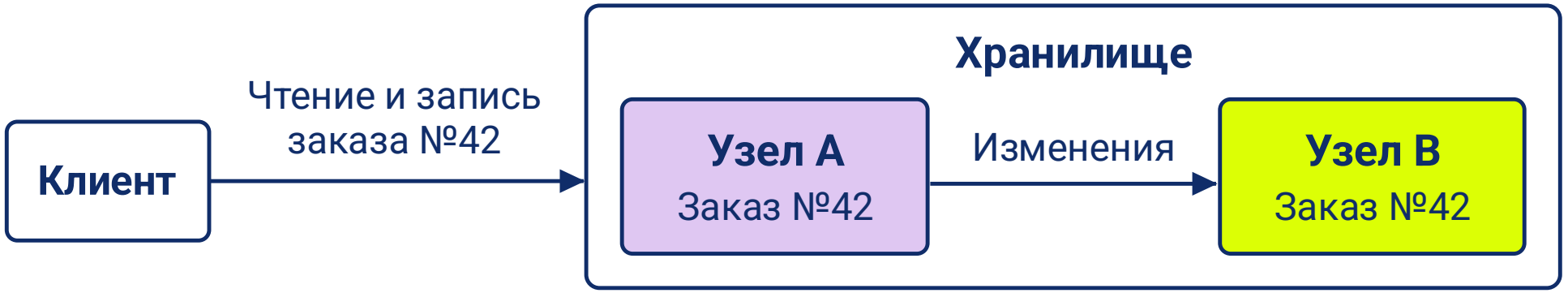
9. Репликация данных

Сухорослов Олег Викторович

09.11.2026

Что такое репликация данных?

Репликация данных – поддержание копий одних и тех же данных на нескольких узлах. Каждая такая копия называется **репликой**.



Клиент может читать и изменять заказ через единый интерфейс системы, не зная о том что запросы могут обслуживать разные реплики.

Зачем используют репликацию?

Мотивация	**Что даёт репликация**
Сохранность данных	Если диск потерян, данные можно восстановить с другой копии.
Доступность	Пока один узел недоступен, другой может продолжить обслуживать запросы.
Меньшая задержка чтения	Клиент может прочитать данные с близкой реплики.
Большая пропускная способность чтения	Несколько реплик могут обслуживать запросы параллельно.

Цена репликации:

- Требуется больше ресурсов.
- Для изменяемых данных требуется обеспечивать согласованность.

Какое значение должен увидеть клиент?



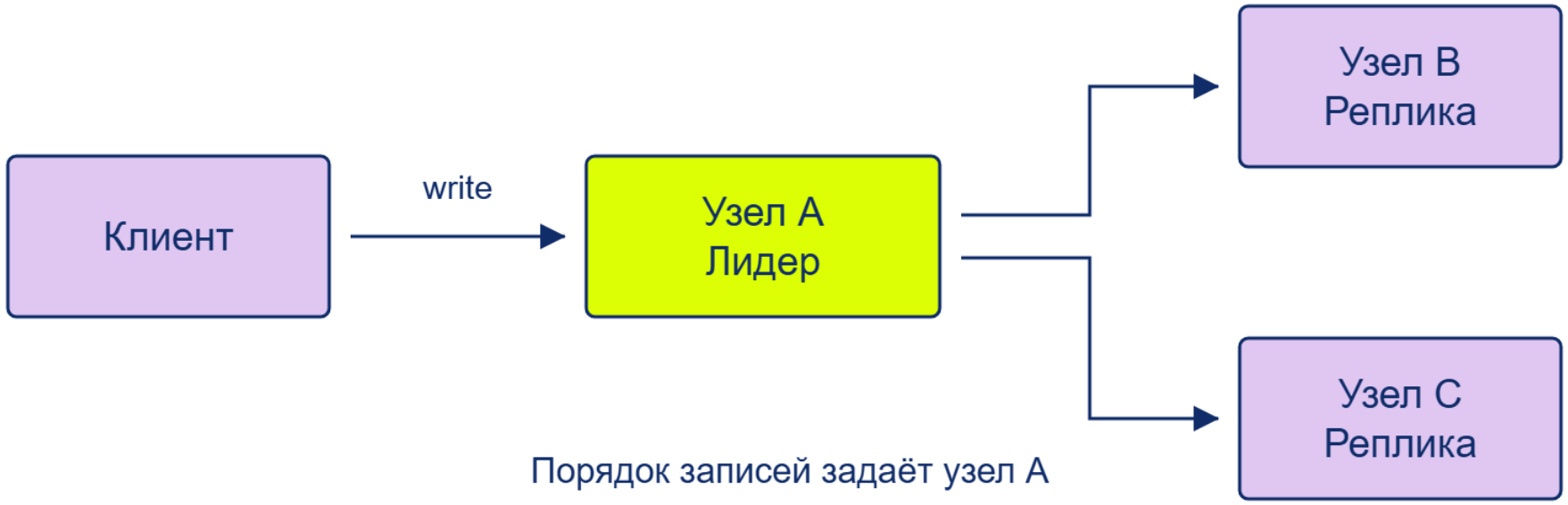
Модель согласованности (consistency model) задаёт допустимые истории операций: какие результаты могут получить клиенты с учётом сделанных записей и порядка вызовов.

Система обеспечивает модель X – любая история вызовов и ответов, которую система может показать клиентам, должна быть допустима в модели X.

План лекции

- Один лидер и подтверждение записи
- Несколько лидеров и конфликты
- Запись без лидеров, кворумы и восстановление
- Репликация через упорядоченную рассылку
- Линеаризуемость, CAP и eventual consistency
- Гарантии сессии и сравнение подходов

Один лидер для записей



- Лидер задаёт порядок изменений.
- Реплики получают журнал или изменения состояния.
- Возможность читать с реплик зависит от требуемой свежести.

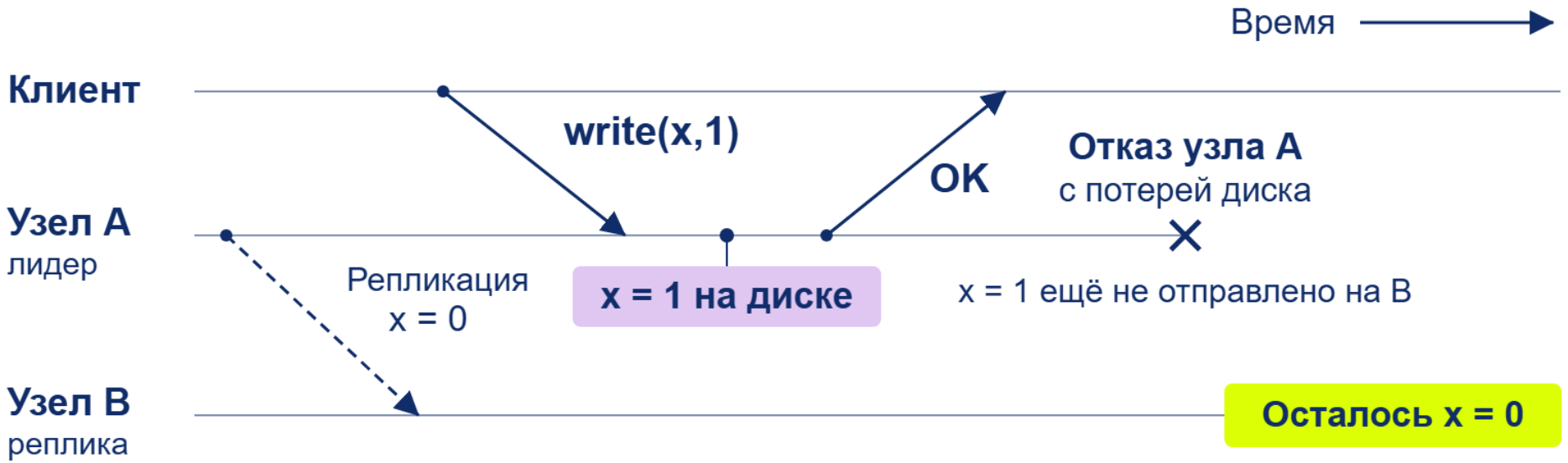
Что подтверждает COMMIT в PostgreSQL?

У лидера A одна синхронная реплика B.

