Redux

JS Library for Application State Management

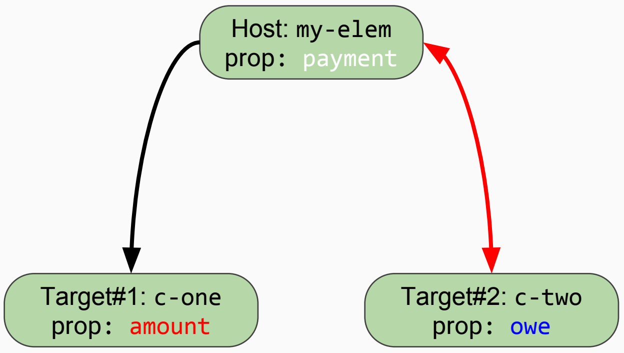
Why Use Web Components?

```
<body>
 <!-- menu --->
 <ul>
   <li><a href=”#home”>Home</a></li>
   <li><a href=”#promo”>Weekly Deals</a></li>
   <li><a href=”#search”>Search</a></li>
   <li><a href=”#orders”>Orders</a></li>
   <li><a href=”#login”>Signin</a></li>
 </ul>
 <div id=”homescreen”>
   <!-- details of home screen here -->
   <table>
     <tr>_____</tr>
     <tr>_____</tr>
   </table>
 </div>
 <div id=”promoscreen”>
   <!-- details of home screen here →
   <span>Don’t miss this one-time offer:</span>
   <ol>
     <li>
   </ol>
 </div>
 <div id=”searchscreen”>
   <span>What are you looking for?</span>
   <form _____>
   </form>
 </div>
</body>
```

```
<body>
  <main-menu>
    <menu-item>Home</menu-item>
    <menu-item>Promotion</menu-item>
    <menu-item>Search</menu-item>
    <menu-item>Orders</menu-item>
    <menu-item>Signin</menu-item>
  </main-menu>
  <page-tabs>
    <tab-item><home-screen></tab-item>
    <tab-item><promo-screen></tab-item>
    <tab-item><search-prod></tab-item>
    <tab-item><order-list></tab-item>
    <tab-item><sign-in></tab-item>
  <page-tabs>
</body>
```

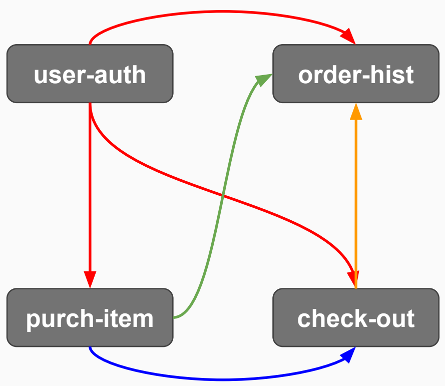
Polymer Data Binding: One-way & Two-way (Revisited)

```
<dom-module id=”my-elem”>
  <template>
    <c-one amount=[[payment]]></c-one>
    <c-two owe={{payment}}></c-two>
  </template>
</body>
```



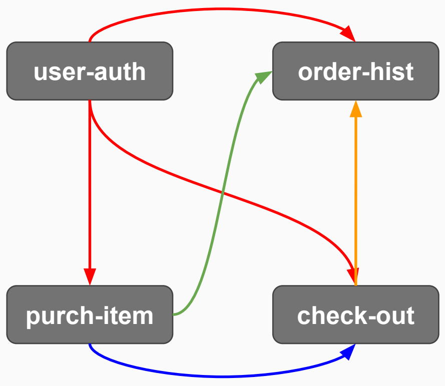
What’s The Problem?

```
<body>
  <user-authentication user=”{{u}}”>
  </user-authentication>
  <order-history user=”[[u]]”
    recent-purchase=”[[plist]]”
    purchase-confirmed=”[[done]]”>
  </order-history>
  <purchase-item
     user=”[[u]” purchase-list=”{{plist}}”
     total=”{{amtTopay}}”></purchase-item>
  <check-out user=”[[u]]”
     amount=”[[amtTopay]]” completed=”{{done}}”>
  </check-out>
</body>
```



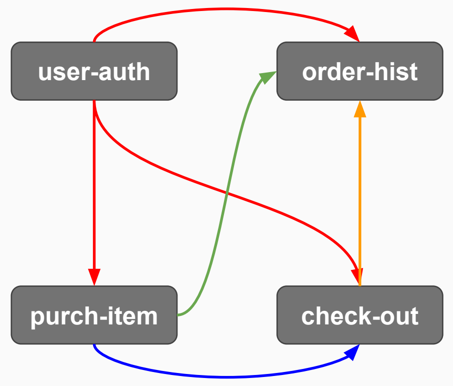
***heavyweight components** that must handle application logic*

