POLITECNICO
MILANO 1863

SYSTEMS AND METHODS FOR BIG AND UNSTRUCTURED DATA

Towards NoSQL

Marco Brambilla

marco.brambilla@polimi.it

@marcobrambi

POLITECNICO
MILANO 1863

Agenda

- Schema
- Scalability
- Transactional Properties
- CAP Theorem
- ACID vs. BASE
- NoSQL models and solutions

Towards Big Data Architectures

Big Data vs. Traditional Data

	Traditional	Big Data
Data Characteristics	Relational (with highly modeled schema)	
Cost	Expensive (storage and compute capacity)	Commodity (storage and compute capacity)
Culture	Rear-view reporting (using relational algebra)	Intelligent action (using relational algebra AND ML, graph, streaming, image processing)

One keyword:

FLEXIBILITY

Schema

The schema-less idea

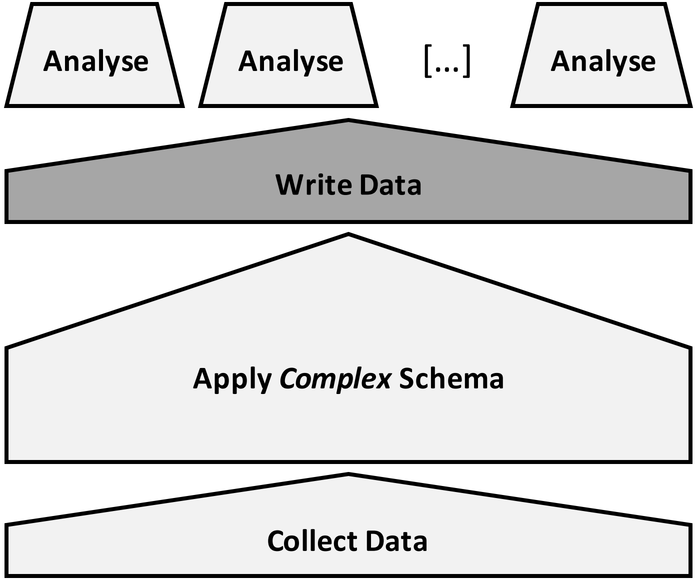
- Schema-less or implicit schema?

- Aggregate-based
- Key-value and document-based
- Relationship-based
- Graph DBs are better than relational!

- Ref. Domain-Driven Design, by Eric Evans.

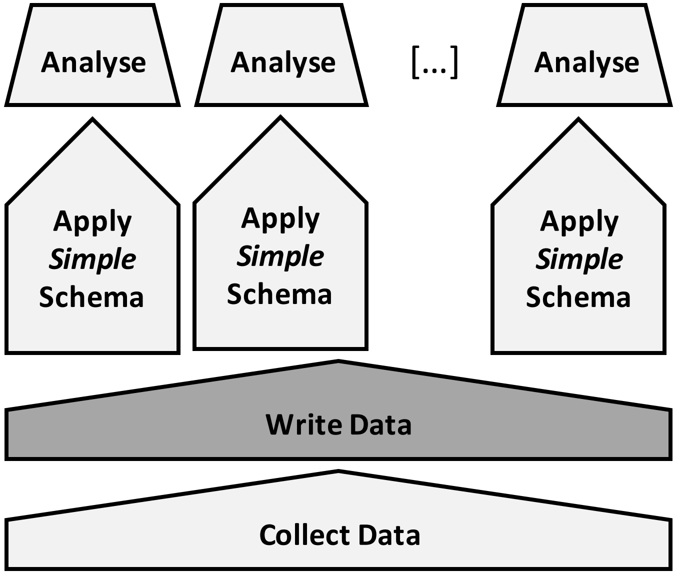
Paradigmatic shift introduce by Big Data: from Schema On Write …

- Long lasting discussion about the schema that can accommodate all needs
- Some analysis can no longer be performed because the data were lost at writing time

