

The Art Of State

The essential of modern frontend architecture is state management

2018-05 @otakustay

Standard Redux Shape

- A developing design of global state shape
- Helpers aligned to edge state management theory
- Redux as infrastructure

<https://github.com/ecomfe/standard-redux-shape>

Redux

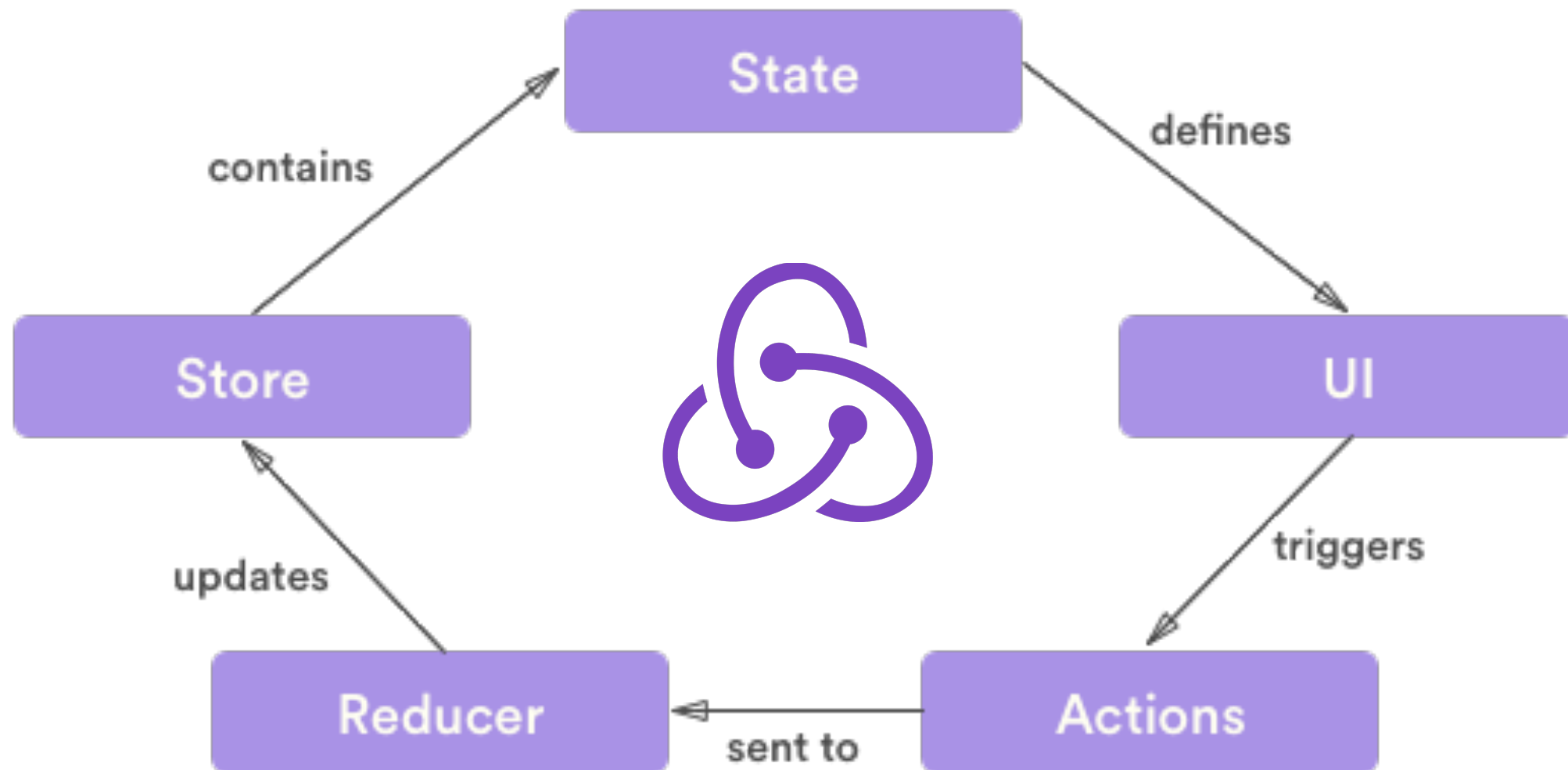
What

- Redux is a **predictable** state **container** for JavaScript apps.
- It helps you write applications that behave **consistently**, run in different environments (client, server, and native), and are **easy to test**.
- It provides a great **developer experience**

Why

- Share state
 - Between types
 - Between instances
- Track & debug
- Pre-rendering
 - Server side rendering
 - Service worker based client side rendering

How



State Driven

我的任务

...

卡片数量: 4

、、EE... ▾

所有状态 ▾

标题	状态	优先级	分派时间
1263 地图改为左右布局	新建 ▾		2018/5/27
1028 【代码健康度】当不选择代码库时, 代码重复文件的链接...	新建 ▾		2018/4/17
953 【前端】dashboard工程能力地图整体结果为NO_DATA时...	新建 ▾	P2-Middle	2018/3/7
992 【代码健康度】GET url请求参数超长导致414	新建 ▾	P0-Highest	2018/3/5

< 1 >



卡片数量:

、、精益... ▾

所有状态 ▾

标题	状态	优先级	分派时间

...

- Filter changed ✗
- Request pending ✗
- Result unavailable ✓

$$UI = f(state)$$

- Render when available
- Pending when unavailable
- Declarative & reactive

Thunk (Creator)

- A dependency injection solution for redux store
- Bridge `dispatch(fn)` to `fn(dispatch)`
- The power of multiple dispatches in a thunk
- Async goes easy as `dispatch` is available any time
- Valid to dispatch a thunk in a thunk
- Also, thunk creator is a function returns a thunk

