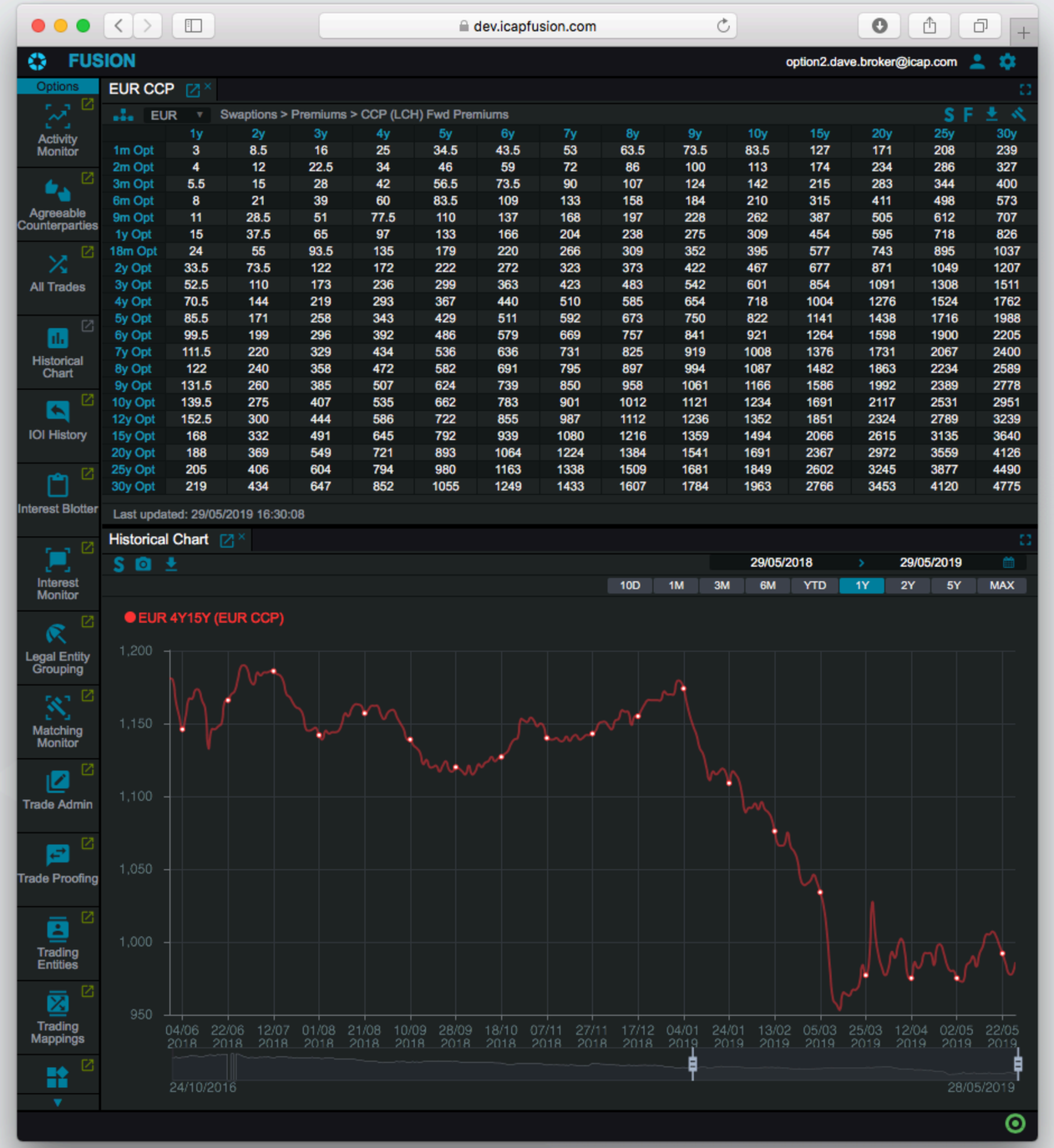
REDUX PERFORMANCE

DAVID J. BRADSHAW

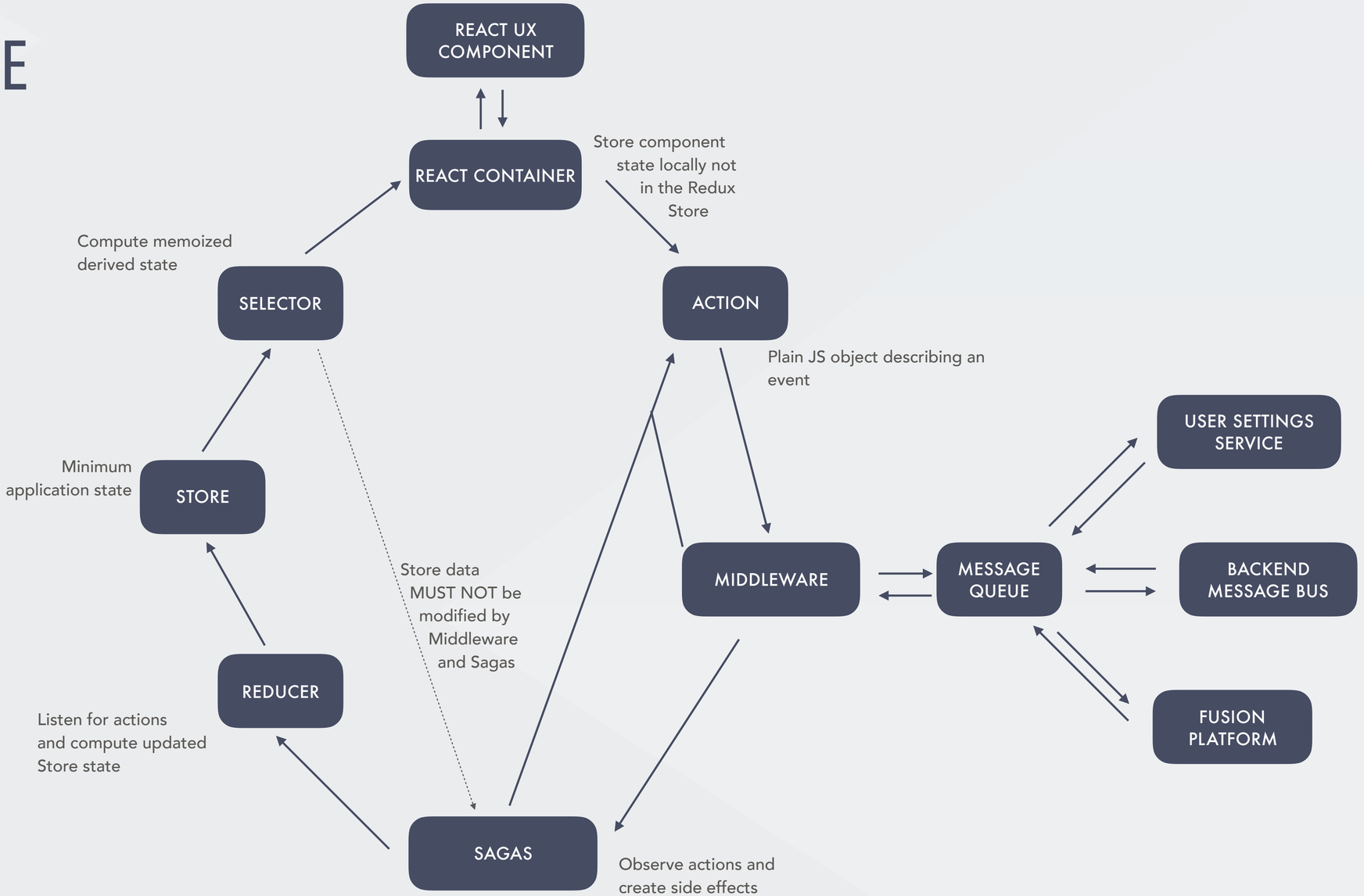
Principle FE Developer - ICAP Fusion Platform

