ТИНЬКОФФ

SRE.

Неочевидные способы прострелить себе колено в попытках сделать приложение надежнее

Алимов Лев

18.09.2023

Лев Алимов

Site Reliability Engineer

- В 2013 году был фрезеровщиком
- Выбирал между Dev и Ops
- Изучал Java, ООП и алгоритмы

За 9 лет прошел путь от эникея до ведущего SRE

Чинил штуки на разных уровнях абстракции: сети, железо, ОС, распределенные системы...

Снимаю метрики не только с сервисов, но и с самого себя. Слежу за показателями с целью предотвращения сбоя на этой production системе

О чем доклад?

100% надежности нет

Любая внешняя система может развалиться

Есть общепринятые паттерны

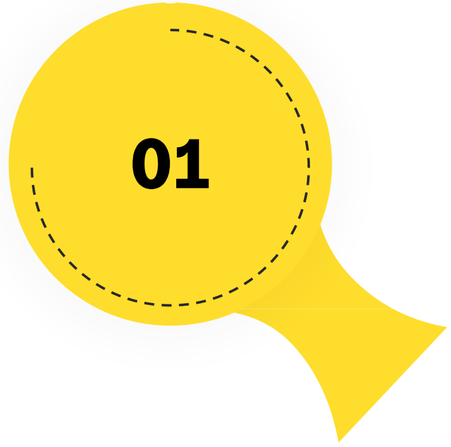
Но так ли они полезны?

У паттернов есть собственные
подводные камни

У паттернов есть собственные подводные камни

Уже есть проблемы

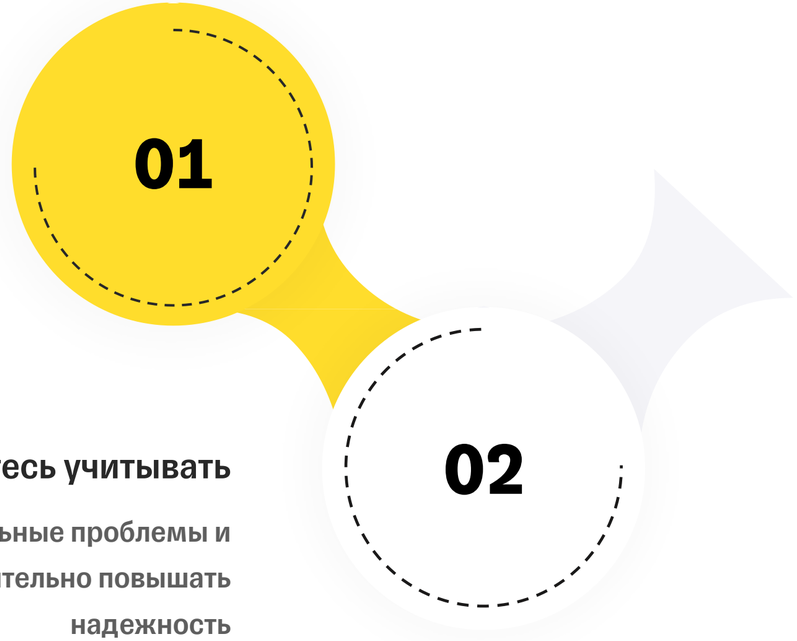
И вы это поймете



У паттернов есть собственные подводные камни

Уже есть проблемы

И вы это поймете



Научитесь учитывать

Потенциальные проблемы и действительно повышать надежность

У паттернов есть собственные подводные камни

Уже есть проблемы

И вы это поймете

01

Научитесь учитывать

Потенциальные проблемы и действительно повышать надежность

02

03

Рассмотрим 3 кейса

С применением «книжных» паттернов

У паттернов есть собственные подводные камни

Уже есть проблемы

И вы это поймете

01

Научитесь учитывать

Потенциальные проблемы и действительно повышать надежность

02

Рассмотрим 3 кейса

С применением «книжных» паттернов

03

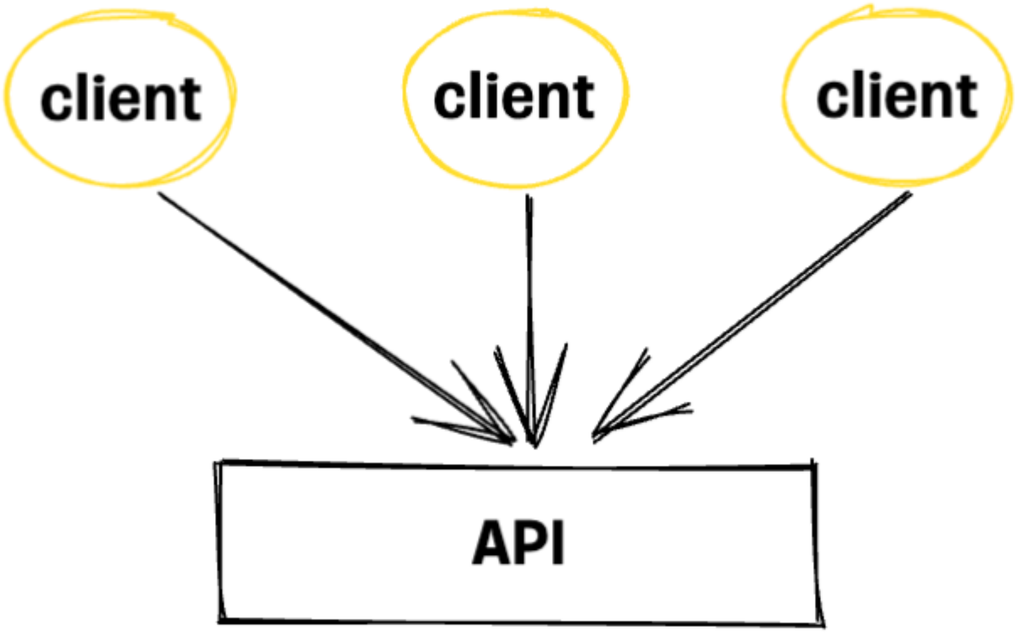
Развалить production

С помощью механизмов для повышения отказоустойчивости

04

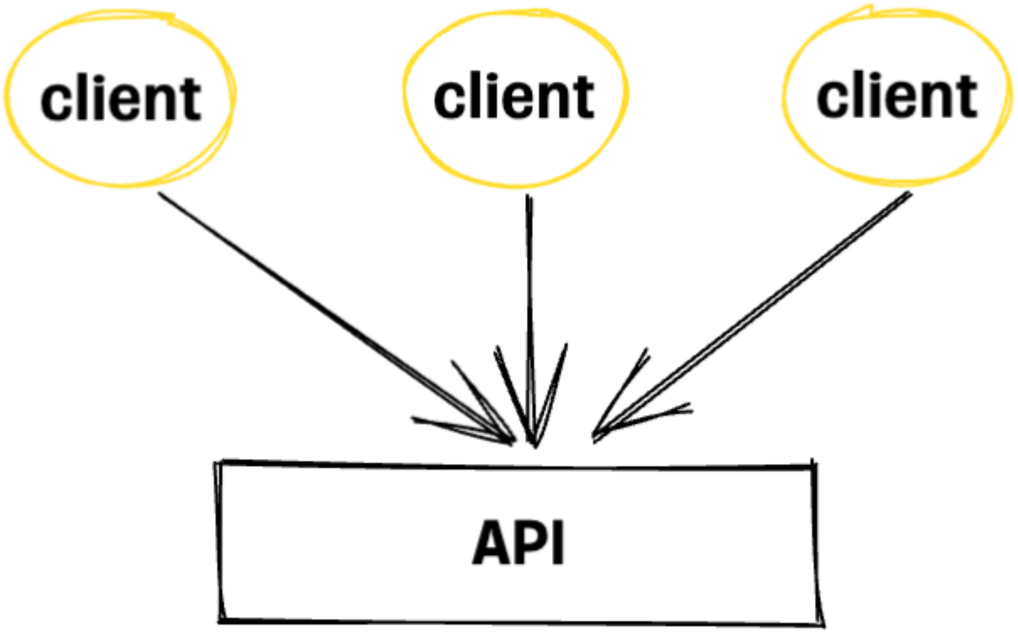
1. Retry Until It Die

Распределенное приложение



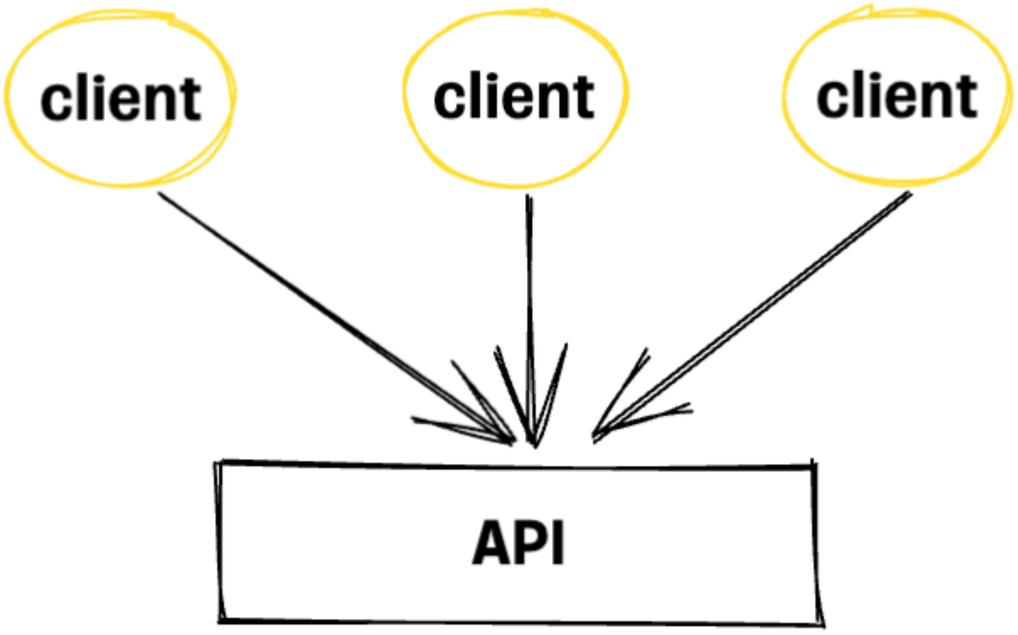
Принимает на себя все запросы

Распределенное приложение



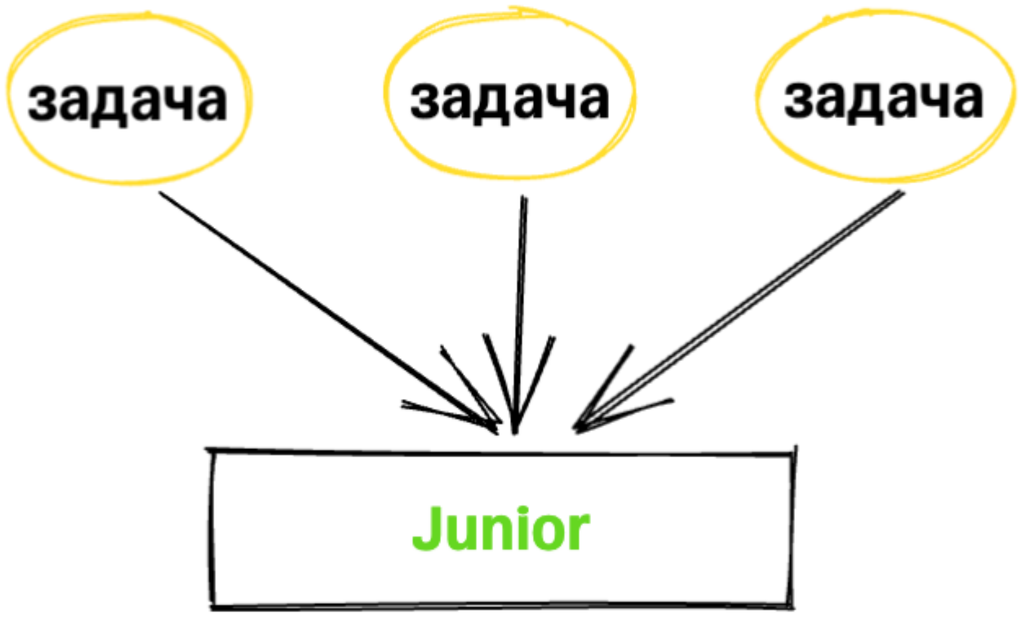
- Принимает на себя все запросы
- Ходит во внешние системы

Распределенное приложение



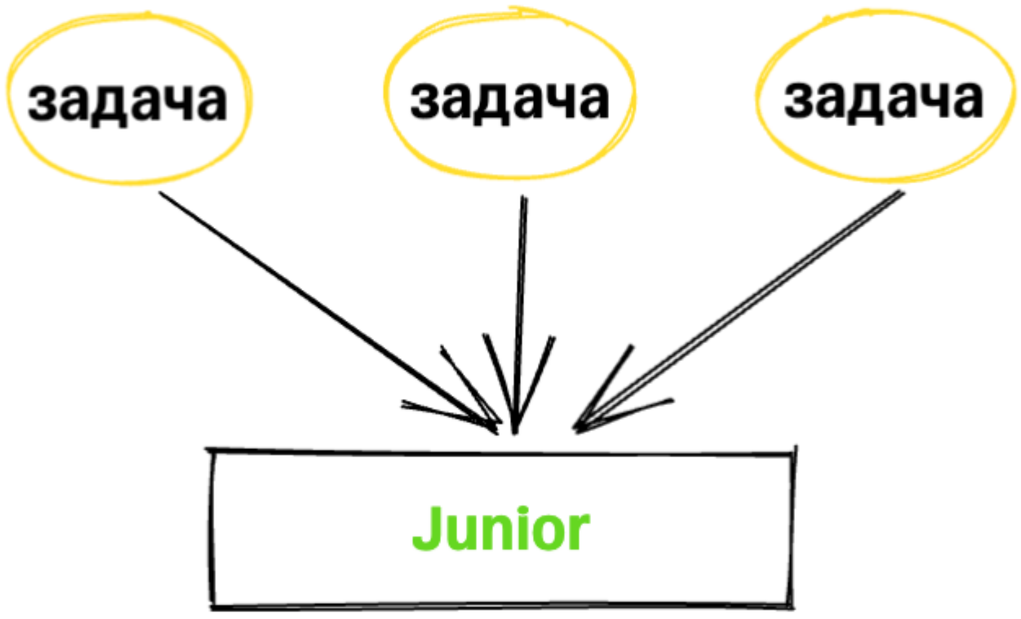
- Принимает на себя все запросы
- Ходит во внешние системы
- Не всегда получает ответы

Распределенное приложение



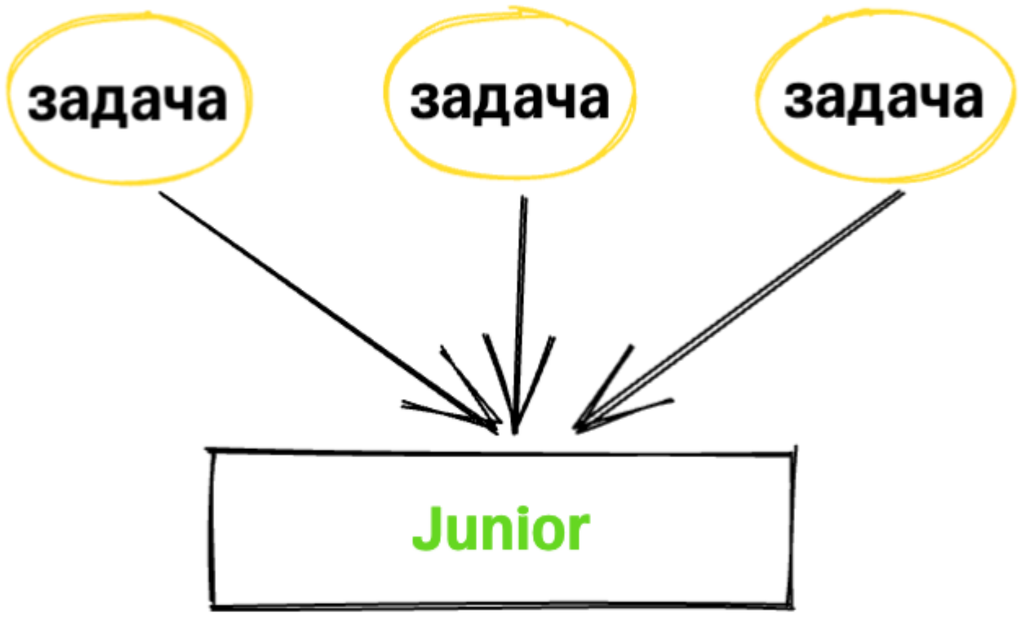
Берет все мелкие задачи

Распределенное приложение



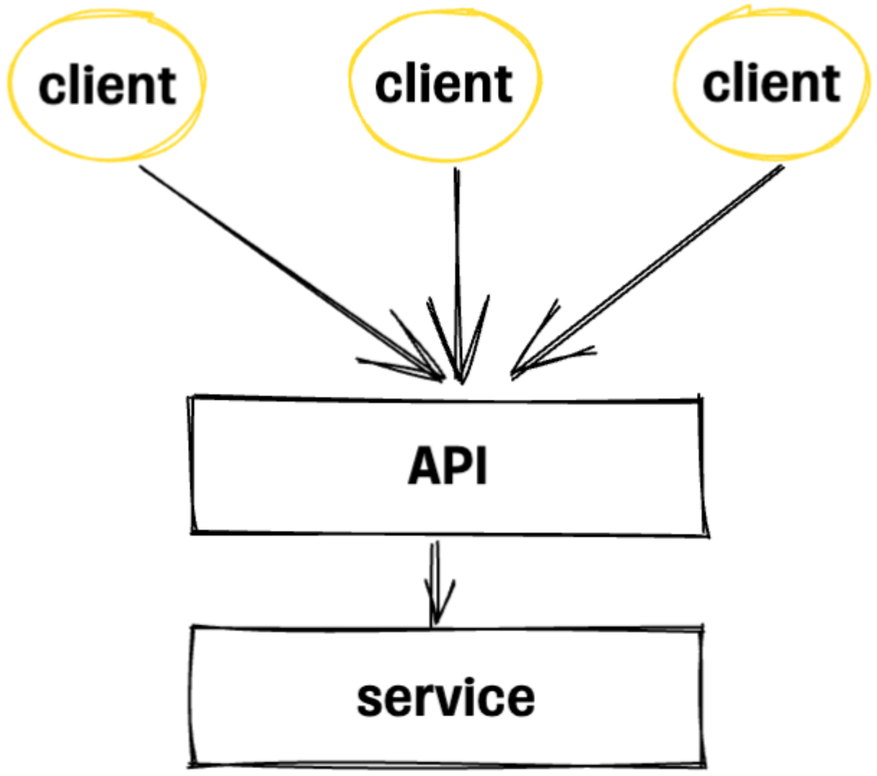
- Берет все мелкие задачи
- Задает вопросы

Распределенное приложение



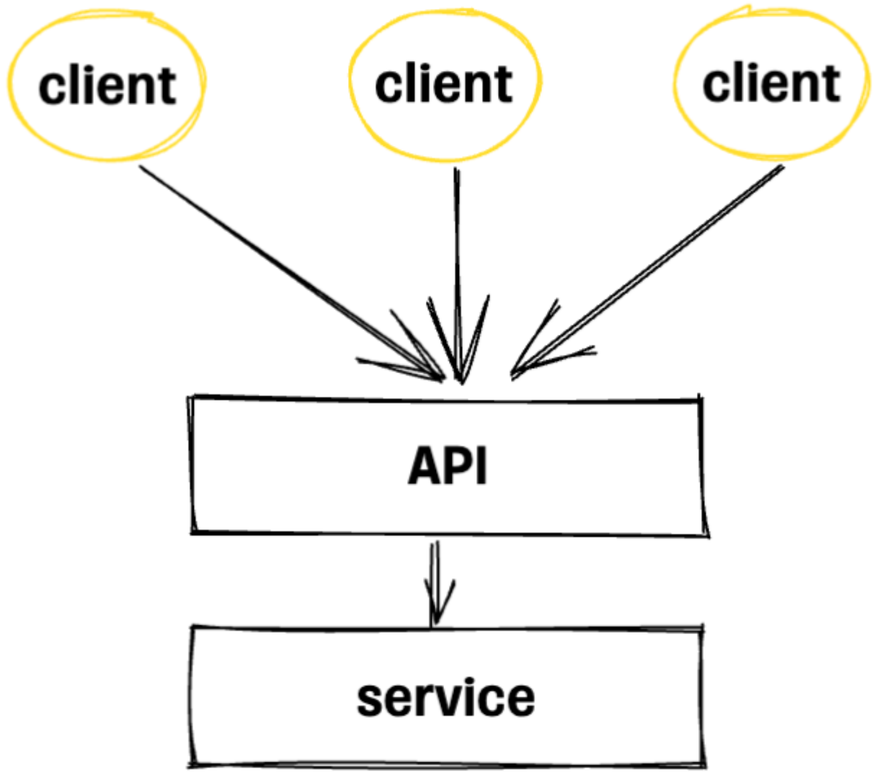
- Берет все мелкие задачи
- Задает вопросы
- Не всегда понимает с первого раза

Распределенное приложение



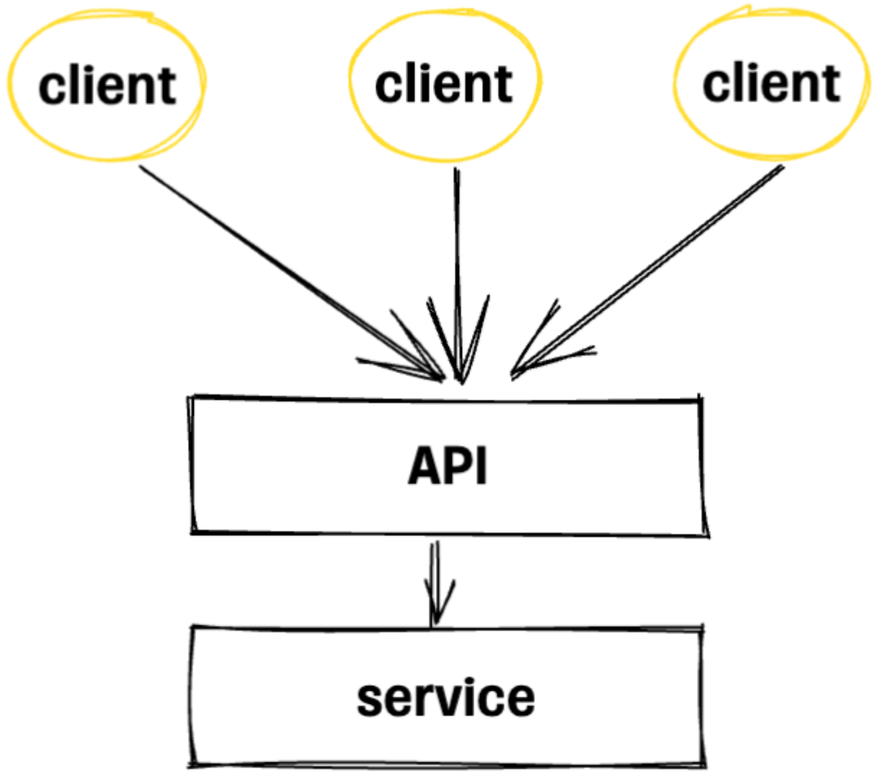
- Принимает основную нагрузку

Распределенное приложение



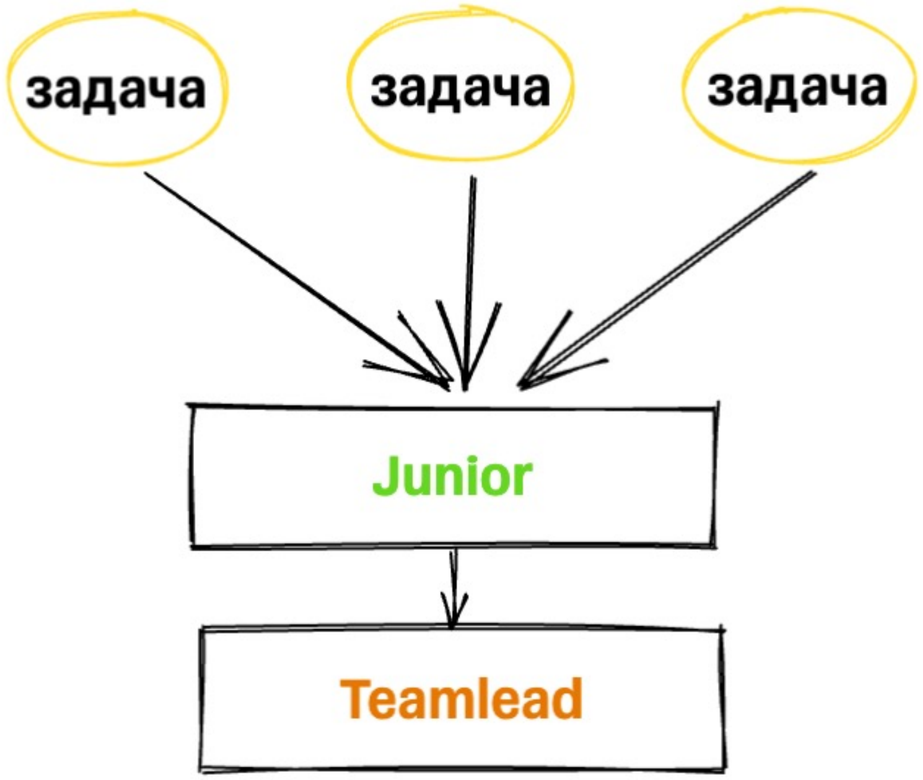
- Принимает основную нагрузку
- Может отвечать быстро

Распределенное приложение



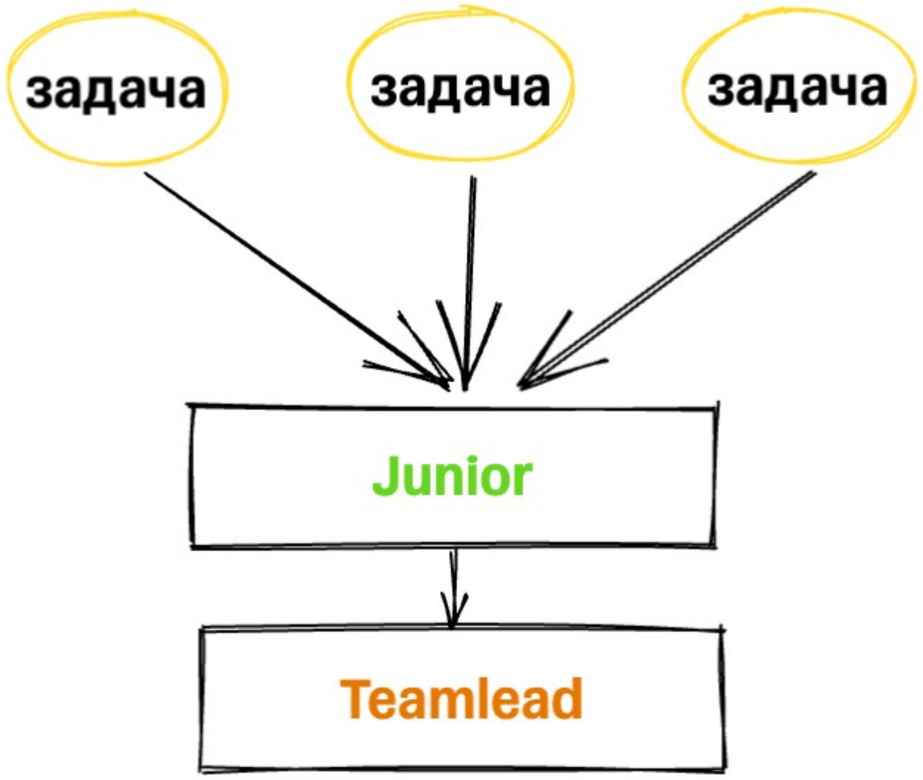
- Принимает основную нагрузку
- Может отвечать быстро
- За деталями идет во внешнюю систему

