Bài 9

React Redux

Module: BOOTCAMP WEB-FRONTEND

Thảo luận

- React Redux
- Redux Actions
- Redux Reducers
- Redux Store

Mục Tiêu

- Trình bày được tổng quan Redux
- Trình bày được khái niệm Actions
- Trình bày được khái niệm Reducers
- Trình bày được khái niệm Store
- Trình bày được khái niệm Middleware

Nguồn gốc Redux?

Năm Quý Tị (2013), Facebook gia tộc bố cáo thiên hạ rằng Ăn Gô La đại pháp (Angular) của Google gia tộc chậm chạp, nặng nề, cho xuất thế một bộ chiêu thức gọi là Rối An Tâm Pháp (React).

Thế nhưng Rối An Tâm Pháp lại chỉ là một bộ tâm pháp cường thân kiện thể, không thể dùng để rèn luyện nội công (chỉ là một library để render view). Do đó, không lâu sau Facebook gia tộc tiếp tục cho ra đời một bộ tâm pháp cơ bản (kiến trúc thiết kế) và một công pháp cùng tên là Phờ Lắc thần công (Flux). Nghe đồn Rối An Tâm Pháp và Phờ Lắc Thần Công kết hợp lại sẽ thành tuyệt học dời non lấp bể, không gì không làm đc. Nhân sĩ giang hồ (coder) vốn nhẹ dạ cả tin lại rủ nhau tu luyện.

Phờ Lắc thần công rối rắm khó học, nhân sĩ 10 phần học thì 4-5 phần tẩu hỏa nhập ma, phần còn lại cũng trầy da tróc vẩy mà công lực cũng chẳng được như lời Facebook gia tộc quảng cáo.

Bấy giờ có một nhân sĩ giang hồ tự là Đan (Dan Abramov), đang tu luyện đồng thời Phờ lắc thần công và Ê La thần công (Elm) mới nhận ra rằng hai môn võ công có nhiều điểm chung, chỉ khác chiêu thức, Đan bèn nảy ra ý định hợp nhất hai môn này lại. Không lâu sau (5/2015), Đan cho xuất thế một bộ công pháp mang tên Rì Đắt thần công (Redux), mang ưu điểm của cả hai môn võ công đồng thời loại bỏ những phức tạp dư thừa của Phờ Lắc thần công.

Nhân sĩ giang hồ nghe vậy mừng lắm, thế là lại kéo nhau đi học Rì Đắt, còn Đan thì được Facebook gia tộc mời về làm tộc nhân.

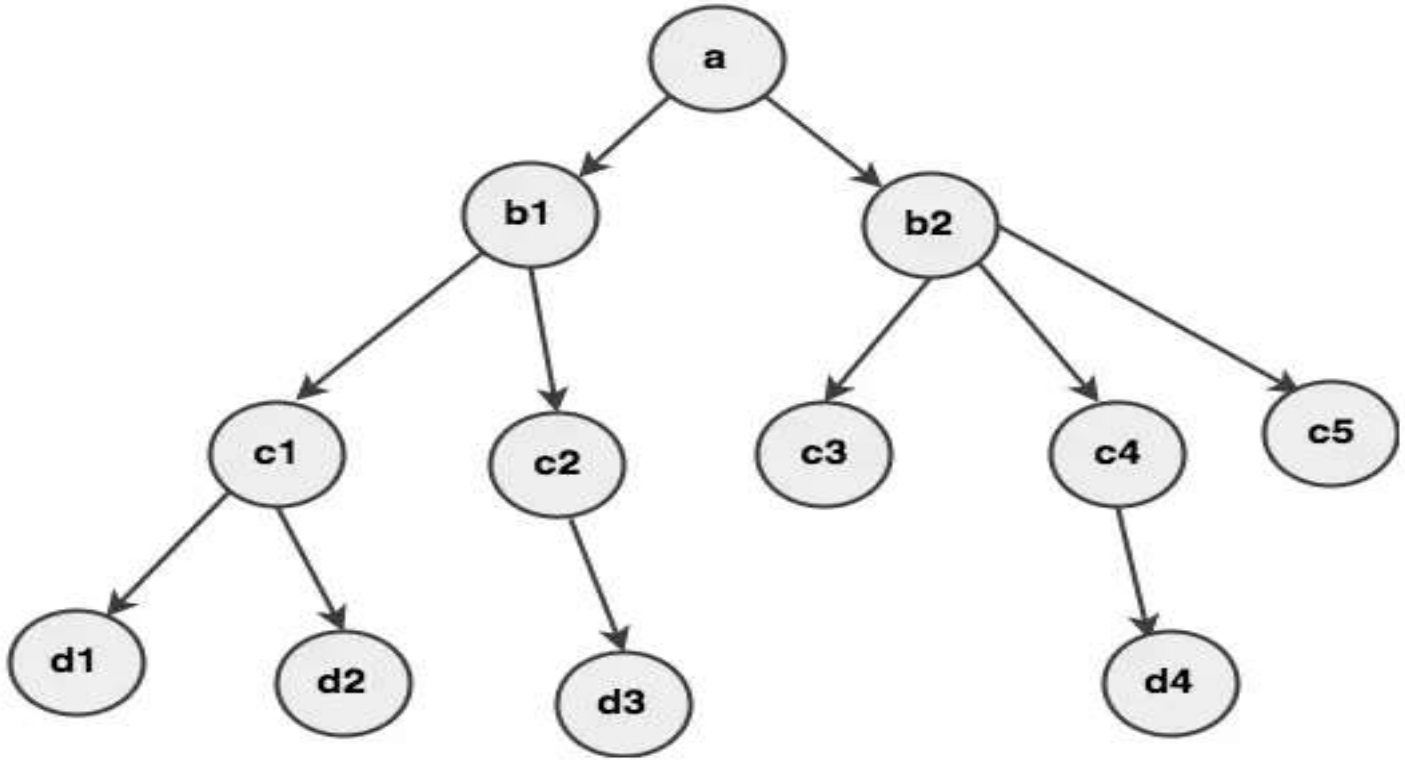
Trích "JavaScript Lược Sử Giang Hồ" có sửa đổi bổ sung (yaoming)

React Redux

- Redux là một công cụ quản lý trạng thái có thể dự đoán được cho ứng dụng Javascript.
- state của ứng dụng được lưu trong một nơi gọi là store và mỗi component đều có thể access bất kỳ state nào mà chúng muốn từ chúng store này.
- Có 3 thành phần của Redux: Actions, Store, Reducers.

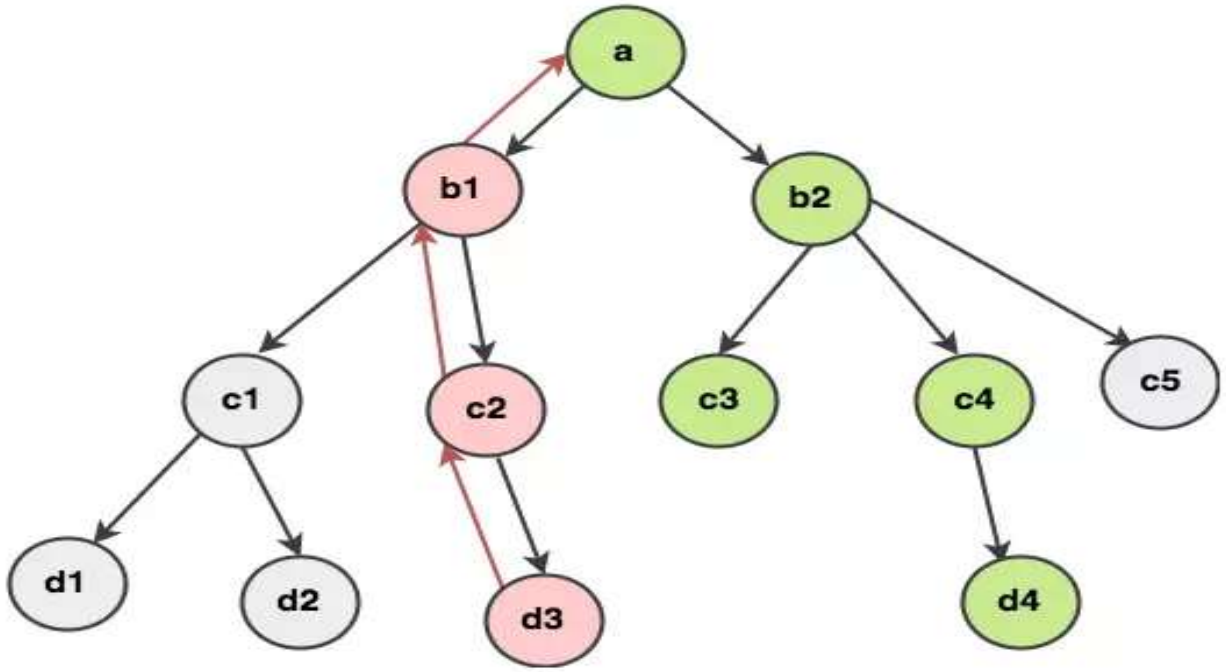
WHY Redux?

