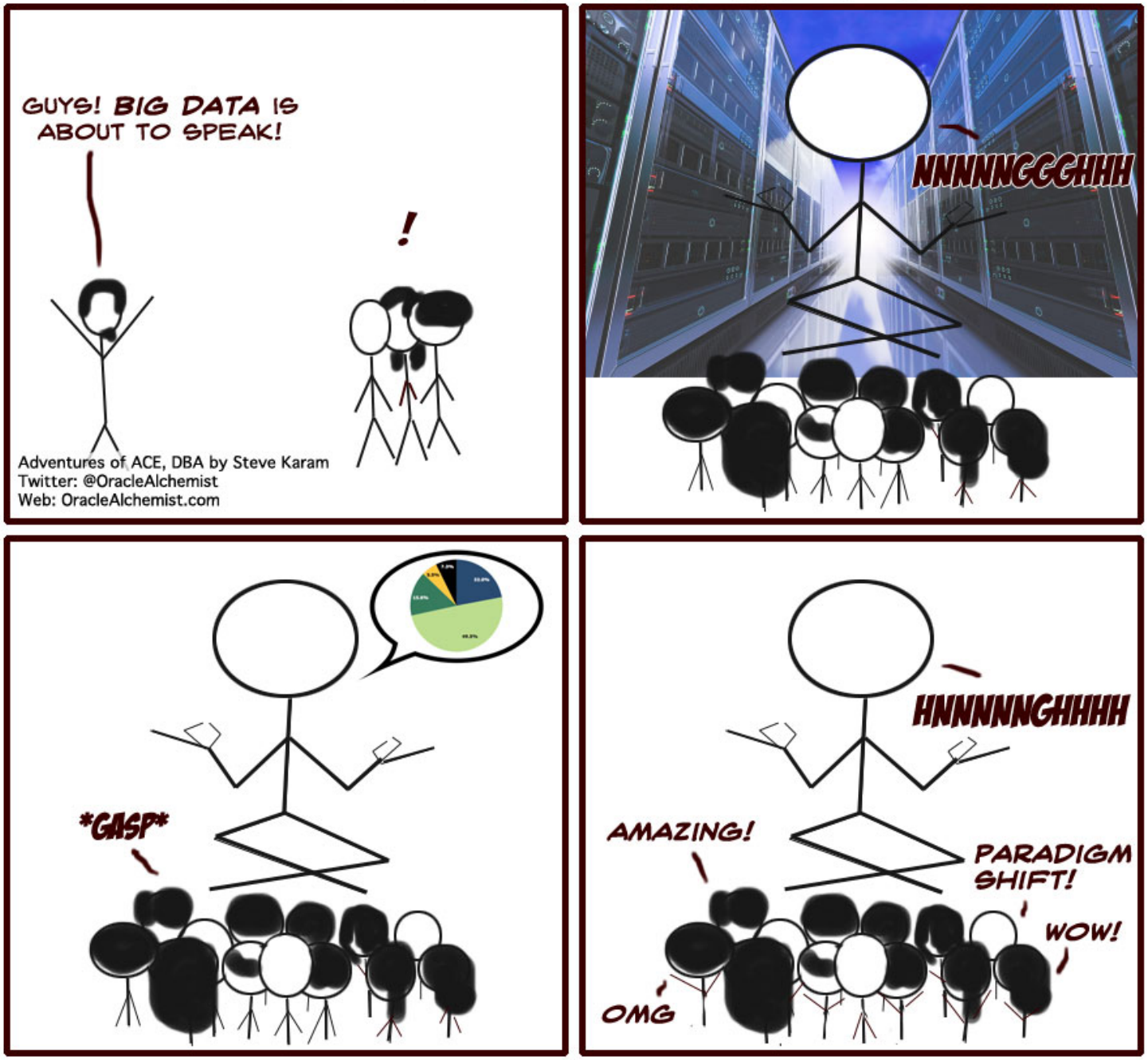
BIG DATA SPEAKS



CSE 487/587
Data Intensive Computing

Lecture 12: noSQL & BigData Part One

Vipin Chaudhary

vipin@buffalo.edu

716.645.4740
305 Davis Hall

Overview of Today's Lecture

- Workshops of Interest

  (http://cdse.buffalo.edu/cdse-days/workshops)

  - Accumulo (March 24, 10AM-1PM, 113A Davis Hall)

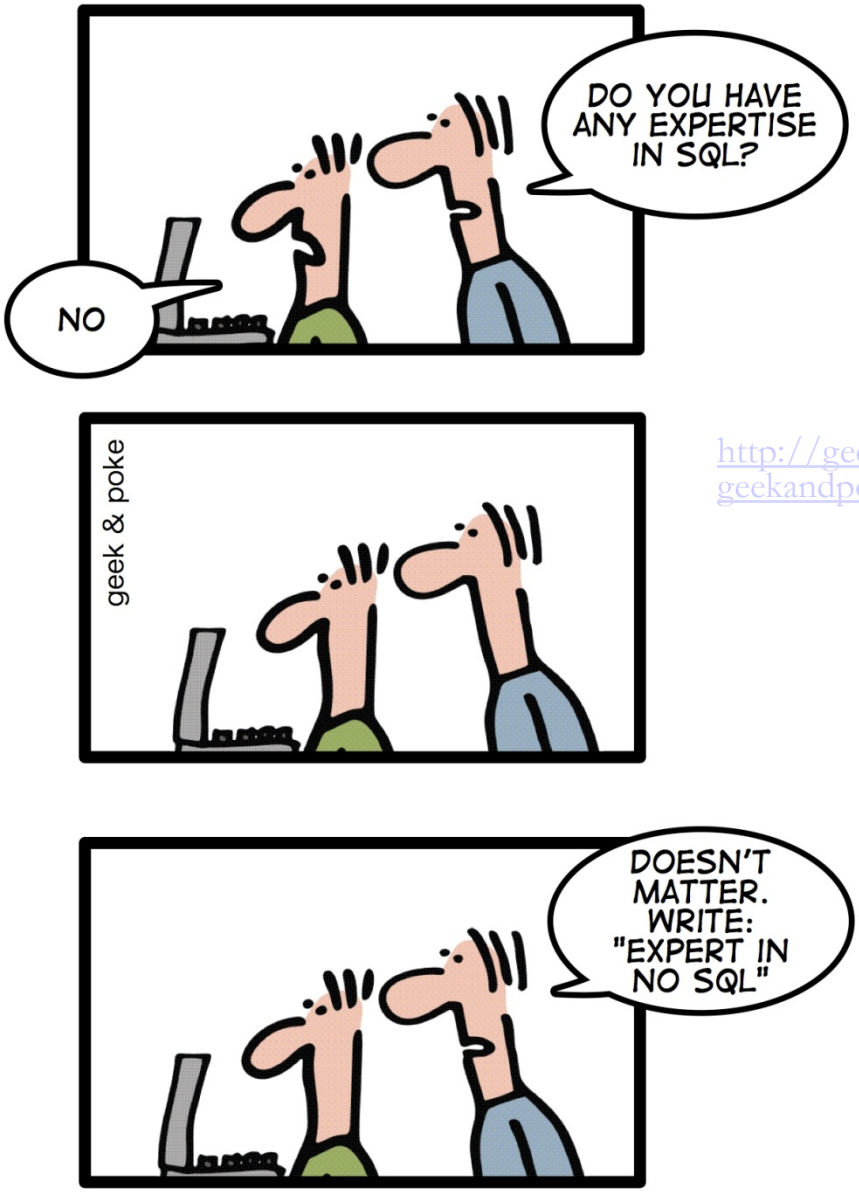
    By Dennis Patrone (Johns Hopkins/APL)

  - R Programming (March 26, 12-3PM, 120 Clemens Hall)

    By D. Gaile, R. Hageman Blair, and J. Miecznikowski (UB)