Data Observers



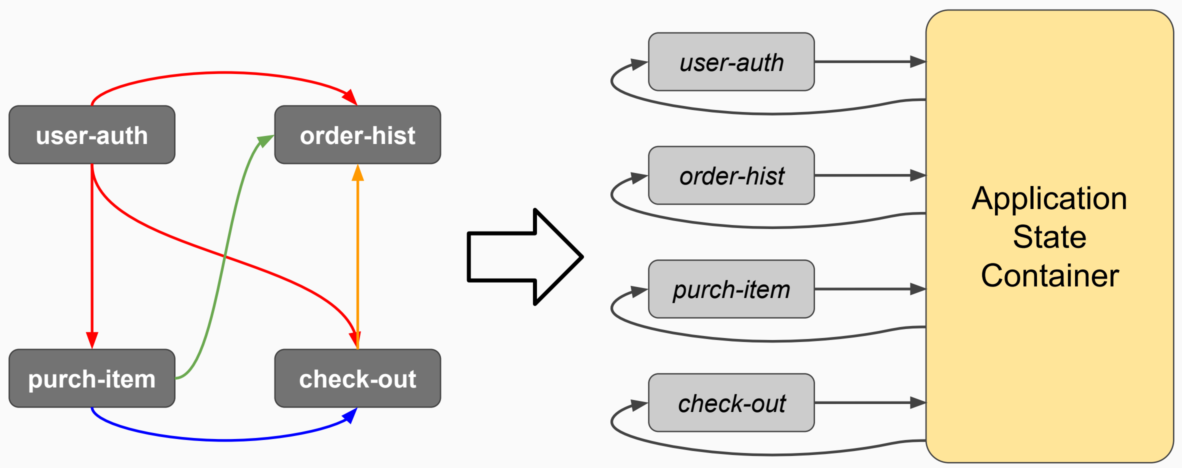
- Components that act as “data sink” must define data change observers
- Three observers in <order-history>
- One observer in <purchase-item>
- Two observers in <check-out>
- Data observers vs. Firebase Listeners?

Data Observers



- *Root of the problem*: components that employ **{{two-way}} data binding** become a data source that trigger cascading updates
- **Solution: avoid two-way data binding**

Centralized State Container



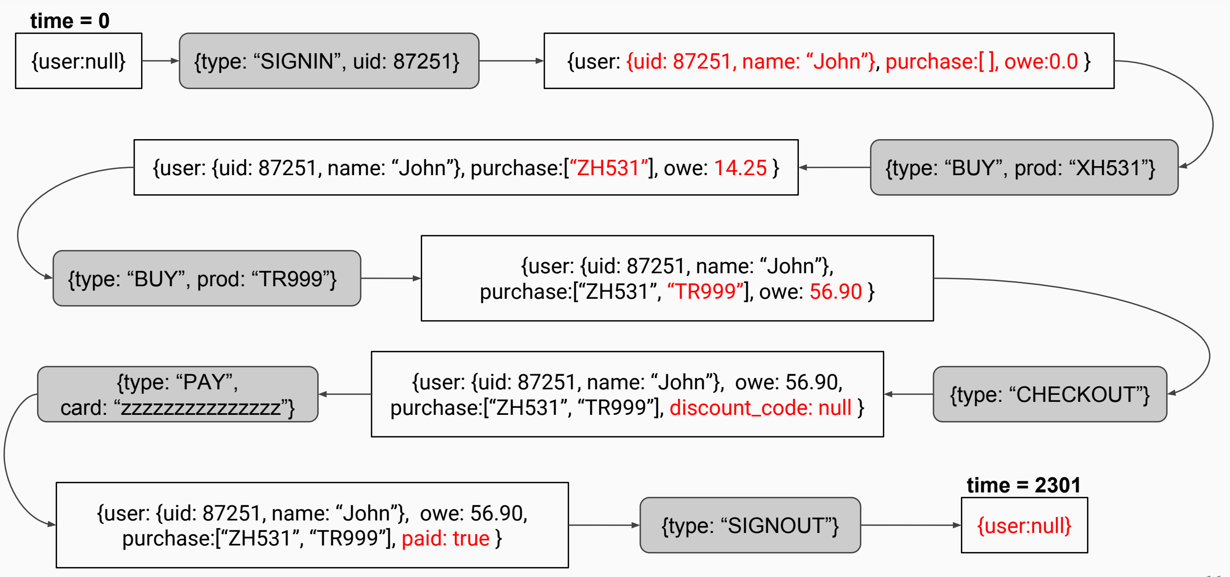
***lighweight components:** handle only UI events and update*

Application State

Application State Management

- Application state changes over time
  - Changes due to user actions (button clicks, menu selections, etc)
  - Changes due to system actions (Firebase DB listeners, download/upload completed, etc.)
- Modular components entice devopers to **distribute application state** across multiple places (individual custom elements)
  - Buggy app
  - Hard to maintain and keep track

State Transitions (over time)



```
time = 0
{user:null}
{type: “SIGNIN”, uid: 87251}
{user: {uid: 87251, name: “John”}, purchase:[ ], owe:0.0 }
{type: “BUY”, prod: “XH531”}
{user: {uid: 87251, name: “John”}, purchase:[“ZH531”], owe: 14.25 }
{type: “BUY”, prod: “TR999”}
{user: {uid: 87251, name: “John”},
purchase:[“ZH531”, “TR999”], owe: 56.90 }
{type: “CHECKOUT”}
{user: {uid: 87251, name: “John”},  owe: 56.90,
purchase:[“ZH531”, “TR999”], discount_code: null }
{type: “PAY”,
card: “zzzzzzzzzzzzzzz”}
{user: {uid: 87251, name: “John”},  owe: 56.90,
purchase:[“ZH531”, “TR999”], paid: true }
{type: “SIGNOUT”}
time = 2301
{user:null}
```

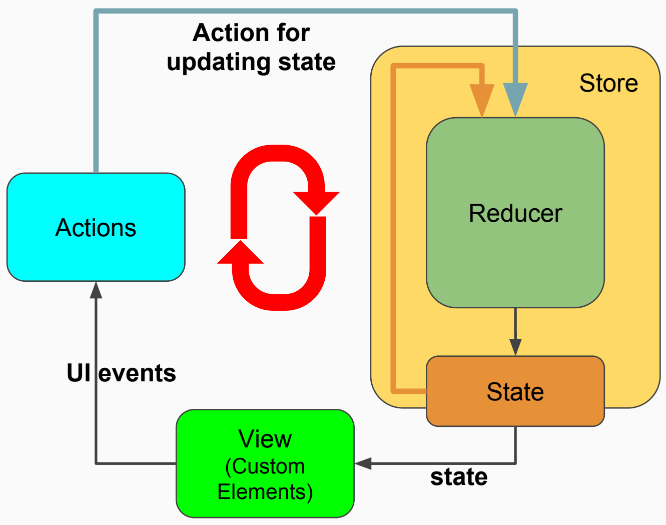
What is Redux?

