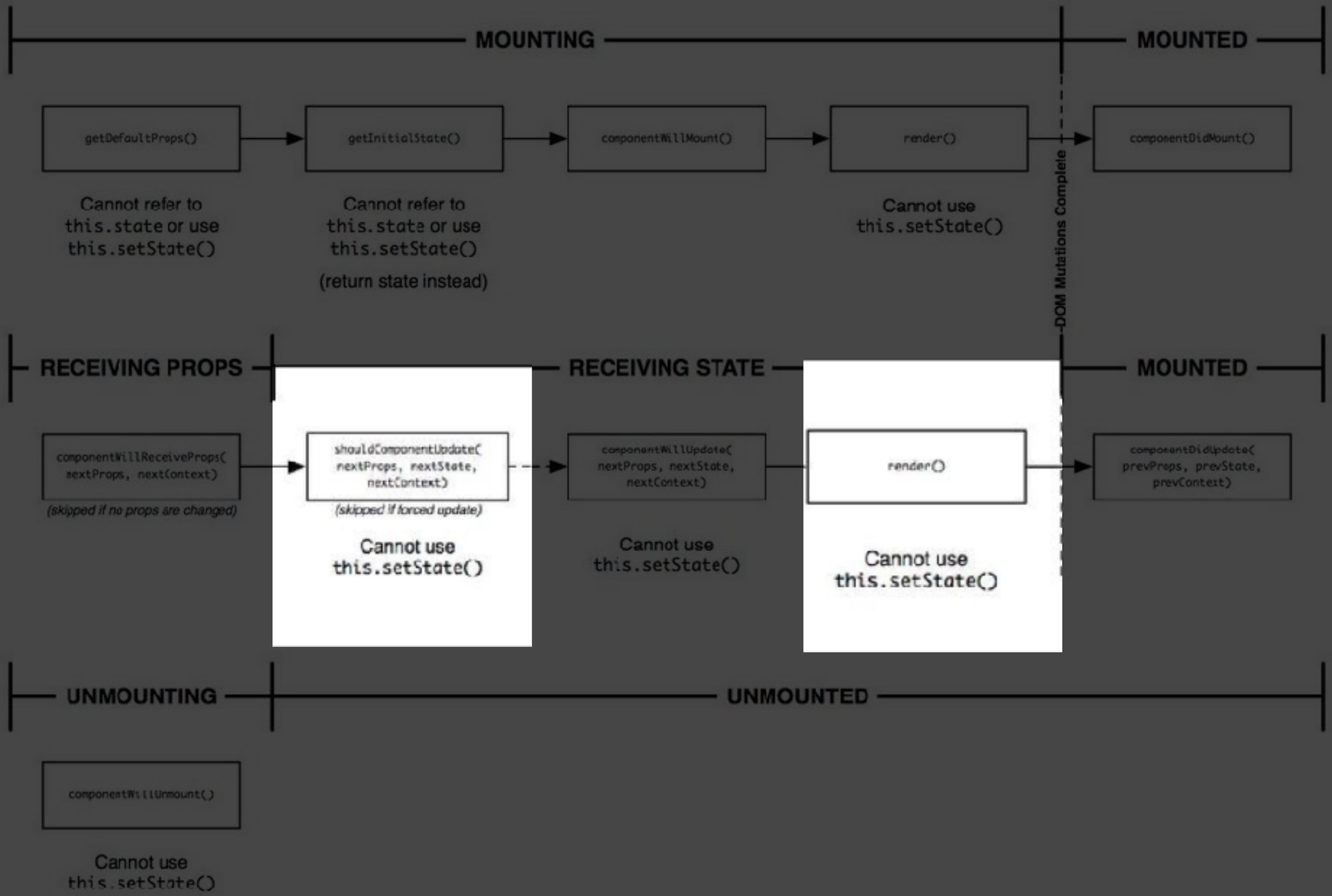
write programs to work together

handle text streams

Separation of Concerns

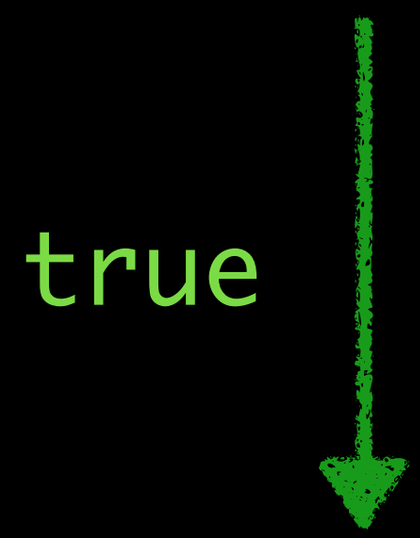

MOUNTING
MOUNTED
getDefaultProps()
getInitialState()
componentWillMount()
render()
componentDidMount()
Cannot refer to this.state or use this.setState()
Cannot refer to this.state or use this.setState()
(return state instead)
Cannot use this.setState()
DOM Mutations Complete
RECEIVING PROPS
RECEIVING STATE
MOUNTED
componentWillReceiveProps(nextProps, nextContext)
(skipped if no props are changed)
shouldComponentUpdate(nextProps, nextState, nextContext)
(skipped if forced update)
componentWillUpdate(nextProps, nextState, nextContext)
render()
componentDidUpdate(prevProps, prevState, prevContext)
Cannot use this.setState()
Cannot use this.setState()
Cannot use this.setState()
UNMOUNTING
UNMOUNTED
componentWillUnmount()
Cannot use this.setState()


