NoSQL Databases

Slides take from

J. Pokorný

KSI MFF UK

Cloud computing, cloud databases

- Cloud computing
  - data intensive applications on hundreds of thousands of commodity servers and storage devices
  - basic features:
    - elasticity,
    - fault-tolerance
    - automatic provisioning
- Cloud databases: traditional scaling up (adding new expensive big servers) is not possible
  - requires higher level of skills
  - is not reliable in some cases
- Architectural principle: scaling out (or horizontal scaling) based on data partitioning, i.e. dividing the database across many (inexpensive) machines

Cloud computing, cloud databases

- Technique: data sharding, i.e. horizontal partitioning of data (e.g. hash or range partitioning)
- Consequences:
  - manage parallel access in the application
  - scales well for both reads and writes
  - not transparent, application needs to be partition-aware

Relaxing ACID properties

- Cloud computing: ACID is hard to achieve, moreover, it is not always required, e.g. for blogs, status updates, product listings, etc.
- Availability
  - Traditionally, thought of as the server/process available 99.999 % of time
  - For a large-scale node system, there is a high probability that a node is either down or that there is a network partitioning
- Partition tolerance
  - ensures that write and read operations are redirected to available replicas when segments of the network become disconnected

Eventual Consistency

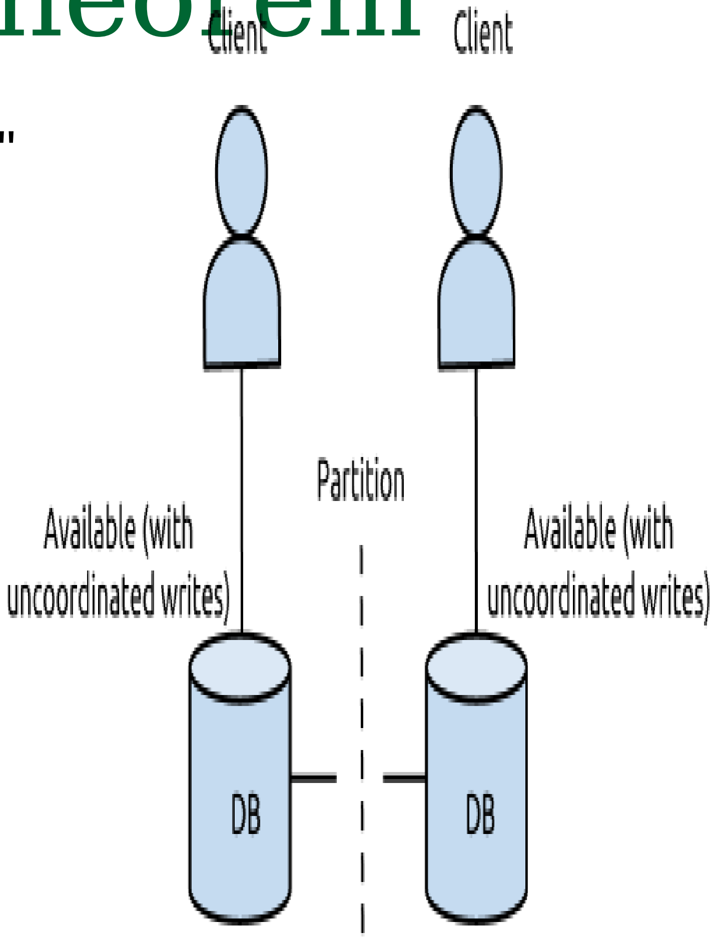
- Eventual Consistency
  - When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
  - For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service
- BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency) properties, as opposed to ACID
  - Soft state: copies of a data item may be inconsistent
  - Eventually Consistent – copies becomes consistent at some later time if there are no more updates to that data item
  - Basically Available – possibilities of faults but not a fault of the whole system