Redux

- Inventor: Dan Abramov (2015)
- Single Source of Application State (Single State Tree)
- Unidirectional Data Flow
- Immutable (Read-Only) State

13

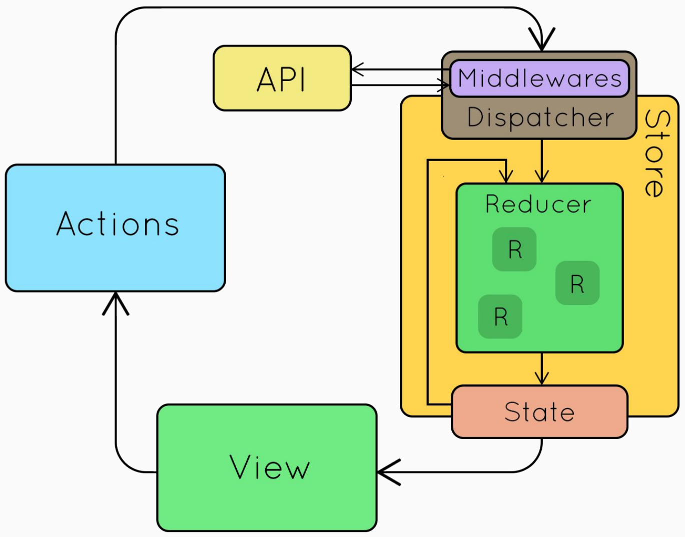
Unidirectional Data Flow



nextState = reducer (currentState, action);

14

Redux



- Incoming actions may initiate async task (such as fetching an external web service)
- The response from the async task may trigger a *follow-up action* that updates the state with more detailed data

Redux Building Blocks

Component	Description	Who Provides?
Actions	Objects that describe updates to the application state	**You**
Dispatcher	Injects actions into the reducer(s)	Redux Framework
Reducer(s)	Apply updates and determine the next application state	**You**
Subscriber	State Change Listener	Redux Framework

Redux Actions

```
{
  type: “ACTION_NAME”,
  payload: {
  }
}
```

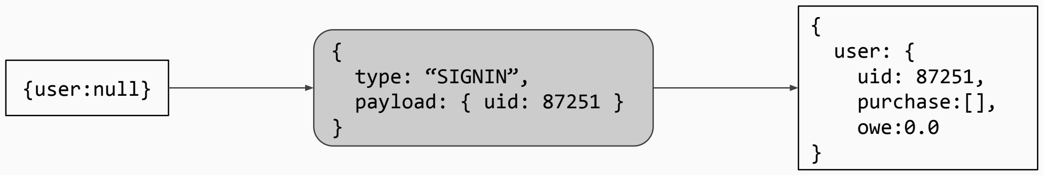
```
{
  type: “SIGN_IN”,
  payload: {
    uid: “YzU663447ER”,
    time: 1510110661,
    admin: false
  }
}
```

```
{
  type: “REVIEW”,
  payload: {
    prod: “TR981XZ”,
    rating: 4,
    comment: “Plenty of storage space”
  }
}
```

Redux Reducers

```
(state, action) ⇒ state
```

Redux Reducer(s): `(state, action) ⇒ state`



```
{user:null}
```

```
{
  type: “SIGNIN”,
  payload: { uid: 87251 }
}
```

```
{
  user: {
    uid: 87251,
    purchase:[],
    owe:0.0
}
```

```
function sampleReducer(state, action) {
  if (typeof state == ‘undefined’)
    return {};     /* default state */
  switch (action.type) {
     case “SIGNIN”:
       return Object.assign({},
                                  state, action.payload, {purchase:[], owe:0.0});
     case “PAY”:
       return ______;
  }
  return state;
}
```

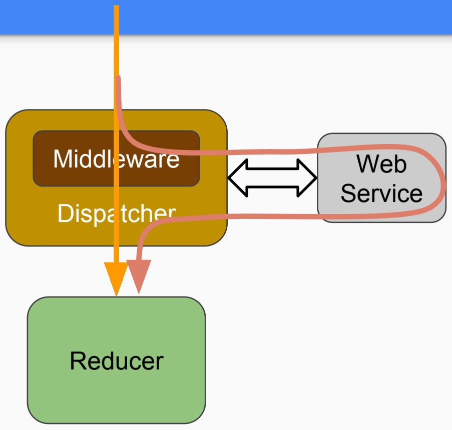
Using Redux Dispatcher & Subscriber

```
// In your custom element function
// (possibly event handler)
store.dispatch ({
  type: “SIGNIN”,
  payload: {
    uid : ____
  }
});
```

