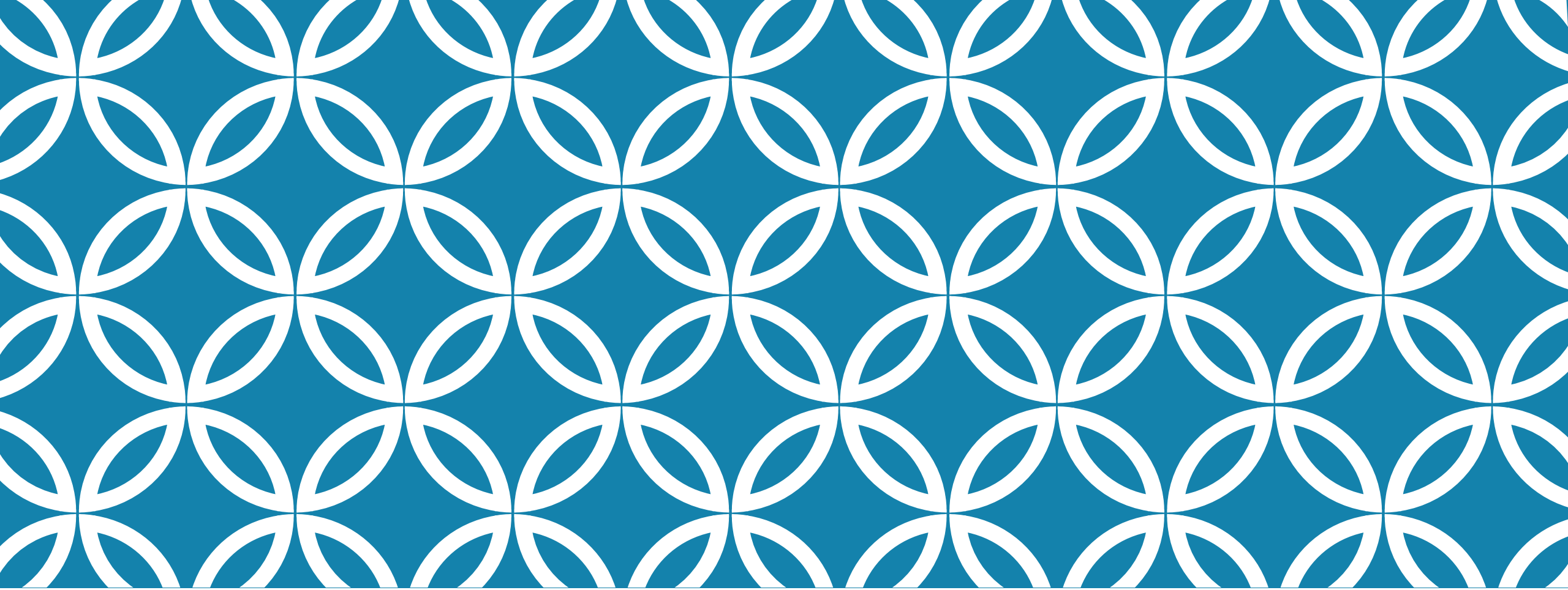


CAP TWELVE YEARS LATER: HOW THE "RULES" HAVE CHANGED

BY: ERIC BREWER, UNIVERSITY OF CALIFORNIA, BERKELEY

Speaker:
Ohoud Alharbi

OUTLINE

CAP Theorem

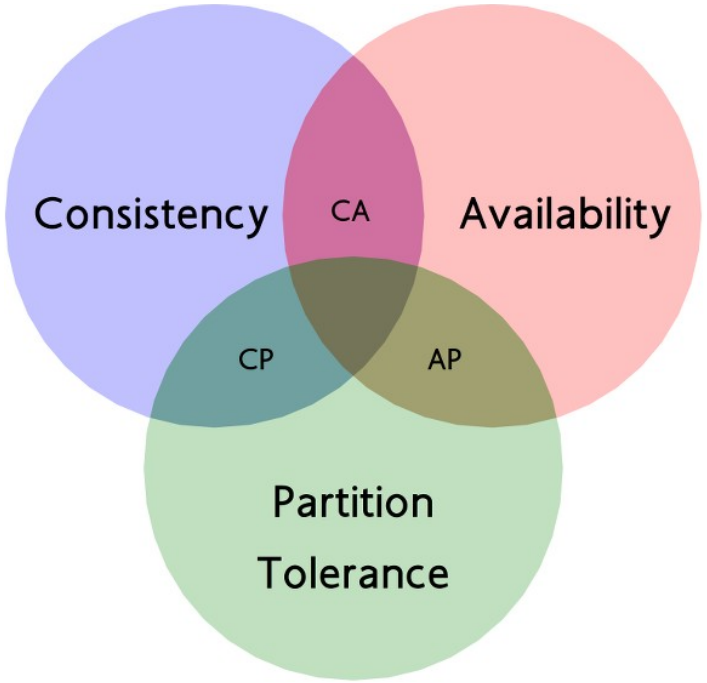
Why 2 of 3 in CAP theorem is misleading?

