2024

System Design

Лекция 2: Масштабирование системы и Load Balancer

CS302

Артемий Мазаев

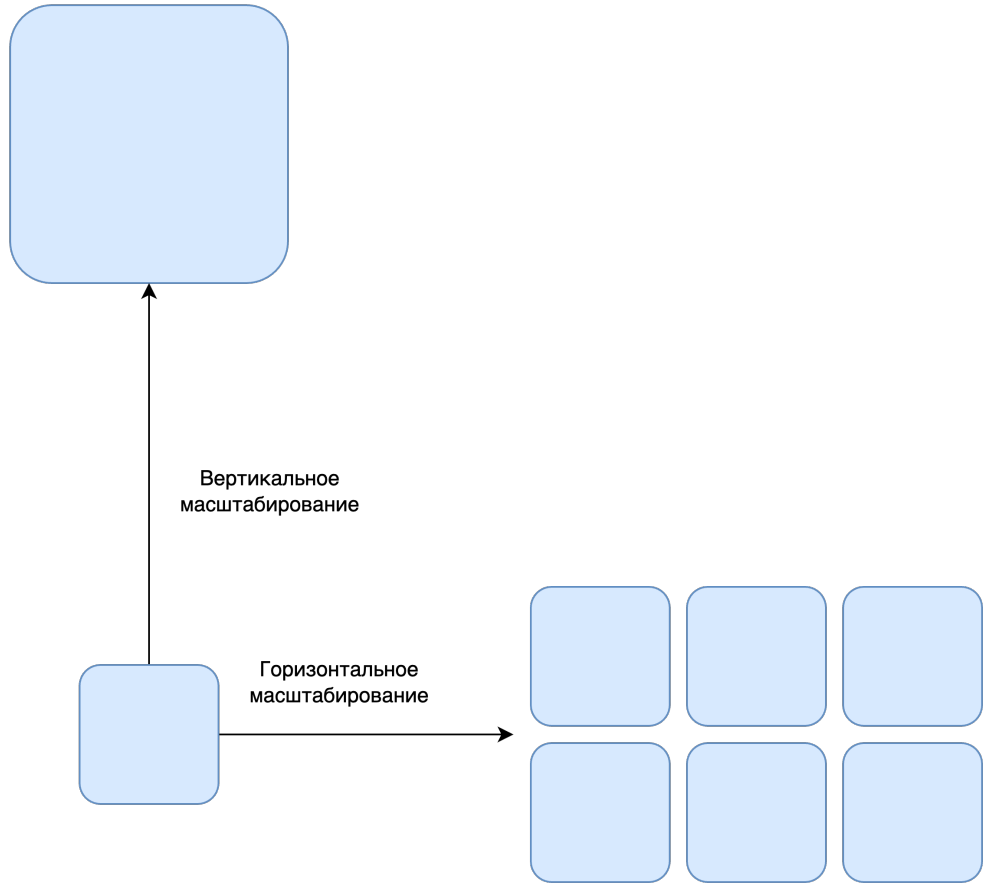
Компьютерные науки и прикладная математика, МАИ

Масштабируемость

Масштабируемость

Масштабируемость - это мера способности системы реагировать на изменения путем добавления или удаления ресурсов для удовлетворения потребностей.

Масштабируемость



Вертикальное масштабирование

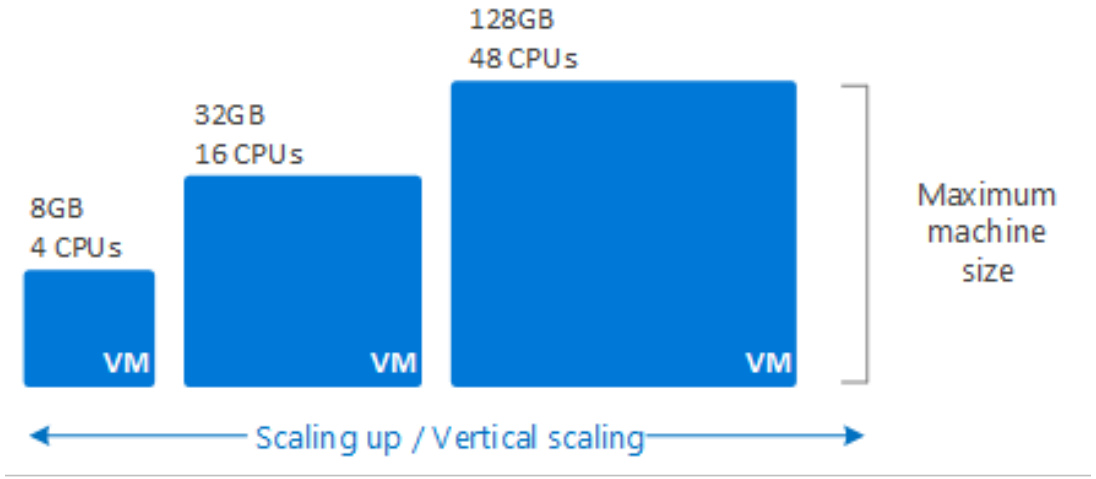
Вертикальное масштабирование - увеличивает мощность системы путем добавления ресурсов к существующей машине, улучшая тем самым возможности приложения.

Преимущества:

- Просто внедрить
- Легко управлять
- Согласованность данных

Недостатки:

- Риск высокого времени простоя
- Сложнее обновлять
- Возможна единственная точка отказа



Горизонтальное масштабирование

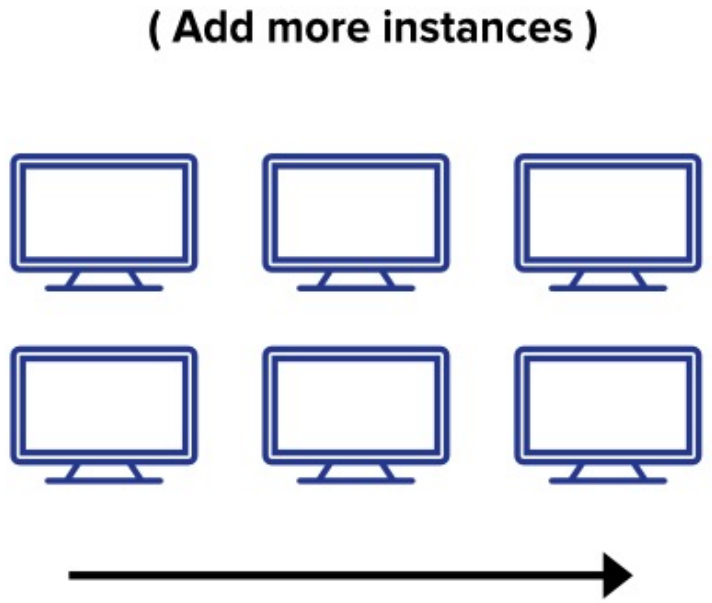
Горизонтальное масштабирование увеличивает масштаб системы путем добавления дополнительных машин для равномерного распределения нагрузки

Преимущества

- Увеличенная избыточность
- Лучшая устойчивость к отказам
- Гибкость и эффективность
- Проще обновлять

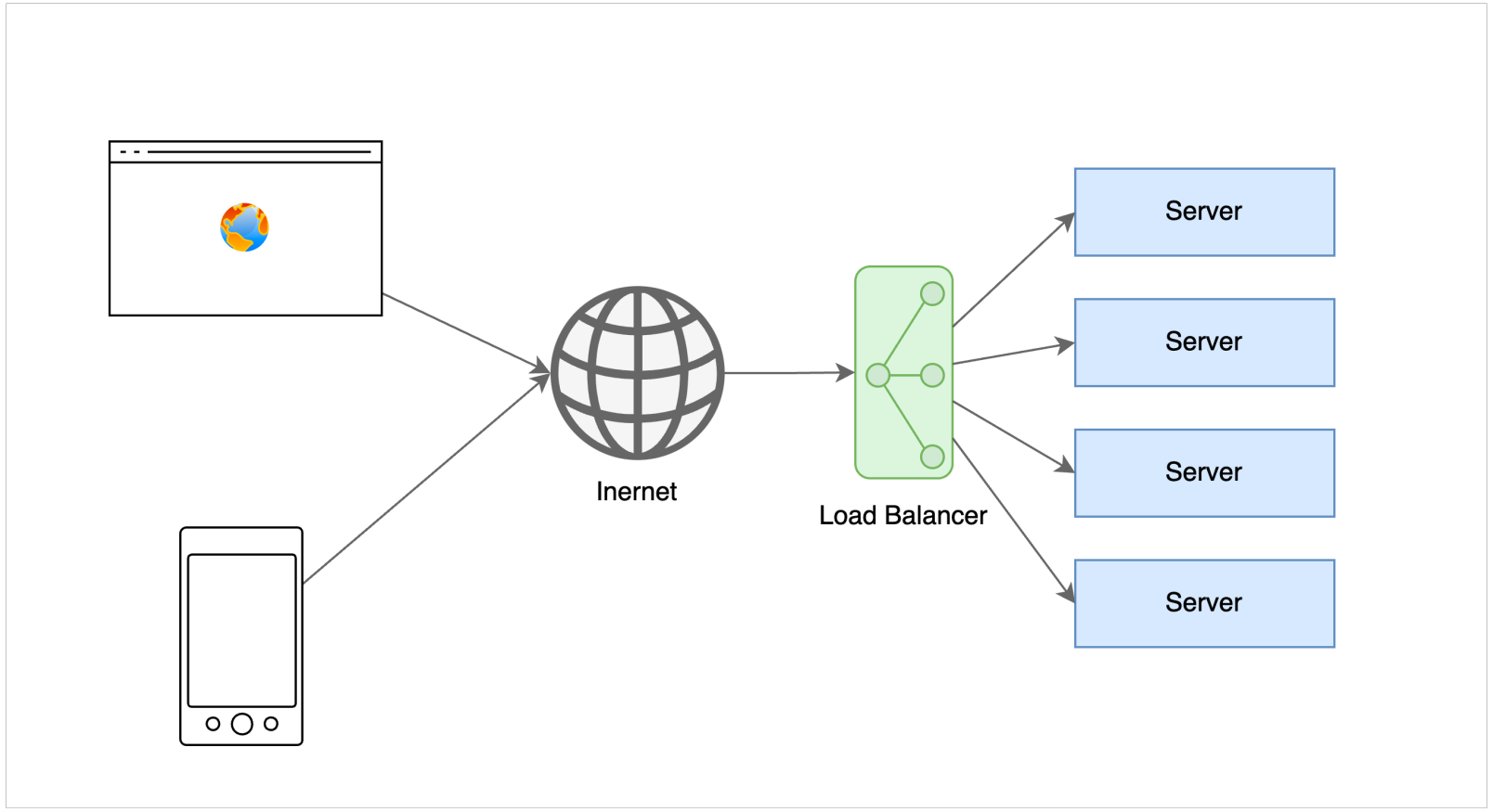
Недостатки

- Увеличенная сложность
- Несогласованность данных
- Увеличенная нагрузка на последующие сервисы