Issue · State Sync

Default branch

`master` Updated 2 months ago by brodeynewman

Default

Change default branch

Your branches

`test` Updated 7 months ago by otakustay

66 | 2

New pull request



`expand-code` Updated 6 months ago by otakustay

42 | 1

New pull request



`demo/externals` Updated 7 months ago by otakustay

51 | 0

New pull request



`de'mo` Updated 7 months ago by otakustay

51 | 0

New pull request



Stale branches

`de'mo` Updated 7 months ago by otakustay

51 | 0

New pull request



`demo/externals` Updated 7 months ago by otakustay

51 | 0

New pull request



`test` Updated 7 months ago by otakustay

66 | 2

New pull request



`expand-code` Updated 6 months ago by otakustay

42 | 1

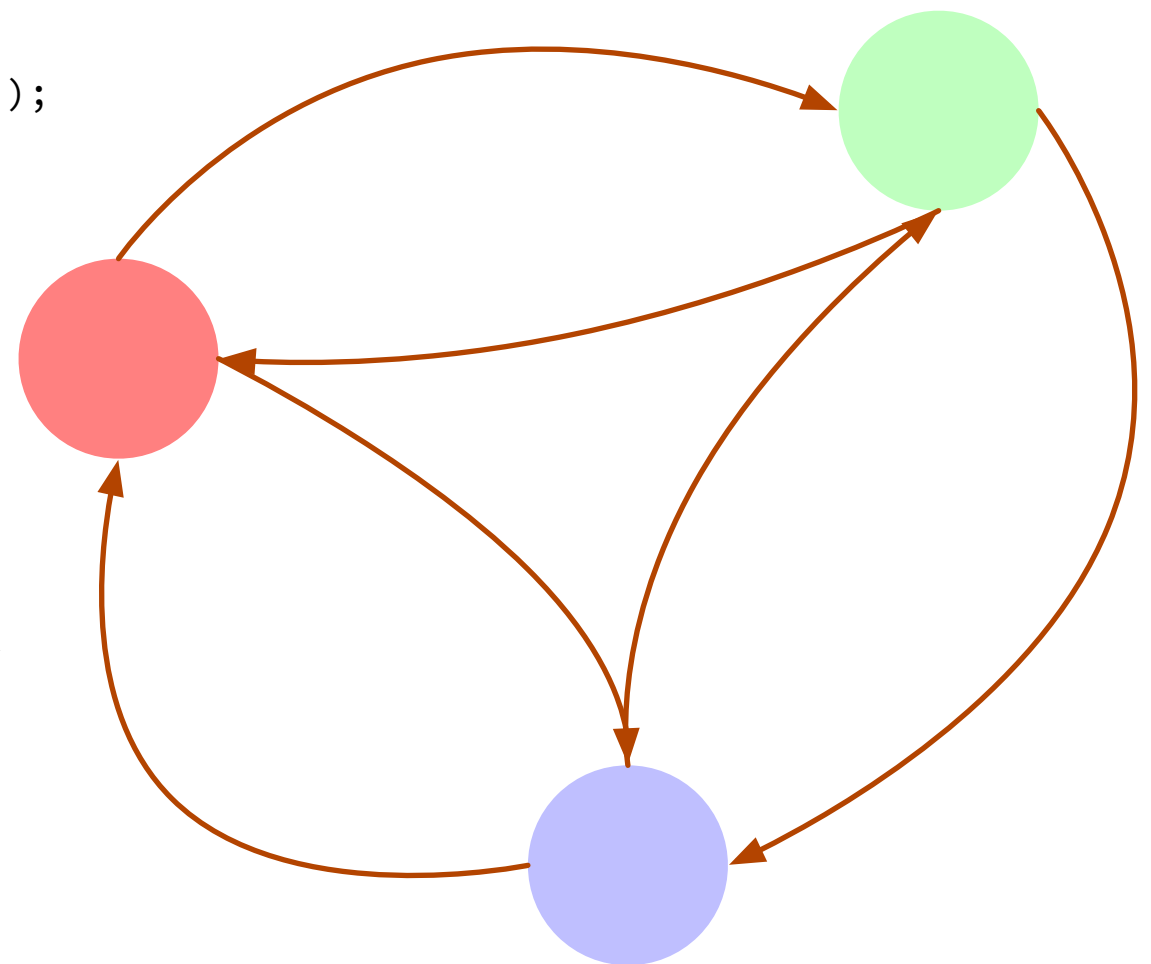
New pull request



Period Of Imperative

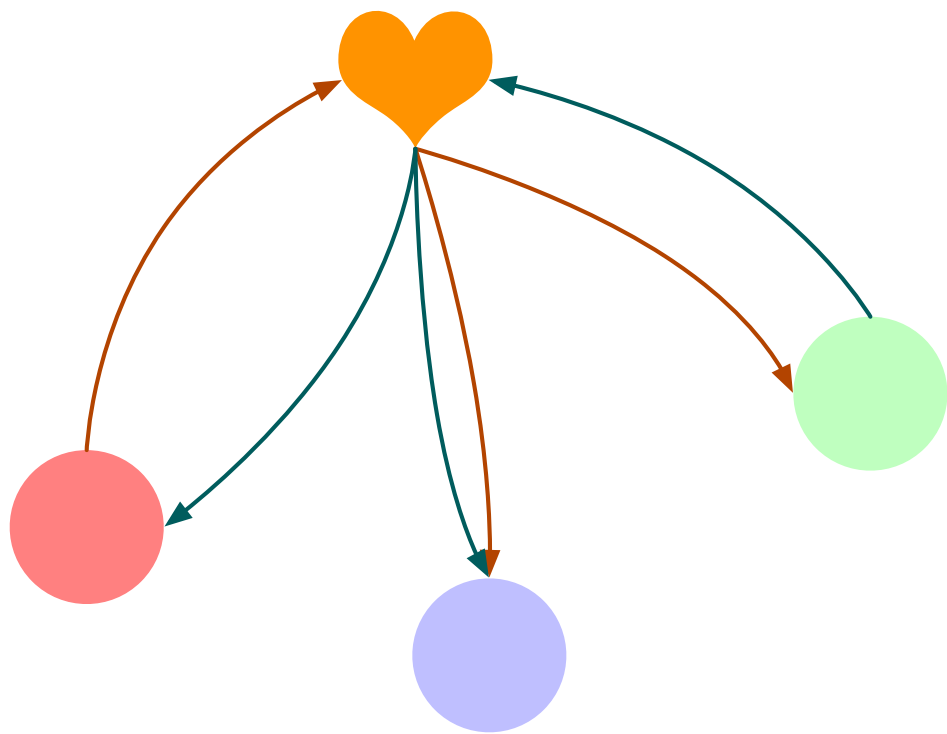
```
$('#page').on(  
  'click',  
  '.delete',  
  async e => {  
    const branchName = e.target.attr('data-branch-name');  
    const sections = $('[data-branch-name="${branchName}"]');  
    sections.addClass('removing');  
    await deleteBranch(branchName);  
    sections.removeClass('removing').addClass('removed');  
  }  
);
```