Режим подтверждения задаёт этап, которого A ждёт от B.

synchronous_commit	**Что успела сделать B**
remote_write	Записать журнал WAL в файловую систему
on	Сохранить WAL на устойчивом носителе
remote_apply	Применить транзакцию

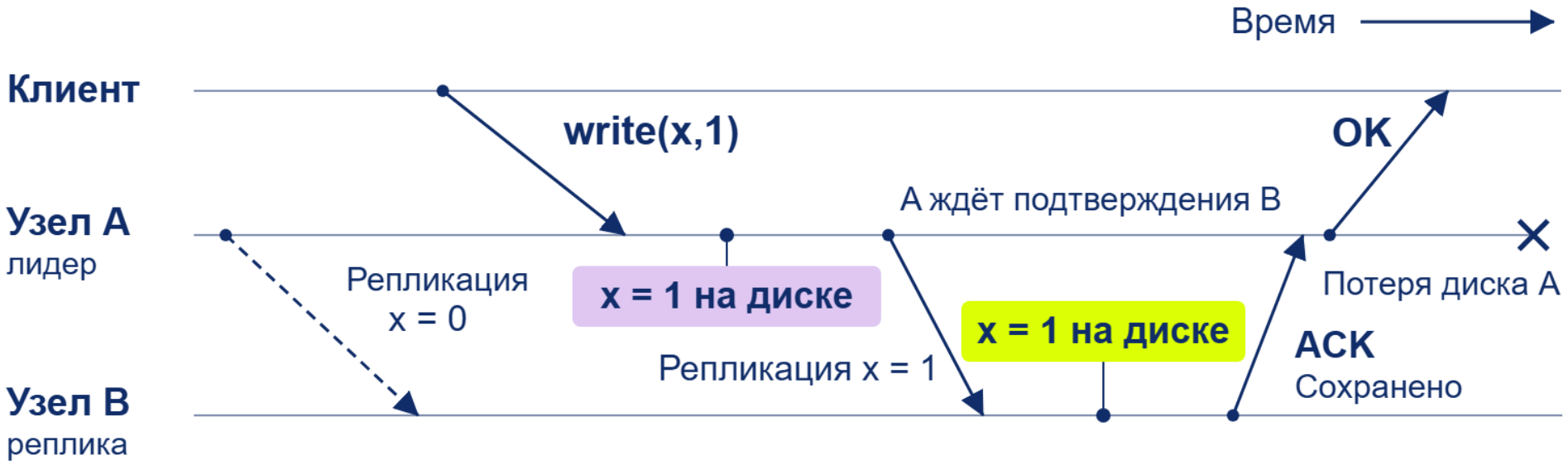
Асинхронная репликация



Лидер отвечает сразу после локального устойчивого сохранения записи.

- Ответ не ждёт обмена с репликами
- Реплики могут отставать
- При потере лидера подтверждённая запись может отсутствовать у остальных

Ожидание реплик перед ответом



Лидер A отвечает клиенту OK, когда сохранил запись на своём диске и получил от B подтверждение, что реплика тоже сохранила запись на диске.

- Запись переживёт потерю A, если копия на B сохранилась
- Задержка включает ожидание B
- Если B недоступен, лидер A не может вернуть клиенту OK и вынужден ожидать B

Как новая реплика догоняет лидера

1. Получить согласованный снимок на позиции *k*.
2. Применить изменения после *k* в порядке журнала.
3. Продолжать получать новые изменения.



Смена лидера после подозрения

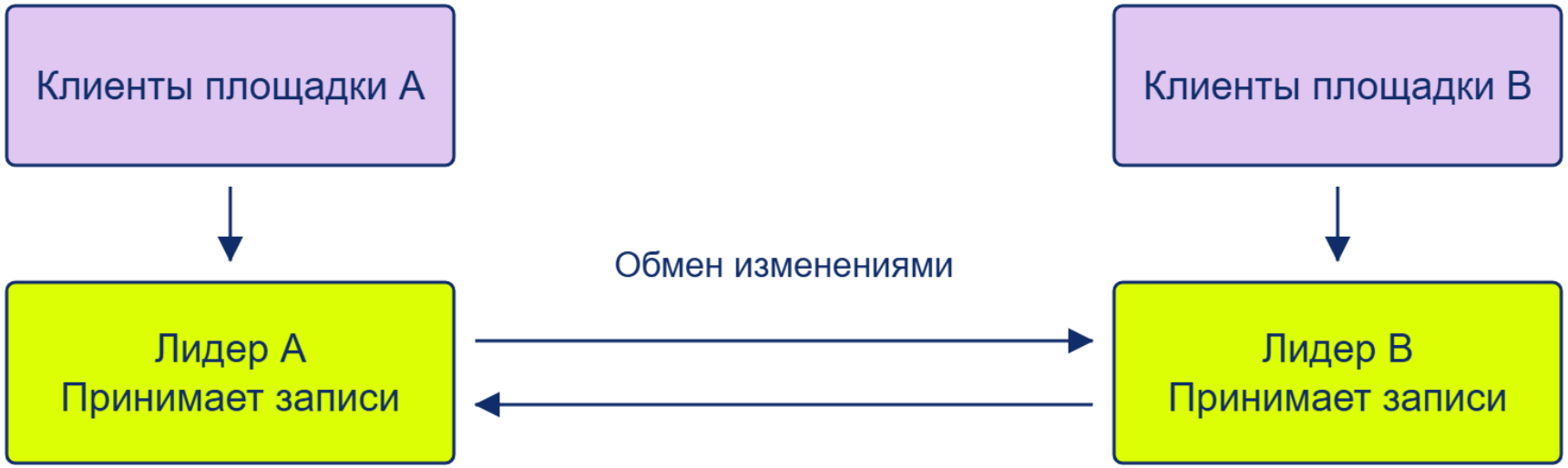


Подозрение в отказе A не означает, что A действительно остановился.

Передача роли лидера между узлами требует:

- сохранить необходимые подтверждённые записи
- согласовать дальнейший порядок
- исключить записи от старого лидера, если он еще активен (fencing)

Несколько лидеров

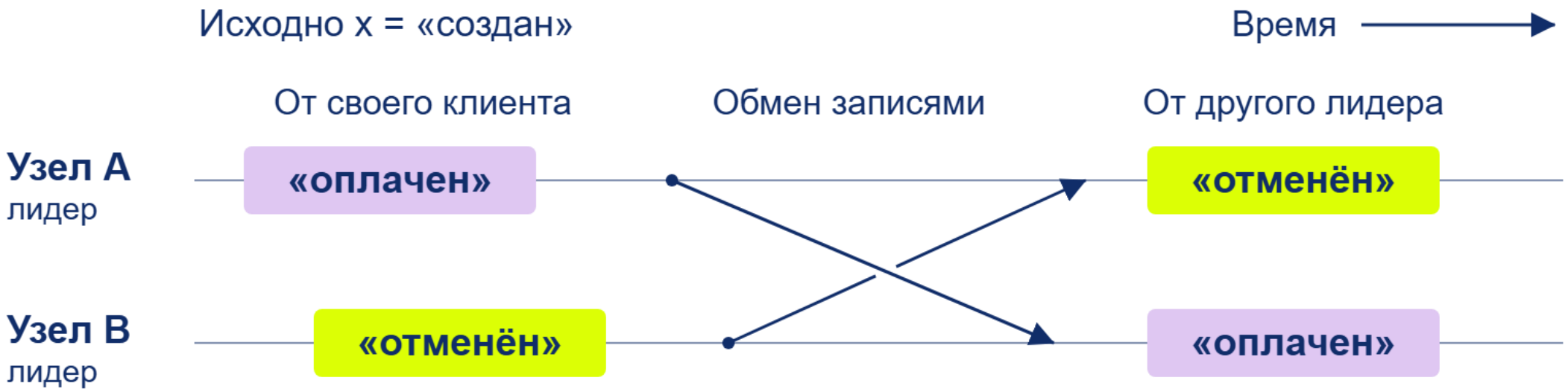


Записи принимаются в нескольких местах и затем передаются между ними.

Пример: в CouchDB две базы могут независимо менять один документ, а затем обменяться изменениями.

У изменений одного объекта может не быть общего порядка при приёме.

Два изменения одного значения



Обработка конкурирующих записей

Что делать с конфликтующими версиями?

