

Автоматизация работы с архитектурой as Code

Руслан Сафин · Бындюсофт

DotNext · 25–26 сентября 2026

Руслан Сафин

Технический директор и архитектор
в [Бындюсофт](#)

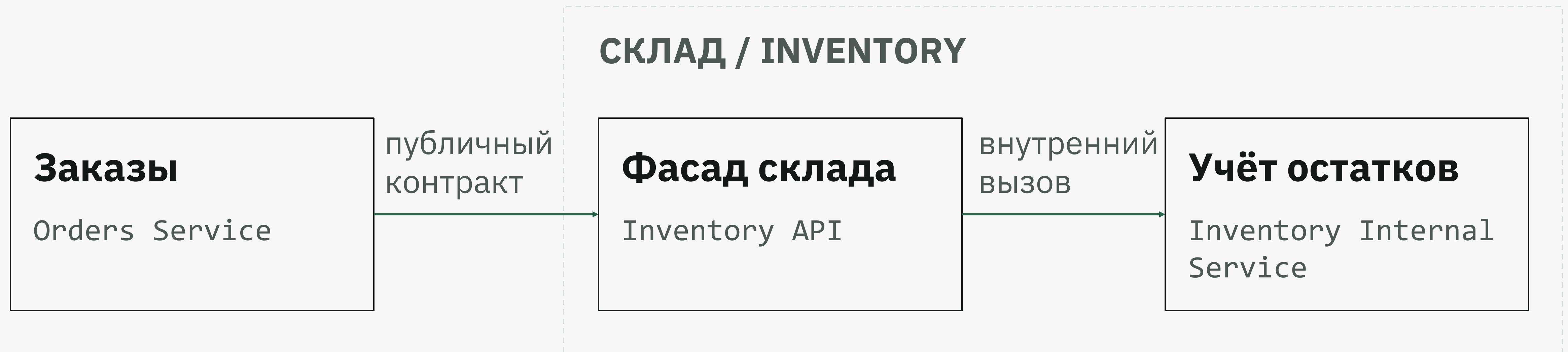
Автор и преподаватель [курса](#)
по микросервисной архитектуре в ИТМО

Член программных комитетов [HardFest](#),
[CodeFest](#), [TechLeadConf](#) и [UWDC](#)



Заказ резервирует товар

Три отдельных микросервиса



Фасад реализует публичный контракт

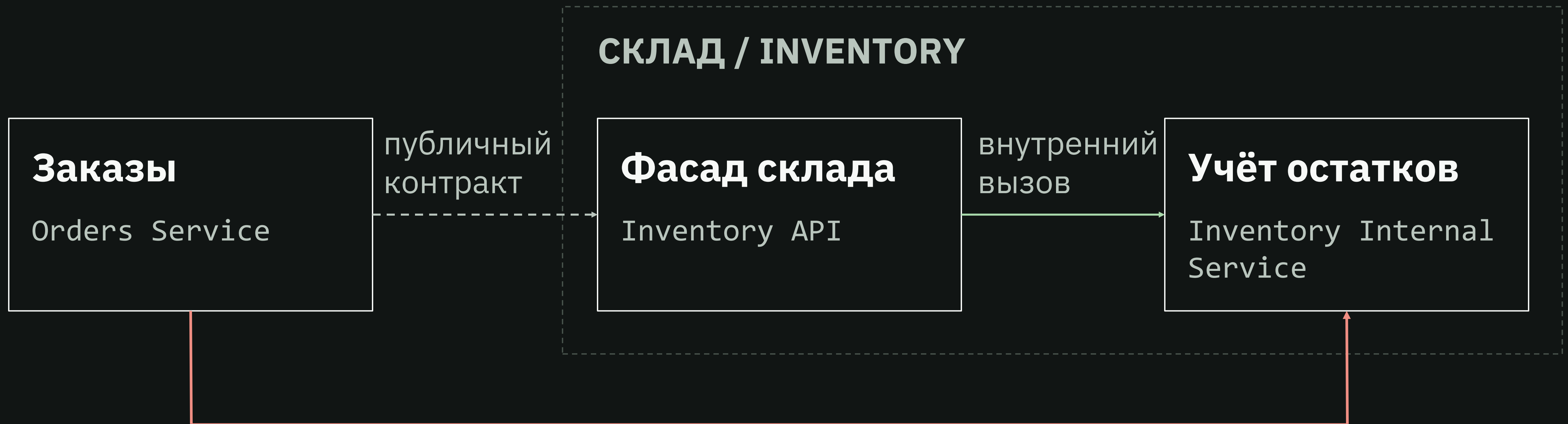
В заказах появилось резервирование товара

Заказы должны резервировать товар на складе.

Во ~~внутреннем~~ сервисе склада уже есть подходящий метод.

Разработчик подключает его напрямую. Функциональные тесты проходят.

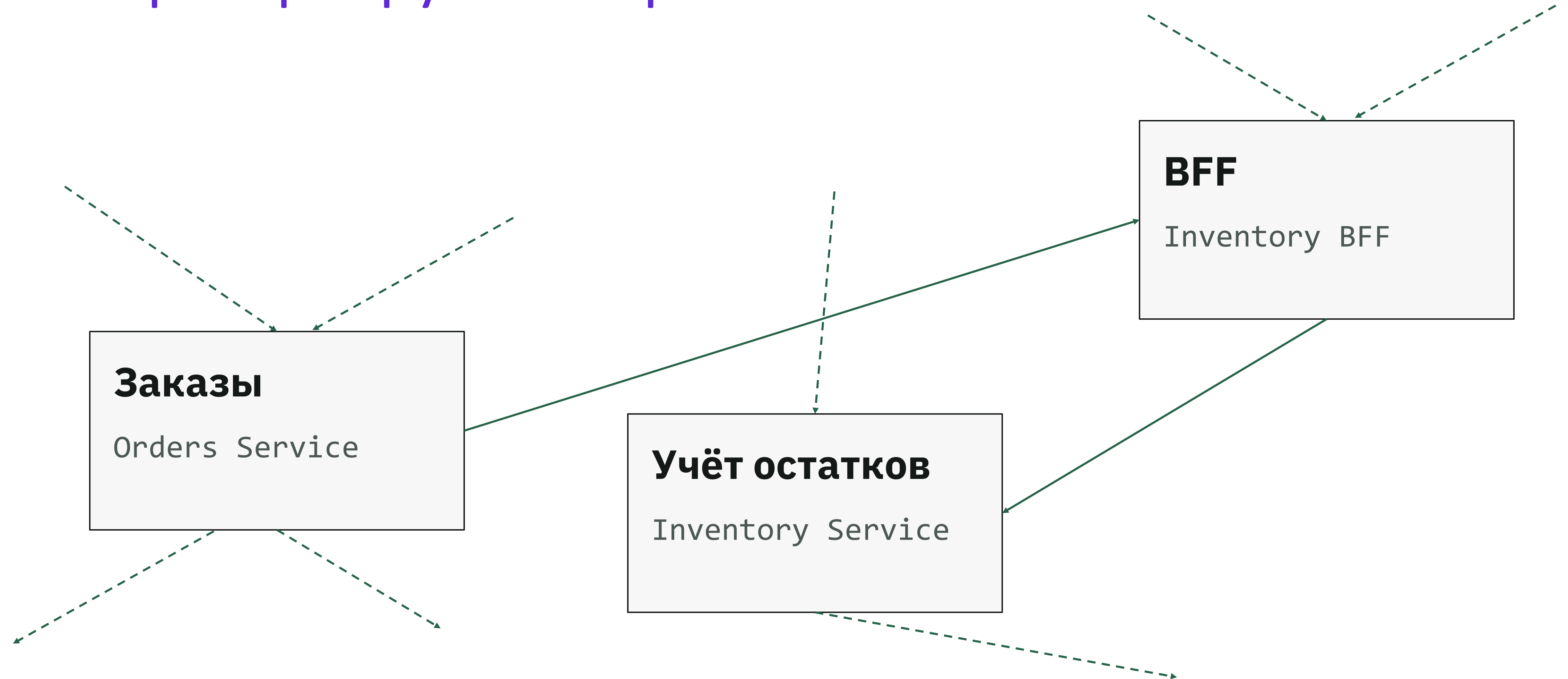
А прямую зависимость мы принимаем?



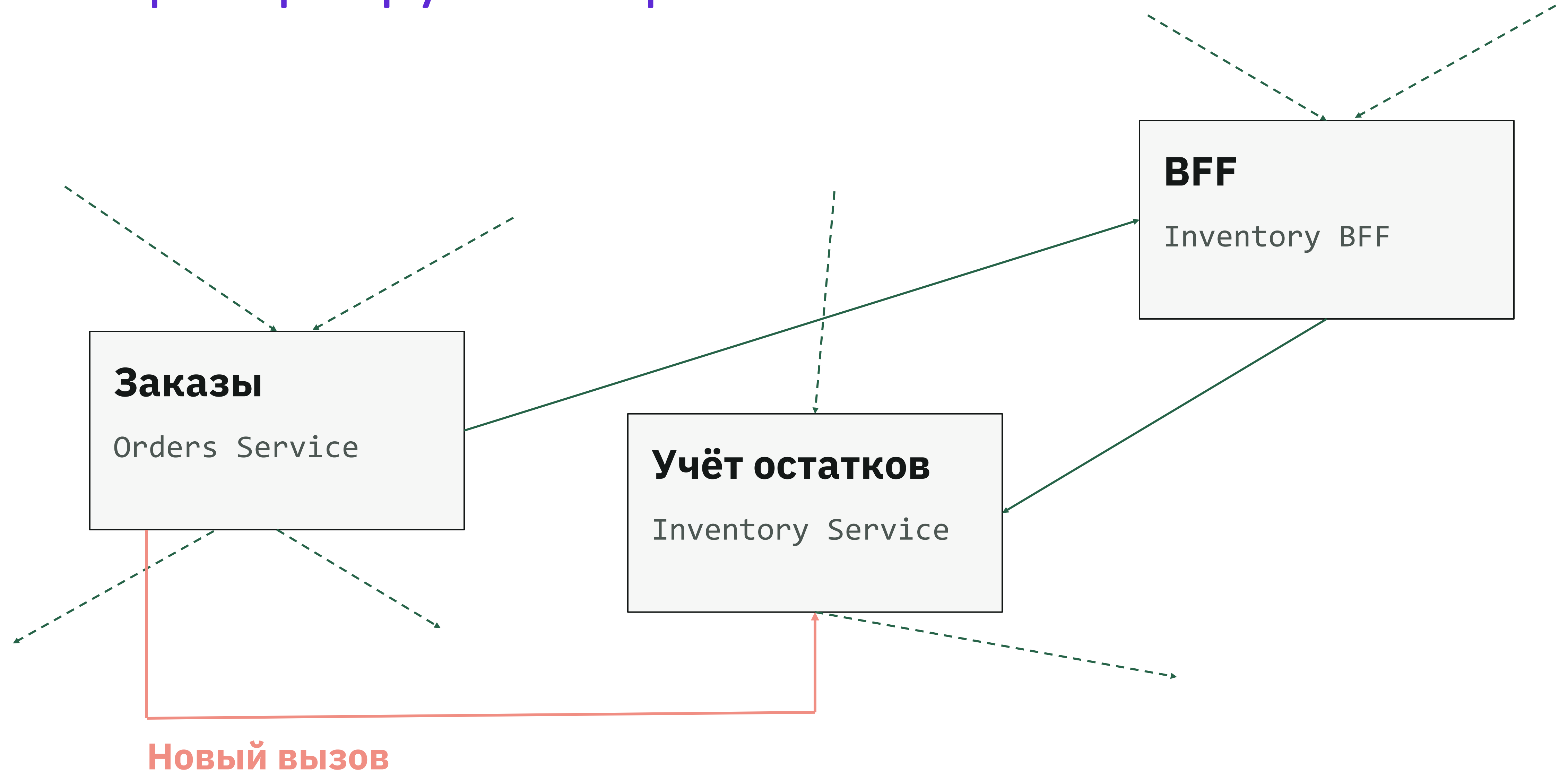
Новый вызов: заказы обходят фасад

Резервирование работает. Заказы теперь зависят от внутреннего сервиса склада.

Заказ резервирует товар



Заказ резервирует товар



В PR изменилась одна строка

```
@startuml
- Rel(orders_svc, inventory_api, "")
+ Rel(orders_svc, inventory_svc, "")
  Rel(inventory_api, inventory_svc, "")
@enduml
```

Можно ли принять такое изменение?

