Меня хорошо слышно && видно?

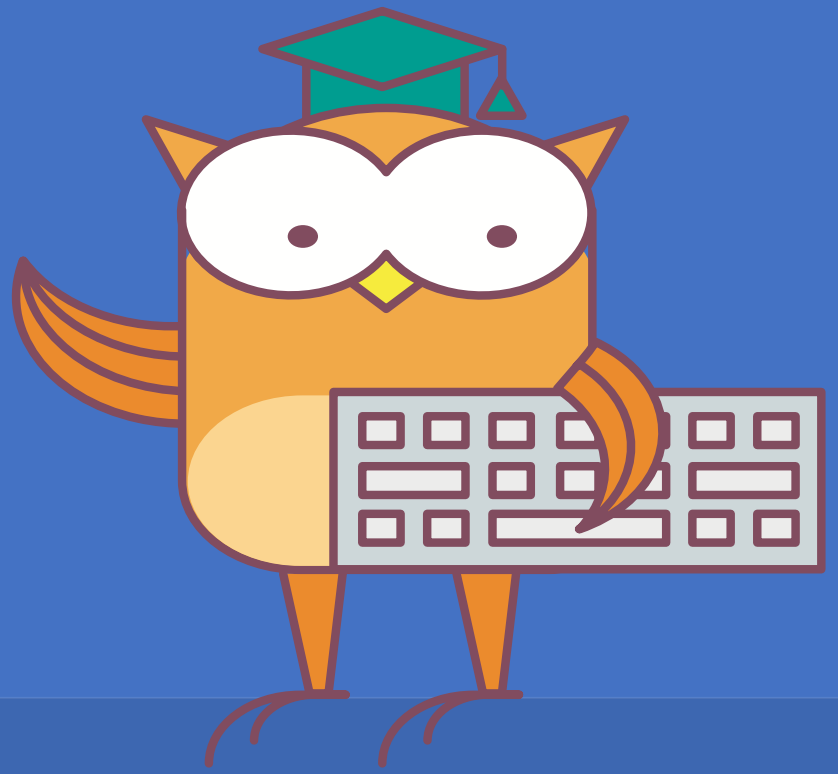
Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

OTUS

Тестирование в микросервисах

Архитектор ПО



Щетинников Стас

Карта вебинара

- Нагрузочное тестирование
- Тестирование в продакшне

01

Нагрузочное тестирование

Нагрузочное тестирование

Под нагрузкой:

- Как будет себя вести новая система, которую мы вводим в эксплуатацию?
- Что будет если мы обновим версию БД?

Нагрузочное тестирование

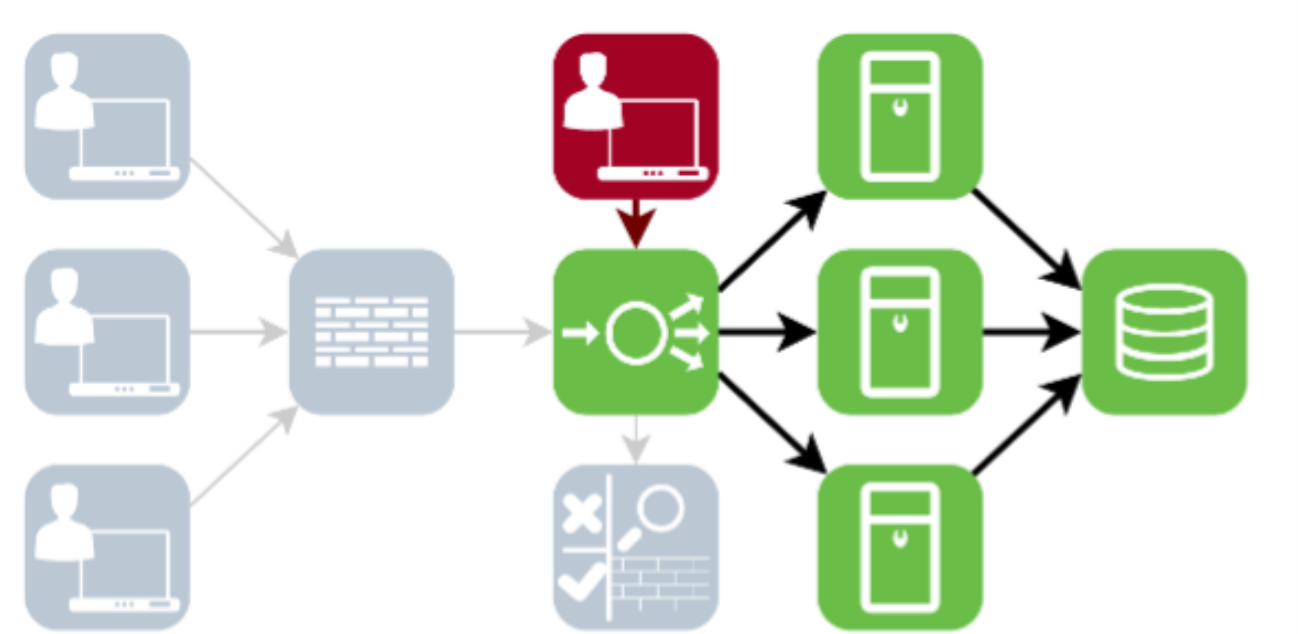
Основные задачи нагрузочного тестирования

- Определение пиковой производительности
- Выявление узких мест
- Проверка надежности

Нагрузочное тестирование

Что можно нагружать?

- Железо
- Отдельные компоненты – БД, балансировщик
- **Приложение или сервис в целом**



Нагрузочное тестирование

Как понять, что происходит с приложением?

- Метрики по latency/rps/error rate
- Метрики по утилизации ресурсов
- Метрики по отдельным компонентам внутри приложения

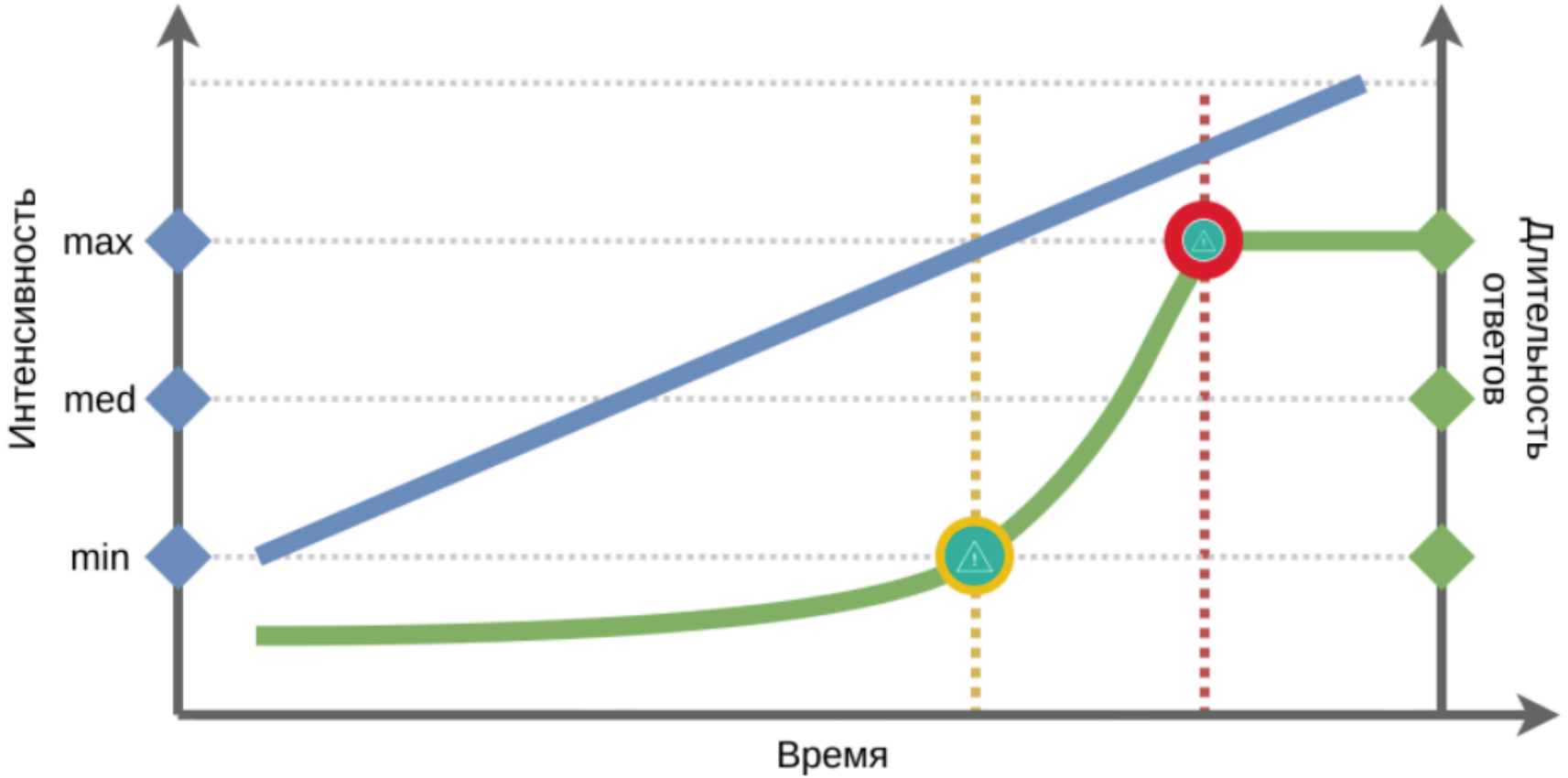
Нагрузочное тестирование

Тестирование производительности:

