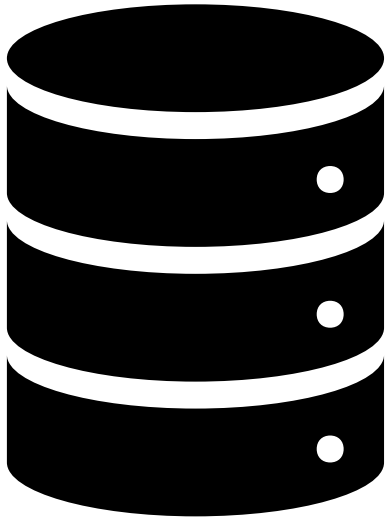


Базы данных

Лекция 11

NoSQL

Меркурьева Надежда
✉ merkurievanad@gmail.com
@merkurievanad

База данных

- совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных

- совокупность данных, организованных в соответствии с концептуальной структурой, описывающей характеристики этих данных и взаимоотношения между ними, причём такое собрание данных, которое поддерживает одну или более областей применения

Эволюция баз данных

- Иерархическая модель (1960-е)
- Сетевая модель (1969 г.)
- Реляционная модель (1969-1970 гг.)

….

А что же дальше?

Предпосылки

- Начало 2000х
  - Взрывной рост объёмов данных
    - Начало освоения интернета бизнесом
- Традиционные базы справляются плохо
  - Вертикальное масштабирование – дорого
    - Производительность не дотягивает до желаемой
- Инвестиции побуждают искать решение

Масштабируемость

- Система называется *масштабируемой*, если она способна увеличивать производительность пропорционально дополнительным ресурсам. Масштабируемость можно оценить через отношение прироста производительности системы к приросту используемых ресурсов. Чем ближе это отношение к единице, тем лучше.

- В системе с плохой масштабируемостью добавление ресурсов приводит лишь к незначительному повышению производительности, а с некоторого «порогового» момента добавление ресурсов не даёт никакого полезного эффекта.

Вертикальная масштабируемость

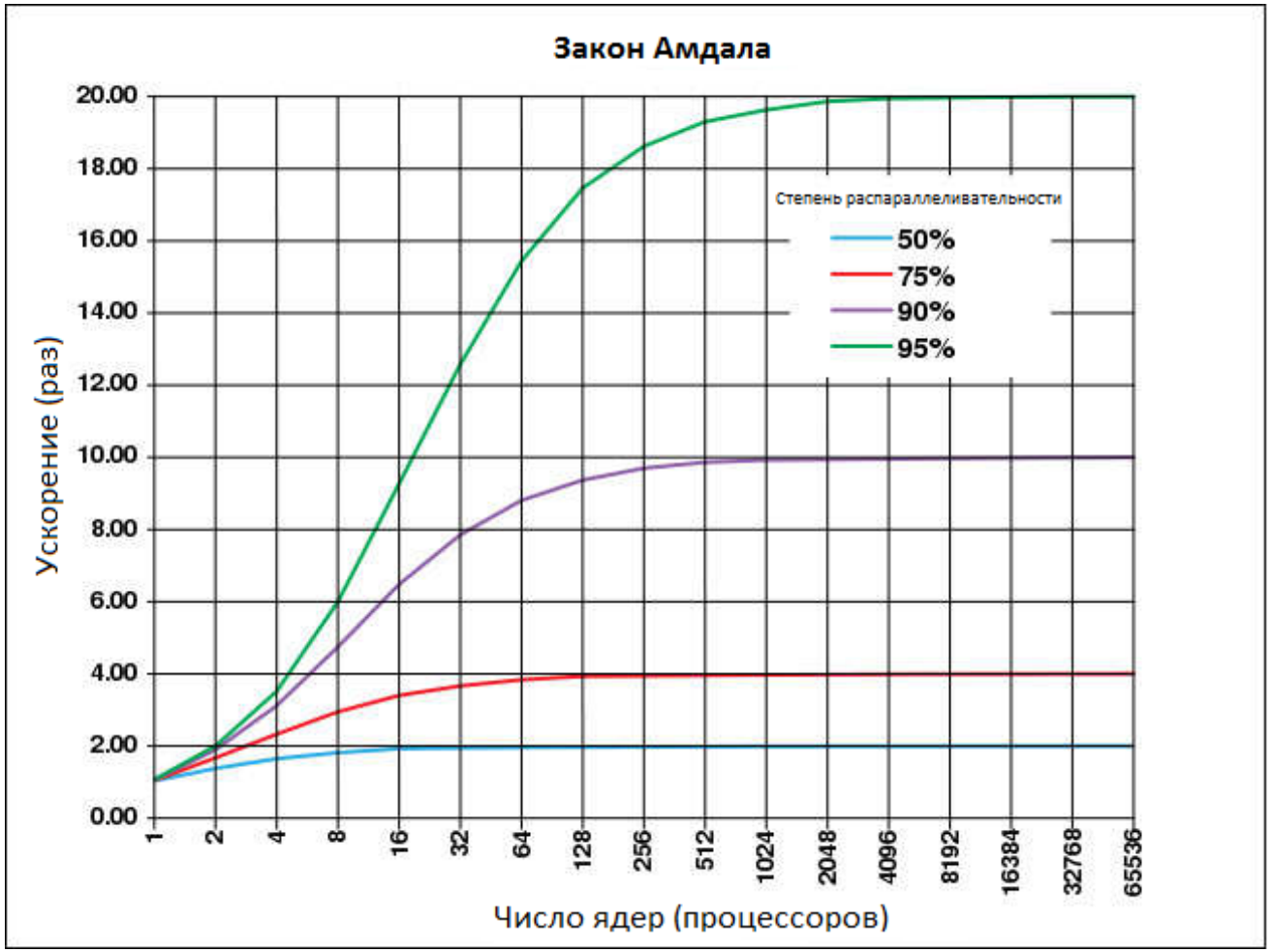
- **Вертикальное масштабирование** — увеличение производительности каждого компонента системы с целью повышения общей производительности. Масштабируемость в этом контексте означает возможность заменять в существующей вычислительной системе компоненты более мощными и быстрыми по мере роста требований и развития технологий. Это самый простой способ масштабирования, так как не требует никаких изменений в прикладных программах, работающих на таких системах.

