Архитектура сервисного слоя

с использованием составных операций

Глеб Новик
Ведущий разработчик iOS



Tinkoff Bank

Предпосылки

- Распространенная задача
- Путаница в терминологии
- Интерес со стороны сообщества
- Недооценённость операций в iOS SDK

Предпосылки

Rambler.Mail - email client and task manager.

- Created the new core layer: networking, mapping, working with database
- Designed and implemented the business logic of the todo task manager unit with the use the "compound operation" approach

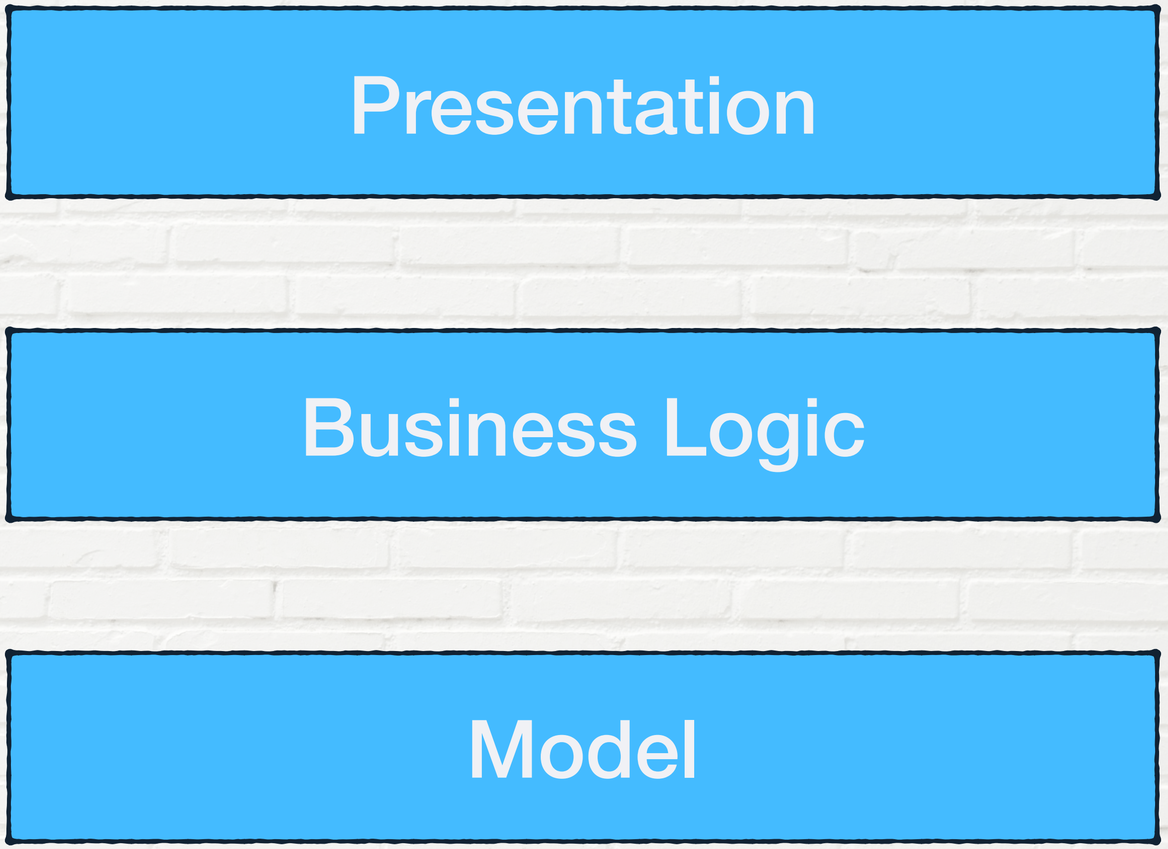
Предпосылки

- Распространенная задача
- Путаница в терминологии
- Интерес со стороны сообщества
- Недооценённость операций в iOS SDK

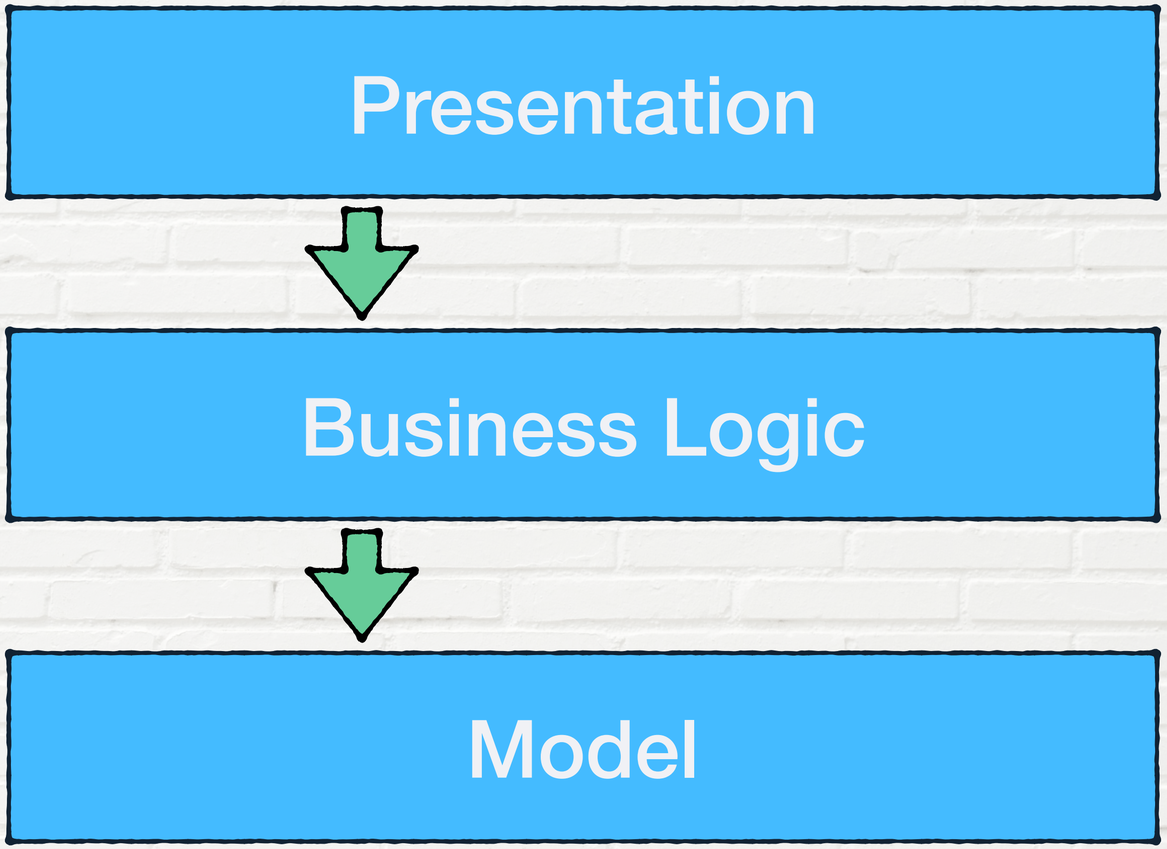
План

- Систематизируем терминологию о бизнес-логике
- Спроектируем сервисную часть iOS-приложения
- Доработаем и будем использовать NSOperation

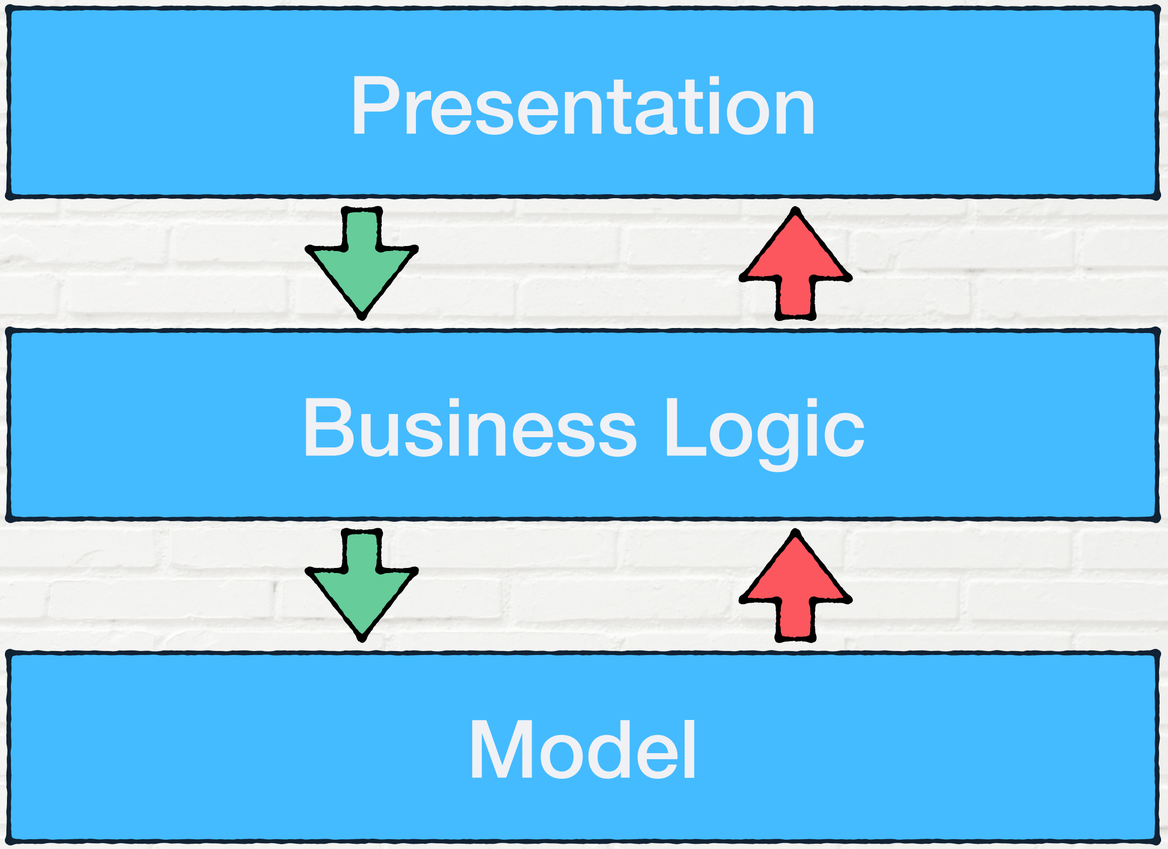
Слоистая архитектура



Слоистая архитектура

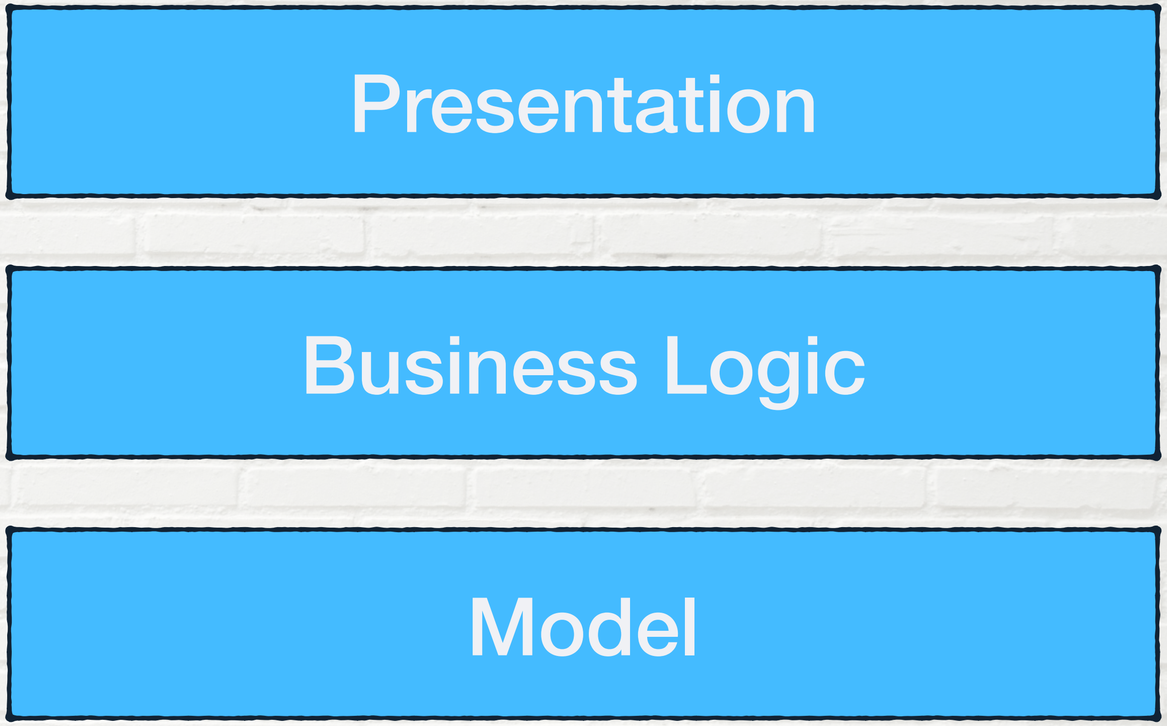


Слоистая архитектура

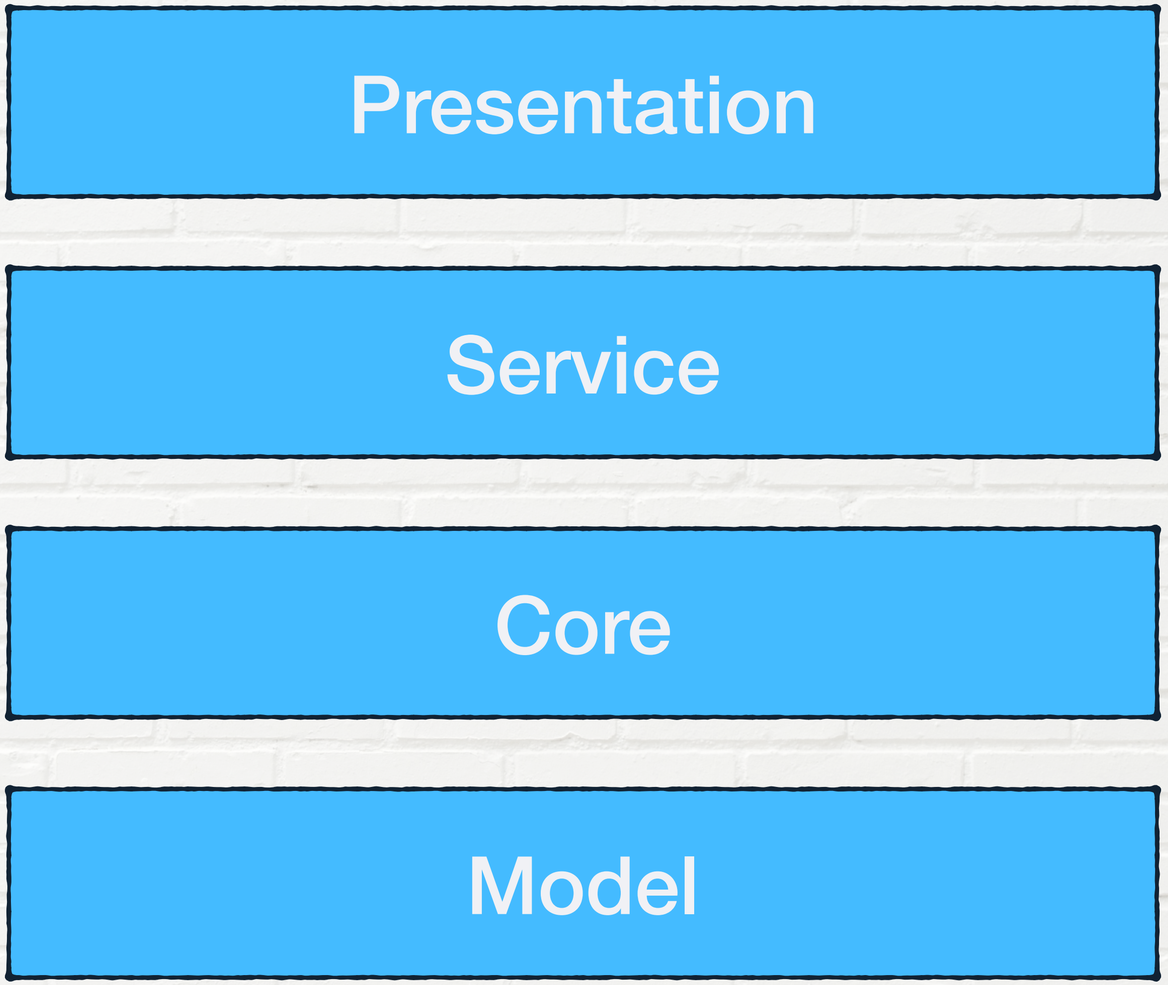


Слоистая архитектура

- Слабосвязный код
- Параллельная разработка
- Ясная структура проекта

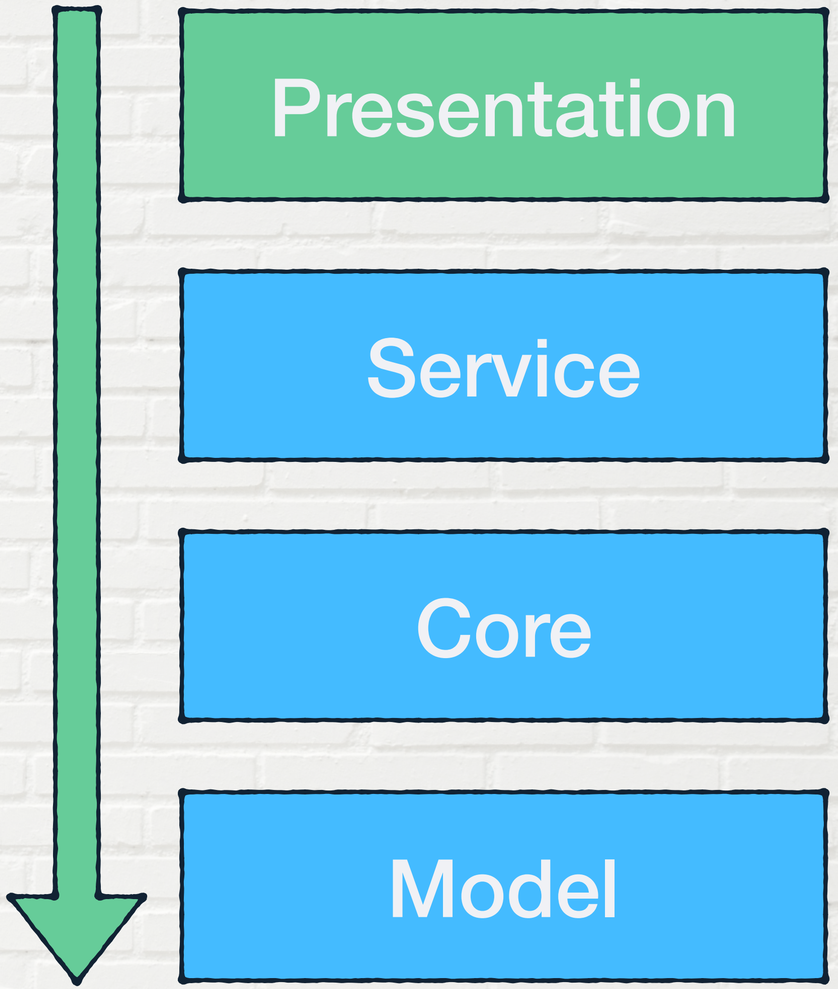


Сервис-ориентированная архитектура



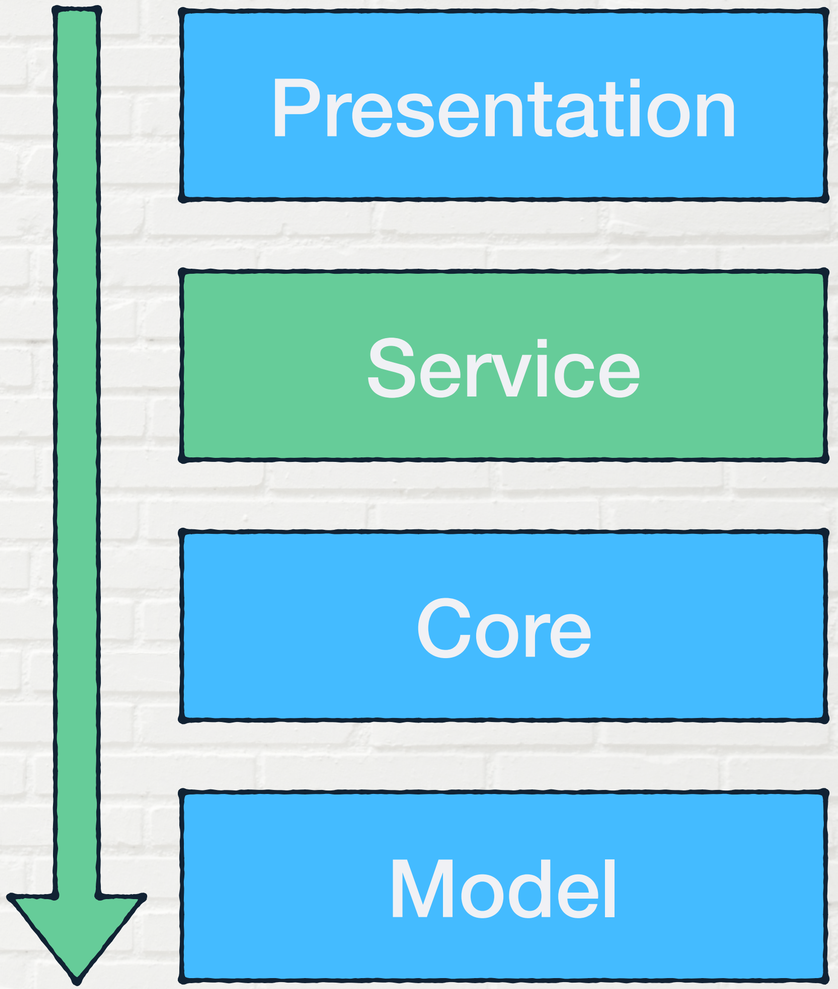
Сервис-ориентированная архитектура

```
override func viewDidLoad() {
    eventsService.obtainEvents { (events) in
        // Draw events
    }
}
```