CAP Theorem

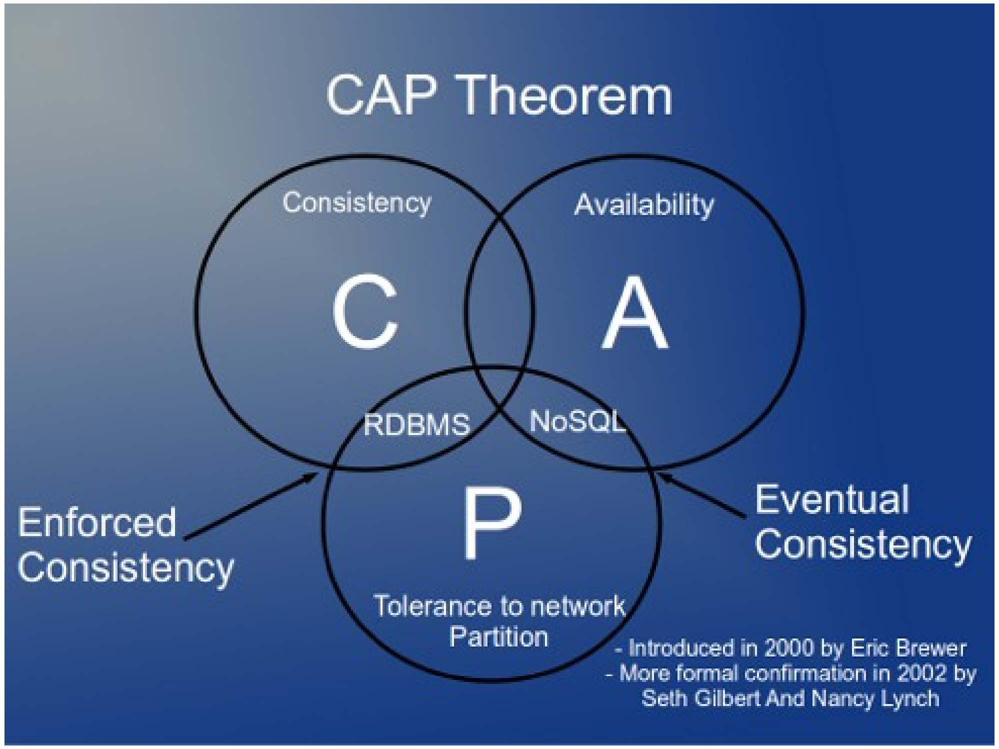
- Suppose three properties of a system
  - Consistency (all copies have same value)
  - Availability (system can run even if parts have failed)
  - Partitions (network can break into two or more parts, each with active systems that can not influence other parts)
- Brewer’s CAP “Theorem”: for any system sharing data it is impossible to guarantee simultaneously all of these three properties
- Very large systems will partition at some point
  - it is necessary to decide between C and A
  - traditional DBMS prefer C over A and P
  - most Web applications choose A (except in specific applications such as order processing)

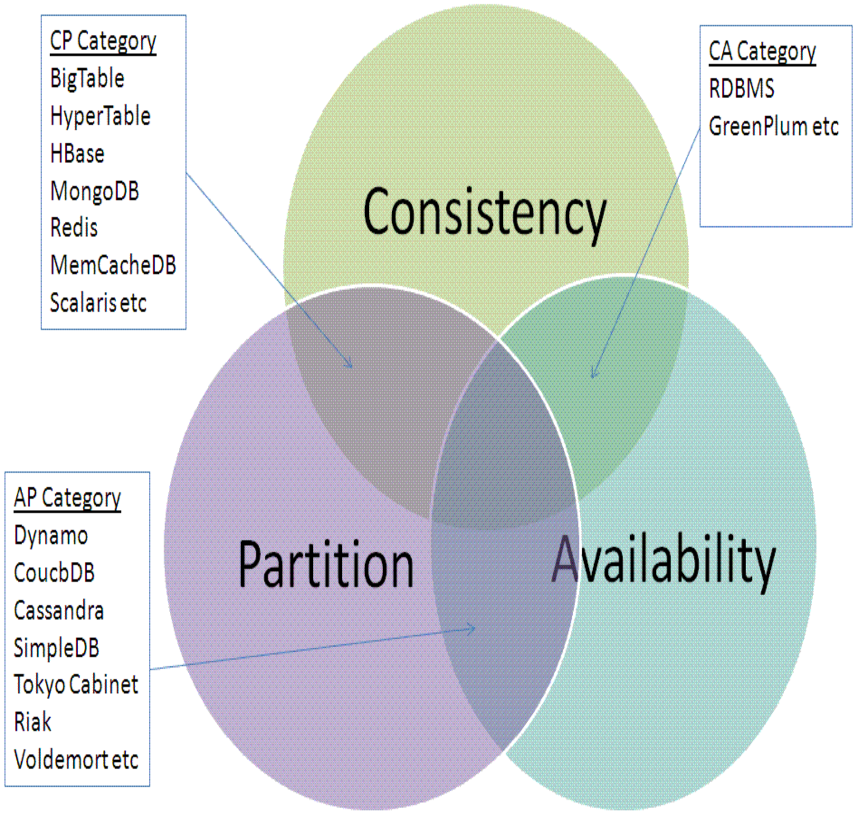
https://foundationdb.com/white-papers/the-cap-theorem