Репозиторий AAST

Примеры и принципы

Каталог patterns.md, ADR и тесты для микросервисов и .NET-монолита

CLI и библиотека

Проверка схем, сравнение с конфигами, метрики и просмотр модели

Под свой проект

Добавляем правила команды и подключаем свои источники данных

github.com/Byndyusoft/aact · npm-пакет aact 3.0.1 · .NET для модульного монолита

План

Архитектура as Code

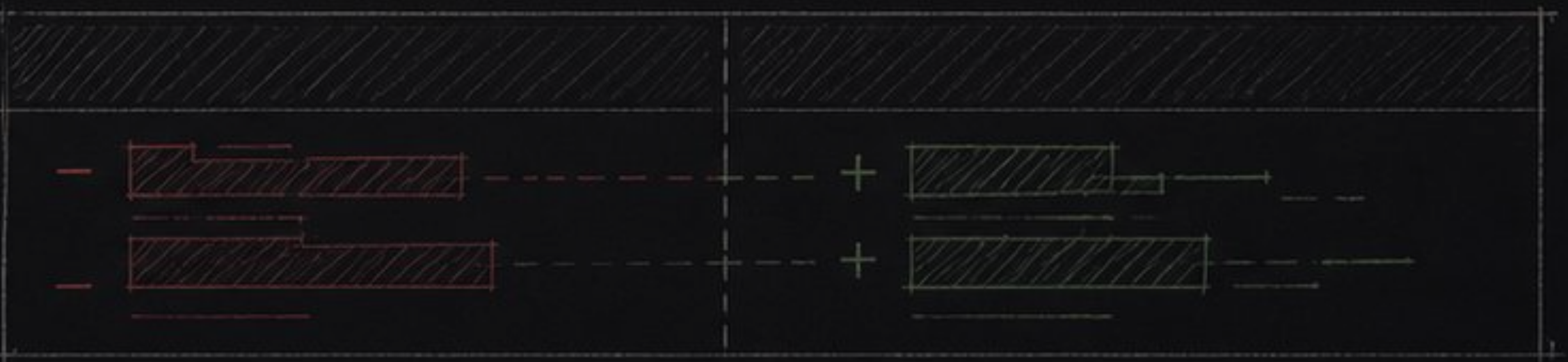
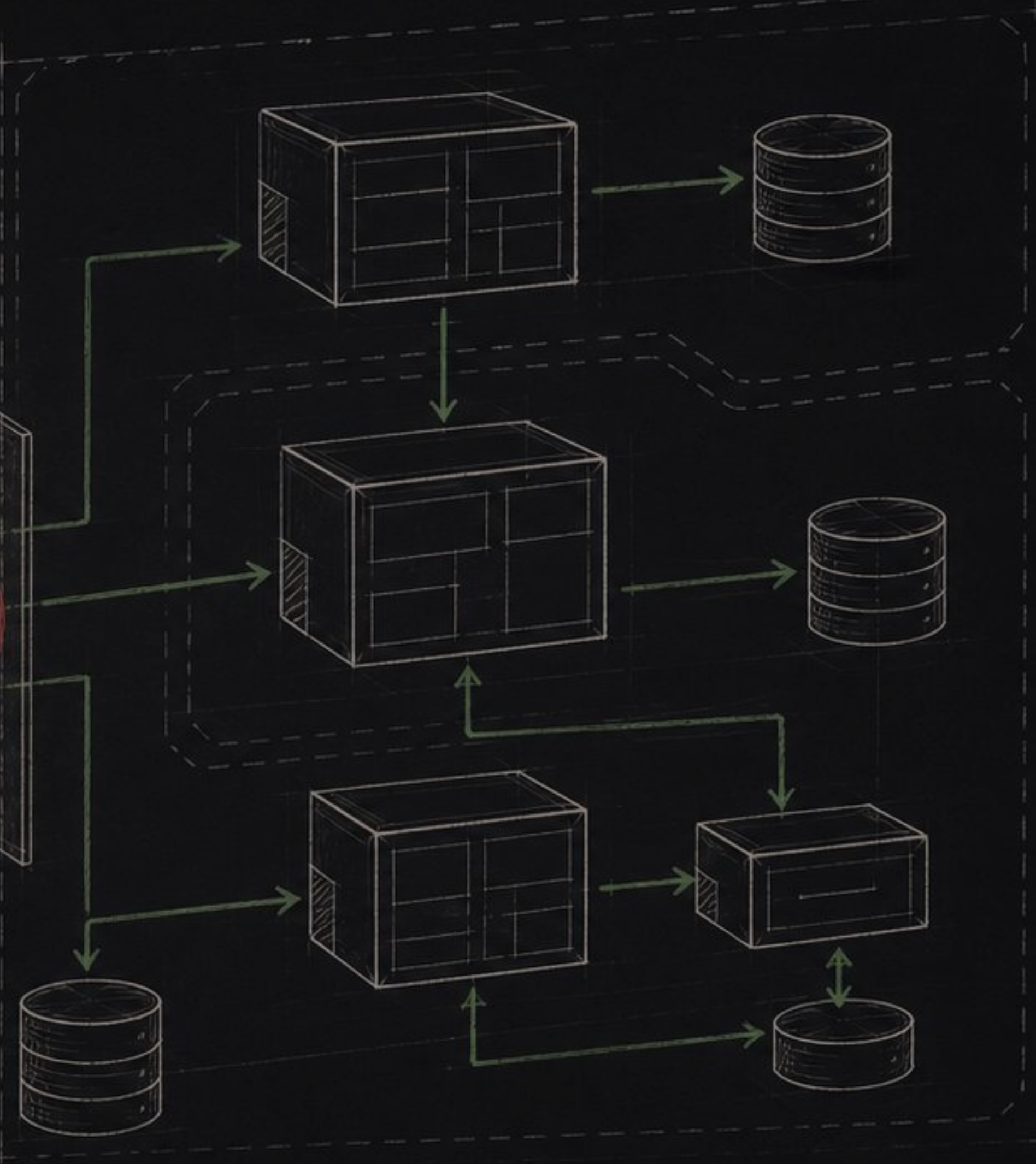
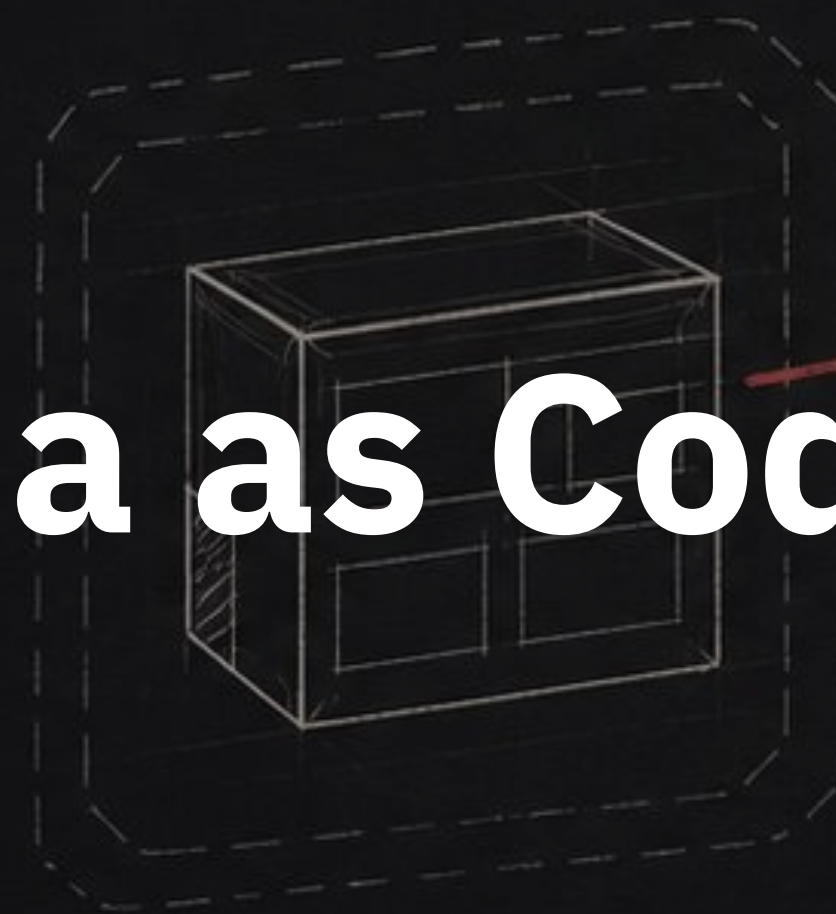
Покрытие архитектуры тестами

Три года: от идеи к инструменту

Модульный монолит (на .NET)

Часть 1

Архитектура as Code

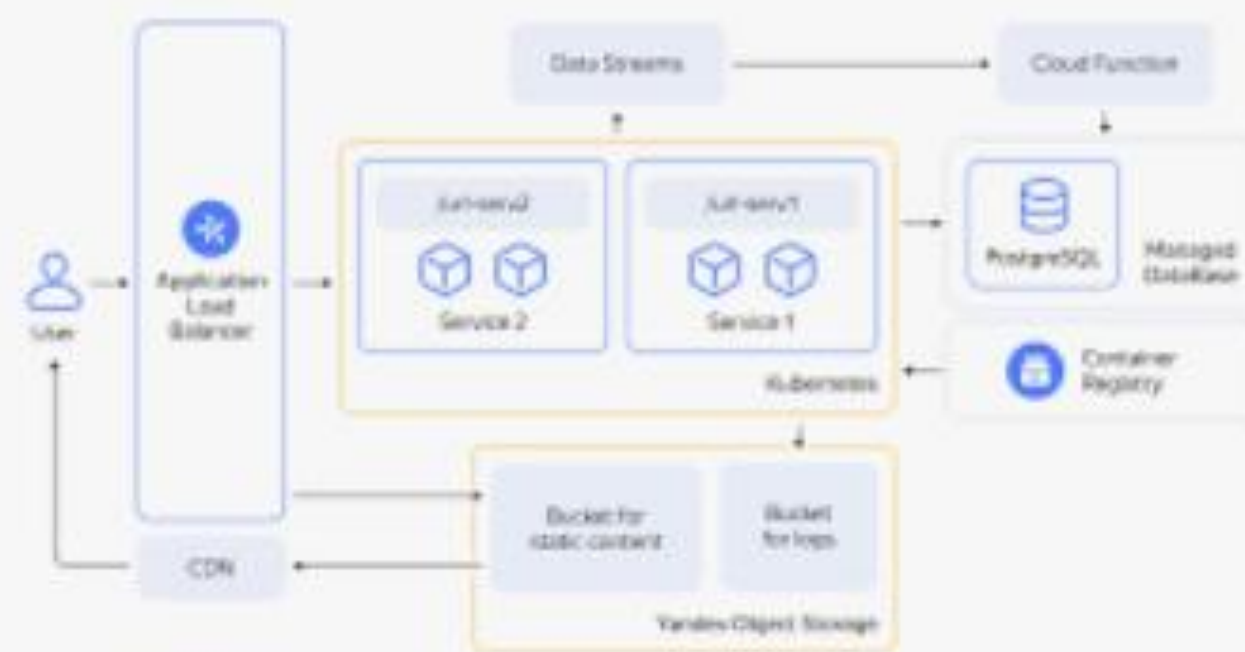
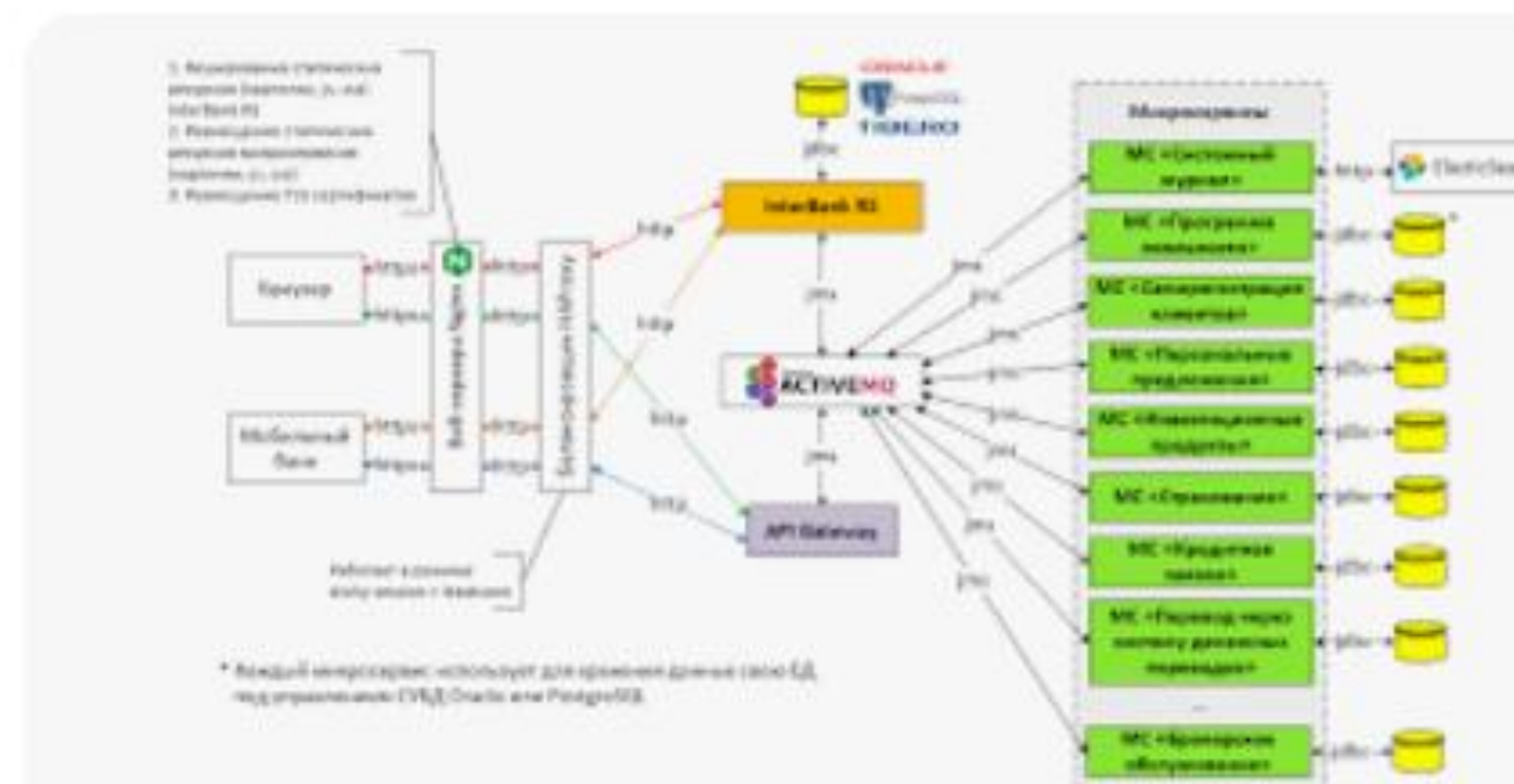
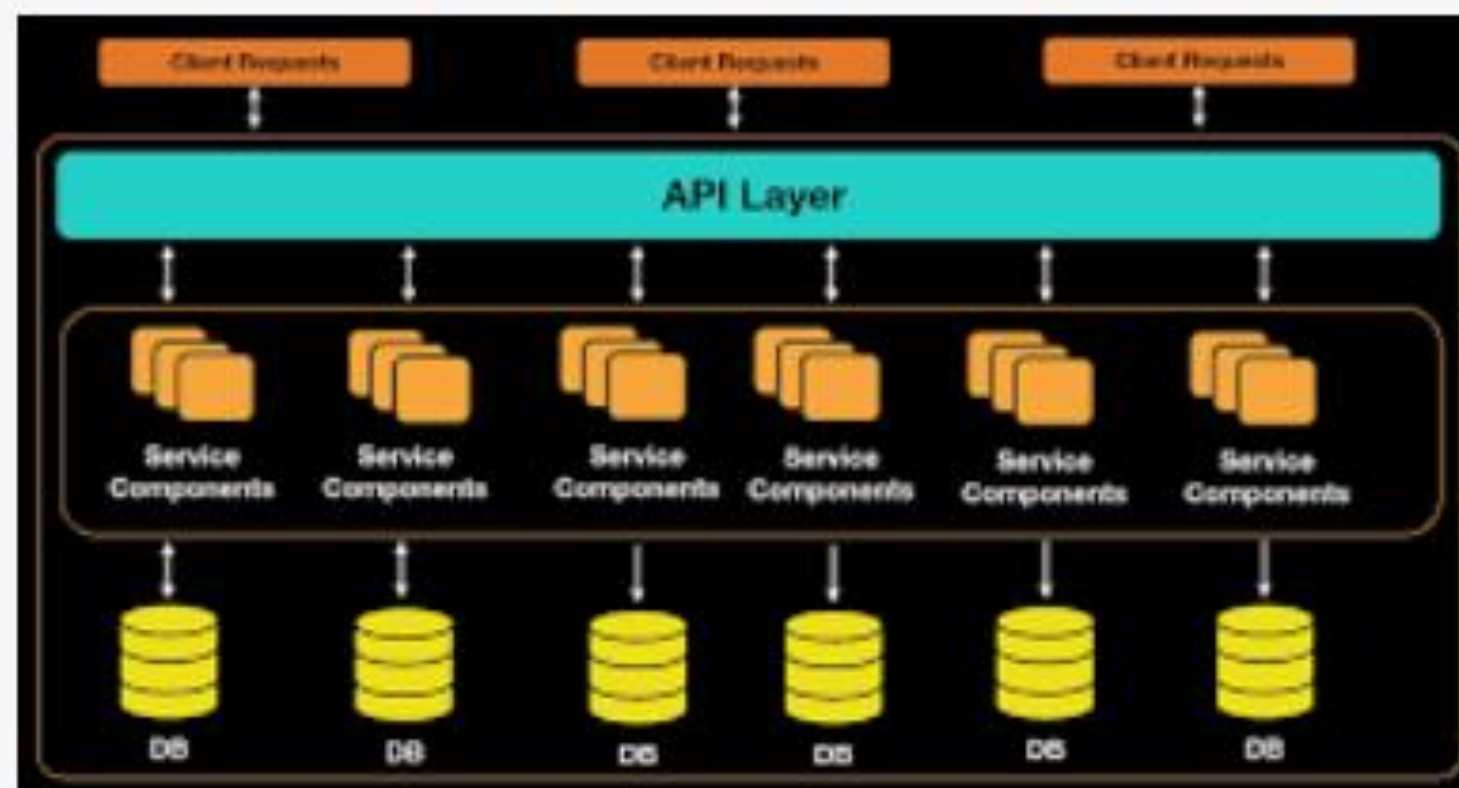


ИТ-архитектура

«Фундаментальная организация системы, реализованная в её компонентах, их взаимоотношениях друг с другом и средой и принципах, определяющих её конструкцию и развитие»

© стандарт ANSI/IEEE Std 1471-2000

Архитектура — картинка



Как мы описываем архитектуру

Вариант 1. Картинка

+

Наглядность

—

Версионирование

Работа с ветками

Проверка соответствия коду

Возможности автоматизации

Ничего не говорит о причинах и принципах

Как мы описываем архитектуру

Вариант 2. Код



- Версионирование
- Работа с ветками
- Проверка соответствия коду
- Возможности автоматизации

Связь с ADR



Наглядность

Как мы описываем архитектуру

Вариант 3. Код → картинка



Наглядность
Версионирование
Работа с ветками

**Проверка соответствия коду и другие
возможности автоматизации**



Схема хранится текстом, картинка рисуется из неё автоматически

Чем описывают

- C4-PlantUML
- Structurizr DSL
- Mermaid C4
- ...

