Хаос-тестирование как часть функционального: наш опыт в Яндекс Маркете

Яндекс Маркет

Светлана Баканова
Руководитель группы тестирования

О чём пойдёт речь

1. Разберёмся с **концепцией** хаос-инжиниринга
2. Рассмотрим **ручные подходы** к эмуляции проблем в сервисах
3. Рассмотрим наш пример, как может выглядеть **процесс регулярного хаос-тестирования**
4. Расскажу про наш **опыт автоматизации**
5. Поделюсь **рецептами**, как реализовать или с какой стороны подойти к хаос-тестированию у себя

Немного истории

и пару слов о концепции

Back to 2000x

01 Рост крупномасштабных распределённых систем

02 Облачные вычисления

03 Развитие веба

04 Микросервисы

Энтропия привела к хаосу...


Mastering Chaos
A Netflix Guide to Microservices
Josh Evans – Engineering Leader
November 8, 2016
QCon
SAN FRANCISCO
Filmed at
QCon San Fr
Brought to yo
InfoQ

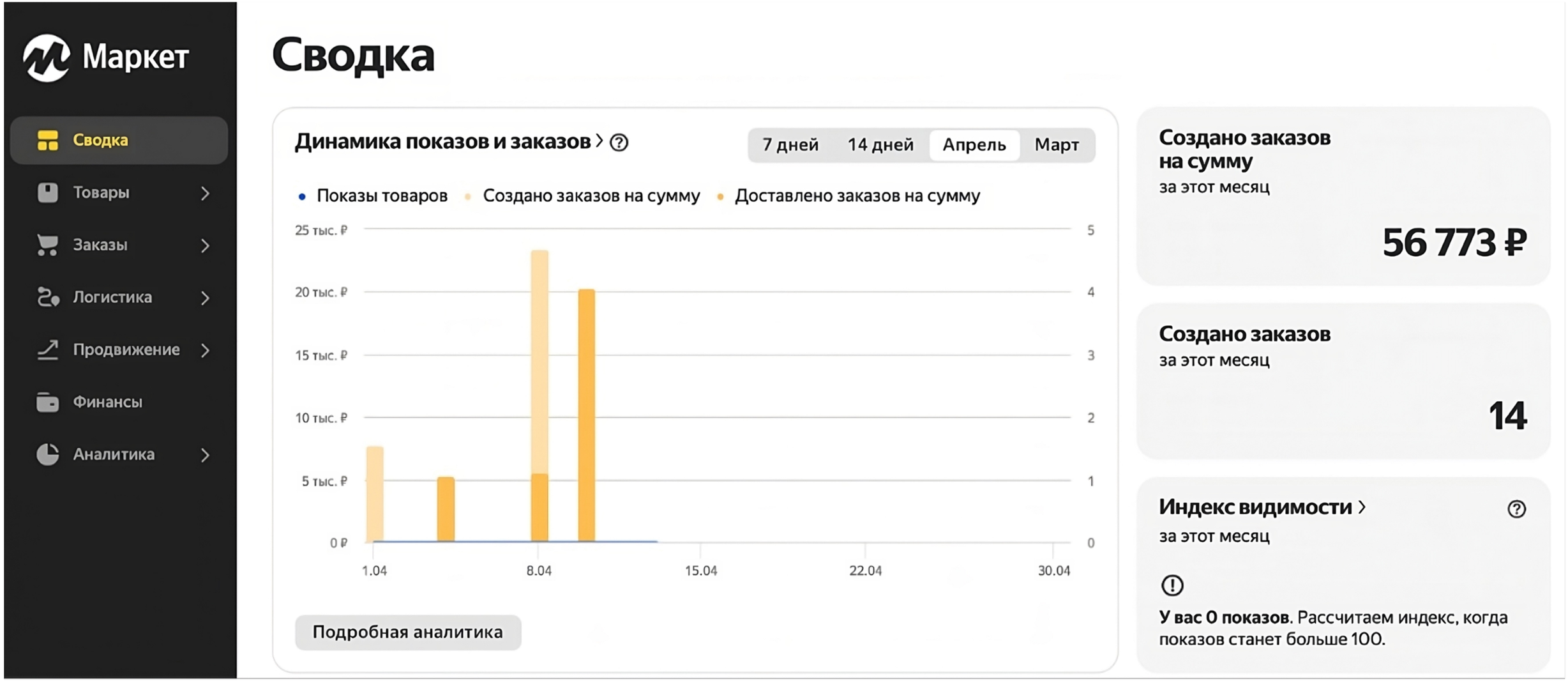

NFX303
The evolution of chaos engineering at Netflix
Aleš Plšek (he/him)
Senior Software Engineer, Resilience
Netflix
Rob Hilton
Principal Solutions Architect
AWS

Что же такое хаос-тестирование?

— симуляция проблем или сбоев внутри системы с целью определить её «слабые места» и повысить отказоустойчивость

