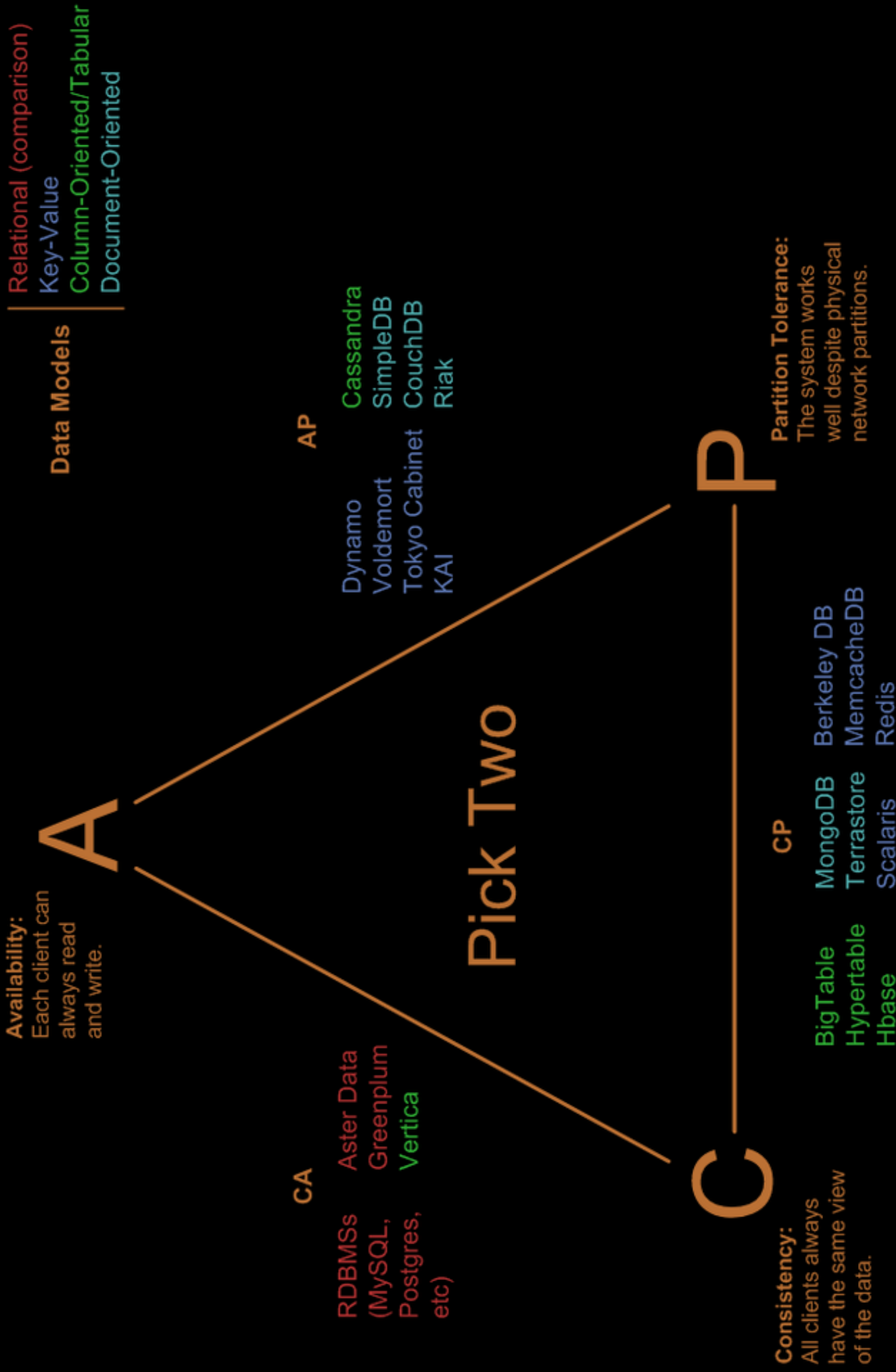


Visual Guide to NoSQL Systems



Visual Guide to NoSQL Systems

<http://blog.nahurst.com/visual-guide-to-nosql-systems>

There are so many NoSQL systems these days that it's hard to get a quick overview of the major trade-offs involved when evaluating relational and non-relational systems in non-single-server environments. I've developed this visual primer with quite a lot of help (see credits at the end), and it's still a work in progress, so let me know if you see anything misplaced or missing, and I'll fix it.

