2.11 The views: smart and dumb components

As mentioned in **5. Divide The App Into in Containers and Components**, we will create **Presentational components** (dumb components) for presentation purposes and a **Container components** (smart components) which are wrapper component responsible for Actions while communicating with Redux.

Smart components are **responsible for the actions**. If a dumb component underneath them needs to trigger an action, the smart component will pass a function through props, and the dumb component can then treat that as a **callback**.

We already have dumb components for presentation purposes in part 1, and will reuse them.

Here we create container components as upper **wrapper** around each dumb components.

2.11.1 The view layer binding

Redux needs some help to connect the store to the view. It needs something to bind the two together. This is called the **view layer binding**. In an app that uses react, this is `react-redux`.

Technically, a container component is just a React component that uses **store.subscribe()** to read a part of the Redux state tree and supply **props** to a presentational component it renders.

Hence, we can manually create container components, but this is not recommended for Redux official docs. This is because **react-redux performs many performance optimizations** that are difficult to perform manually.

For this reason, instead of writing the container component by hand, we write it using the `connect()` function provided by react-redux.

Let’s install the necessary packages first.

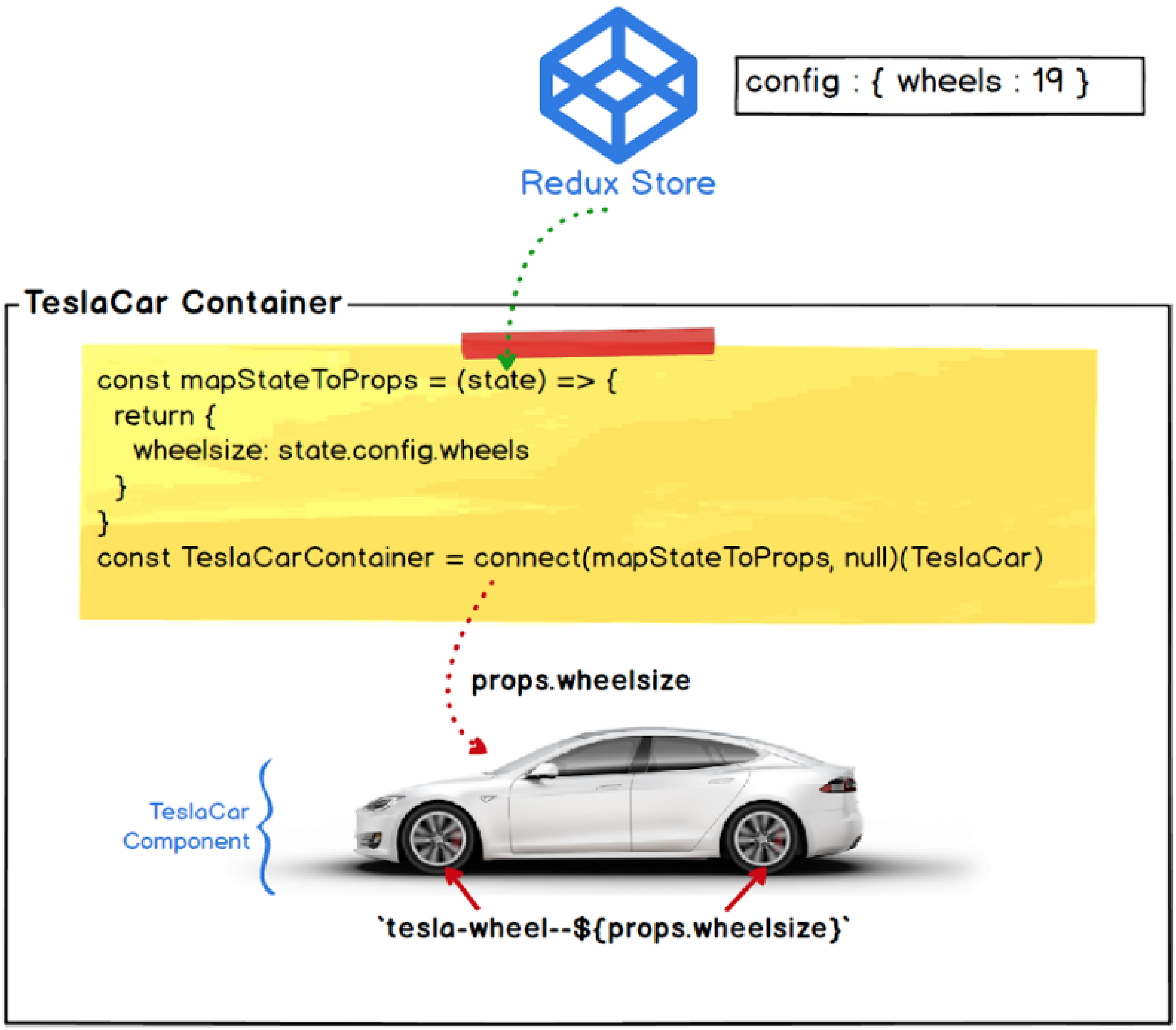
- **npm install --save redux**

- **npm install --save react-redux**

2.11.2 TeslarCar Container