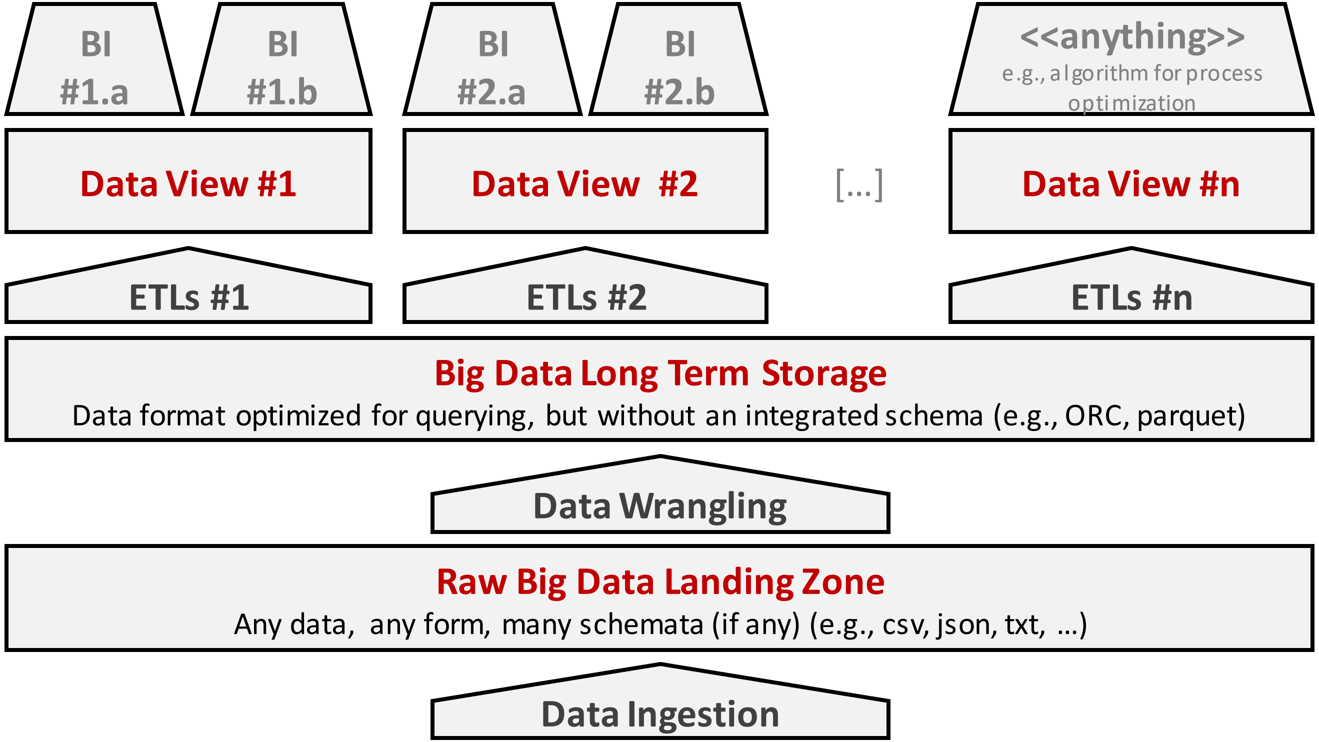


Paradigmatic shift introduce by Big Data: … to Schema On Read

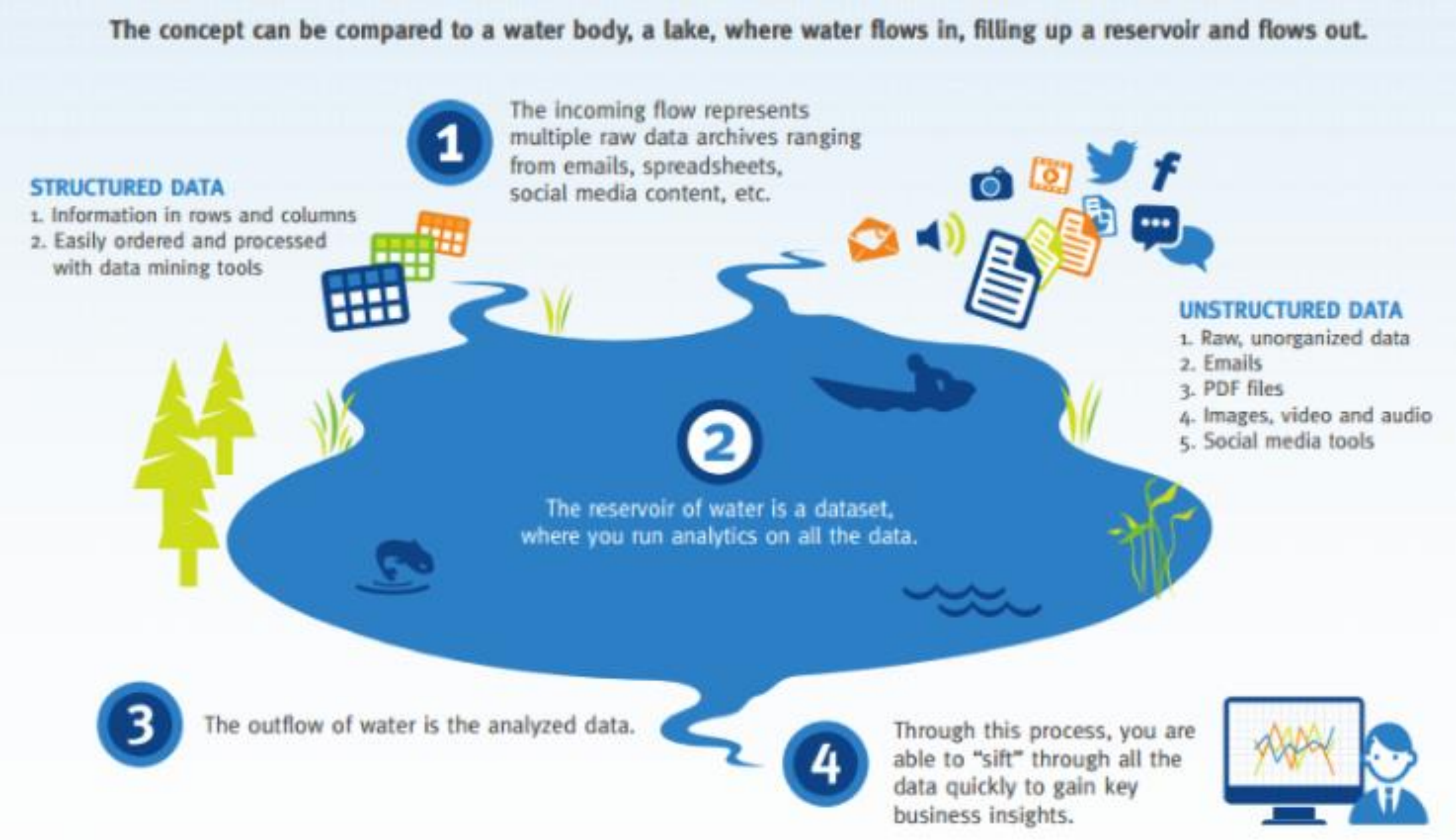
- Load data first, ask question later
- All data are kept, the minimal schema need for an analysis is applied when needed
- New analyses can be introduced in any point in time



Logical Architecture of Big Data


BI #1.a
BI #1.b
BI #2.a
BI #2.b
<<anything>>
e.g., algorithm for process optimization
Data View #1
Data View #2
[...]
Data View #n
ETLs #1
ETLs #2
ETLs #n
Big Data Long Term Storage
Data format optimized for querying, but without an integrated schema (e.g., ORC, parquet)
Data Wrangling
Raw Big Data Landing Zone
Any data, any form, many schemata (if any) (e.g., csv, json, txt, ...)
Data Ingestion

The Concept of Data Lake



The concept can be compared to a water body, a lake, where water flows in, filling up a reservoir and flows out.

1 The incoming flow represents multiple raw data archives ranging from emails, spreadsheets, social media content, etc.

STRUCTURED DATA

1. Information in rows and columns
2. Easily ordered and processed with data mining tools

UNSTRUCTURED DATA

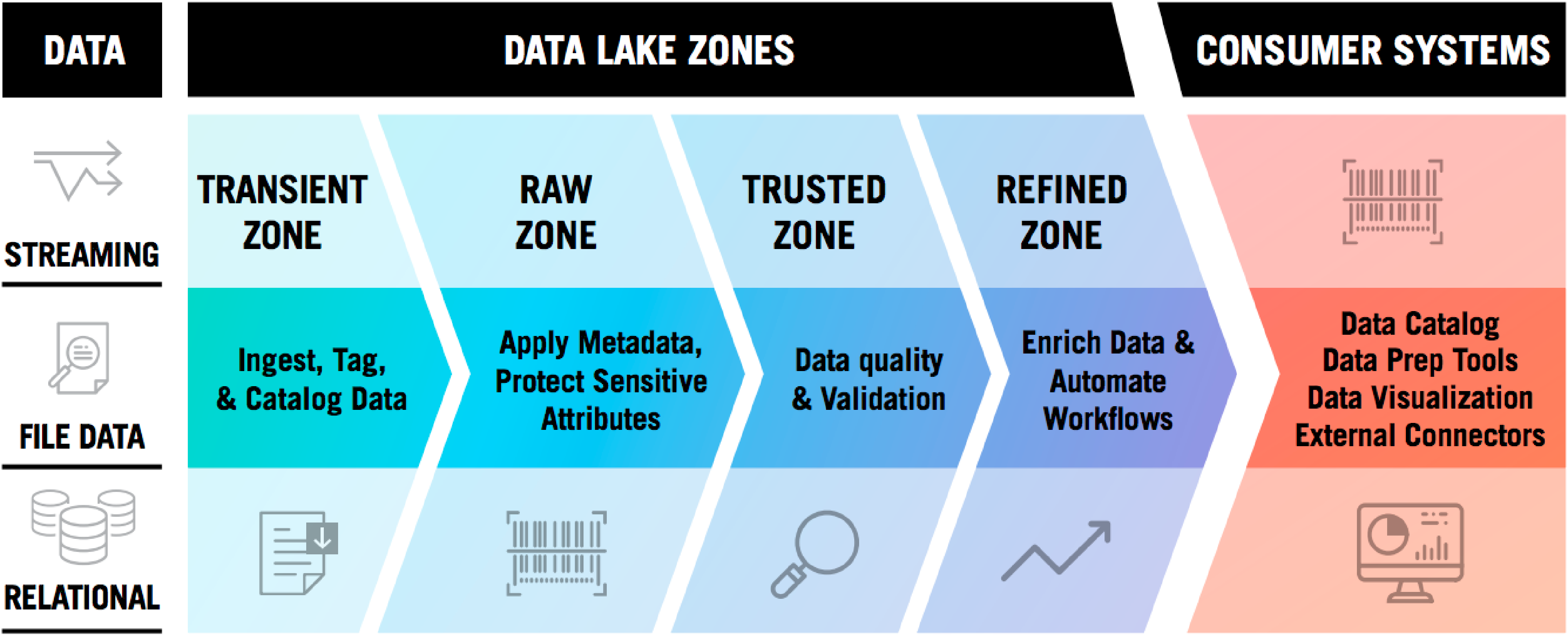
1. Raw, unorganized data
2. Emails
3. PDF files
4. Images, video and audio
5. Social media tools

