How to `reselect`

A simple guide by Grzegorz Rozdzialik

Problem

Extractors

```
const extractUser = (state: State) => state.user;
const extractUserName = (state: State) => extractUser(state).name;

const extractCart = (state: State) => state.cart;
const extractCartItems = (state: State) => extractCart(state).items;
```

Components

UserInfo

```tsx
import { StatelessComponent } from 'react';

interface UserInfoProps {
  userName: string;
}

export const UserInfo: StatelessComponent<UserInfoProps> = ({ userName }) => (
  <div>Hello, {userName}!</div>
);
```

```
const userName = extractUserName(state);
<UserInfo userName={userName} />
```

Hello, Grzenio!

CartInfo

```tsx
import { StatelessComponent } from 'react';

import { Cart } from '../types';

interface CartInfoProps {
  cart: Cart;
}

export const CartInfo: StatelessComponent<CartInfoProps> = ({ cart }) => (
  <ul>
    {...cart.items.map((item) => (
      <li key={item.name}>
        {item.name} ({item.quantity} x ${item.price})
      </li>
    ))}
  </ul>
);
```

```
const cart = extractCart(state);
<CartInfo cart={cart} />
```

- Cup (5 x $1)
- Pen (1 x $20)

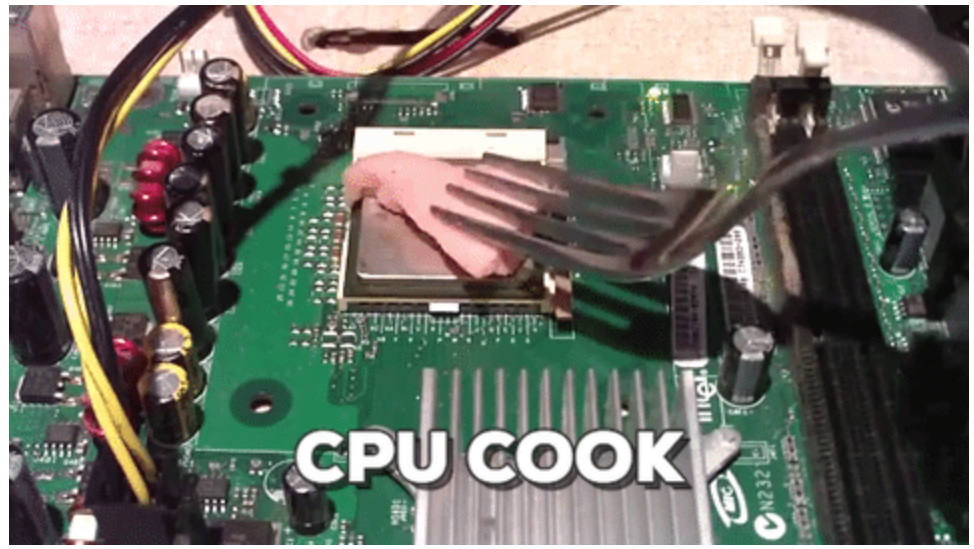
Info = UserInfo + CardInfo

```
import { CartInfoProps, CartInfo } from './cart-info';
import { UserInfoProps, UserInfo } from './user-info';

type InfoProps = CartInfoProps & UserInfoProps;

export const Info: StatelessComponent<InfoProps> = ({ cart, userName }) => (
  <React.Fragment>
    <UserInfo userName={userName} />
    <CartInfo cart={cart} />
  </React.Fragment>
);
```

Performance?



```tsx
export const CartInfo: StatelessComponent<CartInfoProps> = ({ cart }) => {
  // Measure the number of renders
  console.log('Rendering cart info');

  return (
    <ul>
      {...cart.items.map((item) => (
        <li key={item.name}>
          {item.name} ({item.quantity} x ${item.price})
        </li>
      ))}
    </ul>
  );
};
```

```tsx
export const UserInfo: StatelessComponent<UserInfoProps> = ({ userName }) => {
  console.log('Rendering user info');

  return <div>Hello, {userName}!</div>;
};
```

Console was cleared

Rendering user info

Rendering cart info

>


LIVE
OFF TO A GOOD START.

