

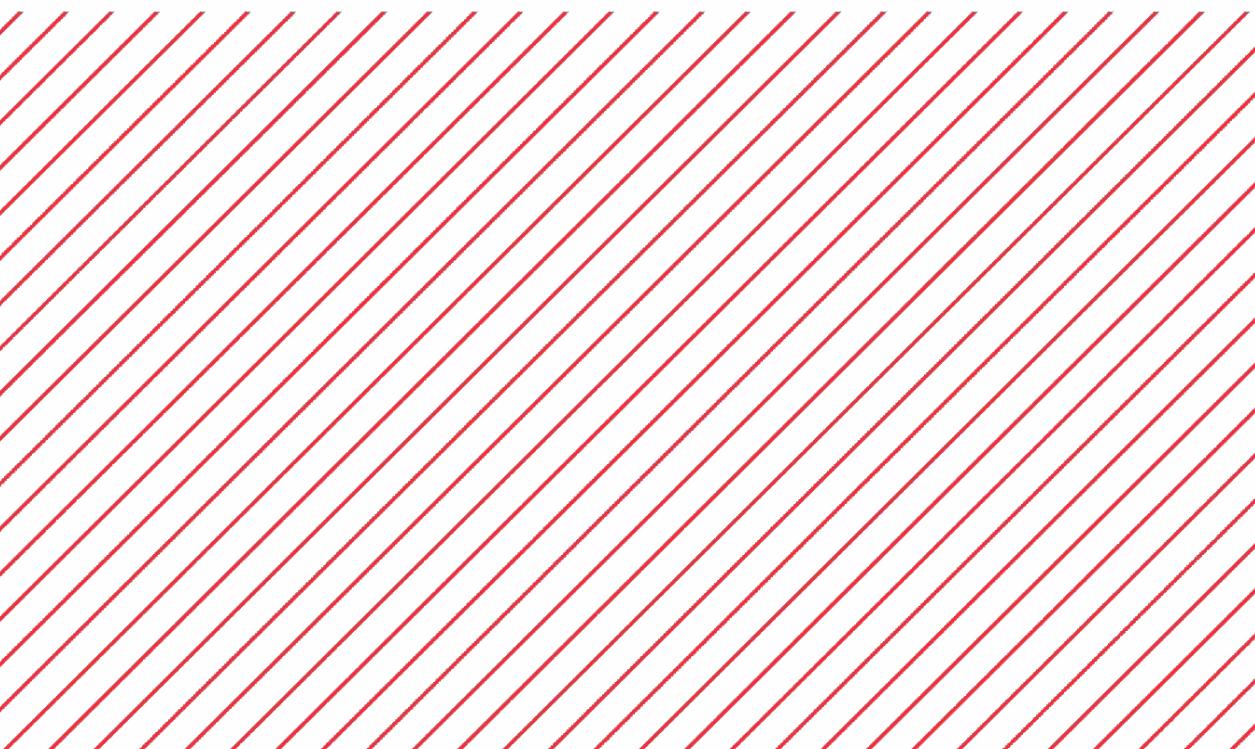
академия  
больших  
данных

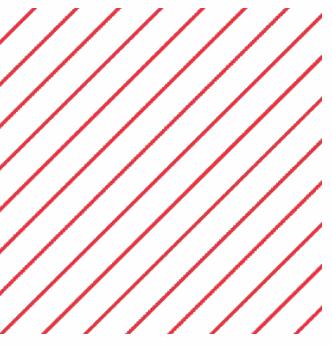


# Деплой в продакшен

Юлиана Лихолай

Руководитель группы С/С++ разработки Почты





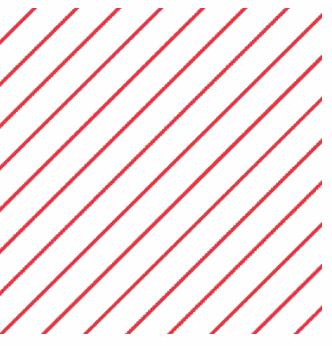
# План лекции

---

- Кратко о Backend разработке
- Особенности Backend'а для обработки изображений
- Деплой Backend'а
- Мониторинг Backend'а
- Примеры проектов



# Немного о Backend разработке



# Веб разработка

---

- **Frontend** - то, что отображается на клиентской стороне в веб-браузере, веб-приложении
  - Языки: HTML, CSS и JavaScript
- **Backend** - то, что работает на удаленном сервере
  - Языки: PHP/Python/Ruby/C++/GO/Java/
  - БД: MySQL, MongoDB, Tarantool
  - Очереди: RabbitMQ, Redis, Tarantool



# Backend

---

Backend - это **асинхронный HTTP Server**

**HTTP** - *HyperText Transfer Protocol*, протокол передачи данных

## Структура HTTP:

1. **Стартовая строка** (англ. Starting line) — определяет тип сообщения;

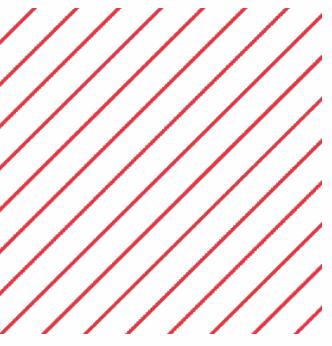
```
###Метод URI HTTP/Версия
POST /api/v1/persons/recognize?oauth_provider=mcs HTTP/1.1
```

2. **Заголовки** (англ. Headers) — характеризуют тело сообщения;

```
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509 Firefox/3.0b5
Content-Type: application/json
Content-Length: 3296531
```

3. **Тело сообщения** (англ. Message Body) — непосредственно данные сообщения.

```
{"model": "frsa12", "score": 0.3}
```



# Backend REST API

---

REST API подразумевает под собой простые правила:

- Каждый URL является ресурсом

```
/api/v1/persons  
/api/v1/objects  
/api/v1/cars
```

- При обращении к ресурсу методом **GET** возвращается **описание**

```
GET /api/v1/persons HTTP/1.1
```

- Метод **POST** **добавляет** новый ресурс

```
POST /api/v1/persons/ HTTP/1.1
```

- Метод **PUT** **изменяет** ресурс

```
PUT /api/v1/persons/4 HTTP/1.1
```

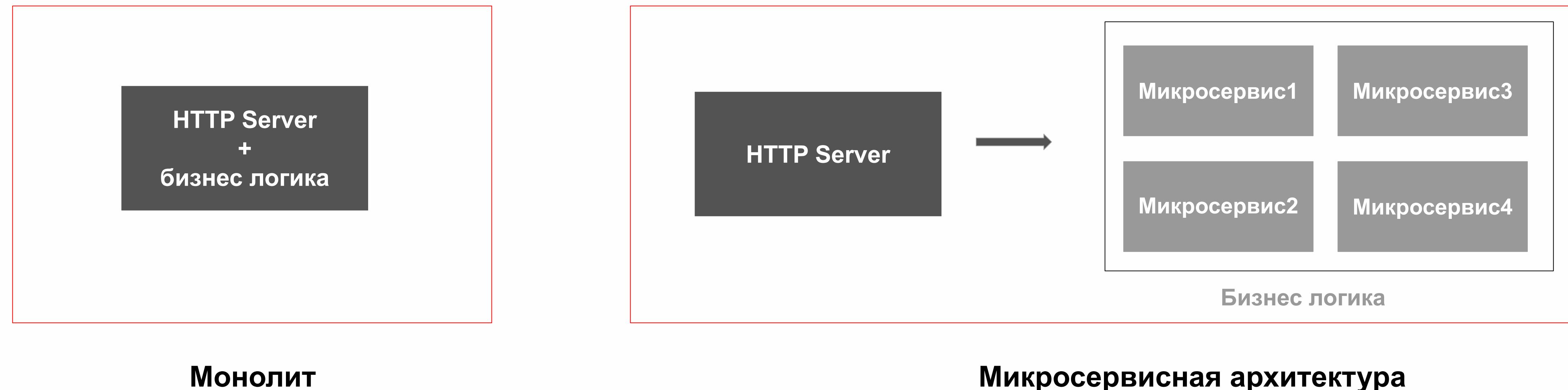
- Метод **DELETE** **удаляет** ресурс

```
DELETE /api/v1/persons/4 HTTP/1.1
```

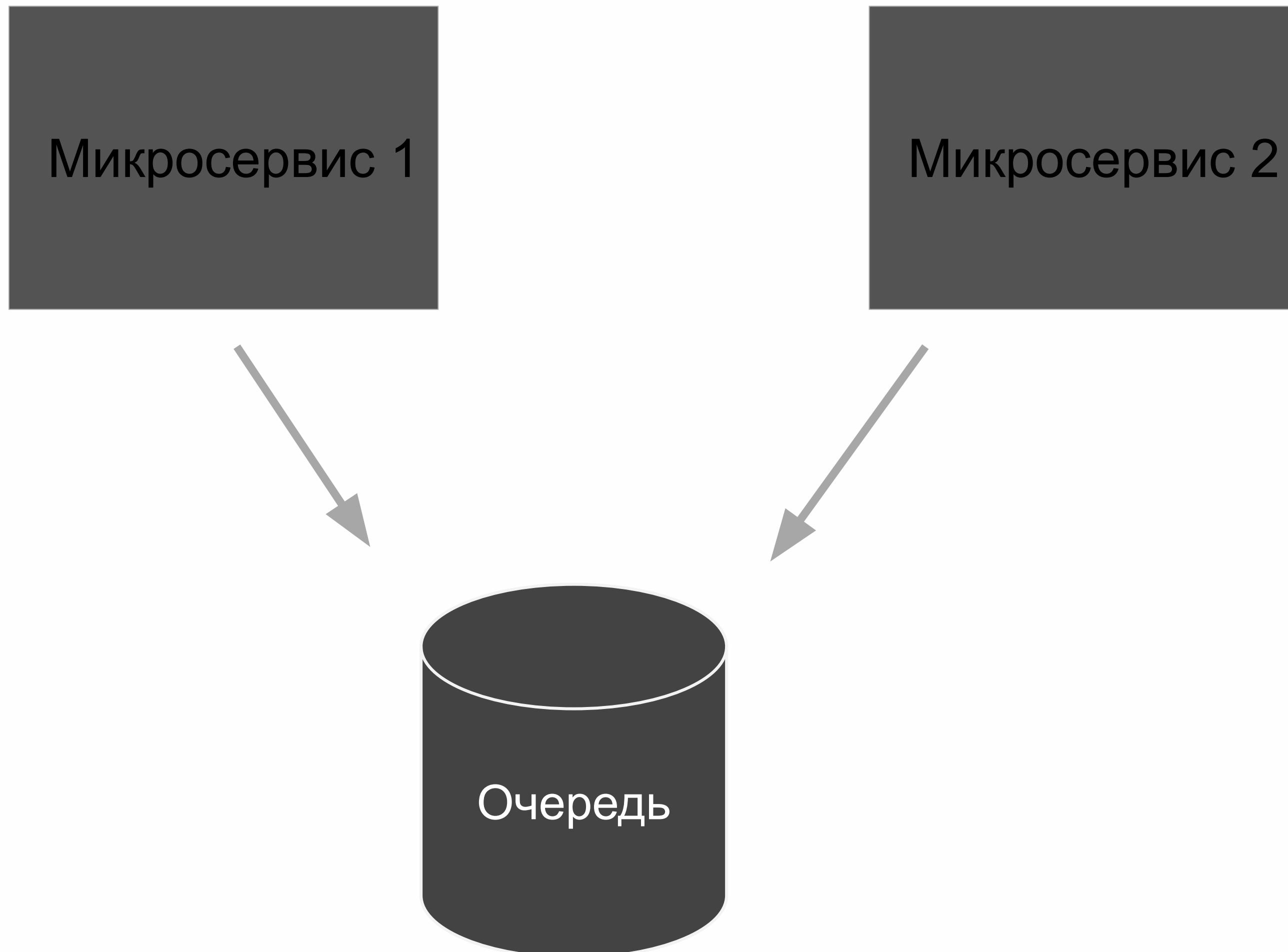
# Backend Архитектура

---

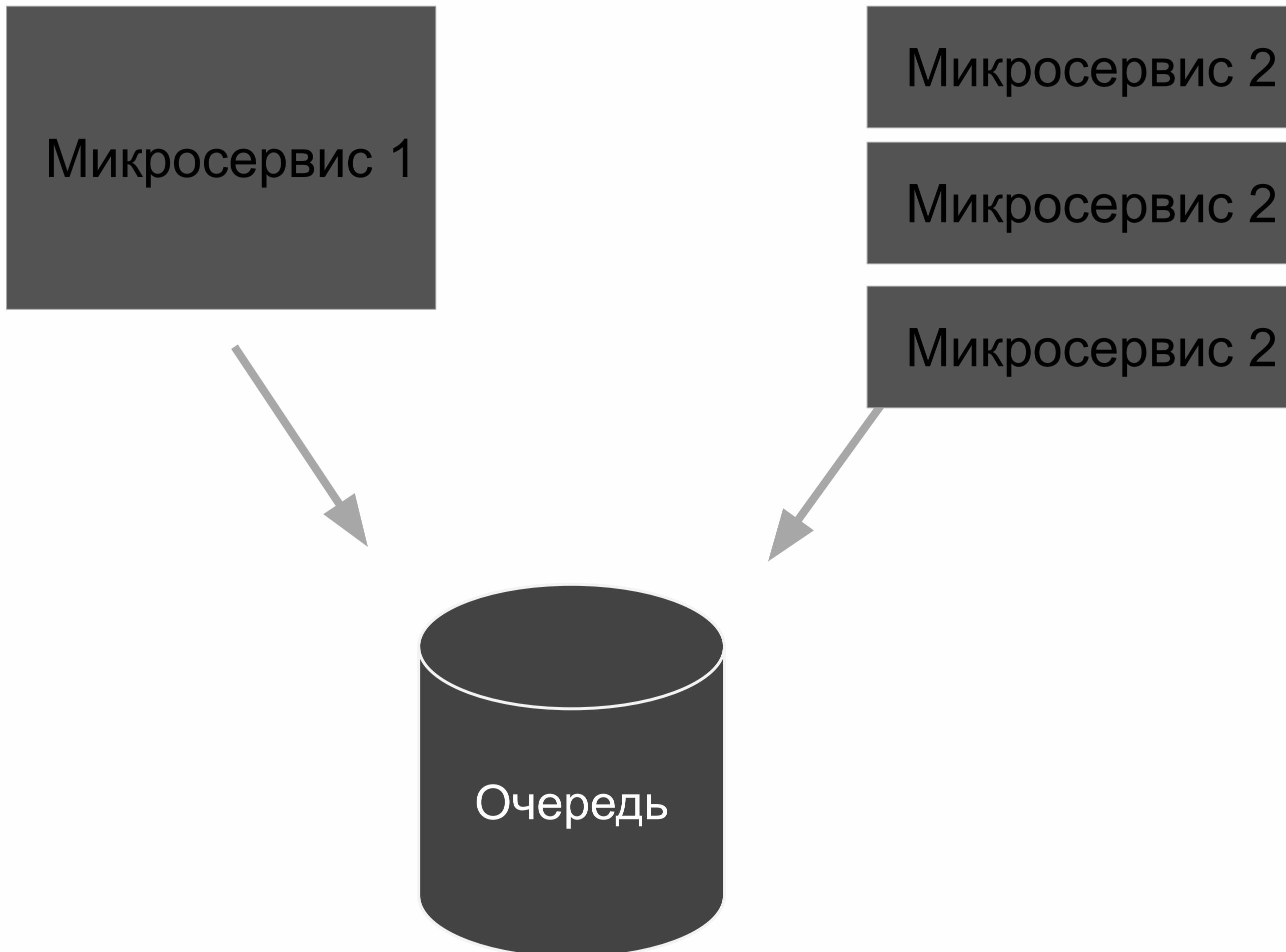
Архитектура Backend'а - монолит или микросервисы