- noSQL Fundamentals

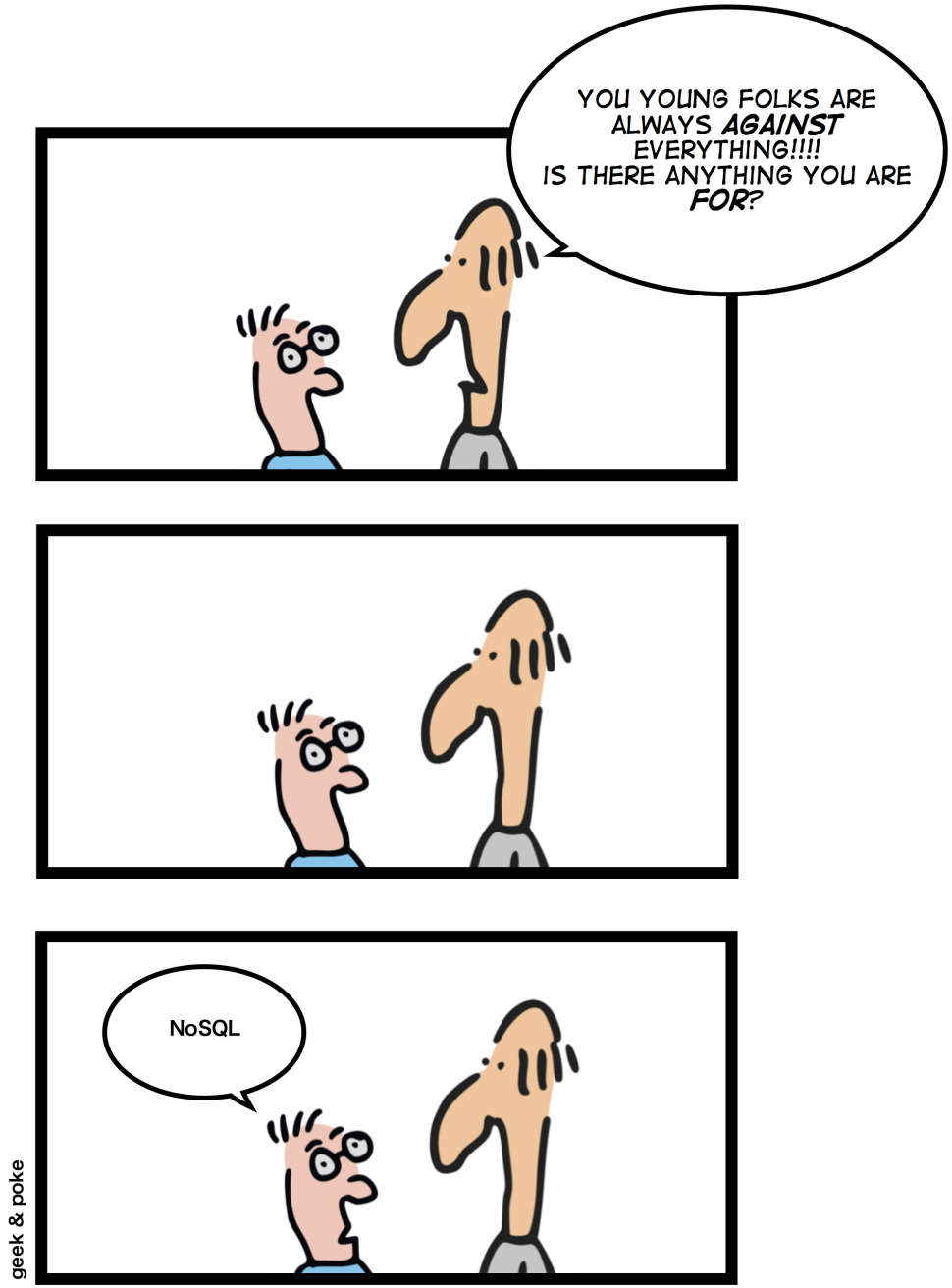
HOW TO WRITE A CV



http://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html

Leverage the NoSQL boom

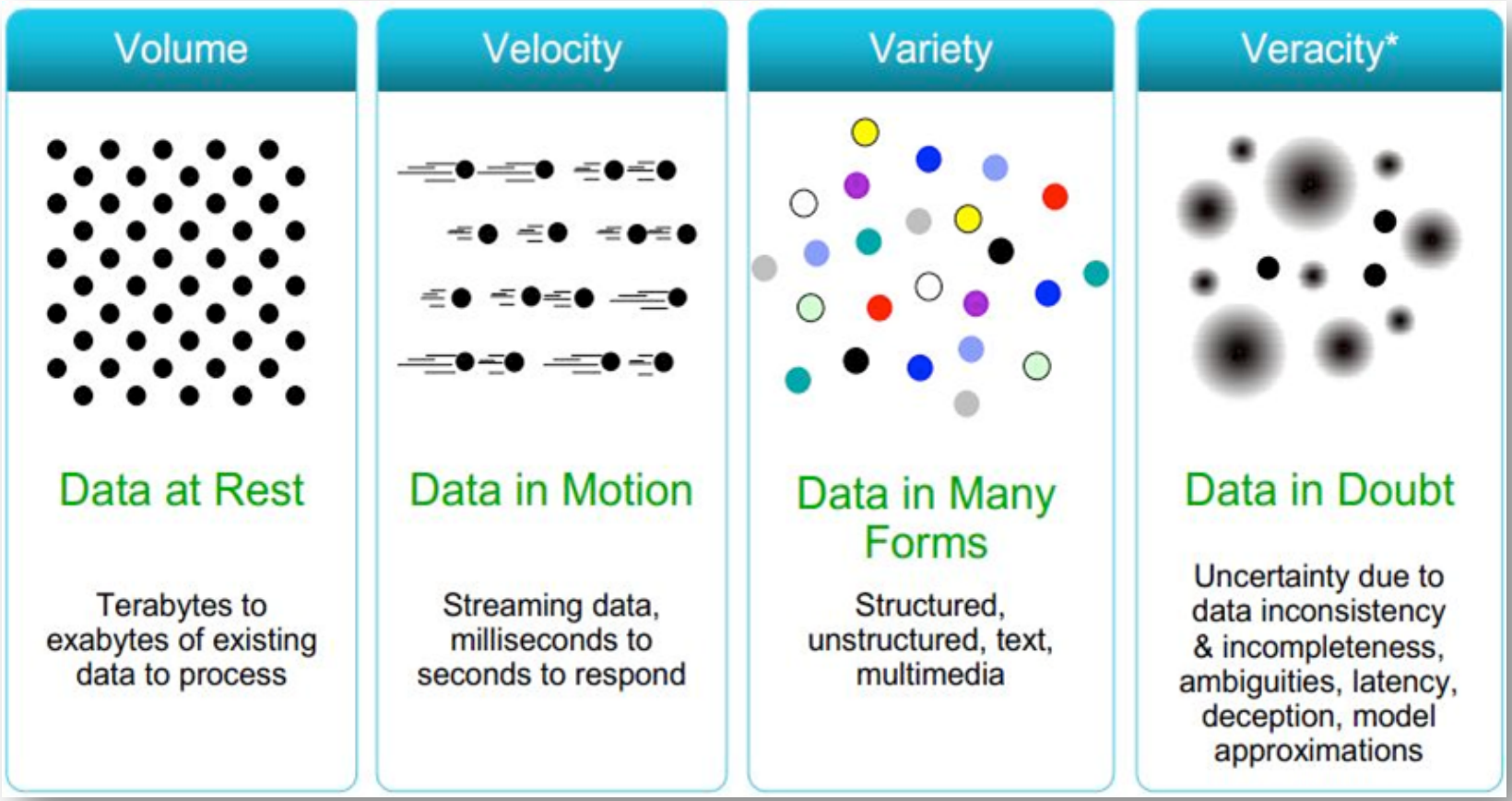
http://www.nosql-database.org/
Currently 150 databases



What is Big Data?

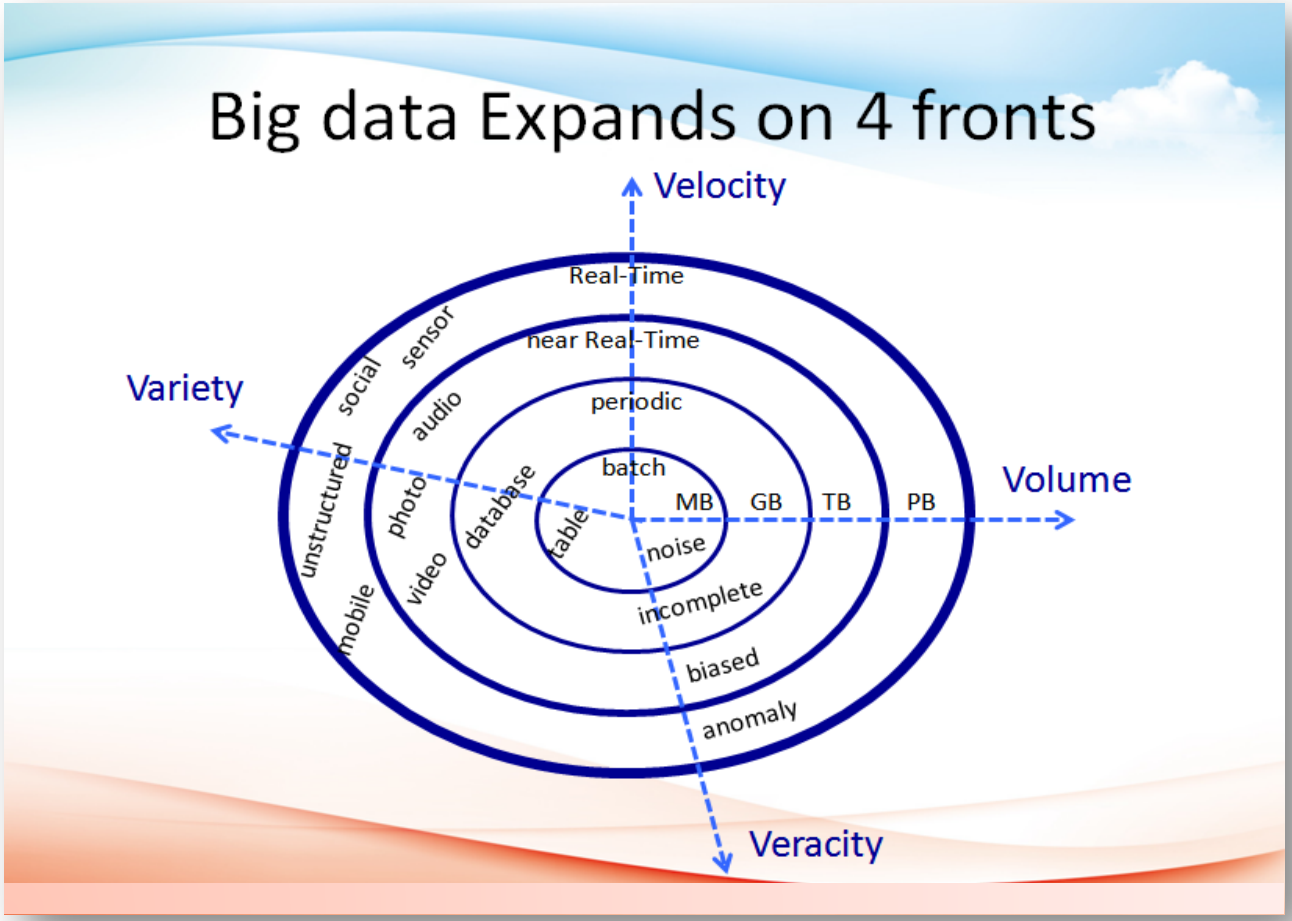
- Large volumes of high velocity, complex, and variable data that require advanced techniques and technologies to enable the capture, storage, distribution, management, and analysis of the information (Source: U.S. Congress Report, Aug 2012: [*])

Big Data Characteristics



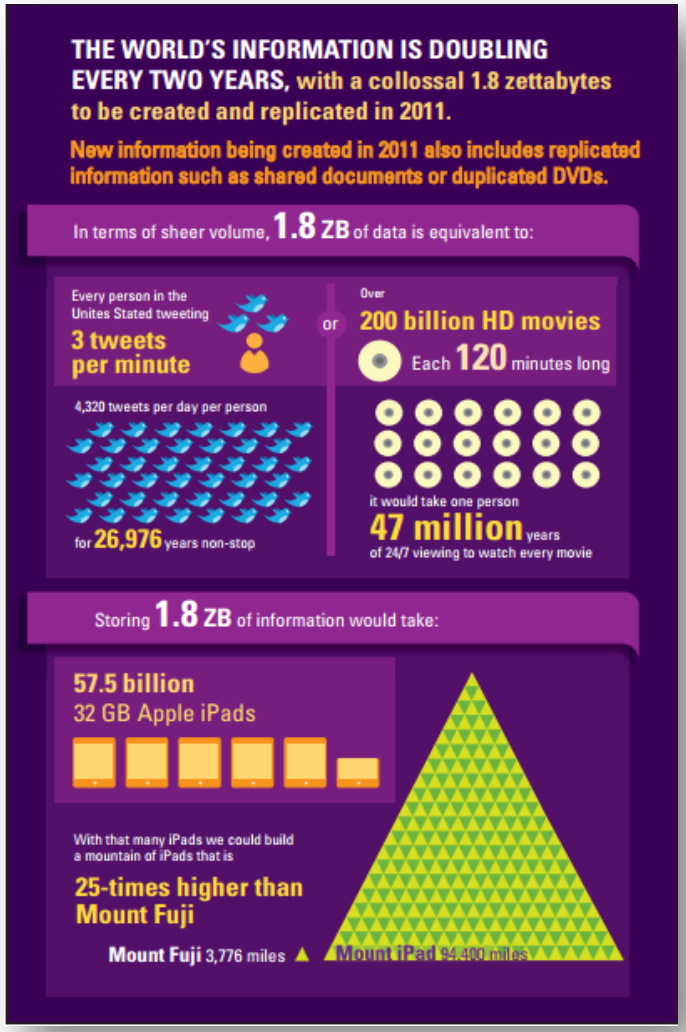
Source: http://www.datasciencecentral.com/profiles/blogs/data-veracity

Big Data Characteristics cont…



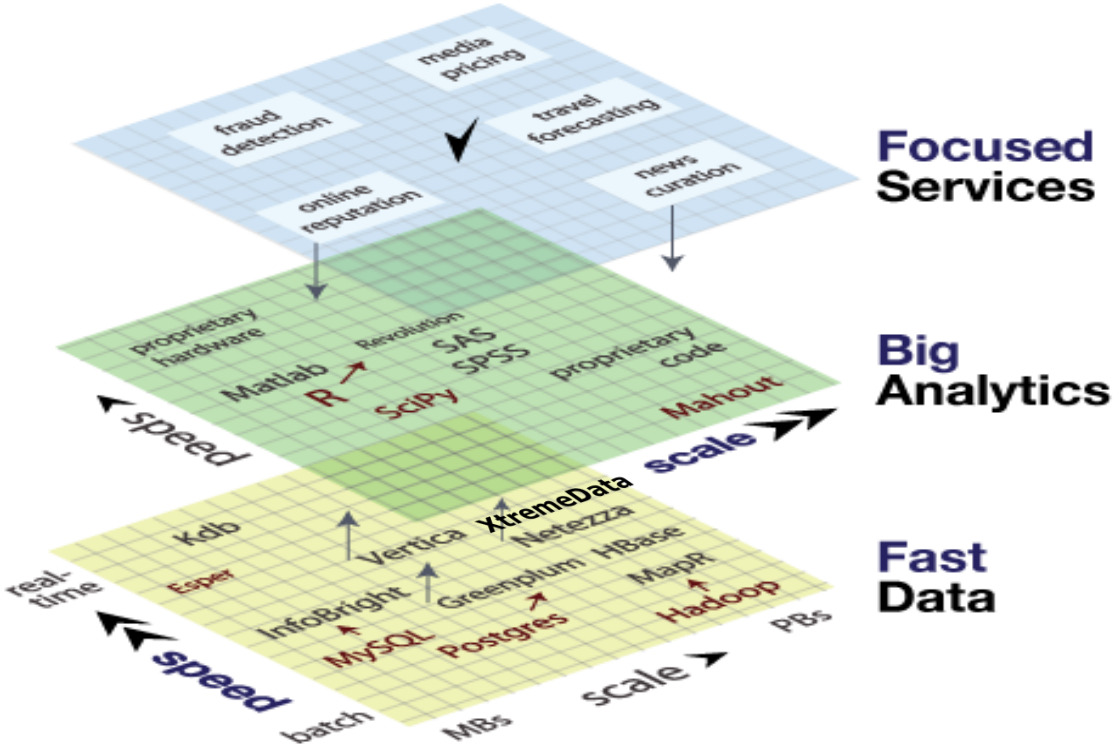
Source: http://whatis.techtarget.com/definition/3Vs

What does Big Data look like ?



