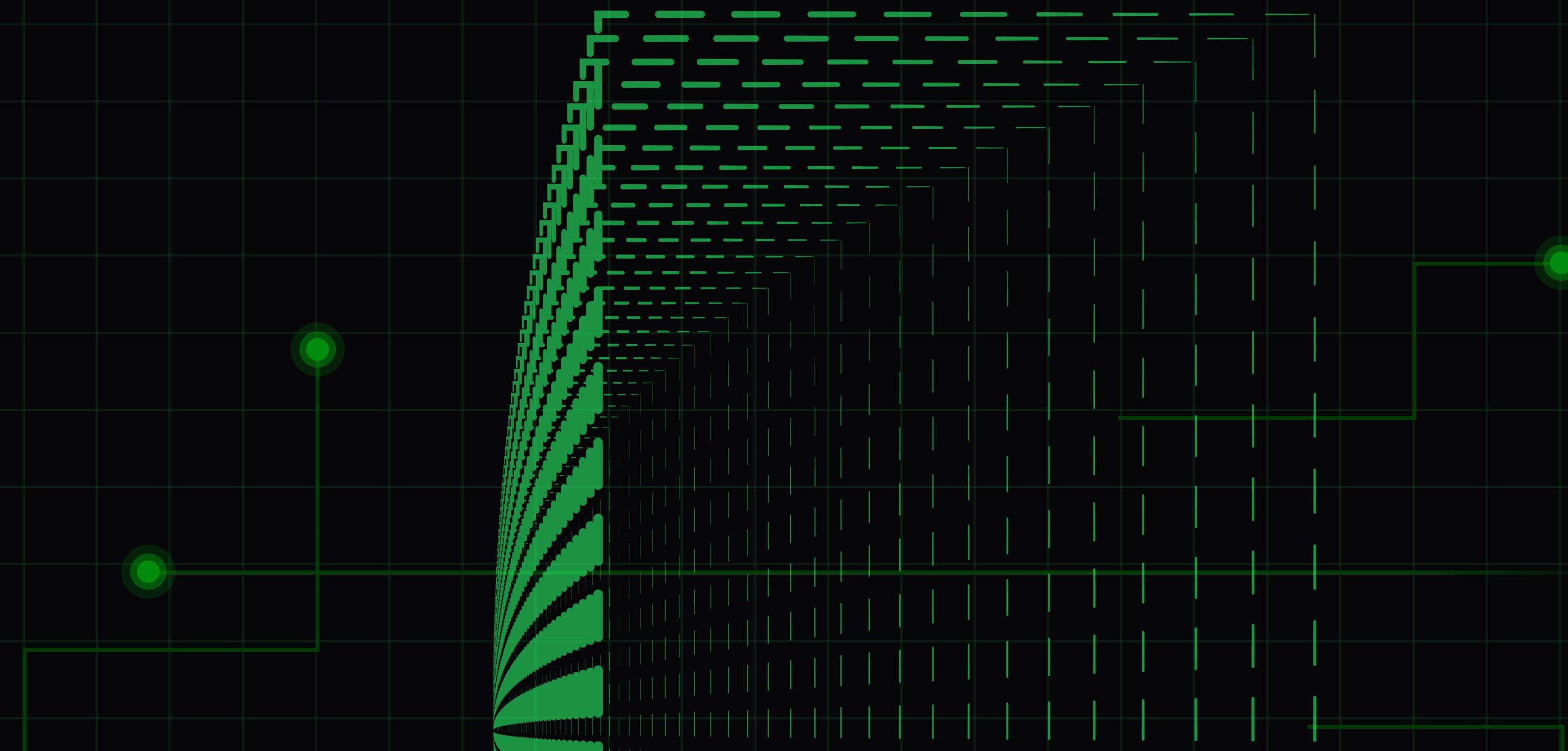
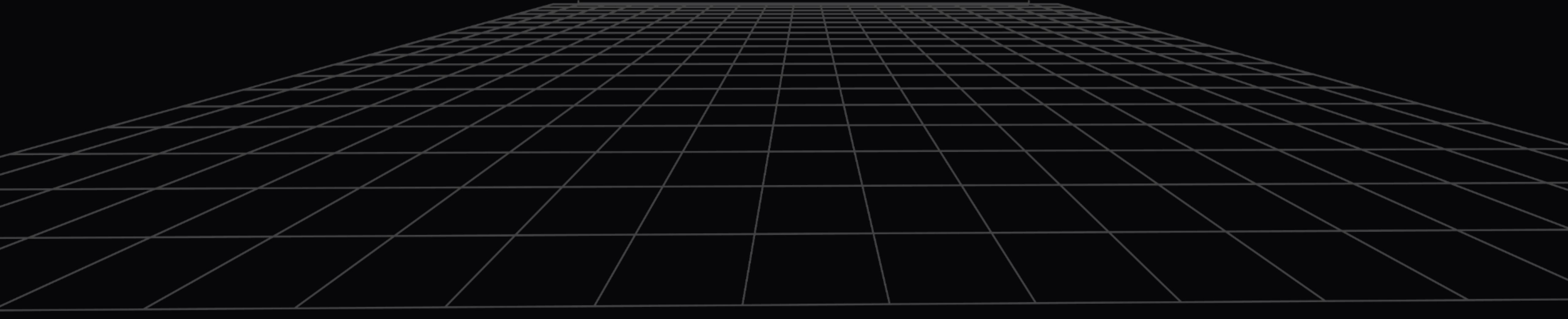


SYSTEM DESIGN

Распределенное хранение данных

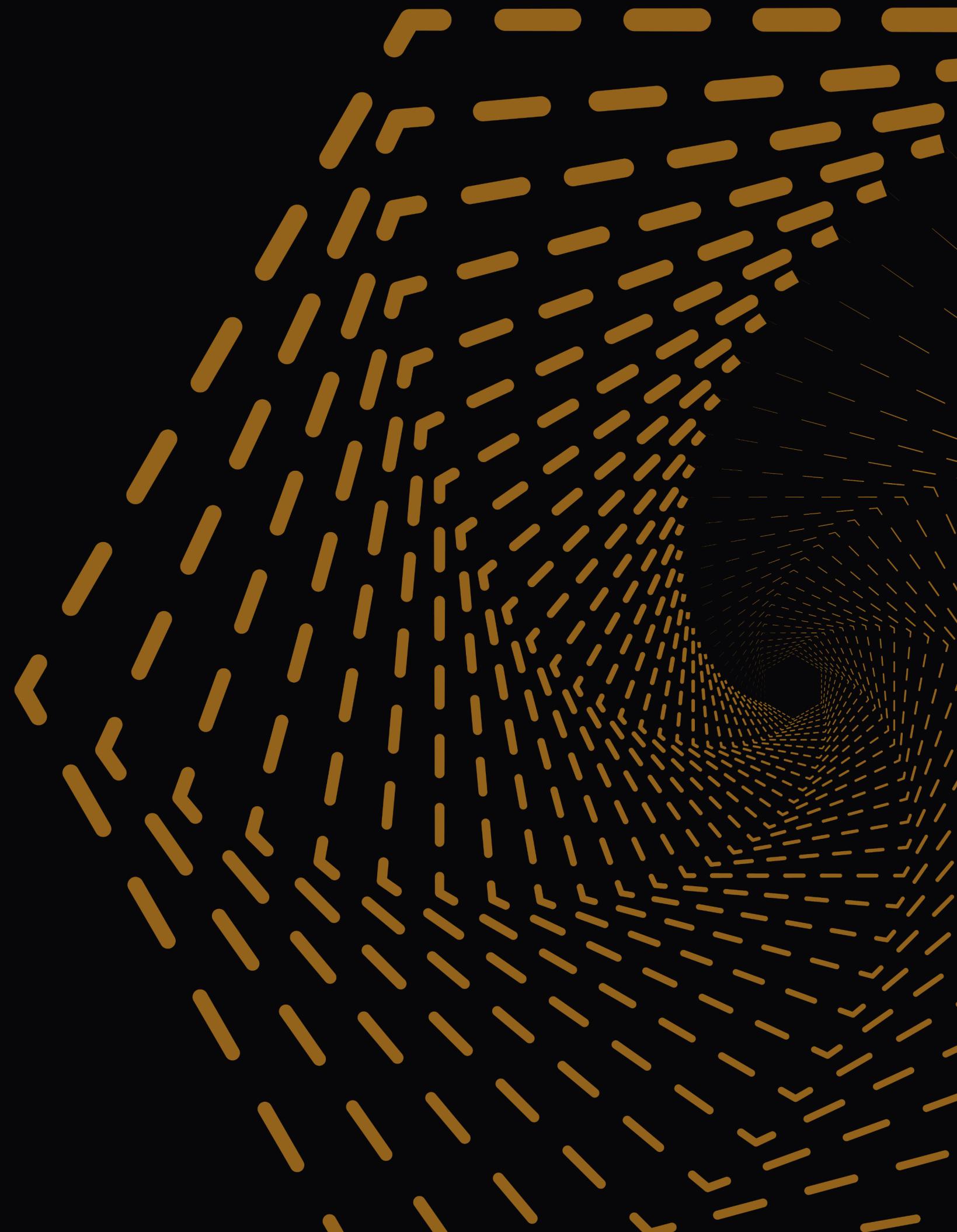


ПРОВЕРЬ ЗАПИСЬ



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. Бэкапы
2. Репликация
3. CAP теорема
4. Партиционирование
5. Шардирование
6. Дополнительное

БЭКАПЫ

Terminal: System Design × + ▾



БЭКАПЫ

Главное назначение резервного копирования
– восстановление данных после их потери или
повреждения, а также для клонирования базы
данных, например для тестирования

РЕКОМЕНДУЕТСЯ СОЗДАВАТЬ РЕЗЕРВНЫЕ КОПИИ БАЗ ДАННЫХ РЕГУЛЯРНО



Это может быть ежедневно, еженедельно или
ежемесячно, в зависимости от важности данных
и частоты их обновления

ОБЫЧНО РЕКОМЕНДУЕТСЯ ХРАНИТЬ НЕ МЕНЕЕ ТРЕХ ПОСЛЕДНИХ ВЕРСИЙ РЕЗЕРВНЫХ КОПИЙ

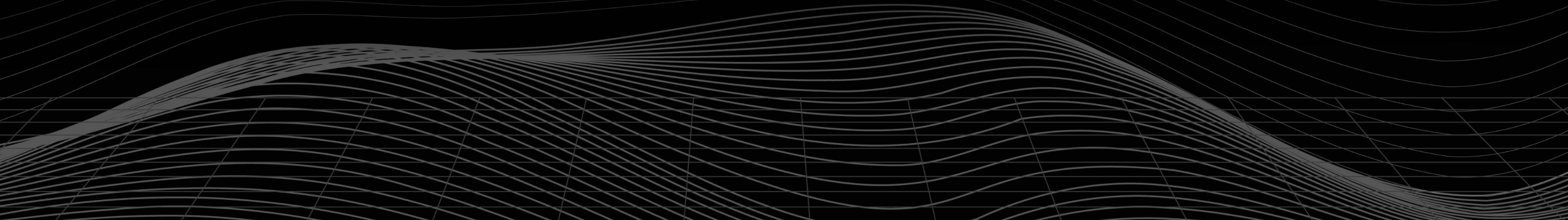
Это обеспечит возможность восстановления данных в случае обнаружения проблемы или ошибки, которая может быть замечена только после нескольких дней или недель

FAQ

Бэкапы

РЕПЛИКАЦИЯ

ЧЕМ РЕПЛИКАЦИЯ
ОТЛИЧАЕТСЯ ОТ БЭКАПОВ?



Terminal: System Design × + ▾



БЭКАПЫ И РЕПЛИКАЦИЯ

Бэкап – это резервное копирование содержимого диска с целью последующего восстановления

Репликация – это процесс создания и поддержания копий (клонов) базы данных на нескольких серверах

Terminal: System Design × + ▾



Важно понимать, что 1ТБ данных в базе данных будет подниматься из бэкапа на HDD примерно **2,5 часа**:

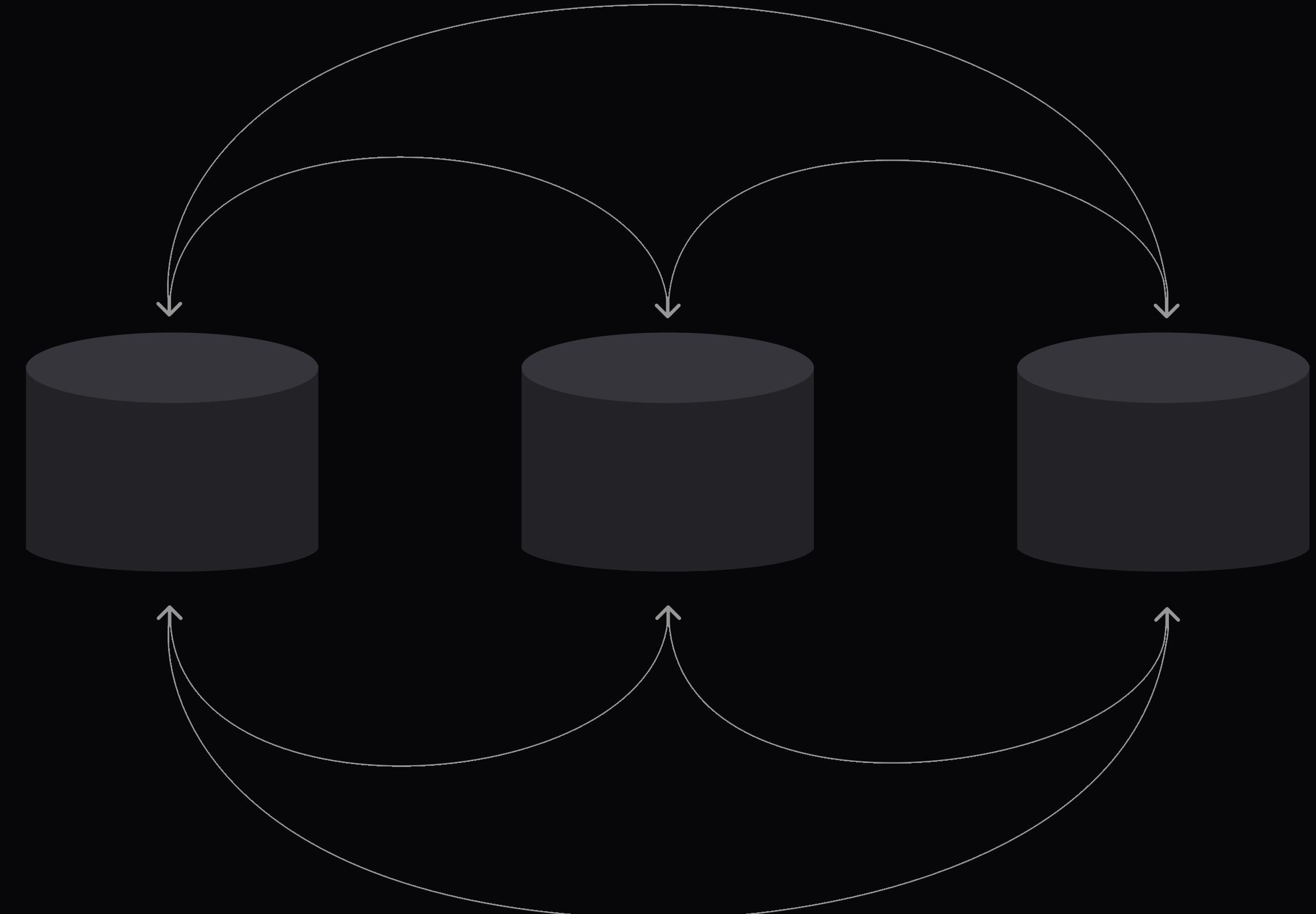


$1\ 000\ 000 \text{ (МБ)} / 100 \text{ (МБ/с)} / 60 \text{ (с)} / 60 \text{ (м)} = 2,5 \text{ часа}$

ЗАЧЕМ ТОГДА НУЖНЫ БЭКАПЫ?

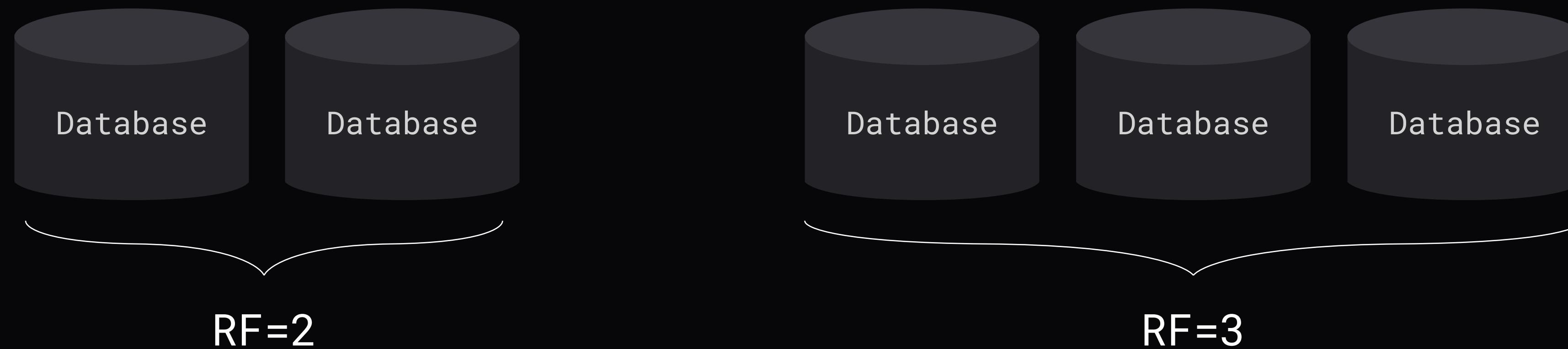
СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ РЕПЛИКАЦИИ

- Увеличение отказоустойчивости
- Масштабирование чтения
- Размещение данных ближе к пользователям



**РЕПЛИКАЦИЯ
НЕ МАСШТАБИРУЕТ ЗАПИСЬ!**

REPLICATION FACTOR



ВИДЫ РЕПЛИКАЦИИ

/1 master-slave / master-master / master-less

/2 sync / async / semisync

/3 logical / physical

/4 push / pull

подтема №1

ВИДЫ РЕПЛИКАЦИИ

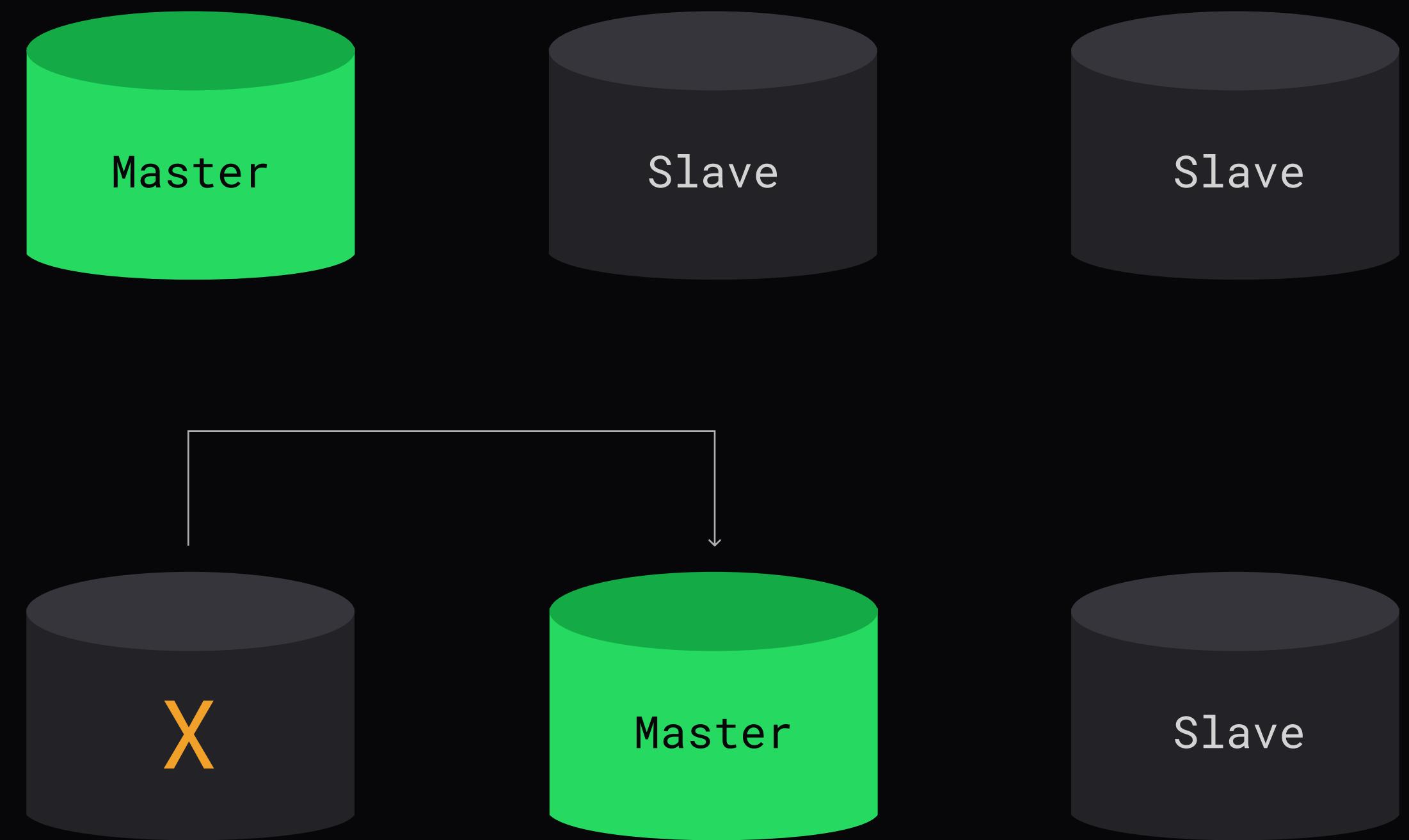
Куда писать данные...

