Coming soon…

README

- 张强
- 美团点评
- madlord.cn@gmail.com

详细信息 编辑

张强张耳朵

62093 我获得的赞同

11837 我获得的感谢

29921 回答被收藏

213 回答被分享

从事行业

互联网

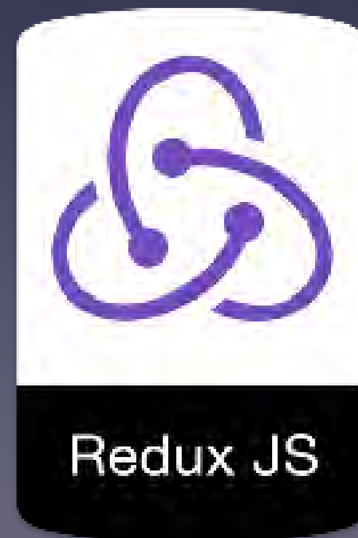
居住地

上海 现在

工作经历

美团点评 · 前端 现在

Redux的打开方式



Overview

- 前端史
- Redux的问题
- 私货

简明前端史

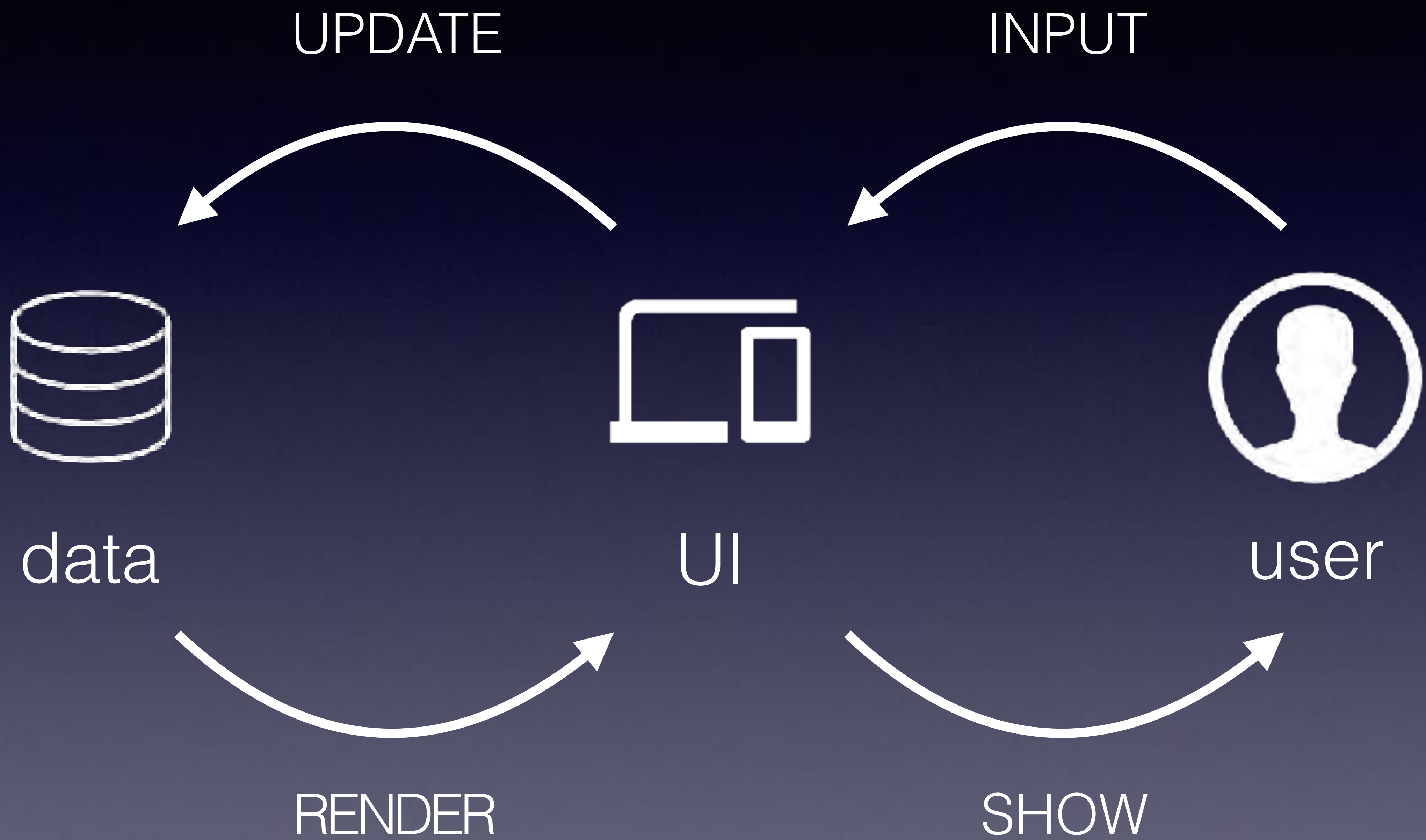
数据&可维护性

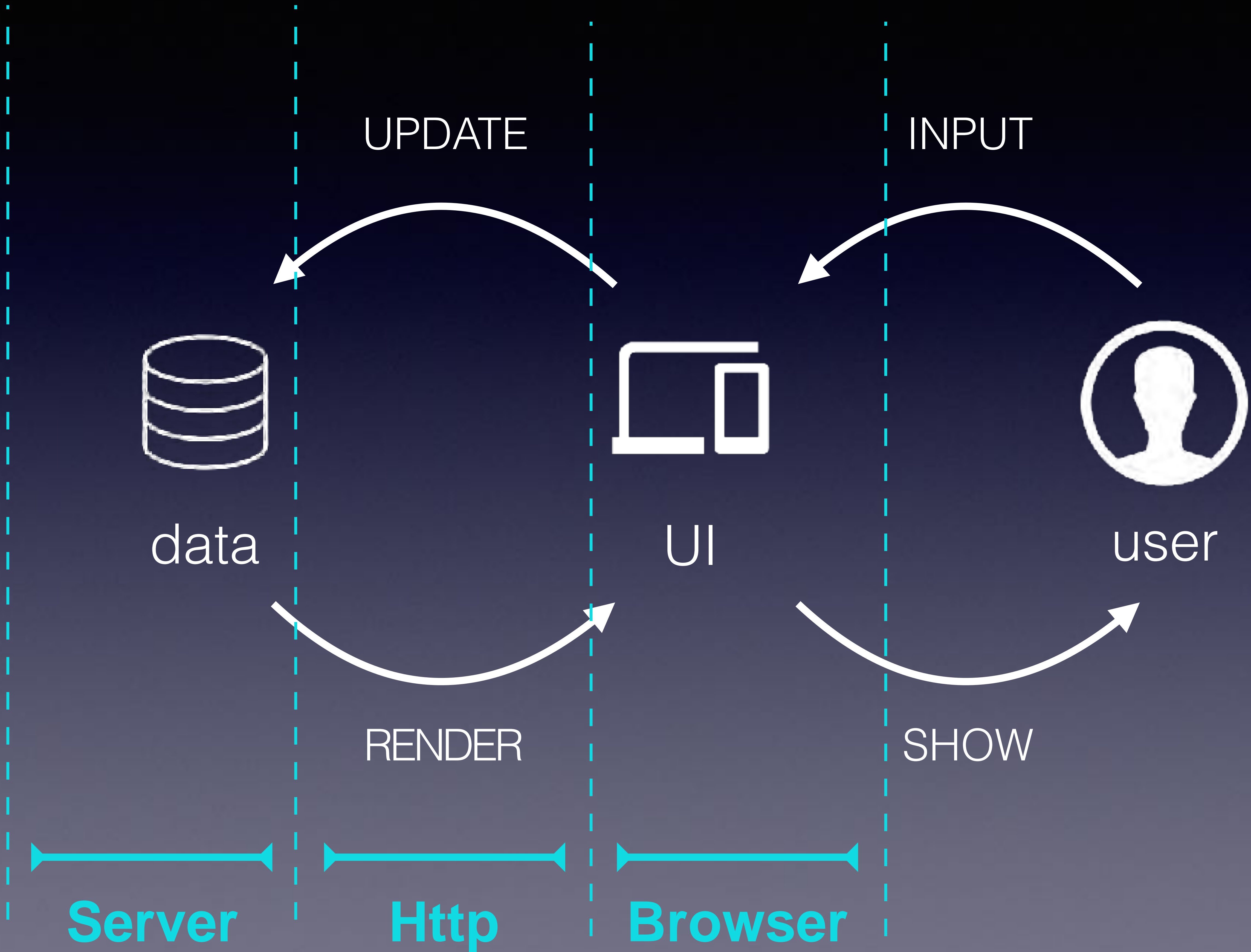
- 古典时代
- 中世纪
- 文艺复兴

前端的要解决的本质问题

data

user


UPDATE
INPUT
data
UI
user
RENDER
SHOW

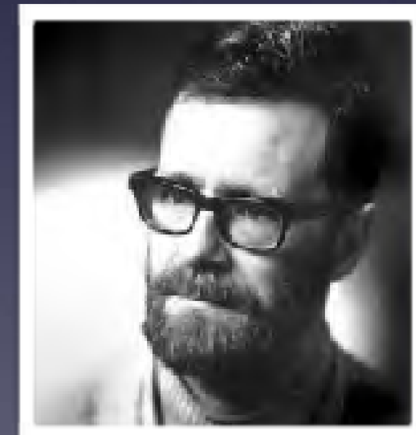

UPDATE
INPUT
data
UI
user
RENDER
SHOW
Server
Http
Browser

最传统的BS架构（90年代）

- 单一数据源：server
- 统一的数据变更方式：URL
- 统一的UI渲染方式：html直出
- 完美的状态管理：history 浏览器的前进后退

“Simplicity is prerequisite for reliability”

