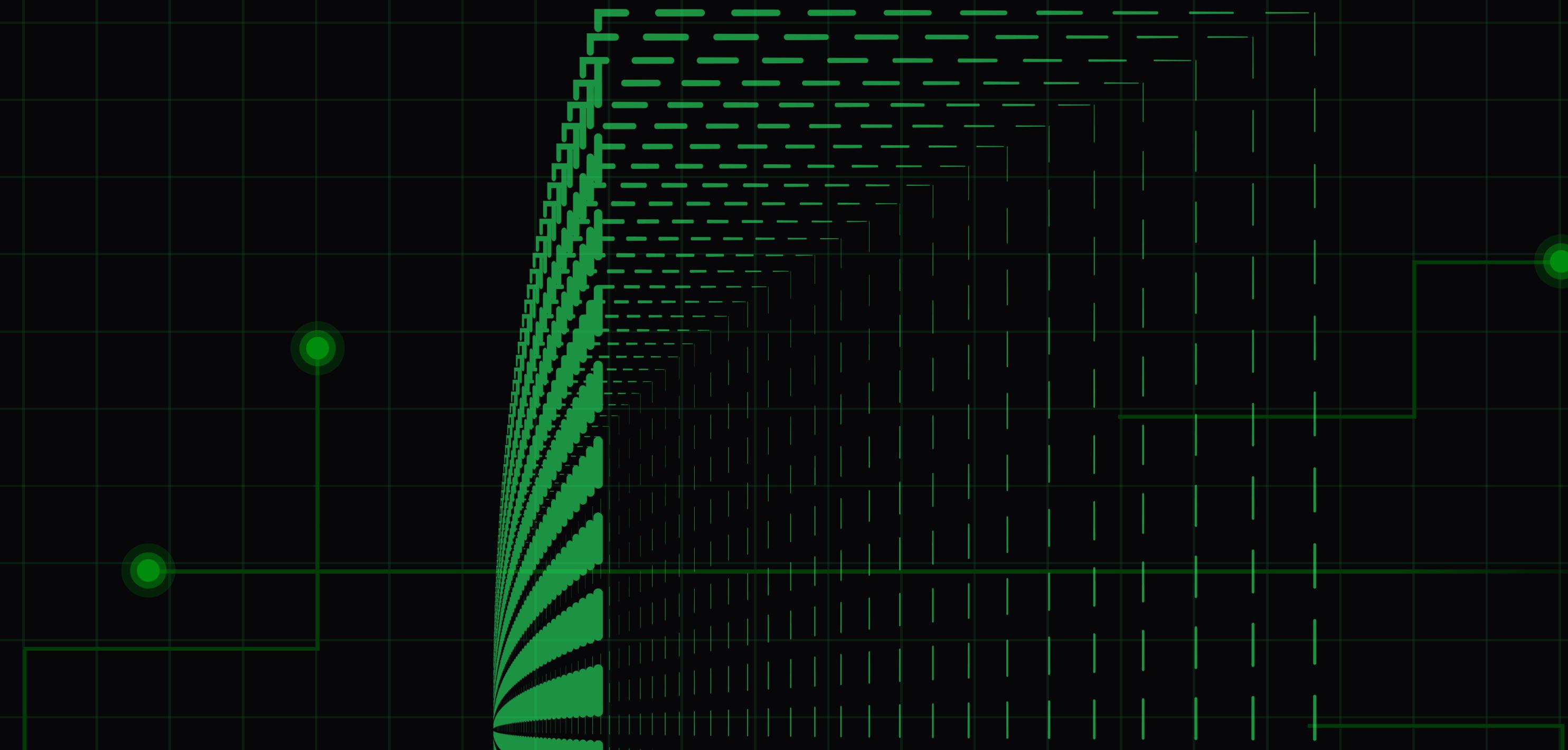
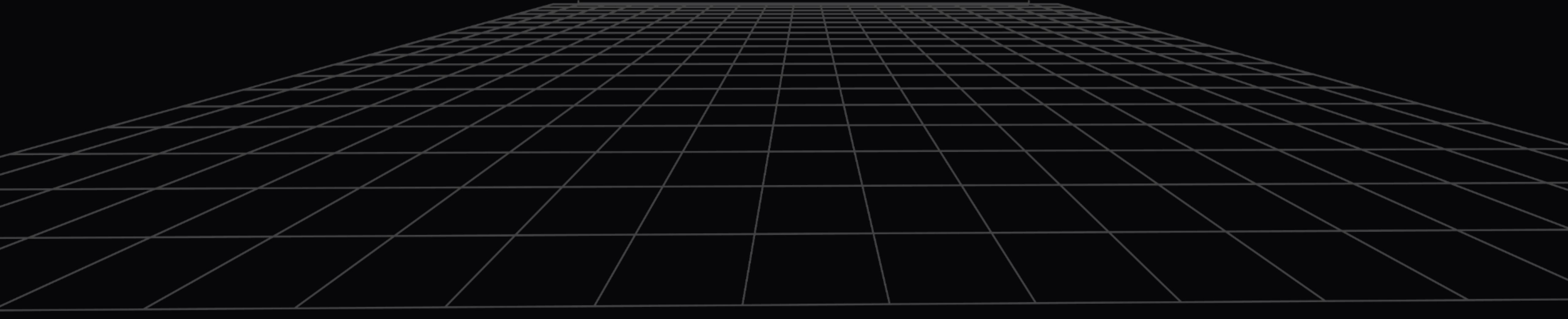


# SYSTEM DESIGN

Кэширование, API и Observability

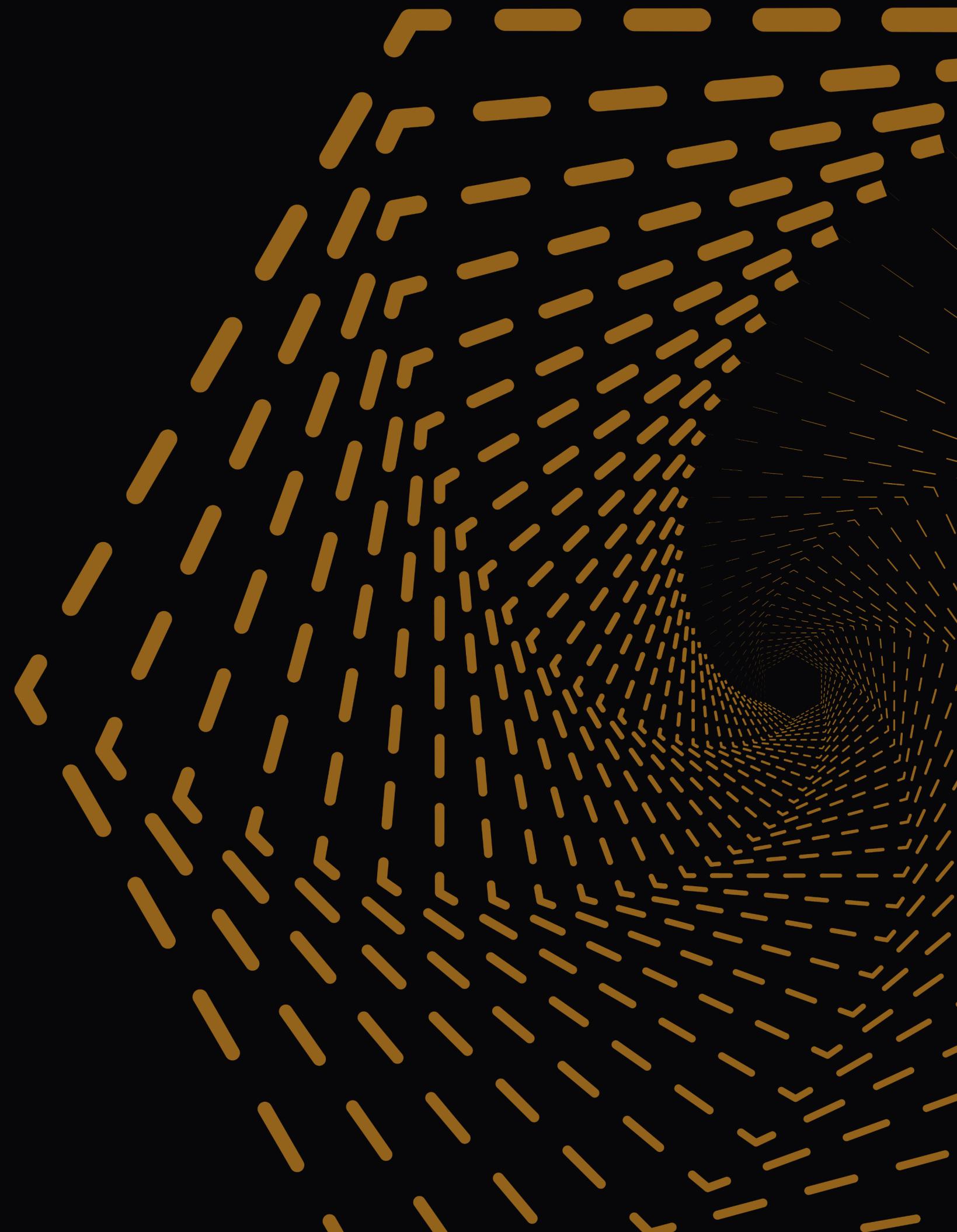


**ПРОВЕРЬ ЗАПИСЬ**



# ПРАВИЛА ЗАНЯТИЯ

1. вопросы в чате можно задавать  
**в любое время**
2. вопросы голосом задаем  
**по поднятой руке в Zoom**
3. ответы на вопросы будут  
**в запланированных местах**



# МАРШРУТ ЗАНЯТИЯ

1. Кэширование
2. API
3. Observability

# **КЭШИРОВАНИЕ**

# КЭШИРОВАНИЕ

---

- /1 Сокращение response time сервисов
  - /2 Снижение лишней нагрузки на сторонние сервисы
  - /3 Переиспользование ранее полученных или вычисленных данных
  - /4 Стабилизация работы при кратковременных отказах систем
-

# ОСНОВНЫЕ ТЕРМИНЫ

---

**Cache miss** промах кэша, запрошенный ключ не был найден в кэше

---

**Cache Hit** попадание в кэш, запрошенный ключ найден в кэше

---

**Hit ratio** процент попаданий запросов в кэш, характеризует эффективность кэширования

---

**Горячий ключ** ключ, на который приходится большая часть запросов

---

**Прогрев кэша** процесс наполнения кэша данными

---

**Инвалидация** удаление кэшированных данных

---

# КАКИЕ ДАННЫЕ КЭШИРОВАТЬ?

---

Меняются часто  
(секунды)

кэшировать чаще всего бессмысленно,  
но иногда может пригодиться

---

Меняются нечасто  
(минуты и часы)

чаще всего здесь задаемся вопросом  
– «стоит ли мне кэшировать»

---

Меняются редко  
(дни, недели, месяцы)

в данном случае можно спокойно  
кэшировать эти данные

Terminal: System Design × + ▾



# КЭШИРОВАНИЕ ОШИБОК

Кэшируем ошибки и тогда последующие запросы  
не будут обращаться к источнику информации,  
а также не будет **cache miss attack**

# ПО-ХОРОШЕМУ, НУЖНО УМЕТЬ ДЕРЖАТЬ НАГРУЗКУ БЕЗ КЭША

Задача кэша – ускорить ответ, а не держать нагрузку

# ПРОБЛЕМА СТАРТА С НЕПРОГРЕТЬМ КЭШЕМ

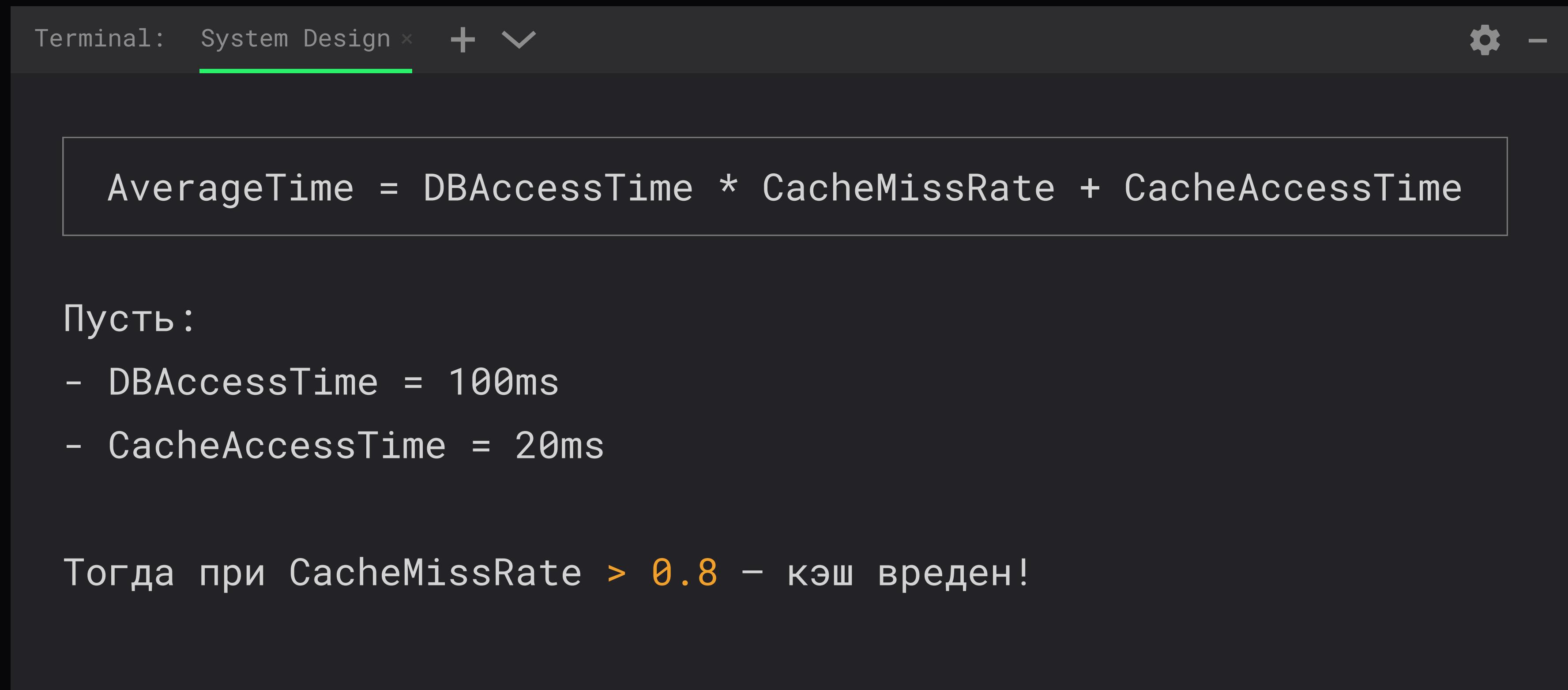
---

Предварительно загружаем данные  
в кэш часто используемых запросов

подтема №1

# ВСЕГДА ЛИ КЭШИРОВАНИЕ ПОЛЕЗНО?

# ЭФФЕКТИВНОСТЬ



A screenshot of a terminal window titled "System Design". The terminal has a dark background and a green status bar at the top. Inside the terminal, there is a highlighted box containing the following mathematical formula:

$$\text{AverageTime} = \text{DBAccessTime} * \text{CacheMissRate} + \text{CacheAccessTime}$$

Below the formula, the text "Пусть:" is displayed. Underneath that, there is a list of two items:

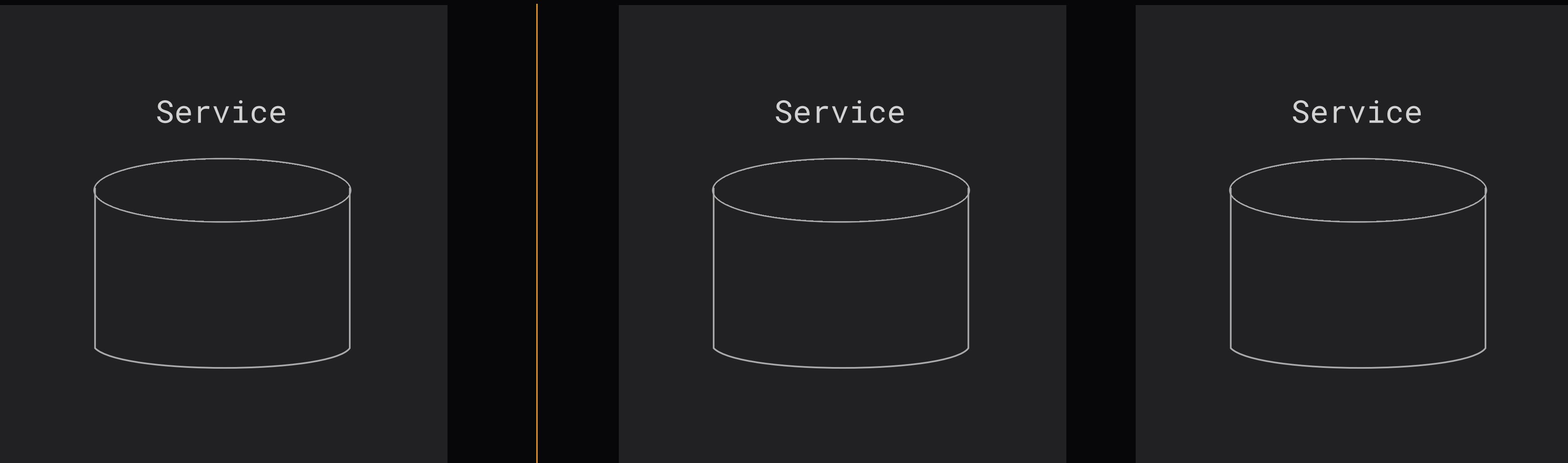
- DBAccessTime = 100ms
- CacheAccessTime = 20ms

At the bottom of the terminal window, the text "Тогда при CacheMissRate > 0.8 – кэш вреден!" is displayed in orange.

