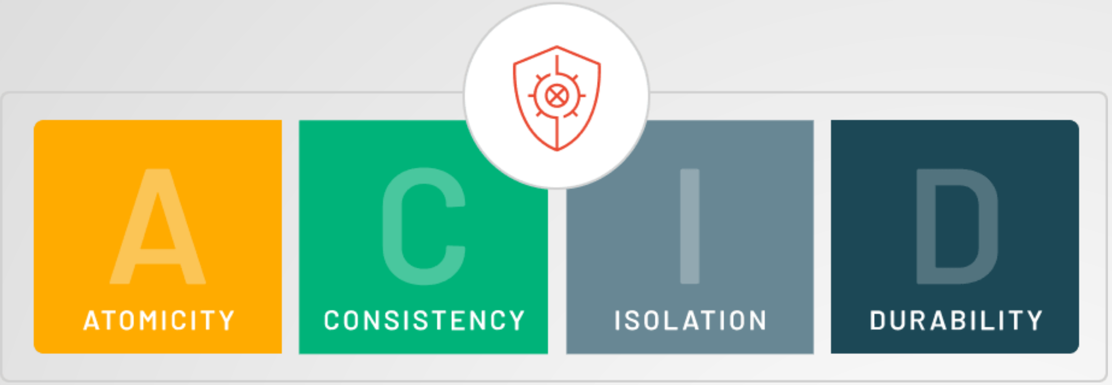

A
ATOMICITY
C
CONSISTENCY
I
ISOLATION
D
DURABILITY

BASICALLY AVAILABLE
SOFT STATE
BASE
EVENTUALLY CONSISTENT

EVENTUAL
CONSISTENCY

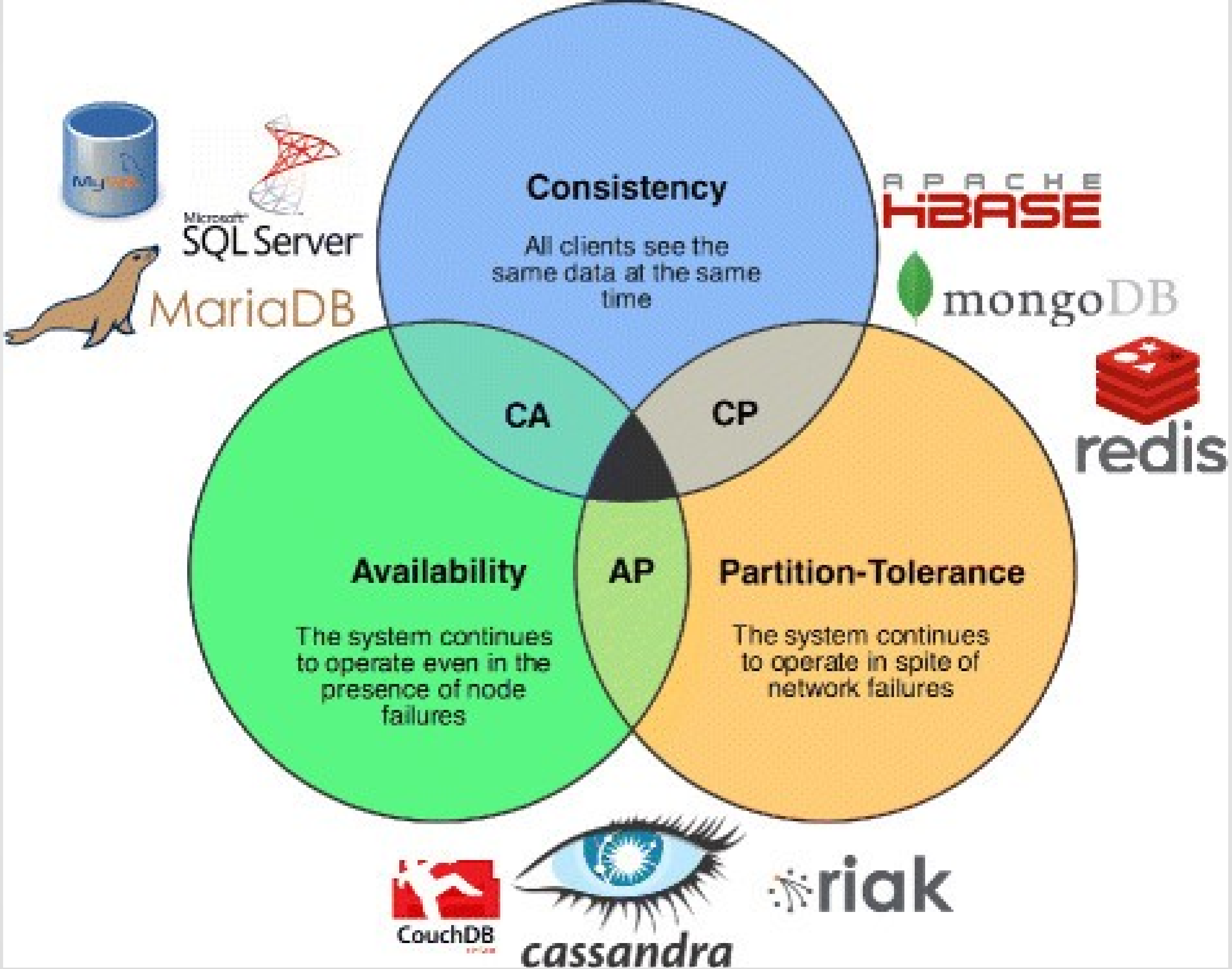
CAP - Theorem

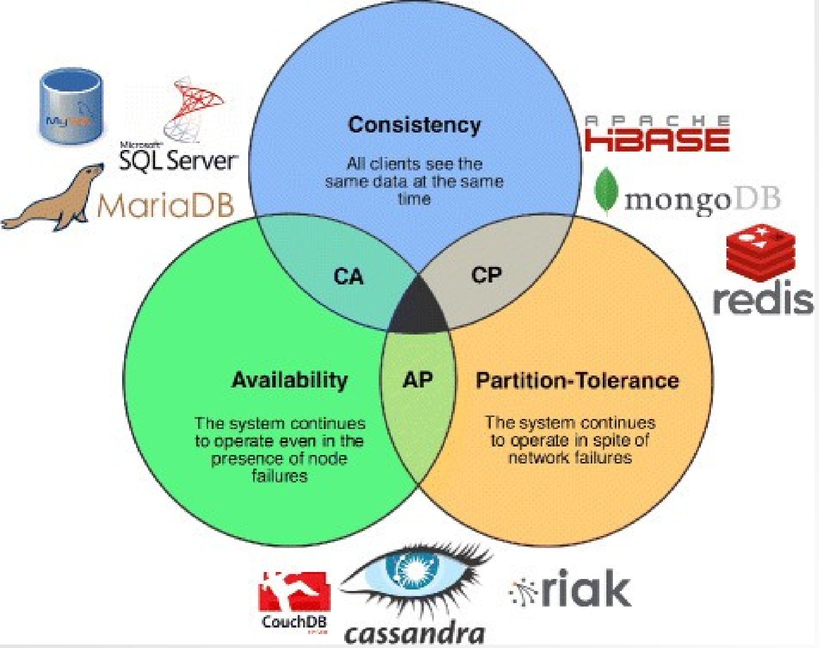
...or Brewer's theorem

- **Consistency**
  Every client views the same data. Every node in distributed cluster receives the same data.
- **Availability**
  During node failure time, every node must respond in a reasonable amount of time.
- **Partition tolerance**
  Gurantees partition tolerance can recover from partitions once the partition heals.

The theorem is...

You may only pick two!

