АСИНХРОННОЕ ВЗАИМОДЕЙСТВИЕ СЕРВИСОВ

Синхронная обработка

- Простота реализации.
  - Безопасность: HTTPS, авторизация.
  - Content Negotiation, сжатие.
  - Балансировка.
- Предсказуемая скорость ответа.
- Прозрачность выполнения запроса.

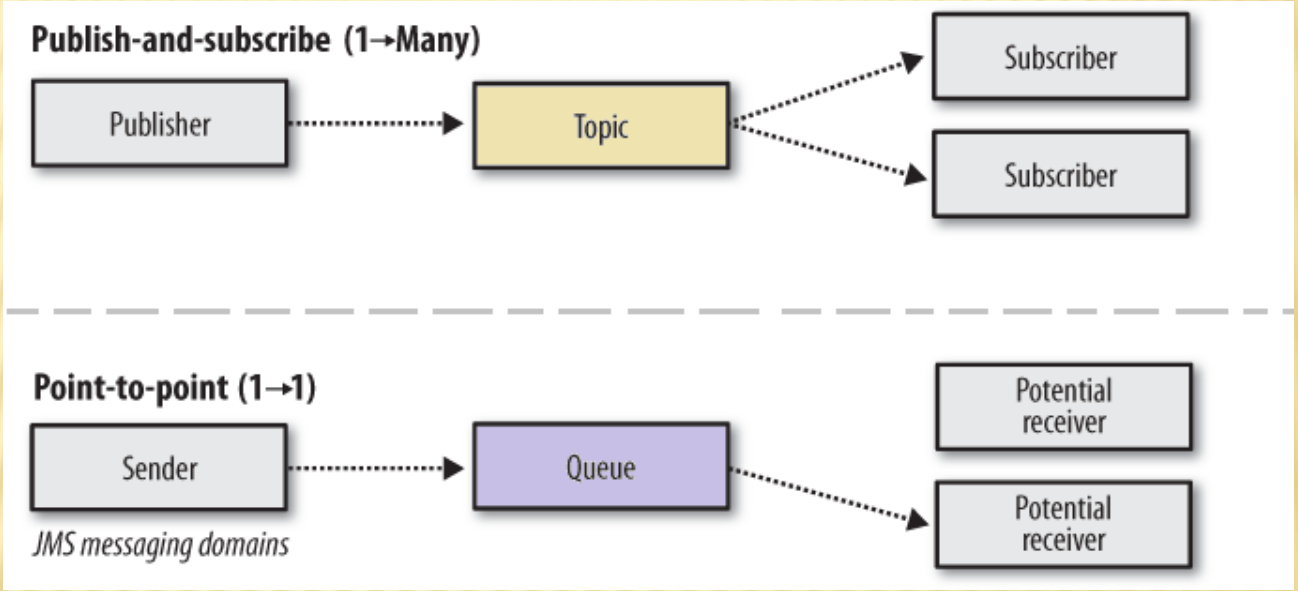
СИНХРОННАЯ ОБРАБОТКА

Проблемы	Решения
Нет гарантии, что ответ от сервера будет всегда быстрым, т.е. нужно предусматривать таймауты	Скорость ответа – устанавливается таймаут запроса, обычно он равен 2 * 95% line
Клиент и сервер должны быть активны в момент вызова	Использование Service Discovery, которое имеет актуальную инфомрацию о доступности сервиса. Так же использовать бесшовный деплой
Если сервер не смог с первого раза обработать запрос (был недоступен, таймаут и т.п.), то ответственность за повтор запроса лежит на клиенте	В случае если запрос завершился timeout'ом, то этот запрос можно выполнить еще раз, при этом нужно помнить про идемпотентность запросов
Пиковая нагрузка может привести к недоступности всей системы	Пиковая нагрузка не происходит внезапно, обычно это какие-то прогнозируемые события (например, черная пятница в интернет магазине), а значит к ним можно подготовиться и масштабировать приложение на несколько дополнительных машин

ОЧЕРЕДИ

Асинхронная обработка характеризуется тем, что запрос и ответ разнесены по времени. Существует целый класс однотипных задач, для которых принципиально порядок обработки. Для подобных целей используют инструмент, называемый очередями.

