Лекция 6

CAP-теорема. Принципы построения распределенных интернет-приложений.

Радченко Глеб Игоревич, каф. СП, ЮУрГУ http://glebradchenko.ru/courses/master/dot

CAP – теорема

Управление Данными в PBC

В любой сетевой системе, *обеспечивающей хранение совместно доступных данных*, разработчики хотят поддерживать следующие свойства:

- **Согласованность данных (Consistency)** – в системе существует единственная версия данных, соответствующая последней по времени операции обновления.
- **Доступность данных (Availability)** – запрос к PBC в любой момент времени должен завершиться корректным откликом, не зависимо от того, к какому серверу производится подключение.
- **Устойчивость к разделению (Partition tolerance)** - расщепление распределённой системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций.

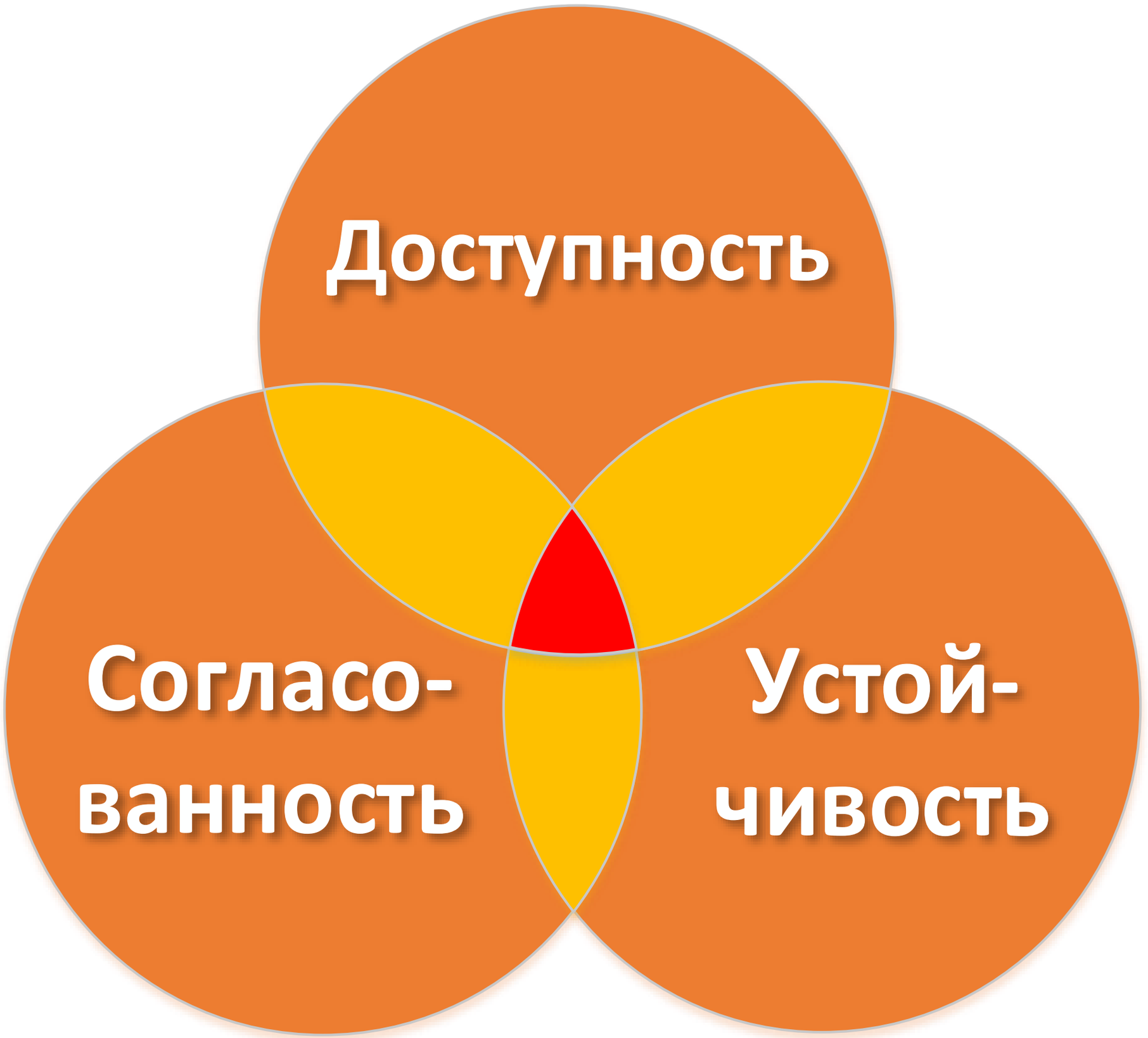
CAP-теорема

В 2000-м году Эрик Брюер (Eric Brewer – проф. Калифорнийского университета) предложил следующую теорему:

В любой сетевой системе, *обеспечивающей хранение совместно доступных данных*, одновременно могут поддерживаться только **два** из следующих трех свойств:

