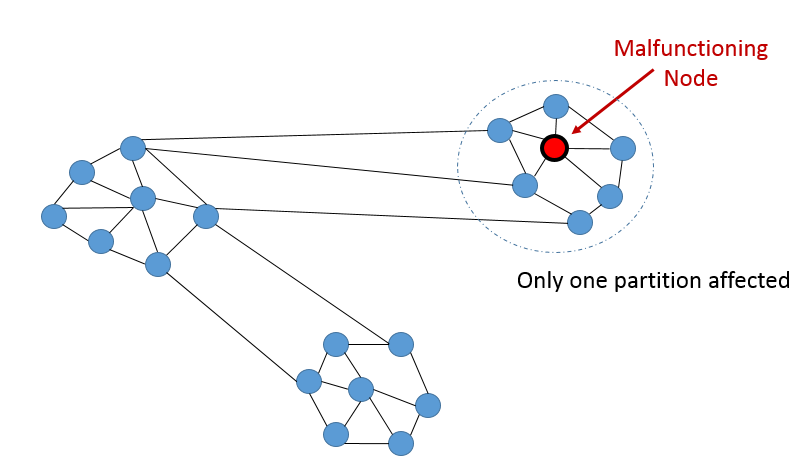


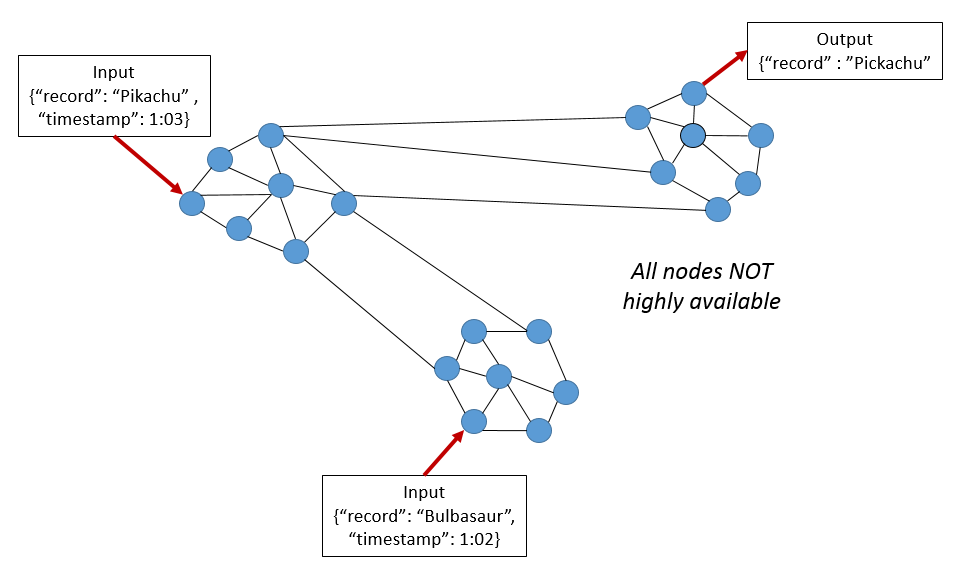
**Partition Tolerance Explained below**

This condition states that the system continues to run, despite the number of messages being delayed by the network between nodes. A system that is partition-tolerant can sustain any amount of network failure that doesn’t result in a failure of the entire network. Data records are sufficiently replicated across combinations of nodes and networks to keep the system up through intermittent outages. When dealing with modern distributed systems, Partition Tolerance is not an option. It’s a necessity. Hence, we have to trade between Consistency and Availability.