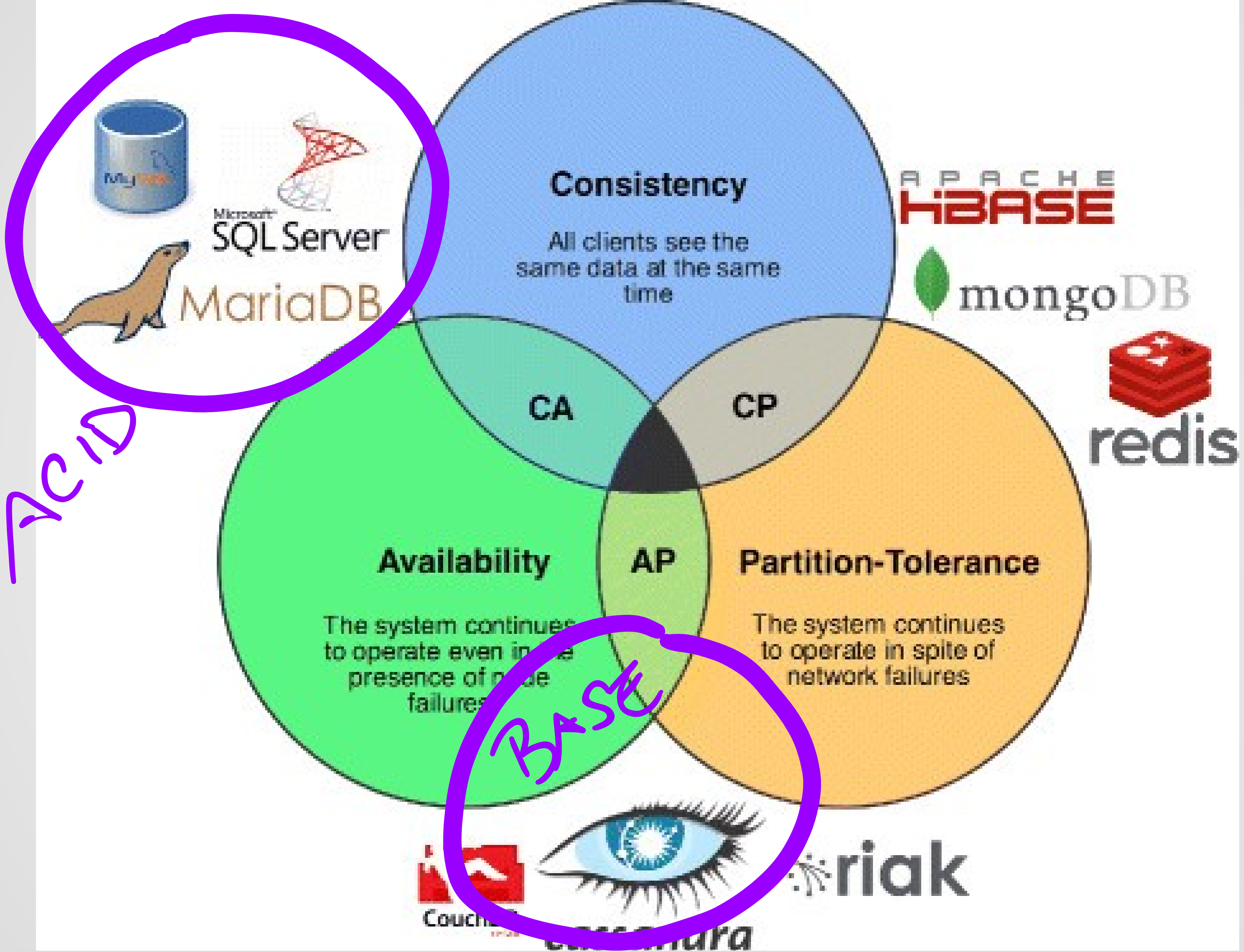

Consistency
All clients see the same data at the same time
CA
CP
AP
Availability
The system continues to operate even in the presence of node failures
Partition-Tolerance
The system continues to operate in spite of network failures
MySQL
Microsoft SQL Server
MariaDB
APACHE HBASE
mongoDB
redis
CouchDB
cassandra
riak


Consistency
All clients see the same data at the same time
CA
CP
AP
Availability
The system continues to operate even in the presence of node failures
Partition-Tolerance
The system continues to operate in spite of network failures
SQL Server
MariaDB
HBASE
mongoDB
redis
CouchDB
cassandra
riak

AP (Availability and Partition tolerance): When availability is chosen over consistency, the system will always process the client request and try to return the most recent available version of the information even if it cannot guarantee it is up to date due to network partitioning.

CA (Consistency and Availability): The correct value is delivered at all times, but no resilience to partitioning at all. Kafka is CA.

