

Докупать нельзя доиспользовать:

как AI-ассистент дорос до production

deckhouse

Александр Подмосковный

директор продуктового
направления ML/AI/GPU, «Флант»

- Kubernetes и DevOps — с 2020 года
- ИИ-инфраструктура — с 2024 года
- Собственный кластер для экспериментов



Модели обновляются постоянно

Бизнес хочет использовать актуальные – быстро проверять и запускать их в production.

И подключать к внутренней базе знаний.

Текущая модель

модель уже
в production

Новая модель

быстро запускаем
в production

База знаний

Первый вариант — под одну основную модель

**Основная
модель**

пишет ответ

Первый вариант – под одну основную модель

**Основная
модель**

пишет ответ

Эмбеддер

ищет похожие
фрагменты

Первый вариант — под одну основную модель

**Основная
модель**

пишет ответ

Эмбеддер

ищет похожие
фрагменты

Реранкер

сортирует
найденное

Первый вариант – под одну основную модель

**Основная
модель**

пишет ответ

Эмбеддер

ищет похожие
фрагменты

Реранкер

сортирует
найденное

API

отдаёт ответ
со ссылкой
на источник

**На стенде всё работает.
Сервис открыли всей компании**

В production сервис тормозит

01 Документы стали длиннее

Схема не изменилась

Основная
модель

Эмбеддер

Реранкер

API

В production сервис тормозит

- 01 Документы стали длиннее
- 02 Сессии пошли параллельно

Схема не изменилась

Основная
модель

Эмбеддер

Реранкер

API

В production сервис тормозит

- 01 Документы стали длиннее
- 02 Сессии пошли параллельно
- 03 Запросы стали сложнее**

Схема не изменилась

**Основная
модель**

Эмбеддер

Реранкер

API

В production сервис тормозит

- 01 Документы стали длиннее
- 02 Сессии пошли параллельно
- 03 Запросы стали сложнее
- 04 Нагрузка стала неравномерной**

Схема не изменилась

**Основная
модель**

Эмбеддер

Реранкер

API

Время до первого токена: p95 — до 18 секунд

Time to First Token, TTFT

Время до первого токена: p95 – до 18 секунд

Один запрос в работе,
до семи – в очереди

Время до первого токена: p95 – до 18 секунд

Один запрос в работе, до семи – в очереди

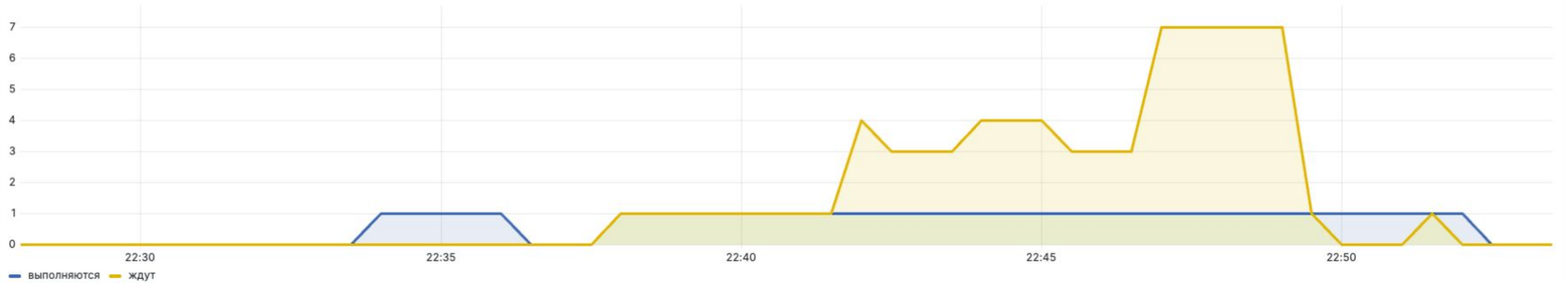
**Память GPU занята, даже когда
нагрузка закончилась**

Очередь тянет за собой время первого токена

Время до первого токена — p95



Запросы в работе и в очереди



Память занята постоянно, вычисления — вспышками

Загрузка GPU — NVIDIA A100



Память GPU, GiB



Докупать, нельзя доиспользовать

**Перед закупкой –
разобраться в ограничениях!**

Один симптом — три уровня проверки

01 Память: помещается ли сервис



Запрос ждёт

очередь не говорит,
чего именно не хватает

Один симптом — три уровня проверки

- 01 Память: помещается ли сервис
- 02 Вычисления: выдерживает ли нагрузку



Запрос ждёт

очередь не говорит,
чего именно не хватает

Один симптом — три уровня проверки

- 01 Память: помещается ли сервис
- 02 Вычисления: выдерживает ли нагрузку
- 03 **Размещение: есть ли подходящий профиль**
Покупка не решит всех проблем — память, вычисления и размещение проверяем отдельно.



Запрос ждёт

очередь не говорит,
чего именно не хватает

Что хотим получить

Актуальные модели – быстро в production,
на текущем парке GPU

Актуальные модели

**Быстро
запускаем**

Новую модель –
в production
без нового
ручного цикла

Контекст

**128–256K
токенов**

Для длинных
документов
и данных
из базы знаний

Рабочая нагрузка

**Сессии работают
одновременно**

TTFT p95 остаётся
в пределах
заданного SLO

GPU-парк

**Сначала
текущий парк**

В бюджет – только
подтверждённый
дефицит

Дальше разберём три ограничения: память, вычисления и размещение

Сначала — что получилось

Затем – как мы к этому пришли

Что нужно знать, чтобы проверить запуск

Модель

архитектура,
веса,
контекст

Что нужно знать, чтобы проверить запуск

Модель

архитектура,
веса,
контекст

Сервис

контекст,
нагрузка,
ограничения

Что нужно знать, чтобы проверить запуск

Модель

архитектура,
веса,
контекст

Сервис

контекст,
нагрузка,
ограничения

Среда

движок
и параметры
запуска

Что нужно знать, чтобы проверить запуск

Модель

архитектура,
веса,
контекст

Сервис

контекст,
нагрузка,
ограничения

Среда

движок
и параметры
запуска

Кластер

GPU, память,
режимы доступа

Сервис инференса управляет запуском

Входные данные

AI-MODELS

Параметры модели

INFERENCESERVICE

**Требования
сервиса**

Планирование

AI-INFERENCE

Планировщик

Подбирает конфигурацию

считает память, проверяет
доступные GPU, объясняет
причину отказа

Исполнение

MAAS

**Запускает
сервис**

GPU DRA

Готовит GPU

До запуска известны
конфигурация, требуемая память
и конкретная причина отказа

Выбираем модель из каталога

Каталог моделей

Найдите модель, скачайте локальную копию и запустите сервис инференса.

