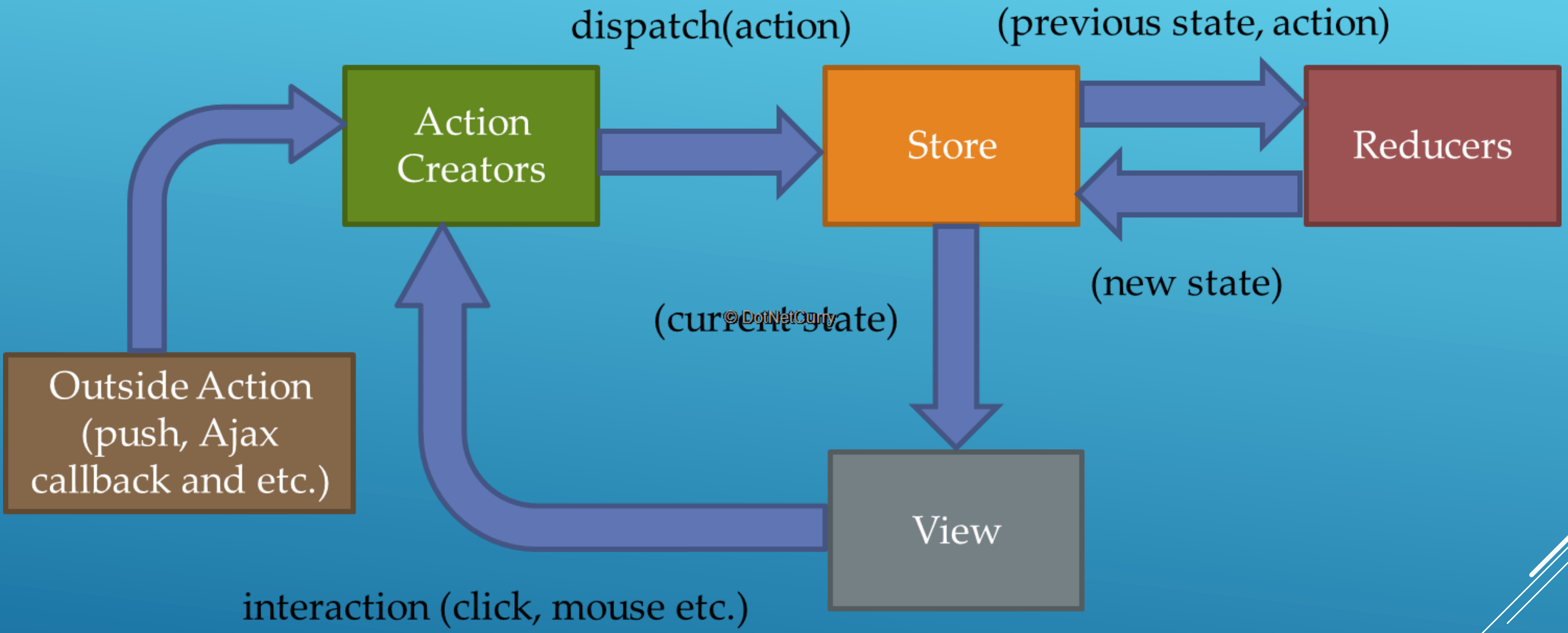
REACT JS

Redux Thunk

Sreekanth M. E.

Freelance Trainer & Consultant

http://www.SreekanthME.com


dispatch(action)
(previous state, action)
Action Creators
Store
Reducers
(new state)
(current state)
Outside Action (push, Ajax callback and etc.)
View
interaction (click, mouse etc.)

In Redux, Action Creators must return an object.

Action creators when triggered perform synchronous data flow by default.

To handle ASYNC requests, the action creator can be made to return a function instead of an action object.

ASYNC call is made inside this returned function.

Such action creators are called Async action creators.

A middleware named redux-thunk can handle action creators that return a function.

Thus it enables Redux to support ASYNC data flow.

Redux-thunk injects the function returned by the action creator with the store methods dispatch() and getState() as parameters.

The function makes an ASYNC call and once the ASYNC response/error is received, it calls dispatch() and passes an action object to it.

The action object so dispatched, by the function, may contain response data or error details from the ASYNC call.

Redux-thunk can delay the dispatch of an action, or dispatch only if a certain condition is met.

ORDINARY ACTION CREATOR

```
function increment() {
  return {
    type: INCREMENT_COUNTER
  };
}
```

ASYNC ACTION CREATOR

```
function incrementIfOdd() {
  return (dispatch, getState) => {
    const counter= getState().counter;

    if (counter % 2 === 0) {
      return;
    }

    dispatch(increment());
  };
}
```

```
npm install --save redux-thunk
```

https://www.npmjs.com/package/redux-thunk

AXIOS

Axios is a Promise-based HTTP client for JS which can be used in front-end application (like React) or Node.js backend.

Axios can send asynchronous HTTP requests to REST endpoints and perform CRUD operations.

```
npm install --save axios
```

https://www.npmjs.com/package/axios

https://unpkg.com/axios/dist/axios.min.js

```
// Make a request for a user with a given ID
axios.get('/user?ID=12345')
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

END OF CHAPTER

APPENDIX

ASYNC REQUEST HANDLING

For any asynchronous API request you'll want to dispatch at least three different kinds of actions:

1. An action informing the reducers that the request began.

The reducers may handle this action by toggling an isFetching flag in the state. This way the UI knows it's time to show a spinner.

2. An action informing the reducers that the request finished successfully.

The reducers may handle this action by merging the new data into the state they manage and resetting isFetching flag. The UI would hide the spinner, and display the fetched data.

3. An action informing the reducers that the request failed.

The reducers may handle this action by resetting isFetching. Additionally, some reducers may want to store the error message so the UI can display it.