CAP-Latency Connection

Managing Partitions

THE CAP THEOREM

Any networked shared-data system can have at most two of the three CAP properties



PROPERTIES OF DISTRIBUTED SYSTEMS

Consistency

Having single up to date copy of the data.

All nodes see the same data at the same time

Availability

A guarantee that every request receives a response about whether it was successful or failed

Partition tolerance

The system continues to operate despite arbitrary message loss or failure of part of the system

WHY “2 OF 3” IS MISLEADING?

- **Oversimplify the tensions among properties.**
- **Partitions are rare, CAP should allow perfect C and A most of the time**
- **There is an incredible range of flexibility for handing partitions and recovering.**
- **The choices between C and A can occur at granular levels (subsystem level, based on operation, based on user, based on data ..etc.)**
- **All three properties are more continuous than binary (0-100%).**

CAP-LATENCY CONNECTION

- The CAP theorem **ignores latency**.
- Latency and partitions are deeply **related**.
- **Operationally,** the essence of CAP takes place during a timeout.

  Timeout: a period when the program must make a fundamental decision:
  - ❑ **Cancel** the operation and decrease availability.
  - ❑ **Proceed** with operation and risk consistency.
- **Retrying** communication just **delays** this decision and indefinite retry is essentially C over A

PRAGMATIC VIEW

Pragmatically, a partition is a time bound on communication. Failing to achieve consistency within the time bound implies a partition and thus a choice between C and A for this operation.

Pragmatic view consequences:

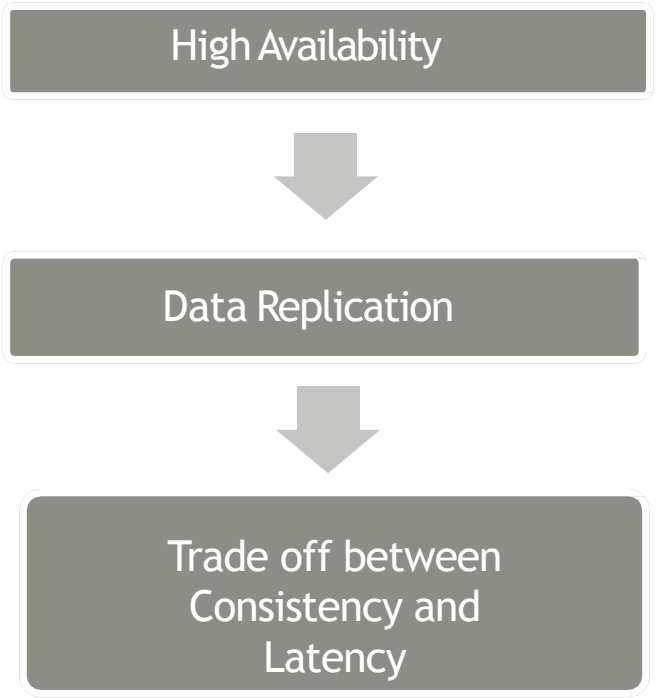
- No global notion of partition: some nodes may detect partition others not.
- Nodes that detected partition can enter **partition mode:** optimize the consistency and availability in partition mode
- Designer can **set time bounds** according to their needs: **tighter** time bounds may make subsystems enter partition mode frequently.

THE CONSISTENCY-LATENCY TRADE-OFF

- Data Replication implies a trade-off between consistency and latency as we have to update replicas.

There are two ways to send data updates

- ❑ Data updates sent to **all replicas** at the same time.
- ❑ Data updates send to **a master copy**.