подтема №2

# ВИДЫ КЭШИРОВАНИЯ

# ВНУТРЕННЕЕ КЭШИРОВАНИЕ



# ВНУТРЕННЕЕ КЭШИРОВАНИЕ

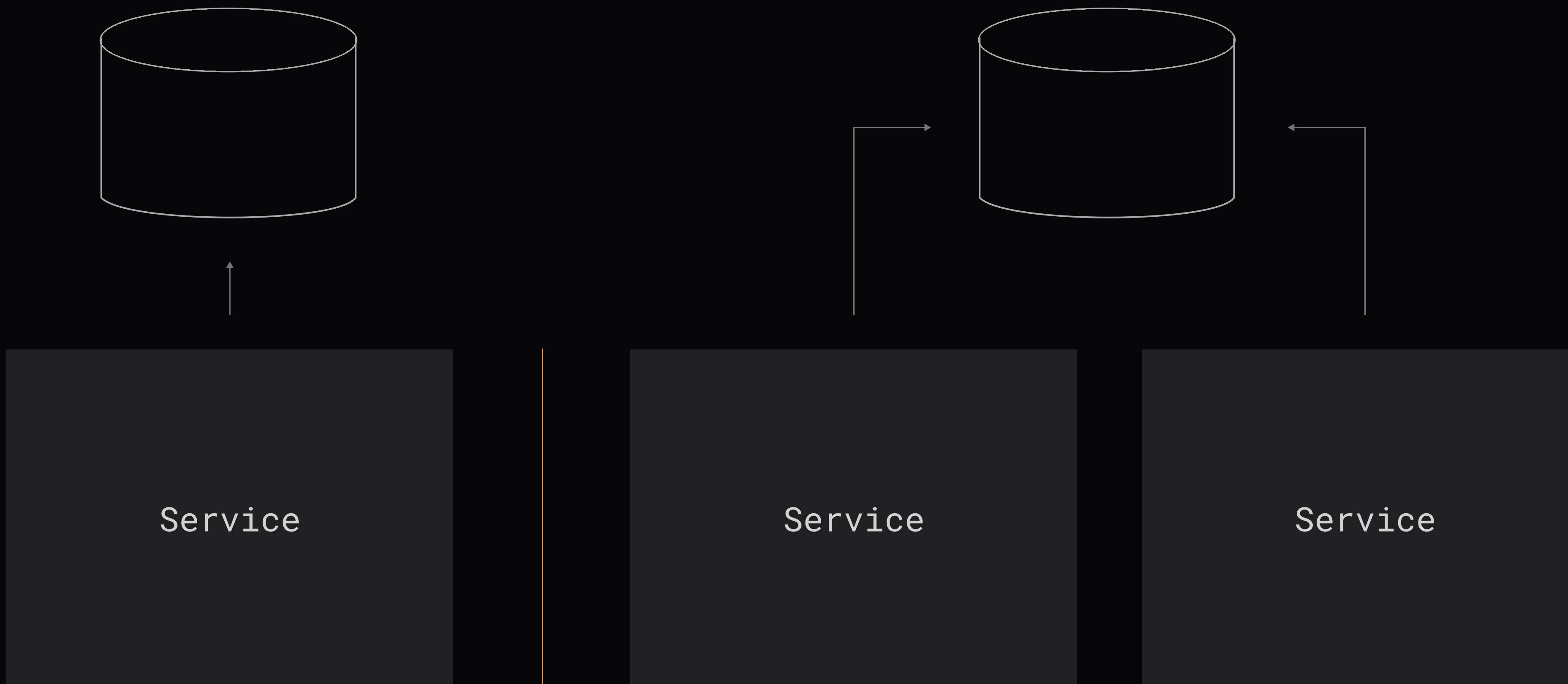
## Плюсы:

- Высокая скорость
- Отсутствие сетевых запросов
- Нет расходов на Marshaling/Unmarshalling данных

## Минусы:

- Горизонтальное масштабирование
- Прогрев кэша после падения сервиса

# ВНЕШНЕЕ КЭШИРОВАНИЕ



# ВНЕШНЕЕ КЭШИРОВАНИЕ

Плюсы:

- Хранение большого объема данных
- Простое горизонтальное масштабирование
- После падения сервиса данные кэша не теряются
- Простой прогрев кэша и простая логика инвалидации

Минусы:

- Скорость работы

подтема №3

# СПОСОБЫ ВЗАИМОДЕЙСТВИЯ С КЭШЕМ

Terminal: System Design × + ▾

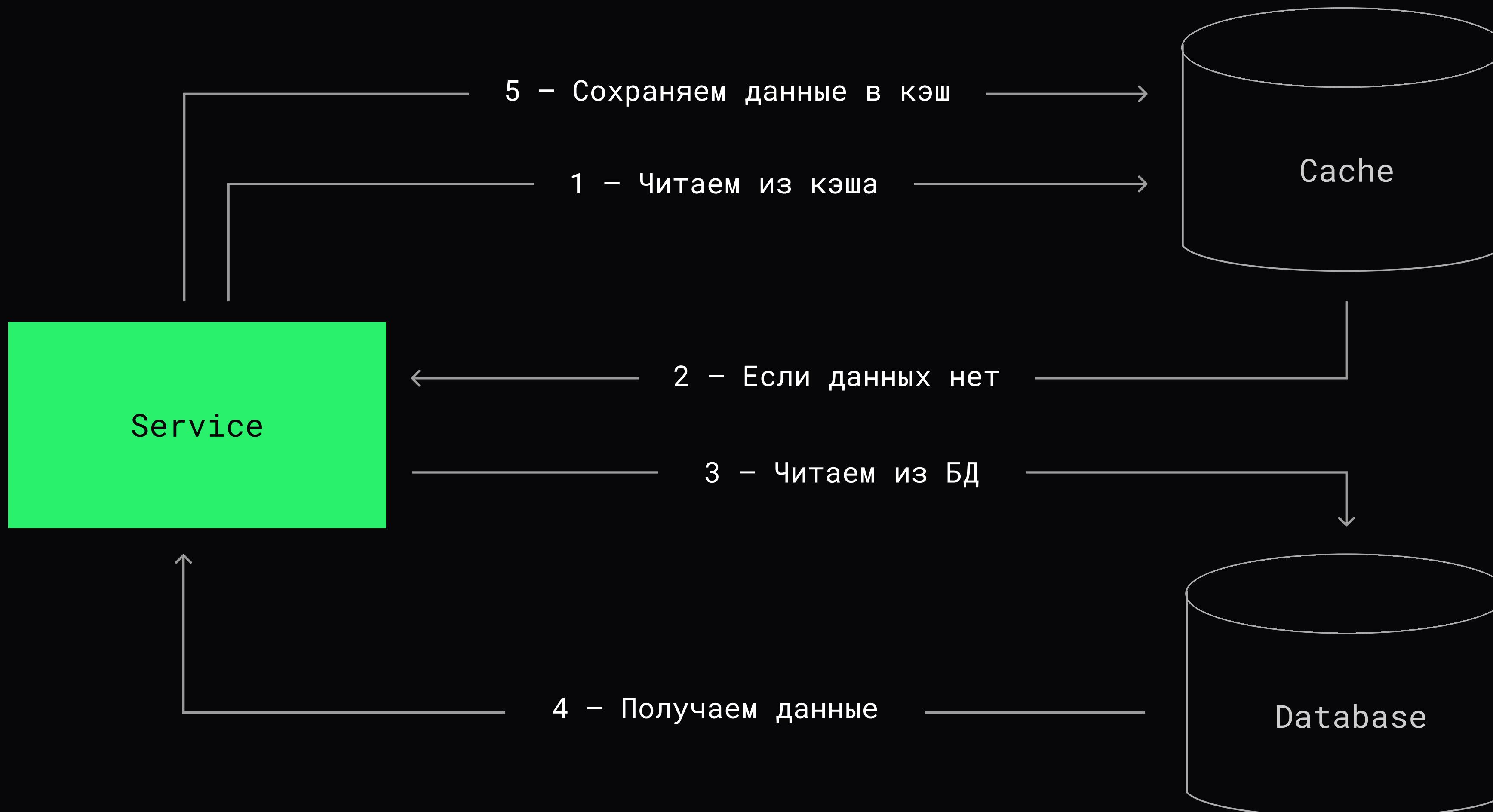


# CACHE ASIDE (КЭШИРОВАНИЕ НА СТОРОНЕ)

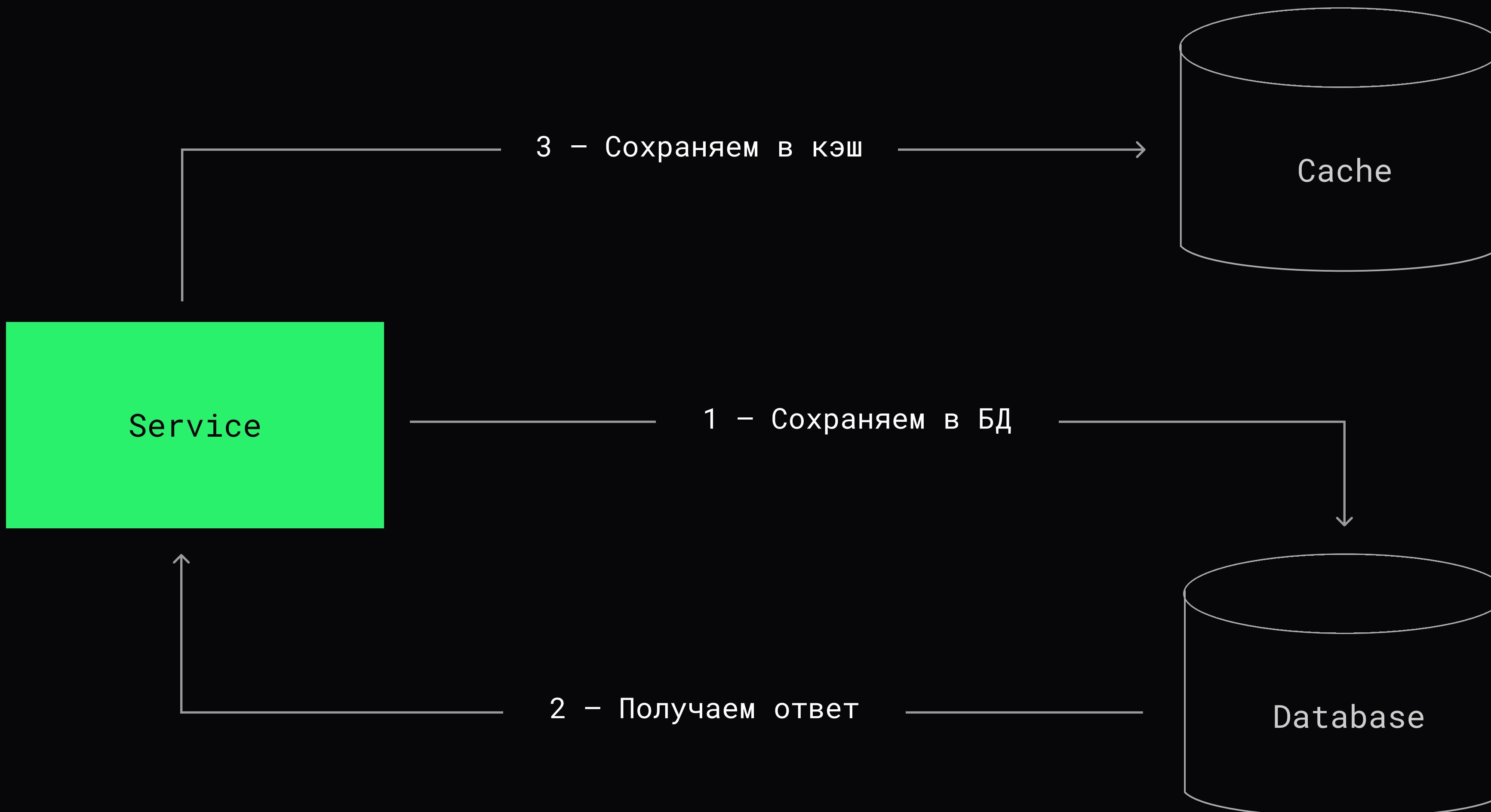
В этой стратегии, приложение координирует запросы в кэш и БД и само решает, куда и в какой момент нужно обращаться



# READ ASIDE



# WRITE ASIDE



Terminal: System Design × + ▾



# CACHE THROUGH (СКВОЗНОЕ КЭШИРОВАНИЕ)

В рамках этой стратегии все запросы от приложения проходят через кэш

# READ THROUGH



# WRITE THROUGH



Terminal: System Design × + ▾



# CACHE AHEAD (ОПЕРЕЖАЮЩЕЕ КЭШИРОВАНИЕ)

Запросы на чтение всегда идут только  
в кэш, никогда не попадая в БД напрямую

**Service**

1.1 – Читаем из кэша  
1.2 – Отдаем данные

Cache

2.1 – Периодически  
загружаем данные из БД

Database

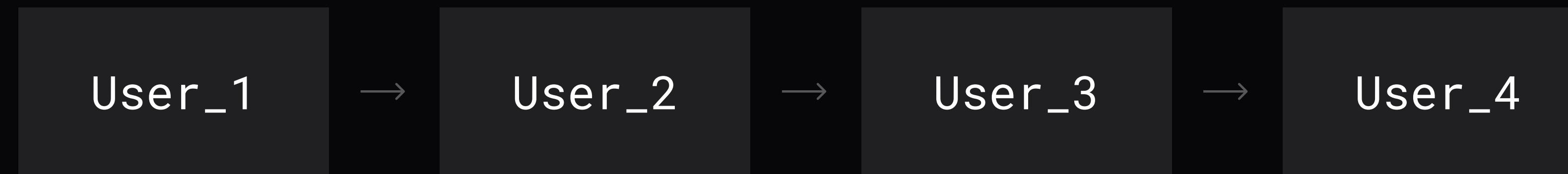
# РЕЗЮМЕ

- Важно учитывать актуализацию данных
- Стоит балансировать между характером нагрузки  
(какая нагрузка преобладает – чтение или запись)
- Также не забыть про отказоустойчивость

подтема №4

# АЛГОРИТМЫ ВЫТЕСНЕНИЯ ДАННЫХ

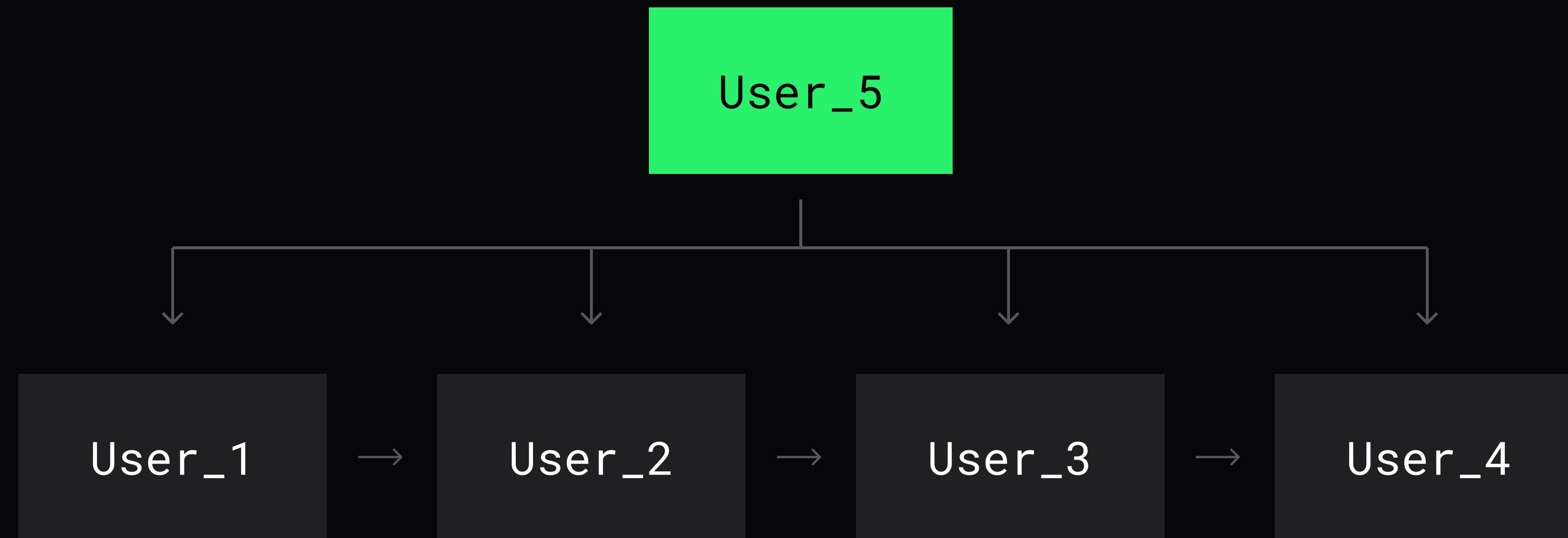
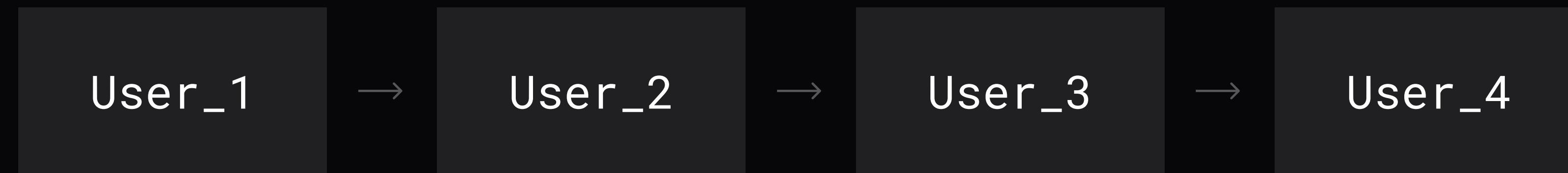
# КОГО ВЫТЕСНЯТЬ?



