GMTC 全球移动技术大会
GLOBAL MOBILE TECH CONFERENCE

用 RxSwift 写
易维护易读的愉悦代码

宋旭陶

靛青K

严肃科技 iOS Developer

聚焦前沿技术　传递实践经验

主办方 Geekbang. 极客邦科技　InfoQ

异步

摘要

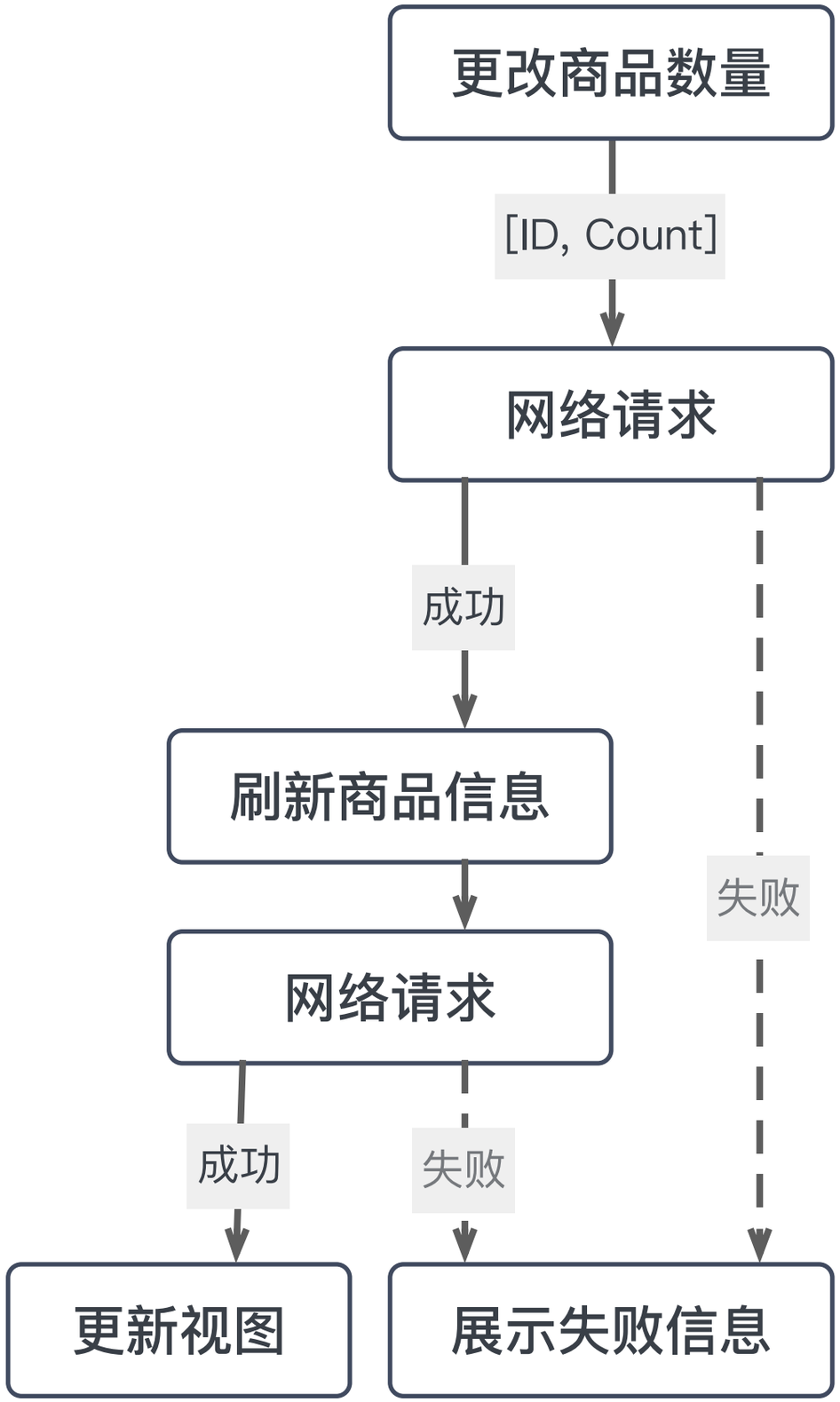
- 多个 API 串行调用
- 多个 API 并行调用
- 获取异步操作的状态
- ViewModel 做什么