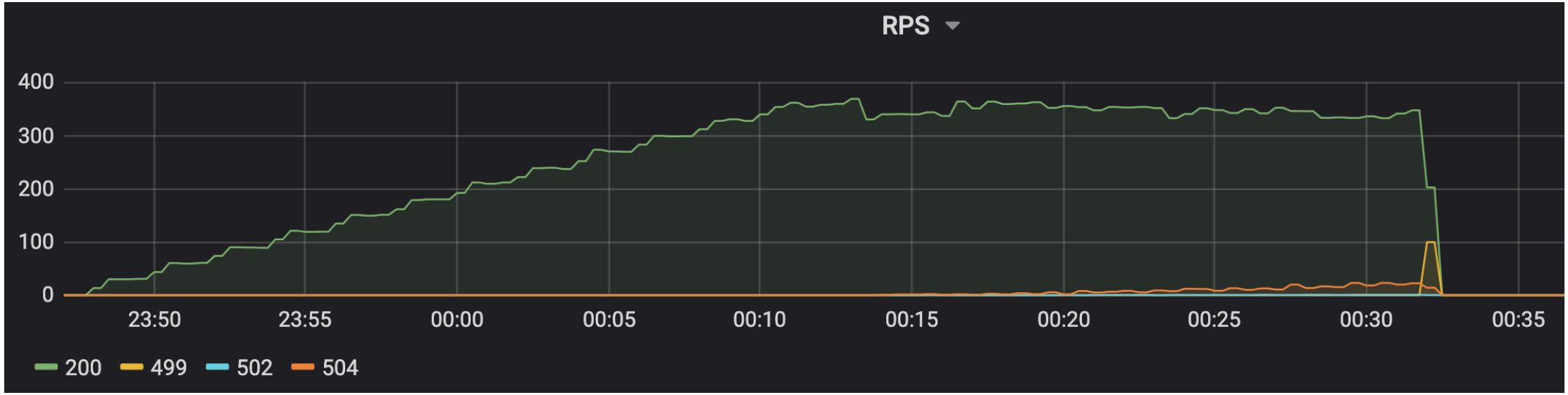
- Нагрузочное
- Стабильности
- Объемное

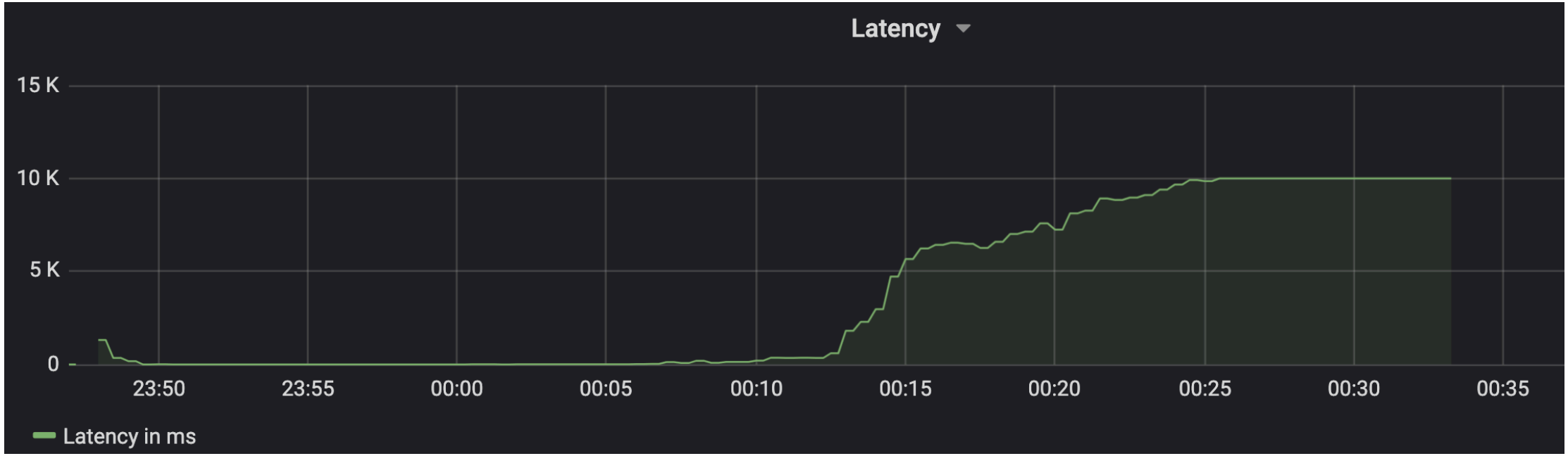
Нагрузочное тестирование

Увеличиваем нагрузку и ищем точку деградации сервиса



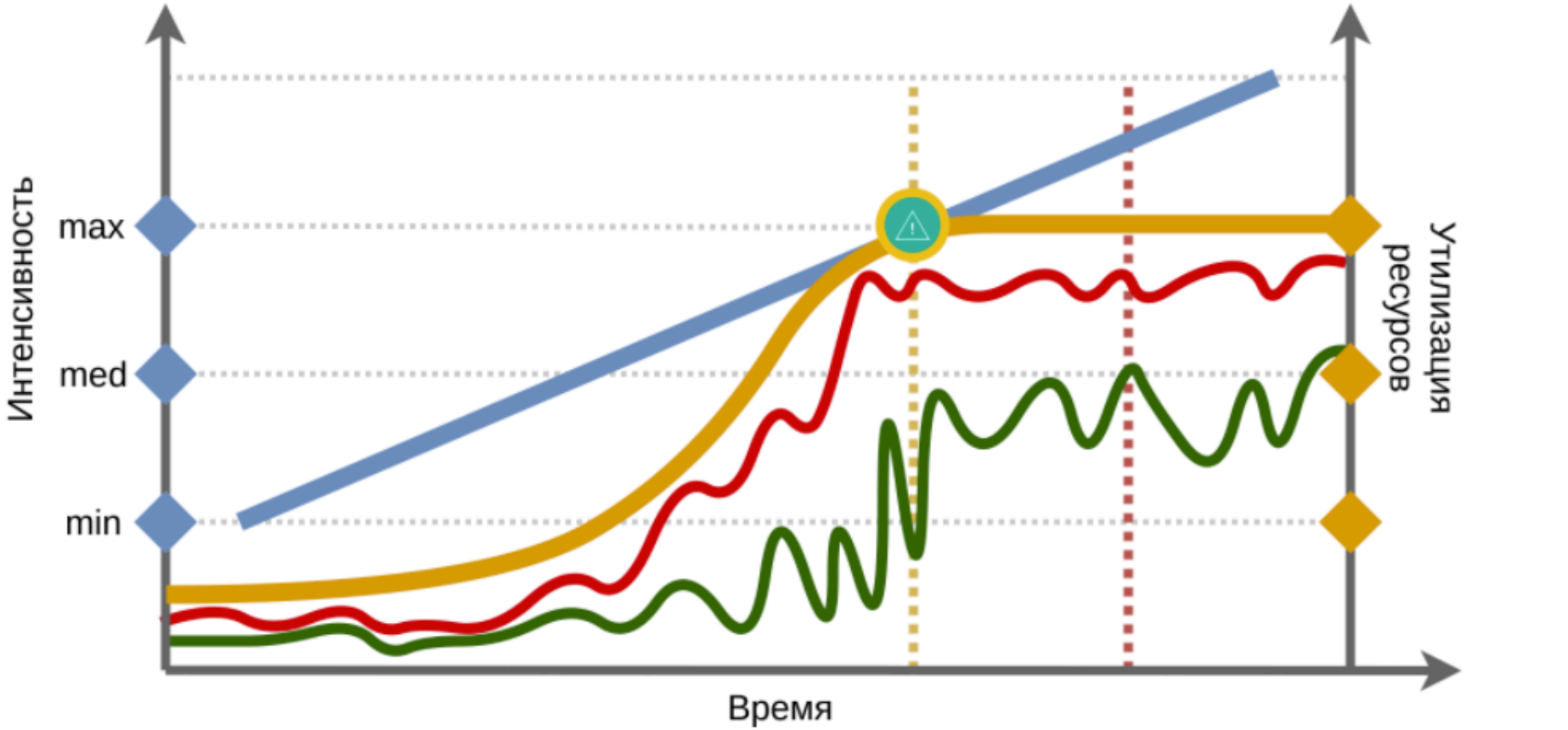
Нагрузочное тестирование



