Welcome to

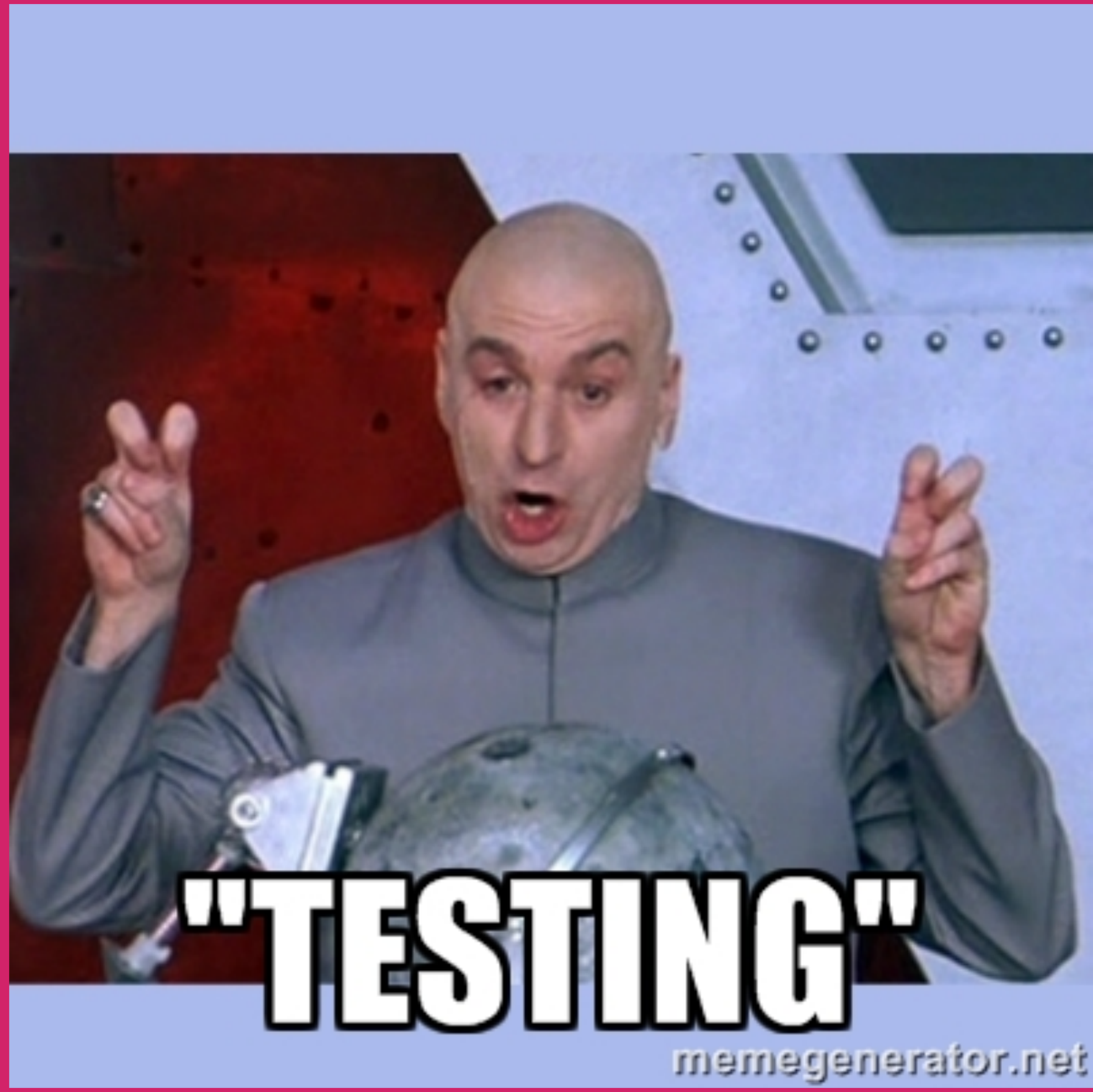
Testing Vuex

Requirements

- Basic Javascript knowledge
- Basic understanding on how Vue.js works
- Basic Javascript Testing knowledge

What we will learn

- What is Vuex
- How does Vuex work
- Testing Vuex getters
- Testing Vuex mutation
- Testing Vuex actions
- In a real world example.