Source: "Extracting Value from Chaos," IDC Universe study, 2011; http://mashable.com/2011/06/28/data-infographic/

Emerging Big Data Stack



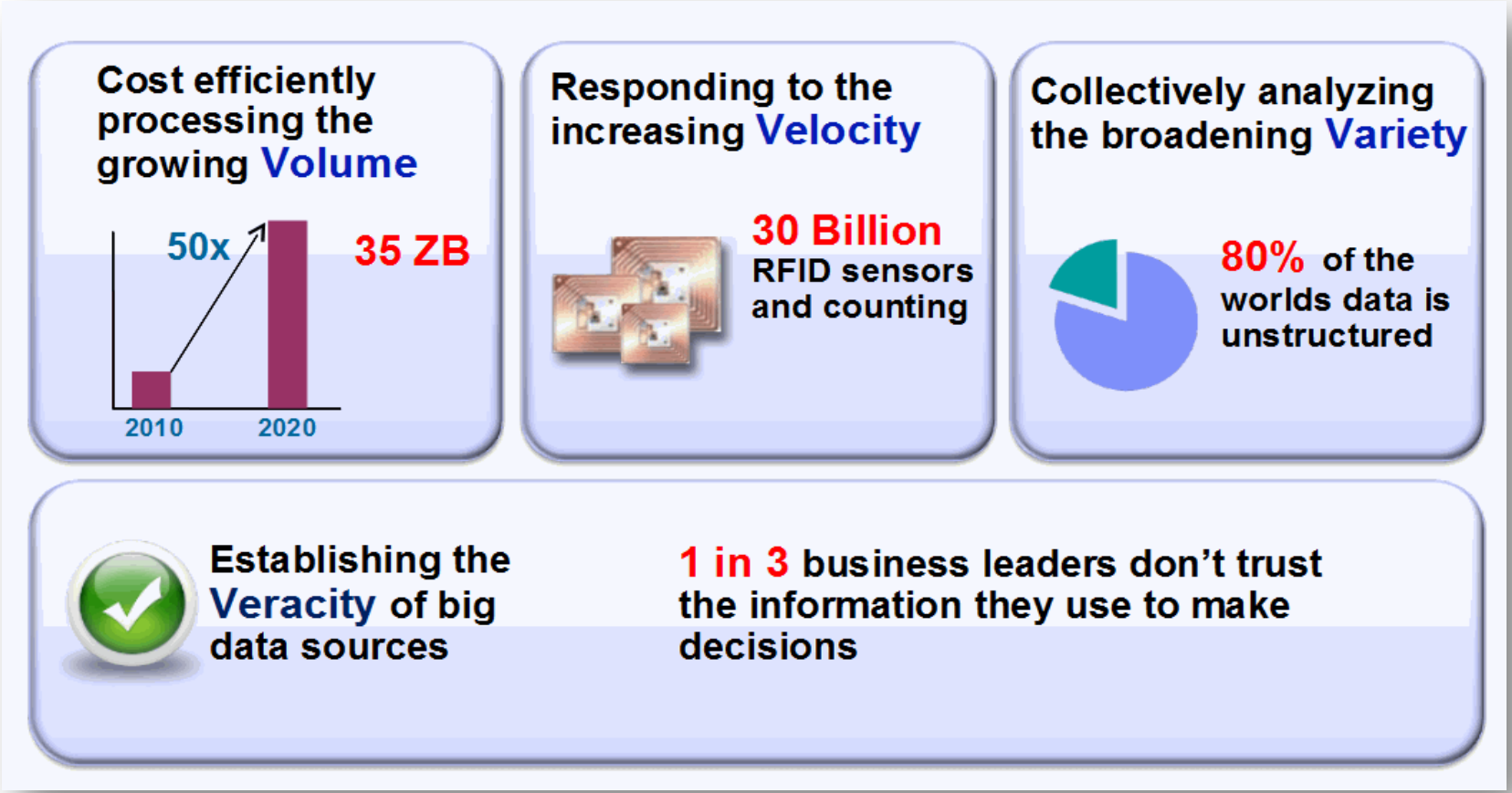
Source: The Emerging Data Stack - Rose Business Technologies: www.rosebt.com

How is Big Data Different?

Traditional Data	BIG Data
Megabytes to Gigabytes or Terabytes	Terabytes or Petabytes to Exabytes
Centralized	Distributed
Structured	Structured, Semi-Structured, and Unstructured
Stable data model	Flat schemas
Known complex interrelationships	Known and Unknown complex interrelationships

Source: http://www.slideshare.net/DataStax/top-5-considerations-for-a-big-data-solution#btnPrevious

Must Haves for Big Data



Source: https://www.ibm.com/developerworks/community/blogs/5things/resource/BLOGS_UPLOADED_IMAGES/Figure1-2.gif

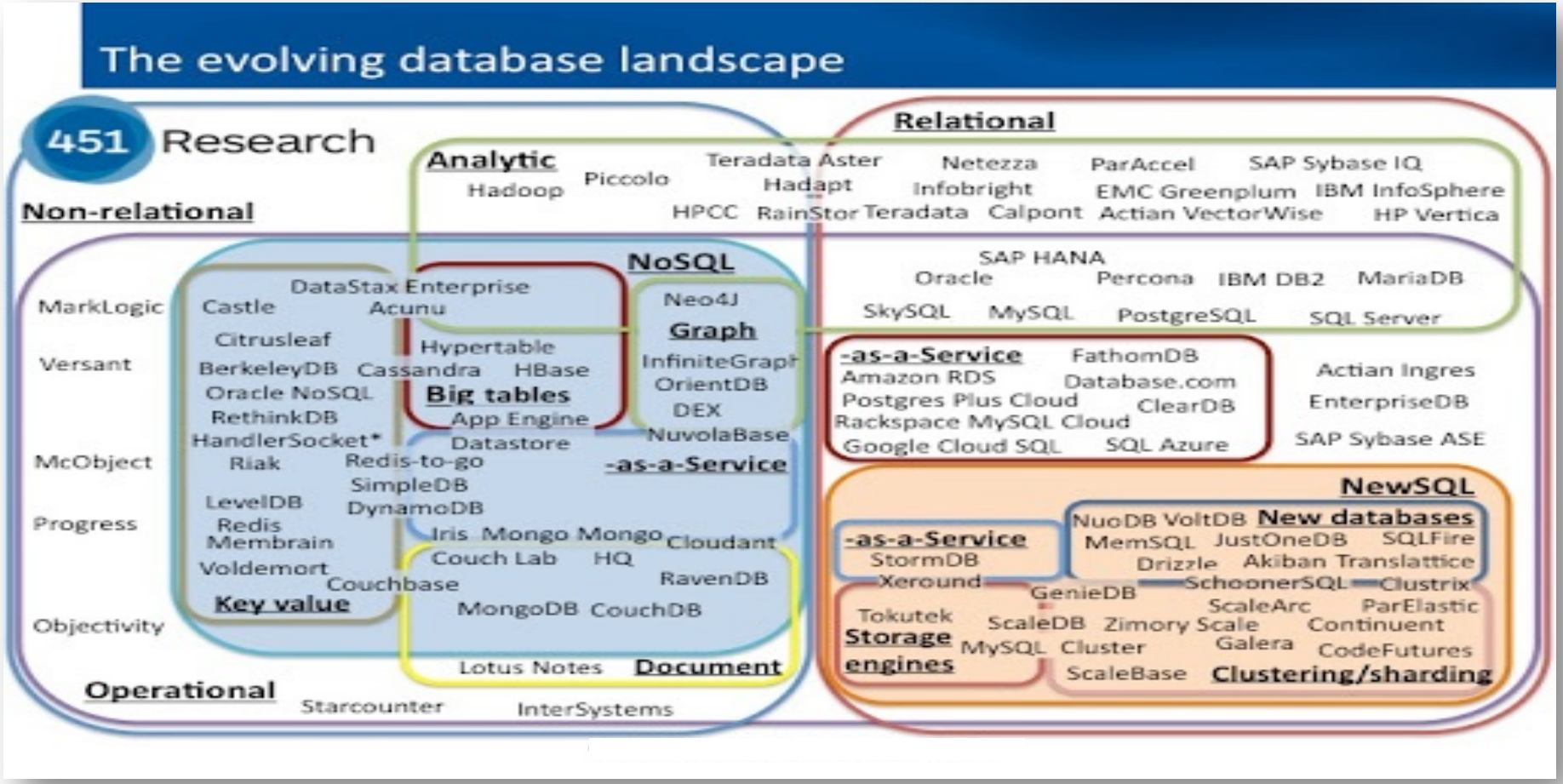
Traditional RDBMS for Big Data

- Inability to efficiently handle semi-structured and unstructured / non-tabular data
- Difficult to scale-out to meet big data needs (TB-PB range data). Clustering beyond few servers remains hard
- Challenging to parallelize to accommodate commodity hardware clusters
- Slow physical disc access speeds, not at par with processor and network speed improvements
- Does not support data integration/aggregation from heterogeneous, unrelated sources

Big Data Solution(s)

- Must support:
  - Fast incoming data
    - Streaming (mobile, financial, web, media), social interactions, patient care, sensors
  - Heterogeneous data
    - Structured, semi-structured, and unstructured
    - Different formats, real-time, search data
  - Voluminous data
    - PB - XBs
  - Inconsistent data
    - Incomplete, anomalous, biased, noisy data

Big Data Solutions - *Where do we stand ?*



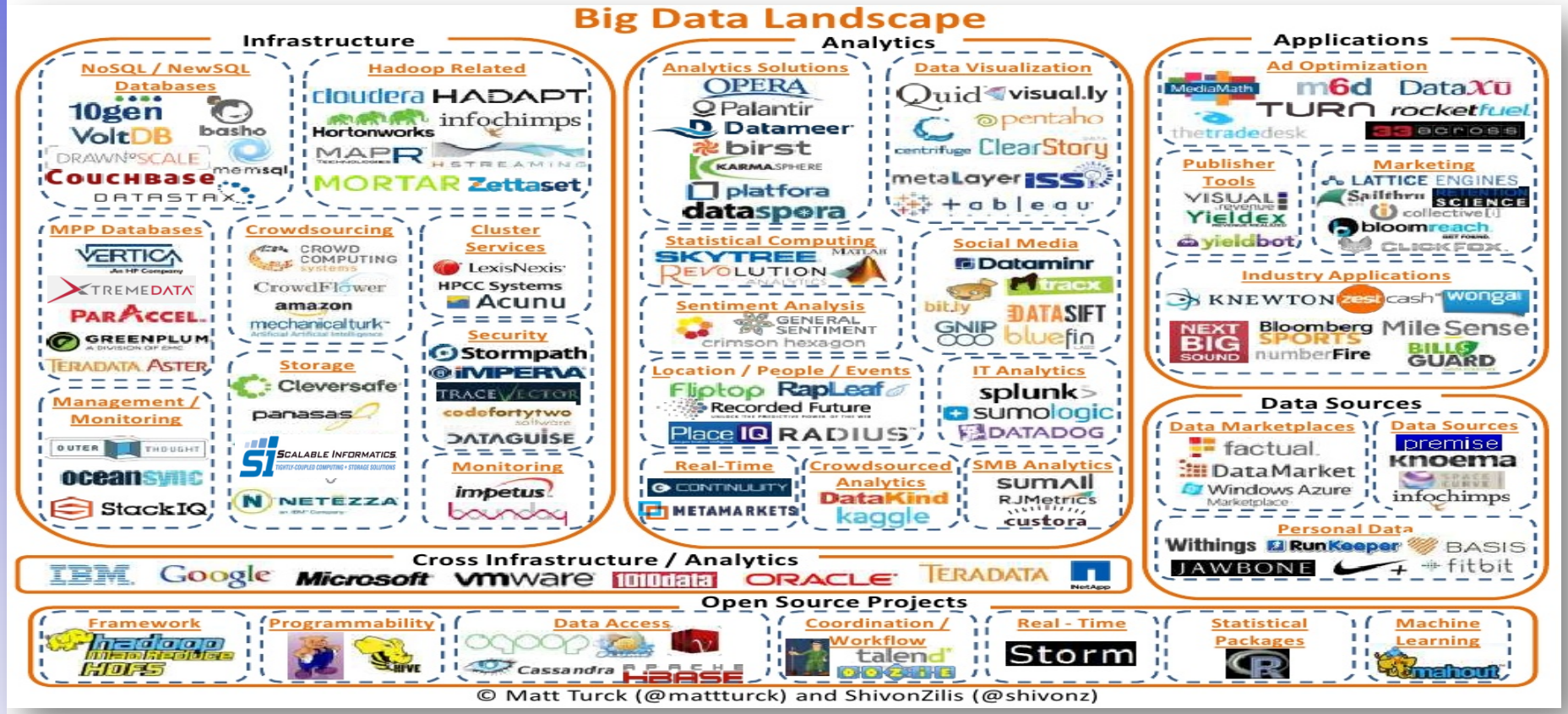
Source: http://blogs.the451group.com/information_management/2012/11/02/updated-database-landscape-graphic/

Big Data Solution(s)

- Many players at different levels/stages
  - Infrastructure
    - Servers, dedicated appliances, network, storage …
      - PC → Shared Everything/Shared Disk → Shared Nothing
  - Data Management
    - Data cleaning, integration, aggregation tools, …
      - Custom vs. Open Source vs. Commercial libraries/solutions
  - Data Analytics
    - Predictive Analytics, Deep Analytics, …
      - Custom vs. Open Source vs. Commercial libraries/solutions
  - Decision Support
    - Visualization, evaluation, collaboration tools, …

