Enhance your User (and Developer) Experience with React & Redux

& /phacks

Hi! I am Nicolas Goutay. I work at Theodo, a web consultancy based in Paris & London. I build JS & Python web applications. I ~~have stage fright~~ am excited to be here with all of you ☺️

You can find me online (Twitter & GitHub) on @phacks.

React & Redux

A Lego analogy

Lego tray

How to build a F1

React application code would be a Lego instruction manual, where bricks are DOM nodes. It takes care of how things look for the end user.

On modern Web apps, how things look are usually a function of user interactions. In this analogy, the user is Elya, my 5 year-old niece. Red is her favorite color, so she wants the car to be red 🏎️.

This is where 🤖 Redux kicks in.

👧 "I want the car to be red."

👨‍🔧 "OK! I take note that you want your car red..."

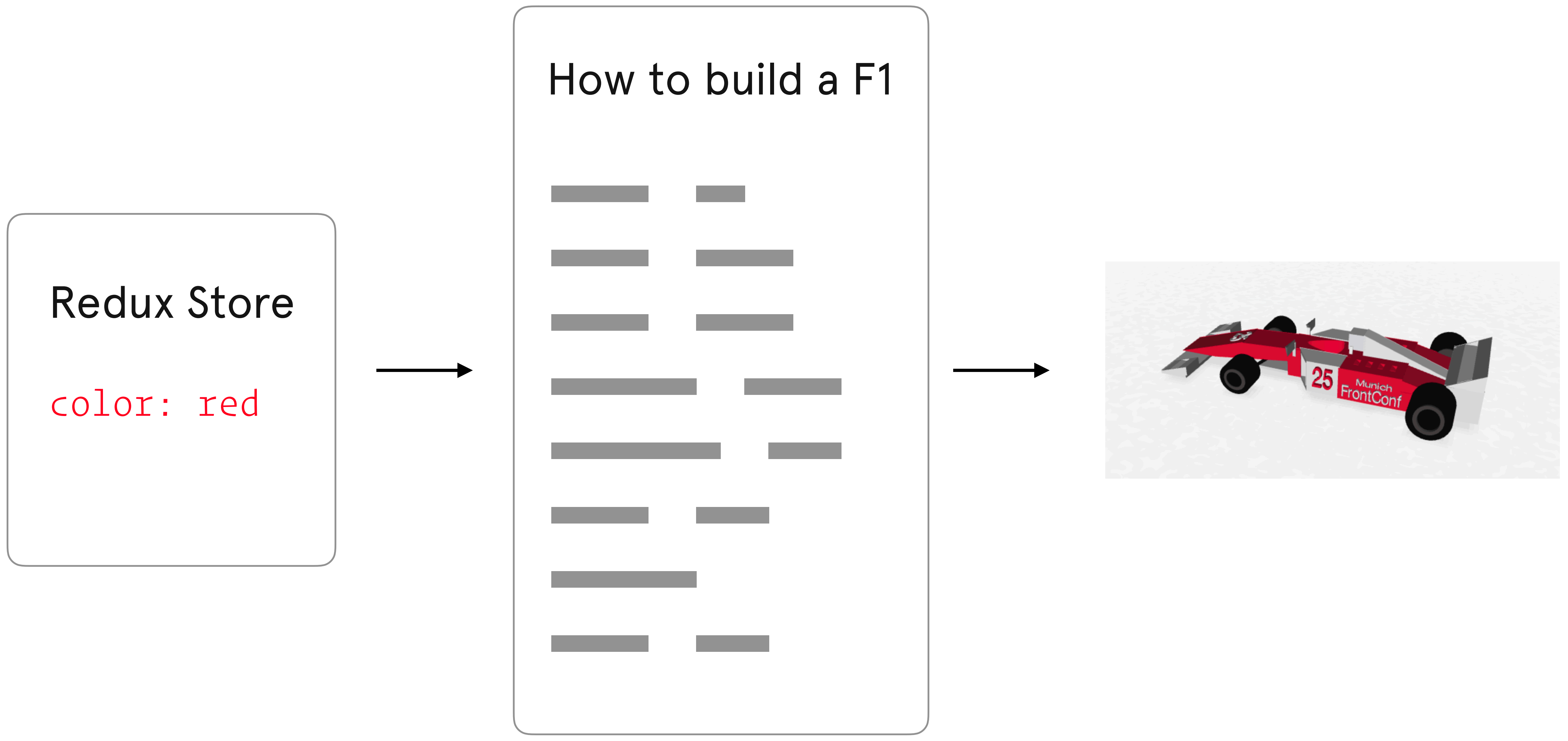
🤖 "Dispatch the `CHANGE_CAR_COLOR` action with the payload `color: red`"

👨‍🔧 "...I sift through all the messy Legos to find red bricks..."