- **Согласованность данных (Consistency)**
- **Доступность данных (Availability)**
- **Устойчивость к разделению (Partition tolerance)**

В 2002-м году теорема была математически доказана в условиях отсутствия синхронизации (общих часов).


Доступность
Согласо-
ванность
Устой-
чивость

Что же такое Partition?

Разделение PBC это не только длительный разрыв между серверами, при котором один из них не может связаться с другим

» *Латентность сети также является разделением PBC.*

> Предположим, что у нас есть 2 сервера баз данных: один в России, другой в США.
> Они настроены на полное реплицированные
> Данные обновились на сервере в России. Через какое время сервер в США узнает об этом?
> **200 миллисекунд – лучший возможный результат** (худший возможный результат – никогда не узнает)
> В это «окно» они разделены – таким образом разделение системы происходит постоянно, даже в нормальных условиях работы PBC

Выбираем ли мы устойчивость?

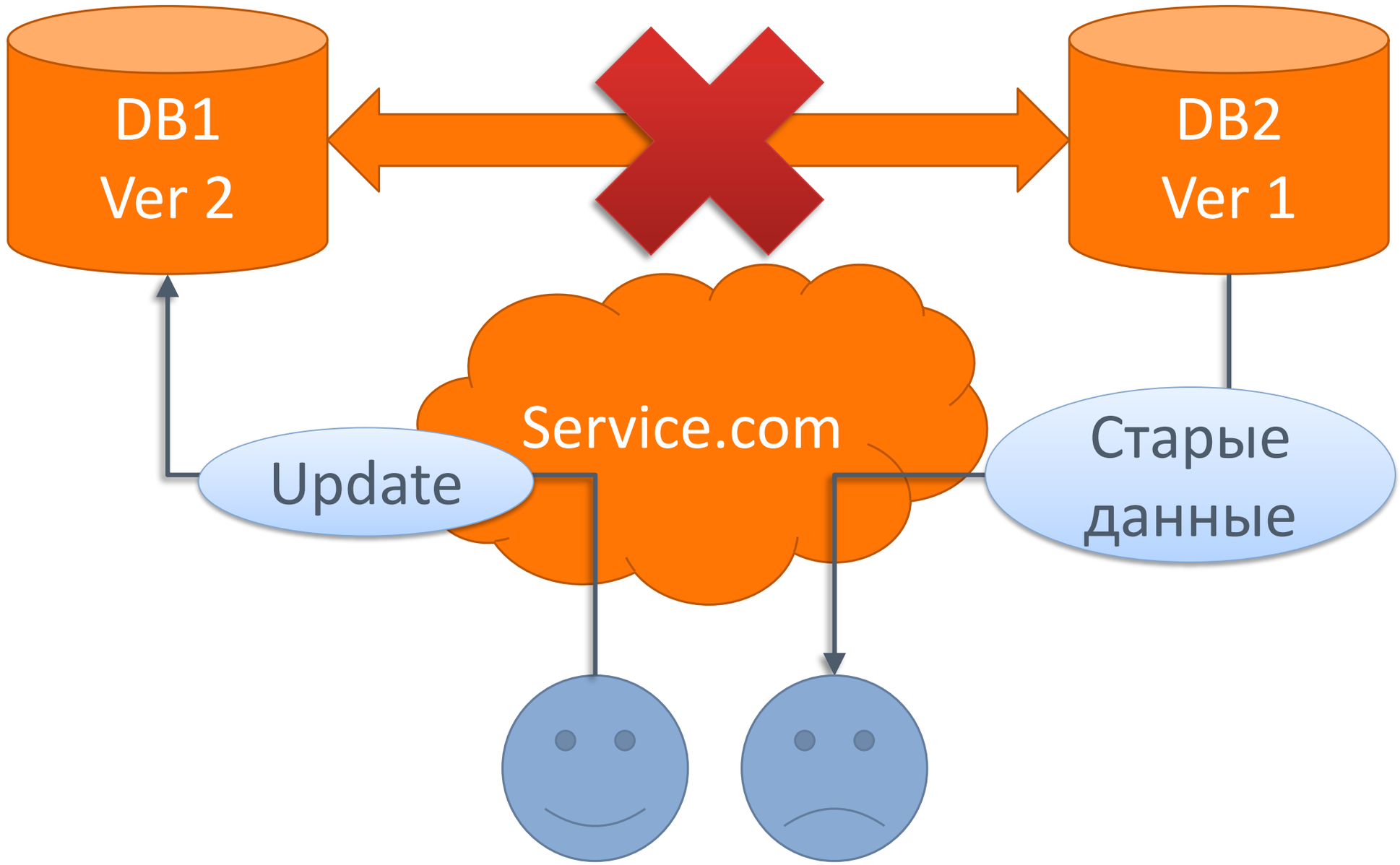
В реальном мире мы не можем выбирать, будут у нас сбои или нет. В PBC всегда будут возникать неполадки, зависит это от нас или нет:

