AT82.02

DATA MODELING AND MANAGEMENT

UNIT 3-2: CAP THEOREM

CHUTIPORN ANUTARIYA (CHUTI AT AIT DOT AC DOT TH)

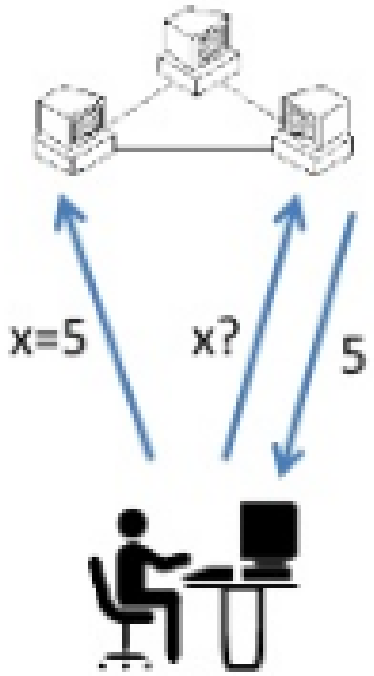
Co-funded by the Erasmus+ Programme of the European Union

DS&AI

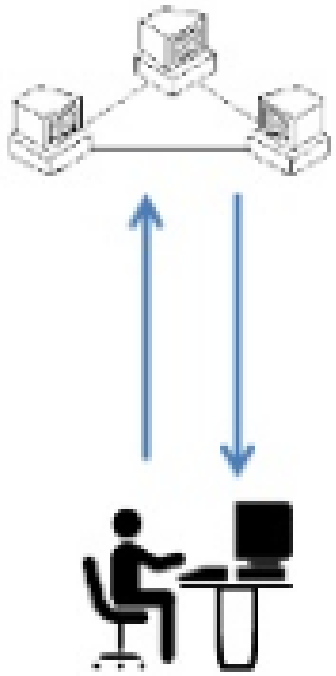
CAP Theorem

CAP Theorem

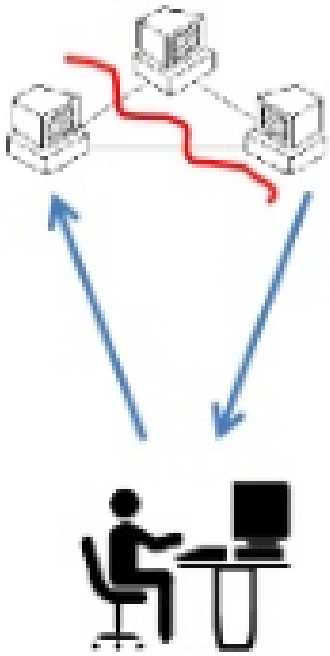
Consistency



Availability



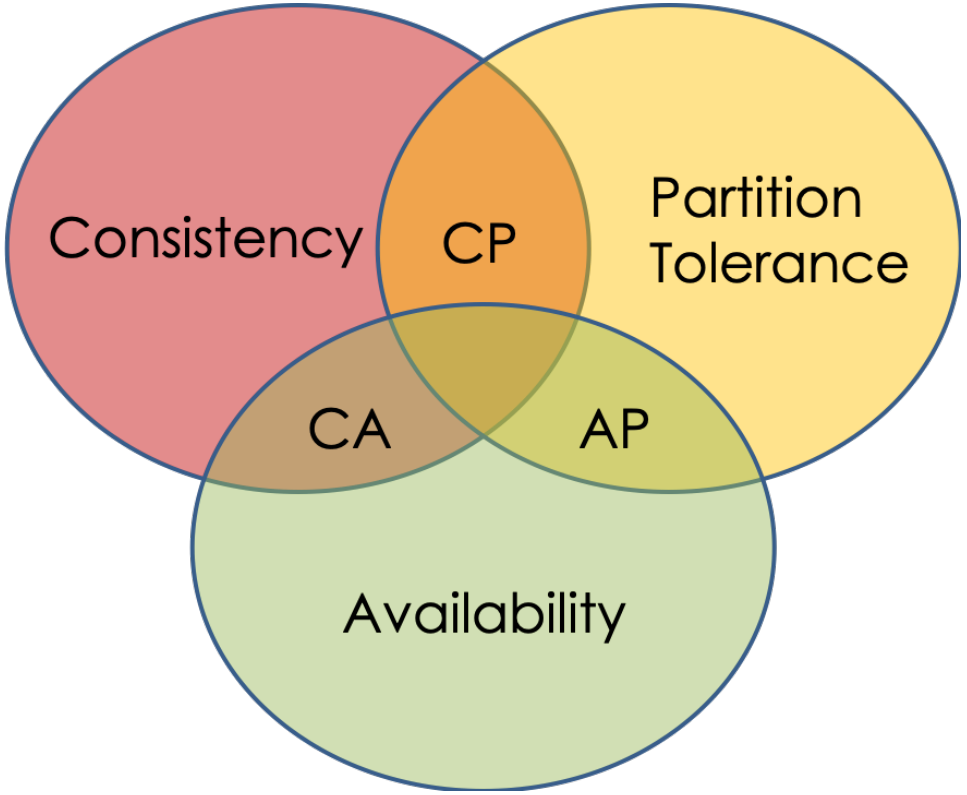
Partition tolerance