🤖 "A reducer will process the action, and will add `color: red` to the Redux store"

👨‍🔧 "...and I follow the instructions again with the red bricks"

🤖 "The store passes the property `color: red` to React components"


Redux Store
color: red
How to build a F1
25
Munich
FrontConf

Redux — Now with actual code

🤖 “Dispatch the CHANGE_CAR_COLOR action with the payload color”

```
export function changeCarColor(color) {
  return {
    type: 'CHANGE_CAR_COLOR',
    color
  }
}
```

Redux — Now with actual code

🤖 "A reducer will process the action, and will add `color` to the Redux store"

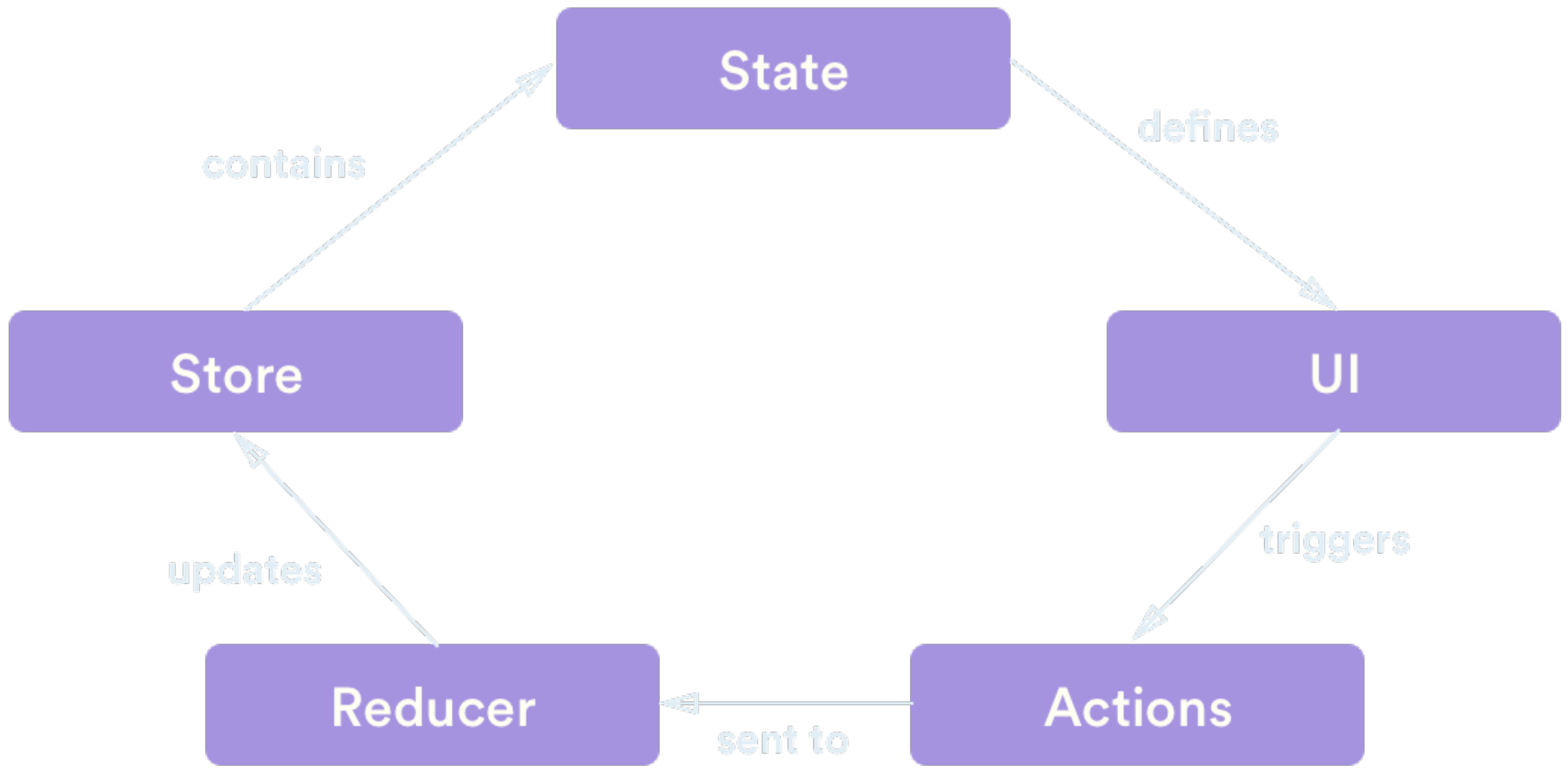
```
function formulaOneApp(state = {}, action) {
  switch (action.type) {
    case 'CHANGE_CAR_COLOR':
      return Object.assign({}, state, {
        color: action.color
      })
    default:
      return state
  }
}
```

