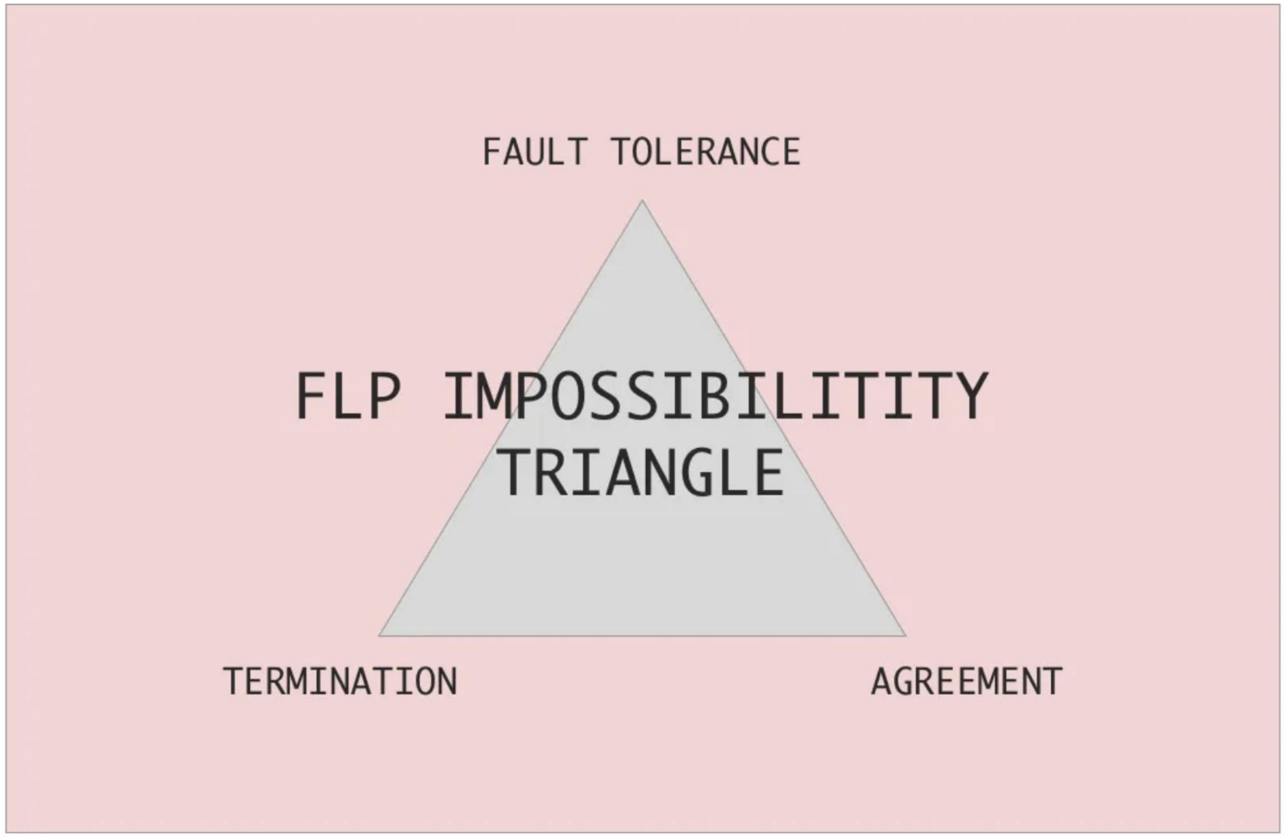
1. True/False

a) In an asynchronous system with processes prone to crash or Byzantine failures, deterministic algorithms cannot implement consensus.

True. See “5.Consensus/ Slide 59”.

Considering FLP impossible theorem, we can only achieve 2 out of 3 properties of consensus with deterministic algorithms. Unless we guarantee 2 properties and then with randomized algorithms, reach to a sacrificed-version of the third property.



FLP theorem: We cannot have all three properties at the same time under the asynchronous network model.