–Edsger W. Dijkstra



总结

- 可靠：不容易出错 出错也好定位
- 简单：学习成本低
- 浏览器搞定了大部分事情，开发者只需关心UI渲染：html+css+~~javascript~~

古典时代

简单即是美

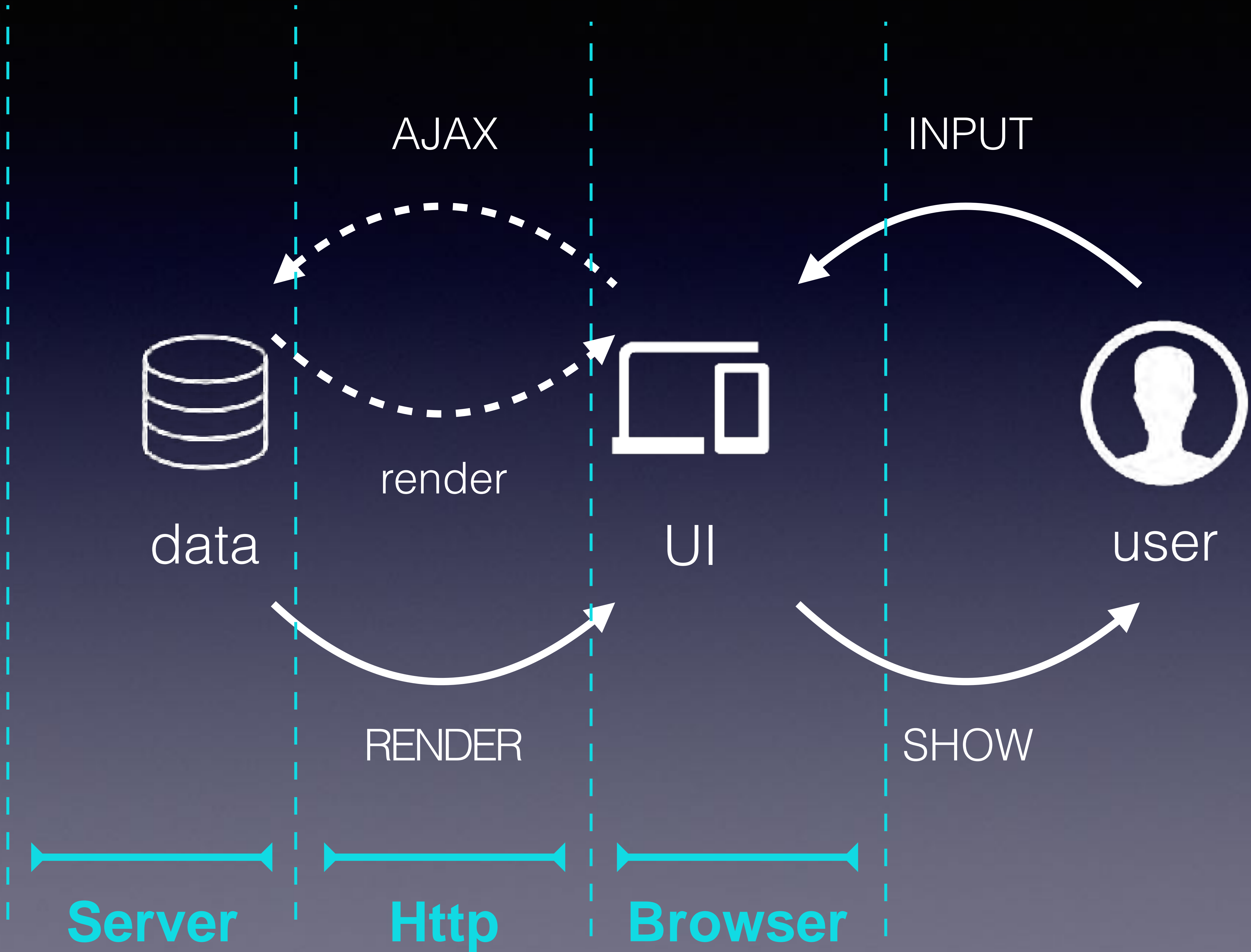
Developer

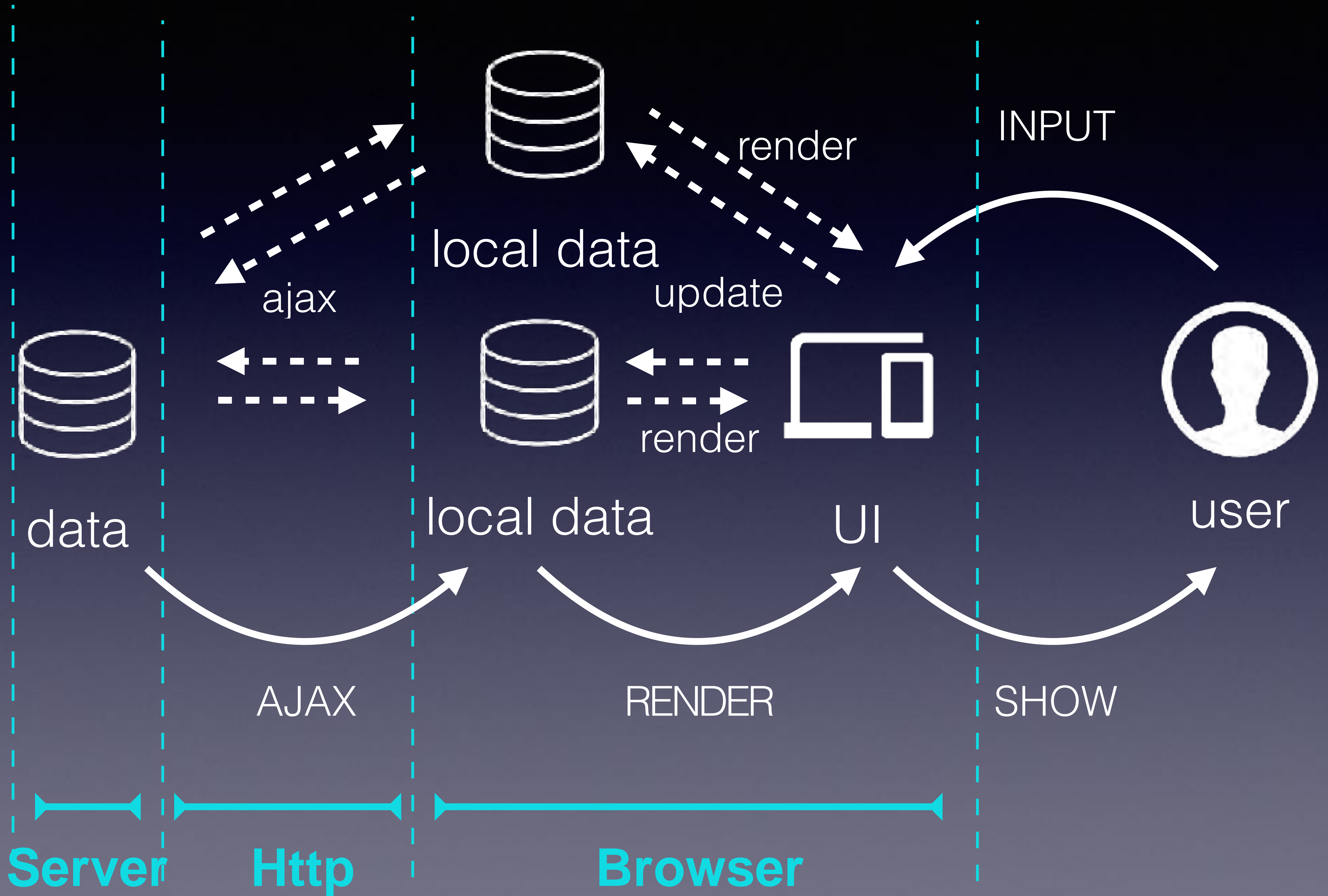
这是最好的时代

User

这是最坏的时代

AJAX来了(2005)


AJAX
INPUT
render
data
UI
user
RENDER
SHOW
Server
Http
Browser

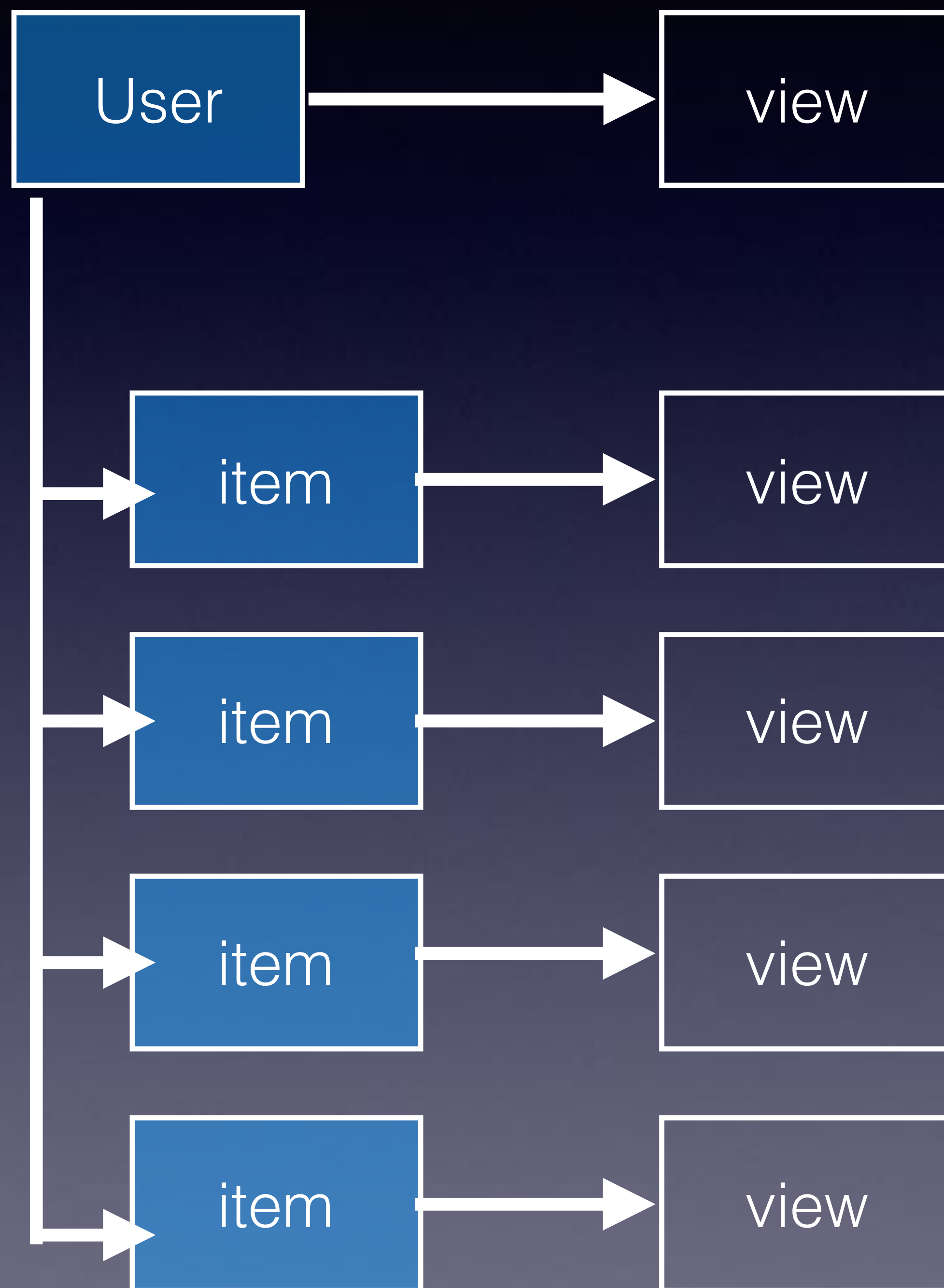


INPUT
render
local data
ajax
update
data
local data
render
UI
user
AJAX
RENDER
SHOW
Server
Http
Browser

AJAX带来的变化

- ~~单一数据源~~： 远程数据+本地数据*N
- ~~统一的数据变更方式~~： URL+AJAX+UI对本地数据的更改
- ~~统一的UI渲染方式~~： html直出+本地全局(局部)dom渲染
- ~~完美的状态管理~~： 浏览器前进后退失效，状态自己管理

local state



总结

- 多数据源：多副本数据的一致性维护困难
- 到处都是数据变更：预测一个操作带来的数据变更愈发困难
- 谁都能改DOM：不知道是谁改了我的DOM
- 状态管理?

bug越修越多，加班修bug

加班

复杂而又混乱

常年加班

招不到人，人手不够还得加班

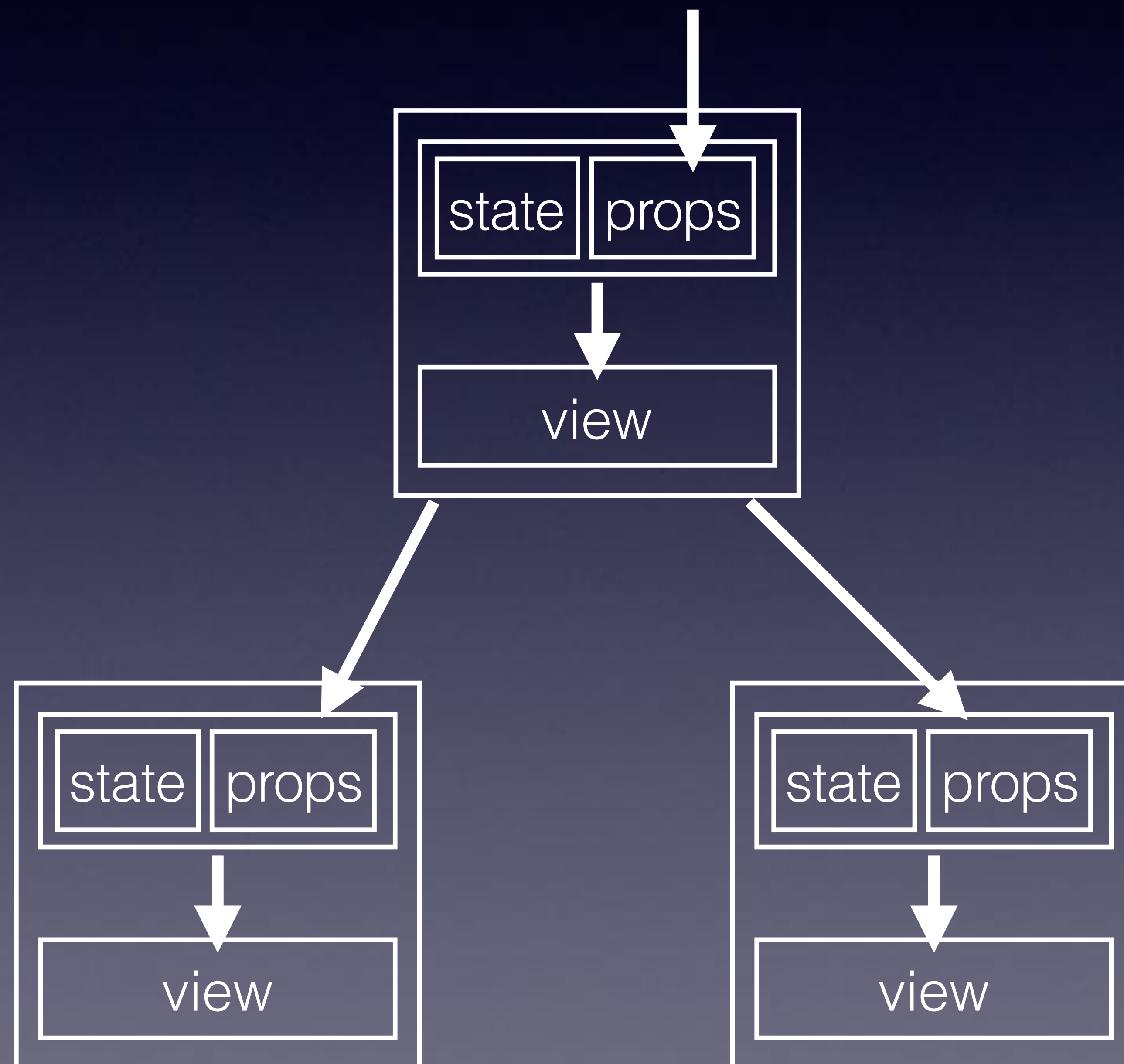
代码无法维护，加班重构

中世纪

无尽的bug,无尽的黑暗

React (2013)