- Brewer originally described this impossibility result as forcing a choice of "two out of the three" **CAP** properties, leaving three viable design options: **CP**, **AP**, and **CA**.
- However, further consideration shows that **CA** is not really a coherent option because a system that is not **P**artition-tolerant will, by definition, be forced to give up **C**onsistency or **A**vailability during a partition.
- A more modern interpretation of the theorem is: *during a network partition, a distributed system must choose either **C**onsistency or **A**vailability.*


Client
Client
Partition
Available (with uncoordinated writes)
Available (with uncoordinated writes)
DB
DB

CAP Theorem


CAP Theorem
Consistency
C
Availability
A
RDBMS
NoSQL
P
Tolerance to network Partition
Enforced Consistency
Eventual Consistency
- Introduced in 2000 by Eric Brewer
- More formal confirmation in 2002 by Seth Gilbert And Nancy Lynch


CP Category
BigTable
HyperTable
HBase
MongoDB
Redis
MemCacheDB
Scalaris etc
CA Category
RDBMS
GreenPlum etc
AP Category
Dynamo
CoucbDB
Cassandra
SimpleDB
Tokyo Cabinet
Riak
Voldemort etc
Consistency
Partition
Availability

CAP Theorem

- Drop A or C of ACID
  - relaxing C makes replication easy, facilitates fault tolerance,
  - relaxing A reduces (or eliminates) need for distributed concurrency control.

NoSQL databases

- The name stands for **N**ot **O**nly **SQL**
- Common features:
  - non-relational
  - usually do not require a fixed table schema
  - horizontal scalable
  - mostly open source
- More characteristics
  - relax one or more of the ACID properties (see CAP theorem)
  - replication support
  - easy API (if SQL, then only its very restricted variant)
- Do not fully support relational features
  - no join operations (except within partitions),
  - no referential integrity constraints across partitions.

Categories of NoSQL databases

- key-value stores
- column NoSQL databases
- document-based
- XML databases (myXMLDB, Tamino, Sedna)
- graph database (neo4j, InfoGrid)

Categories of NoSQL databases

- key-value stores
- column NoSQL databases
- document-based
- XML databases (myXMLDB, Tamino, Sedna)
- graph database (neo4j, InfoGrid)

