html academy
интерактивные
онлайн-курсы

Раздел 8:

Микросервисы

План

План

– Асинхронное поведение Node.js

План

– Асинхронное поведение Node.js

– Масштабируемость

План

– Асинхронное поведение Node.js

– Масштабируемость

– Кластеризация

План

– Асинхронное поведение Node.js

– Масштабируемость

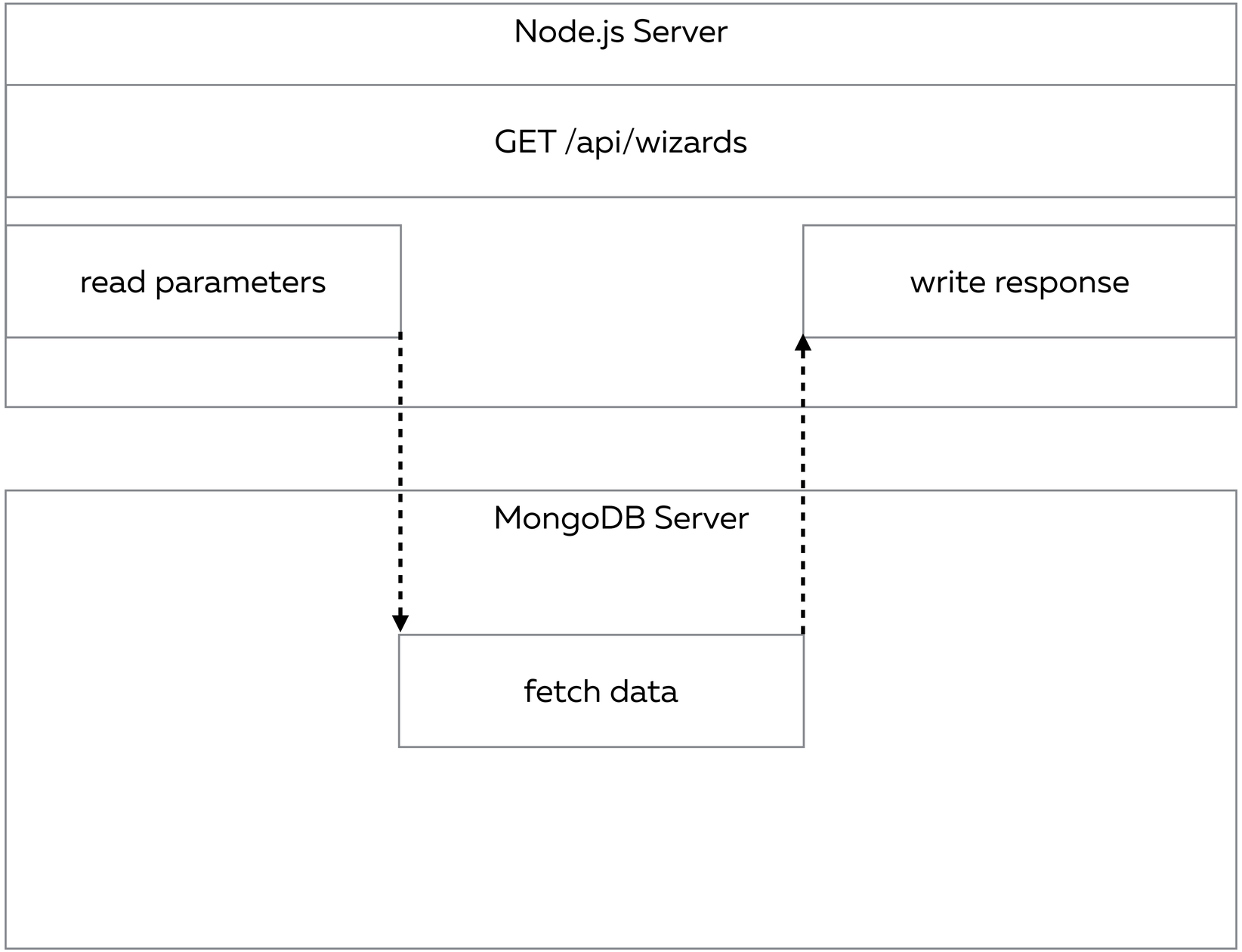
– Кластеризация

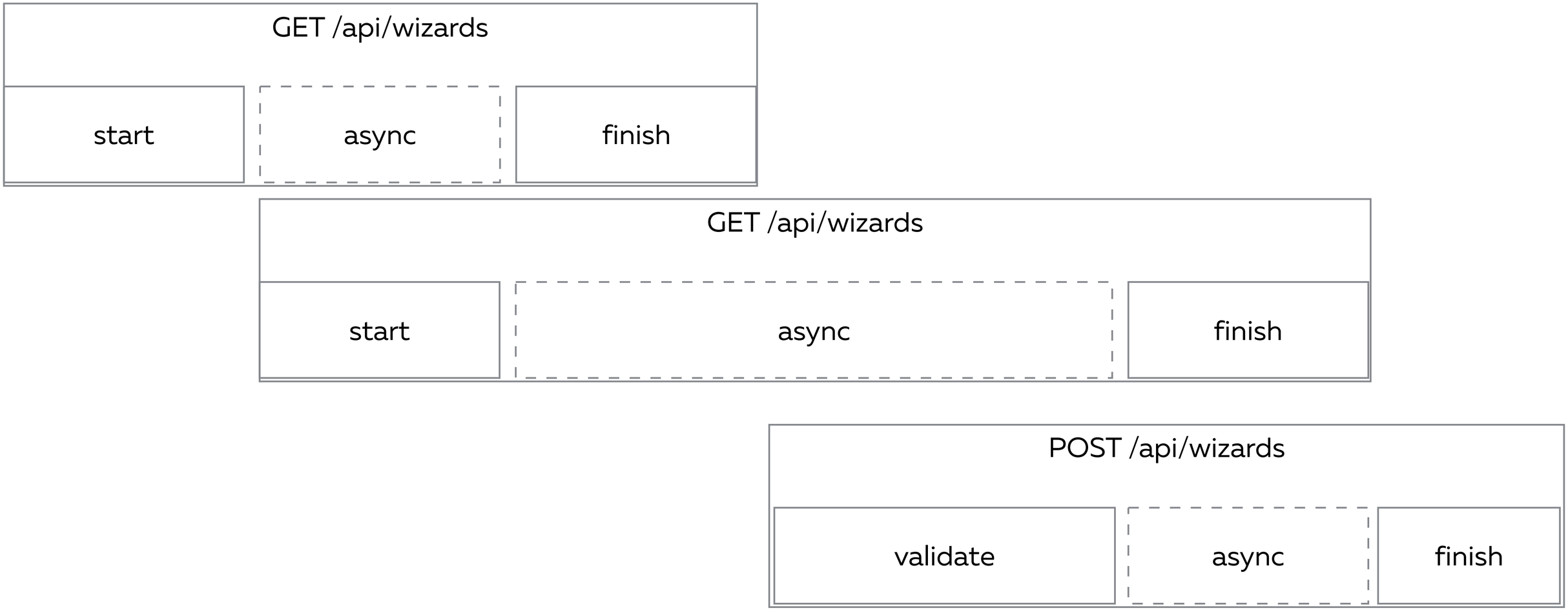
– Состояние приложения

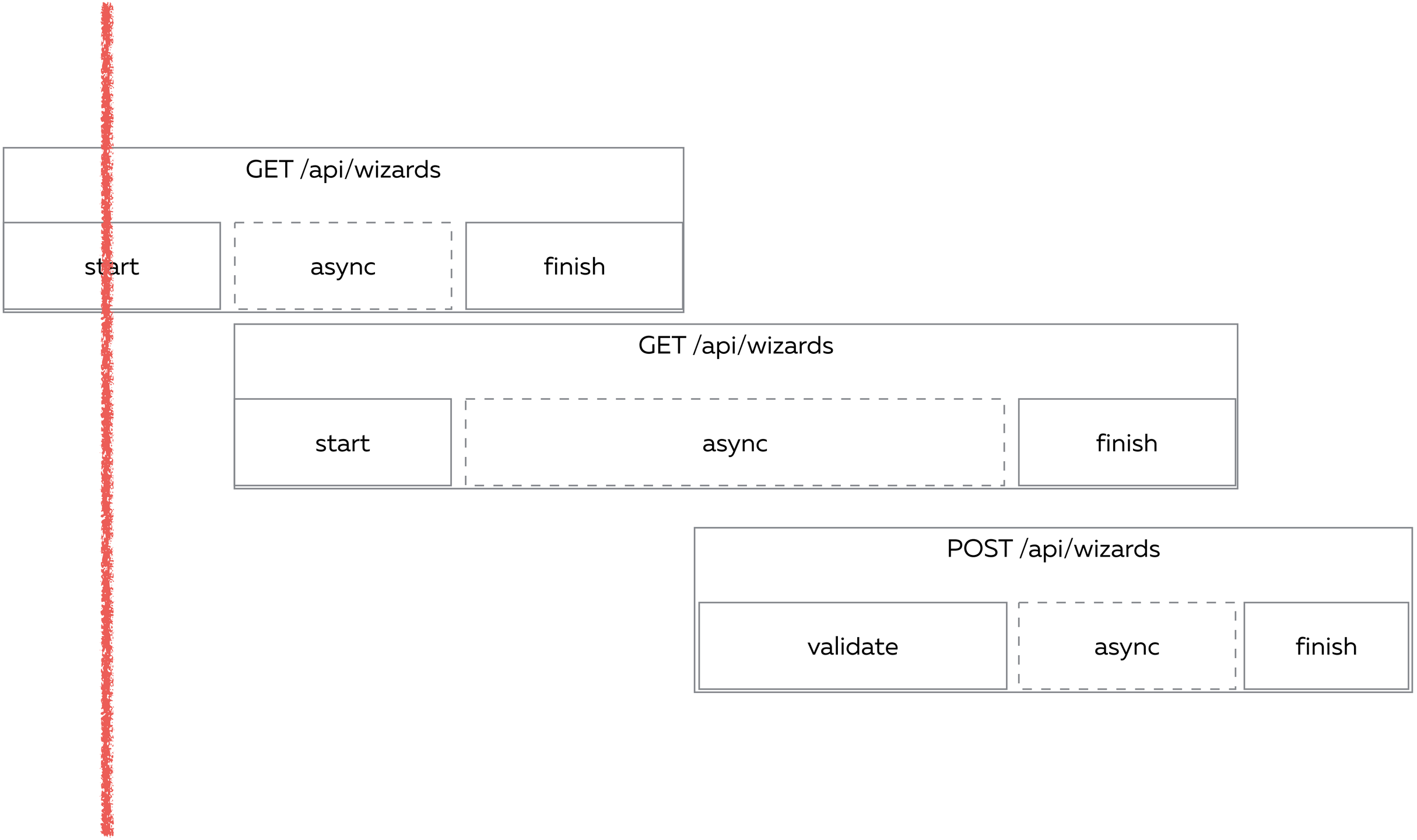
План

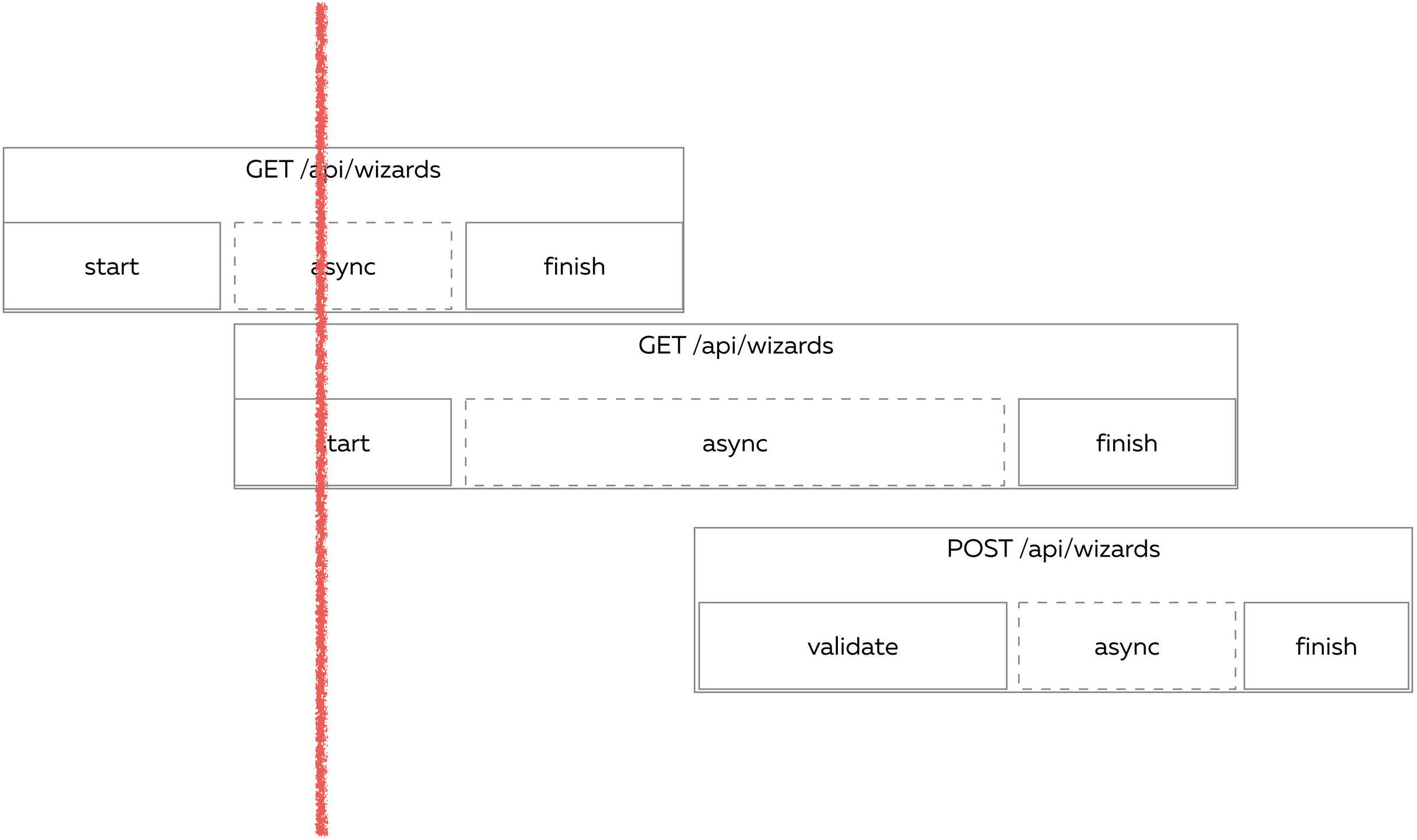
- Асинхронное поведение Node.js
- Масштабируемость
- Кластеризация
- Состояние приложения
- Где хранить состояния