Redux — Now with actual code

🤖 “The store passes the properties color and number to React components”

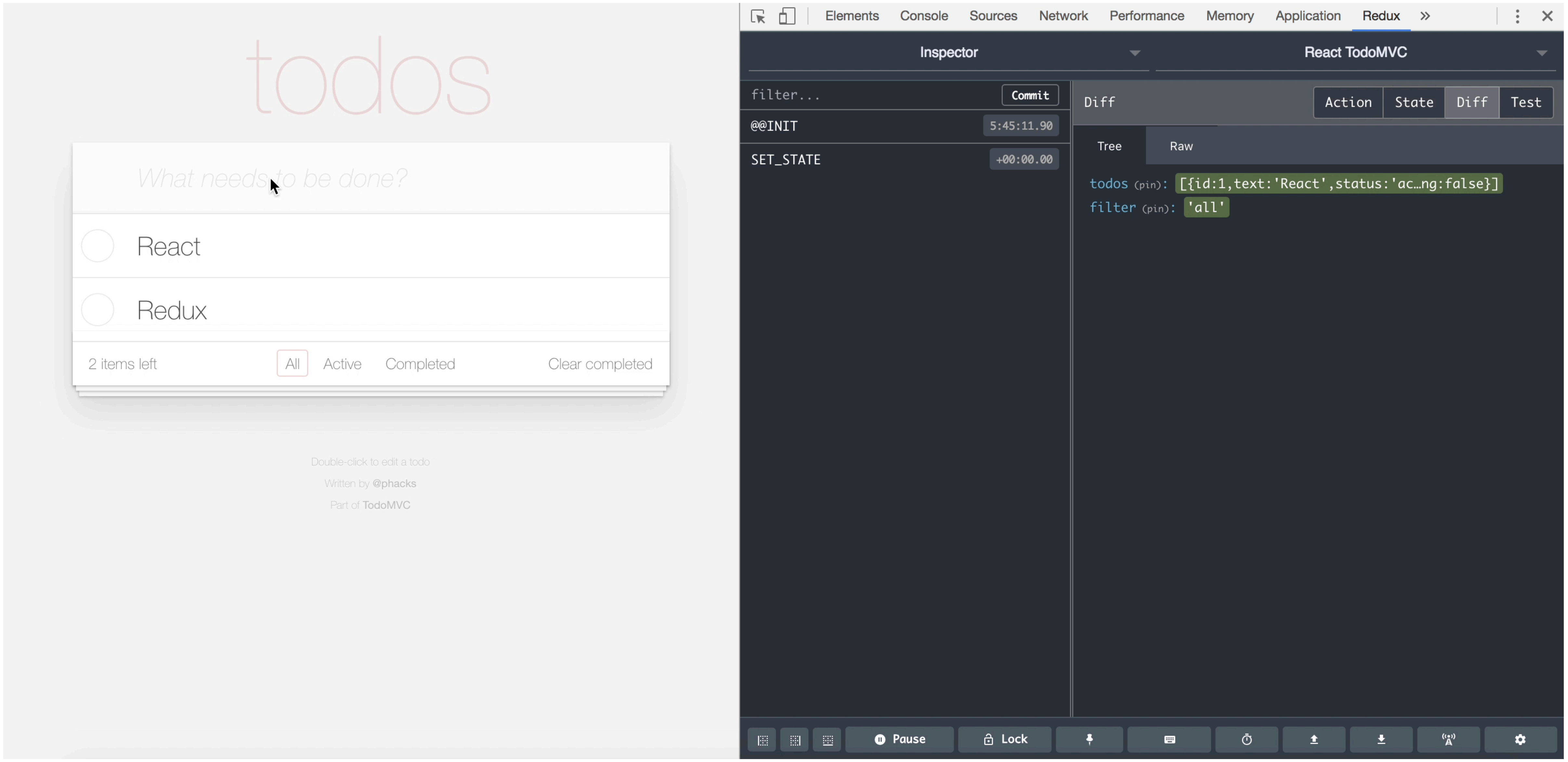
```
import React from 'react'

const FormulaOne = ({ color, number }) => (
  <div>
    <CarBody color={color} />
    <Decorations number={number} />
  </div>
)
```