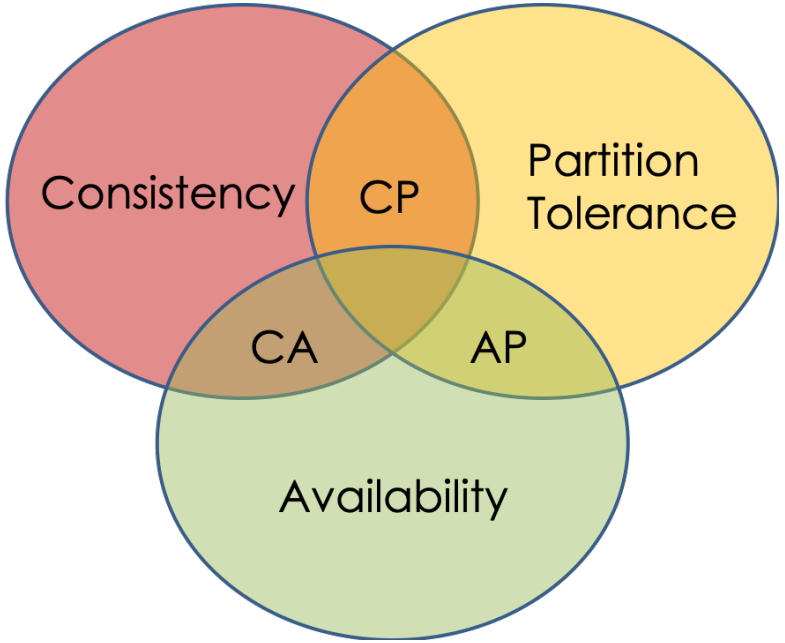
CAP Theorem

The CAP theorem (also called *Brewer's theorem*) states that a **distributed database system** (a networked shared-data system) can only guarantee two out of these three properties:

- Consistency
- Availability
- Partition Tolerance.



CAP Theorem



Consistency

- Having a single up-to-date copy of the data
- All nodes see the same data at the same time