Without further ado, here's what you came here for (and further explanation after the visual).

Note: RDBMSs (MySQL, Postgres, etc) are only featured here for comparison purposes. Also, some of these systems can vary their features by configuration (I use the default configuration here, but will try to delve into others later).

<See visual>

As you can see, there are three primary concerns you must balance when choosing a data management system: consistency, availability, and partition tolerance.

- **Consistency** means that each client always has the same view of the data.
- **Availability** means that all clients can always read and write.
- **Partition tolerance** means that the system works well across physical network partitions.

According to the [CAP Theorem](#), you can only pick two. So how does this all relate to NoSQL systems?

One of the primary goals of NoSQL systems is to bolster horizontal scalability. To scale horizontally, you need strong network partition tolerance which requires giving up either consistency or availability. NoSQL systems typically accomplish this by relaxing relational abilities and/or loosening transactional semantics.

In addition to CAP configurations, another significant way data management systems vary is by the data model they use: relational, key-value, column-oriented, or document-oriented (there are [others](#), but these are the main ones).

- **Relational** systems are the databases we've been using for a while now. RDBMSs and systems that support ACIDity and joins are considered relational.
- **Key-value** systems basically support get, put, and delete operations based on a primary key.
- **Column-oriented** systems still use tables but have no joins (joins must be handled within your application). Obviously, they store data by column as opposed to traditional row-oriented databases. This makes aggregations much easier.
- **Document-oriented** systems store structured "documents" such as JSON or XML but have no joins (joins must be handled within your application). It's very easy to map data from object-oriented software to these systems.

Now for the particulars of each CAP configuration and the systems that use each configuration:

Consistent, Available (CA) Systems have trouble with partitions and typically deal with it with replication. Examples of CA systems include:

- Traditional RDBMSs like Postgres, MySQL, etc (relational)
- Vertica (column-oriented)

- Aster Data (relational)
- Greenplum (relational)

Consistent, Partition-Tolerant (CP) Systems have trouble with availability while keeping data consistent across partitioned nodes. Examples of CP systems include:

- [BigTable](#) (column-oriented/tabular)
- [Hypertable](#) (column-oriented/tabular)
- [HBase](#) (column-oriented/tabular)
- [MongoDB](#) (document-oriented)
- [Terrastore](#) (document-oriented)
- [Redis](#) (key-value)
- [Scalaris](#) (key-value)
- [MemcacheDB](#) (key-value)
- [Berkeley DB](#) (key-value)

Available, Partition-Tolerant (AP) Systems achieve "eventual consistency" through replication and verification. Examples of AP systems include:

- [Dynamo](#) (key-value)
- [Voldemort](#) (key-value)
- [Tokyo Cabinet](#) (key-value)
- [KAI](#) (key-value)
- [Cassandra](#) (column-oriented/tabular)
- [CouchDB](#) (document-oriented)
- [SimpleDB](#) (document-oriented)
- [Riak](#) (document-oriented)

Self promotion and Credits

- If you're a developer and looking for a job or if you're hiring developers and these data systems are important to you, consider coming to [Hirelite: Speed Dating for the Hiring Process](#) on Tuesday in NYC.
- This guide draws heavily from a recent [Ruby](#) meetup (by Matthew Jording and Michael Bryzek) and a recent [MongoDB](#) presentation (given by Dwight Merriman).
- Thanks to [DBNess](#) and [ansonism](#) for their help with validating system categorizations.
- Thanks to those who helped shape the post after it was written: [Stan](#), [Dwight](#), and others who commented here and on this [Hacker News thread](#).