Jeff Barczewski

The story of redux-logic, a new approach to organizing business logic with Redux

Jeff Barczewski

- Catholic
- Married
- Father

- Aerospace Engineer
- 27 years as professional developer
- US Air Force
- RGA
- Consultant
  - Elsevier
  - MasterCard (ApplePay / MDES)
- Founded CodeWinds Training

CodeWinds Training

- Live training (in-person or webinar)
- Self-paced video training classes (codewinds.com)
- Need training for your team on any of these?
  - React
  - Redux
  - RxJS
  - JavaScript
  - Node.js
  - Functional approaches
- I'd love to work with you and your team
- I appreciate any help in spreading the word.

Familiar with Redux?

- General familiarity with Redux?
- Used Redux?
- Using Redux daily?
- Expert with Redux and could be giving this talk?


Where do we put our business logic in Redux so that we aren't painting ourselves into a corner?

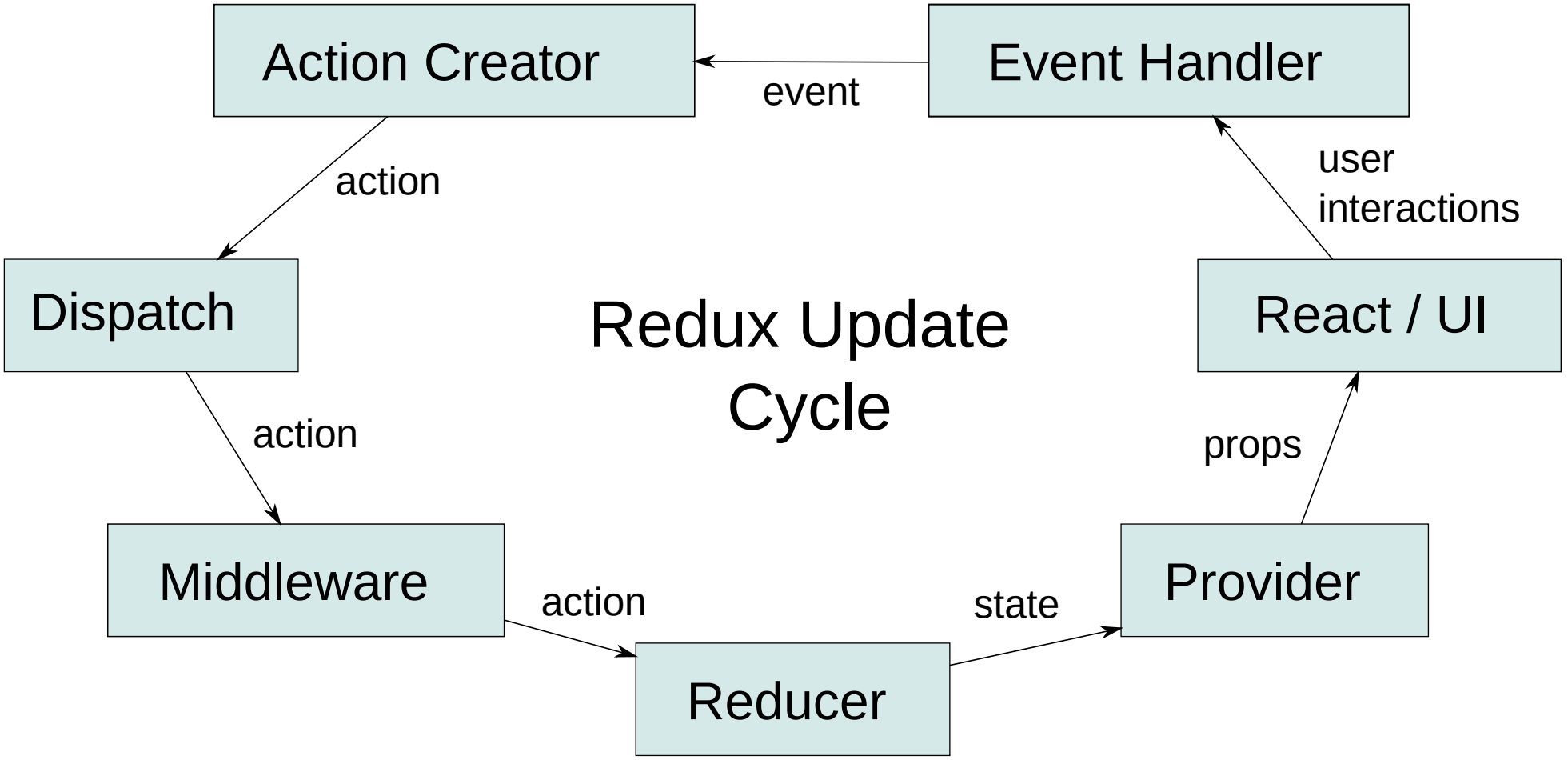
What is business logic?

What is business logic?

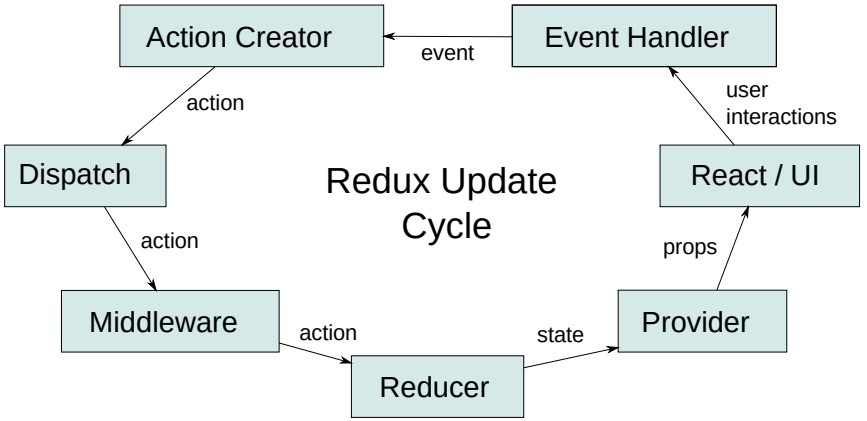
- Validation – name and email required, name < 40 chars
- Verification – Do you have sufficient credits for this transaction?
- Authorization – Does your membership allow you access to this content?
- Transformation – Unauthorized action converted to upsell popup action
- A/B Logic – Half users get X, half get Y
- Augment – Add unique ID; Fill in user profile using uid
- Silence/filter – Resource is at max, ignore further increases
- Async Processing – Fetch data, post order, hide notification after delay, subscribe to updates
- Cancellation – Navigate to different page or user, change search, drag and drop, animation
- Debouncing – Wait till done typing before searching
- Throttling – Game allows you to shoot every 100ms

