Автоматический выбор минимального количества регрессионных тестов для релиза

Алексей Романов, руководитель группы Test Impact Analysis

Обо мне

Алексей Романов

@lexcorp

- Руководитель группы Test Impact Analysis в Озон Банке
- Разрабатываем системы сбора и анализа метрик тестирования
- Декан потока QA (Go) Route 256

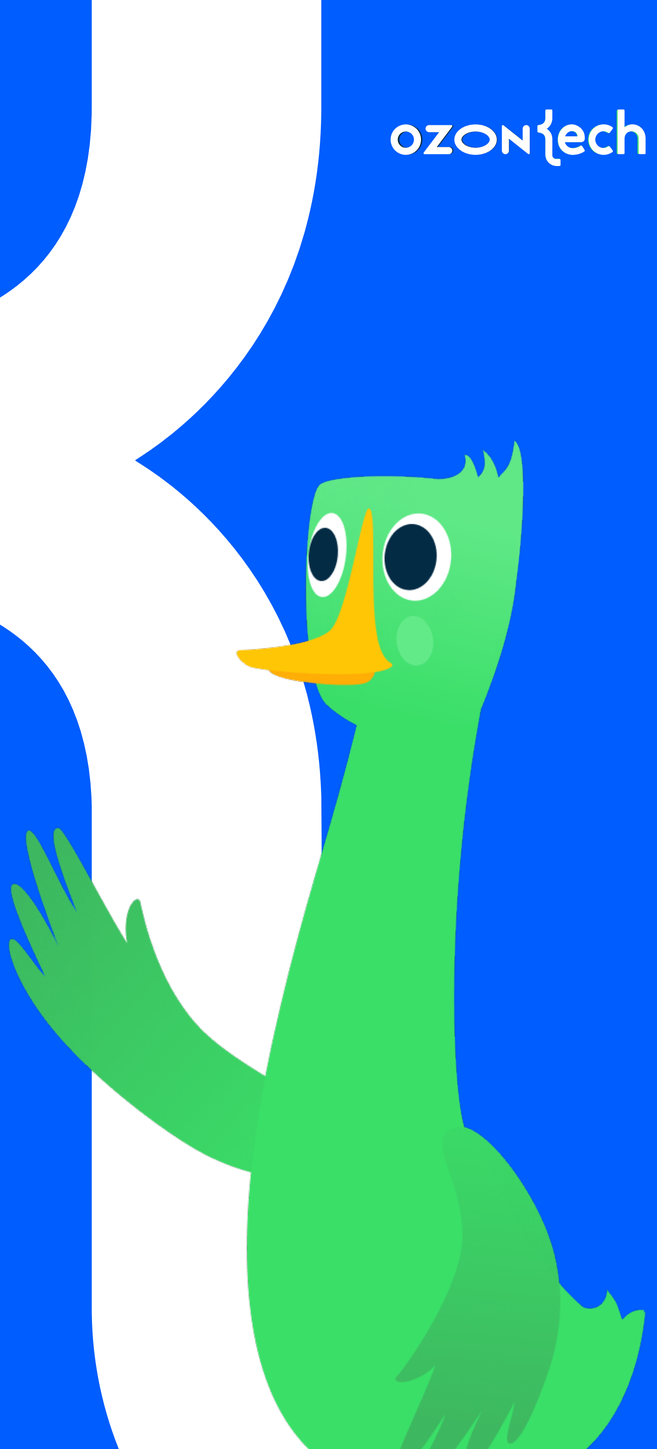
Для кого доклад

OZON tech

1. У кого микросервисы
2. Долгий регресс e2e тестов
3. Много релизов
4. Есть ресурсы на сложные штуки

План выступления

1. Проблематика
2. Трассировка
3. Минимизация
4. Сравнение
5. Статистика
6. Выводы




ОКЕЙ ЛЕТС ГОУ

1

Проблематика

Рано или поздно начинается…

Ну растет и растет, что бубнеть то

OZON tech

Когда рост количества тестов может быть проблемой

A. Хотим катиться быстро	< 20 мин на регресс
B. Хотим катиться часто	700+ релизов в неделю
C. Хотим не упираться в железо	50+ прогонов в час

Нужно минимизировать

регресс

Как бороться с большим количеством тестов

OZON tech

1. Уменьшить число тестов

Тест-дизайн

2. Уменьшить частоту запуска тестов

Ручной запуск

3. Уменьшить число запускаемых тестов

Разметка тестов

БЫЛО

Как можно минимизировать регресс

OZON tech

1. Уменьшить число тестов — Тест-дизайн
2. Уменьшить частоту запуска тестов — Ручной запуск
3. Уменьшить число запускаемых тестов — Разметка тестов — Impact Analysis

Test Impact Analysis с двух сторон

OZON tech

Анализ измененного кода

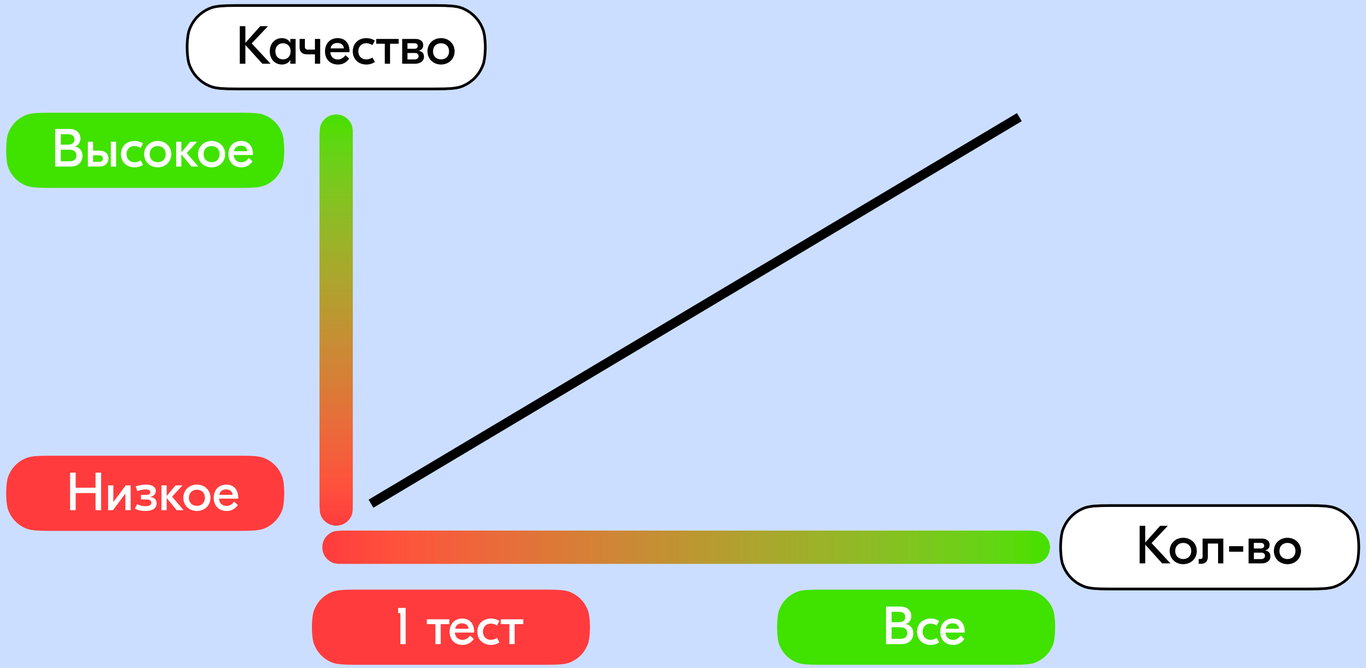
Анализ влияния тестов на качество продукта

В этом докладе мы сфокусируем на том, как уменьшить количество регрессионных тестов, зная как тесты работают

1.1

Зависимость обеспечиваемого качества от количества тестов

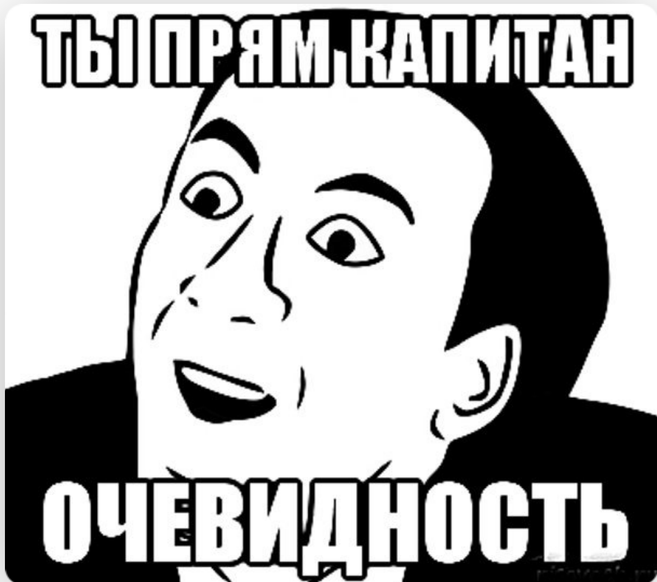
На первый взгляд



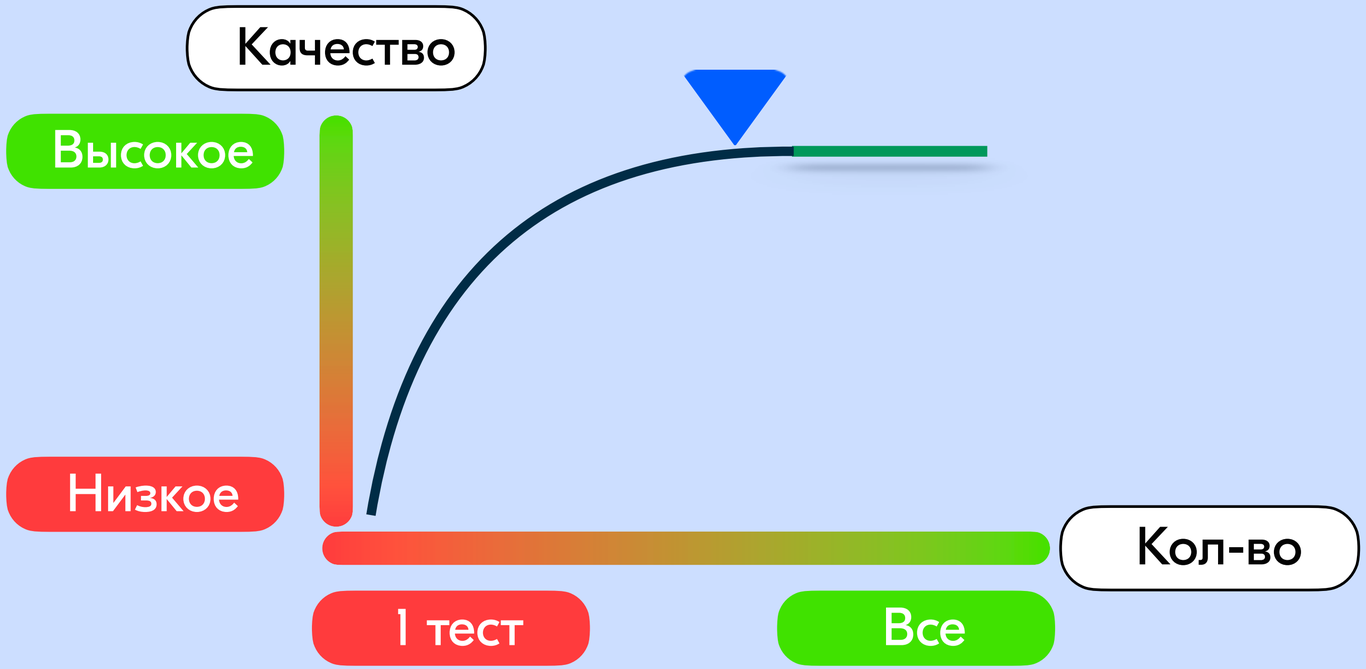
Зависимость качества тестирования от количества тестов

Чем выше качество, тем больше тестов

Чем меньше тестов, тем ниже качество



Но как на самом деле



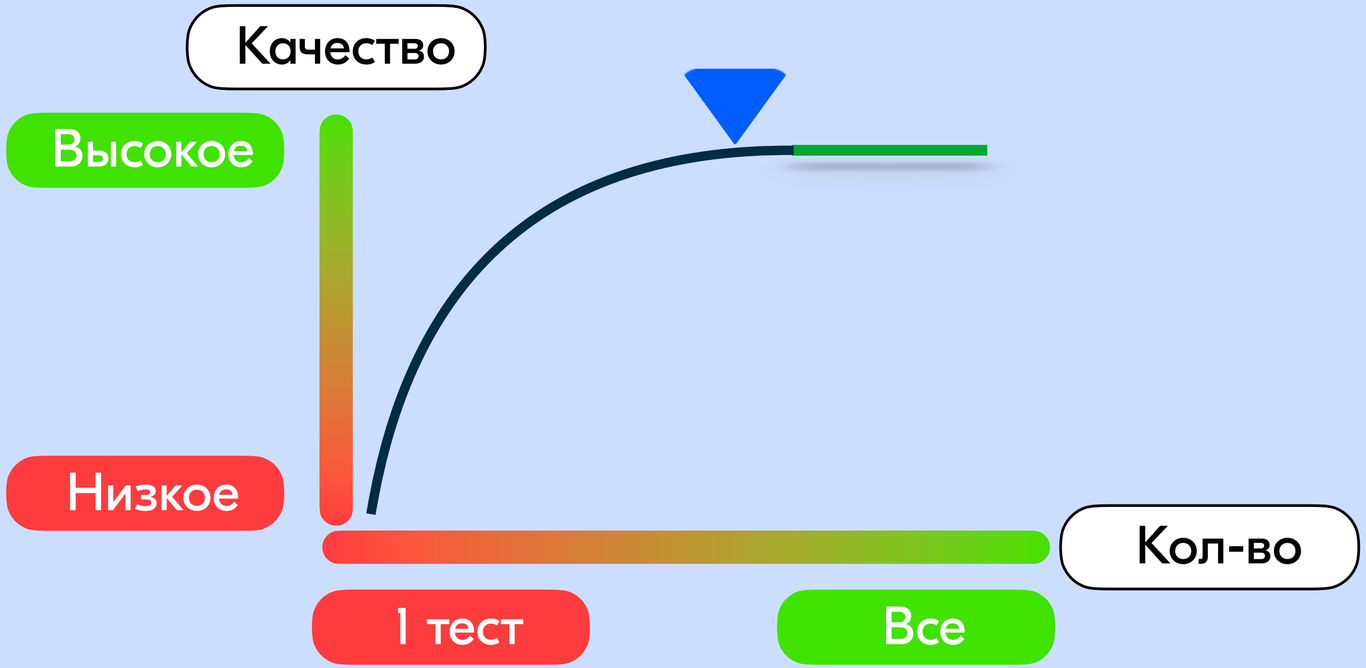
Зависимость качества тестирования от количества тестов

ВАЖНО

Наступает момент, когда каждый новый запущенный тест на сервис не увеличивает качество тестирования

То есть попадает в зону плато

Но как на самом деле



Зависимость качества тестирования от количества тестов

НАША ЗАДАЧА

Собрать такое количество тестов, чтобы обеспечить максимальное качество наименьшим набором тестов

Итого

OZON tech

Проблема

Хотим чаще релизиться

Хотим низкий TTM

Как будем решать

Уменьшать кол-во запускаемых тестов

Анализировать импакт тестов

Как будем проводить анализ

Трассировка запросов

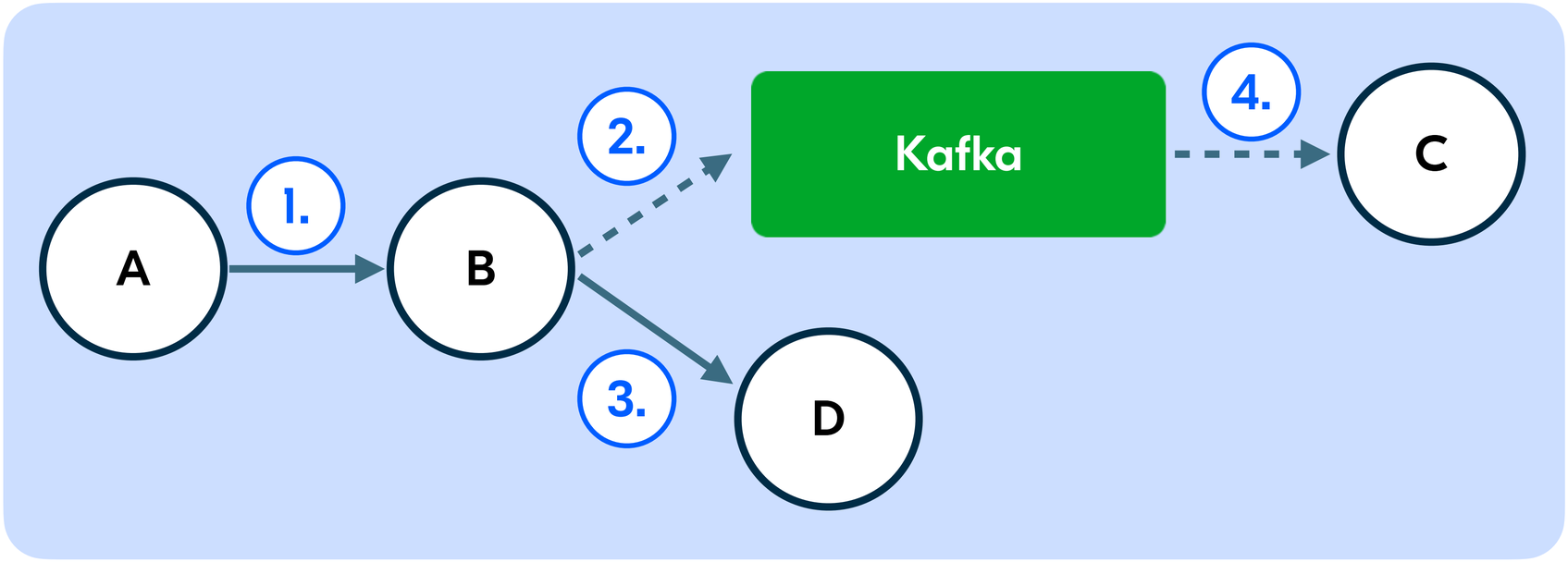
2

Трассировка запросов

Трассировка

OZON tech

Отслеживание последовательности вызова ручек сервисов.



