Reducer Composition

A theoretical view of how each reducer is assigned to a single component of the state object. The use of multiple reducers is known as Reducer Composition.

In all the apps we've create so far, we have used just one reducer to manage the entire state of the applications.

What's the implication of this?

It is like having just one Cashier in the entire bank hall. How scalable is that?

Even if the Cashier can do all the work effectively, it may be more manageable (and perhaps a better customer experience) to have more than one Cashier in the bank hall.

Someone's got to attend to everybody, and it's a lot of work for just one person! The same goes with your Redux applications.

It is common to have multiple reducers in your application as opposed to one reducer handling all the operations of the state. These reducers are then combined into one.

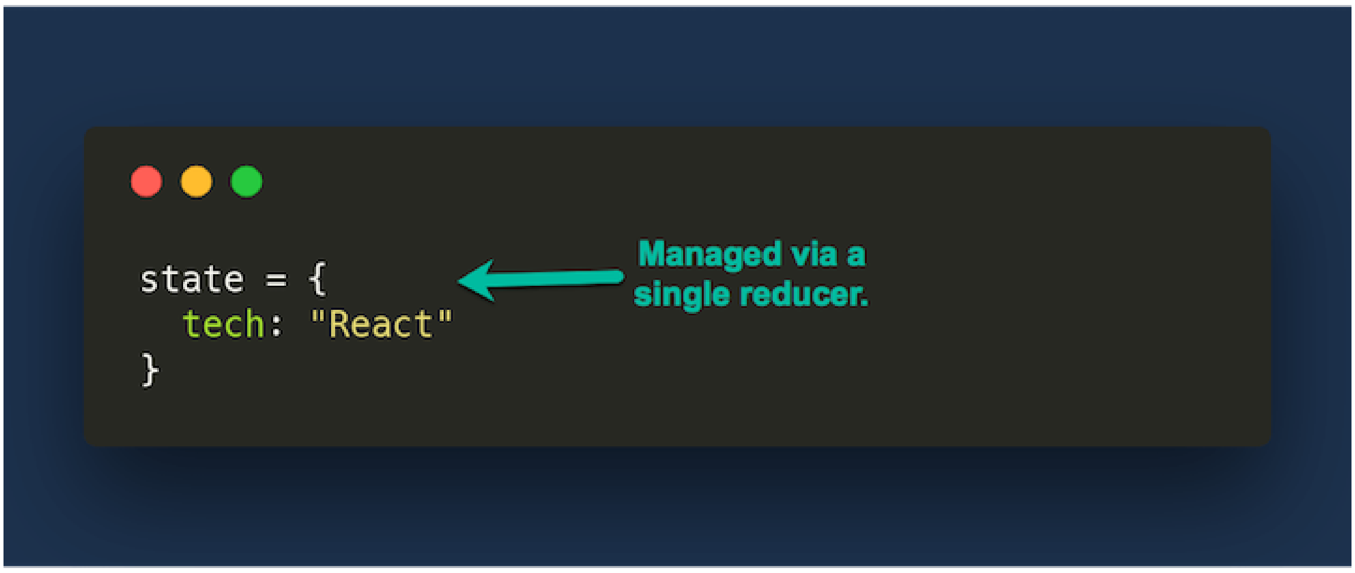
For example, there could be 5 or 10 Cashiers in the bank hall, but all of them combined all serve one purpose. That's how this works as well.

Consider the state object of the Hello World app we built earlier.

```
{
    tech: "React"
}
```