Big Data Solution(s) : *A Closer Look*

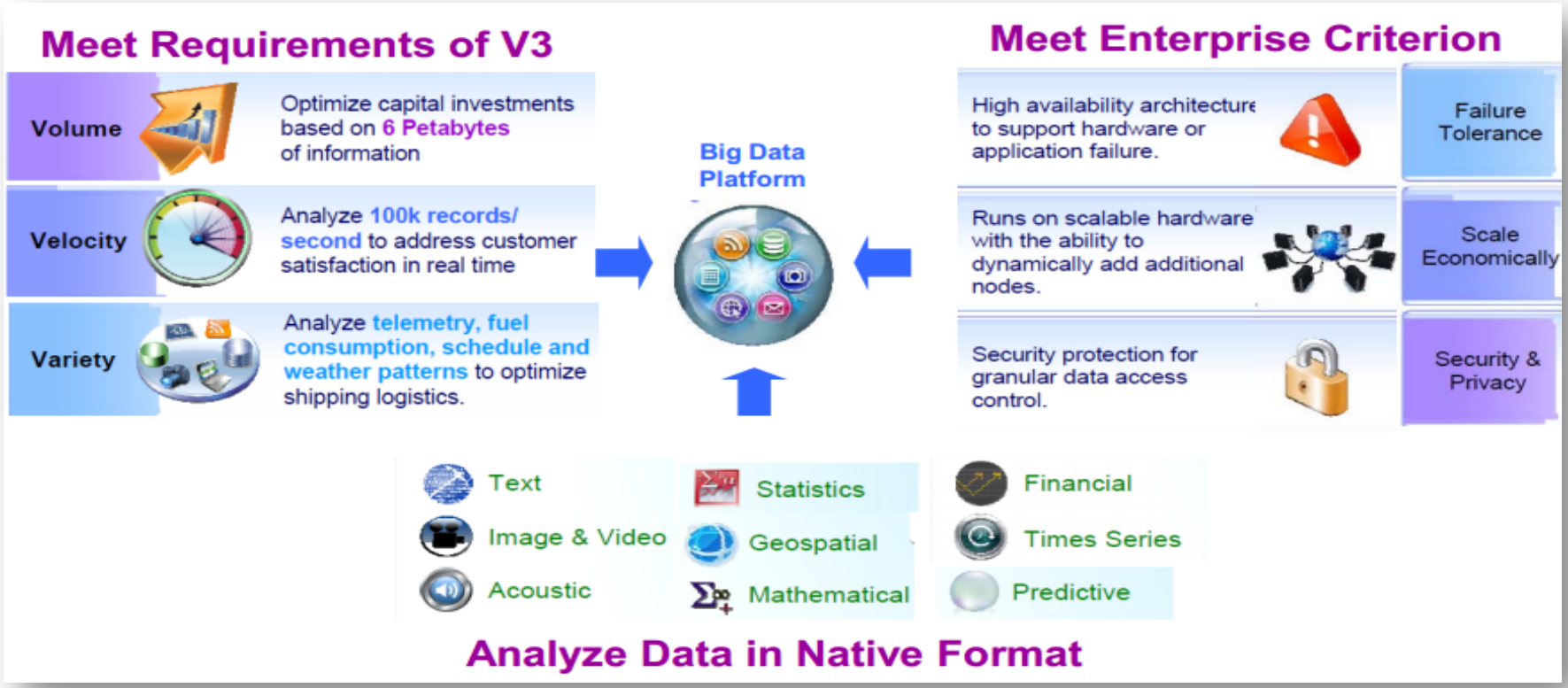
- Many players at different levels/stages



Source: A chart of the big data ecosystem: www.slideshare.net

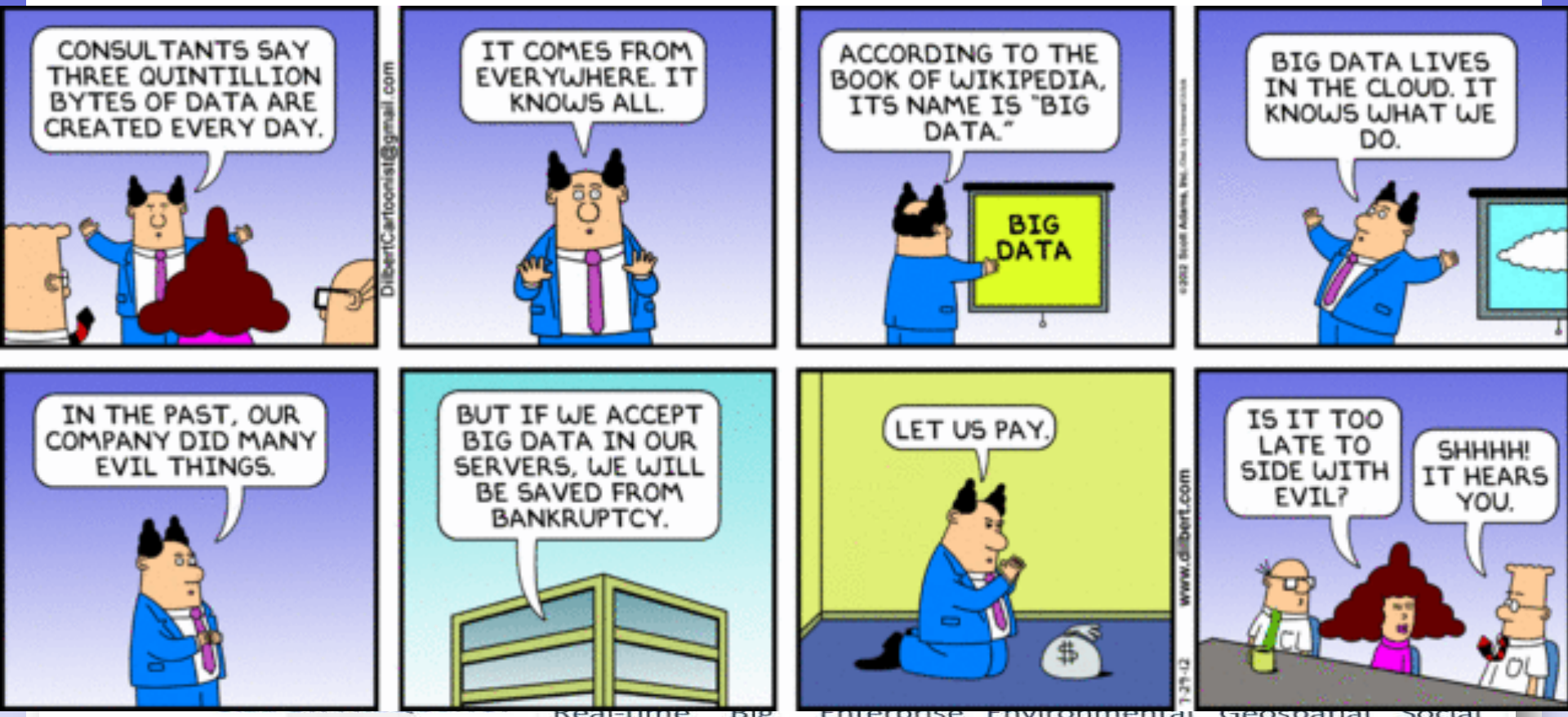
Big Data Solution(s)

- It is all about finding the right fit of tools and technologies that can best serve your Big Data application needs



Source: IMEX Research - Big Data Industry Report 2011-12 ; Storage Networking Industry Association

Big Data - *Harnessing the Power*



Source: Space-Time Insight Takes Predictive Analytics to the Next Level: www.directionsmag.com

Beginning of noSQL

- Relational databases → mainstay of business
- Web-based applications caused spikes
  - explosion of social media sites (Facebook, Twitter) with large data needs
  - rise of cloud-based solutions such as Amazon S3 (simple storage solution)
- Hooking RDBMS to web-based application becomes trouble

noSQL

- Stands for **N**ot **O**nly **SQL** (not **No** To **SQL**)
- The term NOSQL was introduced by Carl Strozzi in 1998 to name his file-based database
- It was again re-introduced by Eric Evans when an event was organized to discuss open source distributed databases
- Eric states that *"… but the whole point of seeking alternatives is that you need to solve a problem that relational databases are a bad fit for. …"*

When to use noSQL?

- Massive write performance.
- Fast key value look ups.
- Flexible schema and data types.
- No single point of failure.
- Out of the box scalability.

Big Data

- Collect.
- Store.
- Organize.
- Analyze.
- Share.

Data growth outruns the ability to manage it so we need **scalable** solutions.

Scale up scalability

- Scale up, Vertical scalability.
  - Increasing server capacity.
  - Adding more CPU, RAM.
  - Reaches limits quickly

Scale Out Scalability

- Scale out, Horizontal scalability.
  - Adding servers to existing system
    - Bugs, hardware errors, things fail all the time.
    - It should become cheaper. Cost efficiency.
  - Shared nothing.
  - Use of commodity/cheap hardware.
  - Heterogeneous systems.
  - Controlled Concurrency (avoid locks).
  - Service Oriented Architecture. Local states.
    - Decentralized to reduce bottlenecks.
    - Avoid Single point of failures.
  - Asynchrony.
  - Symmetry, you don't have to know what is happening. All nodes should be symmetric.

What is Wrong With RDBMS?

- Nothing. One size fits all? Not really.
- Impedance mismatch.
  - Object Relational Mapping doesn't work quite well.
- Rigid schema design.
- Harder (expensive) to scale.
- Replication (expensive)
- Joins across multiple nodes? Hard.
- How does RDMS handle data growth? Hard.
- Many programmers are already familiar with it.
- Transactions and ACID make development easy.
- Lots of tools to use.

Issues with *scaling up*

- Best way to provide ACID and rich query model is to have the dataset on a single machine
- Limits to ***scaling up*** (or ***vertical scaling***: make a "single" machine more powerful) → dataset is just too big!
- ***Scaling out*** (or ***horizontal scaling***: adding more smaller/cheaper servers) is a better choice
- Different approaches for horizontal scaling (multi-node database):
  - Master/Slave
  - Sharding (partitioning)

Scaling out RDBMS: Master/Slave

- Master/Slave
  - All writes are written to the master
  - All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large datasets can pose problems as master needs to duplicate data to slaves

Scaling out RDBMS: Sharding

- Sharding (Partitioning)
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards

Disadvantage of NOSQL?

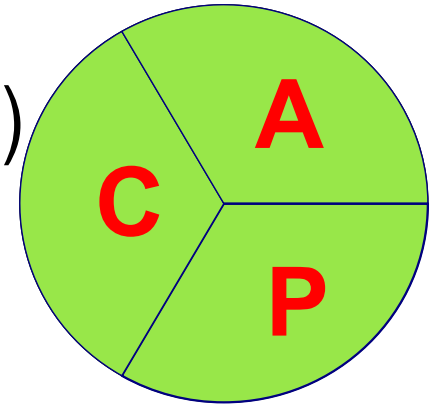
- Don't fully support relational features
  - no join, group by, order by operations (except within partitions)
  - no referential integrity constraints across partitions
- No declarative query language (e.g., SQL) → more programming
- Relaxed ACID (see CAP theorem) → fewer guarantees
- No easy integration with other applications that support SQL

Who is using noSQL?

