Focal

Functional-reactive state management

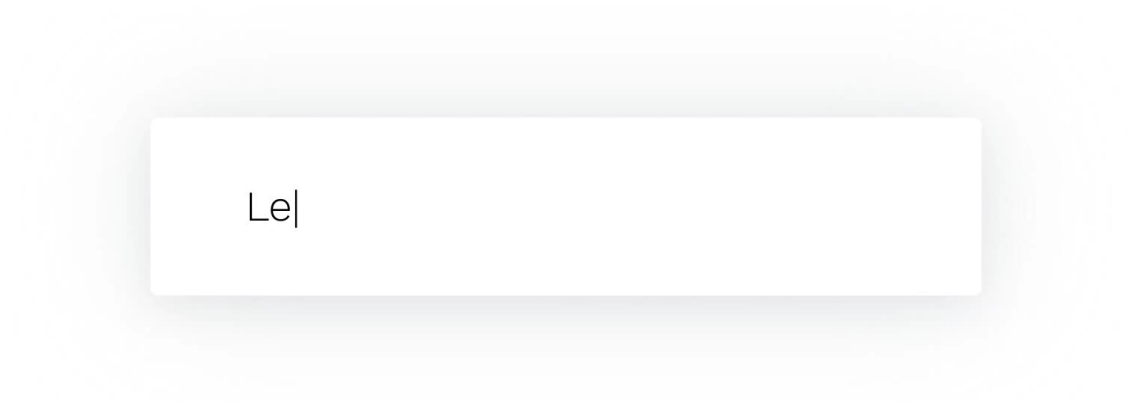
Sergey Yavnyi / @glatteis

Nov, 2018

Hi, my name is Sergey 👋

- User-facing products @ Grammarly
- Functional programming + static typing = ❤️

github.com/blacktaxi



- A writing app *(grammar, spelling and more!)*
- Web *(+ extensions!)*, desktop, mobile
- 10M+ active users

(since 1.8!)

THIS TALK: FOCAL

- State management for React
- Immutable state store
- Easy to use
- Type safe

github.com/grammarly/focal

```
npm i -S @grammarly/focal
```

REDUX?

REDUX

- Predictable, consistent state
- Dev tools

- Conciseness (actions, reducers, middleware, async)
- Modularity (`combineReducers`)
- Type safety (you'll need a guide)
- Performance (`shouldComponentUpdate`?)

PREDICTABLE, CONSISTENT STATE

Worth it.

...but can we do better?

CONCISENESS

`Atom` – the "store"

```
// create the store
const state = Atom.create({ count: 0 })

// update state
state.modify(s => ({ ...s, count: s.count + 1 }))
```

`modify`: *functional update (similar to a reducer)*

CONCISENESS

```
type AppState = number
```

Redux

```
type AppAction = { type: 'INC' }

const increment = () => ({ type: 'INC' })

const reducer = (s: AppState, a: AppAction) => {
  switch (a.type) {
    case 'INC': return s + 1

    // ...
  }
}

//...
<button onClick={() => props.dispatch(increment())} />
```

Focal

```
//...
<button onClick={() => props.state.modify(s => s + 1)} />
```

MODULARITY

```
type AppState = { count: number }
```

MODULARITY

Redux

```
type AppAction = { type: 'INC' }

const increment = () => ({ type: 'INC' })

const reducer = (s: AppState, a: AppAction) => {
  switch (a.type) {
    case 'INC': return { ...s, count: s.count + 1 }

    // ...
  }
}

const mapStateToProps = (state: AppState) => ({
  count: state.count
})

// to be continued...
```

...Redux

```
// ...continued
const Counter = connect(mapStateToProps)(
  (props: {
    count: number; dispatch: (a: AppAction) => void
  }) =>
    <p>
      Value: {props.count}
      <button onClick={() => props.dispatch(increment())} />
    </p>
)

const App = () =>
  <div>
    <Counter />
  </div>

// ...flip over to the next slide 👉
```

...Redux

```
// ...continued

render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
)

// DONE! 💪
```

MODULARITY

Focal

```
const Counter = (props: { count: Atom<number> }) =>
  <F.p>
    Value: {props.count}
    <button onClick={() => props.count.modify(x => x + 1)}/>
  </F.p>

const App = (props: { state: Atom<AppState> }) =>
  <div>
    <Counter count={state.lens('count')} />
  </div>

render(
  <App state={state} />,
  document.getElementById('root')
)

// That's it! 🙌
```