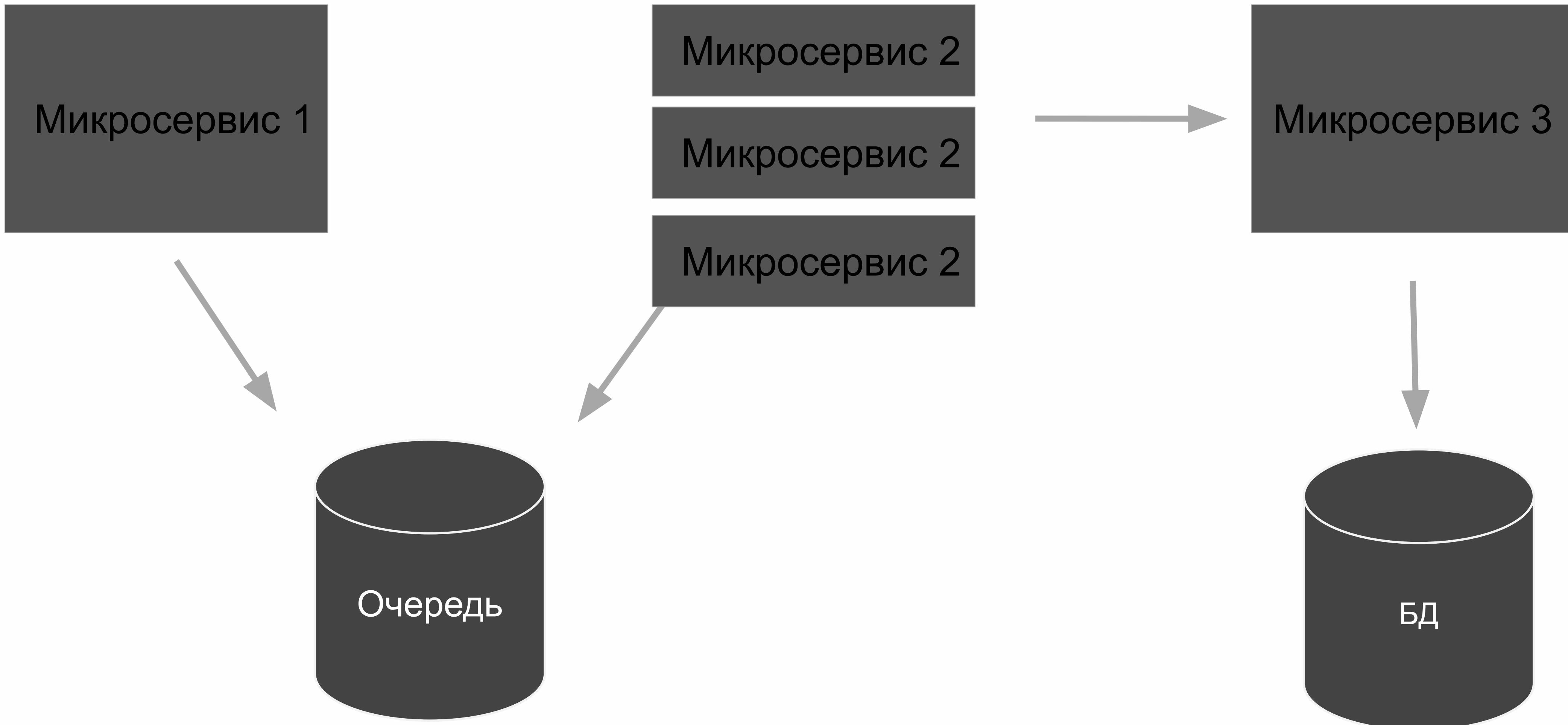
# Архитектура backend'a



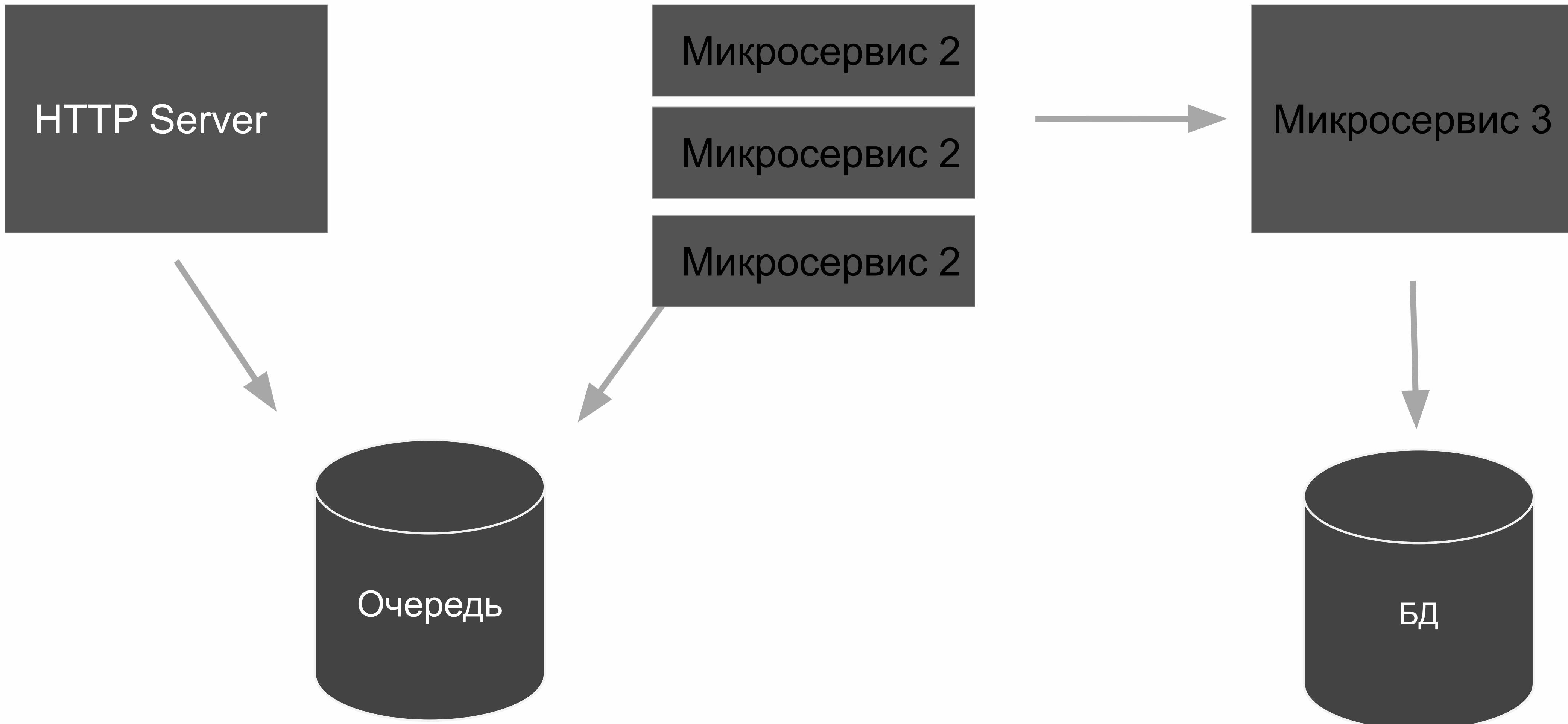
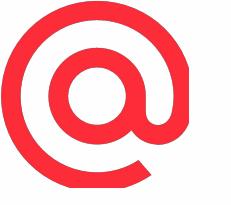
# Архитектура backend'a



# Архитектура backend'a

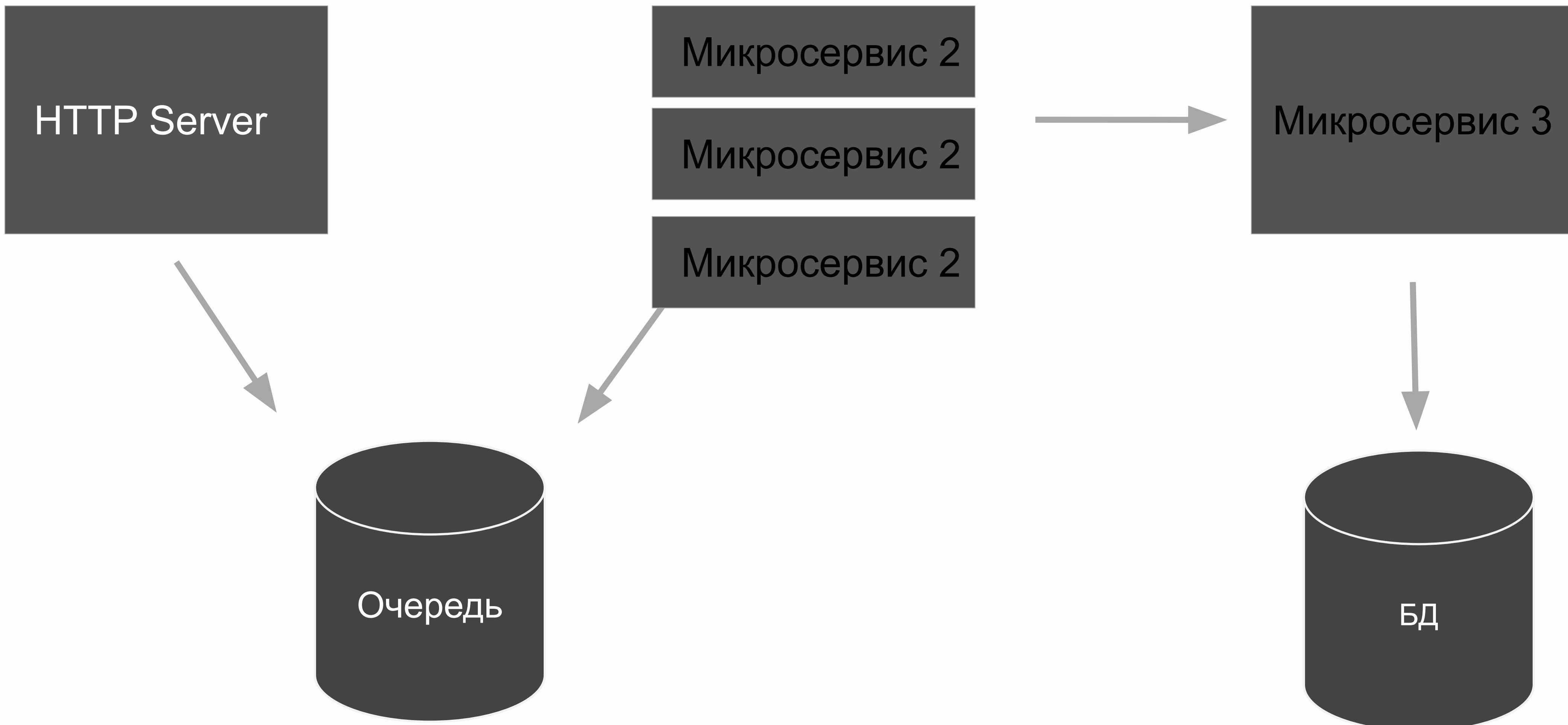
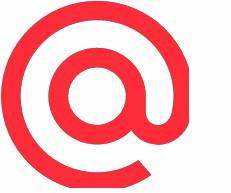


# Архитектура backend'a

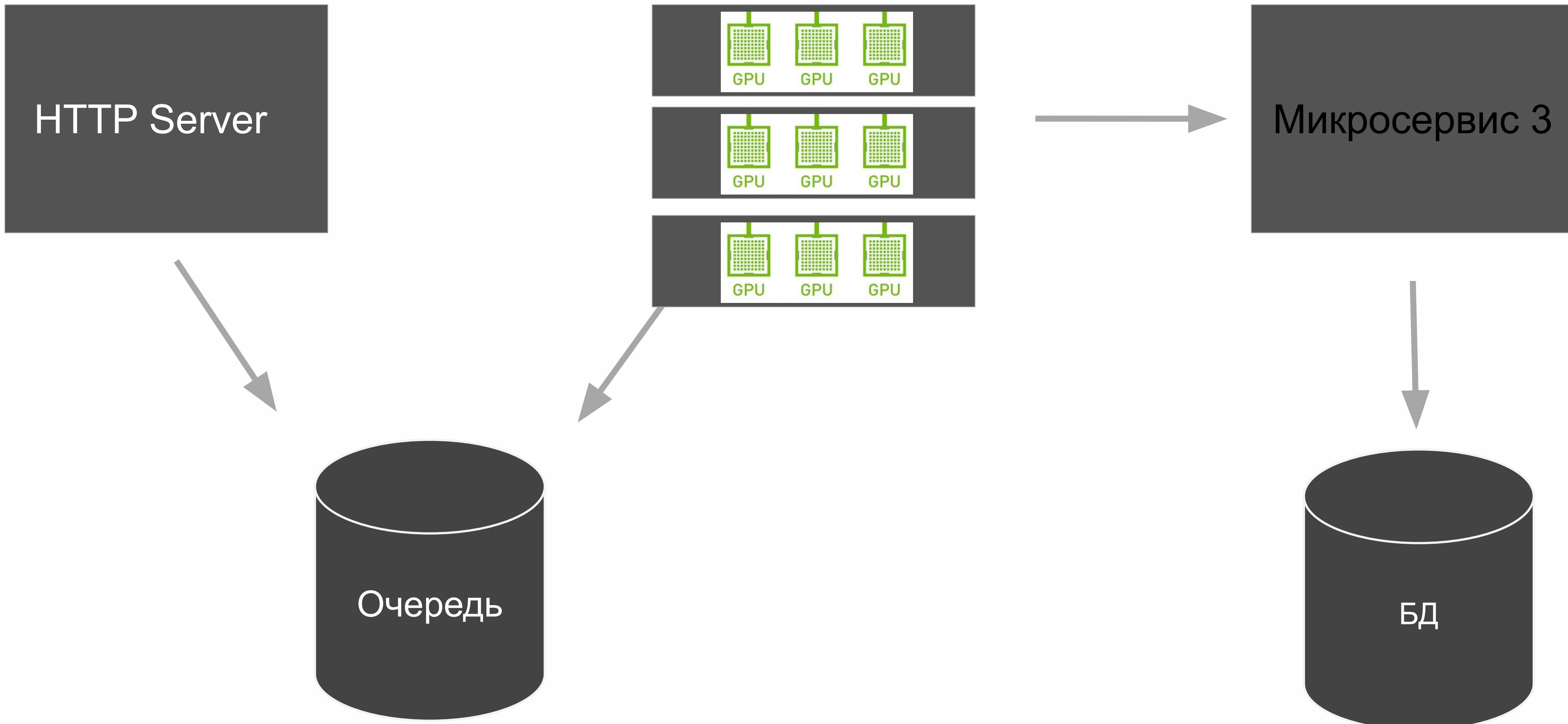


# Особенности Backend'а для обработки изображений

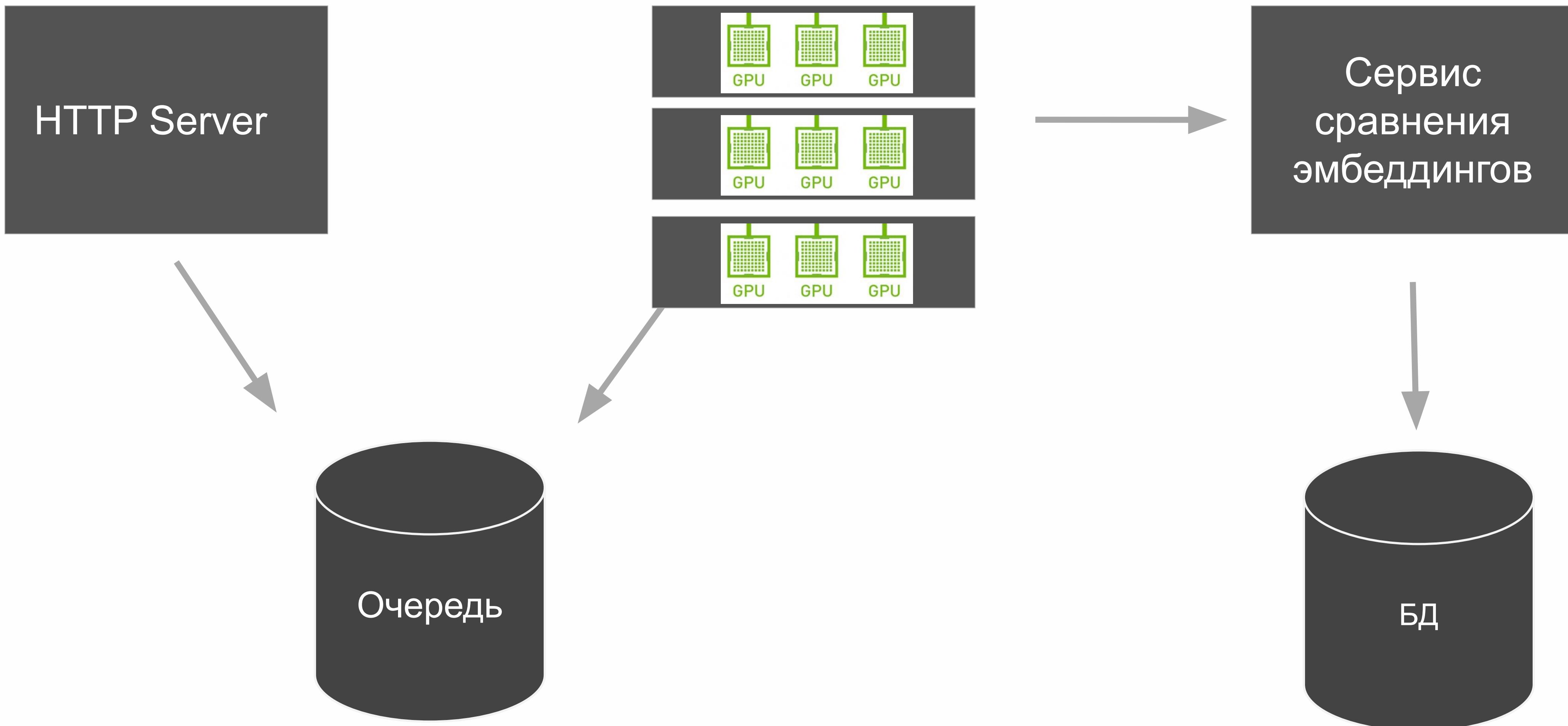
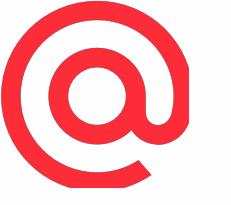
# Архитектура backend'a