- dave@bradshaw.net
- github.com/davidjbradshaw
- @davidjbrashaw

FUSION OPTIONS



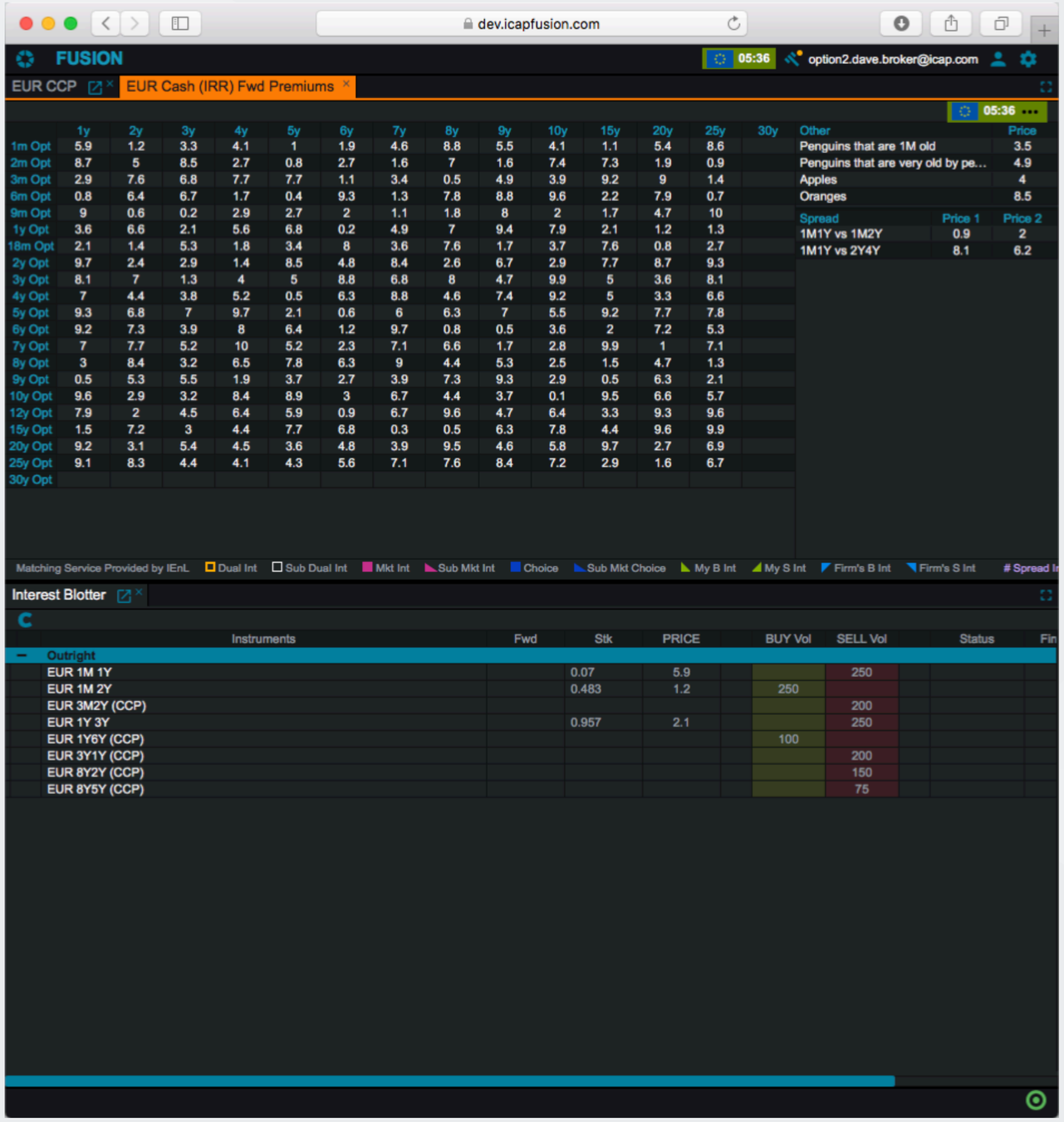
FUSION OPTIONS ARCHITECTURE



SUBSCRIBE

```
{
  TYPE: HEARTBEAT,
  META: {
    ID: 'HEARTBEAT',
    SUBSCRIBE: {
      TPVOL: 'LDN/PS/52/P/GLOBAL/RFQ/E/HB',
    }
  }
}
```

```
{
  type: 'mq/MSG/tpvol/LDN/PS/52/P/GLOBAL/RFQ/E/HB',
  payload: {
    hostName: 'ldnmtpvgwy01',
    interval: 5000,
  },
  meta: {
    batch: true,
    eventStream: false,
  }
}
```



1,000 actions per second

Middlewares

Sagas

Reducers

React

ReactDOM

Browser Paint

MIDDLEWARE

To the rescue

RATE LIMITING

- Redux Throttle

```
const action = {
  type: 'MY_ACTION',
  meta: {
    throttle: {
      wait: 300,
      leading: false
    }
  }
}
```

RATE LIMITING

- Redux Throttle
- Redux Debounced

```
const action = {
  type: 'MY_ACTION',
  meta: {
    debounce: {
      time: 300,
      leading: true,
      trailing: false
    }
  }
}
```

BATCHING

- redux-batched-actions

```
store.dispatch(
  batchActions([
    { type: 'ACTION1' },
    { type: 'ACTION2' },
    { type: 'ACTION3' },
  ])
)
```

BATCHING

- redux-batched-actions
- @manaflair/redux-batch

```
store.dispatch([
  { type: 'ACTION1' },
  { type: 'ACTION2' },
  { type: 'ACTION3' },
])
```

BATCHING ACTIONS

- redux-batched-actions
- @manaflair/redux-batch
- react-redux 7

```
import { batch } from "react-redux"

const { dispatch } = store

batch(() => {
    dispatch({ type: 'ACTION1' })
    dispatch({ type: 'ACTION2' })
    dispatch({ type: 'ACTION3' })
})
```

Dan Abramov
@dan_abramov

If your actions creator names start with set* and you often call multiple in a row, you might be missing the point of using Redux.

♡ 45 1:10 PM - Nov 20, 2016

BATCHING ACTIONS

- redux-batched-actions
- @manaflair/redux-batch
- react-redux 7
- redux-batched-subscribe

```
const debounceNotify =
  debounce(notify => notify())



const store = createStore(
  reducer,
  intialState,
  batchedSubscribe(debounceNotify)
)
```

REDUX-BATCHED-SUBSCRIBED

Async Action Batching

```
import raf from 'raf'
import { batchedSubscribe } from 'redux-batched-subscribe'

let notify = null
let rafId = null

const shouldBatch = action => action?.meta?.batch

export const batchedSubscribeEnhancer = batchedSubscribe(freshNotify => (notify = freshNotify))

export const batchedSubscribeMiddleware = () => next => action => {
  const resolved = next(action)

  if (notify && rafId === null && !shouldBatch(action)) {
    notify()
  } else if (!rafId) {
    rafId = raf(() => {
      rafId = null
      notify()
    })
  }

  return resolved
}
```

