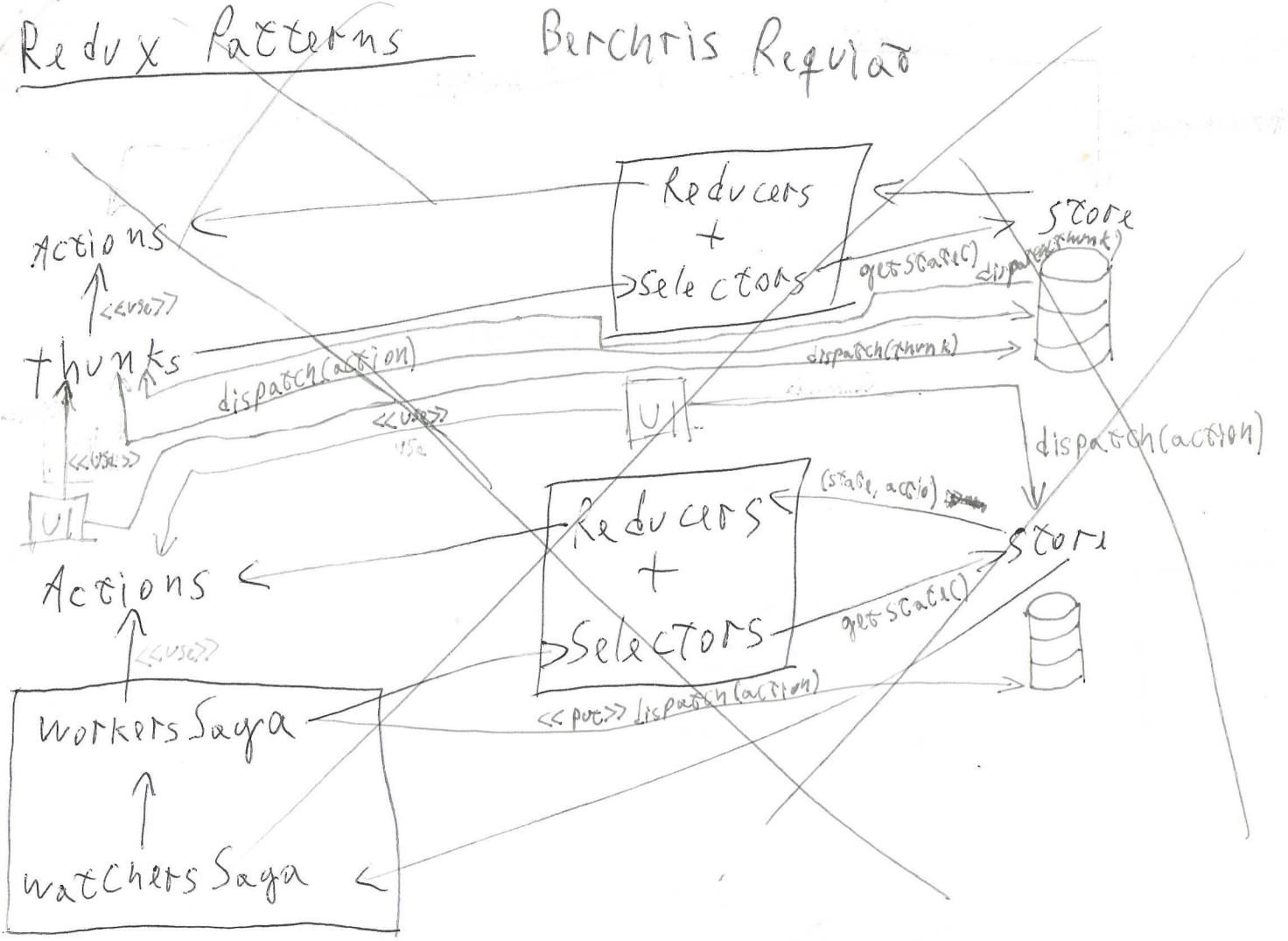
Redux Patterns — Berchris Requiao



Command Query Separation (CQS):

* Every method should either be a **command** that performs an action, or a **query** that returns data to the caller, but **not both**.
(-Meyer, Bertrand.)

