Hello Redux

tl;dr

- Redux is a JavaScript library that tames state management in our applications
- It's not super easy to learn; you have to understand state, pure functions, immutability, and composition.
- Redux is made up of
  - The store
  - Subscriptions
  - Actions
  - Reducers (state-changer functions)
- You dispatch actions to a reducer which generates a new state object -- a very controlled process

It's a library

Let's get this out of the way. Redux is ...

a JavaScript library

- Written by Dan Abramov
- If you want to use redux, include it in your JavaScript program.
- Client-side or server-side? Yes. Either.

You install redux via npm

```
$ npm install --save redux
+ redux@10.3.2
added 3 packages in 6.318s
$
```

You have to include it

```
<script src="redux.js"></script>
```

- Or hopefully you have webpack or another module loader

Why use Redux?

Philosophy behind it

SRP

The single responsibility principle is a computer programming principle that states that every module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class. All its services should be narrowly aligned with that responsibility. Robert C. Martin expresses the principle as, "A class should have only one reason to change."

- From wikipedia

- Redux manages all of the data and access to read/write that data.

SRP simplifies things!

- The cost you pay is you have to learn this tool.

Redux has concepts we need to know

1. State
2. Pure functions
3. Immutability
4. Composition

Concept 1: State

Application state

- A lightswitch can have two states - The "on" state and "off" state
- A color can have 16,777,216 states:

```
{
  red: 0-256,
  green: 0-256,
  blue: 0-256
}
```

- An auto manufacturing plant can have 345,673,568,233,545 states:

```
{
workers with names, addresses, phone numbers, etc
cars with statuses, makes, colors, engines, etc.
balance sheet
assembly line machines
weather conditions,
}
```

The more complex a state becomes, the harder it is ...

to control,

to debug,

to organize

This is what Redux simplifies for you!

Concept 2: Pure functions

- Pop quiz: What does this mean?

```
y = f(x)
```

- Y is a function of X.
- But what does that mean?
- When x changes, y changes in a predictable way.

- Always produce X result from Y input.
- Never change anything external
- Never read from anything external

This is a pure function!

Pure

```
function square(x) {
  return x * x;
}
Math.sin(y);
arr.map((item) => item.id);
```

Impure

```
function get(id) {
  fetch(url, {id}).then(p => person=p);
}
Math.random(y);
```

Functional programming ideas are at the root of Redux

- NOT OBJECT-ORIENTED
- You Java devs are not going to love this.
- You JavaScript devs are!

In redux, state is only changed thru a pure function called a "reducer"

```
NewState = f(oldState, action, payloadData);
```

Concept 3: Immutability

- Something that is immutable won't change.
- In Redux, state must be treated as immutable.

- So how do you change things then?
- We allow redux to change state for us as it runs our reducer function.

Concept 4: Composition

Composition

- When you create an "action" that is made up of small, leaf-level activities.
- We use this with ...
- reducers
- actions
- enhancers

- Functions should be super-small and super-simple.