- BigTable - Google
- Dynamo - Amazon
- Cassandra – Facebook, twitter
- Voldemart - LinkedIn
- Hypertable - Baidu
- Hbase – opensource

CAP Theorem

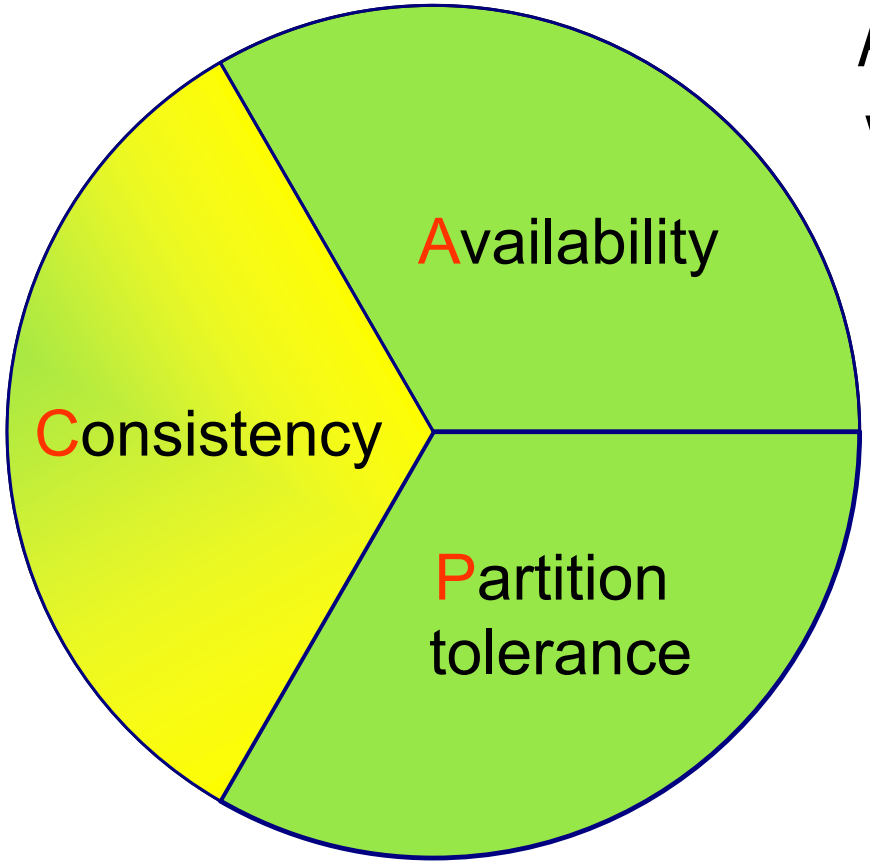
- Suppose three properties of a distributed system (sharing data)
  - **Consistency:**
    - all copies have same value
  - **Availability:**
    - reads and writes always succeed
  - **Partition-tolerance:**
    - system properties (consistency and/or availability) hold even when network failures prevent some machines from communicating with others



CAP Theorem

- **Brewer's CAP Theorem:**
  - *For any system sharing data, it is "impossible" to guarantee simultaneously all of these three properties*
  - You can have at most two of these three properties for any shared-data system
- Very large systems will "partition" at some point:
  - That leaves either **C** or **A** to choose from (traditional DBMS prefers **C** over **A** and **P**)
  - In almost all cases, you would choose **A** over **C** (except in specific applications such as order processing)

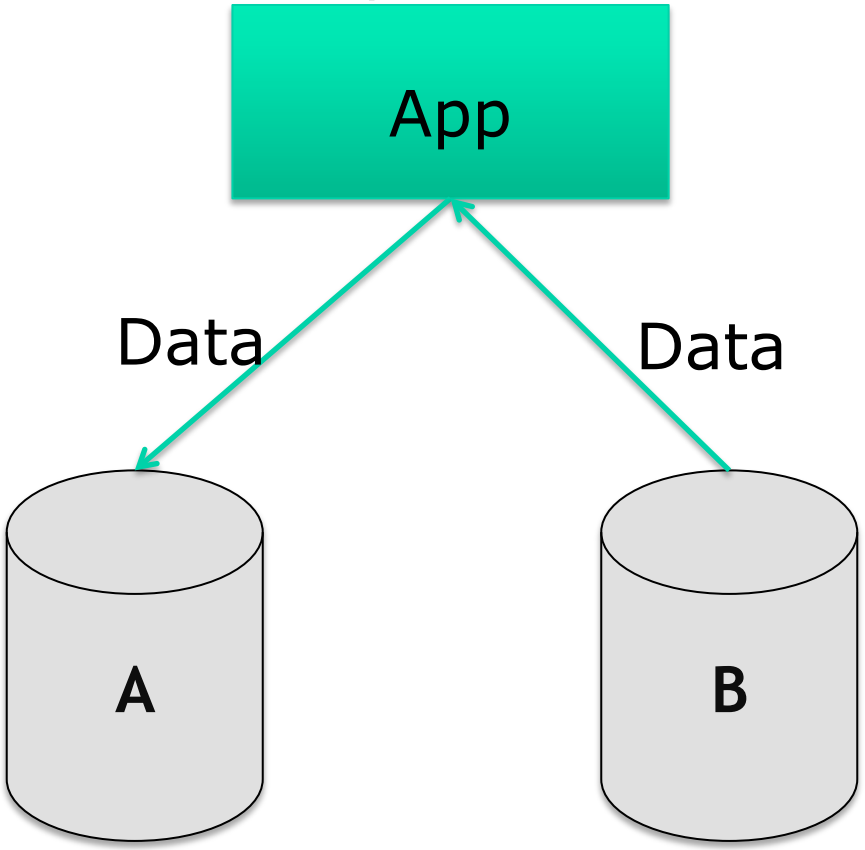
CAP Theorem



All client always have the same view of the data

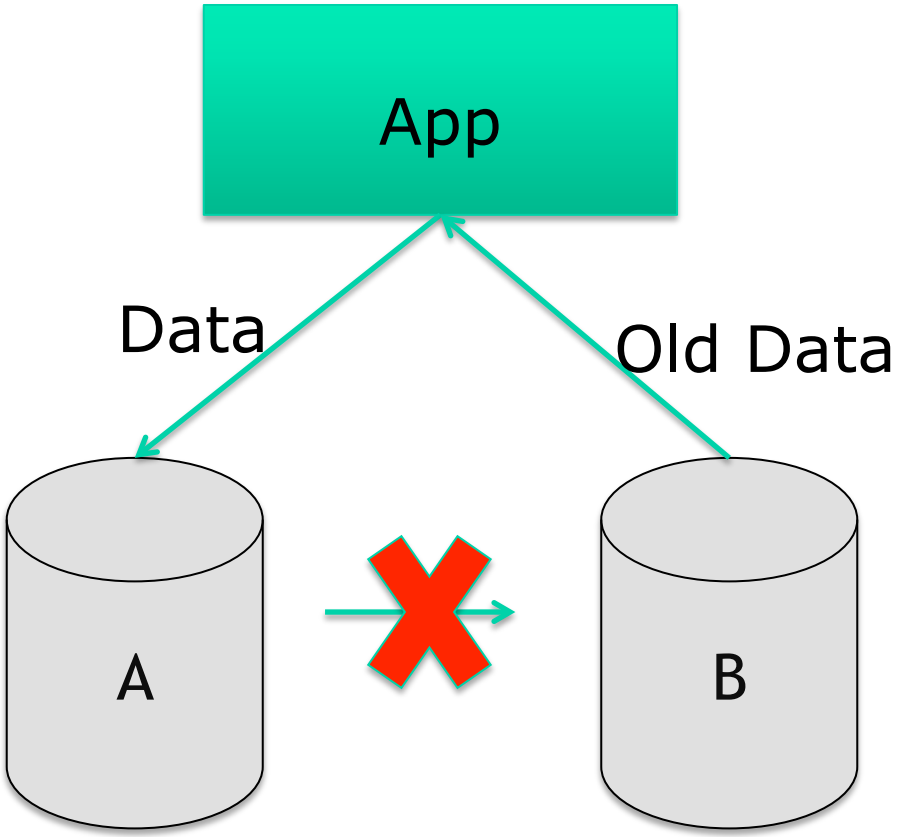
Simple Illustration

Consistent and available
No partition.



Simple Illustration

Available and partitioned
Not consistent, we get back old data.



CAP Theorem (contd.)

- **Consistency**
  - 2 types of consistency:
    1. Strong consistency – ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability)
    2. Weak consistency – BASE (**B**asically **A**vailable **S**oft-state **E**ventual consistency)

CAP Theorem (contd.)

- **ACID**
  - A DBMS is expected to support "ACID transactions," with:
  - **A**tomicity: either the whole process is done or none is
  - **C**onsistency: only valid data are written
  - **I**solation: one operation at a time
  - **D**urability: once committed, it stays that way

- **CAP**
  - **C**onsistency: all data on cluster has the same copies
  - **A**vailability: cluster always accepts reads and writes
  - **P**artition tolerance: guaranteed properties are maintained even when network failures prevent some machines from communicating with others

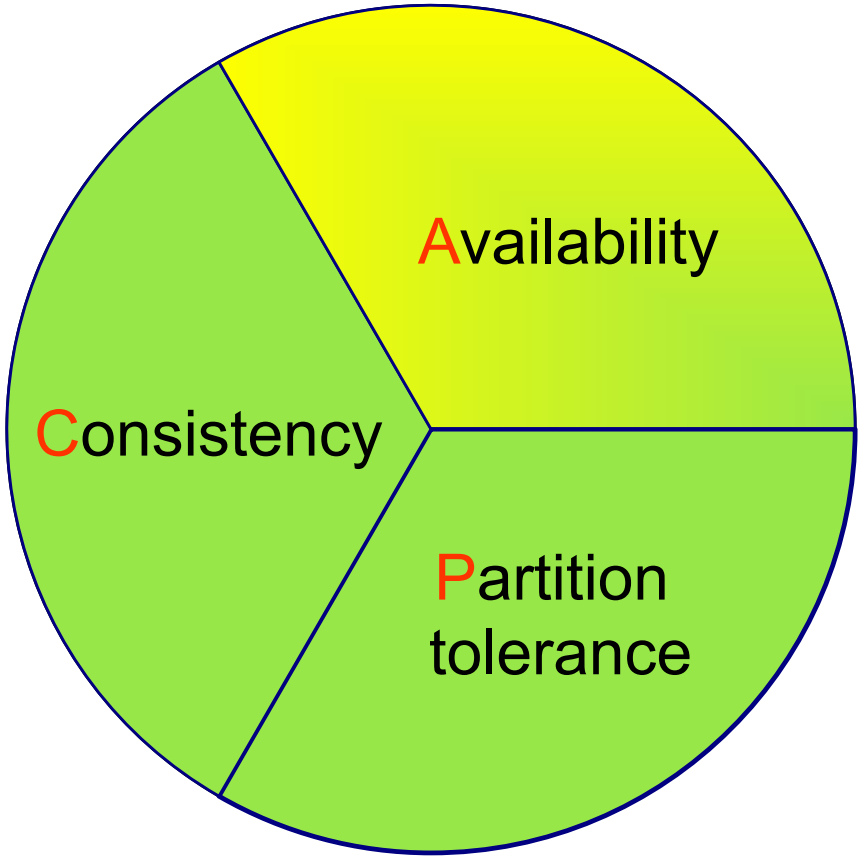
CAP Theorem (contd.)

- A consistency model determines rules for visibility and apparent order of updates
- Example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses
  - Client B reads row X from node M
  - **Does client B see the write from client A?**
  - Consistency is a continuum with tradeoffs
  - **For NOSQL, the answer would be: "maybe"**
  - CAP theorem states: *"strong consistency can't be achieved at the same time as availability and partition-tolerance"*

CAP Theorem (contd.)

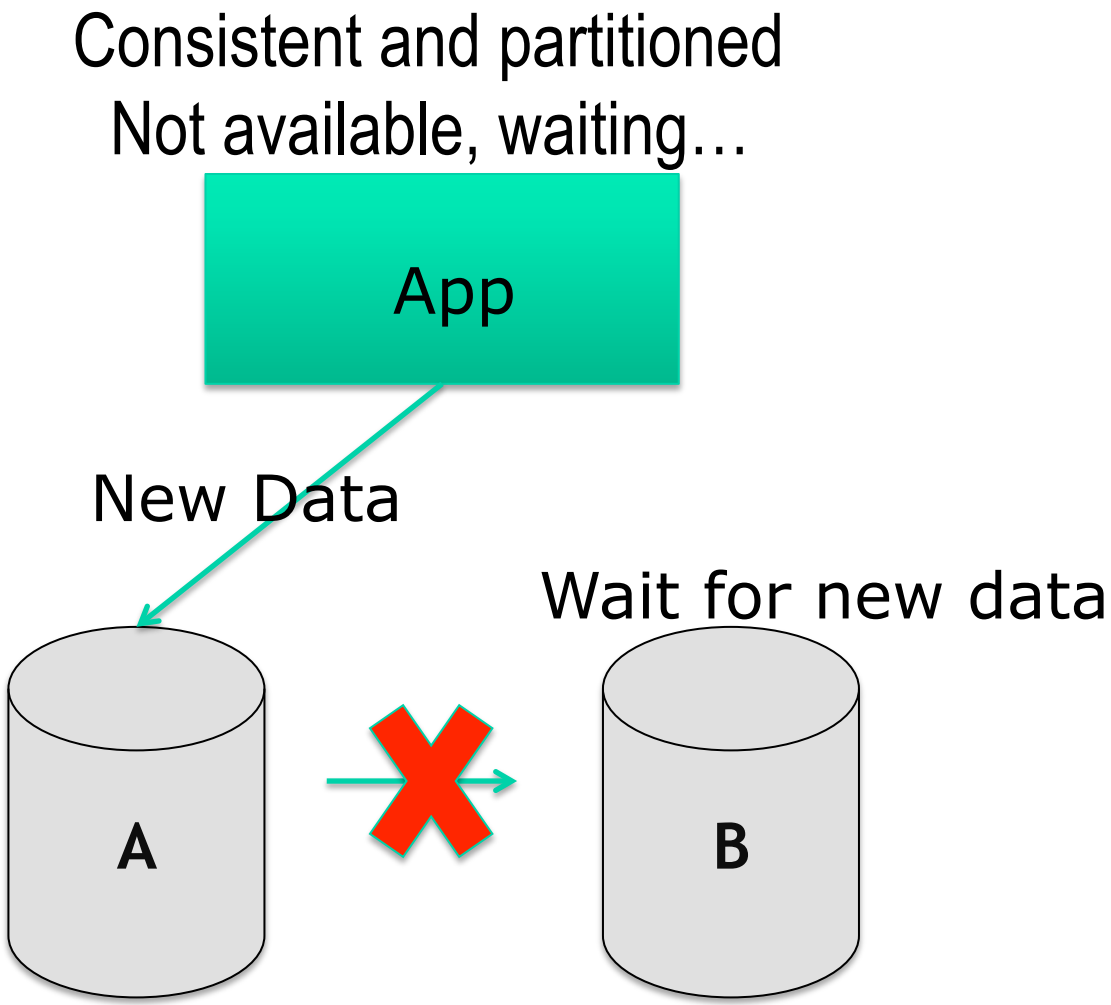
- Eventual consistency
  - When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
- Cloud computing
  - ACID is hard to achieve, moreover, it is not always required, e.g. for blogs, status updates, product listings, etc.

