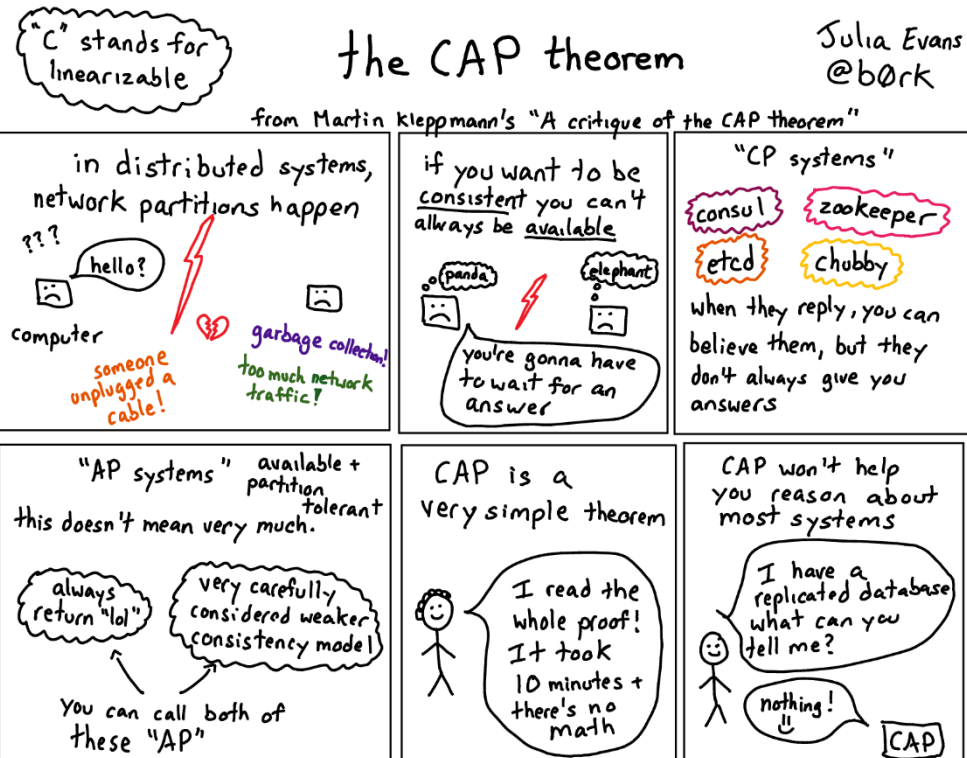


CAP Theorem

Consistency – dictionary.com

- steadfast adherence to the same principles, course, form, etc.:
- agreement, harmony, or compatibility, especially correspondence or uniformity among the parts of a complex thing;
- the condition of cohering or holding together and retaining form; solidity or firmness.

By Julia Evans Blog –
<https://jvns.ca/blog/2016/11/19/a-critique-of-the-cap-theorem/>

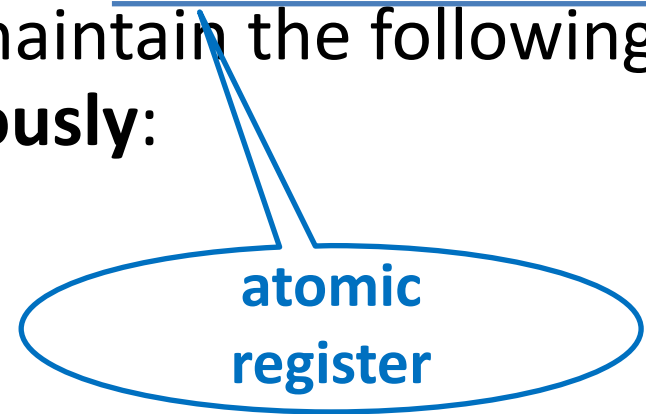


CAP Theorem Introduction

- It is **impossible** for a replicated **read-write store** in an asynchronous system to maintain the following **three guarantees simultaneously**:
 - Consistency
 - Availability
 - Partition-tolerance
- Initially, conjectured by Eric Brewer in 1998, later proven by [Lynch et al.](#)
- Describes *tradeoffs* involved in distributed system design