Неймспейс: ai-demo · скачано: 4 из 15 · хранилище: 24.7 GB / 2 TB

 openai/gpt-oss-20b

• Уже скачана

Hugging Face · Чат и текст · 21B · 128K

Safetensors MXFP4 12.8 GB

• Запустится

Запустить инференс

 Qwen/Qwen3.6-27B-FP8

• Не скачана

Hugging Face · Чат и текст · 27B · 256K

Safetensors FP8 29 GB

• Оценка после скачивания

Скачать

 zai-org/GLM-5.2-FP8

• Не скачана

Hugging Face · Чат и текст · 753B · 1M

Safetensors FP8 753.4 GB

• Оценка после скачивания

Скачать

 openai/whisper-large-v3

• Не скачана

Hugging Face · Речь в текст · 1.5B

Safetensors FP16 3.1 GB

• Оценка после скачивания

Скачать

 BAAI/bge-m3

• Уже скачана

Hugging Face · Эмбединги · 568M · 8K

Safetensors FP16 4.6 GB

• Запустится

Запустить инференс

 BAAI/bge-reranker-v2-m3

• Не скачана

Hugging Face · Ранжирование · 568M · 8K

Safetensors FP16 2.3 GB

• Оценка после скачивания

Скачать

32

Задаём требования к сервису

- **Стратегия**
Latency / Throughput / Balance

Новый сервис инференса

1 Сервис

2 Конфигурация

3 Подтверждение

Стратегия запуска

Latency

Минимальная задержка ответа — интерактивные сценарии (чат).

Размещение

пусто — автоматически

Автоматически

gpu-80gb

80 GB · целая GPU

доступно

gpu-96gb

96 GB · целая GPU

занято, вытеснение возможно

gpu-40gb-mig

40 GB · MIG

несовместимо

InsufficientMemory

Задаём требования к сервису

- **Стратегия**
Latency / Throughput / Balance
- **Размещение**
автоматически или выбранный GPU

Новый сервис инференса

1 Сервис

2 Конфигурация

3 Подтверждение

Стратегия запуска

Latency

Минимальная задержка ответа — интерактивные сценарии (чат).

Размещение

пусто — автоматически

Автоматически

gpu-80gb

80 GB · целая GPU

доступно

gpu-96gb

96 GB · целая GPU

занято, вытеснение возможно

gpu-40gb-mig

40 GB · MIG

несовместимо

InsufficientMemory

Задаём требования к сервису

- **Стратегия**
Latency / Throughput / Balance
- **Размещение**
автоматически или выбранный GPU
- **Контекст и реплики**
сколько текста держать в памяти

Новый сервис инференса

1 Сервис

2 **Конфигурация**

3 Подтверждение

Стратегия запуска

Latency

Минимальная задержка ответа — интерактивные сценарии (чат).

Размещение

пусто — автоматически

Автоматически

gpu-80gb

80 GB · целая GPU

доступно

gpu-96gb

96 GB · целая GPU

занято, вытеснение возможно

gpu-40gb-mig

40 GB · MIG

несовместимо

InsufficientMemory

Задаём требования к сервису

- **Стратегия**
Latency / Throughput / Balance
- **Размещение**
автоматически или выбранный GPU
- **Контекст и реплики**
сколько текста держать в памяти
- **До запуска**
вариант запуска или причина отказа

Новый сервис инференса

1 Сервис

2 Конфигурация

3 Подтверждение

Стратегия запуска

Latency

Минимальная задержка ответа — интерактивные сценарии (чат).

Размещение

пусто — автоматически

Автоматически

gpu-80gb

80 GB · целая GPU

доступно

gpu-96gb

96 GB · целая GPU

занято, вытеснение возможно

gpu-40gb-mig

40 GB · MIG

несовместимо

InsufficientMemory

До запуска известны конфигурация и GPU-профиль

- Конфигурация запуска
- Ресурсы сервиса
- Класс устройства и режим
- Параметры выдачи

Новый сервис инференса

1 Сервис

2 Конфигурация

3 Подтверждение

qwen3-6-27b

Класс сервиса
llm-chat-dedicated

gpu-80gb

qwen3-6-27b

Стратегия запуска
Latency

```
apiVersion: ai.deckhouse.io/v1alpha1
kind: InferenceService
metadata:
  name: qwen3-6-27b
  namespace: ai-demo
spec:
  inferenceServiceClassName: llm-chat-dedicated
  launchStrategy: Latency
  model:
    ref:
      kind: Model
      name: qwen3-6-27b
    src: ai-models
  resources:
    accelerator:
      count: 1
      deviceClasses:
        - gpu-80gb
```

Сегодня разберём планировщик

Как он считает ресурсы, проверяет доступные GPU и формирует конфигурацию запуска

01

Считает память

веса, KV-кеш, среда и запас

02

Проверяет доступные GPU

несовместимые — с причиной отказа

03

Формирует конфигурацию запуска

GPU или MIG; совместный запуск — MPS

Результат:

движок, настройки запуска
и конфигурация GPU

Память

Сначала считаем весь бюджет GPU-памяти

Память: файл весов — только часть бюджета

`model.safetensors`

файл весов ≈ 29 GB



KV-кеш, память vLLM и резерв в файл не входят

К весам добавляется KV-кеш

`model.safetensors`

файл весов ≈ 29 GB



Чем длиннее контекст и больше параллельных сессий, тем больше KV-кеш

Как модель собирает ответ

На каждом шаге модель
выбирает один следующий токен

Как модель собирает ответ

На каждом шаге модель
выбирает один следующий токен

Вопрос: почему робот-пылесос уволился?

Как модель собирает ответ

На каждом шаге модель
выбирает один следующий токен

Вопрос: почему робот-пылесос уволился?

Ответ появляется за пять шагов

1

Надоело

2

снова

3

начинать

4

с

5

начала

Как модель собирает ответ

На каждом шаге модель
выбирает один следующий токен

Перед выбором очередного токена модель учитывает
вопрос и всё уже написанное начало

Пять токенов ответа — пять отдельных шагов генерации

Что модель делает **на одном шаге**

Последний шаг: модель
выбирает слово «начала»

Контекст этого шага

Вопрос + «Надоело снова начинать с»

Q

**Запрос
текущей позиции**

Что сейчас нужно найти?

K

**Ключ прошлого
токена**

Где искать?

V

**Значение
прошлого токена**

Что взять?

K и V сохраняются в KV -кеше.
Новый Q считается на каждом шаге.

KV-кеш сокращает вычисления: 25 → 7 позиций

Без KV-кеша

$3 + 4 + 5 + 6 + 7 = 25$ позиций

С KV-кешем

$3 + 1 + 1 + 1 + 1 = 7$ позиций

Вопрос: «Почему / робот-пылесос / уволился?» – 3 токена

Ответ: «Надоело / снова / начинать / с / **начала**» – 5 токенов

Меньше работы — не значит в 3,6 раза быстрее

Расчёт

25 → 7 позиций

Реальная скорость

**зависит от модели,
движка и GPU**

Архитектура внимания определяет структуру KV-кеша

Одни модели считаются
по числу KV-голов,
другие — по типам слоёв

По числу KV-голов

Llama 2 7B

MHA (Multi-Head Attention)
K/V для каждой головы

MiniMax-M2.1, gpt-oss

GQA (Grouped-Query Attention)
K/V на группу голов

Falcon 7B

MQA (Multi-Query Attention)
одни K/V на все головы

По типам слоёв

Qwen3.6-27B

Gated DeltaNet + GQA
(гибрид из двух типов слоёв)
KV-кеш только в 16 GQA-слоях

DeepSeek-R1, Kimi-K2.5

MLA (Multi-head Latent Attention)
сжатое состояние вместо
K/V по головам

GLM-5.3

DSA + IndexShare
(разреженное внимание)
не сводится к числу KV-голов

KV-кеш считаем по архитектуре модели и запуску

Параметры модели
задают байты на токен;
параметры запуска —
сколько токенов хранить

$$KV = 2 \times L \times T \times N_{kv} \times D \times E$$

Для обычного внимания (MHA / GQA / MQA)