Про наш опыт

Мы делаем B2B сервис Яндекс Маркета



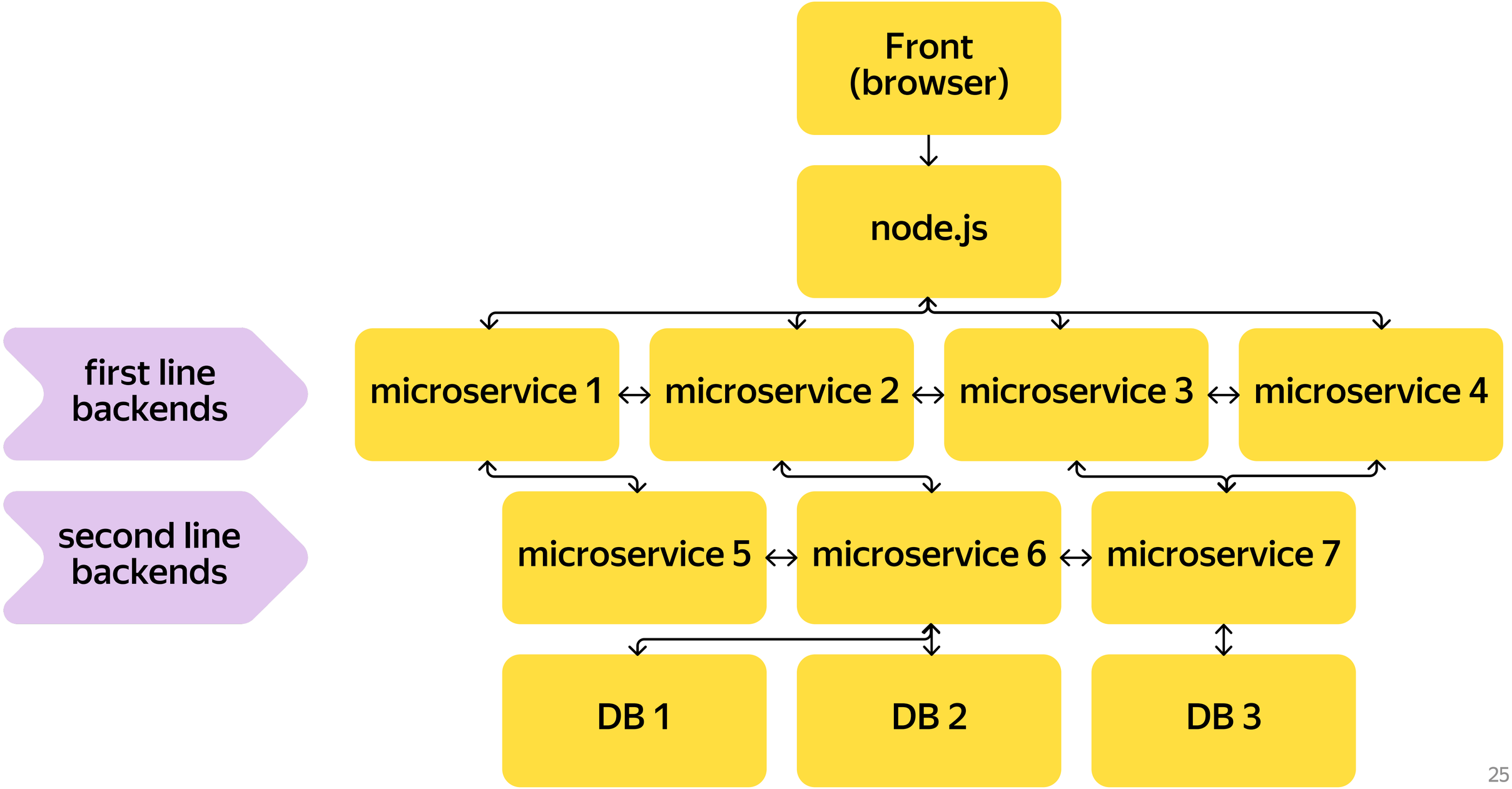
Знакомьтесь — это пёс

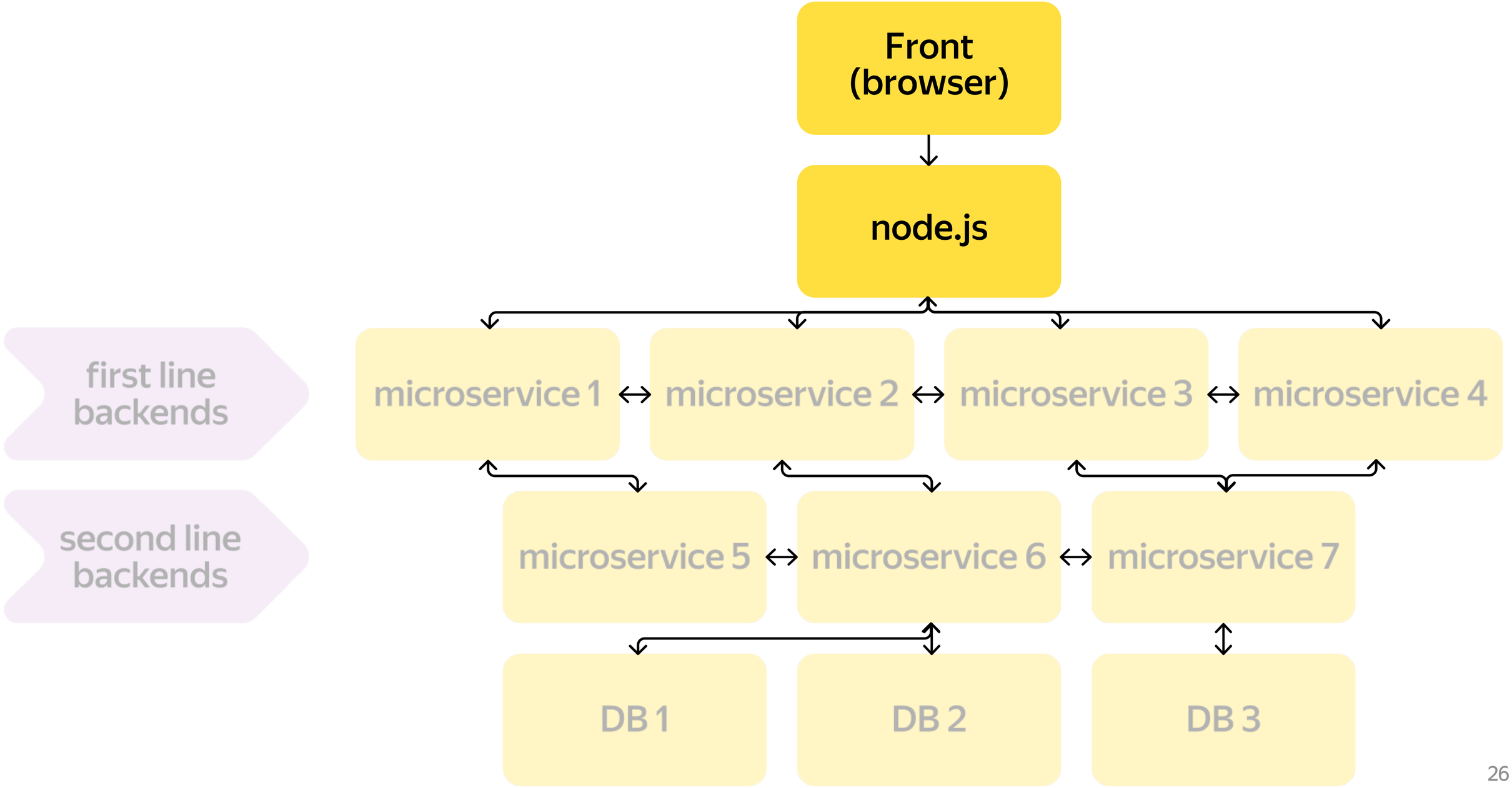
Не удалось загрузить данные

Попробуйте обновить страницу

Обновить

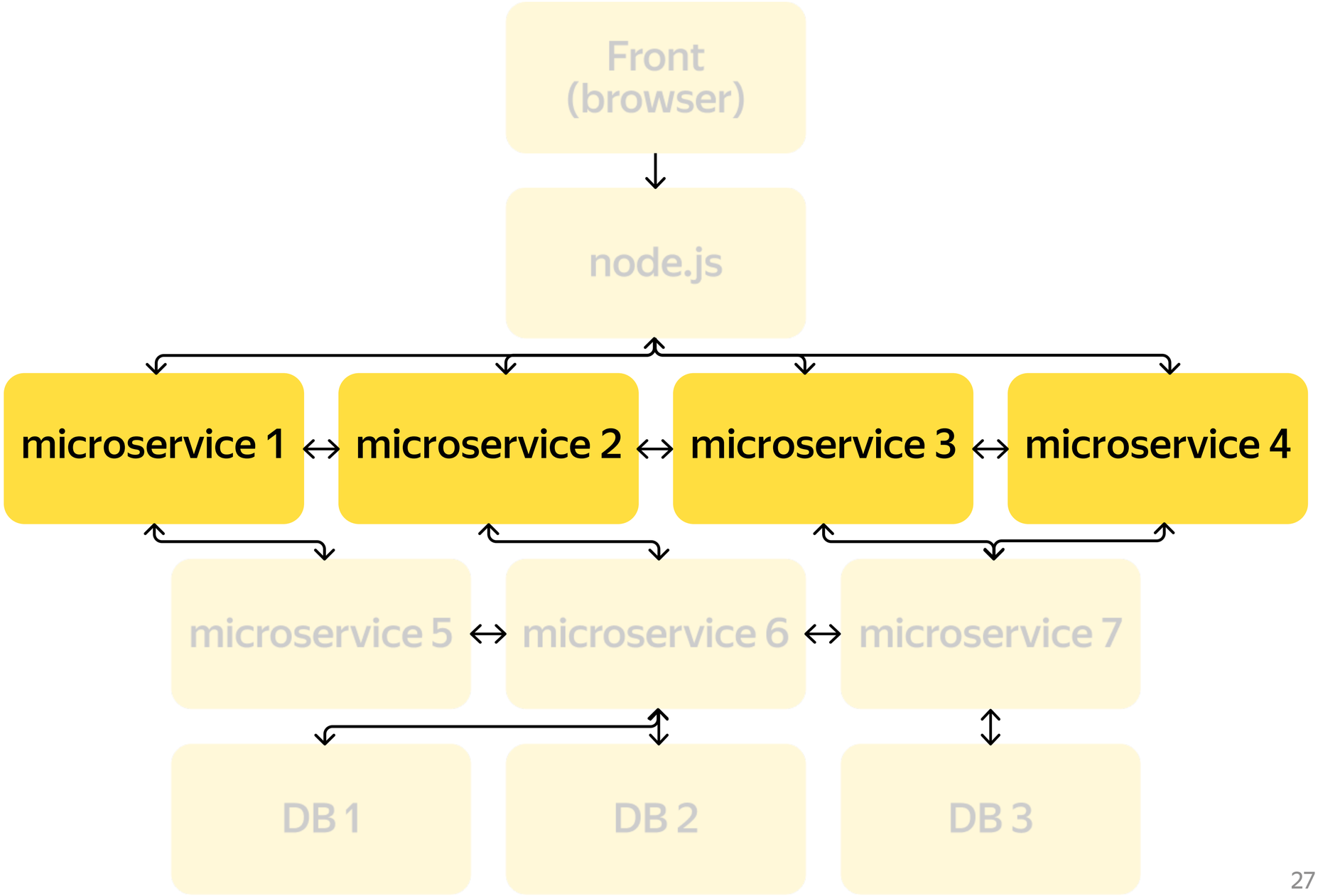
А что под капотом?