CAP Theorem (contd.)

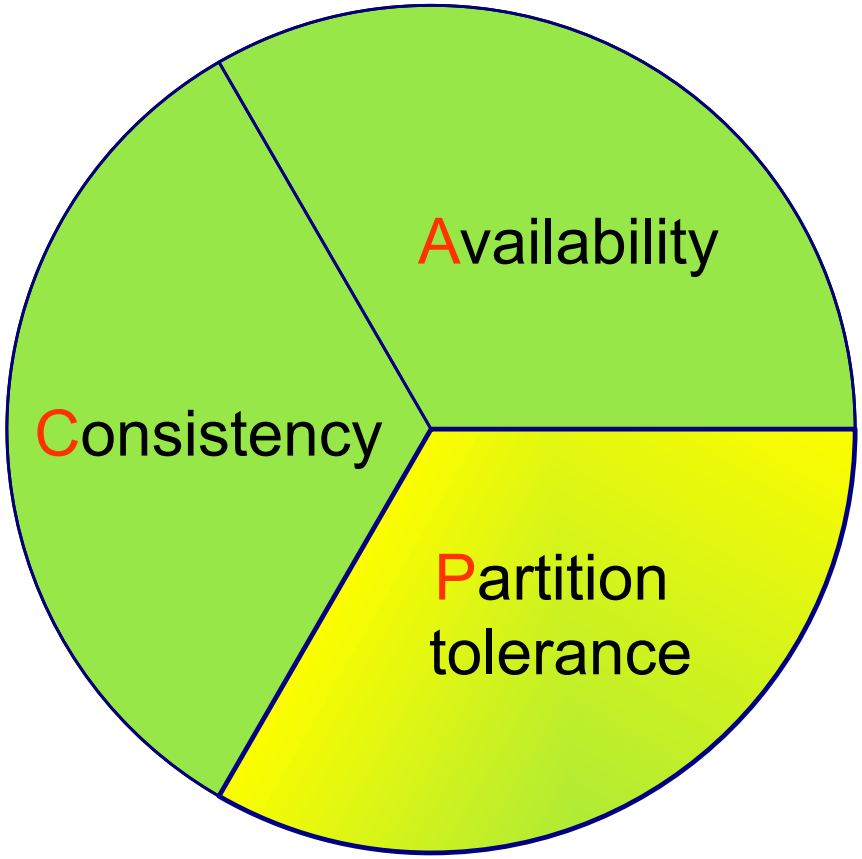


Each client always can read and write.

Simple Illustration



CAP Theorem (contd.)

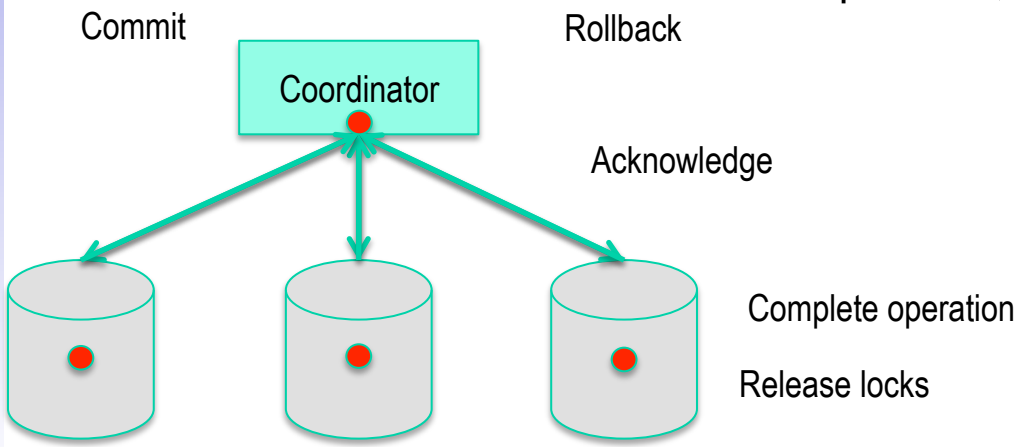


A system can continue to operate in the presence of a network partitions

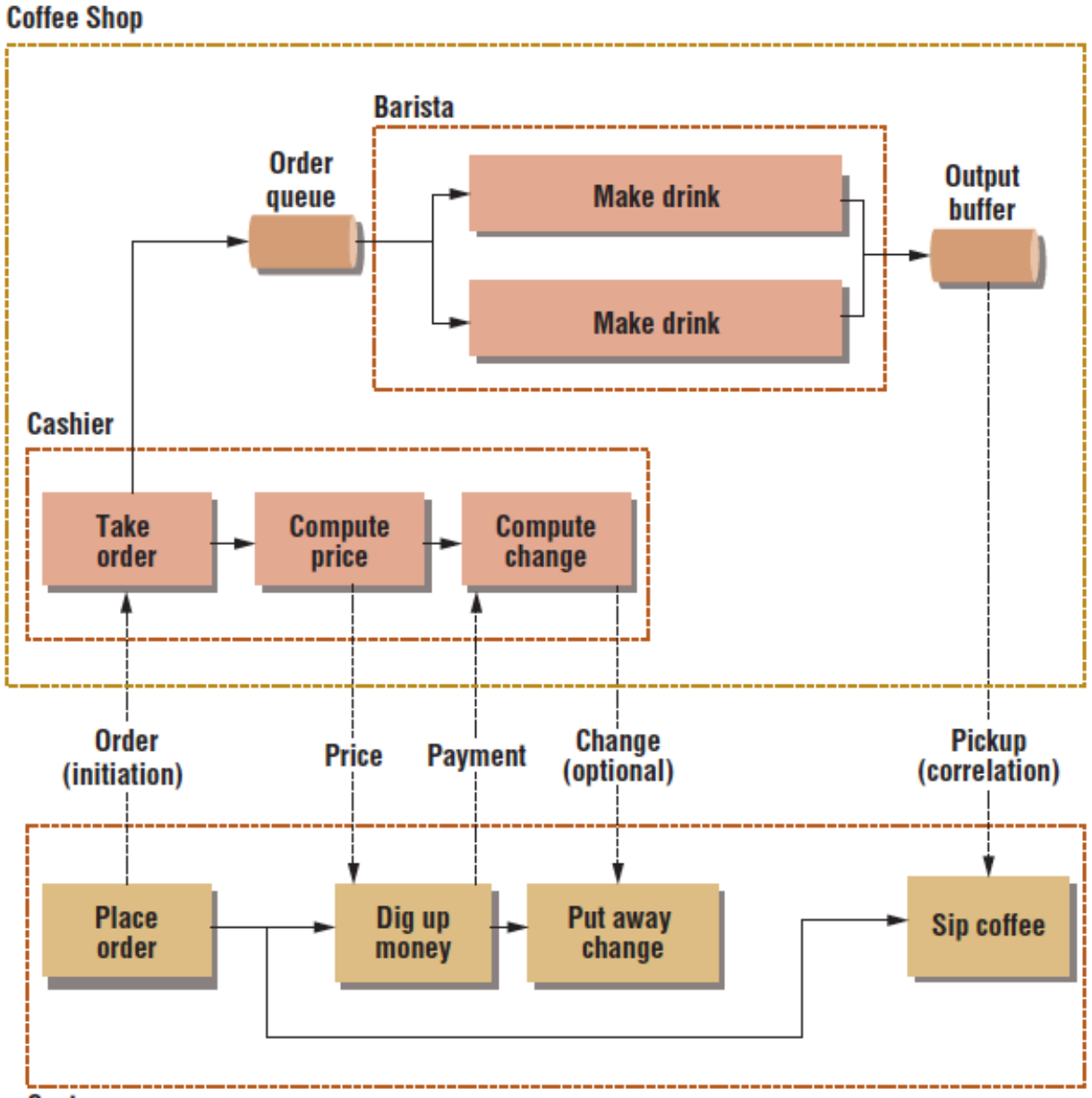
Distributed Transactions

- Two phase commit.
  - Atomic Commitment Protocol
  - Coordinates all processes that participate in a distributed atomic transaction on whether to commit or abort (roll back) the transaction.
- Possible failures
  - Network errors.
  - Node errors.
  - Database errors.

Problems:
Locking the entire cluster if one node is down
Possible to implement timeouts.
Possible to use Quorum.
Quorum: in a distributed environment, if there is partition, then the nodes vote to commit or rollback.