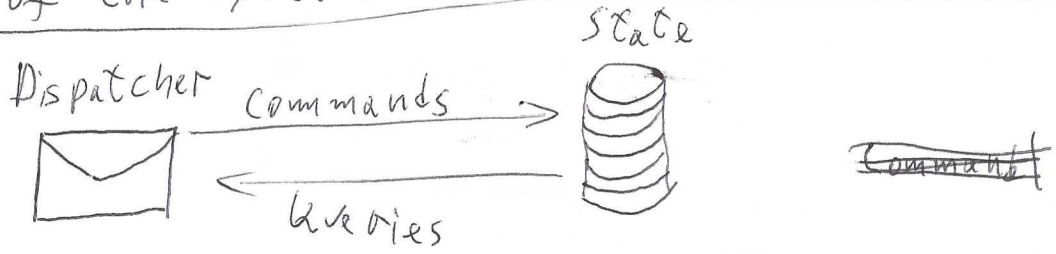
* Asking a question should not change the answer (Redux selectors)
(Meyer, Bertrand)

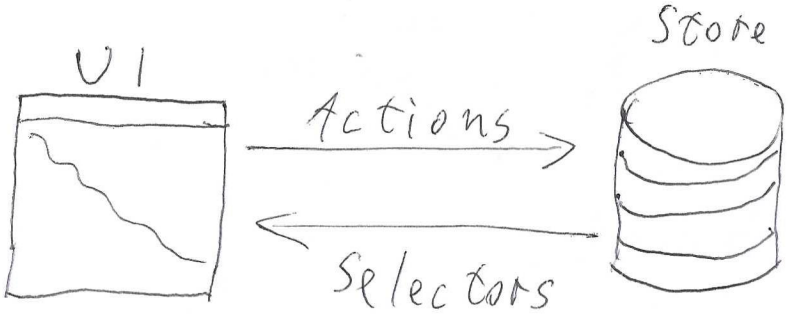
Commands: ~~(Redux selectors) (dispatch~~ (Modifiers)

Change the state of a system, but do not return a value.
(Martin Fowler)

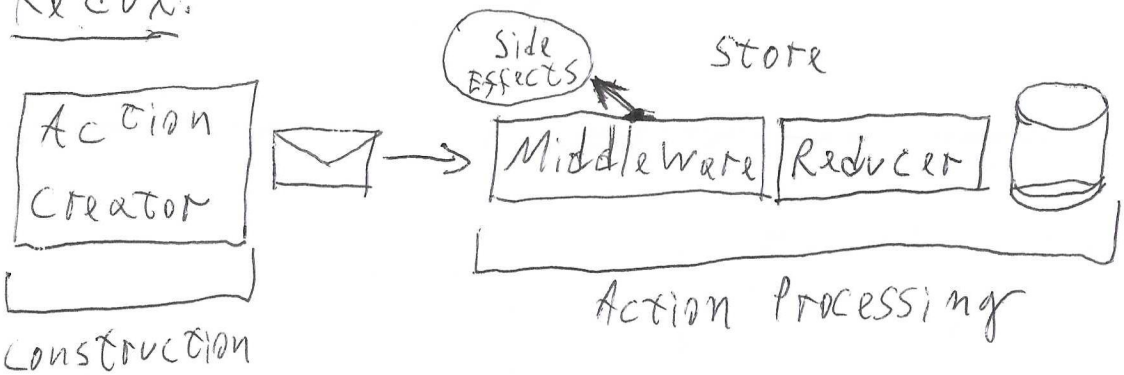
Queries

Return a result and do not change the observable state of the System.





Redux.



* Actions Categories
  - Events (Something happened)
  - Command (Ask for Something)
  - Document (write data to the state)

type	Intention
command	Start procedure
Event	Notify about a change
Document	write data to State

Events

Notify of a Change (more than one handler)

```
{ type: APP_OFFLINE,
  payload: data,
  meta: {...}
}
```

Command

Invoke a procedure (one handler, expect a reply)

```
{ type: GET_USER,
  payload: data,
  meta: {...}
}
```

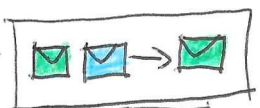
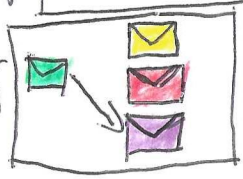
Document

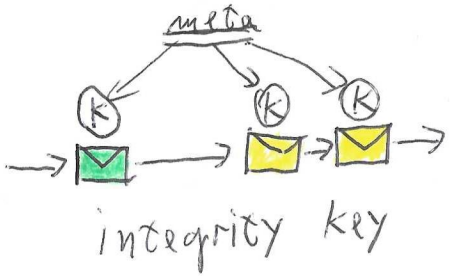
Transfer data (no reply, one reducer)

