Системный дизайн

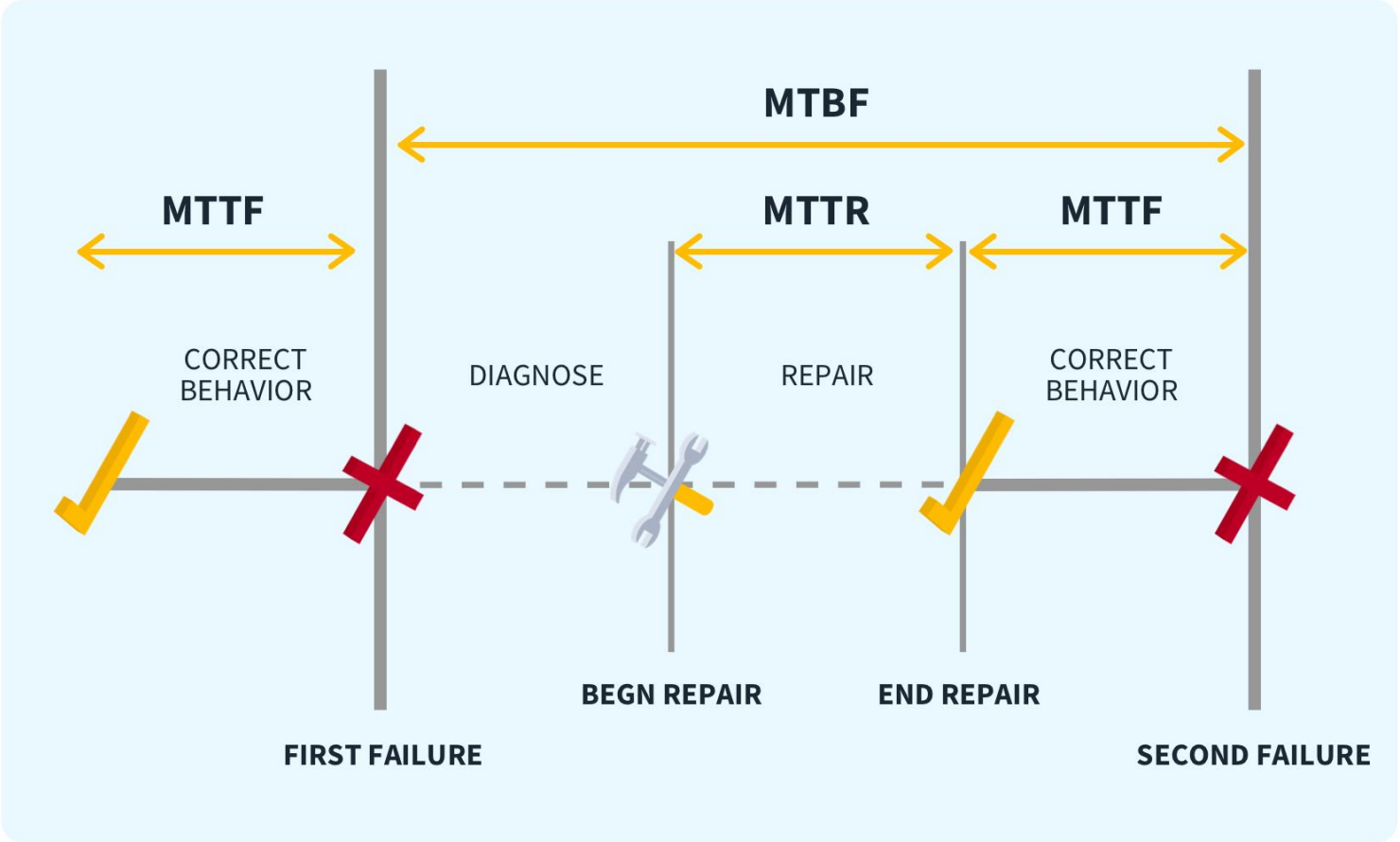
надежность и доступность

Доступность (availability)

определяется как свойство готовности системы к немедленному использованию.

В общем, это относится к вероятности того, что система работает правильно в любой момент и доступна для выполнения своих функций от имени своих пользователей. Другими словами, система высокой доступности – это система, которая, скорее всего, будет работать в данный момент времени.

Авария



https://www.cbtnuggets.com/blog/technology/networking/what-is-mean-time-to-repair-mttr

Подсчет времени доступности

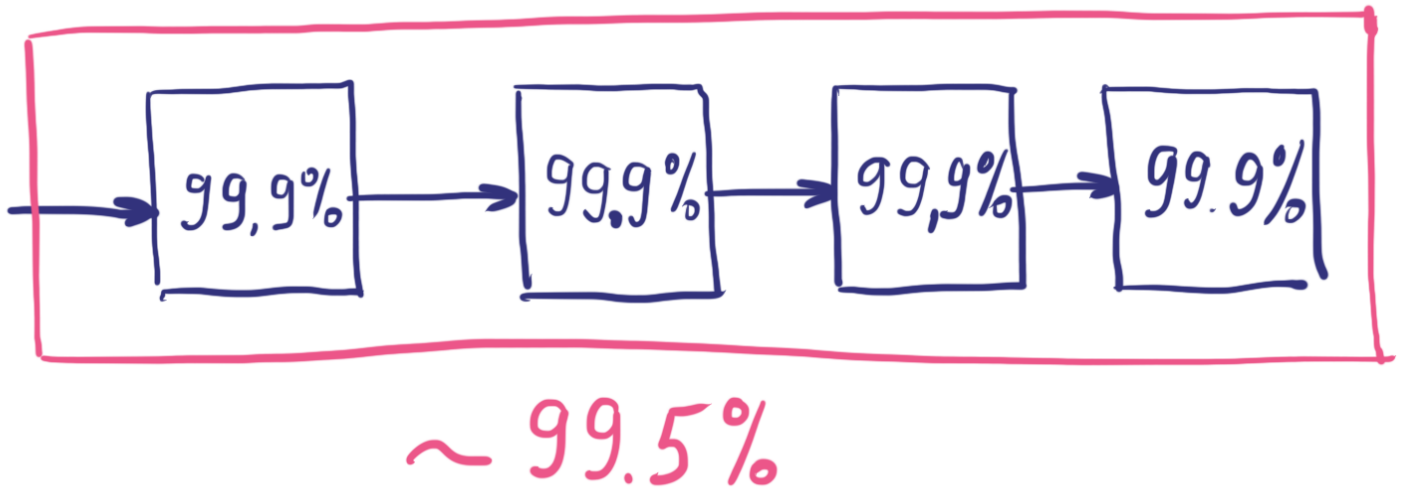
- MTBF - это среднее время между сбоями, которое является средним временем между двумя сбоями системы.
- MTTR - это среднее время восстановления, то есть среднее время устранения неполадок и восстановления системы до ее рабочего состояния. При расчете времени простоя системы учитываются только незапланированные простои.

$$Availability = \frac{MTBF}{(MTBF + MTTR)}$$

Пример

Availability	Downtime per year	Downtime per month	Downtime per week
99.0%	3.65 дня	7.2 часа	1.68 часа
99.9%	8.76 часа	43.2 минуты	10.1 минут
99.99%	52.6 минуты	4.32 минуты	60.5 секунд
99.999%	5.26 минут	25.9 секунд	6.05 секунд

Пример



Надежность (reliability)

относится к свойству, которое обеспечивает продолжительную работу системы без сбоев.

В отличие от доступности, надежность определяется с точки зрения временного интервала, а не момента времени. Высоконадежная система – это система, которая, скорее всего, будет продолжать работать без перерыва в течение относительно длительного периода времени.

Это тонкое, но важное отличие по сравнению с доступностью. Если система выходит из строя в среднем на одну, казалось бы, случайную миллисекунду каждый час, она имеет доступность более 99,9999 %, но все еще ненадежна.

Аналогичным образом система, которая никогда не выходит из строя, но закрывается на две недели в августе каждого года, обладает высокой надежностью, но ее доступность составляет всего 96 %.

Расчет надежности

Расчеты надежности, основанные на времени, учитывают общую продолжительность сбоев, включая время на **ремонт системы программного обеспечения.**

Он не различает, например, две ошибки по 30 минут и одну ошибку по часу, но вычисление, основанное на частоте успешных запросов, делает.

$$\frac{Successful\ Requests}{Total\ Requests}$$

ремонтопригодность (maintainability)

определяет, насколько легко может быть восстановлена отказавшая система.

Высокая ремонтопригодность системы показывает также высокий уровень доступности, особенно если отказ может быть обнаружен и устранен автоматически.

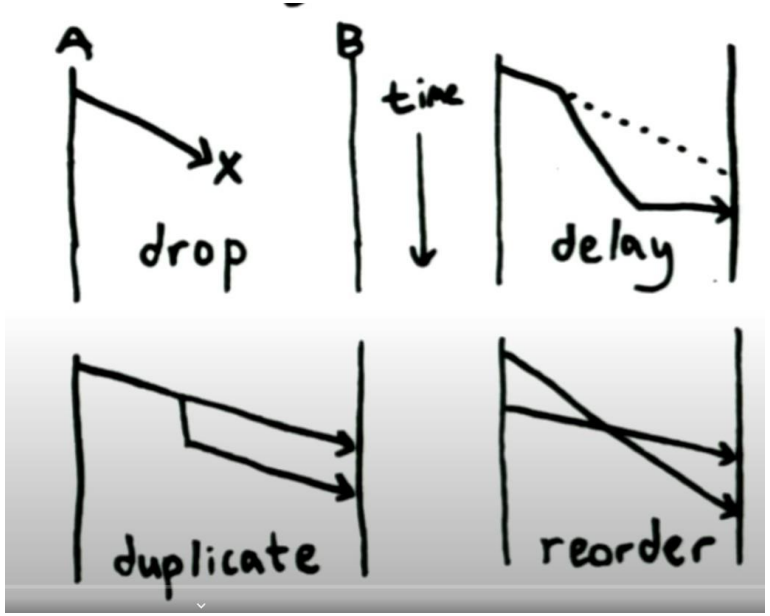
Типы отказов

- Что может пойти не так в распределенных системах?

Отказы сети

- Поскольку в общем случае сети на основе пакетной передачи данных являются асинхронными, то можно наблюдать

  - Произвольные задержки запросов
  - Произвольный порядок запросов
  - Дублирование одних и тех же запросов
  - Потеря запросов



- Потеря сетевой связности между узлами кластера, при этом доступность клиентского трафика (упал канал между ДЦ)

Split brain

Отказы оборудования или приложения

Сами узлы приложения могут переставать работать (отказывать) в произвольные моменты времени.