But wait...

What is ... Vuex ?

Vuex is a state management pattern & framework.

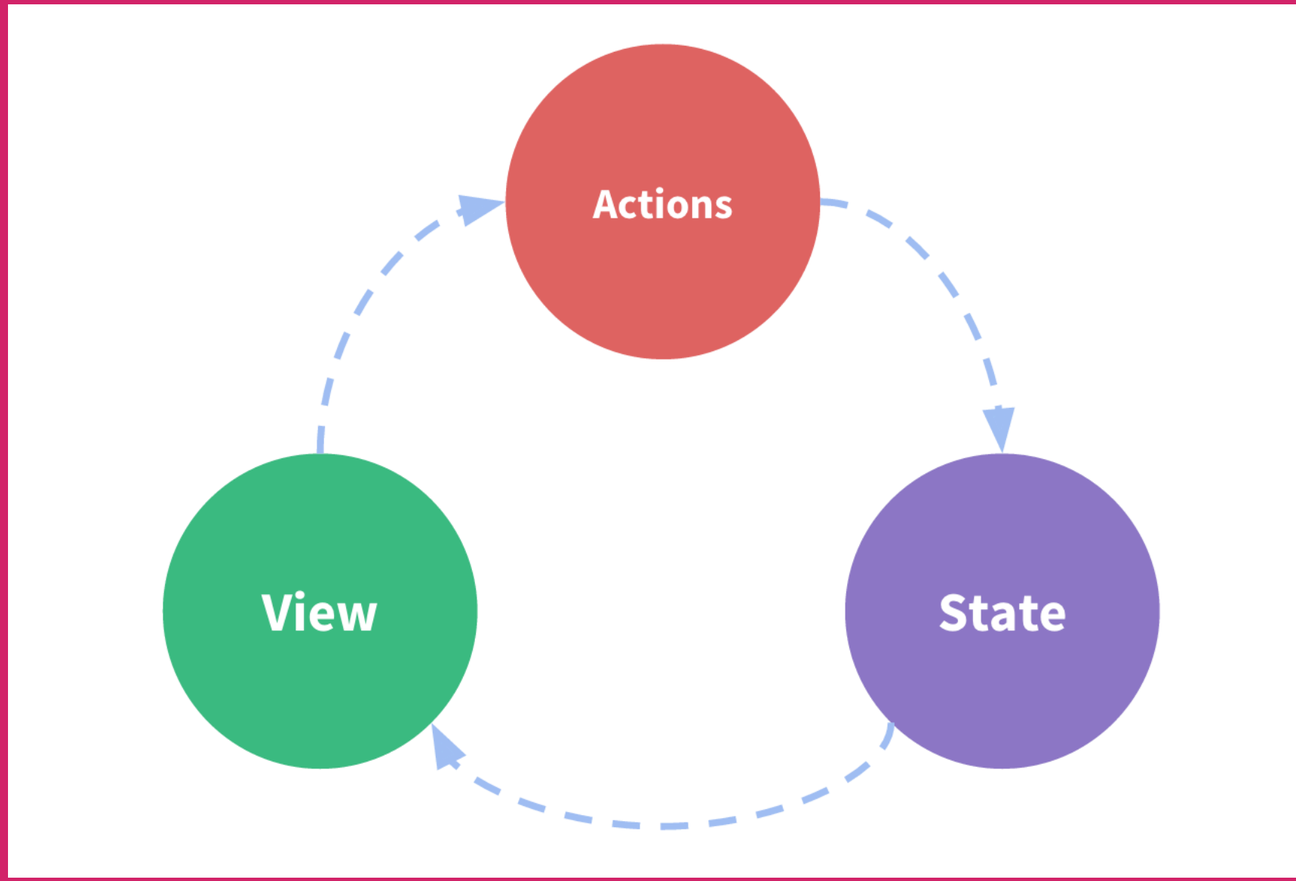
Inspired by facebooks flux pattern and redux implementation of it.

What is a "State Management Pattern" and why do we need it?

