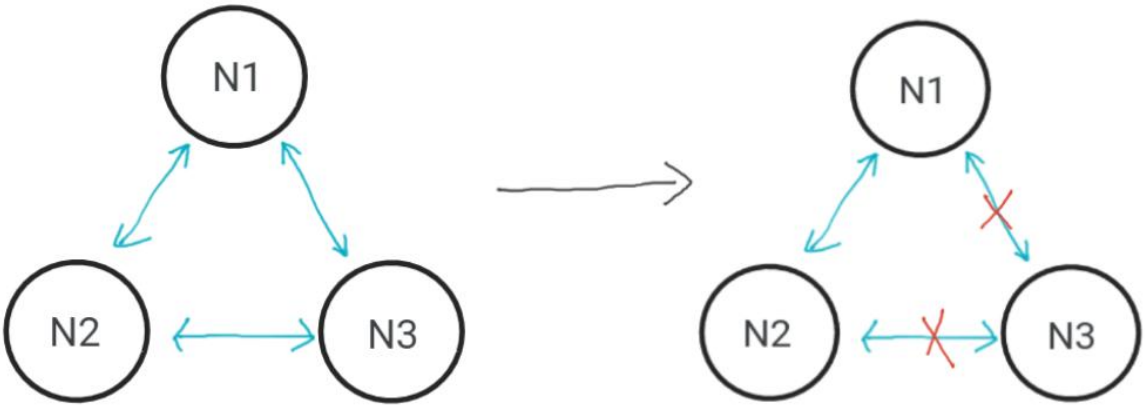
Large Scale Computing

Lab 5

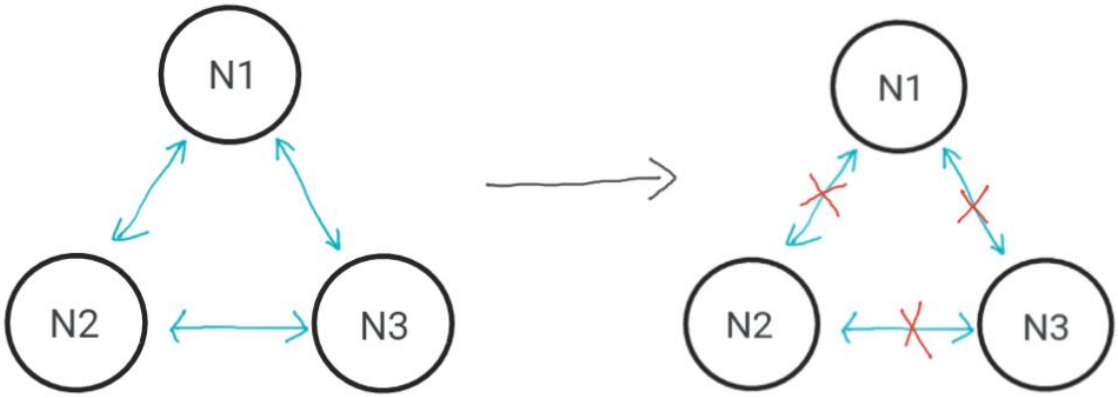
Adrian Madej

1. Exercise 1



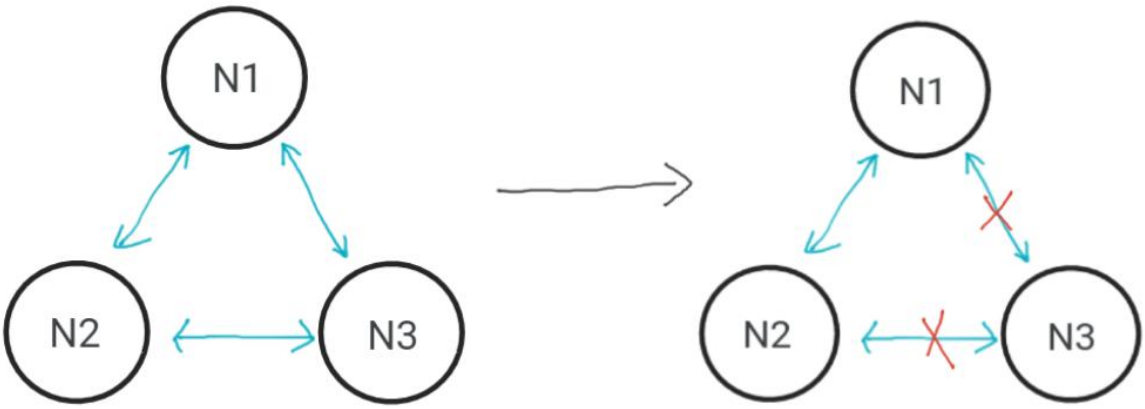
- **Before partition**: We could perform get/add on any node and all operations were consistent due to full connectivity between nodes.
- **During partition**:
  - **Node 3 was isolated** (in the minority) and was unable to complete AtomicLong operations (like get or add) due to lack of a quorum.
  - **Node 1 and 2 remained connected** (in the majority) and were able to perform AtomicLong because they could form a quorum.
- **After healing**: The nodes synchronize and adopt the correct value from the majority. It also performed the add operation that was executed during the partition on the third node.

2. Exercise 2



- **Before partition**: We could perform get/add on any node and all operations were consistent due to full connectivity between nodes.
- **During partition**: Each node was unable to complete AtomicLong operations (like get or add) due to lack of a quorum.
- **After healing**: The nodes synchronize and formed the majority. Each node had applied one local add operation during the partition, and after merging the states we have correct result on every node.

3. Exercise 3



- **Before partition**: We could perform get/add on any node, the same value is visible on each node.
- **During partition**: Each node became isolated and operated independently. Despite the lack of quorum, add operations were still possible thanks to the CRDT-based PNCounter, which allows local updates without coordination. However, the value could temporarily differ across nodes.
- **After healing**: The nodes reconnected and synchronized their states. The local changes made during the partition were automatically merged by the CRDT mechanism, resulting in a final value that reflects all operations performed on any node during the partition.

4. Which data structures -- AP or CP -- require the Raft algorith? Why is this algorithm needed?

   Data structures that fall under the CP (Consistency and Partition tolerance) category typically require the Raft algorithm. It is used to ensure that all nodes agree on the same data, even if some nodes fail or lose connection. Raft keeps everything in sync by electing a leader to manage changes and confirming those changes with a majority of nodes.

5. Read article *Session Guarantees for Weakly Consistent Replicated Data* and explain what it means that the PN Counter in Hazelcast provides *Read-Your-Writes* (RYW) and *Monotonic reads* guarantees and why they are *session guarantees*.

Read-Your-Writes (RYW) ensures that a client sees its own updates immediately within the same session, even if other nodes haven't fully synchronized yet.

- If you increment the counter and then perform a get() within the same session, your update will always be visible. You won't see a value that seems to have missed your write.
- This is called a **session guarantee** because it only applies to the sequence of operations performed in the same session. Outside of that session (e.g., with another client), there's no guarantee that the update will be immediately visible.

Monotonic Reads guarantee that a client's subsequent reads in the same session will not reflect a stale or earlier value than previously observed.
These properties are guaranteed because:

- The storage system enforces them for each read and write operation within a session.
- It can inform the calling application when the guarantees cannot be met.

Why Are These Session Guarantees?

The Read-Your-Writes and **Monotonic Reads** guarantees provided by the **PNCounter** are considered **session guarantees**, because they only apply within the context of a single, continuous interaction (i.e., a session) with the distributed system.