MASTER – SLAVE

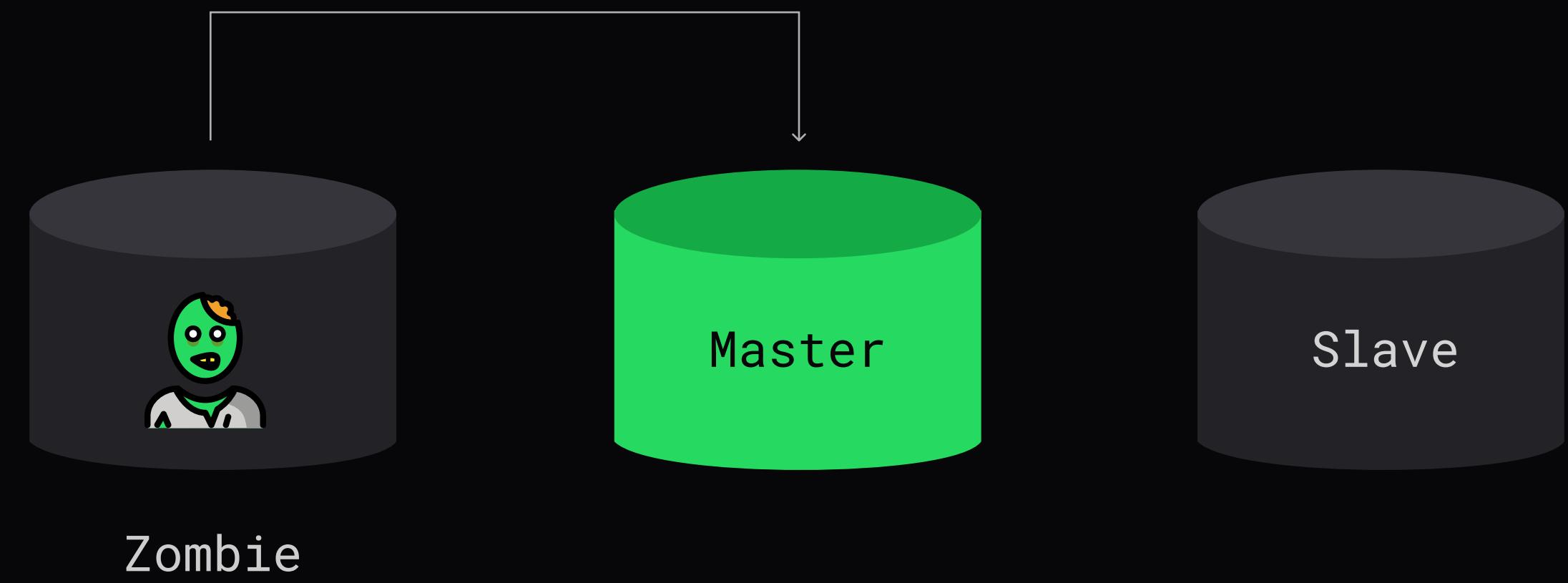
1. Пишем в мастер
2. Читаем из слейвов или из мастера
3. В случае падения мастера получаем downtime на запись



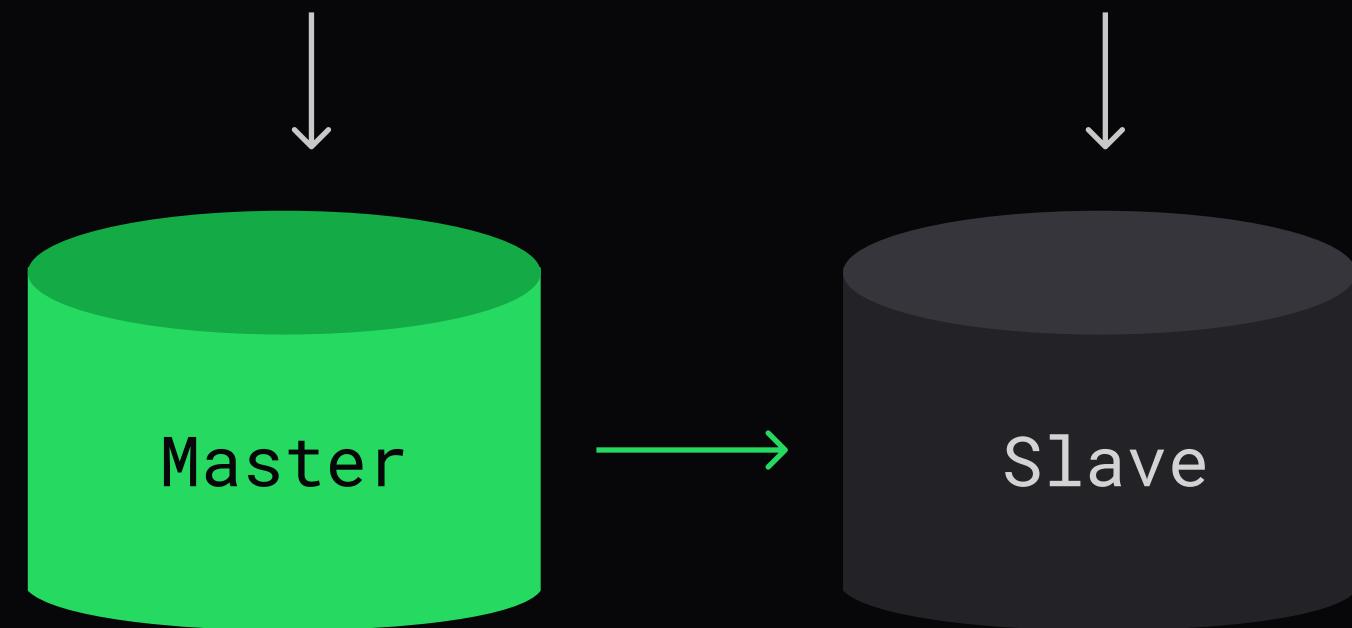
FAILOVER



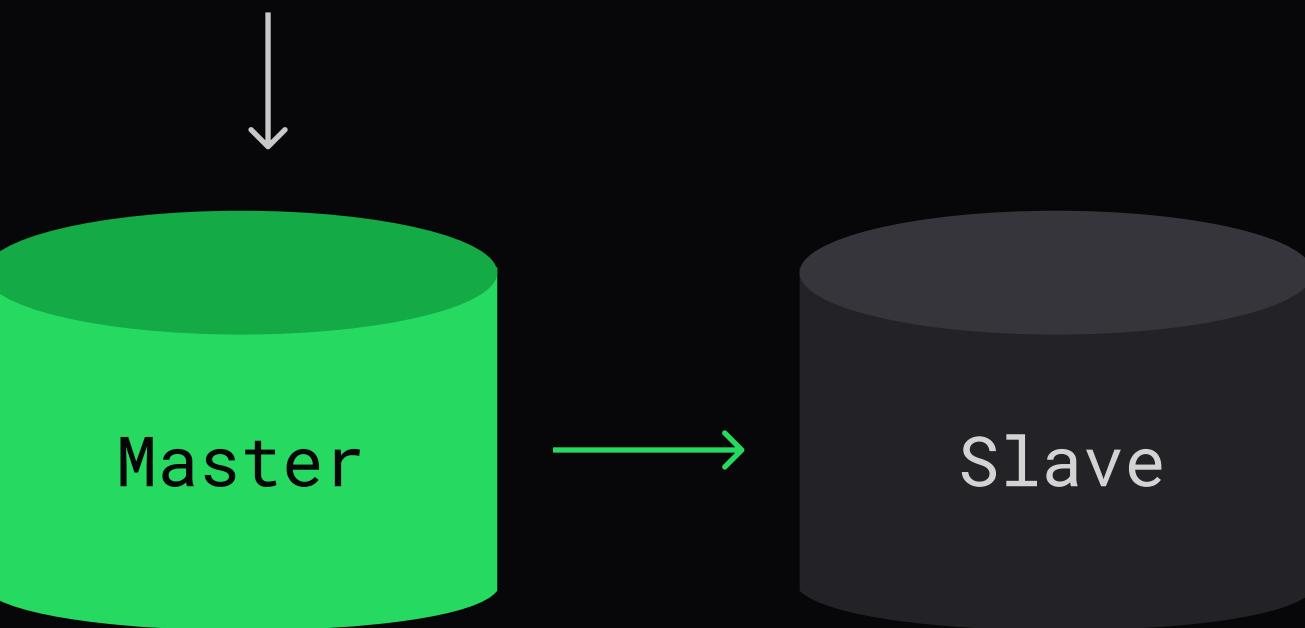
SPLIT BRAIN



HOT STANDBY

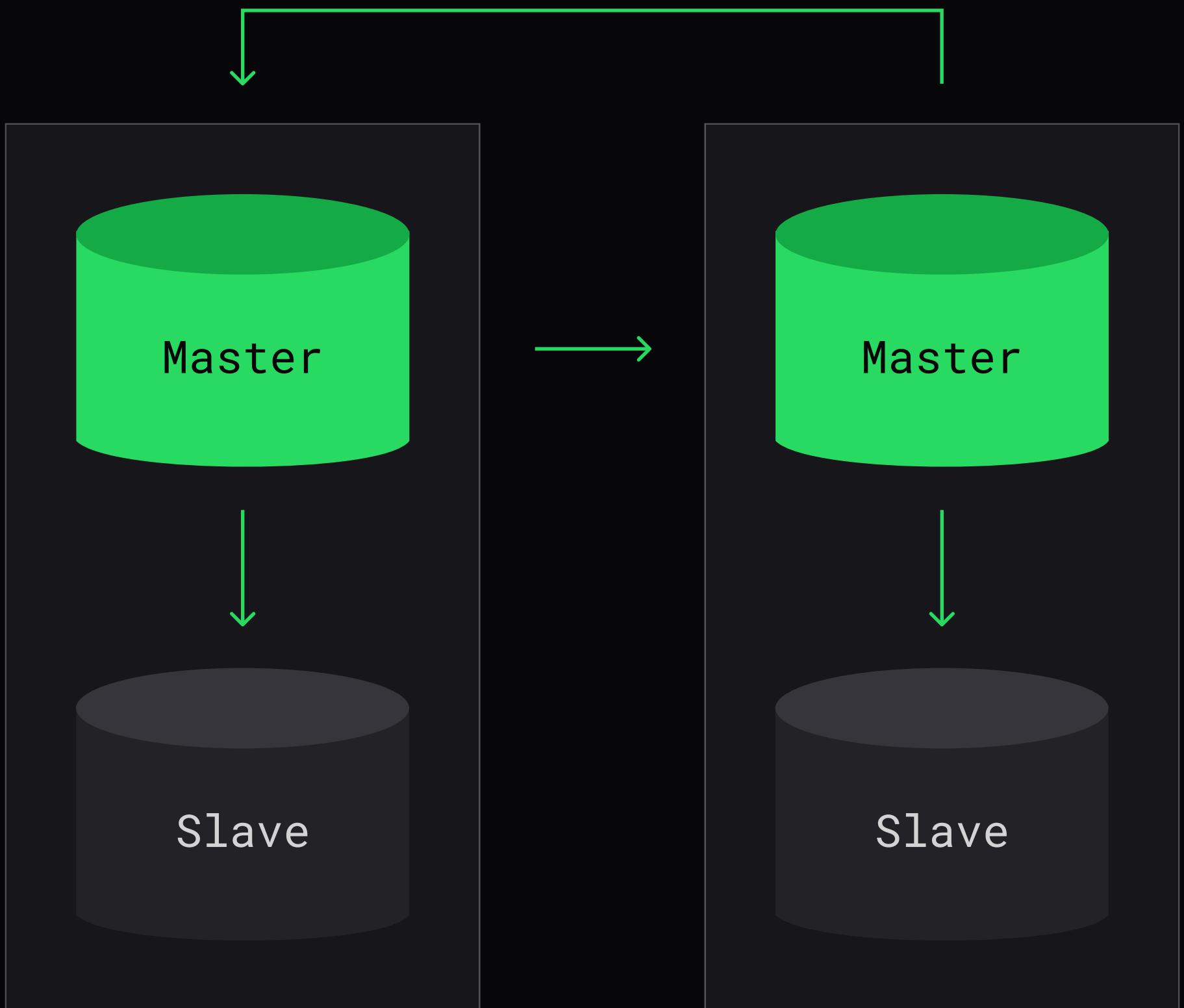


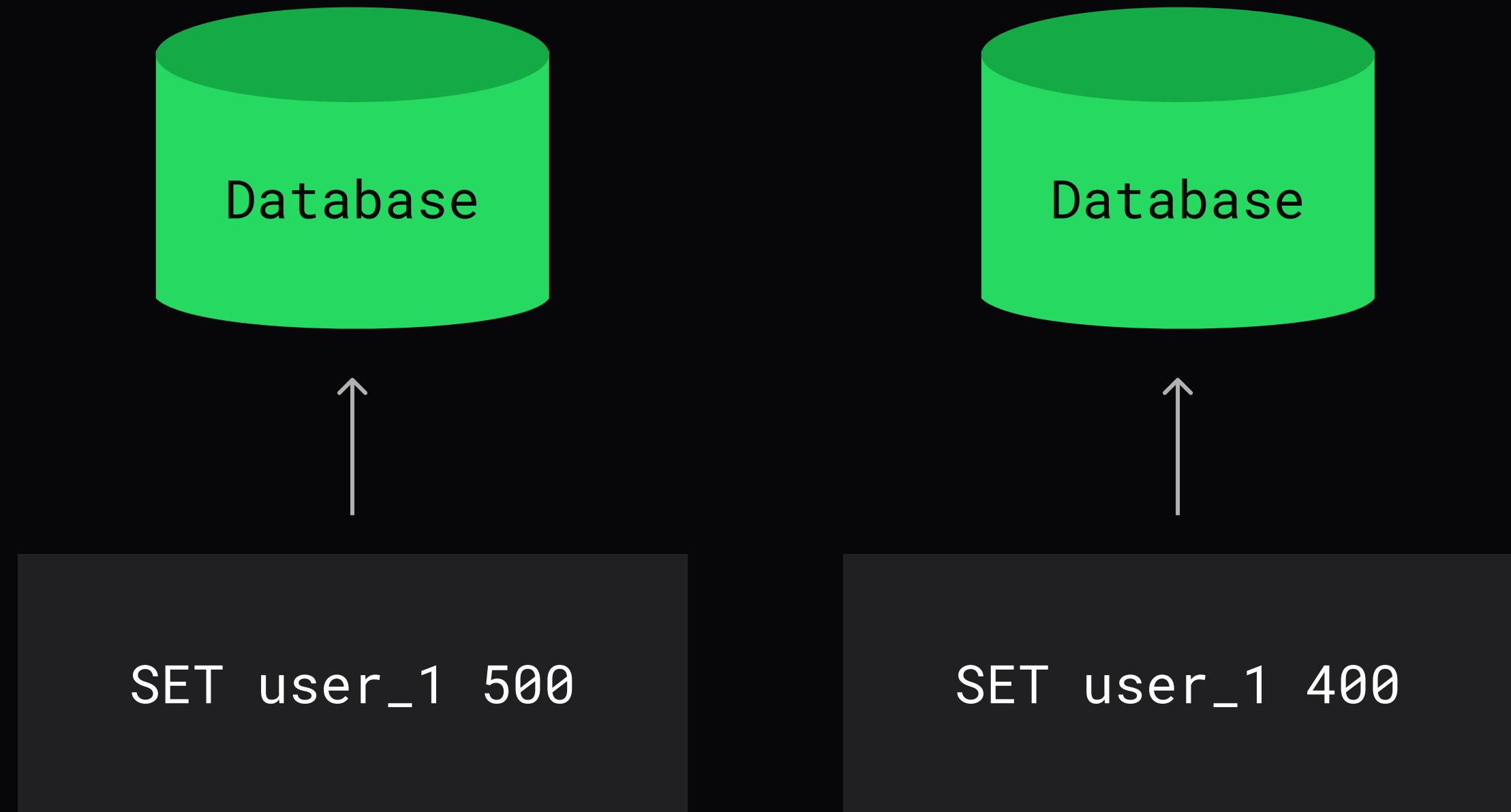
WARM STANDBY



MASTER – MASTER

1. Появляются конфликты
2. Пишем в разные мастера
3. Читаем из слейвов или мастеров
4. В случае падения мастера нет никакого downtime на запись





КАК РЕЗОЛВИТЬ КОНФЛИКТЫ?



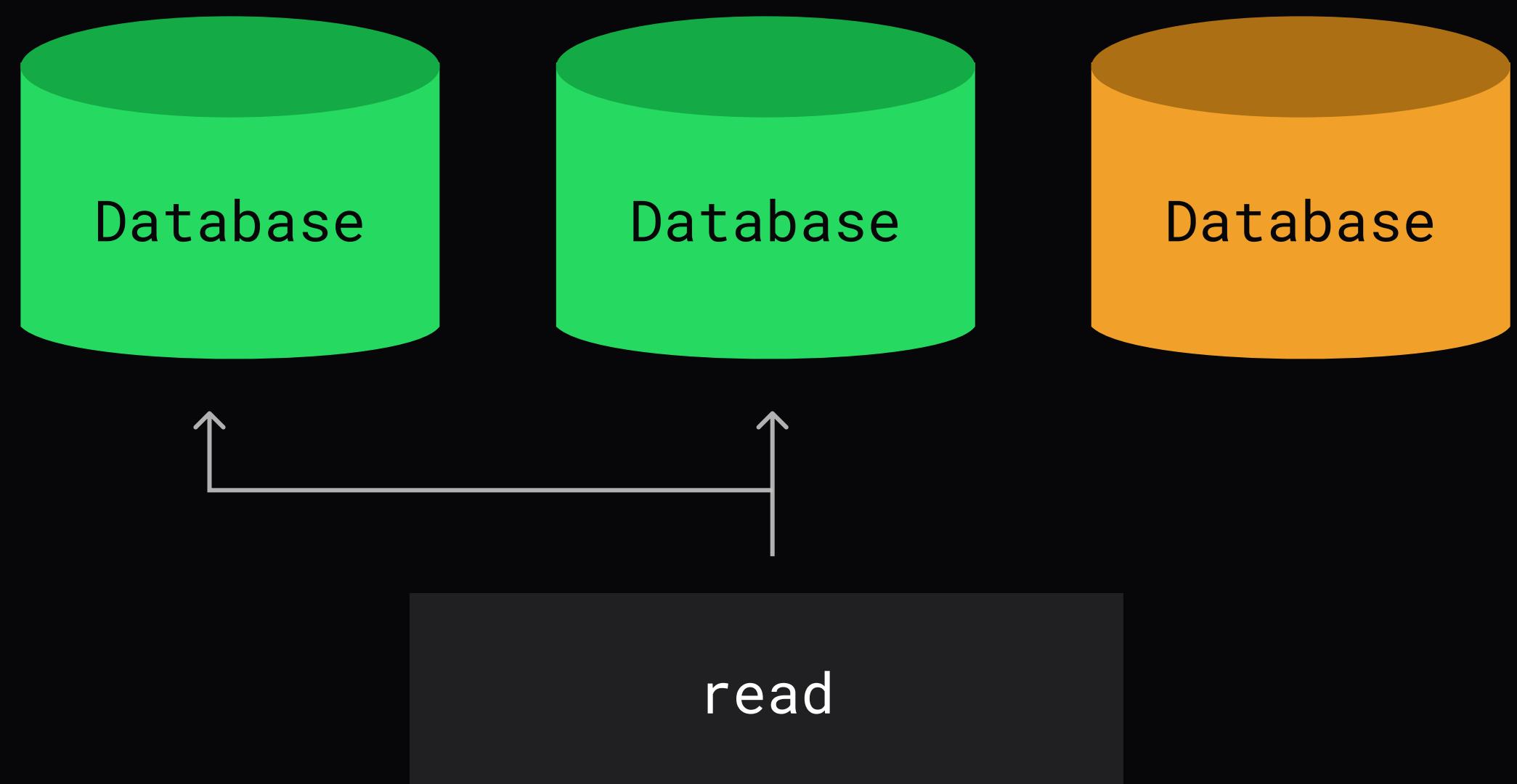
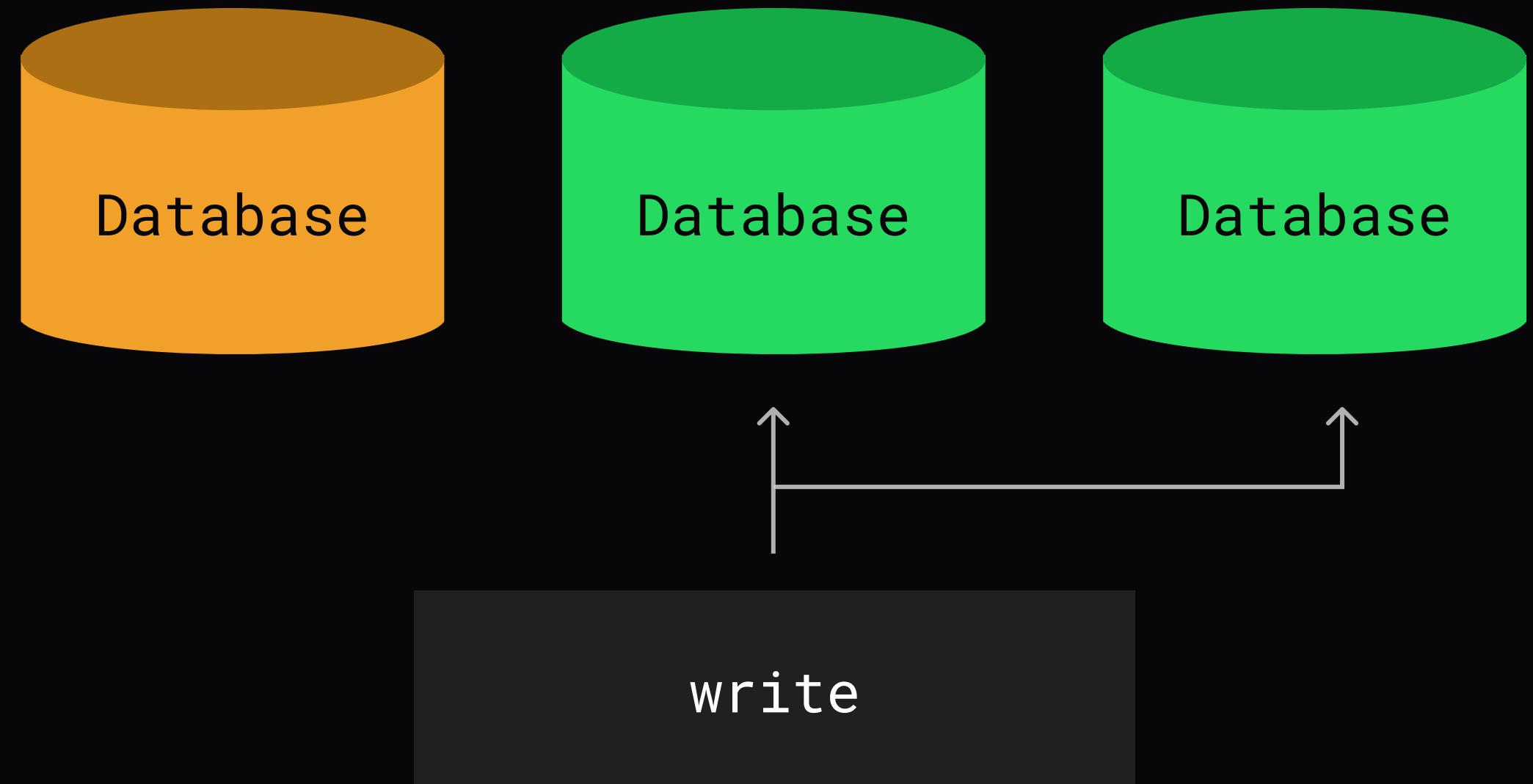
РАЗРЕШЕНИЕ КОНФЛИКТОВ

/1 LWW (Last Write Wins)

/2 Ранг реплик

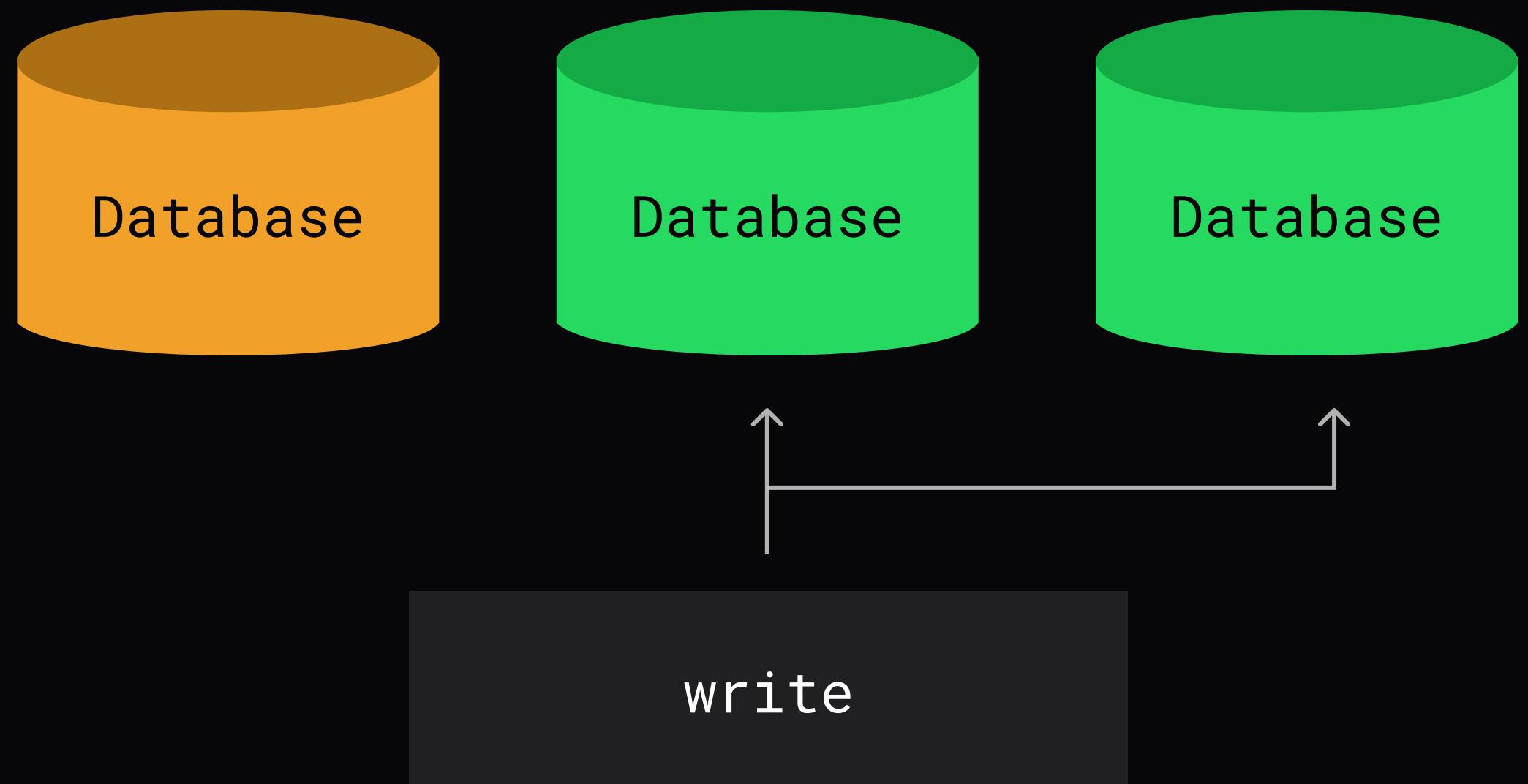
/3 Решение конфликтов на клиенте

/4 Conflict-replicated data type (CRDT)