Византийские типы отказов

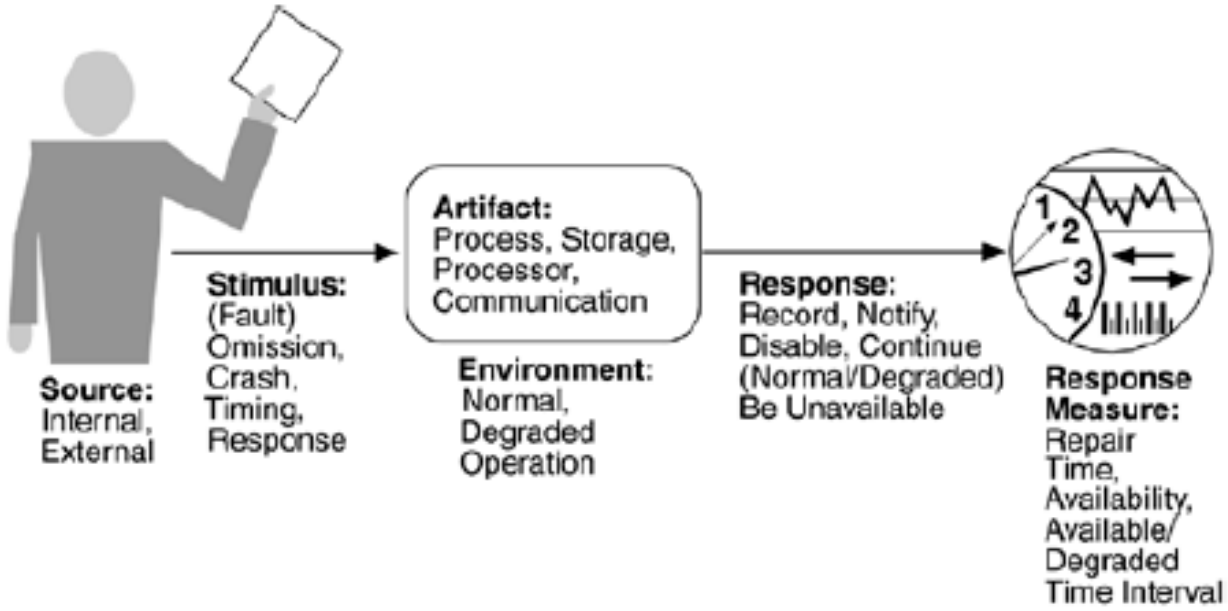
- Задача о византийских генералах - это проблема согласованности в распределенных системах, когда некоторые узлы могут быть ненадежными или даже злонамеренными.
- В этой задаче несколько генералов должны договориться о совместной стратегии атаки или обороны, используя только асинхронную коммуникацию между ними.
- Однако, если один или несколько генералов являются предателями, то они могут отправлять ложные сообщения и нарушать согласованность.
- Задача заключается в том, чтобы разработать алгоритм, который позволяет генералам достичь согласия при любом количестве предателей.

Stateless vs Statefull

- Отказоустойчивость **stateless** приложений в распределенных системах относительно просто осуществлять. Такие приложения горизонтально масштабируются, а в случае падения можно просто перезапустить и пересчитать задачу.

- Самые большие проблемы возникают со **stateful** приложениями. Когда приходится хранить состояние в нескольких узлах.

Как формируются требования к доступности?



Доступность Тактики

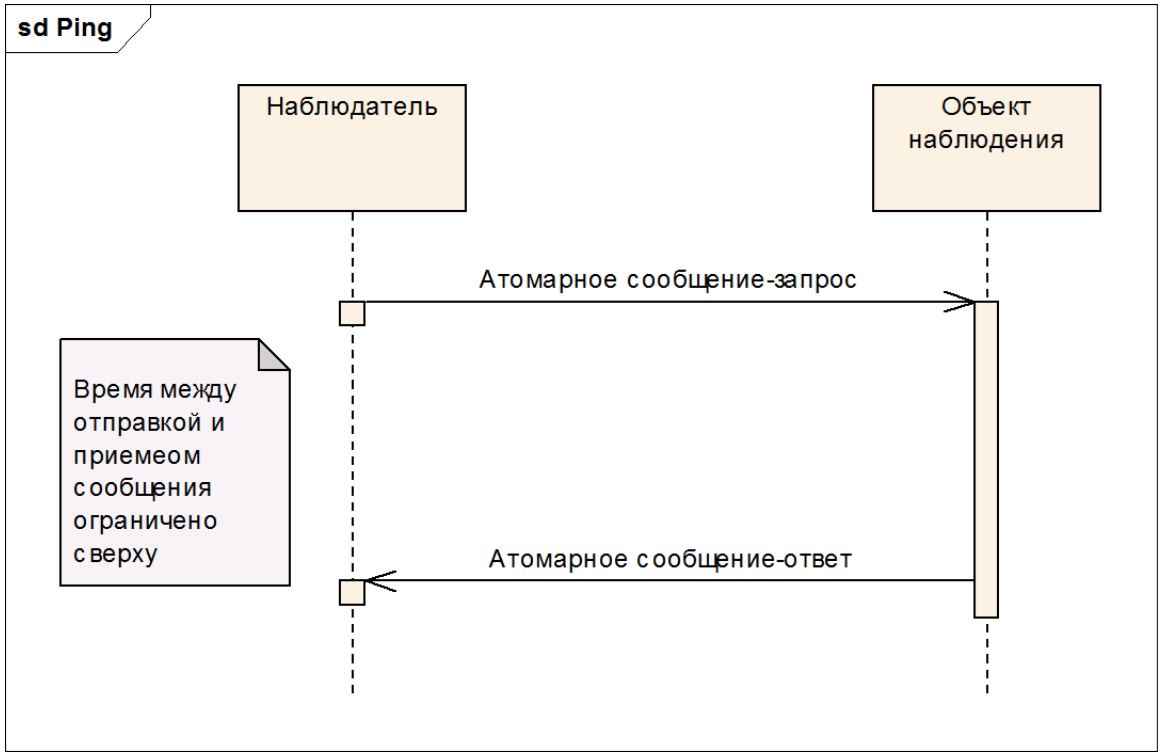
1. Обнаружение сбоя
2. Восстановление после сбоя
3. Предотвращения
4. Транзакции (недопущение сбоя)

1 обнаружение сбоя

Ping/echo обнаружение

- В этом методе обнаружения неисправностей компонент, выступающий в качестве системного монитора, отправляет эхо-запрос протокола управляющих сообщений Интернета (ICMP) другому компоненту (он проверяет связь с компонентом) и ожидает эхо-ответа ICMP.
- Если цель не отвечает на эхо-запрос в течение заранее определенного промежутка времени, компонент, действующий в качестве системного монитора, сообщает о сбое другого компонента.

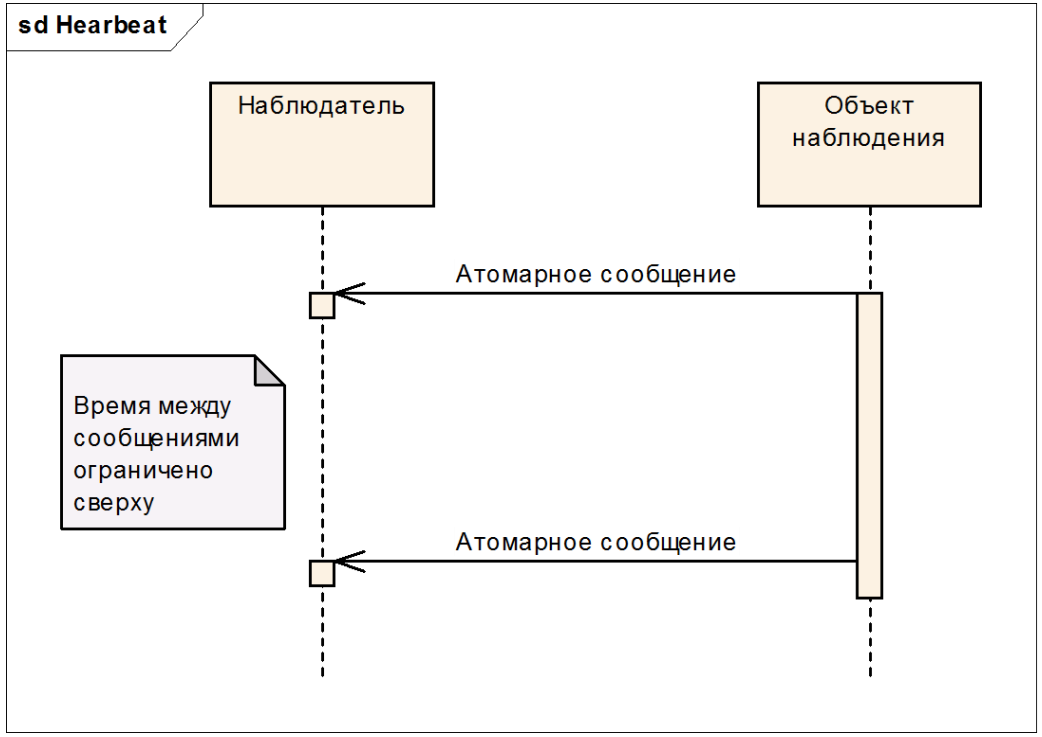
Ping/echo обнаружение

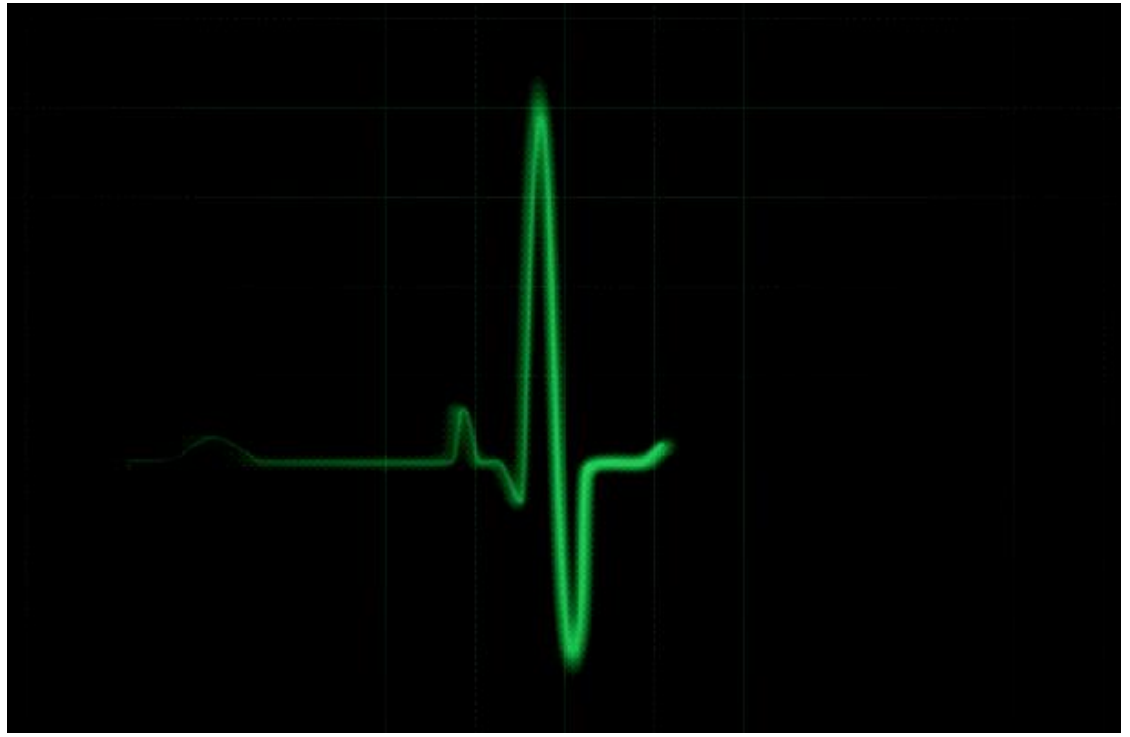


Heartbeat обнаружение

- Этот метод требует, чтобы один компонент периодически отправлял сообщение (пульс), чтобы указать, что он работает нормально.
- Если прослушивающий компонент не получает сообщение пульса в течение предварительно определенного периода времени, он определяет, что произошла системная ошибка, и предпринимает соответствующие действия.