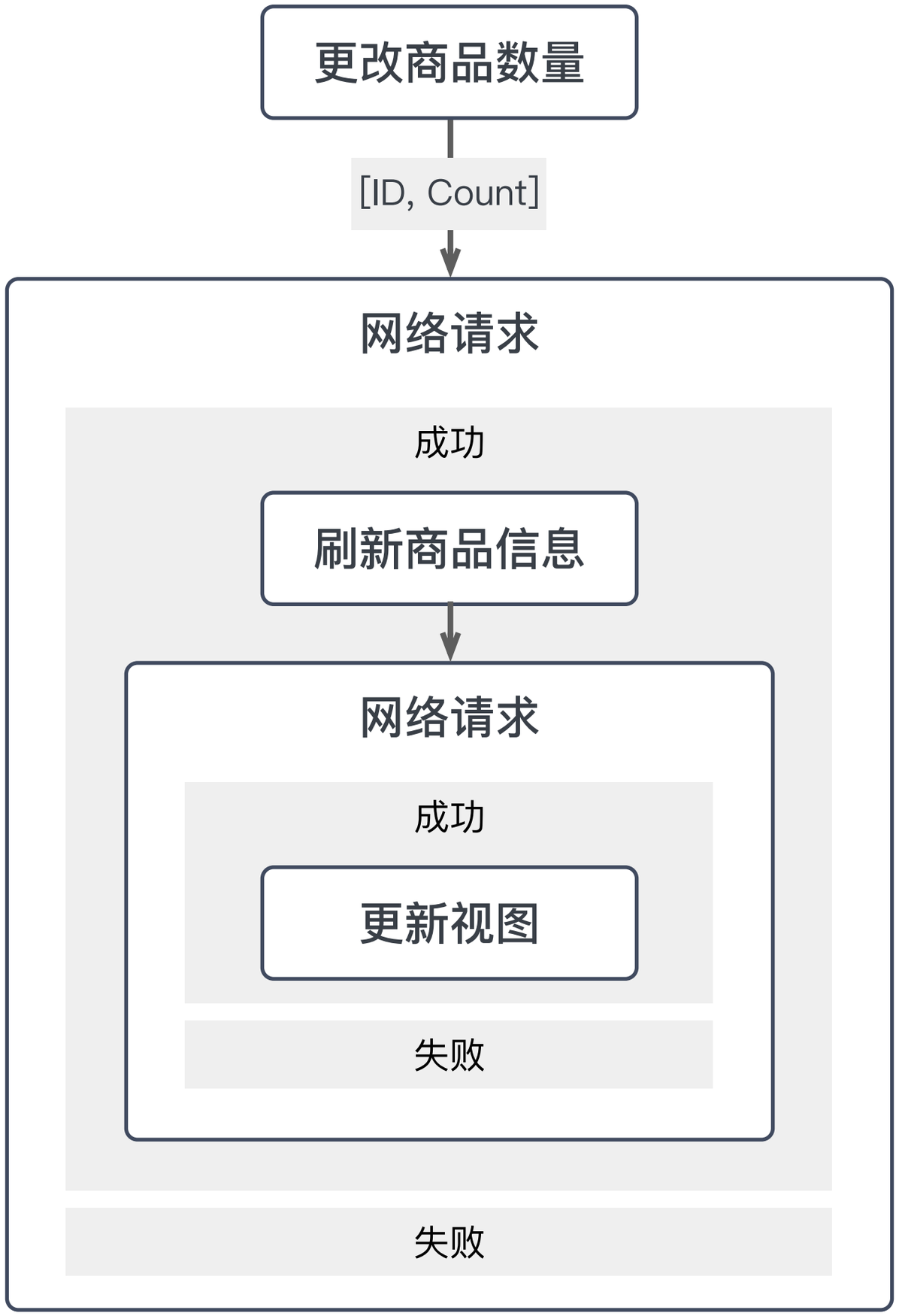
重构 ENJOY 购物车

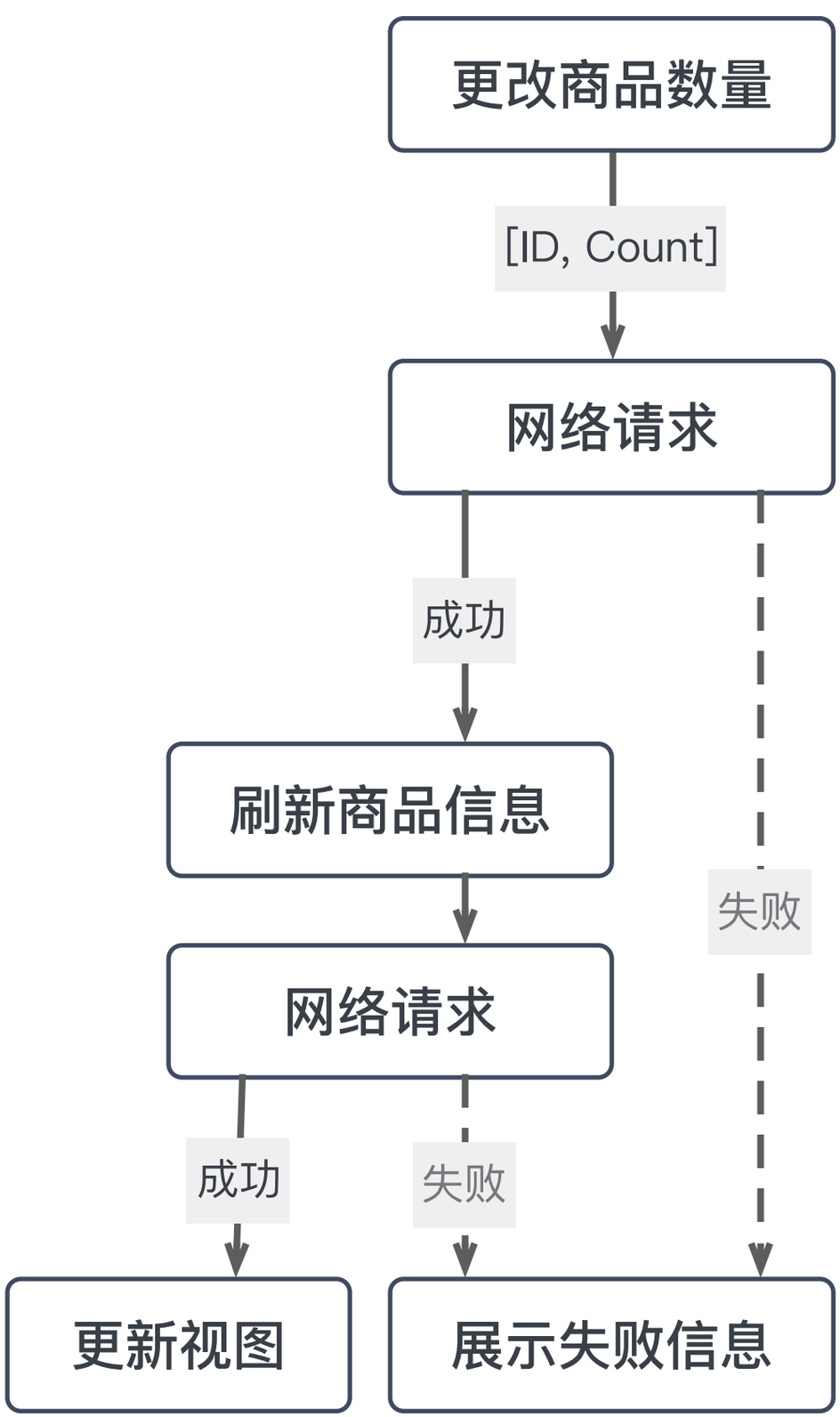
1. 多个 API 串行调用

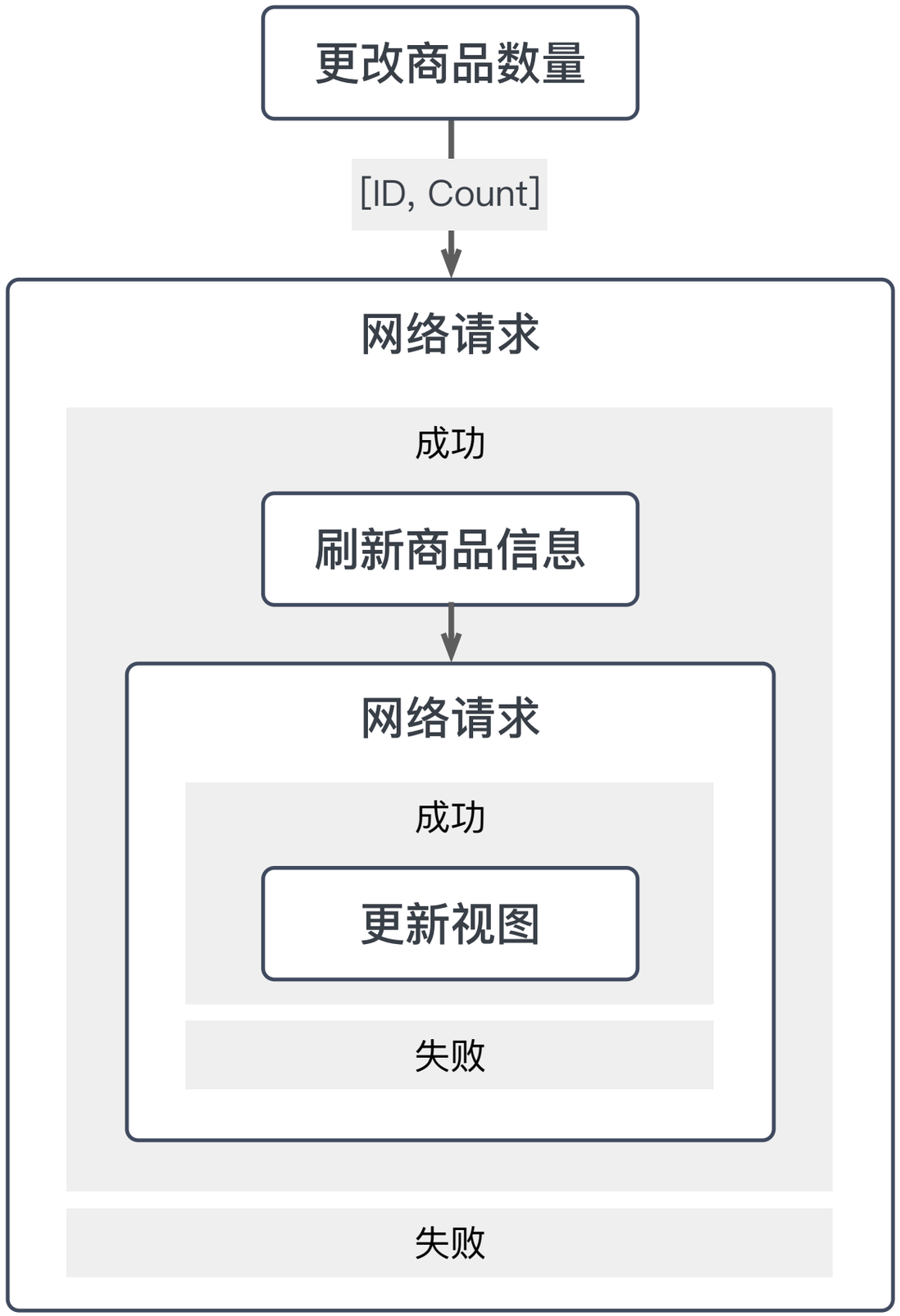
```swift
func change(productID: ID, count: Int)
{
    if isChangingProductCount
    {
        return
    }
    isChangingProductCount = true
    Network
        .requestChange(productID: ID, count: Int)
        { result in
            result
                .success
                { value in
                    Network.requestCart()
                    { result in
                        isChangingProductCount = false
                        result
                            .success
                            { value in
                                // 展示更改后的结果
                            }
                            .failure
                            { error in
                                // 展示刷新商品失败的信息
                            }
                    }
                }
                .failure
                { error in
                    // 展示更改商品失败的信息
                }
        }
}
```

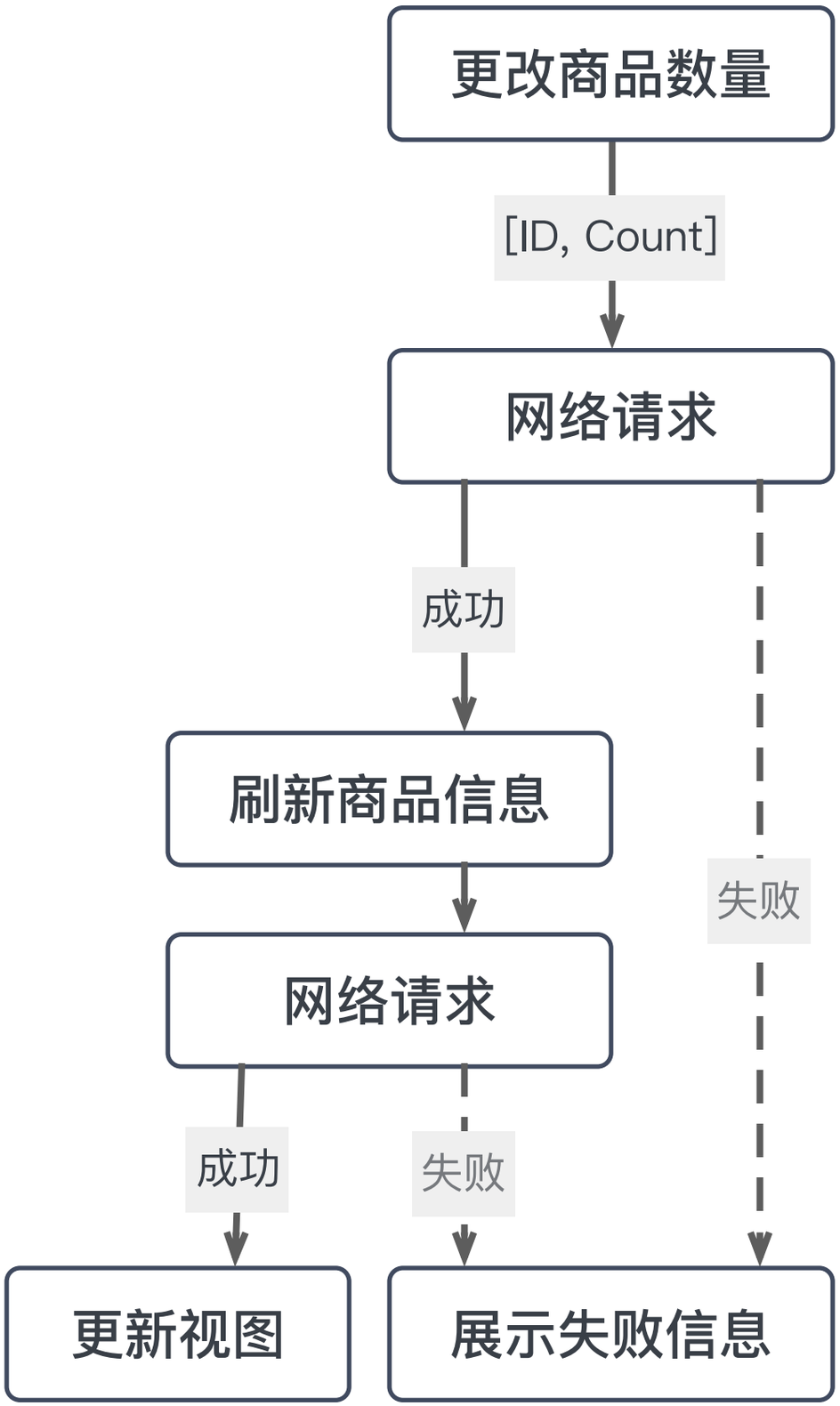
```swift
var isChangingProductCount: Bool
var isFetchingData: Bool

func change(productID: ID, count: Int)
{
    if isChangingProductCount
    {
        return
    }
    isChangingProductCount = true
    Network.requestChange(productID: ID, count: Int)
    { result in
        isChangingProductCount = false
        switch result
        {
            // 成功
            fetchData()
            // 失败
            // 展示错误
        }
    }
}
func fetchData()
{
    if isFetchingData
    {
        return
    }
    isFetchingData = true
    Network.requestCart()
    { result in
        isFetchingData = false
        switch result
        {
        // 成功
        //刷新数据

        // 失败
        // 展示错误
        }
    }
}
```


