BUSINESS LOGIC IN REDUX APPLICATIONS

Patrick Oladimeji

OVERVIEW

- Quick overview of the core parts of a redux app
- Domain logic and where it sits in the application
- Lessons learned from some experiments

CORE REDUX

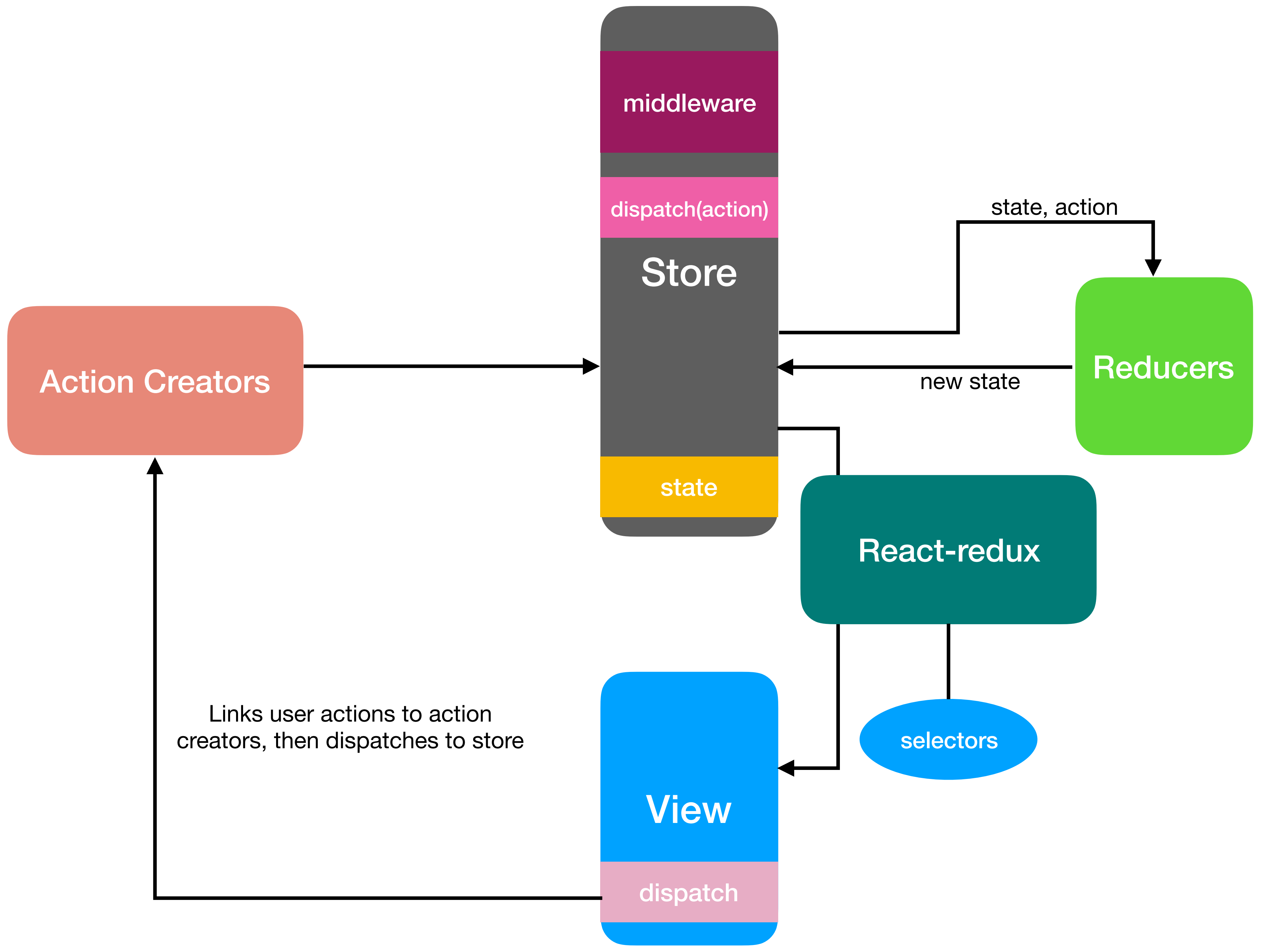
- Single serialisable store
- Actions and action creators
- Reducers
- Middleware

REDUX ADD-ONS

- Selectors (e.g., reselect)
- Middleware Implementations
  - redux-saga, redux-thunk, redux-observable

REACT-REDUX

- Binds redux to react components
- Exposes **connect** a HOC for enhancing the view
- Enables mapping between state and view props
- Allows injection of functions that dispatch actions into view


middleware
dispatch(action)
Store
state
state, action
Reducers
new state
Action Creators
React-redux
selectors
View
dispatch
Links user actions to action creators, then dispatches to store

HOW ABOUT THE DOMAIN LOGIC

Where do we put that?
Who should be aware of it

"Business logic is the portion of an enterprise system which determines how data is transformed or calculated, and how it is routed to people or software (workflow)."