To use **connect()**, you need to define a special function called `mapStateToProps`. This function tells you **how to convert the current Redux store state to props** to be passed to the presentation component.

The TeslarCar container takes the wheelsize stored in the current store and passes it to props so that it can be rendered by the TeslarCar component. This props will be updated every time the state is updated.



```
const mapStateToProps = (state) => {
  return {
    wheelsize: state.config.wheels
  }
}
const TeslaCarContainer = connect(mapStateToProps, null)(TeslaCar)
```

After defining the mapStateToProps function, we defined the connect()function as shown below.

```
const TeslaCarContainer = connect(mapStateToProps, null)(TeslaCar)
```

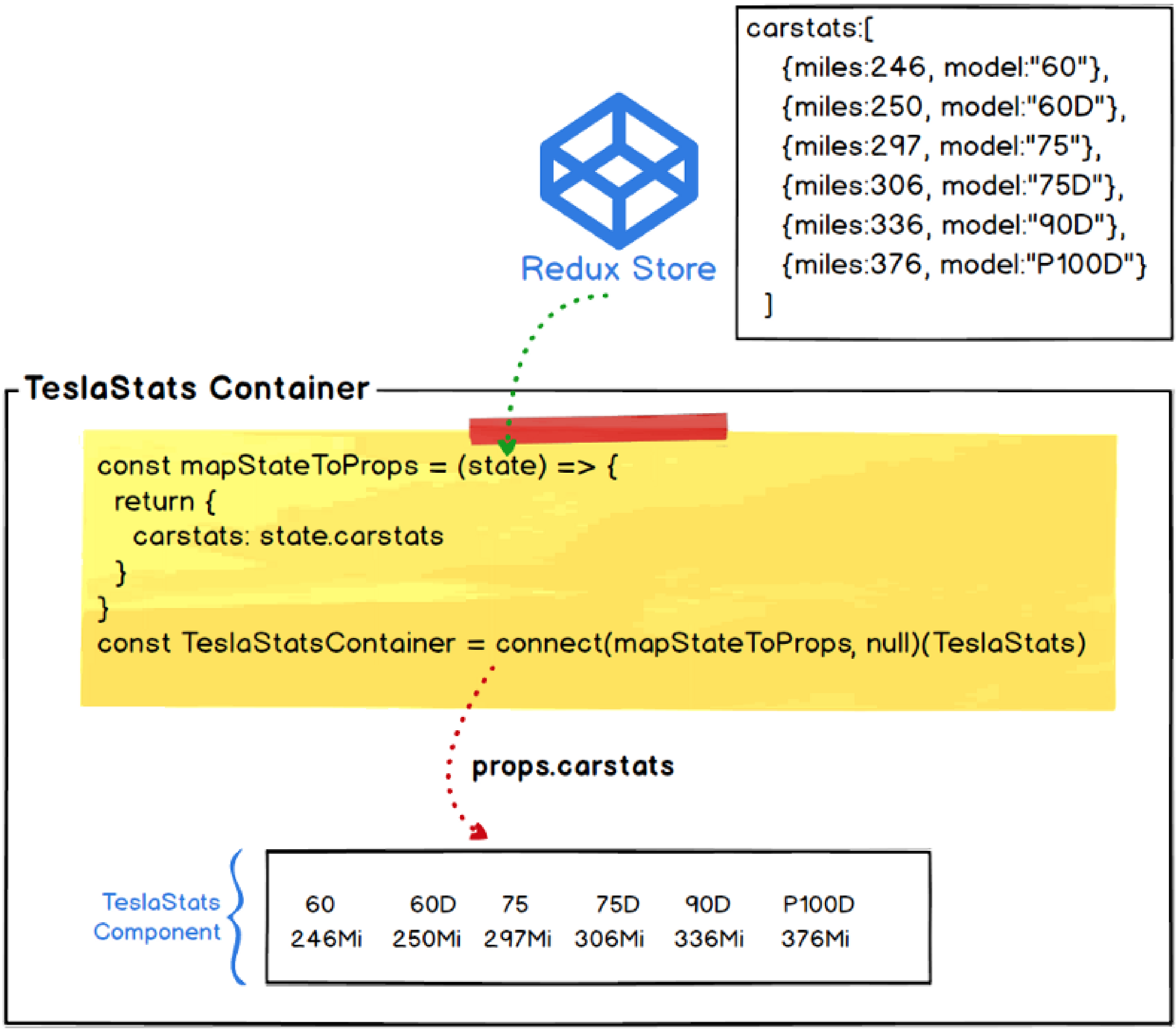
connect() accepts `mapDispatchToProps` as the second argument, which takes the dispatch method of the store as its first argument. In the TeslaCar component, we do not need an action, so we have to pass null.