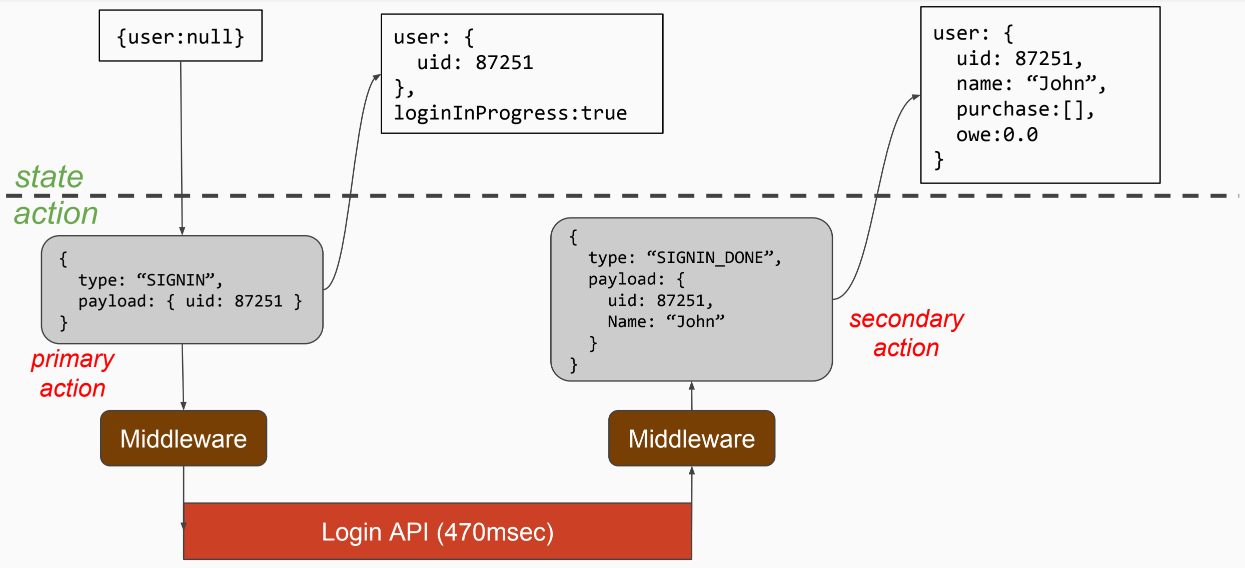
```
// In your custom element ready()
// or connectedCallback()

store.subscribe (() => {
  var currentState = store.getState();
  //
  // Update the UI based on the
  // current state
});
```

Redux Middleware

- Some actions require “side effect”
    - Signin requires verification of user & password to a remove authentication server
    - File upload requires “oncompletion” callback
- Middleware handles the desired “side effect” and dispatch a ***secondary action*** when the “side effect” completes



Middleware Secondary Actions (Example: Login Sequence)



```
user: {
  uid: 87251
},
loginInProgress:true
```

```
user: {
  uid: 87251,
  name: “John”,
  purchase:[],
  owe:0.0
}
```

```
{
  type: “SIGNIN”,
  payload: { uid: 87251 }
}
```

```
{
  type: “SIGNIN_DONE”,
  payload: {
    uid: 87251,
    Name: “John”
  }
}
```

Redux Summary

1. Define unique action verbs (and payload) for your app
2. Write reducer function
3. Supply the reducer function when creating the Redux store
    - The redux store is a global object throughout your webapp
4. Dispatch primary actions from UI event handling functions
5. Dispatch secondary actions from Middleware

JavaScript Syntax for Functions

```
function doWork (one, two) {
  /* code here */
}
```

```
var doWork = function (one, two)
{
  /* code here */
}
```

```
var doWork = (one, two) => {
  /* code here */
}
```

Redux & Polymer 2.0

Step 1: Setup and Download Dependencies

Step 1: Download Dependencies

```
> polymer init                       # will create polymer.json
> bower install --save polymer-redux
> npm init                           # to create package.json
> npm install --save redux           # will create subdir node_modules
```

Two subdirectories:

- `bower_components`: 3rd party custom elements
- `node_modules`: Node.js modules

Step 2: Define Redux Mixin

(write your reducer function)

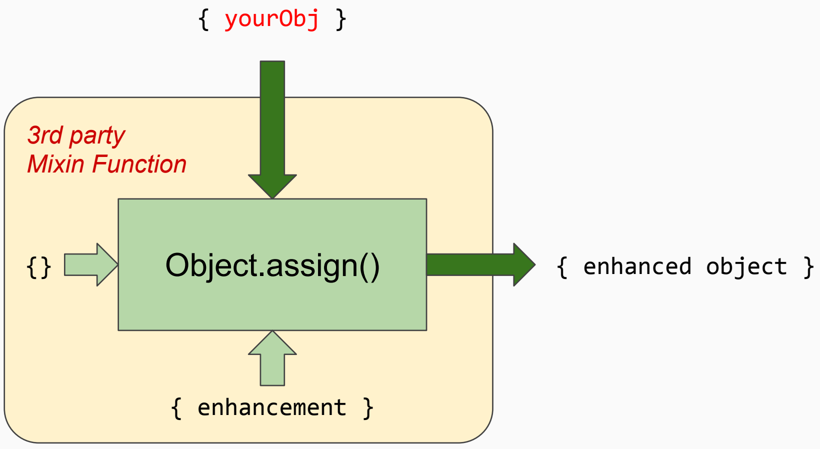
Redux and Polymer 2.0

```
<!-- redux-mixin.html -->
<link rel=”import” href=”../../bower_components/polymer-redux/polymer-redux.html”>
<script src=”../../node_modules/redux/dist/redux.js”></script>
<script>
  const initialState = {};

  const myReducer = (state = initialState, action) => {
    switch (action.type) {
      case “______”: return _____;
      case “______”: return _____;

      default: return state;          /* required !!! */
    }
  };
  const store = Redux.createStore(myReducer);
  ReduxMixin = PolymerRedux(store);    // ReduxMixin has a GLOBAL scope
</script>
```