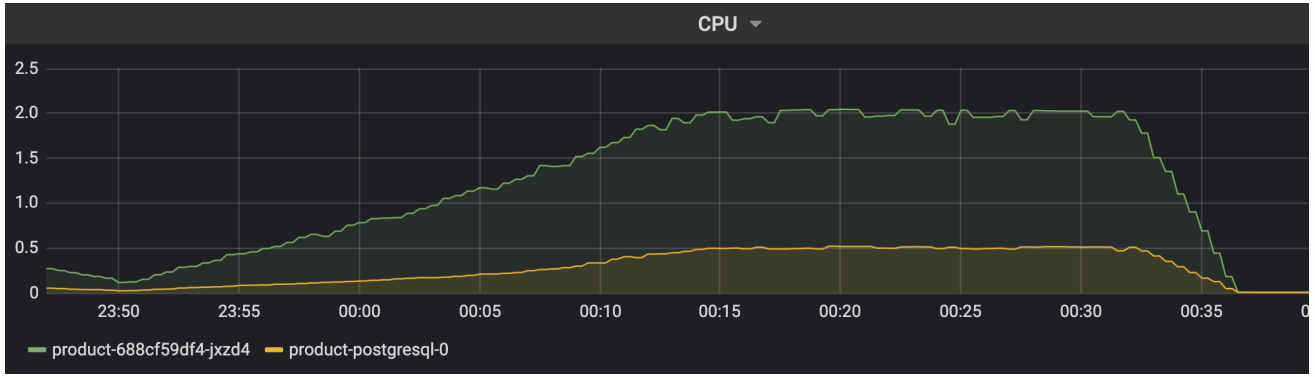


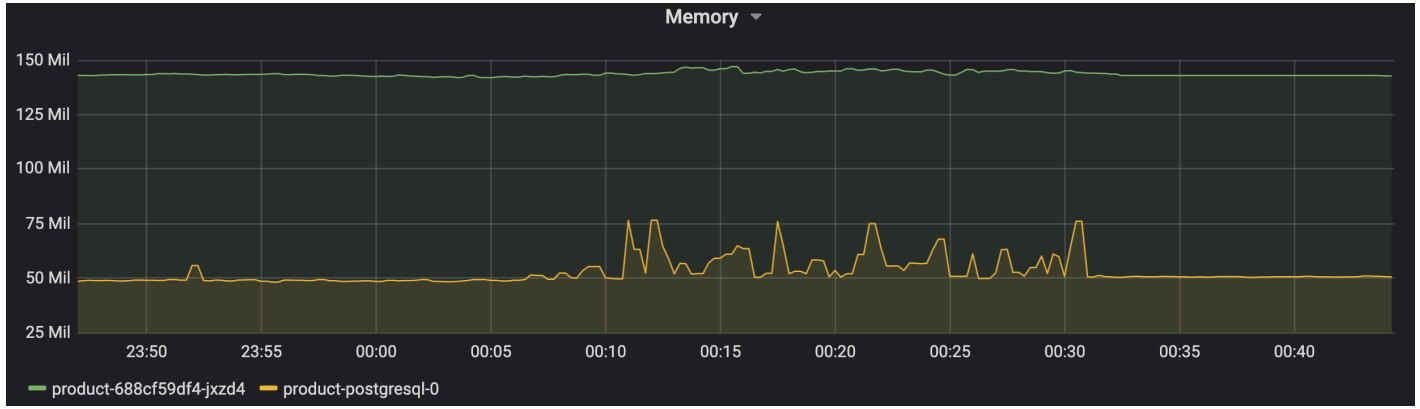
Нагрузочное тестирование

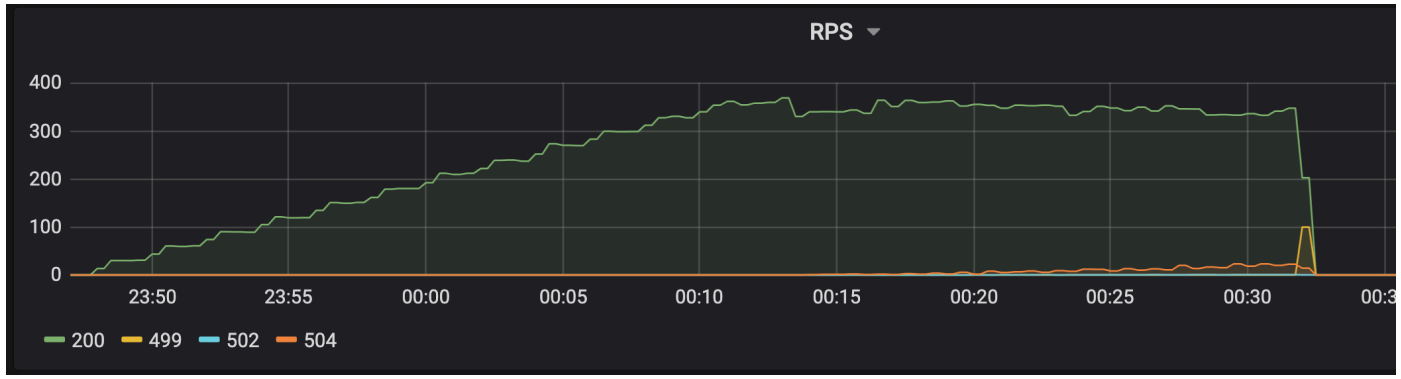
Ищем узкие места в утилизации ресурсов



Нагрузочное тестирование

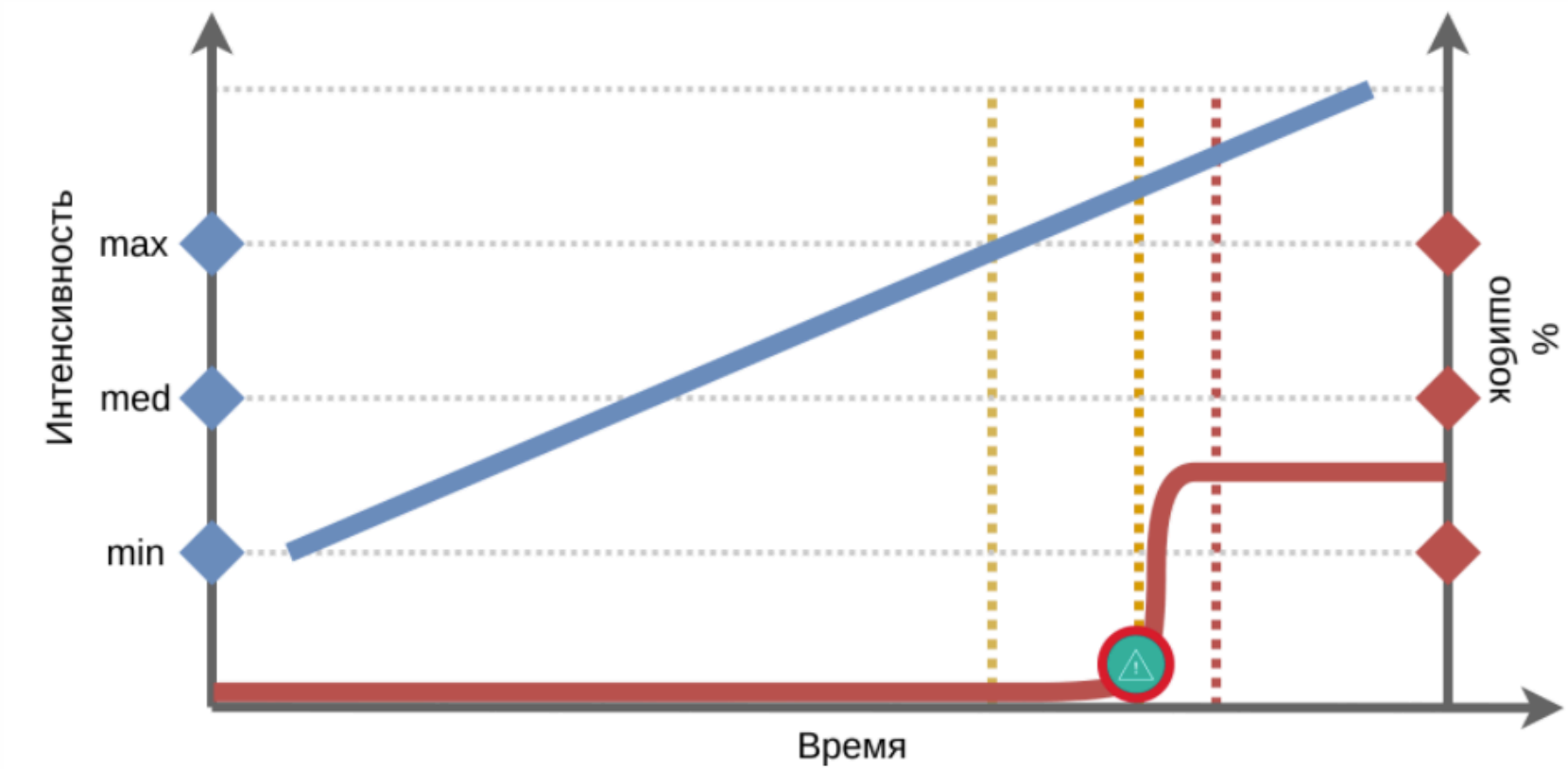




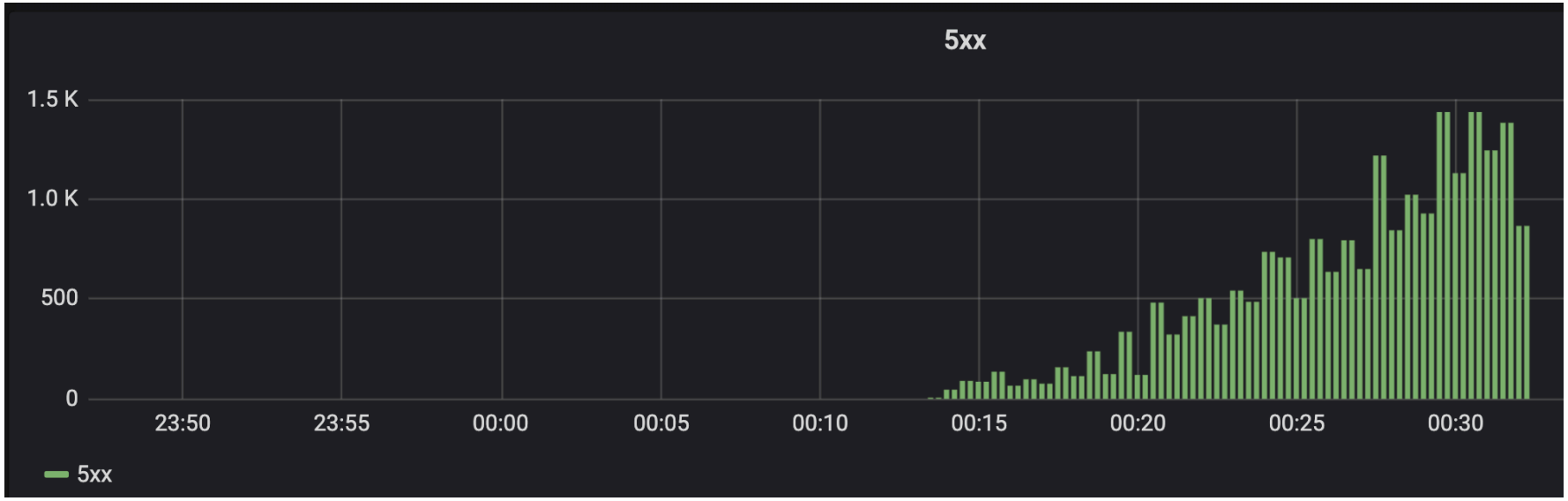


