
DataArt

DataArt

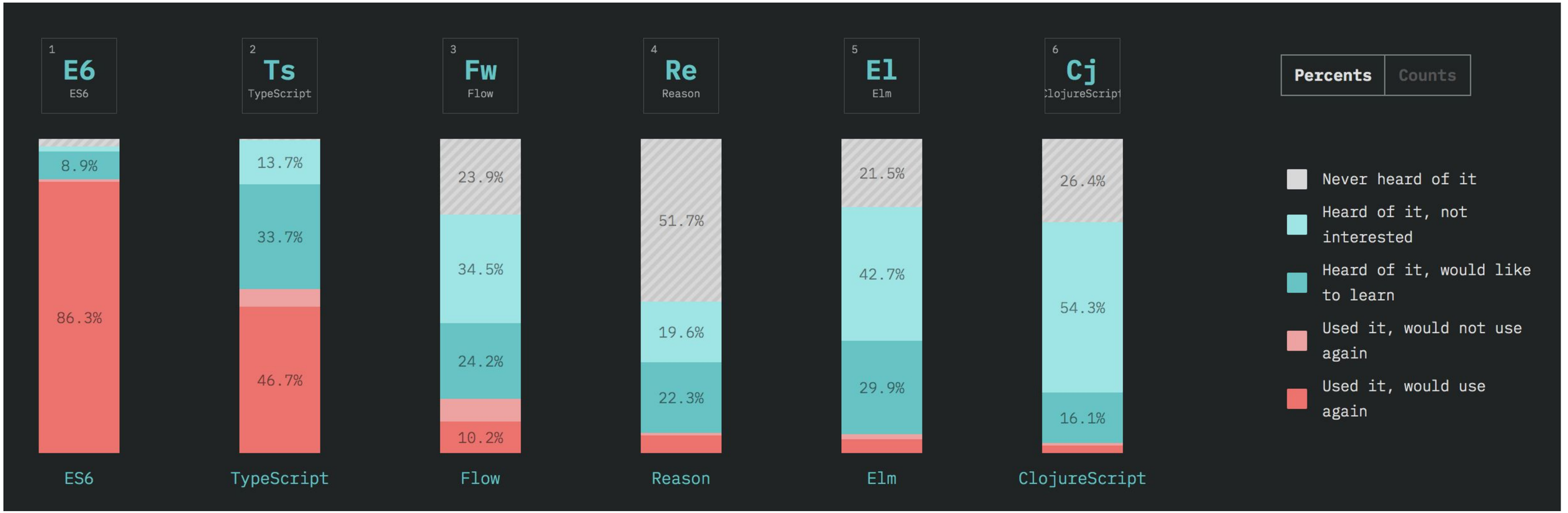
React

Roman Enikeev
Development Lead

Retrospective

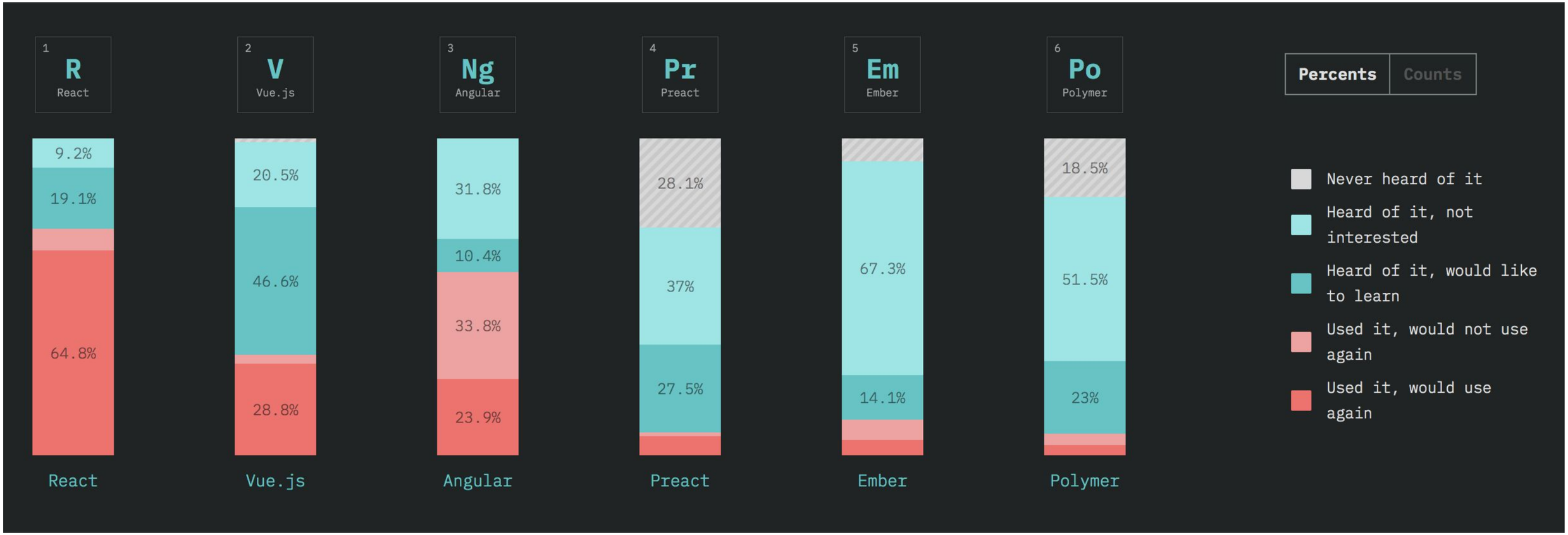
State of JS

DataArt



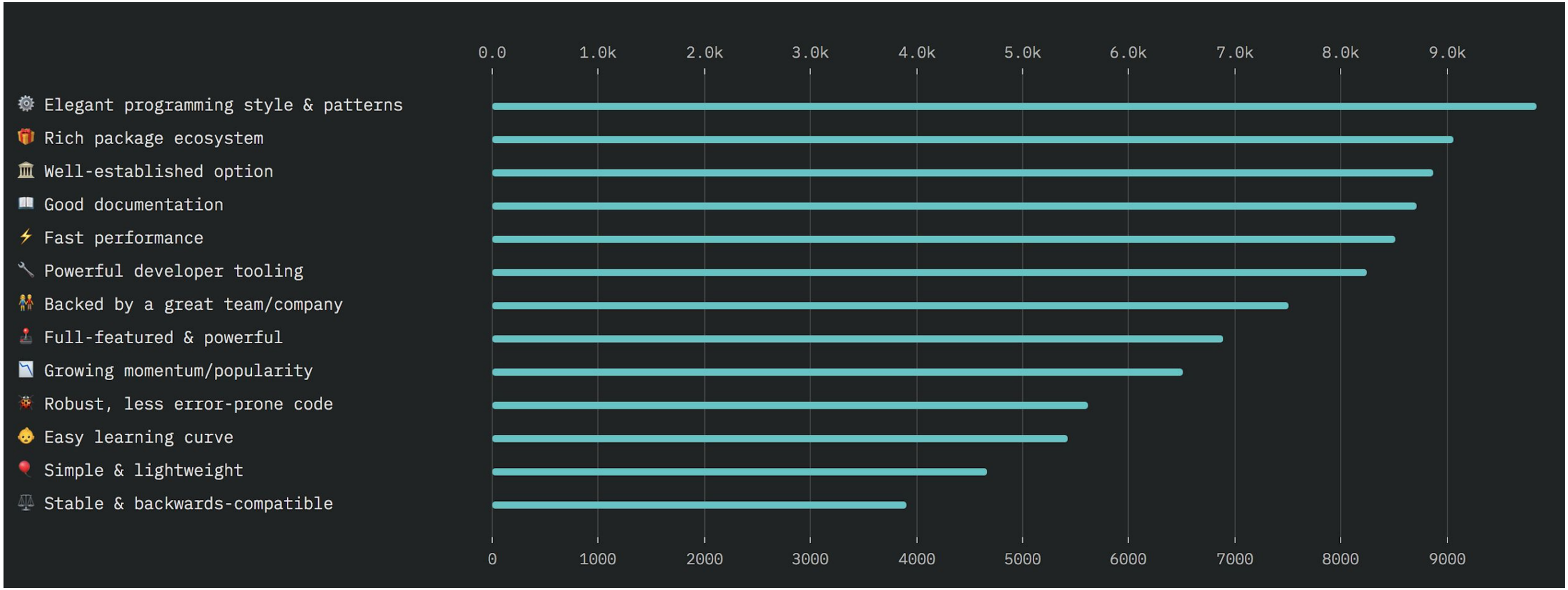
State of Frameworks

DataArt



