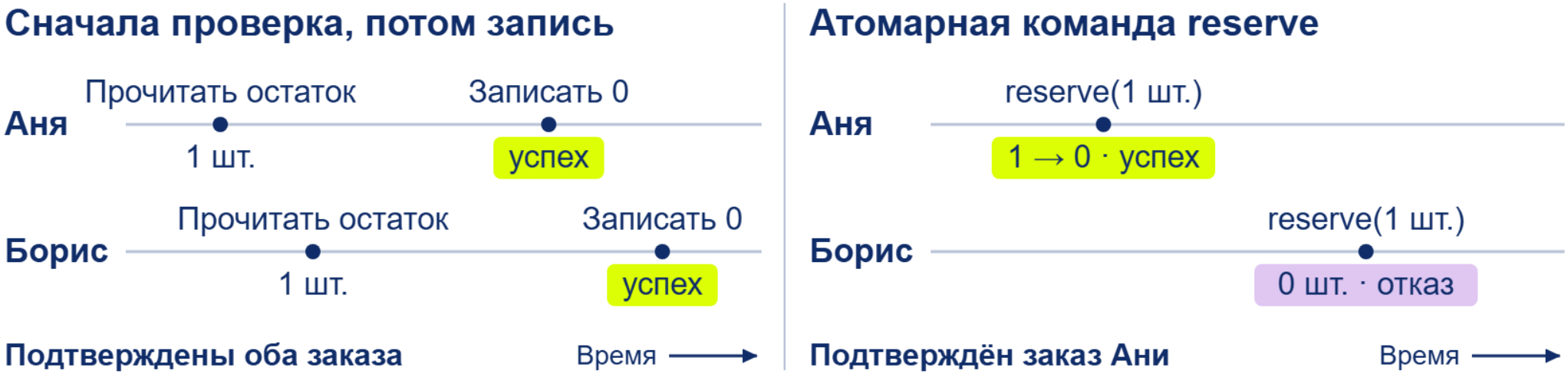
# 11. Консенсус и связанные задачи

Сухорослов Олег Викторович

23.11.2026

# Два заказа на последнюю кофемашину

На складе осталась одна кофемашина. Аня и Борис одновременно оформляют заказы.



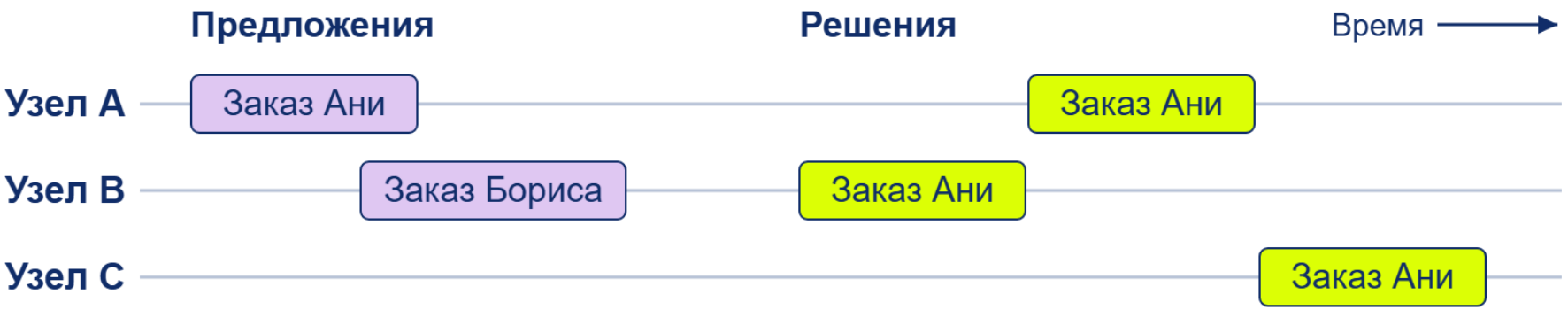

**Как определить, кому достанется товар, если заказы поступили на разные реплики?**

# План лекции

- Консенсус, атомарная рассылка и регистр с CAS
- Теорема FLP и условия завершения консенсуса
- Репликация журнала и смена лидера в Raft
- Чтение и повторные запросы клиента
- Ограничения одной группы Raft и транзакции между шардами
- Сервисы координации, согласованность и задержка операций

# Консенсус для одного решения

Узлы предлагают значения. Требуется выбрать одно из них.



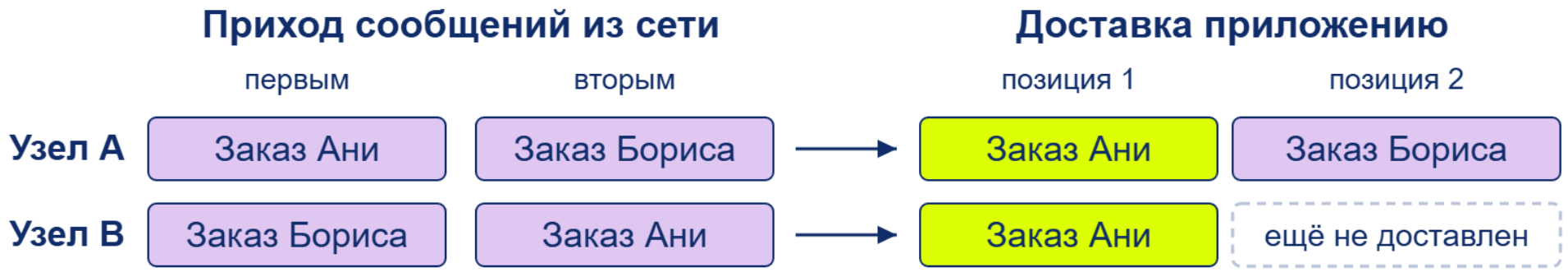

Требуемые свойства:

- **Допустимость.** Выбранное значение предложил хотя бы один узел.
- **Согласие.** Любые два узла, принявшие решение, выбрали одно и то же значение.
- **Однократность.** Каждый узел принимает решение не более одного раза.
- **Завершение.** Каждый корректный узел в конце концов принимает решение.

Допустимость, согласие и однократность – свойства *безопасности (safety)*.
Завершение – свойство *живости (liveness)*.