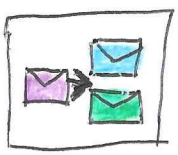
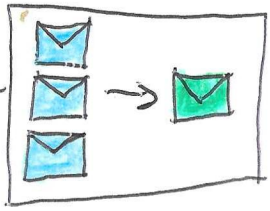
```
{ type: PRODUCT,
  payload: data,
  meta: {...}
}
```

* Usually the reducer get the data structure from the payload and put it on the state.
* The only type of actions that reducers process.

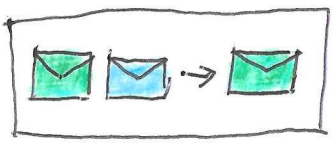
Routing Patterns (Middleware)

- Filtering 
- mapping 



- Splitter 
- Aggregator 

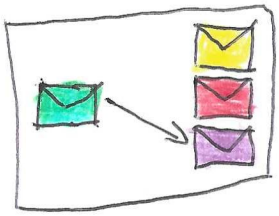
Filtering



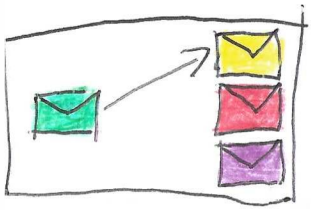
```
if(action.type === GET_DATA) {

}
```

Mapping

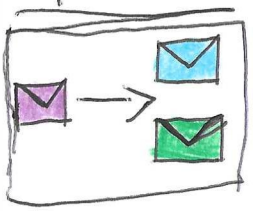


```
if(action.type === GET_DATA) {
  if(action.payload.force) {
    // decision based on content
  }
}
```



```
if(action.type === GET_DATA) {
  if(process.env.dev) {
  // decision based on context
  }
}
```

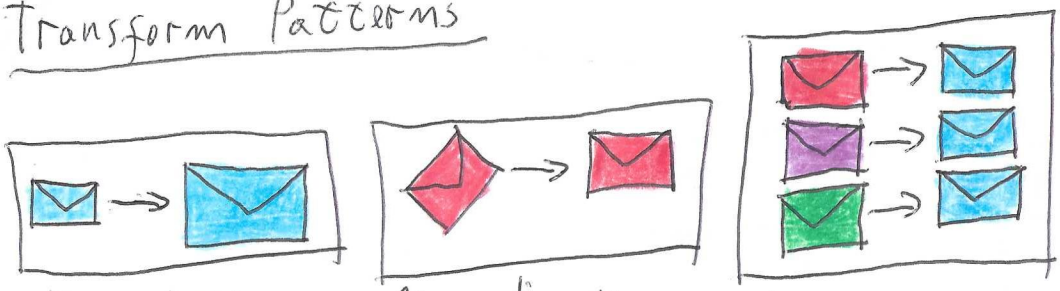
Splitter



```
if(action.type === GET_DATA) {
dispatch(API_REQUEST)
dispatch(APP_PENDING)
}
```

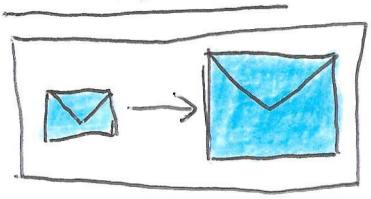
* keep the reducers with single responsibilities.
* payload agnostic.

Transform Patterns



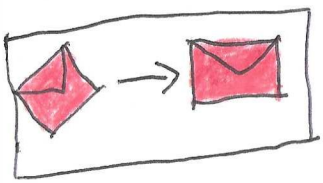
Enricher | Normalizer | Translator

Enricher



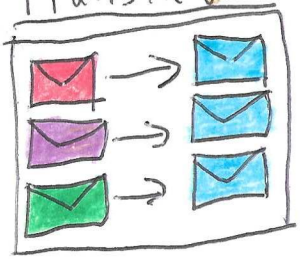
```
if (action.type === SUBMIT_ORDER) {
  let action = {
    ...action,
    created: new Date()
  };
  next(action)
}
```