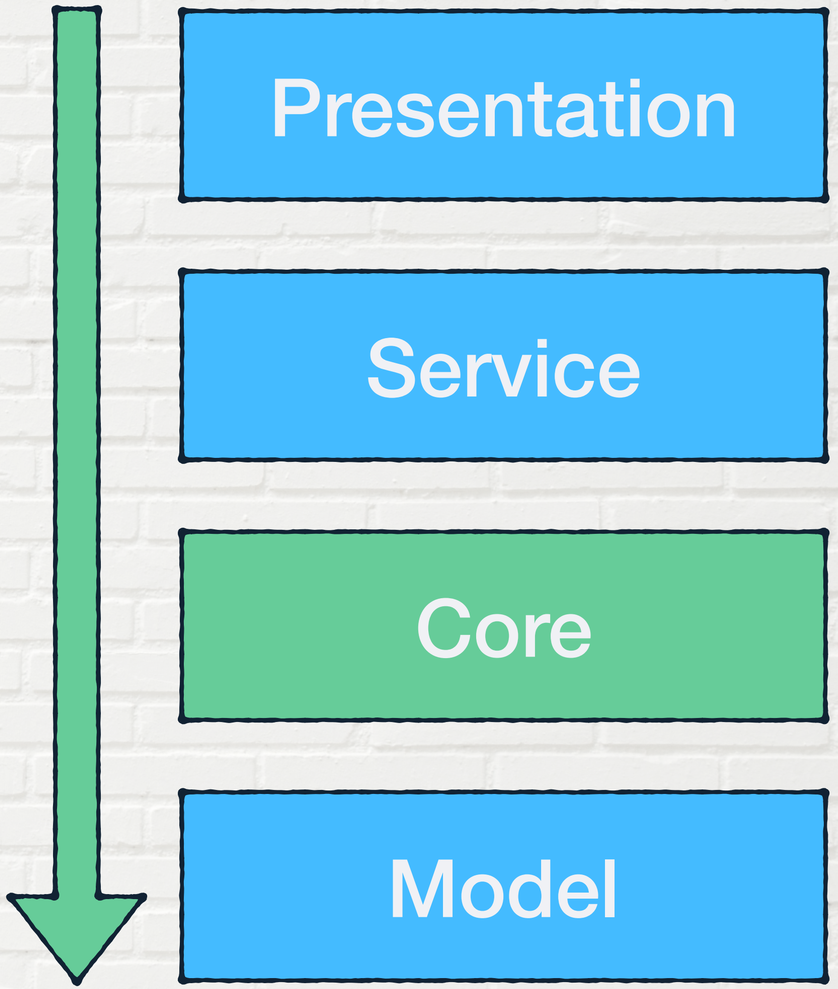
Сервис-ориентированная архитектура

```
func obtainEvents(completion: ([String]?) -> Void) {
    if let cachedEvents = cache.obtainEvents() {
        completion(cachedEvents)
    } else {
        apiManager.obtainEvents(completion: completion)
    }
}
```



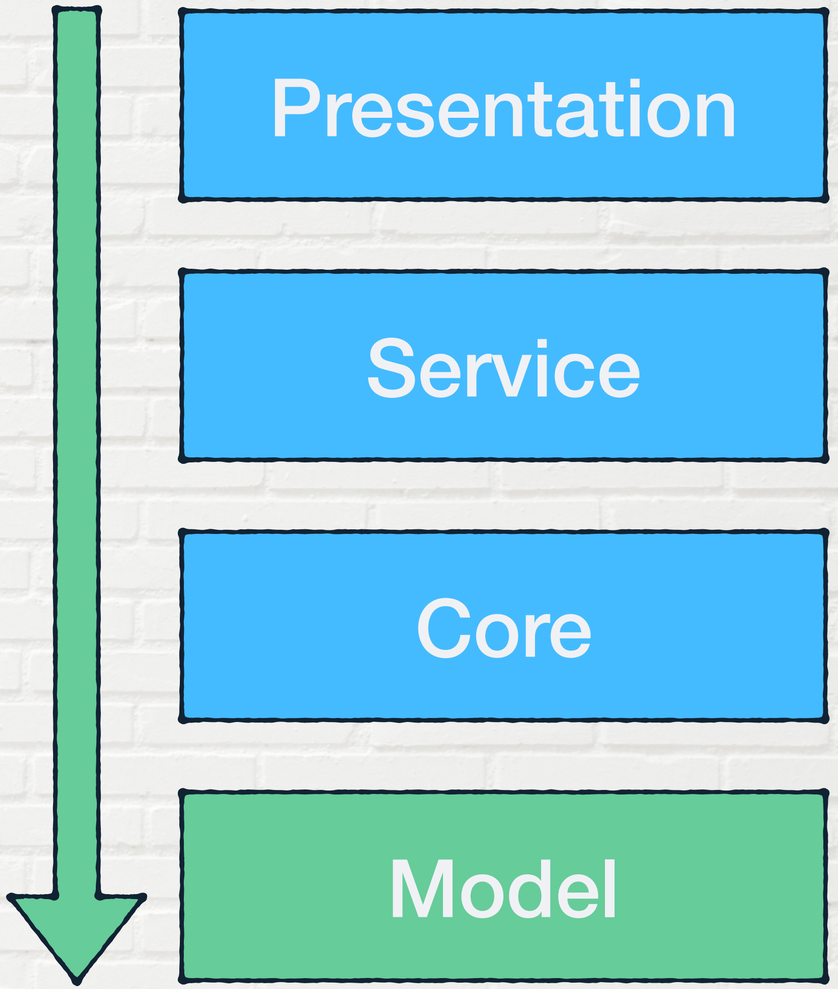
Сервис-ориентированная архитектура

```
func obtainEvents(completion: ([String]?) -> Void) {
    let eventsURL = URL(string: "https://api.com/events")!
    session.dataTask(with: eventsURL) { (data, response, error) in
        // Работа с данными
    }
}
```



Сервис-ориентированная архитектура

```
override func viewDidLoad() {
    eventsService.obtainEvents { (events) in
        // Draw events
    }
}
```



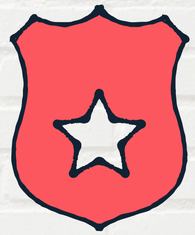


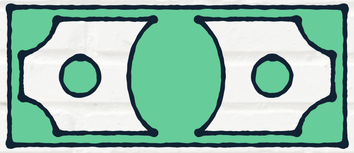














Предметная область

часть реального мира, рассматриваемая в пределах контекста вашего приложения

```
protocol CardService { /* ... */ }

protocol UserService { /* ... */ }

protocol PinCodeService { /* ... */ }
```

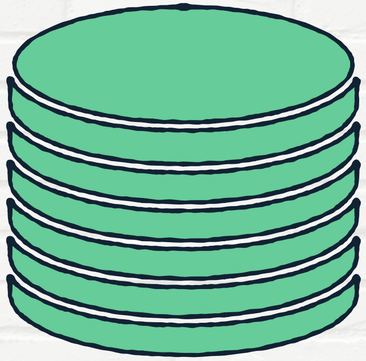






Сервисный слой

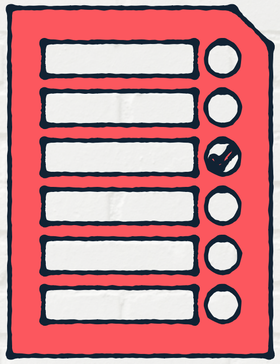
набор интерфейсов к предметной области вашего приложения



База данных



Сервер



Контакты

Инфраструктура и окружение

обеспечивают основу функционирования системы, предоставляя доступ к данным из внешнего мира

```
protocol Storage { /* ... */ }

protocol NetworkClient { /* ... */ }

protocol AddressBook { /* ... */ }
```

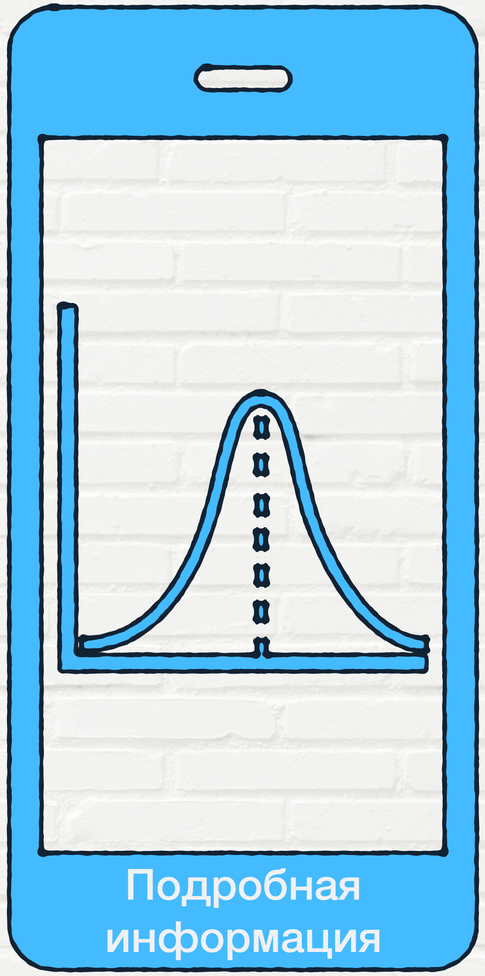


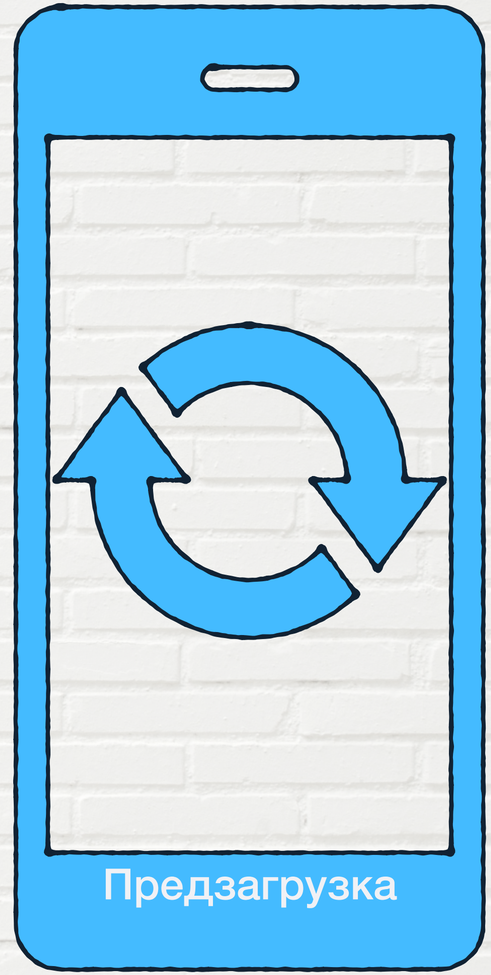
Core-слой

набор интерфейсов к инфраструктуре и окружению вашего приложения

Криптовалютная биржа



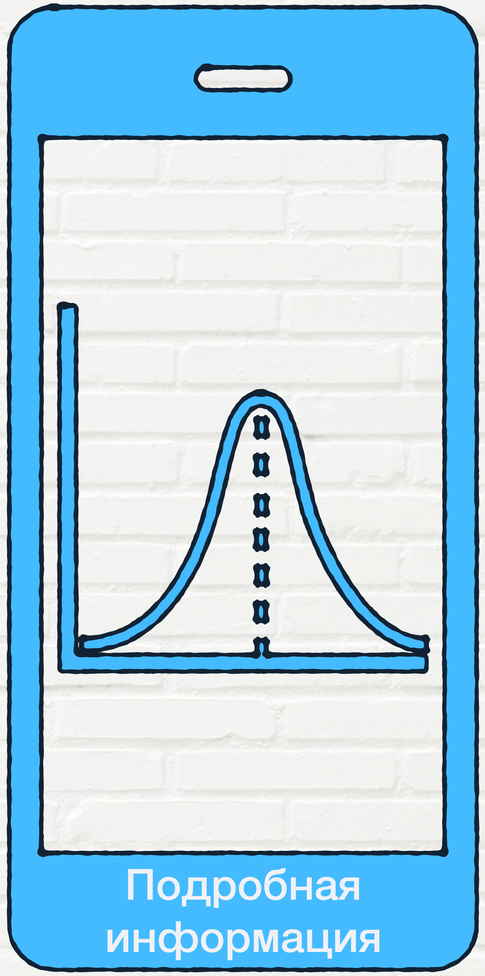




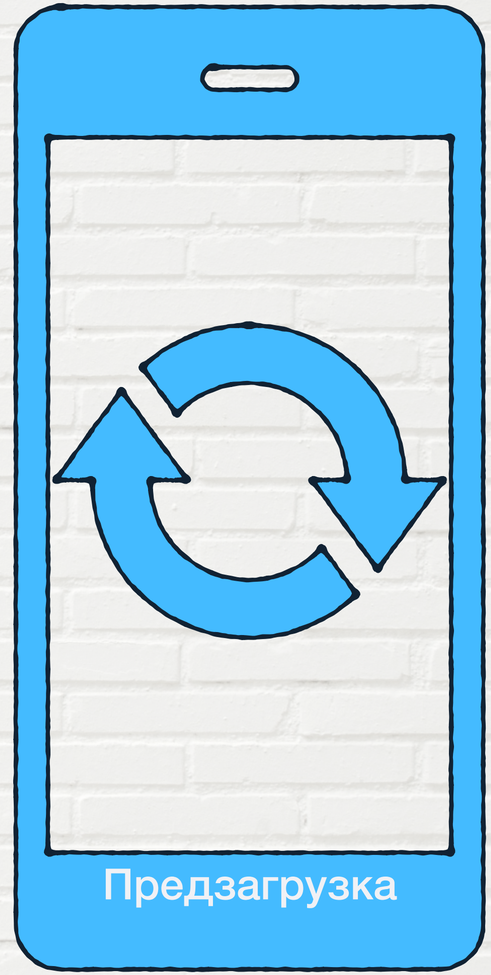
Криптовалютная биржа



Список валют



Подробная информация

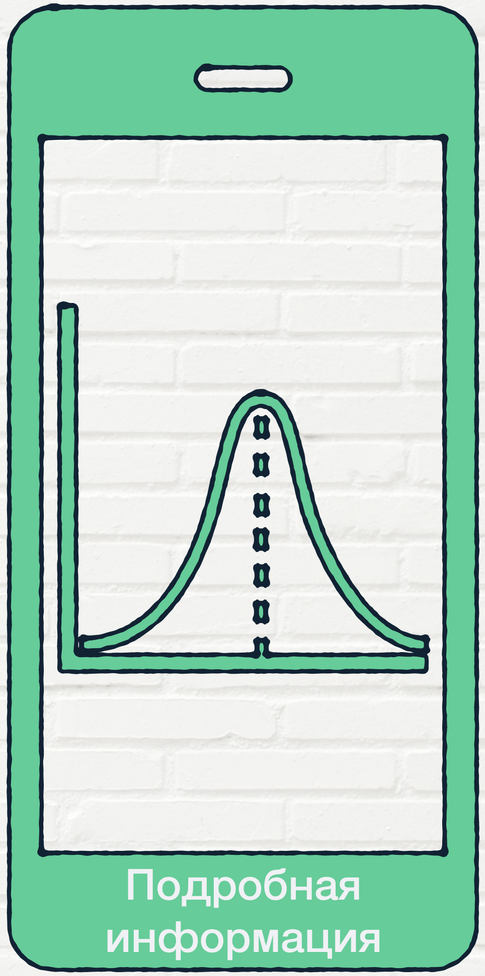


Предзагрузка

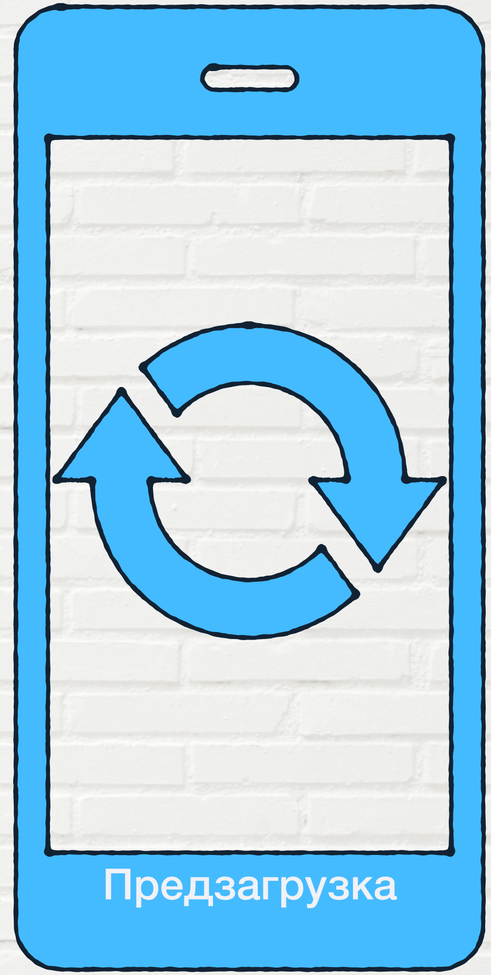
Криптовалютная биржа



Список валют



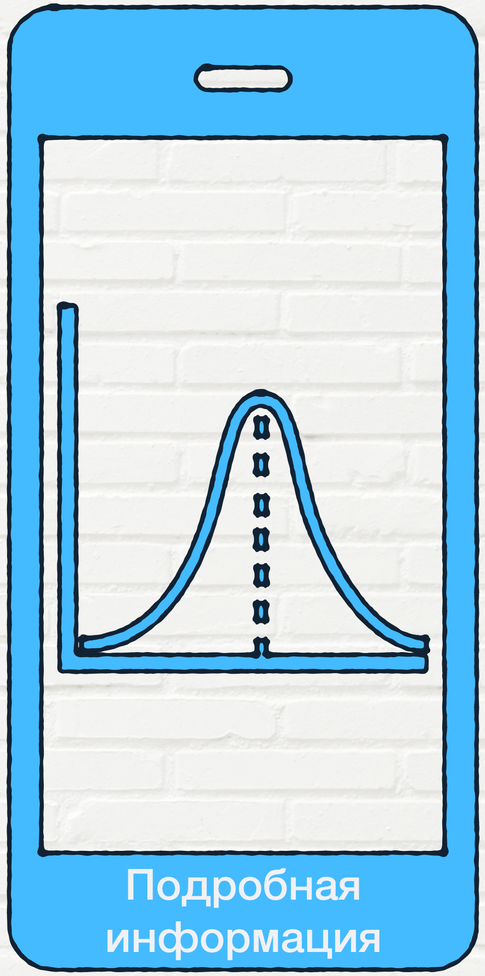
Подробная информация

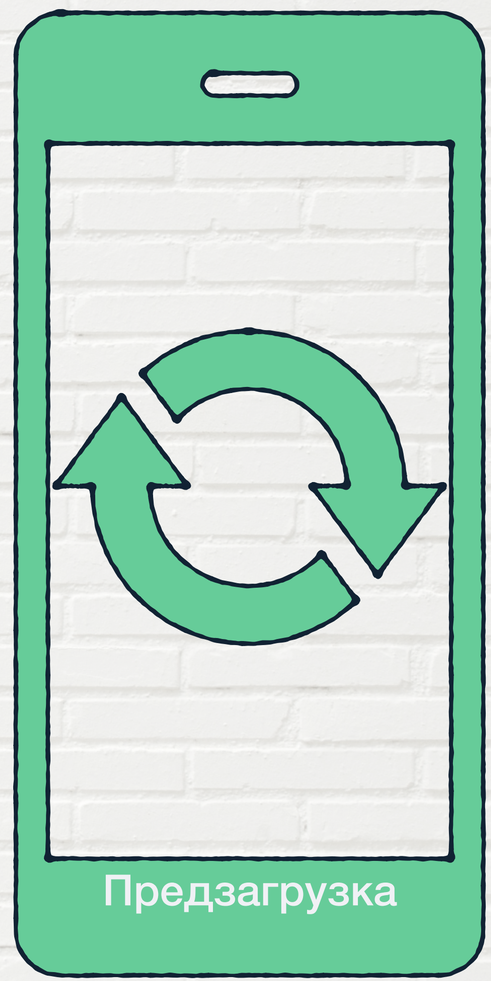


Предзагрузка

Криптовалютная биржа









Запрос

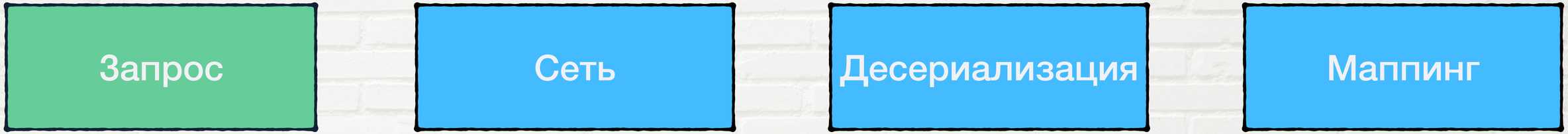
Сеть

Десериализация

Маппинг

Сервис крипотовалют

Получение списка криптовалют с API

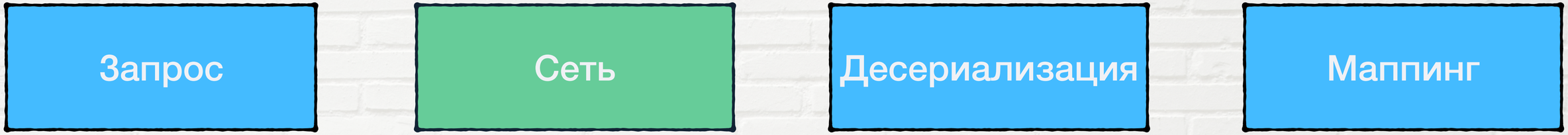



```
func obtainCryptoCurrencies(completion: ([CryptoCurrency]?, Error?) -> Void) {
    let url = URL(string: "https://api.com/cryptocurrency/all")!
    let request = URLRequest(url: url)

    /* Выполнение сетевого запроса */
```