MASTER – LESS

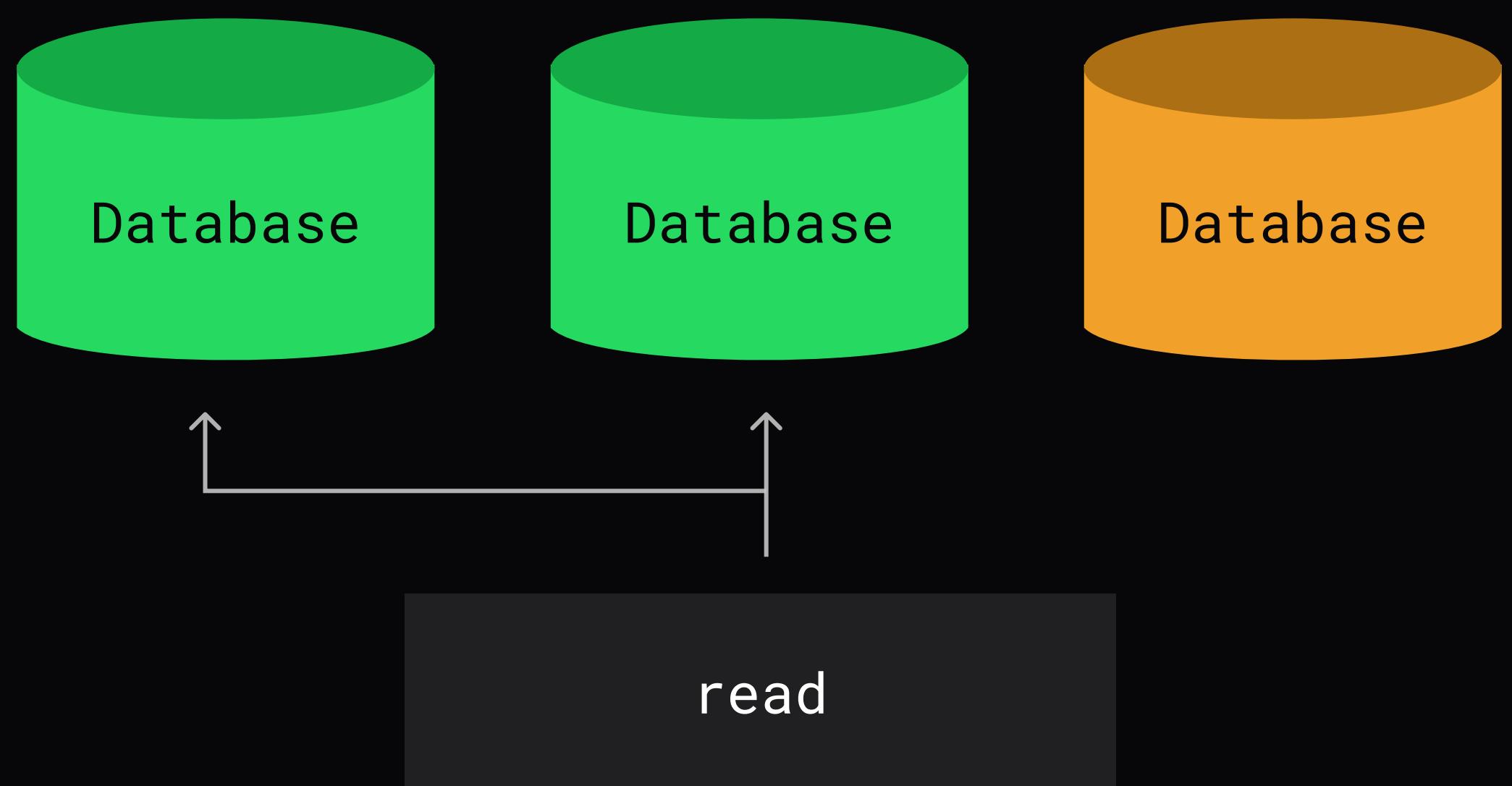
1. Пишем в определенные ноды
2. Читаем из определенных нод



1. $W + R > N$ – гарантируется строгая согласованность

2. $W + R \leq N$ – не гарантируется строгая согласованность

3. $R = 1$ и $W = N$ – система оптимизирована для быстрого чтения



4. $W = 1$ и $R = N$ – система

оптимизирована для быстрой записи

РЕЗЮМЕ

- /1 master-master для нескольких ДЦ

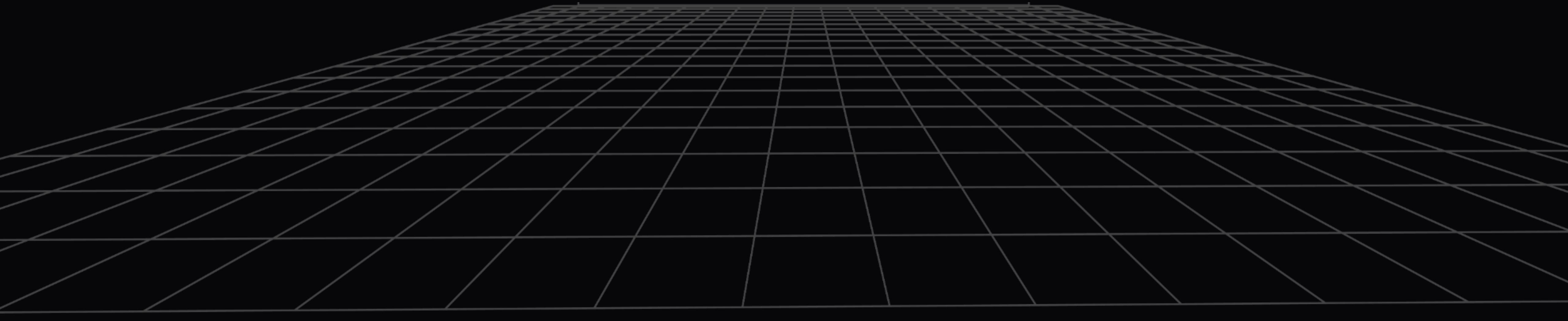
- /2 master-less для децентрализованных систем

- /3 master-slave для всего остального

FAQ

Виды репликации

master-slave, master-master, master-less



подтема №2

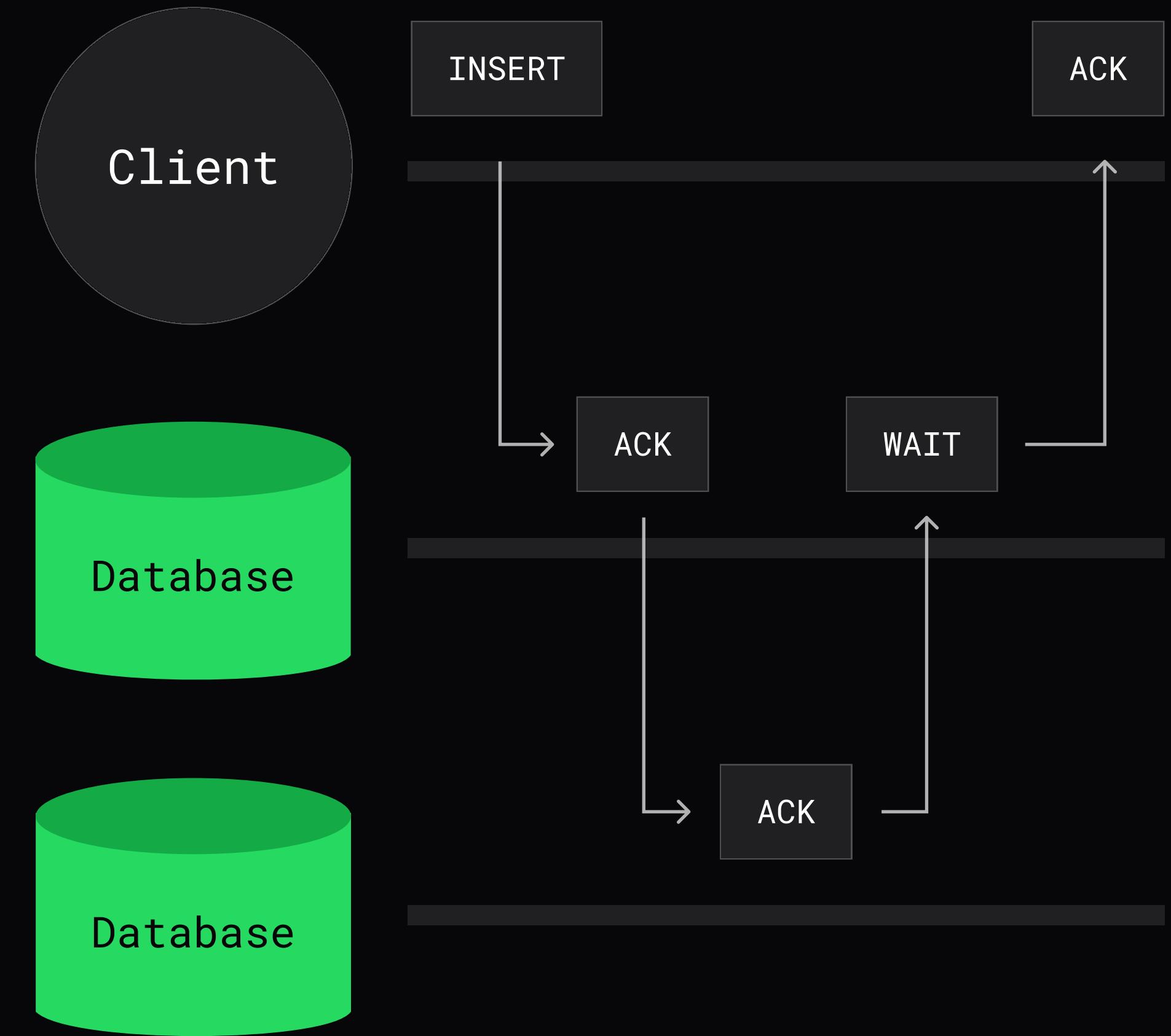
ВИДЫ РЕПЛИКАЦИИ

Когда синхронизировать данные...

СИНХРОННАЯ (SYNC)

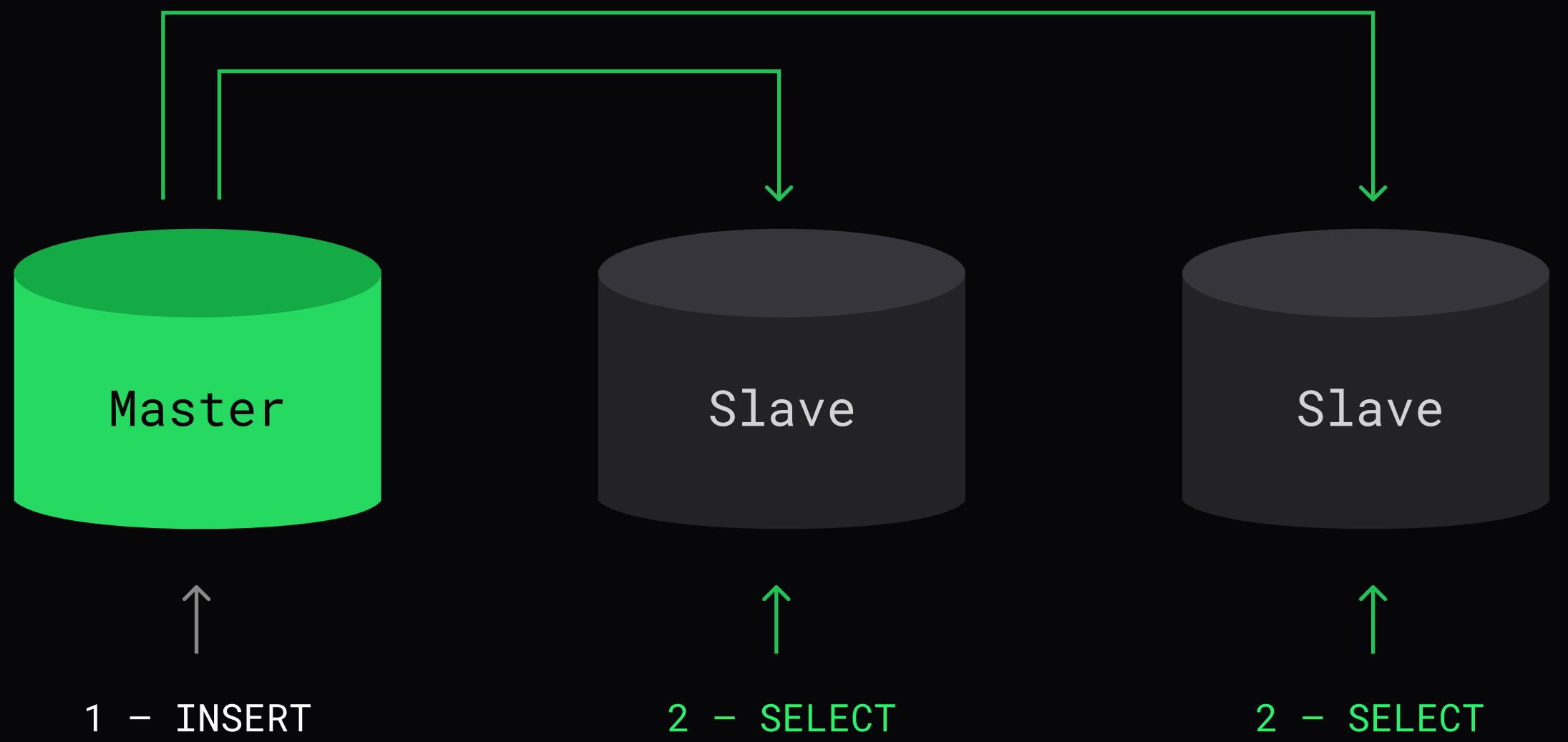
- i Local + remote commit
(данные доступны везде)

1. Локальная запись данных
2. Отправка данных на реплику
3. Получение подтверждения от реплики об удаленной записи данных
4. Возвращение подтверждения клиенту



STRONG CONSISTENCY

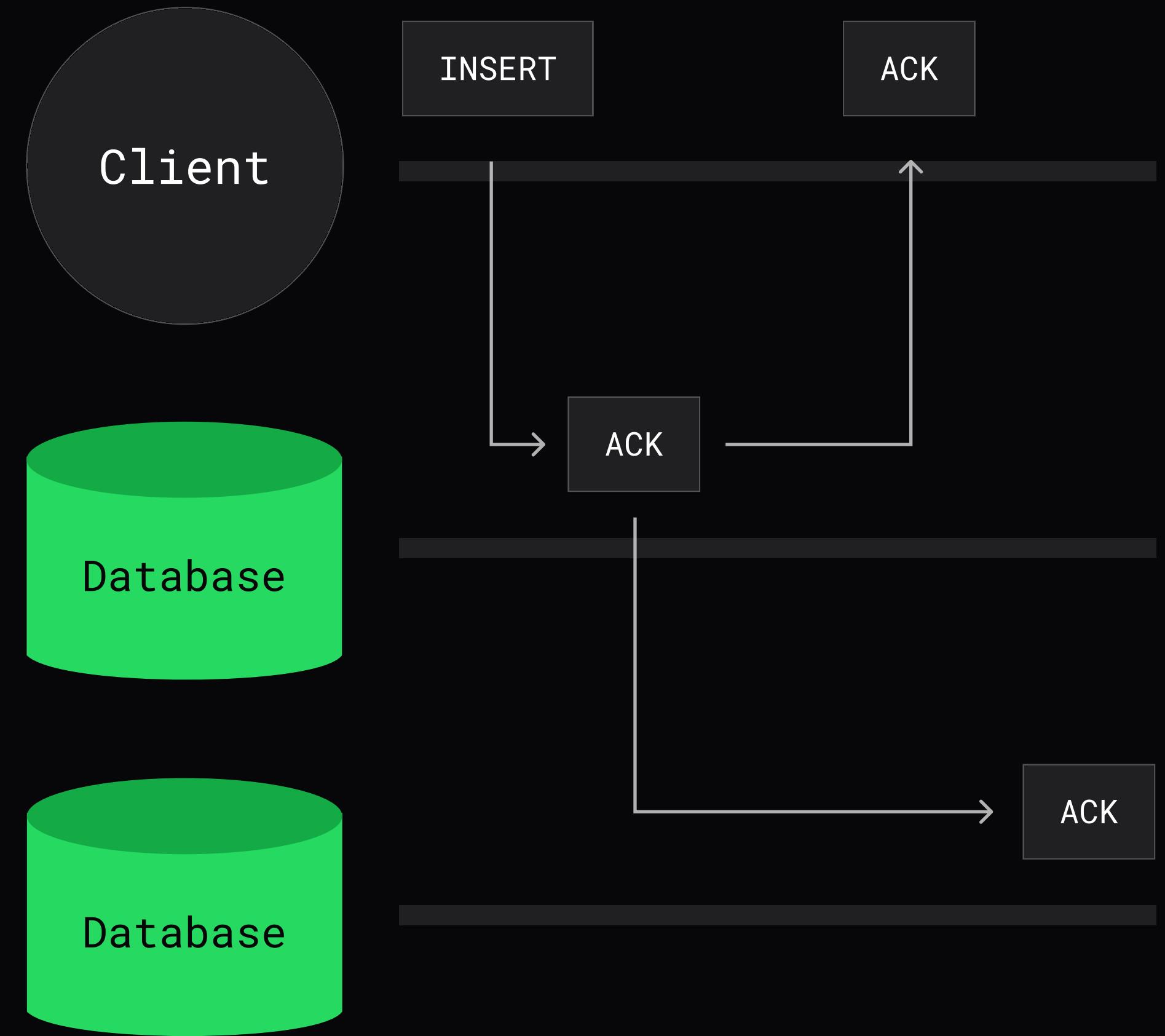
Любая операция чтения из любого узла базы данных венет последнюю операцию записи



АСИНХРОННАЯ (ASYNC)

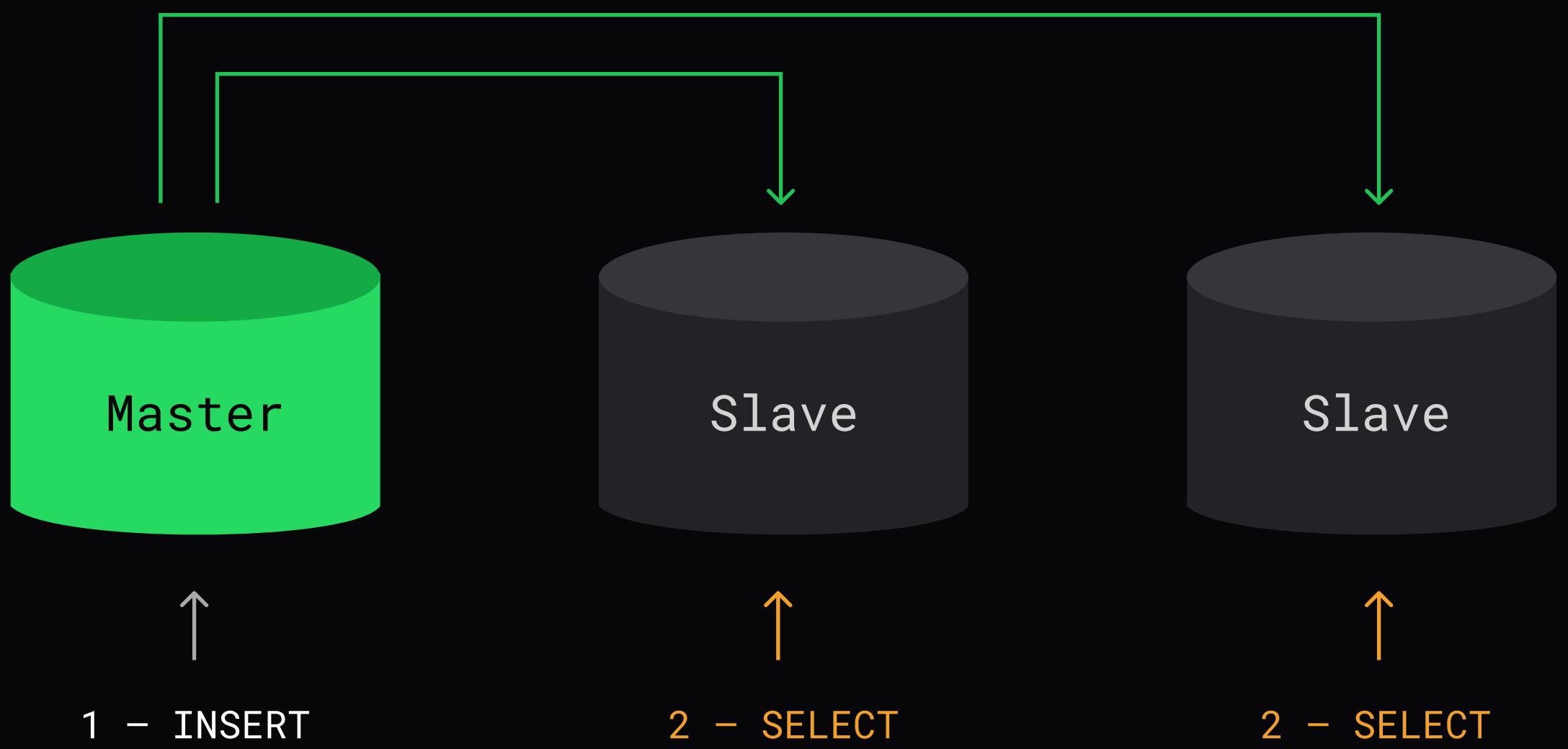
- i Local commit (данные доступны только локально)

1. Локальная запись данных
2. Возвращение подтверждения клиенту
3. Отправка данных на реплику

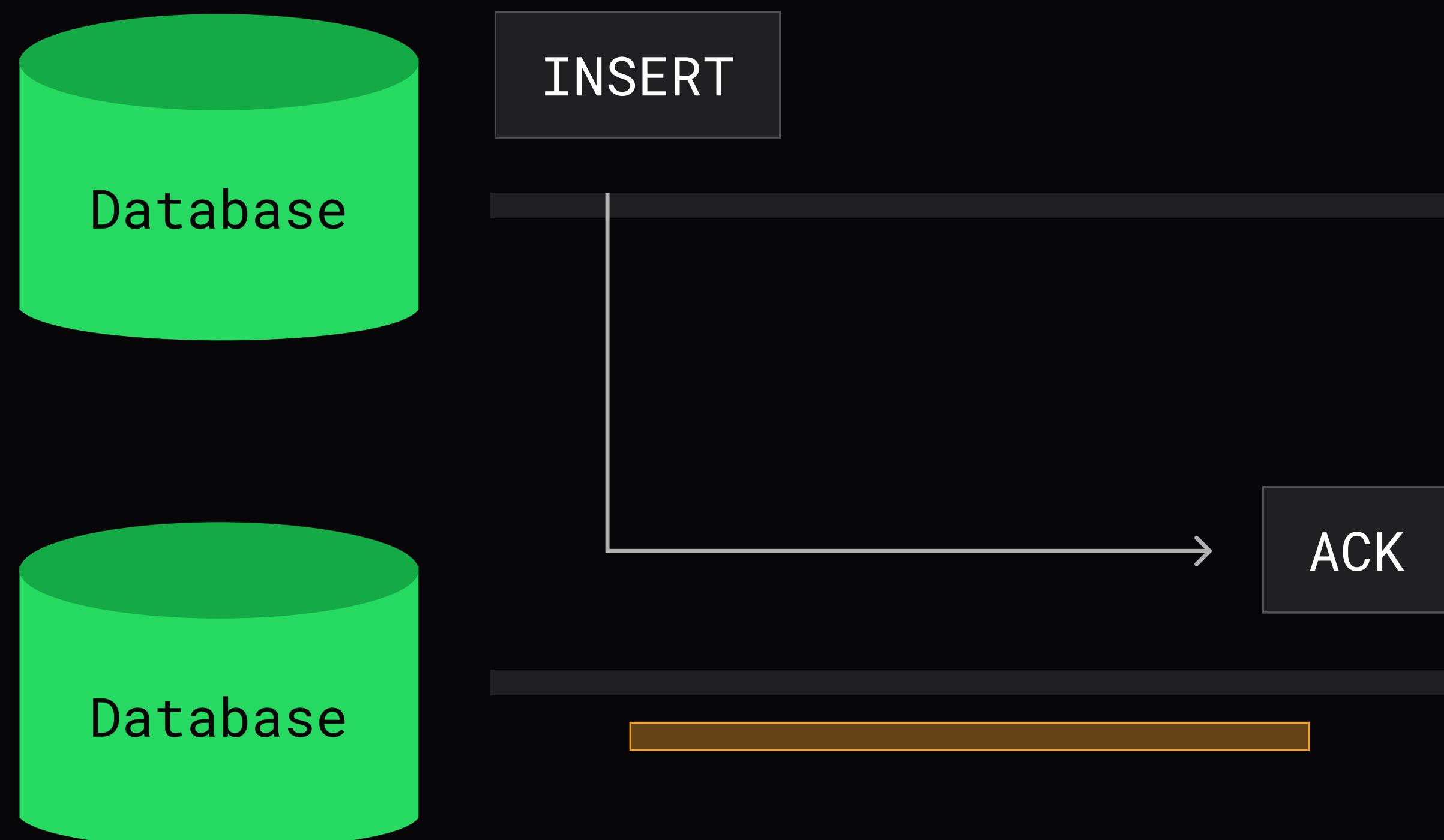


EVENTUAL CONSISTENCY

В отсутствии изменений данных, через какой-то промежуток времени после последнего обновления («в конечном счёте») все запросы будут возвращать последнее обновлённое значение



REPLICATION LAG



Если мастер выходит из строя, а часть данных
не доехала до слейвов – вы теряете эти данные