> *Another parenthesis in **connect()()** may look weird. This form actually means two function calls, the **first connect() returns another function**, and the **second function needs you to pass a React component**. In this case it's our TeslaCar component. This pattern is called **currying** or **partial application** and is a form of functional programming.*

Create **src/containers/TeslaCarContainer.js** and write the code.

2.11.3 TeslaStats Container

As with the TeslaCar container, define only the **mapStatToProps** function and pass it to connect() in TeslaStats container.



```
carstats:[
   {miles:246, model:"60"},
   {miles:250, model:"60D"},
   {miles:297, model:"75"},
   {miles:306, model:"75D"},
   {miles:336, model:"90D"},
   {miles:376, model:"P100D"}
 ]
```

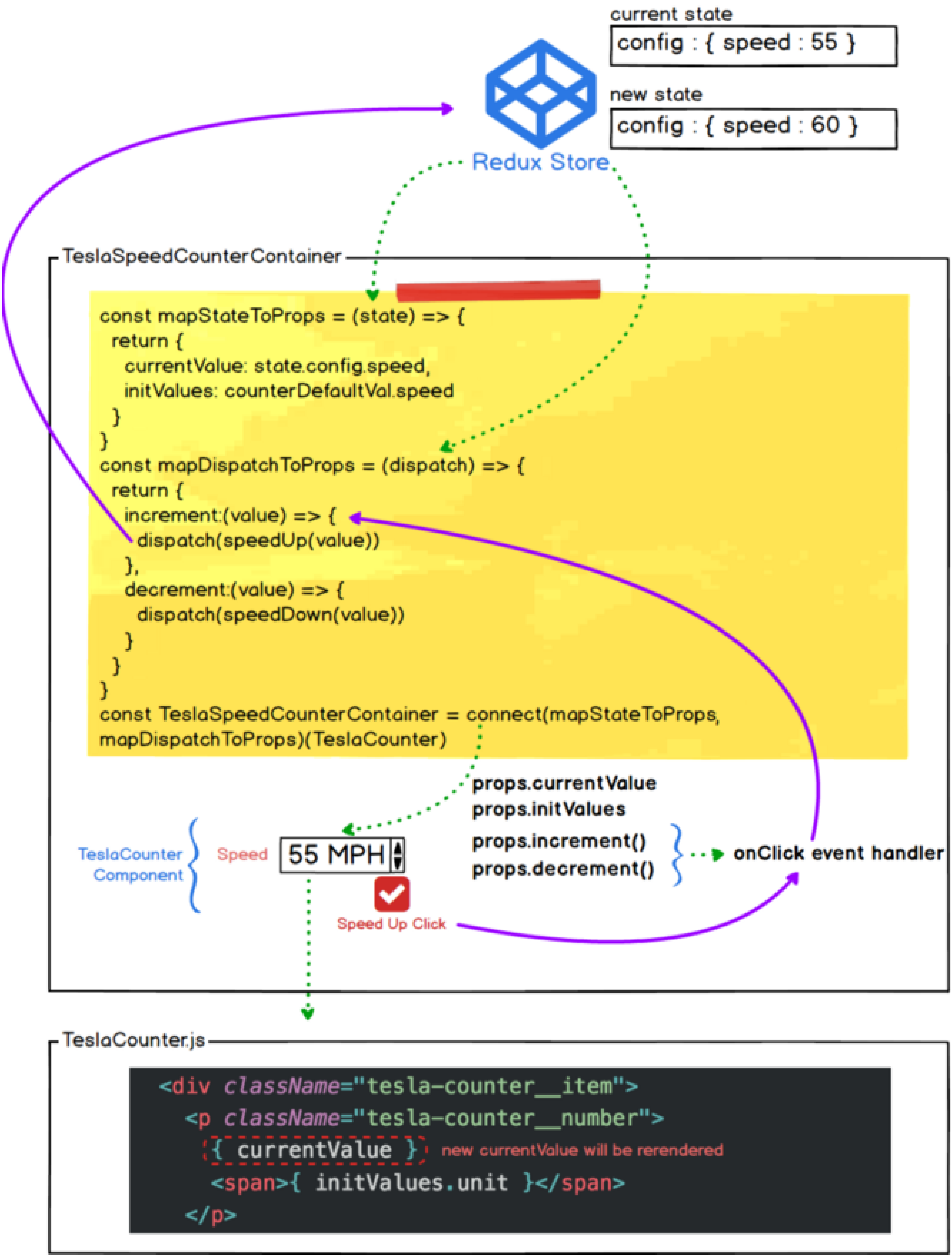
```
const mapStateToProps = (state) => {
  return {
    carstats: state.carstats
  }
}
const TeslaStatsContainer = connect(mapStateToProps, null)(TeslaStats)
```

60	60D	75	75D	90D	P100D
246Mi	250Mi	297Mi	306Mi	336Mi	376Mi

Create **src/containers/TeslaStatsContainer.js** and write the code.

2.11.4 TeslaSpeedCounter Container

The **TeslaSpeedCounter container** defines an additional `mapDispatchToProps` function to handle the user actions that occur in the TeslarSpeedCounter component.



```
const mapStateToProps = (state) => {
  return {
    currentValue: state.config.speed,
    initValues: counterDefaultVal.speed
  }
}
const mapDispatchToProps = (dispatch) => {
  return {
    increment:(value) => {
      dispatch(speedUp(value))
    },
    decrement:(value) => {
      dispatch(speedDown(value))
    }
  }
}
const TeslaSpeedCounterContainer = connect(mapStateToProps,
mapDispatchToProps)(TeslaCounter)
```