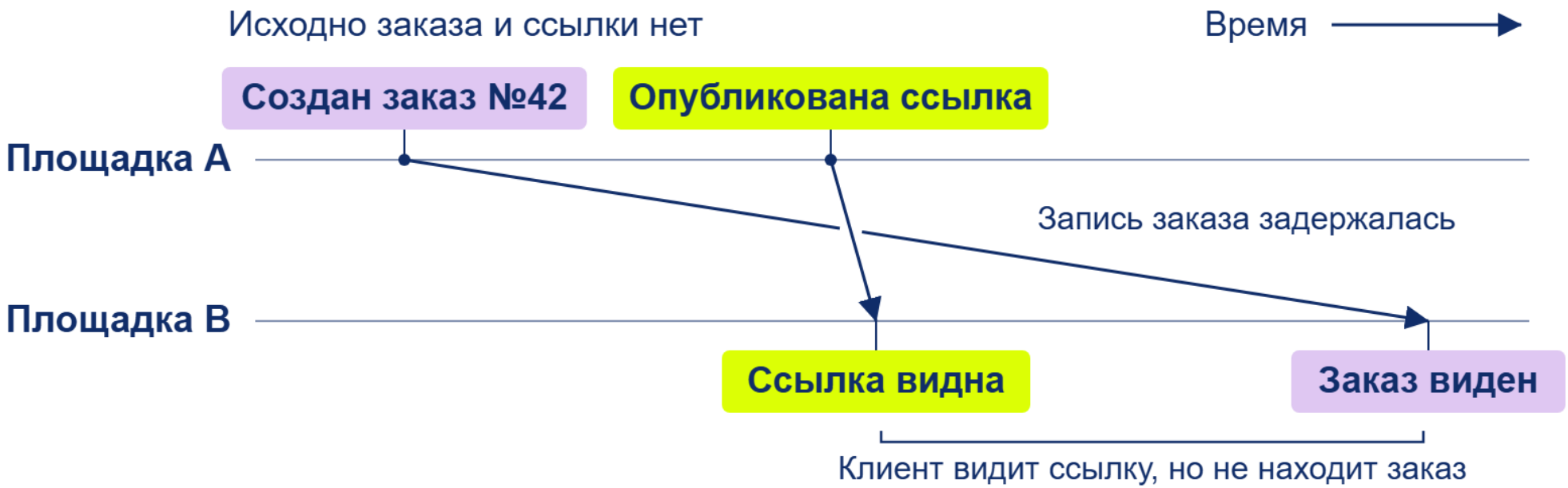
- Выбрать одну версию по общему правилу (например, last write wins)
- Сохранить конфликтующие версии
- Объединить изменения по правилам типа данных или приложения

Одинаковый результат на репликах не означает правильный для приложения.

Конфликты можно избежать используя “домашние площадки”.

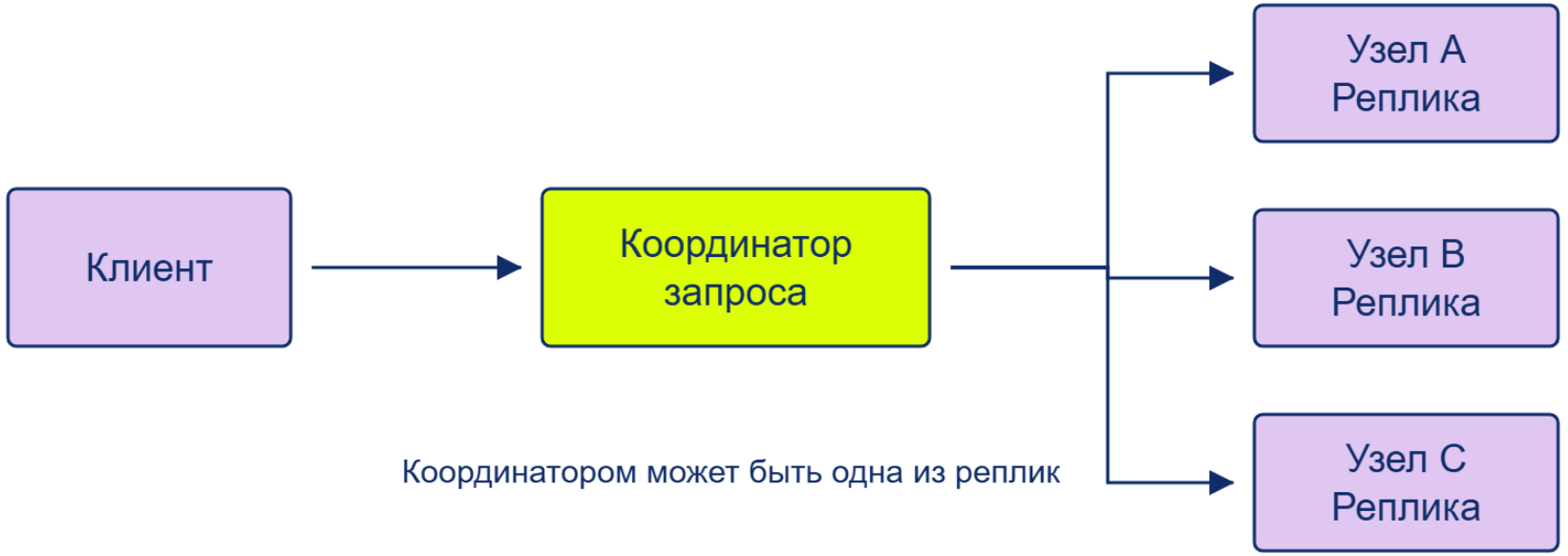
Зависимости между разными ключами

Создан заказ: **order42 = «создан»**. После этого опубликовано: **latest_order = 42**



Конфликт одного ключа и нарушение причинной зависимости – разные проблемы.

Запись без лидеров

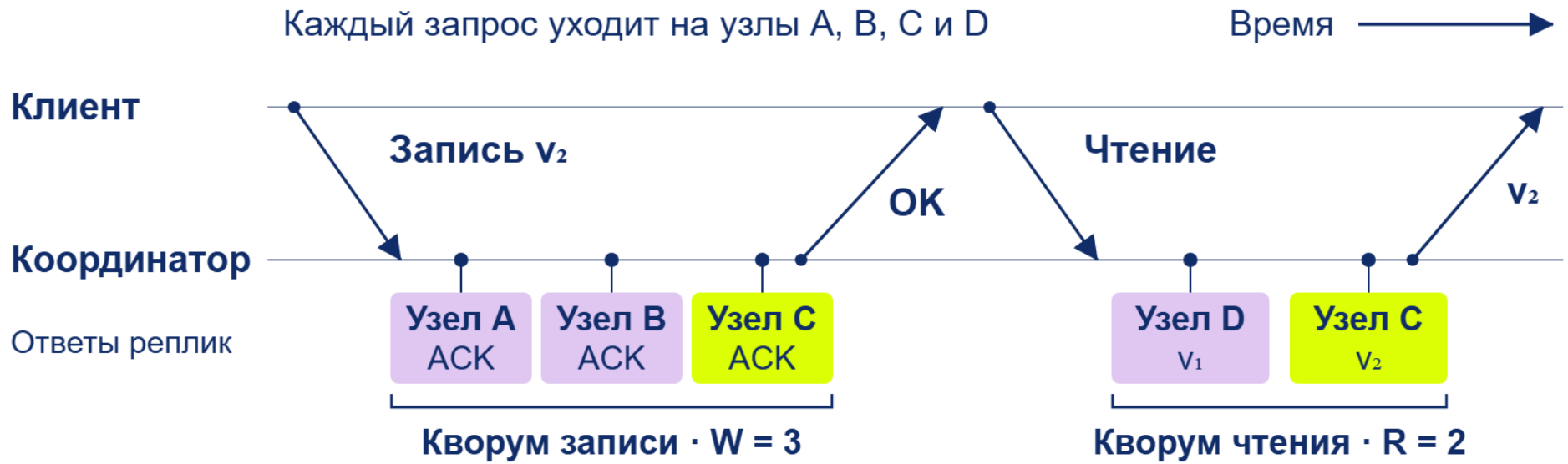


Координатор запроса (одна из реплик или отдельный узел):

- отправляет запись репликам
- собирает подтверждения
- при чтении получает и сравнивает версии

Пример: в Cassandra обычную запись координирует узел, который принял запрос клиента.

