Redux

FE ACADEMY PRESENTATION

Darjan Baričević, average Redux non-enjoyer

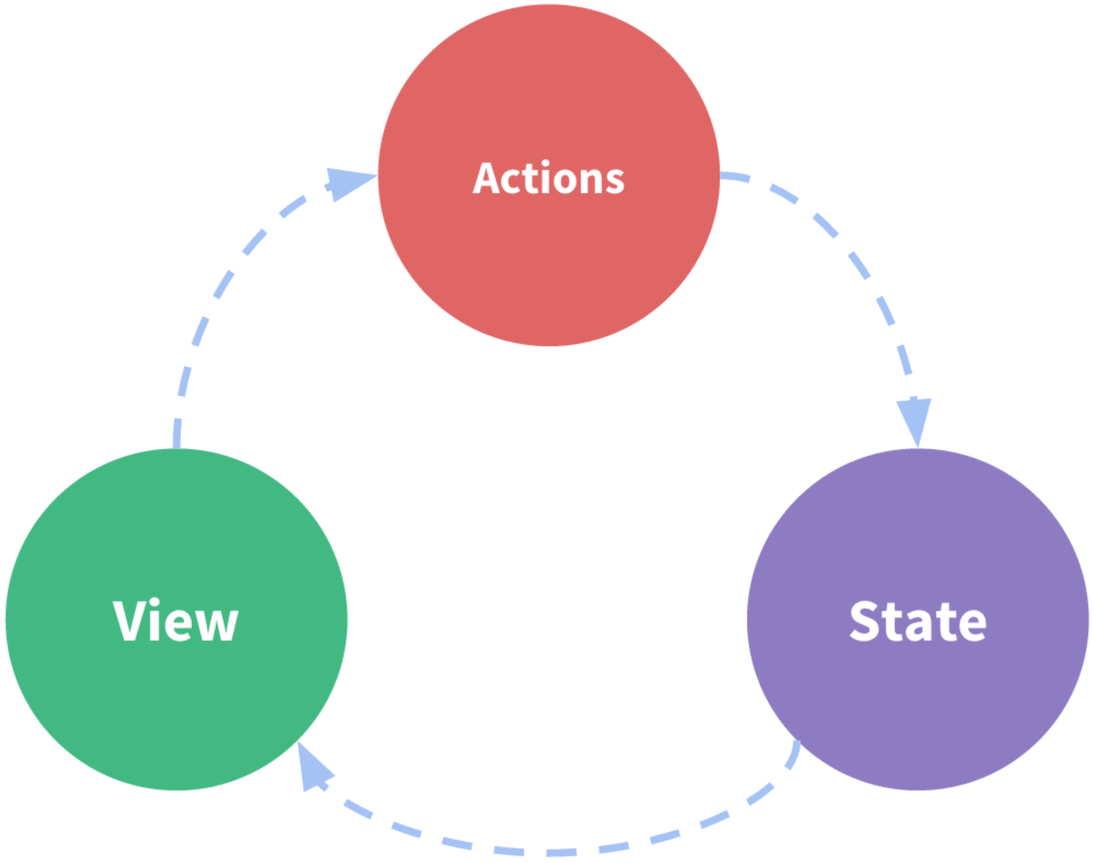
Table of contents

Terminology

Terminology

State Management

- simply refers to way of managing data and application state
- one-way data flow



Terminology

Immutability

- immutable basically means something that cannot be changed
- JS objects and arrays are all mutable by default
- beneficial for several reasons:
    1. Improved performance
    2. Reduced memory usage
    3. Thread safety
    4. Lower developer mental burden

Few examples..

Inserting item in array

Mutable way

```
function insertItem(array, item) {
  array.push(item)
}
```

Immutable way

```
function insertItem(array, item) {
  return [...array, item]
}
```

Removing item from array

Mutable way

```
function removeItem(array, index) {
  array.splice(index, 1)
}
```

Immutable way

```
function removeItem(array, index) {
  return array.filter((item, i) => i !== index)
}
```

Immutability in practice

- few popular libraries:
  - immutable.js
  - immer.js

Immutability in practice

- new array methods will help
  - toSpliced()
  - toSorted()
  - toReversed()
- browser support is still not impressive (less than 90%)