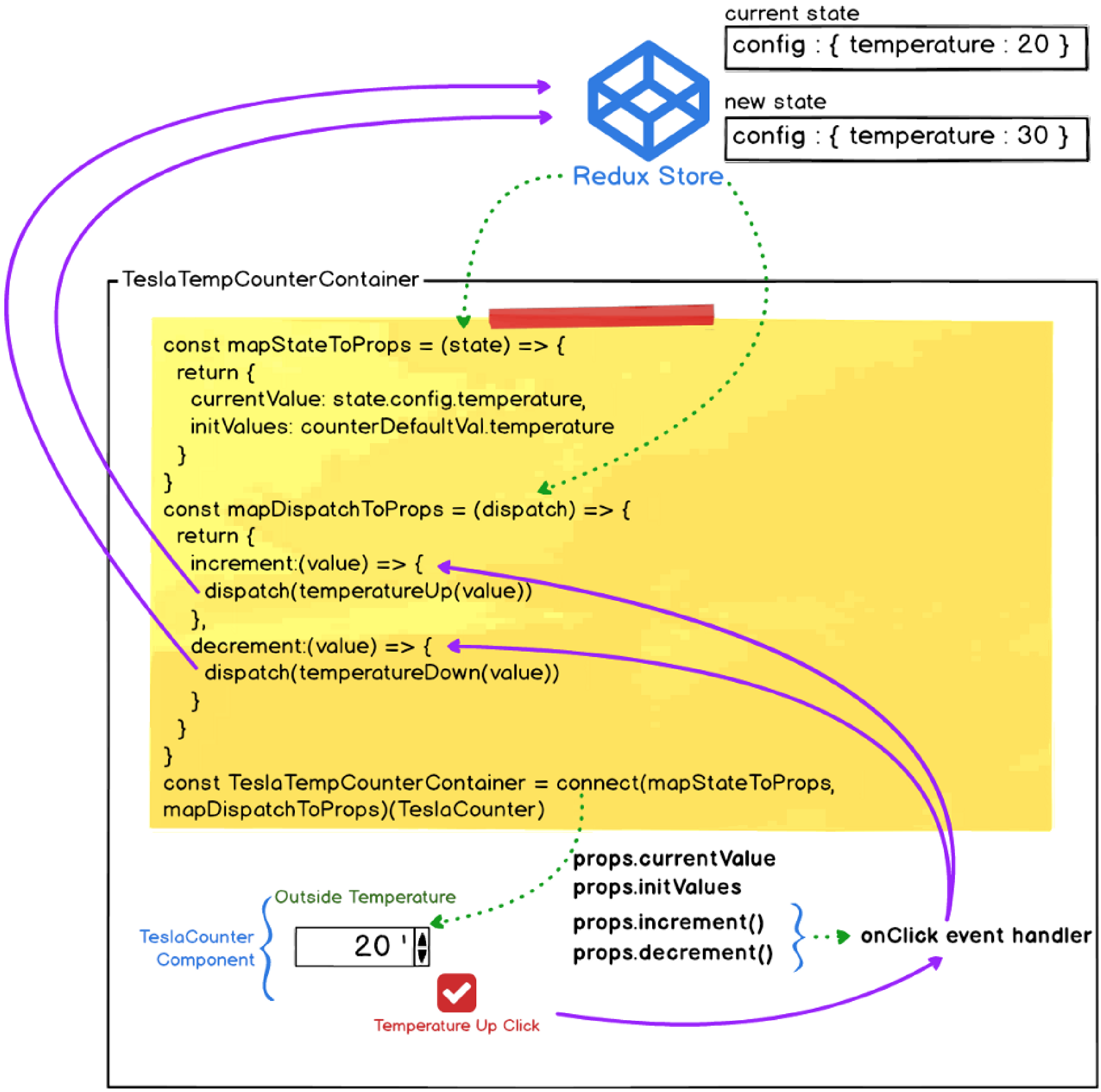
```
<div className="tesla-counter__item">
  <p className="tesla-counter__number">
    { currentValue }
    <span>{ initValues.unit }</span>
  </p>
```

Create **src/containers/TeslaSpeedCounterContainer.js** and write the code.

2.11.5 TeslaTempCounter Container

The **TeslaTempCounter container** is almost identical to the TeslaSpeedCounter except for the state and action creators being passed.

except for the state and action creators being passed.