Схему храним рядом с кодом: меняем в ветке и обсуждаем в PR.

new.puml > {} new

```
1  @startuml
2  !include https://raw.githubusercontent.com/adria
3
4  LAYOUT_WITH_LEGEND()
5
6  {
7  @enduml
```

Legend

person

system

container

external person

external system

Микросервисы в C4-PlantUML

```
System_Ext(goods, "Goods Repo")
```

```
System_Ext(stock, "Stock")
```

```
Container(bff, "BFF", "NestJS")
```

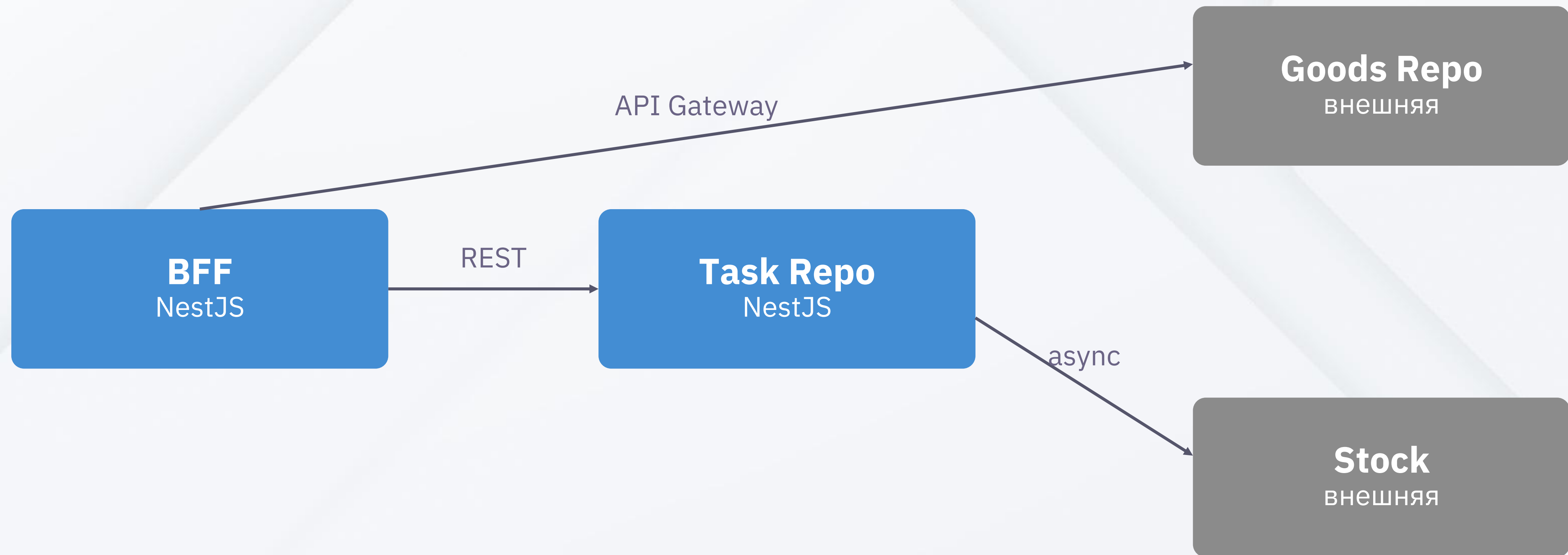
```
Container(task_repository, "Task Repo", "NestJS")
```

```
Rel(bff, goods, "Get products", "API Gateway")
```

```
Rel(task_repository, stock, "Send task", $tags="async")
```

```
Rel(bff, task_repository, "Get task", "REST")
```

...и картинка, которая из них рисуется



```

i(reception_documents_storage_adapter, lookup, "Get stores", "<DP>/locations/v1/stores")
i(reception_bff, "c->", "REST, Socket")

i(reception_bff, reception_aggregator, "")
i(reception_bff, reception_tasks_storage, "")
i(reception_bff, reception_documents_storage, "")
i(reception_bff, stock, "")

i(reception_aggregator, reception_tasks_storage, "")
i(reception_aggregator, reception_documents_storage, "")
i(reception_aggregator, products_adapter, "")
i(D(reception_aggregator, reception_orchestrator_handlers, ""))
i(reception_aggregator, stock, "Get zone for sorting", "<DP>/stocks/vms-slotting/v1 ")

i(printers_service, reception_documents_storage, "")
i(printers_service, reception_tasks_storage, "")
i(U(printers_service, printers_adapter, ""))
i(printers_service, products_adapter, "")
i(printers_service, rms_index, "", "<DP>/orders/supplier-orders/orders/v1")
i(printers_service, operational_units, "", "<DP>/operational-units/v1")
i(printers_service, lookup, "", "<DP>/locations/v1/stores")
i(printers_service, stock, "", "<DP>/locations/topology-manager/v2")
i(printers_service, auth_adapter, "")

i(reception_orchestrator_handlers, stock_management_adapter, "Send containers for placement and sorting")
i(stock_management_adapter, stock, "Send placement and sorting tasks", "sorting-task-create-xxx, placement-task-create-xxx", $stage="async")

i(reception_orchestrator_handlers, returns, "Get info about return by asccv[n](<a href='\"https://github.com/adao/bigfish-ums-reception--deploy--imgu/blob/389366e389878d416ed857ff45c466c25dd91e2/lobarnetes/microservices/reception-orchestrator-handlers.yml#L139C1-L139C21\">https://github.com/adao/bigfish-ums-reception--deploy--imgu/blob/389366e389878d416ed857ff45c466c25dd91e2/lobarnetes/microservices/reception-orchestrator-handlers.yml#L139C1-L139C21\">url</a>))",
i(D(reception_bff, reception_orchestrator_handlers, "Start/execute task/container")
i(reception_orchestrator_handlers, reception_bff, "Trigger for update data")
i(U(reception_bff, printers_service, ""))

i(reception_orchestrator_handlers, camunda, "")
i(reception_orchestrator_handlers, reception_documents_storage, "")
i(reception_orchestrator_handlers, reception_tasks_storage, "")

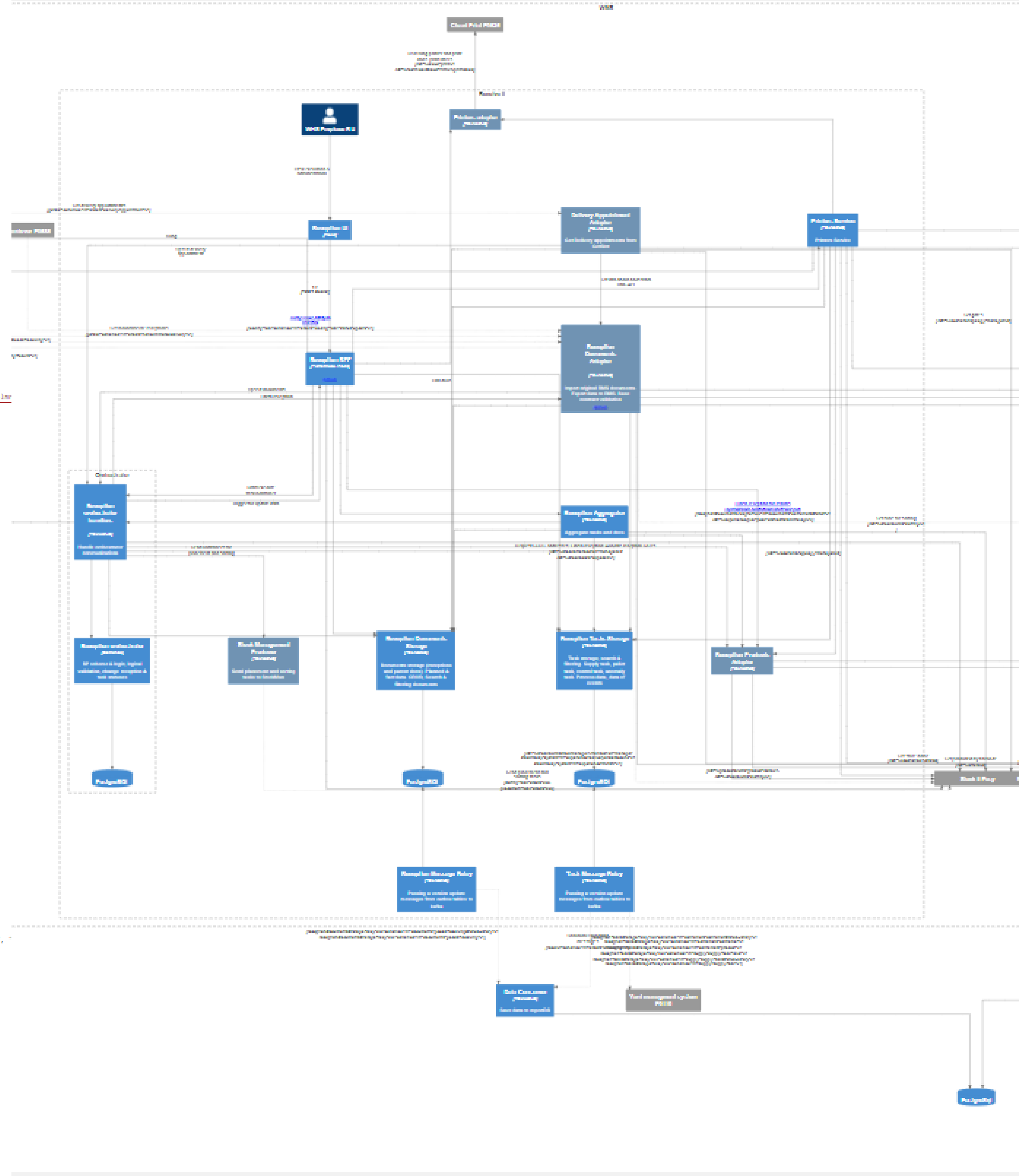
i(reception_tasks_storage, reception_tasks_storage_db, "")
i(camunda, reception_orchestrator_db, "")
i(reception_documents_storage, reception_documents_storage_db, "")
i(D(reception_documents_storage_adapter, reception_orchestrator_handlers, "Insert documents")
i(U(reception_orchestrator_handlers, reception_documents_storage_adapter, "Finish reception")
i(R(reception_orchestrator_handlers, products_adapter, ""))
i(reception_orchestrator_handlers, stock, "Request[<SCC>]<addresses>[]<init>reception:[<allocate>reception[<idU>n", "<DP>/stocks/transaction-manager/v2/n)
P>/stocks/data-exposer/v1")

i(U(reception_tasks_storage_relay, reception_tasks_storage_db, ""))
i(U(reception_documents_storage_relay, reception_documents_storage_db, ""))

i(ums_employee, reception_ui, "Task execution & administration")

i(reception_bff, products_adapter, "")

```



Что это даёт

- Работаем со схемой как с кодом:
 - храним историю изменений
 - создаём ветки, PullRequest'ы, делаем merge
 - проводим ревью схемы вместе с кодом
- Запускаем проверки автоматически