Кворумы чтения и записи



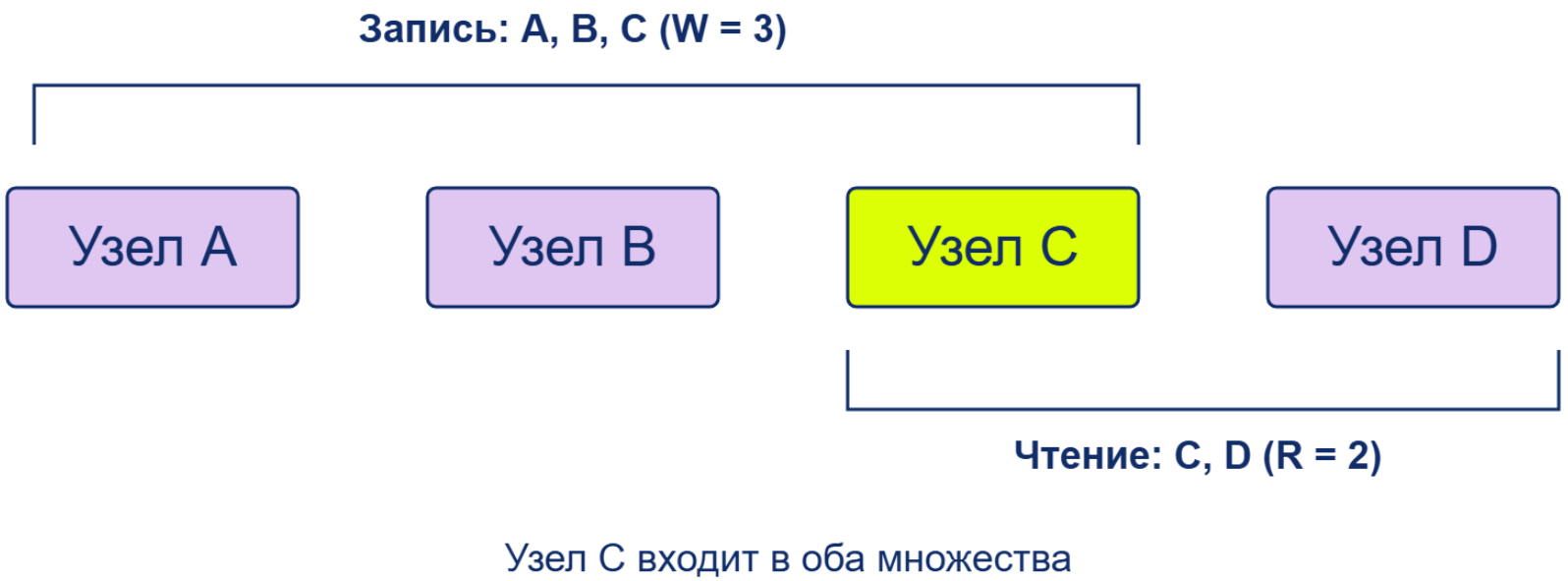
N – число реплик ключа.

W – сколько разных реплик должны подтвердить запись.

R – сколько разных реплик должны ответить на чтение.

В примере N = 4, W = 3, R = 2. Реплика отправляет ACK после устойчивого сохранения версии.

Пересечение кворумов



Для фиксированного набора из N реплик:

W + R > N ⇒ любая запись на W реплик пересекается с любым чтением R реплик.

Хотя бы один ответ чтения приходит от участника кворума записи.

Сколько отказов допускает конфигурация

N = 4, W = 3, R = 2. Остальные реплики отвечают координатору.

Операция	**Нужно ответов**	**Сколько реплик могут быть недоступны**
Чтение	2	N − R = 2
Запись	3	N − W = 1
И чтение, и запись	Для каждого запроса свой порог	N − max(R, W) = 1

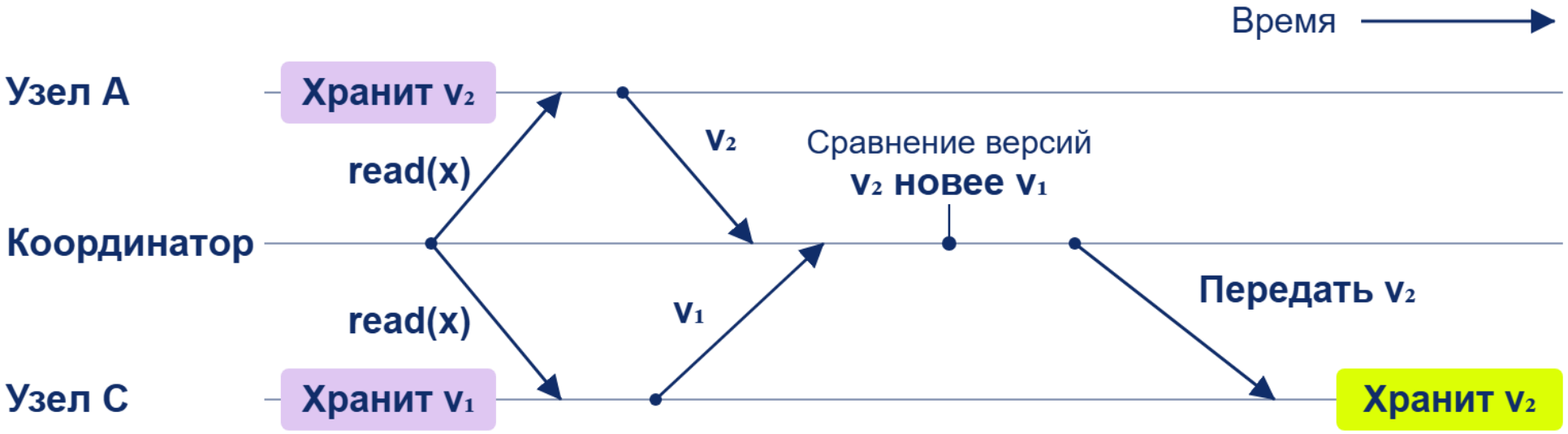
Чего не даёт пересечение кворумов

Правило W + R > N не определяет:

- как учитывать и сравнивать версии данных
- что делать с незавершённой записью
- как учитывать конкурентные записи
- как сохранять гарантии при смене состава узлов

Пересечение кворумов – важная часть протокола репликации без лидеров, но ее недостаточно для полноценных гарантий согласованности.

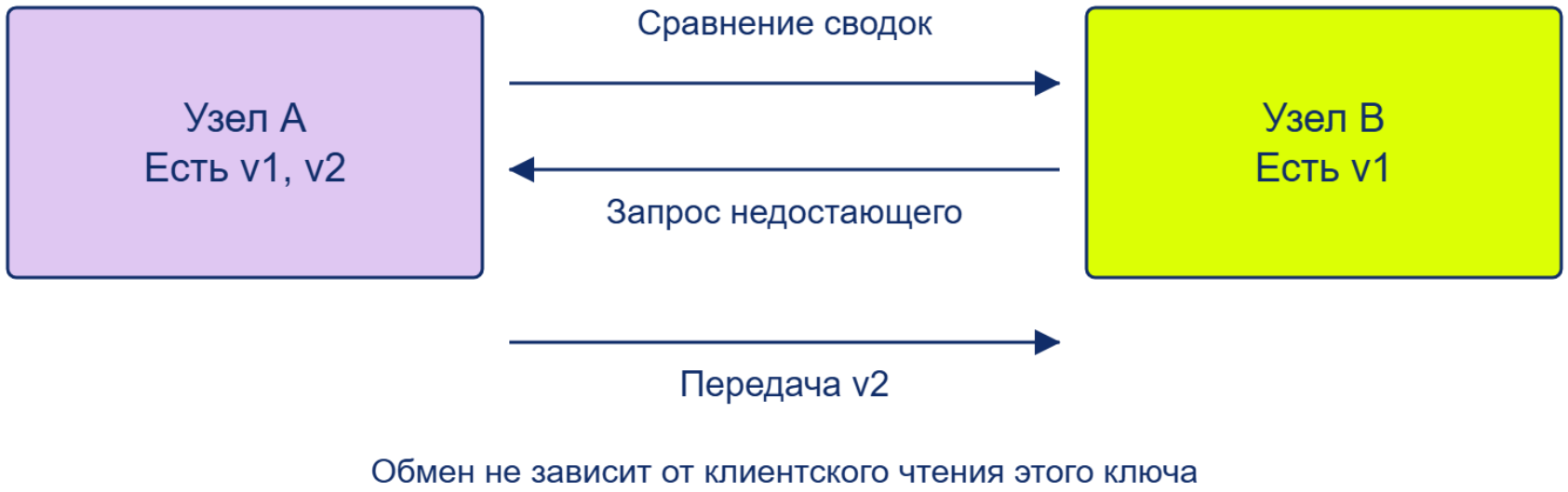
Восстановление при чтении



Механизм **Read repair***:*

1. Получить ответы реплик.
2. Определить актуальную версию или набор версий.
3. Передать недостающие данные отставшим репликам.

