CAP Theorem

CAP Theorem

21 years ago, Eric Brewer introduced the idea that there is a **fundamental tradeoff** between ***consistency, availability, and network partition tolerance***. *This tradeoff, known as the CAP theorem, has been widely discussed ever since.*

CAP Theorem

- *Consistency* - all nodes should see the same data at the same time
- *Availability* - node failures do not prevent survivors from continuing to operate
- *Partition-tolerance* - the system continues to operate despite arbitrary message loss

CAP Theorem

A distributed system can satisfy any two of CAP guarantees at the same time but not all three:

- *Consistency + Availability*
- *Consistency + Partition Tolerance*
- *Availability + Partition Tolerance*

Eric Brewer at ACM Symposium on Principles of Distributed Computing 2000

CAP Background

CAP derives from the fact that it illustrates a general tradeoff in distributed computing: the impossibility of guaranteeing both safety and liveness in an unreliable distributed system.

CAP Background

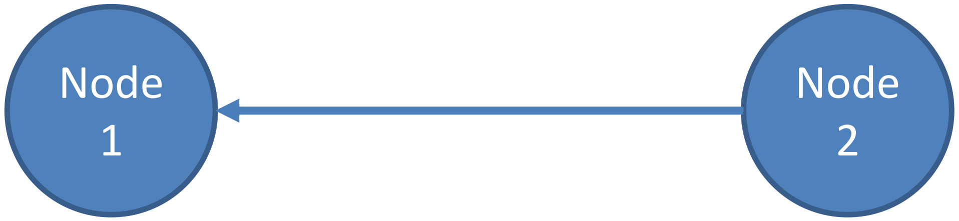
Informally, an algorithm is **safe** if nothing bad ever happens.

Consistency as defined in CAP is a classic **safety** property: every response sent to a client is correct.

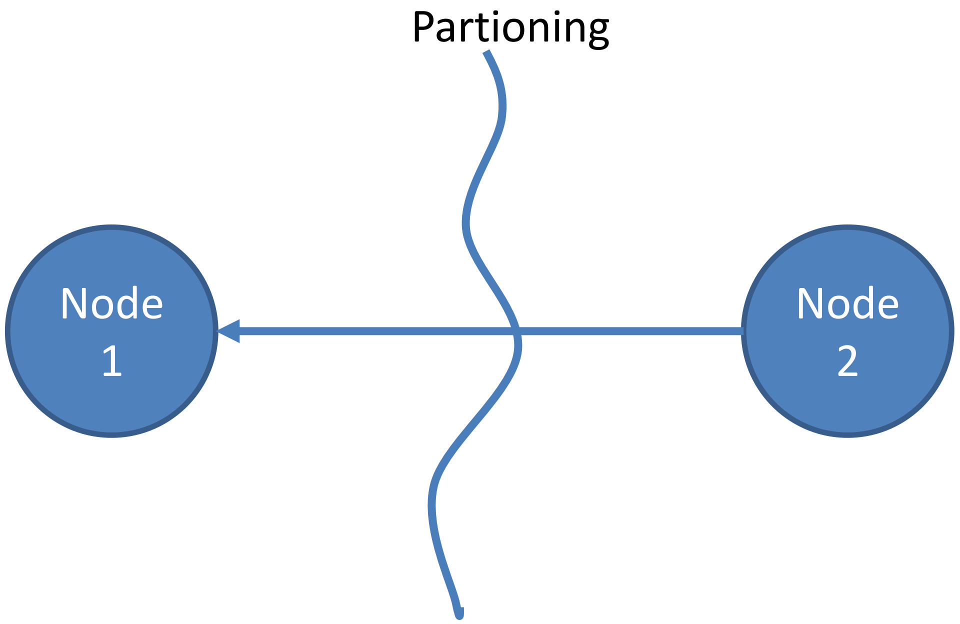
By contrast, an algorithm is **live** if something good eventually happens.

Availability is a classic **liveness** property: every request eventually receives a response.