React独立架构

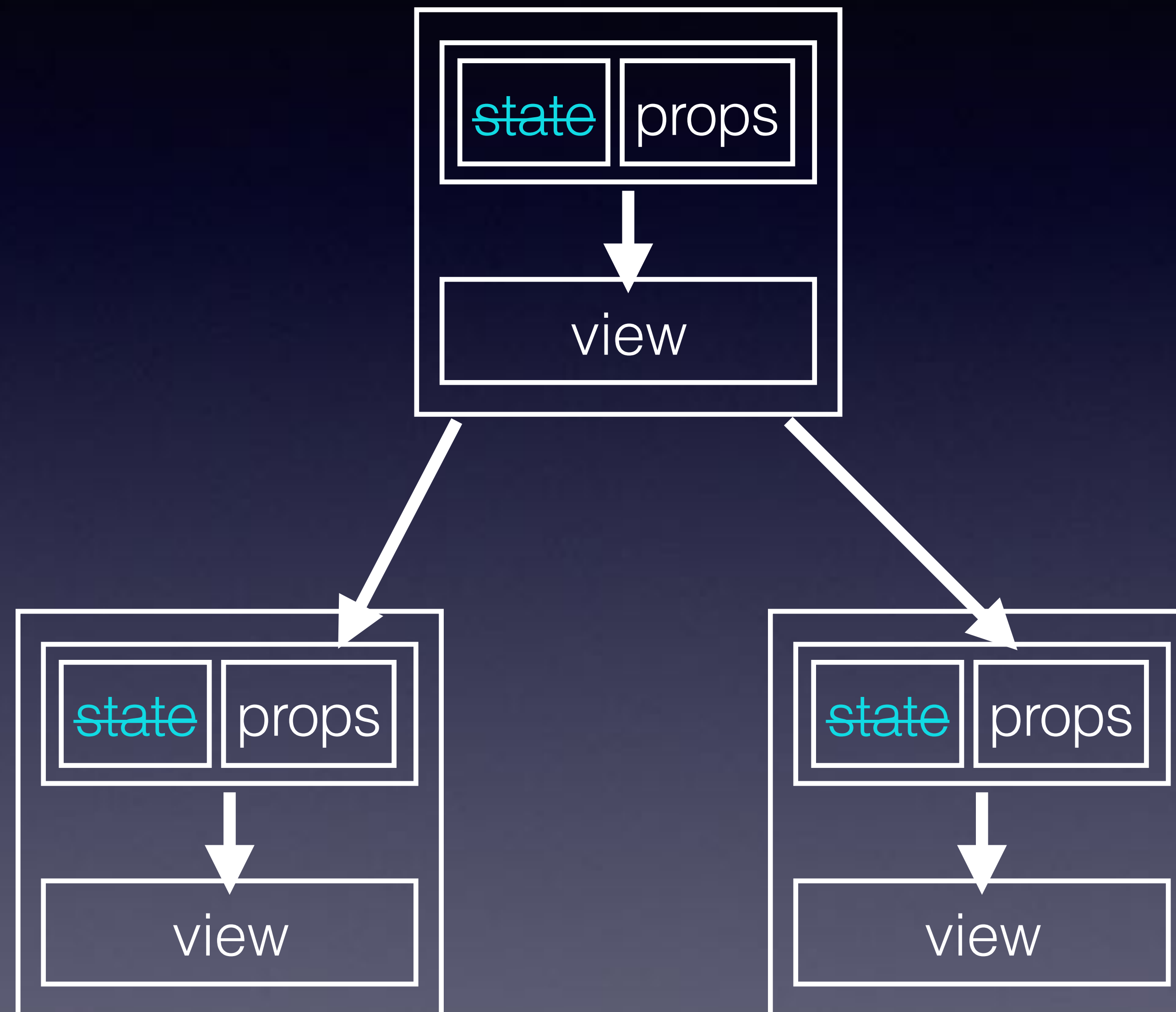


裸用React

- 多数据源：每个组件有自己的本地state
- 数据变更：react没有约束
- UI渲染：没有DOM操作，与真实DOM隔离
- 状态管理?

React+Redux (2015)

