

Introduction to Redux

@axross

React.js meetup #2 on Sep 9 2015

Hello :)

- @axross / Kohei Asai
- Working in  Gunosy Inc.
- #javascript, #nodejs, #reactjs

I developing...

- ・ グノシー内で漫画が読めるコンテンツ
 - ・ 「某少年の事件簿」
 - ・ 「某カンタービレ」
- ・ npm i -S react redux react-router bemmer
- ・ よろしくお願ひします！

Redux

What is Redux?

The screenshot shows the npmjs.com search results for the term "redux". At the top left is the NPM logo. To its right is a search bar containing the text "find packages". Below the search bar, the first result is for the "redux" package. It features a star icon, the word "redux" in large bold letters, and a "public" badge. A red rectangular box highlights the package name "redux". Below the package name is the description: "Predictable state container for JavaScript apps". Another red rectangular box highlights this description. At the bottom of the screenshot, there is a truncated text line: "It helps you write applications that behave consistently, run in different environments (web, mobile, server), and be testable in isolation."

<https://www.npmjs.com/package/redux>

What is Redux?



find packages

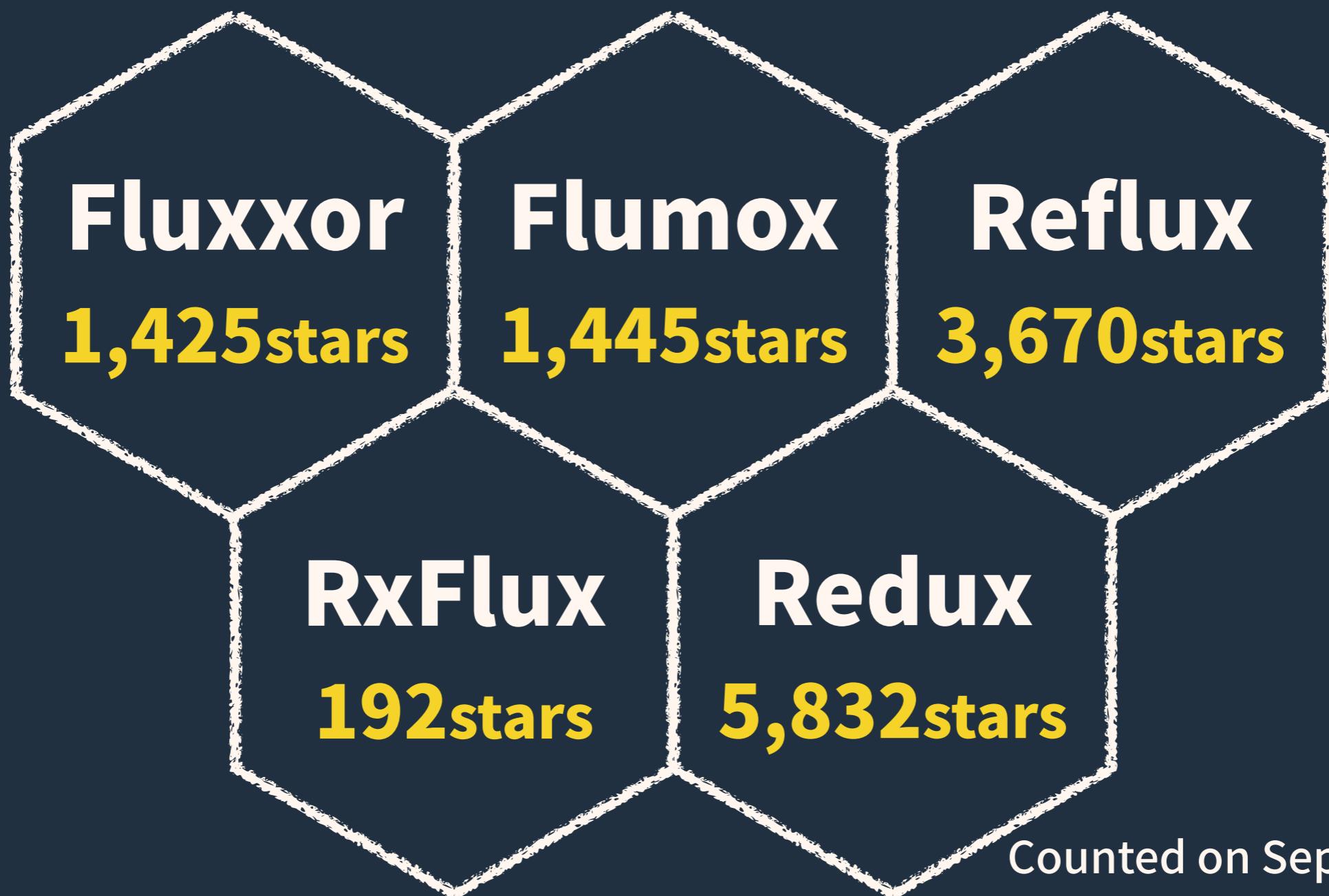
"JavaScriptアプリケーションのための
革新性のない状態コンテナ"

Predictable state container for JavaScript apps

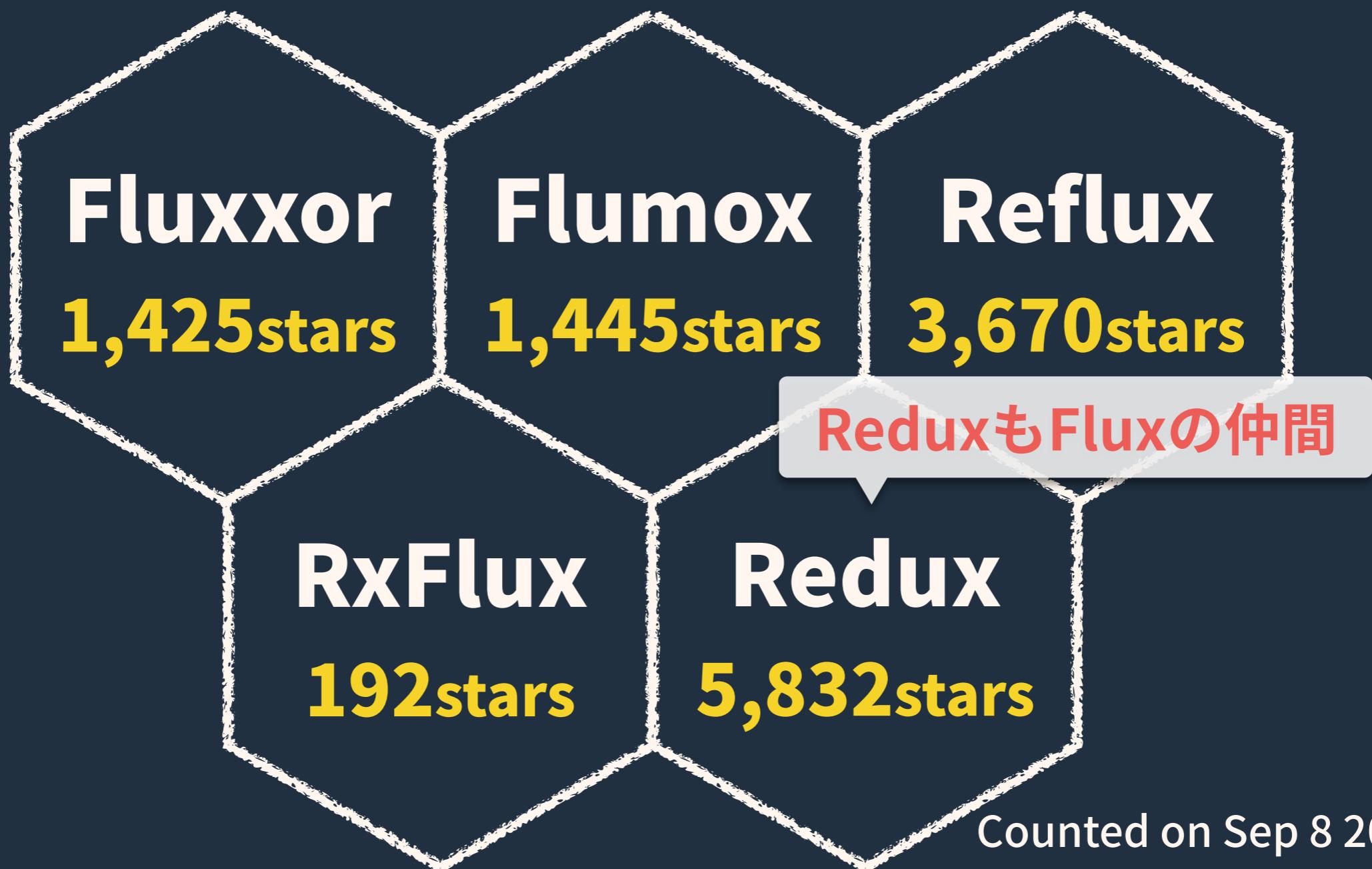
It helps you write applications that behave consistently, run in different environments (Node.js and the browser), and are testable and debuggable.

<https://www.npmjs.com/package/redux>

Family of Flux



Family of Flux



How did you know Redux?

📄 README.md

Flummox

- ☞ **4.0 will likely be the last major release. Use [Redux instead](#). It's really great.**

I know churn can be frustrating but I feel it would be irresponsible for me to continue recommending Flummox when Redux exists and is a significant improvement over classical Flux. Flummox 4.0 will likely be the last major release.

Check out [redux-actions](#) and [redux-promise](#), which implement much of the convenience of Flummox as Redux extensions.

<https://github.com/acdlite/flummox>

How did you know Redux?

README.md

v4.0は最後のメジャーリリースに
Flummox なるだろう。

4.0 will likely be the last major release. Use **Redux** instead. It's
really great.

代わりにReduxを使ってくれ。
あれはマジでグレートだ。

I know churn can be frustrating but I feel it would be irresponsible for me to continue recommending Flummox when Redux exists and is a significant improvement over classical Flux. Flummox 4.0 will likely be the last major release.

Check out **redux-actions** and **redux-promise**, which implement much of the convenience of Flummox as Redux extensions.

<https://github.com/acdlite/flummox>

Redux is...