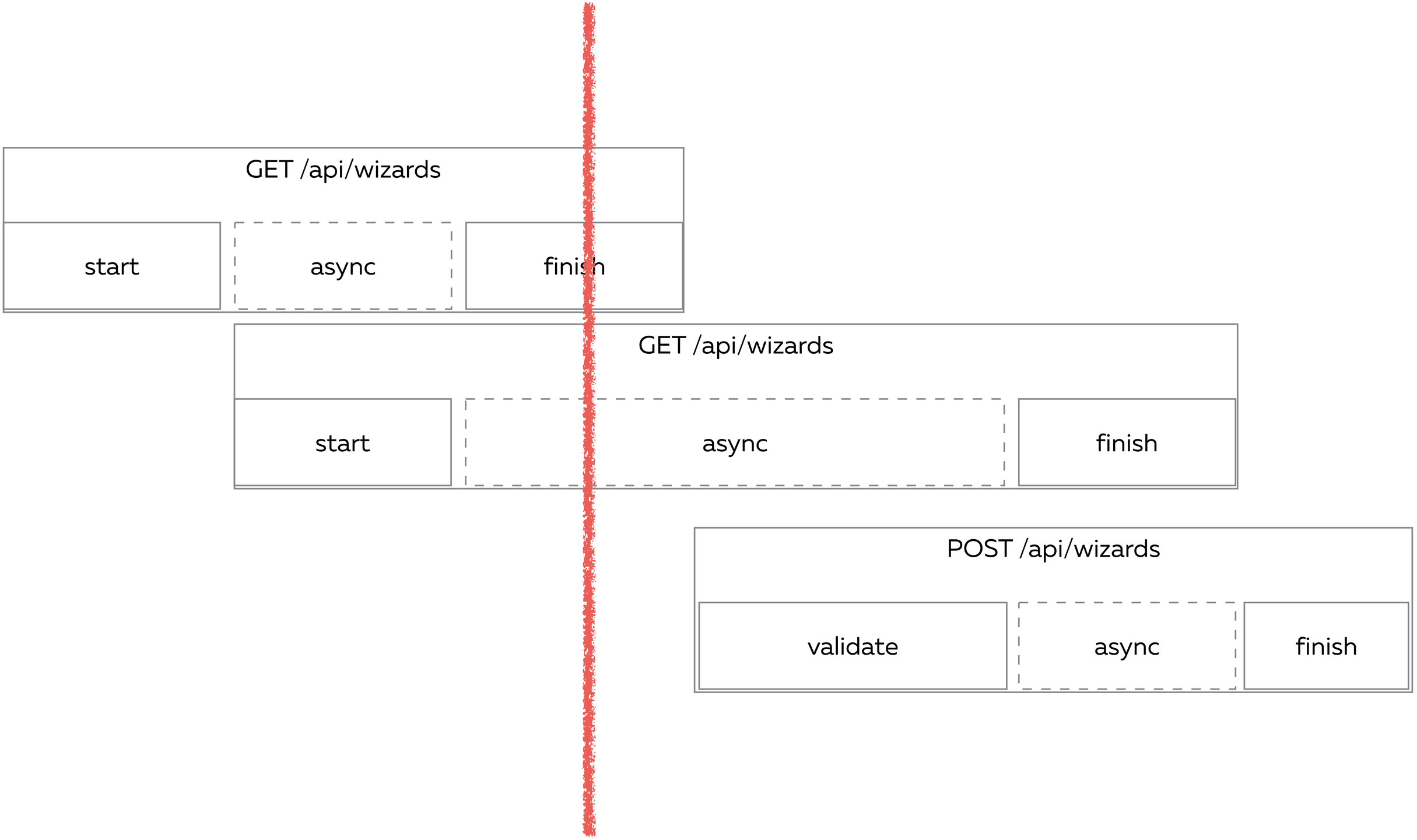
Асинхронность

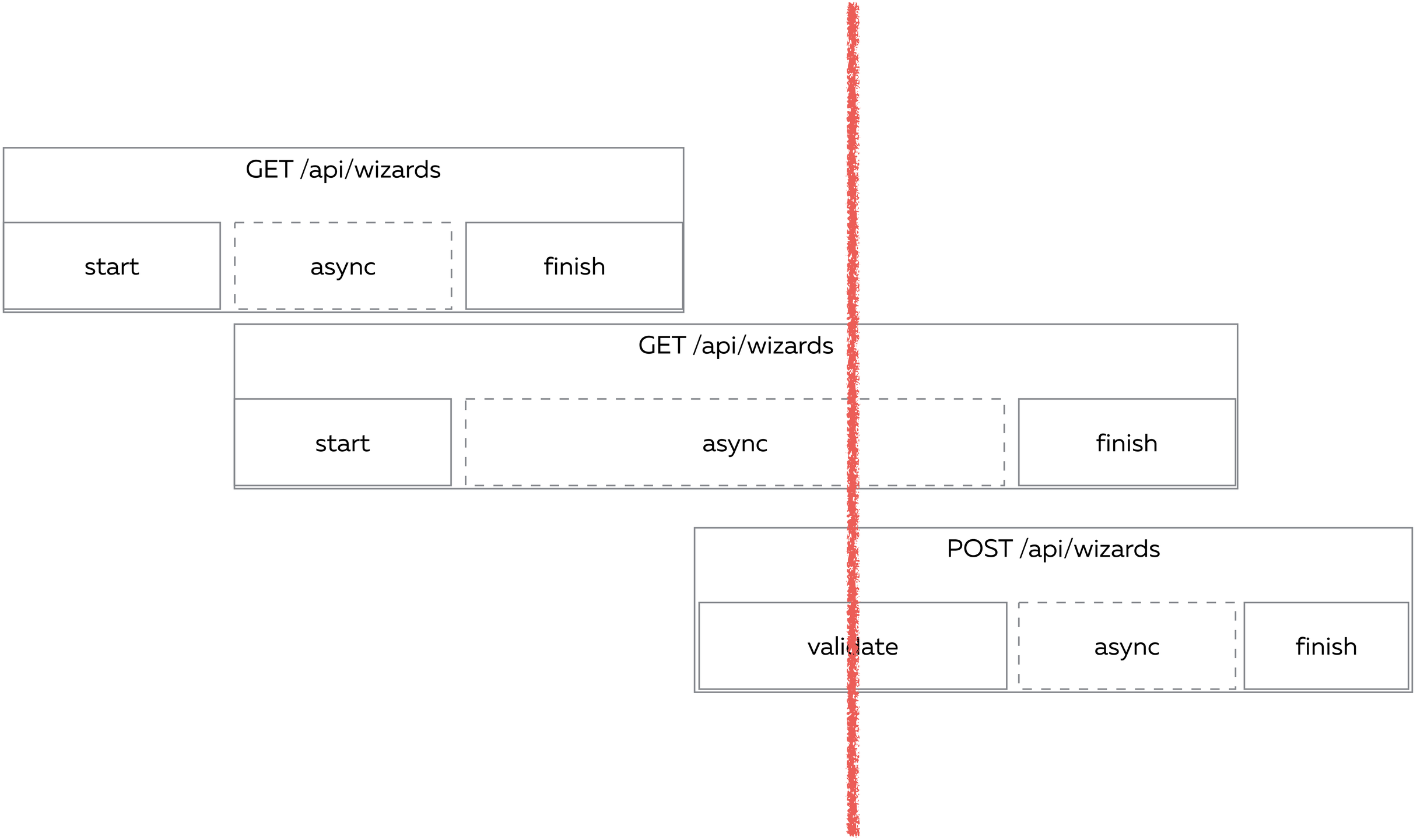

Node.js Server
GET /api/wizards
read parameters
write response
MongoDB Server
fetch data

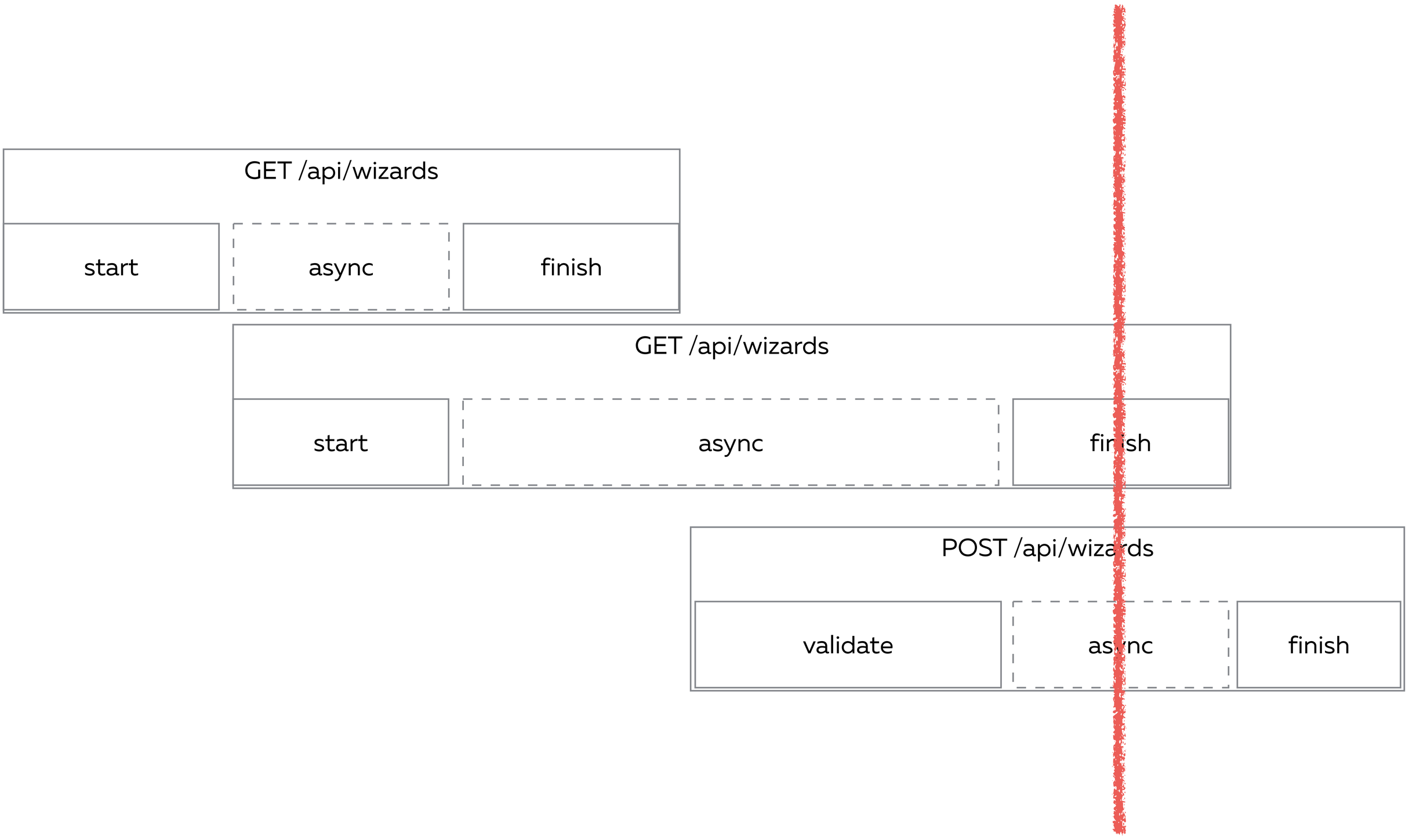

GET /api/wizards
start
async
finish
GET /api/wizards
start
async
finish
POST /api/wizards
validate
async
finish


GET /api/wizards
start
async
finish
GET /api/wizards
start
async
finish
POST /api/wizards
validate
async
finish


GET /api/wizards
start
async
finish
GET /api/wizards
start
async
finish
POST /api/wizards
validate
async
finish


GET /api/wizards
start
async
finish
GET /api/wizards
start
async
finish
POST /api/wizards
validate
async
finish


GET /api/wizards
start
async
finish
GET /api/wizards
start
async
finish
POST /api/wizards
validate
async
finish


GET /api/wizards
start
async
finish
GET /api/wizards
start
async
finish
POST /api/wizards
validate
async
finish

“В каждый момент времени выполняется ровно одна функция, никакой код параллельно не выполняется”

Что блокирует *event loop*

Что блокирует *event loop*

– Дорогостоящие синхронные операции:

Что блокирует *event loop*

- Дорогостоящие синхронные операции:
  - File System

Что блокирует *event loop*

- Дорогостоящие синхронные операции:
  - File System
  - DNS, Crypto, Zlib и прочие "тяжёлые" операции

Что блокирует *event loop*

- Дорогостоящие синхронные операции:
  - File System
  - DNS, Crypto, Zlib и прочие "тяжёлые" операции
- Синхронные операции зависящие от входных данных

Что блокирует *event loop*

- Дорогостоящие синхронные операции:
  - File System
  - DNS, Crypto, Zlib и прочие "тяжёлые" операции
- Синхронные операции зависящие от входных данных
- Регулярные выражения (скорость работы зависит от входных данных)

Что блокирует *event loop*

- Дорогостоящие синхронные операции:
  - File System
  - DNS, Crypto, Zlib и прочие "тяжёлые" операции
- Синхронные операции зависящие от входных данных
- Регулярные выражения (скорость работы зависит от входных данных)
  - «Заглядывания» назад (например, `` `(a.*) \1` ``)

Что блокирует *event loop*

- Дорогостоящие синхронные операции:
  - File System
  - DNS, Crypto, Zlib и прочие "тяжёлые" операции
- Синхронные операции зависящие от входных данных
- Регулярные выражения (скорость работы зависит от входных данных)
  - «Заглядывания» назад (например, `` `(a.*) \1` ``)
  - Непредсказуемые деоптимизации (например: `` `(a+)*` ``, `` `(a|a)*` ``)

Что блокирует *event loop*

- Дорогостоящие синхронные операции:
  - File System
  - DNS, Crypto, Zlib и прочие "тяжёлые" операции
- Синхронные операции зависящие от входных данных
- Регулярные выражения (скорость работы зависит от входных данных)
  - «Заглядывания» назад (например, `` `(a.*) \1` ``)
  - Непредсказуемые деоптимизации (например: `` `(a+)*` ``, `` `(a|a)*` ``)
- Сериализация/десериализация данных (*JSON.stringify*, *JSON.parse*)

Способы борьбы с блокировкой *event loop*

Способы борьбы с блокировкой *event loop*

– Уменьшение операции

Способы борьбы с блокировкой *event loop*

– Уменьшение операции

– Разделение больших синхронных операции на несколько маленьких асинхронных

Способы борьбы с блокировкой *event loop*

– Уменьшение операции

– Разделение больших синхронных операции на несколько маленьких асинхронных

– Вынесение долгих операции в отдельный процесс (*offloading*)

Способы борьбы с блокировкой *event loop*

– Уменьшение операции

– Разделение больших синхронных операции на несколько маленьких асинхронных

– Вынесение долгих операции в отдельный процесс (*offloading*)

– Знание библиотек, которые используются в проекте и как они работают:

Способы борьбы с блокировкой *event loop*

- Уменьшение операции
- Разделение больших синхронных операции на несколько маленьких асинхронных
- Вынесение долгих операции в отдельный процесс (*offloading*)
- Знание библиотек, которые используются в проекте и как они работают:
  - Есть ли API и документация?

Способы борьбы с блокировкой *event loop*

- Уменьшение операции
- Разделение больших синхронных операции на несколько маленьких асинхронных
- Вынесение долгих операции в отдельный процесс (*offloading*)
- Знание библиотек, которые используются в проекте и как они работают:
  - Есть ли API и документация?
  - Какова сложность одного метода вызова API?

Масштабируемость

Масштабирование

Масштабирование

– Вертикальное

увеличение производительности каждого компонента системы с целью повышения общей производительности

Масштабирование

– Вертикальное

  увеличение производительности каждого компонента системы с целью повышения общей производительности

– Горизонтальное

  разбиение системы на более мелкие структурные компоненты и разнесение их по отдельным физическим машинам (или их группам), и (или) увеличение количества серверов, параллельно выполняющих одну и ту же функцию

Вертикальное масштабирование

Вертикальное масштабирование Node.js

Вертикальное масштабирование Node.js

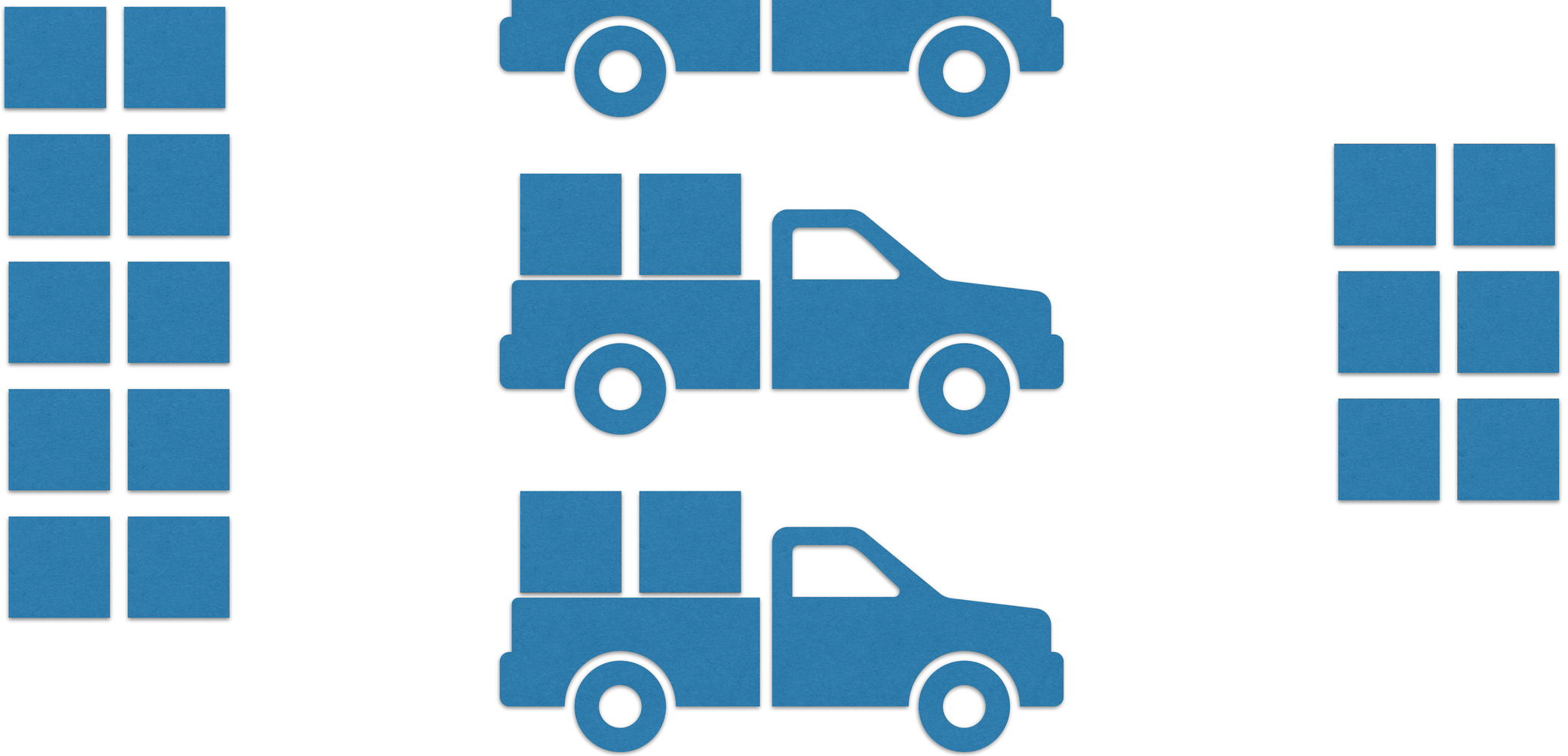
– Фиксированое ограничение памяти — 1.5 ГБ

Вертикальное масштабирование Node.js

– Фиксированое ограничение памяти — 1.5 ГБ

– Использует ровно 1 CPU

Горизонтальное масштабирование



Горизонтальное масштабирование

Горизонтальное масштабирование

– Плюсы:

Горизонтальное масштабирование

- Плюсы:
  - Тратится ровно столько ресурсов, сколько необходимо в данный момент

Горизонтальное масштабирование

- Плюсы:
  - Тратится ровно столько ресурсов, сколько необходимо в данный момент
  - Повышается общая отказоустойчивость системы

Горизонтальное масштабирование

- Плюсы:
  - Тратится ровно столько ресурсов, сколько необходимо в данный момент
  - Повышается общая отказоустойчивость системы
  - Доступность 24/7

Горизонтальное масштабирование

- Плюсы:
  - Тратится ровно столько ресурсов, сколько необходимо в данный момент
  - Повышается общая отказоустойчивость системы
  - Доступность 24/7

Горизонтальное масштабирование

- Плюсы:
  - Тратится ровно столько ресурсов, сколько необходимо в данный момент
  - Повышается общая отказоустойчивость системы
  - Доступность 24/7

- Минусы

Горизонтальное масштабирование

- Плюсы:
  - Тратится ровно столько ресурсов, сколько необходимо в данный момент
  - Повышается общая отказоустойчивость системы
  - Доступность 24/7