Normalizer

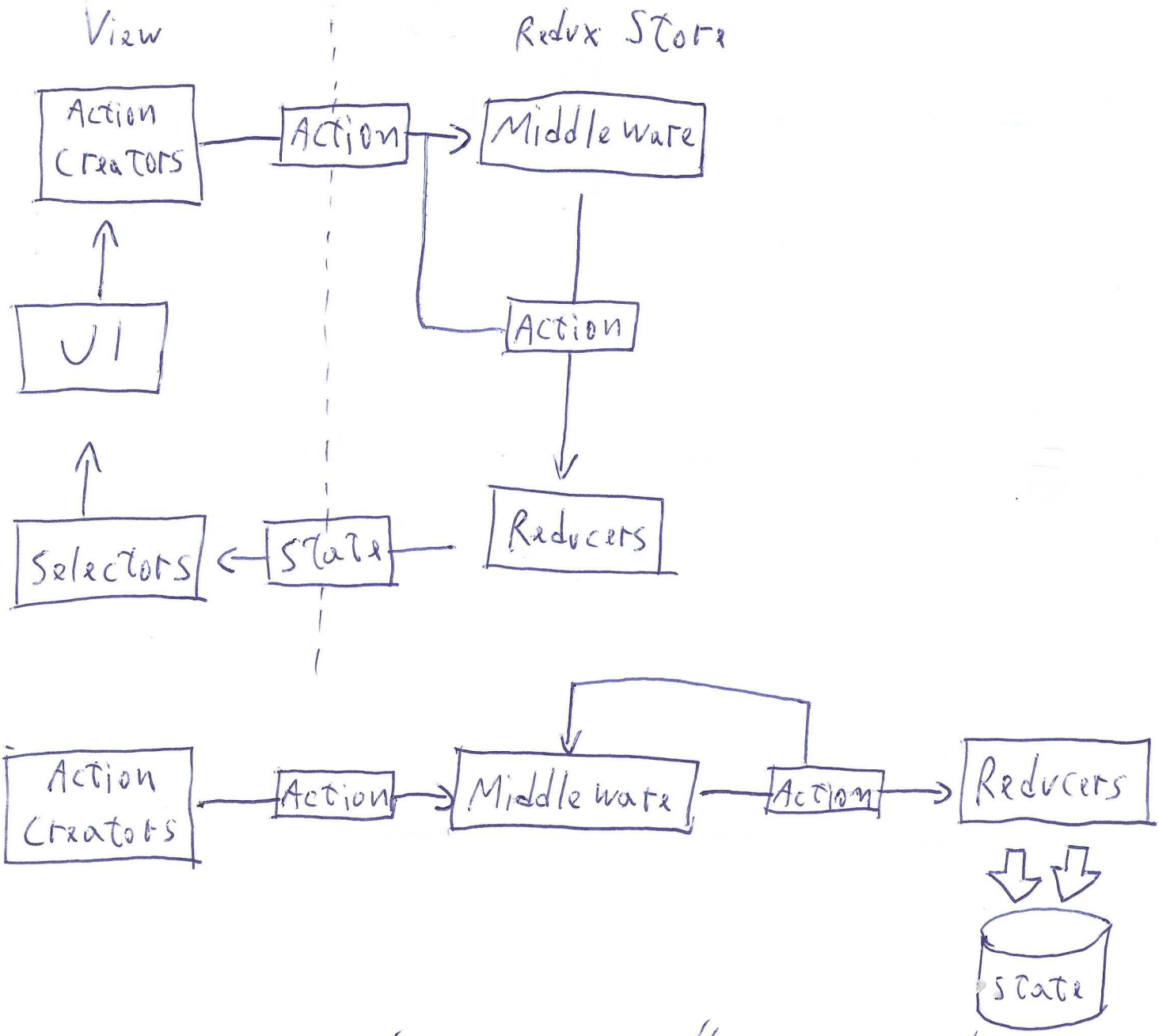


```
if (action.type === API_SUCCESS) {
  const users = action.payload.reduce(...);
  next({type: UPDATE_USERS, payload: users})
}
```