2 The reservoir of water is a dataset, where you run analytics on all the data.

3 The outflow of water is the analyzed data.

4 Through this process, you are able to "sift" through all the data quickly to gain key business insights.

Data Lakes in Process



DATA

STREAMING

FILE DATA

RELATIONAL

DATA LAKE ZONES

TRANSIENT ZONE

Ingest, Tag, & Catalog Data

RAW ZONE

Apply Metadata, Protect Sensitive Attributes

TRUSTED ZONE

Data quality & Validation

REFINED ZONE

Enrich Data & Automate Workflows

CONSUMER SYSTEMS

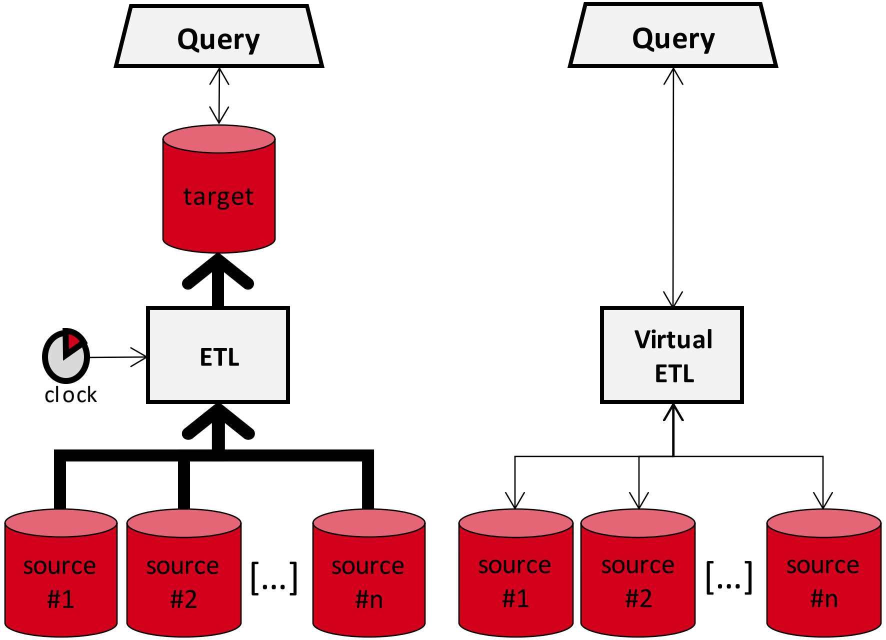
Data Catalog
Data Prep Tools
Data Visualization
External Connectors

Another issue: ORM

- Impedance Mismatch
- Object-Relational Mapping problem (and solution)
- Object Orientation (OODB)
  - Commercial failure!

Extract Transform Load (in the Big Data era)

The process that extracts data from heterogeneous data sources, transforms it in the schema that better fits the analysis to perform and loads it in the system that will perform the analysis.



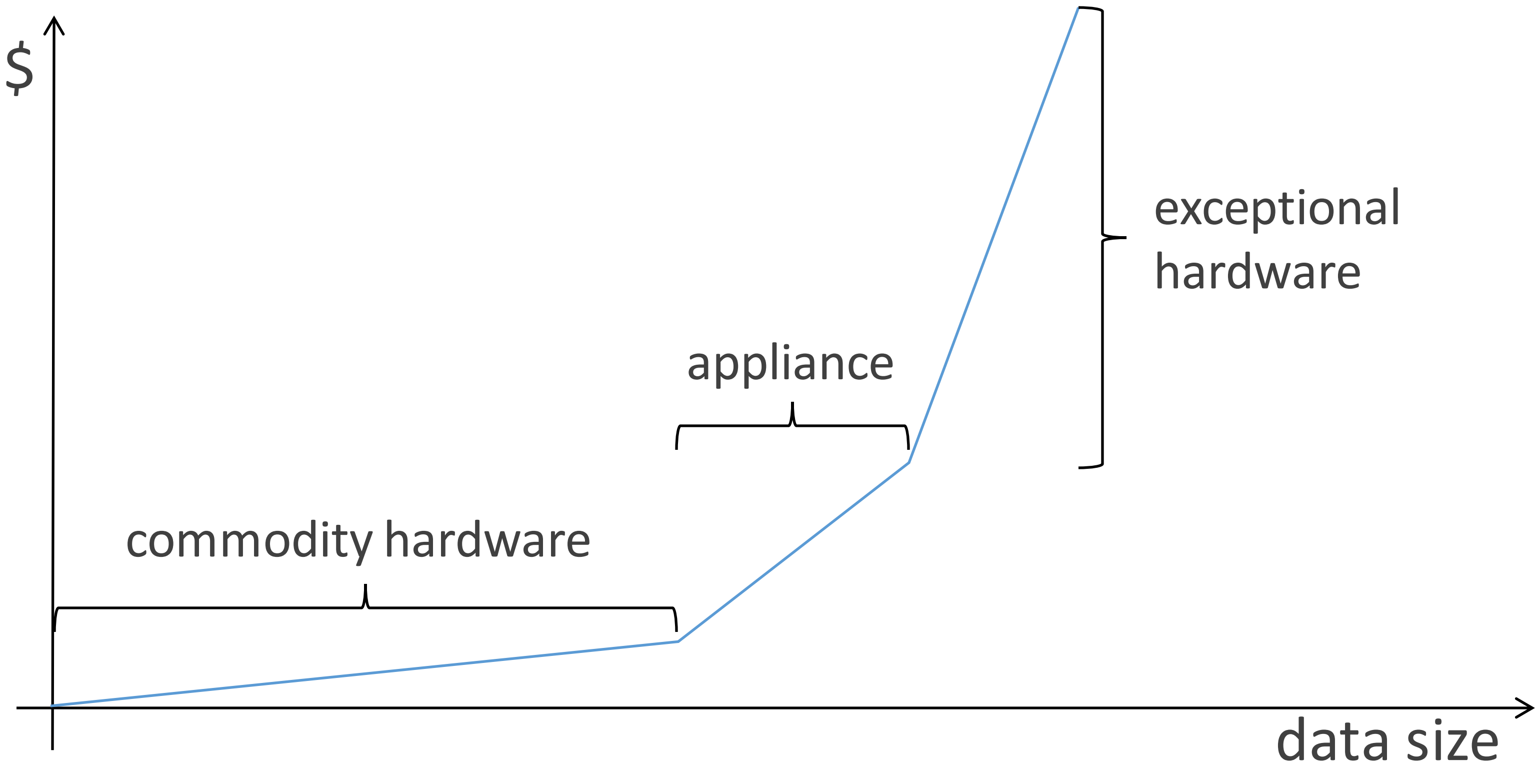
POLITECNICO
MILANO 1863

Scalability for Big Data

Vertical Vs. Horizontal Scalability

- "Traditional" SQL system scale vertically:
  - Adding data to a "traditional" SQL system may degrade its performances
  - When the machine, where the SQL system runs, no longer performs as required, the solution is to buy a better machine (with more RAM, more cores and more disk)
- Big Data solutions scale horizontally
  - Adding data to a Big Data solution may degrade its performances
  - When the machines, where the big data solution runs, no longer performs as required, the solution is to add another machine

