8. Масштабирование

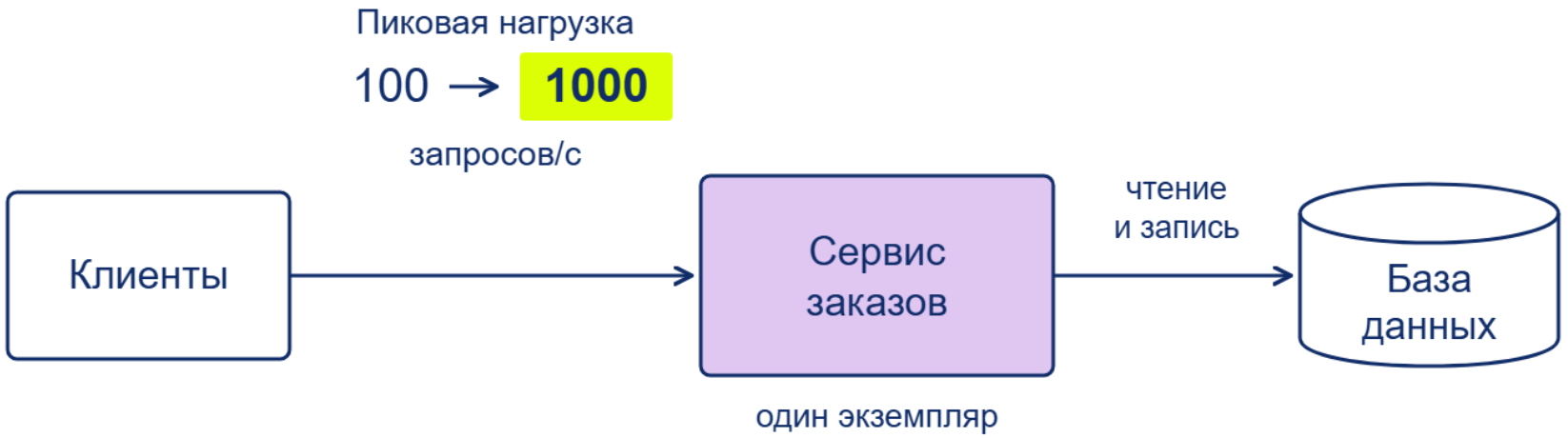
Сухорослов Олег Викторович

02.11.2026

Рост нагрузки на сервис

Запросов становится больше.

Требования к задержке и доле ошибок остаются прежними.



Что будет ограничивать масштабируемость?

Цель масштабирования

Пример требований к сервису заказов:

Показатель	**Требование**
Входящая нагрузка	1000 запросов/с
Задержка успешных ответов	p99 ≤ 200 мс
Ошибки и тайм-ауты	≤ 0,1% запросов

Эксперименты проводятся при одинаковых условиях (состав запросов, объём данных)

Вертикальное и горизонтальное масштабирование



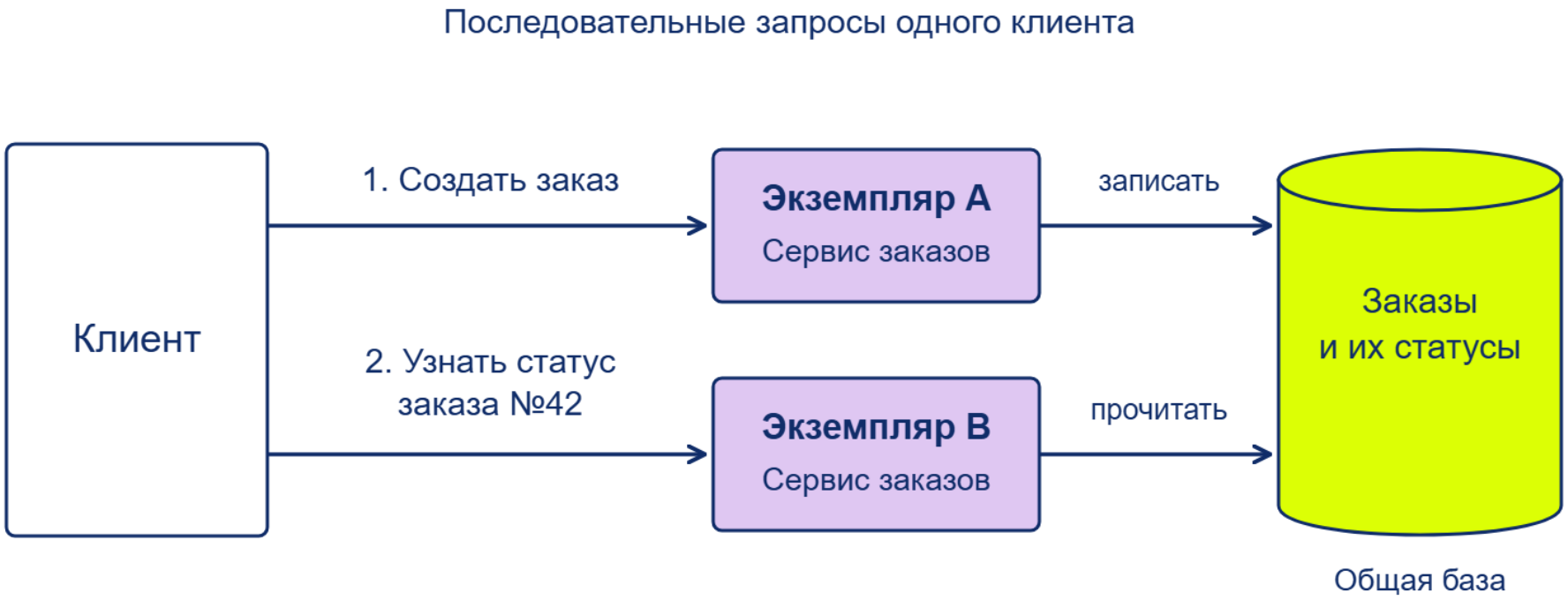
Общий компонент (база данных, хранилище) может ограничивать оба варианта.

План лекции

- Экземпляры сервиса и балансировка нагрузки
- Кэширование
- Шардинг и размещение данных
- Изменения в работающей системе
- Итоги и переход к репликации

Несколько экземпляров сервиса

Stateless: для обработки следующего запроса не требуется тот же экземпляр.



Состояние заказов хранится в общей базе данных.

Состояние пользовательской сессии

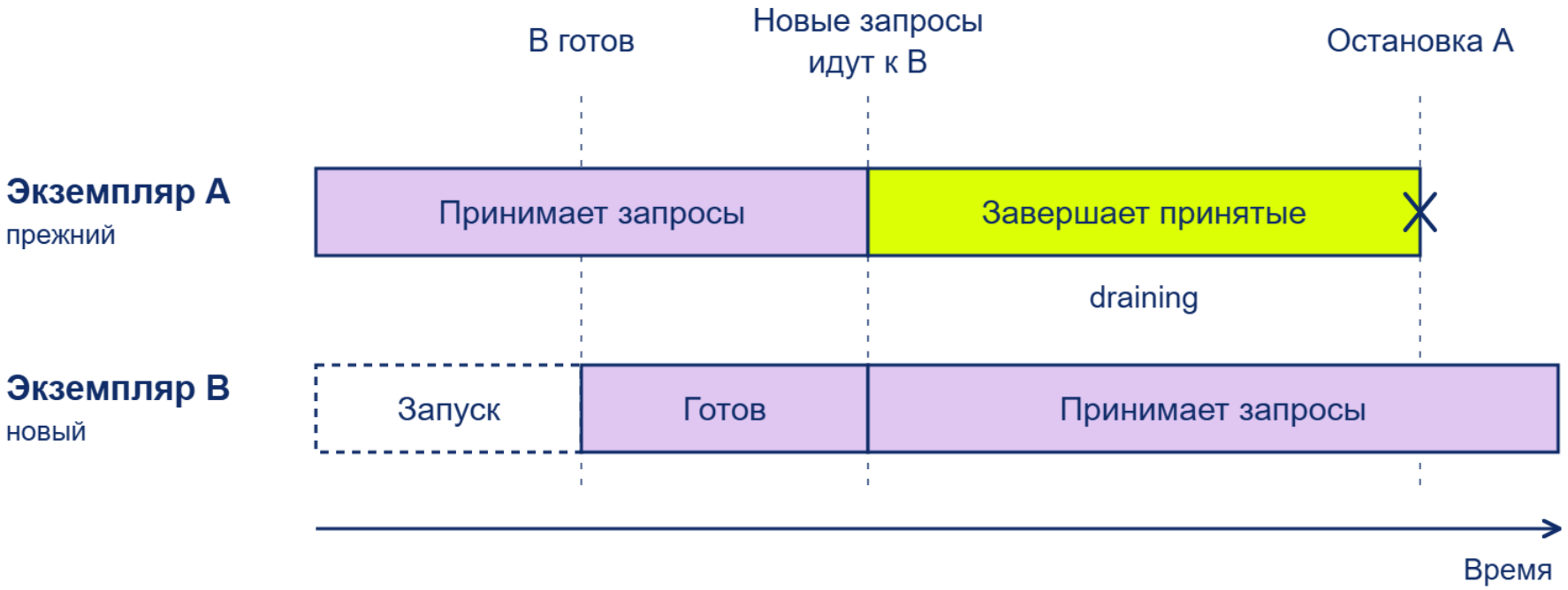
Cookie содержит идентификатор сессии, а её данные хранятся на сервере.



Общее хранилище снимает привязку сессии к экземпляру.