Трассировка

OZON tech

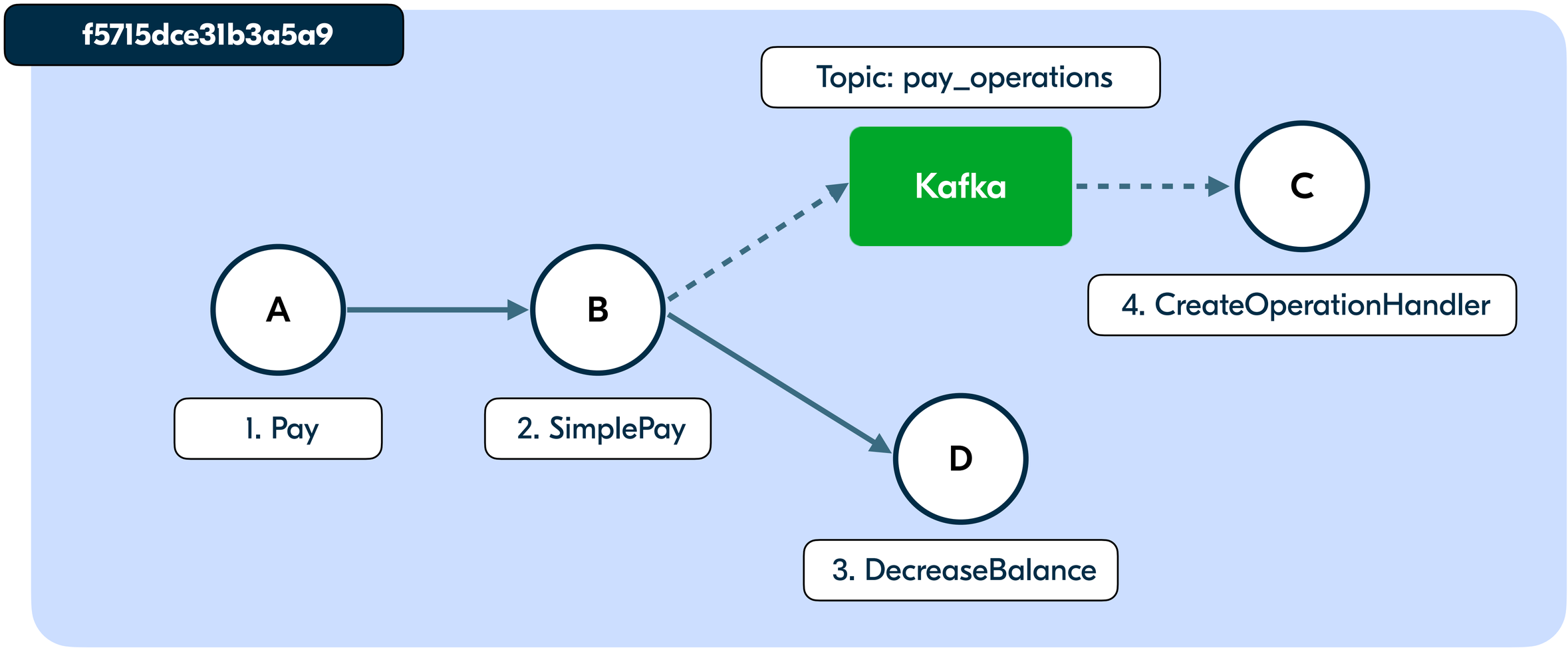
Строит путь запросов, который позволяет:

A. Найти узкие места в производительности

B. Визуализировать последовательности вызовов

C. Собрать информацию для дальнейшего анализа

Трассировка



Какую информацию мы можем получить

1. В какие сервисы попадает запрос
2. В какие ручки сервисов попадает запрос
3. По какому пути ходят запросы

На предыдущем докладе мы разбирали как применить знание о том, в какие сервисы попадают тесты

Но если коротко

OZON tech

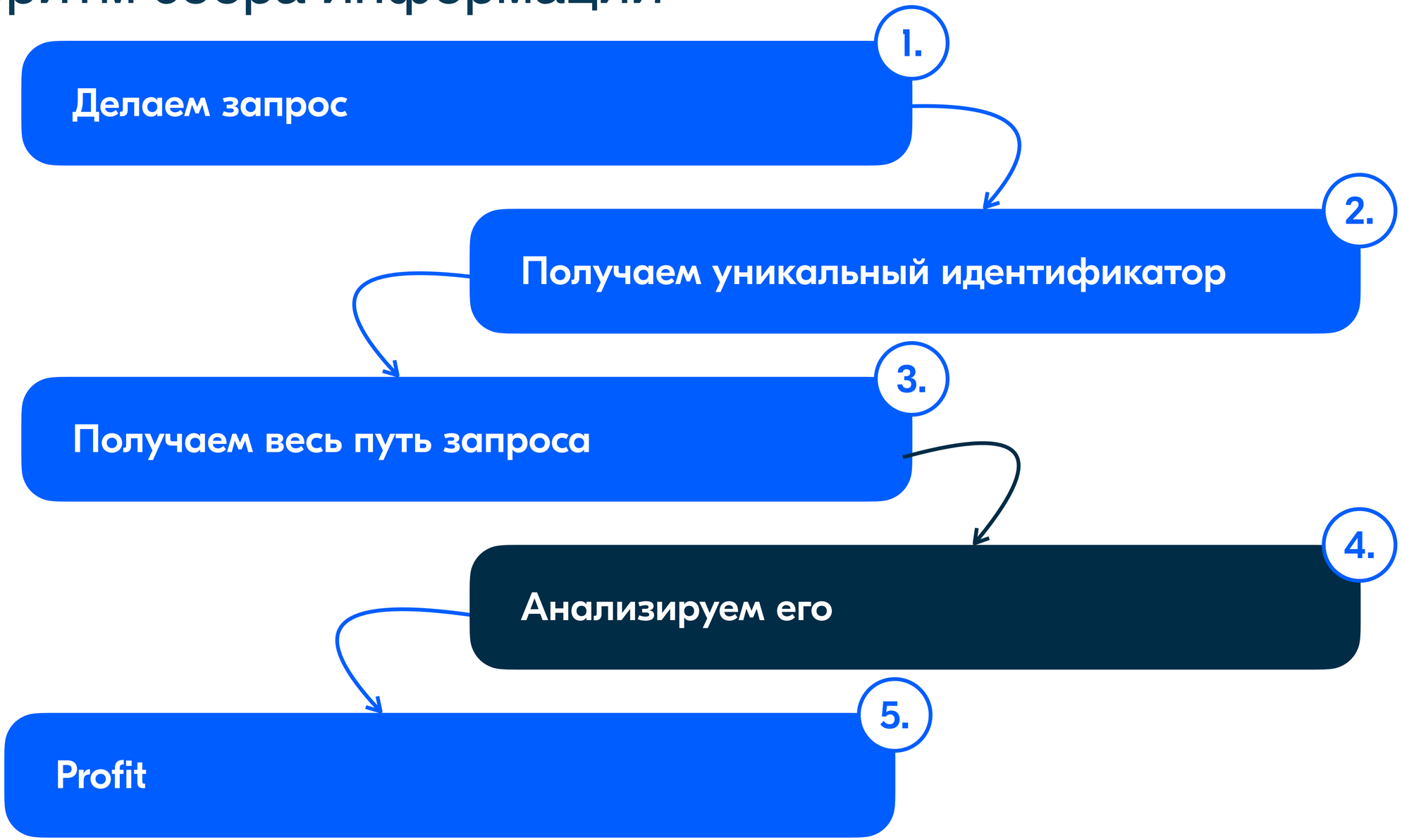
Зная в какие сервисы попадают тесты мы можем:

A. Автоматически размечать тесты для сервисов

B. Понимать какие сервисы не покрыты тестами

C. Не запускать ненужные тесты и запускать нужные

Алгоритм сбора информации



Что еще важно знать о трейсинге

Граф межсервисного взаимодействия

Трассировка запроса

Service A: Handler 1

Service B: Handler 2

Service A: Handler3

Это нам пригодится дальше

3

Минимизация

НАША ЗАДАЧА

Собрать такое количество тестов для сервиса, чтобы обеспечить максимальное качество наименьшим набором тестов

Мы разберем:

1. Минимизация по ручкам
2. Минимизация по сценариям

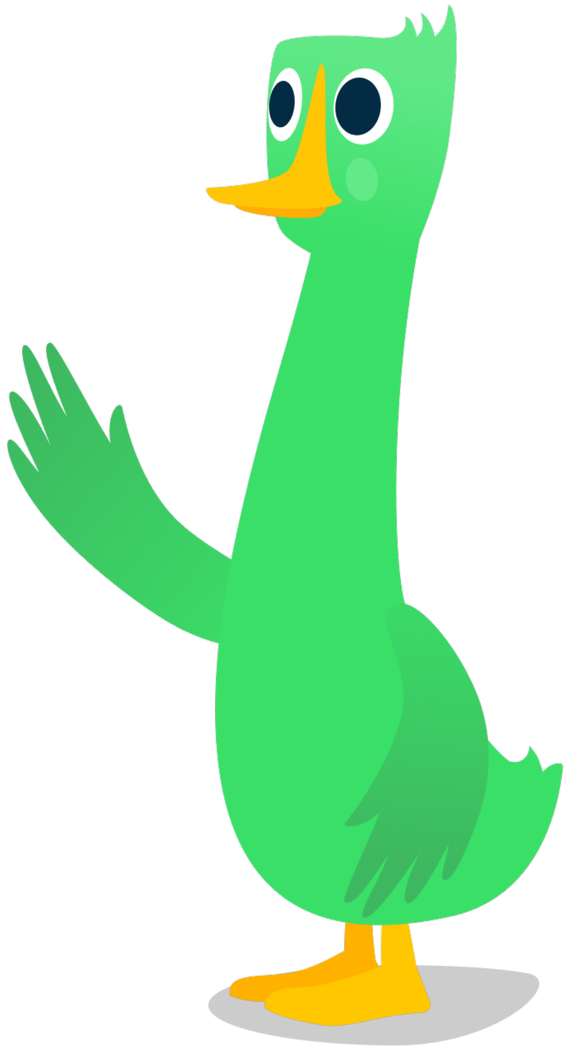
Оба алгоритма похожи между собой основной идеей и проблемами. Но отличаются качеством минимизации и глубиной понимания тестов.

Управляющая идея

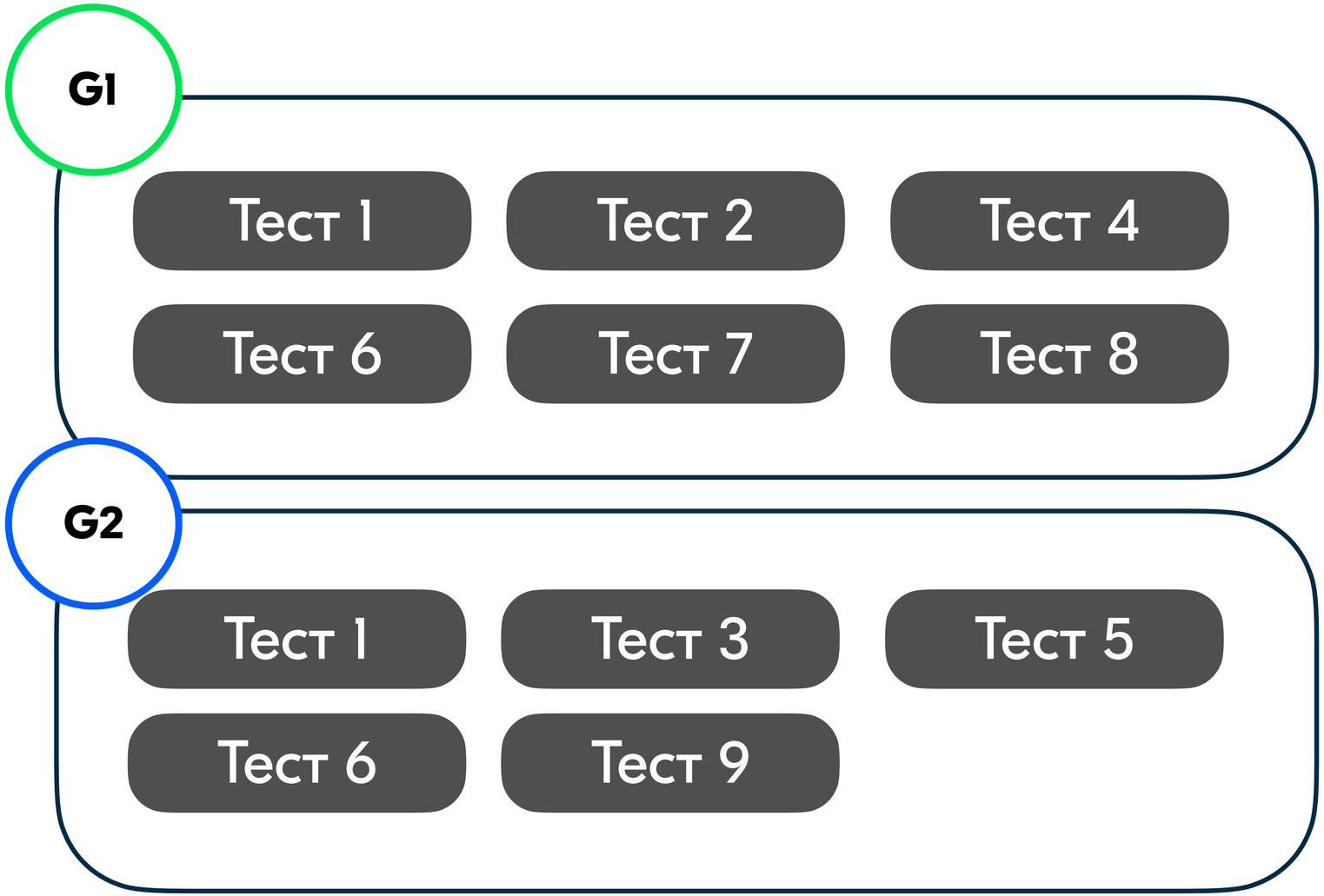
1. Группируем тесты для сервиса по определенному признаку
2. Какие тесты попадают в сервис мы знаем из трейсинга
3. Один тест может попадать сразу в несколько групп
4. Выбираем N тестов из группы по весам