MOUNTING
MOUNTED
getDefaultProps()
getInitialState()
componentWillMount()
render()
componentDidMount()
Cannot refer to this.state or use this.setState()
Cannot refer to this.state or use this.setState()
(return state instead)
Cannot use this.setState()
DOM Mutations Complete
RECEIVING PROPS
RECEIVING STATE
MOUNTED
componentWillReceiveProps(nextProps, nextContext)
(skipped if no props are changed)
shouldComponentUpdate(nextProps, nextState, nextContext)
(skipped if forced update)
componentWillUpdate(nextProps, nextState, nextContext)
render()
componentDidUpdate(prevProps, prevState, prevContext)
Cannot use this.setState()
Cannot use this.setState()
Cannot use this.setState()
UNMOUNTING
UNMOUNTED
componentWillUnmount()
Cannot use this.setState()


MOUNTING
MOUNTED
getDefaultProps()
getInitialState()
componentWillMount()
render()
componentDidMount()
Cannot refer to this.state or use this.setState()
Cannot refer to this.state or use this.setState()
(return state instead)
Cannot use this.setState()
DOM Mutations Complete
RECEIVING PROPS
RECEIVING STATE
MOUNTED
componentWillReceiveProps(nextProps, nextContext)
(skipped if no props are changed)
shouldComponentUpdate(nextProps, nextState, nextContext)
(skipped if forced update)
Cannot use this.setState()
componentWillUpdate(nextProps, nextState, nextContext)
Cannot use this.setState()
render()
Cannot use this.setState()
componentDidUpdate(prevProps, prevState, prevContext)
UNMOUNTING
UNMOUNTED
componentWillUnmount()
Cannot use this.setState()

shouldComponentUpdate

shouldComponentUpdate



render

ES2015

warm-up

```
function hello() {
  return 'world';
}
```

```
function hello() {
  return 'world';
}

// arrow function
var hello = () => {
  return 'world';
}
```

```
function hello() {
  return 'world';
}

// arrow function
var hello = () => {
  return 'world';
}

var hello = () => ‘world';
```

```
var state = {
  foo: foo,
  bar: bar,
}
```

```
var state = {
  foo: foo,
  bar: bar,
}
```

```
// enhanced object literals
var state = {



};
```

```
var state = {
  foo: foo,
  bar: bar,
}
```

```
// enhanced object literals
var state = {
  foo,

};
```

```
var state = {
  foo: foo,
  bar: bar,
}
```

```
// enhanced object literals
var state = {
  foo,
  bar,
};
```

```
var Greet = React.createClass({
  render() {
    return <div />;
  }
});

// class
class Greet extends React.Component {
  render() {
    return <div />;
  }
}
```

```
var PropTypes = React.PropTypes;
var Components = React.Components;

// destructuring
var { PropTypes, Component } = React;
```

```
var PropTypes = React.PropTypes;
var Components = React.Components;

// destructuring
var { PropTypes, Component } = React;
```

```
var PropTypes = React.PropTypes;
var Components = React.Components;

// destructuring
var { PropTypes, Component } = React;
```

```
var todos = this.props.todos.map(todo =>
  <TodoItem



  />
});
```

```
var todos = this.props.todos.map(todo =>
  <TodoItem
    id={todo.id}


  />
});
```

```
var todos = this.props.todos.map(todo =>
  <TodoItem
    id={todo.id}
    content={todo.content}

  />
});
```

```
var todos = this.props.todos.map(todo =>
  <TodoItem
    id={todo.id}
    content={todo.content}
    isCompleted={todo.isCompleted}
  />
});
```

```
var todos = this.props.todos.map(todo =>
  <TodoItem
    id={todo.id}
    content={todo.content}
    isCompleted={todo.isCompleted}
  />
});

// spread operator
var todos = this.props.todos.map(todo =>
  <TodoItem {...todo} />
});
```

```
var data = {
  foo: 'bar',
  hello: 'world',
  answer: 42,
};
```

```
var data = {
  foo: 'bar',
  hello: 'world',
  answer: 42,
};

var { foo, ...rest } = data;
```

```
var data = {
  foo: 'bar',
  hello: 'world',
  answer: 42,
};

var { foo, ...rest } = data;

// rest
{
  hello: 'world',
  answer: 42
}
```

```
// arrow function
var hello = () => ‘world’;

// enhanced object literal
var state = { foo, bar };

// destructuring
var { PropTypes, Component } = React;

// class
class Greet extends React.Component {}

// spread operator
<TodoItem {...todo} />
```