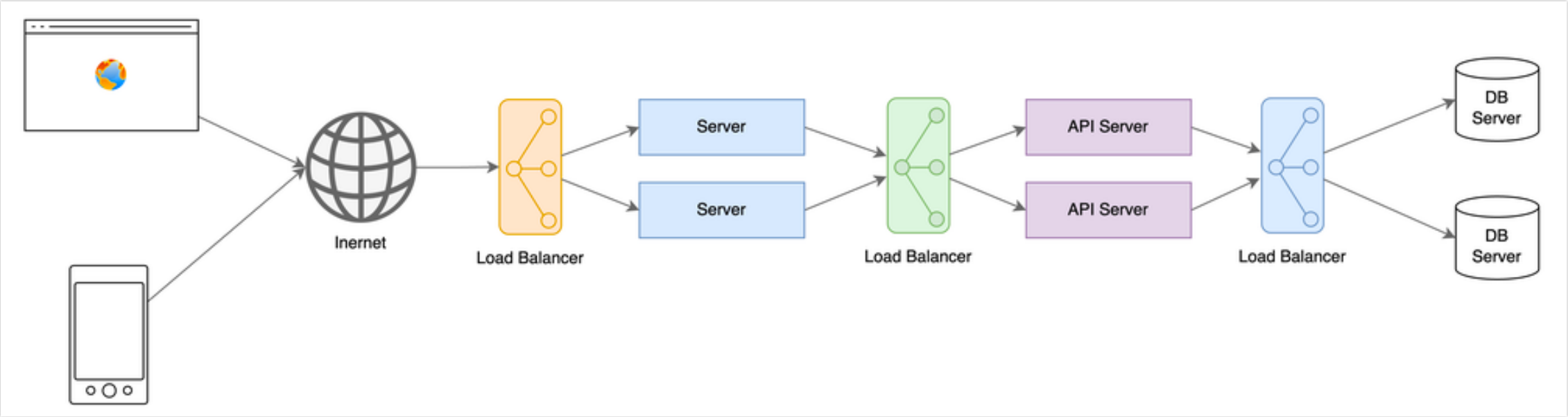


Load Balancer

Load Balancer



Load Balancer



Распределение рабочей нагрузки

- **На основе хоста:** Распределяет запросы на основе запрошенного имени хоста.
- **На основе пути:** Использует весь URL для распределения запросов вместо только имени хоста.
- **На основе содержимого:** Изучает содержимое сообщения запроса. Это позволяет распределять нагрузку на основе содержимого, например, значения параметра.

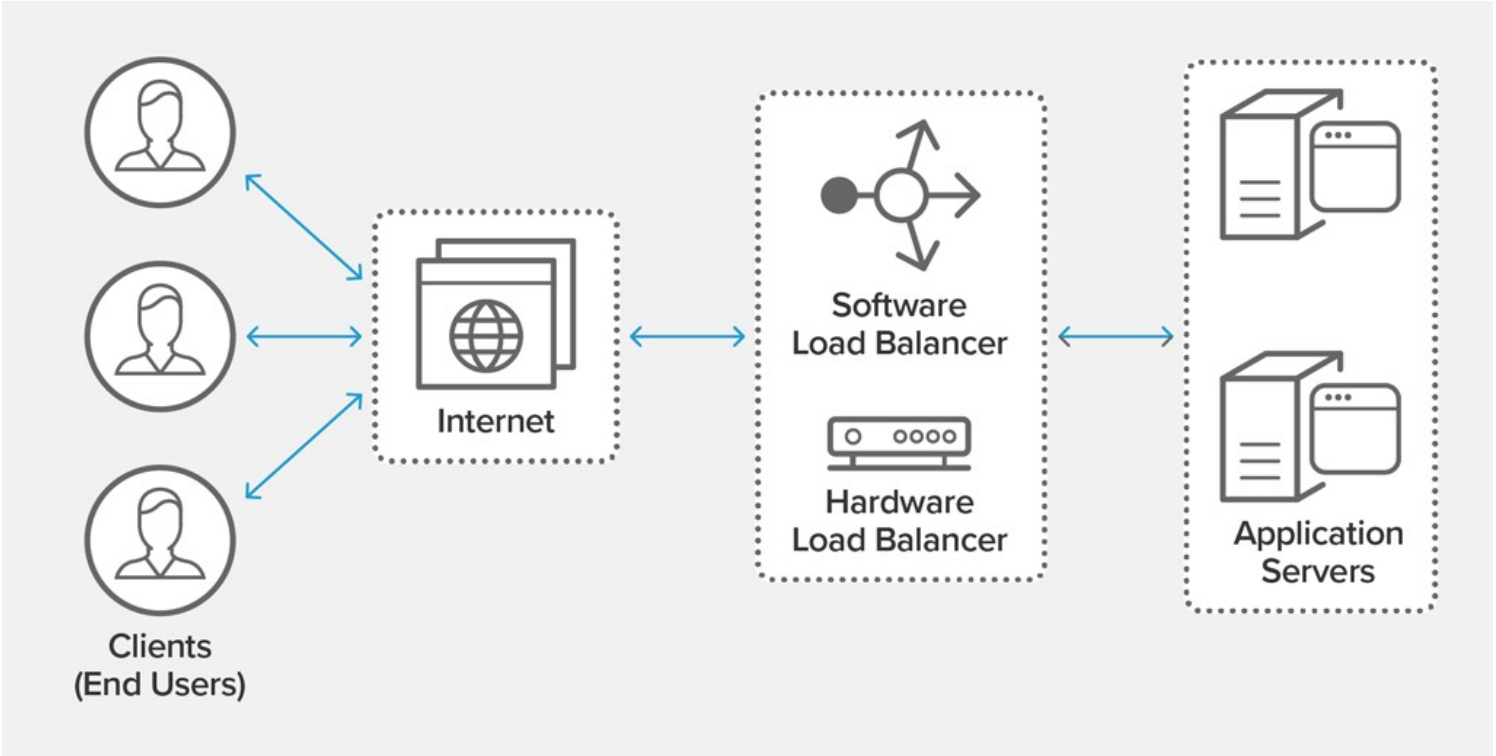
Уровни

Балансировщики нагрузки работают на двух уровнях: сетевом и прикладном.

- Сетевой уровень (Уровень 4):
  - Маршрутизация основана на сетевой информации, такой как IP-адреса.
  - Высокая скорость работы.
  - Не учитывает содержимое запросов.
- Прикладной уровень (Уровень 7):
  - Маршрутизация основана на содержимом запросов.
  - Полное понимание трафика.
  - Управление нагрузкой на более высоком уровне.

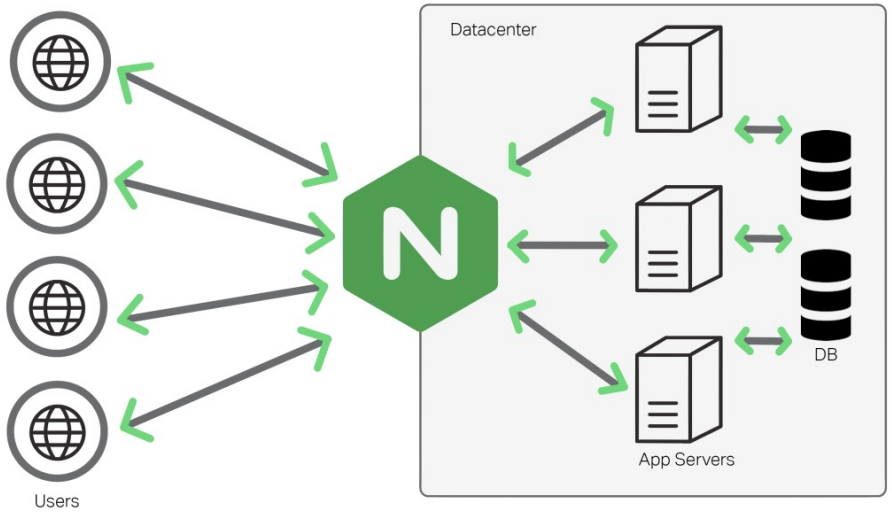
Типы балансировщиков нагрузки:

- **Программные**
- **Аппаратные**



Типы балансировщиков нагрузки:

- **Программные:**
  - Проще в развертывании, экономичны и гибки.
  - Позволяют настраивать под конкретные потребности среды.
  - Могут быть как установычными решениями, так и облачными сервисами.



Типы балансировщиков нагрузки:

- **Аппаратные:**
  - Основаны на физическом оборудовании для распределения трафика.
  - Обрабатывают большие объемы трафика, но дороже и менее гибки.
  - Включают фирменное программное обеспечение, требующее обслуживания и обновлений.

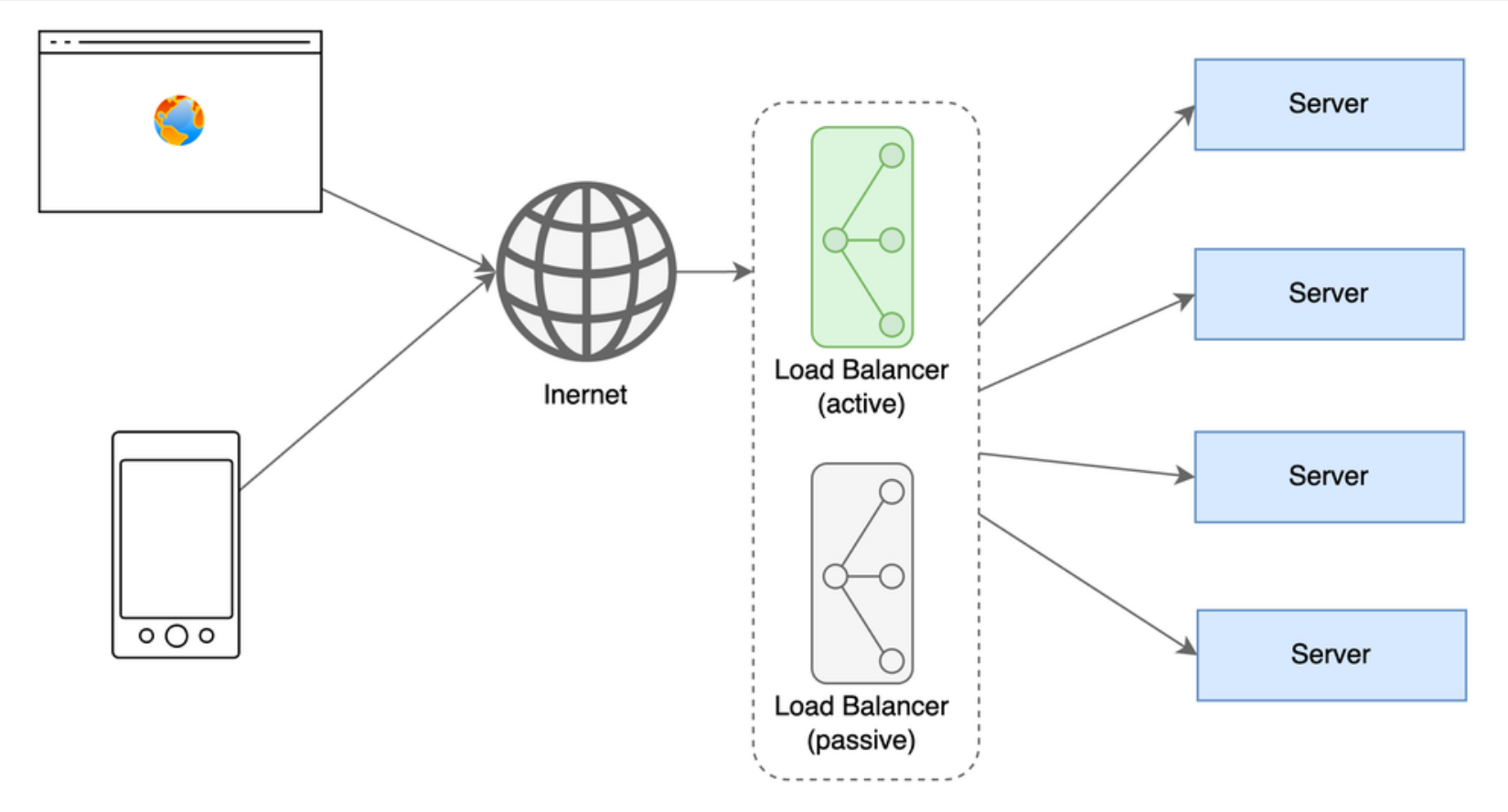
Алгоритмы маршрутизации:

- **Круговой (Round-robin):** Запросы распределяются по серверам поочередно.
- **Взвешенный Круговой (Weighted Round-robin):** Учитывает характеристики серверов, используя веса, назначенные администратором через DNS-записи.
- **Наименьшее Количество Соединений (Least Connections):** Новый запрос отправляется на сервер с наименьшим числом активных соединений с клиентами.
- **Наименьшее Время Ответа (Least Response Time):** Запросы отправляются на сервер с самым быстрым временем ответа и наименьшим количеством активных соединений.
- **Наименьшая Полоса Пропускания (Least Bandwidth):** Запросы направляются на сервер с наименьшим объемом трафика.
- **Хеширование (Hashing):** Распределение запросов на основе ключа, такого как IP-адрес клиента или запрашиваемый URL.

