avito.tech

Москва — 2022

# Как экономить железо для машинлёрнинга

Олег Бугримов

Тимлид в Data Science SWAT

# План доклада

01. Что такое “модель в проде”
02. Постановка проблемы: сервис медленно отвечает
03. Решение проблемы: вынос модели в подпроцесс
04 С чем столкнемся и как решать
05. Ньюансы выкатки моделей в прод
06. Все решения в одной библиотеке

# 50
## Сервисов

Машин-лёрнинг,
нейросетки,
датасаентс
вот это все

# 24
## DS инженеров

Обучают постоянно
модели, все время,
без остановки

# 7
## Python инженера

Помогаем с питоном,
инженерией, паас
будь он неладен. А
еще следим за GPU

# Цели разработки

*01.* Сделать **легким** создание и запуск ML-сервисов

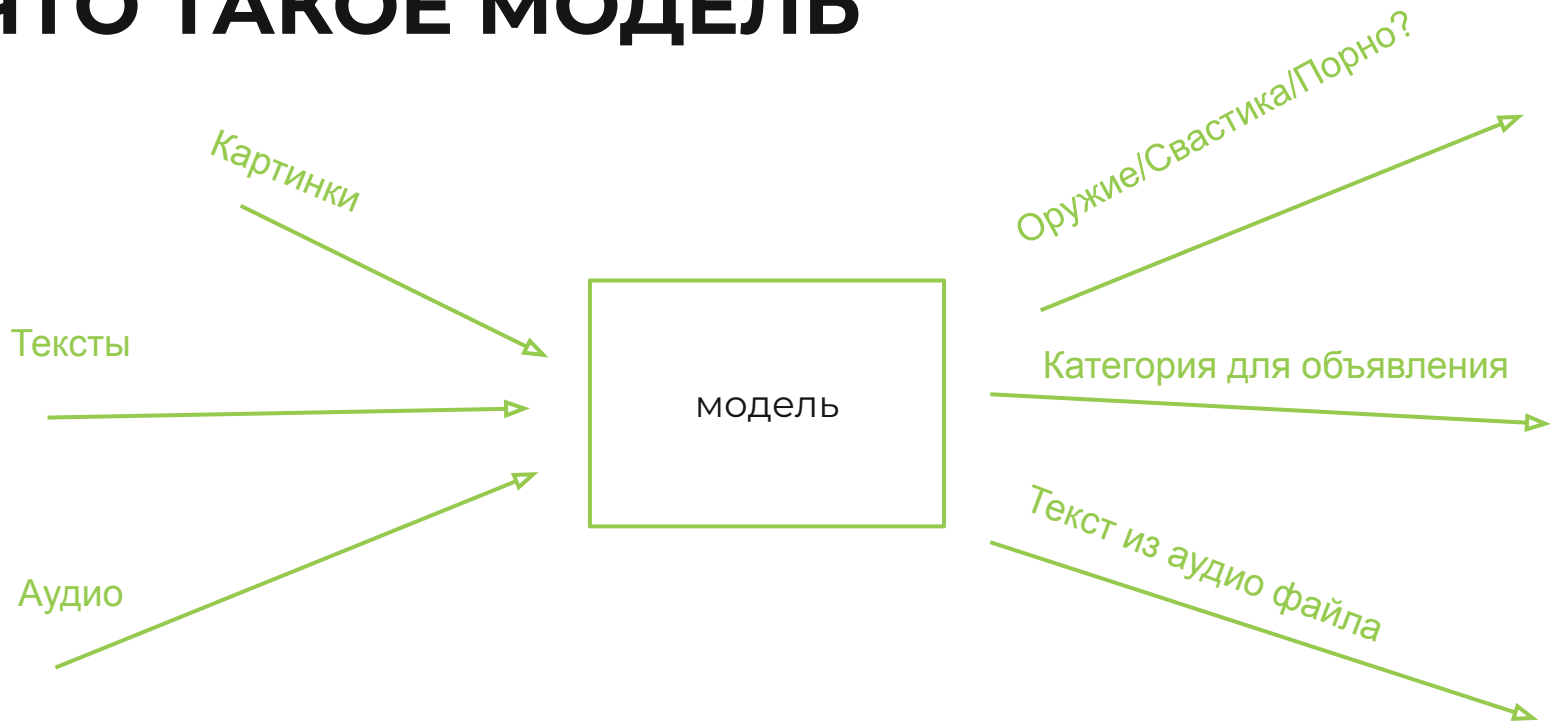
*02.* Сделать **одинаковым** создание и запуск ML-сервисов

