Big Data Infrastructures

Philippe Cudre-Mauroux

Fall 2018

Lecture 5 – NoSQL

On Today's Menu…

Go BIG!

- Parallel DBMS
- CAP
- NoSQL intro
- Column-Family Stores
- Lab!
  - Hadoop bundle
  - HBase

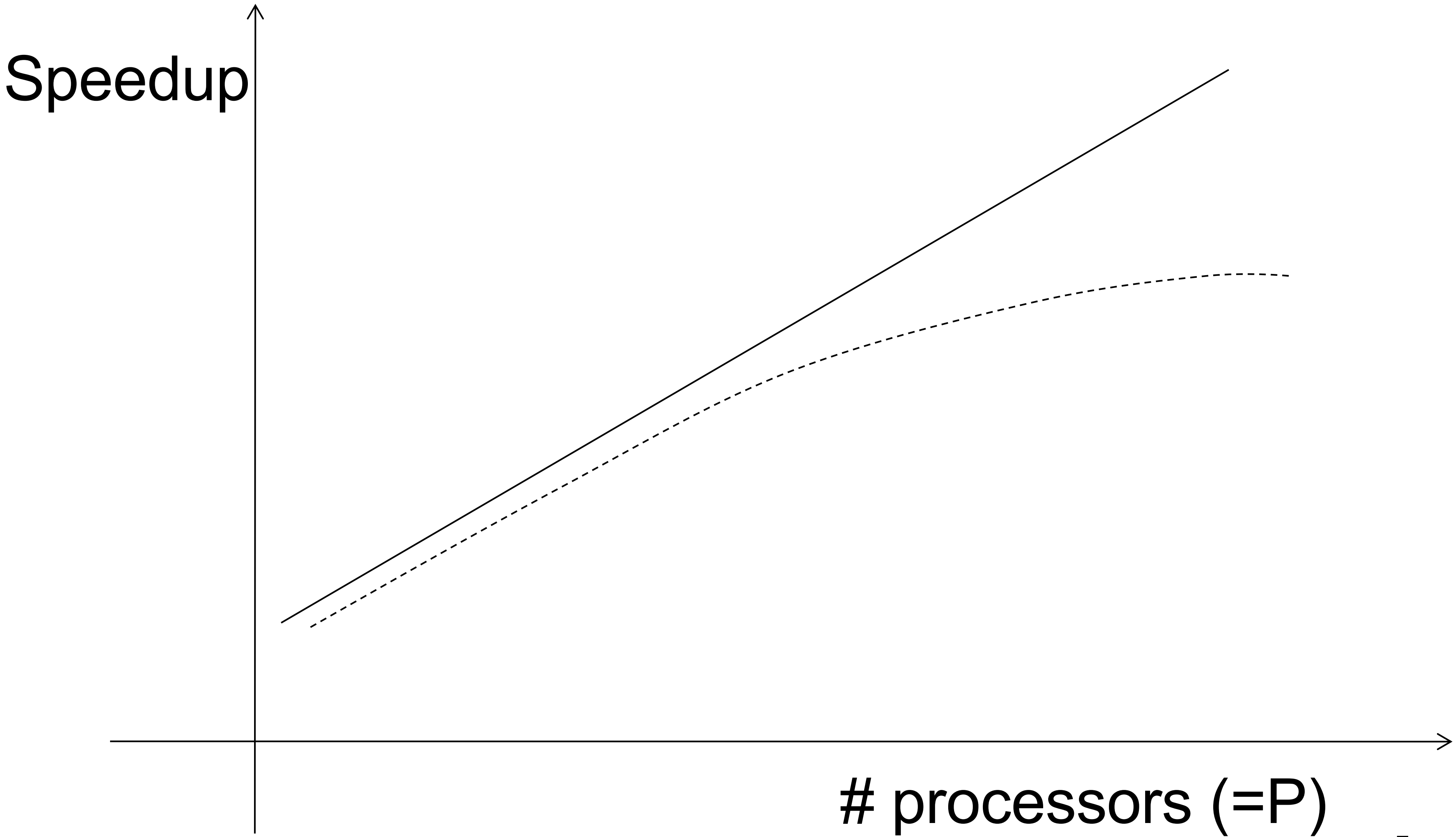
What’s wrong with my old DBMS?