Преимущества

- Масштабируемость
- Избыточность
- Гибкость
- Эффективность

Избыточные балансировщики нагрузки



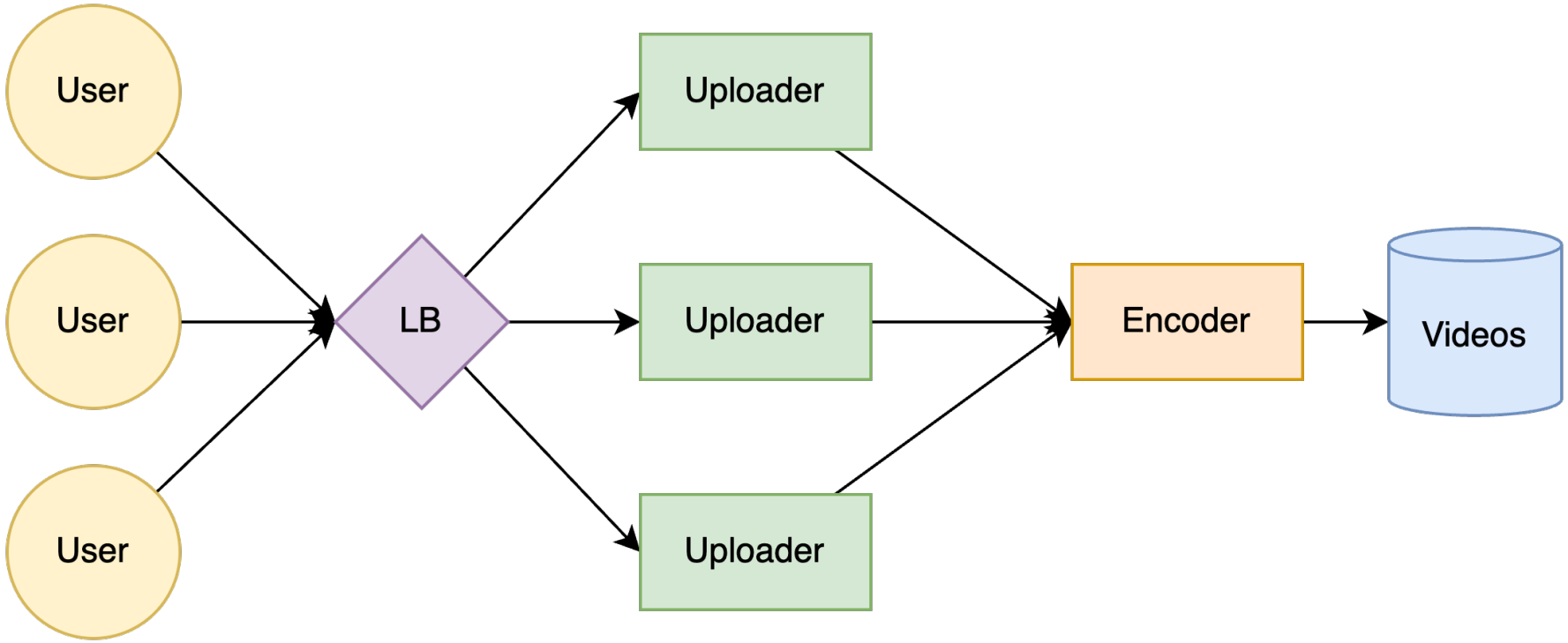
Функции

- **Автомасштабирование:** Подстройка ресурсов под изменяющийся спрос.
- **Привязка сессий:** Гарантированное сохранение состояния сессии для пользователя или устройства.
- **Проверка состояния:** Отслеживание работоспособности ресурсов и их автоматическое исключение при необходимости.
- **Постоянные соединения:** Создание долгосрочных соединений, например, WebSocket.
- **Шифрование и сертификация:** Обеспечение безопасных соединений через TLS и SSL.
- **Сжатие:** Уменьшение размера передаваемых данных.
- **Кэширование:** Ускорение доступа к часто запрашиваемым ресурсам.
- **Логирование:** Фиксация информации о запросах и ответах для аудита и аналитики.
- **Трассировка запросов:** Присвоение уникальных идентификаторов для отслеживания и устранения проблем.
- **Переадресация:** Автоматическое направление запросов в зависимости от условий.
- **Фиксированный ответ:** Предоставление статических ответов на запросы, например, сообщений об ошибках.

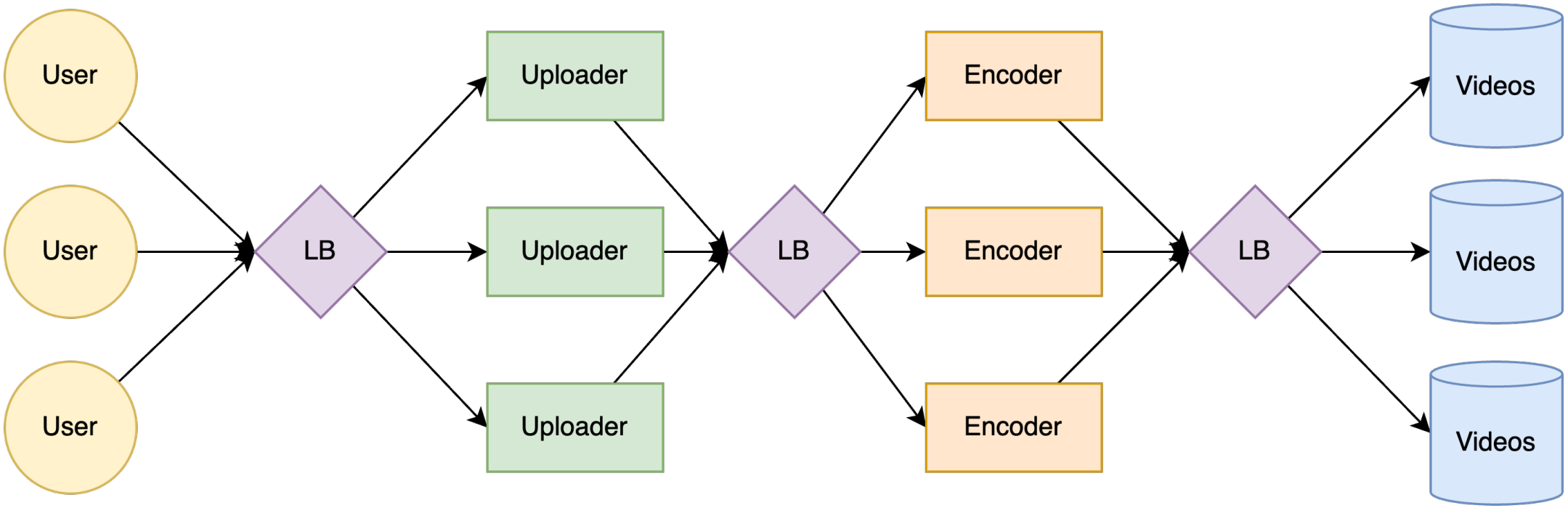
Масштабирование системы

Масштабирование системы

Естественной задачей для балансировщика является балансировкапользовательского трафика — распределение приходящих запросов отпользователя между инстансами сервиса:



Масштабирование системы



Принцип работы балансировщика нагрузки

- Анализ того, какие из серверов готовы принять и обработать запроспользователя
- Выбор наиболее подходящего сервера по заданному алгоритму

Алгоритм выбора подходящего сервера

- **Приоритет подключений:** Минимизация количества подключений к серверу.
- **Приоритет времени ответа:** Минимизация задержки перед ответом.
- **Приоритет трафика:** Минимизация сетевой нагрузки.
- **Round-robin:** Серверы обслуживают запросы по кругу из предварительно определенной очереди.
- **Weighted round-robin:** Серверам назначаются веса на основе вышеуказанных характеристик.
- **Пользовательский хэш:** Выбор сервера основан на хэше (IP, user_id и т. д.).

Преимущества балансировщика:

- **Улучшение пользовательского опыта**: запросы направляются к подходящему серверу, повышая эффективность обработки.
- **Снижение нагрузки на сервера**: запросы распределяются между серверами, снижая нагрузку на каждый из них.
- **Обеспечение отказоустойчивости**: даже при отказе части серверов (в случае stateless сервиса) система остается работоспособной.
- **Автоматическое масштабирование**: интеллектуальные балансировщики могут прогнозировать рост нагрузки и инициировать автоскейлинг.