Из конфигурации модели

L — слоёв с KV-кешем; N_{kv} — наборов K/V в слое; D — элементов в KV-голове

Из конфигурации запуска

T — токенов в KV-кеше; E — байт на элемент кеша; BF16 / FP16 $\rightarrow E = 2$

Почему здесь $\times 2$

отдельно храним K и V для батча T — сумма токенов во всех активных запросах

$$KV = L \times T \times (R_{kv} + Drope) \times E$$

Для MLA (Multi-head Latent Attention)

R_{kv} — сколько чисел занимает сжатая пара K/V

$Drope$ — сколько чисел занимает позиционная часть ключа

L , T и E — те же; отдельного $\times 2$ за K и V нет

$$KV = 2 \times L \times T \times H_{kv} \times D \times E$$

Qwen3.6-27B

Как устроена модель

16 GQA-слоёв с KV-кешем
объём растёт с каждым
токеном

48 слоёв Gated DeltaNet
фиксированное состояние

$$KV = 2 \times L \times T \times H_{kv} \times D \times E$$

Qwen3.6-27B

Как устроена модель

16 GQA-слоёв с KV-кешем
объём растёт с каждым
токеном

48 слоёв Gated DeltaNet
фиксированное состояние

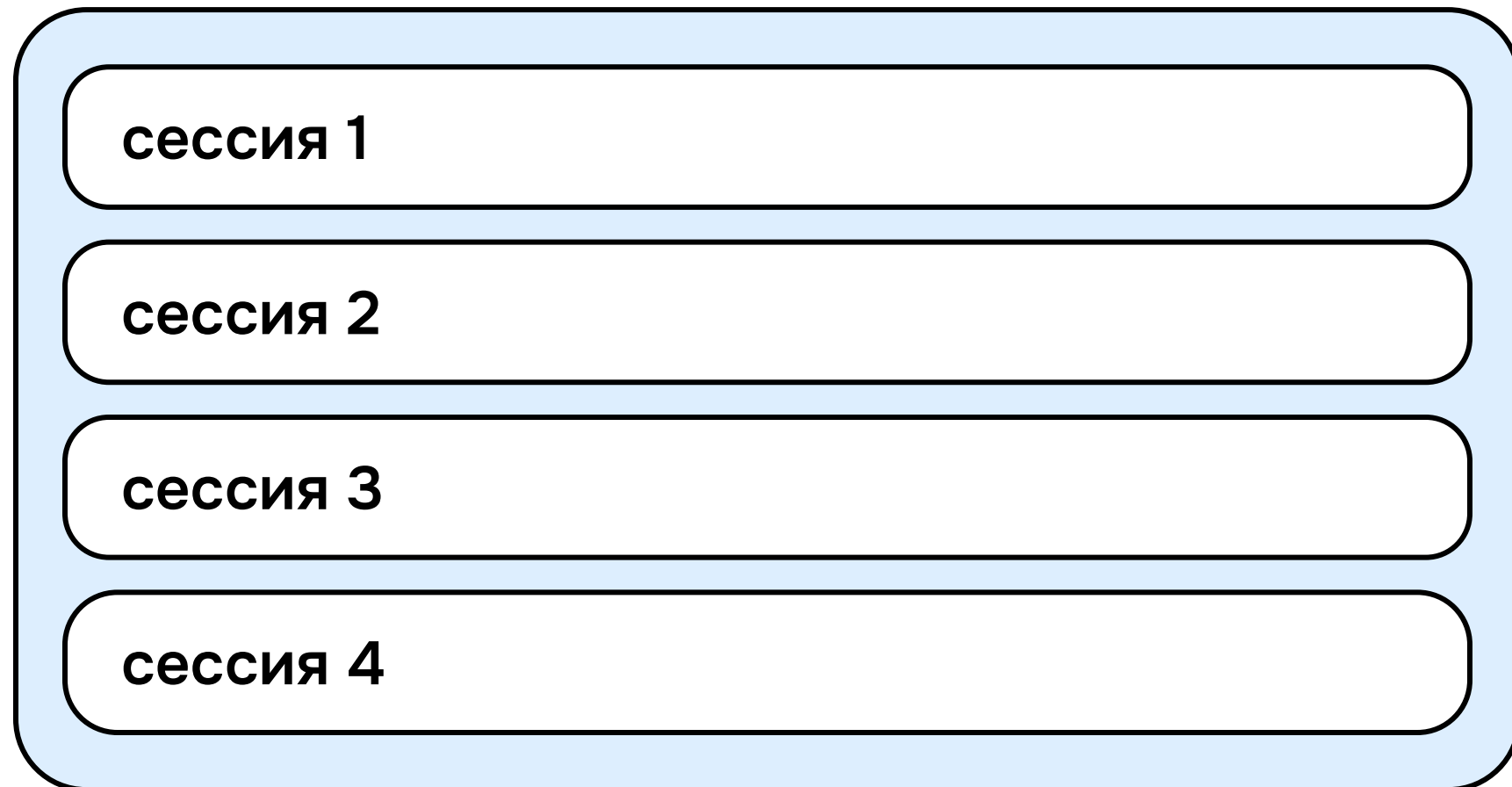
$$2 \times 16 \times 4 \times 256 \times 131\,072 \times 2 = 8,0 \text{ GiB}$$

Это только KV-кеш.

Веса, фиксированное состояние,
vLLM и резерв добавляем
в полный бюджет.