- Managing Big Data is hard…
  - … extremely hard
  - Traditional DBMSs are 30 years old, were not meant for Big Data
    1. New queries (analytics, clustering, prediction, etc.)
    2. New data types (text, images, social graphs, sensor data, etc.)
    3. Impractical logical guarantees (ACID)

Parallel DBMSs

- Goal
  - Improve performance by executing multiple operations in parallel
- Key benefit
  - Cheaper to scale than relying on a single increasingly more powerful processor
- Key challenges
  - ACID compliance
  - Ensure overhead and contention do not kill performance

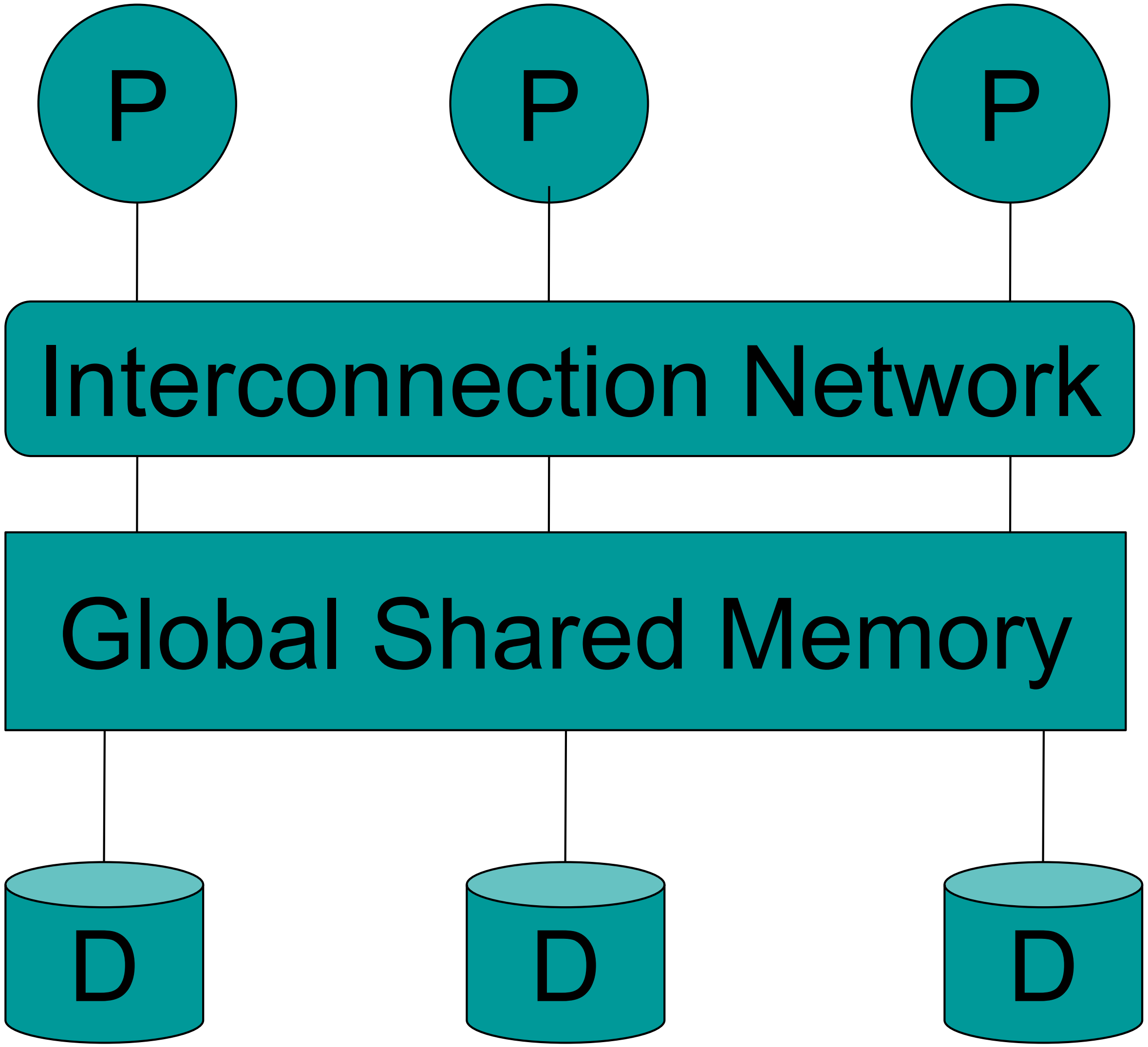
Linear v.s. Non-linear Speedup



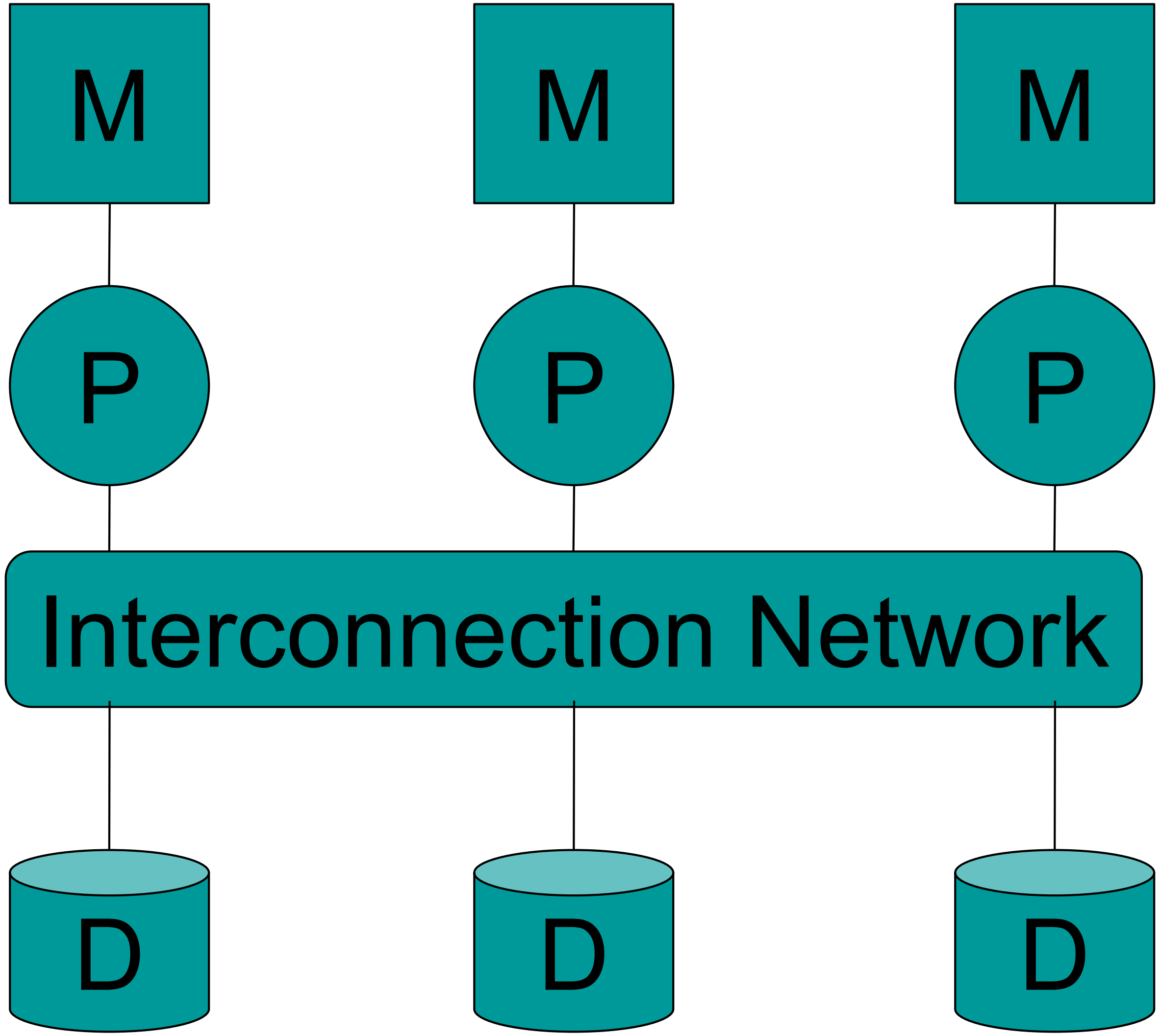