- Giả sử chúng ta có 1 ứng dụng các node như trong hình là tượng trưng cho một single page application



WHY Redux?

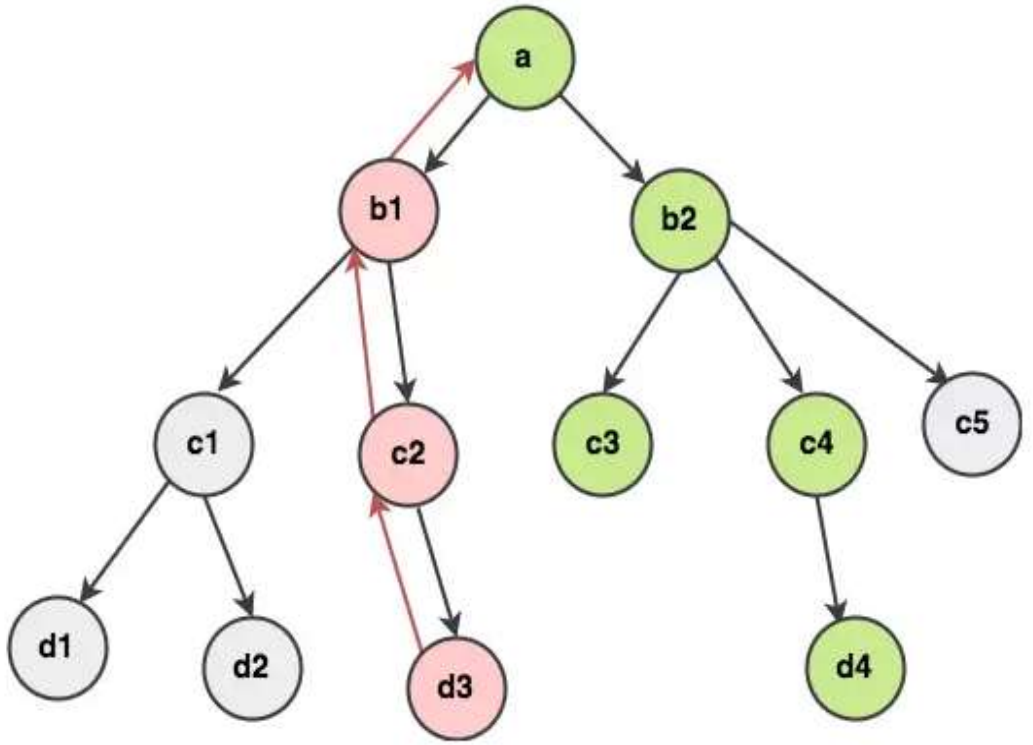
- Có một hành động nào đó được kích hoạt ở node d3 và ta muốn thay đổi trạng thái (state) ở d4 và c3 thì luồng dữ liệu sẽ được truyền từ node d3 trở về node a rồi từ node a mới truyền được đến các node d4 và c3



WHY Redux?

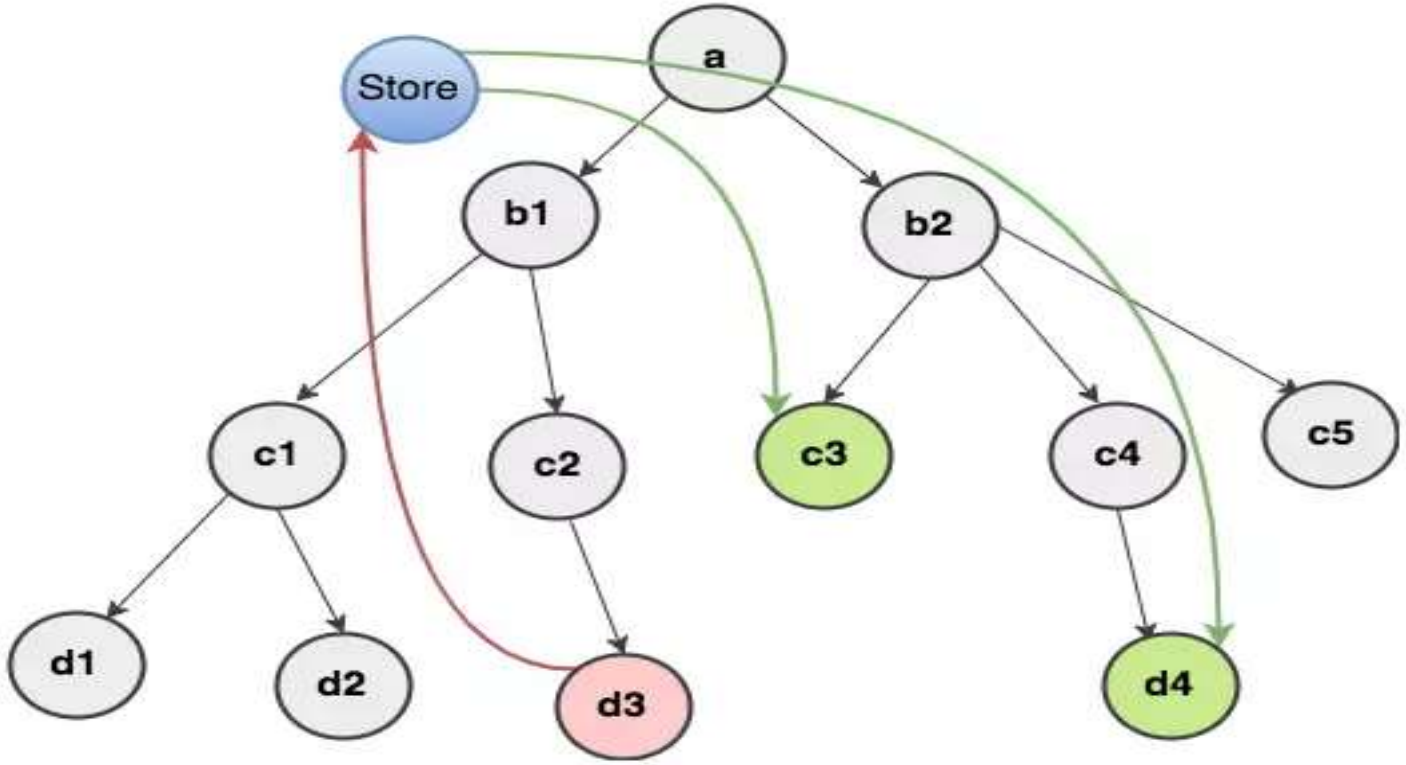
- Cập nhật trạng thái (state) cho node d4: d3-c2-b1-a-b2-c4-d4
- Cập nhật trạng thái (state) cho node c3: d3-c2-b1-a-b2-c3

=> Nếu chỉ sử dụng ReacJs để cập nhật các trạng thái (state) thì thật sự là một khó khăn rất là lớn



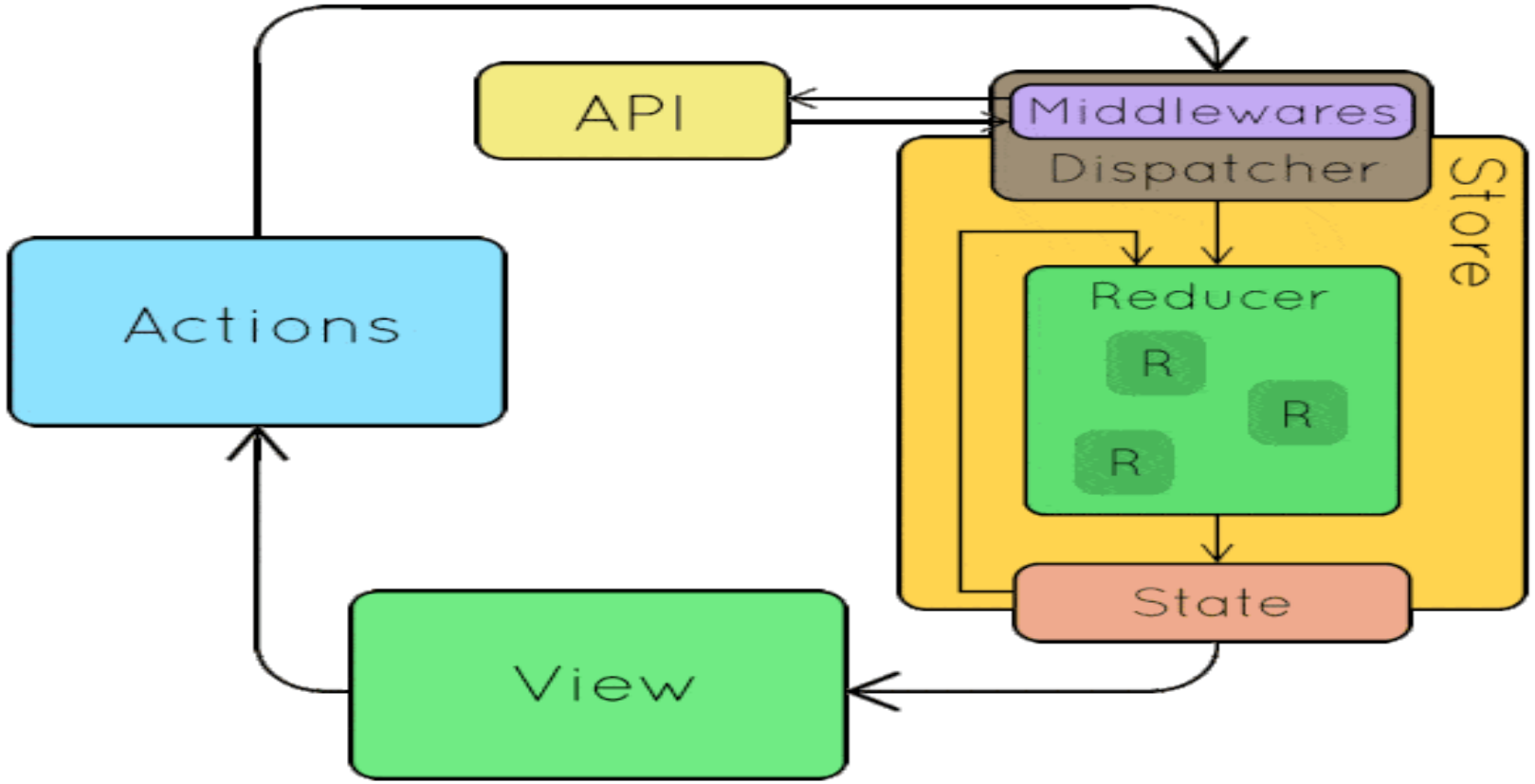
WHY Redux?