Goals

Full state
Dispatch
Intercept (validate/transform)
Async Processing
Cancellation / Latest
Filter / debounce / throttle
Apply across many types
One place for all logic
Simple
Load from split bundles

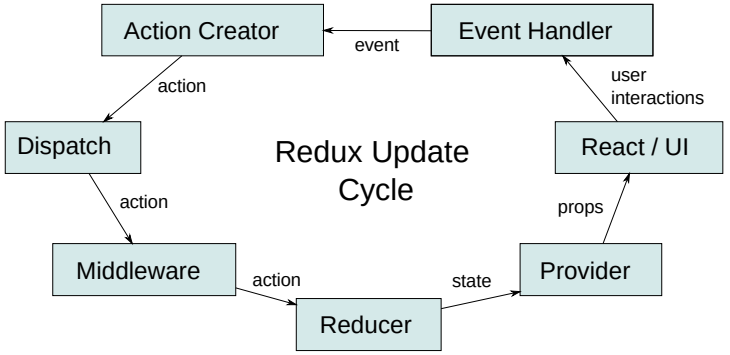

Action Creator
event
Event Handler
action
user
interactions
Dispatch
Redux Update
Cycle
React / UI
action
props
Middleware
action
state
Provider
Reducer

Reducer




```
function r(state, action) {
    // calc new state
    return newState;
}
```

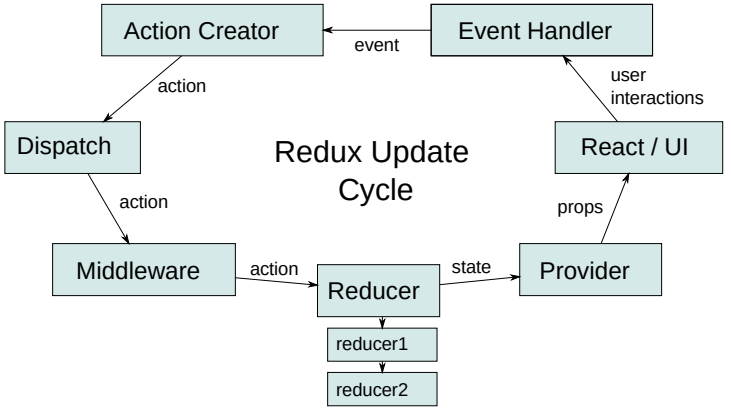
Reducer 2

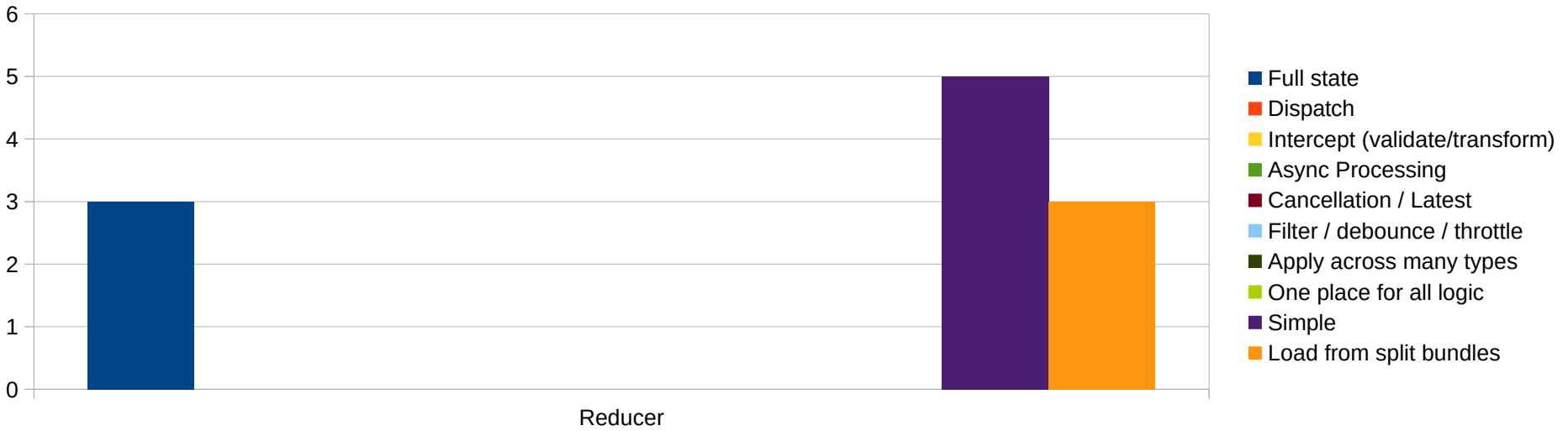



```
// full state
const state = {
    foo: {…},
    bar: {...}
};
```

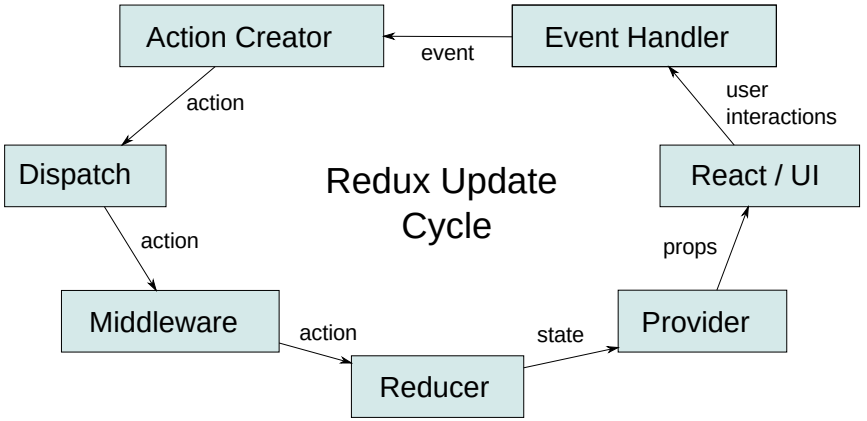
```
const reducer =
    combineReducers({
        foo: fooRed,
        bar: barRed
    });

function fooRed(state, action) {
    // calc new state
    return newState;
}
```

Reducer 3


Action Creator
event
Event Handler
action
user
interactions
Dispatch
Redux Update
Cycle
React / UI
action
props
Middleware
action
Reducer
state
Provider
reducer1
reducer2