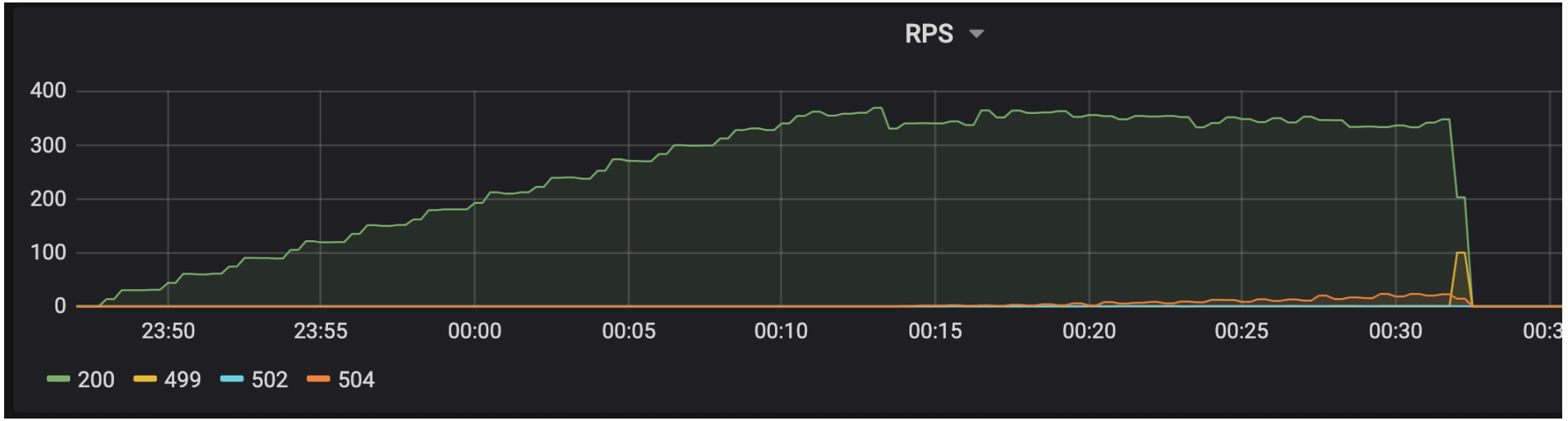
Нагрузочное тестирование

Ищем точку, когда появляются ошибки



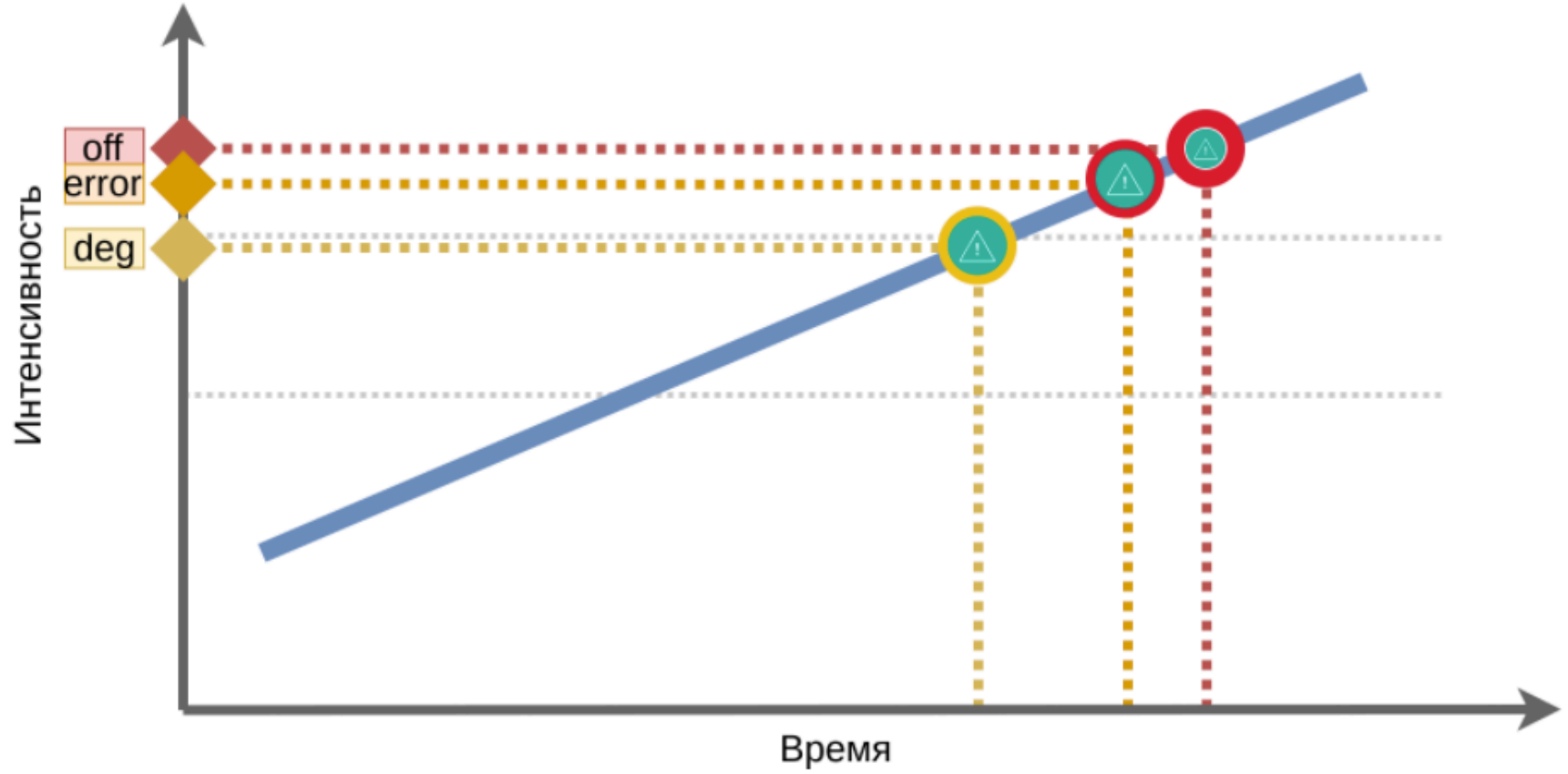
Нагрузочное тестирование





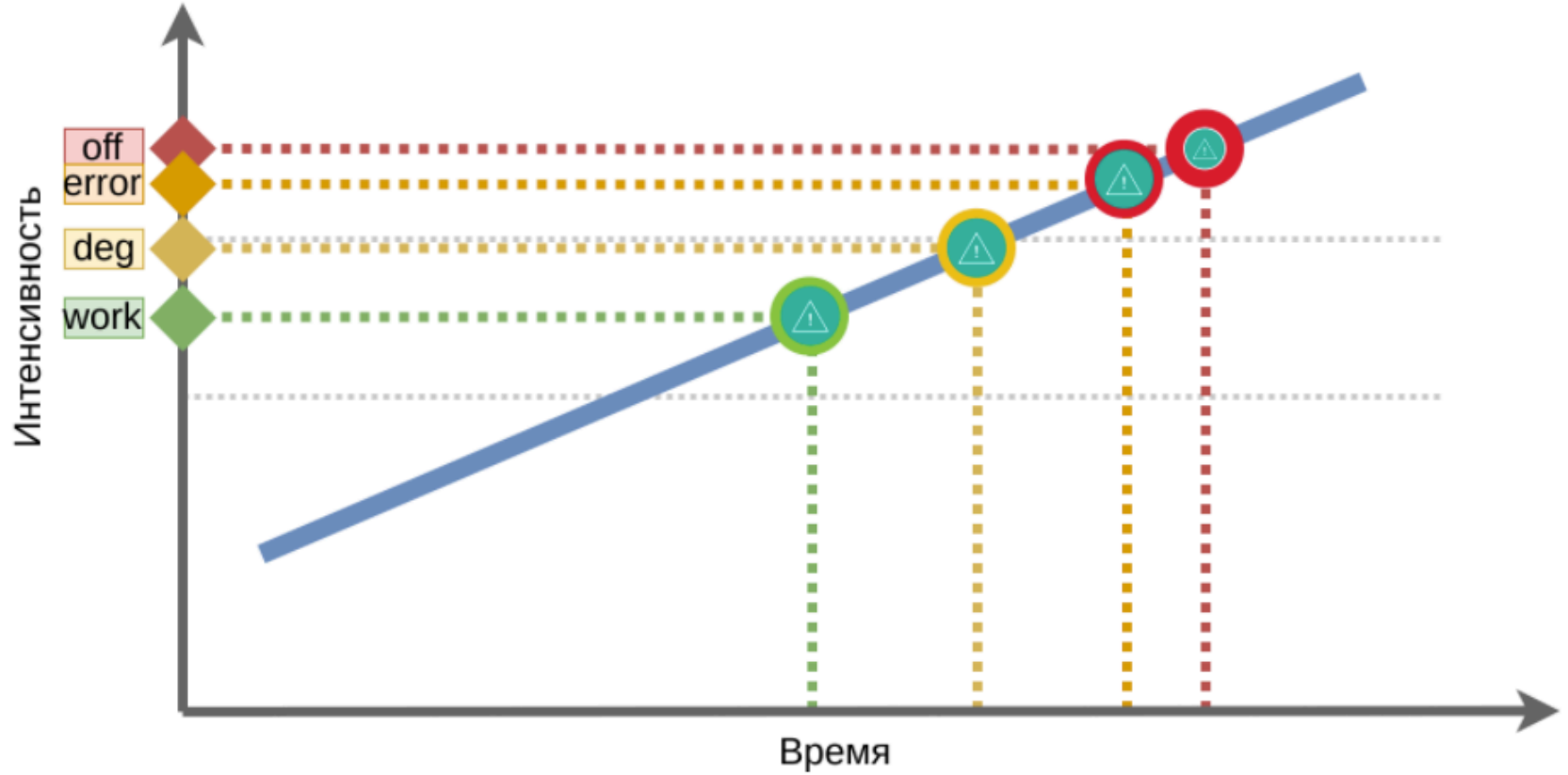
Нагрузочное тестирование

Смотрим, как соотносятся точки отказа, деградации, начала ошибок