The parts of Redux

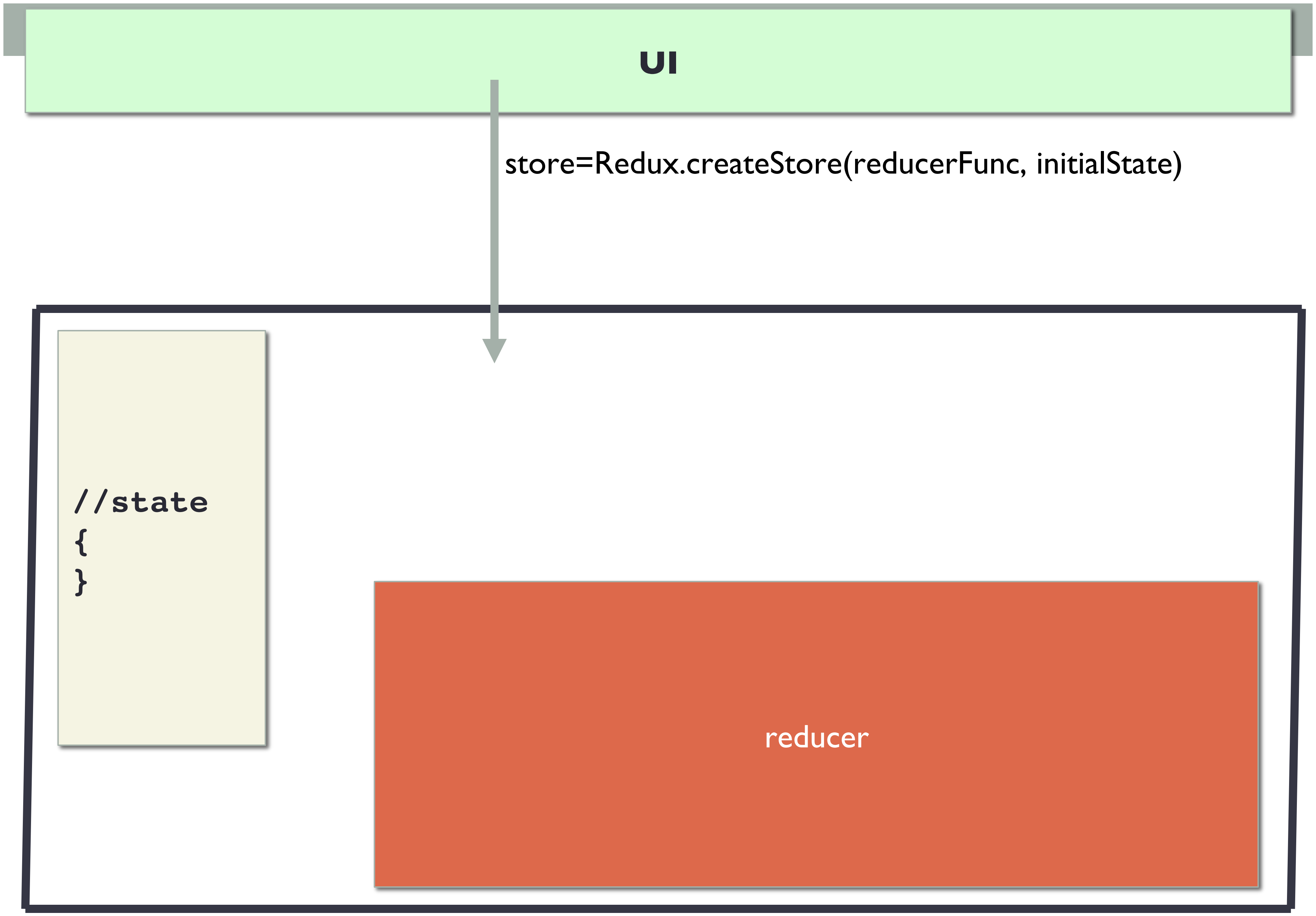
state, store, listeners, actions, and reducers