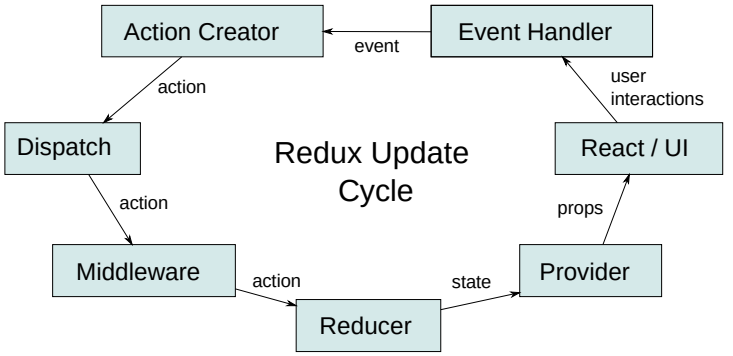

6
5
4
3
2
1
0
Reducer
Full state
Dispatch
Intercept (validate/transform)
Async Processing
Cancellation / Latest
Filter / debounce / throttle
Apply across many types
One place for all logic
Simple
Load from split bundles

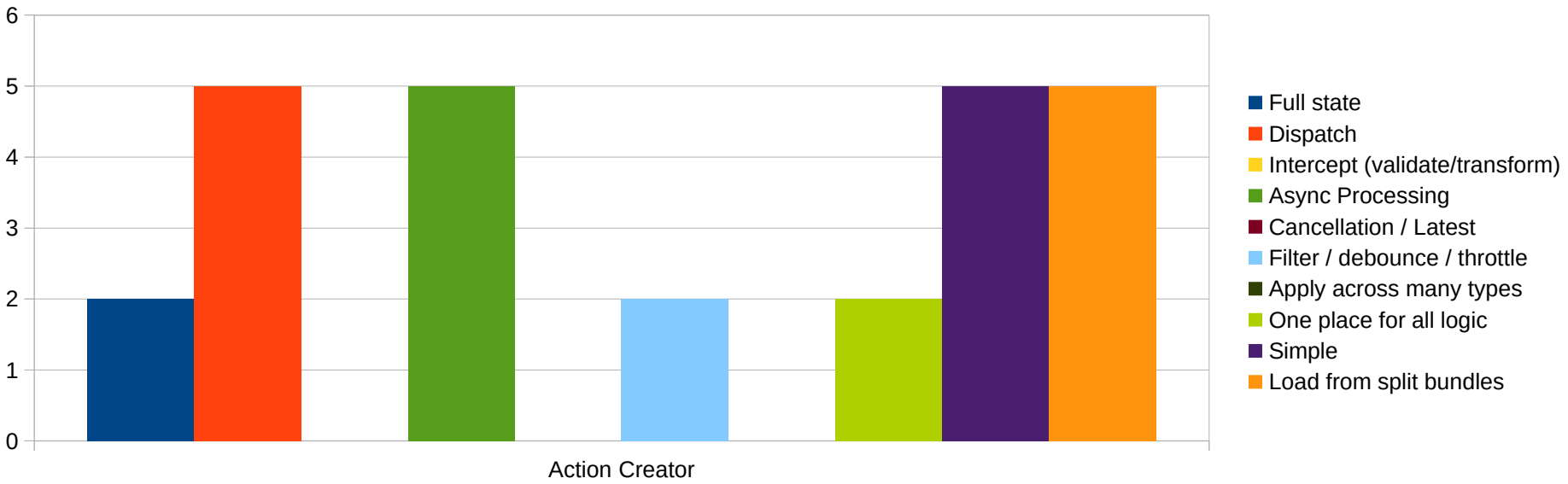
Action Creator



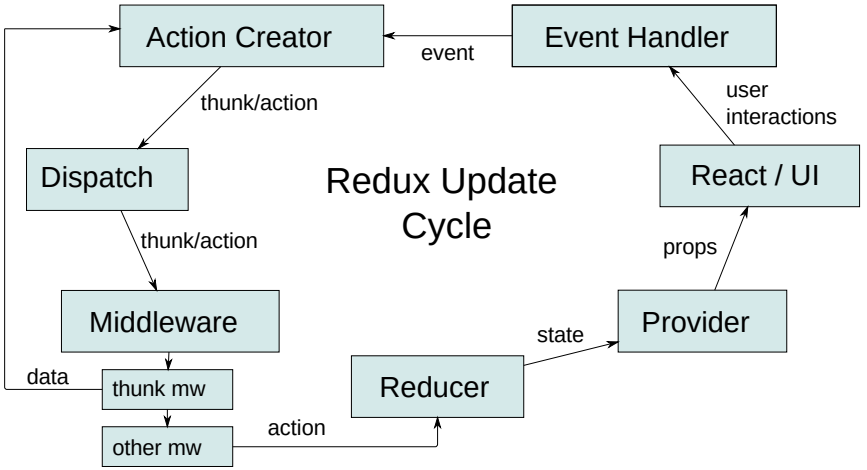

```
// thin action creator
function buyClicked(ev, a, b) {
  return { type: BUY };
}

// fat action creator
function buyClicked(dispatch, ev) {
  // sync or async code here
  dispatch({ type: BUY });
}
```