Let's simulate changing the state (in `app.tsx`)

```
componentDidMount() {
  // Simulates changing the state
  setTimeout(() => {
    this.setState({
      user: {
        ...this.state.uesr,
        name: 'Not Grzenio',
      },
    });
  }, 2000);
}
```

```
Console was cleared
Rendering user info
Rendering cart info
Rendering user info
Rendering cart info
>
```

```
Console was cleared
Rendering user info
Rendering cart info
Rendering user info
Rendering cart info
>
```

```
Console was cleared
Rendering user info ✔
Rendering cart info ✔
Rendering user info
Rendering cart info
>
```

```
Console was cleared
Rendering user info
Rendering cart info
Rendering user info
Rendering cart info
>
```

```
Console was cleared
Rendering user info
Rendering cart info
Rendering user info
Rendering cart info
>
```


WHYYYY

Solution 1

`StatelessComponent` -> `PureComponent`

```tsx
export class UserInfo extends PureComponent<UserInfoProps> {
  public render() {
    console.log('Rendering user info');

    return <div>Hello, {this.props.userName}!</div>;
  }
}
```

Result

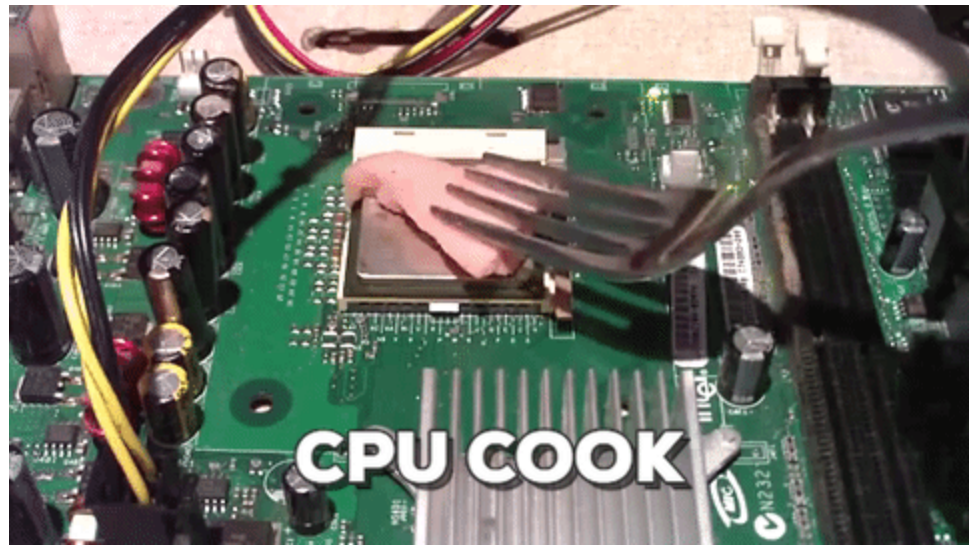
```
Console was cleared
Rendering user info
Rendering cart info
Rendering user info
>
```

But still...

```
const extractCart = (state: State) => state.cart;
const extractCartItems = (state: State) => extractCart(state).items;
```

`cart` did not change

`extractCartItems` is executed again anyway


CPU COOK

Can we do better?

Solution 2

reselect

Building block - selector

```ts
export type Selector<S, R> = (state: S) => R;
```

Basic selector

```ts
const stateSelector: Selector<State, State> = (state) => state;
```

Composing selectors

```
const userSelector = createSelector(
  stateSelector,
  (state) => state.user,
);

// userSelector: (state: State) => User
```

```
const userNameSelector = createSelector(
  userSelector,
  (user) => user.name,
);

// userNameSelector: (state: State) => string
```

Similar selectors for `cart`

```
const cartSelector = createSelector(
  stateSelector,
  (state) => state.cart,
);

const cartItemsSelector = createSelector(
  cartSelector,
  (cart) => cart.items,
);
```

Usage

```
// app.tsx

const userName = userNameSelector(state);
const cart = cartSelector(state);

return <Info userName={userName} cart={cart} />;
```

Benefits

1. Easy selector composition
2. Selectors are not recomputed when it is not needed

```
const userSelector = createSelector(
  stateSelector,
  (state) => state.user,
);

const userNameSelector = createSelector(
  userSelector,
  (user) => user.name,  // <- this function will not run if `user` did not change
);
```