Translator



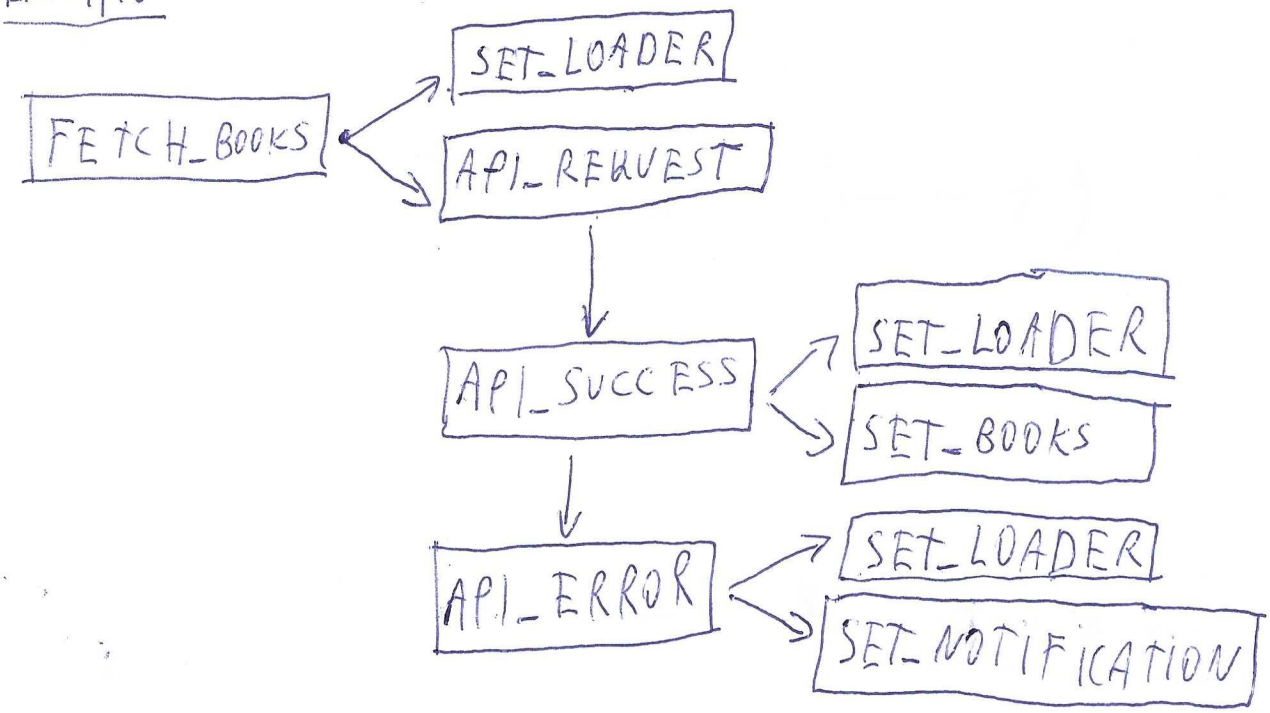
```
if (action.type === SHOW_LOADER) {
  dispatch({type: TOGGLE_LOADER, payload: true});
}
if (action.type === HIDE_LOADER) {
  dispatch({type: TOGGLE_LOADER, payload: false});
}
```