`Atom.lens(...)?`

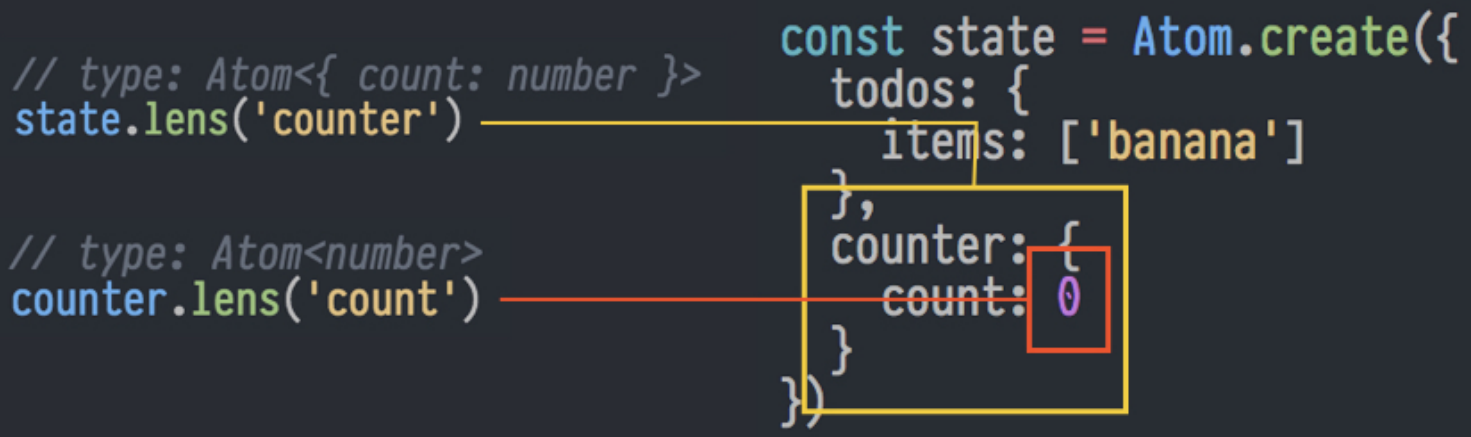
```
// type: Atom<{
//   todos: { items: string[] };
//   counter: { count: number };
// }>
const state = Atom.create({
  todos: {
    items: ['banana']
  },
  counter: {
    count: 0
  }
})
```

Break state into pieces

```
// type: Atom<{ items: string[] }>
const todos = state.lens('todos')

// type: Atom<{ count: number }>
const counter = state.lens('counter')

// type: Atom<number>
const count = counter.lens('count')
```


// type: Atom<{ count: number }>
state.lens('counter')
// type: Atom<number>
counter.lens('count')
const state = Atom.create({
todos: {
items: ['banana']
},
counter: {
count: 0
}
})

LENS?

A way to read/write a *part*[1] of immutable data

```
interface Lens<T, U> {
  get(source: T): U // a getter
  set(newValue: U, source: T): T // immutable update
}
```

[1] – abstractly speaking

```
state.lens('count')
```

...just a short form of

```
state.lens(Lens.create(
  // getter
  (s: AppState) => s.count,
  // setter
  (v, s) => ({ ...s, count: v })
))
```