弃用react的内部state

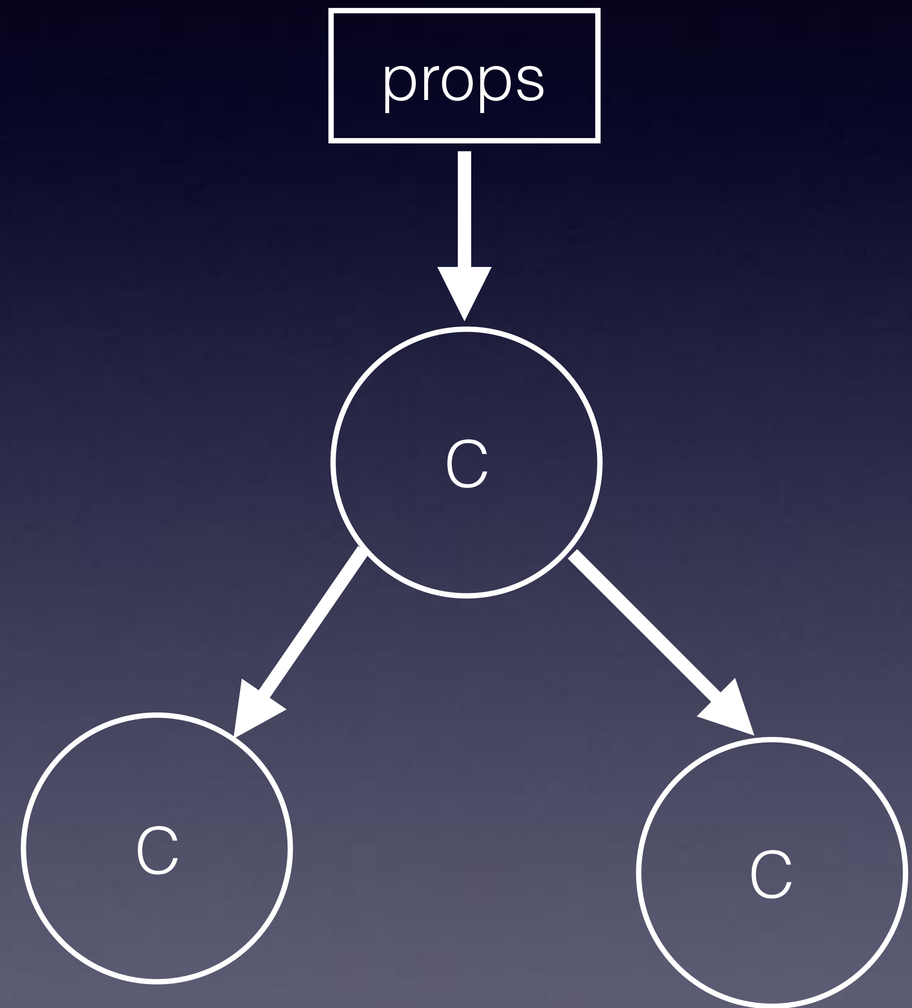


react回归纯渲染

f(props)=View

react component tree

pure function



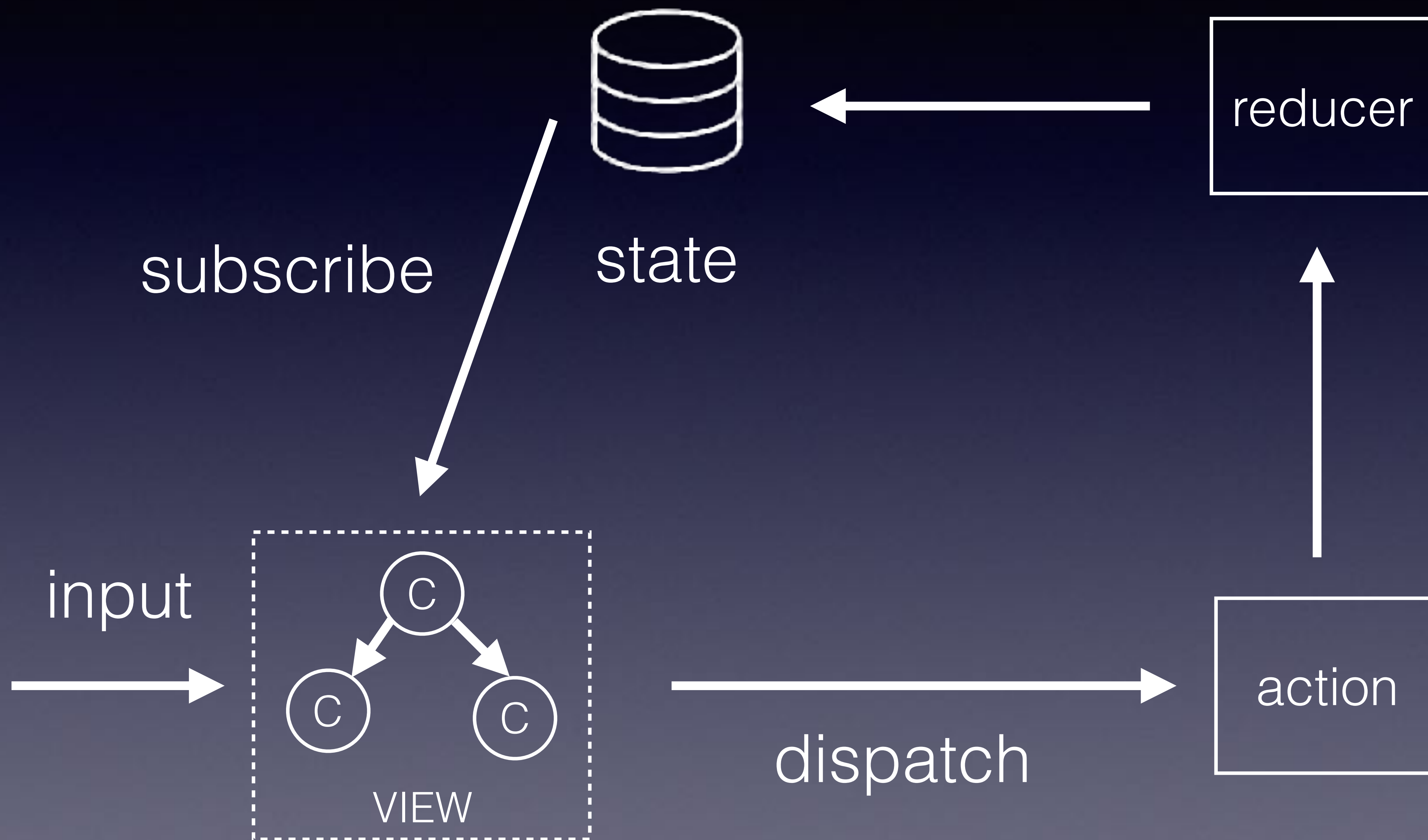
redux

action

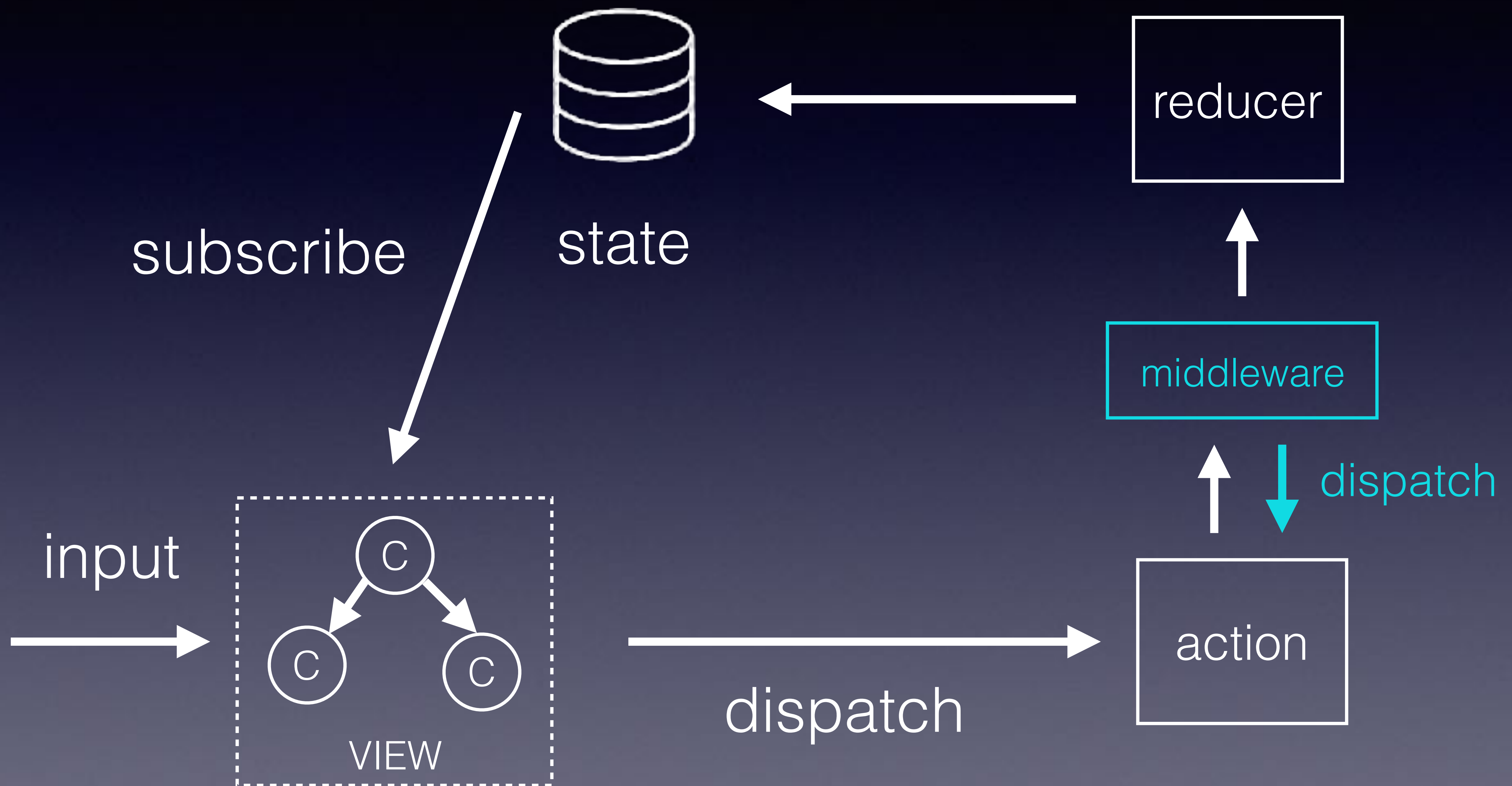
reducer

```
import { createStore } from 'redux'

const store = createStore(counter);

store.dispatch({ type: 'INCREMENT' });

const counter=(state = 0, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return state + 1
    default:
      return state
  }
}

store.subscribe(()=>{
    console.log(store.getState())
})
```

react+redux数据流



react+redux数据流



redux三大原则

- 单一数据源
- state只读，只能通过触发action来进行更改
- action通过reducer来修改state，reducer必须为纯函数

redux状态变更

$$f_{\text{reducer}}(\text{state_0}, \text{action_0}) = \text{state_1}$$

redux状态变更

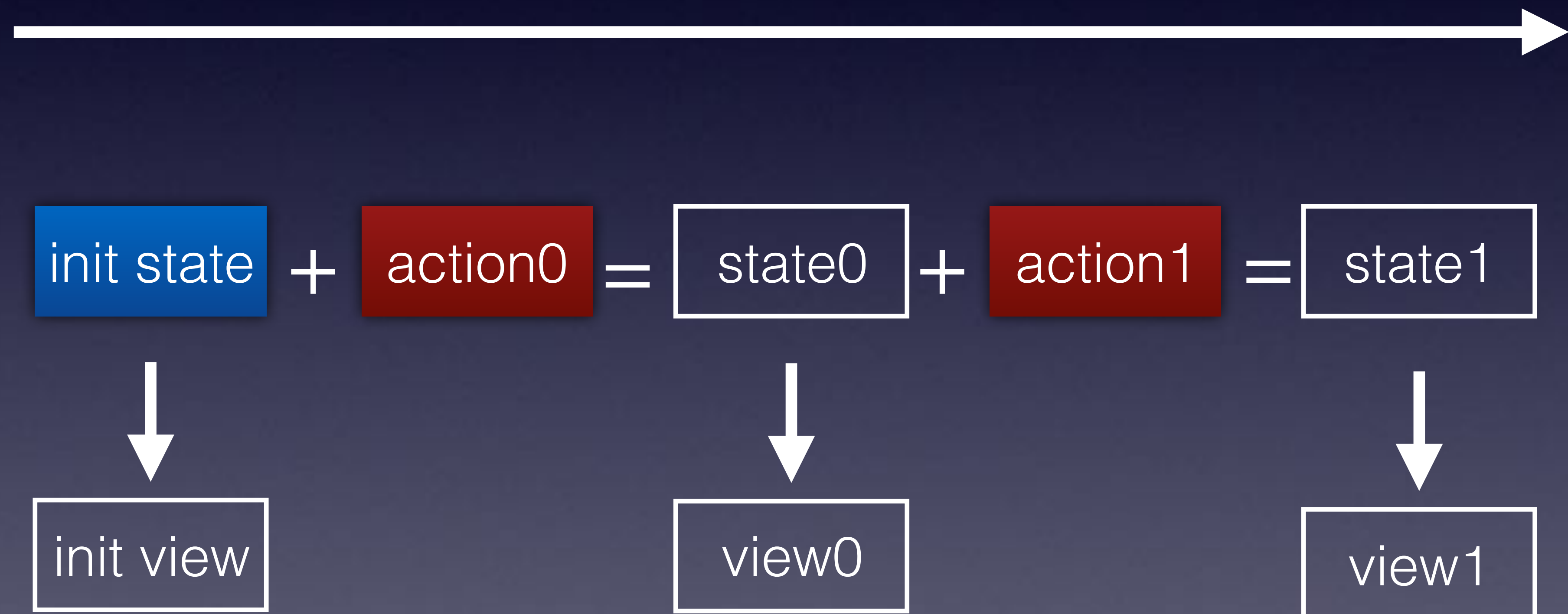
$$f(f(state_0, action_0), action_1) = state_1$$

reducer

pure function

时间旅行

time line



总结

- 单一数据源：redux的全局state
- 统一个的数据变更：action
- 统一的UI渲染：react
- 状态管理：“时间旅行”

server

URL

server直出html

浏览器history

文艺复兴

回归秩序，化繁为简

但是

Redux
看起来简单，用起来却不容易

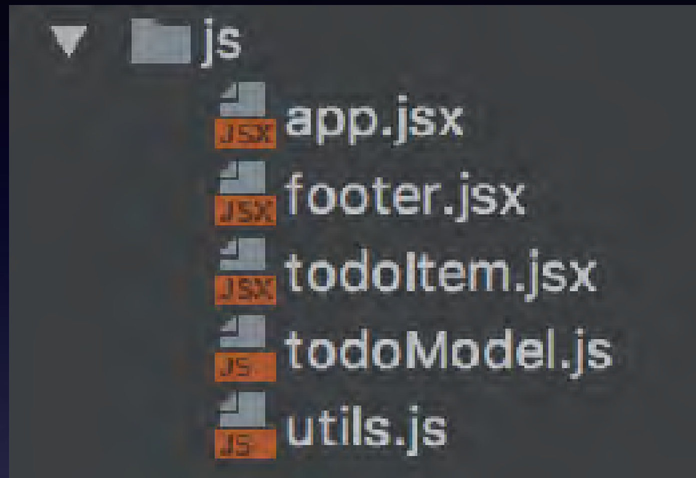
大象关冰箱

- 把冰箱门打开
- 把大象放进去
- 把冰箱门关上

大象关冰箱 with redux

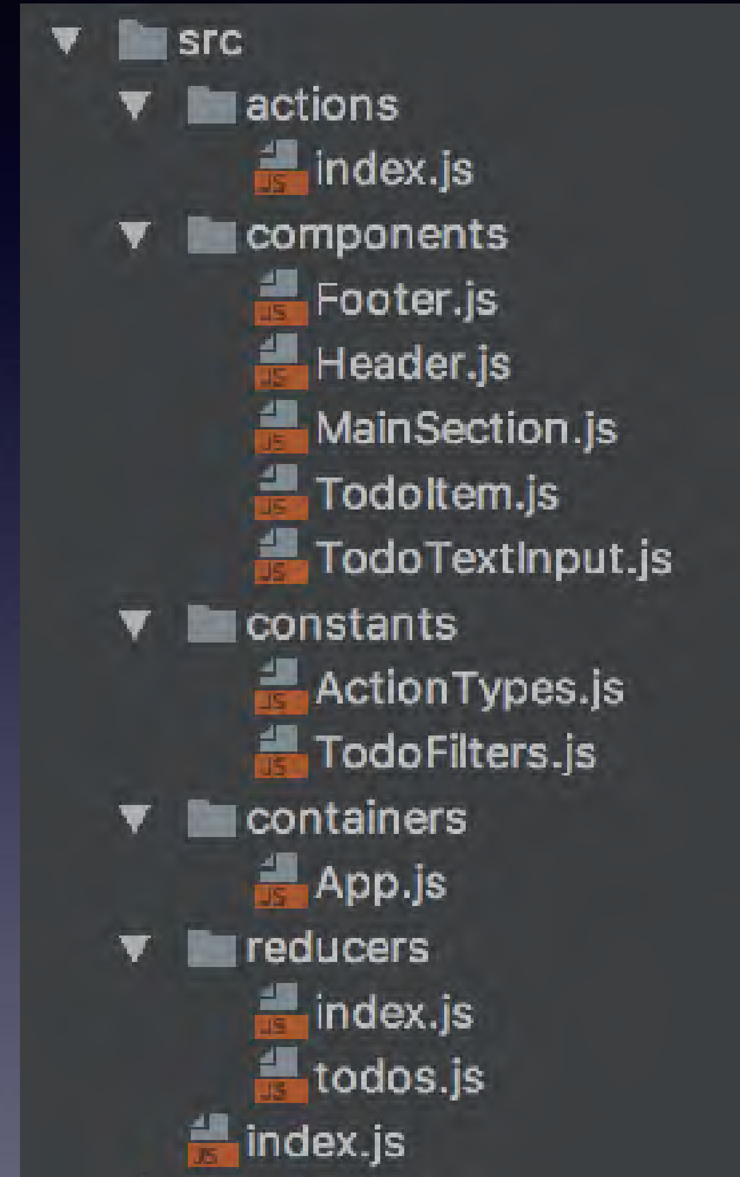
- 定义3个actionType
  - ”打开冰箱门“
  - ”把大象放进去“
  - ”关上冰箱门“
- 实现3个actionCreator
- 实现3个reducer