更改商品数量
[ID, Count]
网络请求
成功
失败
刷新商品信息
网络请求
成功
失败
更新视图
展示失败信息


更改商品数量
[ID, Count]
网络请求
成功
刷新商品信息
网络请求
成功
更新视图
失败
失败


更改商品数量
[ID, Count]
网络请求
成功
失败
刷新商品信息
网络请求
成功
失败
更新视图
展示失败信息

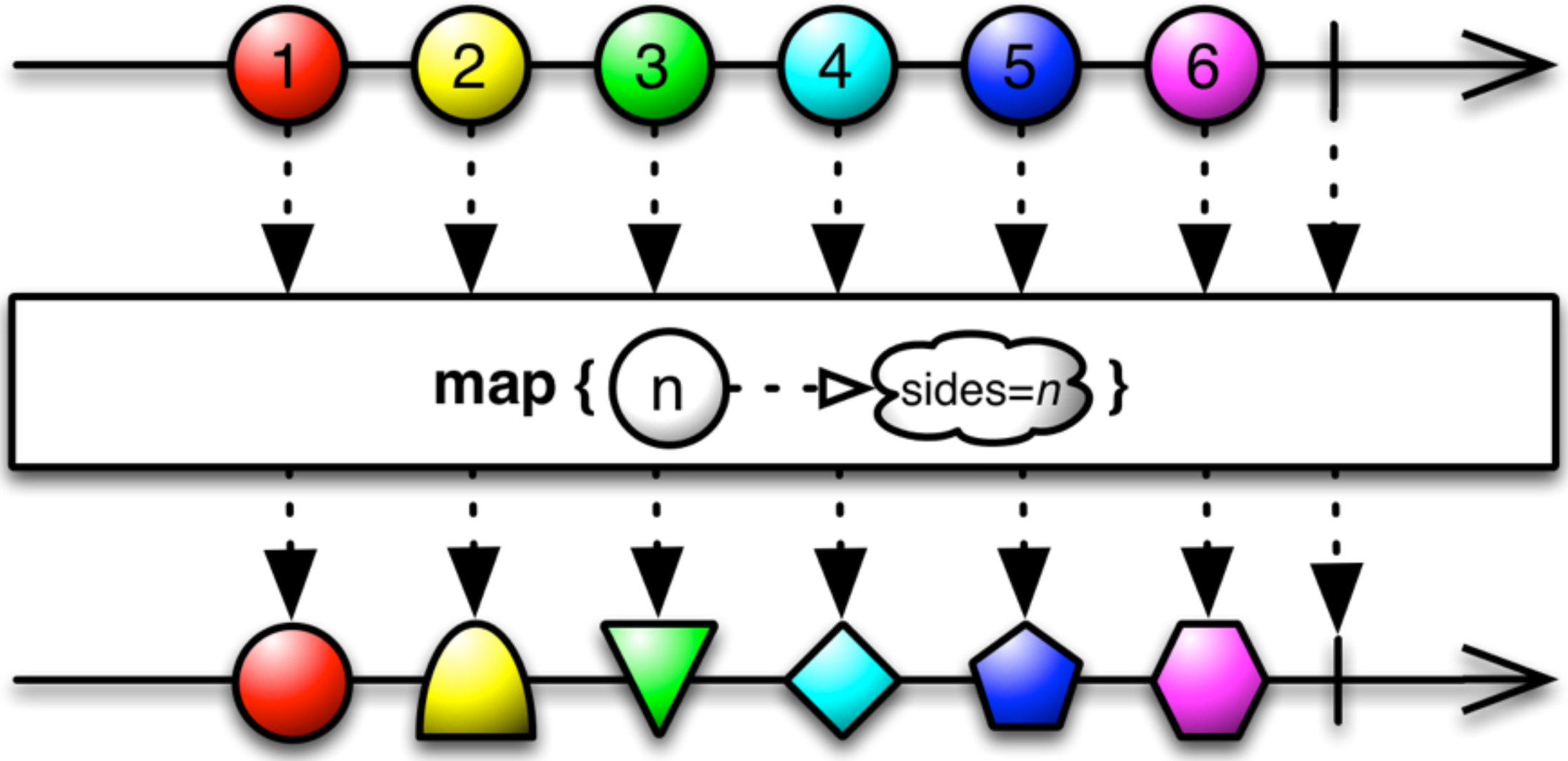

更改商品数量
[ID, Count]
网络请求
成功
刷新商品信息
网络请求
成功
更新视图
失败
失败




```
// 更改商品数量
changeProductCount

    // 网络请求
    .flatMap(requestChange)

    // 刷新商品信息 & 网络请求
    .flatMap(requestCart)

// 处理最终结果
.subscribe
{ result in
    switch result
    {
    case .Next(let value):
    // 更新视图
    case .Error(let error):
    // 展示失败信息
    }
}
```

烧脑时刻

Map



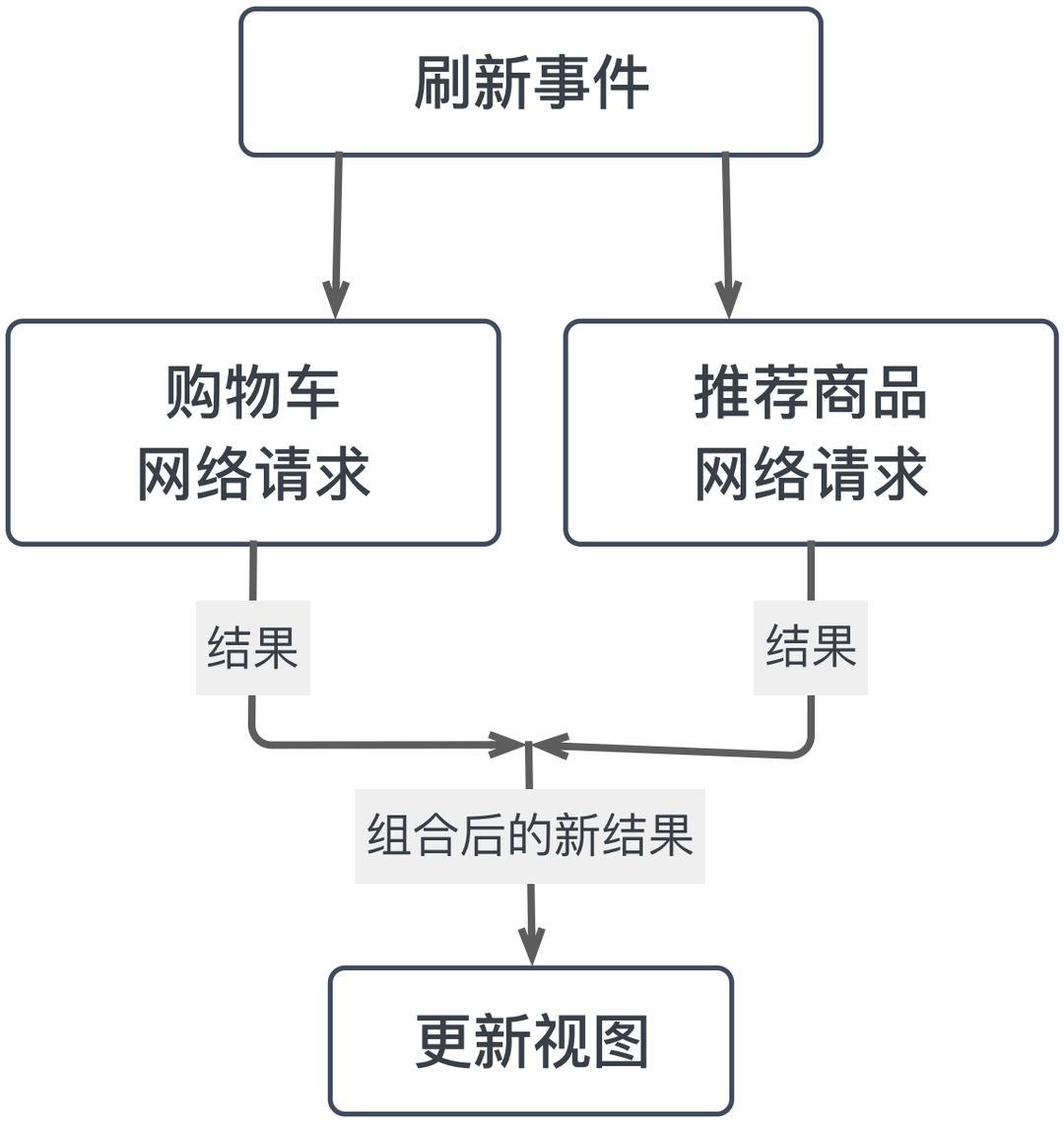
2. 多个 API 并行调用

购物车信息

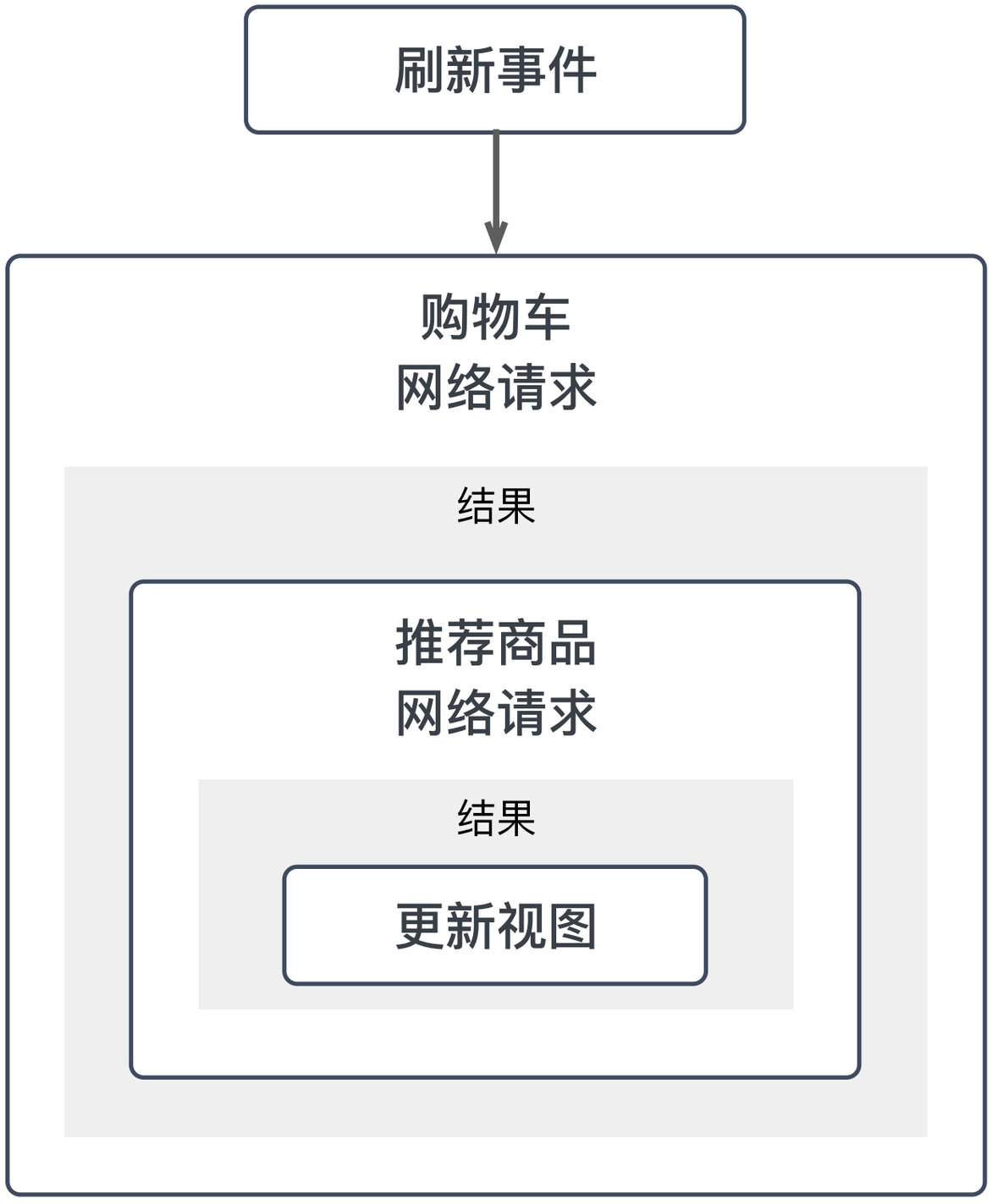
(requestCartInfo)