Распределенное приложение



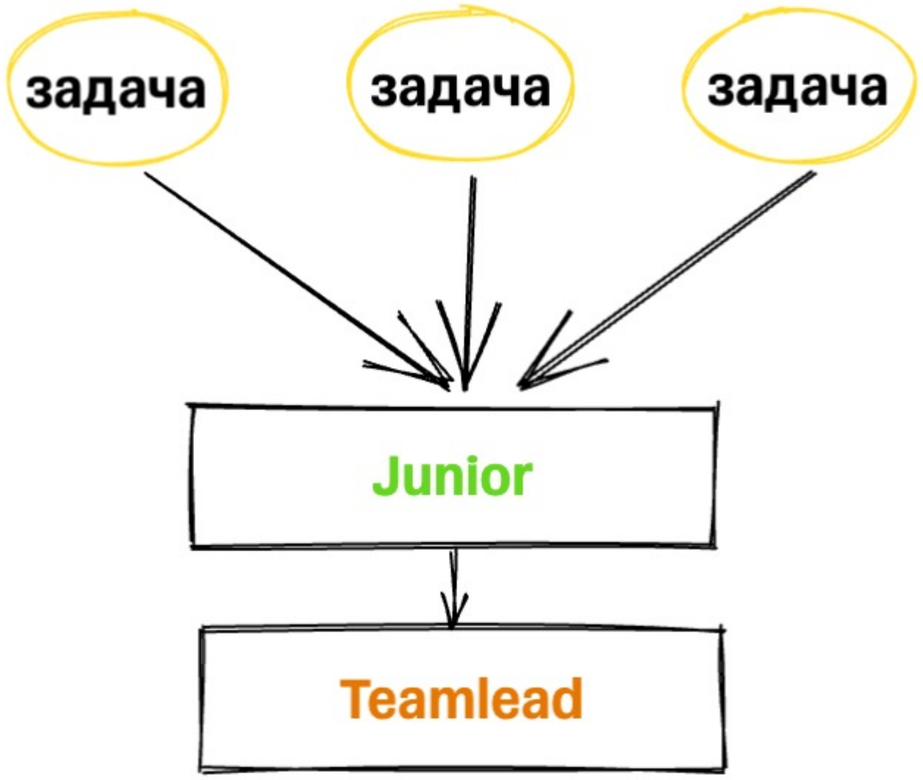
Отвечает на вопросы джунов

Распределенное приложение



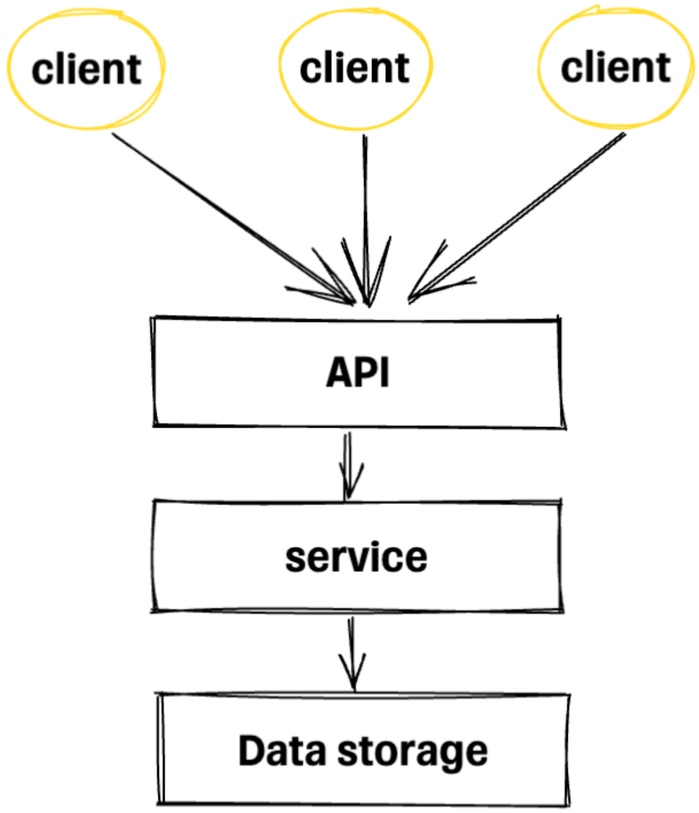
- Отвечает на вопросы джунов
- Имеет широкий кругозор

Распределенное приложение



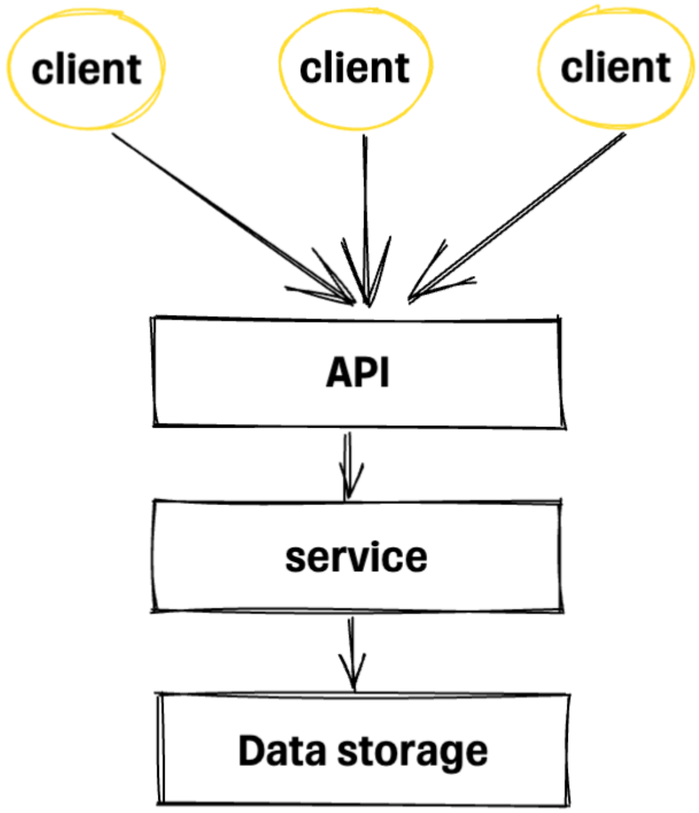
- Отвечает на вопросы джунов
- Имеет широкий кругозор
- За деталями идет к сеньорам

Распределенное приложение



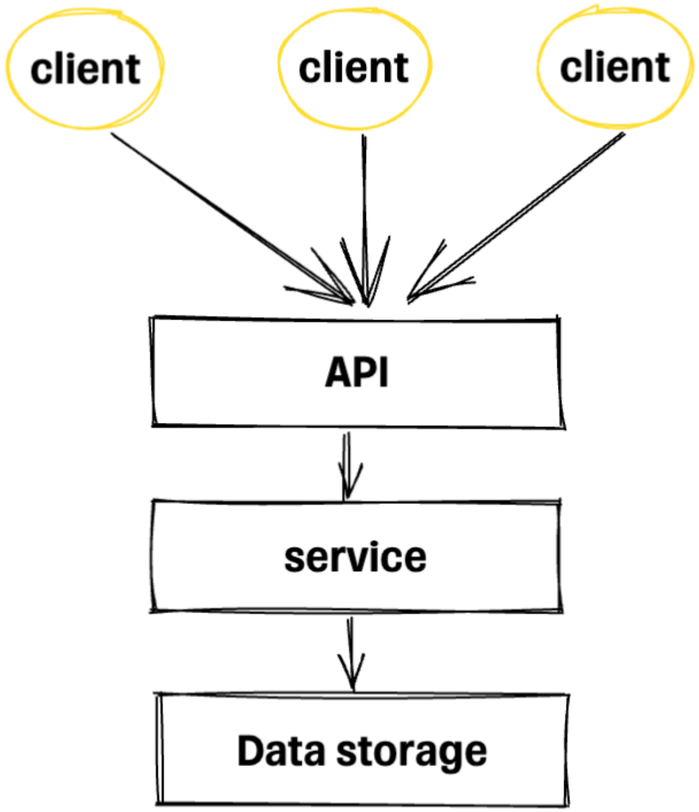
Отвечает подробно

Распределенное приложение



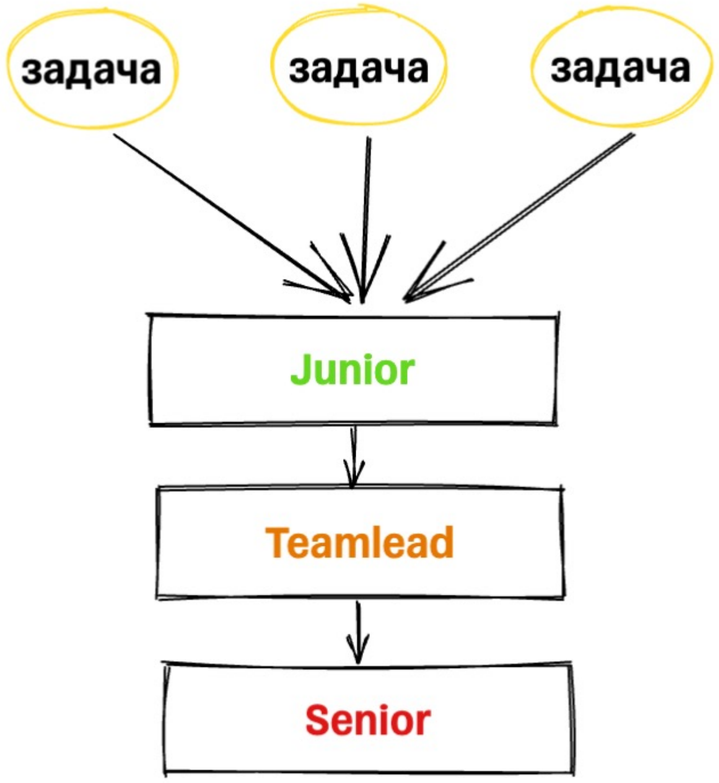
- Отвечает подробно
- Занят «тяжелыми» задачами

Распределенное приложение



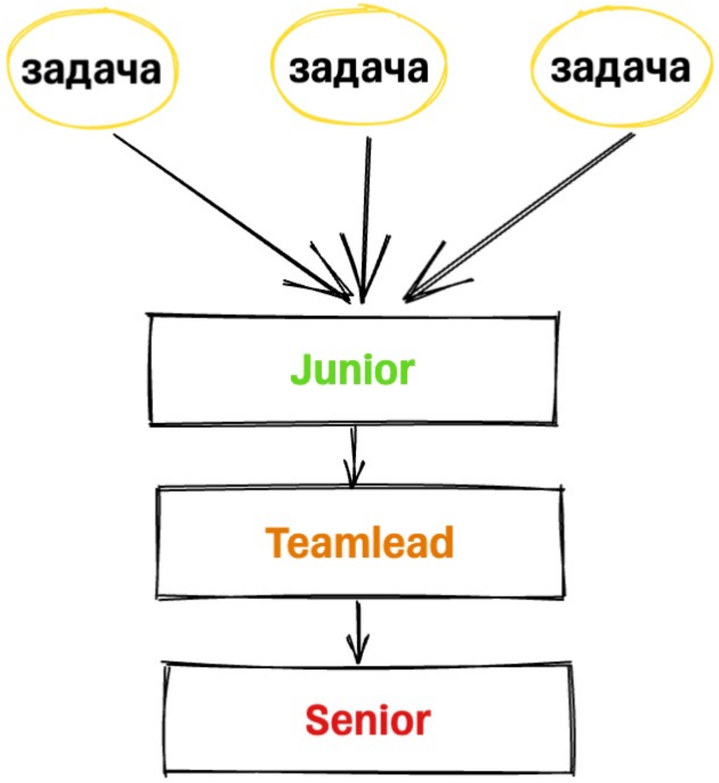
- Отвечает подробно
- Занят «тяжелыми» задачами
- Нет запаса по ресурсам

Распределенное приложение



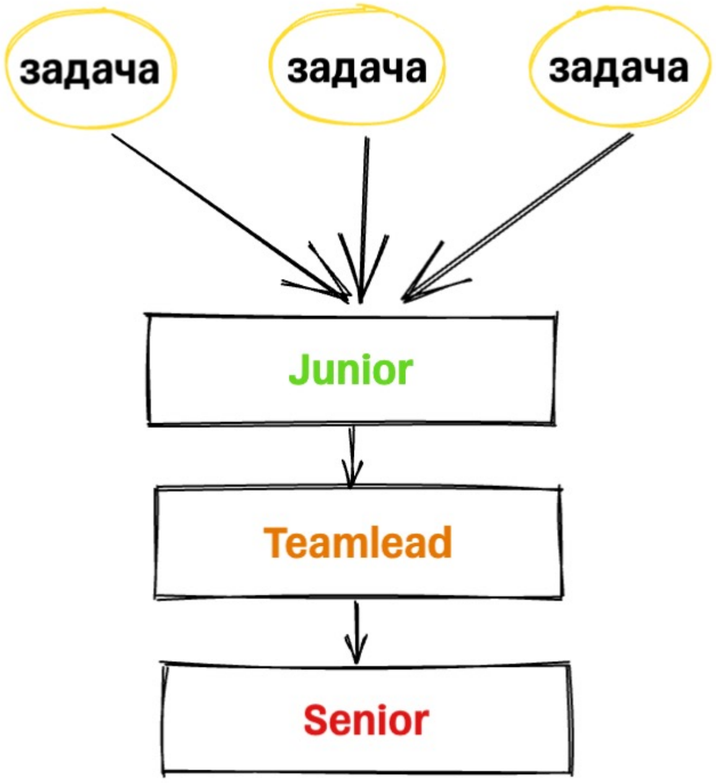
Знает всё. В деталях

Распределенное приложение



- Знает всё. В деталях
- Выполняет объемные задачи

Распределенное приложение



- Знает всё. В деталях
- Выполняет объемные задачи
- Почти сгорел

Это был обычный вторник...

Запросы приходят на API

Задачи появляются на доске

API их обрабатывает

Скорость появления задач на доске - обычная

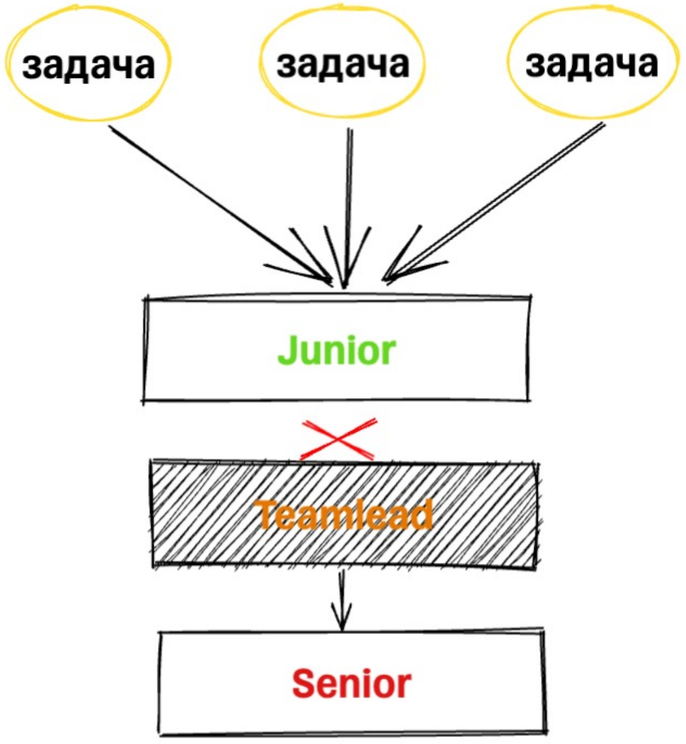
На системе был релиз

Бизнес хочет новую фичу

Запросы требуют детализации

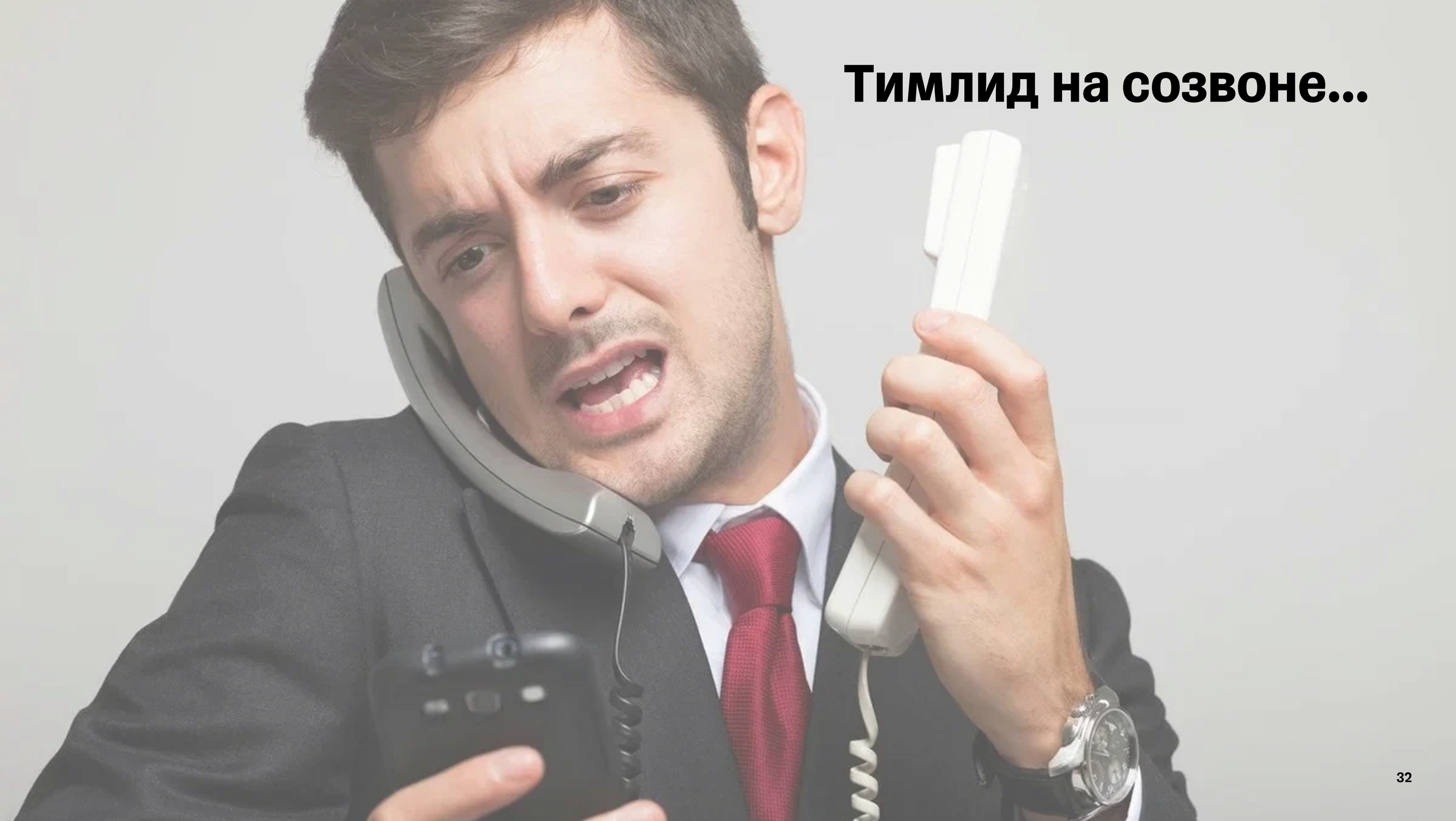
Для джуна задачи - новые

Что-то случилось...

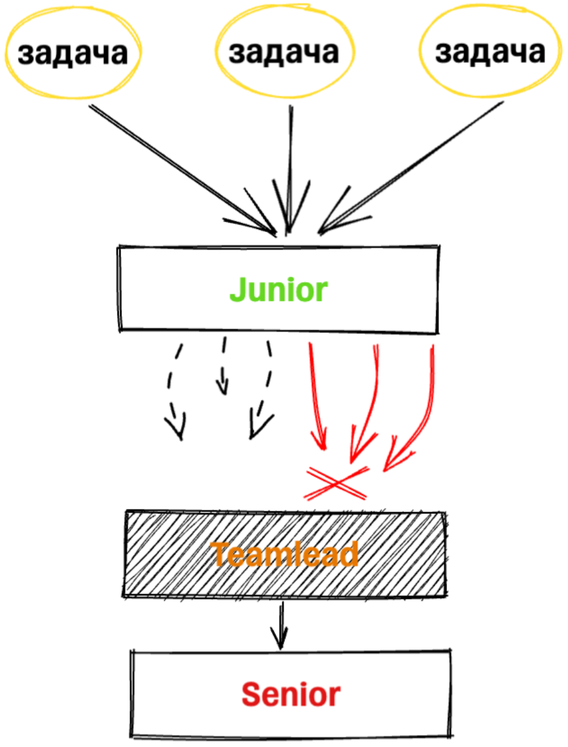


Сервис не отвечает

Тимлид на созвоне...



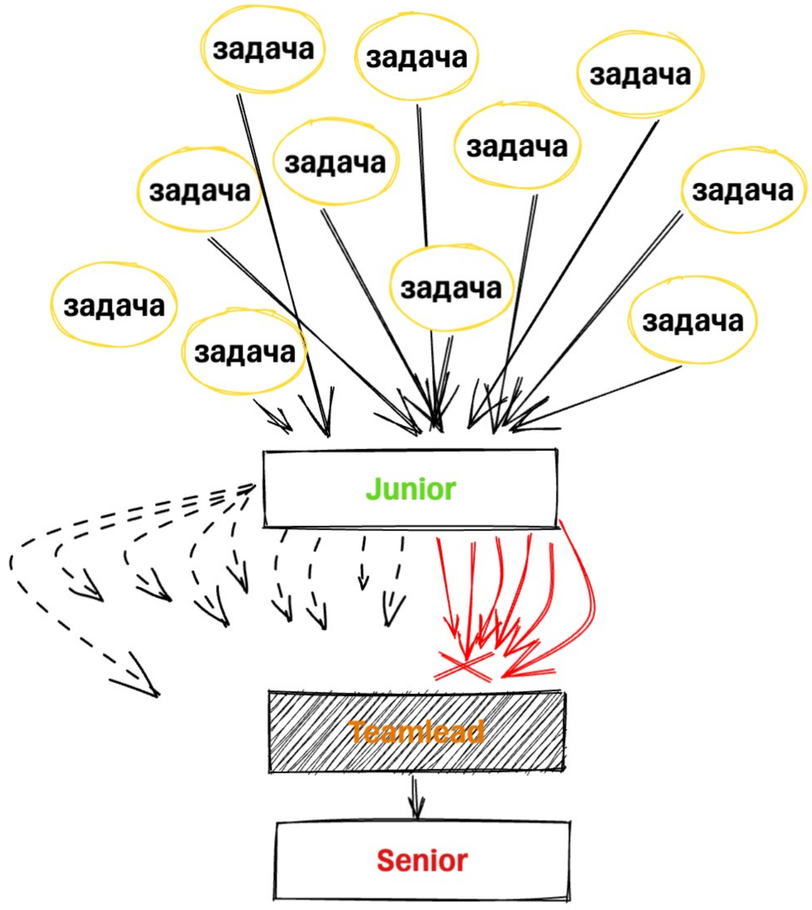
Что-то случилось...



- Сервис не отвечает
- Запросы повторяются (retry)

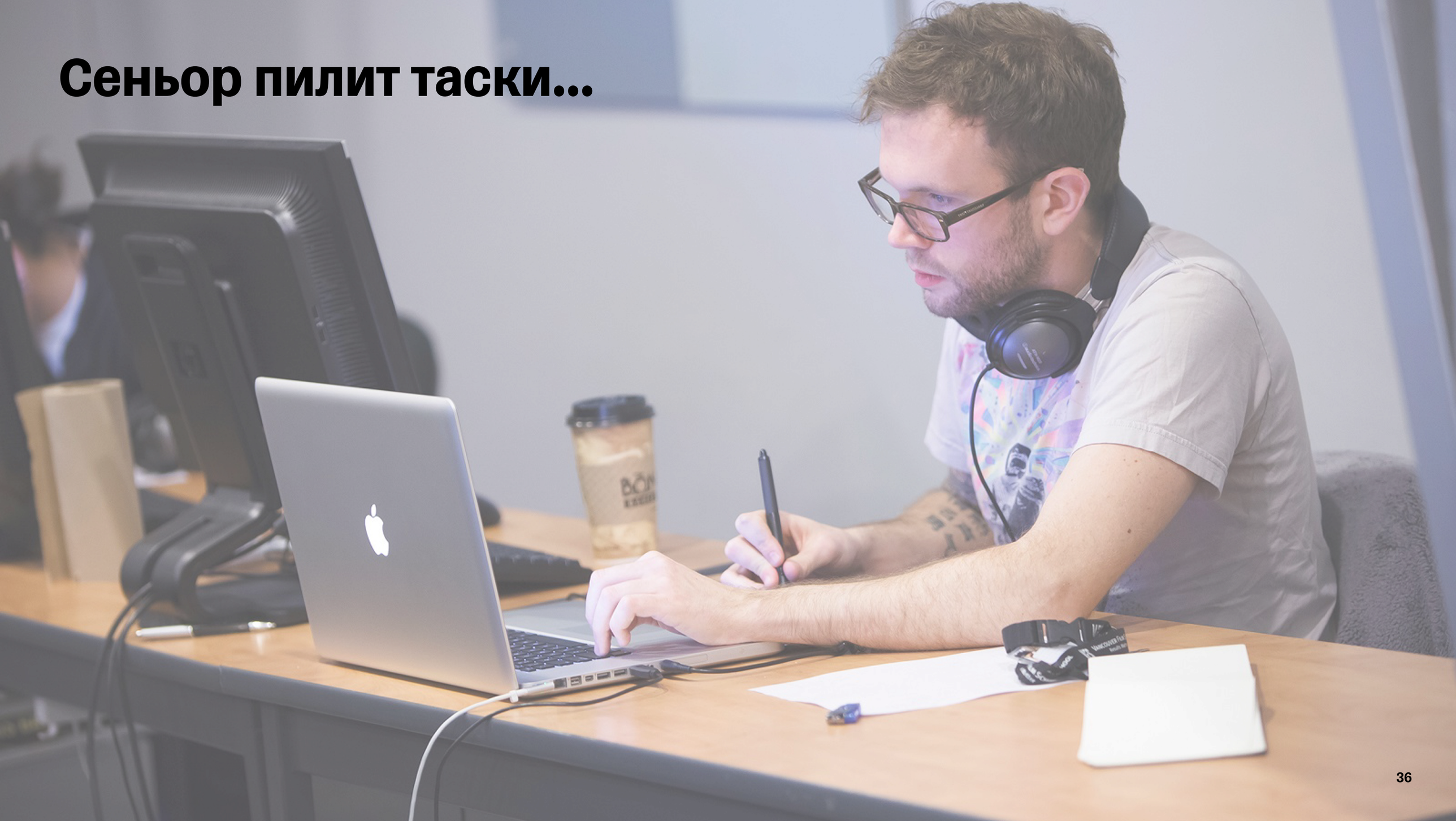
Джун паникует...