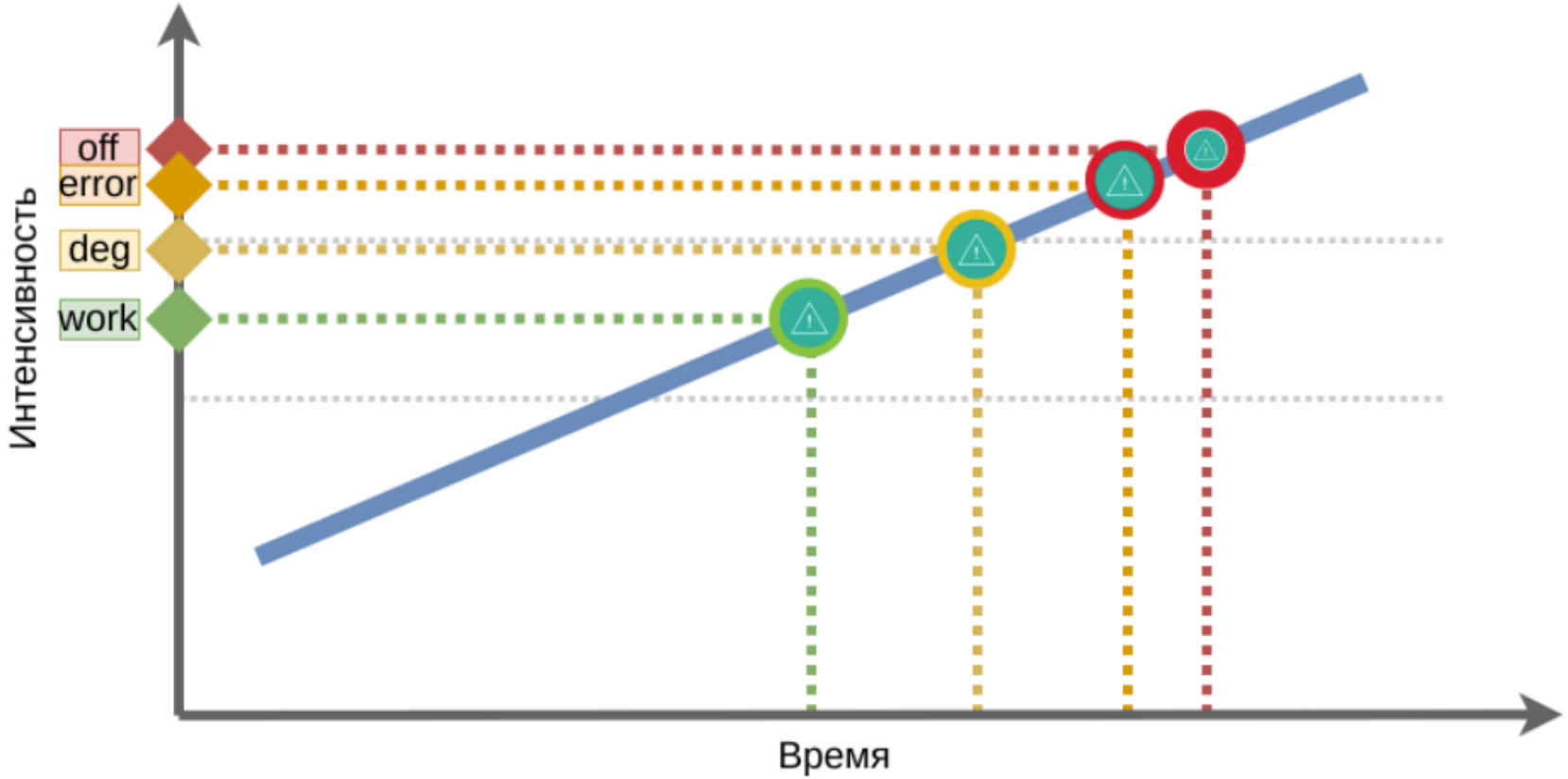
Нагрузочное тестирование

Выбираем точку рабочей нагрузки – обычного 80% от точки деградации



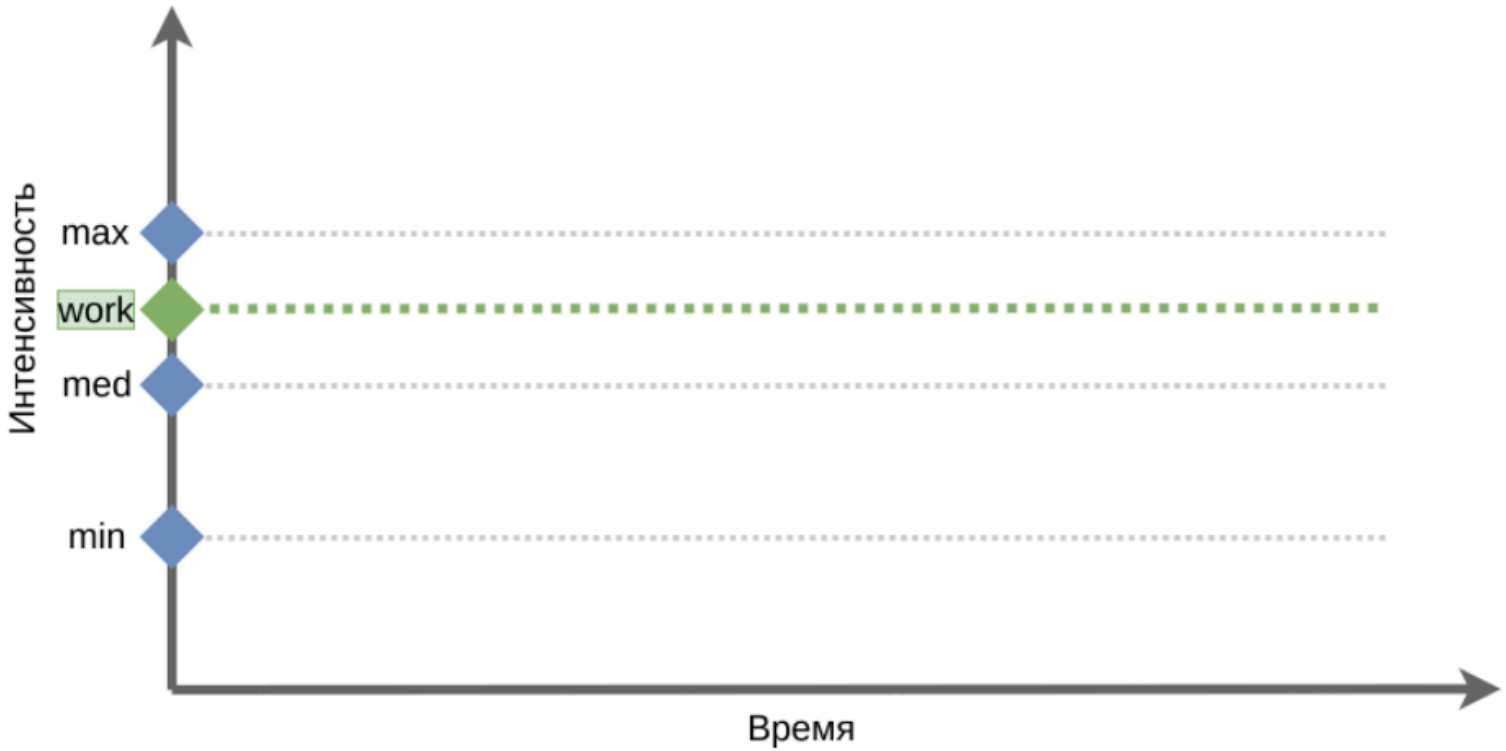
Нагрузочное тестирование

Насколько система стабильно себя ведет 24 x 7? Течет ли память и т.д?



