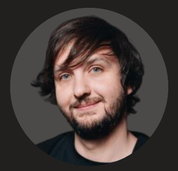
Яндекс Go

Боремся с метастабильными состояниями отказа



Вадим Мартынов

Руководитель команды
платформы надёжности в Яндекс Go

Хочу рассказать вам историю

QR-коды можно не сканировать, все ссылки будут в конце и в чате

Почему это бывает важно

👋 Привет, я Вадим

01 14 лет программирую за деньги

02 В Яндекс Go — для души

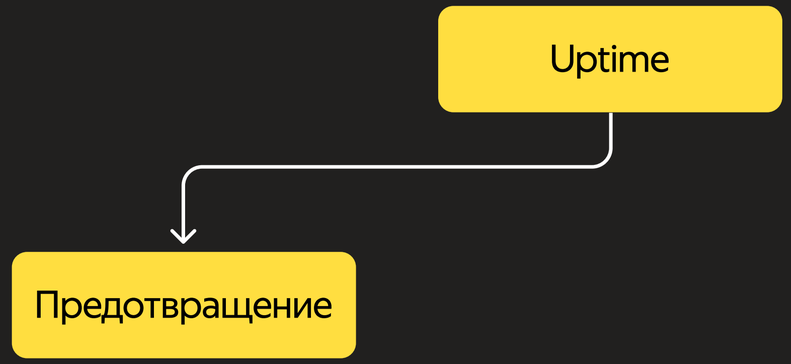
03 SRE?

04 Занимаюсь платформой надёжности

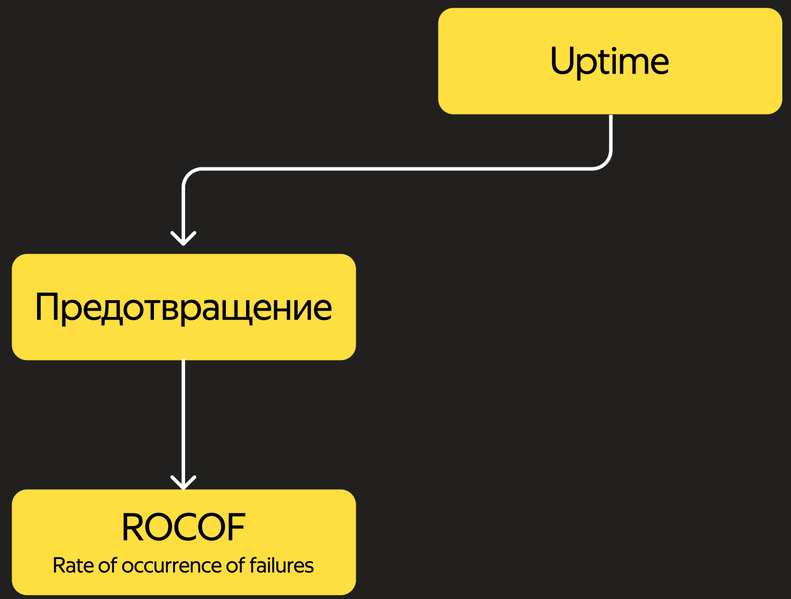
Занимаюсь платформой надёжности

Uptime

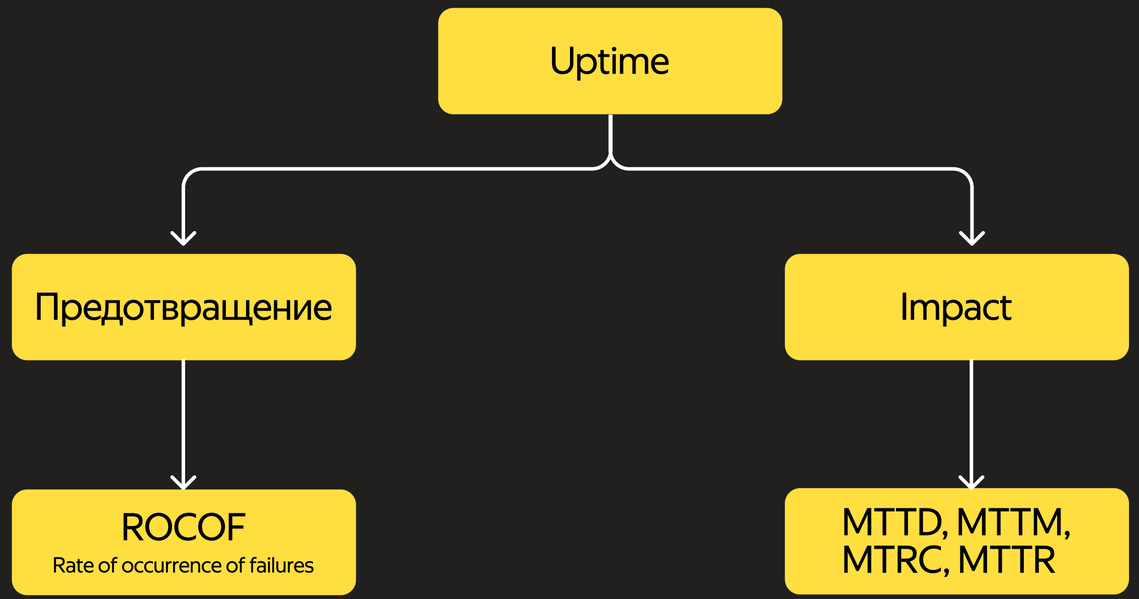
Занимаюсь платформой надёжности



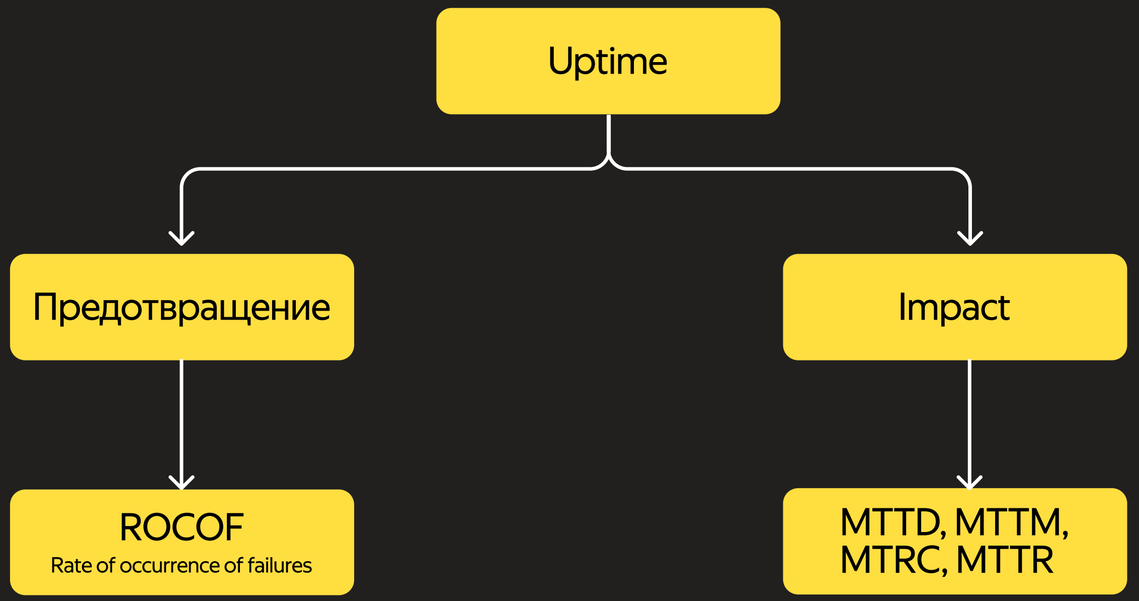
Занимаюсь платформой надёжности



Занимаюсь платформой надёжности

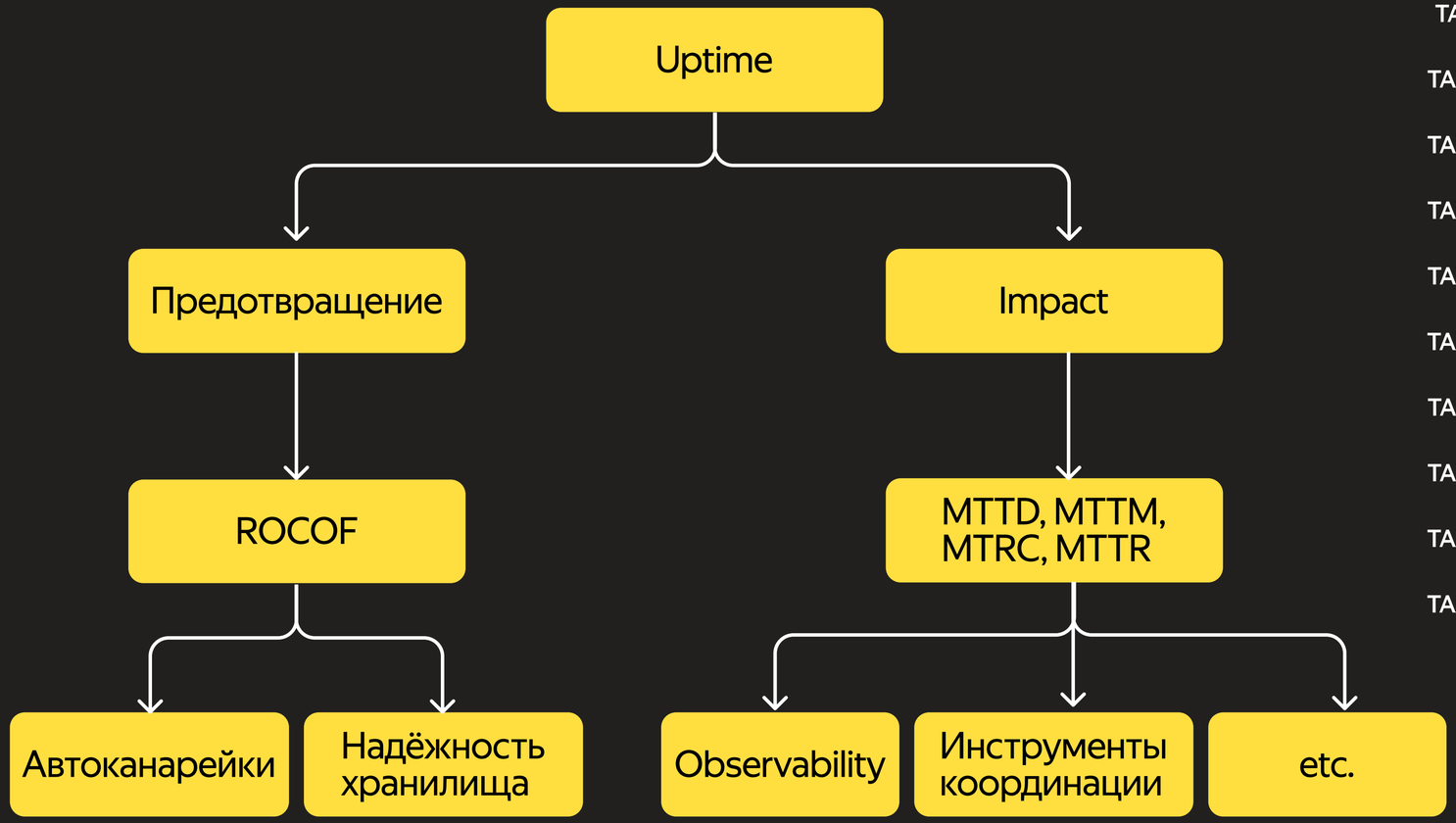


Занимаюсь платформой надёжности



	Root cause	MTTD	MTTRC	MTTR
TAXIINCINDETS-1	Релиз	1	3	7
TAXIINCINDETS-2	Конфиг	6	11	14
TAXIINCINDETS-3	Релиз	13	8	10
TAXIINCINDETS-4	СУБД	14	23	25
TAXIINCINDETS-5	Релиз	1	6	12
TAXIINCINDETS-6	Релиз	2	5	11
TAXIINCINDETS-7	СУБД	9	12	20
TAXIINCINDETS-8	Внешний	6	8	12
TAXIINCINDETS-9	Сеть	5	11	14
TAXIINCINDETS-10	Релиз	5	7	11

Занимаюсь платформой надёжности



	Root cause	MTTD	MTTRC	MTTR
TAXIINCINDETS-1	Релиз	1	3	7
TAXIINCINDETS-2	Конфиг	6	11	14
TAXIINCINDETS-3	Релиз	13	8	10
TAXIINCINDETS-4	СУБД	14	23	25
TAXIINCINDETS-5	Релиз	1	6	12
TAXIINCINDETS-6	Релиз	2	5	11
TAXIINCINDETS-7	СУБД	9	12	20
TAXIINCINDETS-8	Внешний	6	8	12
TAXIINCINDETS-9	Сеть	5	11	14
TAXIINCINDETS-10	Релиз	5	7	11

Занимаюсь платформой надёжности

Модель надёжности

Примеры

Cloud

- Прогнозируемая доступность зоны или региона не ниже 99.99
- Прогнозируемая доступность стойки не ниже 99.9
- Прогнозируемая доступность основных API эндпойнтов управляющего контура не ниже 99.95

Такси

- Максимальная длительность инцидента (MTTR), затрагивающего всех пользователей и/или водителей не больше N минут
- Количество мажорных инцидентов (ROCOF) не больше X

Почта

- Потеря рекламной выручки в месяц не больше Y рублей
- Доступность самых критичных микросервисов, обслуживающих пользователей, не ниже 99.99 в год

Саша Фишер

00 — 34

Как строится надёжность Яндекс Go

Вадим Мартынов

11 — 103

Занимаюсь платформой надёжности

Сложность метастабильных состояний отказа на примере Такси

Что такое MFS

Metastable failure state, метастабильное состояние отказа

Разбираемся в терминах

Отказ

- состояние системы, в котором она не может выполнять свои функции

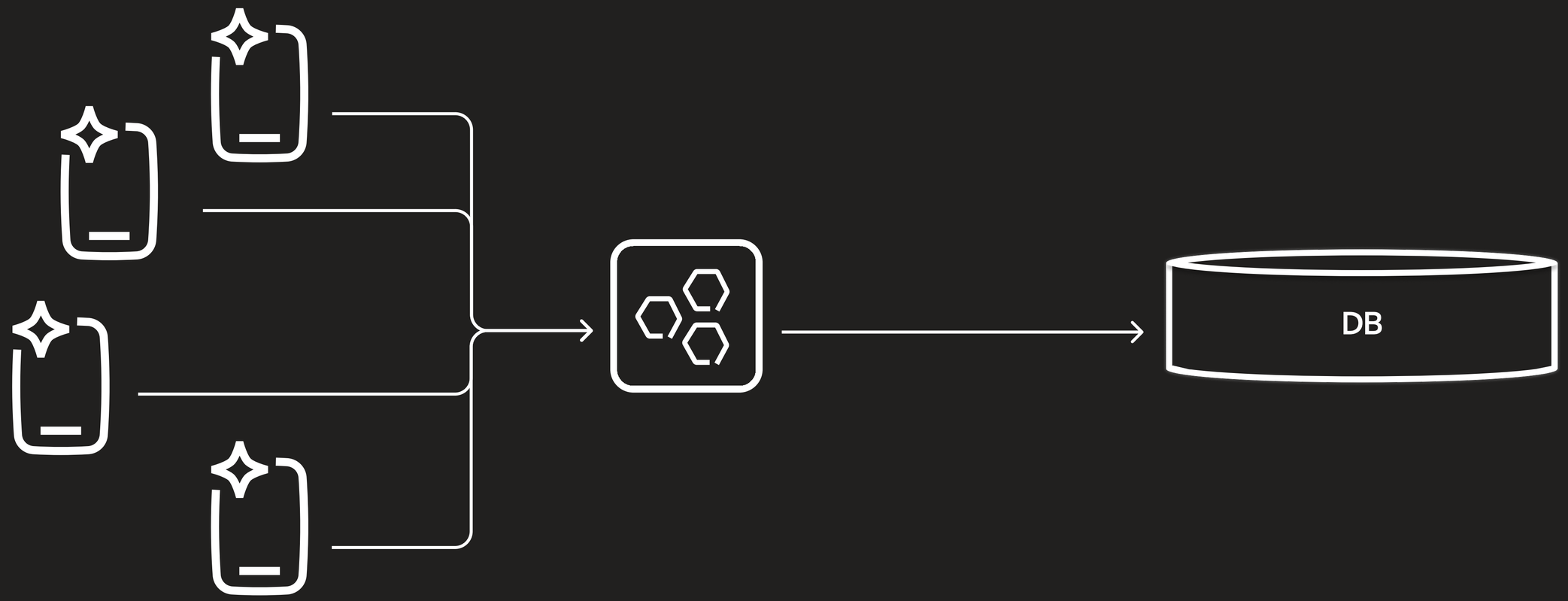
Триггер

- событие, которое привело систему в состояние отказа

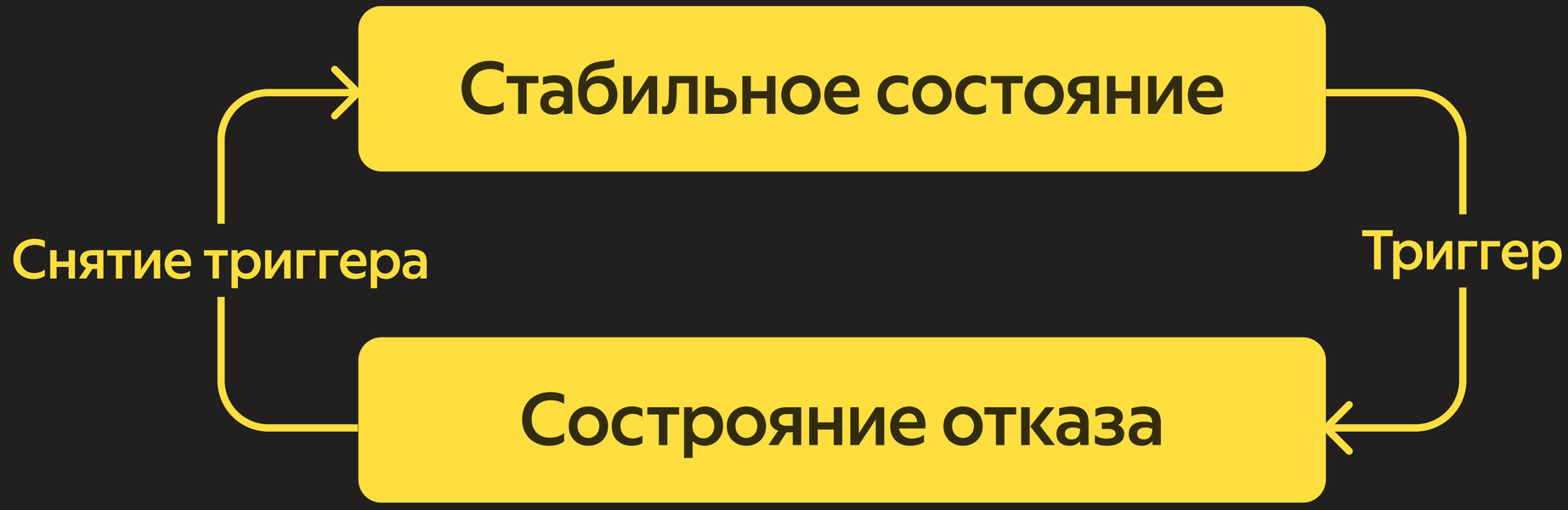
MFS

- состояние системы, когда она не выполняет своей функции после устранения триггера