- Flummoxの作者が手放して賞賛
 - いまやReduxの周辺作ったりしてる
- データフローはFlummoxに似てる
- Flummoxではサポートしていなかった、「状態の管理」に重きを置いている

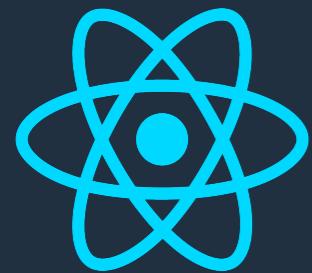
Motivation

SPA get more States

- JavaScriptのSPAを作るのが当たり前になり、凝ったつくりを求められることが多くなった
 - より多くの状態の管理が必要になる

too Hard to managing states

- ・ 絶えず変化する状態を管理するのは難しい
- ・ いずれ手に負えなくなる
 - ・ バグの再現がすぐにできなくなる
 - ・ 機能追加の際に影響範囲が認識しにくい



is great, but...

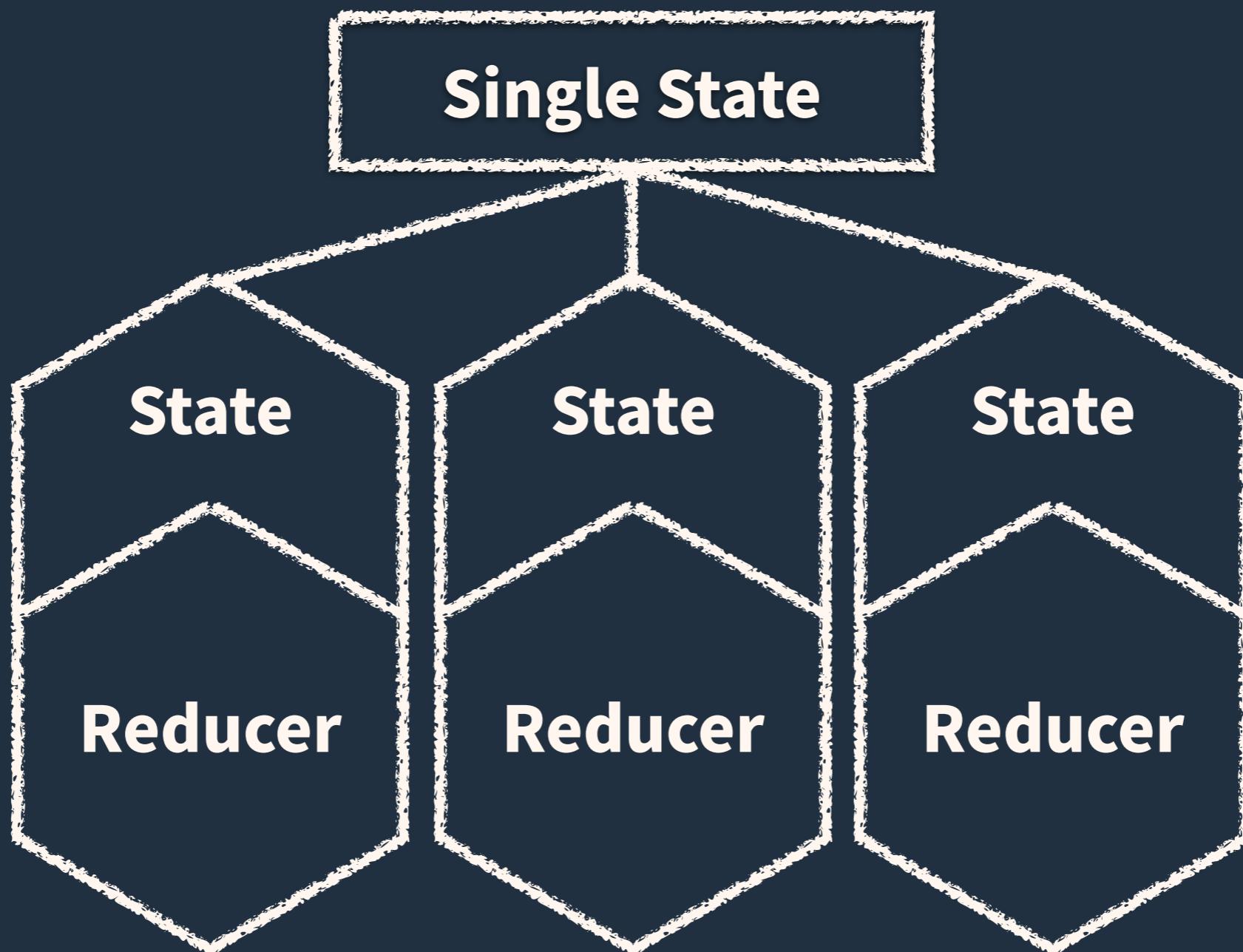
- ReactでDOMと状態は切り離された
 - Componentは値によって参照透過にDOMを操作する
- しかし、その状態の「管理」は手付かず

We are still on one's way Flux

- ・ 「状態の管理」をFluxというフローに当てはめて、僕らは解決しようとした
- ・しかし、まだ「これだ」というものは出てきていなかった
- ・ Reduxはこの「状態の管理」を一番うまくできるFlux