Redux — The Redux Loop



Developer Experience — Easier Debugging

Developer Experience — Easier Debugging

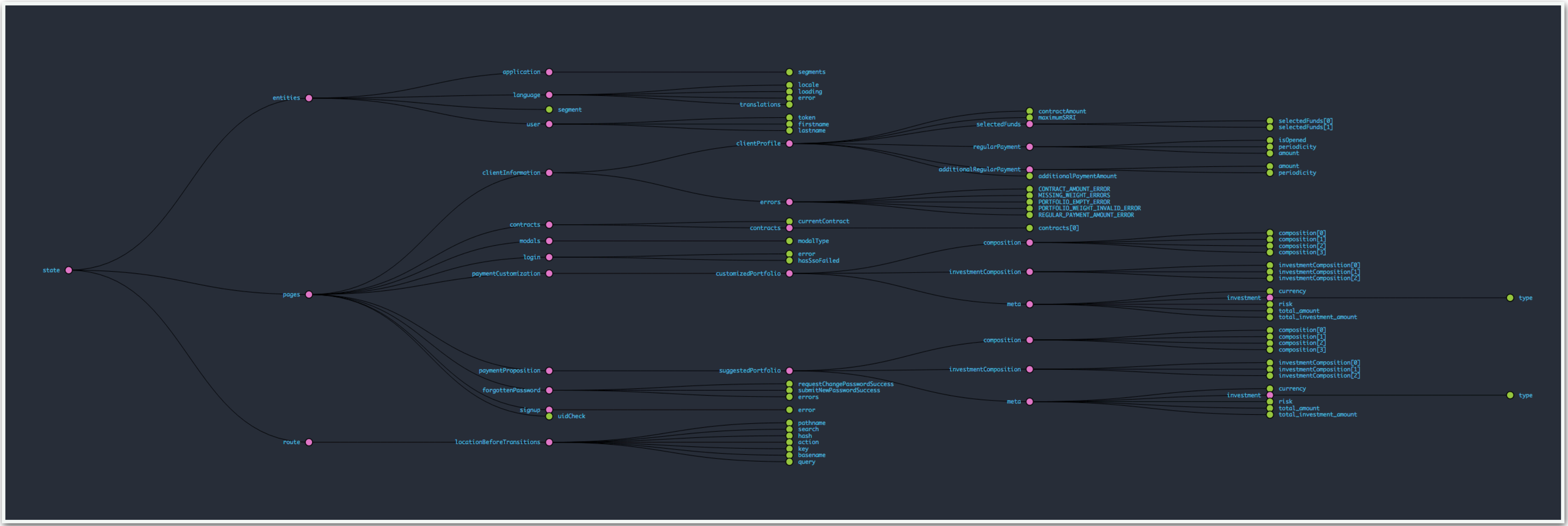


```
@@INIT        5:45:11.90
SET_STATE     +00:00.00
```

```
todos (pin): [{id:1,text:'React',status:'ac…ng:false}]
filter (pin): 'all'
```