Action Creator 2



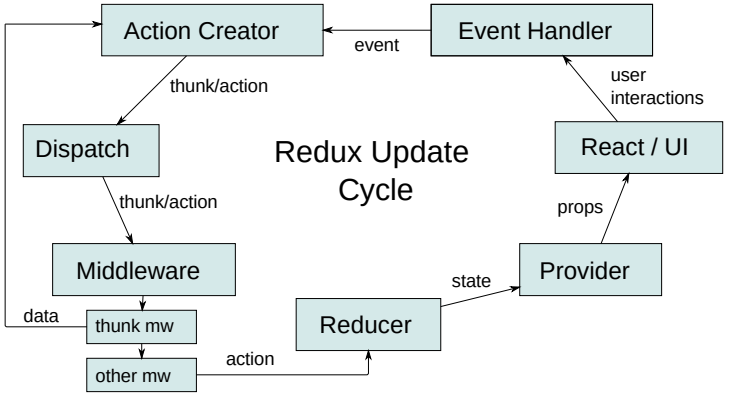


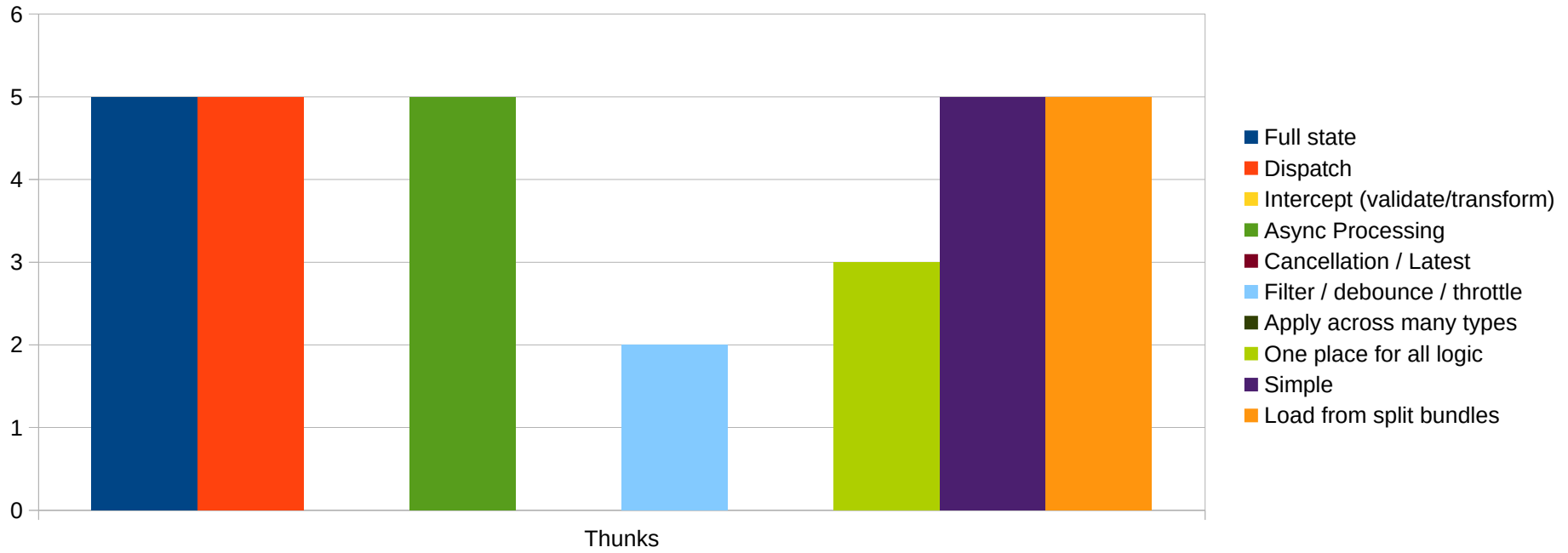
Thunks



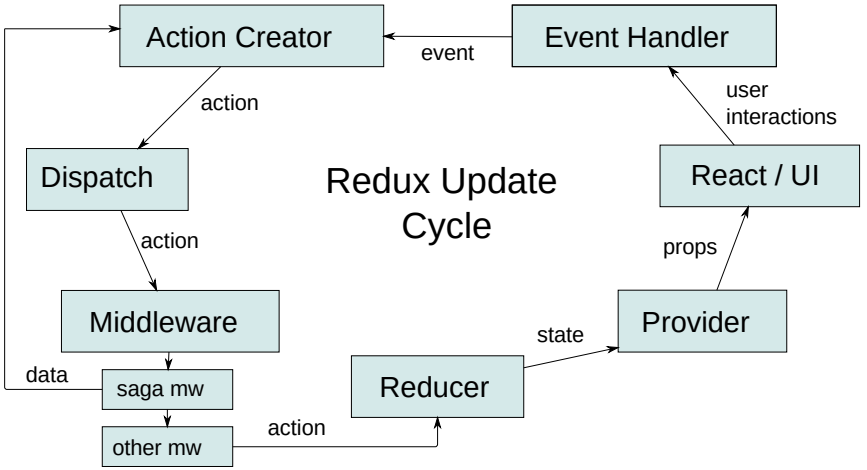

```
function buyClicked(ev, a, b) {
  return function(dispatch, getState) {
    // sync or async code here
    return dispatch({ type: BUY });
  };
}
```

Thunks 2


Action Creator
event
Event Handler
thunk/action
user
interactions
Dispatch
Redux Update
Cycle
React / UI
thunk/action
props
Middleware
state
Provider
data
thunk mw
Reducer
action
other mw

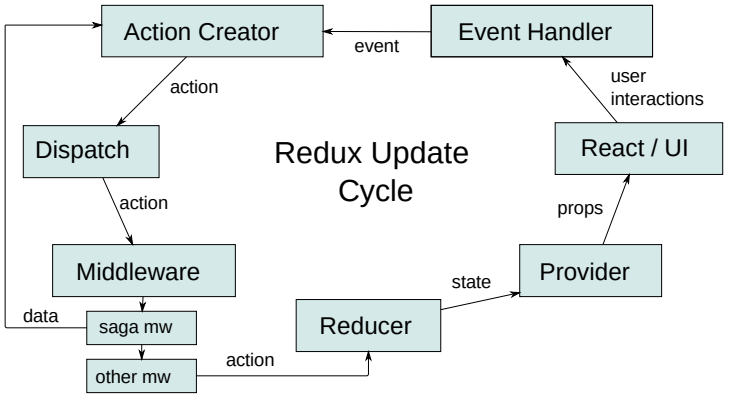

6
5
4
3
2
1
0
Thunks
Full state
Dispatch
Intercept (validate/transform)
Async Processing
Cancellation / Latest
Filter / debounce / throttle
Apply across many types
One place for all logic
Simple
Load from split bundles