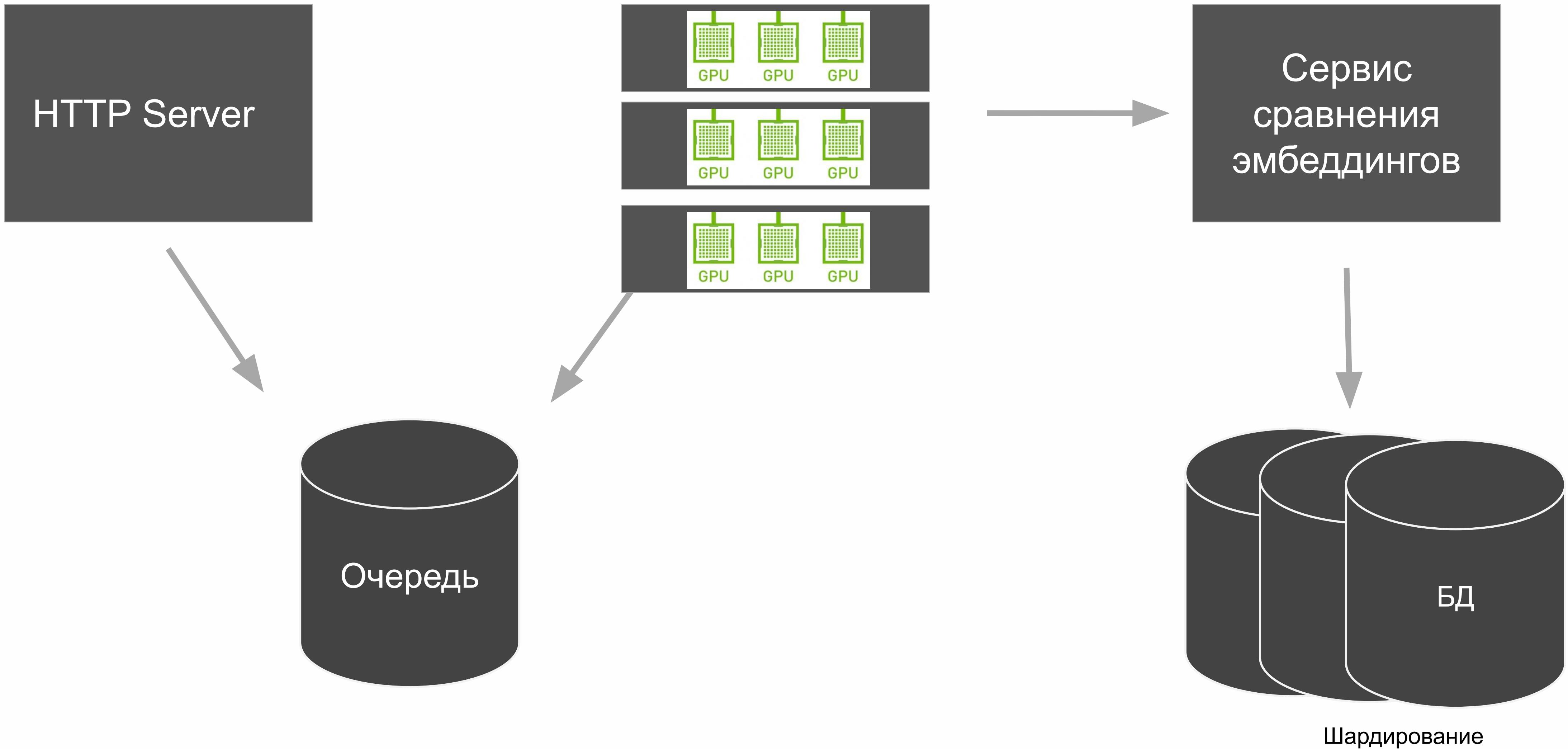
# Архитектура backend'a



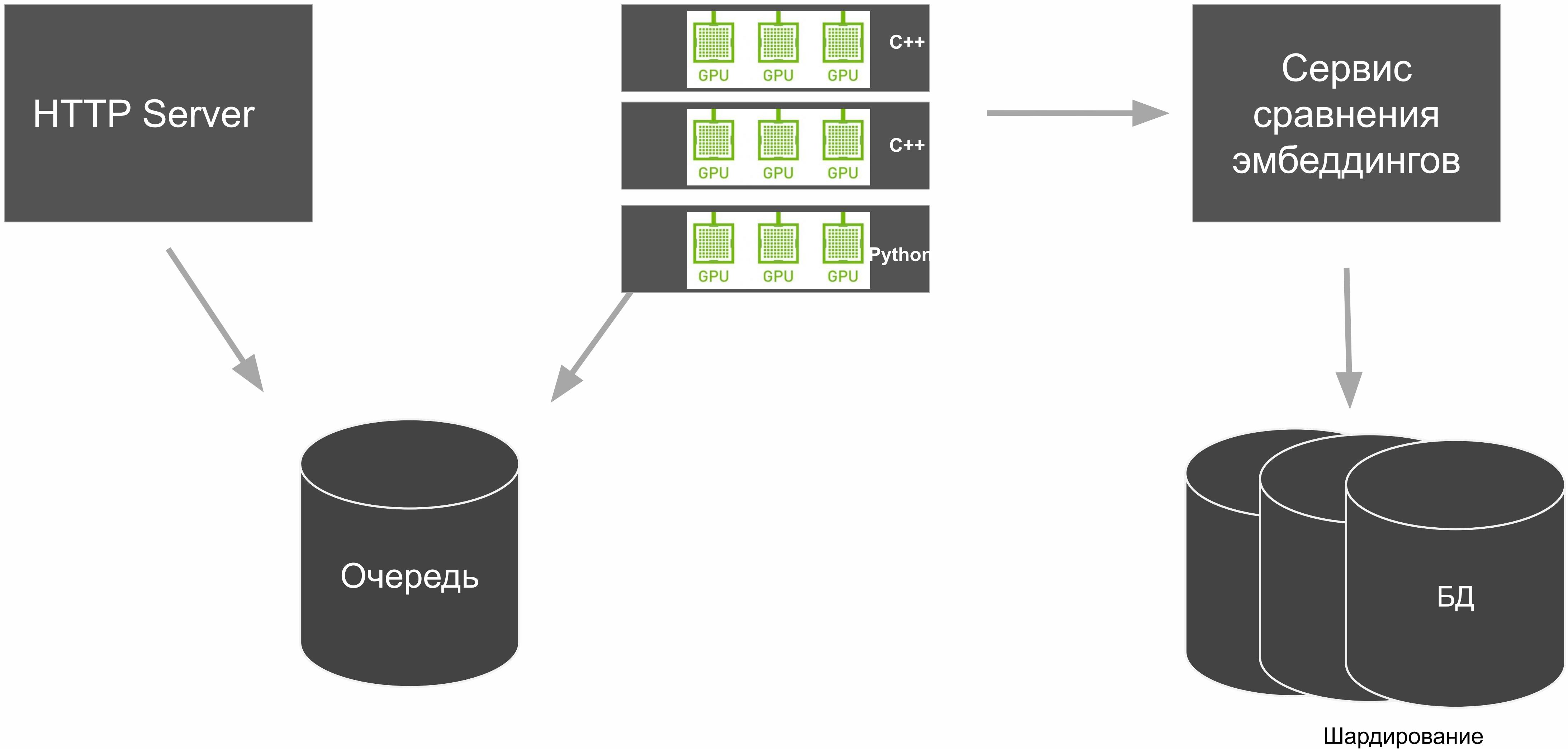
# Архитектура backend'a



# Архитектура backend'a



# Архитектура backend'a

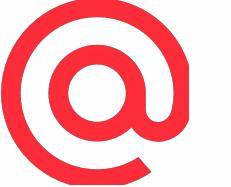


# Архитектура очереди

@



# Реализация: GPU Utilization



Самый дорогой ресурс в сервере - GPU карта  
Надо максимизировать утилизацию видеокарты

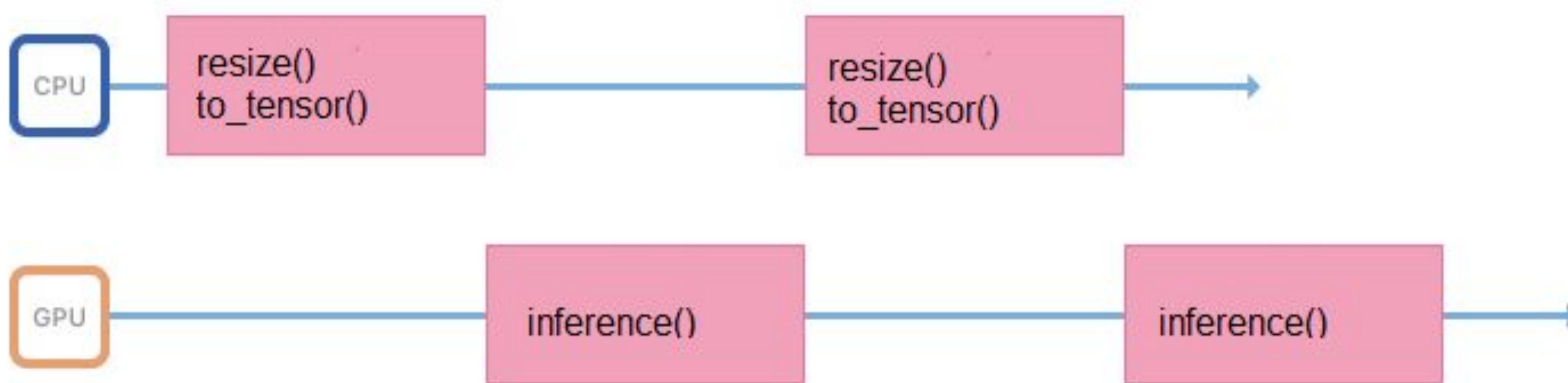
10x GPU 10-20% util  $\Rightarrow$  2x GPU 50-100% util

NVIDIA-SMI 375.26							Driver Version: 375.26
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	
0		... Off	0000:01:00.0	Off	84%	N/A	
22%	50C	P2	158W / 250W	2246MiB / 12206MiB		Default	