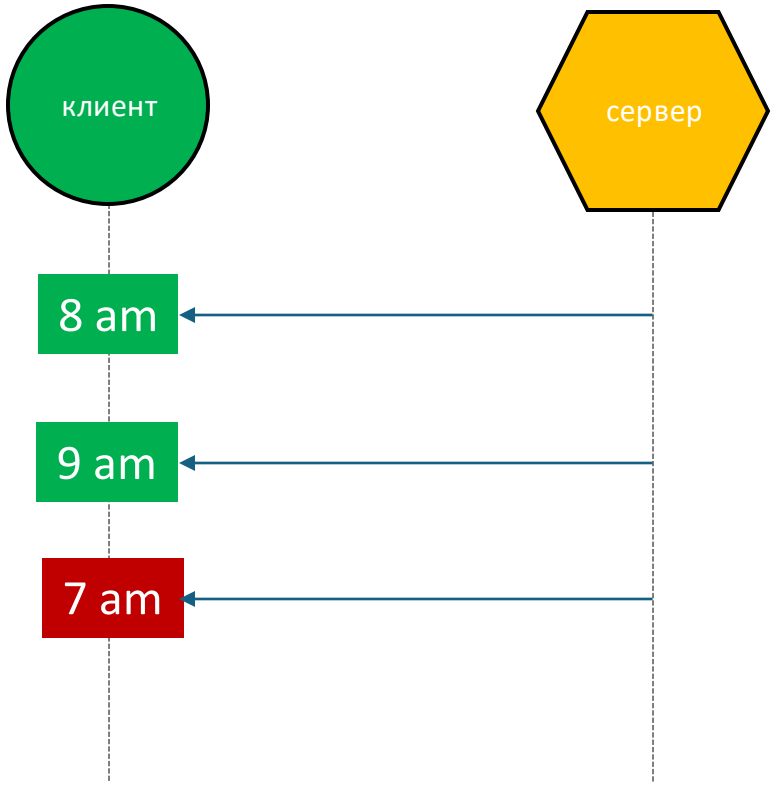
Heartbeat обнаружение





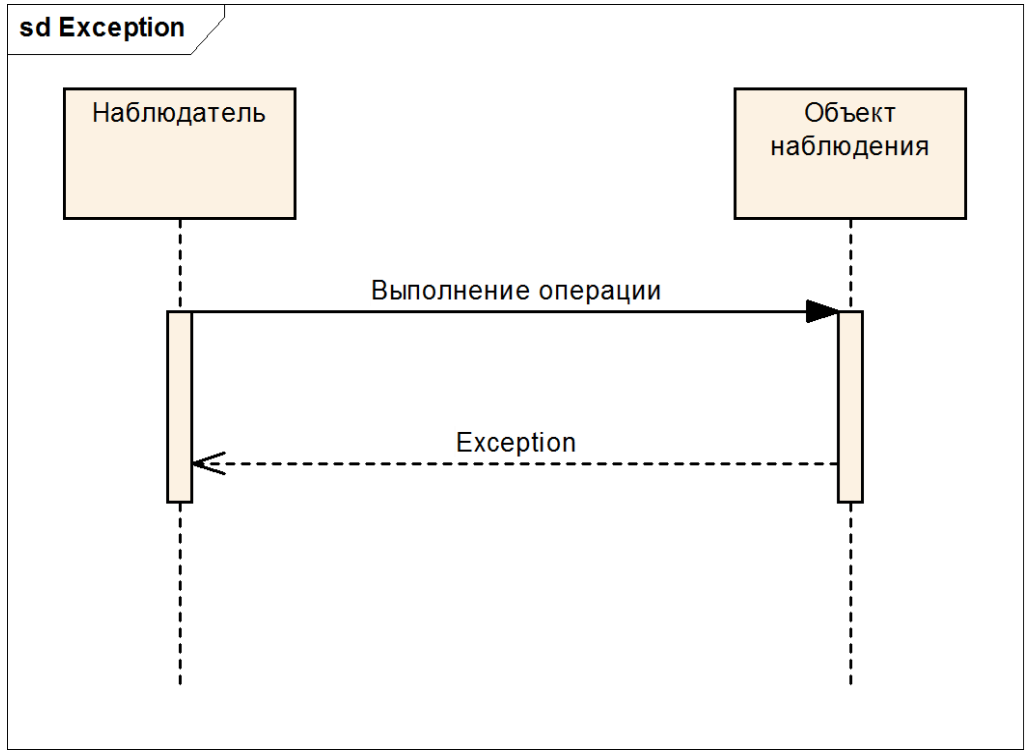
Timestamp обнаружение

- Эта стратегия обнаружения неисправностей направлена на выявление неверной последовательности событий.
- Используя временную метку или даже просто последовательность чисел, о неисправностях можно сообщить, если последовательность находится в неправильном порядке.



Exception обнаружение

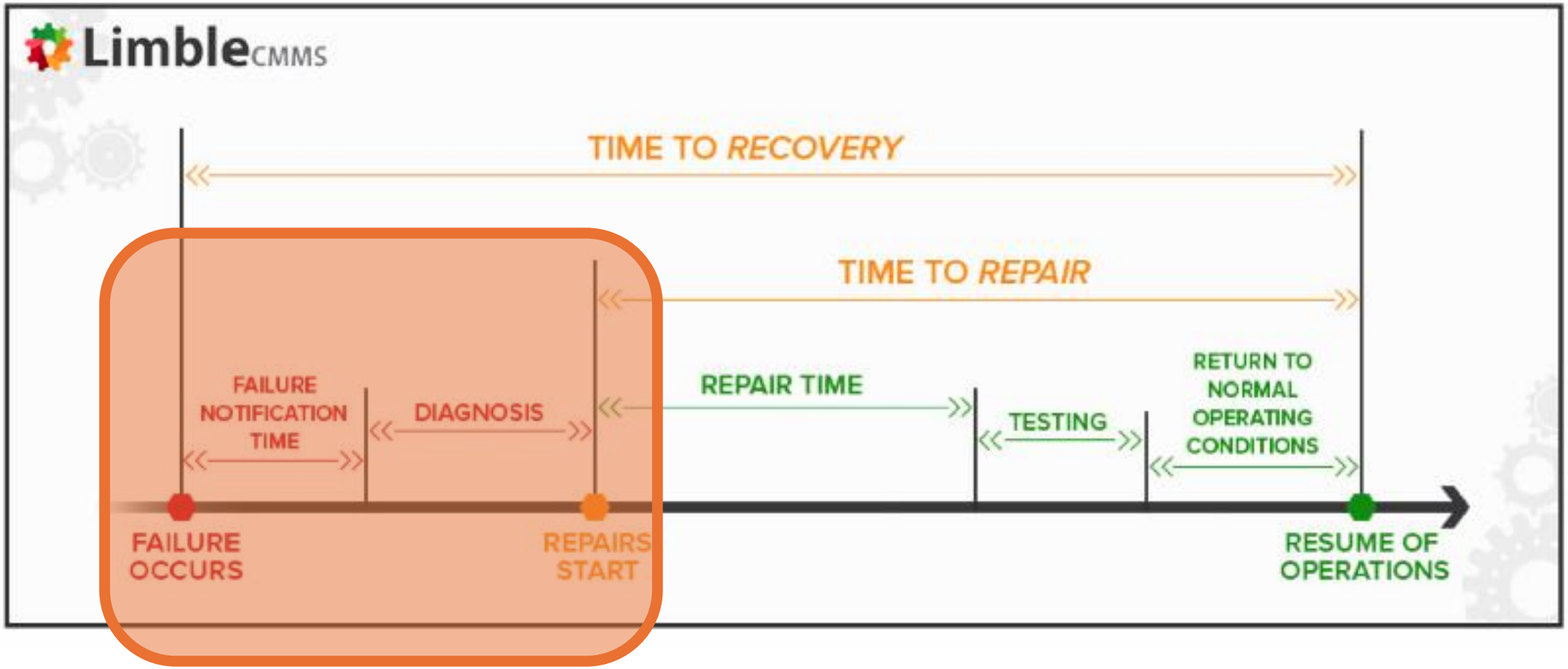
Эта стратегия обнаружения неисправностей требует анализа результата ответа от элемента. В случае если результат содержит код ошибки (в том числе и ошибки соединения), то обнаруживается сбой.



Мониторинг & Алертинг

- **Мониторинг** - это набор инструментов и сервисов, которые позволяют собирать, хранить, агрегировать и визуализировать метрики
- **Алертинг** – это набор инструментов, которые позволяют узнавать о каких-то важных изменениях в поведении системы (с помощью метрик)

Мониторинг и алертинг напрямую влияют на доступность availability сервисов



The Four Golden Signals

- **Latency** – время ответа
- **Traffic** - количество запросов
- **Errors** – количество ошибок
- **Saturation** – насколько ресурс близок к отказу

https://landing.google.com/sre/books/

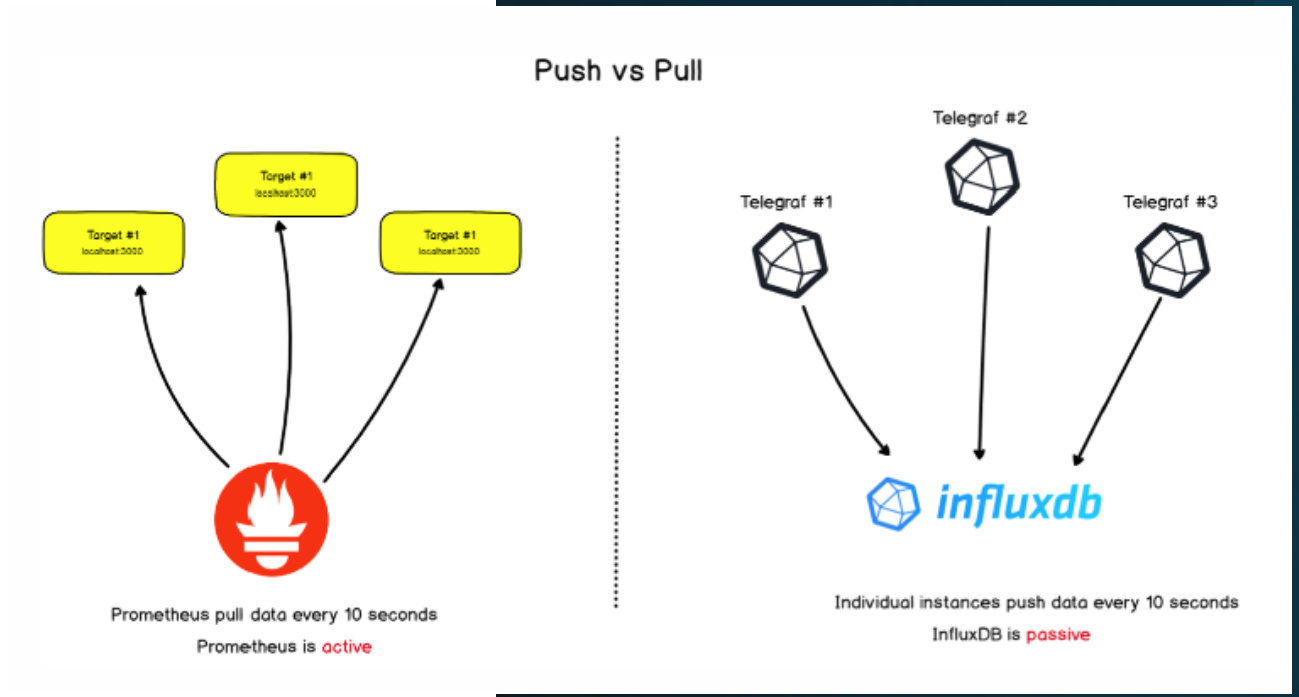
Инструменты

Существует различны системы мониторинга и алертинга.