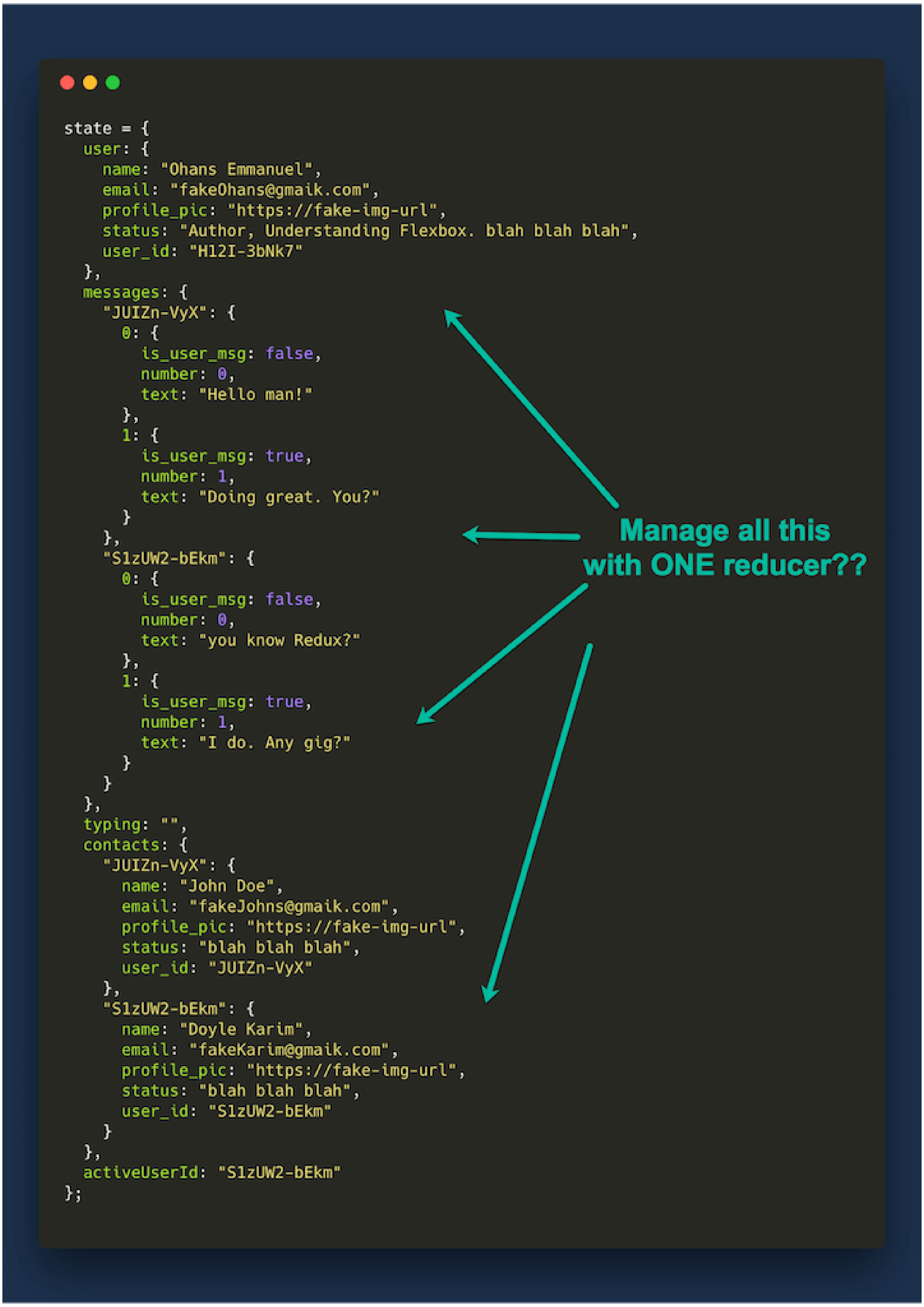
Pretty simple.

All we did was have ONE reducer manage the entire state updates.



```
state = {
  tech: "React"
}
```