User\_5



# RANDOM



# FIFO

add – user\_1

add – user\_2

add – user\_3

add – user\_4

User\_1

User\_2

User\_3

User\_4

add – user\_5

User\_5

User\_2

User\_3

User\_4

# LIFO

add – user\_1

add – user\_2

add – user\_3

add – user\_4

User\_1

User\_2

User\_3

User\_4

add – user\_5

User\_1

User\_2

User\_3

User\_5

# LRU

add - user\_1

add - user\_2

add - user\_3

add - user\_4

User\_1

User\_2

User\_3

User\_4

get - user\_1

add - user\_5

User\_1

User\_5

User\_3

User\_4

# MRU

add - user\_1

add - user\_2

add - user\_3

add - user\_4

User\_1

User\_2

User\_3

User\_4

get - user\_3

add - user\_5

User\_1

User\_2

User\_5

User\_4

# LFU

add - user\_1

add - user\_2

add - user\_3

add - user\_4

User\_1

User\_2

User\_3

User\_4

get - user\_3

get - user\_1

get - user\_4

add - user\_5

User\_1

User\_5

User\_3

User\_4

# РЕЗЮМЕ

LRU

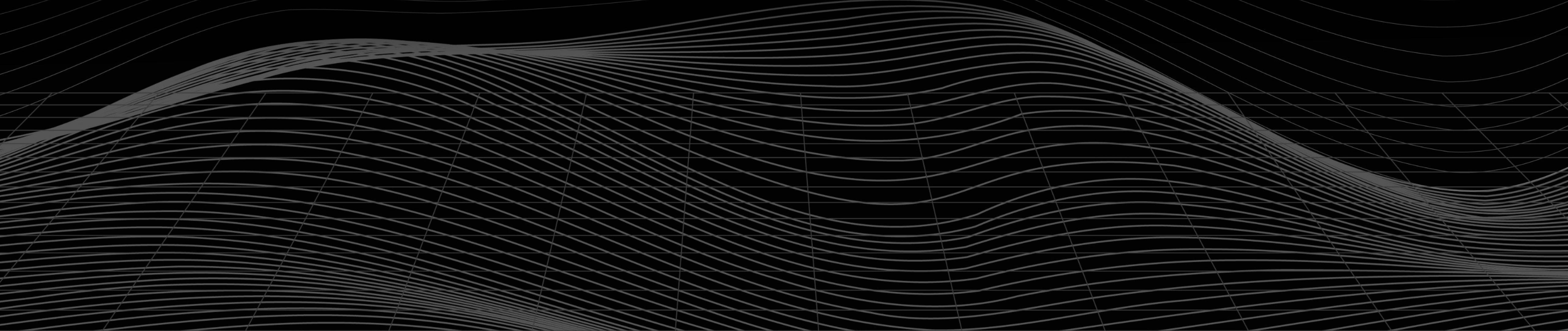
неиспользованный дольше всех вылетает из кэша

MRU

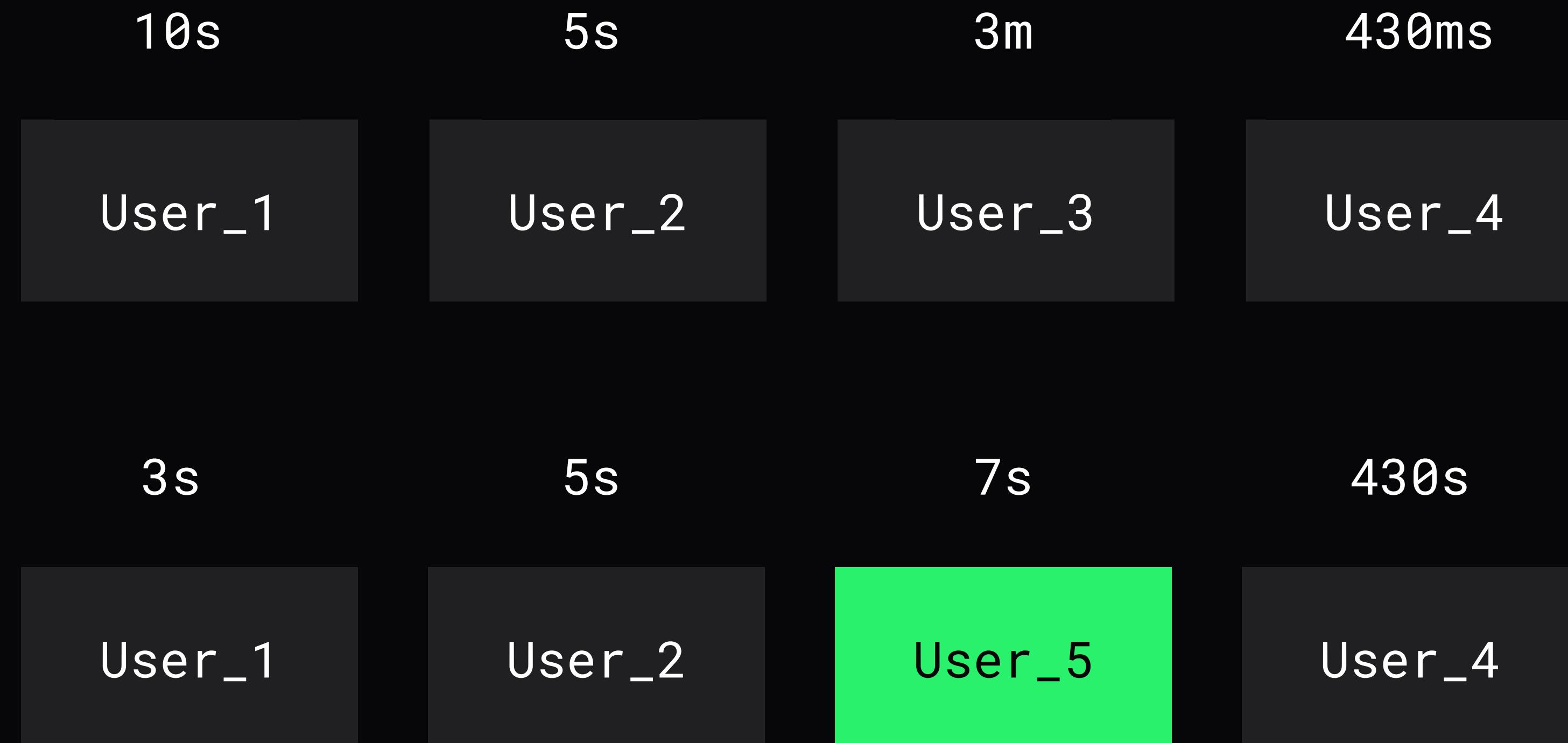
последний использованный вылетает из кэша  
(специфичный кейс, бережем старье)

LFU

реже всего использованный вылетает из кэша



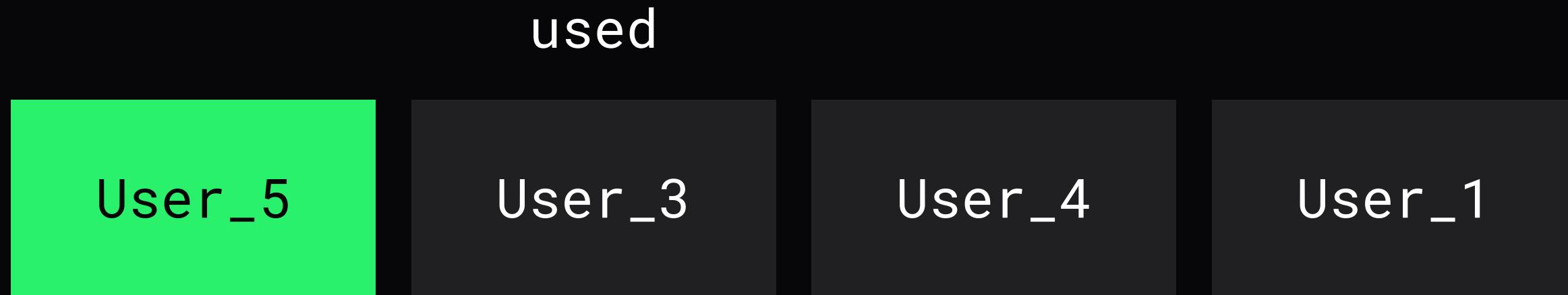
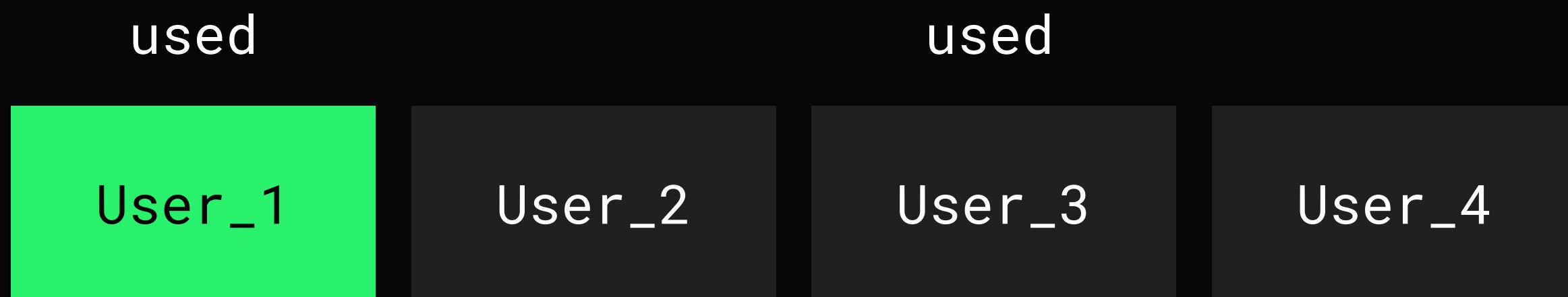
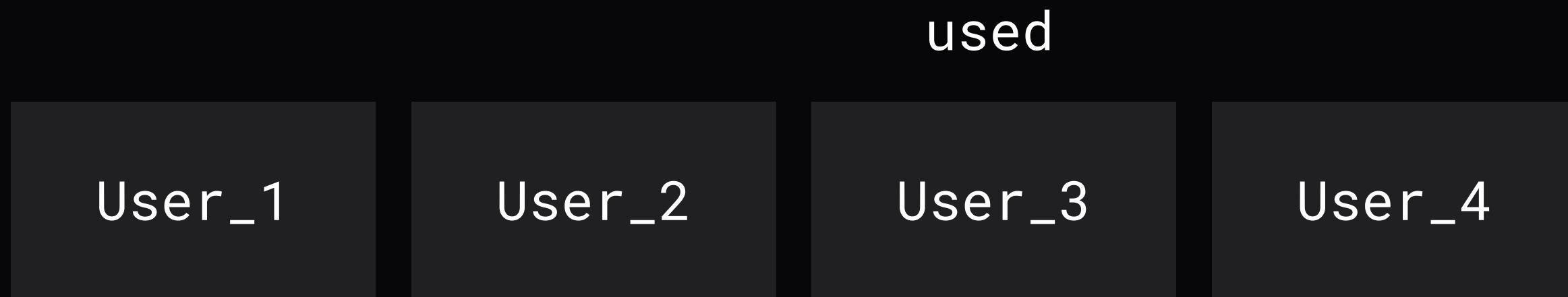
# АЛГОРИТМ БЕЛАДИ (OPT)



# SECOND CHANCE

При обращении выставляем бит присутствия, а при вытеснении удаляем из начала очереди, если бит = 0

Если бит = 1, то меняем бит на ноль, затем переносим элемент в конец очереди и идем к следующему



# CLOCK

Тот же Second Chance, только не нужно двигать элемент из начала в конец очереди, если бит равен единице



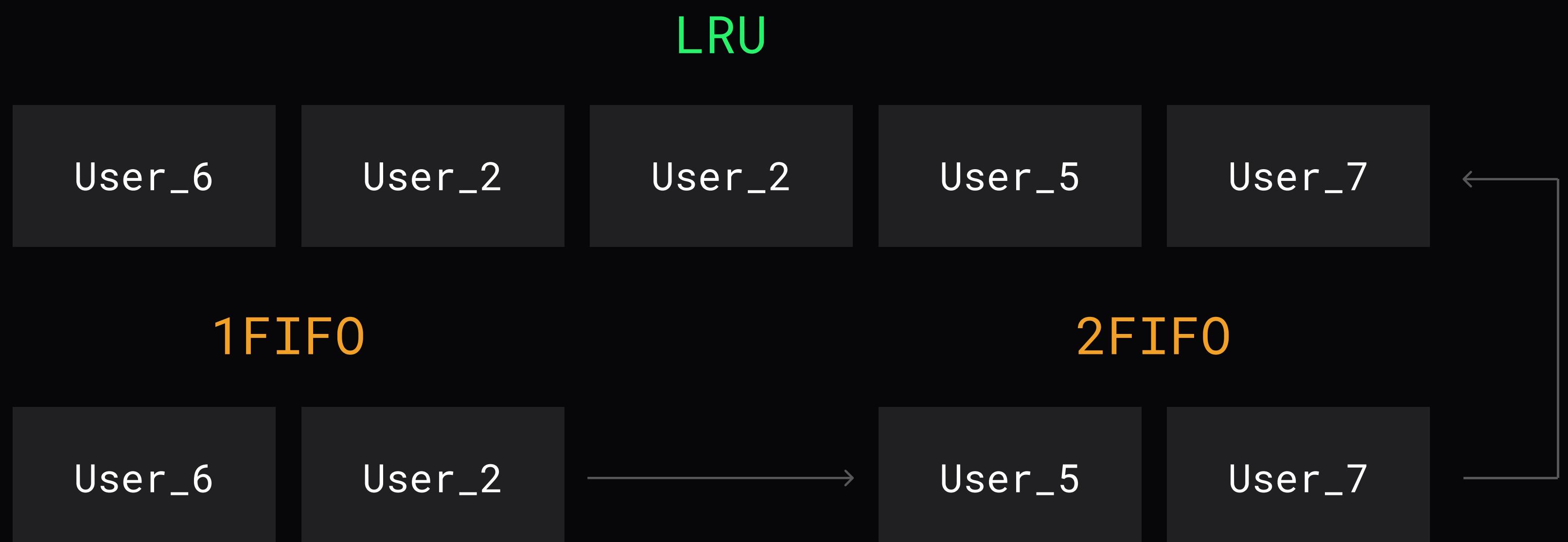
# 2Q

Элементы, запрошенные из 1FIFO, никуда не двигаются.

Вытесненные из 1FIFO перемещаются в 2FIFO

Элементы, запрошенные из 2FIFO, уходят в LRU.

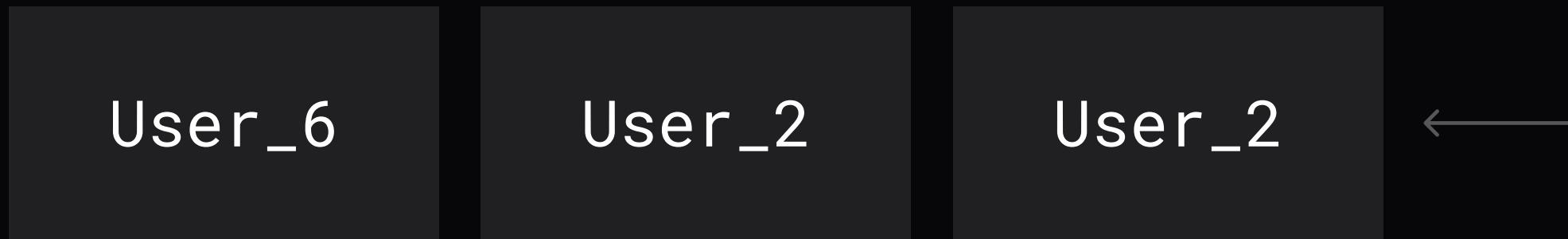
Вытесненные из 2FIFO удаляются



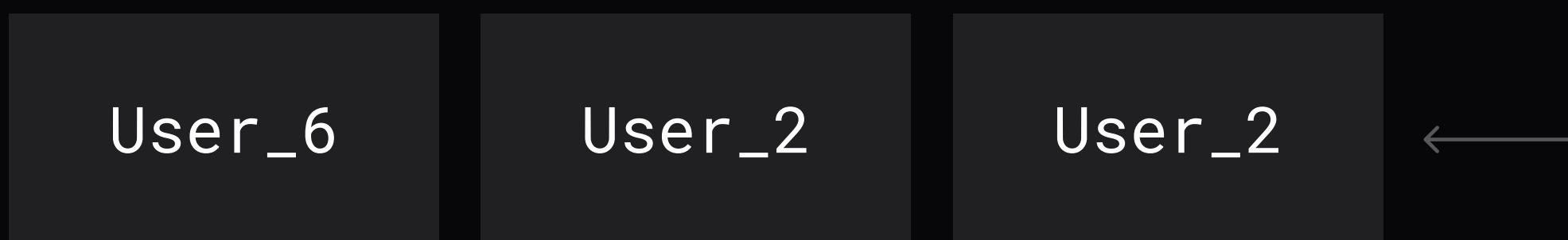
# SLRU

- Сперва кладем в первую коробочку.
- При повторном запросе перекладываем во вторую.
- При еще одном повторном – из второй в третью

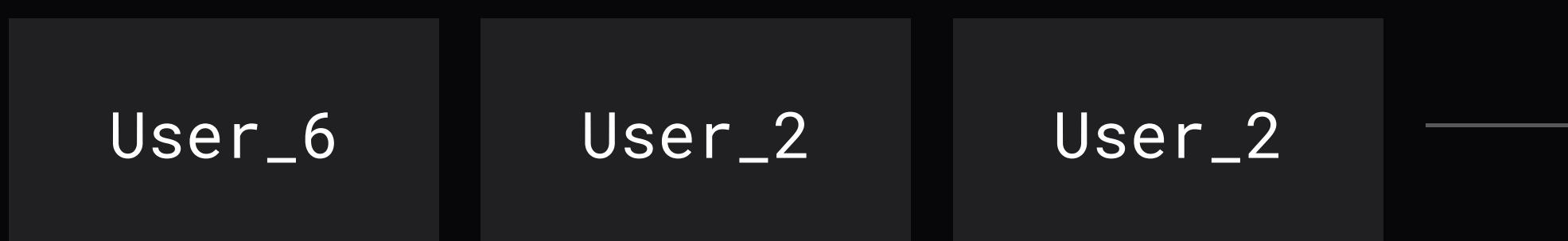
Hot LRU



Warm LRU



Cold LRU



Terminal: System Design × + ▾



# TLRU (TIME AWARE LRU)

Абсолютно такой же алгоритм, только к данным добавляется их время жизни в кэше (TTL), по которому они автоматически удаляются из кэша

Terminal: System Design × + ✓



# LRU-K

Удаляет страницу, K-й последний доступ к которой находится дальше всего в прошлом.

