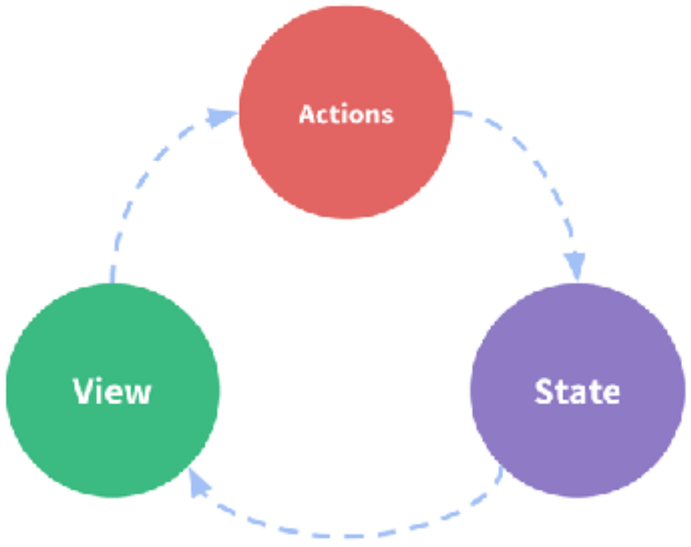
Vue.js

뷰 × 엑스

VUEX

뷰 인스턴스 (컴포넌트)

Vue 생성자를 통해 생성된 객체(컴포넌트)는 상태(State)를 통해 뷰(View)를 그리고, 액션(Actions)을 처리하여 상태 변경을 수행, 뷰를 업데이트 한다.




```
new Vue({
  // 상태
  data () {
    return {
      count: 0
    }
  },
  // 뷰
  template: `
    <div>{{ count }}</div>
  `,
  // 액션
  methods: {
    increment () {
      this.count++
    }
  }
})
```

앱을 구동하는 데이터

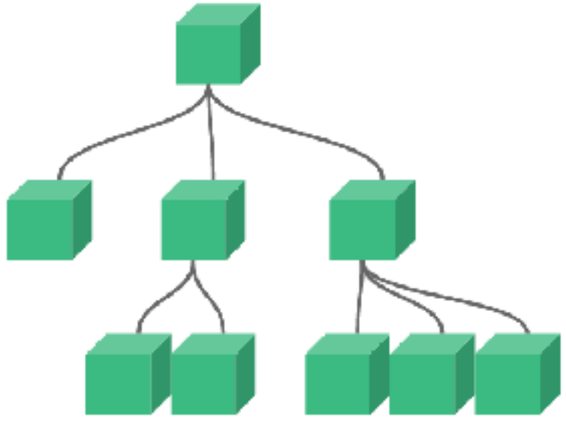
상태의 선언적 매핑

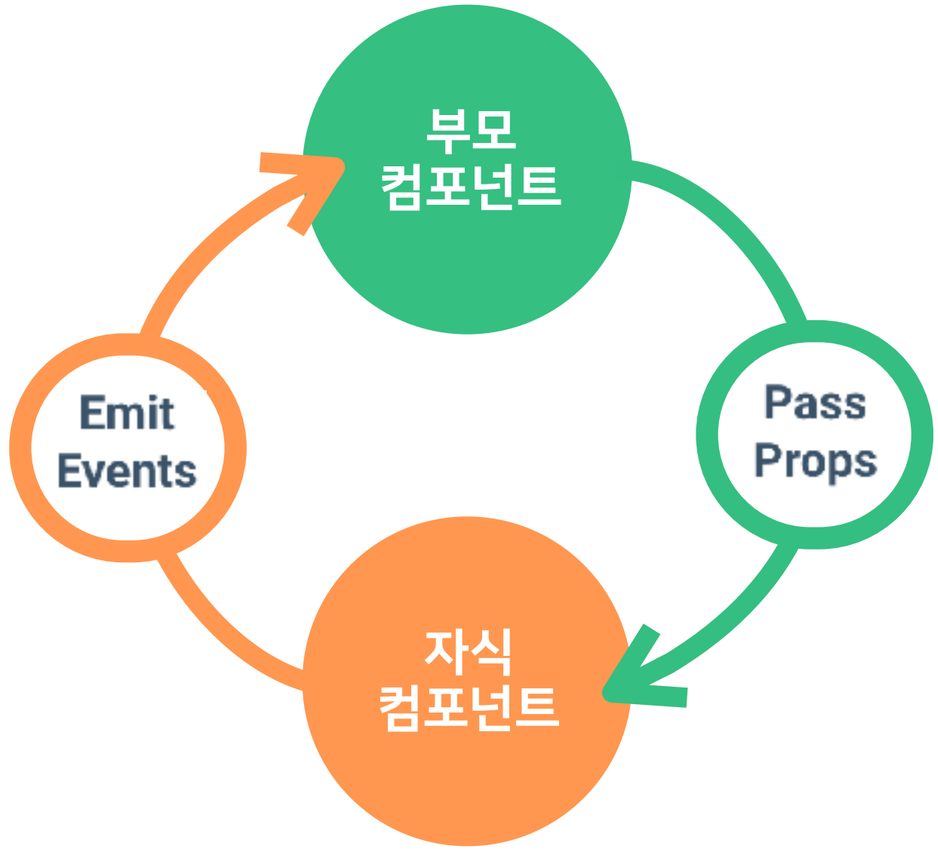
뷰 를 통해 사용자 입력에 반응하여 상태를 바꾸는 메서드

뷰 컴포넌트 간 통신 (상태 공유)

Vue 프레임워크는 컴포넌트들로 구성되며, 컴포넌트 간에 관계 모델(부모/자식/형제)을 가진다. 관계가 형성된 컴포넌트 간에는 통신을 통해 상태(State)를 공유할 수 있다.