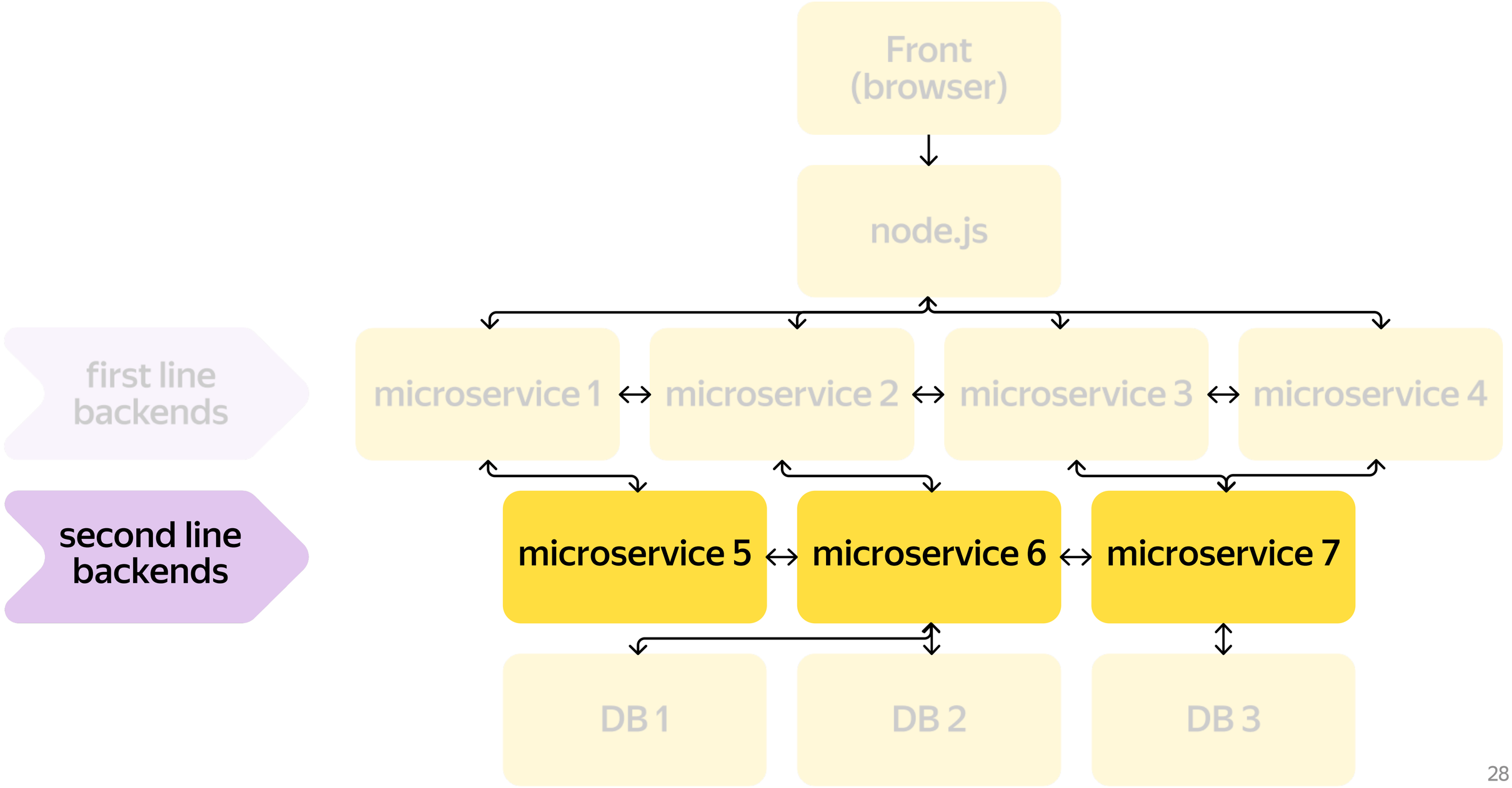

Front
(browser)
node.js
microservice 1
microservice 2
microservice 3
microservice 4
microservice 5
microservice 6
microservice 7
DB 1
DB 2
DB 3
first line
backends
second line
backends

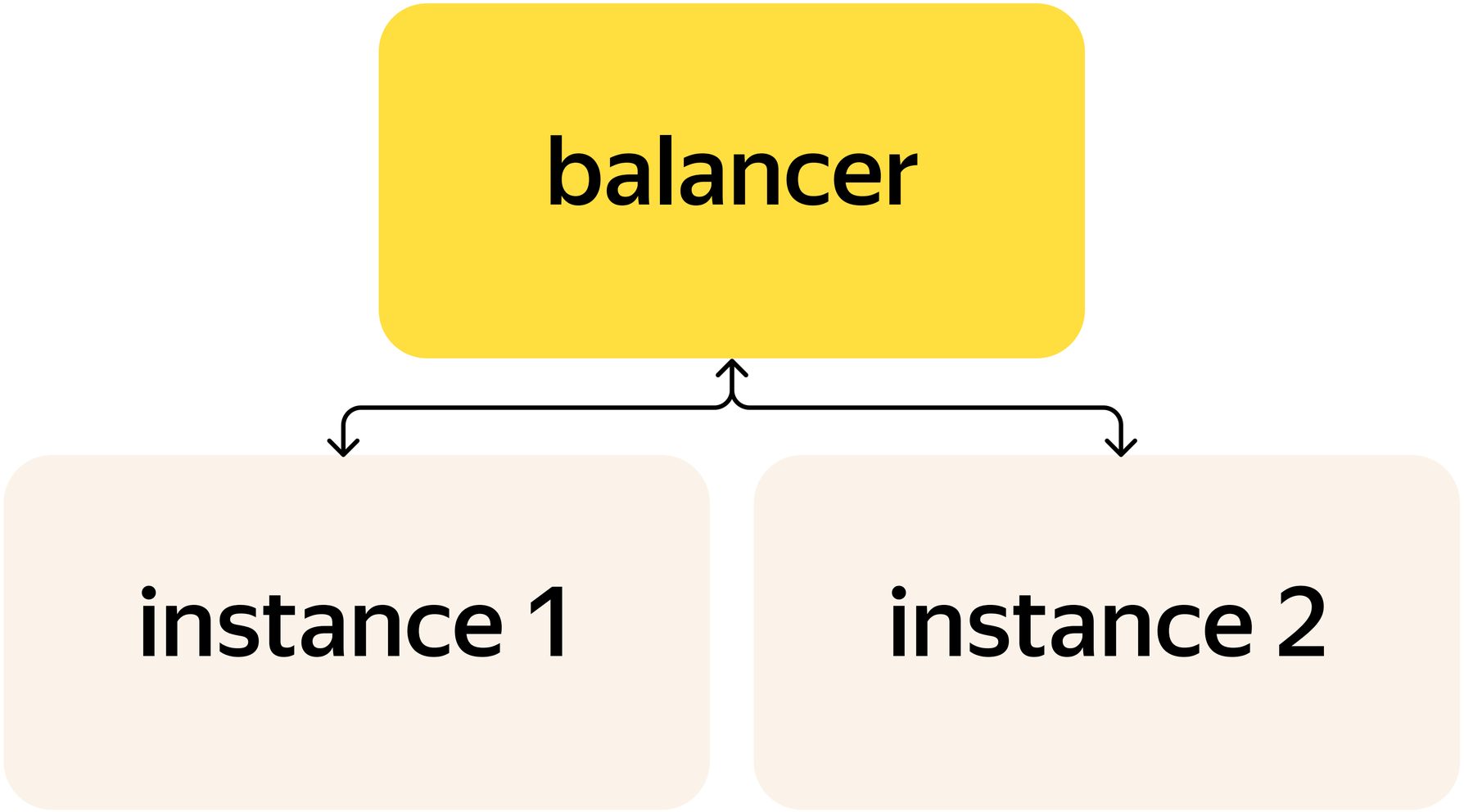

Front
(browser)
node.js
first line
backends
microservice 1
microservice 2
microservice 3
microservice 4
second line
backends
microservice 5
microservice 6
microservice 7
DB 1
DB 2
DB 3