Increasing Efficiency

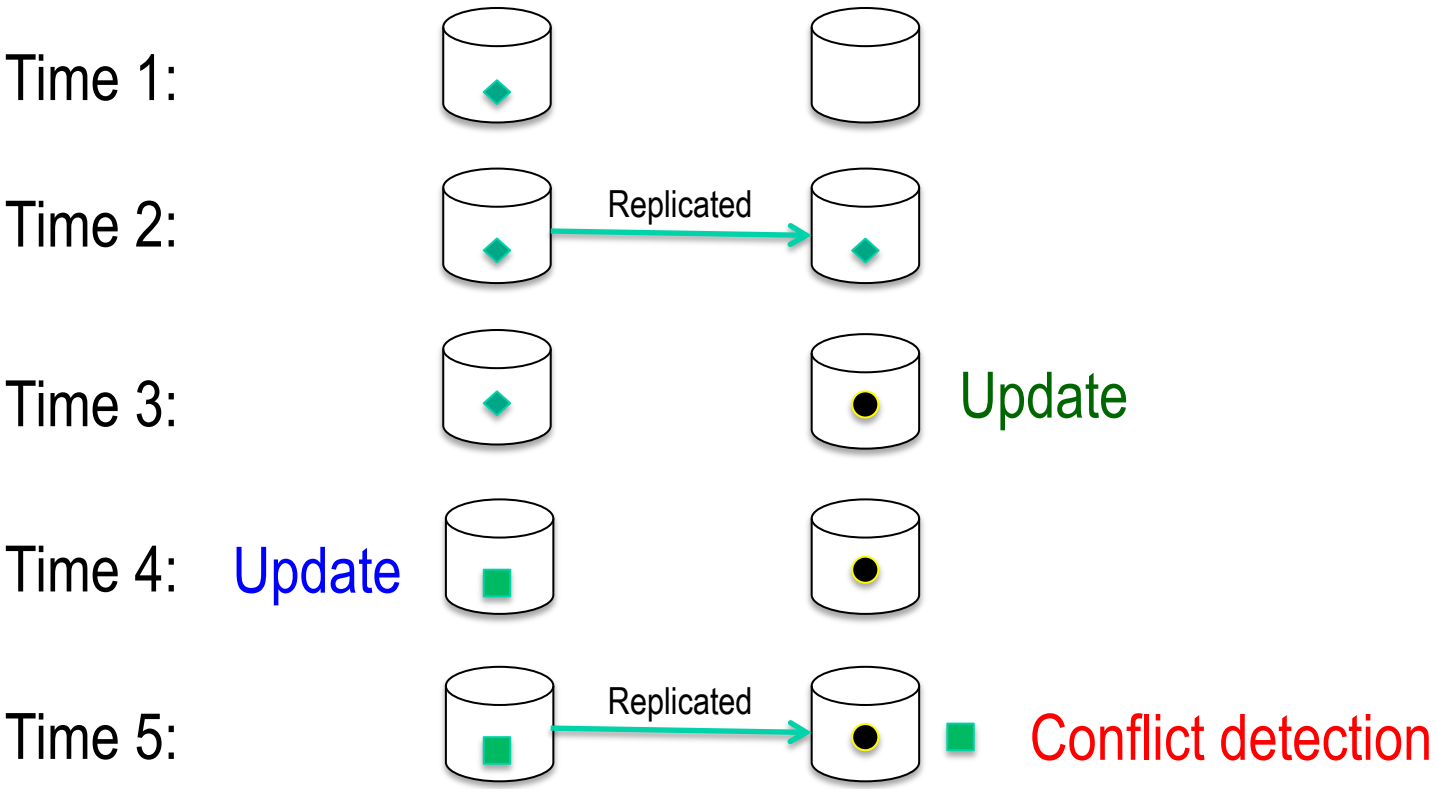


http://www.akamai.com/dl/te

dwideweb.pdf

Vector Clocks

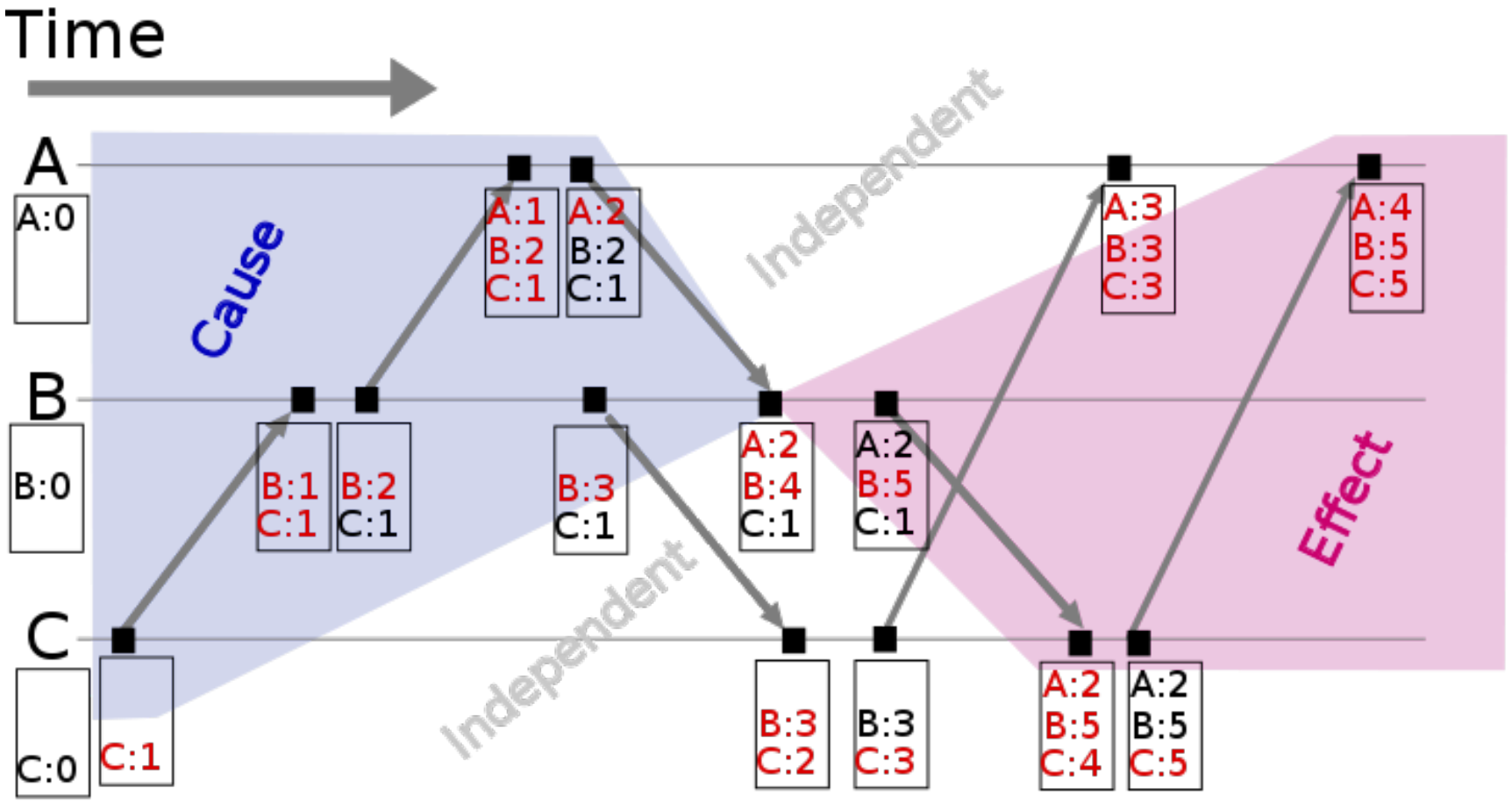
- Used for conflict **detection** of data.
- Timestamp based resolution of conflicts is not enough.



Vector Clocks

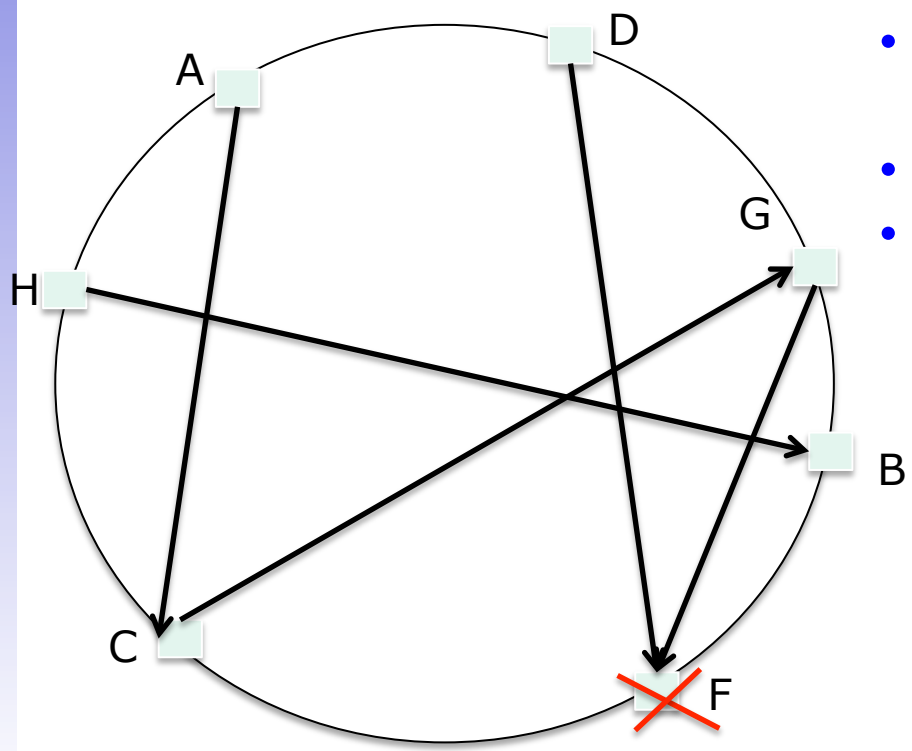
- generating a partial ordering of events in a distributed system and detecting causality violations
- A vector clock of a system of N processes is an array/vector of N logical clocks, one clock per process; a local "smallest possible values" copy of the global clock-array is kept in each process, with the following rules for clock updates:
  - Initially all clocks are zero.
  - Each time a process experiences an internal event, it increments its own logical clock in the vector by one.
  - Each time a process prepares to send a message, it sends its entire vector along with the message being sent.
  - Each time a process receives a message, it increments its own logical clock in the vector by one and updates each element in its vector by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

Vector Clocks



Gossip Protocol & Hinted Handoffs

- Most preferred communication protocol in a distributed environment is Gossip Protocol.



- All the nodes talk to each other peer wise.
- There is no global state.
- No single point of coordinator.
- If one node goes down and there is a Quorum load for that node is shared among others.
- Self managing system.
- If a new node joins, load is also distributed.

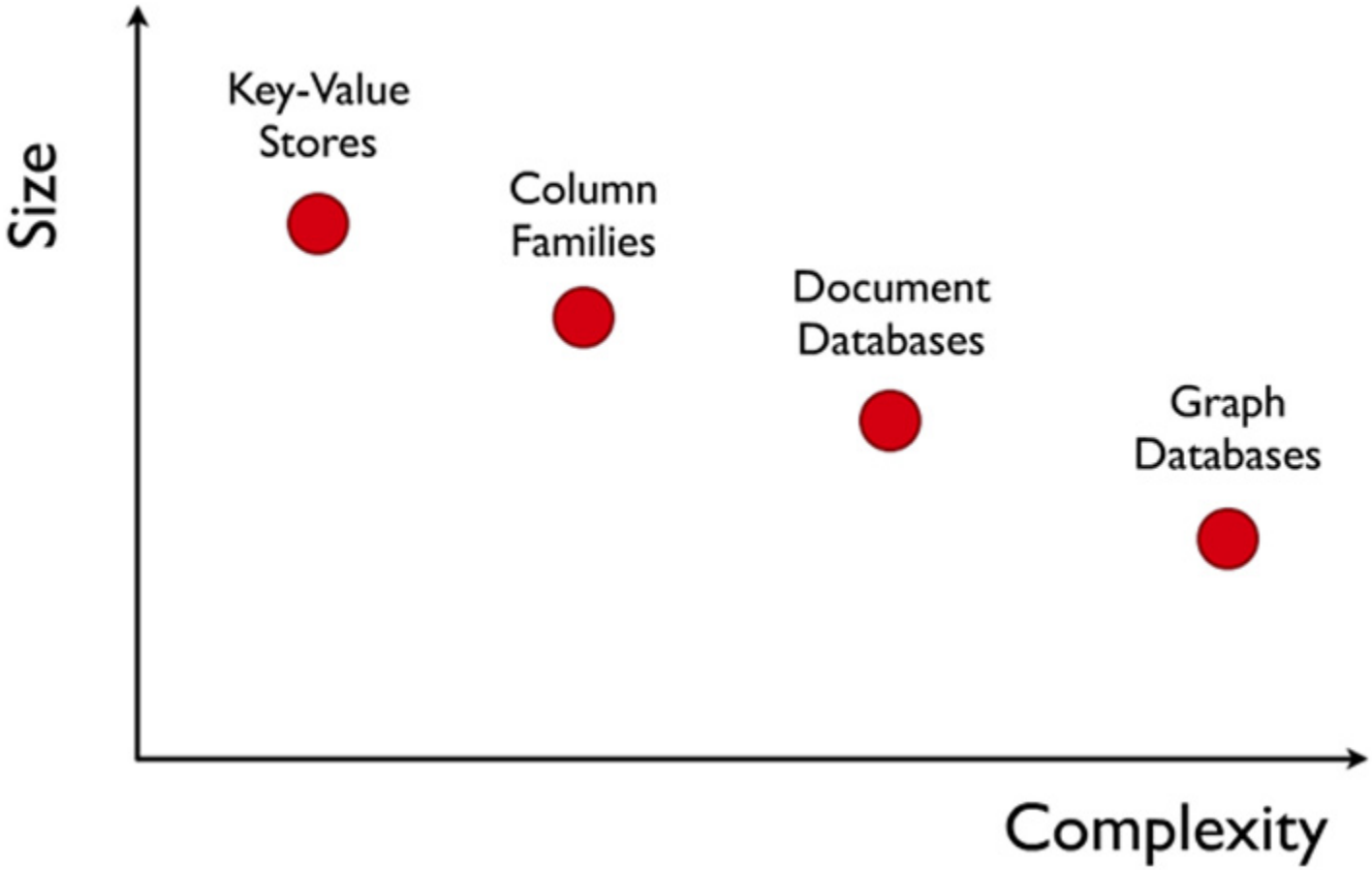
Requests coming to F will be handled by the nodes who takes the load of F, lets say C with the **hint** that it took the requests which was for F, when F becomes available, F will get this Information from C. Self healing property.

NOSQL categories

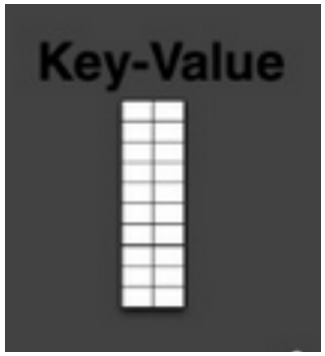
1. Key-value
   - Example: DynamoDB, Voldermort, Scalaris
2. Document-based
   - Example: MongoDB, CouchDB
3. Column-based
   - Example: BigTable, Cassandra, HBase
4. Graph-based
   - Example: Neo4J, InfoGrid

- "No-schema" is a common characteristics of most NOSQL storage systems
- Provide "flexible" data types

Complexity



Key-value



- Focus on scaling to huge amounts of data
- Designed to handle massive load
- Based on Amazon's dynamo paper
- Data model: (global) collection of Key-value pairs
- *Dynamo ring partitioning* and *replication*
- Example: (DynamoDB)
  - *items* having one or more attributes (name, value)
  - An *attribute* can be single-valued or multi-valued like set.
  - items are combined into a *table*

Key-value

- Basic API access:
  - get(key): extract the value given a key
  - put(key, value): create or update the value given its key
  - delete(key): remove the key and its associated value
  - execute(key, operation, parameters): invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map .... etc)

Key-value

Pros:

- very fast
- very scalable (horizontally distributed to nodes based on key)
- simple data model
- eventual consistency
- fault-tolerance

Cons:

- Can’t model more complex data structure such as objects

Key-value

Name	Producer	Data model	Querying
SimpleDB	Amazon	set of couples (key, {attribute}), where attribute is a couple (name, value)	restricted SQL; select, delete, GetAttributes, and PutAttributes operations
Redis	Salvatore Sanfilippo	set of couples (key, value), where value is simple typed value, list, ordered (according to ranking) or unordered set, hash value	primitive operations for each value type
Dynamo	Amazon	like SimpleDB	simple get operation and put in a context
Voldemort	LinkeId	like SimpleDB	similar to Dynamo

Document-based

- Can model more complex objects
- Inspired by Lotus Notes
- Data model: collection of documents
- Document: JSON (**J**ava**S**cript **O**bject **N**otation is a data model, key-value pairs, which supports objects, records, structs, lists, array, maps, dates, Boolean with **nesting**), XML, other semi-structured formats.