UI = f(state)

```
UI = render(state)
```

pure function

is a function where
the return value is only
determined by its input values

```
// f(x) = x + 1;
var increment = (x) => {
  return x + 1;
}

// f(x) = 2^x * x^3 - x - 1;
var complex = (x) => {
  return Math.pow(2, x)
    * Math.pow(x, 3)
    - x - 1;
}
```

pure function

is a function where
the return value is only
determined by its input values

at any given time

```
var counter = 0;

setInterval(() => counter++, 1000);

var timeVariant = (x) => {
  return x + counter;
}

> timeVariant(3)
> 4
// after few seconds
> timeVariant(3)
> 7
```

Predictable

```
class TodoApp extends React.Component {
  render() {
    var { isLoaded, todos } = this.props;
    if (!isLoaded) { return <Spinner />; }
    if (!todos.length) {
      return <EmptyResult />;
    }
    return (
      <TodoList>
        {this.props.todos.map(todo => {
          return <TodoItem {...todo} />;
        })}
      </TodoList>
    );
  }
}
```

```
class TodoApp extends React.Component {
  render() {
    var { isLoaded, todos } = this.props;
    if (!isLoaded) { return <Spinner />; }
    if (!todos.length) {
      return <EmptyResult />;
    }
    return (
      <TodoList>
        {this.props.todos.map(todo => {
          return <TodoItem {...todo} />;
        })}
      </TodoList>
    );
  }
}
```

```
class TodoApp extends React.Component {
  render() {
    var { isLoaded, todos } = this.props;
    if (!isLoaded) { return <Spinner />; }
    if (!todos.length) {
      return <EmptyResult />;
    }
    return (
      <TodoList>
        {this.props.todos.map(todo => {
          return <TodoItem {...todo} />;
        })}
      </TodoList>
    );
  }
}
```

```
class TodoApp extends React.Component {
  render() {
    var { isLoaded, todos } = this.props;
    if (!isLoaded) { return <Spinner />; }
    if (!todos.length) {
      return <EmptyResult />;
    }
    return (
      <TodoList>
        {this.props.todos.map(todo => {
          return <TodoItem {...todo} />;
        })}
      </TodoList>
    );
  }
}
```

```
class TodoApp extends React.Component {
  render() {
    var { isLoaded, todos } = this.props;
    if (!isLoaded) { return <Spinner />; }
    if (!todos.length) {
      return <EmptyResult />;
    }
    return (
      <TodoList>
        {this.props.todos.map(todo => {
          return <TodoItem {...todo} />;
        })}
      </TodoList>
    );
  }
}
```

UX

In reply to @twokul

Dan Abramov
@dan_abramov

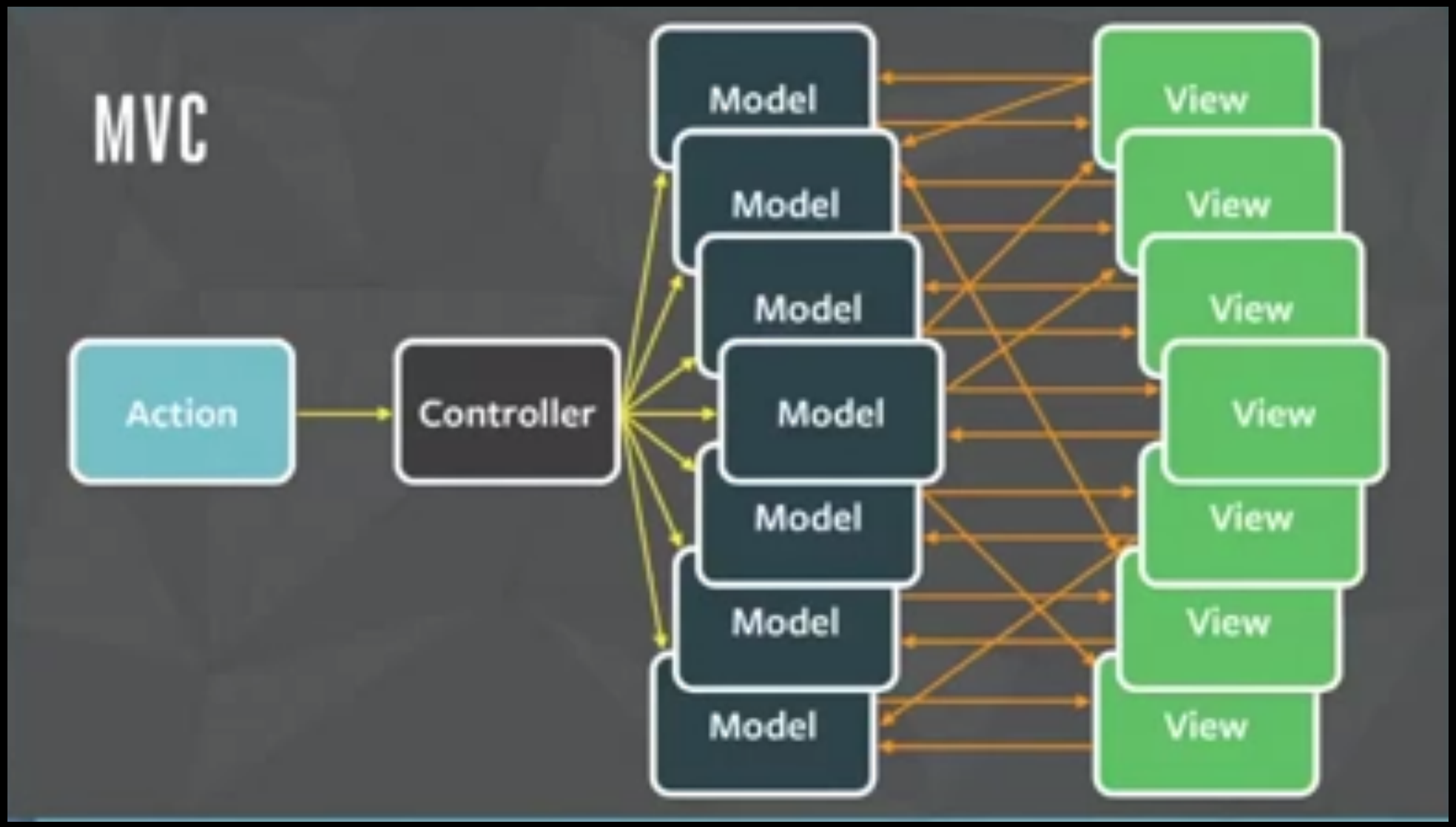
@twokul @reactjs It's predictable even when complex.

