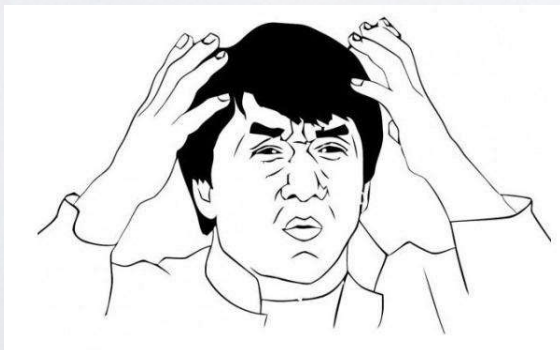
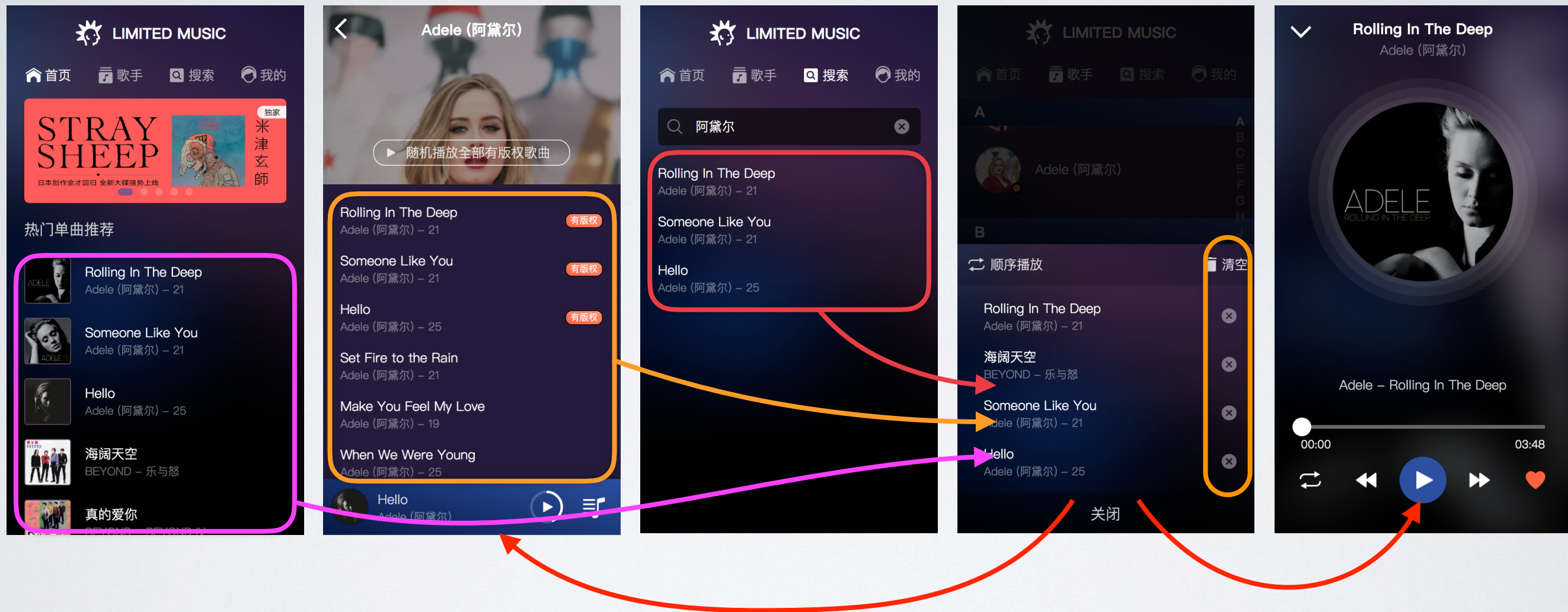