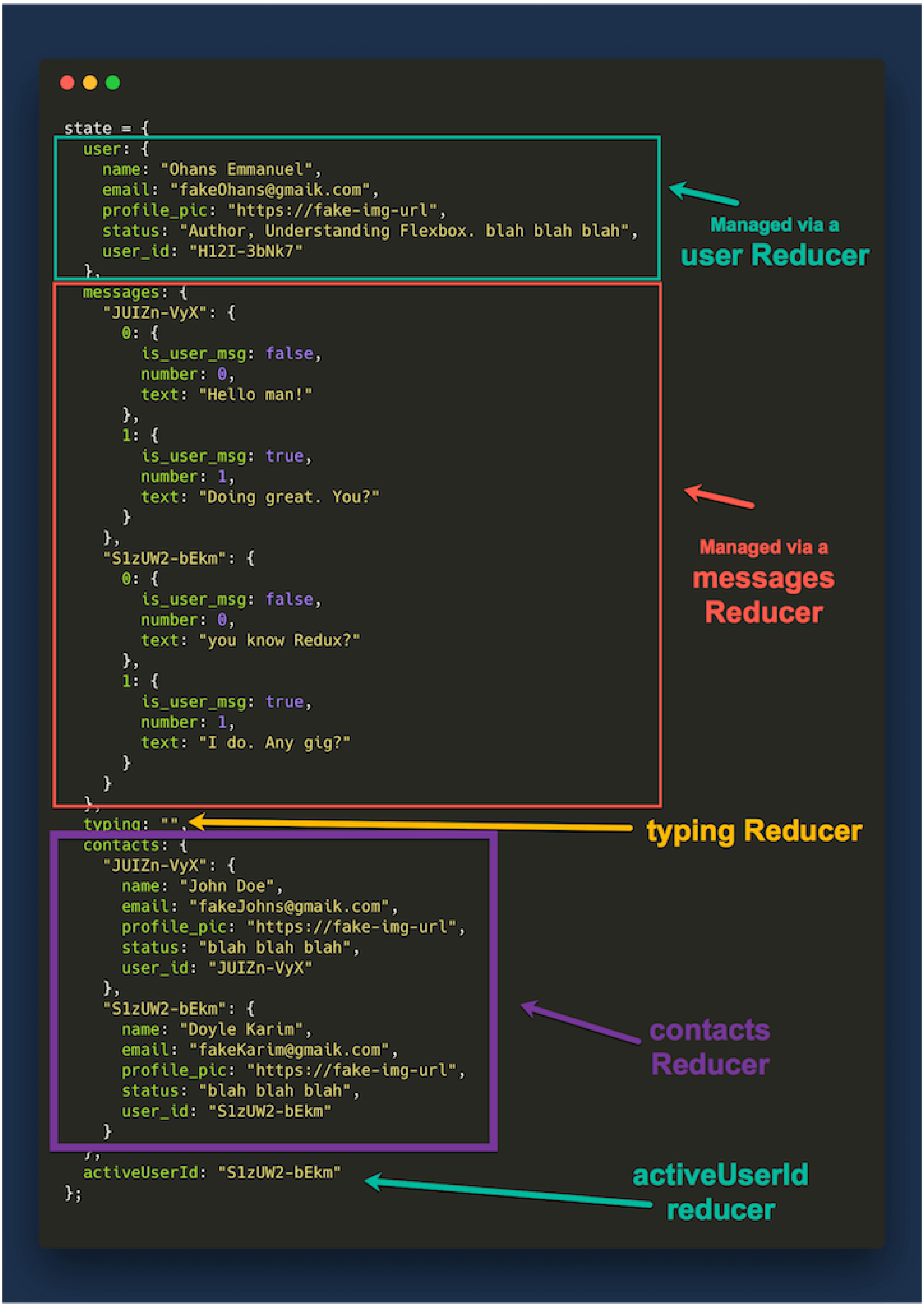
As for the state object of the more complex *Skypey* application, having a single reducer manage the entire state object is doable - but not the best approach.



```
state = {
  user: {
    name: "Ohans Emmanuel",
    email: "fakeOhans@gmaik.com",
    profile_pic: "https://fake-img-url",
    status: "Author, Understanding Flexbox. blah blah blah",
    user_id: "H12I-3bNk7"
  },
  messages: {
    "JUIZn-VyX": {
      0: {
        is_user_msg: false,
        number: 0,
        text: "Hello man!"
      },
      1: {
        is_user_msg: true,
        number: 1,
        text: "Doing great. You?"
      }
    },
    "S1zUW2-bEkm": {
      0: {
        is_user_msg: false,
        number: 0,
        text: "you know Redux?"
      },
      1: {
        is_user_msg: true,
        number: 1,
        text: "I do. Any gig?"
      }
    }
  },
  typing: "",
  contacts: {
    "JUIZn-VyX": {
      name: "John Doe",
      email: "fakeJohns@gmaik.com",
      profile_pic: "https://fake-img-url",
      status: "blah blah blah",
      user_id: "JUIZn-VyX"
    },
    "S1zUW2-bEkm": {
      name: "Doyle Karim",
      email: "fakeKarim@gmaik.com",
      profile_pic: "https://fake-img-url",
      status: "blah blah blah",
      user_id: "S1zUW2-bEkm"
    }
  },
  activeUserId: "S1zUW2-bEkm"
};
```