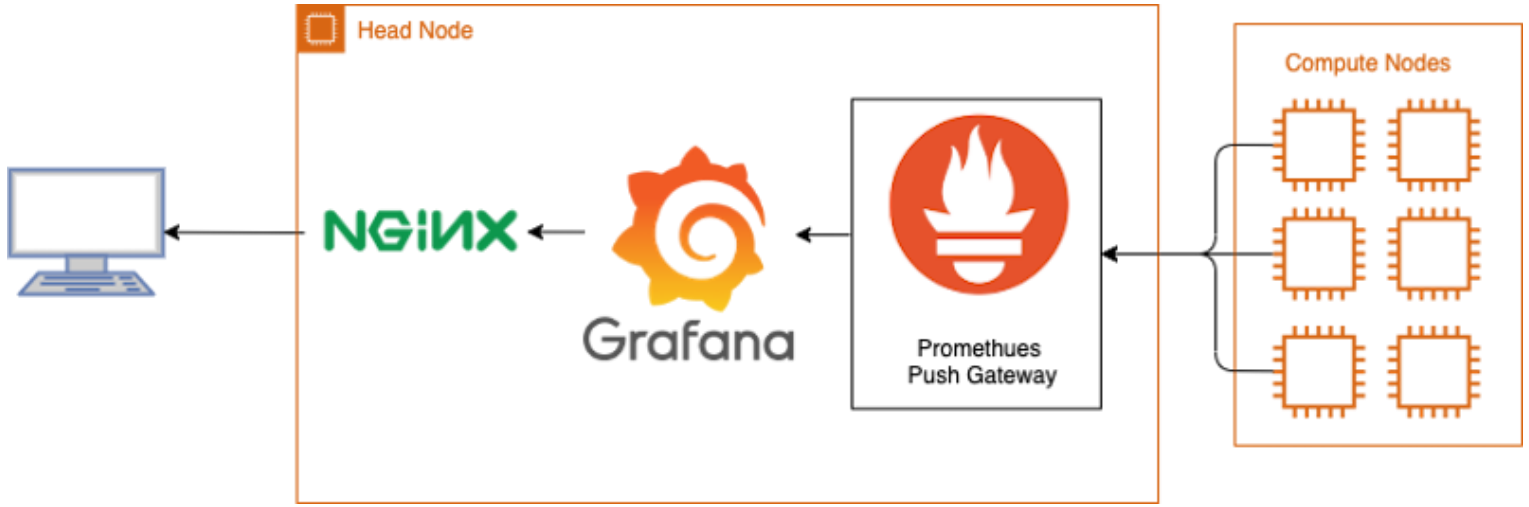
- PRTG, Zabbix, Prometheus, Graphite, Grafana, Sentry и т.д.
- Различные APM: newrelic, Elastic APM, DynaTrace, DataDog, Solar и т.д.

Pull vs Push модель сбора метрик

- **Push модель**: приложение само ходит в сервис метрик и пушит туда все метрики (тактика hearbeat)
- **Pull модель**: приложение выставляет url (обычно /metrics), в котором все метрики, а сервис метрик забирает, и потом отображает (тактика ping/echo).



смотрим пример



2 восстановление после сбоя

RPO (Recovery Point Objective)

RTO (Recovery Time Objective)

- **RPO** определяет максимально допустимую потерю данных в случае сбоя. Например, если **RPO** равен 1 часу, то система должна быть восстановлена до состояния, которое было не более чем за 1 час до сбоя.
- **RTO** определяет максимально допустимое время восстановления системы после сбоя. Например, если **RTO** равен 4 часам, то система должна быть полностью восстановлена в течение 4 часов после сбоя.
- Эти показатели помогают определить требования к резервному копированию данных и восстановлению системы после сбоя. Они также помогают оценить эффективность резервного копирования и восстановления системы.

Обратное восстановление

При обратном восстановлении (**backward recovery**) основной проблемой является приведение системы из ее нынешнего ошибочного состояния обратно в ранее правильное состояние.

Для этого необходимо время от времени регистрировать состояние системы и, когда что-то идет не так, восстанавливать такое записанное состояние.

Каждый раз (когда) записывается текущее состояние системы, говорят, что создается контрольная точка (**checkpoint**).

Прямое восстановление

В этом случае, когда система вошла в ошибочное состояние, вместо того чтобы возвращаться к предыдущему состоянию контрольной точки, делается попытка привести систему в правильное новое состояние, из которого она может продолжить работу.

Основная проблема с механизмами прямого восстановления после ошибок заключается в том, что необходимо **заранее знать, какие ошибки могут возникнуть**. Только в этом случае можно исправить эти ошибки и перейти в новое состояние.

Стратегия повторений восстановление

- **Временные сбои** - это ошибки, возникающие из-за некоторых временных условий, таких как:
  - проблемы с сетевым подключением;
  - временная недоступность службы;
  - тайм-аут запросов к службе;
  - сбои на уровне инфраструктурных элементов;
- Стратегии **повторных попыток** можно использовать для попытки повторить операцию, когда она обнаруживает временную ошибку.
- Политика повторных попыток может быть адаптирована в **зависимости от природы компонента**, вызвавшего ошибку, который может диктовать такие вещи, как количество попыток повторных попыток и период ожидания между попытками.

Стратегия повторений восстановление

1. **Регулярные интервалы**: система программного обеспечения ожидает одинаковое количество времени между каждой попыткой интервала.
2. **Инкрементные интервалы**: система программного обеспечения ожидает короткое время перед первой попыткой, а затем постепенно увеличивает количество времени перед каждой последующей повторной попыткой. Например, повторные попытки могут произойти через 2 секунды, 6 секунд, 12 секунд и т. Д.
3. **Экспоненциальный откат**: система программного обеспечения ждет короткое время перед первой попыткой, а затем экспоненциально увеличивает количество времени перед каждой последующей повторной попыткой.
4. **Немедленная повторная попытка**: повторная попытка может быть выполнена немедленно. Однако не должно быть многократных попыток немедленной попытки. Если одна попытка немедленной попытки не удалась, то любые последующие попытки должны использовать один из других типов интервалов.
5. **Рандомизация**. Любой из вышеупомянутых типов интервалов можно использовать в сочетании с рандомизацией, чтобы предотвратить одновременную отправку попыток повторной попытки из нескольких экземпляров клиента.

Rollback восстановление

- Метод отката **возвращает систему к контрольной точке**, в которой, как известно, система находилась в хорошем состоянии.
- Это требует, чтобы **контрольные точки сохранялись** каким-либо образом.
- Как только система откатывается, обычные операции могут продолжаться снова.
- Этот подход можно использовать **с активной или пассивной избыточностью**, чтобы после сбоя компонент после сбоя мог снова стать активным.

Тактики предотвращения сбоя

предотвращение сбоя

Graceful degradation предотвращение

Graceful degradation предотвращение

- Изящная деградация (Graceful degradation) - это подход к устранению неисправностей, при **котором некоторые функции становятся недоступными**, чтобы вся система не стала непригодной для использования.
- Если сбои не позволяют всей программной системе функционировать**, некоторые функции могут быть отброшены в пользу других**. Например, если в системе недостаточно ресурсов, наиболее важные функции могут быть сохранены, а другие отключены.

Игнорирование ошибки предотвращение

- Другой подход к обработке ошибок - **просто игнорировать ошибку**.
- Если известно, что конкретный тип ошибки из определенного источника может быть проигнорирован, то система может просто игнорировать ошибку, и обработка может продолжаться.
- **Пример:**
  В приложении Yandex.Такси данные о положении водителя могут теряться, но это не страшно, поскольку через 2 секунды будут переданы новые данные.

Прекращение обслуживания

Принудительный вывод из строя компонента до того как он может сбоить.

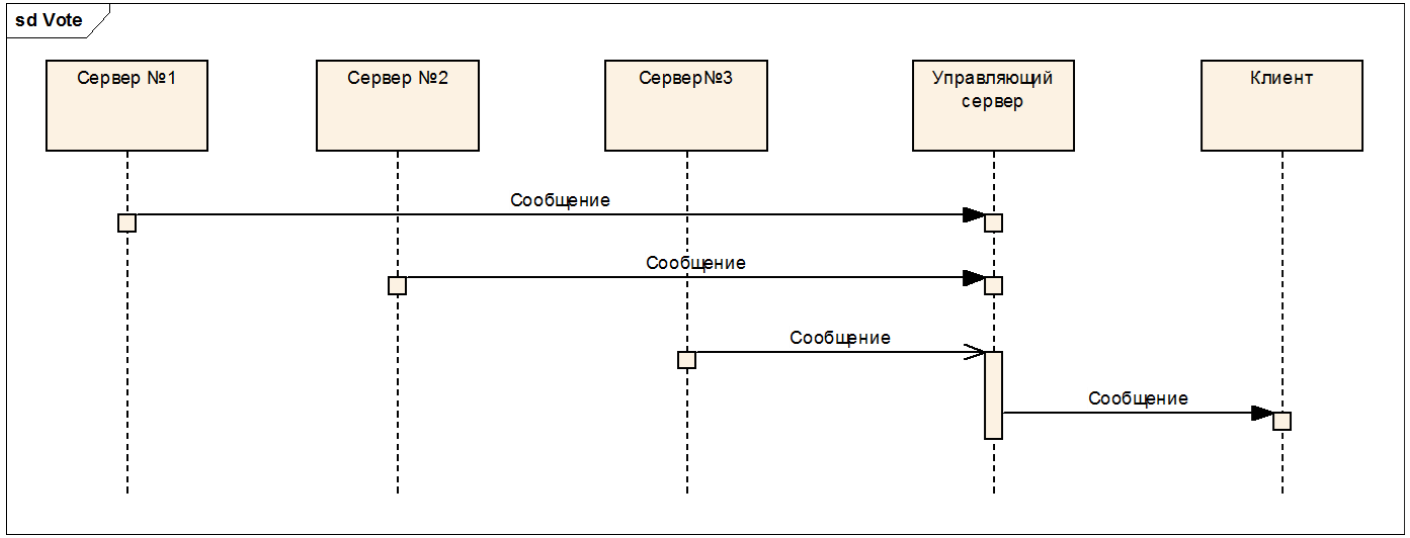
Например, перезагрузка для того что бы не было утечки памяти.

Голосование предотвращение и обнаружение

- Система голосования может использоваться для сообщения о неисправностях. Одним из таких подходов является **тройное модульное резервирование** (TMR), иногда называемое трехрежимным резервированием. Он использует три компонента для выполнения одного и того же процесса.

- Логика голосования будет сравнивать результаты для получения одного выхода. **Если все три компонента выдают одинаковый результат, значит, все работает так, как ожидалось**. Если два из трех компонентов согласуются, они могут исправить ошибку, отказавшись от третьего компонента. Ошибка сообщается о третьем компоненте.

Голосование предотвращение и обнаружение

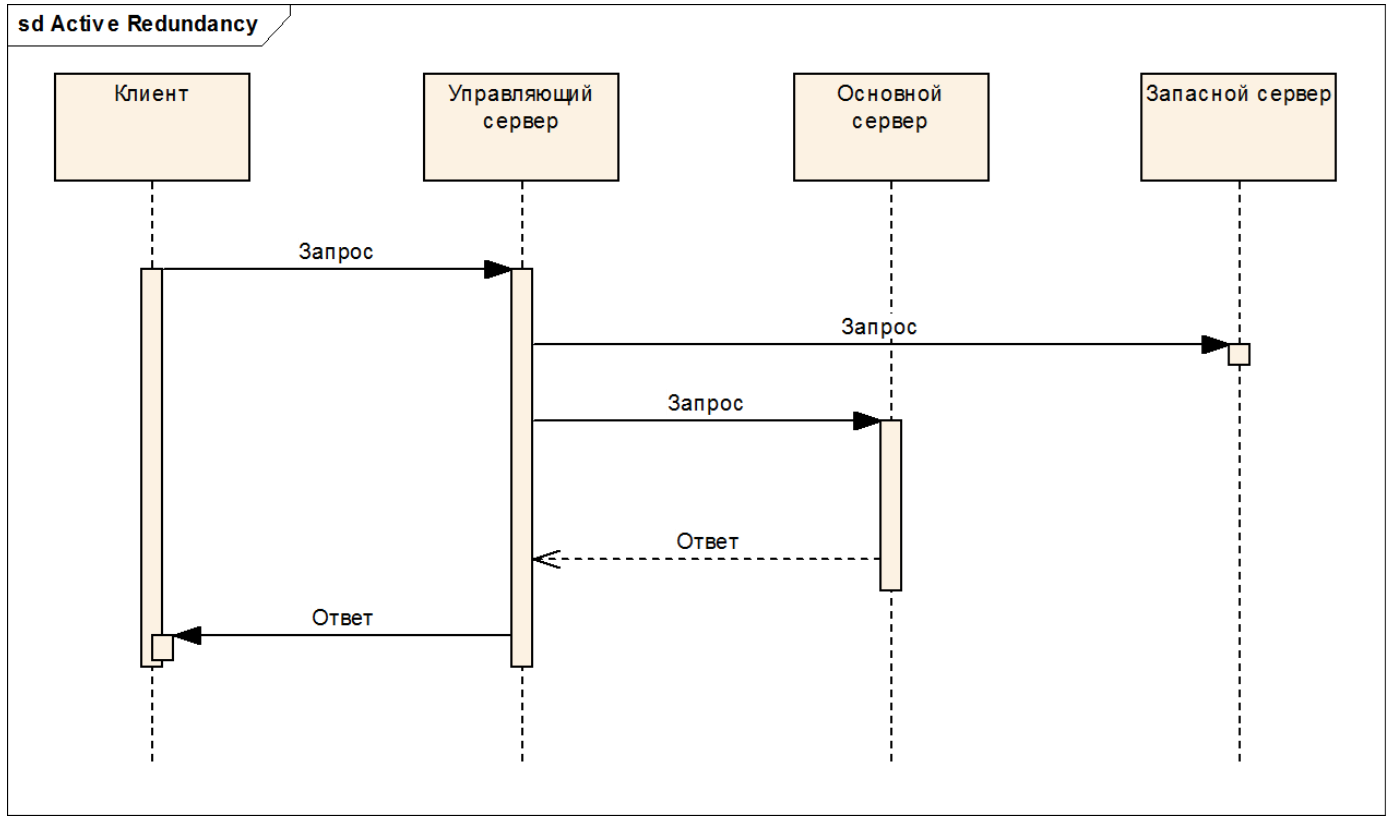


Резервирование

Одним из способов восстановления после сбоев и обеспечения доступности является наличие механизма **аварийного переключения**.