Exponential rise of complexity



Reactive Event Bus

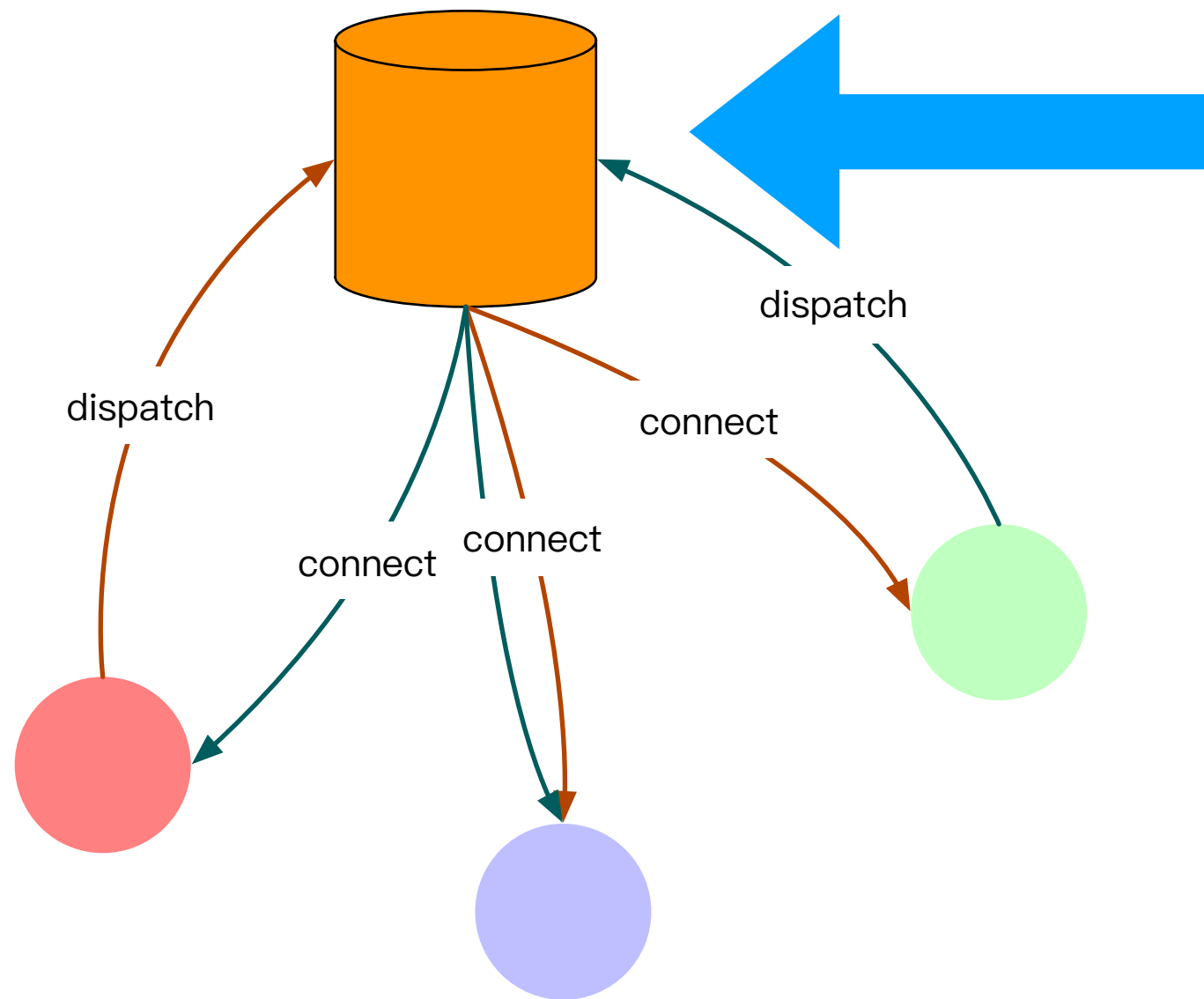
```
$('#page').on(  
  'click',  
  '.delete',  
  async e => {  
    const branchName = e.target.attr('data-branch-name');  
    $(document).trigger('branch-deleting', branchName);  
    await deleteBranch(branchName);  
    $(document).trigger('branch-delete', branchName);  
  }  
);
```



Unpredictable network

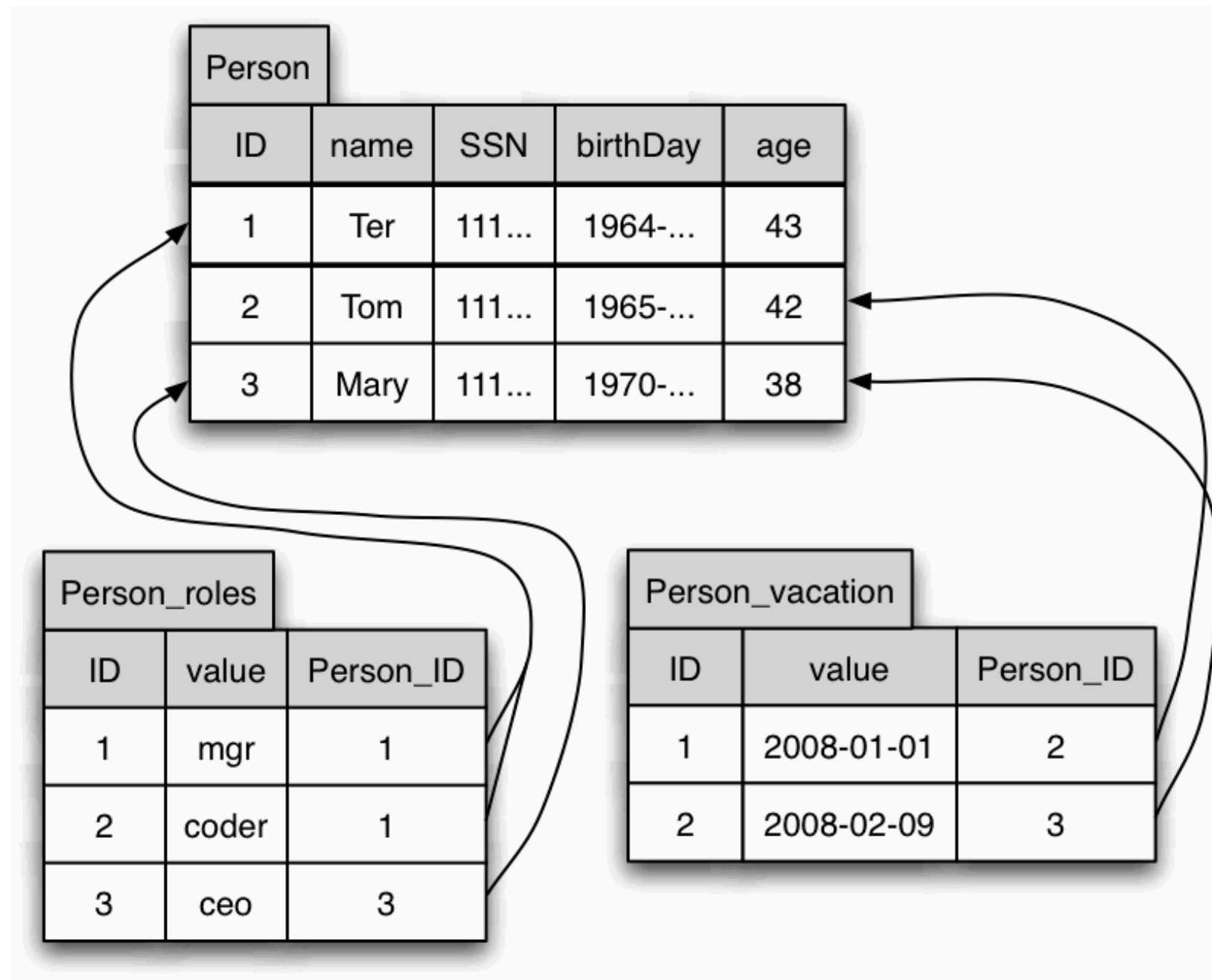
```
const Branchrow = {  
  markDeleting(branchName) {  
    if (branchName === this.branchName) {  
      $('#button').addClass('removing');  
    }  
  },  
  
  markDelete(branchName) {  
    if (branchName === this.branchName) {  
      $('#button').removeClass('removing').addClass('removed');  
    }  
  }  
  
  bindEvents() {  
    $(document).on('branch-deleting', this.markDeleting, this);  
    $(document).on('branch-delete', this.markDelete, this);  
  }  
};
```