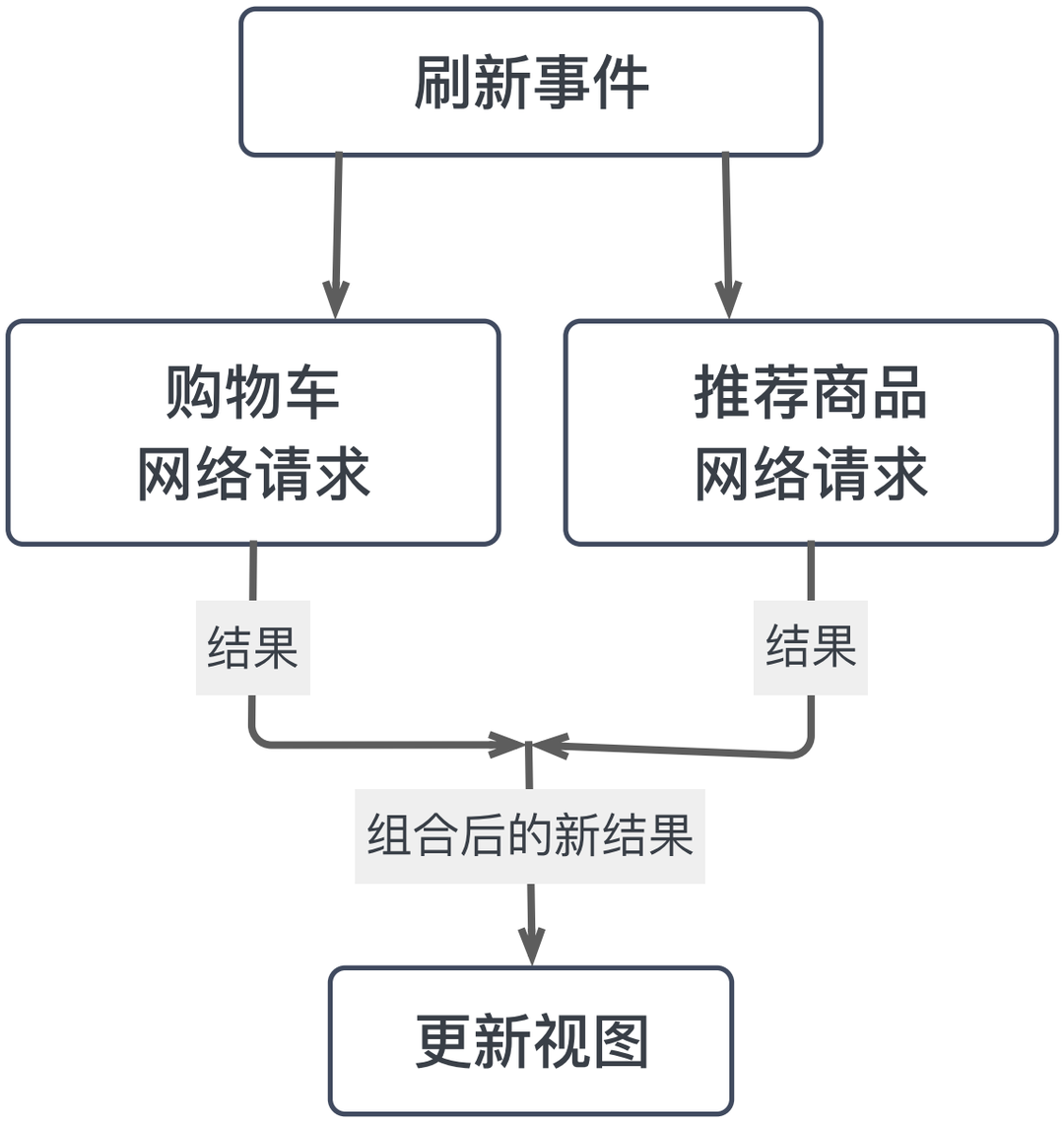
推荐商品

(requestRecommendProducts)


刷新事件
购物车
网络请求
推荐商品
网络请求
结果
结果
组合后的新结果
更新视图

```
func requestCart()
{
    if isRequesting
    {
        return
    }
    isRequesting = true
    Network
        .requestCartInfo()
        { result in
            // 处理购物车结果
            Network.requestRecommendPr
            oducts()
            { result in
                isRequesting = false
                // 处理推荐商品
            }
        }
}
```


刷新事件
购物车
网络请求
结果
推荐商品
网络请求
结果
更新视图

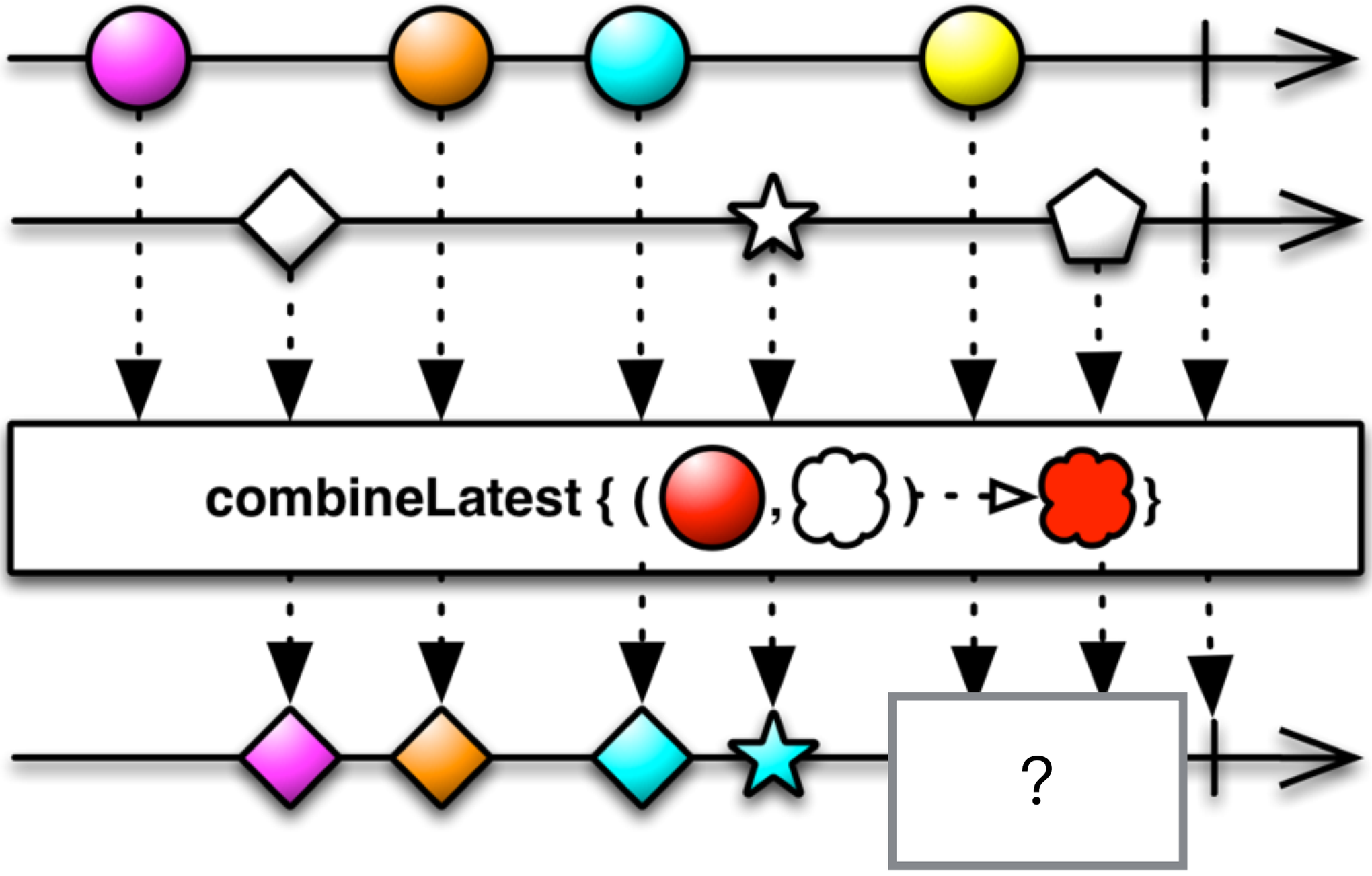

刷新事件
购物车
网络请求
推荐商品
网络请求
结果
结果
组合后的新结果
更新视图

CombineLatest
合并最新的结果

```
Observable
      .combineLatest(requestCartInfo, requestRecommendProducts)
      { cartInfoResult, recommendProducts in
        // 处理结果
        // ...
    return newResult
}
.subscribeNext
{ value in
       // 更新视图
}
```