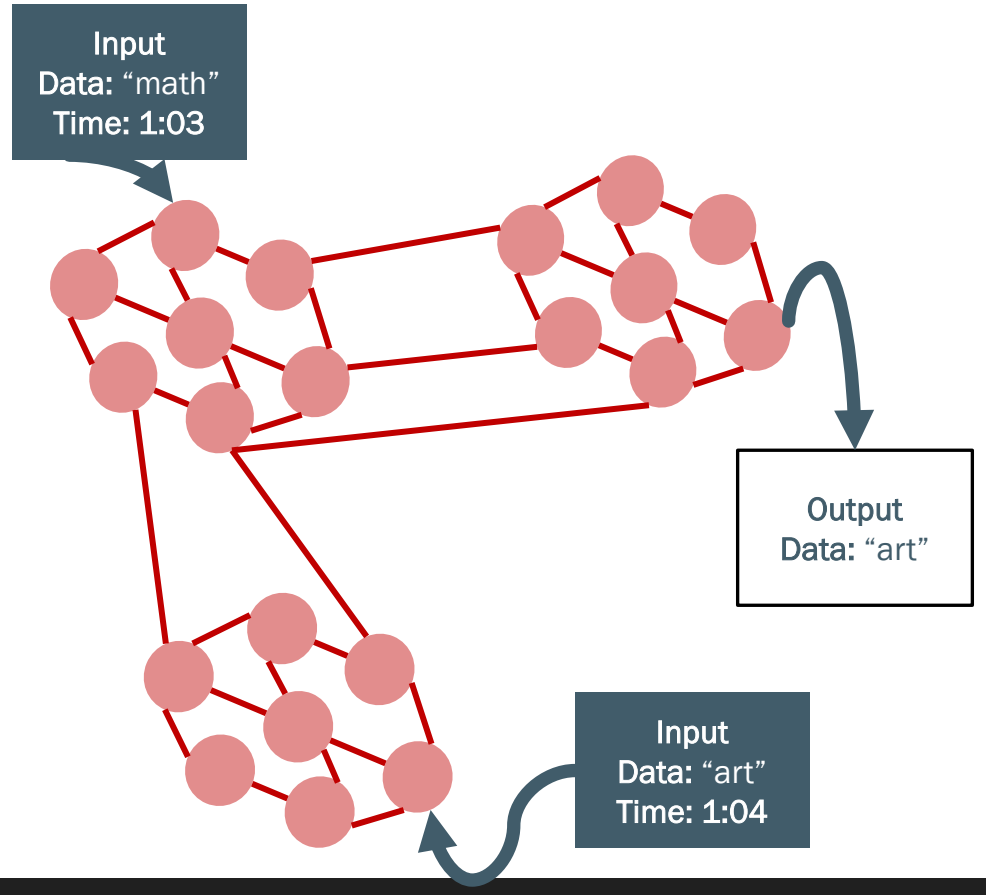
Availability

- Node failures do not prevent other survivors from continuing to operate (a guarantee that every request received by a non-failing node must result in a response)

Partition Tolerance

- The system continues to operate despite arbitrary partitioning due to network failures (e.g., message loss)

CAP Theorem: Consistency



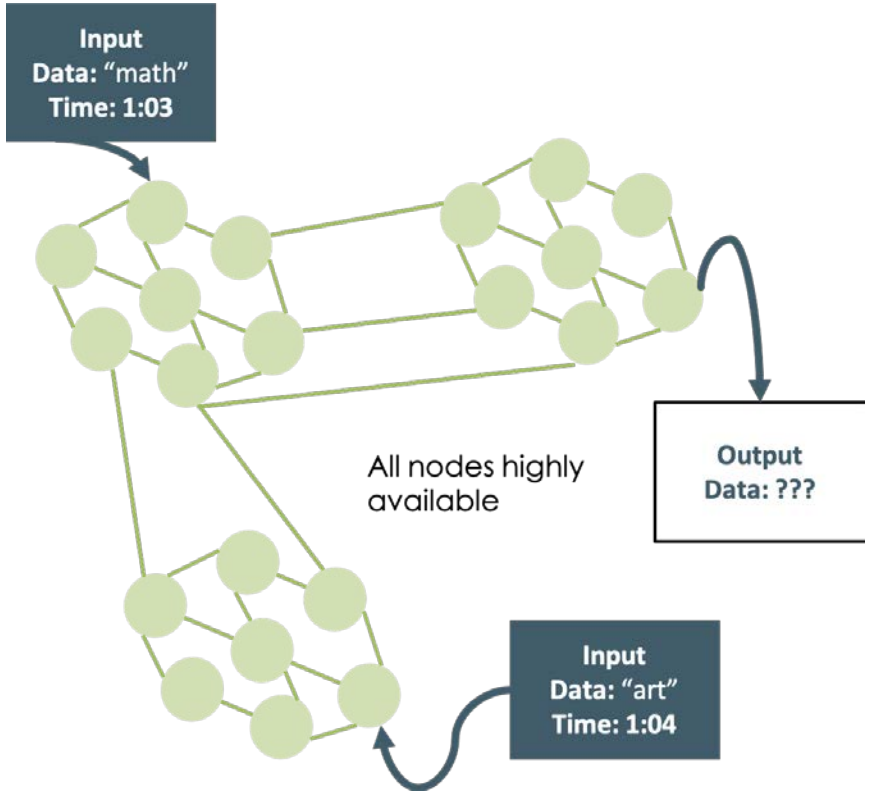
A system is said to be consistent if all nodes see the same data at the same time.

