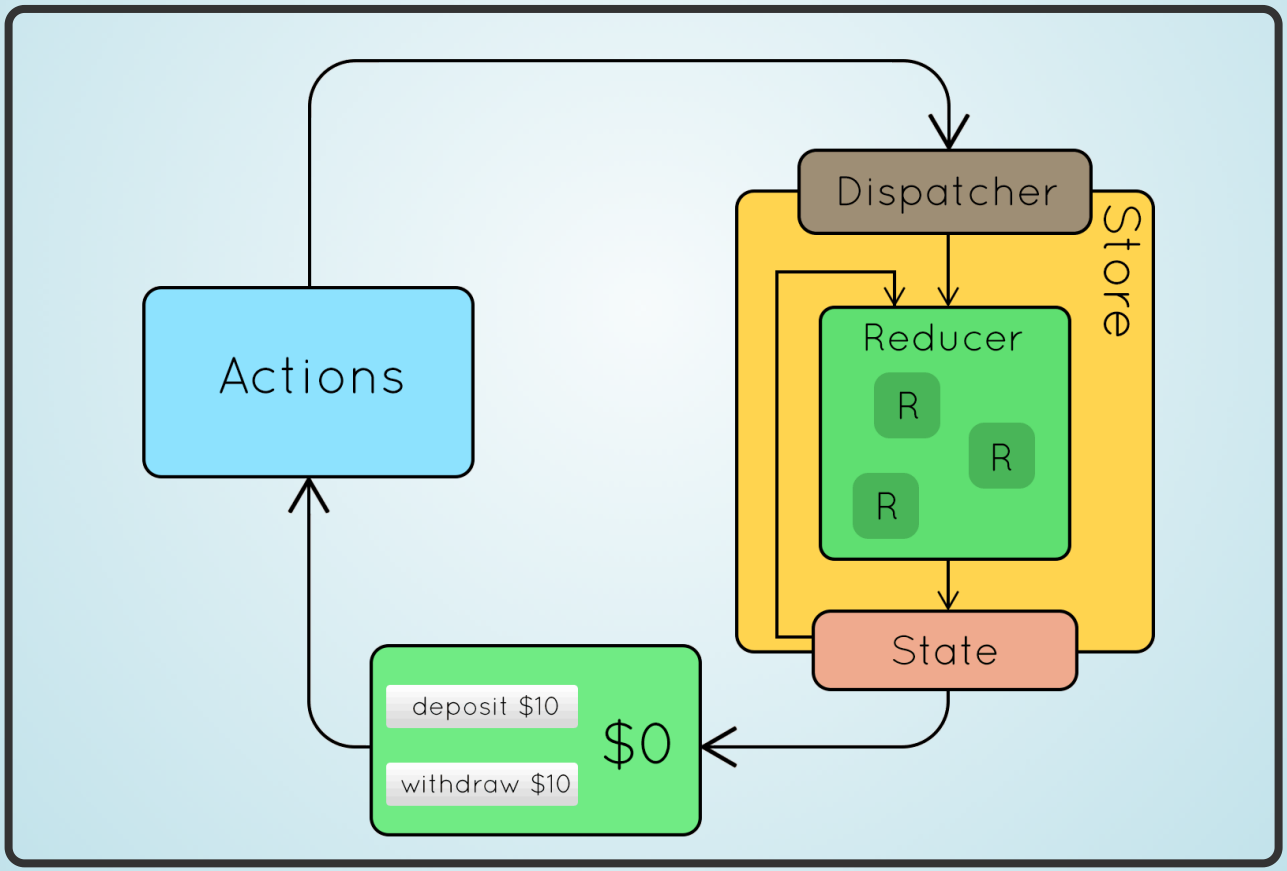
REDUX


Actions
Dispatcher
Store
Reducer
R
R
R
State
deposit $10
withdraw $10
$0

Discovered

Redux was created by Dan Abramov around June 2015. It was inspired by Facebook's Flux. **Redux** got popular very quickly because of its simplicity, small size (only 2 KB) and great documentation

smashingmagazine.com

What is Redux

1. Redux is a pattern and library for managing and updating application state, using events called "actions".
2. It serves as a centralized store for state that needs to be used across your entire application, with rules ensuring that the state can only be updated in a predictable fashion.
3. Based on Flux which is an application architecture that used by Facebook for building client-side web applications. It complements React's composable view components by utilizing a unidirectional data flow.