Например, LRU-1 – это просто LRU, тогда как LRU-2 удаляет страницы в соответствии со временем их предпоследнего доступа

подтема №5

# ИНВАЛИДАЦИЯ ДАННЫХ В КЭШЕ

Terminal: System Design × + ▾



# ИНВАЛИДАЦИЯ ПО TTL (TIME TO LIVE)

При сохранении данных в кэш для них  
устанавливается время жизни (TTL) и данные  
будут автоматически удалены через это время  
(основная проблема заключается в поборе TTL)

Terminal: System Design × + ▾



# JITTER

Если записи становятся недействительными одновременно и в большое количество, то источник данных может пострадать. В данной ситуации помогает Jitter (случайная величина, добавляемая к TTL)

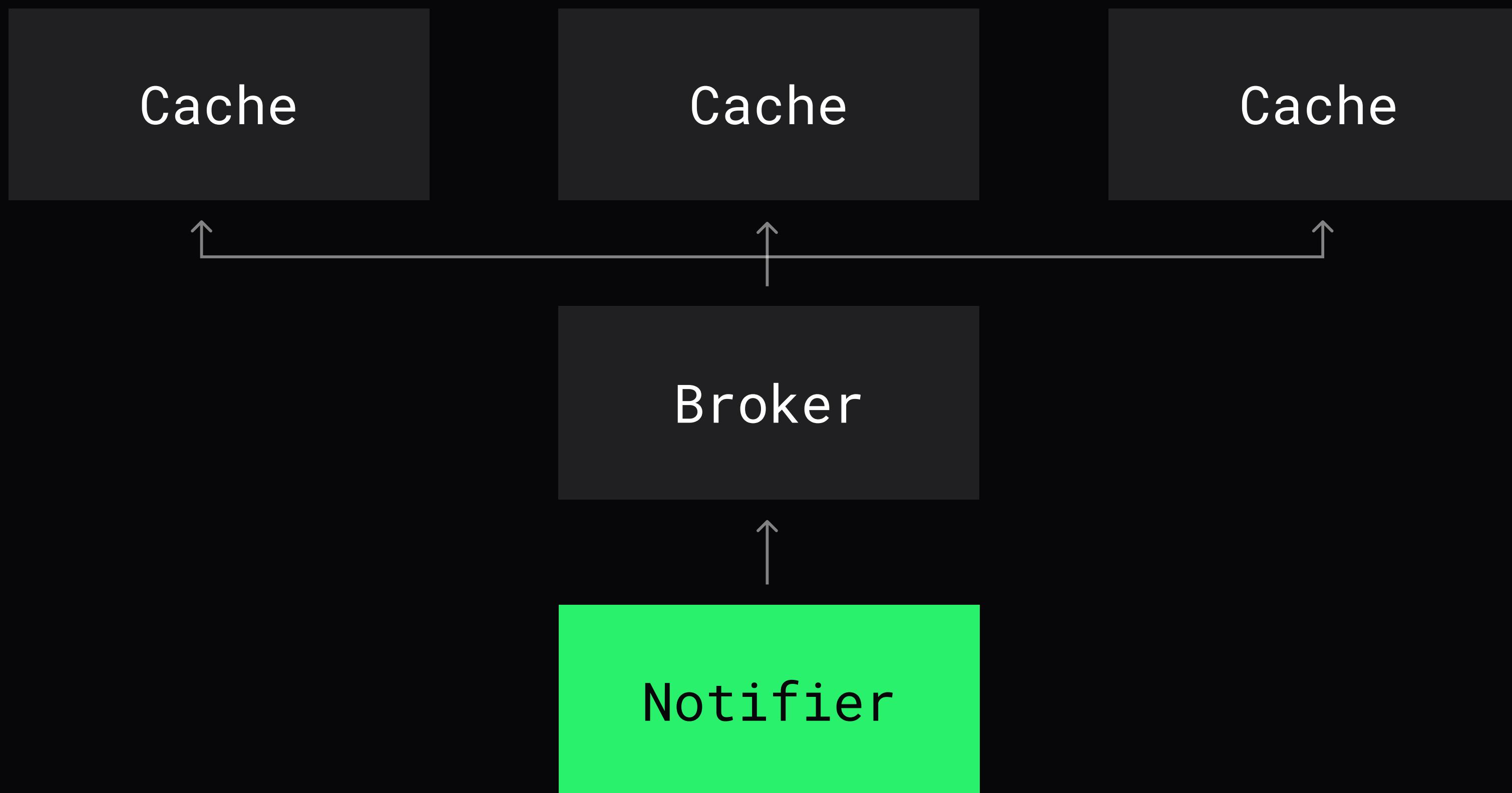
Terminal: System Design × + ▾



# THUNDERING HERD PROBLEM

Резкий рост нагрузки на источник данных, который возникает, когда множество процессов/потоков одновременно запрашивают один ключ кэша, получают cache miss, а затем каждый из них выполняет один и тот же запрос

# ИНВАЛИДАЦИЯ ПО СОБЫТИЮ



Yandex × + ⚙ -

Я

найдётся всё

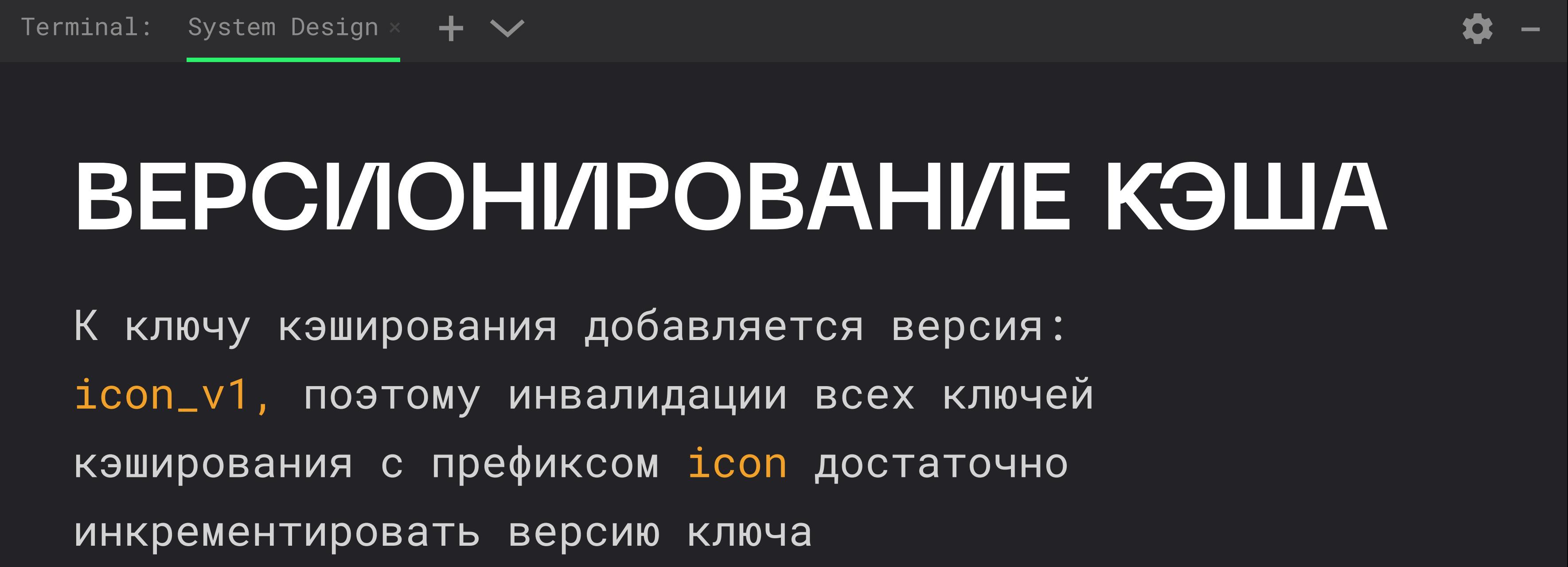
0

0

найдётся всё

11° 4 USD 79,33 EUR 86,85 Москва

0



Yandex × + ⚙ -

@ mail

Последние письма

Владимир Балун  
vladimirbalun@mail.ru

Входящие 8

Написать письмо

Создать почту

ВКонтакте

Вы могли пропустить

Сообщения 1

Облако

Место для ваших файлов

Надежная защита

Облако – это персональное хранилище, где ваши фото и файлы находятся в

Поиск Яндекса Найти

реклама

билайн Меняй условия Тарифа UP

Скидки для семьи

Узнать больше

Новости Спецоперация Москва Спорт Леди Авто Кино Hi-Tech Игры Дети Здоровье ...

Лавров: Россия больше не позволит Западу взрывать газопроводы

Россия впредь будет ориентирована на надежных партнеров, в том числе Китай

Песков ответил на предложение о военном положении в регионах РФ

В Москве началась церемония прощания с Глебом Павловским

Глава села рассказал, как диверсанты атаковали Брянскую область

Эксперты спрогнозировали доллар выше 200 рублей в 2025 году

Москва →

+1° Снег Влажность 72%

Вечером -6° Ночью -12° Утром -7°

# ТЕГИРОВАНИЕ КЭША

ID	Data	Tags
12312	News #1	<span>weather</span>
6423	News #2	
1312	News #3	<span>currency</span> <span>weather</span>

ID	Data
moscow	<span>weather</span> data

подтема №6

# МНОГОМЕРНЫЙ КЭШ

# КЭШИ МОГУТ СКЛАДЫВАТЬСЯ В ЦЕПОЧКУ ИЛИ В ДЕРЕВО

когда, например, происходит обращение к какому-нибудь сервису:

- 
- /1 Кэш браузера

---

  - /2 Кэш прокси сервера

---

  - /3 Внутренний кэш приложения

---

  - /4 Внешний кэш приложения

---

  - /5 Кэш движка базы данных

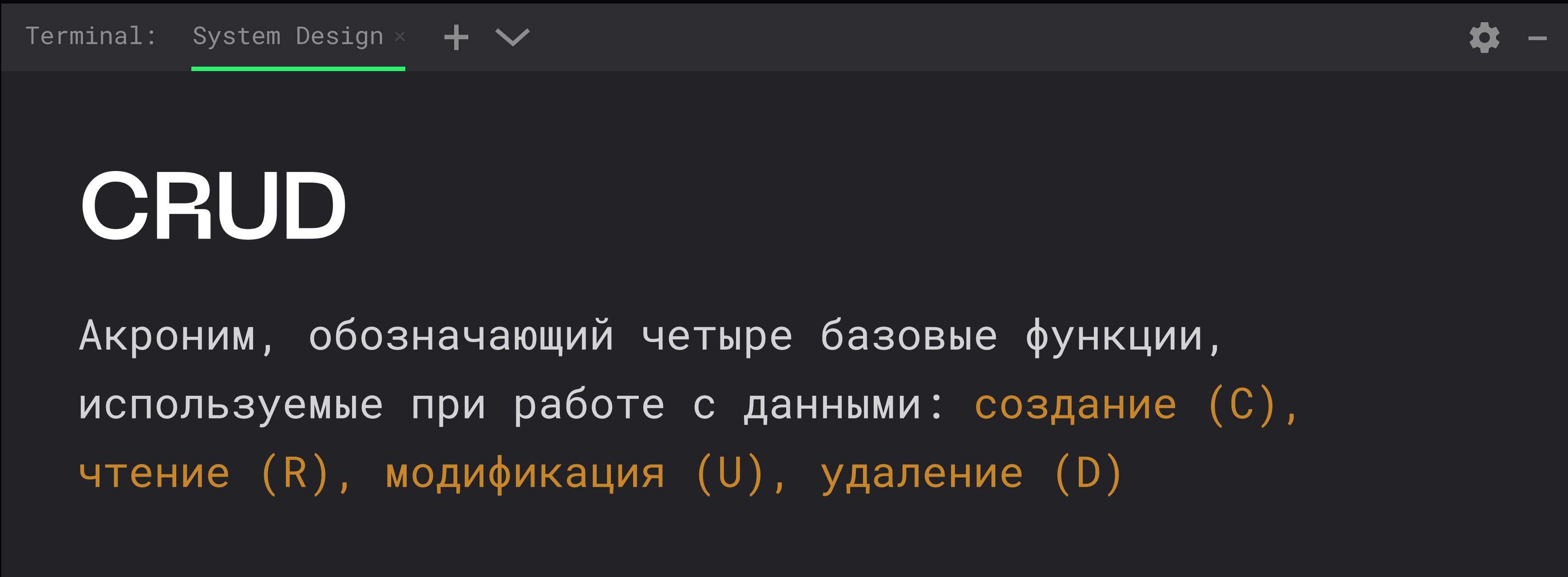
# FAQ

Кэширование

Внутреннее и внешнее, способы взаимодействия,  
алгоритмы вытеснения

**ПЕРЕРЫВ 5 МИНУТ**

# API



# REST

- Глагол – метод запроса (GET, POST, DELETE, PUT)
- Существительное – URI запроса
- Дополнительная информация – хедеры запроса
- Содержимое – тело запроса и ответа в Json
- Результат – код ответа

# REST



```
1 Request:  
2 GET ${address}/employees/14  
3  
4 Response:  
5 200 - Ok  
6 {  
7   "name": "Petr",  
8   "surname": "Ivanov",  
9   "city": "Moscow"  
10 }  
11
```

# SOAP

```
1 <?xml version="1.0"?>
2 <soap:Envelope
3   xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
4   soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
5 <soap:Body>
6   <m:GetPrice xmlns:m="https://online-shop.ru/prices">
7     <m:Item>Dell Vostro 3515-5371</m:Item>
8   </m:GetPrice>
9 </soap:Body>
10 </soap:Envelope>
```

```
1 <?xml version="1.0"?>
2 <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
3   soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
4 <soap:Body>
5   <m:GetPriceResponse xmlns:m="https://online-shop.ru/prices">
6     <m:Price>37299</m:Price>
7   </m:GetPriceResponse>
8 </soap:Body>
9 </soap:Envelope>
```

Terminal: System Design × + ✓



# RPC

Класс технологий, позволяющих программам  
вызывать функции или процедуры в другом  
адресном пространстве

# gRPC



```
1 person := pb.Person{
2   Id:    1234,
3   Name:  "John Doe",
4   Email: "jdoe@example.com",
5   Phones: []*pb.Person_PhoneNumber{
6     {Number: "555-4321", Type: pb.Person_HOME},
7   },
8 }
9
10 _, err := client.SavePerson(context.Background(), person)
11 if err != nil {
12   ...
13 }
```



```
1 message Person {
2   string name = 1;
3   int32 id = 2;
4   string email = 3;
5
6   enum PhoneType {
7     MOBILE = 0;
8     HOME = 1;
9     WORK = 2;
10 }
11
12 message PhoneNumber {
13   string number = 1;
14   PhoneType type = 2;
15 }
16
17 repeated PhoneNumber phones = 4;
18 }
19
20
21 service PersonsList {
22   rpc SavePerson(Person) returns (google.protobuf.Empty){}
23 }
```

# UNDER / OVER FETCHING

# GraphQL

```
1 query {  
2   users {  
3     fname  
4     age  
5   }  
6 }
```

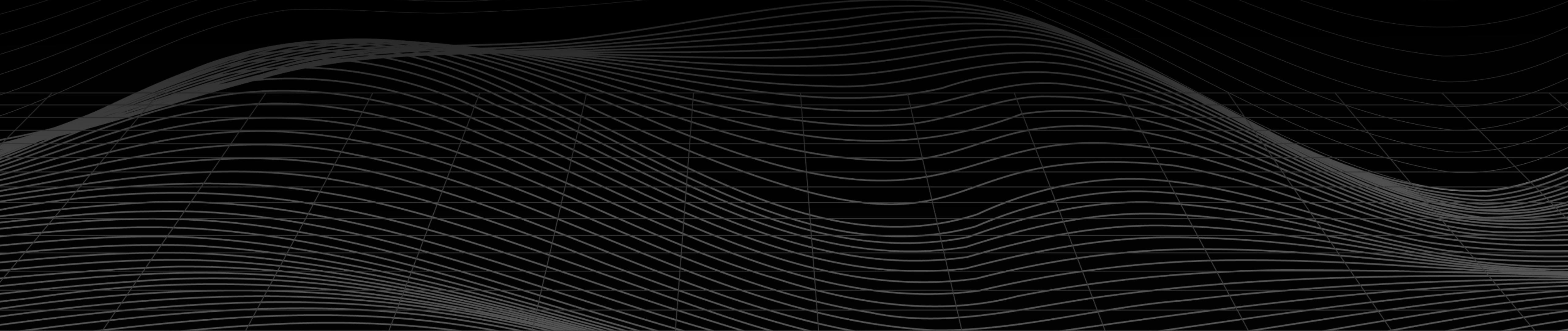
```
• • •  
1 data : {  
2   users [  
3     {  
4       "fname": "Joe",  
5       "age": 23  
6     },  
7     {  
8       "fname": "Betty",  
9       "age": 29  
10    }  
11  ]  
12 }
```