Developer Experience — Easier Debugging

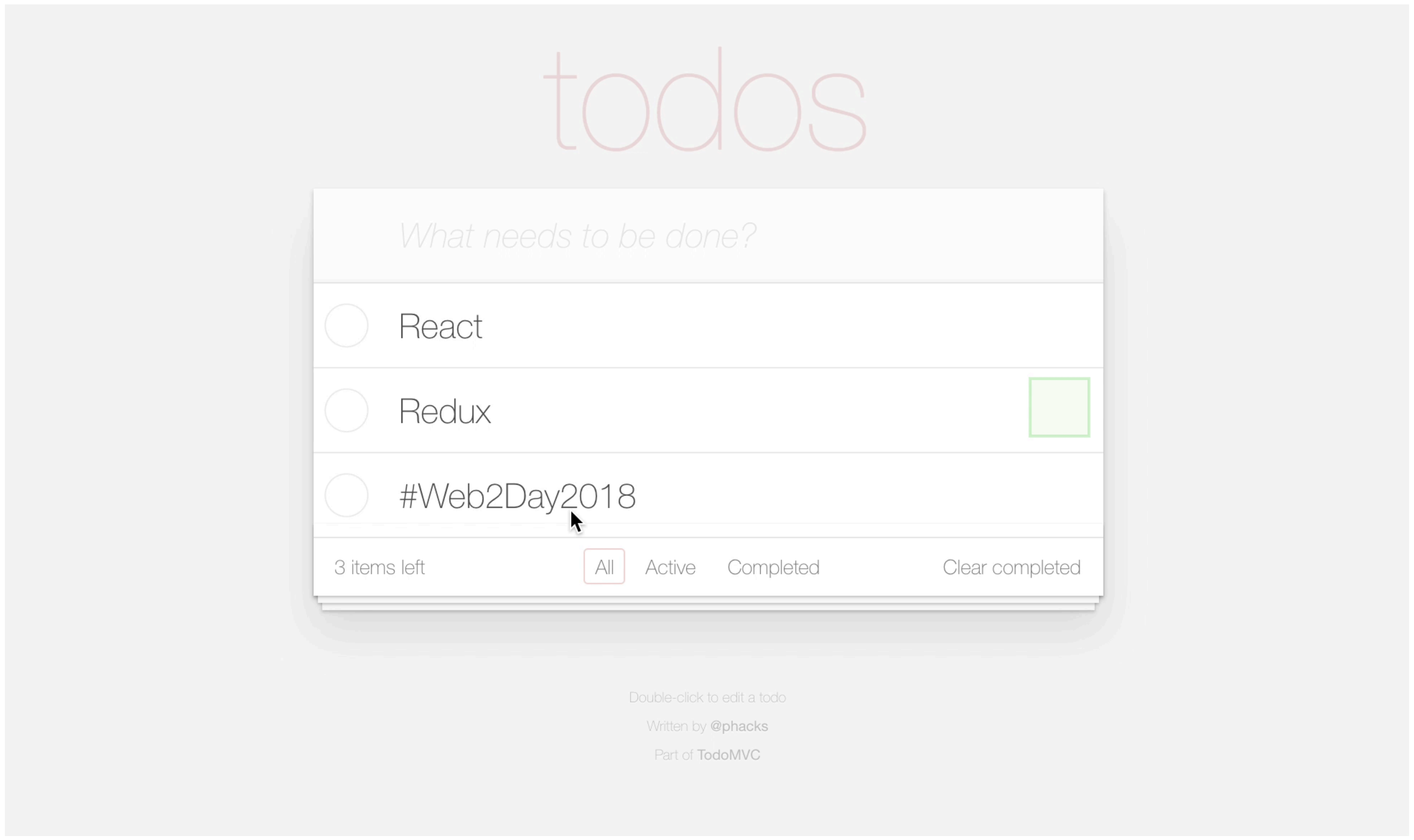
This is what the Redux Store of my current project look like. I can inspect every variable, which are updated in real time.



User Experience — Built-in Performance

User Experience — Built-in Performance

React components are only repainted when their props or inner state change.



React API methods like `shouldComponentUpdate` allow us to have a finer-grained control about render performance.