Architectures for Parallel Databases

- Shared memory
- Shared disk
- Shared nothing

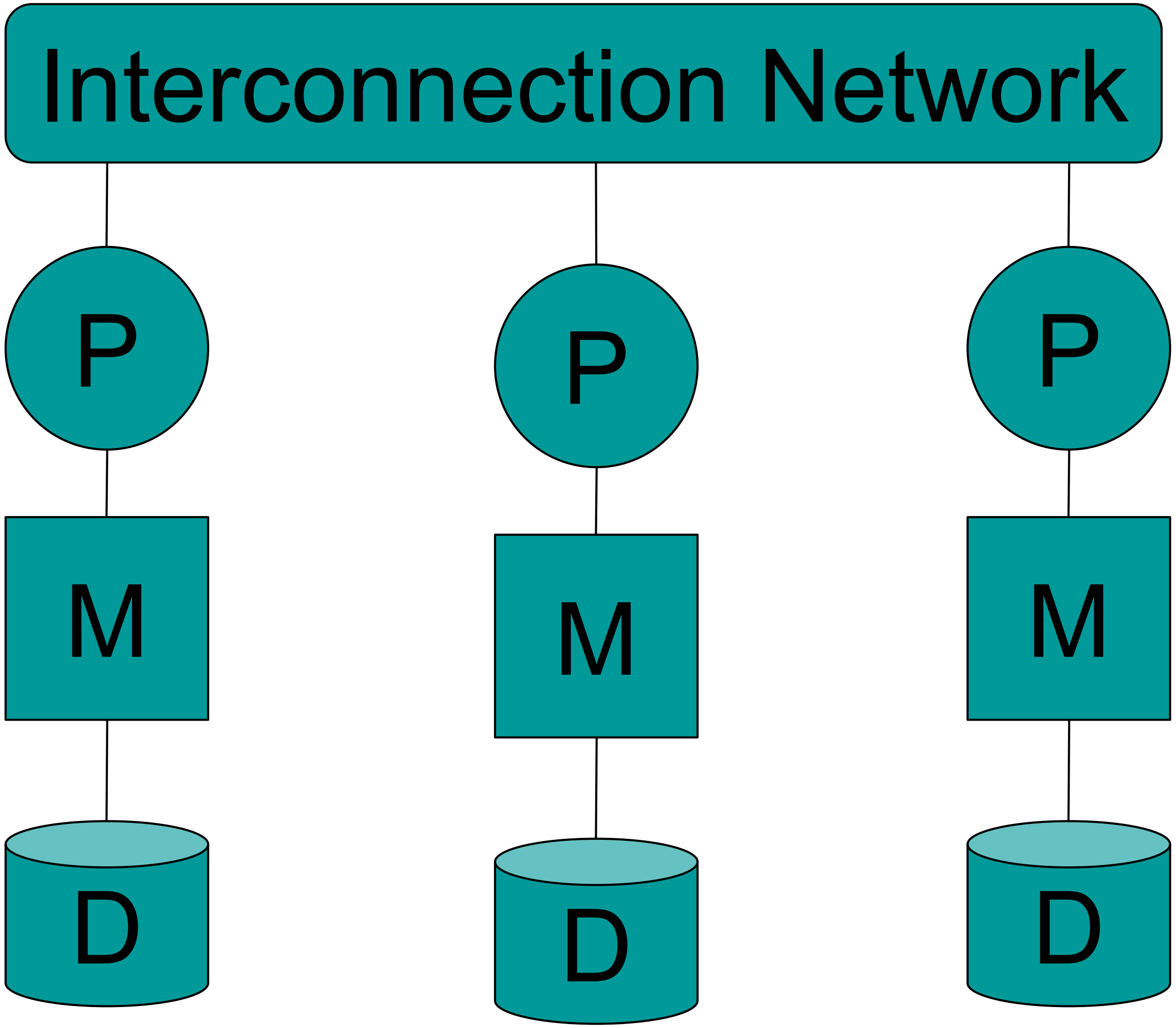
Shared Memory



Shared Disk



Shared Nothing



Shared Nothing

- Most scalable architecture
  - Minimizes interference by minimizing resource sharing
  - Can use commodity hardware
- Also most difficult to program and manage
- Processor = server = node
- P = number of nodes

We will focus on shared nothing

Horizontal Data Partitioning

- A.k.a. *Sharding* (mySQL, Google)
- Typical shared-nothing parallelization

- Relation R split into P chunks $R_0, \ldots, R_{P-1}$, stored at the P nodes
- Round robin: tuple t_i to chunk (i mod P)
- Hash based partitioning on attribute A:
  - Tuple t to chunk h(t.A) mod P
- Range based partitioning on attribute A:
  - Tuple t to chunk i if $v_{i-1} < t.A < v_i$

Parallel Selection

Compute $\sigma_{A=v}(R)$, or $\sigma_{v1<A<v2}(R)$

- On a conventional database: cost = B(R)
- Q: What is the cost on a parallel database with P servers?
  - Round robin
  - Hash partitioned
  - Range partitioned

Parallel Selection

- Answer:
  - Round robin: B(R); all servers do the work in parallel
  - Hash: B(R)/P for $\sigma_{A=v}(R)$; one server works only
    B(R) for $\sigma_{v1<A<v2}(R)$; all servers work in parallel
  - Range: (assuming relatively small range)
    B(R)/P; one server works only

Data Partitioning Revisited

What are the pros and cons ?