- сетевые сбои,
- сбои работы оборудования,
- ошибки администрирования.

Плюс, любая латентность также вызывает разделение.

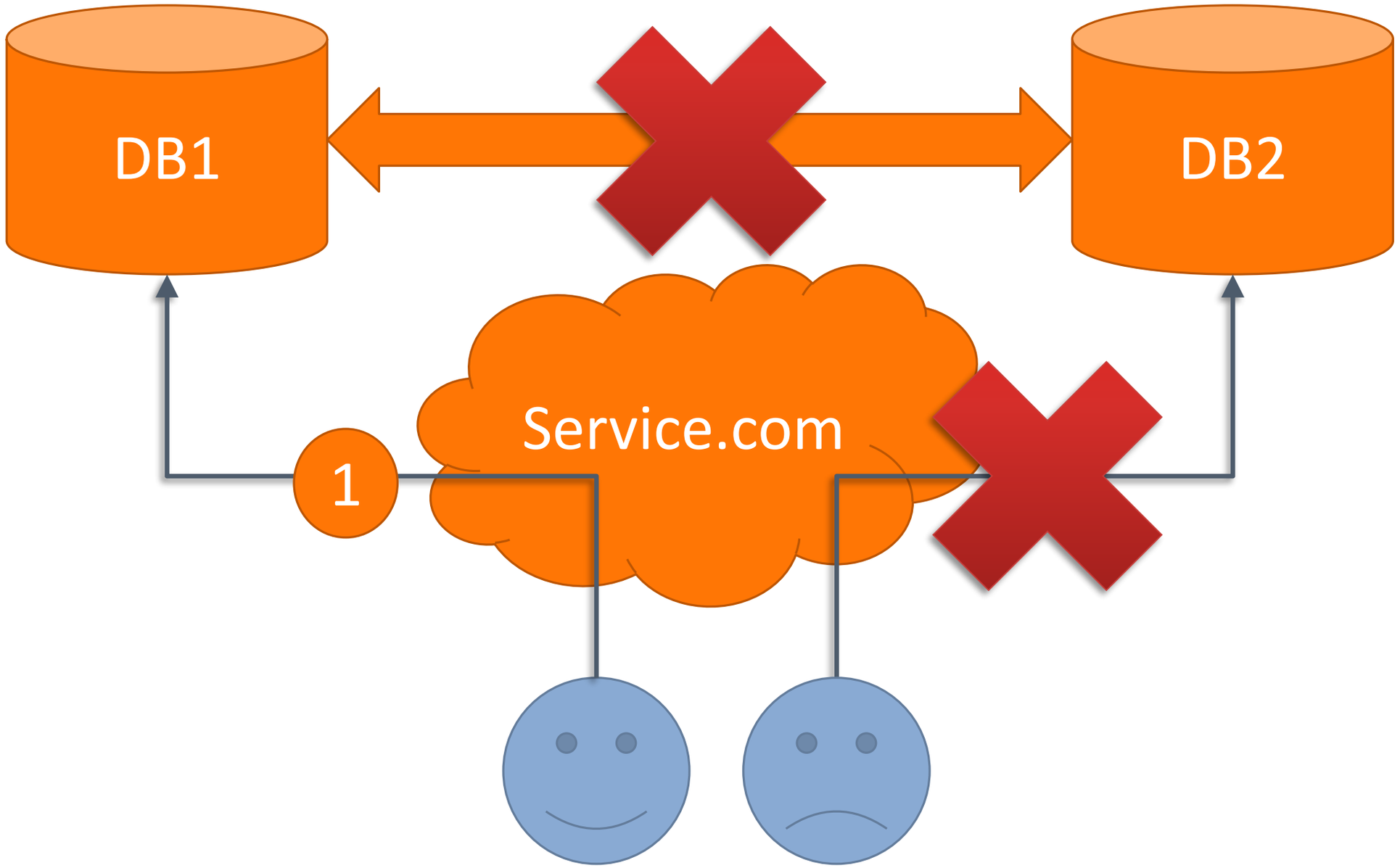
Поэтому приходится выбирать из двух вариантов:

AP: 100% доступность, но несогласованность данных:



Распределённая система, отказывающаяся от целостности результата. Большинство NoSQL-систем принципиально не гарантируют целостности данных («целостные в конечном итоге» - eventually consistent).

CP: 100% согласованность, но недоступность данных при распаде:

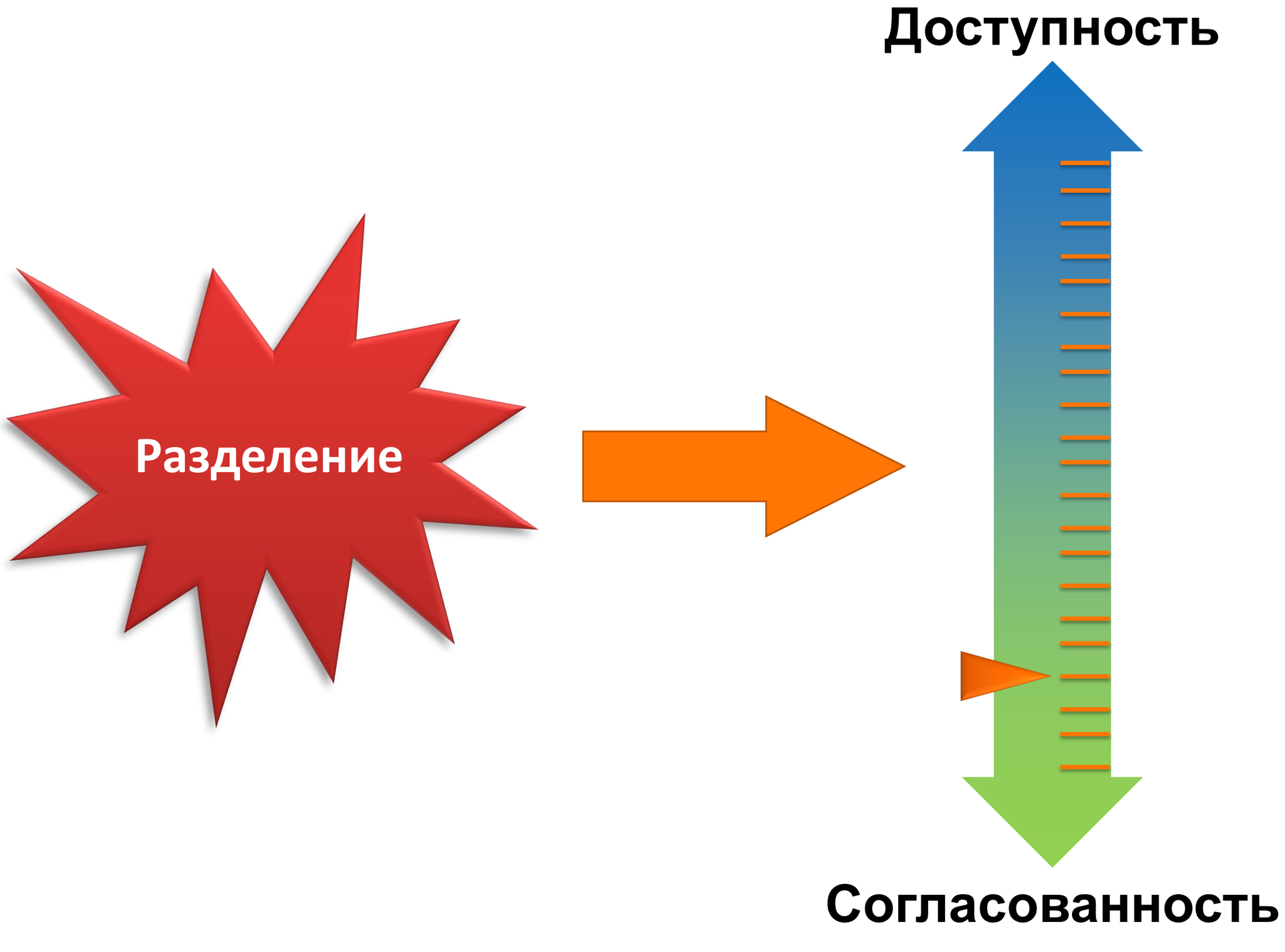


Распределённая система, в каждый момент обеспечивающая целостный результат и способная функционировать в условиях распада, в ущерб доступности может не выдавать отклик. Пессимистические блокировки для сохранения целостности.