Штатная замена экземпляра

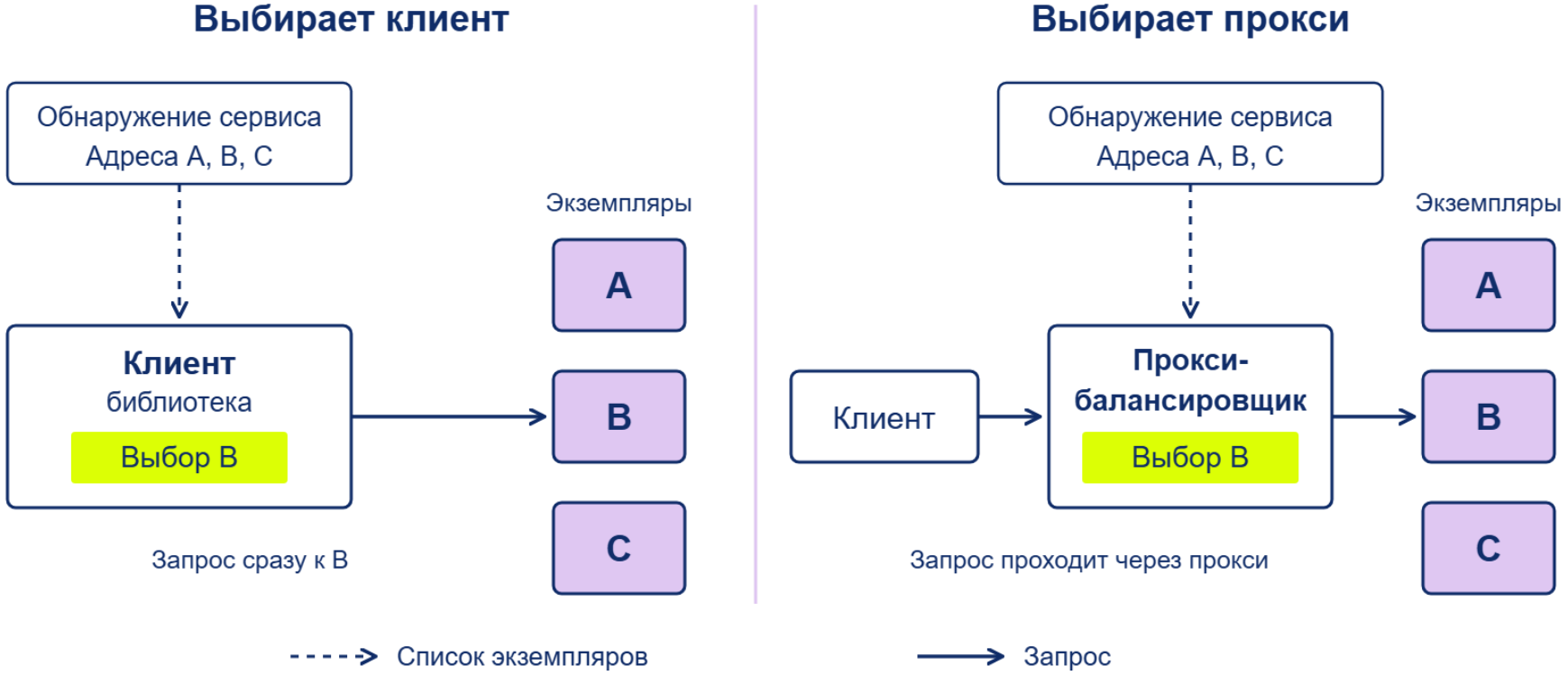
Новые запросы идут к B, принятые запросы завершаются на A.



Надо учесть распространение изменений клиентам и корректное завершение соединений.

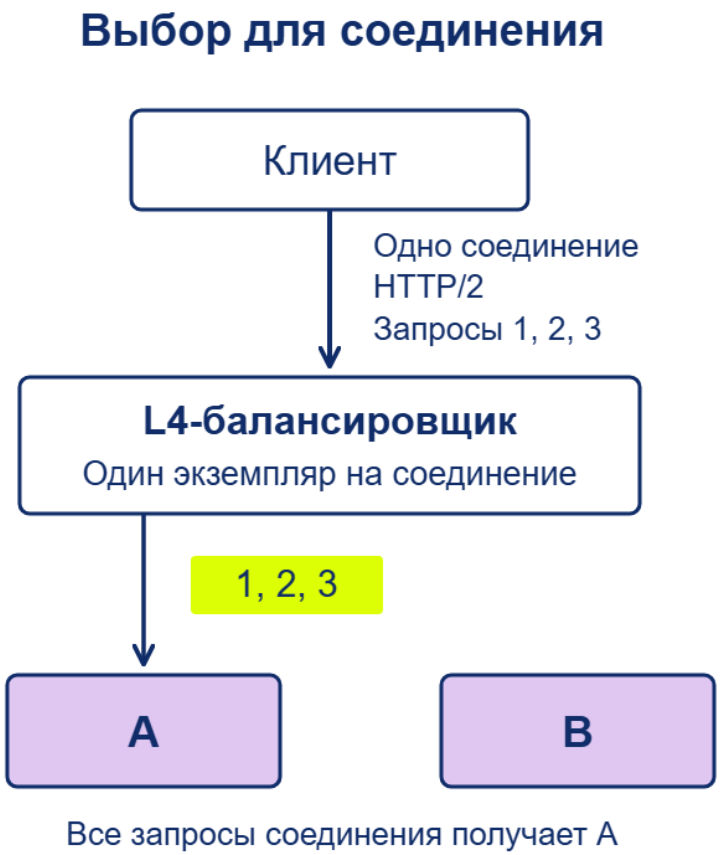
Выбор экземпляра сервиса

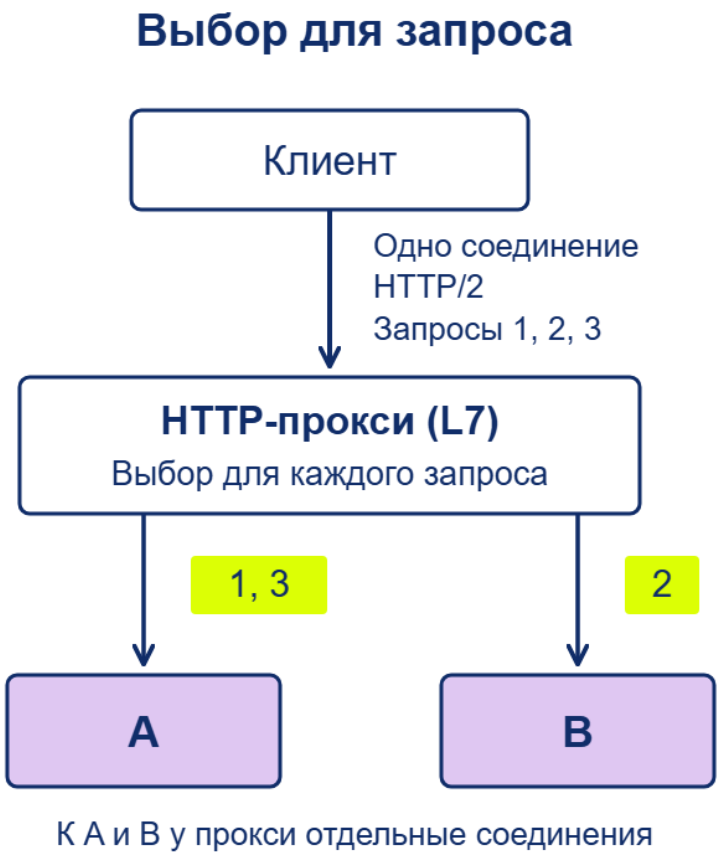
Обнаружение даёт список экземпляров. Балансировка выбирает один из них.



Сведения о составе и состоянии экземпляров могут запаздывать и устаревать.

Единица балансировки





Равное число соединений не означает равной нагрузки.

Соединения, запросы и очередь

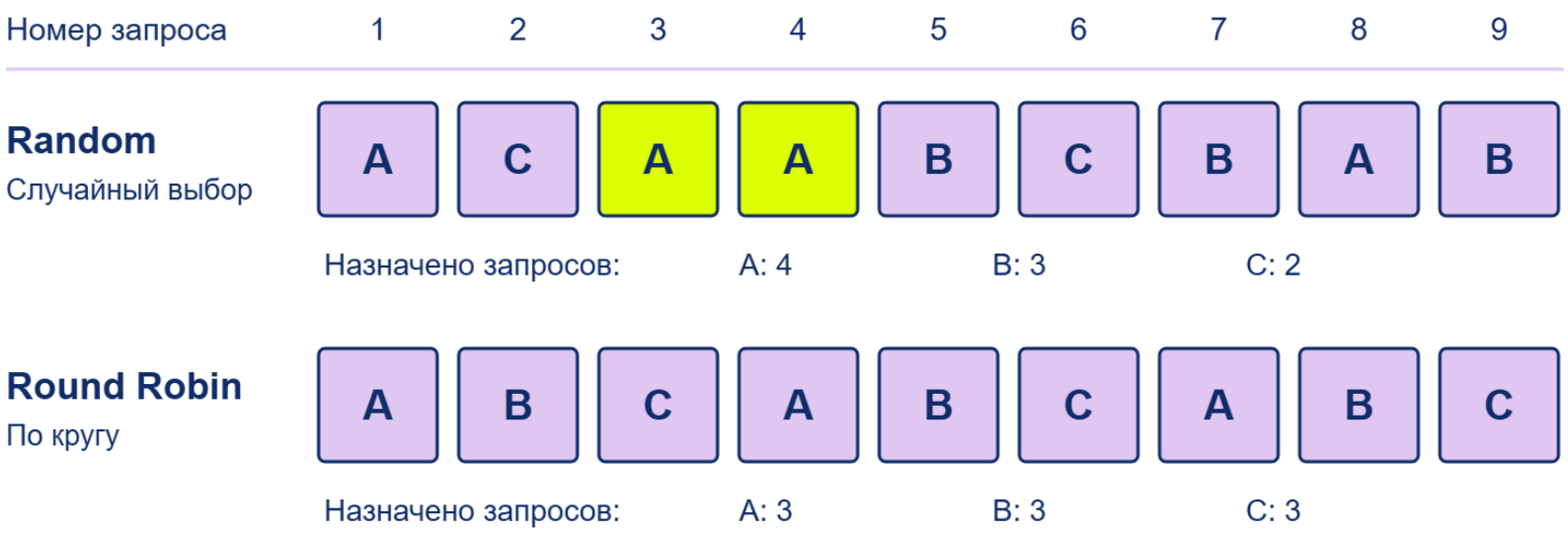
Пример: одинаковые экземпляры, HTTP/2, однотипные запросы.