하지만 형제 관계(비 부모/자식 관계)에서는 상태 공유를 할 수 없다.





상태 관리가 필요한 이유

공통의 상태를 공유하는 여러 컴포넌트 가 있는 경우 단순함이 빠르게 저하

- 여러 뷰 컴포넌트 는 같은 상태에 의존하는 경우가 발생
- 각 컴포넌트는 동일한 상태를 반영해야 함

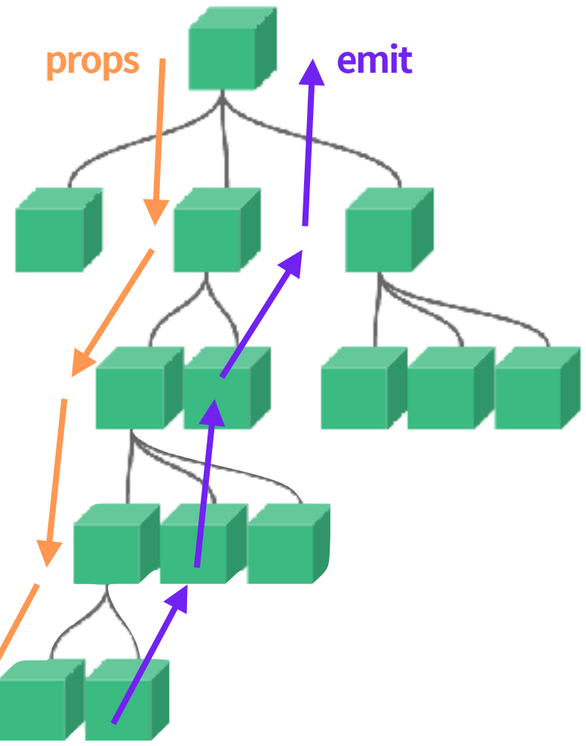
여러 뷰 컴포넌트 는 같은 상태에 의존하는 경우가 발생

지나치게 중첩된 컴포넌트는 부모에서 전달되는 prop 속성이 장황해질 수 있으며, 형제 컴포넌트에서는 작동하지 않는다.

각 컴포넌트는 동일한 상태를 반영해야 함

직접 부모/자식 인스턴스를 참조하거나 이벤트를 통해 상태의 여러 복사본을 변경 및 동기화 하려는 등 해결 방법을 모색해야 함.

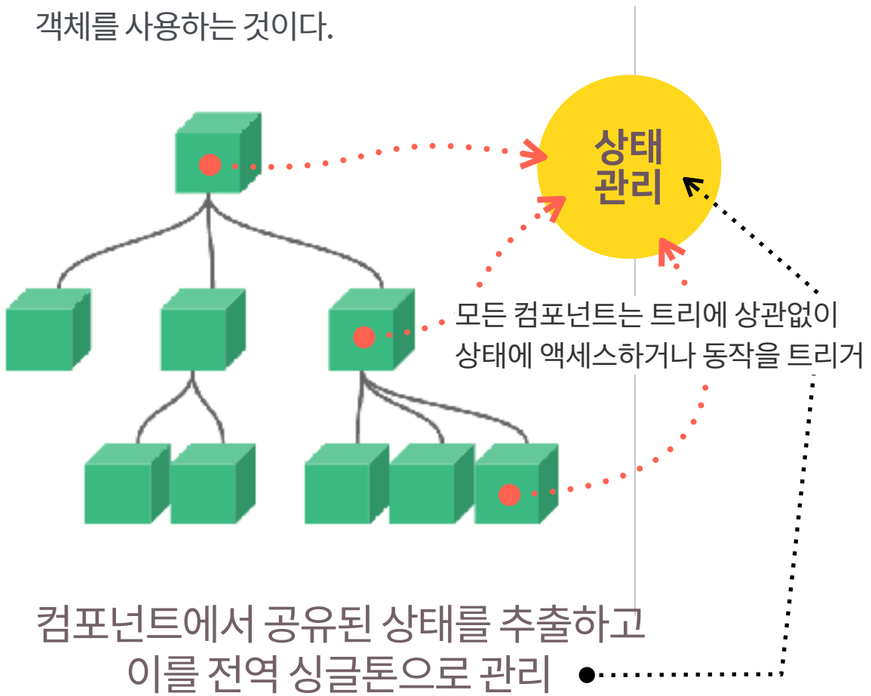
이러한 패턴은 모두 부서지기 쉽고 유지보수가 불가능한 코드로 빠르게 변경되는 문제 발생.



글로벌 이벤트 버스(Global Event Bus)

뷰 컴포넌트 상태 관리는 단순하지만, 공통의 상태를 공유하는 여러 컴포넌트가 있는 경우 단순함이 빠르게 저하 된다.

부모/자식 관계가 아닌 컴포넌트 간 통신을 통해 상태를 관리하기 위한 가장 손쉽고 견고한 방법은 상태를 관리하는 객체를 사용하는 것이다.