# Консенсус и атомарная рассылка

При **атомарной рассылке** все узлы доставляют сообщения приложению в одном порядке. Узел может отставать, но уже доставленные сообщения образуют префикс общей последовательности.




**От рассылки к консенсусу:** Разослать предложения и выбрать первое доставленное.

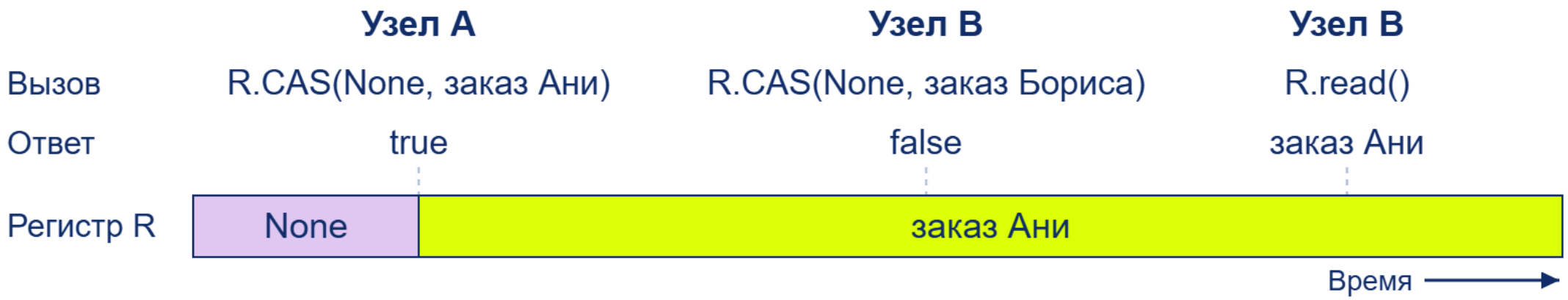
**От консенсуса к рассылке:** Последовательно согласовывать содержимое каждой позиции журнала и доставлять сообщения в порядке позиций.

# Консенсус с помощью compare-and-set

Для одного решения возьмём общий линеаризуемый регистр **R**. Изначально R = None.

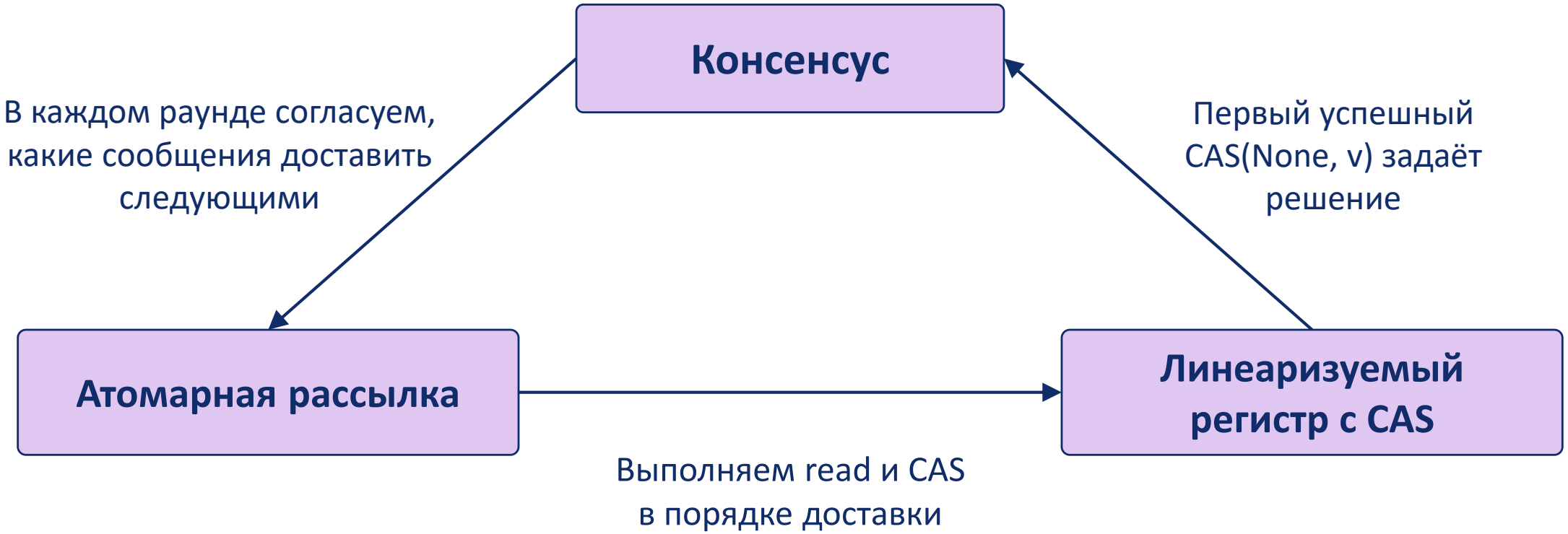
Каждый узел со своим предложением **v** вызывает атомарную операцию **R.CAS(None, v)**:

- **true** – узел записал **v** и принимает его как решение.
- **false** – значение уже выбрано. Узел вызывает **R.read()** и принимает прочитанное значение.



# Три эквивалентные задачи

Консенсус, атомарная рассылка и линеаризуемый регистр с CAS **взаимно реализуемы**.