Сервис крипотовалют

Получение списка криптовалют с API




```
func obtainCryptoCurrencies(completion: ([CryptoCurrency]?, Error?) -> Void) {
    /* Создание сетевого запроса */
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        /* Обработка данных */
    }.resume()
}
```

Сервис крипотовалют

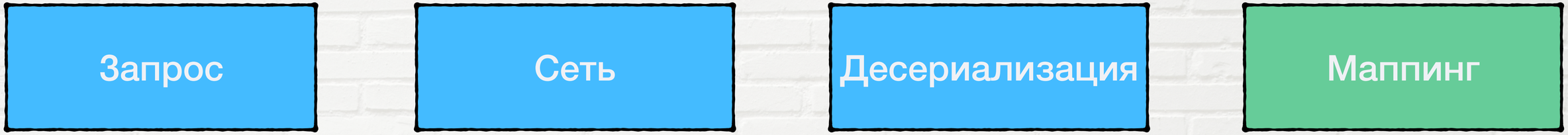
Получение списка криптовалют с API




```
func obtainCryptoCurrencies(completion: ([CryptoCurrency]?, Error?) -> Void) {
    /* Выполнение сетевого запроса */
    let serializedData = try! JSONSerialization.jsonObject(with: data,
                                                           options: [.allowFragments])
    /* Маппинг */
```

Сервис крипотовалют

Получение списка криптовалют с API




```
func obtainCryptoCurrencies(completion: ([CryptoCurrency]?, Error?) -> Void) {
    /* Сериализация данных */
    let cryptoCurrencies = serializedData.map { (cryptoCurrencyJSON) -> CryptoCurrency in
        return CryptoCurrency(JSON: cryptoCurrencyJSON)
    }

    completion(cryptoCurrencies, nil)
}
```

Сервис крипотовалют

Получение списка криптовалют с API

```swift
func obtainCryptoCurrencies(completion: @escaping ([CryptoCurrency]?, Error?) -> Void) {
    let url = URL(string: "https://api.com/cryptocurrency/all")!
    let request = URLRequest(url: url)

    URLSession.shared.dataTask(with: request) { (data, response, error) in
        let serializedData = try! JSONSerialization.jsonObject(with: data!,
                                                               options: [.allowFragments]) as! [[String: String]]

        let cryptoCurrencies = serializedData.map { (cryptoCurrencyJSON) -> CryptoCurrency in
            return CryptoCurrency(JSON: cryptoCurrencyJSON)
        }

        completion(cryptoCurrencies, nil)
    }.resume()
}
```

Сервис крипотовалют

Получение списка криптовалют с API

Сервис крипотовалют

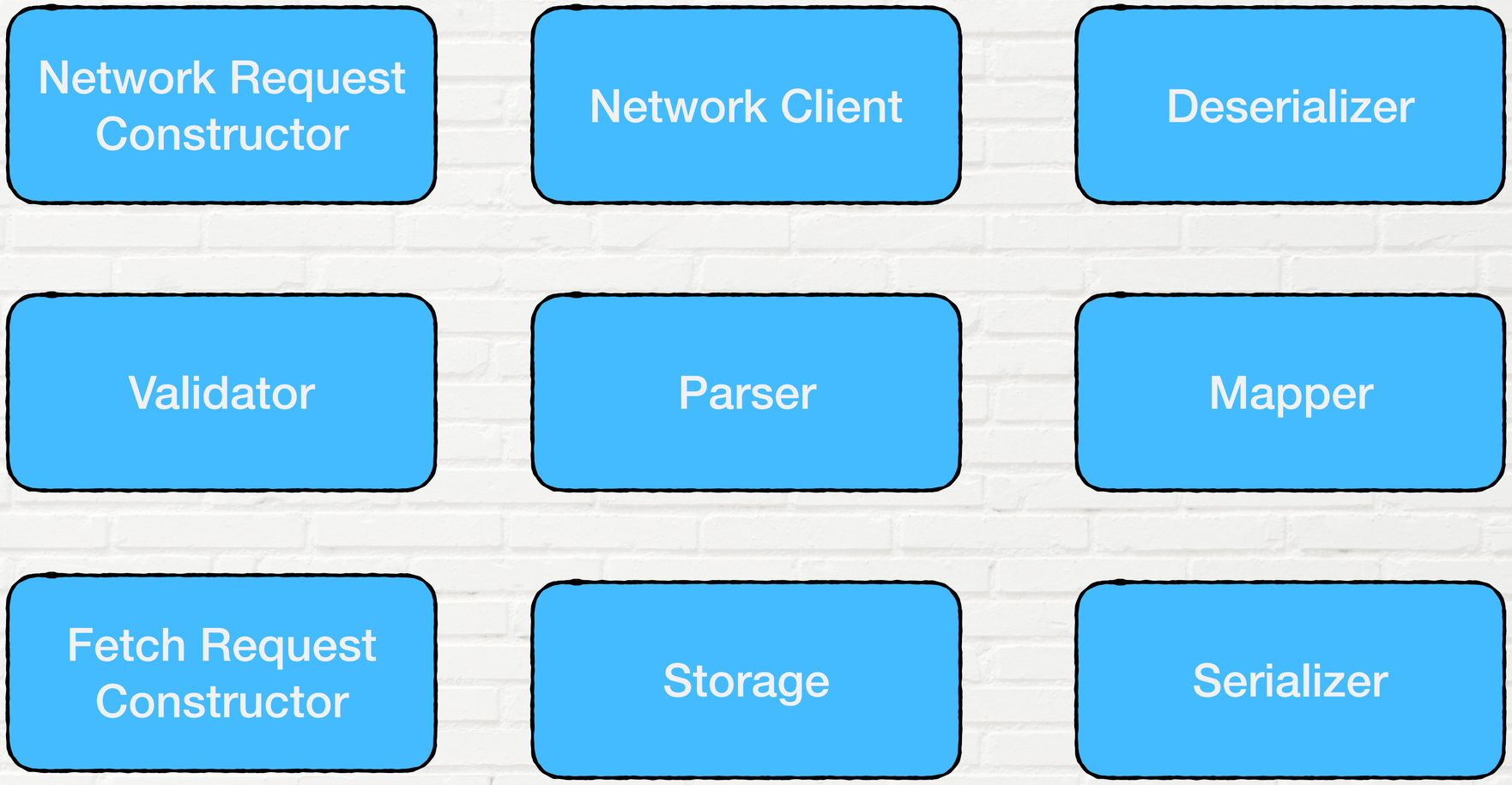
- Повторяющийся код
- Ад обратных вызовов
- Не удобная обработка ошибок



Сервис крипотовалют

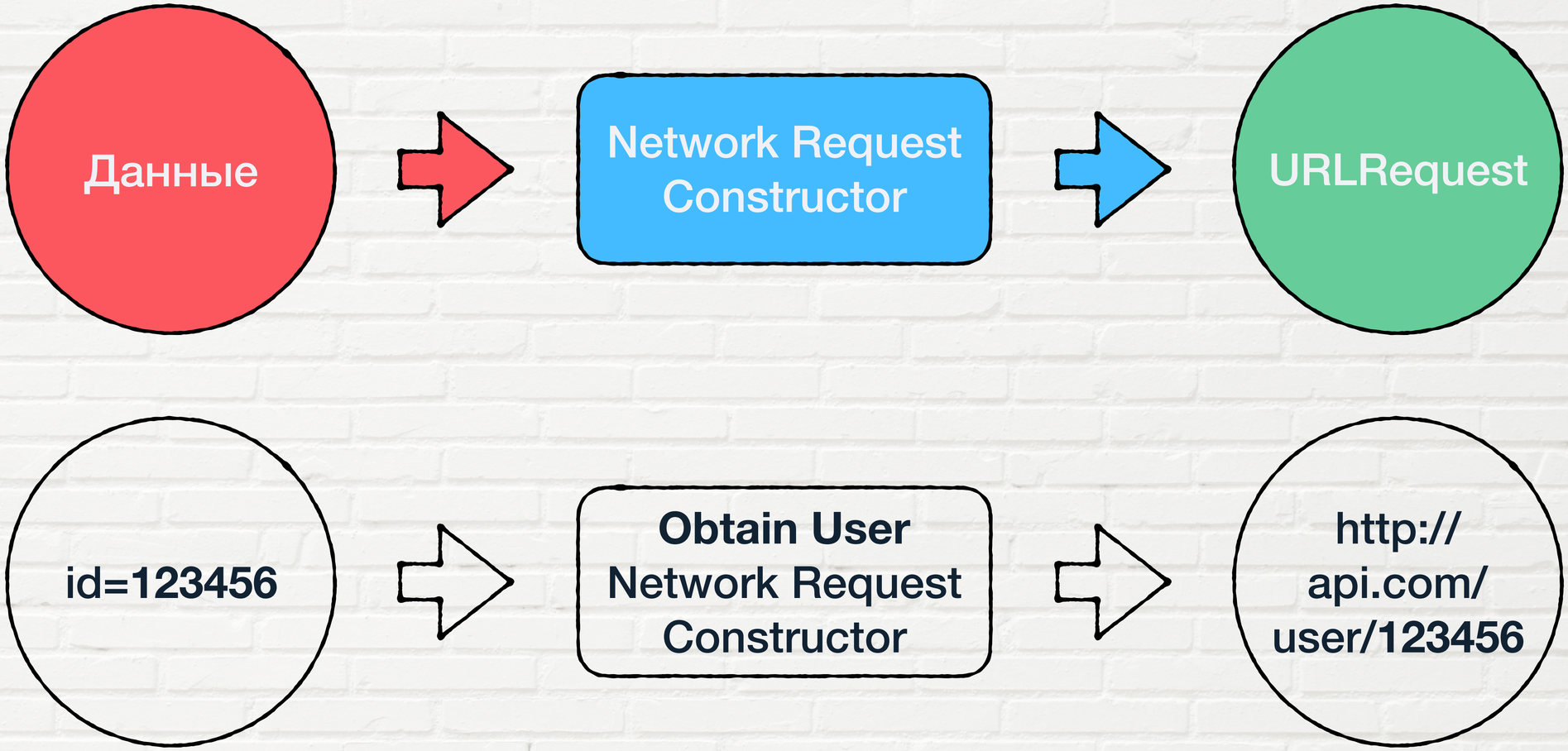
- **Повторяющийся код**
- Ад обратных вызовов
- Вынужденная обработка ошибок