CAP Theorem Introduction

- It is **impossible** for a replicated read-write store in an asynchronous system to maintain the following **three guarantees simultaneously**:
 - Consistency
 - Availability
 - Partition-tolerance
- Initially, conjectured by Eric Brewer in 1998, later proven by *Lynch et al.*
- Describes *tradeoffs* involved in distributed system design

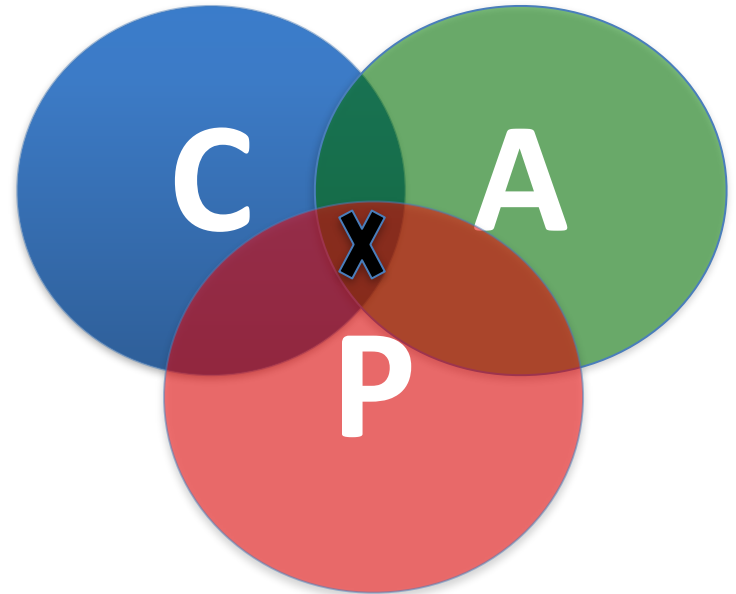


Definition of *Atomic Register*

- Models **replicated read-write store**:
 - `set(X)` sets value of register to X
 - `get()` returns the value of register
- **Replicates and distributes register, must maintain consistency and availability** across all replicas
- Models many distributed system types
 - Key-value store, distributed shared memory
 - Replicated system, distributed file system, etc.

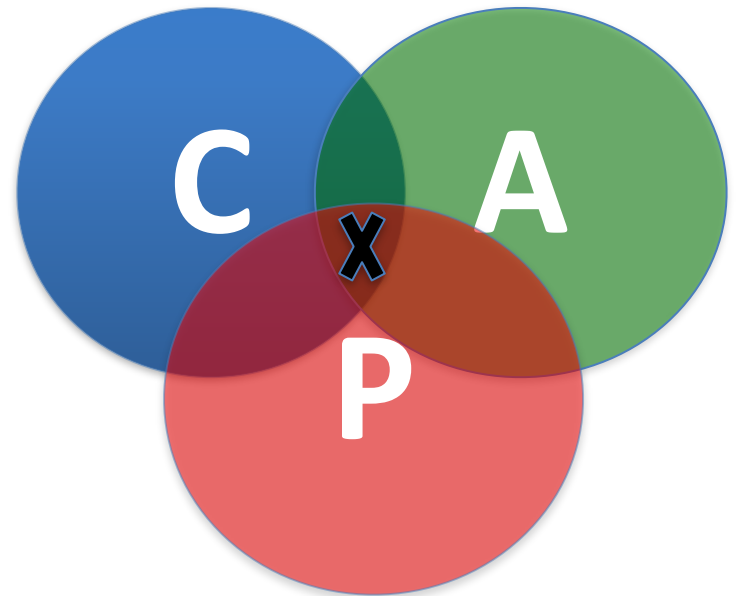
C-A-P: Choose two out of three*

- **Consistency:**
 - All read requests should read the *latest* value (or return an error)
- **Availability:**
 - All requests should *return successfully*
- **Partition-tolerance:**
 - The system can *tolerate* arbitrary number of communication *failures*



C-A-P: Choose two out of three*

- Consistency:
 - All read requests should read the *latest* value (or return an error)
- Availability:
 - All requests should *return successfully*
- Partition-tolerance:
 - The system can *tolerate* arbitrary number of communication *failures*
- Traditional view
 - Today, more a **spectrum**



Definition of Consistency

- Refers to **replication consistency**
 - Not related to $A - C - I D$ properties for transactions
- Could mean **strict consistency**
 - As we know, this is by and large impossible in a distributed system
- Thus, here, assumes **linearizability**
- This usually means replication across sites should be done **eagerly**