СИНХРОННАЯ

Про согласованность данных

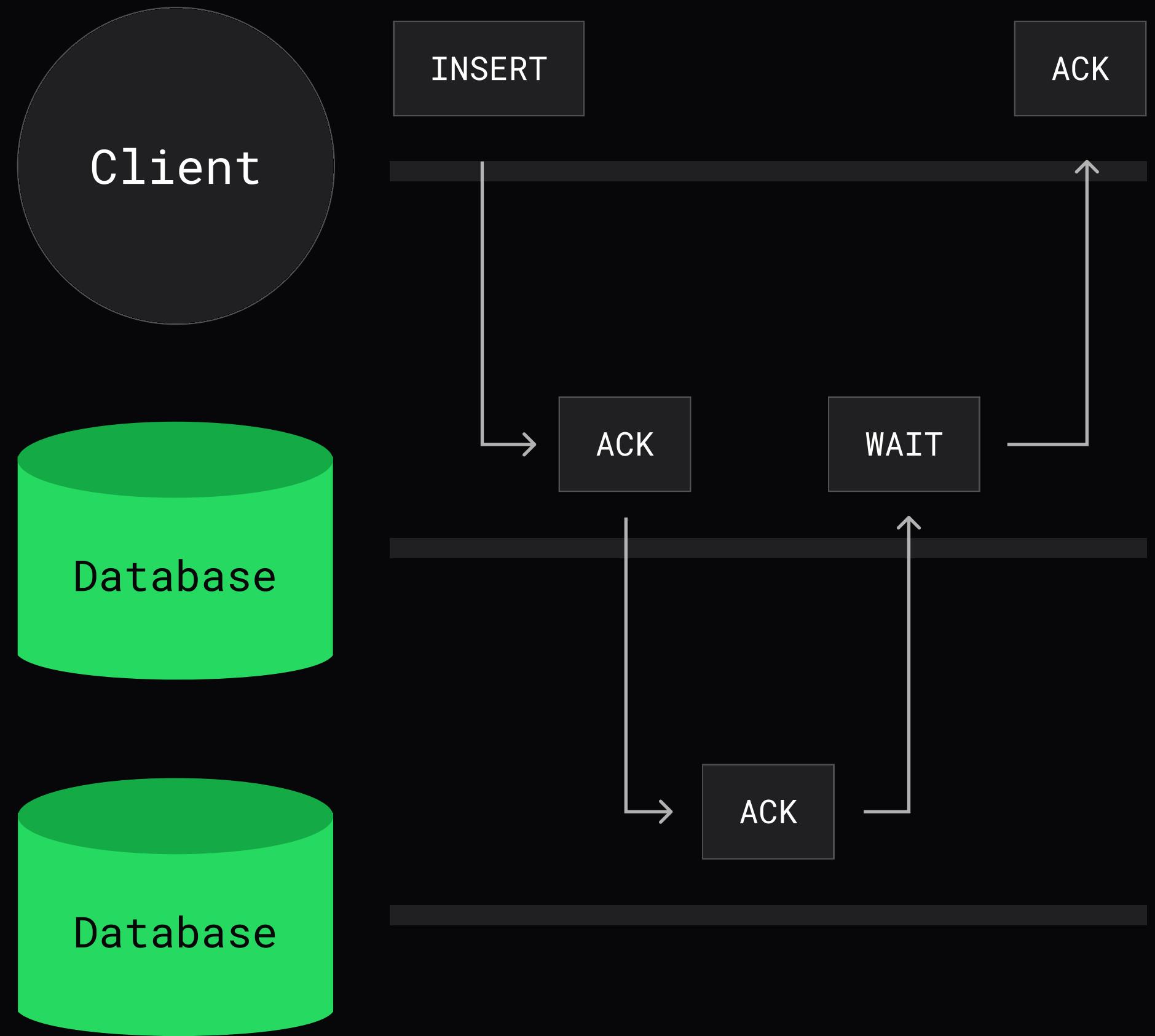
АСИНХРОННАЯ

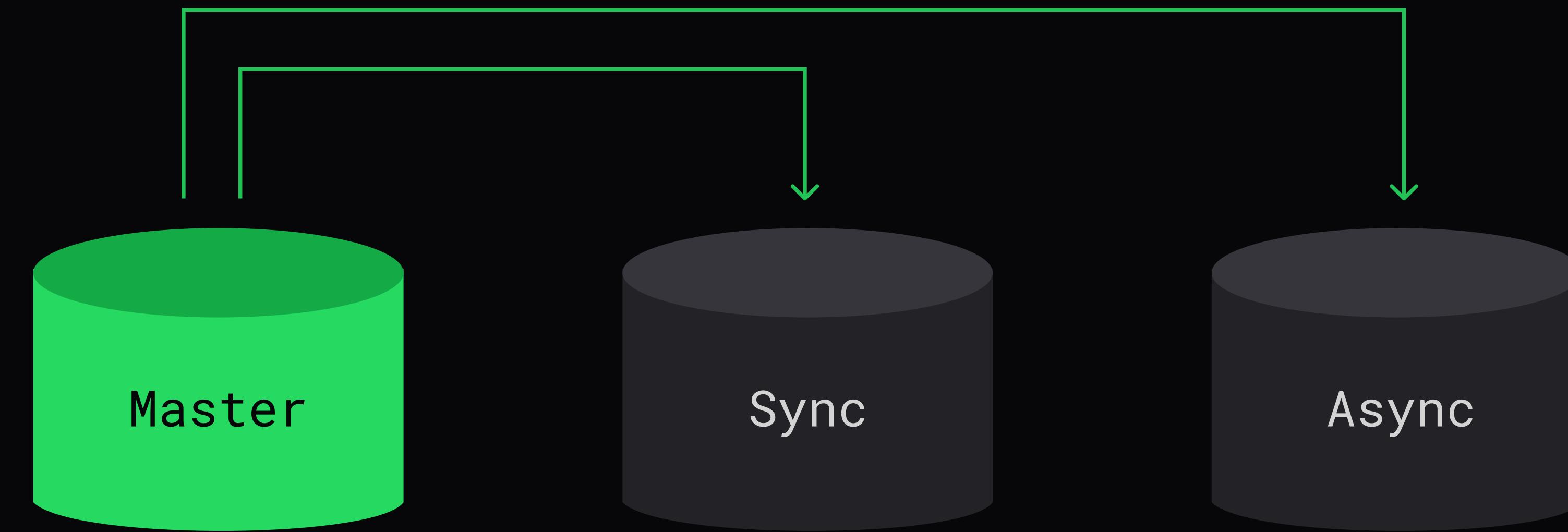
Про скорость работы

ПОЛУСИНХРОННАЯ (SEMISYNC)

- i Local commit + remote ack
(данные доступны только локально,
но уже скопированы везде)

1. Локальная запись данных
2. Отправка данных на реплику
3. Получение подтверждения от реплики
о получении изменений
4. Возвращение подтверждения клиенту

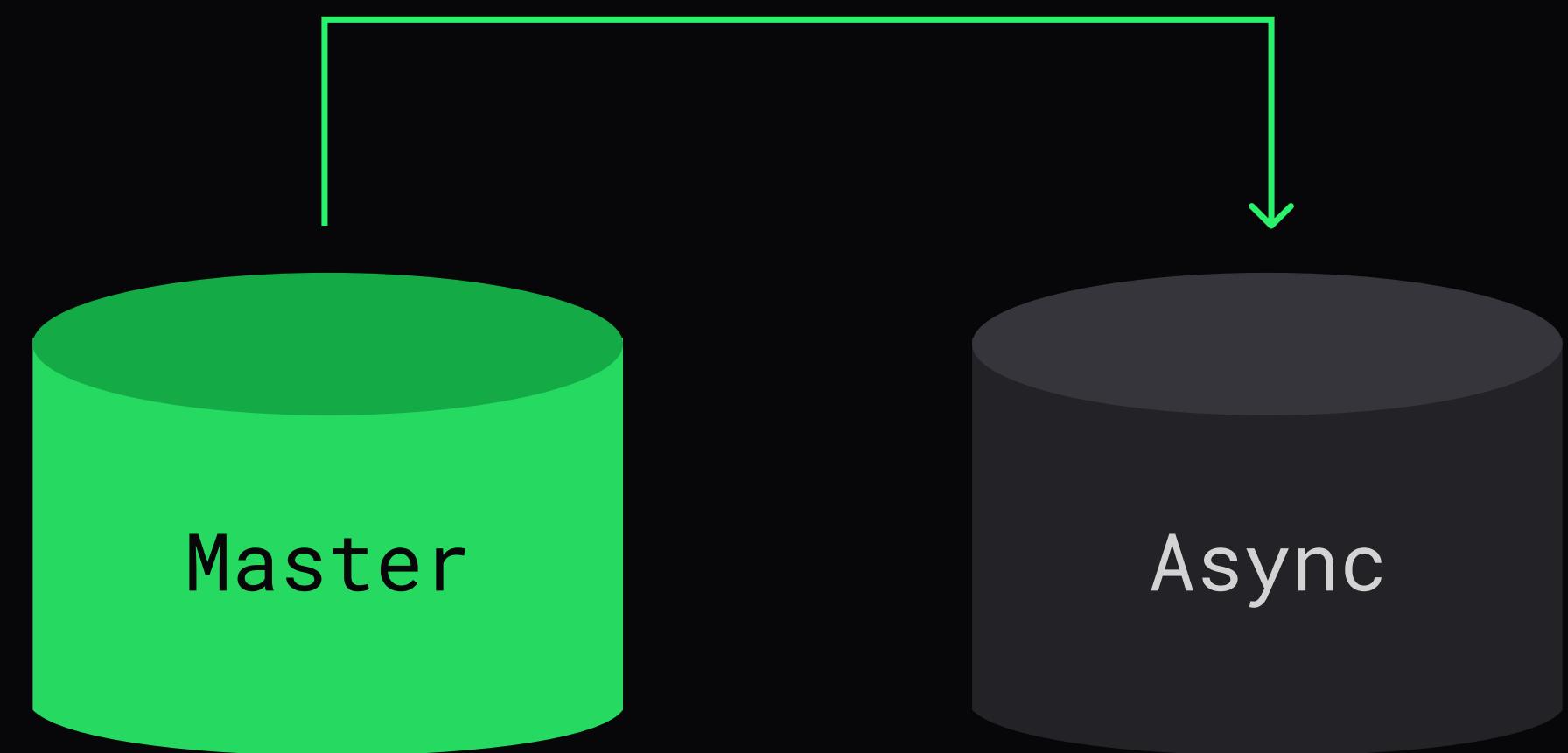




ПРИМЕРЫ

ПРЕДСТАВИМ, ЧТО ВЫ СОЗДАЕТЕ КЛОН TWITTER

Пользователь может опубликовать твит со своего компьютера, который отправит запрос на запись в вашу распределенную базу данных с асинхронной репликацией

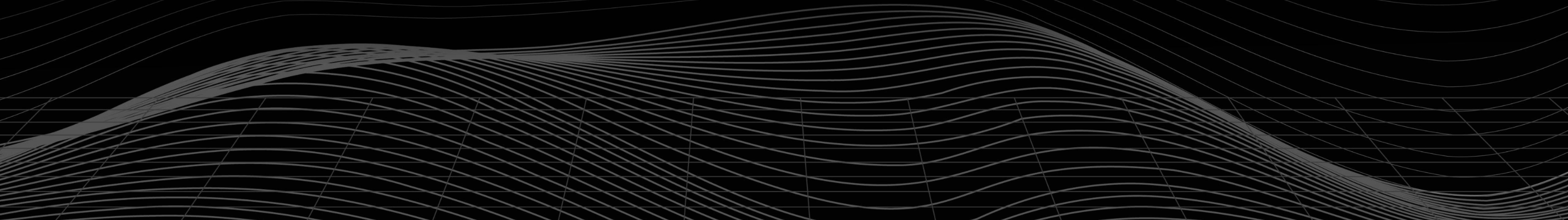


ПОЛЬЗОВАТЕЛЬ МОЖЕТ
НЕ УВИДЕТЬ СВОЙ СОЗДАННЫЙ ТВИТ
– ВОЗМОЖНО МОРГАНИЕ

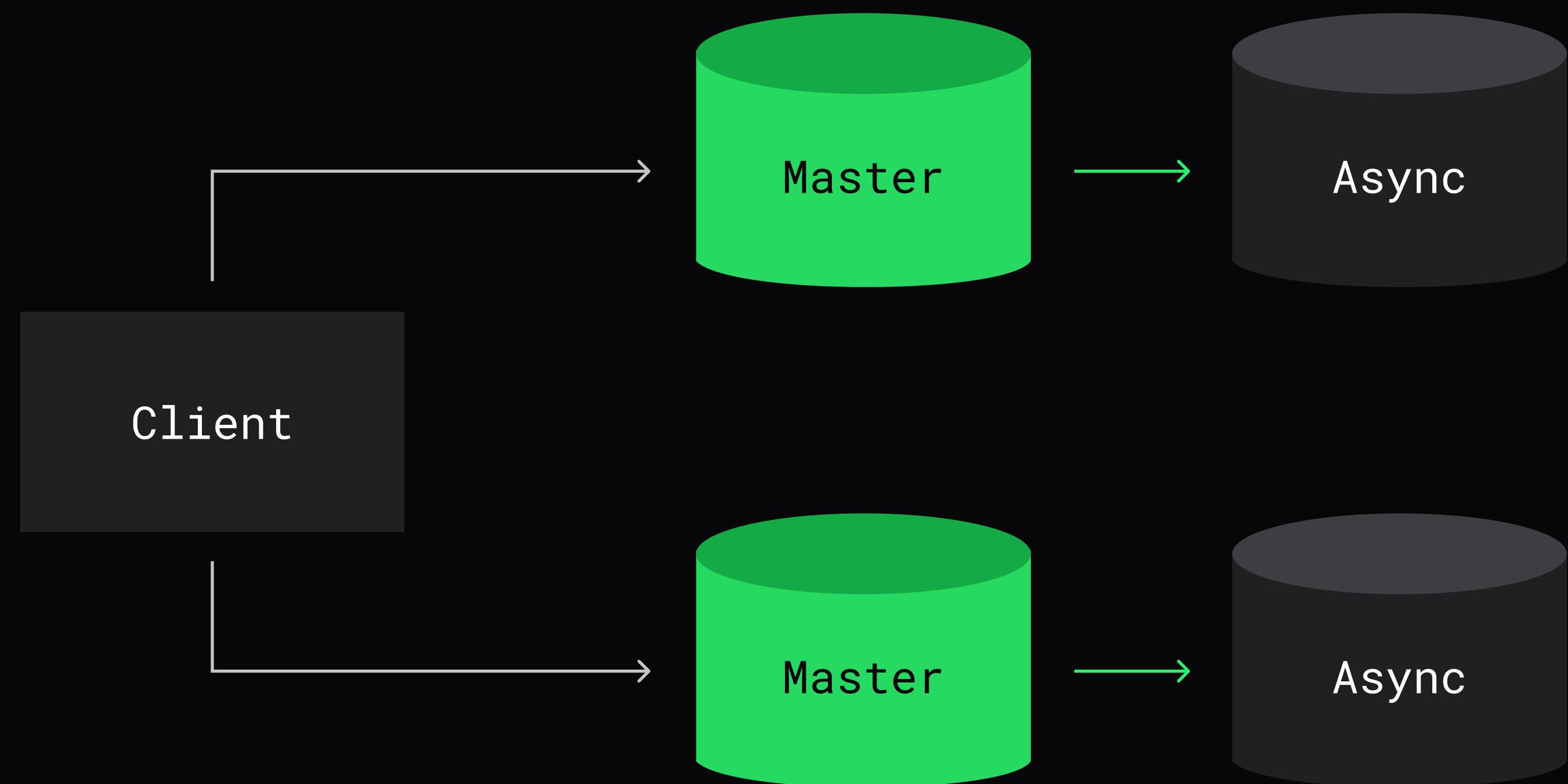
ЧТЕНИЕ СОБСТВЕННЫХ ЗАПИСЕЙ

Можно отследить, когда пользователь в последний раз отправлял обновление. Если он отправил обновление, например в течение последней минуты, то запросы на чтение должен обрабатываться главным узлом

СОЗДАННЫЙ ПОЛЬЗОВАТЕЛЕМ
ТВИТ МОЖЕТ ПОТЕРЯТЬСЯ

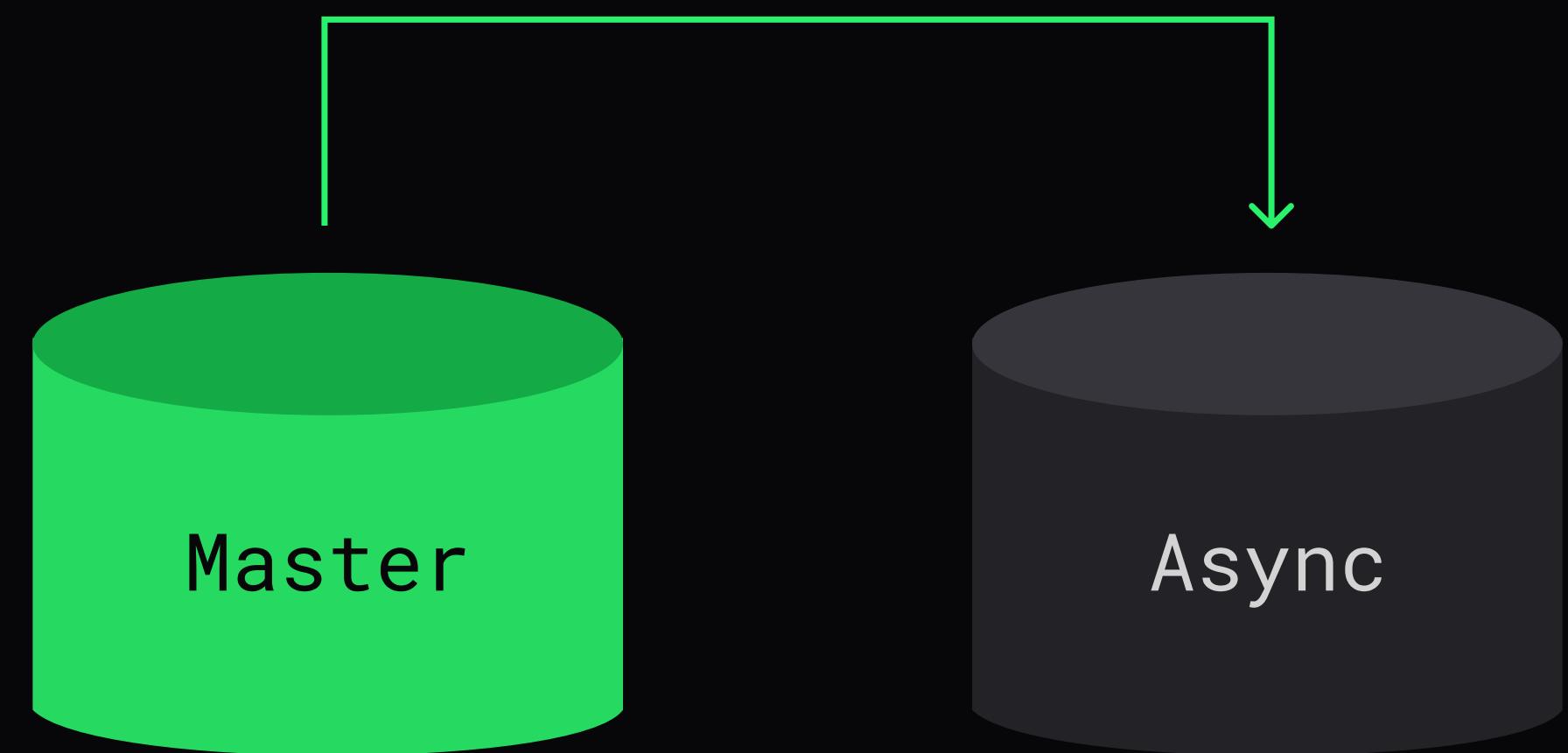


ЗАПИСЬ В ДВЕ РЕПЛИКИ



ПРЕДСТАВИМ, ЧТО ВЫ СОЗДАЕТЕ КЛОН TWITTER

Пользователь может просматривать ленты твитов других пользователей со своего компьютера, который отправит запрос на чтение в вашу распределенную базу данных с асинхронной репликацией



ПОЛЬЗОВАТЕЛЬ МОЖЕТ
НЕ УВИДЕТЬ СОЗДАННЫЙ ТВИТ
ДРУГОГО ПОЛЬЗОВАТЕЛЯ
– ВОЗМОЖНО МОРГАНИЕ

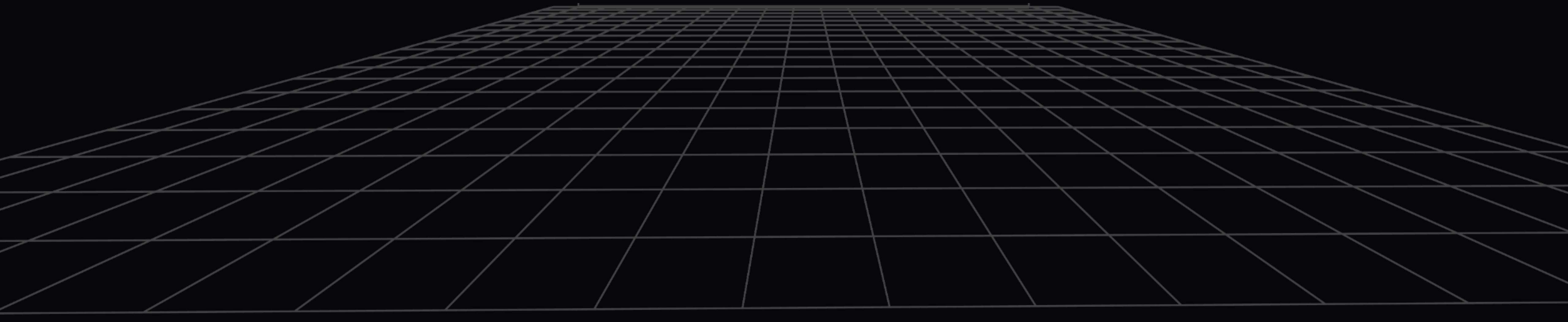
МОНОТОННОЕ ЧТЕНИЕ

Можно обеспечить, чтобы каждый пользователь всегда читал из одной и той же реплики (разные пользователи могут читать с разных реплик)

FAQ

Виды репликации

sync, async, semisync



подтема №3

ВИДЫ РЕПЛИКАЦИИ

Кто ответственный за рассылку изменений...

Terminal: System Design × + ✓



push – мастер рассыпает данные репликам (PgSQL)

pull – реплики стягивают данные сами (MySQL)

подтема №4

ВИДЫ РЕПЛИКАЦИИ

Как передавать данные . . .