–Anonymous Wikipedian

https://en.wikipedia.org/wiki/Business_logic

EXAMPLES OF BUSINESS LOGIC

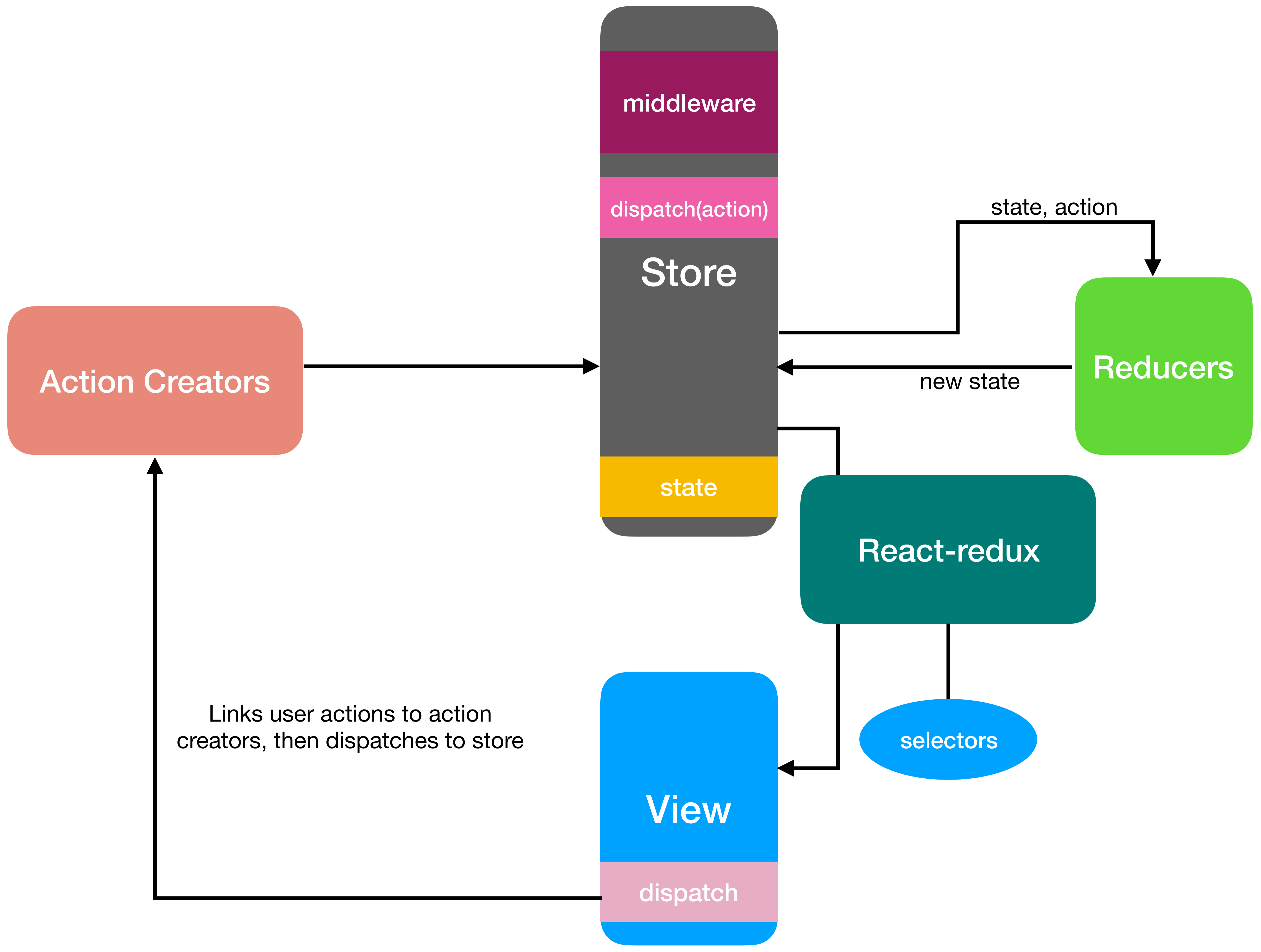
- Calculation routines
- Validation routines
- Authorization/authentication routines