비 부모-자식간 통신

때로는 두 컴포넌트가 서로 통신 할 필요가 있지만 서로 부모/자식이 아닐 수도 있습니다. 간단한 시나리오에서는 비어있는 Vue 인스턴스를 중앙 이벤트 버스로 사용할 수 있습니다.

```
JS
var bus = new Vue()
```

```
JS
// 컴포넌트 A의 메소드
bus.$emit('id-selected', 1)
```

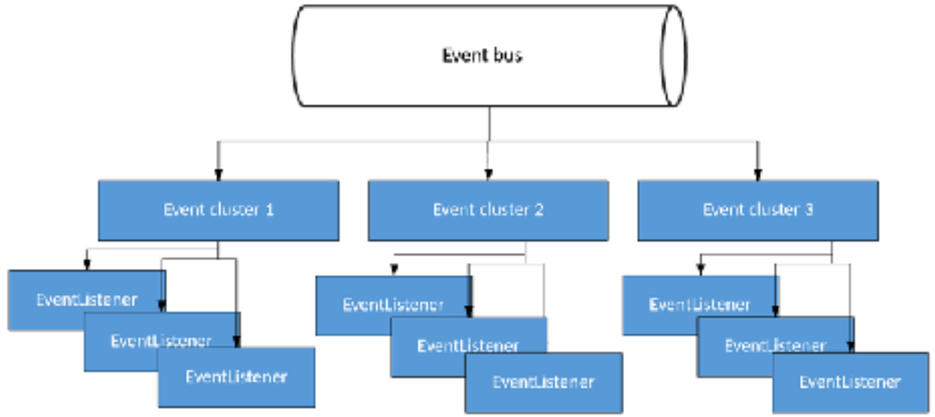
```
JS
// 컴포넌트 B의 created 훅
bus.$on('id-selected', function (id) {
  // ...
})
```

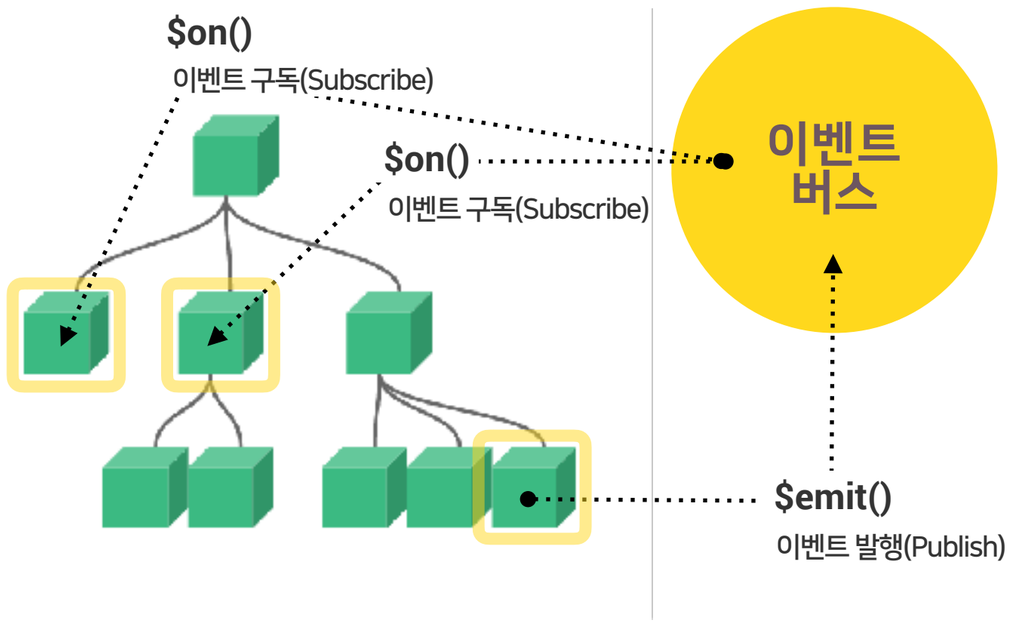
보다 복잡한 경우에는 전용 상태 관리 패턴을 고려해야합니다

글로벌 이벤트 버스 관리 한계

1개의 버스(Bus)를 사용하기 때문에 컴포넌트 간 관계를 관리하려면 금새 복잡해지게 될 것이다.

뿐만 아니라, 변경(변이)을 관찰/추적하는 기능이 없어 디버깅이 어렵다.