Developer Experience — “Reasonaboutability”

Developer Experience — “Reasonaboutability”

React developer Jani Eväkallio coined the term “reasonaboutability” (*easiness to reason about*). I love it, and it matches perfectly what I feel about Redux. Here are the Three Principles of Redux:

Developer Experience — “Reasonaboutability”

React developer Jani Eväkallio coined the term “reasonaboutability” (*easiness to reason about*). I love it, and it matches perfectly what I feel about Redux. Here are the Three Principles of Redux:

Single Source of Truth: all the data/UI state displayed on the app come from the same JS object. Facilitates debugging.

Developer Experience — “Reasonaboutability”

React developer Jani Eväkallio coined the term “reasonaboutability” (*easiness to reason about*). I love it, and it matches perfectly what I feel about Redux. Here are the Three Principles of Redux:

Single Source of Truth: all the data/UI state displayed on the app come from the same JS object. Facilitates debugging.

State is read-only: The only way to change the state is to emit an action, an object describing what happened. Provides a single, robust & semantic way to deal with interactions and to work as a team.

Developer Experience — “Reasonaboutability”

React developer Jani Eväkallio coined the term “reasonaboutability” (*easiness to reason about*). I love it, and it matches perfectly what I feel about Redux. Here are the Three Principles of Redux:

Single Source of Truth: all the data/UI state displayed on the app come from the same JS object. Facilitates debugging.

State is read-only: The only way to change the state is to emit an action, an object describing what happened. Provides a single, robust & semantic way to deal with interactions and to work as a team.

Changes are made with pure functions: the Store can only be updated with pure functions (reducers). Prevents nasty side effects and facilitates testing.

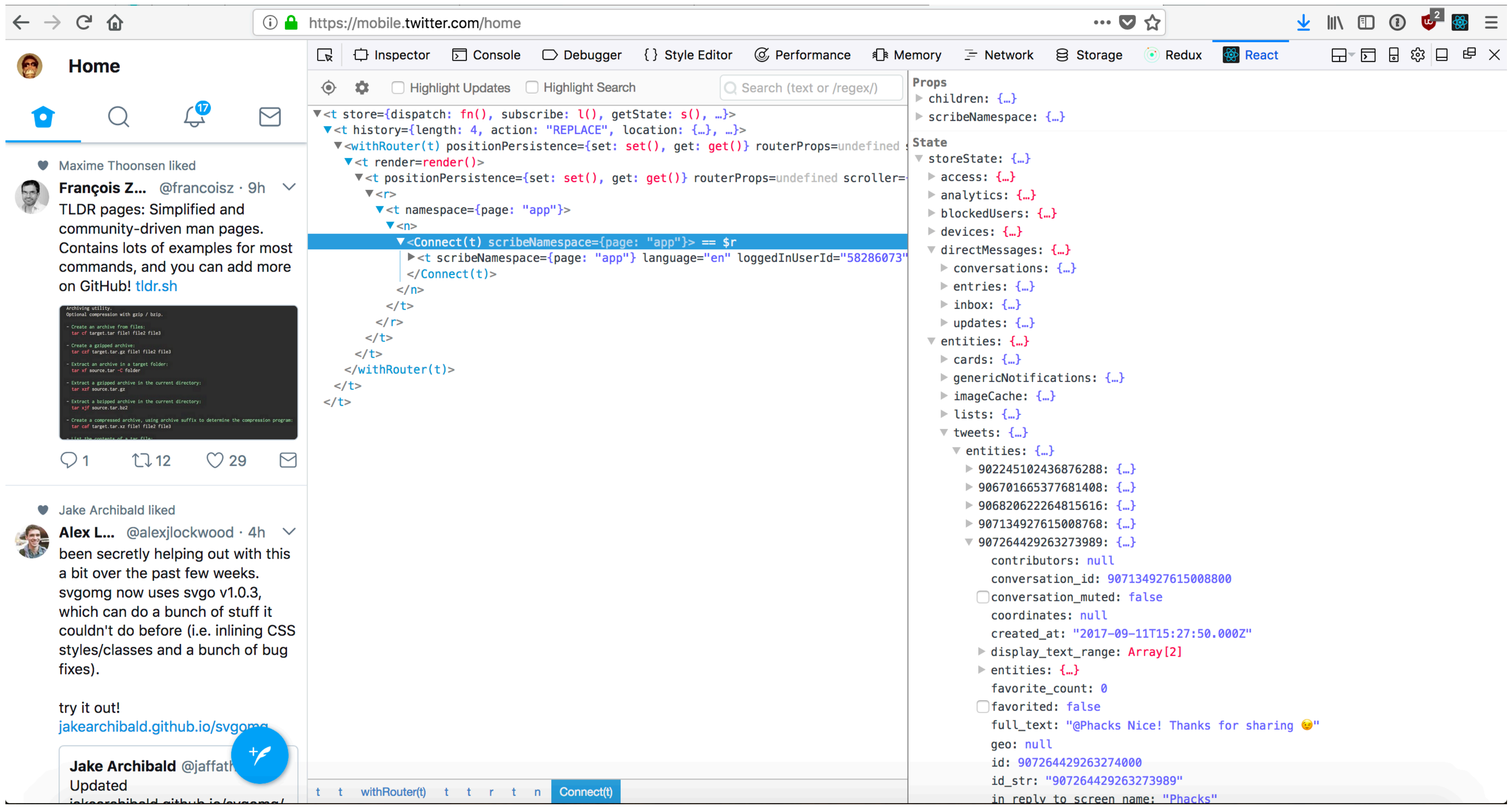
Redux comes with other benefits — and tradeoffs