- Round robin
  – Good load balance but always needs to read all the data
- Hash based partitioning
  – Good load balance but works only for equality predicates and full scans
- Range based partitioning
  – Works well for range predicates but can suffer from data skew

Parallel Group By

- Compute $\gamma_{A,\ sum(B)}(R)$
- Step 1: server i partitions chunk R_i using a hash function h(t.A) mod P: R_{i0}, R_{i1}, …, $R_{i,P-1}$
- Step 2: server i sends partition R_{ij} to server j
- Step 3: server j computes $\gamma_{A,\ sum(B)}$ on R_{0j}, R_{1j}, …, $R_{P-1,j}$

Parallel Join

- Simplest implementation:
- Step 1
  - For all servers in [0,k], server i partitions chunk R_i using a hash function h(t.A) mod P: R_{i0}, R_{i1}, …, $R_{i,P-1}$
  - For all servers in [k+1,P-1], server j partitions chunk S_j using a hash function h(t.A) mod P: S_{j0}, S_{j1}, …, $R_{j,P-1}$

- Step 2:
  - Server i sends partition R_{iu} to server u
  - Server j sends partition S_{ju} to server u

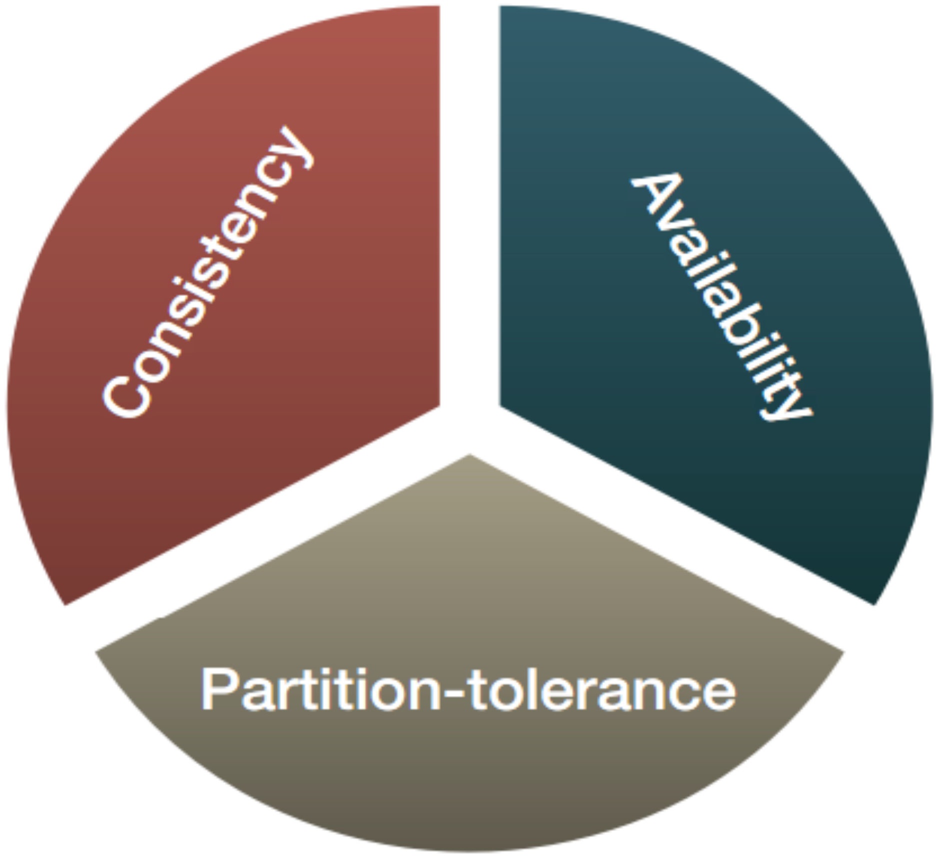
- Steps 3: Server u computes the join of R_{iu} with S_{ju}

ACID Properties (Standard DB)

- Atomicity: Either all changes performed by transaction occur or none occurs
- Consistency: A transaction as a whole does not violate integrity constraints (only valid tuples are written)
- Isolation: Transactions appear to execute one after the other in sequence
- Durability: If a transaction commits, its changes will survive failures