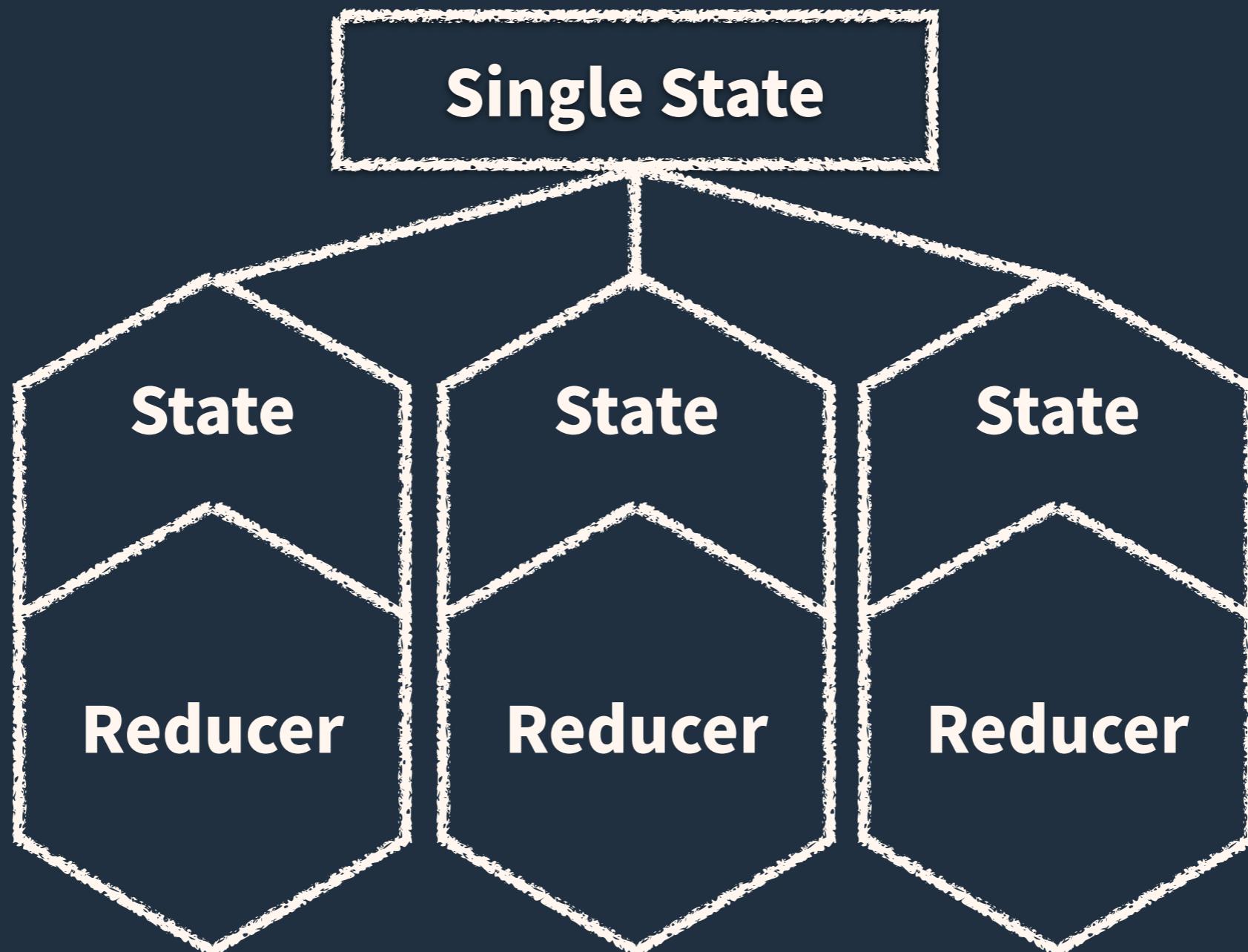
Data Flow

Single State Tree

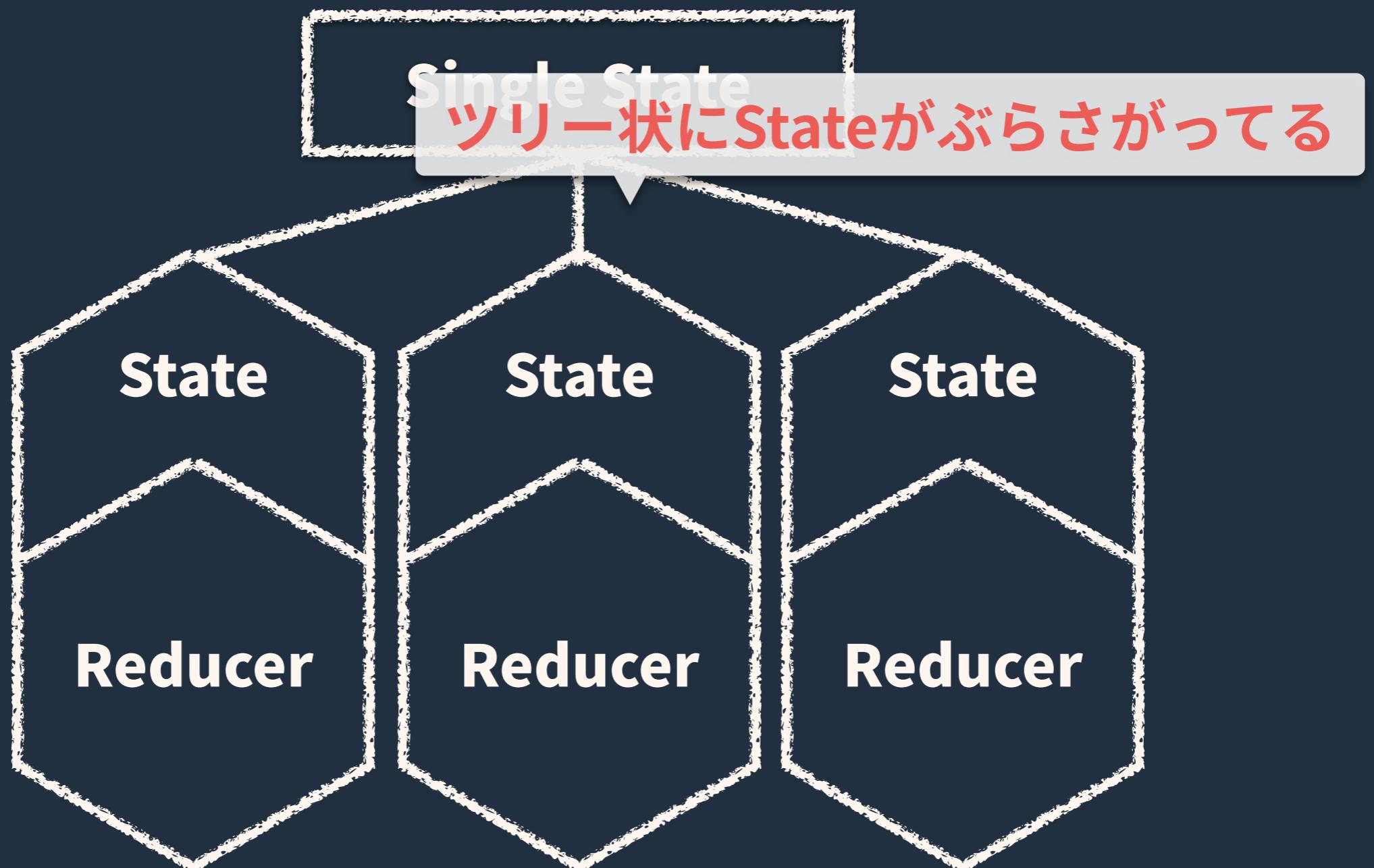


Single State Tree

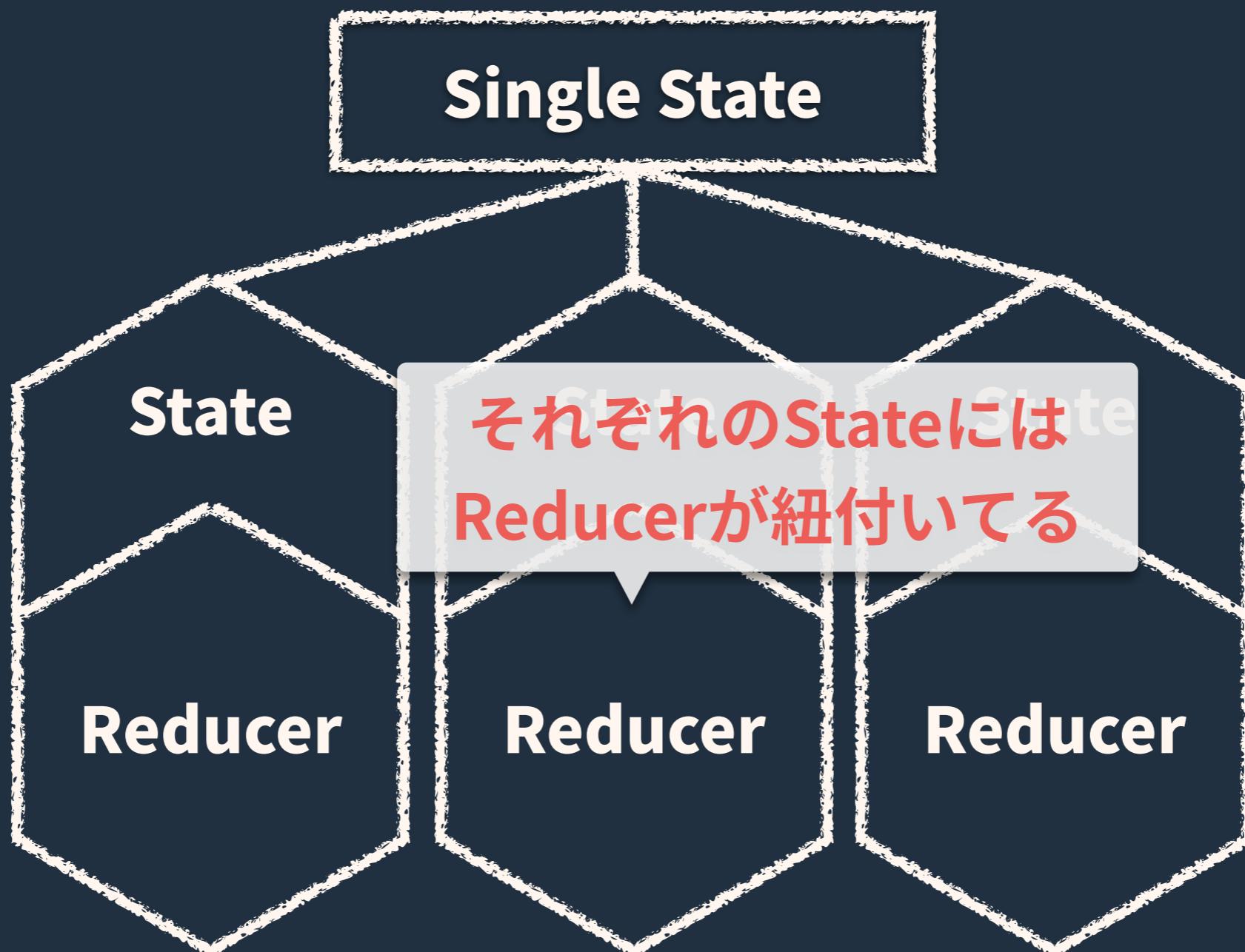
Reduxは1つのStateのルート(根)がある



Single State Tree



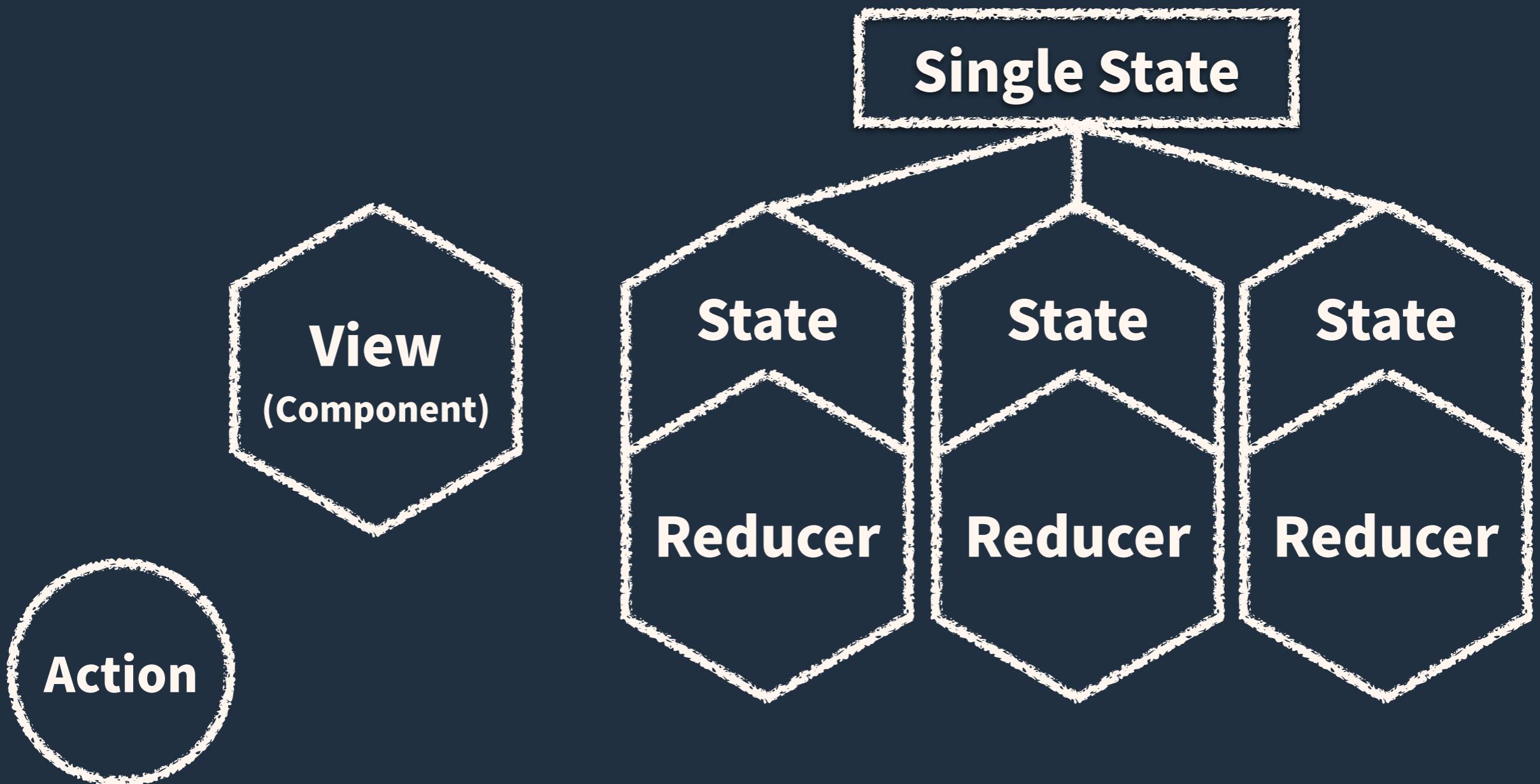
Single State Tree



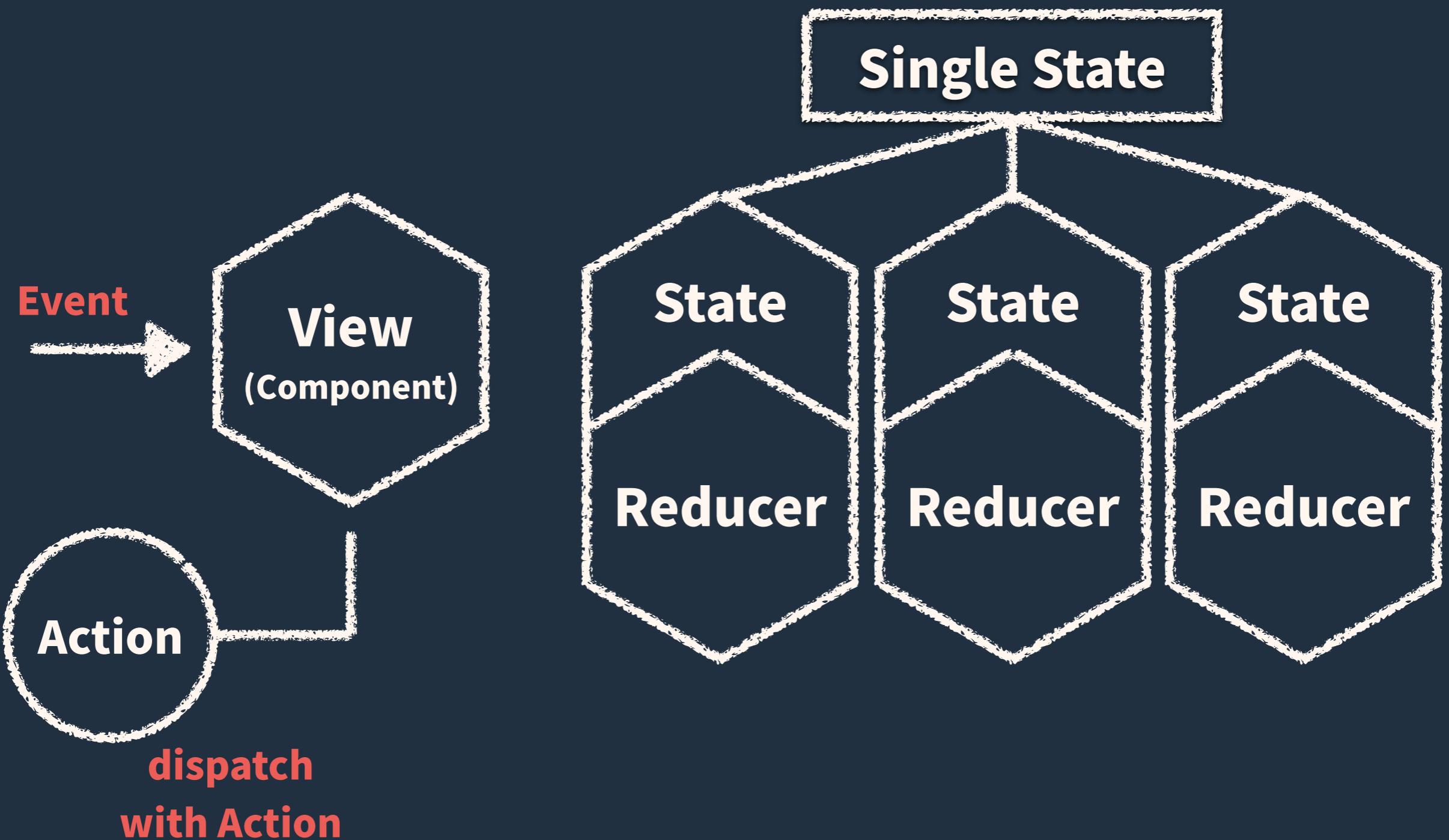
Reducer