🛠 **Rich ecosystem**: Redux has an API to plug-in middlewares. There are tons of them: for logging, offline capabilities, async, forms, optimistic UIs...

Tip: 🔗 github.com/markerikson/react-redux-links is a great place to start!

🤓 **Code structure is key**: Since all UI is derived from a single JS object, it needs to be *carefully designed* and *constantly adjusted* to business requirements.

Tip: Learn from the best! Twitter & Pinterest both use Redux, and the structure is available for anybody to see with the React Dev Tools!



Maxime Thoonsen liked
François Z... @francoisz · 9h
TLDR pages: Simplified and community-driven man pages. Contains lots of examples for most commands, and you can add more on GitHub! tldr.sh
1 12 29

Jake Archibald liked
Alex L... @alexjlockwood · 4h
been secretly helping out with this a bit over the past few weeks. svgomg now uses svgo v1.0.3, which can do a bunch of stuff it couldn't do before (i.e. inlining CSS styles/classes and a bunch of bug fixes).

try it out!
jakearchibald.github.io/svgomg

Jake Archibald @jaffath
Updated

```
▼<t store={dispatch: fn(), subscribe: l(), getState: s(), …}>
 ▼<t history={length: 4, action: "REPLACE", location: {…}, …}>
  ▼<withRouter(t) positionPersistence={set: set(), get: get()} routerProps=undefined
   ▼<t render=render()>
    ▼<t positionPersistence={set: set(), get: get()} routerProps=undefined scroller=
     ▼<r>
      ▼<t namespace={page: "app"}>
       ▼<n>
        ▼<Connect(t) scribeNamespace={page: "app"}> == $r
         ▶<t scribeNamespace={page: "app"} language="en" loggedInUserId="58286073"
         </Connect(t)>
        </n>
       </t>
      </r>
     </t>
    </t>
   </withRouter(t)>
  </t>
 </t>
```