Unidirectional Data Flow



- ✓ Predictable
- ✓ Consistent
- ✓ Serializable
- ✓ Compose-able
- ✓ Trackable
- ✓ Repeatable
- ✓ Debuggable

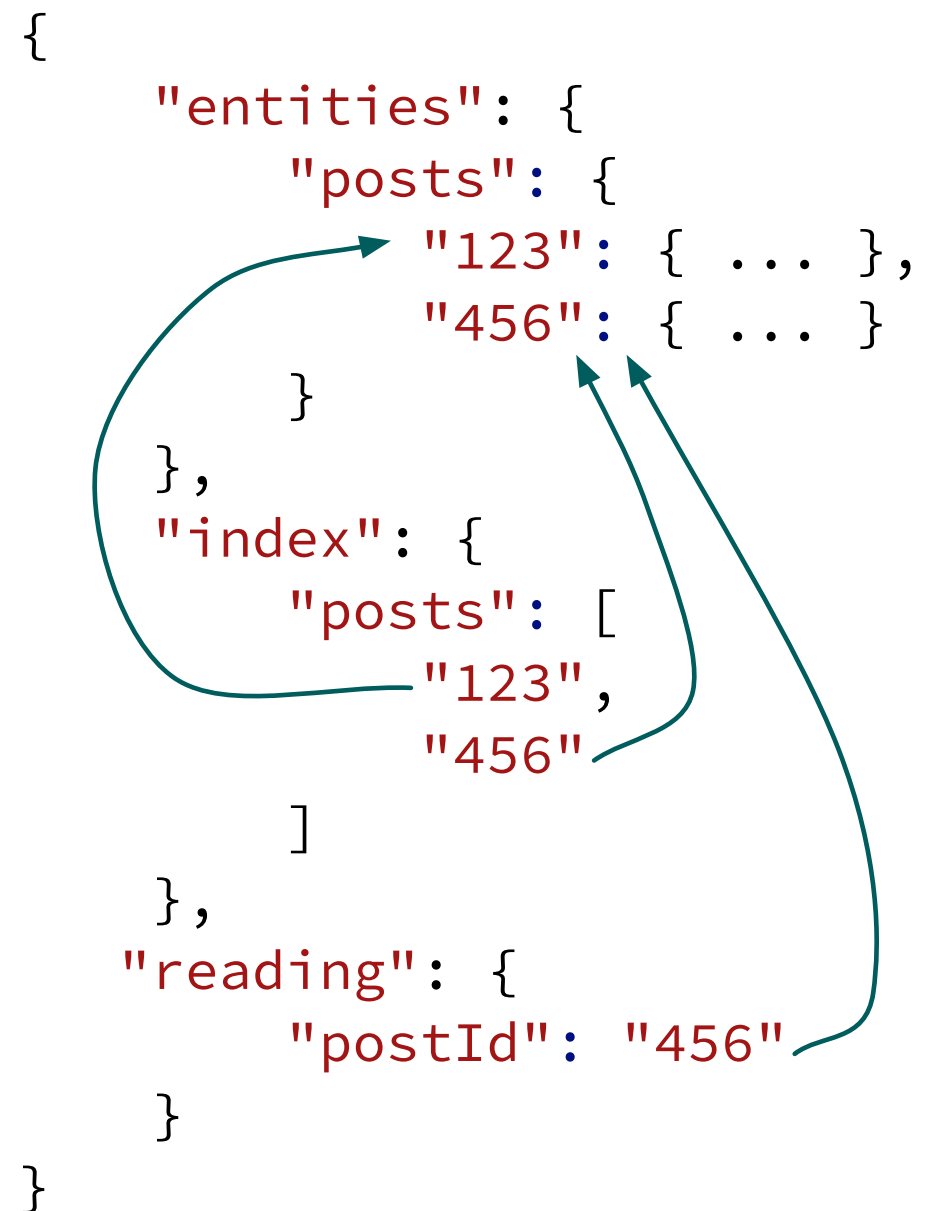
How Database Works



Normalization

State Normalization

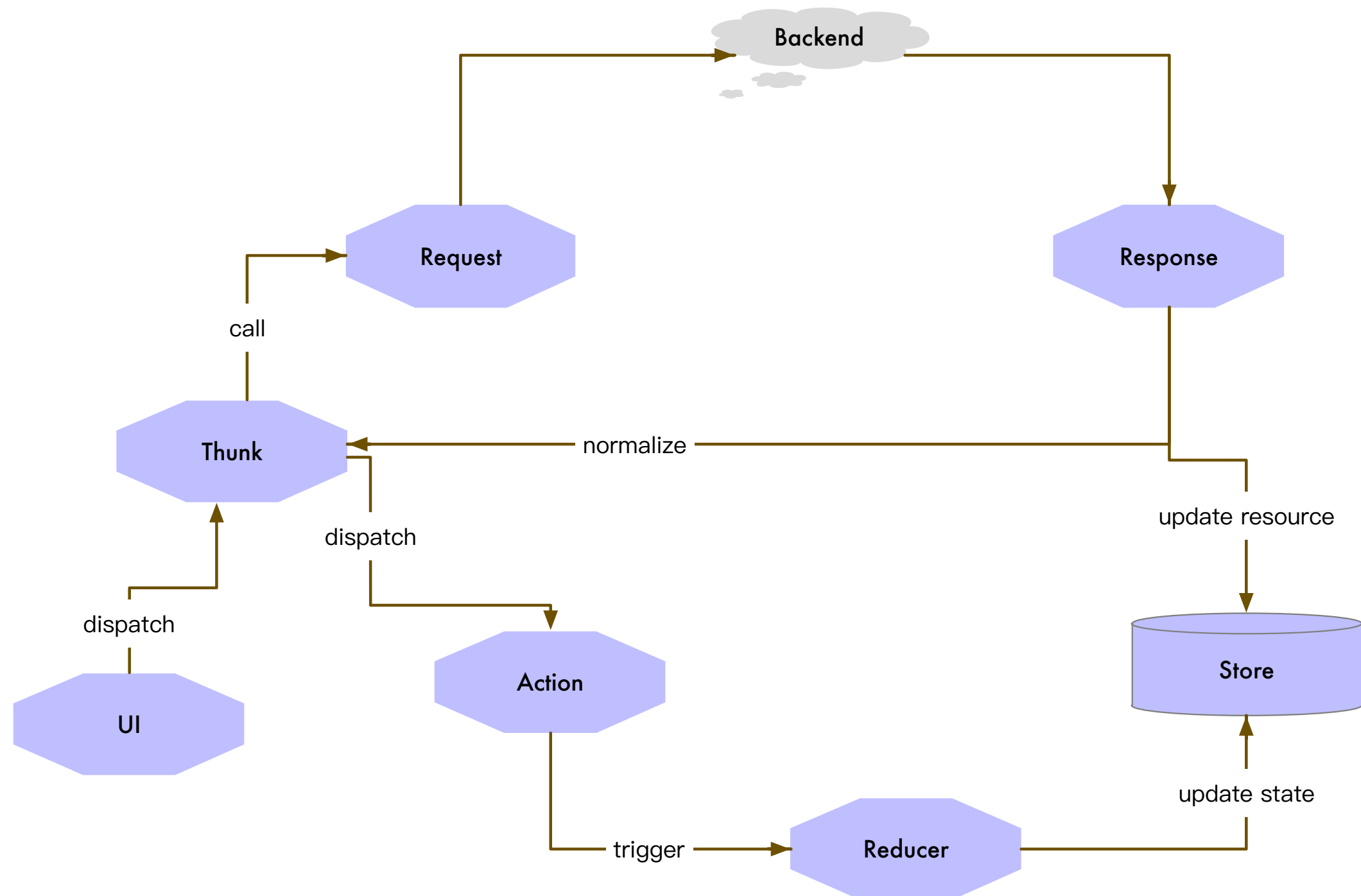
- One resource repository
- Partition by resource name
- Key by resource identity
- Keep only identity elsewhere
- Never duplicate state