Параметры алгоритмов

1. N - количество тестов на группу
2. Параметр группировки
3. Алгоритм выбора веса



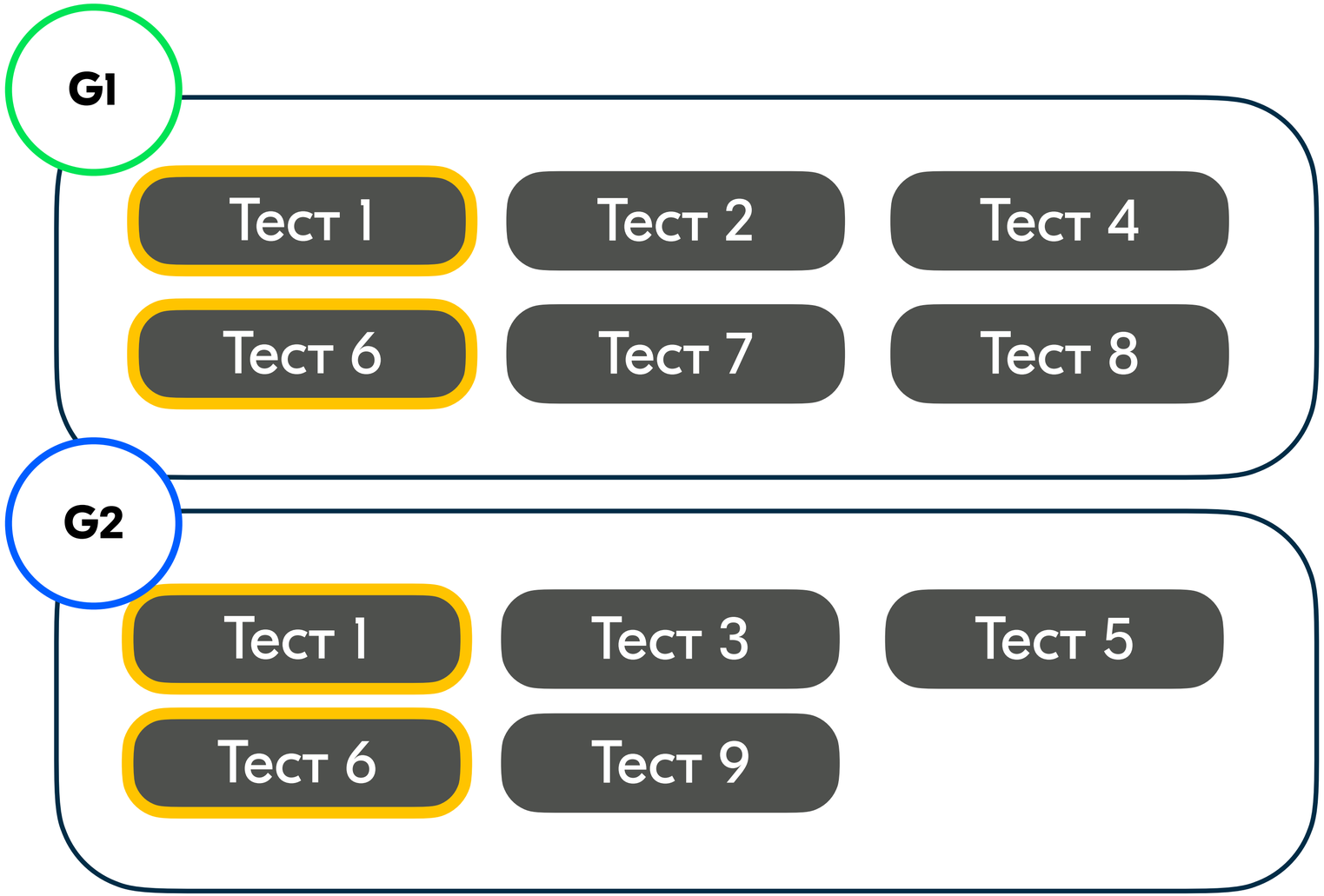
Выбор тестов для сервиса



2 группы тестов

Всего тестов: 9

Выбор тестов для сервиса

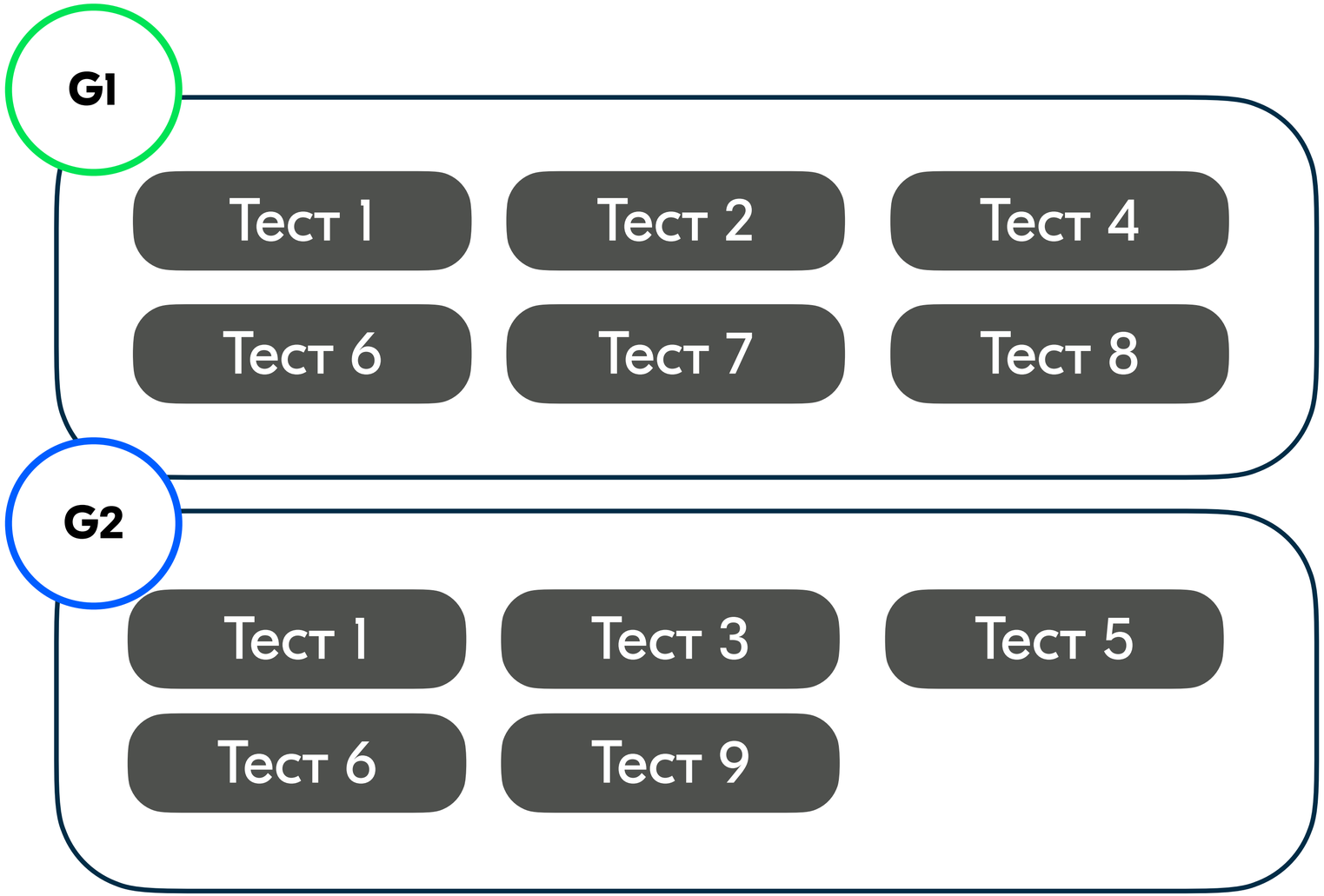


2 группы тестов

Всего тестов: 9

Тесты 1 и 6 в G1 и G2

Выбор тестов для сервиса



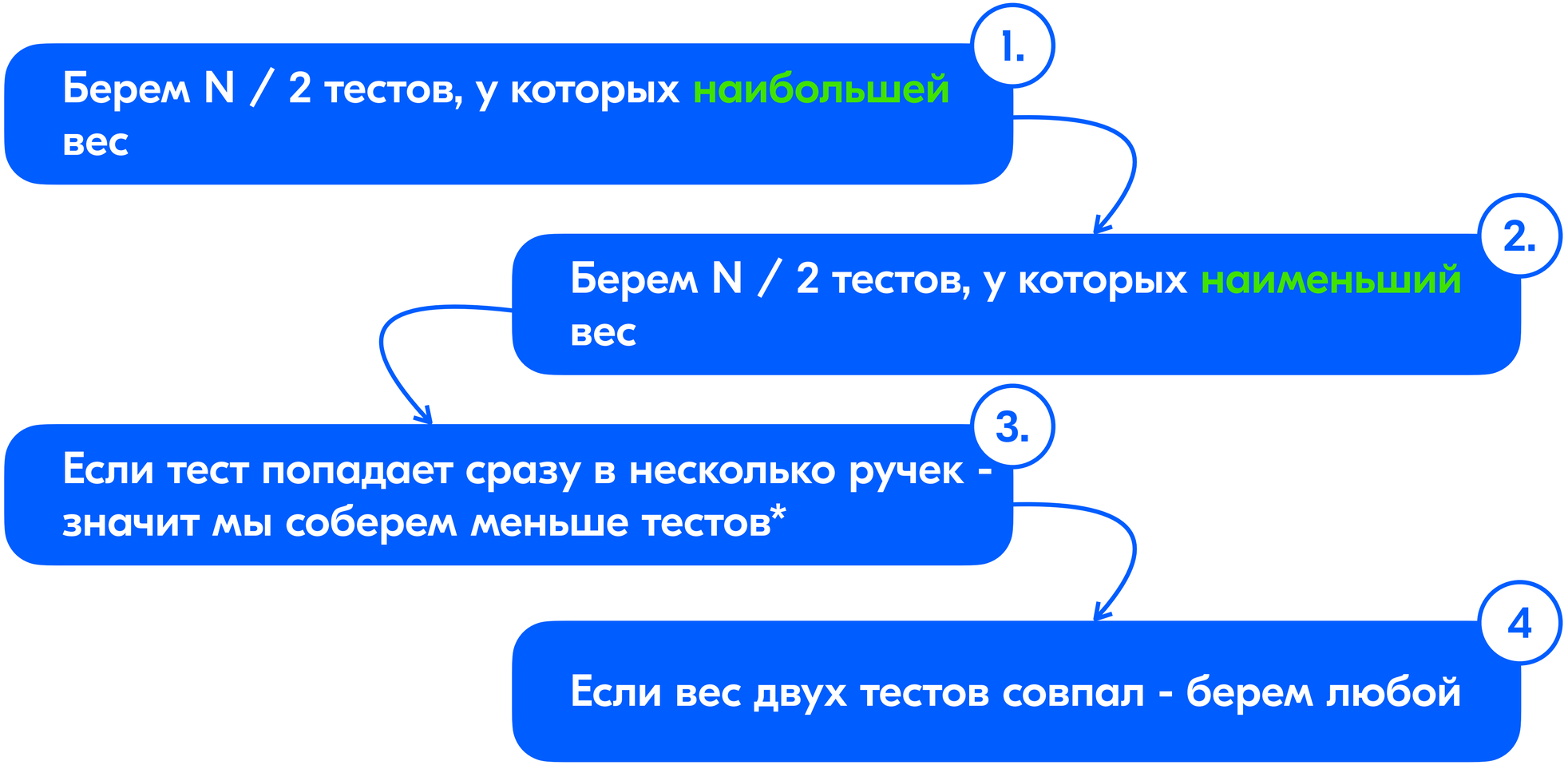
2 группы тестов

Всего тестов: 9

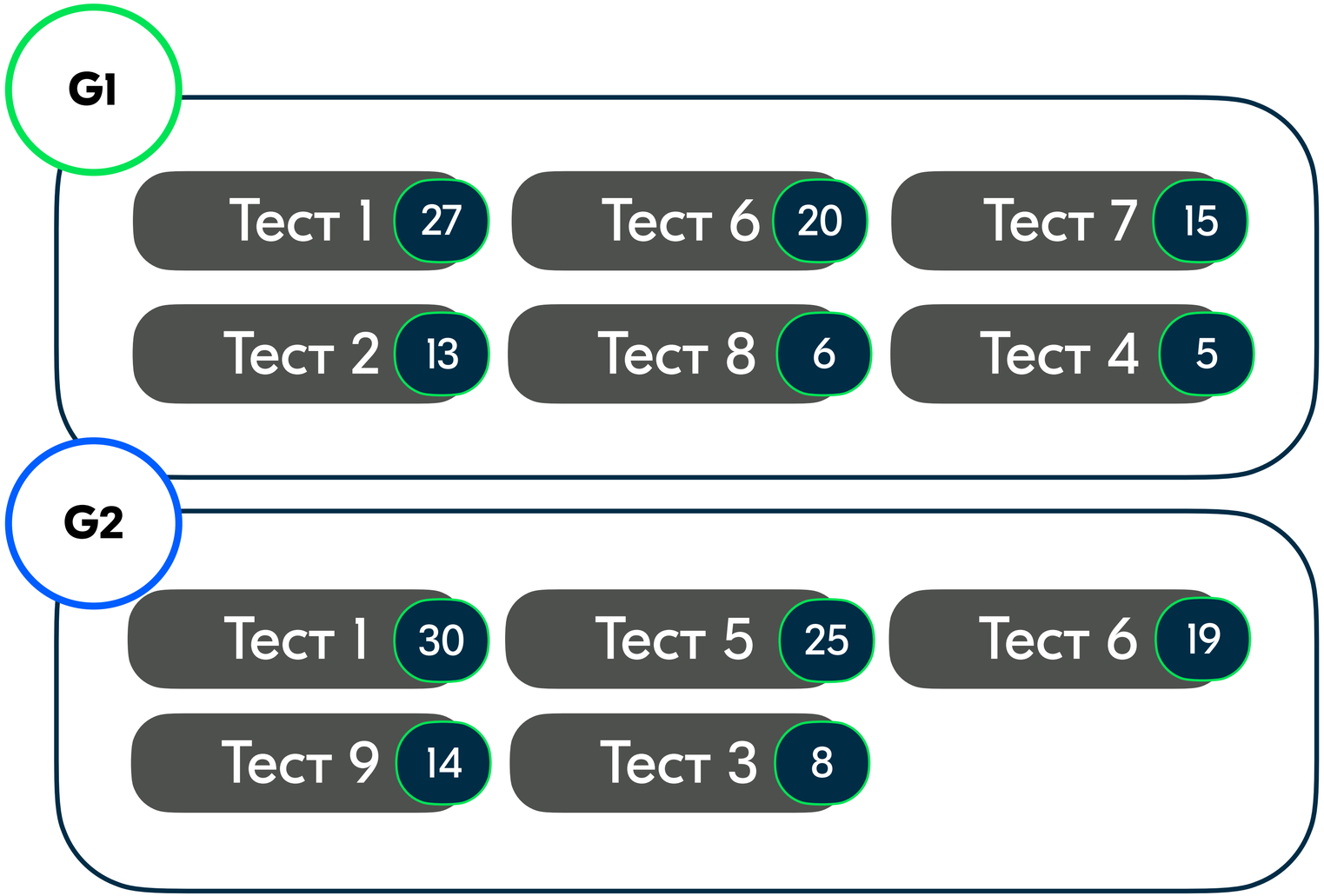
Тесты 1 и 6 в G1 и G2

Возьмем по 2 теста из группы

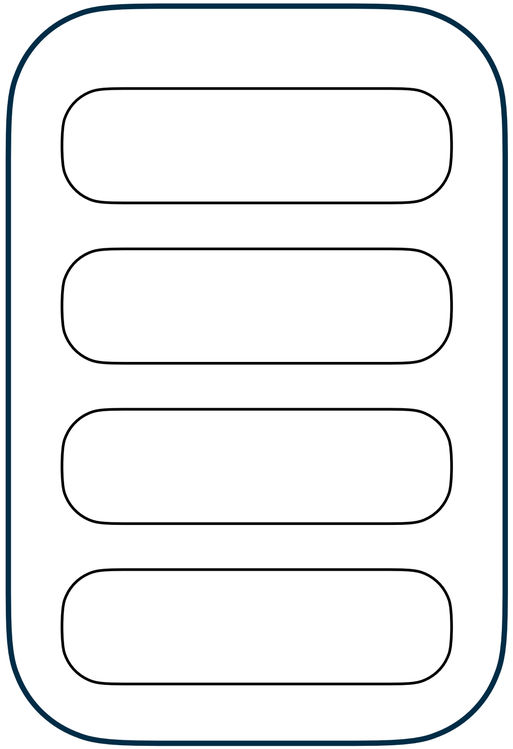
Как выбрать тесты из группы



Добавим веса тестам и отсортируем



По 2 теста из группы
- 1 с наибольшим весом
- 1 с наименьшим



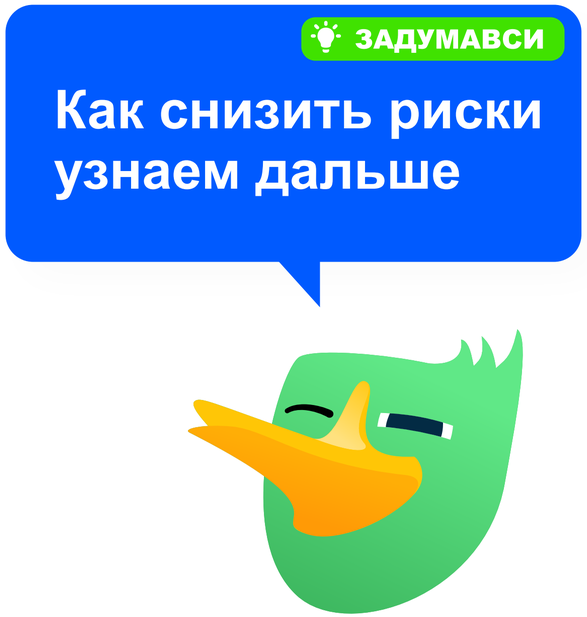
Таким образом вместо 9
тестов мы возьмем только 3 (4)

Звучит конечно прикольно, но

1. Трейсинг позволяет нам узнать только о межсервисном взаимодействии

2. Мы ничего не знаем о внутренней работе сервиса

3. Есть вероятность пропустить баг в прод



3.1

Минимизация по ручкам

Параметры алгоритма

1. Группировка тестов будет по ручкам

2. Тесты, затрагивающие одну и ту же ручку находятся в одной группе

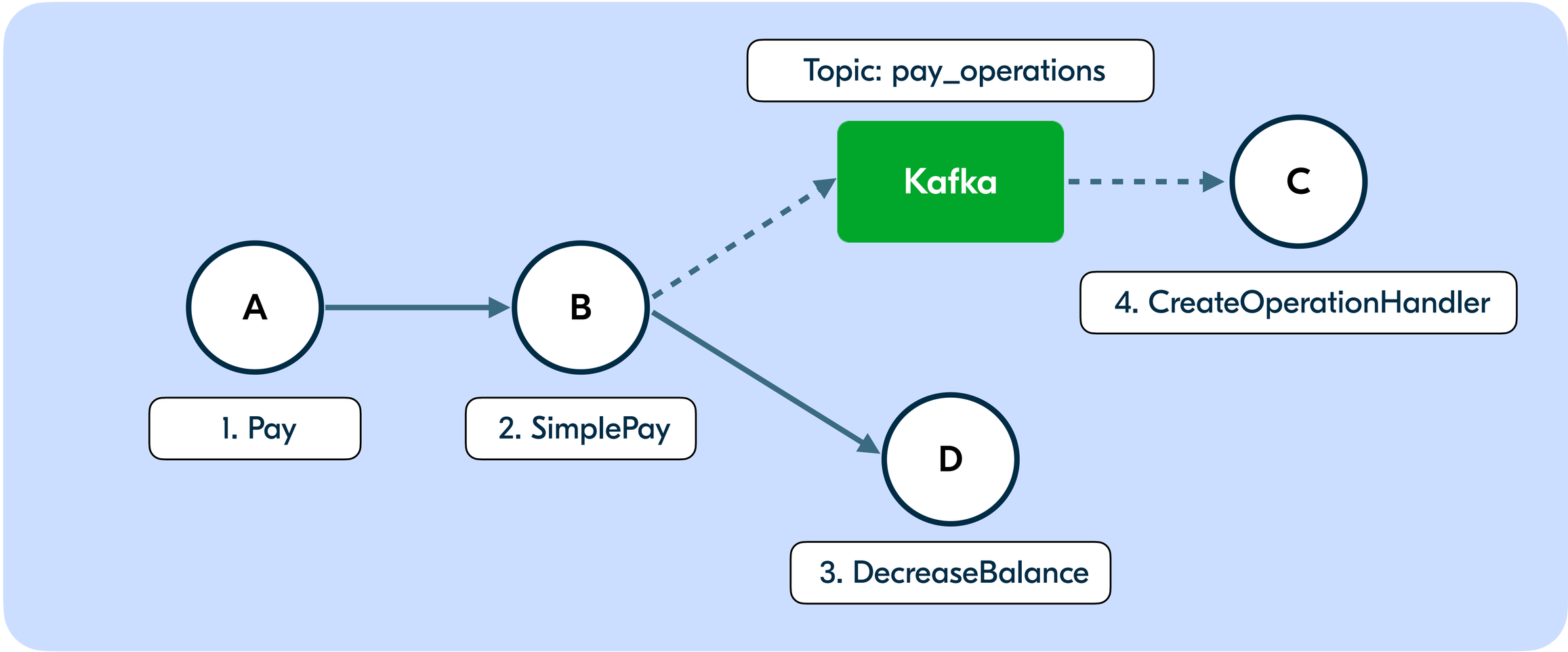
3. Весом теста будет количество сервисов, в которые тест попадет

Что нам нужно

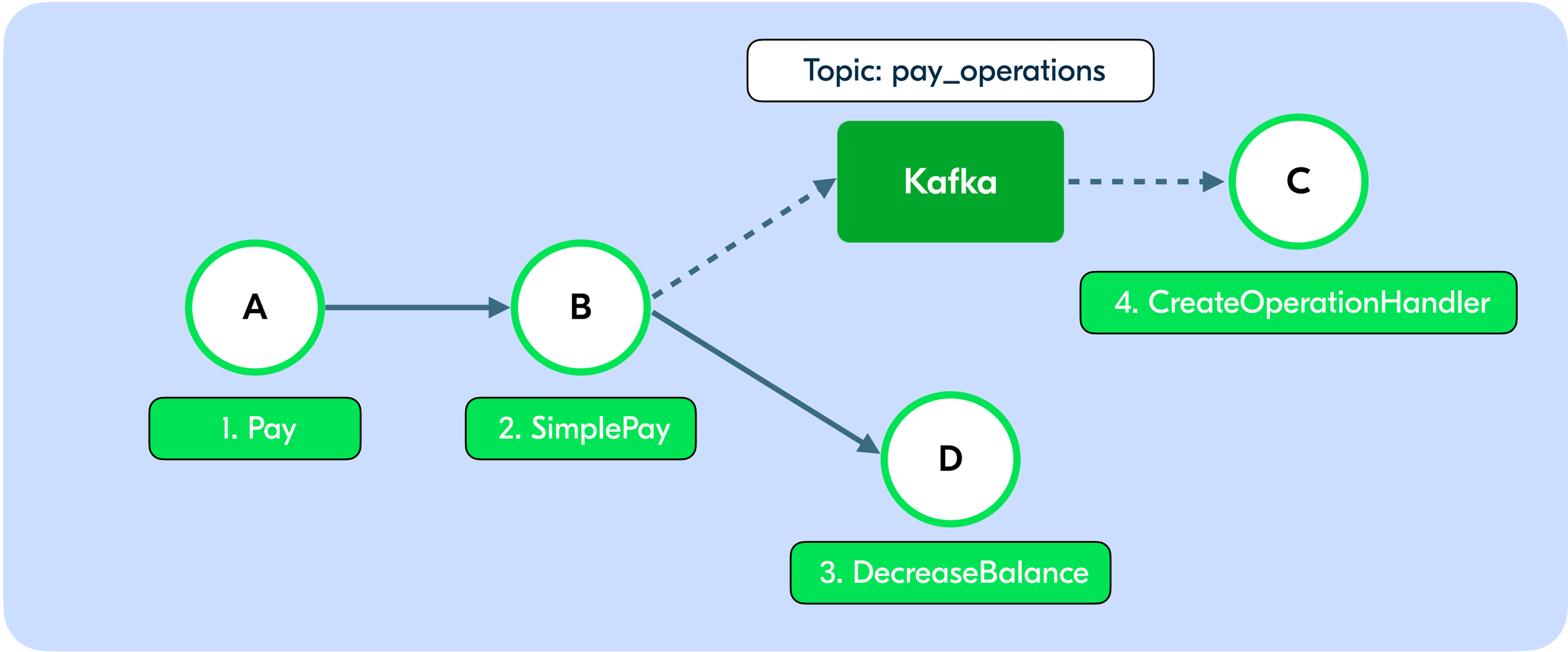
1. Знать в какие сервисы попадает тест

2. Знать в какие ручки попадает тест

Трассировка



Трассировка



Минимизация по ручкам

Ручки сервиса	Кол-во тестов
GetUser	1000
AddProduct	300
Pay	250
AddReview	50
ChangePwd	10

ВНИМАНИЕ

А что если запускать не больше 60 тестов на ручку!