# Реализация: data pipeline

@

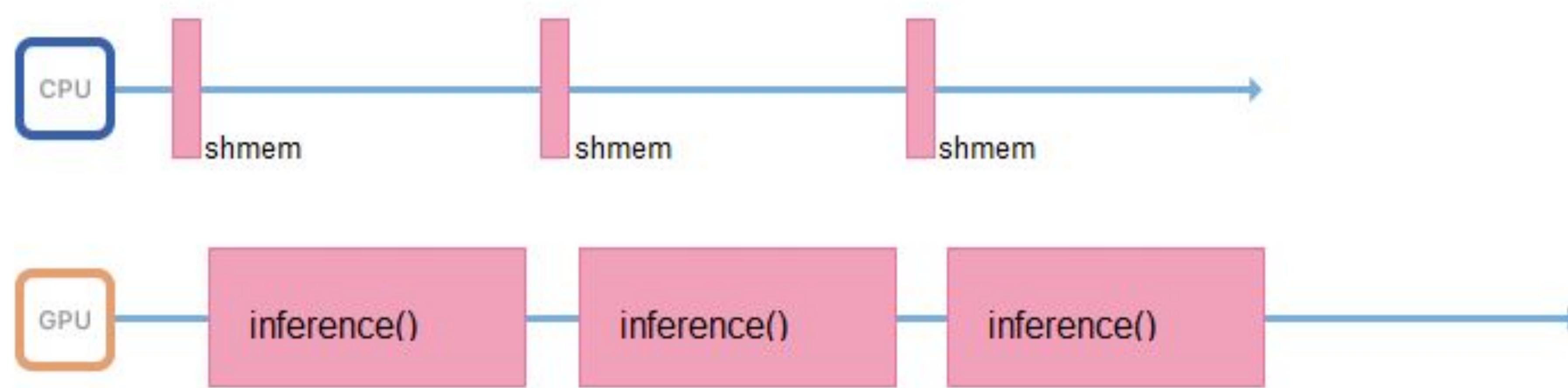
GPU util <= 70%



# Реализация: data pipeline

@

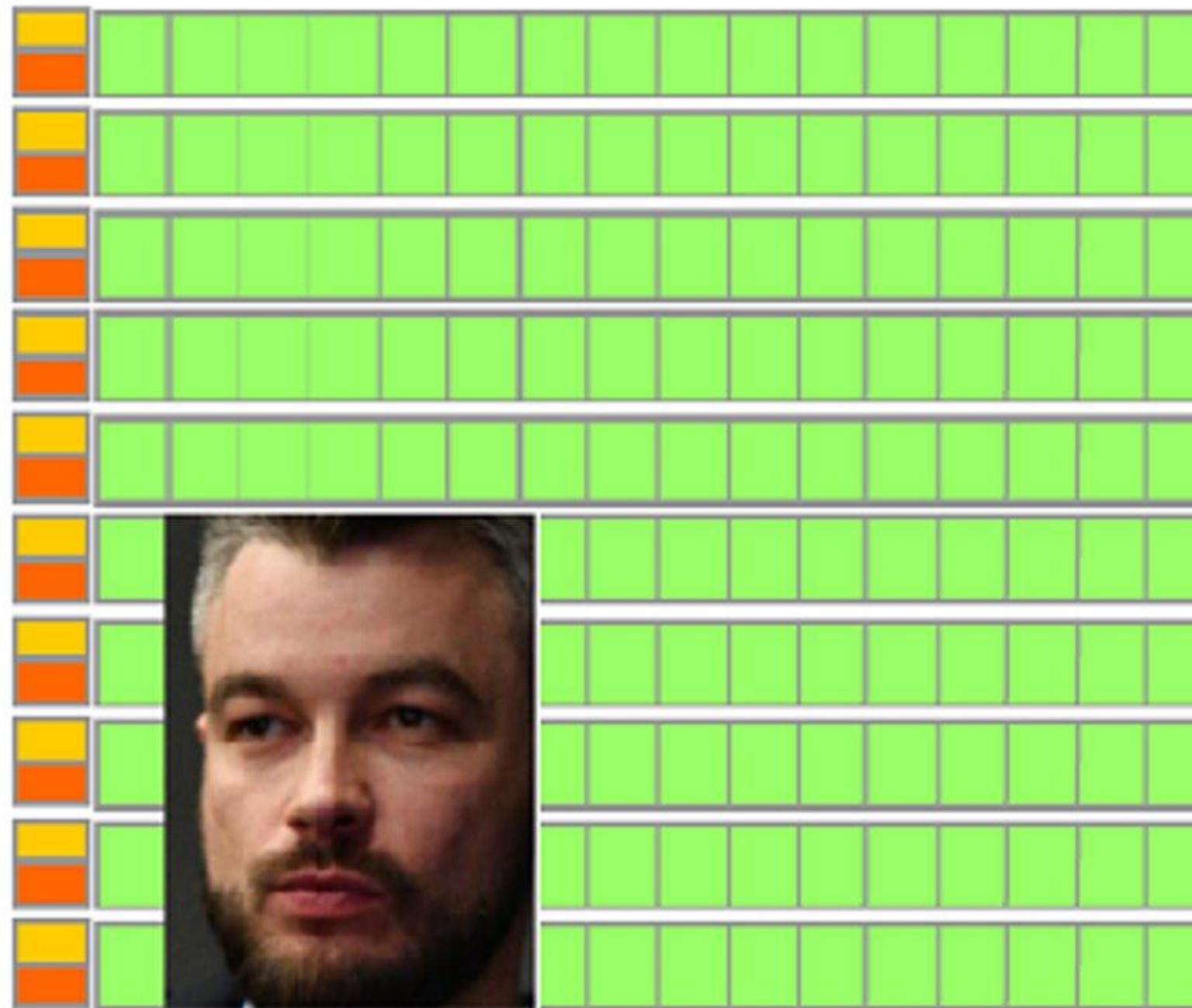
GPU util ~ 95%



# Реализация: Batching

---

GPU util <= 25%

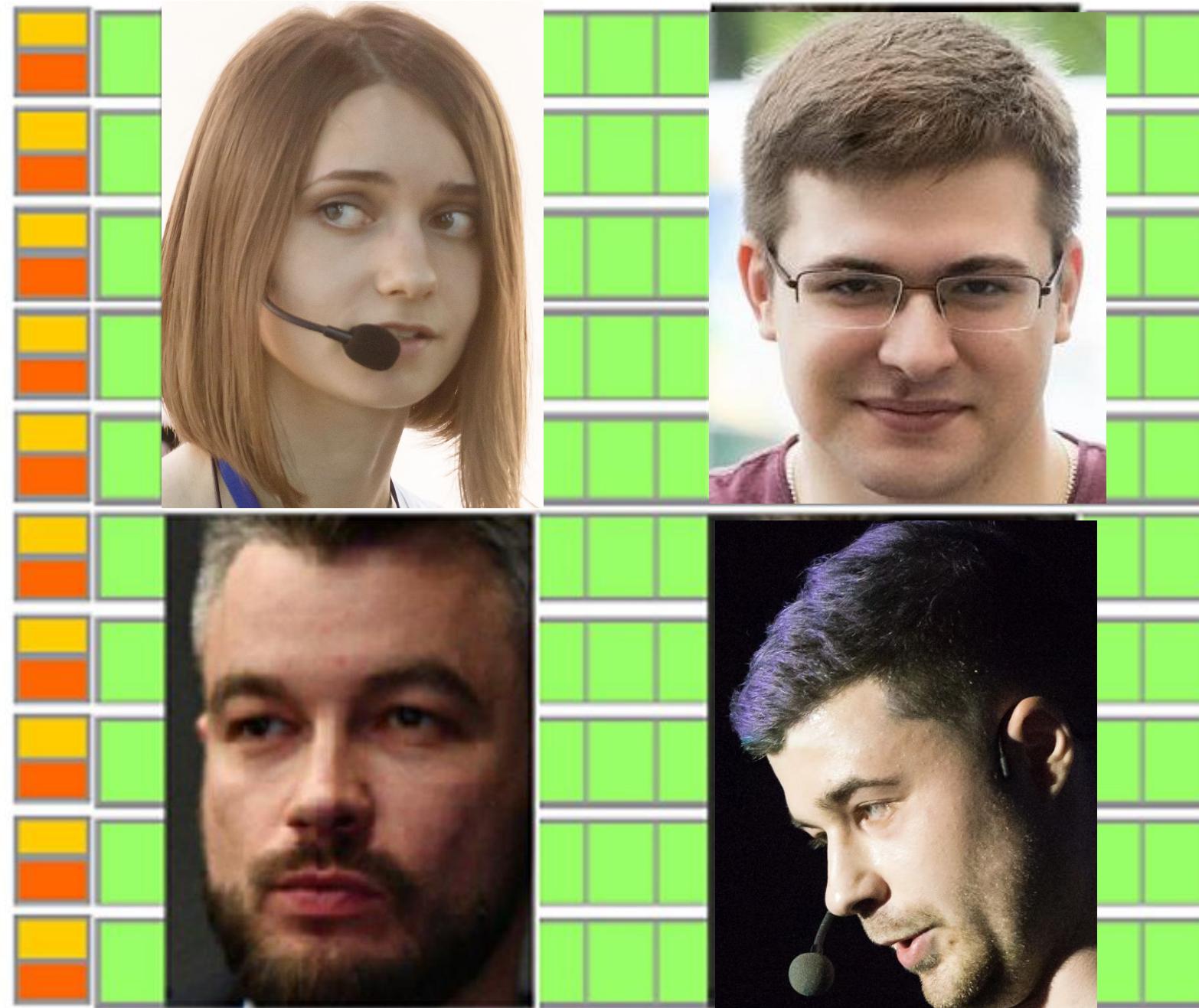


GPU

# Реализация: Batching

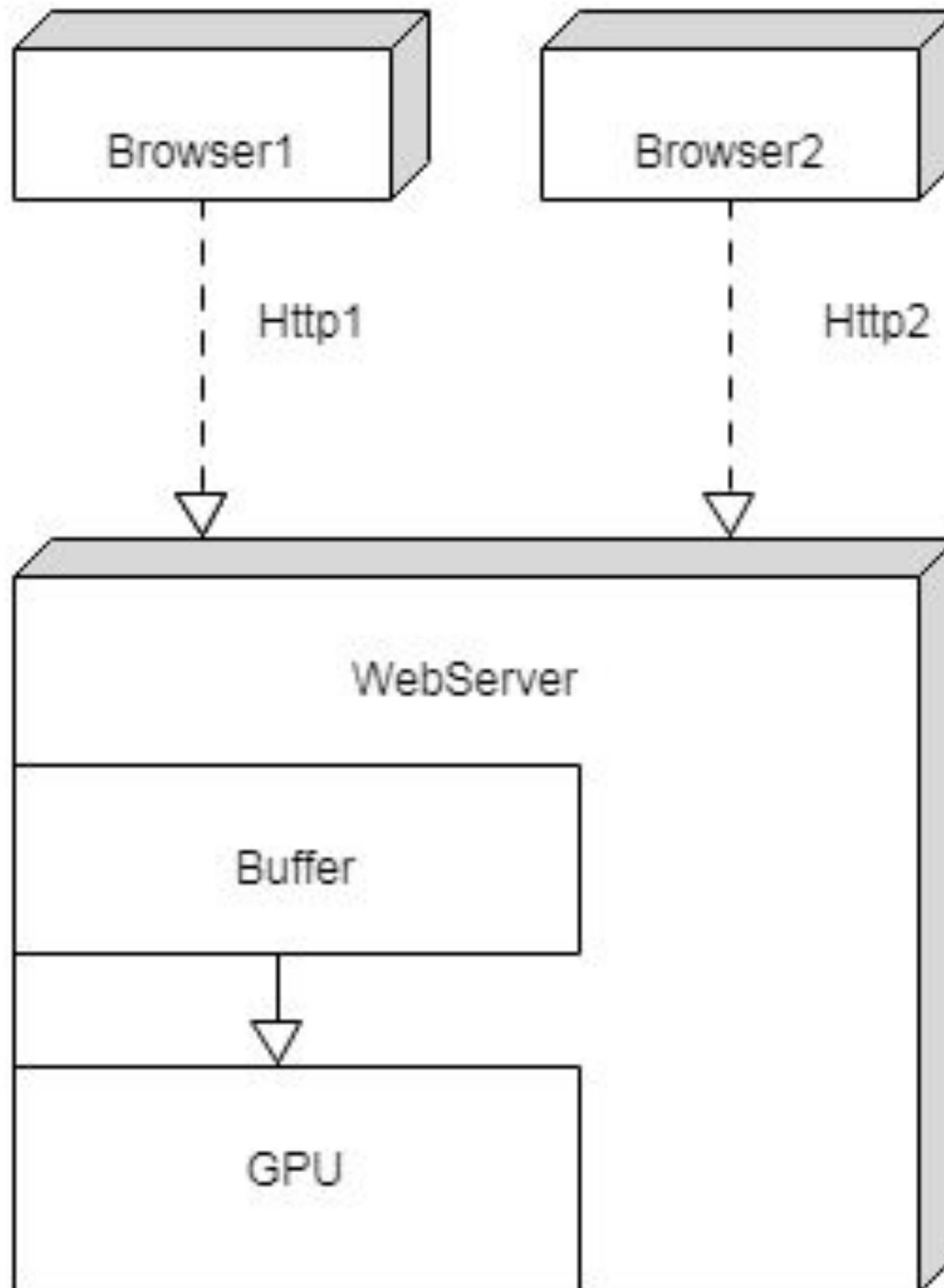
---

GPU util ~ 95%



GPU

# Реализация: fix latency strategy



Стратегия фиксированной задержки времени обработки одного запроса

Запросы копятся во временной буфере для единой обработки (батчинга)

Если в течение 50 msec не удалось скопить нужное количество запросов, обрабатываем то, что есть

# Реализация: выбор метода CV

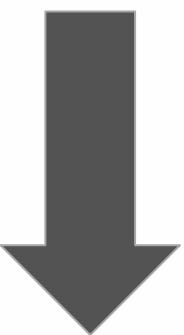
@

Viola-Jones / HOG ~ 20-40ms CPU, <= 75% accuracy

# Реализация: выбор метода CV

@

Viola-Jones / HOG ~ 20-40ms CPU, <= 75% accuracy



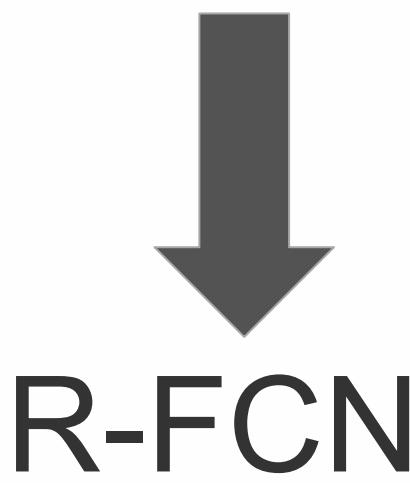
R-FCN

~ 40ms GPU, 92% accuracy

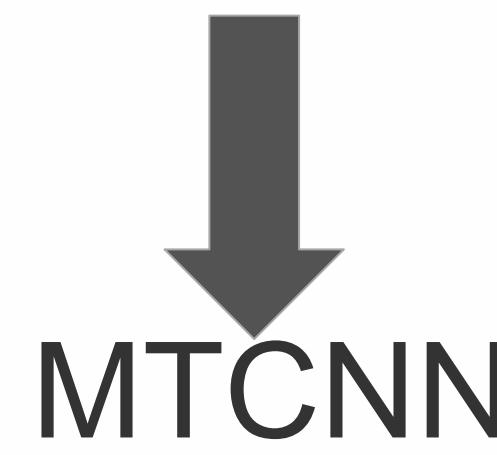
# Реализация: выбор метода CV

@

Viola-Jones / HOG ~ 20-40ms CPU, <= 75% accuracy



~ 40ms GPU, 92% accuracy



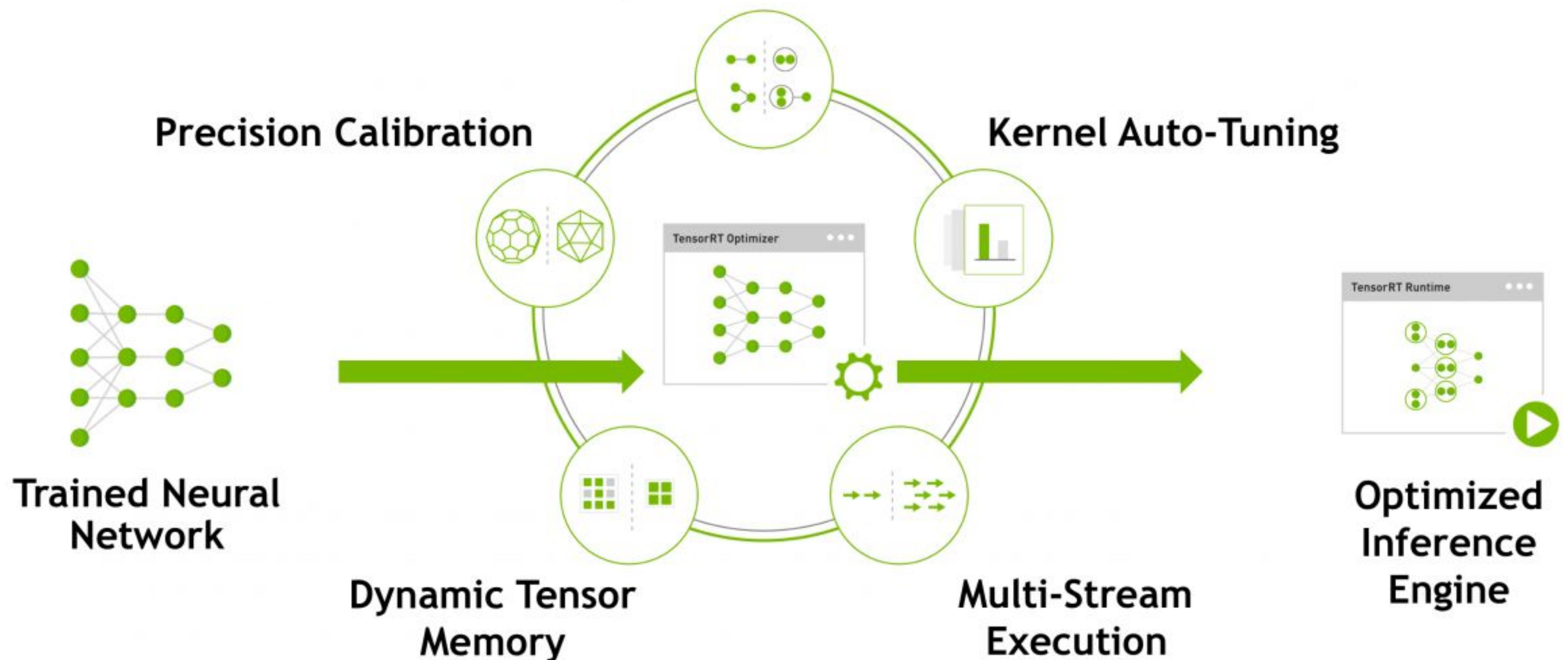
~ 20ms GPU, 90% accuracy

\* R-FCN: Region-based Fully Convolutional Networks

\* MTCNN: Multi-task Cascaded Convolutional Networks

# Реализация: использование TensorRT

@



# Реализация:

@

Caffe

# Реализация:

@

Caffe



# Реализация:

@

Caffe



 Caffe2

The logo icon for Caffe2 features a stylized coffee cup with a plus sign inside it, enclosed in a square frame.

# Реализация: проблемы

@

Caffe



 Caffe2

The Caffe2 logo features a white square containing a dark blue icon of a coffee cup with a plus sign above it, followed by the text "Caffe2" in a dark blue sans-serif font.