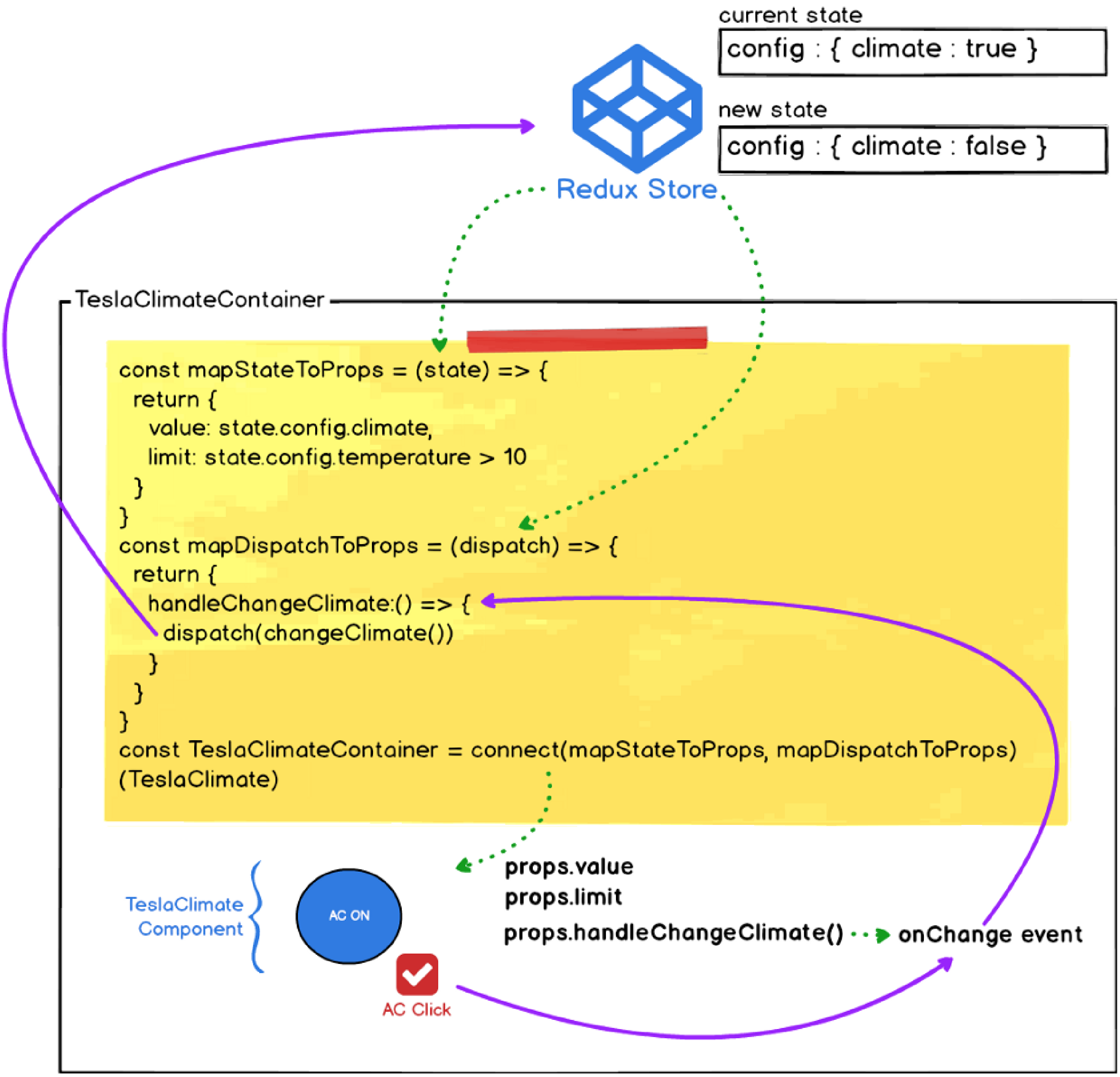


current state
config : { temperature : 20 }
new state
config : { temperature : 30 }
Redux Store
TeslaTempCounterContainer

```
const mapStateToProps = (state) => {
  return {
    currentValue: state.config.temperature,
    initValues: counterDefaultVal.temperature
  }
}
const mapDispatchToProps = (dispatch) => {
  return {
    increment:(value) => {
      dispatch(temperatureUp(value))
    },
    decrement:(value) => {
      dispatch(temperatureDown(value))
    }
  }
}
const TeslaTempCounterContainer = connect(mapStateToProps,
mapDispatchToProps)(TeslaCounter)
```

Create **src/containers/TeslaTempCounterContainer.js** and write the code.