Key-Value Data Stores

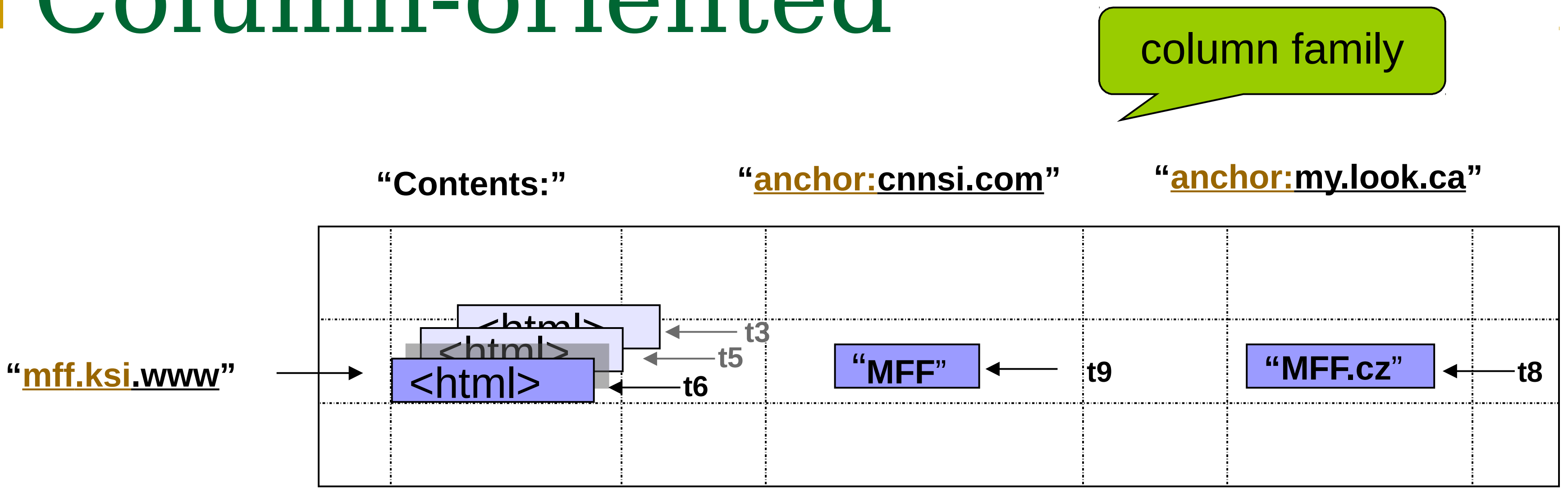
- Example: SimpleDB
  - Based on Amazon's Single Storage Service (S3)
  - items (represent objects) having one or more pairs (name, value), where name denotes an attribute.
  - An attribute can have multiple values.
  - items are combined into domains.

Column-oriented*

- store data in column order
- allow key-value pairs to be stored (and retrieved on key) in a massively parallel system
  - data model: families of attributes defined in a schema, new attributes can be added
  - storing principle: big hashed distributed tables
  - properties: partitioning (horizontally and/or vertically), high availability etc. completely transparent to application

* Better: extendible records

Column-oriented



- Example: BigTable
  - indexed by row key, column key and timestamp. i.e. (row: string , column: string , time: int64) ✉ String.
  - rows are ordered in lexicographic order by row key.
  - row range for a table is dynamically partitioned, each row range is called a tablet.
  - columns: syntax is family:qualifier

A table representation of a row in BigTable

Row key	Time stamp	Column name	Column family Grandchildren		
http://ksi....	t1	"Jack"	"Claire" 7		
	t2	"Jack"	"Claire" 7	"Barbara" 6	
	t3	"Jack"	"Claire" 7	"Barbara" 6	"Magda" 3

Column-oriented

- Example: Cassandra
  - keyspace: Usually the name of the application; e.g., 'Twitter', 'Wordpress‘.
  - column family**:** structure containing an unlimited number of rows
  - column**:** a tuple with name, value and time stamp
  - key**:** name of record
  - super column**:** contains more columns

Document-based

- based on JSON format: a data model which supports lists, maps, dates, Boolean with nesting
- Really: *indexed* semistructured documents
- Example: Mongo
  - {Name:"Jaroslav",

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

    Grandchildren: [Claire: "7", Barbara: "6", "Magda: "3", "Kirsten: "1", "Otis: "3", Richard: "1"]

    }

Typical NoSQL API

- 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).