LENSES ARE FUN 🔍

- Program imperatively using Haskell lenses
- partial.lenses, calmm-js

TYPE SAFETY

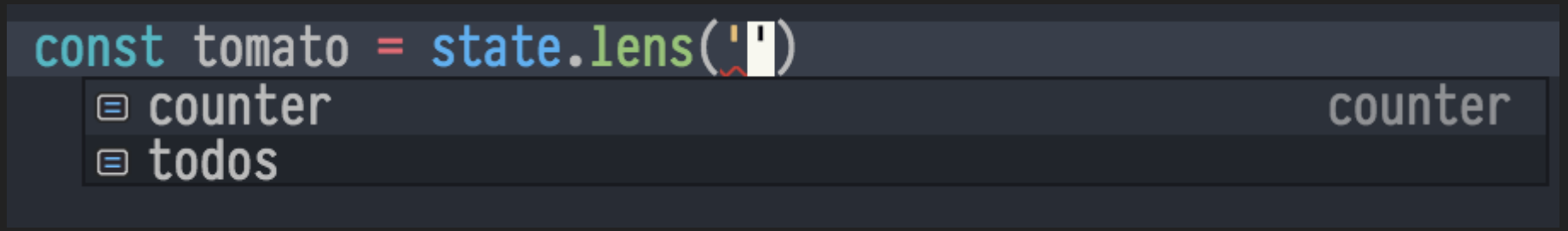
```
const state = Atom.create(
  { counter: { count: 0 }, todos: ['banana'] }
)

const count = state.lens('counter') // ✅ good to go

// [ts] Argument of type '"potato"' is not assignable to
// parameter of type '"counter" | "todos"'.
const potato = state.lens('potato') // ❌ does NOT compile

// [ts] Argument of type '"nope"' is not assignable to
// parameter of type '{ count: number; }'.
counter.set('nope') // ❌ no luck here
```

Auto-completion



REDUX + TS

github.com/piotrwitek/react-redux-typescript-guide

Table of Contents

`<F.p>...?`

`F`-components: put `Atom`s in JSX

Subscribe to `Atom`s on mount

```
<F.p onClick={() => state.modify(x => x + 1)}>
  Count: {state}.
</F.p>
```

`Atom.subscribe(...)`

```
state.subscribe(s => {
  // print the app state as it's changing
  console.log('new app state is:', s)
})
```

SUBSCRIBE?

```
Atom<T> extends Observable<T>
```

`Atom` IS `Observable`

Can use RxJS operators on `Atoms`

```
state
  .debounceTime(500)
  .subscribe(s => {
    window.localStorage.setItem(
      'appState',
      JSON.stringify(s)
    )
  })
```

RXJS + REACT

<F.* /> render Observables

```
<F.div>
  The time is {
    Observable.inverval(1000)
      .startWith(0)
      .map(_ => new Date().toString())
  }.
</F.div>
```

Not just Atoms!

OBSERVABLES

```
// simplified
interface Observable<T> {
  subscribe(
    onNext: (value: T) => void
  ): { unsubscribe(): void }
}
```

- First class event: *event as value*
- A stream of data: *collection of values over time*

OBSERVABLES

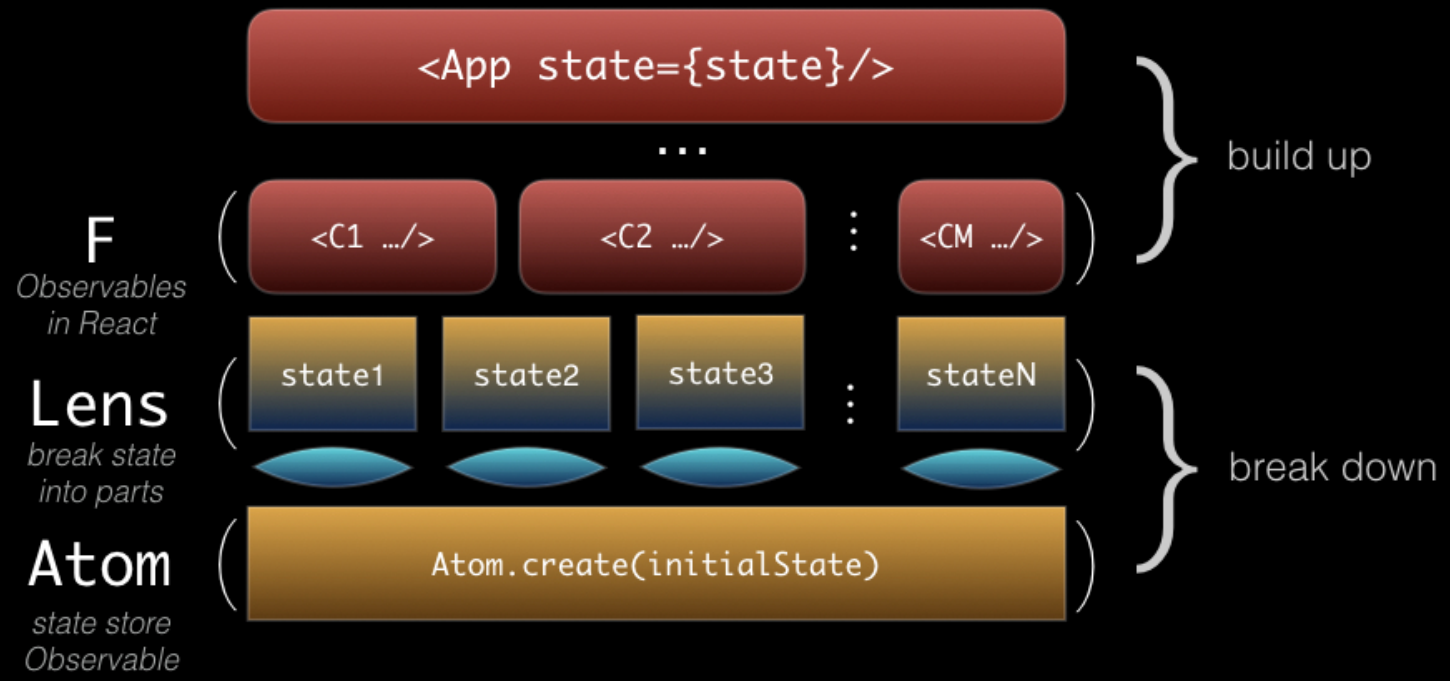
- http://queue.acm.org/detail.cfm?id=2169076
  - Your Mouse is a Database
- https://goo.gl/fQvnzc
  - Learning Observable By Building Observable
- http://rxmarbles.com