- Минусы
  - Нужно программировать

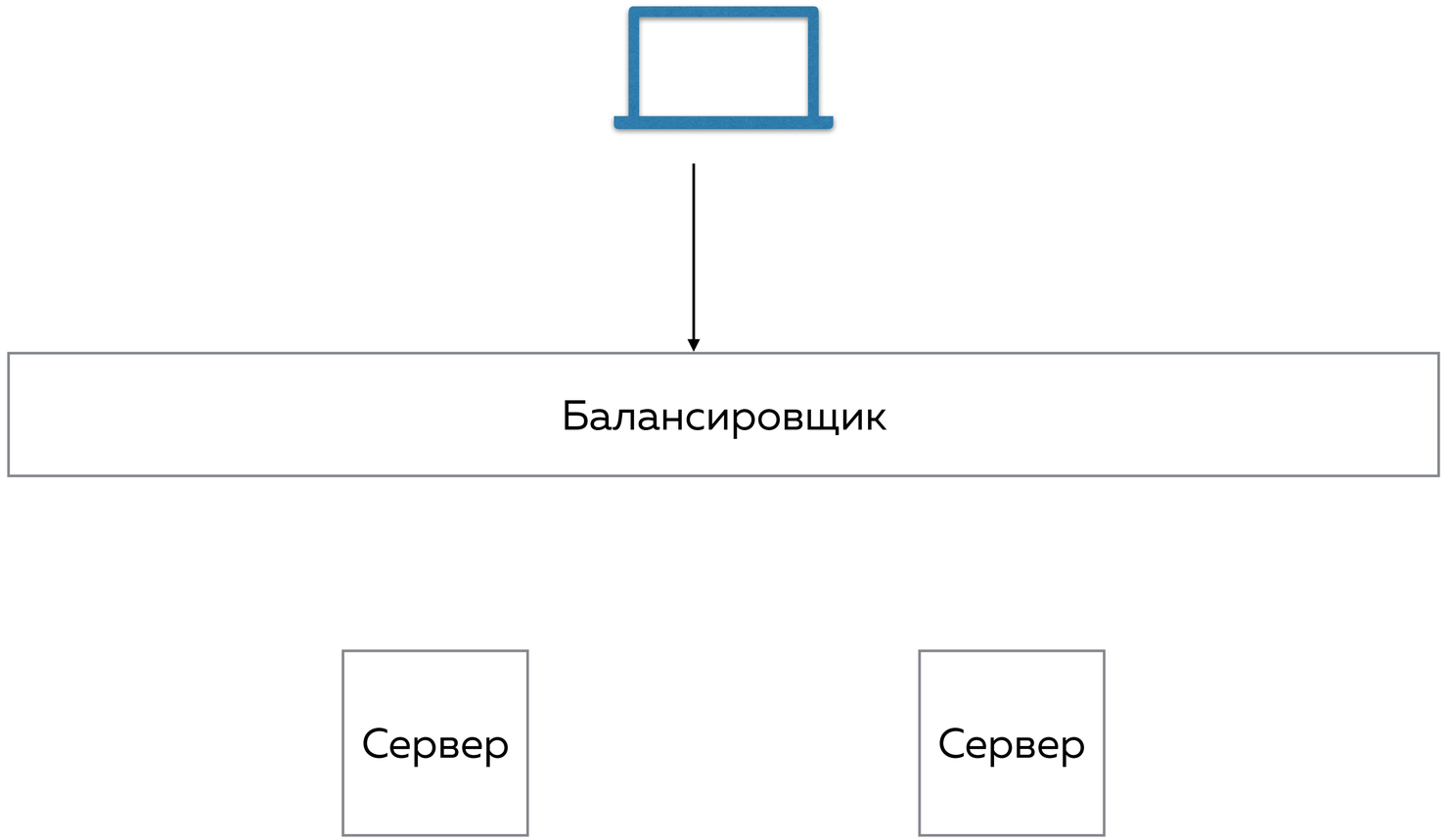
Кластер

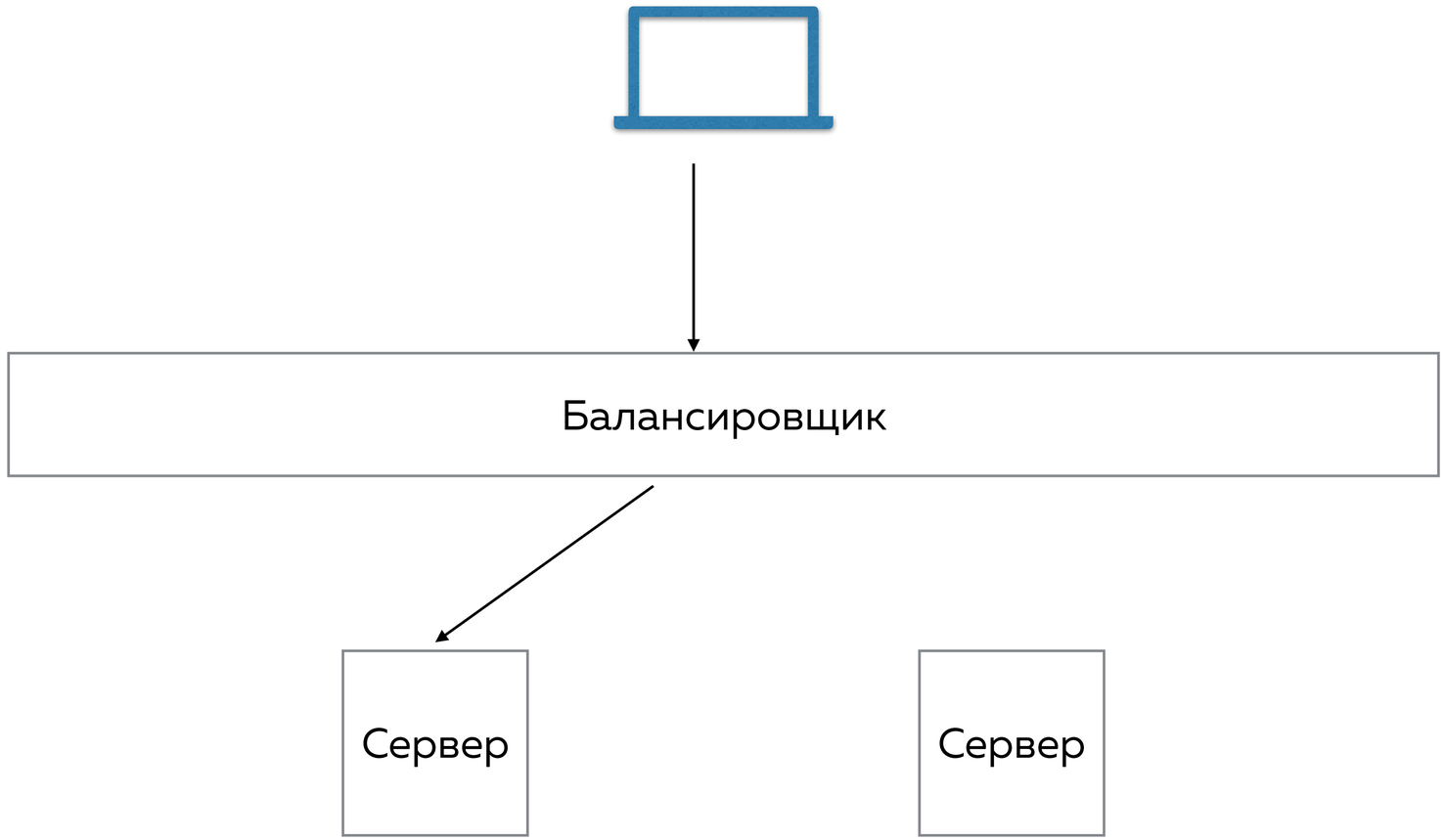
объединение нескольких однородных элементов, которое может рассматриваться как самостоятельная единица, обладающая определёнными свойствами