1/28/15, 2:57 AM

1 RETWEET | **1** FAVORITE

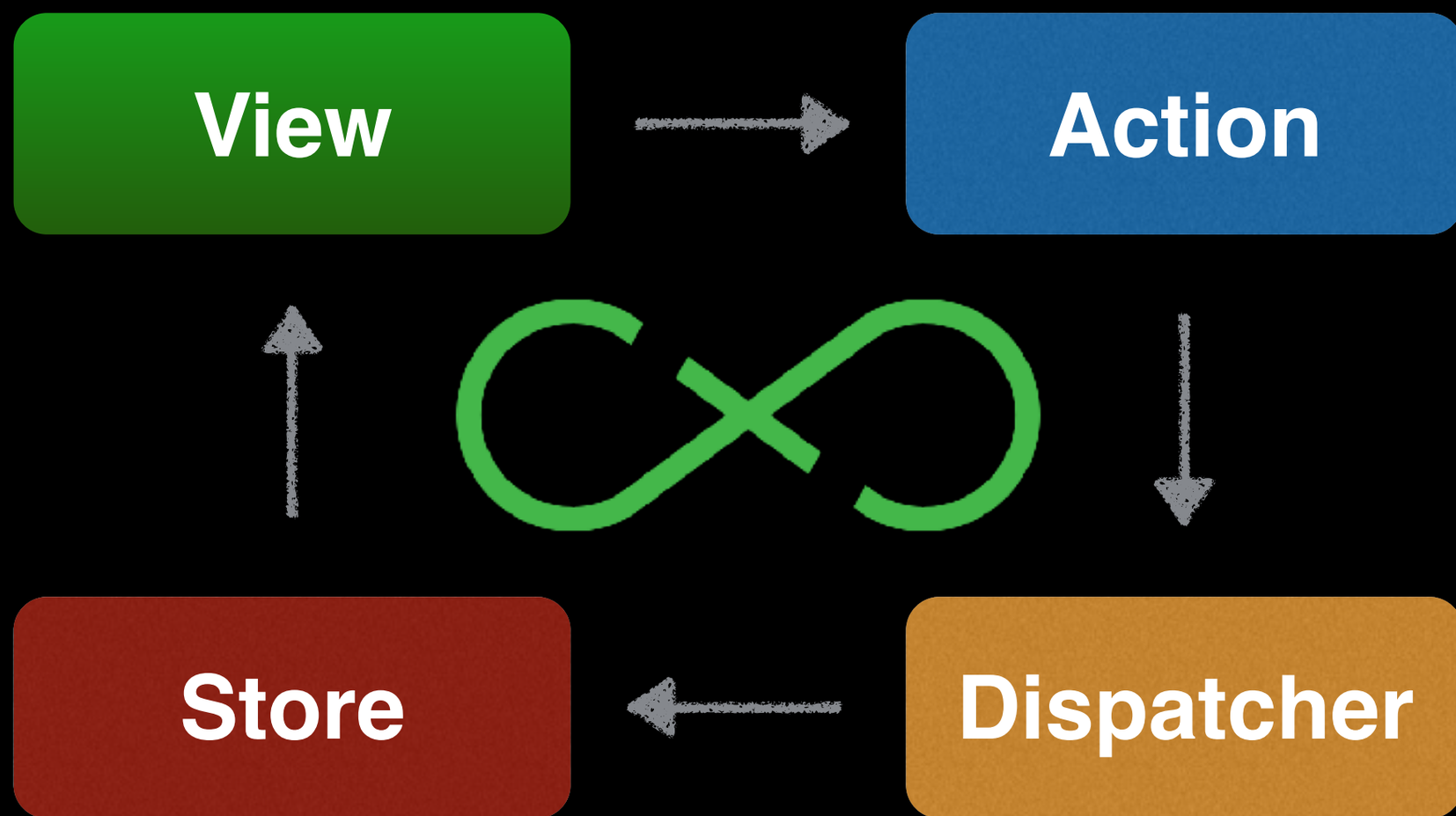

simple ≠ familiar

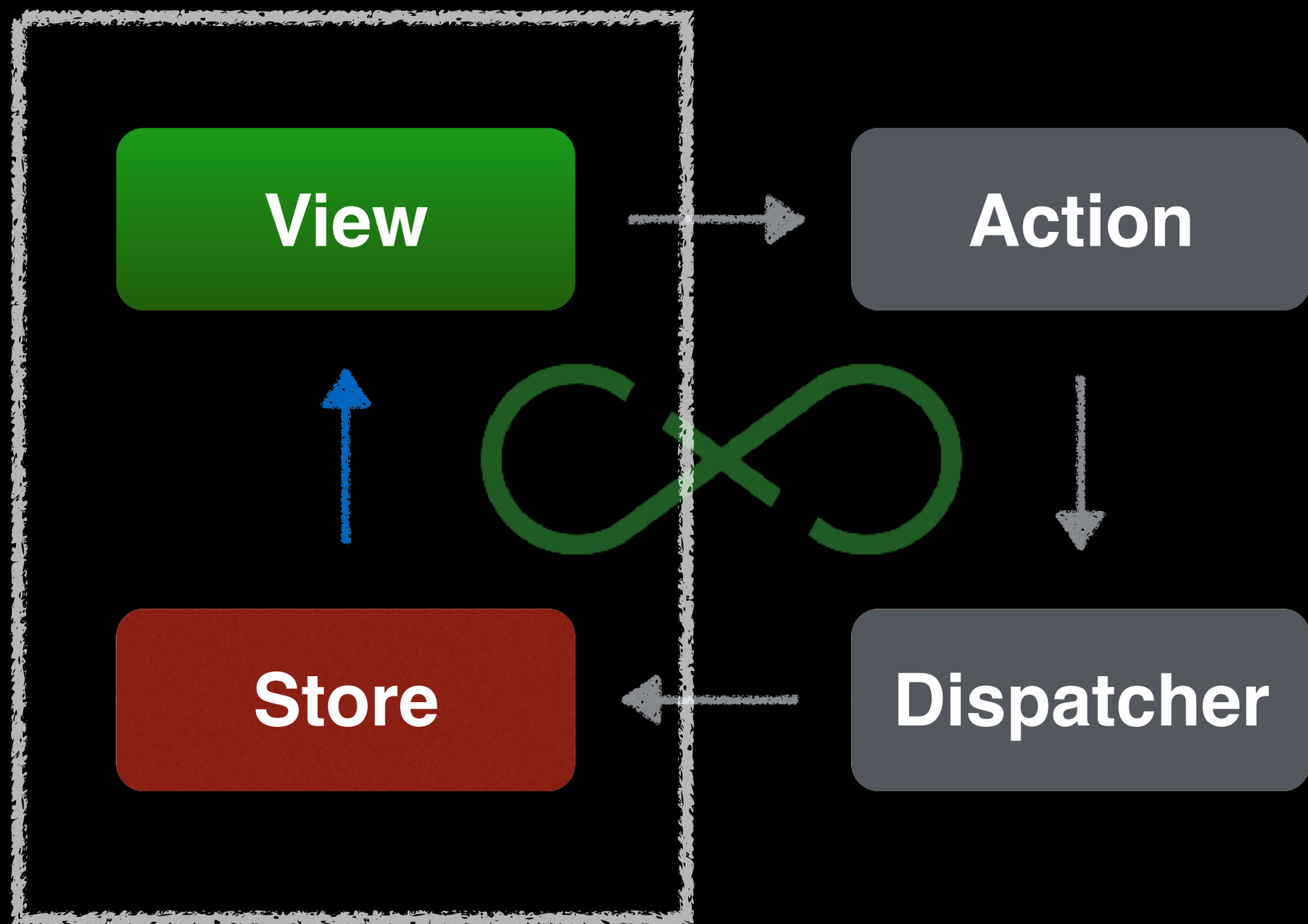
data/state/model?



credits: facebook.github.io/flux

scalable?


View
Action
Store
Dispatcher


View
Action
Store
Dispatcher

unidirectional

unidirectional

re-render

when store changes

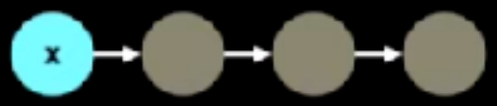
computation intensive?

Immutable!