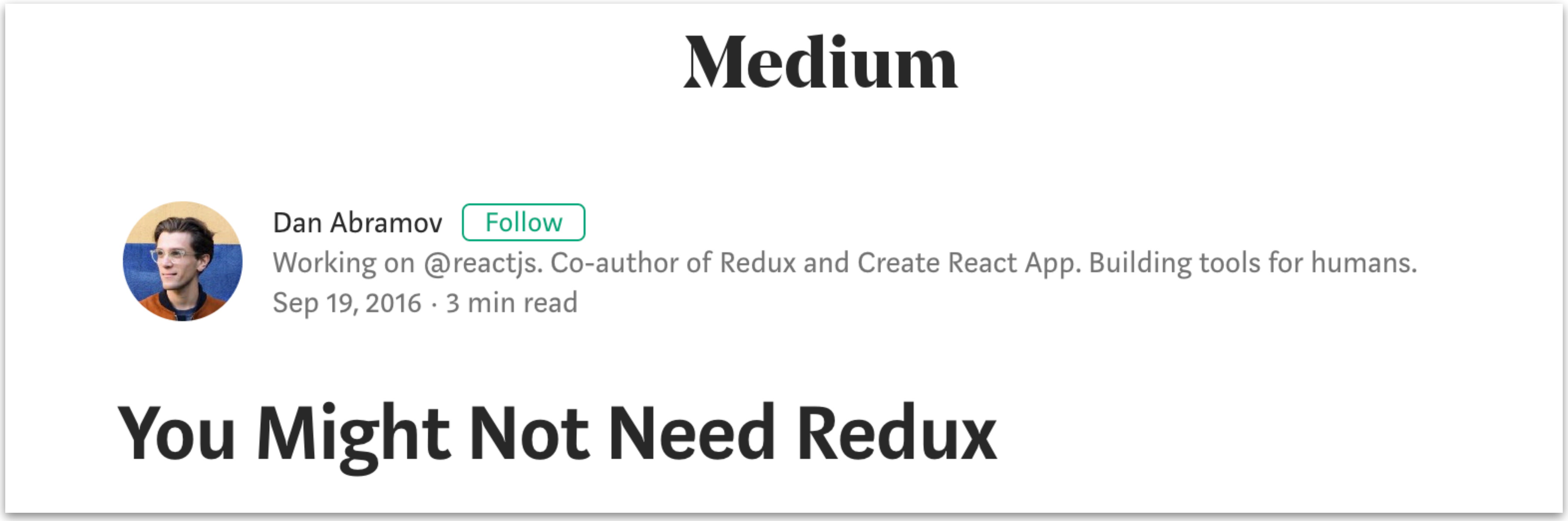
```
t  t  withRouter(t)  t  t  r  t  n  Connect(t)
```

```
Props
▶ children: {…}
▶ scribeNamespace: {…}

State
▼ storeState: {…}
  ▶ access: {…}
  ▶ analytics: {…}
  ▶ blockedUsers: {…}
  ▶ devices: {…}
  ▼ directMessages: {…}
    ▶ conversations: {…}
    ▶ entries: {…}
    ▶ inbox: {…}
    ▶ updates: {…}
  ▼ entities: {…}
    ▶ cards: {…}
    ▶ genericNotifications: {…}
    ▶ imageCache: {…}
    ▶ lists: {…}
    ▼ tweets: {…}
      ▼ entities: {…}
        ▶ 902245102436876288: {…}
        ▶ 906701665377681408: {…}
        ▶ 906820622264815616: {…}
        ▶ 907134927615008768: {…}
        ▼ 907264429263273989: {…}
            contributors: null
            conversation_id: 907134927615008800
          ☐ conversation_muted: false
            coordinates: null
            created_at: "2017-09-11T15:27:50.000Z"
          ▶ display_text_range: Array[2]
          ▶ entities: {…}
            favorite_count: 0
          ☐ favorited: false
            full_text: "@Phacks Nice! Thanks for sharing 😉"
            geo: null
            id: 907264429263274000
            id_str: "907264429263273989"
            in_reply_to_screen_name: "Phacks"
```

😓 **Verbosity**: to write a feature, you would usually need to write an action, a reducer, a *selector*, a *saga*… It can feel quite cumbersome compared to Angular 1.

Tip: I just got used to it. After a while it even feels *more productive* than Angular 1, because you know *exactly* what to do to get everything to work together.

“So, should I use Redux?”



Medium

Dan Abramov Follow

Working on @reactjs. Co-author of Redux and Create React App. Building tools for humans.

Sep 19, 2016 · 3 min read

You Might Not Need Redux

The key points

“If you’re just learning React, don’t make Redux your first choice”

For personal side projects, very small teams (1-2 people) or MVPs with very short time to market, drop Redux & go React

For long-running projects, or larger teams, Redux will help you work better together and lead to a more maintainable code base

Conclusion

React with Redux is a mature framework (React just turned five! 🎉) that empowers developers to produce performant apps with facilitated debugging and a standard yet expressive development flow.

Want to dive in?

I wrote a full-featured, test-driven tutorial on writing a Todo List using React & Redux

🔗 https://github.com/phacks/redux-todomvc

Merci!

Slides are available at phacks.github.io

& /phacks