*03.* Сделать **правильным** создание и запуск ML-сервисов

## Правильный - это как?

- Держим ровный высокий  RPS
- Экономим ядра CPU
- Экономим память/ядра GPU

# ЧТО ТАКОЕ МОДЕЛЬ

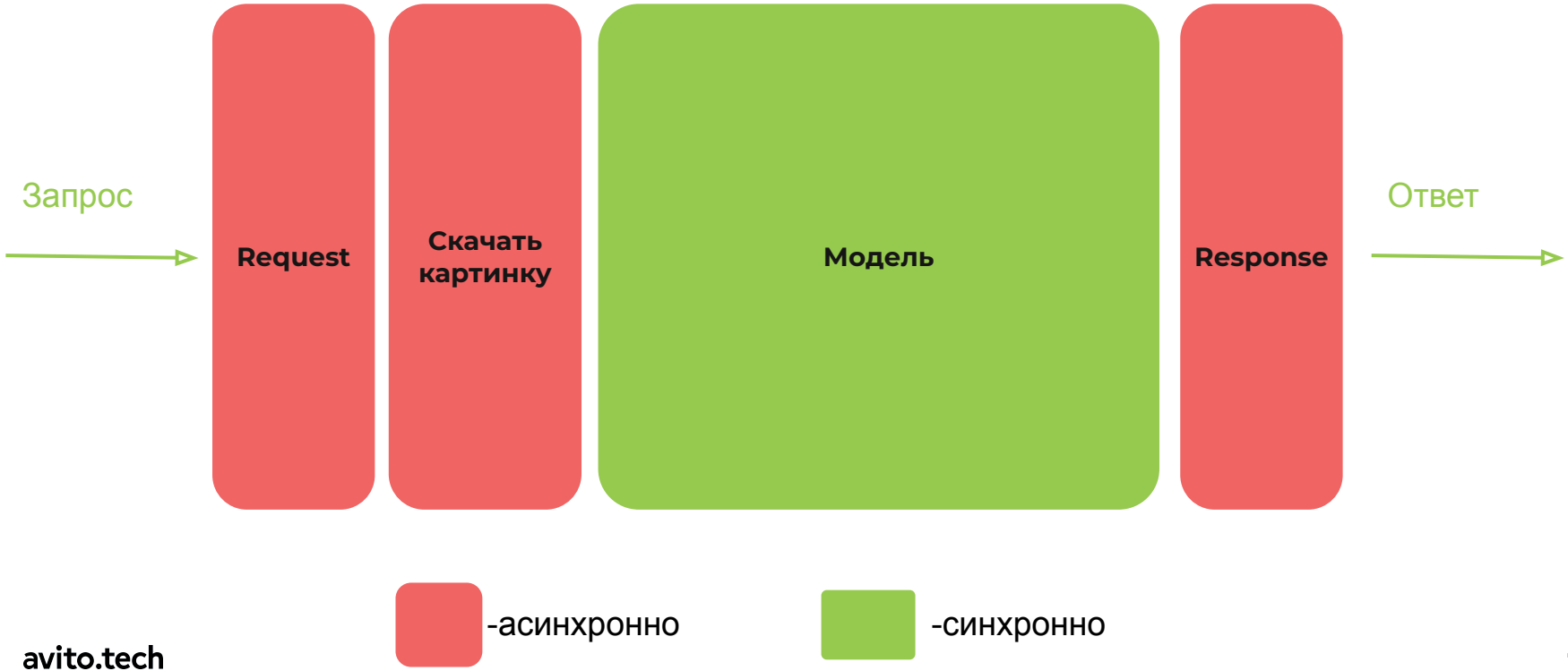


# А В КОДЕ?

Как выглядит вызов модели в python

```python
class MyModelHandler(BaseHandler):

    async def post(self):
        request_data = await self.request_json()

        result = self.app['model'].process(request_data['my_input_dat

        return self.response({
            'result': result.get_dict()
        })
```

# НУ ВОТ МЫ ВСТАВИЛИ МОДЕЛЬ В СЕРВИС

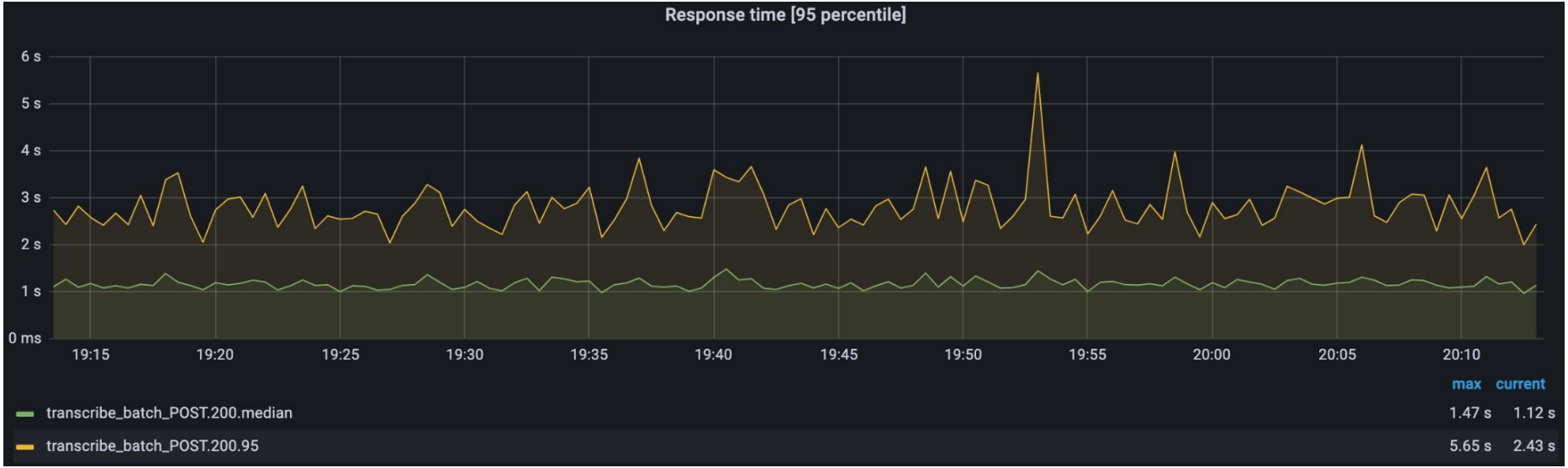


# ПРОСТОЕ РЕШЕНИЕ

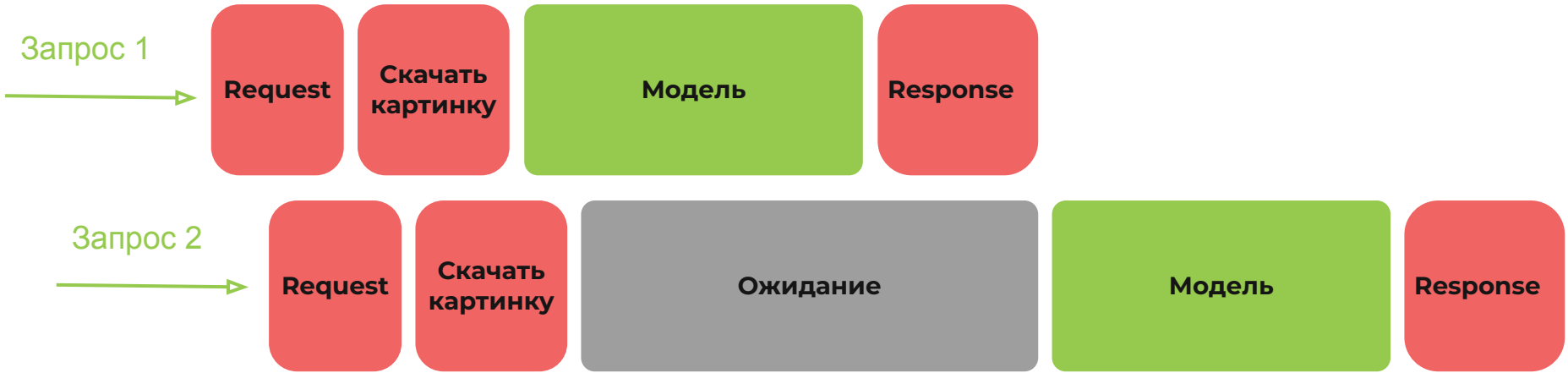
Как выглядит http-хендлер в avio

```python
class MyModelHandler(BaseHandler):

    async def post(self):
        request_data = await self.request_json()

        result = self.app['model'].process(request_data['my_input_data'])

        return self.response({
            'result': result.get_dict()
        })
```