# Реализация: проблемы

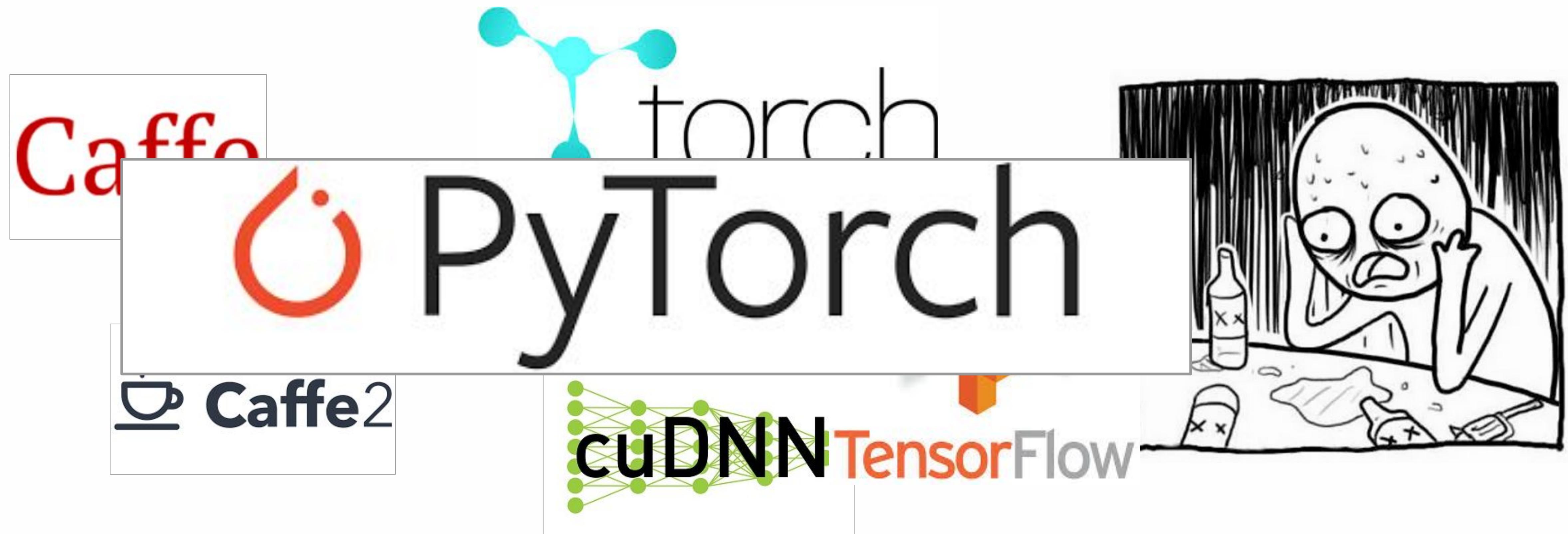
@



# Реализация: проблемы

@

- Запуск нескольких framework'ов на 1 GPU



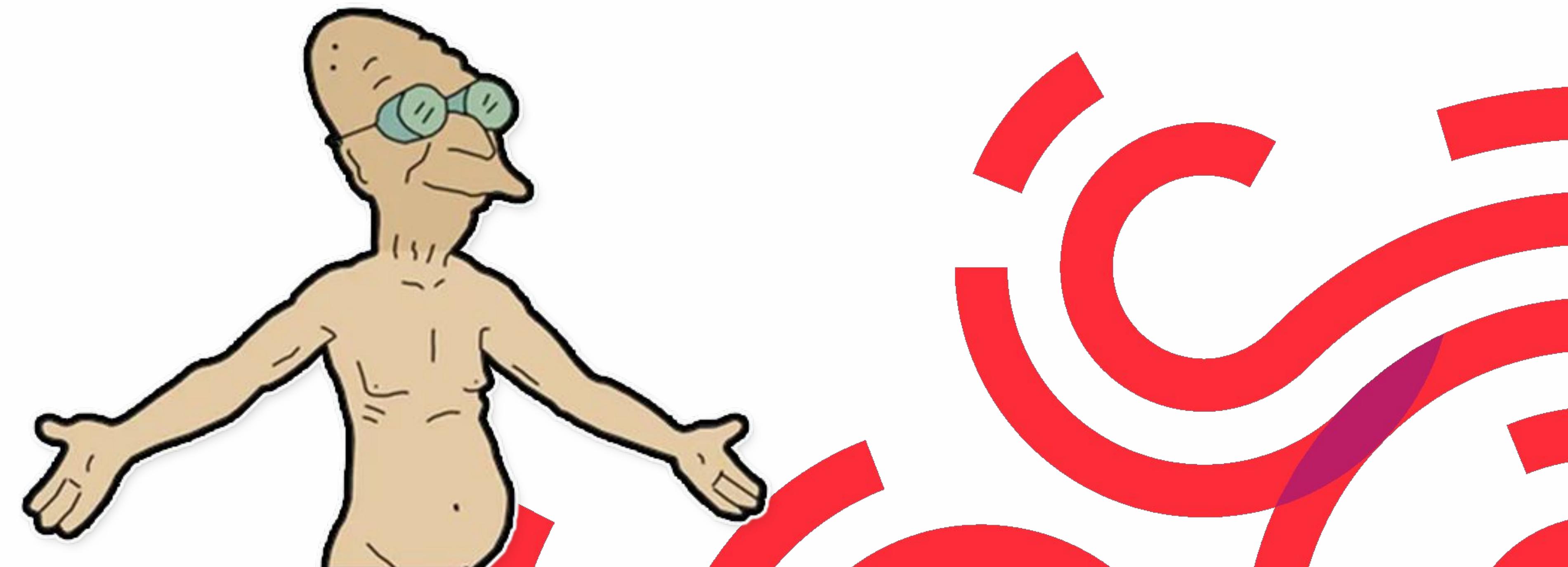


# Реализация: проблемы

---

- При использование нескольких framework'ов на 1 GPU могут возникать взаимные блокировки
- Трудно контролировать размер использованной памяти GPU → трудно наращивать количество инстансов на 1 GPU

**1 GPU = 1 Модель**



# Реализация: стандартная архитектура

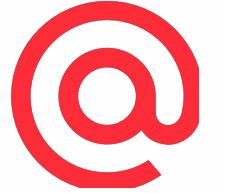




**1 GPU = 1 Модель = 1 Instance  
Worker'a**



# Реализация: Распределение моделей



**visapi1.i**

Edit Delete node

gpu #0: ["photoaws"]  
gpu #1: ["multiobject"]  
gpu #2: ["car"]  
gpu #3: ["doc2"]

20190613T15:56:09

**visapi2.i**

Edit Delete node

gpu #0: ["scene"]  
gpu #1: ["scene"]  
gpu #2: ["scene"]  
gpu #3: ["scene"]

20190613T15:56:09

**visapi4.i**

Edit Delete node

gpu #0: ["pedestrian"]  
gpu #1: ["pedestrian"]  
gpu #2: ["pedestrian"]  
gpu #3: ["pedestrian"]

20190613T15:56:09

**visapi5.i**

Edit Delete node

gpu #0: ["doc"]  
gpu #1: ["doc"]  
gpu #2: ["doc"]  
gpu #3: ["doc"]

20190613T15:56:09

# Деплой Backend'a



# Пакет

---

На серверах чаще всего используется **Linux**, дистрибутивы **centos**, реже **ubuntu**.

Пакет содержит:

- зависимости от других пакетов
- исполняемый файл
  - скомпилированный бинарный файл “exe” (C++, Go)
  - скрипт (Python)
- конфигурационный файл для исполняемого файла



# RPM

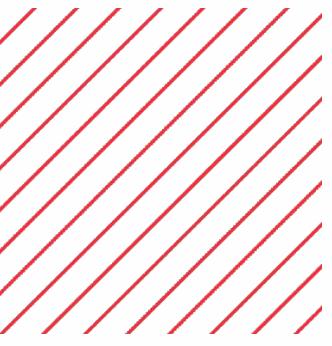
---

**RPM**, Red Hat Package Manager, (Менеджер пакетов Red Hat) - это открытая система управления пакетами, доступная всем, и работающая в Red Hat Enterprise Linux, а также других системах Linux и UNIX.

Для создания пакета нужен **spec-файл**:

- является текстовым файлом
- имеет суффикс .spec
- содержит в себе название пакета, версию, номер выпуска, инструкции по сборке и установке пакета и список изменений
- пакет создаётся командой rpmbuild

```
yum install gpu_package
```



# RPM spec-файл

---

Name: gpu\_backend

Version: 0.13.1

Release: 1

BuildRequires: boost-devel // зависимости сборки

BuildRequires: python-devel

Requires: libtorch = 20200413.1414-1.el7 // зависимости от других пакетов

Obsoletes: cpu\_worker // какие пакеты делает устаревшими

Conflicts: tripwire // с какими пакетами конфликтует

%description

Backend computer vision daemon

%build

make -f Makefile.config

%install

install -D BUILD/gpubackend %{buildroot}/usr/local/gpu/gpubackend

install -D conf/gpubackend.conf %{buildroot}/etc/gpubackend.conf



# Деплой: Puppet

---

**Puppet** – система управления конфигурацией на нескольких серверах.

Архитектура – клиент-серверная, на сервере хранятся конфиги (манифести), клиенты обращаются к серверу, получают их и применяют.

manifest:

```
file ($version_backend = 'present') {
    package { 'grubbackend': ensure => $version_backend
}
```

# Деплой: Docker

---



**Docker** - инструмент для автоматизации развертывания и управления приложением

Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему, а также предоставляет среду по управлению контейнерами.

1. **Образ** – базовый элемент, из которого можно создать много контейнеров. Образ можно залить в registry, сохранить в архив, передать куда угодно. Создается из dockerfile.
2. **Контейнер** – это исполняемый экземпляр образа, инкапсулирует требуемое программное обеспечение. Его можно легко удалить, создать
3. **Порт** – это порт TCP/UDP. Порты могут быть открыты во внешнем мире или подключены к контейнерам (доступны только из этих контейнеров и невидимы для внешнего мира).

# Деплой: Docker

---



Dockerfile:

```
FROM centos:7  
  
RUN yum -y install gpu_backend-20200604.1449-1.el7  
  
CMD ["/usr/local/gpu/gpubackend", "-c", "/etc/gpubackend.conf"]
```