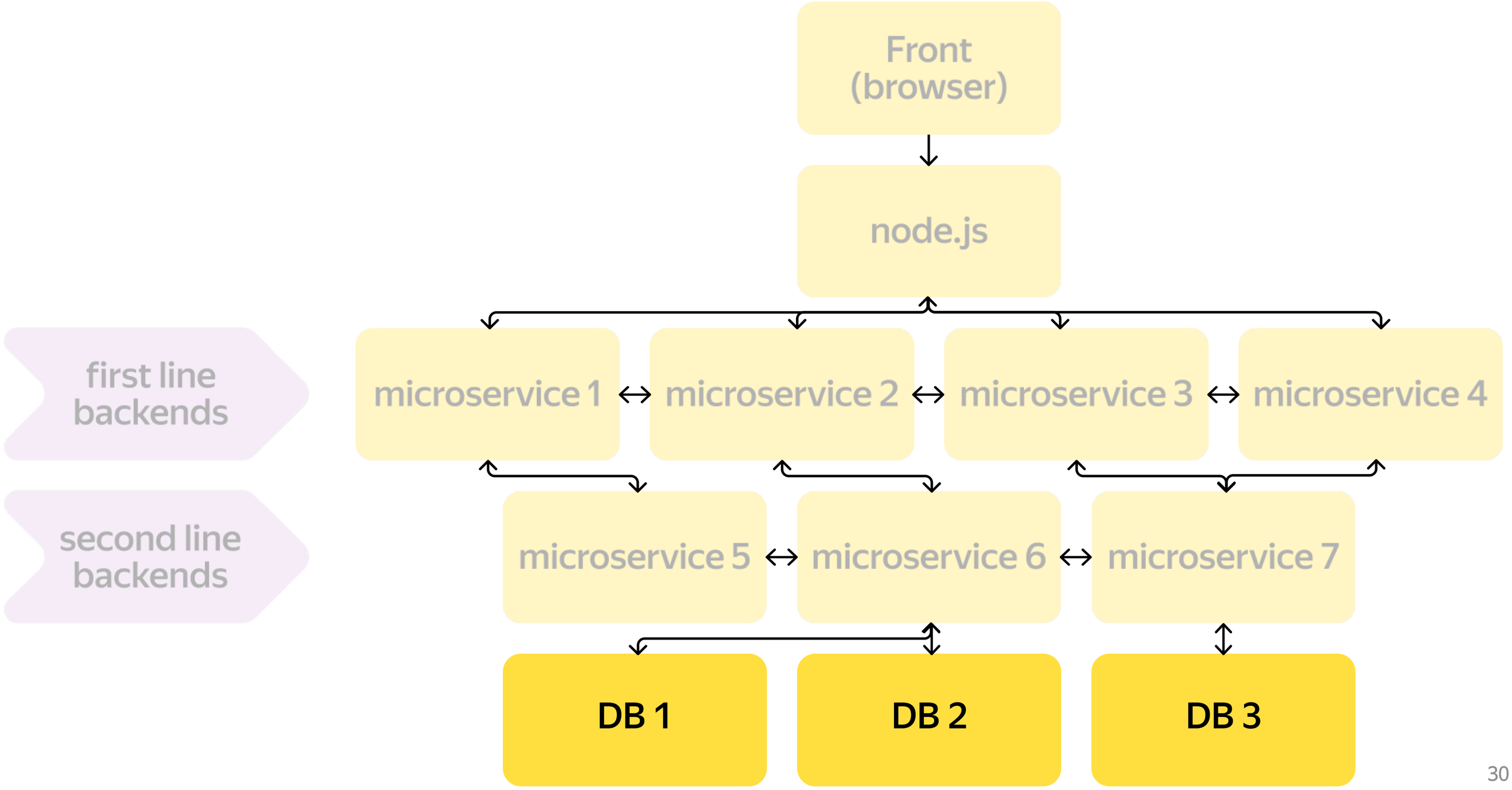
first line backends

second line backends



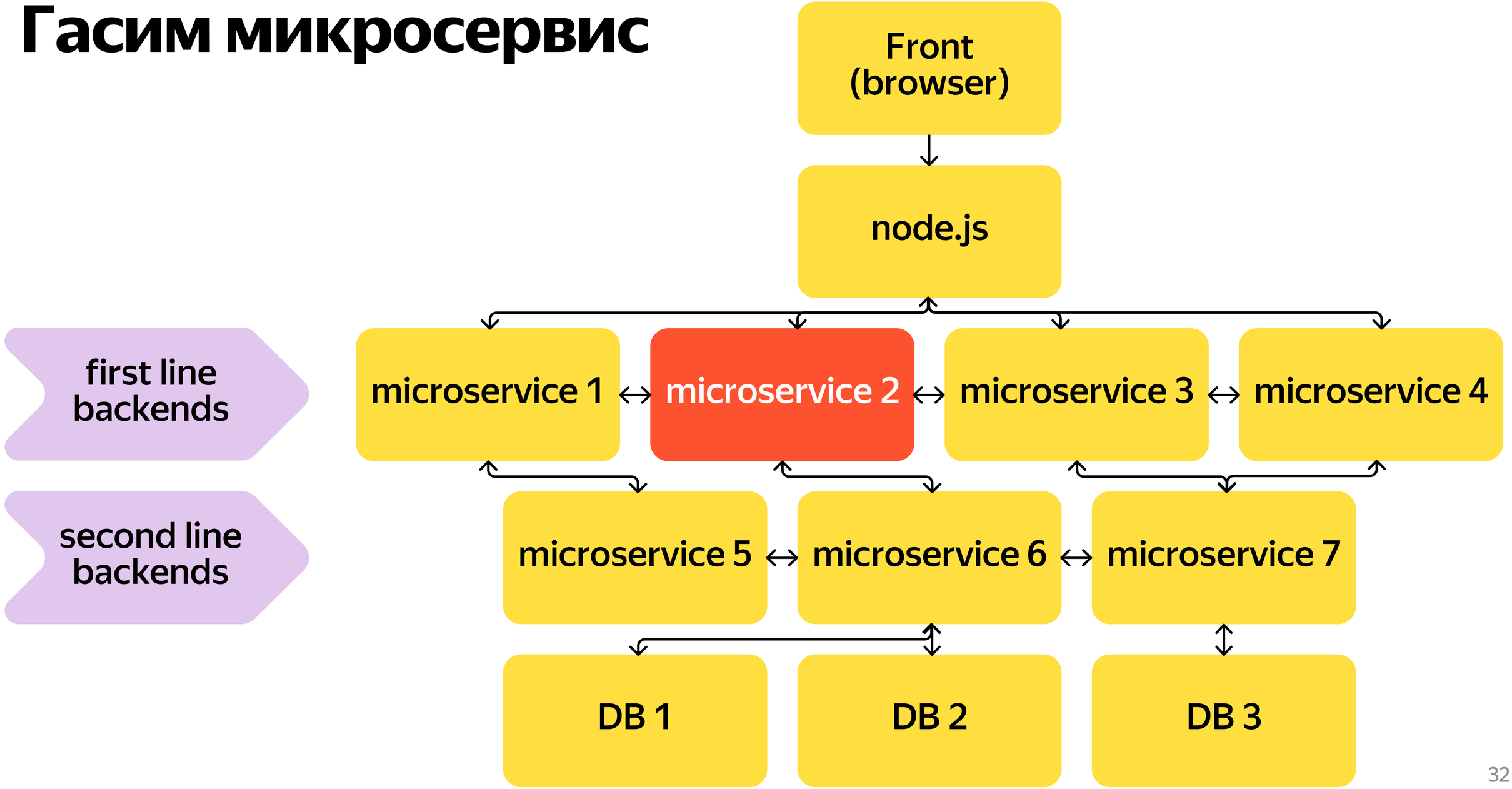

Front
(browser)
node.js
first line
backends
microservice 1
microservice 2
microservice 3
microservice 4
second line
backends
microservice 5
microservice 6
microservice 7
DB 1
DB 2
DB 3


balancer
instance 1
instance 2


Front
(browser)
node.js
first line
backends
microservice 1
microservice 2
microservice 3
microservice 4
second line
backends
microservice 5
microservice 6
microservice 7
DB 1
DB 2
DB 3