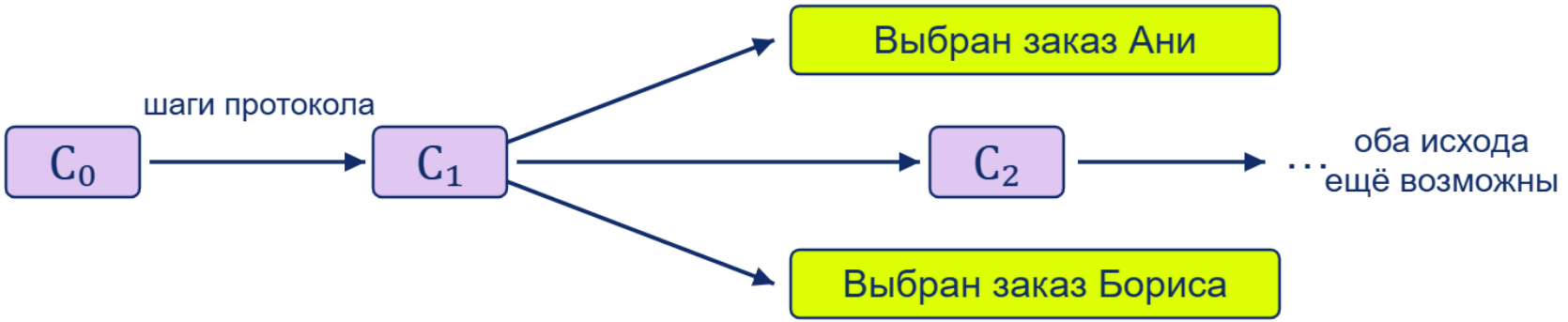
# Достижимость консенсуса

Предположения о системе:

- **Асинхронность.** Задержки сообщений и время между шагами узлов не ограничены сверху.
- **Надёжная доставка.** Сообщения исправным узлам в конце концов доставляются.
- **Отказы crash-stop.** Допускается остановка одного узла навсегда.

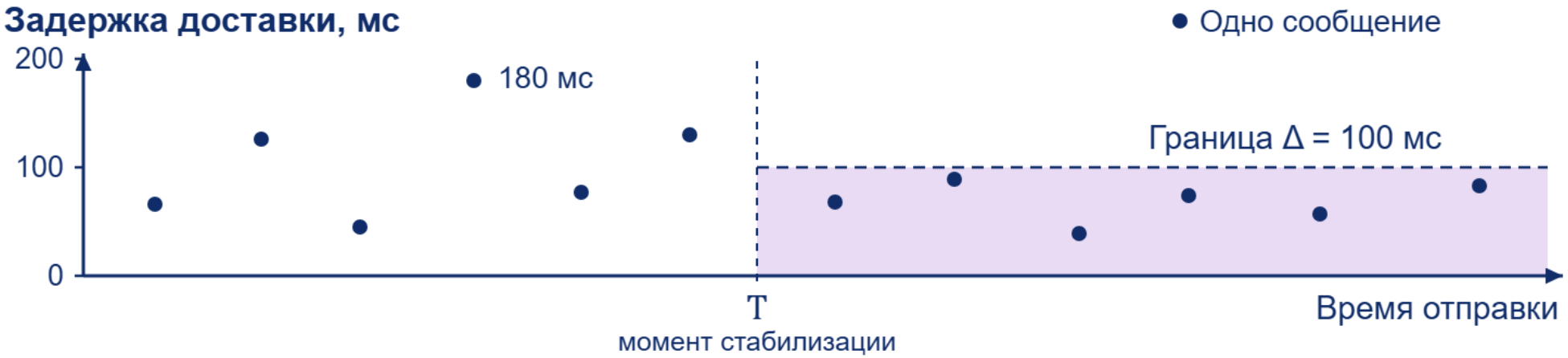
**Теорема FLP.** В асинхронной системе, допускающей отказ crash-stop одного узла, ни один детерминированный протокол консенсуса не может гарантировать одновременно безопасность и завершение во всех допустимых выполнениях.

Фишер, Линч и Патерсон, 1985



# Частичная синхронность

- До некоторого момента стабилизации $T$: задержки сообщений и интервалы между шагами узлов могут быть сколь угодно большими.
- После $T$: задержки сообщений между исправными узлами и время между их шагами ограничены.
- Узлы заранее не знают, когда наступит $T$.



# Как добиться завершения?

FLP рассматривает **детерминированный протокол** в **полностью асинхронной системе**. Существующие подходы выходят за рамки этих предположений.

| **Подход** | **Что предполагаем** | **Как завершается протокол** |
|---|---|---|
| **Частичная синхронность** | После стабилизации у исправных узлов ограничены задержки сообщений и время между шагами. | Если протокол выбирает общего исправного лидера, тот успевает получить ответы большинства и принять решение. |
| **Рандомизированный консенсус** | Задержки могут быть сколь угодно велики. Узлы могут делать случайный выбор. | При исправном большинстве протокол Бен-Ора завершается с вероятностью 1. |

В обоих случаях сохраняется безопасность (допустимость, согласие и однократность).

# Алгоритмы консенсуса

- **Viewstamped Replication** (Oki, Liskov, 1988)
- **Paxos** (Lamport, 1989)
- **Zab** (Junqueira, Reed, Serafini, 2011)
- **Raft** (Ongaro, Ousterhout, 2013)

Многие алгоритмы изначально спроектированы под консенсус не для одного значения, а для последовательности значений, по сути реализуя **репликацию журнала** или **атомарную рассылку**.

# Репликация состояния через журнал

- **Команды хранятся в журнале.** Позиция записи называется индексом.
- Реплики применяют **зафиксированные команды** по порядку индексов.
- Одинаковое начальное состояние и детерминированное исполнение дают одинаковое состояние после одного и того же префикса команд.



# Raft: предположения и гарантии

**Raft** – протокол консенсуса, согласующий состав и порядок команд в журнале.

**Предположения о системе:**

- Узлы могут останавливаться и перезапускаться. Данные на диске сохраняются (crash-recovery). Византийские отказы исключены.
- В сети возможны потери, повторы и перестановки сообщений. При неограниченных повторах сообщения между исправными узлами доходят (fair-loss links).
- Система является частично синхронной.

**Гарантии:**

- **Безопасность.** Если в позиции журнала зафиксирована команда, ни один узел не зафиксирует в этой позиции другую команду.
- **Завершение.** После стабилизации лидер может фиксировать новые записи, если успевает обмениваться сообщениями с исправным большинством узлов.