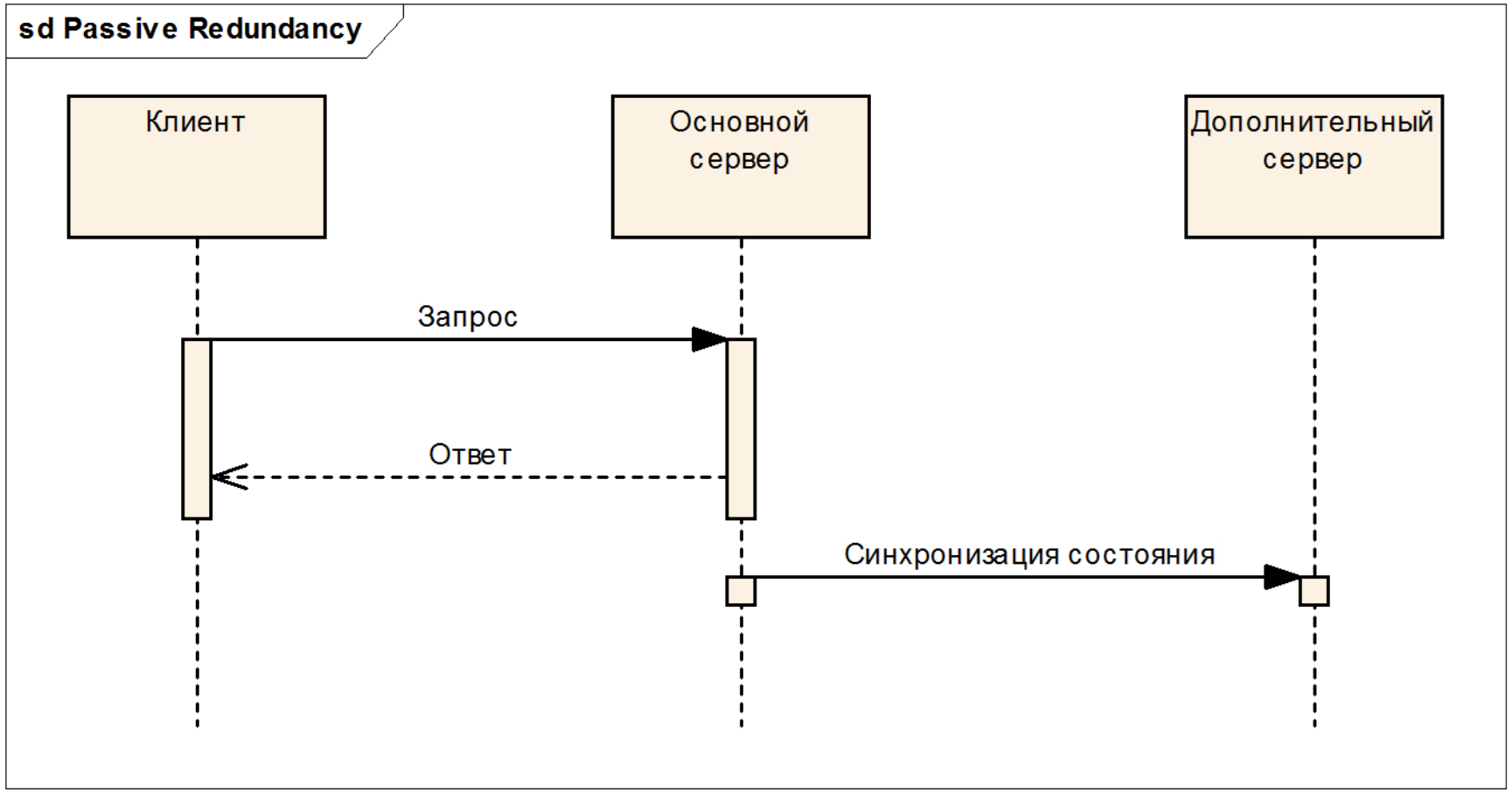
1. В средах с «**активным резервированием»** для каждого компонента существует другой компонент, который выполняет те же процессы с теми же входами, поэтому в случае отказа одного из них в любой момент его может заменить другой компонент. Отказоустойчивость обычно прозрачна, потому что время восстановления почти мгновенно.
2. **«Пассивное резервирование»** - это среда, в которой только активные компоненты выполняют процессы и предоставляют резервным компонентам периодические обновления состояния. Подход с пассивной избыточностью **не так высокодоступен, как активный**, но он дешевле в эксплуатации. В зависимости от того, как часто в резервную копию предоставляются обновления состояния, время восстановления может быть в секундах или минутах.
3. При использовании подхода «**холодного резерва**» дублирующие компоненты компоненты не работают, пока они не потребуются. Неисправный компонент не будет работать до тех пор, пока он не будет отремонтирован или заменен. Использование холодного резерва требует больше времени для запуска избыточного компонента по сравнению с горячим или теплым резервом. Время восстановления может составить несколько часов.

Активное резервирование предотвращение



Пример: синхронная репликация данных

Пассивное резервирование
предотвращение



Репликация

Репликация применяется

Балансировка нагрузки

- Много чтения, мало записи
- Тяжёлые запросы (аналитика, отчёты)
- Бэкапы
- Увеличение надёжности

Отказоустойчивость

- Повышение доступности

Виды репликации по составу участников

Single leader

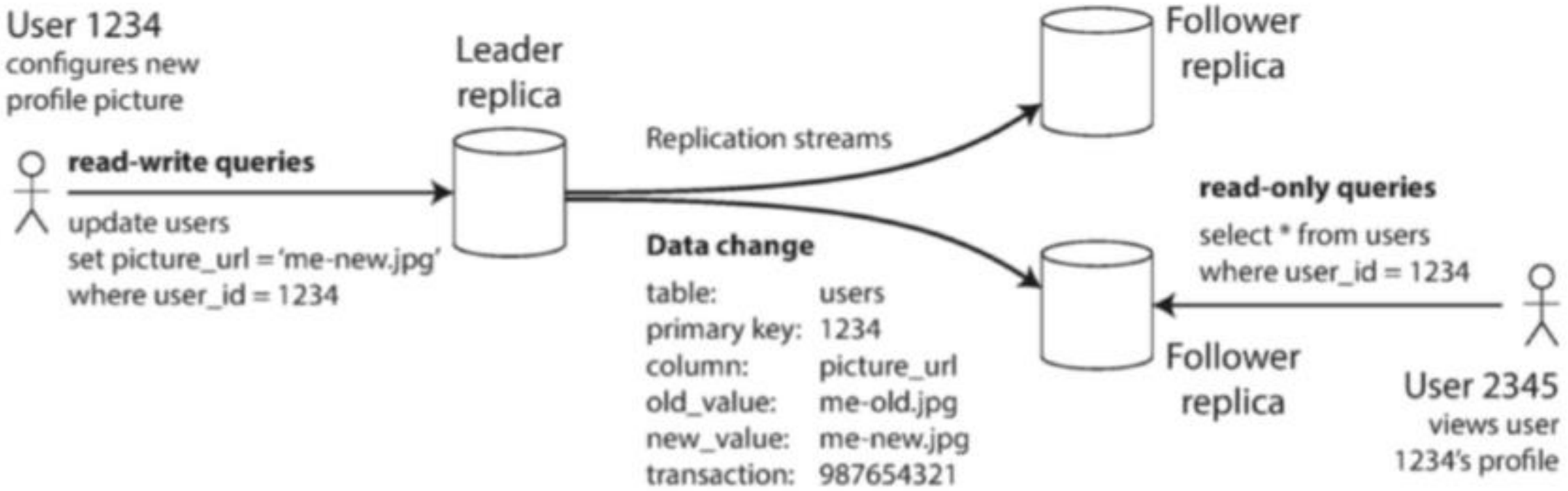
- Один лидер, много реплик

Multi leader

- Несколько кластеров single leader + conflict resolver
- Вырожденный случай, когда есть только лидеры и conflict resolver