DATA UPDATES SENT TO ALL REPLICAS

Data updates sent to **all replicas** at the same time:

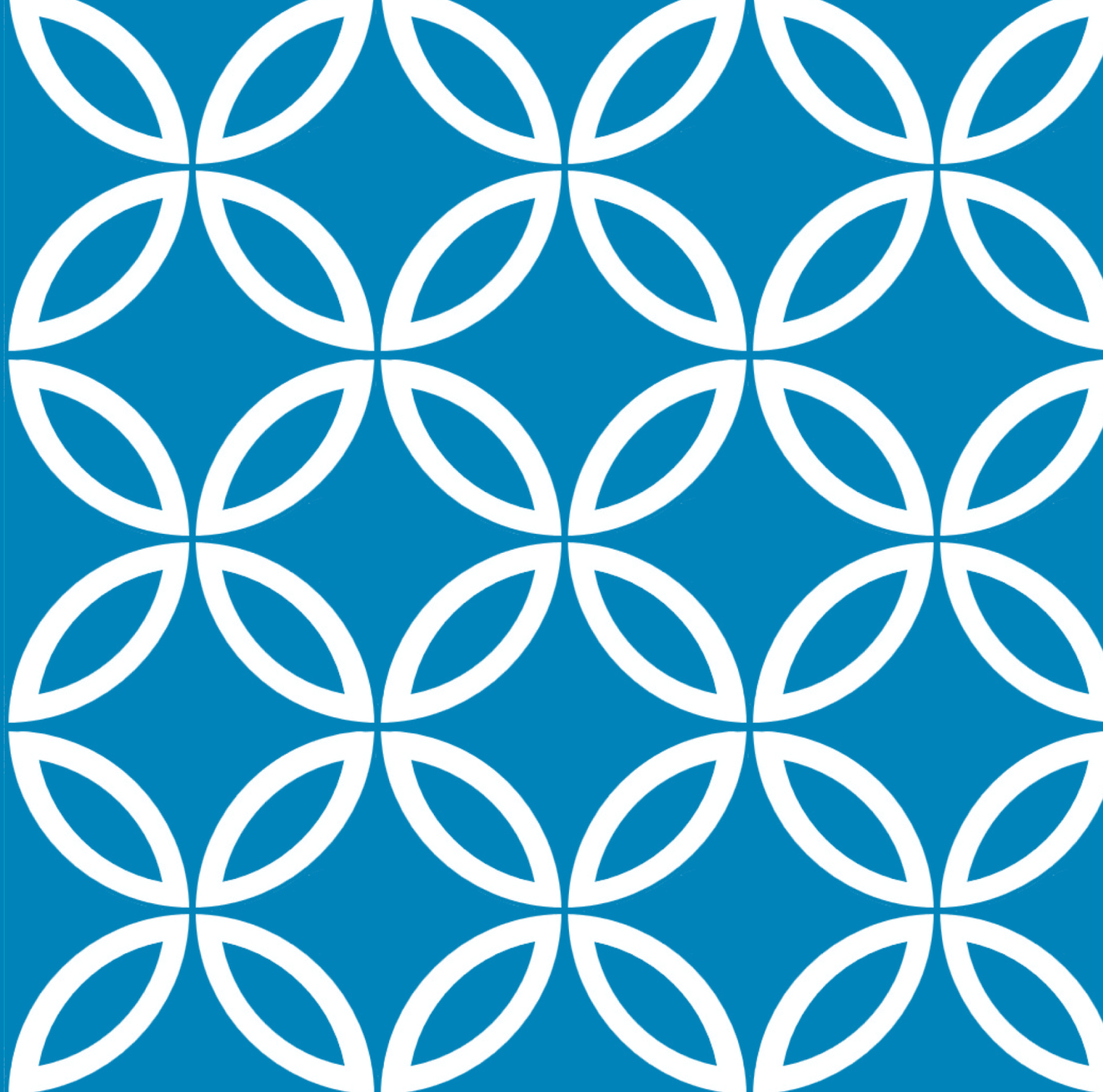
- Result in lack of **consistency.**
- Result in **Latency.**

DATA UPDATES SENT TO A MASTER NODE

- The **master** nodes **resolves updates.**
- There are 3 options for replication of updated data:

1. Replication is **synchronous.** (increase latency)
2. Replication is **asynchronous:**
   a) Systems routes all **read** to the **master** node (increase latency)
   b) Any **node** can serve **read** request (lack of consistency)
3. A **combination** of two above:
   The system sends updates to some **subset of replicas synchronously** and rest **asynchronously.**