CP (Consistency and Partition tolerance): When consistency is chosen over availability, the system will return an error or time-out if particular information cannot be updated to other nodes due to network partition or failures.


Consistency
All clients see the same data at the same time
CA
CP
AP
Availability
The system continues to operate even in the presence of node failures
Partition-Tolerance
The system continues to operate in spite of network failures
MySQL
Microsoft SQL Server
MariaDB
APACHE HBASE
mongoDB
redis
riak
Couch
cassandra
ACID
BASE

ACID

Atomicity

Consistency

Isolation

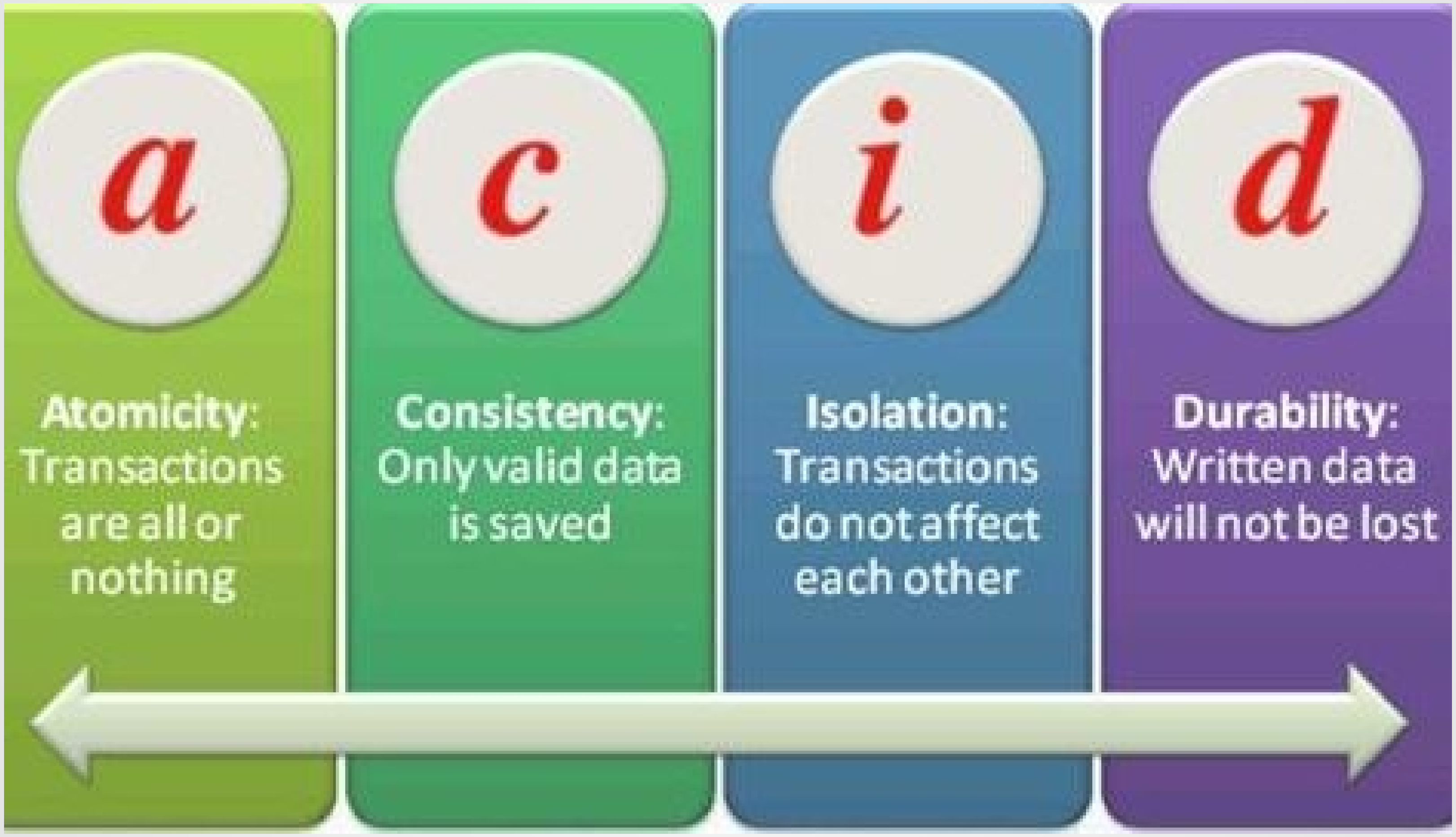
Durability

ACID

In 1983, Andreas Reuter and Theo Härder coined the acronym *ACID*, building on earlier work by Jim Gray.