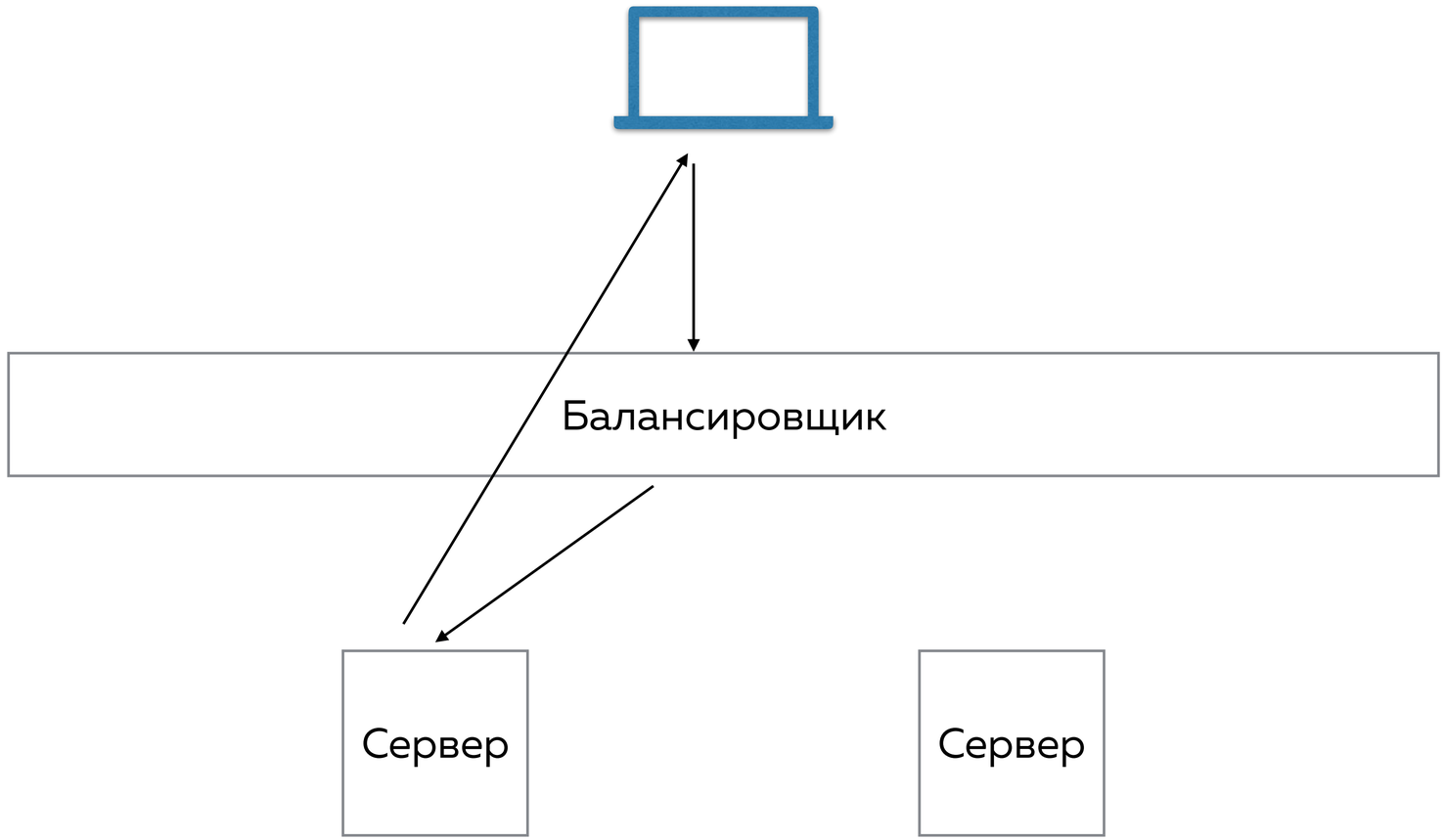
Балансировщик

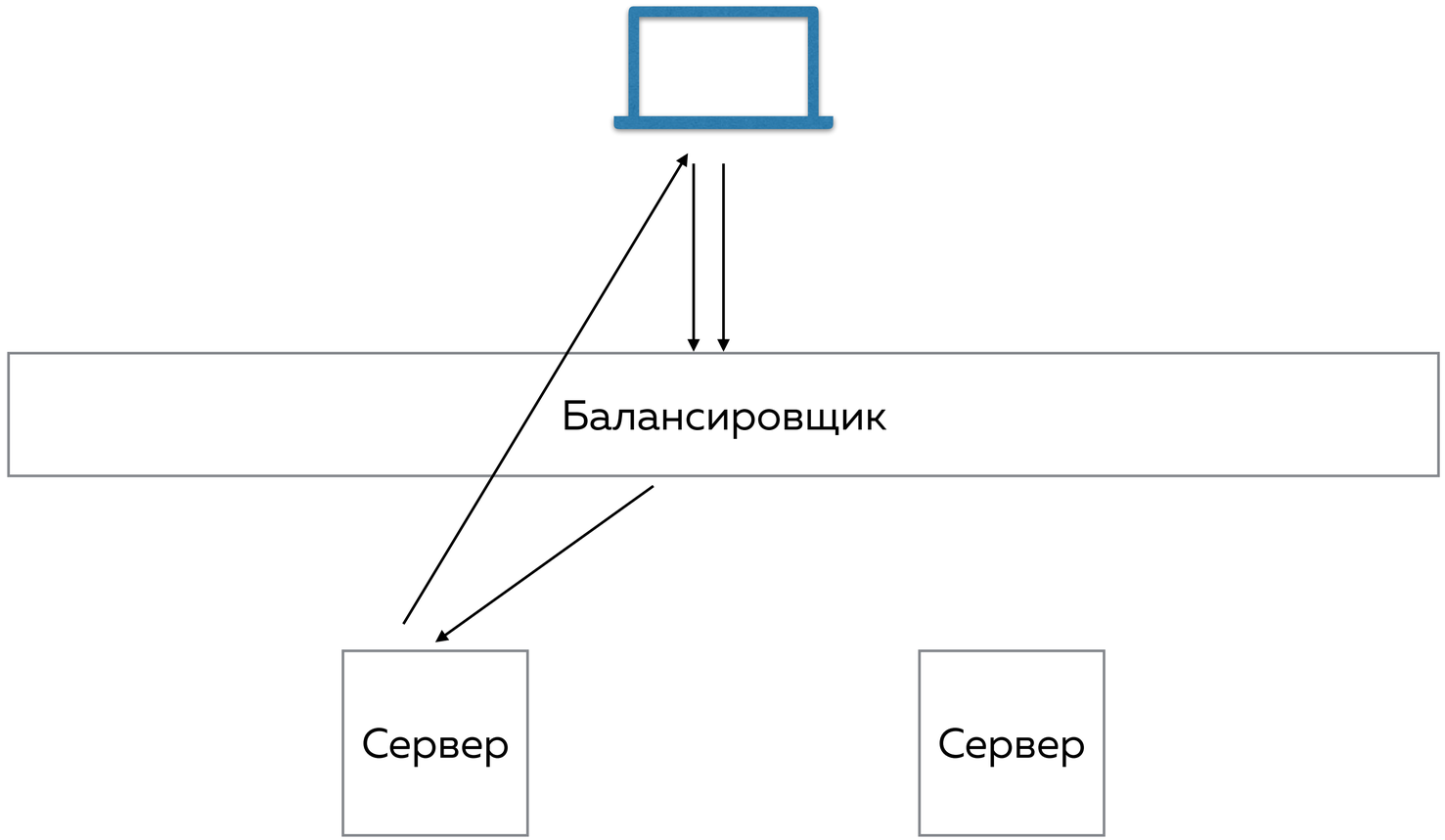
Сервер

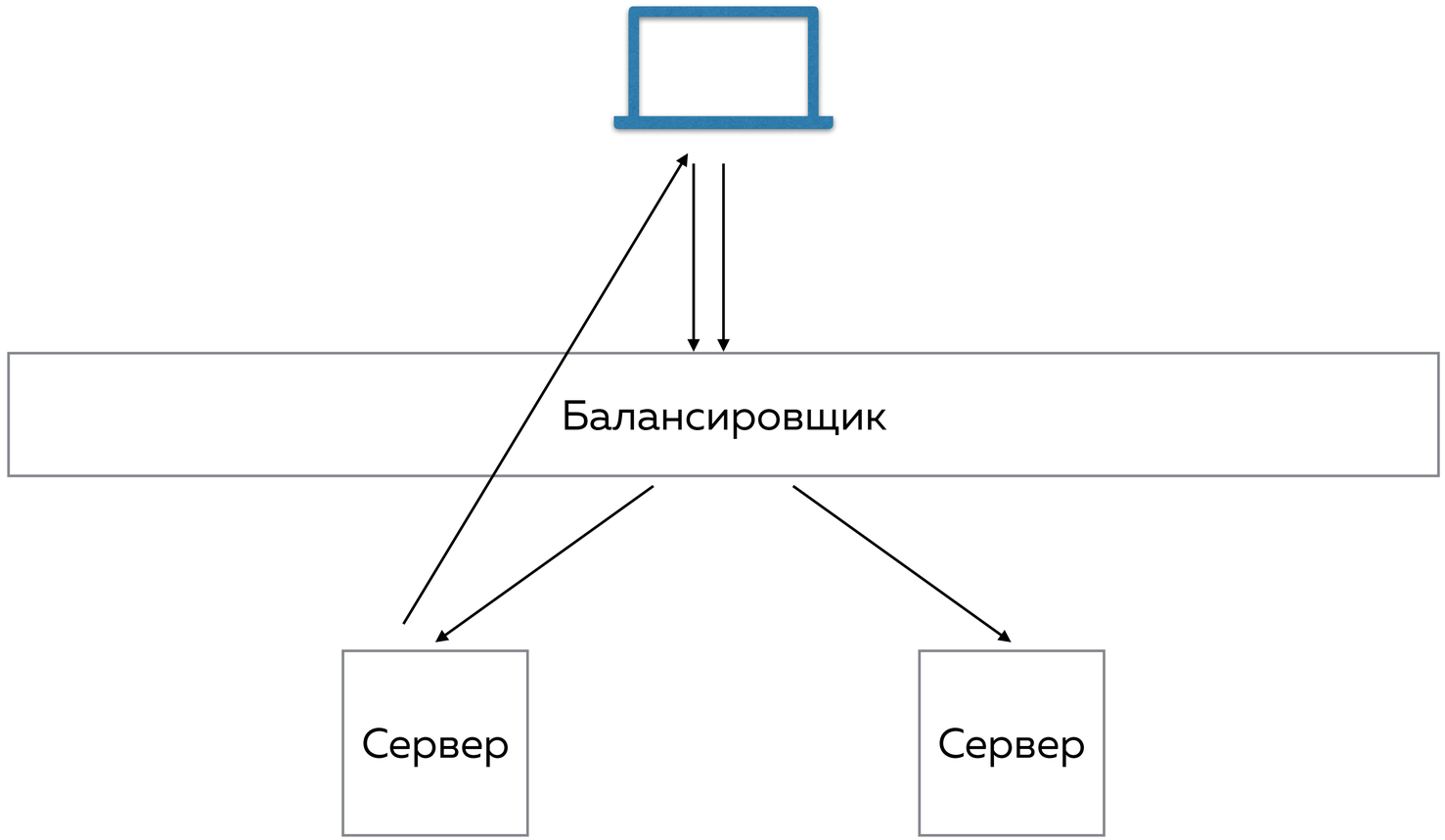
Сервер

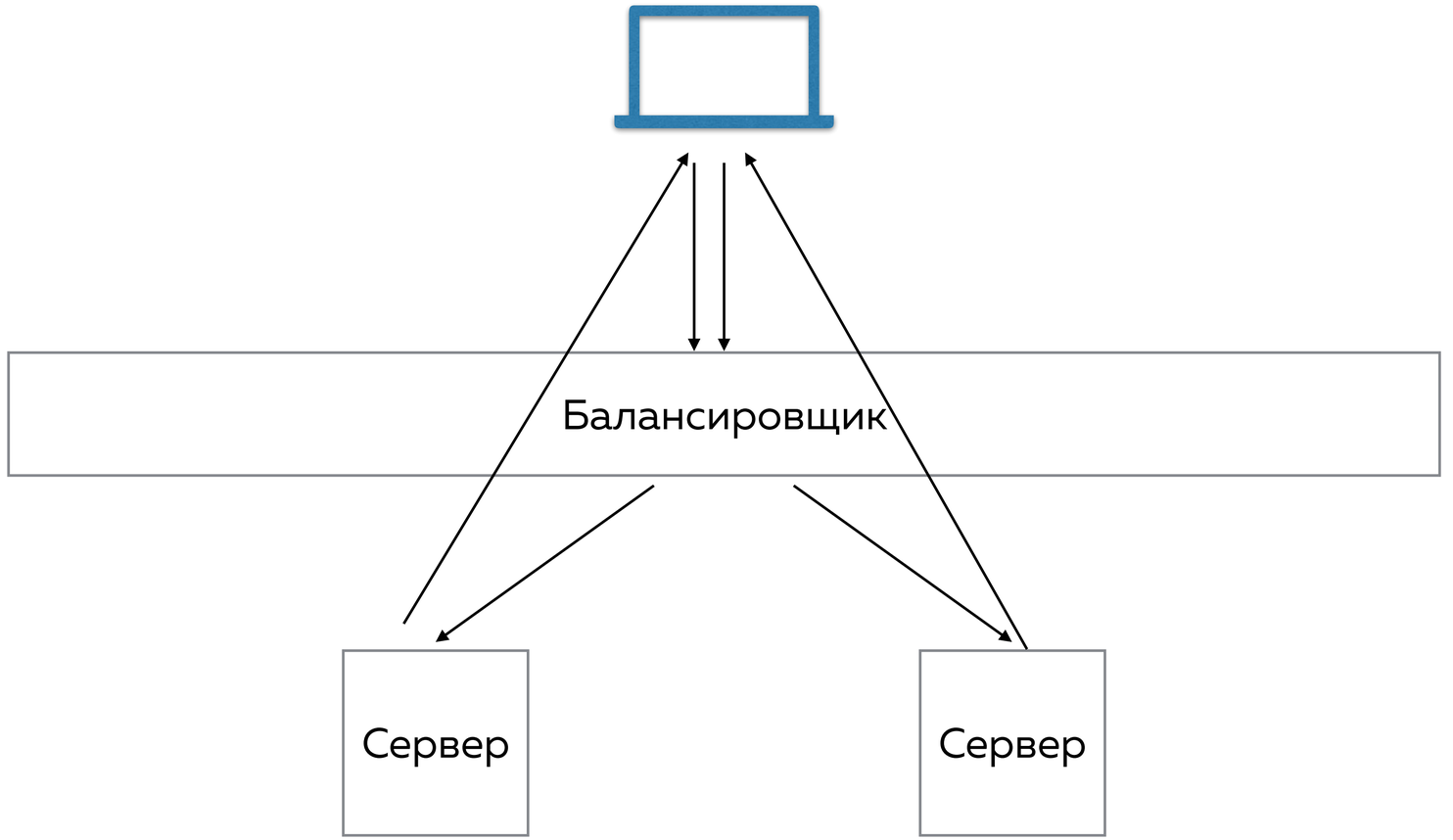

Балансировщик
Сервер
Сервер


Балансировщик
Сервер
Сервер


Балансировщик
Сервер
Сервер

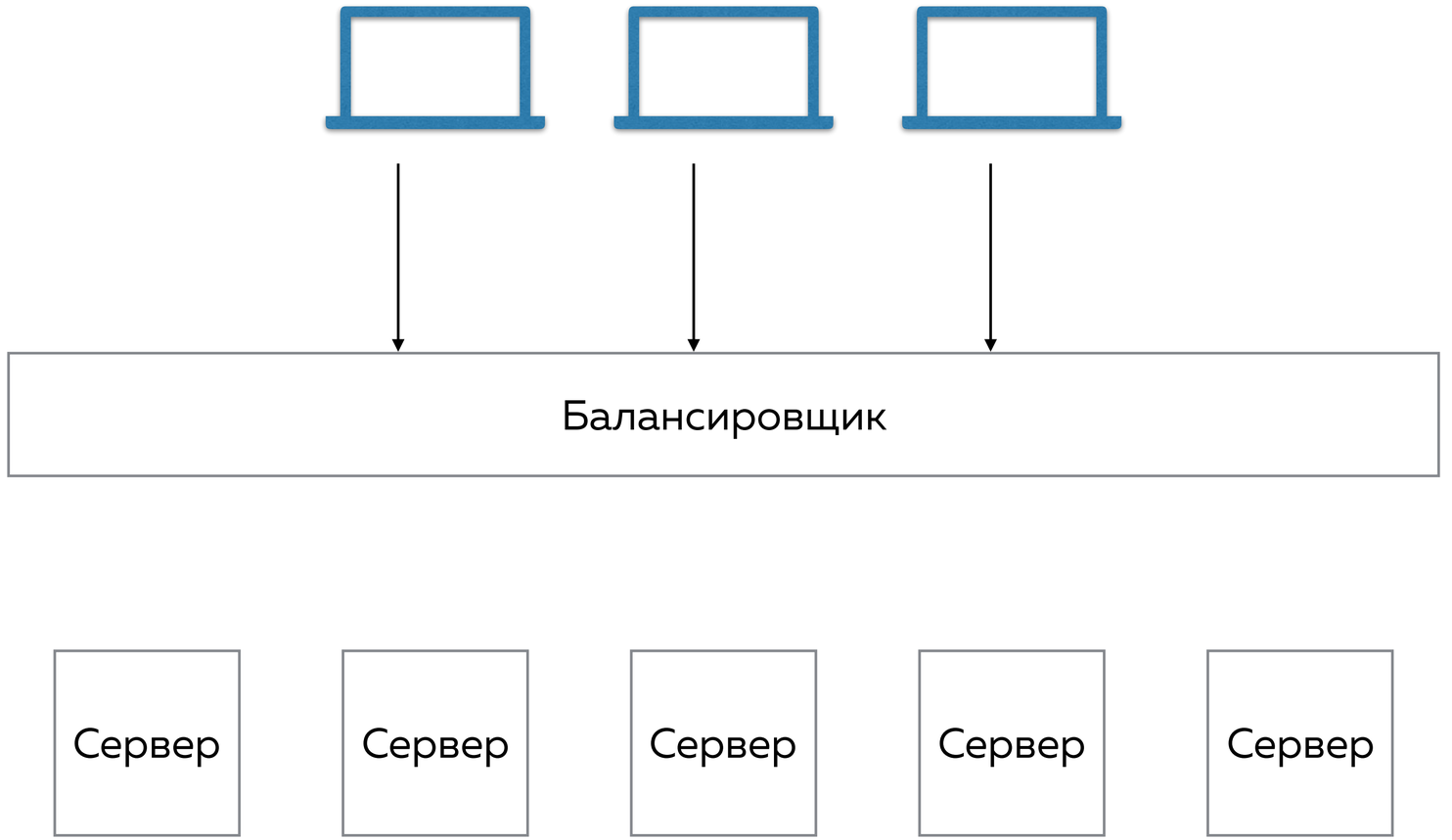

Балансировщик
Сервер
Сервер

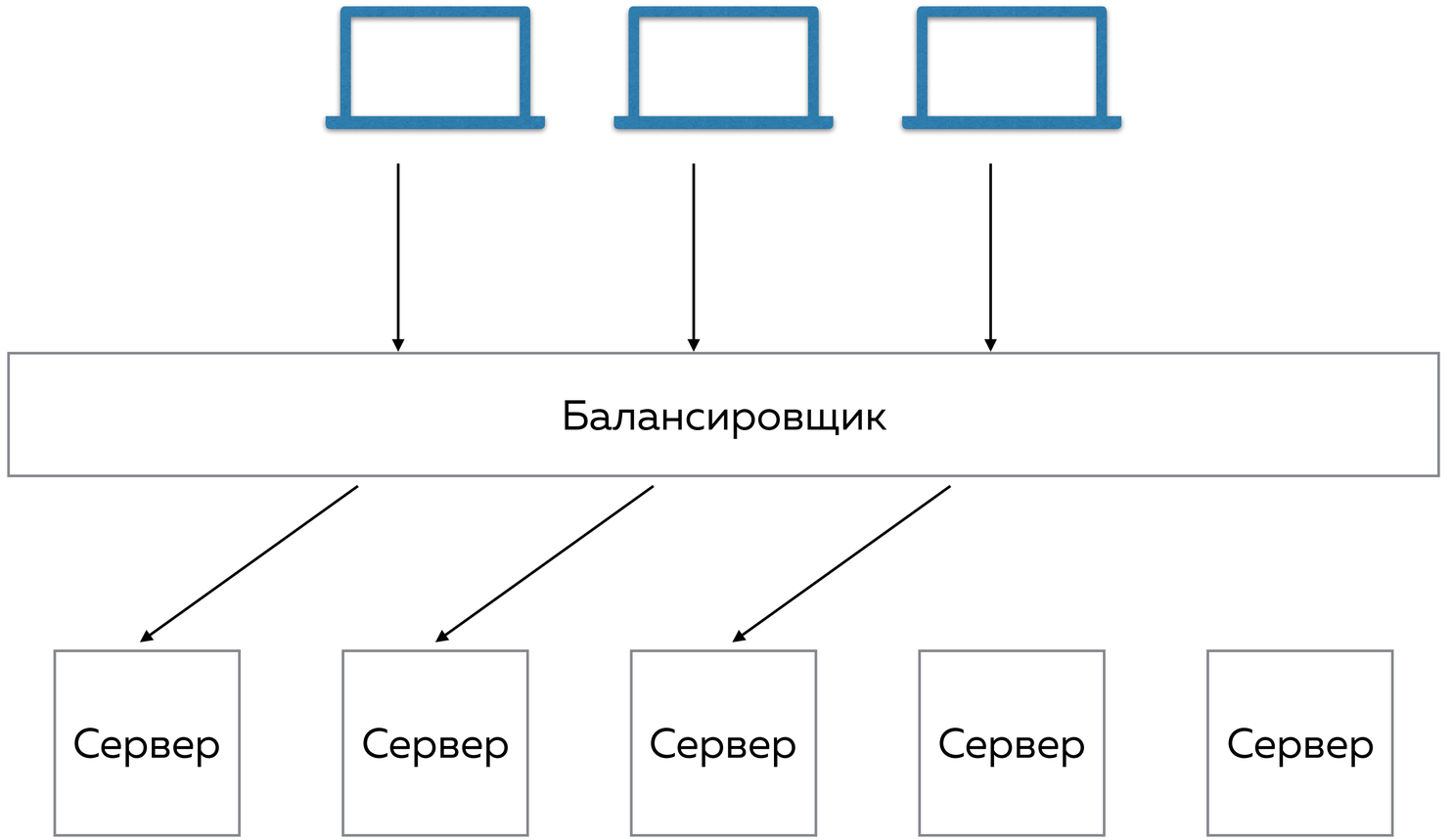

Балансировщик
Сервер
Сервер

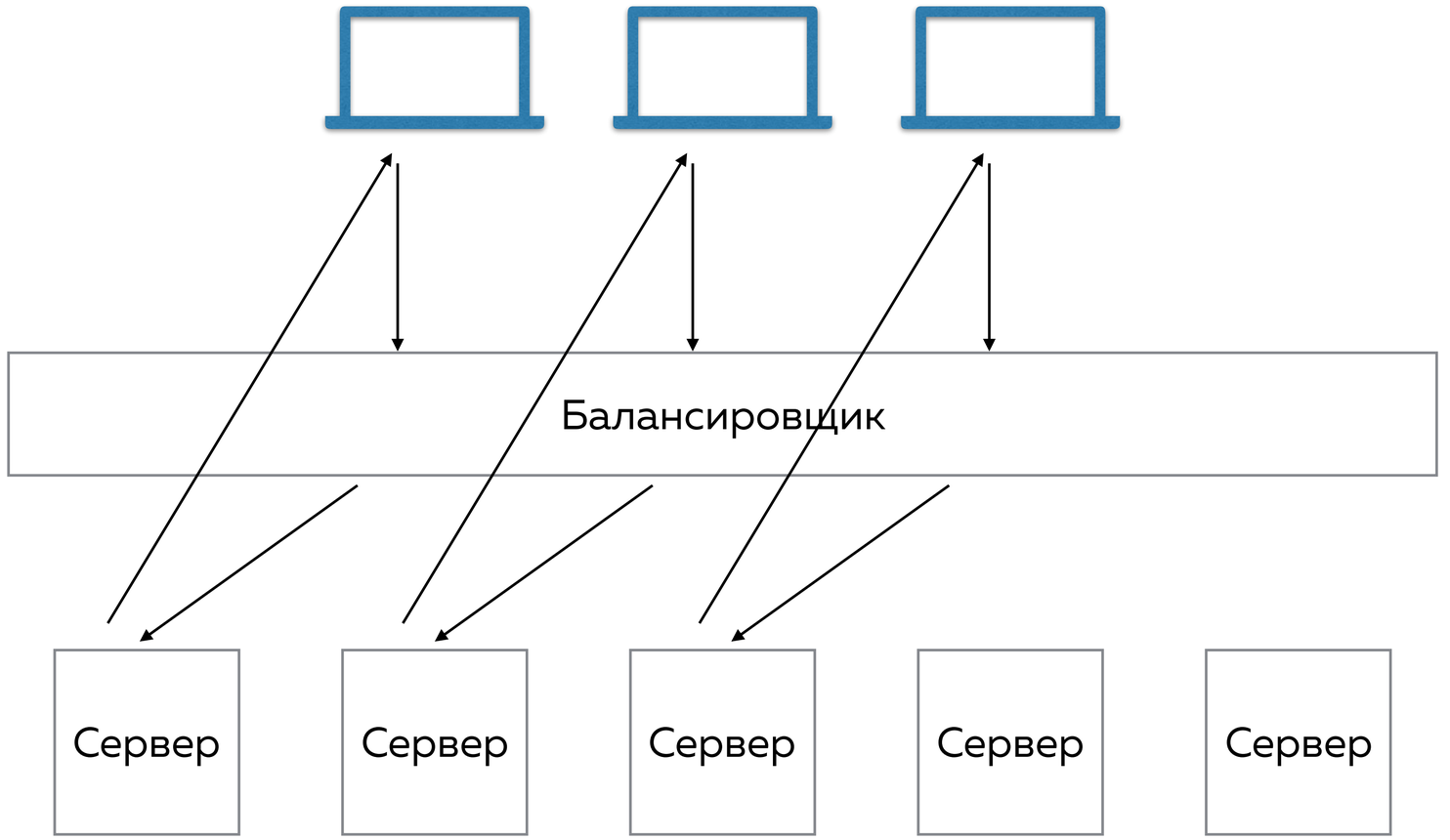

Балансировщик
Сервер
Сервер

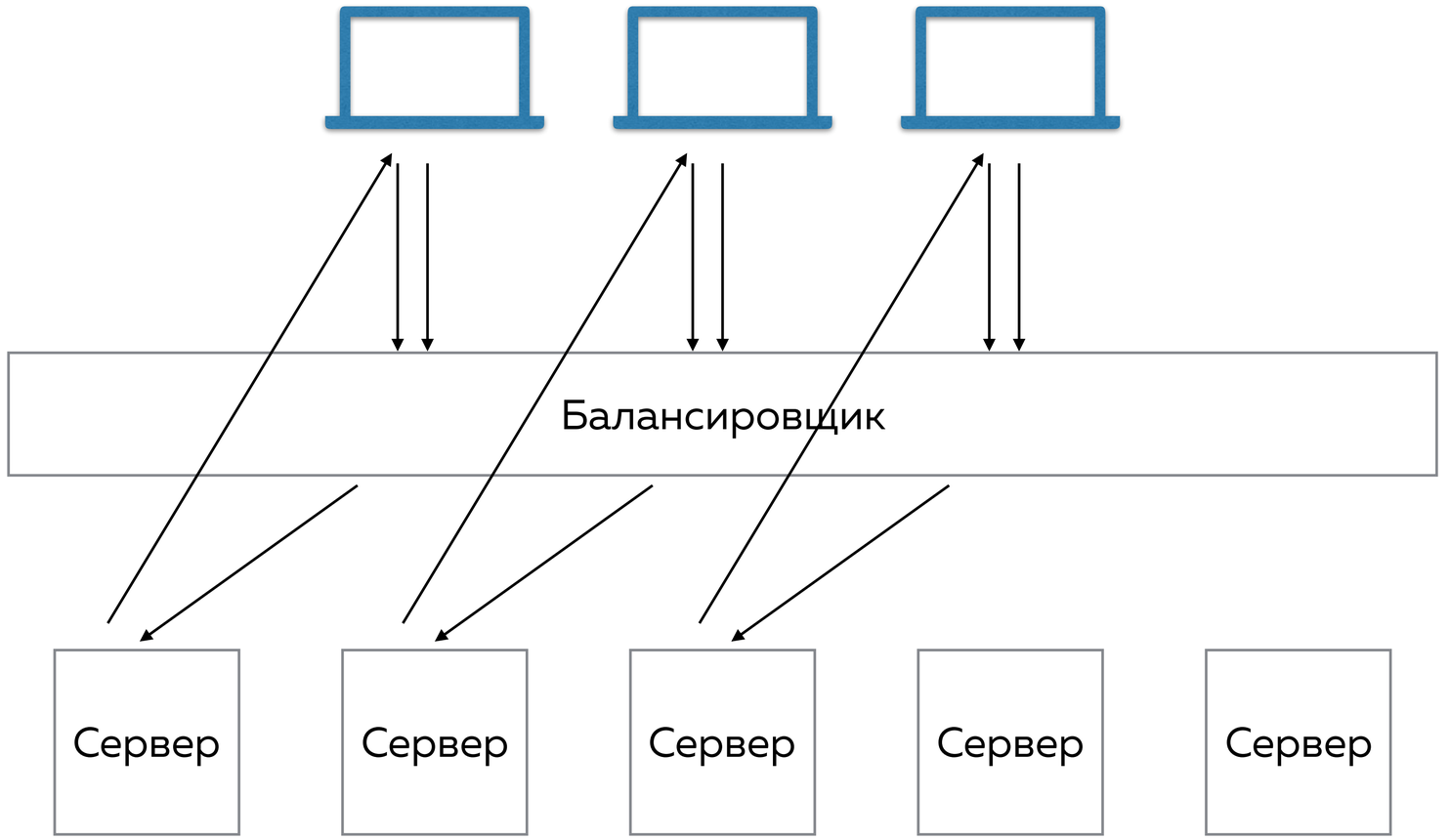
Балансировщик