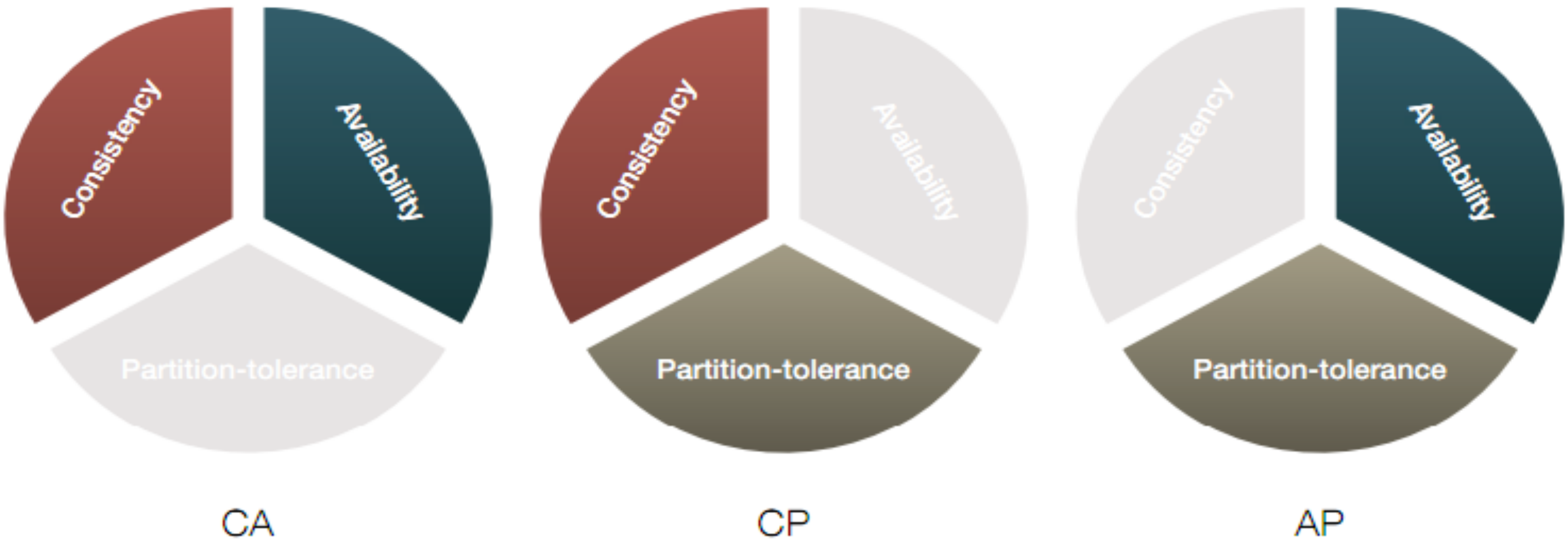
- Q: Benefits & drawbacks of providing ACID transactions?

CAP Properties



CAP Conjecture

- It is impossible for a distributed computer system to simultaneously provide all three guarantees
  - Pick at most two properties

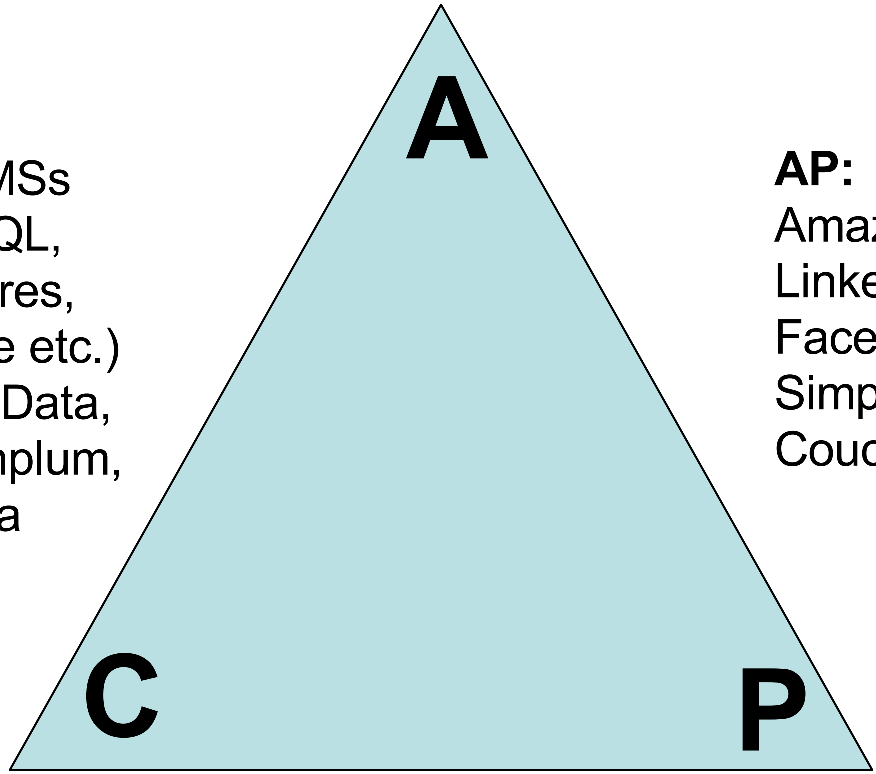


CAP Tradeoffs

- While it is impossible to provide all three properties, any two of those three properties can be achieved! Trivial examples for the asynchronous model:
  - CA: centralized RDBMS
  - CP: ignore all incoming requests
  - AP: always return initial value