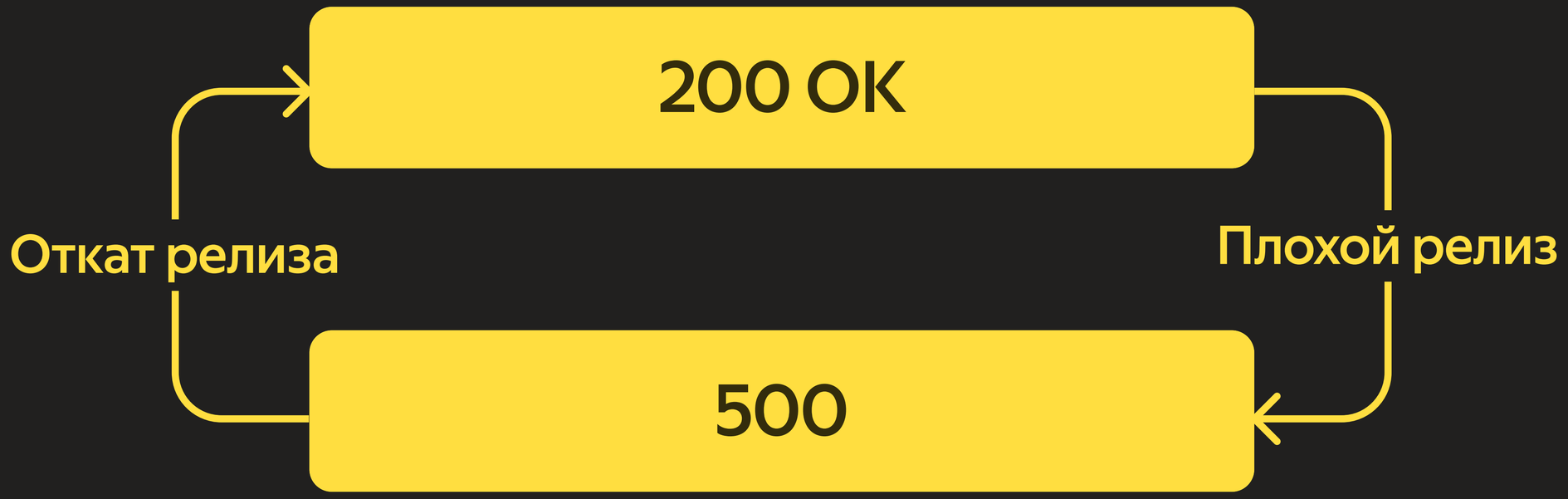
Очень упрощённая схема такси



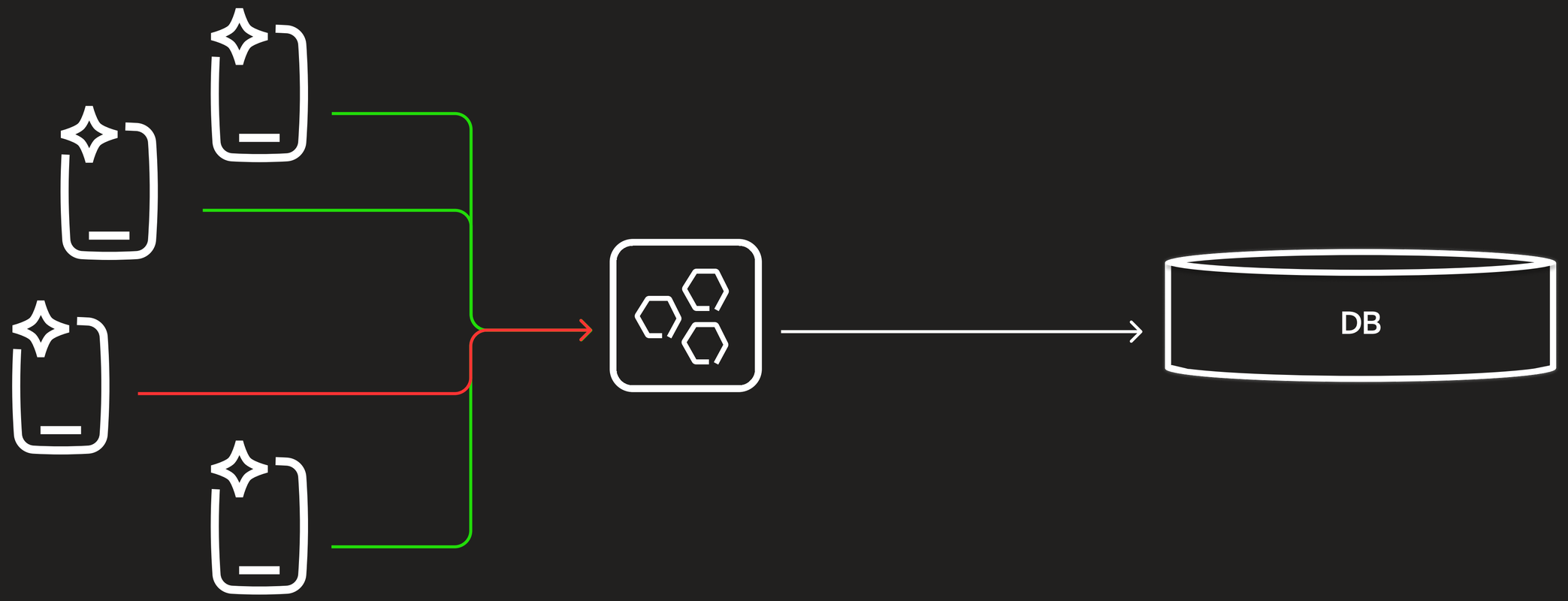
Отказы


Стабильное состояние
Снятие триггера
Триггер
Состояние отказа

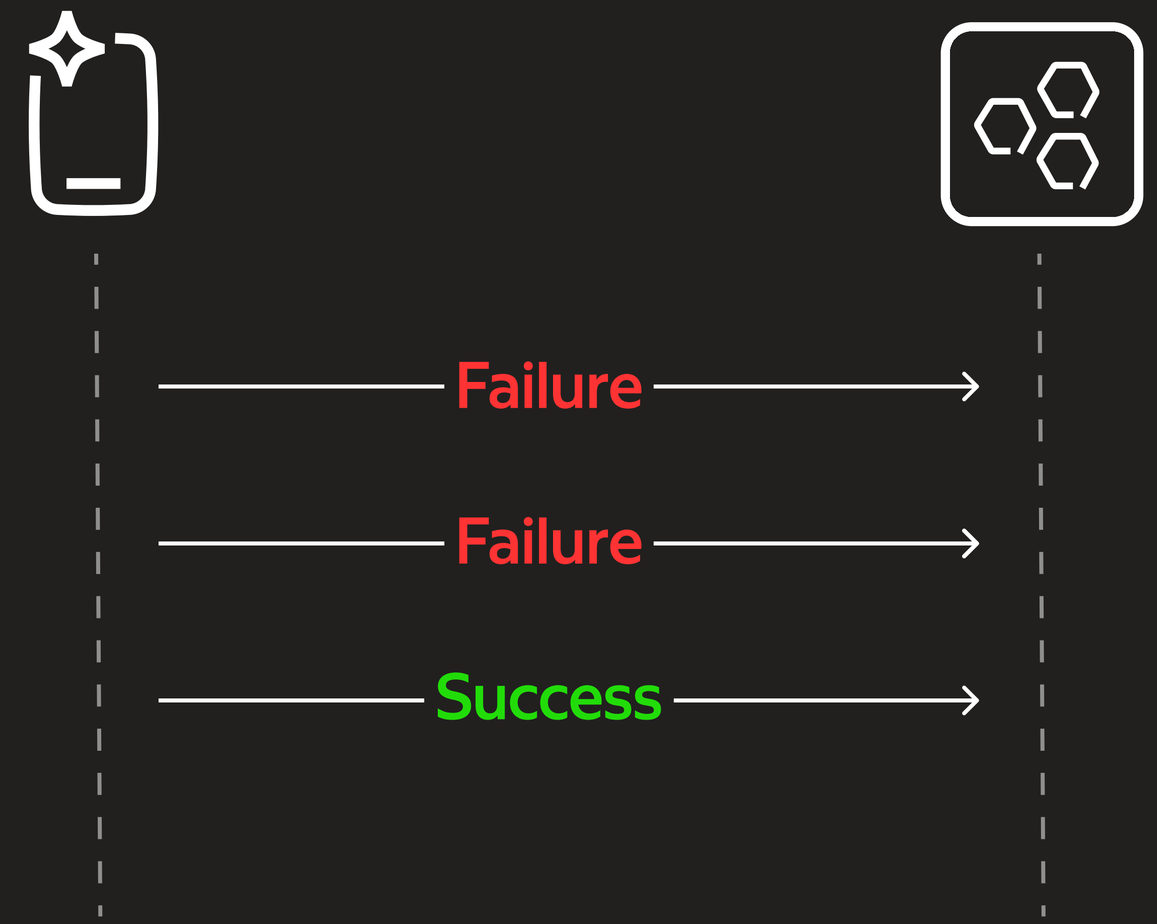
Отказы. Плохой релиз



Таймауты от плохого интернета



Добавим ретраи



Ретраи на графиках сервисов

server requests
server requests (no retries)
client requests ok
client requests timeouts

timings p50, ms
timings p99, ms
timeout, ms

capacity used, %

Ретраи на графиках сервисов

Ретраи на графиках сервисов

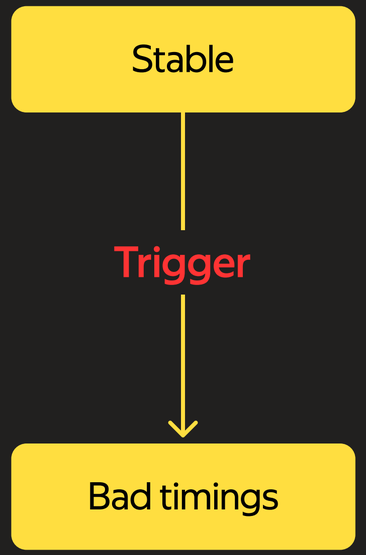
Инцидент длительностью 5 секунд

Инцидент длительностью 5 секунд

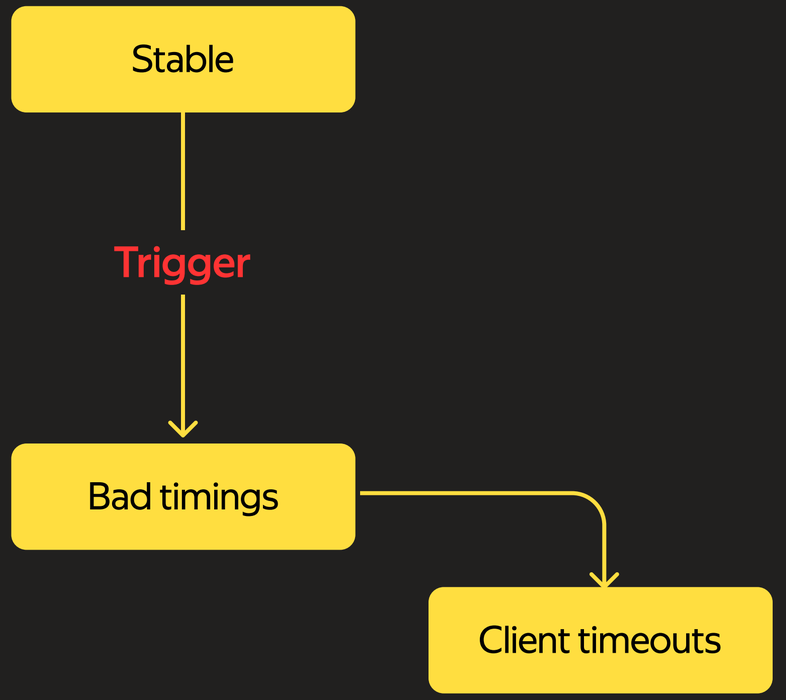
Инцидент длительностью 5 секунд

Stable

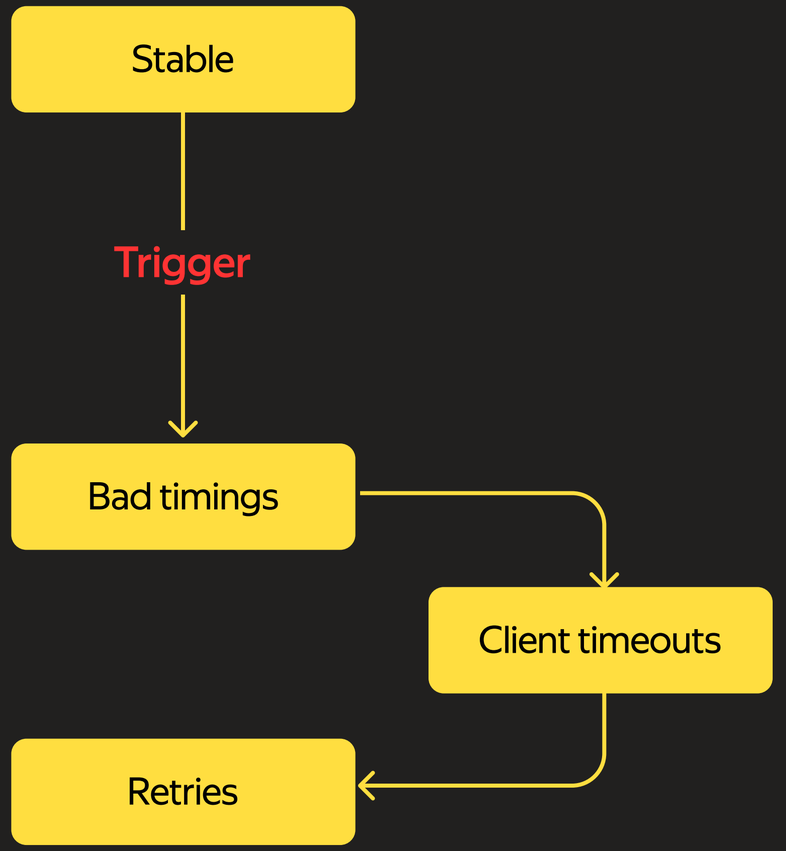
Инцидент длительностью 5 секунд



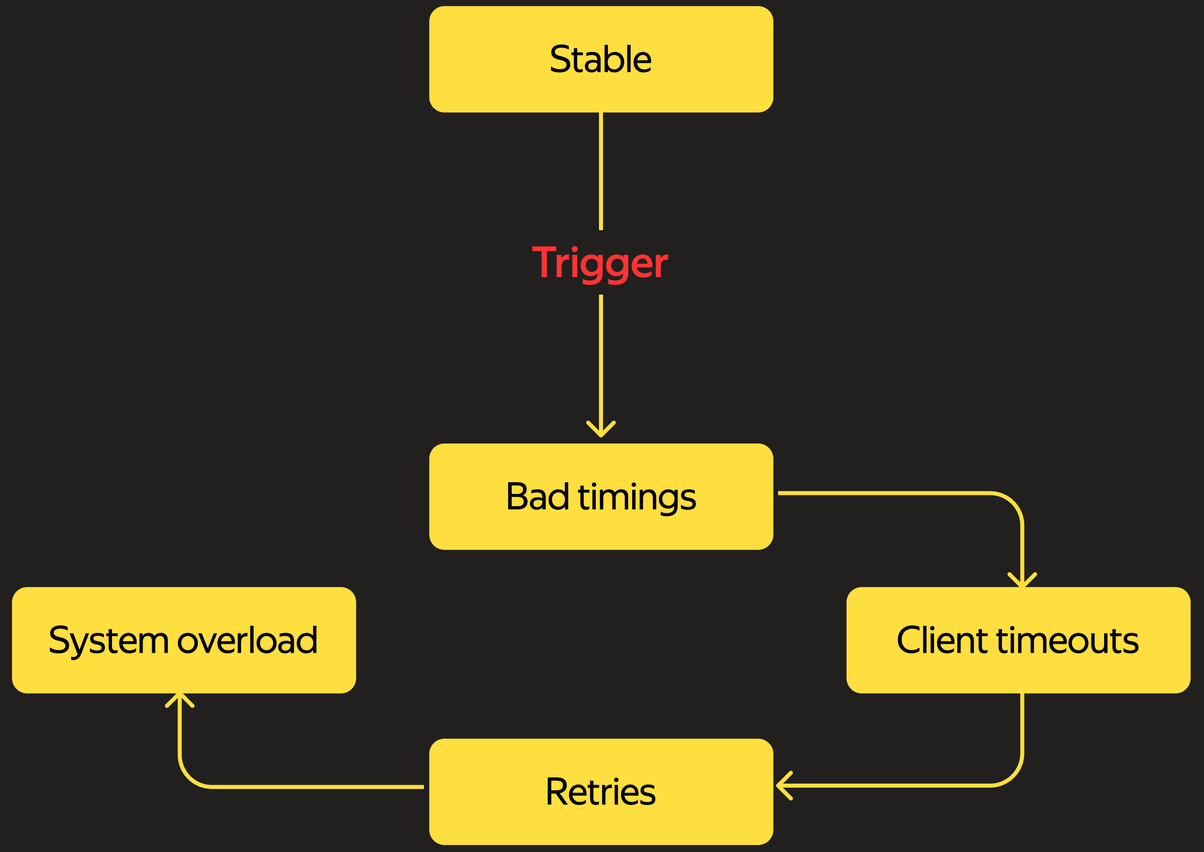
Инцидент длительностью 5 секунд



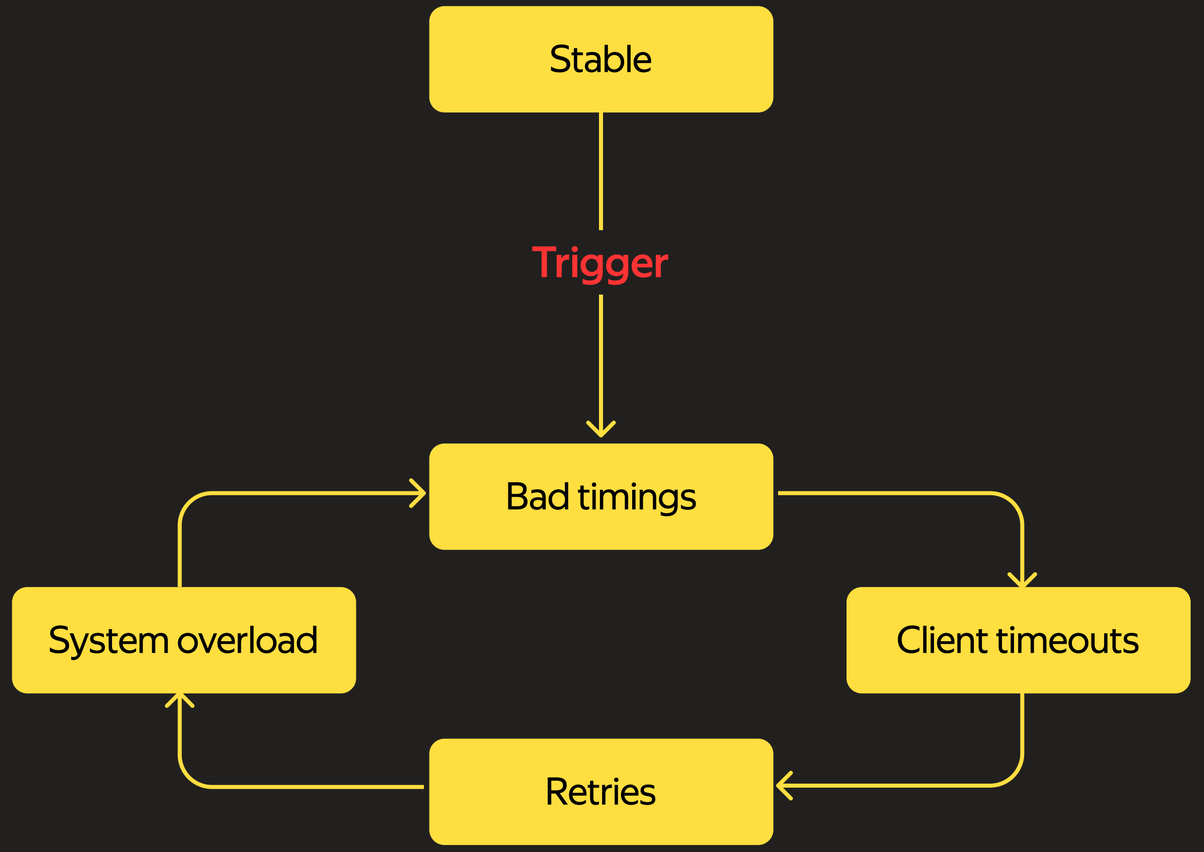
Инцидент длительностью 5 секунд



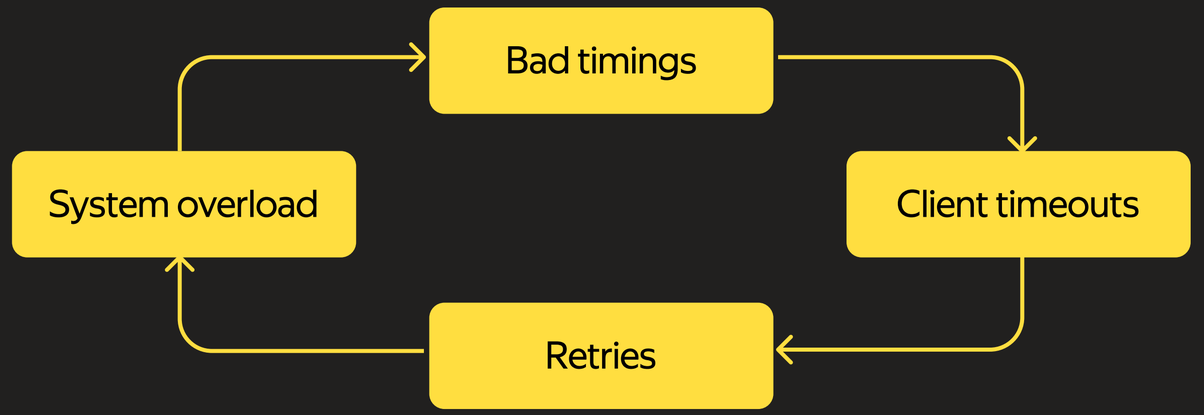
Инцидент длительностью 5 секунд



Инцидент длительностью 5 секунд

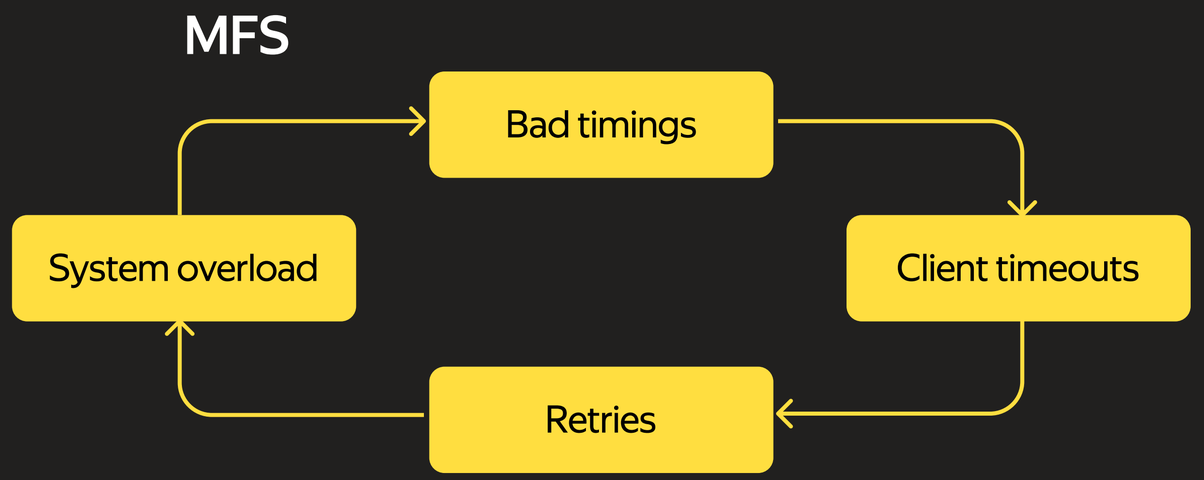


Инцидент длительностью 5 секунд



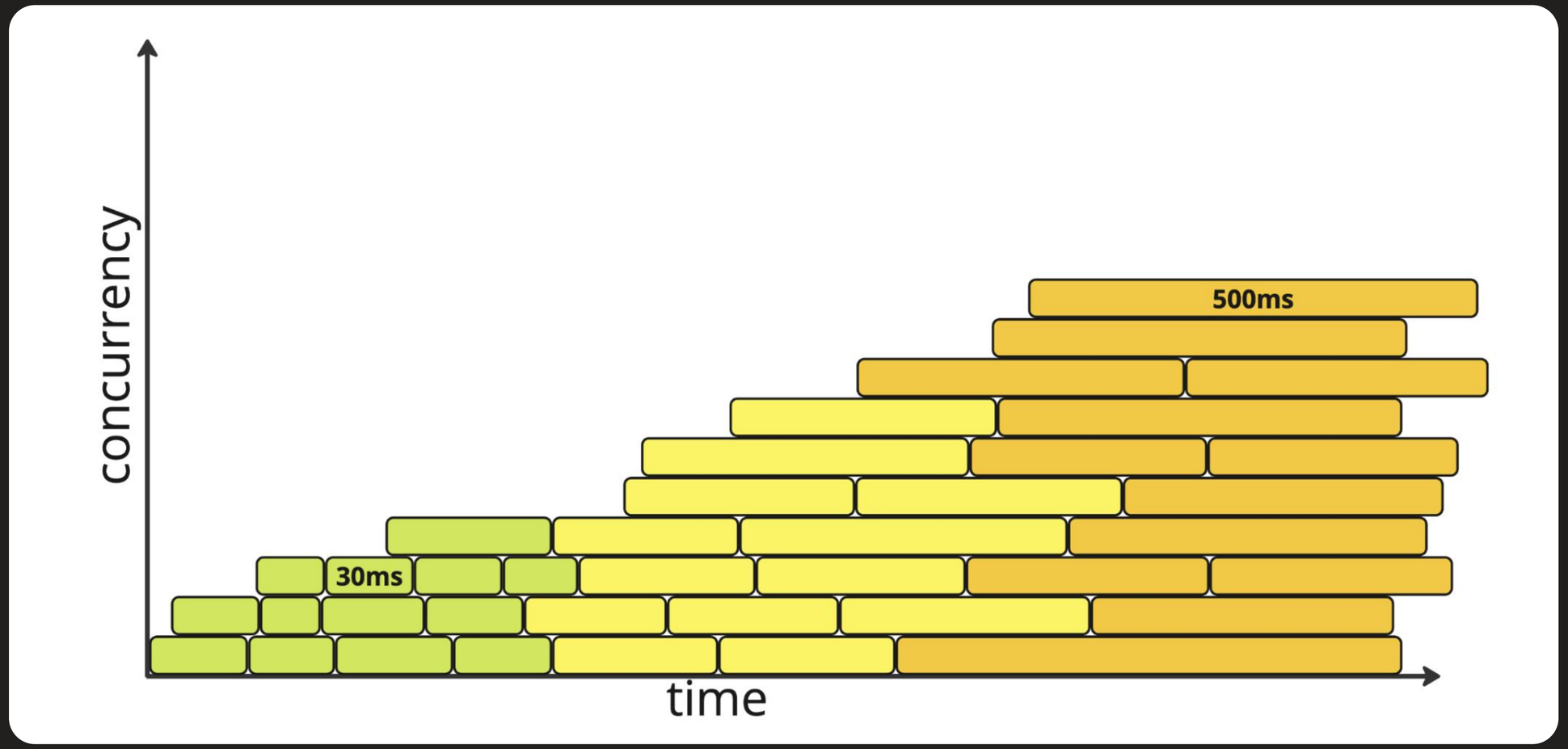
Инцидент длительностью 5 секунд

30 минут



Инцидент длительностью 5 секунд

30 минут

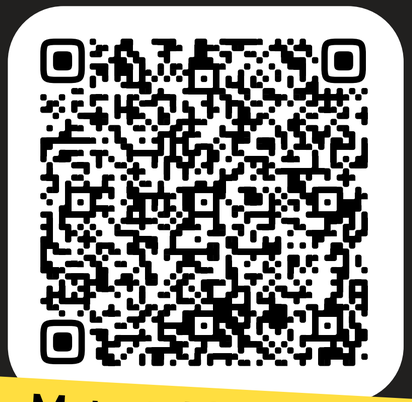


Определим черты MFS

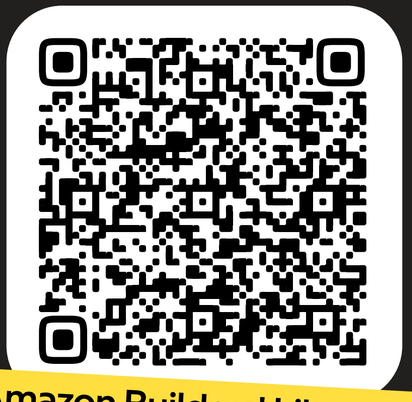
01 Система продолжает находиться в состоянии отказа после снятия триггера

02 В MFS система сохраняет низкий КПД (может быть много работы, которая делается вхолостую)

03 Возникают там, где есть бимодальное поведение системы с нарушением принципа constant work