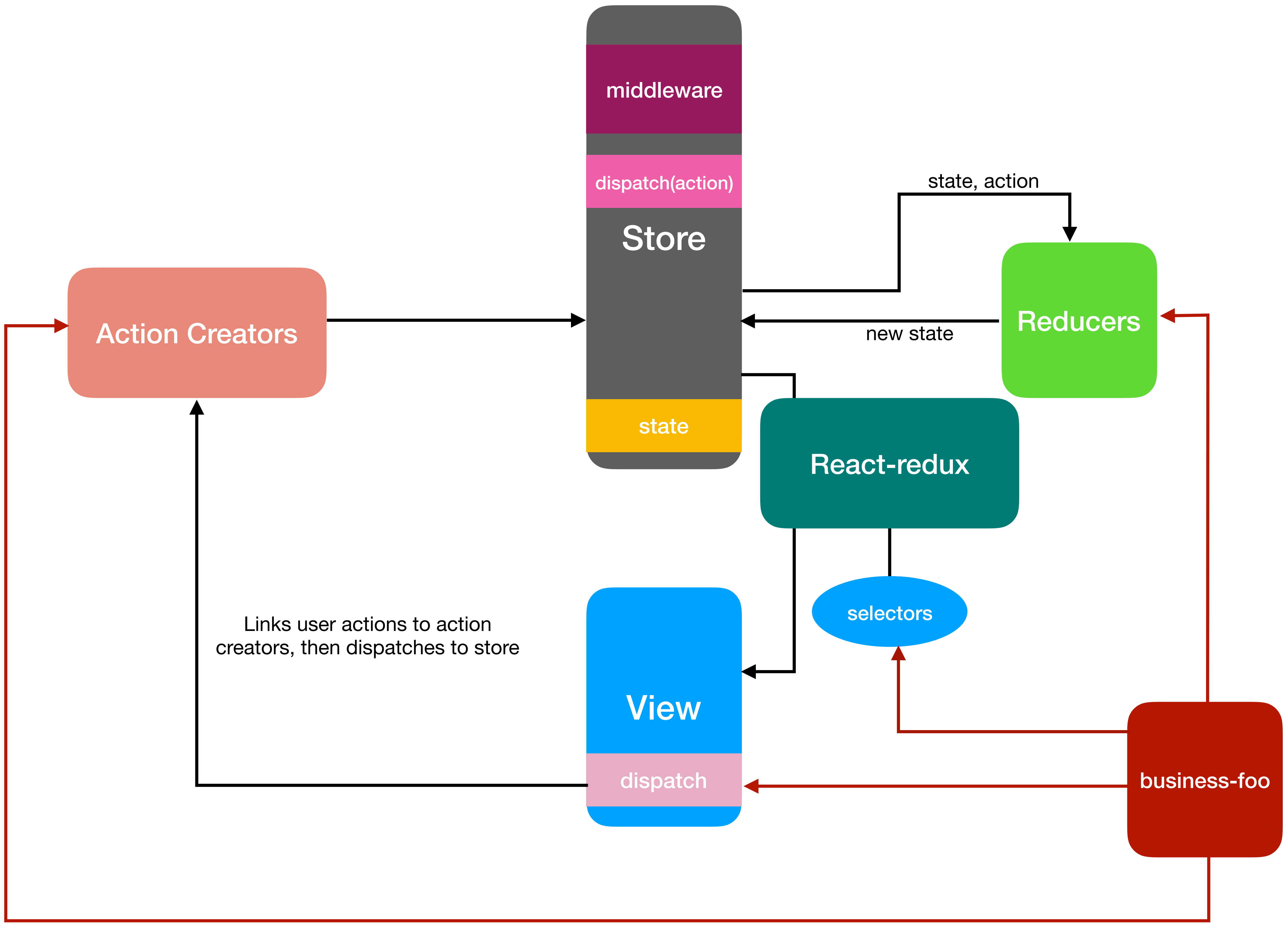
THE WHERE IS EASY

- Create a folder (call it business-foo)
- Structure it by business objects
    - Expose functions that map to operations over the objects
- Think standalone library that serves multiple application types