Показатель	Экземпляр A	Экземпляр B
TCP-соединений	1	10
Запросов выполняются	4	4
Запросов ожидают обработки	96	6
Всего активных запросов	100	10

Активными считаем принятые, но ещё не завершённые запросы.

Random и Round Robin

Выбор без текущих метрик загрузки.



Один балансировщик, три кандидата с равными весами

Равное число назначений не гарантирует равной загрузки.

Балансировка с учётом нагрузки

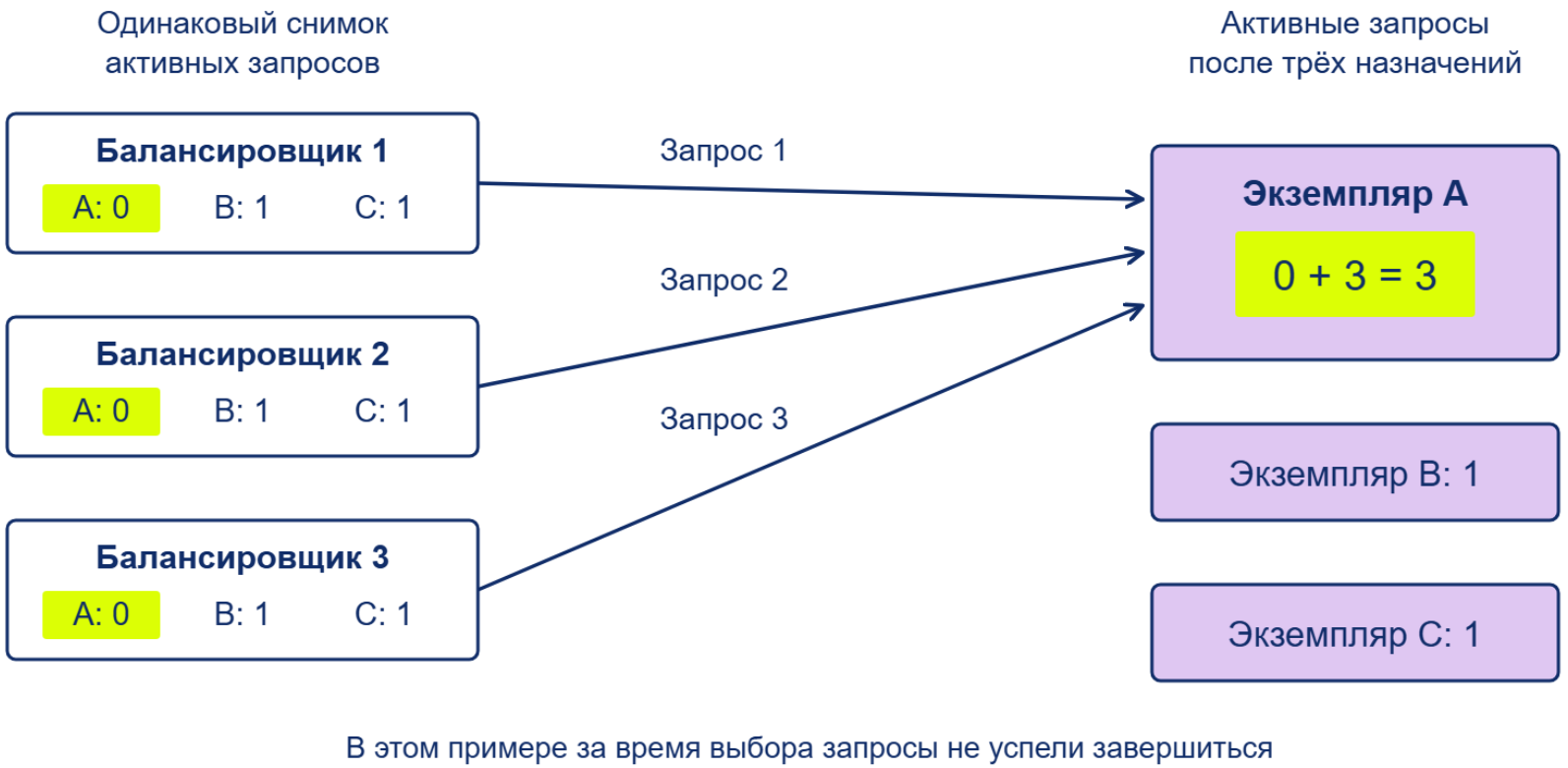
Предпочитаем кандидата с меньшей оценкой нагрузки.

Критерий	**Что сравниваем**	**Ограничение**
Least Connections	Активные соединения	В соединениях разное число запросов
Least Active Requests	Незавершённые запросы	Запросы имеют разную стоимость
Shortest Queue	Ожидающие запросы	Нужны сведения об очереди на экземпляре
Least Time	Оценку задержки и занятости	Прошлые ответы не определяют задержку следующего

Оценки могут быть неполными и устаревшими.

Несколько независимых балансировщиков

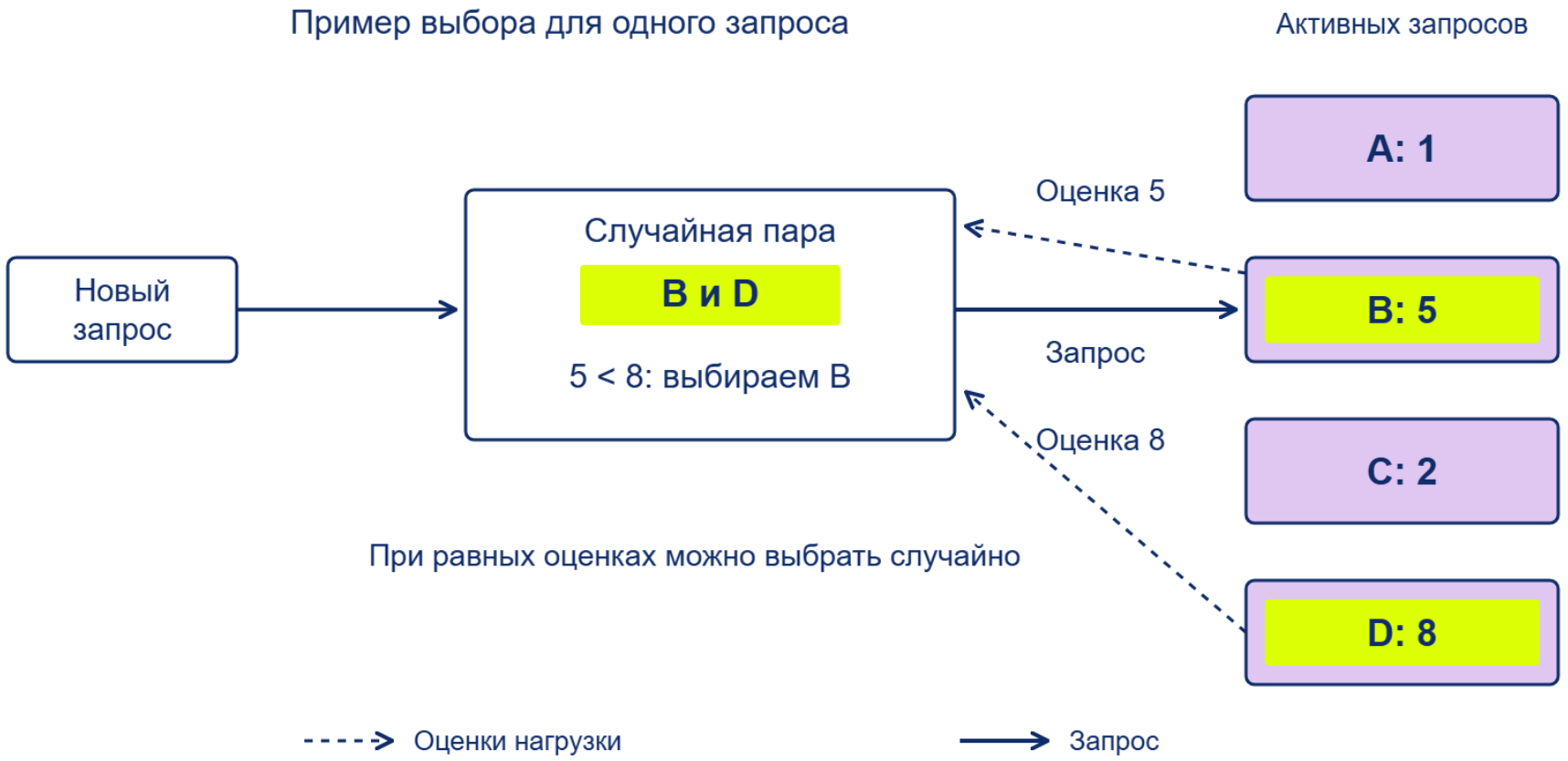
Каждый выбирает минимум по одному и тому же снимку нагрузки.



Балансировщики принимают решения независимо, не учитывая чужие действия.

Power of Two Choices (P2C)

Случайно выбираем двух кандидатов и сравниваем их нагрузку.



Исследование P2C: [The Power of Two Choices in Randomized Load Balancing](#) (2001)

Эффект второго случайного выбора

Размещаем n одинаковых шаров последовательно в n изначально пустых корзинах. Шары не удаляем. Кандидатов выбираем равномерно и независимо, с возвращением. Текущая загрузка известна точно.

$L_{\max}$ – максимальное число шаров в одной корзине после всех размещений.