Representatives of NoSQL databases

key valued

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

Representatives of NoSQL databases

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)

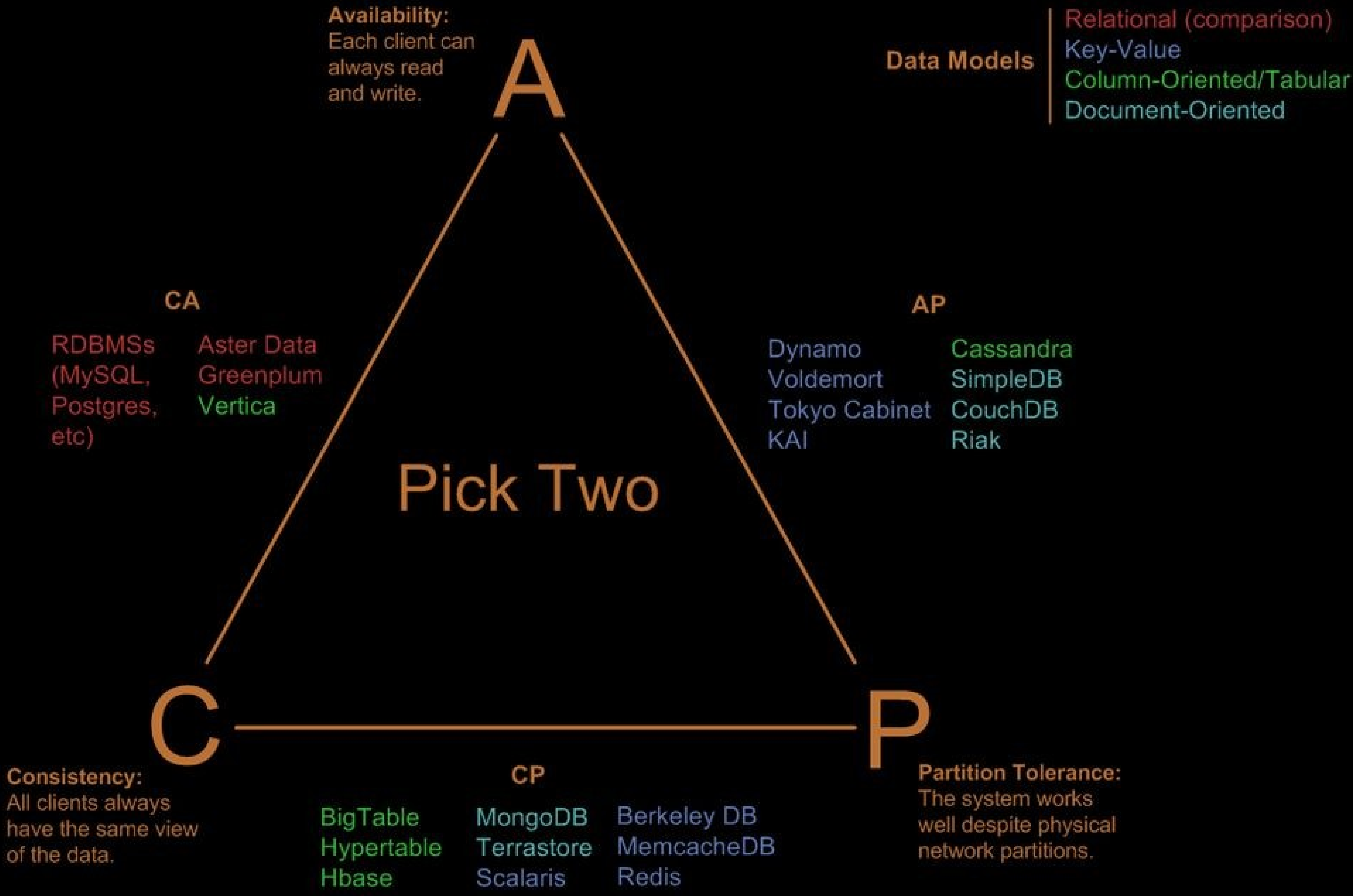
Representatives of NoSQL databases

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

[1]after merging Membase and CouchOne

Visual Guide to NoSQL Systems



Data Models
- Relational (comparison)
- Key-Value
- Column-Oriented/Tabular
- Document-Oriented

Availability: Each client can always read and write.

Consistency: All clients always have the same view of the data.

Partition Tolerance: The system works well despite physical network partitions.