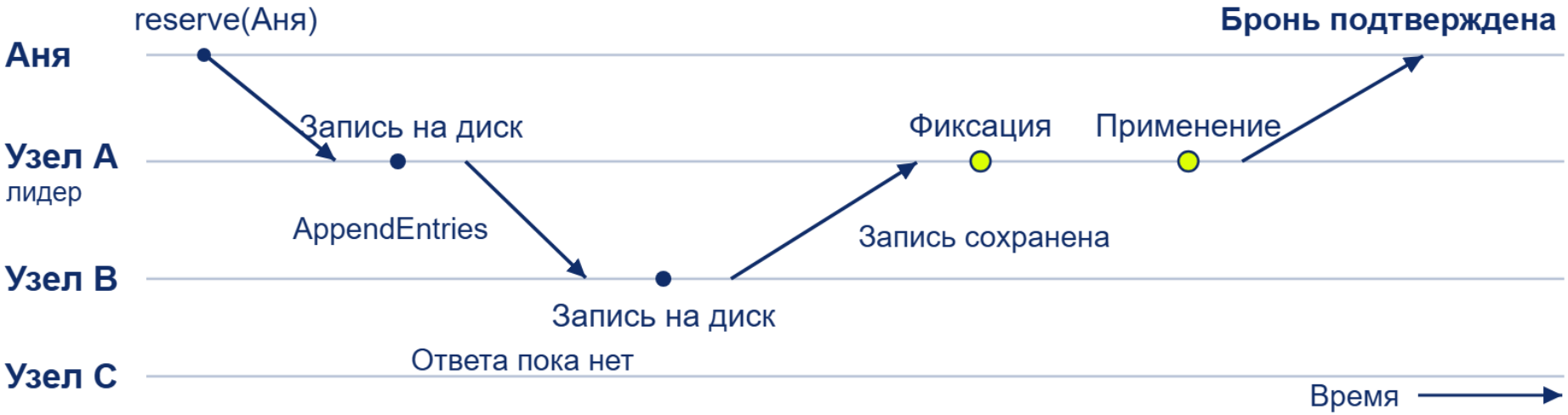
# Лидер и подчинённые

- Выполнение разбито на пронумерованные эпохи. Каждая эпоха начинается с выборов. В одной эпохе может быть выбрано не более одного лидера.
- Лидер задаёт порядок команд клиентов и рассылает записи подчинённым.
- Если подчинённый до истечения тайм-аута не получает сообщений от лидера, он становится кандидатом и начинает выборы в новой эпохе.



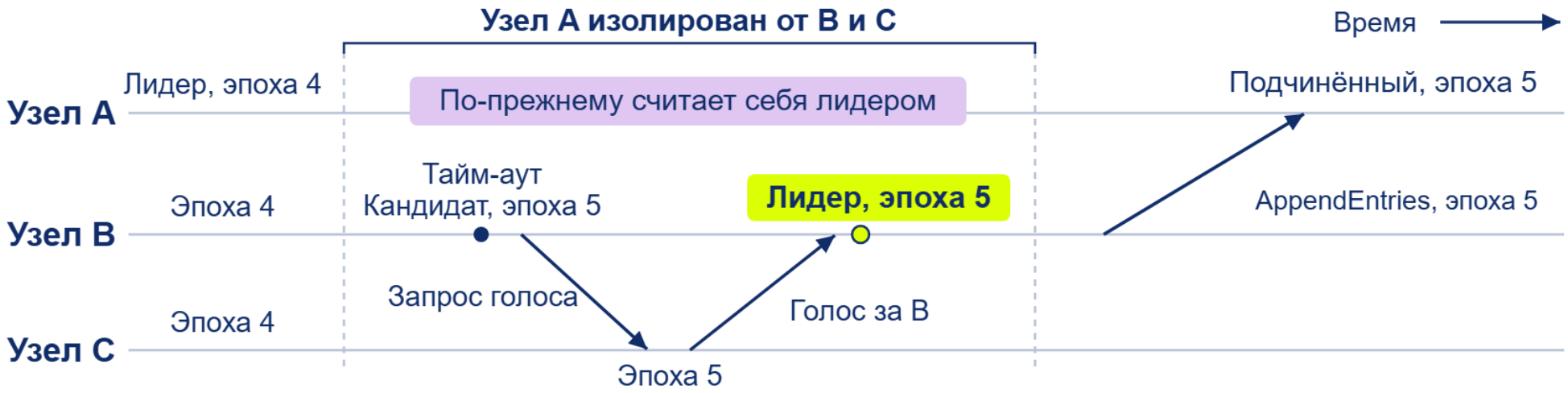
# Когда лидер может подтвердить бронь?

- Лидер A сохраняет новую команду в своём журнале и отправляет её узлам B и C в сообщении *AppendEntries*.
- Узел B сохраняет запись на диске и отвечает A. Теперь лидер знает, что запись его текущей эпохи хранится на большинстве узлов. Лидер **фиксирует запись**.
- Лидер **применяет команду** к состоянию магазина и возвращает результат Ане. Ждать ответа C не требуется.



# Смена лидера в Raft

- По тайм-ауту выборов подчинённый становится **кандидатом**, увеличивает номер эпохи, голосует за себя и запрашивает голоса остальных.
- Для победы нужно **большинство голосов в своей эпохе**. Если голоса разделились, кандидаты выбирают новые случайные тайм-ауты и повторяют выборы.
- Узел отклоняет запросы из прошлых эпох. Лидер, получив запрос или ответ с большей эпохой, становится подчинённым.



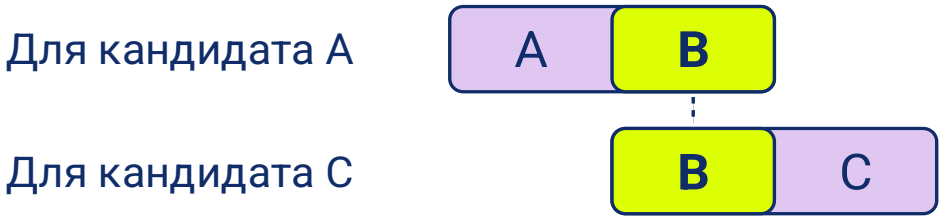
# Зачем Raft нужно большинство?

В Raft кворум – большинство голосующих узлов.

- **Выборы в одной эпохе.** Любые два кворума пересекаются. Узел голосует не более одного раза за эпоху (сохраняя на диске), поэтому двух победителей быть не может.
- **Запись и новые выборы.** Их кворумы пересекаются. Среди голосующих есть узел, подтвердивший запись.
- Для работы при **f** отказавших узлах нужно не менее **2f + 1** узлов.