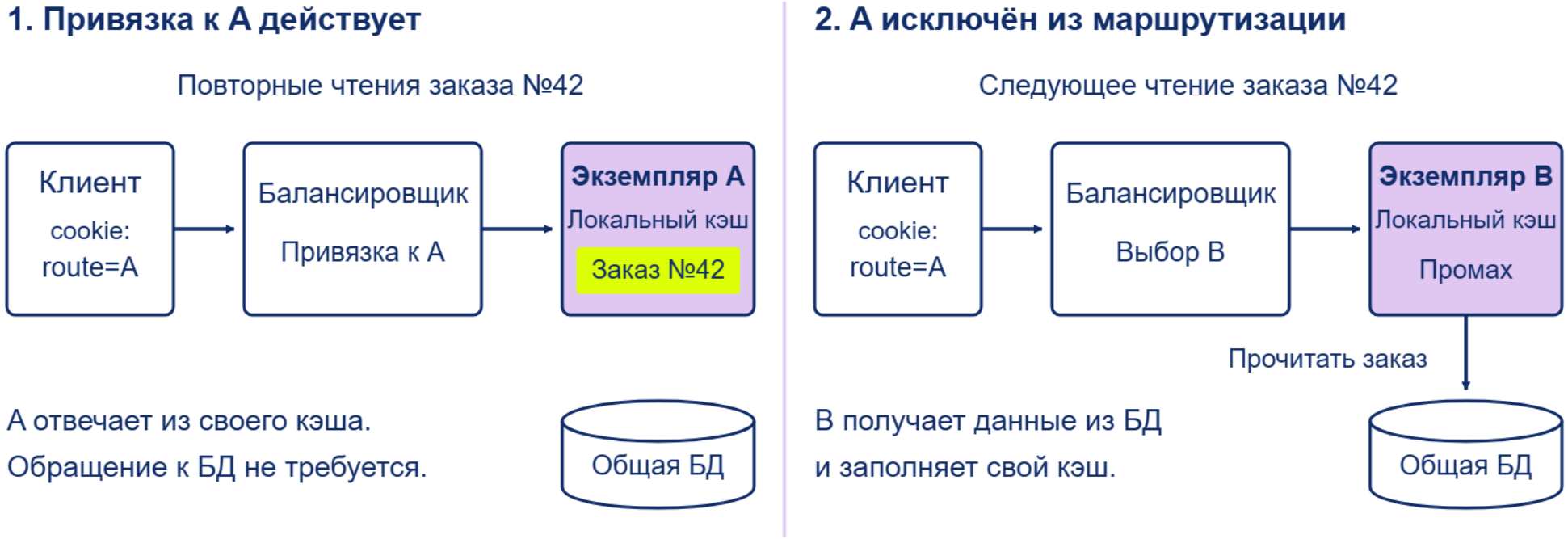
Как размещаем очередной шар	**Максимальная загрузка L_{max}**
Одна случайная корзина	$\Theta\left(\frac{\ln n}{\ln \ln n}\right)$
Менее загруженная из двух случайных	$\frac{\ln \ln n}{\ln 2} + O(1)$

Оценки выполняются с высокой вероятностью при $n \to \infty$.

Средняя загрузка в обоих случаях – 1. Второй выбор уменьшает максимальную загрузку.

Привязка запросов к экземпляру

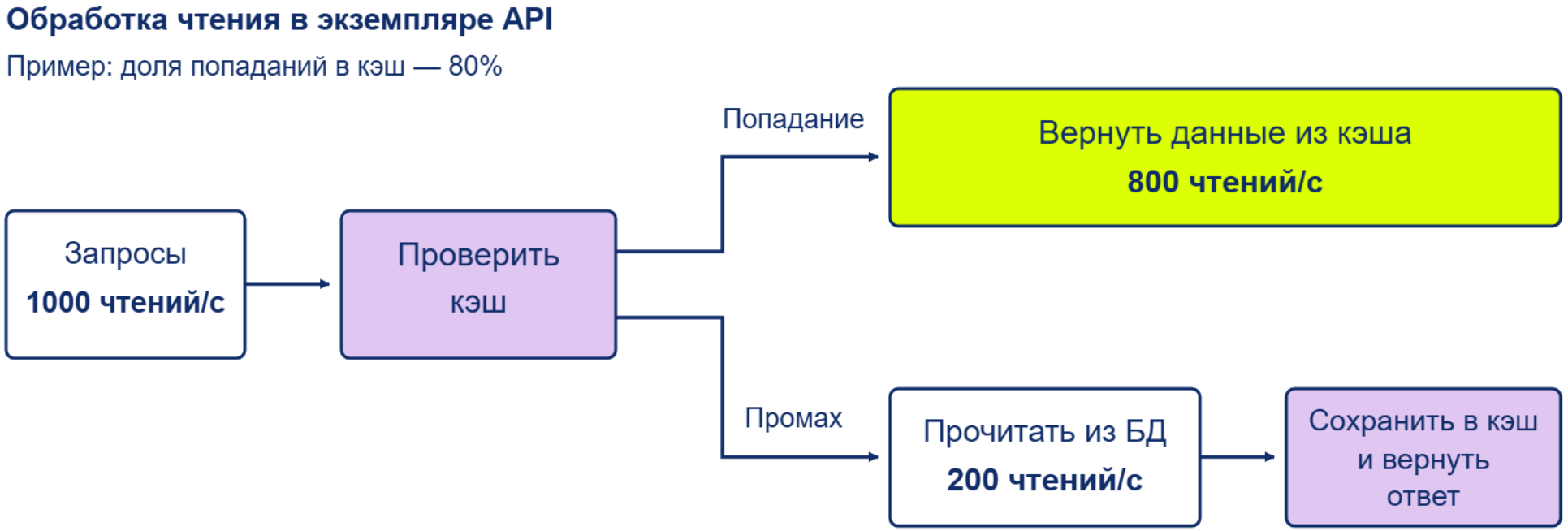
Sticky sessions: сохраняем привязку запросов сессии к выбранному экземпляру.



- Привязка ограничивает перераспределение нагрузки.
- Для смены экземпляра нужное состояние должно быть доступно другим экземплярам.

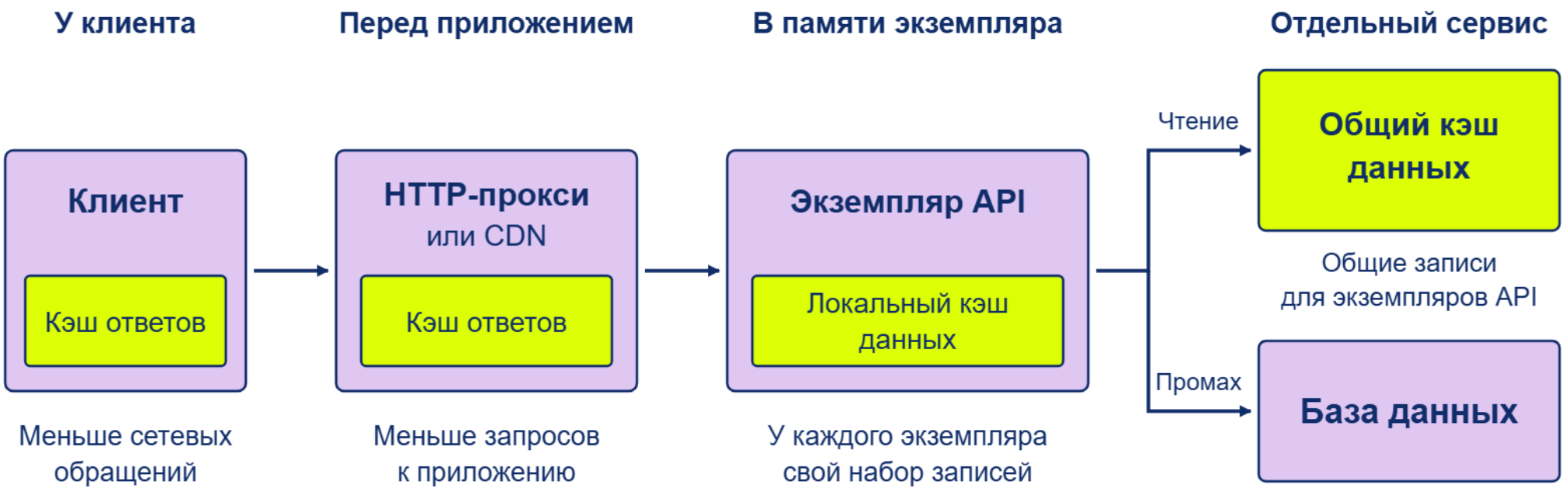
Кэширование сокращает повторную работу

Сохраняем результат чтения и используем его повторно, пока такая копия допустима для ответа.



Где разместить кэш

Кэшировать можно готовый ответ или данные, из которых приложение его формирует.



При промахе в локальном кэше API читает общий, при промахе в общем — БД.

Несколько уровней можно сочетать. Для каждого нужны ключ и правила свежести.

Что включить в ключ кэша

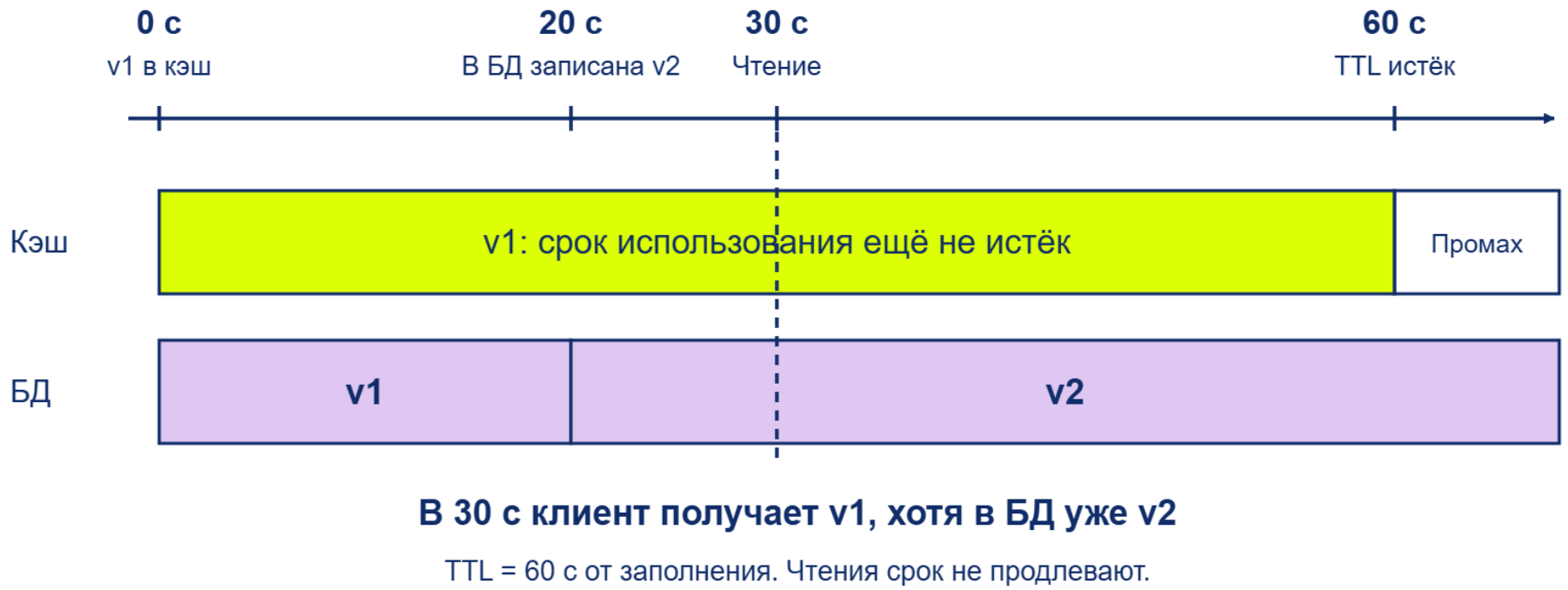
Кэшируем готовый список заказов текущего пользователя.