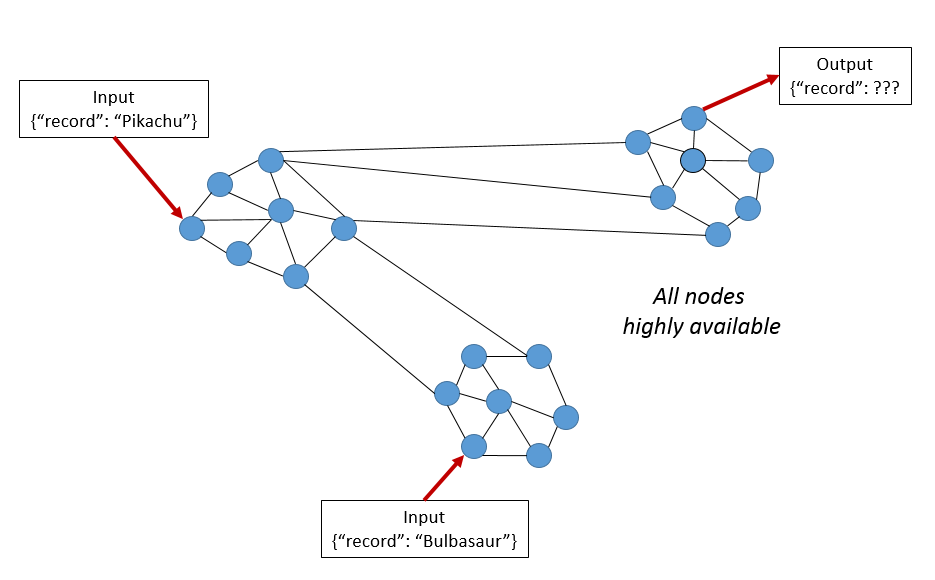
Partition Tolerance

**High Consistency**

This condition states that all nodes see the same data at the same time. Simply put, performing a *read* operation will return the value of the most recent *write* operation causing all nodes to return the same data. A system has consistency if a transaction starts with the system in a consistent state, and ends with the system in a consistent state. In this model, a system can (and does) shift into an inconsistent state during a transaction, but the entire transaction gets rolled back if there is an error during any stage in the process. In the image, we have 2 different records (“Bulbasaur” and “Pikachu”) at different timestamps. The output on the third partition is “Pikachu”, the latest input. However, the nodes will need time to update and will not be Available on the network as often



**High Availability:**



This condition states that every request gets a response on success/failure. Achieving availability in a distributed system requires that the system remains operational 100% of the time. Every client gets a response, regardless of the state of any individual node in the system. This metric is trivial to measure: either you can submit read/write commands, or you cannot. Hence, the databases are time independent as the nodes need to be available online at all times. This means that, unlike the previous example, we do not know if “Pikachu” or “Bulbasaur” was added first. The output could be either one. Hence why, high availability isn’t feasible when analyzing streaming data at high frequency.

**Self Notes :**

Databases structured around Master Slave arrangement (Not to confuse Master slave as just one-to-one , they also have multiple slaves or possibly multiple masters , so replication is there) provide Consistency as a top feature and then comes Partition Tolerance . The moment a slave becomes unavailable or a master is down, another node is appointed a master and it takes over but then this results in UnAvailability hence they all stick around the CP box.

Databases structured around the cluster setup or a Ring Setup where the data is evenly distributed and replicas are stored for each Node on other nodes provide Availability as top feature. The moment a data has copy on different nodes will result in InConsistency but having multiple copies also ensures against Network failures within a cluster hence this falls into AP category.

*Availability in CAP theorem is not the same as the downtime we talk about in our day to day system. Example 99.9% availability of a microservice is not the same as CAP theorem Availability. CAP-Availibilty talks about if the cluster has network partition how the system will behave, whether it will start giving error or keep serving requests successfully.*

So, In simple words,**CAP theorem** means if there is network partition and if you want your system to keep functioning you can provide either **Availability** or **Consistency** and not both.

How is RDBMS Consistent and Available :

* Consistency is understood with RDBMS .
* RDBMS generally are single Node as in one Server which serves reads and writes hence there is NO network partition which makes it CAP Available. As in if the Server is up , it will serve requests.
* In a leader and slave setup , if leader is down , it takes some time to get Leader back up which results in UNAvailability which is what MongoDB is categorized in.

Very good article on Medium :

<https://medium.com/@bikas.katwal10/mongodb-vs-cassandra-vs-rdbms-where-do-they-stand-in-the-cap-theorem-1bae779a7a15>