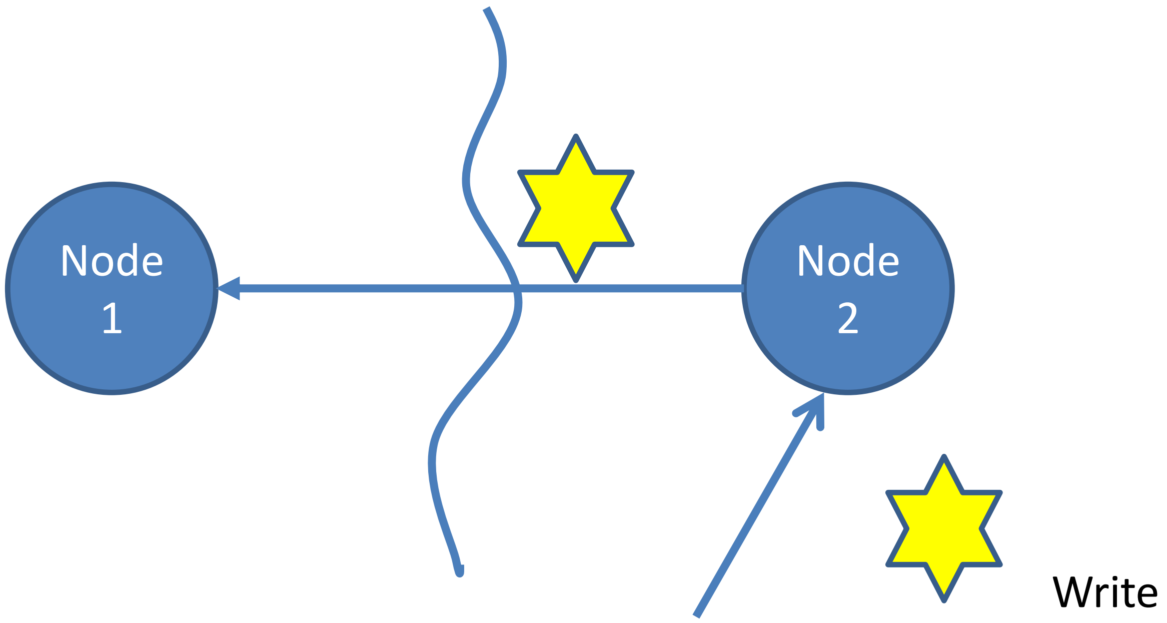
Avaliability & Partionig



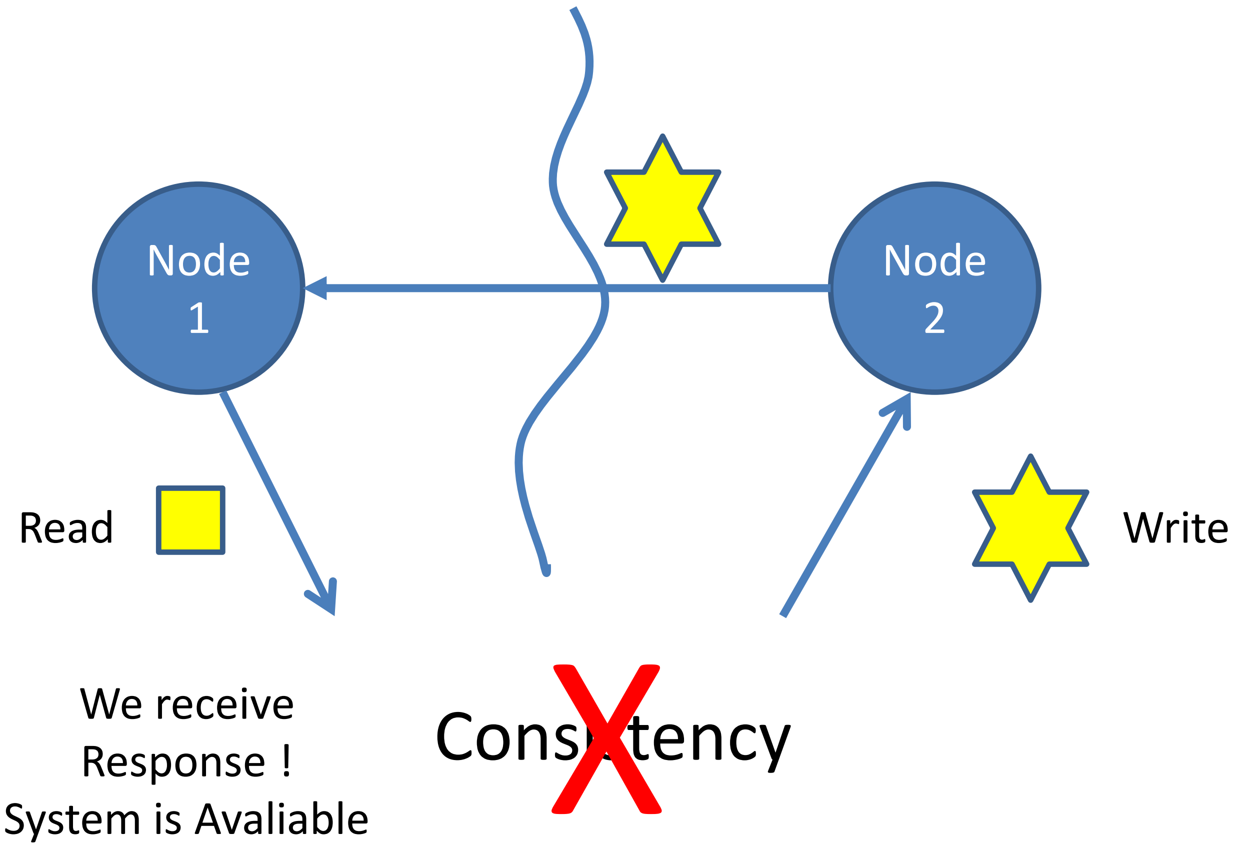
Avaliability & Partionig



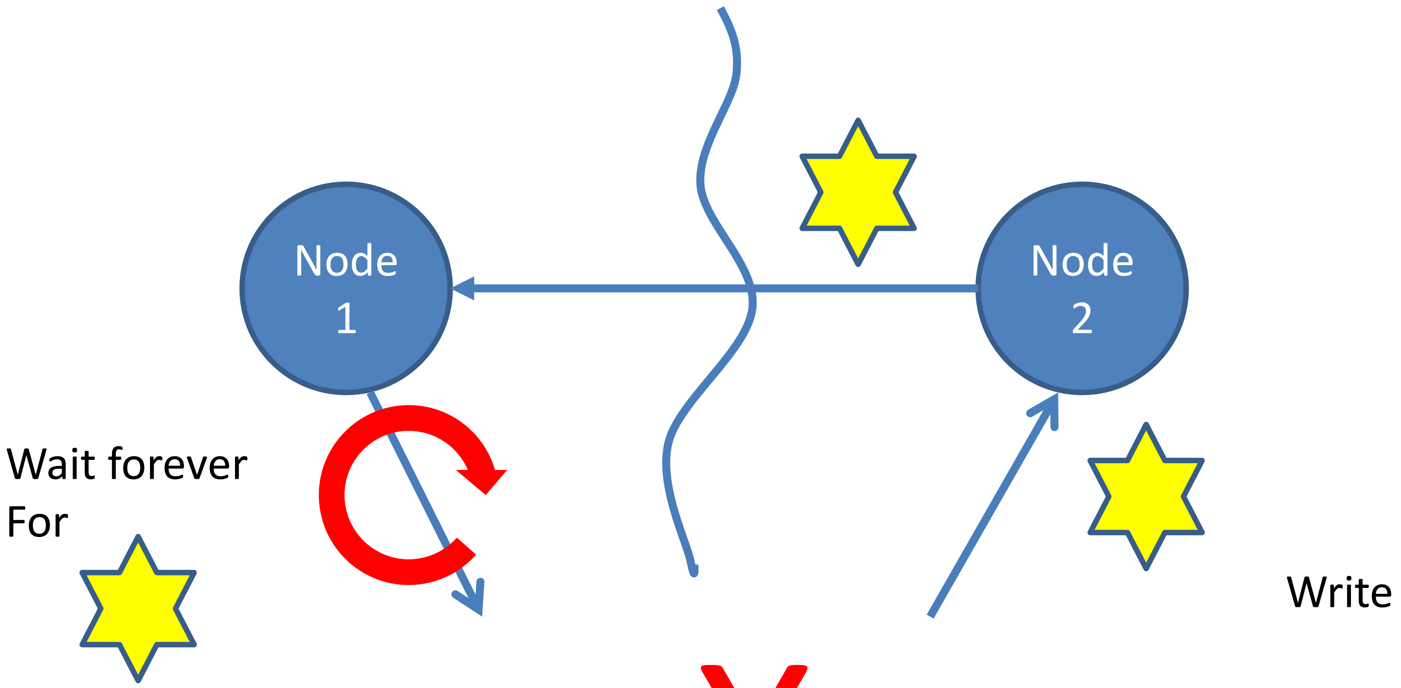
Avaliability &Partionig



Avaliability & Partionig

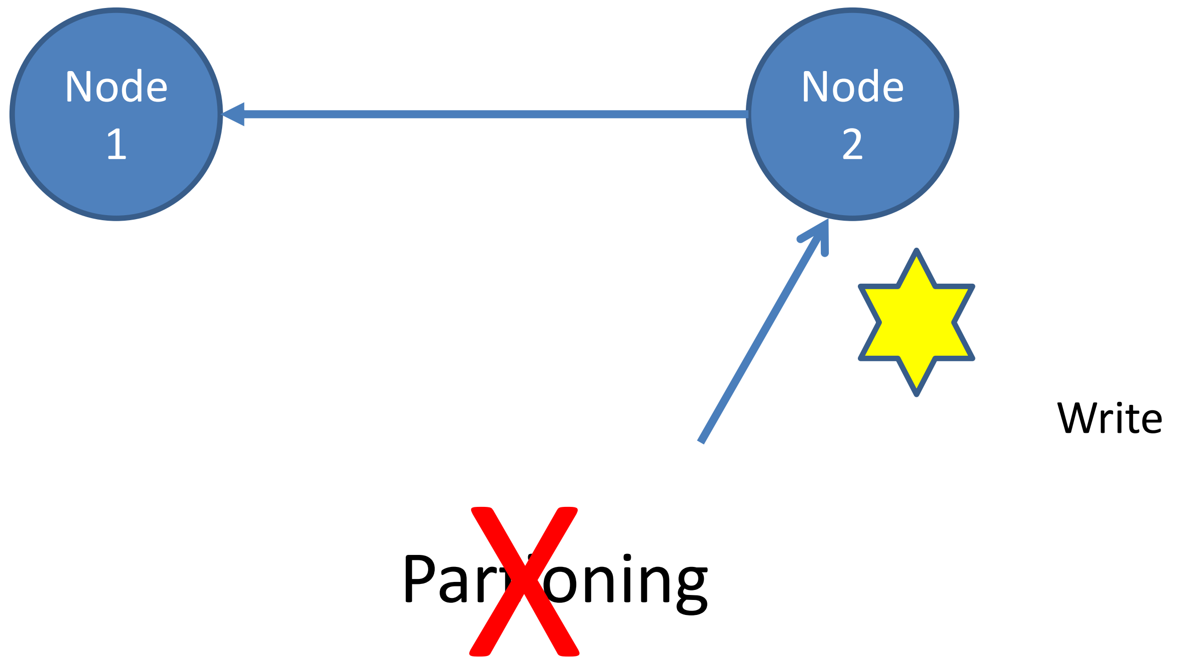


Consitency & Partionig

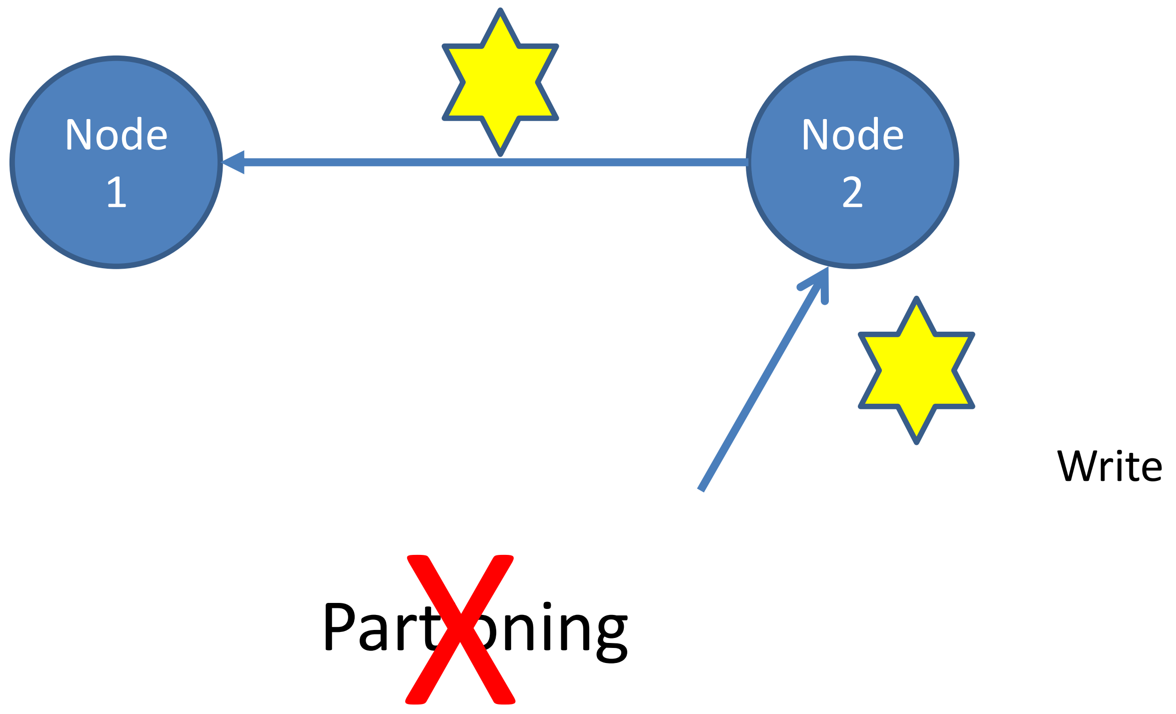


~~Avaliability~~

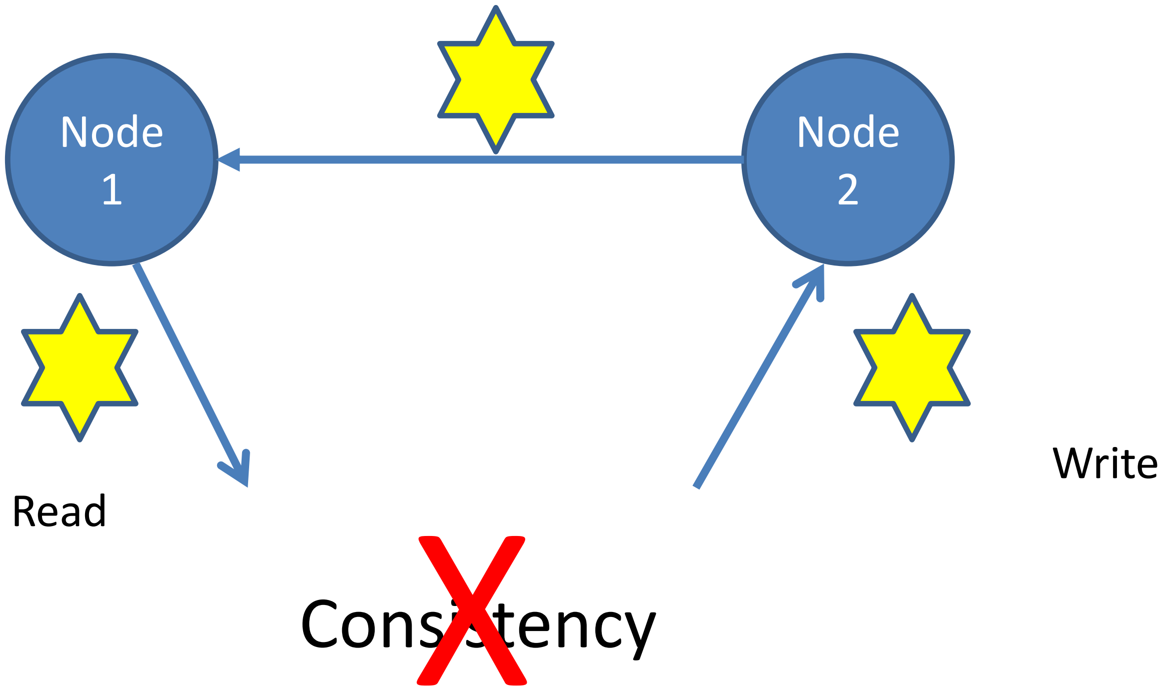
Avaliability Consistency



Avaliability Consistency



Avaliability Consistency



CAP Background

Finally, a system can be ***unreliable*** in many ways, experiencing crash failures, message loss, malicious attacks, Byzantine failures, and so on.

What is CAP?

CAP is simply one example of this tradeoff between **consistency and availability in unreliable systems**.

CAP Context