Simply, if we perform a read operation on a consistent system, it should return the value of the most recent write operation. This means that, the read should cause all nodes to return the same data, i.e., the value of the most recent write.

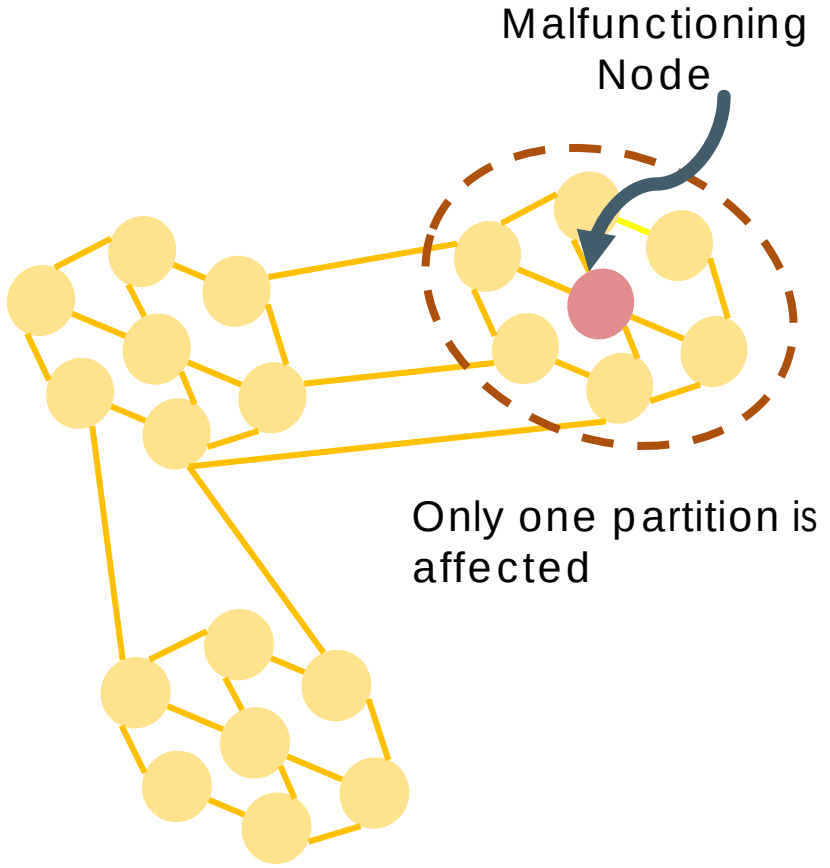
CAP Theorem: Availability

Availability in a distributed system ensures that the system remains operational 100% of the time. Every request gets a (non-error) response regardless of the individual state of a node.

Note: this does not guarantee that the response contains the most recent write.