Redux

Introduction

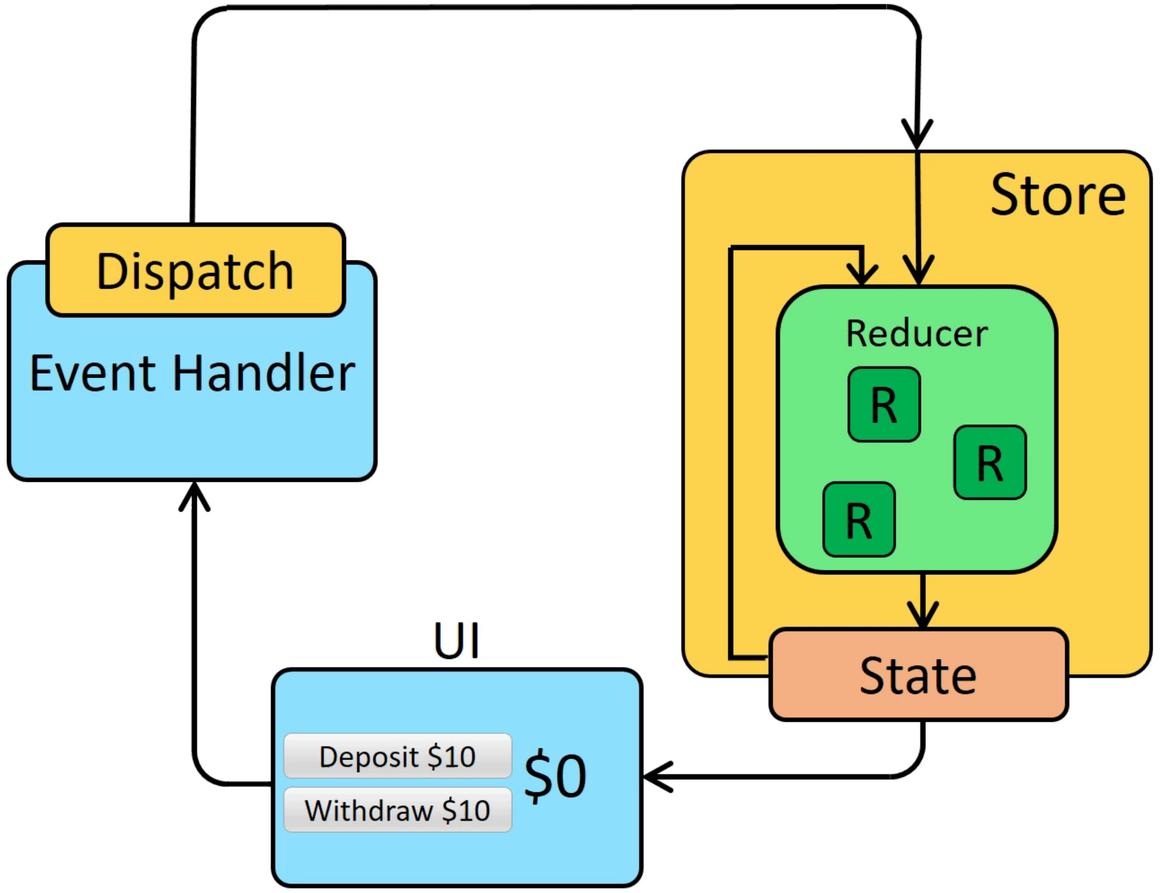
What is Redux?

- **pattern** (and a JS library) for predictable and maintainable global state management
- serves as a centralised store for state that needs to be used across entire application
- pattern includes: **store**, **actions** and **reducers**

Introduction

What is Redux?

- **store** holds state which is reflected in UI, **actions** are dispatched from event handlers that trigger **reducers** and update the state



Step 1

Create a slice with reducers

```
// counterSlice.js
export const counterSlice = createSlice({
  name: 'counter',
  initialState: {
    value: 0
  },
  reducers: {
    increment: (state) => {
      state.value += 1
    },
    decrement: (state) => {
      state.value -= 1
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload
    },
  },
})
```

Step 2

Create a store with the previously defined slice

```
// store.js
import { configureStore } from '@reduxjs/toolkit'
import { counterSlice } from '../features/counter/counterSlice'

export const store = configureStore({
  reducer: {
    counter: counterSlice.reducer,
  },
})
```

