iOS应用Redux介绍

- Redux介绍
- “考勤”模块结构Redux化实践

Redux介绍

- Redux是JavaScript状态容器，提供可预测化的状态管理
    - state：生命周期内会发生改变的影响页面的数据
    - action：用户行为或程序自动行为（定时器、网络下载），仅描述了有事件发生
    - reducer：指定了state与action之间的响应
    - store：联系上述三者，持有state，监听action，响应reducer

“考勤”模块Redux化实践

- 考勤模块页面如图所示：
  - 用户点击按钮和定时器响应行为都会使页面发生变化

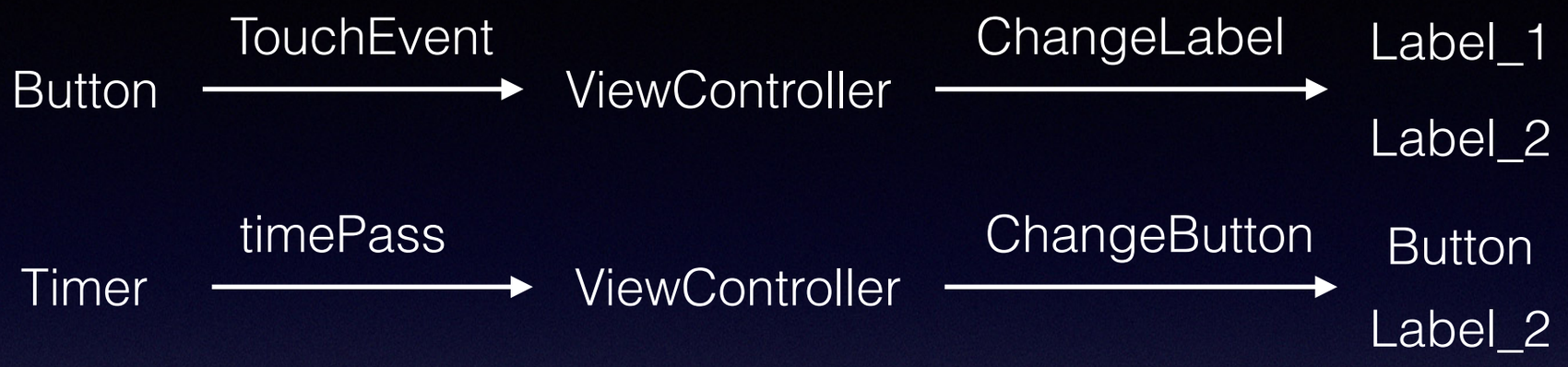
- 简化模型：

[最早可考勤时间] [开始考勤时间] → Label_1

考勤按钮 → Button

[最迟可考勤时间] [结束考勤时间] → Label_2

- 常规MVC的设计及缺陷：



  - 按钮的点击签到/签退，会更改标签的内容；时间的变化会更改按钮点击状态/颜色/文案，以及签退变迁的展示

  - UI的变化分布于ChangeLabel、ChangeButton两个方法中，随着业务细化，UI变化的代码将更加散乱，导致很难追踪UI的变化

  - 功能复杂度的增加会导致UI与变量之间的联系愈发紧密，导致难以重构

- Redux应用（Swift环境）
  - 定义协议，用于泛型约束：

    protocol StateType {}

    protocol ActionType {}

  - State:

```
struct State: StateType {
    var isSignIn: BOOL = false; //签到状态
    var time : timeState = beEarly //时间状态
}

enum timeState {
    case beEarly  //考勤未开始
    case onTime  //考勤中
    case beLate   //考勤已结束
}
```