Vertical scalability



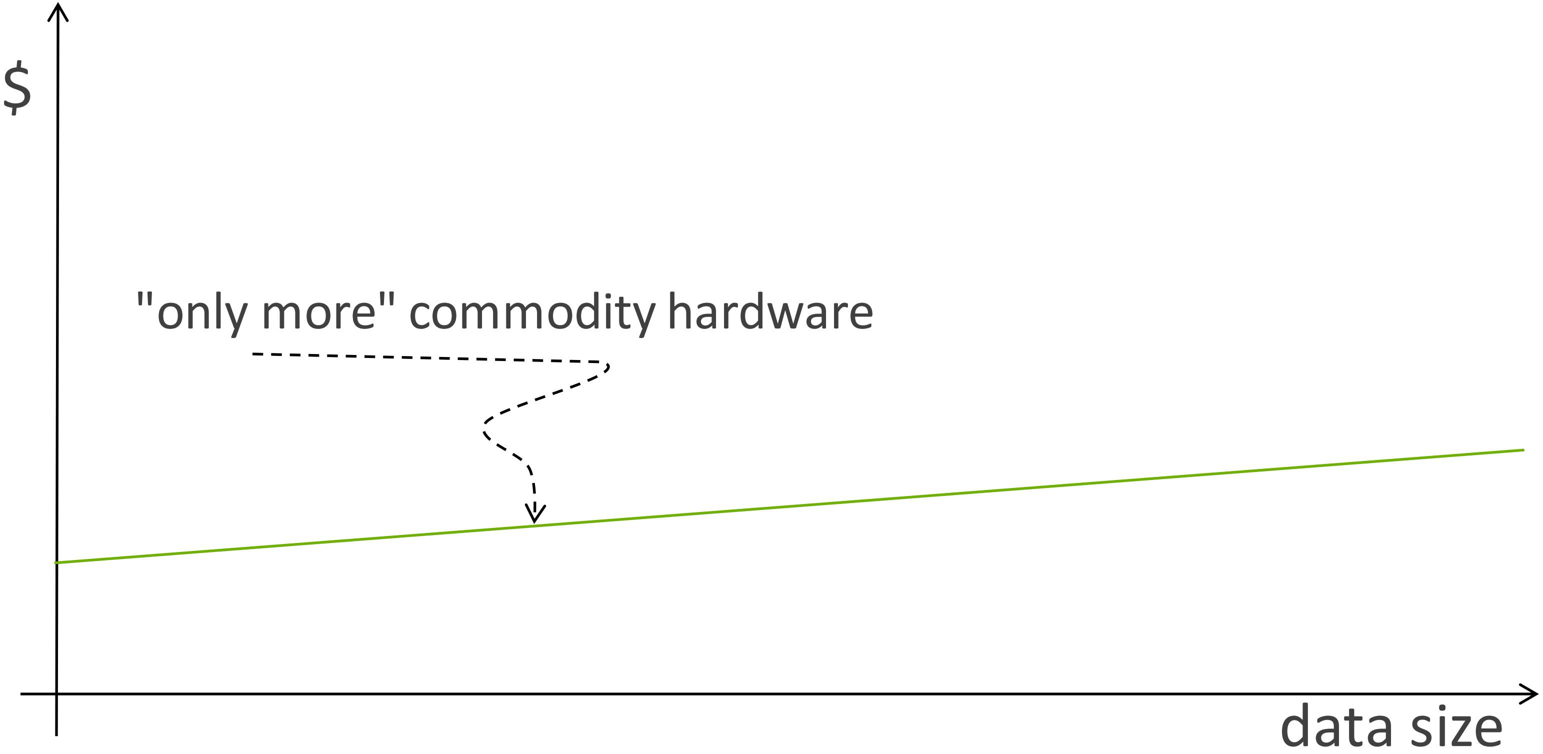
Commodity hardware

- CPU: 8-32 cores
- RAM: 16-64 GB
- Disk: 1-3 TB
- Network: 10 GE

Appliance

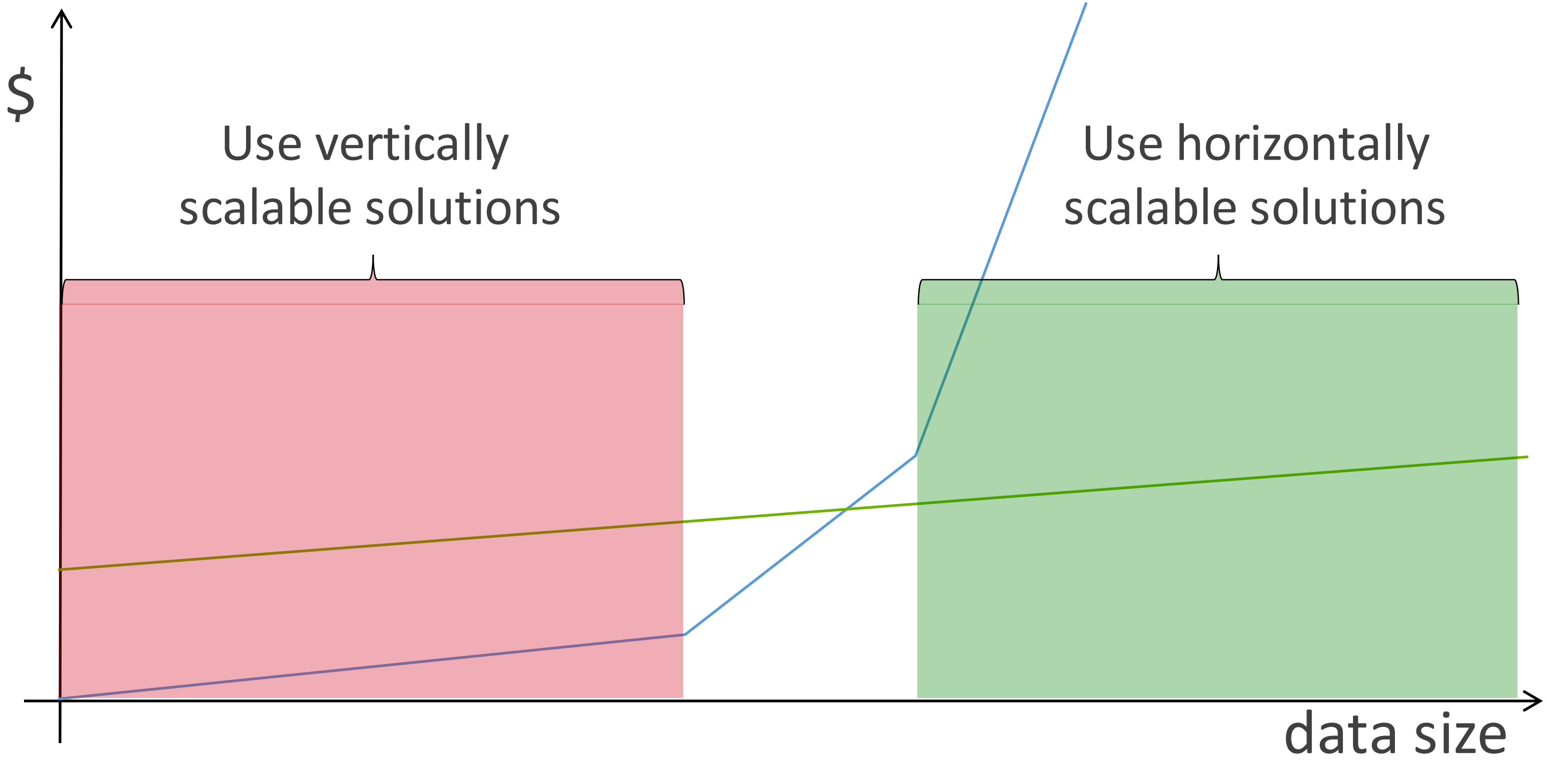
- e.g. ORACLE EXADATA DATABASE MACHINE X6-8
- CPU: 576 cores
- RAM: 24TB
- Disk: 360TB of Flash Storage per rack
- Network: 40 Gb/second InfiniBand