**Выборы в одной эпохе**

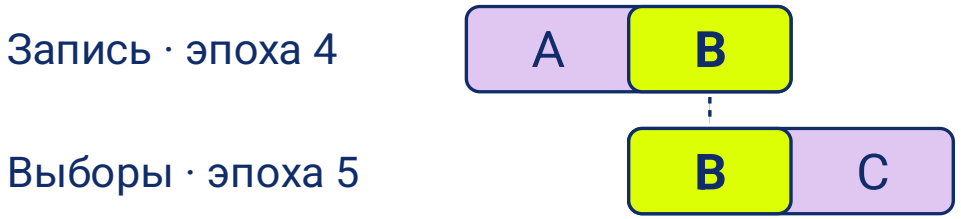
Эпоха 5 · возможные кворумы




B не может голосовать за обоих

**Запись и новые выборы**

Бронь Ани · затем выборы C




B хранит запись и проверяет журнал C

# Сравнение журналов при выборах

$(t_k, i_k)$ – эпоха и индекс последней записи в журнале узла $k$.

Узел $v$ отдает голос за кандидата $c$ только если $t_c > t_v \ \vee (t_c = t_v \ \wedge \ i_c \geq i_v)$.

| Индекс записи | 1 | 2 | 3 | 4 | 5 | Эпоха, индекс | В отдаст голос? |
|---|---|---|---|---|---|---|---|
| **Голосующий B** | 1 | 1 | 2 | 2 | | (2, 4) | Журнал для сравнения |
| Кандидат A | 1 | 1 | 1 | 1 | 1 | (1, 5) | Нет · эпоха меньше |
| Кандидат C | 1 | 1 | 2 | | | (2, 3) | Нет · индекс меньше |
| Кандидат D | 1 | 1 | 3 | | | (3, 3) | Да · эпоха больше |

Число в ячейке — эпоха, в которой появилась запись.

Лидером может стать только узел, в журнале которого есть все ранее зафиксированные записи.

# Как лидер исправляет журнал реплики

В *AppendEntries* лидер указывает **индекс и эпоху предыдущей записи**.

- Если запись не совпадает, реплика отклоняет запрос. Лидер повторяет его с более ранней позиции.
- Когда общая позиция найдена, реплика принимает записи лидера. Она заменяет незафиксированный хвост своего журнала начиная с первого конфликта.



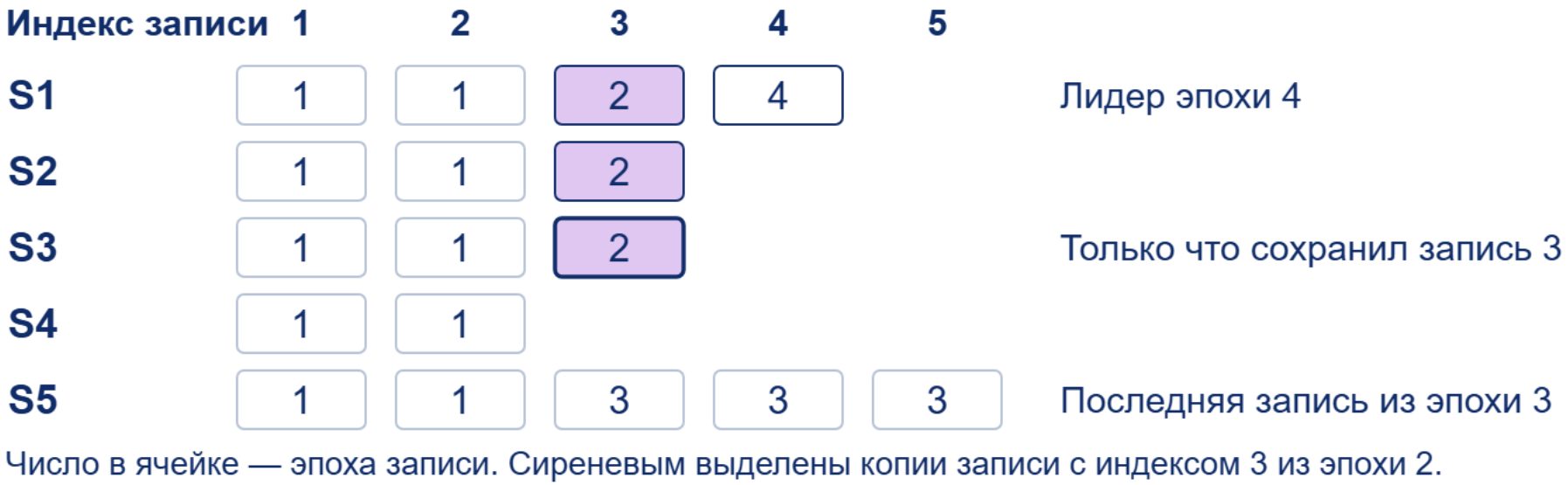

**Проверка предыдущей записи**

**1. Индекс 3**
У D эпоха 3, у B эпоха 2. Отказ.

**2. Индекс 2**
Эпоха 1 у обоих. B принимает запись 3.

# Достаточно ли трёх копий?

Лидер S1 в эпохе 4 унаследовал незафиксированную запись из эпохи 2 с индексом 3. После того как S1 передал эту запись на S3, она хранится на большинстве узлов.




Можно ли подтвердить эту запись клиенту? Что произойдёт, если S1 остановится?

# Правило фиксации в Raft

Лидер может продвинуть **границу фиксации** *commitIndex* до индекса *N*, если запись *N* создана в его текущей эпохе и сохранена на большинстве узлов, включая лидера.

Вместе с ней фиксируются **все записи до *N* включительно**, в т.ч. из прошлых эпох.




Число в ячейке — эпоха записи. Теперь у лидера зафиксированы записи с индексами 1–4.

# Линеаризуемое чтение

Аня получила подтверждение брони **до того, как** Борис запросил остаток.

Значит, чтение должно учесть бронь и вернуть **0**.