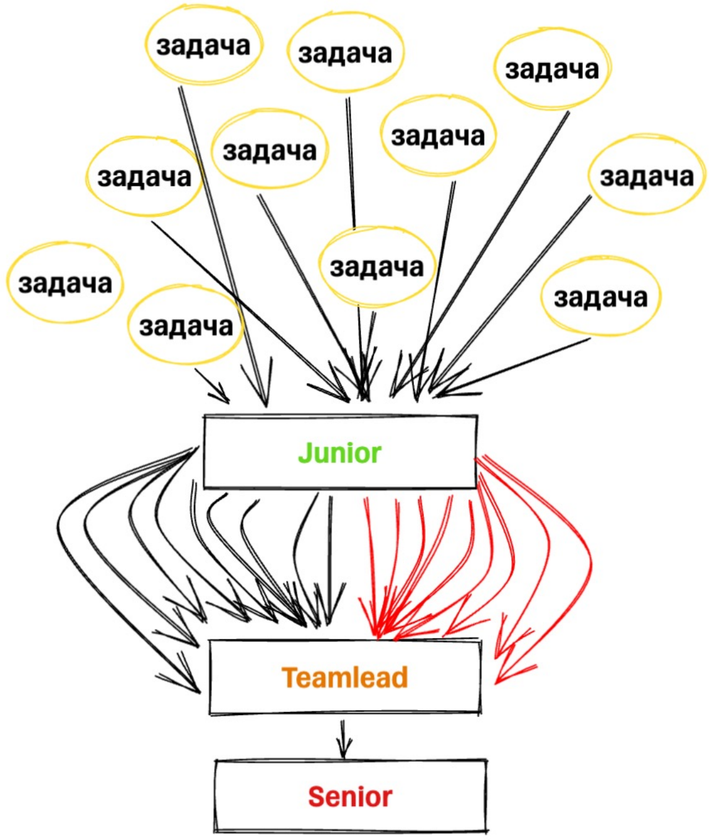
Что-то случилось...



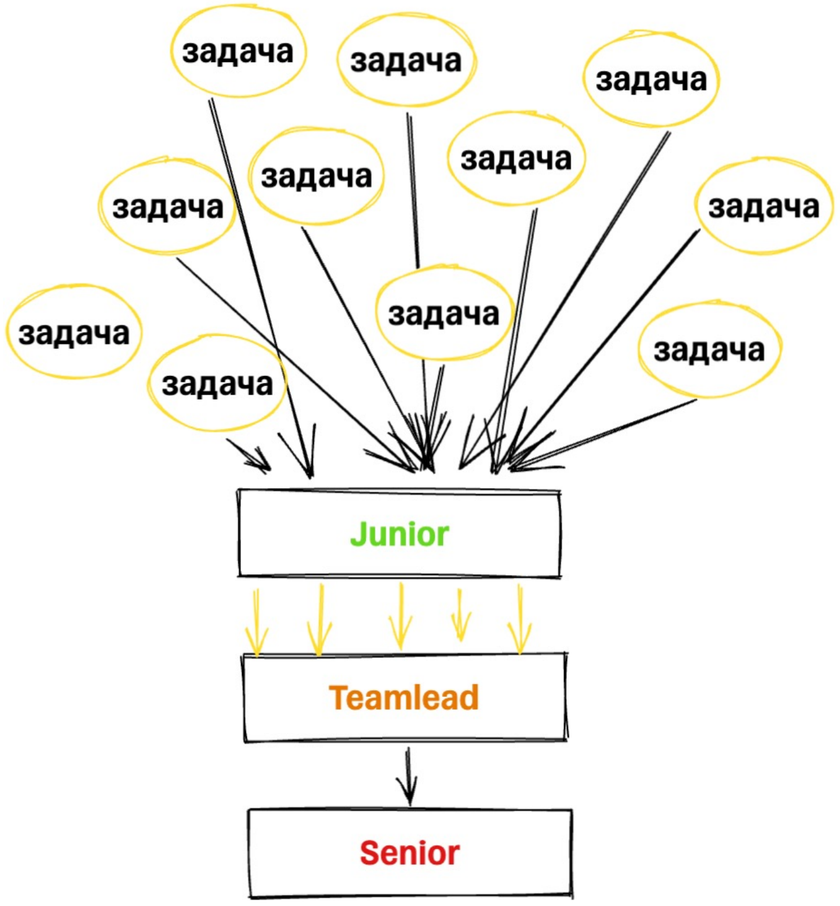
- Сервис не отвечает
- Запросы повторяются (retry)
- Накапливается очередь

Сеньор пилит таски...

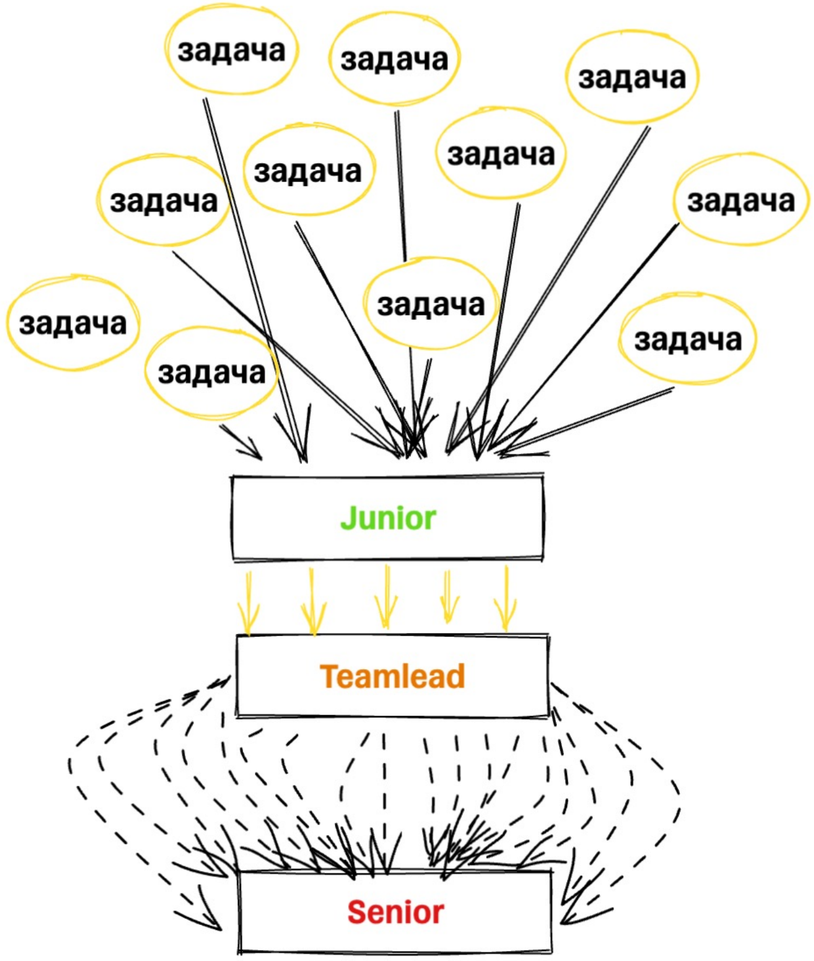




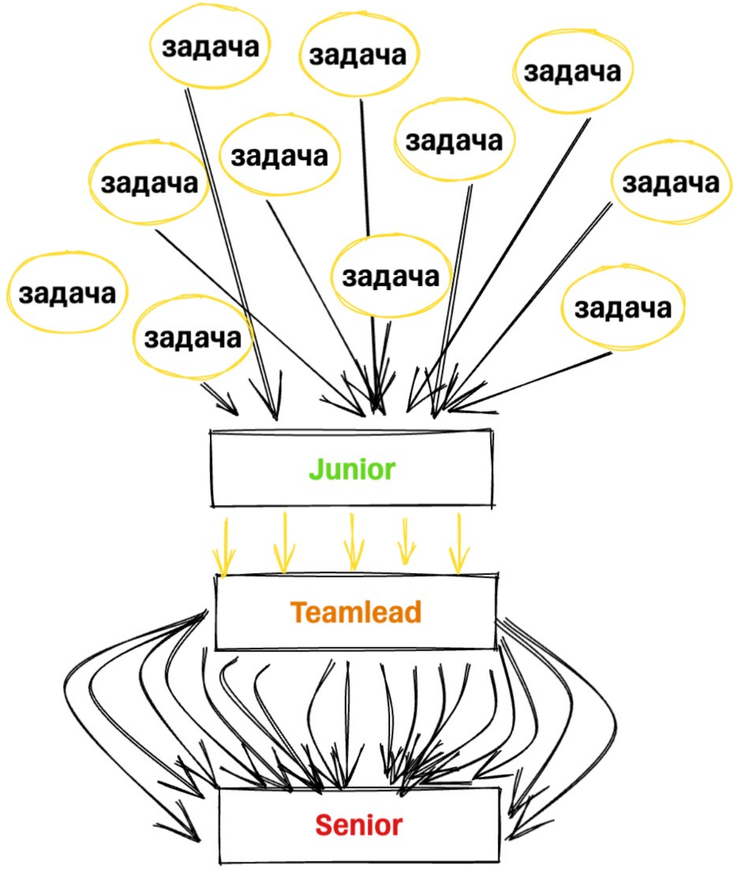
У тимлида закончился созвон



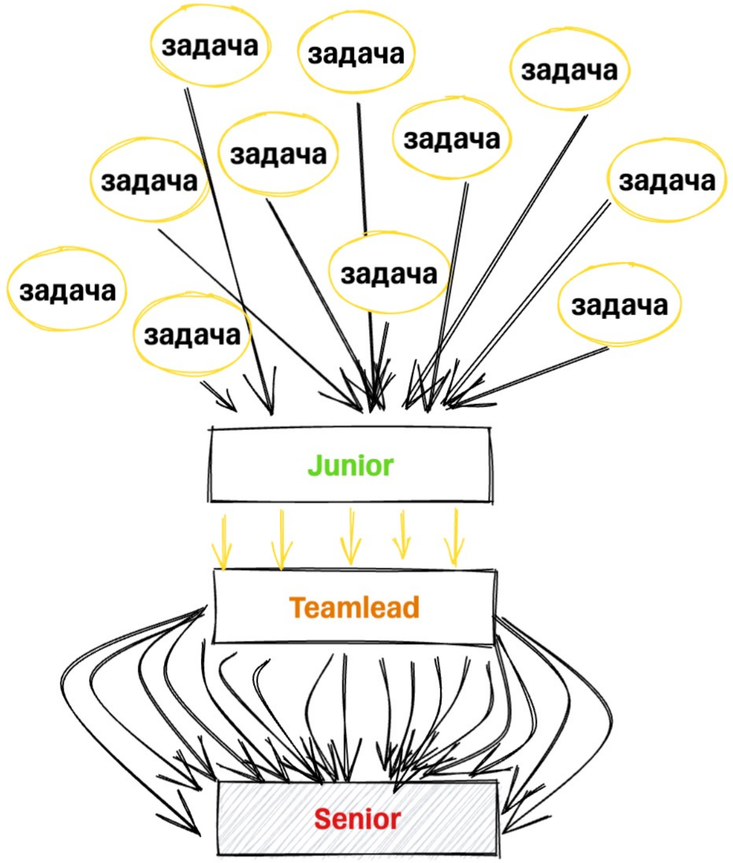
- У тимлида закончился созвон
- Сервис вычитывает очередь



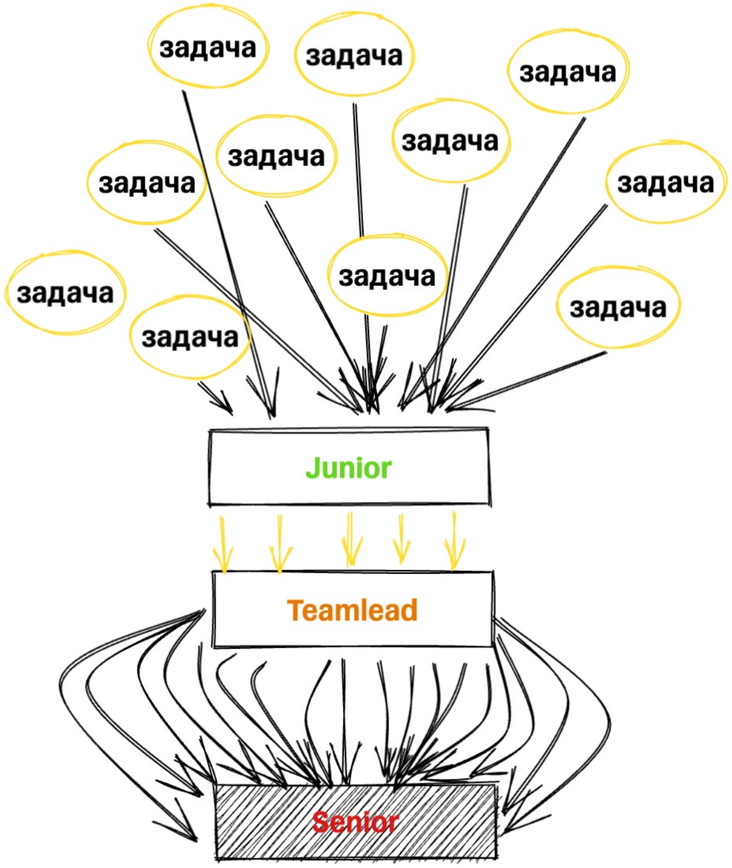
- У тимлида закончился созвон
- Сервис вычитывает очередь
- Понимает что нужны детали



Тимлид направил запросы сеньору

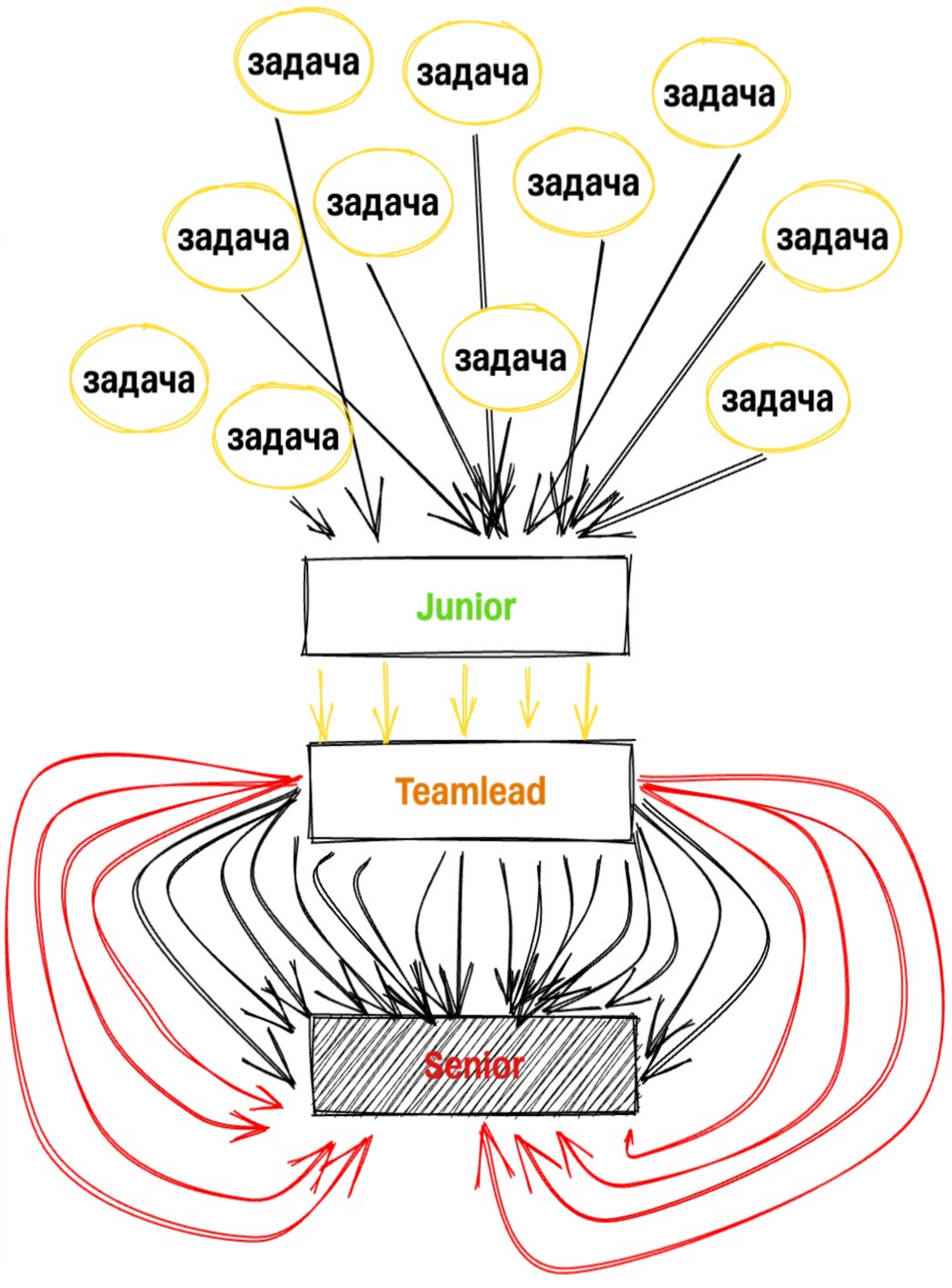


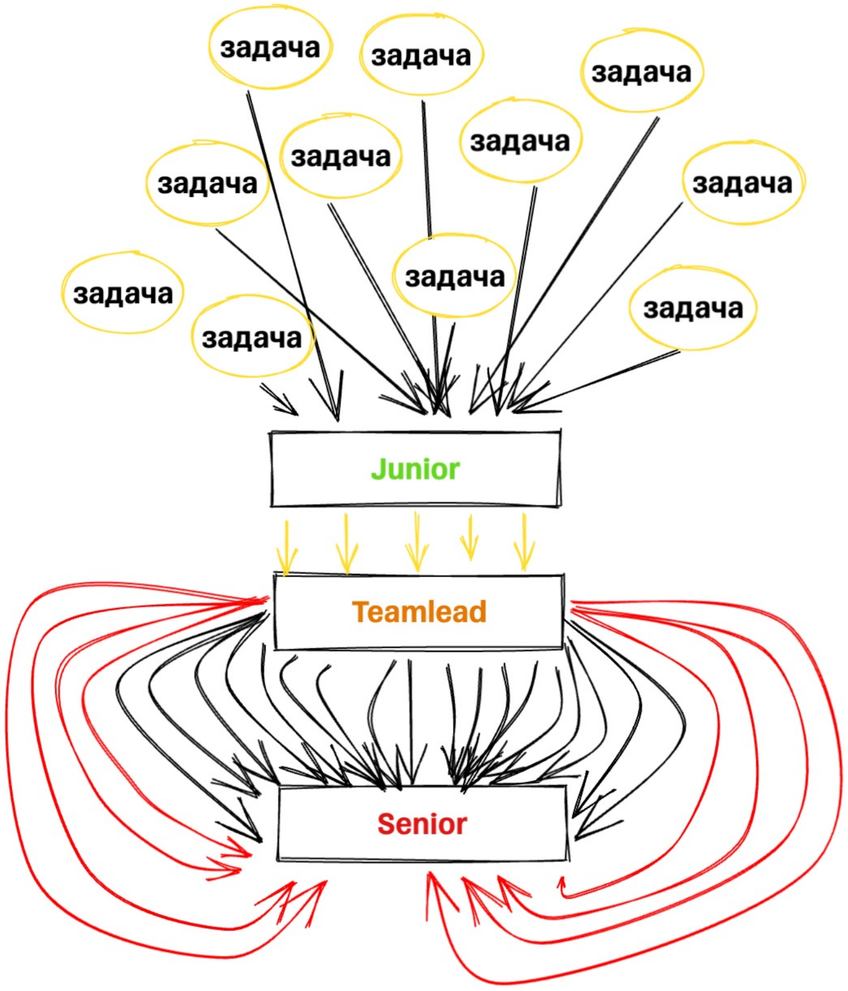
- Тимлид направил запросы сеньору
- Сеньору нужен контекст



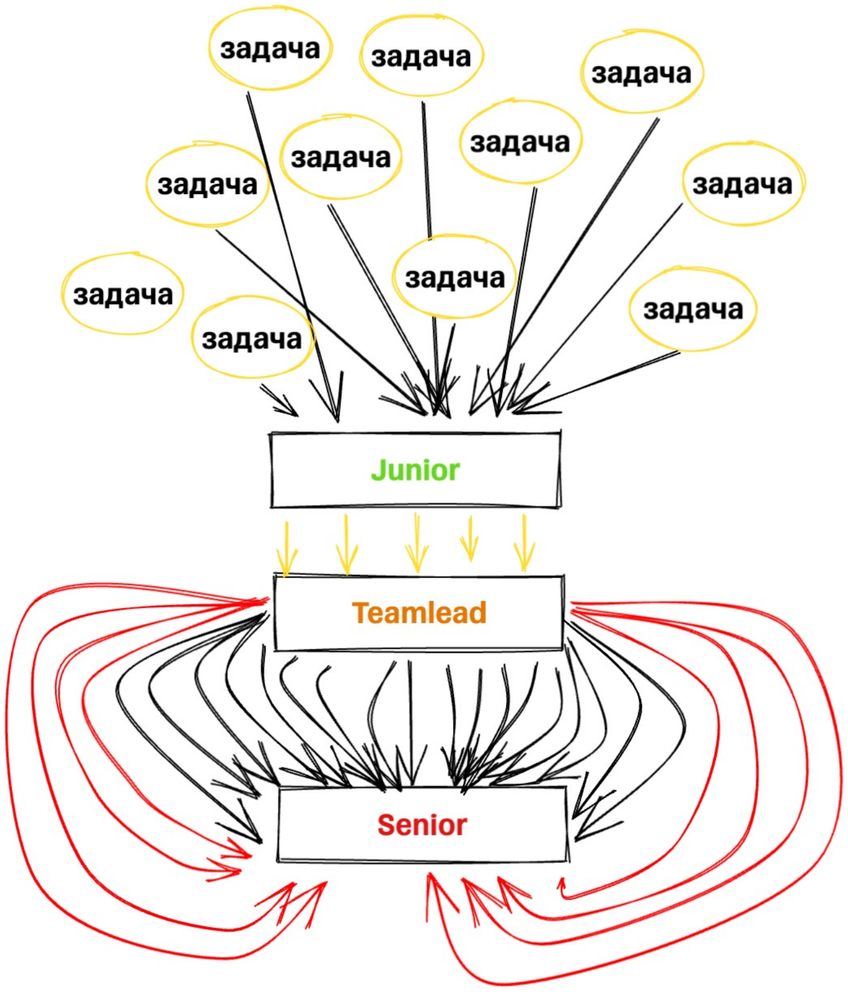
- Тимлид направил запросы сеньору
- Сеньору нужен контекст
- Сервис с данными исчерпал ресурсы

Сеньор устал...

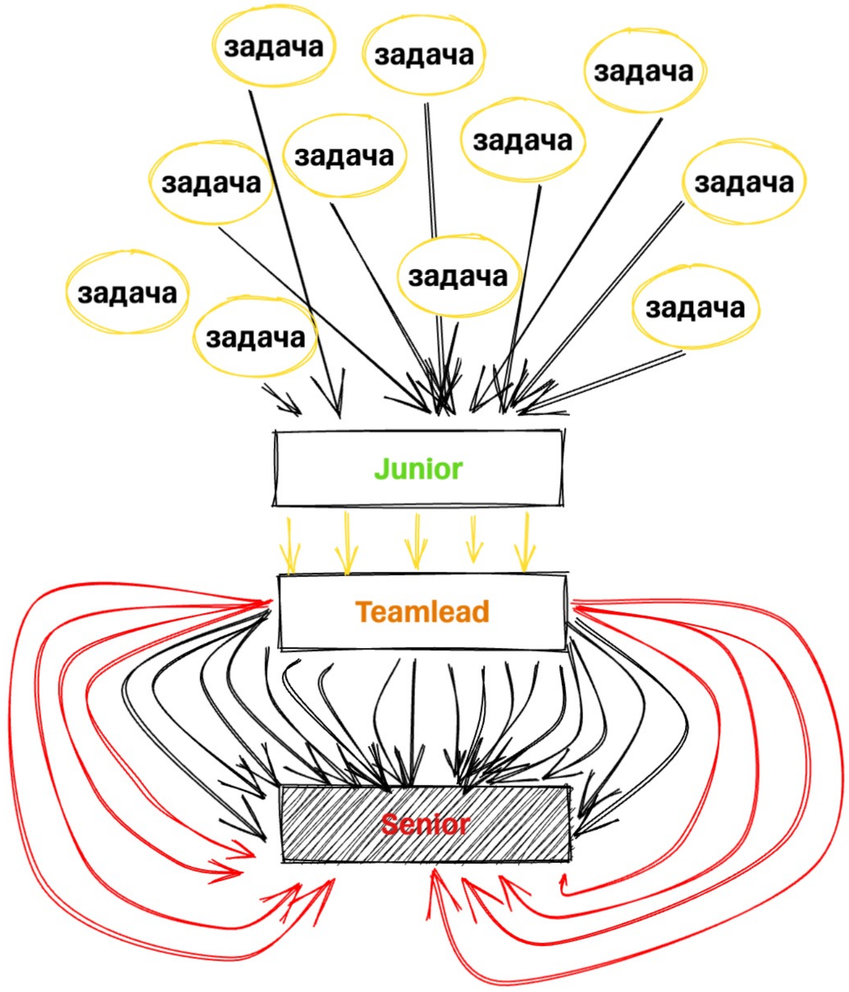

задача
задача
задача
задача
задача
задача
задача
задача
задача
задача
задача
Junior
Teamlead
Senior



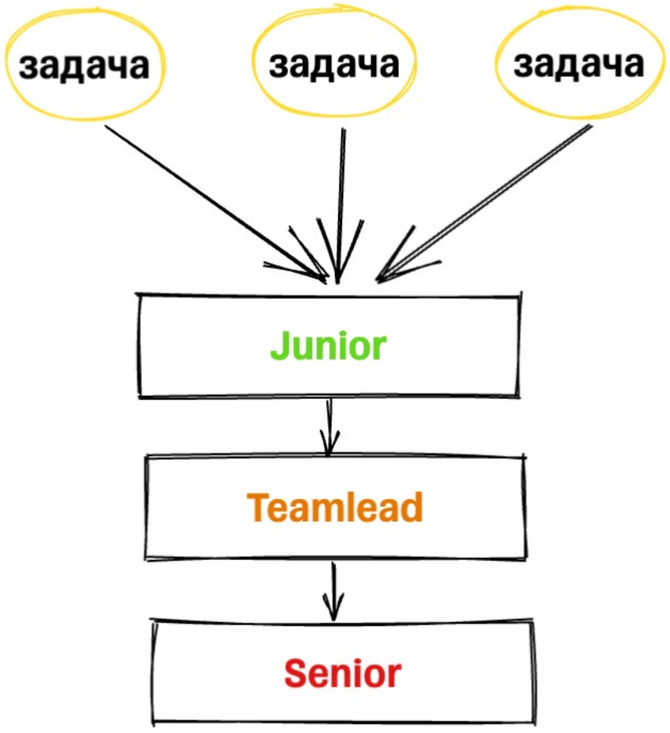
Высоконагруженный сервис вернулся



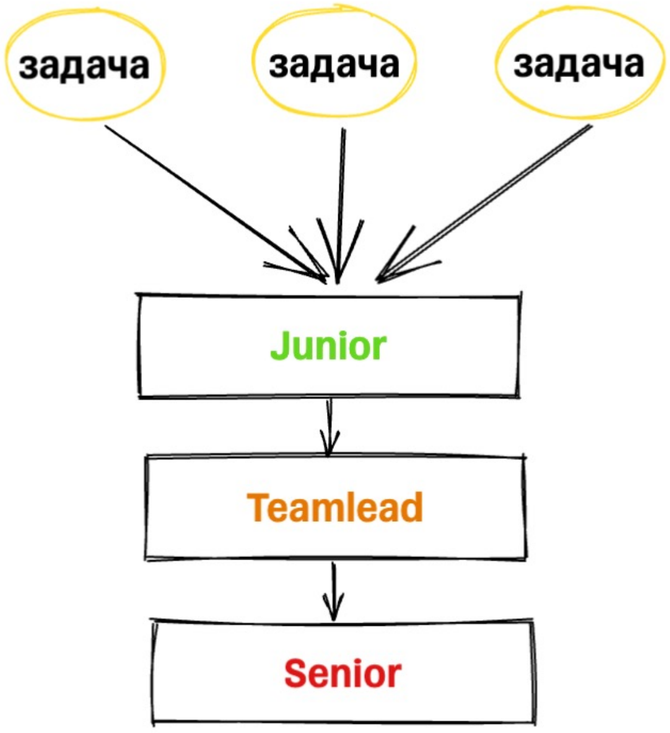
- Высоконагруженный сервис вернулся
- Увидел пачку запросов