Пользователь	Запрос	Ключ кэша
17	GET /orders?status=open	orders:17:open
23	GET /orders?status=open	orders:23:open
17	GET /orders?status=closed	orders:17:closed

- Ключ учитывает параметры и контекст, от которых зависит ответ.
- Права доступа проверяем и при попадании в кэш.

Свежесть данных и TTL

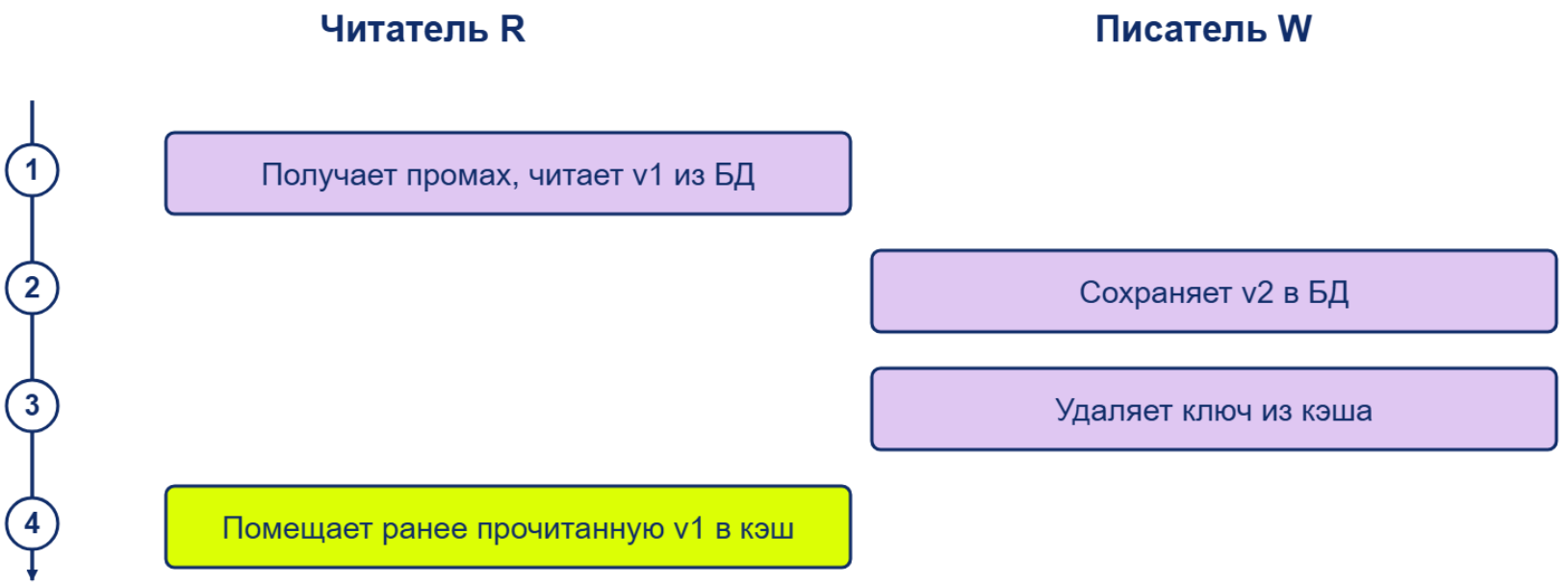
Запись может оставаться в кэше после изменения данных в БД.



TTL задаёт срок использования копии. В этой схеме БД не уведомляет кэш об изменении заказа.

Инвалидация при параллельных запросах

После изменения БД удаляем старую запись из кэша.
Параллельное заполнение может вернуть её обратно.



Итог: в БД v2, в кэше снова v1

Удаление записи не отменяет уже начавшееся чтение.

Вытеснение записей из кэша

Кэш вмещает три записи. Приходит новый ключ D. Какую запись удалить?

Обращения по порядку: **A, B, C, A, A, B, B, A, C**.

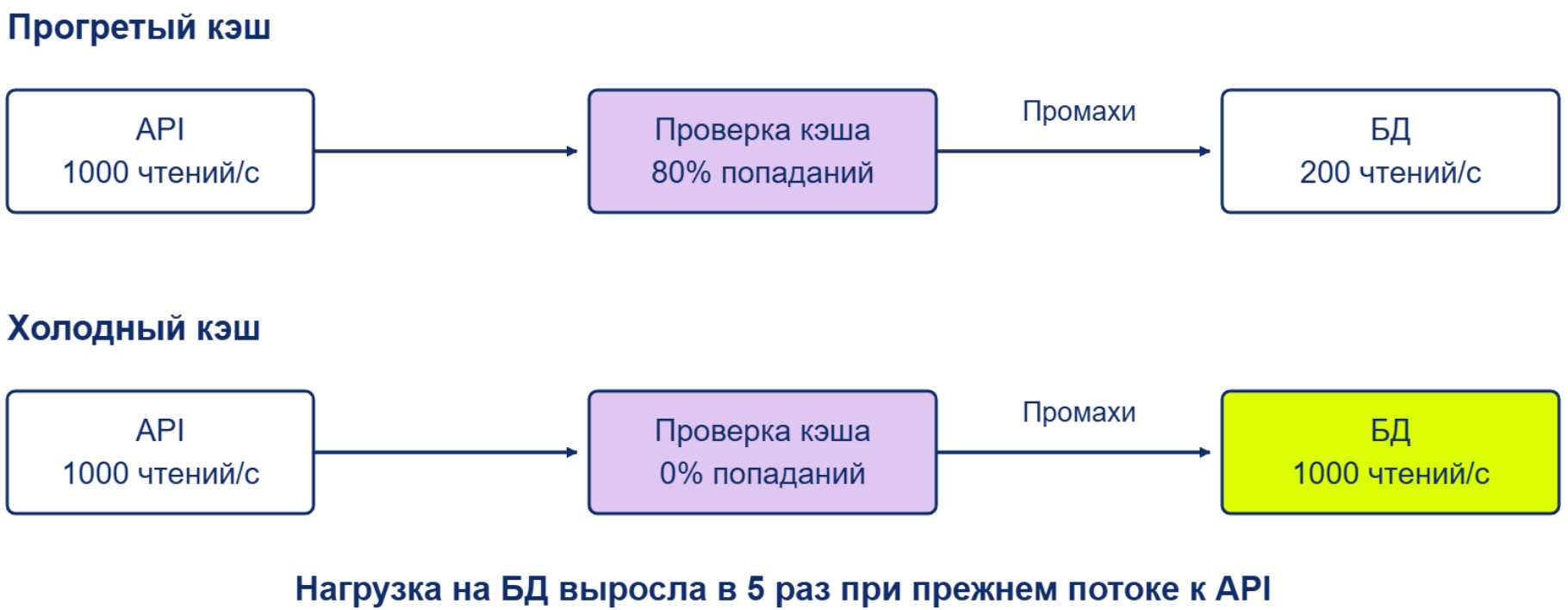
Запись	Порядок добавления	Число обращений	Последнее обращение
A	1	4	8
B	2	3	7
C	3	2	9

Политика	Критерий	Удалит
FIFO	Добавлена раньше остальных	A
LRU	Дольше всего не использовалась	B
LFU	Наименьшая частота обращений	C

Пример исследования:
A large scale analysis of hundreds of in-memory cache clusters at Twitter (2020)