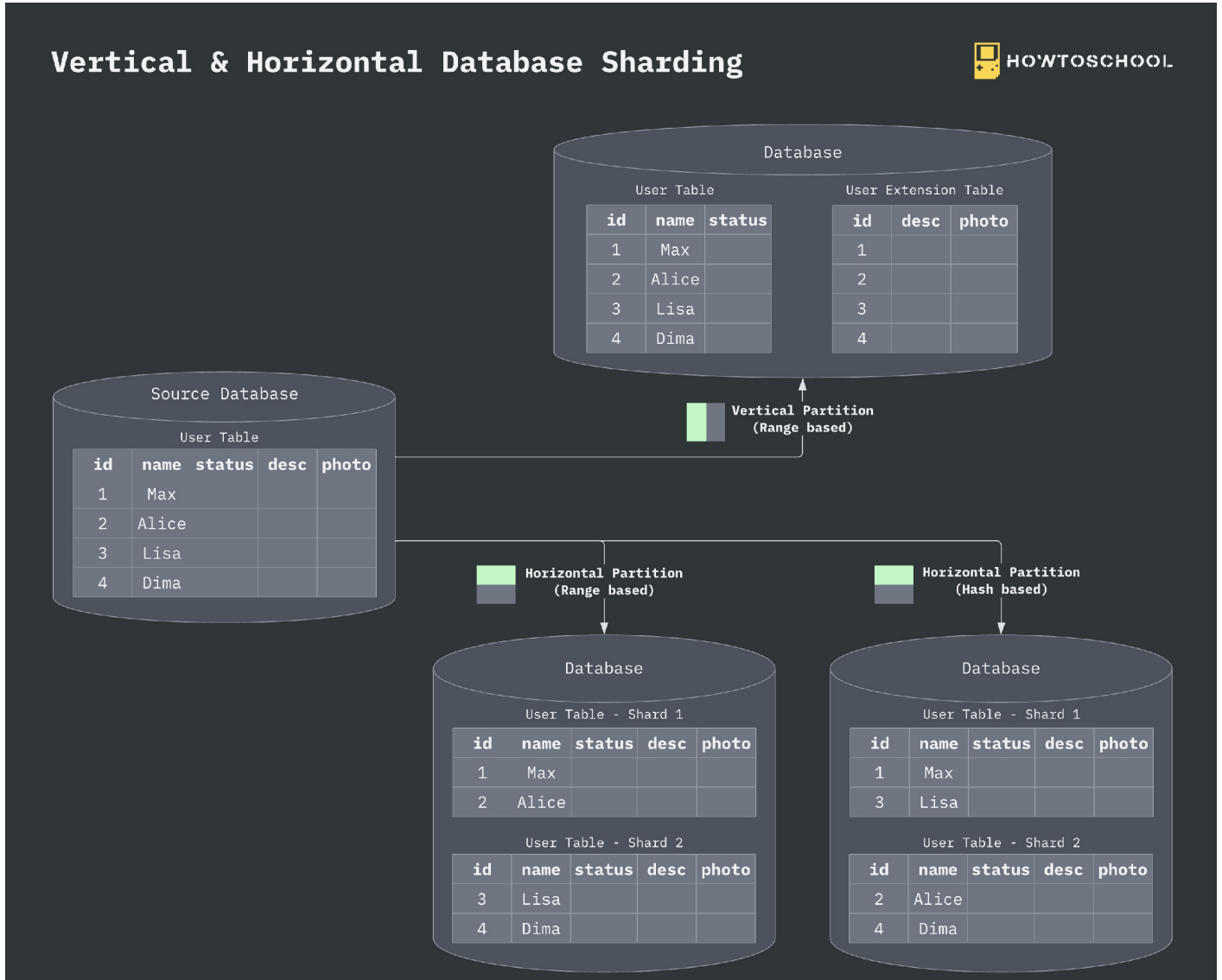
Распределение данных

- **Секционирование:** разделение данных на логические части для улучшения масштабируемости и производительности системы.
- **Избыточность и репликация:** дублирование элементов системы для повышения надежности и копирование данных для обеспечения консистентности.

Секционирование данных

- **Вертикальное секционирование**: разделение данных на разные смысловые части с возможным размещением на разных серверах.
- **Горизонтальное секционирование (шардирование)**: разбиение данных по ключу, например, дню недели или почтовому индексу, на уровне таблицы.

Секционирование данных



Методы секционирования записей:

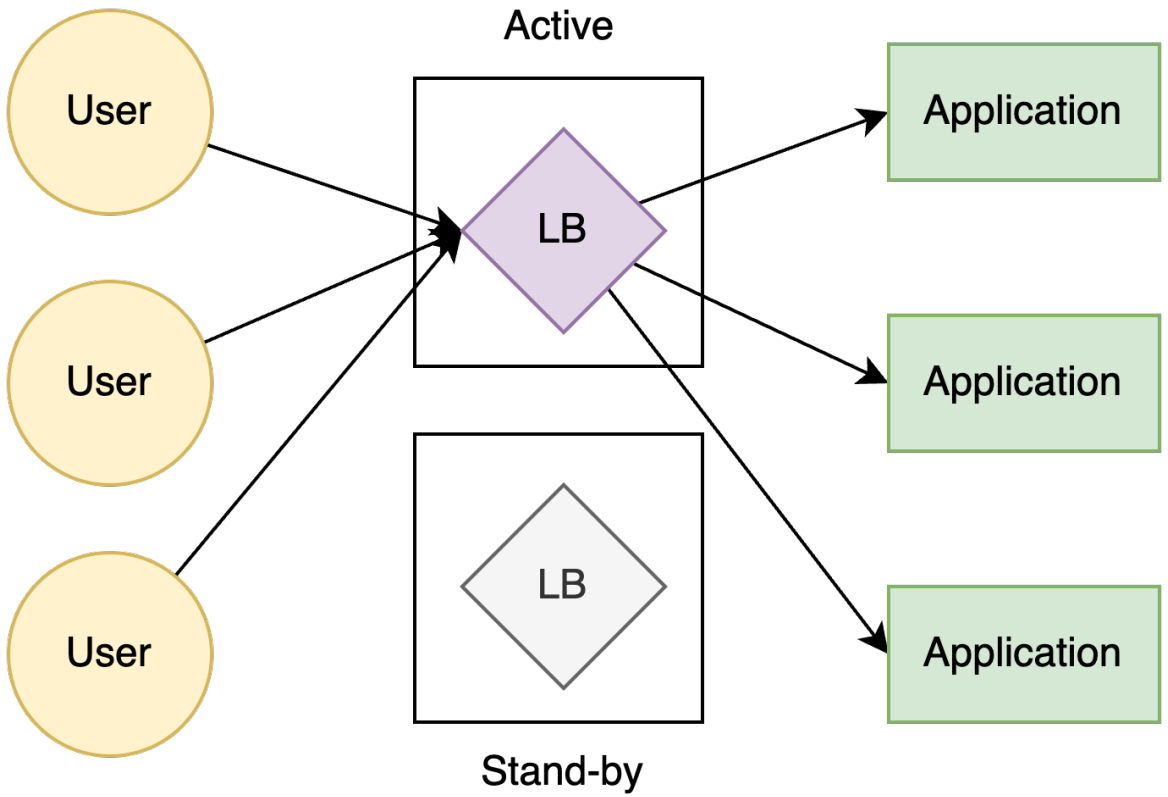
- **По ключу или хешу:** используем хеш-функцию для определения номера партиции на основе поля, например, времени создания записи. Недостатки в фиксированном числе серверов и возможном перекосе нагрузки.
- **По спискам:** предварительно распределяем записи по группам на основе различных параметров, таких как страна или день недели, для более равномерной нагрузки.
- **Round-robin:** циклически изменяем партиции с каждой новой записью.
- **Составное разбиение:** комбинируем вышеуказанные методы для более гибкой настройки.

Недостатки

- JOIN-ы становятся проблемой из-за данных на разных серверах и их сложности в выполнении, частично решаемая денормализацией базы данных.
- Большинство РСУБД не способны контролировать валидность внешних ключей на разных серверах.
- Ребалансировка требует изменения шардирования, если обнаружены проблемы, что приводит к пересылке данных и недоступности системы.