Делай раз
или первый подход к снаряду

Гасим микросервис



Процесс — как он устроен

Цикл тестирования

01 Выбираем бэкенд

02 Назначаем время учений, предупреждаем команды

03 Гасим балансер в тестинге

04 Запускаем АТ и ручные проверки

05 **Анализируем результаты**

Выводы после первых ручных манипуляций

01 Proof of concept — показали возможность применения хаос-тестирования

02 Обкатали процесс

03 Хотим уметь пятисотить и таймаутить

04 Нужна автоматизация

Плюсы и минусы ручного выключения

Эмулируем «падение» микросервиса в интеграционной среде — режем «на живую»

Максимально просто для реализации — тык! и всё

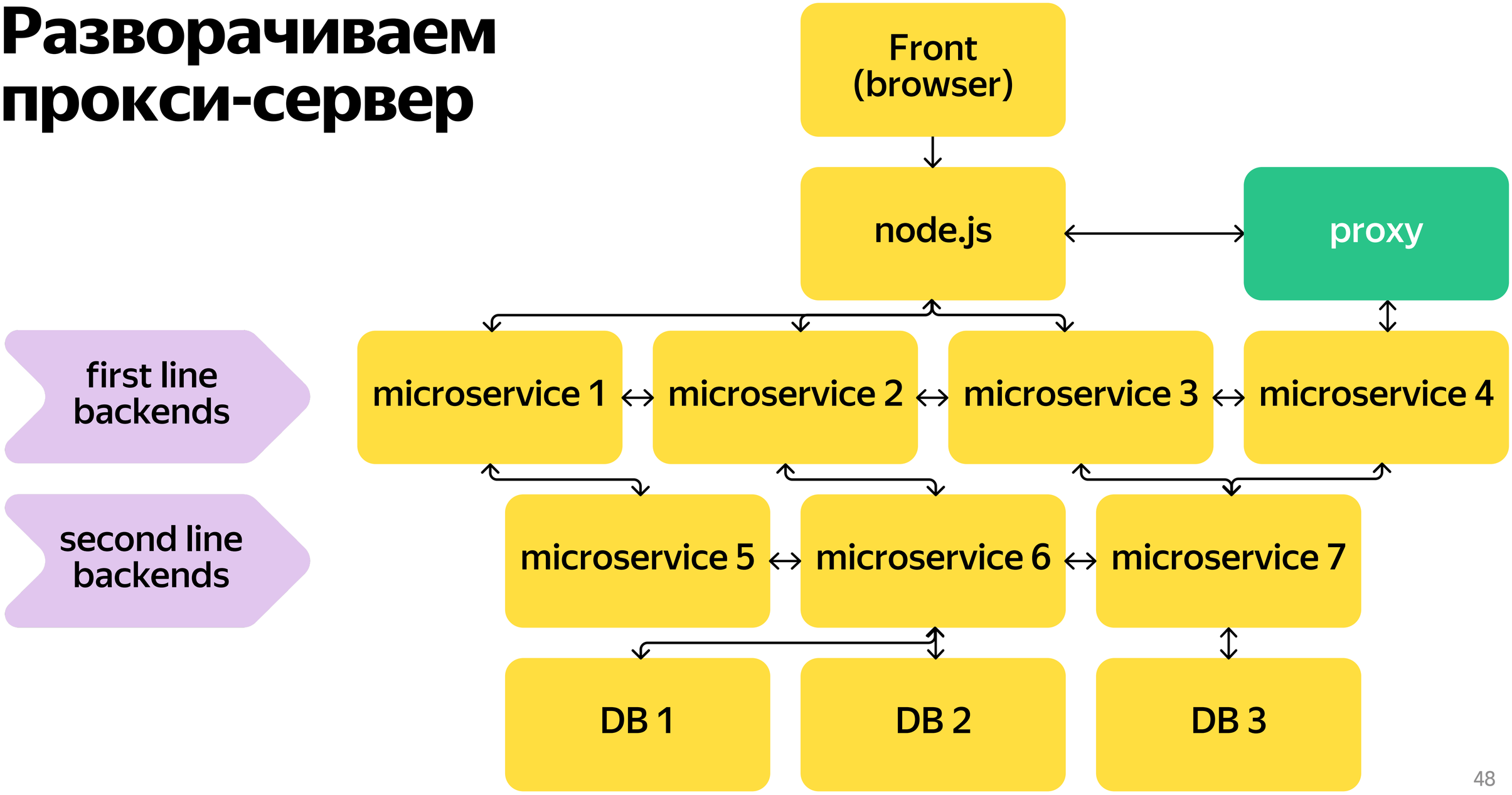
Нет возможности проверить таймауты

Ручные манипуляции в инфраструктуре

Мы проложили маршрут

Делай два — второй подход

Разворачиваем прокси-сервер



Рецепт

01 Разворачиваем nginx

```
sudo systemctl start nginx
```

02 Добавляем echo-модуль

Рецепт

03 Конфигурируем echo-module

```
sudo vim /etc/nginx-proxy/conf/nginx.conf

        location / {
            echo_blocking_sleep 5.0;  # для таймаута
            echo_exec @from_proxy_to_real_banalcer; # для редиректа после таймаута
            # ИЛИ
            echo_status 500; # для 500-ки
            echo “Chaos-monkey. Go to sbakanova”; # для ответа после 500-ки
        }
        location @from_proxy_to_real_balancer {
            proxy_pass http://some-backend.tst.net;
        }
```

Перезапускаем сервер `sudo systemctl restart nginx-proxy`

04 В отдельной инсталляции фронтенд приложения меняем url бекенда на адрес прокси-сервера

Плюсы и минусы подхода с nginx-echo

Эмулируем ошибки и таймауты

Изолируемся от интеграционной тестовой среды — не мешаем работать коллегам

Подходит для эмуляции проблем от сервисов, к которым мы не имеем доступ

Синтетика — не проверяем сценарии взаимодействия между микросервисами

Много ручных приседаний

Процесс — что мы считаем багами деградации

Подробнее про то, что мы считаем багами деградации

01 Является ли бэкенд основным для страницы

02 Ищем оптимальный вариант деградировать красиво — отрисовать блок без данных, но с кнопкой перезагрузки

03 В каждом фиксе мы добавляем обработчик ошибок, чтобы выводить график деградаций

Маркет

- Сводка
- Товары
- Заказы
- Логистика
- Продвижение
- Финансы
- Аналитика

Индекс цен

Не рассчитан

Рассчитаем, когда у вас появятся показы и продажи, а также конкуренты с таким же товаром — на Маркете или других площадках.