What does it do?

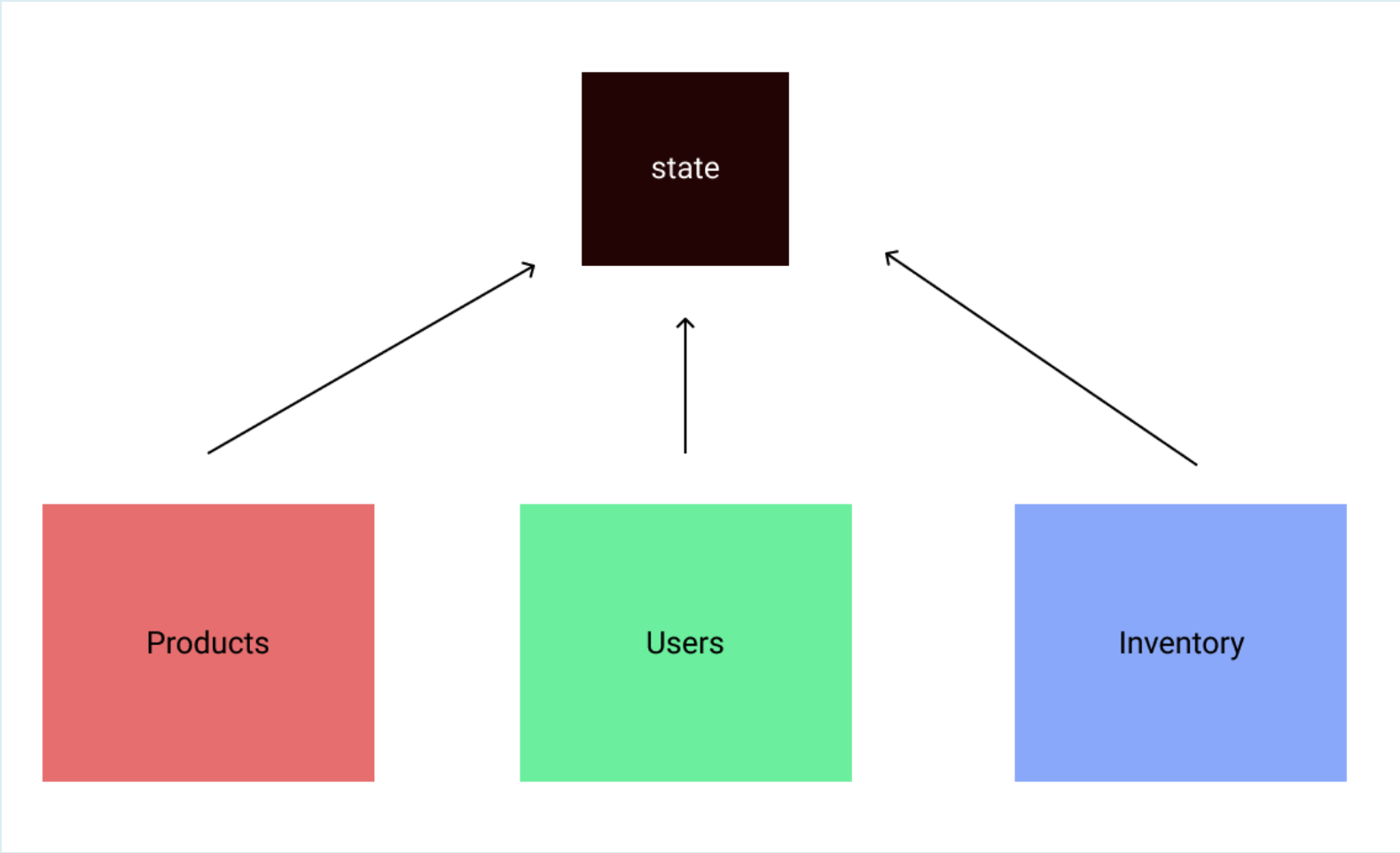
1. Redux helps you manage "global" state - state that is needed across many parts of your application.
2. Redux guides you towards writing code that is predictable and testable
3. The patterns and tools provided by Redux make it easier to reason about how your state is being update

Why Use It?