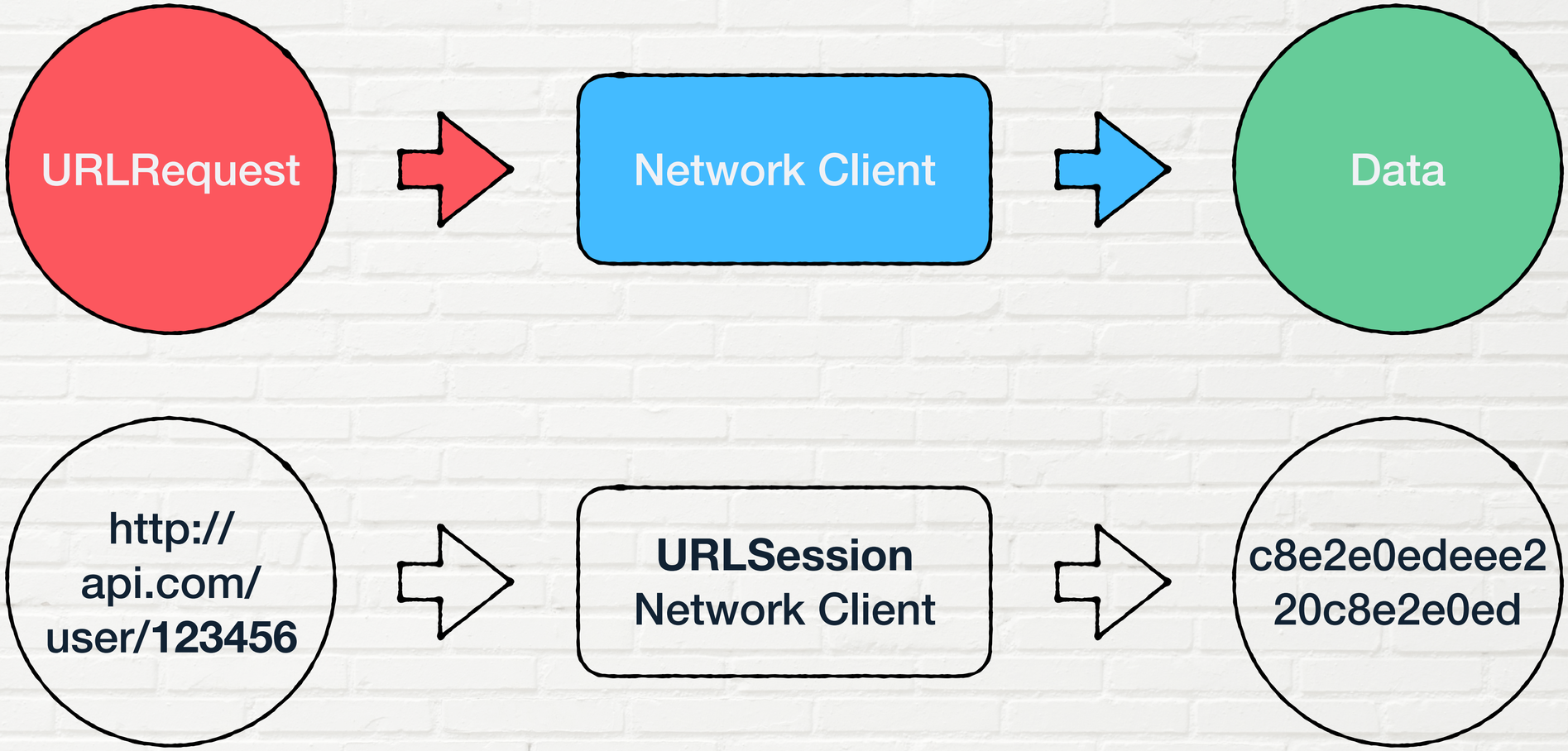
Декомпозиция core-слоя

принцип единственной ответственности в действии



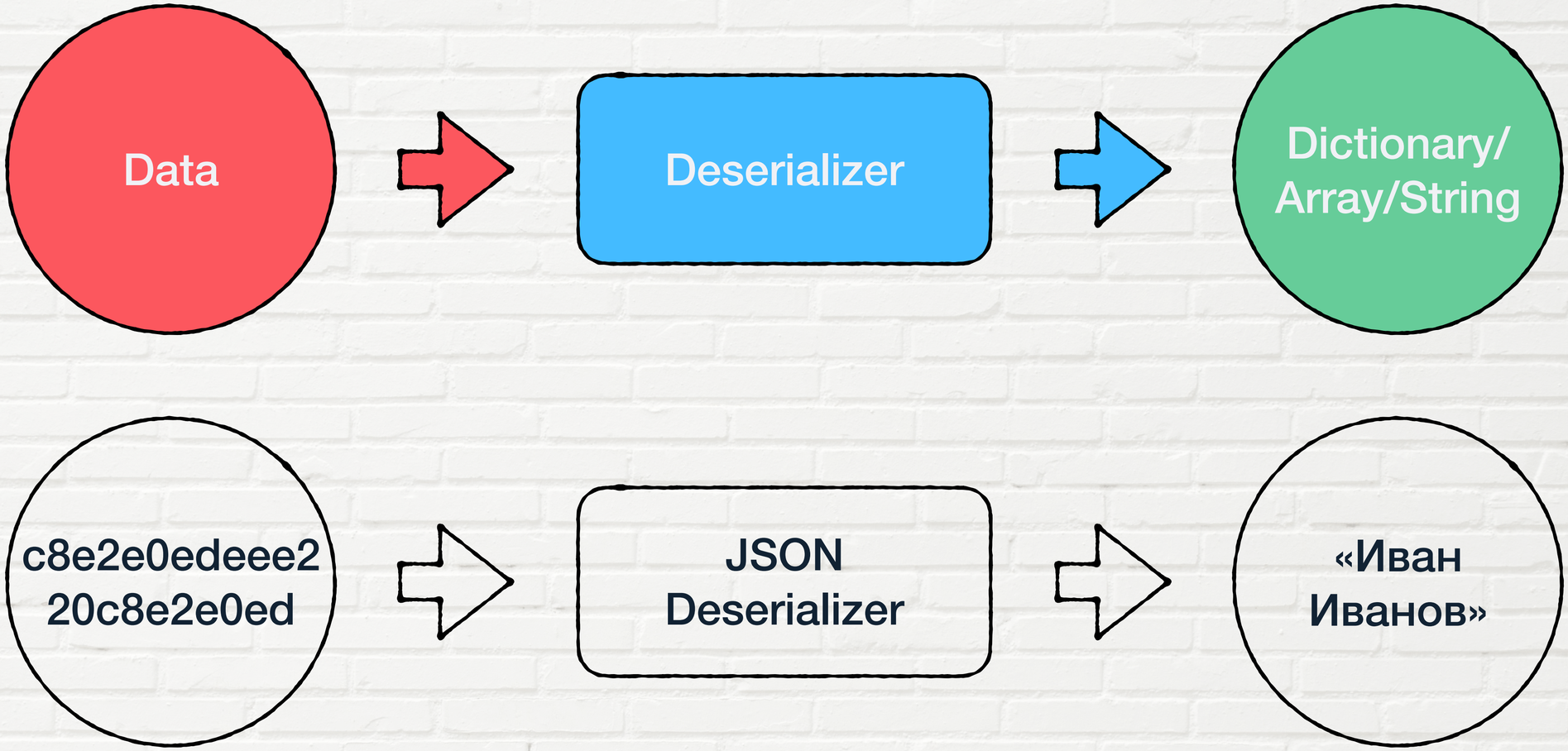
Конструктор сетевых запросов

создает сетевой запрос на основе входных данных



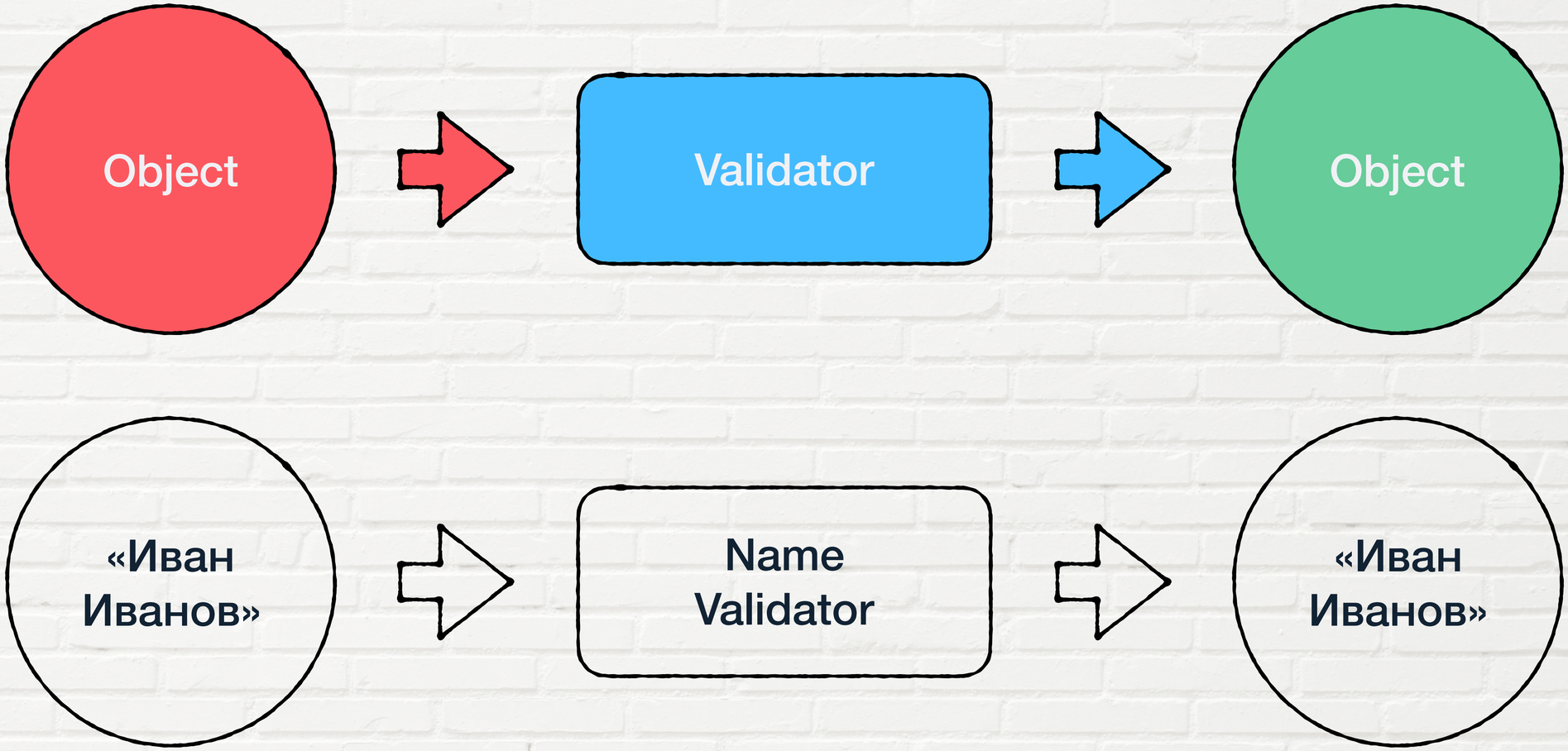
Сетевой клиент

выполняет сетевой запрос



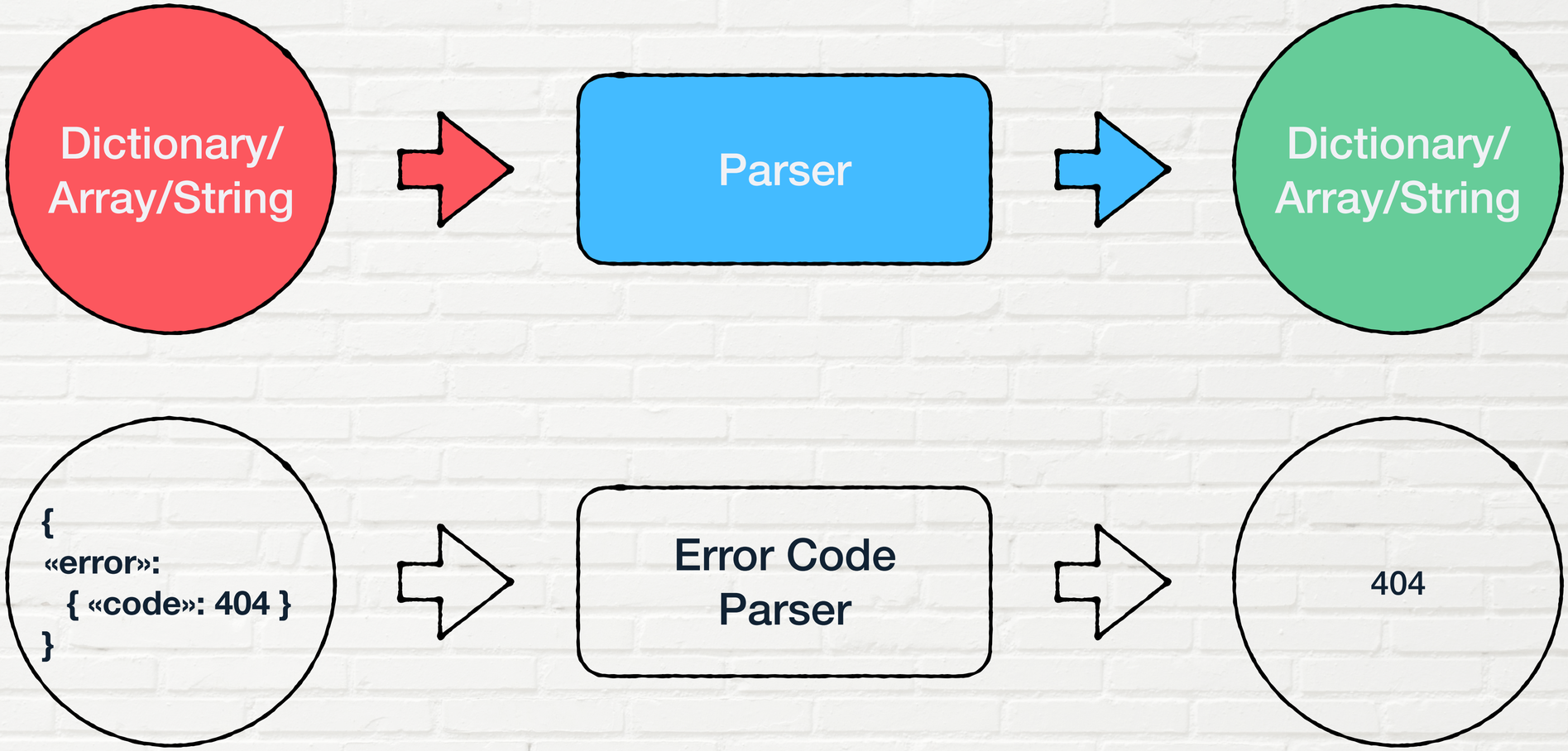
Десериализатор

превращает бинарные данные в читаемые



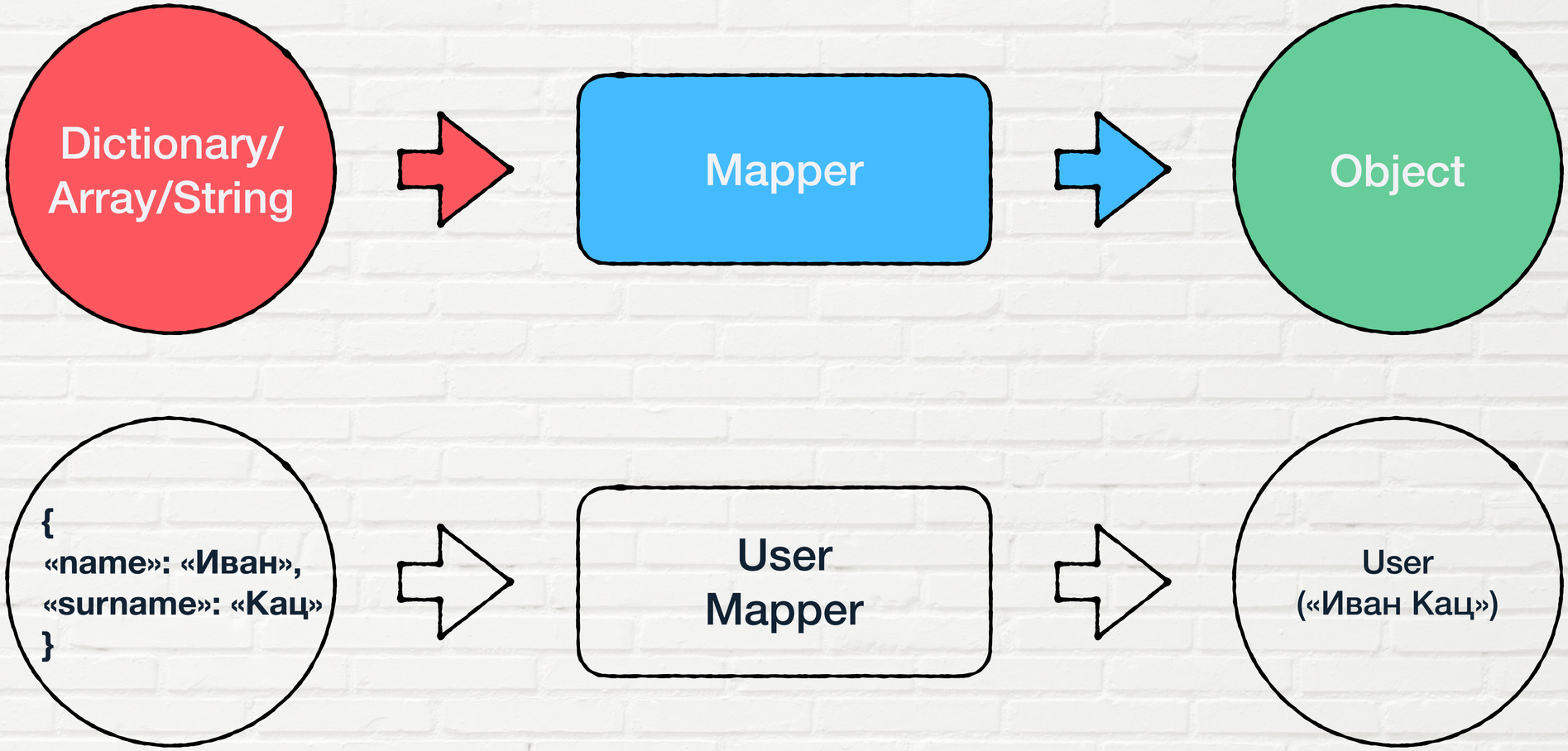
Валидатор

проверяет соответствие данных ожидаемой структуре



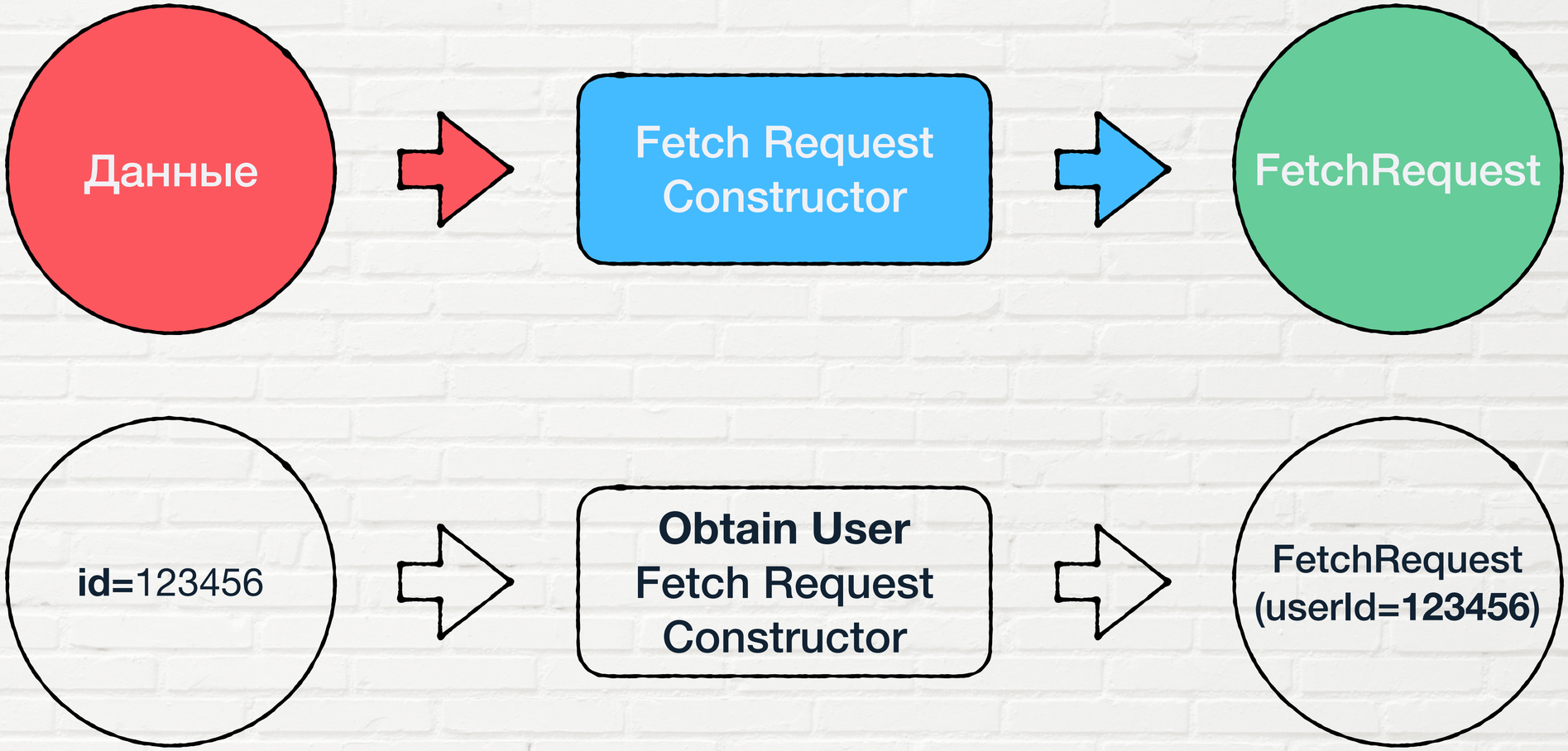
Парсер

извлекает данные из сложной структуры



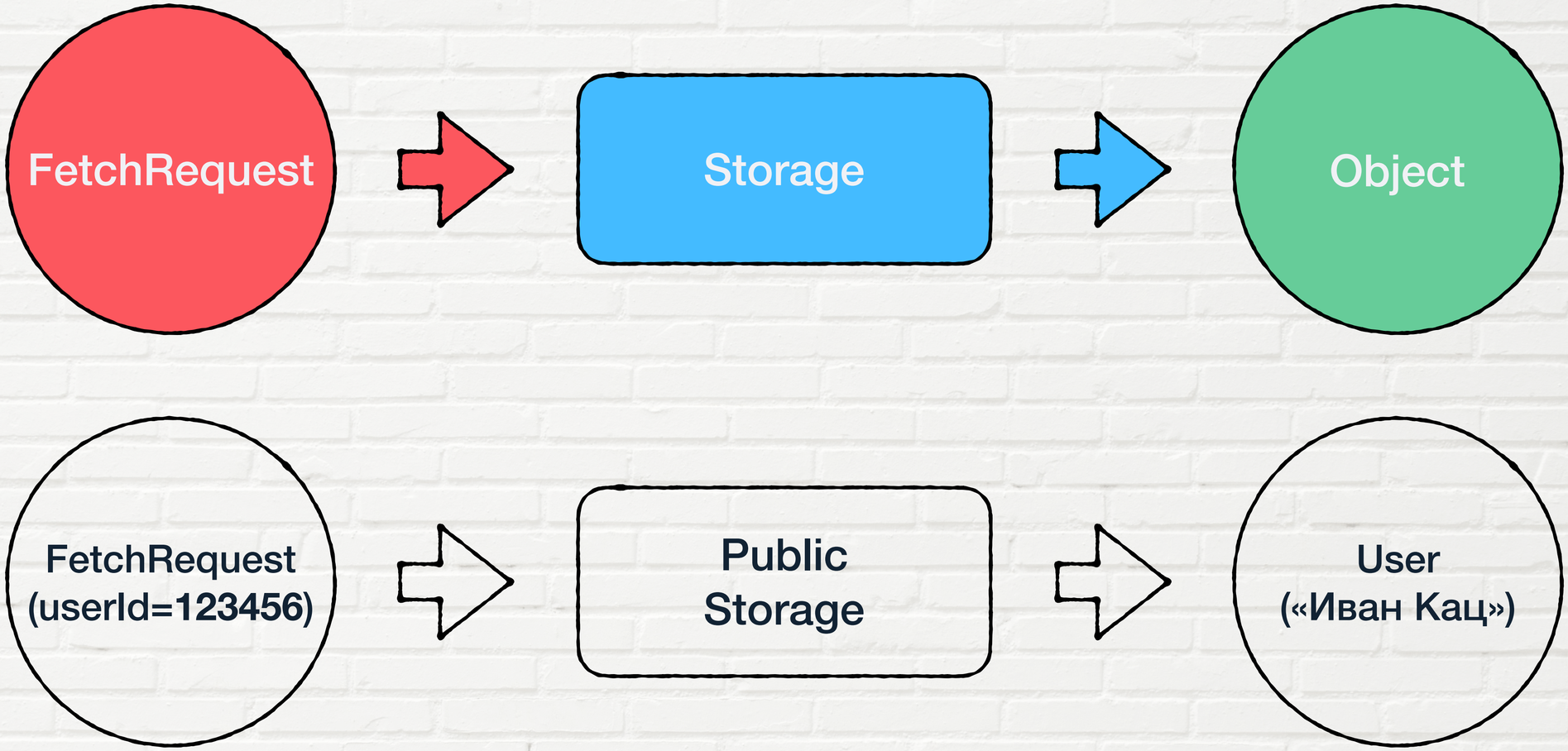
Маппер

превращает читаемые данные в модельные объекты



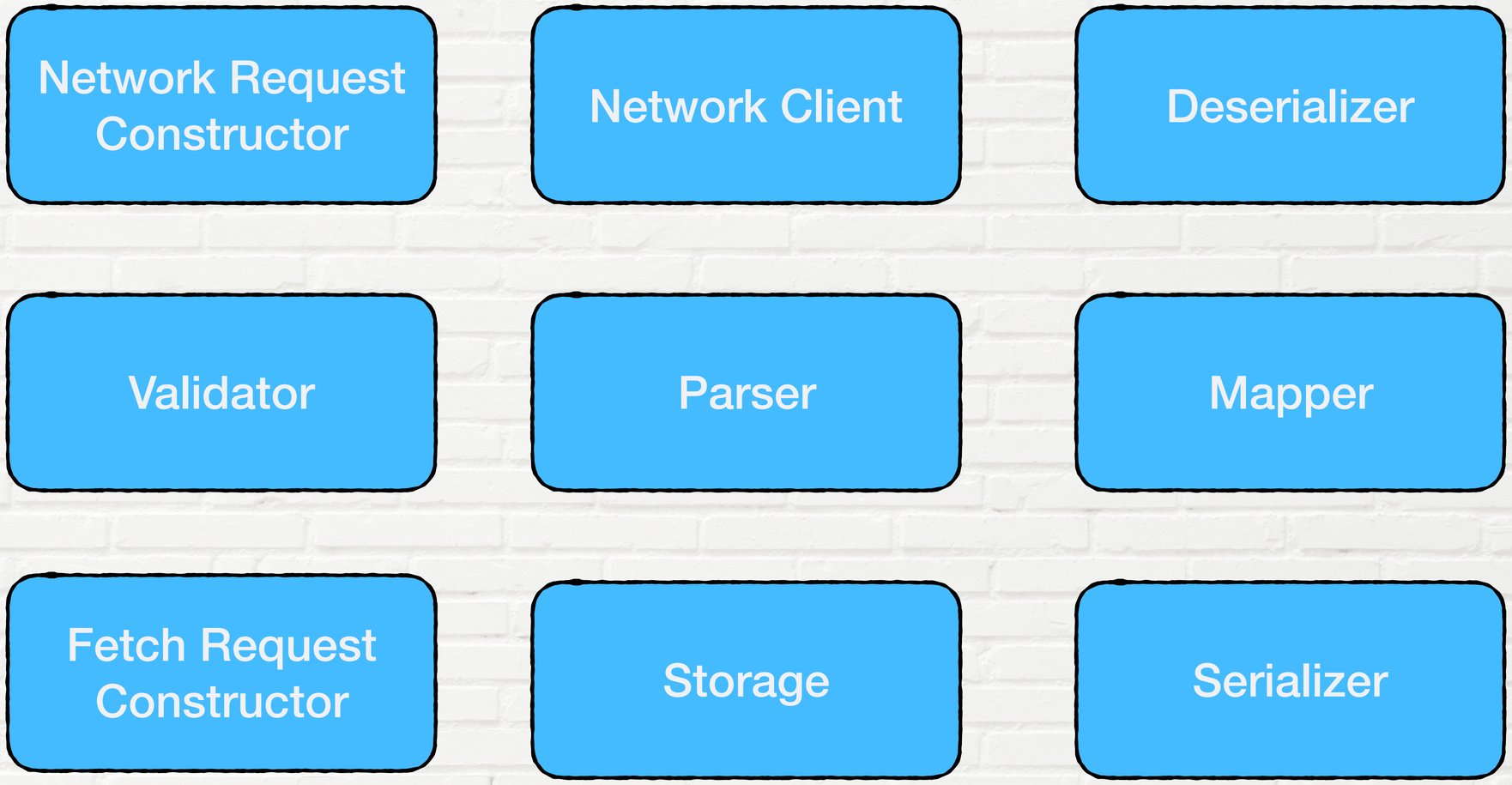
Конструктор запросов к БД

создает запрос к БД на основе входных данных



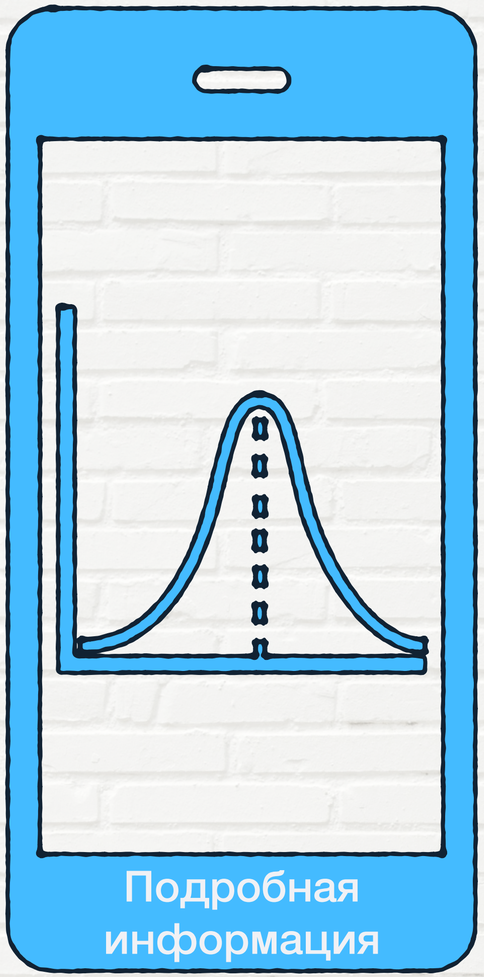
Хранилище

превращает читаемые данные в модельные объекты



Декомпозиция core-слоя

принцип единственной ответственности в действии



Запрос

Сеть

Десериализация

Маппинг

Сохранение

Сервис деталей криптовалюты

получение подробных данных о криптовалюте по её индентификатору



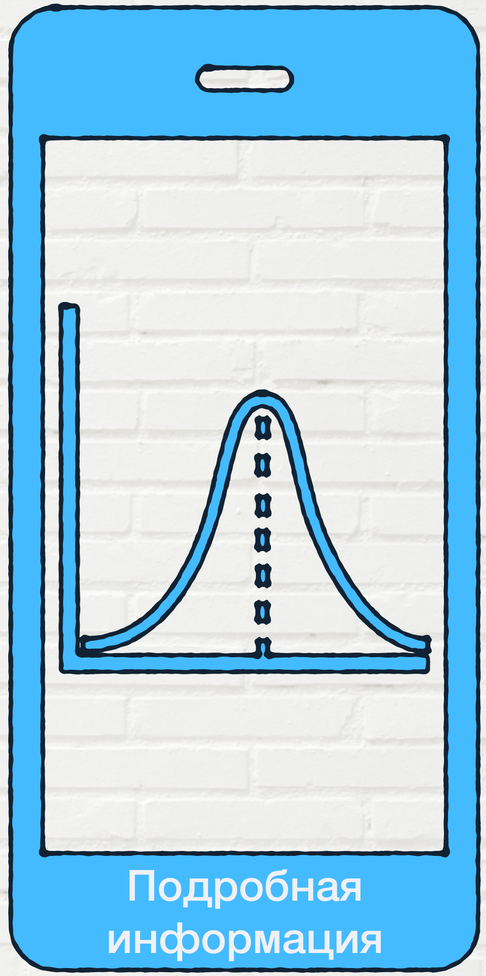

```
func obtainDetailsForCryptoCurrencyWithId(_ cryptoCurrencyId: String,
                                          completion: (CryptoCurrency) -> Void) {
    let request = requestConstructor.constructRequest(with: cryptoCurrencyId)
    networkClient.performRequest(request) { (data) in
        let json = deserializer.deserializeData(data)
        let cryptoCurrency = mapper.mapJSON(json)
        storage.saveObject(cryptoCurrency)
        completion(cryptoCurrency)
    }
}
```

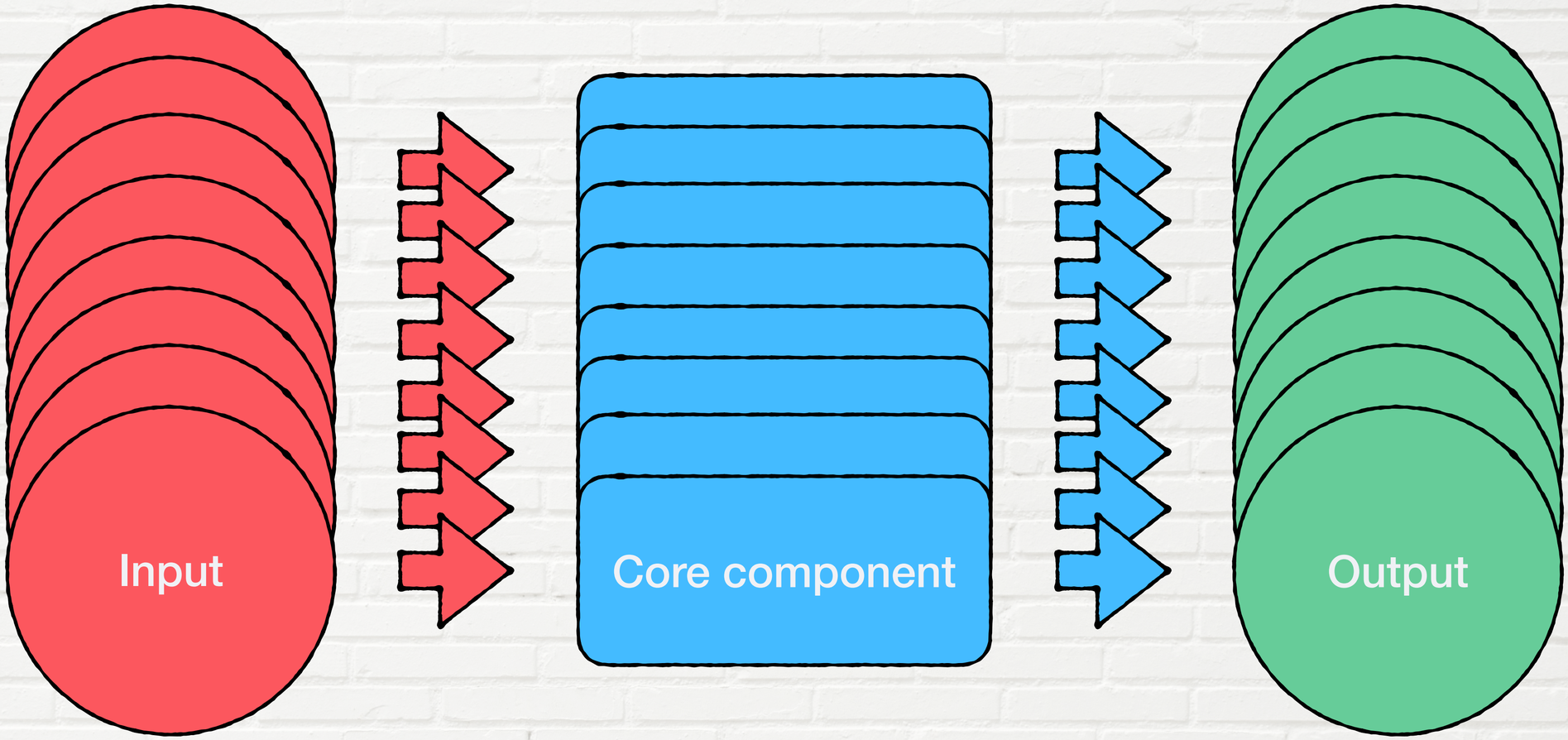
Сервис деталей криптовалюты

получение подробных данных о криптовалюте по её индентификатору

Сервис деталей криптовалюты

- ~~Повторяющийся код~~
- Ад обратных вызовов
- Вынужденная обработка ошибок





Реализация метода сервиса

цепочка преобразований данных

```
func start()
```

Begins the execution of the operation.

```
func main()
```

Performs the receiver’s non-concurrent task.

```
var completionBlock: (() -> Void)?
```

The block to execute after the operation’s main task is completed.

NSOperation

выполнение операции

```
func cancel()
```

Advises the operation object that it should stop executing its task.