Step 3

Wrap the <App /> component with Provider and provide store

```
// index.js
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'
import App from './App'
import { store } from './app/store'
import { Provider } from 'react-redux'

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
)
```

Step 4

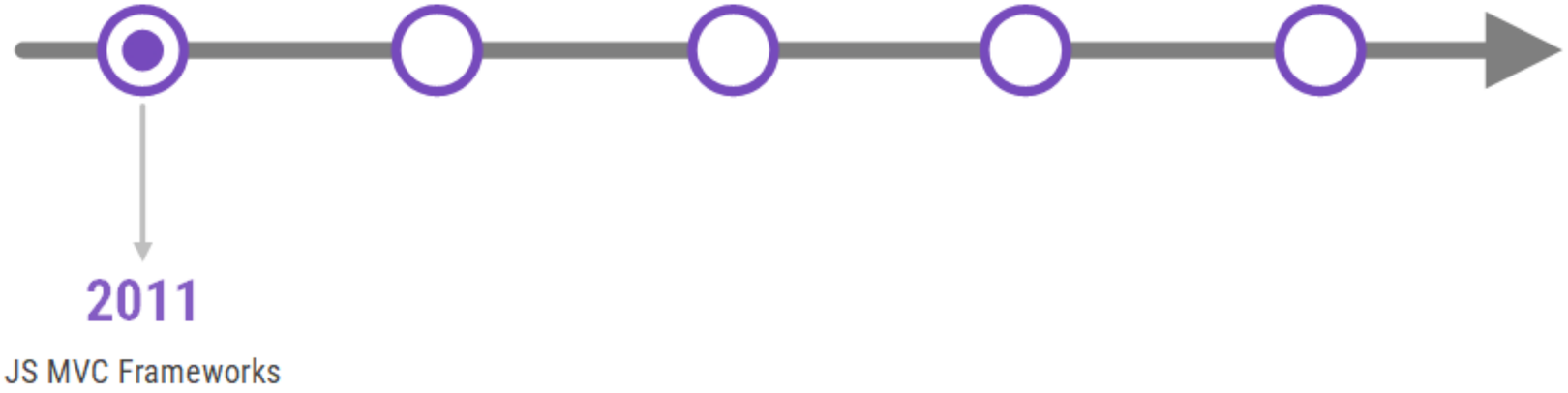
Access state using the useSelector hook or dispatch actions using useDispatch hook

```
// Counter.js
import React from 'react'
import { useSelector, useDispatch } from 'react-redux'
import { counterSlice } from './counterSlice'

export function Counter() {
  const count = useSelector((state) => state.counter.value)
  const dispatch = useDispatch()
  const { decrement, increment } = counterSlice.actions

  return (
    <div>
      <div>
        <button onClick={() => dispatch(increment())}>
          Increment
        </button>
        <span>{count}</span>
        <button onClick={() => dispatch(decrement())}>
          Decrement
        </button>
      </div>
    </div>
  )
}
```

Before we dig deep.. a history lesson.