- Với Redux, ta chỉ cần dispatch một action từ node d3 về store rồi d4 và c3 chỉ cần connect tới store cà cập nhật data thay đổi thế là bài toán được giải quyết một cách dễ dàng



React Redux

Mô hình



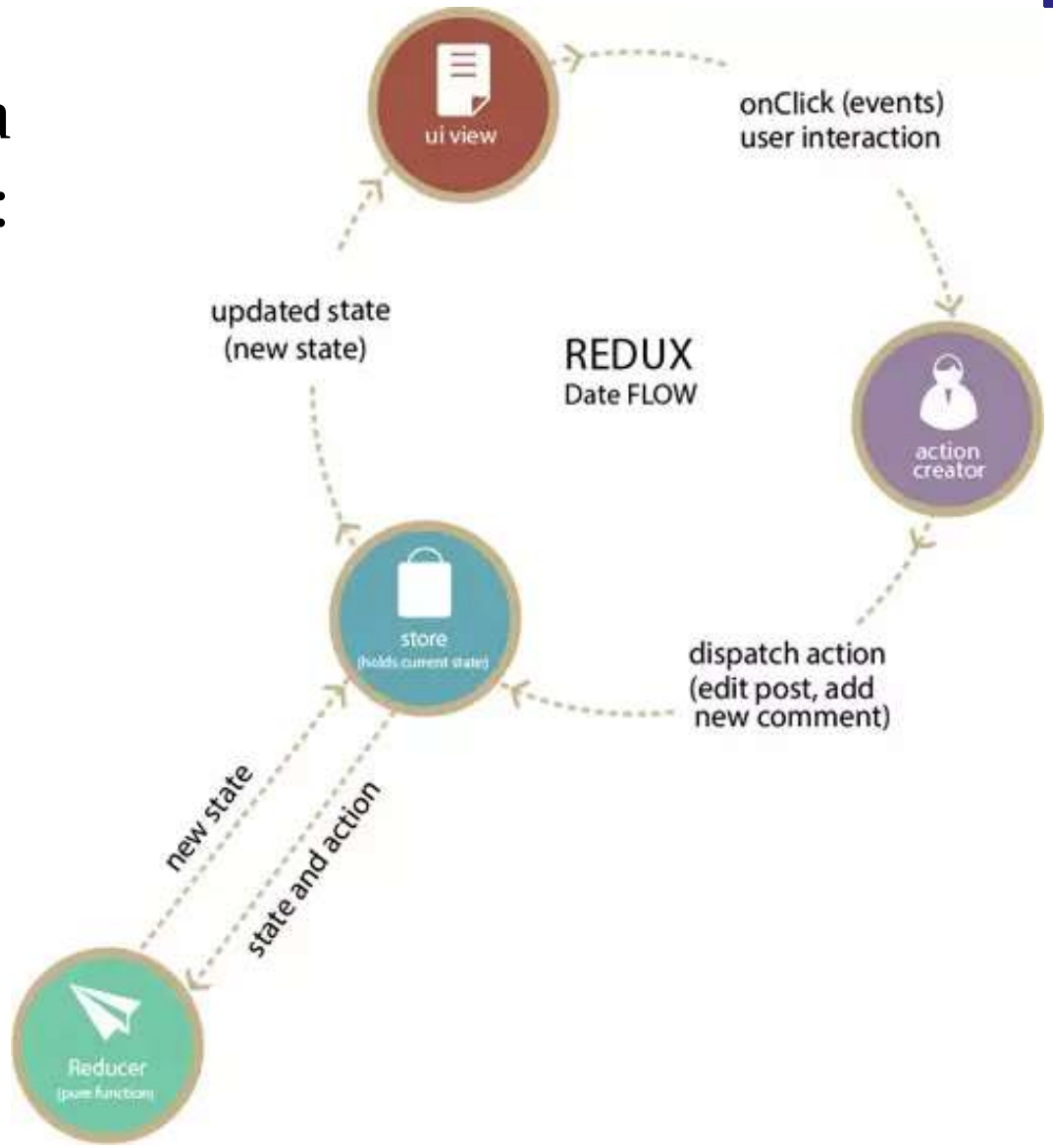
Nguyên lý hoạt động

Redux được xây dựng dựa trên 3 nguyên lý:

- Nguồn dữ liệu tin cậy duy nhất: State của toàn bộ ứng được chứa trong một object tree nằm trong Store duy nhất
- Trạng thái chỉ được phép đọc: Cách duy nhất để thay đổi State của ứng dụng là phát một Action (là 1 object mô tả những gì xảy ra)
- Thay đổi chỉ bằng hàm thuần túy: Để chỉ ra cách mà State được biến đổi bởi Action chúng ta dùng các pure function gọi là Reducer

Các thành phần cơ bản

Nếu muốn sử dụng Redux, chúng ta cần nhớ 4 thành phần chính của Redux: **Store**, **Views** , **Actions**, và **Reducers**



Redux Actions

- Action là sự kiện.
- Là cách duy nhất để gửi dữ liệu từ ứng dụng đến store.
- Các action được gửi bằng phương thức store.dispatch().
- Các action là các đối tượng javascript
- Thuộc tính type để chỉ ra hành động sẽ thực hiện.
- Thuộc tính payload chứa thông tin dữ liệu cần được gửi đi.

Redux Actions

Ví dụ :

```
export const getProduct = (categoryId) => {
  return {
    type : GET_PRODUCT',
    payload: categoryId
  }
};
```

Action này trả về 1 plain object bao gồm
Type: Tên action
Payload: dữ liệu cần gửi đi

Tại nơi gọi action sẽ có mã như sau:

```
Dispatch(getProduct(categoryId));
```

Redux Reducer

- Reducer là các hàm pure function lấy trạng thái hiện tại của ứng dụng.
- Reducer thực hiện một hành động và trả về trạng thái mới.
- Dựa trên hàm reduce trong javascript, trong đó một giá trị được tính từ nhiều giá trị sau khi thực hiện chức năng gọi lại.

Redux Reducer

Ví dụ:

```
const initialState = {
  products: []
};
export default (state = initialState, action) => {
  switch (action.type){
    case GET_PRODUCT':
      return {
        …state,
                products: action.payload
      }
    default:
      return state;
  }
};
```

Có thể coi reducer là một bộ chuyển đổi với
Input là state hiện tại, action
Output là state mới tương ứng

Redux Store

- Store lưu giữ trạng thái của ứng dụng.
- Mỗi reducer sẽ xử lý và trả về state. Tập hợp các state này sẽ thành cây state mà Store nắm giữ.
- Chỉ có một Store

Redux Store

Ví dụ:

```
import {combineReducers} from 'redux'
import todos_reducer from './todos'
import get_product_reducer from './get_Product_filter'

export const rootreducer = combineReducers({todos: todos_reducer, get_product: get_product_reducer});
```

Tại nơi khai báo và sử dụng store.
Tạo store và truyền cho Provider để tạo ra một mạng lưới kết nối