The parts of Redux

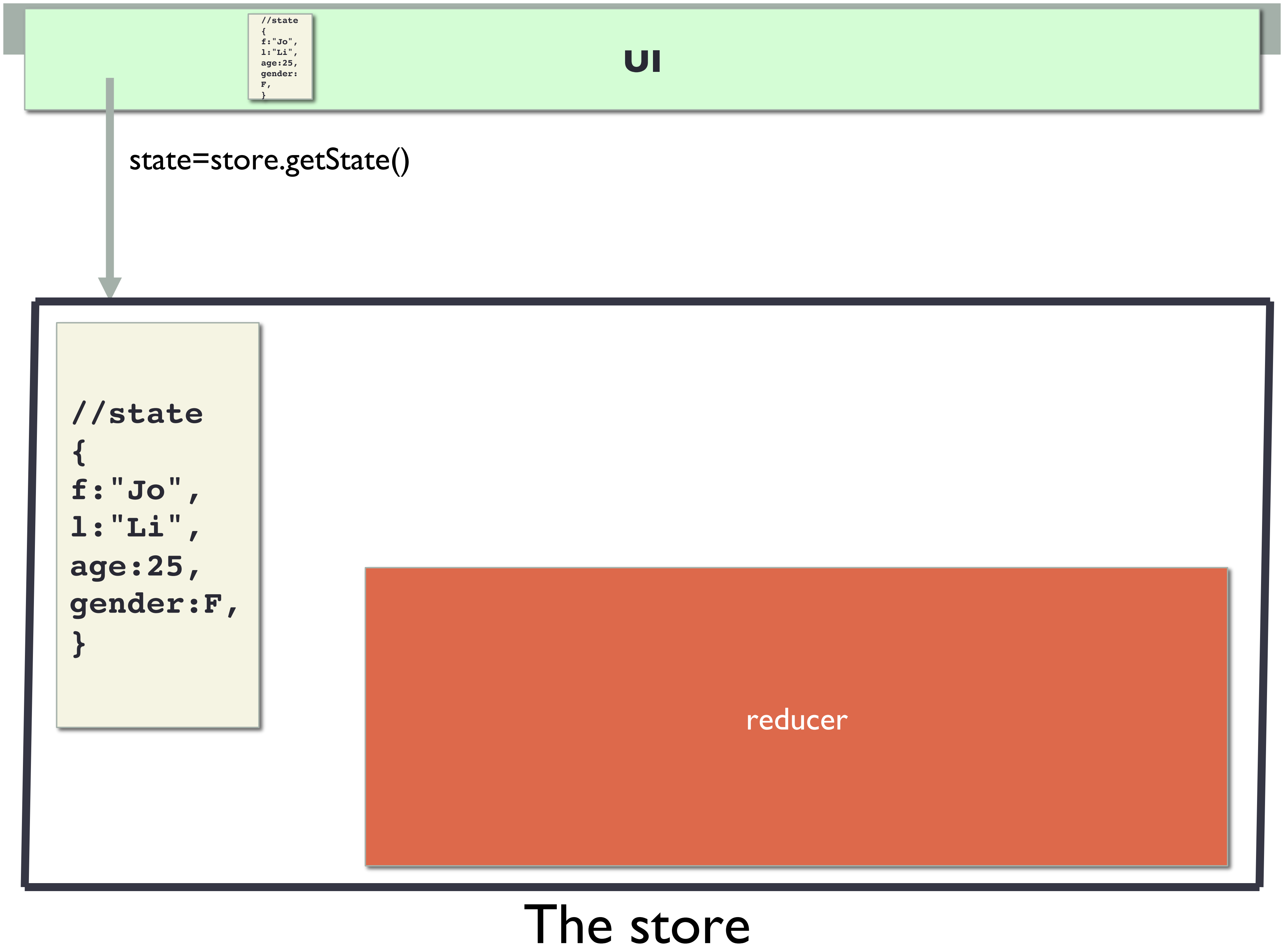
1. The store
2. Subscriptions
3. Actions
4. Reducers

Part 1: The store

- A single object which encapsulates the state of your application
- State is a single JSON object
- No functions. No logic.
- Reading state
- Altering state
- Principle: You must NEVER change state directly