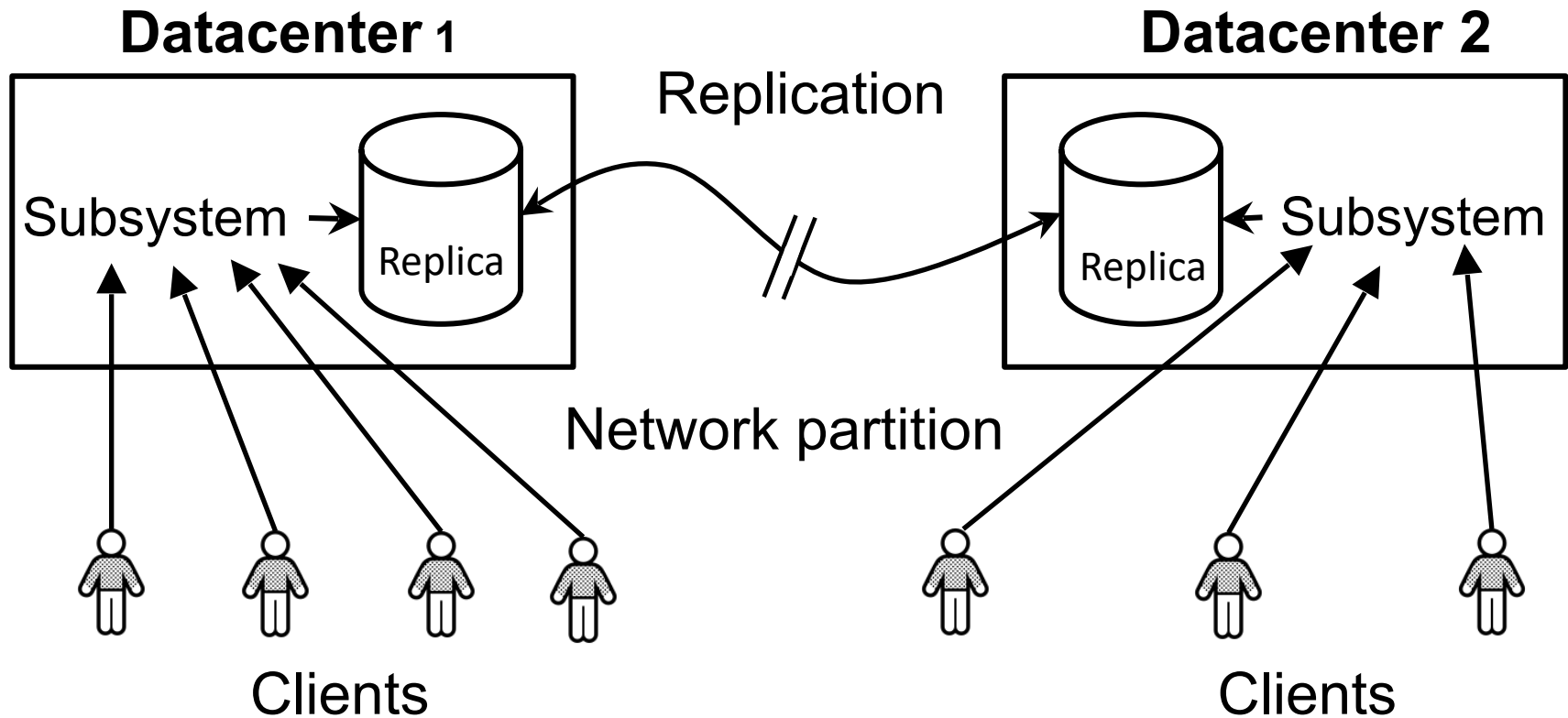
Definition of Availability

- **Every request** received by a non-failed node must result in a **non-error response**
 - **Non-triviality requirement:** a system which always responds with errors is **not available**
- Assumes a **crash failure** model for nodes
 - **Functioning nodes** must continue to operate even if there are **failed nodes** in system
- No requirement on **latency**: response can be very slow but must eventually come through
- Both a **weak and strong** definition: no latency guarantee, but 100% response success