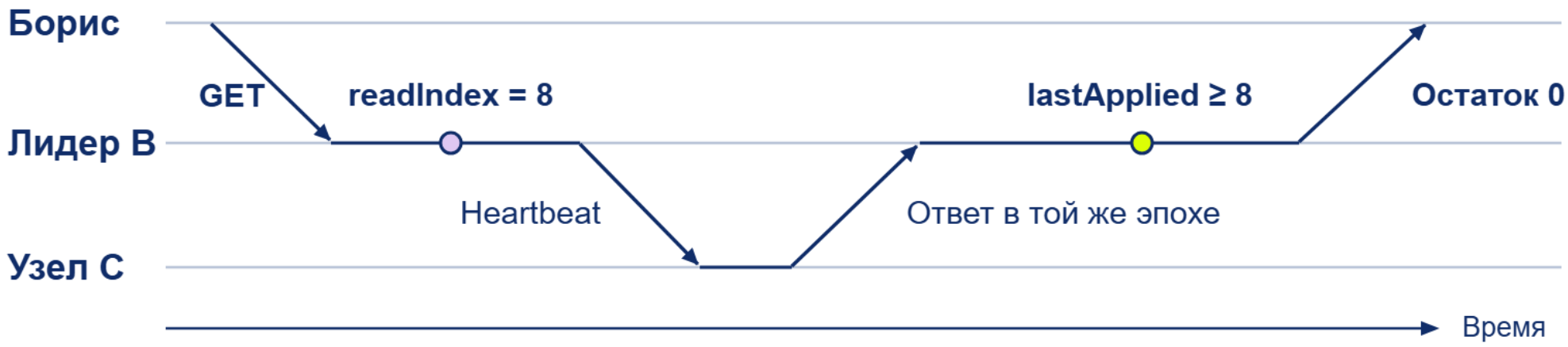

**Чтение через журнал.** Лидер добавляет GET в журнал и отвечает клиенту после того, как зафиксирует и применит эту запись.

# Чтение без записи в журнал

В текущей эпохе лидер уже должен зафиксировать хотя бы одну запись.

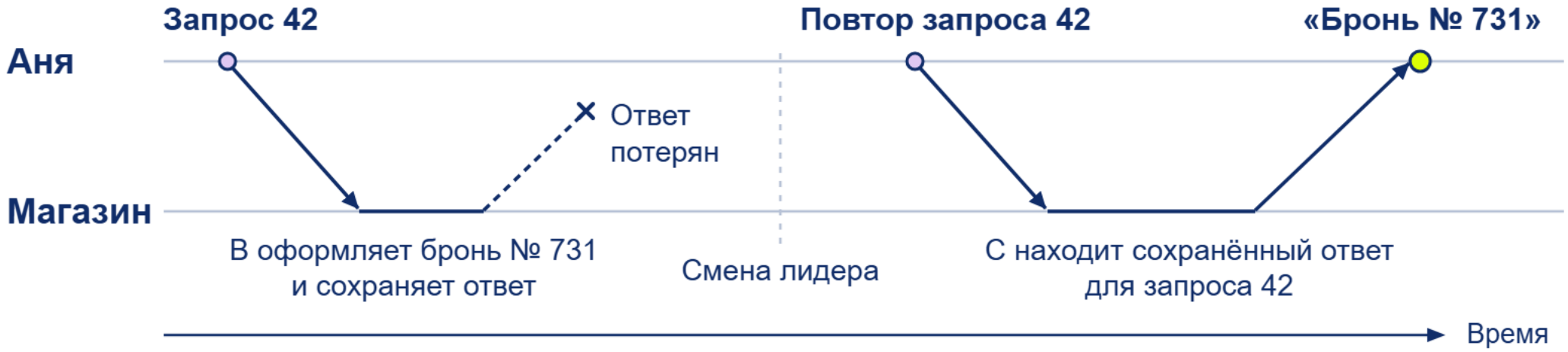
Получив GET, он:

- Запоминает текущую границу фиксации: $readIndex = commitIndex$.
- Подтверждает свои полномочия обменом сообщениями с большинством в текущей эпохе.
- Дожидается применения записей ($lastApplied \geq readIndex$) и читает локальное состояние.
- Возвращает результат клиенту.



# Повтор запроса после тайм-аута

**При тайм-ауте запроса клиент не знает, выполнена ли его команда.**
При повторе клиент отправляет тот же идентификатор запроса.



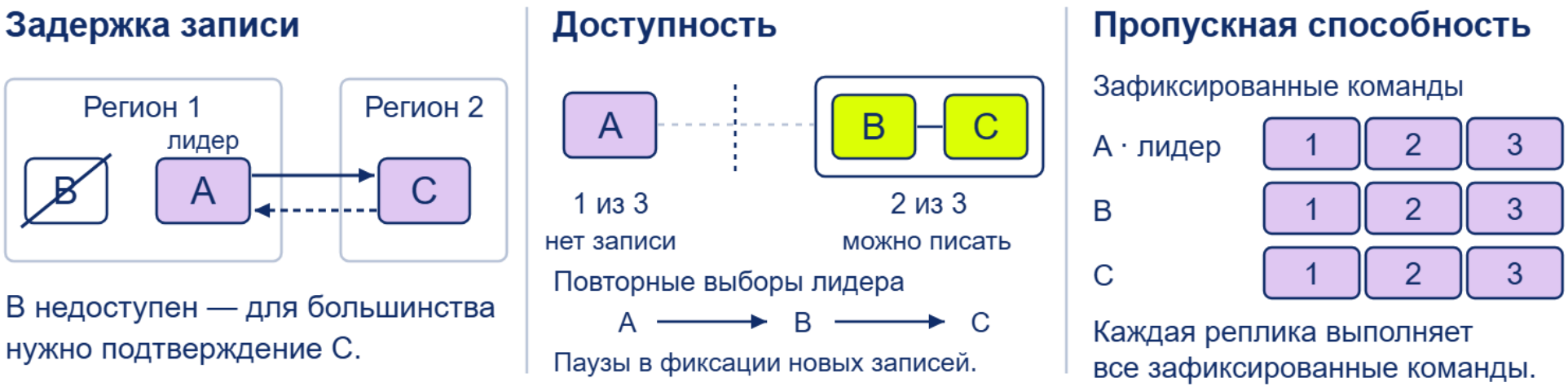

Вместе с бронью магазин сохраняет идентификатор запроса и ответ.
Если клиент повторит запрос с тем же идентификатором, магазин вернёт прежний ответ.