Reference: Practical Understanding of FLP Impossibility for Distributed Consensus | by Melodies Sim | Level Up Coding (gitconnected.com)

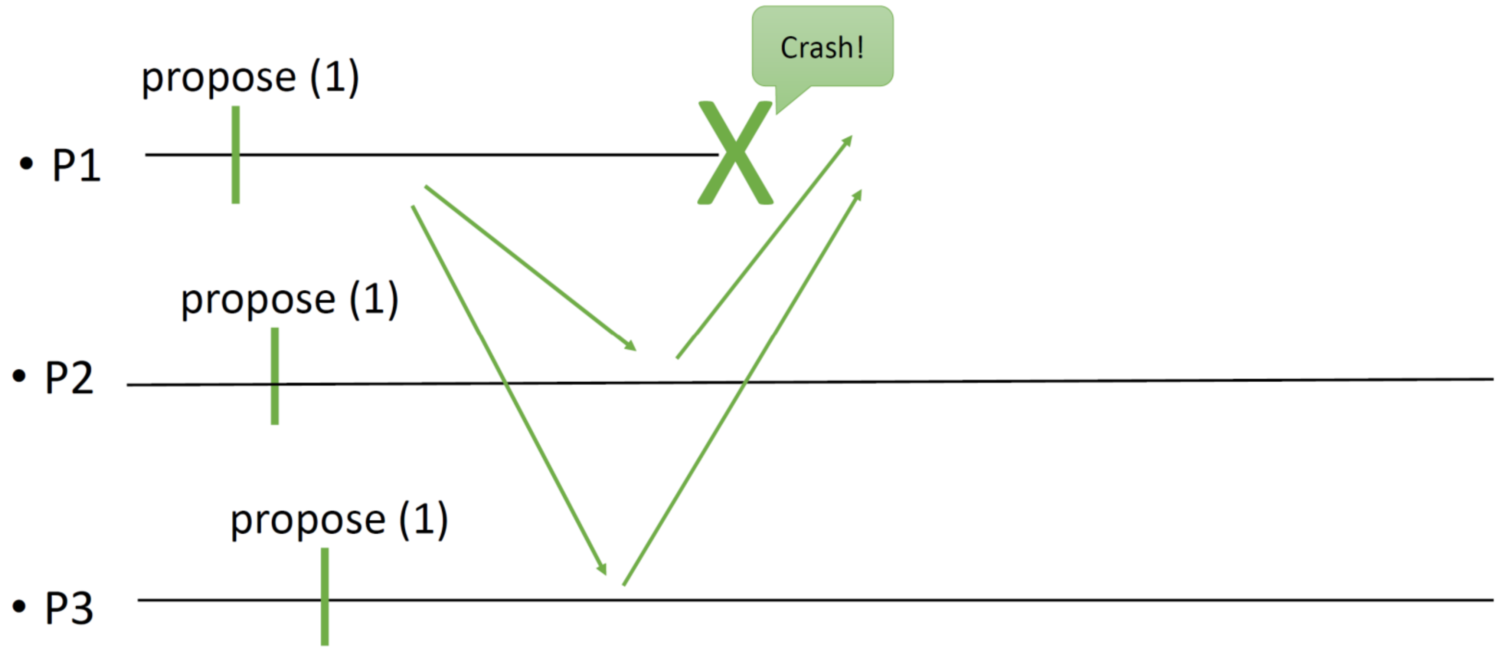
1. True/False

b) The Two-Phase Commit Protocol is a non-blocking commitment protocol.

False. See “8.Atomic-Commit/ Slide 16”.

Week termination: If the leader crashes then the processes are blocked.

I.e., the crash of the coordinator breaks the liveness and termination.