Горизонтальная масштабируемость

- **Горизонтальное масштабирование** — разбиение системы на более мелкие структурные компоненты и разнесение их по отдельным физическим машинам (или их группам), и (или) увеличение количества серверов, параллельно выполняющих одну и ту же функцию. Масштабируемость в этом контексте означает возможность добавлять к системе новые узлы, серверы, процессоры для увеличения общей производительности. Этот способ масштабирования может требовать внесения изменений в программы, чтобы программы могли в полной мере пользоваться возросшим количеством ресурсов.

- Но! Необходимо учитывать закон Амдала.

Закон Амдала (1967 г.)



$$S_{latency}(s) = \frac{1}{(1-t') + \frac{t'}{s}}$$

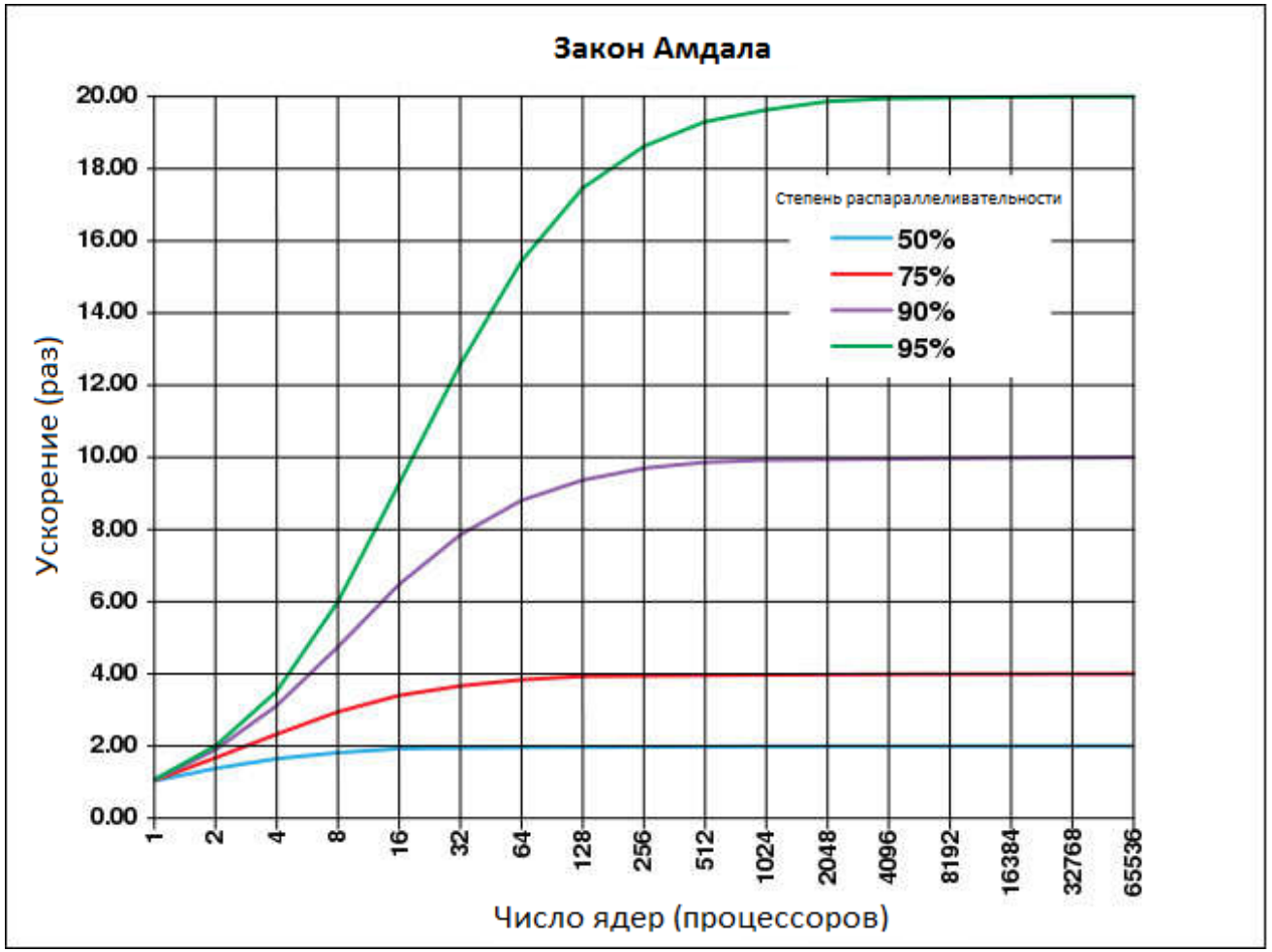
$S_{latency}$ – теоретическое ускорение выполнения всей программы

s – ускорение той части программы, на которую повлияет улучшение ресурсов

$t' = \frac{t}{T}$ – доля изначального времени выполнения распараллеливаемой части от общего времени выполнения программы

$$\begin{cases} S_{latency}(s) \leq \frac{1}{1-t'} \\ \lim\limits_{s\to\infty} S_{latency}(s) = \frac{1}{1-t'} \end{cases}$$