ACTION PIPELINE

Middlewares

Sagas

Reducers

React

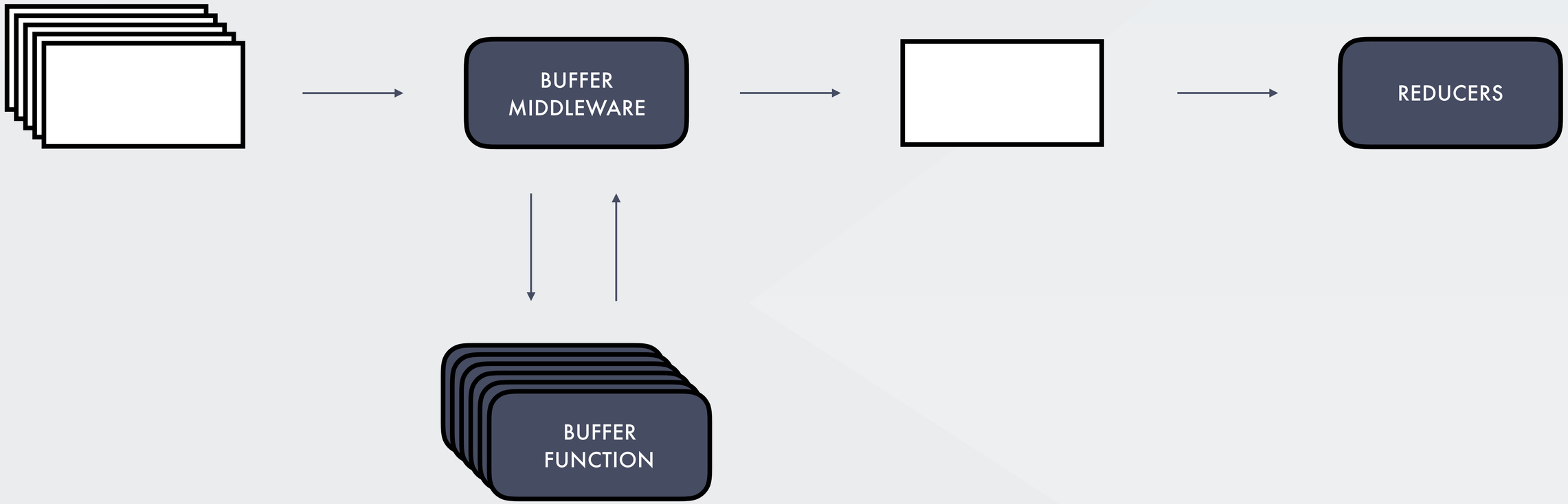
ReactDOM

Browser Paint

BUFFERING

Can we turn multiple asynchronous actions of the same type into a single action?

BUFFERING



ACTION PIPELINE

Middlewares

Sagas

Reducers

React

ReactDOM

Browser Paint

REJECT MIDDLEWARE

```
function reject(action) {
  console.warn('Dropped Action:', action.type)
  return null
}

export default test => () => next => action =>
  test(action) ? reject(action) : next(action)
```

```
rejectMiddleware(action => ignoredActionTypes.includes(action.type))
```

NOPE

A NOPE NINJA DELIVERS A WICKED DRAGON KICK

{ STOP THE ACTION OF ANOTHER PLAYER.
YOU CAN PLAY THIS AT ANY TIME. }

FIRST EDITION

NOPE

WEBWORKERS

Not so fast

REDUCERS

Good store structure is key to performance

Reducers

Data Model

- Avoid modelling state after server API
- Avoid modelling state after what views like to consume

Reducers

Data Model

- Never hold duplicate data in the app state
- Never store derived data in the state

Reducers

Data Model

- Normalize nested objects
- Prefer Maps to Arrays

SERVER API


```
{
  "total": 117,
  "offset": 0,
  "products": [
    {
      "id": "88cd7621-d3e1-42b7-b2b8-8ca82cdac2f0",
      "title": "Blue Shirt",
      "price": 9.99
    },
    {
      "id": "aec17a8e-4793-4687-9be4-02a6cf305590",
      "title": "Red Hat",
      "price": 7.99
    }
  ]
}
```

MAPPED DATA


```
{
  "productsById": {
    "88cd7621-d3e1-42b7-b2b8-8ca82cdac2f0": {
      "title": "Blue Shirt",
      "price": 9.99
    },
    "aec17a8e-4793-4687-9be4-02a6cf305590": {
      "title": "Red Hat",
      "price": 7.99
    }
  }
}
```

MAPPED DATA


```
{
  "productsById": {
    "88cd7621-d3e1-42b7-b2b8-8ca82cdac2f0": {
      "title": "Blue Shirt",
      "price": 9.99
    },
    "aec17a8e-4793-4687-9be4-02a6cf305590": {
      "title": "Red Hat",
      "price": 7.99
    }
  },
  "productIds": [
    "88cd7621-d3e1-42b7-b2b8-8ca82cdac2f0",
    "aec17a8e-4793-4687-9be4-02a6cf305590"
  ]
}
```

redux-ignore

Higher-order reducer to ignore redux actions

```
import { ignoreActions, filterActions } from 'redux-ignore'

ignoreActions(reducer, [ARRAY_OF_ACTIONS])
ignoreActions(reducer, (action) => !action.valid)

filterActions(reducer, [ARRAY_OF_ACTIONS])
filterActions(reducer, (action) => action.valid)
```

redux-log-slow-reducers

Logs a warning if any reducer takes over 8ms

```
import { combineReducers } from 'redux'
import { pipe } from 'ramda'
import logSlowReducers from 'redux-log-slow-reducers'

export default (process.env.NODE_ENV === 'production'
  ? combineReducers
  : pipe(
      logSlowReducers,
      combineReducers
    ))
```