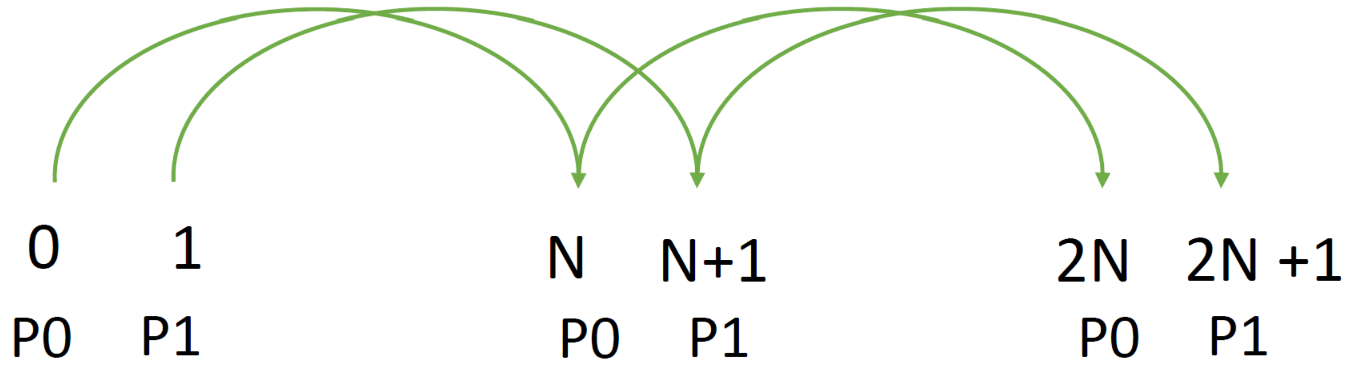


2. In the eventual leader detector abstraction (Ω), explain what happens if we don't partition the space of timestamps in each process? Think about the properties of this abstraction

The consistency property of the abstraction is violated. Consistency guarantees that if a correct process starts an epoch (ts, l) and another correct process starts an epoch (ts', l') with ts = ts', then l = l’. Therefore, not partitioning the space of timestamps results in different leaders or orders of the leaders in different processes.

- However, “before eventually”, Ω does not guarantee the same leader or order of leaders at different processes. This can violate consistency. Therefore, the timestamp domain is disjointly divided between processes.
- For a new leader, we jump to the next timestamp for that process.

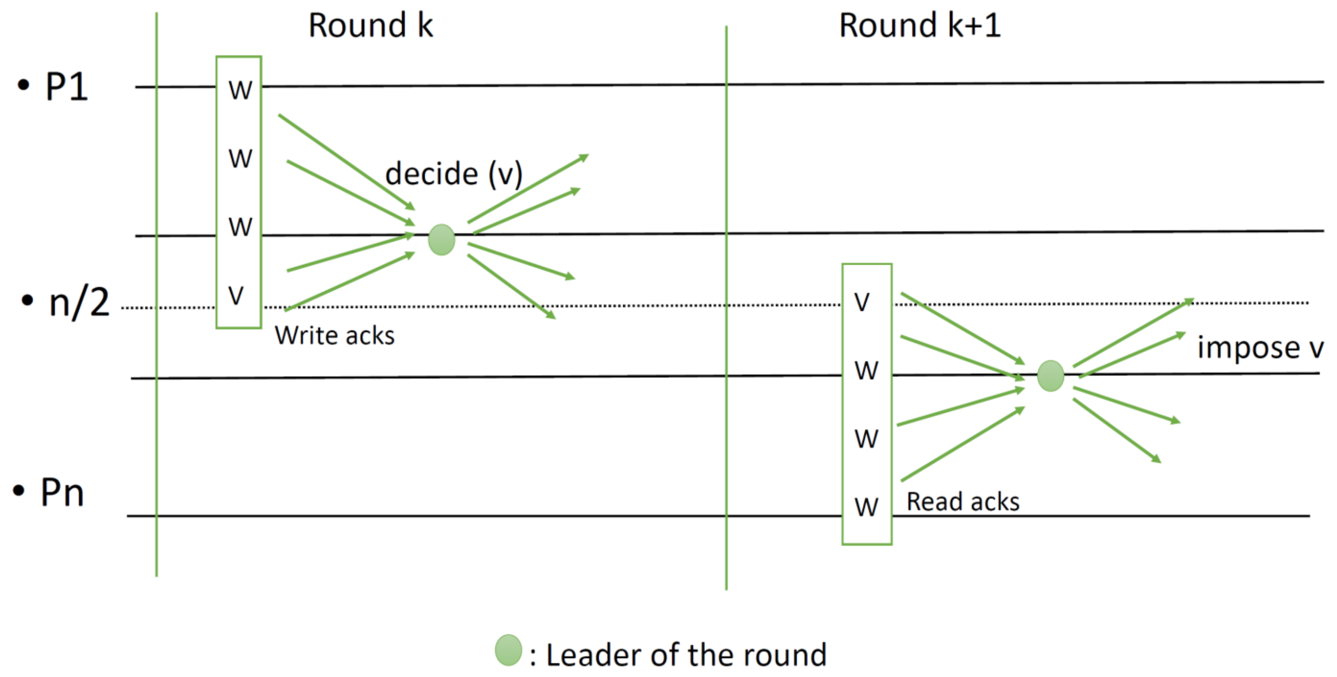
See “5.Consensus/ Slide 63”.