Закон Амдала (1967 г.)



$$S_{latency}(p) = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

$S_{latency}(p)$ – теоретическое ускорение выполнения всей программы

α – доля вычислений, которые могут выполняться **только** последовательно
$(1-\alpha)$ – доля вычислений, которые могут быть распараллелены идеально

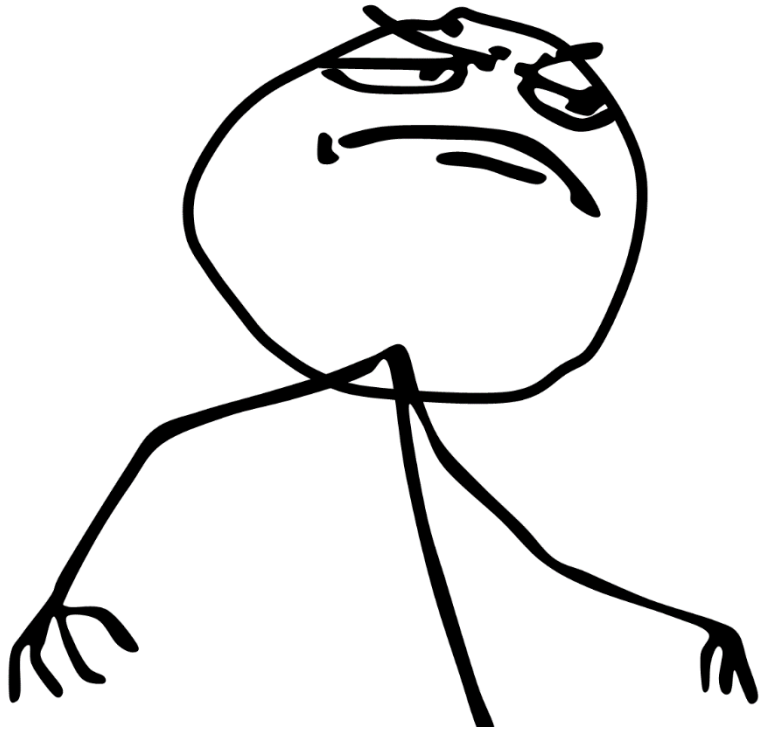
p – число задействованных узлов

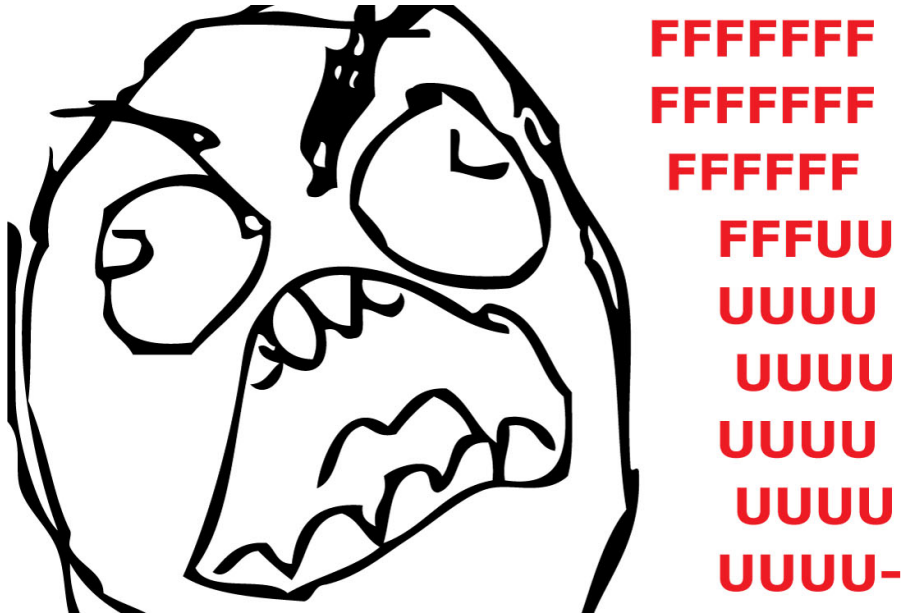
$$\begin{cases} S_{latency}(p) \leq \frac{1}{\alpha} \\ \lim\limits_{p \to \infty} S_{latency}(s) = \frac{1}{\alpha} \end{cases}$$

Закон Амдала (1967 г.)

α \ P	10	100	1000	10000	100000
0	10	100	1000	10000	100000
0,1	5,2632	9,1743	9,9108	9,991	9,9991
0,25	3,0769	3,8835	3,988	3,9988	3,9999
0,5	1,8182	1,9802	1,998	1,9998	1,99998
0,75	1,2903	1,3289	1,3329	1,33329	1,33333
1	1	1	1	1	1

Ожидание	Реальность
• Купил 100500 мощных компов • Собрал супербыструю распределенную систему …… • Все расчеты готовы за 3 наносекунды • Гугл и фейсбук борются за такого работника мечты	• Купил 100500 мощных компов • Собрал супербыструю распределенную систему …… • Не учел закон Амдала • 100500 компов тратят все время на взаимодействие друг с другом • Все считается медленнее, чем раньше





Закон Амдала (1967 г.)