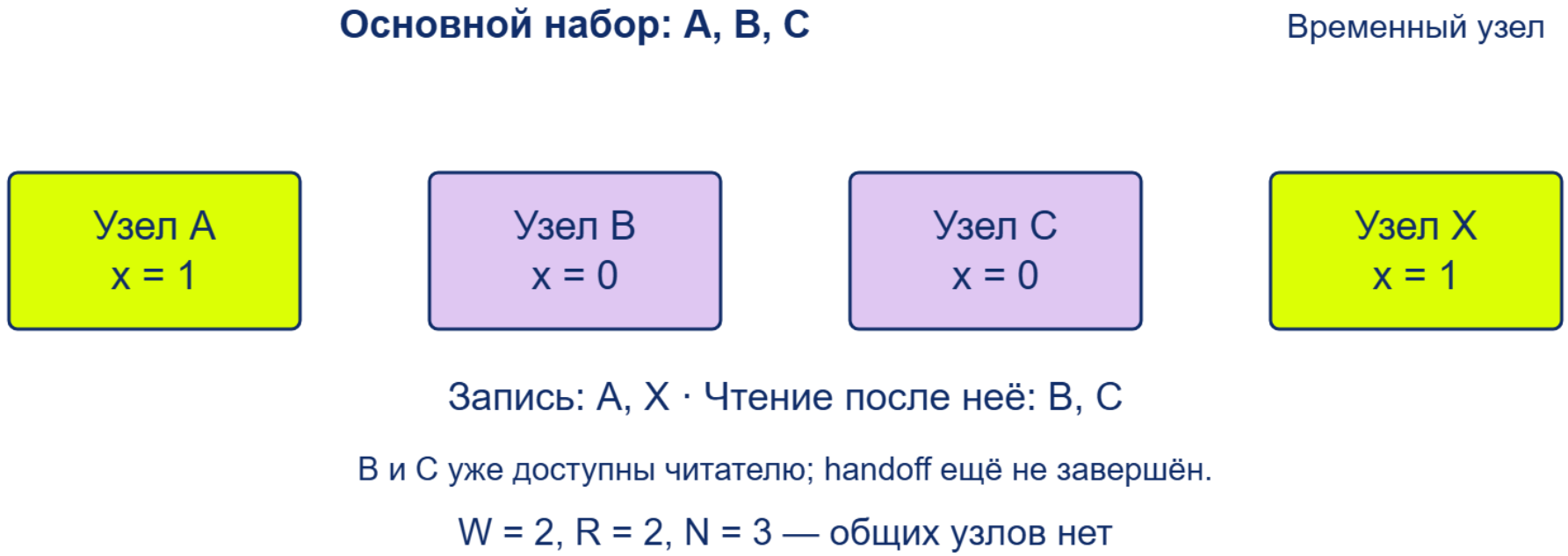
Фоновая сверка реплик



Механизм **Anti-entropy** работает независимо от клиентских чтений.

- Реплики сравнивают версии или сводки данных.
- Передают недостающие изменения.
- Повторяют обмен, пока расхождения сохраняются.

Временное размещение при недоступности



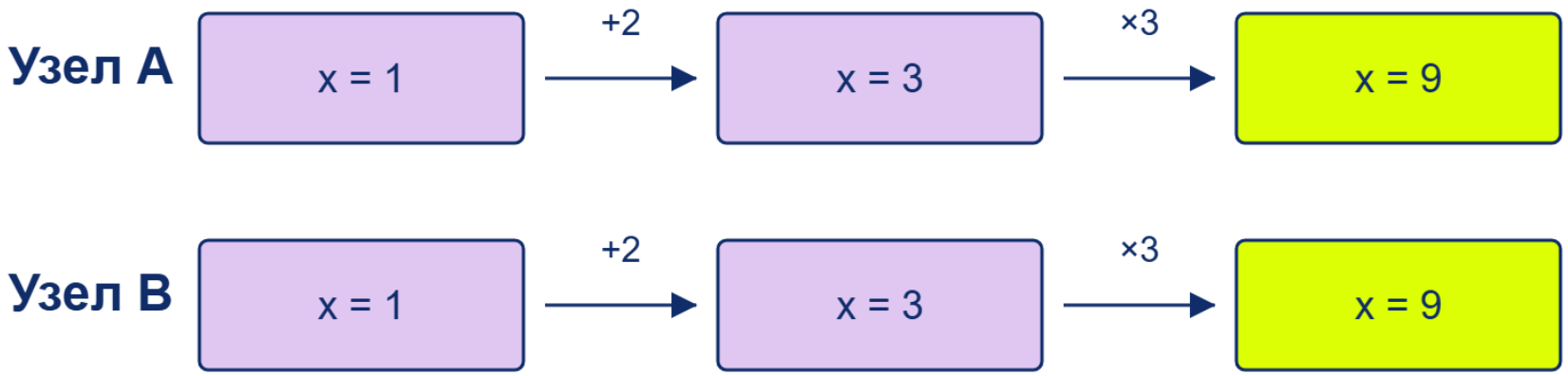
Sloppy quorum*:* часть записей временно сохраняется вне основного набора реплик.

Hinted handoff: временный хранитель позднее передаёт запись целевой реплике.

Если ответы взяты из разных наборов, W + R > N больше не доказывает их пересечение.

State Machine Replication (SMR)

- Одинаковое начальное состояние
- Одинаковая последовательность команд
- Детерминированное исполнение



После одинакового префикса команд состояния реплик совпадают.

Границы упорядоченной рассылки

Общий порядок команд нужно дополнить правилами:

- когда отвечать клиенту
- как сохранять принятые решения при отказах
- как обрабатывать повтор запроса
- откуда выполнять чтение

Общий порядок доставки сам по себе не задаёт контракт для read/write.

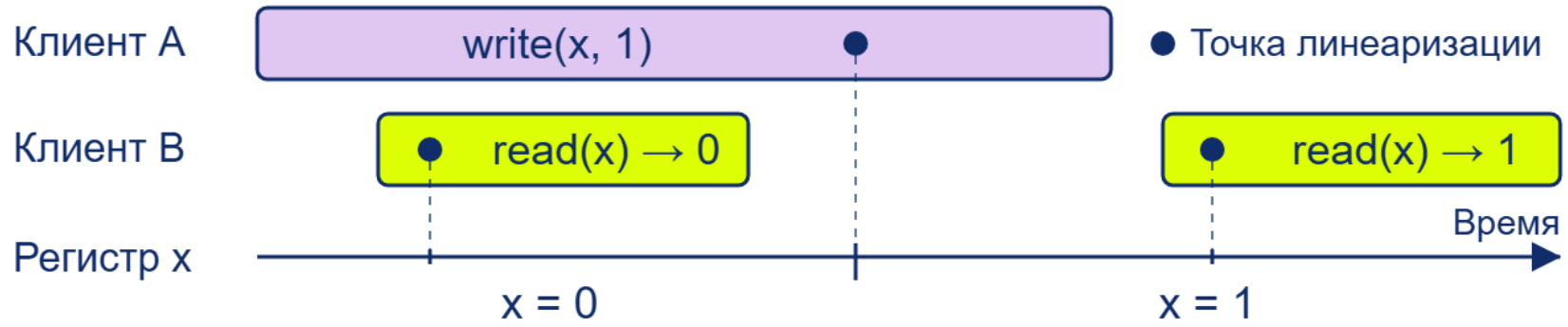
История операций для одного ключа



Операция занимает интервал от вызова до ответа.

- Непересекающиеся операции уже упорядочены реальным временем.
- Пересекающиеся операции могут допускать разные порядки.