NSOperation

отмена операции

```
func addDependency(Operation)
```

Makes the receiver dependent on the completion of the specified operation.

```
func removeDependency(Operation)
```

Removes the receiver’s dependence on the specified operation.

```
var dependencies: [Operation]
```

An array of the operation objects that must finish executing before the current object can begin executing.

NSOperation

зависимость по управлению

```
func addOperation(Operation)
```

Adds the specified operation object to the receiver.

```
func addOperations([Operation], waitUntilFinished: Bool)
```

Adds the specified array of operations to the queue.

```
func addOperation(() -> Void)
```

Wraps the specified block in an operation object and adds it to the receiver.

NSOperationQueue

добавление операций в очередь

```
var qualityOfService: QualityOfService
```

The default service level to apply to operations executed using the queue.

```
var maxConcurrentOperationCount: Int
```

The maximum number of queued operations that can execute at the same time.

```
var isSuspended: Bool
```

A Boolean value indicating whether the queue is actively scheduling operations for execution.

NSOperationQueue

конфигурация очереди операций

```
func cancelAllOperations()
```

Cancels all queued and executing operations.

```
func waitUntilAllOperationsAreFinished()
```

Blocks the current thread until all of the receiver’s queued and executing operations finish executing.

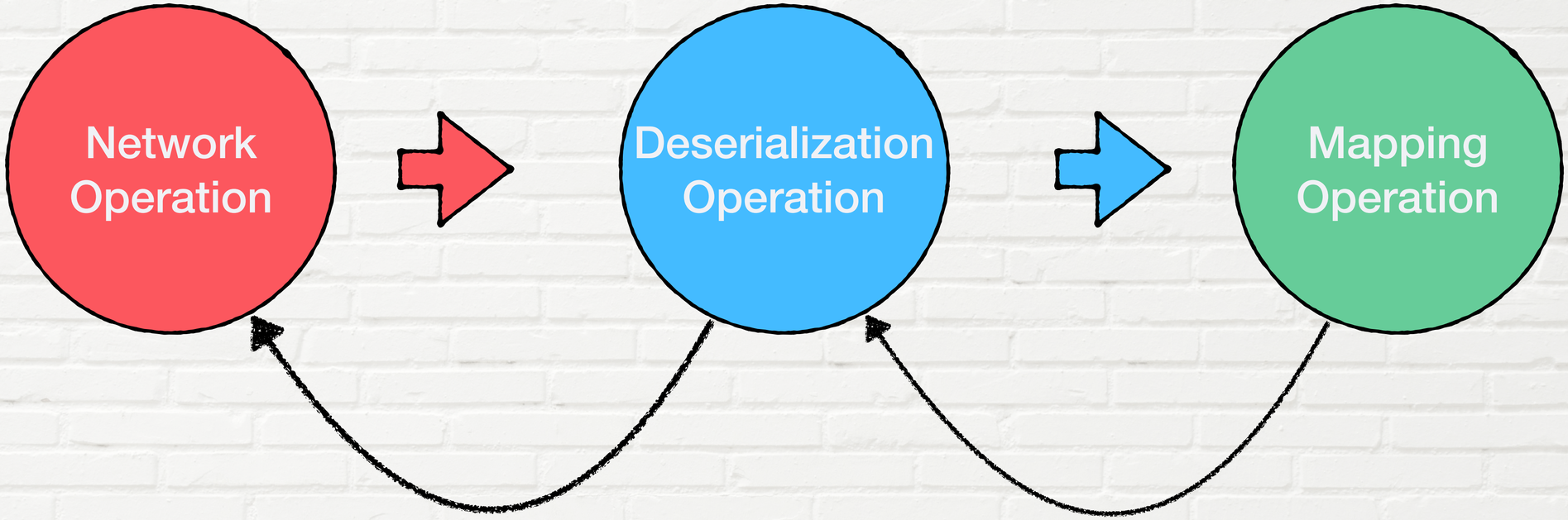
NSOperationQueue

завершение очереди операций



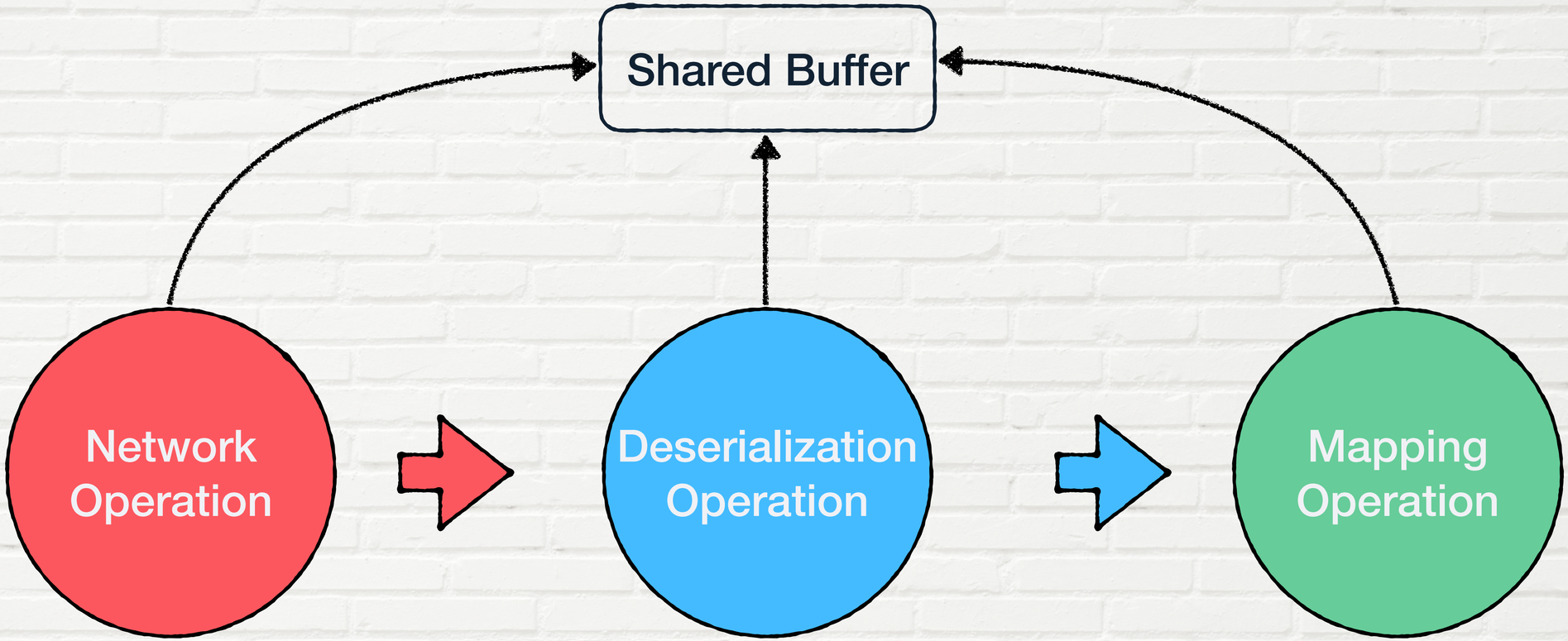
Операции

отсутствует способ передачи данных между последовательными операциями



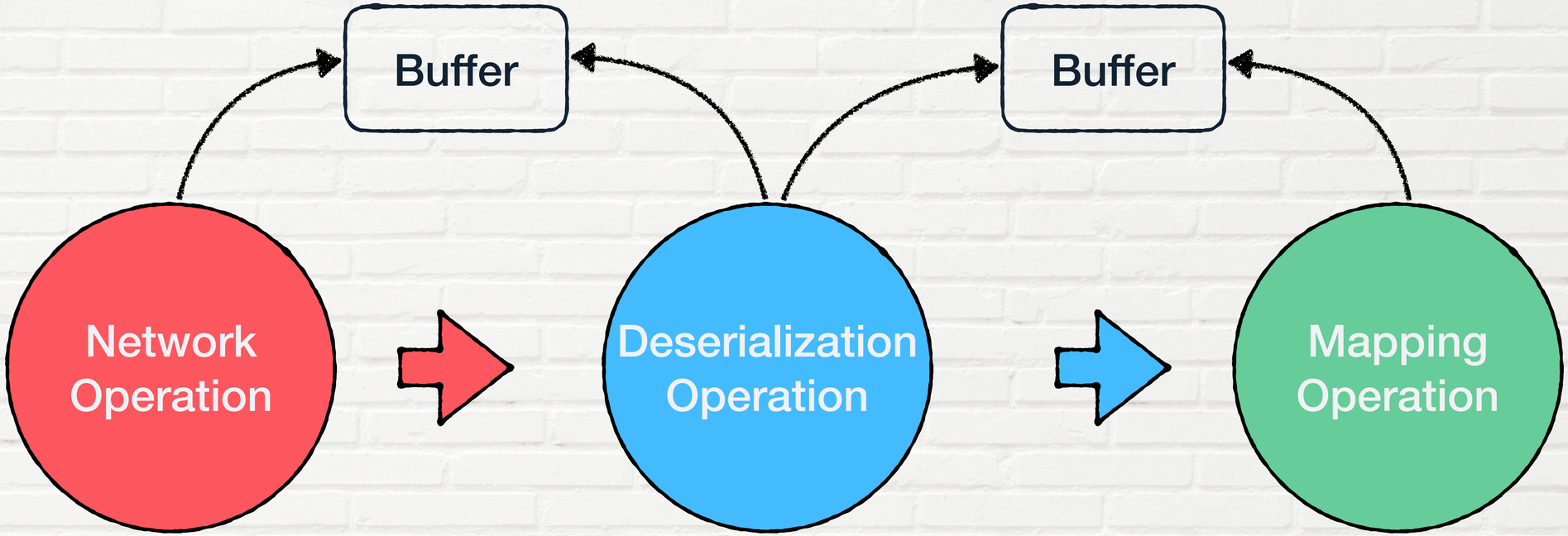
Передача данных между операциями

явная ссылка на предыдущую



Передача данных между операциями

общее место хранения данных



Передача данных между операциями

промежуточные буферы

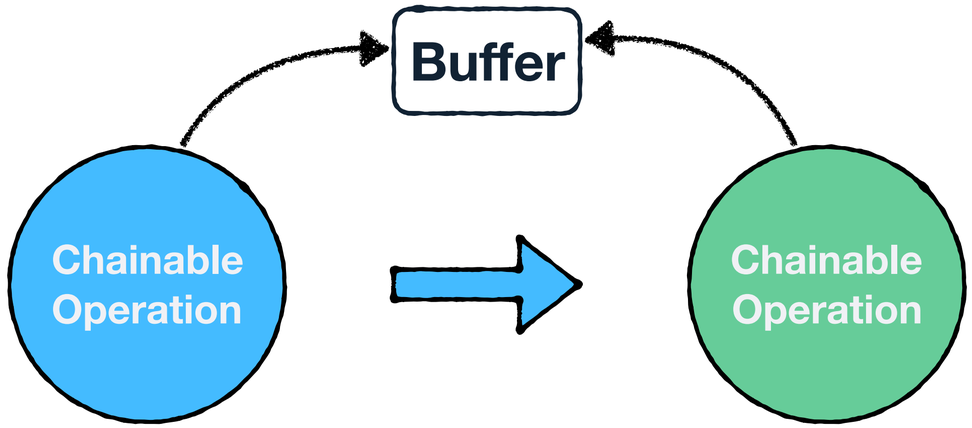
Связываемые операции

```
protocol ChainableOperationInput {
    func receiveInputData() -> Any
}

protocol ChainableOperationOutput {
    func sendOutputData(data: Any)
}

protocol ChainableOperation {
    var input: ChainableOperationInput? { get set }
    var output: ChainableOperationInput? { get set }
}

typealias ChainableOperationBuffer = ChainableOperationInput & ChainableOperationOutput
```