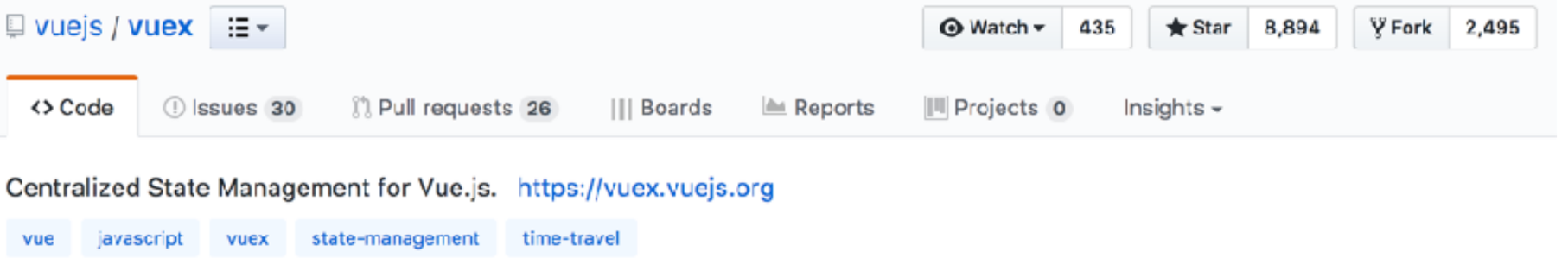


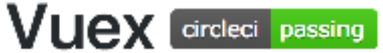


Vuex

애플리케이션 상태 관리 패턴 라이브러리

애플리케이션을 구성하는 모든 컴포넌트가 참조 가능한
상태를 중앙에서 관리하는 저장소



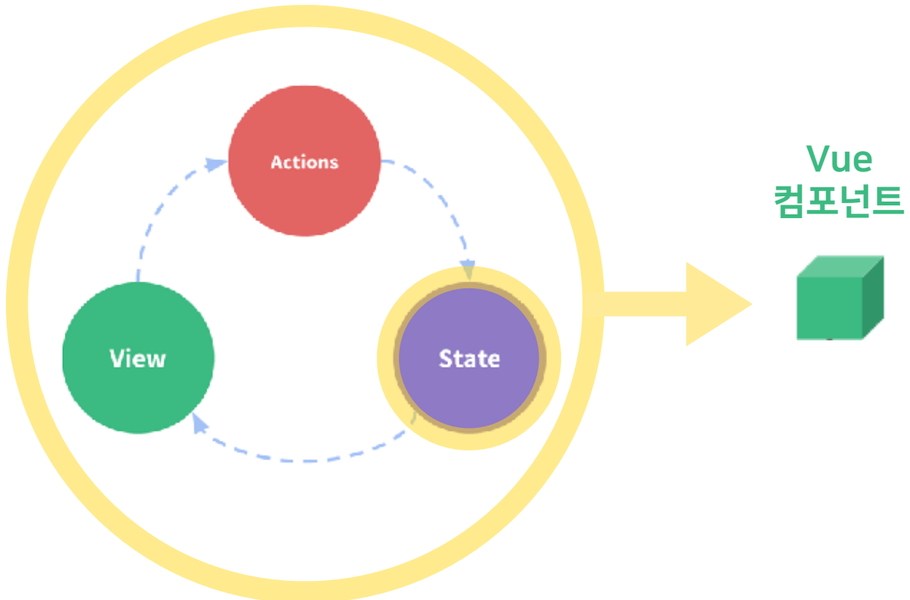


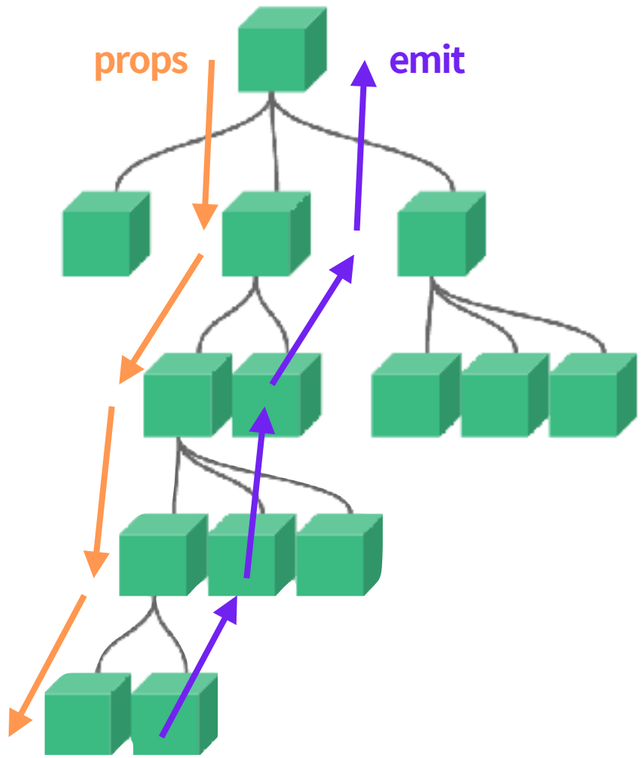
> Centralized State Management for Vue.js.

https://github.com/vuejs/vuex/

대규모 애플리케이션 상태 관리의 복잡함

대규모 애플리케이션은 여러 컴포넌트에 분산되어있는
여러 상태와 그 상호 작용으로 인해 복잡해진다.





간단한 상태 관리를 위한 글로벌 객체

Vue 애플리케이션에서 가장 근본이 되는 것은 data 객체로,
Vue 인스턴스는 단순히 그것에 대한 액세스를 프록시를 제공.
여러 인스턴스에서 공유해야하는 상태가 있으면
ID로 간단히 공유 가능.

```
const sourceOfTruth = {}

const vmA = new Vue({
  data: sourceOfTruth
})

const vmB = new Vue({
  data: sourceOfTruth
})
```

상태 관리

상태가 변경되면, 상태를 공유한 Vue 컴포넌트는
각각의 뷰를 업데이트한다. 간단한 애플리케이션이라면
이와 같은 방법이 문제가 되진 않는다.

하지만 애플리케이션이 복잡해 졌을 때는 이야기가 달라진다.
코드 관리가 어려워지는 문제가 발생하게 되기 때문.

이러한 문제를 해결하기 위해 Store 패턴을 사용한다.