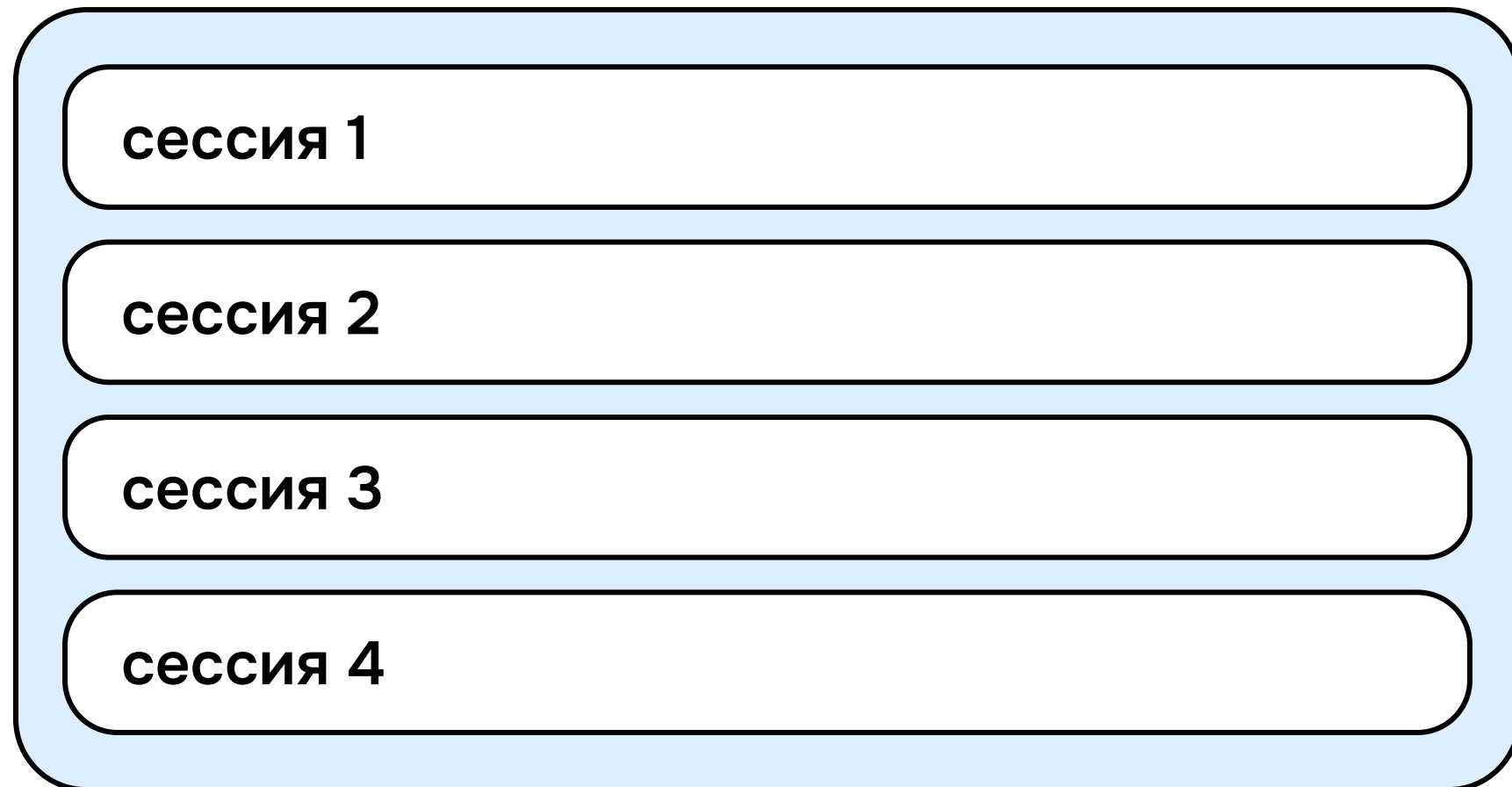
Один пул кеша: много коротких или одна длинная

Несколько коротких сессий



Длинный контекст вытесняет параллельные сессии

Несколько коротких сессий



Одна длинная сессия



Добавляем память vLLM и резерв на пик

`model.safetensors`

файл весов ≈ 29 GB

Бюджет запуска

Веса

KV-кеш

Среда

Запас

Итог сравниваем с доступной памятью конкретного
профиля GPU

Бюджет: на запуск и на сессию

Qwen3.6-27B, профиль 128–256K

веса FP8

KV-кеш при 128K / 256K

состояние линейных слоёв

≈ 29 GB на запуск

8,0 / 16,0 GiB на сессию

на каждую сессию

vLLM и запас

На запуск

Каждая новая GPU – новый ручной цикл

Первая попытка

Движок → параметры → режим GPU → запуск

A100, 40 GB
не поместилось

Каждая новая GPU – новый ручной цикл

Первая попытка

Движок → параметры → режим GPU → запуск

A100, 40 GB
не поместилось

Вторая попытка

Снова: движок → параметры → режим GPU → запуск

H100, 80 GB
повторяем всё вручную

Каждая новая GPU – новый ручной цикл

Первая попытка

Движок → параметры → режим GPU → запуск

A100, 40 GB
не поместилось

Вторая попытка

Снова: движок → параметры → режим GPU → запуск

H100, 80 GB
повторяем всё вручную

Следующая карта

тот же ручной цикл

Смена карты не отменяет подбор –
она запускает его заново

Повторяющееся начало берётся из кеша

Запрос А

Общий документ — считаем

Вопрос А

Повторяющееся начало берётся из кеша

Запрос А

Общий документ — считаем

Вопрос А

Запрос В

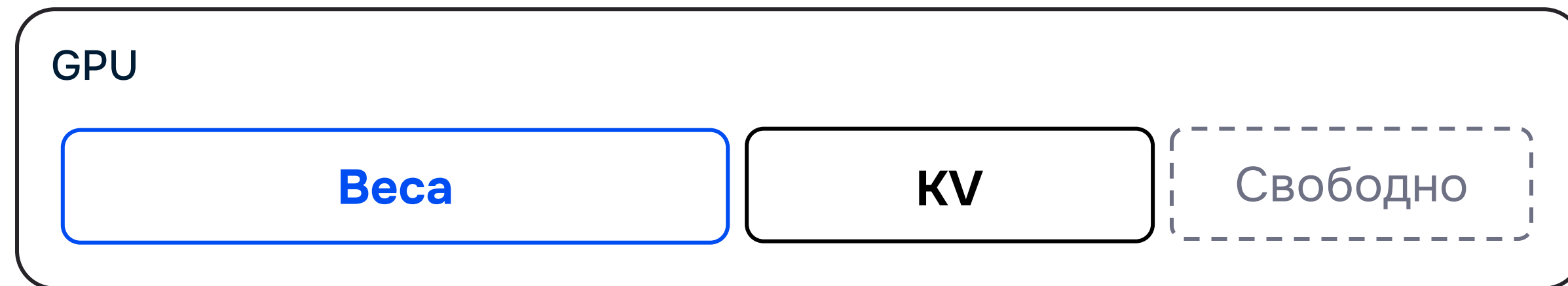
Общий документ — уже в кеше

Вопрос В

Считается заново только
изменившаяся часть

Часть KV-кеша можно вынести в оперативную память

Это освобождает память GPU, но добавляет обмен по PCIe



Часть KV-кеша можно вынести в оперативную память

Память GPU освобождается, обмен идёт по PCIe (KV cache offloading)



Пример: 4 GiB / 20–50 GiB/s $\approx$ 80–200 мс

Это нижняя граница только для переноса по PCIe

Память: считаем и оптимизируем

Память

Префиксное кеширование (prefix caching)

Повторяющийся префикс
не считается заново

Выгрузка KV-кеша в RAM (KV cache offloading)