《Vue进阶》第一单元

Vuex 状态管理



存在的问题

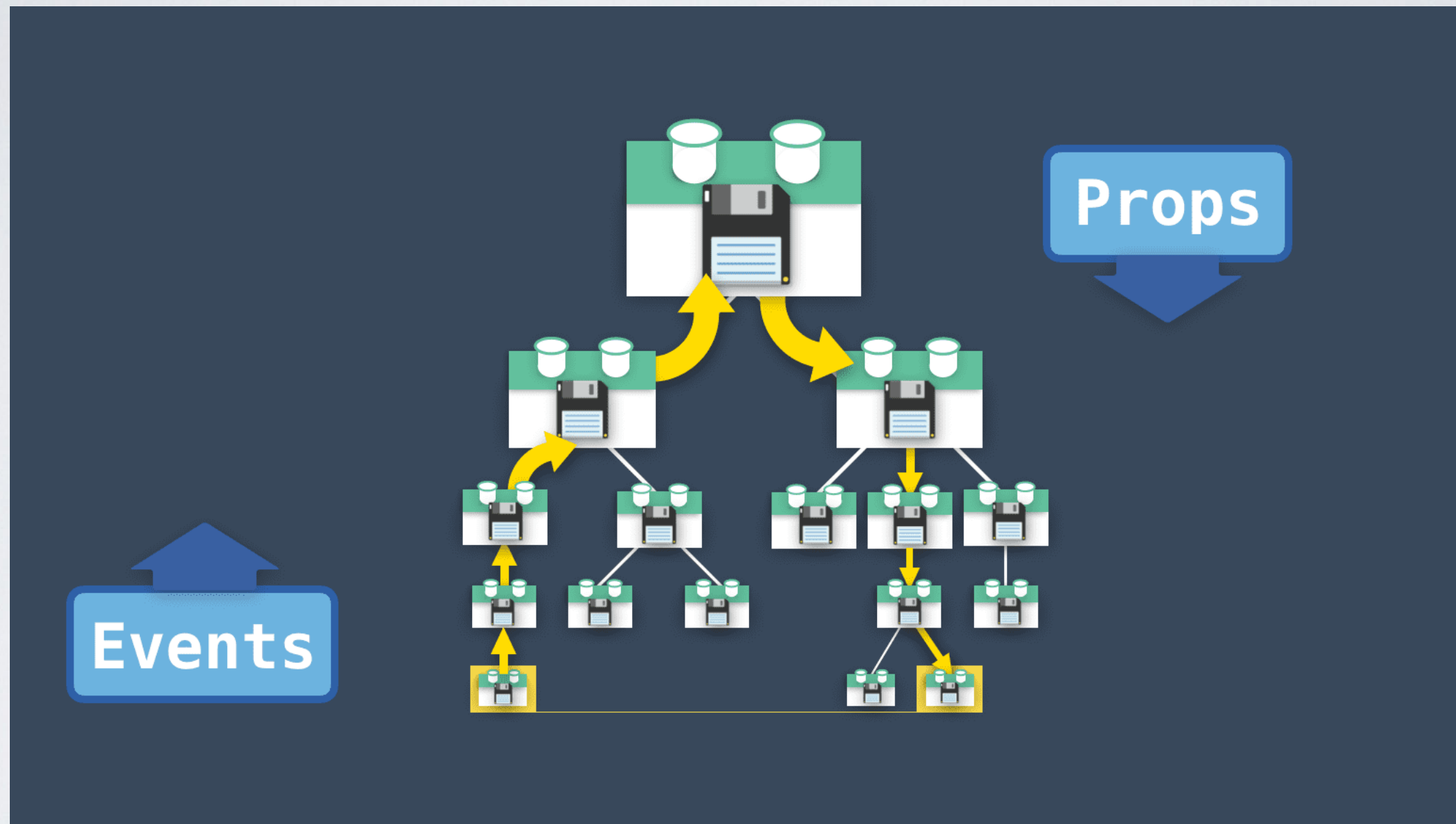


- 多处使用同一数据
- 多处都需要修改同一数据