# РЕЗЮМЕ

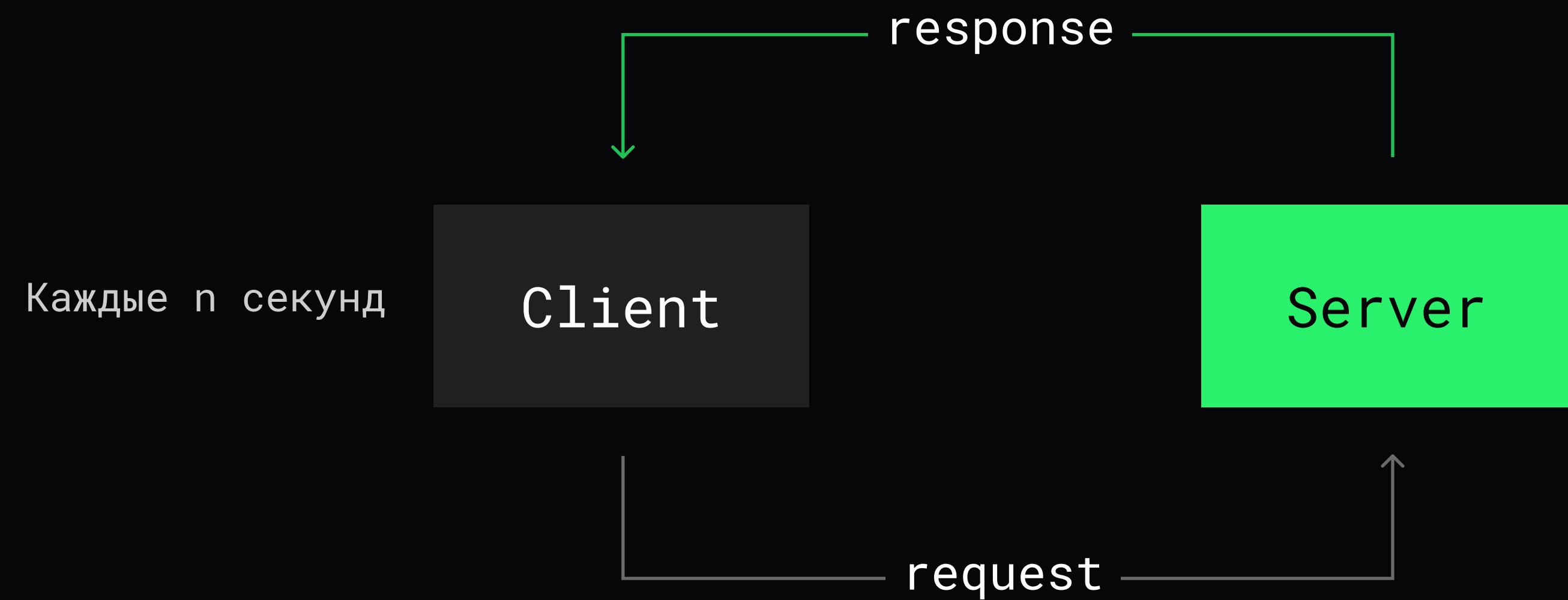
---

- /1 SOAP уже умер
- /2 REST для клиентского взаимодействия
- /3 gRPC для внутреннего взаимодействия
- /4 GraphQL для кастомизации запросов

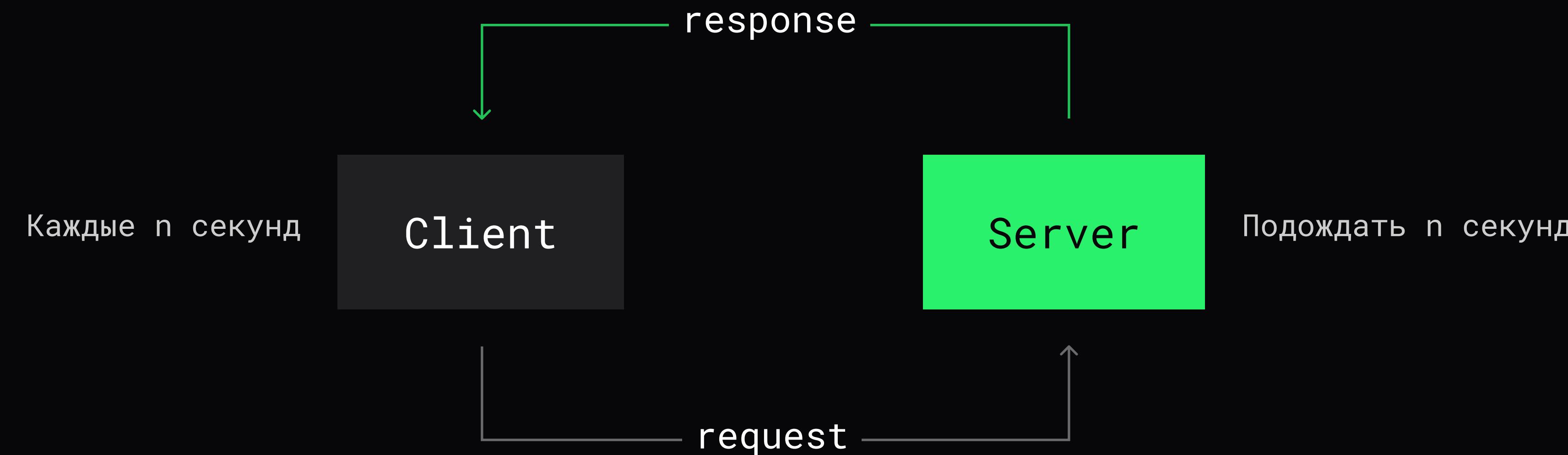
# КАК ПЕРИОДИЧЕСКИ ПОЛУЧАТЬ ОБНОВЛЕНИЯ ОТ СЕРВЕРА?



# POLLING

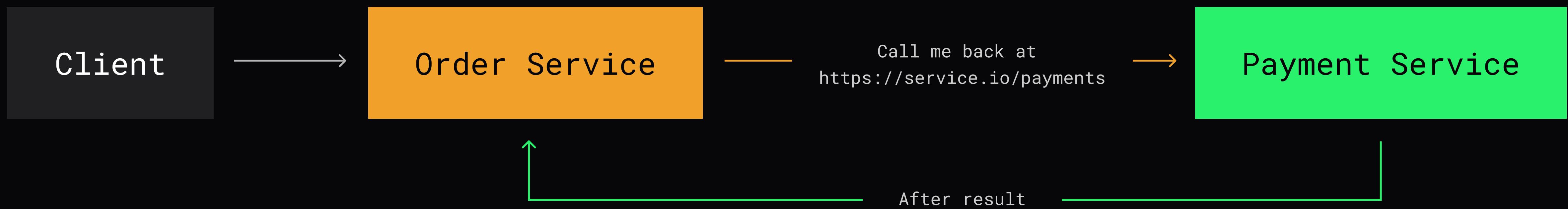


# LONG POLLING



# WEBHOOK

Метод, позволяющий приложению автоматически передавать данные или уведомления другому приложению по определенным событиям или действиям



# STREAMING



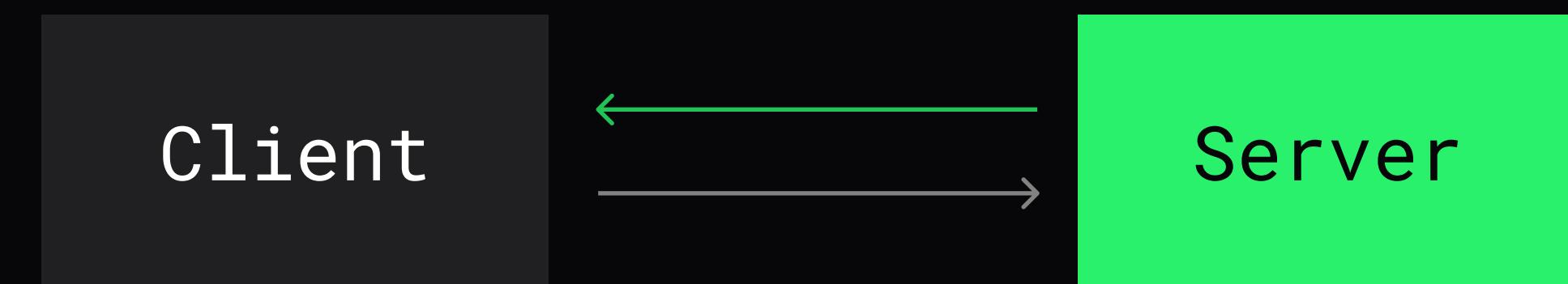
# SSE (SERVER SIDE EVENTS)

Клиент подписывается на события сервера и как только происходит событие – клиент сразу же получает уведомление по HTTP и некоторые данные, связанные с этим событием



# WEBSOCKET

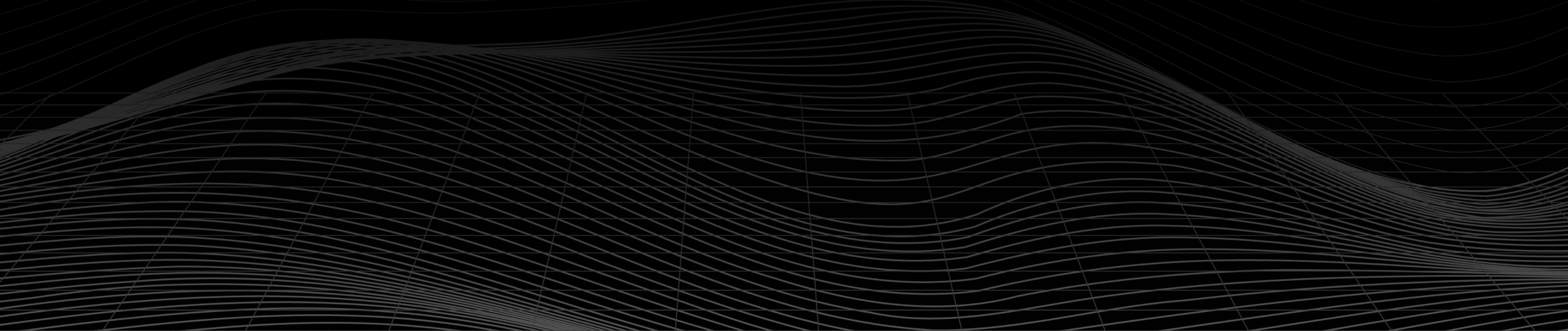
Протокол для общения между клиентом и сервером, предоставляющий возможность двухсторонней коммуникации поверх TCP



# РЕЗЮМЕ

- 
- |                  |  |
|------------------|--|
| <b>Pooling</b>   | используется, когда клиенту необходимо регулярно обновлять информацию                  |
| <hr/>            |  |
| <b>Streaming</b> | используется, когда клиенту необходимо получать непрерывный или хаотичный поток данных |
| <hr/>            |  |
| <b>Webhooks</b>  | используются, когда серверу необходимо уведомить клиента о каком-то событии            |

# КАК ОТДАВАТЬ БОЛЬШОЕ КОЛИЧЕСТВО СУЩНОСТЕЙ ЧЕРЕЗ API?



Ответ:

# ПАГИНАЦИЯ

Большое количество сущностей отправляется  
клиенту по частям (чанками – chunks)



```
GET /api/products?offset=200&limit=100
```

```
{"items" : [...100 products]}
```



```
GET /api/products?page=10
```

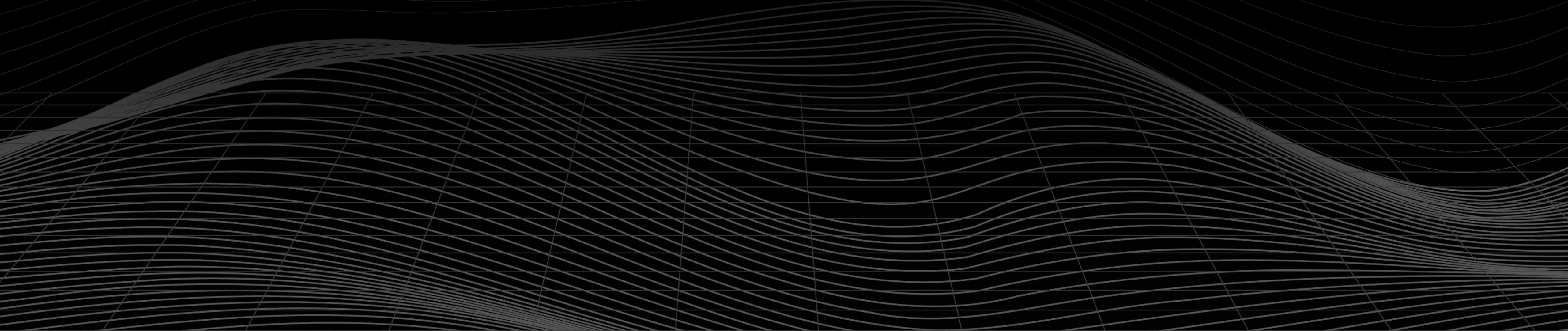
```
{"items" : [...100 products]}
```



```
GET /api/products?cursor=qWe
```

```
{"items" : [...100 products],"cursor" : "qWr"}
```

# КАК УЛУЧШИТЬ ПРОИЗВОДИТЕЛЬНОСТЬ API?



# ЗАПРОСЫ И ОТВЕТЫ МОЖНО СЖИМАТЬ

Чем меньше размер данных, тем быстрее  
они передаются по сети

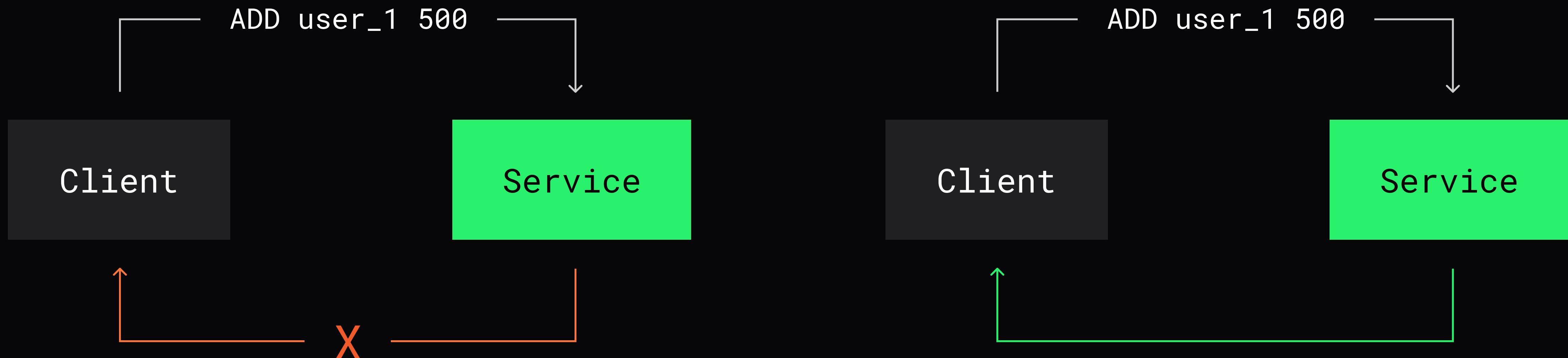
- ➊ Сжатие ускоряет загрузку и скачивание данных

Вариант №2

# МОЖНО СОХРАНЯТЬ ЗАПРАШИВАЕМЫЕ ДАННЫЕ В КЕШЭ У КЛИЕНТА

И возвращать их клиентам без повторного  
обращения к сервису

# КАК ПРЕДОТВРАТИТЬ ПОВТОРНОЕ ВЫПОЛНЕНИЕ ОПЕРАЦИИ?

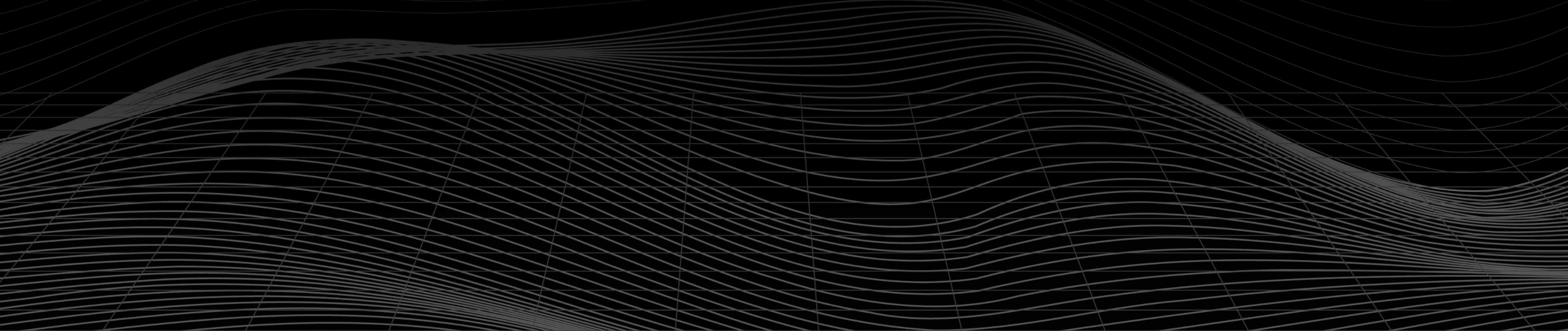


# КЛЮЧ ИДЕМПОТЕНТНОСТИ

На стороне клиента генерируем ключ и отправляем его сервису через API. Сервис у себя проверит, выполнял ли он уже эту операцию по ключу или нет

```
● ● ●  
{  
  ...,  
  "idempotence_key" : "7356bc5e-21c1-4fda-a790-2811b353bbc9"  
}
```

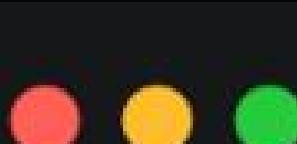
# КАК ВНОСИТЬ ИЗМЕНЕНИЯ В API?