На GPU остаётся больше
места для контекста

Вычисления

Теперь проверяем, выдержит ли сервис
нужную нагрузку

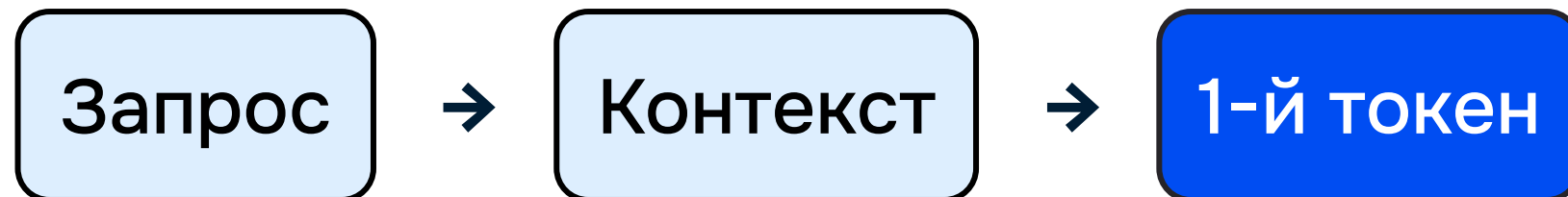
Как модель формирует ответ

До первого токена — читает весь вход

После первого — продолжает по одному

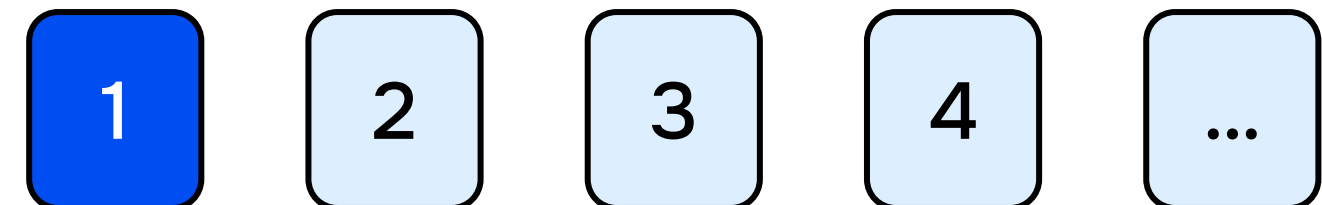
PREFILL

Все токены запроса обрабатываются параллельно



DECODE

Каждый следующий токен зависит от предыдущего



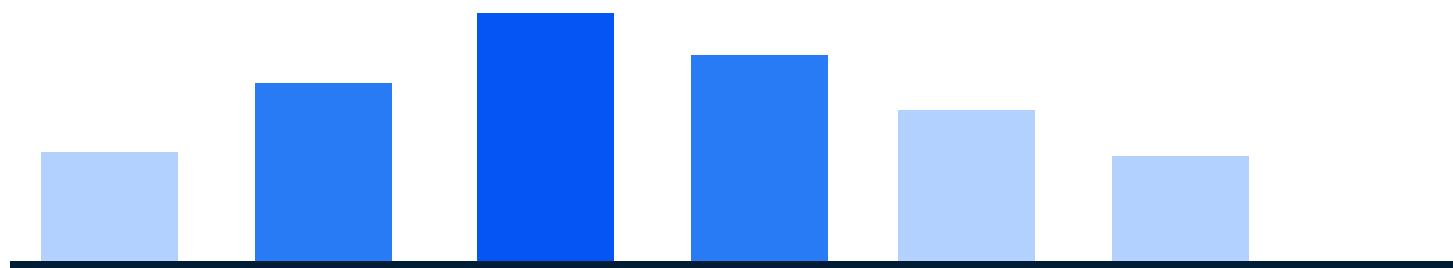
Одна модель делает обе работы

Фазы по-разному нагружают GPU

Чтение входа — широкий всплеск

PREFILL

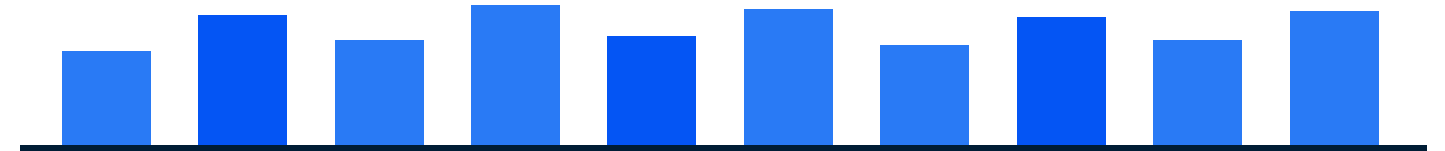
длиннее контекст → больше работы до первого токена



Генерация — много коротких шагов

DECODE

длиннее ответ → больше последовательных проходов



PREFILL и DECODE оптимизируем отдельно

На деле оптимизаций больше

PREFILL: вход делим на части

короткие запросы не ждут длинного
CHUNKED PREFILL

DECODE: проверяем токены

draft-модель предлагает несколько,
основная проверяет **SPECULATIVE
DECODING**

На деле оптимизаций больше

PREFILL: вход делим на части

короткие запросы не ждут длинного
CHUNKED PREFILL

DECODE: проверяем токены

draft-модель предлагает несколько,
основная проверяет **SPECULATIVE
DECODING**

Ещё три механизма уровня сервиса

Непрерывный батчинг (continuous batching)

новые запросы входят
без ожидания всей пачки

PREFILL / DECODE — по разным репликам

Маршрутизация по KV-кешу

запрос идёт туда,
где его кеш уже есть

Размещение

Кому нужна целая GPU, а кто может работать рядом?



Вычисления: у сервисов разный профиль нагрузки

Основная модель



длинный контекст
и строгий TTFT

Вычисления: у сервисов разный профиль нагрузки

Основная модель



длинный контекст
и строгий TTFT

Векторизация



много коротких
запросов

Вычисления: у сервисов разный профиль нагрузки

Основная модель



длинный контекст
и строгий TTFT

Векторизация



много коротких
запросов

Переранжирование



короткие вычислительные
всплески

Вычисления: у сервисов разный профиль нагрузки

Основная модель



длинный контекст
и строгий TTFT

Векторизация



много коротких
запросов

Переранжирование



короткие вычислительные
всплески

Речь и OCR



нагрузка возникает
эпизодически

GPU DRA готовит устройство к запуску

ResourceClaim

**Заявка
на ресурс**

класс GPU
и режим запуска

GPU DRA

**Резервирует
и готовит GPU**

полная GPU или MIG;
совместно – MPS



Наbr, DRAйверы
для GPU

Сервис получает готовое
устройство в выбранном режиме —
далее разберём сами режимы

Не каждой модели нужен целый GPU

Основная модель

Qwen3.6-27B: веса ≈ 29 GB
и строгие требования к задержке

Вспомогательные сервисы

Короткие запросы и всплески:
векторизация, переранжирование,
речь, OCR

Отдать целиком одному сервису?

Один GPU

Память

Вычисления

Если каждый сервис получает
целый GPU, часть памяти
и вычислений остаётся без работы

NVIDIA GPU: устройство и совместный запуск

Полный GPU

Одно устройство

(вся память и вычисления)

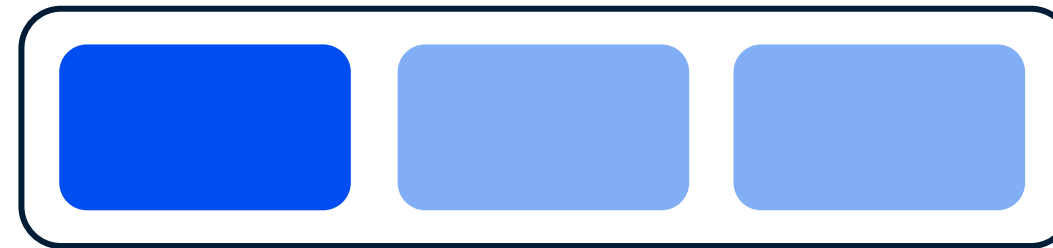


Если сервису нужна вся карта или совместный запуск запрещён

MIG

Отдельное устройство

(аппаратная изоляция)



