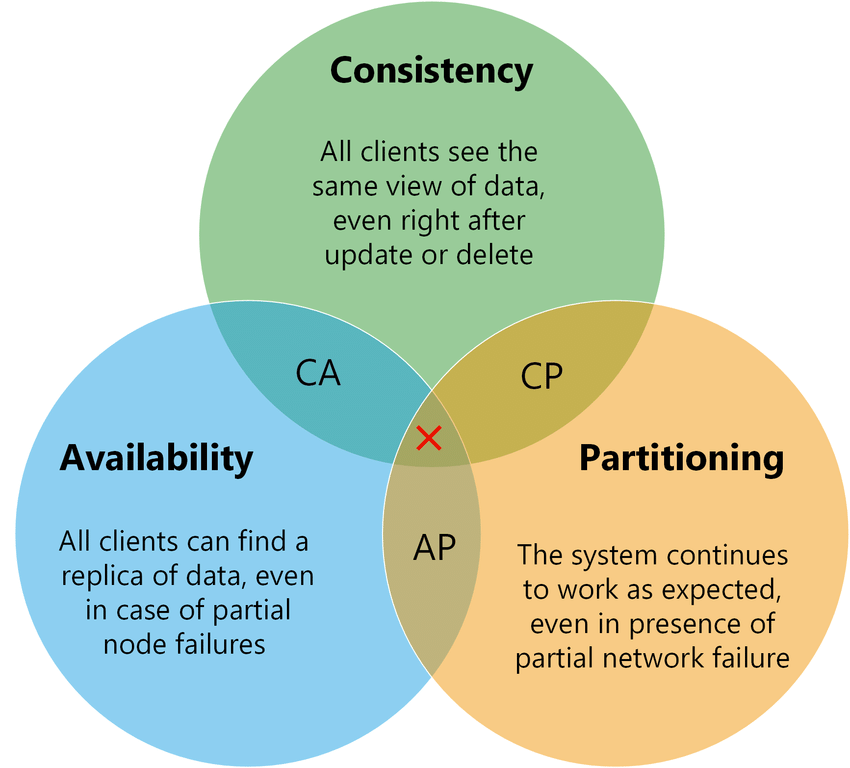
# What is CAP theorem?

The CAP theorem suggests that, at best, any distributed system can only satisfy CP (Consistency & Partition Tolerance), AP (Availability & Partition Tolerance), or somewhere between the two. As a consequence, interesting tradeoffs between consistency and availability arise.

The traditional “*choose 2 of 3*" argument doesn’t make sense. You cannot give up partition tolerance, because that would mean “behavior of the operations performed during the partition is undefined”, and in that case the database isn’t really consistent.

Reaching the limit of the CAP theorem is not given by default. There are many databases which are neither consistent, available, nor partition tolerant. Achieving the limits of the CAP theorem requires careful design and implementation.



## Consistency

Consistency implies the following:

* When data is partitioned (distributed), all the nodes see the same data at a given time, and this should be true for all times
* When queried, each node will return the latest data. If not, the system will just error out.
* Consistency is achieved by updating several nodes before allowing further reads.

## Availability

Availability implies the following:

* At all times, every request being fired at the system generates a valid response.
* While doing this, it doesn't mean that every request will receive a response with the latest information (data). Availability is achieved by replicating the data across different servers.