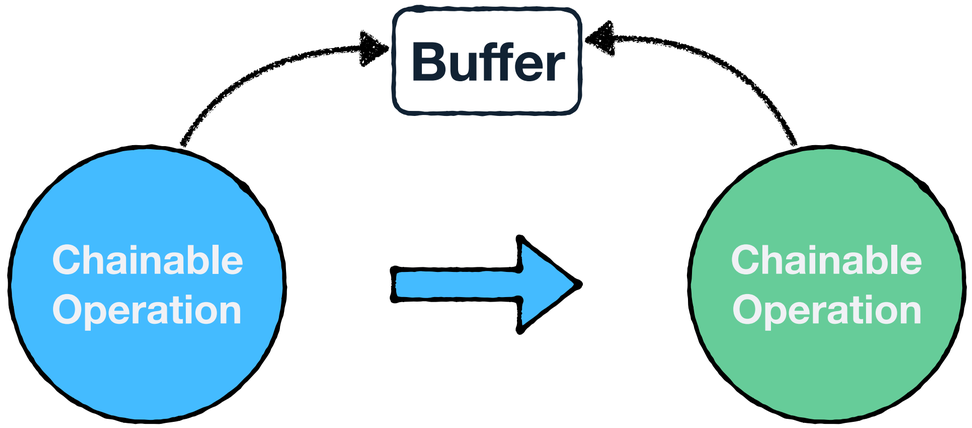
Связываемые операции

```
protocol ChainableOperationInput {
    func receiveInputData() -> Any
}

protocol ChainableOperationOutput {
    func sendOutputData(data: Any)
}

protocol ChainableOperation {
    var input: ChainableOperationInput? { get set }
    var output: ChainableOperationInput? { get set }
}

typealias ChainableOperationBuffer = ChainableOperationInput & ChainableOperationOutput
```



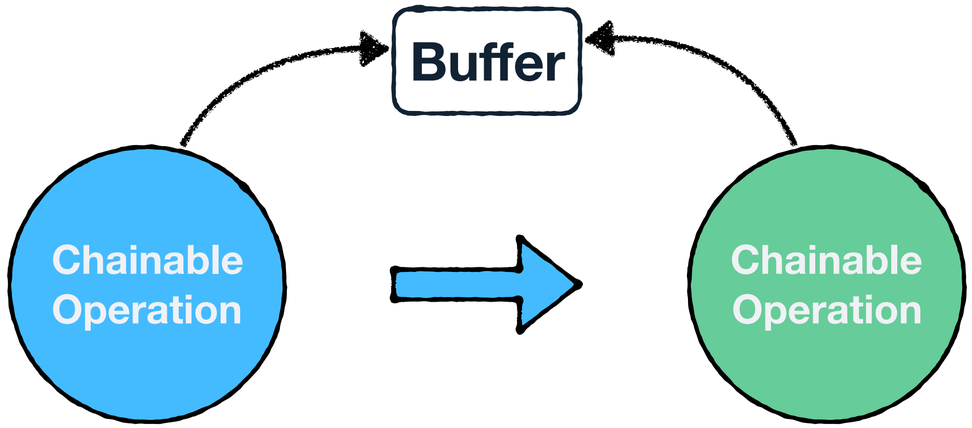
Связываемые операции

```
protocol ChainableOperationInput {
    func receiveInputData() -> Any
}

protocol ChainableOperationOutput {
    func sendOutputData(data: Any)
}

protocol ChainableOperation {
    var input: ChainableOperationInput? { get set }
    var output: ChainableOperationInput? { get set }
}

typealias ChainableOperationBuffer = ChainableOperationInput & ChainableOperationOutput
```



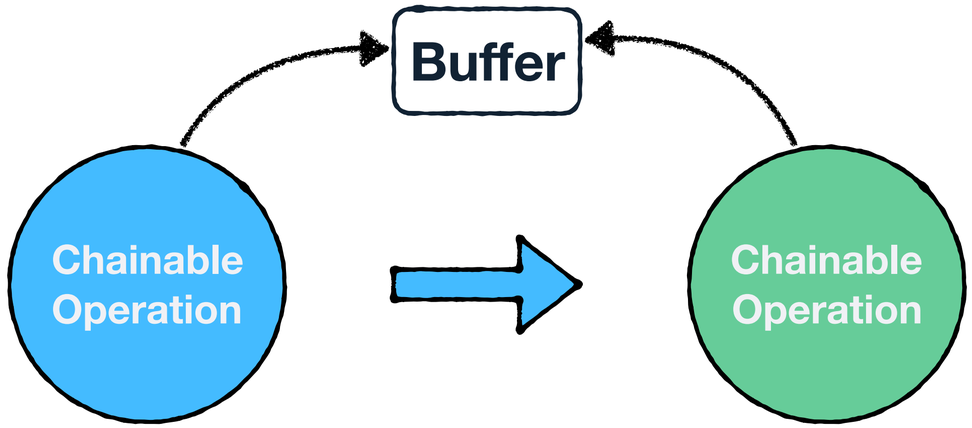
Связываемые операции

```
protocol ChainableOperationInput {
    func receiveInputData() -> Any
}

protocol ChainableOperationOutput {
    func sendOutputData(data: Any)
}

protocol ChainableOperation {
    var input: ChainableOperationInput? { get set }
    var output: ChainableOperationOutput? { get set }
}

typealias ChainableOperationBuffer = ChainableOperationInput & ChainableOperationOutput
```



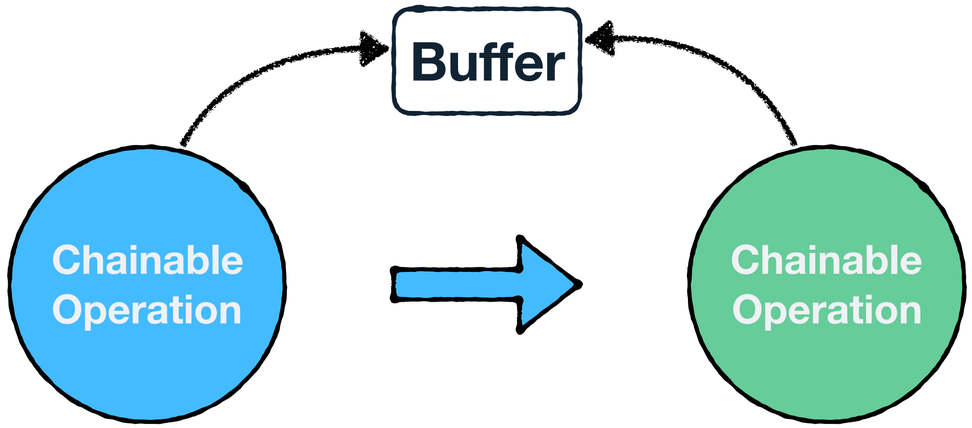
Связываемые операции

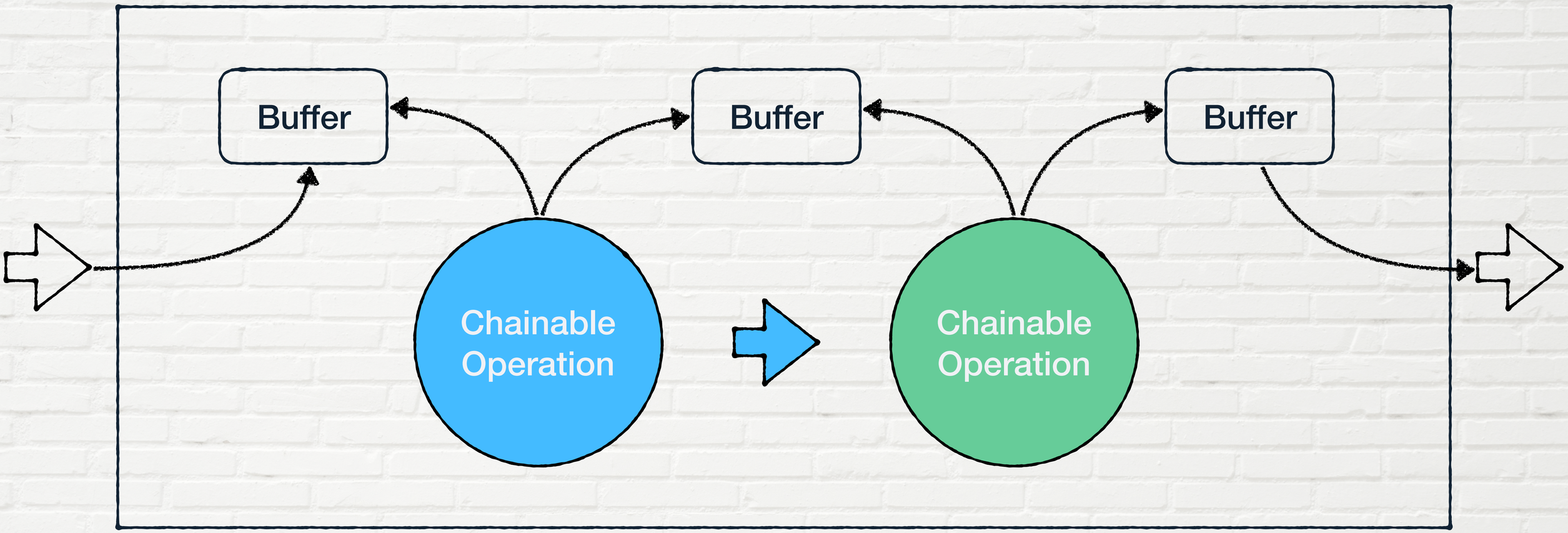
```
protocol ChainableOperationInput {
    func receiveInputData() -> Any
}

protocol ChainableOperationOutput {
    func sendOutputData(data: Any)
}

protocol ChainableOperation {
    var input: ChainableOperationInput? { get set }
    var output: ChainableOperationInput? { get set }
}

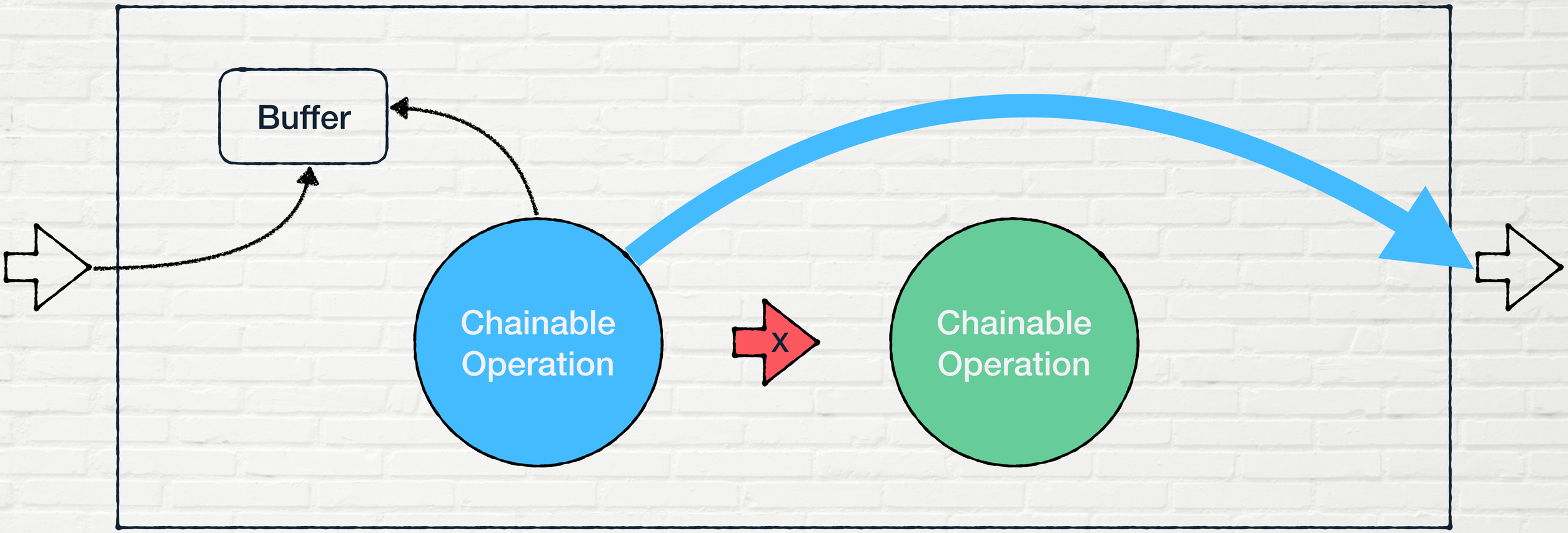
typealias ChainableOperationBuffer = ChainableOperationInput & ChainableOperationOutput
```





Составная операция

конфигурация и управления цепочкой связываемых операций

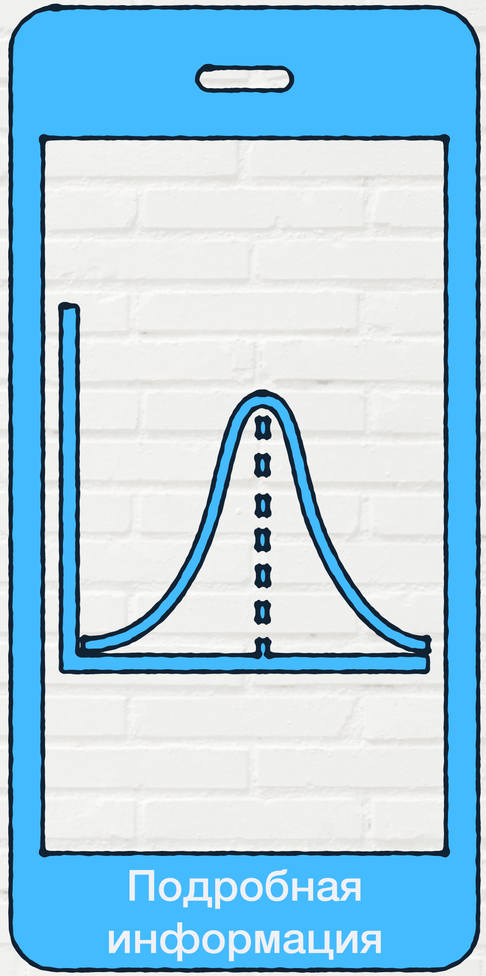


Составная операция

централизованная обработка ошибок и отмена операций

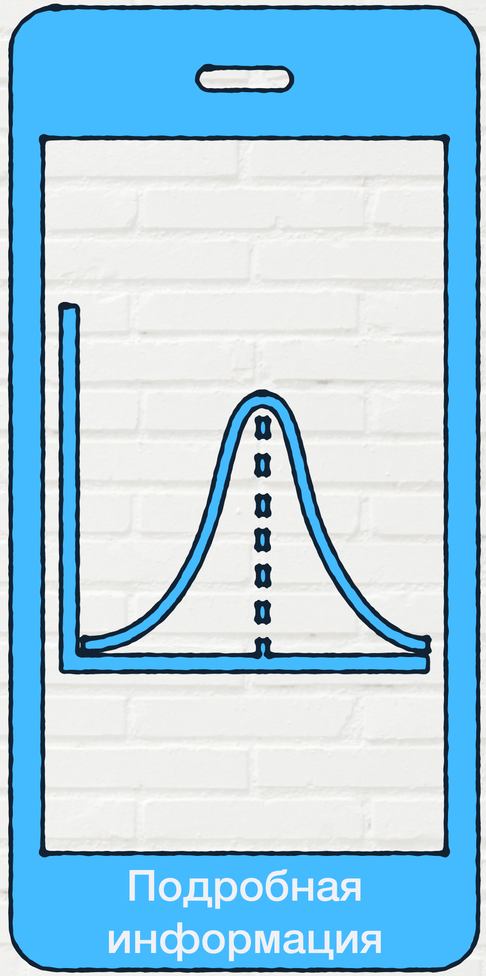
Сервис деталей криптовалюты

- ~~Повторяющийся код~~
- Ад обратных вызовов
- Вынужденная обработка ошибок



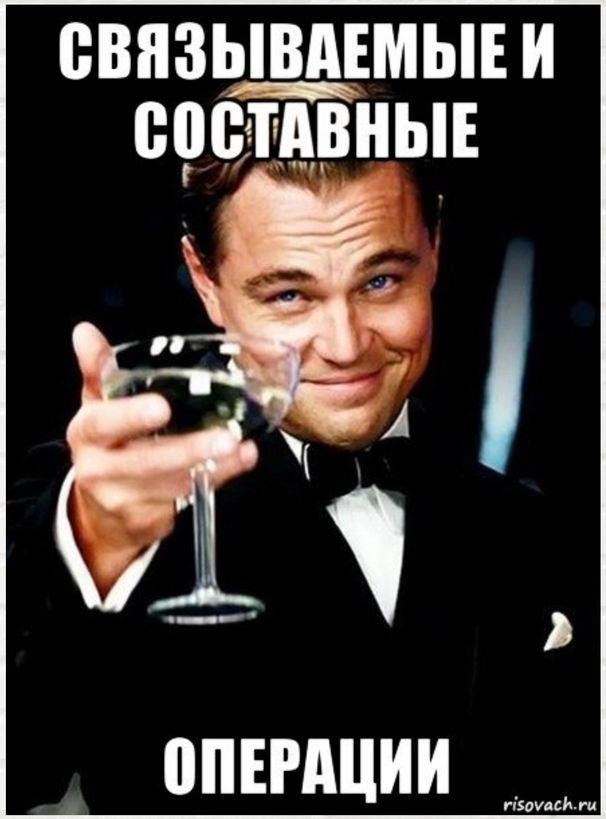
Сервис деталей криптовалюты

- ~~Повторяющийся код~~
- ~~Ад обратных вызовов~~
- Вынужденная обработка ошибок



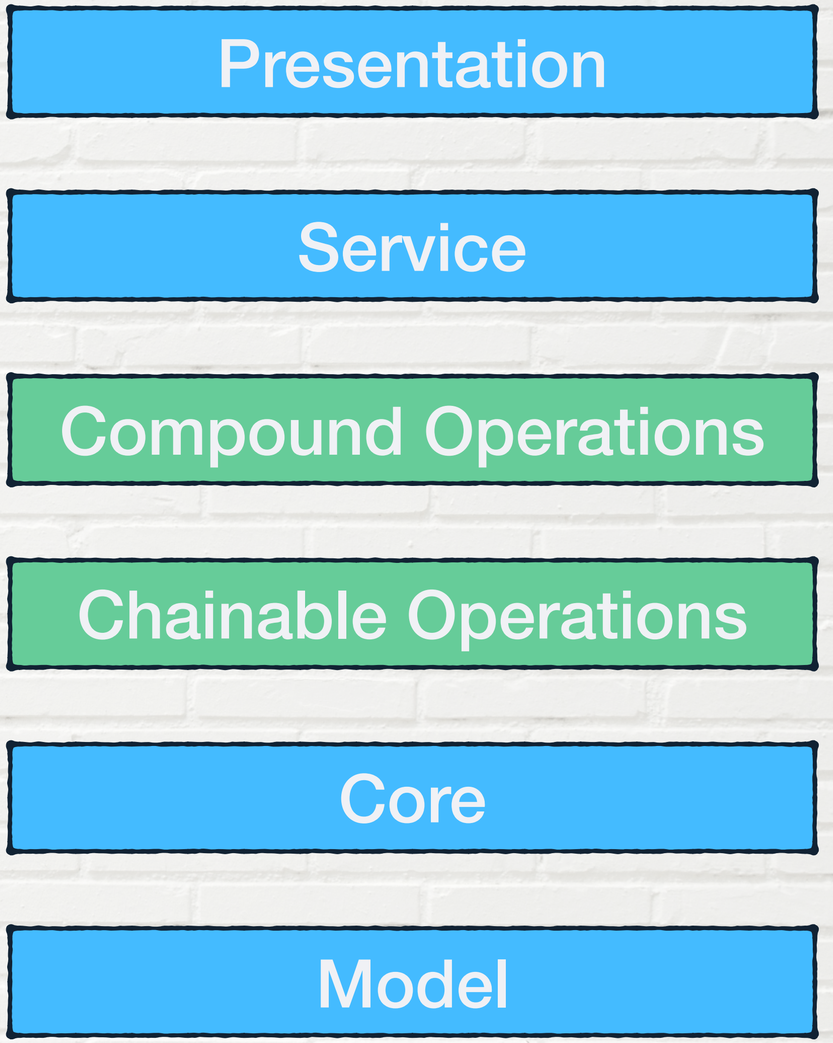
Связываемые и составные операции

- ~~Повторяющийся код~~
- ~~Ад обратных вызовов~~
- ~~Вынужденная обработка ошибок~~