Смотреть детали

Полки

Прокачайте продажи: разместите товары там, где покупатели их заметят

Хочу больше продаж

Каталог

Всего товаров	134
Есть проблемы	27% (36)
Готово	73% (98)

Индекс качества

Ничего не нашлось

Обновить

Маркет

- Сводка
- Товары
- Заказы
- Логистика
- Продвижение
- Финансы
- Аналитика

Сводка

Динамика показов и заказов

Ничего не нашлось

Обновить

Создано заказов на сумму

Ничего не нашлось

Обновить

Создано заказов

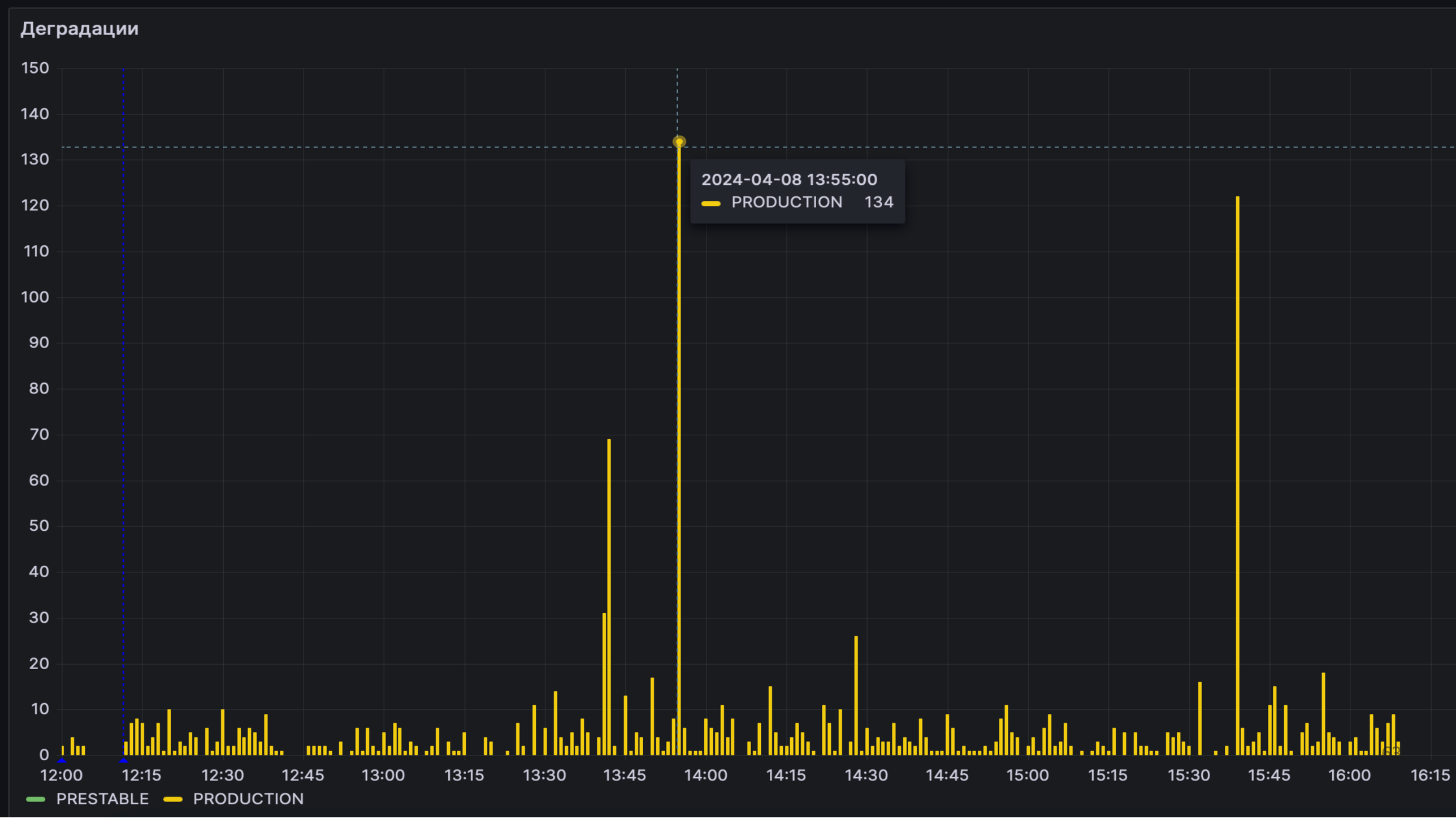
Ничего не нашлось

Обновить

Индекс видимости

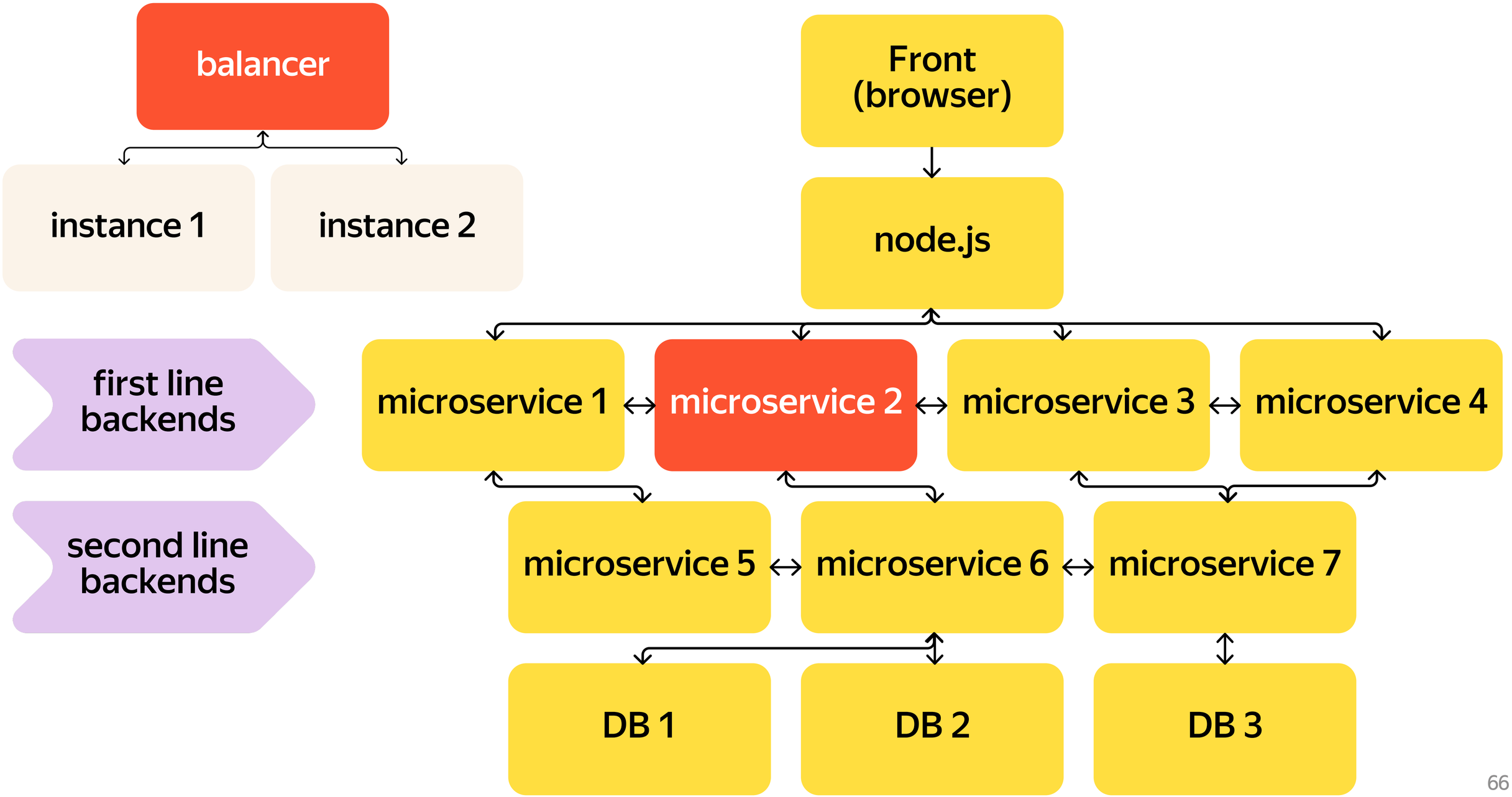
за этот месяц

У вас 0 показов. Рассчитаем индекс, когда показов станет больше 100.