Simple Vue App

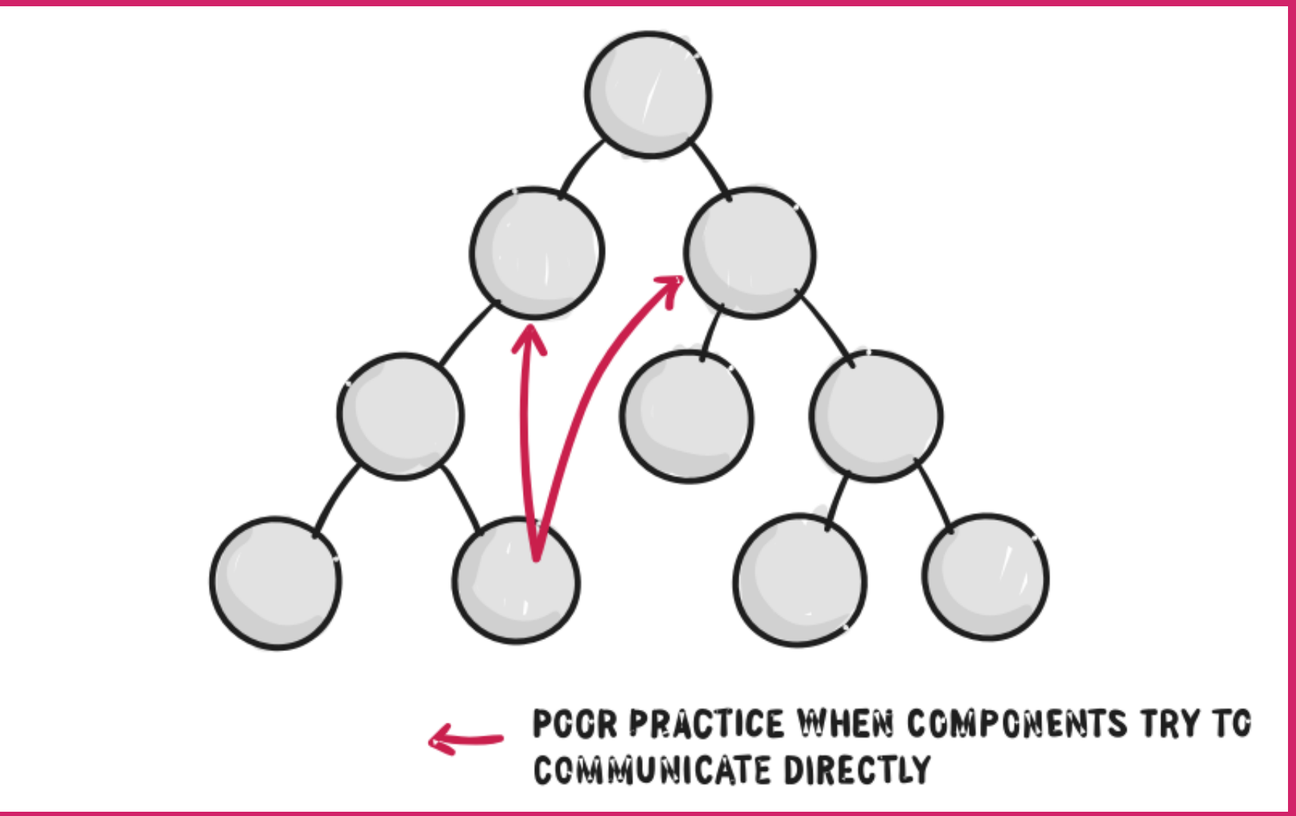
```
new Vue({
  // state
  data () {
    return {
      count: 0
    }
  },
  // view
  template: `
    <div>{{ count }}</div>
  `,
  // actions
  methods: {
    increment () {
      this.count++
    }
  }
})
```