Построение образа

```
docker build -f Dockerfile -t gpubackend:1 .
```

Запуск контейнера

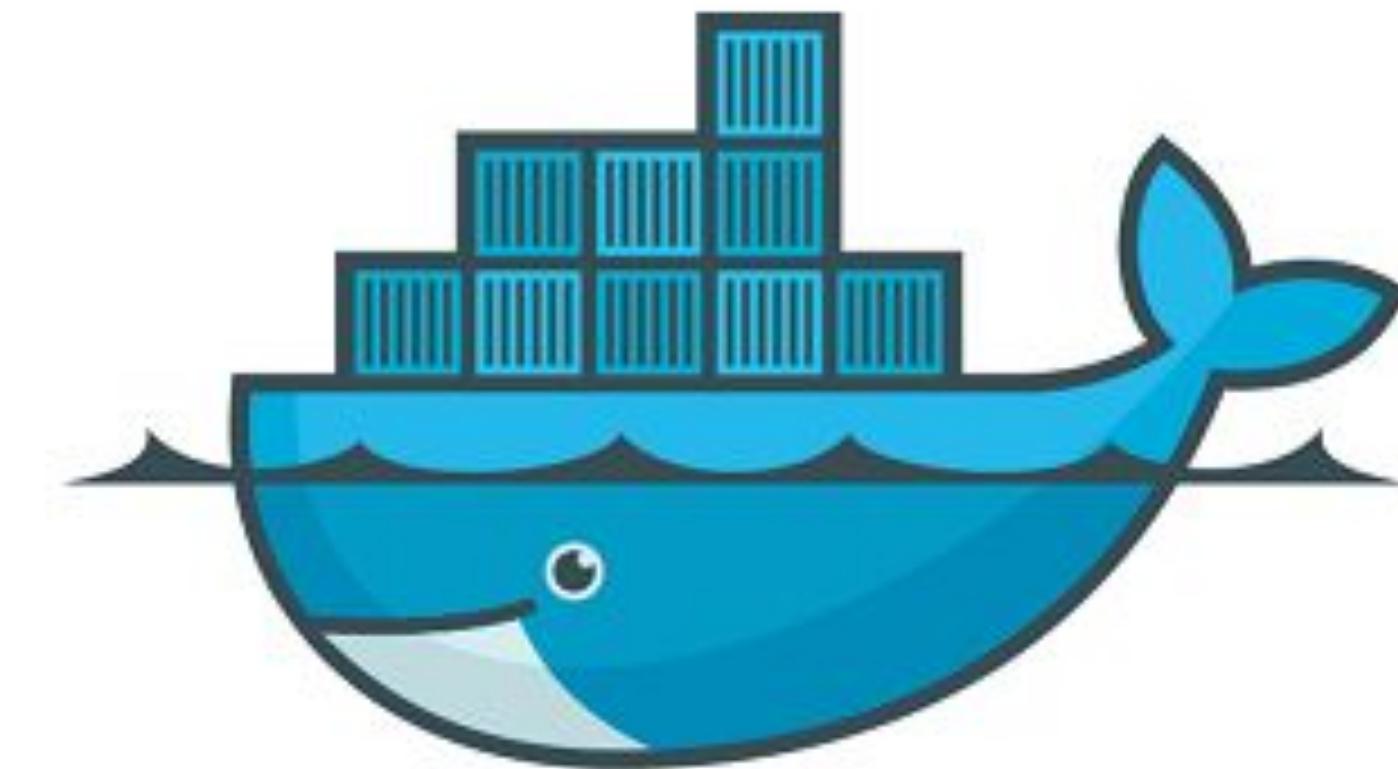
```
docker run -d gpubackend:1
```

# Деплой: Docker с GPU

@

## Требования

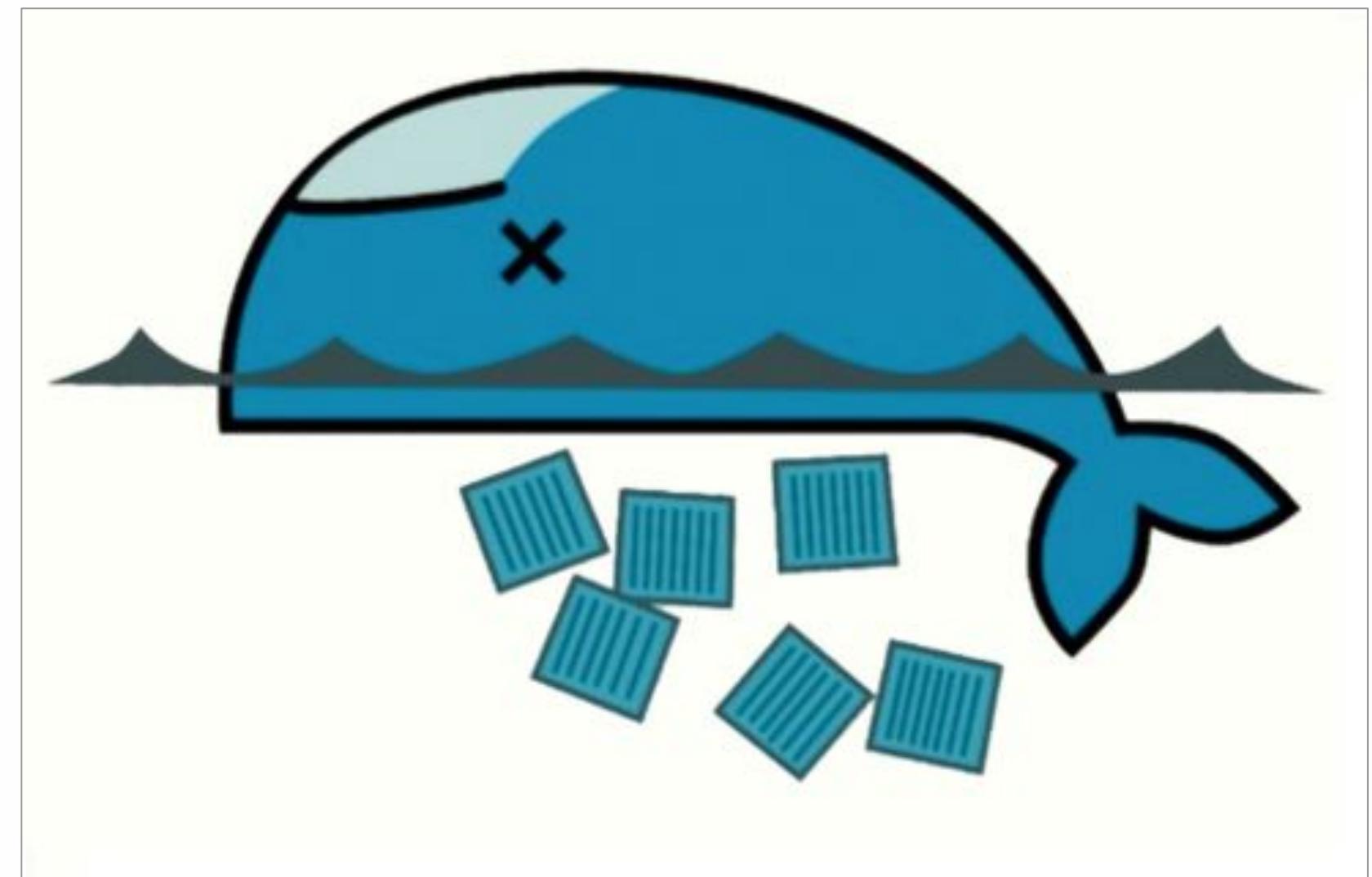
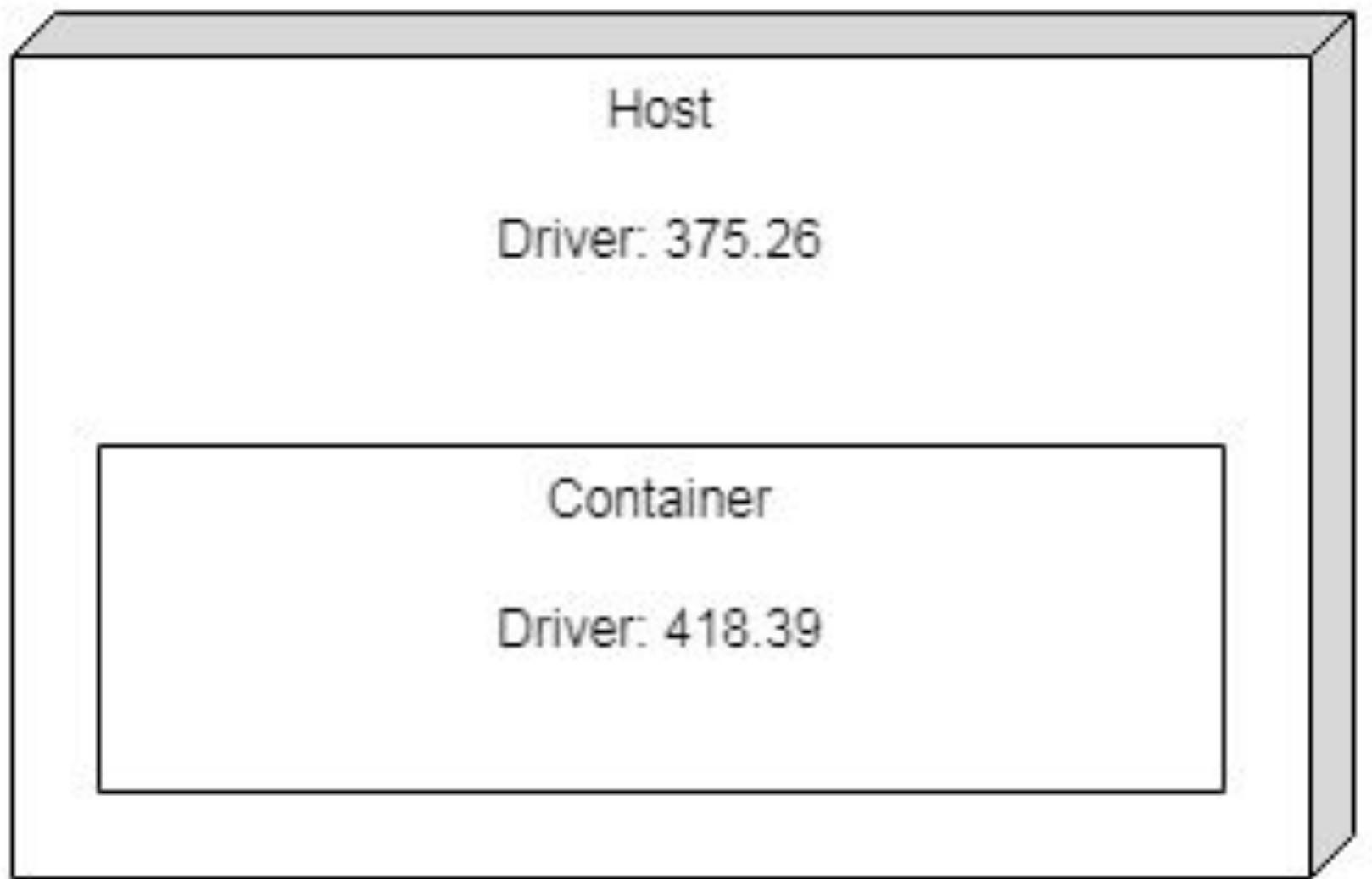
- драйверы на хост-машине
- драйверы внутри контейнера



```
v.mogilin@ml-1gpu-dev-6:~ $ sudo docker run -it --rm --device /dev/nvidia0:/dev/nvidia0 \
> --device /dev/nvidiactl:/dev/nvidiactl \
> --device /dev/nvidia-uvm:/dev/nvidia-uvm \
> simple_docker_ubuntu16_drivers_384.130_work /bin/bash
root@c5657430c349:/#
```

# Деплой: Docker с GPU

@



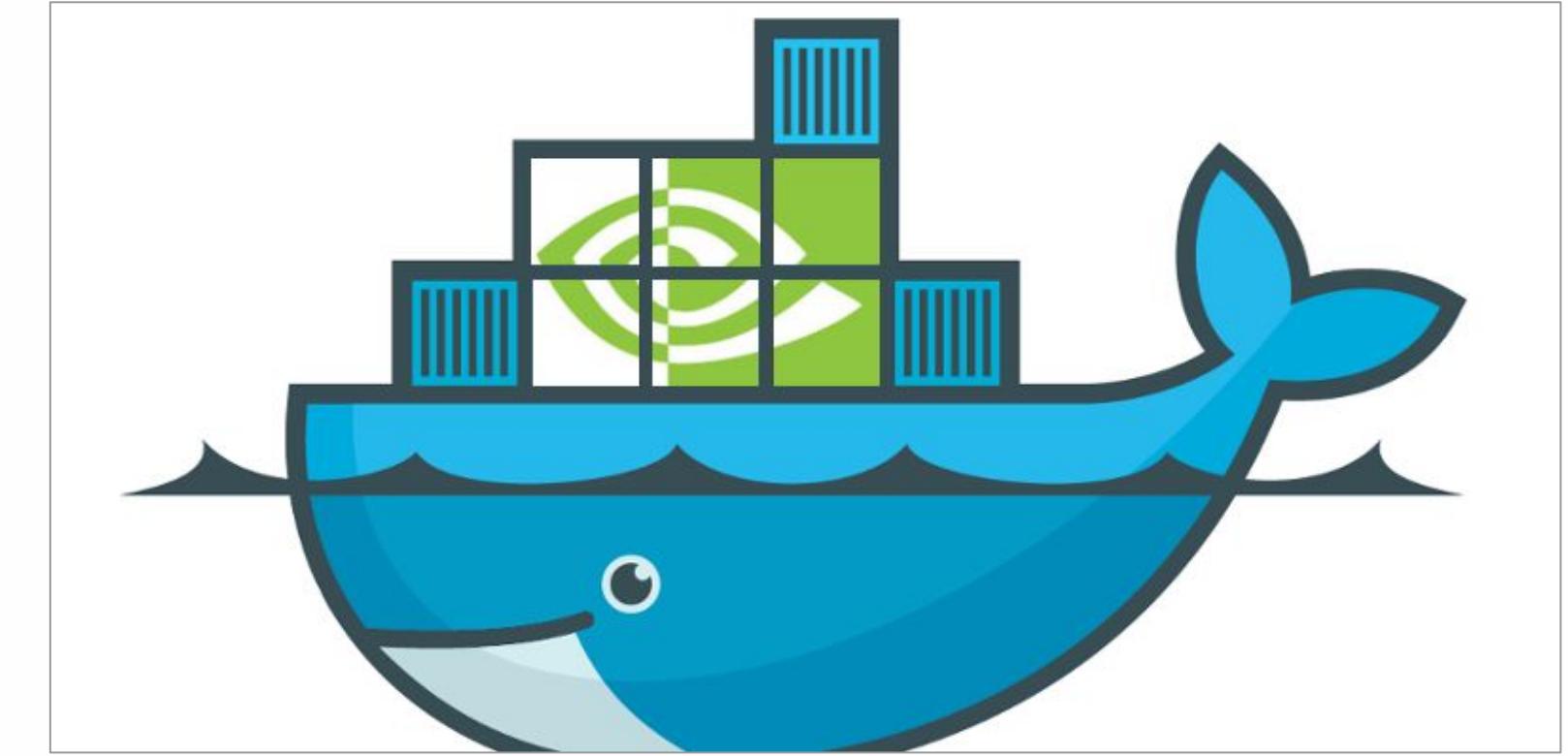
```
root@c5657430c349:#
root@c5657430c349:# nvidia-smi
Failed to initialize NVML: Driver/library version mismatch
root@c5657430c349:#
```

# Деплой: nvidia-docker



## Требования

- драйверы на хост-машине
- docker >= 1.12



gpu workload: overhead <= 1%

```
v.mogilin@ml-1gpu-dev-6:~ $ sudo nvidia-docker run -it --rm nvidia/cuda:8.0-devel-centos7 /bin/bash
[root@a80017dbdf29 /]# nvidia-smi
Thu Mar 21 15:12:44 2019
+-----+
| NVIDIA-SMI 375.26                 Driver Version: 375.26 |
+-----+
```

# Деплой: Kubernetes



@

# kubegnetes

**Kubernetes** - инструмент для автоматизации развёртывания, масштабирования контейнеризированных приложений и управления ими

**Node**: Нода это машина в кластере Kubernetes.

**Pod**: Pod это группа контейнеров с общими разделами, запускаемых как единое целое.

**Replication Controllers**: replication controller гарантирует, что определенное количество «реплик» pod'ы будут запущены в любой момент времени.

**Labels** : Label'ы это пары ключ/значение которые прикрепляются к объектам, например pod'ам. Label'ы могут быть использованы для создания и выбора наборов объектов.



# Деплой: Kubernetes

@

## Плюсы

- autodiscovering
- self-healing
- gpu support  
(v1.8+ device plugin system)



# kubernetes



# Деплой: Kubernetes GPU

@

## Требования

- nvidia-drivers >= 361.93
- nvidia-docker2
- “default-runtime”: “nvidia”  
(/etc/docker/daemon.json)
- device plugin installed



# kubernetes

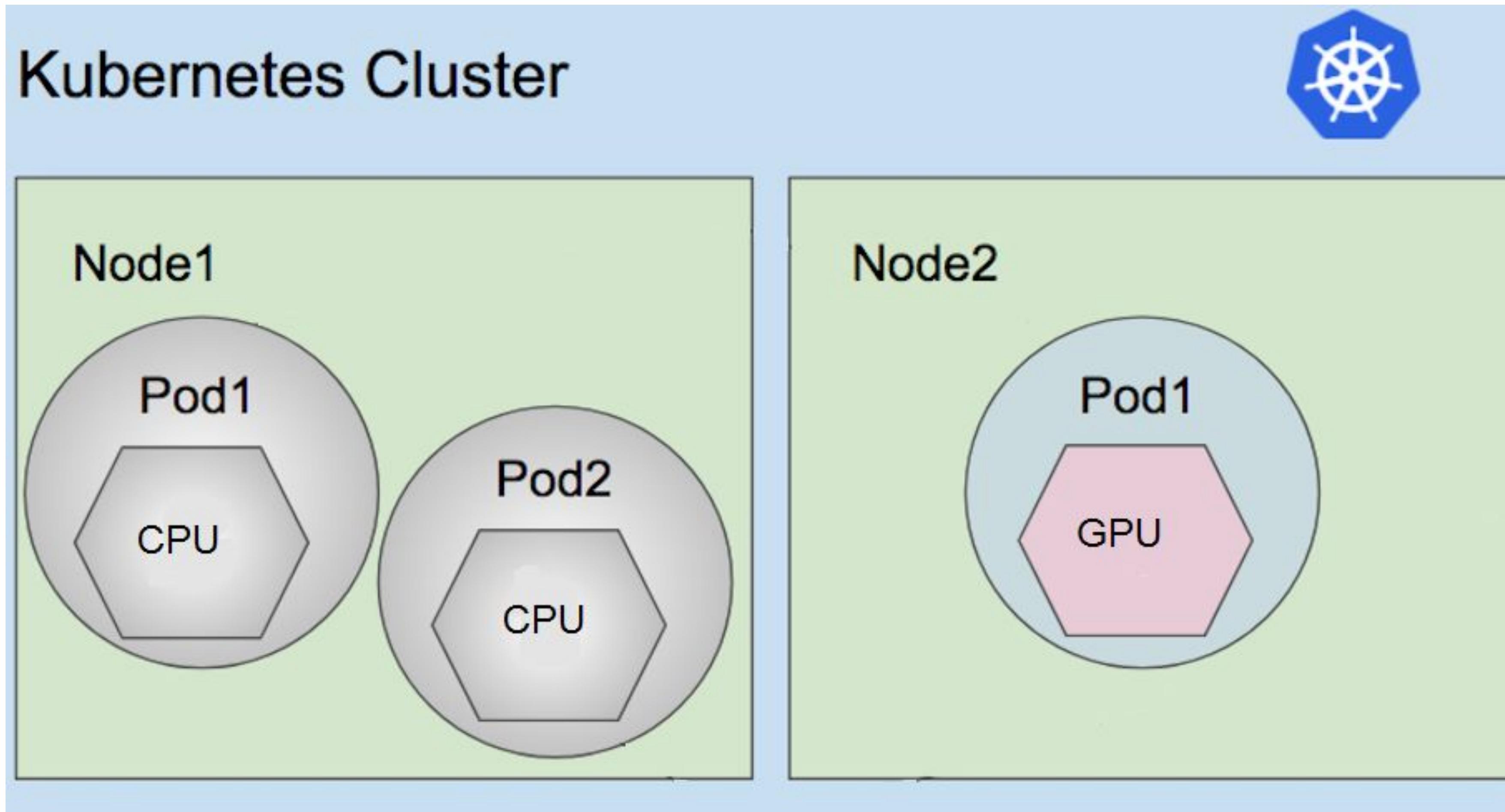
```
kubectl create -f  
https://githubusercontent.com/k8s-device-plugin/v1.12/plugin.yml
```