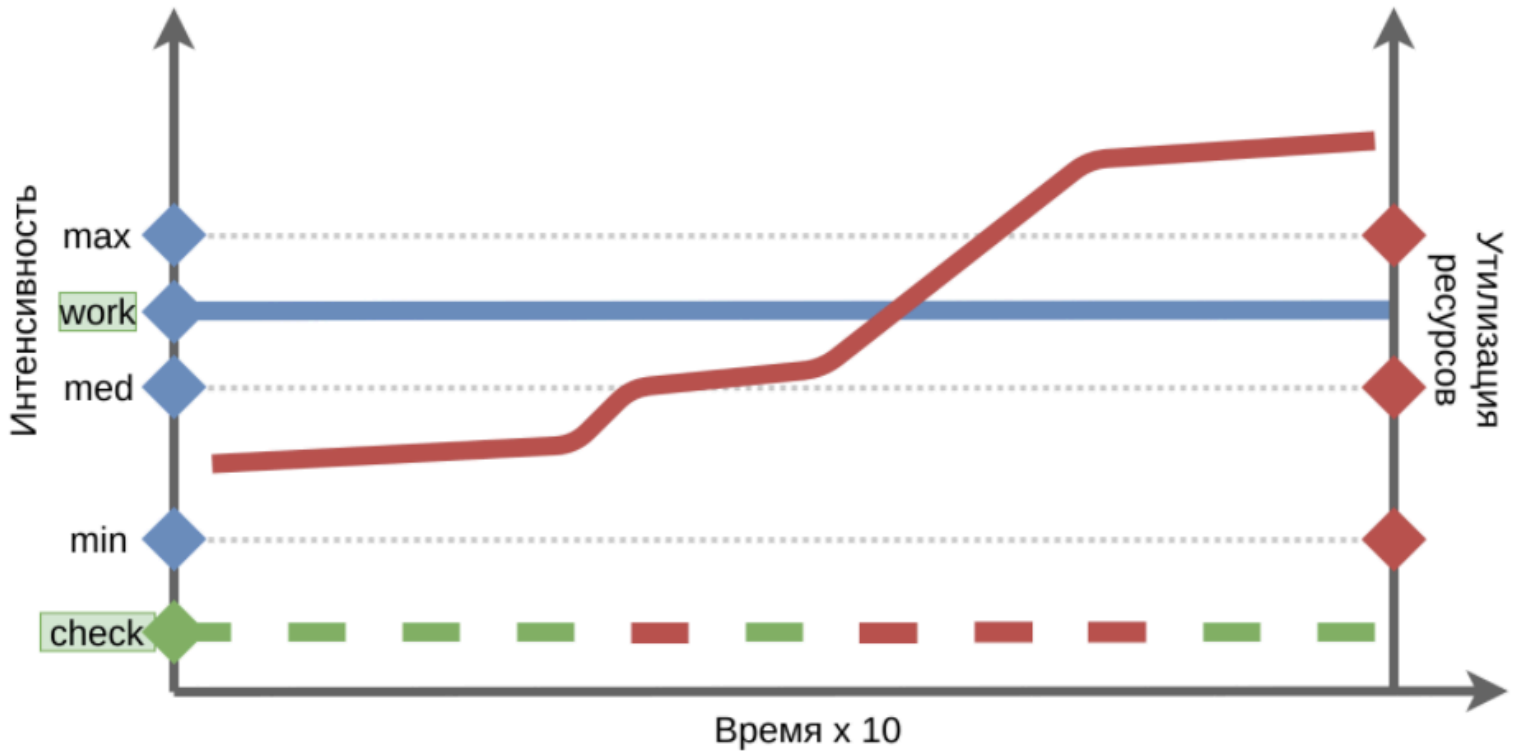
Нагрузочное тестирование

Подаем рабочую нагрузку, так чтобы не было ошибок.



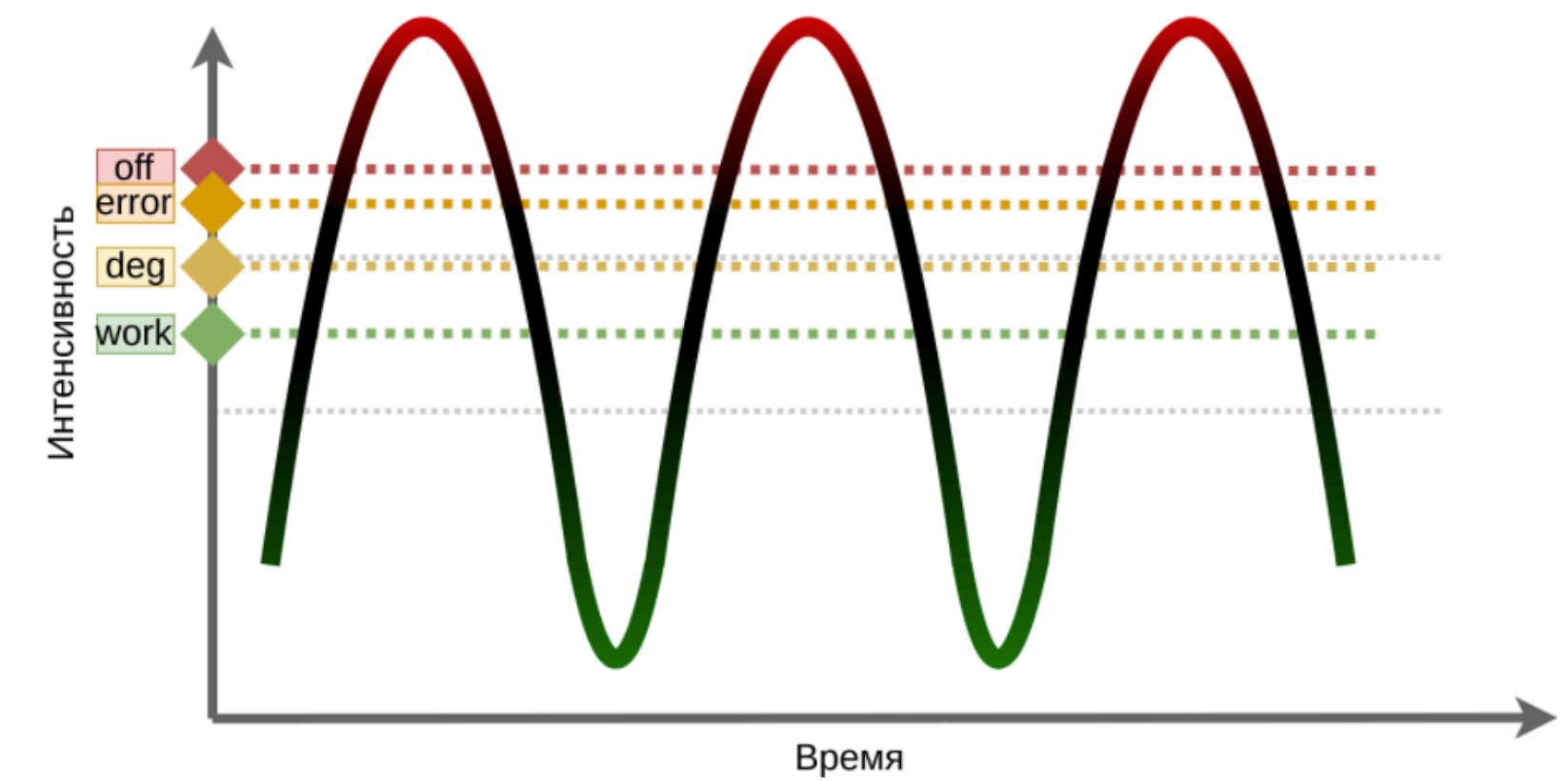
Нагрузочное тестирование

Подаем рабочую нагрузку, так чтобы не было ошибок.



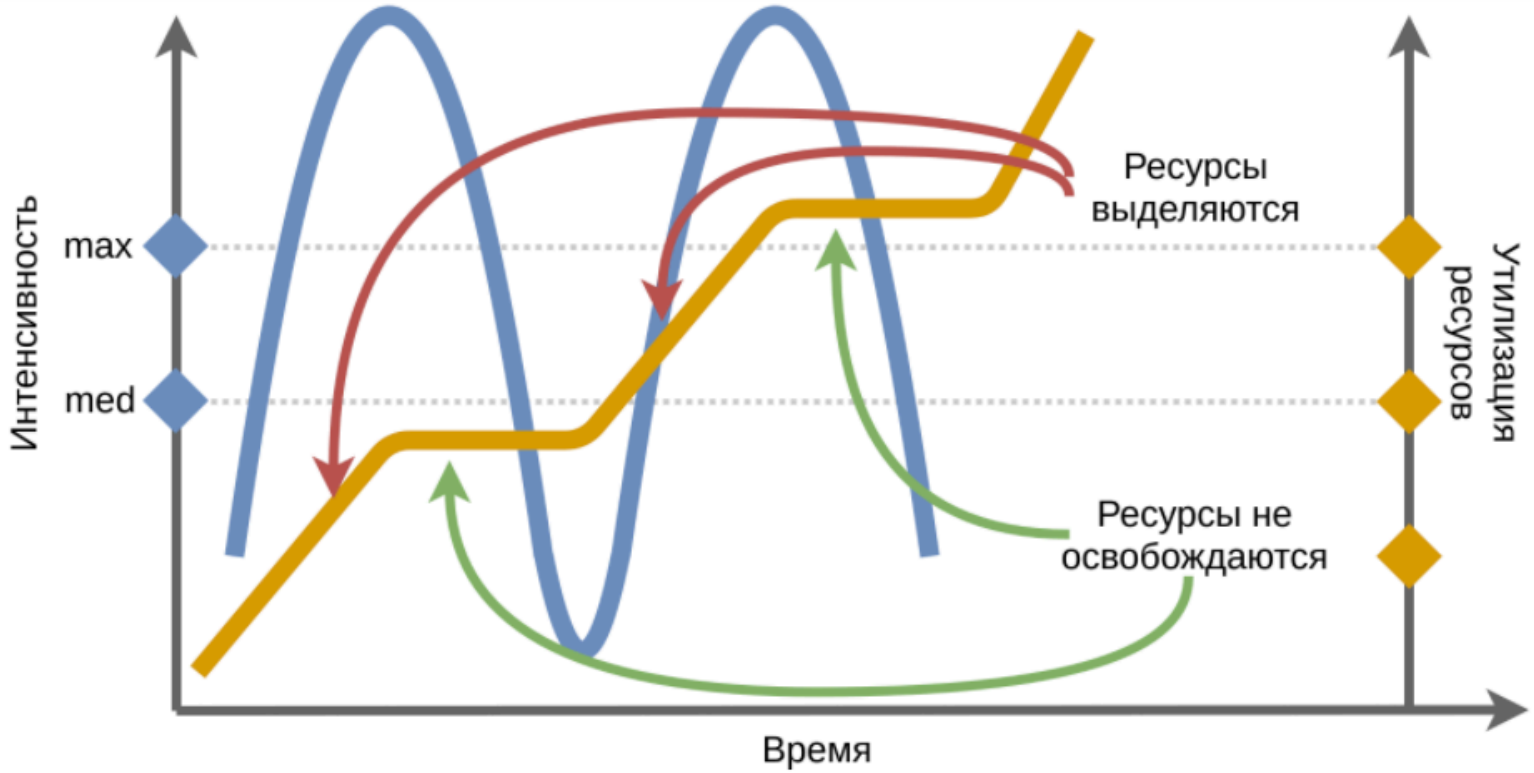
Нагрузочное тестирование

Стресс тестирование– подаем максимальную нагрузку и смотрим, восстанавливается сервис

