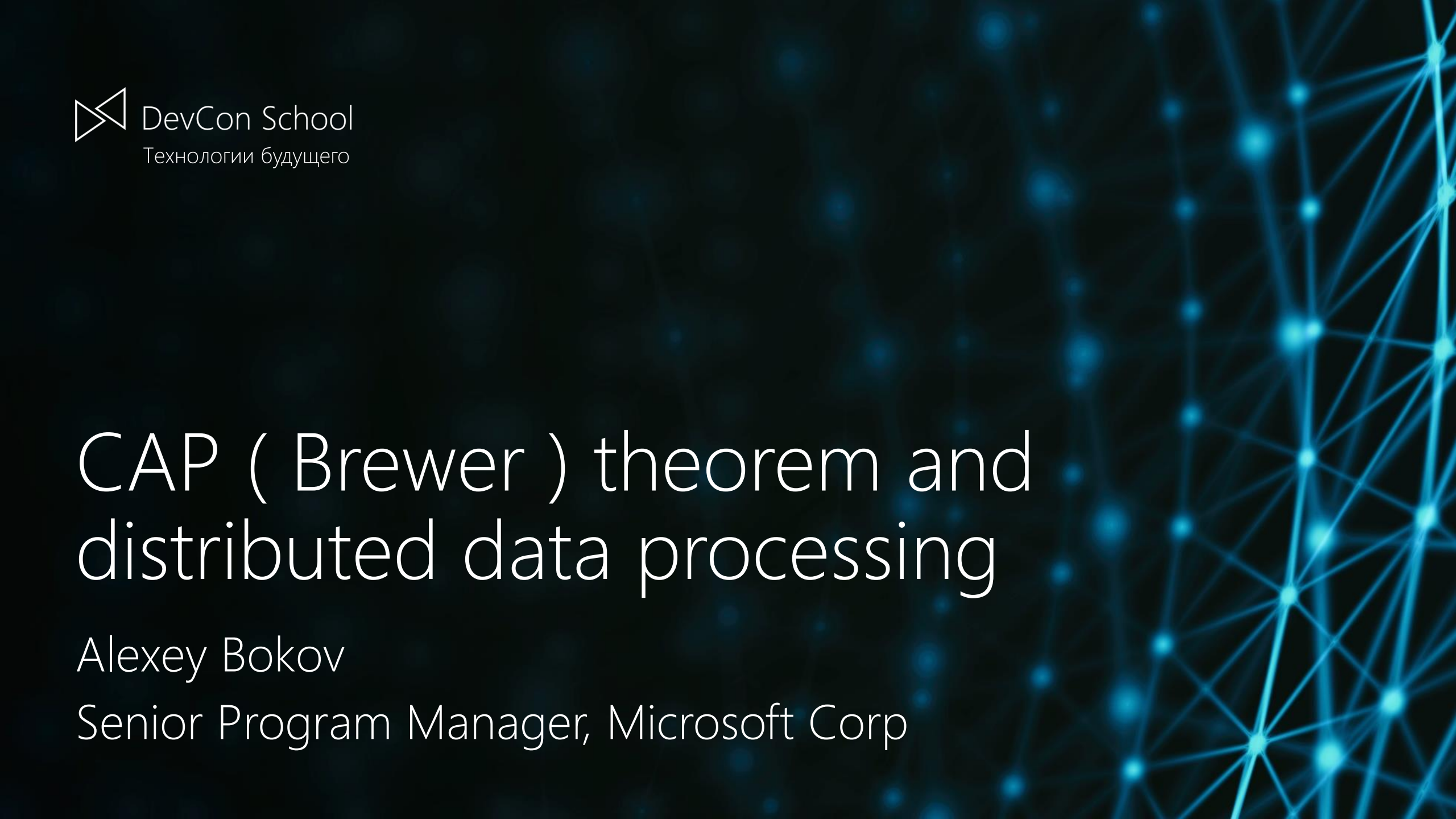
# DevCon School

Технологии будущего




# CAP ( Brewer ) theorem and distributed data processing

Alexey Bokov
Senior Program Manager, Microsoft Corp

# CAP теорема и немного Эрика Брьюера

Опубликована в 2000, доказана в 2002.