Brewer first presented CAP in the context of a Web service implemented by a set of servers distributed over a set of geographically diverse datacenters.

Clients issue requests to the service, which sends back responses. This notion of a Web service is intentionally abstract and can embrace a wide variety of applications including search engines, e-commerce, online music services, and cloud-based data storage.

CAP terminology

Consistency.

The first part of CAP refers to consistency, which is, informally, the property that each server returns the right response to each request, that is, a response that is appropriate to the desired service specification. The exact meaning of consistency depends on the type of service.

Consistency Meaning

Trivial services do not require any coordination among the servers

Weakly consistent services involve some distributed coordination, but each server can make progress on its own.

Atomic services are defined in terms of atomic operations, which are described by a sequential specification. A sequential specification describes a service in terms of its execution on a single, centralized server.

Complicated services either cannot be specified sequentially or require more intricate coordination, transactional semantics, and so on.

CAP terminology

Availability

The second part of CAP refers to availability, which is, informally, the property that each request eventually receives a response.

CAP terminology

Partition tolerance

The third part of CAP refers to partition tolerance. Unlike the other two requirements, partition tolerance is really a statement about the underlying system rather than the service itself: communication among the servers is unreliable, and the servers can be partitioned into multiple groups that cannot communicate with one another. We model a partition-prone system as one that is subject to faulty communication: messages can be delayed and sometimes lost forever.

Proof of the CAP Theorem

Lynch, Nancy, and Seth Gilbert. „Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services”.
ACM SIGACT News, v. 33 issue 2, 2002, p. 51-59.

Proving CAP

There is a relatively simple proof of the theorem. Assume the service consists of servers *p1, p2, ..., pn,* along with an arbitrary set of clients.

Consider an execution in which the servers are partitioned into two disjoint sets: ***{p1}*** *and* ***{p2, ..., pn}***. Some client sends a read request to server p2. Because p1 is in a different component of the partition from p2, the **system loses every message from p1 to p2.**

Proving CAP

Now consider the following two cases:

1. A previous write of value *v1 has been requested of p1, and p1 has sent an ok response.*
2. A previous write of value *v2 has been requested of p1, and p1 has sent an ok response.*

No matter how long *p2 waits, it cannot distinguish these two cases*, *hence it cannot determine whether to return response v1 or v2.*

CAP Consequence

It is p2 choice is to either **eventually return** a (possibly wrong) response or to never return a response.

In fact, if communication is asynchronous—if processes have no a priori bound on how long it takes to deliver a message—then the same situation can occur even in executions in which there are no permanent partitions and no messages are lost.