Минимизация по ручкам

Ручки сервиса	Кол-во тестов	
GetUser	1000	60
AddProduct	300	60
Pay	250	60
AddReview	50	50
ChangePwd	10	10

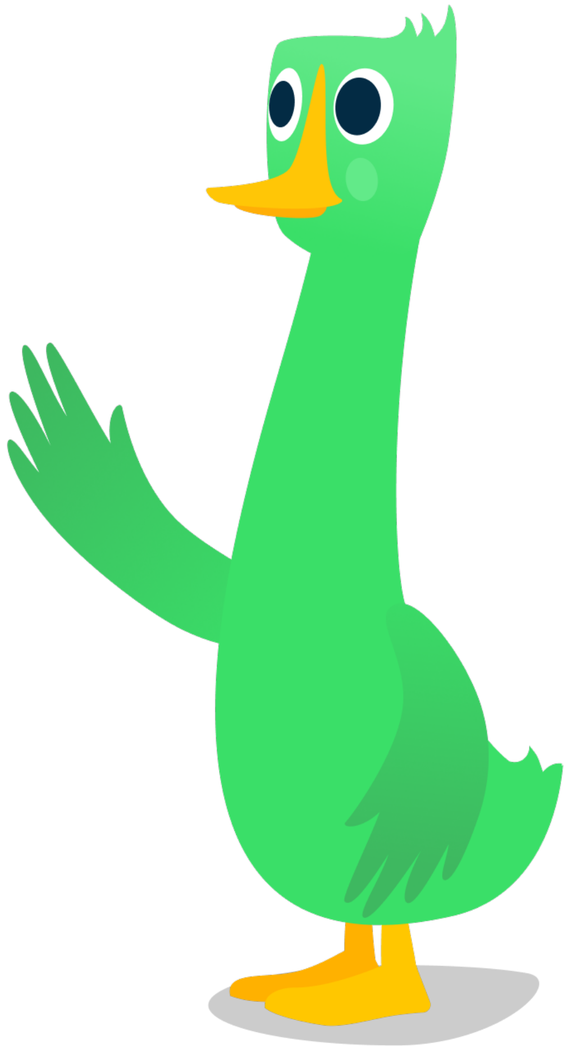
ВНИМАНИЕ

А что если запускать не больше 60 тестов на ручку!

Тезисы

1. Чем больше тестов попадает в ручку - тем больше минимизируем

2. Чем меньше тестов попадает в ручку - тем меньше минимизируем



Выбор тестов для запуска

Номер	Тест	Кол-во сервисов	
1.	Test1	50	✓
2.	Test2	50	✓
3.	Test3	30	✗
4.	Test4	10	✓
5.	Test5	5	✓

Итого

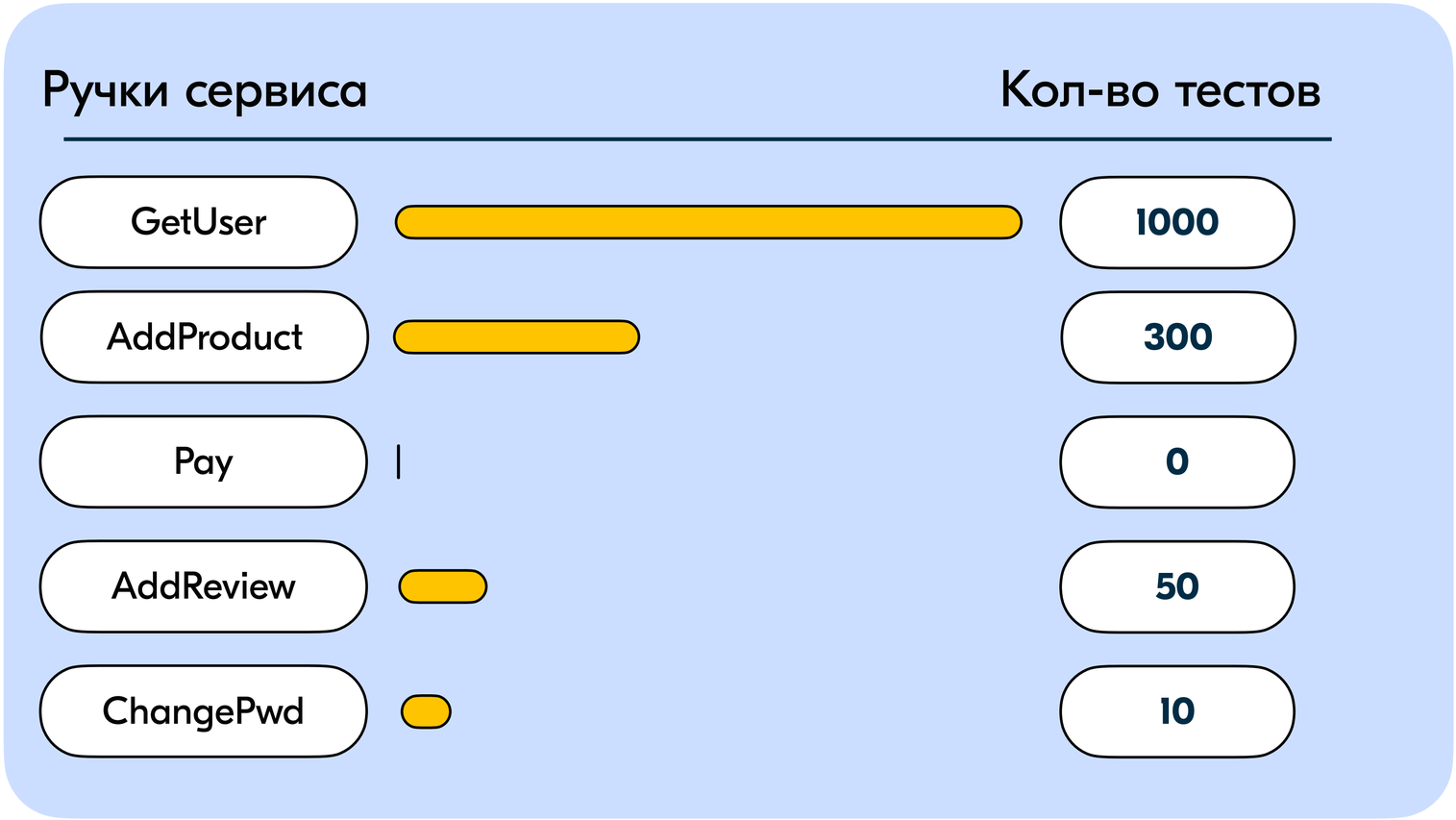
Зная в какие ручки попадают наши тесты мы можем

A. Уменьшить количество тестов для каждой ручки

B. Узнать покрытие* тестами ручек сервиса

C. Понять какие ручки не задевают наши тесты

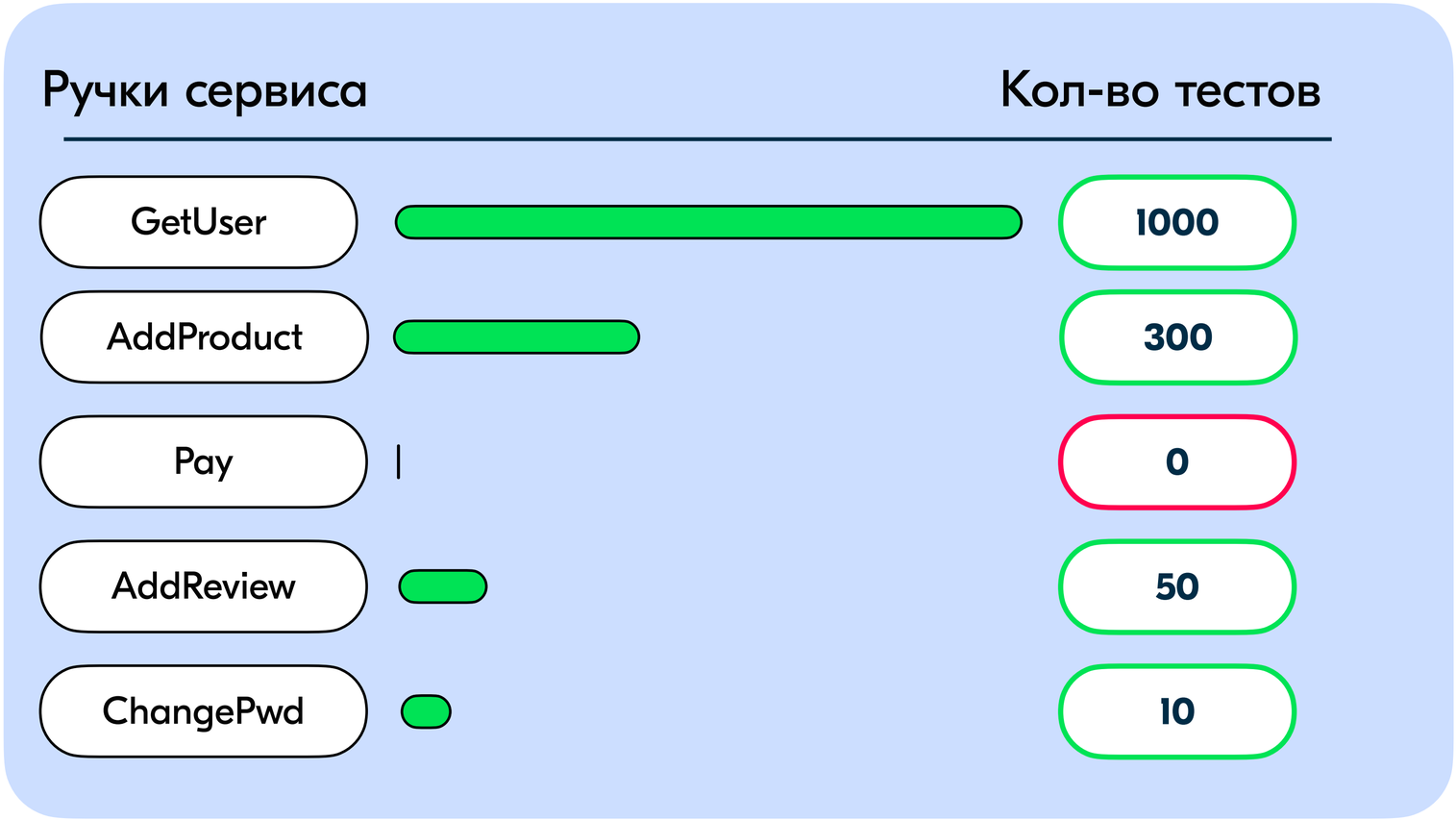
Покрытие по ручкам



Всего 5 ручек

Покрыто 4 ручки

Покрытие по ручкам



Всего 5 ручек

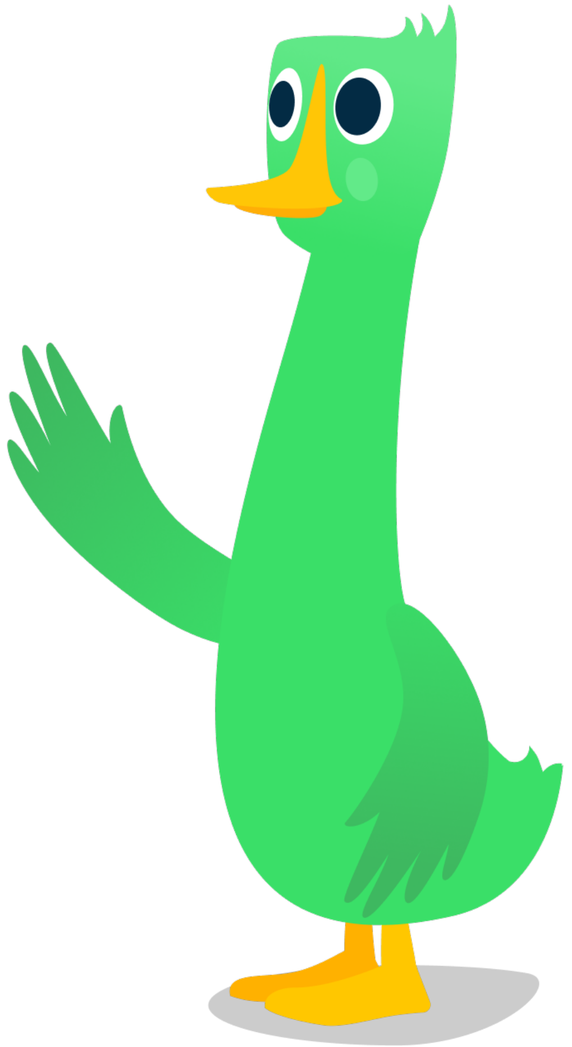
Покрыто 4 ручки

80% покрытие

Результаты

Параметры алгоритма

1. 30 - количество тестов на группу
2. Группировка по ручкам
3. Вес = количество сервисов теста



Результаты при 30 тестах на ручку

< 2000 тестов

В рамках любого запуска

ВАЖНО

на 53%

Запускаем меньше тестов (AVG)

ПРОЦЕНТ МИНИМИЗАЦИИ

20%

Для сервисов с < 500 тестами

~18 мин*

Экономия времени на прогонах крупных сервисов

16 минут

Максимальный прогон тестов (90 персентиль)

ПРОЦЕНТ МИНИМИЗАЦИИ

67%

Для сервисов с >= 500 тестами

Что нас не устраивало в этом алгоритме

1. Низкая гранулярность

2. Вообще ничего не понимаем о работе теста

3.2

Минимизация по сценариям

Параметры алгоритма

1. Группировка тестов будет по «сценариям» работы ручки

2. Тесты, затрагивающие один и тот же сценарий находятся в одной группе

3. Весом теста будет количество сценариев, в которые тест попадет в рамках сервиса

Да кто такой этот ваш сценарий

Определение сценария

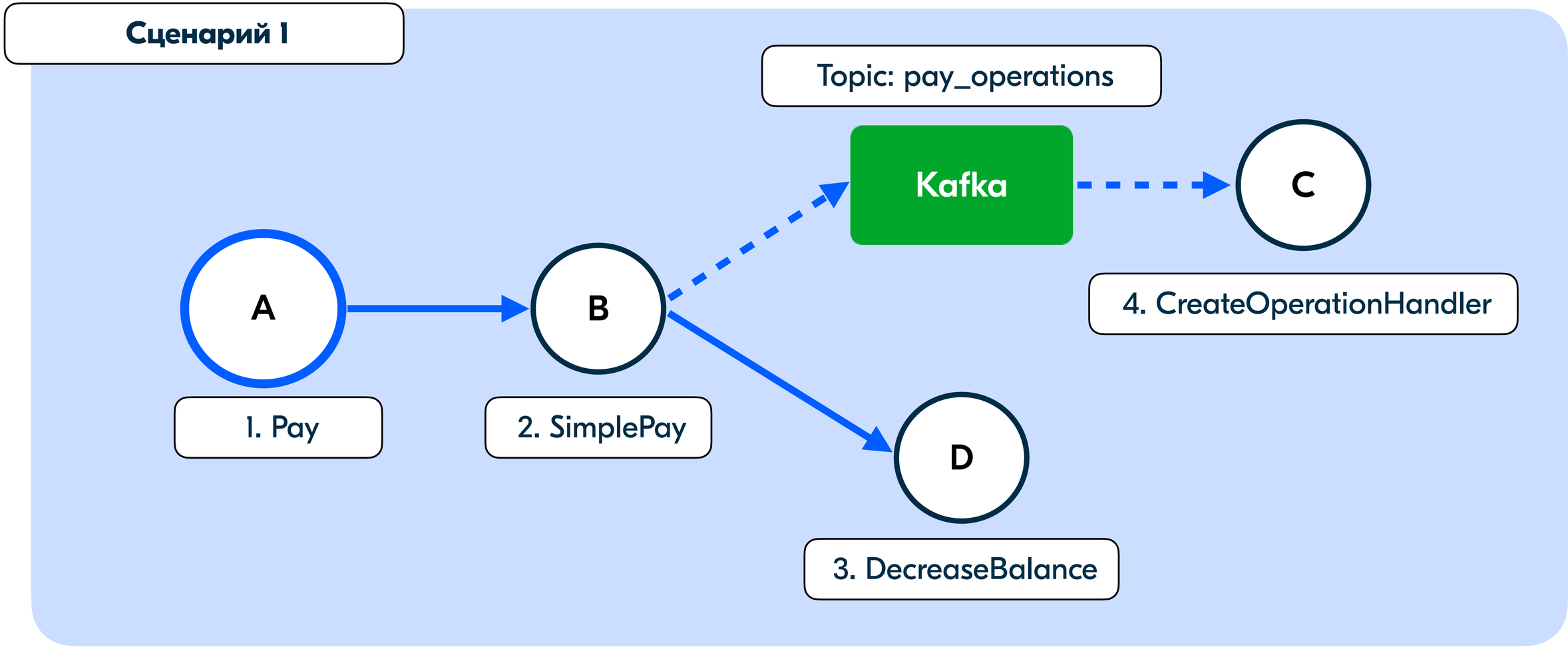
Определение

Сценарий - уникальный путь, который проходит запрос относительно ручки сервиса

Считаем, что тесты проверяют одно и то же, если в рамках ручки у них одинаковый сценарий