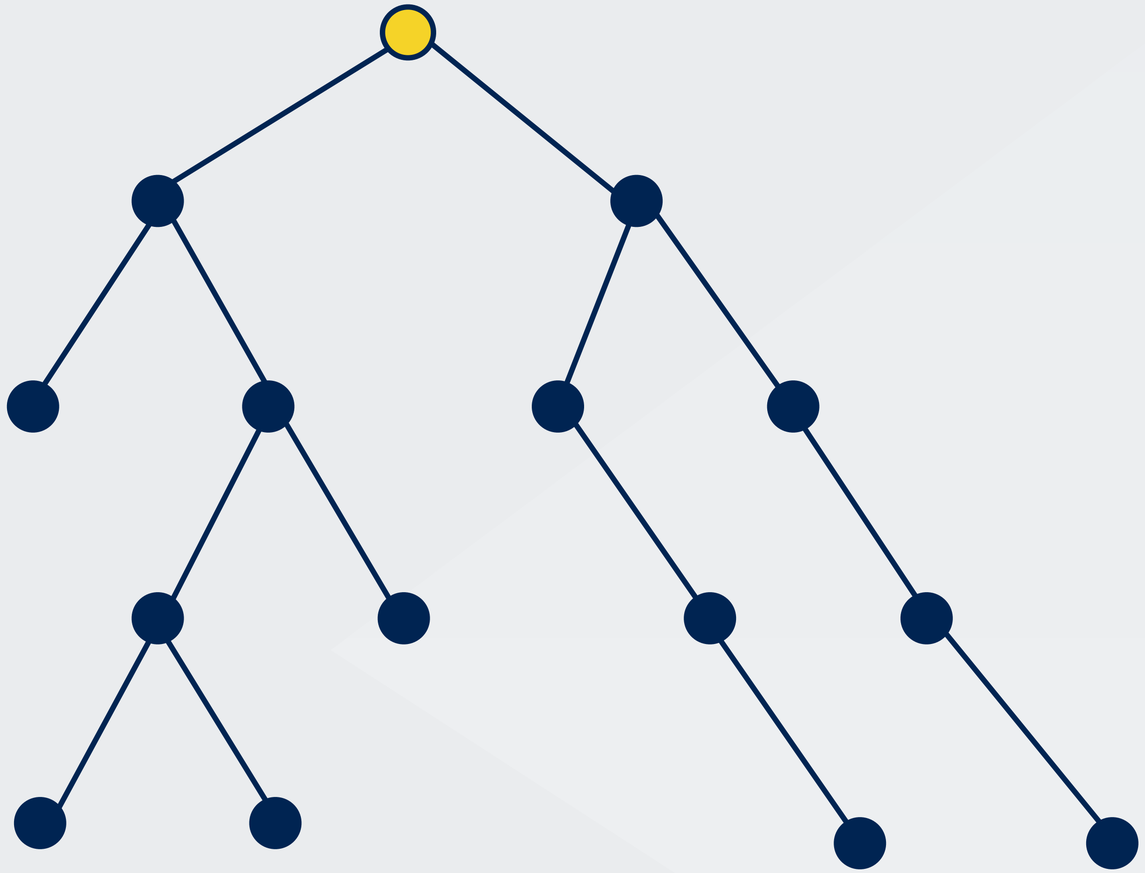
REACT-REDUX

Help it help you

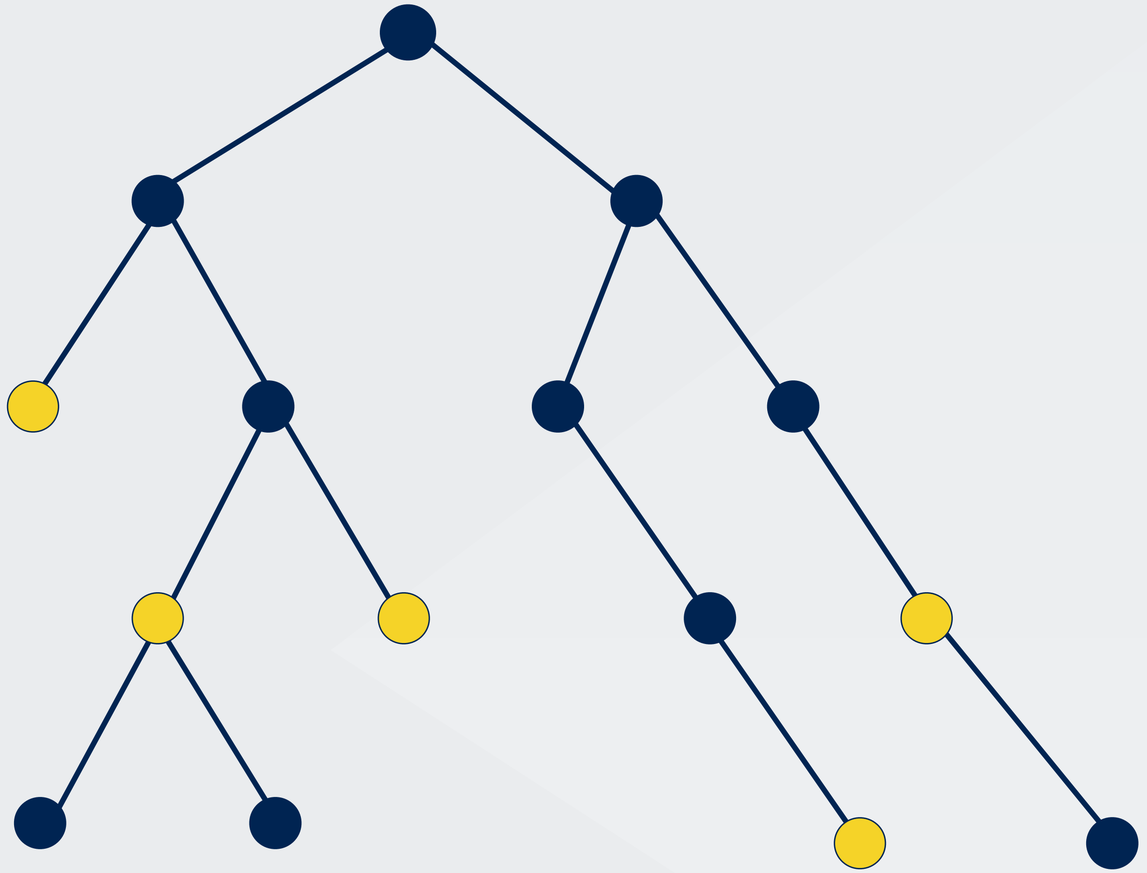
CONNECT()