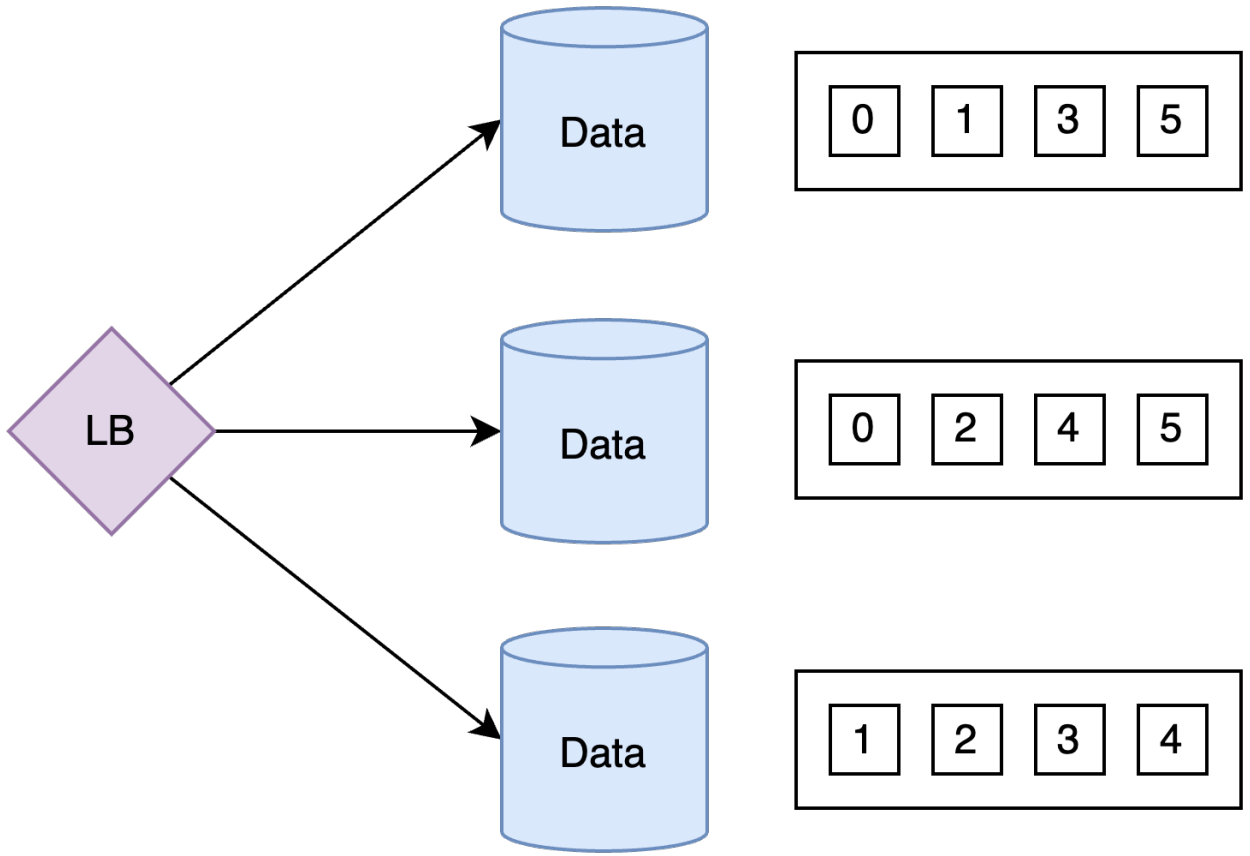
Избыточность и репликация

Избыточность



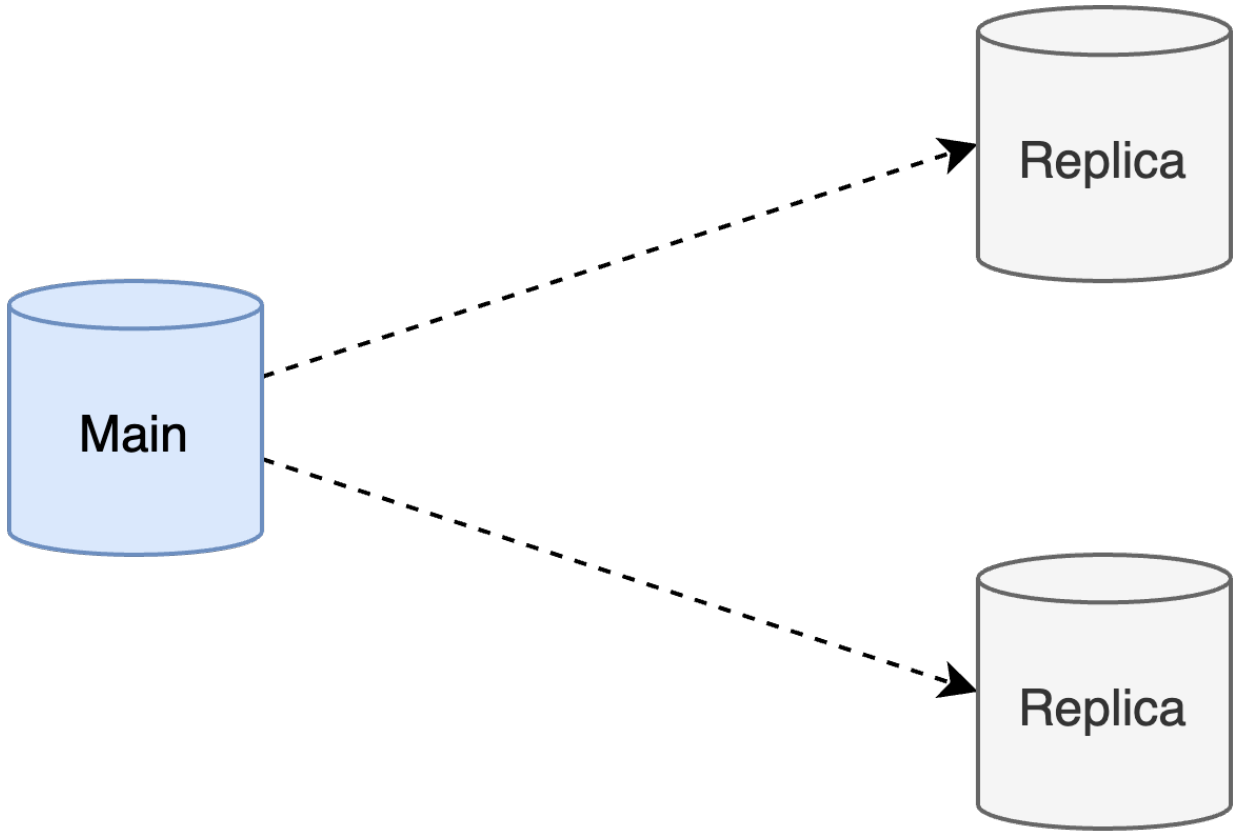
Резервный сервис в пару к действующему на отдельной машине

Избыточность



Резервный сервис в пару к действующему на отдельной машине

Репликация



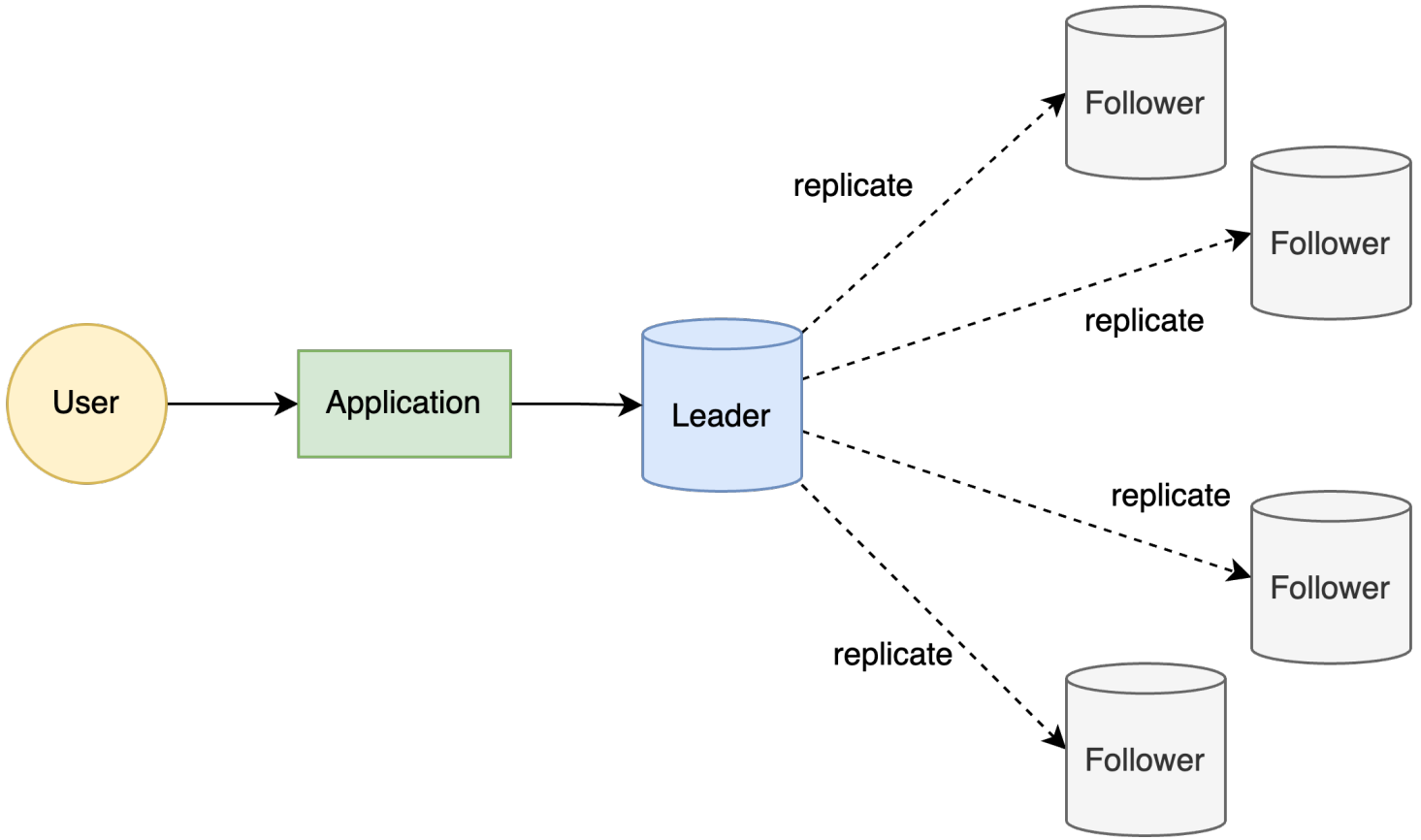
Репликация повышает надежность, устойчивость к отказам и доступность системы.

Кворум

Кворум - это минимальное количество узлов, необходимое для согласованности операций в сети. В случае отказа узлов, система может продолжать работу, если количество живых узлов превышает $2f + 1$. Операции чтения и записи требуют кворума для успешного выполнения, что обеспечивает консистентность данных, но может привести к временной недоступности сервиса.

Лидер и последователи

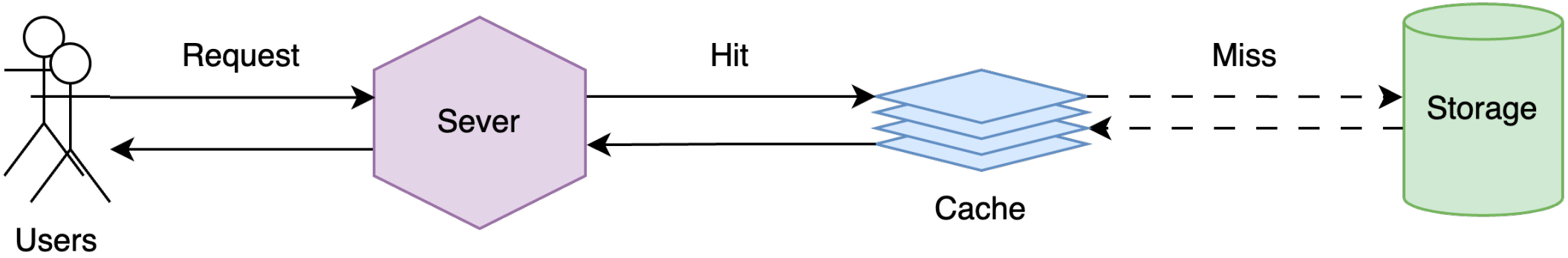
Лидер (Master): Отвечает за запись и обновление данных.
Последователи (Slaves): Получают копии данных от лидера и обеспечивают доступ к данным для пользователей.



Кэширование данных

Самый лучший способ ускорить вычисления – не совершать их вообще.

Кэширование



Кэширование и память

Кэш - быстрая память в компьютере, хранящая данные в нескольких уровнях (L1, L2, L3).

Данные читаются и записываются блоками с использованием тегов для их местоположения.

Поиск данных начинается с L1, затем L2, L3 и так далее, пока не будут найдены.

Если данные отсутствуют в кэше, они записываются для будущего быстрого доступа.

Попадание в кэш

- Попадание в кэш описывает ситуацию, когда содержимое успешно обслуживается из кэша.
- Быстрый поиск тегов в памяти.
- Найденные данные считаются успешным попаданием в кэш.

Типы Кэша

- Горячий кэш - это самая быстрая память, в которой данные читаются с максимальной скоростью, например, из L1.
- Холодный кэш - это медленная память, где данные читаются с более низкой скоростью, такая как L3 или ниже.
- Теплый кэш находится между горячим и холодным кэшем, например, в L2 или L3, и обеспечивает среднюю скорость чтения данных.

Промах кэша

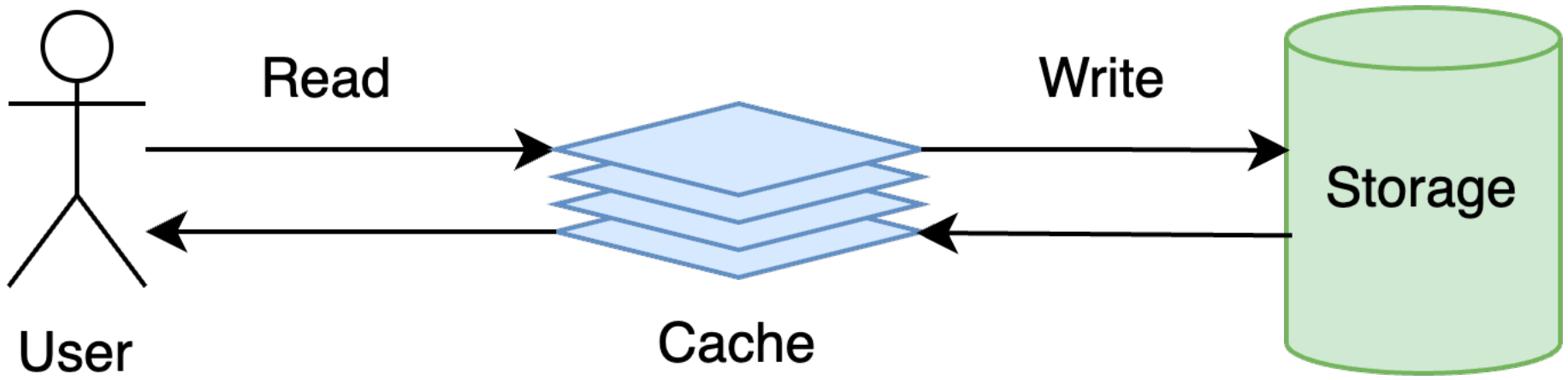
Данные не найдены в памяти, поэтому они передаются и записываются в кэш.

Инвалидация кэша

Инвалидация кэша - процесс удаления или замены недействительных записей в кэше компьютерной системы. Это важно для поддержания согласованного поведения приложений при изменении данных.