- В случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого длинного последовательного фрагмента

- Для любой задачи с $\alpha \neq 0$ $\exists$ $\frac{1}{\alpha}$ максимальный **теоретический** прирост ускорения при идеальном распараллеливании
- Закон Амдала не учитывает время, затрачиваемое на взаимодействие параллельных процессов друг с другом. Начиная с некоторого момента, добавление узлов начинает тормозить систему, а не ускорять

Закон Густавсона-Барсиса (1988 г.)

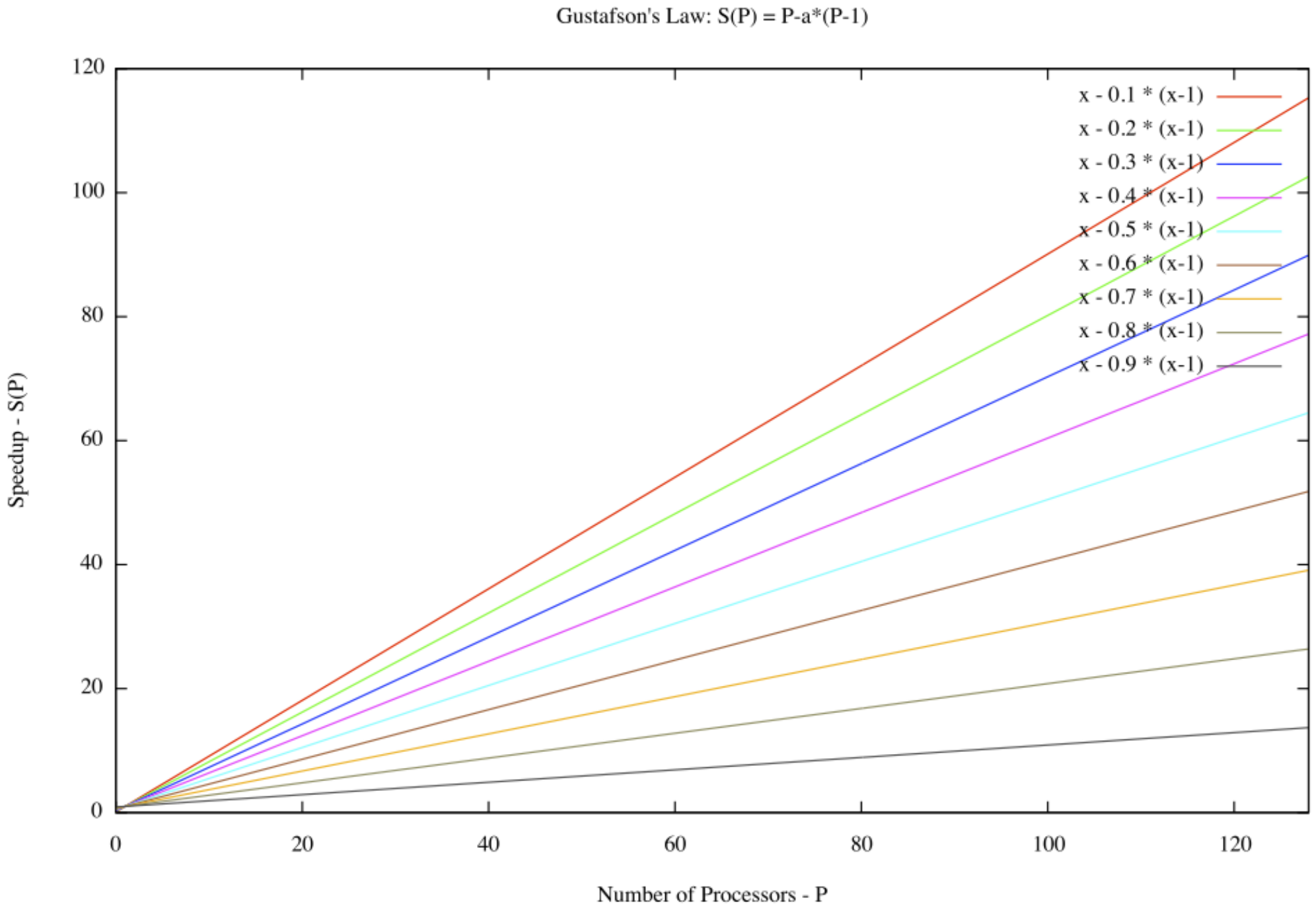
- Закон Амдала оценивает ускорение выполнения задачи фиксированного объема. Закон Густавсона-Барсиса оценивает ускорение с т.з объемов задач, которые можно решать за то же самое время:

$$S_{latency}(n, s) = s + (1 - s)n = n + (1 - n)s$$

$S_{latency}$ – теоретическое ускорение выполнения задачи

s – доля последовательных расчетов в программе

n – количество процессов

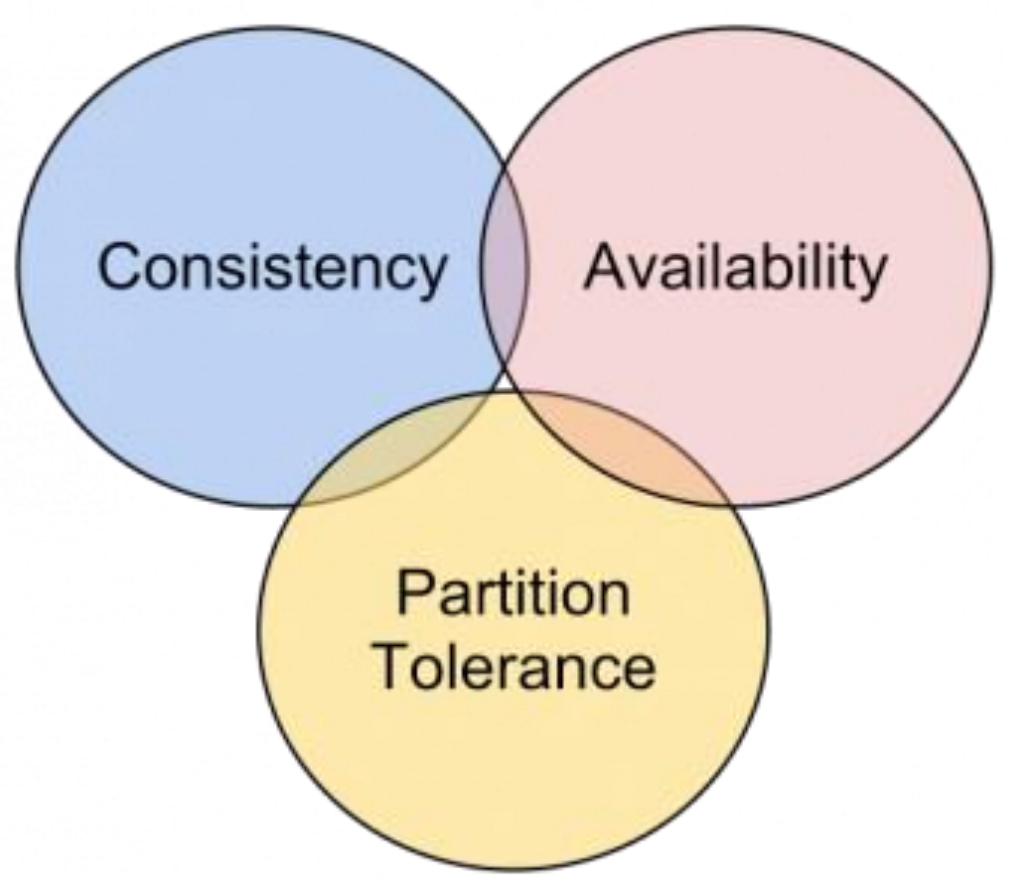

Gustafson's Law: S(P) = P-a*(P-1)
Speedup - S(P)
Number of Processors - P
0
20
40
60
80
100
120
x - 0.1 * (x-1)
x - 0.2 * (x-1)
x - 0.3 * (x-1)
x - 0.4 * (x-1)
x - 0.5 * (x-1)
x - 0.6 * (x-1)
x - 0.7 * (x-1)
x - 0.8 * (x-1)
x - 0.9 * (x-1)

Первые решения

- Google
  - Google File System (GFS) – распределённая файловая система
    - Первые публикации в 2003 году
    - Стал основой HDFS
    - Распределённое горизонтально-масштабируемое хранилище
      - Оптимально для append-only данных
- Google Bigtable - key-value хранилище
  - Первые публикации в 2006 году
  - Стал основой для Hbase
    - И Cassandra, но уже в Facebook

CAP-теорема

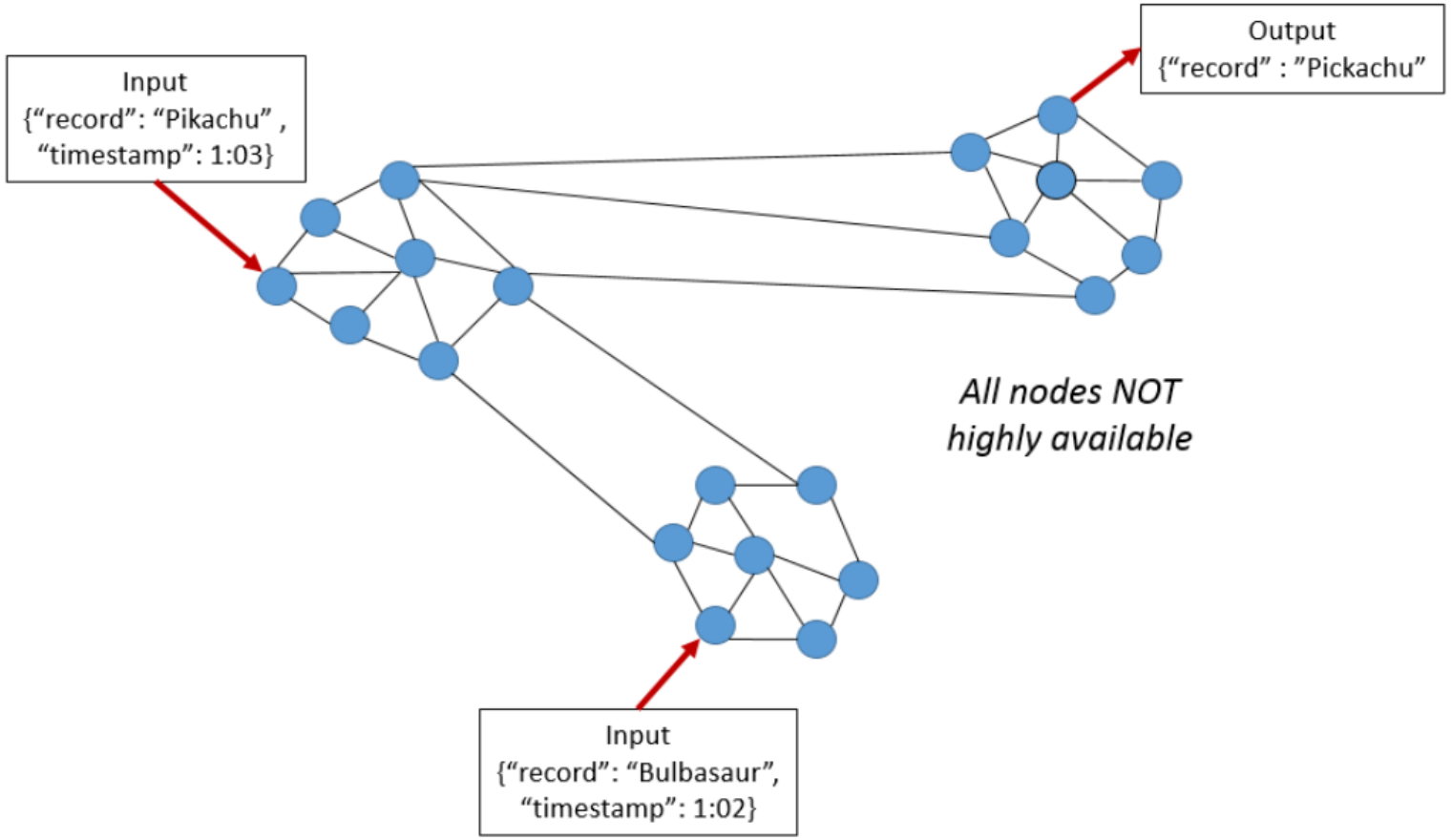
- Consistency
  - Согласованность
- Availability
  - Доступность
- Partitioning
  - Устойчивость к разделению