Combining multiple selectors

```
const totallyUnnecessarySelectors = createSelector(
  userNameSelector,
  cartSelector,
  (userName, cart) => `${userName}'s store updated at ${cart.updatedAt}`,
);
```

Parametrizing selectors

```
const cartItemSelector = (itemName: string) =>
  createSelector(cartItemsSelector, (cartItems) =>
    cartItems.find((item) => item.name === itemName),
  );

// usage:
cartItemSelector('Pen')(state)
```

Parametrized selectors in `mapStateToProps`

```
interface SomeComponentsProps {
  item: CartItem;
}

const mapStateToProps: MapStateToProps<SomeComponentsProps, null, State> = (
  state,
) => {
  return {
    item: cartItemSelector('Pen')(state),
  };
};

export default connect(mapStateToProps)(SomeComponent);
```


IT'S BAD

But why?

```
const mapStateToProps: MapStateToProps<SomeComponentsProps, null, State> = (
  state,
) => {
  const selector = cartItemSelector('Pen'); // creates new selectors each time

  return {
    item: selector(state),
  };
};

export default connect(mapStateToProps)(SomeComponent);
```

The proper way

```
const mapStateToPropsFactory: MapStateToPropsFactory<
  SomeComponentsProps,
  null,
  State
> = () => {
  // selector is created only once
  const selector = cartItemSelector('Pen');

  return (state) => ({
    item: selector(state),
  });
};

export default connect(mapStateToPropsFactory)(SomeComponent);
```


(CHEERING)

Structured selectors

```
const mapStateToProps = createStructuredSelector({
  item: cartItemSelector('Pen'),
});

export default connect(mapStateToProps)(SomeComponent);
```

Conclusion

`reselect` provides

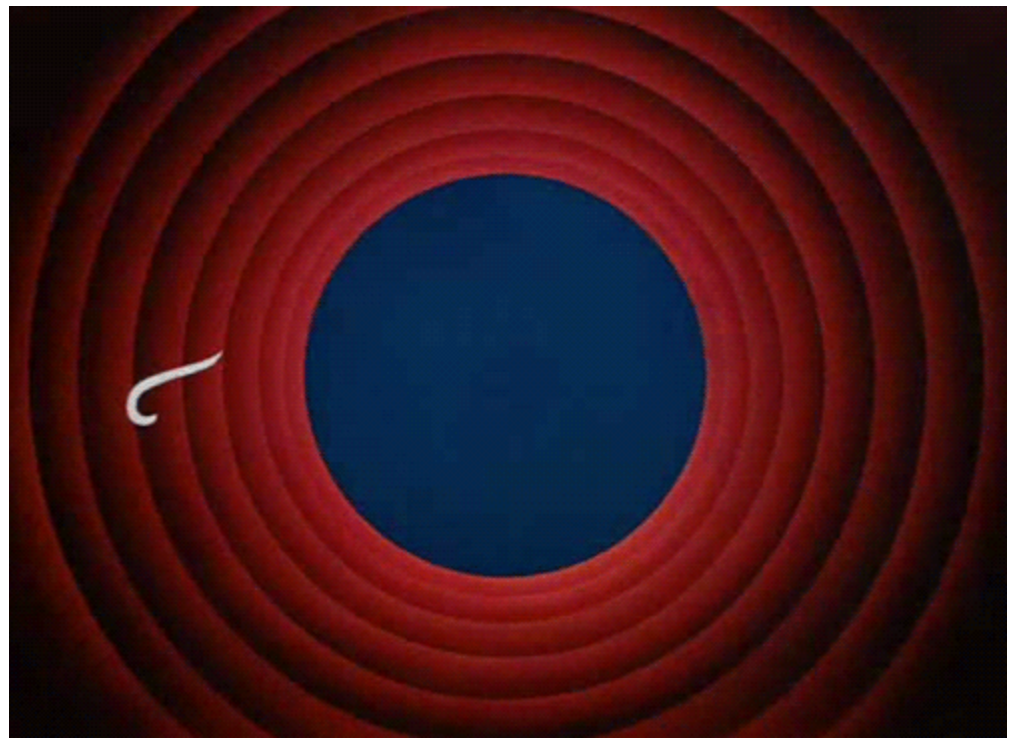
More info

https://github.com/reduxjs/reselect

Commonly used selectors

`app/fabric/resources/entities/selectors.ts`

(and many other files that use selectors)



Questions?