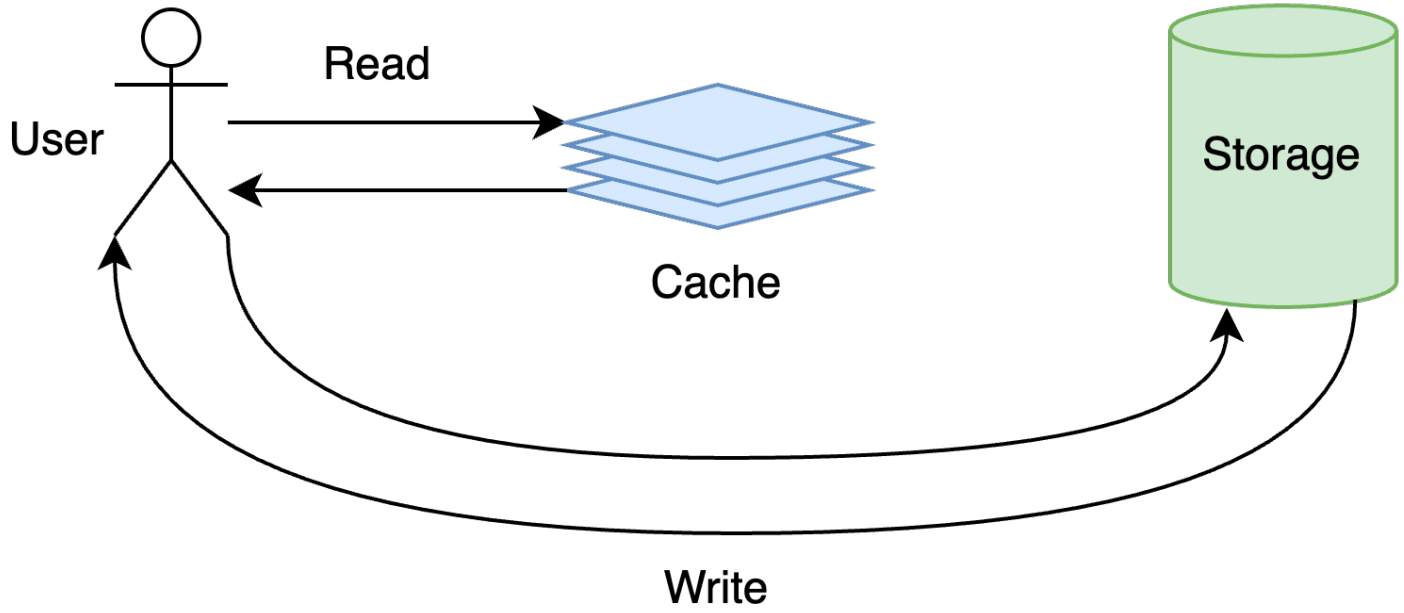
Кэш с прямой записью

Данные записываются одновременно в кэш и соответствующую базу данных.
Плюсы: Быстрое извлечение, полное соответствие данных между кэшем и хранилищем.
Минусы: Высокая задержка при операциях записи.



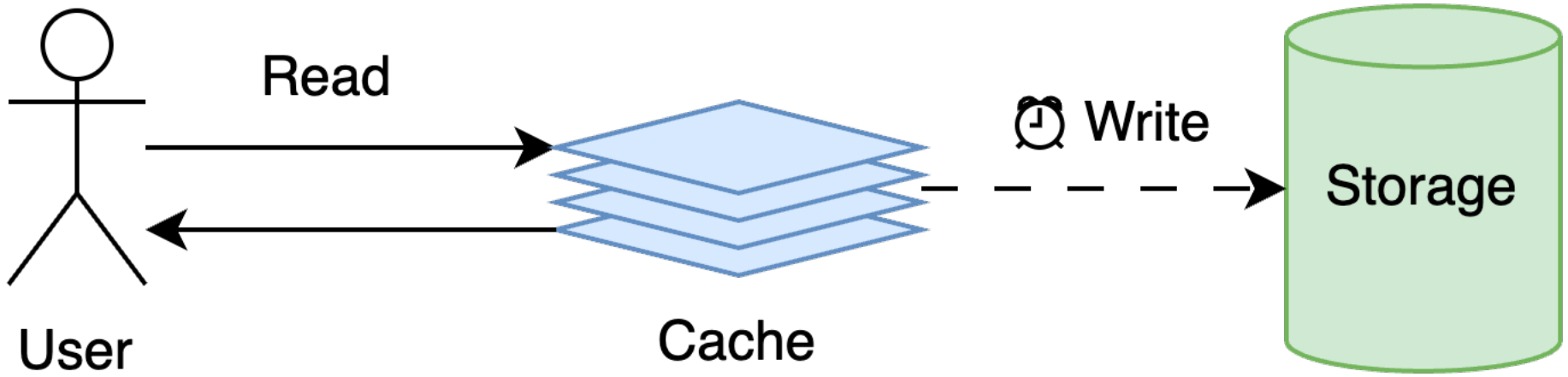
Кэш с прямой записью

Прямая запись в базу данных обходит кэш.
Плюсы: Уменьшает задержку.
Минусы: Увеличивает промахи кэша, требует чтения из базы данных при промахе кэша, что может увеличить время чтения из-за медленного бэк-энд хранилища.



Кэш с отложенной записью

Запись происходит в слое кэширования, подтверждаясь после завершения записи в кэше. Кэш асинхронно синхронизирует данные с базой данных.
Плюсы: Уменьшение задержек и высокая пропускная способность для приложений с интенсивной записью.
Минусы: Риск потери данных при сбое кэша. Улучшение возможно с несколькими репликами, подтверждающими запись в кэш.

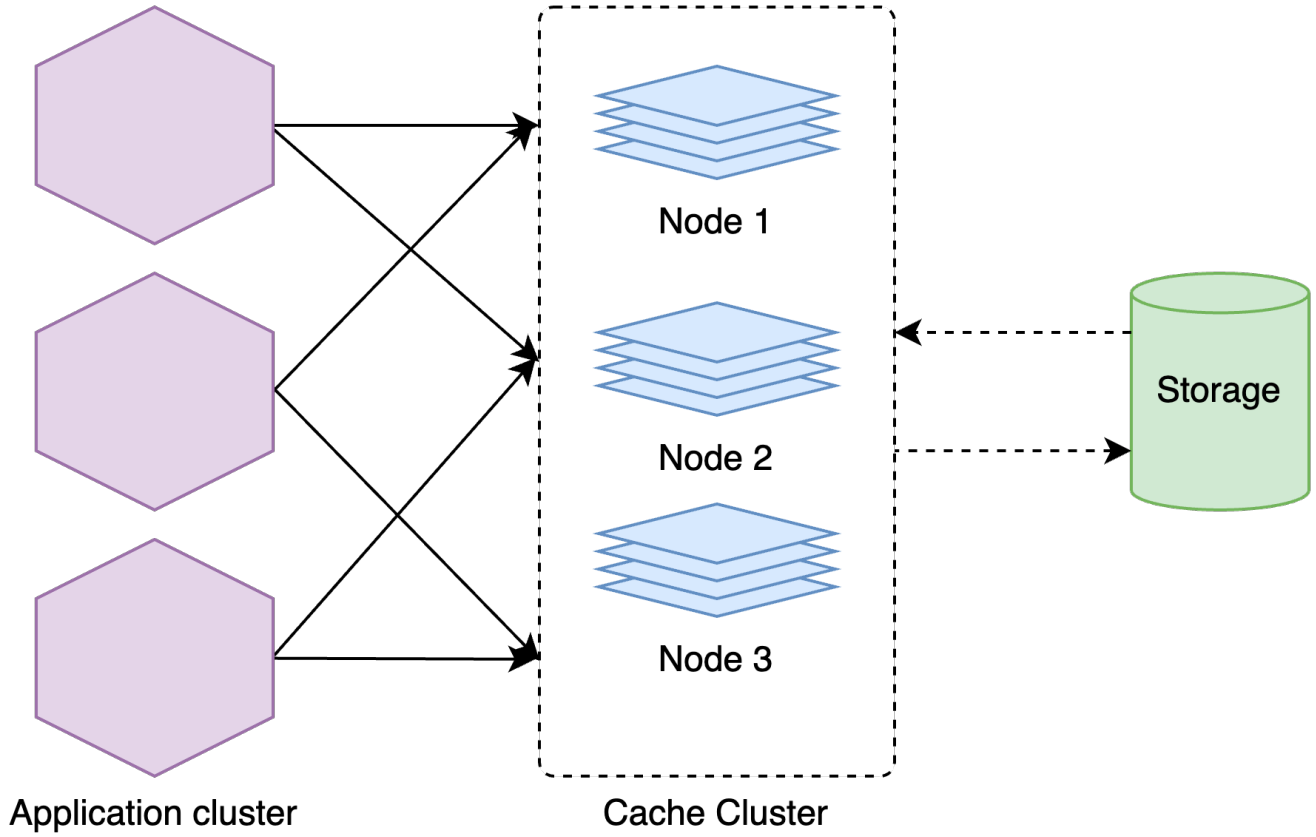


Популярные политики вытеснения кэша

- **Первым Пришёл — Первым Ушёл (FIFO)** : Кэш вытесняет первый доступный блок без учёта того, насколько часто или сколько раз он был доступен ранее.
- **Последним Пришёл — Первым Ушёл (LIFO)** : Кэш вытесняет последний доступный блок без учёта того, насколько часто или сколько раз он был доступен ранее.
- **Наименее Недавно Использованные (LRU)** : Удаляет первыми элементы, которые использовались наименее недавно.
- **Наиболее Недавно Использованные (MRU)** : В отличие от LRU, первыми удаляет наиболее недавно использованные элементы.
- **Наименее Часто Использованные (LFU)** : Подсчитывает, насколько часто требуется элемент. Те, которые используются наименее часто, удаляются первыми.
- **Случайная Замена (RR)** : Случайным образом выбирает кандидата на удаление и удаляет его для освобождения места при необходимости.