- Stateフィルター/マッパー
- Reducerは
 - 自分のStateを変更するかどうか
 - Stateをどう更新するか、を担う
- Reducer 자체はStateではない

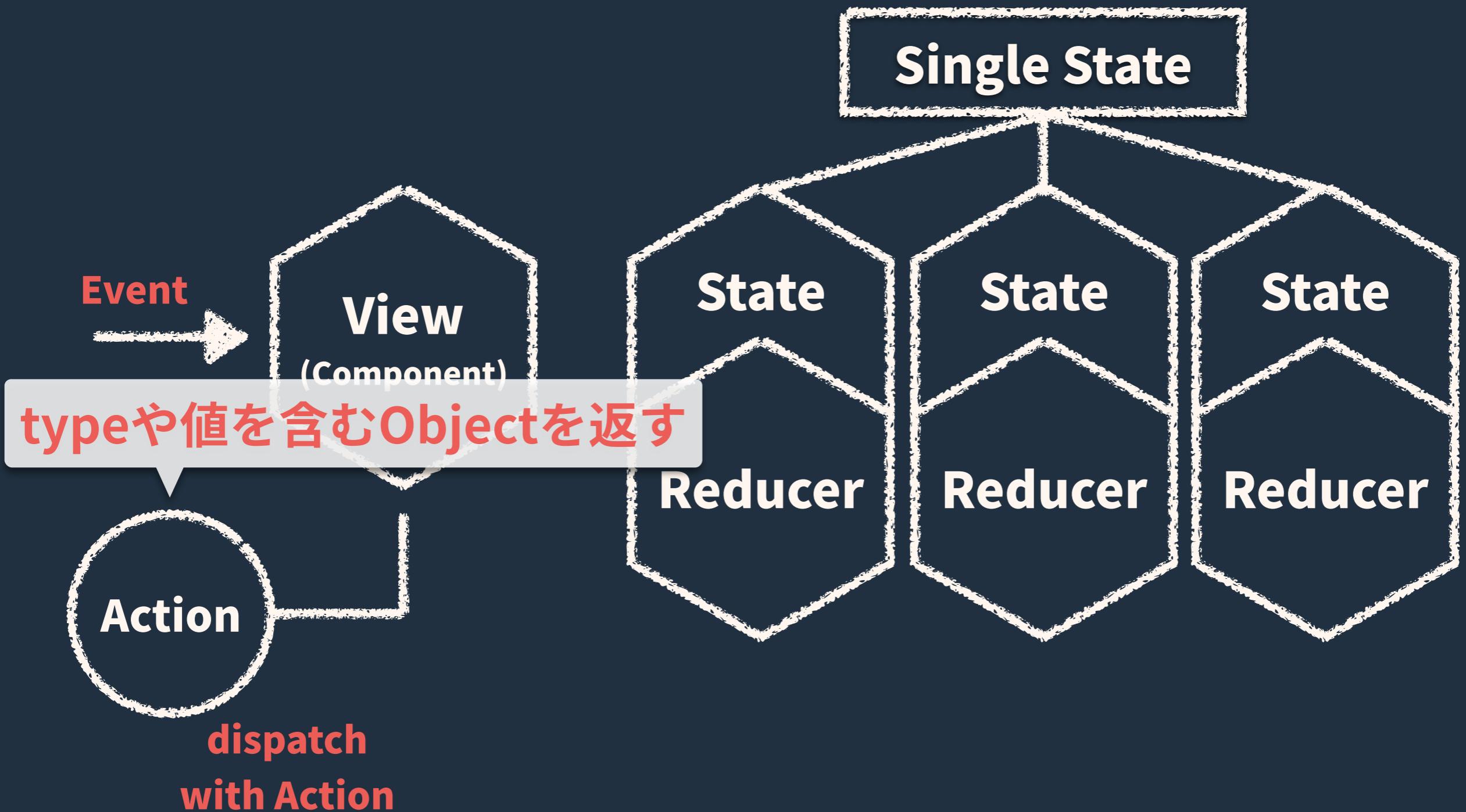
Data Flow



Data Flow



Data Flow



Like this:

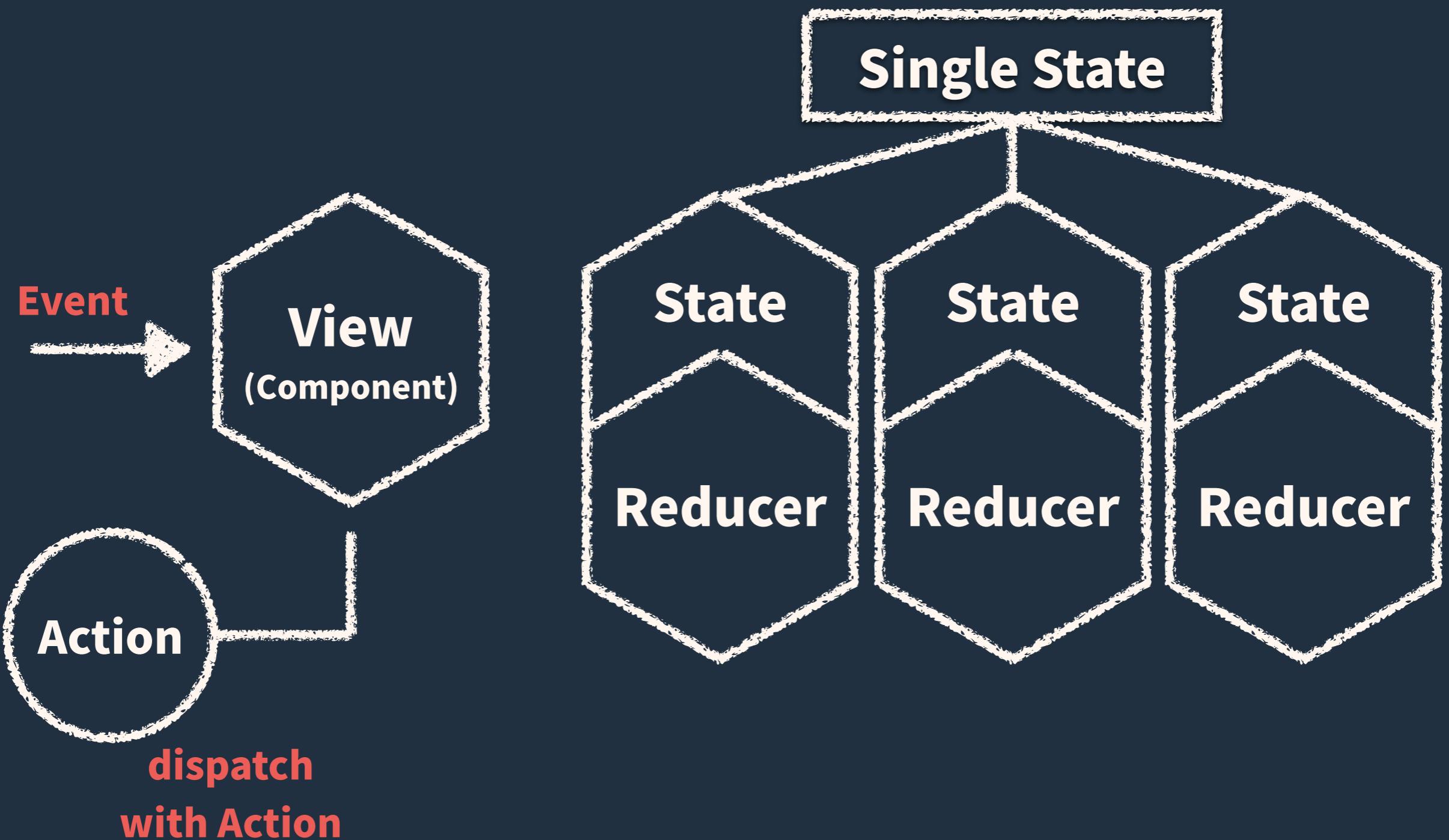
```
export const addTodo = body => {
  return {
    type: 'TODO_ADD',
    todo: new Todo(body),
  };
};
```