3. Consider Algorithm III (uniform consensus with eventually perfect failure detector). Assume that the process 0, which is the leader of the initial epoch, is correct. How can we simplify the algorithm so that it uses fewer communication rounds?

Since the leader of the first epoch is correct, it may skip the first round of message exchanges for reading because, in the initial epoch, process 0 knows that no decision could have been made in a previous epoch as there is no previous epoch. This first round in every epoch consensus instance is actually only needed to make sure that the leader will propose a value that might have been decided in previous rounds. Therefore, the value will be decided in only three rounds of communications.

Quorums intersect

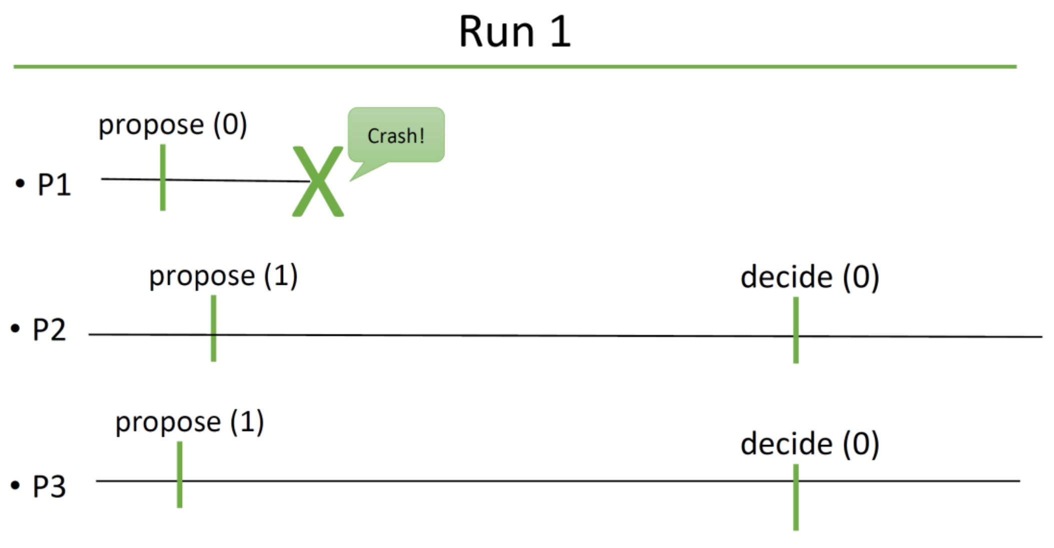


4. Describe a transformation that given uniform total-order broadcast abstraction implements a uniform consensus abstraction. Use the total-order broadcast abstraction with ***broadcast*** and ***deliver*** interface as a blackbox.