Step 3: Use Mixin in Custom Elements

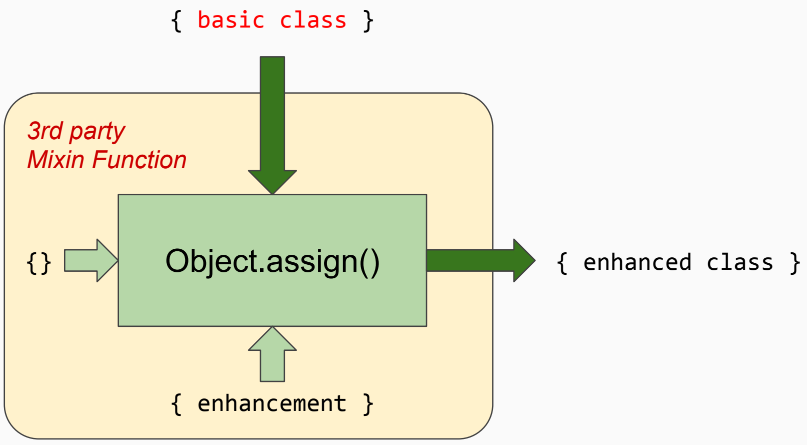
Mixin by 3rd party libraries

```
enhancedObj = MixinByThirdPartyLib (yourObj)
```



Mixin by 3rd party libraries

```
enhancedClass = MixinByThirdPartyLib (basicClass)
```



Defining Custom Element with Mixin

```
<link rel=”import” href=”../../bower_components/polymer/polymer-element.html”>
<link rel=”import” href=”redux-mixin.html”>
<dom-module id=”sam-ple”>
  <script>
    class Sample extends ReduxMixin(Polymer.Element) {
      static get is() { return ‘sam-ple’; }

    }
    customElements.define (Sample.is, Sample);
  </script>
</dom-module>
```

ReduxMixin enhances Polymer.Element with Redux functionalities

Dispatching Actions

```
<link rel=”import” href=”../../bower_components/polymer/polymer-element.html”>
<link rel=”import” href=”redux-mixin.html”>
<dom-module id=”sam-ple”>
  <template>
    <paper-button on-click=”doIt”>OK</paper-button>
  </template>
  <script>
    class Sample extends ReduxMixin(Polymer.Element) {
      static get is() { return ‘sam-ple’; }

      doIt() {
        this.dispatch ( {type: _______, payload: ______}); // Redux dispatcher
      }
    }
    customElements.define (Sample.is, Sample);
  </script>
</dom-module>
```

Object.assign() examples

```
Object.assign({}, {num: 5, flag: false}, {name:”GLX”}) ⇒ {num: 5, flag: false, name:”GLX”}
```

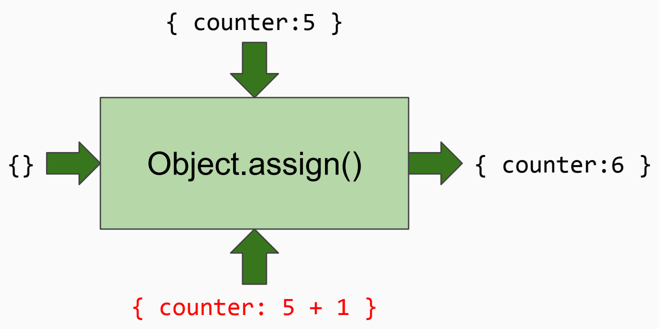
```
Object.assign({}, {num: 5, flag: false}, {flag:true}) ⇒ {num: 5, flag: true}
```

```
Object.assign({}, {num: 5, flag: false}, {name:”GLX”, num: 10}) ⇒
                                                     {num: 10, flag: false, name:”GLX”}
```

```
Object.assign({}, {num: 5, flag: false}, {flag: “YES”}) ⇒ {num: 5, flag: “YES”}  // AVOID!!!
```

JavaScript Object.assign()

```
state = { counter: 5 };
Object.assign({}, state, {counter: state.counter + 1})
```



Redux State Change Listener

```
// Using plain JavaScript

store.subscribe (() => {
  var currentState = store.getState();
  if (currentState.user != null) {
    // Update the UI based on the
    // current state
  }
});
```

```
// Using Polymer-Redux
<template>
  <span>You login as [[who]]</span>
</template>
<script>
// Using Polymer-Redux
class XYZ extends ReduxMixin(Polymer.Element) {
  static get properties() {
    return {
      who : {
        type: String,
        statePath: "user"
      }
    }
  }
}
</script>
```

Subscribe To State Changes

```
<link rel="import" href="../../bower_components/polymer/polymer-element.html">
<link rel="import" href="redux-mixin.html">
<dom-module id="sam-ple">
  <template>
    <span>Hello [[world]]</span>
  </template>
  <script>
    class Sample extends ReduxMixin(Polymer.Element) {
      static get is() { return 'sam-ple'; }

      static get properties() {
        return {
          world: {
            type: Number, statePath: 'counter'  // counter is a state in Redux store
          }
        }
      }
    }
    customElements.define (Sample.is, Sample);
  </script>
</dom-module>
```