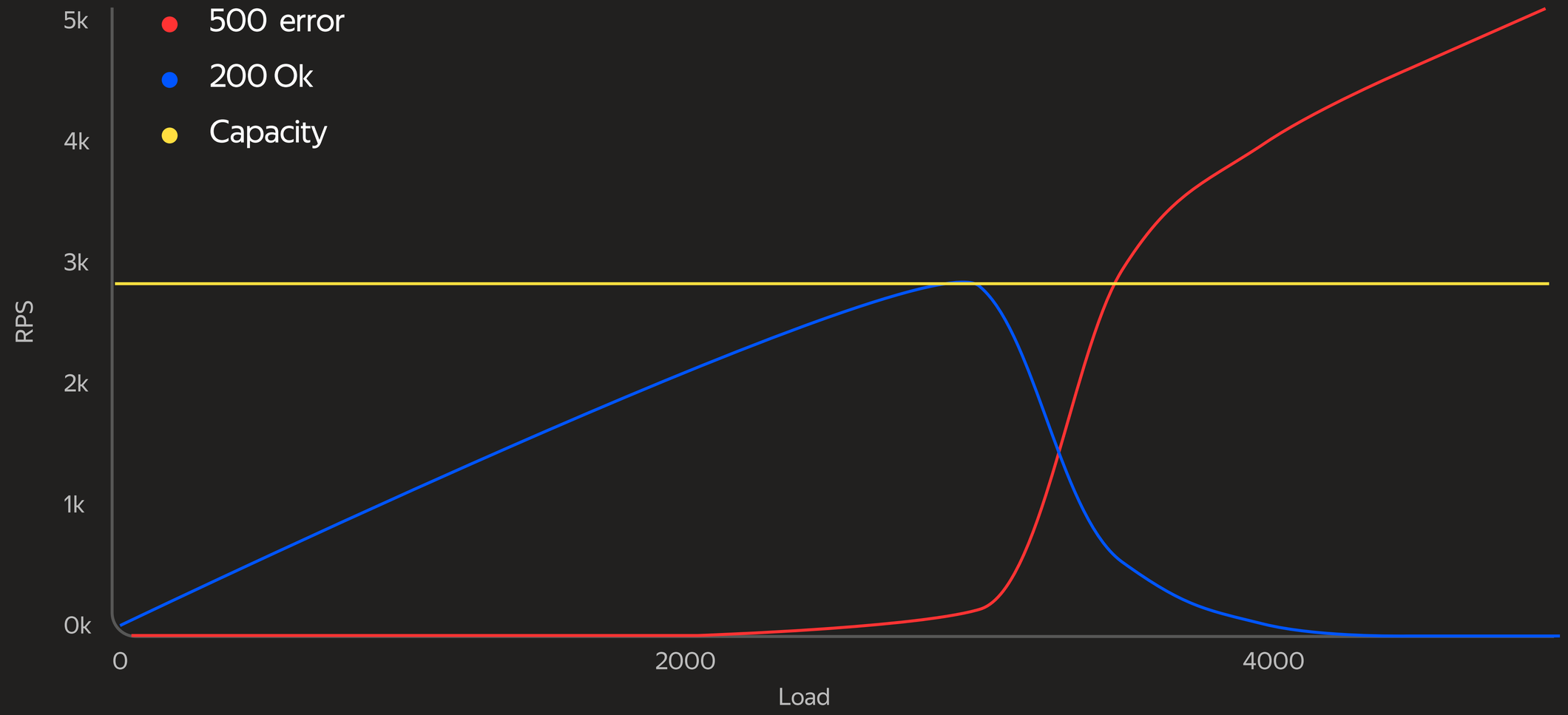
Metastable Failures in Distributed Systems



Amazon Builders' Library “Reliability, constant work, and a good cup of coffee”

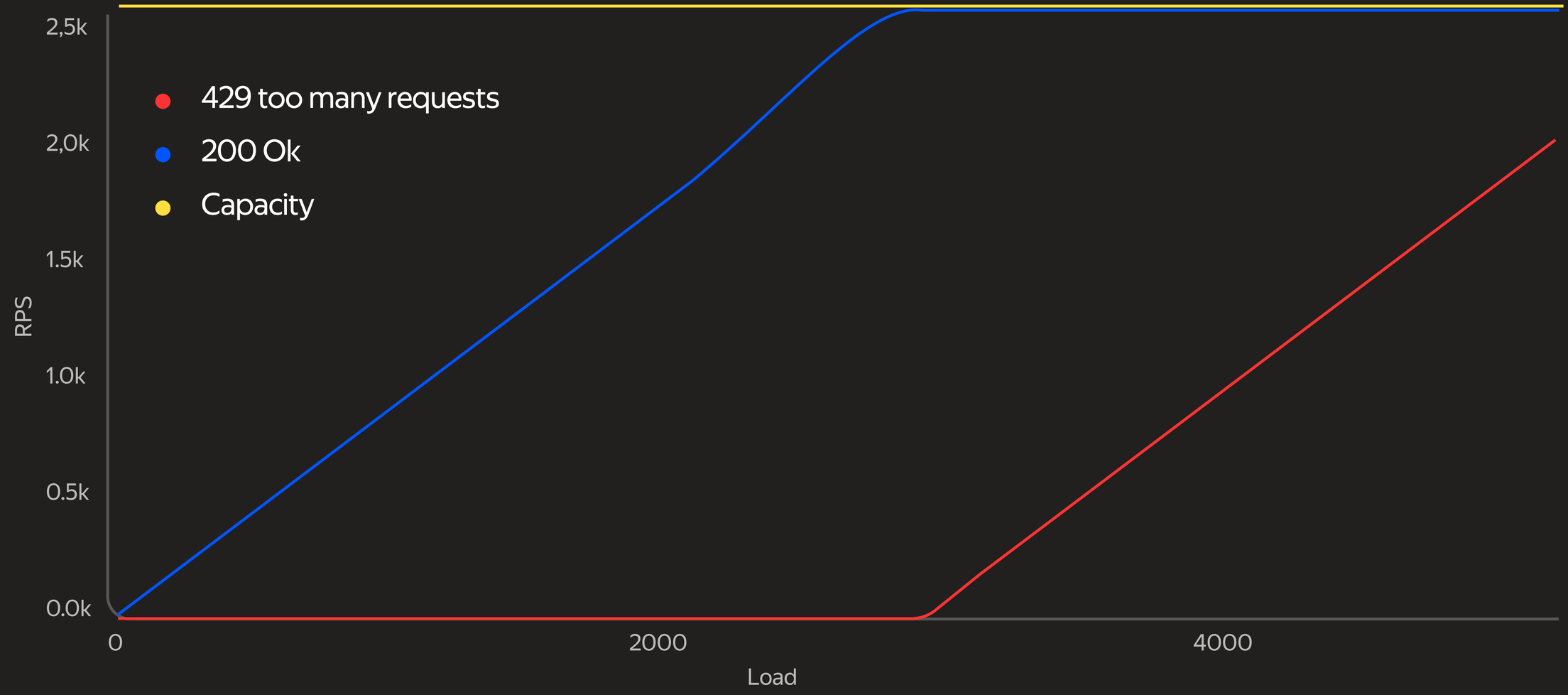
Какие бывают MFS и какие инструменты борьбы с ними

Какие есть механизмы защиты от MFS
от избыточной нагрузки

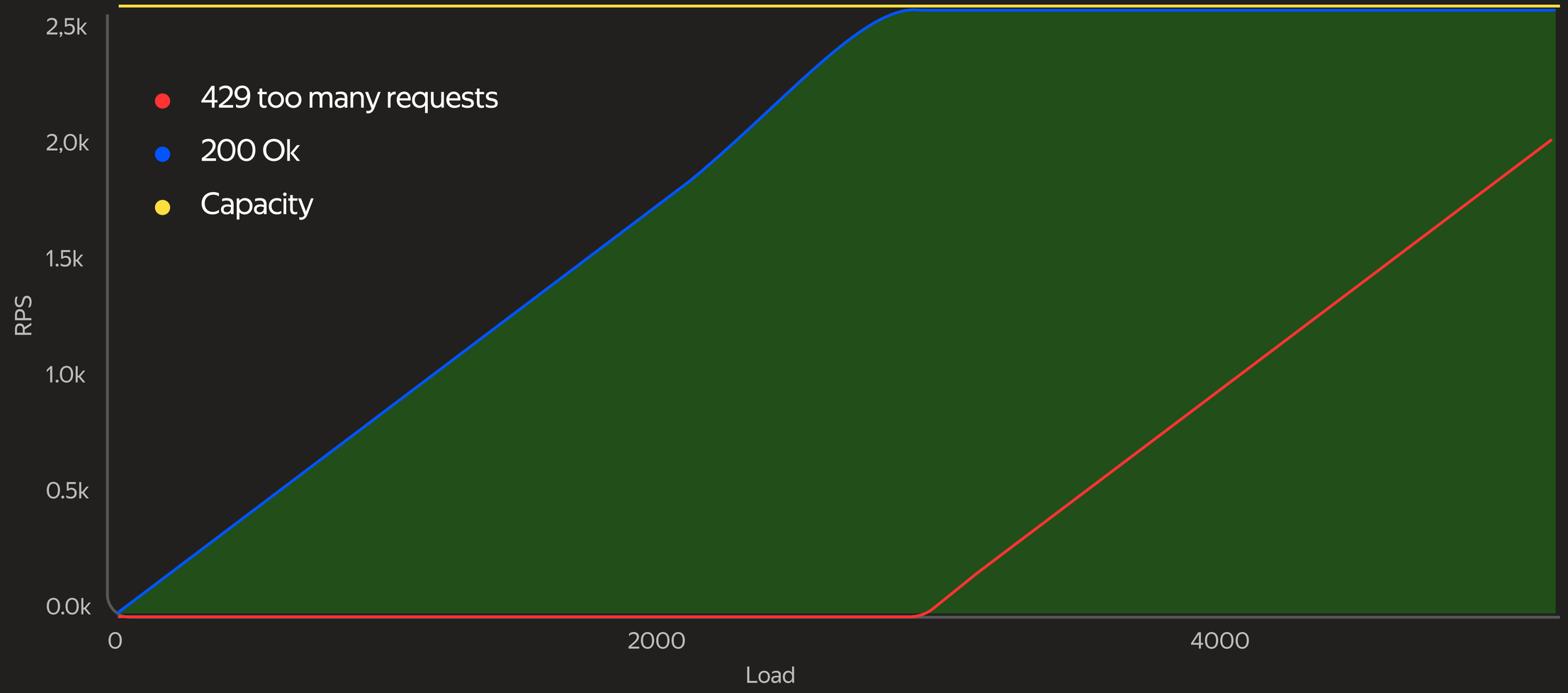


Какие есть механизмы защиты от MFS
от избыточной нагрузки

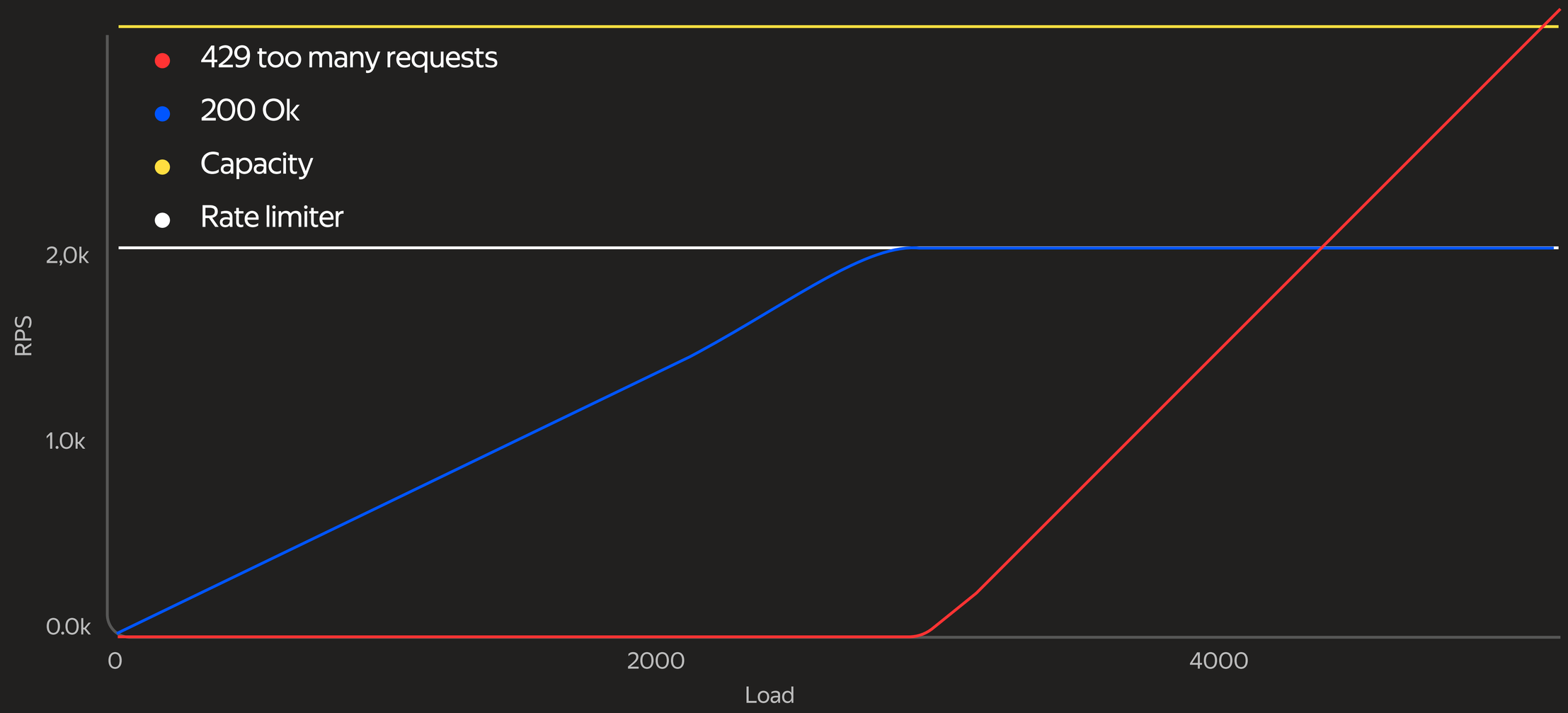
Какие есть механизмы защиты от MFS
от избыточной нагрузки



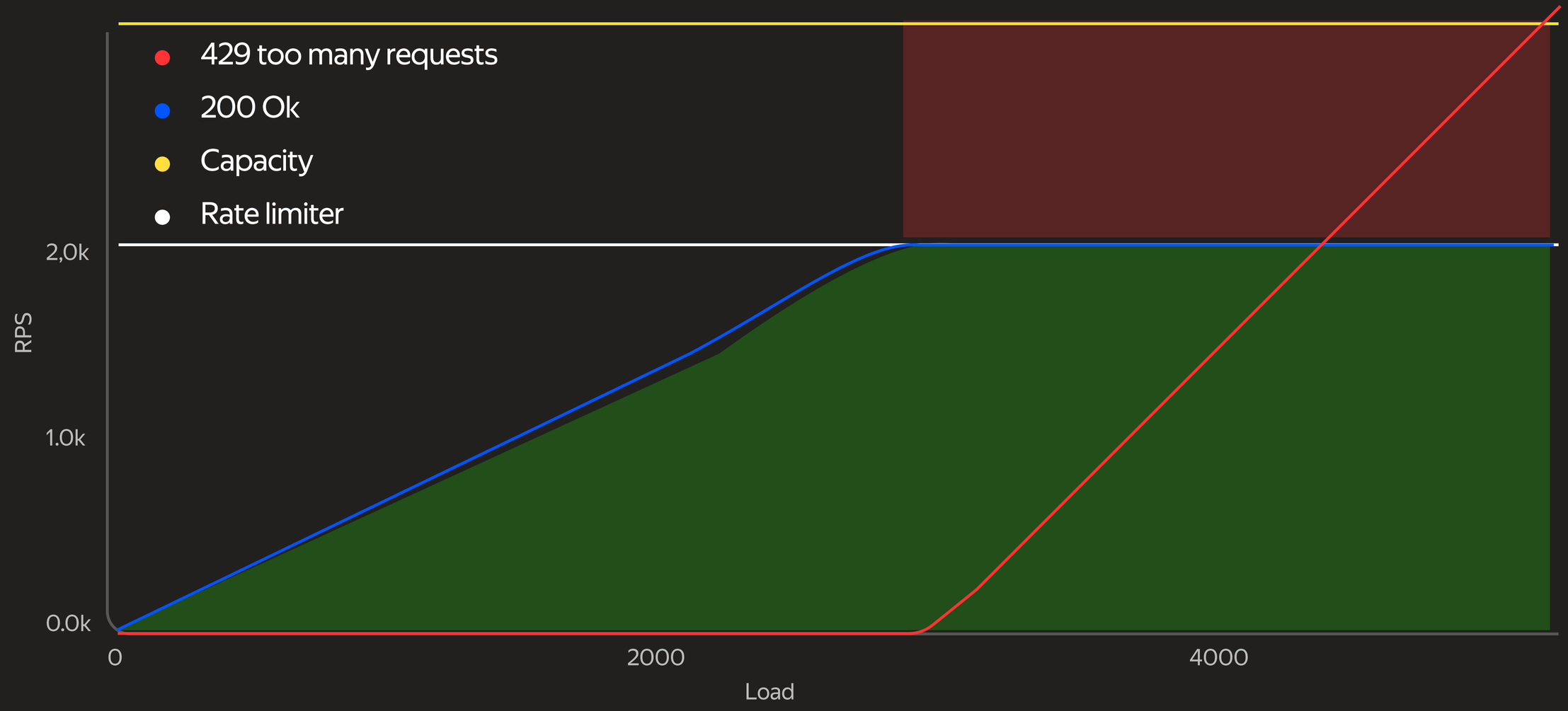
Какие есть механизмы защиты от MFS
от избыточной нагрузки



Rate-limiter



Rate-limiter

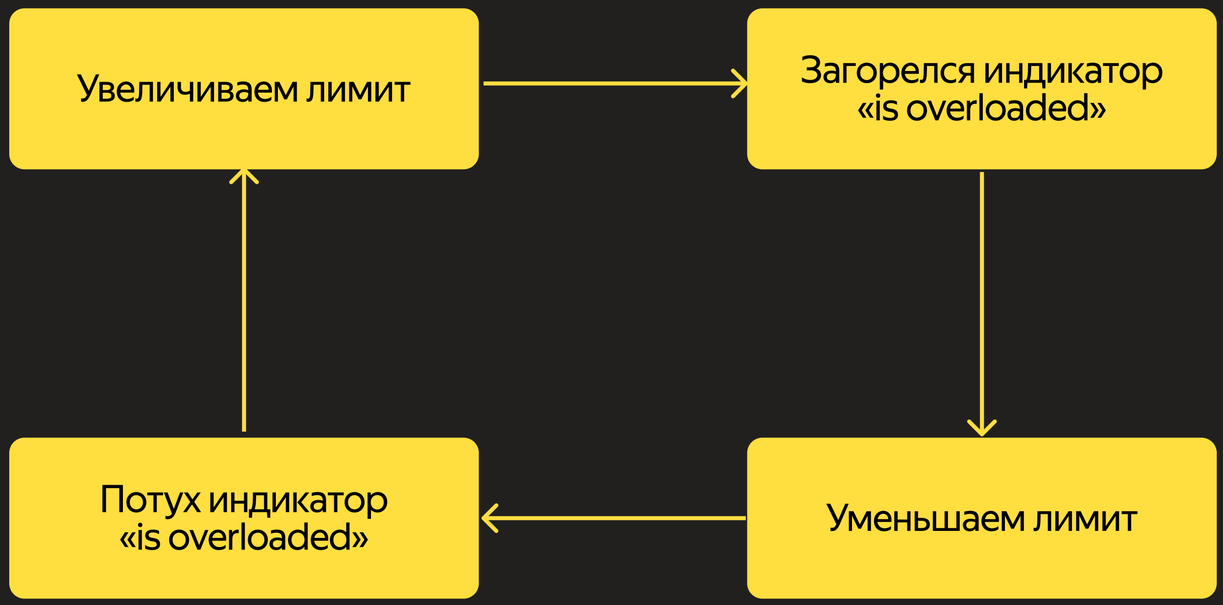


Rate-limiter

- Гарантированный хардстоп
- Можно настраивать для каждой отдельной ручки

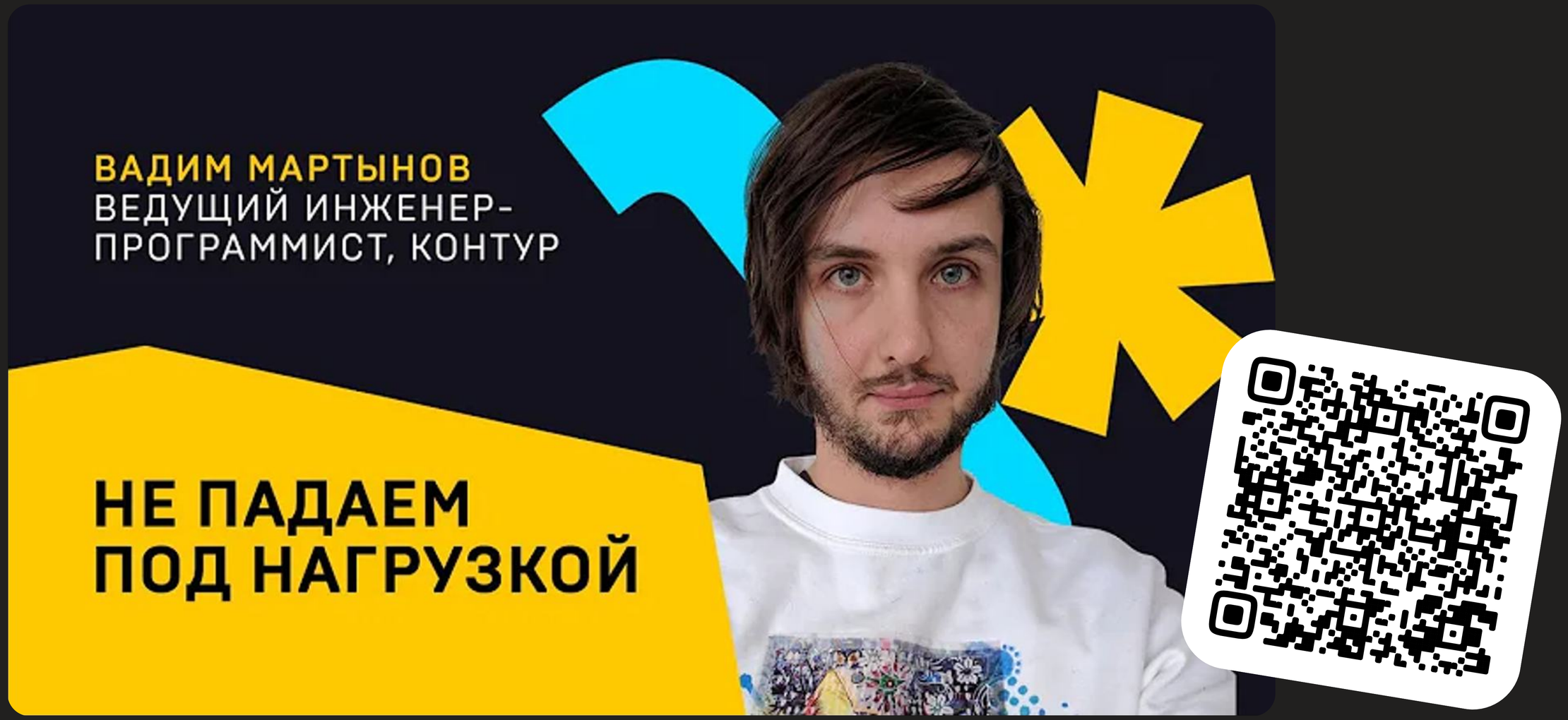
- **Недоутилизация**
- Требует ручной настройки
- Нужна регулярная поддержка и обновление лимитов
- Можно ошибиться