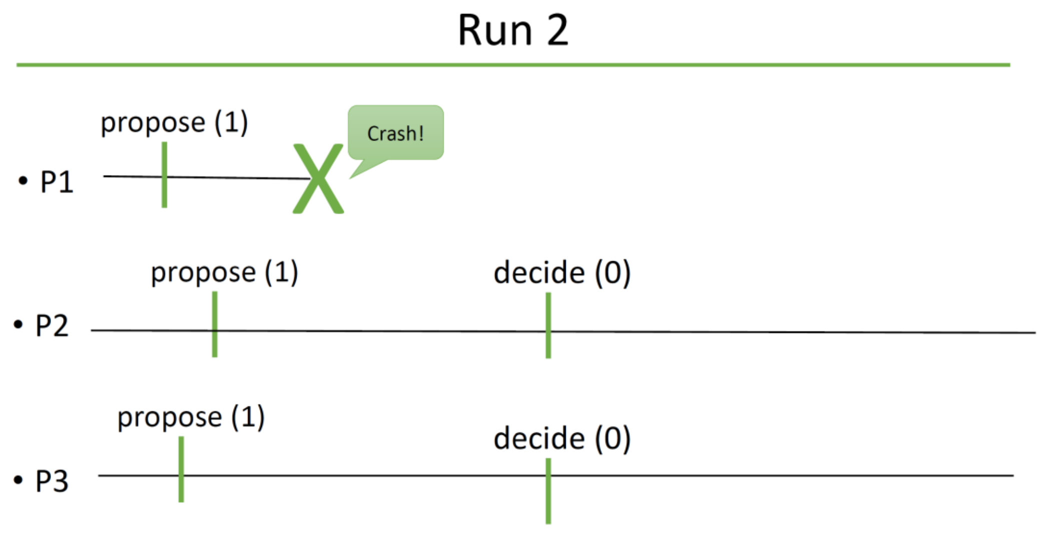
When a process proposes a value v in consensus, we issue broadcast(v). Upon receiving the first message in deliver(x), a process decides x. Since the total-order broadcast delivers the same sequence of messages at every correct process, and every deliver response comes after broadcast, this reduction implements a consensus abstraction.

```
Implements: Uniform Consensus (uCons)
Uses:
        Uniform total-order broadcast (uToB)

Upon event <uCons, propose | v> do
        trigger <uTOB, broadcast | v>

Upon event <uToB, deliver | v> do
        trigger <uCons, decide | v>
```

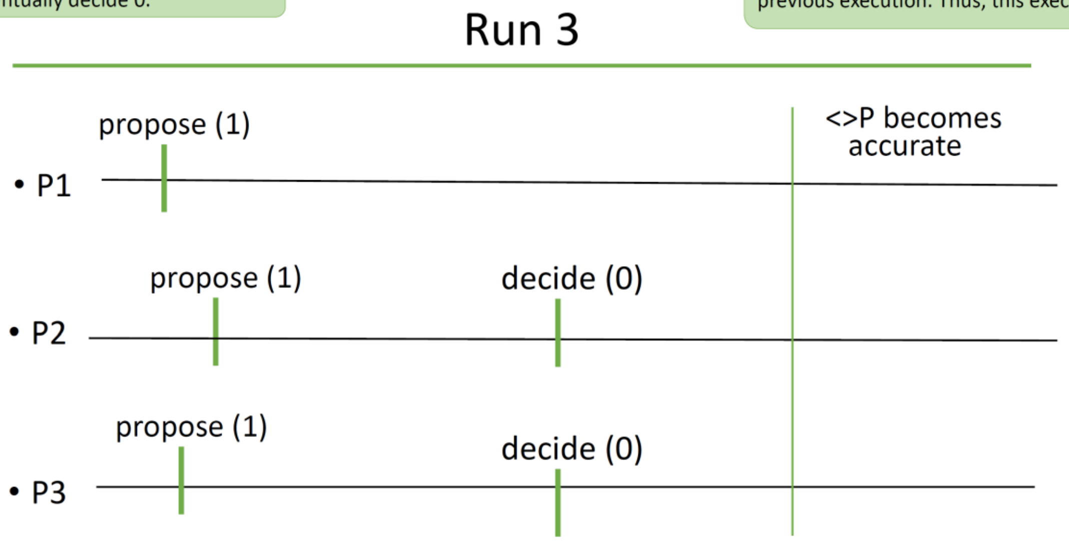
5. Use indistinguishability argument to prove that a non-blocking atomic commit abstraction is not possible with <>P.



The process p1 is proposing 0. Therefore, according to the commit-validity and termination properties, other processes eventually decide 0.



The process p1 is now proposing 1. However, p1 did not send any message in this and the previous execution. Thus, this execution should have the same decision as the previous one.



Now, the process p1 does not crash anymore. Only its messages are slow. Thus, it is still suspected. So the algorithm still decides 0 that violates abort-validity.

TRB:

Use the indistinguishability argument to show that a Terminating Reliable Broadcast (TRB) abstraction cannot be implemented with an eventually perfect failure-detector (<>P) even if only one process can crash.

Consider an execution Run1, in which process s crashes initially and observe the possible actions for some correct process p: due to the termination property of TRB, there must be a time T at which p trb-delivers φ.