Horizontal scalability

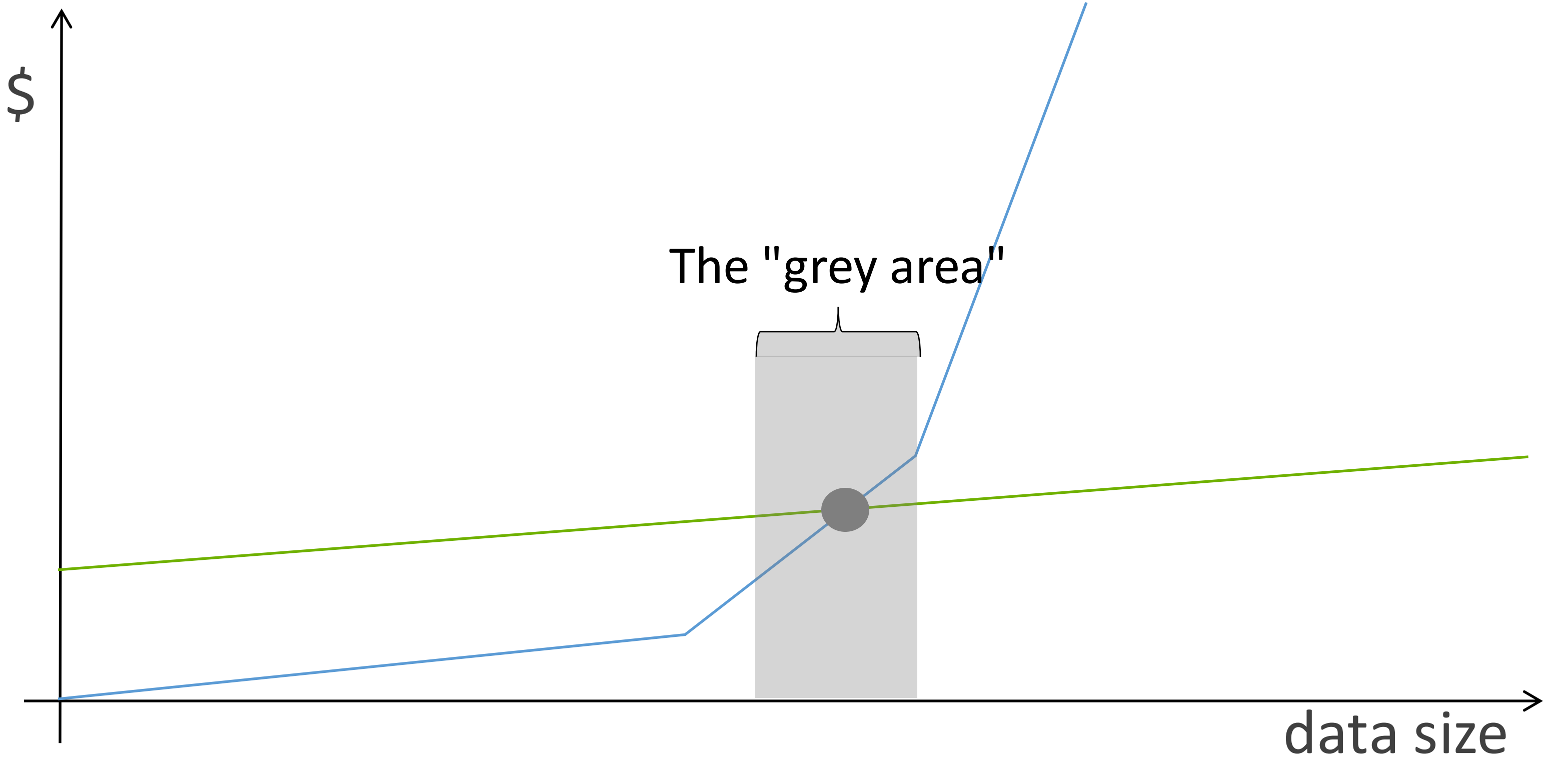


POLITECNICO MILANO 1863

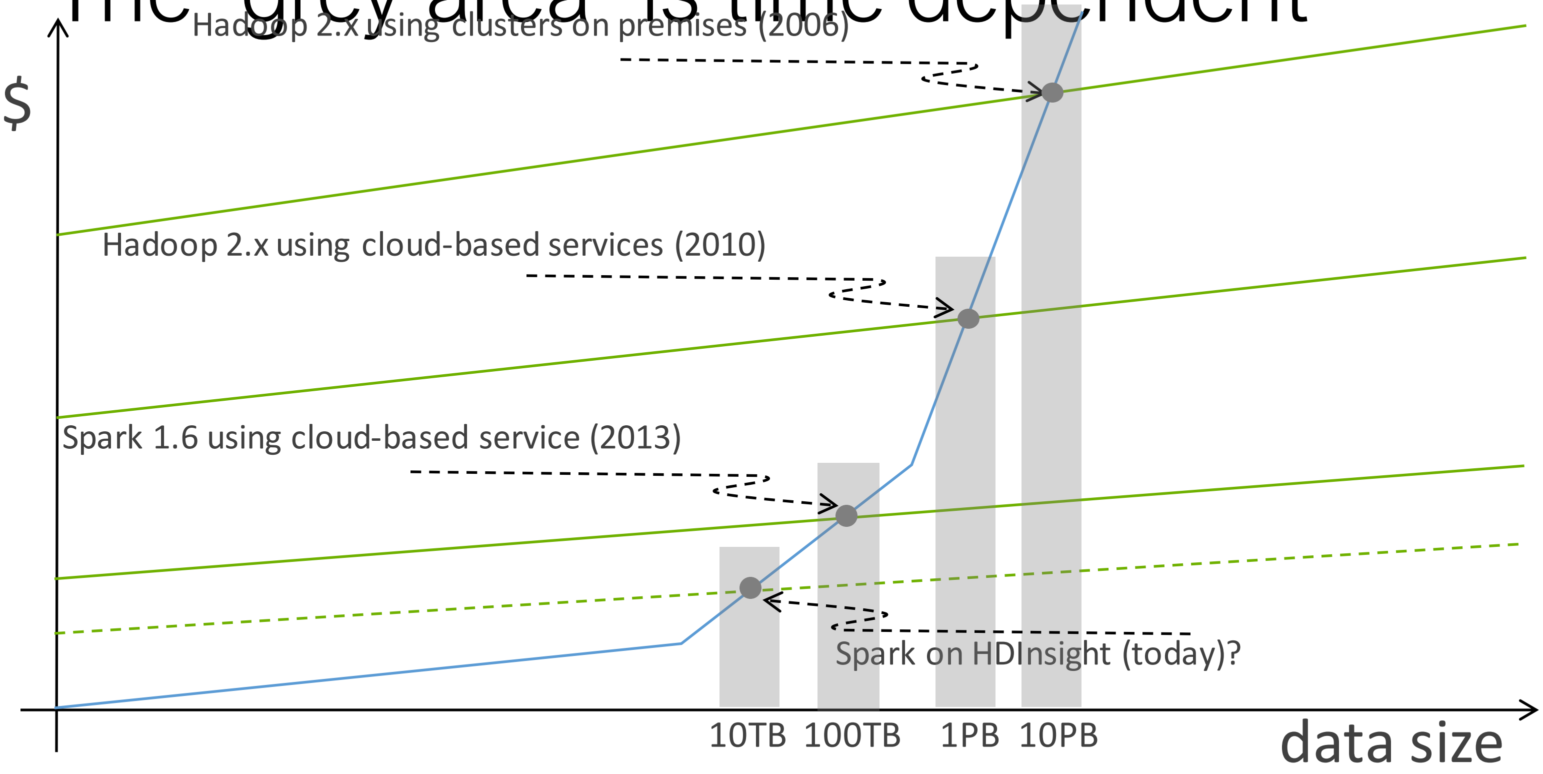
Vertical vs. Horizontal scalability



Vertical vs. Horizontal scalability



The "grey area" is time dependent



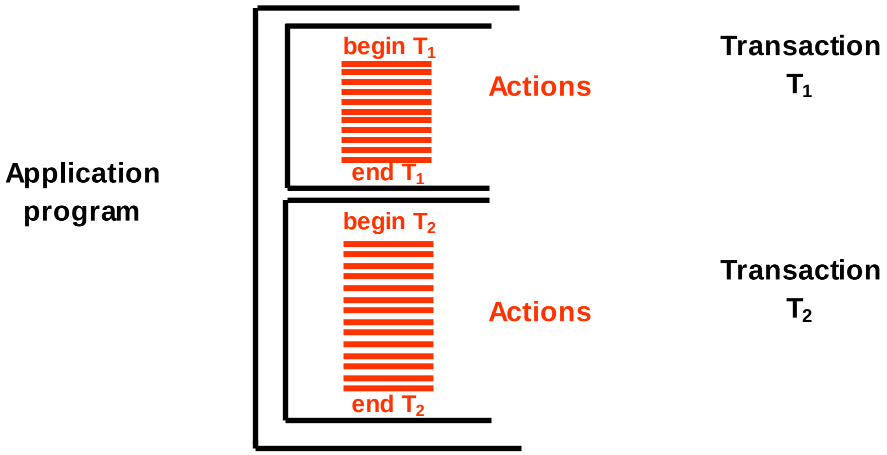
Transactional Properties

In OLTP systems

Definition of Transaction

- An elementary unit of work performed by an application
- Each transaction is encapsulated within two commands:
  - **`begin transaction`** (bot) and **`end transaction`** (eot)
- Within a transaction one of the commands below is executed (exactly once):
  - **`commit work`** (commit) and **`rollback work`** (abort)
- Transactional System (OLTP): a system capable of providing the definition and execution of transactions on behalf of multiple, concurrent applications

Application and Transaction

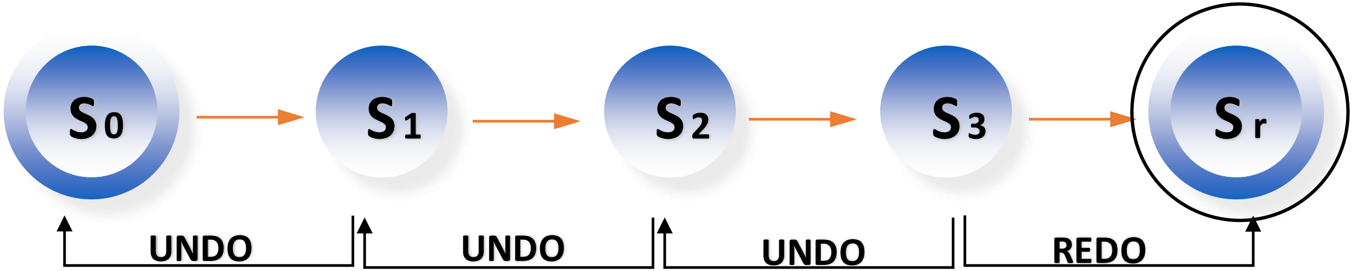


ACID Properties of Transactions

- A transaction is a unit of work enjoying the following properties:
    - Atomicity
    - Consistency
    - Isolation
    - Durability

Atomicity

- A transaction is an atomic transformation from the initial state to the final state
- Possible behaviors:
  1. Commit work: SUCCESS
  2. Rollback work or error prior to commit: UNDO
  3. Fault after commit: REDO

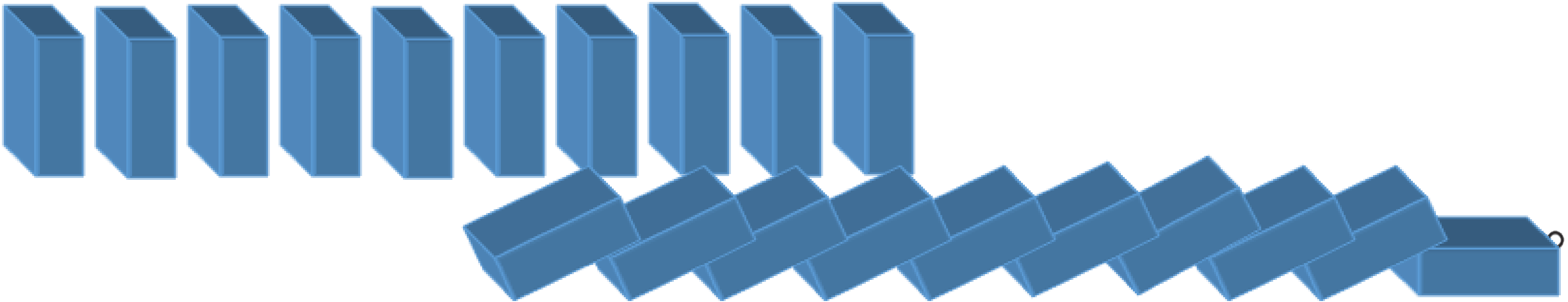


Consistency

- The transaction satisfies the integrity constraints
- Consequence:
    - If the initial state is consistent
    - Then the final state is also consistent

Isolation

- A transaction is not affected by the behavior of other, concurrent transactions
- Consequence:
  - Its intermediate states are not exposed
  - The “domino effect” is avoided



Durability

- The effect of a transaction that has successfully committed will last “forever”