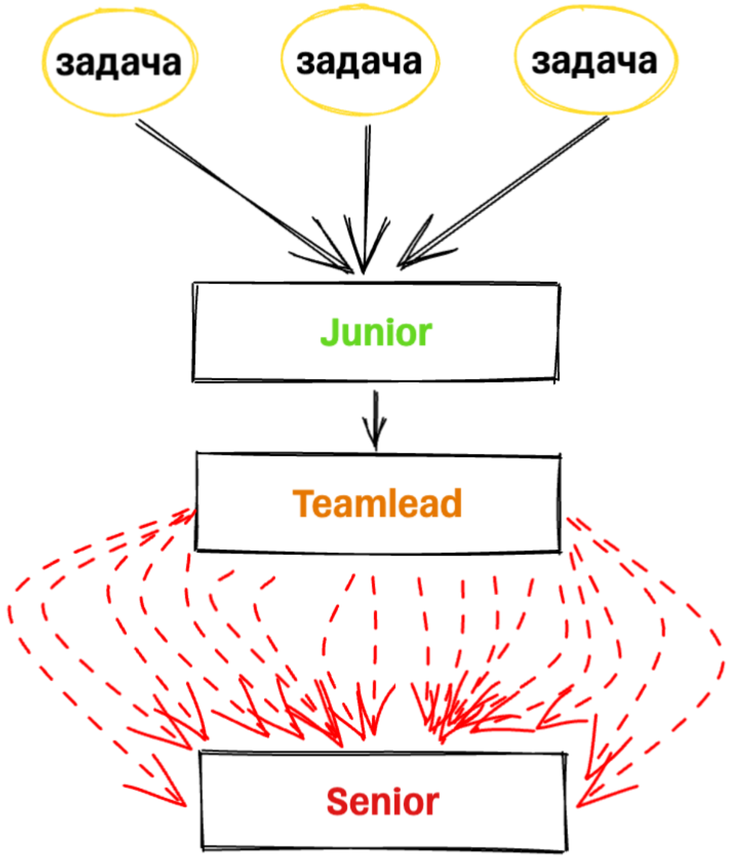
- Высоконагруженный сервис вернулся
- Увидел пачку запросов
- Снова упал



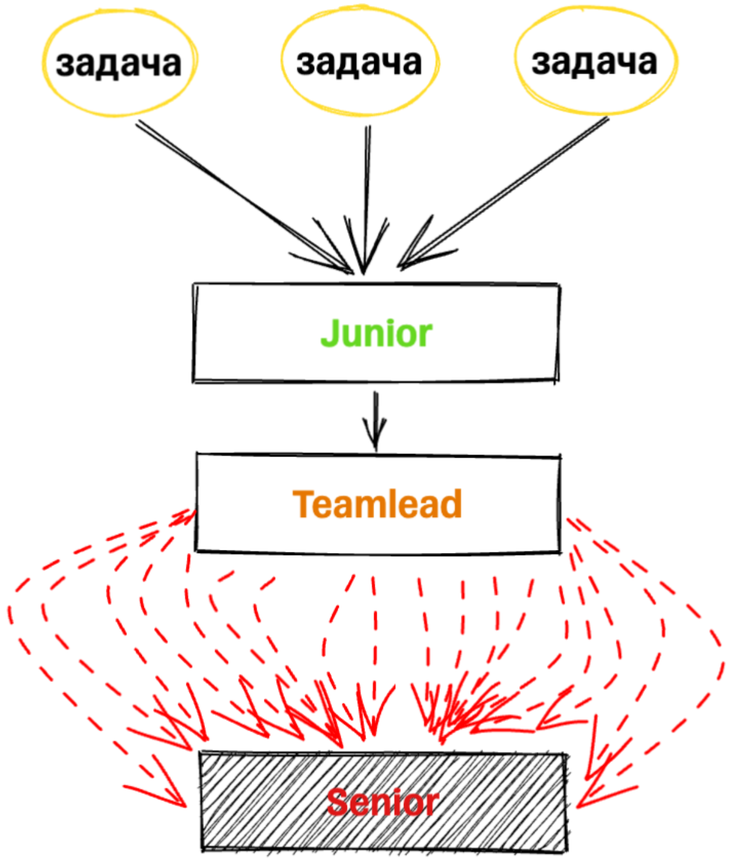
Откатили релиз



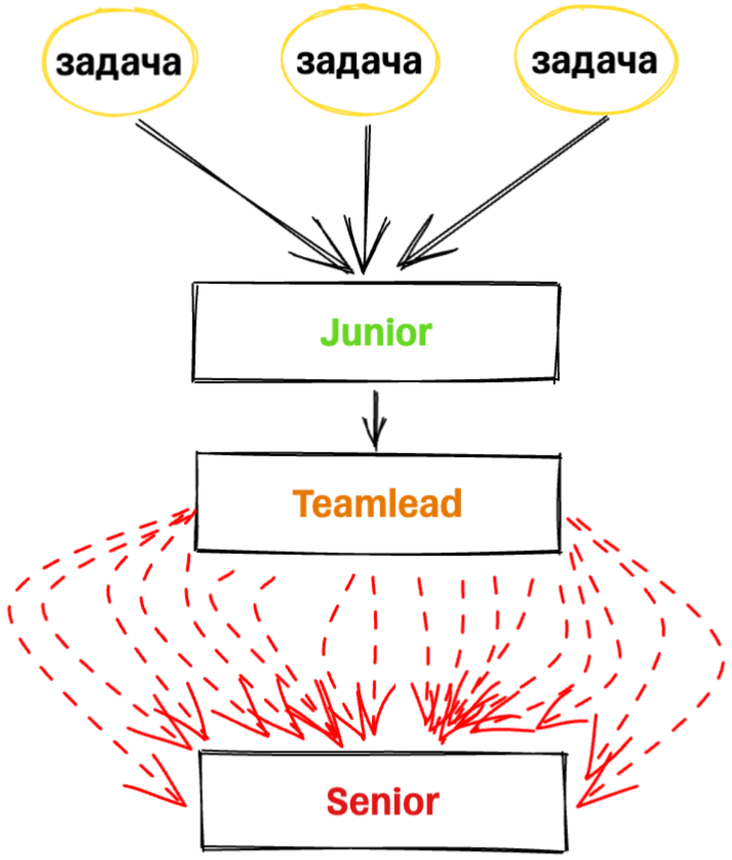
- Откатили релиз
- Перестали получать «тяжелые» запросы



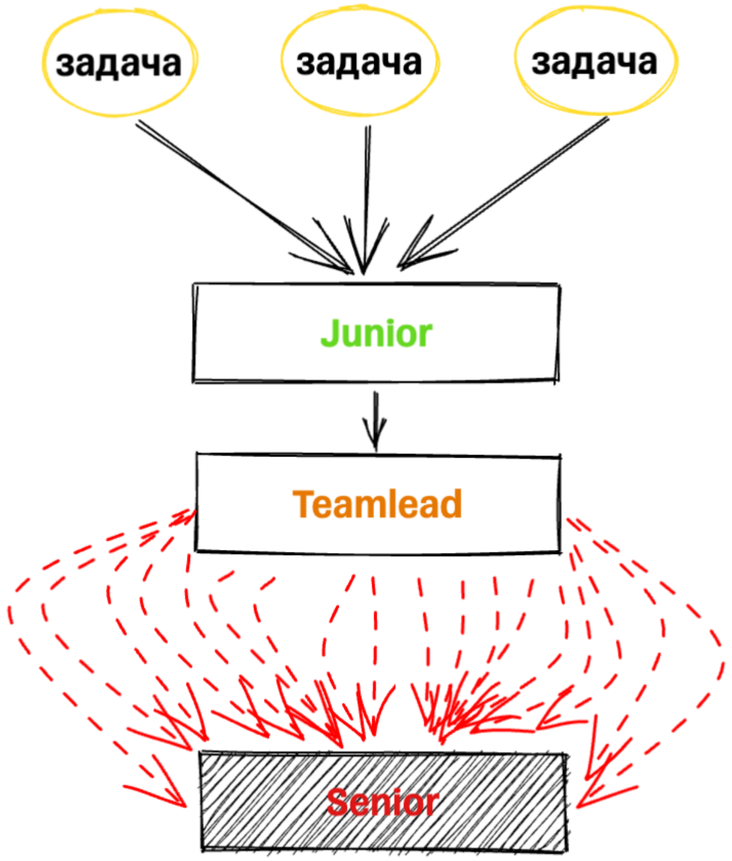
- Откатили релиз
- Перестали получать «тяжелые» запросы
- Тимлид – хороший. Тимлид запомнил



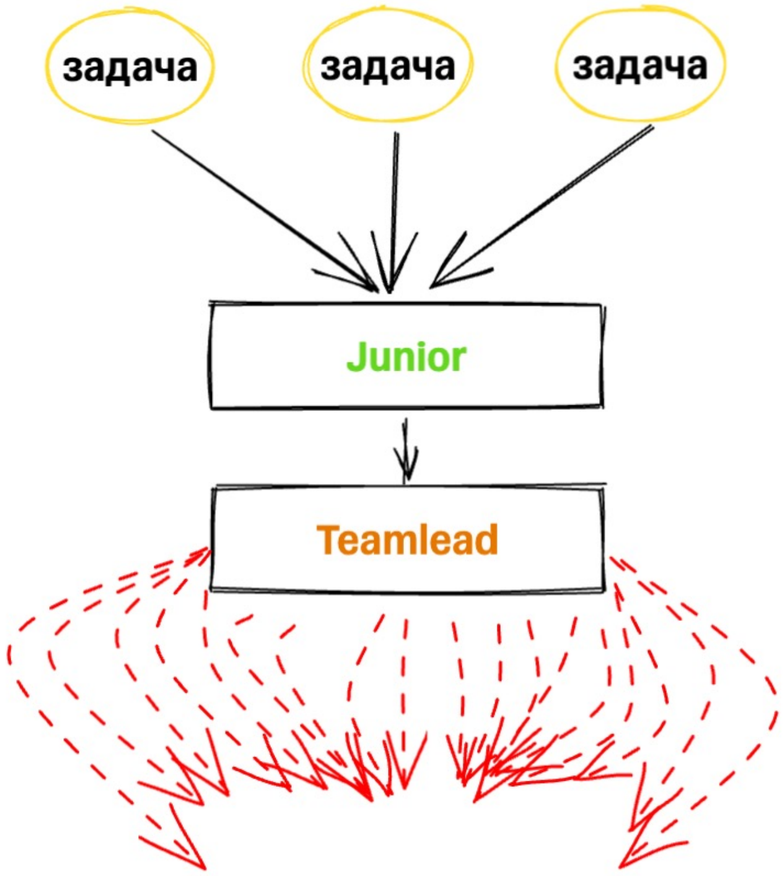
- Откатили релиз
- Перестали получать «тяжелые» запросы
- Тимлид – хороший. Тимлид запомнил



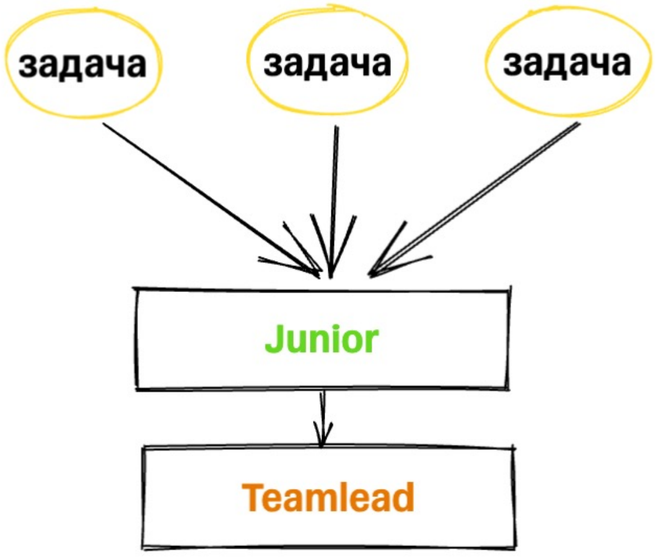
- Откатили релиз
- Перестали получать «тяжелые» запросы
- Тимлид – хороший. Тимлид запомнил



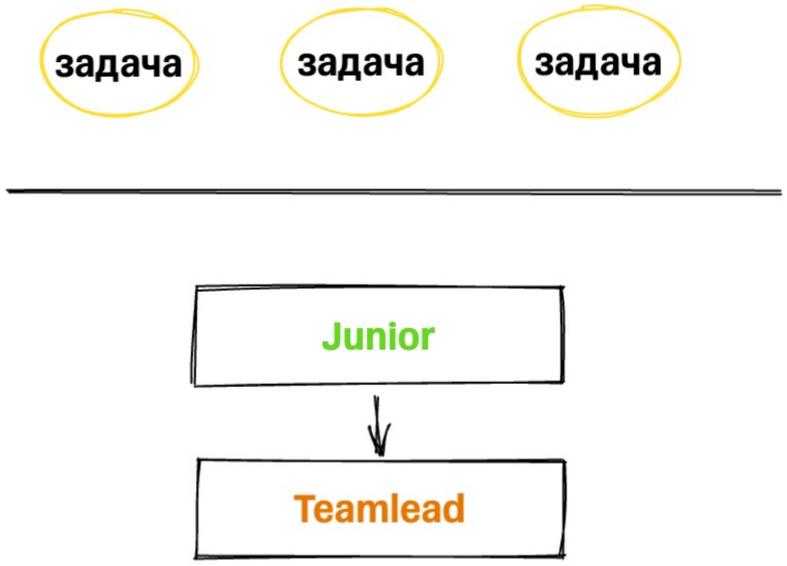
- Откатили релиз
- Перестали получать «тяжелые» запросы
- Тимлид – хороший. Тимлид запомнил



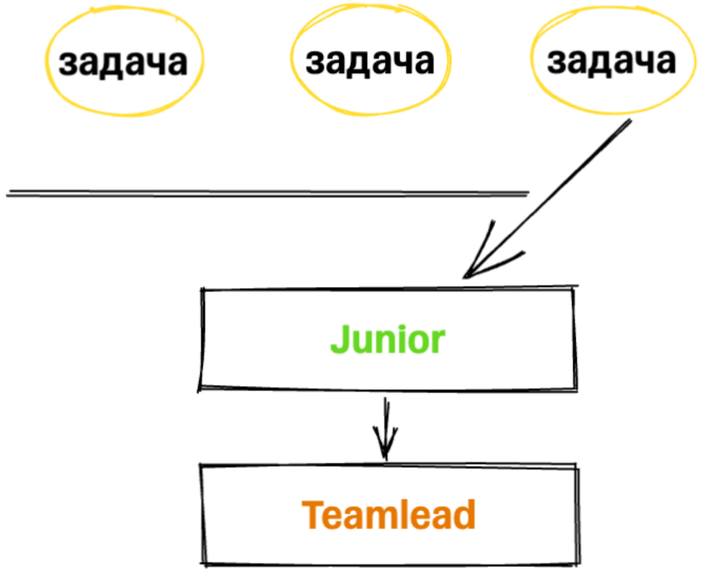
Потушили нагруженный сервис



- Потушили нагруженный сервис
- Очистили очередь



- Потушили нагруженный сервис
- Очистили очередь
- Прекратили поток запросов



- Потушили нагруженный сервис
- Очистили очередь
- Прекратили поток запросов
- Медленно запустили трафик

Итоги

Хотели как лучше...

Сделали retry

На случай отказа внешних систем

Не смогли подняться

Из-за волны ретраев

Как не допустить выгорание сеньора?

Circuit breaker

Понимает что внешняя система недоступна и рвет цепь. Периодически проверяет доступность. Возвращает трафик при нормализации работы внешней системы

Exponential backoff

Наращиваем таймауты для ретраев. Умножаем на случайный jitter. Это помогает равномернее распределить нагрузку от ретраев

Graceful degradation

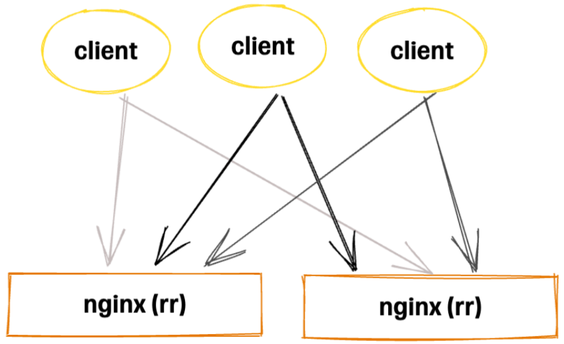
Если к нашему сервису обращаются: можем снижать качество или количество ответов при достижении определенного порога по нагрузке

Last In, First Out

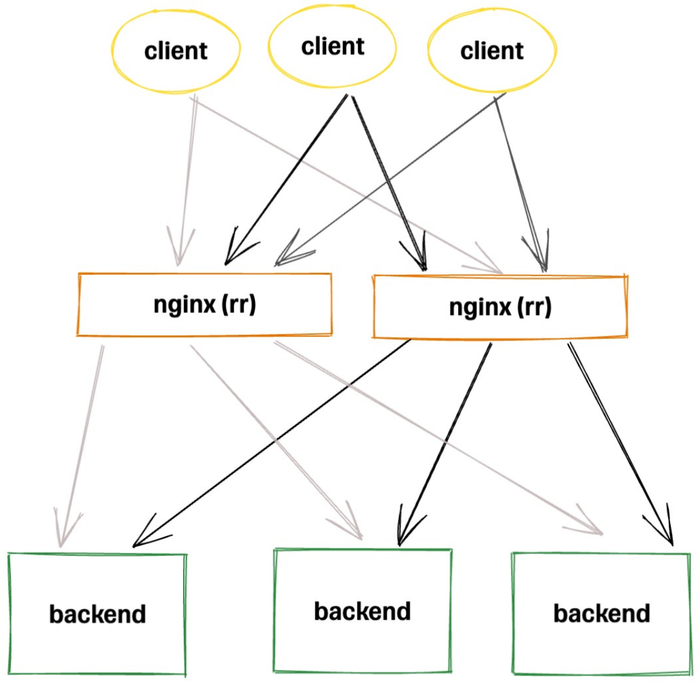
Нет смысла обрабатывать запрос на двенадцатой секунде, если клиент посчитает его неуспешным на десятой

2. Слишком умная балансировка

Есть клиент. Не важно какой



- Есть клиент. Не важно какой
- Есть балансировщики Nginx



- Есть клиент. Не важно какой
- Есть балансировщики Nginx
- Есть backend

Ручной привод

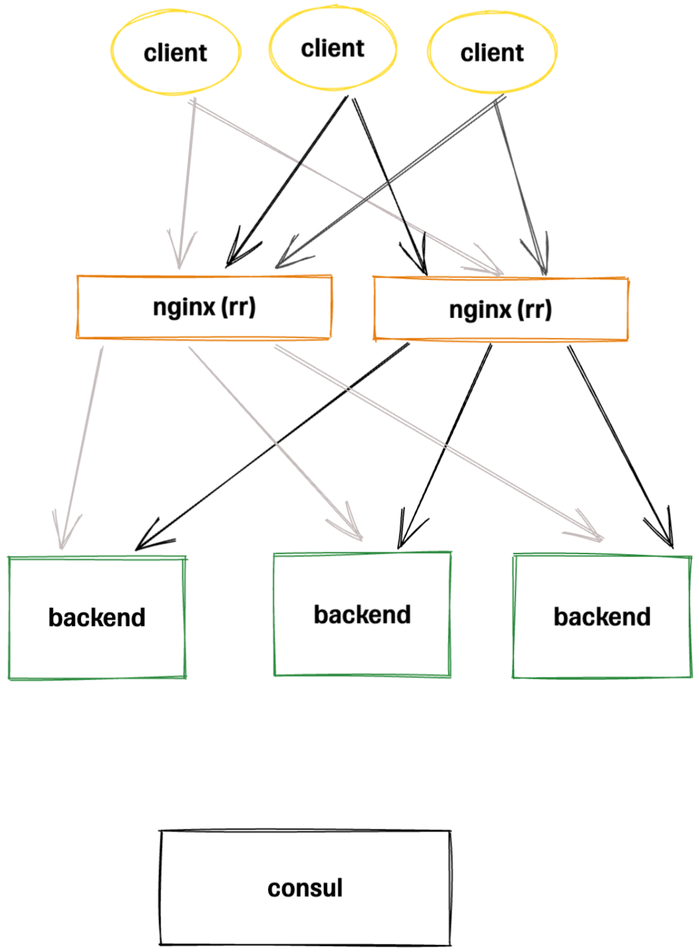
Конфиг Nginx - статичен

Service discovery

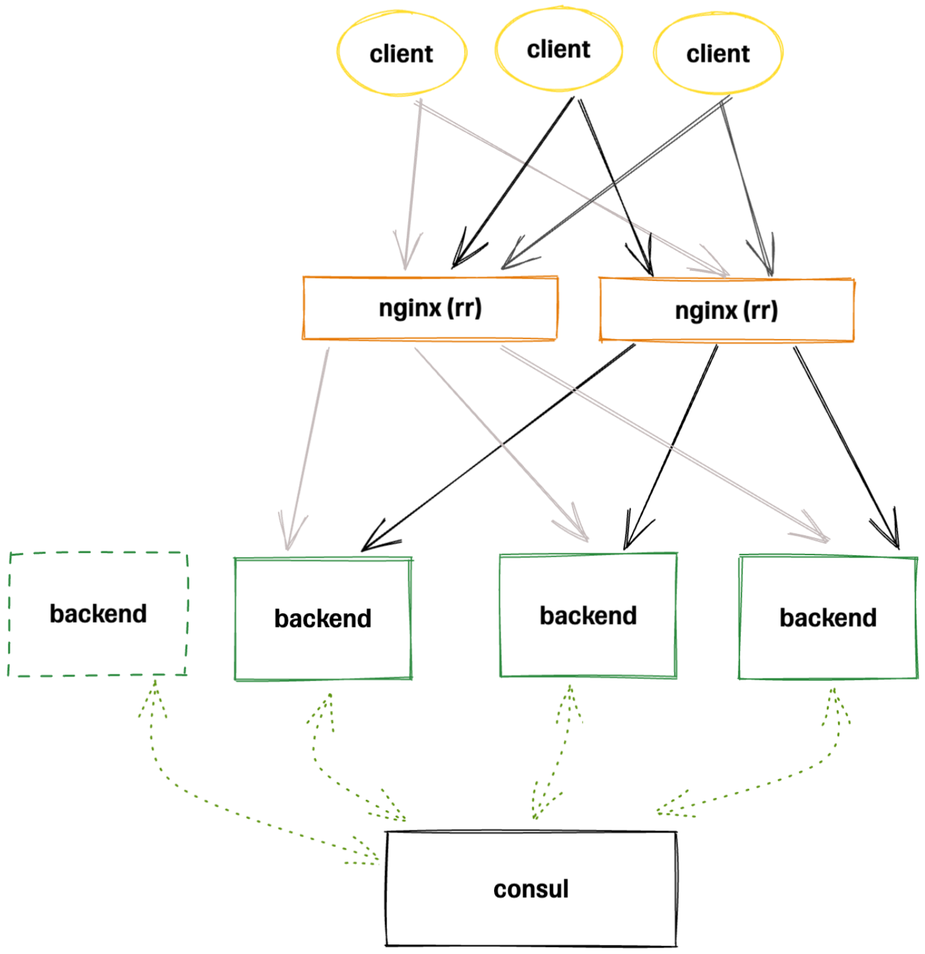
Добавлять и удалять серверы из апстримов автоматически – отличная идея

Healthchecks

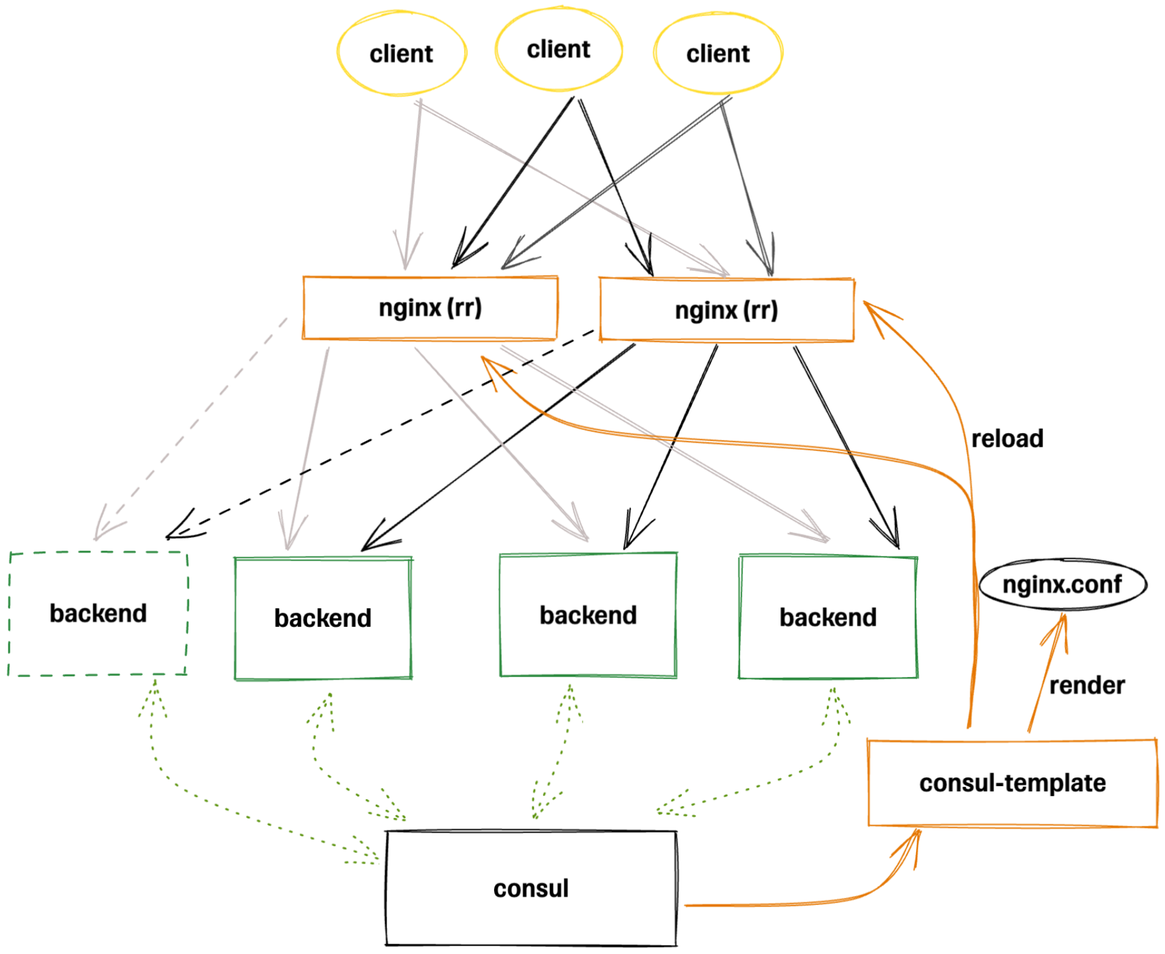
Еще лучше – автоматически убирать «больной» сервер из балансировки


client
client
client
nginx (rr)
nginx (rr)
backend
backend
backend
consul

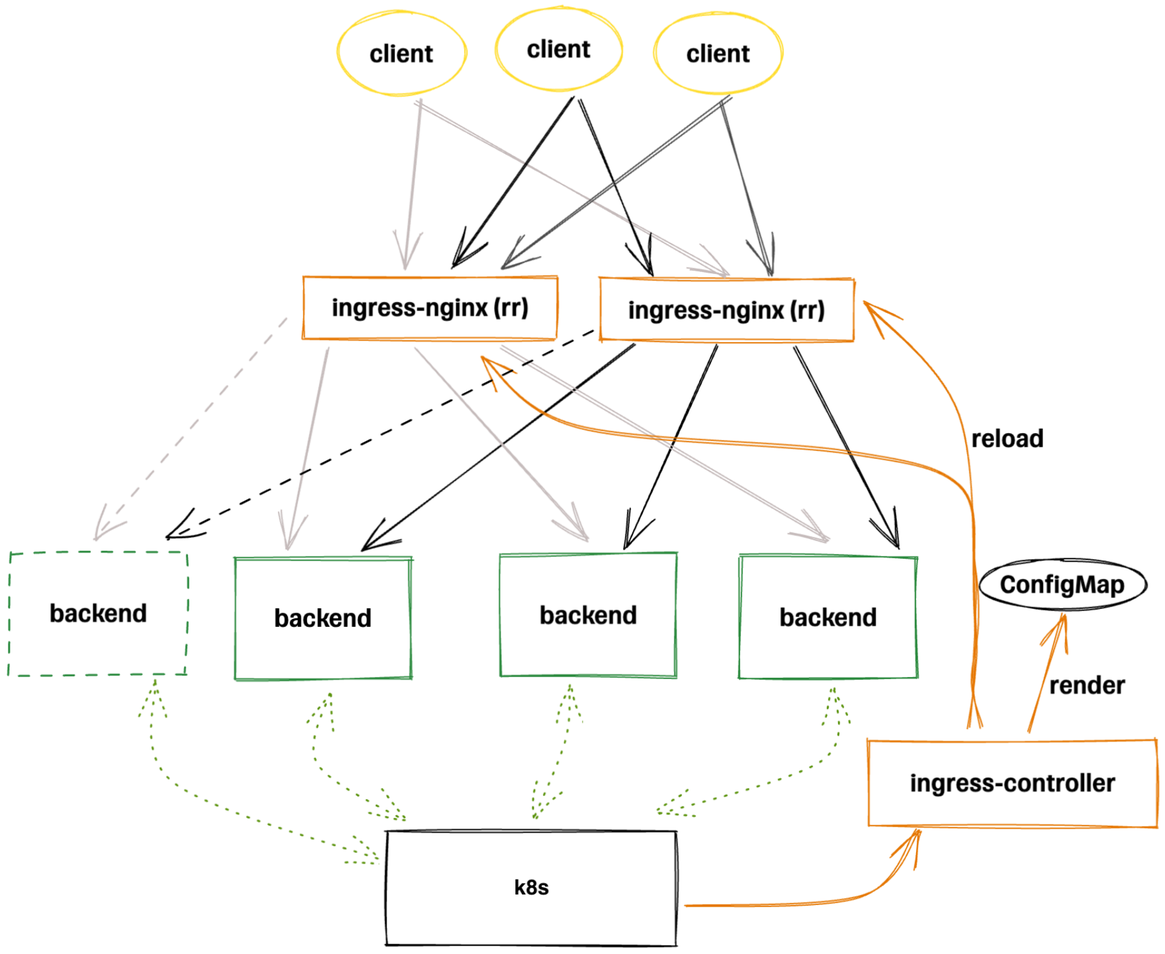
Consul



- Consul
- SD + Healthchecks



- Consul
- SD + Healthchecks
- Consul-template



- Consul
- SD + Healthchecks
- Consul-template

Все работает идеально...