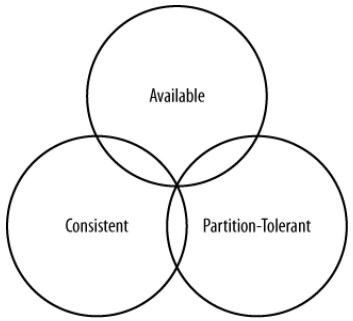
## @eric_brewer


CAP теорема указывает, что распределенная система может иметь максимум два свойства из трех одновременно:

1. Согласованность (Consistency) (атомарность) - каждая операция чтения, должна возвращать значение, установленное в последней операции записи: то есть у каждого узла есть актуальные данные о определенном объекте ( или данные во всех узлах не противоречат друг другу )

2. Доступность (Availability ) = Каждый запрос, полученный исправным узлом, влечет за собой ответ. Изначально было что : «почти все запросы должны получать ответы». Отсутствие ответа – это неответ по таймауту, и все ошибки вроде “сервер занят”.

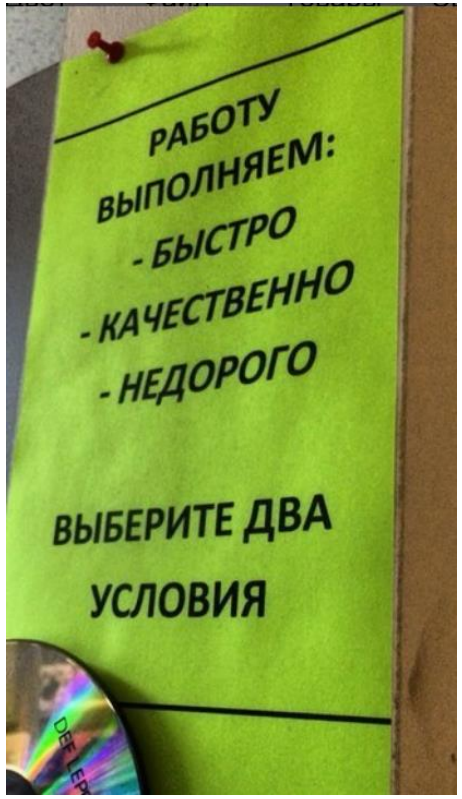
3. Устойчивость к разделению (Partition Tolerance) . Кластер сохраняет работоспособность при условии потери произвольного количества сообщений, посылаемых между узлами сети. Разделение означает, что все пакеты от узлов одной партиции не доходят до узлов другой партиции




- Consistency (согласованность данных)
- Availability (доступность)
- Partition tolerance (устойчивость к разделению)

$$PT \Rightarrow \neg(A \wedge C)$$

# Теорема Брюера и реальность





# CAP теорема как двигатель прогресса

## CAP теорема в 2000х годах была воспринята ИТ индустрией на “ура”

## Потому что теперь (наконец то, как раз в бум доткомов !) :

Приложение стали больше использовать веб технологии, поэтому самое время перестать беспокоиться насчет гарантированной целостности данных

Доступность ( т.е. чтобы сайт работал ) для веб приложений совсем даже не подразумевает надежности

Более того – гарантированная консистентность ( т.е. корретные данные на сайте ) это как раз то чего у нас быть не может ( см. Теорему Брюера )

Теперь любой у кого больше двух серверов может смело начинать интернет бизнес!

## В 2002 году утверждение Брюера было формально доказано в MIT

# Для начала это всё придумал не Брюер ;-)

4 июня 1976 году Sex Pistols выступили на сцене в первой раз – это было доказательством того что можно называться рок-группой и играть песни в которых будет не больше 2 аккордов.

```
[Verse 2]

     C                          F          C
   Anarchy for the UK, it's coming some time and maybe,
       C
    I give a wrong time stop a traffic line.
                C
    Your future dream is a shopping scheme,

First guitar solo:

E---------12----------------------14------------|
B---13-13----13-13-13-13-----15-15----15-15-15-15|
G-----------------------------------------------|  Repeat x3
D-----------------------------------------------|
A-----------------------------------------------|
E-----------------------------------------------|

E---------12----------------------|
B---13-13----13-13-13--------13---|
G---------------------------14----|
D---------------------------------|
A---------------------------------|
E---------------------------------|
```

***Наша группа знает 3 аккорда, но в одной песне может быть только 2****

# Всё придумал не Брюер ;-)

В последующем теорему 2 аккордов независимо доказали:
1976 – ‘Judy is a Punk’ by The Ramones (Eb5, Bb5 )
***1980 – Atmosphere by Joy Division (A, D ) ****
1985 – ‘Just Like Honey’ by Jesus and the Mary Chain (G#, D# )
***1995 – Common People by Pulp (C, G) ****
2007 – ‘505’ by Arctic Monkeys (Dm, Em)
***2010 – ‘Bloodbuzz Ohio’ by The National (A, F#m ) ****

** в коде и припеве используется еще один аккорд из основной тональности, но это не считается, т.к бас-гитарист все равно обыгрывает только два из них*