## Partition Tolerance

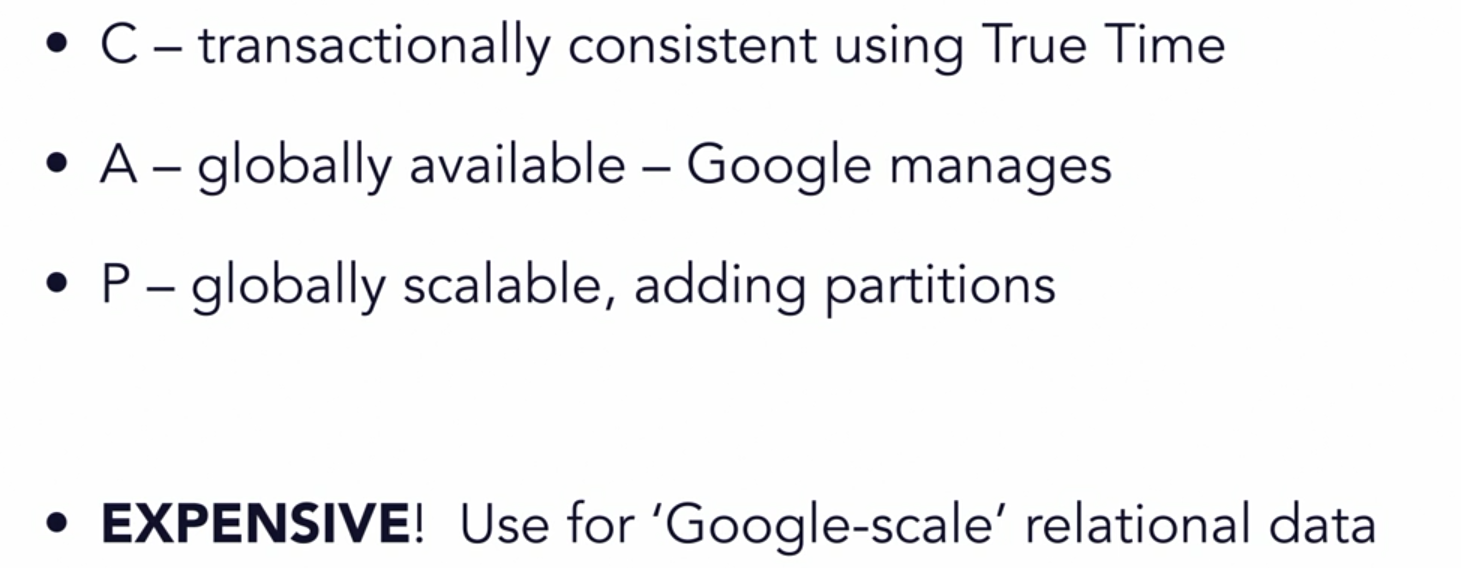
Partition tolerance implies the following:

* System continues to work despite message loss or partial failure. A system that is partition-tolerant can sustain any amount of network failure that doesn’t result in a failure of the entire network.
* Partition tolerance can be achieved by replicating data and system functionality sufficiently across a cluster of nodes and network. The redundancy introduced ensures the system as a whole continues to function even in situations where a node or a set of nodes cannot communicate with each other.