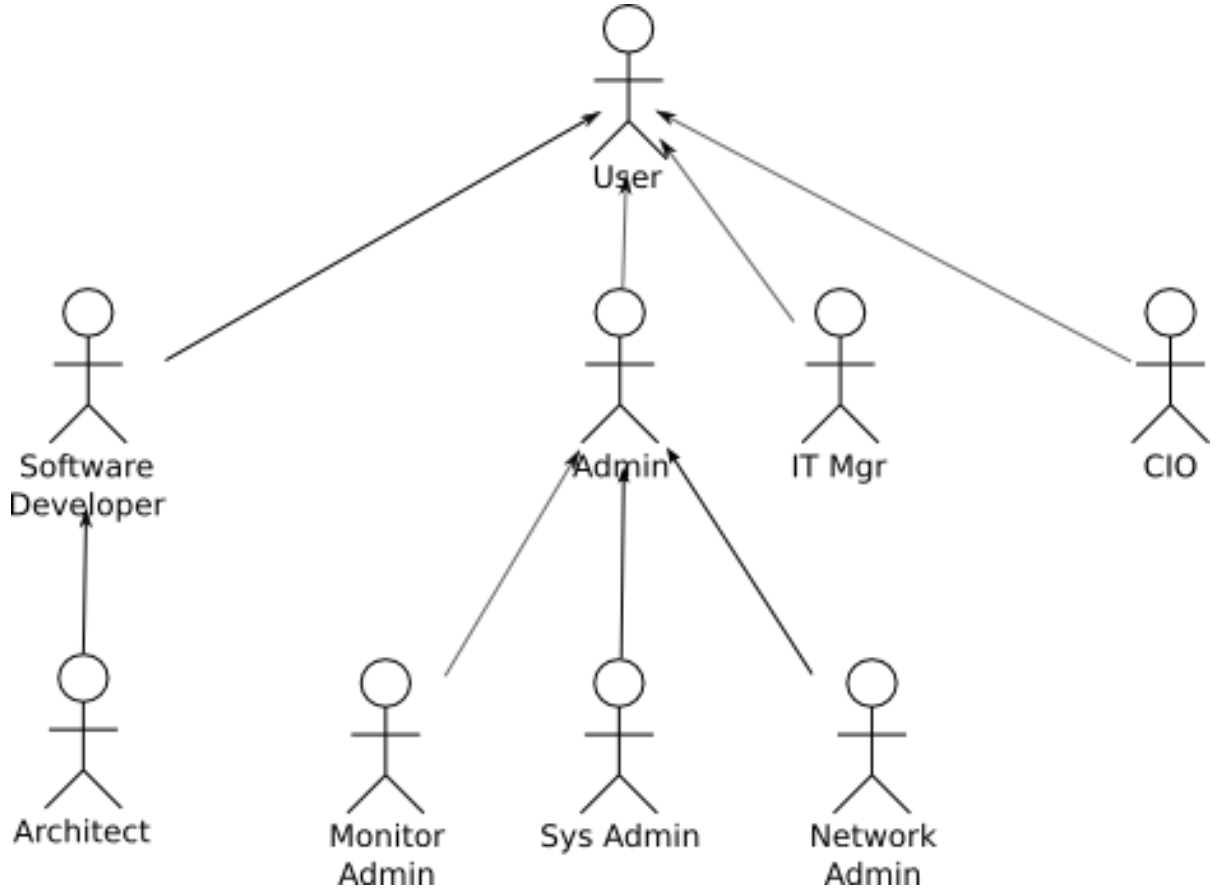
```
const store = createStore(rootreducer, window.__REDUX_DEVTOOLS_EXTENSION__ &&
window.__REDUX_DEVTOOLS_EXTENSION__());
<Provider store={store}>
    <App/>
</Provider>, document.getElementById('root')
```

Redux Middleware

- Middleware cho phép chúng ta can thiệp vào giữa thời điểm dispatch một action và thời điểm action đến được reducer.
- Một middleware function là 1 function return 1 function return 1 function
- Một số thư viện để làm việc với Middleware như: redux-thunk, redux-saga, redux-observable

Data flow

Tưởng tượng ứng dụng của chúng ta là một văn phòng, có các thành viên làm việc với nhau để giải quyết công việc chung. Cùng điểm qua các nhân vật trong phòng trước khi xem các họ tương tác với nhau để giải quyết công việc



Data flow

Anh ta giữ nhiệm vụ tạo ra các Action, là bước đầu tiên trong luồng mà các thay đổi và tương tác đều đi qua. Bất cứ khi nào trạng thái của app hay là render của view thay đổi thì đầu tiên là một hành động sẽ được tạo ra.

Hãy hình dung anh này như một anh chàng đánh máy, anh ta biết bạn cần truyền đạt điều gì và cần phải đánh ra văn bản theo định dạng nào cho mọi người đều hiểu được.

The Action Creator tạo ra một action là formated object chứa type và thông tin của action đó. Type thường sẽ là một hằng số được định nghĩa trước, kiểu như INCREASE hay DECREASE.

The Action Creator

Data flow

Hãy hình dung đây là ông sếp đầy quyền lực, toàn bộ các thao tác với State tree (getState, updateState, registerListener...) đều do ông này quản lý.

Cái ông này làm nhiều việc nhỉ (?) Không thực ra ông ấy chỉ quản lý trạng thái của State tree thôi. Khi nhận được Action ông ấy sẽ đi hỏi The reducers xem State sẽ thay đổi ra sao chứ không tự làm.

The Store

Data flow

Khi The Store muốn biết State thay đổi như thế nào, ông ấy sẽ gọi cho The Reducers. Ở đây có một ông là Root Reducer nữa, ông này sẽ chịu trách nhiệm cắt ra State cần thay đổi dựa trên keys mà The Store gửi cho và đưa nó cho Reducer biết cách xử lý.

Hãy hình dung đây là một nhóm các thanh niên cuồng Photocopy. Họ không thích làm rối tung những thứ họ được đưa cho, nên họ tạo ra bản sao của chúng và thực hiện thay đổi trên bản sao đó.

The Reducers

Data flow

Trong Redux có 2 khái khái niệm: smart and dumb components.

- ❖ Smart component: có thể gọi là containers
- ➢ Hãy hình dung đây là anh quản lý của nhóm nhỏ, anh ta phụ trách các Action. Khi các thành viên dưới anh ta (dumb components) cần phát 1 action, anh ta sẽ gửi action cho các thành viên dưới dạng props, các thành viên chỉ cần coi đó là các callback mà không quan tâm nó là cái gì.
- ➢ Anh ta không thích ăn diện (không có css).
- ➢ Khi có việc cần thay đổi (DOM) thì anh ta sẽ sắp xếp các thành viên dưới làm chứ hiếm khi tự làm.

The Viewers

Data flow

Trong Redux có 2 khái khái niệm: smart and dumb components.

- Dumb components: có thể gọi là components
- Hãy hình dung đây là mấy thanh niên học việc, thiên lôi chỉ đâu đánh đấy.
- Mấy thanh niên này không phụ thuộc trực tiếp vào các Action, vì được anh quản lý đưa cho rồi. Điều này có nghĩa là mấy thanh niên này có thể đưa sang bộ phận khác làm cũng đc, miễn là có anh quản lý đưa "hàng" cho xài.
- Mấy thanh niên này thì đẹp trai, tóc tai vuốt vuốt các thứ (có css riêng), nhưng đôi khi bị cấp trên bắt mặc theo ý sếp (nhận props style) thì vẫn phải chịu.

The Viewers

Các nhân vật đó phối hợp với nhau như thế nào?

Setup

Các bộ phận cần được nối với nhau. Việc này xảy ra lần đầu vào app.

1. Bảo ông Store sẵn sàng

Ông giám đốc Root component tạo ra Store, chỉ cho ông Store dùng Root Reducer nào thông qua createStore(). Ông Root Reducer thì đã có team sẵn rồi, được tập hợp lại thông qua combineReducers()

Setup

2. Chuẩn bị liên lạc giữa ông Store và các bộ phận khác.

Root component bao các subcomponents với provider component (The view layer binding) và tạo kết nối giữa Store với các Provider.