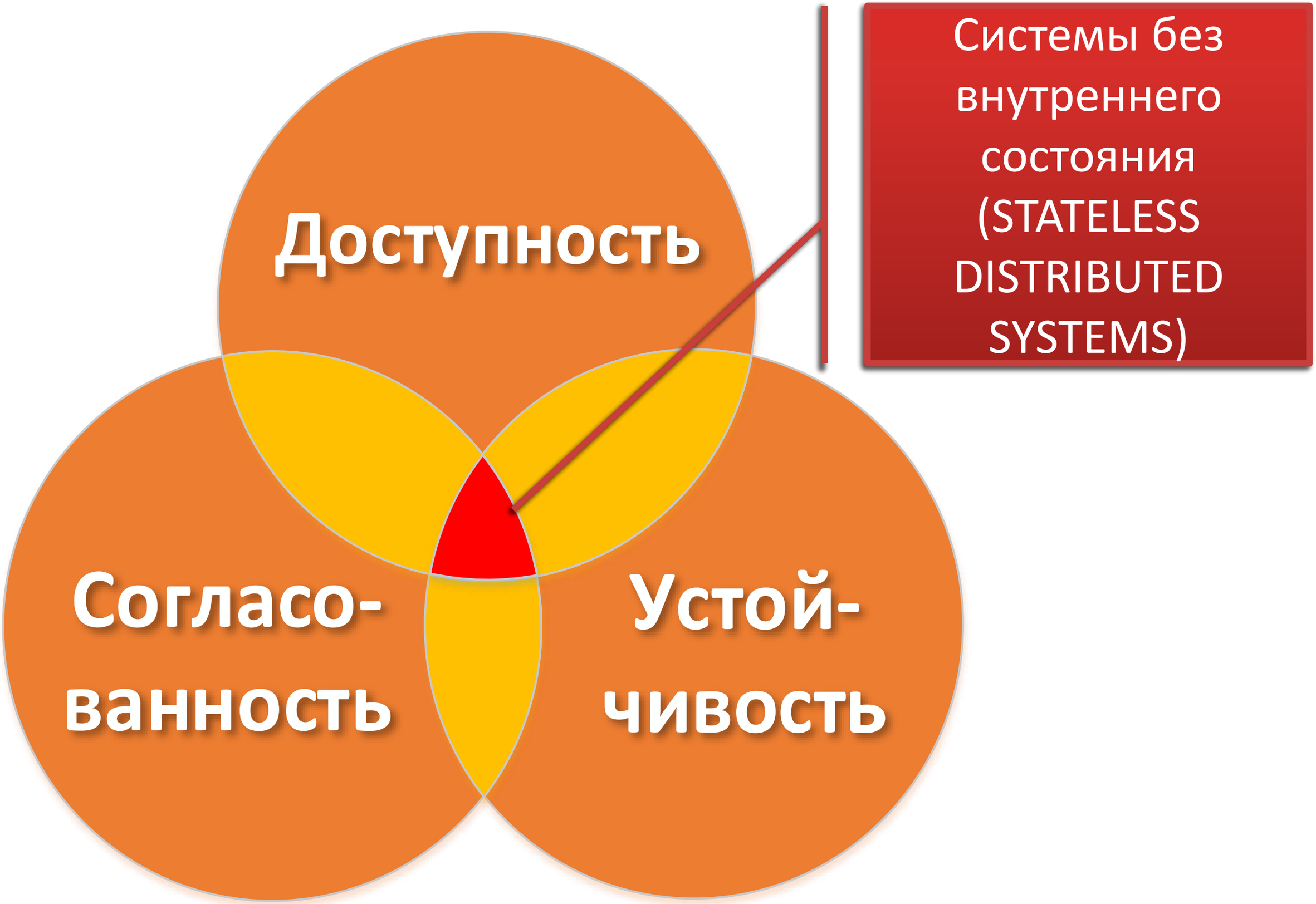
Выбор между C и A

- Нет 100% работающего универсального решения, каким путем решать задачу обработки запросов при возникновении расщепления распределенной системы – выбирать доступность или же выбирать
- Этот выбор может быть сделан только разработчиком, на основе анализа **бизнес-модели** приложения.
- Более того, этот выбор может быть сделан не для всего приложения в целом, а отдельно для каждой его части:
  - *Согласованность* для процедуры покупки и оплаты в интернет-магазине;
  - *Доступность* для процедуры просмотра каталога товаров, чтения отзывов и т.п.
- Или же смириться с несогласованностью, в случае коротких периодов (1-2 минуты) разделения системы

Доступность и согласованность



Исключение из CAP- теоремы?



Eventual Consistency

В связи с невозможностью 100% времени поддерживать согласованность и доступность системы, пришлось искать компромисс.

- *Согласованность в конечном счете (eventual consistency) или слабая согласованность (weak consistency)* - означает, что если в течение достаточно долгого периода времени в систему не поступают новые операции обновления данных, то можно ожидать, что результаты всех предыдущих операций обновления данных в конце концов распространятся по всем узлам системы, и все реплики данных **согласуются**.
- При отсутствии сбоев, максимальный размер окна несогласованности может быть определен на основании таких факторов, как задержка связи, загруженность системы и количество реплик в соответствии со схемой репликации.