Definition of Partition-Tolerance

- Asynchronous system model
- Message loss (failure model)
- Partition means total communication loss between partitioned subsystems
- Subsystems **continue request processing** even if a network partition causes communication loss within system
- If the system requires a **stronger system model**, or a **weaker failure model**, then it is **not partition-tolerant**
- No guarantee that **partitions recover**, but it doesn't mean they are **always present** either

Definition of Partition-Tolerance



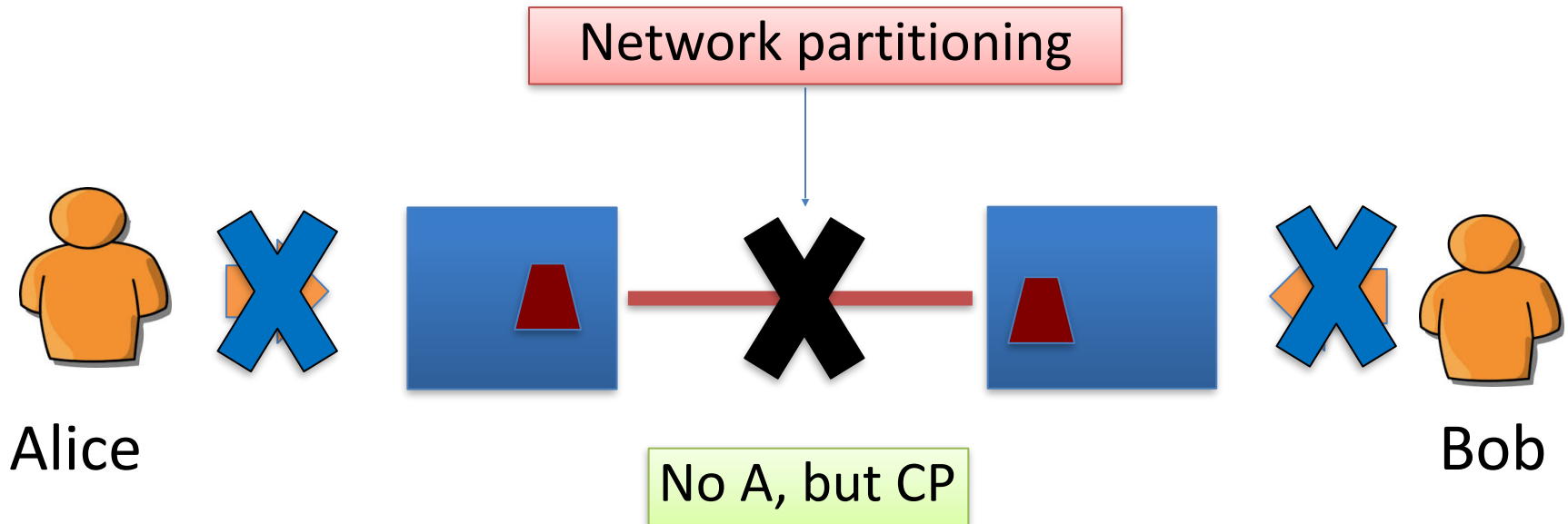
Illustrating Example

Hotel Booking: are we double-booking the same room?