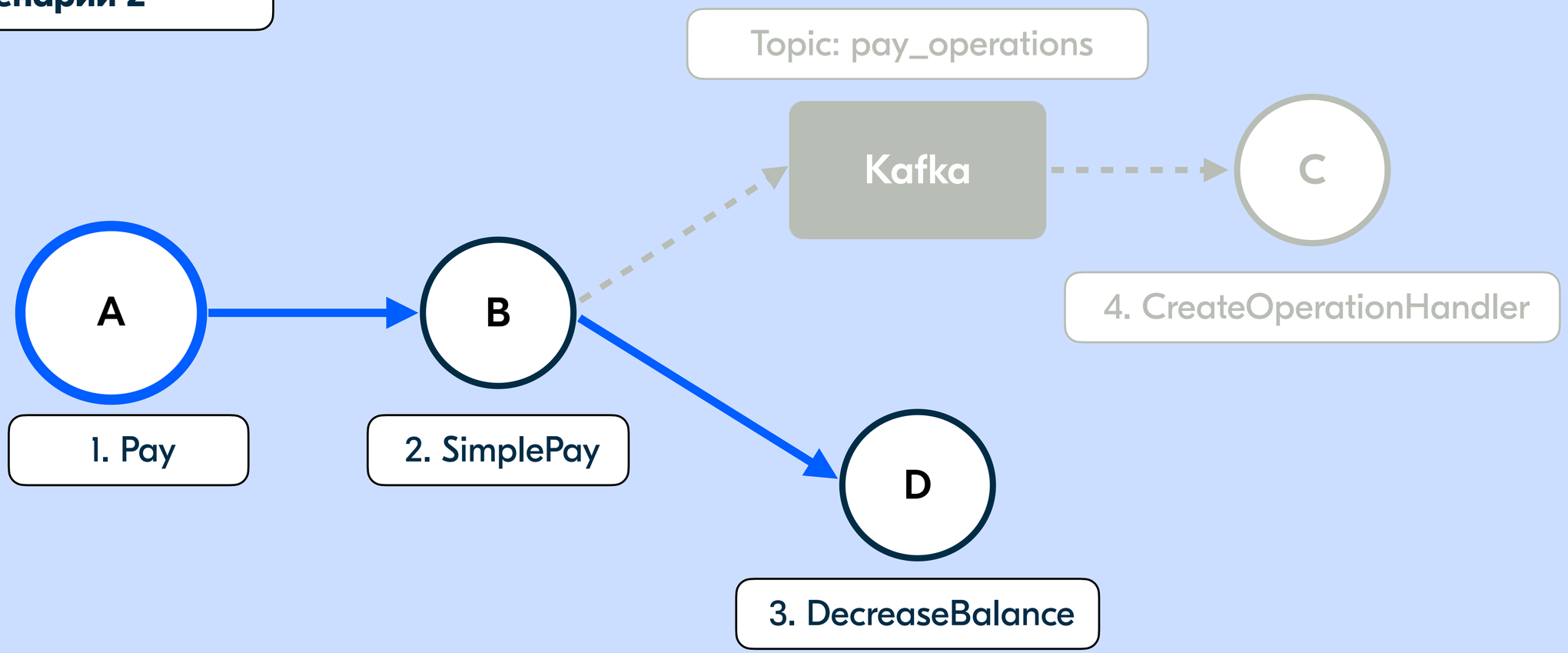
То есть разные пути запросов = разные сценарии

Путь запроса относительно сервиса A



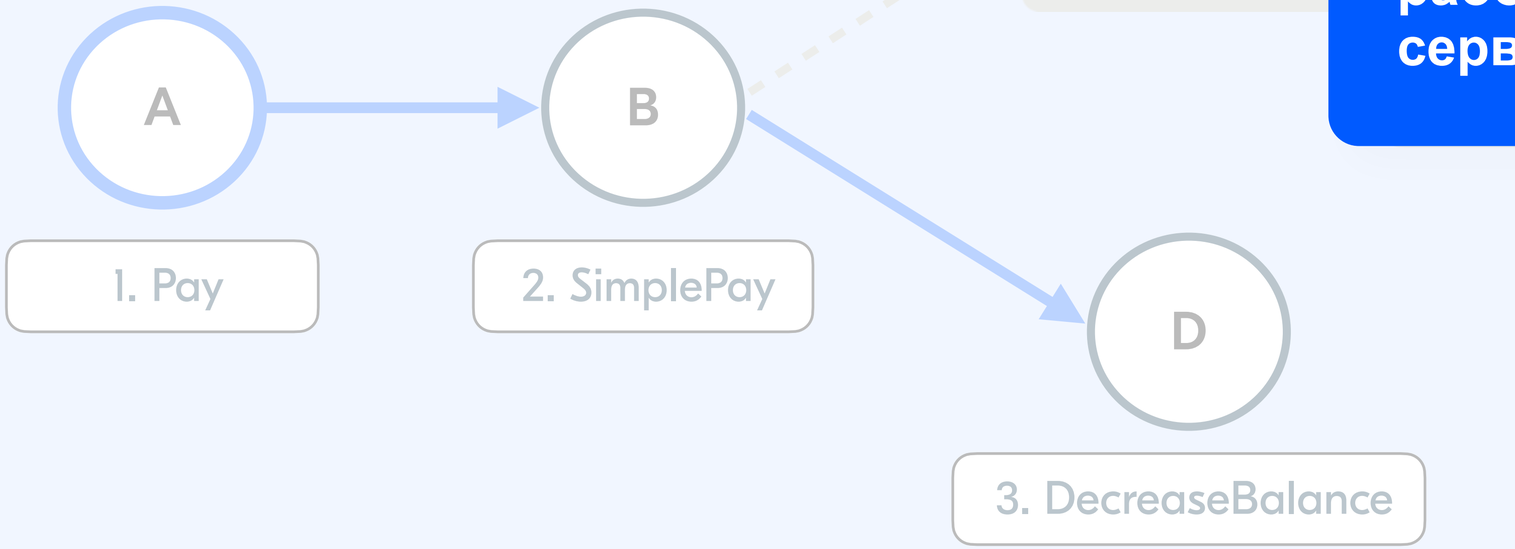
Путь запроса относительно сервиса A

Сценарий 2



Путь запроса относительно сервиса A

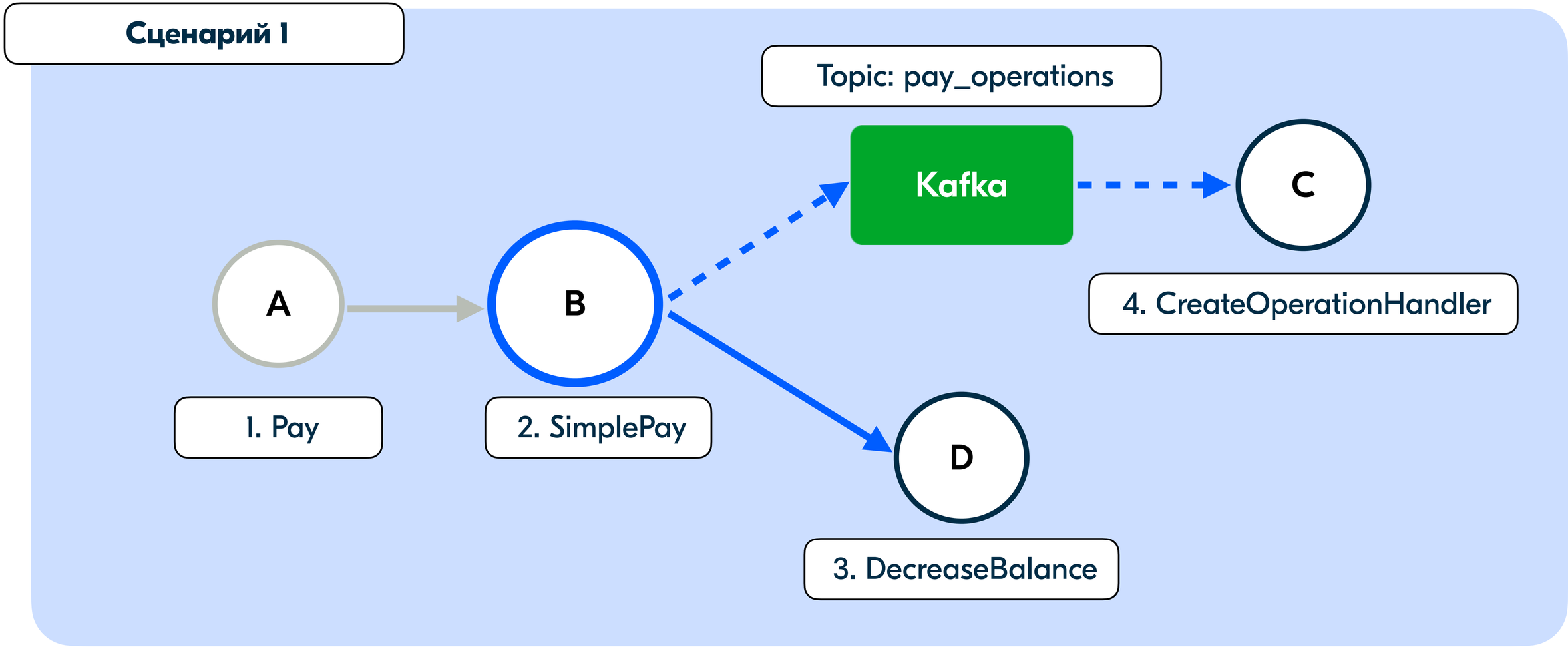
Сценарий 2



ВНИМАНИЕ

Мы посмотрели 2 разных сценария работы ручки Pay сервиса A

Путь запроса относительно сервиса B



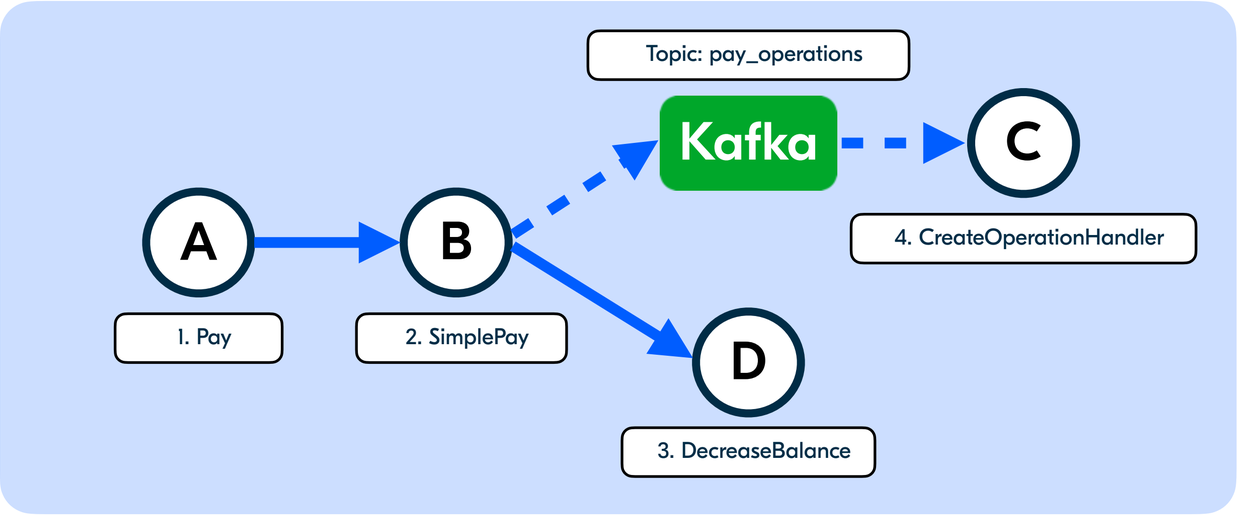
То есть

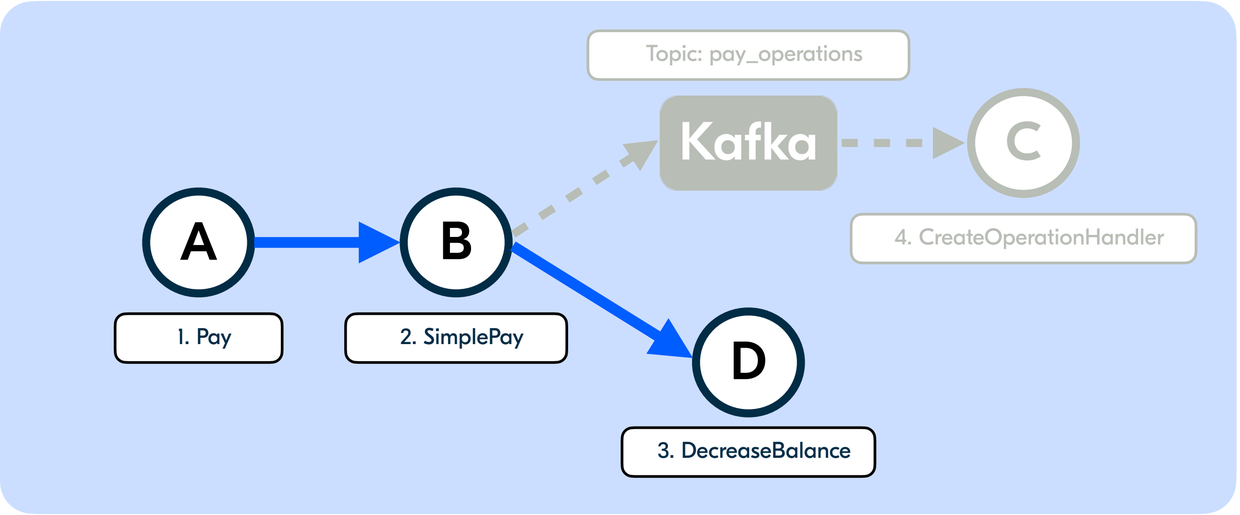
Для сервиса A и B:

A. Есть сценарий полного обхода сервисов

B. Есть сценарий, при котором взаимодействуют не все сервисы

C. Уникальный путь запроса относительно сервиса - сценарий





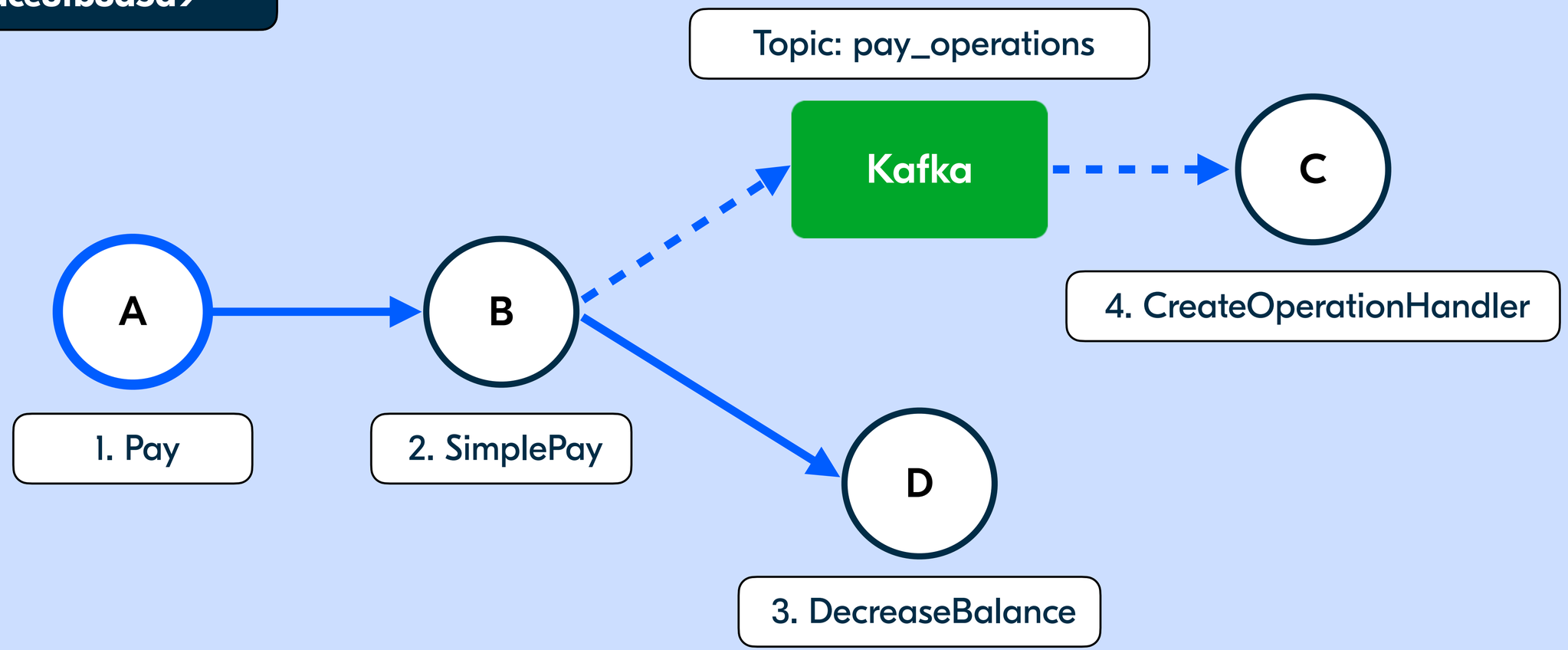
Чтобы получить сценарий нам нужно

1. Знать в какие сервисы попадает тест
2. Знать в какие ручки попадает тест
3. Знать по каким путям ходят запросы теста