Design with Redux

Design Steps

1. Create a list of action verbs and their corresponding payload
2. Determine important states of your application
3. Use switch-case statment in reducer function; for each action verb
    a. Create one case label
    b. Implement the logic of updating the state with the data passed in the payload
    c. Return a new state object

Example: Increment/Decrementer

1. Action verbs: INC, DEC, INC_BY, DEC_BY
   a. `{type: “INC”}`
   b. `{type: “DEC”}`
   c. `{type: “INC_BY”, payload: { amt: __ }}` or `{type: “INC_BY”, amt: ___ }`
   d. `{type: “DEC_BY”, payload: { amt: __ }}` or `{type: “DEC_BY”, amt: ___ }`
2. State variable: `counter` (Number)
3. Reducer function (next slide)

Reducer Function

```
// pseudocode
function myReducer(state, action) {
  switch (action.type) {
    case “INC”:
      nextState.counter = state.counter + 1;
      break;
    case “INC_BY”:
      nextState.counter = state.counter + action.payload.amt;
      break;
  }
  return nextState;
}
```

```
action: {
  type: “INC”
}
```

```
action: {
  type: “INC_BY”,
  payload: {
    amt: ____
  }
}
```

Reducer Function

```
function myReducer(state, action) { // pseudo code
  switch (action.type) {
    case “INC”:
      nextState.counter = state.counter + 1;
      break;
    case “INC_BY”:
      nextState.counter = state.counter + action.payload.amt;
      break;
  }
  return nextState;
}
```

```
const myReducer = (state, action) => {
  switch (action.type) {
    case “INC”:
      return Object.assign({},
              state, { counter: state.counter + 1 });
    case “INC_BY”:
      return Object.assign({},
              state, { counter: state.counter + action.payload.amt });
    default:
      return state; /* unchanged state */
  }
}
```

Reducer Example: Incrementer/Decrementer

```
<!-- redux-mixin.html -->
<link rel=”import” href=”../../bower_components/polymer-redux/polymer-redux.html”>
<script src=”../../node_modules/redux/dist/redux.js”></script>
<script>
  const initialState = {counter: 0};
  const myReducer = (state = initialState, action) => {
    switch (action.type) {
      case “INC”:
        return Object.assign({}, state, {counter: state.counter + 1});
      case “INC_BY”:
        return Object.assign({}, state,
           {counter: state.counter + action.payload.amt } );
      default: return state;          /* required !!! */
    }
  };
  const store = Redux.createStore(myReducer);
  ReduxMixin = PolymerRedux(store);    // ReduxMixin has a GLOBAL scope
</script>
```

Demo:
Data Producer & Consumer

Demo Details

- Objective: show how components communicate via Redux store
- Two elements
  - `data-source`: a producer that generates data
  - `data-sink`: a consumer that consumes the data
- One state variable
  - counter: Number
- Two actions:
  - `{ type: ‘INC’}`
  - `{ type: ‘DEC’}`

Polymer-Redux Summary

1. Setup (polymer and npm)
2. Write your reducer(s)
3. Create a Redux Store and Mixin
4. Declare custom elements using Mixin
    a. `class XYZ extends ReduxMixin(Polymer.Element)`
5. Inside custom elements
    a. Inside an event handler call `this.dispatch({_____})` to dispatch an action
    b. Connect selected properties to Redux state using ‘`statePath`’ keyword

Redux Middleware

Using Middleware

- Without Middleware: actions = objects
- To enable Middleware: actions = objects OR functions
- Redux plugin: redux-thunk

```
> npm install --save redux-thunk

# new node_modules/redux-thunk/dist/redux-thunk.js
```

Redux-Thunk and Polymer 2.0

```
<!-- redux-mixin.html -->
<link rel=”import” href=”../../bower_components/polymer-redux/polymer-redux.html”>
<script src=”../../node_modules/redux/dist/redux.js”></script>
<script src=”../../node_modules/redux-thunk/dist/redux-thunk.js”></script>
<script>
  const initialState = {};

  const myReducer = (state = initialState, action) => {

  };
  const store = Redux.createStore(myReducer,
     Redux.applyMiddleware (ReduxThunk.default));
  ReduxMixin = PolymerRedux(store);    // ReduxMixin has a GLOBAL scope
</script>
```

Dispatching Functions (as an action)

```
myEventHandler() {
  this.dispatch( function(dsptch, getState) {
    /* work */
  });
}
```

*When the red function is invoked (by the Redux Store), it is passed an instance of the Redux **dispatcher** and Redux **getState** function*

```
myEventHandler() {
  this.dispatch( (dsptch, getState) => {
     /* work */
     dsptch({type: ‘ACT1’, payload: { ___ }});

     /* do other work here */
     dsptch({type: ‘ACT2’, payload: { ___ }});
  });
}
```

Dispatching Functions (as an action)

```
myEventHandler() {
  this.dispatch((dsptch) => {
    dsptch({type: ‘SIGNIN’, payload: {user: “me”}});

    firebase.auth().signInWithEmailAndPassword(_____)
      .then((result) => {

        dsptch({type: ‘SIGNIN_DONE’, payload: {user: “me”, email: ____}});

      });
  });
}
```

Organizing Actions

- The same action may be dispatched from several places
  - Several places in one custom element
  - Across multiple custome elements
    - ‘SIGN_OUT’
    - ‘SHOW_ORDERS’: from order list, from purchase history, etc
- Use action creaters in place of action object literals
  - ~~this.dispatch ({type: ‘SHOW_ORDER’, payload: { start: “xxxxxx”, end: “yyyyyy”}})~~ ✕
  - this.dispatch (‘showOrder’, “xxxxxx”, “yyyyyy”); ✔
- Enhancement by mixin function compositon