Серверный троттлинг

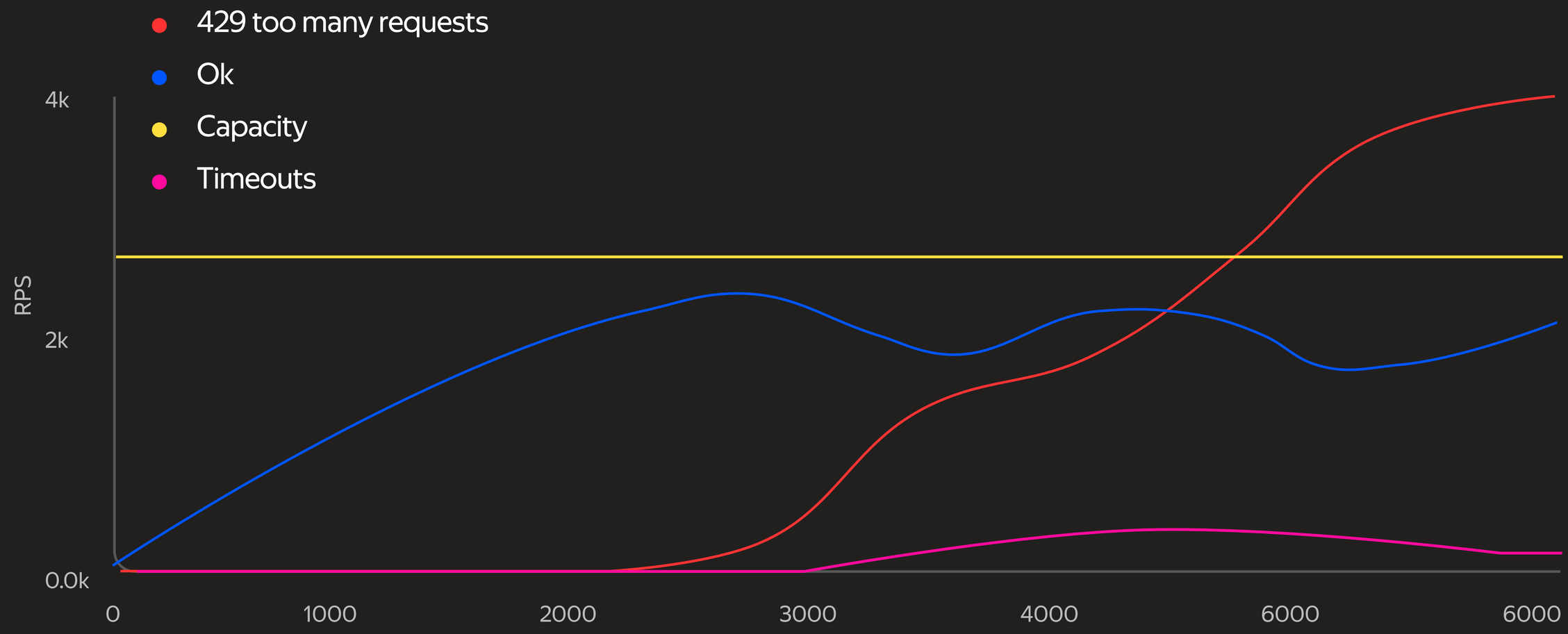


Серверный троттлинг

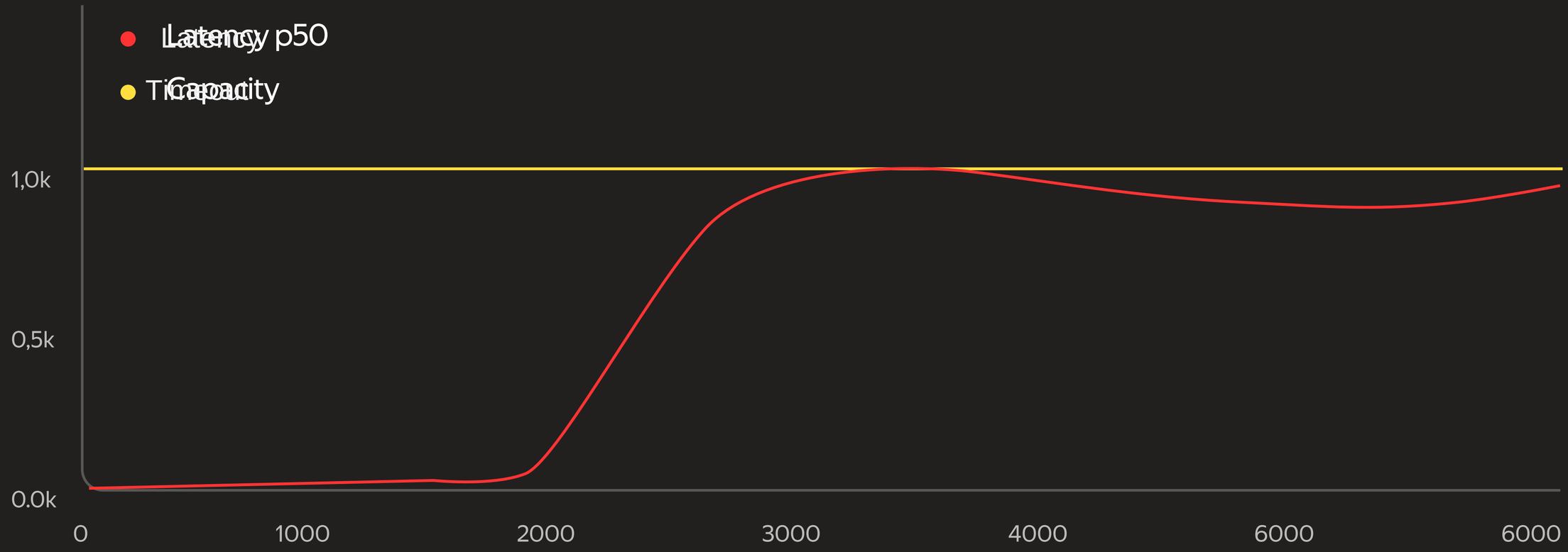
Серверный троттлинг



Серверный троттлинг



Серверный троттлинг

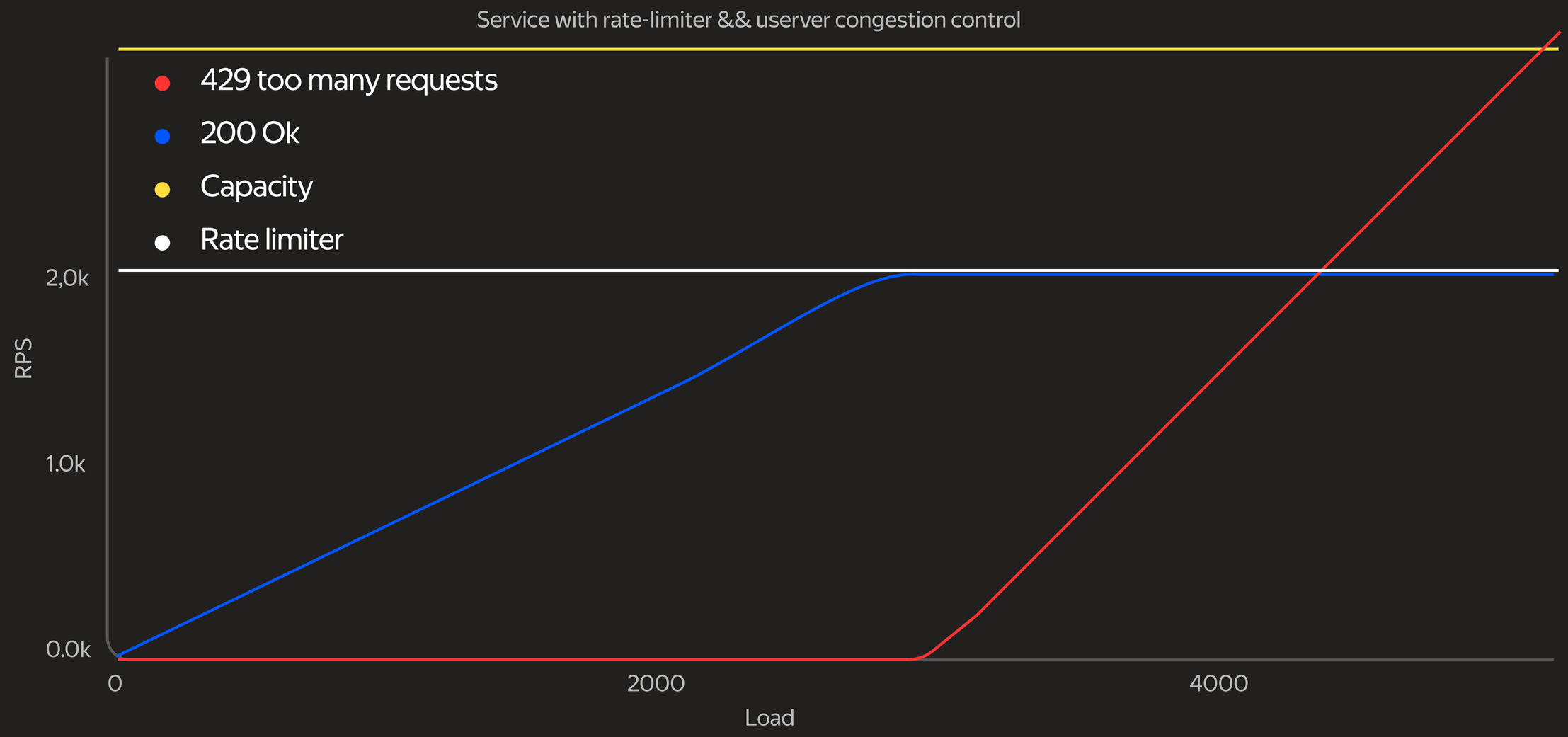


Серверный троттлинг

- Учитывает ёмкость сервиса
- Имеет настройки по умолчанию

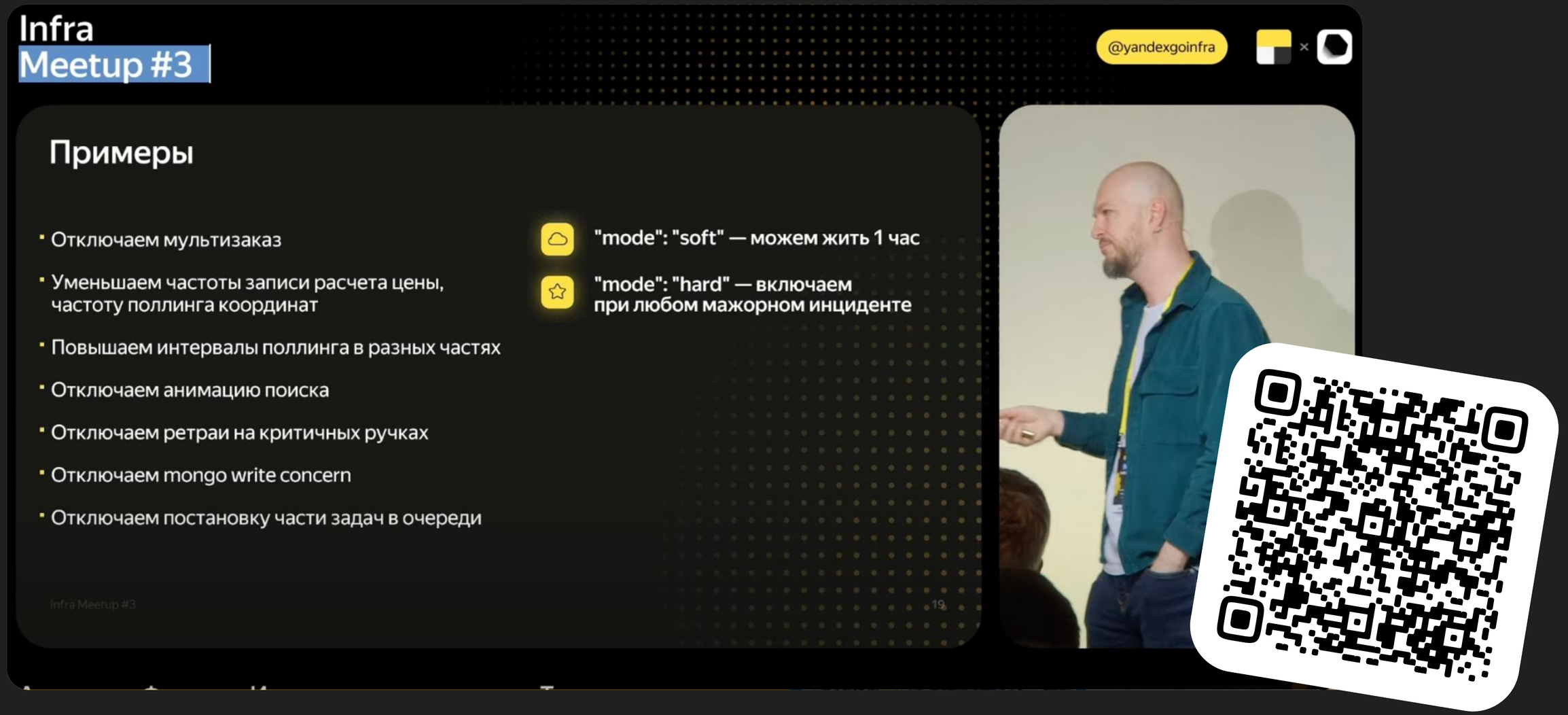
- Рост latency при росте нагрузки
- Нестабильный лимит
- Не защищает зависимости

Совместная работа rate-limiter и серверного троттлинга



Троттлинг на клиентах

Продуктовые деградации



Срезание ретраев

Infra
Meetup #3

@yandexgoinfra

Retries, RPS Limits

Протокол управления ретраями:

- Retry-Number: 1
- Retry-Last-Http-Status-Code: 429
- Retry-Action: stop
- Retry-Interval-Ms:5000

Протокол управления поллингом:

- App-State: idle=true
  итд.

Конфигурация политики ретраев:

- Условия как по отдельным ручкам, так и глобально

Infra Meetup #3 20

Поддерживаемые стратегии ретраев:

- **Repeat**
  После N-го ретрая будет использоваться последний таймаут из списка
- **Exponential back-off**
  500мс, 1000мс, 2000мс, 4000мс, 8000мс
- **Stop**
  После 3-го ретрая отключает ретраи посылкой заголовка Retry-Action:stop
- **Полное отключение ретраев сразу**

Отключение неприоритетных пользователей или отключение виновников избыточной нагрузки

Работает только для бизнесов с разными классами обслуживания (платный SLA, крупные клиенты)

А что такое пользователь? UserId? OrderId?

Какие есть механизмы защиты от MFS от избыточной нагрузки

01 **Rate limiter**
лимитирует число запросов в сервис

02 **Серверный троттлинг**
congestion control в сервисе обрезает нагрузку на основе индикатора перегрузки в сервисе, CPU

03 **Backpressure**
регулирует скорость работы потребителя, троттлинг в клиентах

04 **Продуктовые деградации**
отключают неважные фичи чтобы ёмкости хватило на критичные запросы

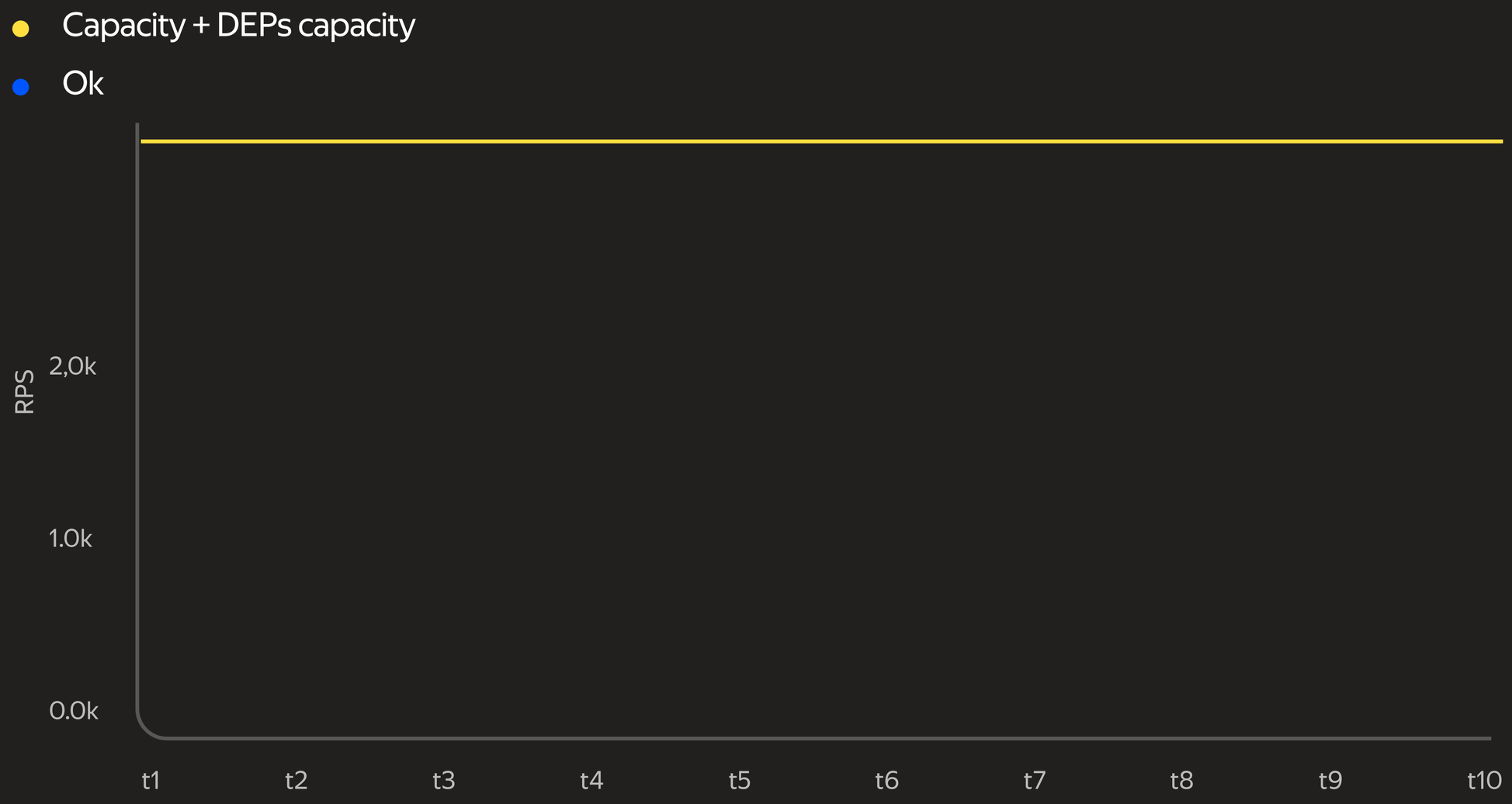
05 **Отключение ретраев**
рубильник, который отключает повторные походы в сервисы