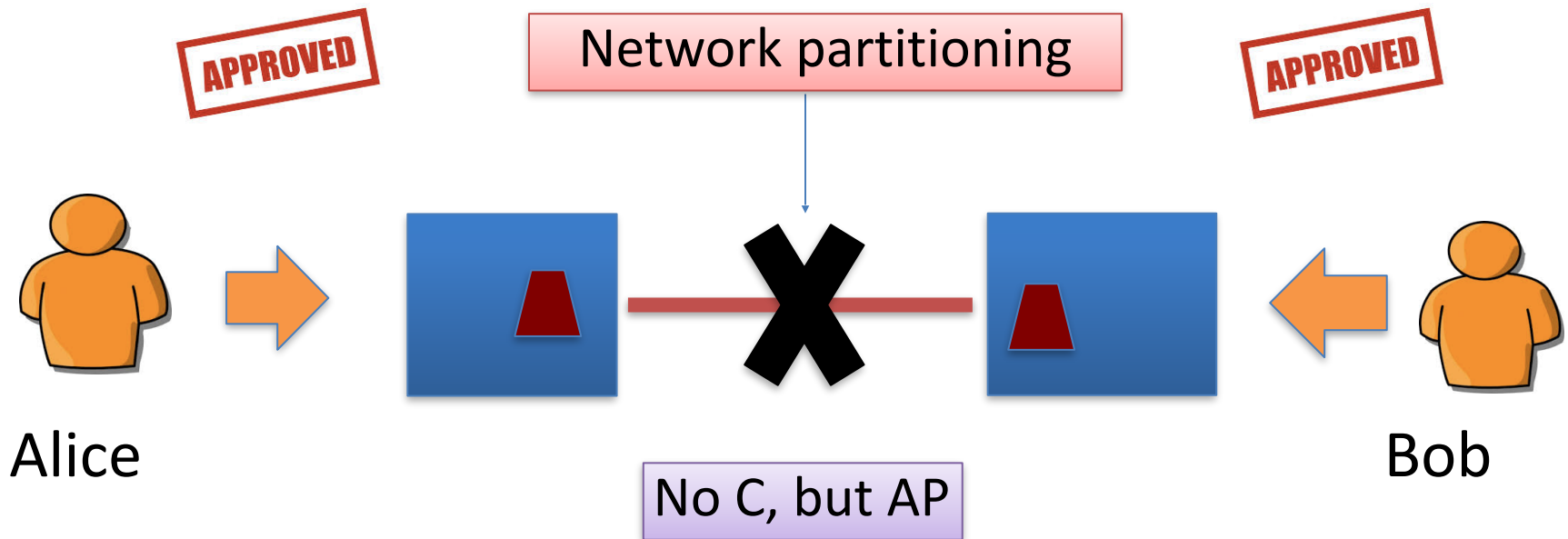
Illustrating Example

Hotel Booking: are we double-booking the same room?



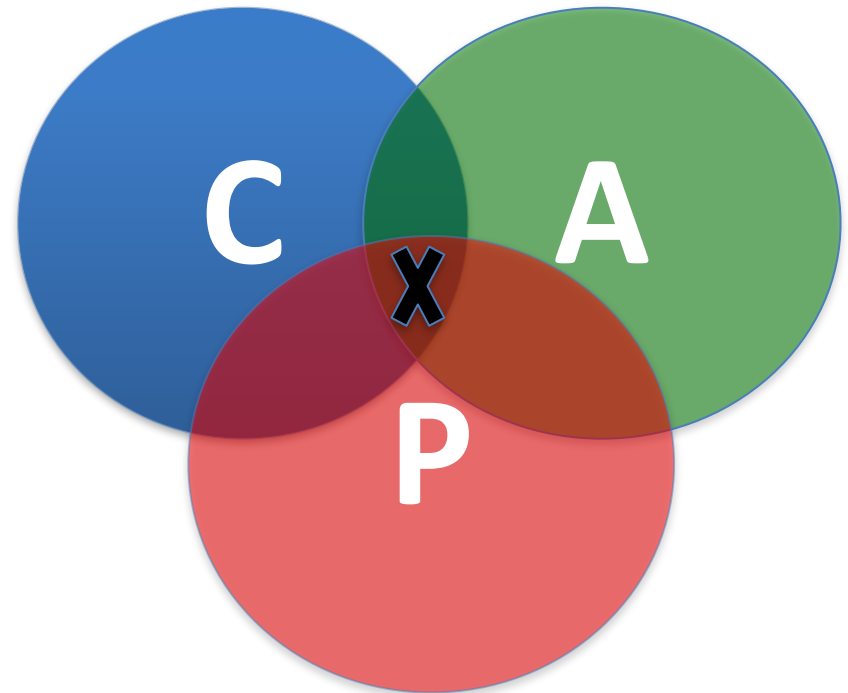
Illustrating Example

Hotel Booking: are we double-booking the same room?

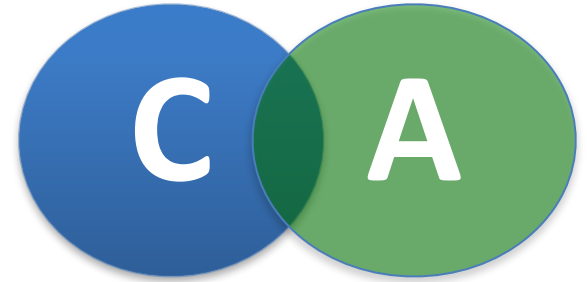


Types of CAP systems

- CAP says “*pick two out of three*”
- The following systems seem possible:
 - CP
 - AP
 - CA
- But it’s not so easy...