Visual Guide to Recent Systems

CA:
RDBMSs
(MySQL,
Postgres,
Oracle etc.)
Aster Data,
Greenplum,
Vertica



AP:
Amazon Dynamo,
LinkedIn Voldemort,
Facebook Cassandra,
SimpleDB,
CouchDB

CP: Google BigTable, Hbase, Berkeley DB, MemcachDB, MongoDB

http://blog.nahurst.com/visual-guide-to-nosql-systems; classification subject to discussion…

Weak Consistency Models

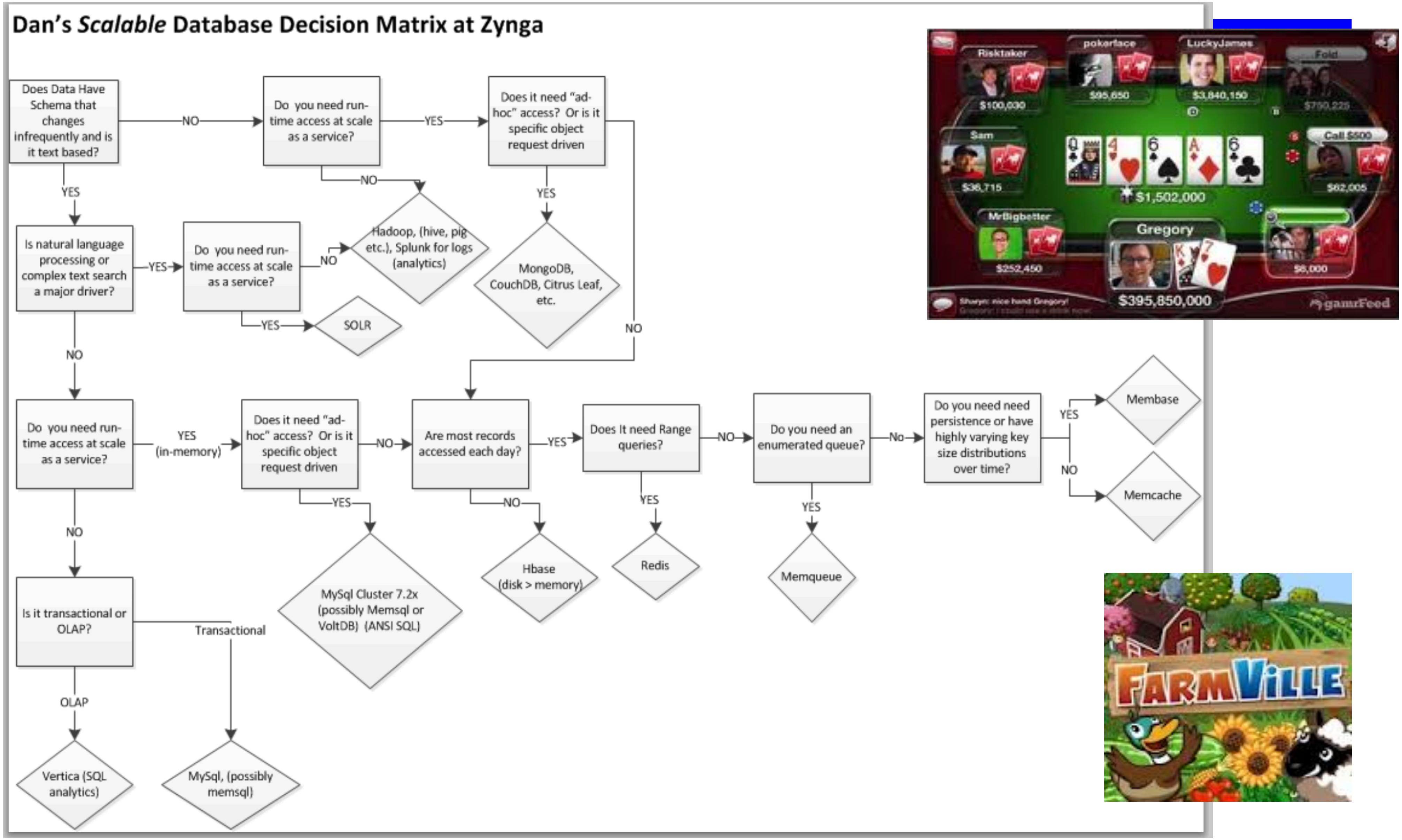
- Following the CAP theorem, systems started to appear providing weaker consistency models
  - failures are unavoidable in large scale systems
  - availability is a must for many services
  - compromise on consistency!
- Example: t-Connected Consistency (Lynch)
  - in the presence of no partitions, the system is consistent
  - in the presence of partitions, stale data can be returned
  - once a partition heals, there is a time limit on how long it takes for consistency to return

$\Rightarrow$ “Eventual” consistency in many noSQL systems

NoSQL / NewSQL Solutions

- The end of one-size-fits-all! [Stonebraker]
- Specialized solutions to ensure efficiency
  - Premium Data Warehousing [e.g., Teradata]
  - Column-stores [e.g., Vertica]
  - Wide columns [e.g., Cassandra]
  - Document Stores [e.g., MongoDB]
  - Graphs [e.g.,neo4j]
  - Arrays [e.g., SciDB]
  - Streams [e.g., Storm]
  - Etc.