ТИПЫ ОЧЕРЕДЕЙ



BACKPRESSURE

В очередях есть важное понятие *Backpressure* – механизм, защищающий от перегрузки Consumer'а.

Есть несколько стратегий работы с Backpressure:

- Buffer – данные временно сохраняются, пока Consumer не будет готов их обработать.
- Drop – данные, которые Consumer не может обработать в текущий момент времени отбрасываются.
- Control – если Consumer не справляется со входным потоком, то:
  - отправляется команда на Producer на остановку публикации новых данных;
  - данные временно сохраняются в очереди, Consumer запрашивает эти данные, когда становится готов их обработать.

АСИНХРОННАЯ ОБРАБОТКА

- Сервер может быть недоступен в момент запроса.
- Система легко масштабируется, для увеличения пропускной способности достаточно просто добавить новых обработчиков, брокер сам начнет присылать им задания.
- Если возрастает нагрузка, то система не захлебнется запросами, т.к. есть backpressure.

ОТЛОЖЕННЫЕ ОПЕРАЦИИ

- Отложенная обработка: *агрегация данных и отправка их в сервис статистики раз в час*.
- Фоновый процесс: *отправка письма о регистрации пользователя в фоновом потоке*.

Проблемы асинхронной обработки

- Клиент и сервер должны гарантировать доставку и успешную обработку сообщений;
- Каждый запрос становится заявкой и должен иметь персистентное состояние, чтобы гарантировать нахождение системы всегда в детерминированном положении. такую систему сложно мониторить, т.к. недоступность или ошибка в системе может выливаться лишь в увеличении очереди;
- Система не прозрачна с точки зрения бизнес-процессов, т.к. синхронные процессы разрываются на асинхронные независимые этапы. Ее очень сложно проектировать и расширять.

Заявка. Жизненный цикл заявки

Заявка – единица работы, передаваемой для выполнения внешней системе.

- Уникальный ID на отправителе и получателе, чтобы можно было ее опознать.
- Начальное и конечные состояния.
- Параметры заявки.