Холодный кэш и всплеск нагрузки

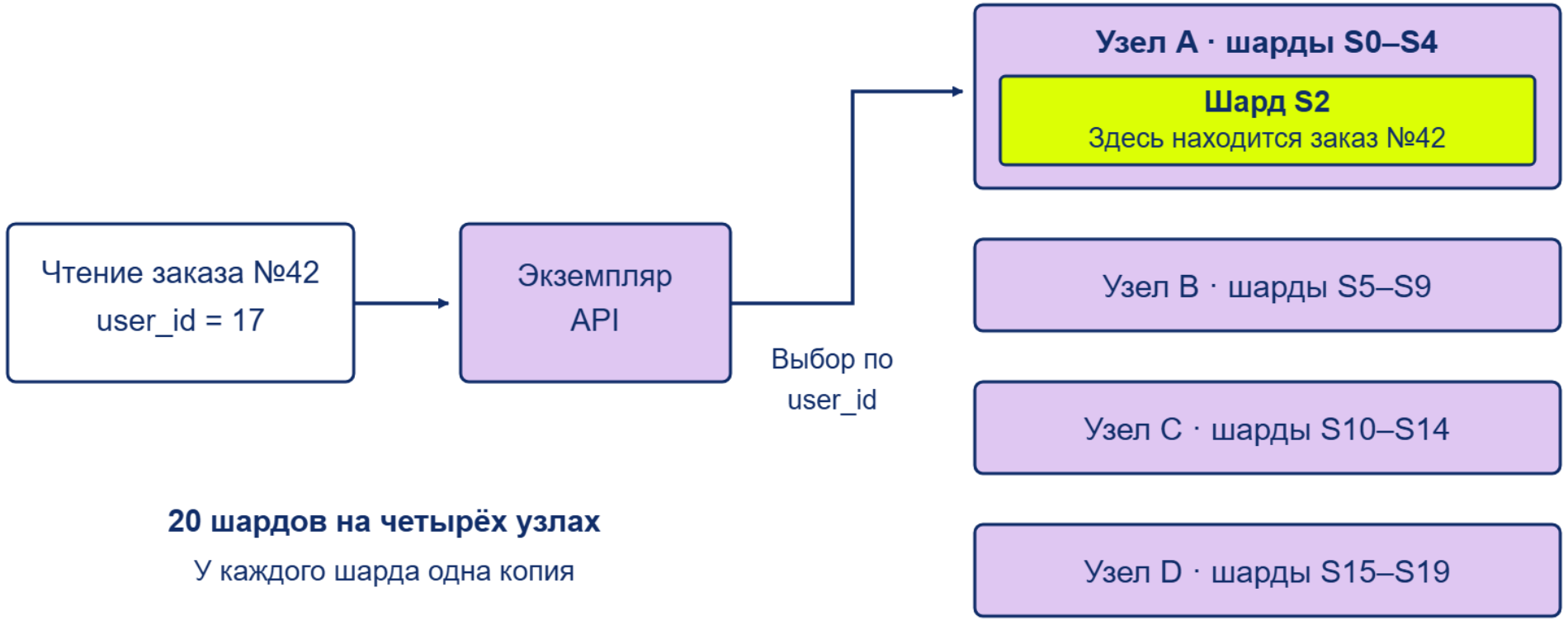
Данные сохранились в БД, но после потери кэша поток обращений к ней резко вырос.



Нужны запас мощности и ограничение потока промахов.

Шардинг: распределяем записи

Делим набор данных на части и размещаем их на разных узлах.



Запрос с известным ключом можно направить к владельцу нужной части.

Шард, реплика и узел

Термин	Значение	Пример
Шард	Логическая часть набора данных	Заказы некоторой группы пользователей
Реплика шарда	Копия данных этого шарда	Две копии S2
Узел	Место размещения и обработки данных	Сервер A хранит S0 и S2

Один узел может хранить несколько шардов.
Реплики одного шарда размещают на разных узлах.

В следующих примерах у каждого шарда одна копия.

Выбор ключа шардирования

Ключ определяет, какие записи окажутся вместе и какие запросы станут локальными.

Ключ для заказов	**Удобный запрос**	**Что потребует дополнительной работы**
user_id	Заказы одного пользователя	Поиск только по order_id
order_id	Один заказ по его ID	Все заказы пользователя
created_at по диапазонам	Заказы за период	Поток новых записей в последний диапазон

Учитываем частые запросы, объём данных и распределение нагрузки.

Запрос без ключа может потребовать индекс или обращение к нескольким шардам.

Диапазоны и хэширование

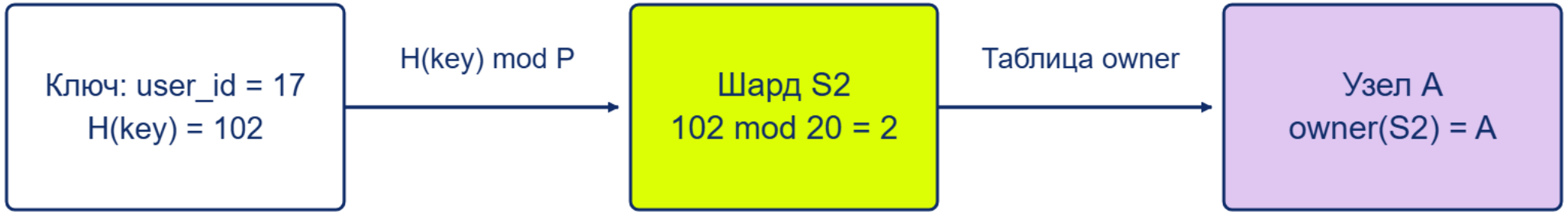
Два способа определить логическую часть по значению ключа.

	По диапазонам ключа	**По хэшу ключа**
Правило	Соседние значения в одном диапазоне	Часть определяется H(key)
Запрос за диапазон	Можно обратиться к нужным диапазонам	Часто затрагивает много частей
Последовательные новые ключи	Нагрузка может сосредоточиться в конце	Разные ключи можно рассеять
Популярный отдельный ключ	Остаётся в одной части	Тоже остаётся в одной части

Даже при равномерном распределении ключей нагрузка на шарды может различаться:
к одним ключам обращаются чаще, чем к другим.

Как найти узел по ключу

По ключу определяем шард, а по номеру шарда – узел.



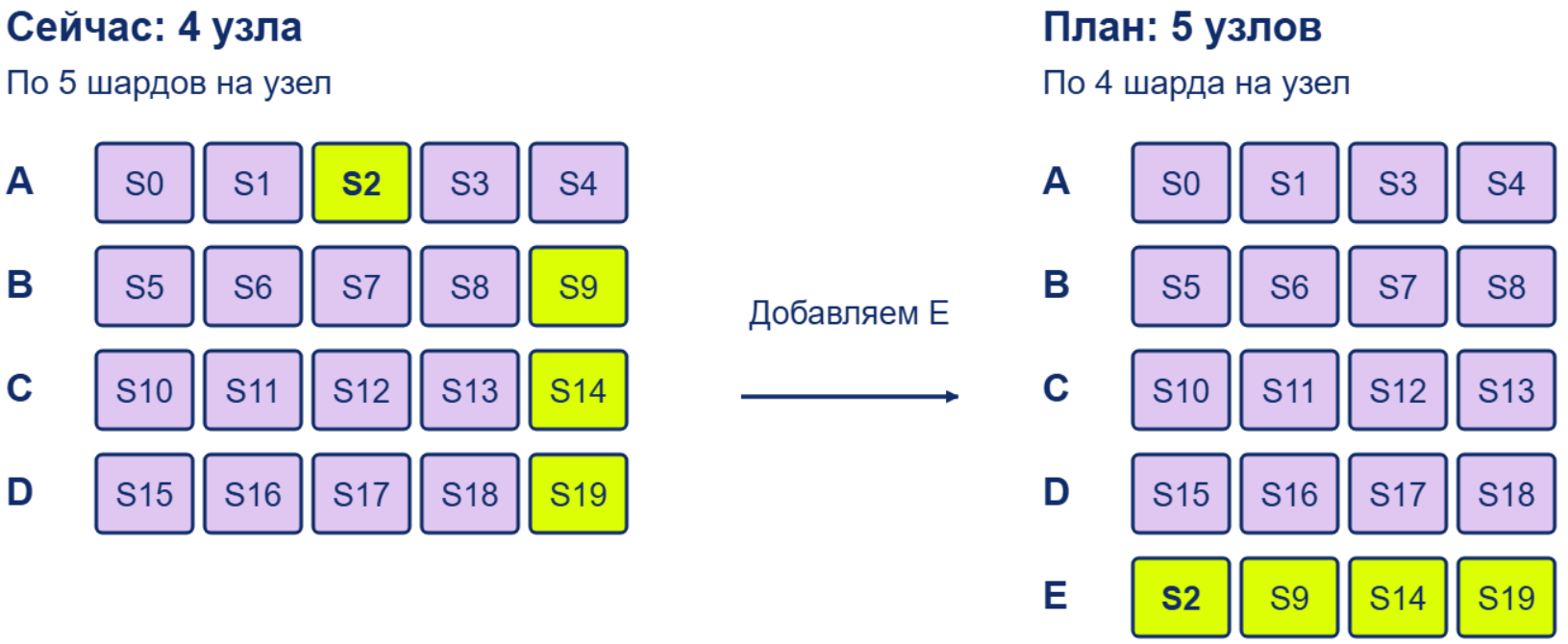
До расширения: P = 20 шардов; N = 4 узла

При переносе S2 меняется owner(S2), а P остаётся прежним

При переносе шарда меняется его узел, но ключи остаются в том же шарде.

Добавление узла при фиксированных шардах

Добавляем узел E. Число логических шардов остаётся равным 20.



Передаём на E шарды S2, S9, S14 и S19

Остальные 16 шардов сохраняют владельцев

Что меняется при H(key) mod N

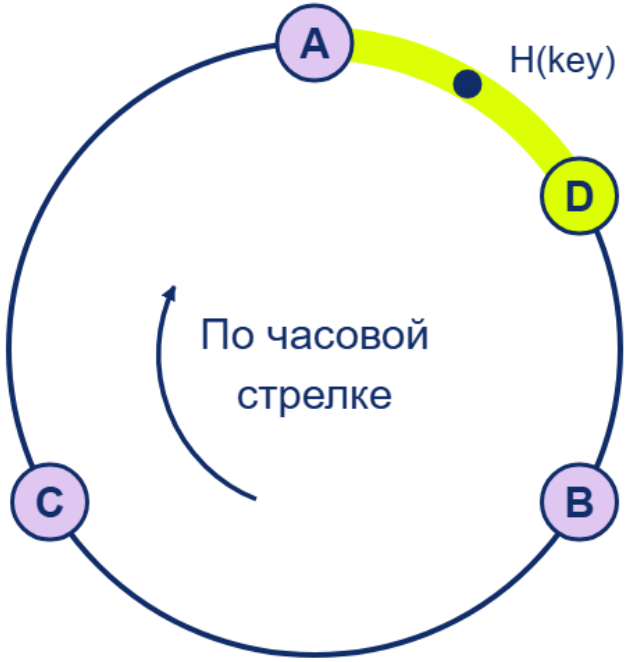
Если выбирать узел по H(key) mod N, добавление узла меняет размещение многих ключей.



Лаймовым отмечены 16 изменившихся назначений из 20

Кольцо consistent hashing

Ключ хранится на первом узле по часовой стрелке от H(key).