스토어(Store, 데이터 기억 저장장치) 패턴

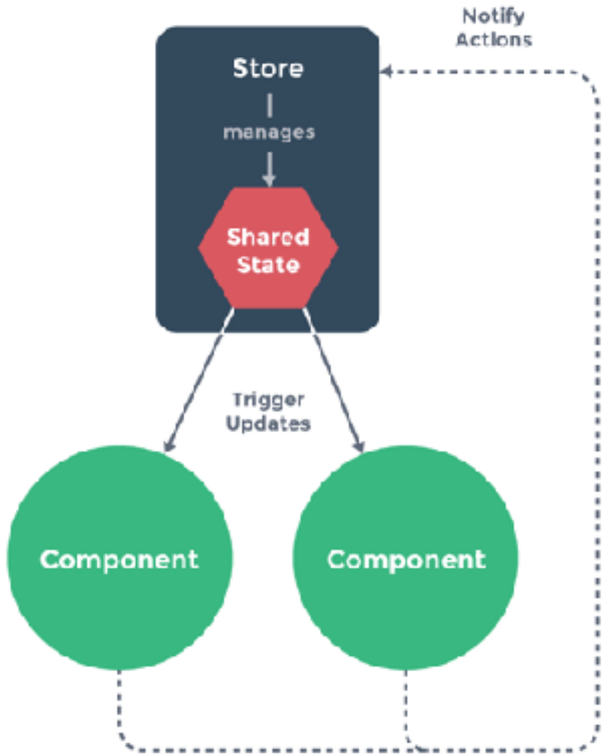
스토어 객체 내부에는 데이터(State)와 데이터를 관리하는 메서드(Actions)로 구성된다. 디버깅을 위한 debug 속성 또한 가진다. 이러한 유형의 중앙 집중식 상태 관리는 어떤 유형의 변이가 발생할 수 있는지, 어떻게 유발 되는지를 보다 쉽게 이해할 수있게 한다.

```
var store = {
  debug: true,
  state: {
    message: 'Hello!'
  },
  setMessageAction (newValue) {
    this.debug && console.log('setMessageAction triggered with', newValue)
    this.state.message = newValue
  },
  clearMessageAction () {
    this.debug && console.log('clearMessageAction triggered')
    this.state.message = ''
  }
}
```

스토어(Store, 데이터 기억 저장장치) 패턴

컴포넌트가 스토어에 속한 상태를 직접 변이 시킬 수 없지만, 스토어에 조작을 수행하도록 알리는 이벤트를 보내야하는 컨벤션(규칙)을 계속 개발할 때 결국 Flux 아키텍처에 다다르게 된다.
이 컨벤션의 이점은 스토어에서 발생하는 모든 상태 변이를 기록하고, 스냅 샷 및 히스토리 되돌리기와 같은 고급 디버깅 도우미를 구현할 수 있다.

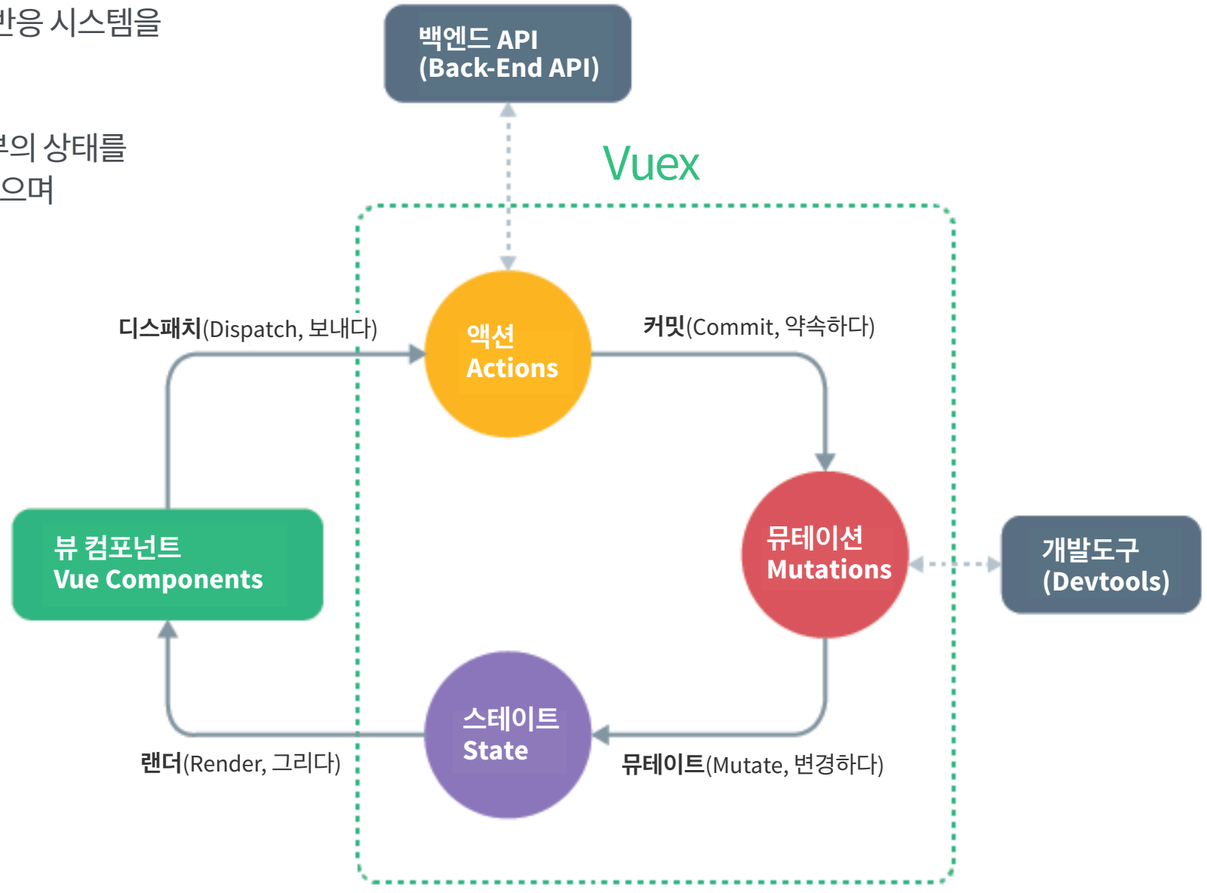
```
var vmA = new Vue({
  data: {
    privateState: {},
    sharedState: store.state
  }
})

var vmB = new Vue({
  data: {
    privateState: {},
    sharedState: store.state
  }
})
```