Immutable?

get a new value
whenever make a change

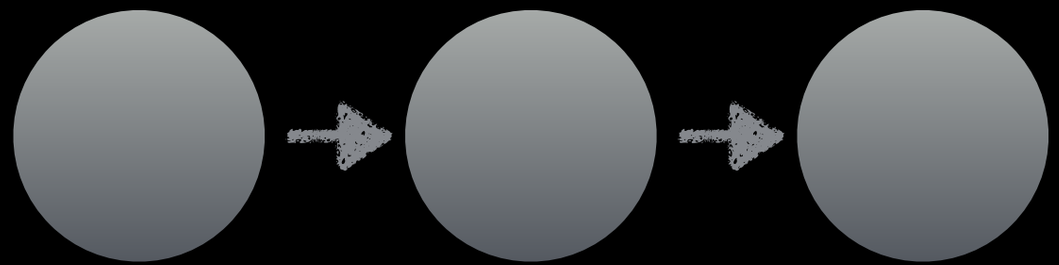
Simple example: Linked List



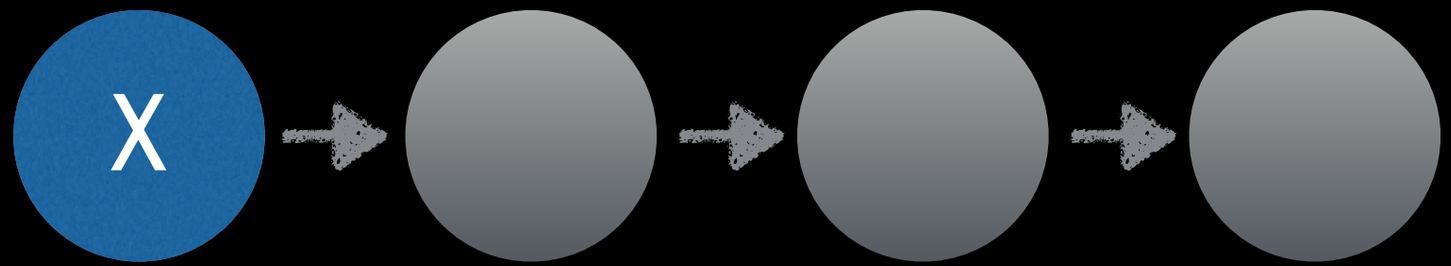
JSCONF 2014!