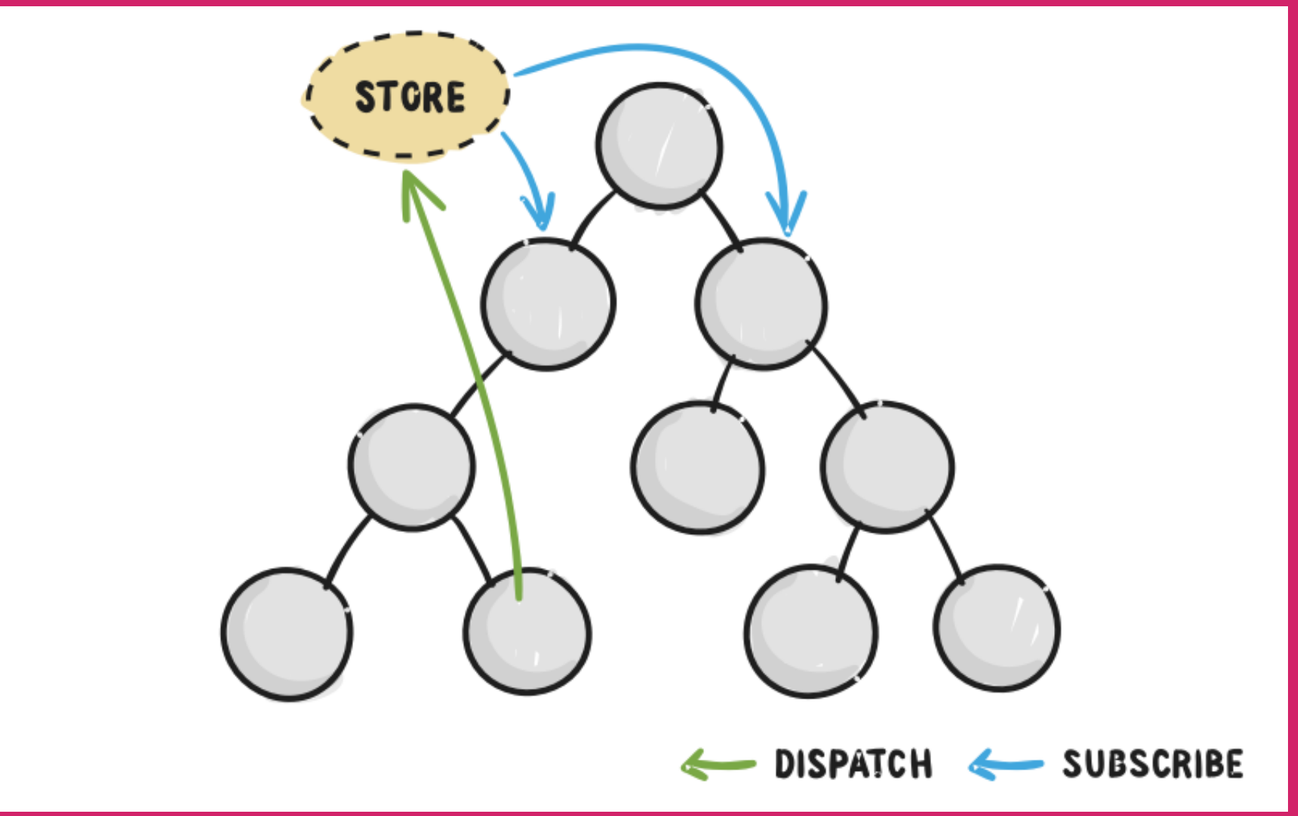
Simple one-way data flow



But ...

A complex application have different components, with child components and looks more like this.