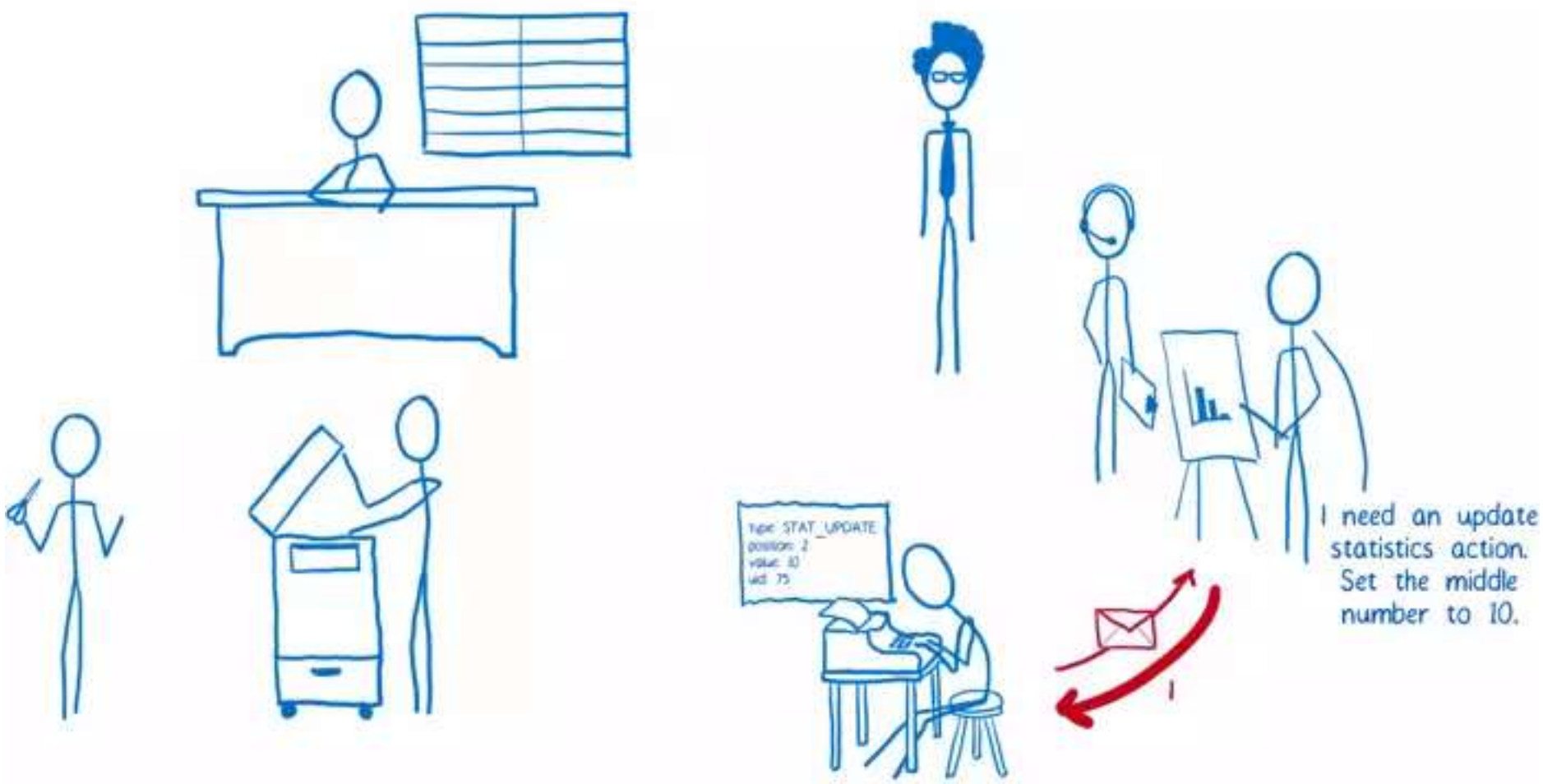
Provider tạo ra 1 mạng cơ bản để cập nhật các components. Smart Components kết nối vào mạng bằng connect(), điều này đảm bảo họ nhận được cập nhật State.

Setup

3. Chuẩn bị các actions callback.

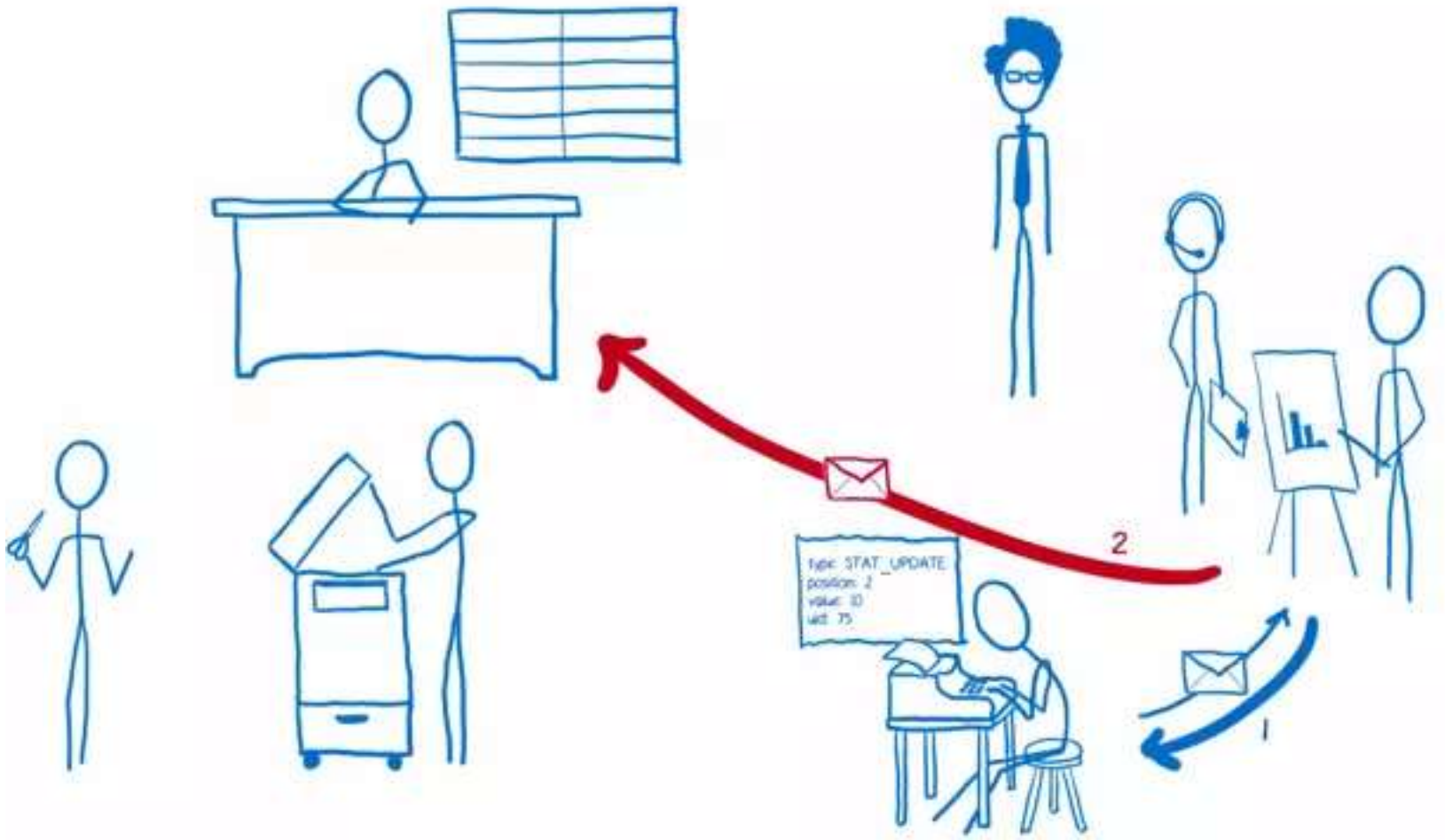
Để các Dump Components làm việc với Action dễ dàng hơn, các Smart Components có thể chuẩn bị các action callback thông qua bindActionCreators(). Bằng cách này, họ chỉ có cần truyền các callback cho Dump Components. Các Actions sẽ được tự động gửi đi sau khi nó được định dạng.

Flow



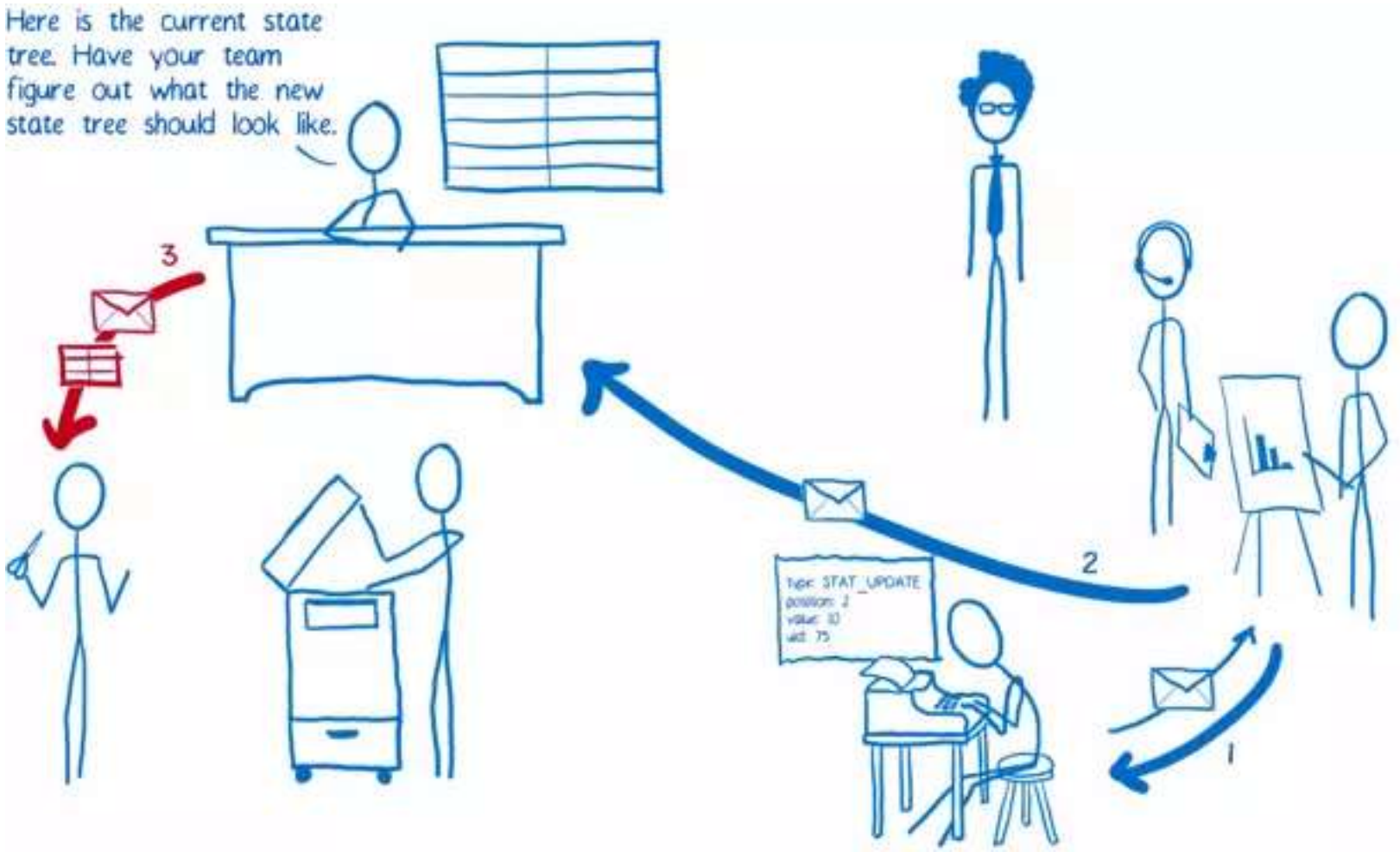
1. The View yêu cầu 1 action. Action Creator định dạng (format) yêu cầu và gửi lại