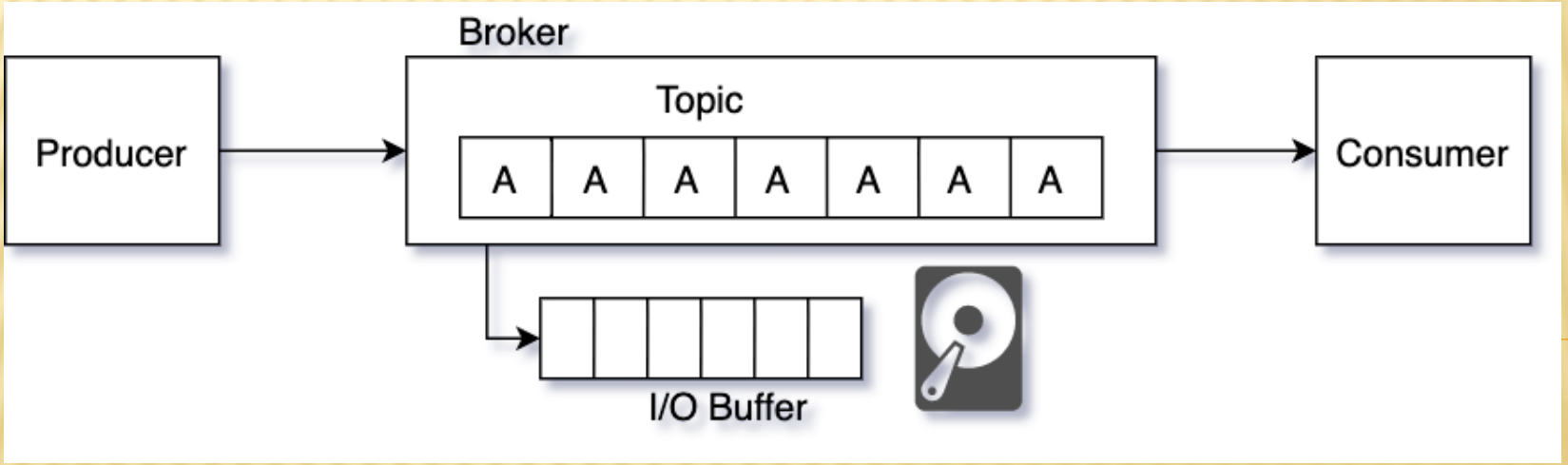
ПРОБЛЕМЫ ПРИ РАБОТЕ С ОЧЕРЕДЯМИ

Mathias Verraes
@mathiasverraes

There are only two hard problems in distributed systems: 2. Exactly-once delivery 1. Guaranteed order of messages 2. Exactly-once delivery

ПРОБЛЕМЫ ПРИ РАБОТЕ С ОЧЕРЕДЯМИ

- Не бывает потерянных сообщений.
- Не бывает дубликатов сообщений.



Некорректное выполнение операций

- Деградация функциональности.
- Повторная попытка.
- Отмена всей операции.
- Распределенная транзакция.

EVENT DRIVEN ARCHITECTURE ПРЕИМУЩЕСТВА

- Отправители и получатели независимы друг от друга.
- Нет интеграции "point - point", легко добавлять в систему новые объекты-получатели.
- Получатели могут реагировать на события сразу при их поступлении.
- Высокая масштабируемость и распределение.
- Подсистемы получают независимые представления потока событий.

EVENT DRIVEN ARCHITECTURE СЛОЖНОСТИ РЕАЛИЗАЦИИ

- Гарантированная доставка: в некоторых системах важно гарантировать доставку событий.
- Обработка событий в строгом порядке и (или) строго один раз.
- Очень сильно усложняется откат операций.
- Непрозрачность бизнес-операций.
- Дублирование данных.

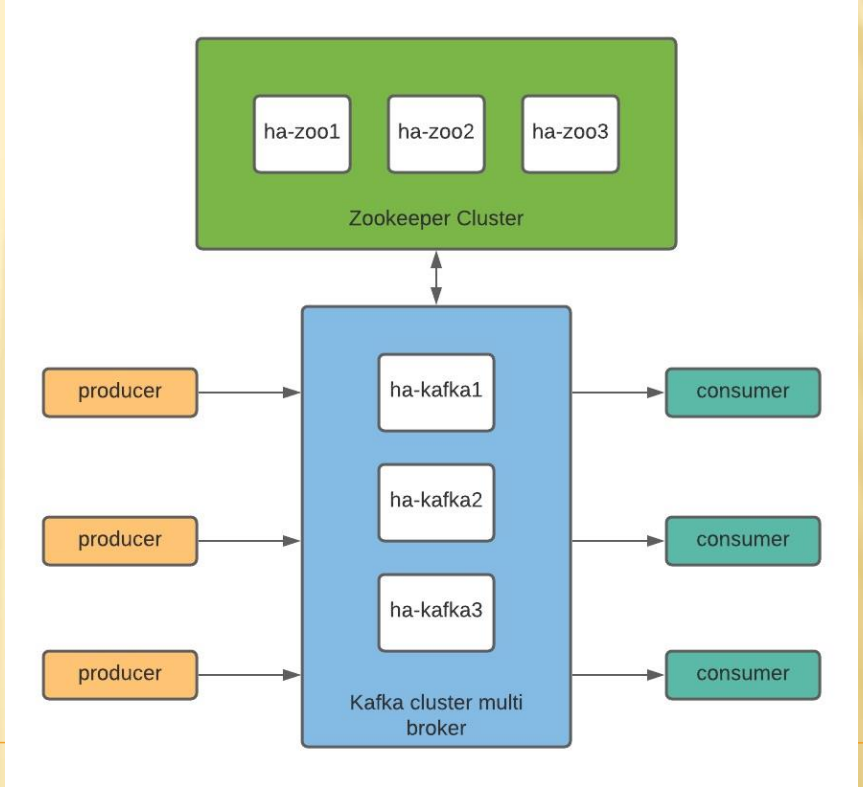
EVENT DRIVEN ARCHITECTURE ИСПОЛЬЗОВАНИЕ

- Обработка одних и тех же событий несколькими системами.
- Обработка в режиме реального времени с минимальными задержками.
- Обработка сложных событий, например сопоставления шаблонов или статистической обработки за некоторый период.
- Большие объемы и скорости поступления данных.