Polymer Static Getter Functions

- `is()`: returns a string
- `properties()`: returns JS object (key-value pairs)
  - Key: name of property
  - Value: characteristics of each property
- `observers()`: returns an array of function names (as strings)
- **`actions()`**: returns an object (key-value pairs)
  - Key: function name, value: function body
  - Each function returns either an **immediate action object** or **a middleware function**

JavaScript: a list of functions

```
var myFunList = {
   addFive: function(a) { return a + 5 ; },

   addFiveTo(a) { return a + 5; },

   addBoth(x, y) { return x + y; },

   createFun (x,y) {
     return function(a) {  return a * (x - y); };
   }
};

myFunList.addFive(23);   // returns 28

funnel = myFunList.createFun (8, 3); // funnel is a function with ONE arg
// funnel(a) { return a * 5; }

funnel(4);               // returns 20
```

Action Creator (in a separate HTML)

```
<!-- in action-mixin.html -->
<link rel=”import” href=”./redux-mixin.html”>
<script>
  ActionMixin = Parent => class ActionMixin extends ReduxMixin(Parent) {
    static get actions() {  /* ACTION CREATOR */
      return {
        showOrder (startDate, endDate) {
          return {type: ‘SHOW_ORDER’, payload: {
                                        start: startDate,
                                        end: endData}};
        },
        anotherActionFunction() {
        }
      }
    }
  }
</script>
```

Use this technique for actions shared across multiple components

ReduxMixin => ActionMixin

Without action creator

```
<link rel="import"
href="../../____/__/polymer-element.html">
<link rel="import" href="redux-mixin.html">
<dom-module id="sam-ple">
  <template>
    <paper-button on-click="doIt">Show</paper-button>
  </template>
  <script>
    class Sample extends ReduxMixin(Polymer.Element) {
      static get is() { return 'sam-ple'; }

      doIt() {
        this.dispatch ( {type: 'SHOW_ORDER',
                         payload: ______});
      }
    }
    customElements.define (Sample.is, Sample);
  </script>
</dom-module>
```

With action creator

```
<link rel="import"
href="../../____/__/polymer-element.html">
<link rel="import" href="action-mixin.html">
<dom-module id="sam-ple">
  <template>
    <paper-button on-click="doIt">Show</paper-button>
  </template>
  <script>
    class Sample extends ActionMixin(Polymer.Element) {
      static get is() { return 'sam-ple'; }

      doIt() {
        this.dispatch ('showOrder', '090317', 110417');
      }
    }
    customElements.define (Sample.is, Sample);
  </script>
</dom-module>
```

Action Creator (in a custom element HTML)

```
<!-- in custom-sample.html -->
<link rel="import" ________>
<link rel="import" href="./redux-mixin.html">
<script>
  class CustomSample extends ReduxMixin(Polymer.Element) {
    static get is() { return 'custom-sample'; }

    static get actions() {  /* ACTION CREATOR */
      return {
        showOrder (startDate, endDate) {
          return {type: 'SHOW_ORDER', payload: { start: StartDate, end: endData}};
        },
        checkOut (amount, paymentMethod) {
          return { type: 'CHECK_OUT', payload: { ____ } };
        }
      }
    }
  }
</script>
```

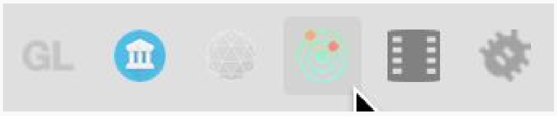
Action Creator Details: Object creator and Function creator

```
static get actions() {  /* ACTION CREATOR */
  return {
    showOrder (startDate, endDate) {
      return {type: ‘SHOW_ORDER’, payload: { start: StartDate, end: endData}};
    },
    login (us, pw, auth) {   // returns a function for Middleware
      return function(dispatch, getState) {
        dispatch ({type: ‘LOGIN’, payload: { user: us, password: pw}});
        auth.verifyEmailPassword(us, pw)
          .then(result => {
            dispatch ({type: ‘LOGIN_DONE’});
          });
      }
    },
    login2 (us, pw, auth) {   // alternate syntax for middleware functions
      return (dispatch, getState) => {
        /* same contents as above */
      }
    }
  };
}
```

Debugging Application State

Browser Extensions

- Redux DevTools
  - Chrome Extensions or Firefox Add-Ons
  - http://github.com/zalmoxisus/redux-devtools-extension



Using Redux Tools Browser Extension

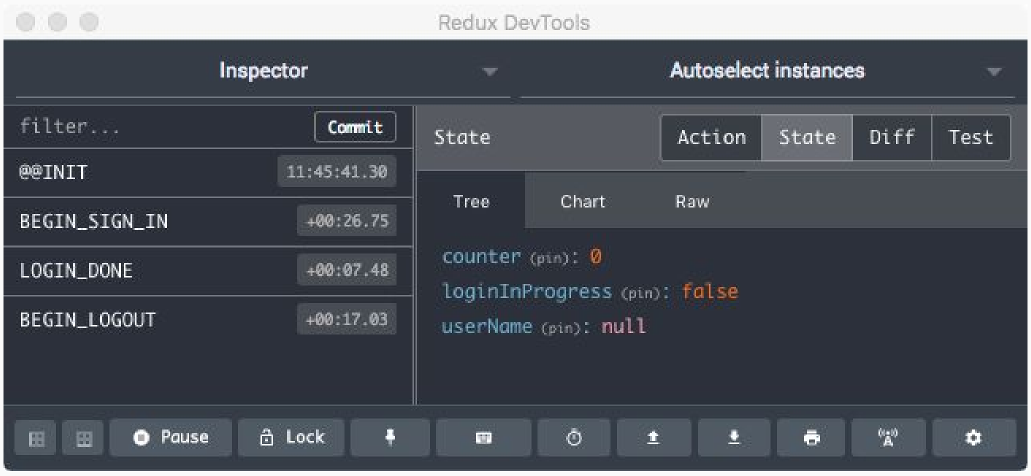
Without Redux DevTools

```
<!-- redux-mixin.html -->
<script>
// other code not shown
const store = Redux.createStore(myReducer,
   Redux.applyMiddleware (ReduxThunk.default));
ReduxMixin = PolymerRedux(store);    // ReduxMixin has a GLOBAL scope
</script>
```