Instead of having the entire object managed by one reducer, what if we had ONE reducer manage ONE field in the state object?

Like a one to one mapping?



```
state = {
  user: {
    name: "Ohans Emmanuel",
    email: "fakeOhans@gmaik.com",
    profile_pic: "https://fake-img-url",
    status: "Author, Understanding Flexbox. blah blah blah",
    user_id: "H12I-3bNk7"
  },
  messages: {
    "JUIZn-VyX": {
      0: {
        is_user_msg: false,
        number: 0,
        text: "Hello man!"
      },
      1: {
        is_user_msg: true,
        number: 1,
        text: "Doing great. You?"
      }
    },
    "S1zUW2-bEkm": {
      0: {
        is_user_msg: false,
        number: 0,
        text: "you know Redux?"
      },
      1: {
        is_user_msg: true,
        number: 1,
        text: "I do. Any gig?"
      }
    }
  },
  typing: "",
  contacts: {
    "JUIZn-VyX": {
      name: "John Doe",
      email: "fakeJohns@gmaik.com",
      profile_pic: "https://fake-img-url",
      status: "blah blah blah",
      user_id: "JUIZn-VyX"
    },
    "S1zUW2-bEkm": {
      name: "Doyle Karim",
      email: "fakeKarim@gmaik.com",
      profile_pic: "https://fake-img-url",
      status: "blah blah blah",
      user_id: "S1zUW2-bEkm"
    }
  },
  activeUserId: "S1zUW2-bEkm"
};
```

You see what we're doing there? Introducing more Cashiers!

Reducer composition requires that a single reducer handles the state update for a single field in the state object.

For example, for the messages field, you have a messagesReducer. For a contacts field, you also have a contactsReducer and so on.

One more important thing to point out is that the return value from each of the reducer is solely for the field they represent.