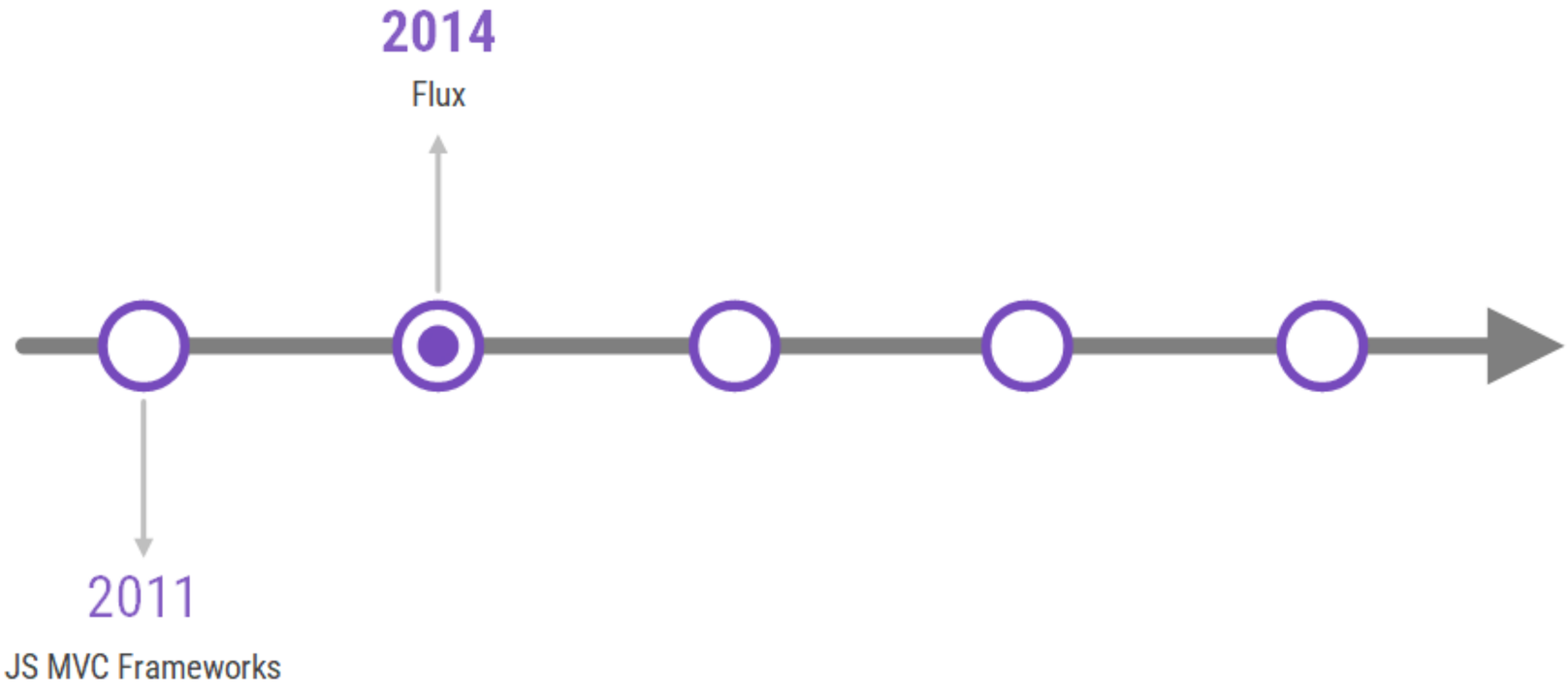
Introduction

A brief history



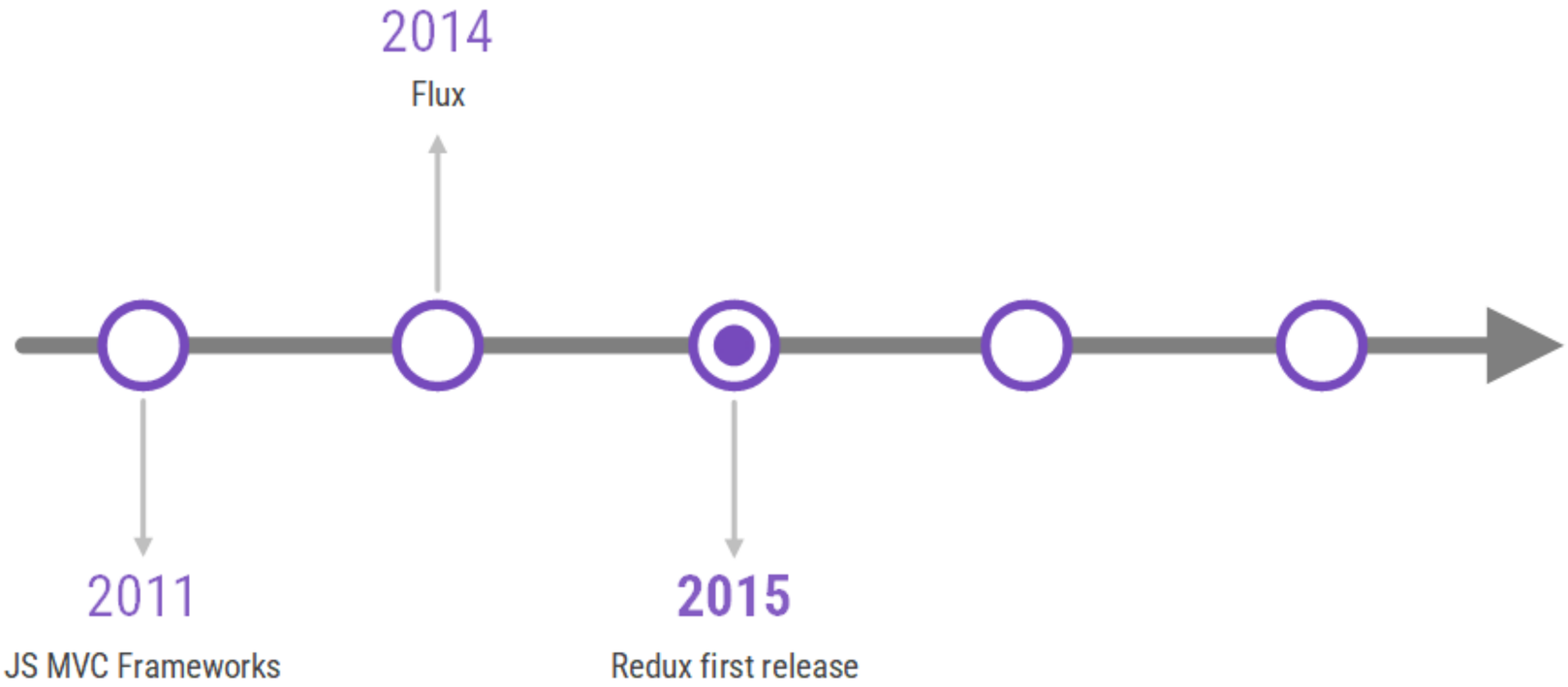
Introduction