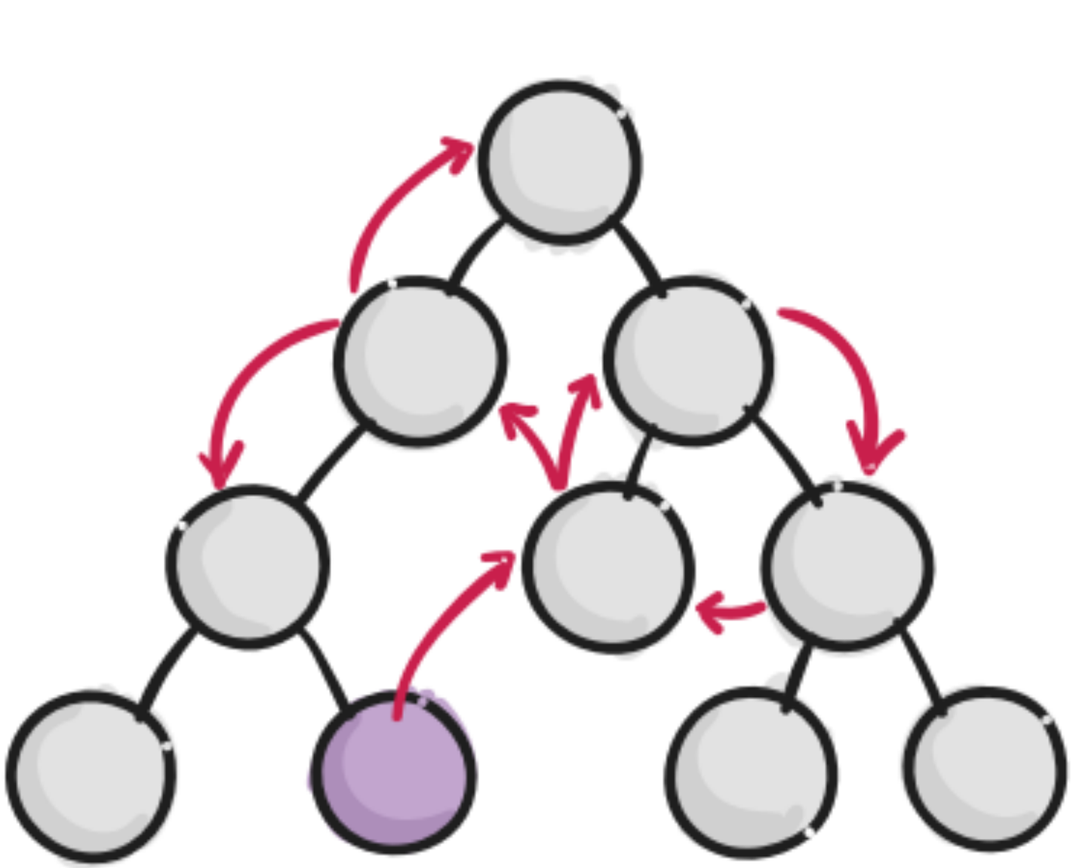


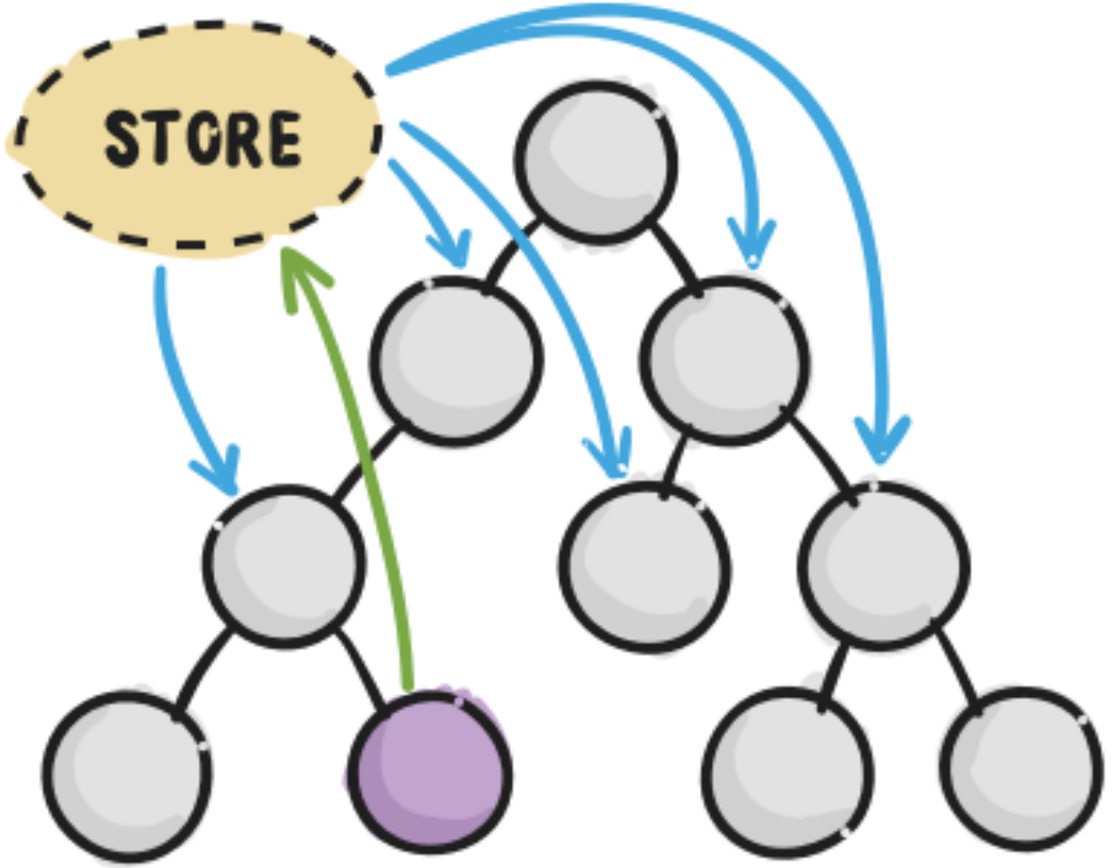


Flux Pattern for the rescue