В России это целое музыкальное направление:

```
          Am                  Am
Цыпленок жареный, цыпленок пареный
                  E
Пошел по улице гулять.
        E           E
Его поймали, арестовали,
                   Am
Велели паспорт показать.

Паспорта нету - гони монету.
Монеты нет - снимай пиджак.
Цыпленок жареный, цыпленок пареный,
Цыпленка можно обижать.
```

# Вернёмся обратно в IT

## IT теория

- 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

## IT 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

# Вернёмся обратно в IT: важные замечания

Consistency подразумевает что при чтении мы получаем данные от _последней_ записи

Availability включает в себя понятие разумного ответа за определенное время – то есть если система не отвечает по таймауту или выдает ответ вида “сервер занят”, или любой другой нерелевантный цели запроса ответ, то это приравнивается к  неответу системы

Partition tolerance – устойчивость к разделению. Мы считаем потерянными также те сообщения, доставка которых превысила определенный лимит времени.

Если узел вышел из строя – это не ситуация разделения, потому что все равно есть только одна партиция ( а не 1  и N-1 как можно полагать ).

При выходе узла из строя система как раз может быть одновременно и согласованной и доступной, т.к. этот узел в любом случае не обрабатывает запросы, а когда он восстановится и у нас будет 1 и N-1 это узел может быть инициализирован актуальной репликой _всех_ данных.

Вывод: во время разделения мы выбираем либо доступность (минус консистентность) либо конситентность ( минус доступность ).

# Виды сервисов согласено CAP теореме

1) *Доступность сервиса важнее всего = AP ( Availability + Partition Tolerance - best effort availability)* системы – отвечаем на запросы, однако возвращенные данные не всегда могут быть актуальными и консистентными – например это DNS

2) *Важнее всего целостность данных при разделении данных ( минус доступность ) = CP ( Consistency + Partition tolerance ) системы* – отвечают на запросы всегда с корретными данными, но при проблемах с сетью, узлы могут не отвечать – Hbase, Google bigtable, большинство noSQL БД

3) *Важнее всего консистеность и доступность,но без возможности разделения данных = AC ( Availability + Consistency )* – возвращают всегда корретные данные, однако не имеют возможности разделения данных по сети при этом – примеры большинство реляционных баз данных

# На самом деле все сложнее

1) До тех пор, пока не наступила ситуация разделения система может вести себя как CA и только в случае возникновения партиций ей нужно выбирать между согласованностью и доступностью.

2) Свойства ( C, A, P ) на практике не являются бинарными :
    1) Доступность, очевидно, может измеряться по времени прошедшем между посылкой запроса и получением ответа.
    2) Какие-то системы могут позволить себе долго отвечать на запрос, другие должны сделать за определенный промежуток времени.
    3) Свойство устойчивости к разделению напрямую зависит от того, каким образом классифицировать сложившуюся ситуацию как разделение.
    4) У консистентности данных тоже могут быть разные степени

# Степени консистентности

1) Сильная/строгая согласованность (strong consistency). Операции чтения, произошешие после выполнения операции записи, всегда возвращают результат последней операции записи. Данный тип согласованности используется в CA системах, в частности в классических реляционных базах данных

2) Слабая согласованность (weak consistency). При данном типе согласованности система не гарантирует того, что операция чтения вернет результат последней операции записи.
    1) Существует ряд условий, необходимых для того, чтобы чтение вернуло самое последнее доступное значение.
    2) Период, в течении которого операция чтения возвращает устаревшие данные называется периодом несогласованности.

# Степени слабой консистентности

Согласованность в конечном счете (eventual consistency) - если операций изменения объекта нет, то в конечном счете операции чтения начнут давать наиболее актуальное значение. Наиболее популярной системой, использующей согласованность в конечном счете, является DNS: изменения имени распространяются по заданному алгоритму

Причинная согласованность: Если процесс А сообщил процессу Б, что А обновил объект, последующие запросы на чтение к Б вернут актуальное состояние объекта.

Read-your-writes согласованность: процесс А, после того, как он обновил объект, всегда возвращает обновленное состояние объекта.

Сессионная согласованность: процесс читает из системы хранения данных в контексте сессий. До тех пор, пока сессия существует, система гарантирует read-your-writes согласованность.

Согласованность монотонной записи: в этом случае, система гарантирует, что запись будет произведена. Данный тип согласованности является практически повсеместным.

# Немного про разные варианты

R – колиство read узлов, W – количество write узлов, N – количество узлов которые хранят реплику данных

1) R+W>N – строгая консистенстность, всегда найдутся такие узлы системы, которые будут участвовать как в операции записи, так и в операции чтения.

