**How they all work together**

Let’s see how these parts work together to create a functioning app.

**The setup**

The different parts of the app need to be wired up together. This happens in setup.

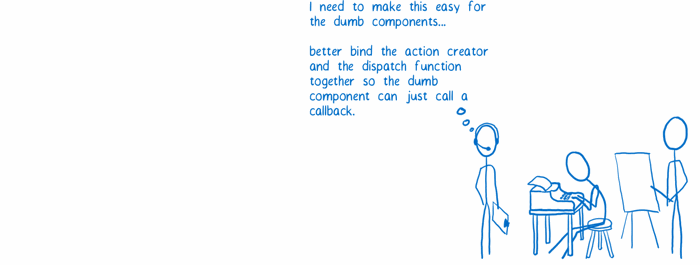
1. **Get the store ready.** The root component creates the store, telling it what root reducer to use, using createStore(). This root reducer already has a team of reducers which report to it. It assembled that team of reducers using combineReducers().



2. **Set up the communication between the store and the components.**The root component wraps its subcomponents with the provider component and makes the connection between the store and the provider.

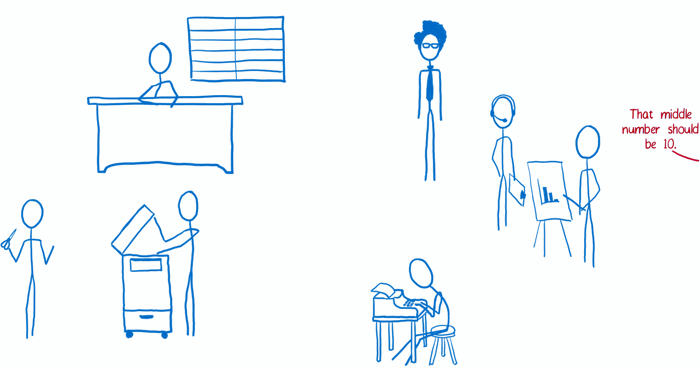
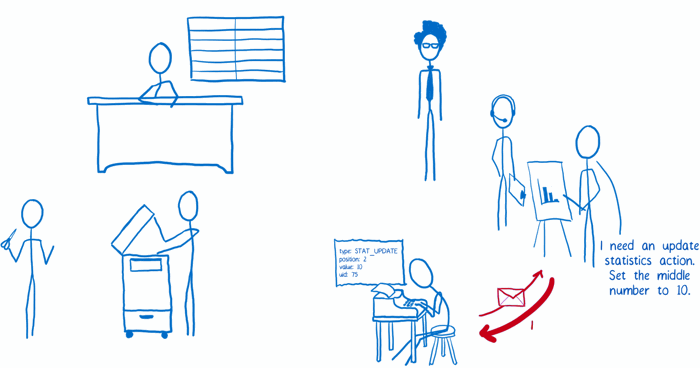
The Provider creates what’s basically a network to update the components. The smart components connect to network using connect(). This ensures they’ll get state updates.



3. **Prepare the action callbacks.** To make it easier for dumb components to work with actions, the smart components can setup action callbacks by using bindActionCreators(). This way, they can just pass down a callback to the dumb component. The action will be automatically dispatched after it is formatted.

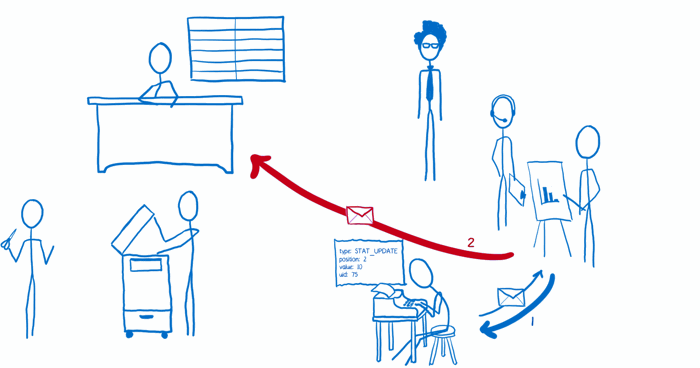
**The data flow**

Now that the application is set up, the user can start interacting with it. Let’s trigger an action to see the data flow.

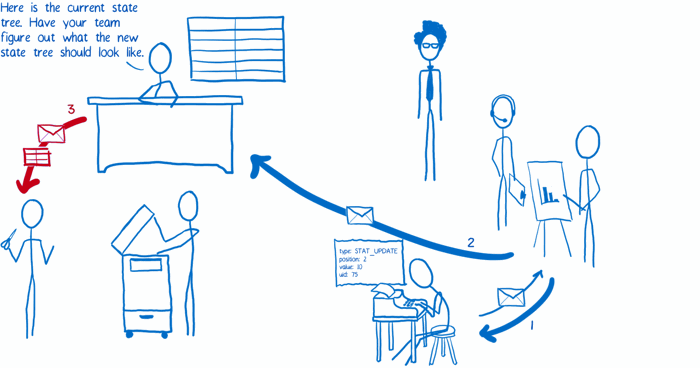


1. The view requests an action. The action creator formats it and returns it.

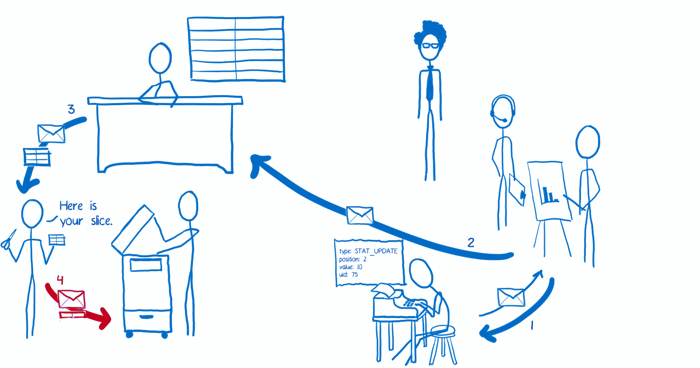
2. The action is either dispatched automatically (if bindActionCreators() was used in setup), or the view dispatches the action.