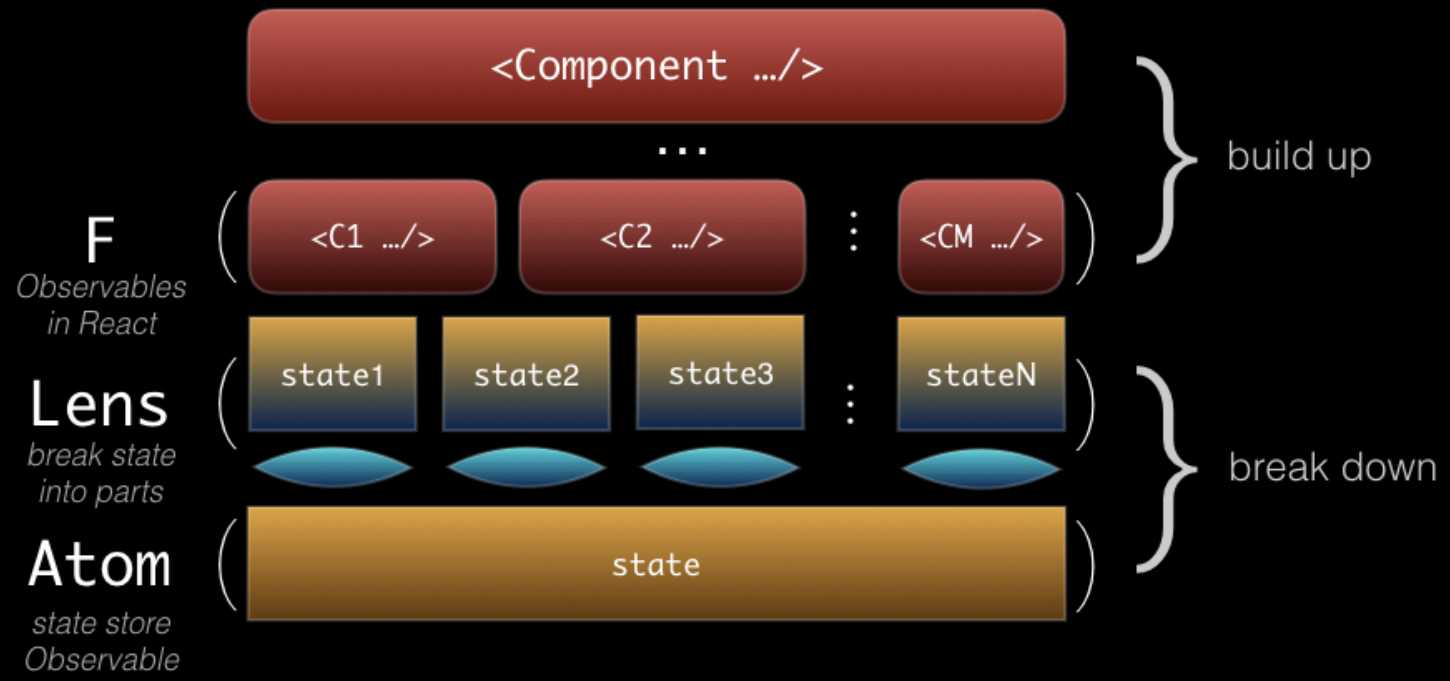
PERFORMANCE

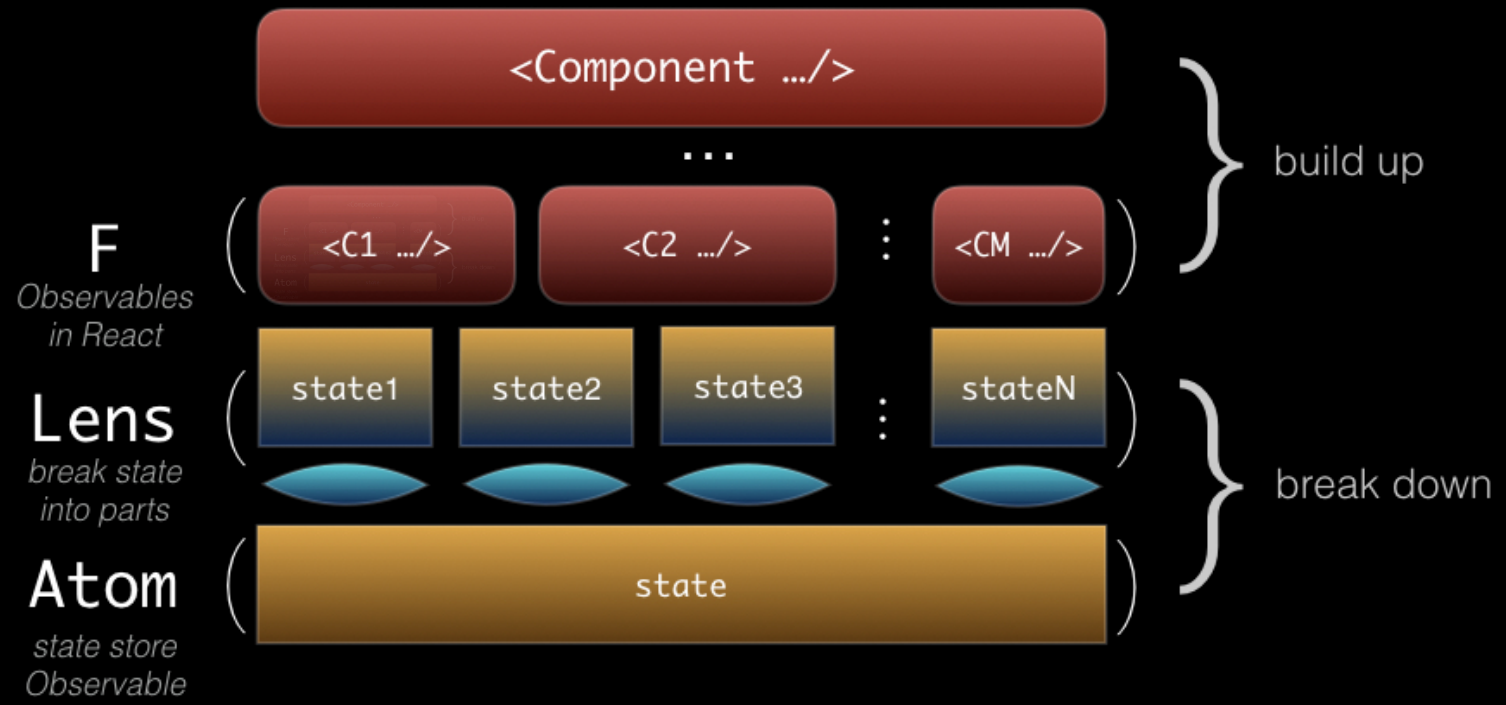
~~shouldComponentUpdate~~

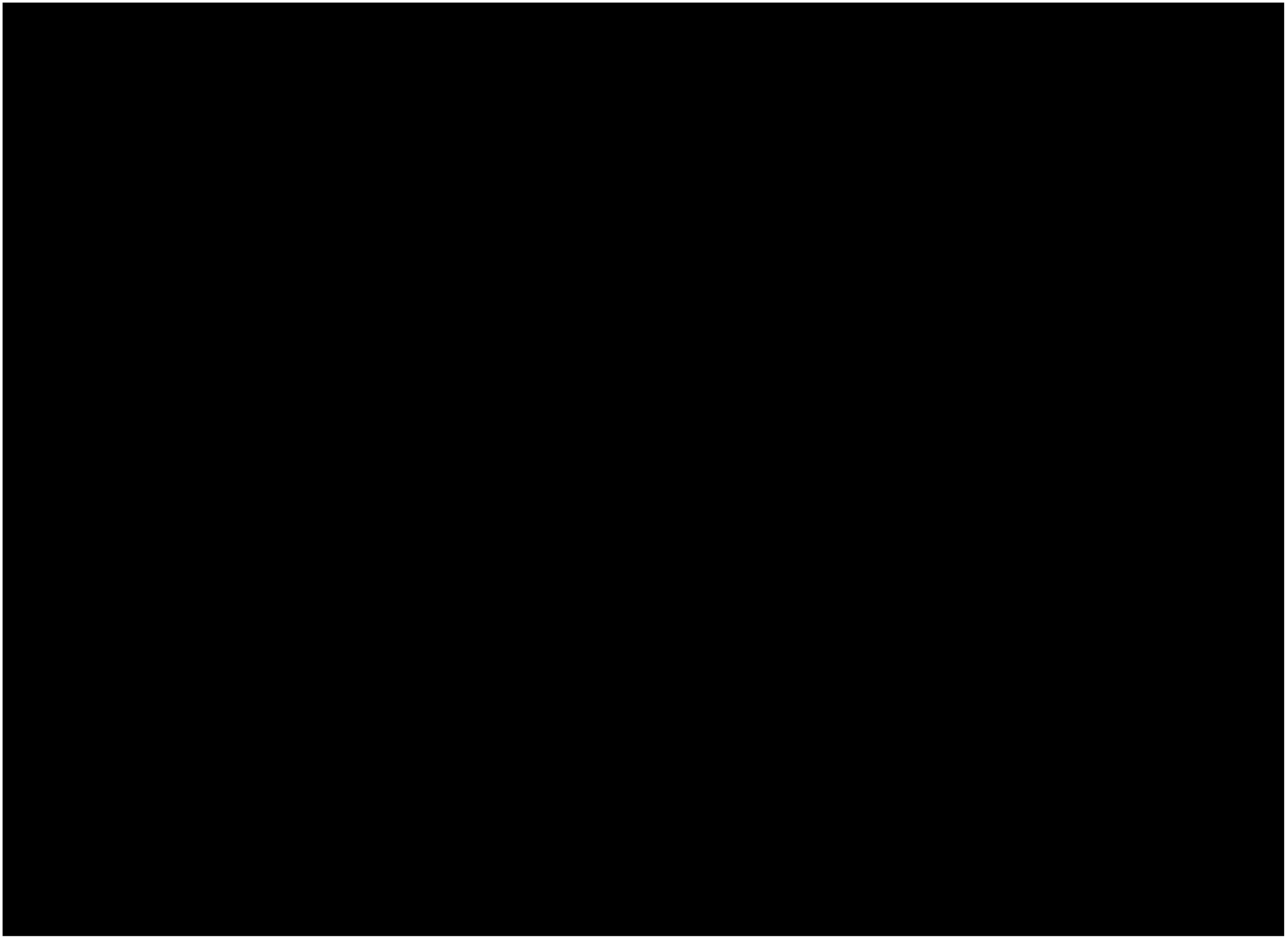
BONUS

Can use time-travel debugging!


<App state={state}/>
...
<C1 …/>
<C2 …/>
<CM …/>
state1
state2
state3
stateN
Atom.create(initialState)
F
Observables
in React
Lens
break state
into parts
Atom
state store
Observable
build up
break down


<Component …/>
<C1 …/>
<C2 …/>
<CM …/>
state1
state2
state3
stateN
state
build up
break down
F
Observables
in React
Lens
break state
into parts
Atom
state store
Observable


<Component …/>
<C1 …/>
<C2 …/>
<CM …/>
state1
state2
state3
stateN
state
F
Observables
in React
Lens
break state
into parts
Atom
state store
Observable
build up
break down



It's hacking time!

source

more examples

Q & A

github.com/grammarly/focal

```
npm i -S @grammarly/focal
```


WE’RE HIRING!
grammarly.com/jobs