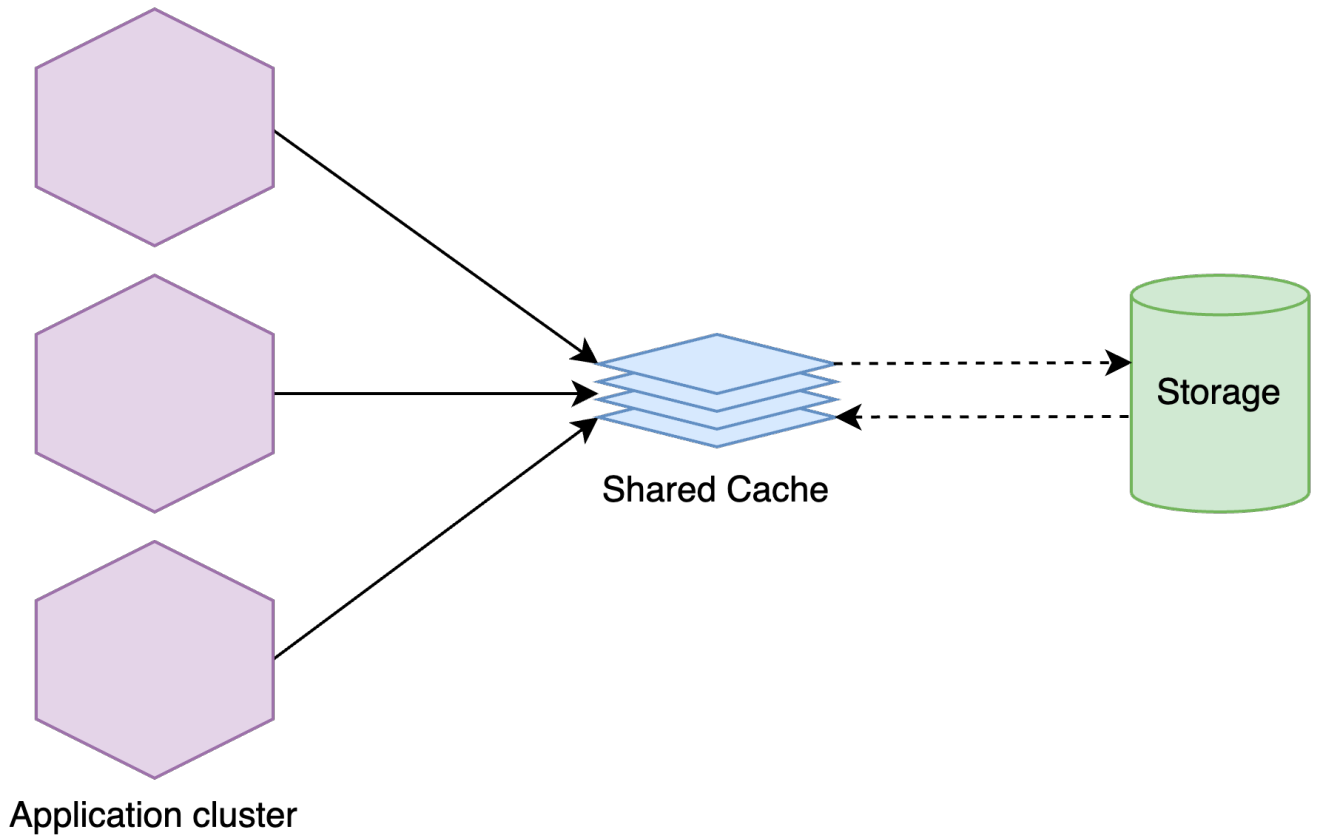
Распределённый Кэш

Распределённый кэш объединяет оперативную память нескольких компьютеров для быстрого доступа к данным, превосходя традиционные кэши, которые ограничены одним сервером или устройством.



Глобальный Кэш

Общий кэш для всех узлов. Если данные не в кэше, он найдет их в основном хранилище.



Примеры использования

- Кэширование баз данных
- Сеть доставки контента (CDN)
- Кэширование системы доменных имён (DNS)
- Кэширование API

Когда не использовать кэширование?

- Если доступ к кэшу занимает столько же времени, сколько и доступ к основному хранилищу данных.
- При низком повторении запросов, так как эффективность кэширования зависит от повторяющихся паттернов доступа.
- Если данные часто изменяются, так как кэшированная версия может выйти из синхронизации с основным хранилищем.

Кэш не следует использовать как постоянное хранилище данных, так как он реализуется в летучей памяти для повышения скорости доступа.

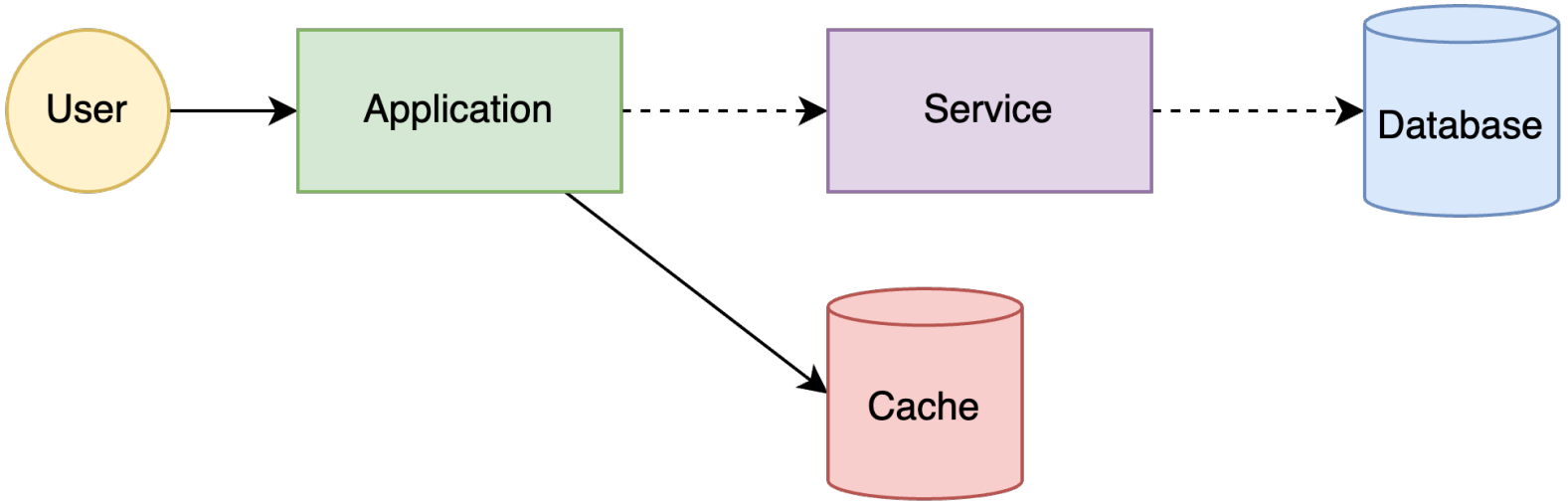
Преимущества

- Повышение производительности
- Снижение задержек
- Уменьшение нагрузки на базу данных
- Снижение сетевых затрат
- Увеличение пропускной способности чтения

Повышение отзывчивости

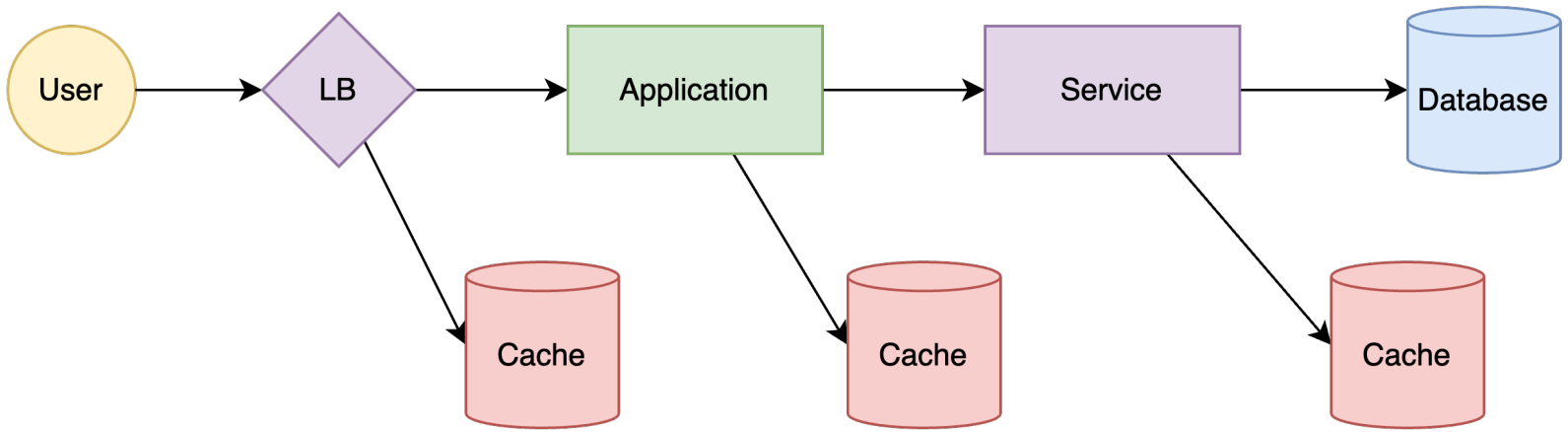
Кэширование данных

- Эффективность кэширования базируется на предположениях, что:
  - Более вероятно, что новые обращения будут происходить к недавним данным
  - Большая часть нагрузки приходится на малую часть запросов (принцип Парето 80 — 20)

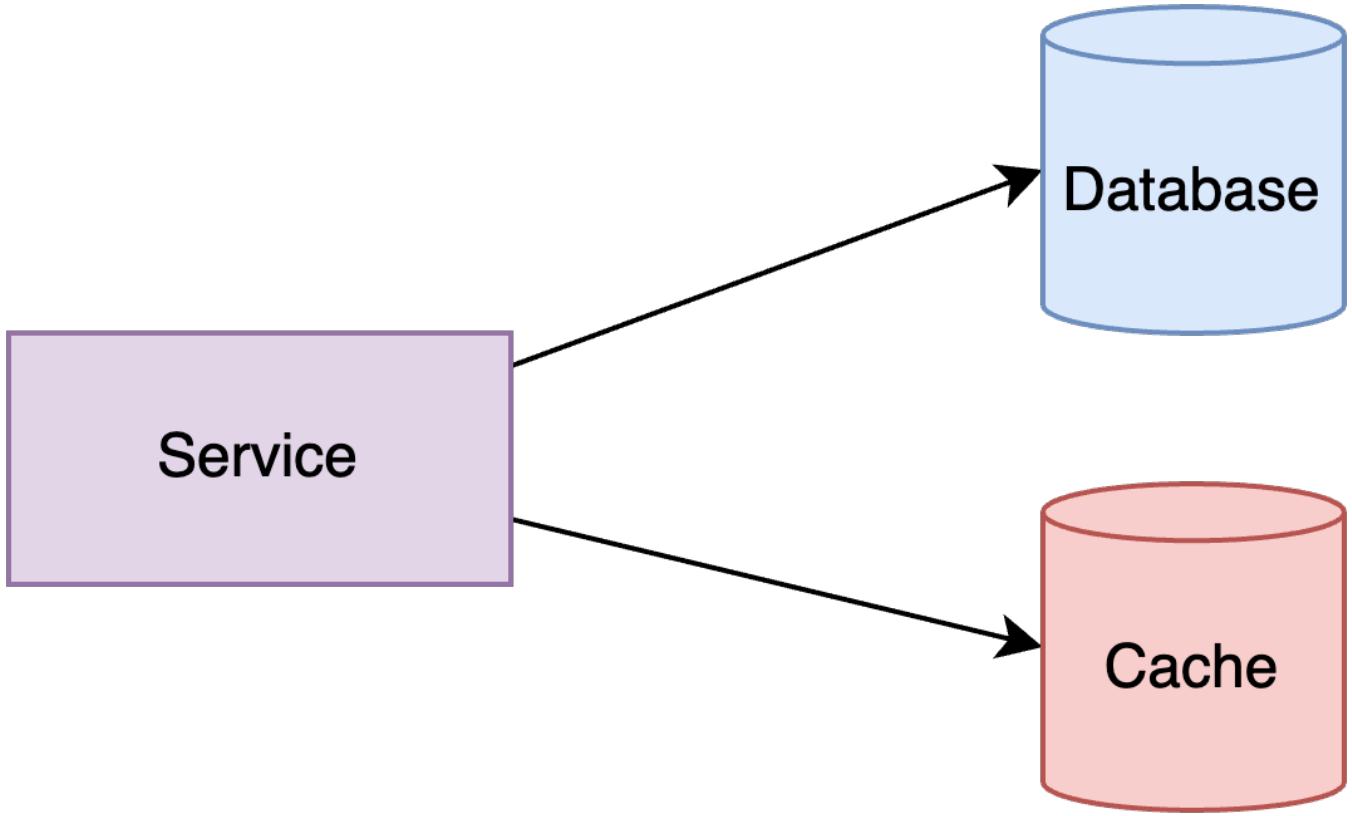


Располагаться на практически любом из уровней

- На запрос к фронтенду отдаем уже собранную страницу
- Для запрос на вычисления предоставляем сразу готовый ответ
- Для данных используем RAM вместо SSD, SSD вместо HDD и HDD вместосети