Линеаризуемость

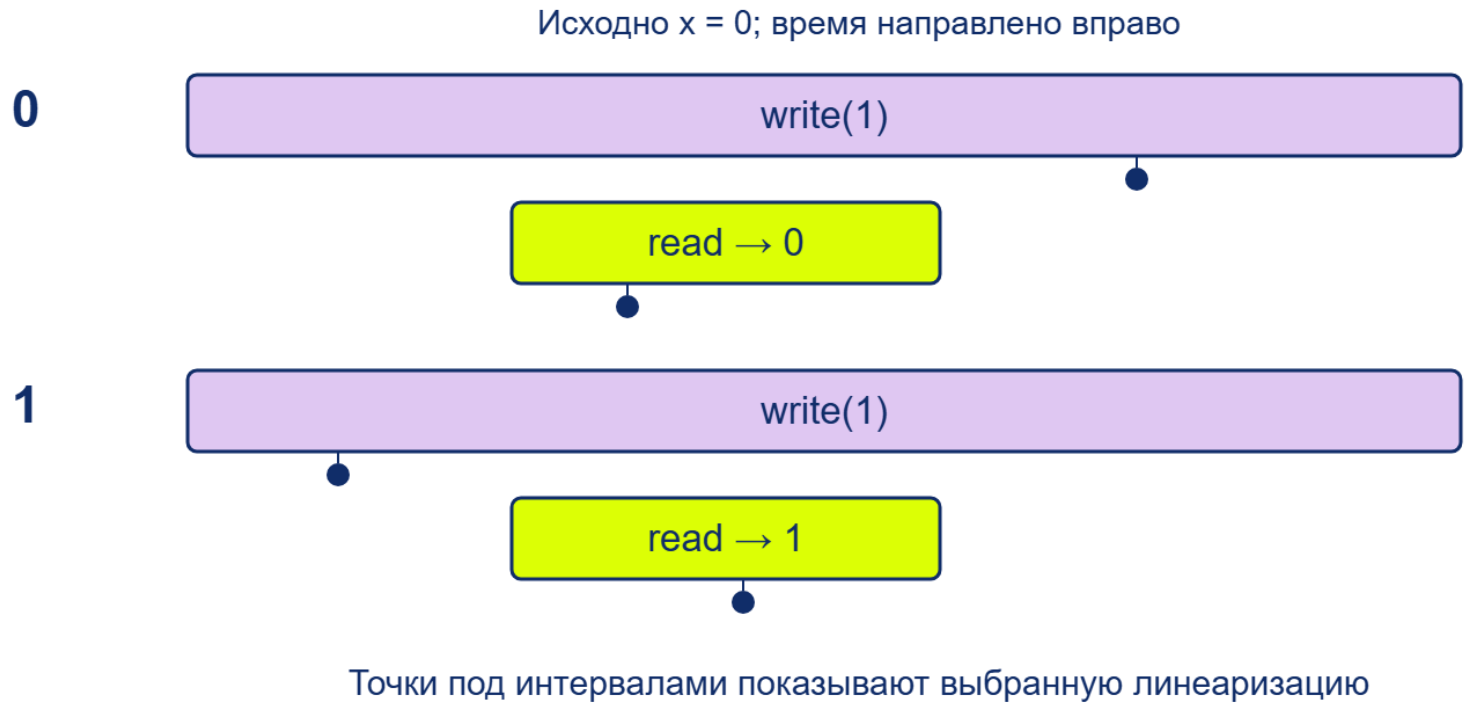


История операций линеаризуема, если все наблюдаемые ответы можно объяснить **одним допустимым порядком** операций над одной копией объекта. Каждая операция как будто происходит мгновенно между её вызовом и ответом.

Если одна операция завершилась до вызова другой, в общем порядке она должна стоять раньше.

Для регистра read(x) возвращает значение последней предшествующей записи в этом порядке или начальное значение, если такой записи нет.

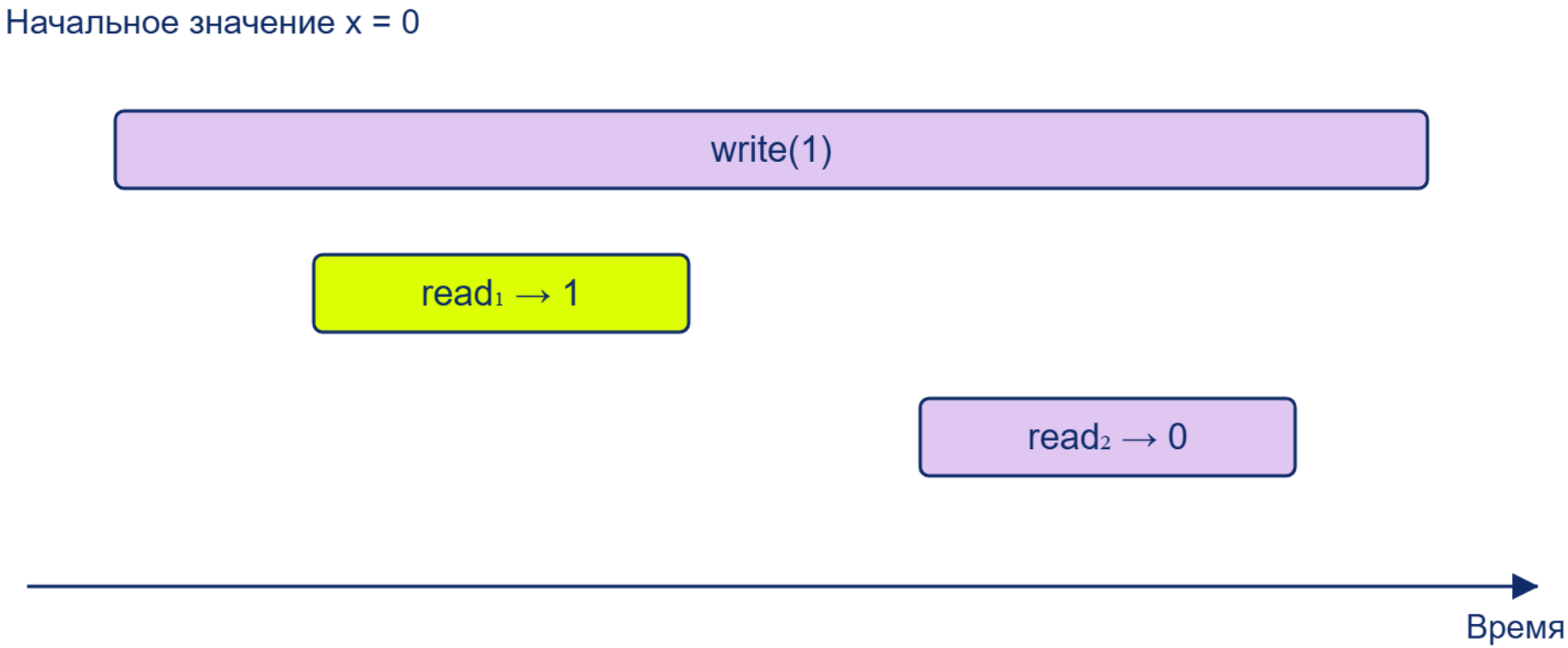
Чтение во время записи



Для показанной истории допустимы оба результата чтения:

- **0** – чтение расположено до записи
- **1** – запись расположена до чтения

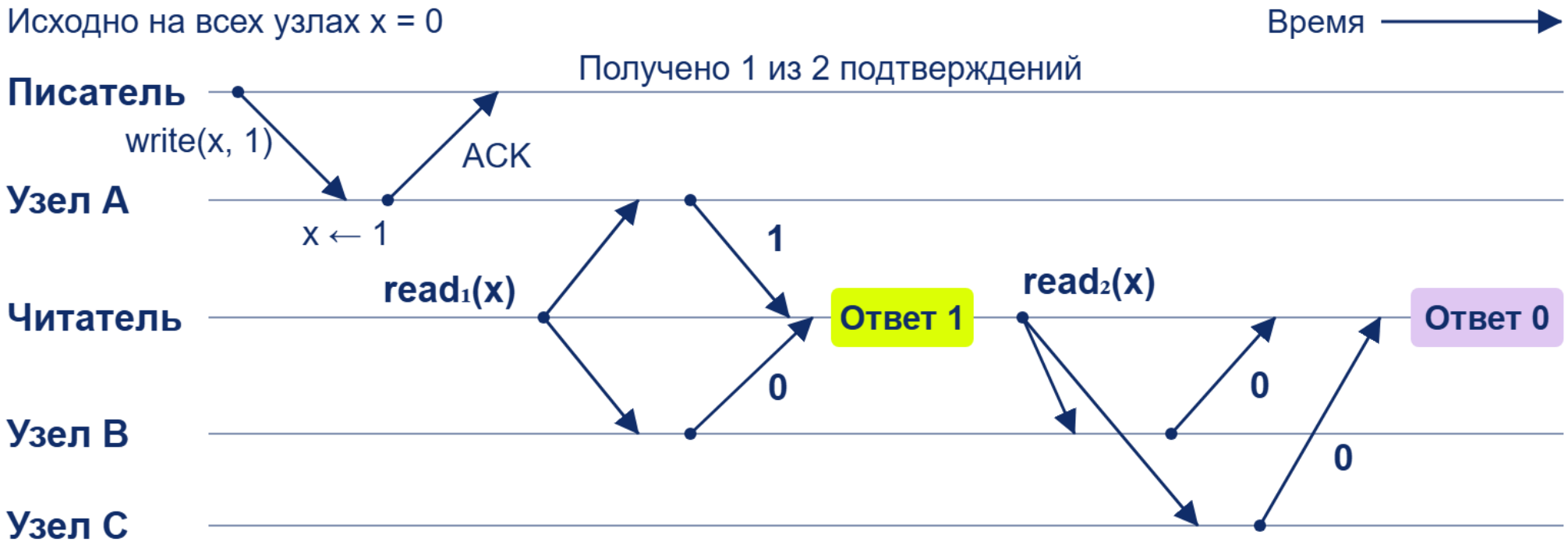
Можно ли снова увидеть старое значение?