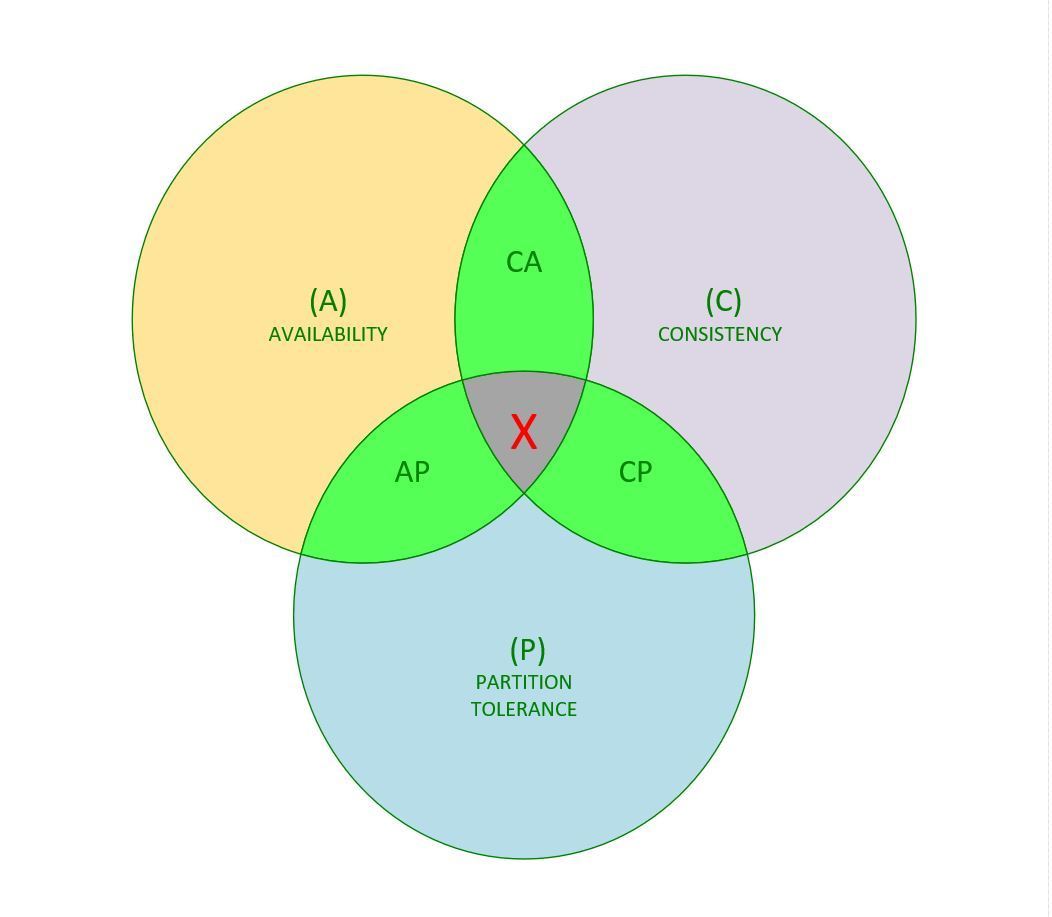
# System classification based on CAP theorem

Because only two of the three properties stated by CAP can be guaranteed at any time, systems are usually classified into three types under CAP Theorem:

* ****CA System****: Data is consistent between all nodes, and you can read/write from any node, while you cannot afford to let your network go down. (For example: RDBMS like MSSQL Server, Oracle and columnar relational stores) (一旦分区了，因为由于要包A，那么C无法实现了)
* ****CP System****: Data is consistent and maintains tolerance for partitioning and preventing data going out of sync. (For example: Google Big Table, MongoDB (document oriented), and HBase (columnar)) （一旦分区了，由于要保C，那么A就无法实现了）
* ****AP System****: Nodes are online always, but they may not get you the latest data; however, they sync whenever the lines are up. (For example: CouchDB (document oriented), and Cassandra (columnar)) （一旦分区了，可以去读旧数据，反正不要C）
* CAP System : Google Spanner