Пробуем решить проблему комплексно

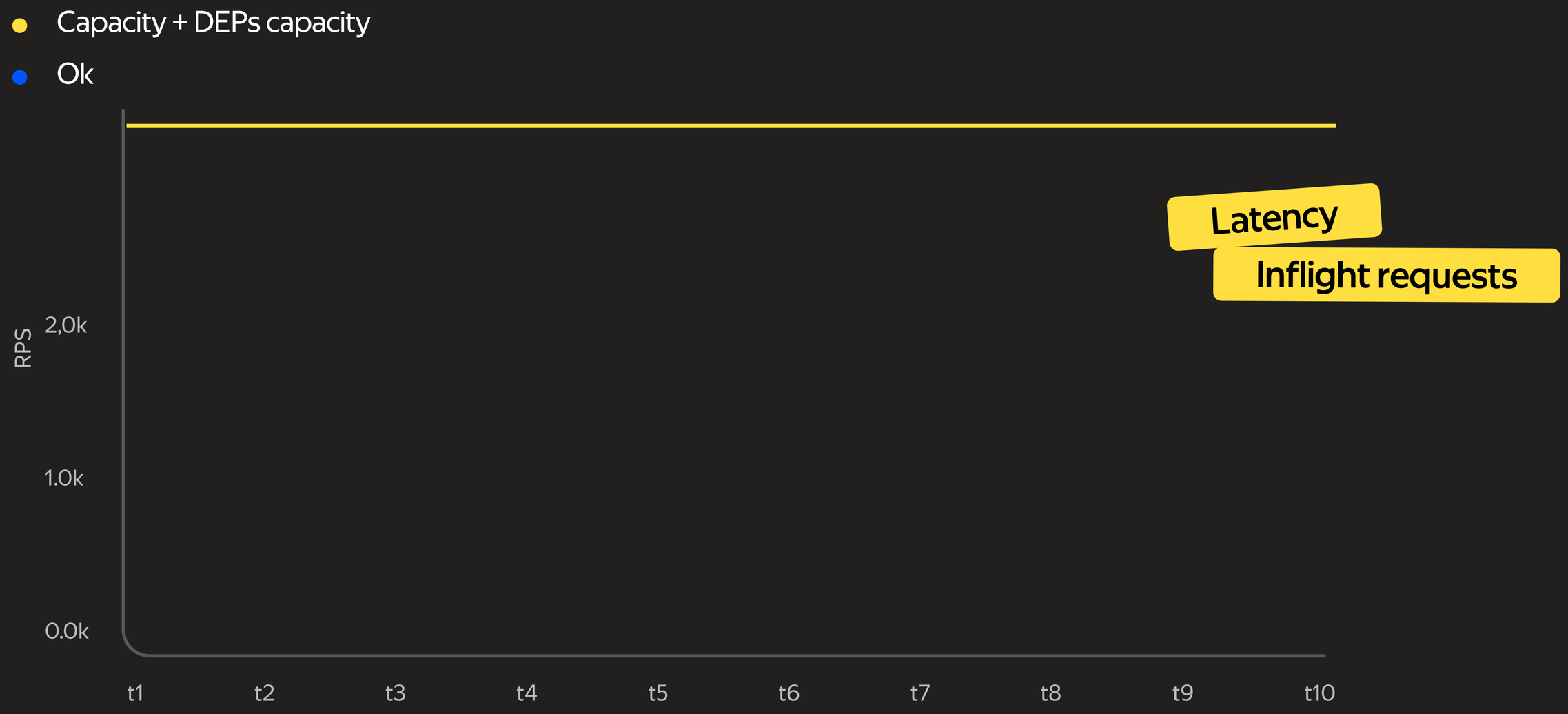
Spoiler alert

Cinnamon: Using Century Old Tech to Build a Mean Load Shedder

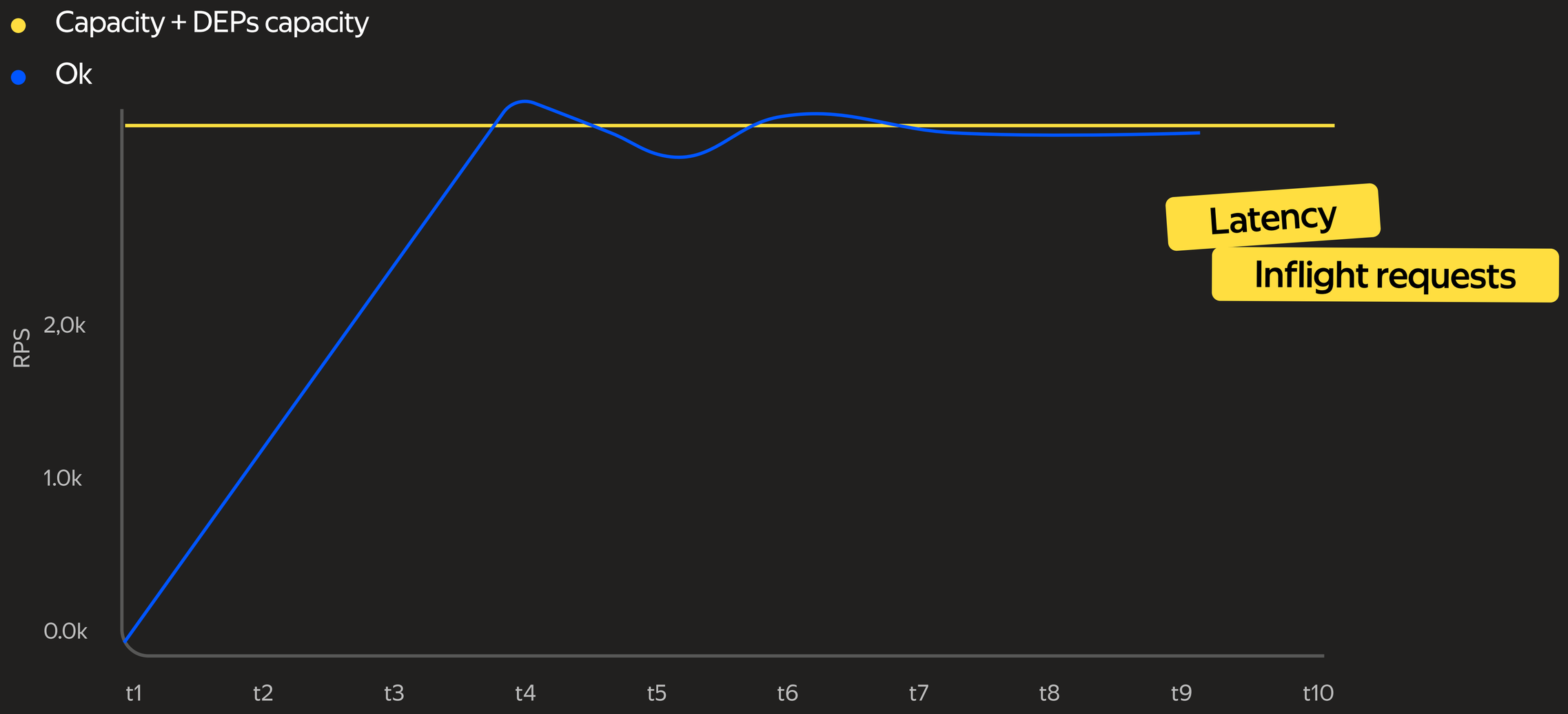
А чего хочется

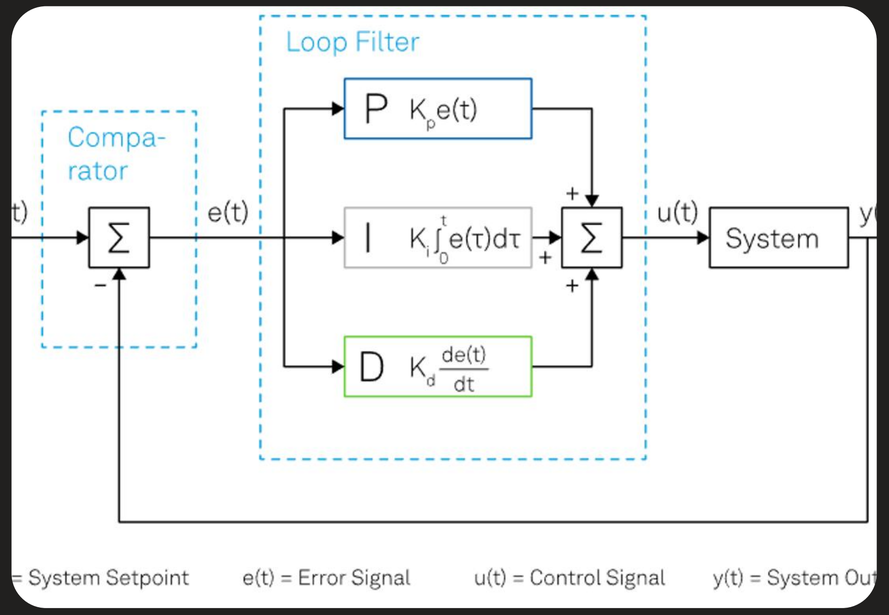


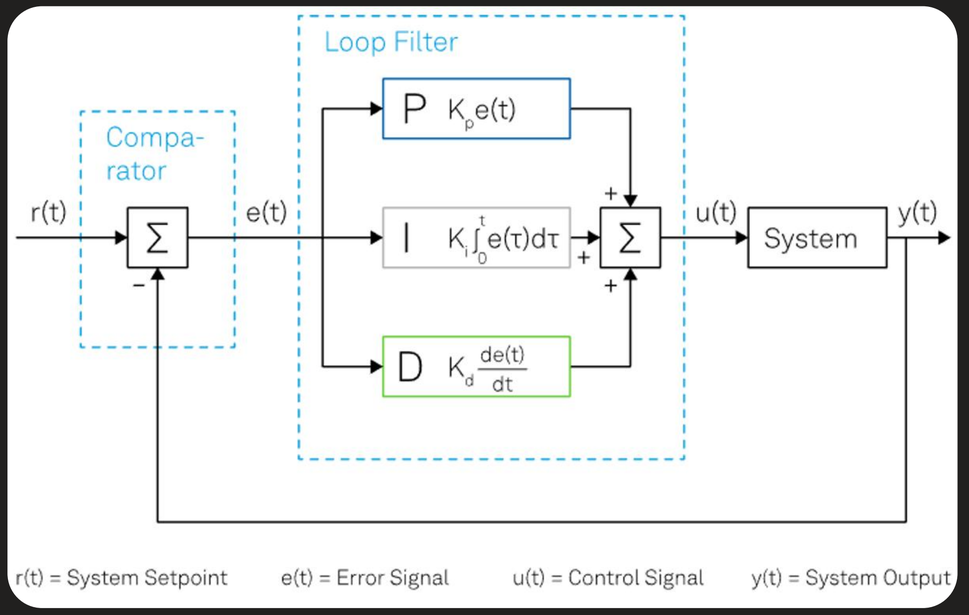
А чего хочется

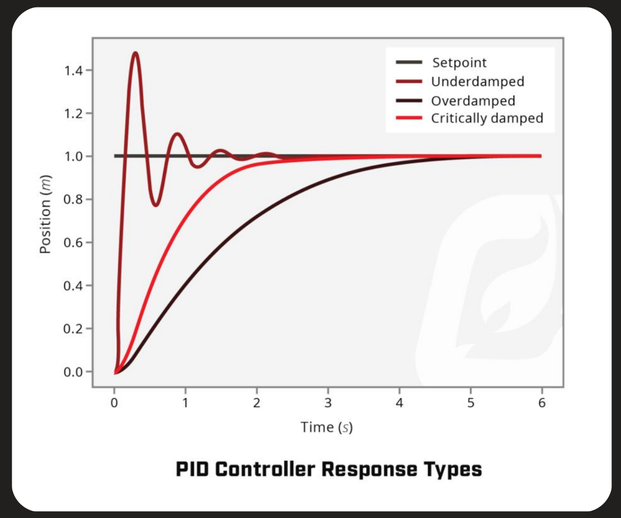


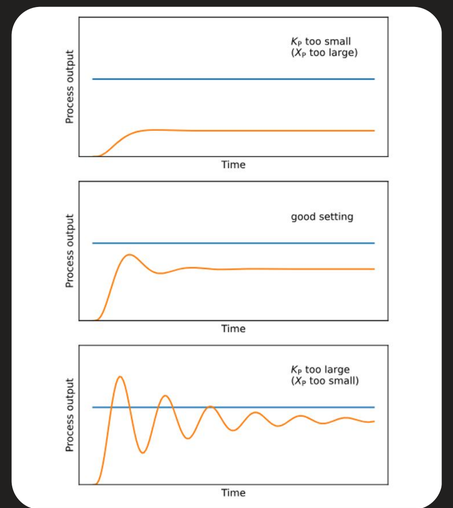
А чего хочется

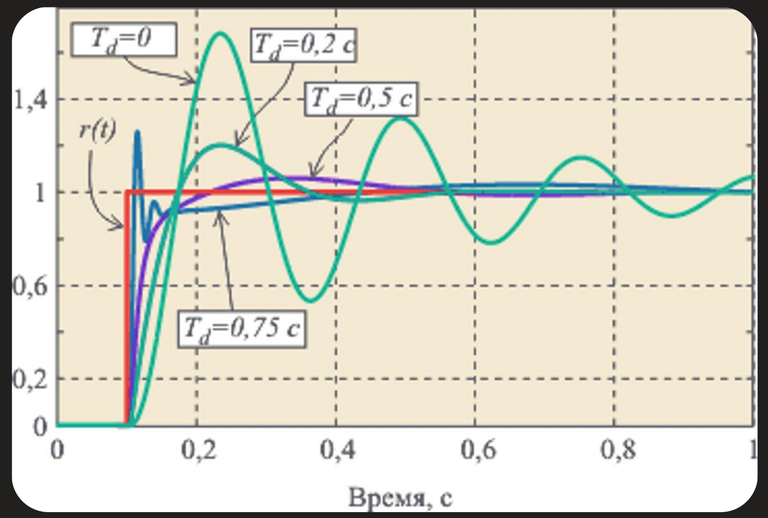



Loop Filter
Compa-
rator
P $K_p e(t)$
I $K_i \int_0^t e(\tau)d\tau$
D $K_d \frac{de(t)}{dt}$
t)
e(t)
u(t)
System
y
= System Setpoint
e(t) = Error Signal
u(t) = Control Signal
y(t) = System Out

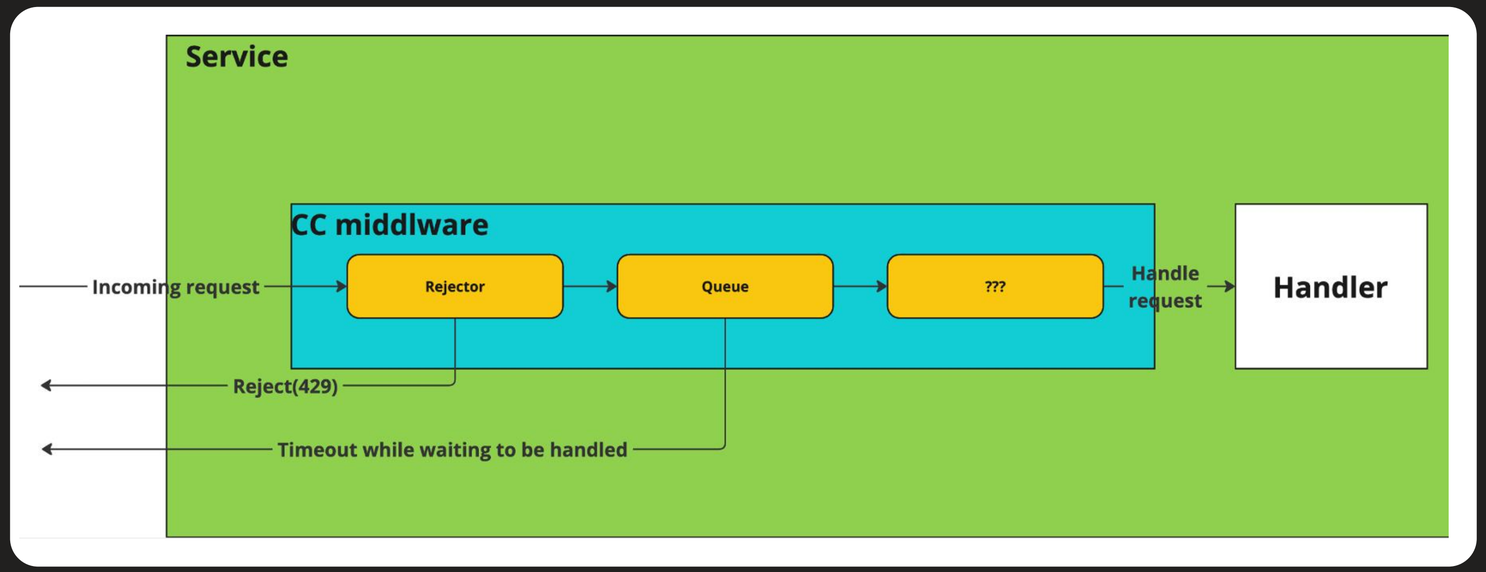

Loop Filter
Compa-
rator
P $K_p e(t)$
I $K_i \int_0^t e(\tau)d\tau$
D $K_d \frac{de(t)}{dt}$
r(t)
e(t)
u(t)
System
y(t)
r(t) = System Setpoint
e(t) = Error Signal
u(t) = Control Signal
y(t) = System Output


Setpoint
Underdamped
Overdamped
Critically damped
Position (m)
Time (s)
PID Controller Response Types

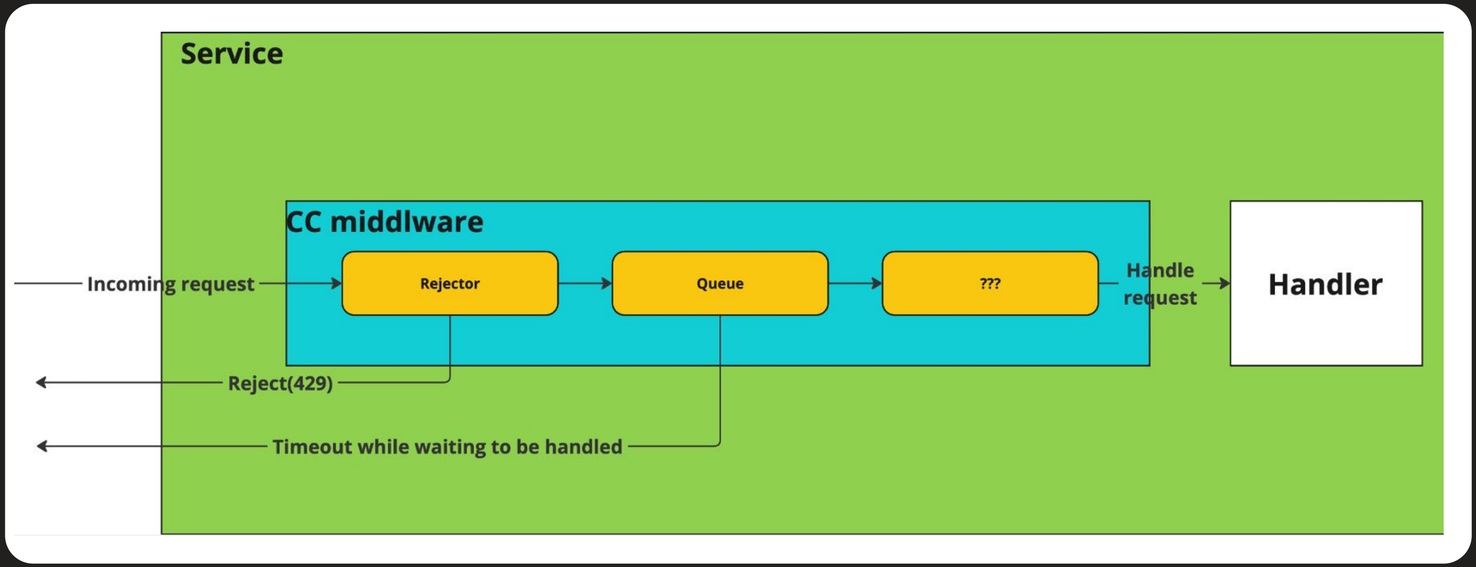

K_P too small
(X_P too large)
Process output
Time
good setting
Process output
Time
K_P too large
(X_P too small)
Process output
Time


T_d=0
T_d=0,2 с
T_d=0,5 с
r(t)
T_d=0,75 с
1,4
1
0,6
0,2
0
0
0,2
0,4
0,6
0,8
Время, с

Rejector


Service
CC middlware
Incoming request
Rejector
Queue
???
Handle request
Handler
Reject(429)
Timeout while waiting to be handled

Rejector

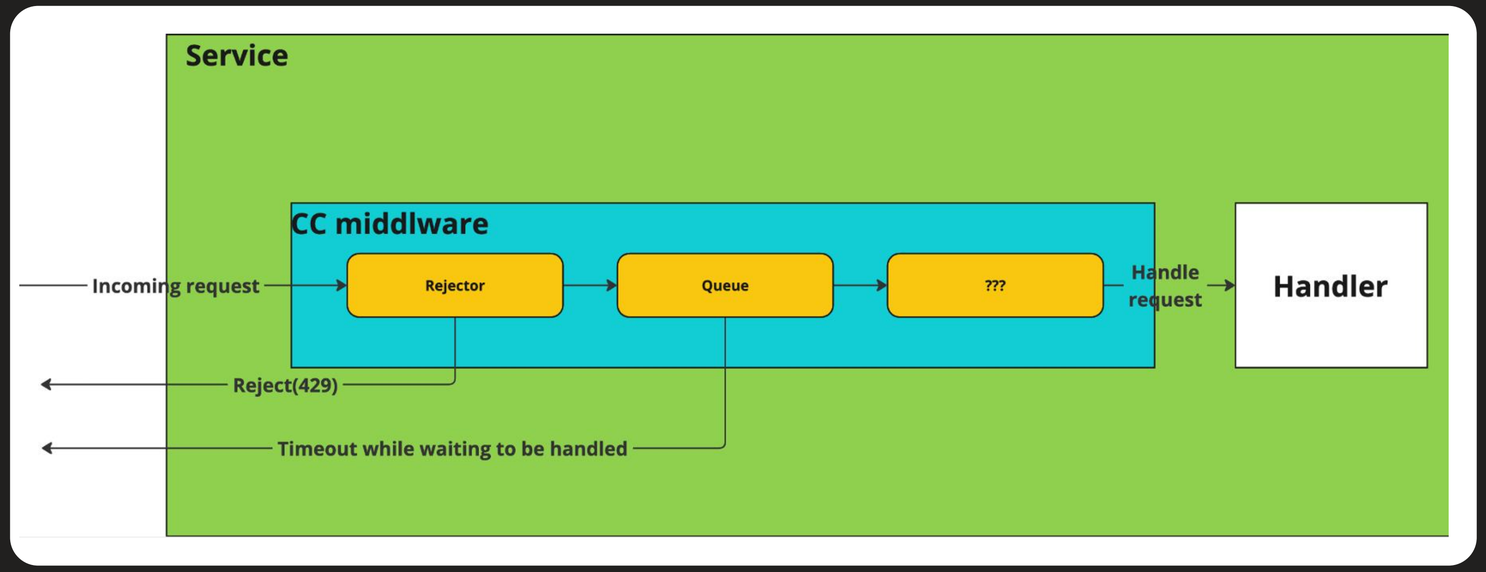


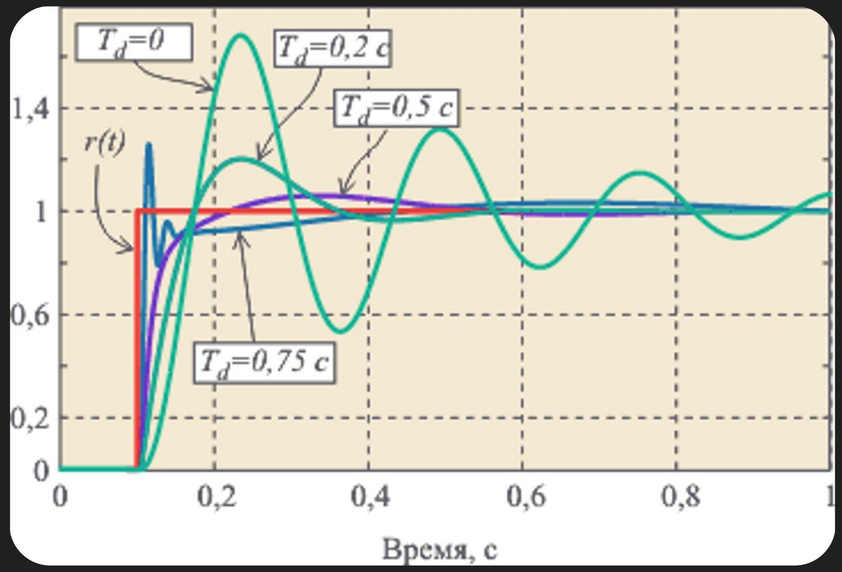
01 Как посчитать сколько reject-ить?

02 Как выбирать, что reject-им?

03 Почему тут очередь вместо мгновенной передачи запросов в handler?

Rejector



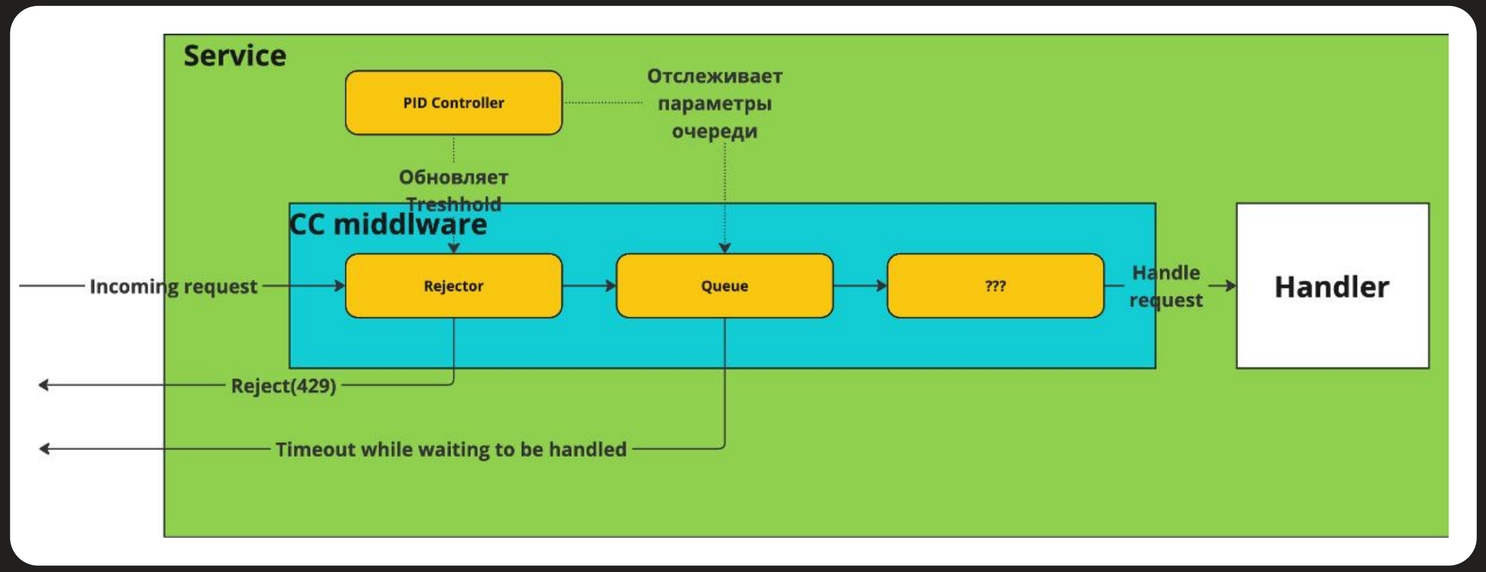


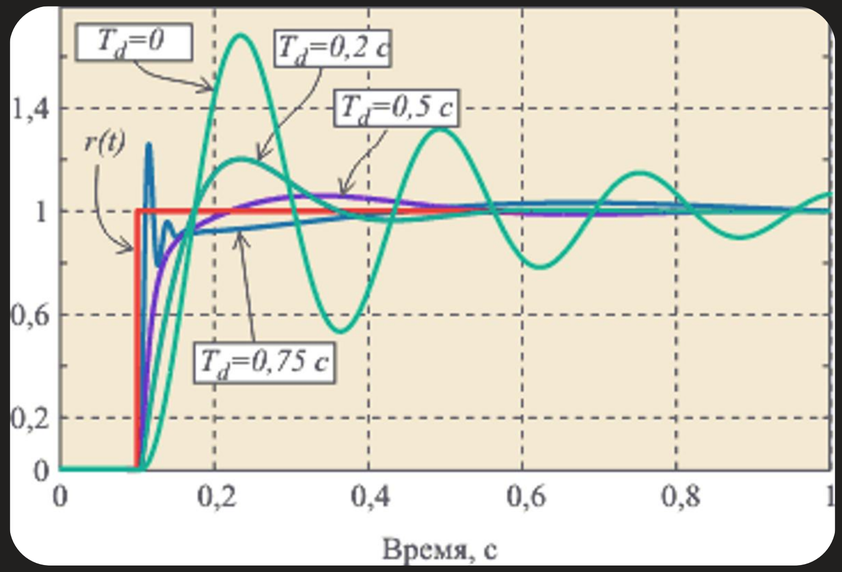
01 Как посчитать сколько reject-ить?

02 Как выбирать, что reject-им?

03 Почему тут очередь вместо мгновенной передачи запросов в handler?