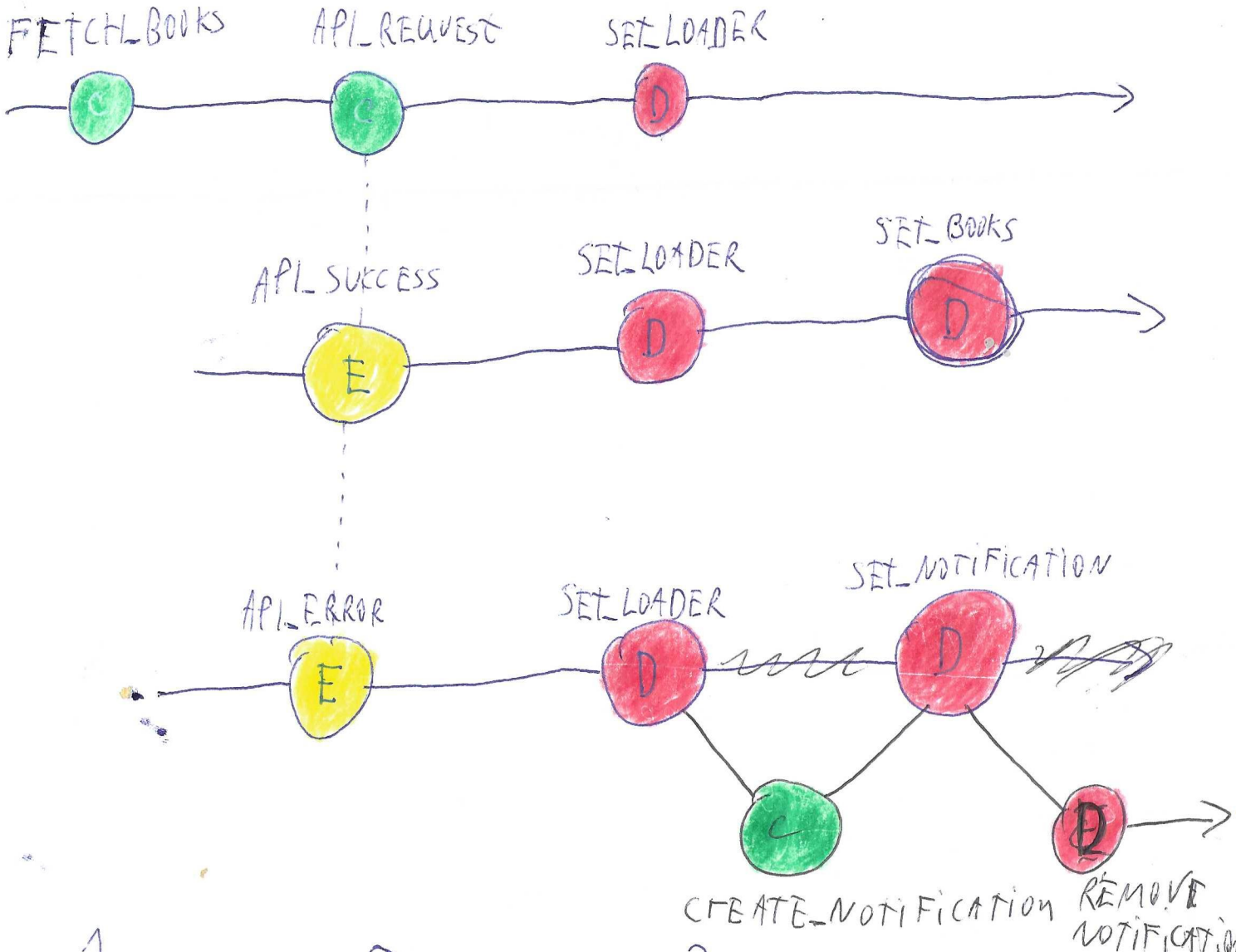

View
Redux Store
Action Creators
Action
Middle ware
UI
Action
Selectors
State
Reducers