2) R = N/2, W=N/2 - когда операции записи и чтения имеют равную значимость для систем.

3) R->1, W->N - операция чтения выполняется намного чаще, чем операция записи, и перед системой не стоит требования «запись должна осуществляться всегда». Конфигурация (R=1, W=N) обеспечивает высокую доступность и устойчивость к разделению для операции чтения, т.к. узлу, чтобы возвратить ответ, не нужно соединение с другими узлами.

4) R->N, W->1 - переносит нагрузку с операции записи на операцию чтения. (R=N,W=1) характеризуется высокой доступностью и устойчивостью к разделению при операции записи, минус – отсутствие оптимистичкой блокировки

5) R+W<=N, то есть вероятность того, что узлов, участвующих в операции чтения и в операции записи, не будет, из чего следует, что система с такими настройками не может гарантировать сильную согласованность.

# Время ответа – от CAP к PACELC

CAP теорема не учитывает время ответа узла системы на запрос в качестве свойства, задержка и ситуация разделения тесно связаны друг с другом

Эту момент учитывает PACELC (in case of Partition, choose between Availability and Consistency, else – between Latency and Consistency ) – в случае разделения данных выбираем между CA ( доступностью и консистентностью ) и LC (задержкой и консистентностью) (Абади):

1. Разделение между доступностью и устойчивостью к разделению нечеткое.

2. Исходя из наблюдений Абади, те базы данных, которые предпочитают жертвовать консистетностью данных, имеют тенденцию жертвовать ей не только во время разделения, но также и при обычном функционировании.

Вывод: в случае нормального функционирования системы, следует выбирать между уменьшением времени ответа и согласованностью.

# Минимизация эффектов

1) обработка ситуаций  разделения как в вопросе обнаружения разделения
2) Процесс восстановления после разделения и поддержанию вариантов системы, которые могли  быть нарушены

Для выполненных операций храним версионные векторы, представляющие собой пары узлов, на которых они выполнялись, и времени, когда они выполнялись. если в системе будет две версии объекта, система сможет узнать, какая из них актуальней. Иногда возможна ситуация, когда актуальность определить нельзя – в таком случае считается, что некоторые изменения объекта выполнялись параллельно (Amazon Dynamo, Voldemort, Apache Cassandra, Riak, MongoDB )

Сегментирование архитектуры:

По данным
По операциям
По требованиям к функционалу
По пользователям
По иерархии

# Восстановление после разделения

Задачи:

1. Состояние объектов на всех узлах должно стать полностью консистентным
2. Компенсация ошибок, допущенных в результате разделения

Основным подходом к определению текущего состояния объекта является пошаговое исследование операций, происходившими с данными с момента начала разделения, и произведение корректирующих действий на каждом шаге.

# Компенсация ошибок

1) "последняя запись - финальная запись”

2) Перекладываем задачу на пользователя

Пример из реальной жизни - Примером процесса компенсации ошибок после разделения является ситуация, когда на самолет было продано больше билетов, чем есть в наличии. Процедура посадки может рассматриваться в данном случае как своего рода компенсация ошибки - в самолет пройдет не больше, чем есть сидений

Архитектурное решение - выполнение рискованных операций в ситуации разделения следует откладывать настолько, насколько это возможно

# Выводы

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.