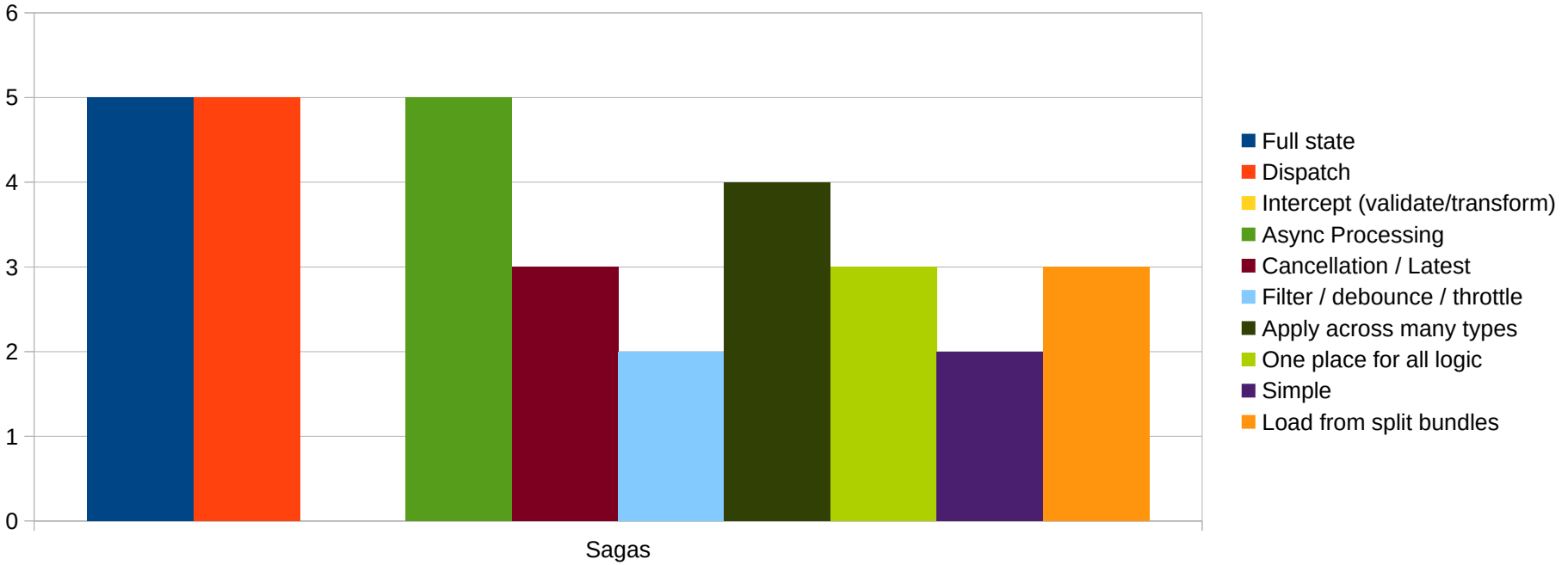
Sagas



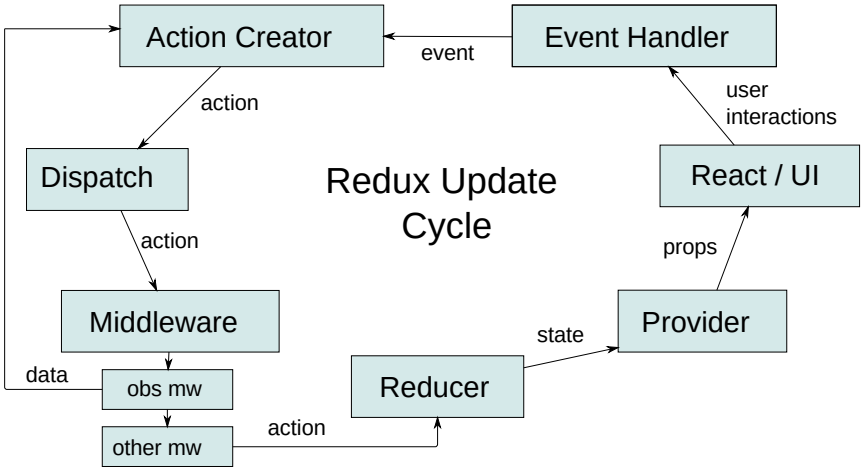

```
function* fetchUser(action) {
  try {
    const user = yield call(Api.fetchUser,
                            action.payload.userId);
    yield put({ type: "USER_FETCH_SUCCEEDED",
                user: user });
  } catch (e) {
    yield put({ type: "USER_FETCH_FAILED",
                message: e.message });
  }
}


function* mySaga() {
 yield* takeLatest("USER_FETCH_REQUESTED",
                   fetchUser);
}
```

Sagas 2


Action Creator
event
Event Handler
action
user
interactions
Dispatch
Redux Update
Cycle
React / UI
action
props
Middleware
state
Provider
data
saga mw
Reducer
action
other mw

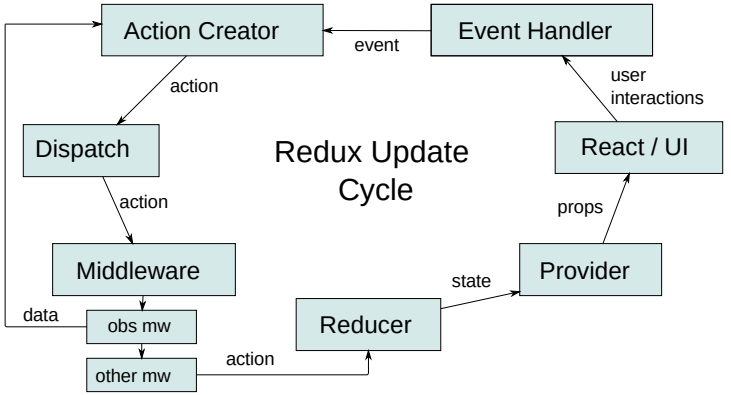

6
5
4
3
2
1
0
Sagas
Full state
Dispatch
Intercept (validate/transform)
Async Processing
Cancellation / Latest
Filter / debounce / throttle
Apply across many types
One place for all logic
Simple
Load from split bundles