  - Independently of any system fault

Transaction Properties and Mechanisms

- Atomicity
  - Abort-rollback-restart
  - Commit protocols
- Consistency
  - Integrity checking of the DBMS
- Isolation
  - Concurrency control
- Durability
  - Recovery management

CAP Theorem

POLITECNICO
MILANO 1863

CAP Theorem (Brewer's Theorem)

It is impossible for a *distributed* computer system to simultaneously provide all three of the following guarantees:

- *Consistency*: all nodes see the same data at the same time
- *Availability*: Node failures do not prevent other survivors from continuing to operate (a guarantee that every request receives a response about whether it succeeded or failed)
- *Partition tolerance*: the system continues to operate despite arbitrary partitioning due to network failures (e.g., message loss)

- A distributed system can satisfy any two of these guarantees at the same time but not all three.

CAP Theorem (Brewer's Theorem): So, what can be done?

In a distributed system, a network (of networks) in inevitable (by definition).

Failures can, and will, occur to a networked system -> **partitioned tolerance should be accommodated**.

Then, the only option left is **choosing between Consistency and Availability** - i.e., CA doesn't make any sense (expect when we have, e.g., a single-site databases; 2-phase commit, cache validation protocols))

Not necessarily in a mutually exclusive manner, but possibly by partial accommodation of both
→ trade-off analysis important

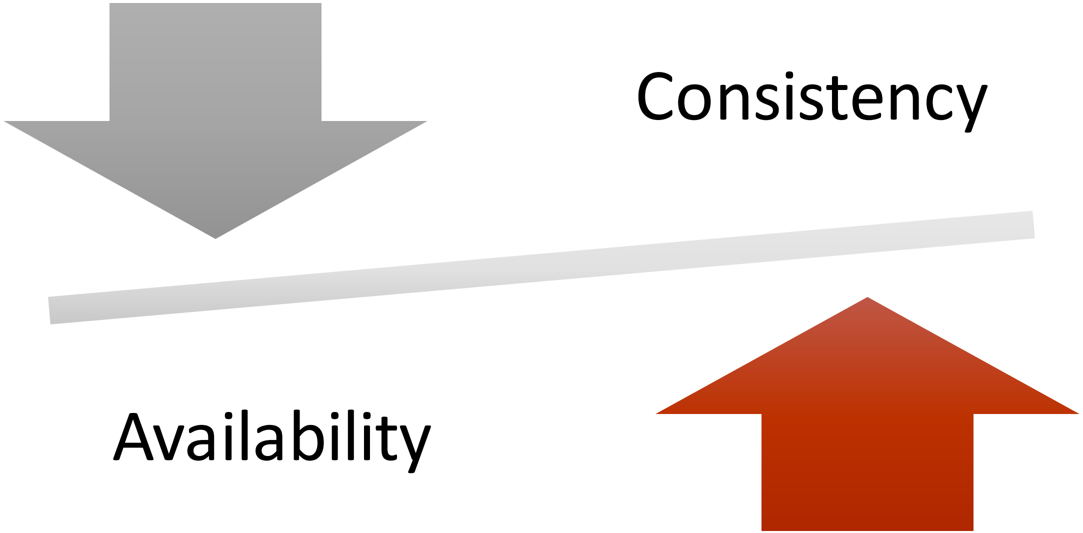
CP: A partitioned node returns

a correct value, if in a consistent state;

a timeout error or an error, otherwise

AP: A partitioned note returns the most recent version of the data, which could be stale.