Vuex 는 상태 관리 패턴을 사용하는 매니저

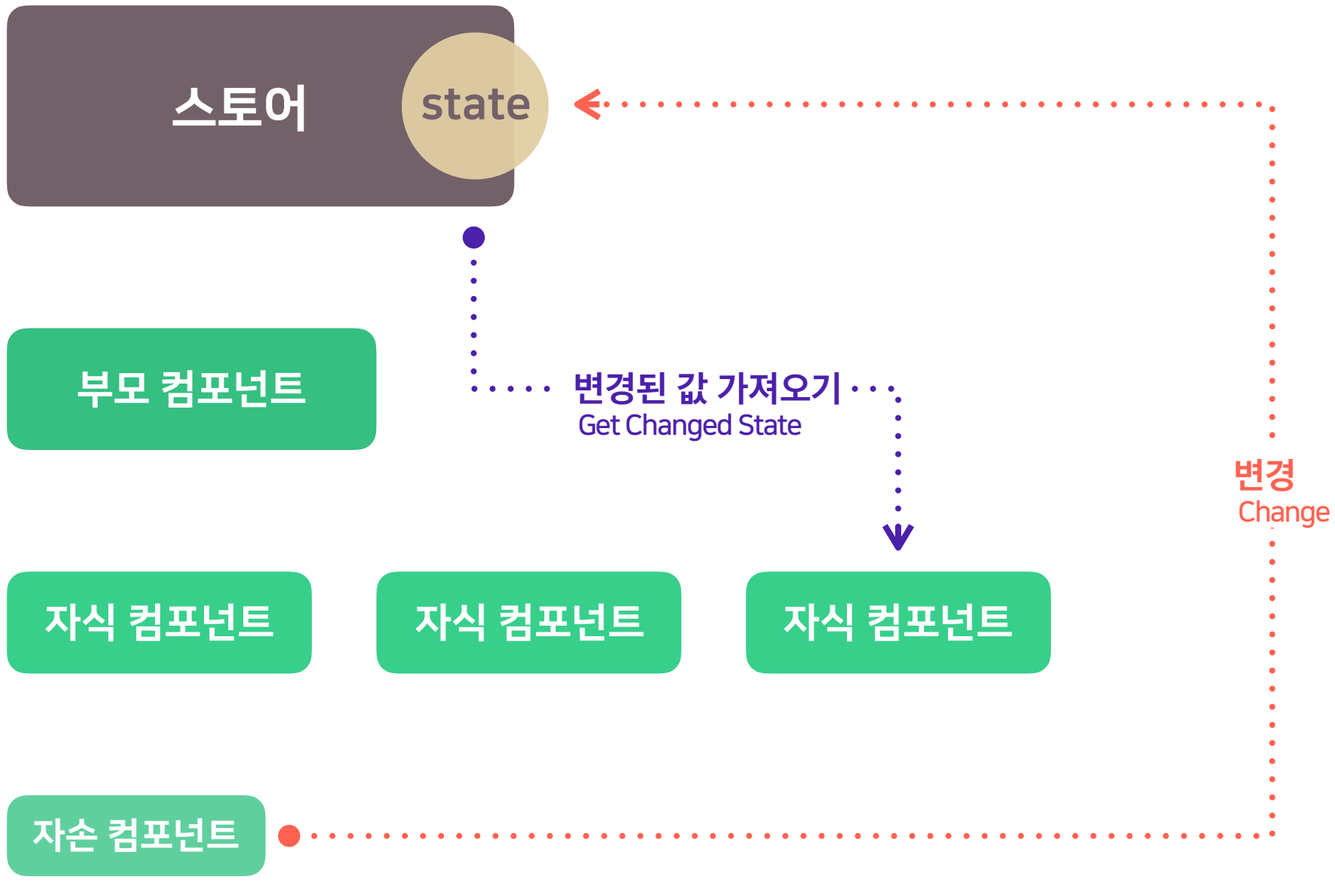
Vuex는 Vue.js가 효율적인 업데이트를 위해 세분화된 반응 시스템을 활용하도록 특별히 고안된 라이브러리

중대형 규모의 SPA를 구축하는 경우 Vue컴포넌트 외부의 상태를 보다 잘 처리할 수 있는 방법을 생각하게 될 가능성이 있으며 Vuex는 자연스럽게 선택할 수 있는 단계가 될 것이다.