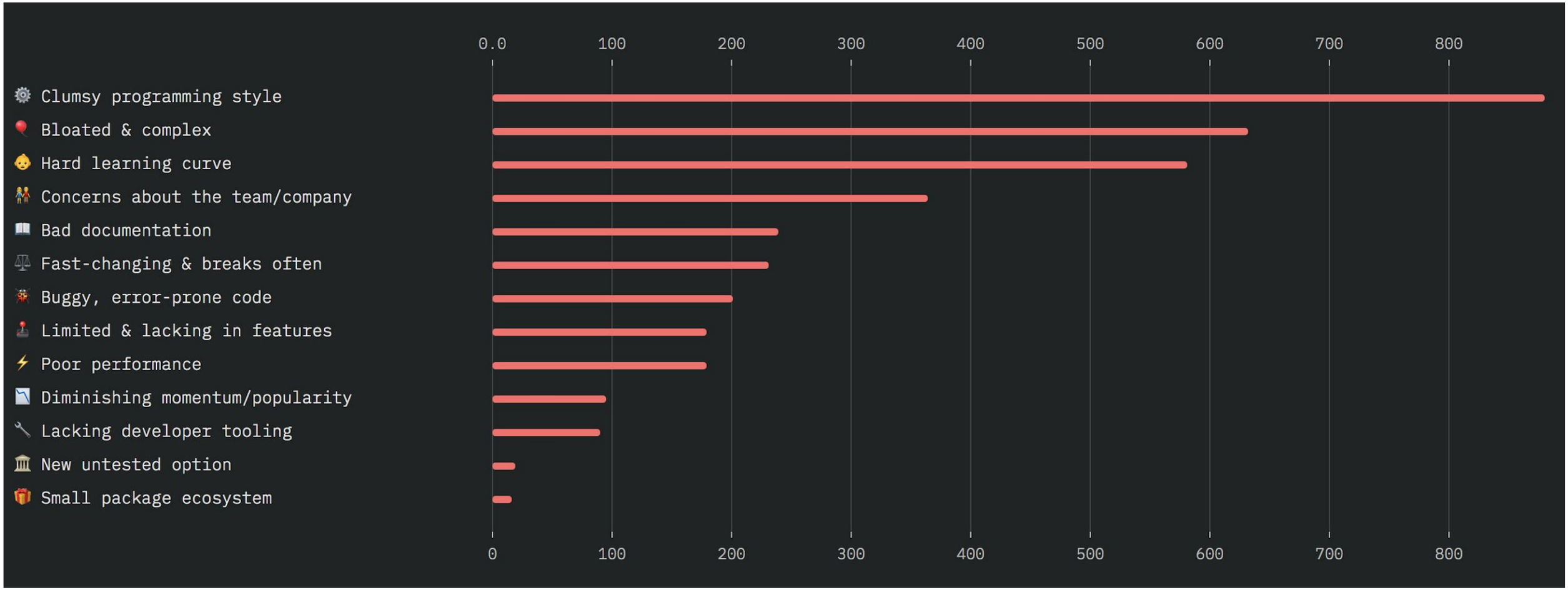
ReactJS pros

DataArt



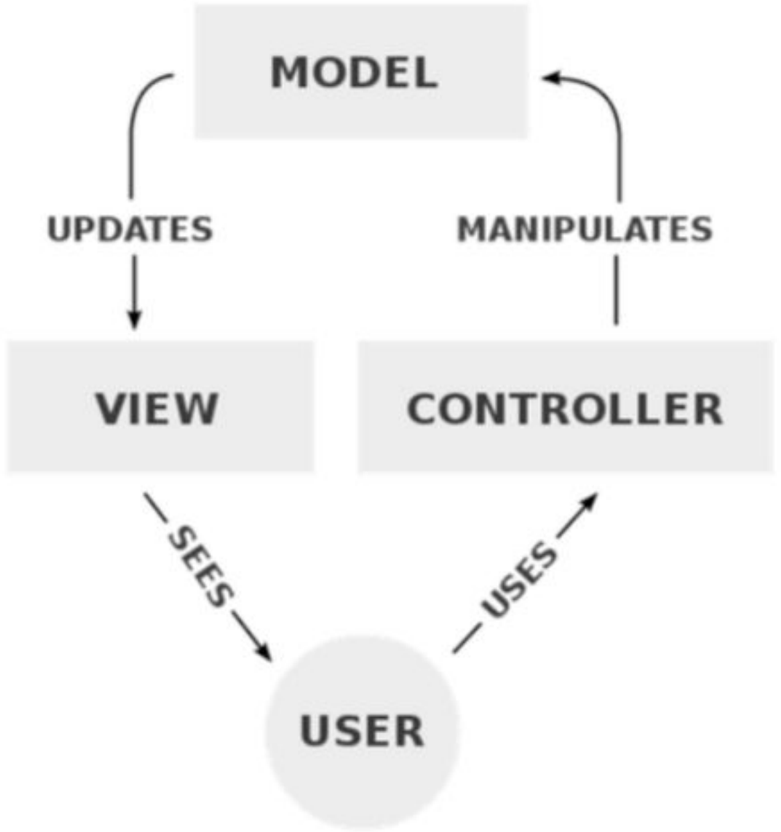
ReactJS cons

DataArt



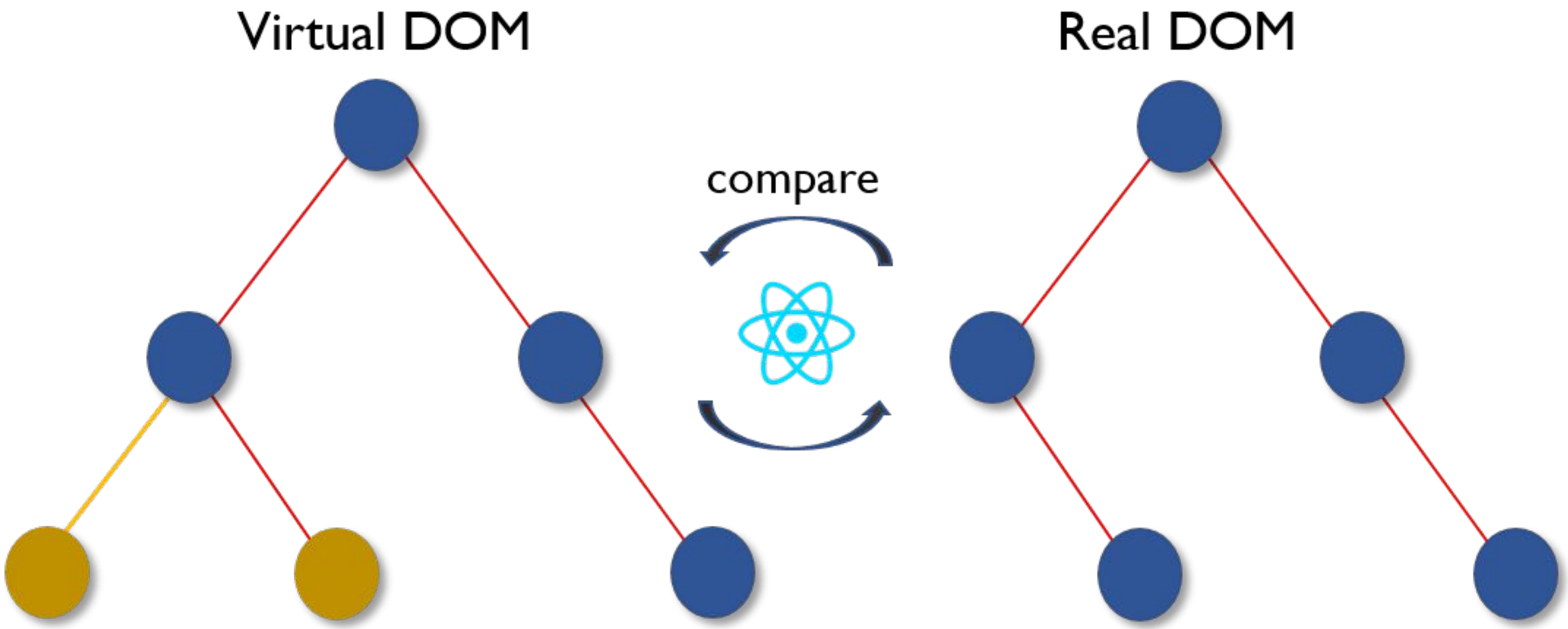
What is React

DataArt



What is Virtual DOM?

DataArt



State vs Props