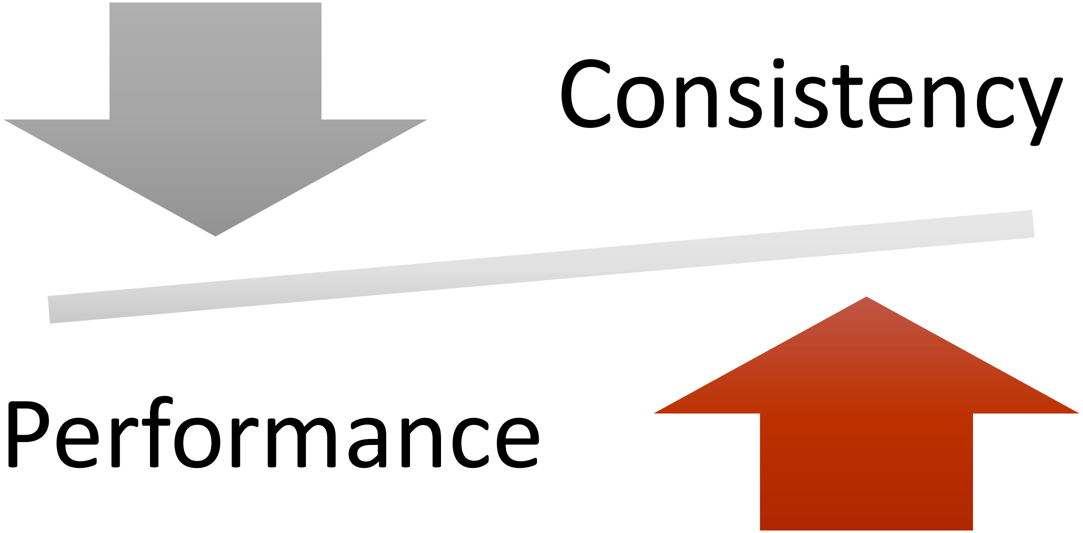
CAP: what’s really about?



Consistency

Availability

CAP: what’s really about?


Consistency

Performance

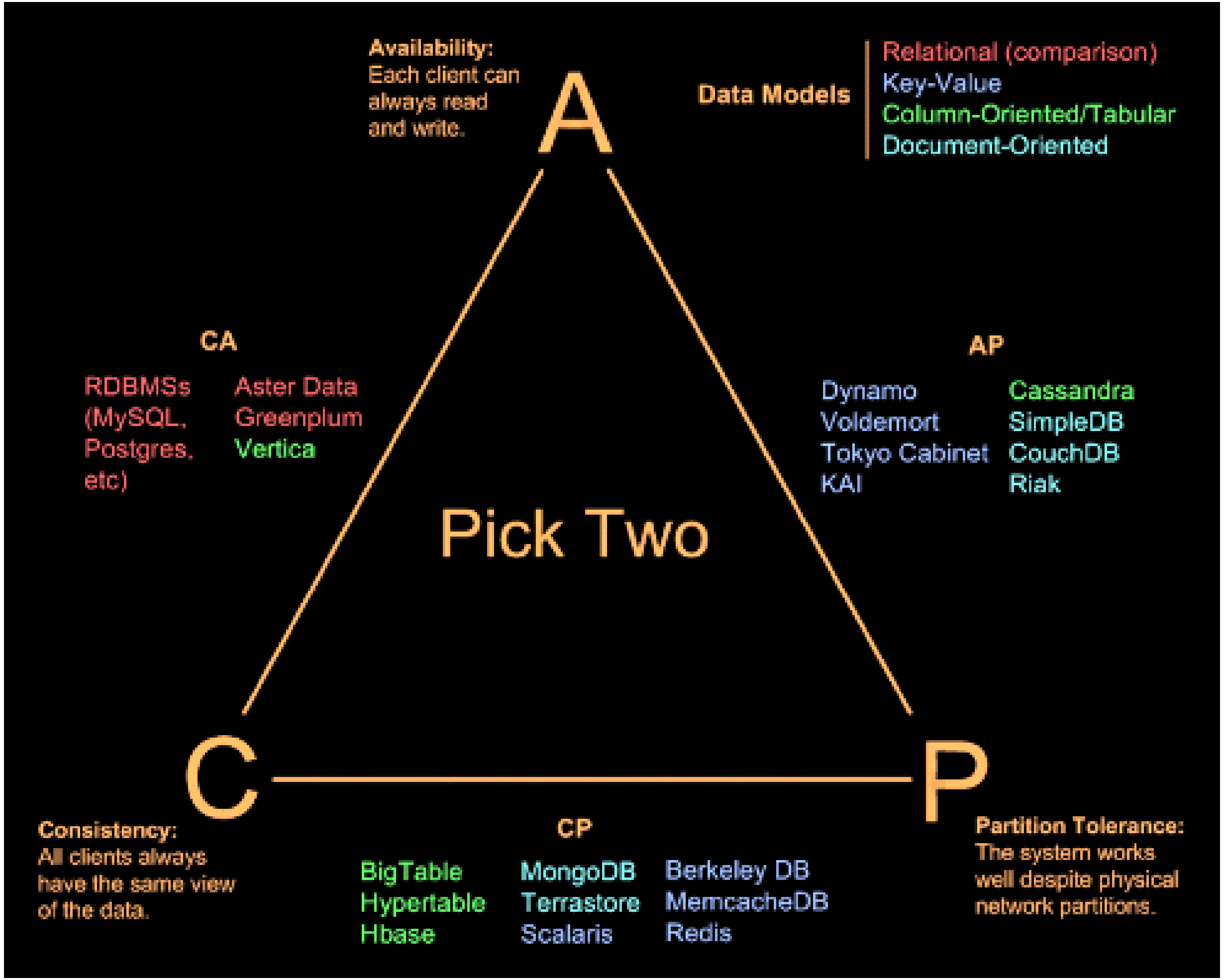
It’s a trade-off!

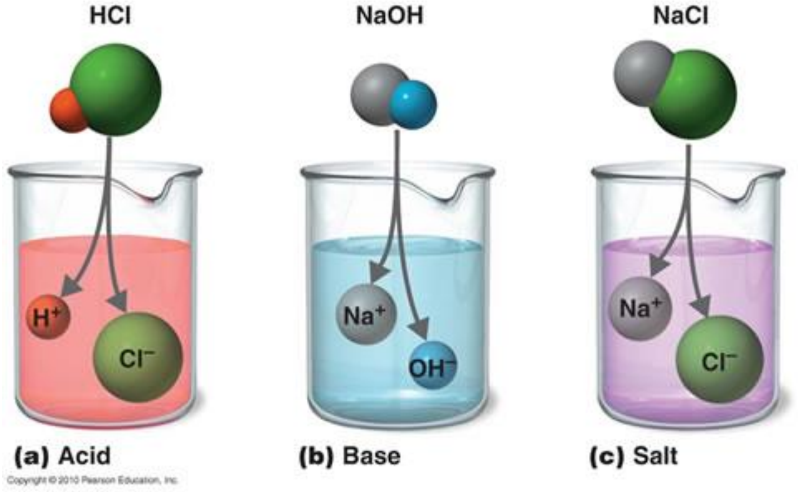
It’s a continuous space!