WITHOUT REDUX

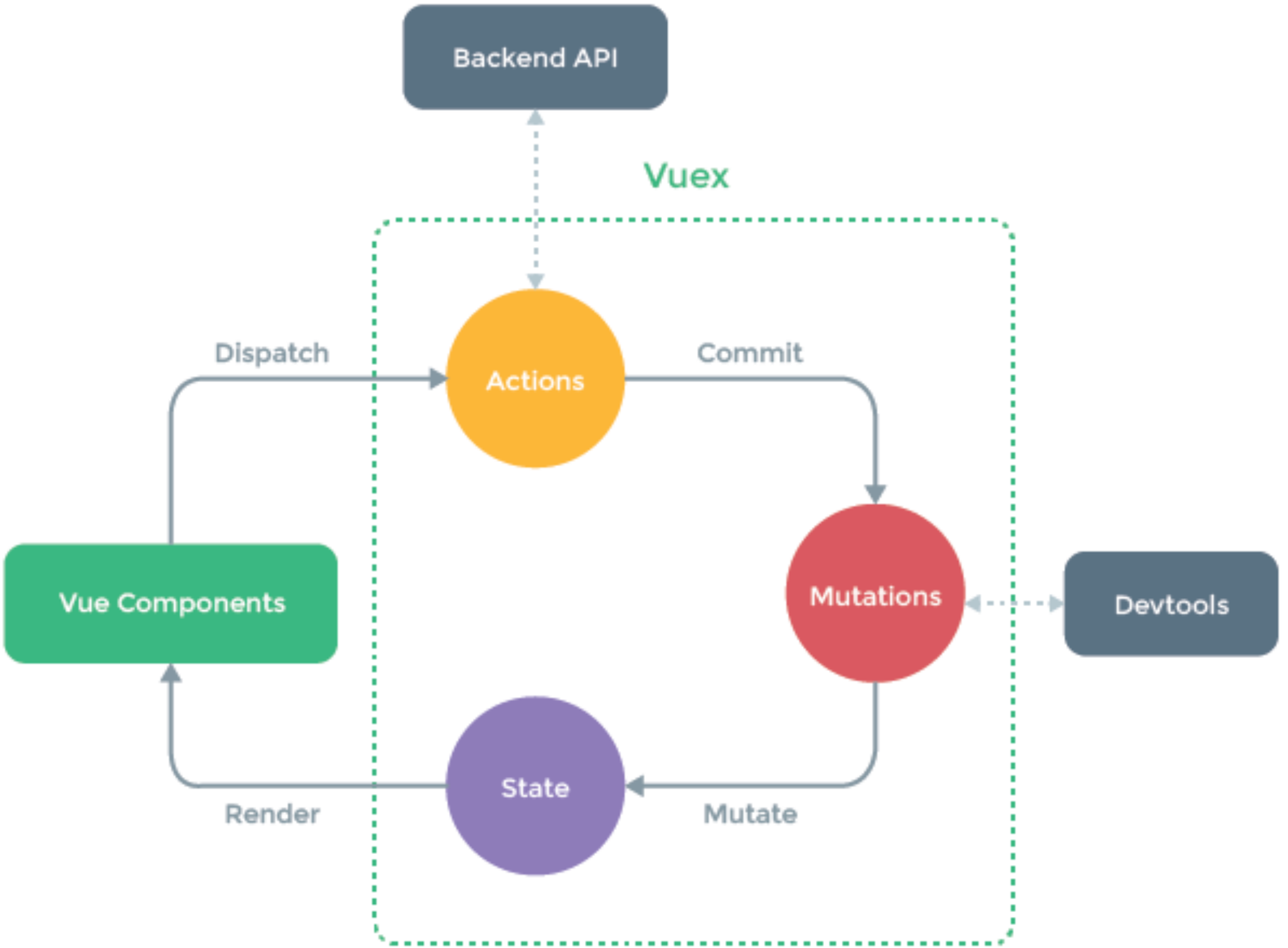
WITH REDUX





COMPONENT INITIATING CHANGE

We need this for Vue.js!