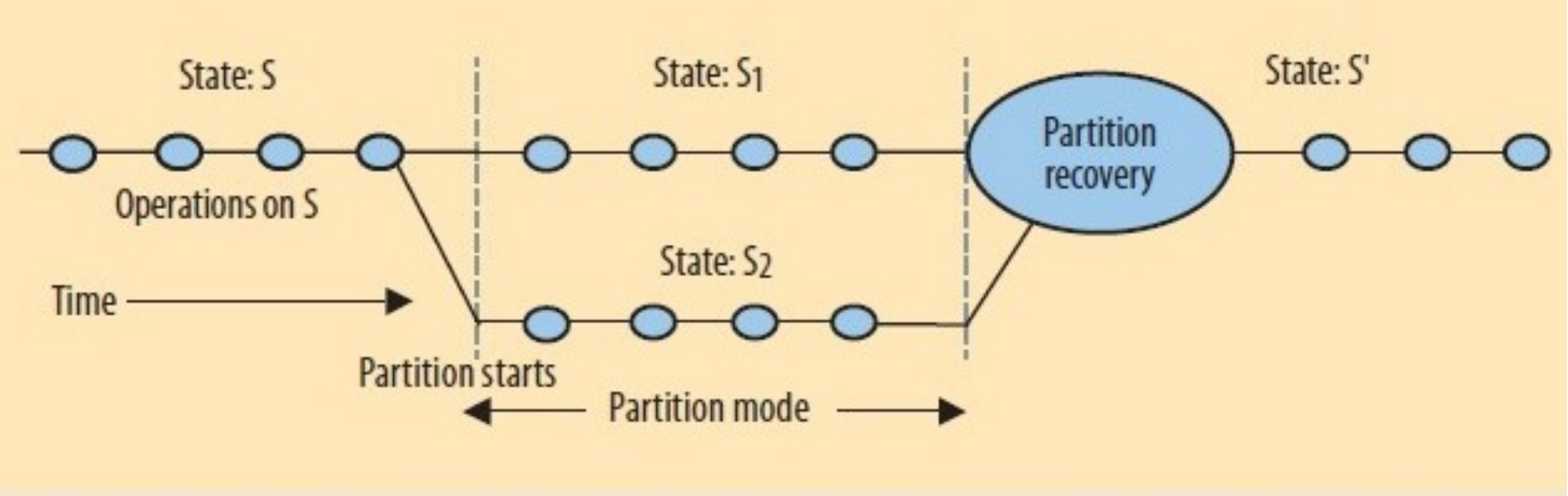
MANAGING PARTITIONS



MANAGING PARTITIONS

1. **Detect** partitions.
2. **Enter** an explicit **partition mode** that can limit some operations
3. **Initiate** a **recovery process** to restore consistency and compensate for mistakes made during a partition.

MANAGING PARTITIONS



MANAGING PARTITIONS

- Once the system **times out,** it detects a **partition.**
- The detecting side enters **partition mode.**
- Once the system enters partition mode, two strategies are possible:
  1. **Limit some operations,** thereby reducing availability.
  2. **Record extra information** about the operations that will be helpful during partition recovery.

WHICH OPERATIONS CAN PROCEED IN PARTITION MODE?

The designer must decide whether:

- **Maintain a particular invariant** during partition mode

  or

- **Risk violating it** with the intent of restoring it during recovery.

  E.g. Designers allow duplicate keys during a partition. Duplicate keys are easy to detect during recovery, and, assuming that they can be merged.

WHICH OPERATIONS CAN PROCEED IN PARTITION MODE?