Flow



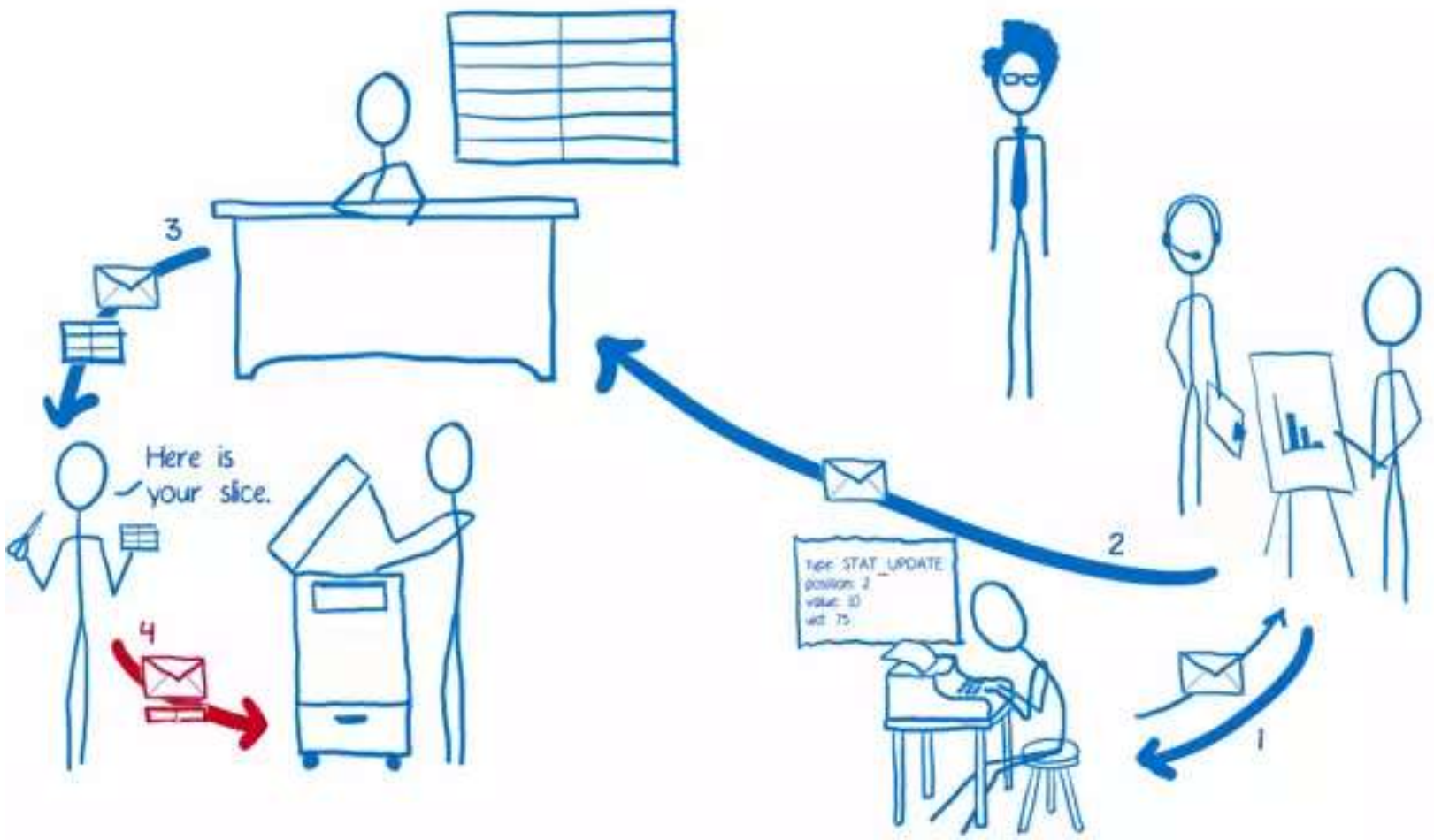
2. Action được gửi tự động (nếu bindActionCreators() đã được chuẩn bị) hoặc The View sẽ gửi

Flow



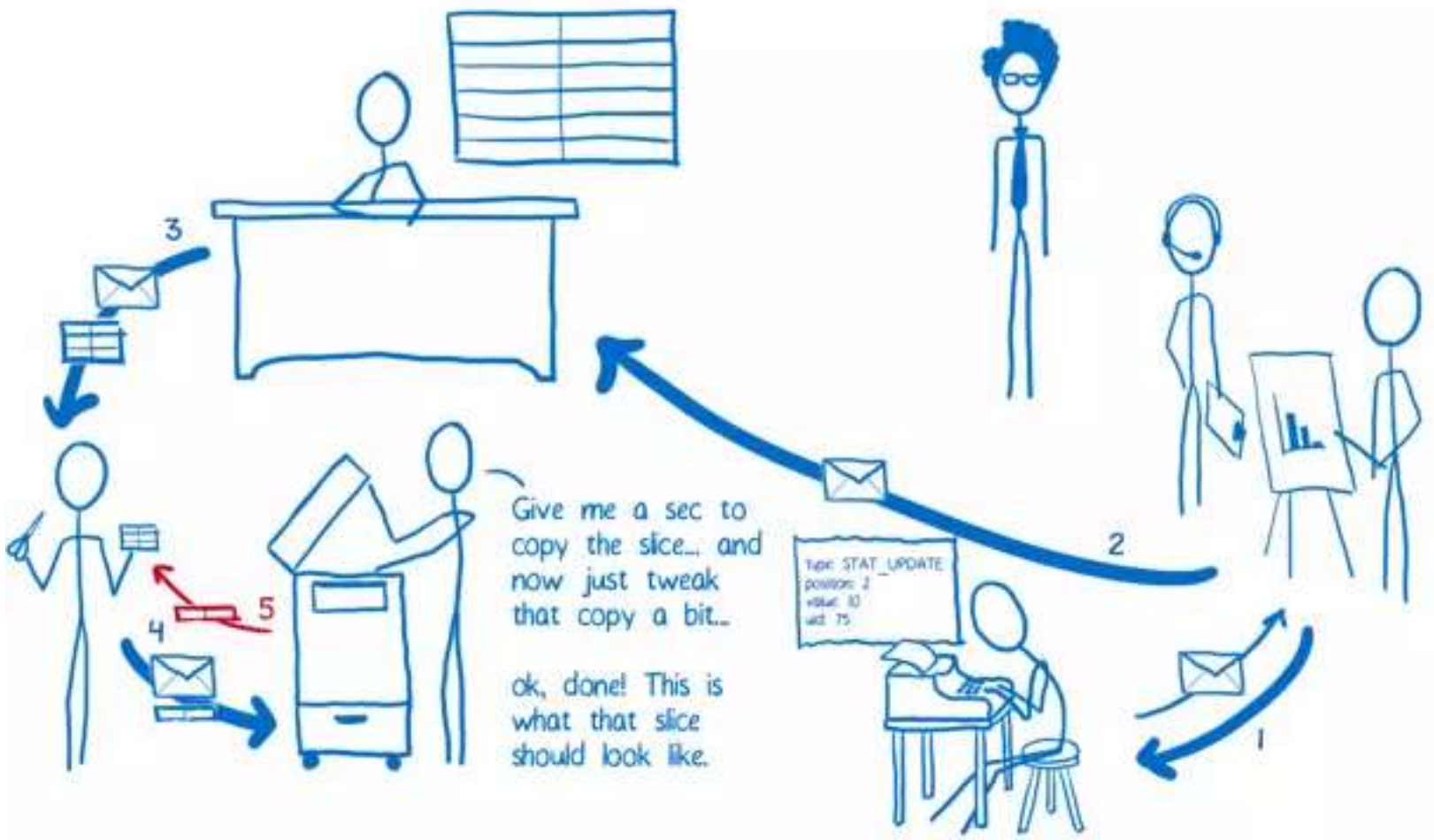
3. The Store nhận Action sau đó gửi State tree hiện tại và Action cho Root Reducer