Implements many performance optimizations internally, so that your own component only re-renders when it actually needs to

CONNECT()

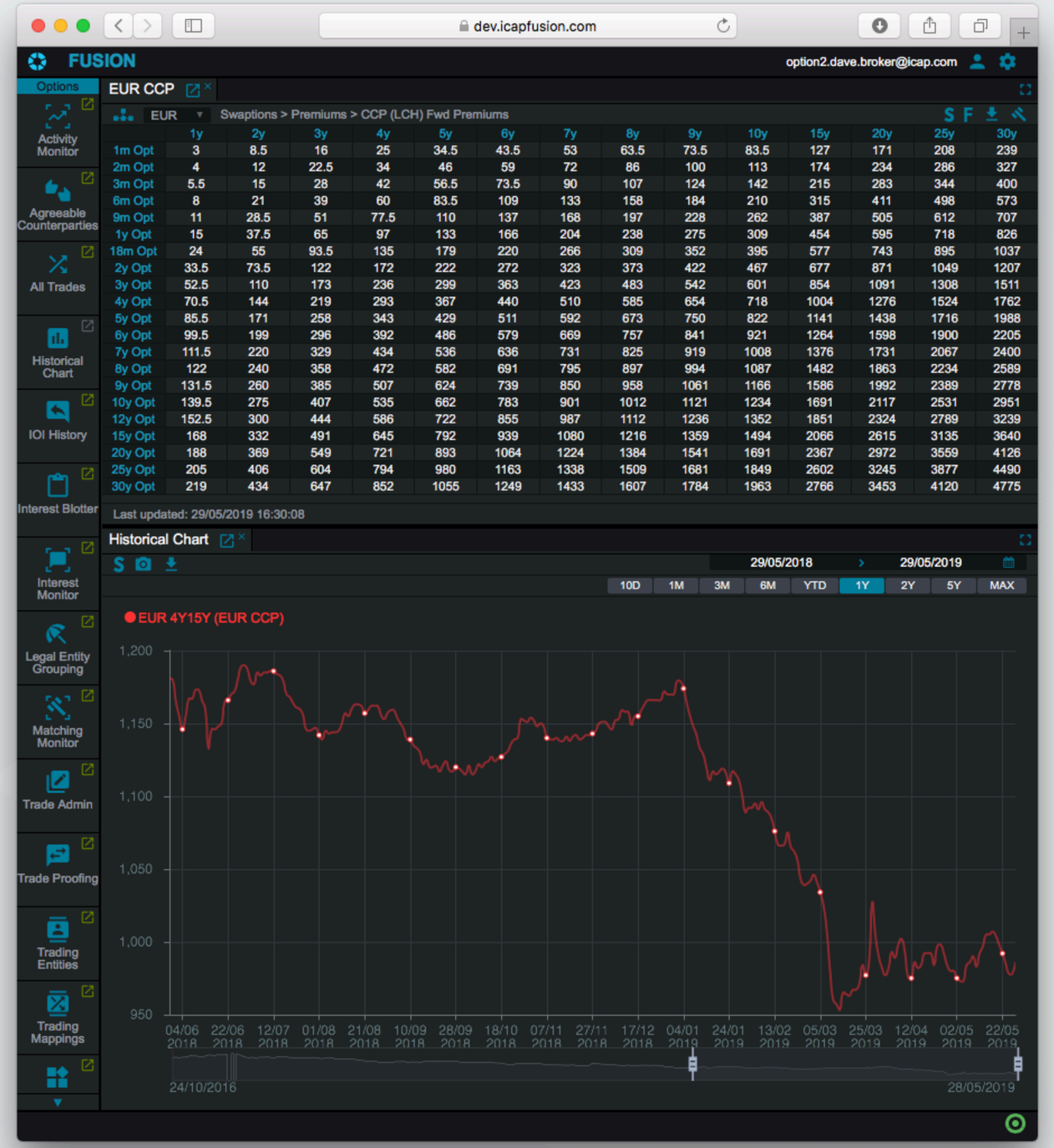


CONNECT()



FUSION OPTIONS

Every cell on the grid has it's own instance of connect



CONNECT()

```
connect(mapStateToProps, mapDispatchToProps)(Component)
```

CONNECT()

```
connect(mapStateToProps, mapDispatchToProps, mergeProps, options)(Component)
```

Options

```
// spec
{
  context?: Object,
  pure?: boolean,
  areStatesEqual?: Function,
  areOwnPropsEqual?: Function,
  areStatePropsEqual?: Function,
  areMergedPropsEqual?: Function,
  forwardRef?: boolean,
}
```