# ПОЛУЧАЕМ БОЛЬШОЙ 95 %%

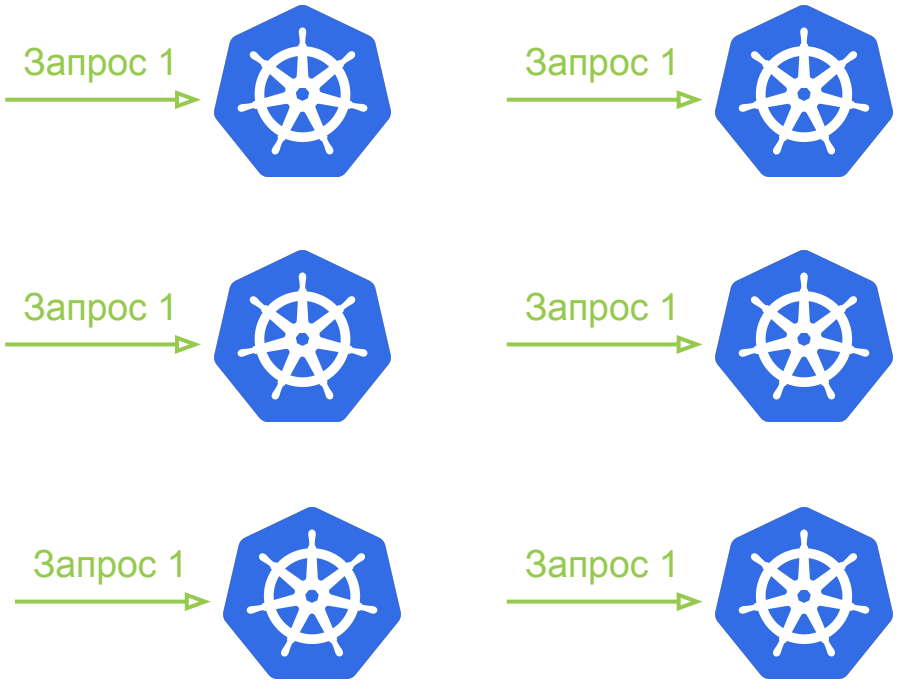


# Проблема: битва за ЦПУ

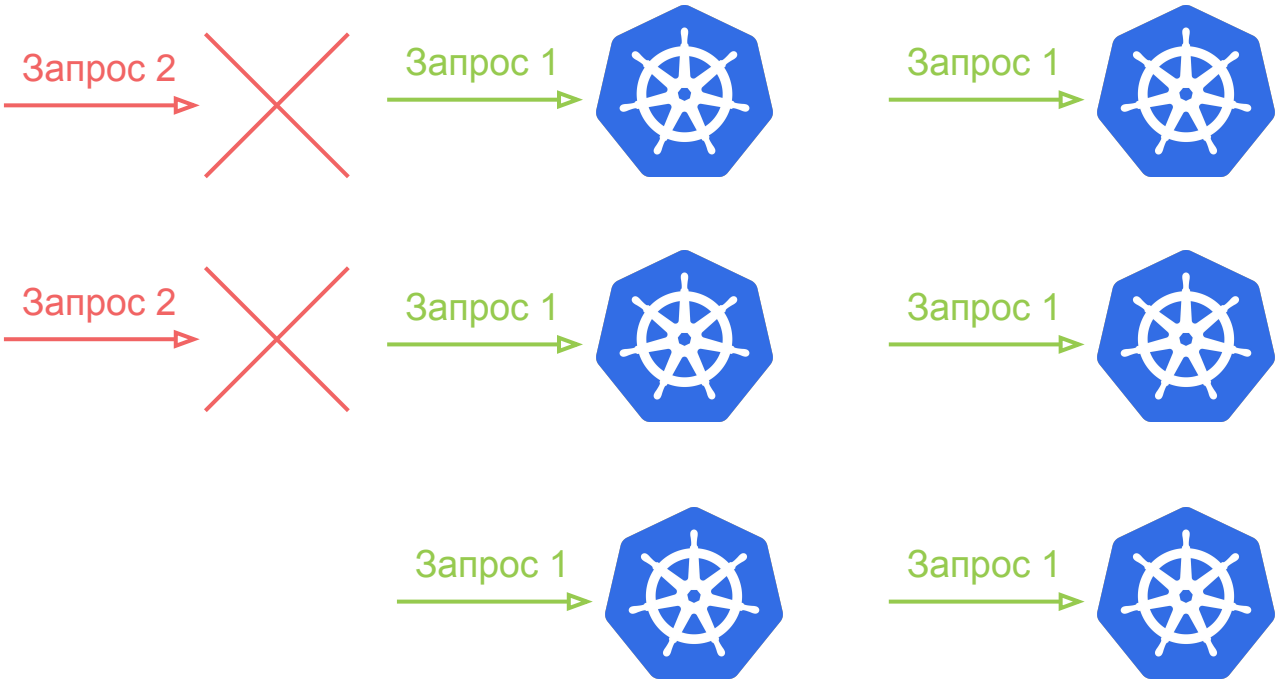


# Проблема: битва за ЦПУ

# РЕШЕНИЕ: ЗАКИДАТЬ ЖЕЛЕЗОМ



# ЗАКИДАТЬ ЖЕЛЕЗОМ



# ЗАКИДАТЬ ЖЕЛЕЗОМ

GPU: их мало, они дорогие

# План решения без перерасхода железа

01. Выносим модель в подпроцесс

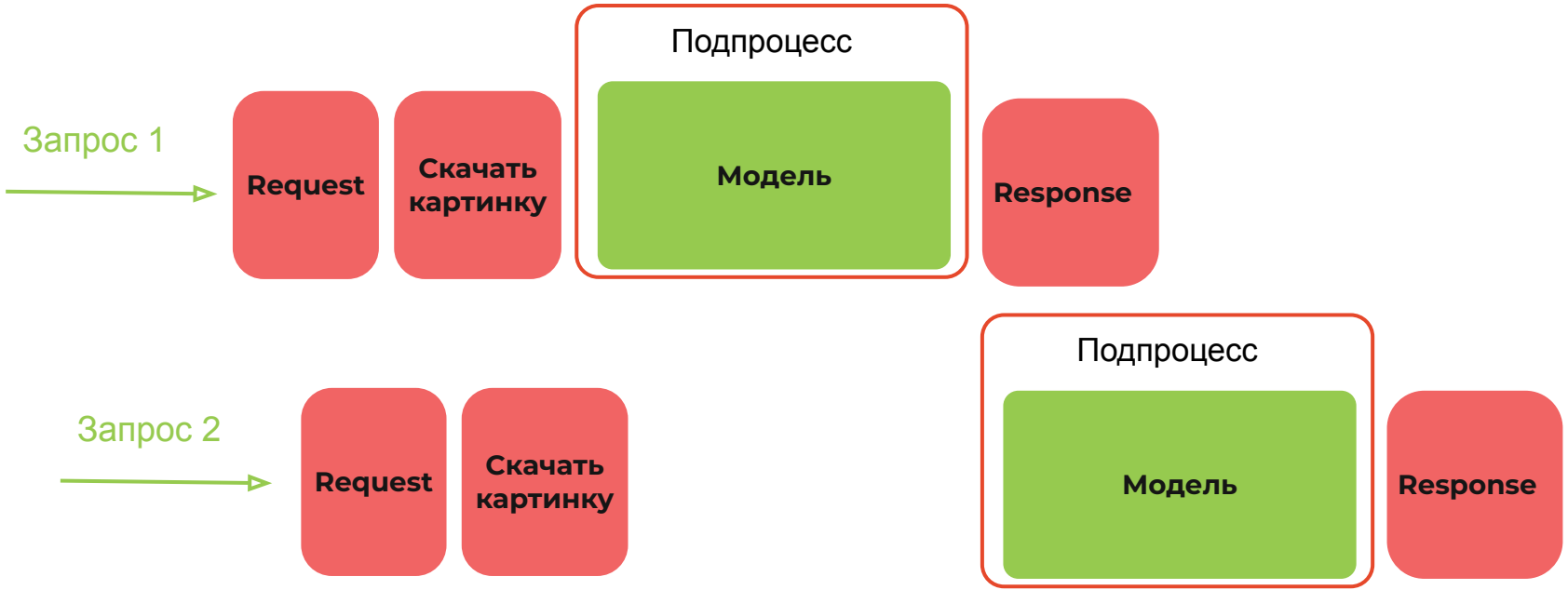
02. Разбиваем процесс на этапы и подпроцессы