До добавления D

Интервал (A, B] принадлежал B

После добавления D

D получает только (A, D]

Остальные назначения сохраняются

Позиции на рисунке заданы условно

Оригинальная статья: Consistent Hashing and Random Trees (1997)

Виртуальные узлы

Каждому физическому узлу назначаем несколько позиций на кольце.

Одна позиция на физический узел



Четыре позиции на физический узел



Суммарные доли: A — 35%, B — 32%, C — 33%

Условный пример долей пространства хэшей. Точное равенство не гарантируется.

Rendezvous Hashing

owner(key) = arg $\max_n$ H(key, n), где n – один из текущих узлов.

Другое название - highest random weight (HRW).

Ключ	**A**	**B**	**C**	**Новый D**	**Владелец до / после**
k42	0,72	**0,91**	0,64	0,88	B / B
k73	0,83	0,26	0,74	**0,95**	A / D

На новый узел D переходят только ключи, для которых его оценка оказалась максимальной.

Оригинальная статья: Using Name-Based Mappings to Increase Hit Rates (1998)

Сравнение способов размещения

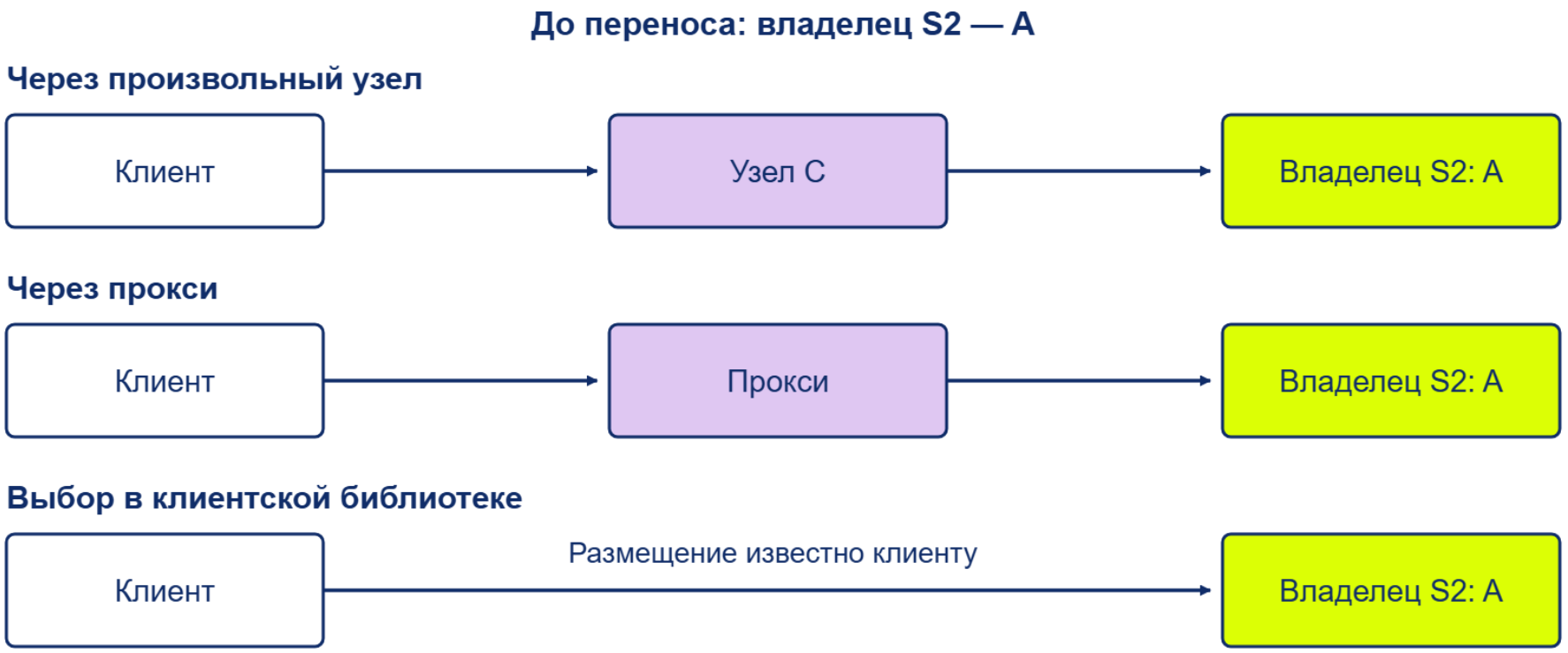
Вариант	Метаданные	Выбор владельца	Добавление узла
Фиксированные шарды + таблица owner	P записей о владельцах шардов	Вычислить шард; обращение к owner – O(1)	Переназначить выбранные шарды
Кольцо	M упорядоченных позиций и их владельцы	Поиск следующей позиции – O(log M)	Передать интервалы перед новыми позициями
Rendezvous Hashing (HRW)	Список из N узлов	Прямой перебор – O(N)	Передать ключи, для которых новый узел получил максимальную оценку

Подходы можно сочетать: например, назначать фиксированные шарды узлам с помощью HRW.

Для переноса данных нужен отдельный протокол.

Маршрутизация к владельцу данных

Как запрос попадает на узел, где хранятся нужные данные?

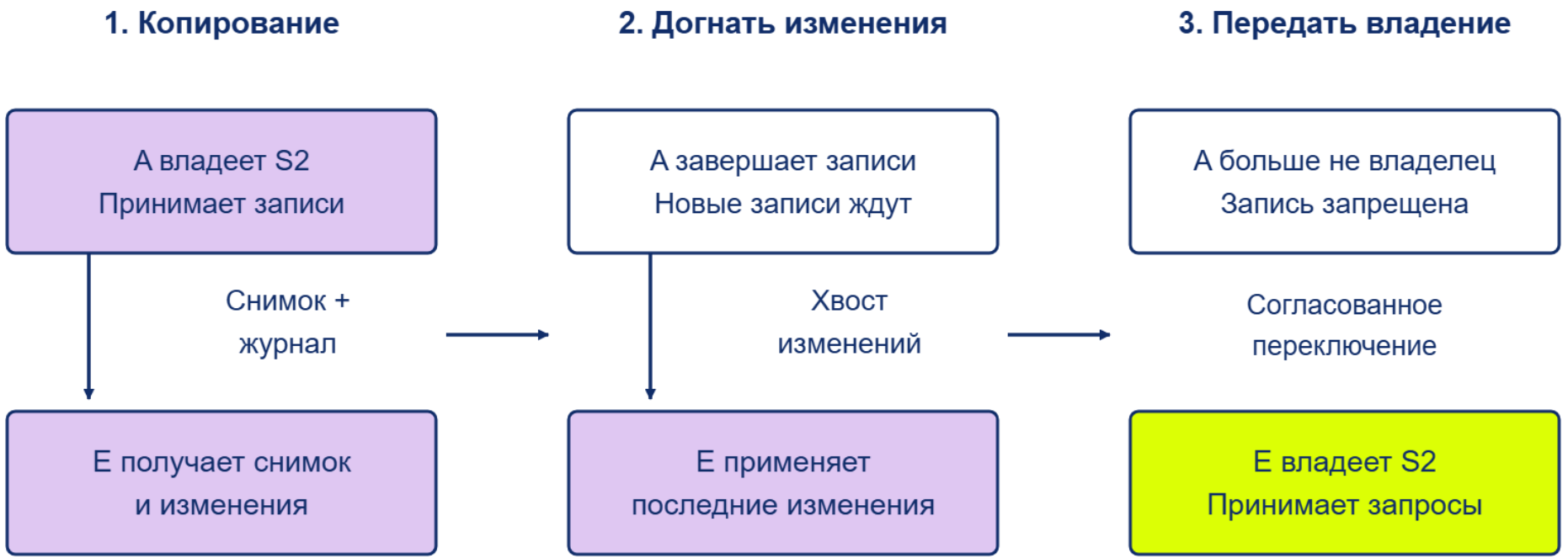


Три варианта пути одного запроса к заказу №42

Компонент, который выбирает узел для запроса, должен обновлять сведения о размещении данных.

Перенос шарда при продолжающихся записях

Переносим S2 с A на E. Во время копирования данные продолжают меняться.