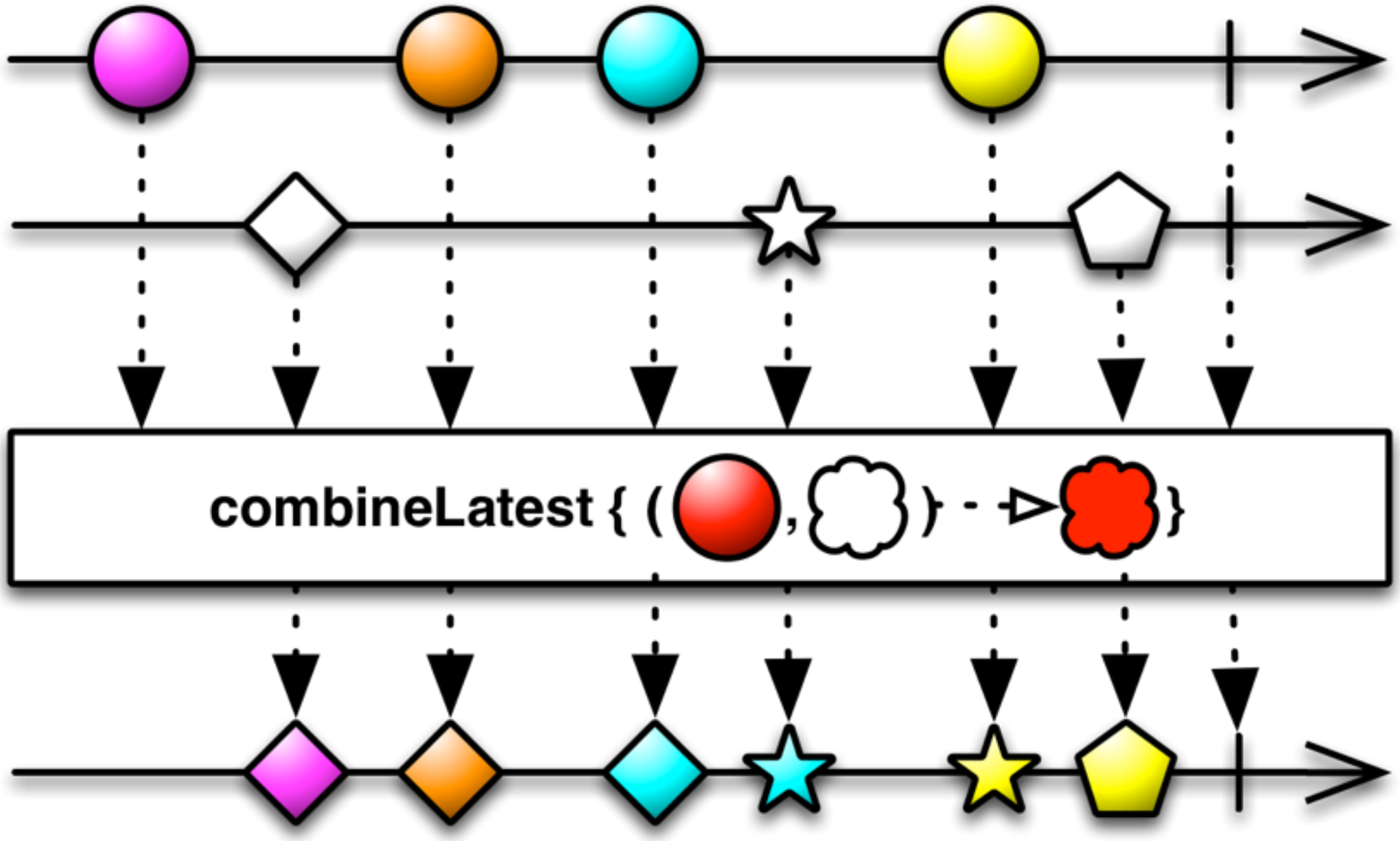
烧脑时刻

CombineLatest



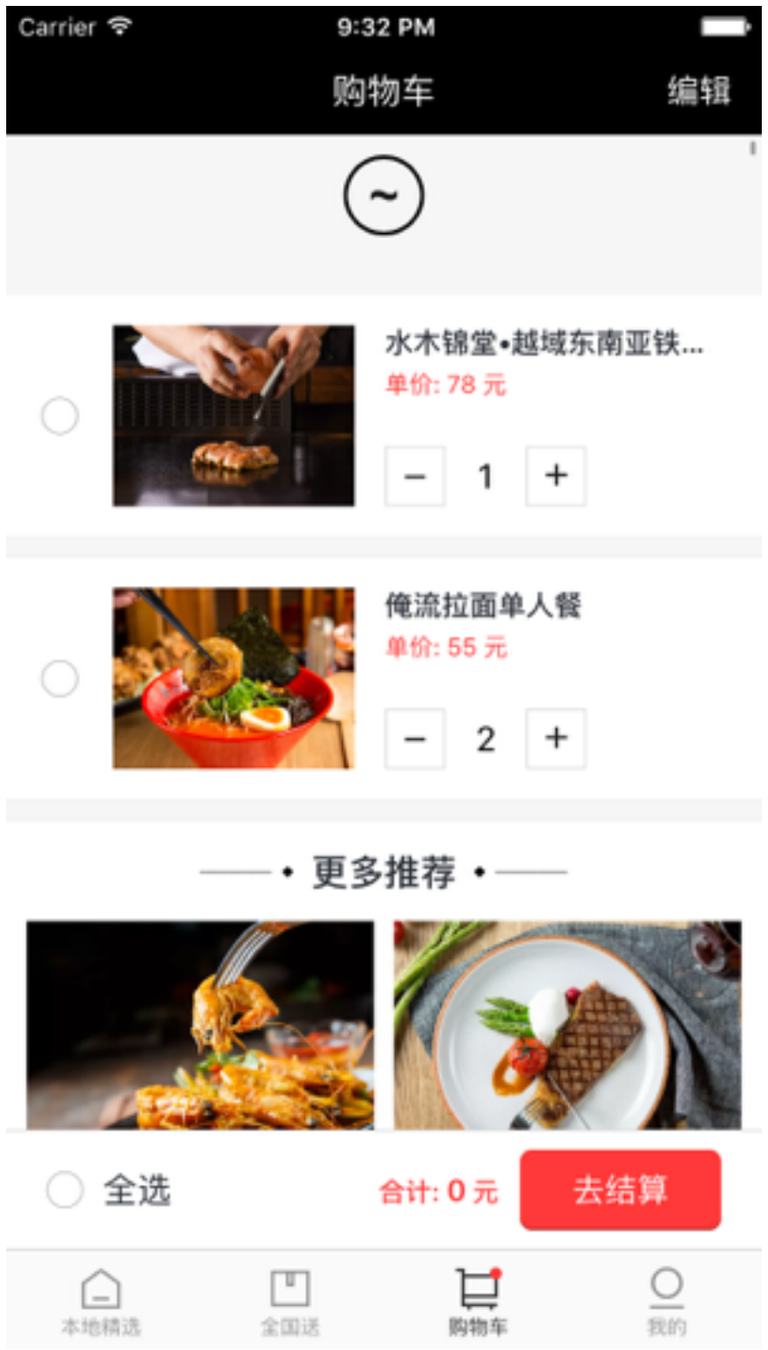
烧脑时刻

CombineLatest

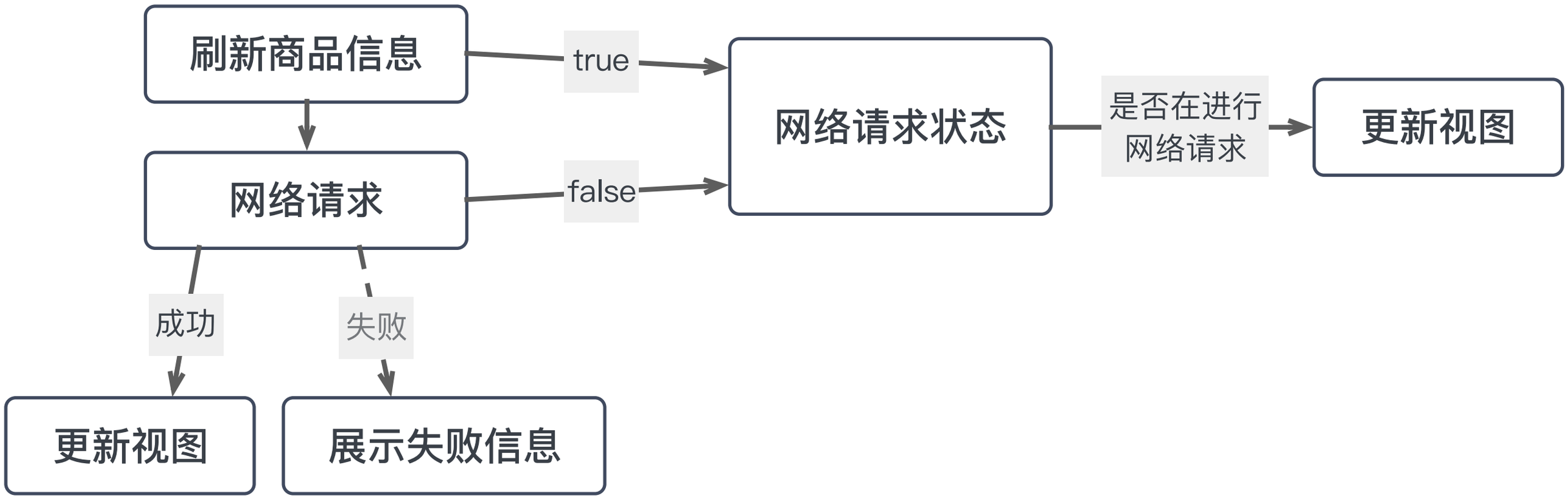


3. 获取异步操作的状态

刷新操作的状态

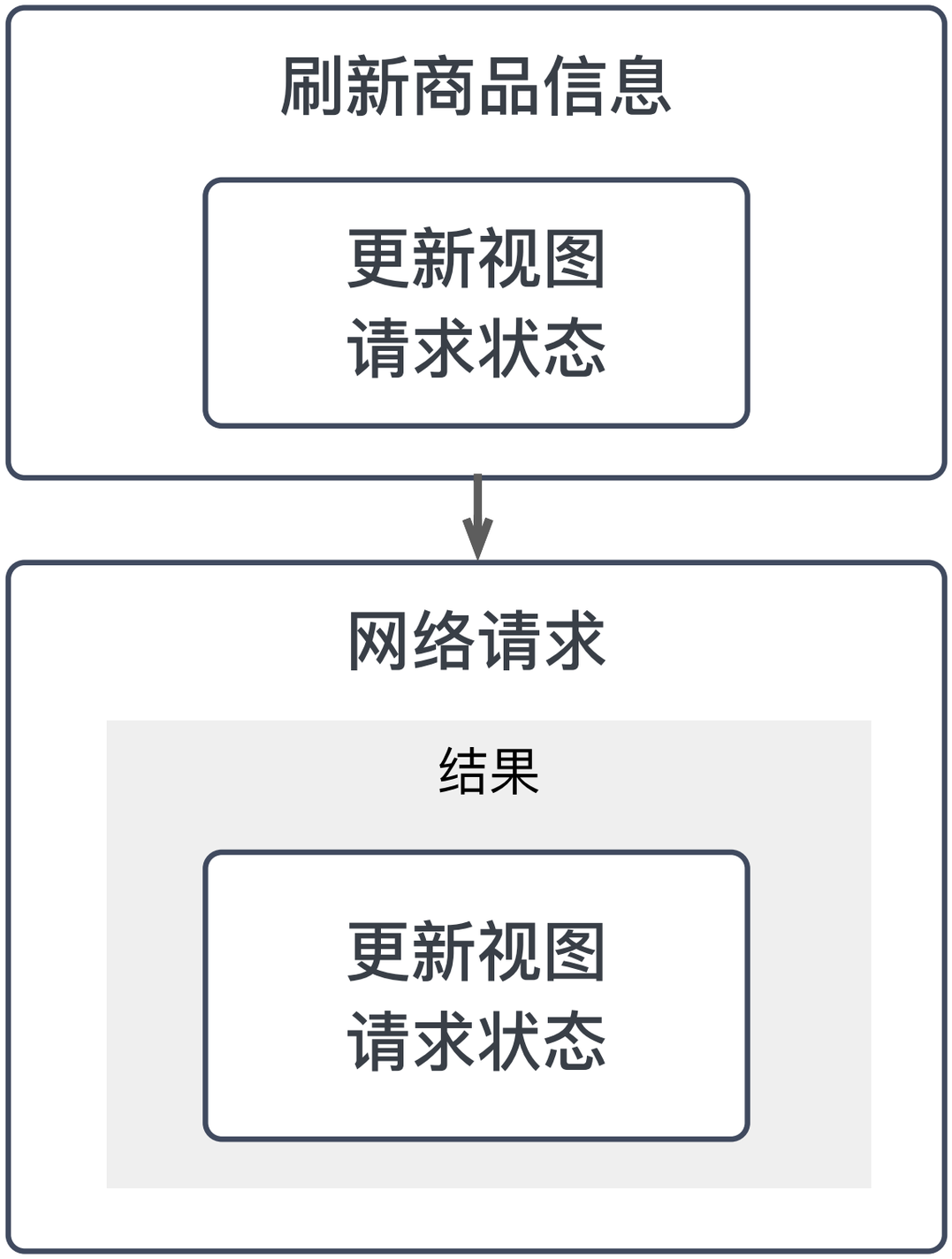

Carrier
9:32 PM
购物车
编辑
水木锦堂•越域东南亚铁...
单价: 78 元
1
俺流拉面单人餐
单价: 55 元
2
更多推荐
全选
合计: 0 元
去结算
本地精选
全国送
购物车
我的