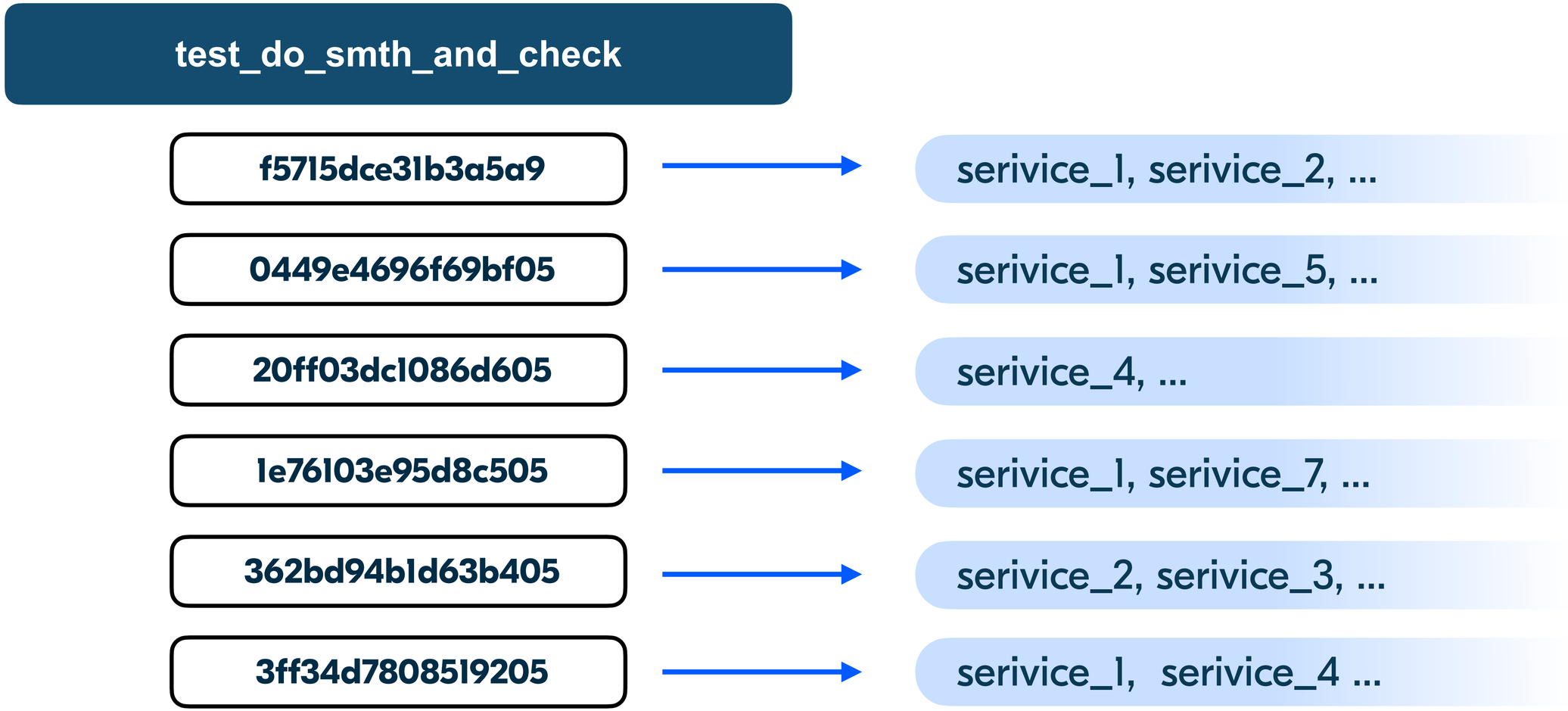
Как собирать сценарии

Запрос 1

f5715dce31b3a5a9



Запросы в рамках теста



Запросы в рамках теста

С каждым тестом может ассоциироваться много деревьев

3ff34d7808519205 → bozon-levinson-ng, bozon-...

Посчитаем

OZON tech

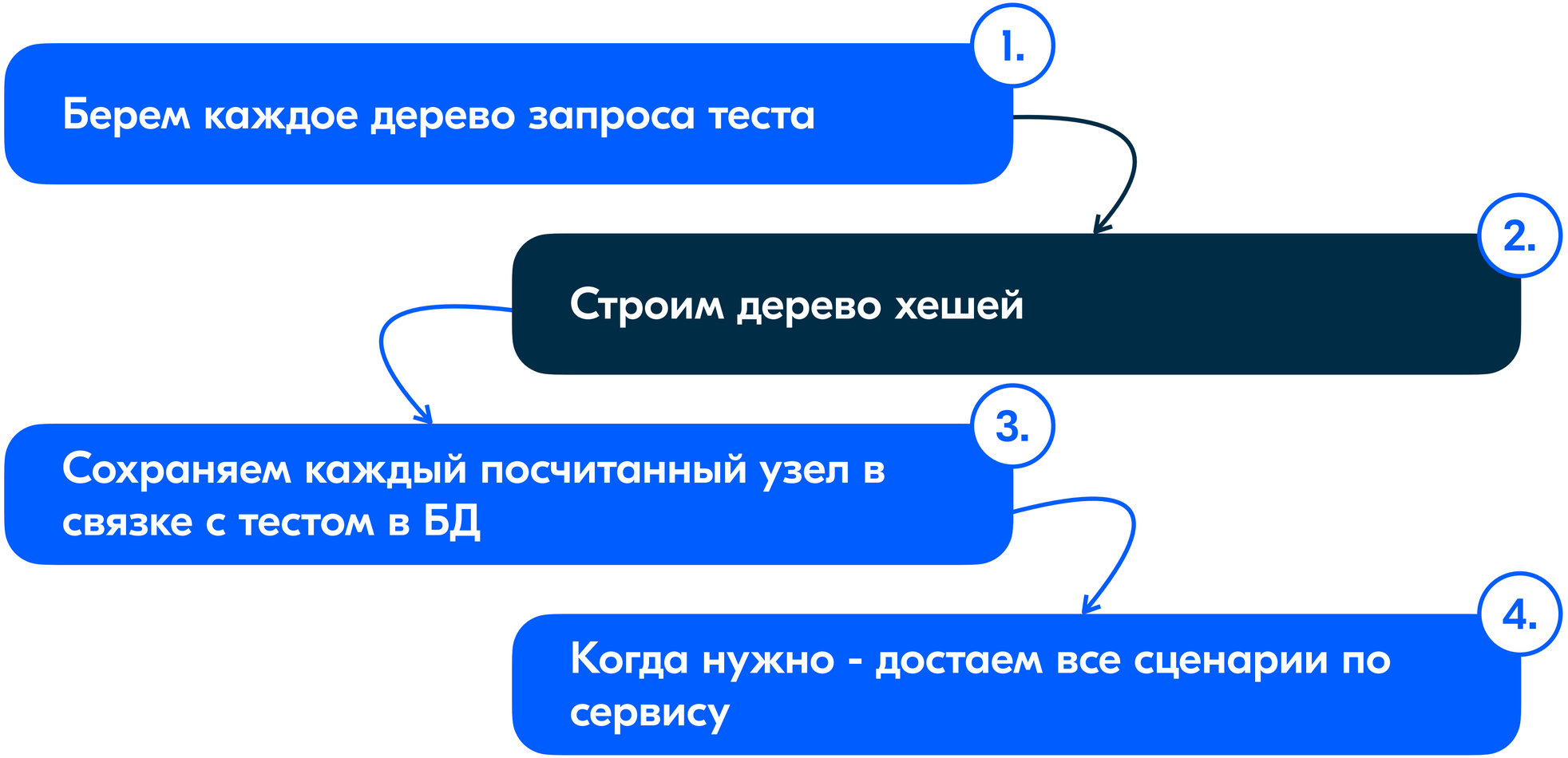
✅ **1000 тестов**

✅ **20 трейсов на тест**

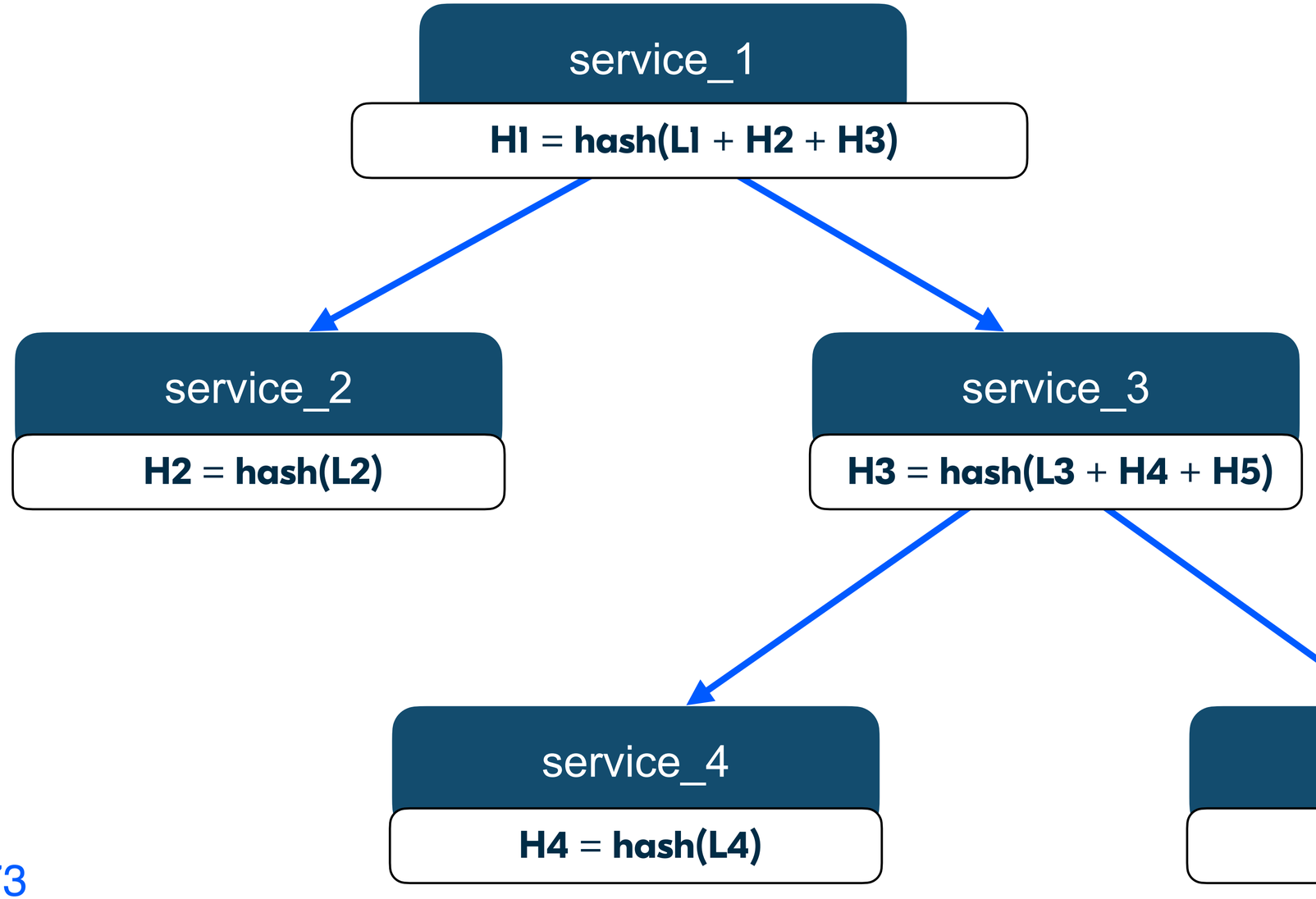
Итого: **1000** Кол-во тестов * **20** Кол-во трейсов = **20000** Различные деревьев

И все эти 20000 деревьев нужно проанализировать и сформировать пути относительно каждого сервиса, чтобы получить сценарии работы сервиса

Алгоритм сбора информации



Дерево хешей (дерево Меркла)



hash(x) - функция хеширования

L1 - название ручки сервиса 1

H1 - хеш сценария сервиса 1

Другими словами

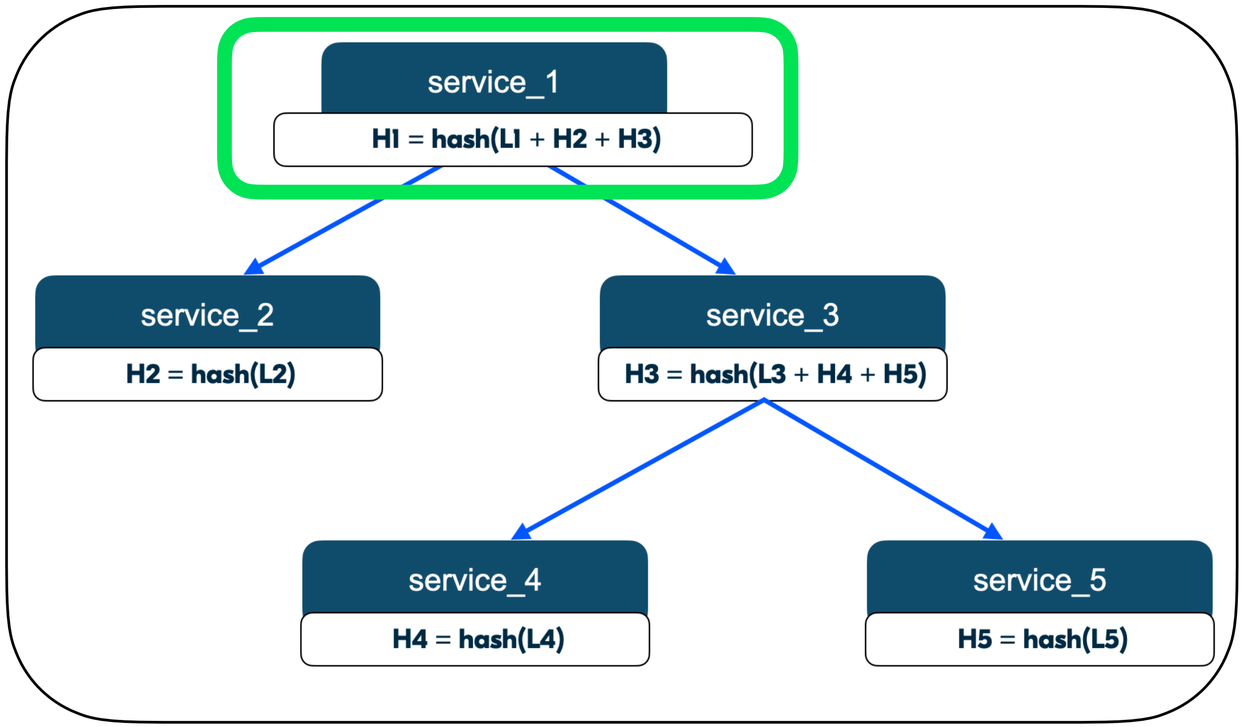
1. Нам не нужно сохранять каждое дерево запроса в рамках теста

2. Нам достаточно преобразовать каждое поддерево, относительно сервиса, в хеш и сохранить только его

3. Таким образом мы упрощаем себе как поиск сценариев, так и хранение связей

От дерева к таблице в бд

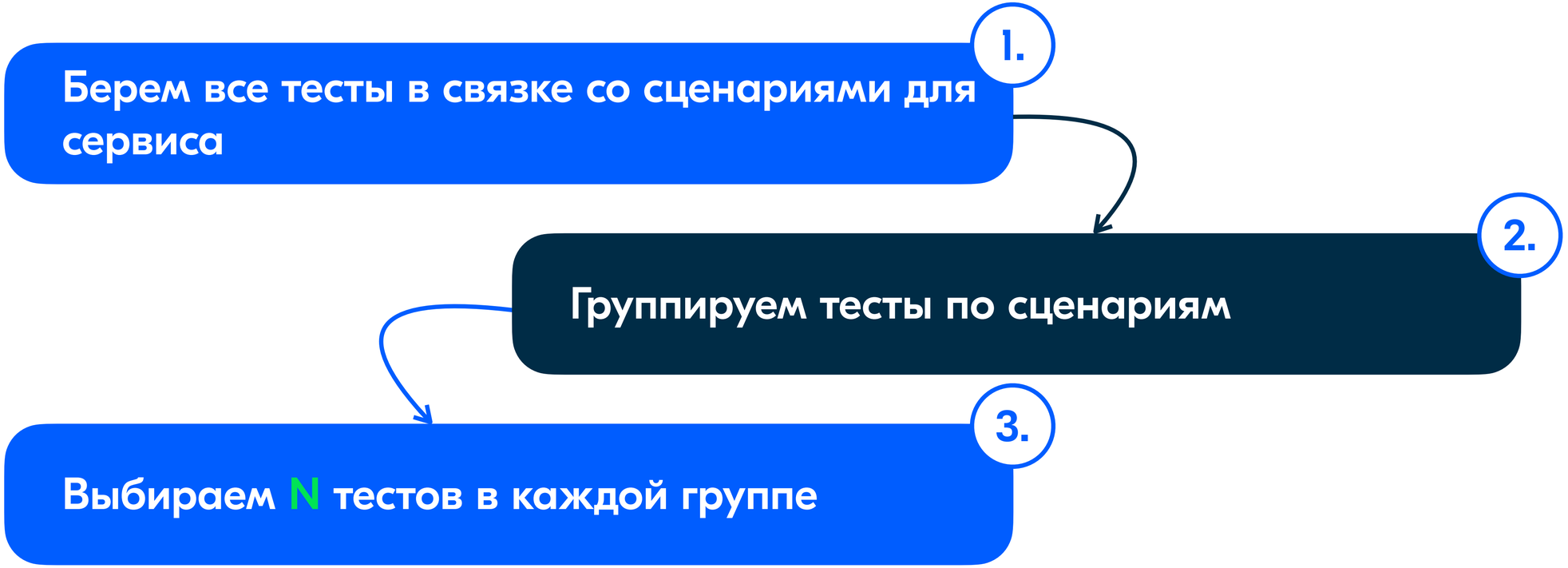
Сервис	Ручка	Сценарий	Тест
service_1	L1	H1	test_1
service_2	L2	H2	test_1
service_3	L3	H3	test_1
service_4	L4	H4	test_1
service_5	L5	H5	test_1



Сценарии в рамках одной ручки

Сервис	Ручка	Сценарий	Тест
service_1	L1	H1	test_1
service_1	L1	H2	test_2
service_1	L1	H3	test_1
service_1	L1	H4	test_3
service_1	L1	H5	test_3

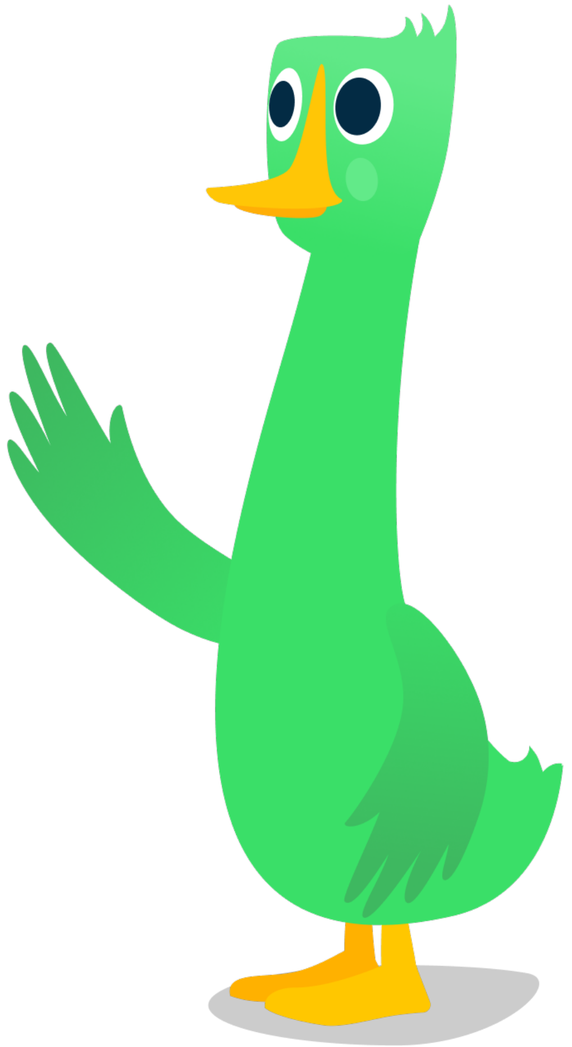
Алгоритм минимизации тестов



Выбор тестов в рамках сценария

Тезисы

1. Для конечных (листовых) сервисов кол-во сценариев = кол-ву ручек
2. Много сценариев = много тестов
3. Мало сценариев = мало тестов