ЛОГИЧЕСКАЯ

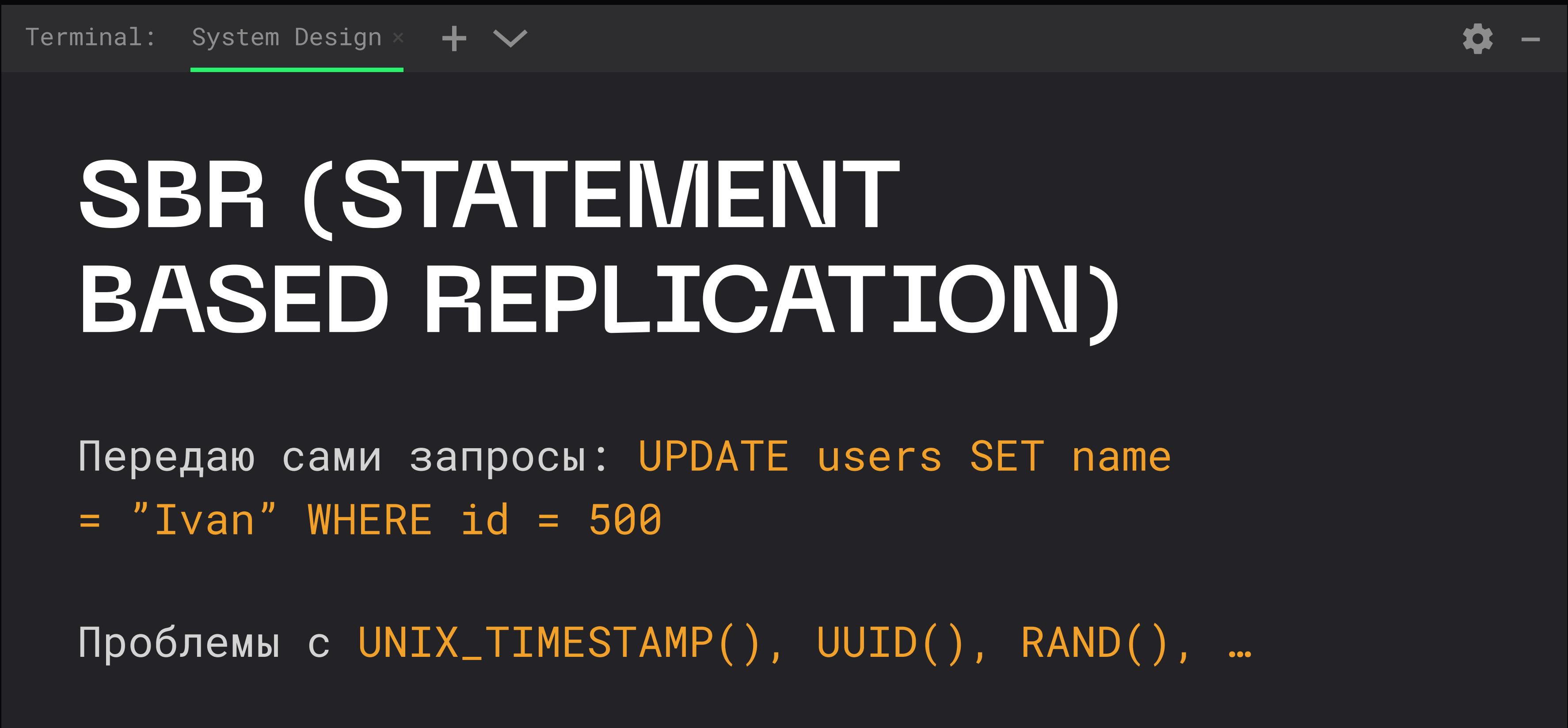
Копируем записи

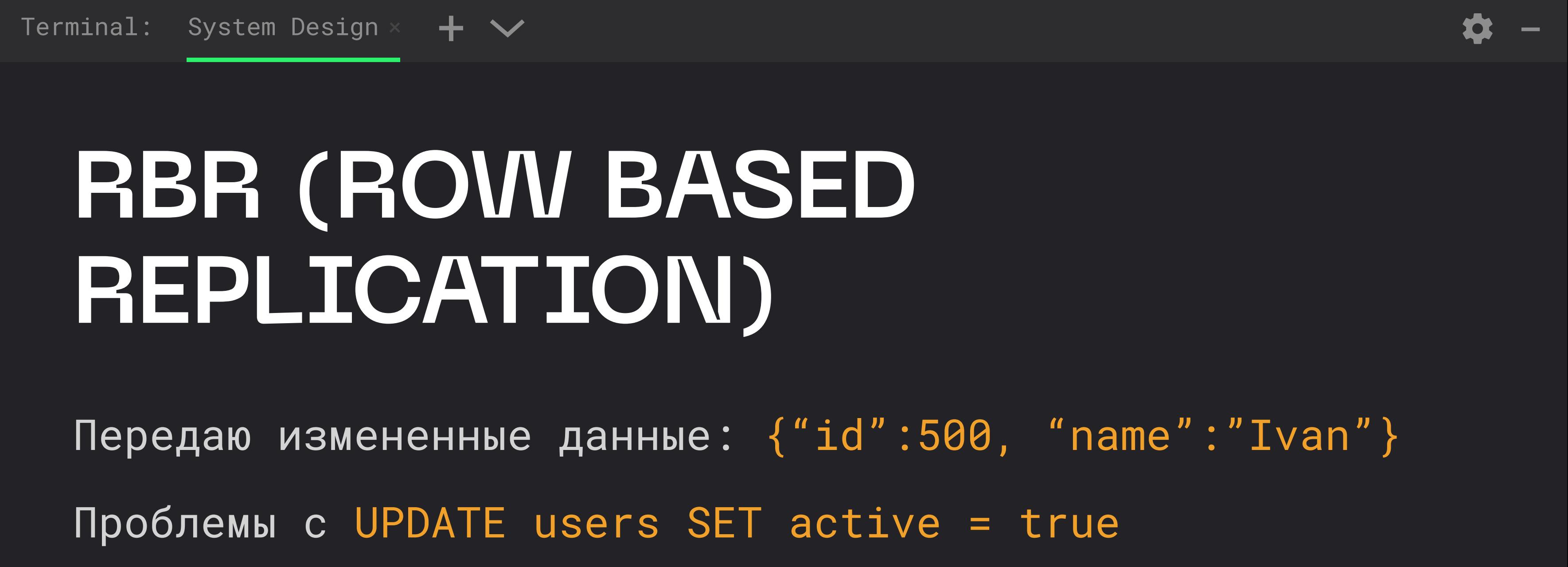
ФИЗИЧЕСКАЯ

Копируем файлы

SBR И RBR

Логическая репликация
разделяется на SBR и RBR





ЕСТЬ ЕЩЕ MIXED

ФИЛЬТРАЦИЯ РЕПЛИКАЦИЙ

Можно реплицировать данные частично

FAQ

Репликация

Синхронная, асинхронная, полусинхронная, master slave,
master master, master less, способы передачи данных

CAP TEOPEMA

Terminal: System Design × + ▾

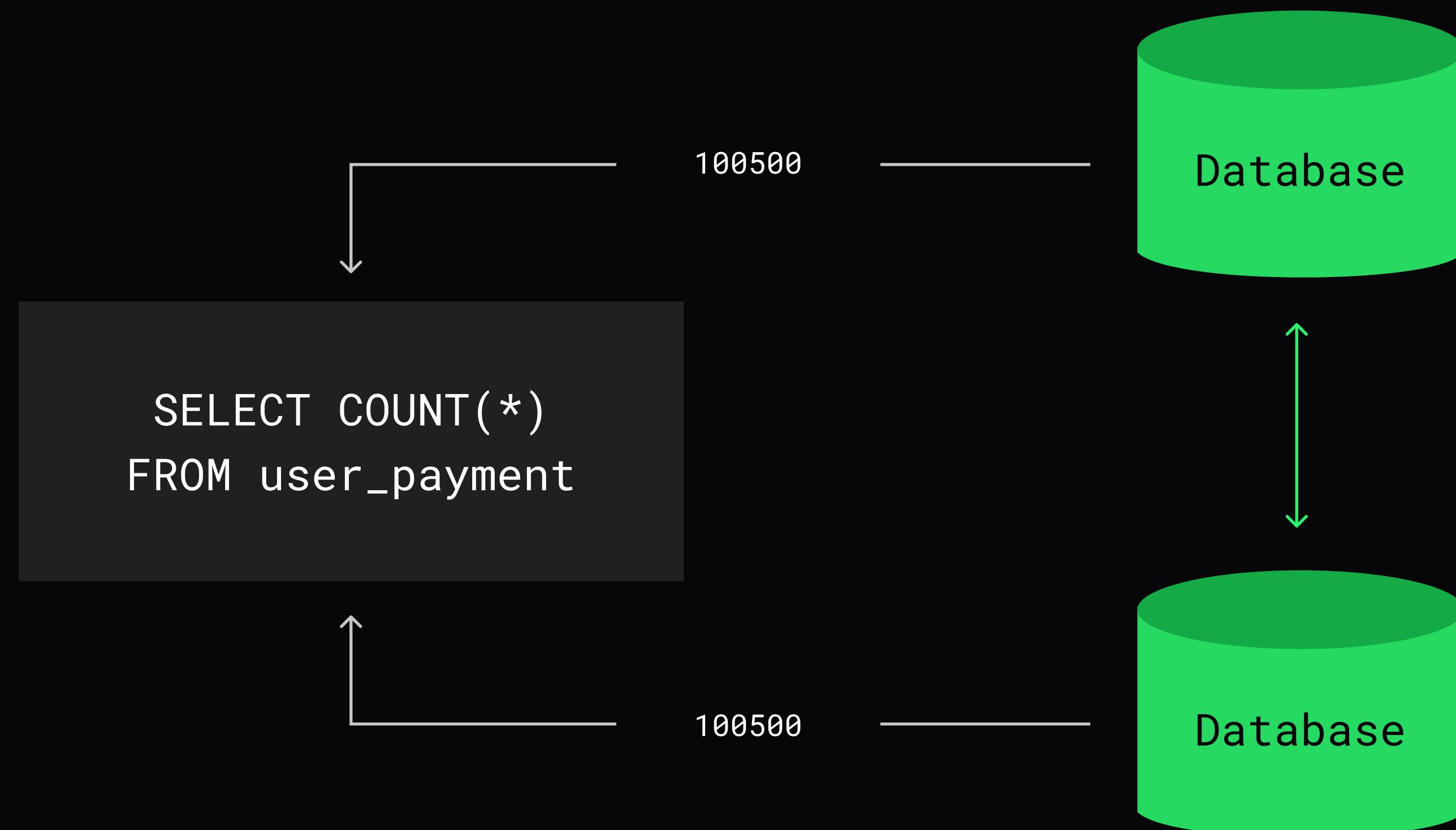


CAP ТЕОРЕМА

Утверждение о том, что в любой реализации распределенных вычислений возможно обеспечить **не более двух из трёх следующих свойств** (согласованность, доступность, устойчивость к разделению)

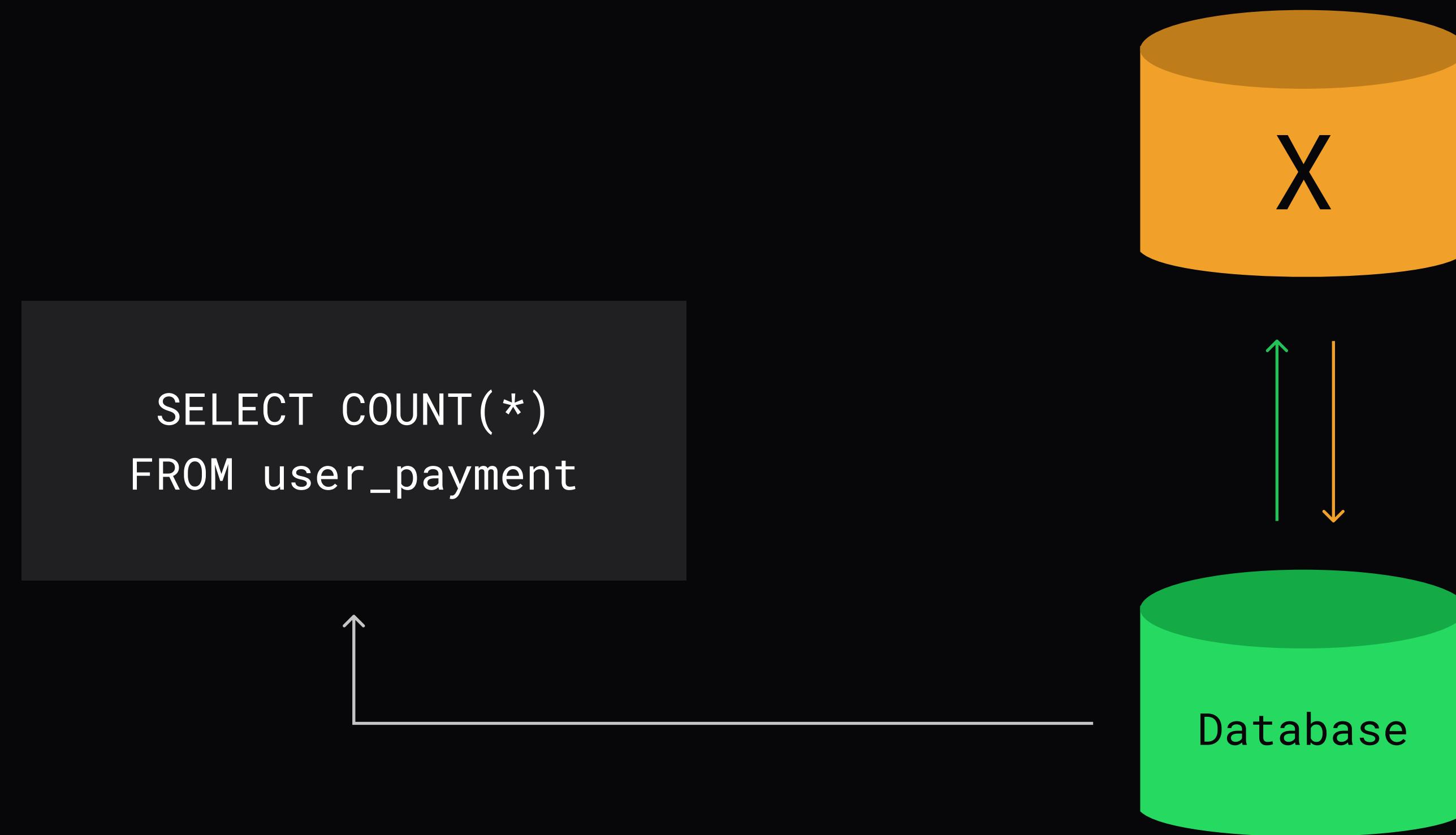
КОНСИСТЕНТНОСТЬ

Данные во всех нодах одинаковы



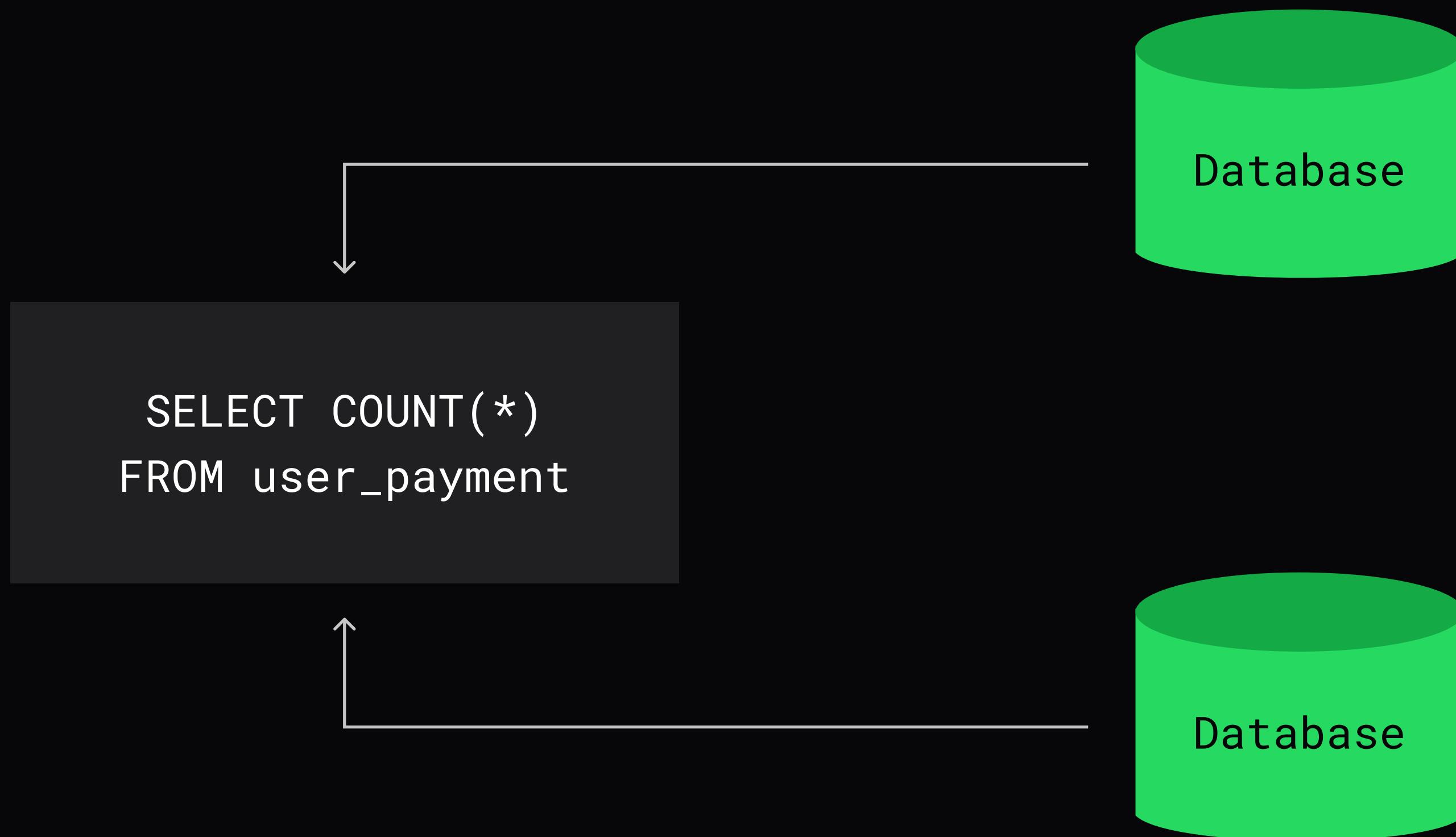
доступность

Если запрос пришел на живую ноду,
запрос будет получен за конечное время



УСТОЙЧИВОСТЬ К РАЗДЕЛЕНИЮ

Продолжаем работать, несмотря на отсутствие связи



ОБРУБАЕМ СВЯЗЬ

1. Наш мир не идеален, поэтому Р должна быть – АС система
2. Разрешаем из нод читать, но запрещаем писать – СР система
3. Разрешаем читать и писать – АР система

Terminal: System Design × + ▾



PACELC — ЭТО...

Расширение CAP теоремы, которое гласит, что в случае разделения сети (Р) в распределенной системе необходимо выбирать между доступностью (A) и согласованностью (C) (согласно теореме CAP), но в любом случае, даже если система работает normally в отсутствии разделения (E), нужно выбирать между задержками (L) и согласованностью (C)

FAQ

CAP теорема

Консистентность, доступность,
устойчивость к разделению

ПАРТИЦИОНИРОВАНИЕ

Terminal: System Design × + ▾



ПАРТИЦИОНИРОВАНИЕ

Метод разделения больших таблиц на много маленьких, и желательно, чтобы это происходило прозрачным для приложения способом



Секции находятся на одном и том же инстансе базы данных



СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1. Ограничения на размер таблицы, индекса, количества строк (когда-то в PostgreSQL был 32ТБ максимальный размер таблицы)
2. Распределение нагрузки (партиции на разных дисках)
3. Охлаждение данных (одни партиции на SSD, а другие на HDD)
4. Удаление данных (партиционирование по дням)

подтема №1

ВИДЫ ПАРТИЦИОНИРОВАНИЯ

Как разделять таблицы . . .

ВЕРТИКАЛЬНОЕ

ID	Name	Status	Description	Photo
10				
20				
30				
50				

ID	Name	Status
10		
20		
30		
50		

ID	Description	Photo
10		
20		
30		
50		

ГОРИЗОНТАЛЬНОЕ

ID	Name	Status	Description	Photo
10				
20				
30				
50				

ID	Name	Status	Description	Photo
10				
20				

ID	Name	Status	Description	Photo
30				
50				

подтема №2

СПОСОБЫ ПАРТИЦИОНИРОВАНИЯ

Как распределять данные между партициями . . .

RANGE BASED

ID Price

10	30
20	57
30	64
50	123

Partition #1 (0...50)

10	30
----	----

Partition #2 (50...100)

20	57
20	57

Partition #3 (100+)

50	123
----	-----

ID Data

10	37.2
20	37.1
30	36.8

Partition #1

10	37.2
20	36.8

KEY BASED

ID Value

10	1
20	2
30	1

Partition #2

10	30
----	----

Hash
Function



Zone Data

101	37.2
201	37.1
303	36.8

Partition #1

101	37.2
-----	------

Zone Shard

101	1
202	2
303	3

Partition #2

202	37.1
-----	------

Partition #3

303	36.8
-----	------

**DIRECTORY
BASED**

FAQ

Партиционирование

Виды партиционирования,
способы партиционирования

ШАРДИРОВАНИЕ

Terminal: System Design × + ▾



ШАРДИРОВАНИЕ

Подход, предполагающий разделение таблиц
на независимые сегменты, каждый из которых
управляется отдельным инстансом базы данных



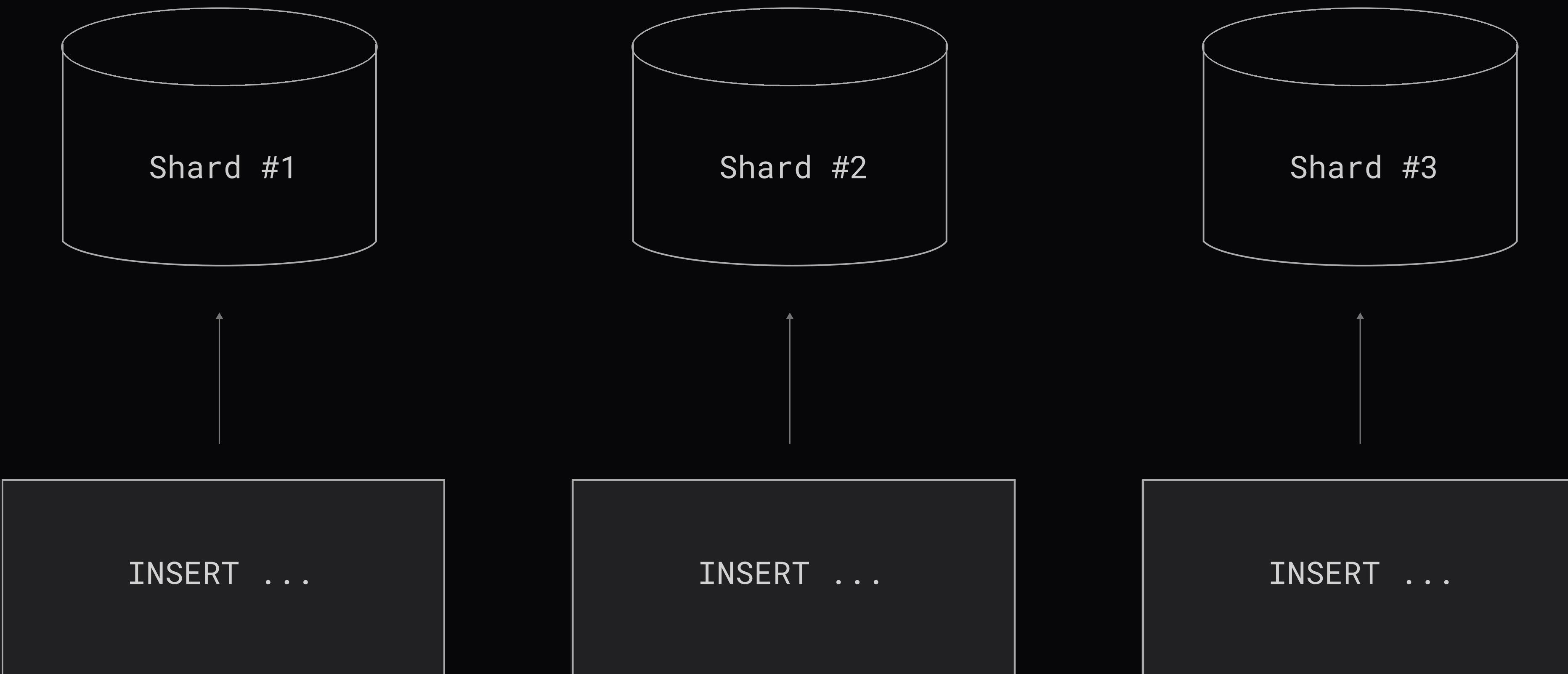
СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1. Уперлись в ресурсы одного сервера
2. Данные не помещаются на одном сервере

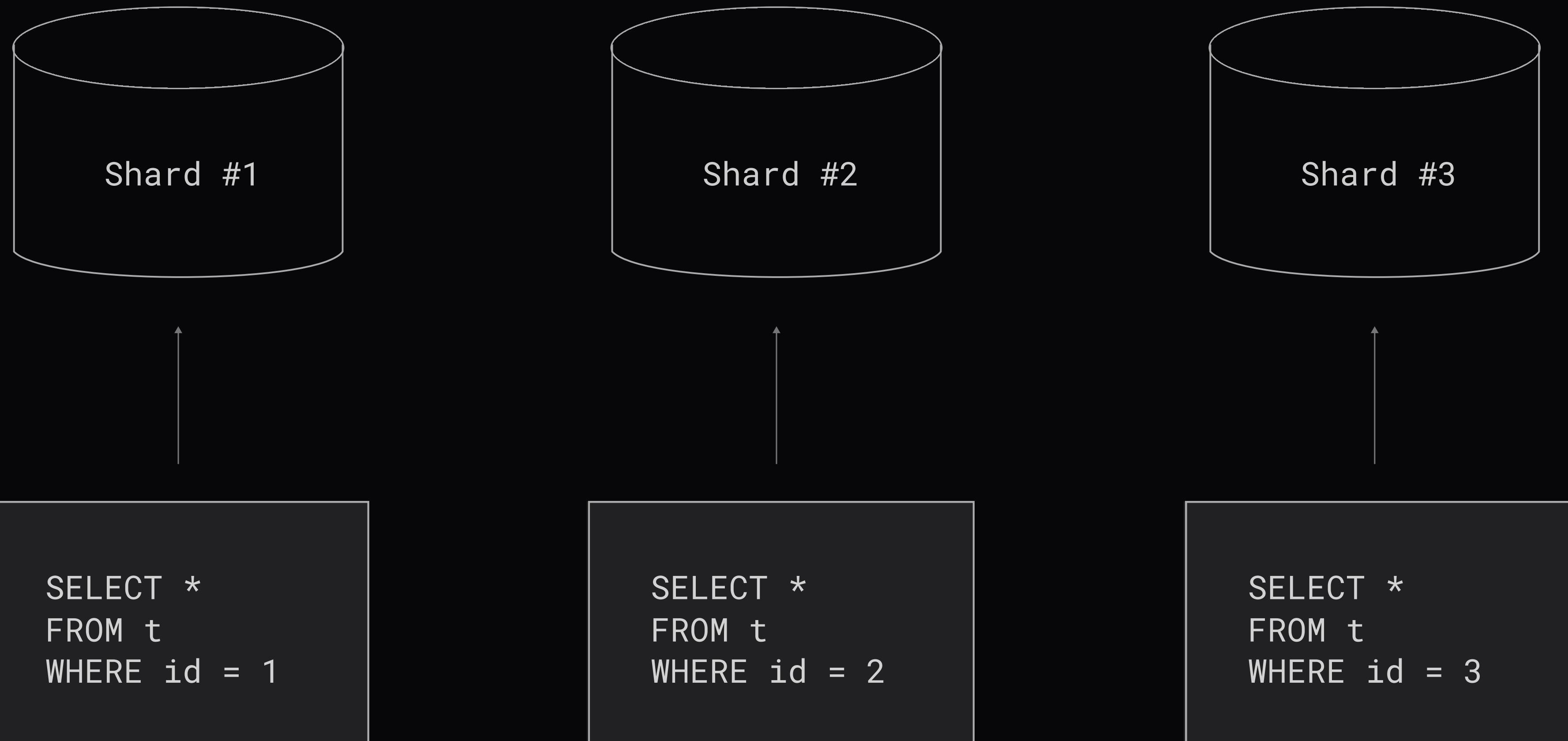
БАЗОВЫЕ ПРИНЦИПЫ

1. Данные, которые в дальнейшем потребуются вместе, должны храниться вместе
2. Специально храню данные по разным местам, чтобы параллельно их обрабатывать – МРР (Massively Parallel Processing) подход