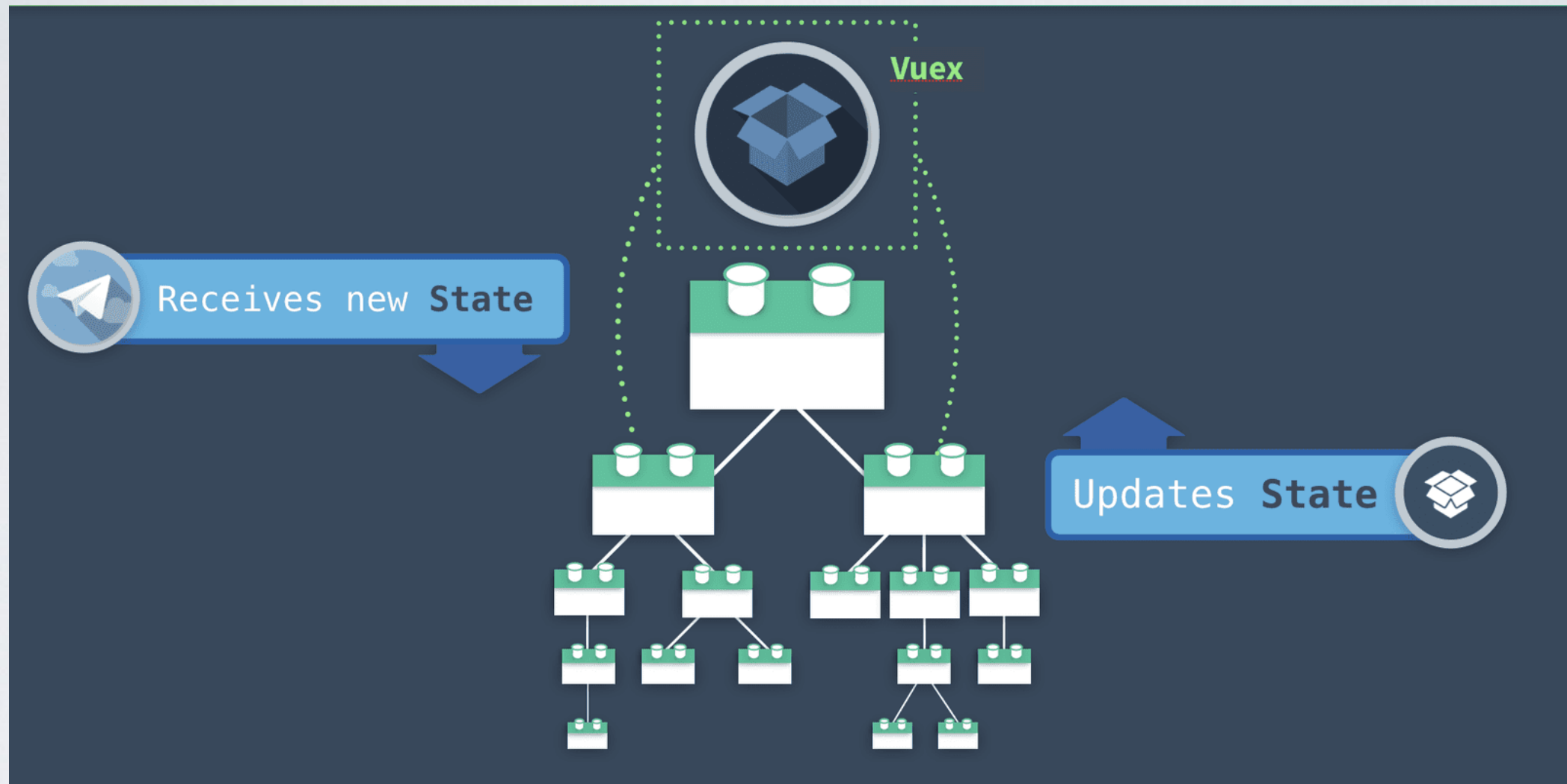
存在的问题

- 数据源 / view 使用数据 / action 变更数据
- 多层嵌套组件 / 兄弟组件之间传参繁琐易错
- 同一份数据会在不同组件中存在多份拷贝，
不易维护、会产生冲突