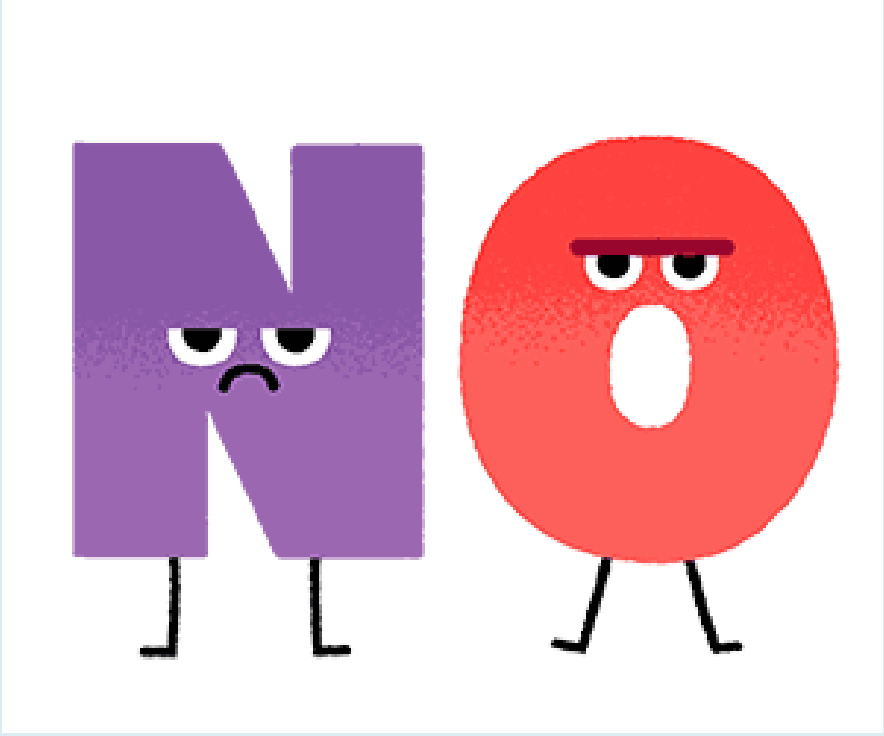
What are the tradeoffs:

1. It helps you deal with shared state management
2. There's more concepts to learn
3. Theres's more code to write.
4. It also adds some indirection to your code,
5. It asks you to follow certain restrictions.

Multiple Subjects Need the Same State

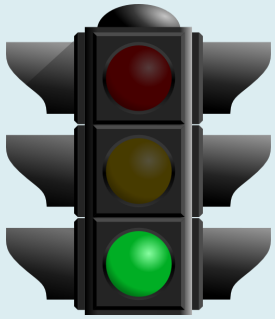


Is Redux part of React?



Always use Redux?

So when?