Архитектура с использованием составных операций

- **Связываемые операции** - обертки над одним или несколькими core-компонентами
- **Составные операции** - контейнеры для цепочек связываемых операций



One more thing…



Фреймворк для работы

со связываемыми и составными операциями

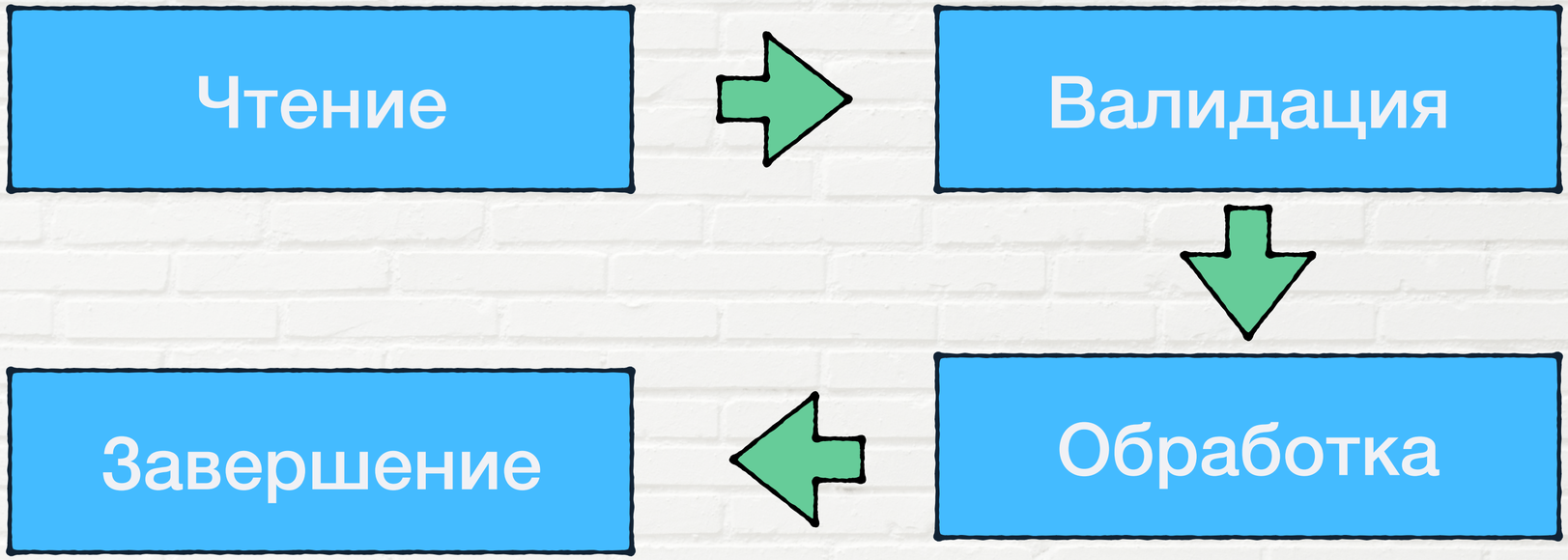
```
class NetworkRequestChainableOperation: ChainableOperationBase {

    private let networkClient: NetworkClient

    init(networkClient: NetworkClient) {
        self.networkClient = networkClient
    }
}
```

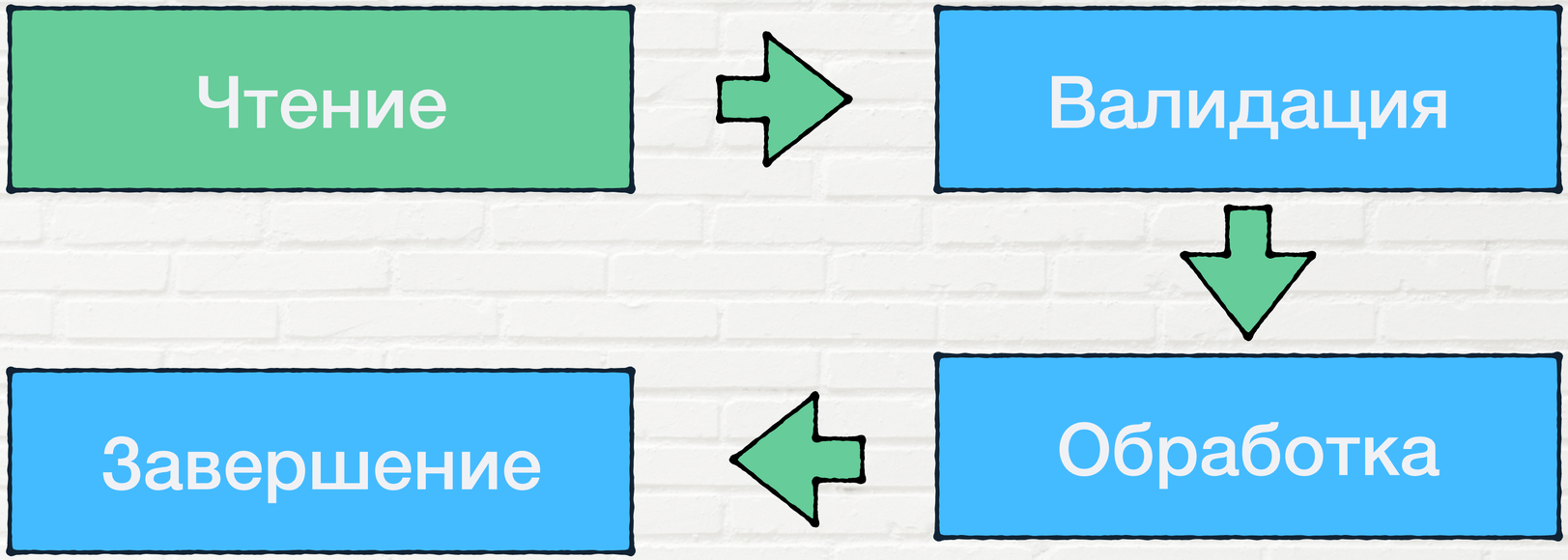
COOperation

ChainableOperationBase



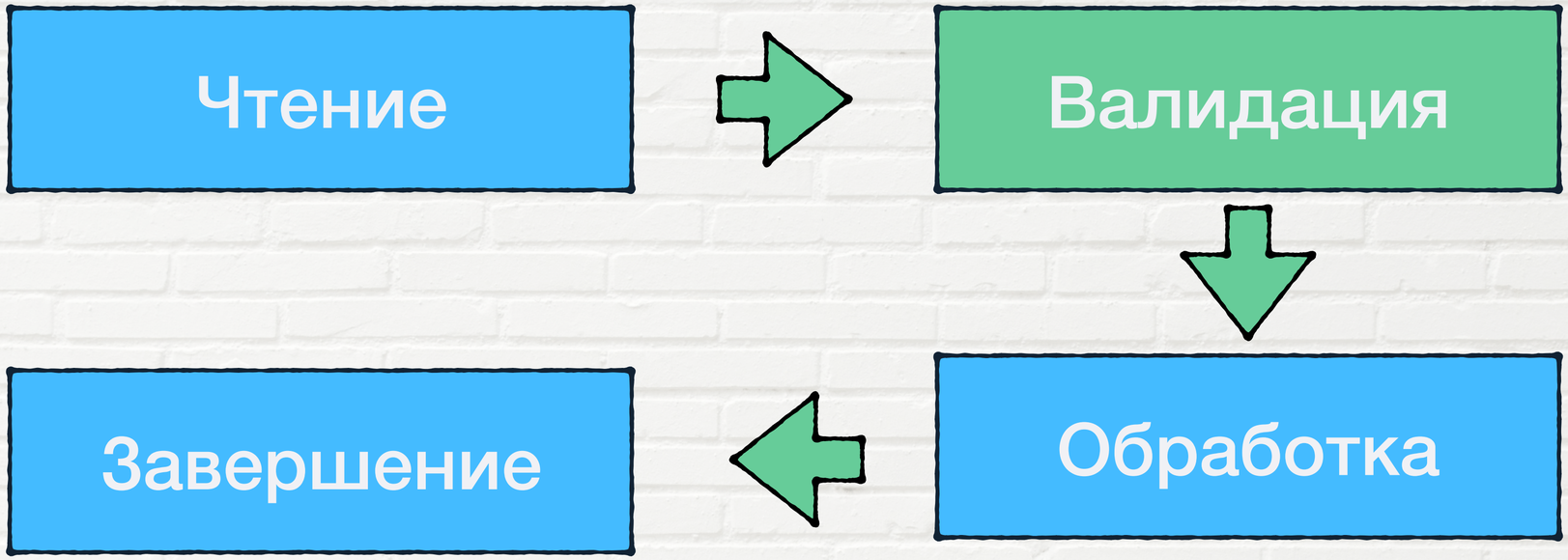
COOperation

ChainableOperationBase



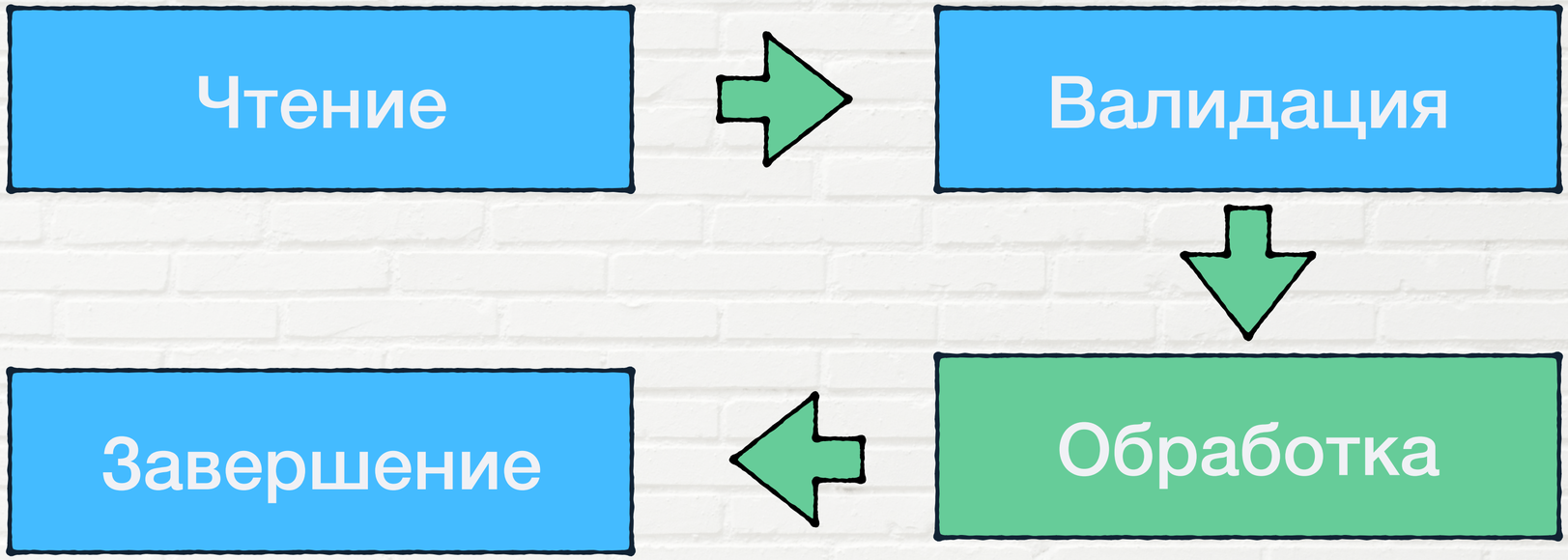
COOperation

ChainableOperationBase



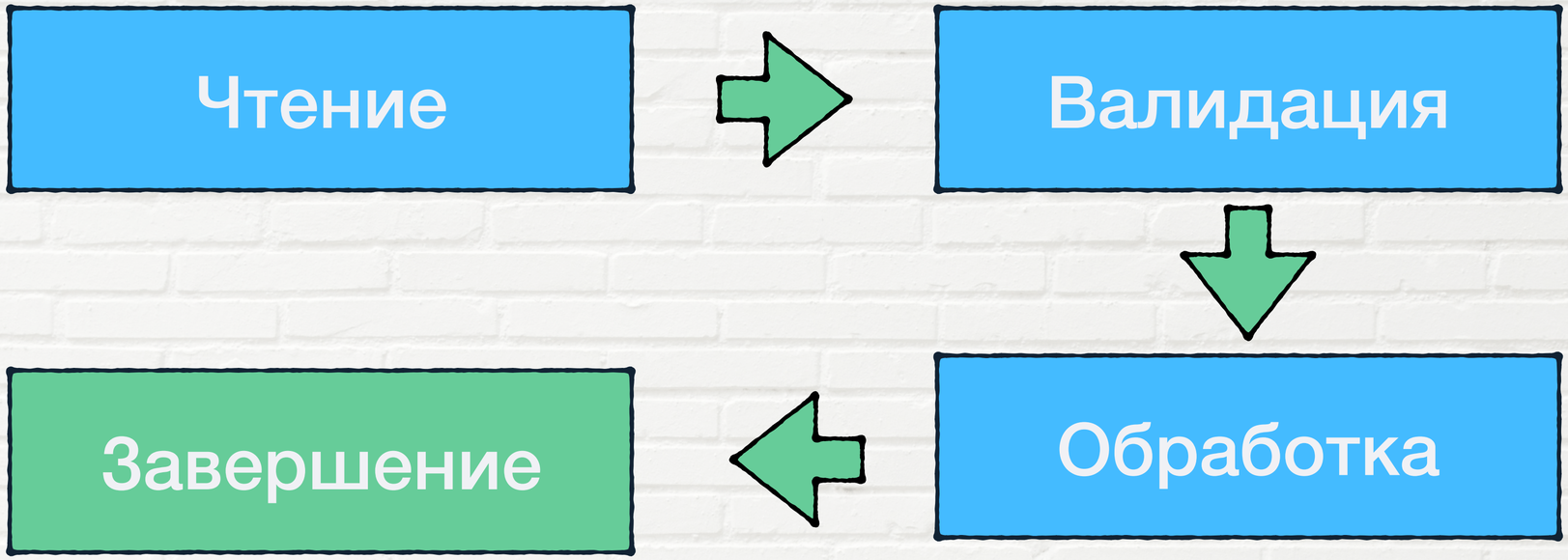
COOperation

ChainableOperationBase



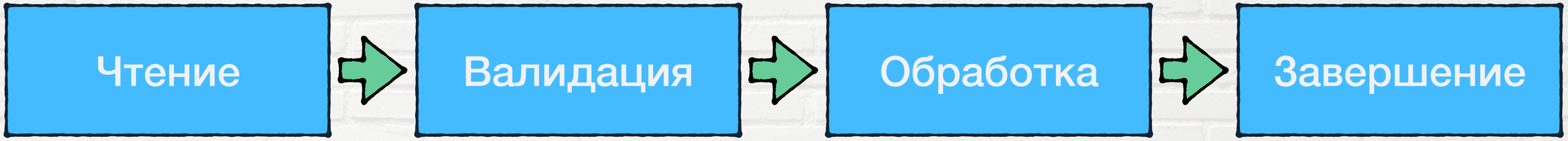
COOperation

ChainableOperationBase



COOperation

ChainableOperationBase

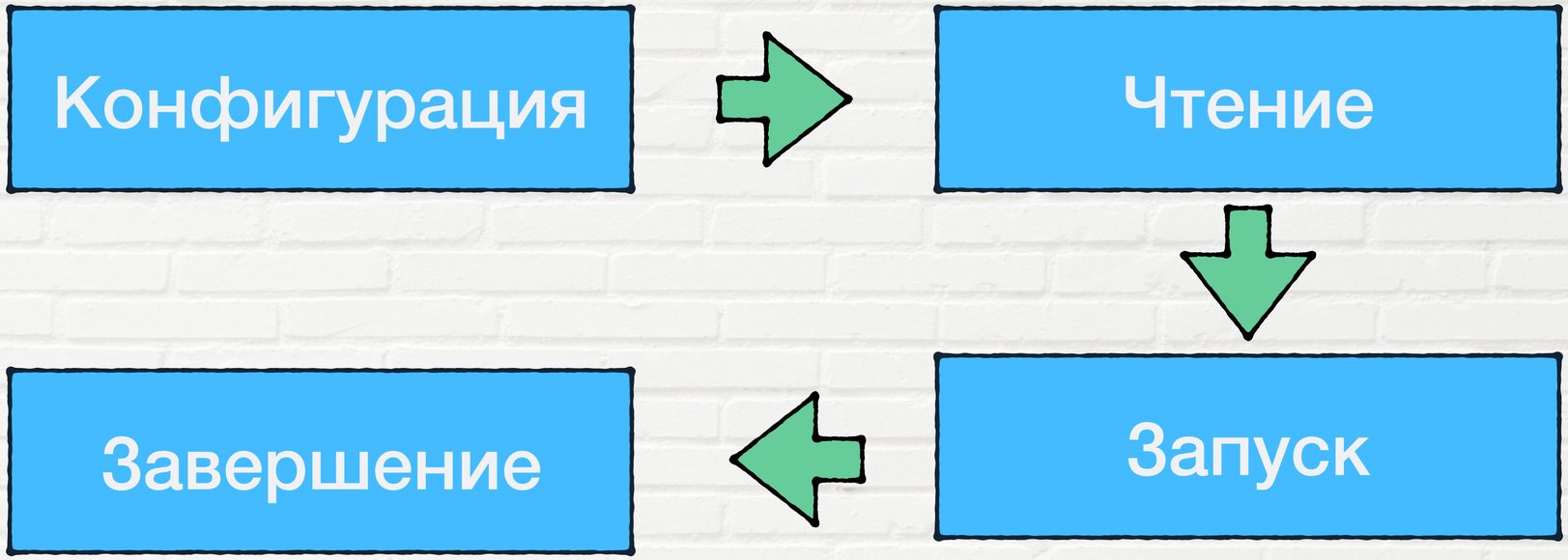



```
override func inputDataClass() -> AnyClass? {
    return NSURLRequest.self
}

override func processInputData(inputData: AnyObject?,
                               completionBlock: ChainableOperationBaseOutputDataBlock) {
    networkClient.performRequest(inputData as! NSURLRequest) { (data, error) in
        completionBlock(data, error)
    }
}
```