- В любой реализации распределённых вычислений возможно обеспечить не более двух из трёх свойств

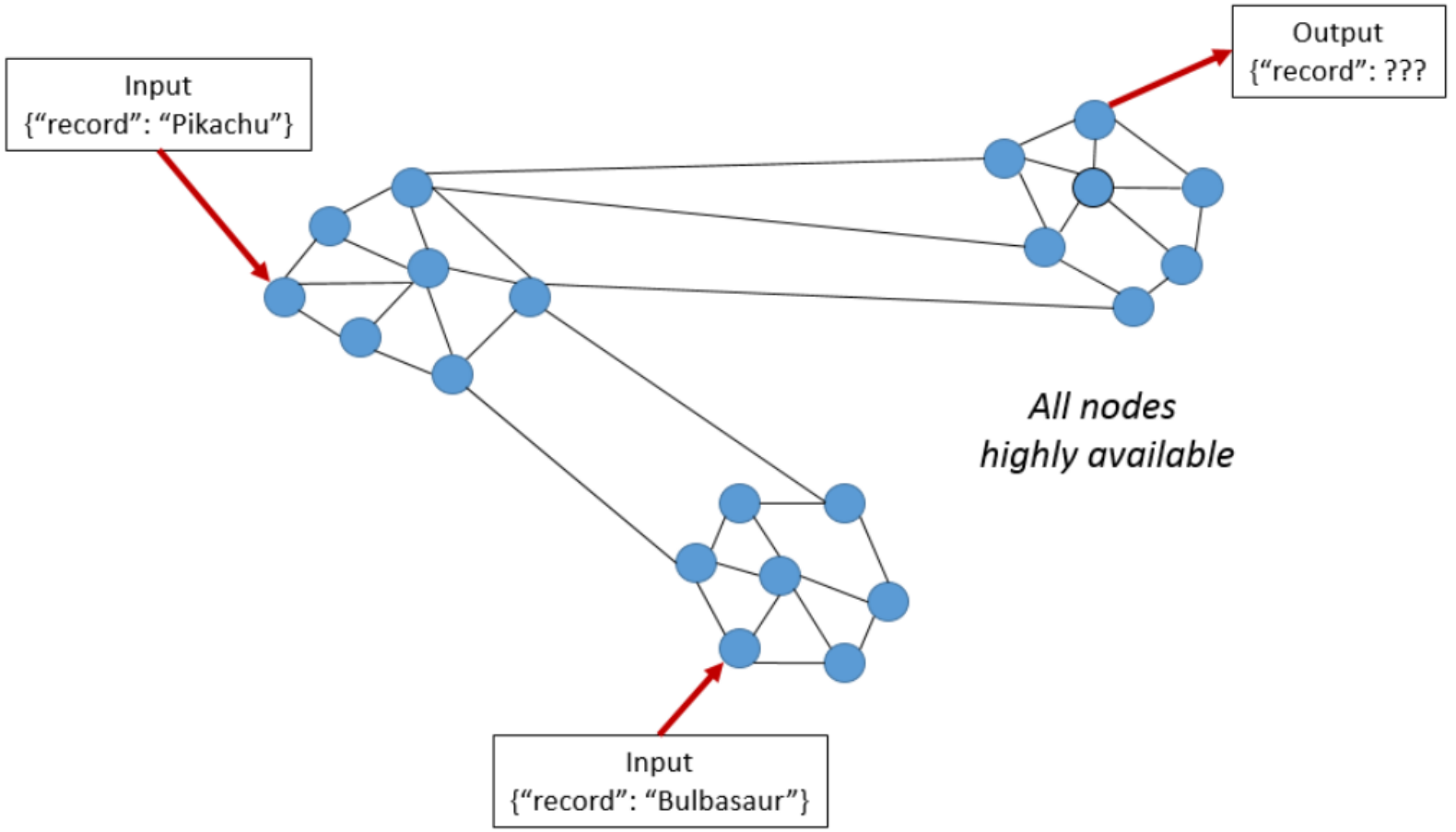
CAP: Consistency

при чтении гарантированно будут получены наиболее актуальные данные, либо ошибка



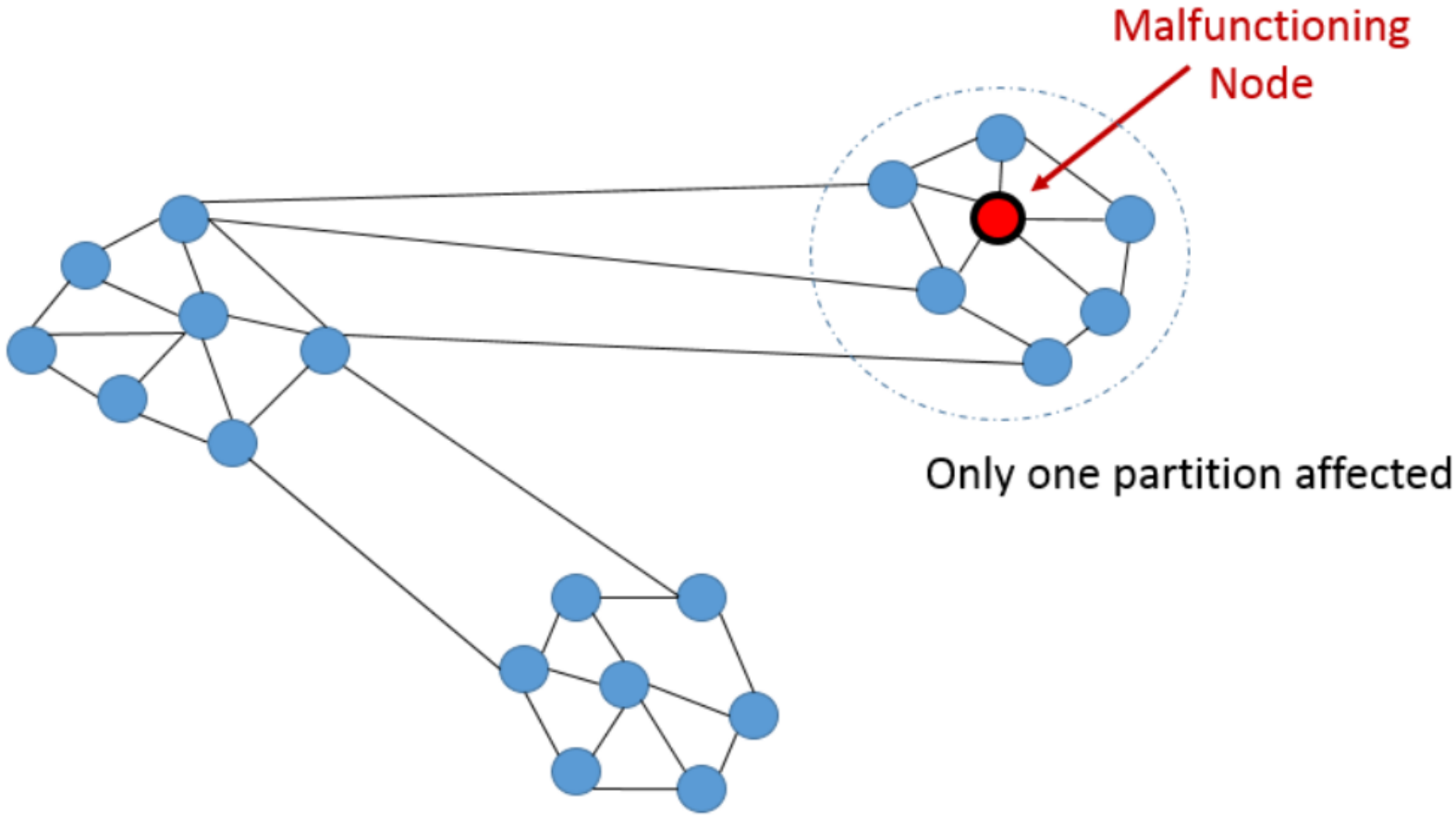
CAP: Availability

любой запрос к распределённой системе завершается корректным откликом, но без гарантии актуальности данных



CAP: Partitioning

система работает несмотря на потери / задержки в передачи сообщений между узлами



CAP: классификация систем

- CA характерные черты:
  - 2PC (two-phase commit protocol)
- CP характерные черты:
  - Pessimistic locking
- AP характерные черты:
  - Разрешение конфликтов
  - Time-to-live

Мораль CAP-теоремы

1. Нельзя получить все и сразу
2. Вопрос отказа от устойчивости к разделению обычно не стоит, поэтому выбор сводится к отказу от согласованности или доступности
3. При выборе между согласованностью и доступностью необходимо ориентироваться на конкретную задачу
4. Универсальных решений не бывает!