Redux的样板代码

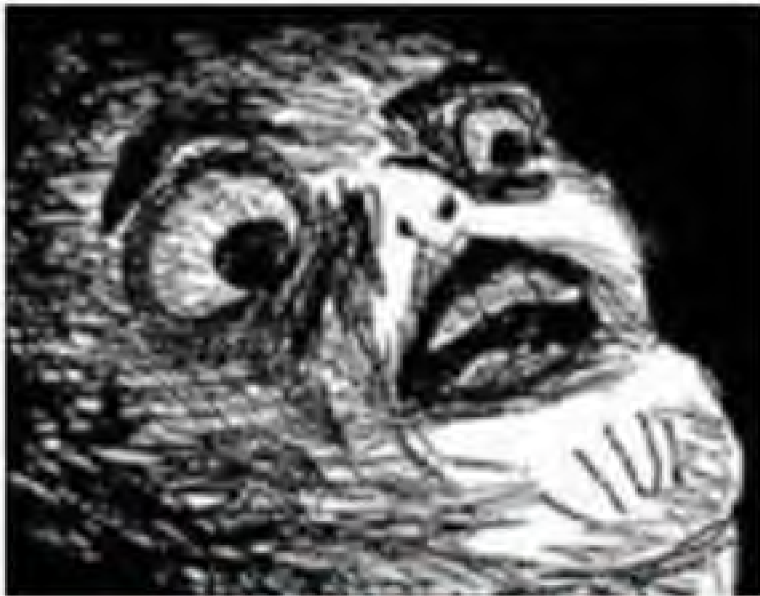


纯React
Files **x5**

todoMVC



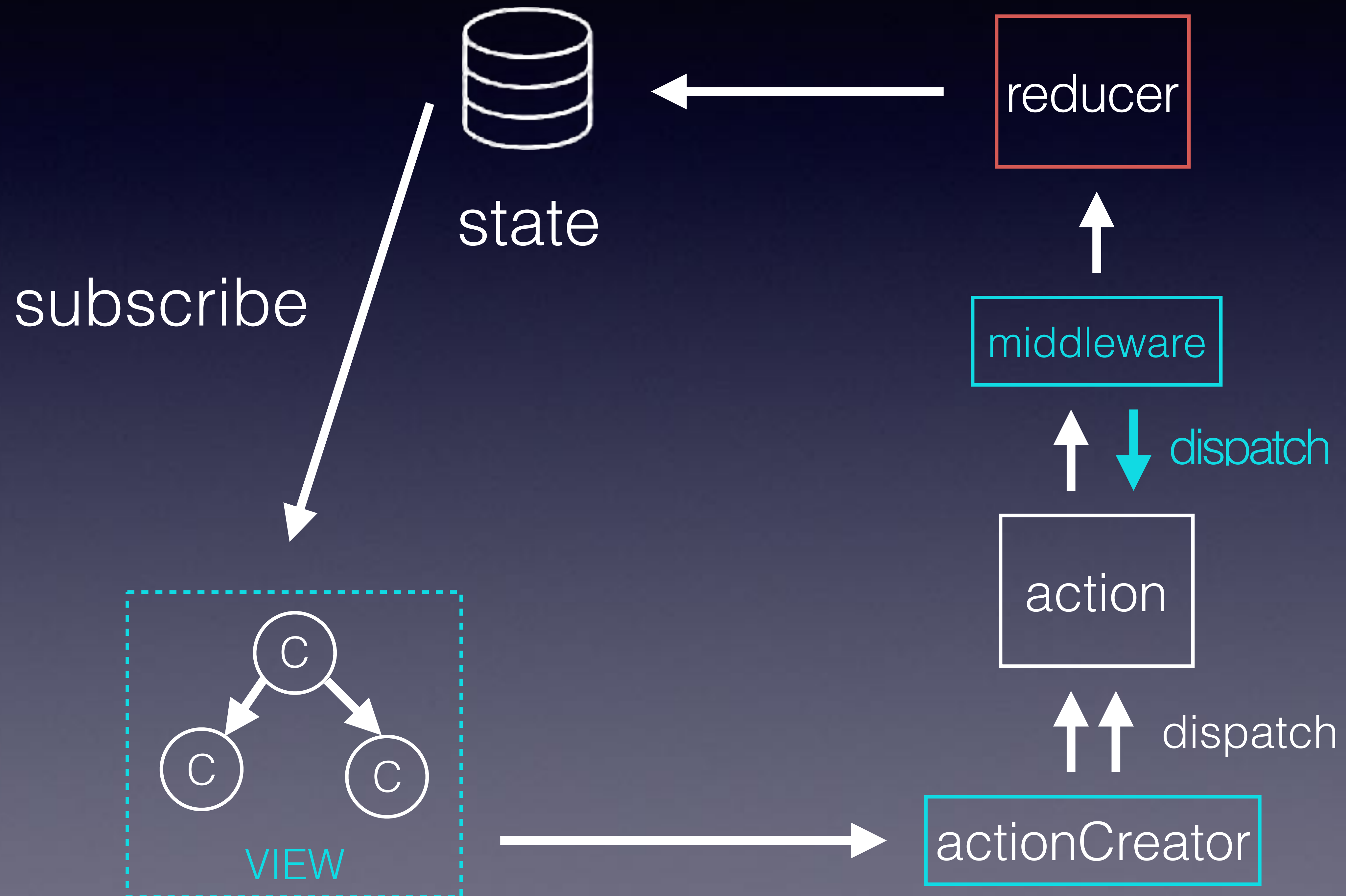
Redux
Files **x12**



第一次看redux文档

第一次写redux应用

Redux处理异步



第一次看小伙伴们写的redux代码

Redux的模块化

- 大型应用->协同开发->模块化
- 每个模块拥有自己的action/actionType/reducer
- actionType重名的问题：约定+规范
- 模块间通信的问题
- 模块动态加载的问题：破坏了reducer的纯净

Redux的API

- createStore
- **combineReducers**
- **applyMiddleware**
- **bindActionCreators**
- **compose**
- getState
- dispatch
- subscribe
- getReducer
- **replaceReducer**

全部API x10

与扩展相关的API x5

Redux不适合直接用于生产环境

我们需要一款框架对redux进行封装和约束

duxjs

duxjs是什么？

- 可用于生产环境的，基于react+redux的前端框架
- 部分借鉴了ducks（关于redux模块化的提议） 和choo

duxjs特性

- 声明式API，没有样板代码
- 模块化/组件化，可嵌套，可动态加载
- 统一的异步处理
- 同构
- HMR热替换
- 插件

component

```
{
    "state":{},
    "actions":{},
    "selectors":{},
    "views":{},
    "subscriptions":{}

    "components":{}
}
```

同步action

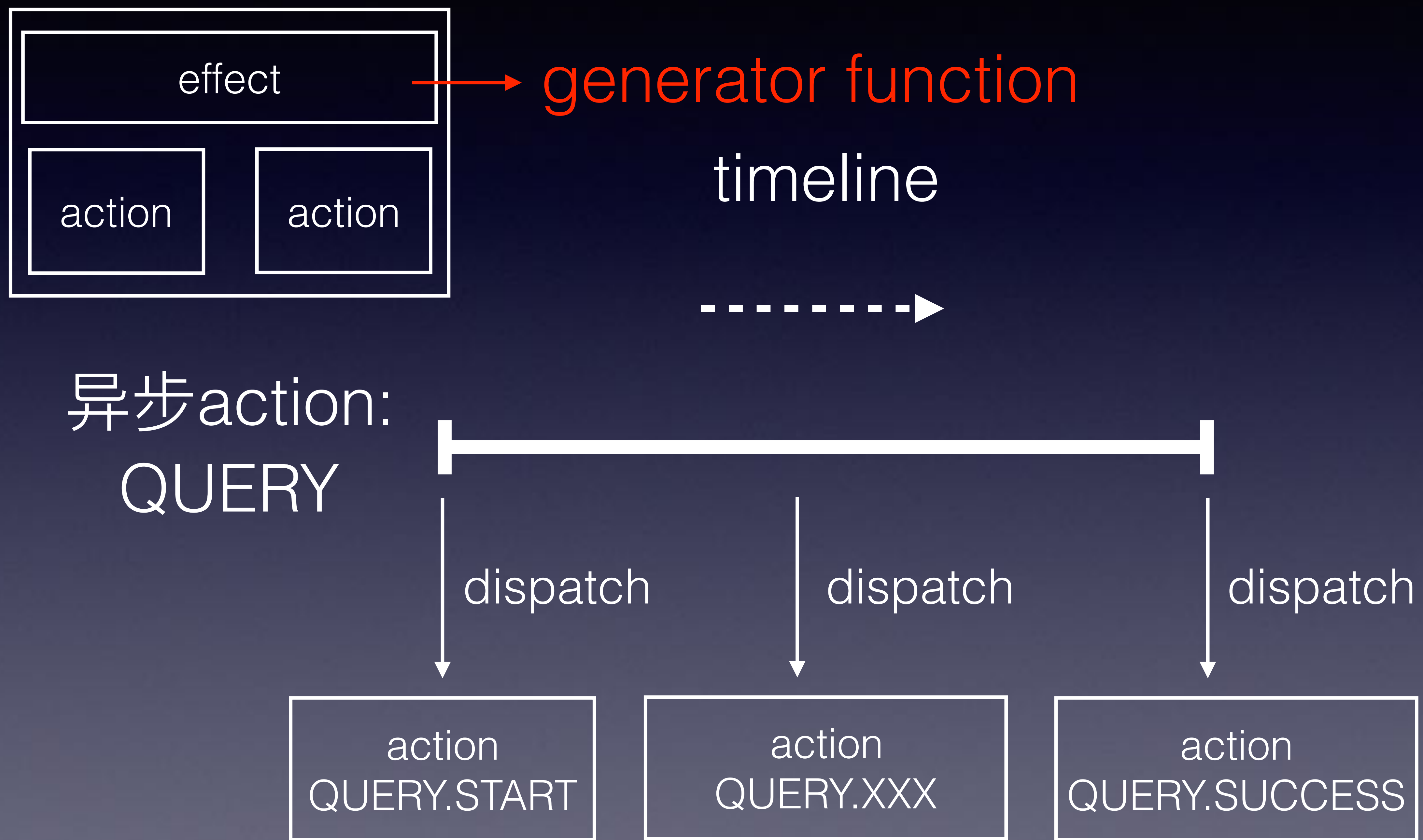
- 符合Flux标准action
- 整个action实体都可被序列化

```
{
    type:”INCREMENT”,
    payload:1,
    meta:{
        …
    }
}
```

同步action

```
{
    "actions": {
        "INCREMENT": {
            "payload": (v) => v,
            reducer(state, action)
            {
                return {
                    ...state,
                    "timer": state.timer + action.payload
                }
            }
        }
    }
}
```

异步action



异步action

```
{
    "actions": {
        "INIT": {
            "effect": (...args) => function*({$action}) {
                return yield dispatch($action("QUERY")());
            },
            "SUCCESS": {
                "payload": (v) => v,
                "reducer": (state, action) => {
                    return {
                        ...state,
                        "list": action.payload
                    };
                }
            }
        },
        "QUERY":{
            "effect": (...args) => function*({$action}) {
                return yield fetch_data();
            }
        }
    }
}
```

state

```
{
    state: {
        timer: 10
    }
}
```

定义组件内的初始化state

selector

```
{
    selectors:{
        user_list:({
            state,
            $selector
        }) =>{
            return state.list
        }
    },
}
```

对外暴露的state数据接口

view

```
{
    views: {
        default: (connect) => connect(({$view, $selector}) => {
            return {
                TopView: $view('a->default'),
                BottomView: $view('b_view'),
                timer: $selector('timer')
            }

        }, ($action) => ({
            "CLICK": $action("MOUNT_AND_INIT_A")
        }))(View)
    }
}
```

集成与组件相关的View (React Component)

subscription

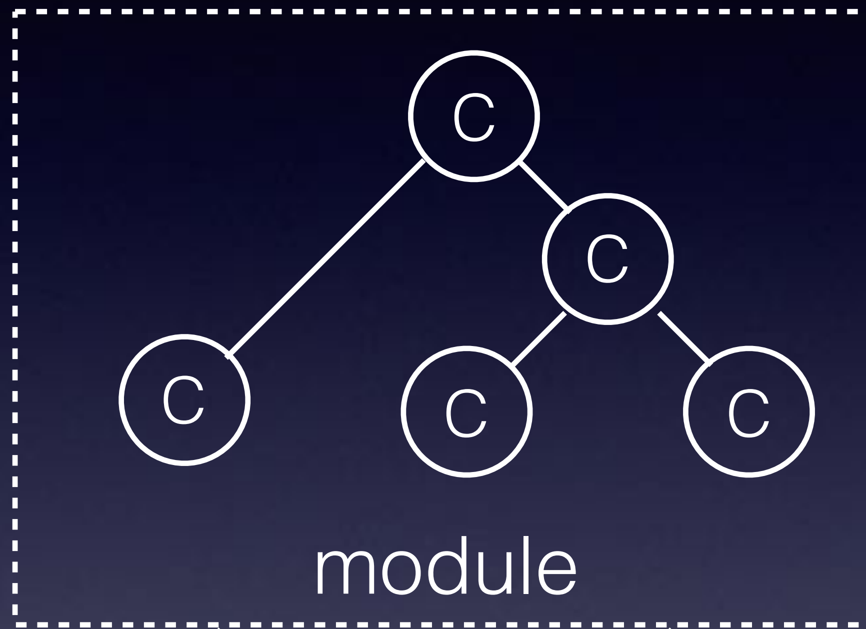
```
{
    subscriptions: {
        "web_socket": ({$action,dispatch,$selector}) => {
            socket.on('msg', function (data) {
                dispatch($action("RECEIVE_MSG")(data));
            });
        }
    }
}
```