03. Масштабируем подпроцессы для выравнивания нагрузки

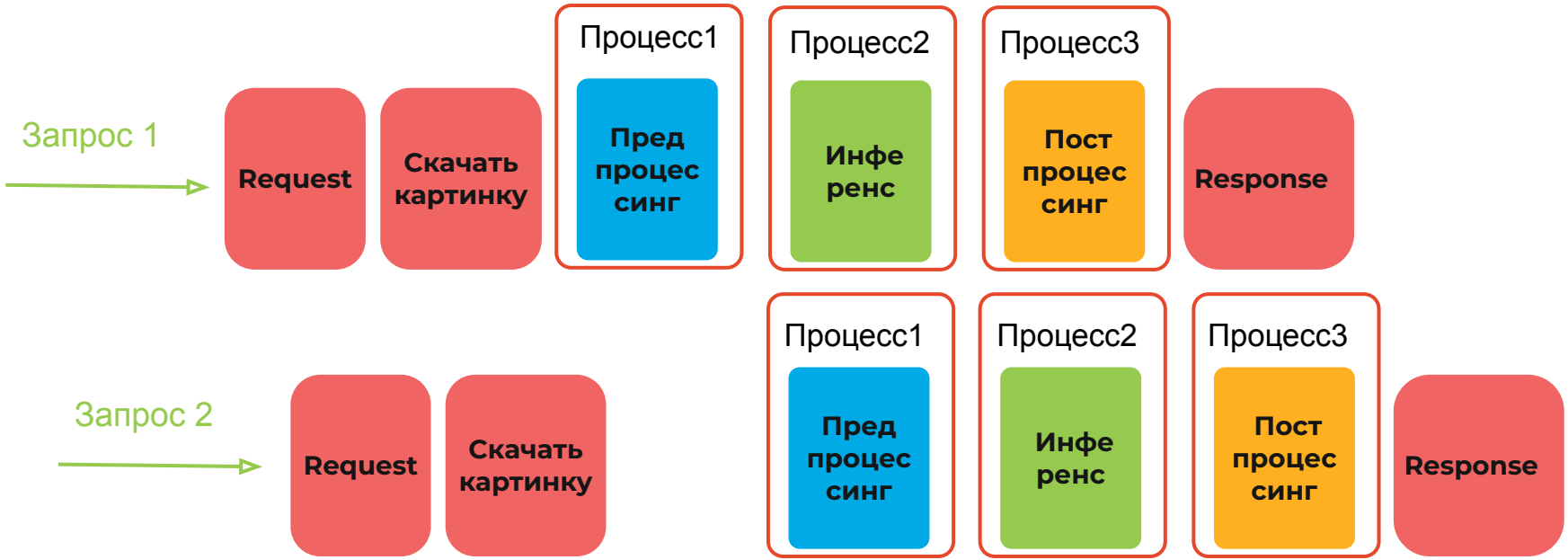
04 Используем батчирование

05 Используем шаренную память

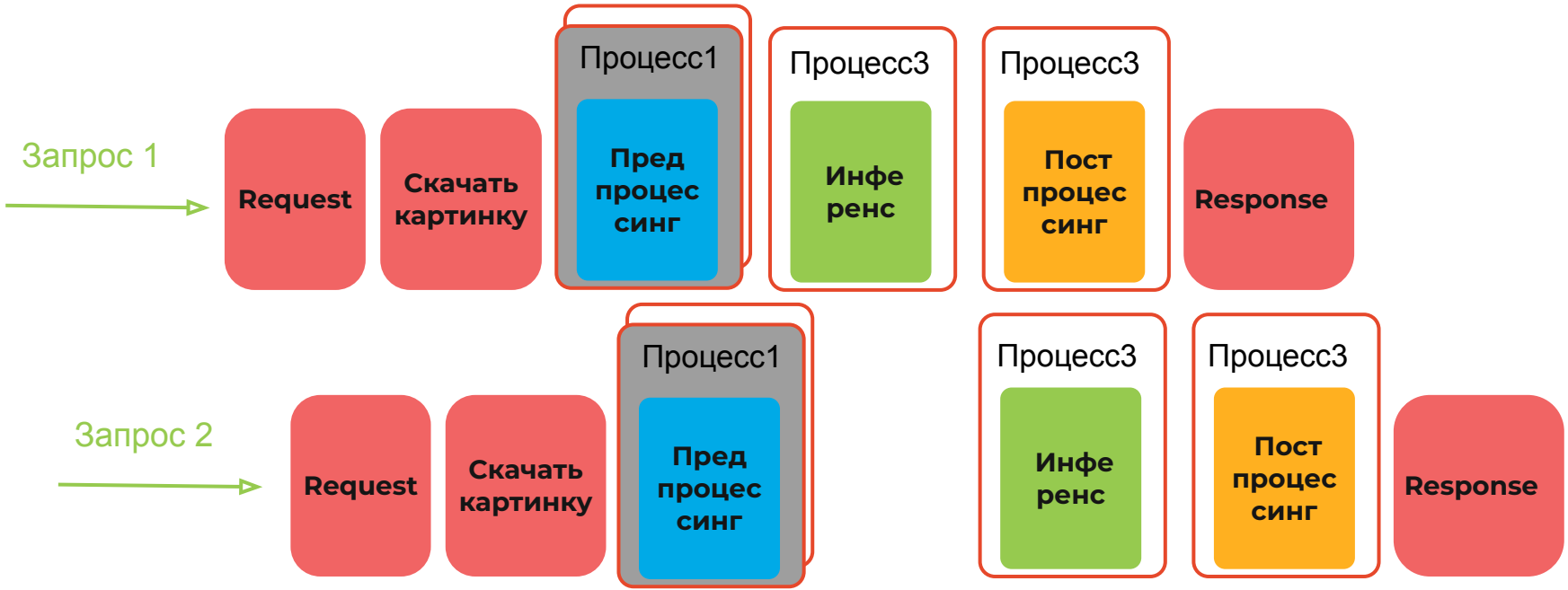
# Выносим модель в подпроцесс



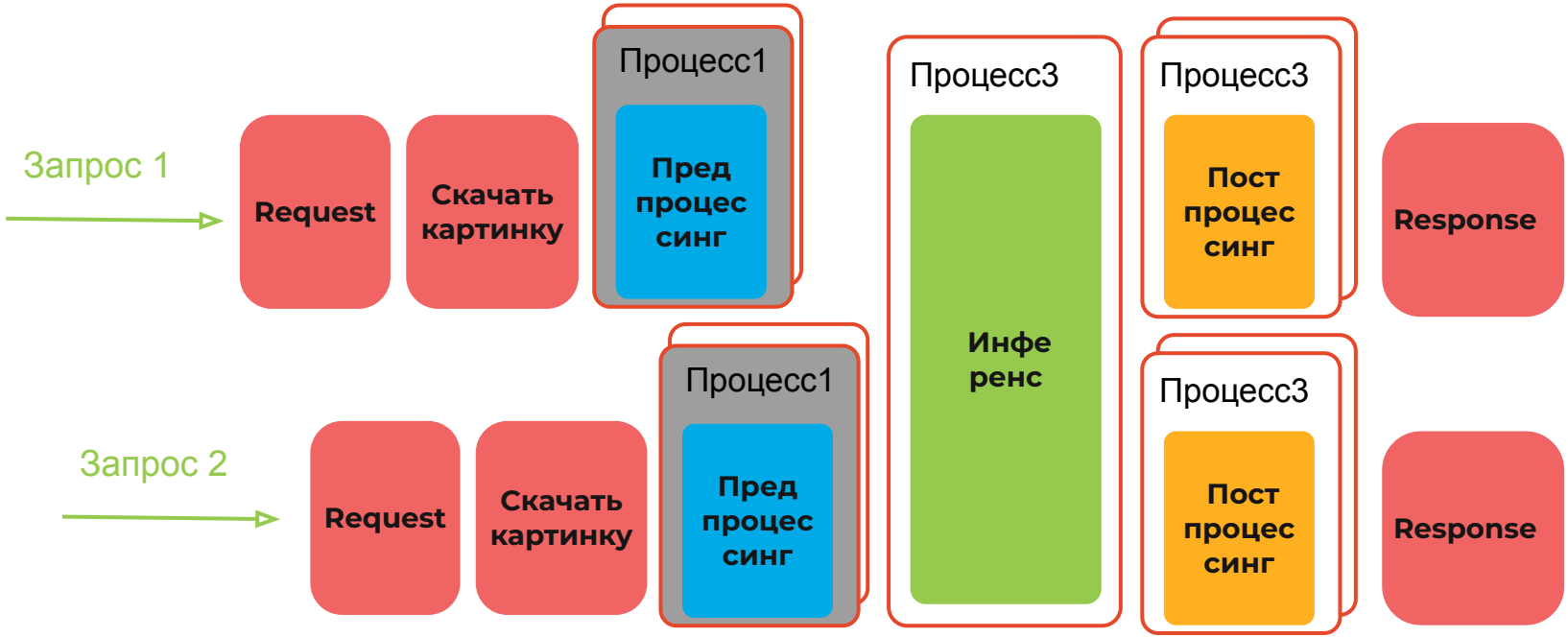
# Разбиваем процесс на этапы и подпроцессы



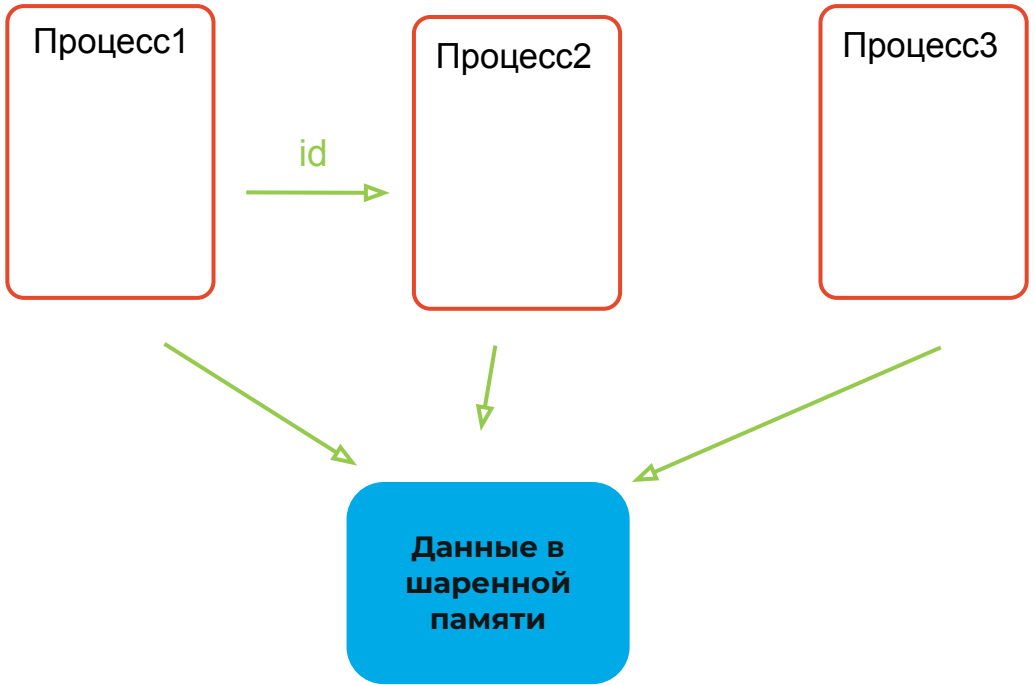
# Масштабируем подпроцессы для выравнивания нагрузки



# Используем батчирование



# МОЖНО ПЕРЕДАВАТЬ ЧЕРЕЗ ШАРЕННУЮ ПАМЯТЬ

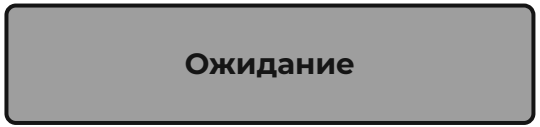


# ШАРЕННАЯ ПАМЯТЬ - ЭТО БЫСТРО

# ИТОГО В БАТОНАХ:

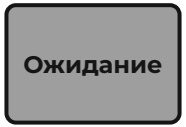
Модель в основном процессе:


Модель в дочернем процессе:




Каждый шаг в отдельном процессе:


Еще и батчирование:



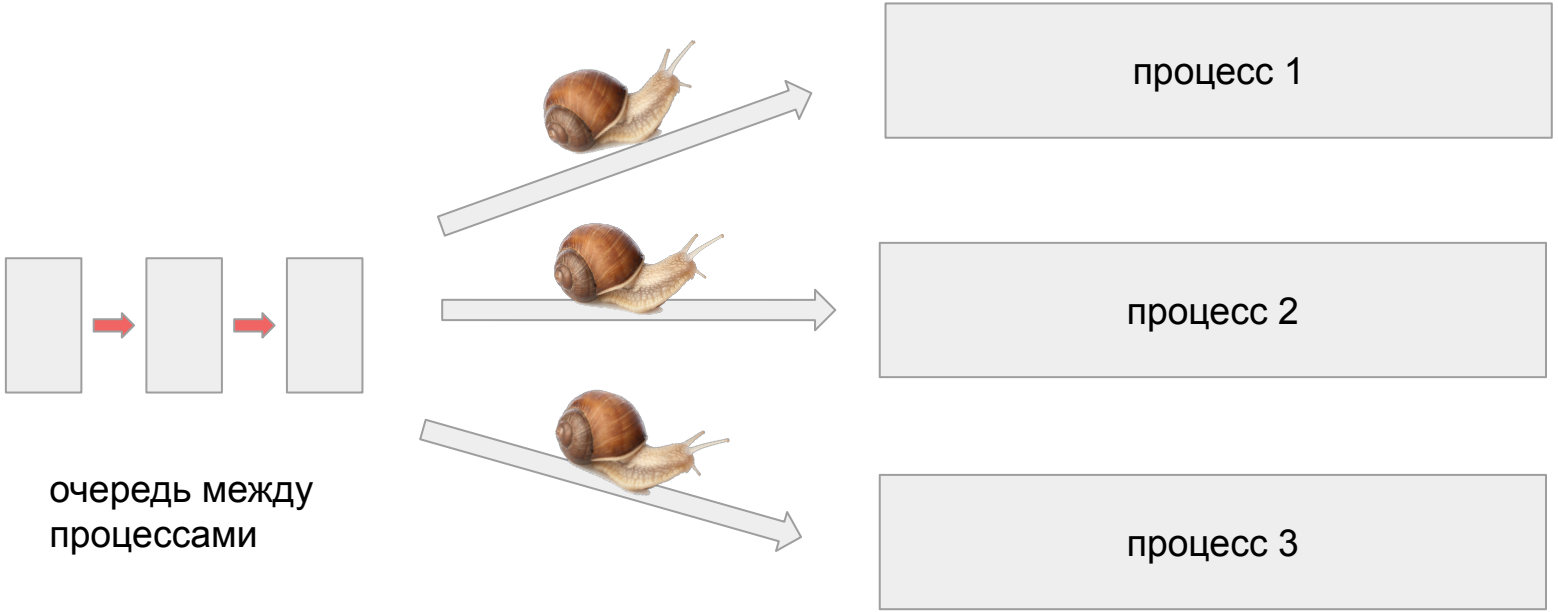
# С чем столкнемся при решении этих проблем

01. Баг питона при работе с очередями

02. Счетчики ссылок на шаренную память

03. Детальные метрики для отладки и поиска проблем

# Баг питона при работе с очередями



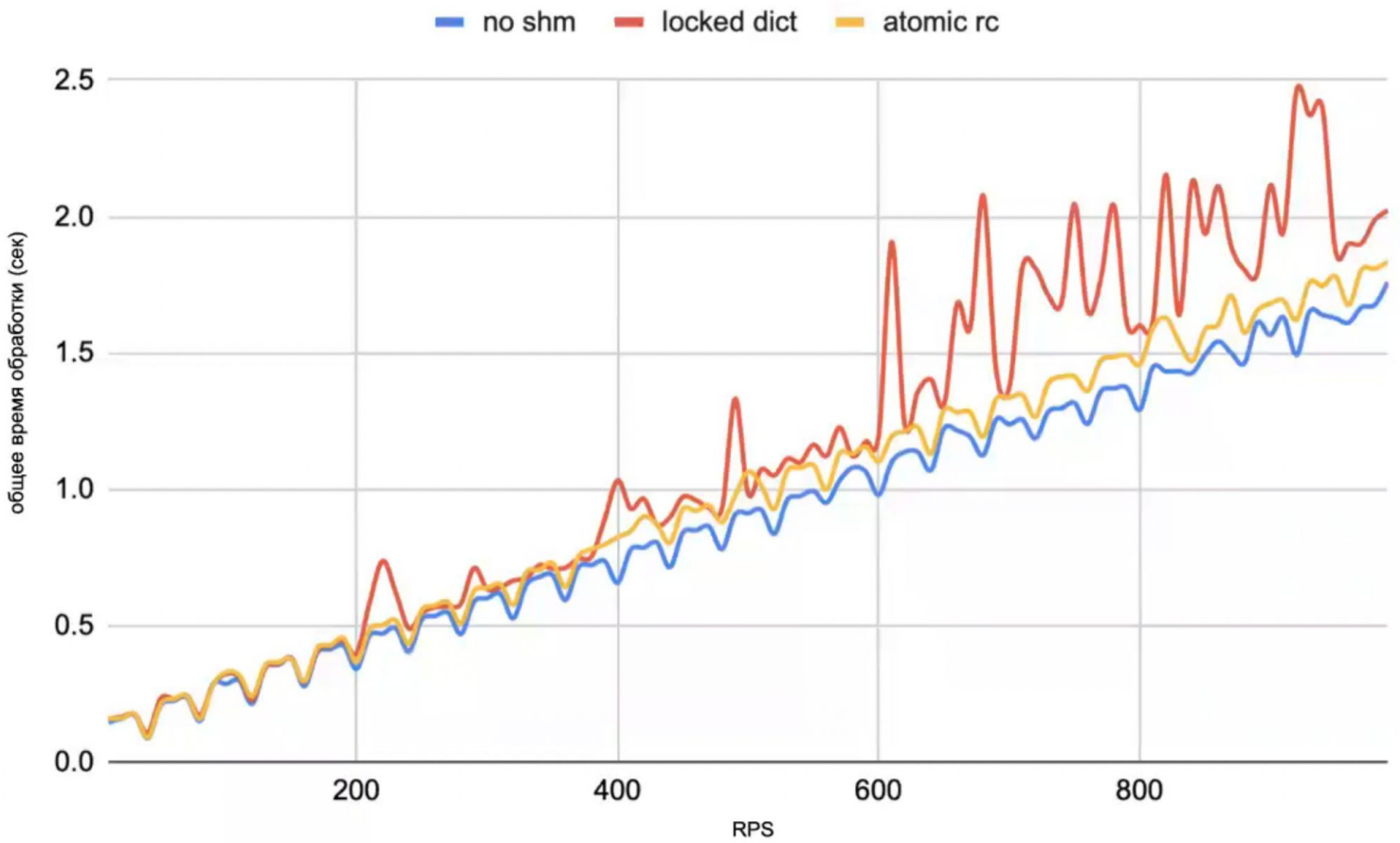
# Баг питона при работе с очередями, решение:

```
i = 0
while not task and i < MAX_SPIN_ITERATIONS:
    i += 1
    try:
        task = self.queue_in.get(timeout=timeout)
    except queue.Empty:
        if self.queue_in.qsize() == 0:
            break
```