CAP Theorem: Partition Tolerance

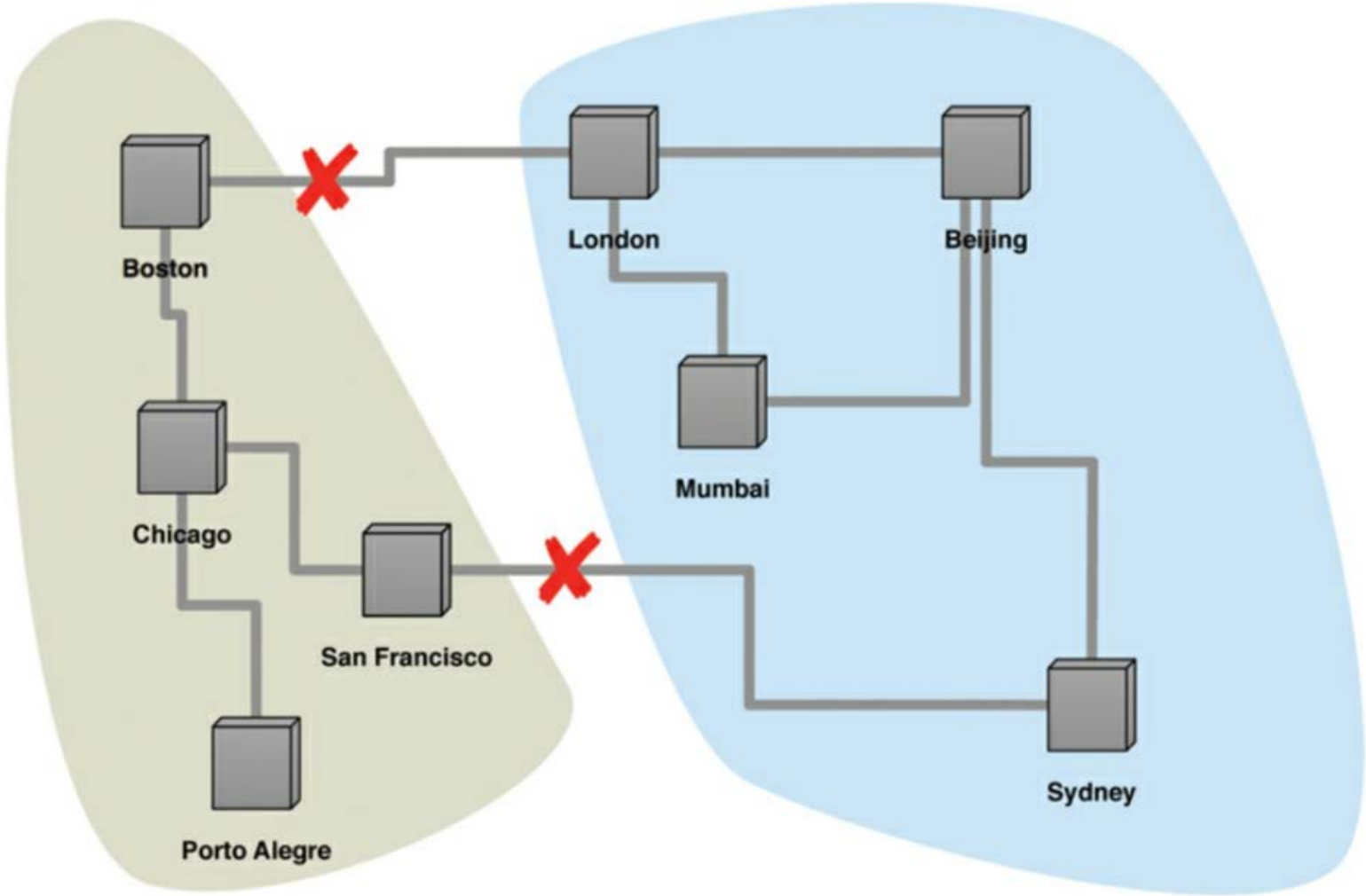


This condition states that the system does not fail, regardless of if messages are dropped or delayed between nodes in a system.