It describes the four properties that are the major guarantees of the transaction paradigm.

According to Gray and Reuter, the IBM Information Management System supported ACID transactions as early as 1973 (although the acronym was created later).



Atomicity: Transactions are all or nothing

Consistency: Only valid data is saved

Isolation: Transactions do not affect each other

Durability: Written data will not be lost

Atomicity

An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes.

Transactions consist of statements and atomicity guarantees that a transaction is either fully committed or not committed at all.

Consistency

Consistency ensures that a transaction can only bring the database from one valid state to another.

Isolation

Transactions are often executed concurrently (e.g., multiple transactions reading and writing to a table at the same time).

Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially.

Isolation

Isolation is the main goal of concurrency control; depending on the method used, the effects of an incomplete transaction might not even be visible to other transactions.

Durability

Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure (e.g., power outage or crash).

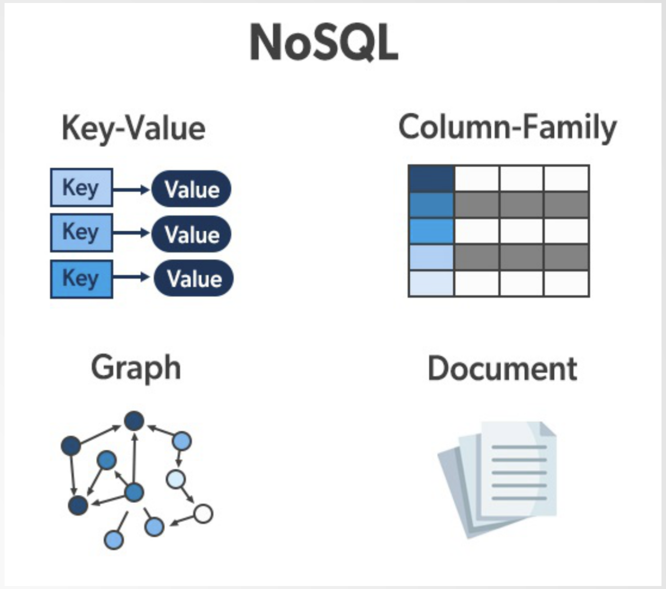
This usually means that completed transactions (or their effects) are recorded in non-volatile memory.