Toil elliminated

Никаких ручных правок. Конфиги меняются, Nginx подхватывает. Инженеры освобождены от ручного труда

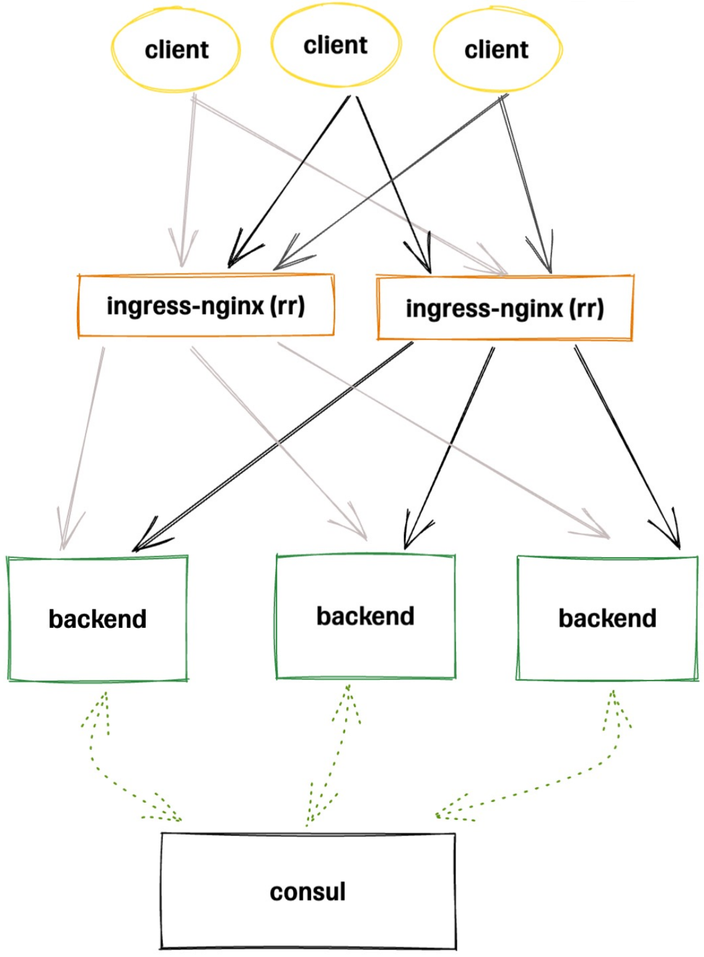
Zero-downtime релизы

Без участия инженеров

Апстримы обновляются

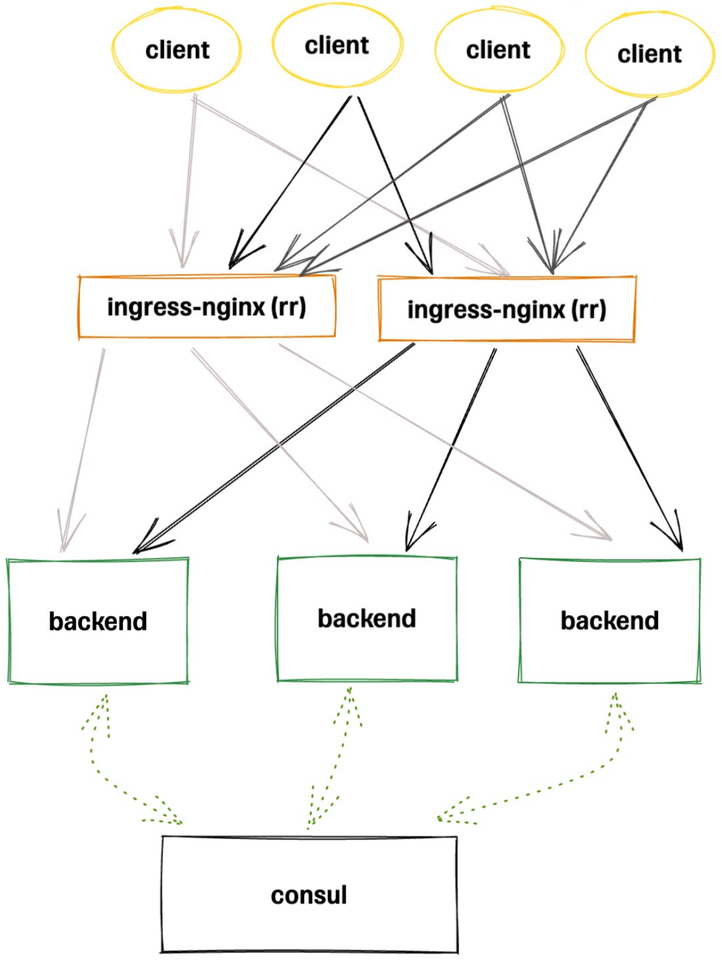
Ввод и вывод из эксплуатации – на основании данных из consul

До одного момента



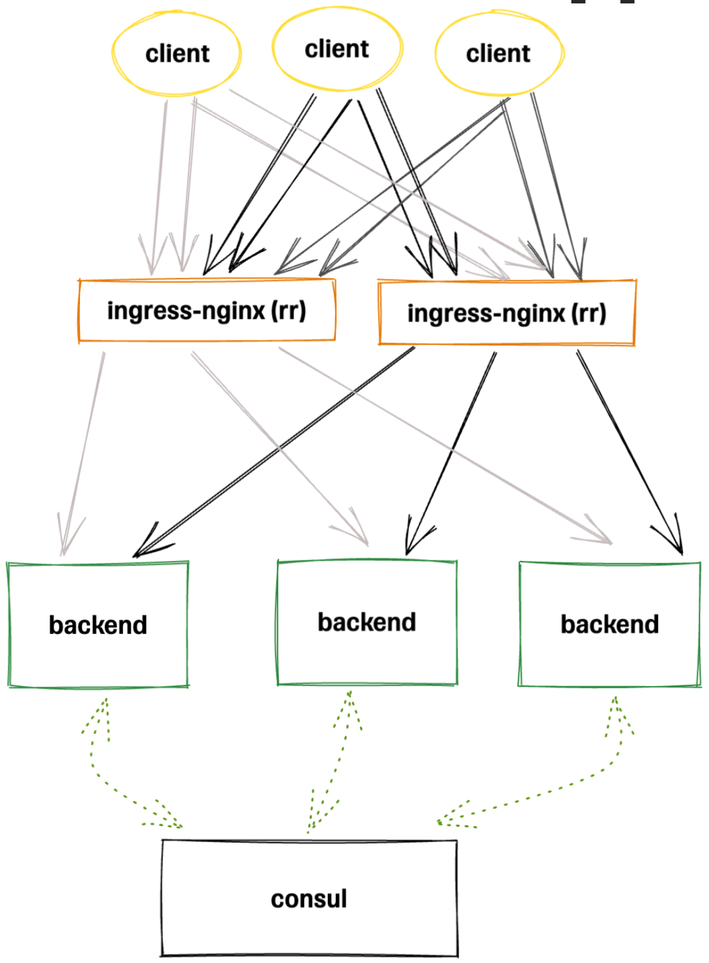
Что-то случилось

До одного момента



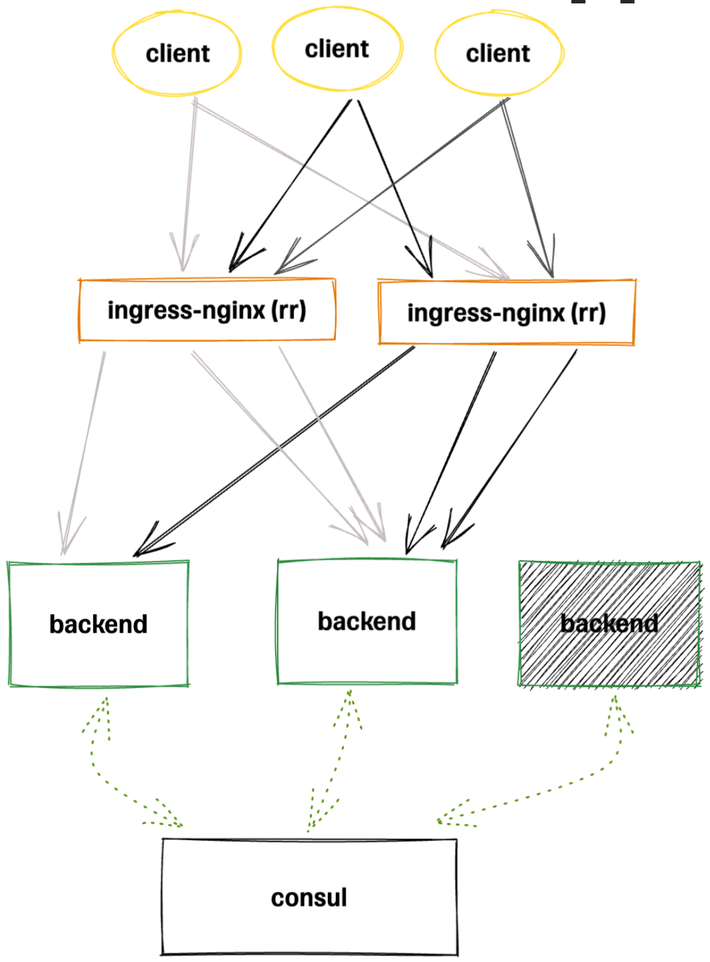
Что-то случилось

До одного момента



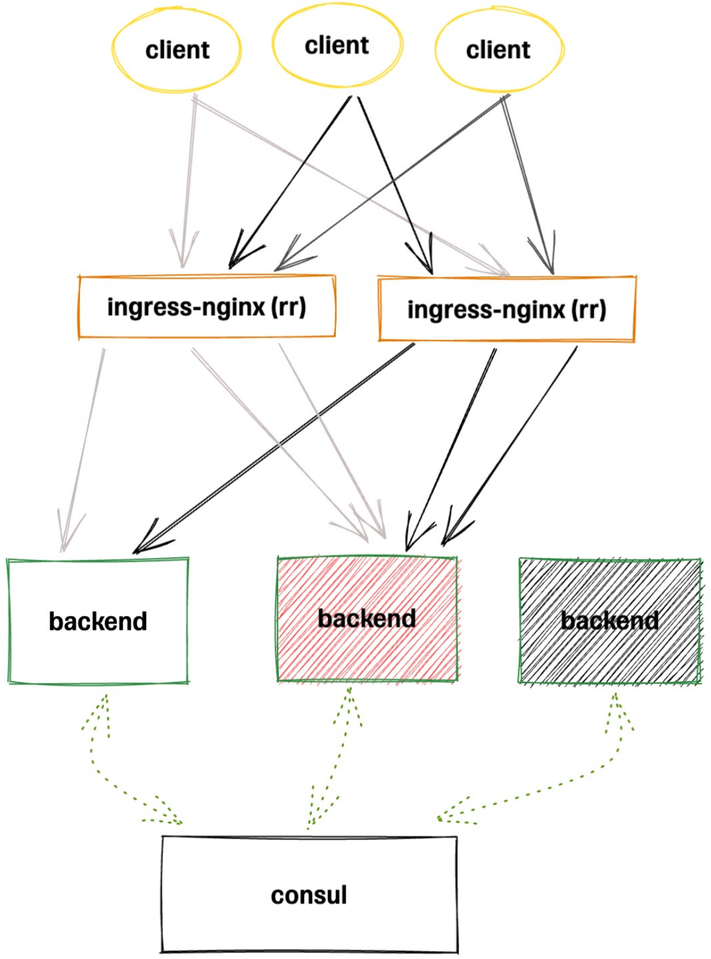
Что-то случилось

До одного момента



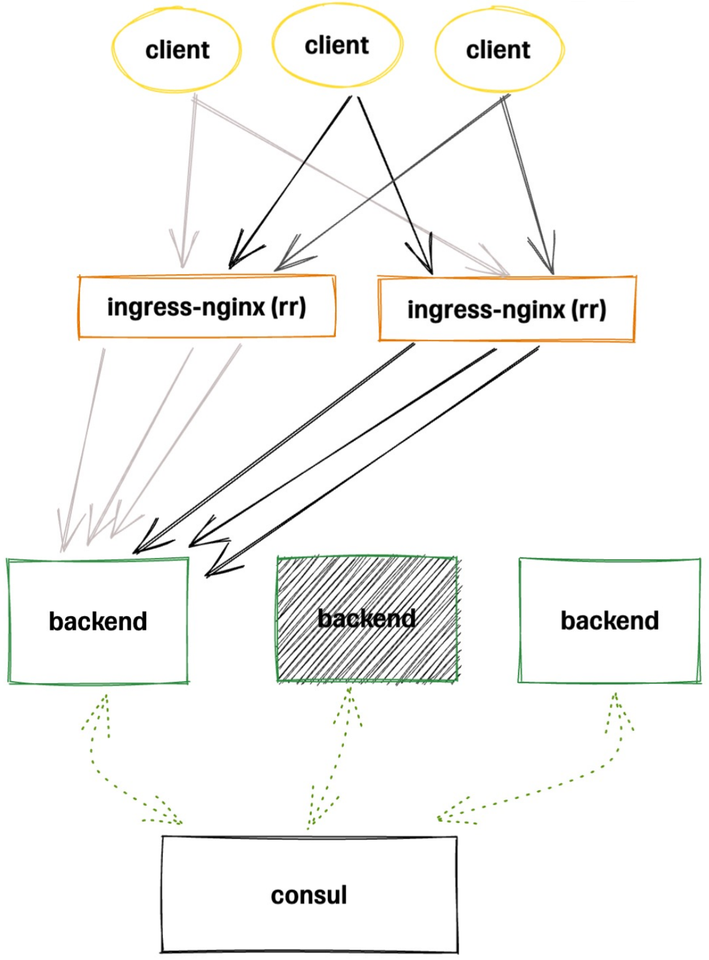
Что-то случилось

До одного момента



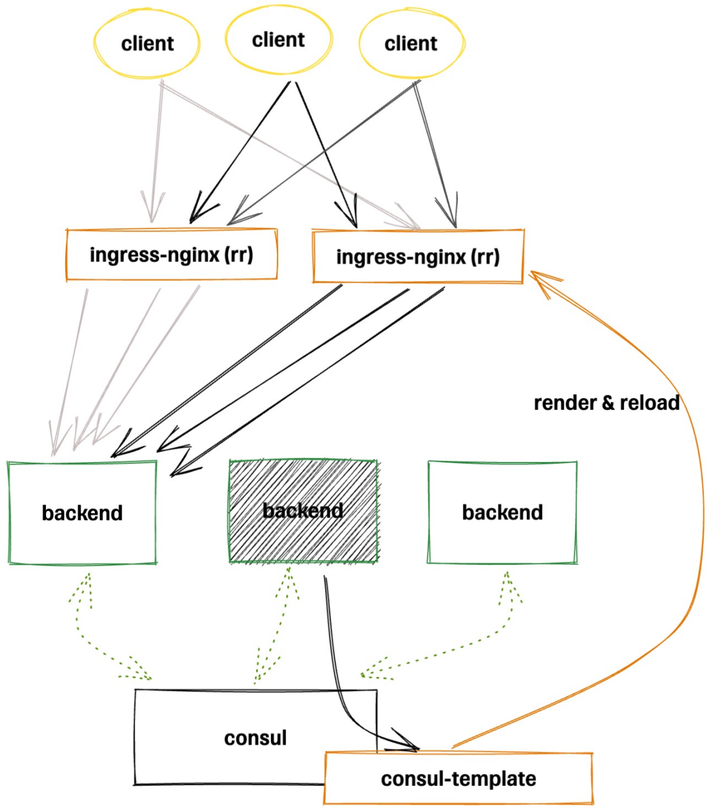
- Что-то случилось
- Нагрузка выросла

До одного момента



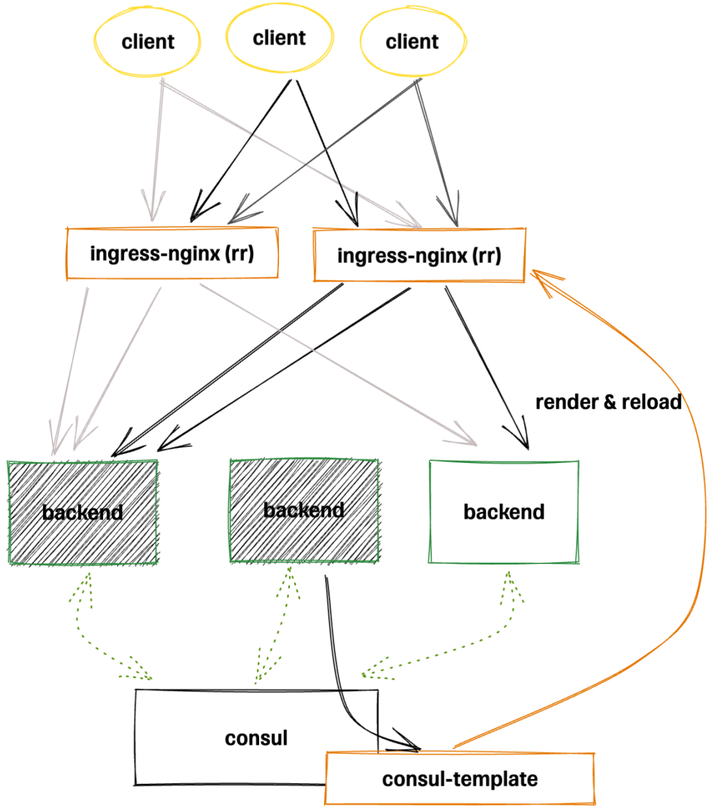
- Что-то случилось
- Нагрузка выросла
- Реплика бэкенда упала

Consul - молодец



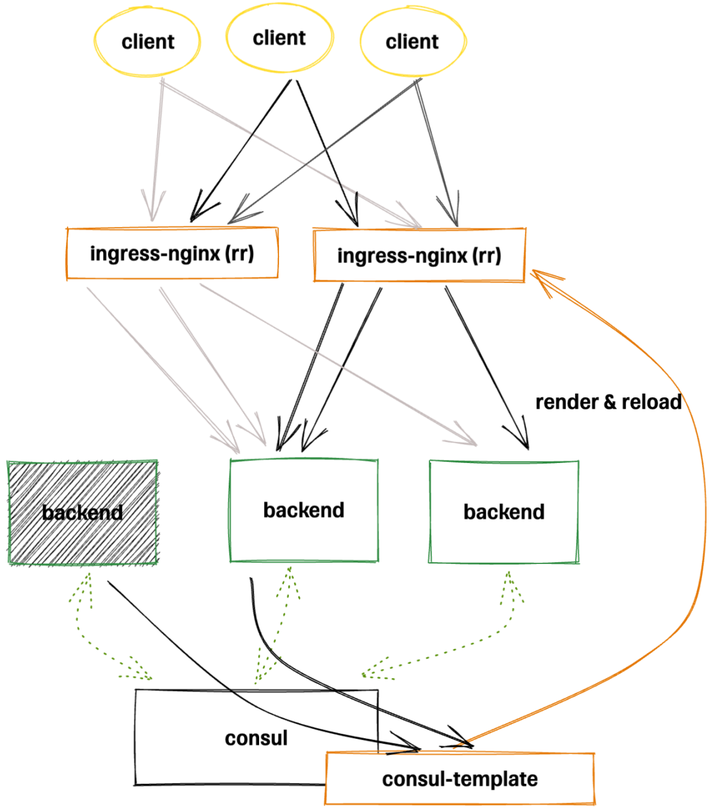
Нагрузка ушла на соседнюю

Consul - молодец



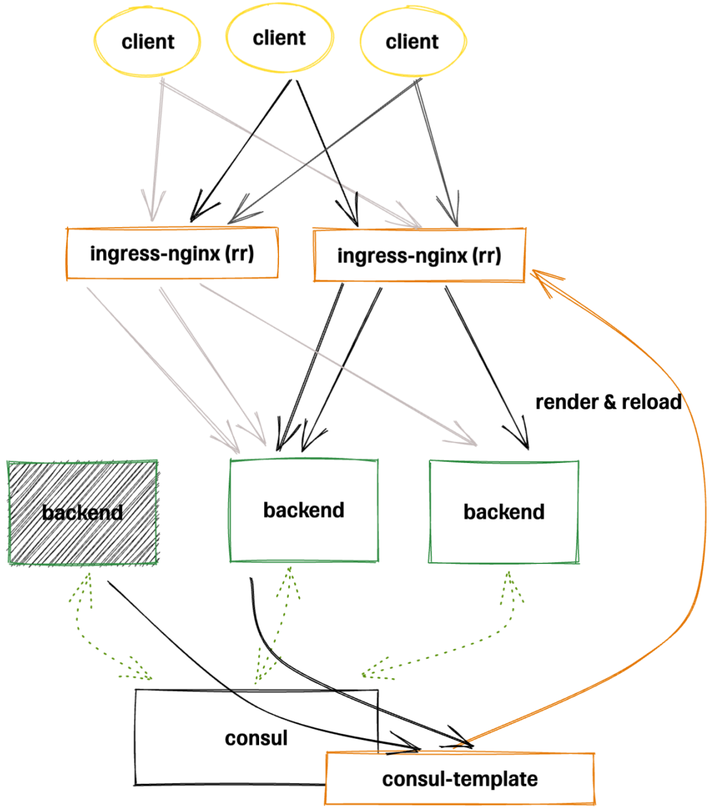
- Нагрузка ушла на соседнюю
- Другая реплика тоже упала

Consul - молодец



- Нагрузка ушла на соседнюю
- Другая реплика тоже упала
- Первая реплика поднялась

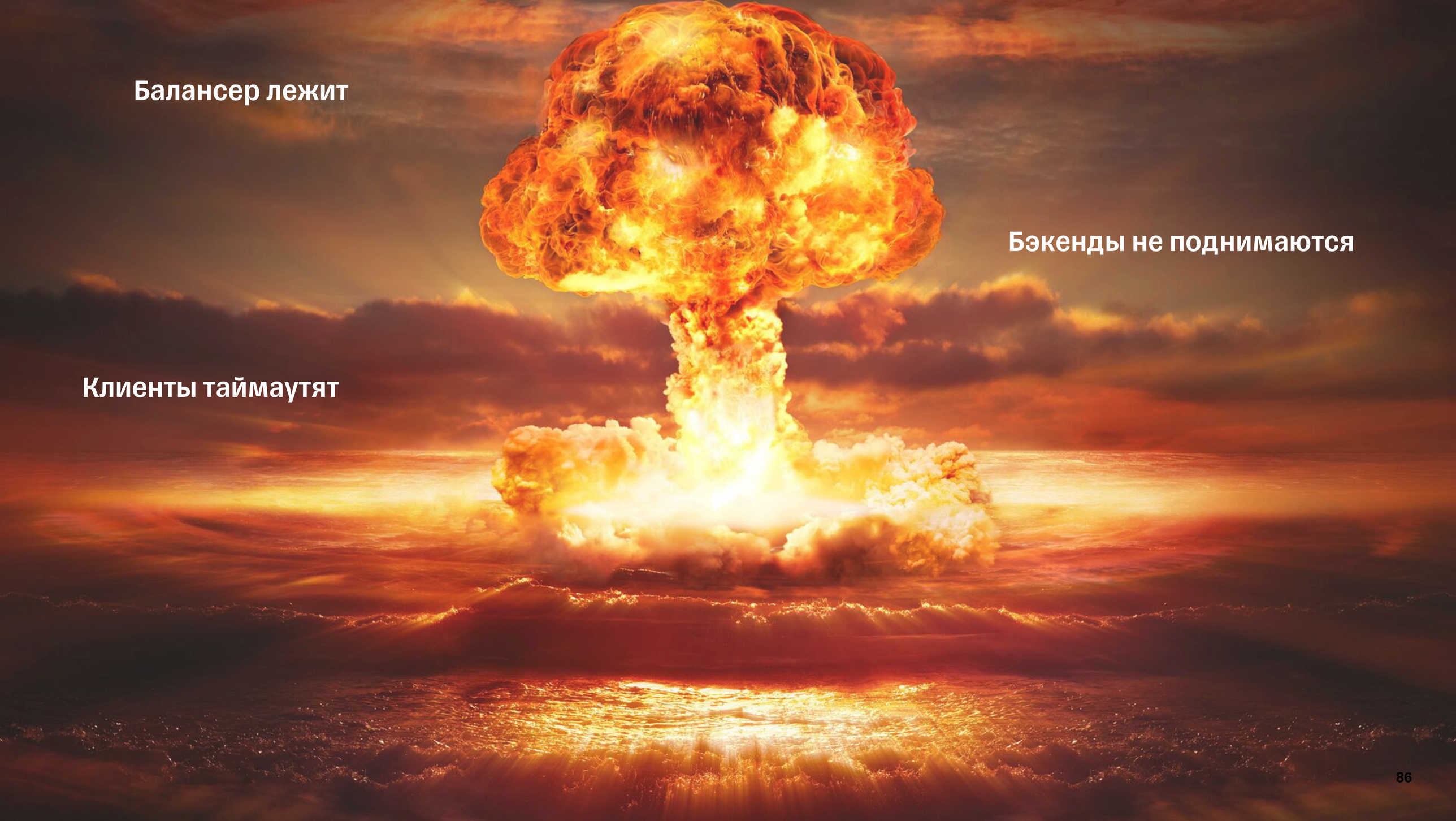
Consul - молодец



- Нагрузка ушла на соседнюю
- Другая реплика тоже упала
- Первая реплика поднялась
- В целом, все хорошо