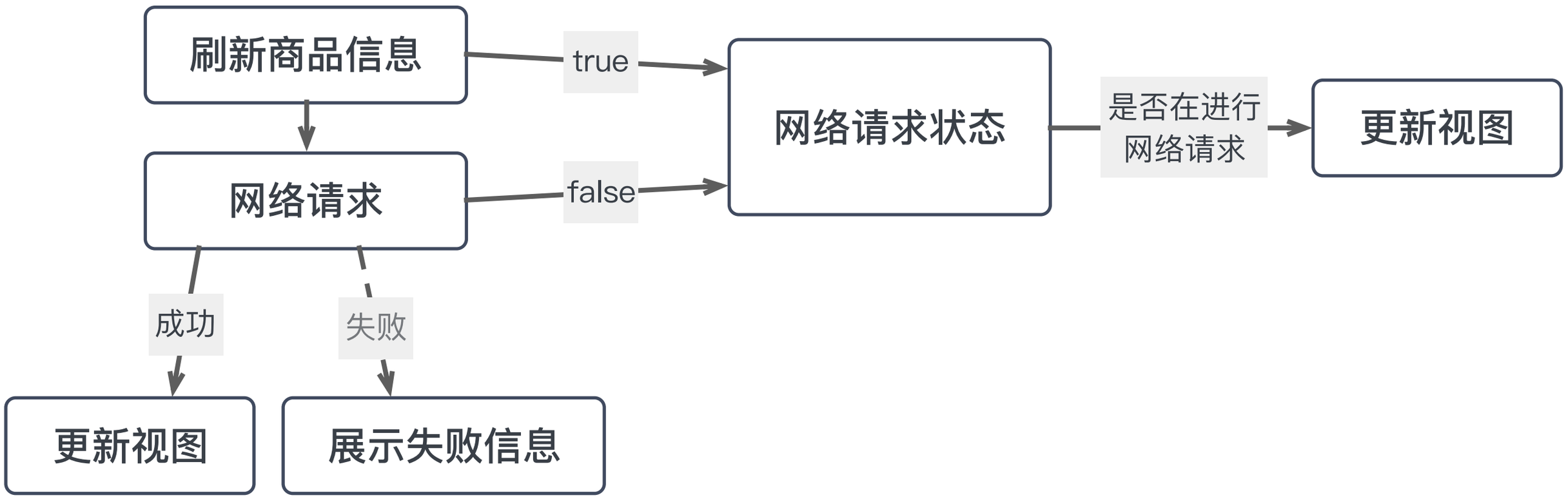
刷新操作的状态


刷新商品信息
网络请求
成功
失败
更新视图
展示失败信息
true
false
网络请求状态
是否在进行网络请求
更新视图

```swift
var isChangingProductCount: Bool
var isFetchingData: Bool

func change(productID: ID, count: Int)
{
    if isChangingProductCount
    {
        return
    }
    isChangingProductCount = true
    Network.requestChange(productID: ID, count: Int)
    { result in
        isChangingProductCount = false
        switch result
        {
            // 成功
            fetchData()
            // 失败
            // 展示错误
        }
    }
}
func fetchData()
{
    if isFetchingData
    {
        return
    }
    isFetchingData = true
    Network.requestCart()
    { result in
        isFetchingData = false
        switch result
        {
        // 成功
        //刷新数据

        // 失败
        // 展示错误
        }
    }
}
```

```
var isFetchingData: Bool

func fetchData()
{
    if isFetchingData
    {
        return
    }
    isFetchingData = true
        // 设置视图为加载中
    Network.requestCart()
    { result in
        isFetchingData = false
        // 设置视图为加载完成
        switch result
        {
        // 成功
        //刷新数据

        // 失败
        // 展示错误
        }
    }
}
```


刷新商品信息
更新视图
请求状态
网络请求
结果
更新视图
请求状态

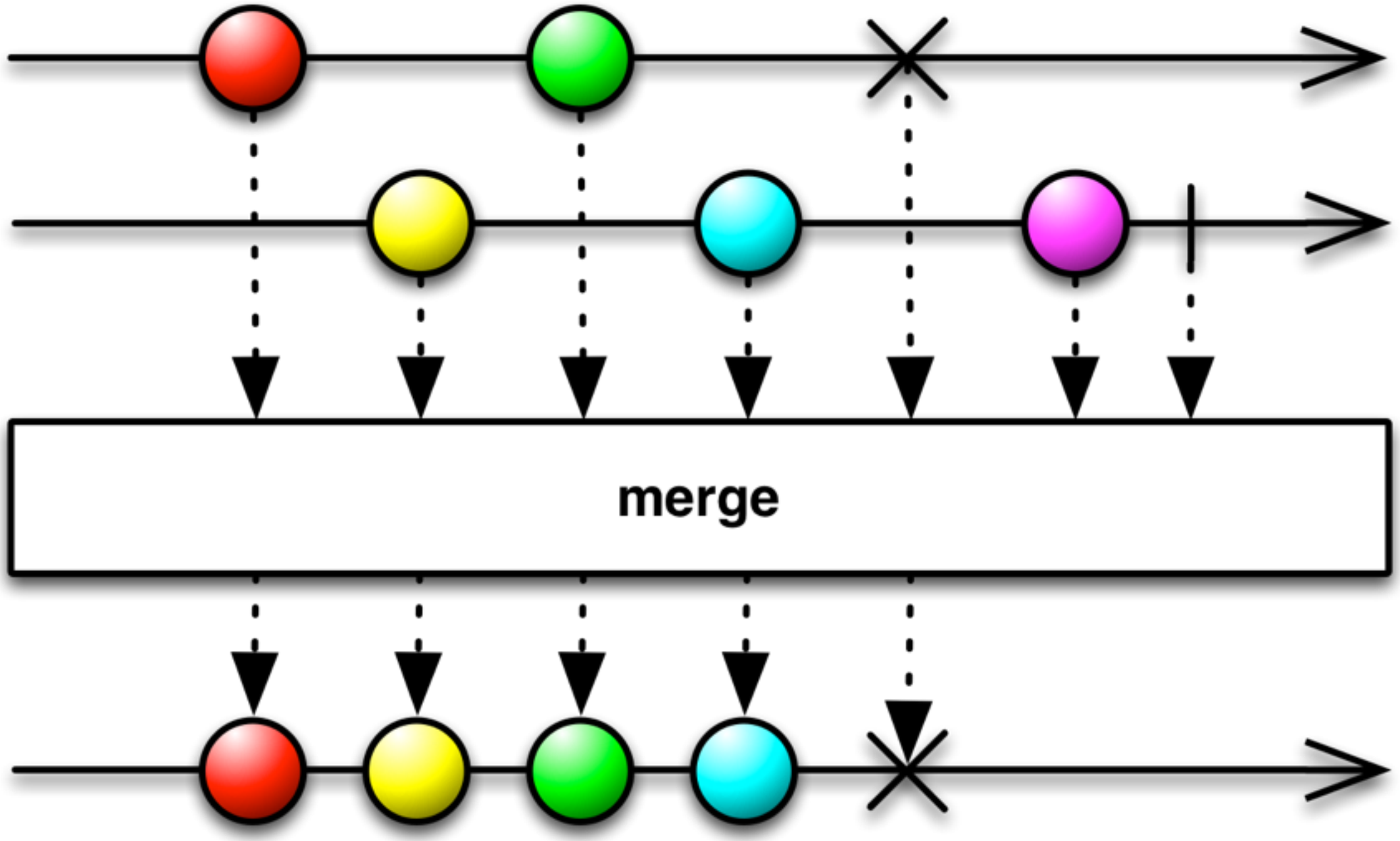

刷新商品信息
true
网络请求状态
是否在进行网络请求
更新视图
网络请求
false
成功
失败
更新视图
展示失败信息

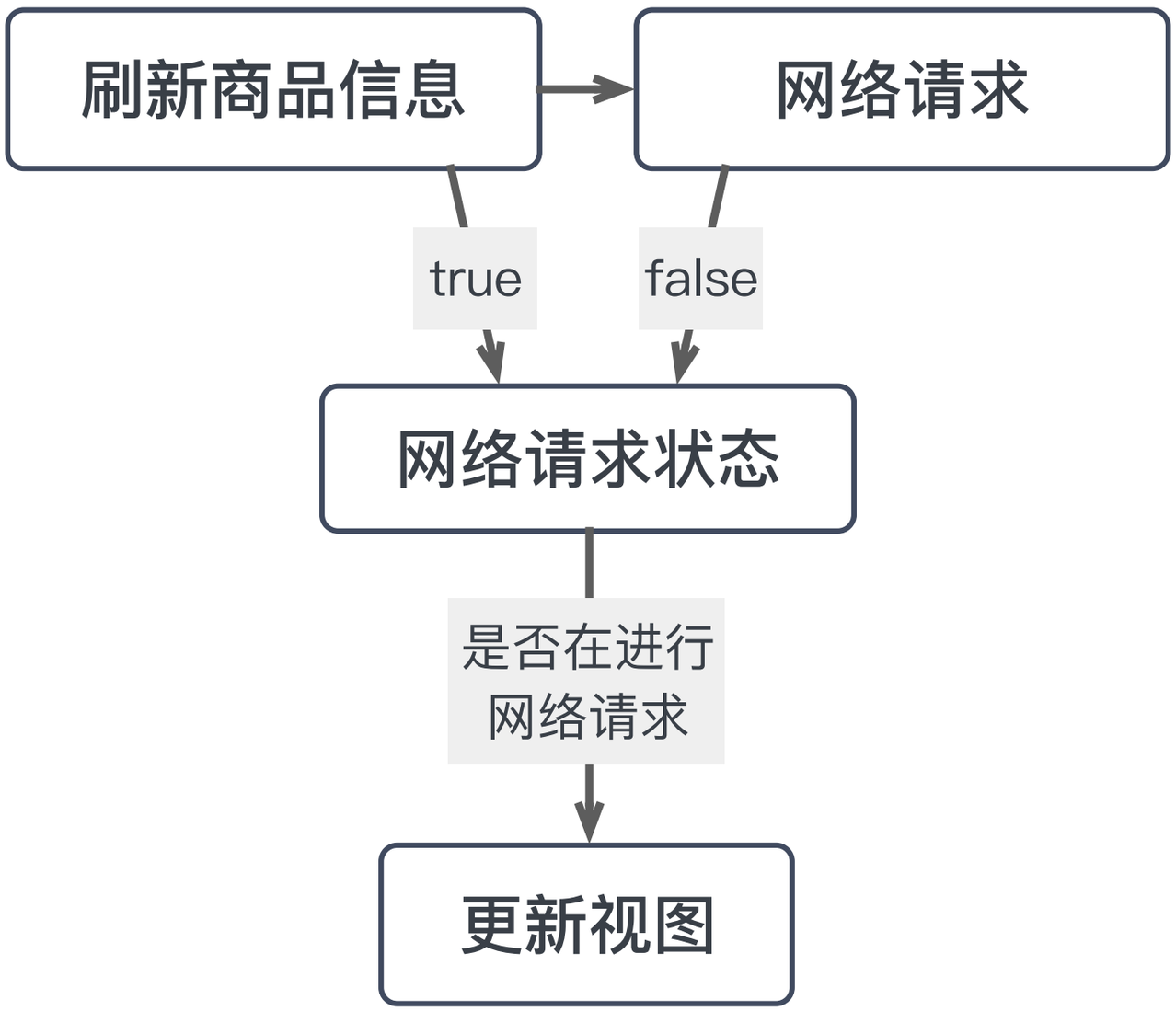

true
false
true
...

```
[
    requestCartInfo.map { _ in true },
    response.map { _ in false }
]

    .toObservable()
    .merge()

    .subscribeNext
    { isRefreshing in
        // 更新视图
    }
```