Like this:

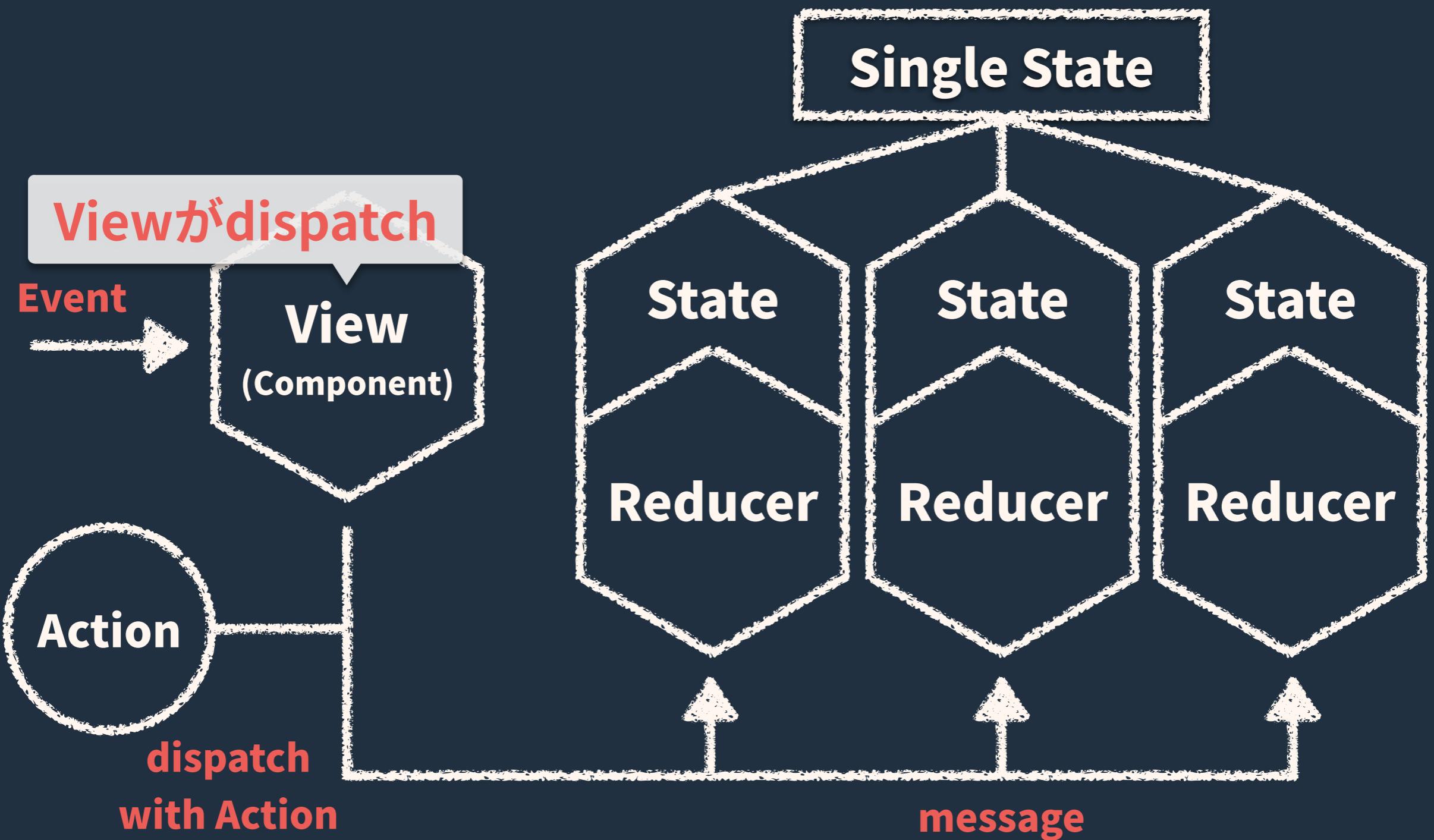
```
export const addTodo = body => {
  return {
    type: 'TODO_ADD',
    todo: new Todo(body),
  };
};
```