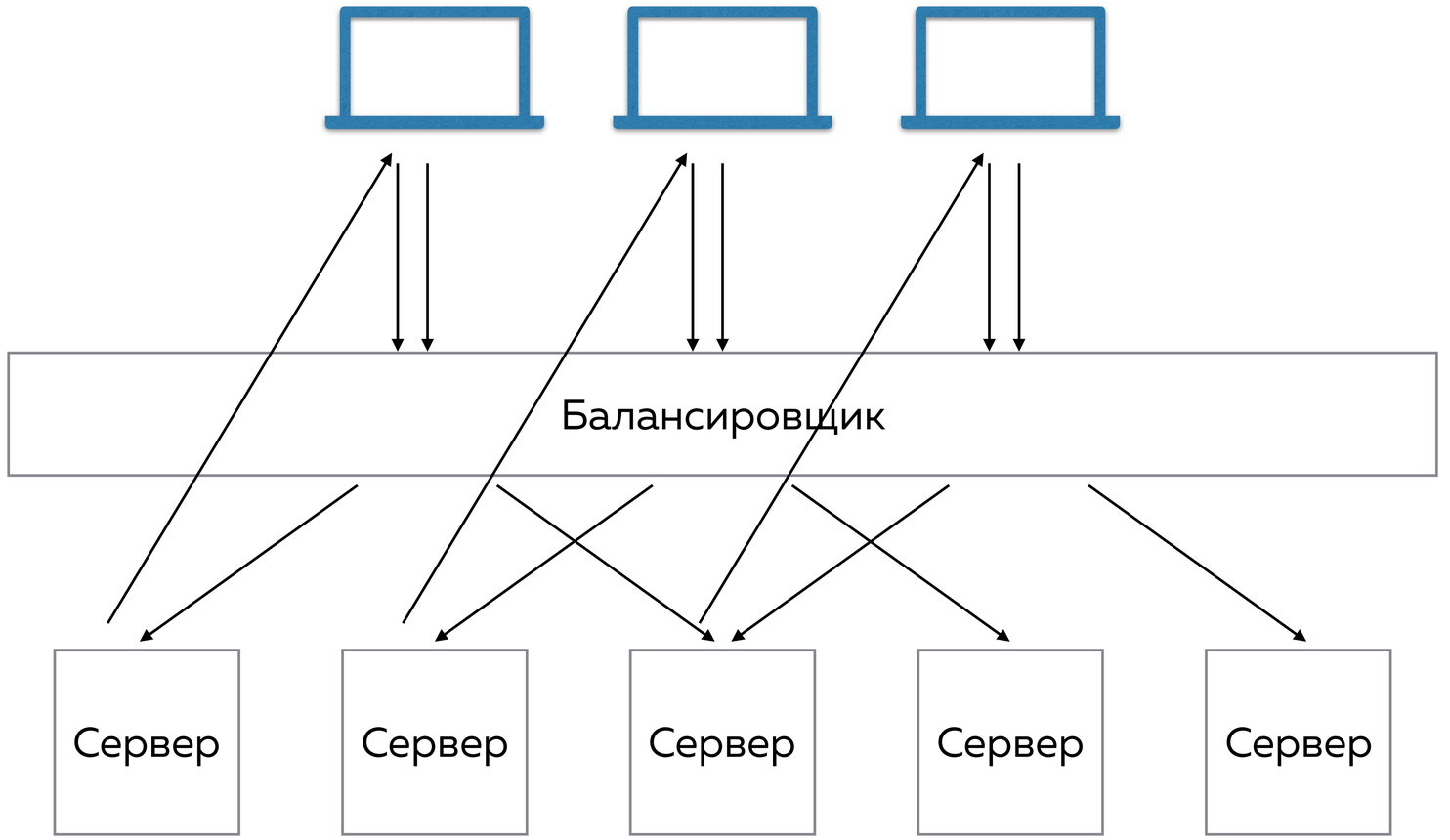
Сервер | Сервер | Сервер | Сервер | Сервер

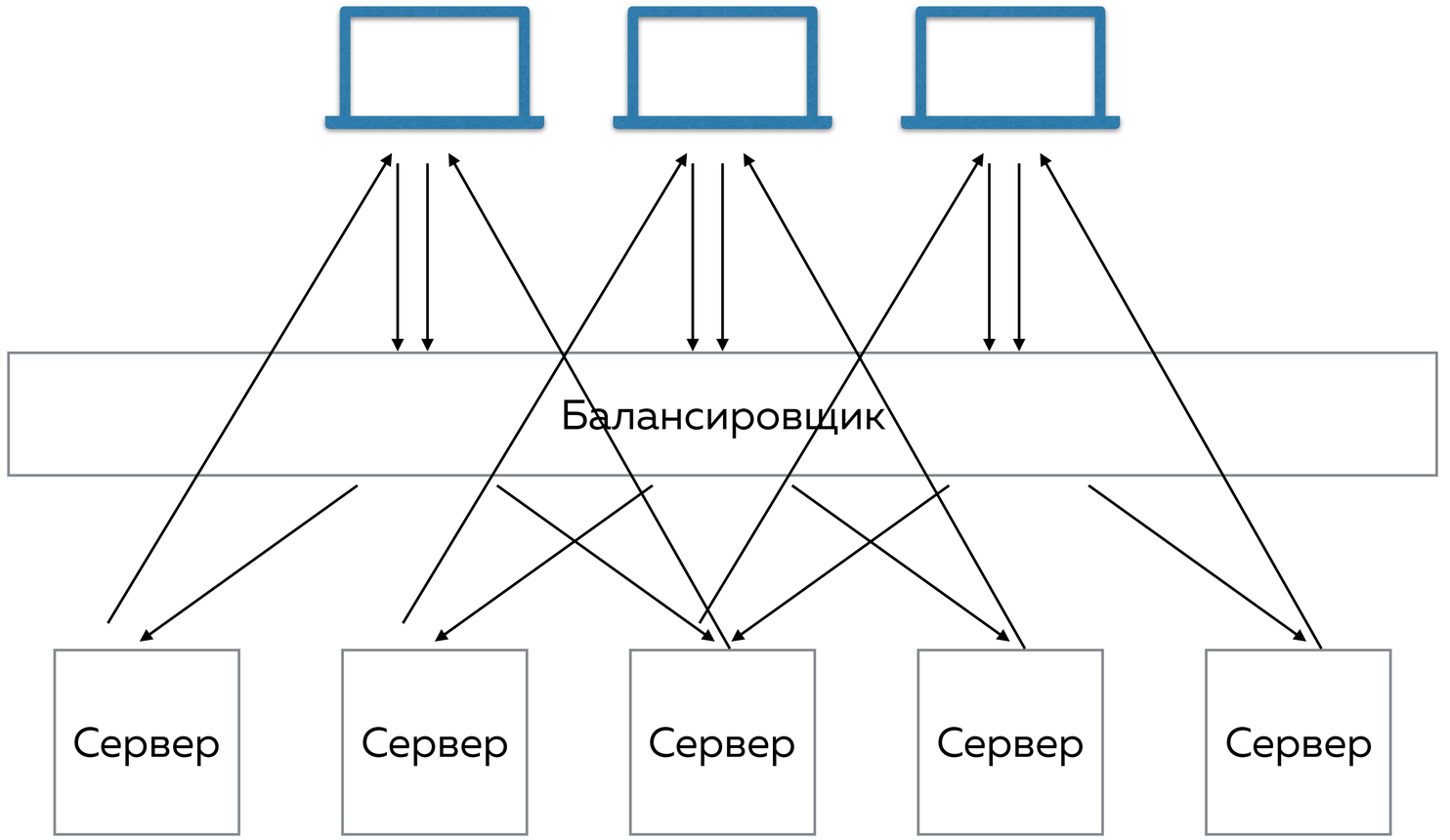

Балансировщик
Сервер
Сервер
Сервер
Сервер
Сервер


Балансировщик
Сервер
Сервер
Сервер
Сервер
Сервер


Балансировщик
Сервер
Сервер
Сервер
Сервер
Сервер


Балансировщик
Сервер
Сервер
Сервер
Сервер
Сервер


Балансировщик
Сервер
Сервер
Сервер
Сервер
Сервер


Балансировщик
Сервер
Сервер
Сервер
Сервер
Сервер

Модуль *cluster*

Встроенный модуль, позволяет выносить трудоёмкие вычисления в отдельный процесс, а так же вручную организовывать кластер