Итого

OZON tech

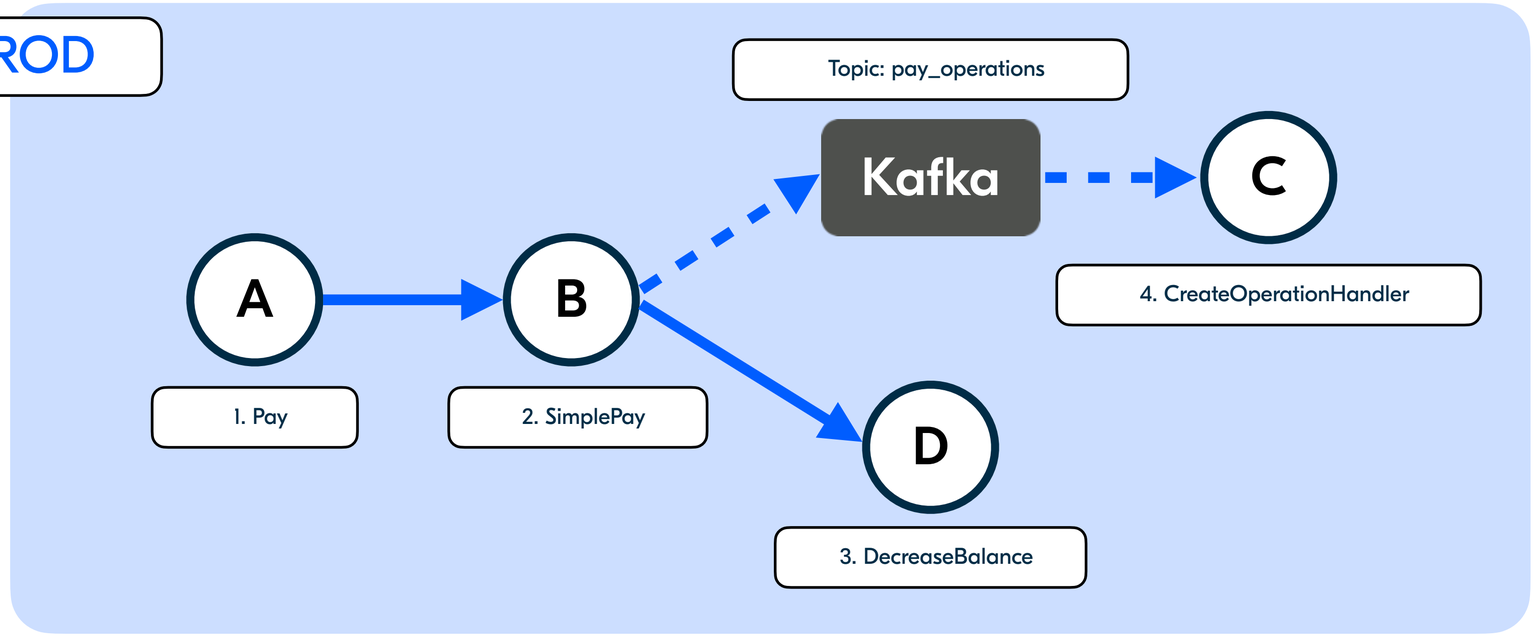
Зная пути запросов наших тестов мы можем

A. Точнее уменьшать кол-во тестов для сервисов

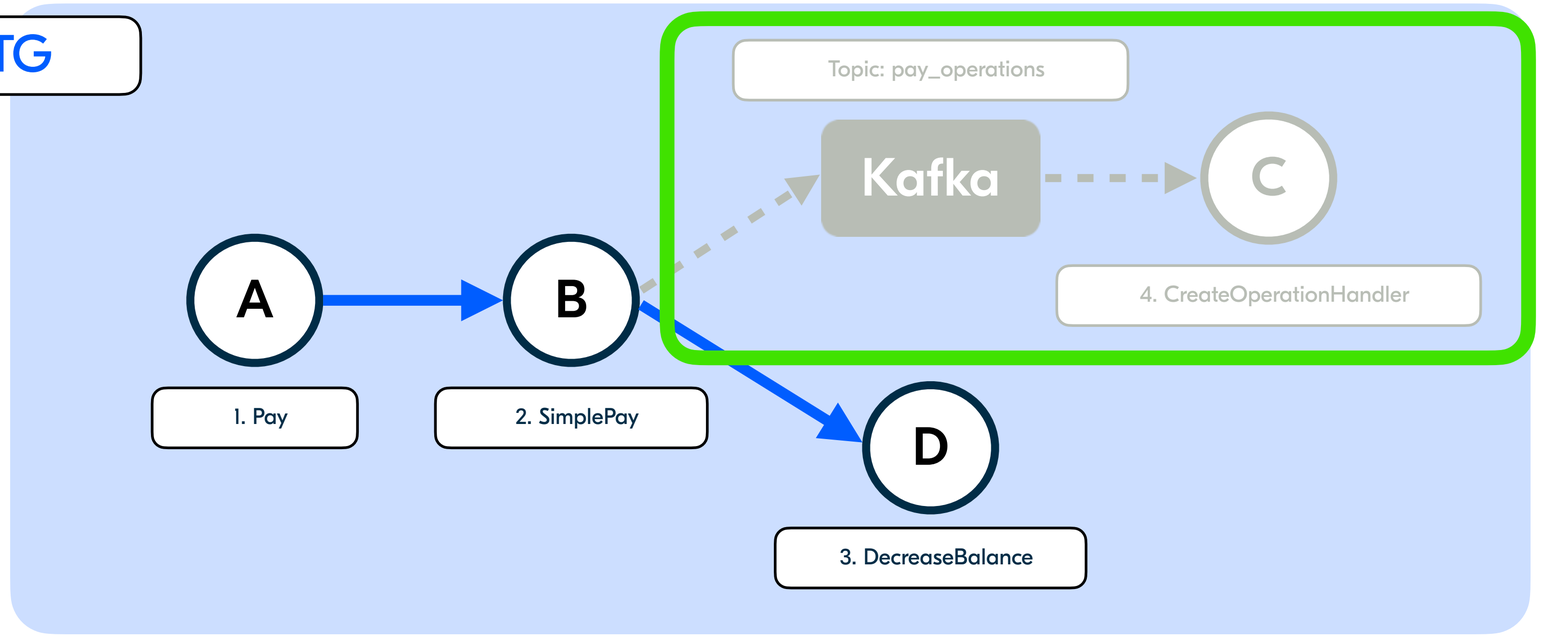
B. Построить граф межсервисного взаимодействия в рамках тестов

C. Посчитать покрытие сценариев с прода

PROD



STG



4 ребра

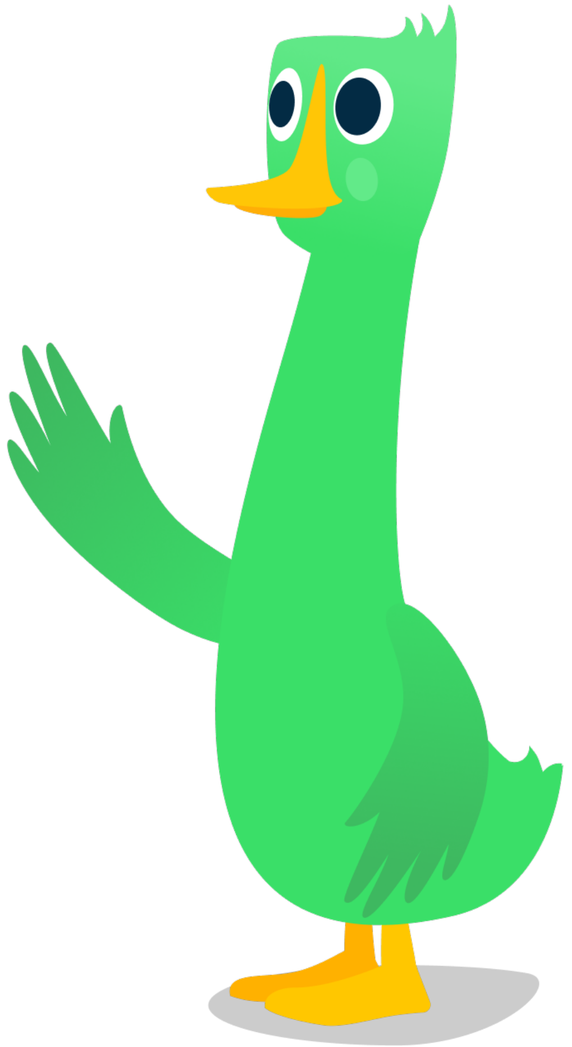
50% покрытие

2 ребра

Результаты

Параметры алгоритма

1. 10 - количество тестов на группу
2. Группировка по сценариям
3. Вес = количество сценариев теста + количество сервисов теста



Результаты при 10 тестах на сценарий

< 2000 тестов

В рамках любого запуска

ВАЖНО

+ 7%

на 60%

Запускаем меньше тестов (AVG)

ПРОЦЕНТ МИНИМИЗАЦИИ

+ 7%

27%

Для сервисов с < 500 тестами

- 2 минуты

~20 мин*

Экономия времени на прогонах крупных сервисов

- 1 минута

15 минут

Максимальный прогон тестов (90 персентиль)

ПРОЦЕНТ МИНИМИЗАЦИИ

+ 1%

69%

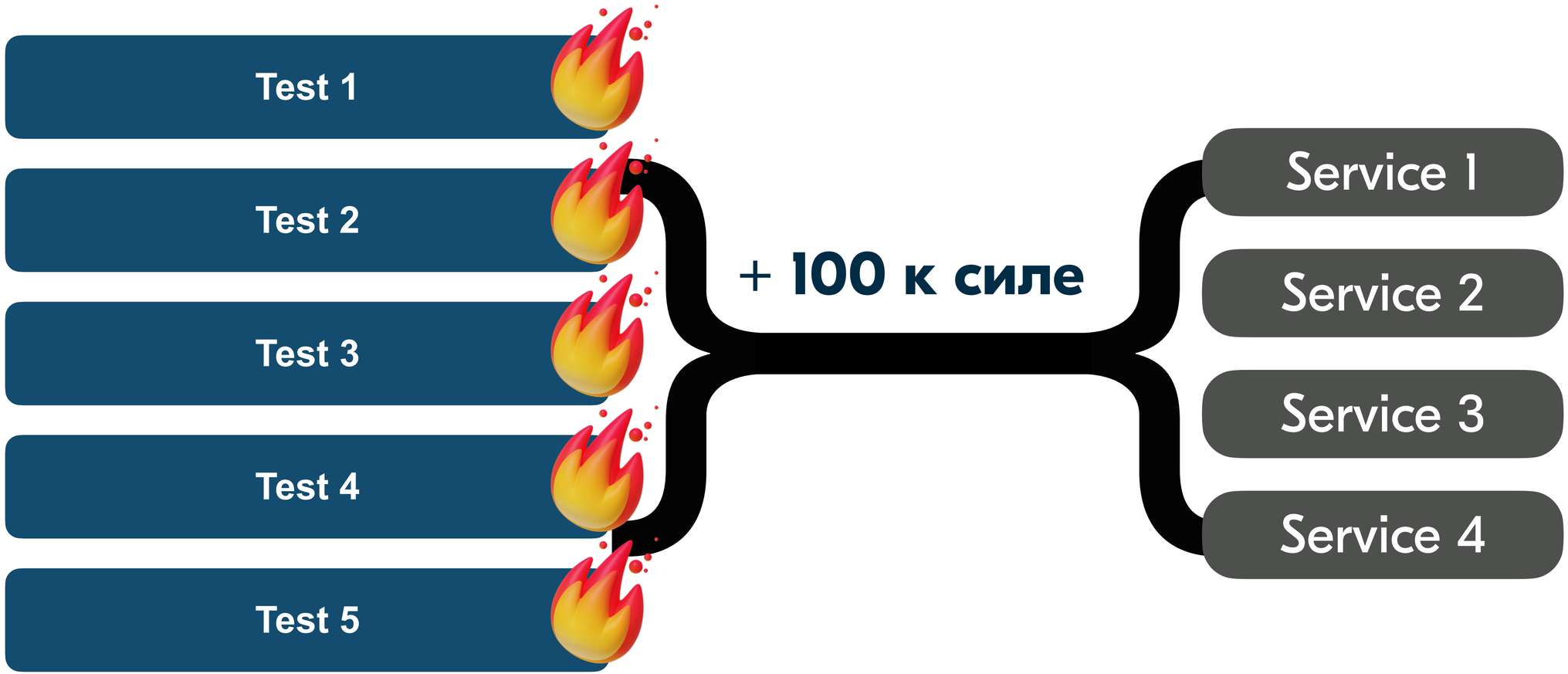
Для сервисов с >= 500 тестами

Но внутри сервисов также есть логика, которая очень важна

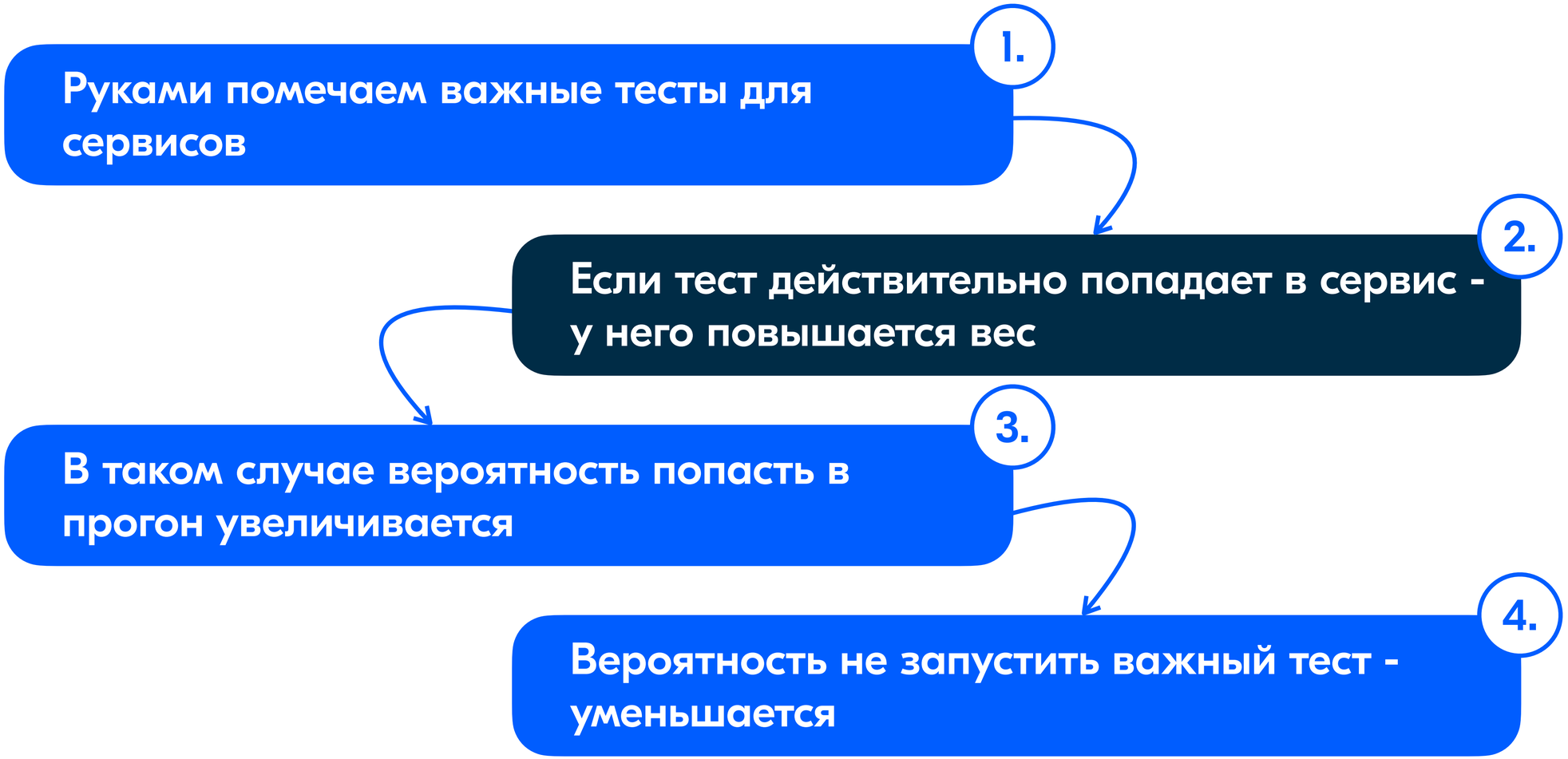
3.3

Ручная помощь алгоритмам минимизации

Помогаем алгоритму



Другими словами



А зОчем нам это усе, го просто руками разметим

Сравним

OZON tech

Размечаем руками

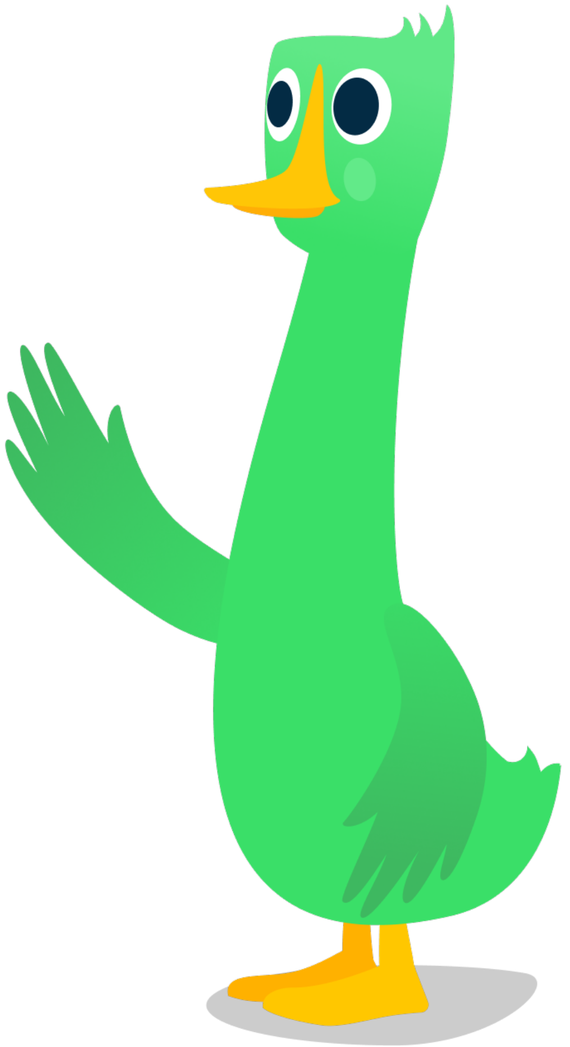
1. Не знаем, тестирует тест сервис или нет
2. Размечать придется очень много и с высокой точностью
3. Не знаем какие сценарии в рамках работы сервисов реально проверяются