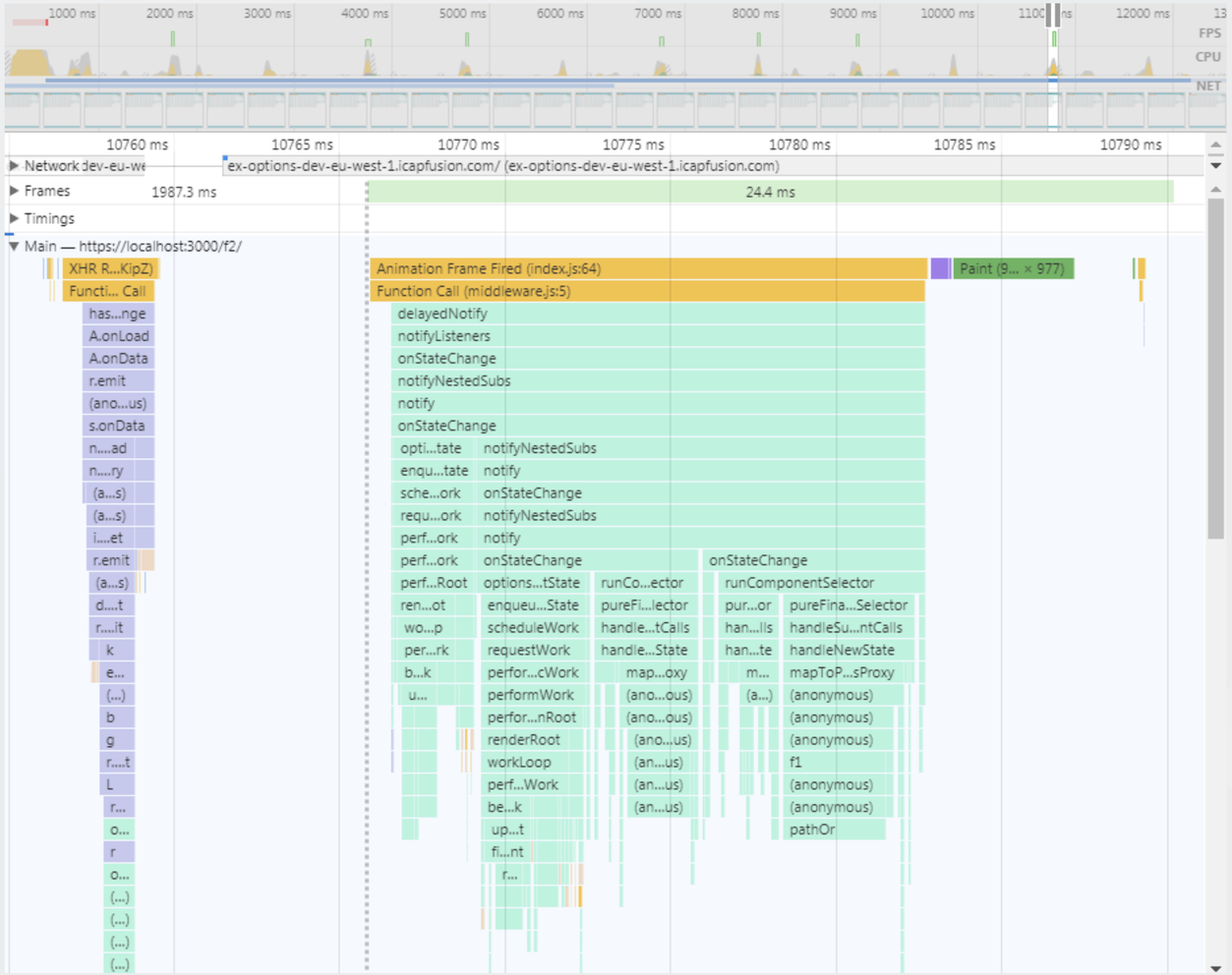
Options

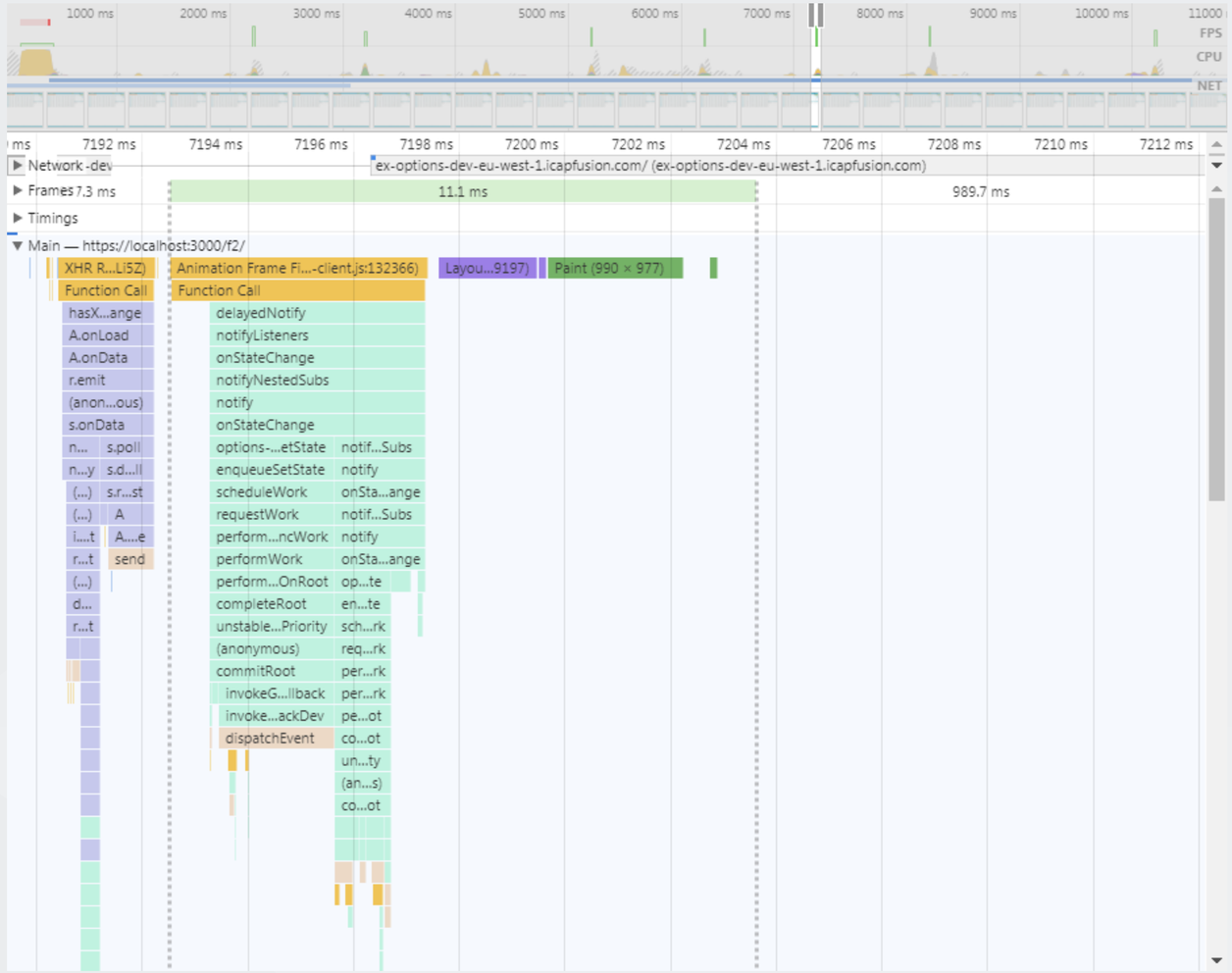
```
// Default values
{
  context: undefined,
  pure: true,
  areStatesEqual: (next, prev) => next === prev,
  areOwnPropsEqual: shallowEqual,
  areStatePropsEqual: shallowEqual,
  areMergedPropsEqual: shallowEqual,
  forwardRef: false,
}
```