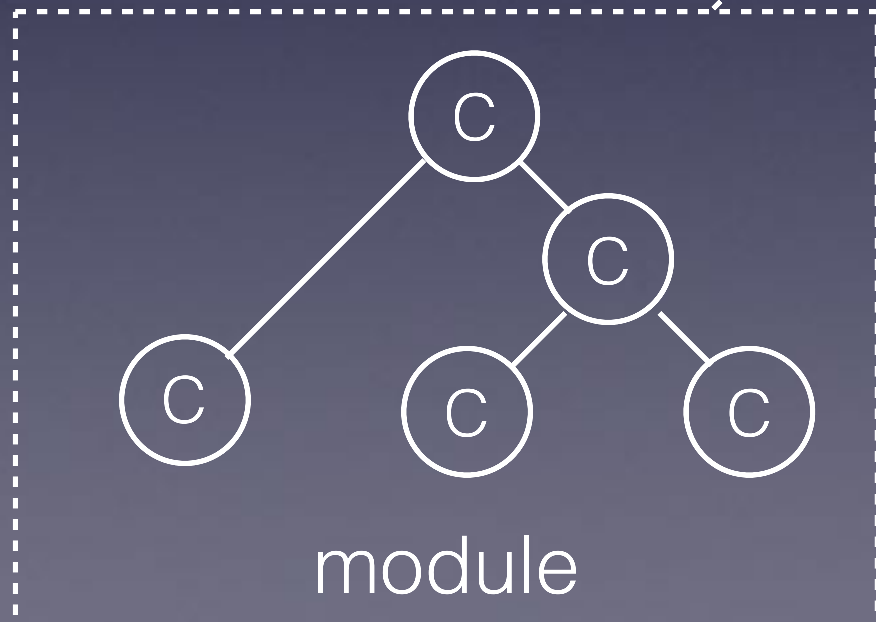
订阅来自外部系统的消息

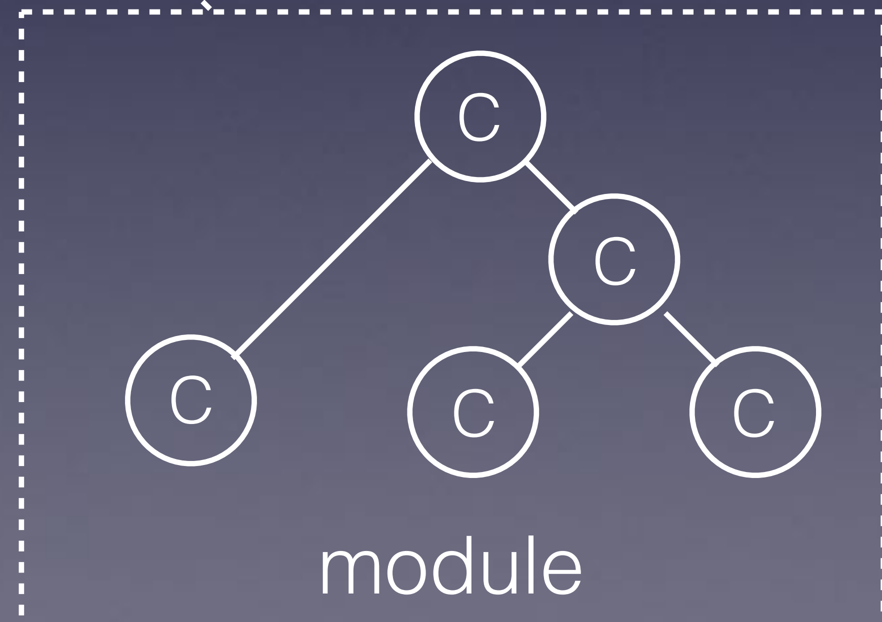
- WebSocket
- 全局键盘事件
- jsbridge通知

module

APP







module

```
{
    modules:{
        "module_a":{
            getModule(cb) {
                require.ensure([],()=>{
                    const ModuleA=require("@modules/a").default;
                    cb(ModuleA);
                })
            }
        },
        "module_b":{
            module:require("@modules/b").default
        }
    }
}
```

在component中定义子module

module

```
{
    modules:{
        "module_a":{
            getModule(cb) {
                require.ensure([],()=>{
                    const ModuleA=require("@modules/a").default;
                    cb(ModuleA);
                })
            },
            "A_CLICK":({
                dispatch,
                $action
            }, action) => {
                dispatch($action("MODULE_A_CLICK")(action.payload))
            }
        }
    }
}
```

module间通信： 子->父

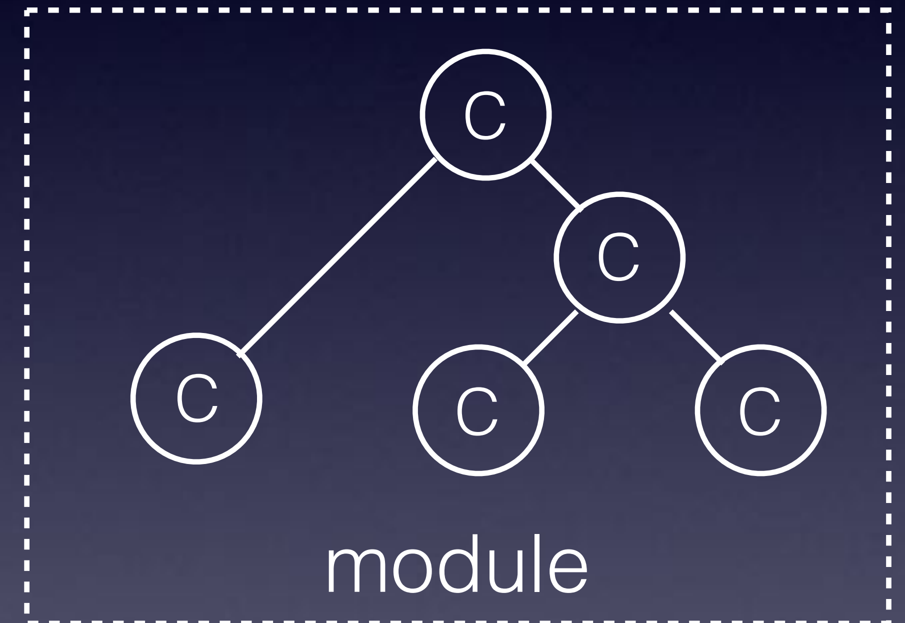
module

```
dispatch($action("module_b->INIT")())

const userList = $selector("module_b->user_list")

const editView = $view("module_b->edit_view")
```

module间通信：父->子

module间解耦

- 独立的命名/状态空间：由父组件指定
- 只有父子module能通信，禁止隔代通信



action标记

```
{
  type: 'ROOT->INIT.SUCCESS',
  meta: {
    id: '36781bd0-1a26-11e7-9e90-0912e9fa0611',
    src: 'ROOT->INIT.START[3674c070-1a26-11e7-9e90-0912e9fa0611]@ROOT'
  }
}
```