BASE

Basic **A**vailability

Soft-state

Eventually consistent



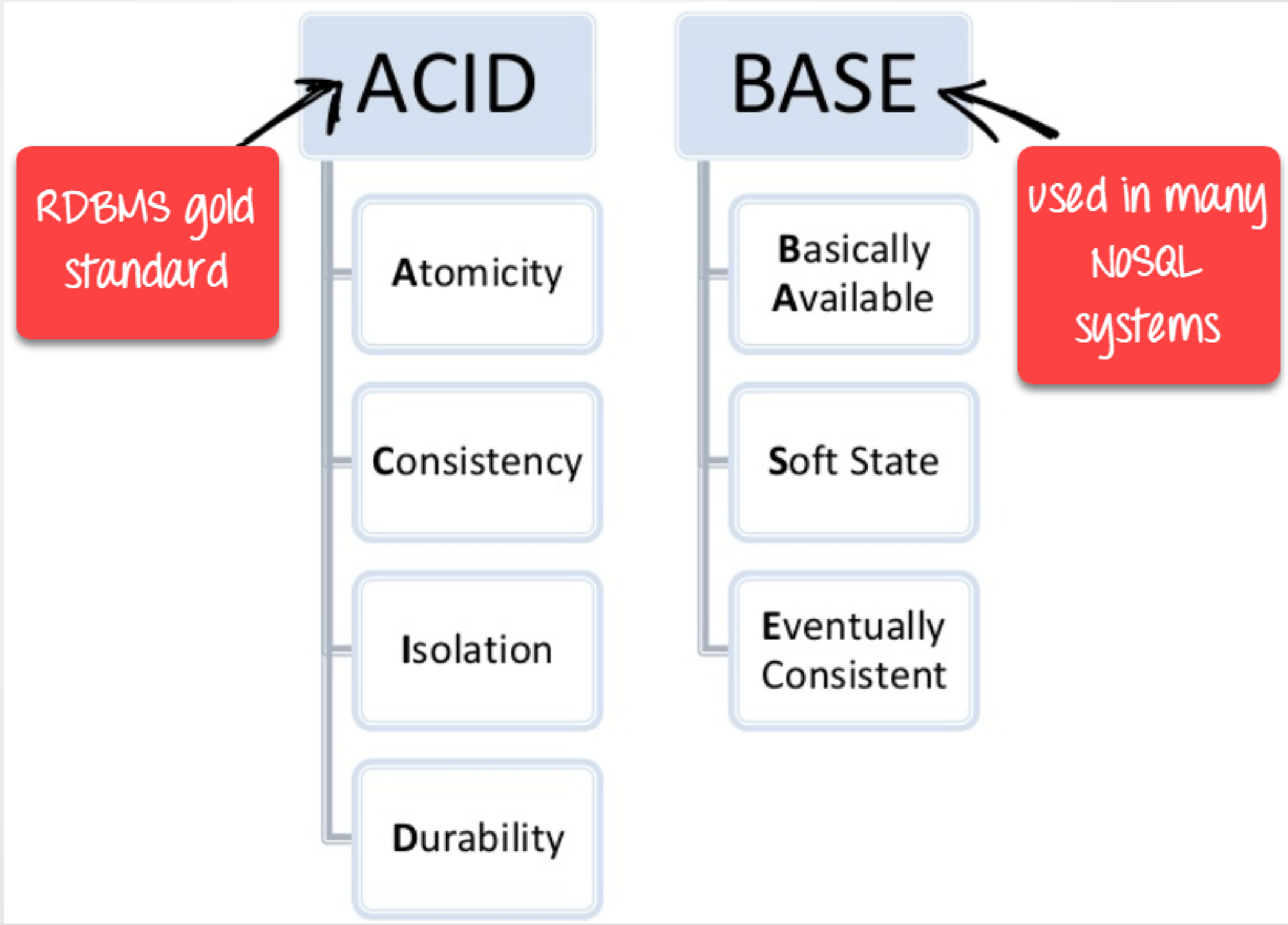
DynamoDB,
Cassandra,
CouchDB,
SimpleDB

Eventual Consistency

Eventual consistency is a consistency model used in distributed computing to achieve high availability.

It informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

BASE


ACID
Atomicity
Consistency
Isolation
Durability
BASE
Basically Available
Soft State
Eventually Consistent
RDBMS gold standard
used in many NoSQL systems

Basically Available

Reading and writing operations are available as much as possible (using all nodes of a database cluster), but might not be consistent.

(The write might not persist after conflicts are reconciled, and the read might not get the latest write.)

Soft-State

Without consistency guarantees, after some amount of time, we only have some probability of knowing the state, since it might not yet have converged.

Eventually Consistent

If we execute some writes and then the system functions long enough, we can know the state of the data.