↑
typeと値を返すだけの単純な関数

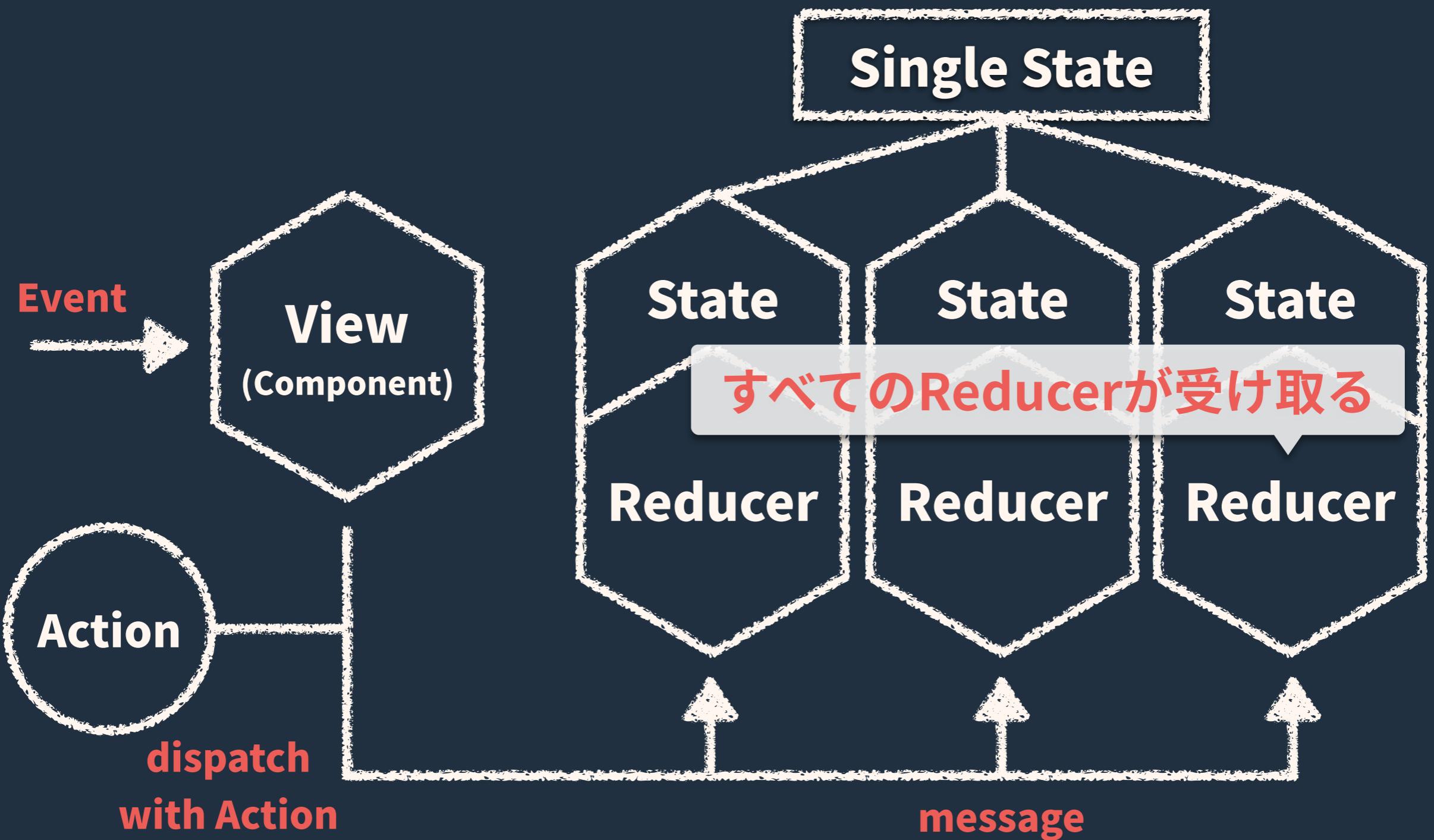
Data Flow



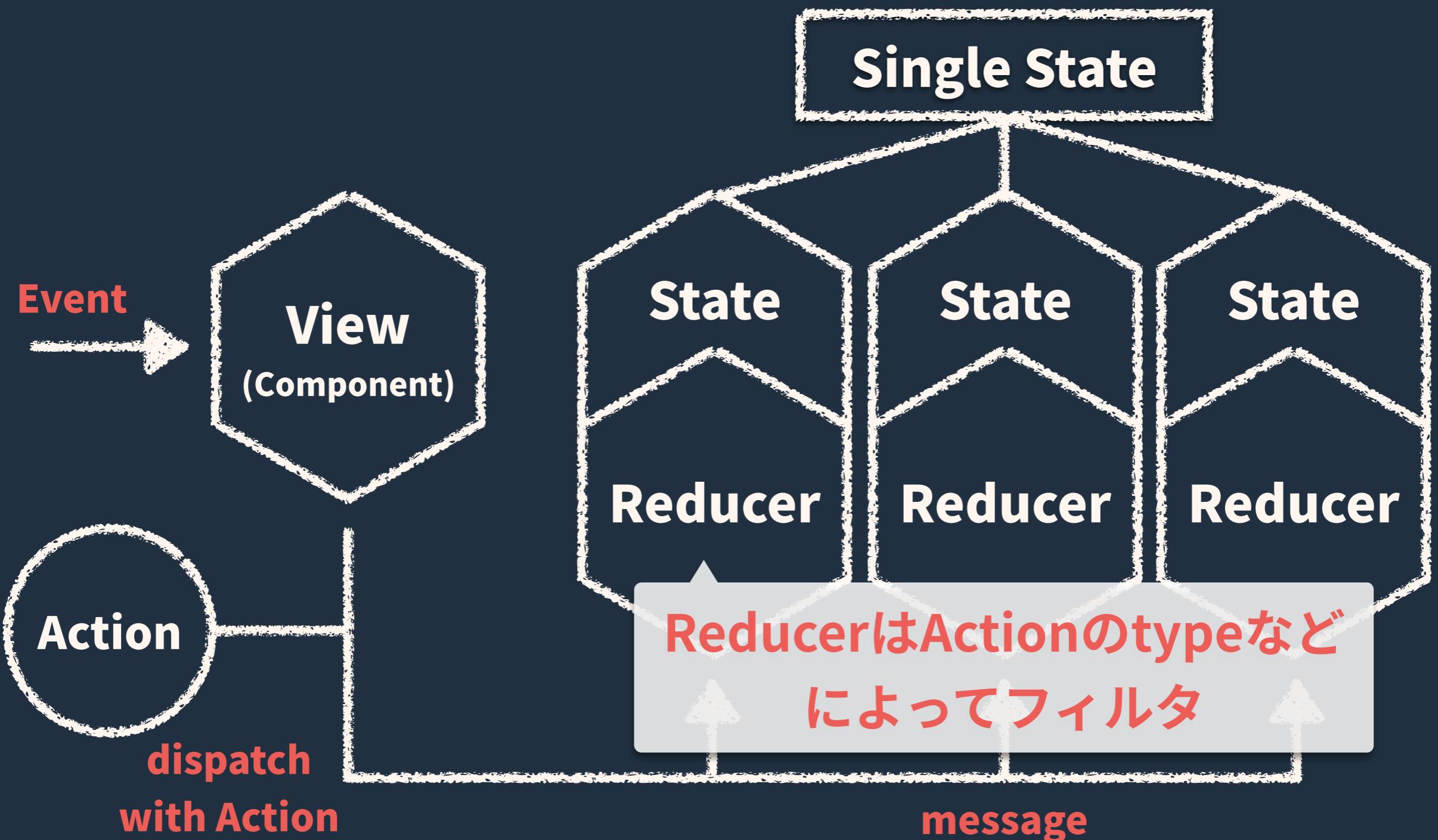
Data Flow



Data Flow



Data Flow



Like this:

```
const initialState = { items: [] };

export const todo = (state = initialState, action) {
  switch (action.type) {
    case 'TODO_ADD':
      return {
        items: state.items.concat(action.todo),
      };
    default:
      return state;
  }
};
```

Like this:

```
const initialState = { items: [] }

Reducer、メッセージを受け取ると呼ばれる

export const todo = (state = initialState, action) {
  switch (action.type) {
    case 'TODO_ADD':
      return {
        ...state,
        items: state.items.concat(action.todo),
      };
    default:
      return state;
  }
};
```

戻り値によって「更新後の値」を決める

引数と同じ参照を返すと「変更なし」

Like this:

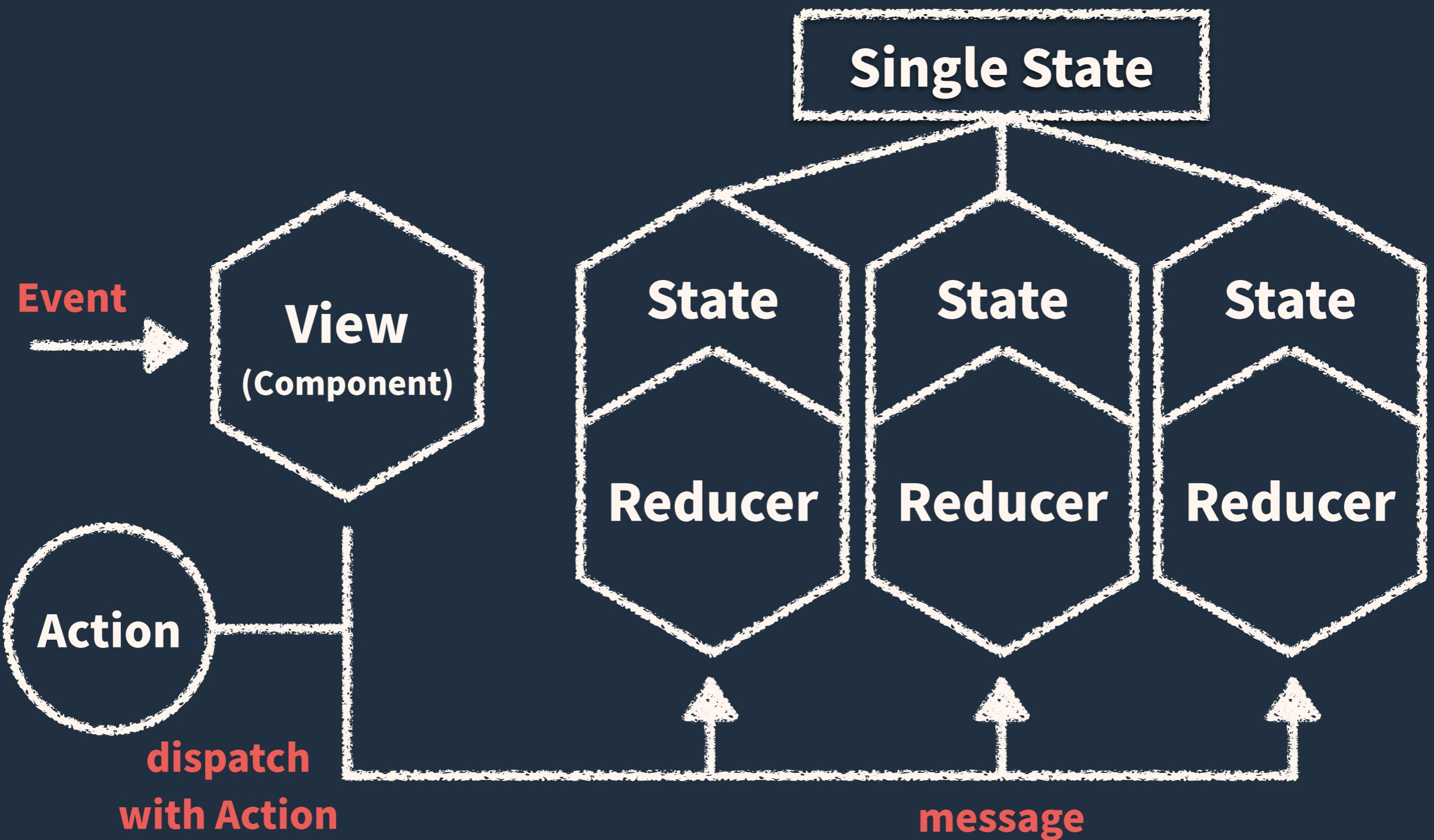
```
const initialState = { items: [] }; Stateの初期状態
export const todo = (state = initialState, action) {
  switch (action.type) {
    case 'TODO_ADD':
      return {
        items: state.items.concat(action.todo),
      };
    default:
      return state;
  }
};
```

Like this:

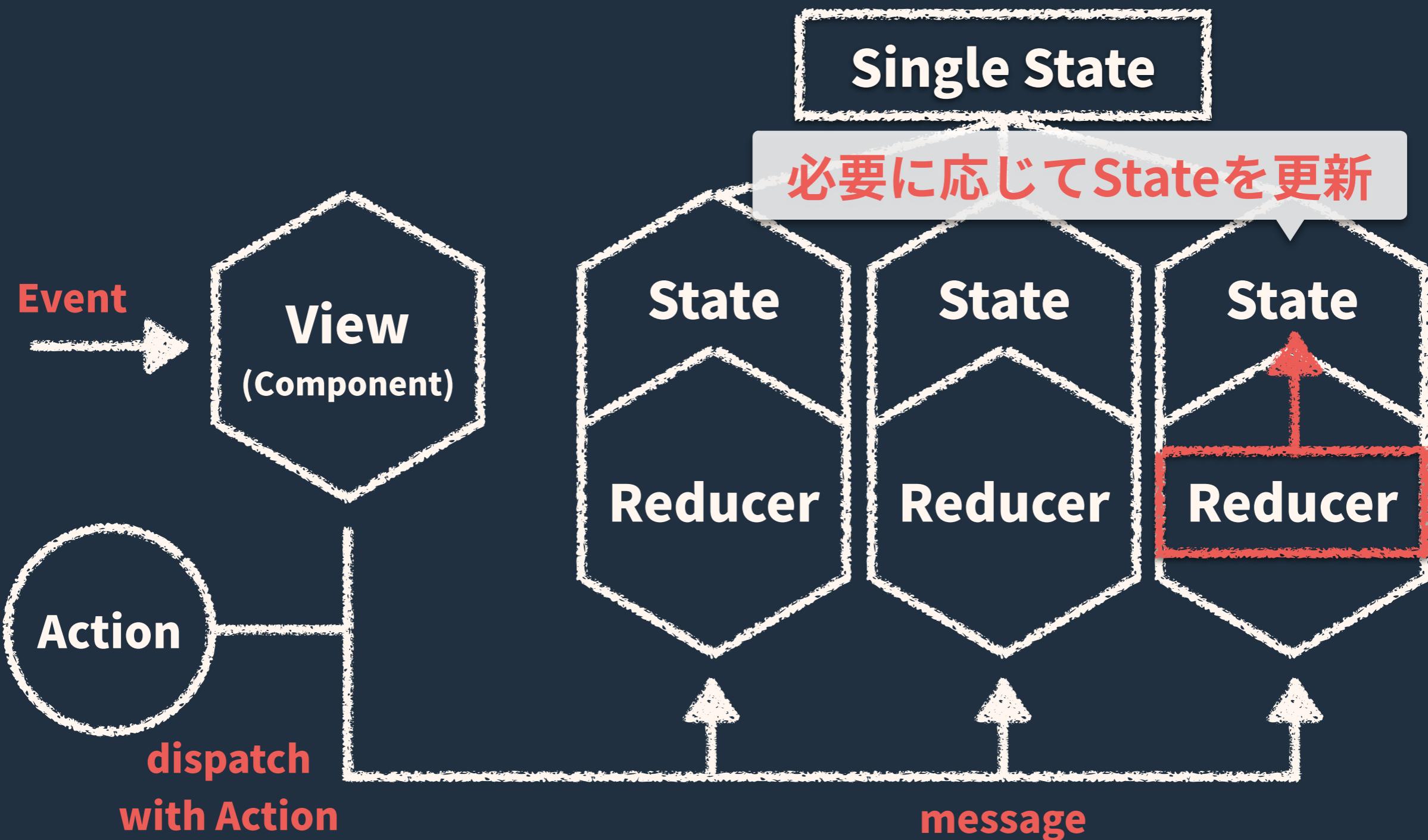
```
const initialState = { items: [] };

Actionをフィルタリング
export const todo = (state = initialState, action) {
  switch (action.type) {
    case 'TODO_ADD':
      return {
        items: state.items.concat(action.todo),
      };
    default:
      return state;
  }
};
```