KAFKA & ZOOKEEPER

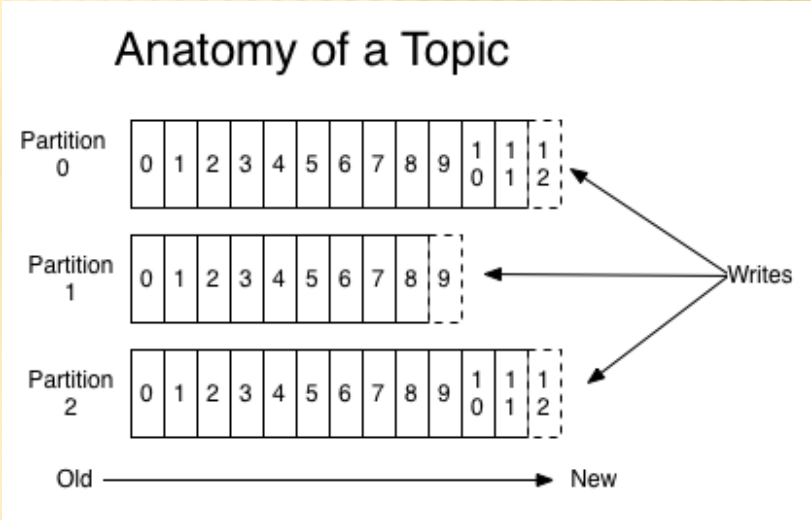
- Apache Zookeeper – это централизованная служба для поддержки информации о конфигурации, именования, обеспечения синхронизации распределенных приложений. Используется для координации внутри кластера Kafka.

- Apache Kafka – это распределенная система передачи сообщений.
  - Kafka может масштабироваться на чтение и на запись.
  - Для персистентности используется Append-only log.



ANATOMY OF TOPIC

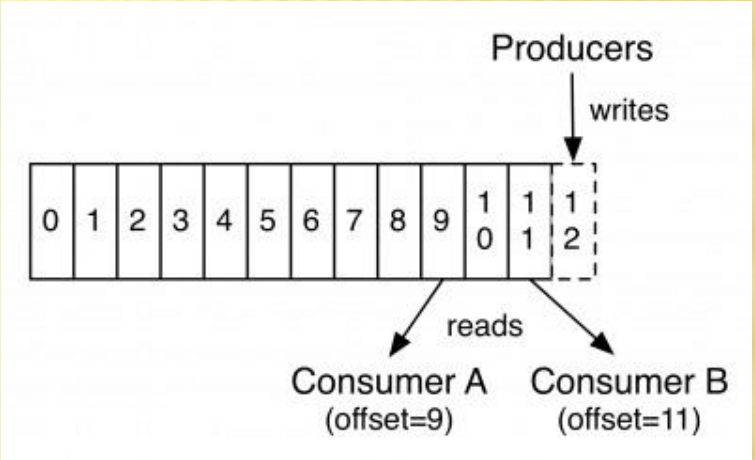
- Topic – поток сообщений определенного типа.
- Каждый topic делится на один или более патриции (partitions).
- Сообщения внутри патриции упорядочены по времени добавления и имеют уникальный идентификатор (уникальный внутри партиции), называющийся смещением (offset).



CONSUMER OFFSET

Каждый Consumer имеет своё смещение.