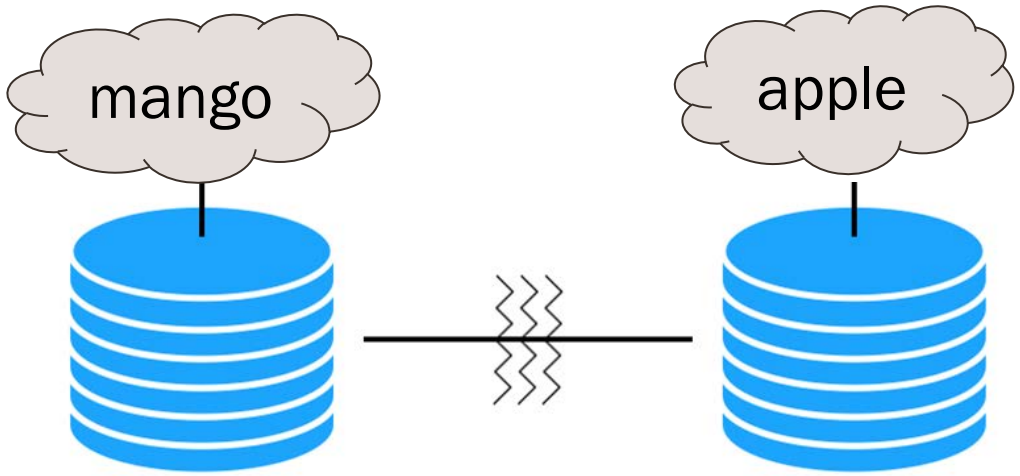
Partition tolerance has become more of a necessity than an option in distributed systems. It is made possible by sufficiently replicating records across combinations of nodes and networks.

Network Partition

Example of network partition



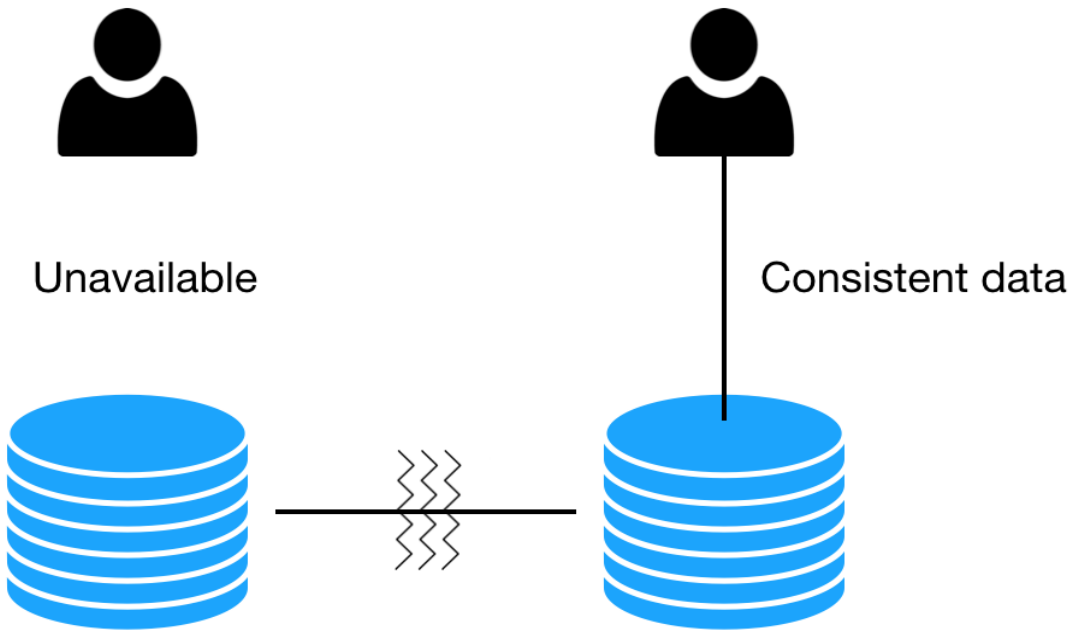
CAP Theorem



The **CAP theorem** states that in the presence of a network partition, one has to choose between **Consistency** and **Availability.**