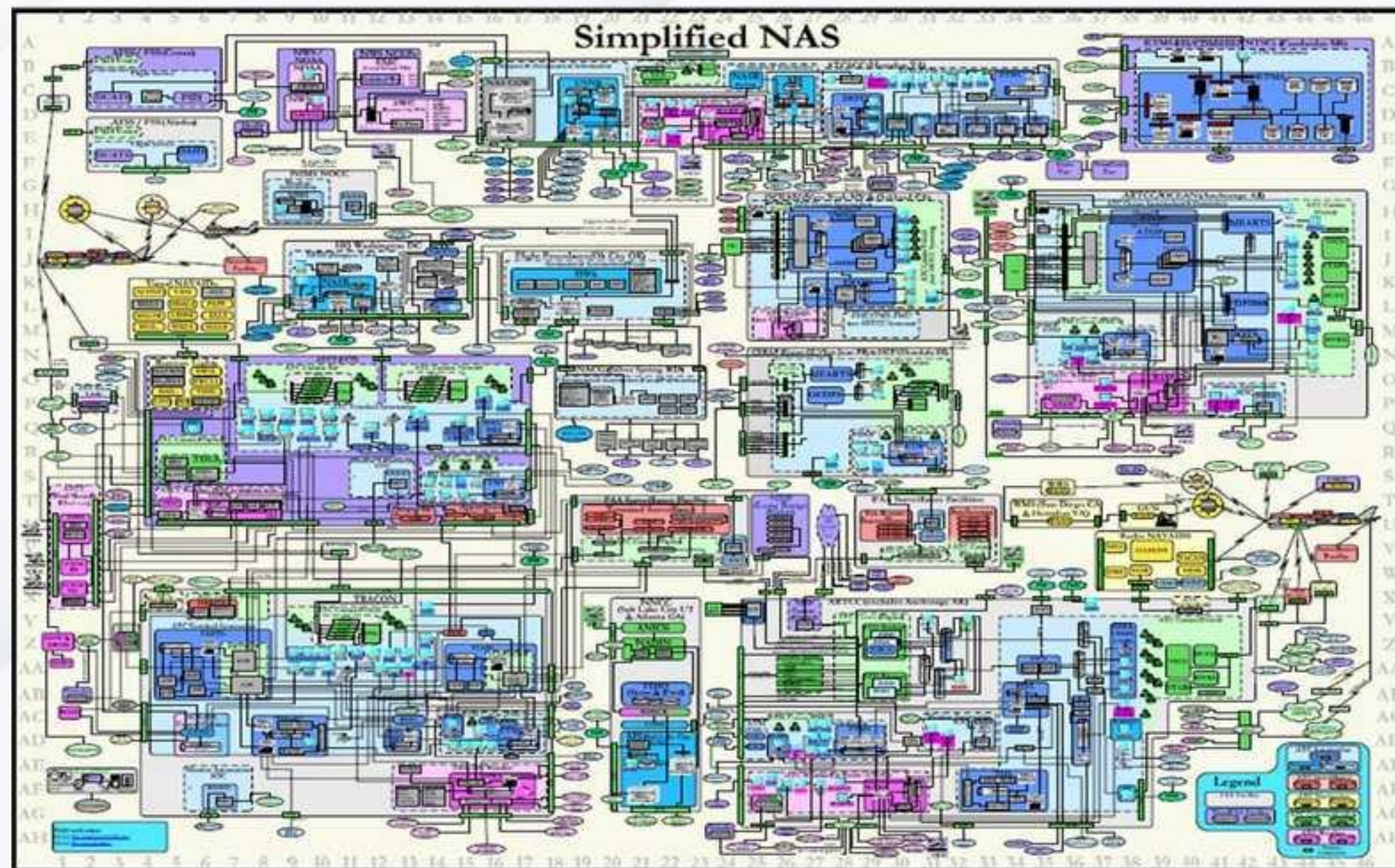
Проблемы архитектурных схем

1. Неактуальность
2. Декларативность
3. Отсутствие контроля

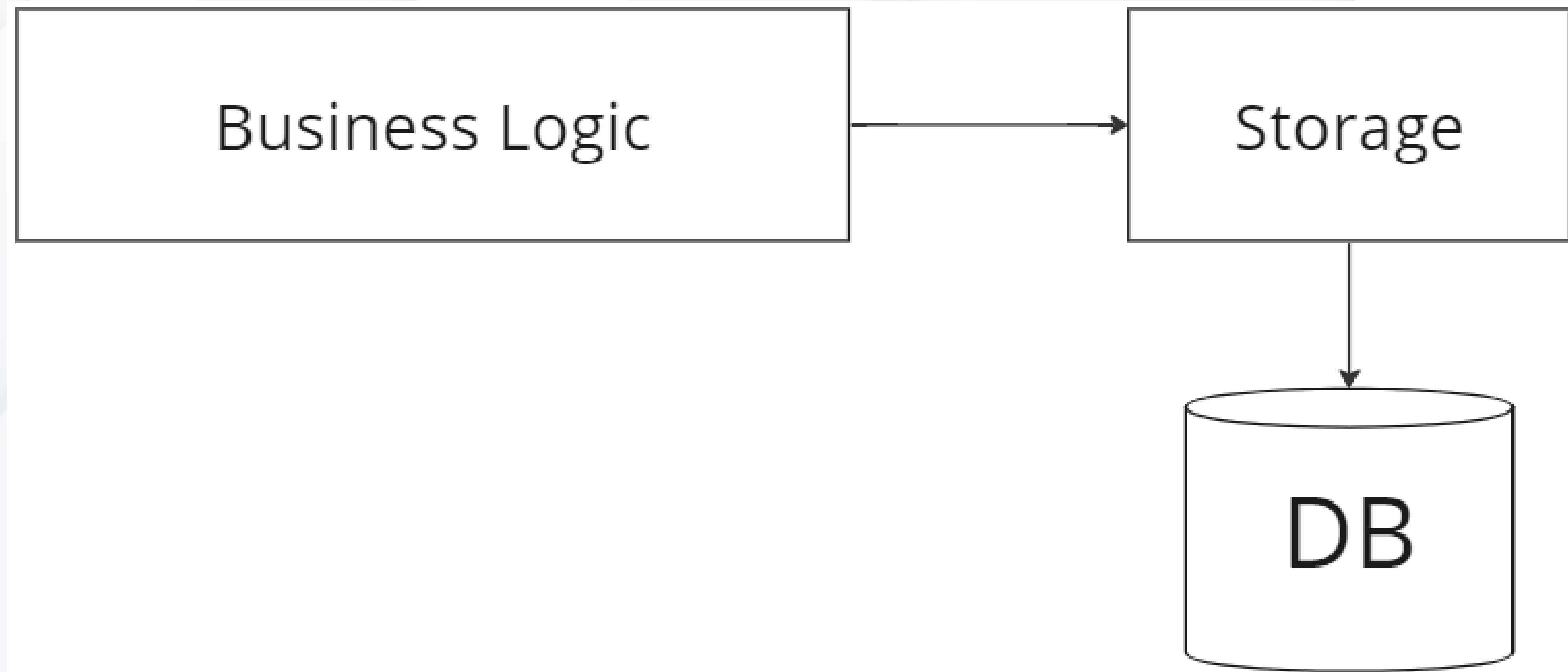
1. Неактуальность архитектуры



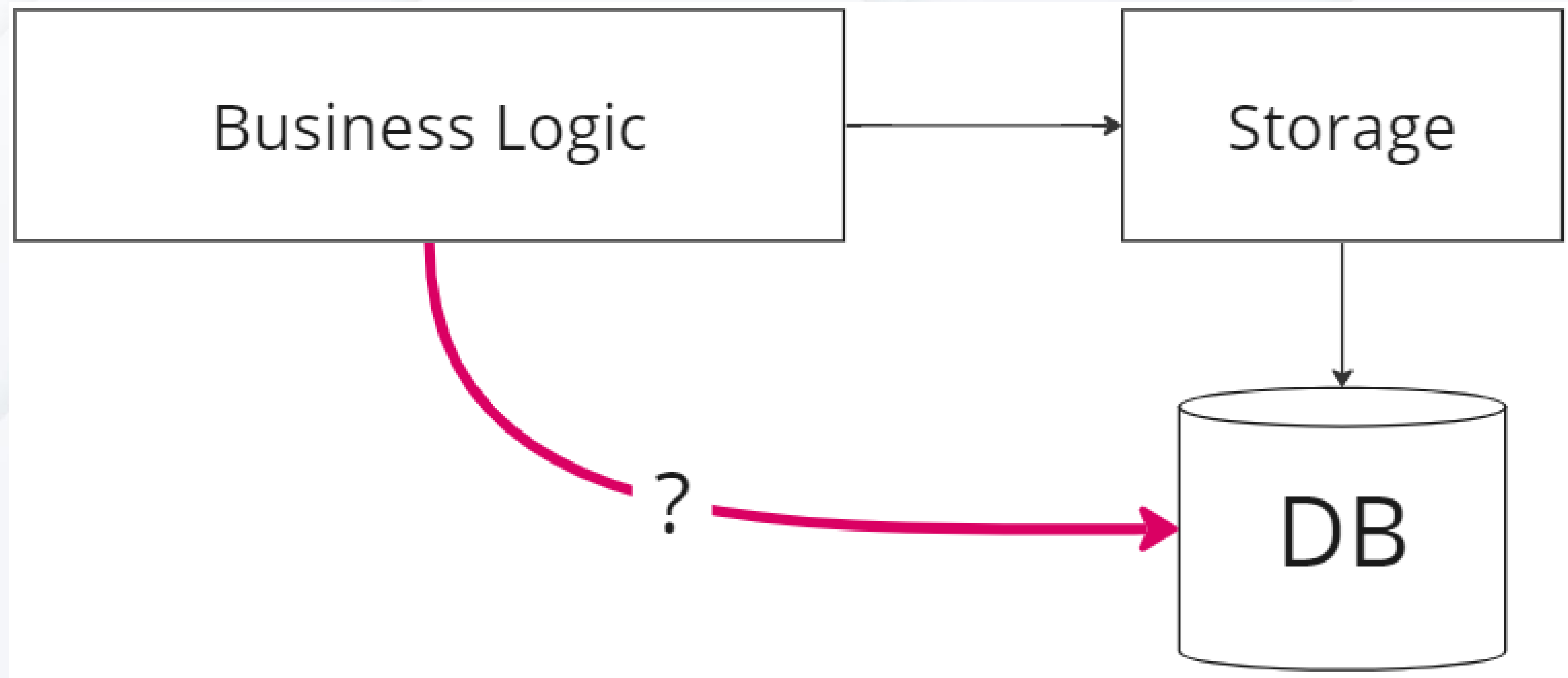
1. Неактуальность архитектуры



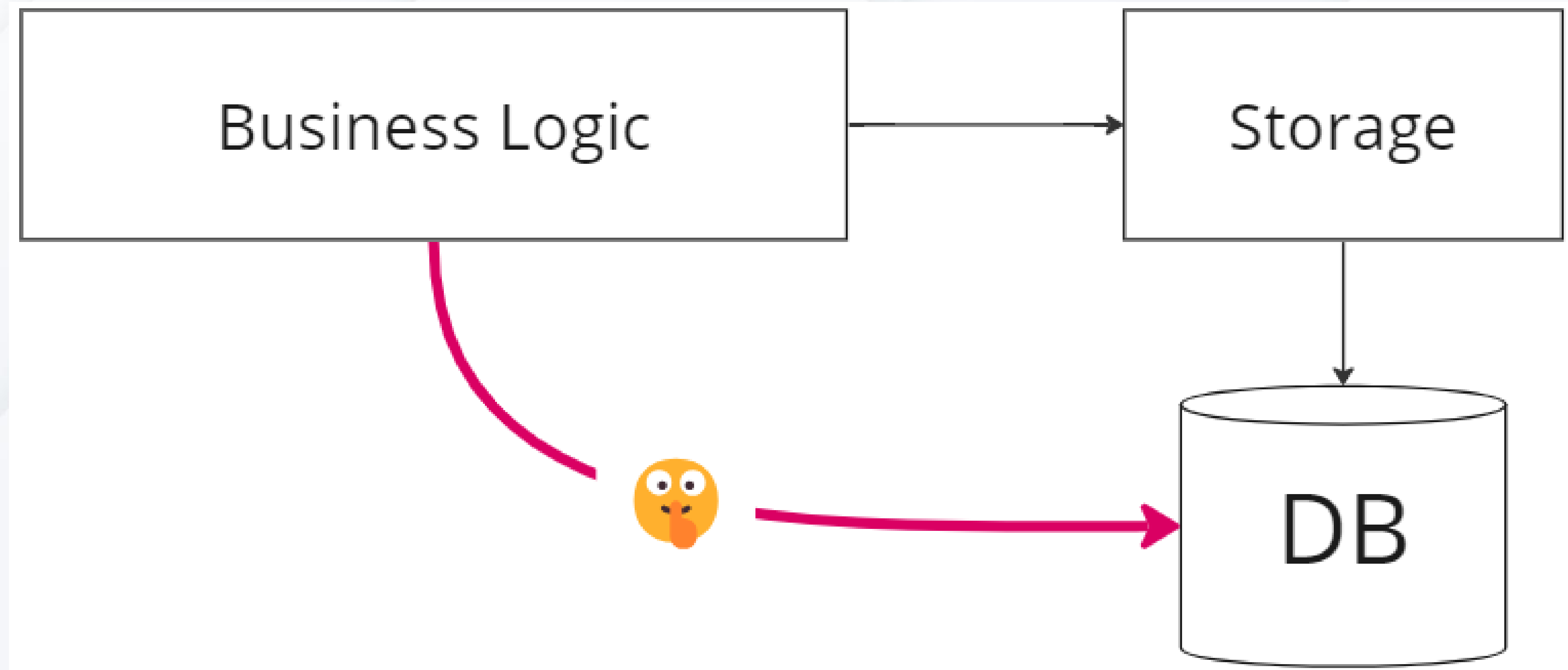
2. Декларативность



2. Декларативность



3. Отсутствие контроля



Есть проблемы с архитектурой



Архитектура теперь описывается кодом



Значит, это проблемы кода!

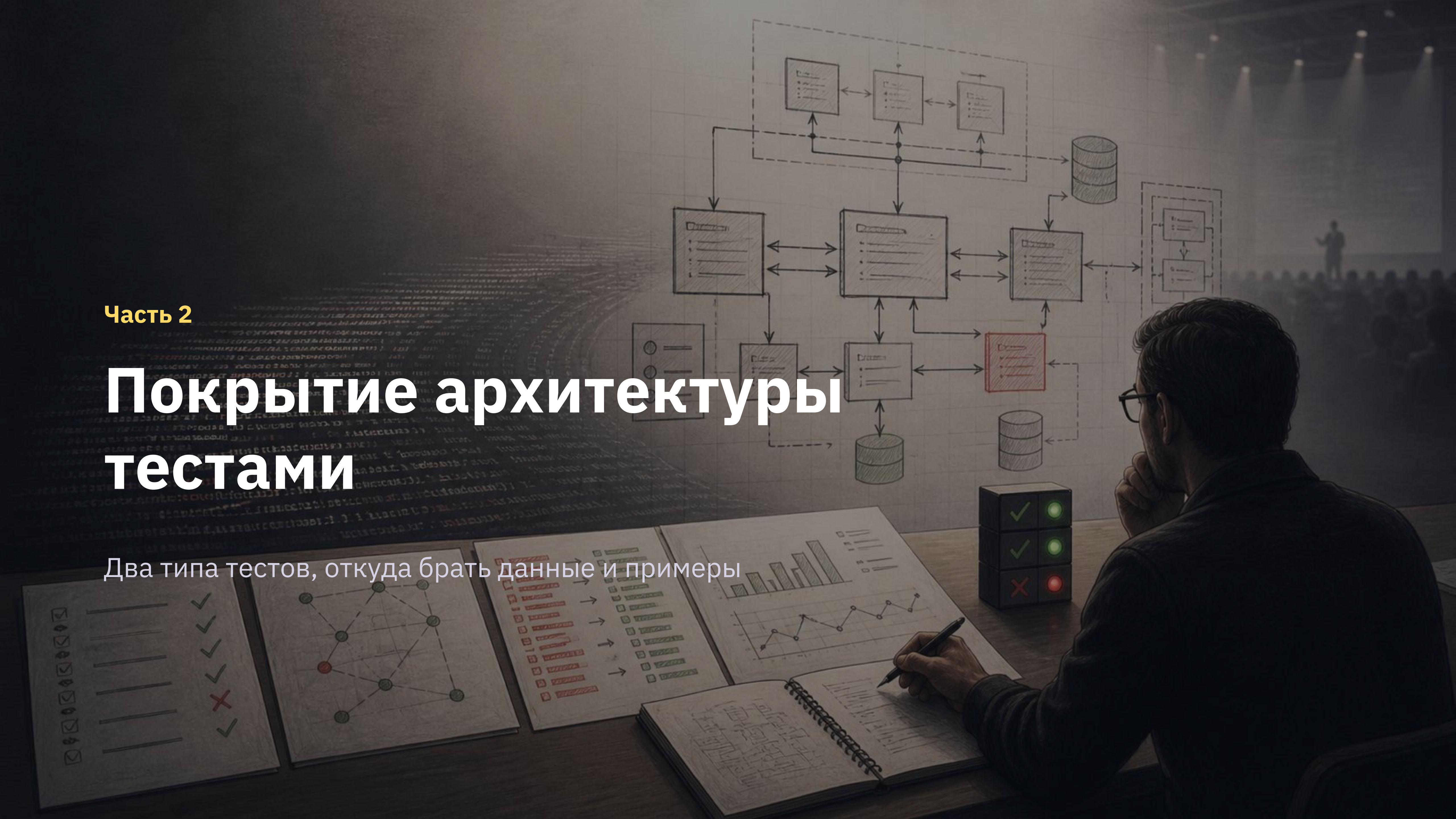
Что мы говорим проблемам в коде?

Напиши тест!

Часть 2

Покрытие архитектуры тестами

Два типа тестов, откуда брать данные и примеры



Два типа архитектурных тестов

Тип 1

Соответствует ли схема тому, что реально работает?

лечит неактуальность

Тип 2

Соблюдает ли схема принципы и договорённости команды?

лечит декларативность и отсутствие контроля

Тип 1. С чем сравнивать схему?

Зависимости можно извлечь из кода, конфигов или сетевого трафика.

У каждого источника будут свои ограничения.

Источник фактических связей

Код сервисов

+

Первоисточник поведения сервисов

—

Сложность извлечения данных
Нет инфраструктурных связей

Источник фактических связей

Service Mesh



Реальный трафик



Нет «редких» связей

Сложность симуляции на препроде

Данные только после появления трафика

Источник фактических связей

Конфиги инфраструктуры (IaC)



Заявленные связи, включая редкие
Адреса, топики и базы в одном месте
Данные не требуют трафика



Конфиг описывает деплой, но не
показывает реальные вызовы

Зависимости есть в конфиге

```
environment:  
  GOODS_REPO_BASE_URL:  
    default: https://gateway.int.com:443/goods/v1  
  TASK_REPO_BASE_URL:  
    default: http://tasks-repository:8080  
  KAFKA_TASKS_BROKERS:  
    default: msg-q8s.int.cloud:9092  
  KAFKA_TASKS_TOPIC:  
    default: stock-tasks-q8s-v1
```

Сравниваем два массива

Из схемы

```
[  
  Сервис1 { [ связь1, связь2 ] },  
  Сервис2 { [ связь1, связь2 ] },  
]
```

Из конфигов

```
[  
  Сервис1 { [ связь1, связь3 ] },  
  Сервис2 { [ связь1 ] },  
  Сервис3  
]
```

Списки разошлись — тест падает. Запускаем его вместе с остальными unit-тестами в CI.

Как выглядит такой тест

```
it("finds diff in configs and uml containers", () => {  
  const namesFromDeploy = deployConfigs.map((x) => x.name);  
  const containerNamesFromPuml = containersFromPuml  
    .filter((x) => x.type == "Container")  
    .map((x) => x.name);  
  
  expect(namesFromDeploy).toEqual(containerNamesFromPuml);  
});
```

Слева имена сервисов из конфигов деплоя, справа контейнеры из схемы.

test/architecture.test.ts — так это выглядело до v2

И как он падает

```
$ npx jest architecture
```

- Architecture › finds diff in configs and uml containers

```
expect(received).toEqual(expected)  // deep equality
```

```
- Expected   - 1
```

```
+ Received   + 1
```

```
Array [  
  "bff",  
-  "stock",  
+  "goods",
```

```
Tests: 1 failed, 4 passed
```

В конфиге указан goods, а на схеме stock. Обычный Jest-тест обнаружил расхождение.

v1

Что ещё можно проверять по конфигу?

Всё, что есть в конфиге и важно отразить на архитектурной схеме 😊

Например:

- Тип и параметры связей
- Параметры контейнеров
- Репликация и автоскейлинг
- Разделение по хостам и дата-центрам
- Метрики мониторинга
- Форматирование и конвенции
- ...

Но если мы проверяем соответствие 1 к 1 —
почему бы просто не сгенерировать
одно из другого?



Проблемы архитектурных схем

0. Их отсутствие 😊

1. ...

Автогенерация схемы из конфигов

Из конфигов можно собрать схему сервисов и их зависимостей.