Деградации
150
140
130
120
110
100
90
80
70
60
50
40
30
20
10
0
2024-04-08 13:55:00
PRODUCTION 134
12:00
12:15
12:30
12:45
13:00
13:15
13:30
13:45
14:00
14:15
14:30
14:45
15:00
15:15
15:30
15:45
16:00
16:15
PRESTABLE
PRODUCTION

Мы запустили процесс

Делай три — автоматизация цикла тестирования


balancer
instance 1
instance 2
Front
(browser)
node.js
first line
backends
second line
backends
microservice 1
microservice 2
microservice 3
microservice 4
microservice 5
microservice 6
microservice 7
DB 1
DB 2
DB 3

Собрали центр управления хаосом

1. Инфраструктурная магия: настройка lua для nginx + memcached для хранения настроек
2. Умеем эмулировать любой код ошибки и таймауты. Даже на процент входящих запосов
3. Пайплайн запуска в сервисе пайплайнов Маркета

Параметры

- микросервис
- режим хаоса (ошибки или таймаут)
- время работы

→ Запуск магии — в микросервисе включается настройка хаоса — сервис начинает пятисотить/таймаутить

→ Запуск автотестов

→ Завершение работы в режиме хаоса, удаление настройки из микросервиса

Плюсы и минусы подхода

Автоматизированный запуск цикла тестирования — включаем хаос-режим для бэкенда и запускаем автотесты

Максимально приближенный к реальности режим

Можно запускать по крону — хоть ночью, хоть на выходных

Проходить весь пул бэкендов долго

На этом этапе мы стали анализировать накопленный опыт

01 Нам интересны проблемы от бэкендов первой линии

02 Проблемами между бэкендами мы можем пренебречь

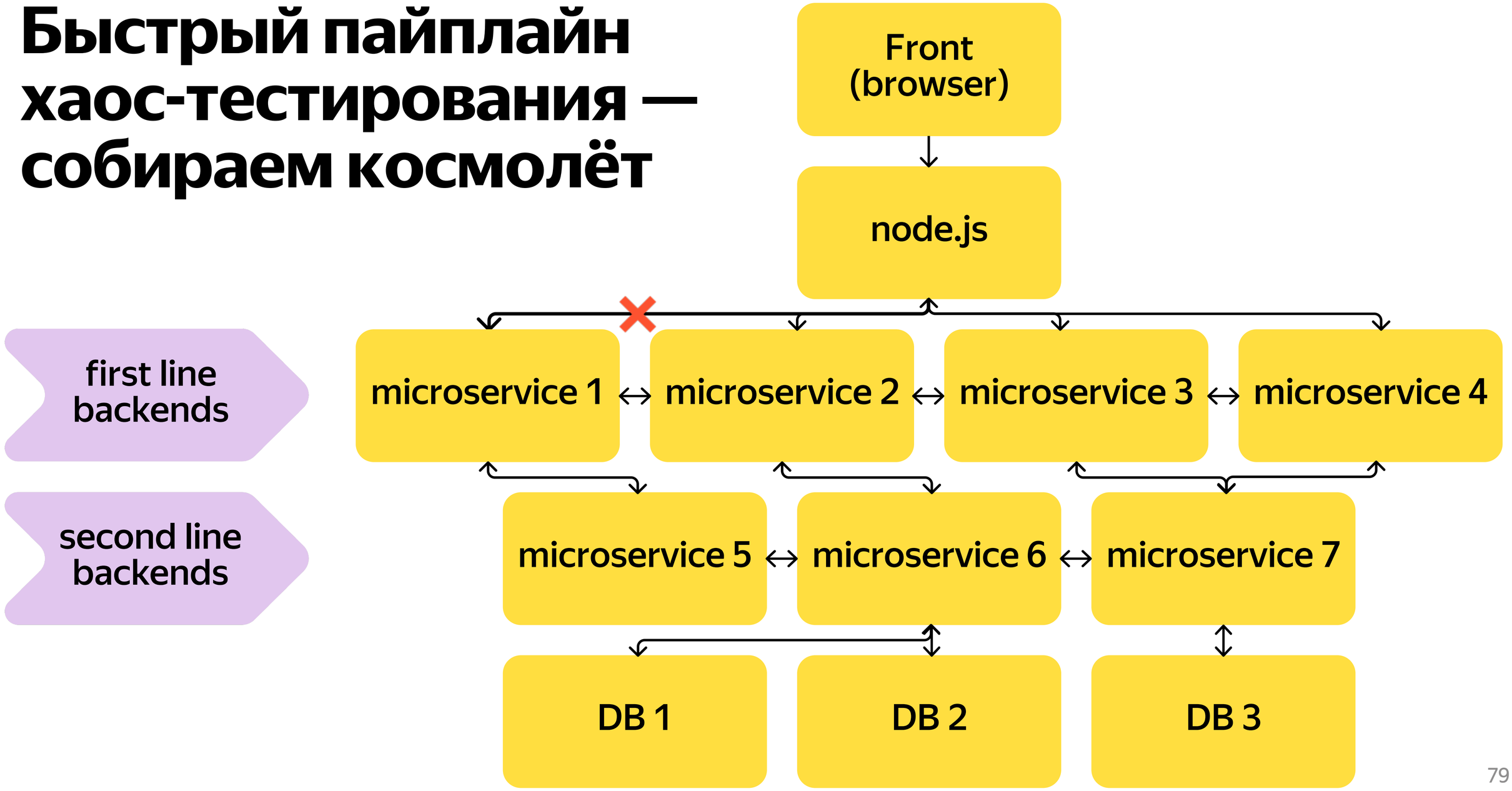
03 Весь цикл проходить долго (при нашей скорости около 2 лет)

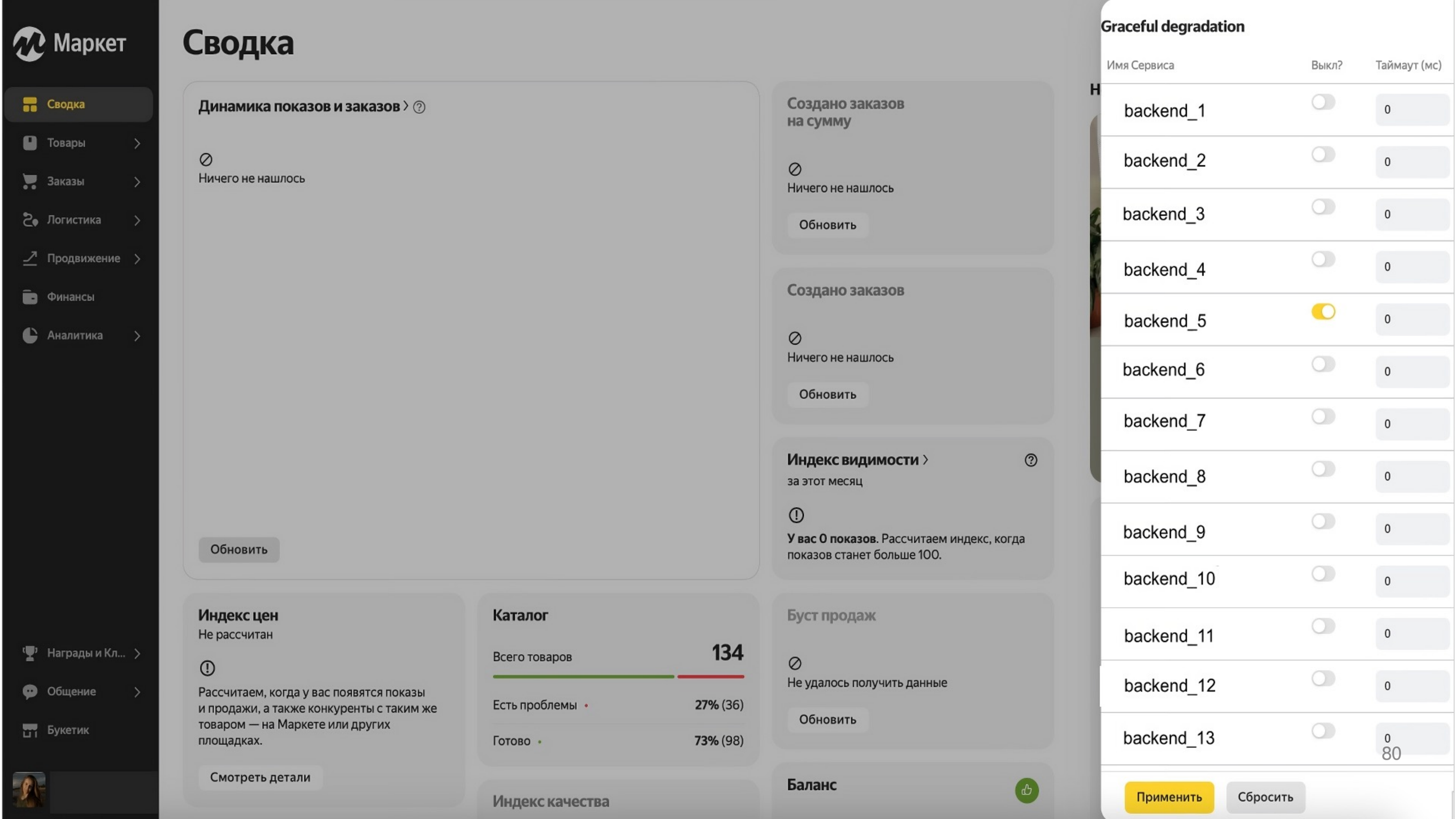
Вывод — хотим быстрое решение для регресса