credits: David Nolen

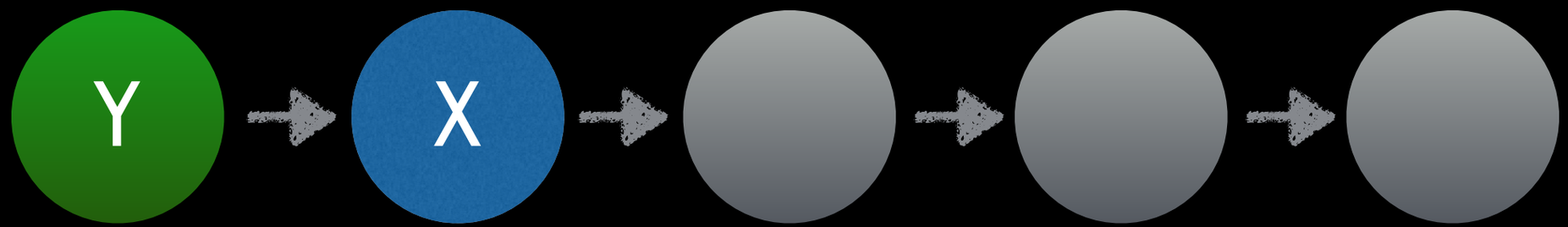
linked list



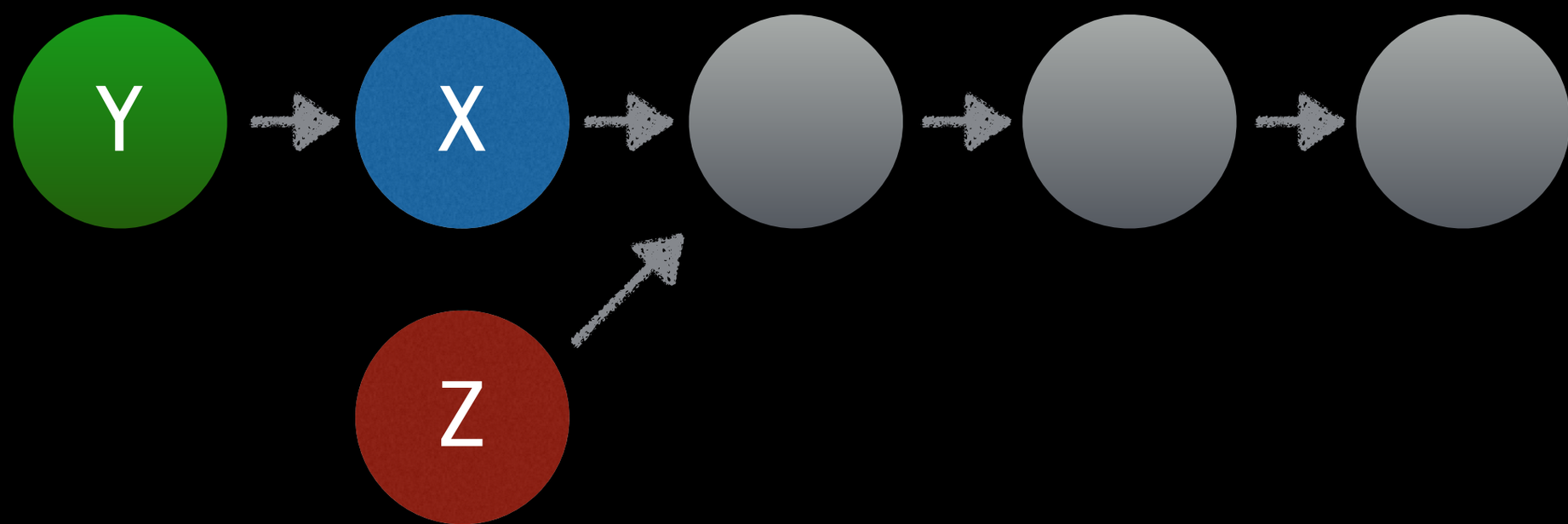
linked list



linked list

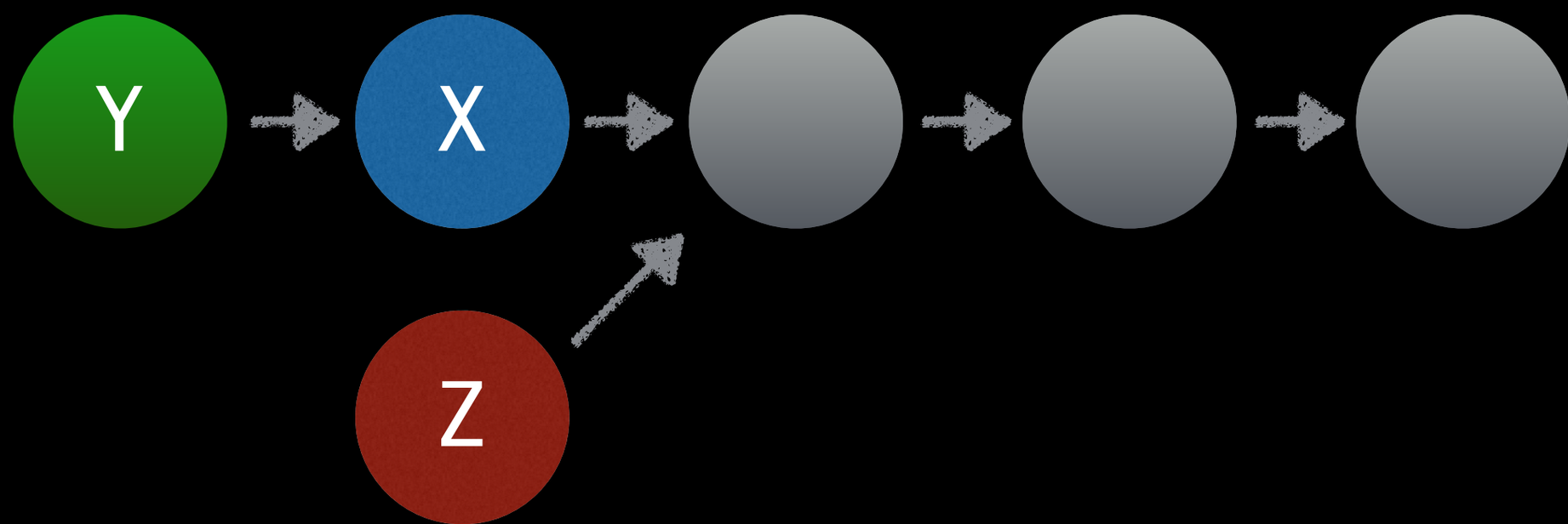


linked list



linked list

structure sharing



structure sharing

structure sharing

- space efficiency

structure sharing

- space efficiency
- computation efficiency

```
class TodoItem extends React.Component {
  shouldComponentUpdate(nextProps, nextState) {


  }
  render() {
    // ...
  }
}
```

```
class TodoItem extends React.Component {
  shouldComponentUpdate(nextProps, nextState) {
    var { content, isCompleted } = this.props;


  }
  render() {
    // ...
  }
}
```

```
class TodoItem extends React.Component {
  shouldComponentUpdate(nextProps, nextState) {
    var { content, isCompleted } = this.props;
    return (
      content !== nextProps.content &&
      isCompleted !== nextProps.isCompleted
    );
  }
  render() {
    // ...
  }
}
```

```
class TodoItem extends React.Component {
  shouldComponentUpdate(nextProps, nextState) {
    var { content, isCompleted } = this.props;
    return (
      this.props !== nextProps

    );
  }
  render() {
    // ...
  }
}
```

```
class TodoItem extends React.Component {
  shouldComponentUpdate(nextProps, nextState) {
    var { content, isCompleted } = this.props;
    return (
      this.prop        nextPro

    );
  }
  render() {
    // ...
  }
}
```

```
class TodoItem extends React.Component {
  shouldComponentUpdate(nextProps, nextState) {
    var { content, isCompleted } = this.props;
    return (
      this.prop        nextPro

    );
```

pros / nextProps are mutable and we can’t simply compare it’s reference

```
}
```

```
var Immutable = require('immutable');
```