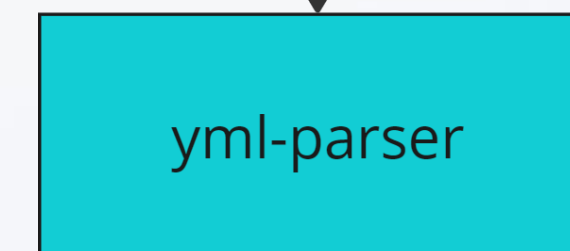
Часть данных придётся уточнить:

- как называются внешние системы
- кто отправляет сообщения, а кто их читает
- каких связей и пояснений не хватает в конфиге

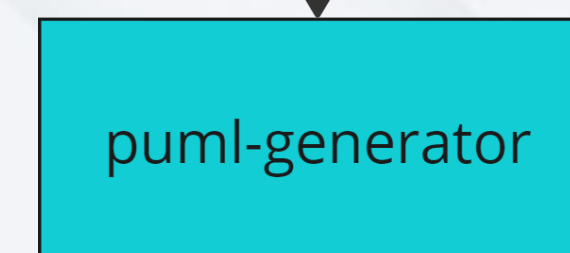
Инфраструктура



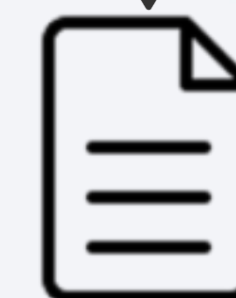
.yaml



mapper



puml-generator

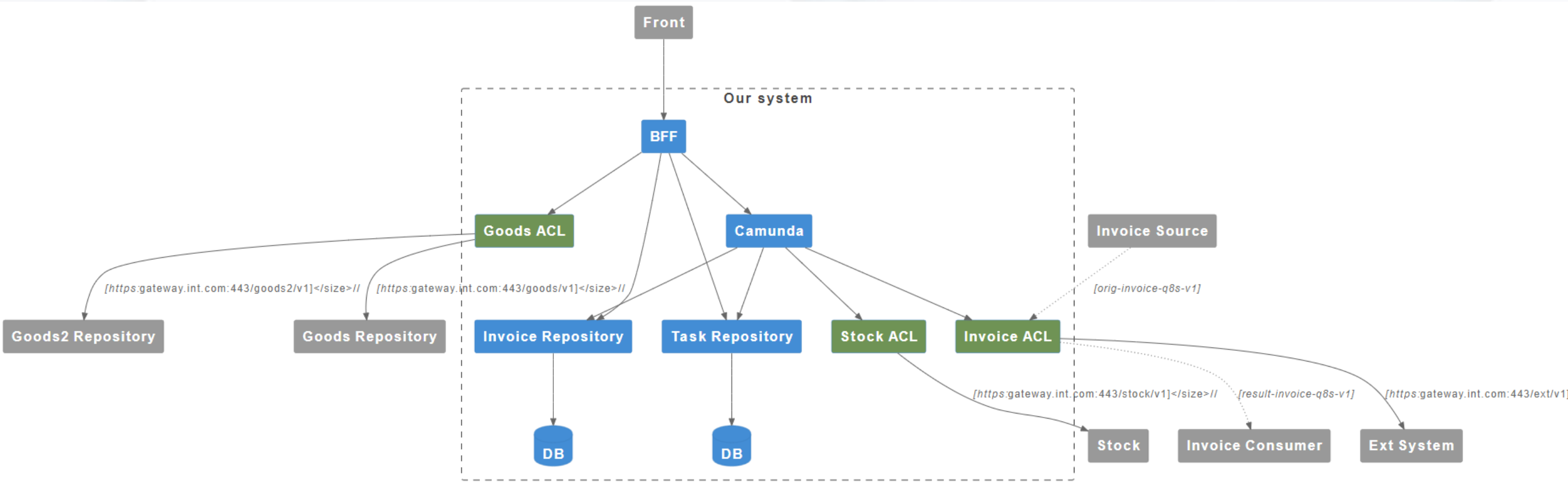


.puml

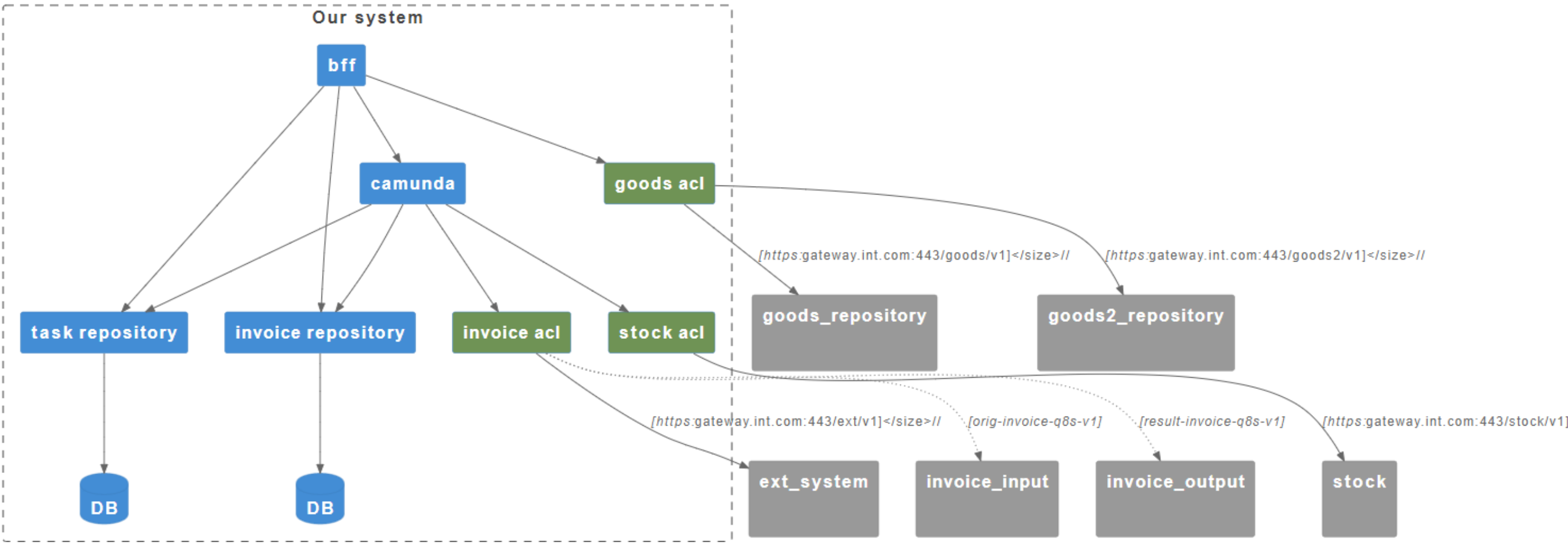
Архитектура

Результат генерации

Нарисовано вручную
PlantUML



Сгенерировано
из конфигов деплоя



И что выбрать?

Генерировать каждый раз vs сгенерировать один раз

+

Схема соответствует текущим конфигам
Не нужно вручную переносить их
изменения

—

Теряется то, чего нет в конфигах
Целевую архитектуру придётся
описывать отдельно

Можно сгенерировать основу, дополнить её вручную и дальше проверять соответствие конфигам тестом.



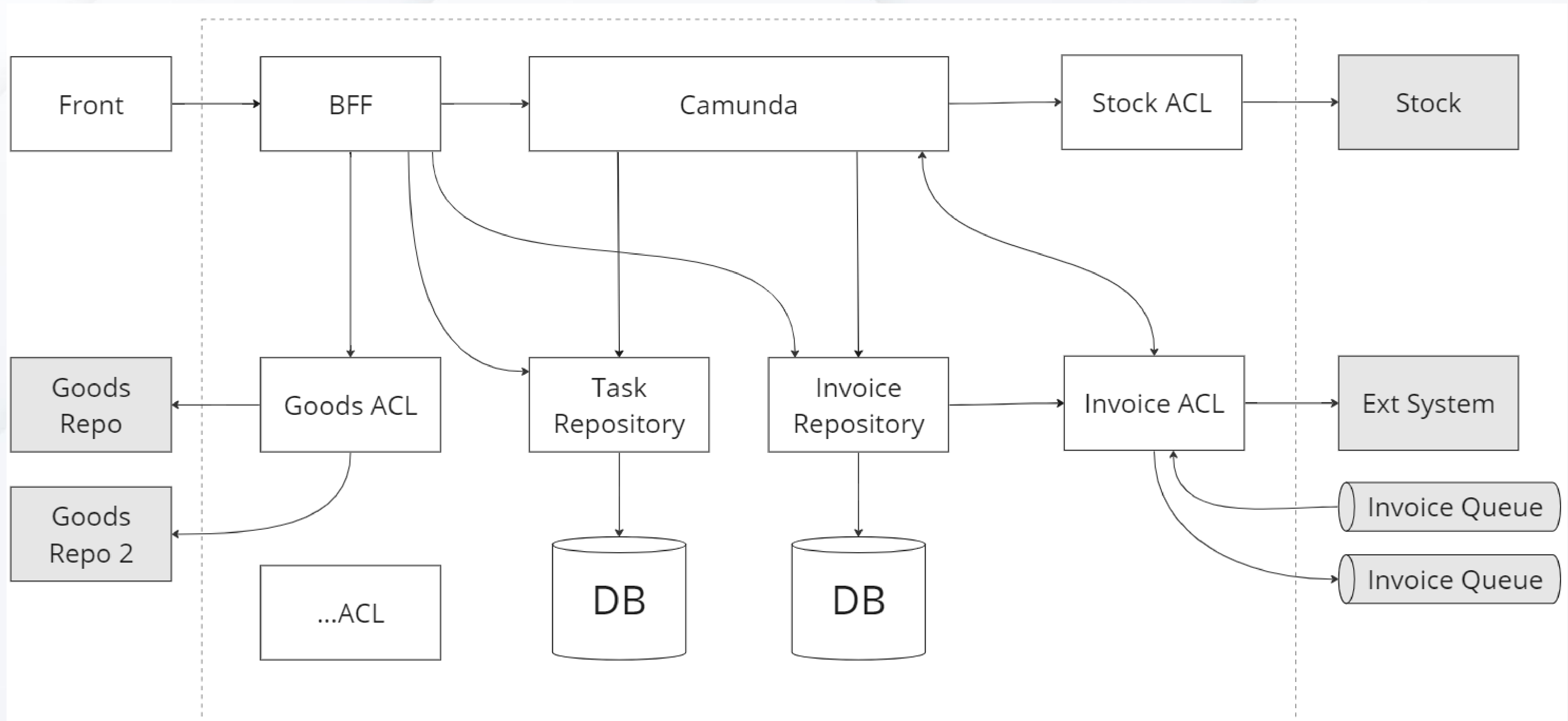
Проблемы архитектурных схем

- 0. Их отсутствие ✓
- 1. Неактуальность ✓
- 2. Декларативность
- 3. Отсутствие контроля

Тип 2. Тесты на принципы

Принцип (ADR) → конвенция → тест

На примере архитектуры микросервисов



Архитектура микросервисов в PlantUML



```
System_Ext(goods, "Goods Repo")  
System_Ext(stock, "Stock")
```

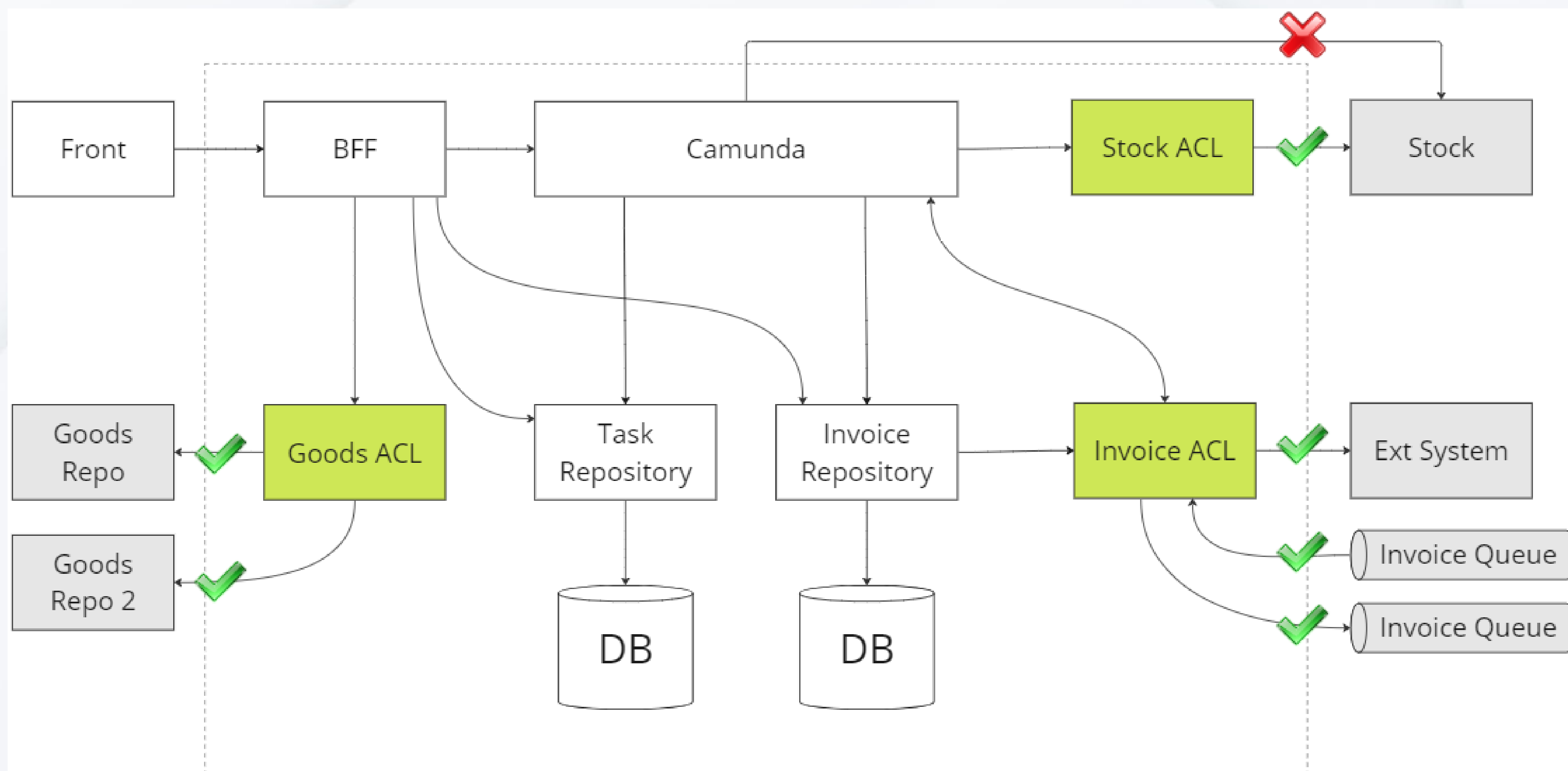
```
Container(bff, "BFF", "NestJS")  
Container(task_repository, "Task Repo", "NestJS")
```

```
Rel(bff, goods, "Get products", "API Gateway")  
Rel(task_repository, stock, "Send task", $tags="async")  
Rel(bff, task_repository, "Get task", "REST")
```

Пример 1. ACL (Anti-corruption Layer)

Принцип

За интеграцию с внешними системами отвечают отдельные сервисы-адаптеры (ACL), инкапсулирующие знания о внешней системе.



Пример 1. ACL (Anti-corruption Layer)

Принцип

За интеграцию с внешними системами отвечают отдельные сервисы-адаптеры, инкапсулирующие знания о внешней системе.

Конвенция

Помечаем такие сервисы тегом на схеме.

Как проверить

```
Container(goods_adapter, "Goods ACL", "NestJS", $tags="acl")
```

→ СВЯЗИ С ВНЕШНИМИ СИСТЕМАМИ ЕСТЬ ТОЛЬКО У СЕРВИСОВ С ЭТИМ ТЕГОМ

Как выглядит проверка ACL

```
it("external systems are called only through ACL", () => {  
  const externalCalls = model.relations  
    .filter((r) => target(r).external);  
  
  for (const call of externalCalls) {  
    expect(source(call).tags).toContain("acl");  
  }  
});
```

У сервиса, который вызывает внешнюю систему, проверяем тег acl.

В v3 для этого есть правило acl. В сообщении будут место нарушения и его описание.