Мы автоматизировали процесс

Делай четыре — фронтовая магия вне Хогвартса

Быстрый пайплайн хаос-тестирования — собираем космолёт




Маркет
Сводка
Товары
Заказы
Логистика
Продвижение
Финансы
Аналитика
Награды и Кл...
Общение
Букетик
Сводка
Динамика показов и заказов
Ничего не нашлось
Обновить
Создано заказов на сумму
Ничего не нашлось
Обновить
Создано заказов
Ничего не нашлось
Обновить
Индекс видимости
за этот месяц
У вас 0 показов. Рассчитаем индекс, когда показов станет больше 100.
Индекс цен
Не рассчитан
Рассчитаем, когда у вас появятся показы и продажи, а также конкуренты с таким же товаром — на Маркете или других площадках.
Смотреть детали
Каталог
Всего товаров 134
Есть проблемы 27% (36)
Готово 73% (98)
Буст продаж
Не удалось получить данные
Обновить
Индекс качества
Баланс
Graceful degradation
Имя Сервиса
Выкл?
Таймаут (мс)
backend_1 0
backend_2 0
backend_3 0
backend_4 0
backend_5 0
backend_6 0
backend_7 0
backend_8 0
backend_9 0
backend_10 0
backend_11 0
backend_12 0
backend_13 0
Применить
Сбросить
80

Graceful degradation tool — как сделан?

01 В каждый запрос с фронта отправляется кука

02 Все запросы с node.js приложения идут через ~~одно~~ единое место в коде, в котором выставлена проверка наличия во входящем запросе куки

03 Если в запросе есть shutdown_cookie, в бэкенд запрос не уйдет, node.js вернёт на фронт 500-ку — profit!

```
if (shutdownCookieString) {
    let cookieObject;
    try {
        cookieObject = JSON.parse(shutdownCookieString);
    } catch (e) {
        cookieObject = {};
    }
    const timeout = Number(cookieObject[backend]?.timeout);
    if (timeout) {
        await pause(timeout);
    }
    const shutdown = cookieObject?.[backend]?.shutdown;
    if (shutdown) {
        throw new Error(`Backend “${backend}” is switched off by “${SERVICE_SHUTDOWN_COOKIE}”cookie`);
    }
}
```

Graceful degradation tool - что умеет

01 Можно выключить хоть все бэкенды сразу

02 Можно выставить любой таймаут

03 Можно управлять через дебаг-панель или через параметр в запросе

```
&shutdown_cookie={“Backend_1”:{“shutdown”:true}}
```

04 Можно деградировать отдельный метод бэкенда

```
&shutdown_cookie={“Backend_1”:{“endpoints”:“POST:/catalog/offers,POST:/catalog/offerId”}}
```

Пингер-тесты

01 Простое TS приложение, которое мы запускаем на сервере, используя ts-node

02 Реализовано с использованием Puppeteer и Ramda

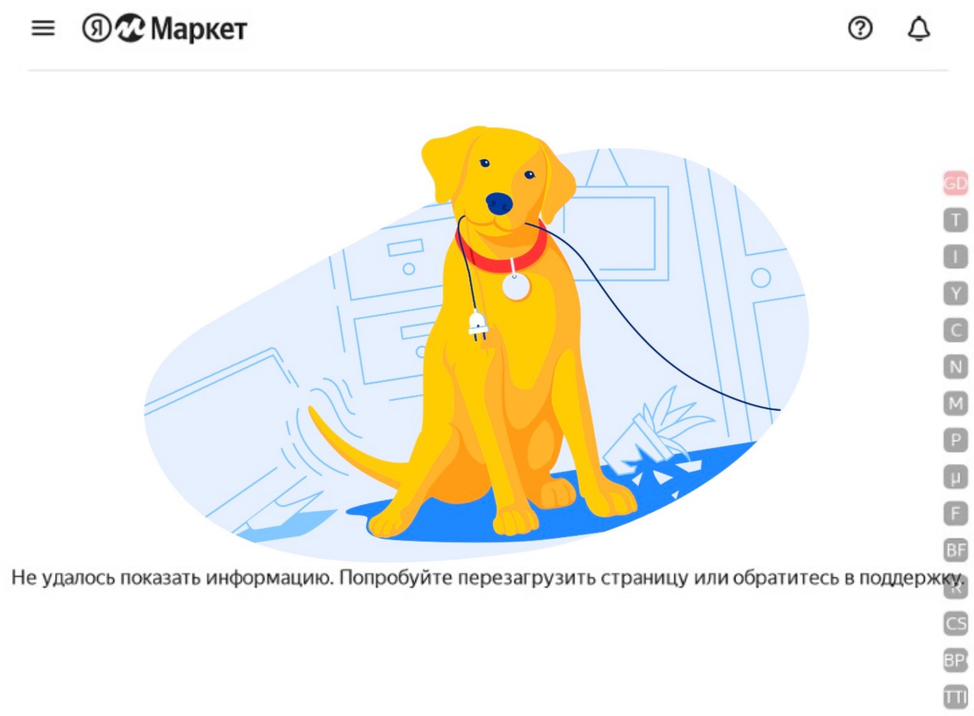
03 Источник данных — табличный список, в котором каждая строчка — отдельный тестовый сценарий с ожидаемыми элементами

Страница	Относительный URL	Регулярки для кода ответа	Сроки, которые должны быть в ответе
Регистрация	/registration	200	Регистрация
Каталог	/assortment	200	Каталог
Поставки	/supply	200	Поставки

market-partner:html:business-access:get

```
| -> Название теста: market-partner:html:business-access:get
| | -> Причина падения: missed expected text or selector: Сотрудники и доступы
| | -> Открыть страницу: Ссылка с залогином и отключенным бэкендом
```

Первая загрузка



Пингер может

01 Проверить совпадение ожидаемого кода ответа страницы с фактическим

02 Проверить наличие заданной строки на странице

03 Проверить отсутствие заданной строки на странице

04 Найти элемент по css-селектору и кликнуть на него, чтобы:

- Проверить код ответа страницы после клика
- Проверить наличие строки на странице после клика

05 Посчитать среднее время открытия каждой страницы

06 Провести запуск тестов в режиме chaos-тестирования

Но в этом подходе есть минусы

- Мы не проверяем взаимодействие между бэкендами
- Мы теряем кейсы, когда проблема возникает в момент работы пользователя на странице (сабмит форм, редактирование, ответ в чате)

Юра, мы всё запустили

Как сейчас работает процесс

01 Гоняем регресс каждую неделю — на все бэкенды, по всем страницам

- Всего нашли проблем — 112
- «Быстрым» регрессом — 29

02 По необходимости отключаем бэкенды на уровне балансеров — на процент и для других экспериментов

Над чем мы сейчас думаем

Это хорошо работает для http,
а что с другими протоколами (rpc, websocket)?

А если смотреть на бэкенд гранулярно
и пытаться деградировать отдельные методы?

Как автоматизировать регресс таким образом,
чтобы получать только новые проблемы
и не триггериться на собак, которые валидны?

А что делать с core-микросервисами,
которые используются во всём приложении
(например, авторизация)?

Выводы

01 Хаос-тестирование расширяет функциональное

02 Реализация может быть простой

03 Нужно хорошо понимать архитектуру вашего продукта

04 Это совместная работа с разработчиками, SRE и продуктом

Внедряйте хаос!

Светлана Баканова

Руководитель группы тестирования