Simplify Normalization

- Backend is the single source of truth
- Most backend resources come from HTTP response
- API tends to be RESTful
- Possible to directly connect response to store

How SRS Helps



Prepare The Store

```
// store/reducer.js
import {combineReducers} from 'redux';
import {createTableUpdateReducer} from 'standard-redux-shape';
import index from './index';
import reading from './reading';

const modules = {
  index,
  reading,
  entities: createTableUpdateReducer()
};

export default combineReducers(modules);

// store/index.js
import {createStore} from 'redux';
import thunk from 'redux-thunk';
import reducer from '../reducer';

export const store = createStore(
  reducer,
  compose(
    applyMiddleware(thunk),
    window.devToolsExtension ? window.devToolsExtension() : f => f
  )
);
```

Ready For API

```
// utils/fetch.js
import {createTableUpdater} from 'standard-redux-shape';
import axios from 'axios'

export const fetch = (method, url, options) => {
  // ...
};

// Tell SRS how to resolve the store instance
const resolveStore = () => import('store').then(m => m.store);

// Create a table update higher order function
export const withTableUpdate = createTableUpdater(resolveStore);
```

Wrap Your API

```
// api/posts.js
import {fetch, withTableUpdate} from 'utils';

// Define API function
const postsByQuery = query => fetch('GET', '/posts', query);

// Define how to extract resources from response
// Supposed response structure: [{id: 123, ...}, {id: 456, ...}]
const selectPosts = response => response.reduce(
  (posts, post) => ({...posts, [post.id]: post}),
  {}
);

// Tell SRS to update `entities.post` when response arrives
export const getPostsByQuery = withTableUpdate(selectPosts, 'post')(postsByQuery);
```

Move To Thunk Creator

```
// actions/posts.js
import {getPostsByQuery} from 'api';

export const FETCH_POSTS = 'FETCH_POSTS';
export const RECEIVE_POSTS = 'RECEIVE_POSTS';

export const fetchPostsByQuery = query => async dispatch => {
  dispatch({type: FETCH_POSTS, payload: query});

  const posts = await getPostsByQuery(query);

  // By now `state.entities.post` is updated to contain all posts

  // Normalize response to contain only identities
  const data = posts.map(post => post.id);
const data = posts.map(post => post.id);

  dispatch({type: RECEIVE_POSTS, payload: {query, data}});
};
```

Side Work A

The GET /posts API returns an array of posts in the format of {postId, title, content, authorId}

The GET /users/{id} API returns a single user in the format of {id, username}

Initialize a store with the entities property managing all resource entities

Implement a thunk creator that retrieves all posts as well as their authors, filling entities with post and user entity sets

To Sum Up

1. Create entities reducer and combine to store
2. Initialize with `TableUpdate` function
3. Write a simple fetch API
4. Write a function to extract entities from response
5. Wrap the API via `withTableUpdate`
6. Invoke API in action
7. Normalize the response
8. Dispatch action with normalized identity information