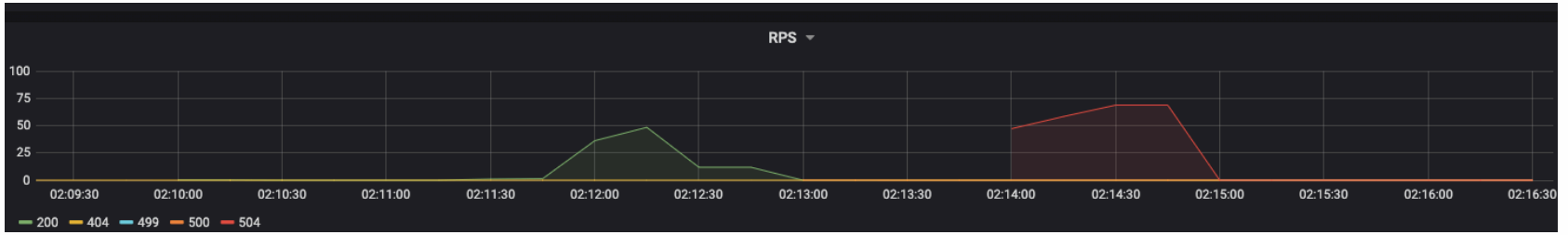


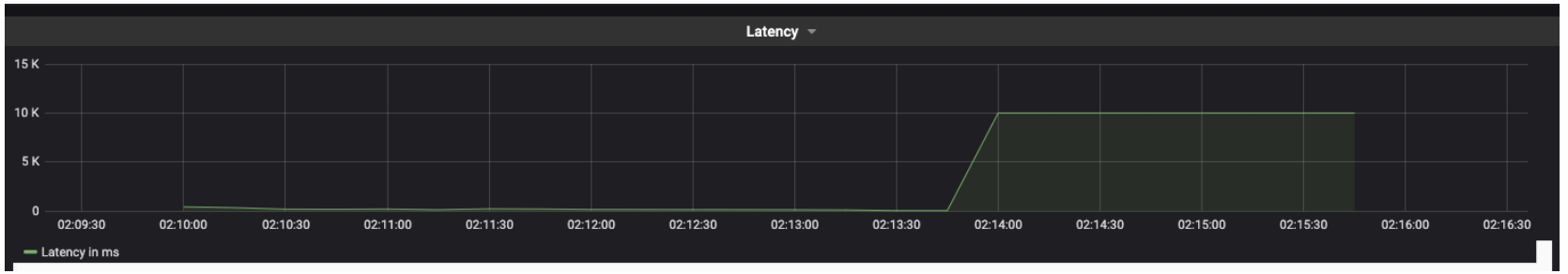
Нагрузочное тестирование

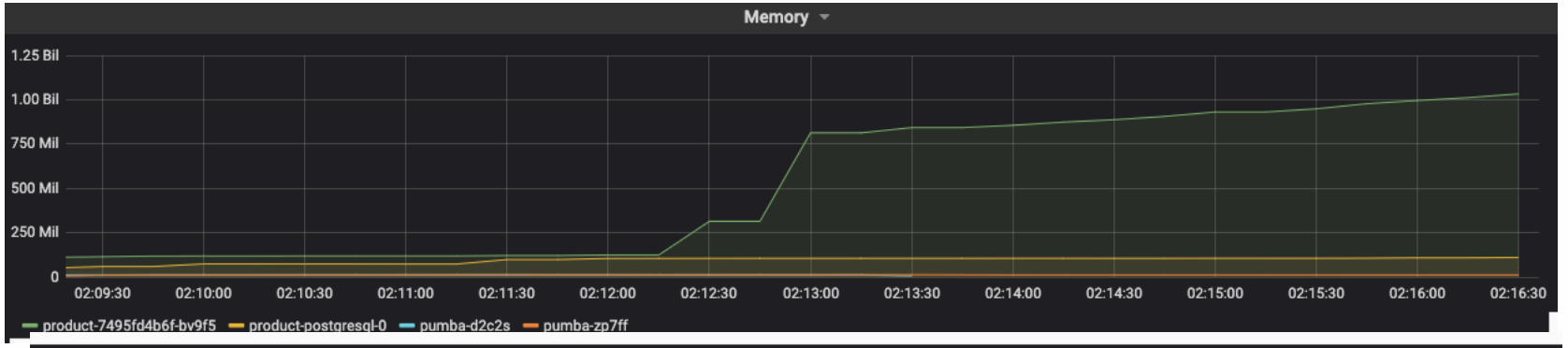
Важно следить за тем, что ресурсы освобождаются

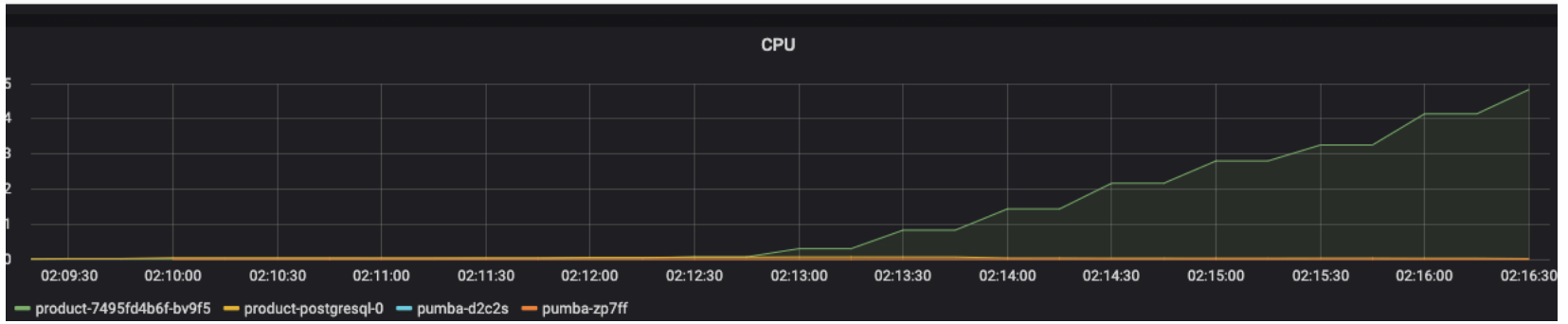


Нагрузочное тестирование



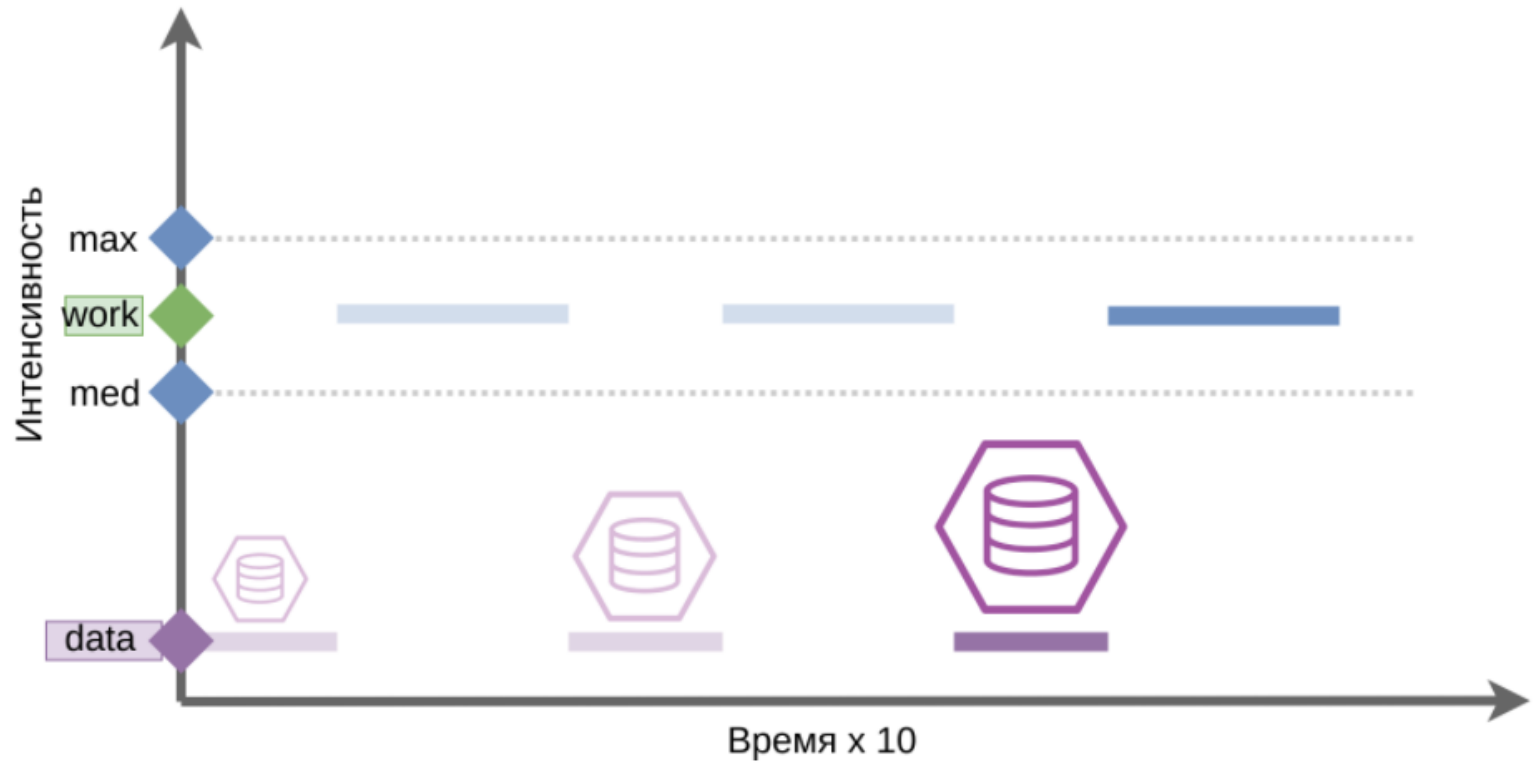






Нагрузочное тестирование

Объемное тестирование – как будет себя вести система, если количество данные увеличится. Не увеличиваем интенсивность – увеличиваем объем данных



Генерация данных

Генерация данных

- Генерация SQL файлов
- Хранимые процедуры
- Использовать python/Java/Ruby для генерации тестовых данных

Нагрузочное тестирование

Чем можно нагружать

- Простые запросы
- Сценарные запросы

Нагрузочное тестирование

Для микросервисных приложений, которые состоят не из одного или двух, а множества сервисов-компонент, нагрузочное тестирование отдельных компонент не всегда показательно.