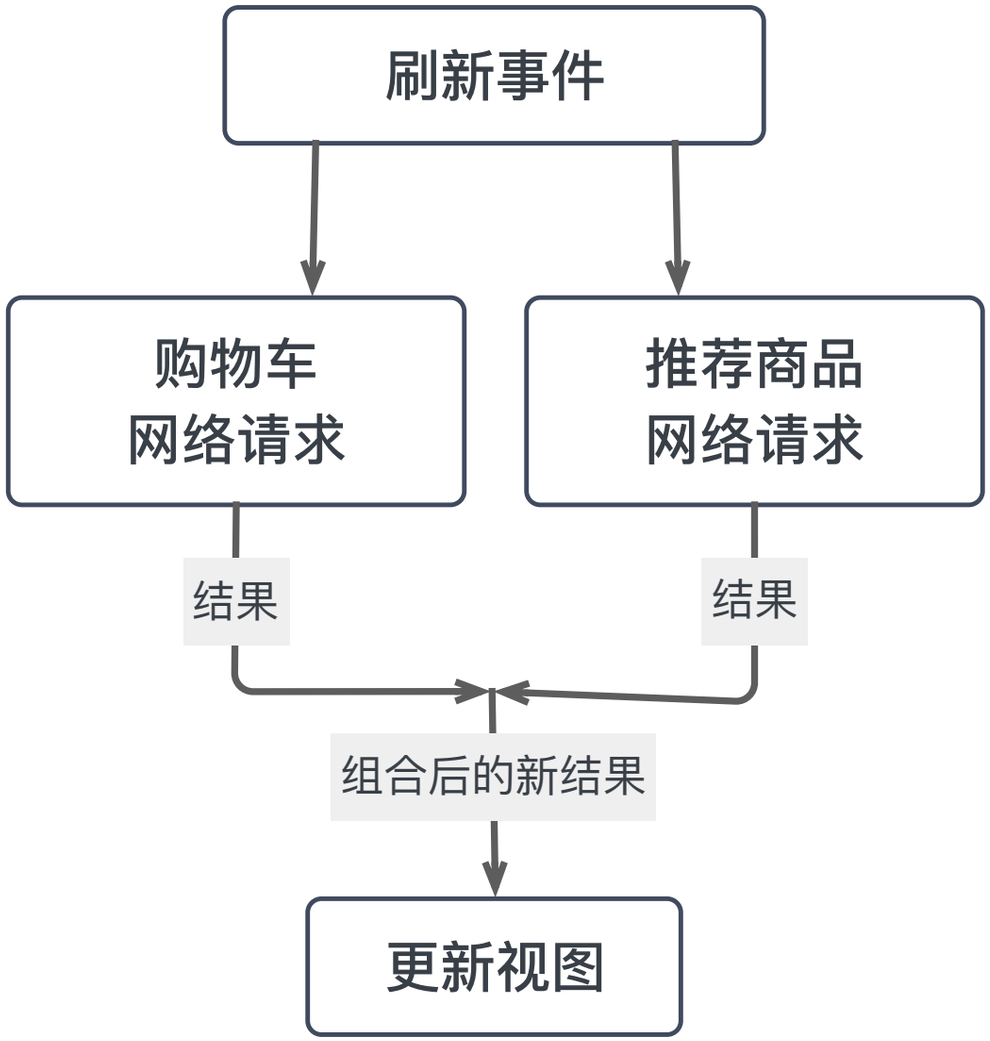
烧脑时刻

Merge

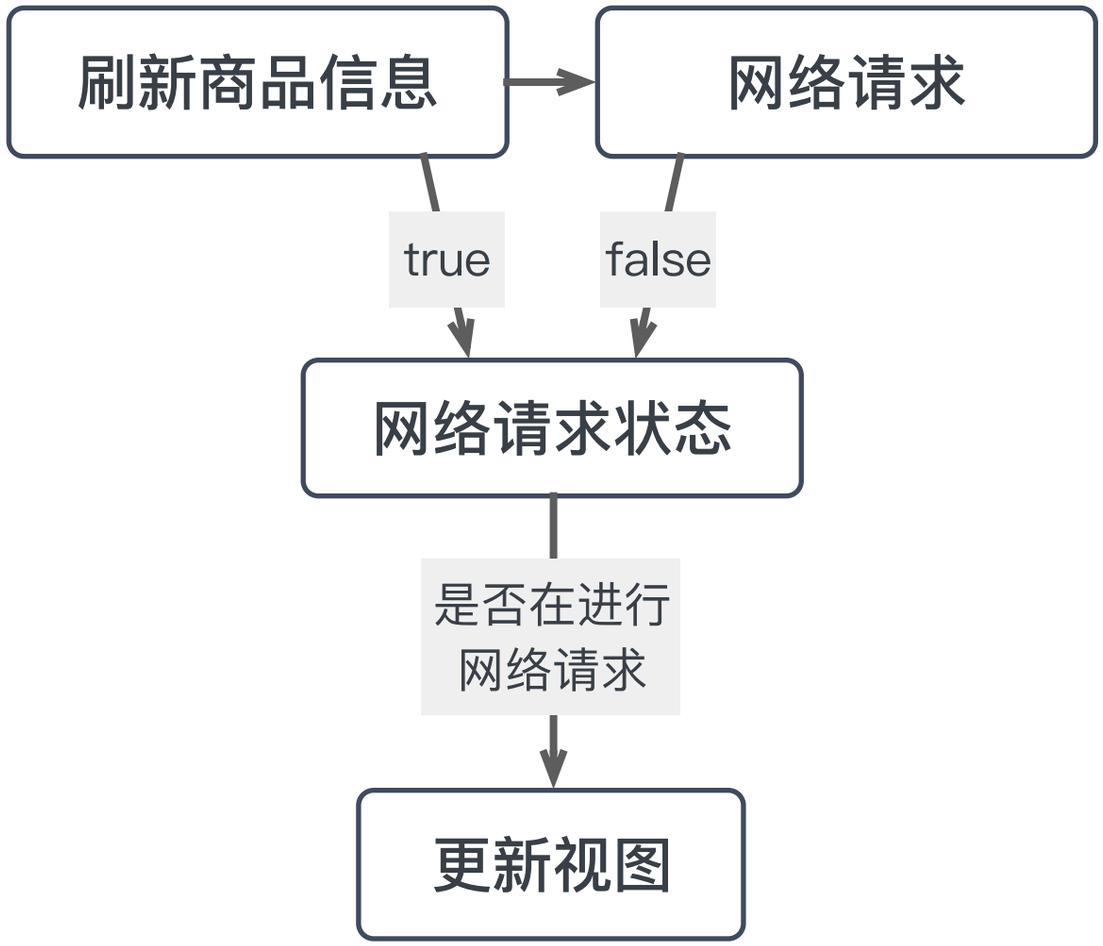



刷新商品信息
网络请求
true
false
网络请求状态
是否在进行
网络请求
更新视图

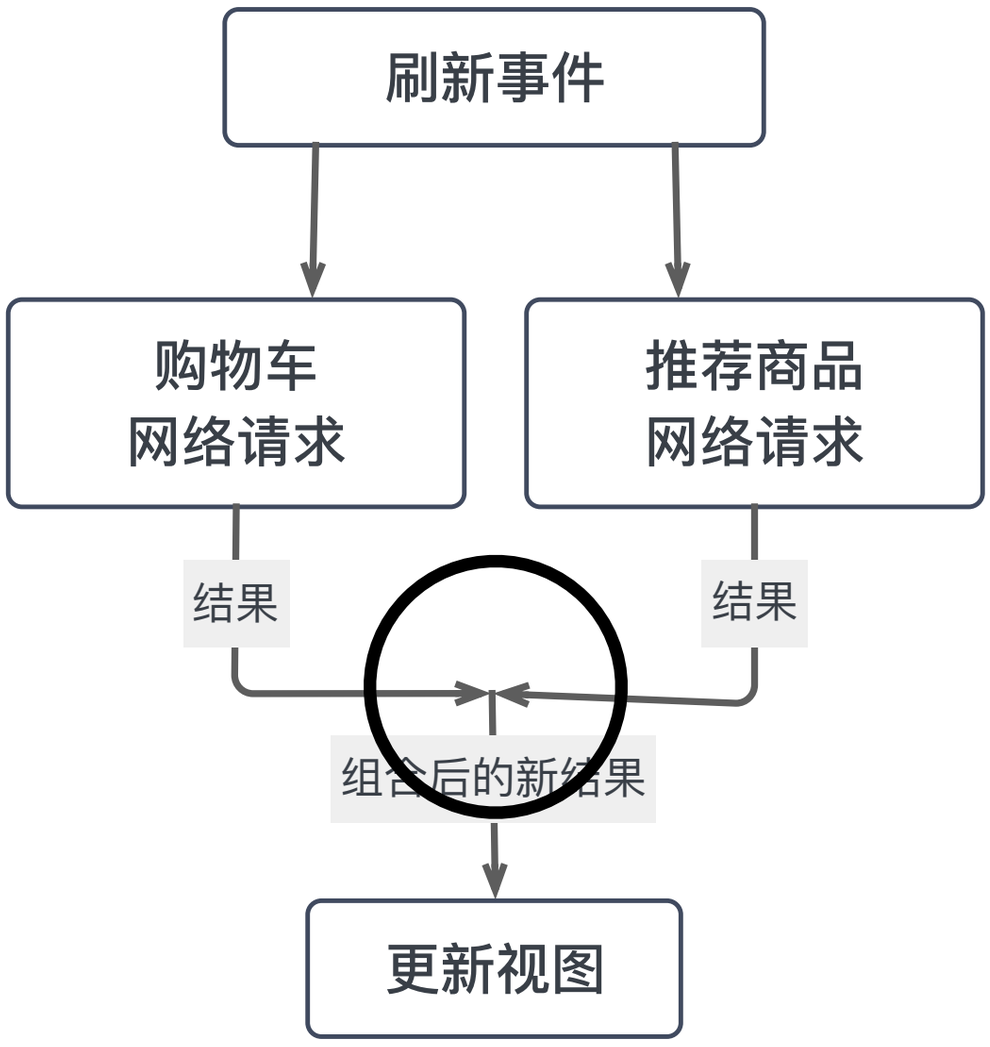
CombineLatest



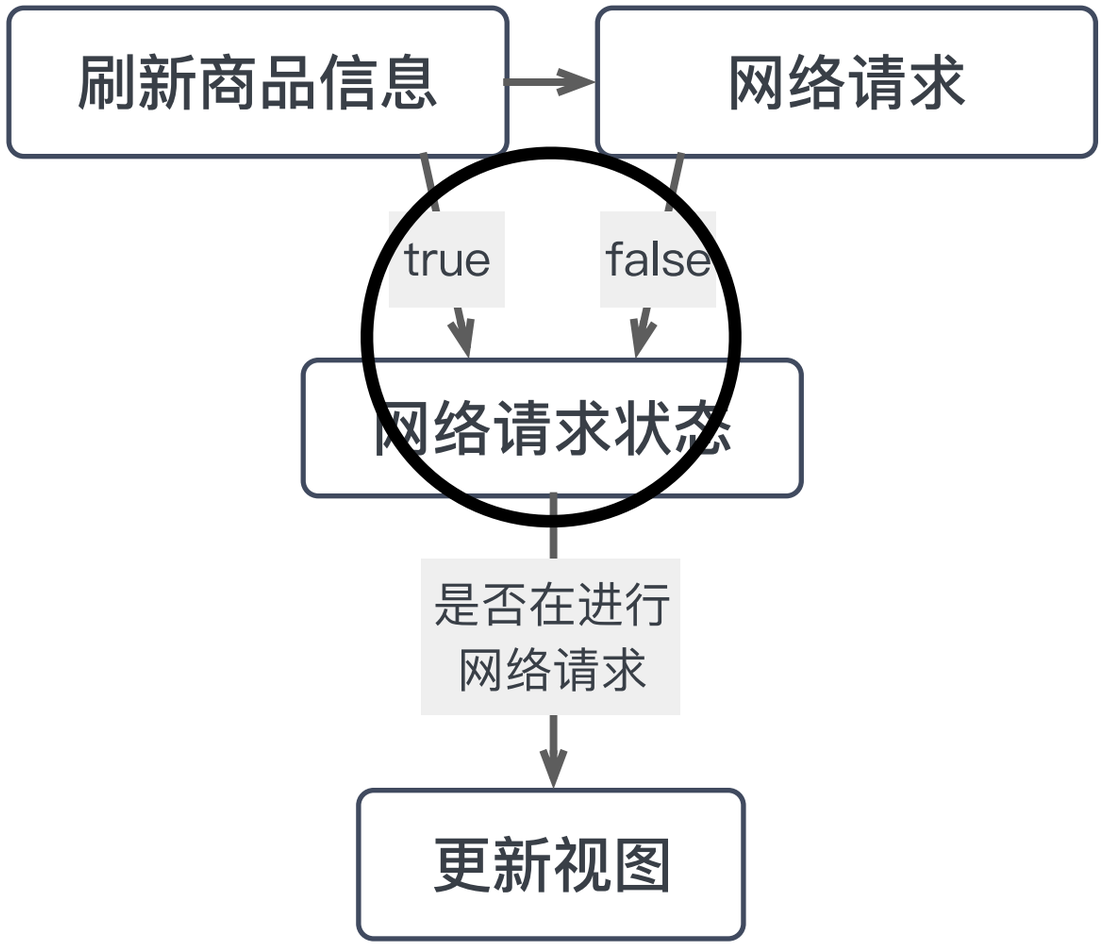
Merge



CombineLatest

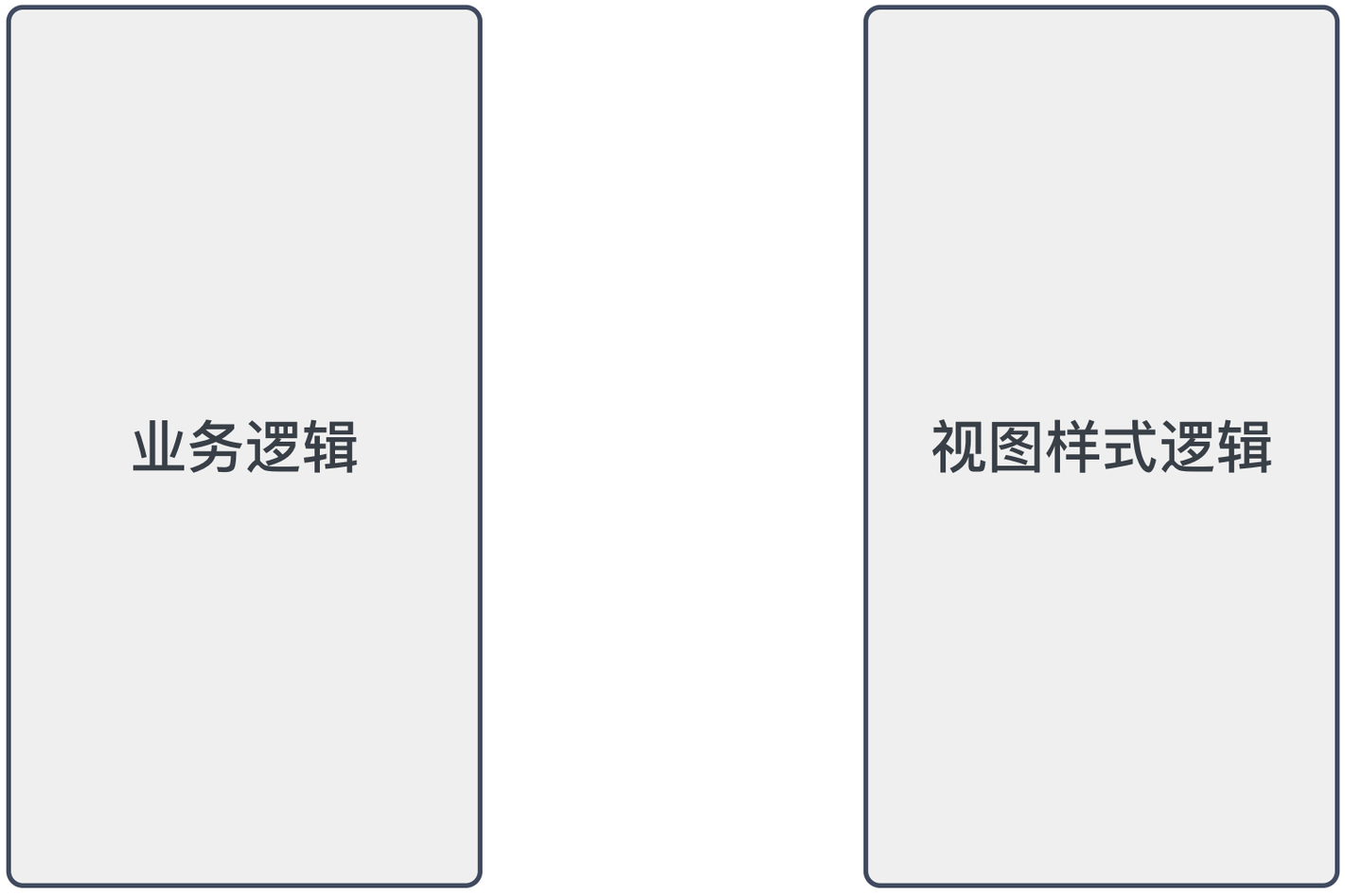


Merge



4. ViewModel 做什么

分离业务逻辑和视图样式的代码



分离业务逻辑和视图样式的代码

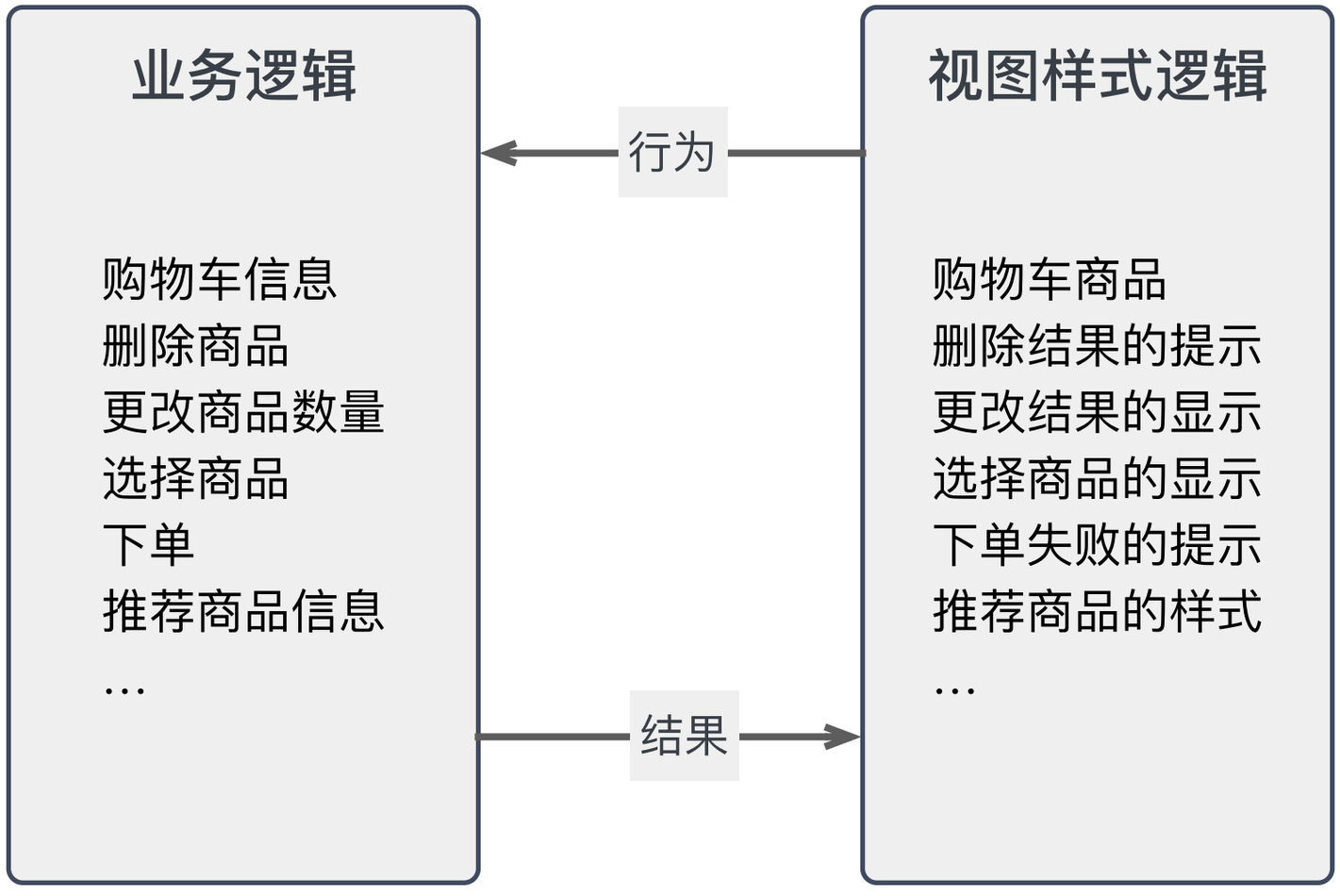
业务逻辑

购物车信息
删除商品
更改商品数量
选择商品
下单
推荐商品信息
…

视图样式逻辑

购物车商品
删除结果的提示
更改结果的显示
选择商品的显示
下单失败的提示
推荐商品的样式
…

分离业务逻辑和视图样式的代码



ViewController
确定触发事件方案

```
// 更改商品数量
changeProductCount
```

ViewModel
处理内部逻辑

```
// 网络请求
.flatMap(requestChange)

// 刷新商品信息 & 网络请求
.flatMap(requestCart)
```

ViewController
展示视图

```
// 处理最终结果
.subscribe
{ result in
    switch result
    {
    case .Next(let value):
    // 更新视图
    case .Error(let error):
    // 展示失败信息
    }
}
```

```
class CartViewModel
{

    let elements = Observable<[CartProductModel]>
    /// 删除商品结果，是否删除成功
    let deleteResult: Observable<APIResult<Bool>>
    /// 下订单结果
    let confirmResult: Observable<CartConfirmResult>
    /// 是否在刷新数据
    let isRefreshing = Variable(false)