Data Flow

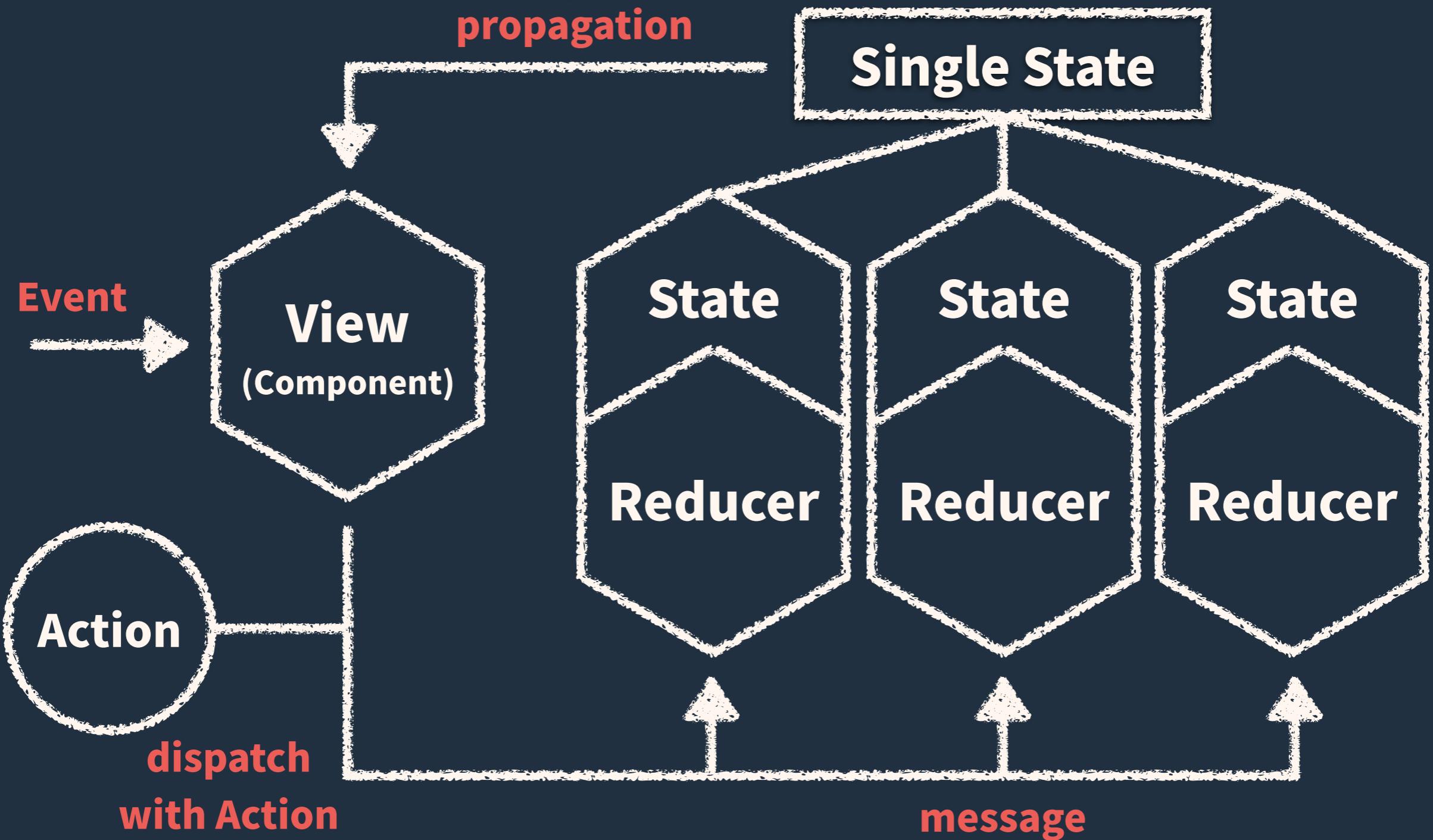


Data Flow

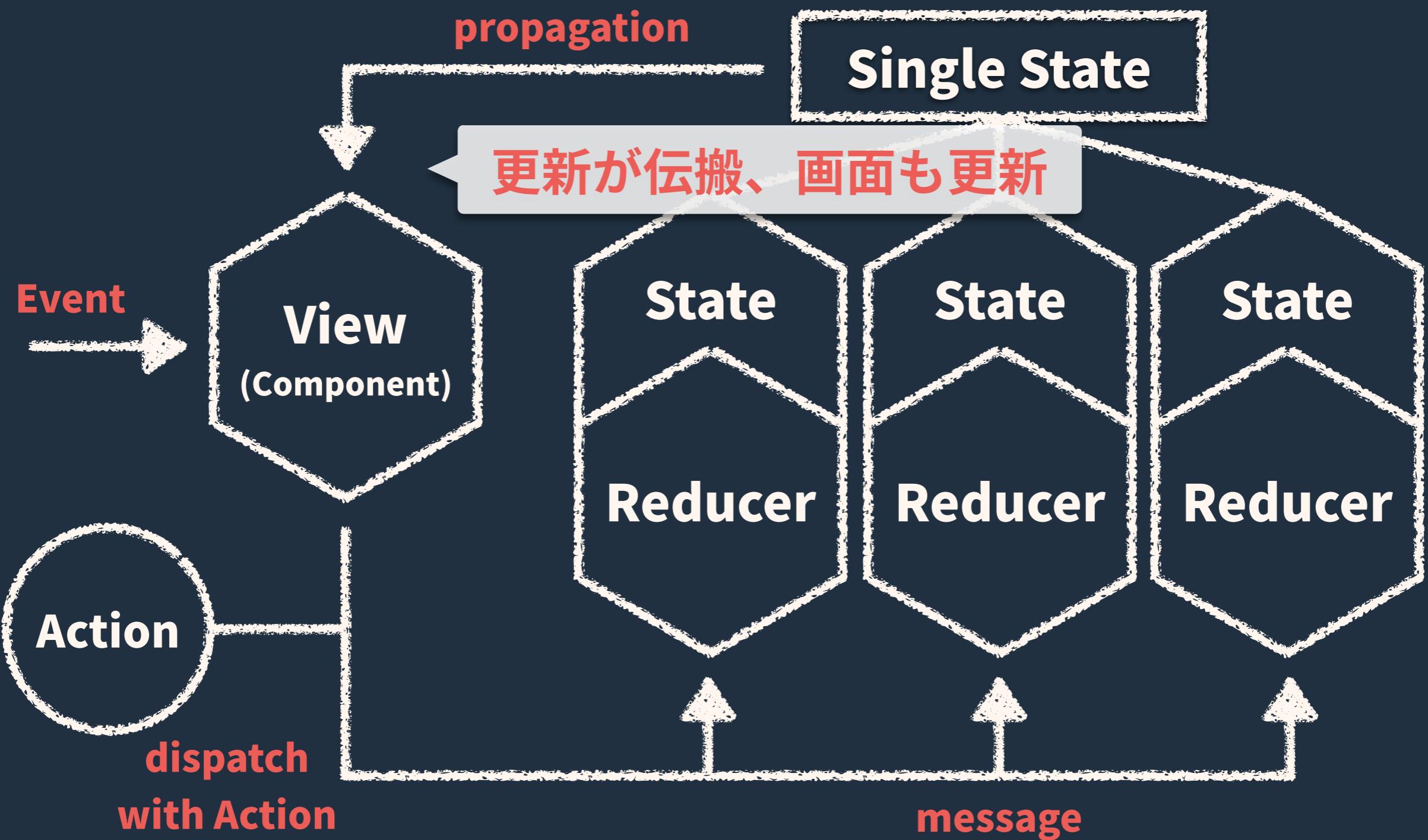


Data Flow

ViewはStateを購読する



Data Flow

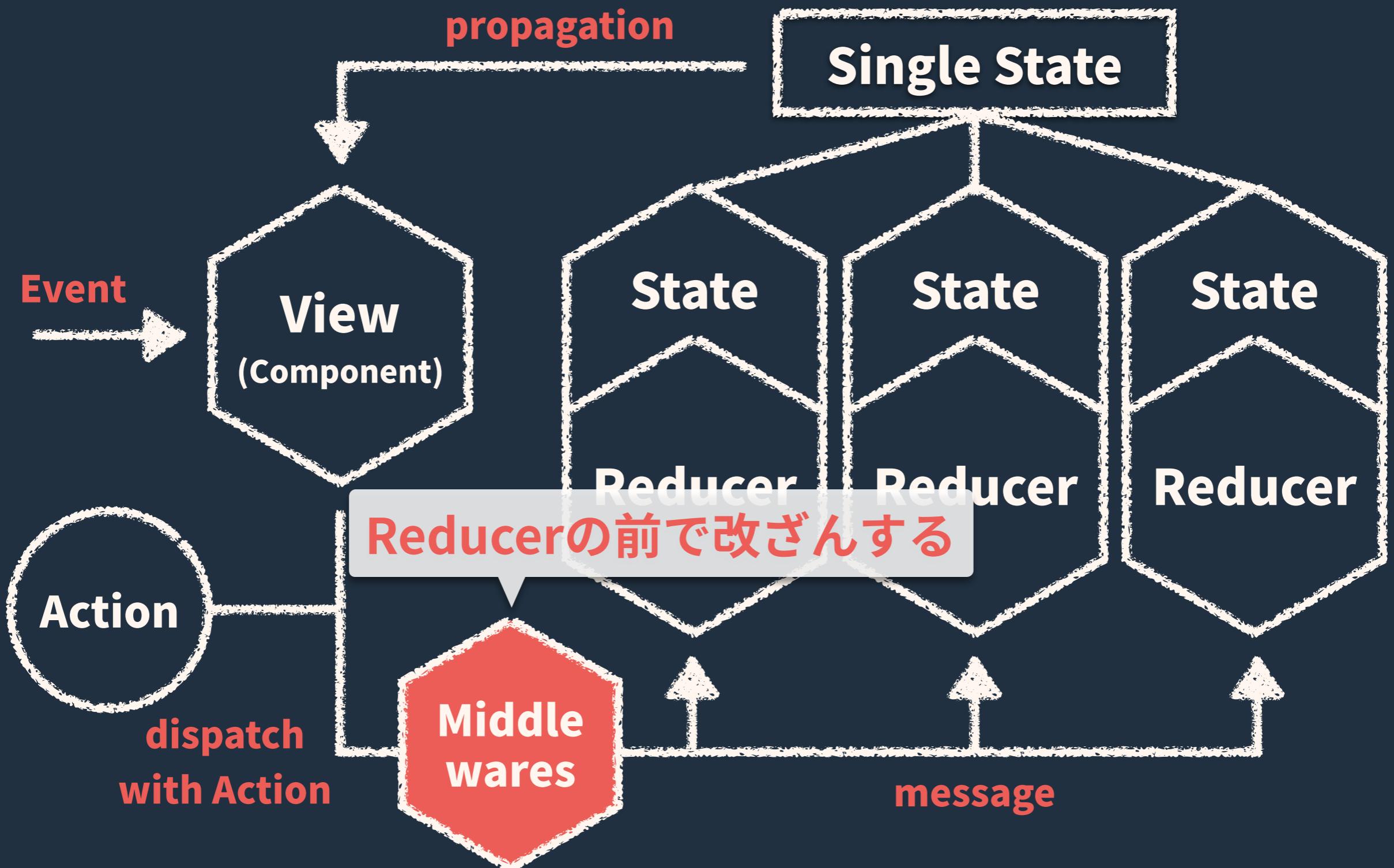


Middlewares

Middlewares

- ・ メッセージを伝搬途中で改ざんする機能
- ・ 複数登録可
- ・ イメージはNode.jsのConnectのそれに近い

Where are Middlewares?



Examples of Middlewares

- たとえばこんなことができる
 - 流れてくるメッセージをロギング
 - "redux-devtools"とか
 - ActionやReducerからCallbackを排除する
 - Google AnalyticsにEvent通知する ...etc

Logging each message

```
export const loggingMessage = store => {
  return next => {
    return action => {
      console.group(action.type);
      console.info(action);
      next(action);
      console.groupEnd(action.type);
    };
  };
};
```

本当はもうちょっと簡潔に書けるけど…

Logging each message

```
export const loggingMessage = store => {
  return next => {
    流れてきたメッセージをそのままconsoleに
    return action => {
      console.group(action.type);
      console.info(action);
      next(action);
      console.groupEnd(action.type);
    };
  };
};
```

本当はもうちょっと簡潔に書けるけど…

Logging each message

```
export const loggingMessage = store => {
  return next => {
    next(); // すると次のMiddleware
    // またはReducerに流れる
    console.group(action.type);
    console.info(action);
    next(action);
    console.groupEnd(action.type);
  };
};
};
```

本当はもうちょっと簡潔に書けるけど…

Async convert to like a Sync

```
export const promiseSyncify = store => {
  return next => {
    return action => {
      if (!action.promise) return next(action);
      action.promise
        .then(value => next({ type: action.type, value }))
        .catch(err => {
          console.error(err);
          next({ type: action.type, value: null });
        });
    };
  };
};
```

Async convert to like a Sync

```
export const promiseSyncify = store => {
  return next = action => {
    if (!action.promise) return next(action);
    action.promise
      .then(value => next({ type: action.type, value }))
      .catch(err => {
        console.error(err);
        next({ type: action.type, value: null });
      });
  };
};
```

Async convert to like a Sync

```
export const promiseSyncify = store => {
  return next => {
    return action => {
      if (!action.promise) return next(action);
      action.promise
        .then(value => next({ type: action.type, value }))
        .catch(err => {
          console.error(err);
          next({ type: action.type, value: null });
        });
    };
  };
};
```

Promiseを解決する

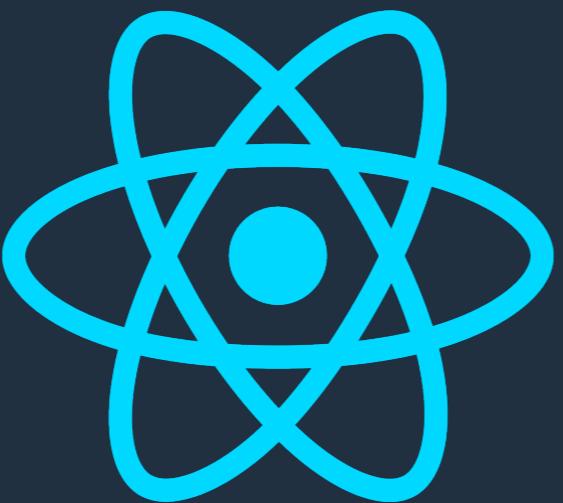
Async convert to like a Sync

```
export const promiseSyncify = store => {
  return next => {
    return action => {
      if (!action.promise) next(action);
      action.promise
        .then(value => next({ type: action.type, value }))
        .catch(err => {
          console.error(err);
          next({ type: action.type, value: null });
        });
    };
  };
};
```

各Reducerは流れてくる"value"を
同期的に処理すればよい

Using in
the Real World

with



with

- Redux自体はReactがなくても動く
 - Reduxはただの状態コンテナにすぎない
 - “ng-redux”とかもある
- Reactとの併用には"react-redux"が便利

react-redux

- Reactバインディングを提供する