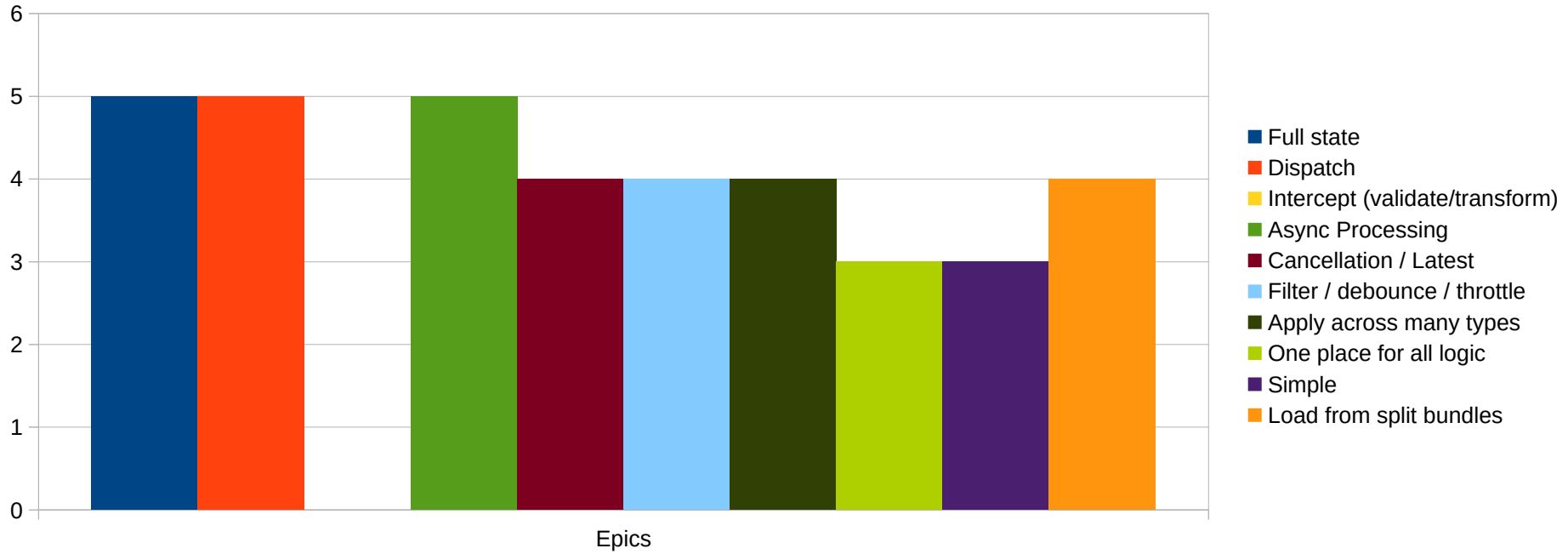
Epics – redux-observable



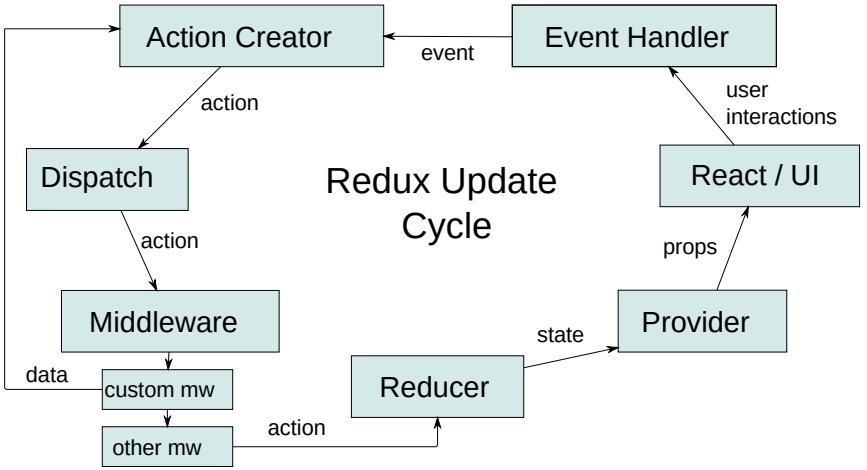

```
const fetchUserEpic = action$ => {
  const URL = `/api/users/${action.payload}`;
  return action$.ofType(FETCH_USER)
    .switchMap(action =>
      ajax.getJSON(URL)
        .map(fetchUserFulfilled)
        .takeUntil(
          action$.ofType(
            FETCH_USER_CANCELLED))
        .catch(err =>
          Observable.of({
            type: FETCH_ERROR,
            err })));
  );
};
```

Epics redux-observable 2



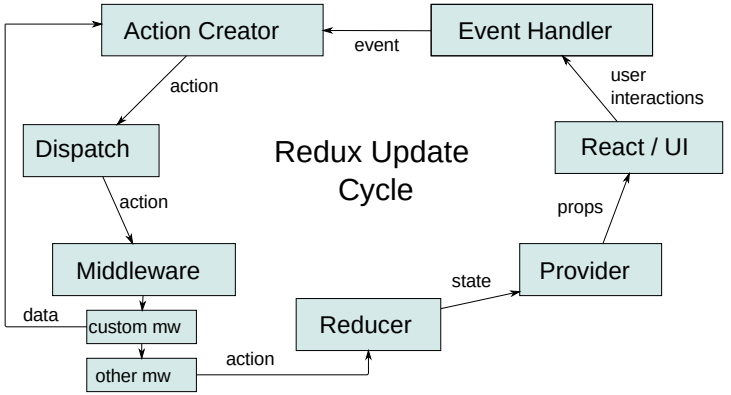


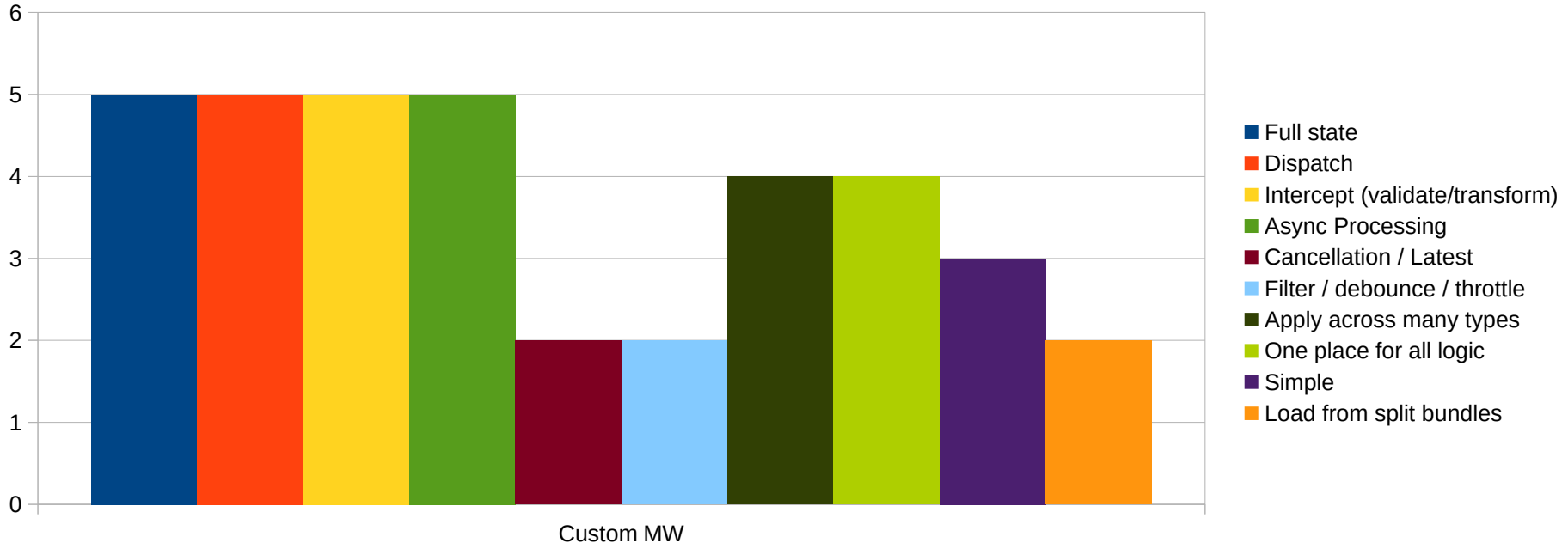
Custom Middleware




```
const logger = store => next => action => {
  console.log('dispatching', action)
  let result = next(action)
  console.log('next state', store.getState())
  return result
}

const fetchUser = store => next => action => {
  if (action.type !== 'FETCH_USER') {
    return next(action);
  }
  // could modify, silence, log action
  let result = next(action);
  const state = store.getState();
  const { dispatch } = store;
  return fetch(url)
    .then(response => response.json())
    .then(user => dispatch({ type: USER_SUCCESS, user }))
    .catch(err => {
      console.error(err); // might be render err
      dispatch({ type: USER_ERROR, err });
    });
}
```