- Action：　enum Action: ActionType {

　　case buttonTap　//按钮点击

　　case timePass　//时间

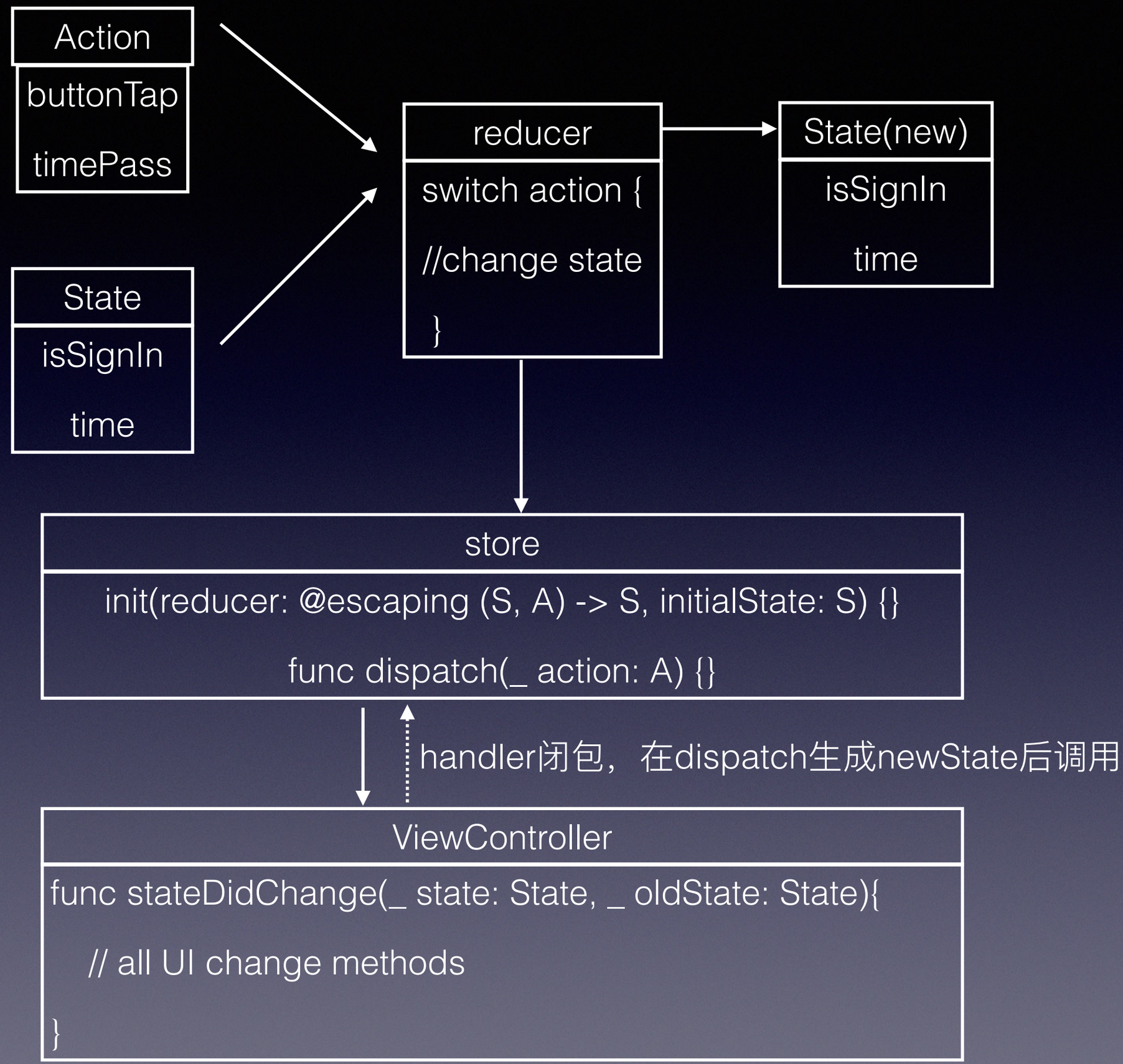
　}

- reducer：var reducer: (State, Action) -> (newState: State)，一个闭包，接收旧的State和发生的Action，返回新的State

- store：　class Store<A: ActionType, S: StateType>{}　//声明

　　init(reducer: @escaping (S, A) -> S, initialState: S) {}　//初始化并持有State

　　func dispatch(_ action: A) {}　//监听Action,并响应reducer


Action
buttonTap
timePass
State
isSignIn
time
reducer
switch action {
//change state
}
State(new)
isSignIn
time
store
init(reducer: @escaping (S, A) -> S, initialState: S) {}
func dispatch(_ action: A) {}
handler闭包，在dispatch生成newState后调用
ViewController
func stateDidChange(_ state: State, _ oldState: State){
// all UI change methods
}