CA Systems



- A “perfect” system!
- But with strong assumptions:
 - Either the network is reliable (Fallacy of DS...)
 - Or, the network is not used (then it is not a DS!)
- *“Of the CAP theorem’s Consistency, Availability, and Partition-Tolerance, **Partition Tolerance is mandatory in distributed systems.** You cannot not choose it.”*
 - Coda Hale, Yammer software engineer

Misconception #1: Always choose two

- CA systems cannot be used in practice
- **But**, when there are **no network partitions**, every system can behave like CA!
- In other words, “*choose two*” only takes effect **during network partitions**
- So in reality, there are **only two types of systems** ... i.e., if there is a partition, does the system **give up availability or consistency?**
 - Daniel Abadi, co-founder of Hadapt

Misconception #2: C, A are binary

- CAP theorem uses very narrow definitions of C,A:
 - Consistency: Linearizability
 - Availability: Infinite latency budget, 100% successful
- *“The “2 of 3” formulation was always **misleading** because it tended to oversimplify the tensions among properties. **CAP prohibits only a tiny part of the design space: perfect availability and consistency in the presence of partitions, which are rare.**”*
 - Eric Brewer, CAP Theorem

Reality of the CAP Theorem

- *“Many designers incorrectly conclude that the theorem imposes certain restrictions on a DDBS **during normal operation** and therefore implement an **unnecessarily limited system**.”*
 - Daniel Abadi, Co-founder of Hadapt
- All systems are, in fact, CAP, but tune how much C and A are provided during P
 - $CP \rightarrow C(a)P$, and $AP \rightarrow (c)AP$
- Provides **freedom** to design system to **suit** application requirements
 - E.g., by choosing appropriate consistency level

AP: Best Effort Consistency

- Example:
 - Web caching (cf., cache consistency)
 - Network File System (cf., concurrent writes)
- Characteristics:
 - Optimistic replication (i.e., lazy replication)
 - Expiration/time-to-live
 - Conflict resolution (e.g., CRDTs)
- Cassandra, Dynamo are AP systems:
 - Eventual consistency: stale data can be read
 - Tunable towards CP

CP: Best Effort Availability

- Example:
 - Distributed lock services (Chubby, ZooKeeper)
 - Paxos (safe, not live)
- Characteristics:
 - Eager replication
 - Pessimistic locking
 - Minority partition becomes unavailable
- BigTable / HBase are CP systems:
 - Provide linearizability
 - Under partitions, cannot access TabletServer/
RegionServer