BASE

- В условиях CAP-теоремы ACID превращается в BASE
  - Basically Available
    - базовая доступность
      - Сбой части системы приводит к отказу в обслуживании только части пользователей
  - Soft-state
    - неустойчивое состояние
      - Отказ от гарантий хранения промежуточных данных
  - Eventually consistent
    - Итоговая согласованность
      - Существуют моменты в работе системы, когда можно видеть несогласованные данные

Классификации

- Ключ-значение (key-value)
  - Berkeley DB, MemcacheDB, Redis, Riak, Amazon DynamoDB
- Документно-ориентированные (document store)
  - Couchbase, MarkLogic, MongoDB, eXist, Berkeley DB XML
- Хранилище семейств колонок (column database)
  - Apache HBase, Apache Cassandra, Apache Accumulo
- Графовые базы данных (graph database)
  - Neo4j, OrientDB, AllegroGraph, InfiniteGraph, FlockDB, Titan
- Multi-model
  - Apache Ignite, ArangoDB, Couchbase, FoundationDB

Key-value database

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

- В основе – словарь / хэш-таблица
- Может иметь как итоговую согласованность, так и постоянную согласованность
- Может поддерживать порядок ключей
- Может хранить данные как в памяти, так и на диске
- Не имеет строгой типизации
- Начали активно развиваться после 2010

Document database

- Предназначена для хранения документов
- Является подклассом key-value БД
- Одна из наиболее распространенных NoSQL БД
- Имеют подкласс БД, оптимизированных под работу с XML
- Для извлечения данных использует внутреннюю структуру документа
- 1 объект – 1 экземпляр документа

Document database

- Данные в формате json:

```
{
    "FirstName": "Bob",
    "Address": "5 Oak St.",
    "Hobby": "sailing"
}
```

Document database

- Данные в формате xml:

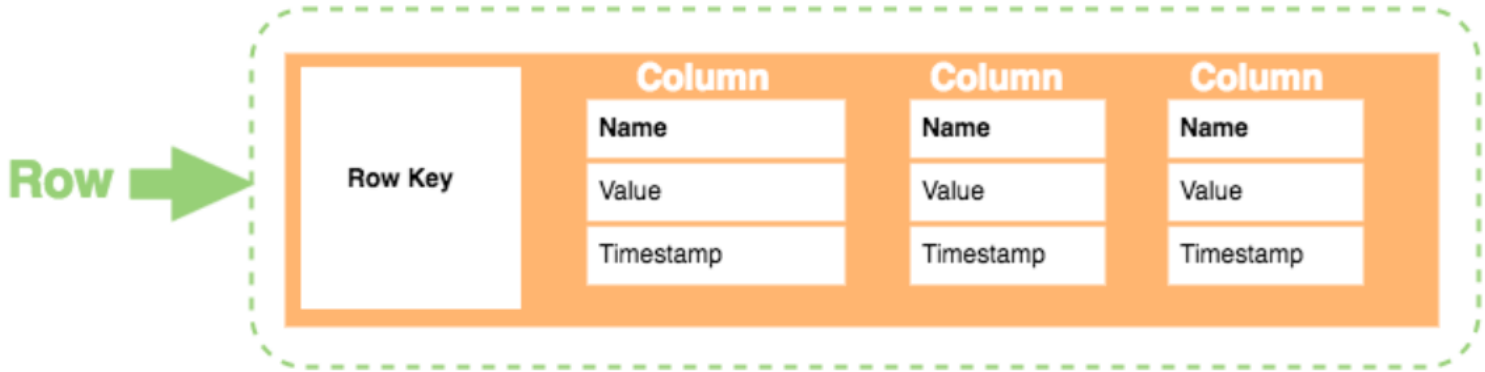
```
<contact>
   <firstname>Bob</firstname>
   <lastname>Smith</lastname>
   <phone type="Cell">(123) 555-0178</phone>
   <phone type="Work">(890) 555-0133</phone>
   <address>
     <type>Home</type>
     <street1>123 Back St.</street1>
     <city>Boys</city>
     <state>AR</state>
     <zip>32225</zip>
     <country>US</country>
   </address>
 </contact>
```

Document database

- Основные операции:
  - **C**reation (or insertion)
  - **R**etrieval (or query, search, read or find)
  - **U**pdate (or edit)
  - **D**eletion (or removal)

Column database

- Распределенная база данных:
  - Row Key – уникальный ключ строки
  - Column – значения, хранящиеся в колонке
- Хранение key-(value-pair):
  - Unique name – однозначно определяет имя колонки
  - Value – содержимое колонки
  - Timestamp – системное значение, которое служит для определения валидности значения



Column database

- Распределенная база данных
- Колонка соответствует строке
- Строки не обязаны содержать равное число колонок

Graph database

- В основе – граф, связывающий элементы данных
- Похожи на сетевые БД (см. лекцию 1), но позволяют быстрее проходиться по цепочке ребер
- Ни один язык запроса на графах на 2017г. не был официально принят, как SQL для реляционных
- Бывают полезны для описания моделей, которые можно представить в виде графов

Работа с данными

Операция JOIN не поддерживается, как тогда быть?

- Multiple queries
  - Если нужна пара джоинов, пишем несколько запросов, т.к. в общем случае запросы NoSQL быстрее, чем SQL
  - Если нужно много джоинов, пользуемся другим вариантом
- Caching, replication and non-normalized data
  - Храним не просто FK, но и сами внешние значения
  - Подходит для ситуаций, когда чтение намного чаще записи / обновления
- Nesting data
  - Для документно-ориентированных БД: сохраняем в документе все интересующие нас данные