# Деплой: Kubernetes Pod

@

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: gpubackend-face
    service-name: gpubackend
spec:
  containers:
  - name: gpubackend
    image: registry-gitlab.corp.mail.ru/e-mail-ru/gpubackend:923
    command: ["/root/gpubackend/backend/src/BUILD/gpubackend"]
    args: ["-c", "gpubackend.dev.conf", "--gpu", "0", "--models", "face"]
    imagePullPolicy: IfNotPresent
  resources:
    requests:
      nvidia.com/gpu: "1"
```

# Деплой: Kubernetes Pod



# Деплой: Kubernetes Pod

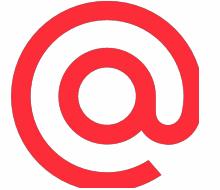
@

**POD**

1 Инстанс Приложения

1 GPU = 1 Model

# Деплой: Kubernetes Pod



Количество pod'ов = Количество видеокарт

POD

1 Инстанс Приложения

1 GPU = 1 Model

POD

1 Инстанс Приложения

1 GPU = 1 Model



POD

1 Инстанс Приложения

1 GPU = 1 Model

# Деплой: Kubernetes Deployment

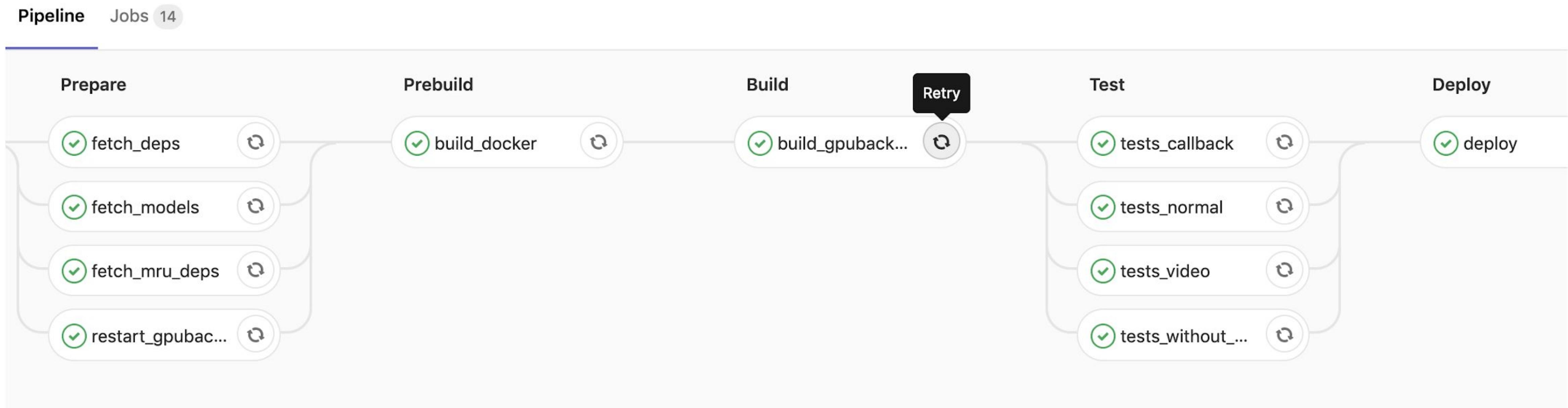


- scale (replicas: n)



# Деплой: Kubernetes Deployment

@

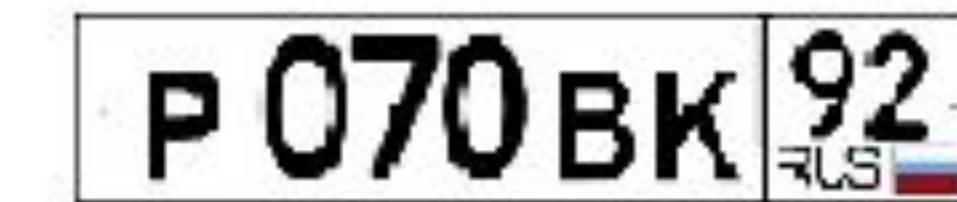
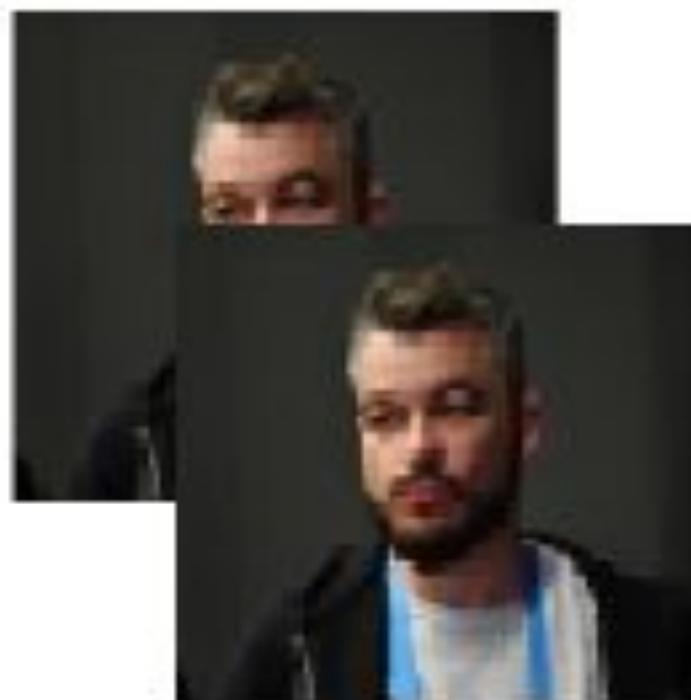


# Масштабирование:

@

Память GPU: ~12 Gb

Модель: ~ 5 Gb



GPU  
face

GPU  
face

GPU  
object

GPU  
car

# Масштабирование:

@



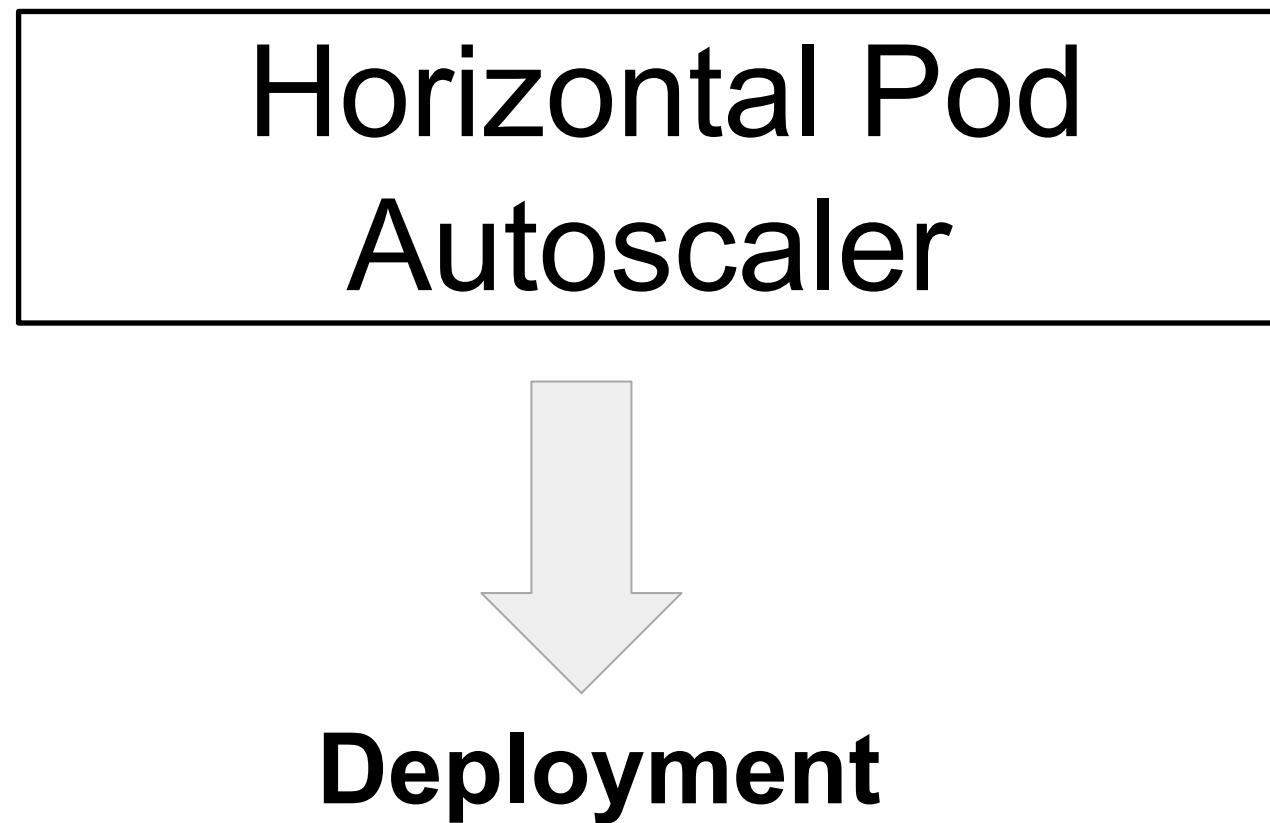
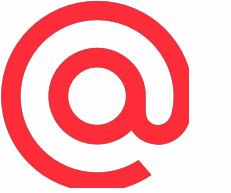
GPU  
face

GPU  
face

GPU  
object

GPU  
car

# Масштабирование



POD

1 Инстанс Приложения

1 GPU = 1 Model

POD

1 Инстанс Приложения

1 GPU = 1 Model

POD

1 Инстанс Приложения

1 GPU = 1 Model

# Расчет количества pod'ов

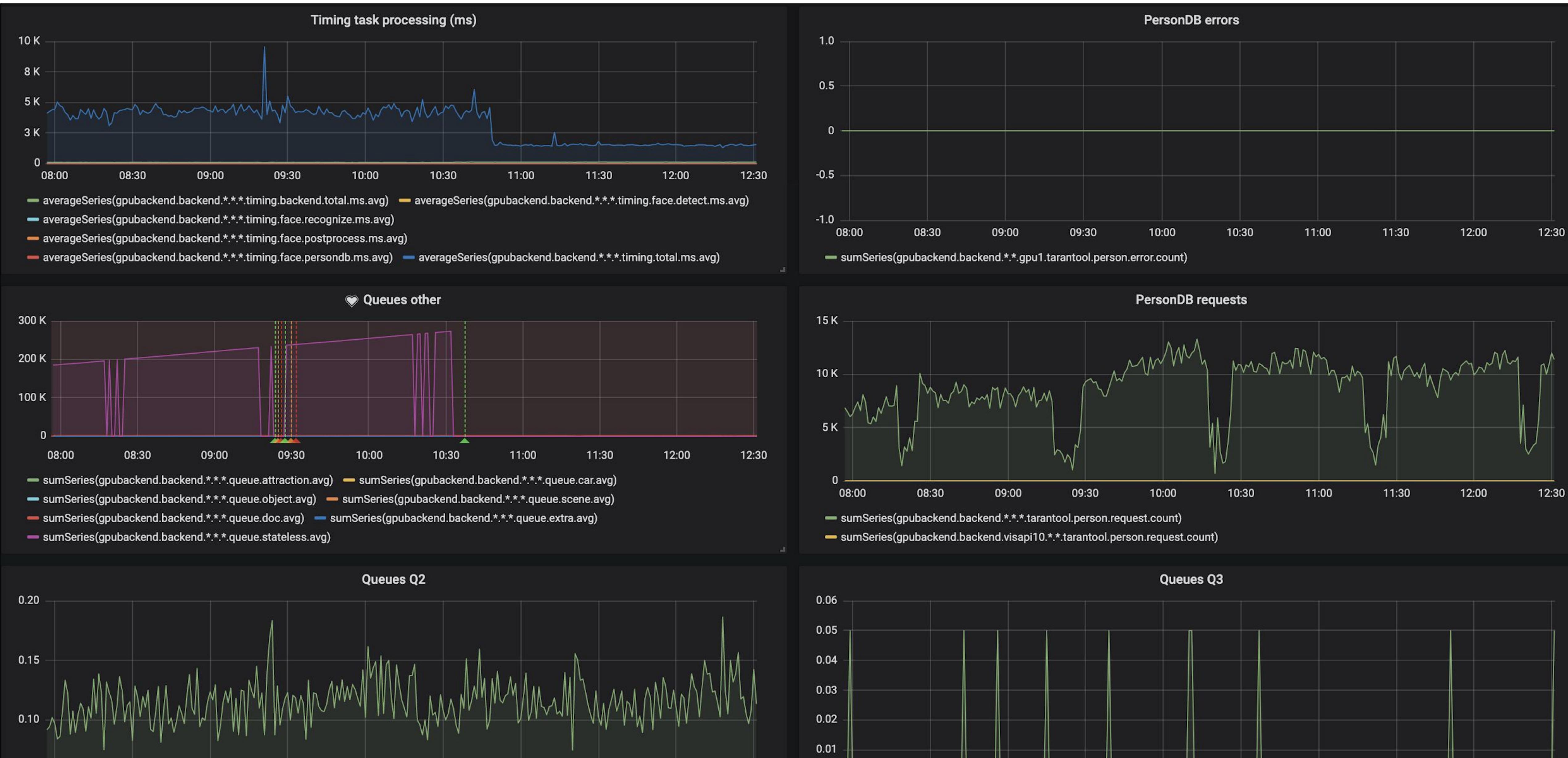
@

```
spec:  
    maxReplicas: 30  
    minReplicas: 1  
    metrics:  
        - type: Object  
          object:  
              metricName: queue_size_face  
              targetValue: 3000  
        target:  
            apiVersion: v1  
            kind: Service  
            name: tarantool-queue-size
```

```
replicas = ceil(curReplicas * (curMetricValue / desiredMetricValue))
```

# Мониторинг

# Эксплуатация: Мониторинг



# Примеры

# Computer Vision API



## Распознавание:

- лиц
- объектов на изображении
- документов (OCR)
- автономеров

## Обработка изображений:

- раскрашивание фотографий
- повышение разрешения



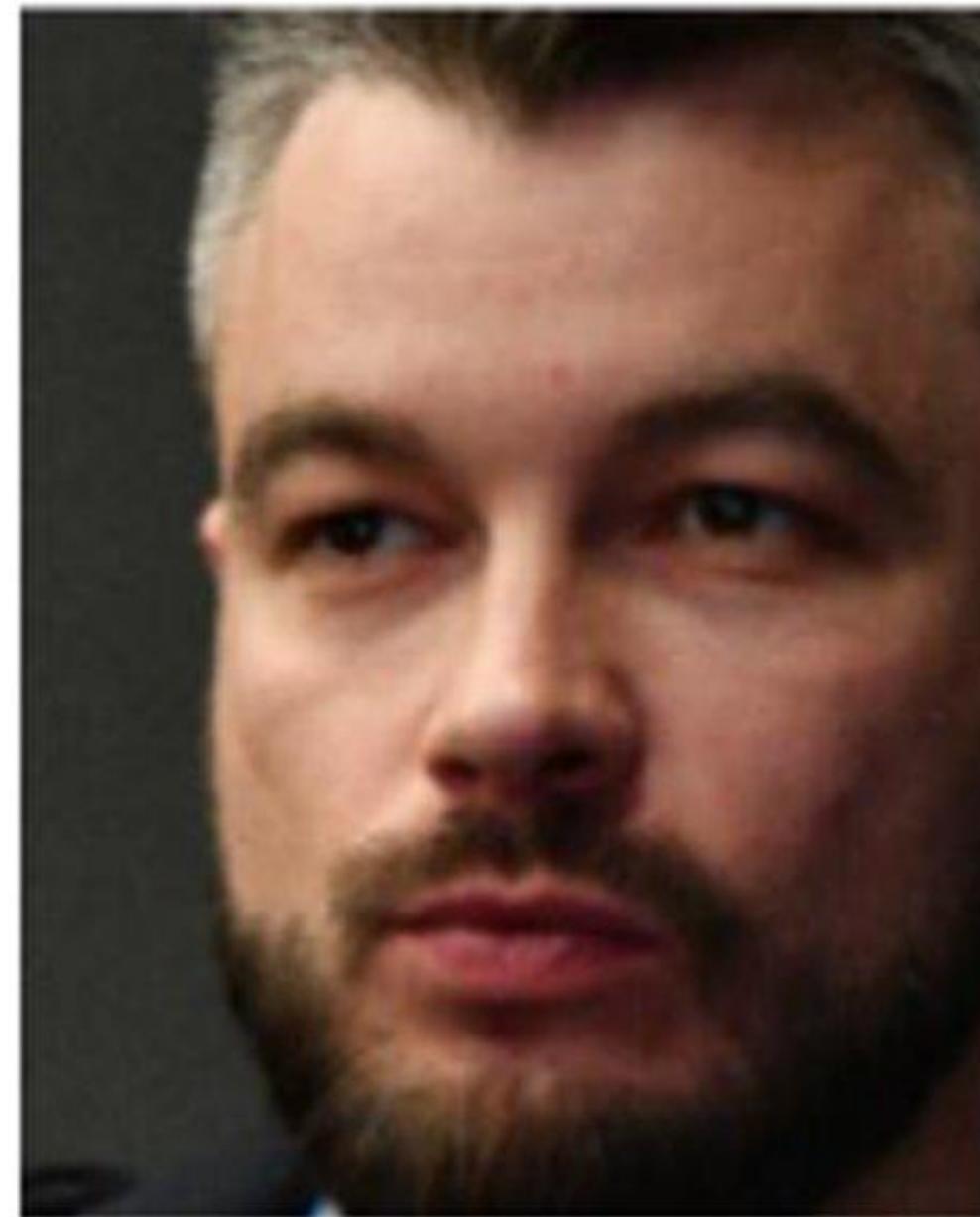
# Распознавание объектов



```
{  
    "eng":"Cat",  
    "eng_categories":["Animals","Cats"],  
    "prob":0.746,  
    "coord":[0,22,780,803]  
}
```

# Распознавание лиц

@