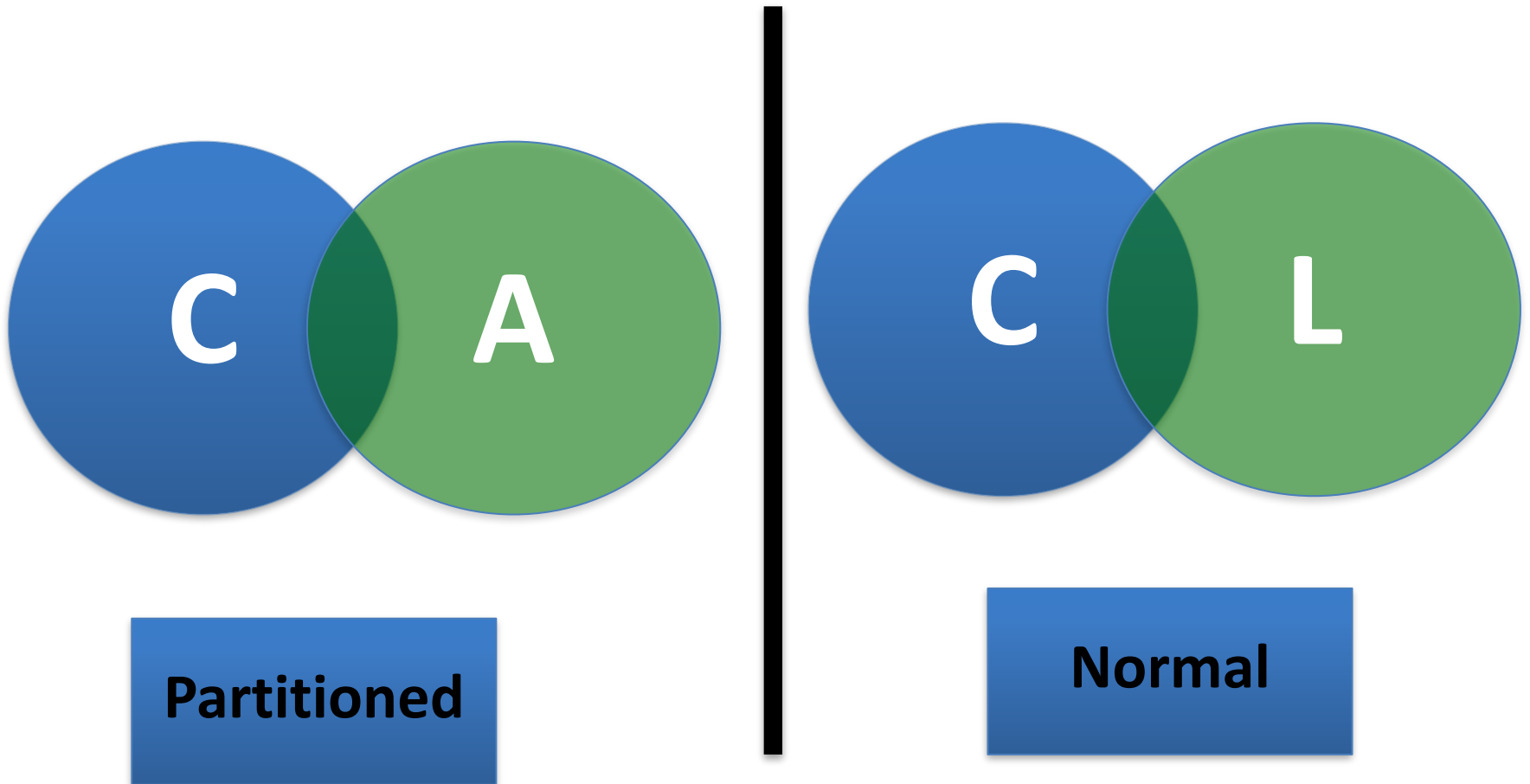
Misconception #3: Static systems

- Systems don't always behave the same way at run-time
- It is possible to design systems which behave differently depending on the operational situation when **partitioning**
- Example: airline reservation system
 - When most seats available: system **behaves AP**, worries about capacity limit later
 - When plane is close to capacity: system **behaves CP**, ensures no overbooking, or **behaves AP** to maximize profit and handle compensations out-of-band

Extended Model: PACELC

- A more complete description of the space of potential tradeoffs for distributed system:
 - If there is a partition (P), how does the system trade off availability and consistency (A and C); else (E), when the system is running normally in the absence of partitions, how does the system trade off **latency (L) and consistency (C)**?
- Original CAP theorem ignores latency, yet important in practice

PAC/ELC



Examples

- PA/EL: Give up consistency at all times for availability and lower latency
 - Dynamo, Cassandra (tuneable), Riak, web caching
- PC/EC: Refuse to give up consistency, pay the cost in availability and latency
 - BigTable, HBase, VoltDB/H-Store
- PA/EC: Give up consistency when partitions occur, but keep consistency during normal operation
 - MongoDB
- PC/EL: Keep consistency when partitions occur, but give up consistency for latency during normal operations
 - Yahoo! PNUTS

Summary CAP Theorem

- Classically described as “*pick two out of three*”: **consistency**, **availability**, **partition-tolerance**
- Really boils down to **choosing C or A**, since P is a **must-have** for practical systems
- During normal operations, systems can all be **CA**
- Only concerns “**perfect**” **notions** of C and A
- In reality, C and A **are tunable**, systems tend to maintain C and A during P to some degree
- Systems can **adapt** dynamically to become AP or CP for different operational situations
- PACELC: extended model which considers differentiations between P and not P and consistency vs. latency

Self-study Questions

- Go through the distributed systems discussed throughout the course and determine where in the CAP space they lie?
- At a high-level, specify systems that are CA, CP and AP – what systems are CA?
- Is partition-tolerance a binary property? Discuss.
- See if you can specify a spectrum of C vs. A, given P.