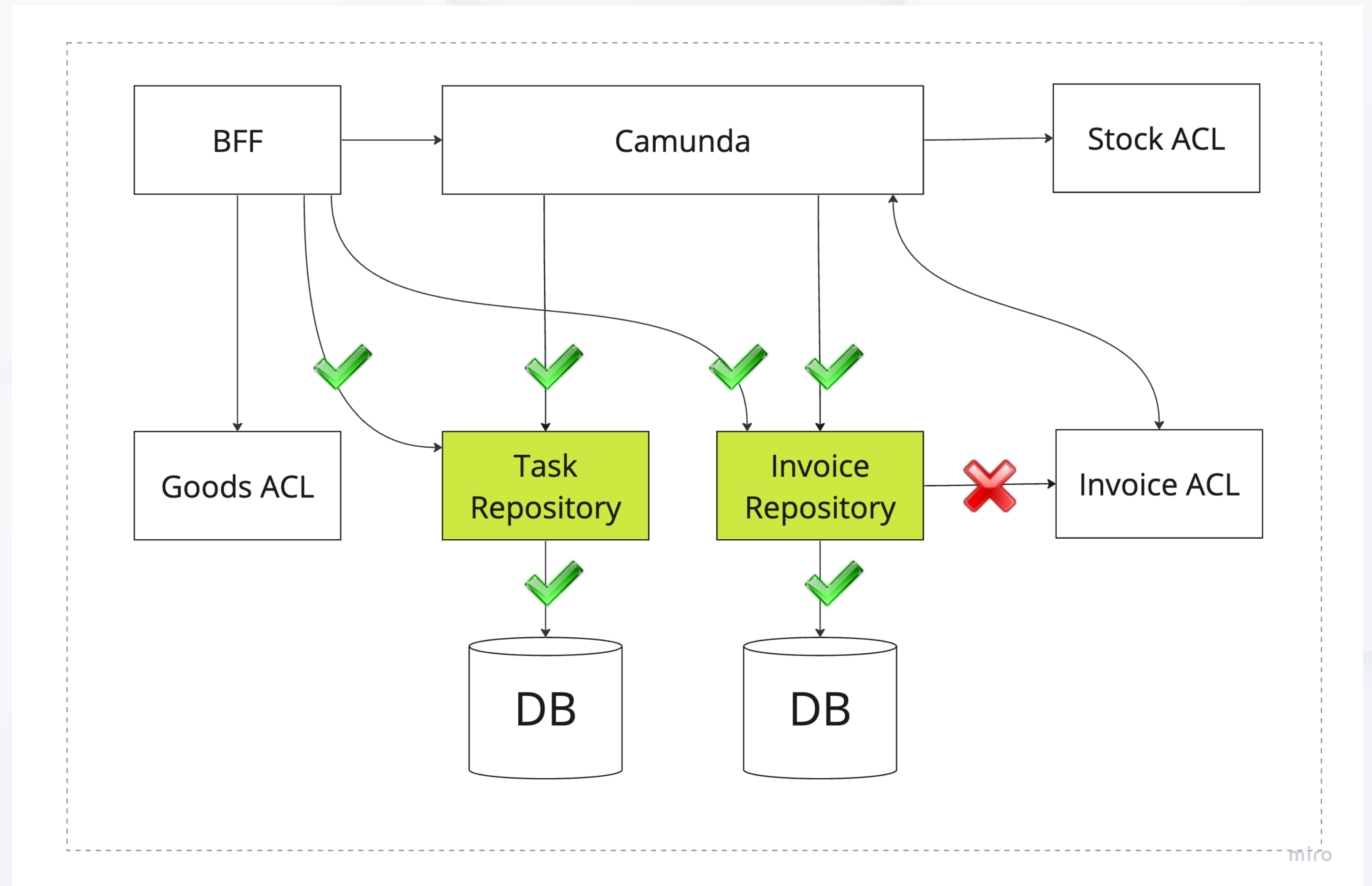
[Правило acl](#)

[ADR: Anti-corruption Layer](#)

Пример 2. Пассивные Repository

Принцип

Сервисы доступа к мастер-данным не содержат бизнес-логики и не имеют зависимостей, кроме своей БД.



Пример 2. Пассивные Repository

Принцип

Сервисы доступа к мастер-данным не содержат бизнес-логики и не имеют зависимостей, кроме своей БД.

Конвенция

Помечаем такие сервисы тегом repo.

Как проверить

```
Rel(task_repository, task_db, "")  
Rel(camunda, task_repository, "")  
Rel(task_repository, invoice_acl, "") ← нарушение
```

И проверка Repository

```
it("repositories depend only on their own database", () => {  
  for (const repo of elements(model, hasTag("repo"))) {  
    const outgoing = repo.relations.map((r) => target(r));  
  
    expect(outgoing.every(isDatabase)).toBe(true);  
  }  
});
```

У repo проверяем исходящие связи: каждая должна вести в БД.

[Правило crud](#)

[ADR: Database per CRUD-service](#)

Какие договорённости из ADR можно проверить

- Тегирование критичности сервисов
- Правила репликации и автоскейлинга по критичности
- Разнесение сервисов по хостам и дата-центрам
- Наличие DLQ у очередей
- Поставка метрик в мониторинг
- Максимальная сумма запрошенных ресурсов
- ...

Данные для таких проверок берём из конфигов и кода. Пользу решения оцениваем сами.

ADR и каталог принципов

<Название>

Контекст

<описание контекста, в рамках которого применяется решение>

Краткое описание решения и его обоснование

Схема

<архитектурная схема решения>

Решение

<описание решения>

Обоснование

<обоснование решения>

Как покрыть тестами

<описание механизма тестирования>

Примеры тестов

<ссылки на тесты>

Паттерн	Описание
Anti-corruption Layer	За интеграцию с внешними системами ответственны отдельные микросервисы-адаптеры, инкапсулирующие в себе знания о внешних системах
Пассивные CRUD-сервисы	Микросервисы, отвечающие за доступ к мастер-данным, не обладают дополнительной логикой и не имеют зависимостей помимо своей БД
API-Gateway	Внешние для периметра REST-вызовы не должны идти напрямую
Оркестратор распределённых транзакций	Запись данных идёт только через оркестратор бизнес-процессов, чтение доступно напрямую
Database per service	К БД имеет доступ только один микросервис
Stable Dependencies Principle	Зависимости направлены от менее стабильных микросервисов к более стабильным
Acyclic Dependencies Principle	Граф зависимостей не должен содержать циклов
Common Reuse Principle	Зависимость от контекста микросервисов использует внешнеконтекстные API всех микросервисов контекста

[ADRs/ADR template.md](#) — шаблон

[patterns.md](#) — каталог принципов

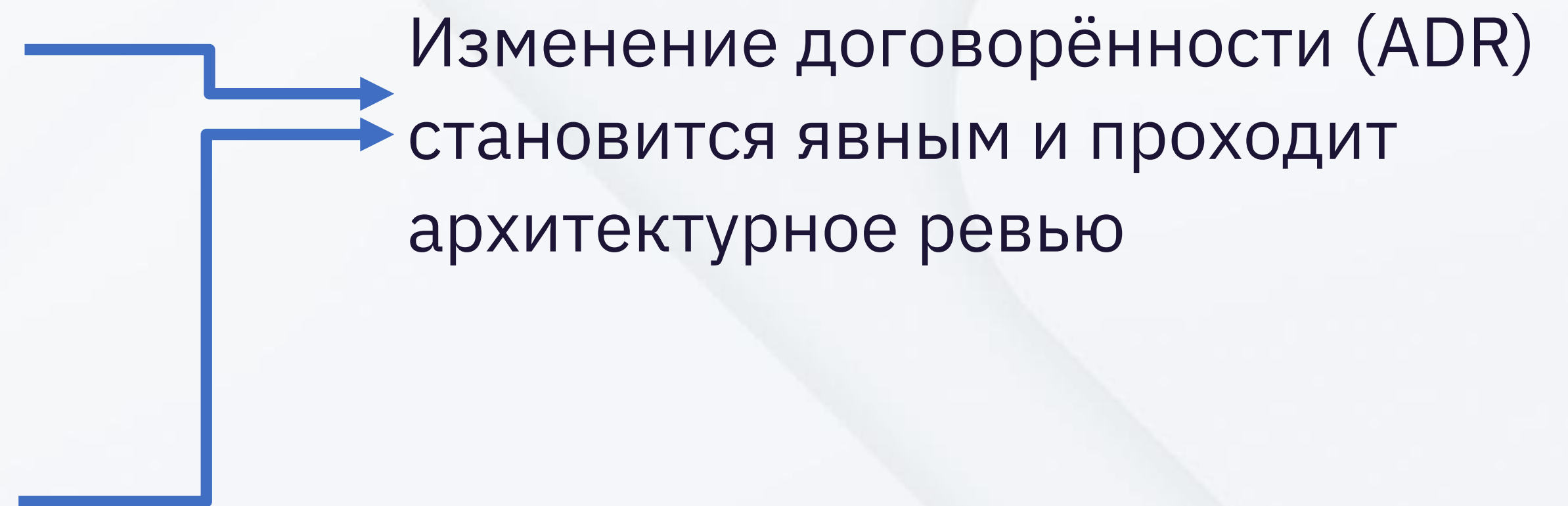


Проблемы архитектурных схем

- 0. Их отсутствие ✓
- 1. Неактуальность ✓
- 2. Декларативность ✓
- 3. Отсутствие контроля

Договорённости нарушают по незнанию или осознанно

- Упавшие тесты отсеют нарушения по незнанию
- Изменения правил (тестов) и исключений проходят ревью архитектора



Личный опыт

Сколько стоила разработка с нуля

2 чел/нед разработчика

2 чел/нед архитектора

0,5 чел/нед девопса

В продакшне с 2023 года. Это не оценка подключения v3

Что нашли и починили

Актуализировали схему

Привели к единым конвенциям

Топики Kafka с продьюсерами, но без подписчиков

Неиспользуемые зависимости в конфигах и коде

Перевели зависимости на API Gateway

Зафиксировали архдолг числом

**Нашёл проблему –
напиши тест!**



Как не наращивать архитектурный долг

4 сервиса вызывают внешние системы без ACL.

Пока исправить их не можем, но тест отключать не будем.

```
expect(servicesWithoutAcl).toHaveLength(4);
```

- 4 Известный долг. Тест проходит.
- 5 Добавили нарушение. Тест падает, исправляем код.
- 3 Исправили сервис. Тест падает, меняем в нём 4 на 3.



Aact (Architecture as Code Tools)

Три года: от примеров тестов к инструменту

Как это росло

2023

Первые доклады и первые тесты на реальных проектах. Aact v1

2024

[Статья](#) на Хабре — статья года в номинации «Бэкенд»

2026

Версии Aact: v2 весной, v3 сейчас

В v1 тесты переносили в проект вручную. В v2 появился общий CLI.
В v3 парсеры находят элементы схемы их координаты в исходнике.



Вернёмся к нашему PR

Тот же пример, что в начале: заказы обходят фасад склада

Почему заказы должны вызывать Inventory API

Зачем

Хотим менять внутренний сервис склада независимо от заказов

О чём договорились

Заказы обращаются к складу через Inventory API

Что проверяем

У заказов нет прямых вызовов Inventory Internal Service

В ADR записываем, почему выбрали такую границу и когда готовы её пересмотреть.

Модель

```
Container(orders_svc, "Orders Service", "dotnet", $tags="bc:orders")
Container(inventory_api, "Inventory API", "dotnet", $tags="bc:inventory+public")
Container(inventory_svc, "Inventory Service", "dotnet", $tags="bc:inventory")

Rel(orders_svc, inventory_api, "")
Rel(inventory_api, inventory_svc, "")
```

Начиная с v2 aact — CLI и прт-пакет

```
npx aact init  
npx aact check  
npx aact check --fix  
npx aact analyze  
npx aact generate
```

Восемь встроенных правил, метрики связанности, автофикс части нарушений и генератор Kubernetes.

Что содержит сообщение check

```
$ npx aact check
```

```
architecture.puml:11:1 error crud orders: directly accesses  
                        database orders_db  
  ↳ database: orders_db: architecture.puml:7:3
```

```
2 violations in 2 rules
```

```
1 rule has auto-fix — run with --fix
```

По ссылке `architecture.puml:11:1` открываем место нарушения.
Ниже указана база `orders_db` и её положение в исходнике.

Своё правило для ограниченного контекста склада

```
export const bcIsolationRule = defineRule({  
  name: "bcIsolation",  
  description: "Cross-context dependencies use a public contract",  
  check(model) { ... },    // цикл по связям — в примере дальше  
  // fix нет: обход контракта автоматически не исправить  
});
```

Встроенные правила проверяют общие паттерны. Для ADR своего проекта можно написать свои.

Внутри правила — обычный цикл по связям

```
check(model) {  
  for (const source of Object.values(model.elements)) {  
    // ... пропускаем чужие типы, требуем ровно один тег bc:*  
    for (const relation of source.relations) {  
      const target = model.elements[relation.to];  
      if (sourceContexts[0] === targetContexts[0]) continue;  
      if (target.tags.includes('public')) continue;  
      report(source, `crosses into "${target.name}"; use a public contract`);  
    }  
  }  
}
```

[Гайд: свои правила](#)

[Публичный пример bcIsolation.ts](#)

check находит обход контракта

```
$ npx aact check change.puml
```

```
change.puml:2:1 error bcIsolation
```

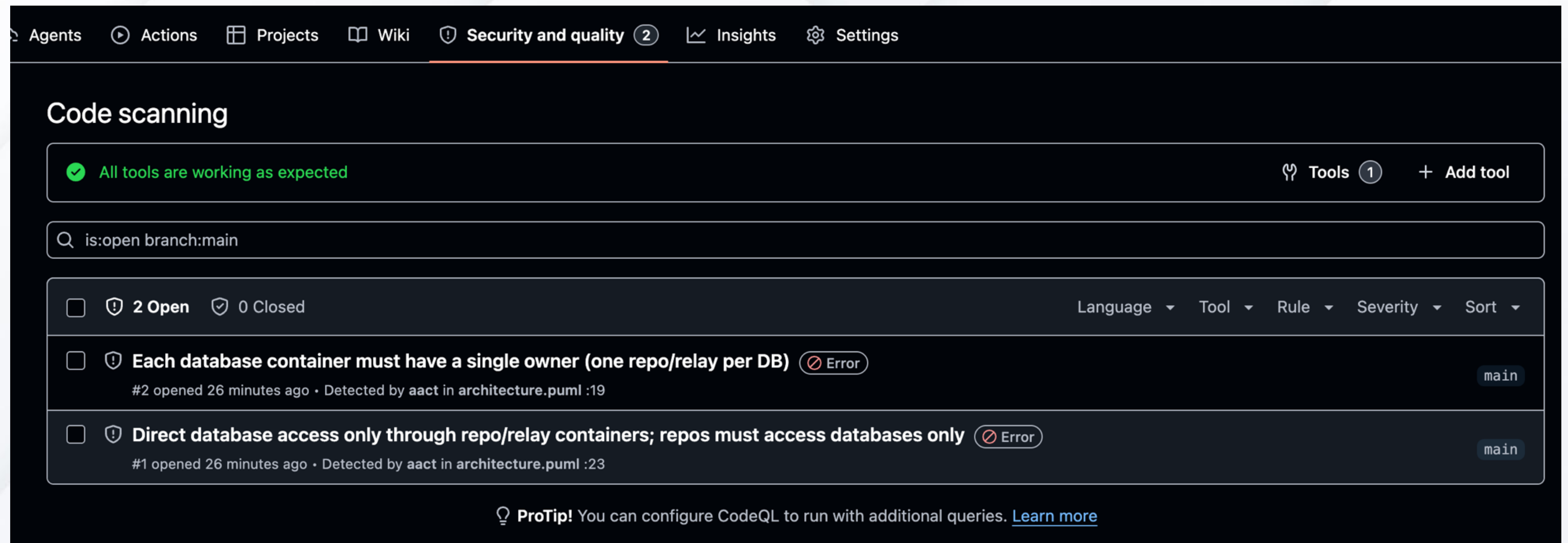
```
orders_svc: crosses into "inventory_svc"; use a public contract
```

```
X 1 violation in 1 rule
```

```
exit code 1
```

По ссылке `change.puml:2:1` переходим к элементу, у которого нашли нарушение.

Результаты проверки в GitHub Code Scanning



The screenshot shows the GitHub Code Scanning interface. At the top, there's a navigation bar with links to Agents, Actions, Projects, Wiki, Security and quality (2), Insights, and Settings. Below this, the 'Code scanning' section is active. It shows a green checkmark indicating 'All tools are working as expected'. There's a search bar with the query 'is:open branch:main'. Below the search bar, there's a summary showing '2 Open' and '0 Closed' issues. Two specific issues are listed:

- Each database container must have a single owner (one repo/relay per DB)** (Error) - #2 opened 26 minutes ago - Detected by aact in architecture.puml :19
- Direct database access only through repo/relay containers; repos must access databases only** (Error) - #1 opened 26 minutes ago - Detected by aact in architecture.puml :23

At the bottom, there's a 'ProTip!' suggesting to configure CodeQL with additional queries, with a link to 'Learn more'.