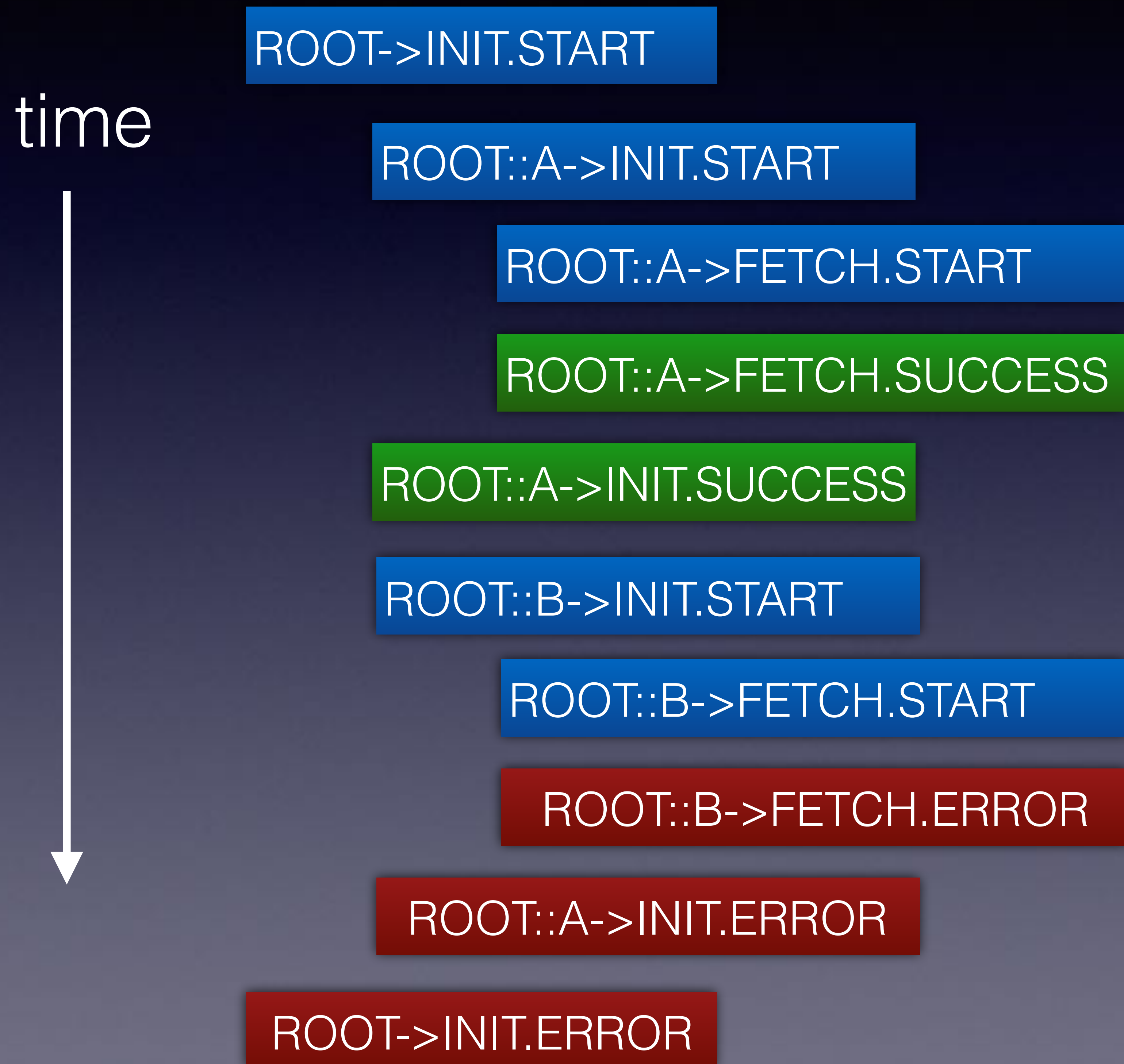
当前action ID

源actionType

源action ID

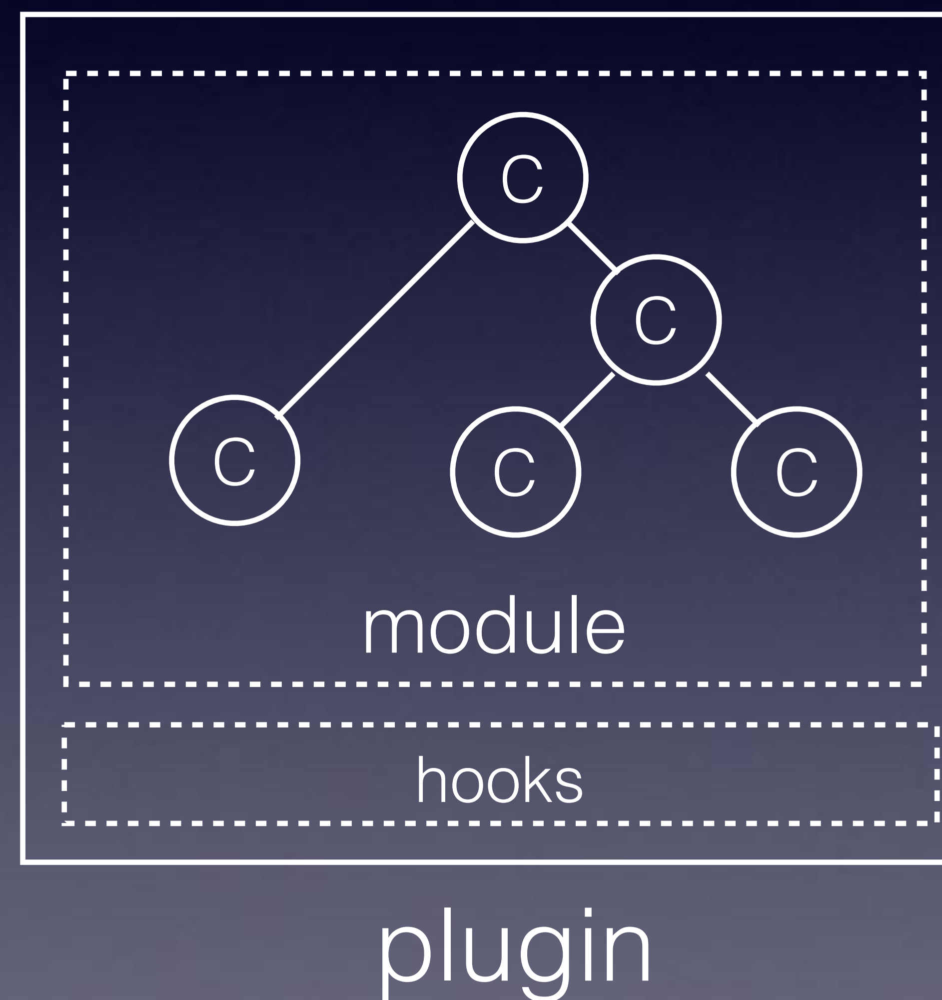
触发时所在的模块

action调用栈图表

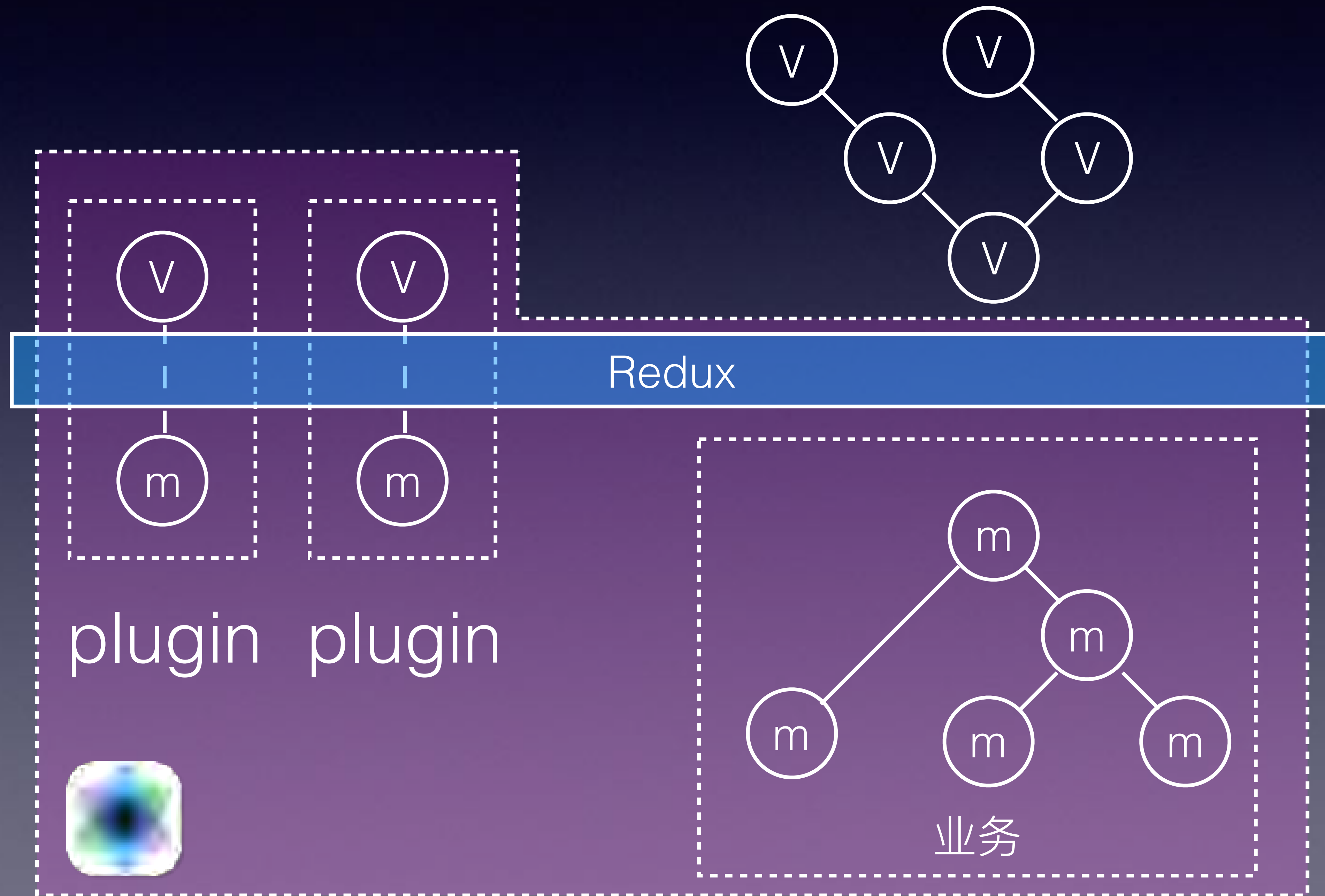


plugin

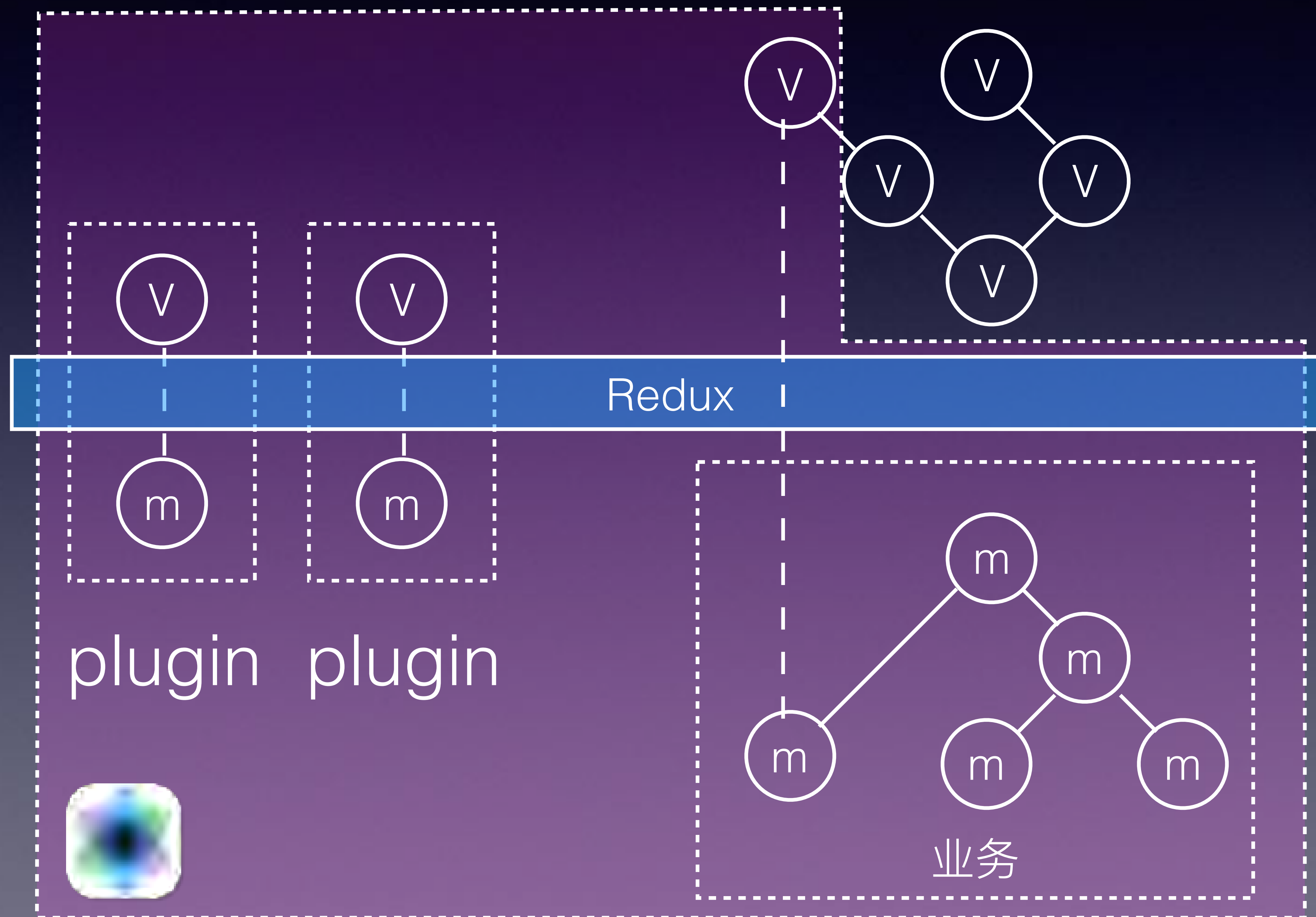
- 加强版的module
- 劫持全局的同步/异步action
- 监听全局的state变化
- 捕获全局的异常
- 有自己的view