Кэш

промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью

Когда пользователь заходил на сайт

Балансировщик

Сервер

Сервер

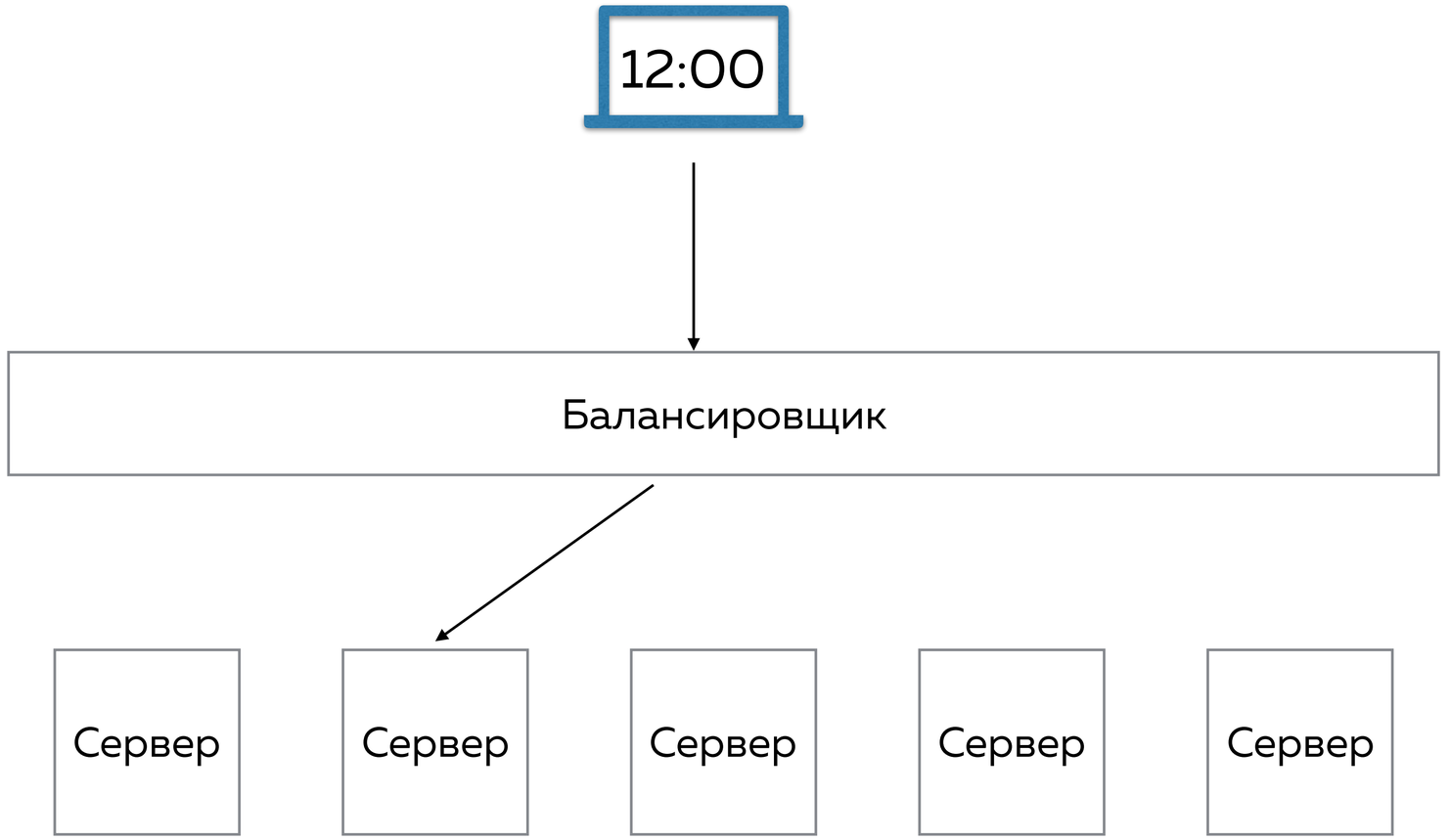
Сервер

Сервер

Сервер

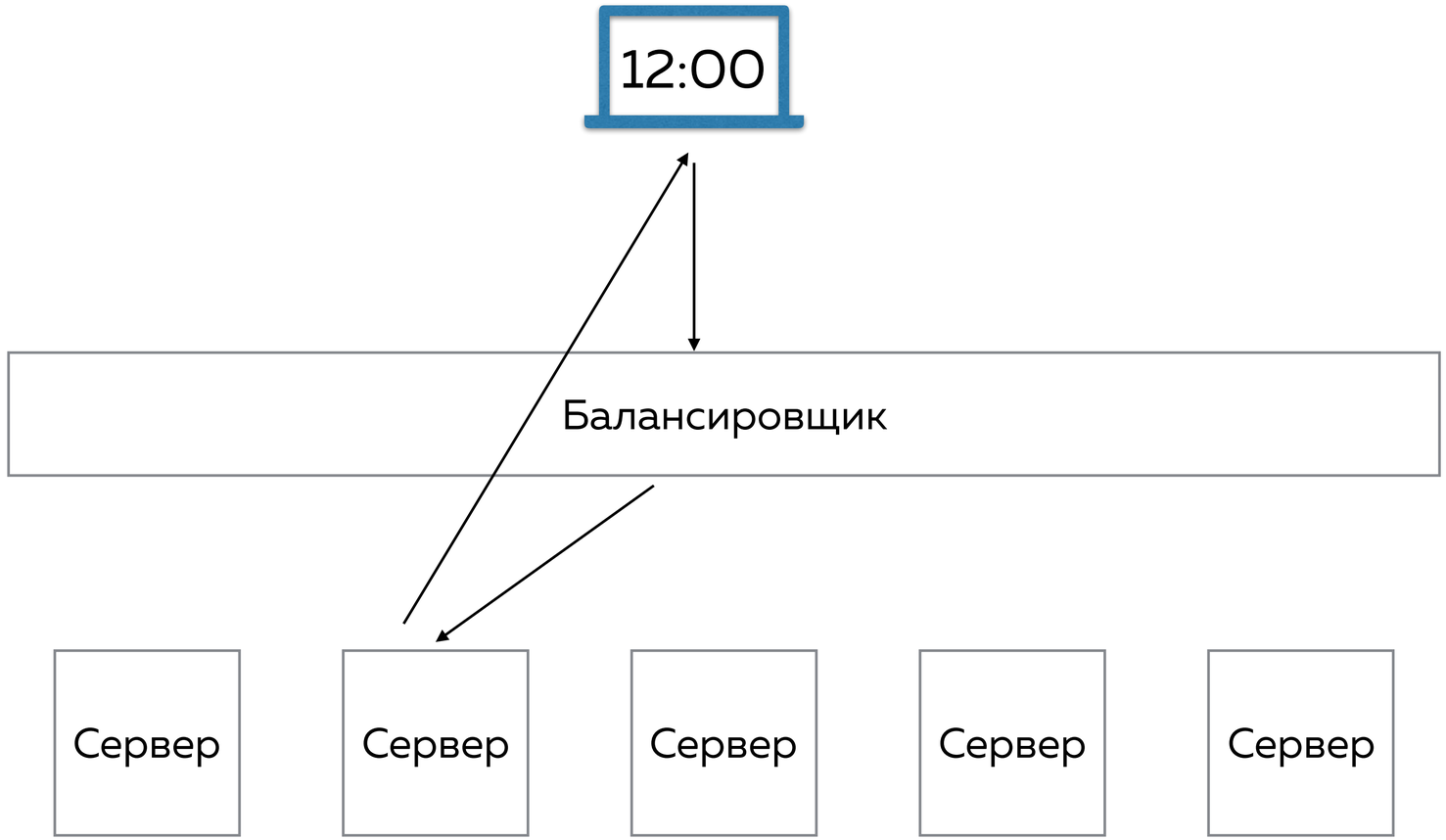
Когда пользователь заходил на сайт

Когда пользователь заходил на сайт



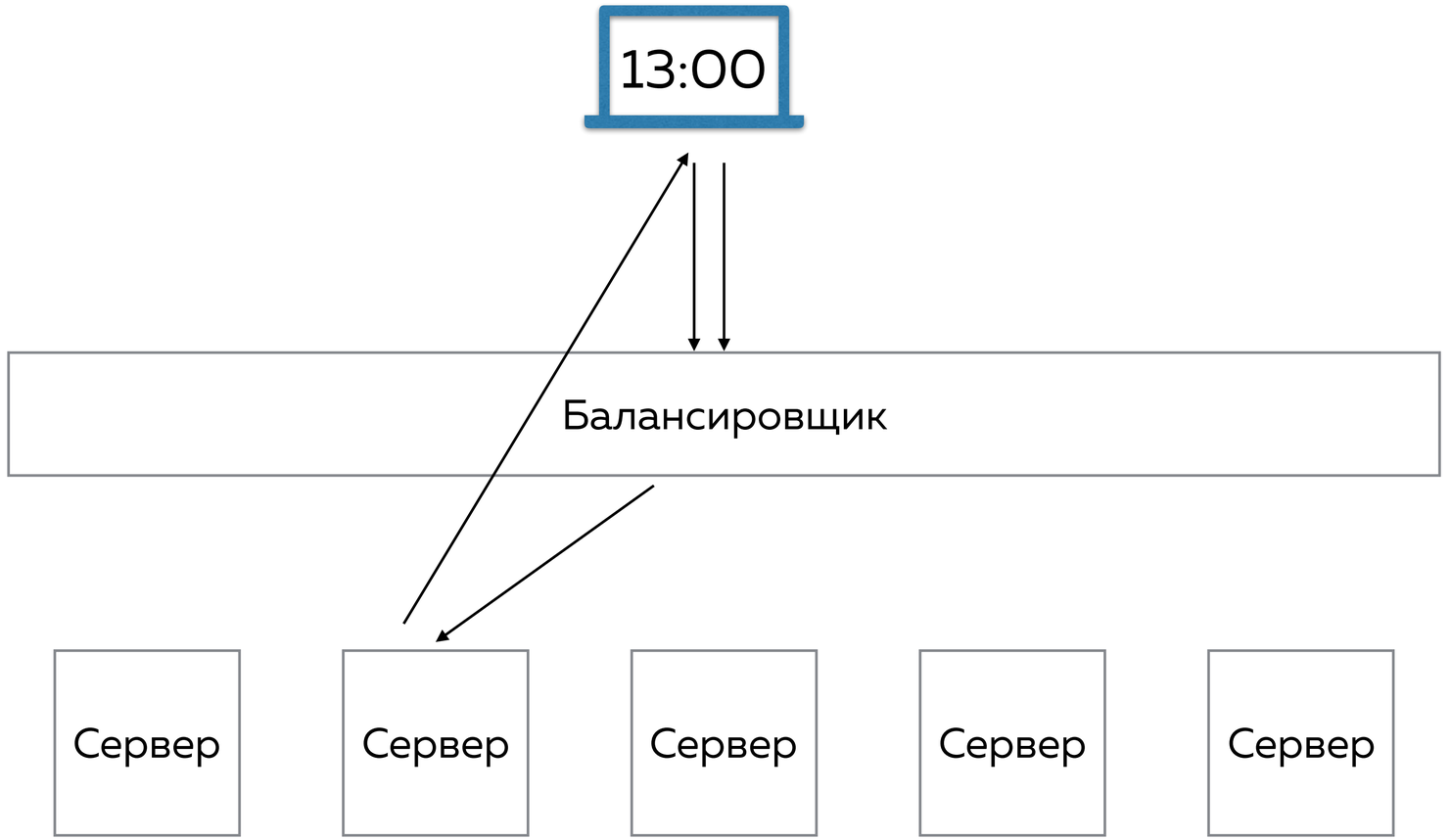
До этого никогда

Когда пользователь заходил на сайт



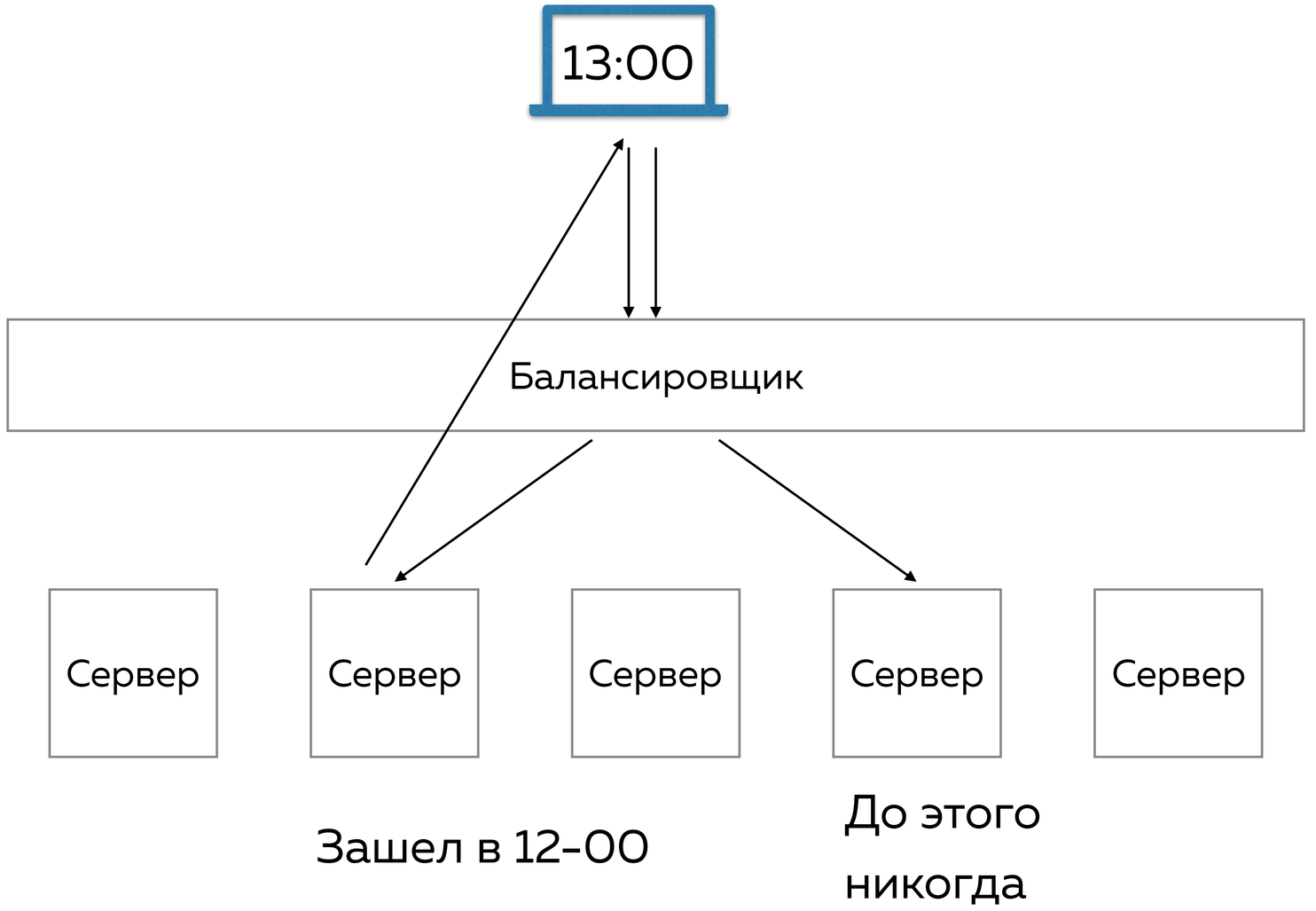
Зашел в 12-00

Когда пользователь заходил на сайт

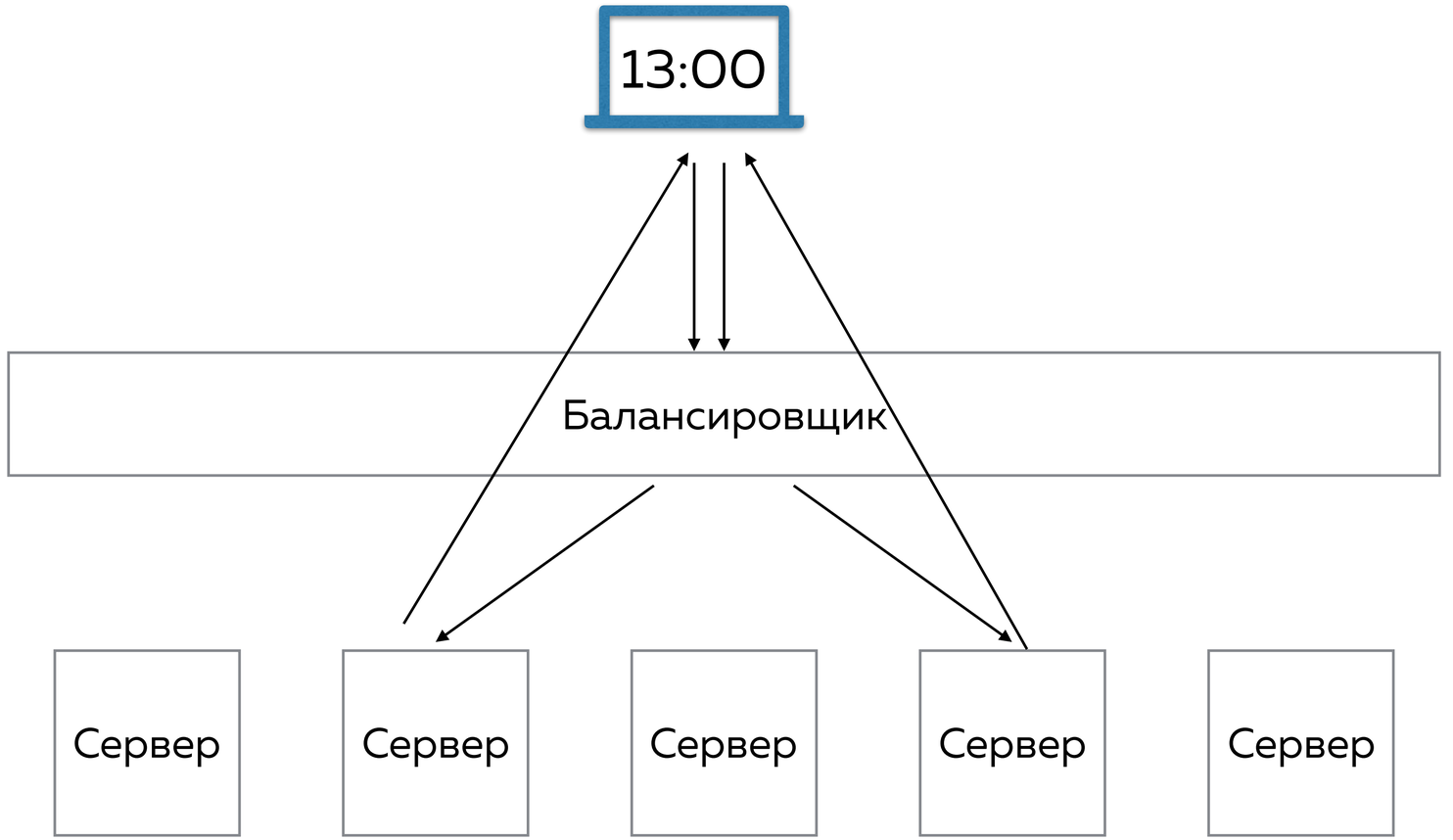


Зашел в 12-00

Когда пользователь заходил на сайт



Когда пользователь заходил на сайт



Зашел в 12-00

Зашел в 13-00

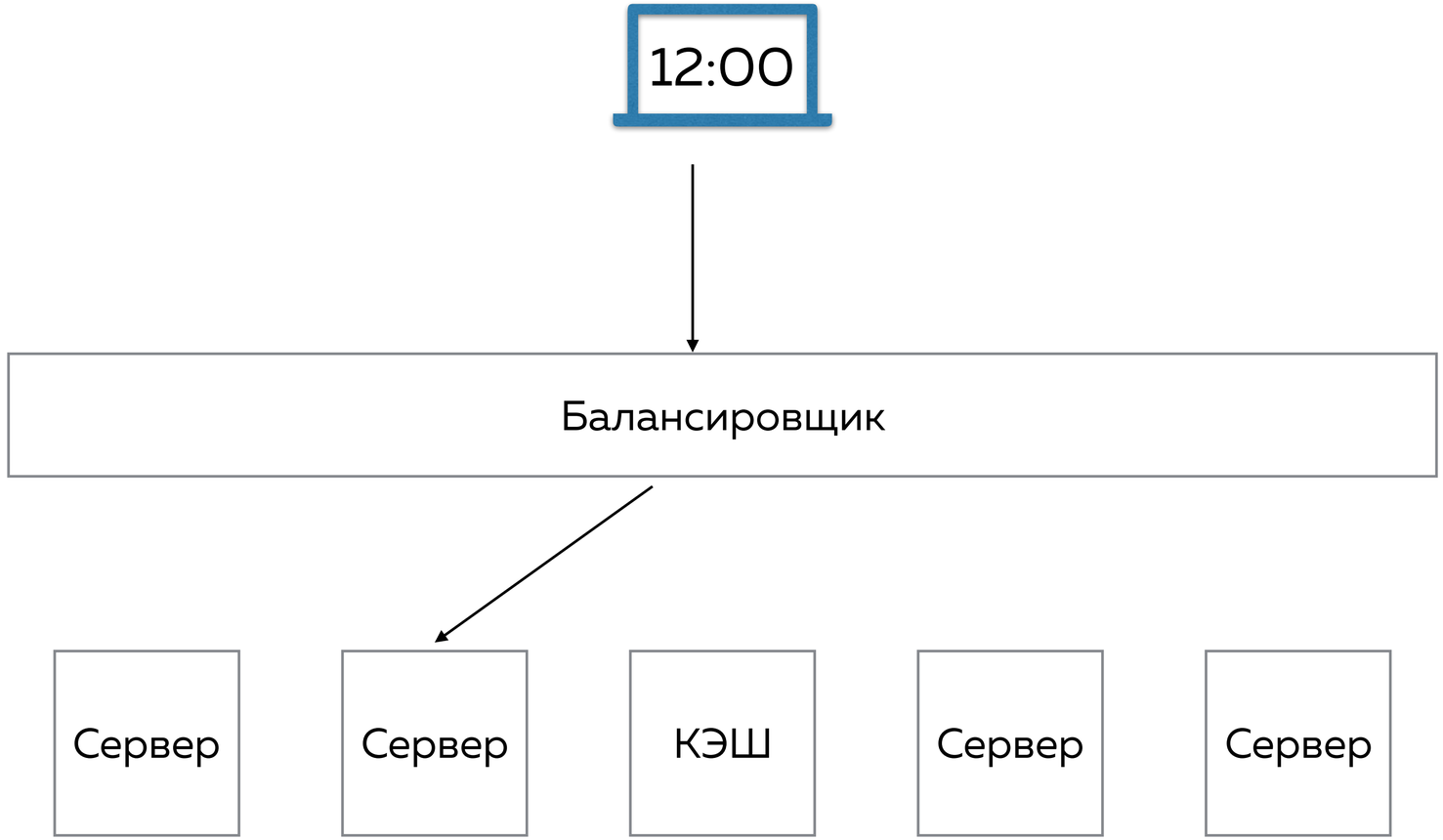
Где можно хранить промежуточную информацию