Помогаем алгоритму

1. Тест запустится только в том случае, если тест тестирует сервис
2. Размечаем только точечные кейсы
3. Алгоритм сам подберет все межсервисные сценарии

Итого

1. Где боимся - руками размечаем тесты к сервисам
2. Тем самым уменьшаем риски
3. Но при этом не уходим в бесконечную разметку



4

Сравнение

Критерии сравнения

1. Группировка
2. Процент минимизации
3. Максимальное время прогона (90 персентиль)
4. Сложность реализации
5. Анализ работы теста

Сравнительная табличка

	Группировка	Процент минимизации	MAX время прогона	Сложность реализации	Анализ работы теста
По ручкам	По ручкам	**53 %**	**16**	1 чел/недели + 1 чел/месяц	В какие ручки/ сервисы попадают тесты
По сценариям	По сценариям	**60 %**	**15**	3 чел/недели + 1 чел/месяц	По каким путям ходят тесты

Решать только Вам, готовы ли вы считаться с возможными последствиями ради таких преимуществ или нет :)))

5

Статистика

Озон банк

OZON tech

700+
релизов в неделю

500+
сервисов

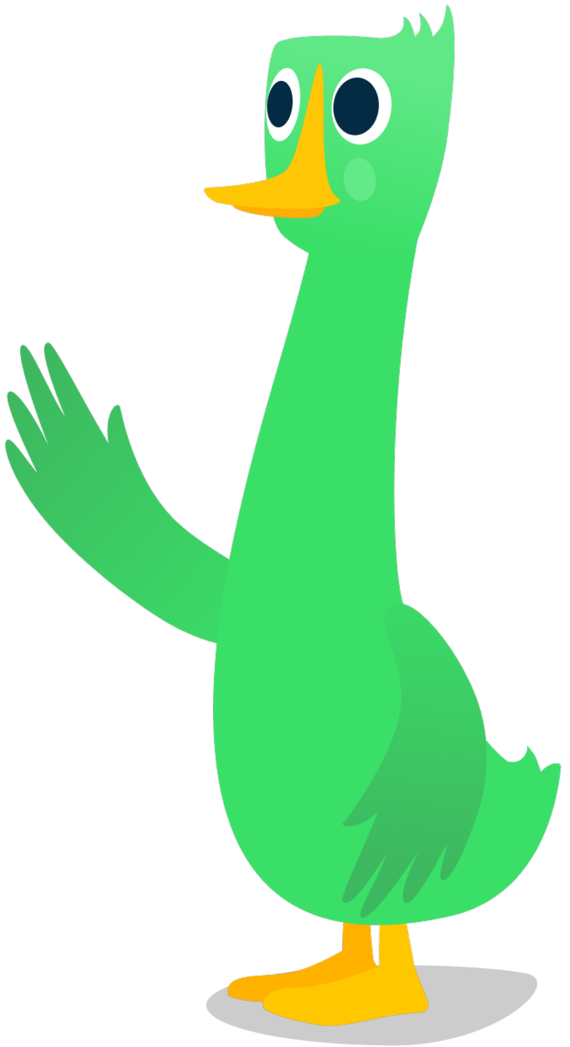
15 мин
максимальное время прогона для сервиса

2500+
прогонов в неделю

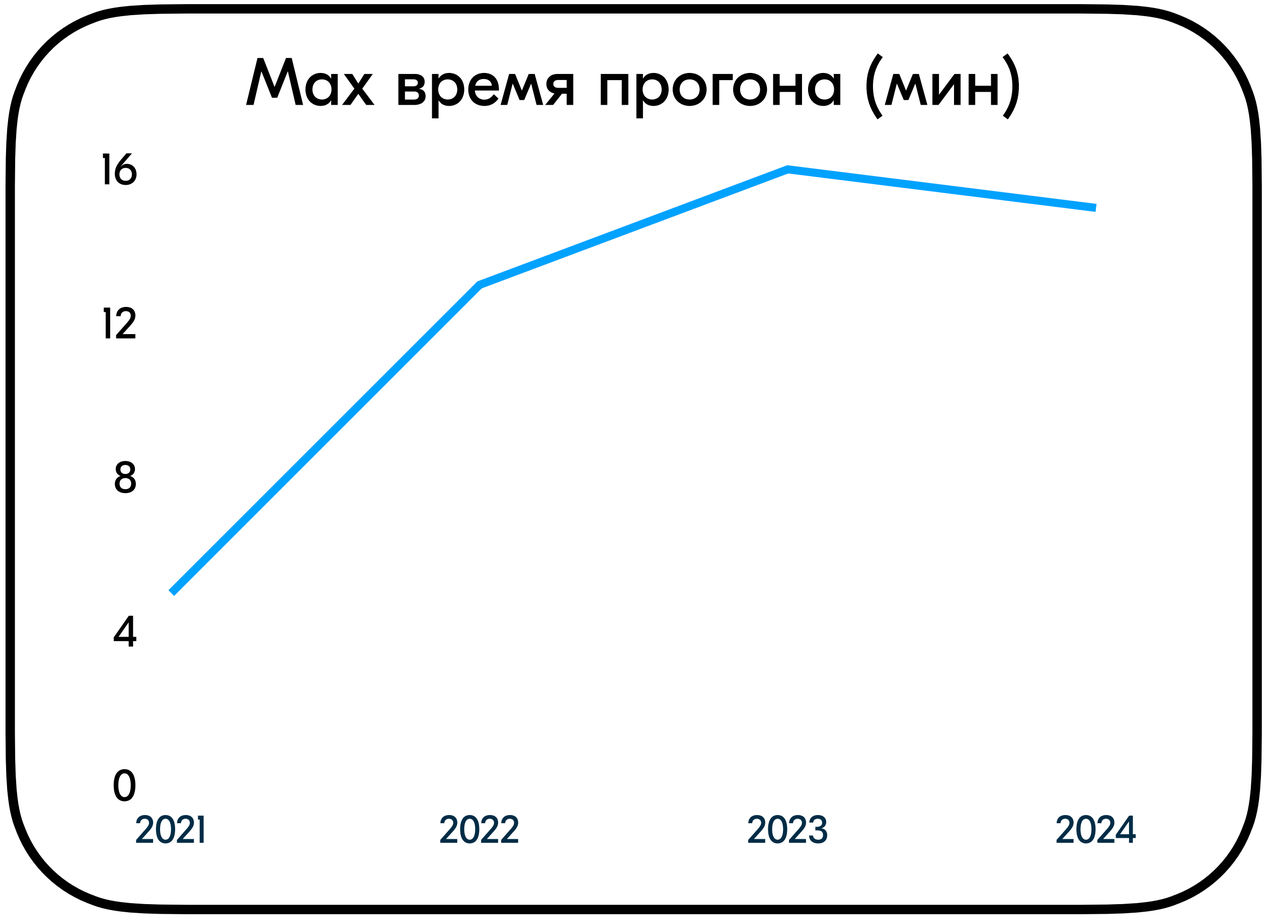
6500+
автотестов

Таймлайн

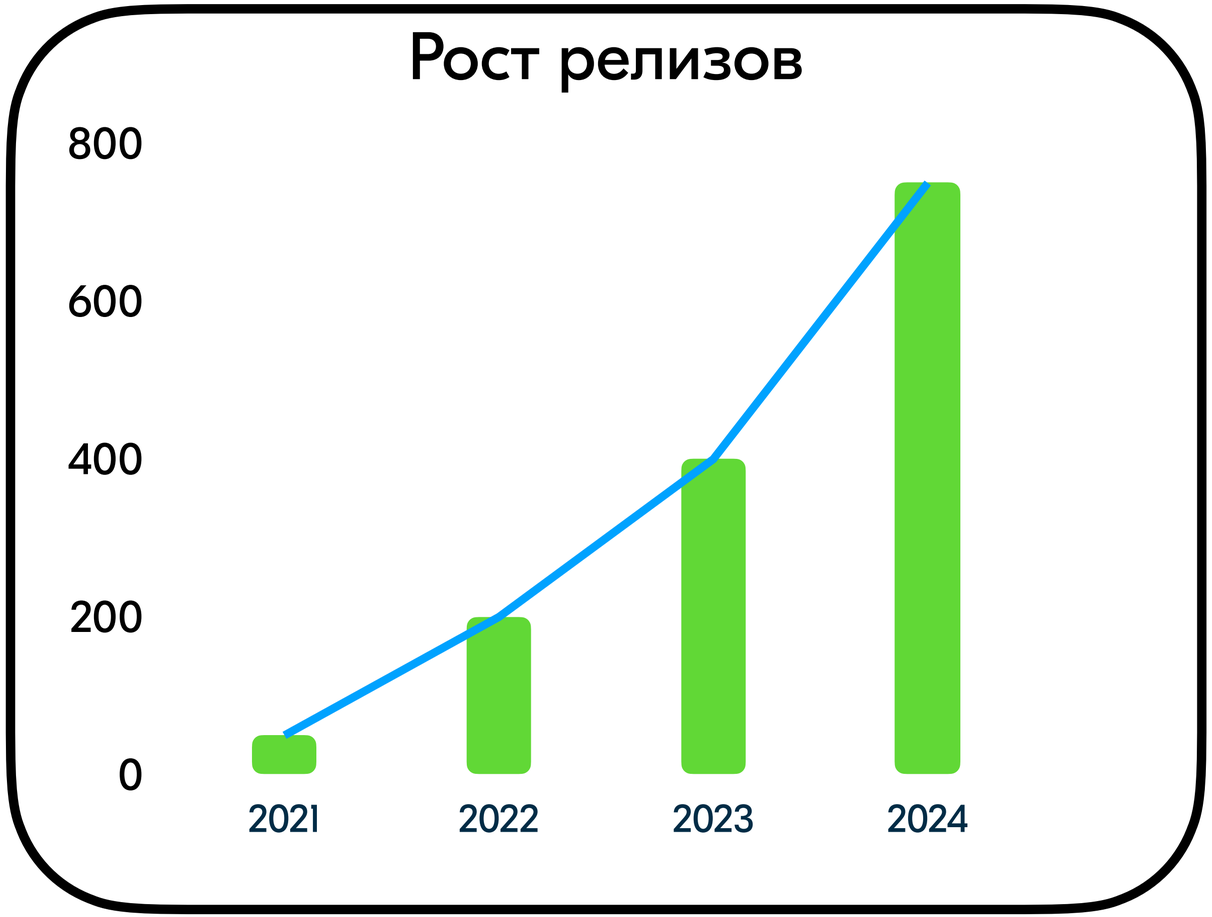
1. 2022 - автоматическая разметка
2. 2023 - минимизация по ручкам
3. 2024 - минимизация по сценариям

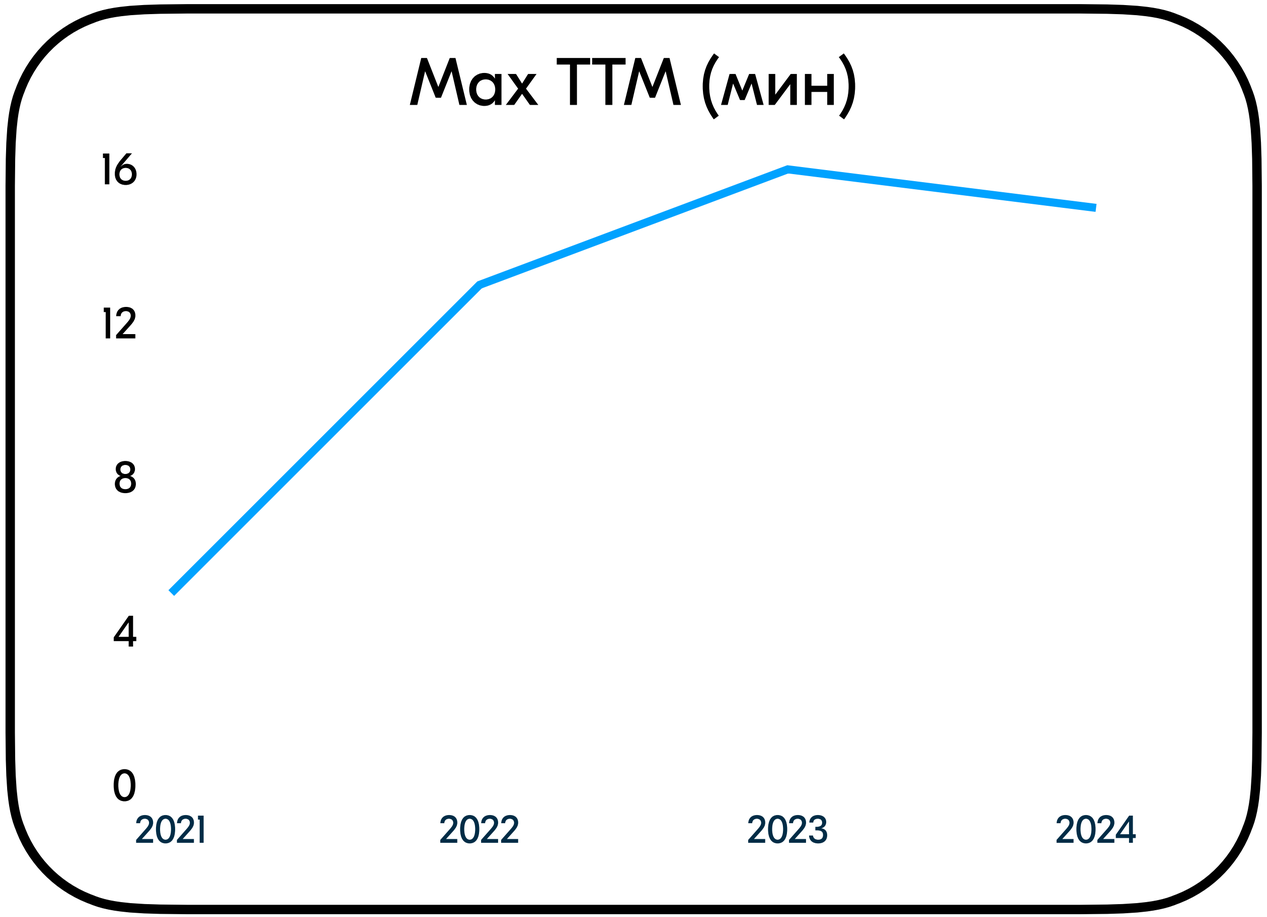


Тесты растут, а время - нет



Релизы растут, а время - нет






NOICE

6

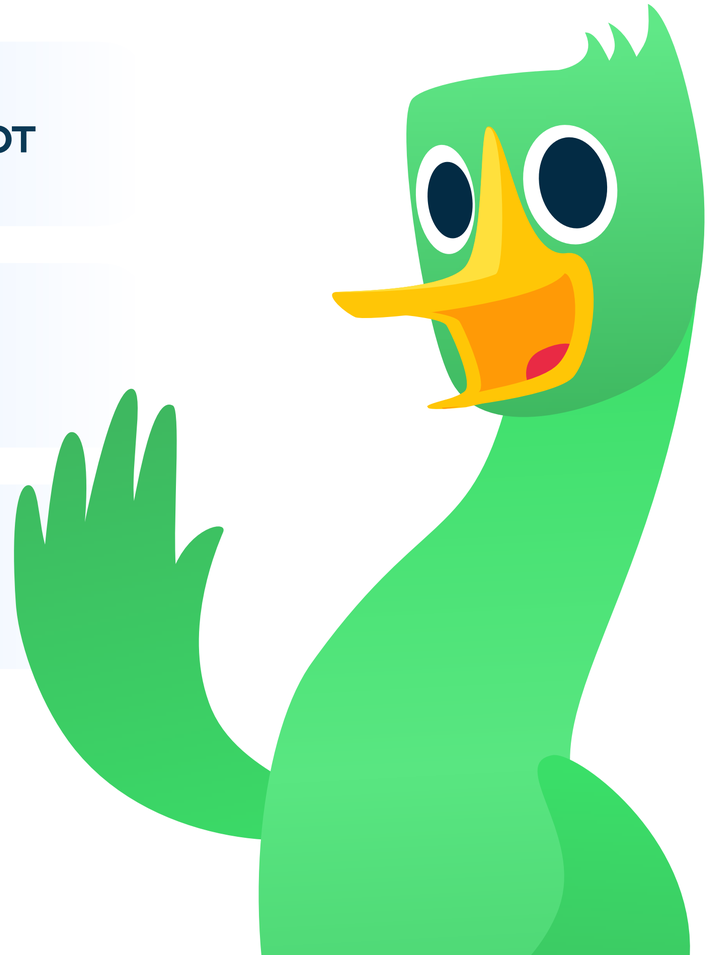
Выводы

Выводы

OZON tech

1. Когда нужно думать об автоматической минимизации регресса

- Хочется чаще релизиться и обычные методы не помогают
- Тестов и релизов становится слишком много
- НЕ хочется делать из регресса - ад



Выводы

2. К чему нужно быть готовым

- Разработка требует больших ресурсов
- У представленого подхода есть свои минусы
- Нужны люди, которые смогут такое накалякать

Выводы

OZON tech

3. Как реализовать такое у себя?

- Посмотреть предыдущий доклад
- Разобраться в перечисленных тут алгоритмах
- Выделить время и ресурсы на кодинг сервиса

Выводы

OZON tech

4. При каких условиях стоит пилить такое

- У вас уже есть трейсинг в компании
- У вас микросервисная архитектура
- Все другие подходы уже не приносят результат

Команда Test Impact Analysis

Романов
Алексей

Зенков
Кирилл

Кузейкин
Николай

Тараканов
Никита

Пендрак
Олег

EX

Данилов
Илья

Выступления Озон Банка

ozontech

Спасибо за внимание

Алексей Романов, руководитель группы Test Impact Analysis в Ozon Банке

lr.romanov1999@mail.ru

@lexcorp

Алексей Романов

@lexcorp