DataArt

Props:

- Passed in from parent
- **<MyComp foo="bar" />**
- **this.props** read-only within
- Can be defaulted and validated

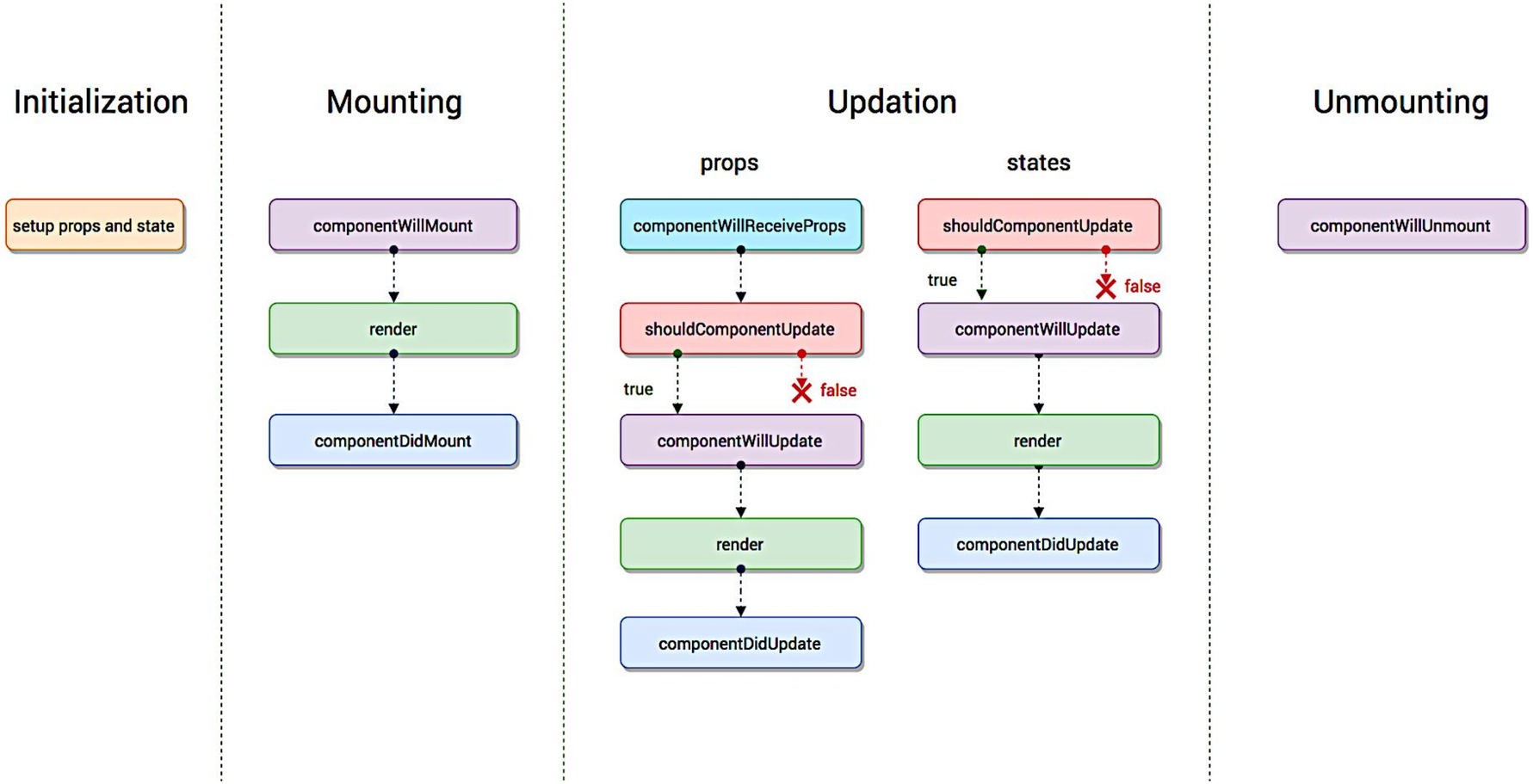
props get passed to the component similar to function parameters