# Ограничения одной группы Raft

**Задержка записи.** Если для большинства нужна реплика в другом регионе, лидер ждёт её подтверждения.

**Доступность.** При разделении сети часть группы без большинства не может фиксировать новые записи. Нестабильная связь может вызывать повторные выборы и паузы в записи.

**Пропускная способность.** Один лидер упорядочивает команды, и каждая реплика применяет их все. Для горизонтального масштабирования записи нужны отдельные группы – шарды.



# Перевод между шардами

Аня оплачивает кофемашину. Её счёт и счёт магазина находятся на разных шардах. **Списание и зачисление должны составлять одну транзакцию:** либо выполняются обе операции, либо ни одна.



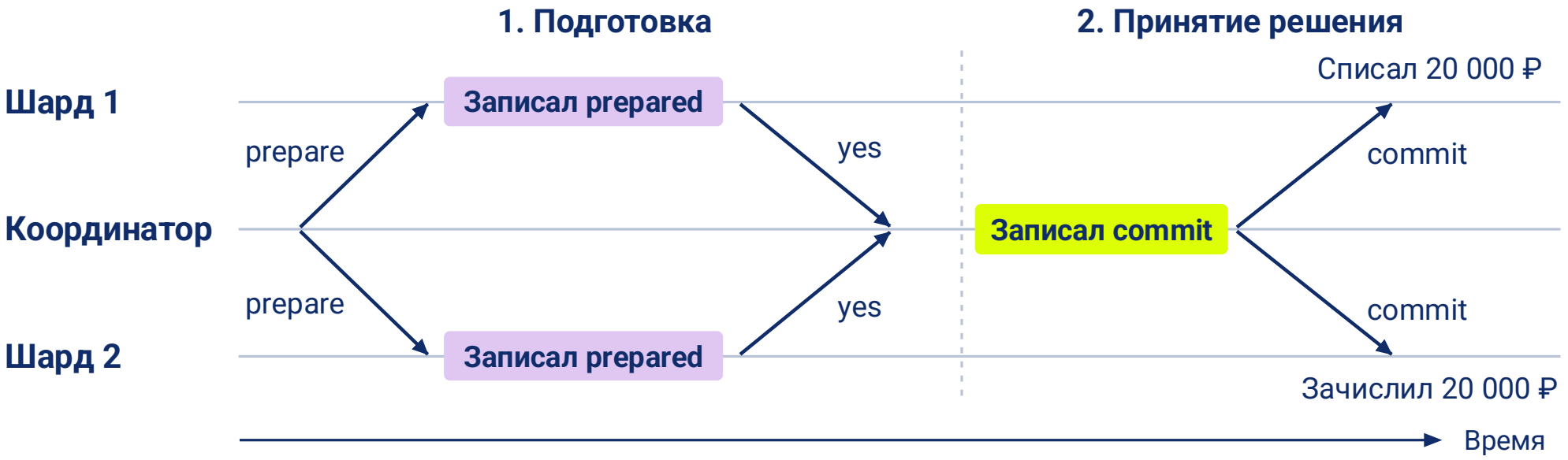

Шард 1 готов к списанию. Все три его реплики сохранили эту запись. Ответа шарда 2 пока нет.

Можно ли сообщить Ане, что оплата прошла?

# Двухфазная фиксация (2PC)

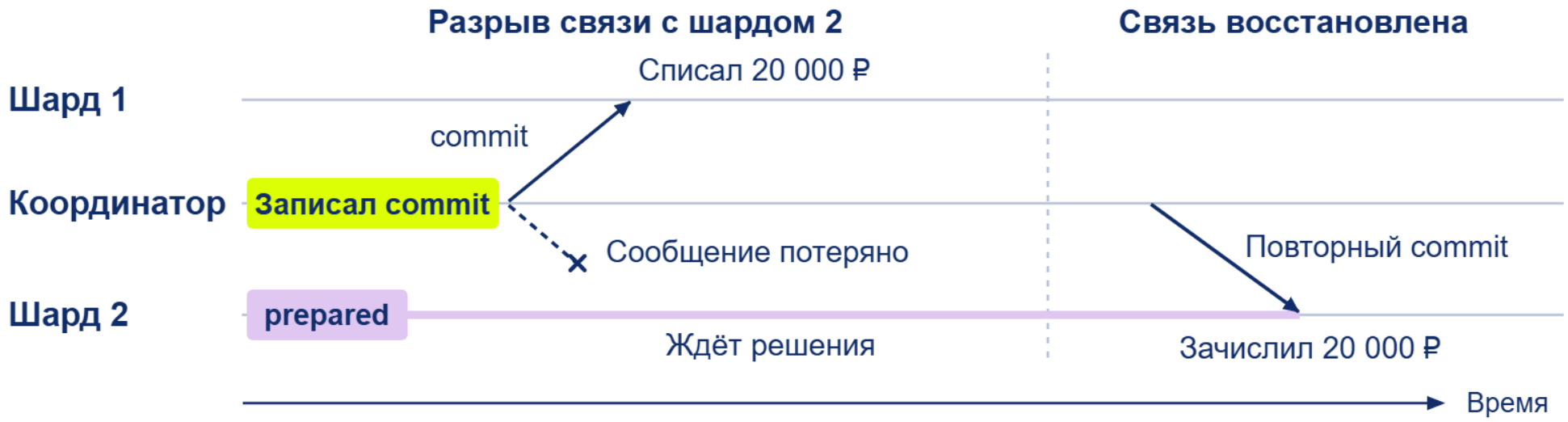
**1. Подготовка.** Координатор рассылает **prepare**. Каждый шард сохраняет состояние **prepared** в журнале и только потом отвечает **yes**. Если подготовиться не удалось, шард отвечает **no**.

**2. Принятие решения.** Получив **yes** от всех участников, координатор записывает **commit** в журнал и рассылает решение. Если хотя бы один участник ответил **no**, координатор отменяет транзакцию (**abort**).



# Если commit дошёл только до одного шарда

Координатор уже записал **commit**. Шард 1 получил решение и завершил списание.
Шард 2 всё ещё в состоянии **prepared**.