Rejector





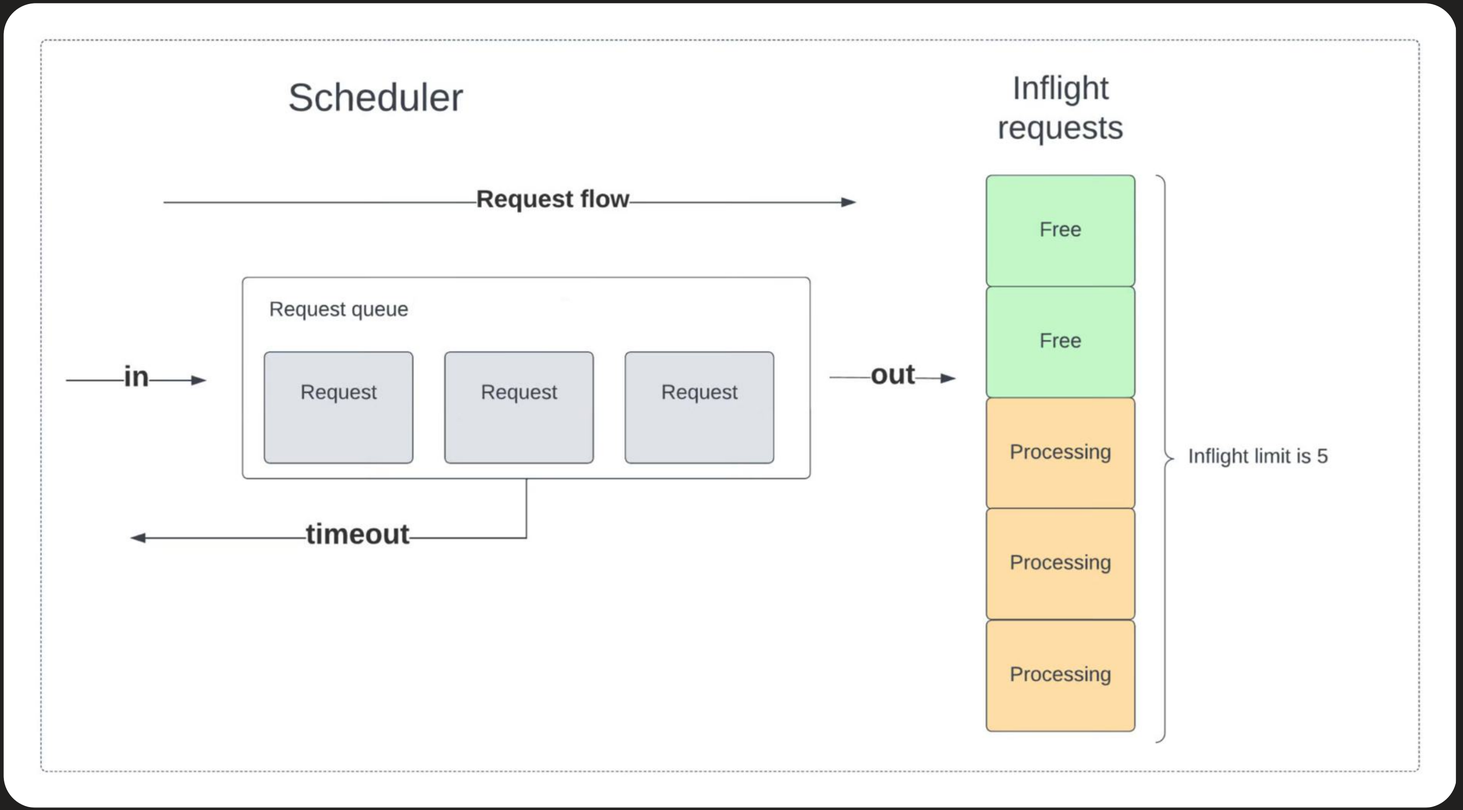
01 Как посчитать сколько reject-ить?

02 Как выбирать, что reject-им?

03 Почему тут очередь вместо мгновенной передачи запросов в handler?

Rejector

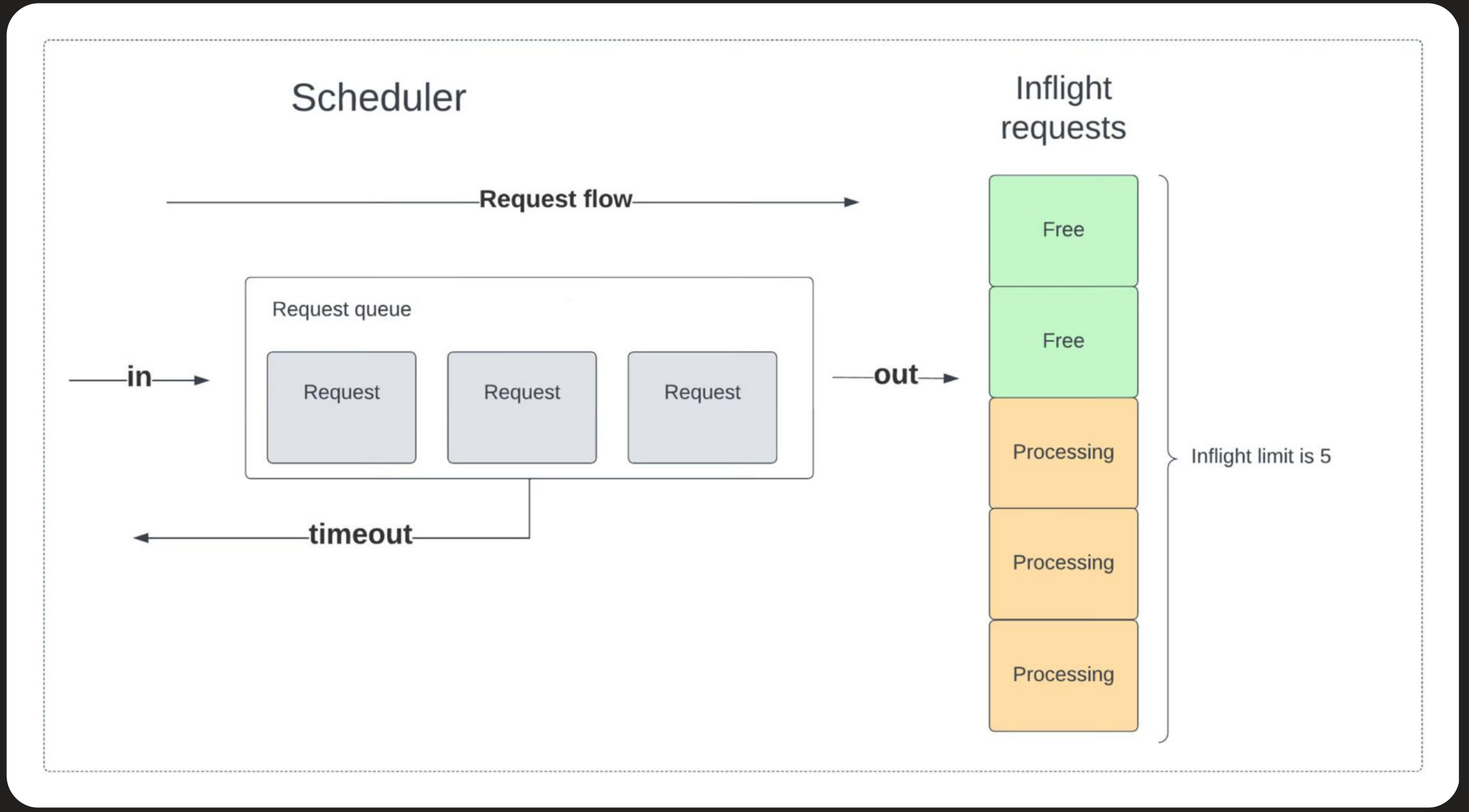
01 Считаем статистику



Rejector

01 Считаем статистику

02 Вычисляем отклонение от целевого значения



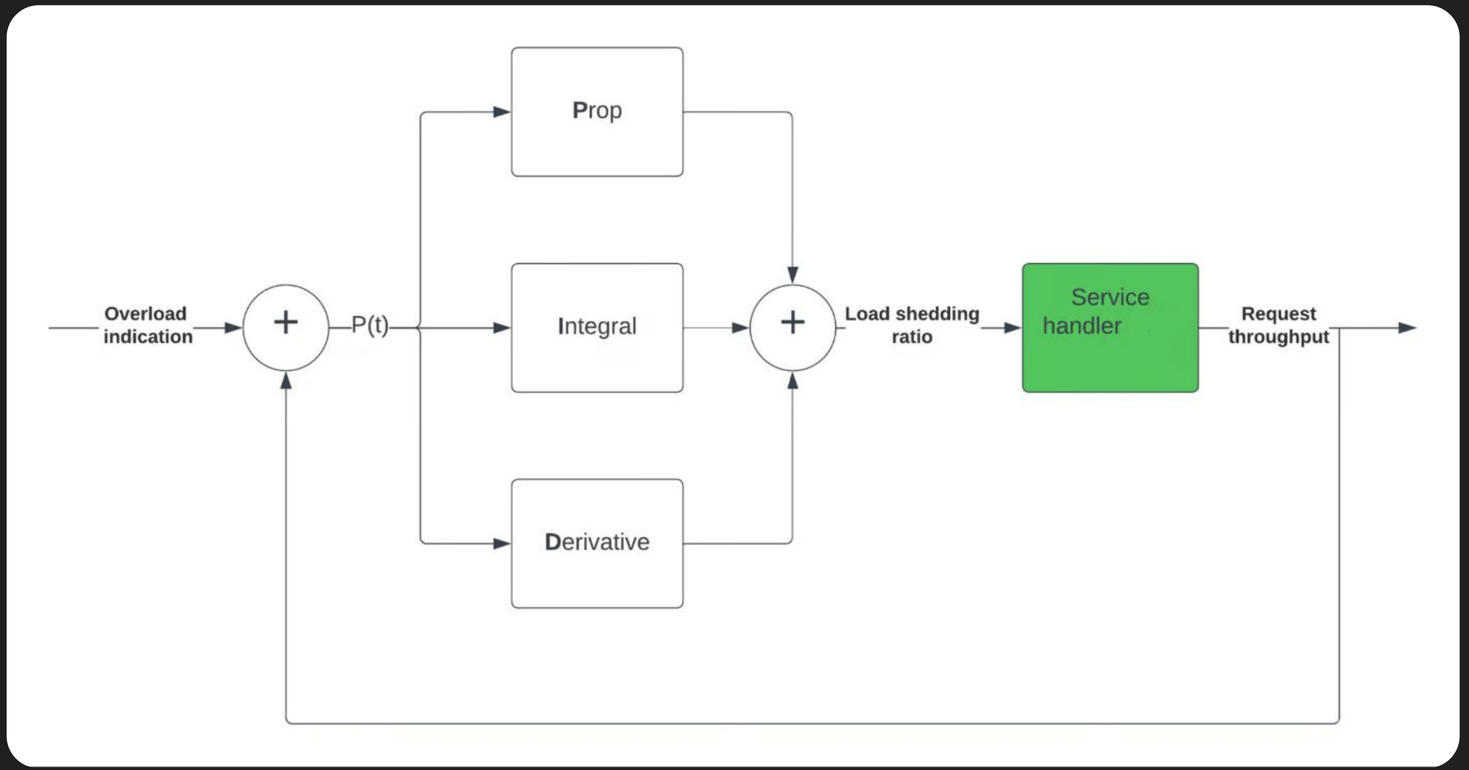
$$P(t) = \frac{in(t) - (out(t) + freeInflight(t))}{out'(t)}$$

Rejector

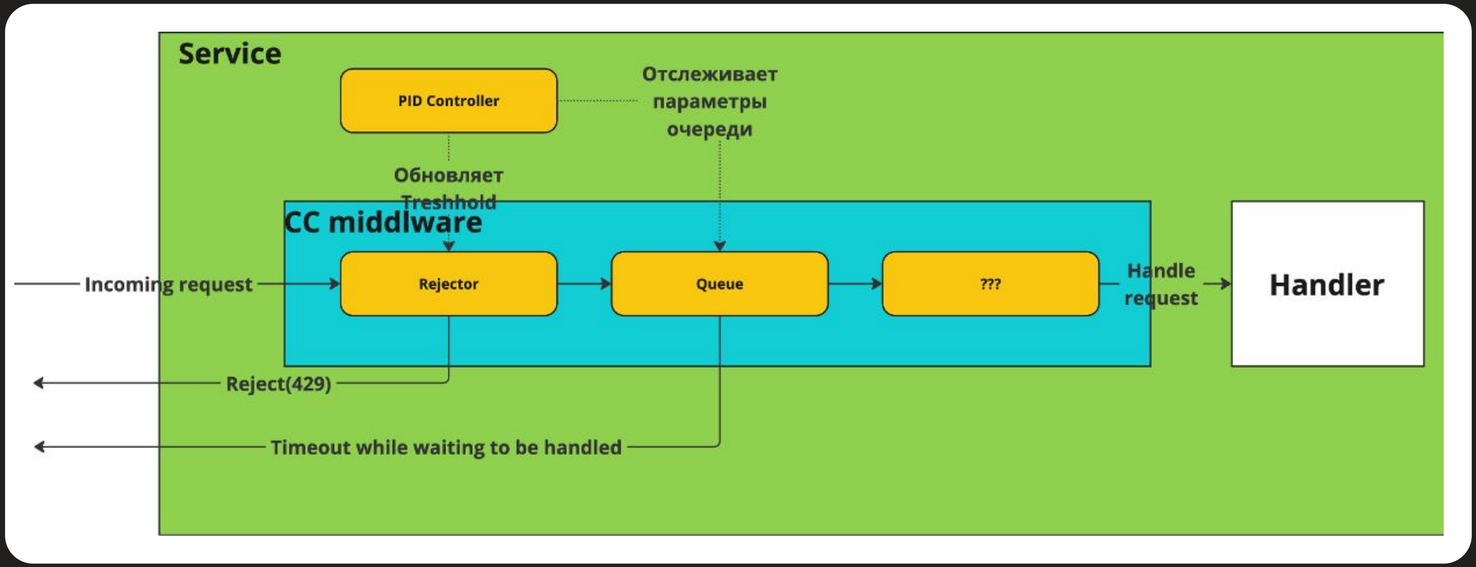
01 Считаем статистику

02 Вычисляем отклонение от целевого значения

03 Вычисляем долю отклоняемых запросов



Rejector



01 Как посчитать сколько reject-ить?

02 **Как выбирать, что reject-им?**

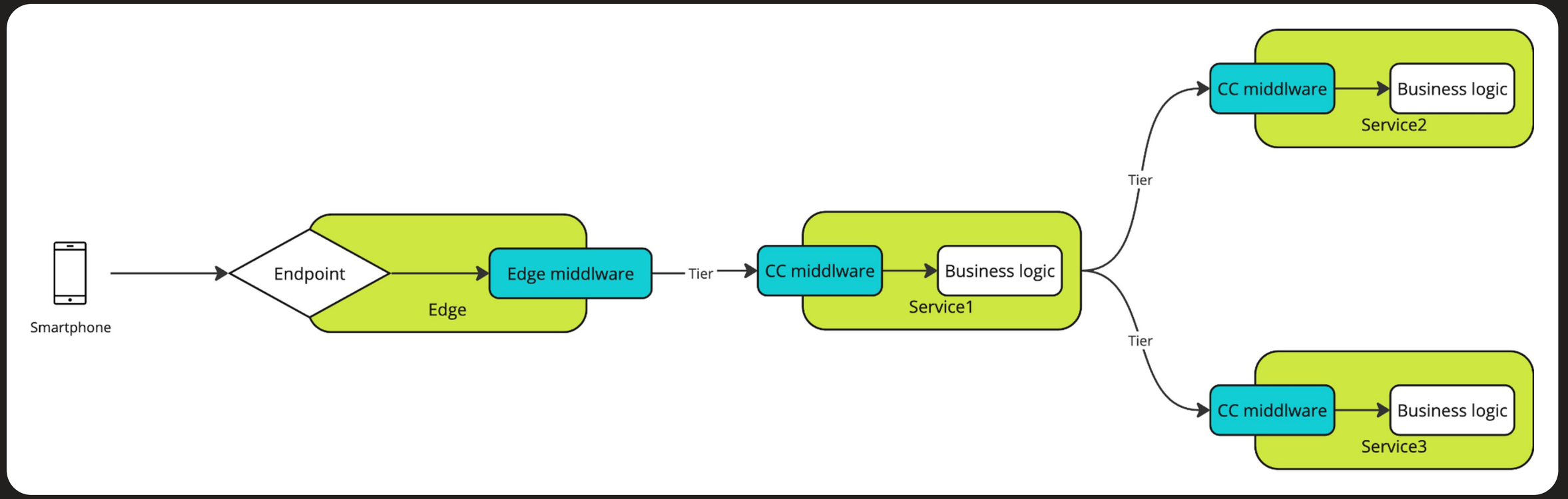
03 Почему тут очередь вместо мгновенной передачи запросов в handler?

А чего хочется

01 Резать в первую очередь некритичные запросы

Приоритизация трафика

Каждый запрос на edge proxy размечается тиром.
Тиры пробрасываются через всю трассировку



Приоритизация трафика

Tier

- Tier0 — цикл заказа
- Tier1 — UX
  ETA, доступность тарифов
- Tier2 — market efficiency
- Tier3 — редкие сценарии
  мультизаказ, грузовое такси
- Tier4 — delayable MQ-s
  начисление денег, пересчёт рейтинга
- Tier5 — аналитические кроны

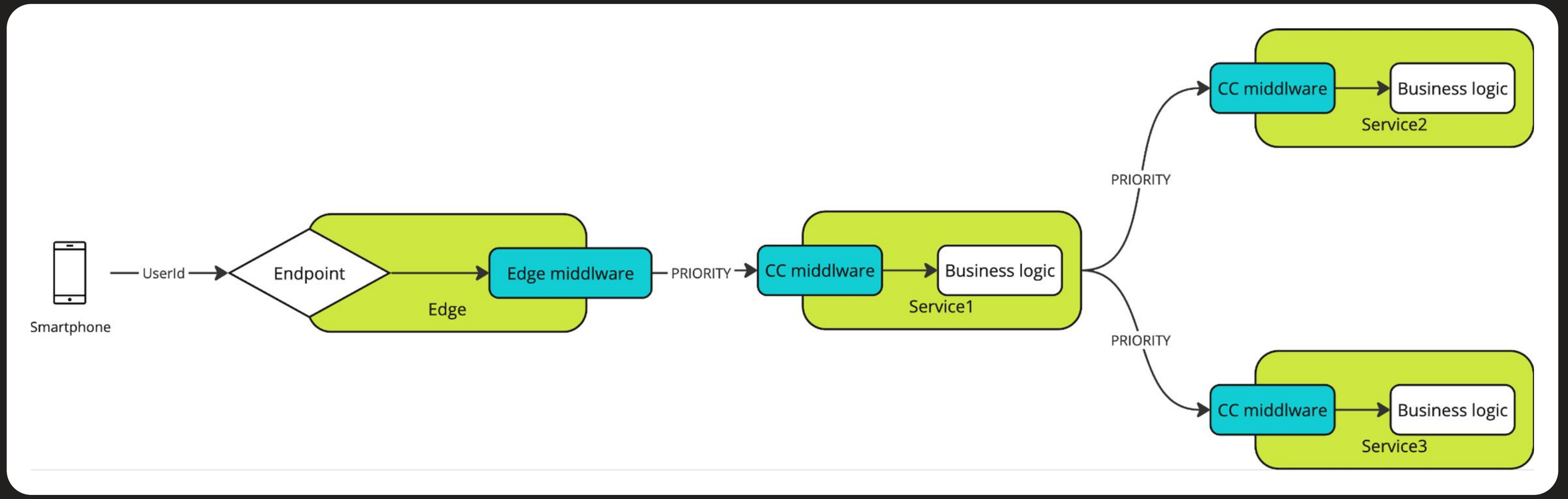
А чего хочется

01 Резать в первую очередь некритичные запросы

02 Если недостаточно отрезать некритичные запросы,
то предоставлять сервис для всех пользователей
в конечном счёте

Пробуем решить проблему комплексно
Приоритеты + когорты

Cohort = UserId % count(cohorts)
Priority = EndpointTier * count(cohorts) + Cohort

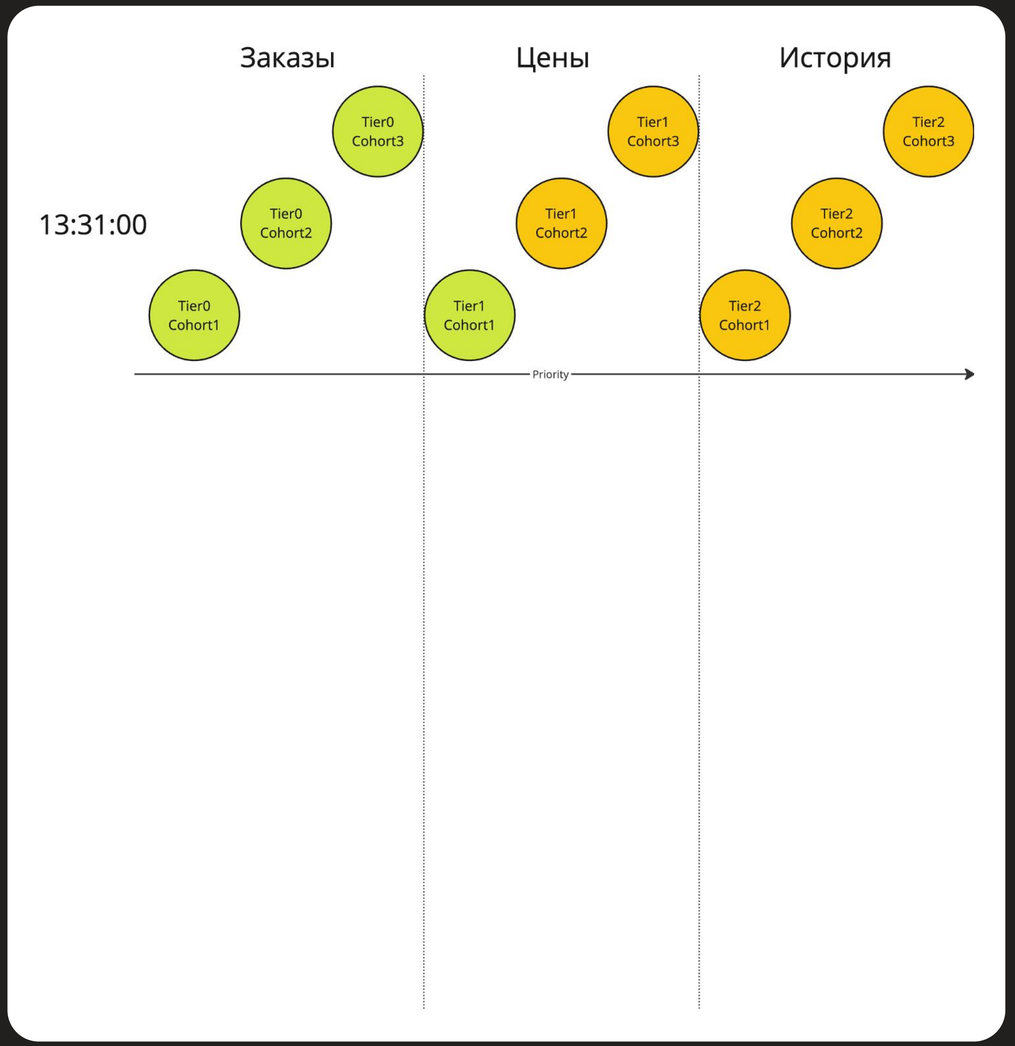


Приоритеты + когорты

Cohort = UserId % count(cohorts)
Priority = EndpointTier * count(cohorts) + Cohort