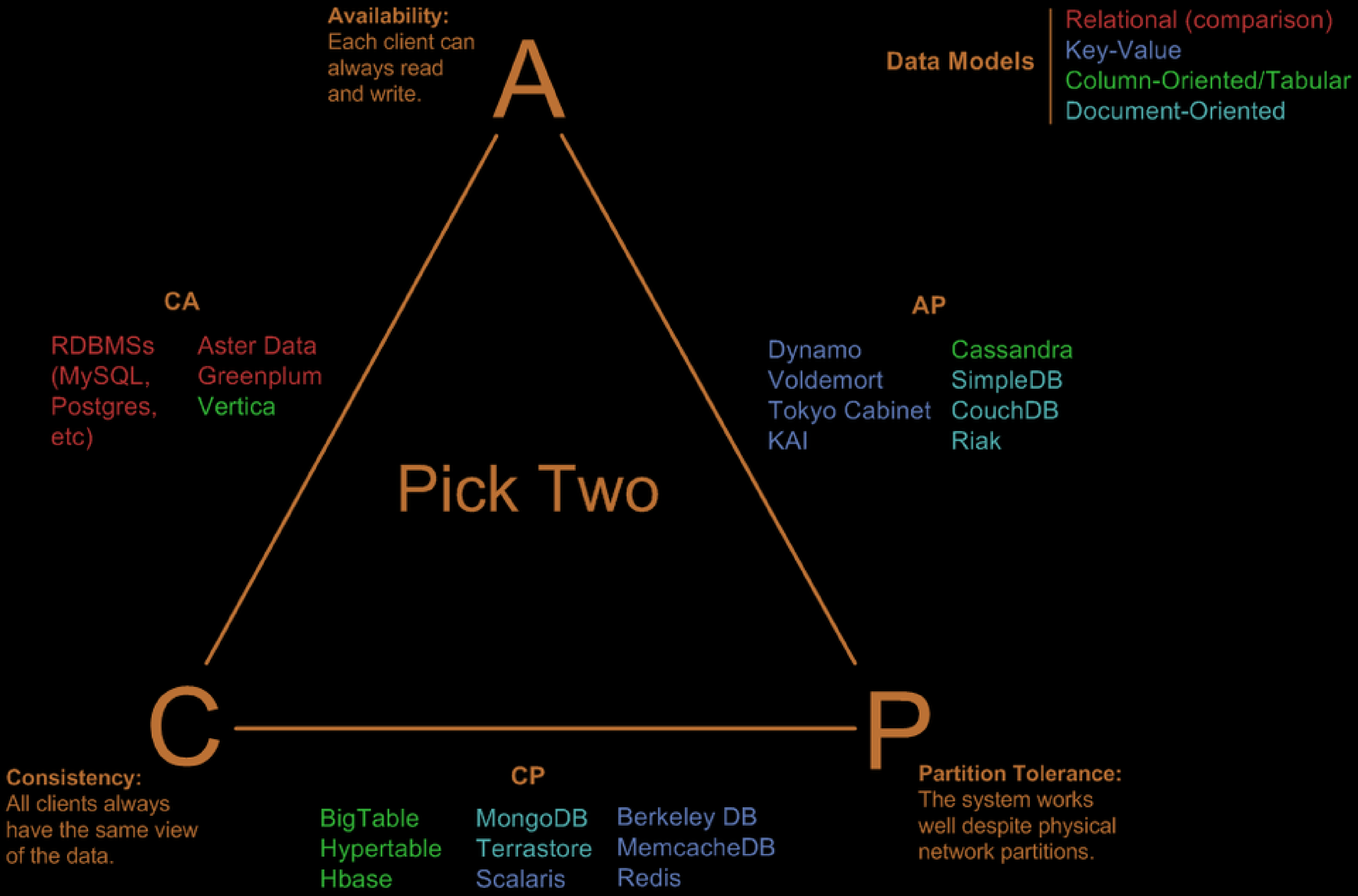
CAP Consequence

In the above scenario, server *p2* ***eventually must return a response***, even if the system is partitioned; if the message delay from p1 to p2 is sufficiently large that p2 believes the system to be partitioned, then it may return an incorrect response despite the lack of partitions. Thus it is even impossible to provide good responses when there are no partitions and bad responses only when there are partitions.

Enterprise System CAP Classification

- RDBMS : **CA** *(Master/Slave replication, Sharding)*
- Amazon Dynamo : **AP** *(Read-repair, application hooks)*
- Terracota : **CA** *(Quorum vote, majority partition survival)*
- Apache Cassandra : **AP** *(Partitioning, Read-repair)*
- Apache Zookeeper: **AP** *(Consensus protocol)*
- Google BigTable : **CA**
- Apache CouchDB : **AP**

Visual Guide to NoSQL Systems



END