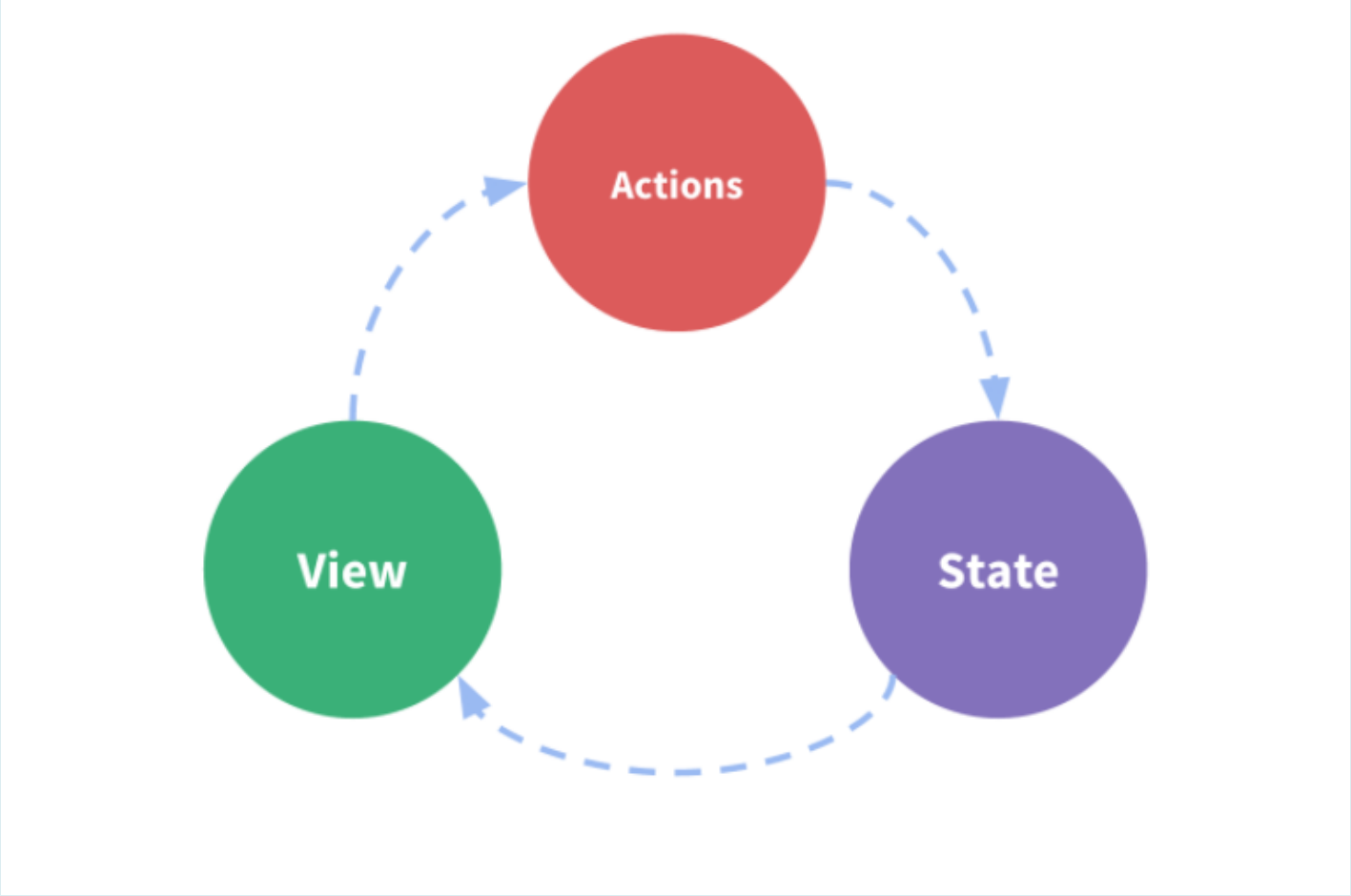
1. You have large amounts of application state that are needed in many places in the app
2. The app state is updated frequently over time
3. The logic to update that state may be complex
4. The app has a medium or large-sized codebase, and might be worked on by many people

* Not all apps need Redux.