Issue · Race Condition

- 我负责的
- 我创建的
- 我参与的
- 已完成的
- 个人报表
- 共享报表

我负责的 120

所有空间

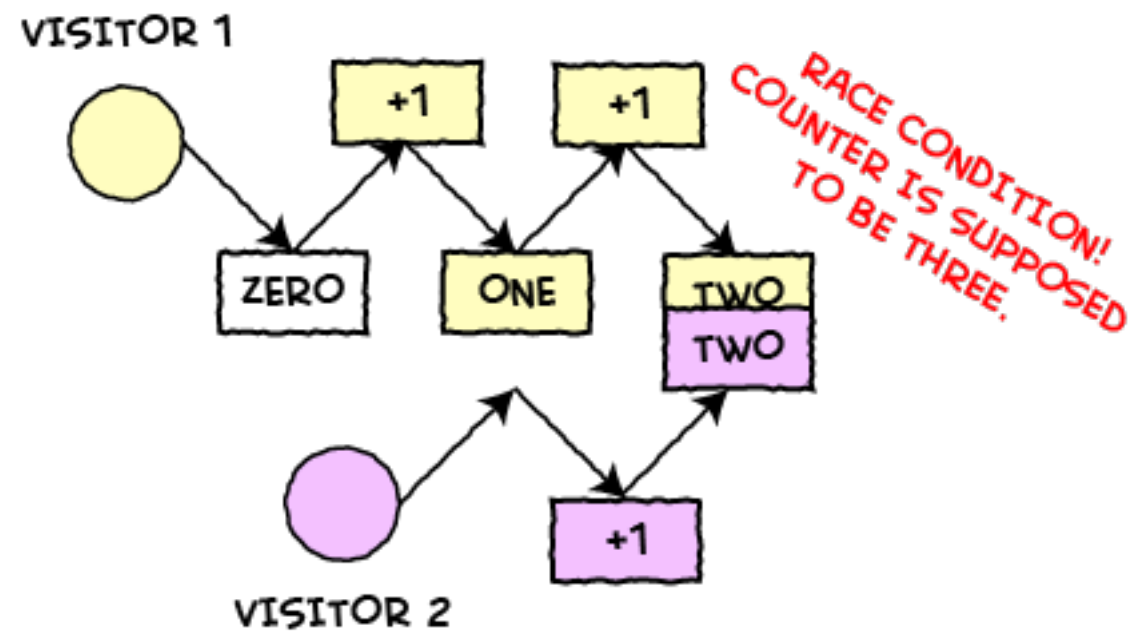
所有状态

EE-tech-992	【代码健康度】GET url请求参数超长导致414	Bug(线上)	EE软件研发智能化	2018年5月	2018-02-05	PO	新建
COLUMBUS-2976	在左侧导航栏，点击某个广告客户，新建的订单根本不显示，搜索亦搜不...	Bug	app/ecom/columbus		2013-12-02	PO	Active
EE-tech-1263	地图改为左右布局	Story	EE软件研发智能化	2018年5月	2018-05-27		新建
icode-9802	实现更优秀的代码高亮	用户体验	iCode	FE-Backlog	2018-04-20		新建
icode-9690	tab键显示问题反馈	用户反馈	iCode	RB25 05.22~05.28	2018-04-10		已临时处理
EE-tech-1028	【代码健康度】当不选择代码库时，代码重复文件的链接不应该被展示，...	Bug	EE软件研发智能化	2018年5月	2018-03-06		新建
sharepf-5625	接入ee-fe-tools	任务	百度河图		2018-04-13		新建
icode-9595	【线上BUG】代码对比修改代码没有显示高亮。	BUG (线上)	iCode		2018-03-28		新建
icode-9478	【线上BUG】两边如果不是同一种 编码格式的话，有一边会显示乱码						新建

体验新版



Source Of Race



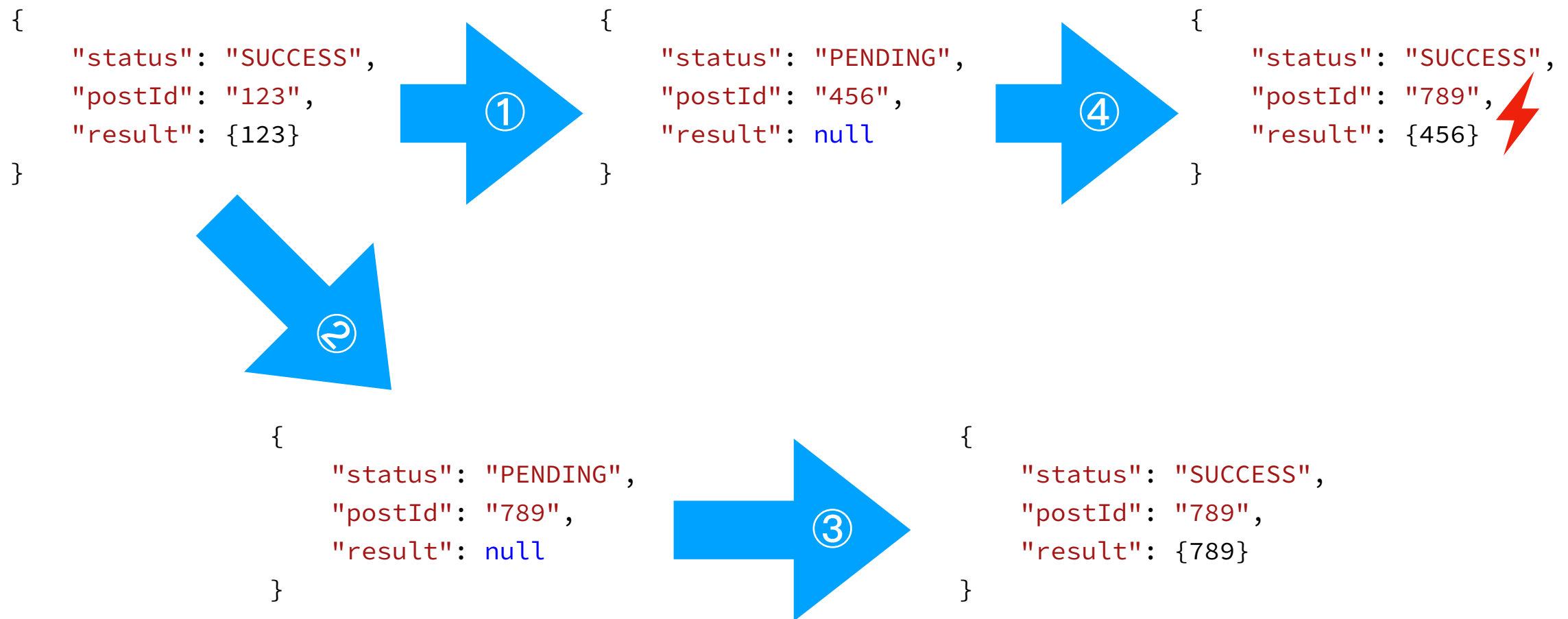
Multiple async process

Non-linear resolve timing

Missing state validation

Share of state

Traditional State Design



Possible Resolution

- To make async process singleton

Series all async works, **heavy performance drop**

- To keep resolution of async linear

Use an async manager, **complex dependency graph**

- To validate state before reduction

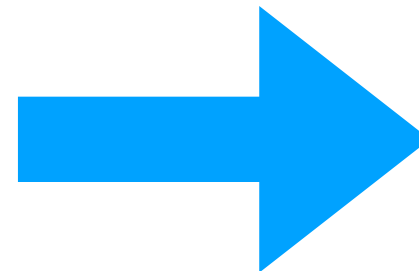
Case by case validation, **undesired waste of payload**

- To eliminate state sharing

Separate state for each work, **cheap memory cost**

Think Query As Entity

```
{
  "entities": {
    "posts": {
      "123": {
        "id": "123",
        "title": "Hello World",
        "content": "...",
        "createdAt": "2018-05-28T09:55:32.428Z"
      },
      "456": {
        "id": "456",
        "title": "Good Day",
        "content": "...",
        "createdAt": "2018-05-28T09:55:32.428Z"
      }
    }
  }
}
```



```
{
  "queries": {
    "postList": {
      "keyword=abc": [
        "123",
        "456"
      ],
      "keyword=xyz": [
        "789"
      ]
    }
  }
}
```

More Robust Structure

```
{
  A query set {
    "postList": {
      "{\\"keyword\\"=\\"abc\\",\\"pageIndex\\":1}": { Key by JSON.stringify(params)
        "params": {
          "keyword": "abc", Store original params
          "pageIndex": 1
        },
        "pendingMutex": 0, Count of pending requests
        "response": {
          "arrivedAt": 1527501781725, When response arrives
          "data": [
            // ... Actual data dispatched with action
          ],
          "error": {
            // ... Possible error when async work fails
          }
        },
        Unaccepted next response "nextResponse": {
          // Same as response
        }
      }
    }
  }
}
```