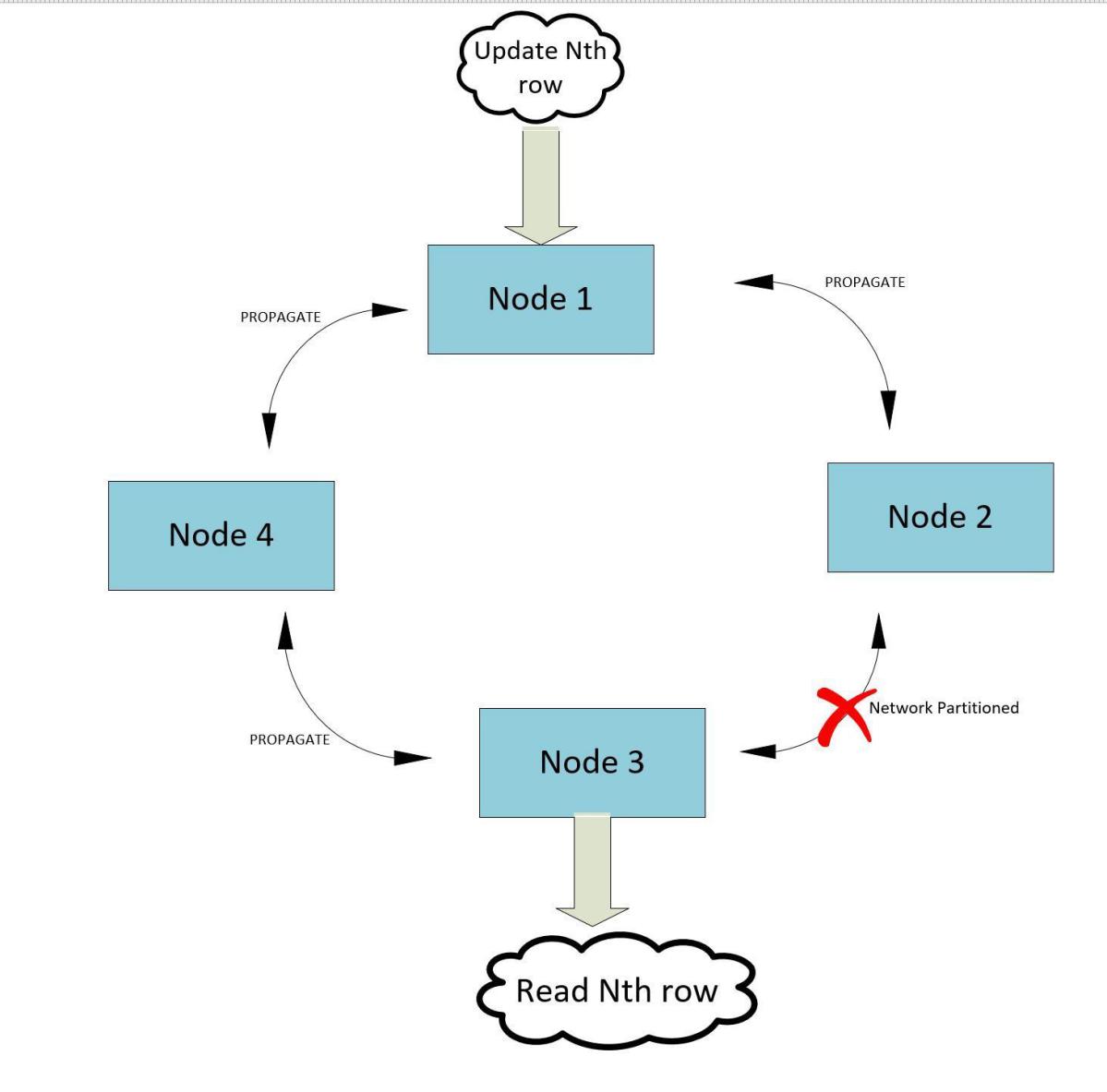
We cannot build a general data store that is continually available, sequentially consistent and tolerant to any partition failures. We can only build a system that has any two of these three properties. Because, to be consistent, all nodes should see the same set of updates in the same order. But if the network suffers a partition, updates in one partition might not make it to the other partitions before a client reads from the out-of-date partition after having read from the up-to-date one. The only thing that can be done to cope with this possibility is to stop serving requests from the out-of-date partition, but then the service is no longer 100% available.

[](https://i0.wp.com/www.acodersjourney.com/wp-content/uploads/2018/07/CAP-Theorem.jpg)

System Classification Based on CAP Theorem

# CAP theorem in action

**In real life distributed systems, network partition will happen and we need to ensure that when network partition does happen, the system as a whole is still functional. So, really, our choice boils down to either Consistency Guarantee or Availability guarantee in most cases.**

[](https://i1.wp.com/www.acodersjourney.com/wp-content/uploads/2018/07/CAP-Theorem-Network-Partition.jpg)

CAP Theorem in action during network partition

           A row is updated in ****Node 1**** as shown in the preceding figure. Before ****Node 3**** receives the update propagation, it gets a request for the same row. Now ****Node 3**** has two choices--either provide the copy of the row it has (which may be stale) or deny the request. In the first case, it is compromising on consistency and in the second case, on availability.