State:

- Created within component
- **getInitialState**

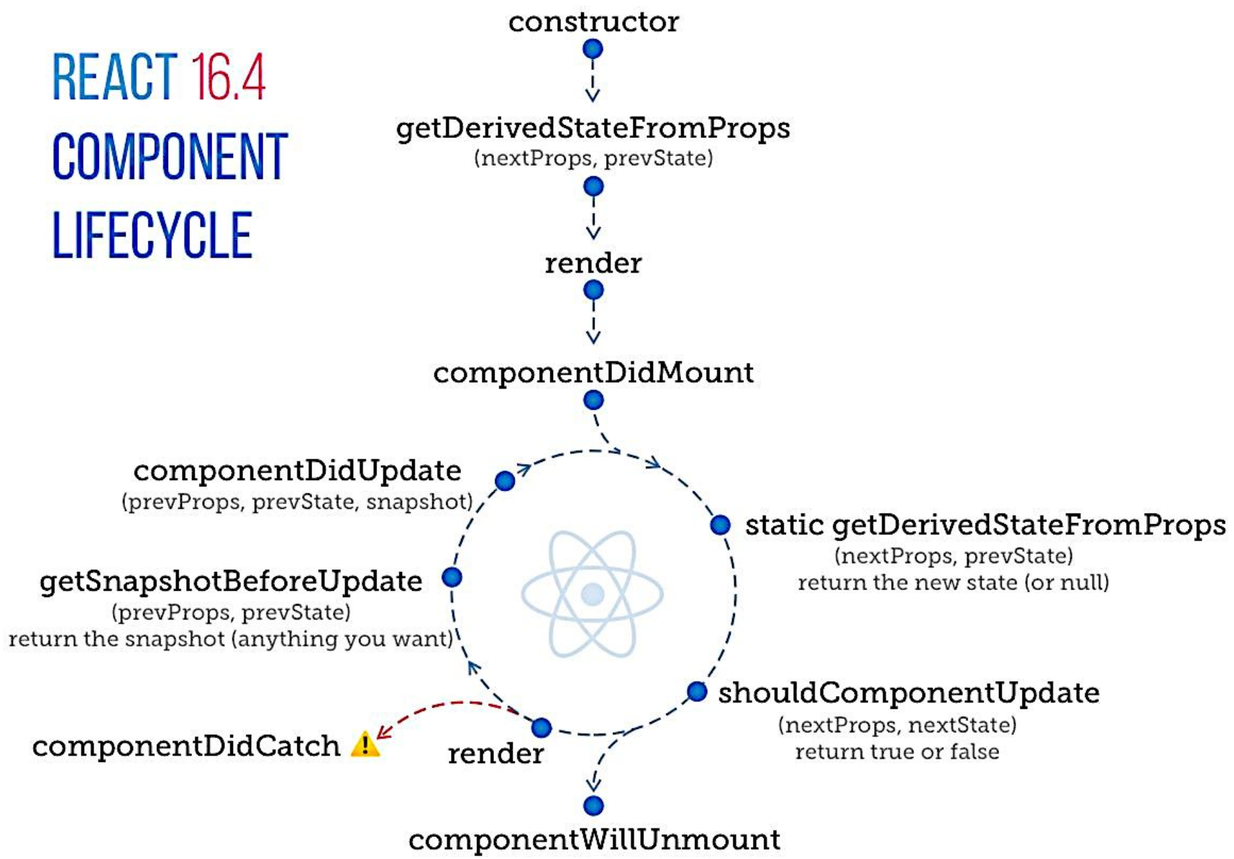
Component lifecycle

DataArt



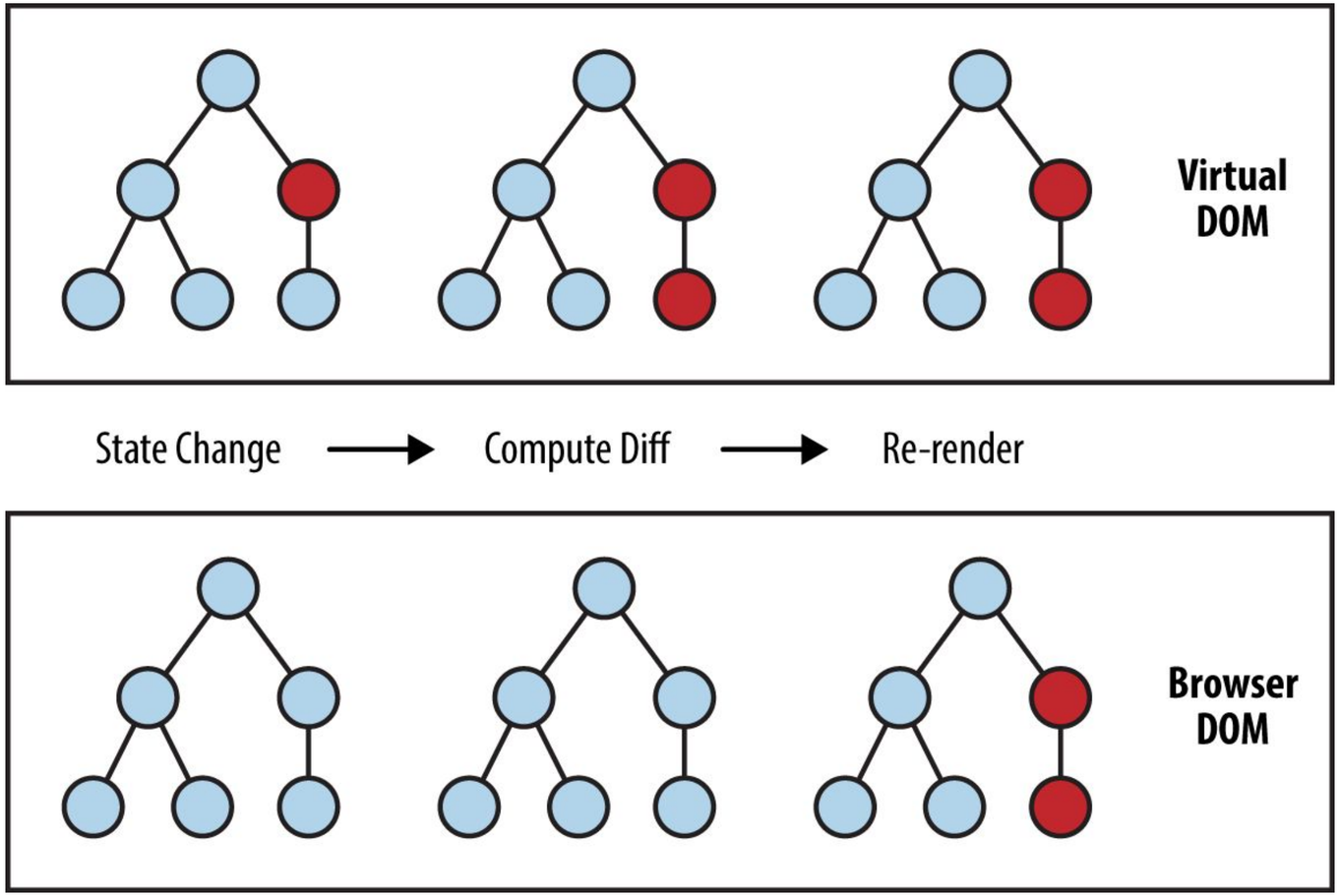
Component lifecycle

DataArt



How React works?

DataArt



What is JSX?

DataArt

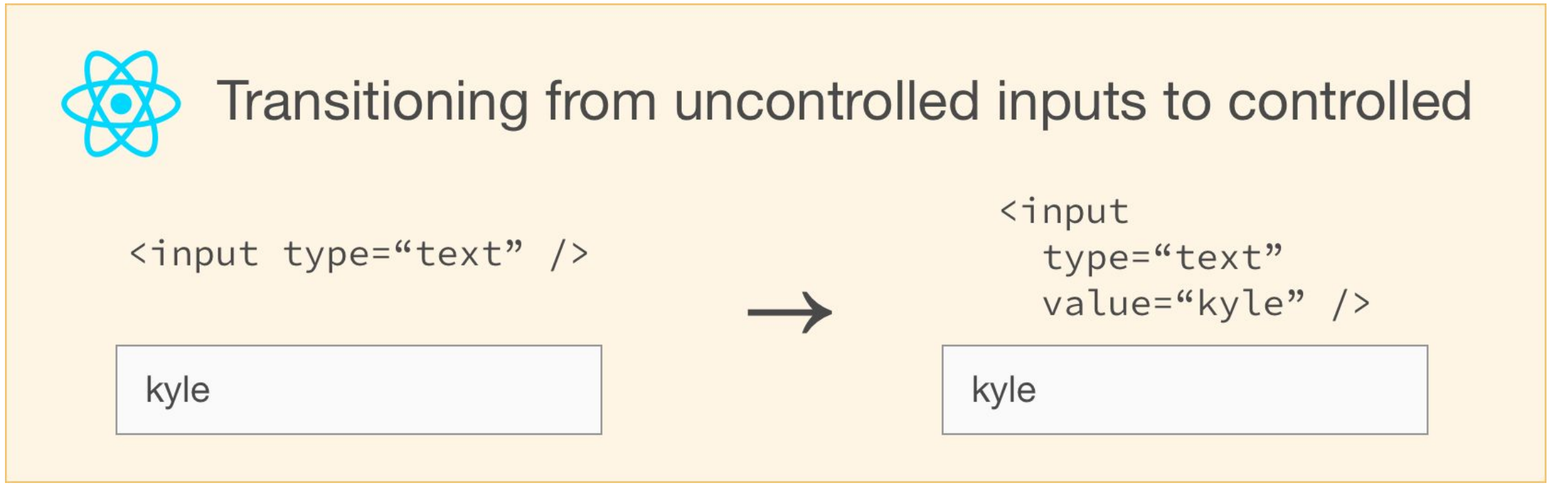
```
React.createElement(component, props, ...children);
```

```
{/* JSX */}
<p>Hello</p>
<div id="intro">Hello {name}</div>


// JavaScript Equivalent
React.createElement('p', null, 'Hello');
React.createElement('div', {id: 'intro'}, `Hello {name}`);
```

```
{/* JSX */}
<div id="intro">
  <h1>Hello, World</h1>
  <p>This is my world</p>
</div>


// JavaScript Equivalent
React.createElement('div', {id: 'intro'},
  React.createElement('h1, null, 'Hello World'),
  React.createElement('p', null, 'This is my world)
);
```

What are controlled components?