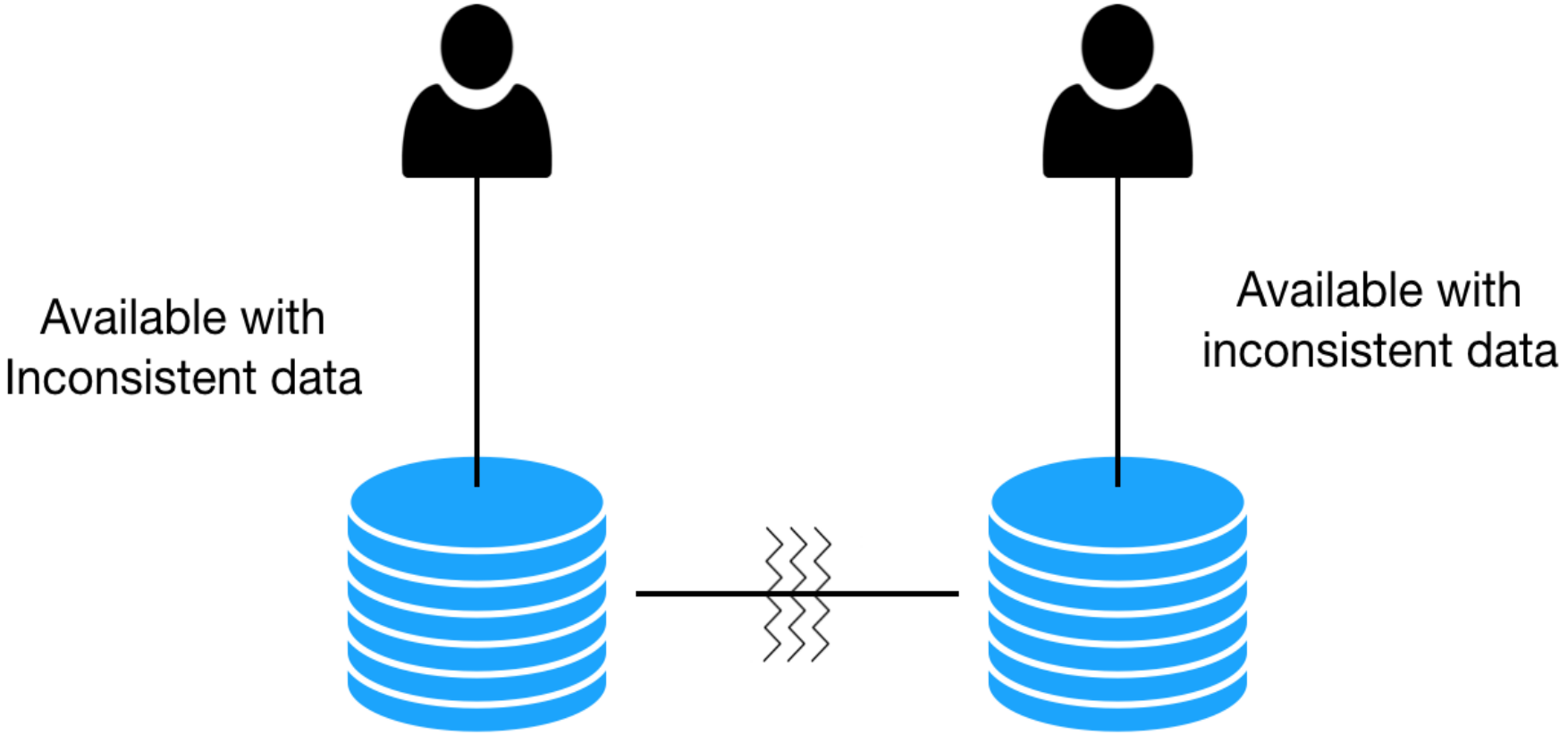
Meaning that “If you want to be consistent, you can’t always be available”

CP: Consistent + Partition Tolerant



When the system replies, you can believe them but they don’t always give your answers

AP: Available + Partition Tolerant



- *In a system that may suffer from network partitions, as distributed system do, you have to trade off consistency versus availability!*
- *This is not a binary choice, you can trade off a little consistency to get some availability.*

*Which is **right**?*

Which one to choose AP or CP?