Document-based

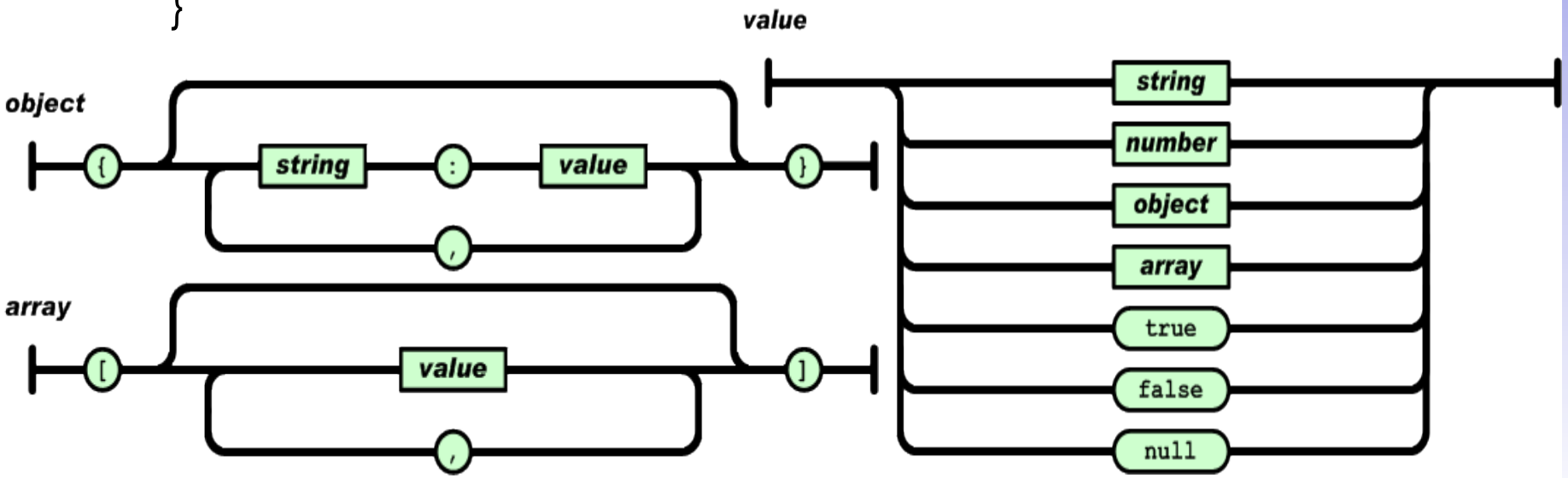
- Example: (MongoDB) document
  - {Name:"Jaroslav",

    Address:"Malostranske nám. 25, 118 00 Praha 1”,

    Grandchildren: {Claire: "7", Barbara: "6", "Magda: "3", "Kirsten: "1", "Otis: "3", Richard: "1“}

    Phones: [“123-456-7890”, “234-567-8963”]

    }



Document-based

Name	Producer	Data model	Querying
MongoDB	10gen	object-structured documents stored in collections; each object has a primary key called ObjectId	manipulations with objects in collections (find object or objects via simple selections and logical expressions, delete, update,)
Couchbase	Couchbase[1]	document as a list of named (structured) items (JSON document)	by key and key range, views via Javascript and MapReduce

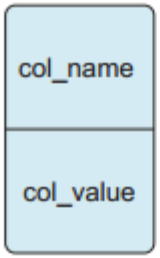
Column-based

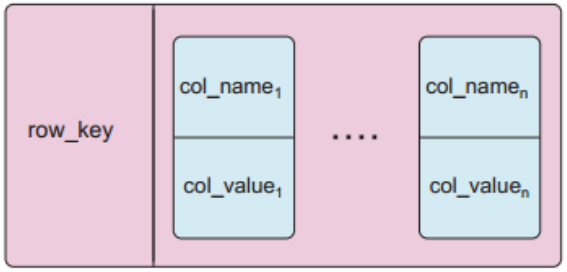
- Based on Google's BigTable paper, Like column oriented relational databases (store data in column order) but with a twist, Tables similarly to RDBMS, but handle semi-structured
- Data model:
  - Collection of Column Families
  - Column family = (key, value) where value = set of **related** columns (standard, super)
  - indexed by *row key*, *column key* and *timestamp*

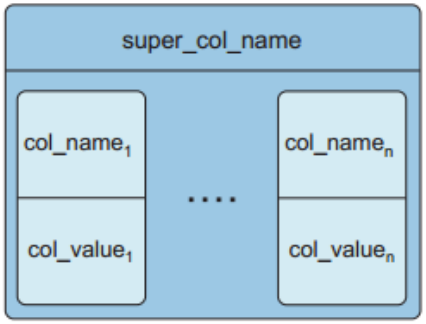
allow key-value pairs to be stored (and retrieved on key) in a massively parallel system

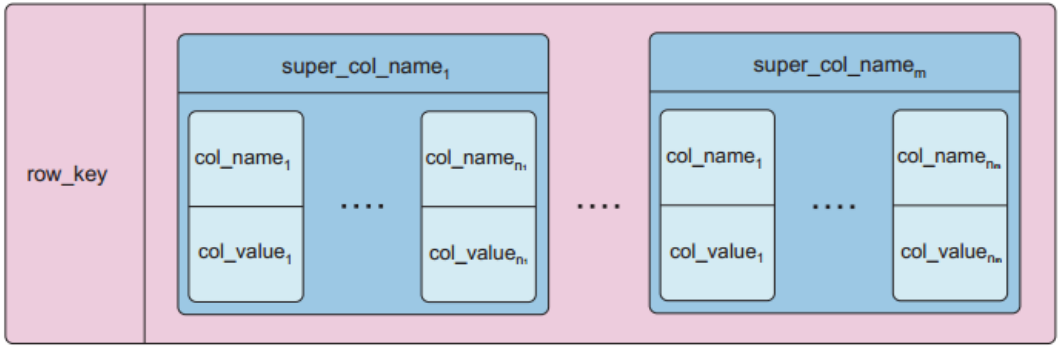
storing principle: big hashed distributed tables

properties: partitioning (horizontally and/or vertically), high availability etc. completely transparent to application









Column-based

- One column family can have variable numbers of columns
- Cells within a column family are sorted “physically”
- Very sparse, most cells have null values
- **Comparison:** RDBMS vs column-based NOSQL
  - Query on multiple tables
    - **RDBMS:** must fetch data from several places on disk and glue together
    - **Column-based NOSQL:** only fetch column families of those columns that are required by a query (all columns in a column family are stored together on the disk, so multiple rows can be retrieved in one read operation → data locality)

Column-based

- Example: (Cassandra column family--timestamps removed for simplicity)

UserProfile = {

Cassandra = { emailAddress:”casandra@apache.org” , age:”20”}

TerryCho = { emailAddress:”terry.cho@apache.org” , gender:”male”}

Cath = { emailAddress:”cath@apache.org” , age:”20”,gender:”female”,address:”Seoul”}

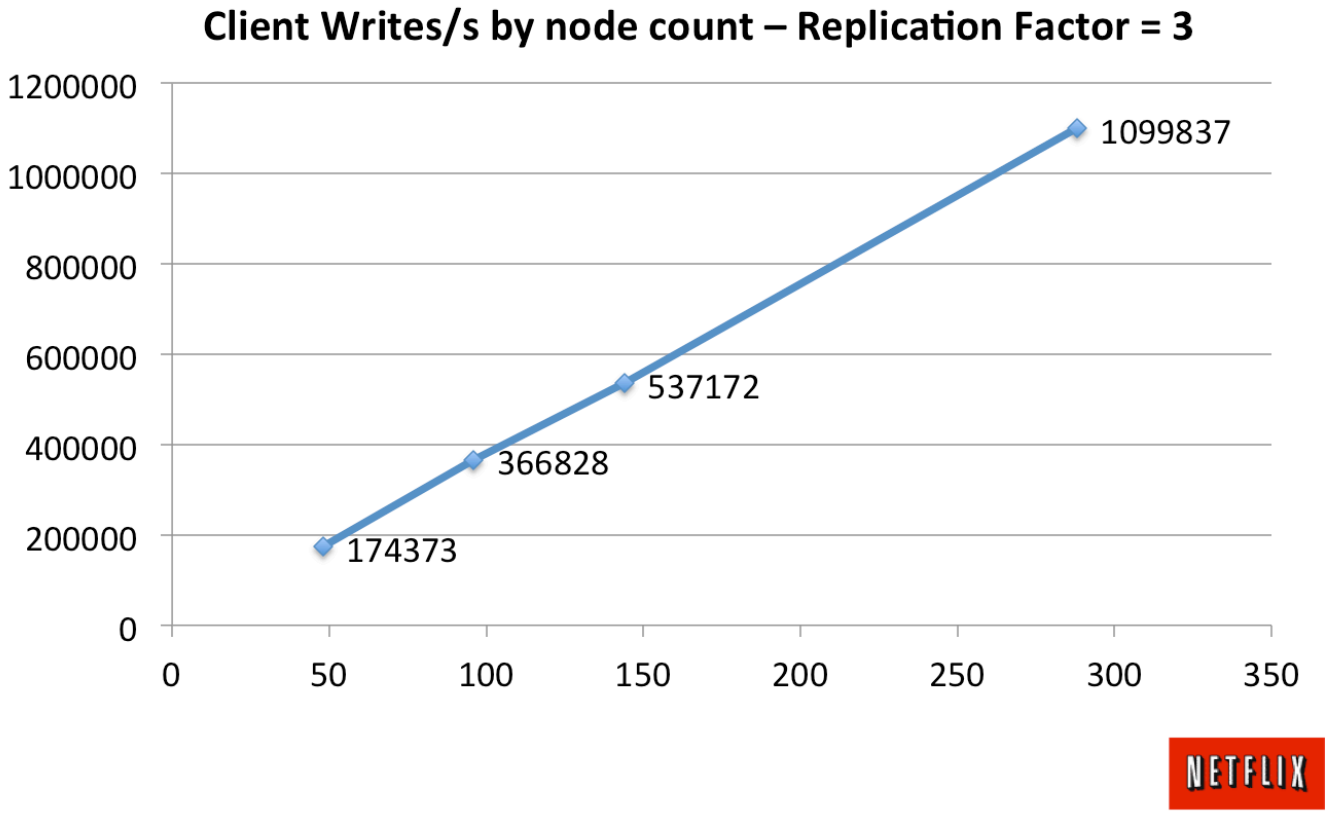
}

Column-based

Name	Producer	Data model	Querying
BigTable	Google	set of couples (key, {value})	selection (by combination of row, column, and time stamp ranges)
HBase	Apache	groups of columns (a BigTable clone)	JRUBY IRB-based shell (similar to SQL)
Hypertable	Hypertable	like BigTable	HQL (Hypertext Query Language)
CASSANDRA	Apache (originally Facebook)	columns, groups of columns corresponding to a key (supercolumns)	simple selections on key, range queries, column or columns ranges
PNUTS	Yahoo	(hashed or ordered) tables, typed arrays, flexible schema	selection and projection from a single table (retrieve an arbitrary single record by primary key, range queries, complex predicates, ordering, top-k)

Cassandra at Netflix

Scale-Up Linearity



http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html

Graph Stores

- Based on Graph Theory.
- Focus on modeling the structure of data (*interconnectivity*)
- Scale vertically, no clustering.
- You can use graph algorithms easily.

Which one to use?

- Key-value stores:
  - Processing a constant stream of small reads and writes.
- Document databases:
  - Natural data modeling. Programmer friendly. Rapid development. Web friendly, CRUD.
- RDMBS:
  - OLTP. SQL. Transactions. Relations.
- OODBMS
  - Complex object models.
- Data Structure Server:
  - Quirky stuff.
- Columnar:
  - Handles size well. Massive write loads. High availability. Multiple-data centers. MapReduce
- Graph:
  - Graph algorithms and relations.