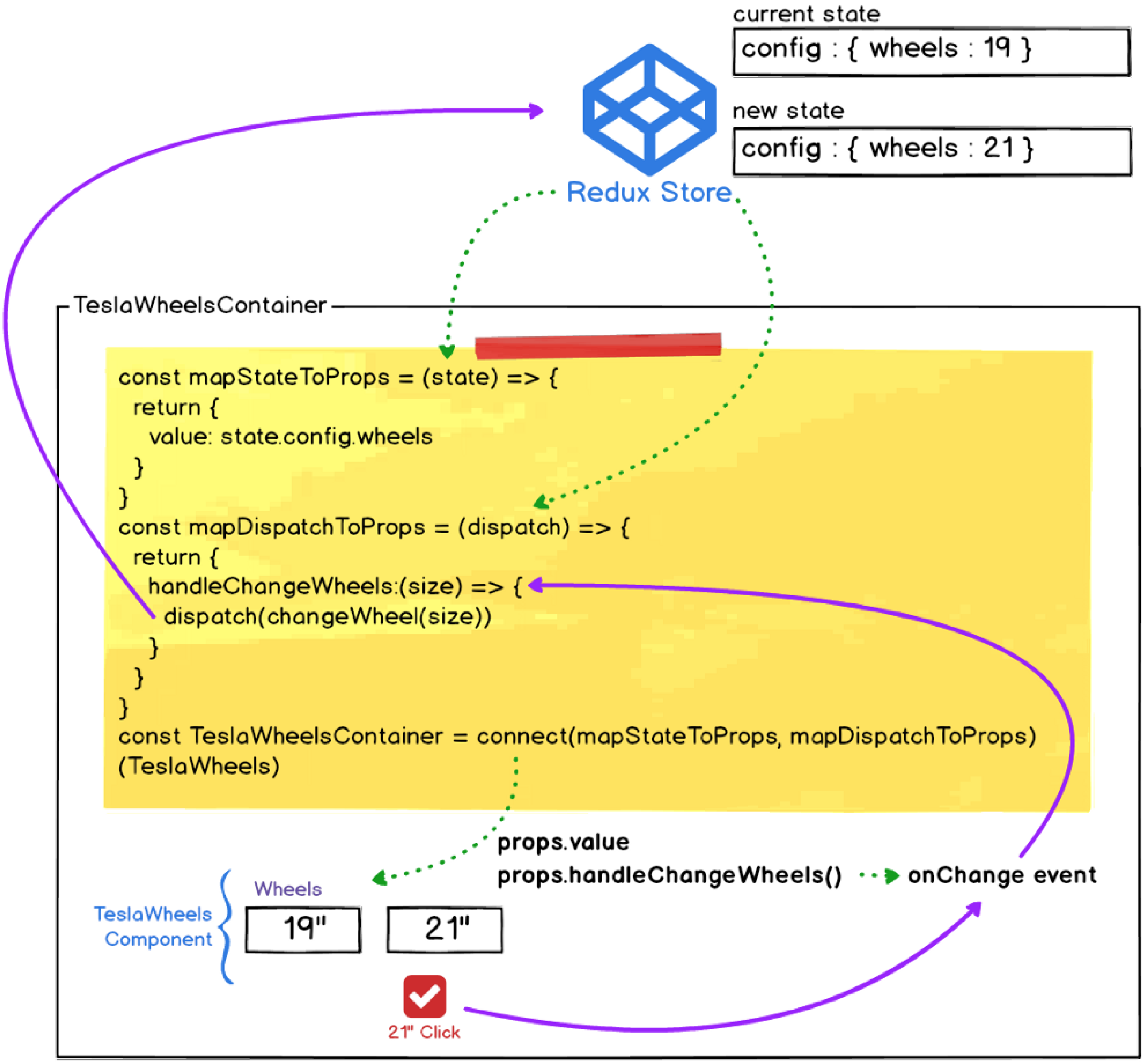
2.11.6 TeslaClimateContainer



```
const mapStateToProps = (state) => {
  return {
    value: state.config.climate,
    limit: state.config.temperature > 10
  }
}
const mapDispatchToProps = (dispatch) => {
  return {
    handleChangeClimate:() => {
      dispatch(changeClimate())
    }
  }
}
const TeslaClimateContainer = connect(mapStateToProps, mapDispatchToProps)
(TeslaClimate)
```