 - connect()
 - this.props経由で、指定したStateや dispatch()が得られる
 - subscribeの自動化
 - ComponentのFlux非依存化

Like this:

```
export const TodoList = React.createClass({
  onAddClick() {
    this.props.dispatch(/* any action */);
  },
  render() {
    // can access to this.props.todos
  },
});
```



```
export default connect(state => {
  return { todos: state.todos.items };
});
```

Like this:

```
export const TodoList = React.createClass({  
  onAddClick() {  
    this.props.dispatch(/* any action */);  
  },  
  render() {  
    // can access to this.props.todos  
  },  
});
```

connect()は
import { connect } from 'react-redux';
で得られる

```
export default connect(state => {  
  return { todos: state.todos.items };  
});
```

Like this:

```
export const TodoList = React.createClass({  
  onClick() {  
    this.props.dispatch(/* any action */);  
  },  
  render() {  
    // can access to this.props.todos  
  },  
});
```

Single State Treeのルートが
コールバックで得られる

```
export default connect(state => {  
  return { todos: state.todos.items };  
});
```

Like this:

```
export const TodoList = React.createClass({
  onClick() {
    this.props.dispatch(/* any action */);
  },
  render() {
    // can access to this.props.todos
  },
});
```

**Componentで必要なStateを
戻り値で指定する**

```
export default connect(state => {
  return { todos: state.todos.items };
});
```

Like this:

```
export const TodoList = React.createClass({
  onClick() {
    this.props.dispatch(/* any action */);
  },
  render() {
    // can access to this.props.todos
  },
});
```

Stateはthis.props経由で得られる

```
export default connect(state => {
  return { todos: state.todos.items };
});
```

Like this:

```
export const TodoList = React.createClass({  
  connect()すると自動的にdispatch()も得られる  
  onAddClick() {  
    this.props.dispatch(/* any action */);  
  },  
  render() {  
    // can access to this.props.todos  
  },  
});  
  
export default connect(state => {  
  return { todos: state.todos.items };  
});
```

Like this:

```
export const TodoList = React.createClass({  
  onAddClick() {  
    this.props.dispatch({ type: 'ADD_TODO' });  
  },  
  render() {  
    // can access to this.props.todos  
  },  
});
```

分けてexportすれば、元のComponentはpropsまでしか参照していないないので

テストも楽

```
export default connect(state => {  
  return { todos: state.todos.items };  
});
```

with React Router

with React Router

- ・ 併用にはreact-router@1.0.0-beta2以降を
 - ・ <Router> Componentが欲しいため
- ・ 詳しく話すと長くなる上にコアな話なので、必要なら懇親会で聞いてください
- ・ または @axross_ on Twitterまで

Conclusion

Conclusion

- ・ 現状、我々の最もっとも大きな悩みは状態管理
 - ・ 複雑さは上昇傾向にある
 - ・ React/Fluxで100%解決されてはいない
 - ・ そろそろ状態管理について向き合おう
- ・ Reduxは「状態管理」に重きを置いたFlux

Conclusion

- Single State Tree
 - 1つの状態ツリーでアプリケーション全体を
 - モックを注入し、即座に再現が可能

Conclusion

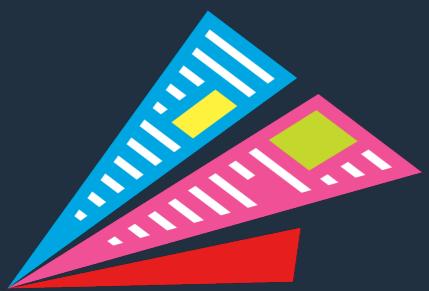
- Data Flow
 - Actionは本当に「ただの関数」
 - Reducerも単純なフィルター/マッパー
 - どこで値が生成され、どのように流れていったのか、容易に辿ることができる

Conclusion

- Middlewaresがある
 - すべてのメッセージの改ざんが可能
 - ロギング
 - 非同期を同期処理っぽく扱う
 - 認証が済んでるかどうかの確認なども

**Redux is
really great!**

Thank you for
listening :)



want
Front-end
Engineers!