- Самая популярная система, реализующая «согласованность в конечном счете» – DNS. Обновленная запись распространяется в соответствии с параметрами конфигурации и настройками интервалов кэшированя. В конечном счете, все клиенты увидят обновление.

Более подробно о согласованности в конечном итоге: http://habrahabr.ru/post/100891/

Заблуждения

- Сетевые соединения надежны
- Латентность связи всегда нулевая
- Пропускная способность сети бесконечна
- Сетевое соединение безопасно
- Топология сети не меняется
- У вашей системы всегда 1 администратор
- Стоимость передачи данных нулевая
- Сеть гомогенна

Реальность

- Сетевые соединения не надежны и часто падают
- Латентность всегда оказывает существенное влияние на работу приложения
- Пропускная способность сети очень даже конечна
- Сетевое соединение не безопасно
- Топология сети меняется очень часто, и чем дальше узлы друг от друга, тем чаще меняется топология
- У вашей системы множество администраторов, которые делают одни и те же вещи по-разному
- Стоимость передачи данных бывает очень высока
- Сеть гетерогенна – пакеты могут разделяться, их атрибуты могут быть затерты, часть из них может быть отфильтрована

Принципы проектирования распределенных интернет-приложений

Проектирование с учетом отказов

- Каждый компонент системы должен быть спроектирован с учетом возможных отказов.
- Проанализировать наиболее возможные виды отказов и продумать альтернативные потоки деятельности («bad path») для работы в периоды отказов
- Выбрать те части системы, которые могут продолжить работу, даже с учетом отказа, и предоставить их пользователю.