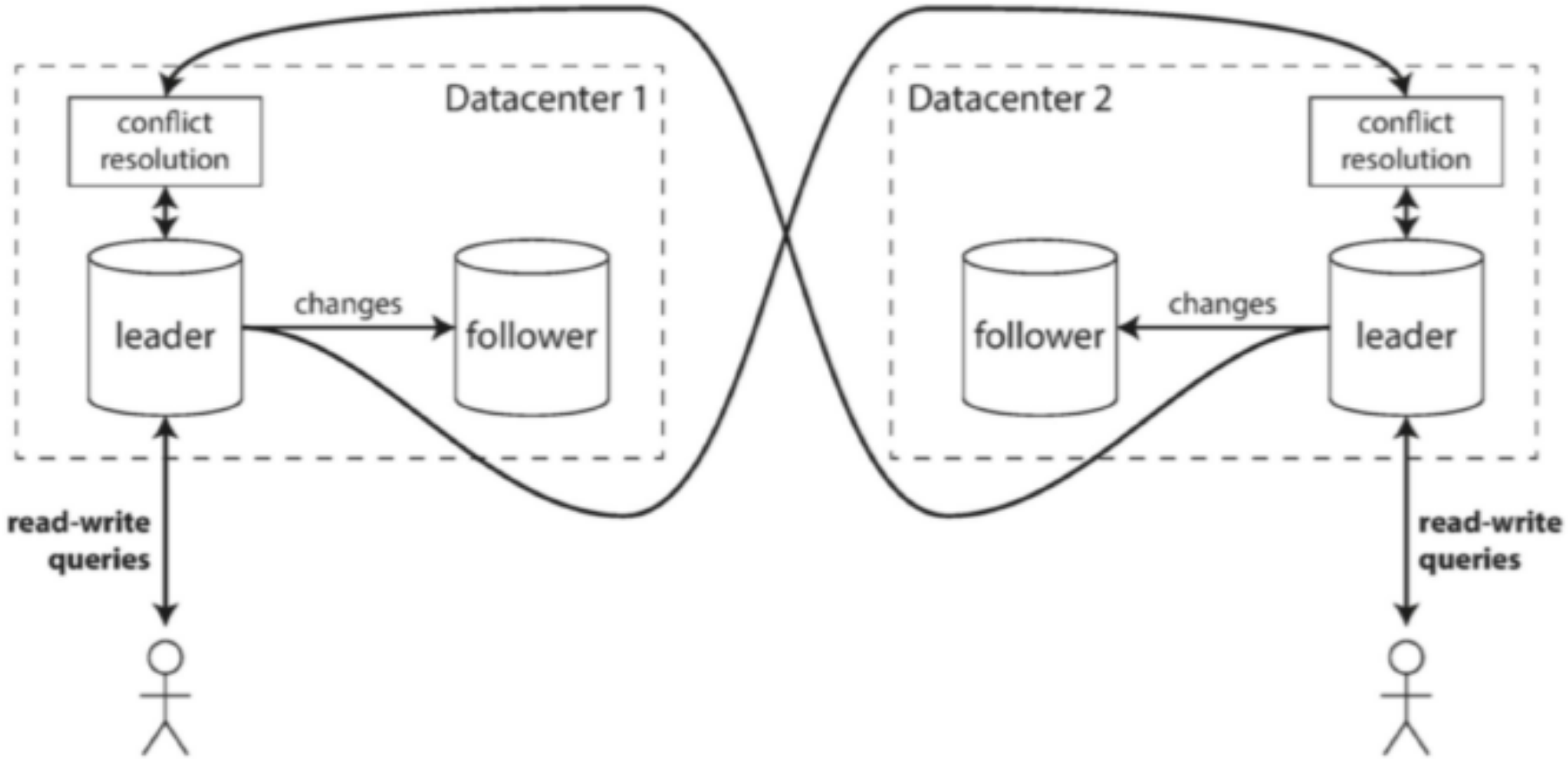
Leaderless

- Алгоритм консенсуса
- Устойчива к отключению узлов
- Блокчейн



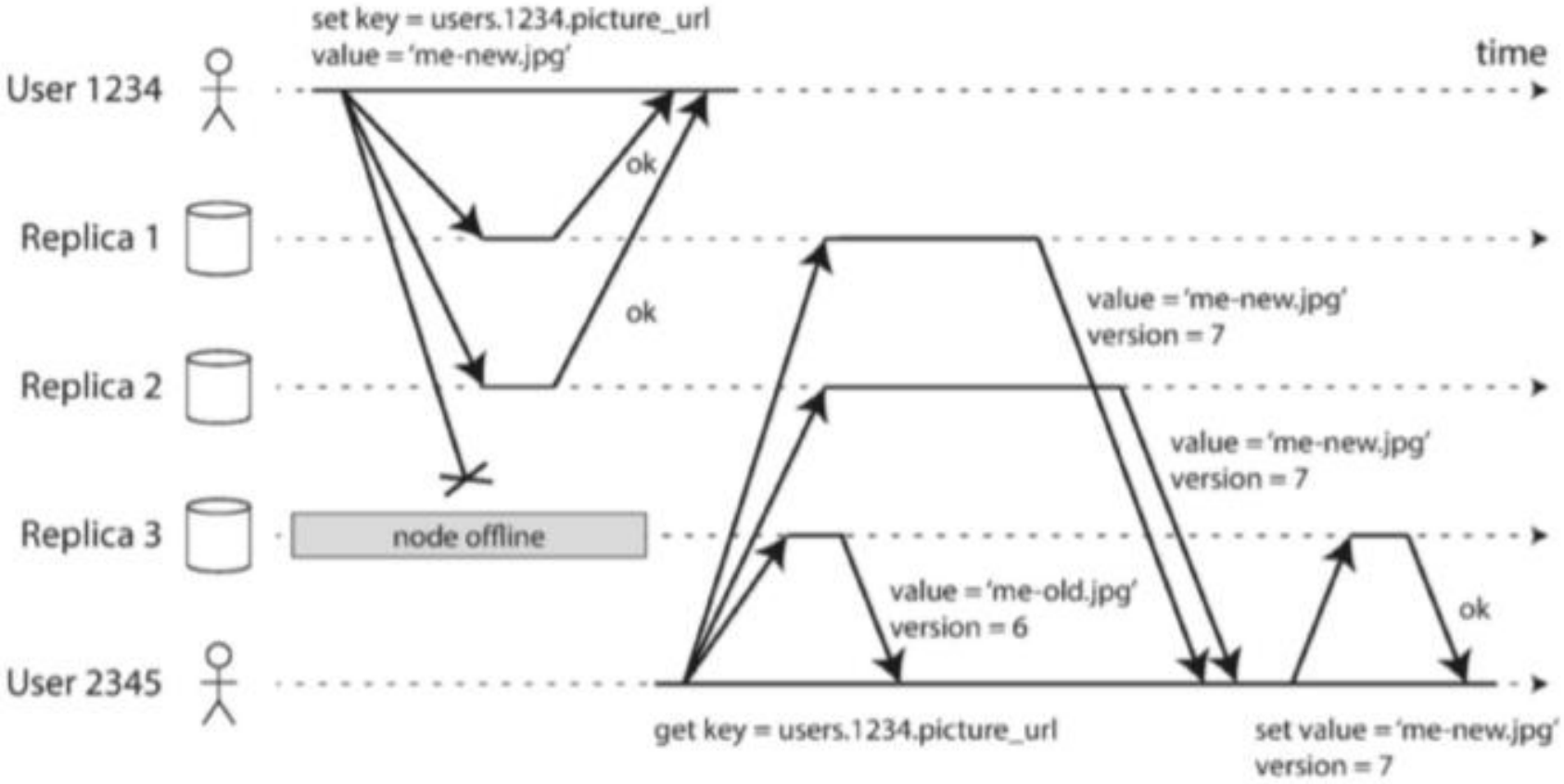
Single leader

https://www.amazon.com/Designing-Data-Intensive-Applications-Reliable-Maintainable/dp/1449373321



Multi leader

https://www.amazon.com/Designing-Data-Intensive-Applications-Reliable-Maintainable/dp/1449373321



Leaderless

https://www.amazon.com/Designing-Data-Intensive-Applications-Reliable-Maintainable/dp/1449373321

Виды репликации по гарантии доставки

Синхронная

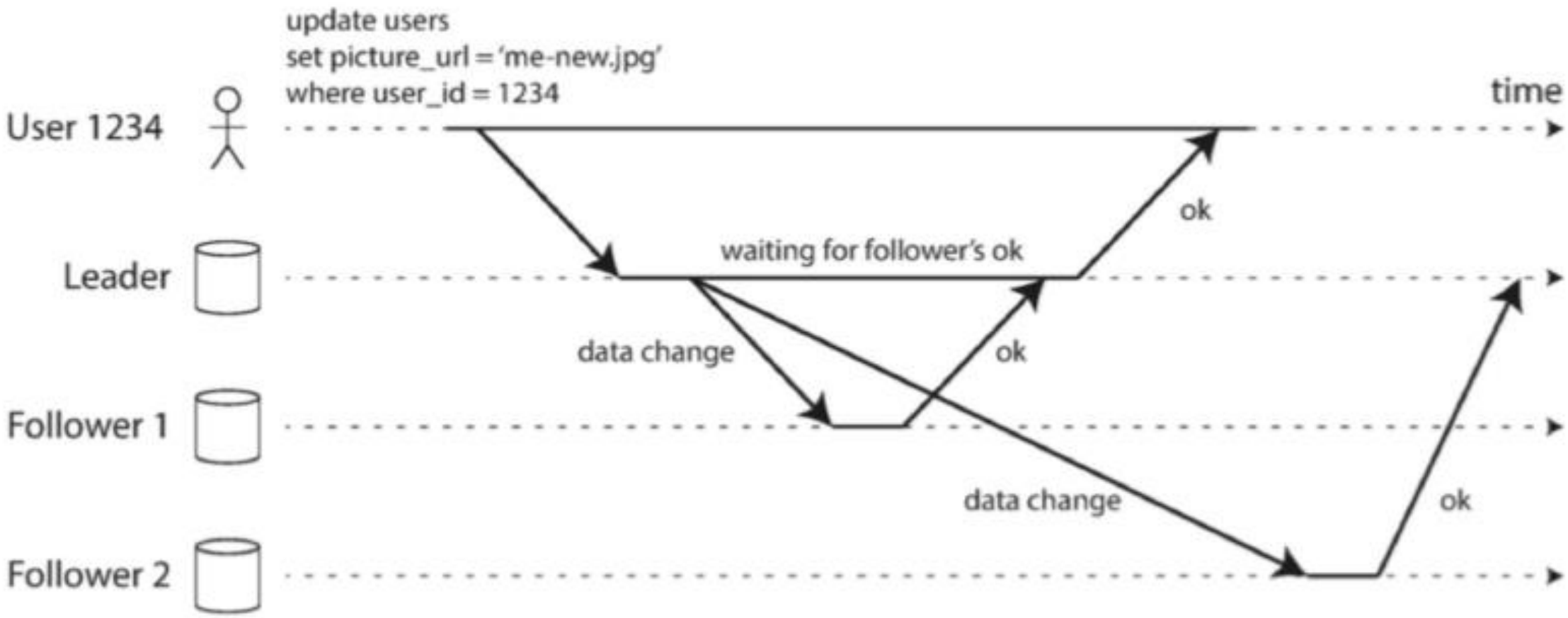
- Ждём подтверждения от реплики, только после этого возвращаем ответ
- Гарантирует **сильную согласованность** данных
- **Медленный** ответ

Асинхронная

- Отвечаем сразу, подтверждение от реплики приходит позднее
- **Не гарантирует сильную согласованность данных**
- Гарантирует согласованность в конечном счёте
- **Быстрый ответ**

Семи-синхронная

- Дожидаемся подтверждения от реплики о получении запроса на запись
- Отвечаем пользователю после получения подтверждения от всех реплик
- Не гарантирует сильную согласованность данных
- Гарантирует согласованность в конечном счёте



Синхронная и асинхронная

https://www.amazon.com/Designing-Data-Intensive-Applications-Reliable-Maintainable/dp/1449373321

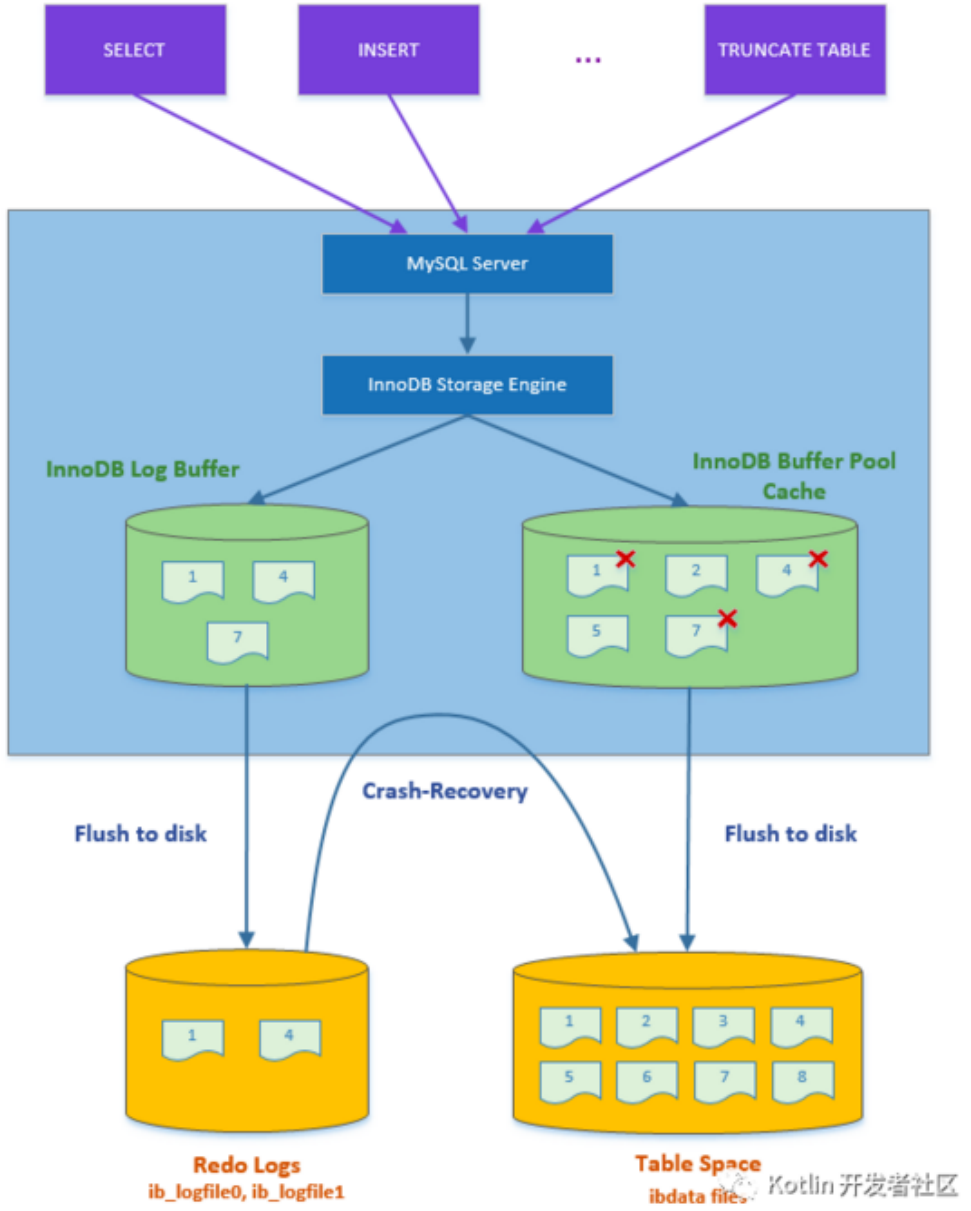
Виды репликации по способу передачи данных