DataArt




```
state = {
  name: '',
};
```

type "a"

```
calls handleNameChange(a):

setState({
  name: 'a',
});
```

type "b"

```
calls handleNameChange(ab):

setState({
  name: 'ab',
});
```

Renders

Renders

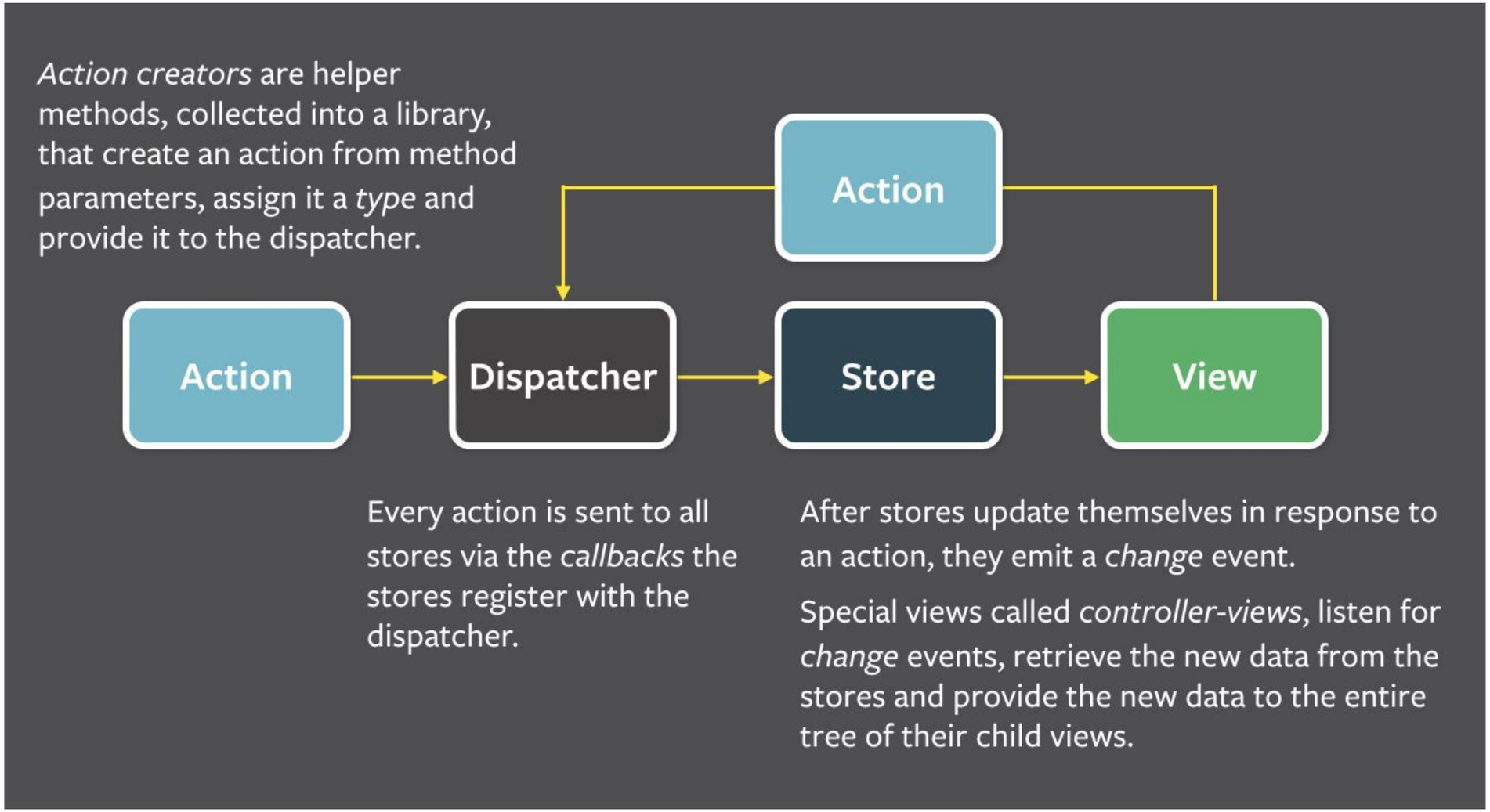
a

Renders

ab

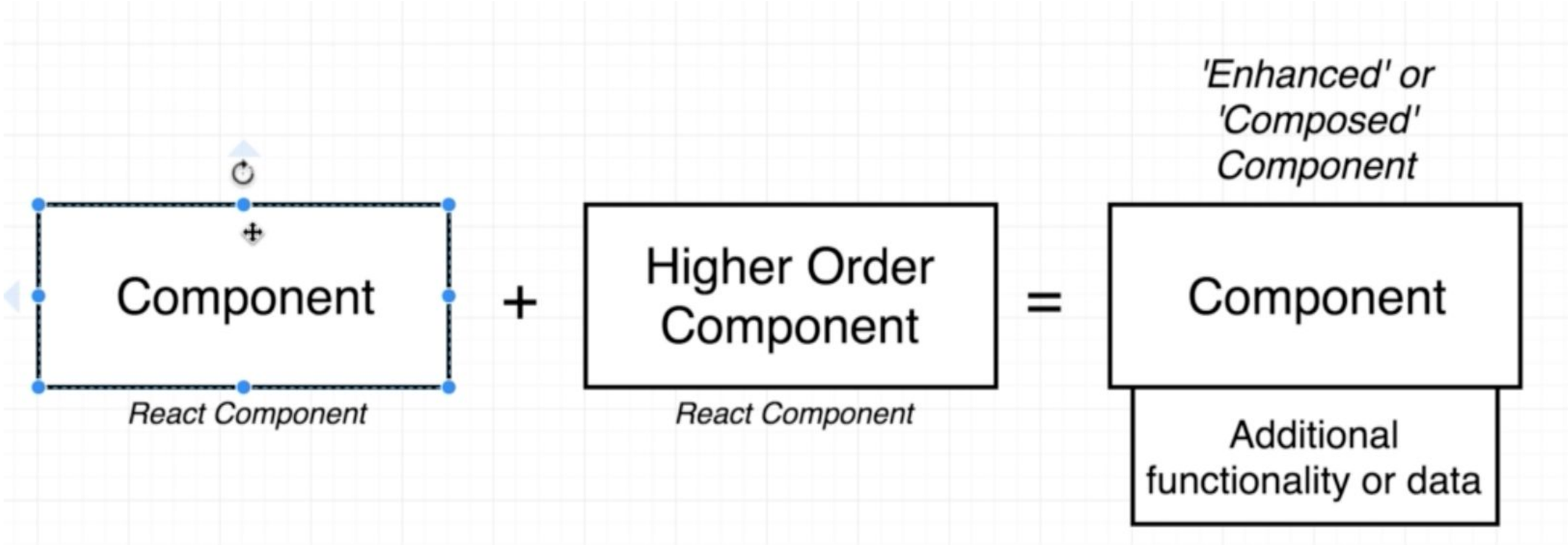
What is Flux?

DataArt



High Order Components

DataArt