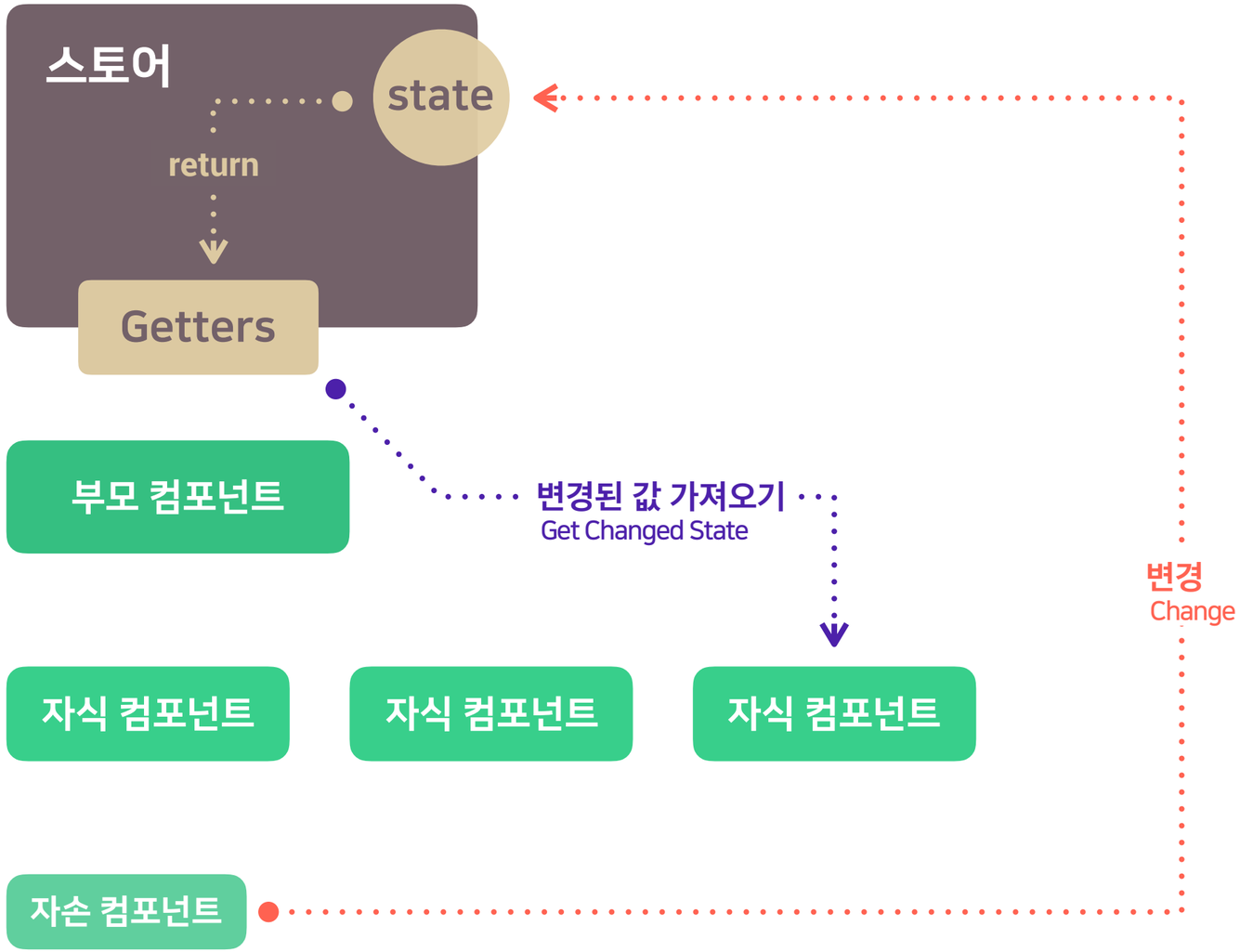


중앙 저장소 관리

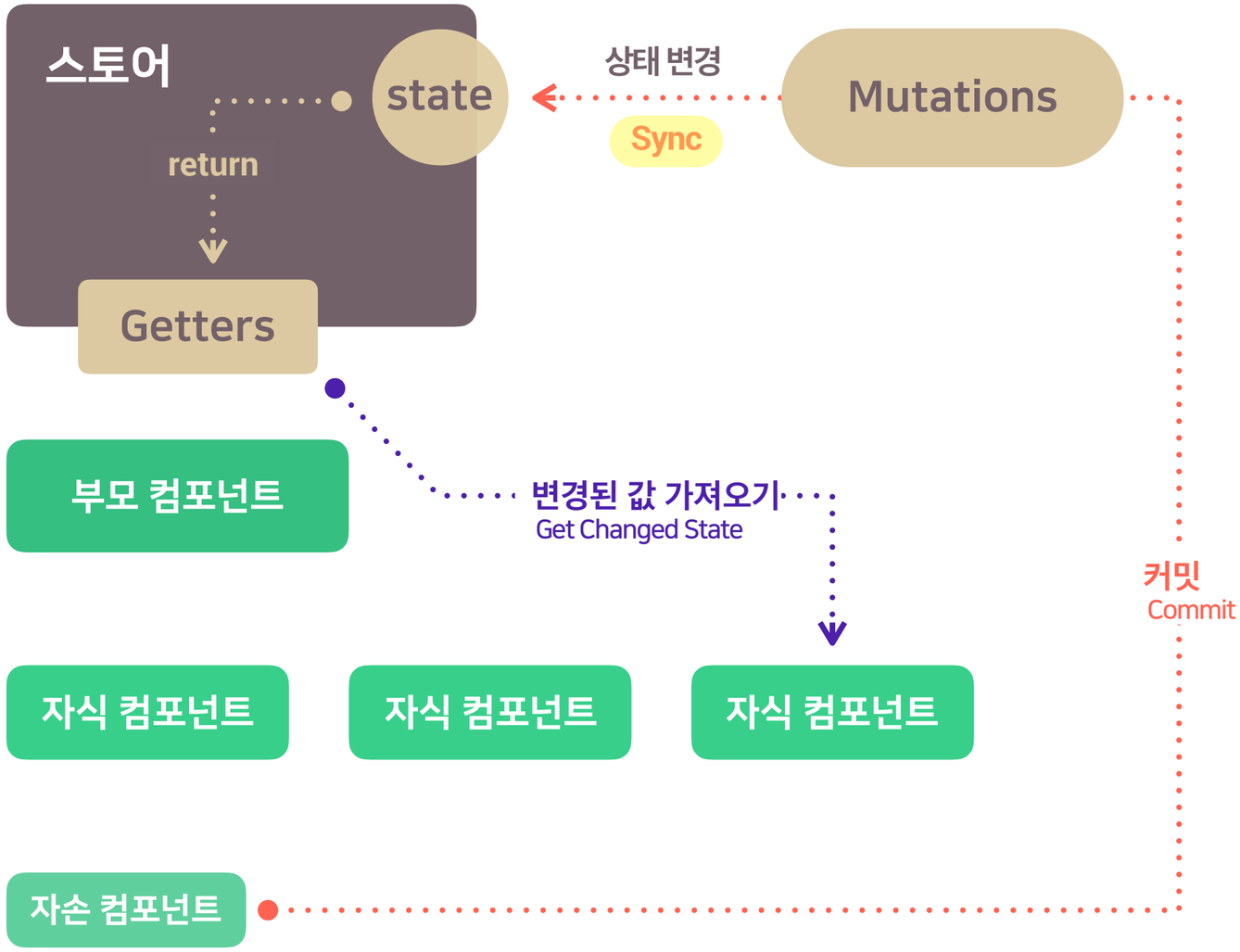
상태(state)