存在的问题

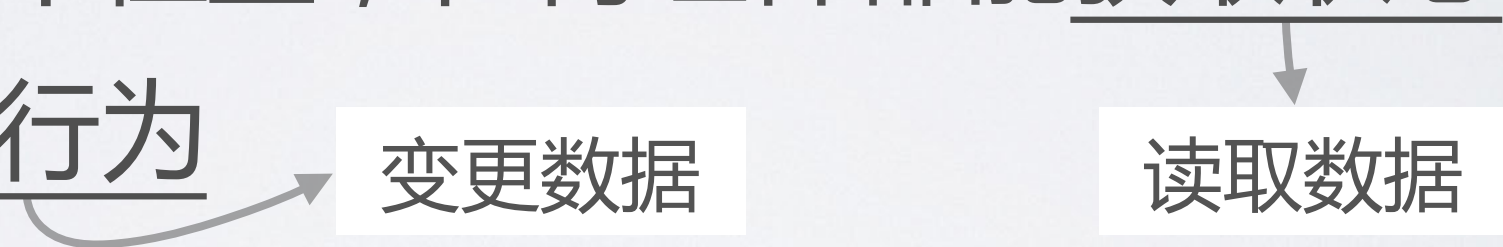


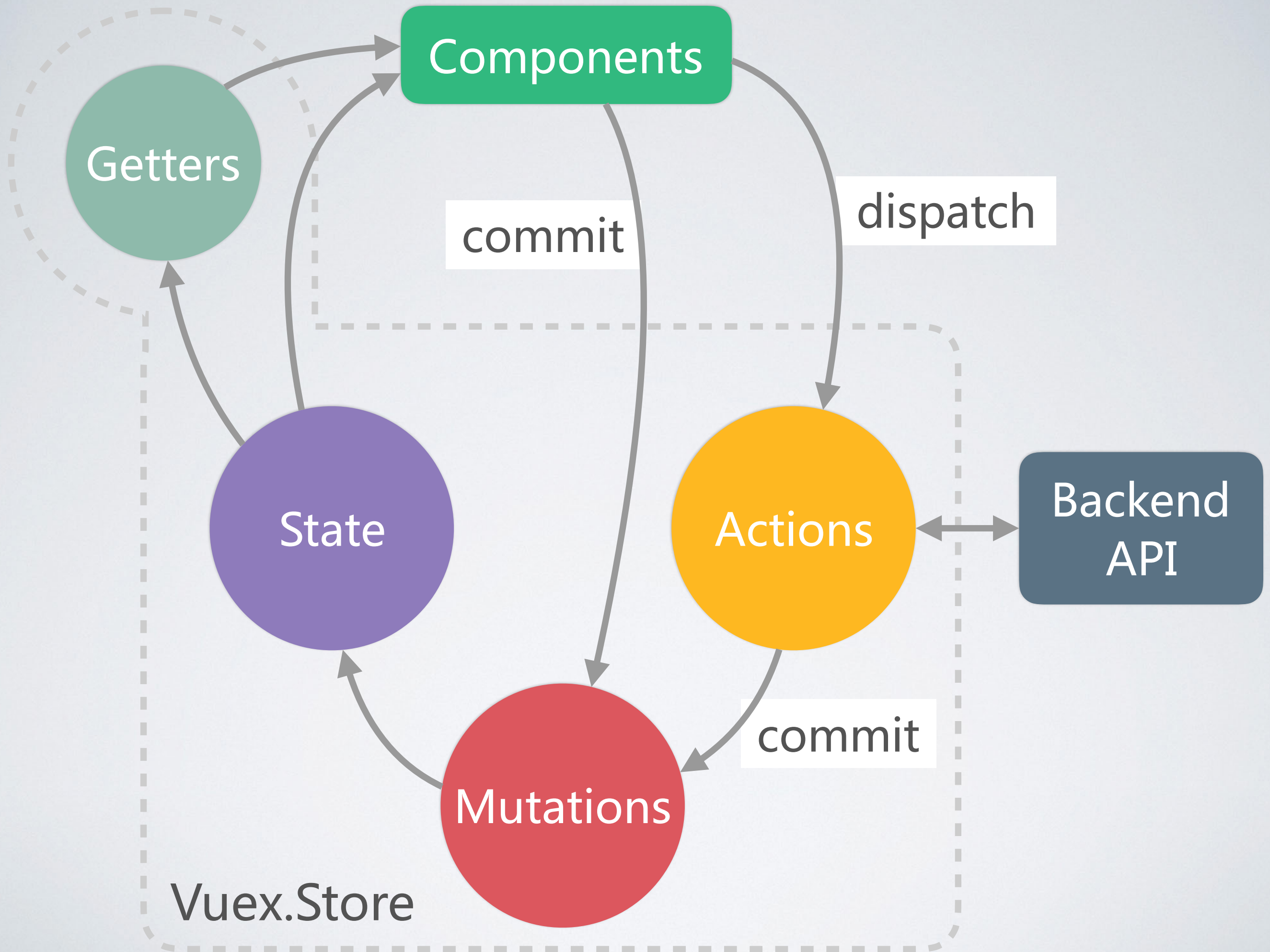
解决方法



- 把组件的共享状态抽取出来，
以全局单例模式统一管理

Vuex

- 抽取组件的共享状态，全局统一管理
- 组件树构成了一个巨大的 view，不管在树的哪个位置，任何组件都能获取状态或者触发行为
 - 变更数据
 - 读取数据
- 通过强制规则维持视图和状态间的独立性



使用 Vuex

1. 导入 vue 和 vuex
2. `Vue.use(Vuex)`
3. 创建 `Vuex.Store` 实例，传入 state、getters、mutations、actions 定义
4. 将 Store 实例注入 vue 根实例

使用 Vuex

导入

```
import Vue from 'vue';
import Vuex from 'vuex';
import state from './state.js';
import getters from './getters.js';
import mutations from './mutations.js';
import actions from './actions.js';
```

安装 vuex

```
Vue.use(Vuex);
```

创建 Store 实例

```
export default new Vuex.Store({
  state,
  getters,
  mutations,
  actions
});
```

state

- 唯一数据源
- 组件通过 `this.$store.state` 访问

```
const state = {  
  ...playing: false,  
  ...playlist: []  
};
```

```
this.$store.state.playing
```

getters

- state 的派生状态，可以理解为计算属性
- 组件通过 `this.$store.getters` 访问

```
const getters = {  
  playing: state => state.playing,  
  currentSong: state => {  
    return state.playlist[state.currentIndex] || {};  
  }  
};
```

以 state 为参数

mutations

- 更改 state 的唯一方法
- 组件通过 `this.$store.commit` 触发
- mutation 必须是同步函数

```
const mutations = {  
  ...setPlayingState(state, val) {  
    ...state.playing = val;  
  },  
  ...setPlaylist(state, val) {  
    ...state.playlist = val;  
  }  
};
```

以 state 为参数

```
this.$store.commit('setPlayingState', true);
```

actions