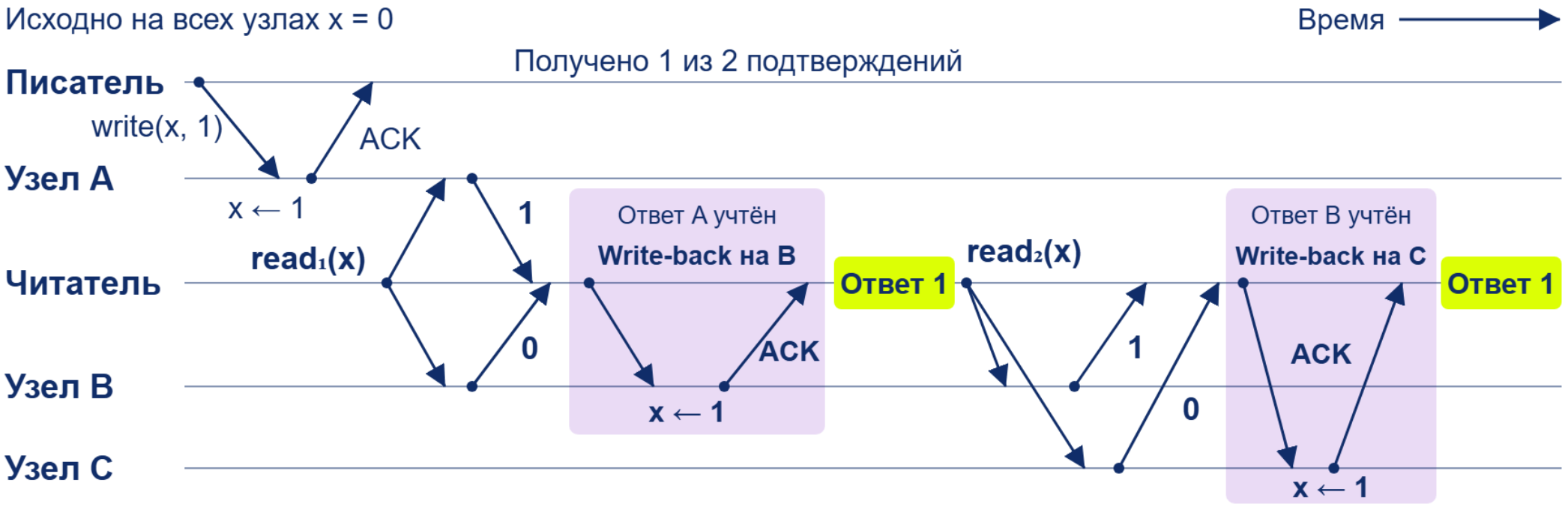
Требуемый порядок **write(1) → read$_1$ → read$_2$** исключает возврат 0 вторым чтением.

Нарушение при кворумном чтении

W + R > N, но завершённого кворума записи ещё нет.



Write-back перед ответом чтения



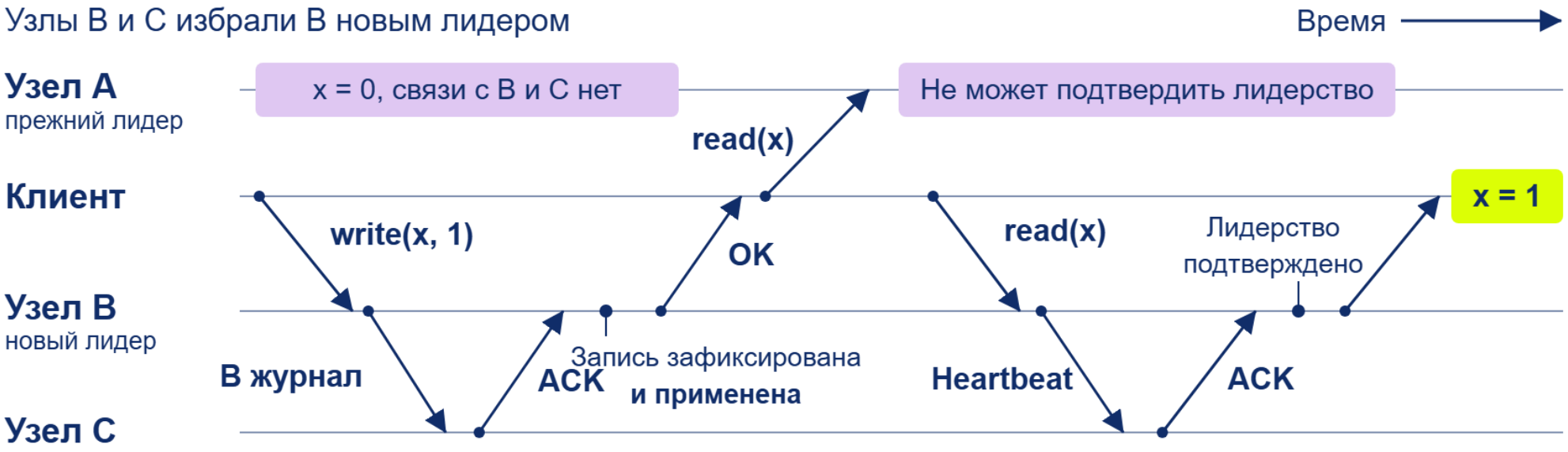
До ответа читатель должен получить подтверждения от **K** реплик, что они устойчиво хранят версию с меткой не ниже **t**, где **t** – наибольшая метка среди ответов на это чтение.

Условие **K + R > N** гарантирует пересечение с любым кворумом чтения размера **R**.

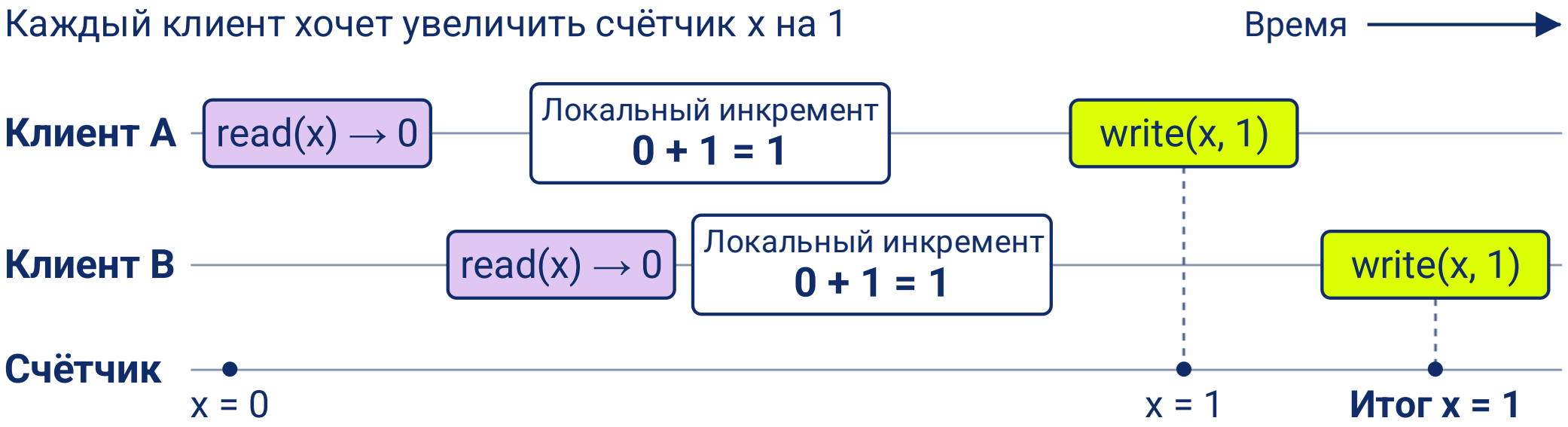
Лидер и большинство

Протокол репликации с лидером (например, Raft) может обеспечивать линеаризуемость без ожидания всех реплик.

- Запись фиксируется по правилам протокола с большинством.
- Новый лидер сохраняет необходимые решения.
- Чтение проверяет актуальность лидерства и применённого состояния.



Линеаризуемость и потерянное обновление

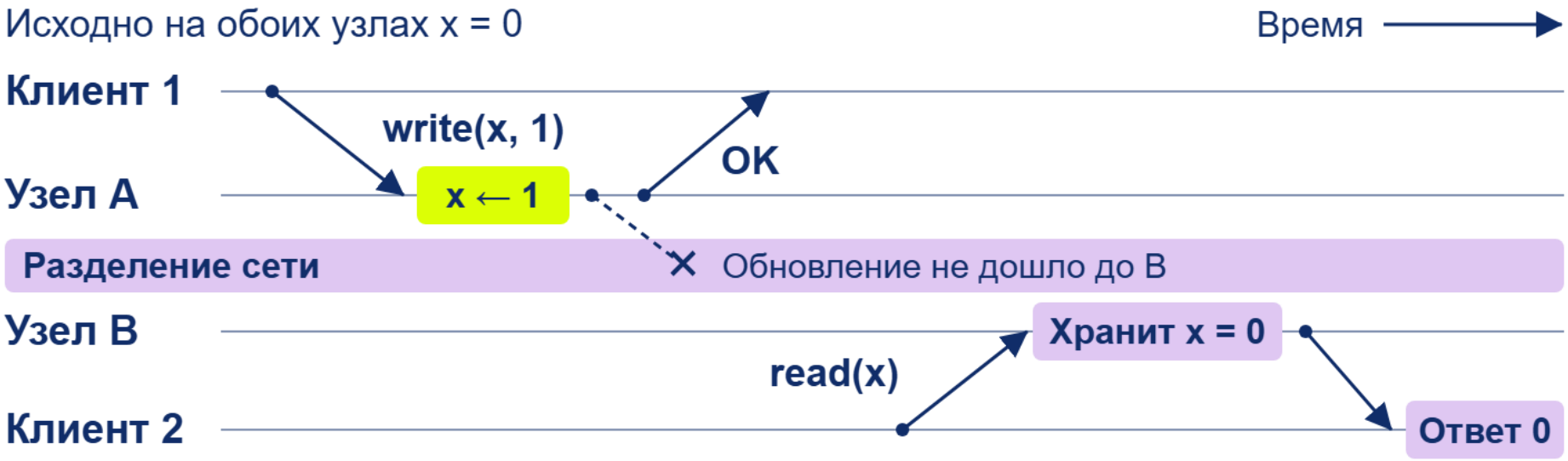


Даже при линеаризуемых **read** и **write** оба клиента могут прочитать одно значение до того, как первый из них запишет результат.