`npx aact check --sarif > aact.sarif`

Загружаем отчёт в GitHub. Для запрета merge отдельно включаем обязательную проверку в CI.

[Настройка GitHub Code Scanning](#)

rule explain: почему запрещён прямой вызов

```
$ npx aact rule explain bcIsolation
```

bcIsolation

Cross-context dependencies use a public contract

source: custom · enabled: yes · auto-fix: no

Rationale

Orders depends on the public Inventory contract. Inventory can change its internal implementation while preserving that contract. Each checked element must have exactly one bc:* tag.

Что делать автору PR?

Исправляем зависимость

Заказы снова вызывают Inventory API
Правим код и конфигурацию
Повторяем проверки

Меняем договорённость (ADR)

Обсуждаем с архитектором склада
Обновляем ADR и правило
Причину фиксируем в PR

А если обновили только схему?

В Compose добавили зависимость, которой нет в проверяемой схеме.

```
$ npx aact check target.puml
```

```
✓ No violations found.
```

```
exit code 0
```

```
# compose.yaml
```

```
orders_svc:
```

```
  depends_on: [inventory_svc]
```

```
  labels:
```

```
    aact.tags: bc:orders
```

Compose: нарушение и расхождение со схемой

```
$ npx aact check compose.yaml --format compose
compose.yaml:2:3 error bcIsolation
orders_svc: crosses into "inventory_svc"; use a public contract
```

```
$ npx aact diff target.puml compose.yaml --current-format compose
- Relation orders_svc → inventory_api
+ Relation orders_svc → inventory_svc
2 structural / 0 semantic / 2 cosmetic
```

Что считается изменением

structural

Появился или исчез элемент, граница, связь. Переезд между границами, смена типа

semantic

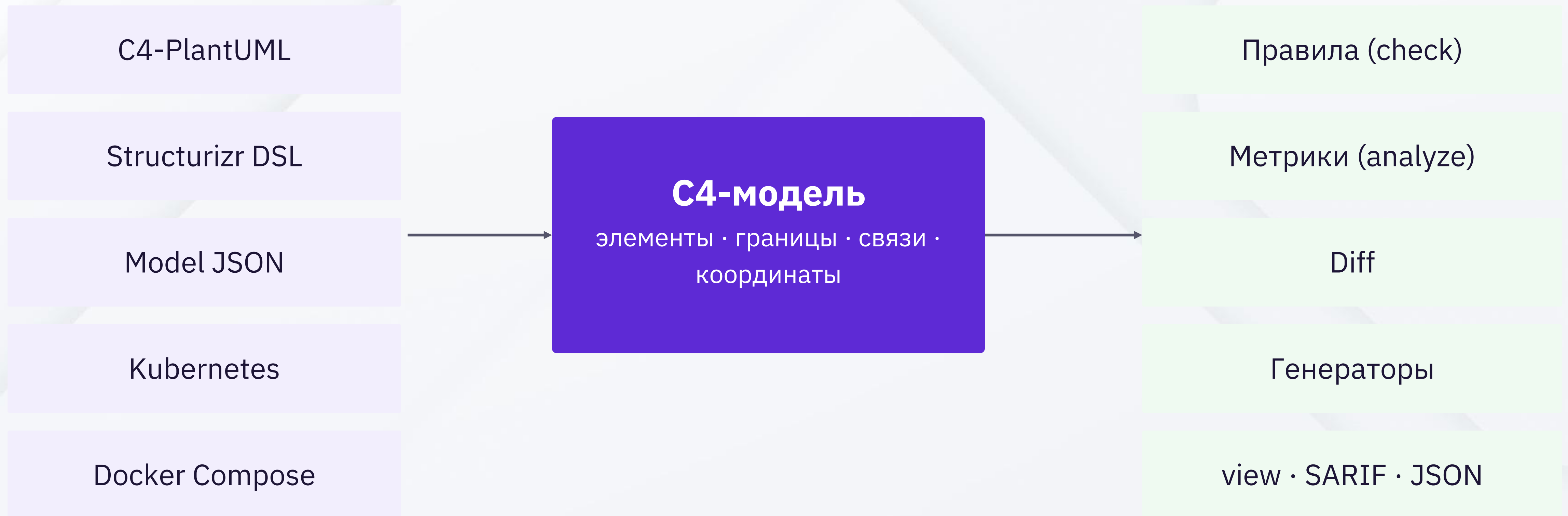
Технология, теги, свойства. Граф тот же

cosmetic

Подпись, описание, ссылка, метаданные схемы

Изменились только подписи — diff вернёт 0. С флагом --strict такие изменения тоже дадут код 1.

Ядро: одна нормализованная C4-модель



Загрузчики приводят разные форматы к общей модели. Правила проверяют её элементы и связи.

[Поддерживаемые форматы](#)

Сравнение схемы с Kubernetes-манифестами

```
$ npx aact diff target.puml ./k8s/

- Relation orders_svc → inventory_api
+ Relation orders_svc → inventory_svc
+ Element metrics_collector          (Container)
~ Element inventory_svc              technology: dotnet → go
~ Element inventory_svc              label: Inventory Service → ... (legacy)

3 structural / 1 semantic / 1 cosmetic
exit code 1
```

В манифестах заказы зависят от внутреннего сервиса склада. Появился metrics_collector, а для склада указан Go.

[Сверка архитектуры с инфраструктурой](#)
[examples/kubernetes-drift](#)

Diff группирует связанные изменения

```
$ npx aact diff main:architecture.puml architecture.puml
```

```
~ Group   Repository layer introduced between orders  
         and orders_db (confidence 0.95)
```

```
- Relation orders → orders_db (PostgreSQL)
```

```
+ Element  orders_repo           (Container)
```

```
+ Relation orders → orders_repo
```

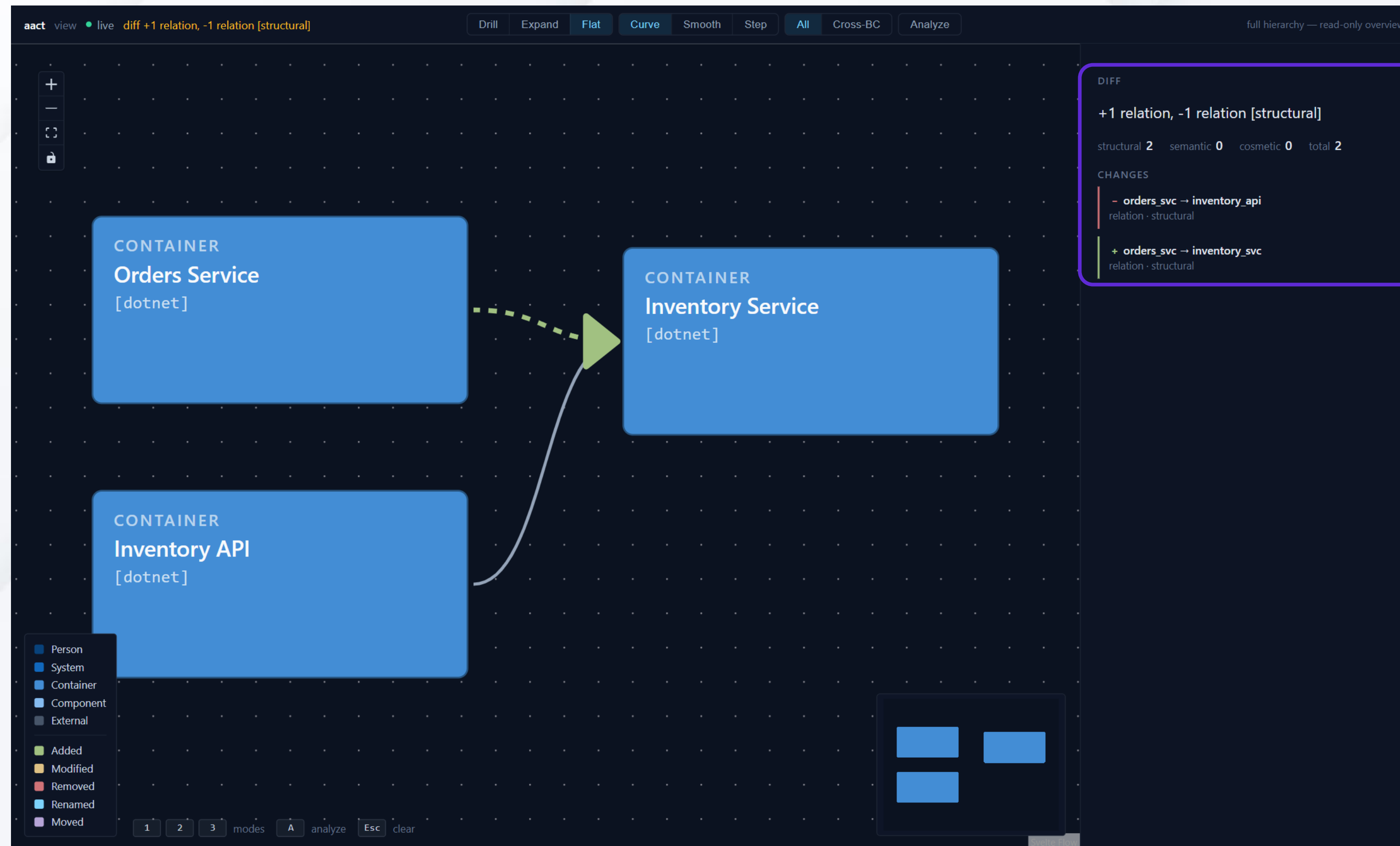
```
~ Element  orders_db   technology: PostgreSQL → MySQL
```

```
5 structural / 1 semantic / 0 cosmetic
```

Добавление Repository показано общей группой. Ниже можно посмотреть отдельные изменения.

[Гайд по review и diff](#)

Тот же diff — в workbench



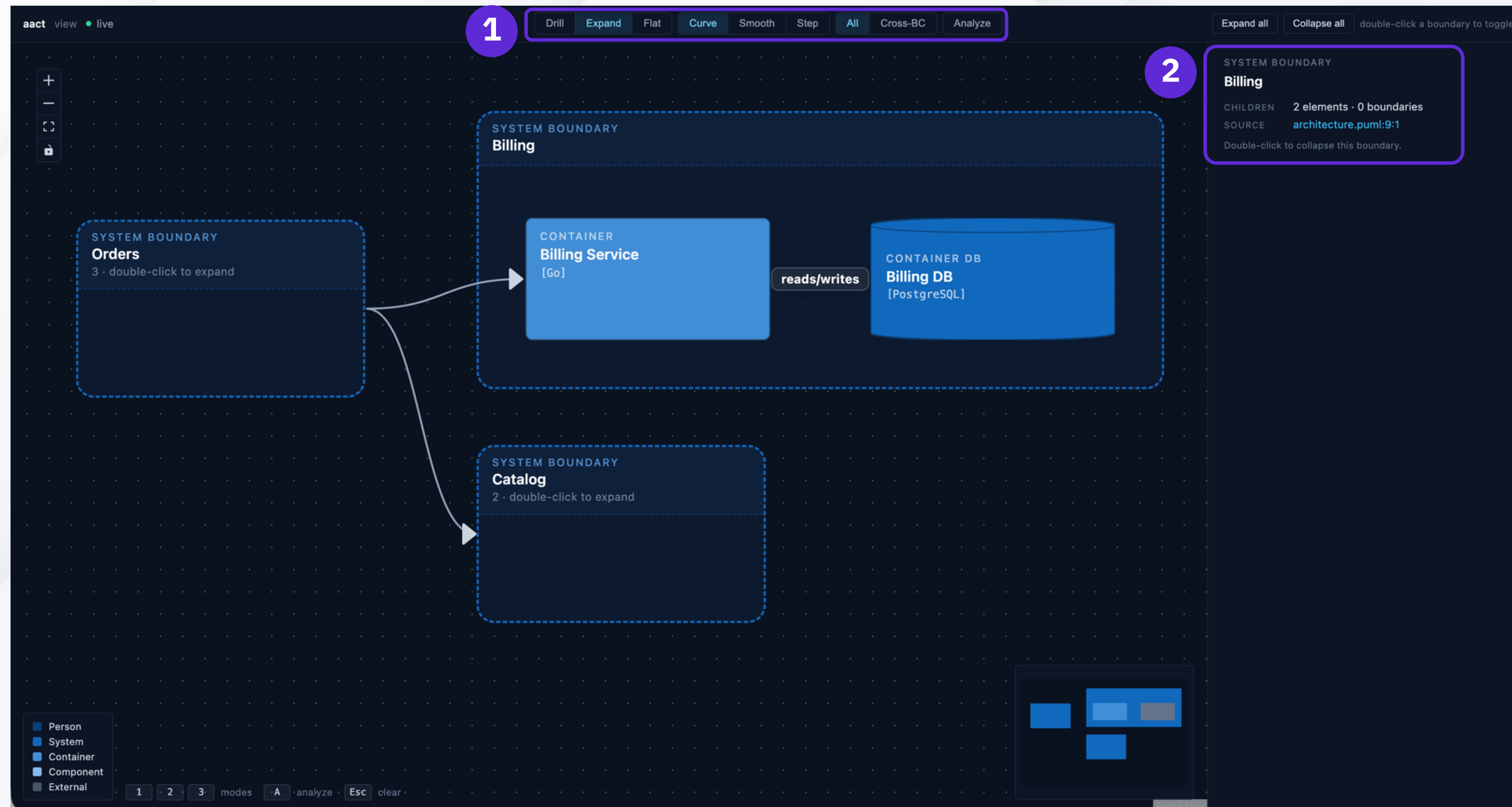
Изменения связей

- `orders_svc → inventory_api`
+ `orders_svc → inventory_svc`

Клик по элементу — файл и строка в DSL

Режимы: drill, flat, cross-BC, analyze
Граф перестраивается при сохранении файла

Просмотр модели через aact view



1 Режимы: drill, flat, cross-BC, analyze

2 Клик по элементу — файл и строка в DSL

Граф перестраивается при сохранении файла

npx aact view

Вернули вызов через Inventory API

```
$ npx aact check fixed/compose.yaml --format compose
✓ No violations found.
exit code 0

$ npx aact diff target.puml fixed/compose.yaml --current-format compose
0 structural / 0 semantic / 3 cosmetic
exit code 0
```

Заказы снова зависят от фасада склада. Check не находит нарушений, diff показывает только косметические различия.



Модульный монолит (на .NET)

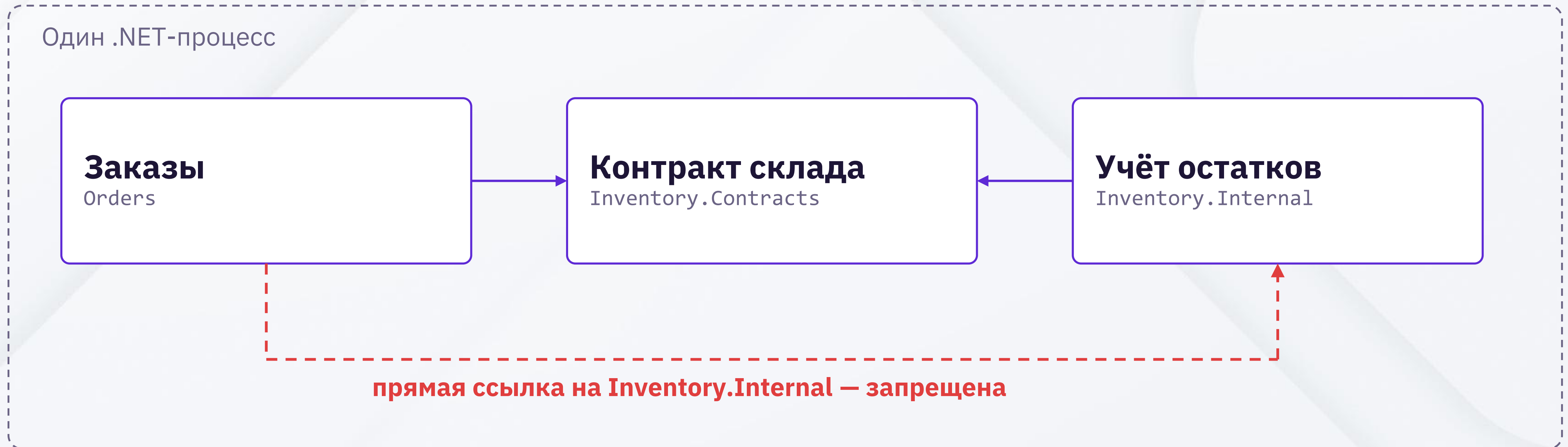
Проверяем зависимости модулей внутри одного приложения

Откуда взять зависимости модулей

Конфиги у монолита тоже есть, но межмодульные зависимости ищем в проектах, сборках и коде.