Stage Of Request

```
{  
  "params": {  
    // ...  
  },  
  "pendingMutex": 0,  
  "response": null,  
  "nextResponse": null  
}
```

FETCH

```
{  
  "params": {  
    // ...  
  },  
  "pendingMutex": 1,  
  "response": null,  
  "nextResponse": null  
}
```

RECEIVE

Usually receive &
accept are merged
into a single step

```
{  
  "params": {  
    // ...  
  },  
  "pendingMutex": 0,  
  "response": {  
    "arrivedAt": 1527501781725,  
    "data": [  
      // ...  
    ]  
  },  
  "nextResponse": null  
}
```

ACCEPT

```
{  
  "params": {  
    // ...  
  },  
  "pendingMutex": 0,  
  "response": null,  
  "nextResponse": {  
    "arrivedAt": 1527501781725,  
    "data": [  
      // ...  
    ]  
  }  
}
```

Side Work B

Currently we have a query set in the place of queries.postList, each query includes {keyword, author, pageIndex}

Aside of that, the index.ui property of state keeps current filter information in the format of {keyword, author}

Please implement a selector (using reselect) to grab all pages of posts matching the filter, keep them in paging order, remove duplicate entries and return a flat array

```
{
  "queries": {
    "postList": {
      // ...
    }
  },
  "index": {
    "ui": {
      "keyword": "hello",
      "author": "otakustay"
    }
  }
}
```


Side Work C

By the time we implement side work B, now we decide to introduce a "write new post" button which opens a form to create a new post in top navigator.

After a new post is submitted, it should appear in top of any post list with a "NEW" badge aside of it's title, no re-fetching should happen here.

The next time this post is fetched from backend, "NEW" badge will disappear and everything goes normal.

Please design a store structure to satisfy above feature requests.

```
{  
  "queries": {  
    "postList": {  
      // ...  
    }  
  },  
  "index": {  
    "ui": {  
      "keyword": "hello",  
      "author": "otakustay"  
    }  
  }  
}
```

Patch Entity Table

```
{
  "entities": {
    "post": {
      "123": {...},
      "456": {...}
    }
  },
  "insertions": {
    "post": [
      {"id": "789", ...},
    ]
  },
  "deletions": {
    "post": [
      "456"
    ]
  },
  "queries": {
    "postList": [
      "123",
      "456"
    ]
  }
}

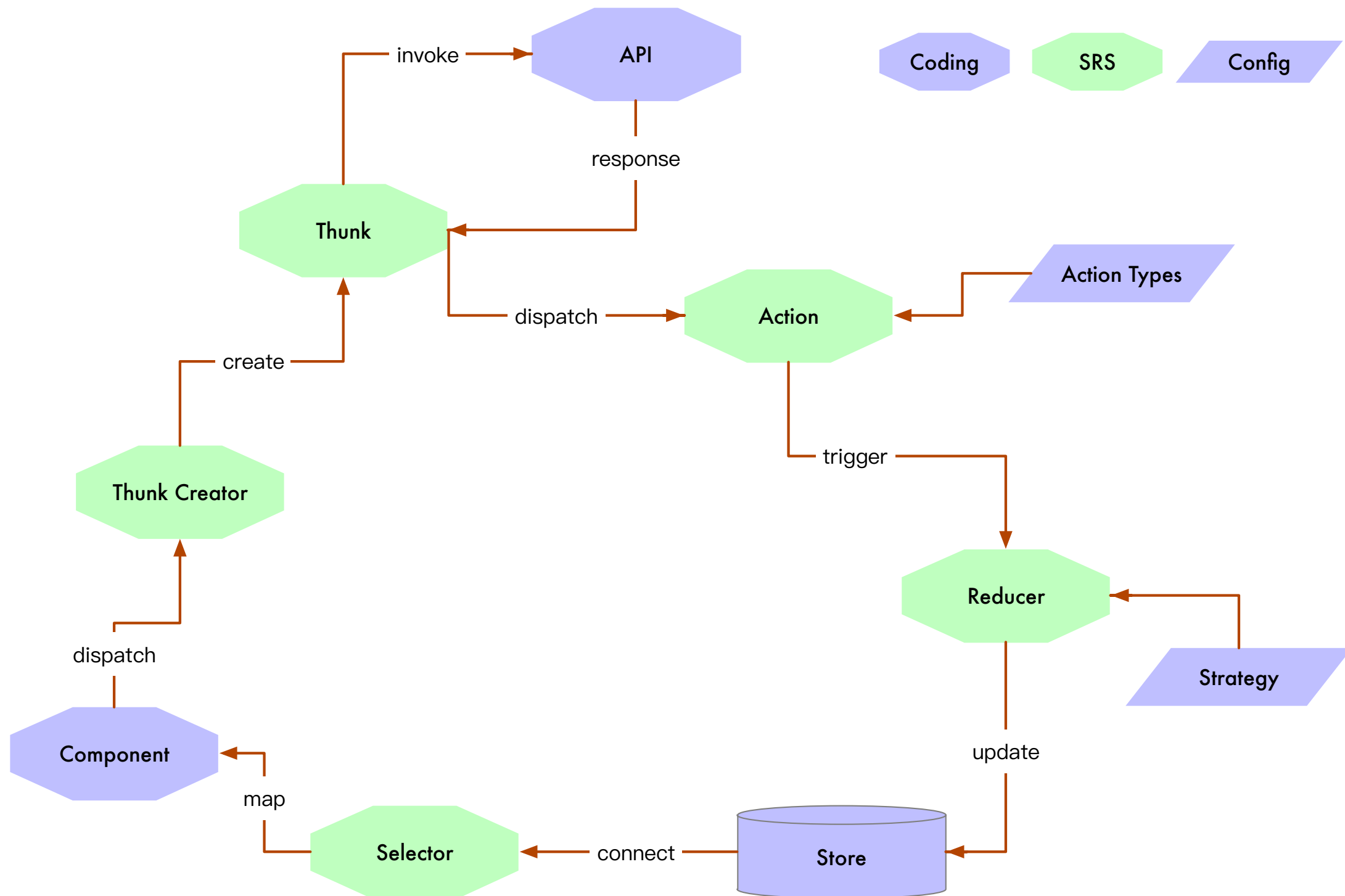
// To select a list of posts
const selectPostList = createSelector(
  get('entities.post'),
  get('insertions.post'),
  get('deletions.post'),
  get('queries.postList'),
  (postsByID, insertions, deletions, posts) => {
    const availables = difference(posts, deletions);

    return {
      insertions: insertions,
      posts: availables.map(id => postsByID[id])
    };
  }
);

// To remove patches after sync
const fetchPostsByQuery = query => dispatch => {
  dispatch({type: 'FETCH_POSTS', payload: query});
  const data = await getPostsByQuery(query);
  dispatch({type: 'RECEIVE_POSTS', payload: {data, query}});

  // Will empty insertions.post
  dispatch({type: 'CLEAR_INSERTION_PATCH', payload: 'post'});
};
```