UI
store=Redux.createStore(reducerFunc, initialState)
//state
{
}
reducer

The store



```
//state
{
f:"Jo",
l:"Li",
age:25,
gender:F,
}
```

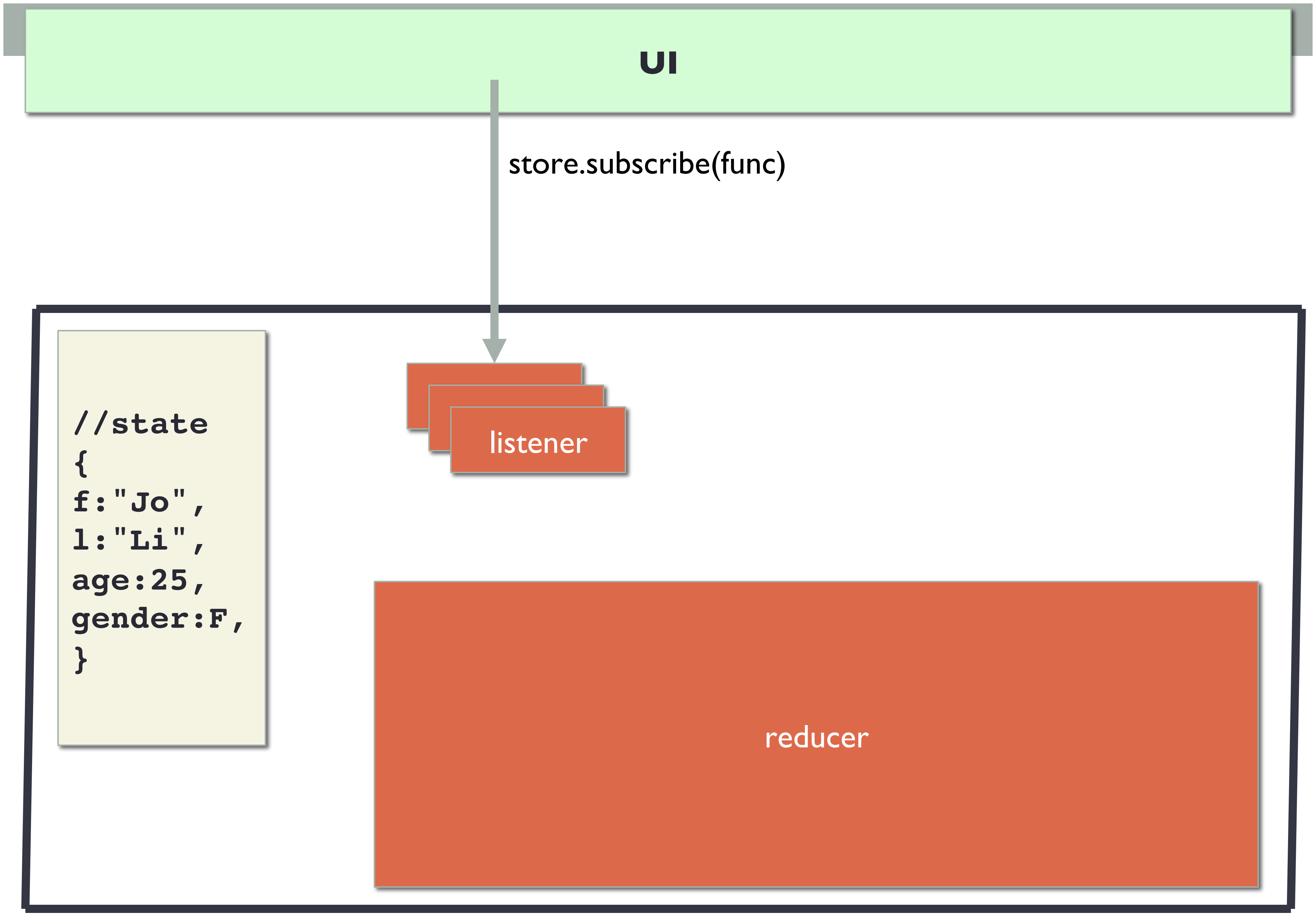
```
//state
{
f:"Jo",
l:"Li",
age:25,
gender:
F,
}
```

Part 2: Subscriptions

- When data changes, you want your app to respond.
- ie. re-render the UI so the new data appears.

- To make that happen, you can register a JavaScript callback to be run whenever any dispatch happens.
- This is called *subscribing*

- The store maintains a list of subscriptions and automatically runs them when a dispatch happens



```
//state
{
f:"Jo",
l:"Li",
age:25,
gender:F,
}
```

The store

Part 3: Actions

- An structure that describes the change to be made to the state.
- Always has a type
  - A string
  - One member of a finite pre-defined set
  - Kind of like an enum
- Usually has a payload
  - The data that should be changed
- Examples:

```
const action = {type:"HAVE_A_BIRTHDAY"};
const action = {
  type:"SET_COURSE",
  heading: "207-mark-99",
  speed:"warp 1.0"
};
```

There is a **finite** list of things that can be done to change state.
That finite list defines all of your actions

Actions are always *dispatched* to change state.