Options

```
// Faster values
{
  areStatesEqual: (next, prev) => next.grid === prev.grid,
  areOwnPropsEqual: () => true,
  areMergedPropsEqual: () => true,
}
```

Options





24.4 ms

11.1 ms

mergeProps

The forgotten parameter

mergeProps

```
function defaultMergeProps(stateProps, dispatchProps, ownProps) {
  return {
    ...ownProps,
    ...stateProps,
    ...dispatchProps
  }
}
```

mergeProps

```
import * as actionCreators from './actionCreators'

function mapStateToProps(state) {
  return { todos: state.todos }
}

function mergeProps(stateProps, dispatchProps, ownProps) {
  return Object.assign({}, ownProps, {
    todos: stateProps.todos[ownProps.userId],
    addTodo: (text) => dispatchProps.addTodo(ownProps.userId, text)
  })
}

export default connect(mapStateToProps, actionCreators, mergeProps)(TodoApp)
```

```
() => null !== () => null
```

mergeProps

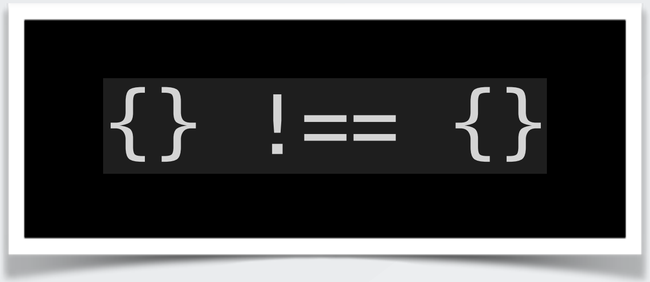
```
import * as actionCreators from './actionCreators'

function mapStateToProps(state) {
  return { todos: state.todos }
}

function mergeProps(stateProps, dispatchProps, ownProps) {
  return Object.assign({}, ownProps, {
    todos: stateProps.todos[ownProps.userId],
    addTodo: (text) => dispatchProps.addTodo(ownProps.userId, text)
  })
}

export default connect(mapStateToProps, actionCreators, mergeProps)(TodoApp)
```

mergeProps

```
import { identity, memorizeWith } from 'ramda'
import * as actionCreators from './actionCreators'

const memorize = memorizeWith(identity)
const addTodo = memorize(
  (userId, dispatchProps) => (text) => dispatchProps.addTodo(userId, text)
)

function mapStateToProps(state) {
  return { todos: state.todos }
}

function mergeProps(stateProps, dispatchProps, ownProps) {
  return Object.assign({}, ownProps, {
    todos: stateProps.todos[ownProps.userId],
    addTodo: addTodo(ownProps.userId, dispatchProps)
  })
}

export default connect(mapStateToProps, actionCreators, mergeProps)(TodoApp)
```

mapStateToProps

```
mapStateToProps = state => ({
  coords: {
    x: state.map.latitude,
    y: state.map.longitude
  }
})
```



Reselect

mapStateToProps

- (state) => { ... }
- (state, ownProps) => { ... }
- () => (state) => { ... }
- () => (state, ownProps) => { ... }

CONCLUSION

- Normalize Data
- Memorize Selectors
- Optimize Rendering

"There is no doubt that the grail of efficiency leads to abuse. Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: ***Premature optimisation is the root of all evil****. Yet we should not pass up our opportunities in that critical 3%."*

Donald Knuth - Structured Programming with Goto statements (1974)