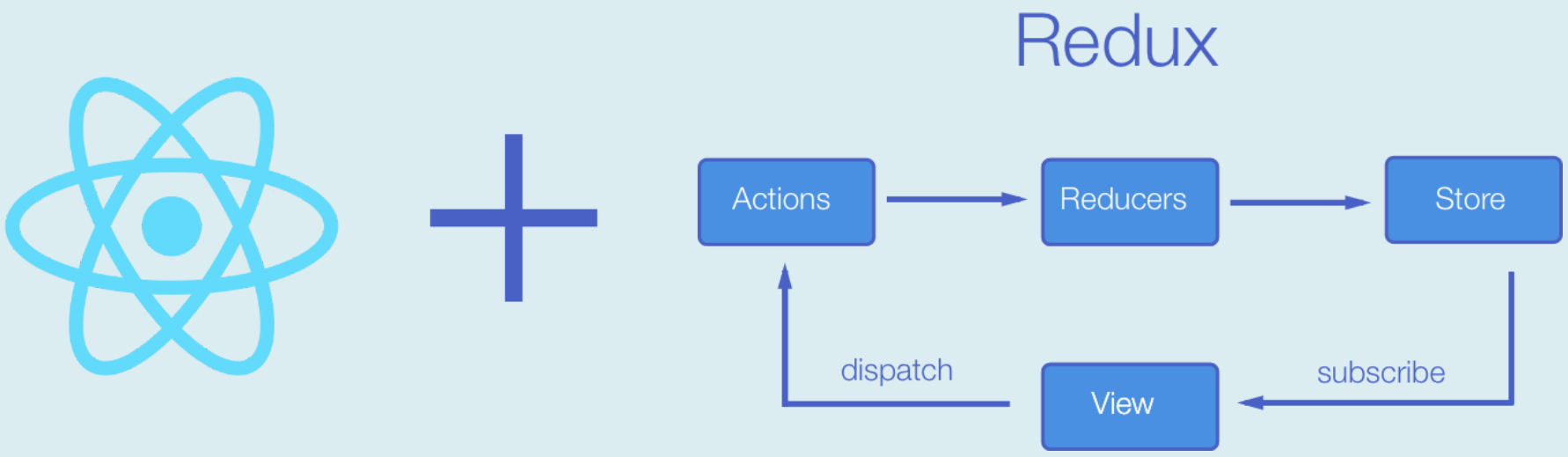
* Take some time to think about the kind of app you are building,

* Decide what tools would be best to help solve the problems you're working on.

React No Redux



Redux Flow



Store

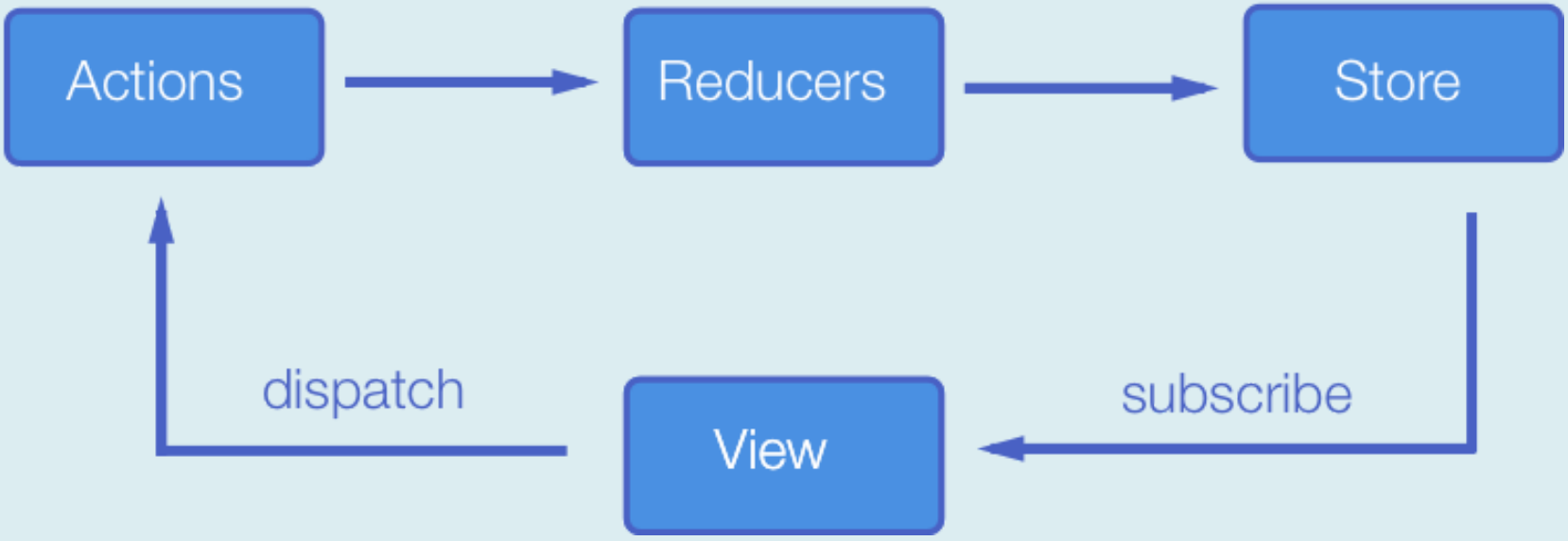
The center of every Redux application is the store.