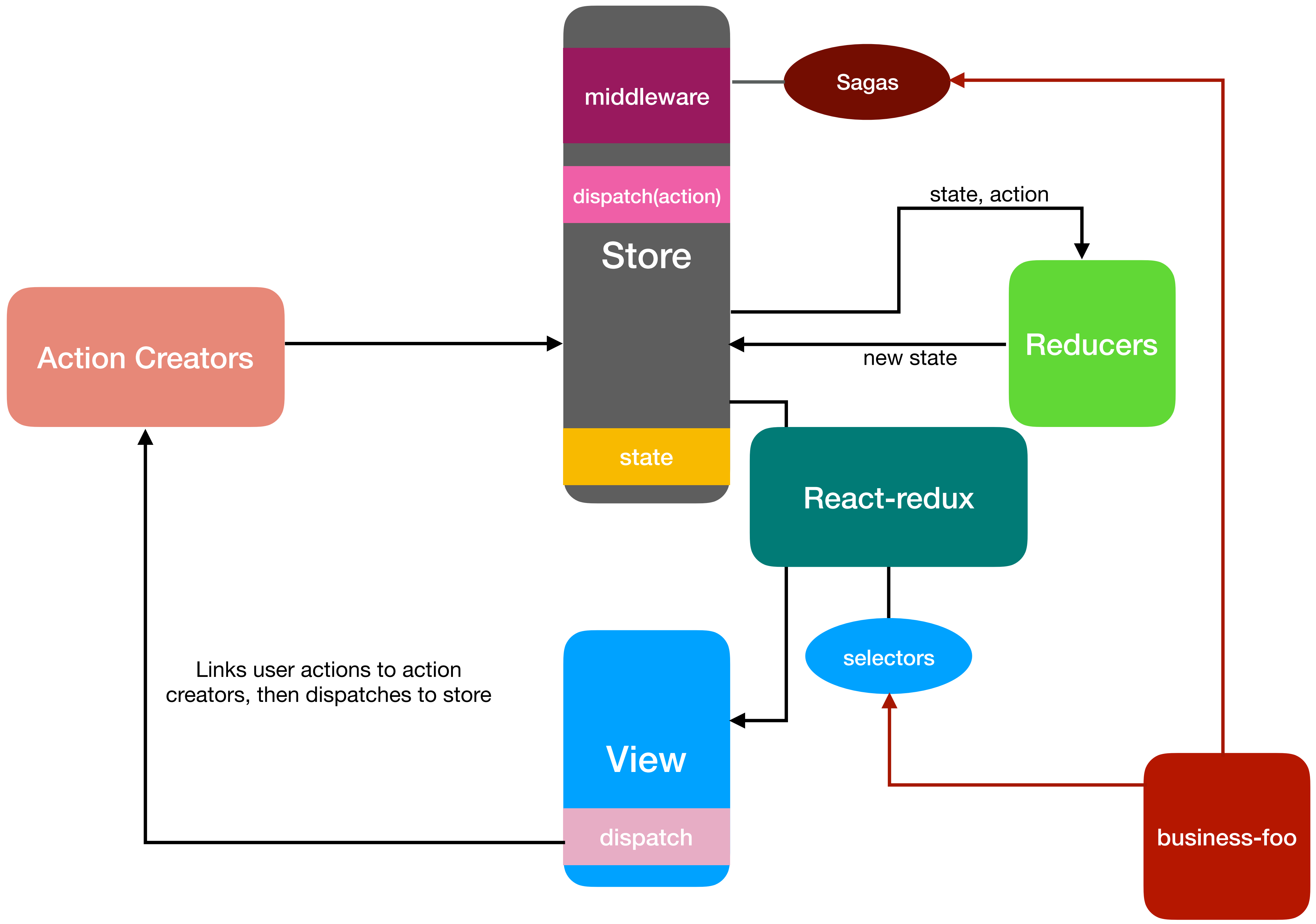
WHO SHOULD KNOW ABOUT BUSINESS LOGIC?

Slightly harder question


middleware
dispatch(action)
Store
state
state, action
Reducers
new state
Action Creators
React-redux
selectors
View
dispatch
Links user actions to action creators, then dispatches to store


middleware
dispatch(action)
Store
state
state, action
Reducers
new state
Action Creators
React-redux
selectors
Links user actions to action creators, then dispatches to store
View
dispatch
business-foo

ANYONE



SAGA OR SELECTOR

ANY OPINIONS?

PROS FOR SELECTORS

- Data in state can be minimal
    - No need to add validation or derived values into state
- Less code since there are less actions and state data
- Selector functions are lazily evaluated for the intended view

CONS FOR SELECTORS

- Might be difficult to track dependencies of selectors

```
import { createSelector } from 'reselect'

const shopItemsSelector = state => state.shop.items
const taxPercentSelector = state => state.shop.taxPercent

const subtotalSelector = createSelector(
  shopItemsSelector,
  items => items.reduce((acc, item) => acc + item.value, 0)
)

const taxSelector = createSelector(
  subtotalSelector,
  taxPercentSelector,
  (subtotal, taxPercent) => subtotal * (taxPercent / 100)
)

export const totalSelector = createSelector(
  subtotalSelector,
  taxSelector,
  (subtotal, tax) => ({ total: subtotal + tax })
)
```

- Difficult to handle async business logic

PROS FOR SAGAS

- Easy to handle async business logic
- Selector functions can be simpler and easily tested
- No ambiguity about where to invoke domain logic
- Dependencies between actions and effects more explicit

CONS FOR SAGAS

- *More action creators and state data*
- Sagas are difficult to typecheck
- Saga unit tests are tightly coupled to implementation

* Repeated predictable work is better than ambiguity in a team environment.
The former is an opportunity for tooling. The latter is a recipe for chaos. *

–Personal Opinion

CONCLUSIONS

- Make your domain logic a self-contained library
- Allow only your middleware to be aware of it

THANKS FOR LISTENING!

Questions?

dodopat

thehogfather