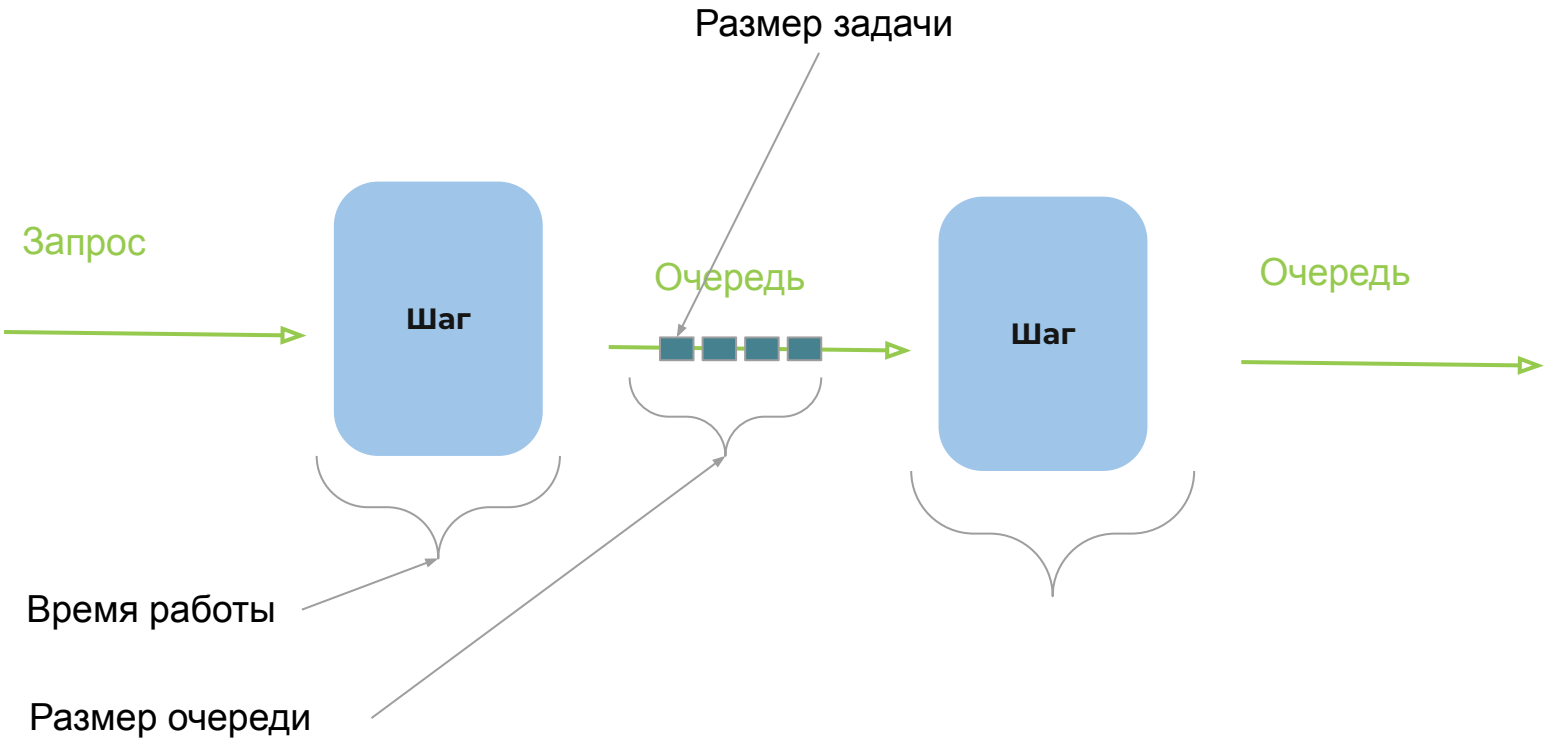
# Счетчики ссылок на шаренную память

```
uint32_t load_uint32(uint32_t *v) {
   return __atomic_load_n(v, __ATOMIC_SEQ_CST);
};
void store_uint32(uint32_t *v, uint32_t n) {
   uint32_t i = n;
   __atomic_store(v, &i, __ATOMIC_SEQ_CST);
};
uint32_t add_and_fetch_uint32(uint32_t *v, uint32_t i) {
   return __atomic_add_fetch(v, i, __ATOMIC_SEQ_CST);
};
uint32_t sub_and_fetch_uint32(uint32_t *v, uint32_t i) {
   return __atomic_sub_fetch(v, i, __ATOMIC_SEQ_CST);
};
```

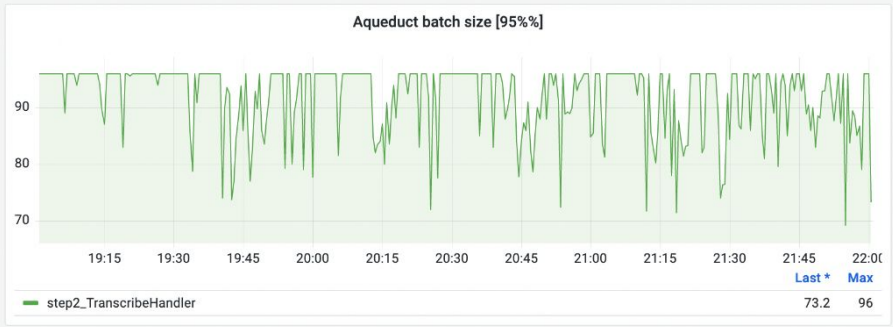
# Счетчики ссылок на шаренную память

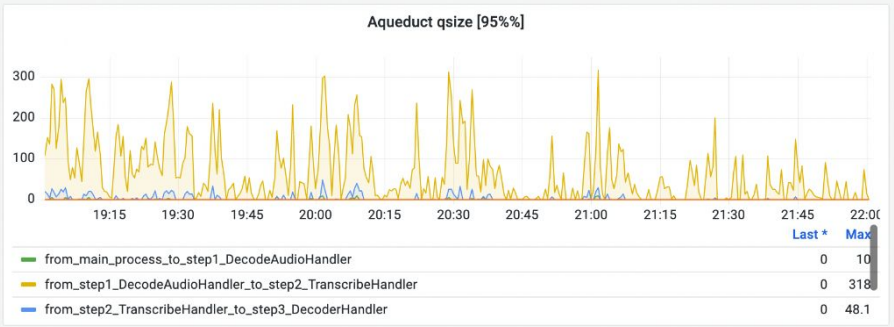


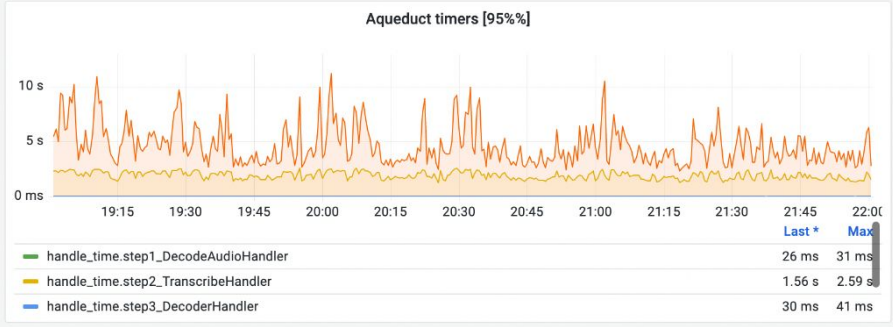
# Детальные метрики для отладки и поиска проблем