Если подходит один из MIG-профилей этого GPU

MPS

Несколько процессов

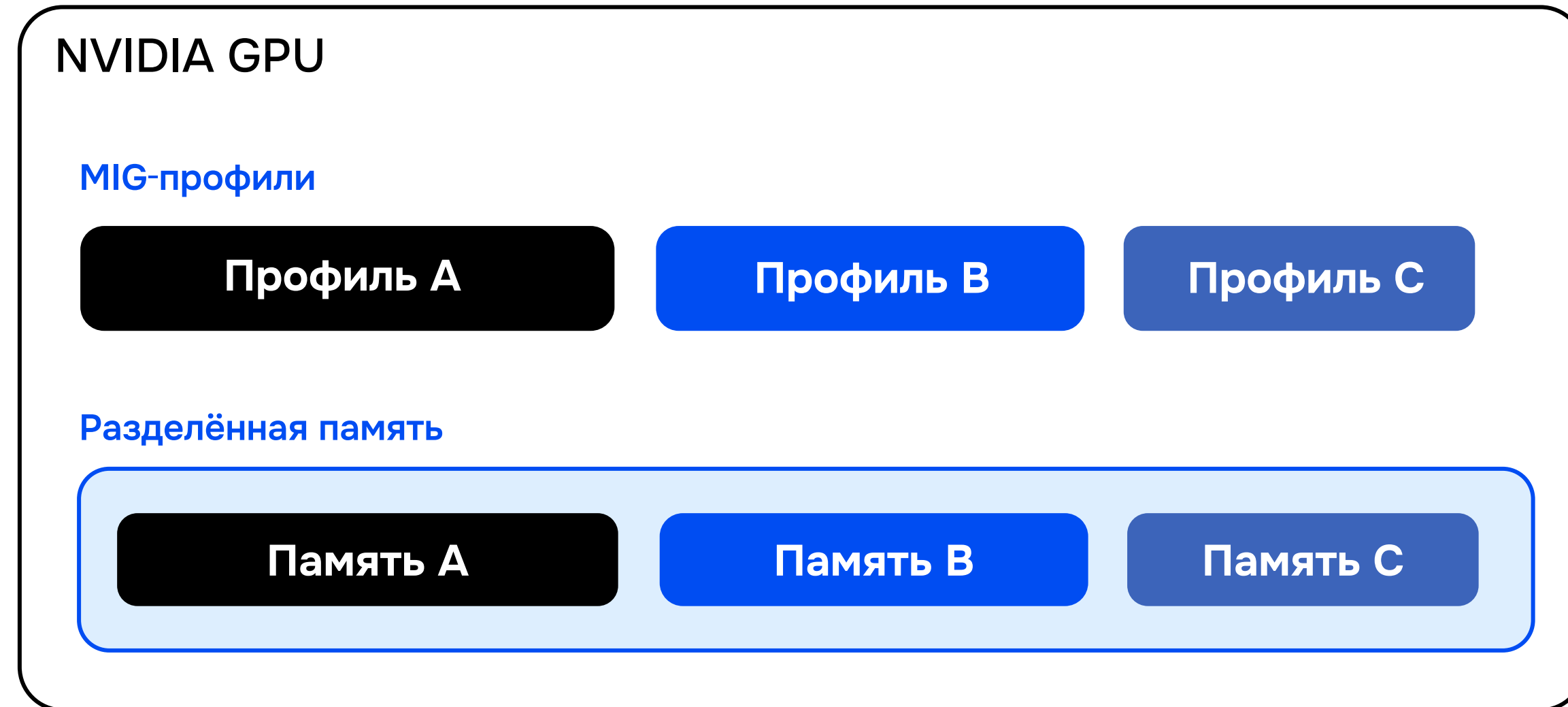
(на полном GPU или MIG)



Если разрешён совместный запуск; лимиты удерживает MPS-сервер

MIG — отдельная часть NVIDIA GPU

Размер и состав профилей определяет модель GPU



✔ **Подходит, если**

нужна аппаратная изоляция
и есть MIG-профиль нужного
размера

❗ **Ограничение:**

размеры задаются заранее;
произвольный объём
не запросить

MPS: несколько процессов на одном устройстве

Клиенты подключаются через CUDA-драйвер



Первый клиент

Control daemon запускает
MPS-сервер для устройства

Следующие клиенты

Подключаются к тому же серверу;
сервер удерживает лимит каждого

MPS: общая память, лимиты вычислений

Общая память, управляемые вычисления

Выбранное устройство

Общий пул памяти

Лимиты клиентов держит MPS

У каждого клиента

Отдельное адресное пространство;
заданная доля вычислений

Общее для процессов

Память остаётся общей;
совместный запуск разрешаем явно

Две настройки – два профиля запуска

Файл весов

Qwen3.6-27B FP8

≈ 29 GB весов

А – быстрый первый токен

KV-кеш на GPU, минимум обмена по PCIe

Один и тот же checkpoint

Две настройки – два профиля запуска

Файл весов

Qwen3.6-27B FP8

≈ 29 GB весов

Один и тот же checkpoint

А – быстрый первый токен

KV-кеш на GPU, минимум обмена по PCIe

В – высокая пропускная способность

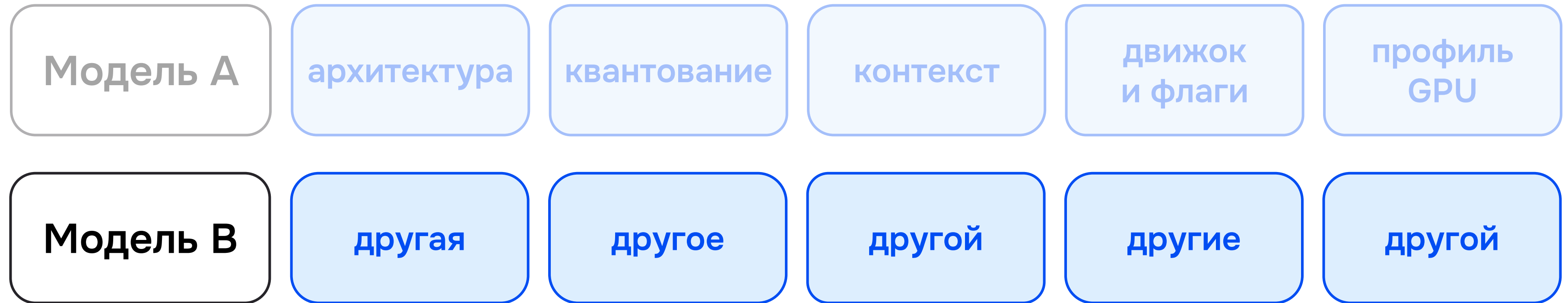
Больше сессий; часть KV-кеша в RAM,
итог проверяем замером

Для первой модели всё подобрано вручную



После нескольких запусков остаётся
одна рабочая конфигурация

Вторая модель – и проверка начинается сначала



Для новой модели заново сверяем
архитектуру, флаги, контекст и доступные GPU

Ручной подбор стал планировщиком

Входные данные

Модель

архитектура,
веса, контекст

Сервис

SLO, нагрузка,
число реплик

GPU-инвентарь

память, профили,
занятость

Планировщик

Перебирает доступные варианты

для каждого профиля
проверяет совместимость

Результат

Конфигурация запуска

движок, параметры и профиль
GPU или конкретная причина
отказа

Проверка вариантов

Каждый доступный профиль проходит
одинаковые обязательные проверки

Каждый вариант проходит пять проверок

01

Конфигурация
найдена

02

Устройство
подходит

03

Делить GPU
разрешено

04

Среда
доступна

05

Памяти
хватает

Для каждой конфигурации фиксируем
первую причину отказа

Берём минимально достаточный GPU

заказ на чат-сервис

Qwen3.6-27B FP8

контекст: 128K токенов

2 сессии одновременно

KV-кеш: $2 \times 8 = 16$ GiB

бюджет запуска: ≈ 45 GB

40 GB

не подходит: $40 < 45$ GB

NoAcceleratorWithEnoughMemory

80 GB

45 GB помещаются

96 GB

не занимаем без необходимости

Планировщик задаёт класс GPU, DRA назначает карту

План запуска

ResourceClaim

После выделения

класс устройства
gpu-80gb

заявка на класс
ресурса
без имени
конкретной карты

gpu-3 на node-07
GPU 80 GB

Pod не привязан к PCI-адресу:
устройство выбирает DRA (Dynamic Resource Allocation)