Балансер лежит


Балансер лежит
Бэкенды не поднимаются


Балансер лежит
Бэкенды не поднимаются
Клиенты таймаутят


Балансер лежит
Бэкенды не поднимаются
Клиенты таймаутят
Поддержку порвало

Troubleshooting

- Экспоненциальный рост нагрузки

Troubleshooting

- ✕ Экспоненциальный рост нагрузки
- ✕ Рост коннектов на nginx

Troubleshooting

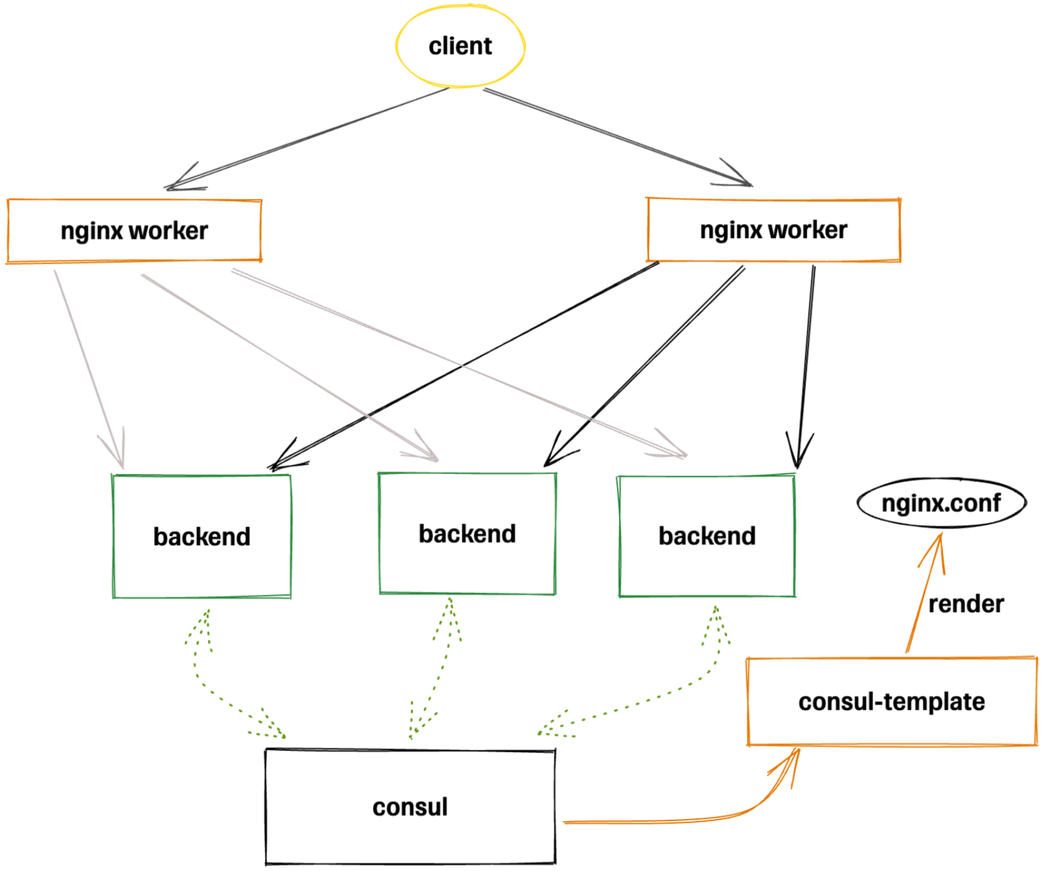
- Экспоненциальный рост нагрузки
- Рост коннектов на nginx
- Бэкенд сложился под нагрузкой x10

Опять retry

- Опять retry
- Почему росли коннекты?

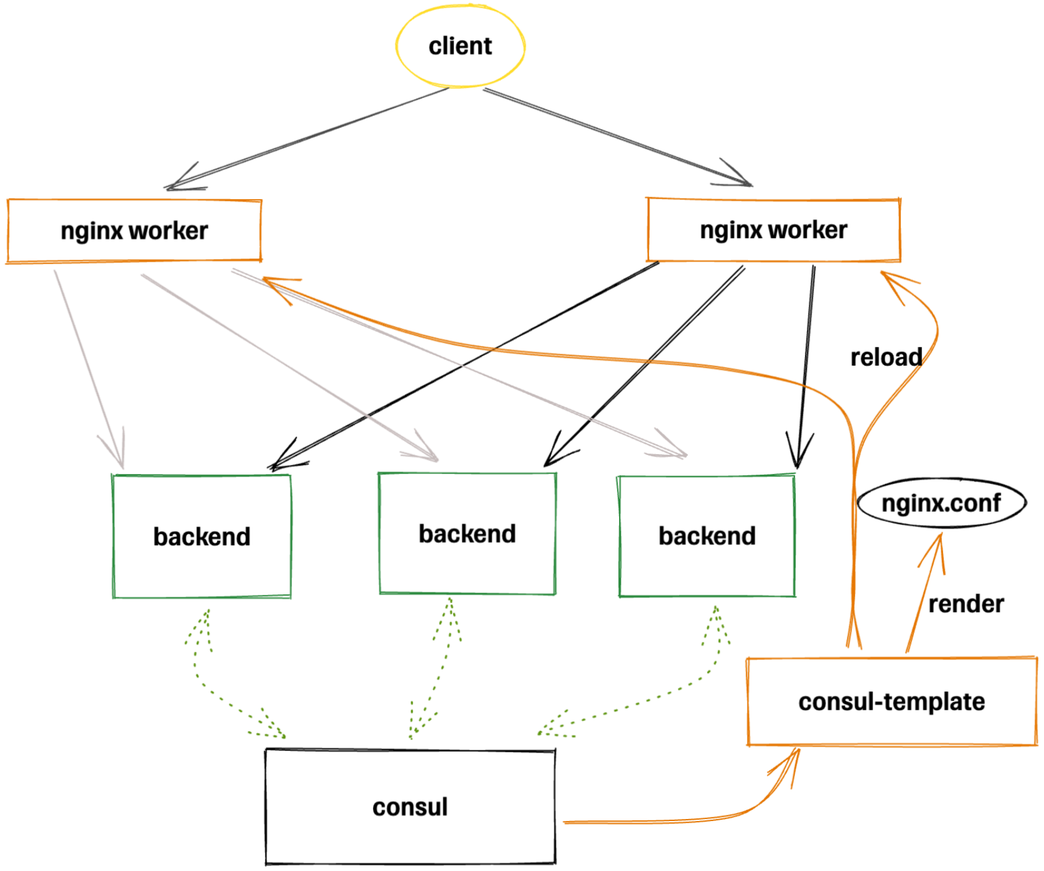
- Опять retry
- Почему росли коннекты?
- Почему умер nginx?

Consul SD + Nginx



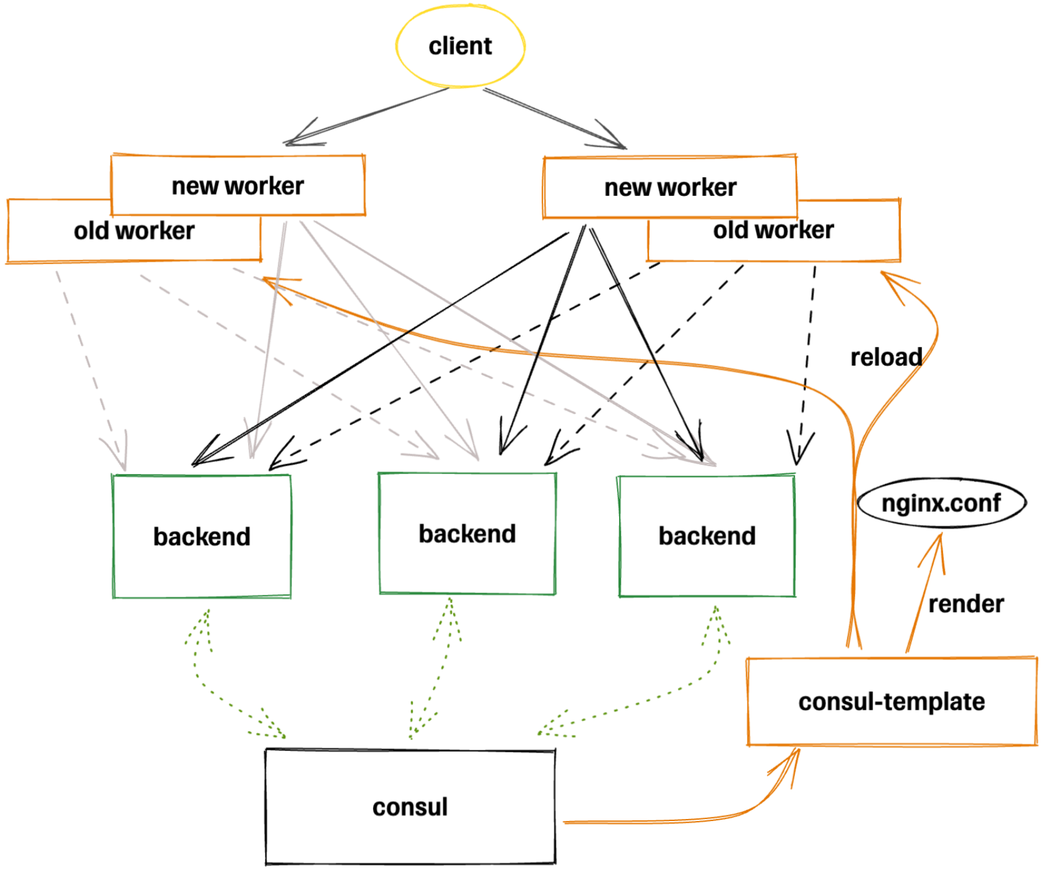
Конфиг статичный

Consul SD + Nginx



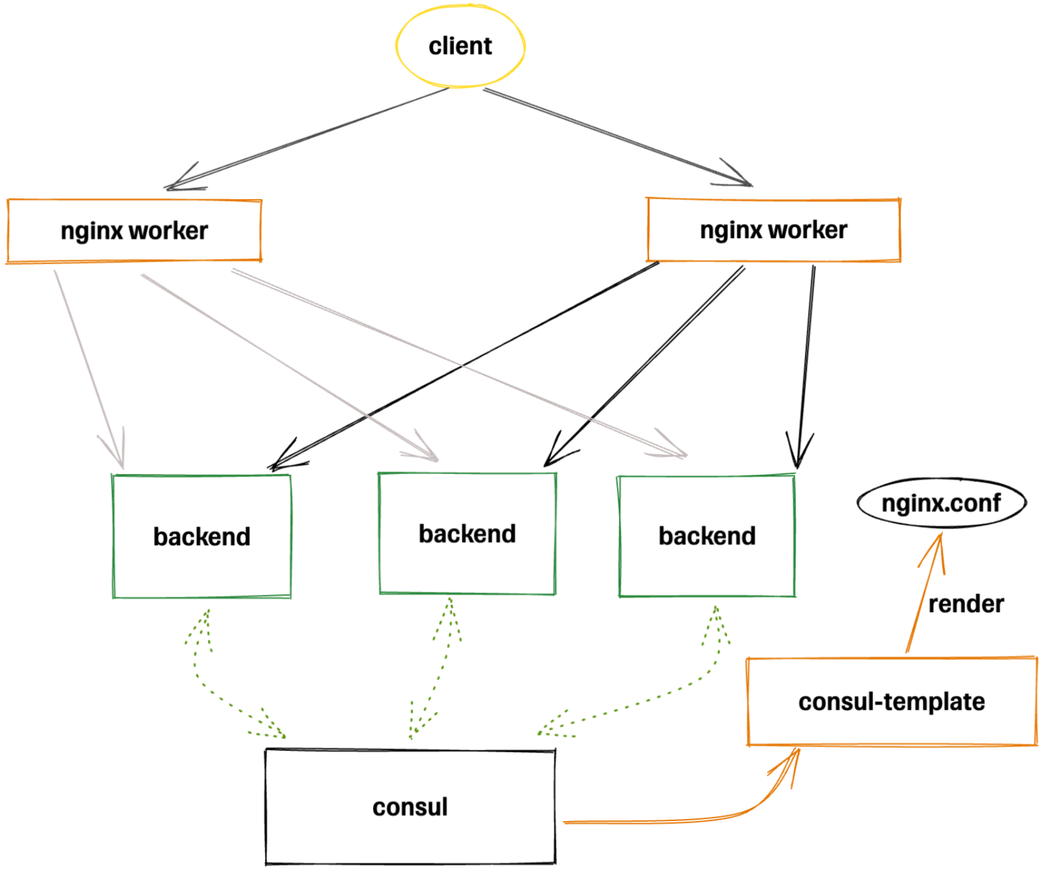
- Конфиг статичный
- Обновление через reload

Consul SD + Nginx



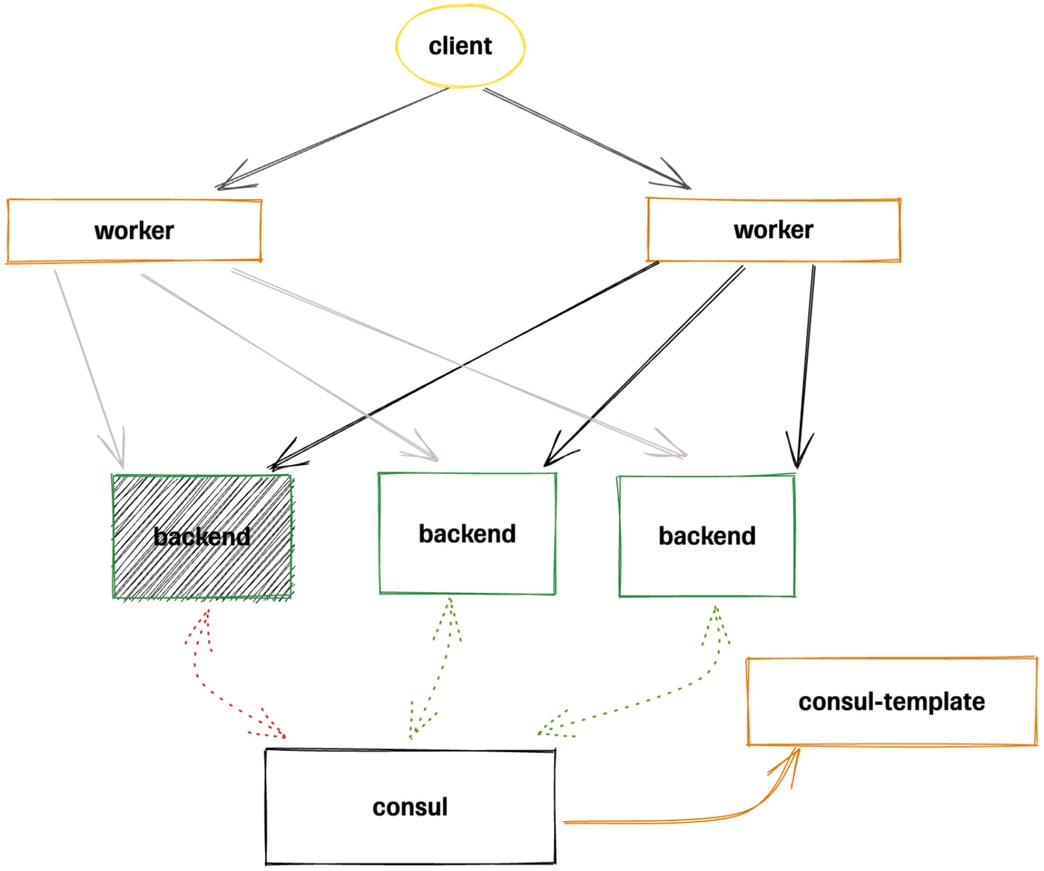
- Конфиг статичный
- Обновление через reload
- Reload порождает процесс

Consul SD + Nginx

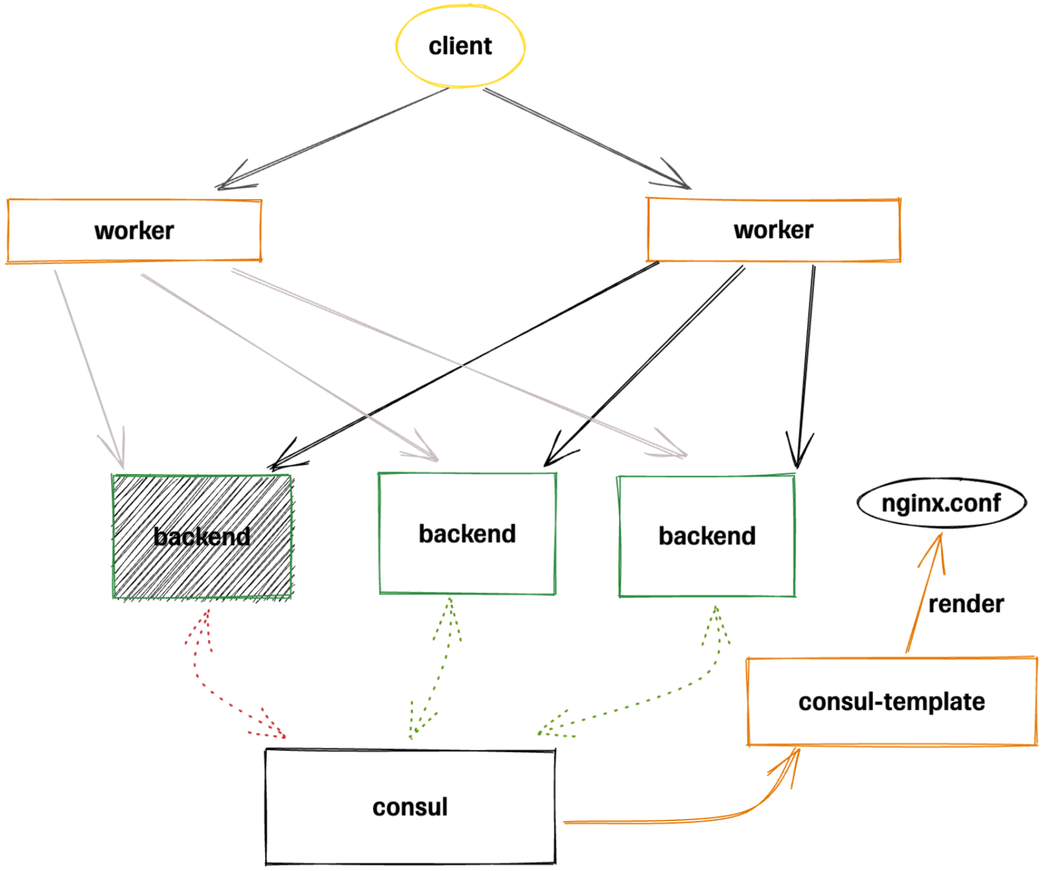


- Конфиг статичный
- Обновление через reload
- Reload порождает процесс
- Старый процесс завершается

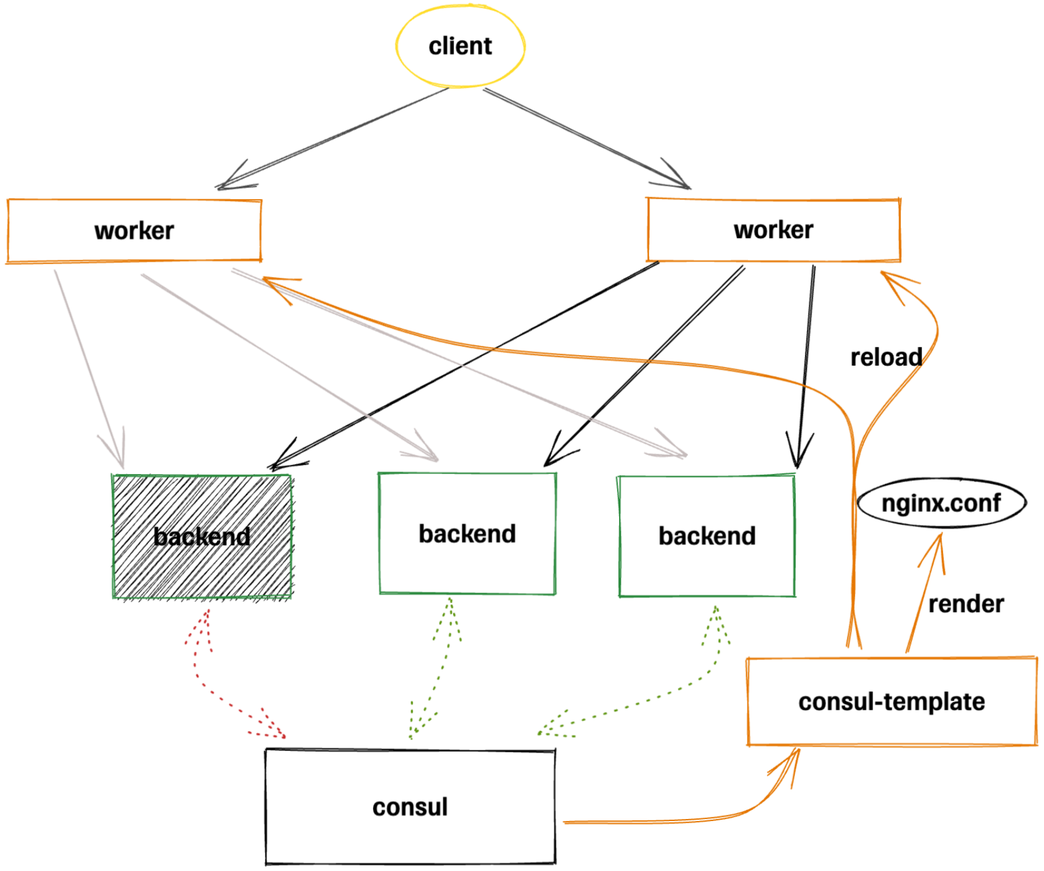
И почему же все сломалось?



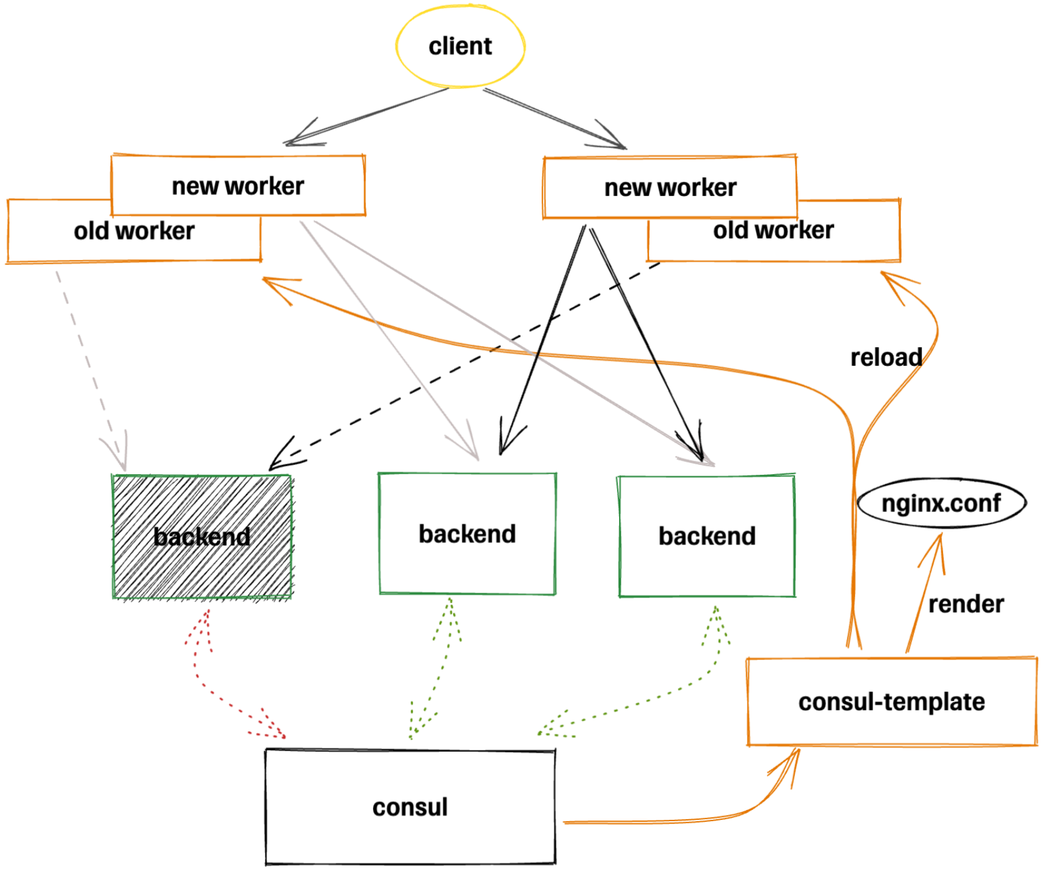
Consul видит failed healthcheck



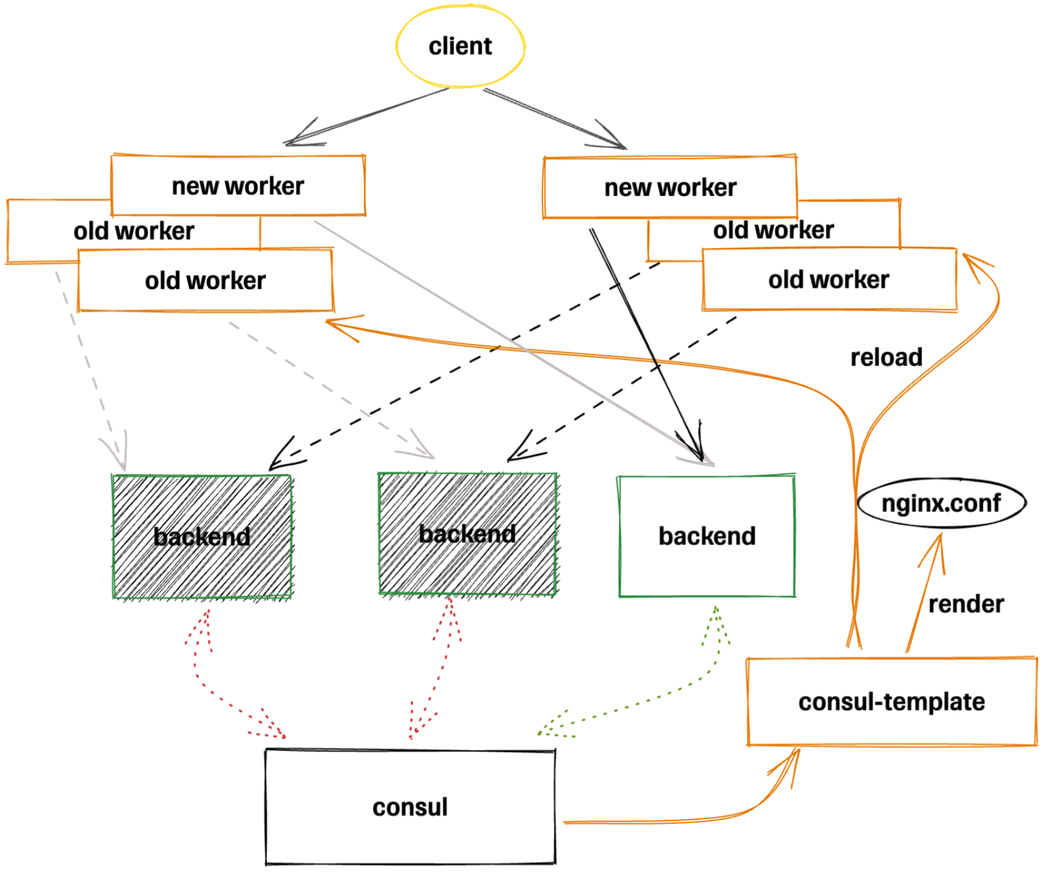
- Consul видит failed healthcheck
- Consul-template рендерит конфиг