# ВЕРСИОНИРОВАНИЕ API

Так как API со временем развивается, важно иметь стратегию управления версиями для управления будущими изменениями

- 1 Этот этап включает в себя реализацию механизмов управления версиями, включая номер версии



```
1 GET ${address}/api/v1/users/56
2 GET ${address}/api/v2/users/56
```



LET'S PRACTISE

# FAQ

API

REST, SOAP, gRPC, GraphQL,  
WebSocket, SSE, Webhooks

# OBSERVABILITY

подтема №1

# МОНИТОРИНГ

Terminal: System Design × + ▾



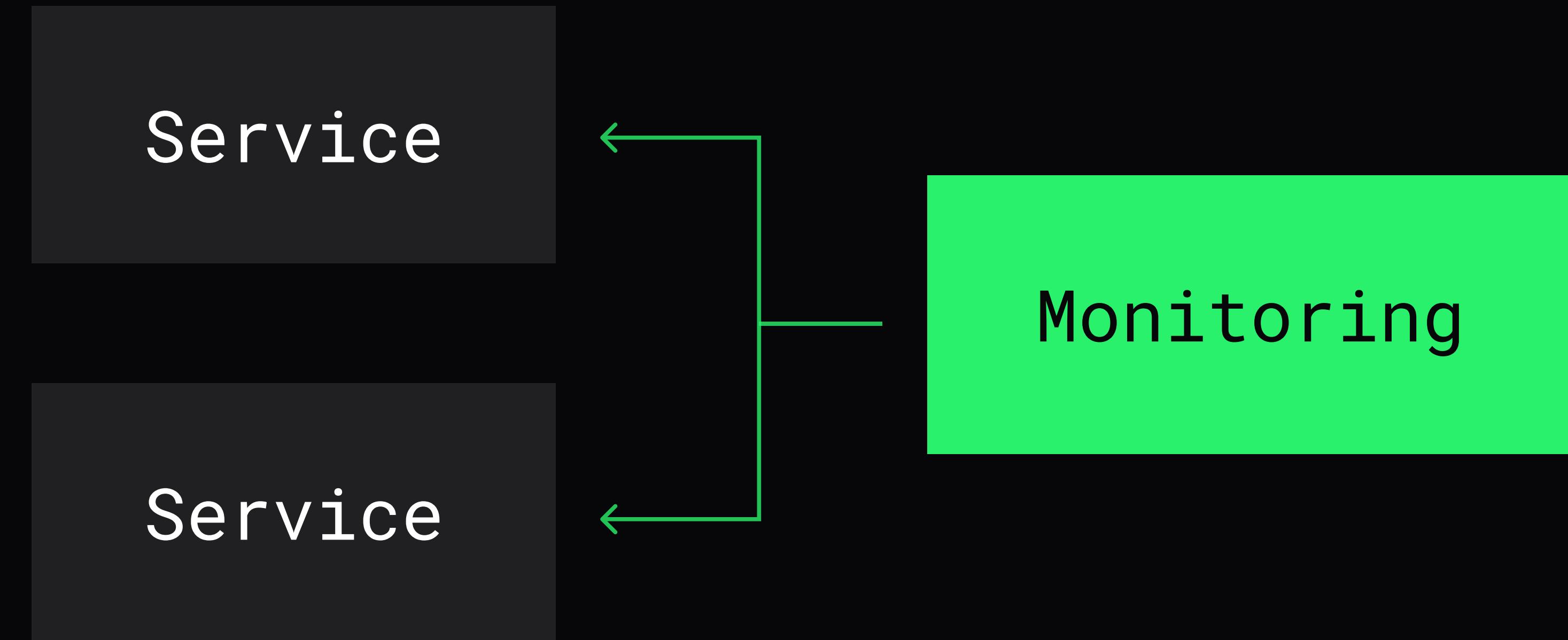
# МОНИТОРИНГ

Процесс сбора, анализа и визуализации данных о работе приложения с целью обеспечения его надежности, производительности и доступности



# GRAFANA





Terminal: System Design × + ▾



# LTES (GOOGLE SRE)

- **Latency** – время на обработку одного запроса
- **Traffic** – количество запросов
- **Errors** – количество ошибок
- **Saturation** – насколько компонент использует свои ресурсы

Terminal: System Design × + ▾



# RED

- Rate – количество запросов
- Errors – количество ошибок
- Duration – время обработки одного запроса

# USE

- **Utilization** – время или процент использования ресурса
- **Saturation** – количество отложенной работы
- **Errors** – количество ошибок

# НЕЛЬЗЯ СКАЗАТЬ, ЧТО КАКАЯ-ТО ИЗ ЭТИХ МЕТОДОЛОГИЙ «ПРАВИЛЬНАЯ»

Каждый отдельный компонент системы нужно мониторить, основываясь на здравом смысле

Terminal: System Design × + ▾



# АЛЕРТИНГ

Алерting отслеживает изменения определенных метрик с помощью алертов и отправляет уведомления через подходящий для вас канал уведомлений (основывается на концепциях SLI, SLA и SL0)

АЛЕРТ ДОЛЖЕН СРАБАТЬВАТЬ,  
КОГДА ВОЗНИКЛА ОШИБКА, НО ЕЩЕ  
МОЖНО ЧТО-ТО ИСПРАВИТЬ

а не когда «всё уже очень  
плохо» и не от каждой «помехи»

# СИСТЕМЫ МОНИТОРИНГА



Prometheus



ZABBIX



# FAQ

Мониторинг

# **ЛОГИРОВАНИЕ**

# ЛОГИРОВАНИЕ

Логированием называют запись логов - оно позволяет ответить на вопросы, что происходило, когда и при каких обстоятельствах



Без логов сложно понять, из-за чего появляется ошибка, если она возникает периодически и только при определенных условиях

#1

logs

Server

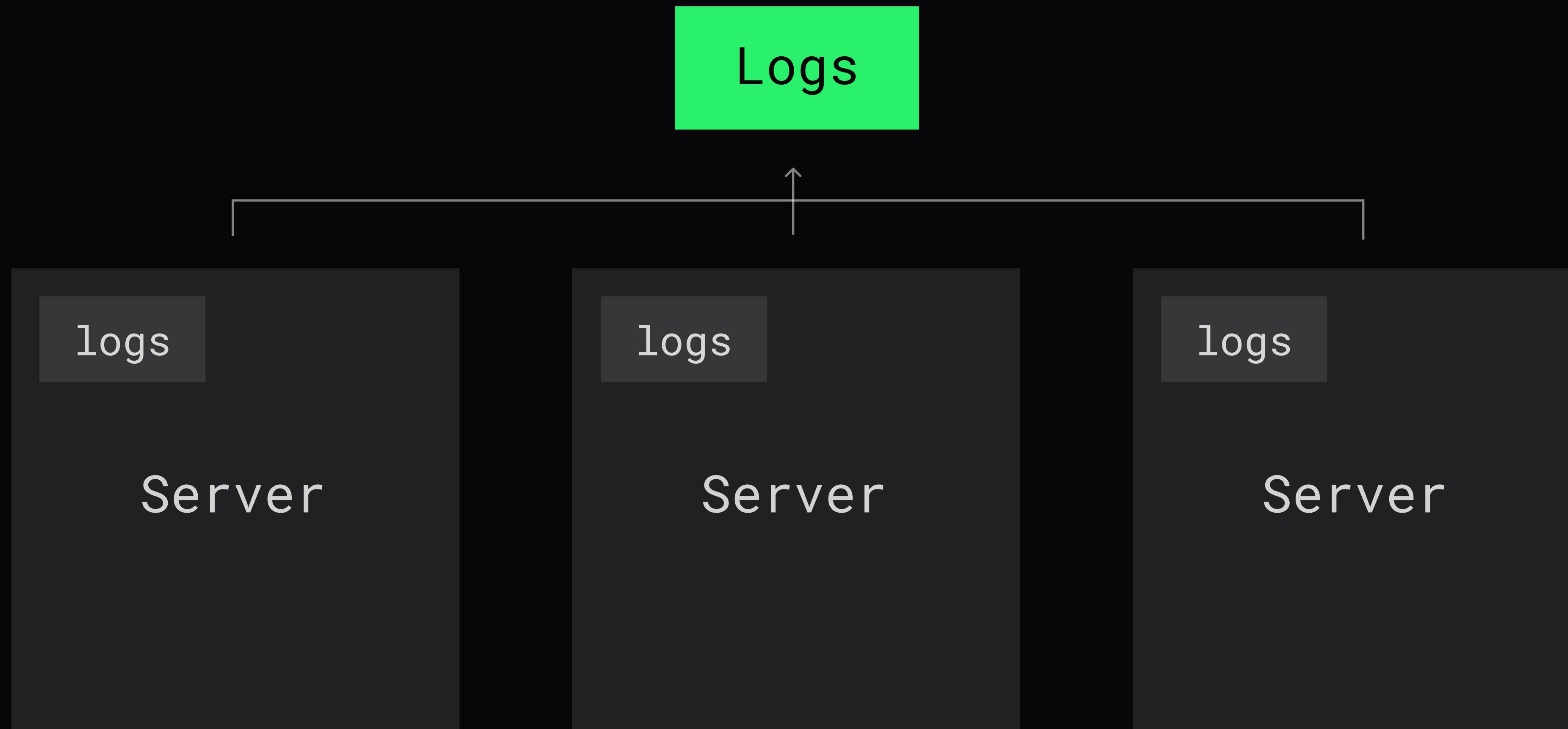
logs

Server

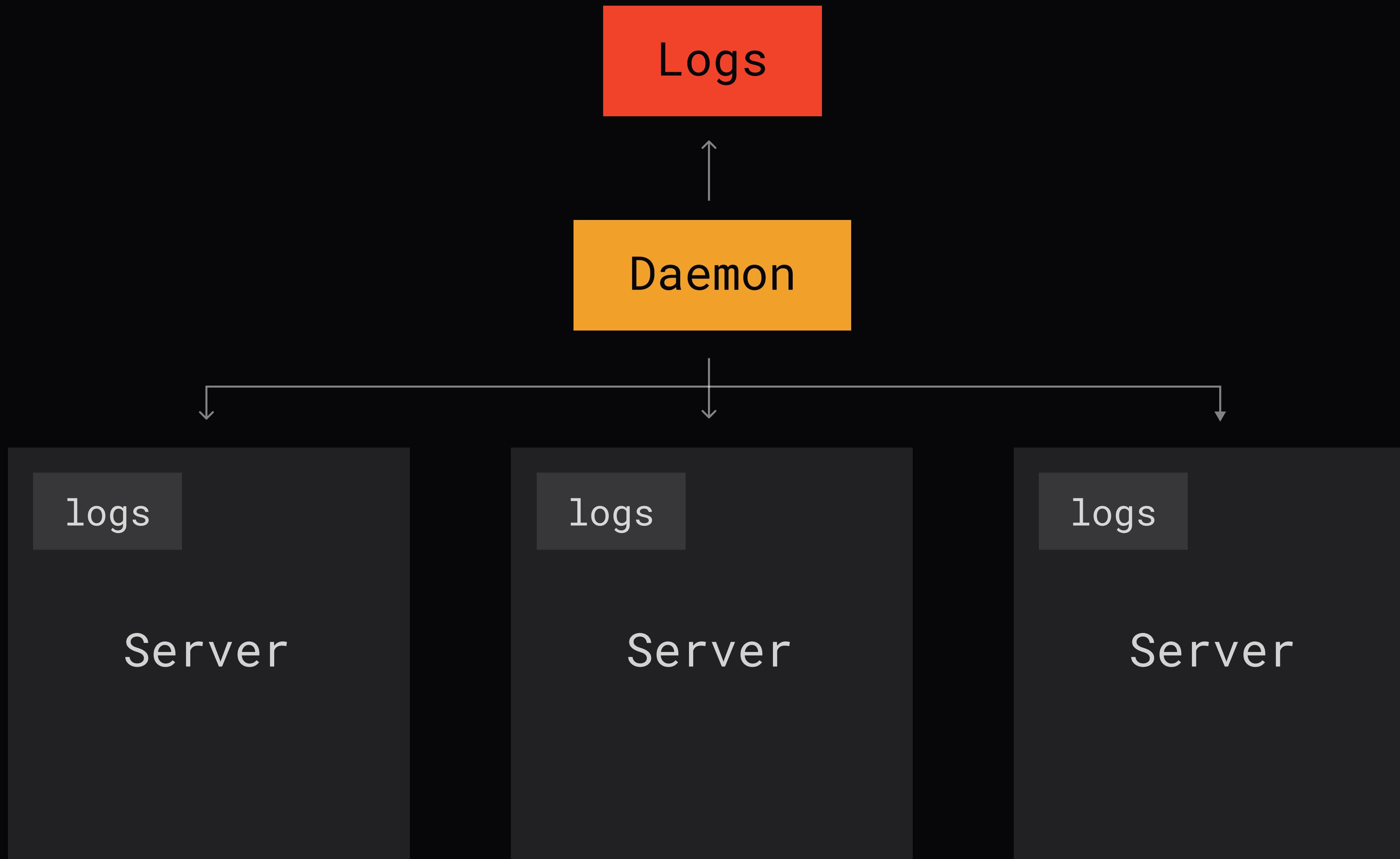
logs

Server

#2



#3





Search in the last 5 minutes ▾



Not updating ▾

Saved searches ▾



gl2\_source\_input:58be0be8f0e1fc0e94f80450



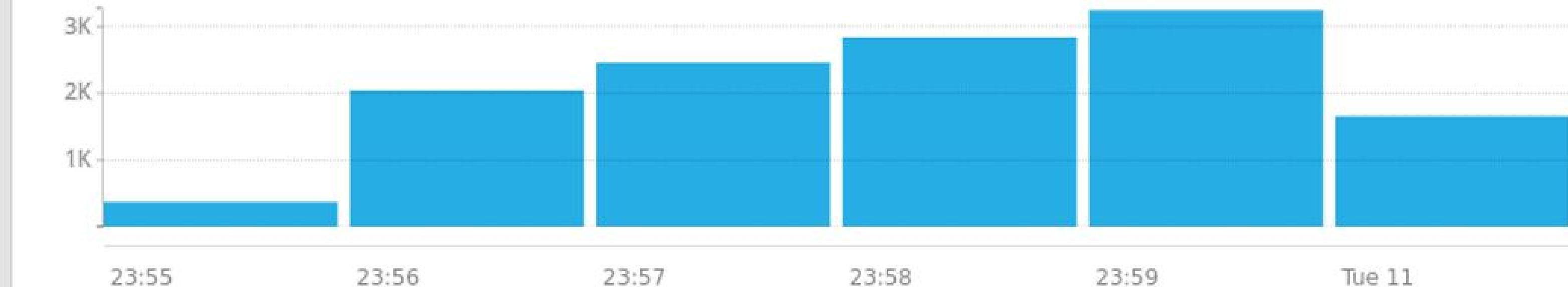
## Search result

Found 12,529 messages in 26 ms, searched in 1 index.

Results retrieved at 2017-04-11 00:00:31.