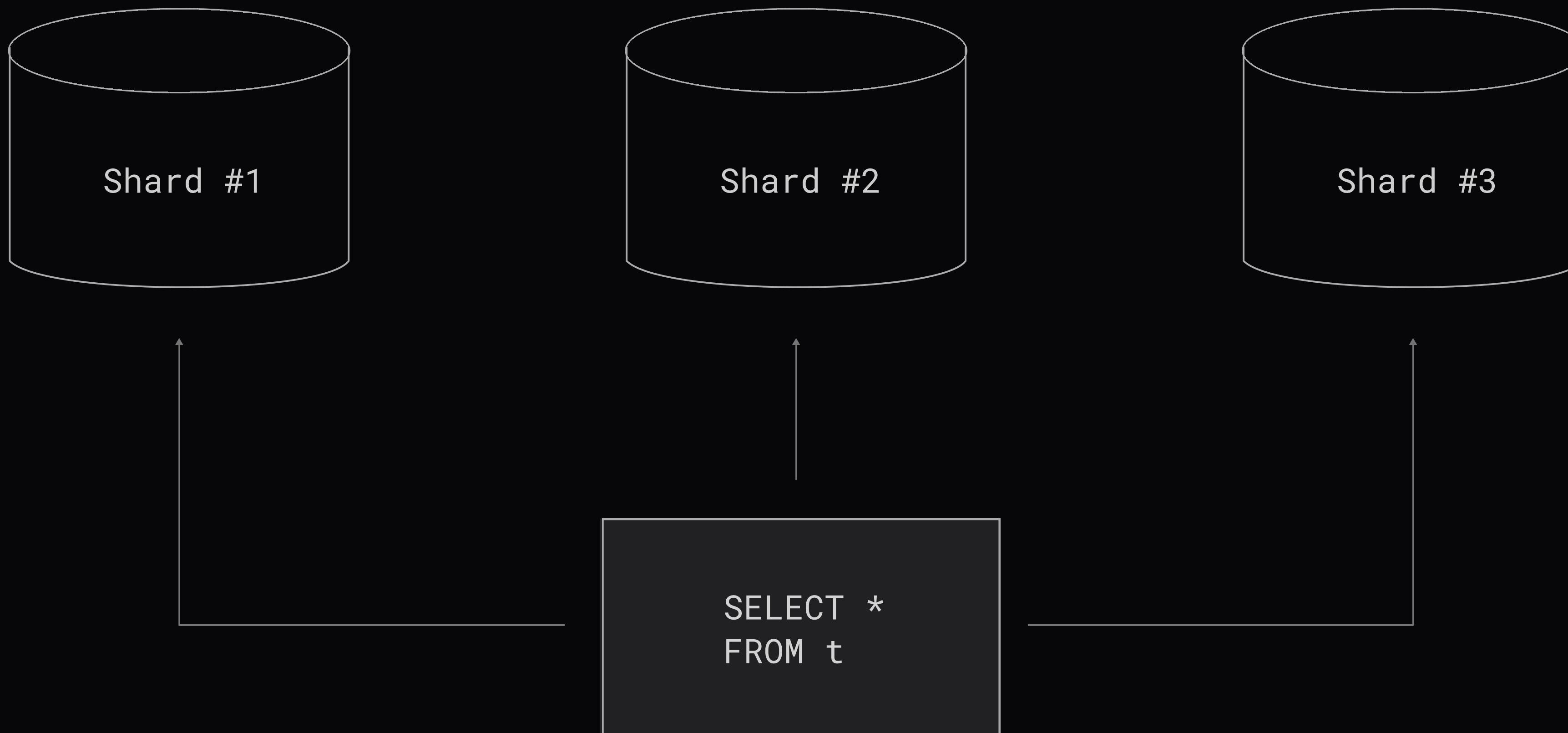
ШАРДИРОВАНИЕ



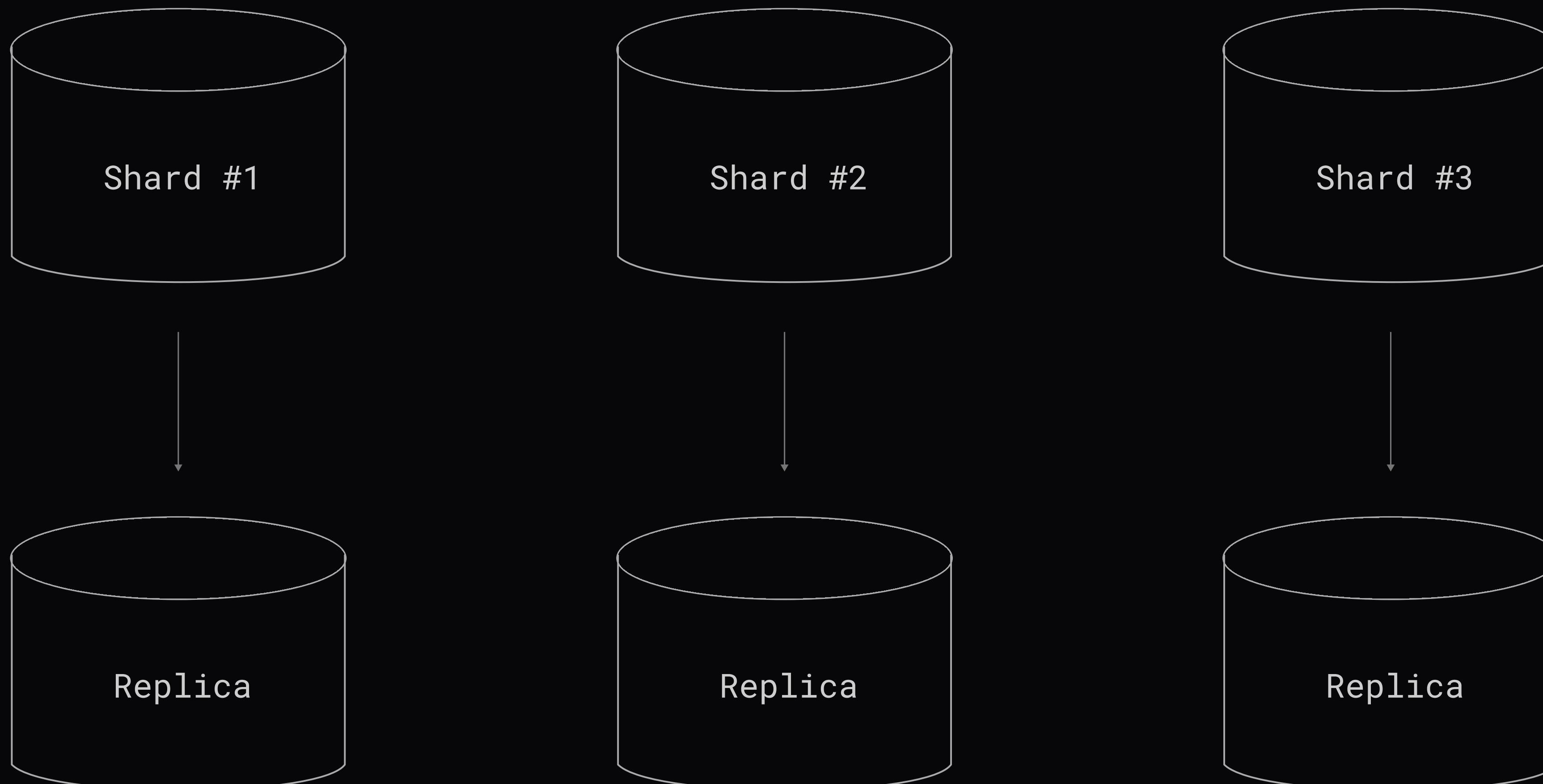
ШАРДИРОВАНИЕ



ШАРДИРОВАНИЕ



ШАРДИРОВАНИЕ С РЕПЛИКАЦИЕЙ



подтема №1

ROUTING

Как понять в какой шард идти за данными . . .

НА УРОВНЕ БАЗЫ ДАННЫХ ОЧЕНЬ ХОРОШИЙ СПОСОБ

но не все базы данных его поддерживают

КЛИЕНТСКИЙ

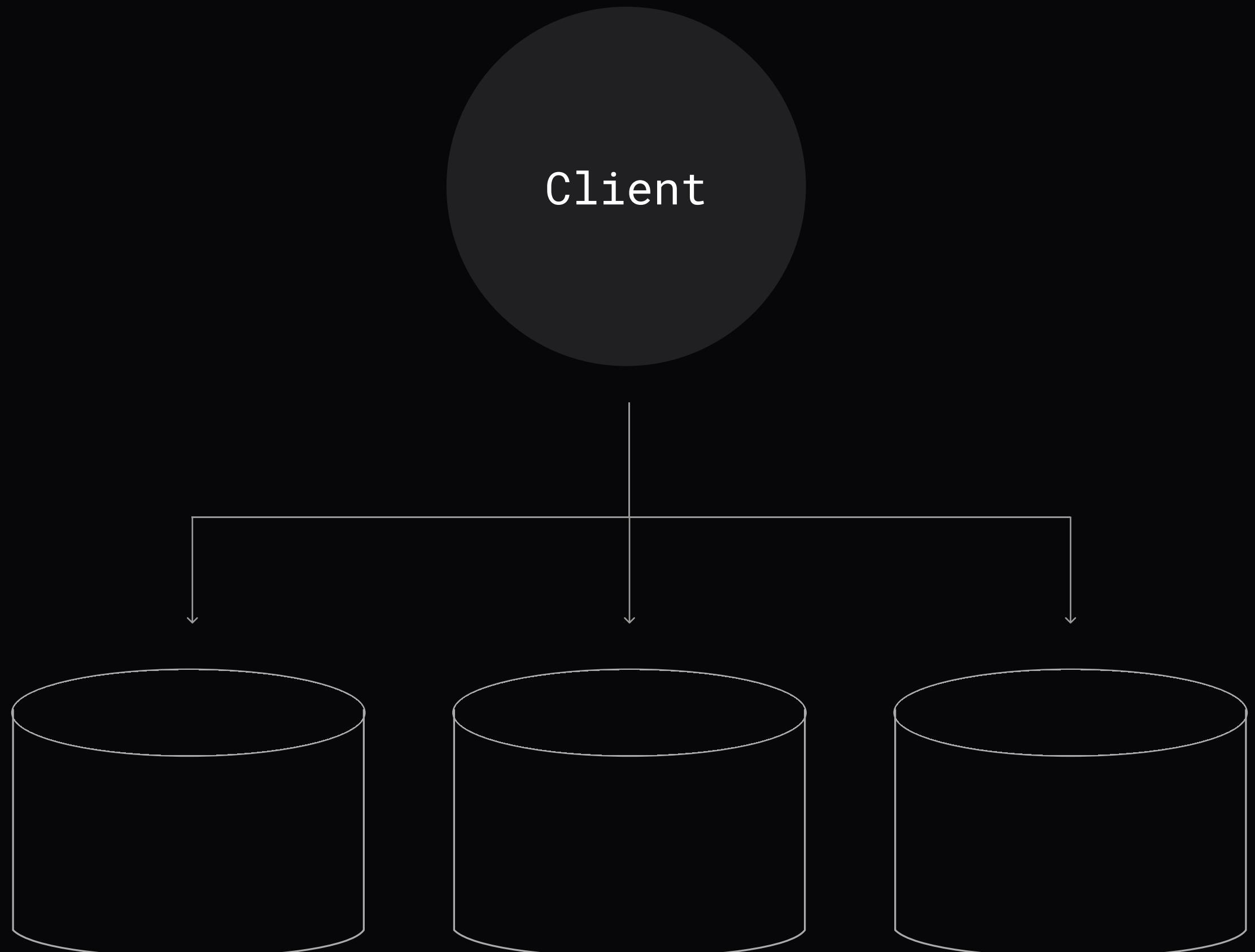
Плюсы:

- 1/ нет лишних узлов
-

Минусы:

- 1/ дополнительная логика в клиенте
-

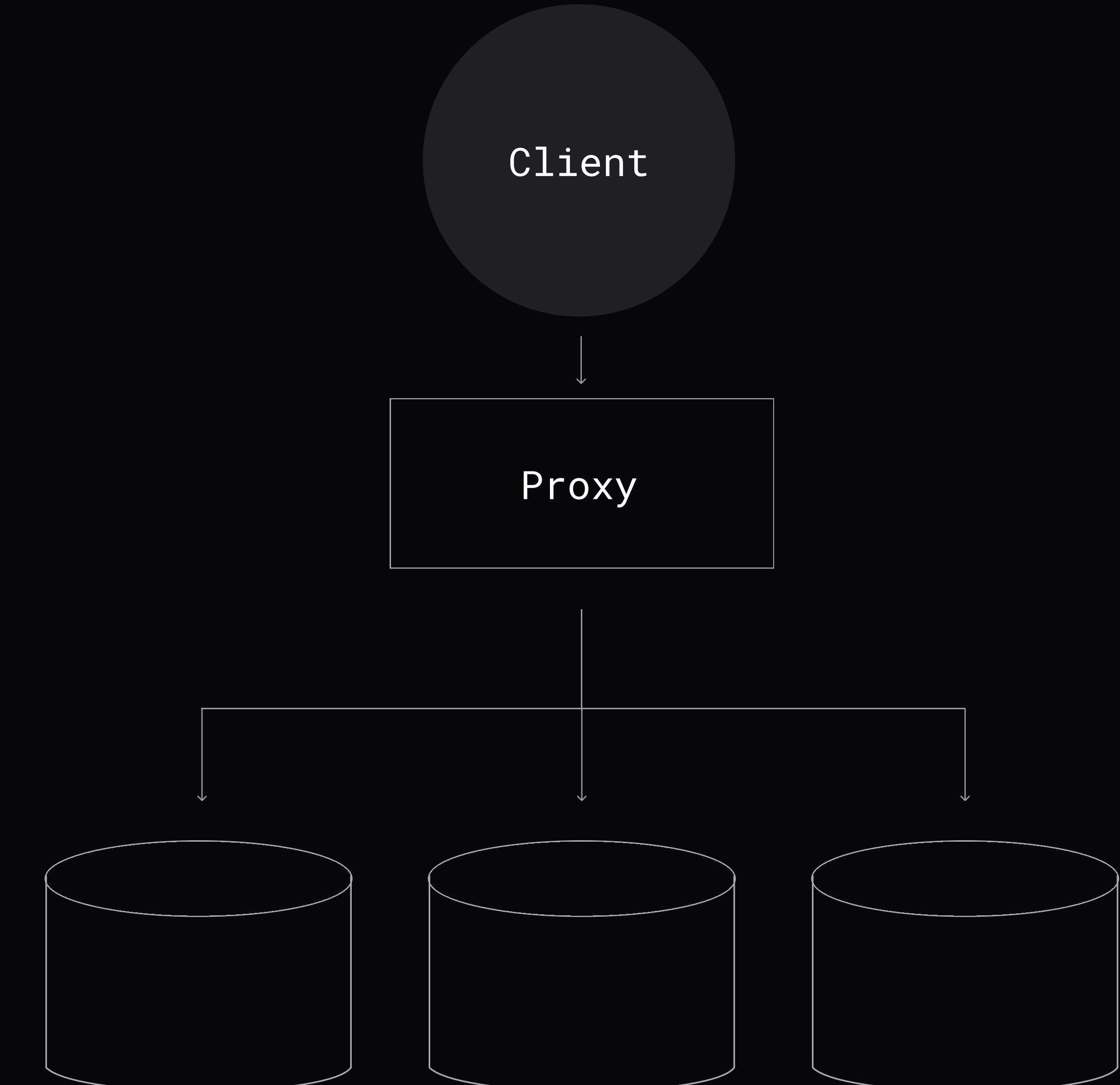
- 2/ сложности с обновлением хостов
-



PROXY

Плюсы:

- 1/ приложение не знает о шардинге



Минусы:

- 1/ дополнительный сетевой узел
- 2/ потеря функциональности
- 3/ единая точка отказа

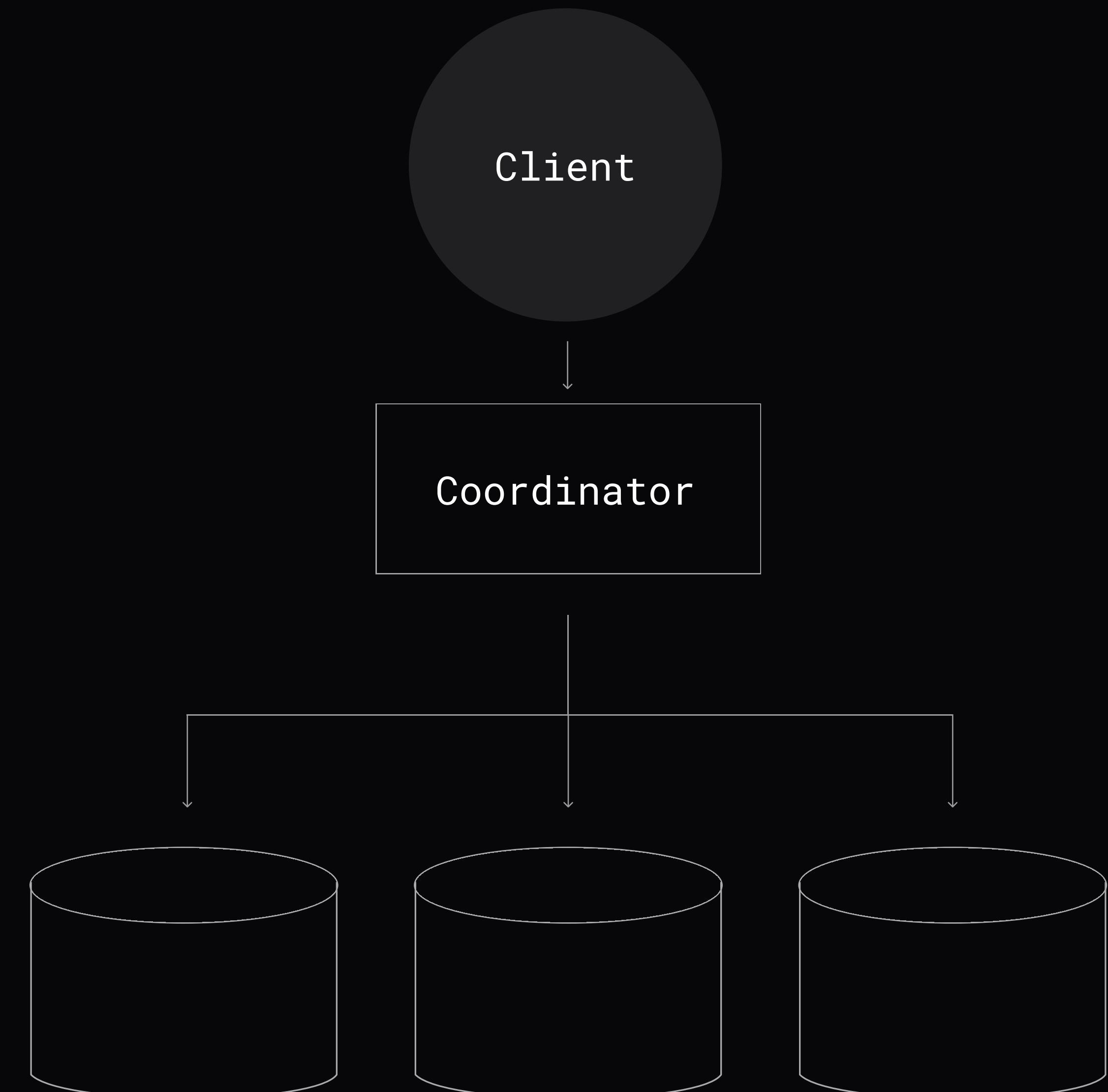
COORDINATOR

Плюсы:

- 1/ кэширование
- 2/ приложение не знает о шардинге

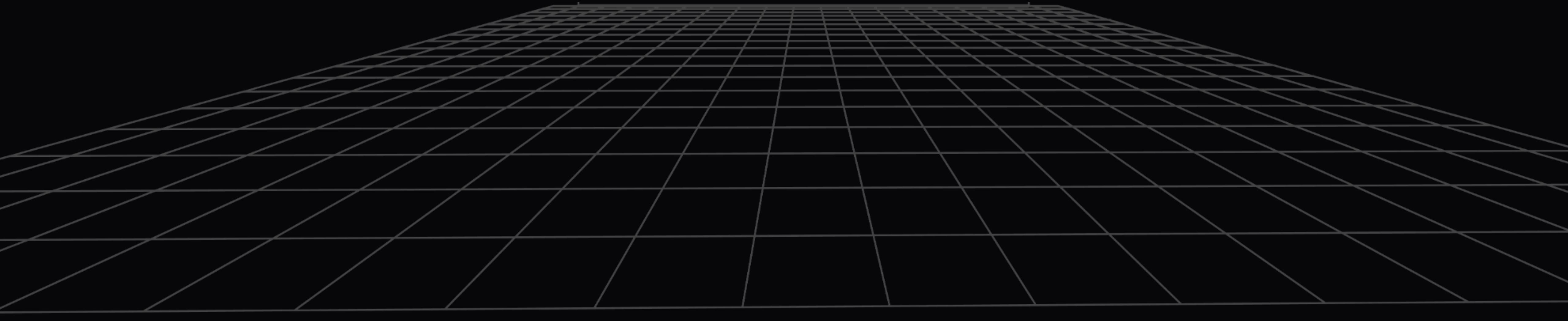
Минусы:

- 1/ дополнительный сетевой узел
- 2/ инфраструктурная сложность
- 3/ единая точка отказа
- 4/ нагрузка



FAQ

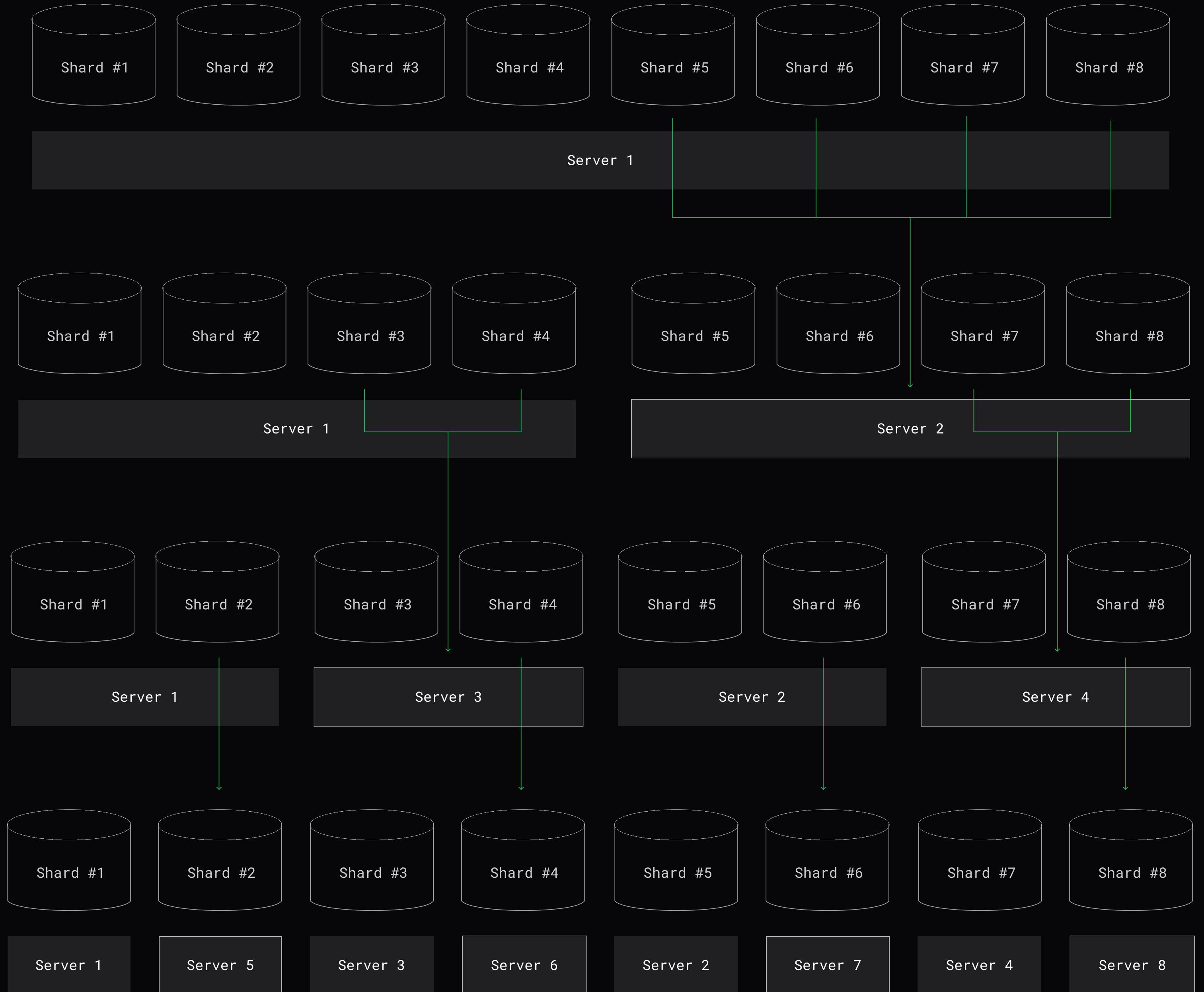
ROUTING



подтема №2

ПЕРЕБАЛАНСИРОВКА

Как перенести данные из одного шарда на другой?