Pure functions

DataArt

A **pure function** is a function which:

- Given the same input, will always return the same output
- Produces no side effects
- Relies on no external state

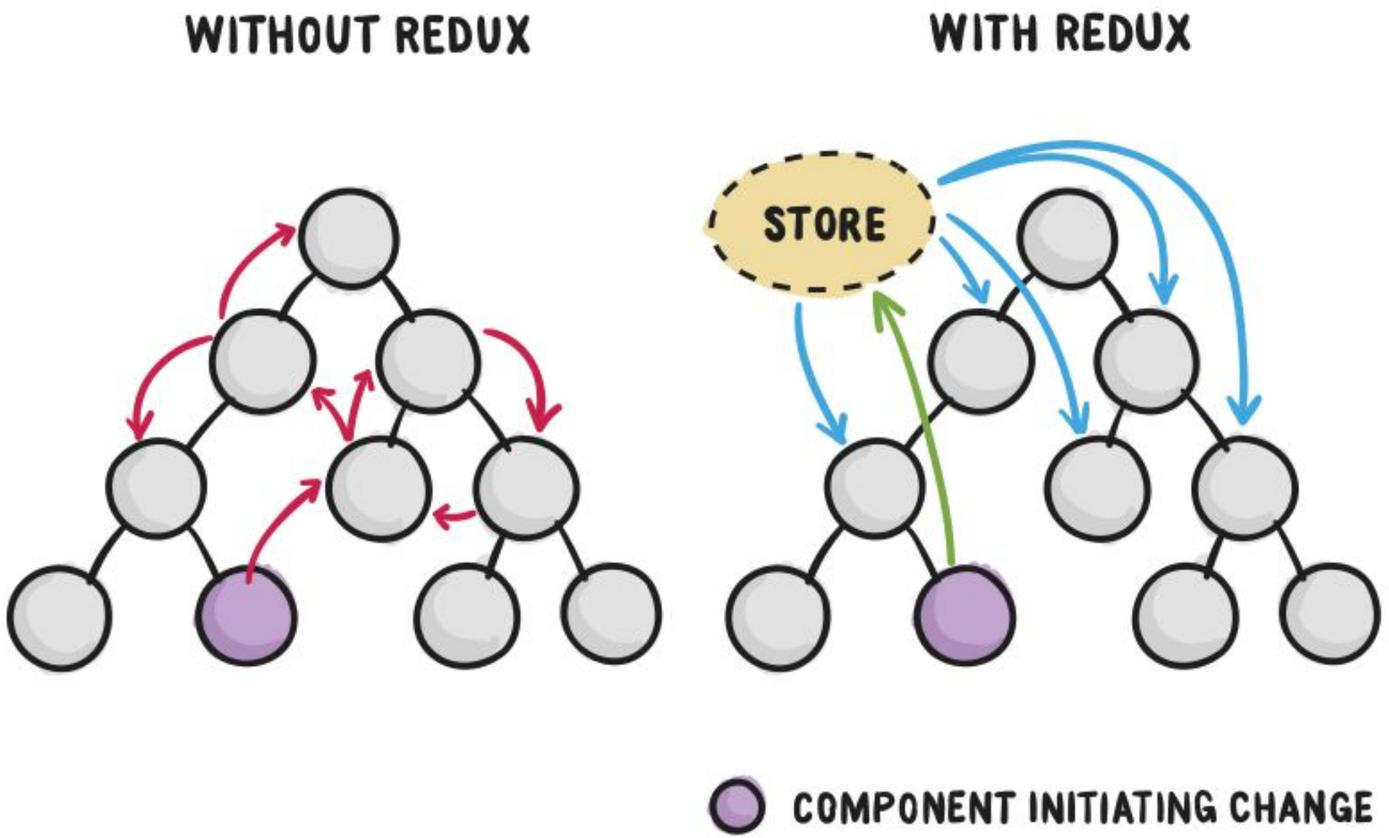
To make a copy of the input object using the **{…}** spread operator or **Object.assign()** and make changes to the new object, leaving the original unchanged.