Балансировщик

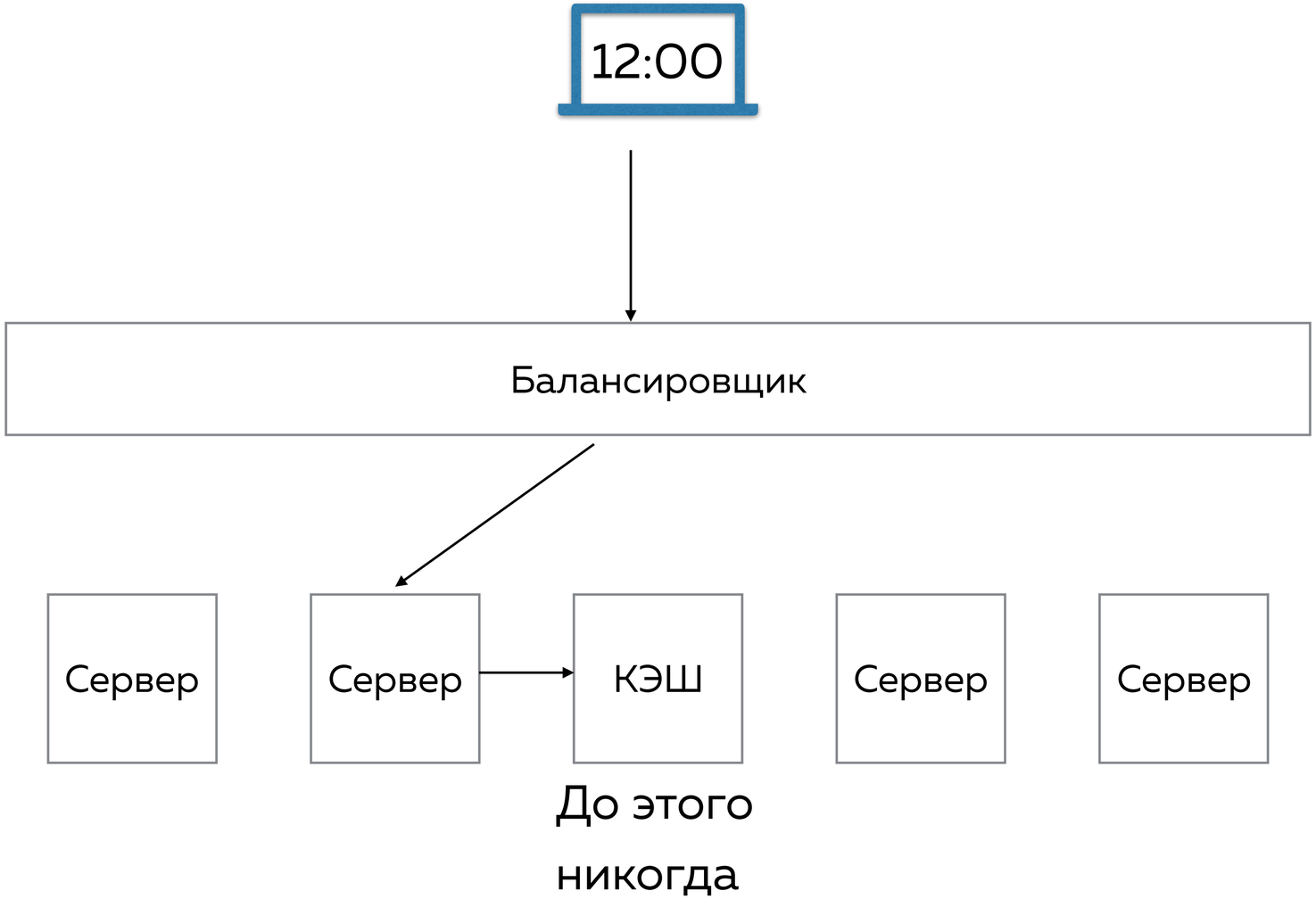
Сервер | Сервер | КЭШ | Сервер | Сервер

Где можно хранить промежуточную информацию

Где можно хранить промежуточную информацию

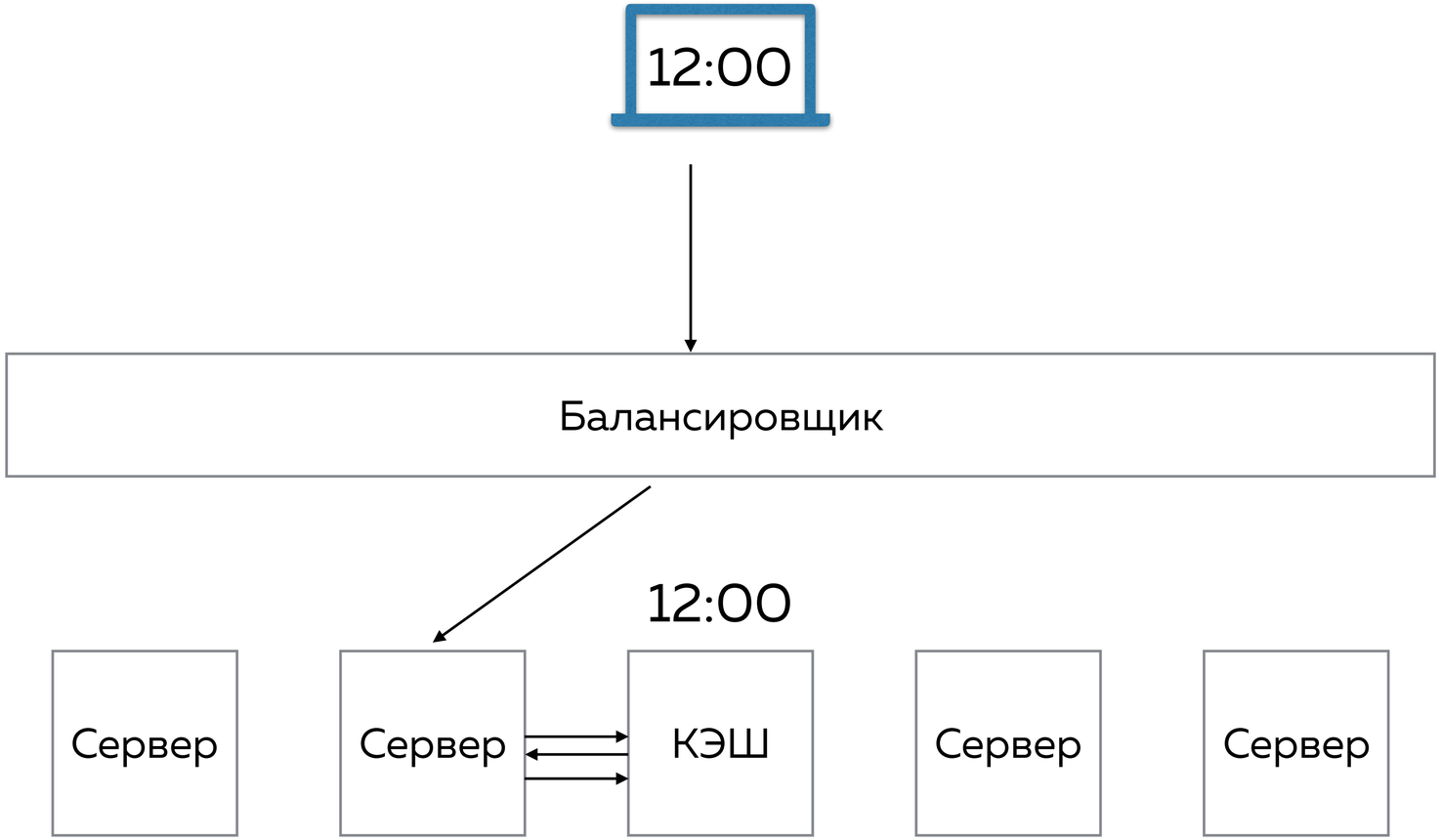


Где можно хранить промежуточную информацию



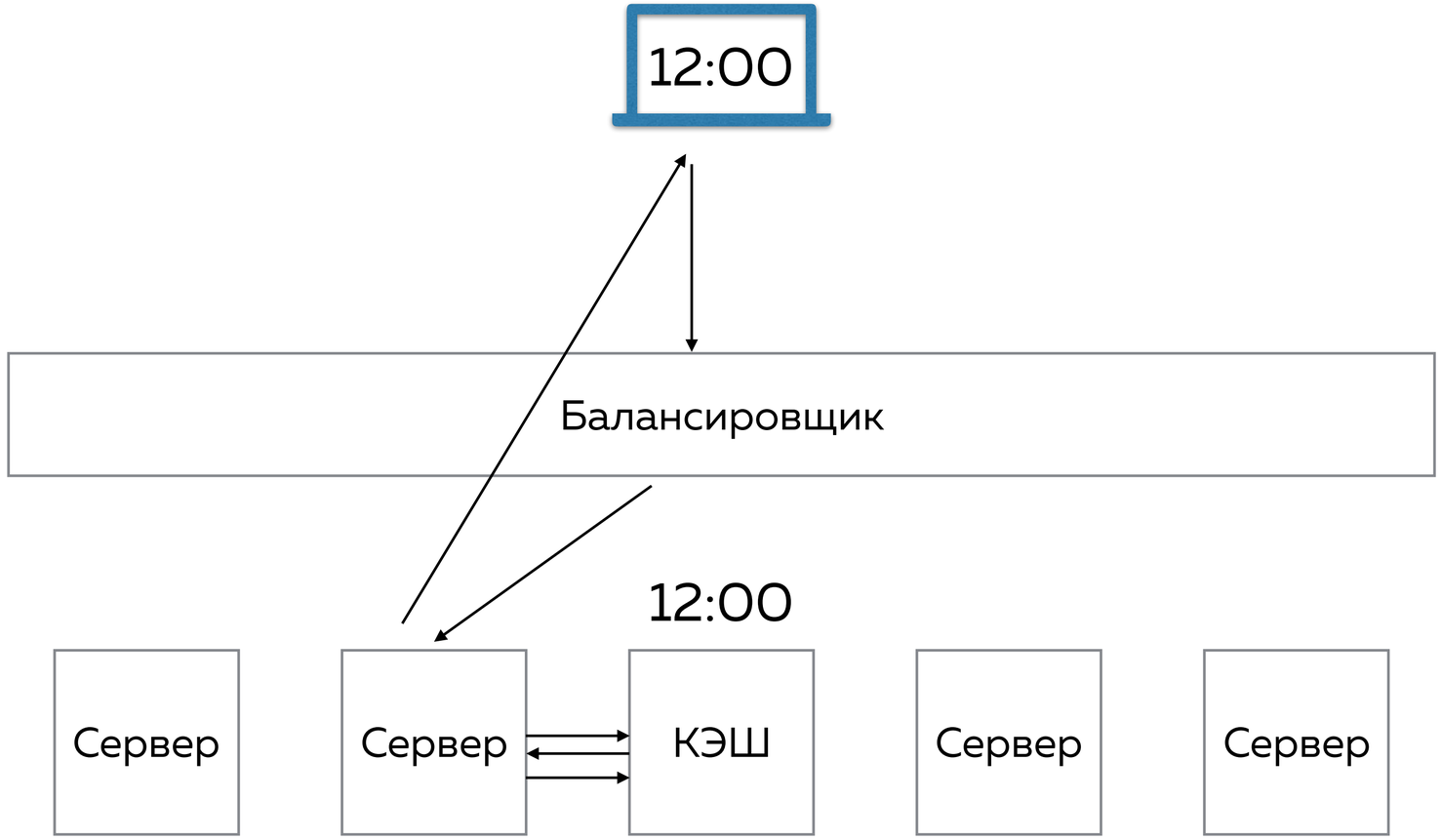
До этого никогда

Где можно хранить промежуточную информацию



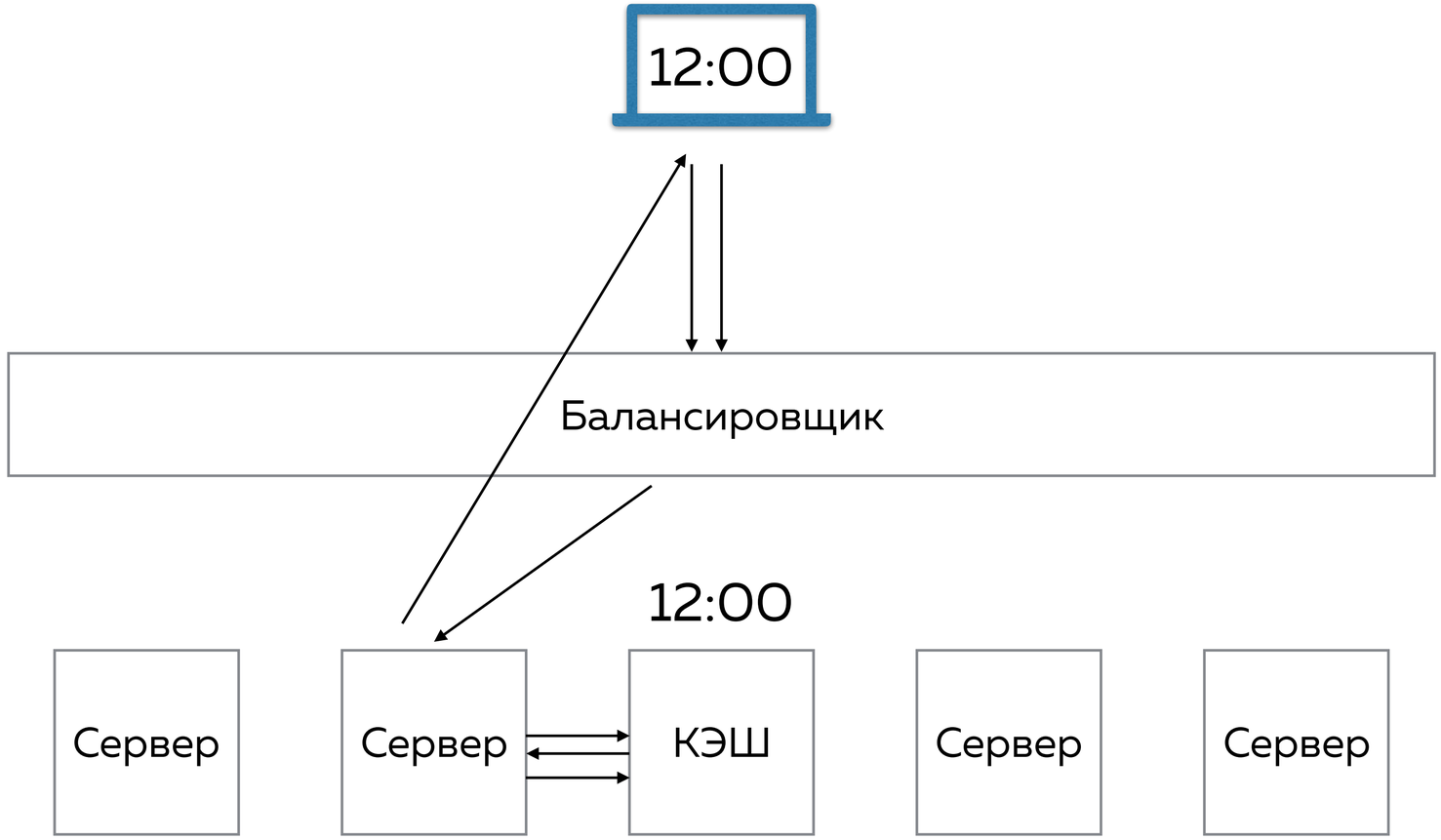
Зашел в 12-00

Где можно хранить промежуточную информацию



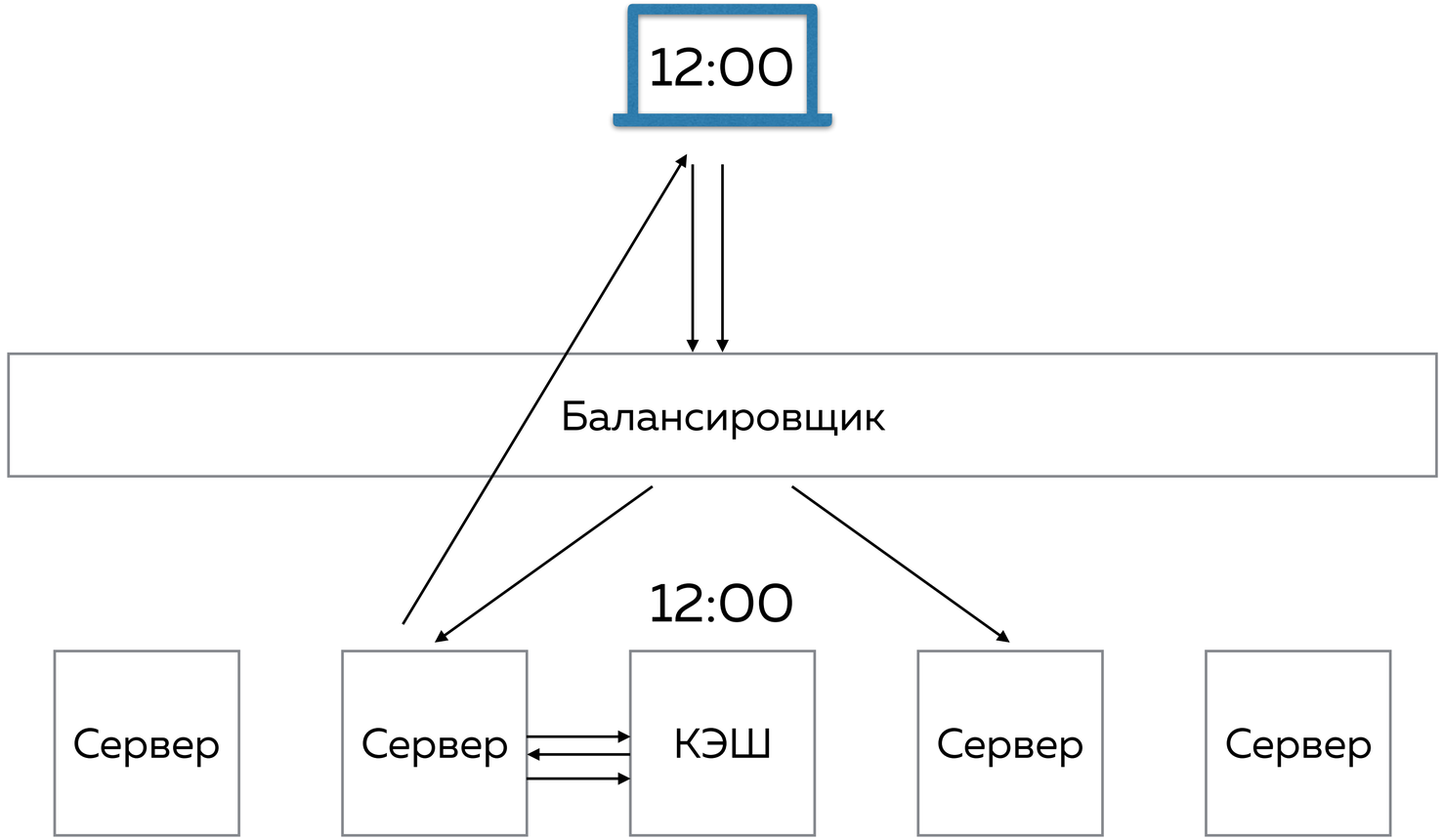
Зашел в 12-00

Где можно хранить промежуточную информацию



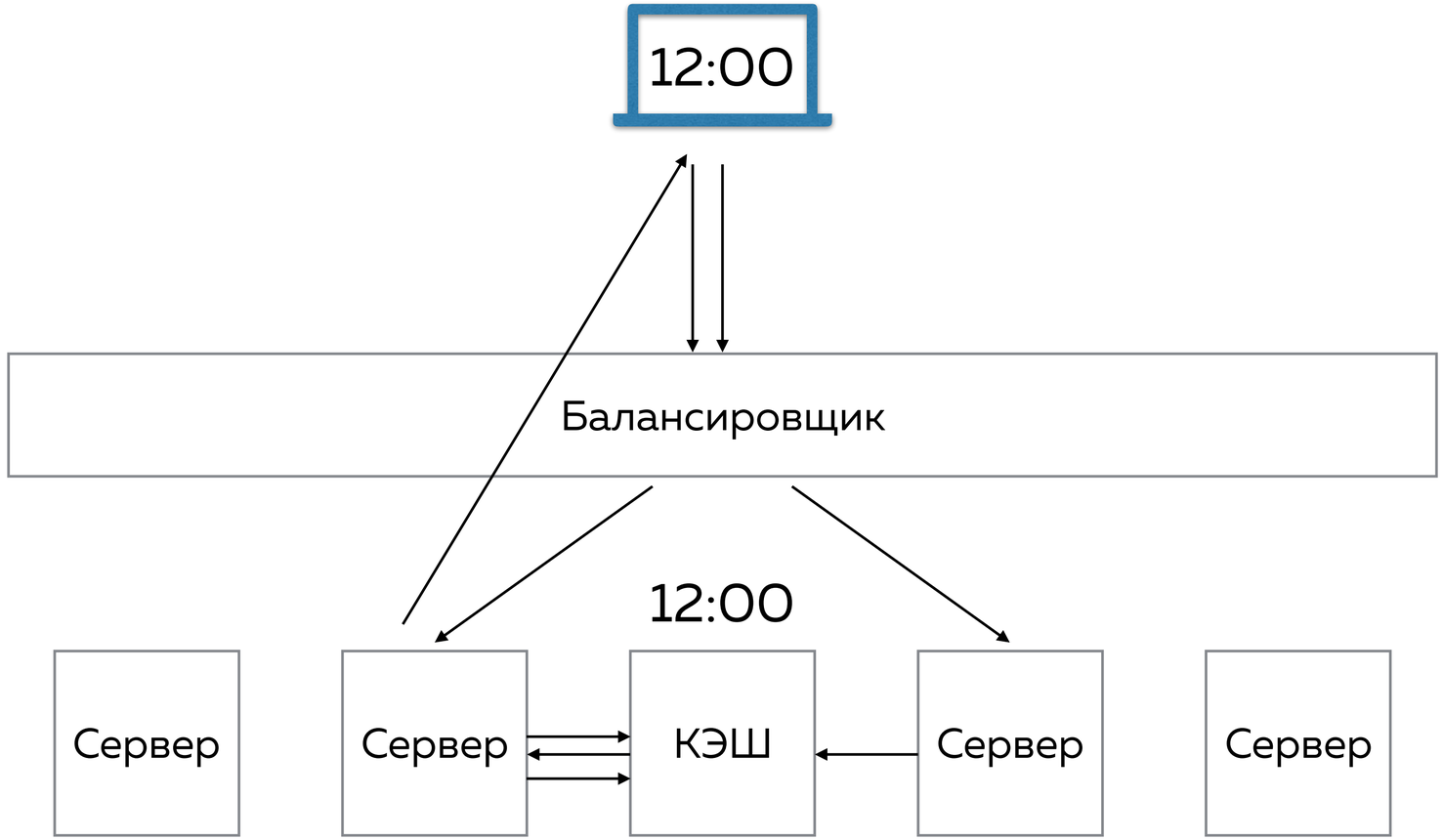
Зашел в 12-00

Где можно хранить промежуточную информацию



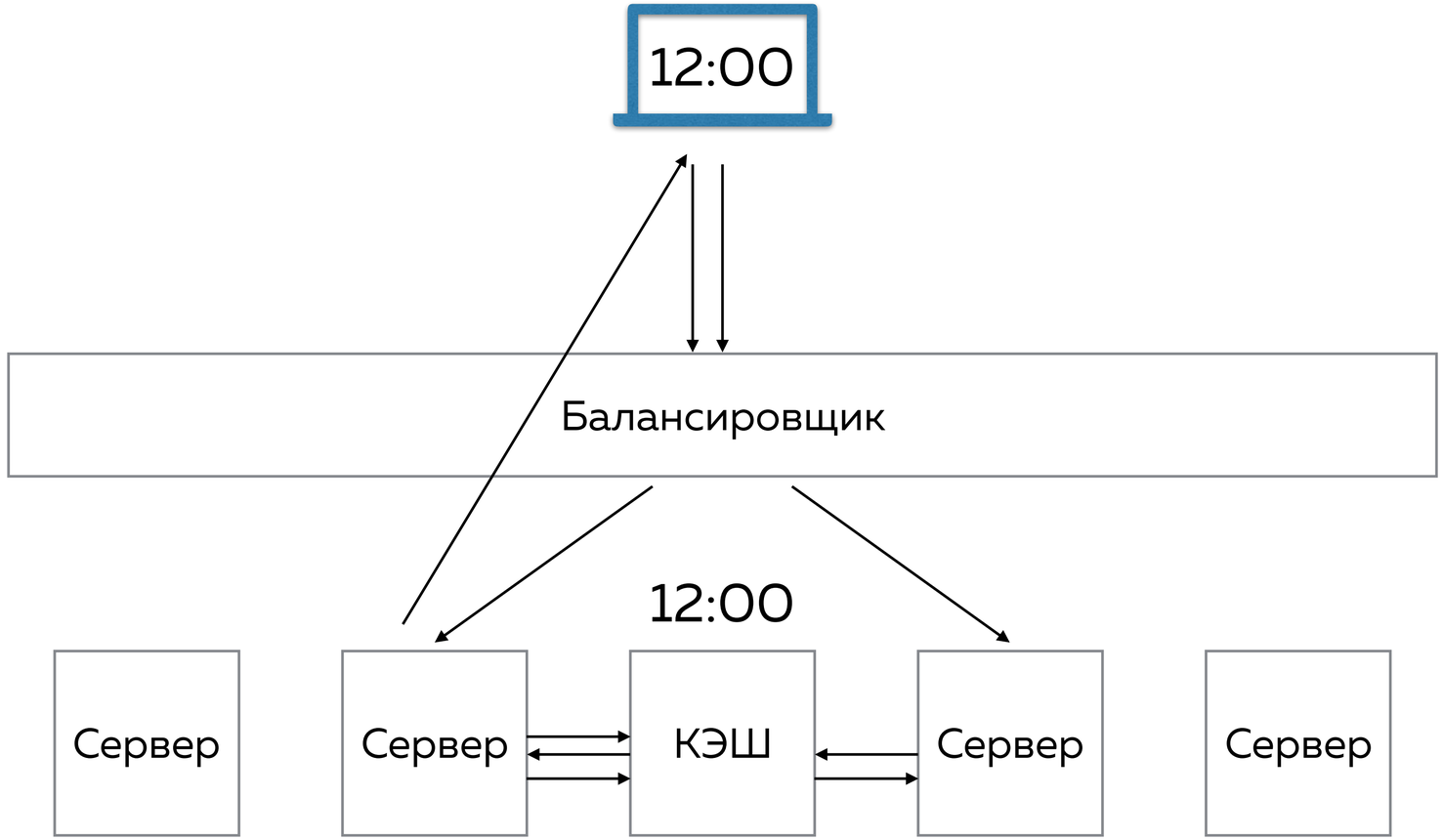
Зашел в 12-00

Где можно хранить промежуточную информацию



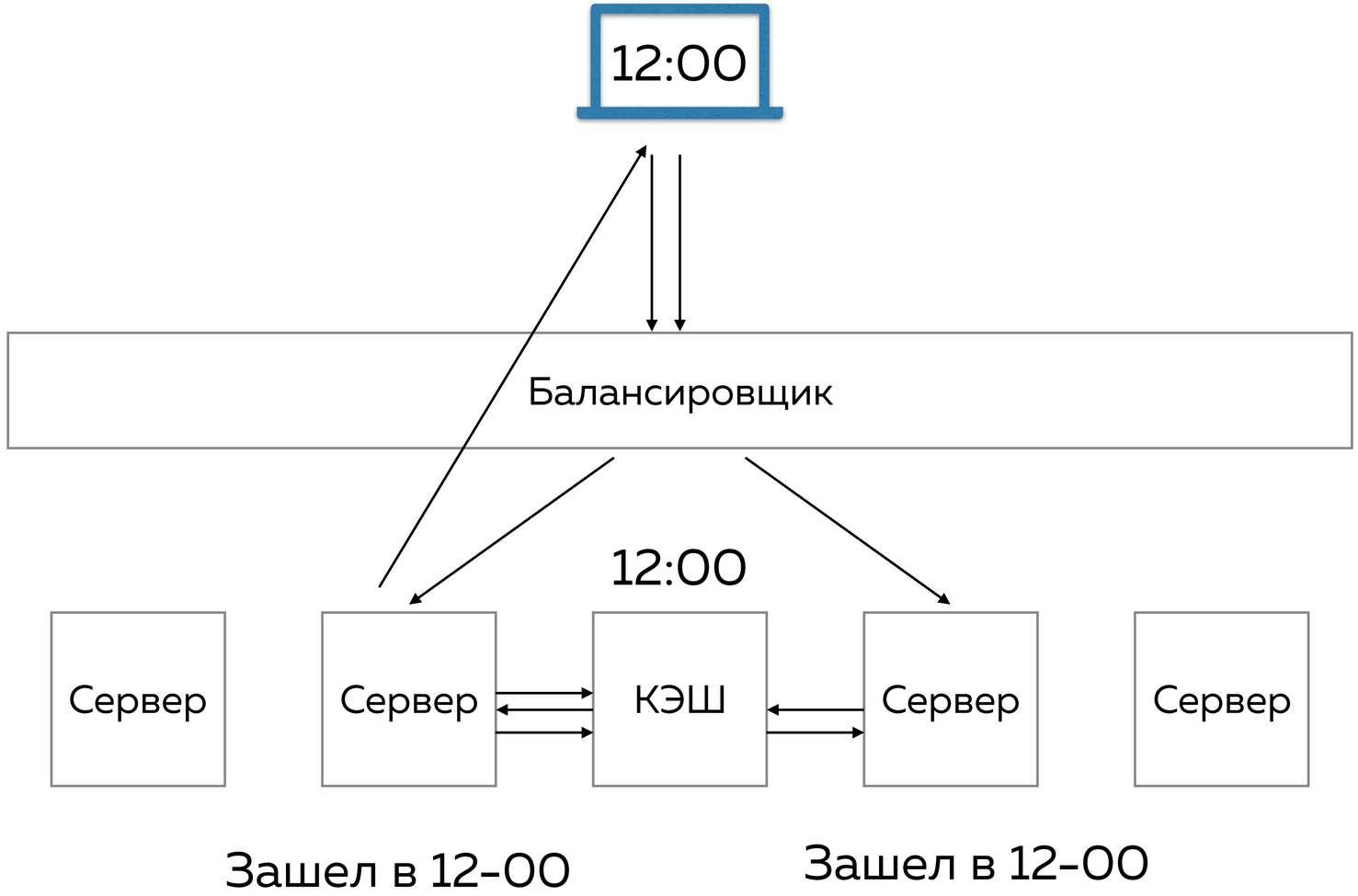
Зашел в 12-00

Где можно хранить промежуточную информацию

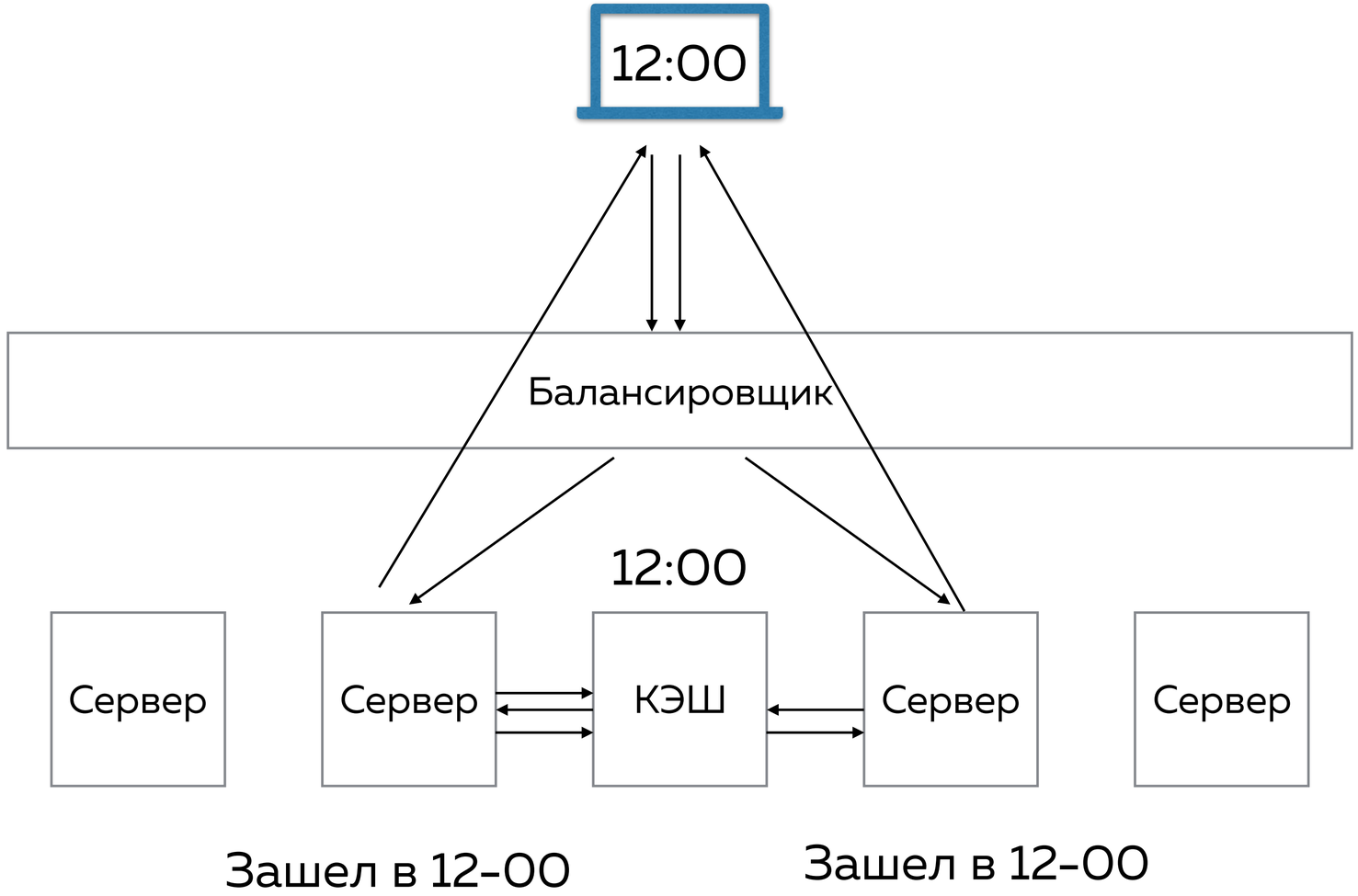


Зашел в 12-00

Где можно хранить промежуточную информацию



Где можно хранить промежуточную информацию



Состояние

набор устойчивых значений переменных параметров системы

“Первоочередное условие для успешного горизонтального масштабирование — избавиться от состояния в памяти”

Где хранятся состояния

Где хранятся состояния

– В базе данных

  Такое состояние можно передавать между разными серверами/сервисами и хранить его неопределённо долгое время

Где хранятся состояния

– В базе данных
  Такое состояние можно передавать между разными серверами/сервисами и хранить его неопределённо долгое время

– В кэше
  Такое состояние можно передавать между разными серверами/сервисам и хранить/изменять и обновлять его можно оперативно

Где хранятся состояния

- В базе данных
  Такое состояние можно передавать между разными серверами/сервисами и хранить его неопределённо долгое время

- В кэше
  Такое состояние можно передавать между разными серверами/сервисам и хранить/изменять и обновлять его можно оперативно

- На жёстком диске (данные — тоже хранят какое-то фиксированное состояние или результат)