Flow



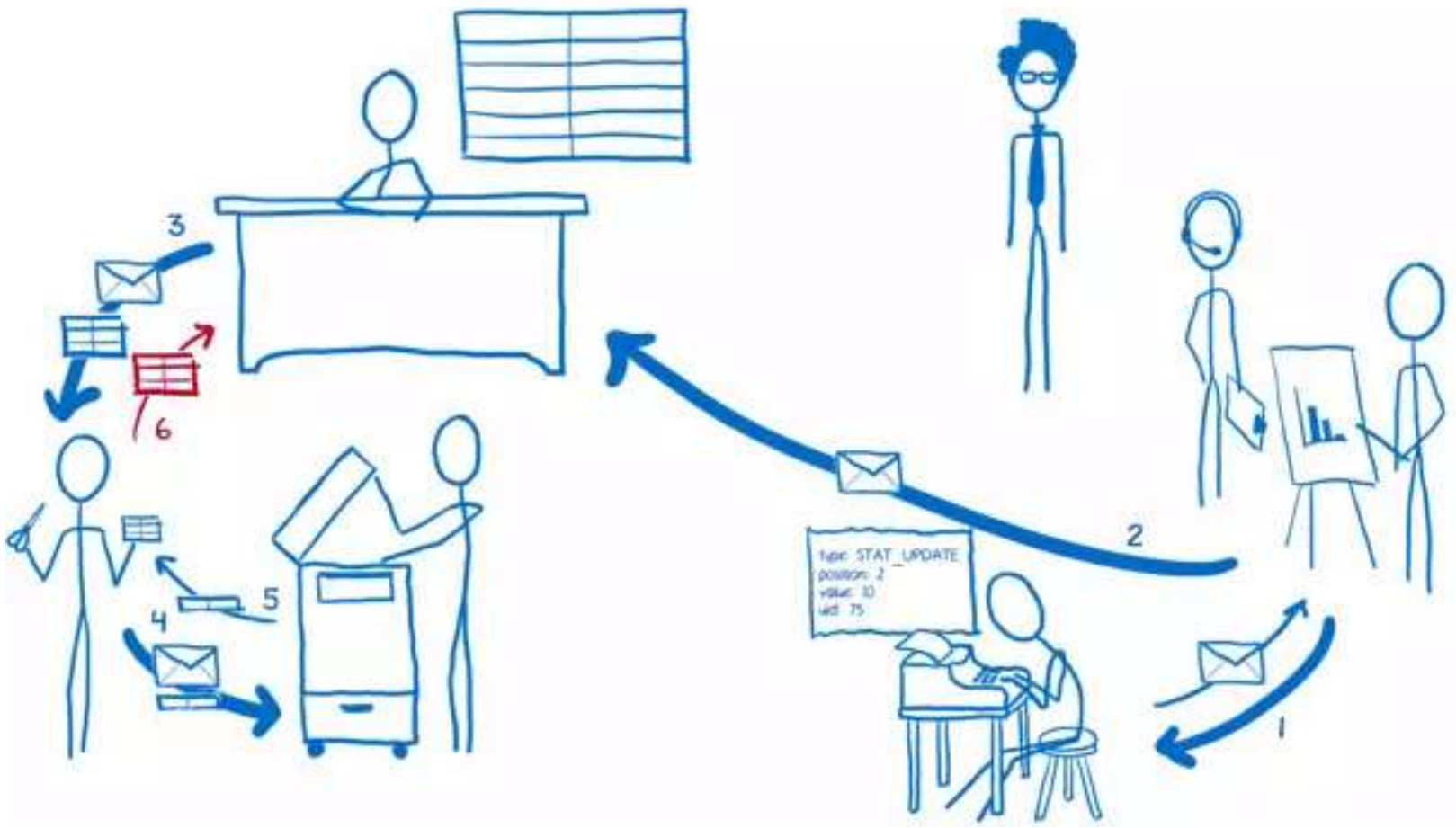
4. Root Reducer chia State ra thành nhiều phần và gửi cho từng subreducers biết cách xử lý chúng

Flow



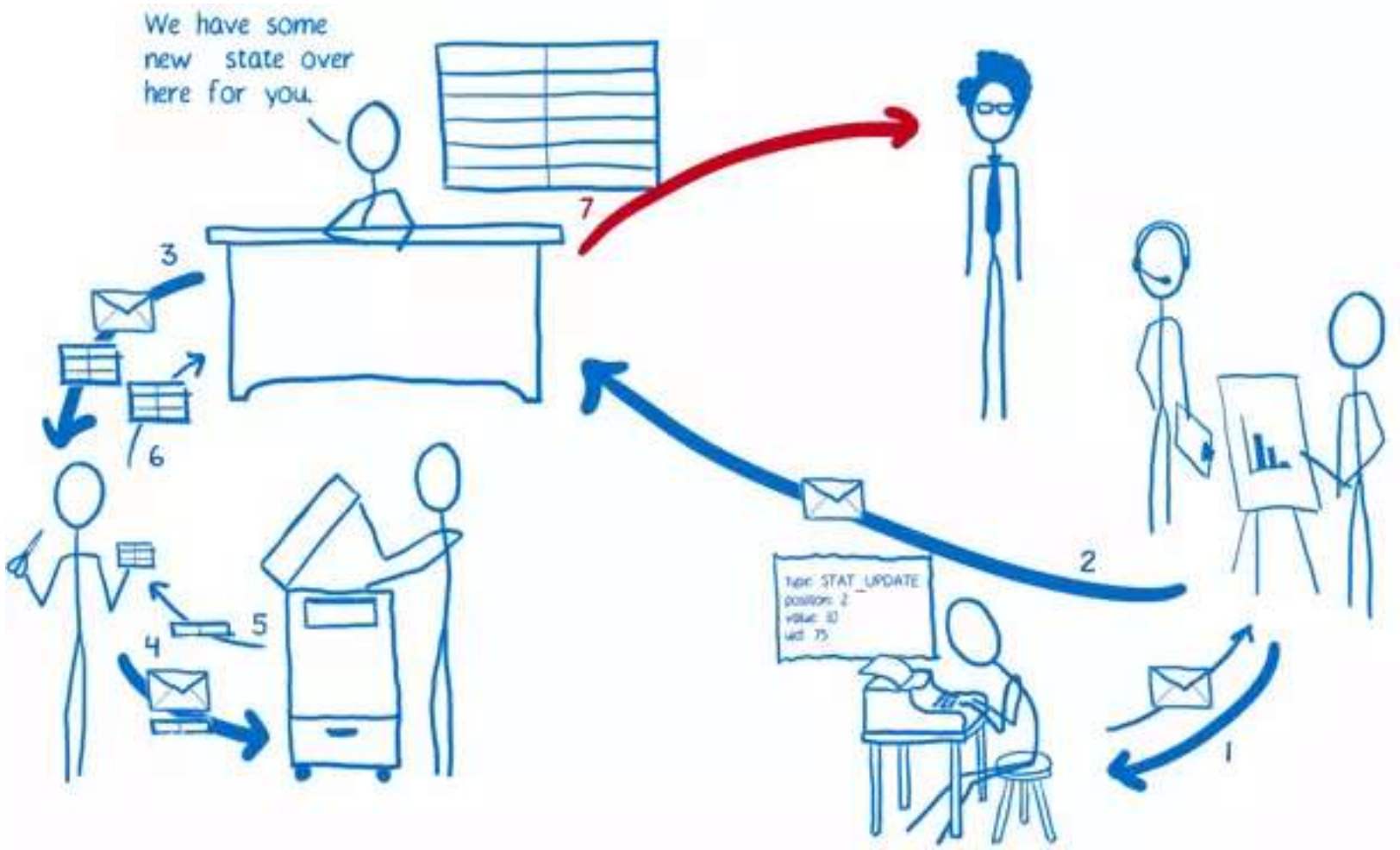
5. Subreducers tạo ra 1 bản copy từ phần nhận được và thay đổi trên bản copy. Sau đó gửi lại bản copy cho Root Reducer