A "store" is a container that holds your application's global state. Its data structure is a JavaScript object with a few special functions and abilities that make it different than a plain object literal.

Store Rules

- You must never directly modify or change the state that is kept inside the Redux store
- The only way to trigger an update to the state is to create a plain action object that describes "something that happened in the application", and then dispatch the action to the store to tell it what happened.
- When an action is dispatched, the store runs the root reducer function, and lets it calculate the new state based on the old state and the action
- Finally, the store notifies subscribers that the state has been updated so the UI can be updated with the new data.

Redux



Creating a Store

```
const {createStore} = redux
const store = createStore()
```

1. **store.getState()**
2. **store.dispatch()**
3. **store.subscribe()**

Actions

```
{ type: ADD, payload: num }
```

Actions are plain JavaScript objects that have a type field. You can think of an action as an event that tells the store to do something based on its type

Action Examples

(action verbs)

- Add a new todo entry based on the text the user entered
- Toggle the completed status of a todo
- Select a color category for a todo
- Delete a todo
- Mark all todos as completed
- Clear all completed todos
- Choose a different "completed" filter value
- Add a new color filter
- Remove a color filter

Actions MUST always have a type key,value pair

We can optionally have a payload property to deliver extra data needed to describe what's happening. This could be a number, a string, or an object with multiple fields inside. It is not required to be called payload

Actions should contain the smallest amount of information needed to describe what happened

Action Creators

It is another common convention that, instead of creating action objects inline in the places where you dispatch the actions, you would create functions generating them.

```
const todoAdded = ()={
    return{
        type: TODO_ADDED,
        payload: todoText
    }
}
```

Why Action Creators?

1. Action creators let you decouple additional logic around dispatching an action, from the actual components emitting those actions.
2. when the application is under heavy development and the requirements change often you don't have to search the whole application to make changes

Reducers

```
const rootReducer = (state=[], action) => {
    switch(action.type){
        case TODO_ADDED:
            return [...state, action.todoText]
    }
}
```

Reducers are functions that take the current state and an action as arguments, and return a new state result. In other words, (state, action) => newState

One Single Reducer

A Redux app really only has one reducer function:

the "root reducer" function that you will pass to createStore later on.

```
const {createStore} = redux
const store = createStore(rootReducer)
```

That one root reducer function is responsible for handling *all* of the actions that are dispatched, and calculating what the *entire* new state result should be every time.

Reducers must *always* follow some special rules:

- They should only calculate the new state value based on the state and action arguments
- They are not allowed to modify the existing state. Instead, they must make *immutable updates*, by copying the existing state and making changes to the copied values.
- PURE FUNCTIONS...They must not do any asynchronous logic or produce "side effects"

Side Effect???

A "side effect" is any change to state or behavior that can be seen outside of returning a value from a function. Some common kinds of side effects are things like:

- **Logging a value to the console**
- **Saving a file**
- **Setting an async timer**
- **Making an AJAX HTTP request**
- **Modifying some state that exists outside of a function, or mutating arguments to a function**
- **Generating random numbers or unique random IDs (such as Math.random() or Date.now())**

Any function that follows these rules is also known as a "pure" function, even if it's not specifically written as a reducer function.

LETS CODE