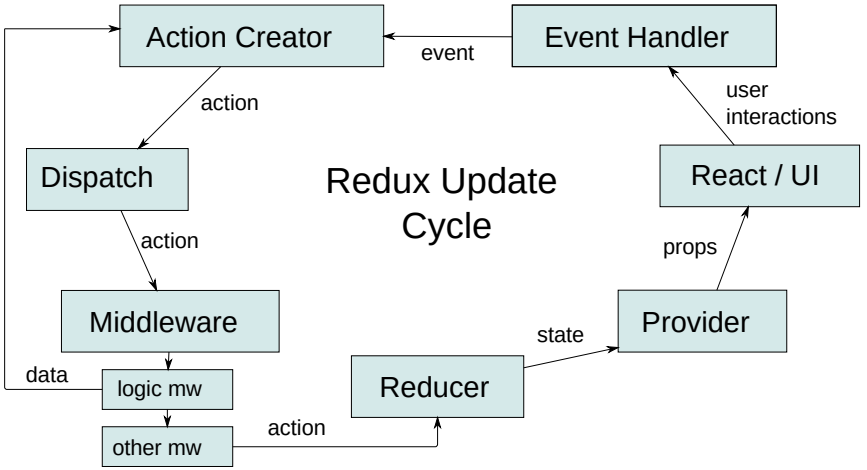
Custom Middleware 2





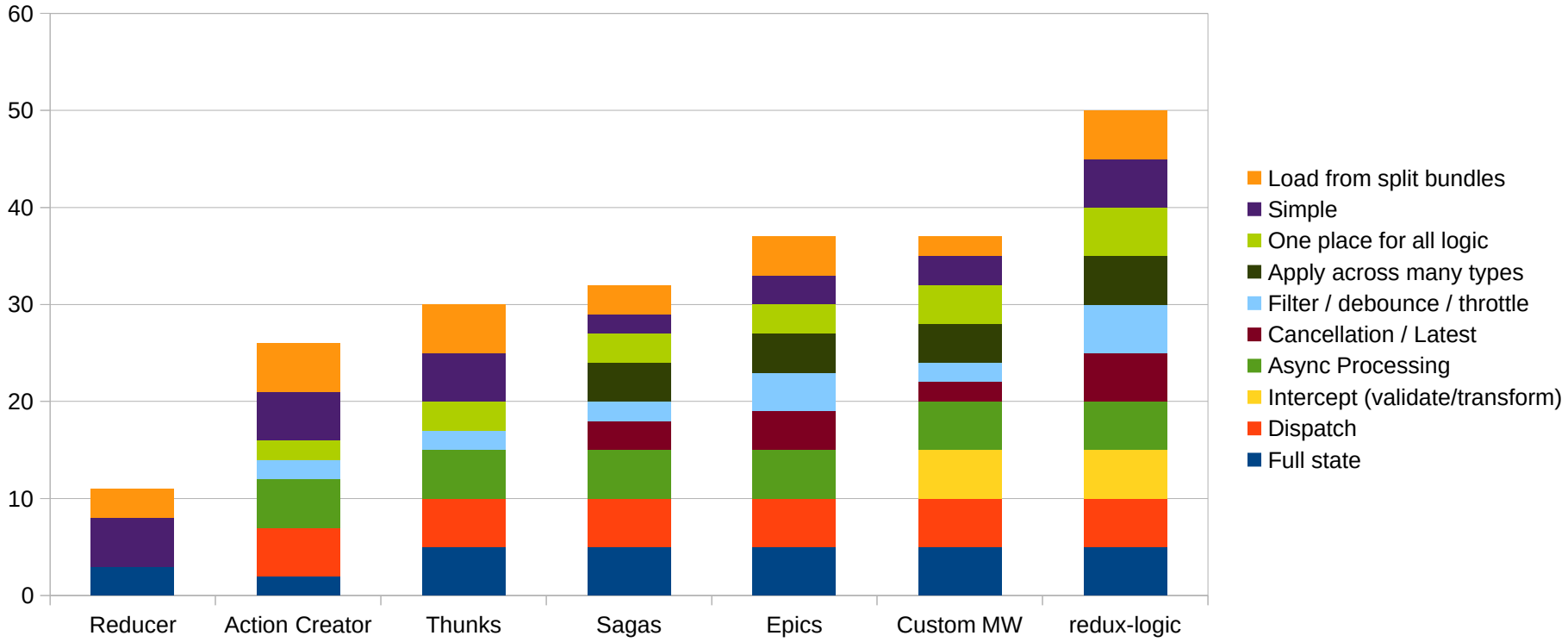
Now what?

redux-logic

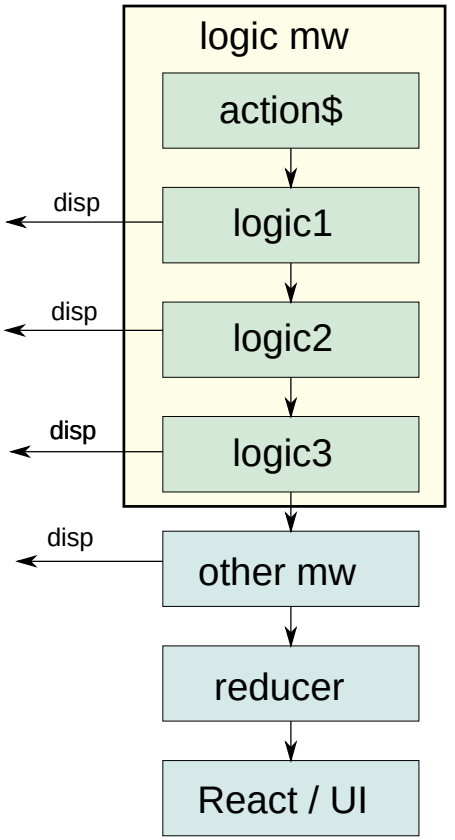



```
const logger = createLogic({
  type: '*',

  transform({ getState, action }, next) {
    console.log('dispatching', action)
    next(action);
    console.log('next state', getState())
  }
});

const fetchUser = createLogic({
  type: FETCH_POLLS,
  cancelType: FETCH_POLLS_CANCEL,
  latest: true,

  process({ getState, action }, dispatch) {
    axios.get('https://survey.codewinds.com/polls')
        .then(resp => resp.data.polls)
        .then(polls =>
          dispatch({ type: FETCH_POLLS_SUCCESS,
                     payload: polls }))
        .catch(err => {
          console.error(err); // could be render err
          dispatch({ type: FETCH_POLLS_FAILED,
                     payload: err,
                     error: true })
    });
  }
});
```