With Redux DevTools

```
<!-- redux-mixin.html -->
<script>
// other code not shown
const store = Redux.createStore(
  myReducer,
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__(),
  Redux.applyMiddleware (ReduxThunk.default));
ReduxMixin = PolymerRedux(store);    // ReduxMixin has a GLOBAL scope
</script>
```

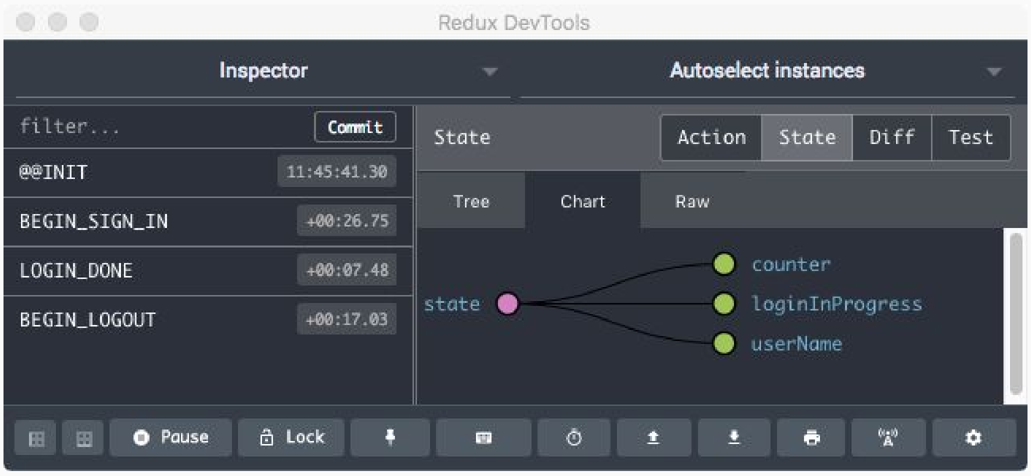
State Inspectors (Tree View)



```
@@INIT            11:45:41.30
BEGIN_SIGN_IN     +00:26.75
LOGIN_DONE        +00:07.48
BEGIN_LOGOUT      +00:17.03
```

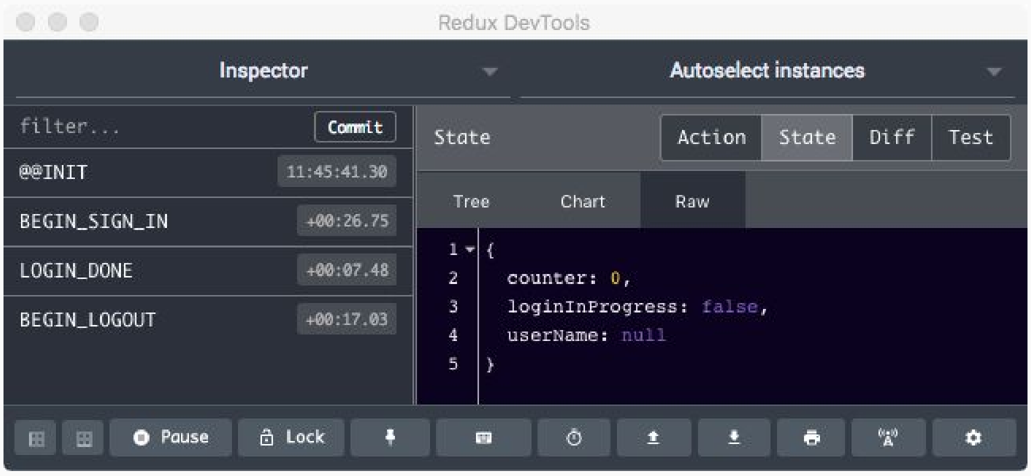
```
counter (pin): 0
loginInProgress (pin): false
userName (pin): null
```

State Inspectors (Chart View)



```
@@INIT            11:45:41.30
BEGIN_SIGN_IN     +00:26.75
LOGIN_DONE        +00:07.48
BEGIN_LOGOUT      +00:17.03
```

State Inspectors (Raw View)



```
@@INIT            11:45:41.30
BEGIN_SIGN_IN     +00:26.75
LOGIN_DONE        +00:07.48
BEGIN_LOGOUT      +00:17.03
```

```
{
  counter: 0,
  loginInProgress: false,
  userName: null
}
```

65

Selectors

- Each action requires one switch-case label in the reducer function
- Large applications require a large number of actions
    - As many switch-case labes in the reducer function, harder to maintain code
- Solution: split the monolithinc reducer function into to task specific reducer functions
    - Authentication reducer function
    - User settings reducer function
    - Purchase Order reducer function
    - Etc

68