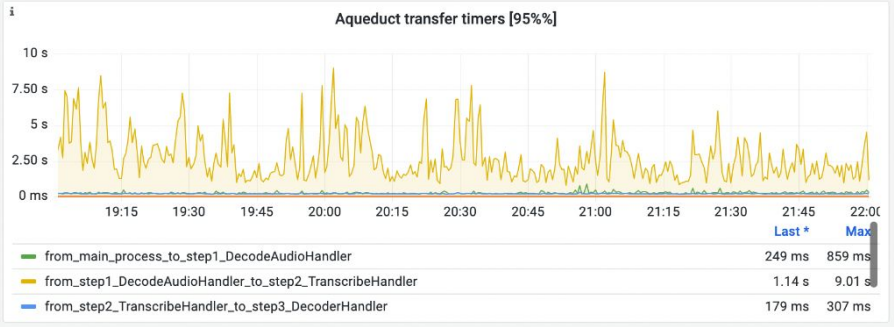


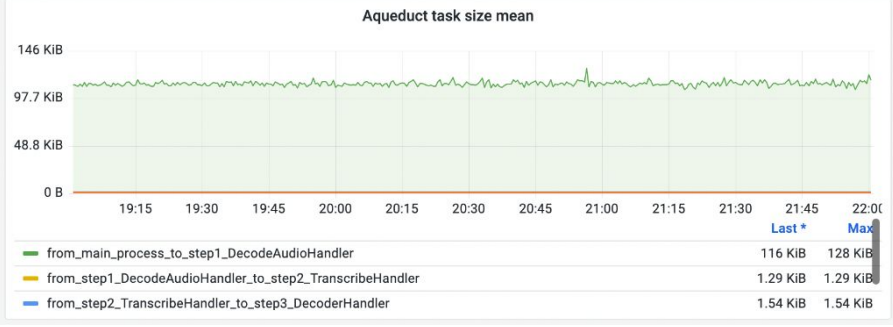
# Детальные метрики для отладки и поиска проблем

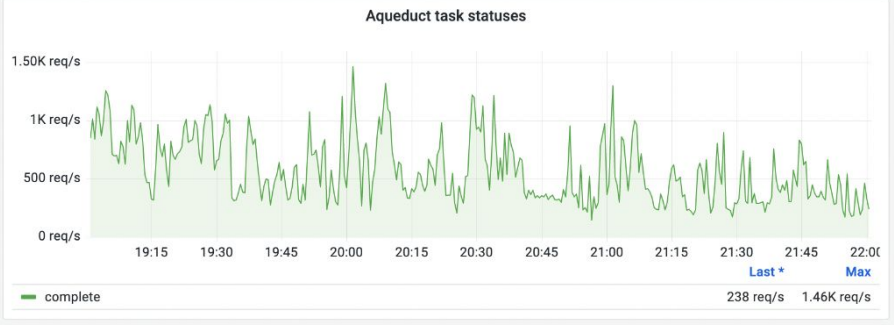


# Ньюансы выкатки МЛ-модели в продакшн

01. Отслеживание liveness вашей модели

02. Пробрасывание ошибок в сентри

03. Кастомное логирование

# Отслеживание liveness вашей модели

```
async def observe_flows(app: web.Application, check_interval: float = 1.):
    flows = {name: app[name] for name in app[AQUEDUCT_FLOW_NAMES]}
    while True:
        for flow_name, flow in flows.items():
            if not flow.is_running:
                log.info(f'Flow {flow_name} is not running, application will be stopped')
                pid = os.getpid()
                os.kill(pid, signal.SIGTERM)
                return

        await asyncio.sleep(check_interval)
```

# Пробрасывание ошибок в sentry

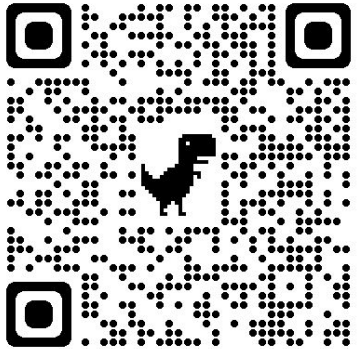
```
from raven import Client
from raven.handlers.logging import SentryHandler
from raven.transport.http import HTTPTransport
from aqueduct.logger import log

if os.getenv('SENTRY_ENABLED') is True:
    dsn = os.getenv('SENTRY_DSN')
    sentry_handler = SentryHandler(client=Client(dsn=dsn, transport=HTTPTransport),
                                   level=logging.ERROR)
    log.addHandler(sentry_handler)
```

Обычно логер настраивается в родительском процессе, поэтому при спавне важно не забыть про настройки логера

# Кастомное логирование

```
import logging
import sys

LOGGER_NAME = 'aqueduct'

log = logging.getLogger(LOGGER_NAME)
log.setLevel(logging.DEBUG)
formatter = logging.Formatter('[%(asctime)s] %(levelname)s [Flow] [pid:%(process)d] %(message)s')
ch = logging.StreamHandler(sys.stderr)
ch.setFormatter(formatter)
log.addHandler(ch)



def replace_logger(logger: logging.Logger) -> None:
    """Replaces default aqueduct logger with custom one"""
    log.name = logger.name
    log.level = logger.level
    log.parent = logger.parent
    log.propagate = logger.propagate
    log.handlers = logger.handlers
    log.disabled = logger.disabled
```

# Акведук (Aqueduct)



Мы собрали решения этих проблем в одной библиотеке:

- Вынесение вычислений в подпроцессы
- Передача данных  к модели через очереди
- Скейлинг
- Батчирование
- Счетчики ссылок на шаренную память
- Баг питона при работе с очередями
- Детальные метрики для отладки и поиска проблем
- Отслеживание liveness вашей модели
- Пробрасывание ошибок в сентри
- Логирование

# Спасибо!

## Акведук (Aqueduct)

- Вынесение вычислений в подпроцессы
- Передача данных  к модели через очереди
- Скейлинг
- Батчирование
- Счетчики ссылок на шаренную память
- Баг питона при работе с очередями
- Детальные метрики для отладки и поиска проблем
- Отслеживание liveness вашей модели
- Пробрасывание ошибок в сентри
- Логирование