Flow



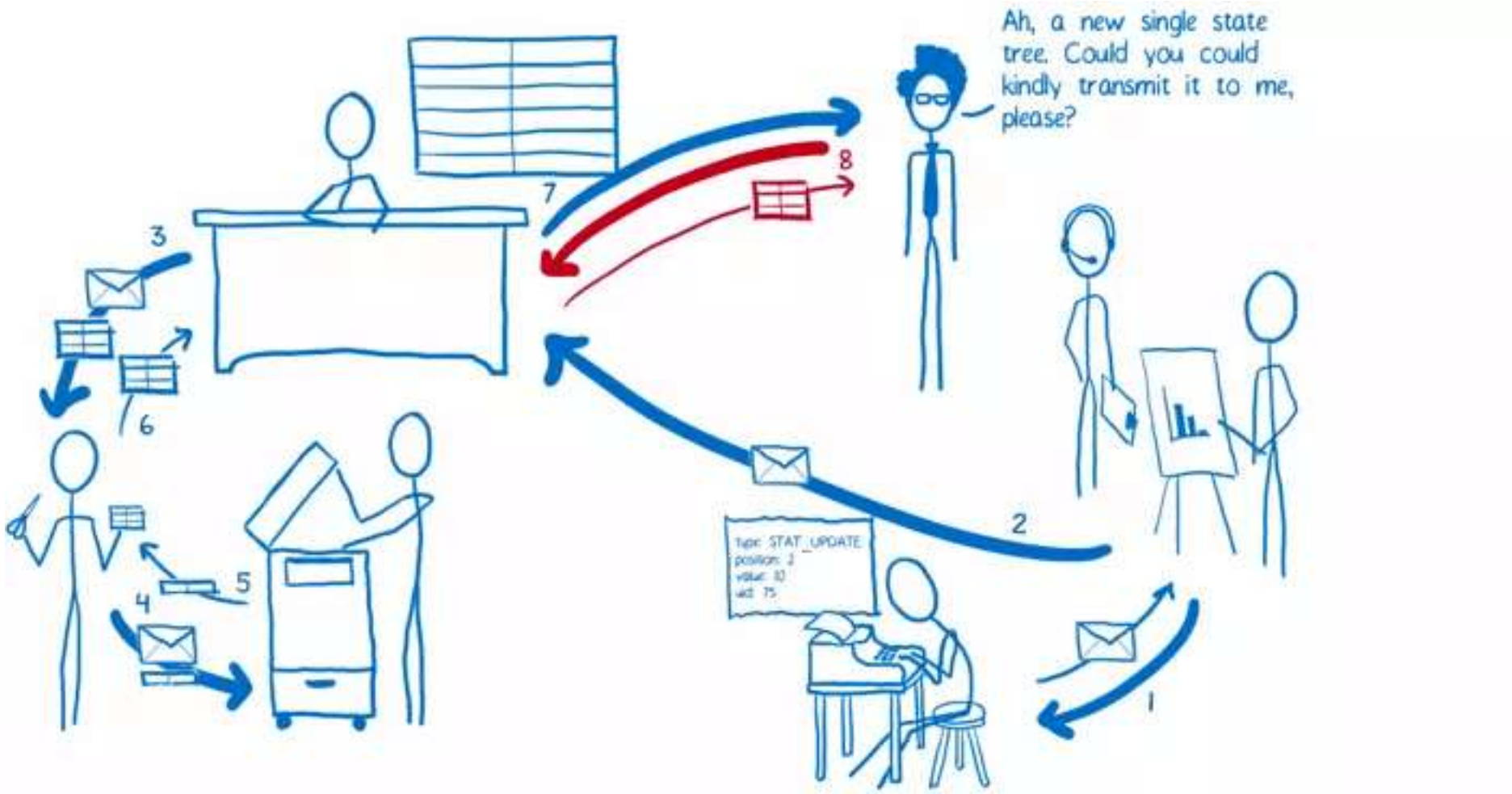
6. Khi tất cả subreducers trả về các phần copies, Root Reducer ghép chúng lại tạo thành 1 update State tree và gửi lại cho Store. Store thay thế State tree cũ bằng State tree mới.

Flow



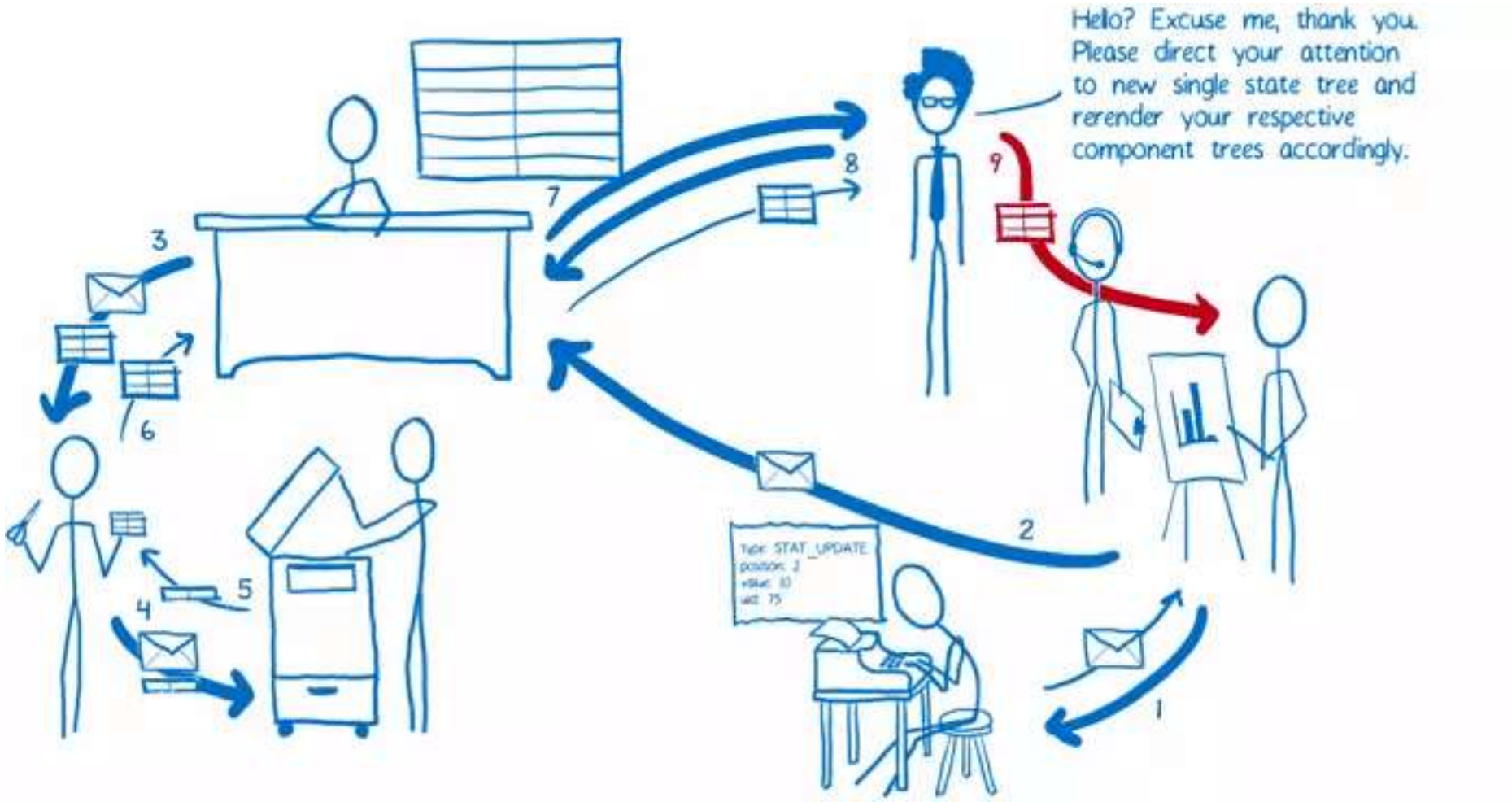
7. Store nói với The view layer binding là có State mới

Flow



8. The view layer binding báo Store gửi State mới cho mình

Flow



9. The view layer binding kích hoạt render view

CÁCH SỬ DỤNG REDUX CƠ BẢN

Một ví dụ về Redux

```
import { createStore } from 'redux'

// Step 1: Define a reducer
// A pure js function
// that transform the old state to the new one
// based on the action.type
function counter(state = 0, action) {
  switch (action.type) {
    case 'INCREMENT':
      return state + 1
    case 'DECREMENT':
      return state - 1
    default:
      return state
  }
}

// Step 2: Init your store with the reducer
// Its API is { subscribe, dispatch, getState }.
let store = createStore(counter)

// Step 3: Subscribe to state changes to update UI
store.subscribe(() => console.log(store.getState()))

// Step 4: Dispatch action to update redux state
// The only way to mutate the internal state is to dispatch an action.
store.dispatch({ type: 'INCREMENT' }) // 1
store.dispatch({ type: 'INCREMENT' }) // 2
store.dispatch({ type: 'DECREMENT' }) // 1
```

Một ví dụ về Redux: Counter Vanilla

Tóm tắt bài học

- Redux là một công cụ quản lý trạng thái có thể dự đoán được cho Javascript ứng dụng.
- Actions là các sự kiện, là cách mà chúng ta gửi dữ liệu từ ứng dụng đến store.
- Reducer là các hàm nguyên thuỷ lấy state hiện tại của ứng dụng, thực hiện một action và trả về state mới.
- Store lưu trạng thái của ứng dụng, và nó là duy nhất.
- Middleware React là lớp trung gian (giữa) Reducer và dispatch Action