    init(input:
        (
        /// 刷新触发事件
        refreshTrigger: Observable<Void>,
        /// 加载触发事件
        loadTrigger: Observable<Void>,
        /// 下订单触发事件
        confirmTrigger: Observable<Void>,
        /// 选择商品触发事件
        selectProductsTrigger: Observable<[(subProductID: Int64, selected: Bool)]>,
        // 改变商品数量触发事件
        changeProductCountTrigger: Observable<(subProductID: Int64, count: Int)>,
        // 删除商品触发事件
        deleteProductsTrigger: Observable<[Int64]>,
        )
    ) { … }
}
```

总结

- 多个 API 串行调用
- 多个 API 并行调用
- 获取异步操作状态
- 异步时间上相关
- 异步时间上不相关
- 关心异步状态

必须用 RxSwift 吗

~~肯定用啊~~

RxSwift 的优势

- 优雅的处理异步
- 无状态的数据流
- 跨平台、跨语言

RxSwift 的劣势

- 学习曲线
- 团队协作
- 难调试

RxSwift 的适用场景

- 登录/注册
- 购物车
- 音乐播放
- …

More

- RxSwift
- ReactiveX
- Airbnb: 我们安卓客户端是如何使用 RxJava 的
- RxMarbles
- RxDataSources
- RxVisualDebugger
- Expert to Expert: Brian Beckman and Erik Meijer – Inside the .NET Reactive Framework (Rx)
- Swift 烧脑体操（四） – map 和 flatMap

“代码写的开心才有心情继续写代码。”

— DianQK

GMTC 全球移动技术大会
GLOBAL MOBILE TECH CONFERENCE

Let’s enjoy RxSwift.

Q&A

聚焦前沿技术　传递实践经验

主办方 Geekbang. 极客邦科技　InfoQ