Драйвер готовит GPU для контейнера

01

Заявка на ресурс
выделена

Драйвер готовит GPU для контейнера

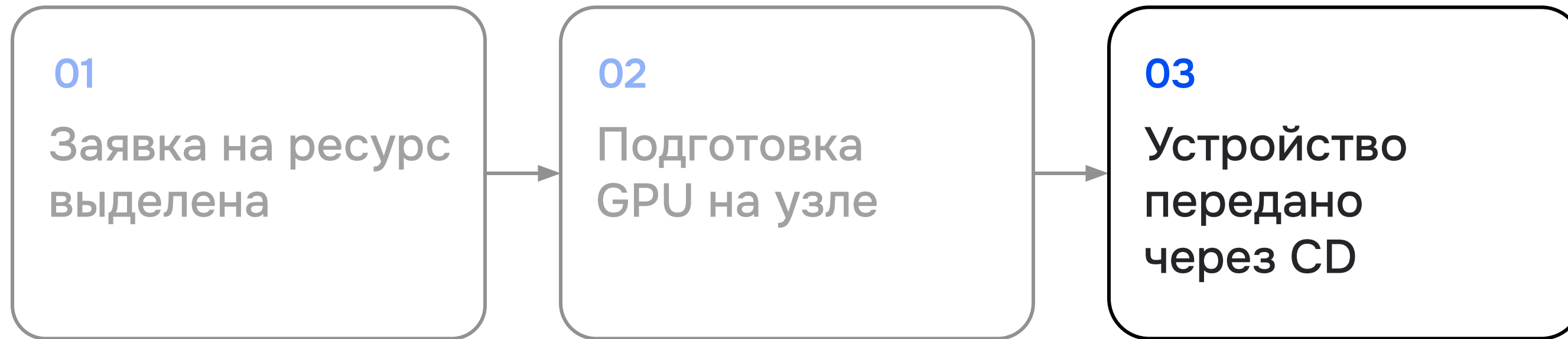
01

Заявка на ресурс
выделена

02

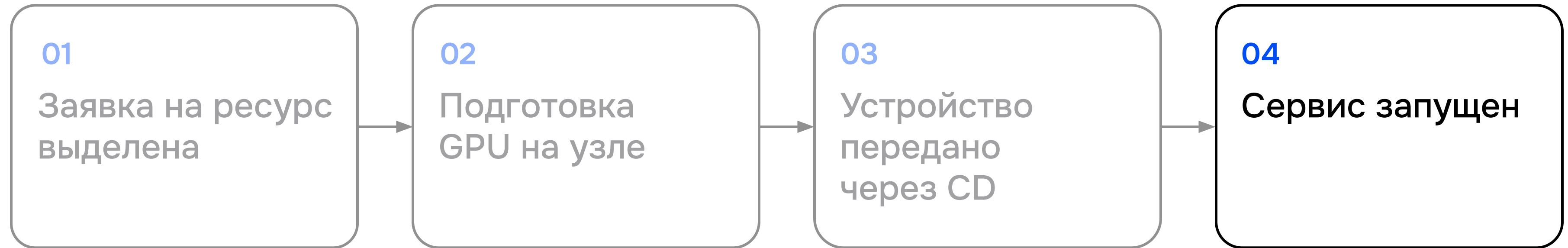
Подготовка
GPU на узле

Драйвер готовит GPU для контейнера



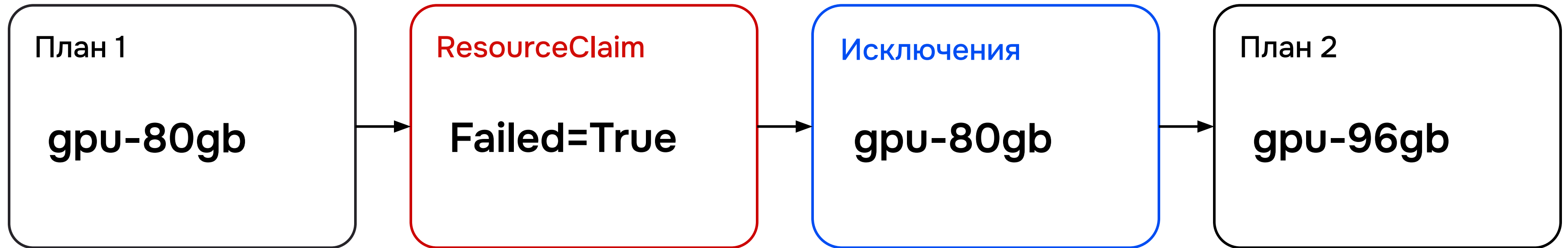
CDI сообщает контейнеру, какое устройство подключить, без ручной передачи PCI-адреса

Драйвер готовит GPU для контейнера



CDI сообщает контейнеру, какое устройство подключить, без ручной передачи PCI-адреса

Если класс GPU не сработал – пробуем следующий



Требования не меняются; исключённый класс остаётся в статусе

Статус отвечает на два вопроса

Что будет запущено

- GPU и режим размещения
- Память на реплику
- Движок и параметры запуска

Что происходит сейчас

- Рабочая нагрузка готова или создаётся
- Какой класс GPU уже не подошёл
- Сколько попыток запуска уже было

При сбое следующий вариант выбирается с учётом уже исключённых классов

Планы: распределённый запуск

Две фазы сервиса

Разделяем чтение и генерацию (P/D disaggregation)

Подбирать ресурсы для чтения контекста и генерации независимо; учитывать передачу KV-блоков

Несколько реплик

Маршрутизация по KV-кешу (cache-aware routing)

Учитывать расположение KV-блоков, очередь и ожидаемое время до первого токена

Для инженера

Сначала разберите, где упёрлись

Проверьте отдельно память, загрузку GPU и размещение. Тогда понятно, что менять: параметры vLLM, профиль GPU или сам парк.

Для бизнеса

В бюджет идёт только подтверждённый дефицит

Нужны три числа: целевая нагрузка, ёмкость текущего парка после оптимизаций и остаток, который он не закрывает.