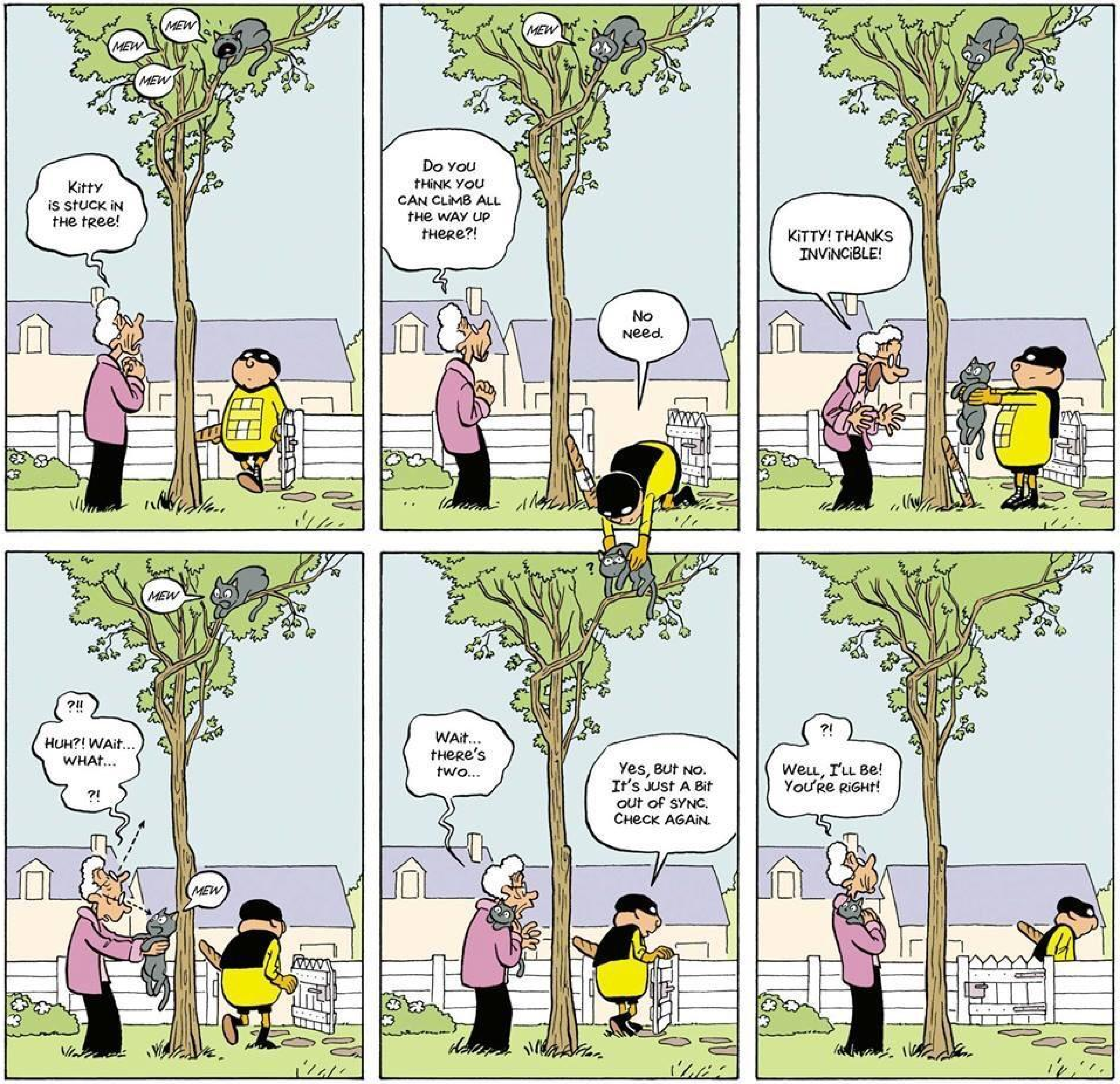
Any further reads of that data item will return the same value.

It is purely a liveness guarantee (reads eventually return the same value) and does not guarantee safety.

An eventually consistent system can return **any value before it converges**.

Eventual Consistency

Example


MEW
MEW
MEW
Kitty is stuck in the tree!
MEW
Do you think you can climb all the way up there?!
No need.
Kitty! Thanks Invincible!
MEW
?!!
Huh?! Wait... what...
?!
MEW
?
Wait... there's two...
Yes, but no. It's just a bit out of sync. Check again.
?!
Well, I'll be! You're right!

References

- https://www.techopedia.com/definition/29165/eventual-consistency
- https://en.wikipedia.org/wiki/ACID
- https://en.wikipedia.org/wiki/Eventual_consistency
- https://systemdesignbasic.wordpress.com/2020/06/20/15-acid-vs-base-database-transactions/
- https://www.linkedin.com/pulse/what-acid-properties-database-aseem-jain/
- https://en.wikipedia.org/wiki/CAP_theorem
- https://www.debadityachakravorty.com/system_design/captheorem/