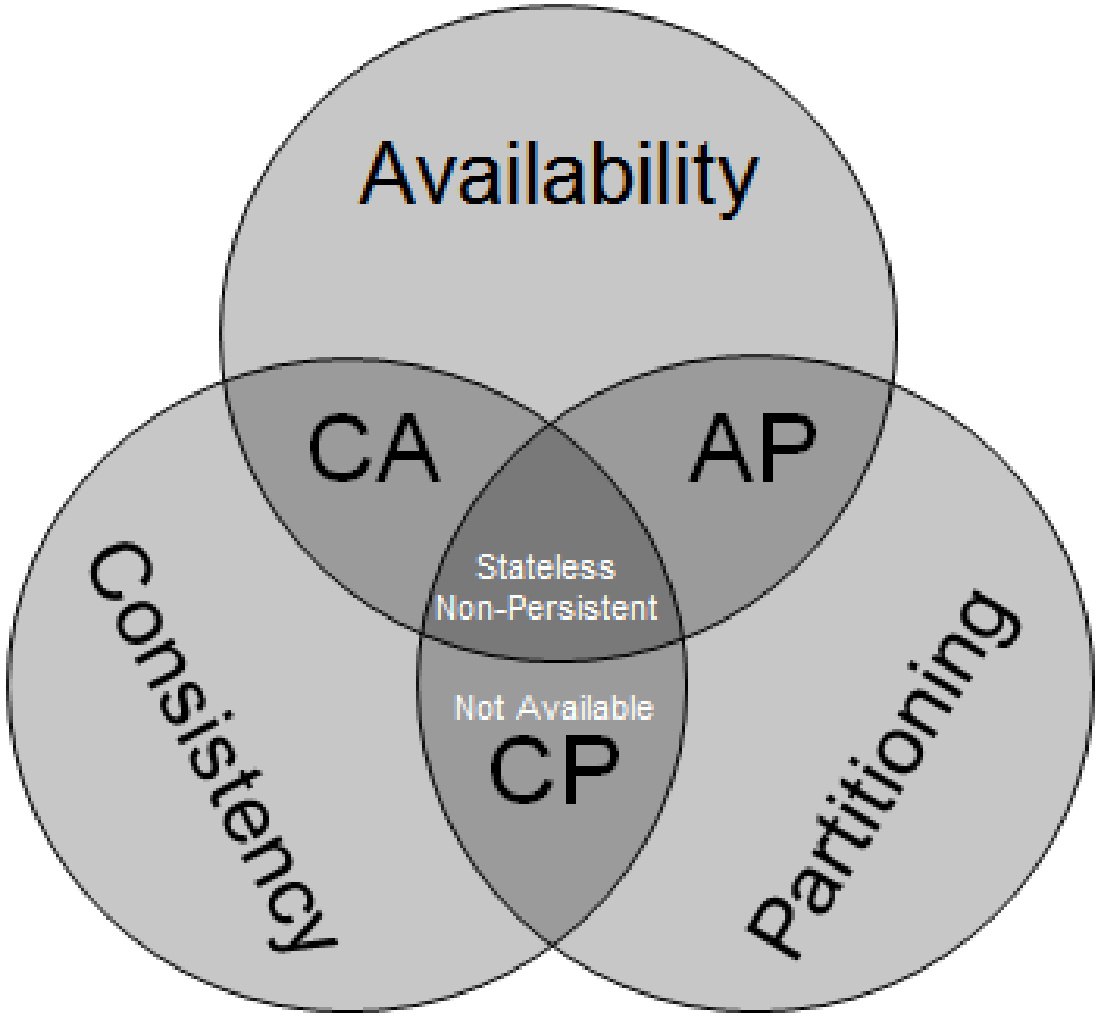
Более того, этот выбор может быть сделан не для всего приложения в целом, а отдельно для каждой его части:

Согласованность для процедуры покупки и оплаты в интернетмагазине;

Доступность для процедуры просмотра каталога товаров, чтения отзывов и т.п.

Можем смириться с несогласованностью, в случае коротких периодов (1-2 минуты) разделения системы

# Есть исключения из теоремы



# Что дальше

Масштабируемость – опять же возвращаемся к выборы между масштабируемостью и консистентностью

Устойчивость к атакам  - например DDoS не укладывется в модель разделения данных. DDoS атаки и прерывание работы веб сервисов требуют другого понимания выбора между консистентностью и доступностью.

Мобильные сети – параметры мобильных сетей (огромная вариативность всех параметров сети – скорость, доступность, задережки ) не укладываются в модель сетевого взаимодействия CAP теоремы:

Модель CAP теоремы предполагает что у нас есть относительно стабильные партиции которые изменяется раз в несколько минут. Для беспроводных сетей изменения гораздо более сильные, потери пакетов и изменение задержек.

Приложения в мобильных сетях больше ориентированы на геотаргетинг и социальное взаимодействие, в то время как CAP теорема больше ориентирована на веб 2000-х – большие веб порталы и e-commerce.

# Примеры из практики – рисуем на флипчарте

Сайт большого спортивного соревнования

Онлайн ресурсы для школы

Поисковая система – архитектуры, неответы, золотой шард

Системы по продаже авиабилетов

Q&A

# CAP ( Brewer ) theorem and distributed data processing

Alexey Bokov

Senior Program Manager, Microsoft Corp

abokov@microsoft.com; twitter: @abokov

#msdevcon

DevCon School

Технологии будущего

*Отзывы*

# Помогите нам стать лучше!

На вашу почту отправлена индивидуальная ссылка на электронную анкету. 2 ноября в 23:30 незаполненная анкета превратится в тыкву.

Заполните анкету и подходите к стойке регистрации за приятным сюрпризом!

# #msdevcon

Оставляйте отзывы в социальных сетях. Мы все читаем. Спасибо вам! ☺

DevCon School
Технологии будущего