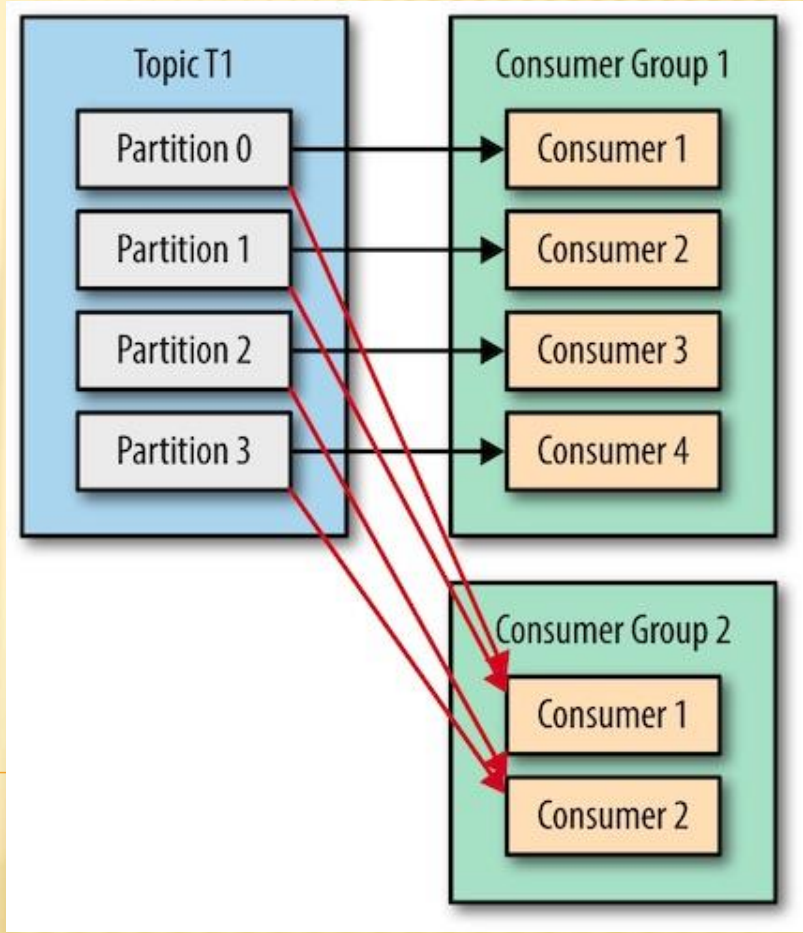
Эти смещения хранятся в служебных очередях внутри самой Kafka.



CONSUMER GROUPS

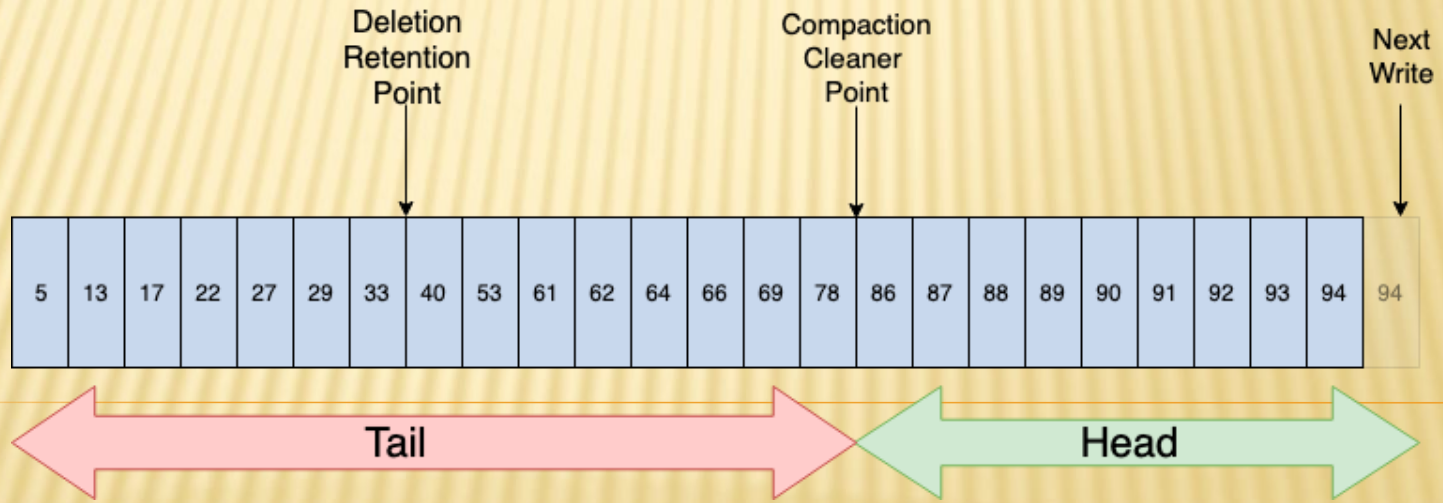
Consumer Groups – объединение нескольких потребителей в группу таким образом, чтобы каждое сообщение приходило одному потребителю.