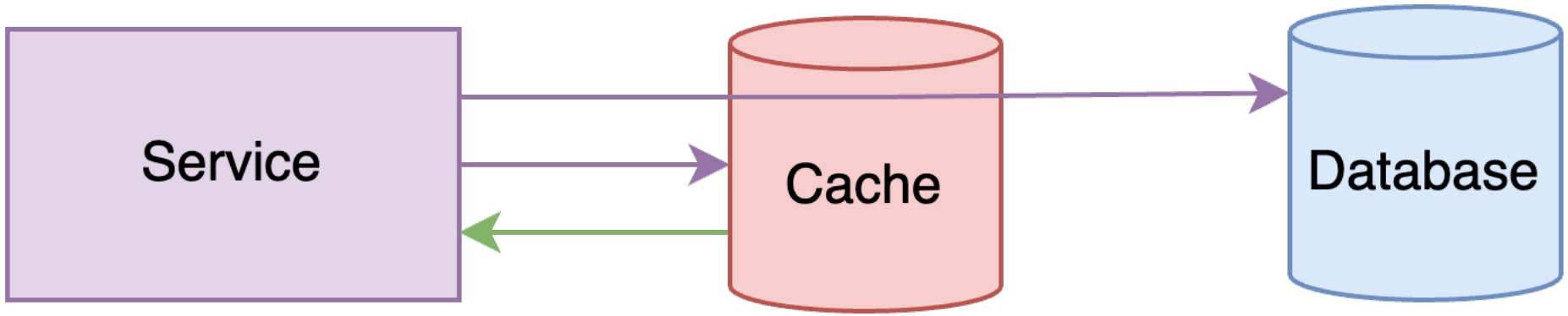


Инвалидация кэша



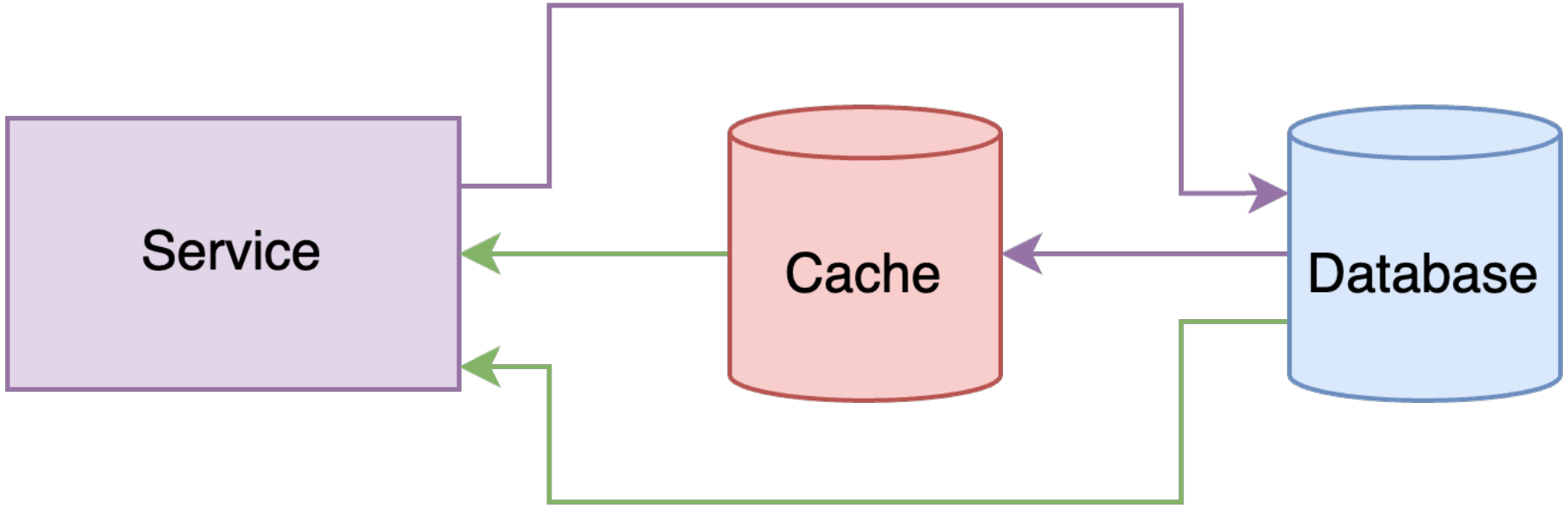
Сквозная запись

Данные сразу записываются и в источник, и в кэша.
Плюсы: гарантированная консистентность, минимизация потерь.
Минусы: запись становится даже медленнее из-за двух операций



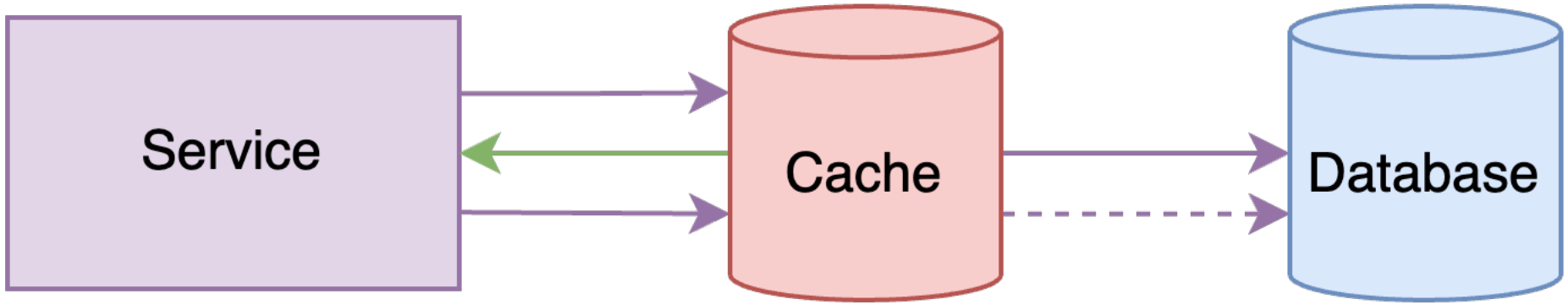
Запись в обход

Данные записываются напрямую в источника
Плюсы: не нагружаем кэш записью невостребованных данных
Минусы: для недавних данных считывать нечего, идем в источник



Реверсивная запись

Данные сначала записываются в кэша.
Плюсы: низкие задержки и высокая пропускная способность на запись
Минусы: можно потерять недавние данные, не продублированные висточник

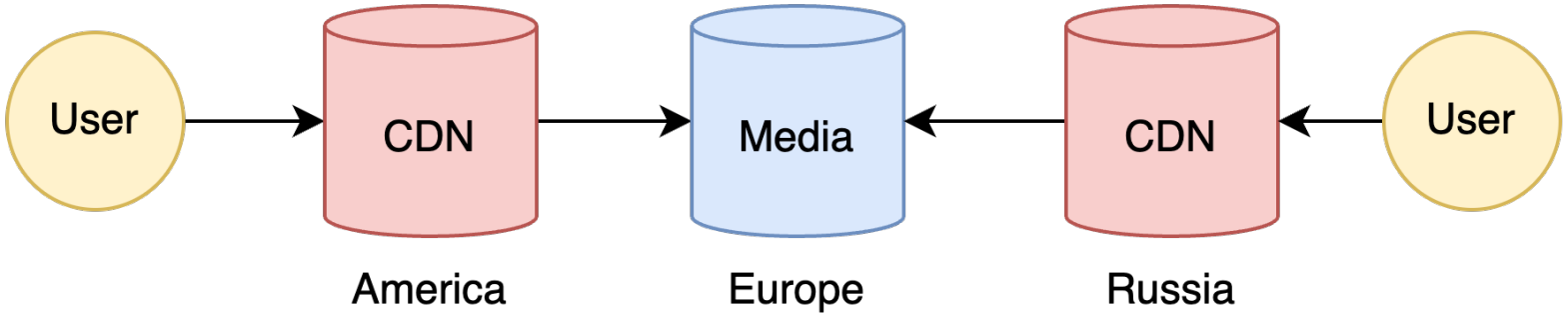


Вытеснение данных

- **First In First Out (FIFO):** удаляем самые давние записи несмотря на ихпопулярность
- **Last In First Out (LIFO):** удаляем самые свежие записи несмотря на ихпопулярность
- **Least Recently Used (LRU):** удаляем записи, не использованные дольшевсего
- **Most Recently Used (MRU):** удаляем самые недавние из использованныхзаписей
- **Least Frequently Used (LFU):** ведем счет обращений, удаляем самыенепопулярные
- **Random Replacement (RR):** удаляем случайно выбранные записи.

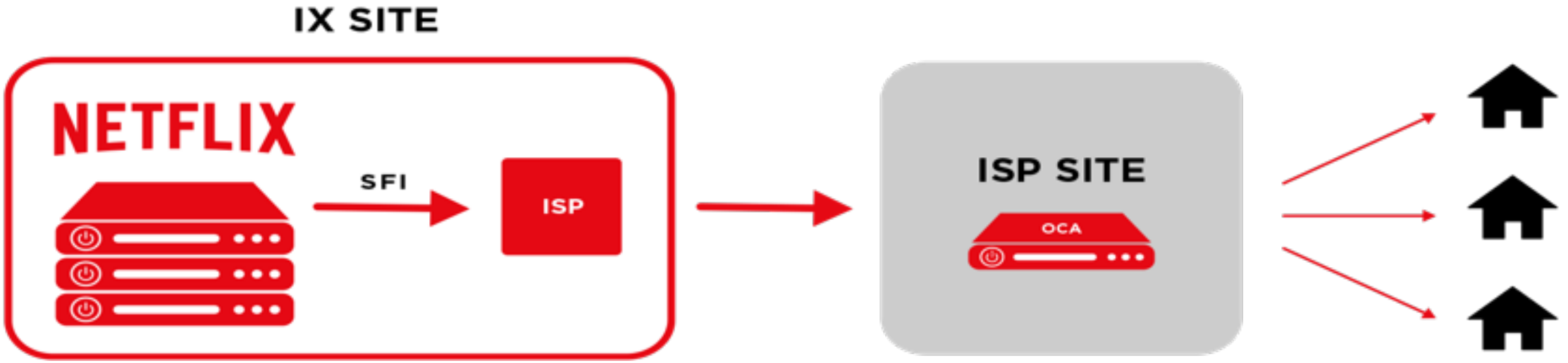
Content Delivery Network

CDN – это распределенные хранилища данных для крупных систем, обслуживающих различные регионы и даже весь мир. Используются для хранения и доставки часто запрашиваемых медиа файлов (изображений, звука, видео). При запросе файла CDN предоставит его непосредственно, если он доступен локально, иначе обратится к центральным серверам.



Content Delivery Network

Netflix предоставляет сервера провайдерам через инициативу Open Connect для улучшения опыта пользователей и оптимизации трафика.



Systems Design

Далее: Расчет требований для систем