Бинарная
передача файлов журналов
нет атомарности

Логическая
передача SQL-команд
атомарность по SQL-командам

Триггеры
для частичной репликации
атомарность вплоть до значения одного поля в записи

Undo/ Redo log

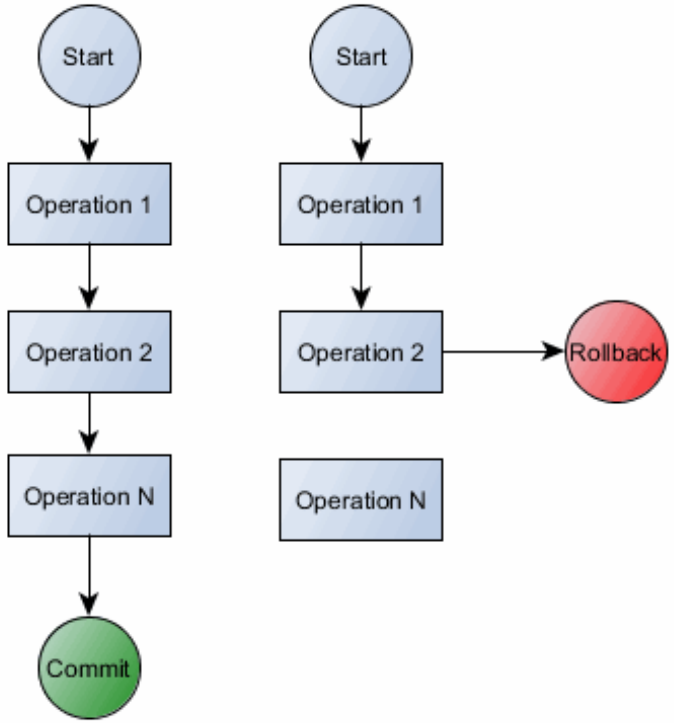


Консистентность данных в монолитных системах

Транзакции

Транзакции предотвращение

- Транзакции могут быть использованы для предотвращения ошибок. Несколько шагов в процессе могут быть объединены в транзакции, так что если один шаг завершится неудачей, весь пакет может быть отменен.
- Этот подход может предотвратить сохранение данных в неправильном состоянии или предотвратить состояние гонки, когда более чем один процесс пытается получить доступ или изменить одни и те же данные.

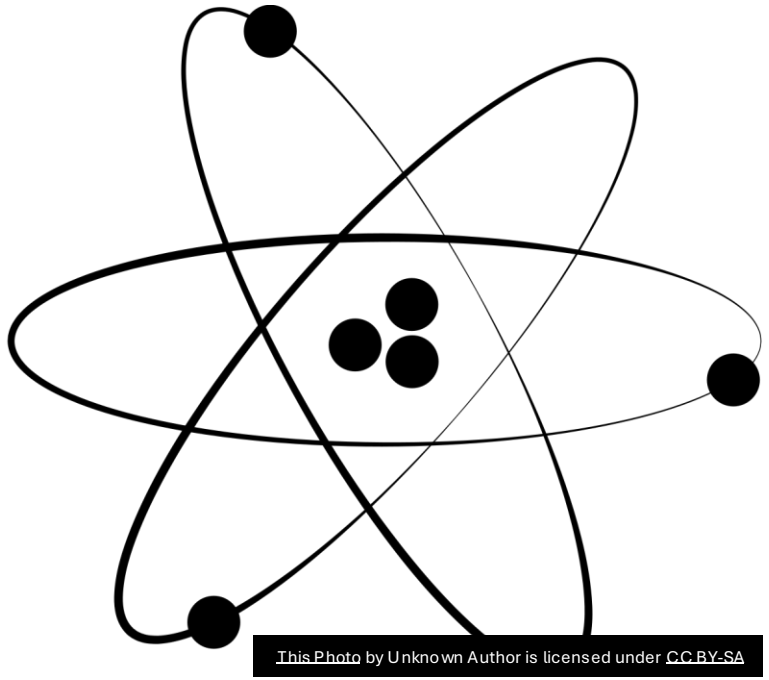


ACID
работа с данными

- Atomicity
- Consistency
- Isolation
- Durability

atomicity

- Атомарность означает "все или ничего".
- Гарантирует, что либо все операции транзакции будут успешно применены, либо не будет применена ни одна из них.
- Благодаря этому появляется возможность повторять прерванные транзакции, не опасаясь что часть операций уже была выполнена



consistency

Поддержка целостности
и согласованности
данных

isolation

- Изоляция -свойство, которое позволяет выполнять параллельные транзакции как последовательные.
- Существуют несколько уровней изоляции, которые дают различные гарантии.

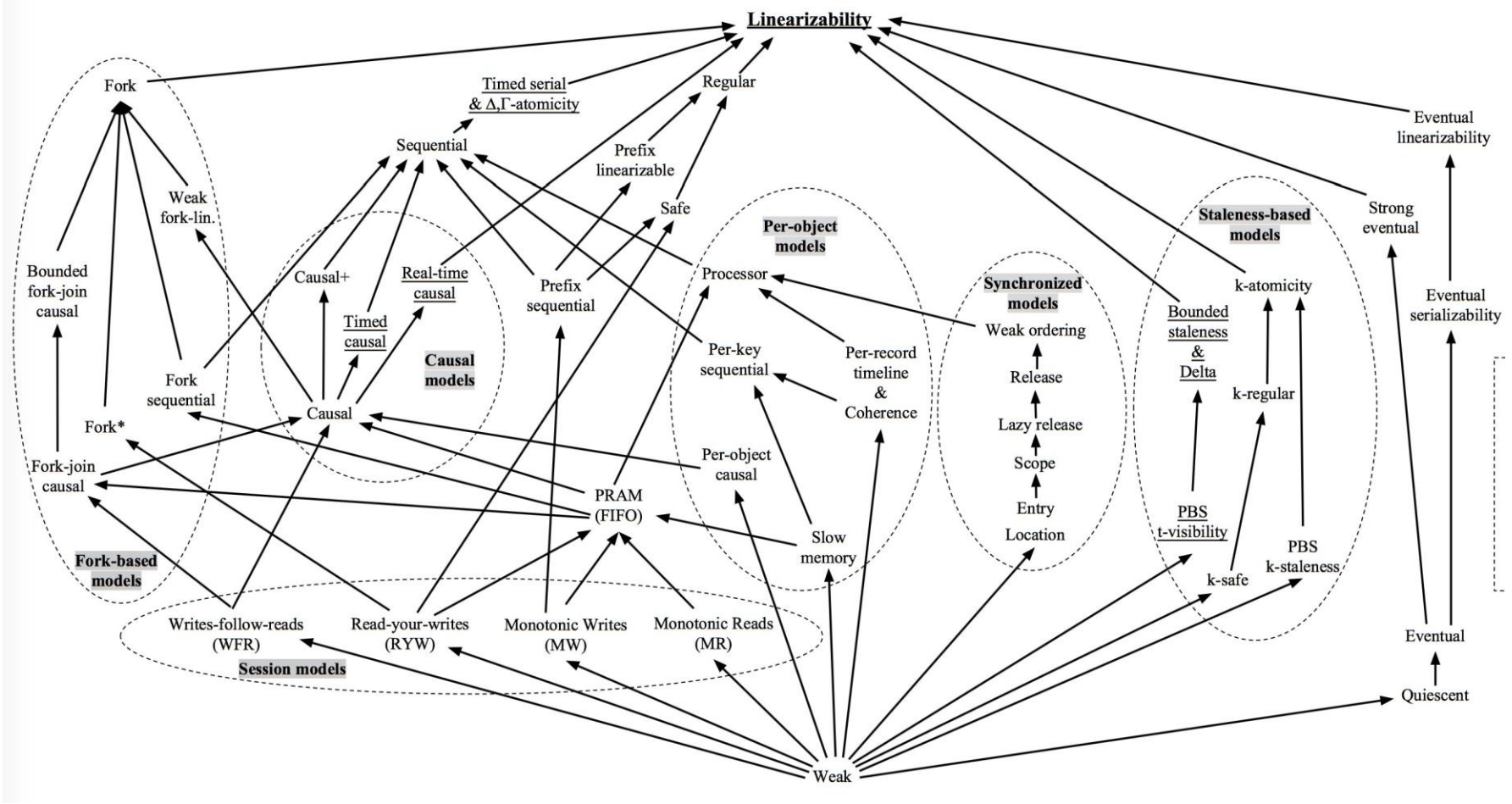
durability

- Данные зафиксированы в энергонезависимой памяти (fsync).
- Полностью это свойство обеспечить невозможно. Диски сбоят.

Модель согласованности (consistency model)

– это, по сути, контракт между процессами и хранилищем данных. В ней говорится, что если процессы соглашаются подчиняться определенным правилам, хранилище обещает работать правильно.

Например, процесс, который выполняет операцию чтения над элементом данных, ожидает, что операция вернет значение, которое показывает результаты последней операции записи в этих данных.



Модели согласованности

https://pbs.twimg.com/media/CgUIyYSXIAAerb2?format=jpg&name=4096x4096

Сильная согласованность

- **Strong consistency**(сильная согласованность) – после завершения обновления данных все последующие чтения вернут новые данные
- **Monotonic read –** после того как процесс считал значение переменной, любая последующая операция вернет либо ее либо эти же данные либо более свежие
- **Write consistency** –операция записи данных для процесса должна закончится раньше чем начнется следующая запись

Слабые согласованности

- **Weak consistency** (слабая согласованность) –есть окно несогласованности
- **Eventual consistency**(согласованность в конечном счёте) –размер окна несогласованности не фиксирован, зависит от внешних условий
- **Causal consistency** (причинная согласованность) – только процессы, уведомлённые об обновлениях гарантированно прочитают новые данные
- **Read-your-writes consistency** –только сам процесс гарантированно читает обновлённые им самим данные
- **Session consistency** –процесс получает доступ к актуальным данным в рамках сессии

Изоляция транзакций

Уровни изолированности транзакций - это стандартные уровни, определяющие, насколько одна транзакция может влиять на другую транзакцию, выполняющуюся параллельно.

Выбор уровня изолированности зависит от требований к целостности данных и скорости выполнения операций. Чем выше уровень изолированности, тем медленнее будет выполнение операций, но тем выше будет гарантия целостности данных.