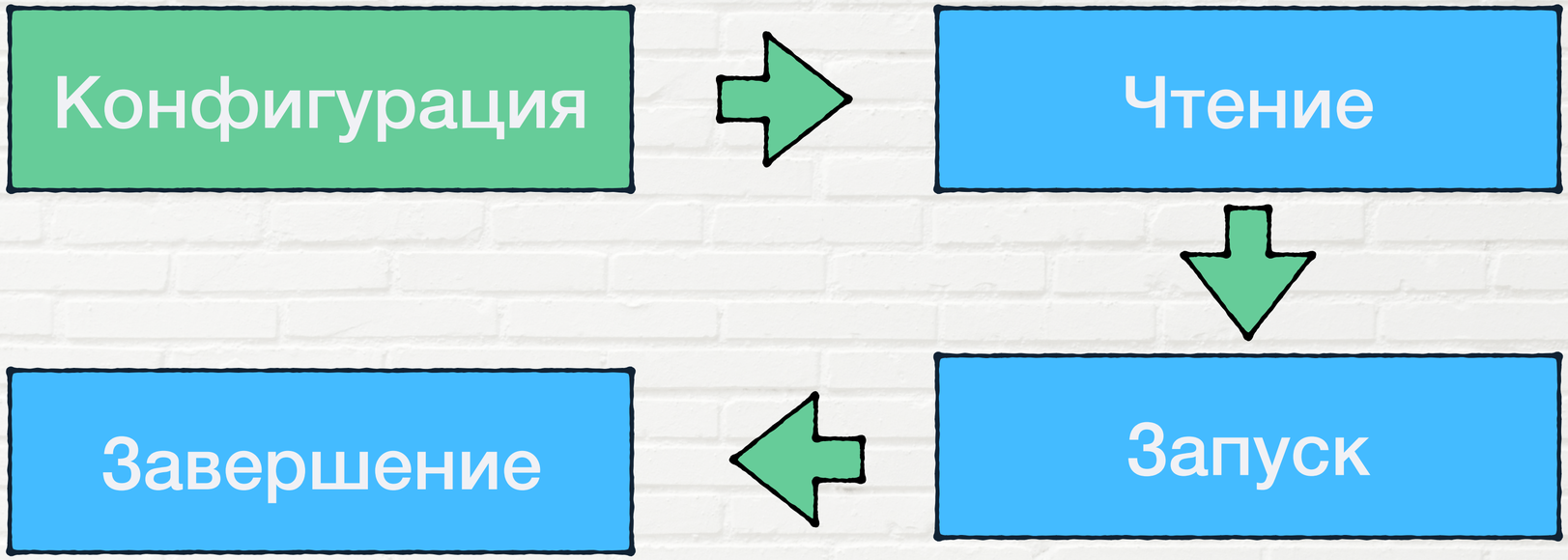
COOperation

ChainableOperationBase



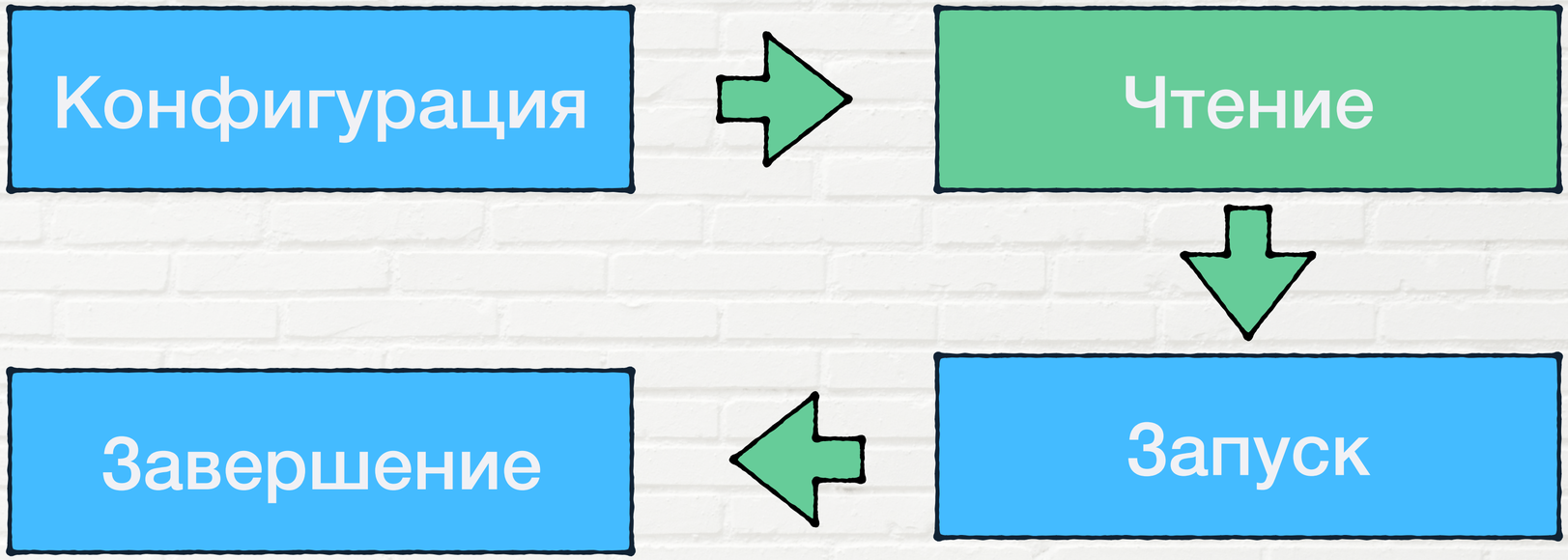
COOperation

CompoundOperation



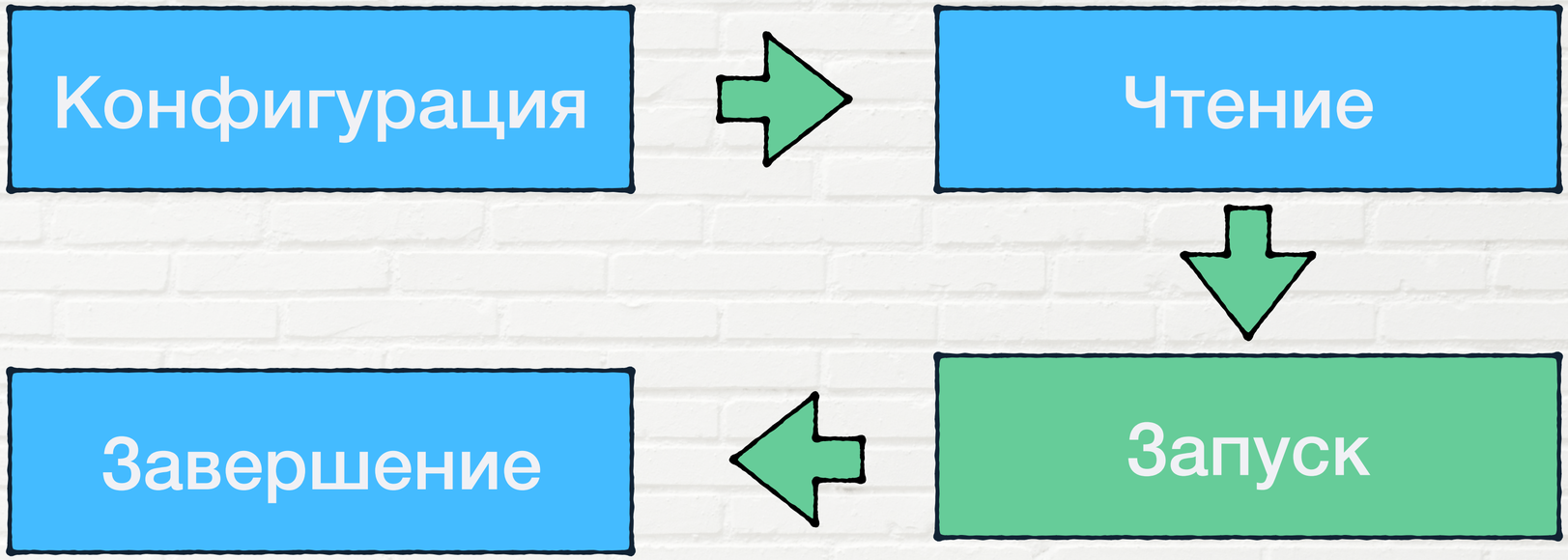
COOperation

CompoundOperation



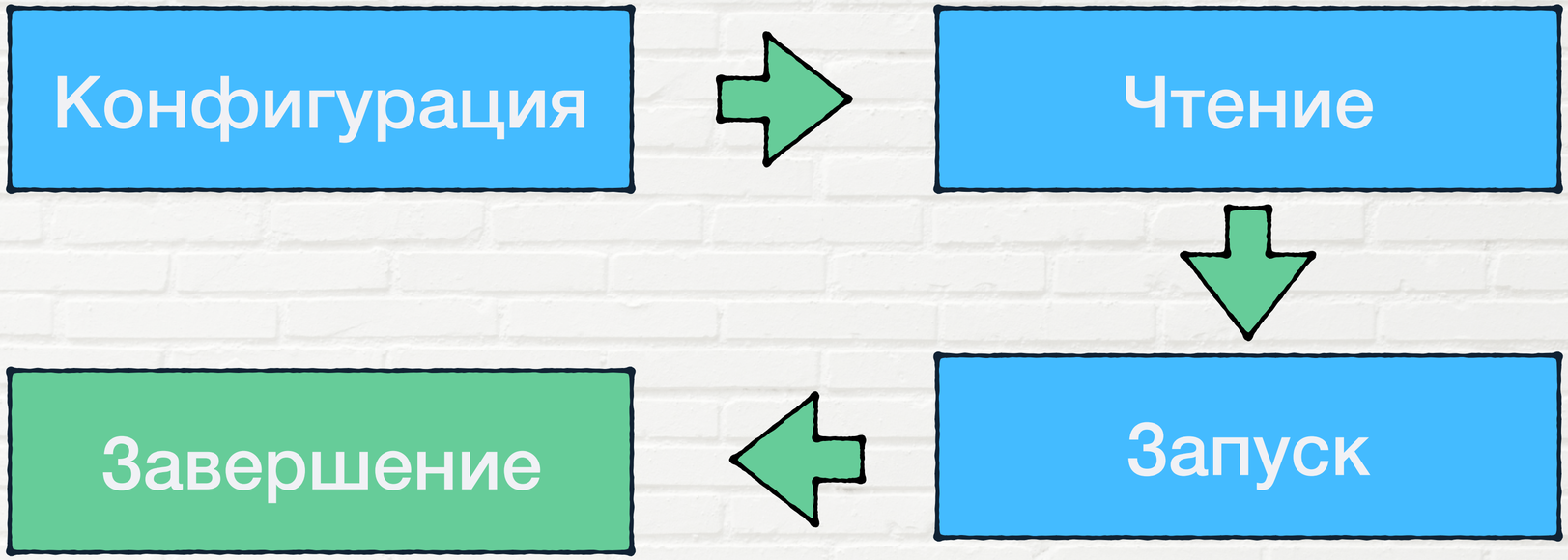
COOperation

CompoundOperation



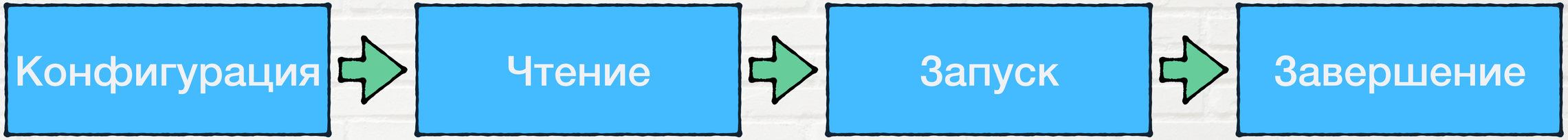
COOperation

CompoundOperation



COOperation

CompoundOperation




```
let chainableOperations = [
    networkRequestOperation,
    deserializationOperation
]

let operation = CompoundOperation()
operation.configureWithChainableOperations(chainableOperations)
```

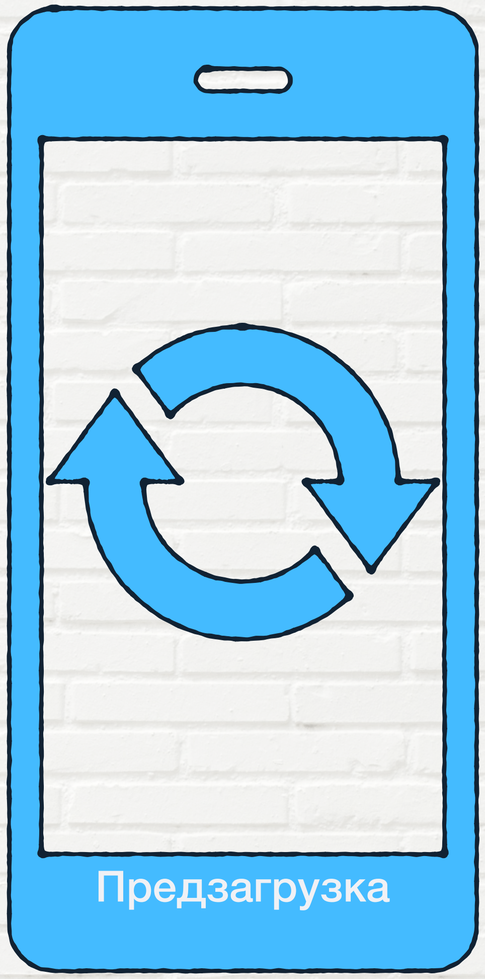
COOperation

CompoundOperation



Фреймворк для работы

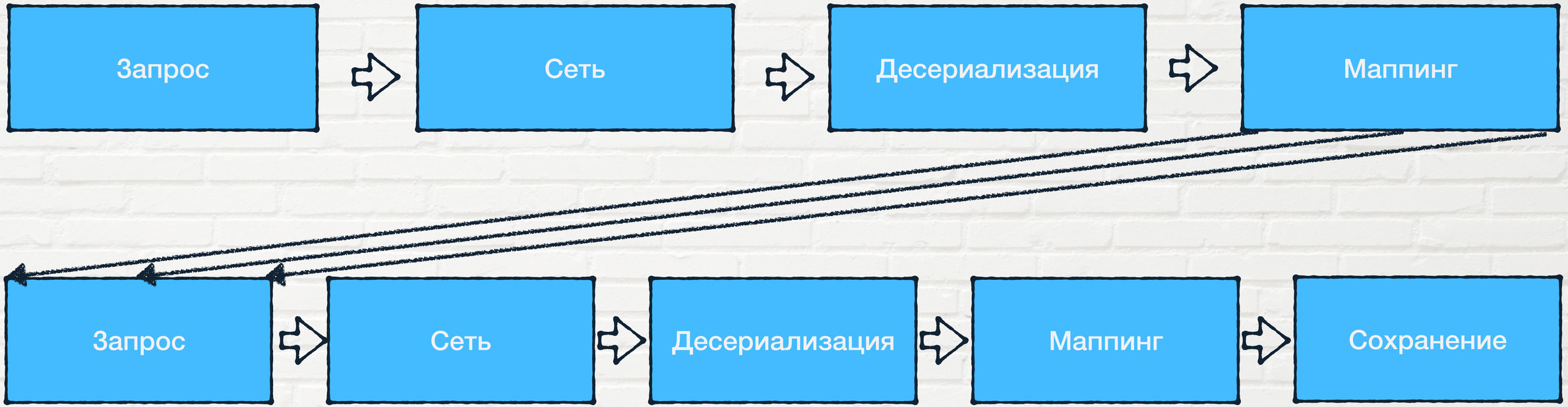
со связываемыми и составными операциями



Сервис предзагрузки

получение подробных данных о всех доступных криптовалютах

ObtainCryptoCurrenciesCompoundOperation



ObtainCryptoCurrencyDetailsCompoundOperation

Сервис предзагрузки

получение подробных данных о всех доступных криптовалютах

```
class NetworkRequestChainableOperation: ChainableOperationBase {

    private let networkClient: NetworkClient

    init(networkClient: NetworkClient) {
        self.networkClient = networkClient
    }
}
```

Сервис предзагрузки

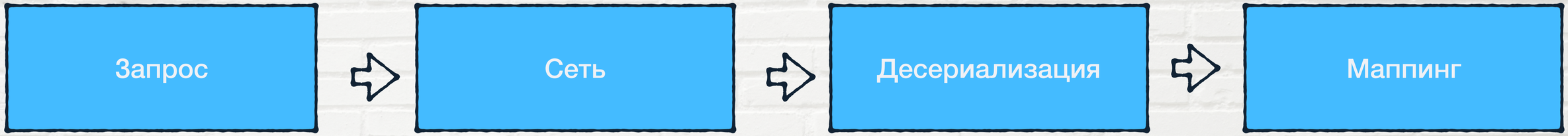
создание связываемых операций

```
override func inputDataClass() -> AnyClass? {
    return NSURLRequest.self
}

override func processInputData(inputData: AnyObject?,
                               completionBlock: ChainableOperationBaseOutputDataBlock) {
    networkClient.performRequest(inputData as! NSURLRequest) { (data, error) in
        completionBlock(data, error)
    }
}
```

Сервис предзагрузки

создание связываемых операций




```
let chainableOperations = [
    obtainCryptoCurrenciesRequestConstruction,
    networkRequest,
    jsonDeserialization,
    cryptoCurrenciesMapping
]

obtainCryptoCurrencies.configureWithChainableOperations(chainableOperations,
                                                        completion: completion)
```

Сервис предзагрузки

создание составной операции получения списка криптовалют




```
let chainableOperations = [
    obtainCryptoCurrencyDetailsRequestConstruction,
    networkRequest,
    jsonDeserialization,
    cryptoCurrencyDetailsMapping,
    saveToStorage
]

obtainCryptoCurrencyDetails.configureWithChainableOperations(chainableOperations,
                                                             input: cryptoCurrencyId,
                                                             completion: completion)
```

Сервис предзагрузки

создание составной операции получения деталей криптовалюты

```
func precacheCryptoCurrencies() {
    obtainCryptoCurrencies.completion = { (cryptoCurrenciesIds, error) in
        let precacheOperations = cryptoCurrenciesIds.map({ (cryptoCurrenciesId) -> CompoundOperation in
            return ObtainCryptoCurrencyDetails(cryptoCurrencyId: cryptoCurrenciesId)
        })

        self.operationQueue.addOperations(precacheOperations, waitUntilFinished: false)
    }

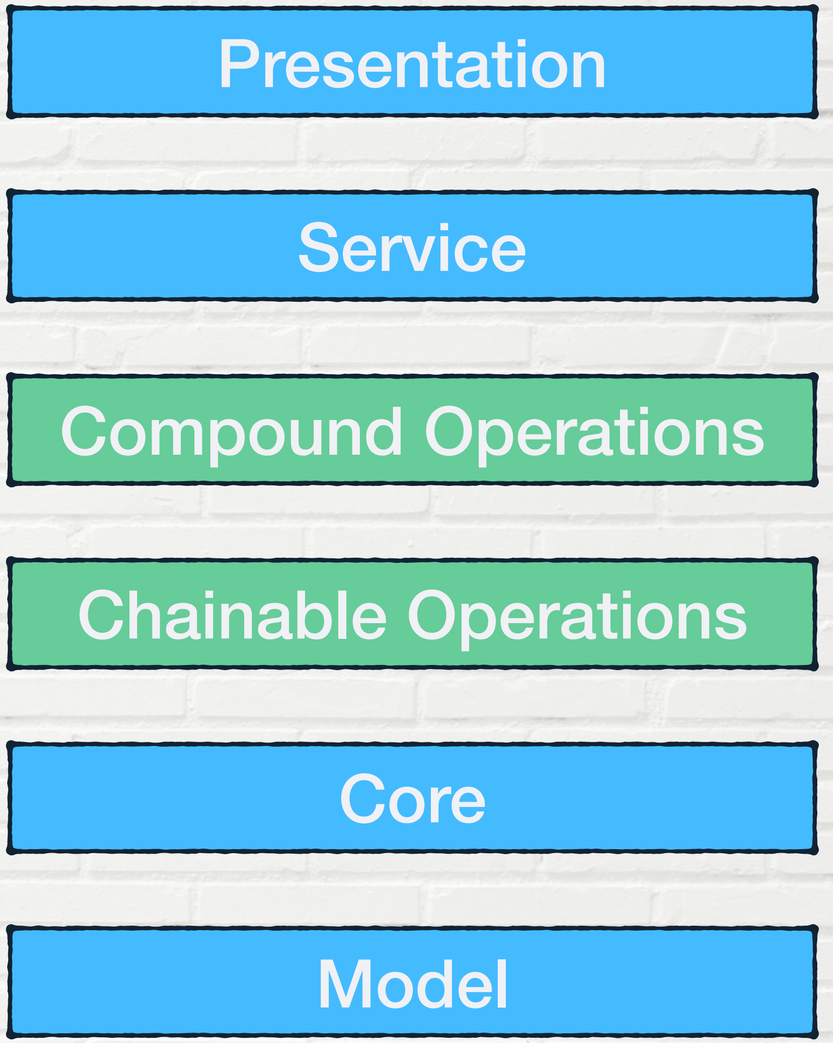
    operationQueue.addOperation(obtainCryptoCurrencies)
}
```

Сервис предзагрузки

запуск кооперации обеих составных операций

Архитектура с использованием составных операций

- Декларативное описание бизнес-логики
- SOLID из коробки
- Переиспользуемый код
- Тестируемость



Архитектура с использование составных операций

- Используется в крупных реальных проектах
- Используется в домашних проектах
- Открытый исходный код



COOperation

- COOperation 2.0 (Objective-C -> Swift)
- Управление сложными последовательностями составных операций
- Core-слой как фреймворк

О чем поговорили

- Роли сервисов и core-компонентов
- Проблемы классических реализаций сервисов
- Декомпозиция сетевого слоя
- Связываемые и составные операции
- COOperation

Спасибо

Глеб Новик
novik.gleb@gmail.com



COOperation
https://github.com/strongself/COOperation



Podlodka
https://soundcloud.com/podlodka/