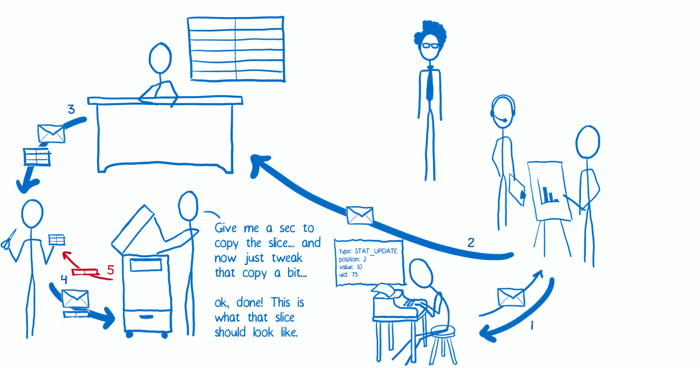
3. The store receives the action. It sends the current state tree and the action to the root reducer.



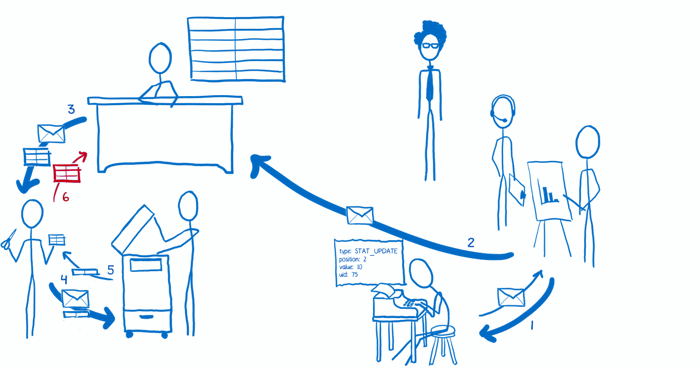
4. The root reducer cuts apart the state tree into slices. Then it passes each slice to the subreducer that knows how to deal with it.



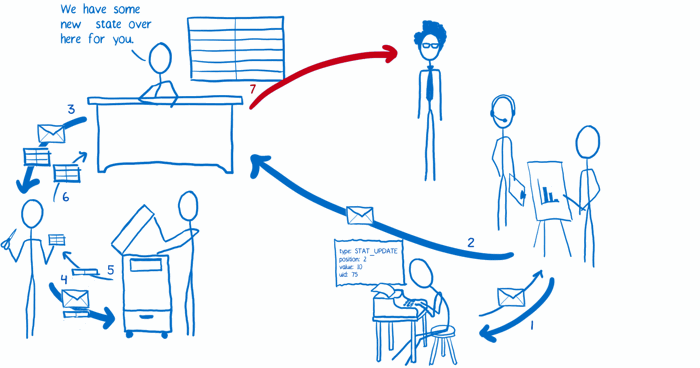
5. The subreducer copies the slice and makes changes to the copy. It returns the copy of the slice to the root reducer.



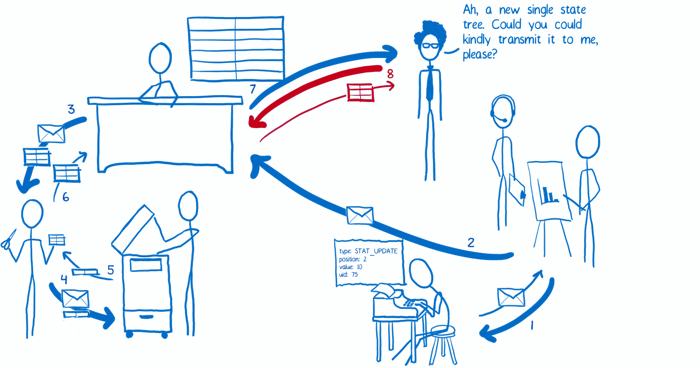
6. Once all of the subreducers have returned their slice copies, the root reducer pastes all of them together to form the whole updated state tree, which it returns to the store. The store replaces the old state tree with the new one.



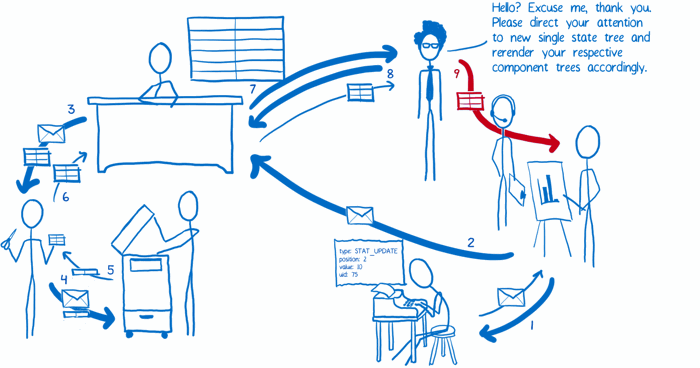
7. The store tells the view layer binding that there’s new state.



8. The view layer binding asks the store to send over the new state.



9. The view layer binding triggers a rerender.



So that’s how I think of Redux and its differences from Flux. Hope it helps!