Самая дорогая карта в кластере — та, что простаивает

Докупать нельзя, доиспользовать

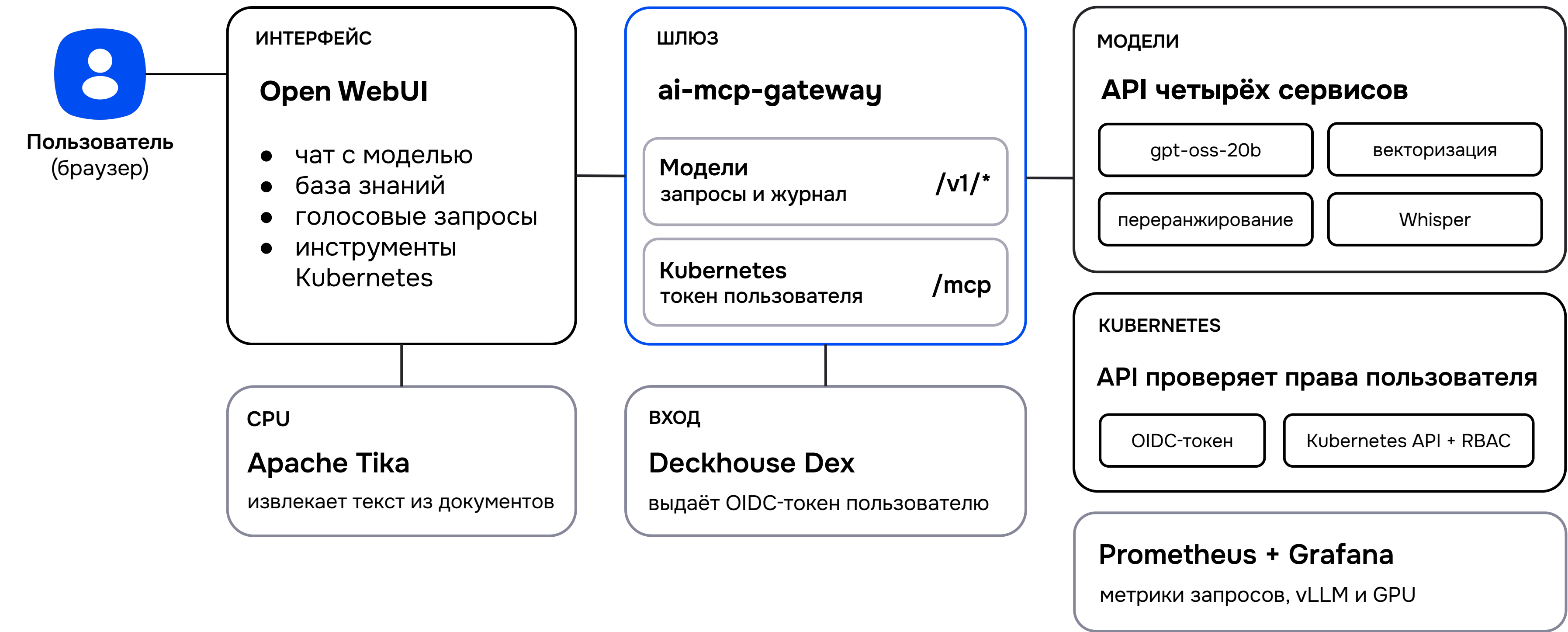
Сначала используйте уже купленные GPU

Стенд: две RTX 5060 Ti

gpt-oss-20b · по 16 ГБ на карту

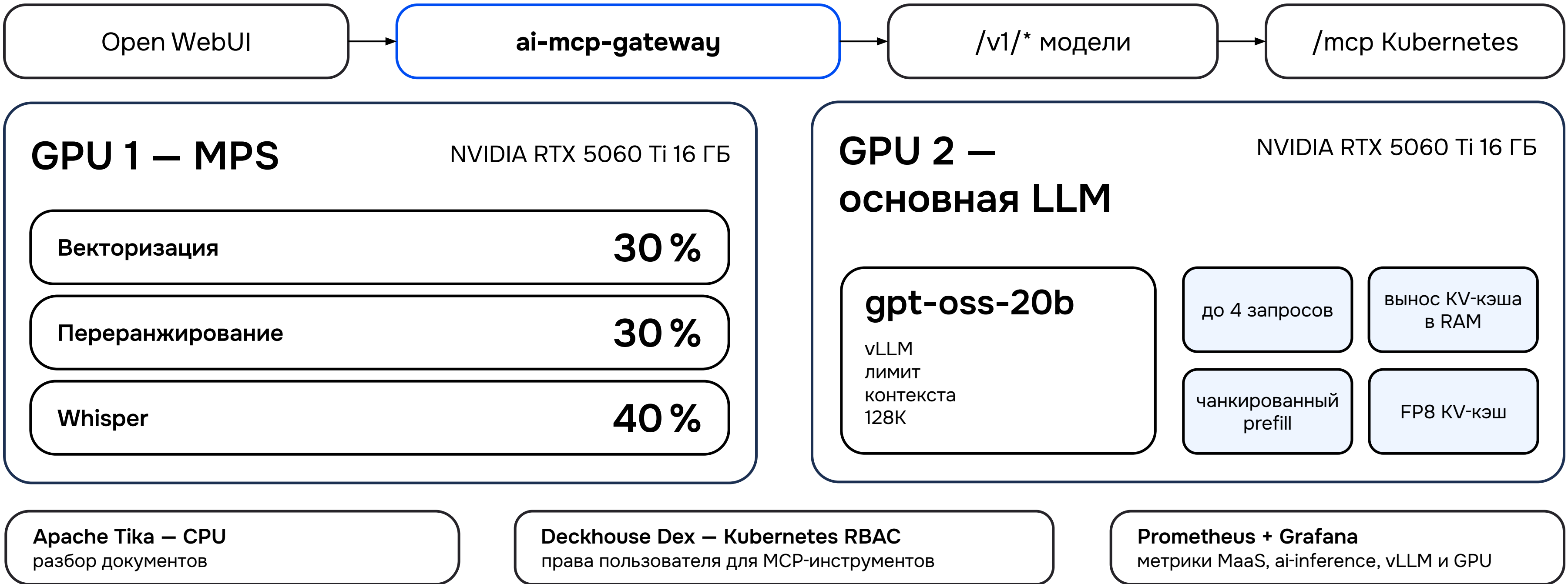
Как запросы идут к моделям и Kubernetes

/v1/* ведёт к моделям, /mcp — к Kubernetes. Оба маршрута проходят через ai-mcp-gateway.



LLM — на одном GPU. Три сервиса — на втором

Две RTX 5060 Ti по 16 ГБ: основная модель принимает контекст до 128K токенов; векторизация, переранжирование и Whisper делят второй GPU через MPS



При 8 клиентах ожидание сократилось на 82 %

p95 до первого токена при 8 клиентах

39,4 → 7,1 с

ожидание сократилось на 82 %

Пропускная способность сервиса

21,7 → 76,3 токена/с

в 3,5 раза выше при 8 клиентах

Контрольный прогон: менялись несколько настроек

KV-кеш хранится в FP8

Одновременно обрабатываются
до четырёх запросов

Результат — эффект
всей комбинации

Эффект одной настройки отдельно не измерялся

128K-контекст: 2,92 GiB KV-кеша ушло в RAM

Холодный запрос

127 537 токенов на входе

до первого токена: 270,7 с
полный ответ: 277,3 с

Во время запроса

2,92 GiB KV-кеша

выгружено из GPU в RAM
64 токена на выходе

Что подтверждает холодный прогон. Рабочая стратегия:

Контекст 128K
поместился

KV-offload
освободил GPU

Цена – обмен
по PCIe

deckhouse

**Спасибо
за внимание**

 @deckhouse_news

 github.com/deckhouse