This concept is fundamental to the central “state” model used in React and Redux, where state is an object containing the single source for your whole application.

What is Redux?

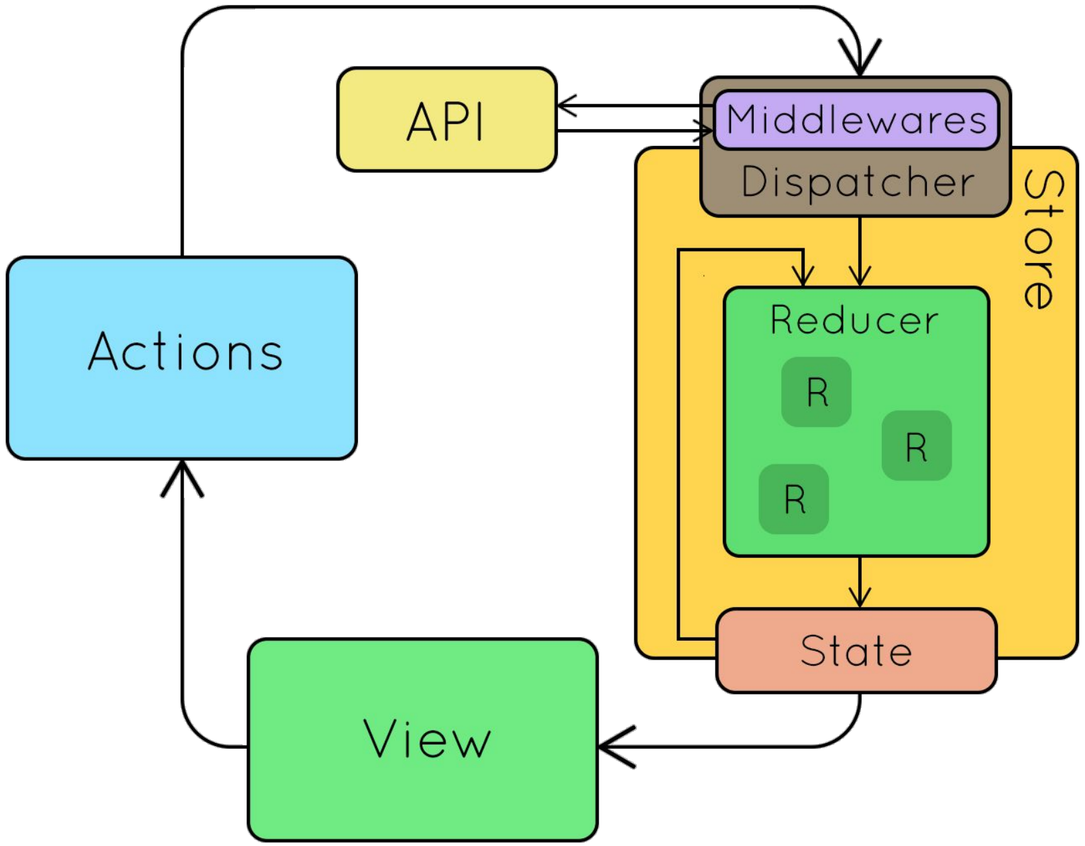
DataArt

Redux is a predictable state container for JavaScript apps.



How is state changed in Redux?

DataArt



Uni-Directional Architecture

DataArt

1. The application has a central / root state.
2. A state change triggers View updates.
3. Only special functions can change the state.
4. A user interaction triggers these special, state changing functions.
5. Only one change takes place at a time.

Redux – Confusions and Myths

DataArt

1. Redux is Flux – Wrong!
2. Redux is ONLY for React – Wrong!
3. Redux Makes Your Application Faster – Wrong!

Principles of Redux

DataArt

1. Single Source of Truth
    1. The state of your whole application is stored in an object tree within a single store.
2. State is Read Only
    1. The only way to change the state is to emit an action, an object describing what happened.
    2. Views cannot change the state DIRECTLY!
3. Use Pure Functions for Changes
    1. To specify how the state tree is transformed by actions, you write pure reducers.

Three Pillars of Redux

DataArt

- Store
- Action
- Reducers

Store

DataArt

```
1 import { createStore } from 'redux';
2 import reducer from './reducer';
3
4 const store = createStore(reducer);
```

getState:

```
1 store.getState();
```

dispatch:

```
1 const action = {
2   type: 'SUBTRACT',
3   payload: { value: 10 },
4 };
5
6 store.dispatch(action)
```

subscribe:

unsubscribe:

```
1 // To subscribe
2 const unsubscribe = store.subscribe(() => {
3   console.log('Application state updated');
4 });
5
6 // To unsubscribe
7 unsubscribe();
```

Action

DataArt

Note: Redux does not have explicit rules for how you should structure actions. In fact, Redux doesn't have any strict rules other than the three principles.