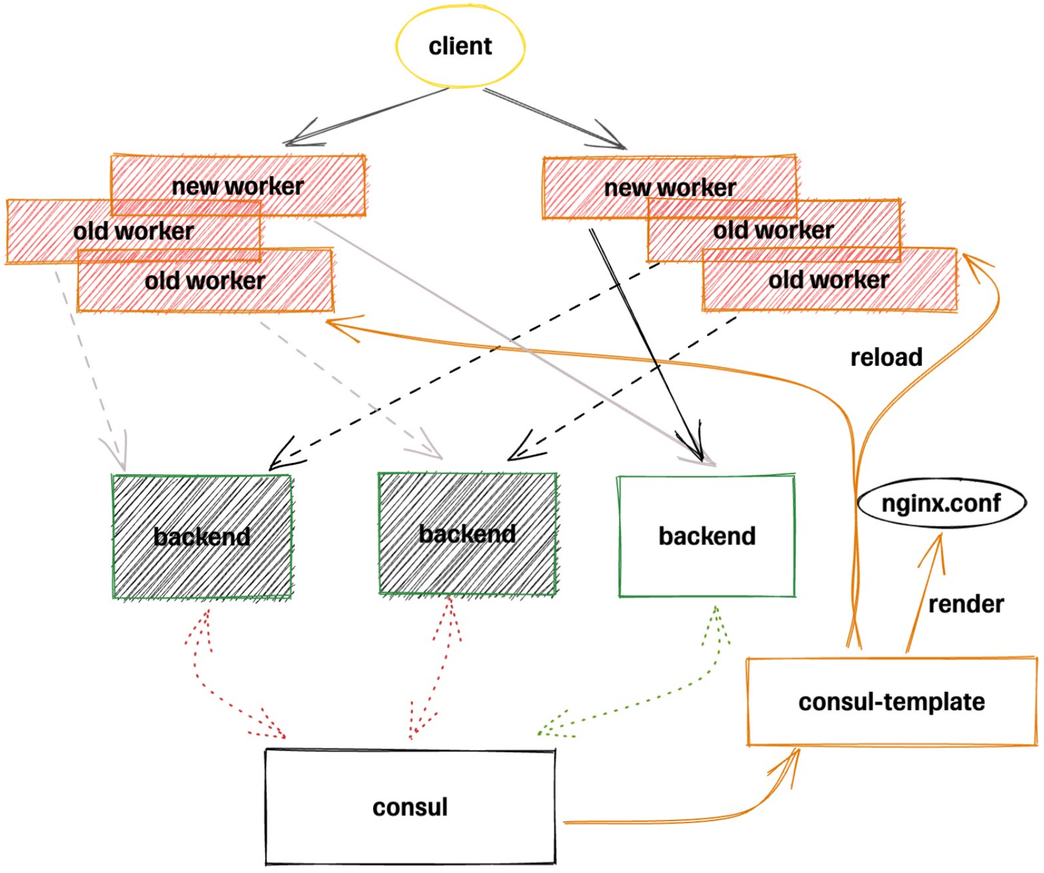
- Consul видит failed healthcheck
- Consul-template рендерит конфиг
- Сервис релоадится



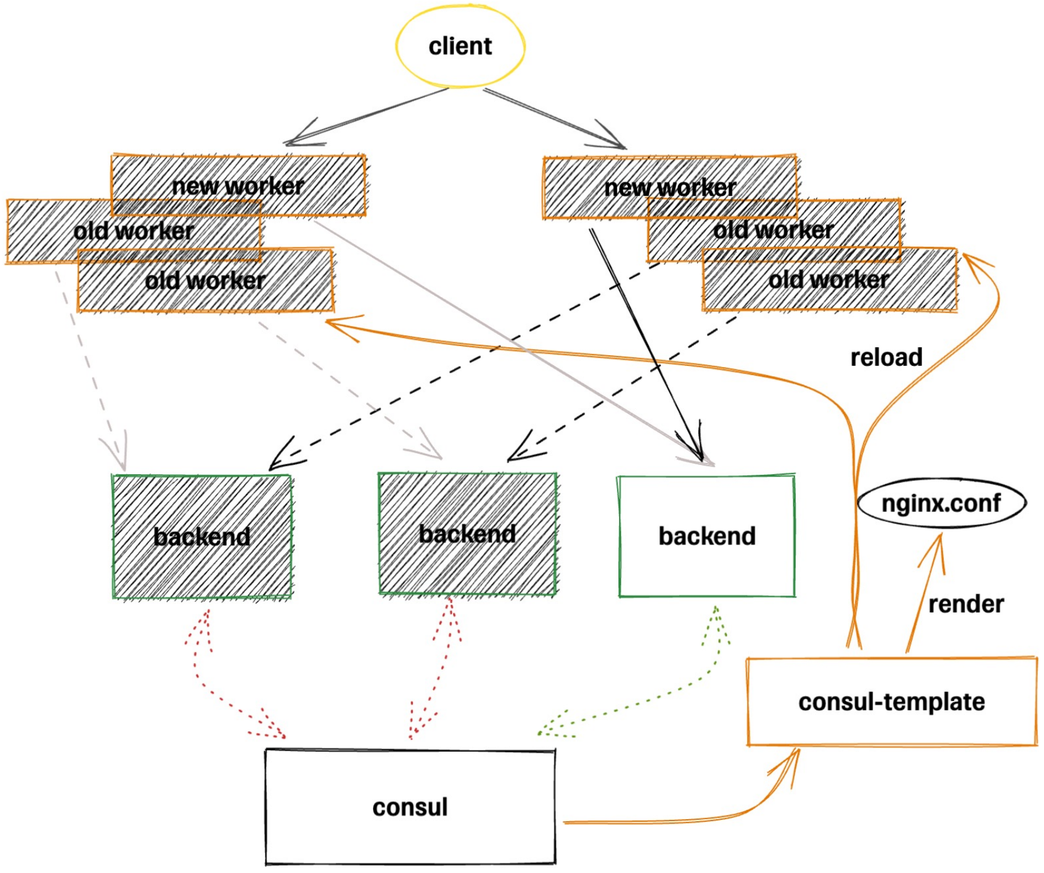
- Consul видит failed healthcheck
- Consul-template рендерит конфиг
- Сервис релоадится
- Старый воркер остается



- Количество релоадов - сотни
- Новые воркеры плодятся
- Старые – держат коннекты



- Количество релоадов - сотни
- Новые воркеры плодятся
- Старые – держат коннекты
- Ресурсы nginx - кончаются



- Количество релоадов - сотни
- Новые воркеры плодятся
- Старые – держат коннекты
- Ресурсы nginx - кончаются
- Nginx перестает отвечать

Directed by
ROBERT B. WEIDE

Итоги

Хотели как лучше...

Сделали service-discovery

Чтобы автоматически отрабатывать отказ реплики

Не смогли подняться

Из-за множественных релоадов и исчерпания ресурсов на Nginx

Что с этим делать?

Менять конфиг в рантайме

Избегать решений, требующих релоада конфига. Примеры: Envoy, Nginx Plus, HAProxy Runtime API etc.

Active healthcheck

Nginx умеет определять недоступность сервиса и не отправлять трафик на ноду, если хелсчек упал

Мониторинг

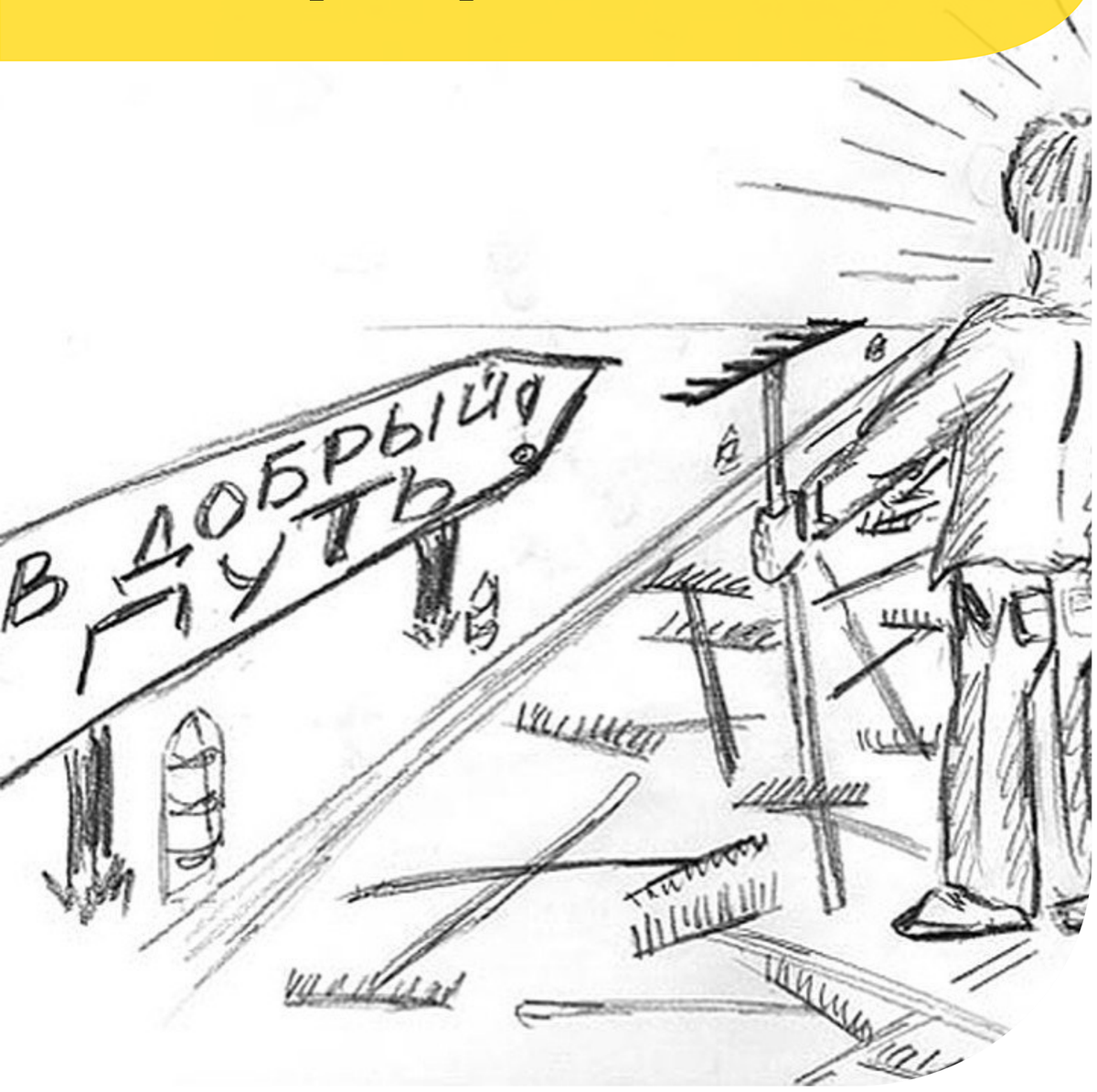
Можно следить за количеством релоадов и реагировать до того как ситуация стала критической

Ограничения

Регулируем частоту изменений конфигурации, не допускаем частых событий релоада

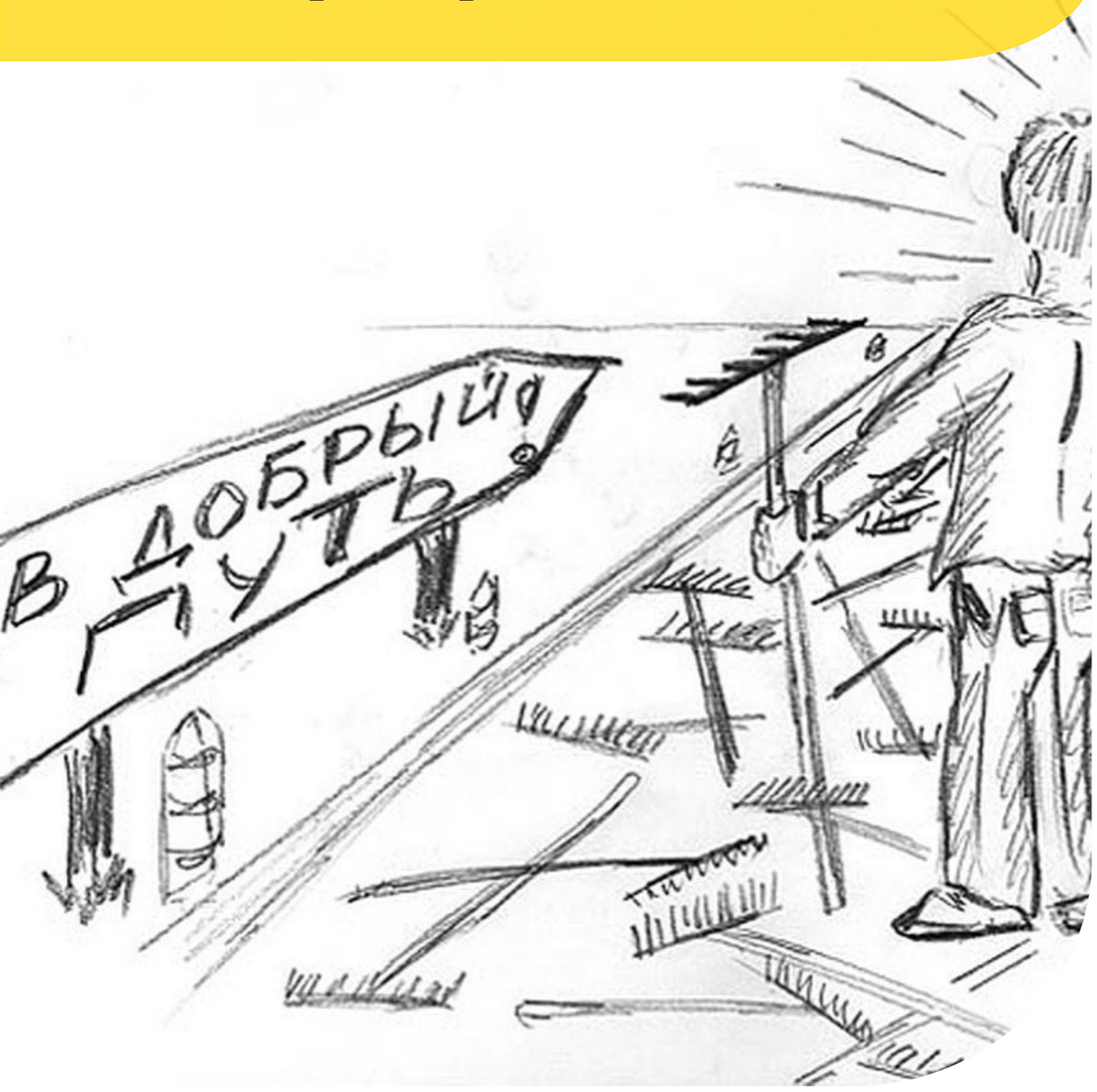
3. Доверительное резервирование

Резервирование



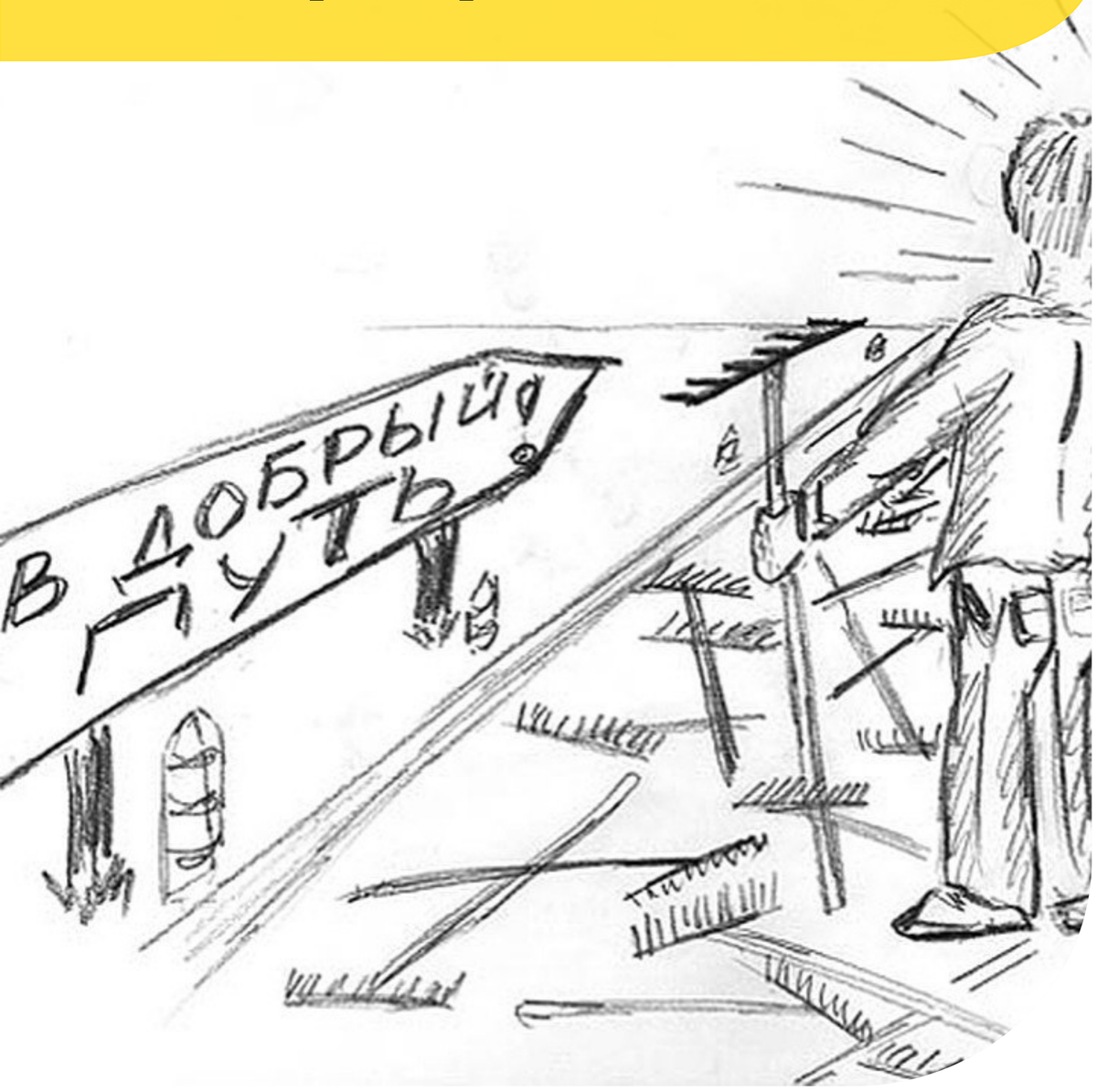
- Кейсы скучные

Резервирование



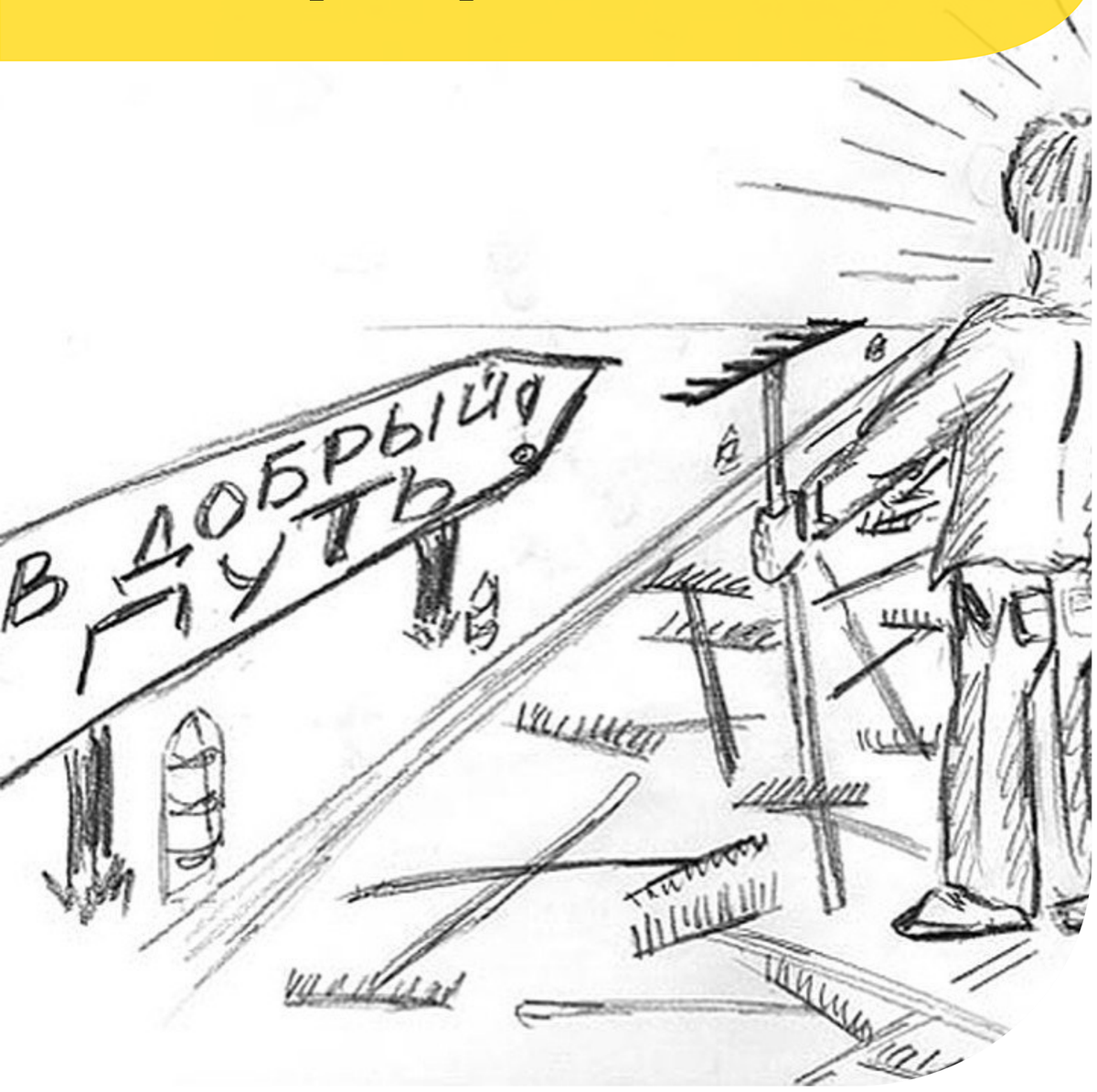
- Кейсы скучные
- Встречаются часто

Резервирование

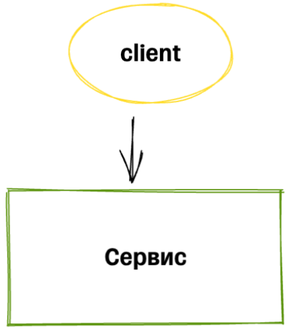


- Кейсы скучные
- Встречаются часто
- Ошибки минорные

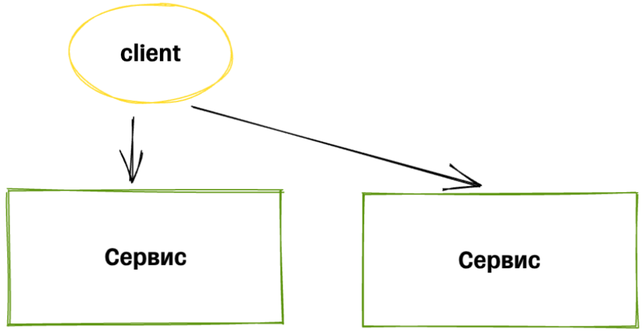
Резервирование



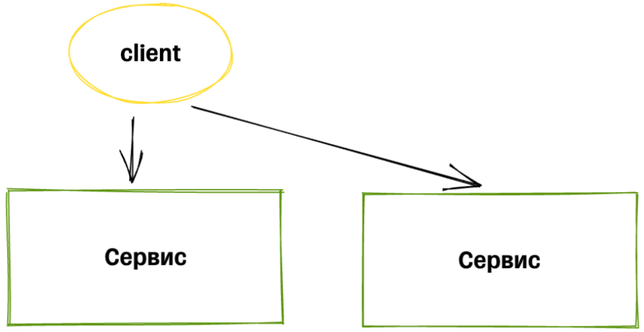
- Кейсы скучные
- Встречаются часто
- Ошибки минорные
- Последствия - мажорные



Наш сервис не идеален

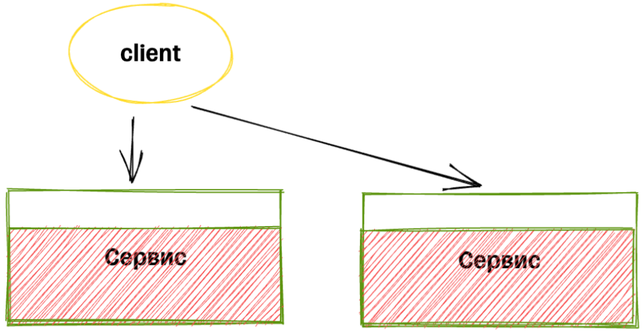


- Наш сервис не идеален
- Масштабируем!

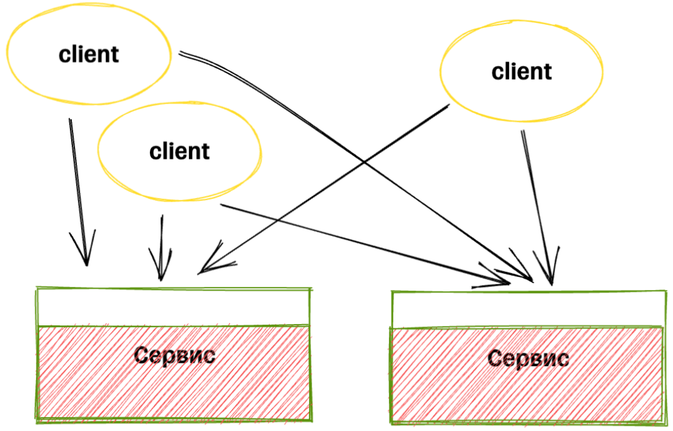


- Наш сервис не идеален
- Масштабируем!
- И верим...