Dispatching sends an action to the reducer

Part 4: Reducers

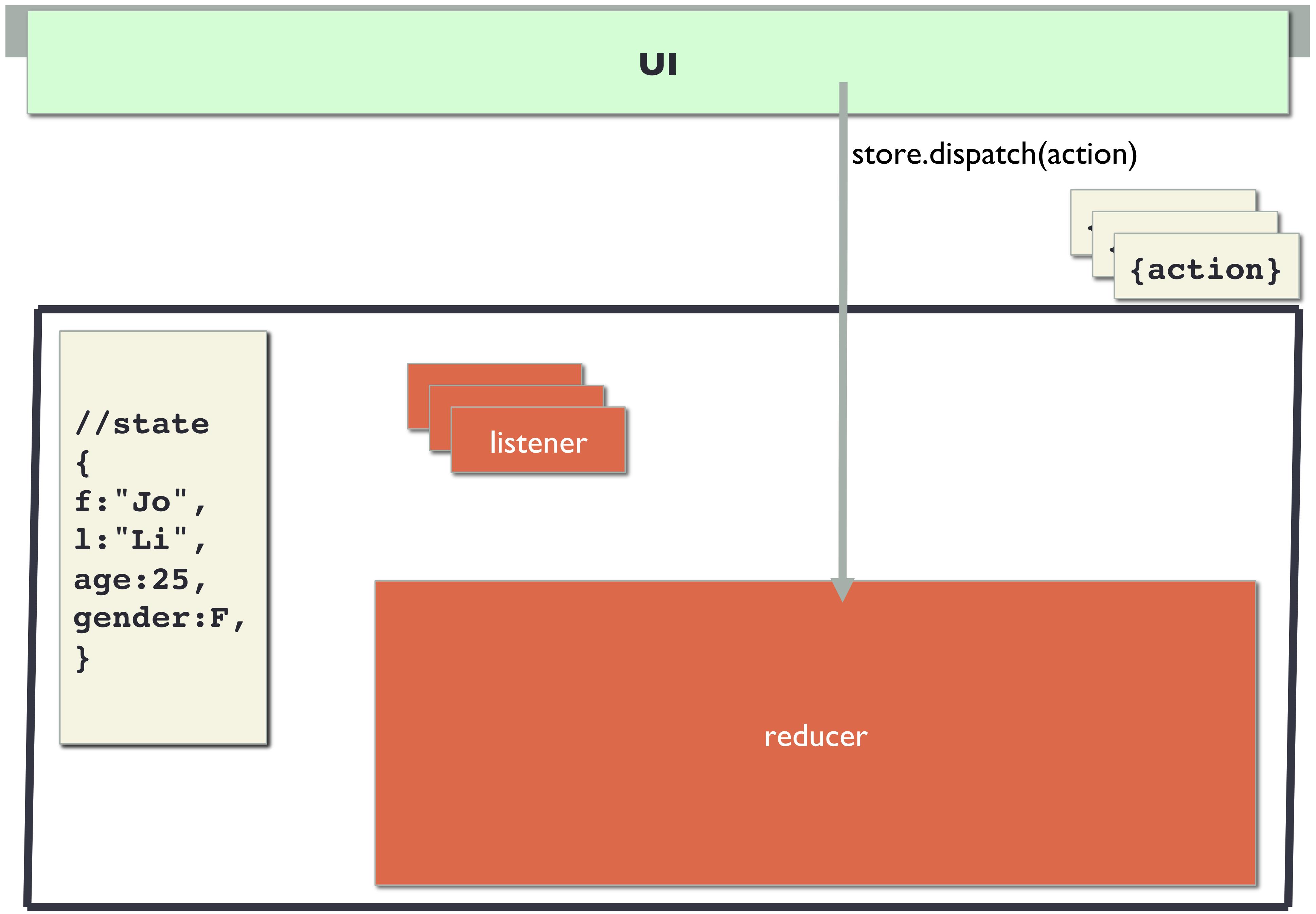
- Reducers are functions that know how to mutate (change) state, once given an action.

```
NewState = fn(oldState, action);
```

- We activate the reducer by *dispatching* an action.
- A pure and synchronous function! Thus it cannot change state. It can read the oldState as a parameter and will return a copy of that state that is different by the action.
- There will always be one case for each action.

Behind the scenes, Redux does something like this:

```
function dispatch(action) {
  const newState = reducer(state, action);
  state = newState;
  for (let func of subscriptions)
    func()
}
```



```
//state
{
f:"Jo",
l:"Li",
age:25,
gender:F,
}
```

The store

There's more!

- But this gives us plenty to think about for now.
- More to come later.

tl;dr

- Redux is a JavaScript library that tames state management in our applications
- It’s not super easy to learn; you have to understand state, pure functions, immutability, and composition.
- Redux is made up of
  - The store
  - Subscriptions
  - Actions
  - Reducers (state-changer functions)
- You dispatch actions to a reducer which generates a new state object -- a very controlled process