How SRS Helps



Quick Thunk Creator

```
// actions/posts.js
import {getPostsByQuery} from 'api';
import {thunkCreatorFor} from 'standard-redux-shape';

export const FETCH_POSTS = 'FETCH_POSTS';
export const RECEIVE_POSTS = 'RECEIVE_POSTS';

export const fetchPostsByQuery = thunkCreatorFor(
  getPostsByQuery, // API function
  FETCH_POSTS, // Action type for fetch stage
  RECEIVE_POSTS, // Action type for receive stage
  {
    once: true, // Always reuse existing response
    trustPending: true, // Only fetch once
    selectQuerySet(state) { // Where is the query set
      return state.queries.postList;
    }
  }
);
```

Reducer By Strategy

```
// reducers/posts.js
import {acceptLatest} from 'standard-redux-shape';
import {combineReducers} from 'redux';
import {FETCH_POSTS, RECEIVE_POSTS} from 'actions';

const strategies = {
  // Always accept the latest arrived response immediately
  postList: acceptLatest(FETCH_POSTS, RECEIVE_POSTS)
};

const queries = combineReducers(strategies);

const ui = (state = {keyword: '', author: ''}, {type, payload}) => {
  // ...
};

export default combineReducers({queries, ui});
```

Back To Container

```
// containers/Index.js
import {get} from 'lodash/fp';
import {createSelector} from 'reselect';
import {createQueryDataSelector} from 'standard-redux-shape';
import {Index} from 'components';
import {fetchPostsByQuery} from 'actions';

const selectFilter = createSelector(
  get('index.ui.keyword'), get('index.ui.author'),
  (keyword, author) => ({keyword, author})
);

// Create a selector retrieving the `query.response.data` property
const selectPostList = createQueryDataSelector(
  get('queries.postList'), // Where the query set is
  selectFilter // How to select params
);

const mapStateToProps = state => ({filter: selectFilter(state), posts: selectPostList(state)});

const mapDispatchToProps = {onRequestPostList: fetchPostsByQuery};

export default connect(mapStateToProps, mapDispatchToProps)(Index);
```

Side Work D

As above presentation illustrates, it's quite easy to go around a full redux data flow, however when the API rejects, our UI will not receive a valid data object.

Please refine the selectors in container module to handle errors when API rejects, it should pass an error prop to Index dumb component in case of rejection.

Also, write a simple Index dumb component to illustrate how error and data work in different cases.

Side Work E

Now we are able to handle API rejections smoothly, but multiple pending requests give another challenge, the UI will update times in a short period if pending requests arrived one by one.

Please continue to refine the selectors to avoid such problems, we'd prefer UI to keep loading when any request is pending, it could render after all responses arrive.

To Sum Up

- thunkCreatorFor to quickly create a thunk creator around existing API function
- Strategies to define reducers in a single line of code
- createQuery(Response|Data|Error)Selector to select query, response, data or error with joy
- More API documented at [github repo](#)

Issue · Transient State

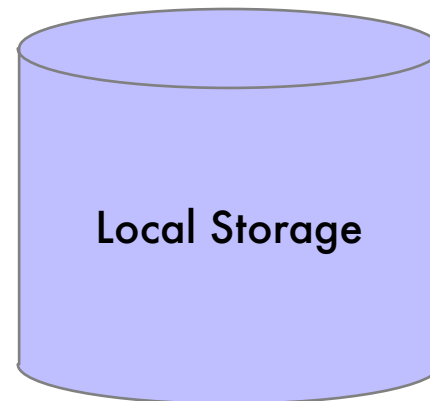
请输入您的意见：

|

提交

Transient vs Integrity

```
{  
  feedback: {  
    submitting: true,  
    content: 'Feature Request...'  
  }  
}
```



```
{  
  feedback: {  
    submitting: true,  
    content: 'Feature Request...'  
  }  
}
```

Async never resolves
Forever submitting state

Keep Transient Local

```
import {posts} from 'api';

const updatePost = post => dispatch => {
  dispatch({type: 'POST_UPDATING', payload: post});

  const newPost = await posts.put(post.id, post);

  dispatch({type: 'POST_UPDATED', payload: newPost});

  return newPost;
};
```

```
class PostForm extends PureComponent {

  state = {
    submitting: false
  };

  @bind()
  updatePost(post) {
    const {id, updatePost} = this.props;

    this.setState({submitting: true});
    await updatePost({...post, id});
    this.setState({submitting: false});
  }
}
```

Strip Transient States

```
// redux-storage-decorator-clone
import {cloneWith, identity} from 'lodash'
export default (engine, customizer = identity) => {
  return {
    ...engine,

    save(state) {
      const customizedState = cloneWith(state, customizer);
      return engine.save(state);
    }
  };
};

const stripMutex = obj => {
  if (typeof obj === 'object' && 'pendingMutex' in obj) {
    return {
      ...obj,
      pendingMutex: 0
    };
  }

  return obj;
};
```

Go Optimistic

```
import {posts} from 'api';

const updatePost = post => dispatch => {
  dispatch({type: 'POST_UPDATING', payload: post});

  const newPost = await posts.put(post.id, post);


  dispatch({type: 'POST_UPDATED', payload: newPost});

  return newPost;
};

const updatePostOptimistic = post => dispatch => {
  dispatch({type: 'POST_UPDATING', payload: post});
  dispatch({type: 'POST_UPDATED', payload: post});

  return Promise.resolve(post);
};
```

```
const mapDispatchToProps = dispatch => {
  return {
    onPostCommit(post) {
      const thunks = [
        updatePost(post),
        updatePostOptimistic(post)
      ];
      dispatch(thunks);
    }
  };
};
```



Traditional thunk

Immediate response

<https://github.com/ecomfe/redux-optimistic-thunk>

Side Work F

Our `redux-optimistic-thunk` is a great tool to create optimistic UI, however a little redundant code is introduced.

In most cases, the only difference between actual and optimistic thunk is the payload of final action, which comes from response in actual thunk and is a mock data in optimistic thunk.

Please write a helper function that receives an async API function and a sync mock response generator, then return both the actual and optimistic thunk, it will be something like:

```
optimisticThunkPairCreatorFor(api, toMockResponse);
```


Side Work G

Now we have another issue, implement of onPostCommit inside mapDispatchToProps function is a little complex, this could be a significant cost when more and more container components appears.

Please write a helper function to simplify the process of connecting optimistic thunk creator pairs to containers, we'd prefer something like:

```
bindOptimisticThunkPairCreator(creator, dispatch);
```

Homework

- Read all API definitions of SRS
- Explain the differences between strategies
- Write a new strategy that overrides old response which arrives 5 or more minutes earlier
- Explain how once & pendingTrust option work
- Implement the simple blog app yourself in redux without SRS
- Again implement the blog app using SRS
- Write unit tests for SRS and commit to github