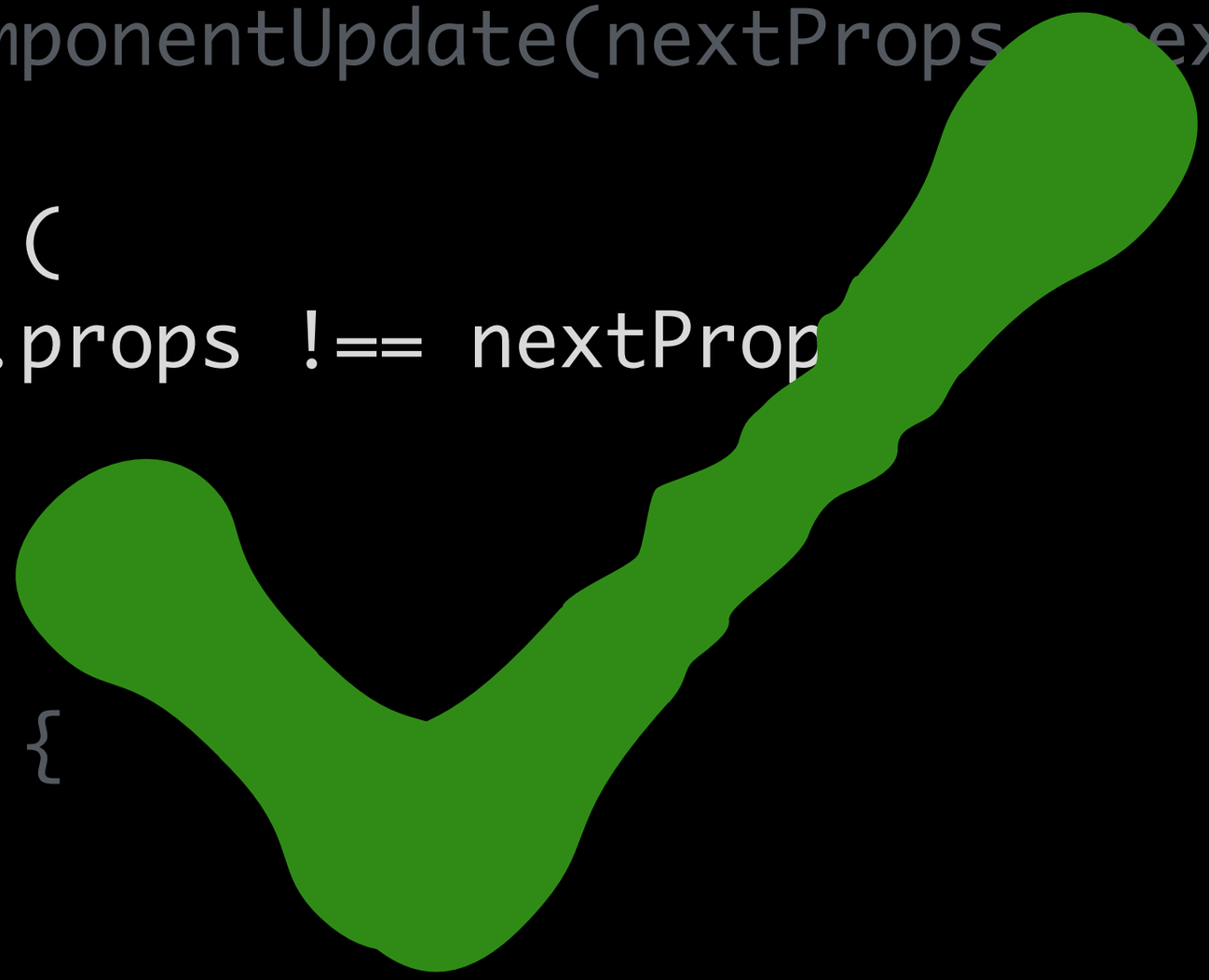
```
var Immutable = require('immutable');
var todo = Immutable.Map({
  content: 'say hello',
  isCompleted: false
});
```

```
var Immutable = require('immutable');
var todo = Immutable.Map({
  content: 'say hello',
  isCompleted: false
});

var mutatedTodo = todo.set(
  “isCompleted”, true
);
```

```
var Immutable = require('immutable');
var todo = Immutable.Map({
  content: 'say hello',
  isCompleted: false
});

var mutatedTodo = todo.set(
  “isCompleted”, true
);
if (todo !== mutatedTodo) {
  console.log('changed!');
  // > changed!
}
```

```
class TodoItem extends React.Component {
  shouldComponentUpdate(nextProps, nextState) {

    return (
      this.props !== nextProps

    );
  }
  render() {
    // ...
  }
}
```

```
class TodoItem extends React.Component {
  shouldComponentUpdate(nextProps, nextState) {

    return (
      this.props !== nextProp

    );
  }
  render() {
    // ...
  }
}
```



recap

recap

- UI = f(state)

recap

- UI = f(state)
- Predictable

recap

- UI = f(state)
- Predictable
- Immutable

<Questions />