Минимизация меж-серверного траффика

- Чем меньше данных передается от одного сервера другому, тем меньше шансов, что что-то пойдет не так.
- Также, с учетом ограничений на пропускную способность сети, такая стратегия может значительно сократить время обработки операций и уменьшить затраты на работу
- Чем дальше располагаются узлы друг от друга, тем дороже в конечном итоге будет стоимость трафика между ними

Меньше состояния – меньше проблем

- Чем ближе система к «stateless», тем легче ее разрабатывать, проектировать, поддерживать, масштабировать.
- Для каждого блока данных, которые должны быть синхронизированы между серверами, разработчик должен решить, какую часть CAP-теоремы он будет для них поддерживать и каким образом.
- Возможным вариантом «перехитрить» CAP-теорему в этом случае – хранение данных у пользователя. Но это влечет за собой другие проблемы (безопасность и т.п.)

Разделяй и властвуй

» Чем лучше ваша система разделена на независимые, слабосвязанные компоненты, тем легче поддерживать ее работу в условиях возможных разделений:

> Возможно выделить те компоненты, которые останутся доступными пользователям в условиях разделения PBC

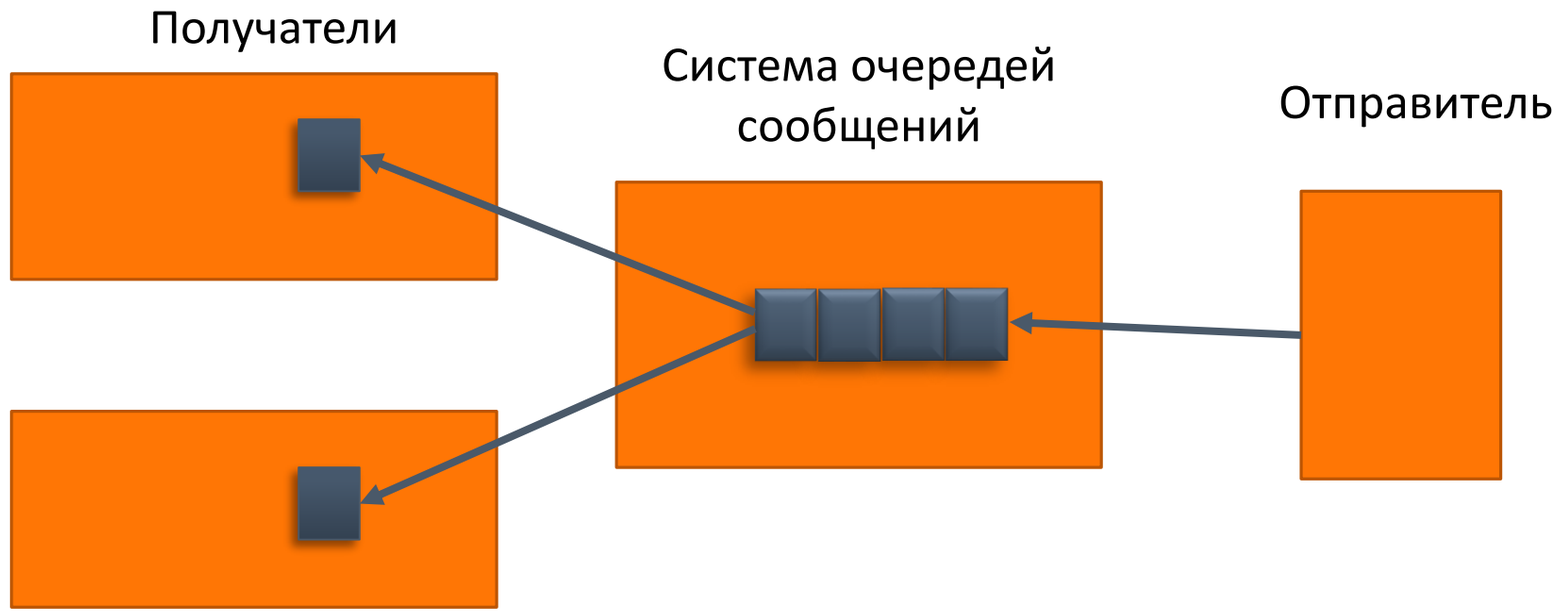
> Для каждого компонента можно сделать отдельное решение в координатах CAP-теоремы