ВИРТУАЛЬНЫЕ ШАРДЫ (VBUCKETS)

ПРОЦЕСС ПЕРЕБАЛАНСИРОВКИ

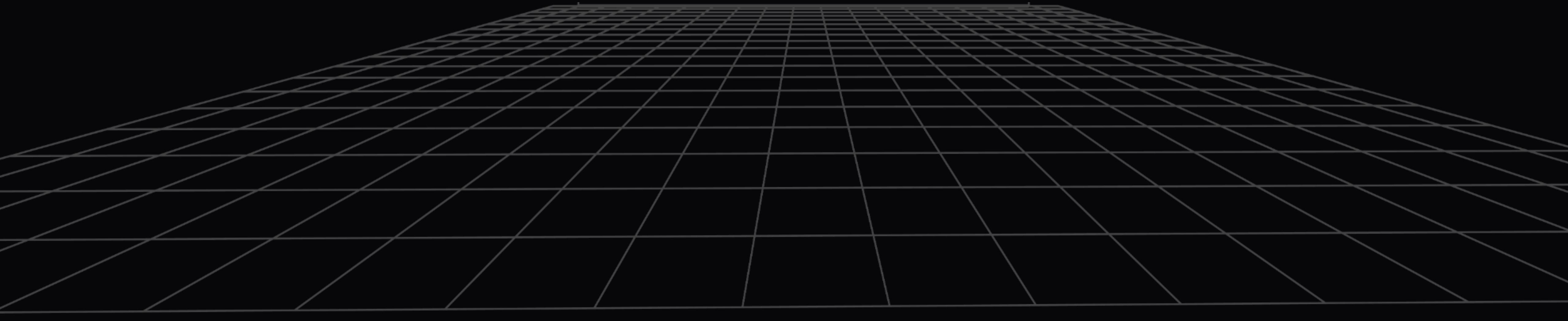
1. Только чтение
2. Все данные неизменяемые
(пишем в tgt, читаем из src и tgt)
3. Логическая репликация с src на tgt,
после синхронизации переключаемся на tgt
4. Смешанный подход

i src - откуда мигрируем данные

tgt - куда мигрируем данные

FAQ

Перебалансировка



подтема №3

RESHARDING

Terminal: System Design × + ▾



RESHARDING

- нужно добавить / удалить ноды
- исправление ошибок при выборе стратегии шардирования

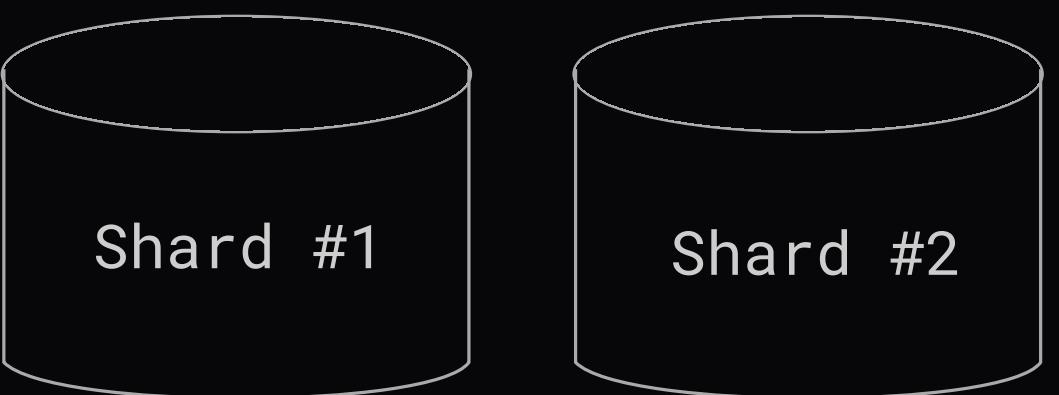
HASHING

$F(\text{key}) = \text{hash}(\text{key}) \% \text{shards_number}$



$$\begin{aligned}5 \% 3 &= 2 \\6 \% 3 &= 0 \\7 \% 3 &= 1\end{aligned}$$

$F(\text{key}) = \text{hash}(\text{key}) \% \text{shards_number}$

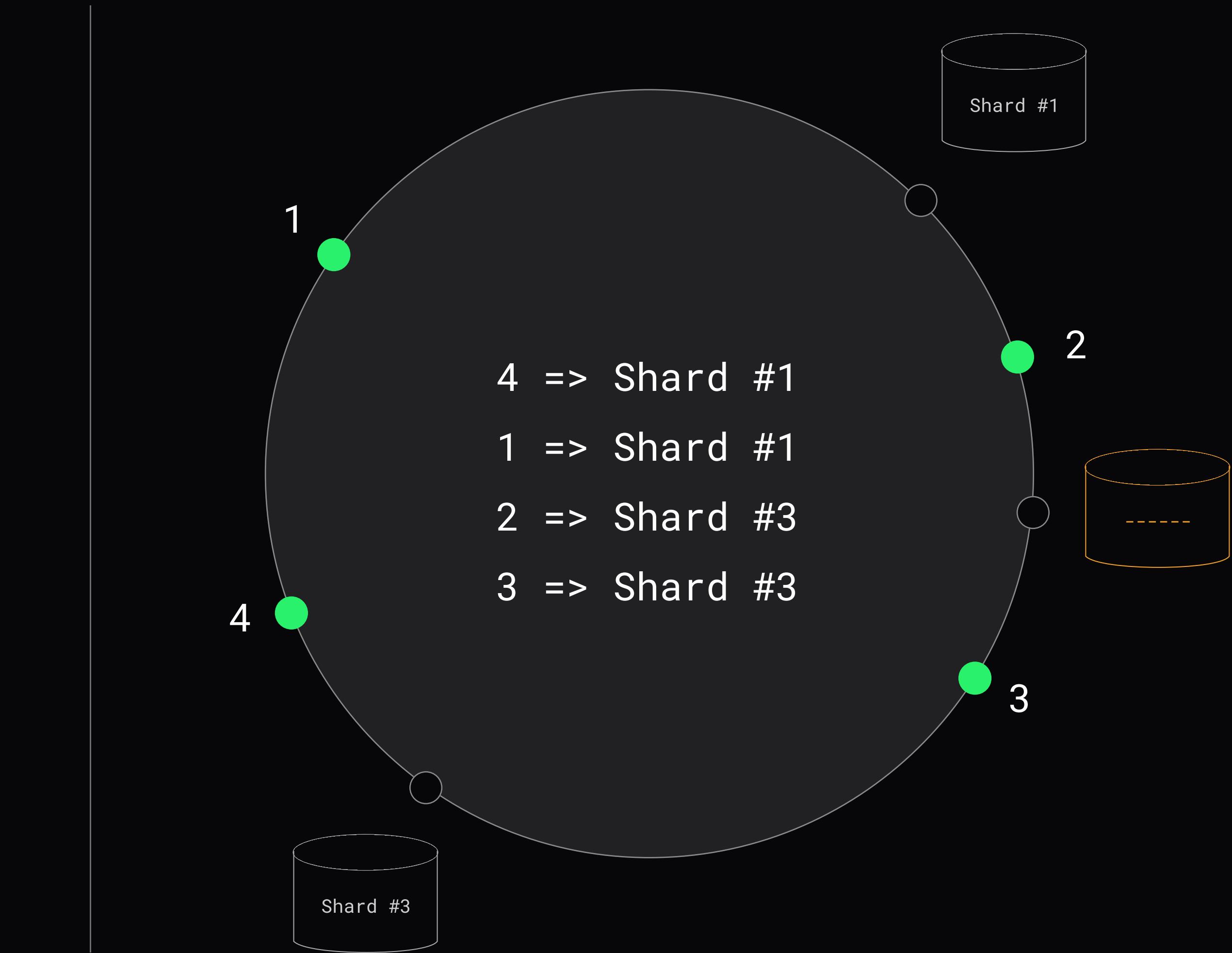
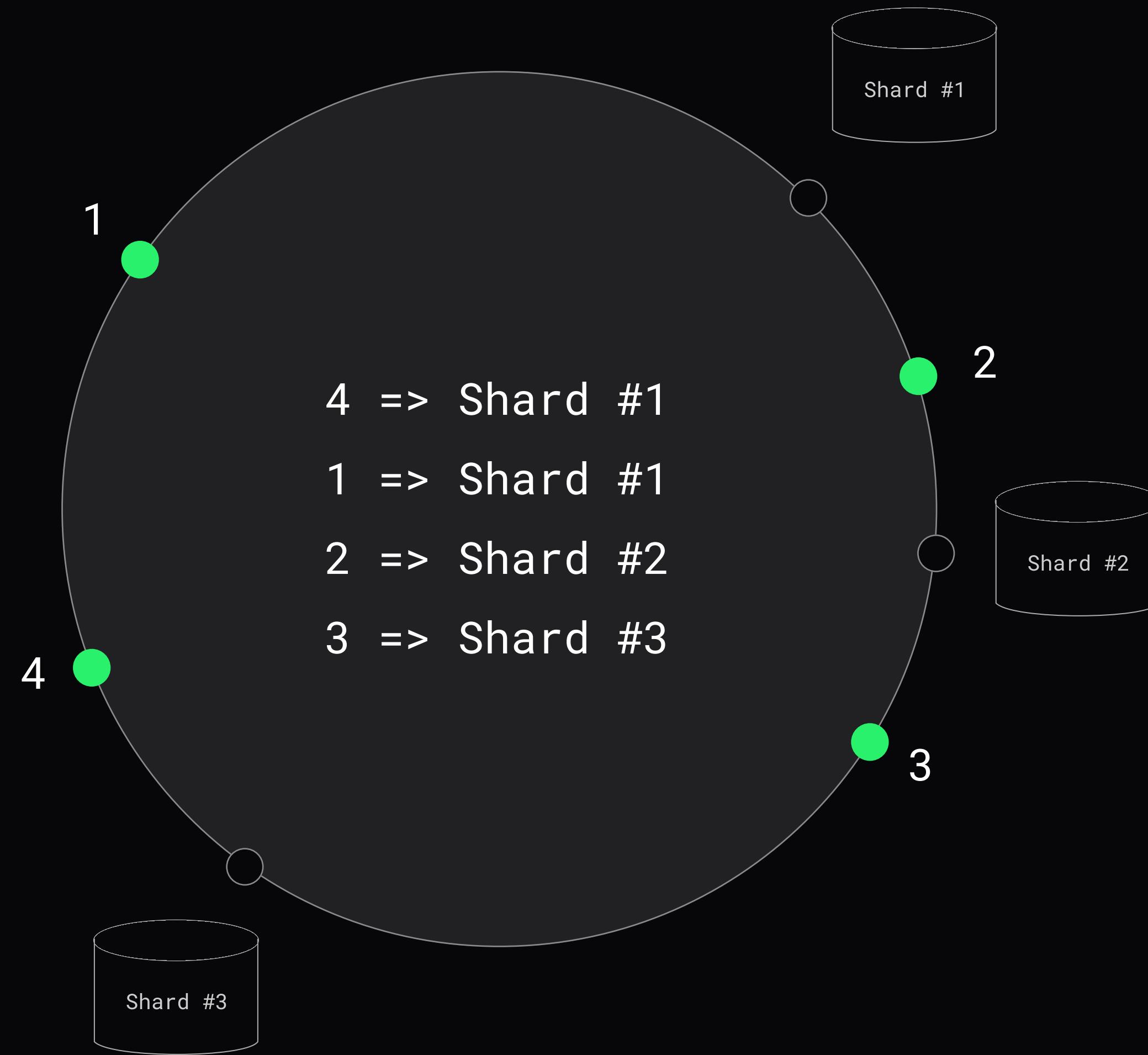


$F(\text{key}) = \text{hash}(\text{key}) \% \text{shards_number}$

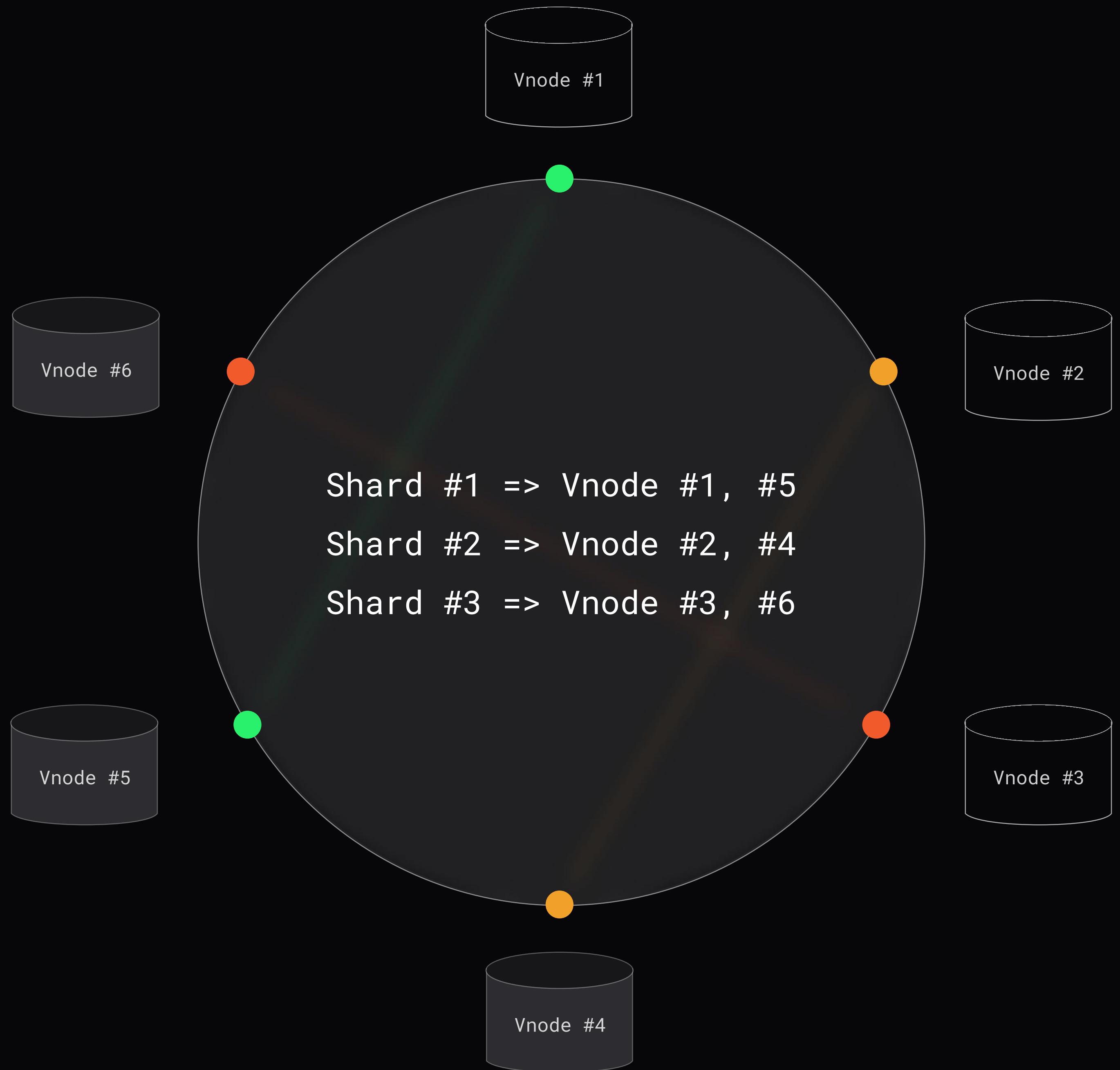


$$\begin{aligned}5 \% 4 &= 1 \\6 \% 4 &= 2 \\7 \% 4 &= 3\end{aligned}$$

CONSISTENT HASHING



CONSISTENT HASHING



$F(123, 1) = \text{hash}(123, 1) = 345$
 $F(123, 2) = \text{hash}(123, 2) = 456$
 $F(123, 3) = \text{hash}(123, 3) = 121$



$F(242, 1) = \text{hash}(242, 1) = 233$
 $F(242, 2) = \text{hash}(242, 2) = 124$
 $F(242, 3) = \text{hash}(242, 3) = 434$



$F(123, 1) = \text{hash}(123, 1) = 345$
 $F(123, 2) = \text{hash}(123, 2) = 456$



$F(242, 1) = \text{hash}(242, 1) = 233$
 $F(242, 2) = \text{hash}(242, 2) = 124$



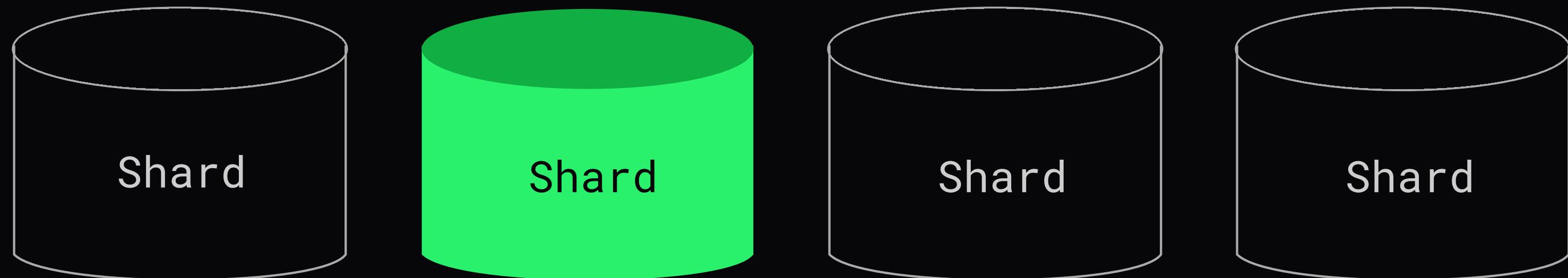
RANDEZVOUS HASHING

ПРИМЕРЫ

КАК РАСПРЕДЕЛИТЬ ДАННЫЕ ПОЛЬЗОВАТЕЛЕЙ (СЧЕТА, ТРАНЗАКЦИИ)

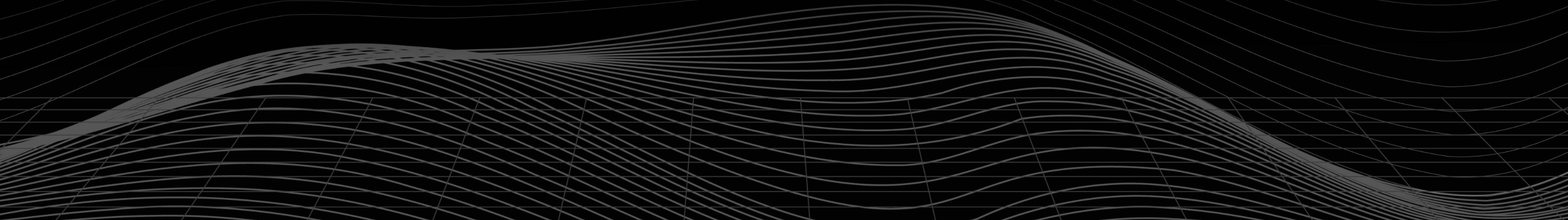
Из мобильного кошелька по шардам?

Пишем в один шард

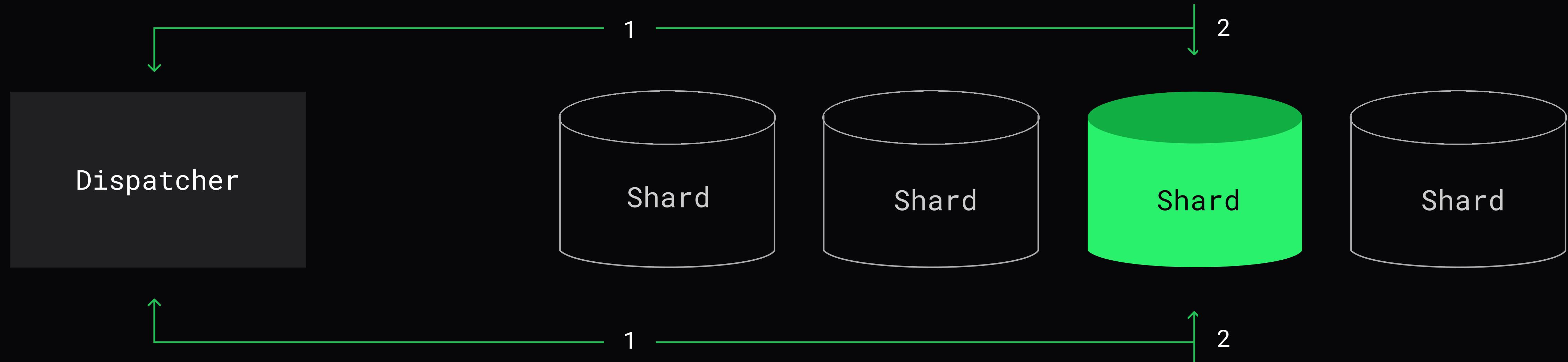


Читает из одного шарда

КАК РАСПРЕДЕЛИТЬ СООБЩЕНИЯ В ШАРДАХ ПО ЧАТАМ?



SMART SHARDING



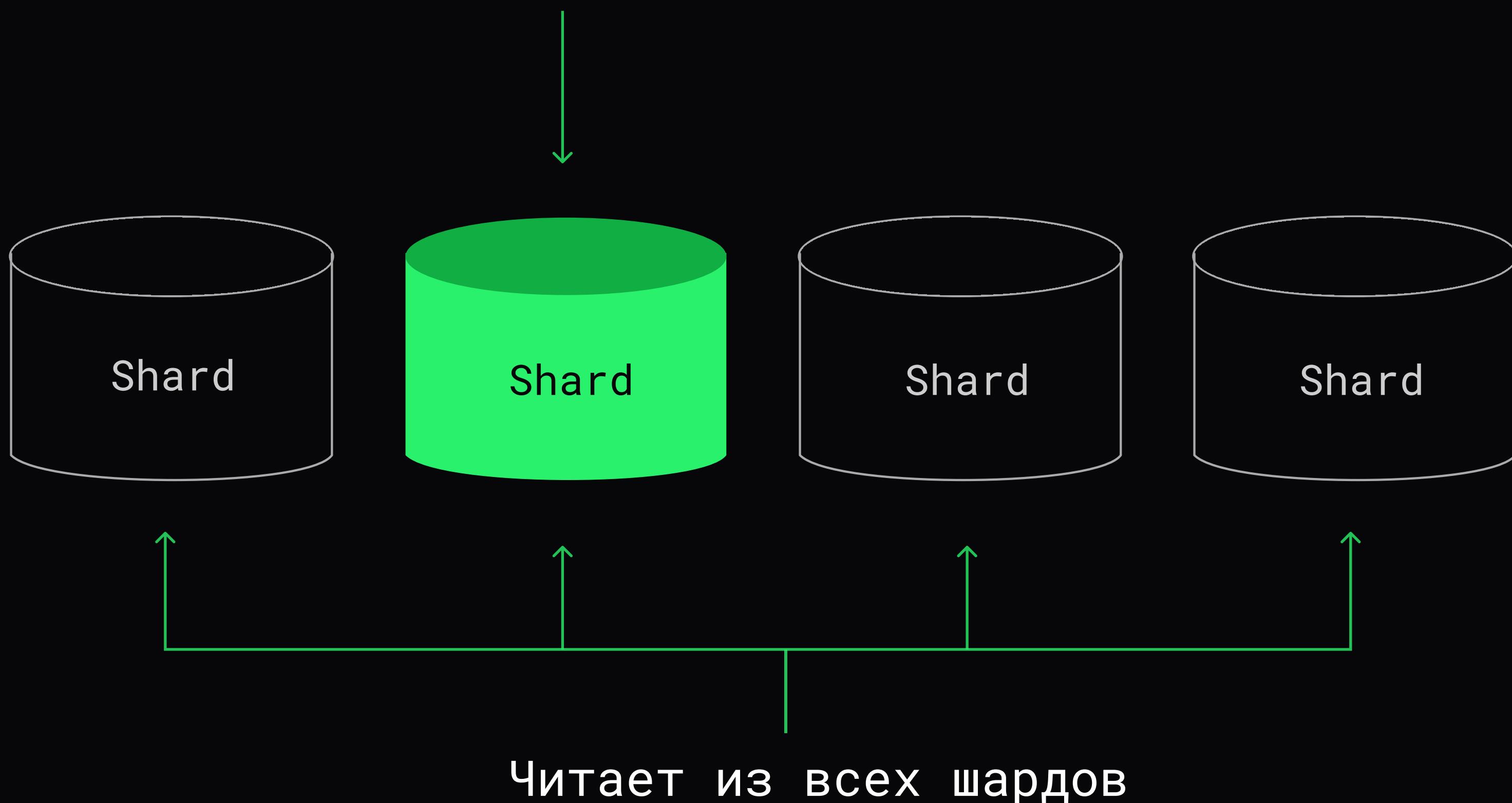
Пишем один шард
после похода в диспетчер

Читает из одного шарда
после похода в диспетчер

КАК РАСПРЕДЕЛИТЬ ТРЕЙСЫ ПОЛЬЗОВАТЕЛЕЙ ПО ШАРДАМ?