A brief history



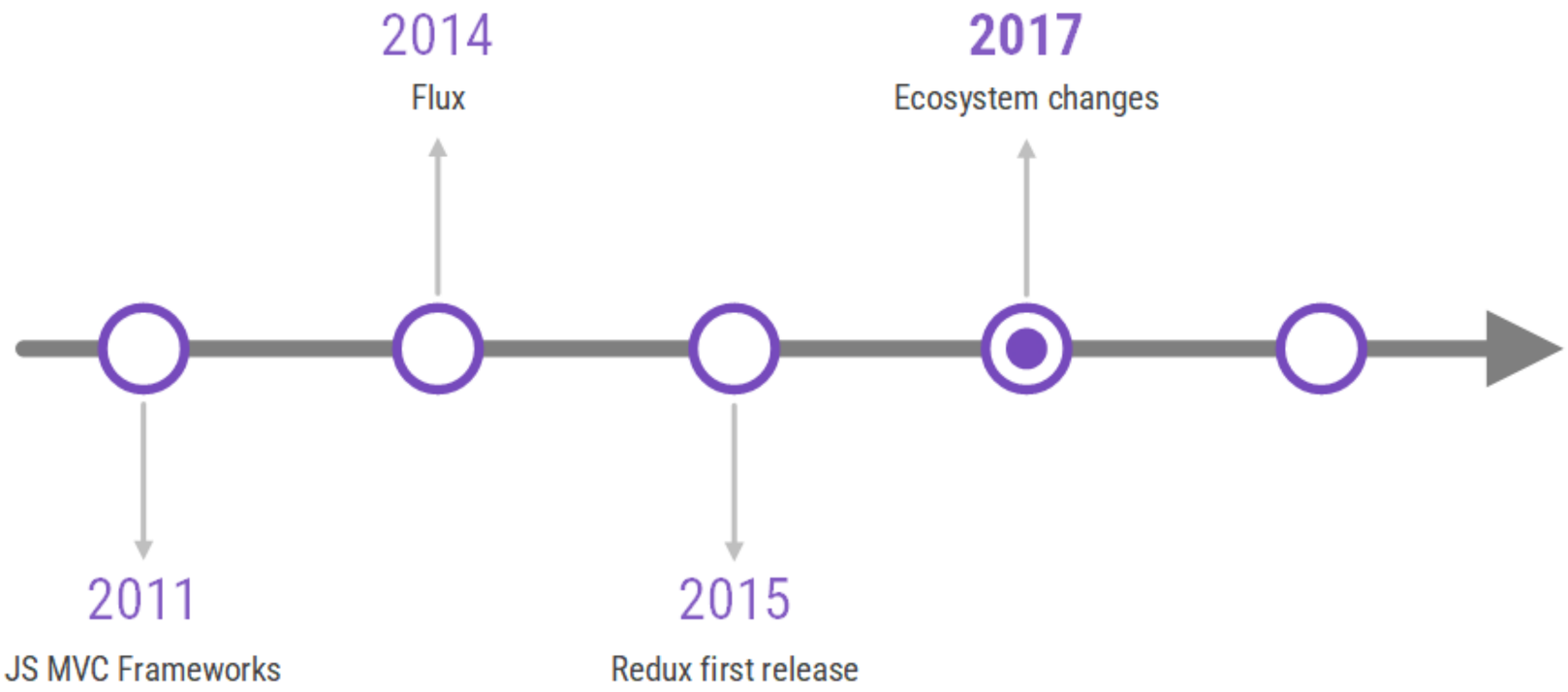
Introduction

A brief history



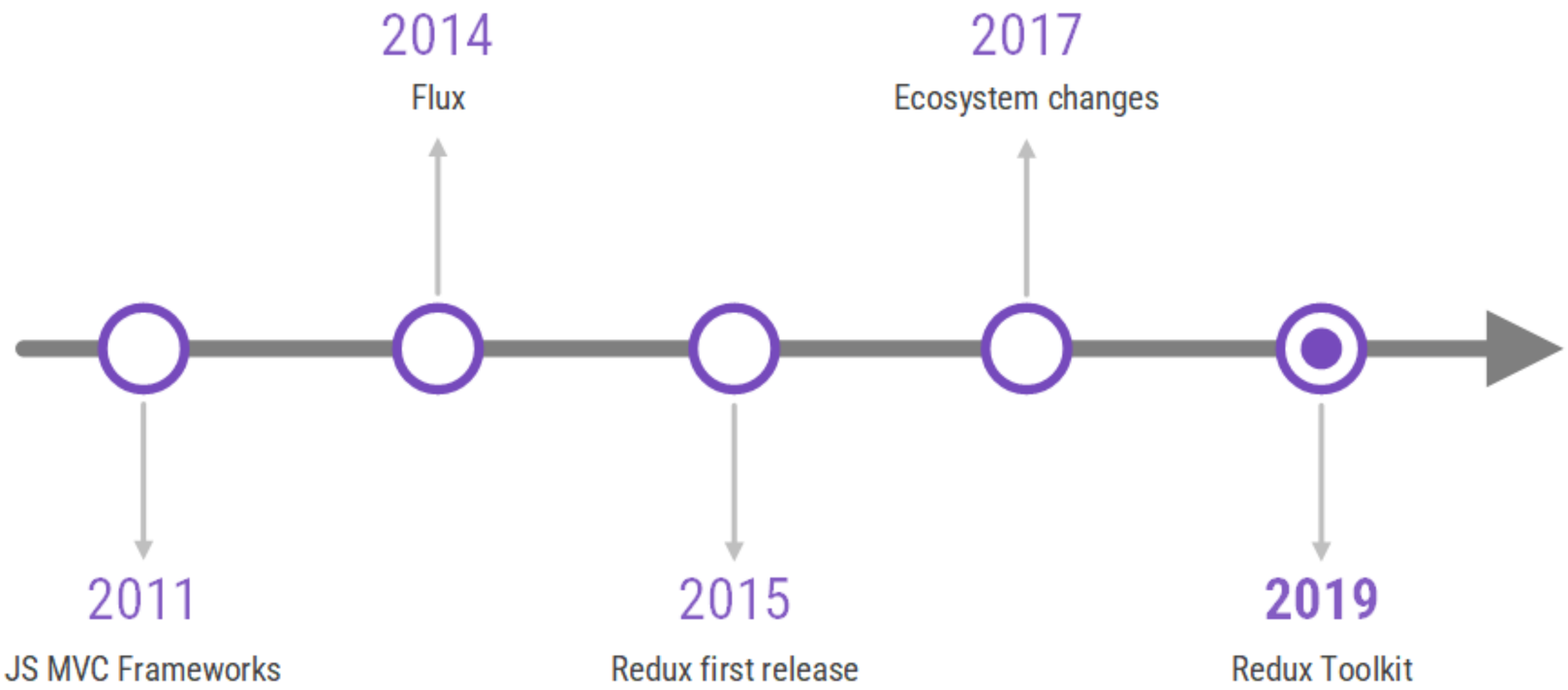
Introduction

A brief history



Introduction

A brief history



Redux Toolkit

Redux Toolkit

What is RTK?