Шифрование данных

- Необходимо обеспечить 100% шифрование данных которыми обмениваются ваши сервера
- Передача данных через Итернет – это очень небезопасная процедура
- Решением может быть применение таких технологий как VPN или SSH-туннели

Использование систем очередей

- Применение систем очередей сообщений (MQS - Message Queue Services) позволяет перевести в область асинхронной работы операции по записи данных, и избежать, таким образом, проблем при разрывах.
- MQS играют роль промежуточного ПО между серверами и обеспечивают возможность асинхронного обмена сообщениями, масштабирования и др.



Пакетная пересылка данных

- Латентность сети является серьезной проблемой при передаче большого количества маленьких наборов данных

- Лучше организовать сбор информации и переслать ее одним большим пакетом с одного узла на другой.

- Если есть возможность сделать передачу асинхронной – тем лучше.

«Обезьяна хаоса»

- «Обезьяна хаоса» (Chaos Monkey) – это подход к тестированию программно-аппаратных систем, основанный на постоянной генерации случайных сбоев во всех частях **функционирующей** системы.
- Такой подход приводит к необходимости разработки стабильной, легко-реконфигурируемой архитектуры которая не подвержена случайным программным или аппаратным сбоям

Документация проектных решений

- При разработке РВС придется сделать множество проектных решений, в том числе, каким образом каждый из разработанных компонентов ведет себя в условиях разрыва соединения
- Необходимо документировать все принятые проектные решения, с указанием мотивов, побудивших вас на принятие того или иного решения.

Паттерны «распределения» интернет-приложений

https://www.bluebox.net/about/blog/2012/04/distributed-systems-design-part-24/

Нулевой паттерн

- Это не паттерн распределенной системы, а база для всех остальных паттернов
- Для достижения данного паттерна приложения необходимо обеспечить:
  - > Стабильную работу системы на выделенном вычислительном узле (кластере);
  - > Архивацию всех критически-важных данных приложения на внешний узел.

«Ручной режим»

- В «ручном режиме» выделяется отдельный сервер, на удаленном ЦОД, на который вручную копируется определенная часть основного приложения.
- При возникновении проблем с основным сервером, администратор переключает DNS-сервис и все запросы перенаправляются на запасной сервер, который обеспечивает минимально-необходимую функциональность.
- Лучше всего, конечно же, автоматизировать процесс переключения, хотя бы посредством скриптов

Активный + ограниченная фунциональность (ожидает)

- Используется внешний DNS-сервис, который постоянно проверяет доступность базового «Активного» узла.
- Если он обнаруживает недоступность базового узла, он перенаправляет данные на запасной сайт с ограниченными функциональными возможностями.
- Лучше всего, если запасной сайт не будет завесить от текущих данных на базовом сайте и будет осуществлять работу в режиме «только для чтения».

Активный + полная фунциональность (ожидает)

- Аналогичен предыдущему, но запасной сайт должен постоянно синхронизироваться с активным сайтом для поддержания актуального состояния.
- В этом случае значительно усложняется процедура входа и выхода из исключительного состояния: проверка на корректность синхронизированных данных, актуальность и т.п.
- Более того, проблема, которая «выбила» основной сайт может быть синхронизирована на запасной, что может также привести к его нестабильности.

Активный + Активный

- Приложение делается распределенным изначально, а траффик перераспределяется между существующими активными сайтами.
- Такой паттерн часто реализуется на основе технологий *Глобальной Серверной Балансировки Нагрузки* (Global Server Load Balancing – GSLB), которая обеспечивает балансировку нагрузки серверов, расположенных в разных частях Интернета.
- Необходимо учитывать возможность отказа одного из активных узлов => закладывать возможность распределения дополнительной нагрузки на существующие узлы (50% при 2-х узлах; 33% при 3-х узлах и т.п.).
- Такой паттерн требует разработки приложения «с нуля», с учетом всех возможных решений для отдельных модулей в координатах CAP-теоремы.