Create **src/containers/TeslaClimateContainer.js** and write the code.

2.11.7 TeslaWheelsContainer



```
const mapStateToProps = (state) => {
  return {
    value: state.config.wheels
  }
}
const mapDispatchToProps = (dispatch) => {
  return {
    handleChangeWheels:(size) => {
      dispatch(changeWheel(size))
    }
  }
}
const TeslaWheelsContainer = connect(mapStateToProps, mapDispatchToProps)
(TeslaWheels)
```

Create **src/containers/TeslaWheelsContainer.js** and write the code.

We have created the container components corresponding to the presentation components generated in part 1 through connect() of react-redux.

```
import { getModelData } from '../services/BatteryService';

const initialState = {
  carstats: [
    { miles: 246, model: "60" },
    { miles: 250, model: "60D" },
    { miles: 297, model: "75" },
    { miles: 306, model: "75D" },
    { miles: 336, model: "90D" },
    { miles: 376, model: "P100D" }
  ],
  config: {
    speed: 55,
    temperature: 20,
    climate: true,

    wheels: 19
  }
}

function updateStats(state, newState) {
  return {
    ...state,
    config: newState.config,
    carstats: calculateStats(newState)
  }
}

function calculateStats(state) {
  const models = ['60', '60D', '75', '75D', '90D', 'P100D'];
  const dataModels = getModelData();
  return models.map(model => {
    const { speed, temperature, climate, wheels } = state.config;
    const miles = dataModels[model][wheels][climate ? 'on' : 'off'].speed[speed][temperature];
    return {
      model,
      miles
    };
  });
}

function appReducer(state = initialState, action) {
  switch (action.type) {
    case 'CHANGE_CLIMATE': {
      const newState = {
        ...state,
        config: {
          climate: !state.config.climate,
          speed: state.config.speed,
          temperature: state.config.temperature,
          wheels: state.config.wheels
        }
      };
      return updateStats(state, newState);
    }
    case 'SPEED_UP': {
      const newState = {
          ...state,
          config: {
            climate:state.config.climate,
            speed:action.value + action.step,
            temperature:state.config.temperature,
            wheels:state.config.wheels
          }
      };
      return updateStats(state, newState);
    }
    case 'SPEED_DOWN': {
      const newState = {
          ...state,
          config: {
            climate:state.config.climate,
            speed:action.value - action.step,
            temperature:state.config.temperature,
            wheels:state.config.wheels
          }
      };
      return updateStats(state, newState);
    }
    case 'TEMPERATURE_UP': {
      const newState = {
          ...state,
          config: {
            climate:state.config.climate,
            speed:state.config.speed,
            temperature:action.value + action.step,
            wheels:state.config.wheels
          }
      };
      return updateStats(state, newState);
    }
    case 'TEMPERATURE_DOWN': {
      const newState = {
          ...state,
          config: {
            climate:state.config.climate,
            speed:state.config.speed,
            temperature:action.value - action.step,
            wheels:state.config.wheels
          }
      };
      return updateStats(state, newState);
    }
    case 'CHANGE_WHEEL': {
      const newState = {
          ...state,
          config: {
            climate:state.config.climate,
            speed:state.config.speed,
            temperature:state.config.temperature,
            wheels:action.value
          }
      };
      return updateStats(state, newState);
    }
    default:
      return state
  }
}

export default appReducer;
```