# How to use CAP theorem during system design interview?

As you saw in the previous section, different types of data stores provides different types of guarantees - some provide consistency guarantees while some provide availability guarantees.

During a system design interview, you should always clarify what type of guarantee the interviewer is looking for in the system under discussion. For example, if you're building a news feed like twitter- it might have the following requirements:

* Highly available
* Consistency can take a hit in favor of availability, if a news feed does not show up for a little while, it should be fine.

With the above requirements, you immediately know that you need an AP system, that is, a system which offers availability during a network partition. So you can choose a NoSQL solution like Cassandra or Dynamo DB.

For bonus points during the interview, you can use the following data from [Greg Linden's blog](https://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html" \t "/home/gnuhpc/Documents\\x/_blank):

*" Amazon found every 100ms of latency cost them 1% in sales. "*

##### *" Google reports half a second delay caused a 20% drop in traffic."*

Please find below a handy cheat sheet for choosing the right data store for your system.

### AP Systems

* Dynamo DB
* Voldemort
* Cassandra
* SimpleDB
* CouchDB
* Riak

### CP Systems

* Google BigTable
* Hbase
* MongoDB
* MemcacheDB
* Redis

### CA Systems

* RDBMS( MySQL, MSSQL Server)

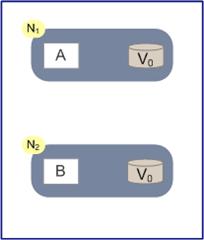
# Final thoughts on CAP theorem

Each one of the three properties, namely,***Availability, Consistency and Partition Tolerance***, should not be viewed as a binary off/on switch , but rather as tunable parameters when you're designing a distributed system. That is, if you opt for more consistency, you'll need to make your availability or partition tolerance requirements little lax. Conversely, you can tune up your availability if you are prepared to sacrifice some consistency or network partition tolerance.

# **CAP定理的证明**

关于CAP这三个特性我们就介绍完了，接下来我们试着证明一下**为什么CAP不能同时满足**。

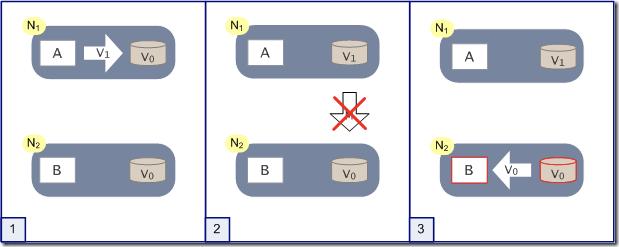
为了简化证明的过程，我们假设整个集群里只有两个N1和N2两个节点，如下图：



N1和N2当中各自有一个应用程序AB和数据库，当系统满足一致性的时候，我们认为N1和N2数据库中的数据保持一致。在满足可用性的时候，我们认为无论用户访问N1还是N2，都可以获得正确的结果，在满足分区容错性的时候，我们认为无论N1还是N2宕机或者是两者的通信中断，都不影响系统的运行。

我们假设一种极端情况，假设某个时刻N1和N2之间的**网络通信突然中断**了。如果系统满足分区容错性，那么显然可以支持这种异常。问题是在此前提下，一致性和可用性是否可以做到不受影响呢？

我们做个假象实验，如下图，突然某一时刻N1和N2之间的关联断开：



有用户向N1发送了请求更改了数据，将数据库从V0更新成了V1。由于网络断开，所以N2数据库依然是V0，如果这个时候有一个请求发给了N2，但是N2并没有办法可以直接给出最新的结果V1，这个时候该怎么办呢？

这个时候无法两种方法，**一种是将错就错，将错误的V0数据返回给用户。第二种是阻塞等待，等待网络通信恢复，N2中的数据更新之后再返回给用户**。显然前者牺牲了一致性，后者牺牲了可用性。