Т.к. узких мест много, они распределены по сети, то для анализа лучше всего подходит сценарное тестирование, а не «долбилка» по статическим адресам

Нагрузочное тестирование

Для анализа нагрузочного тестирования зачастую не хватает просто отчета из инструмента нагрузочного тестирования. Тут очень сильно помогает мониторинг и алертинг всех компонент (сервисов), входящих в приложение.

Инструменты нагрузочного тестирования

Инструменты:

- Jmeter
- Gatling
- Yandex.Tank
- Tsung
- Locust

JMeter

JMeter – один из старейших инструментов нагрузочного тестирования

- Основные действия из UI и он сложный и развернутый
- Threaded модель (производительность может быть не очень хорошая)
- Довольно гибкий
- Написание сценариев на языках программирования возможно, но довольно таки тяжело

Locust

Locust – простой и современных инструмент для нагрузочного сценарного тестирования

- Сценарии пишутся на Python
- Достаточно производительный
- Есть UI, но он очень простой

02

Тестирование в продакшне

Нагрузочное тестирование

Нагрузочное тестирование – это всегда эксперимент в более или менее лабораторных условиях. Жизнь всегда богаче наших представлений о ней.

Ответить на вопрос «как будет вести себя то или иное приложение под нагрузкой» с помощью нагрузочного тестирования можно лишь приблизительно (с теми или иными ограничениями)

Тестирование в продакшне

Поддерживать prodlike среду дорого. Особенно, если мы говорим про микросервисную архитектуру, ее поддерживать в раз дороже и сложнее, чем монолитную за счет динамичности и сложности связей.

Тестирование в продакшне

Мы можем проводить эксперименты не только в лабораторных условиях и на наживую. Можем проверять, как будет вести себя наше приложение в случае отказов или повышения нагрузки в продакшн среде.

Пассивная и активная надежность

Пассивная надежность:

- Инерционность и консервативность
- Сначала долго думаем, потом принимаем решение
- Изменения – это плохо

Активная надежность:

- Быстрая реакция на изменения
- Маневренность
- Стойкость к неожиданным изменениям
- Изменения – это нормально

https://static.sched.com/hosted_files/leankanbanru2014/da/%D0%9D%D0%B5%20%D1%81%D1%82%D0%BE%D0%B8%D1%82%20%D0%B1%D0%BE%D1%80%D0%BE%D1%82%D1%8C%D1%81%D1%8F%20%D1%81%20%D0%BD%D0%B5%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C%D1%8E%20-%20%D0%BF%D0%B5%D1%80%D0%B5%D0%BE%D1%81%D0%BC%D1%8B%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%BF%D0%BE%D0%B4%D1%85%D0%BE%D0%B4%D0%B0%20%D0%BA%20%D1%80%D0%B8%D1%81%D0%BA-%D0%BC%D0%B5%D0%BD%D0%B5%D0%B4%D0%B6%D0%BC%D0%B5%D0%BD%D1%82%D1%83.pdf

Тестирование в продакшне

Максимально проверить код и инфраструктуру перед выкаткой релиза.

- Тяжелые, долгие и редкие релизы
- Prodlike стейдж, qa стенды
- Предрелизное тестирование, чаще всего руками

Тестирование в продакшне

Подготовить инфраструктуру и приложение к постоянным изменениям и возможности быстрого реагирования на проблемы

- Частые релизы
- Отсутствие или «легкие» стенды
- Максимально развитые средства мониторинга и алертинга
- CI/CD
- Микросервисная архитектура
- Graceful degradation

Тестирование в продакшне

Лучше всего – баланс. Лучше иметь и средства тестирования перед выкаткой на продакшн, и иметь возможность проводить безопасные эксперименты.

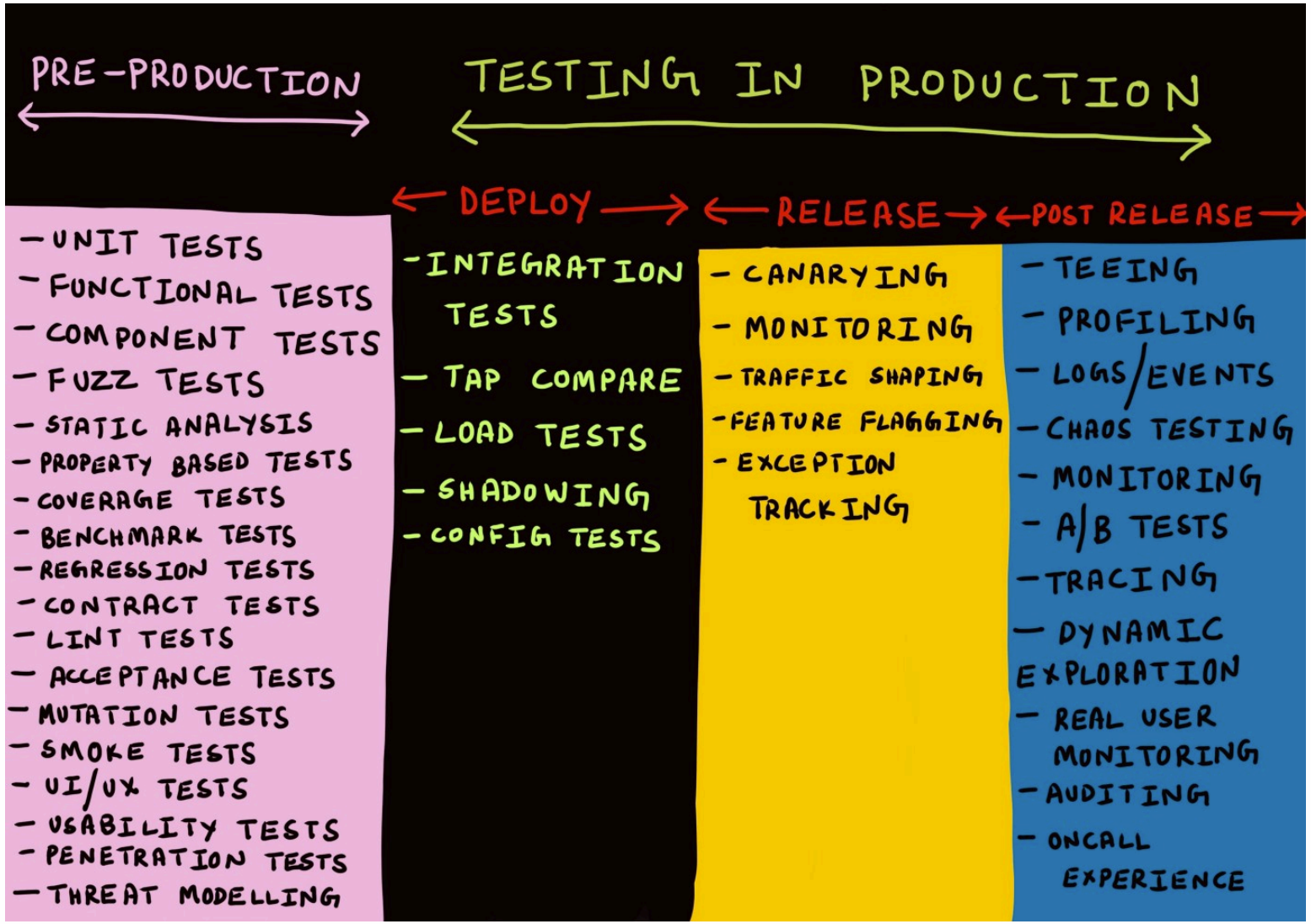
- Все, что дешевле проверить до выкатки на прод: критичные релизы, изменения в инфре – лучше проверить.

- Все, что можно выкатить безопасно на продакшн и посмотреть, как оно работает – лучше выкатить на продакшн.

Тестирование на продакшне

https://medium.com/@copyconstruct/testing-in-production-the-safe-way-18ca102d0ef1

Chaos Engeneering



https://medium.com/@copyconstruct/testing-in-production-the-safe-way-18ca102d0ef1

Chaos Engeneering

Типовые атаки:

- Удаление подов, контейнеров, убивание процессов
- Удаление нод
- Изменение времени
- Сеть: задержки, потери, порча
- Ресурсы: загружаем на 100% CPU, память, IO и т.д.

http://principlesofchaos.org/?lang=ENcontent

Тестирование на продакшне

- Выявление узких мест в рамках контролируемого теста
- Возможность осознанного планирования по улучшению системы

Инструменты Chaos Engeneering

На текущий момент все в состоянии альфы или беты.
Активно развиваются инструменты под Kubernetes.

- Gremlin (https://www.gremlin.com/)
- ChaosBlade (https://github.com/chaosblade-io/chaosblade-operator)
- ChaosToolkit (https://github.com/chaostoolkit/chaostoolkit-kubernetes)
- Pumba (https://github.com/alexei-led/pumba)

Pumba

Один из самых простых и удобных инструментов.

Включает в себя возможность привносить хаос в сеть: сетевые потери, задержки и т.д. и т.п.

https://github.com/alexei-led/pumba

Спасибо
за внимание!