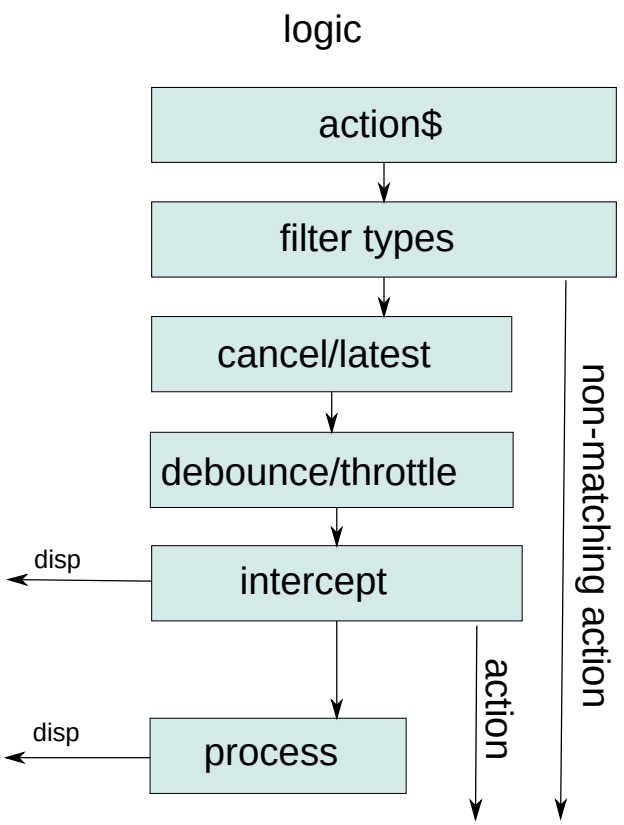
Unscientifc Results :-)



Logic stack


logic mw
action$
disp
logic1
disp
logic2
disp
logic3
disp
other mw
reducer
React / UI

Logic




```
const usersAddLogic = createLogic({
  type: USERS_ADD,
  validate({ getState, action }, allow, reject) {
    const fields = userSel.fields(getState());
    const errors = validateFields(fields);
    if (!errors.length) {
      allow(action); // no errors, let USERS_ADD go through
    } else { // still has errors
      // Errors should already be on screen so just reject silently
      reject();
    }
  },

  // if it passed the validation hook then this will be executed
  process({ httpClient, getState }, dispatch) {
    const state = getState();
    const fields = userSel.fields(state);
    httpClient.post('http://reqres.in/api/users', fields)
      .then(resp => resp.data) // new user created is returned
      .then(user => dispatch(usersAddSuccess(user)))
      .catch(err => {
        console.error(err); // might be a render err
        dispatch(usersAddFailed(err))
      });
  }
});
```

createLogic

```
const fooLogic = createLogic({
  type: T, // req string, regex, array of str/regex, '*' for all
  cancelType: CT, // string, regex, array of str or REs

  // limiting - optionally define any of these
  debounce: 0, // debounce for N ms, default 0
  throttle: 0, // throttle for N ms, default 0
  latest: true, // only take latest, default false

  // Put your business logic into one or more of these
  // execution hooks: validate, transform, process
  validate({ getState, action }, allow, reject) {
    // If reject is used,process hook will not be executed
    allow(action); // OR reject(action)
  }),

  transform({ getState, action }, next /*, reject */) {
    next(action);
  }),


  // options influencing process hook, defaults to {}
  processOptions: { ... }

  process({ getState, action }, ?dispatch, ?done) {
    dispatch(myNewAction); // in single dispatch mode this also completes
    done(); // performing multiple dispatches
  })
});
```

createLogicMiddleware

```
const logicMiddleware = createLogicMiddleware(
  arrLogic, // array of logic items, no duplicate refs to same logic
  deps   // optional injected deps/config, supplied to logic
);

logicMiddleware.addLogic(arrNewLogic); // append logic
logicMiddleware.mergeNewLogic(arrMergeLogic); // only merge new
logicMiddleware.replaceLogic(arrReplacementLogic); // replace all logic

// for server side use and testing
logicMiddleware.whenComplete(fn); // returns promise

// observable for monitoring the internal operations
logicMiddleware.monitor$
```

Demos / Code Review

Links to Demo examples

- Main usage example
- Notifications
- Async/Await and processOptions
- Search with cancellation and debouncing
- JSFiddle live examples
- Full examples

Lessons Learned

- Finding the right abstraction is hard and takes time
  - Try to help devs to fall into the pit of success
  - Ongoing evolution
- mw.monitor$ - exposing a way to monitor the internals was extremely helpful
  - Testing much simpler and more complete
  - Enabled easier server-side render mw.whenComplete(fn)
- RxJS is a great tool for dealing with events over time but it takes a while to master

redux-logic v0.10

- Process arity sets some default processOptions
  - process({ getState, action }) {
    return objOrPromiseOrObs;
    }
  - process({ getState, action }, dispatch) {
    dispatch(objOrPromiseOrObs); // single dispatch
    }
  - process({ getState, action }, dispatch, done) {
    dispatch(objOrPromiseOrObs); // any number of dispatches
    dispatch(objOrPromiseOrObs);
    done(); // when finished call done
    }

redux-logic v0.10

- **logicMW.mergeNewLogic(arrLogic)** – for split bundle loading, only adds logic that is new, ignores refs to existing logic
- **logicMW.monitor$** - observable to monitor the internals of what is happening inside logicMW
- **logicMW.whenComplete(fn)** // run fn if provided and return promise when all in-flight action processing has completed. Great for testing and server-side rendering.

Learn more

- Video / Slides / Notes – https://codewinds.com/nc2016
- Subscribe to my newsletter to learn more about redux-logic, rxjs, react, functional js, and related topics – https://codewinds.com
- Questions, discussion, or to hire me
  - jeff@codewinds.com
  - @jeffbski