*Redux recommends that you give each action a **type** and that's a good idea. I also recommend using **payload** to store any more information related to the action. This keeps everything consistent.*

```
1 const action = {
2   type: 'ADD',
3   payload: { value: 5 },
4 };
```

Reducers

DataArt

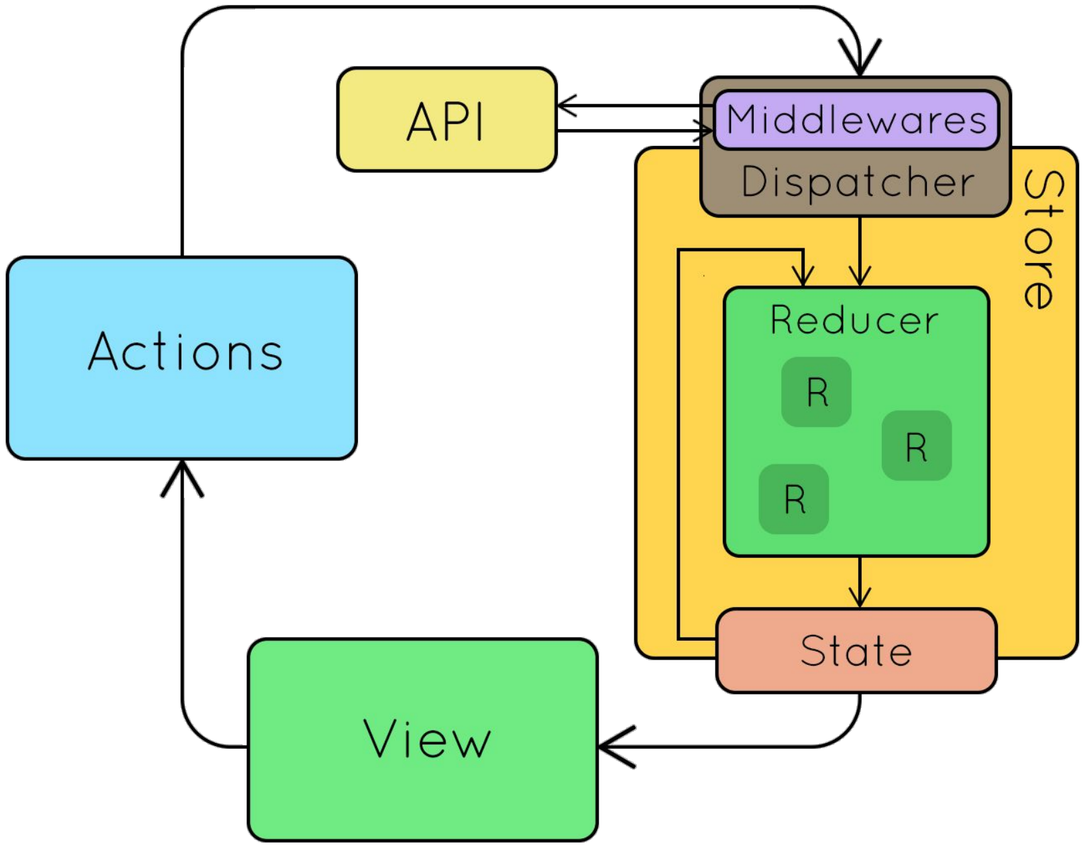
Reducers are the pure functions. They know what to do with an action and its information (payload).

They take in the current state and an action and return a new state.

Unlike Flux, Redux has a single store. Your entire applications state is in one object. That means using a single reducer function is not practical. We’d end up with a 1000-line file that nobody would want to read.

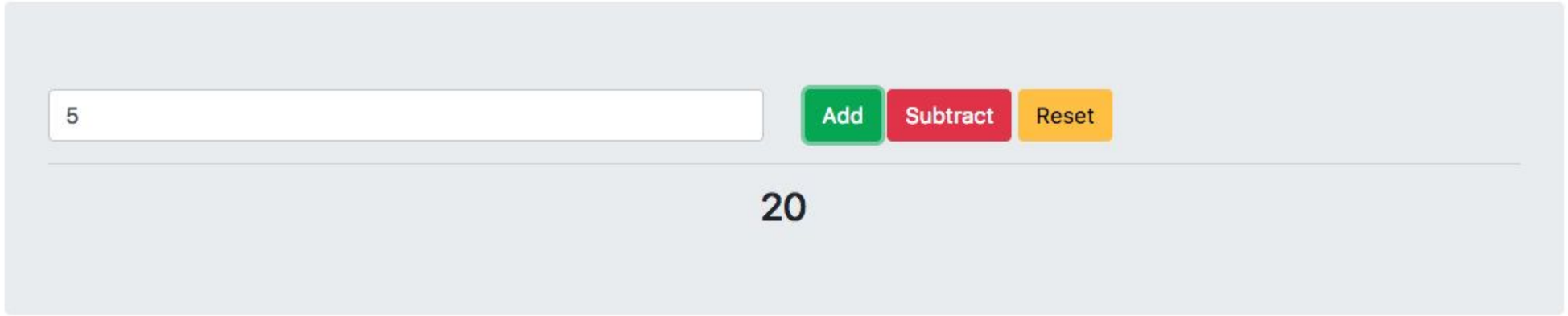
How is state changed in Redux?

DataArt



Demo app

DataArt



Setup

DataArt

```
1  const reducer = (state = 0, action) => {
2    console.log(action);
3
4    switch(action.type) {
5      case 'ADD':
6        return state + action.payload.value;
7      case 'SUBTRACT':
8        return state - action.payload.value;
9      case 'RESET':
10       return 0;
11     default:
12       return state;
13   }
14 };
15
16 const store = createStore(reducer);
```

Utility Functions

DataArt

```
1  /**
2   * Gets the value of the input field
3   *
4   * @return {Number} Value of the input field
5   */
6  const getValue = () => {
7    const value = parseInt(document.getElementById('op-number')
8    return isNaN(value) ? 0 : value;
9  };
10
11 /**
12  * Sets the total value as returned by the store
13  */
14 const setTotal = value => {
15   document.getElementById('grand-total').innerHTML = value;
16 };
```

Action Creators

DataArt

```
1  /**
2   * Action Creator. Returns an action of the type 'ADD'
3   */
4  const add = () => ({
5    type: 'ADD',
6    payload: { value: getValue() },
7  });
8
9  /**
10  * Action Creator. Returns an action of the type 'SUBTRACT'
11  */
12 const subtract = () => ({
13   type: 'SUBTRACT',
14   payload: { value: getValue() },
15 });
16
17 /**
18  * Action Creator. Returns an action of the type 'RESET'
19  */
20 const reset = () => ({ type: 'RESET' });
```

Hook Behavior

DataArt

```
// Handle add button click
document.getElementById('add-btn').addEventListener('click', () => {
  store.dispatch(add());
});

// Handle subtract button click
document.getElementById('subtract-btn').addEventListener('click', () => {
  store.dispatch(subtract());
});

// Handle reset button click
document.getElementById('reset-btn').addEventListener('click', () => {
  store.dispatch(reset());
});
```

Finally

DataArt

```
1 // Subscribe to updates
2 store.subscribe(() => {
3   setTotal(store.getState());
4 });
```

Вопросы?

DataArt