Что проверяем	Микросервисы	Модульный монолит
Состав системы	Deployment / Service	Проекты в solution
Связи	env-переменные, адреса	ProjectReference в .csproj
Выход наружу	URL внешних систем	вызовы HTTP-клиентов в коде
Границы	namespace кластера	сборка, каталог, неймспейс

Тот же PR в модульном монолите



Модуль не обязан совпадать с одной DLL

Проверка ссылок сборки через юнит-тест

```
[Fact]
public void Orders_uses_only_inventory_contract()
{
    var references = typeof(OrderService).Assembly
        .GetReferencedAssemblies()
        .Select(assembly => assembly.Name);
    Assert.DoesNotContain("Inventory.Internal", references);
}
```

ReservationTests.cs. Тест читает ссылки собранной сборки Orders и запрещает Inventory.Internal.

Orders использует Inventory.Internal

```
$ dotnet test monolith/ArchitectureTests -p:BrokenBoundary=true
```

Не пройден ReservationTests.Orders_uses_only_inventory_contract

Assert.DoesNotContain() Failure: Item found in collection

Collection: ["System.Runtime", "Inventory.Contracts", "Inventory.Internal"]

Found: "Inventory.Internal"

Не пройден!: не пройдено 1, пройдено 2, пропущено 0, всего 3

✗ В ссылках Orders нашли Inventory.Internal — архитектурный тест упал

Проверка графа сборок через AACT

```
$ dotnet run --project monolith/GraphExport -- <bin> monolith-bad.aact.json
Exported 3 component assemblies

$ npx aact check monolith-bad.aact.json
error bcIsolation orders: crosses into "inventory_internal"; use a public contract

$ npx aact diff monolith-target.puml monolith-bad.aact.json
+ Relation orders → inventory_internal (AssemblyReference)
```

Экспортируем ссылки сборок в Model JSON и запускаем bcIsolation. Оно находит запрещённую зависимость Orders.

[Пример такого экспортёра](#)

[Пример .NET-монолита](#)

[Архитектурные тесты ModularMonolith](#)

После исправления проходят все проверки

```
$ dotnet test monolith/ArchitectureTests -p:BrokenBoundary=false
```

```
Пройден!: не пройдено 0, пройдено 3, пропущено 0, всего 3
```

```
$ npx aact check monolith-good.aact.json
```

```
✓ No violations found.
```

```
$ npx aact diff monolith-target.puml monolith-good.aact.json
```

```
no changes · No structural changes.
```

OrderService снова использует контракт склада. Сценарий резервирования по-прежнему работает, запрещённой ссылки больше нет.

Два источника графа .NET

ProjectReference

Читаем .csproj, компилировать не нужно
Видно, что проект объявил зависимость
Используется ли она в коде — не видно

[tools/csproj-to-aact.mjs](#)

AssemblyReference

Читаем собранные сборки
Видно, что осталось после компиляции
Неиспользуемую ссылку компилятор обычно выбрасывает

[tools/assembly-to-aact](#)

Оба адаптера есть в `examples/dotnet-monolith` и возвращают Model JSON.

Динамические зависимости через `reflection` в этот граф не попадут.

Где два источника расходятся

```
$ npx aact diff as-built.aact.json compiled.aact.json

- Relation orders → inventory_contracts (ProjectReference)
~ Relation orders → inventory_internal
  technology: ProjectReference → AssemblyReference

1 structural / 2 semantic / 0 cosmetic
```

В .csproj есть ссылка на Inventory.Contracts, но код Orders не использует её типы. В собранной DLL этой ссылки нет.

Пример: [examples/dotnet-monolith](https://github.com/byndyusoft/examples/blob/main/dotnet-monolith/Inventory.Contracts/Inventory.Contracts.csproj)

Что нашли на живом монолите

- Слой бизнес-логики тянет базовый контроллер
- «Общий» модуль зависит от двух фичевых
- Четыре интеграции идут мимо адаптеров
- Внешняя система заявлена в конфиге, а вызовов в коде нет
- Циклы между компонентами внутри хоста

Монолит на .NET 8: 8 сборок, 61 компонент.

Архитектурный тест в тест-проекте .NET

```
[Fact(DisplayName = "Ссылки идут только вниз по слоям")]
public void References_should_point_downwards()
{
    var graph = ProjectGraph.Load("backend");    // читает .csproj

    var violations = graph.References
        .Where(r => !PointsDownwards(r))
        .Where(r => !KnownDebt.Contains(r));    // исключения с причиной

    Assert.Empty(violations);
}
```

Добавляем тест в существующий проект. ProjectGraph и список исключений реализуем у себя.

«Но у нас .NET, а aact — npm-пакет»

ААСТ можно запускать отдельным шагом CI. Для этого нужен Node.js, переписывать приложение на TypeScript не требуется.

Если проверку написали на .NET, запускаем её через dotnet test.

В обоих случаях нужно получить зависимости системы: например, из .NET-проектов или конфигов микросервисов.

ИИ-агент отлично работает с любыми CLI

Как использовать результаты ААСТ

Разработчик

терминал, клик в редактор

CI / CD

коды возврата, SARIF

ИИ-агент

json + отдельный skill

Ваш инструмент

движок как библиотека

[Контракт CLI для агента](#)
[LibraryAPI](#)



Как начать

Пять шагов

1. Взять один комментарий, который вы чаще всего пишете на ревью
2. Записать договорённость в ADR и написать проверку
3. Зафиксировать существующие нарушения в тесте или списке исключений
4. Сначала проверить срабатывания, включить предупреждения, затем сделать проверку обязательной для merge
5. Добавить сверку схемы с конфигами и ревью значимых изменений

[Getting started](#)

Какие проверки запускать в CI

```
# модель и договорённости
npx aact check target.puml
# соответствие независимому источнику
npx aact check compose.yaml --format compose
npx aact diff target.puml compose.yaml --current-format compose
# замечания в ревью
npx aact check --sarif > aact.sarif
# код .NET
dotnet test ArchitectureTests
```

ААСТ возвращает 0 при успехе, 1 при нарушениях или значимых расхождениях, 2 при ошибке запуска.

Загрузку SARIF и обязательный статус настраиваем отдельно.

Как работаем с архдолгом – как временно оставляем нарушения

Что оставили

Записываем конкретную зависимость и причину исключения

Кто отвечает

Договариваемся, кто и когда вернётся к исправлению

После исправления

Убираем зависимость из списка исключений

Число может остаться прежним, если одно нарушение исправили, а другое добавили. Поэтому полезно хранить и сам список.

Что останется сделать команде

- Дополнить сгенерированные Kubernetes-манифесты настройками приложения и эксплуатации
- Проверить поведение под нагрузкой: сравнения схемы с конфигами для этого недостаточно
- Описать правила своего проекта, которых нет среди встроенных
- Разобрать найденные нарушения: автоисправление поддерживают только некоторые правила и форматы

Если aast check прошёл

Что проверили

В доступной модели нет нарушений включённых правил

Что ещё выяснить

Все ли связи попали в модель и полезно ли само решение

После правки DSL

Изменения кода приложения проверяем отдельно

Можно автоматизировать и плохую договорённость. Помогло ли решение, проверяем по метрикам результата: бизнес-метрикам, времени поставки.

Теперь можно ответить на исходный PR

```
- Rel(orders_svc, inventory_api, "")  
+ Rel(orders_svc, inventory_svc, "")
```

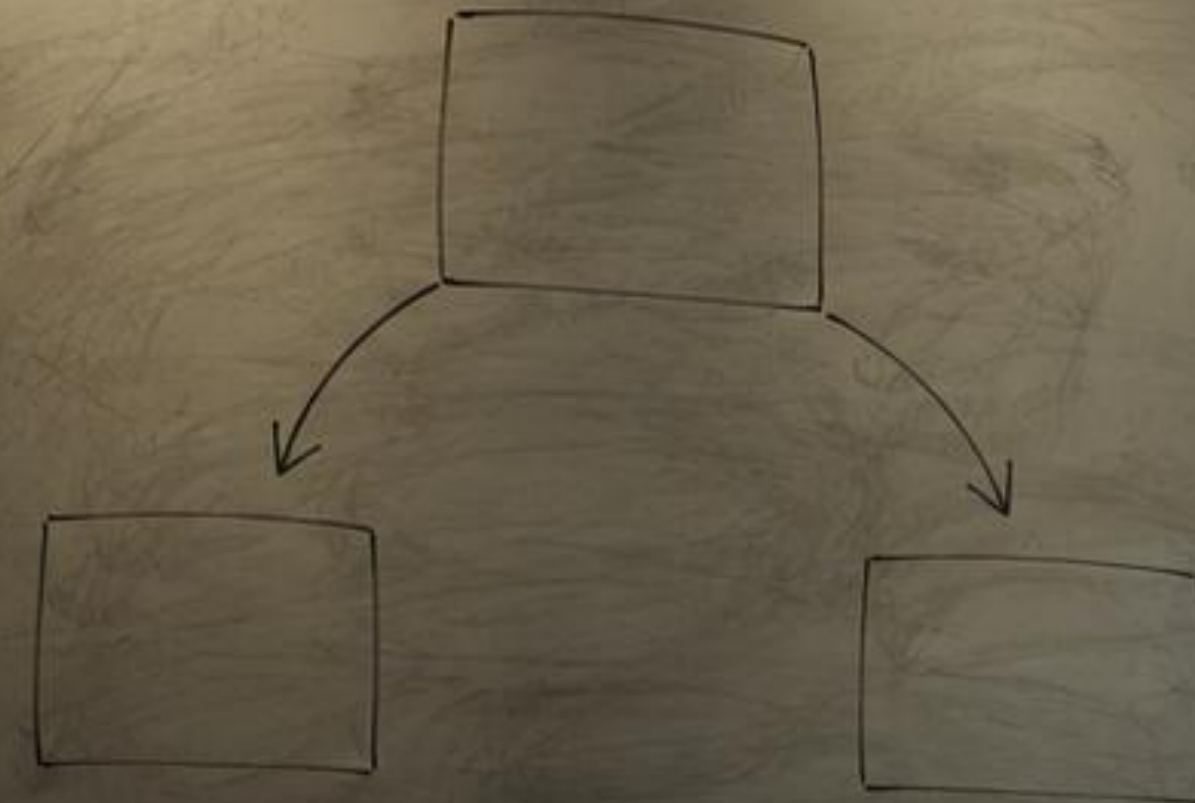
- ✓ Записали, почему складу нужна независимость от заказов
- ✓ Проверки обнаружили обход публичного интерфейса
- ✓ Автор PR исправляет вызов или согласует изменение ADR

Проверка находит нарушение. Решение о том, менять код или договорённость, принимает команда.

Кто будет следить за архитектурой в каждом PR?

Пусть это проверяет тест

Обоснование сохраняем в ADR.
Тест запускаем в каждом PR.





Итоги

- Архитектурную схему можно проверять вместе с кодом
- Одни тесты сверяют её с реализацией, другие проверяют договорённости
- Нарушение можно обнаружить в PR, до merge

Повторить на своём проекте

github.com/Byndyusoft/aact

examples/custom-rules/	своё правило
examples/kubernetes-drift/	сверка с манифестами
ModularMonolith/	.NET-монолит и тесты
ADRs/ patterns.md	решения и каталог
docs/guides/	гайды по командам
examples/dotnet-monolith/	два источника графа .NET

npm: aact 3.0.1 · @aact/view 3.0.1



github.com/Byndyusoft/aact

[docs/guides](#)

Присоединяйтесь

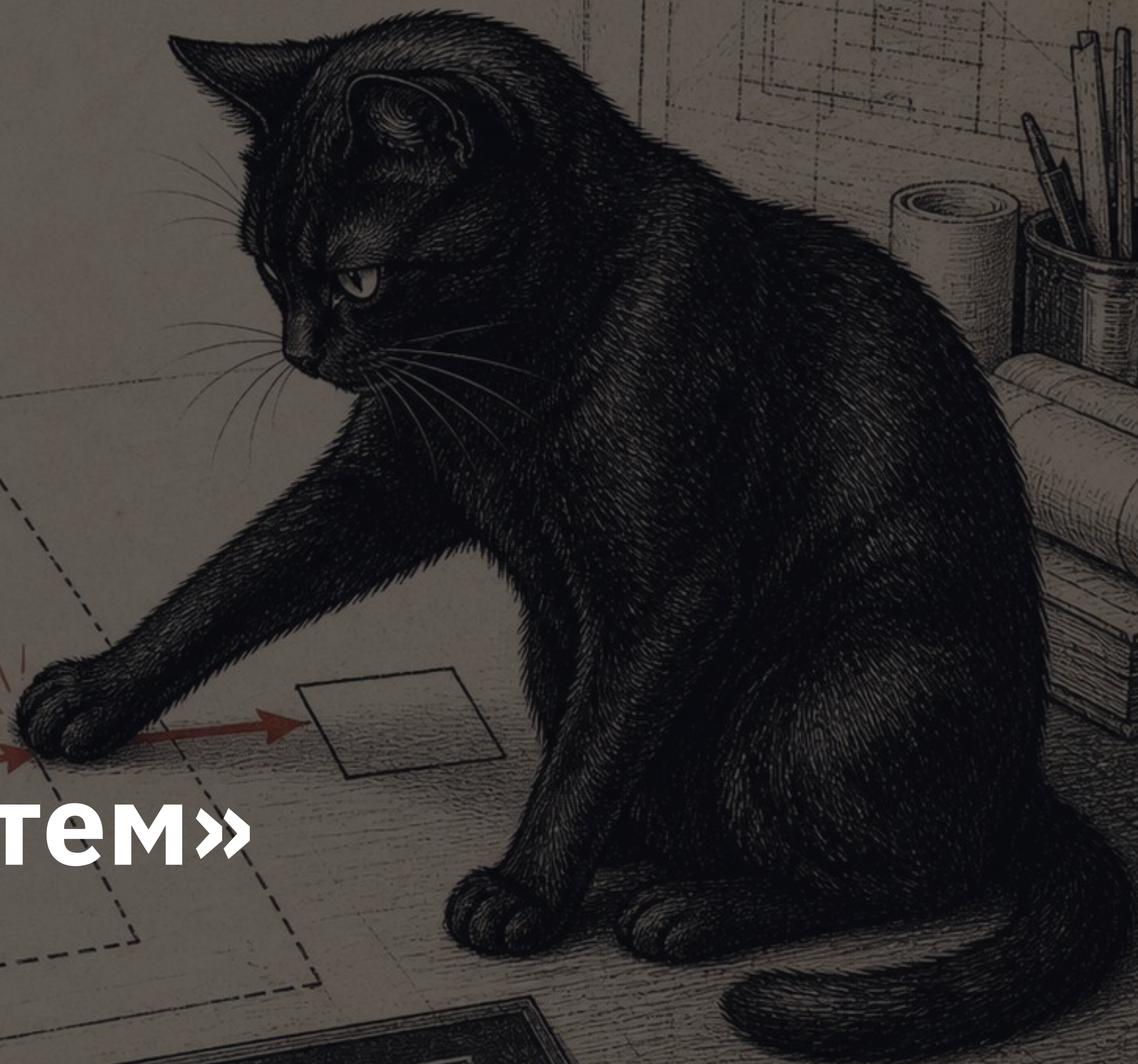
Issues и PR приветствуются!

Если у вас есть полезная проверка архитектуры, её можно добавить в ААСТ.



«Архитектура распределённых систем»

t.me/rsa_enc



Спасибо!

Руслан Сафин · технический директор Бындюсофт

github.com/Byndyusoft/aact