- 调用异步 API , commit 多个 mutation
- 组件通过 this.\$store.dispatch 分发

```
const actions = {  
  ...randomPlay({ commit }, list) {  
    ...const randomList = shuffle(list);  
    ...commit('setPlaylist', randomList);  
    ...commit('setCurrentIndex', 0);  
    ...commit('setPlayingState', true);  
  },  
  ...clearSongList({ commit }) {  
    ...commit('setPlayingState', false);  
    ...commit('setCurrentIndex', -1);  
    ...commit('setPlaylist', []);  
  }  
};  
  
this.$store.dispatch('randomPlay', songList);
```

第一参数
context 对象中
包含和 store 实例
相同的属性和方法

辅助函数

- `mapState` 将 `this.$store.state` 自动生成计算属性
- `mapGetters` 将 `this.$store.getters` 自动生成计算属性
- `mapMutations` 将 `this.$store.commit` 自动映射成方法
- `mapActions` 将 `this.$store.dispatch` 自动映射成方法

辅助函数

```
import { mapGetters, mapMutations, mapActions } from 'vuex';
```

```
  computed: {  
    ...mapGetters(['playing', 'playlist'])  
  },
```

```
  methods: {  
    ...mapMutations(['setPlayingState', 'setPlaylist']),  
    ...mapActions(['randomPlay', 'clearSongList'])  
  },
```

mutations 事件类型使用常量

- mutations 函数名使用常量
- 所有常量放在单独文件中

```
const SET_PLAYING_STATE = 'setPlayingState';  
const SET_PLAYLIST = 'setPlaylist';
```

types.js

```
import * as types from './types.js';  
  
const mutations = {  
  [types.SET_PLAYING_STATE](state, val) {  
    state.playing = val;  
  },  
  [types.SET_PLAYLIST](state, val) {  
    state.playlist = val;  
  }  
};
```

课后作业

- 了解 store 模块分割 module