Consider a second execution Run2 that is similar to Run1 up to time T, except that the sender s is correct and trb-broadcasts some message m, but all communication messages to and from s are delayed until after time T. The failure detector <>P behaves in Run2 as in Run1 until after time T. This is possible because the failure detector is only eventually perfect. Up to time T, process p cannot distinguish Run1 from Run2 and trb-delivers φ.

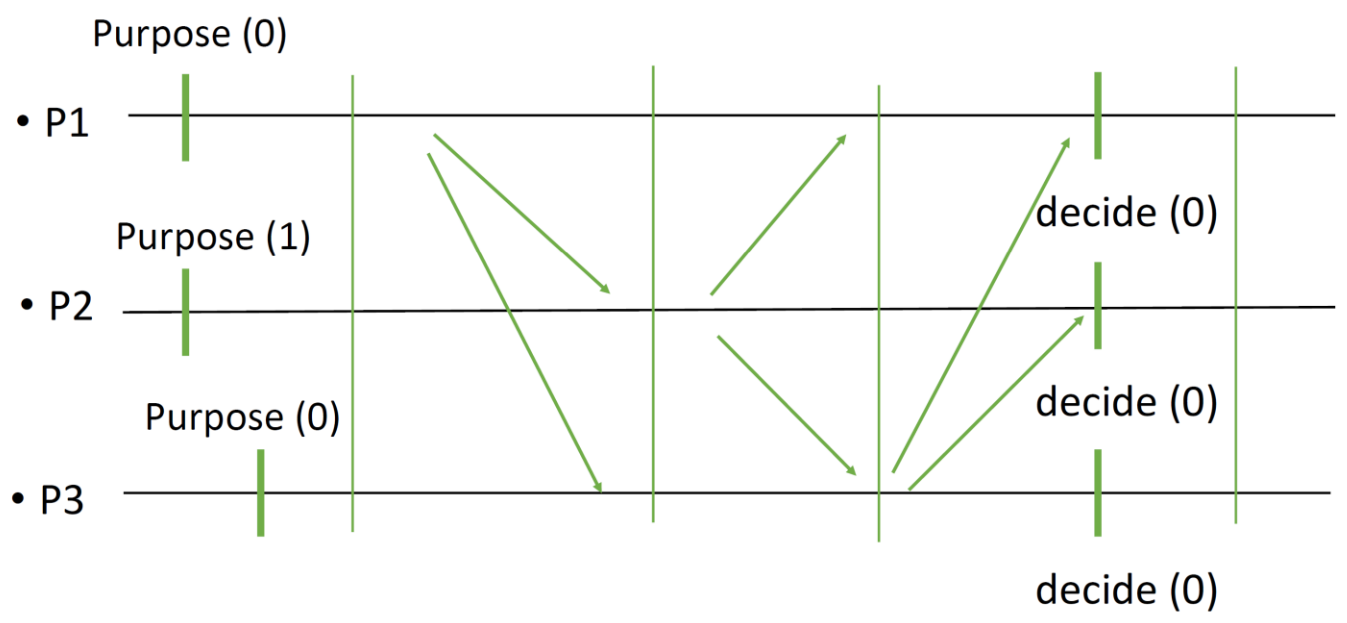
According to the agreement property of TRB, process s must trb-deliver φ as well, and s delivers exactly one message due to the termination property. But this contradicts the validity property of TRB, since s is correct, has trb-broadcast some message m≠ φ, and must trb-deliver m. (It is ok to draw executions instead)

Sample Questions2:

Q.4. Let us consider a variant of the consensus algorithm II (the uniform consensus algorithm that uses a perfect failure detector). In this variant, the last process pn skips broadcasting its value in round n. Is this algorithm correct? Why?

The algorithm still correct. The last process has adopted the values of the previous processes and broadcasting its value does not change their values

Consensus algorithm II



- The processes go through rounds incrementally (1 to n).
- In each round i, process pi sends its currentProposal to all.
- A process adopts any currentProposal it receives.
- Processes decide on their currentProposal values at the end of round n.

The processes exchange and update their proposals in rounds, and after n rounds decide on the current proposal value