It’s not at DBMS level,

It’s at operation level!

Visual Guide to CAP Theorem





ACID vs. BASE properties

Recall the CAP theorem!

No transactional properties?!

- ACID properties may not hold -> no properties at all then???
- focuses on availability of data even in the presence of multiple failures
- spread data across many storage systems with a high degree of replication

ACID vs. BASE

- **Rationale:**
  - It's ok to use stale data (Accounting systems do this all the time. It's called "closing out the books.") ; it's ok to give approximate answers
  - Use resource versioning -> say what the data really is about – no more, no less.
  - The value of x is 5, at time T and date D
  - - So, shift the PH from 0-6 (acidic) to 8-14 (basic) – pure water's PH is 7 and neutral

- **Can some compromise be made between C and A?:**
  - instead of completely giving up on C, for A
  - Instead of completely giving up on A, instead of C

BASE
(Basically Available, Soft-State, Eventually Consistent)

- Basic Availability: fulfill request, even in partial consistency.
- Soft State: abandon the consistency requirements of the ACID model pretty much completely
- Eventual Consistency: at some point in the future, data will converge to a consistent state; delayed consistency, as opposed to immediate consistency of the ACID properties.
    - purely a liveness guarantee (reads eventually return the requested value); but
    - does not make safety guarantees, i.e.,
    - an eventually consistent system can return any value before it converges

ACID vs. BASE trade-off

- No general answer to whether your application needs an ACID versus BASE consistency model.

- Given **BASE**'s loose consistency, developers need to be more **knowledgeable and rigorous about consistent data** if they choose a BASE store for their application.
- Planning around BASE limitations can sometimes be a major disadvantage when compared to the simplicity of ACID transactions.
- A fully ACID database is the perfect fit for use cases where data reliability and consistency are essential.

Why didn’t we change Relational?

- Consolidated
- Integration platform
- Granting ACID properties!

The NOSQL World

The path to the Big Data Era

Good Quote!

- *Google, Amazon, Facebook, and DARPA all recognized that* ***when you scale systems large enough, you can never put enough iron in one place to get the job done*** *(and you wouldn't want to, to prevent a single point of failure).*
- *Once you accept that you* ***have a distributed system, you need to give up consistency or availability****, which the fundamental transactionality of traditional RDBMSs cannot abide.*

- - Cedric Beust

NoSQL History

- MultiValue databases at TRW in 1965.
- DBM is released by AT&T in 1979.
- Lotus Domino released in 1989.
- Carlo Strozzi used the term NoSQL in 1998 to name his lightweight, open-source relational database that did not expose the standard SQL interface.
- Graph database Neo4j is started in 2000.
- Google BigTable is started in 2004. Paper published in 2006.
- CouchDB is started in 2005.

NoSQL History

- The research paper on Amazon Dynamo is released in 2007.
- The document database MongoDB is started in 2007 as a part of a open
- source cloud computing stack and first standalone release in 2009.
- Facebooks open sources the Cassandra project in 2008.
- Project Voldemort started in 2008.
- The term NoSQL was reintroduced in early 2009.

The #NOSQL story

- A meetup
  - Johan Oskarsson in SF, CA, June 2009
- A hashtag: #nosql

Kinds of NoSQL

- NoSQL solutions fall into two major areas:
  - Key/Value or ‘the big hash table’ (remember caching?)
    - Amazon S3 (Dynamo)
    - Voldemort
    - Scalaris
    - MemcacheDB,
    - Azure Table Storage,
    - Redis,
    - Riak
  - Schema-less
    - Cassandra (column-based)
    - CouchDB (document-based)
    - Neo4J (graph-based)
    - HBase (column-based)

Different Types of NoSQL

- Key-Value Store
  - A key that refers to a payload (actual content / data)
  - MemcacheDB, Azure Table Storage, Redis

- Column Store
  - Column data is saved together, as opposed to row data
  - Super useful for data analytics
  - Hadoop, Cassandra, Hypertable

Different Types of NoSQL

- Document / XML / Object Store
  - Key (and possibly other indexes) point at a serialized object
  - DB can operate against values in document
  - MongoDB, CouchDB, RavenDB

- Graph Store
  - Nodes are stored independently, and the relationship between nodes (edges) are stored with data
  - Neo4j

NoSQL originators

- Google (BigTable, LevelDB)
- LinkedIn (Voldemort)
- Facebook (Cassandra)
- Twitter (Hadoop/Hbase, FlockDB, Cassandra)
- Netflix (SimpleDB, Hadoop/HBase, Cassandra)
- CERN (CouchDB)

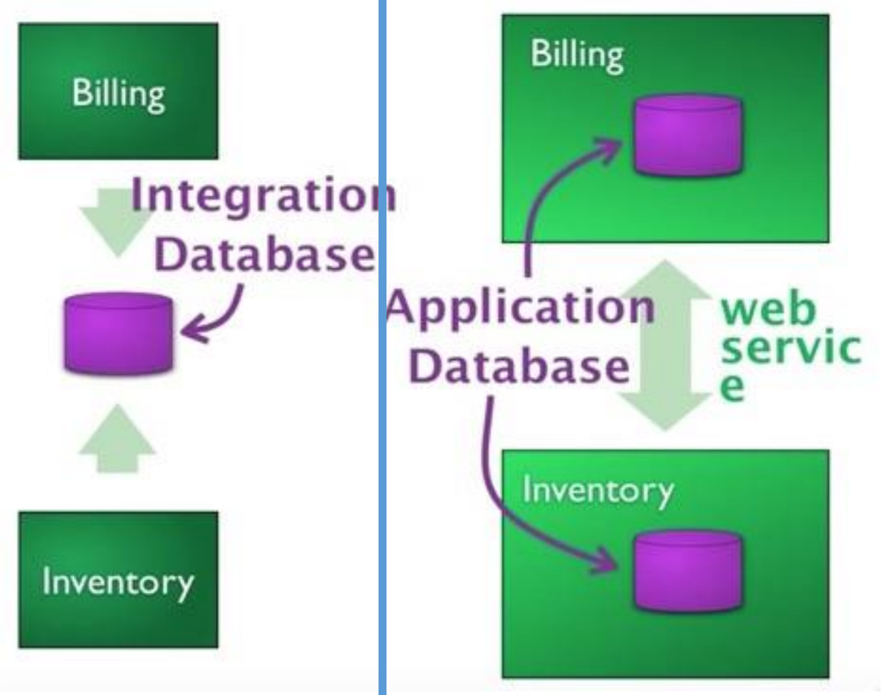
Transactional properties?!

- SQL databases:
  - Structured query language for
  - Traditional relational databases (unique keys, single valued, no update/insertion/deletion anomalies)
  - Well structured data
  - ACID properties should hold

- NoSQL (Not only SQL) databases:
  - triggered by the storage needs of Web 2.0 companies such as Facebook, Google and Amazon.com
  - Not necessarily well structured – e.g., pictures, documents, web page description, video clips, etc.
  - Lately of increasing importance due to big data
  - BASE properties may be enough

Integration: from data to services

SQL world



NO-SQL world

POLITECNICO
MILANO 1863

SYSTEMS AND METHODS FOR BIG AND UNSTRUCTURED DATA

Towards NoSQL

Marco Brambilla

marco.brambilla@polimi.it

@marcobrambi