◎ There is no right or wrong.
◎ Choosing what suits best for the use case is the definitive resolution.

*Who makes the **decision**?*

*The development team **or** the business?*

What is the cost of achieving Consistency or how much would it cost in the case of Inconsistency?

How important is it to achieve high Availability or is it ok to throw Errors?

Criteria to consider:

How much Latency is tolerated? (high latency approaching ∞ is equal to no availability)

How Complex can a solution get?

PACELC

If there is *Partition*,

how does the system trade-off between *Availability* and *Consistency*

Else

how does the system trade-off between *Latency* and *Consistency*

Note: As mentioned before high **L**atency can be termed as no **A**vailability

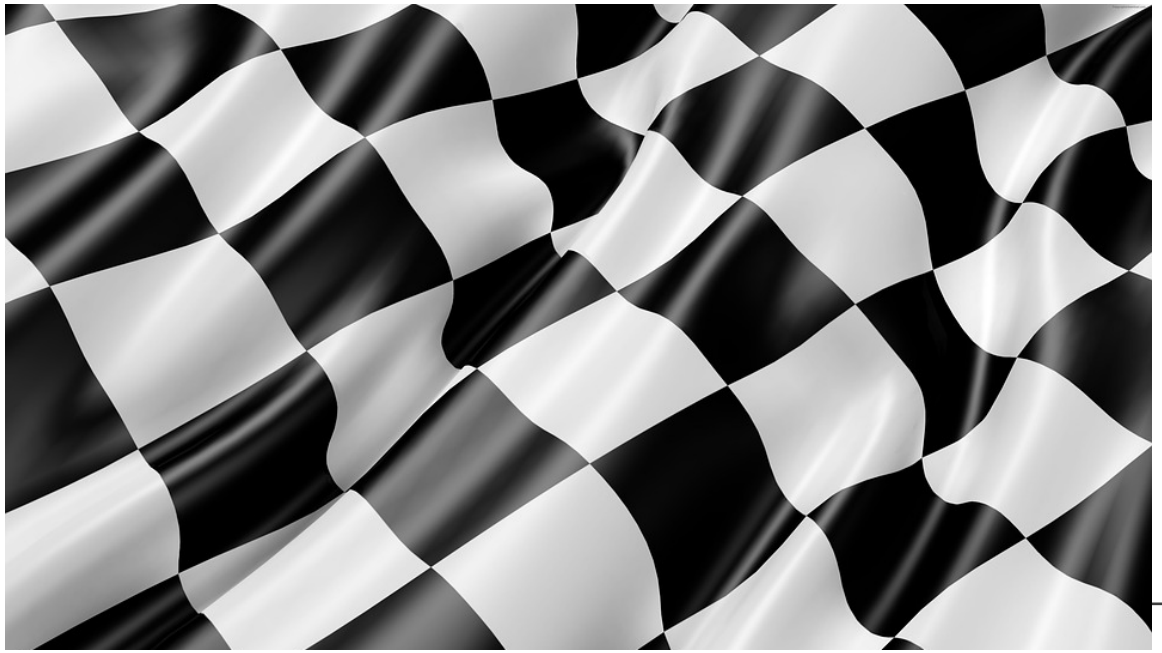
What we desire?

High **A**vailability and high **C**onsistency

But we know we can't have both **A**vailability (low **L**atency) and **C**onsistency when there is network **P**artition ????

How about having high Availability (low Latency) over Consistency for the time being and getting Eventually Consistent?

- P. Sadalage and M. Fowler: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Addison-Wesley Professional, 2013



Thank you.

Let’s Summarize!