Приоритеты + когорты + ротация когорт

Обрабатываются запросы Priority <= 4

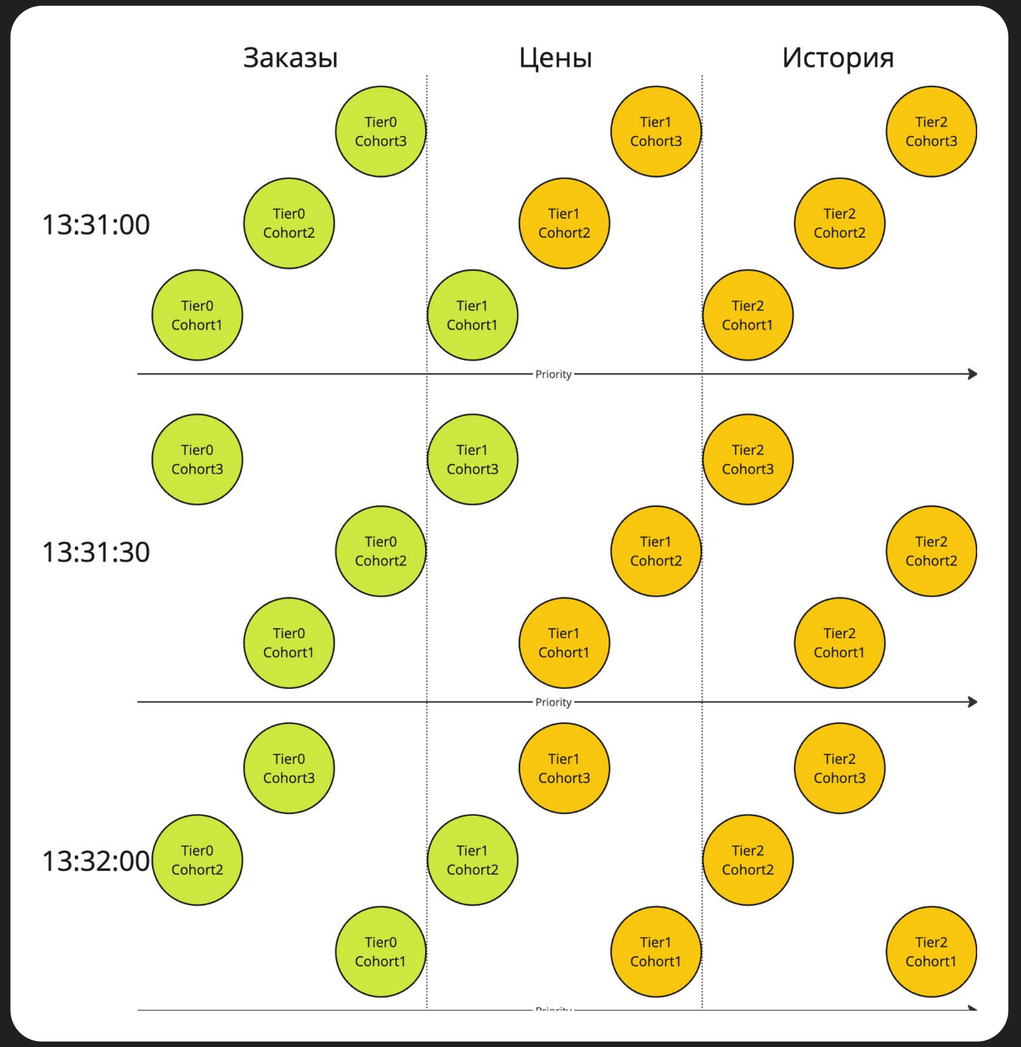


Приоритеты + когорты + ротация когорт

Обрабатываются запросы Priority <= 4

Каждые n секунд пользователиротируются в соседнюю когортусдвигом

Обслуживание получают все пользователи со временем



Приоритеты + когорты + ротация когорт

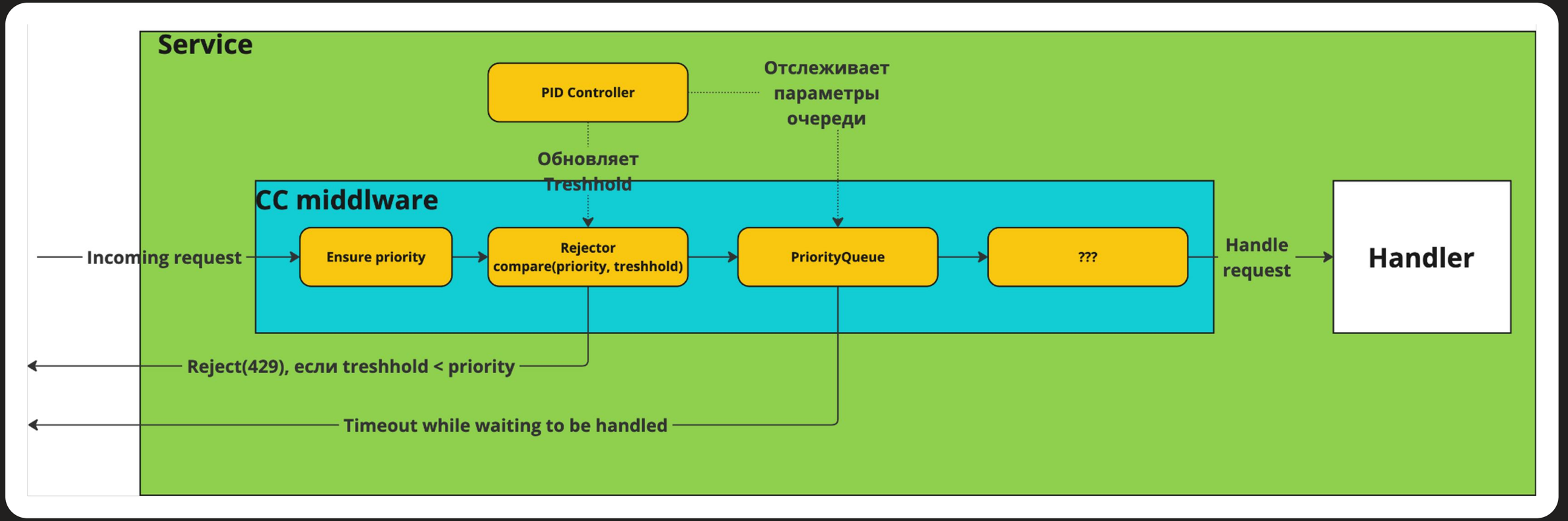
Обрабатываются запросы Priority <= 4

Каждые n секунд пользователиротируются
в соседнюю когортусдвигом

Обслуживание получают все пользователи
со временем

~~Cohort = UserId % count(c)~~
Cohort = ((UserId % count(c)) + floor(timestamp / n)) % count(c)

Rejector


Service
PID Controller
Отслеживает параметры очереди
Обновляет Treshhold
CC middlware
Incoming request
Ensure priority
Rejector compare(priority, treshhold)
PriorityQueue
???
Handle request
Handler
Reject(429), если treshhold < priority
Timeout while waiting to be handled

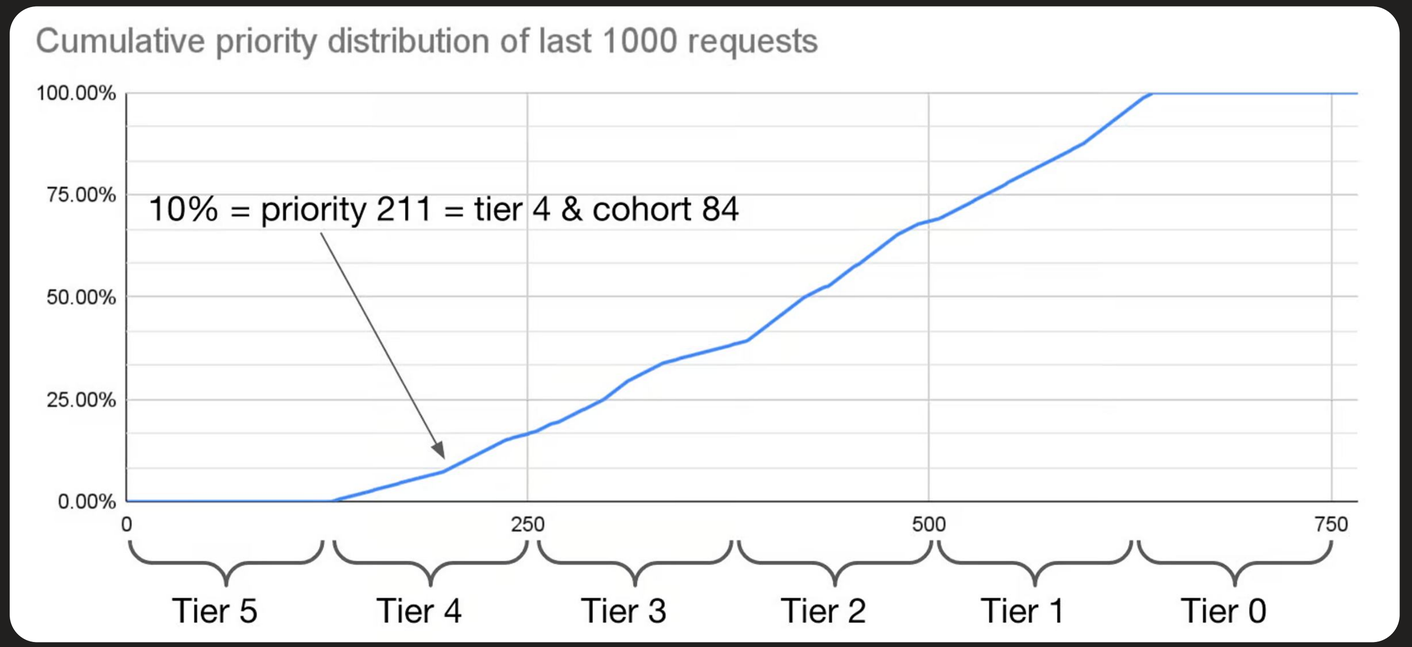
Rejector

01 Считаем статистику

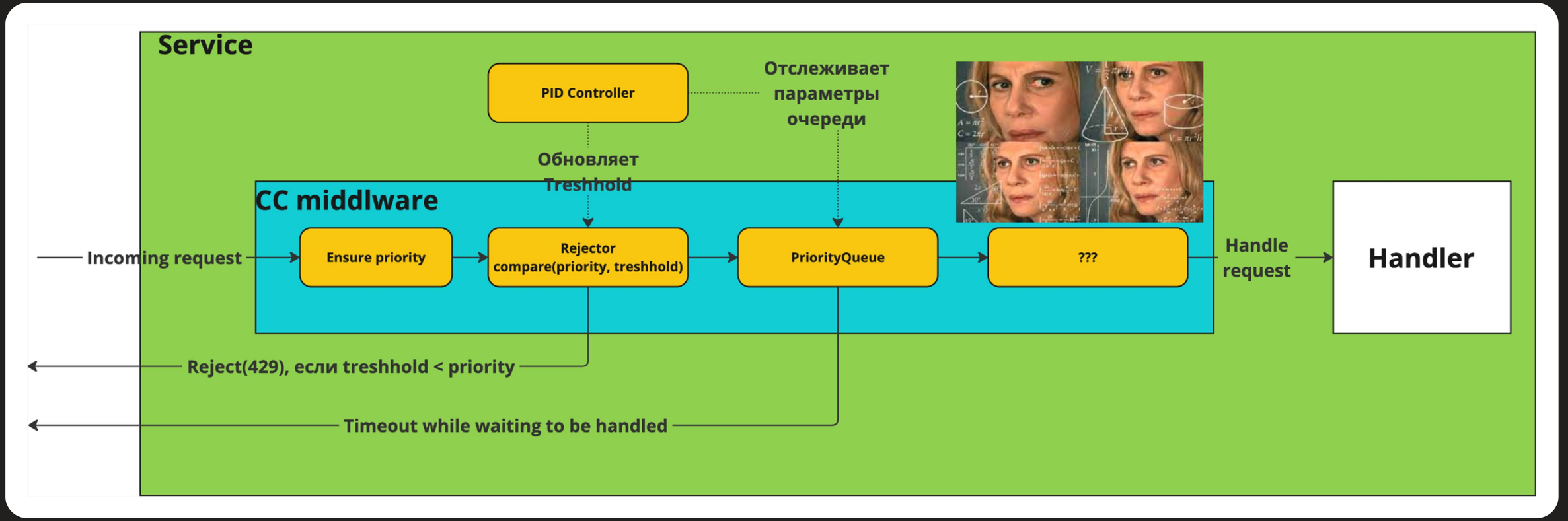
02 Вычисляем отклонение от целевого значения

03 Вычисляем долю отклоняемых запросов

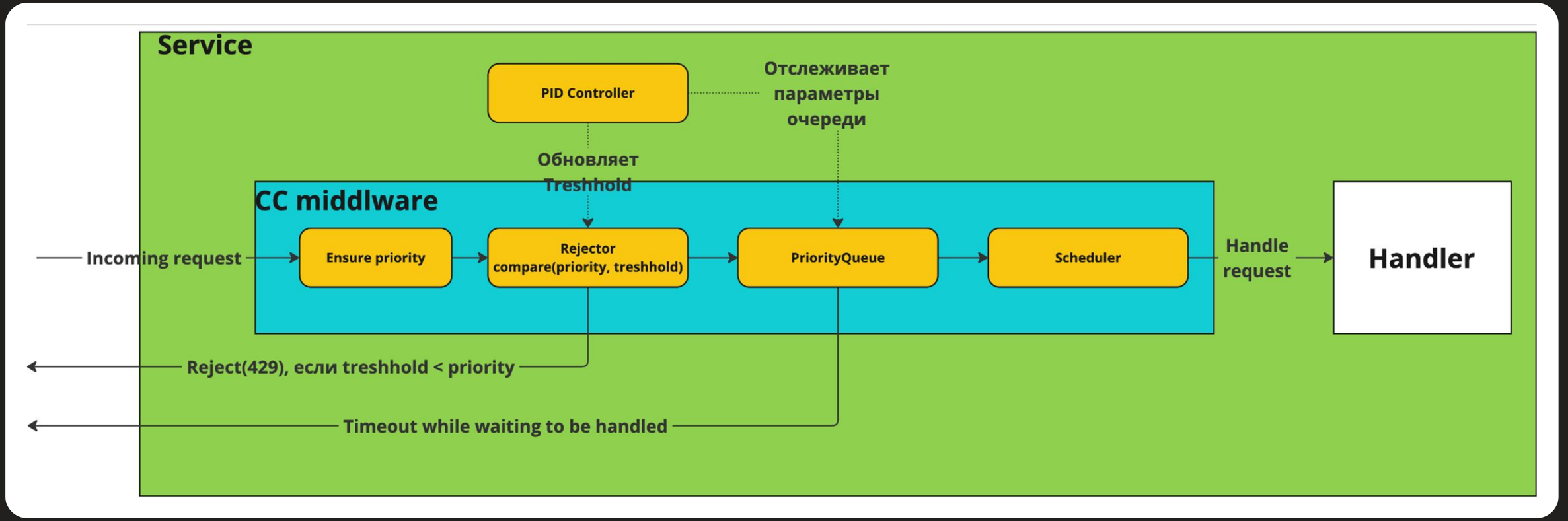
04 Считаем пороговый приоритет по статистическим данным распределения приоритетов



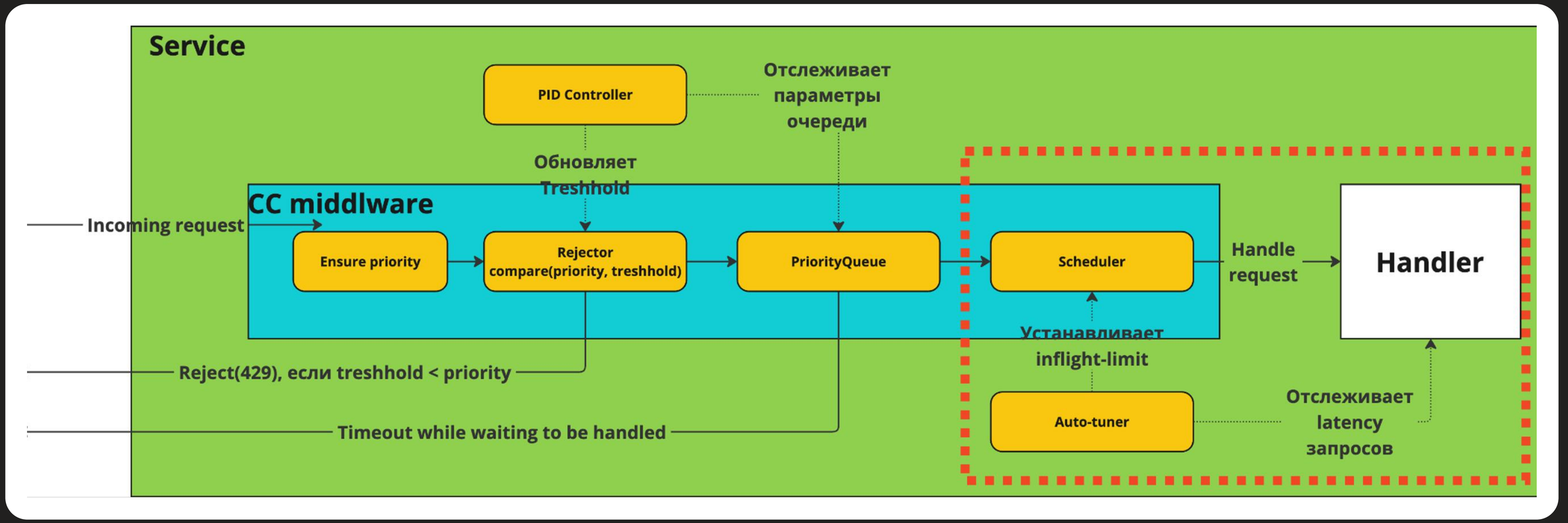
Rejector


Service
PID Controller
Отслеживает параметры очереди
Обновляет Treshhold
CC middlware
Incoming request
Ensure priority
Rejector compare(priority, treshhold)
PriorityQueue
???
Handle request
Handler
Reject(429), если treshhold < priority
Timeout while waiting to be handled

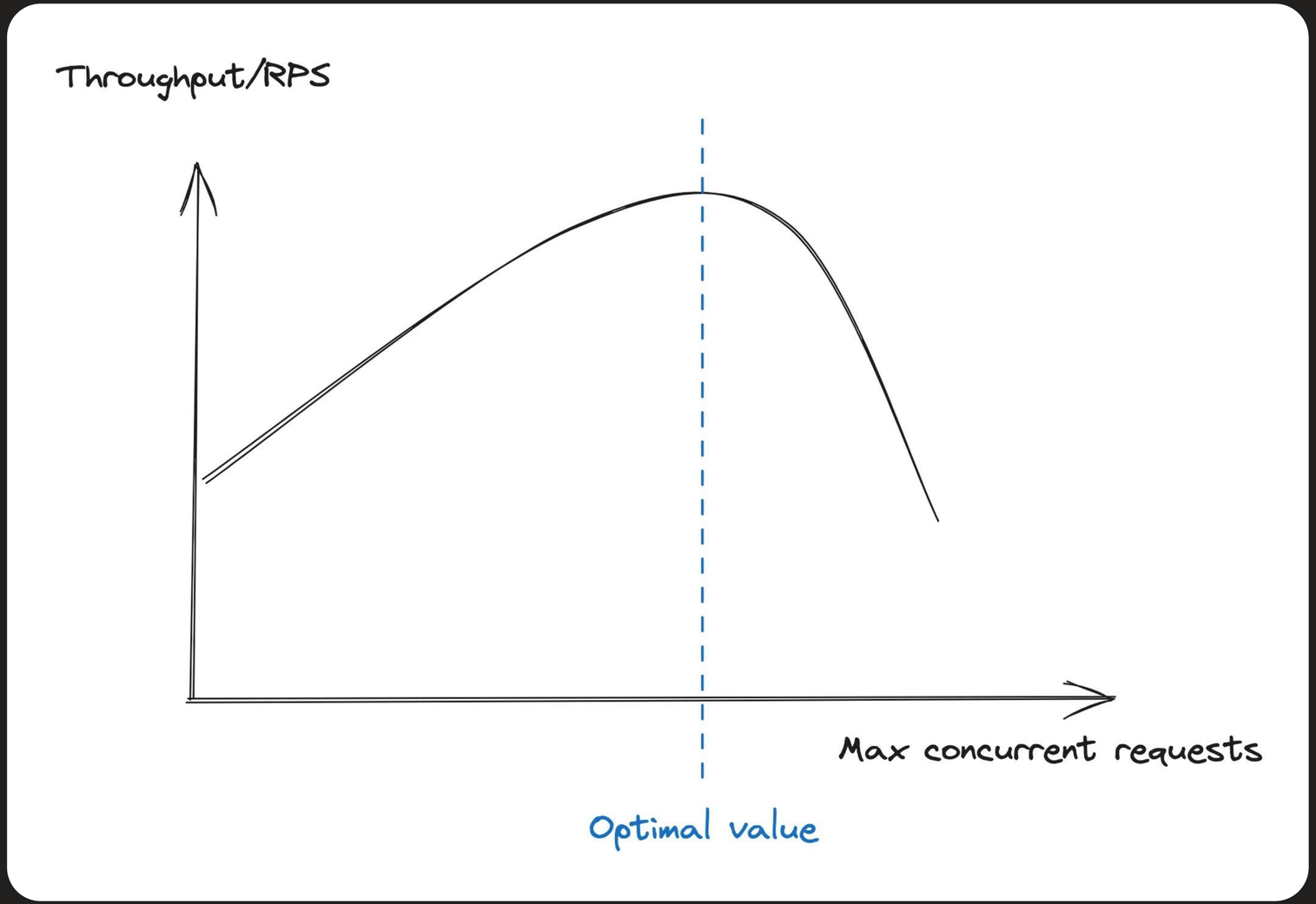
Scheduler


Service
PID Controller
Отслеживает параметры очереди
Обновляет Treshhold
CC middlware
Incoming request
Ensure priority
Rejector compare(priority, treshhold)
PriorityQueue
Scheduler
Handle request
Handler
Reject(429), если treshhold < priority
Timeout while waiting to be handled

Auto-tuner

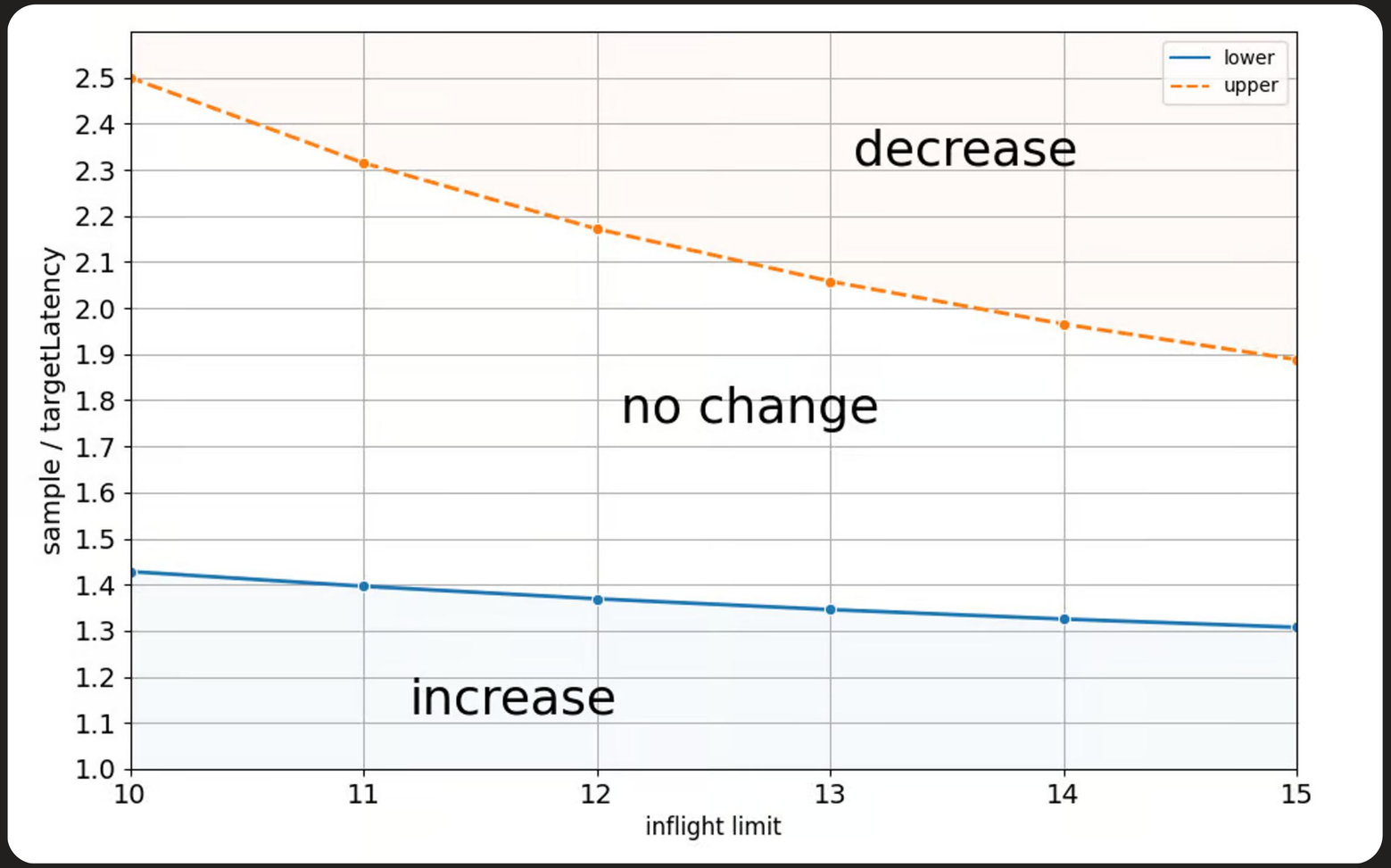

Service
PID Controller
Отслеживает параметры очереди
Обновляет Treshhold
CC middlware
Incoming request
Ensure priority
Rejector compare(priority, treshhold)
PriorityQueue
Scheduler
Handle request
Handler
Устанавливает inflight-limit
Reject(429), если treshhold < priority
Timeout while waiting to be handled
Auto-tuner
Отслеживает latency запросов