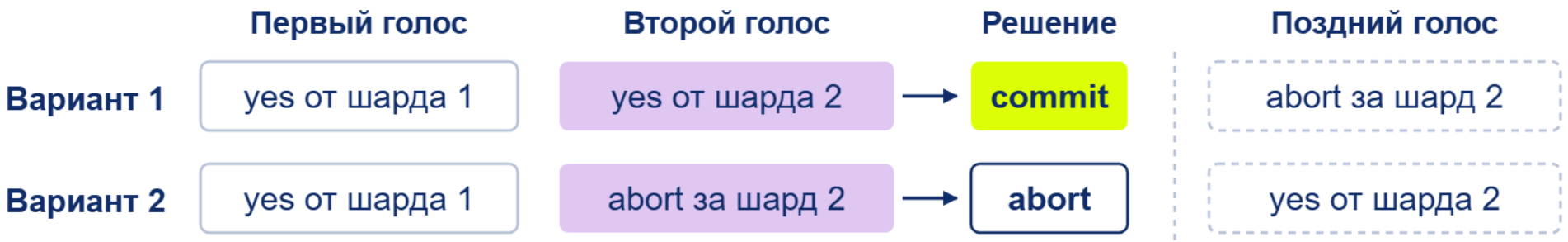



После ответа **yes** шард не может сам отменить транзакцию по тайм-ауту.
Он должен узнать её исход.

# Атомарная рассылка голосов

Атомарная рассылка доставляет голоса всем участникам в одном порядке.
Для каждого шарда учитывается только первый доставленный голос.

Если голос шарда задержался, другой участник может предложить за него **abort**.
Подтвердить готовность (**yes**) может только сам шард.




Фиксация транзакции (**commit**) возможна, когда от каждого участника учтён **yes**.

Для отмены достаточно одного учтённого **abort**.

# Перевод между шардами в Spanner

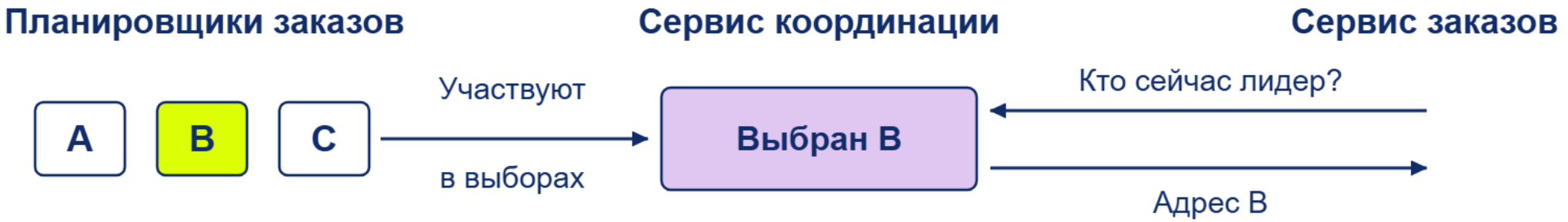
**Paxos внутри группы.** Большинство реплик подтверждает запись в журнале. Независимые группы работают параллельно.

**2PC между группами.** Лидеры участвующих групп согласуют исход перевода. Один из них выступает координатором.



# Сервисы координации

**Сервис координации** хранит общие служебные данные, с помощью которых процессы согласовывают свои действия (выбирают лидера, согласуют исход транзакций, публикуют адреса сервисов, управляют конфигурацией…).




Примеры:

- **etcd** хранит пары «ключ – значение» и реплицирует журнал с помощью **Raft**.
- **ZooKeeper** хранит данные в дереве узлов и упорядочивает изменения с помощью протокола атомарной рассылки **Zab**.

# Гарантии чтения в etcd и ZooKeeper

Лидером планировщиков стал C. Сервис координации подтвердил запись нового адреса. **Только после этого** сервис заказов запросил адрес лидера. Может ли он всё ещё получить адрес B?

| Сервис и вызов | Как читается запись | Адрес в ответе |
|---|---|---|
| etcd · Range по умолчанию | Линеаризуемое чтение | C |
| etcd · Range с serializable = true | Из локальной копии | B или C |
| ZooKeeper · getData | На сервере, принявшем запрос | B или C |

# Согласованность и задержка операций

Сравним время чтения и записи регистров в худшем случае. Пусть $d$ – задержка сообщения между репликами. Гарантированной границы расхождения их часов нет.

| **Модель согласованности и вариант реализации** | **Запись** | **Чтение** |
|---|---|---|
| Линеаризуемость | $O(d)$ | $O(d)$ |
| Последовательная, с локальным чтением | $O(d)$ | $O(1)$ |
| Последовательная, с локальной записью | $O(1)$ | $O(d)$ |
| Причинная согласованность | $O(1)$ | $O(1)$ |

$O(d)$ – время ответа в худшем случае растёт с задержкой между репликами.
$O(1)$ – верхняя граница времени ответа от этой задержки не зависит.

# Материалы к лекции

## Консенсус и фиксация транзакций

Клеппман. Distributed Systems, гл. 6 и § 7.1
Консенсус, атомарная рассылка и 2PC

Designing Data-Intensive Applications
«Согласованность и консенсус»
гл. 9 в 1-м издании, гл. 10 во 2-м

Ван Стен и Таненбаум. Distributed Systems, §§ 8.2, 8.5
Отказоустойчивость и распределённая фиксация

Онгаро и Оустерхаут. Статья о Raft, §§ 5 и 8
Репликация журнала, чтения и повторные запросы

Грей и Лэмпорт. Consensus on Transaction Commit
Почему 2PC может блокироваться

## Сравнения и практические разборы

Ховард и Мортье. Paxos vs Raft
Выборы лидера и сохранение журнала

Raft Does Not Guarantee Liveness in the Face of Network Faults
Почему при частичном отказе сети выборы могут повторяться

VK. «Raft (не)всемогущий»
Как устроены предварительные выборы в Tarantool

Google SRE. Managing Critical State, гл. 23
Консенсус в сервисах координации

Корбетт и соавторы. Статья о Spanner, §§ 2.1 и 4.2.1
Paxos внутри группы и 2PC между группами