  Такое состояние можно передавать между процессами одной машины

Где хранятся состояния

- В базе данных
  Такое состояние можно передавать между разными серверами/сервисами и хранить его неопределённо долгое время

- В кэше
  Такое состояние можно передавать между разными серверами/сервисам и хранить/изменять и обновлять его можно оперативно

- На жёстком диске (данные — тоже хранят какое-то фиксированное состояние или результат)
  Такое состояние можно передавать между процессами одной машины

- В памяти
  Такое состояние принадлежит только одному процессу и передать его нельзя

Очередь сообщений (*Message Queue*)

программно-инженерный компонент, используемый для межпроцессного или межпотокового взаимодействия внутри одного процесса.
Для обмена сообщениями используется очередь.

Брокер сообщений (Message Broker)

– Redis

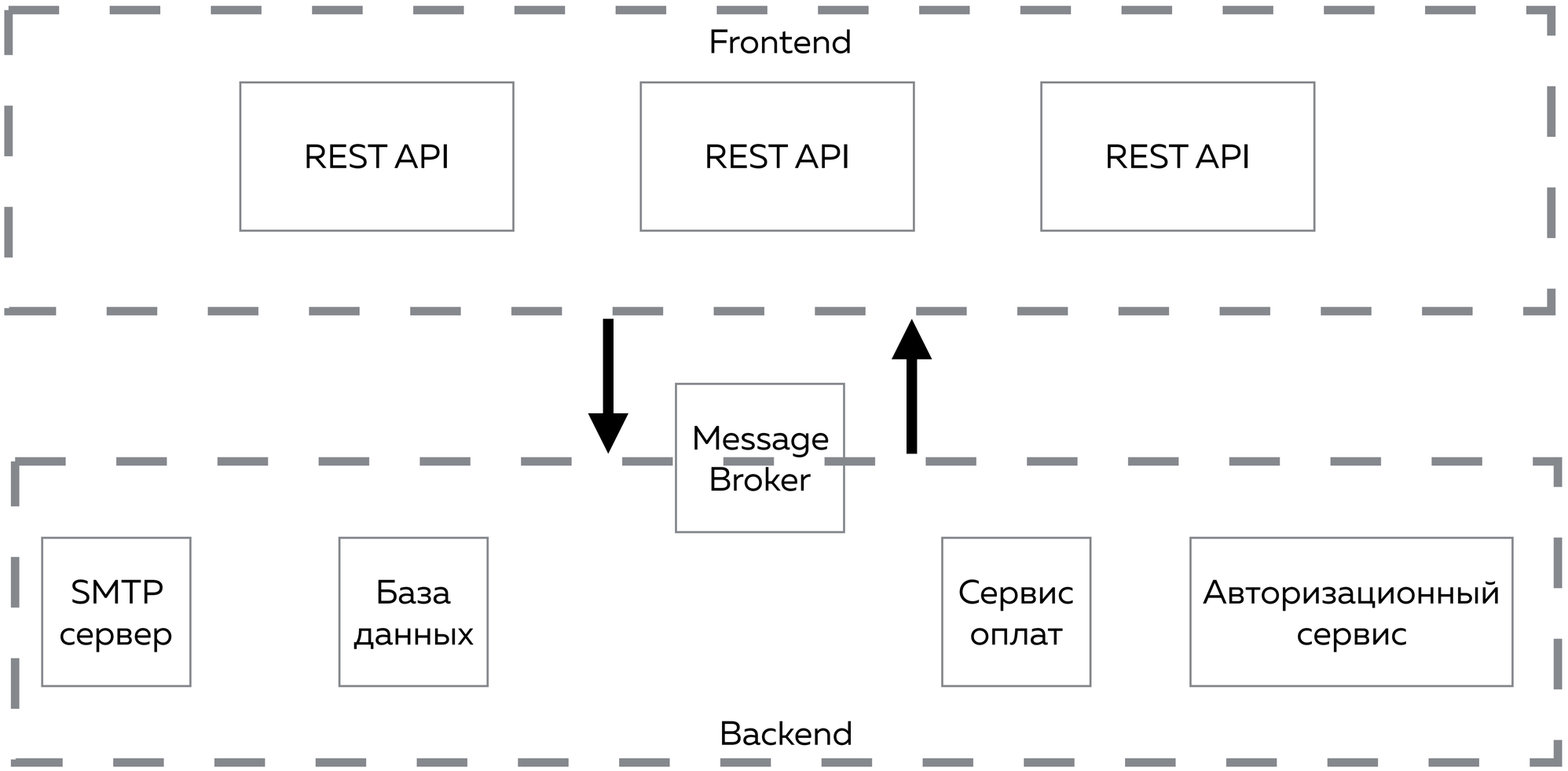
– RabbitMQ

– Apache ActiveMQ

– И прочие

en.wikipedia.org/wiki/Message_broker

Экосистема *Node.js*



12 факторов

– Приложение и настройки лежат в одном репозитории

– Зависимости всегда зафиксированы

– Настройки передаются как переменные окружения среды

– Все сервисы являются равноправными участниками кластера

– Сборка и релиз всегда связаны

– Горизонтальное масштабирование по построению

– Приложение не имеет своего внутреннего состояния

– ...

https://12factor.net/

Преимущества *Node.js* приложений

Преимущества *Node.js* приложений

– Легковесность

  Создать/опустить новый процесс ничего не стоит

Преимущества *Node.js* приложений

– Легковесность

  Создать/опустить новый процесс ничего не стоит

– Однопоточность

  Природа JavaScript позволяет не думать о проблеме многопоточности

Преимущества *Node.js* приложений

- Легковесность

  Создать/опустить новый процесс ничего не стоит

- Однопоточность

  Природа JavaScript позволяет не думать о проблеме многопоточности

- Заменяемость

  Любой компонент системы может быть безболезненно заменён или перенесён на любое кол-во серверов

```
10 PRINT "さようなら！"
20 END
```