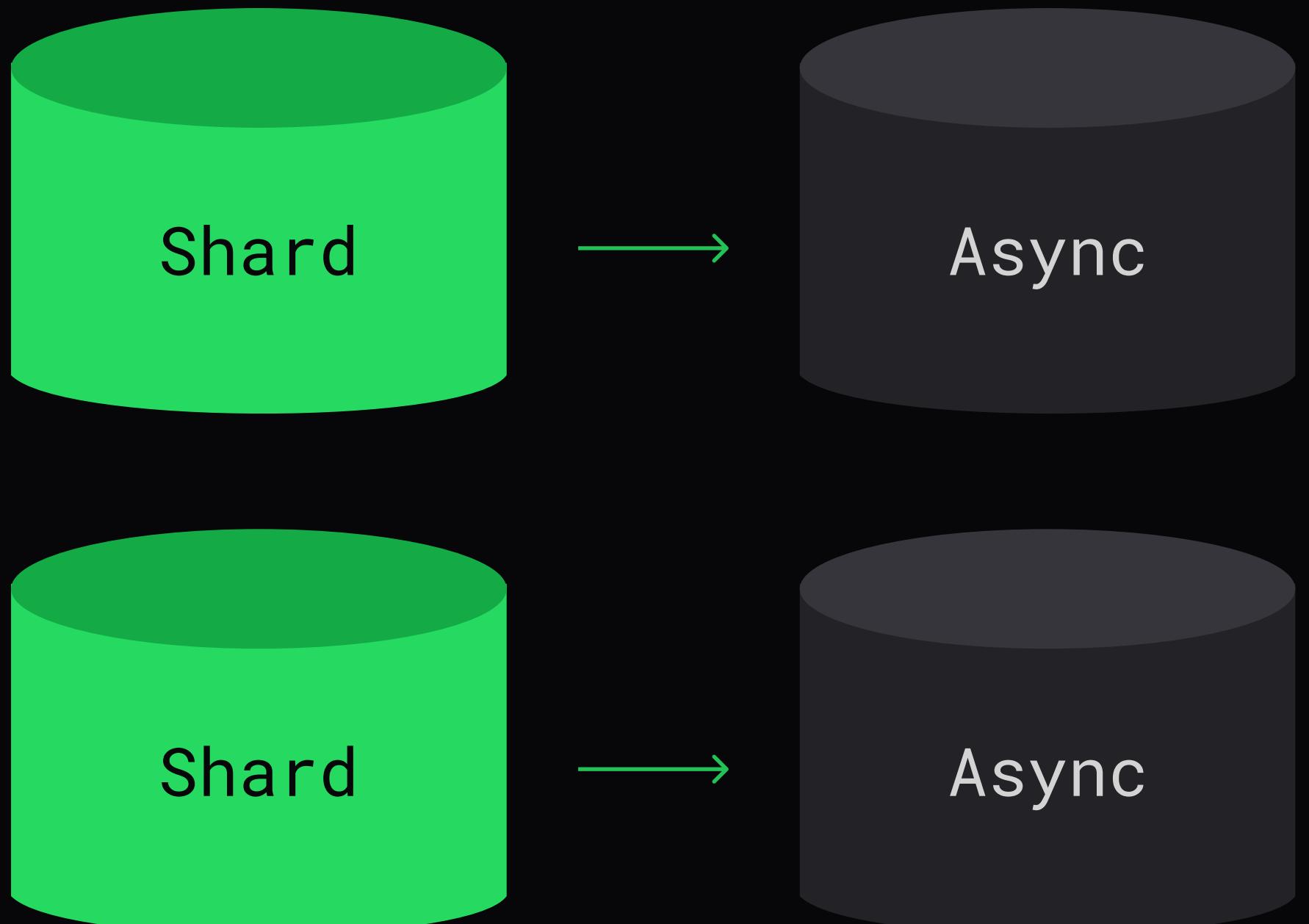
Соотношение чтения к записи 1 : 300

Пишем в случайный шард

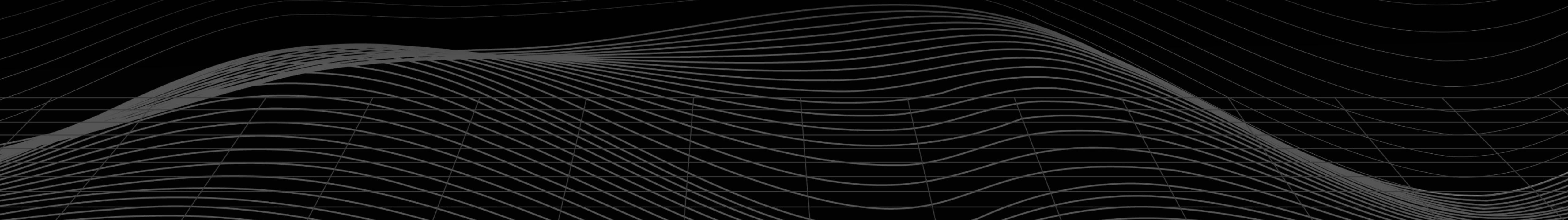


ПРЕДСТАВИМ, ЧТО ВЫ ТАКЖЕ РАЗРАБАТЫВАЕТЕ TWITTER

в какой-то момент пользователь
А публикует в Твиттере фотографию
своей собаки, а пользователь Б
отвечает на это фото в твиттере
комплimentом собаке



ПОЛЬЗОВАТЕЛЬ МОЖЕТ
НЕ УВИДЕТЬ ТВИТ,
НА КОТОРЫЙ ЕСТЬ ОТВЕТ



Terminal: System Design × + ▾



СОГЛАСОВАННОЕ ПРЕФИКСНОЕ ЧТЕНИЕ

Можно гарантировать, чтобы база данных всегда применяла зависимые операции в одном и том же порядке, а именно писала в один и тот же шард



FAQ

Шардирование

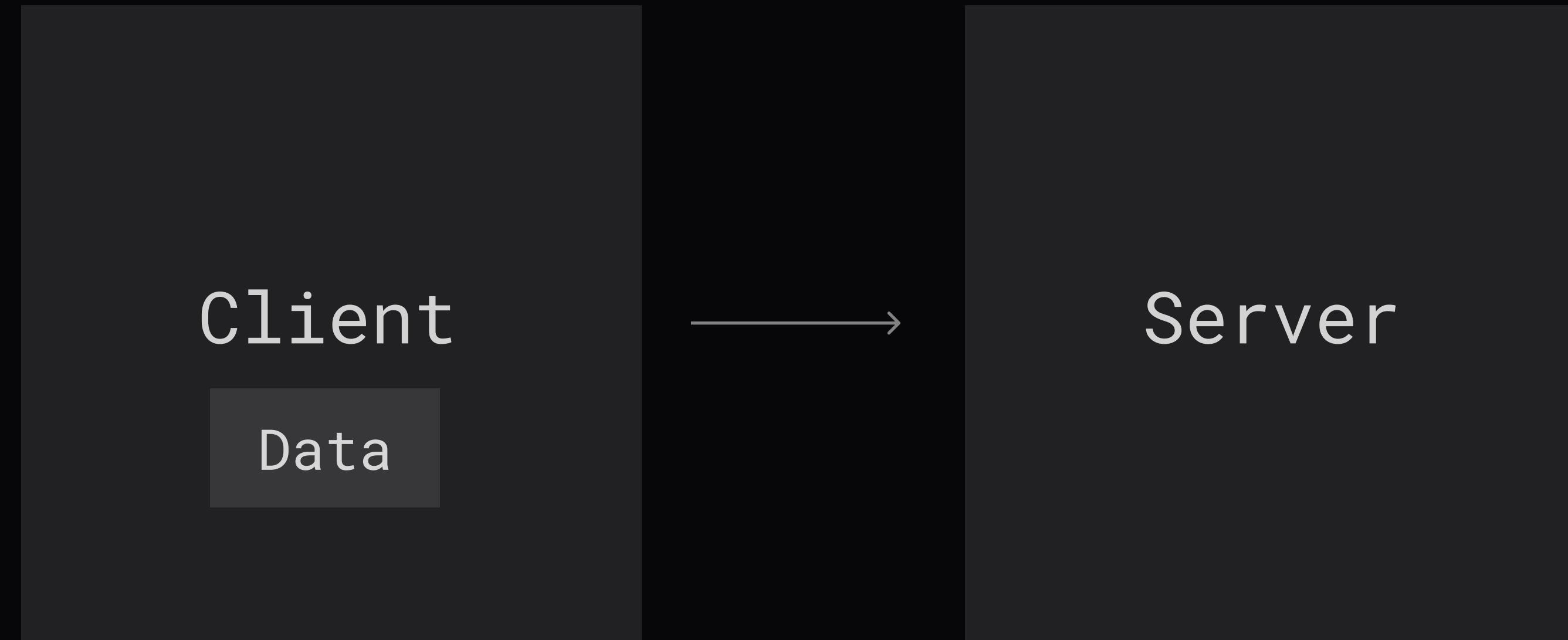
Способ шардирования, routing,
перебалансировка, решардинг



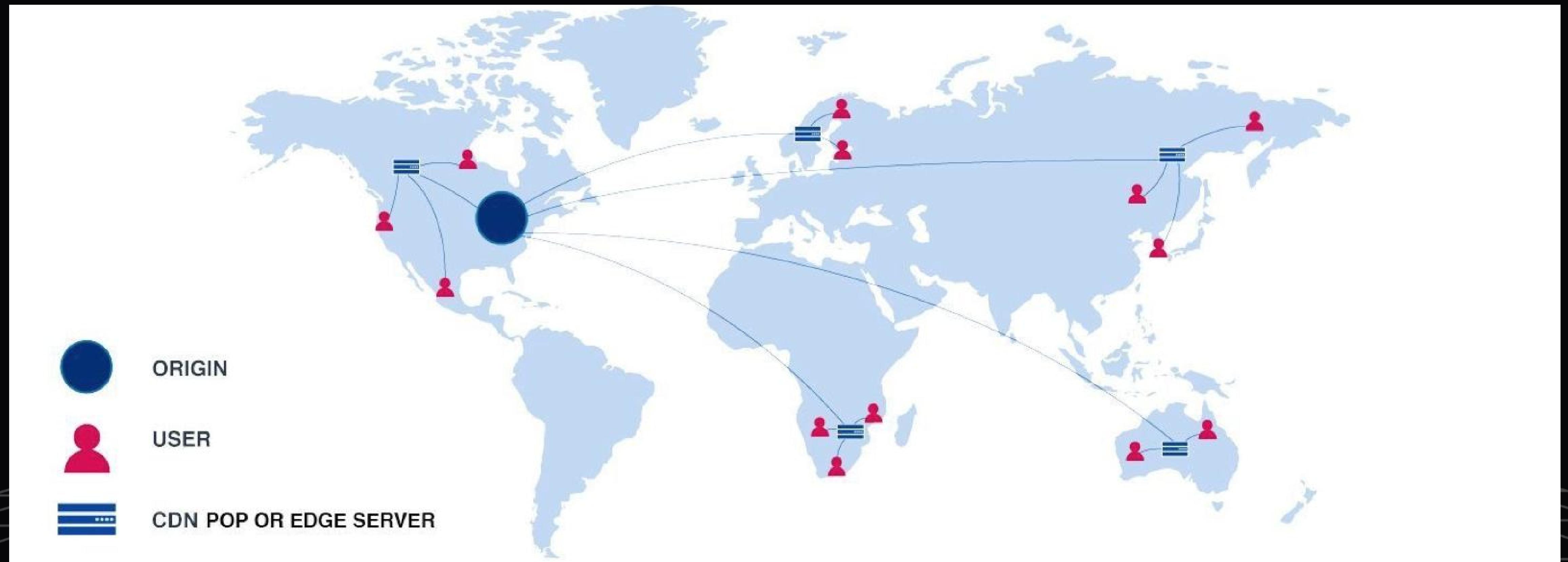
LET'S PRACTISE

АЛЬТЕРНАТИВНЫЕ СПОСОБЫ ХРАНЕНИЯ ДАННЫХ

ХРАНЕНИЕ НА КЛИЕНТЕ



CDN



CDN

Исходный (origin)

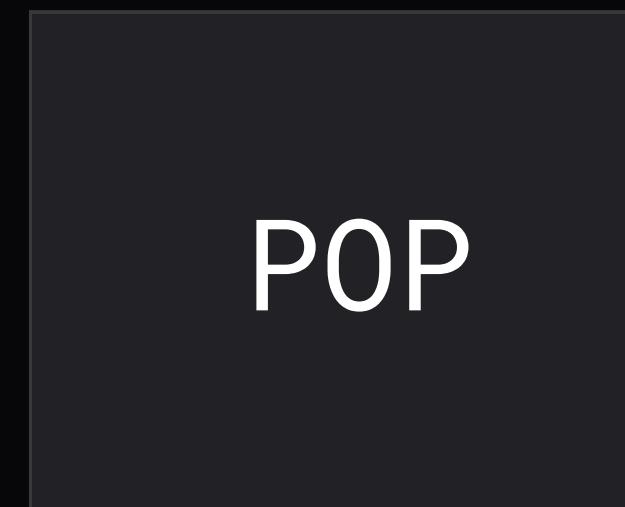
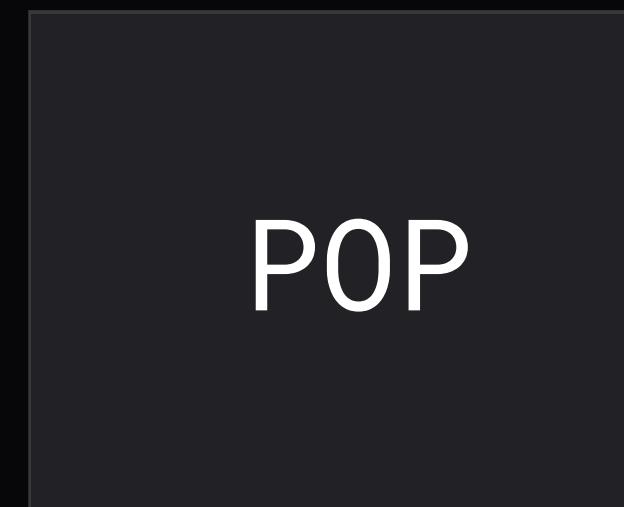
на котором размещен запрашиваемый сайт, а также связанный с ним визуальный, музыкальный, видеоконтент

PoP (point of presence)

точка присутствия вспомогательных серверов.
Их сеть размещается в различных регионах

Proxy-сервер

это промежуточное звено между пользователем и исходным сервером. Он отвечает за перенаправление, оптимизацию и преобразование передаваемого трафика

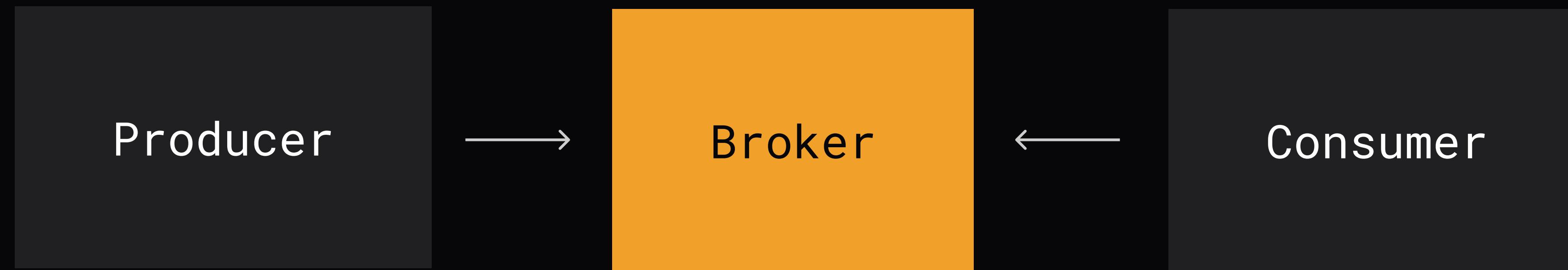


FAQ

Брокеры сообщений

CDN, Хранение на клиенте,
Сжатие данных, Охлаждение данных

дополнительное



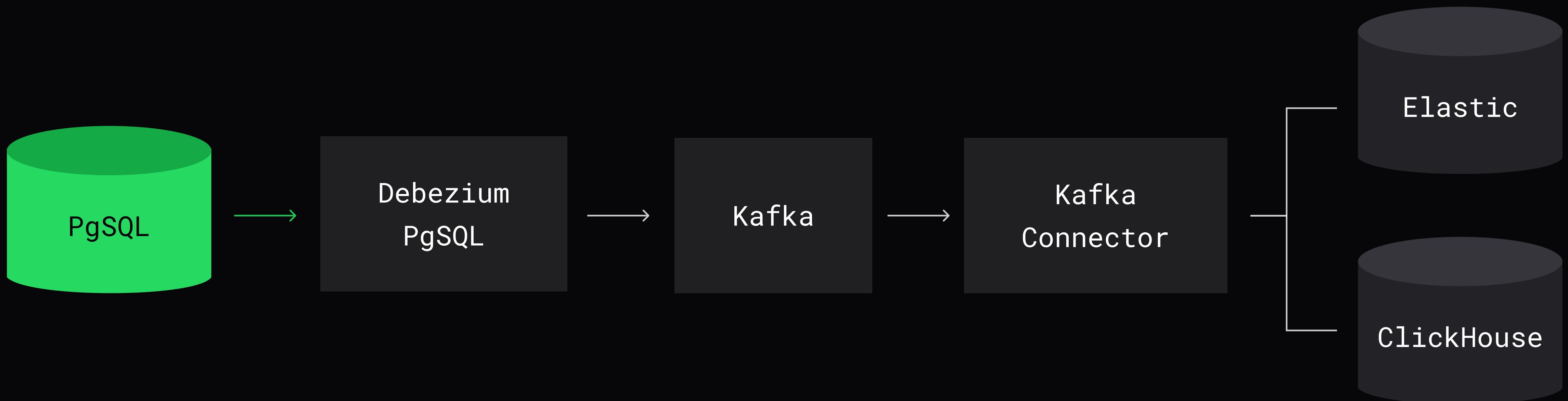
Единая точка отказа

KAFKA CLUSTER

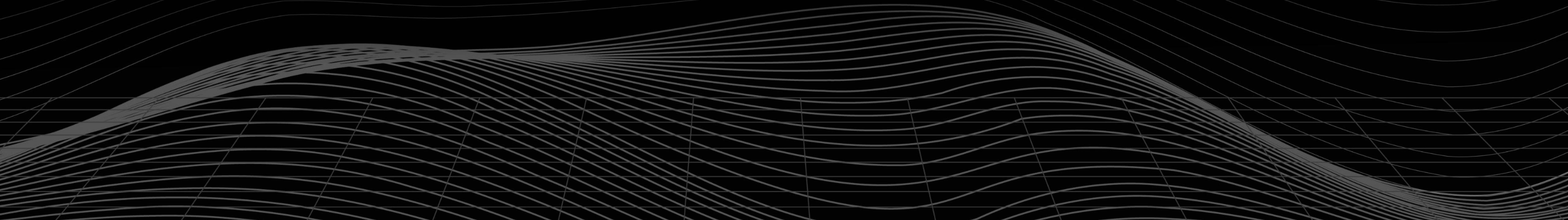


**КАК доносить изменения
из одного хранилища в другое?**

CAPTURE DATA CHANGE (DEBEZIUM)



КАК В ПРИЛОЖЕНИЕ
ПОСТОЯННО ДОНОСИТЬ
ИЗМЕНЕНИЯ ИЗ ХРАНИЛИЩА?



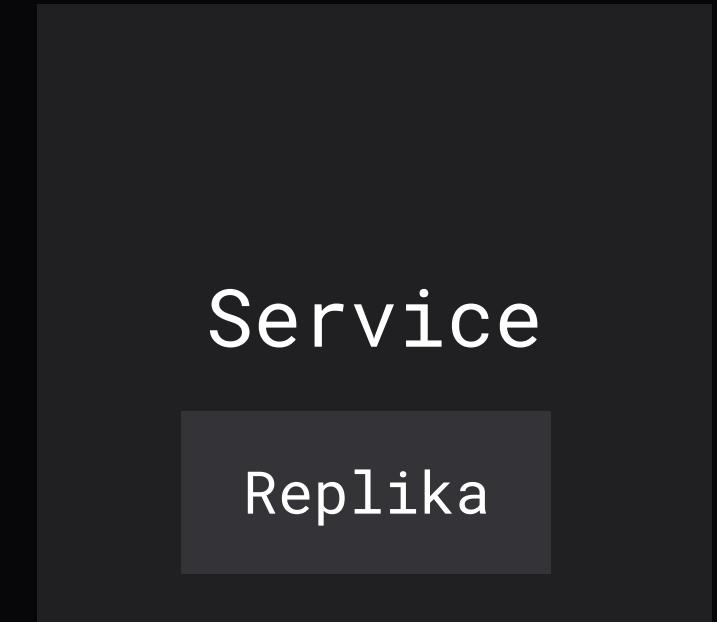
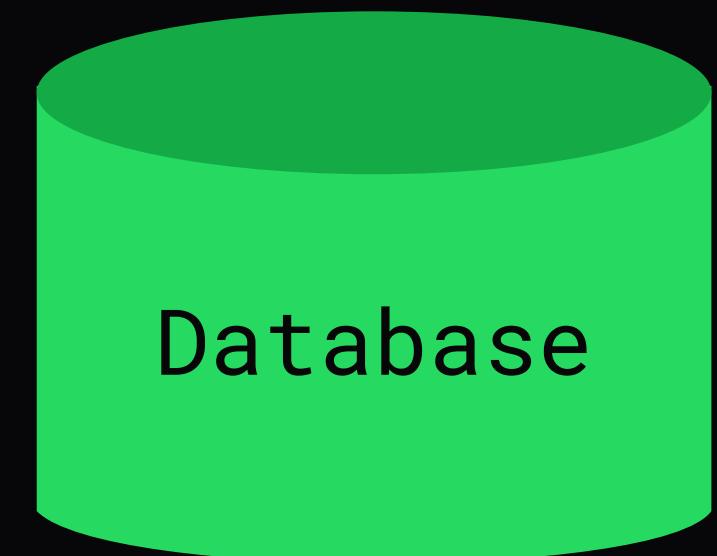
LIBSLAVE

ABOUT

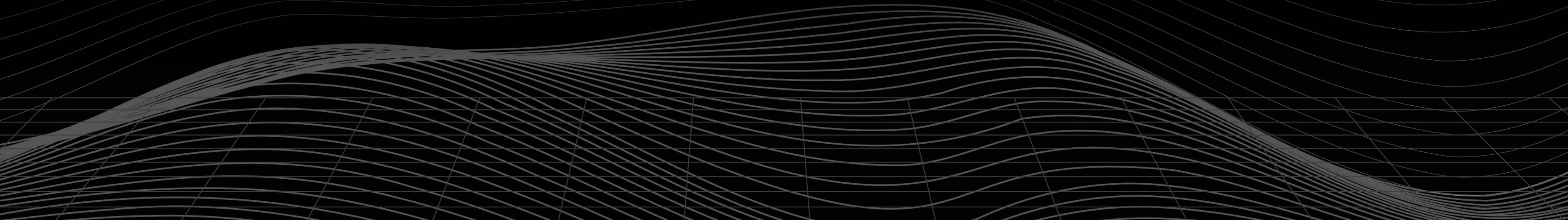
This is a library that allows any arbitrary C++ application to connect to a Mysql replication master and read/parse the replication binary logs.

In effect, any application can now act like a Mysql replication slave, without having to compile or link with any Mysql server code.

One important use-case for this library is for receiving changes in the master database in real-time, without having to store the master's data on the client server.



КАК СДЕЛАТЬ ДЕЦЕНТРАЛИЗОВАННУЮ ОЧЕРЕДЬ СООБЩЕНИЙ?





FAQ

Дополнительное

Kafka cluster, Debezium, Libslave

ДОМАШНЕЕ ЗАДАНИЕ

ДО ВСТРЕЧИ
НА СЛЕДУЮЩЕМ
ЗАНЯТИИ!