```
{  
  "persons":  
  [  
    {"tag":"person42",  
     "coord": [264, 84, 392, 269]  
    }]  
}
```



# MCS

---



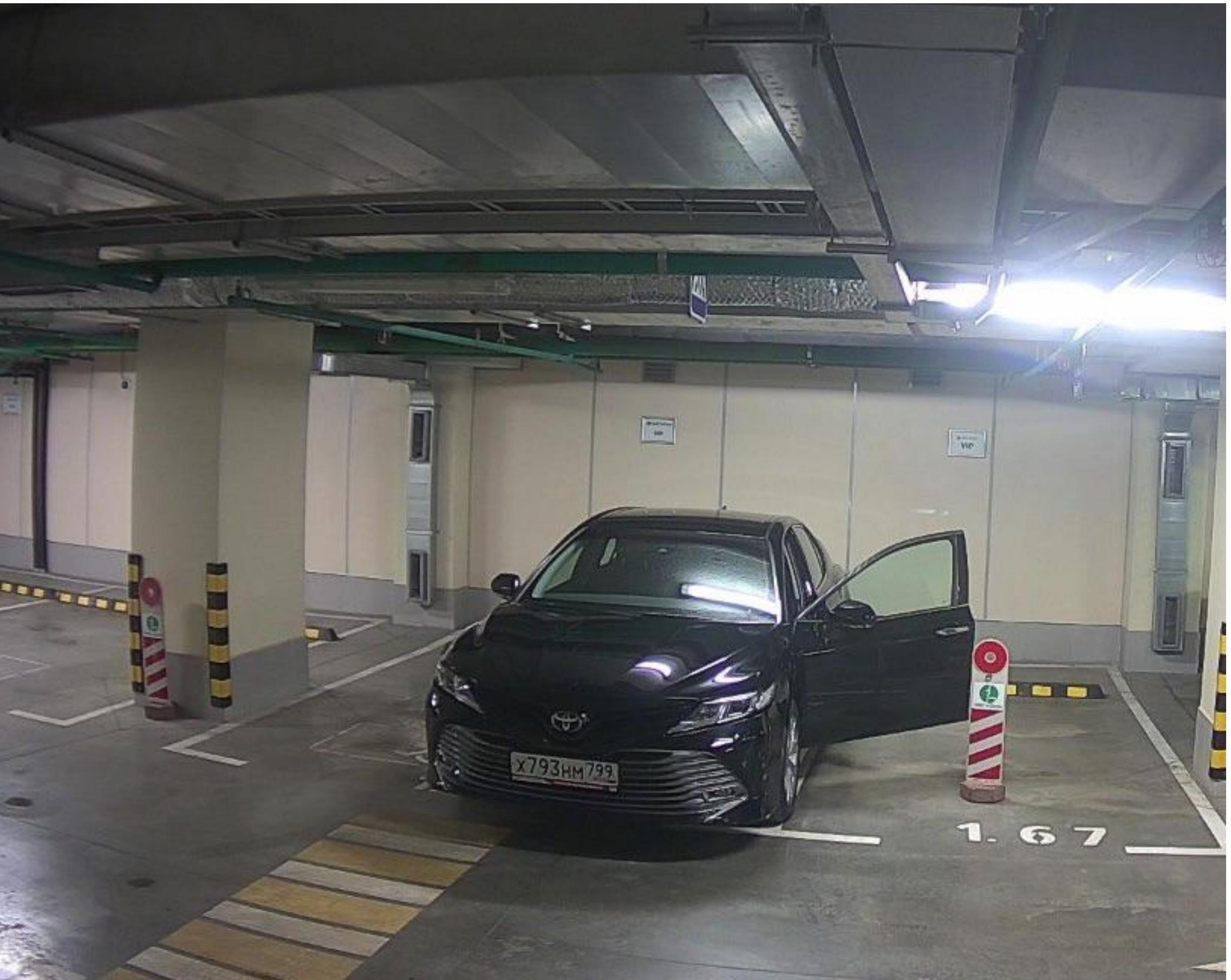
mail.ru  
cloud  
solutions

Запрос:

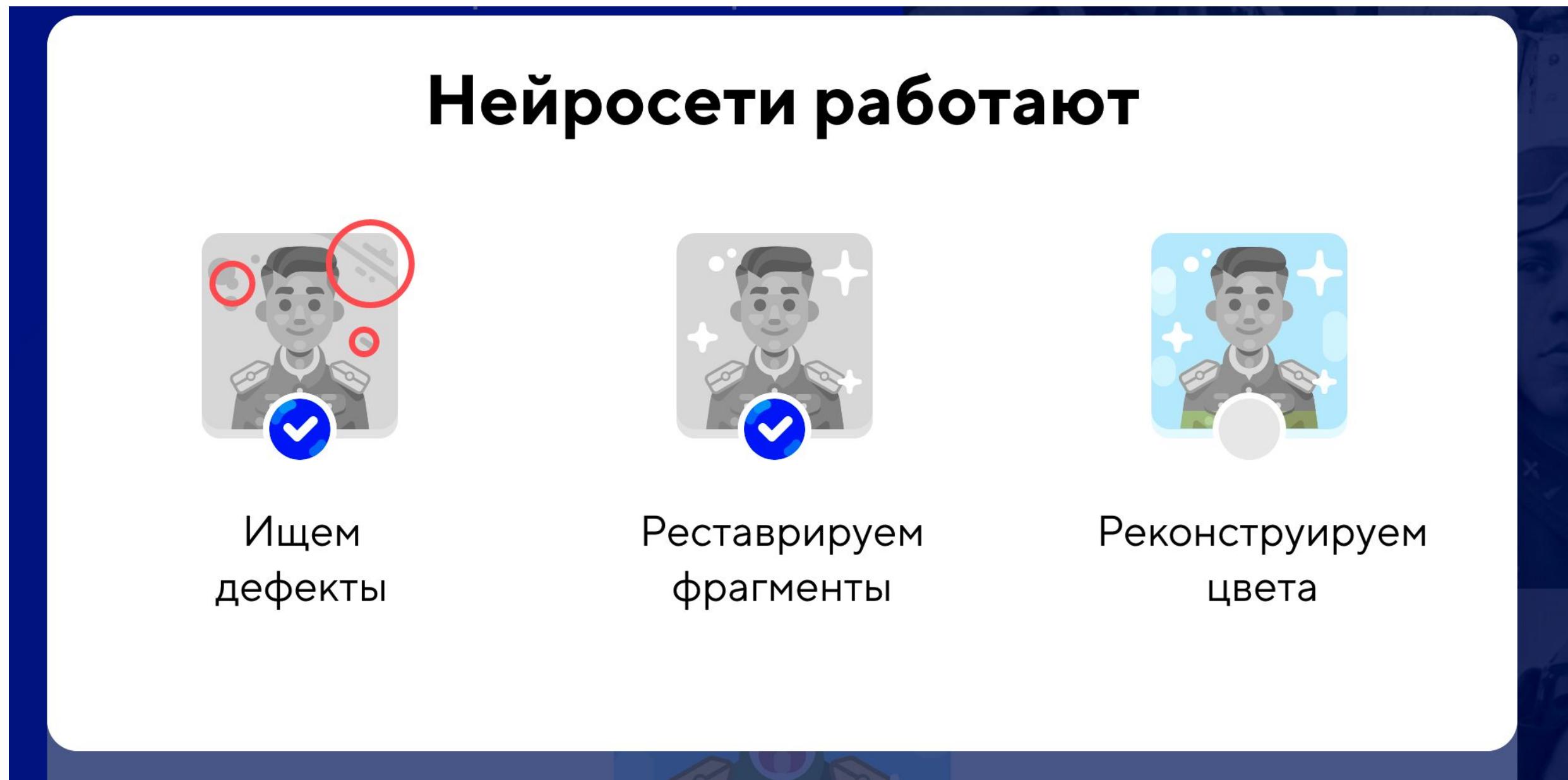
```
curl -k -v  
"https://smarty.mail.ru/api/v1/objects/detect?oauth_provider=mcs&oauth_to  
ken=26nWSke41J6GztH7J" -F file_0=@car.jpg  
-F meta='{"images": [{"name": "file_0"}], "mode": ["car_number"]}'
```

Ответ:

```
{"status":200,"body":{"car_number_labels": [  
 {"status":0,"name":"file_0","labels": [  
 {"eng":"X793HM799","rus": "  
 X793HM799","prob":0.9938,"coord":[337,494,413,520],  
 "types_prob": [  
 {"type":"ru","prob":0.9998},  
 {"type":"cis","prob":0.9989},  
 {"type":"eu","prob":0.0018}  
 ]}]]}}, "htmlencoded":false, "last_modified":0}
```



# Окрашивание фотографий



<https://9may.mail.ru/restoration/>

@ mail Ru

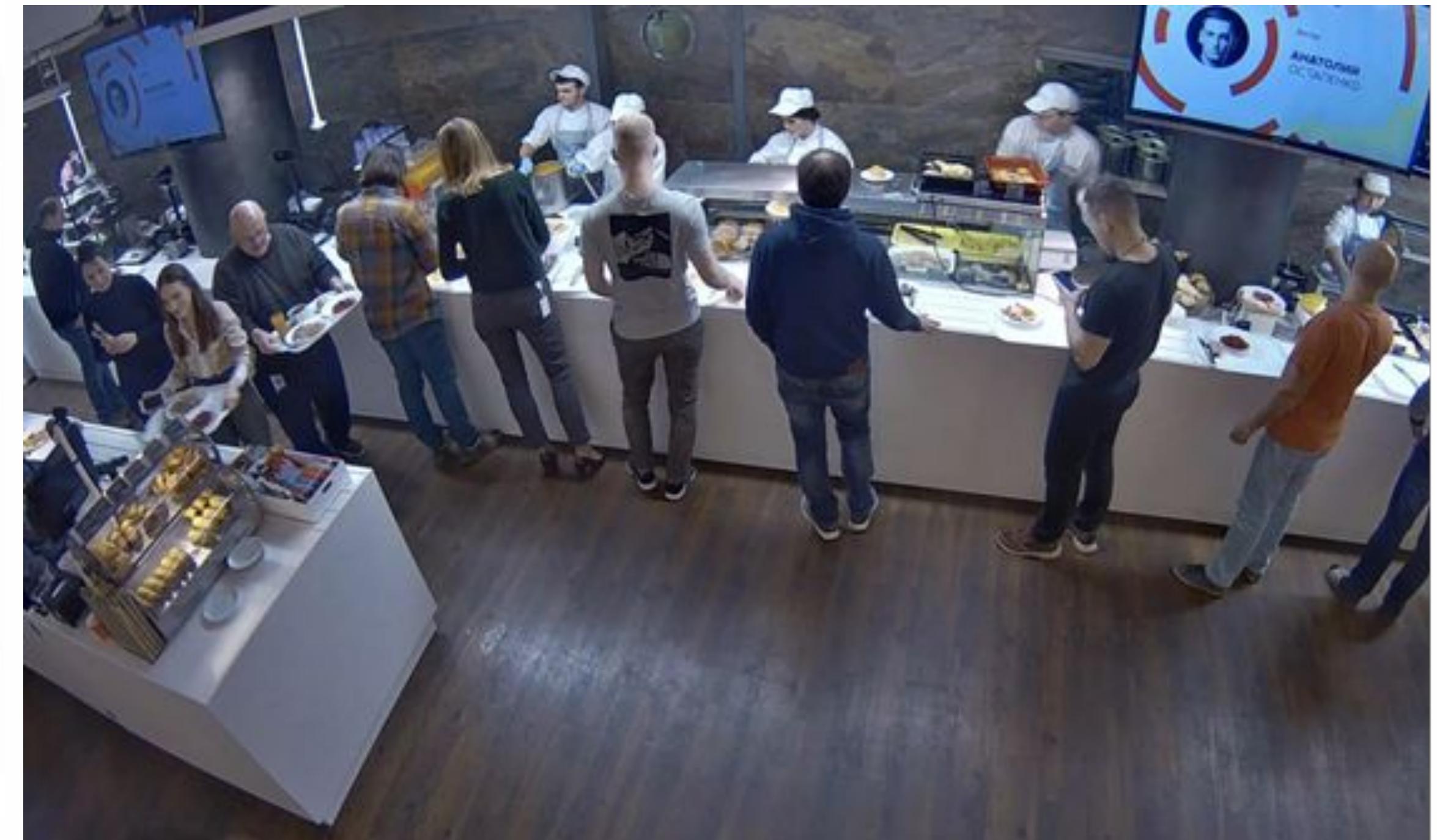
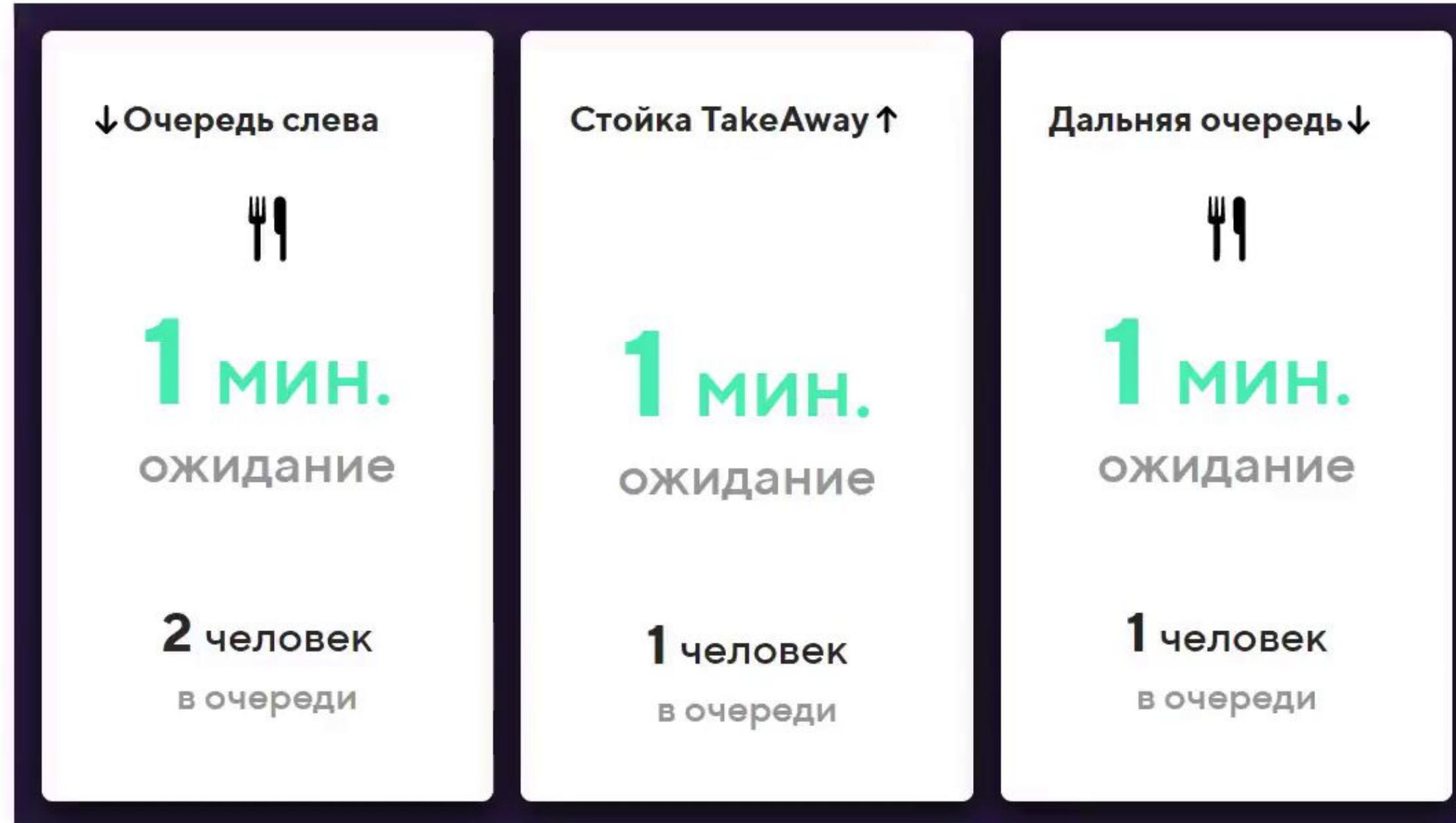
Фотография обработана

Отправить на почту

В Облако

Скачать

# Длина очереди



Вопросы?