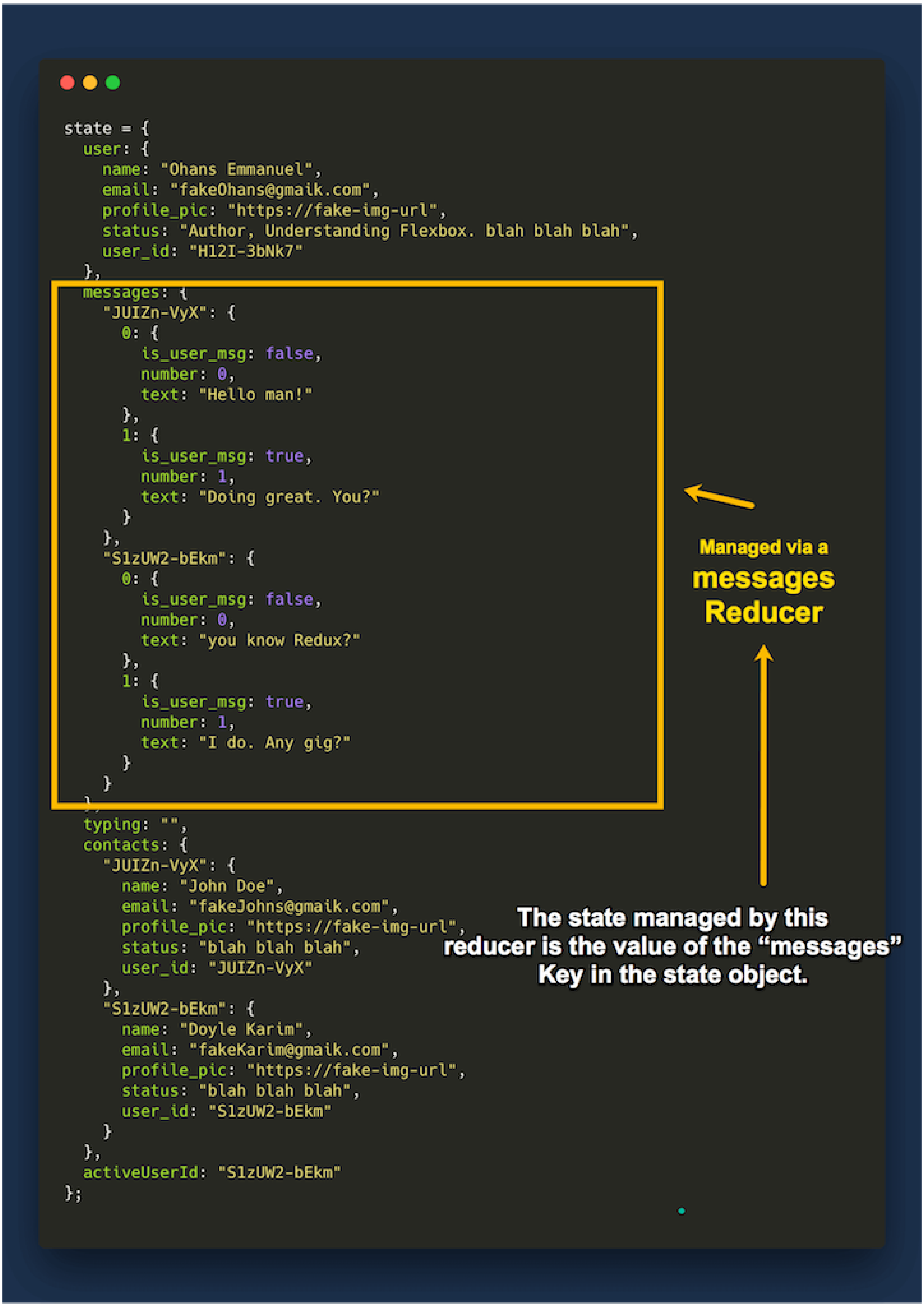
So, if I had messagesReducer written like this:

```
export const function messagesReducer (state={}, action) {
  return state
}
```

The state returned here is NOT the state of the entire application.

No.

It is only the value of the **messages** field.



```
state = {
  user: {
    name: "Ohans Emmanuel",
    email: "fakeOhans@gmaik.com",
    profile_pic: "https://fake-img-url",
    status: "Author, Understanding Flexbox. blah blah blah",
    user_id: "H12I-3bNk7"
  },
  messages: {
    "JUIZn-VyX": {
      0: {
        is_user_msg: false,
        number: 0,
        text: "Hello man!"
      },
      1: {
        is_user_msg: true,
        number: 1,
        text: "Doing great. You?"
      }
    },
    "S1zUW2-bEkm": {
      0: {
        is_user_msg: false,
        number: 0,
        text: "you know Redux?"
      },
      1: {
        is_user_msg: true,
        number: 1,
        text: "I do. Any gig?"
      }
    }
  },
  typing: "",
  contacts: {
    "JUIZn-VyX": {
      name: "John Doe",
      email: "fakeJohns@gmaik.com",
      profile_pic: "https://fake-img-url",
      status: "blah blah blah",
      user_id: "JUIZn-VyX"
    },
    "S1zUW2-bEkm": {
      name: "Doyle Karim",
      email: "fakeKarim@gmaik.com",
      profile_pic: "https://fake-img-url",
      status: "blah blah blah",
      user_id: "S1zUW2-bEkm"
    }
  },
  activeUserId: "S1zUW2-bEkm"
};
```

Managed via a messages Reducer
The state managed by this reducer is the value of the "messages" Key in the state object.

The same goes for the other reducers.

Got that?

Let’s see this in practice, and how exactly these reducers are combined for a single purpose.