Auto-tuner



Auto-tuner

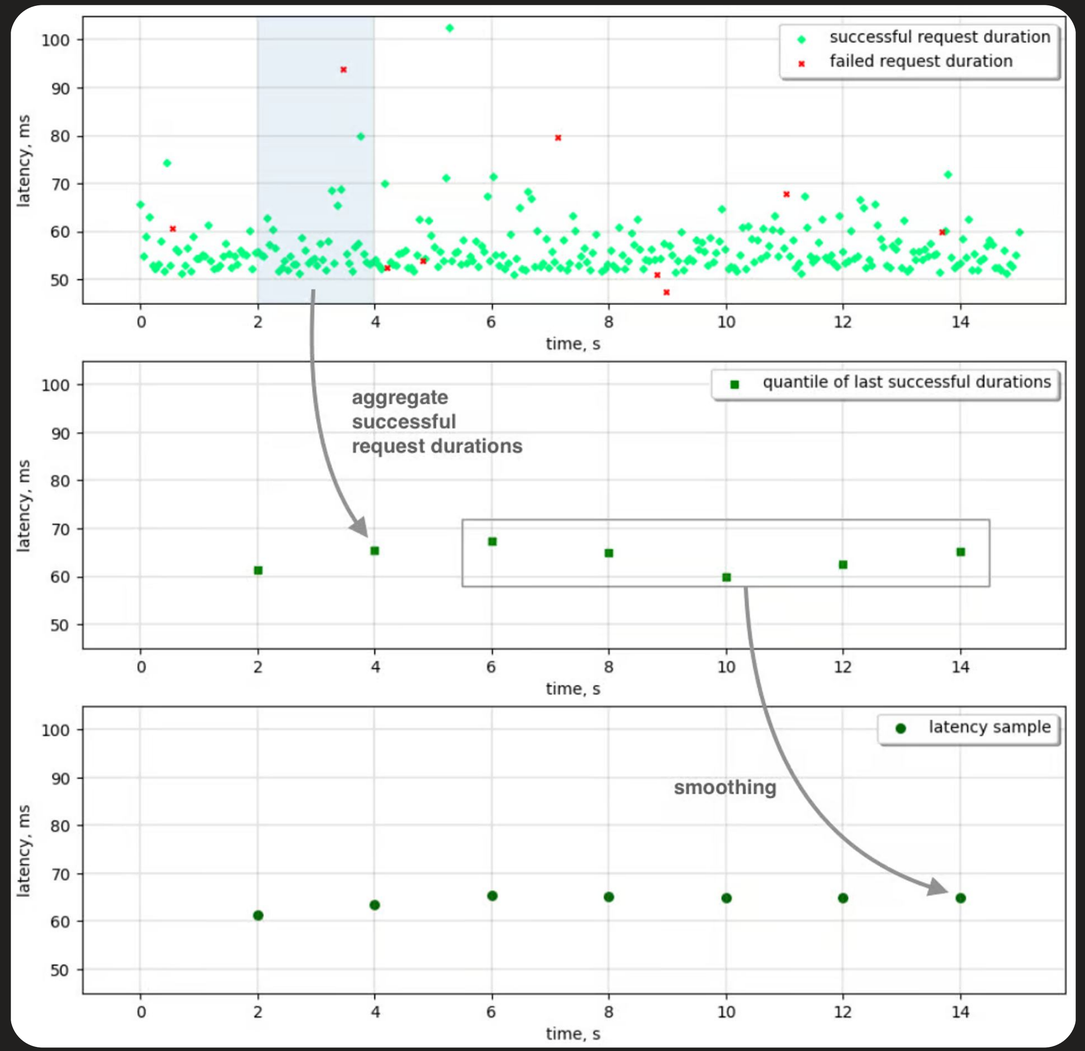
01 TCP-Vegas



Auto-tuner

01 TCP-Vegas

02 С сэмплированием latency

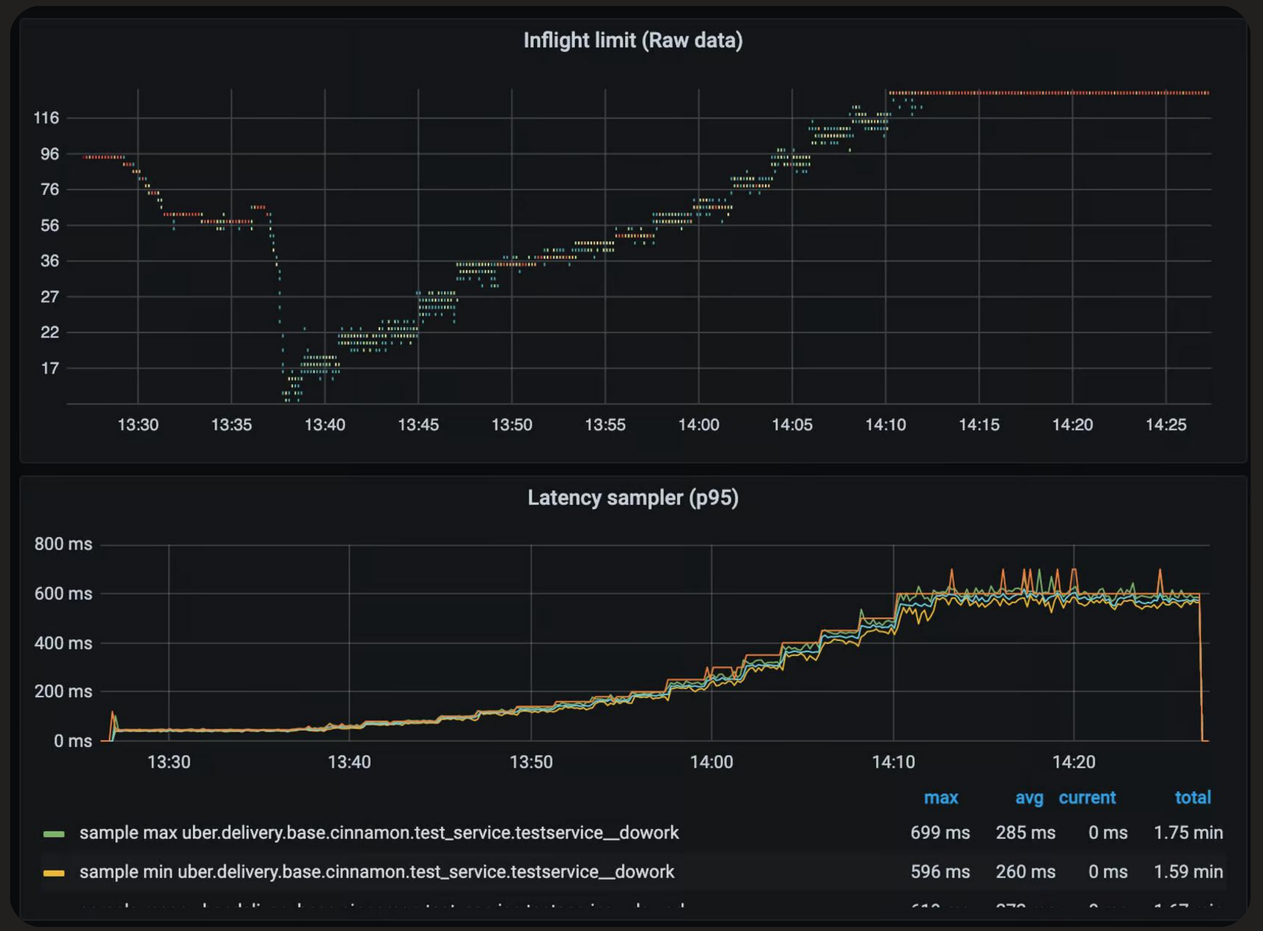


Auto-tuner

01 TCP-Vegas

02 С сэмплированием latency

03 С периодическим сбросом значений, чтобы избежать бесконечного роста числа обработчиков



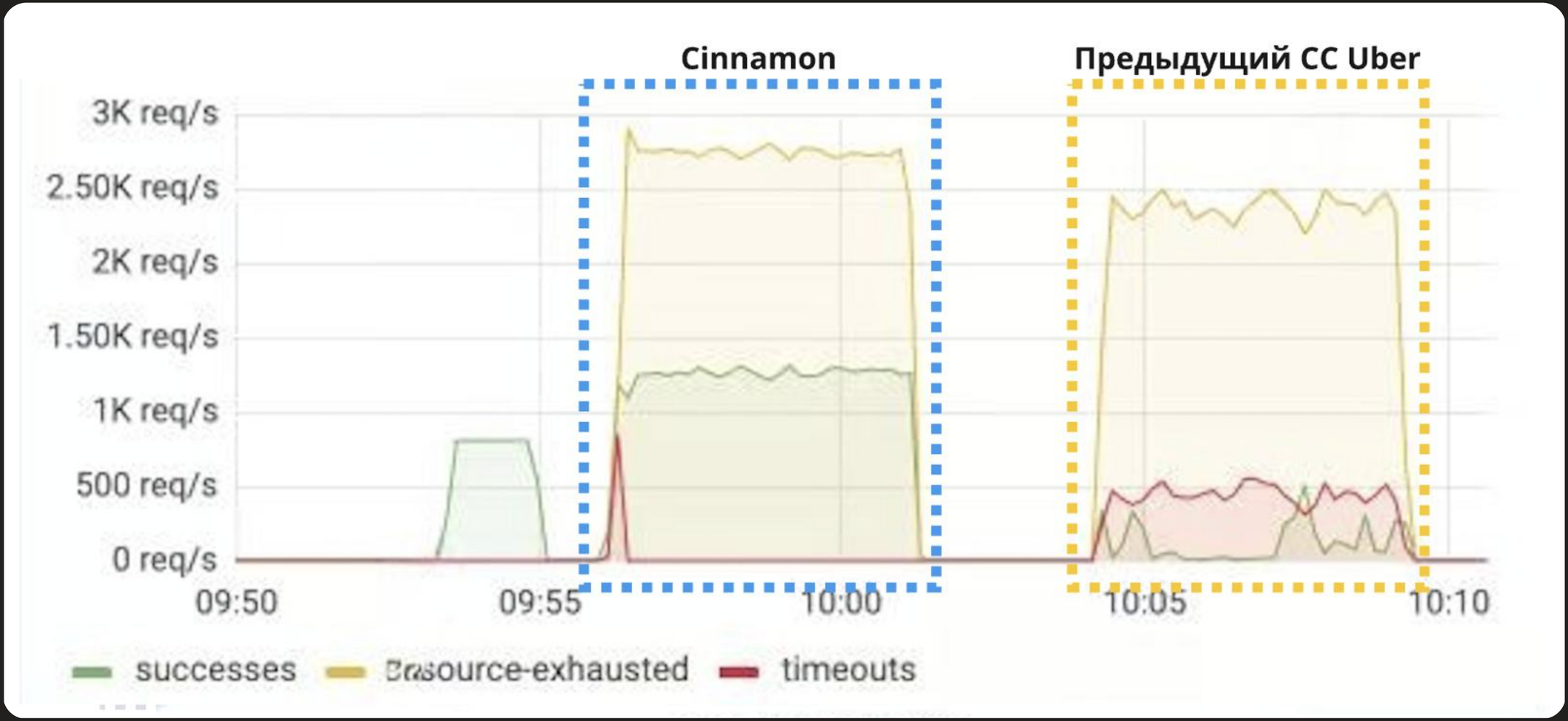
Auto-tuner

01 TCP-Vegas

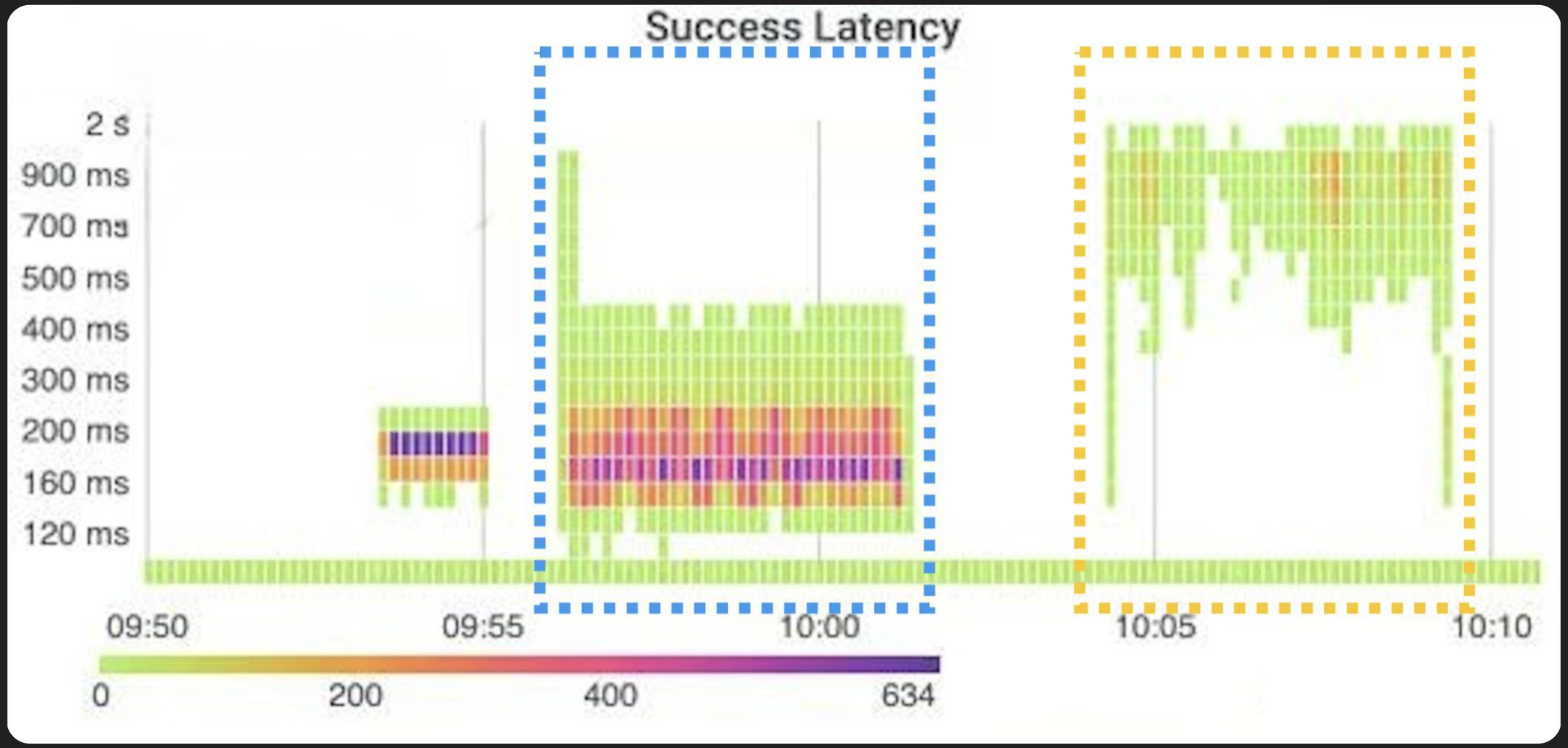
02 С сэмплированием latency

03 С периодическим сбросом значений, чтобы избежать бесконечного роста числа обработчиков

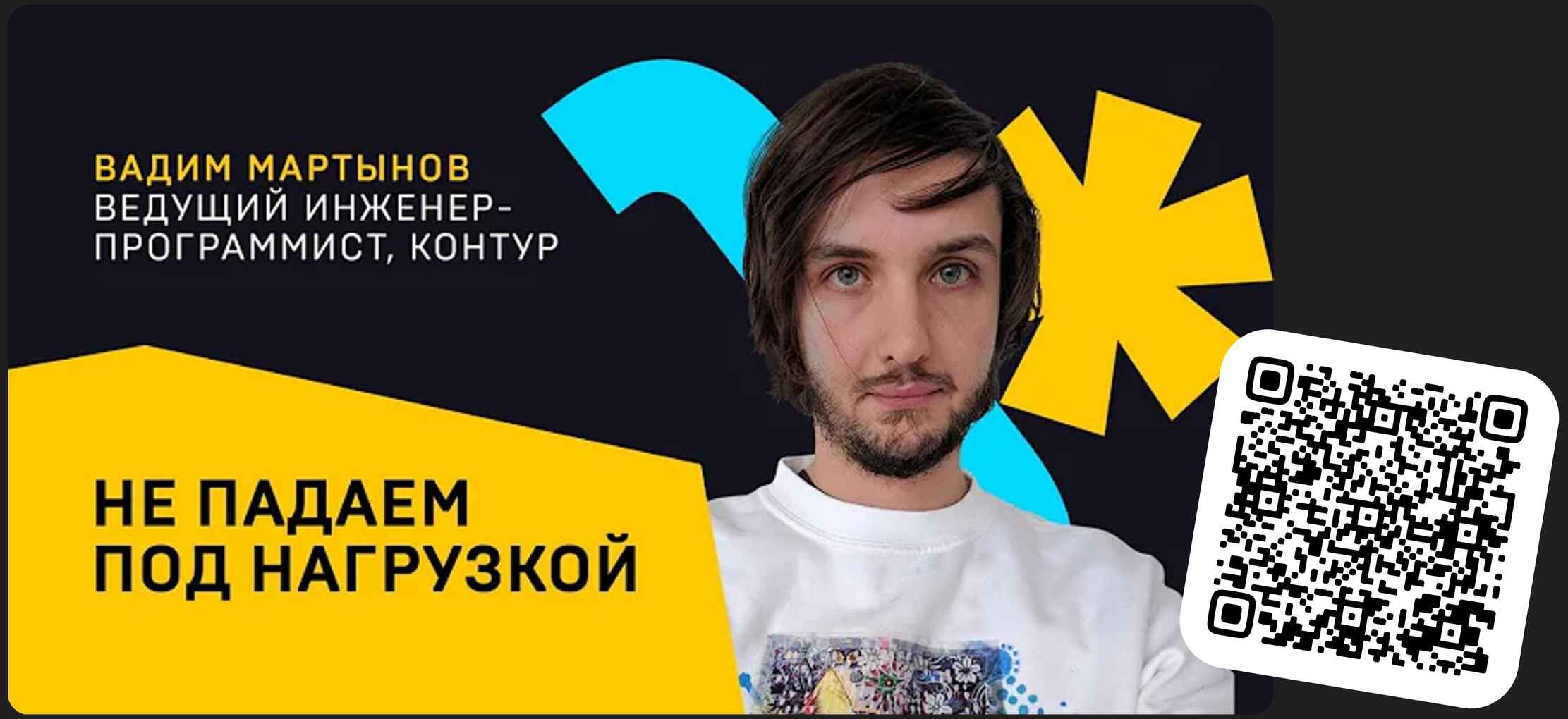
Замеры производительности Uber



Замеры производительности Uber



Cinnamon. Замеры производительности



Замеры производительности в Яндекс Go

Вадим Мартынов

Замеры производительности в Яндекс Go

Критичные запросы

	3580253 (Cinnamon)	3580266 (Userver CC)
200 - OK	71.488 (+18.852)	52.636 %
429 - Too Many Requests	28.512 (-18.852)	47.364 %

Замеры производительности в Яндекс Go

Критичные запросы

	3580253 (Cinnamon)	3580266 (Userver CC)
200 - OK	71.488 (+18.852)	52.636 %
429 - Too Many Requests	28.512 (-18.852)	47.364 %

Некритичные запросы

	3580253 (Cinnamon)	3580266 (Userver CC)
200 - OK	42.066 (-10.830)	52.896 %
429 - Too Many Requests	57.934 (+10.830)	47.104 %

Все запросы

	3580253 (Cinnamon)	3580266 (Userver CC)
200 - OK	56.777 (+4.011)	52.766 %
429 - Too Many Requests	43.223 (-4.011)	47.234 %

Cinnamon ❤️ Taxi

As is

- Продуктовые деградации для снятия нагрузки отрезанием второстепенных фич

To be

- Приоритеты для отрезания функциональности автоматикой

Cinnamon ❤️ Taxi

As is

- Продуктовые деградации для снятия нагрузки отрезанием второстепенных фич
- Срабатывание congestion control приводит к скачкам latency

To be

- Приоритеты для отрезания функциональности автоматикой
- Latency сохраняется при повышенной нагрузке

Cinnamon ❤️ Taxi

As is

- Продуктовые деградации для снятия нагрузки отрезанием второстепенных фич
- Срабатывание congestion control приводит к скачкам latency
- Настраиваем rate-limiter для защиты БД и зависимостей
- Делаем CC в клиентах к зависимостям

To be

- Приоритеты для отрезания функциональности автоматикой
- Latency сохраняется при повышенной нагрузке
- Cinnamon защищает зависимости, ориентируясь на latency и регулирует inflight requests

Cinnamon ❤️ Taxi

As is

- Продуктовые деградации для снятия нагрузки отрезанием второстепенных фич
- Срабатывание congestion control приводит к скачкам latency
- Настраиваем rate-limiter для защиты БД и зависимостей
- Делаем CC в клиентах к зависимостям
- Пользователи по гео могут влиять на всю систему

To be

- Приоритеты для отрезания функциональности автоматикой
- Latency сохраняется при повышенной нагрузке
- Cinnamon защищает зависимости, ориентируясь на latency и регулирует inflight requests
- Ротация когорт частично решает проблему влияния части пользователей на всю систему

Открытые вопросы
правильный ответ — «зависит»

Cinnamon ❤️ ваш сервис

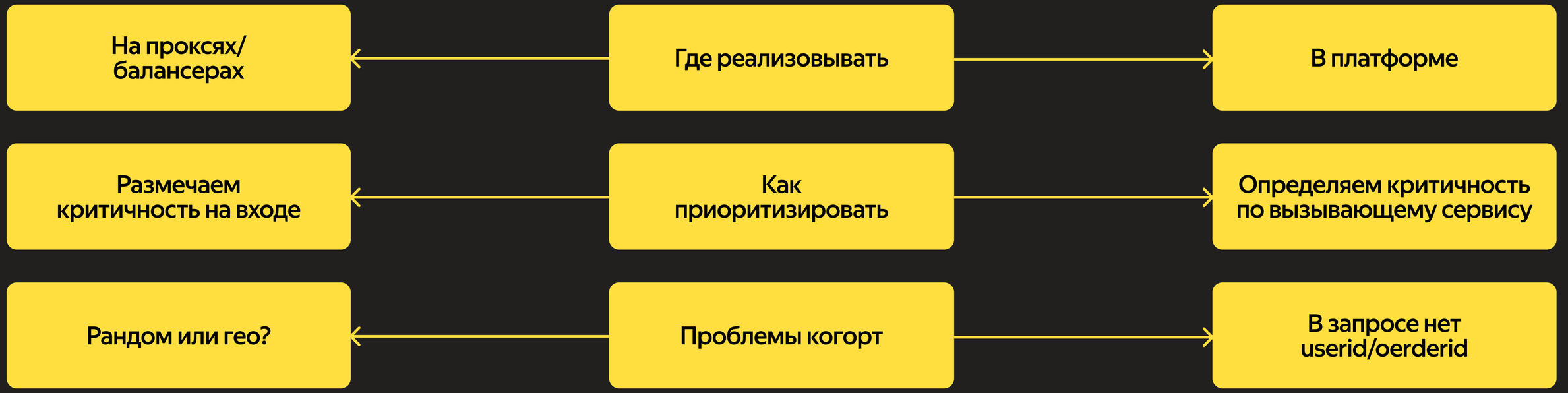
Открытые вопросы
правильный ответ — «зависит»

Cinnamon ❤️ ваш сервис

Открытые вопросы
правильный ответ — «зависит»

Cinnamon ❤️ ваш сервис

Открытые вопросы
правильный ответ — «зависит»



Ссылки и контакты

tg: https://t.me/Vadimyan

Как строится надежность Яндекс Такси / Александр Фишер https://vk.com/video-151223562_456239601

Сложность метастабильных состояний отказа на примере Такси / Яндекс Go Infra Meetup #2 https://youtu.be/HM1Ua_X7O-I

Математика надежности / Яндекс Go Dev Day&Night https://youtu.be/8JpMTF0wY98

Reliability, constant work, and a good cup of coffee https://aws.amazon.com/ru/builders-library/reliability-and-constant-work/

Не падаем под нагрузкой / Вадим Мартынов https://youtu.be/lwBGNuMd5oQ

Троттлинг от А до AAAAAAA!!11 | Вадим Мартынов, Контур | HighLoad Meetup Tgn https://youtu.be/QTXXyi78Xdk

Инструменты надежности Такси \ Александр Фишер, Яндекс Такси https://youtu.be/ATTXtS7Dzqg

Cinnamon: Using Century Old Tech to Build a Mean Load Shedder https://www.uber.com/blog/cinnamon-using-century-old-tech-to-build-a-mean-load-shedder/

Metastable Failures in Distributed Systems https://sigops.org/s/conferences/hotos/2021/papers/hotos21-s11-bronson.pdf

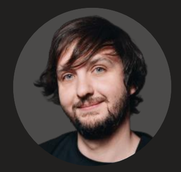
Metastable Failures in the Wild https://www.usenix.org/publications/loginonline/metastable-failures-wild

Fail at Scale. Reliability in the face of rapid change / Ben Maurer, Facebook https://youtu.be/dlixGkeIP9U

Metastable Failures in Distributed Systems / University of New Hampshire https://sigops.org/s/conferences/hotos/2021/papers/hotos21-s11-bronson.pdf

Яндекс Go

Спасибо за внимание!



Вадим Мартынов

Руководитель команды
платформы надёжности в Яндекс Go