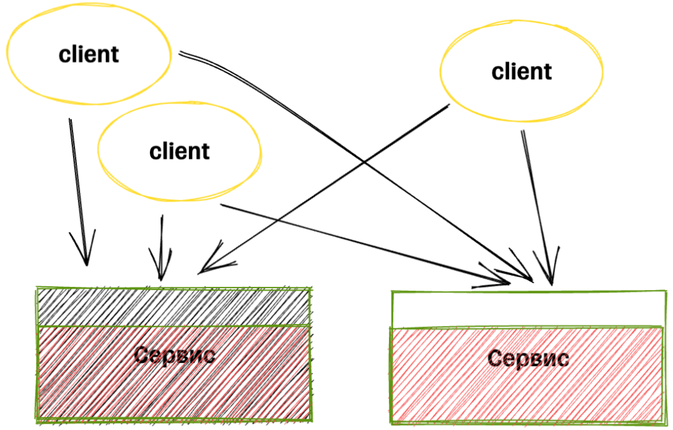
Failover роль утеряна...



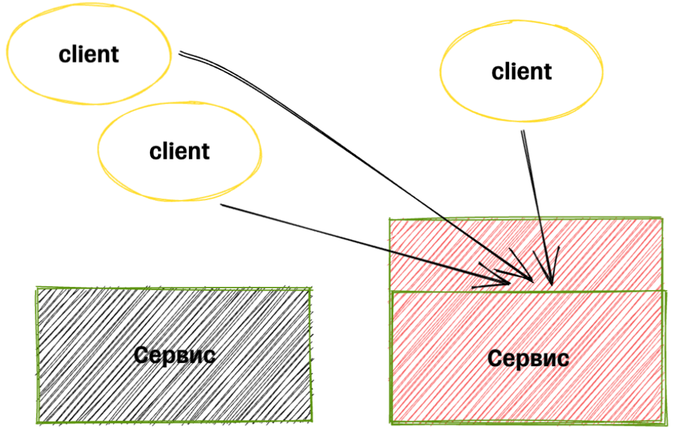
Утилизация >50%



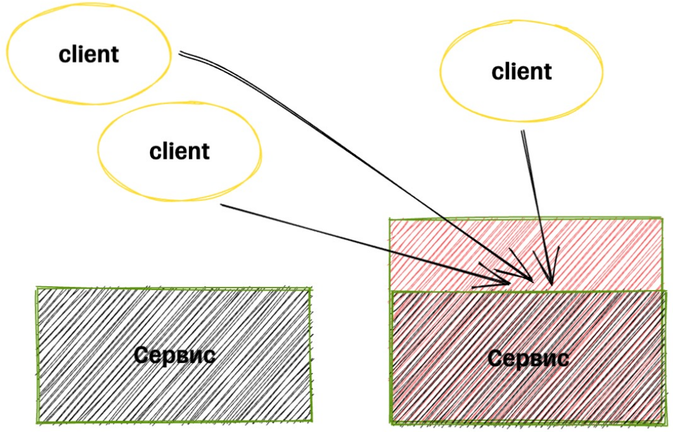
- Утилизация >50%
- Что-то случилось...



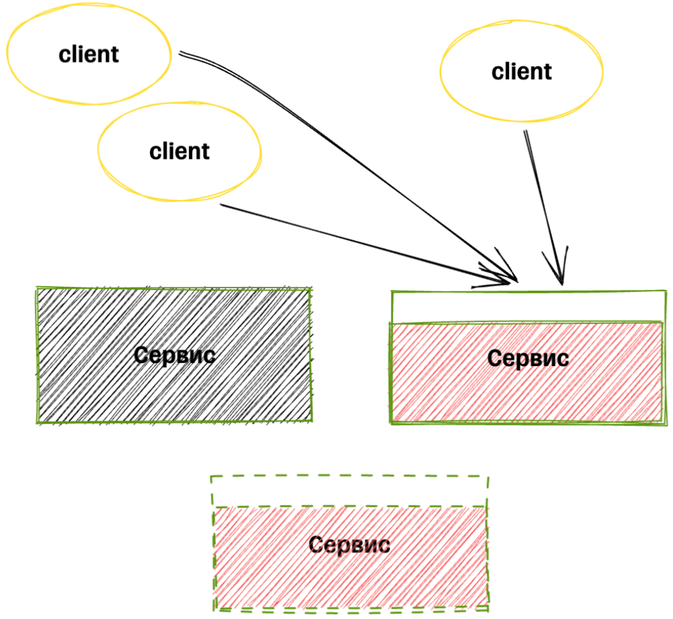
- Утилизация >50%
- Что-то случилось...
- Реплика упала



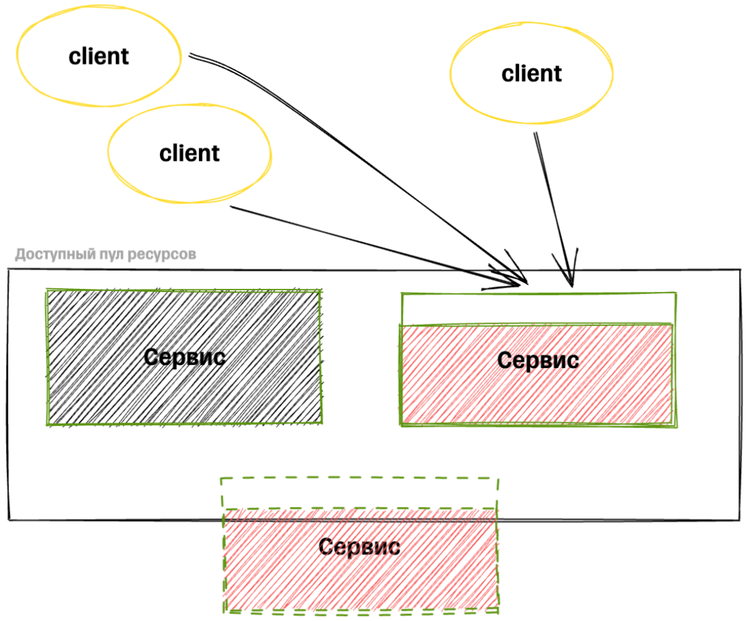
- Утилизация >50%
- Что-то случилось...
- Реплика упала



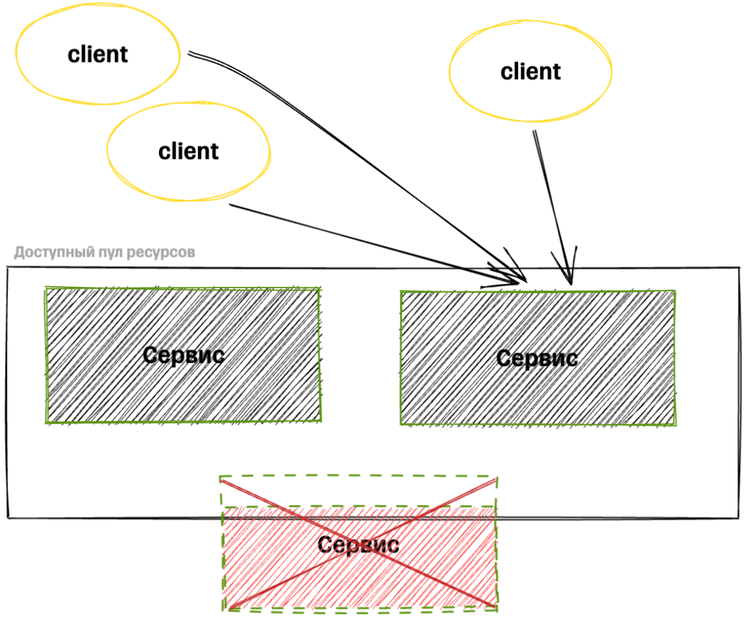
- Утилизация >50%
- Что-то случилось...
- Реплика упала
- Вторая не вынесла нагрузки



- Утилизация >50%
- Что-то случилось...
- Реплика упала
- Вторая не вынесла нагрузки
- Масштабируем!

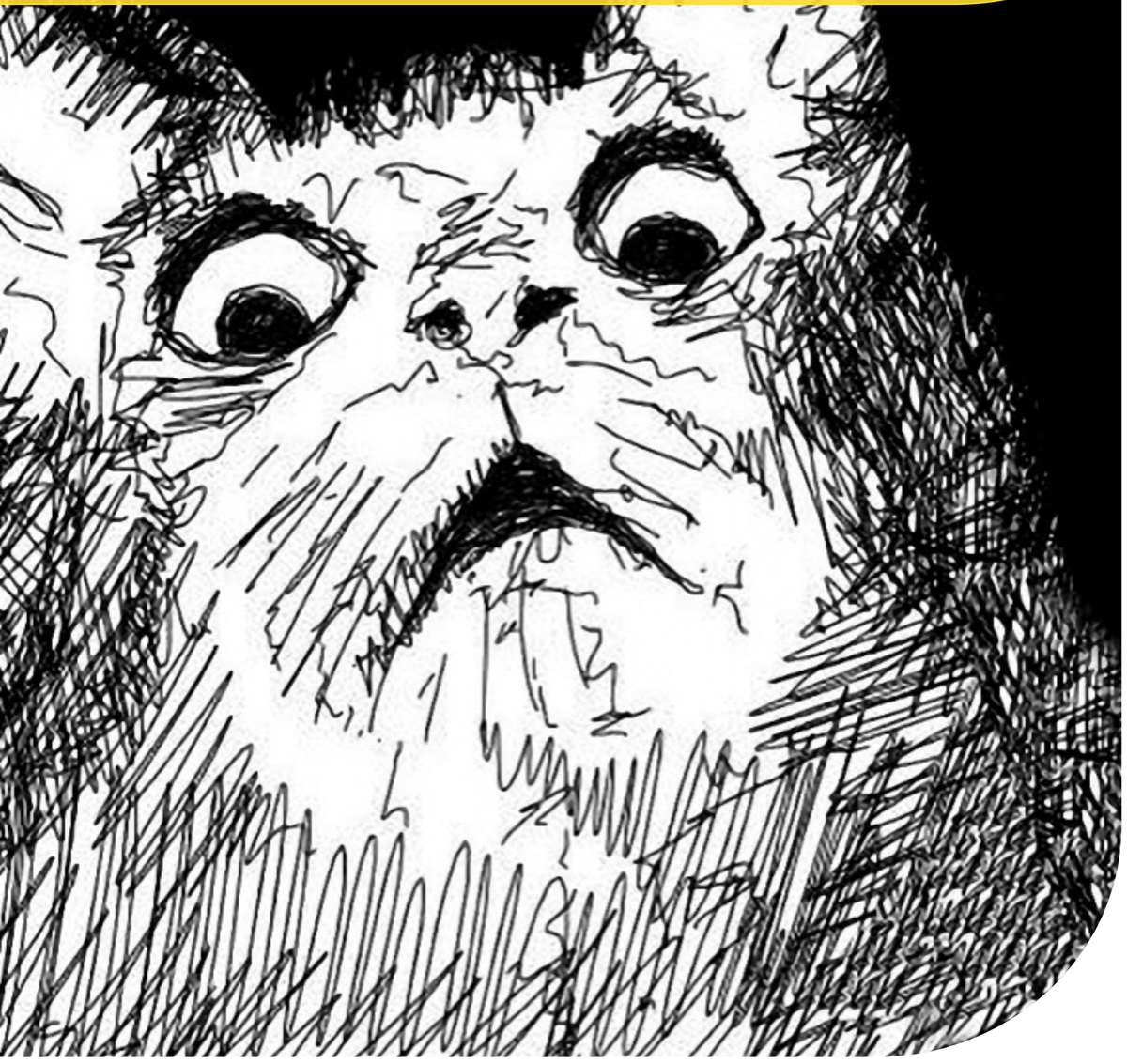


- Утилизация >50%
- Что-то случилось...
- Реплика упала
- Вторая не вынесла нагрузки
- Масштабируем!



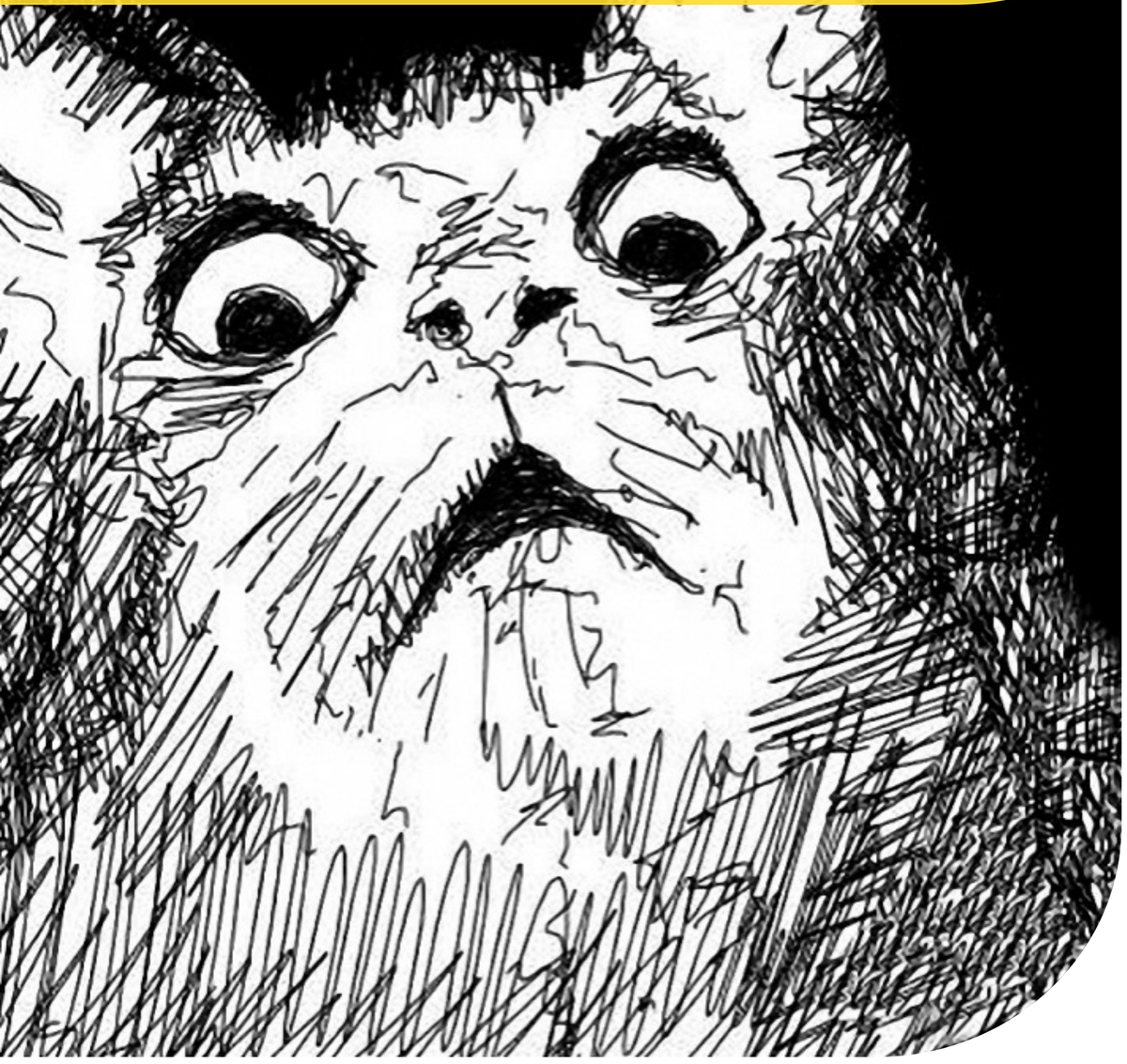
- Утилизация >50%
- Что-то случилось...
- Реплика упала
- Вторая не вынесла нагрузки
- Масштабируем!

Внезапно...



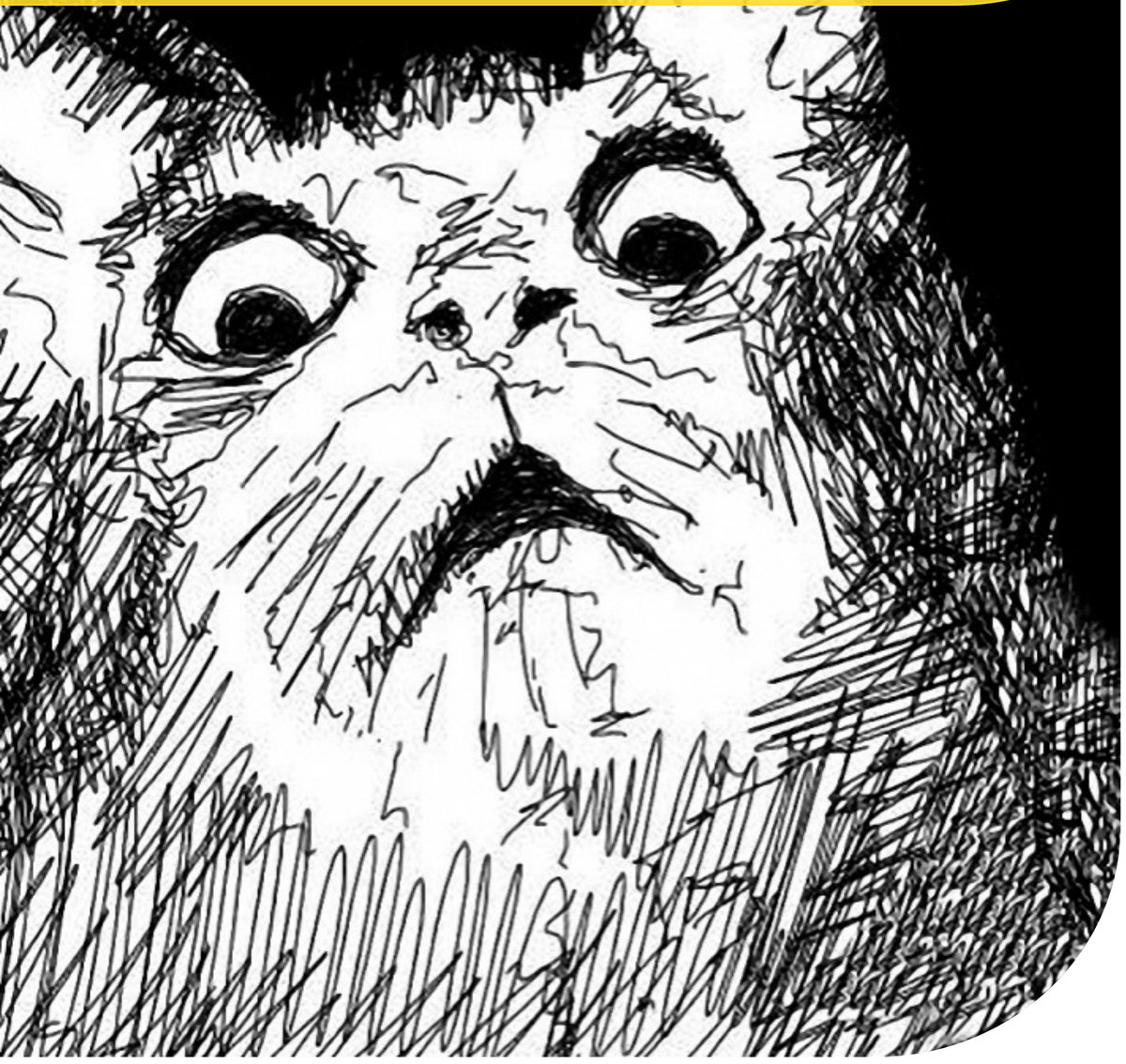
- Нет квот

Внезапно...



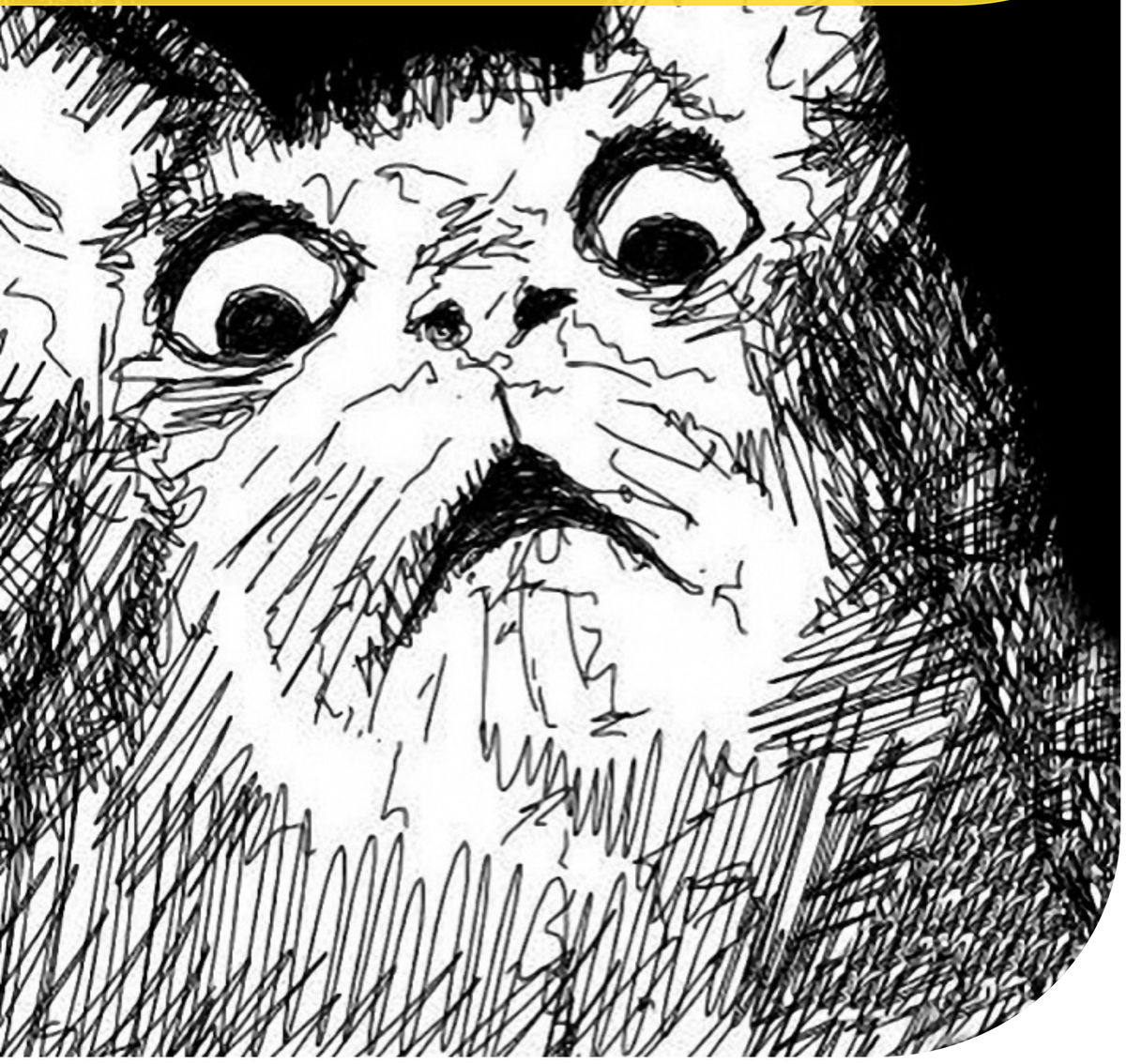
- Нет квот
- Capacity planning f_ckup

Внезапно...



- Нет квот
- Capacity planning f_ckup
- Кончились скрытые ресурсы

Внезапно...



- Нет квот
- Capacity planning f_ckup
- Кончились скрытые ресурсы
- Deadlock

Итоги

Хотели как лучше...

Заложили масштабирование

В надежде подстраховать сервис и быстро восстановиться

Не смогли подняться

Оказалось невозможно быстро и предсказуемо добавить ресурсов

Что делать?

Awareness

- Полезно следить за утилизацией и не терять failover-роли масштабирования

Мониторинг

- Добавляем метрики, алерты и отслеживаем capacity скрытых ресурсов

Load-тесты

- Важно проверять насколько сервис готов к масштабированию

Подведем итоги

Подведем итоги

Что угодно может сломаться

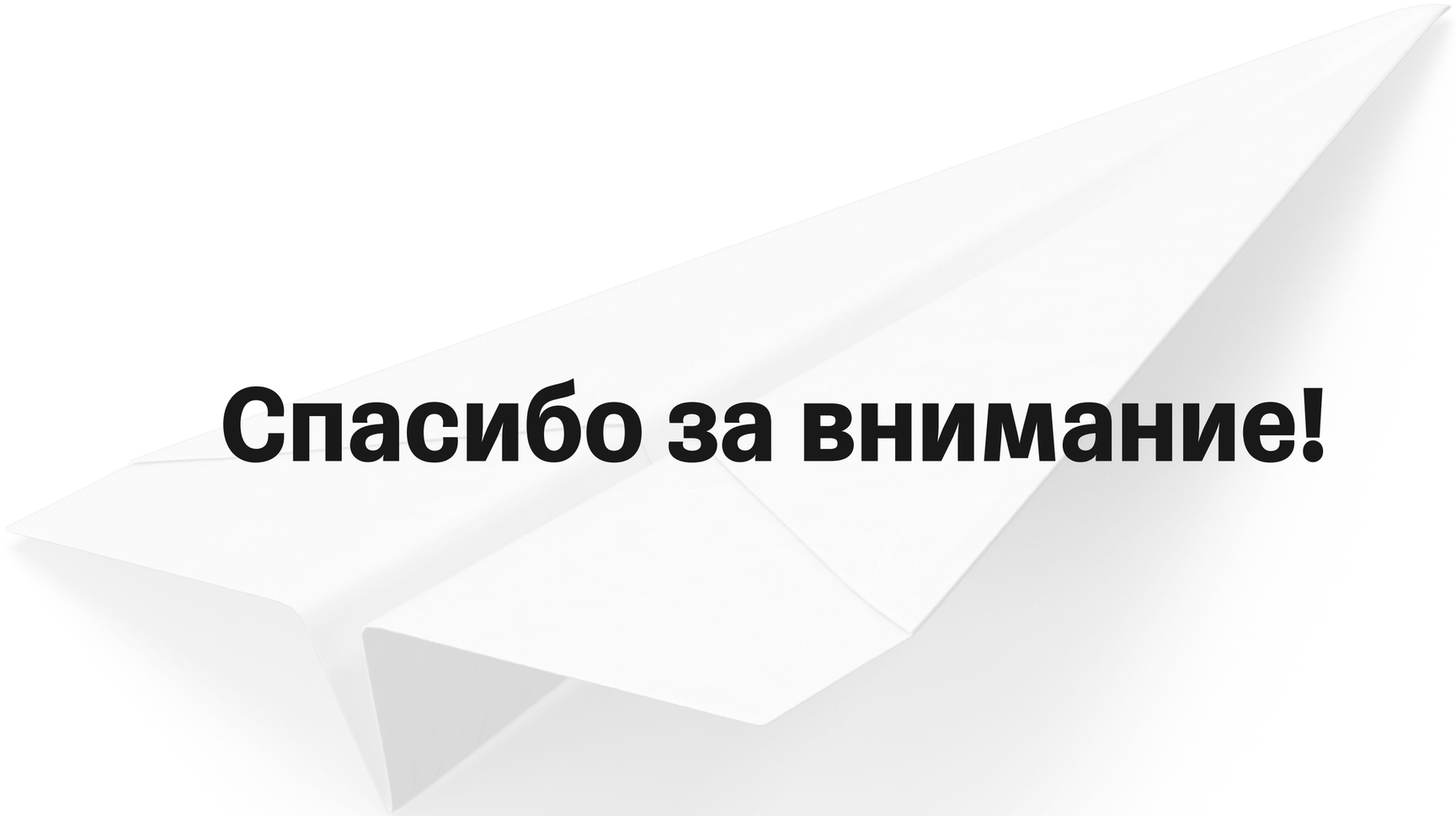
Сервис не выдержит нагрузки, сторонняя система перестанет отвечать, сервер сломается, сетевая связность пропадет, база данных окажется перегружена

Не доверяем реализации

Самих паттернов надежности. Кривая реализация во время сбоя может оказаться даже хуже, чем ее отсутствие

Проверяем как это работает

На практике. Нагрузочное тестирование, оценка возможностей масштабирования, потенциальный аффект на внешние системы и само приложение



Спасибо за внимание!