Два успешных вызова **атомарной** операции **increment(x)** увеличили бы счётчик с 0 до 2.

Разделение сети и гарантии операций

Связь между A и B пропала. Может ли B гарантированно ответить на чтение и при этом сохранить линеаризуемость?



Consistency (C) – линеаризуемость (согласованность данных)

Availability (A) – запрос к неотказавшему узлу в конце концов получает результат операции

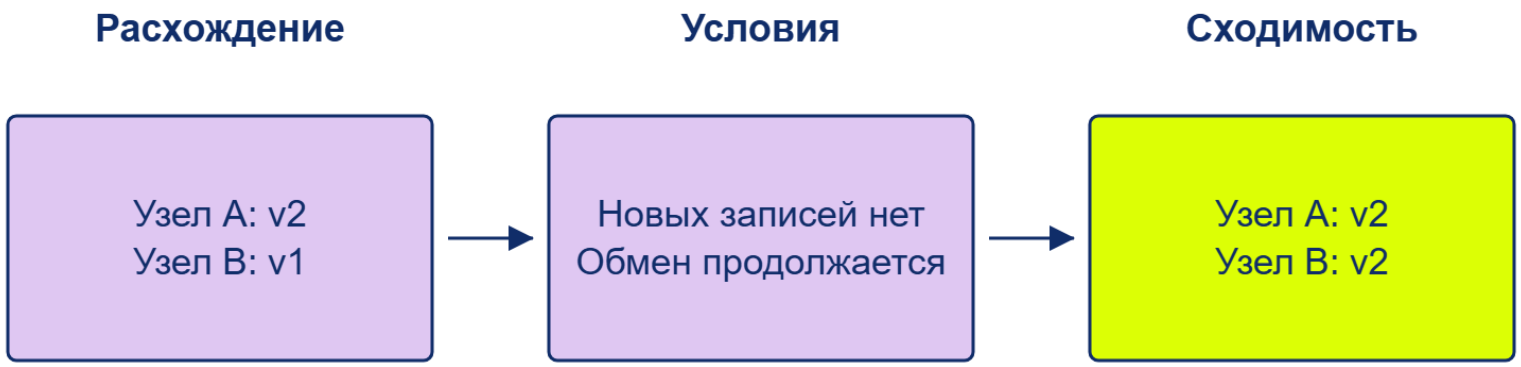
Partition Tolerance (P) – допускается разделение сети

Что выбирать при разделении сети?

Приоритет	**Возможное поведение**
Сохранить линеаризуемость	Часть запросов ждёт или не обслуживается
Продолжать чтения и записи на обеих сторонах	Допускаются истории, нарушающие линеаризуемость

После восстановления связи всё равно нужен протокол восстановления состояния.

Eventual consistency



Пример: v2 заменяет v1. Фиксированный срок не обещан.

Eventual consistency: после прекращения обновлений реплики в итоге сходятся, если обмен продолжается. (Срок сходимости не задан, промежуточные чтения могут различаться)

Strong eventual consistency (SEC) требует, чтобы:

- обновления неотказавших реплик в итоге учитывались всеми неотказавшими репликами;
- одинаковые наборы учтённых обновлений давали эквивалентные состояния;
- операции неотказавших реплик завершались.

Сравнение подходов к репликации

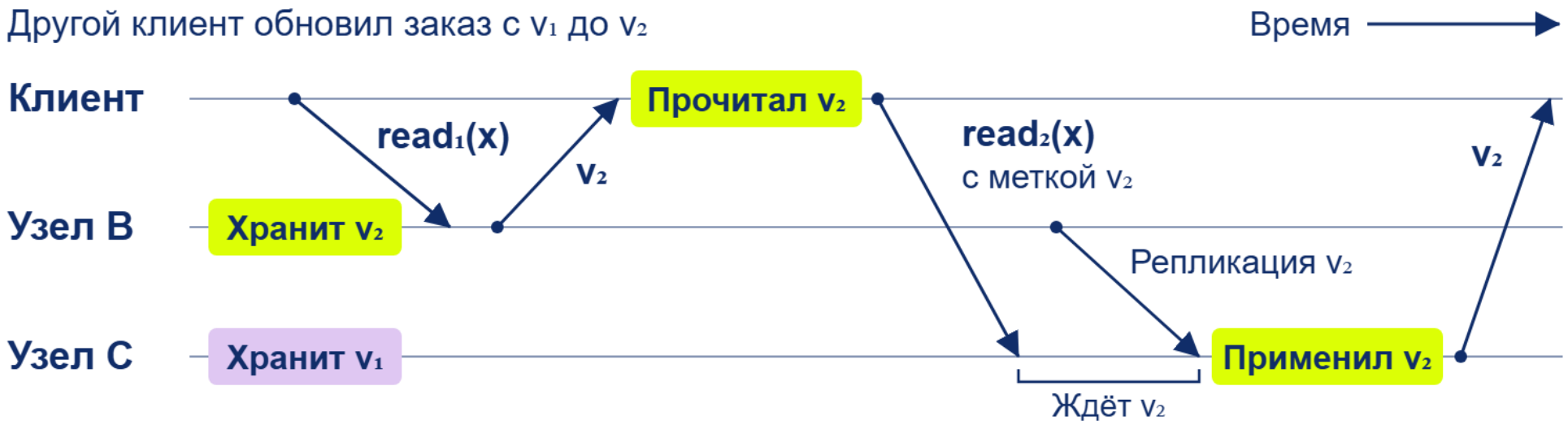
Подход	Организация записи	Основной вопрос
Один лидер (PostgreSQL)	Один порядок записей на шард	ACK, отставание, безопасная смена лидера
Несколько лидеров (CouchDB)	Несколько принимающих сторон	Конкурентные версии и зависимости
Без лидеров (Cassandra)	Координатор и ответы реплик	Кворумы, версии, восстановление реплик

SMR описывает порядок воспроизведения команд и может использовать лидера.

Модель согласованности определяется всем протоколом.

Гарантии внутри клиентской сессии

Гарантия сессии	Что должно учитывать следующее чтение
Read-your-writes	Завершённые записи этого клиента
Monotonic reads	Записи, эффект которых клиент уже наблюдал



Итоги

- **Подтверждение записи.** После OK от лидера отстающая реплика ещё может вернуть старый статус заказа.
- **Лидерство.** Новый лидер сохраняет зафиксированные записи. Для линеаризуемого чтения он проверяет свои полномочия и применяет записи, зафиксированные к началу чтения.
- **Кворумы.** Одного пересечения кворумов для линеаризуемости недостаточно. Важно, как протокол выбирает и распространяет версии.
- **Согласованность.** При линеаризуемости клиент видит одну логическую копию заказа. При eventual consistency статусы на репликах могут временно различаться.
- **Цена гарантии.** При разделении сети нельзя гарантировать одновременно линеаризуемость и доступность на каждом исправном узле.

Следующая лекция: как отслеживать порядок событий, учитывать причинные зависимости, сравнивать версии и сливать конкурентные изменения.

Материалы к лекции

Репликация

Designing Data-Intensive Applications
Replication, гл. 5 в 1-м изд. и гл. 6 во 2-м изд.

Distributed Systems
Кембриджский курс, гл. 5 и §7.2–7.3

Database Internals
Гл. 11. Репликация и согласованность

A Primer on Database Replication
Краткий обзор схем с лидерами и без них

Dynamo
Кворумы, версии и восстановление реплик

PostgreSQL и Cassandra
Подтверждение COMMIT и read repair

Согласованность

Consistency Models
Карта моделей согласованности от Jepsen

The Many Faces of Consistency
Что понимают под согласованностью

Eventually Consistent
Что видит клиент, пока обновление в пути

Session Guarantees
Что гарантирует сессия при смене реплики

A Critique of the CAP Theorem
Сетевые разделения, доступность и задержка

MongoDB stale reads
Исследование Jepsen для MongoDB 2.6.7