[Add count to dashboard ▾](#)[Save search criteria](#)[More actions ▾](#)[Fields](#)[Decorators](#)[Default](#)[All](#)[None](#)[Filter fields](#) [@timestamp](#) [@version](#) [category](#) [destip](#) [destPort](#) [mac](#) [message](#) [NetworkSecurityGroup](#) [operationName](#) [protocol](#)

## Histogram

[Year, Quarter, Month, Week, Day, Hour, Minute](#)

## Messages

[Previous](#)[1](#)[2](#)[3](#)[4](#)[5](#)[6](#)[7](#)[8](#)[9](#)[10](#)[Next](#)[Timestamp](#) [source](#)

2017-04-11 00:00:29.027

unknown

[%{Message}](#)

2017-04-11 00:00:29.027

unknown

[%{Message}](#)

2017-04-11 00:00:29.027

unknown

[%{Message}](#)

# ЛОГИРУЕМ

/1 ошибки и исключения

---

/2 события и действия пользователей

---

/3 запросы к серверу и сторонним ресурсам

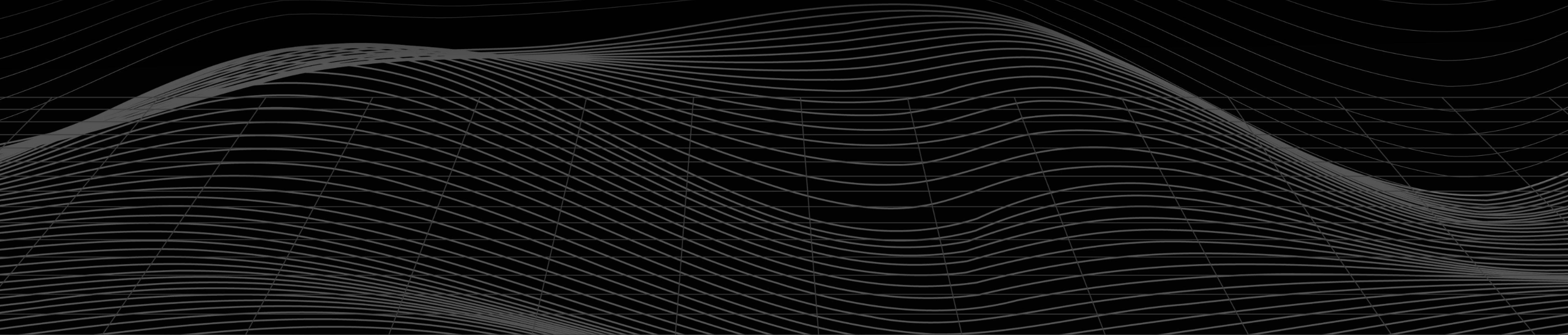
---

/4 информацию о состоянии приложения

---

ИСПОЛЬЗУЕМ  
УРОВНИ ЛОГИРОВАНИЯ

ПЕРСОНАЛЬНЫЕ ДАННЫЕ  
НЕ ЛОГИРУЕМ



# СИСТЕМЫ ЛОГИРОВАНИЯ



# FAQ

Логирование

# ТРЕЙСИНГ

Terminal: System Design × + ▾

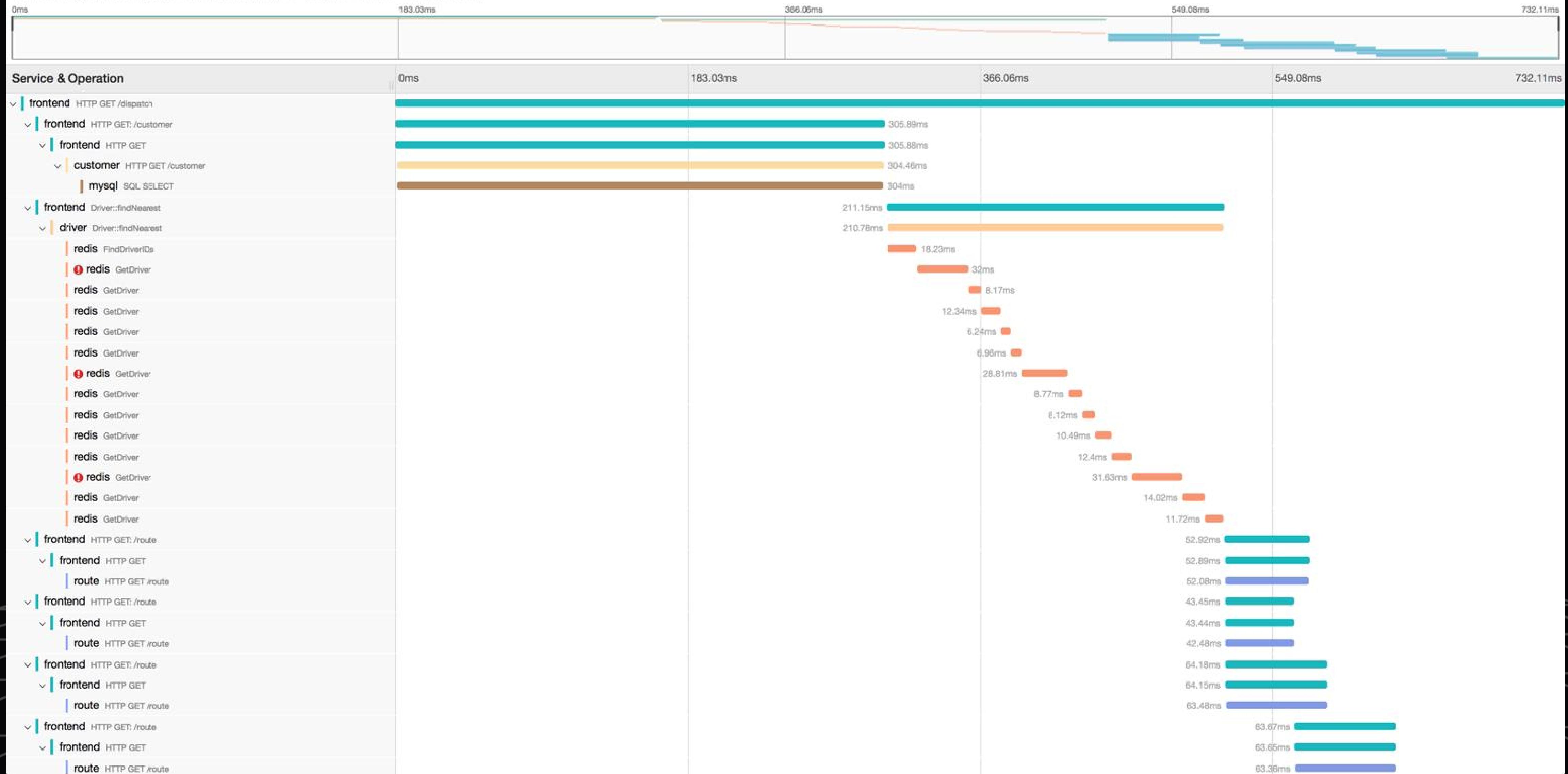


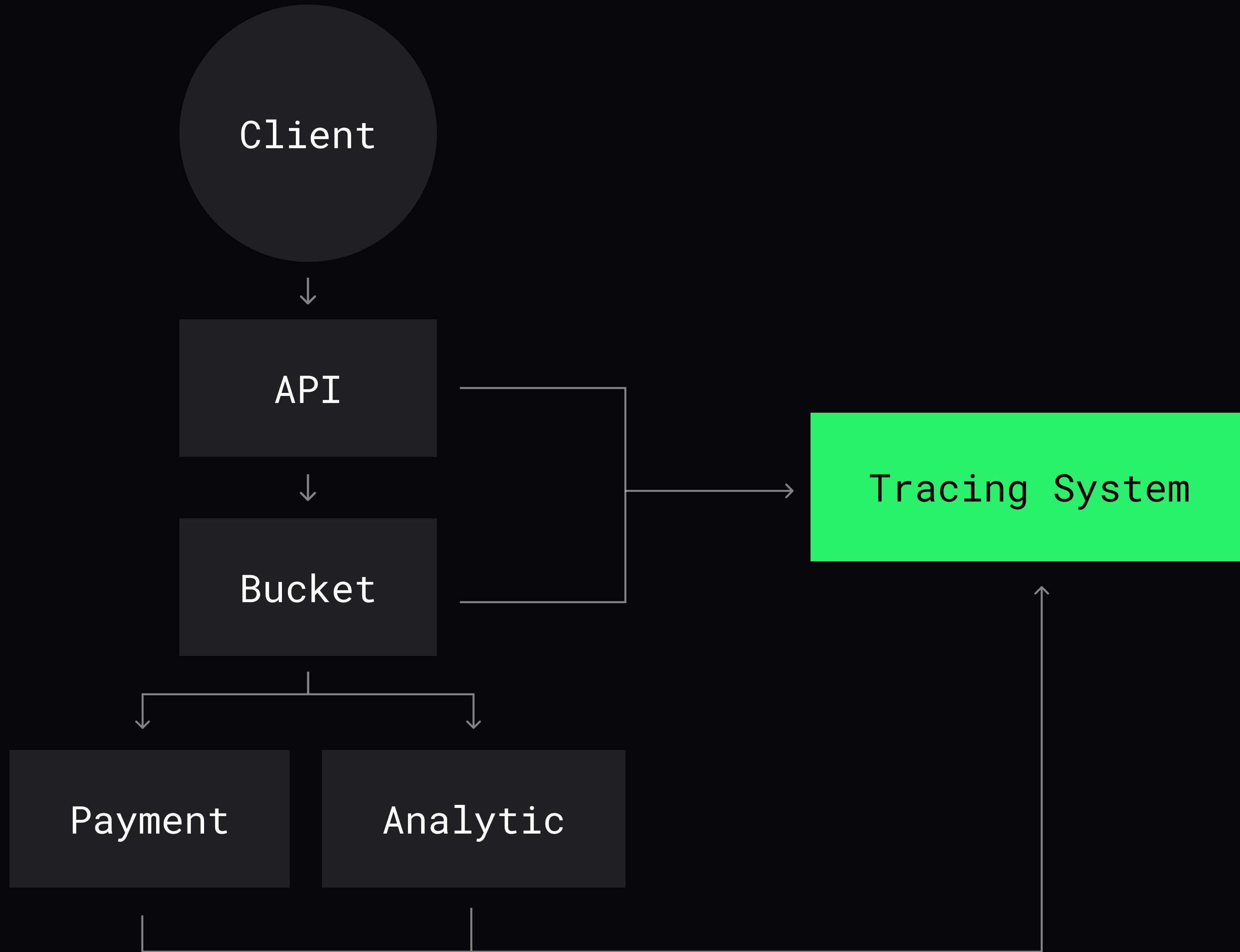
# ТРЕЙСИНГ

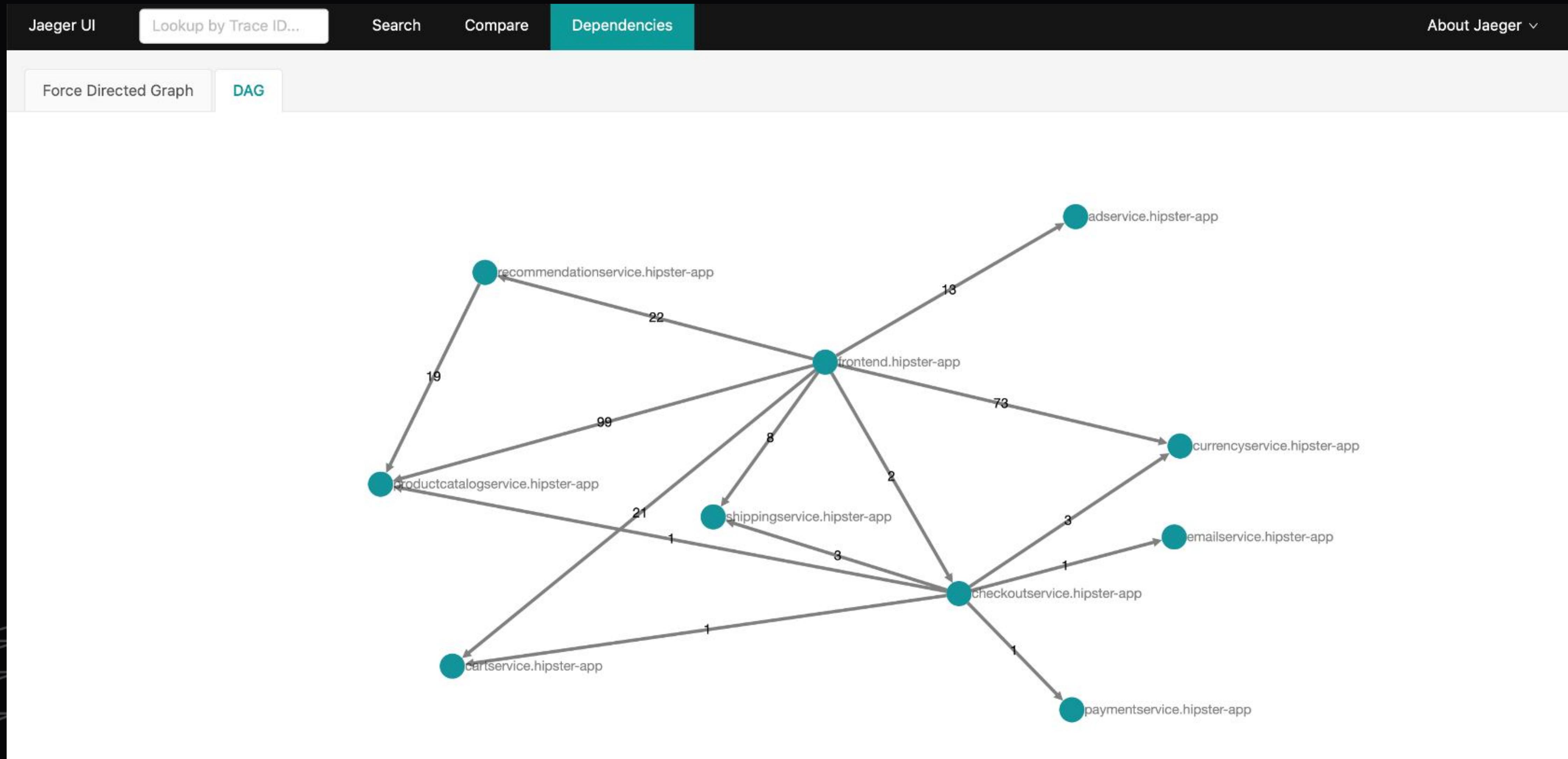
Инструмент для трассировки и анализа распределенных запросов, благодаря которому можно определить, где происходят сбои, что вызывает низкую производительность, а также как происходит процесс выполнения конкретных запросов

## ▼ frontend: HTTP GET /dispatch

Trace Start: July 20, 2018 2:48 PM | Duration: 732.11ms | Services: 6 | Depth: 5 Total Spans: 51