Уровни изоляции транзакций

Существует четыре уровня изолированности транзакций:

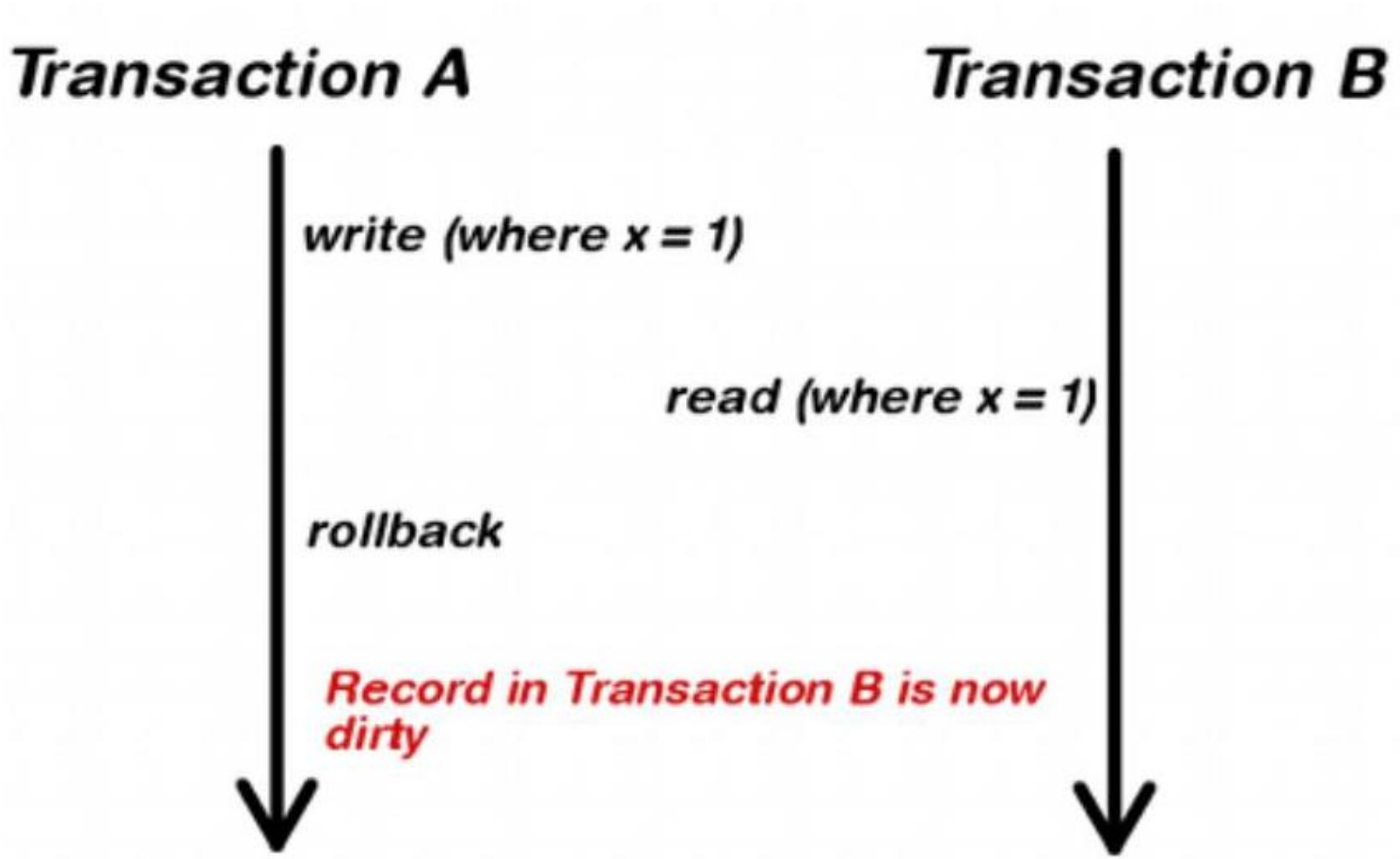
1. Уровень чтения неподтвержденных данных (Read uncommitted) - транзакция может прочитать данные, которые еще не были подтверждены другой транзакцией.

2. Уровень чтения подтвержденных данных (Read committed) - транзакция может прочитать только данные, которые уже были подтверждены другой транзакцией.

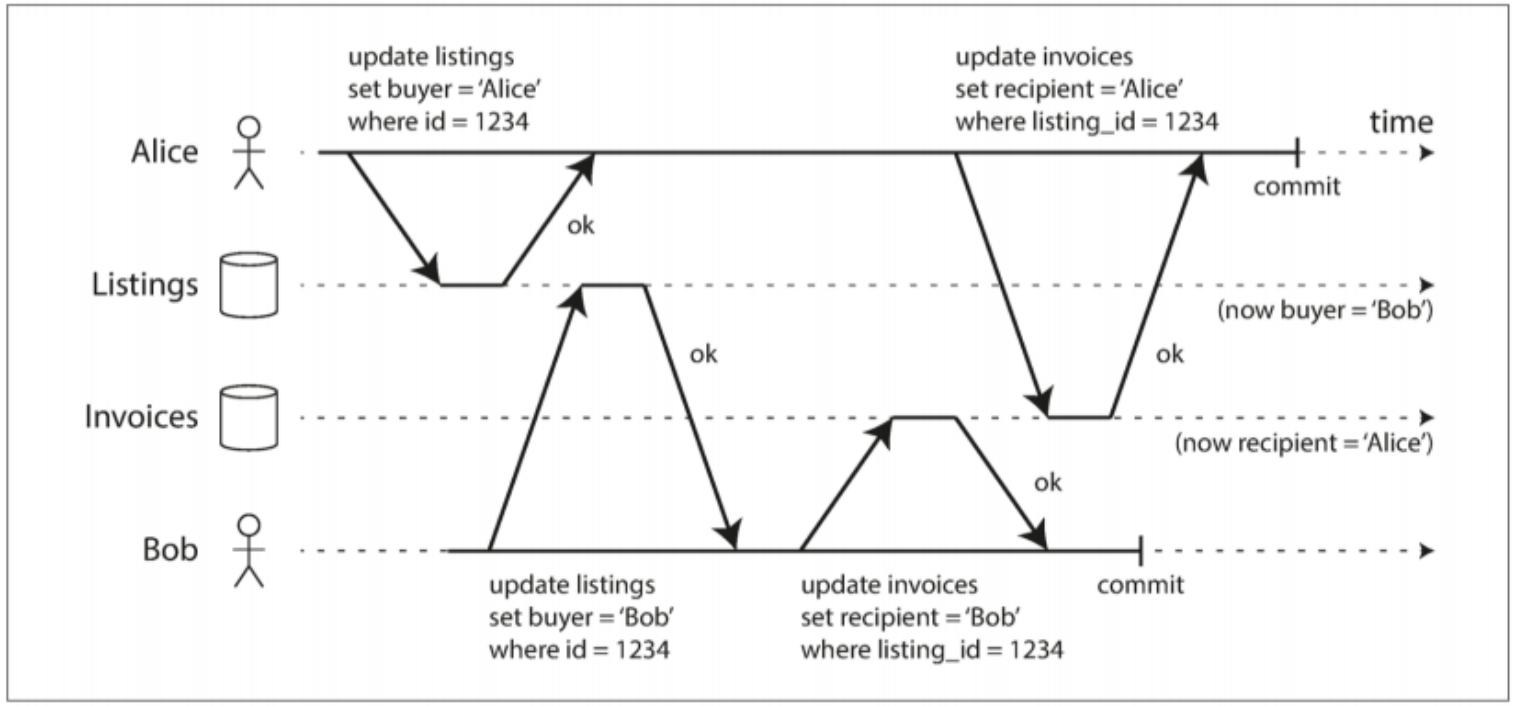
3. Уровень повторяемого чтения (Repeatable read) - транзакция может прочитать только данные, которые уже были подтверждены другой транзакцией, и эти данные не могут быть изменены другой транзакцией.

4. Уровень сериализации (Serializable) - транзакции выполняются последовательно, чтобы избежать конфликтов и гарантировать, что результаты будут согласованными.

проблема №1 dirty read



проблема №2 dirty writes



Изоляция

read commited

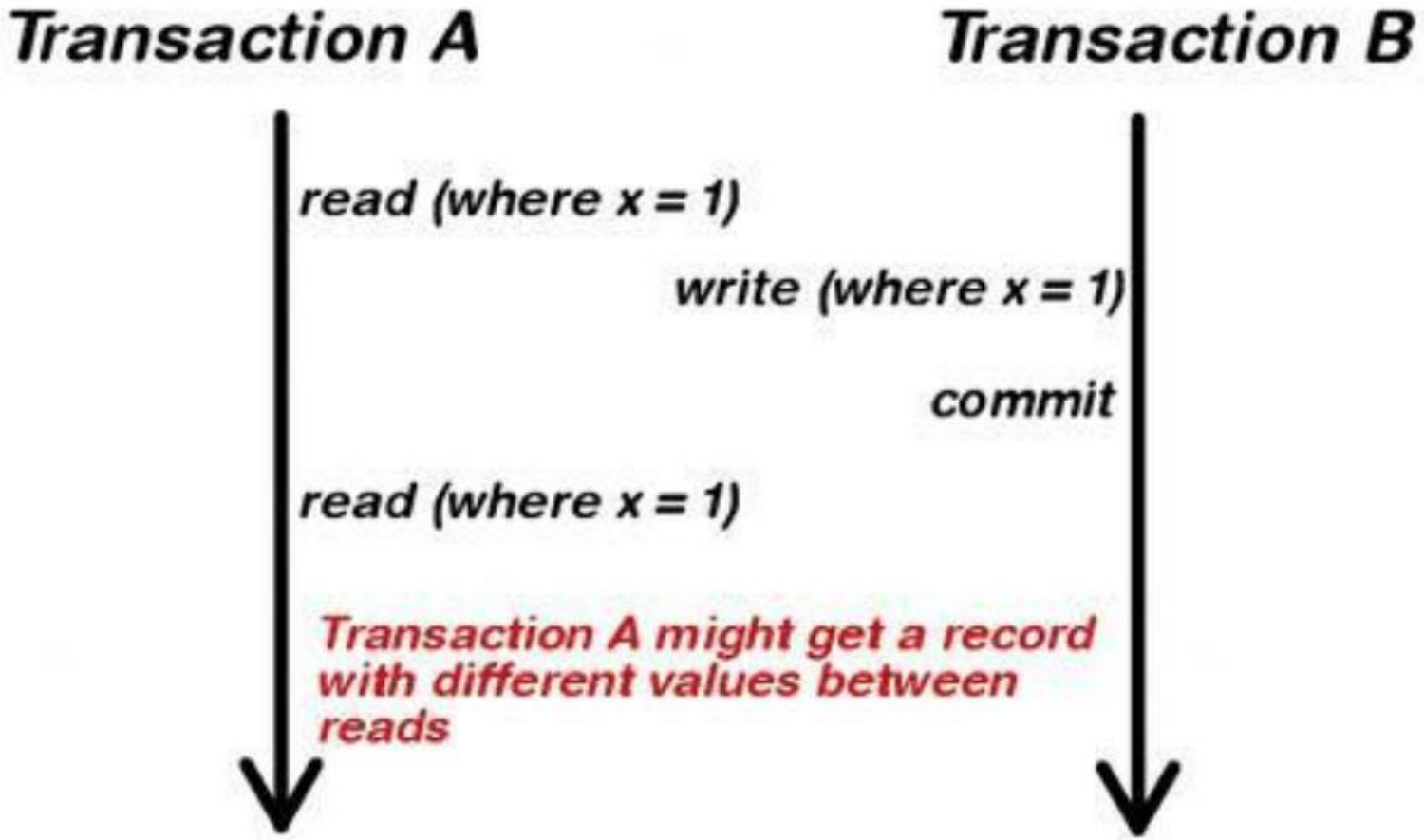
- Нет dirty reads
- Нет dirty writes

Реализация:

- Блокировка строк при записи
- Сохраняем после записи старые значения строк до коммита изменений

Проблема №3

unrepetable reads



Изоляция

repetable reads

- Нет unrepeatable reads

Реализация:

- Блокировки (чтение не блокирует запись, а запись не блокирует чтение)

На сегодня все

ddzuba@yandex.ru