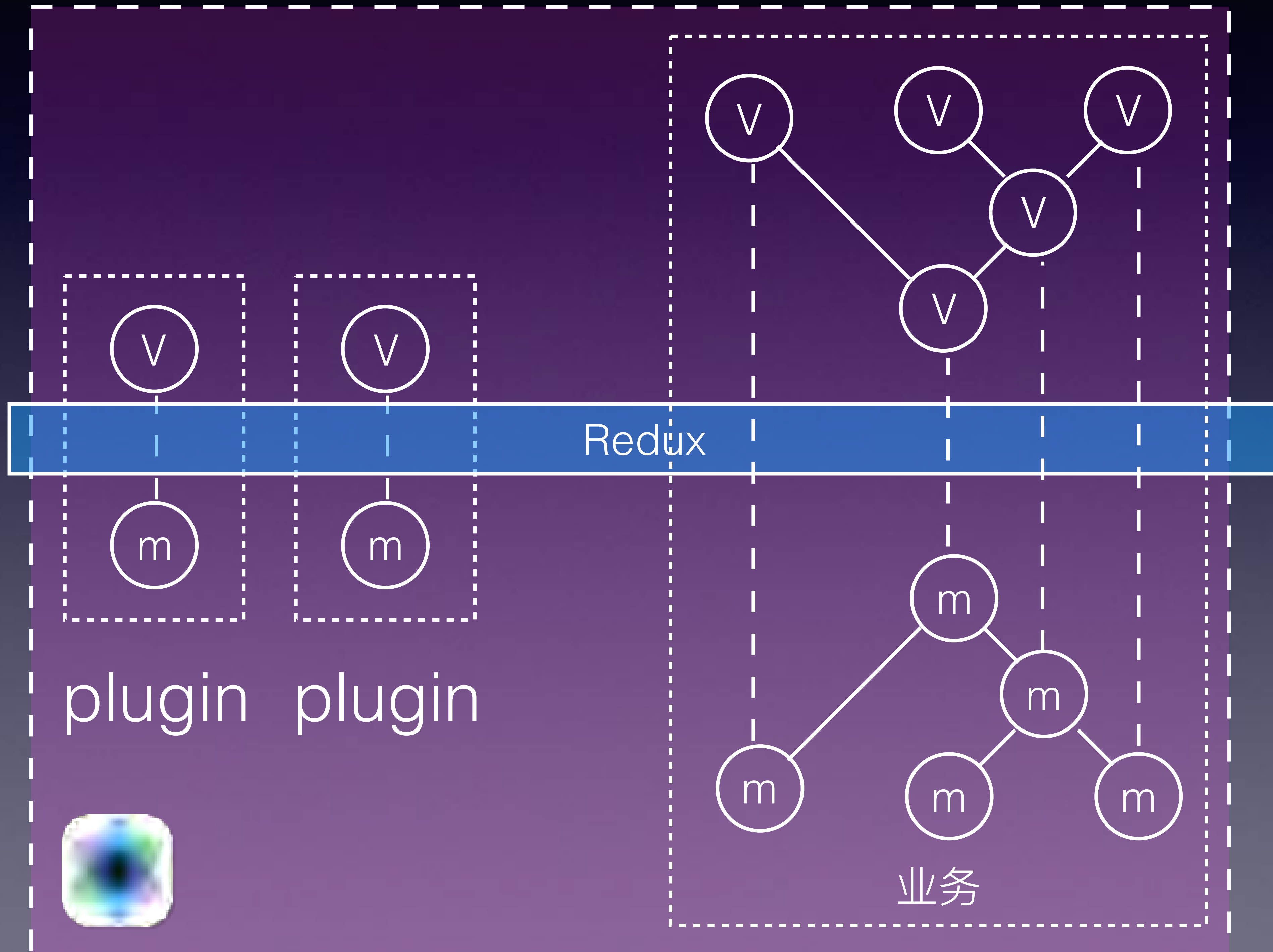
纯数据管理模式



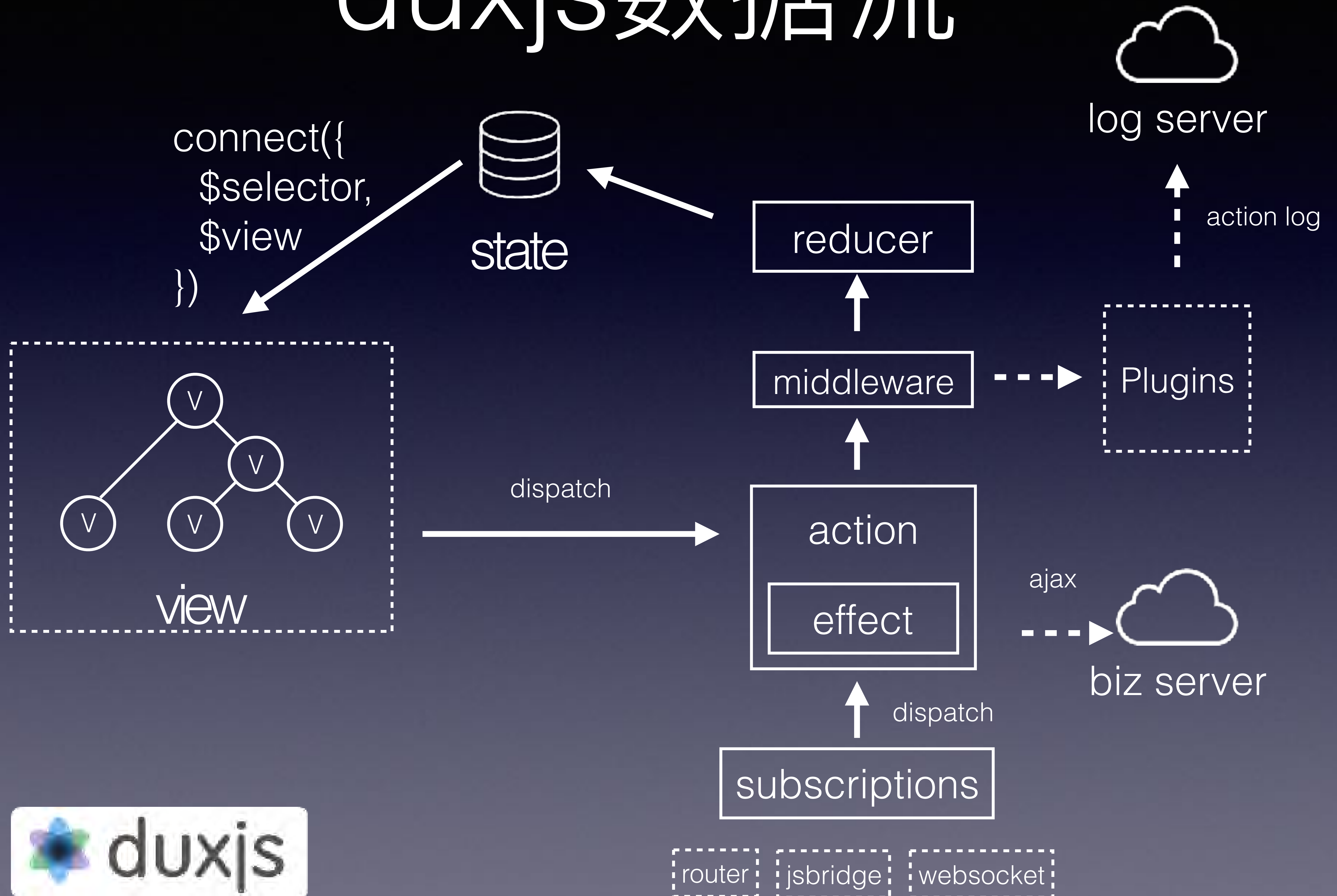
混合模式



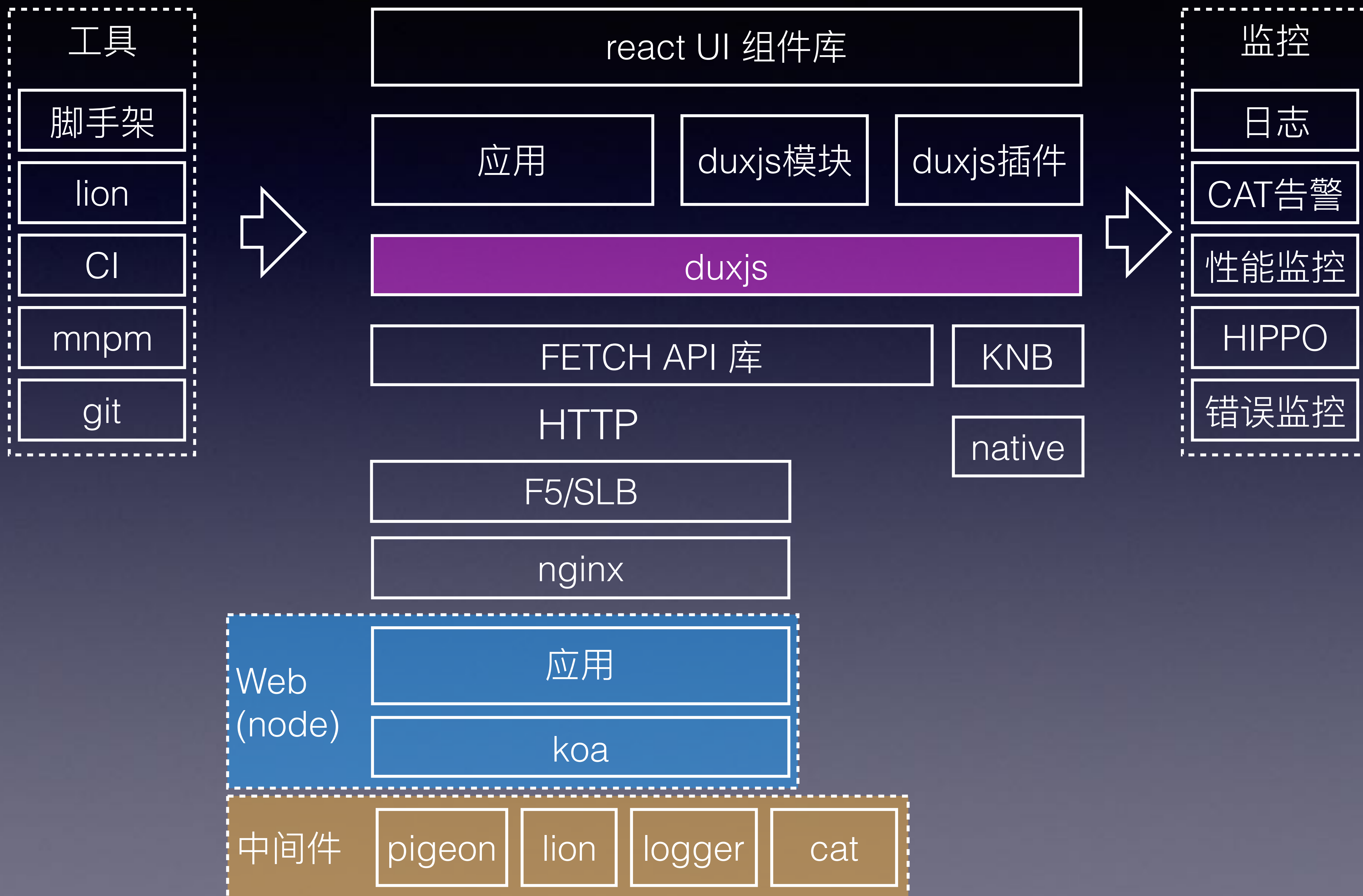
全承载模式



duxjs数据流



美团点评前端架构

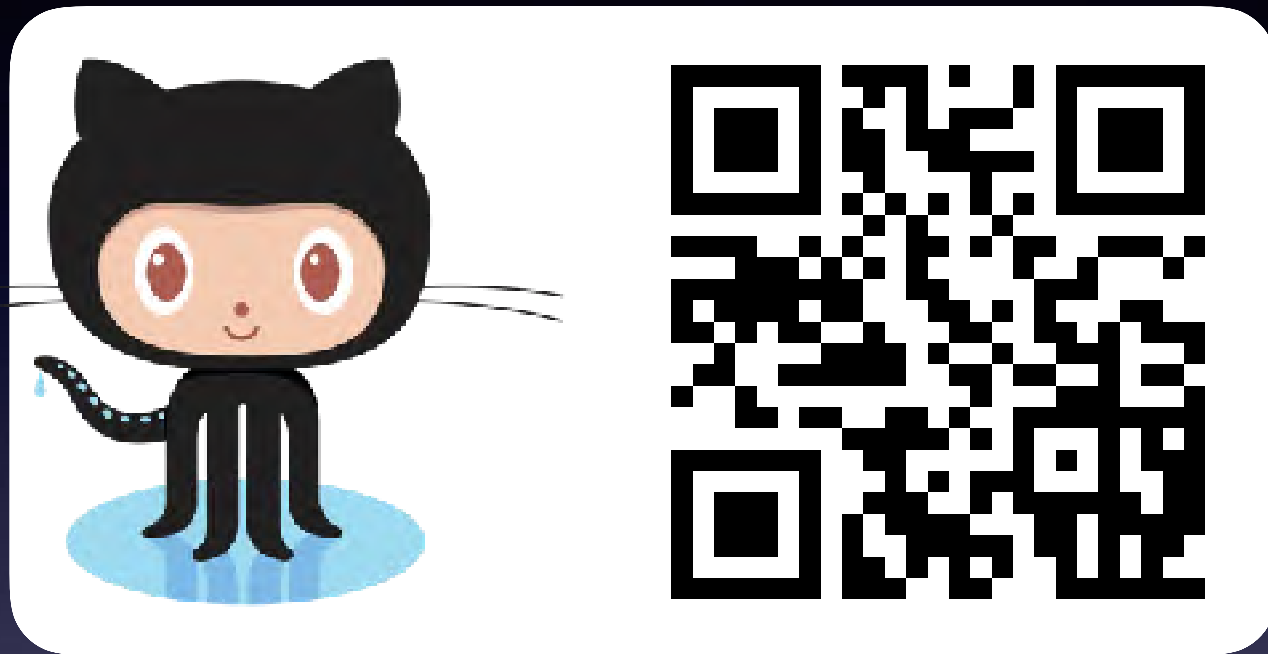


应用
duxjs模块
duxjs插件
duxjs
FETCH API 库
KNB
HTTP
native
F5/SLB
nginx
Web (node)
应用
koa
中间件
pigeon
lion
logger
cat
java service
mysql mysql mysql redis

duxjs现状

- 美团点评内测中
- 内测完毕后将会开源

duxjs



https://github.com/duxjs/duxjs

Thanks

- 张强
- 美团点评
- madlord.cn@gmail.com

Q&A