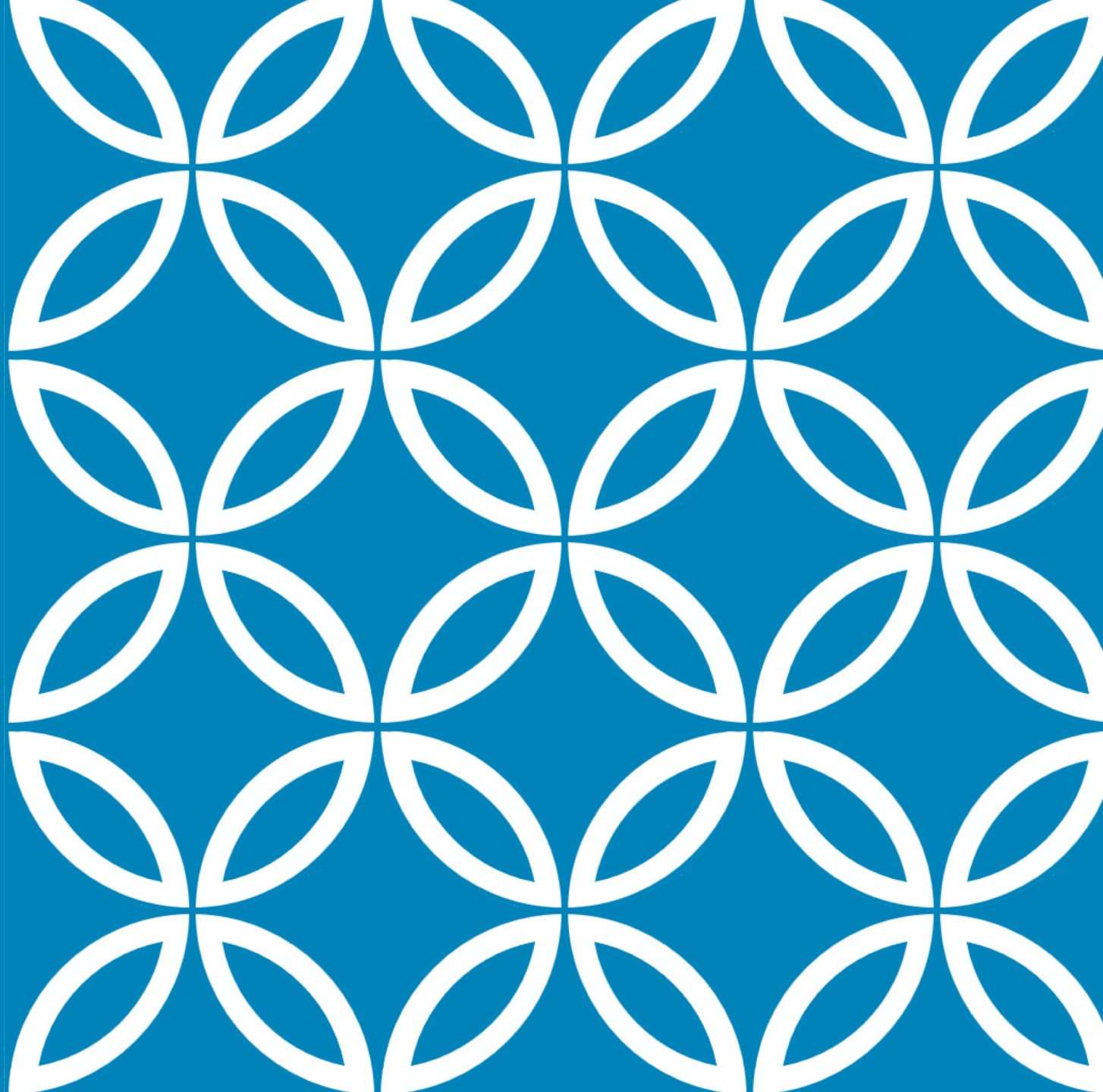
- Partition mode gives rise to a fundamental **user-interface challenge.**

  E.g. cloud services with an offline mode such Google Docs.

- The best way to track the **history of operation** on both side is to use version vectors

  Vector's elements are a pair **(node, logical time)**.

PARTITION RECOVERY



PARTITION RECOVERY

The designer must solve two hard problems during recovery:

1. **Re-enforce consistency on both sides**

   - ❑ Handle merge conflicts
     - Manual conflict merging
       (Wiki offline mode, GitHub)
     - Merge conflicts by following certain rules
       (Google Docs)

PARTITION RECOVERY

The designer must solve two hard problems during recovery:

1. **Re-enforce consistency on both sides**

   - Automatic state convergence
     - Delaying risky operations.

     (constrain the use of certain operations during partitioning)

     - Commutative operations.

     (The system links logs together, sorts them into some order, and then executes them)

PARTITION RECOVERY

The designer must solve two hard problems during recovery:

2. Compensate for the mistakes made during partition mode

- The designer create a **restoration strategy** for each invariant.
- The system discovers the violation during recovery and must **fix at that time:**
  - ❑ "last writer wins" (which ignores some updates).
  - ❑ Merge operations, and human escalation (e.g. overbooking).

PARTITION RECOVERY

The designer must solve two hard problems during recovery:

- Recovering from **externalized mistakes** typically requires some **history** about externalized outputs.
- Issuing **compensating actions.**

  E.g. reverse transactions, refunds, coupons, charging a fee.

RECAP

- The **CAP theorem** asserts that networked shared-data system can have only **two of three** properties.
- System designers should not **sacrifice consistency or availability** when partitions exist.
- By explicitly **handling partitions,** designers can **optimize consistency and availability**.
- Designers can choose to **constrain the use of certain operations** during partitioning so that the system can automatically merge state during recovery.
- Designers can choose to **risk violating invariants** with the intent of restoring it during recovery.
- Explicit **details of all system invariants** during partition are needed to enable recovery.

THANK YOU!