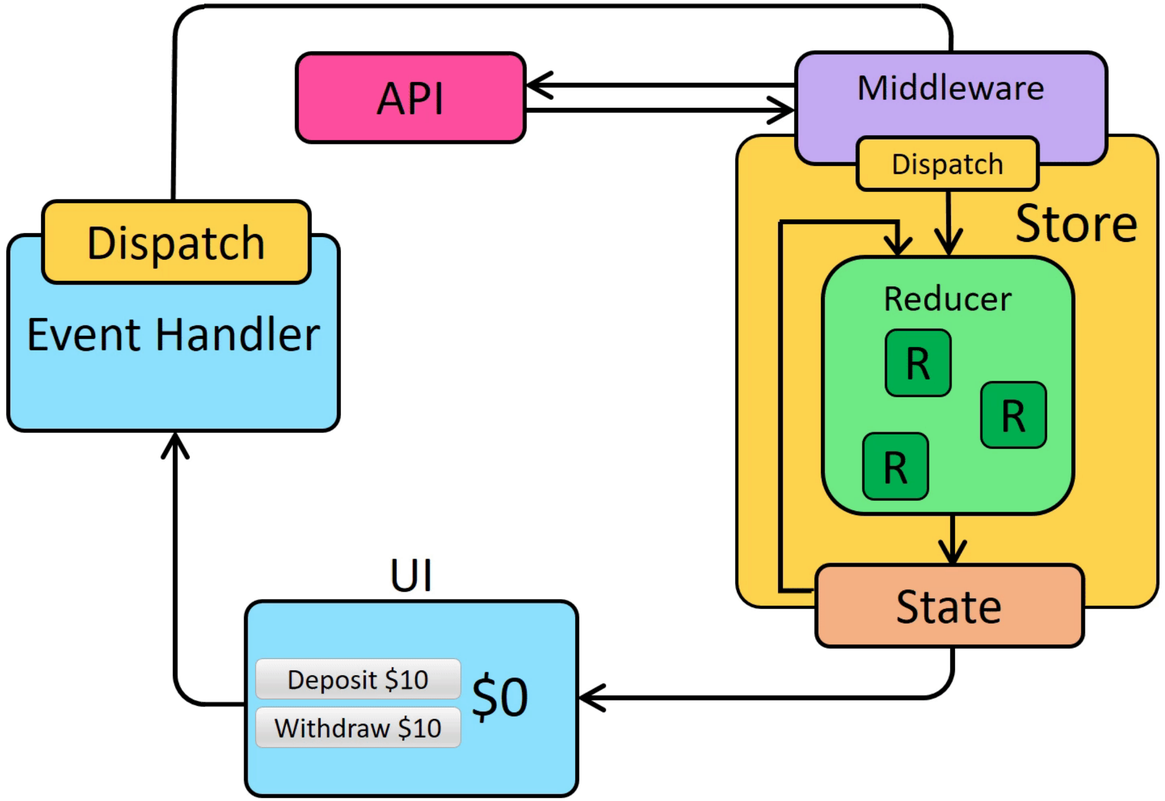
- official toolset for developing Redux applications
- batteries-included (RTK Query, thunk)
- simpler API, less boilerplate

Redux Toolkit in action

Asynchronous flow

Redux thunk

- by itself, a Redux store doesn't know anything about async logic
- any asynchronicity has to happen outside the store
- this is where Redux middleware steps in



Additional libraries

Debugging

Redux DevTools

- official tool for debugging Redux applications
- initially presented as a part of "*Time travel debugging*" presentation on React Europe 2015 by Dan Abramov
- you will most likely use it

Persistance

redux-persist

- library for persisting and rehydrating Redux stores
- can use localStorage, sessionStorage, etc.
- you will also most likely use it

Orchestration

redux-saga

- Redux side effect manager
- also a Redux middleware
- “saga” acts as a separate thread to our application

Alternatives

zustand

- based on flux pattern
- small library with simple API based on hooks
- low boilerplate

Alternatives

Jotai

- atomic approach
- minimal core API (2kb)
- TypeScript-oriented

Alternatives

MobX

- signal-based
- utilises reactive programming
- supports decorators

Conclusion

- Redux was essential before Context API
- today, most application logic can be handled using useReducer + Context API
- Redux makes sense when:
    1. multiple components need to access the same application state
    2. you are working in a large team
    3. application state is updated frequently

Thank you!