Action Creators
Action
Middle ware
Action
Reducers
State

Example

Exemplo



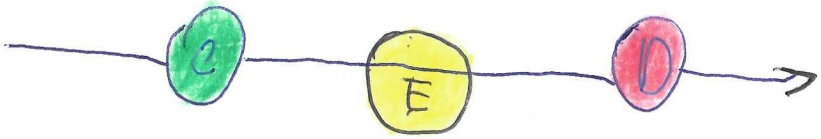


1 Actions TYPES

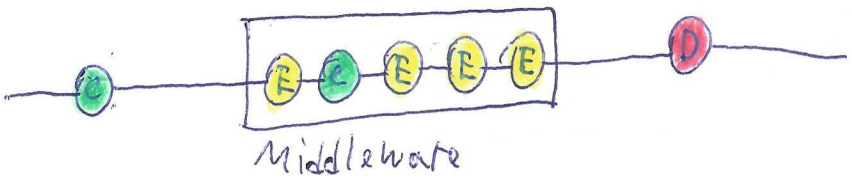
2 Actions Creators

3 Reducers

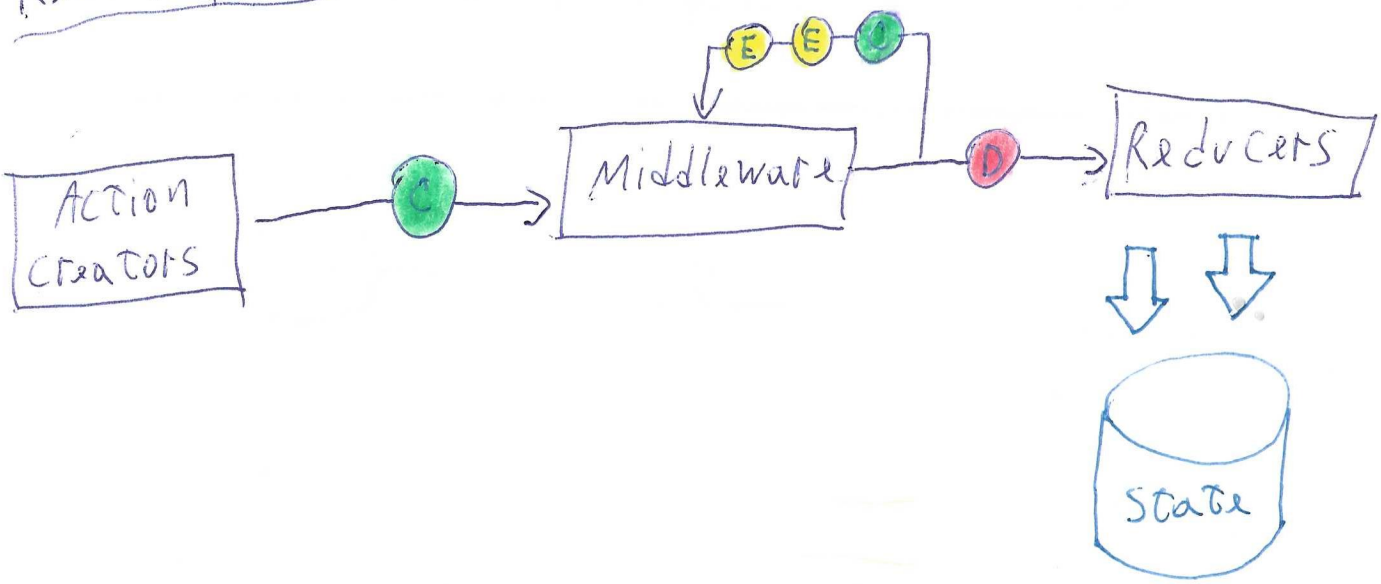
From Command to Document



Action processing Patterns



Redux flow



Middleware types

Core

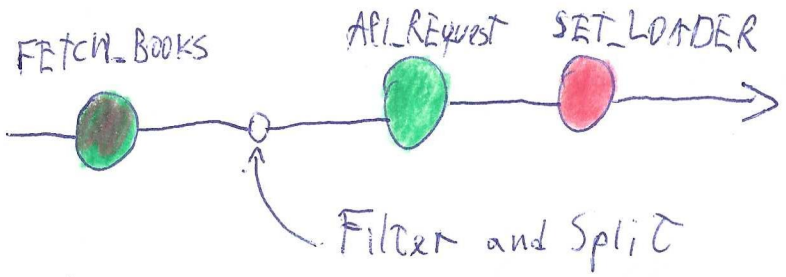
Generic and reusable

Application

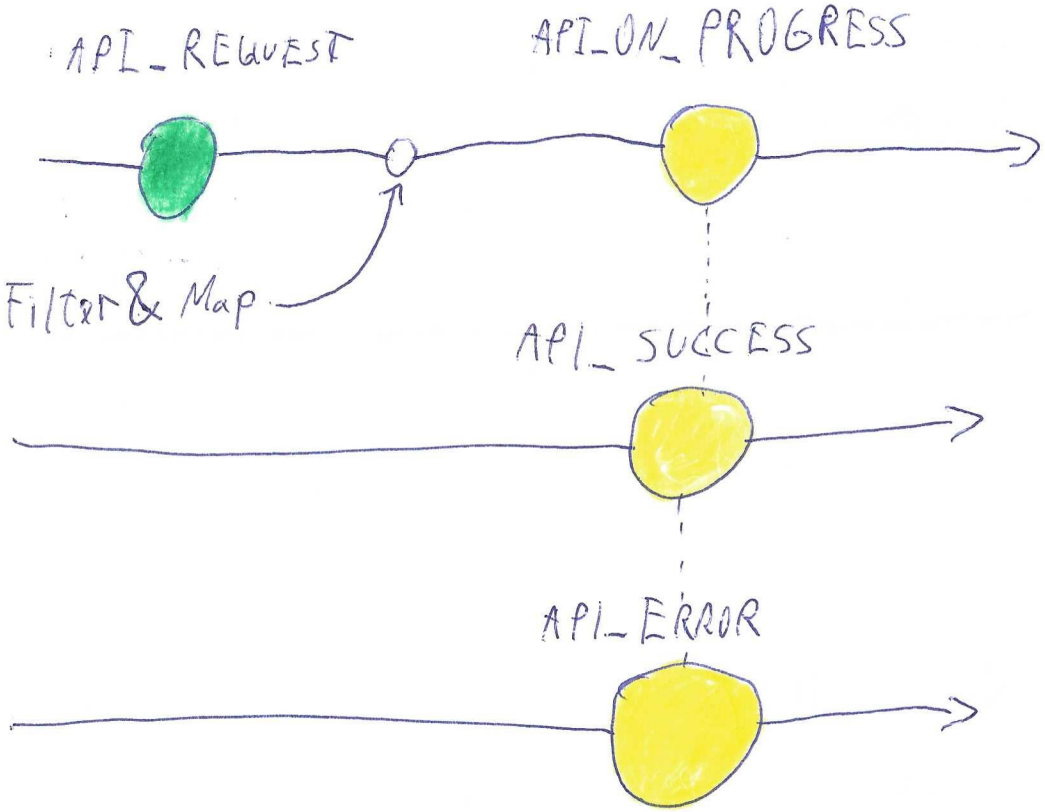
Describe a specific flow

Single Responsibility Principal

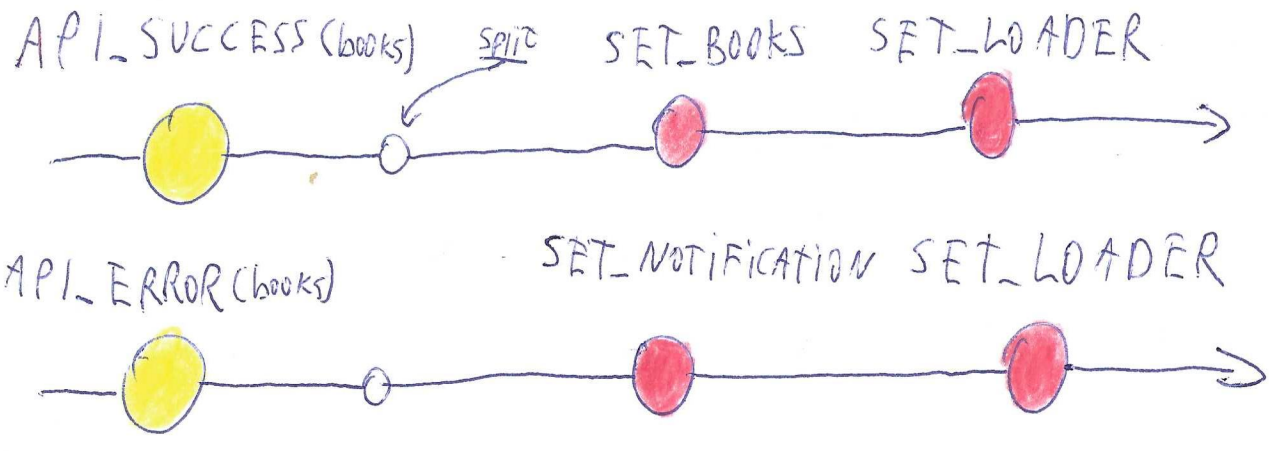
the Books Middleware:



The API Middleware:



Complete the flow



Exemplo

```
export const booksMiddleware = ({dispatch}) => next => action => {
  next(action);
  switch(action.type) {
    case FETCH_BOOKS:
      dispatch(apiRequest(null, 'GET', API.BOOKS, BOOKS));
      dispatch(setLoader(true));
      break;

    case `${BOOKS}${API_SUCCESS}`:
      dispatch(setBooks(action.payload));
      dispatch(setLoader(false));
      break;

};//Action-in -> Action-out

export const apiMiddleware = ({dispatch}) => next => action => {
  next(action);

  if(action.type.includes(API_REQUEST)) {
    const {entity, method, url} = action.payload.meta;

    fetch(url, {method})
      .then(res => res.json())
      .then(data => apiSuccess(data, entity))
      .catch(error => apiError(error, entity))
  }

};<- {type:'[Books] API_REQUEST', payload: {data: null, meta: {
    method: 'GET', url: 'http://www...', entity: '[Books]'}}}
```

Action Categories

type	Intention	Dispatched by	Processed by
Command	start a procedure	UI component, middleware	Middleware
Event	Notify about a change	Middleware	Middleware, Reducer
Document	write data to state	Middleware	Reducer