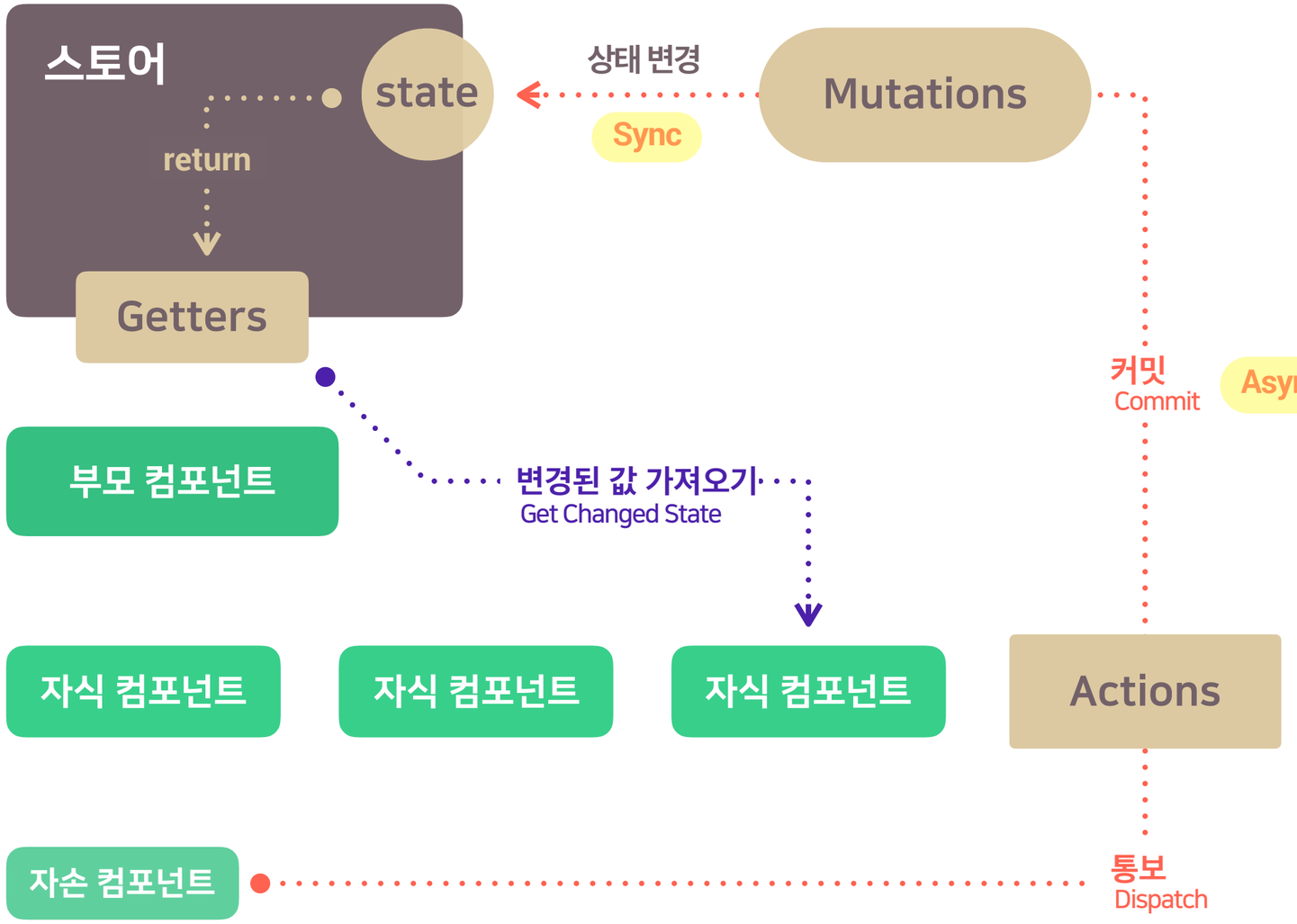
중앙 저장소 관리
가져오기(Getters)



중앙 저장소 관리
변경(Mutations)



중앙 저장소 관리
액션(Actions)



EDIT THIS PAGE

Vuex가 무엇인가요?

Vuex는 Vue.js 애플리케이션에 대한 **상태 관리 패턴 + 라이브러리** 입니다. 애플리케이션의 모든 컴포넌트에 대한 중앙 집중식 저장소 역할을 하며 예측 가능한 방식으로 상태를 변경할 수 있습니다. 또한 Vue의 공식 **devtools 확장 프로그램**과 통합되어 설정 시간이 필요 없는 디버깅 및 상태 스냅 샷 내보내기/가져오기와 같은 고급 기능을 제공합니다.

"상태 관리 패턴"이란 무엇인가요?

간단한 Vue 카운터 앱부터 시작 해보겠습니다.

```js
new Vue({
  // 상태
  data () {
    return {
      count: 0
    }
  },
  // 뷰
  template: `
    <div>{{ count }}</div>
  `,
  // 액션
  methods: {
    increment () {
      this.count++
    }
  }
})
```