A Concrete Example: Zynga



Key-Value (Column-Family) Store: Google BigTable

- **Many different types of data to manage today**
  - **Structured data**
    - All data conforms to a strict schema. Ex: business data
  - **Semi-structured data**
    - Some structure in the data but implicit and/or irregular
    - Ex: resume, ads, the Web in general
  - **Unstructured data**
    - All other data. Ex: text, sound, video, images
    - They actually do have some structure, but not for DBMSs!

Column-Family Store

- Distributed storage system storing (complex) key-value pairs
  - Key: (row+column) / value: (String)
- Designed to
  - Hold widely heterogeneous semi-structured data
  - Scale to thousands of servers
  - Store up to several hundred terabytes (maybe even petabytes)
  - Perform backend bulk processing
  - Perform real-time data serving
- To scale, Bigtable has a limited set of features and data types

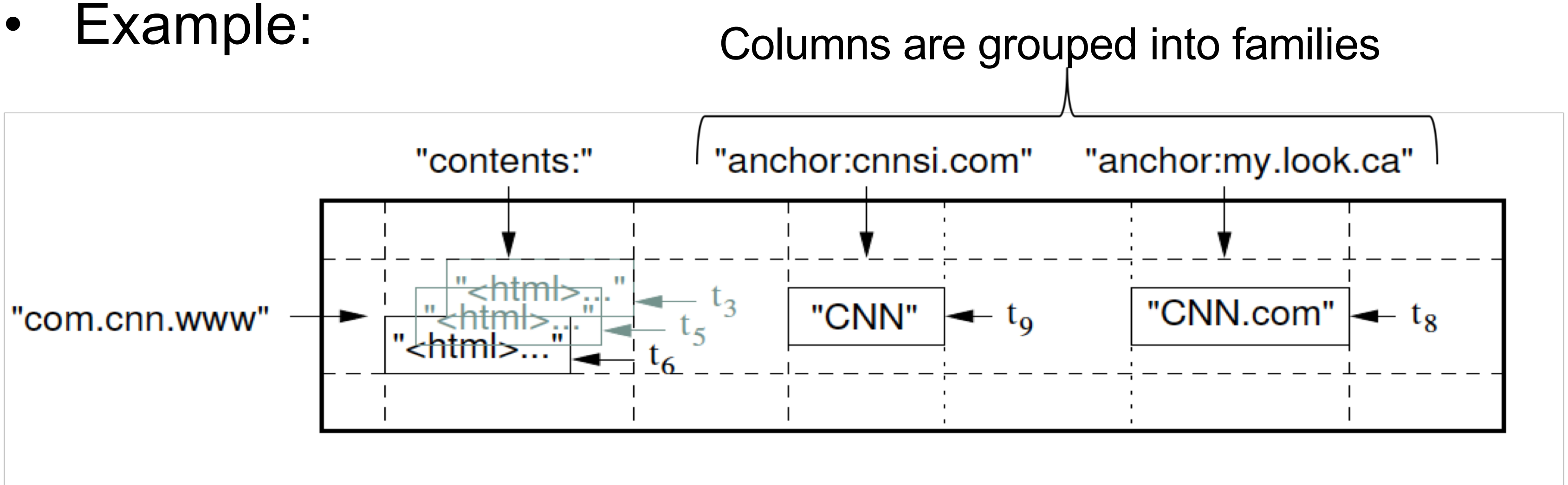
Bigtable Data Model

- Sparse, multidimensional sorted map

  `(row:string, column:string, time:int64)` ➔ `string`

  Notice how everything but time is a string

- Example:



- Above: storing Web pages and references

Other Key Facts about Big Table

- Read/writes of data under single row key is atomic
  - Only single-row transactions!
- Data is stored in lexicographical order
  - Improves data access locality
- Column families are unit of access control
- Data is versioned (old versions can be garbage-collected)
  - Example: most recent three crawls of each page, with times
- Current open-source systems (Apache Cassandra, Hadoop's HBase, etc.) are directly inspired from BigTable