За счет этого достигается масштабирование на чтение. Участники группы читают из разных партиций. Отсюда следует, что количество партиций должно быть не меньше, чем средний размер группы.

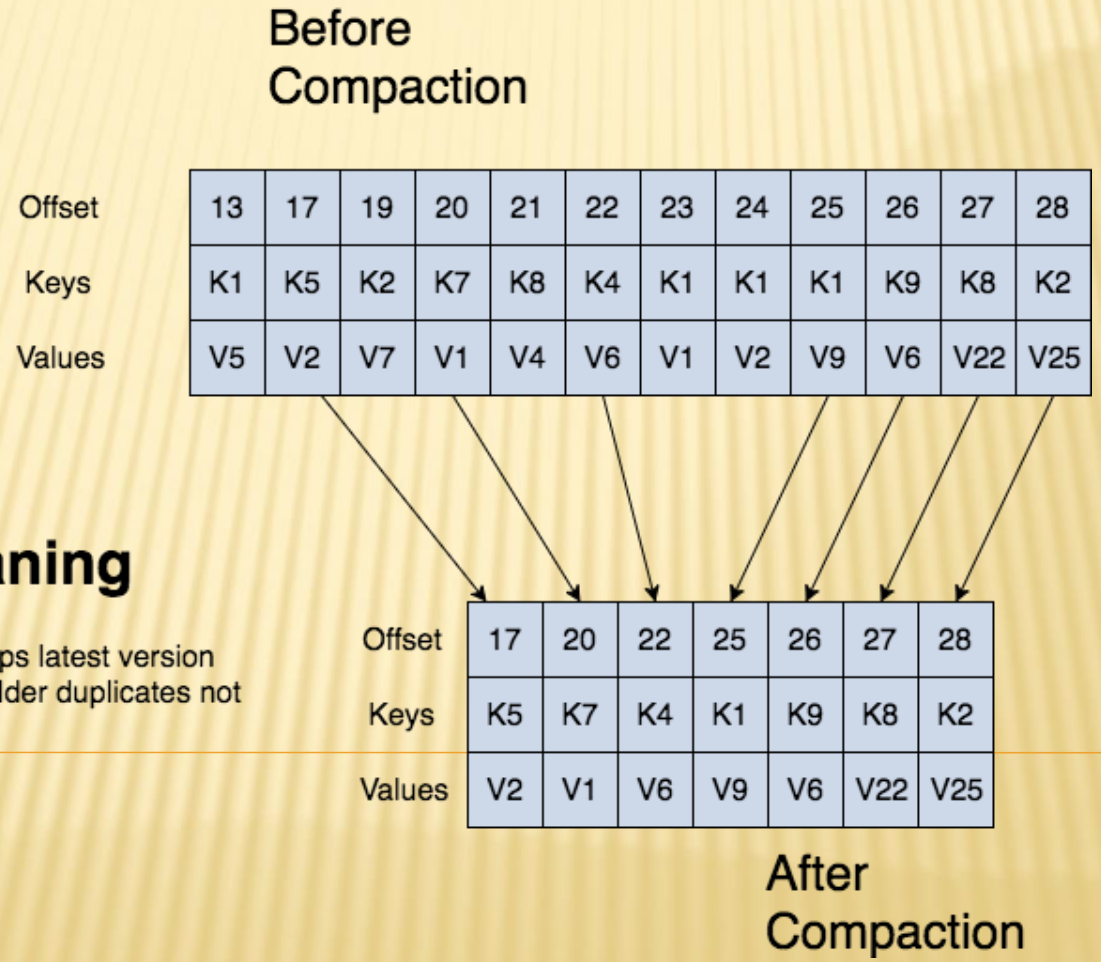


КОНТРОЛЬ ОБЪЕМА ТОПИКА

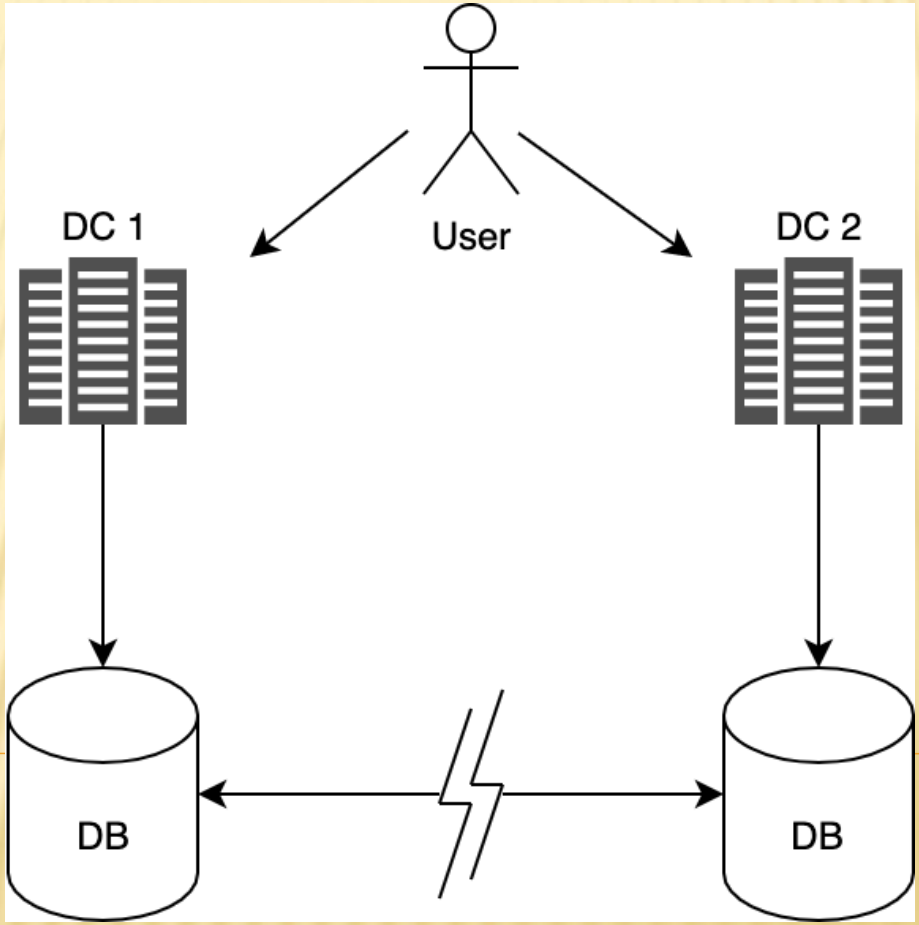
- Remove old items.
- Log compaction.



LOG COMPACTION



CAP ТЕОРЕМА



CAP ТЕОРЕМА

Суть теоремы CAP заключается в том, что в распределенных системах есть три компромиссных по отношению друг к другу свойства: **согласованность**, **доступность** и **терпимость к разделению** и при отказе удается сохранить только два из них.

- Согласованность (Consistency) является характеристикой системы, означающей, что при обращении к нескольким узлам будет получен один и тот же ответ.
- Доступность (Availability) означает, что на каждый запрос будет получен ответ не содержащий ошибок.
- Терпимость к разделению (Partition tolerance) является способностью системы справляться с тем фактом, что установить связь между её частями порой становится невозможно. Тут важно заметить, что под терпимостью к разделению имеется ввиду именно умение переживать сетевую недоступность.