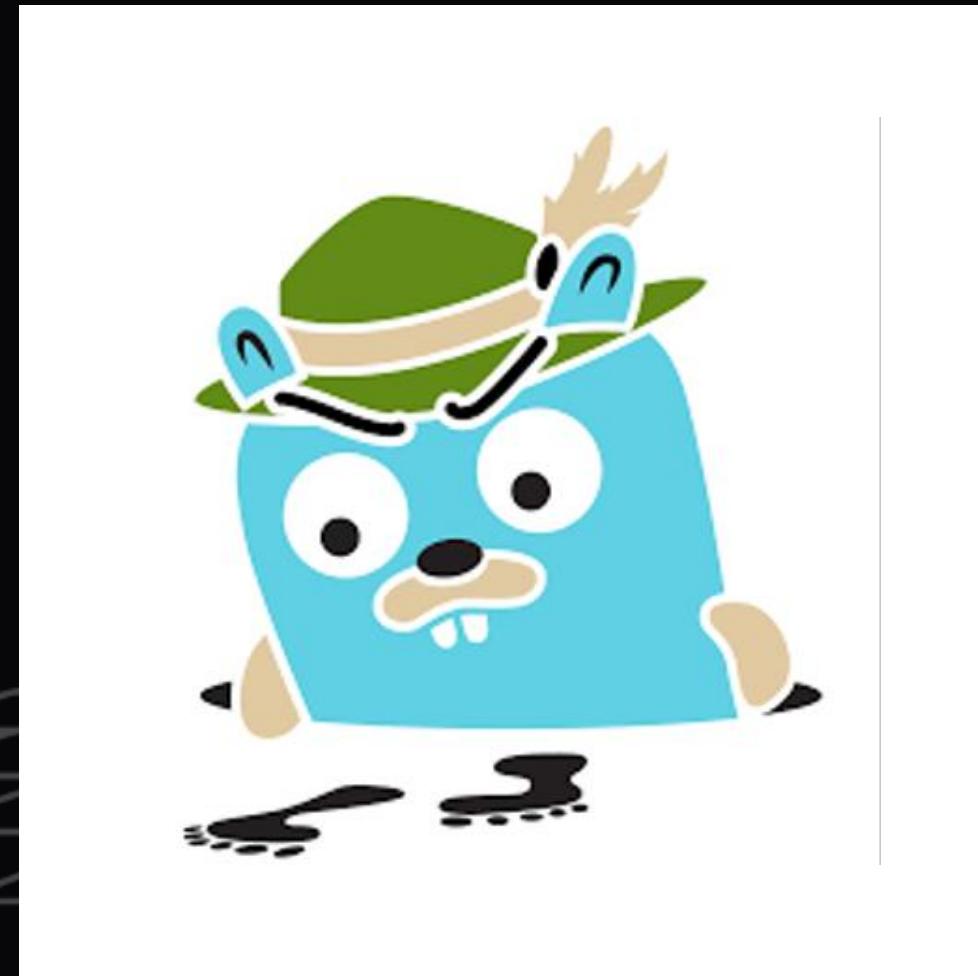




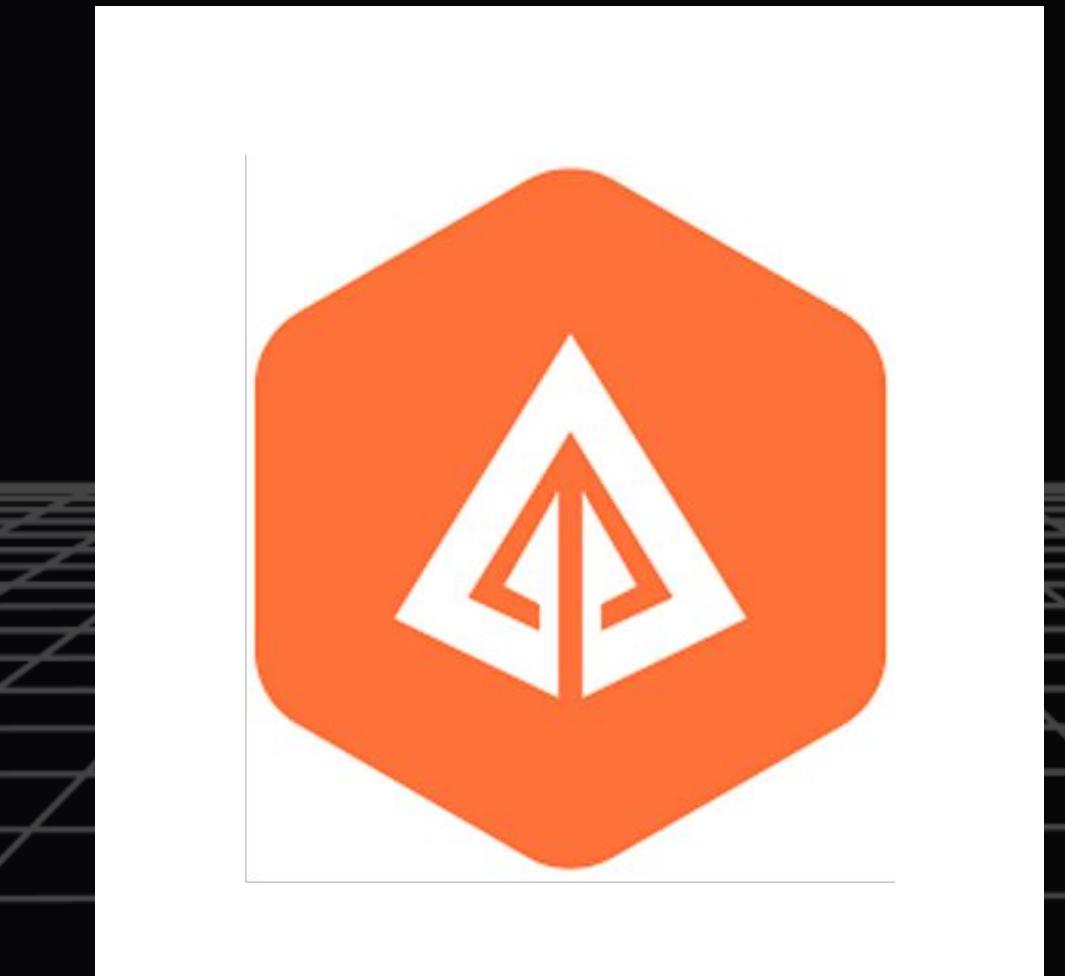


# СИСТЕМЫ ТРЕЙСИНГА

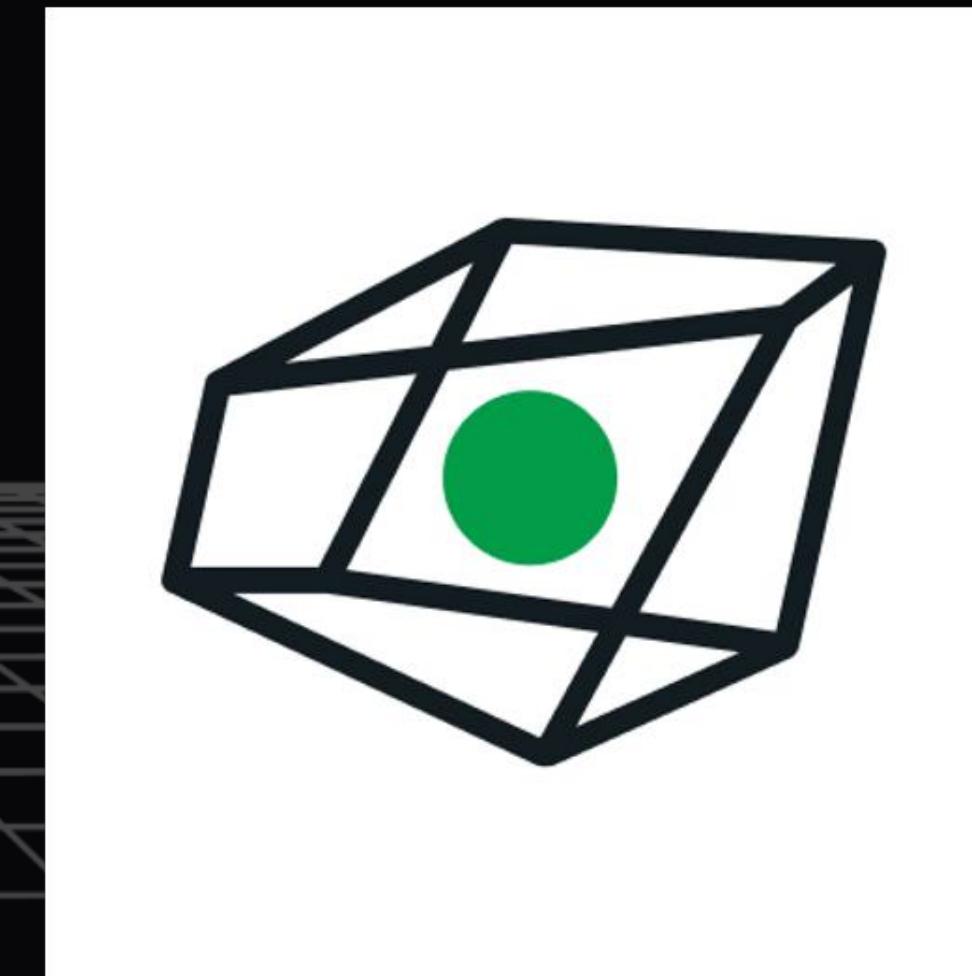
Jaeger



Zipkin



LightStep



# **FAQ**

Трейсинг

# **НЕПРЕРЫВНОЕ ПРОФИЛИРОВАНИЕ**

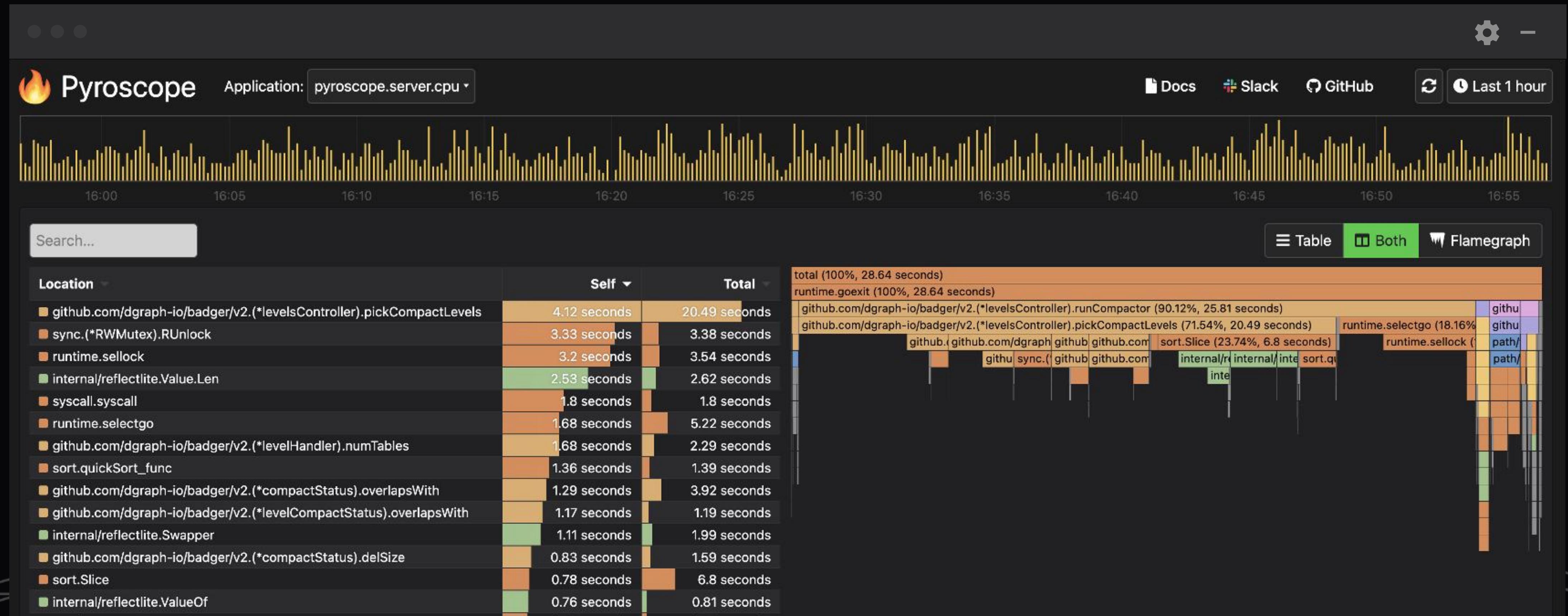
Terminal: System Design × + ▾

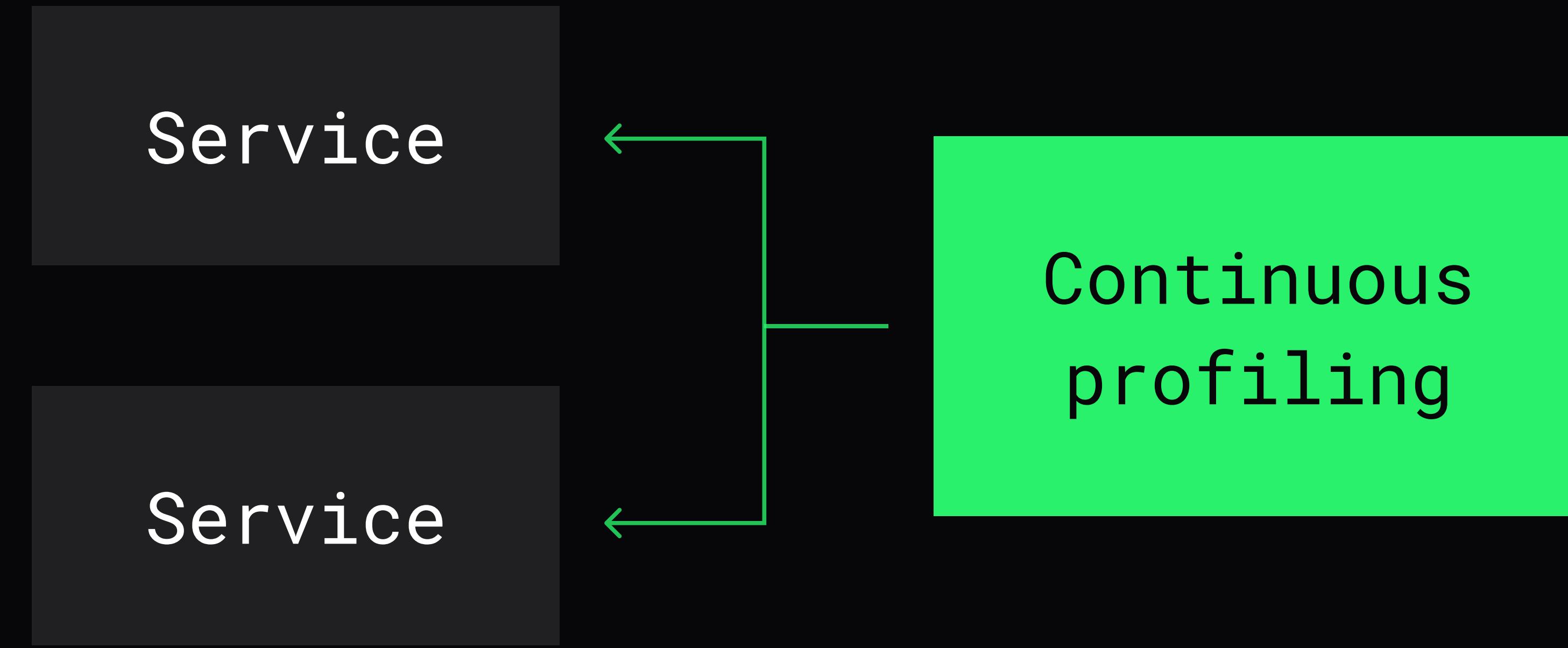


# НЕПРЕРЫВНОЕ ПРОФИЛИРОВАНИЕ

Метод анализа и измерения производительности программного обеспечения, который позволяет получать данные о работе программы в режиме реального времени







# СИСТЕМЫ НЕПРЕРЫВНОГО ПРОФИЛИРОВАНИЯ

parca



Pyroscope

# ЕЩЕ ЕСТЬ SENTRY

Grafana × +

Empower Plant Jane Schmidt

All Projects All Environments Last 14 days

**IntegrityError** pytest.runttest.call tests/sentry/api/endpoints/test\_sentry\_apps\_stats ISSUE # DJANGO-32 EVENTS 5 USERS 0 ASSIGNEE

IntegrityError('duplicate key value violates unique constraint "sentry\_project\_organization\_id..."')

Resolve Ignore Share Open in Discover ⌂ ⌂

Details Activity User Feedback Attachments Tags Events Merged Issues Similar Issues

Event bc74353509dn098y08hf Older Newer ▶

March 24, 2021 9:10:21 PM UTC JSON (14.4kb)

charlotte\_alameda@sentry.io ID: 387936 Chrome Version: 85.0.5353 Mac OSX Version: 10.15.1

Dashboards Activity Stats Settings

TAGS

browser Chrome 89.0.4389 browser.name Chrome duration 16675 environment prod groupID 2292909355

interval 1000 level warning organization 557498 organization.slug caffeinatedduo os.name Linux

plan aml\_f plan.category metered plan.max\_members 1 plan.name Developer plan.tier aml

EXCEPTION (most recent call first)

App Only Full Raw

IntegrityError

IntegrityError('duplicate key value violates unique constraint "sentry\_project\_organization\_id\_slug\_BacIdSae'

All Environments LAST 24 HOURS

LAST 30 DAYS

LAST SEEN

3 minutes ago in release 25c62c66dd9e

FIRST SEEN

7 months ago in release 026451250466

Linked Issues

+ Link GitHub Issue

+ Link Jira Issue

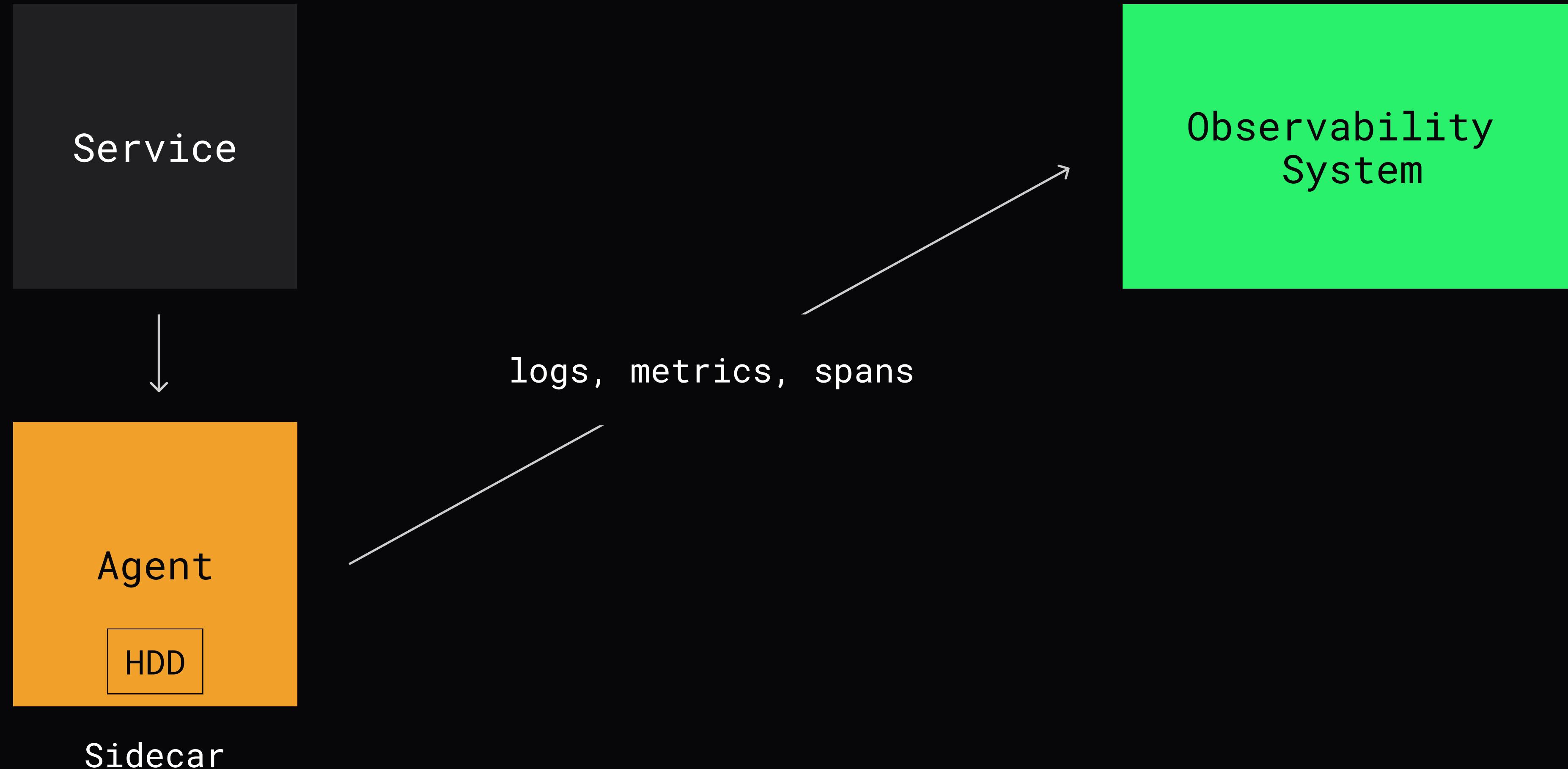
This screenshot shows a Grafana dashboard integrated with Sentry. The main content area displays an issue titled 'IntegrityError' from the 'DJANGO-32' project. The issue details include the error message 'IntegrityError('duplicate key value violates unique constraint "sentry\_project\_organization\_id..."')'. Below the details, there's a section for the event 'bc74353509dn098y08hf' with a timestamp of 'March 24, 2021 9:10:21 PM UTC' and a file size of 'JSON (14.4kb)'. The event is associated with a user 'charlotte\_alameda@sentry.io' (ID: 387936), a 'Chrome' browser (Version: 85.0.5353), and a 'Mac OSX' operating system (Version: 10.15.1). The bottom section shows the exception stack trace with the most recent call first, listing 'IntegrityError'. On the left, a sidebar menu includes 'Projects', 'Issues', 'Discover', 'Performance', 'Alerts', 'Releases', 'User Feedback', 'Dashboards', 'Activity', 'Stats', and 'Settings'. On the right, there are two sections: 'All Environments' showing data for the last 24 hours and 30 days, and 'Linked Issues' with options to link to GitHub or Jira.

# КАК НАДЕЖНО ПОСТАВЛЯТЬ ДАННЫЕ ТЕЛЕМЕТРИИ?

Service

— logs, metrics, spans —→

Observability System



# FAQ

**Observability**

Трейсинг, мониторинг, логирование,  
непрерывное профилирование

# **ДОМАШНЕЕ ЗАДАНИЕ**

ДО ВСТРЕЧИ  
НА СЛЕДУЮЩЕМ  
ЗАНЯТИИ!