Backend API
Vuex
Dispatch
Actions
Commit
Vue Components
Mutations
Devtools
State
Render
Mutate

And this is.

Vuex

Our Test Application

https://github.com/apertureless/vuex-notes-app

Testing getters

```
// getters.js
export const activeNote = state => state.activeNote
```

```
// getters.spec.js

describe('getters', () => {
    // Mock state
    const state = {
        notes: [
                { text: 'Mock', favorite: false },
                { text: 'Mock2', favorite: true },
                { text: 'Mock3', favorite: false },
            ],
        activeNote: {}
    }


    it('notes', () => {
        // Get results with mocked state
        const result = getters.notes(state)

        // Assign results with state
        expect(result).to.deep.equal(state.notes)
    })

})
```

Testing mutations

```
// mutations.js

export const mutations = {
    addNote (state) {
        const newNote = {
            text: 'Neue Notiz',
            favorite: false
        }
        state.notes.push(newNote)
        state.activeNote = newNote
    },
    ...
}
```

```
// mutations.spec.js
import { mutations } from '../../vuex/mutations'

// destructure assign mutations
const { addNote, editNote, deleteNote } = mutations

describe('mutations', () => {
    it('addNote', () => {
        // mock state
        const state = { notes: [] }

        // apply mutation
        addNote(state)

        // assert result
        expect(state.notes).to.be.an('array')
        expect(state.notes[0].text).to.equal('Neue Notiz')
    })
})
```

Testing Actions

```
// actions.js

export const addNote = ({ commit }) => commit('addNote')
export const editNote = ({ commit }, e) => commit('editNote', e.target.value)
export const deleteNote = ({ commit }) => commit('deleteNote')
```

We need a helper function

```
const testAction = (action, args, state, expectedMutations, done) => {
    let count = 0

    // Mock Commit
    const commit = (type, payload) => {
        const mutation = expectedMutations[0]
        expect(mutation.type).to.equal(type)
        if (payload) {
            expect(mutation.payload).to.deep.equal(payload)
        }
        count ++

        if (count >= expectedMutations.length) {
            done()
        }
    }

    // call the action with mocked store and arguments
    action({ commit, state }, ...args)

    // check if no mutations should have been dispatched
    if (expectedMutations.length === 0) {
        expect(count).to.equal(0)
        done()
    }
}
```

HELPER.JS

- Mocks a vuex commit
- Checks the expected mutation type with the dispatched
- Checks payload

```
// actions.spec.js

describe('actions', () => {
    it('addNote', (done) => {
        const state = store.state
        testAction(actions.addNote, [], state, [
            { type: 'addNote' }
        ], done)
    })
    ....
```

Advanced Action Testing

Actions can be tricky, if you interact with external APIs / Services.

So we need to mock the endpoints.

Example action

```
// actions.js
import shop from '../api/shop'

export const getAllProducts = ({ dispatch }) => {
  dispatch('REQUEST_PRODUCTS')
  shop.getProducts(products => {
    dispatch('RECEIVE_PRODUCTS', products)
  })
}
```

Testing action with inject loader

```
// use require syntax for inline loaders.
// with inject-loader, this returns a module factory
// that allows us to inject mocked dependencies.

const actionsInjector = require('inject!./actions')

// create the module with our mocks
const actions = actionsInjector({
  '../api/shop': {
    getProducts (cb) {
      setTimeout(() => {
        cb([ /* mocked response */ ])
      }, 100)
    }
  }
})
```

Testing it

```
describe('actions', () => {
  it('getAllProducts', done => {
    testAction(actions.getAllProducts, [], {}, [
      { type: 'REQUEST_PRODUCTS' },
      { type: 'RECEIVE_PRODUCTS', payload: { /* mocked response */ } }
    ], done)
  })
})
```