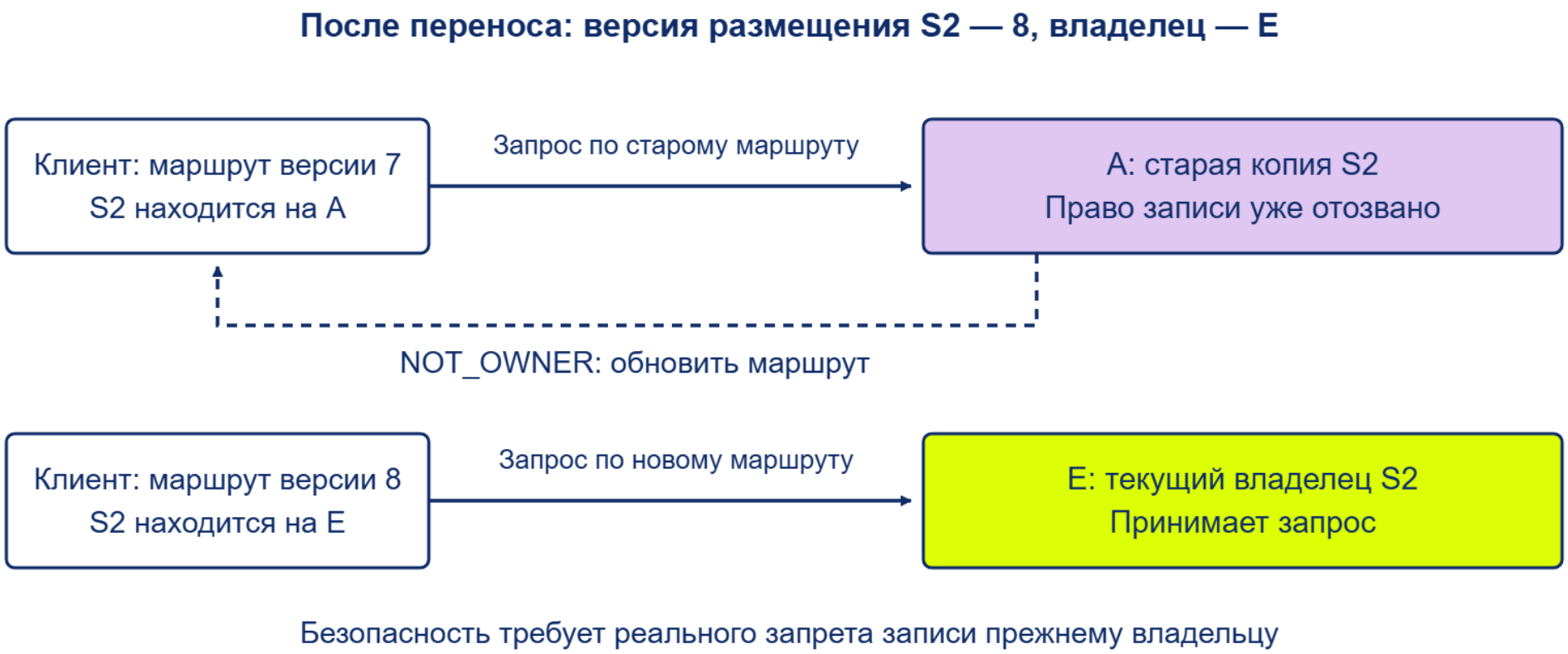


Переносим S2 с A на E с короткой паузой записи

Прежде чем E начнёт принимать записи, он должен получить все изменения, принятые A.

Устаревший маршрут и право записи

S2 уже перенесён на E, но часть клиентов всё ещё обращается к A.



Обновить маршрут недостаточно, нужно запретить запись прежнему владельцу.

Горячий шард и горячий ключ

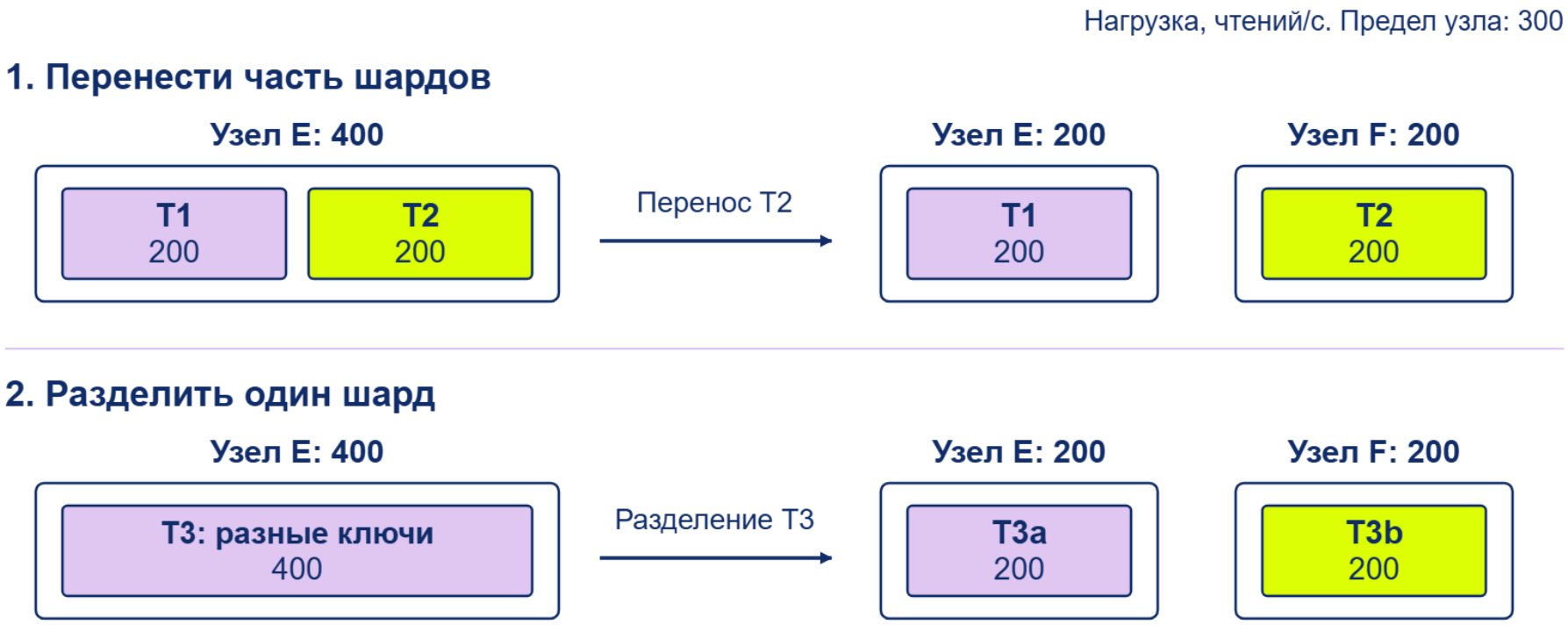
Во время распродажи на один товар приходится **40% чтений каталога**.



Дополнительные виртуальные узлы не распределят чтения одного товара между серверами.

Как разгрузить горячий шард

Нагрузку создают обращения к разным товарам каталога.

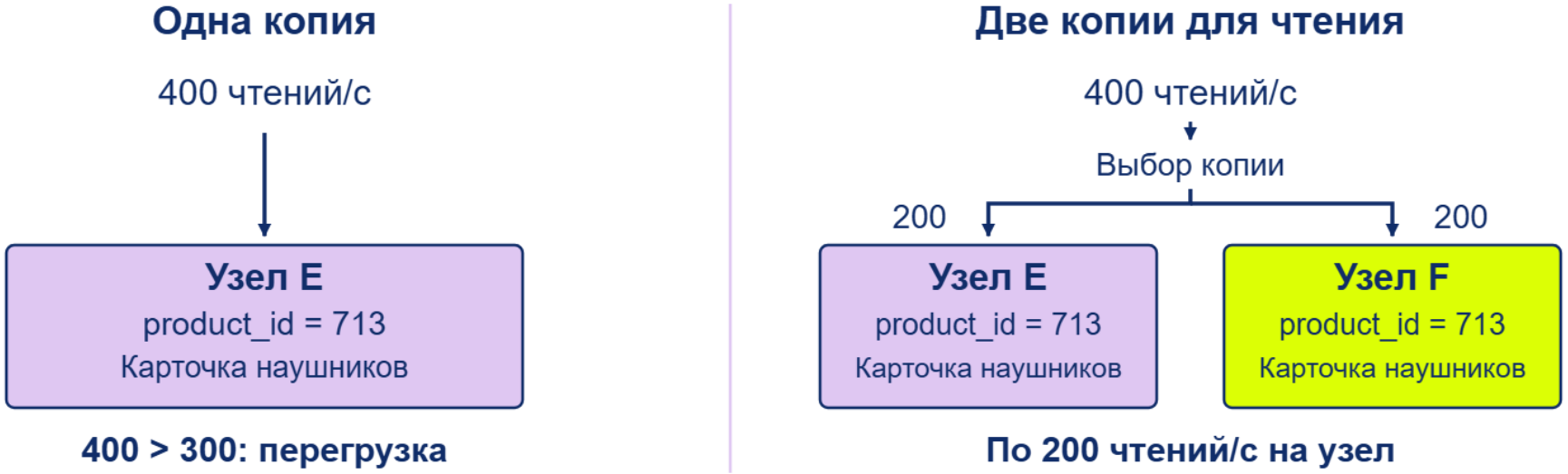


T1–T3 — шарды каталога. F — новый узел. Два независимых условных примера.

При переносе состав шарда сохраняется. При разделении ключи распределяются по новым шардам.

Как разгрузить горячий ключ

- **Чтения:** кэшировать данные и направлять запросы к разным копиям.
- **Промахи:** объединять одновременные загрузки одного значения.
- **Записи:** разделять состояние на независимые части, если это допустимо.



Предел узла: 300 чтений/с. F — новый узел без другой нагрузки. Числа условные.

Нужно определить, как обновлять копии и насколько они могут устаревать.

Как выбрать способ масштабирования

Цель – выдерживать заданную нагрузку при допустимых задержках и доле ошибок.

Что ограничивает систему	Возможное действие	Что проверить
Не хватает мощности API	Добавить экземпляры	Состояние, балансировка, общие зависимости
Много повторных чтений	Добавить кэш	Ключ, свежесть, промахи и холодный старт
Один узел не вмещает данные или не справляется с запросами	Разделить на шарды	Ключ, неравномерность нагрузки, перенос данных
Один объект создаёт основную нагрузку	Изменить обработку этого объекта	Чтения, записи и требуемые гарантии

Проверяем пропускную способность, p99 задержки, долю ошибок и рост очередей.

Зачем нужны реплики данных

Шардинг распределяет данные между узлами.

Репликация создаёт несколько копий каждого шарда.



Следующая лекция: как поддерживать копии данных и работать при отказах.

Материалы к лекции

Книги и практика

- Designing Distributed Systems (2-е изд., гл. 6-7)
- Site Reliability Engineering (гл. 19–20; дополнительно – 21)
- Designing Data-Intensive Applications (2-е изд., гл. 7)
- Caching challenges and strategies

Алгоритмы и исследования

- The Power of Two Random Choices: A Survey (введение и § 1)
- A large scale analysis of hundreds of in-memory cache clusters at Twitter
- Consistent Hashing and Random Trees
- Using Name-Based Mappings to Increase Hit Rates

Дополнительные материалы

Балансировка нагрузки

- Matt Klein. Load balancing and proxying.
- Marc Brooker. The power of two random choices.
- HAProxy. Test driving P2C load balancing.
- Stanford CS265. Balls in Bins and Power-of-Two-Choices.

Кэширование и шардинг

- Etsy. How Etsy caches.
- Stanford CS168. Consistent Hashing (§ 1).
- VShard в Tarantool.
- Damian Gryski. Consistent Hashing: Algorithmic Tradeoffs.