# Shared Memory: Impossibility of asynchronous consensus with halting failures using read/write registers

**Linearizable object:** object behaves as though the operations happen instantaneously.

**Definition:** a configuration of a system consists of internal states of all processes, and the states of all shared objects at some moment in the execution (it is basically a snapshot of whole system at that moment)

Assume there is an asynchronous consensus algorithm where processes use only registers to communicate and halting failures may occur.

We say that a configuration C of the algorithm is

- 0-valent, if every execution continuing from C leads to output 0.
- 1-valent, if every execution continuing from C leads to output 1.
- multivalent, if some execution from C leads to output 0 and some other execution from C has output 1.

**Intuition:**

- multivalent configuration - decision has not yet been made.
- 0-valent or 1-valent - decision has been made.

**Fact 1:** once any process has produced an output, the system is not in a multivalent configuration.

**Fact 2:** there exists initial configuration that is multivalent.

Let $C_i$ be the initial configuration of the algorithm when processes $1..i$ have input 0 and the others have input 1.

| | input P1 | P2 | ... | Pn | |
|---|---|---|---|---|---|
| C0 | 0 | 0 | ... | 0 | output 0 |
| C1 | 1 | 0 | ... | 0 | |
| C2 | 1 | 1 | ... | 0 | |
| Cn | 1 | 1 | ... | 1 | output 1 |

Figure 1:

Assume that the initial configuration of each execution is 0-valent or 1-valent, then for some $i$, $C_i$ is 0-valent initial configuration, and $C_{i+1}$ is 1-valent initial configuration. Only process with different inputs in $C_i$ and $C_{i+1}$ is $P_{i+1}$. Please see figure 2. Consider an execution starting from $C_i$ where $P_{i+1}$ takes no steps. Eventually processes will output 0 (since $C_i$ is 0-valent). This same execution can happen starting from $C_{i+1}$, but in this case the output must be 1 (since $C_{i+1}$ is 1-valent). This is a contradiction: there is no way that the same execution can produce two different outputs.

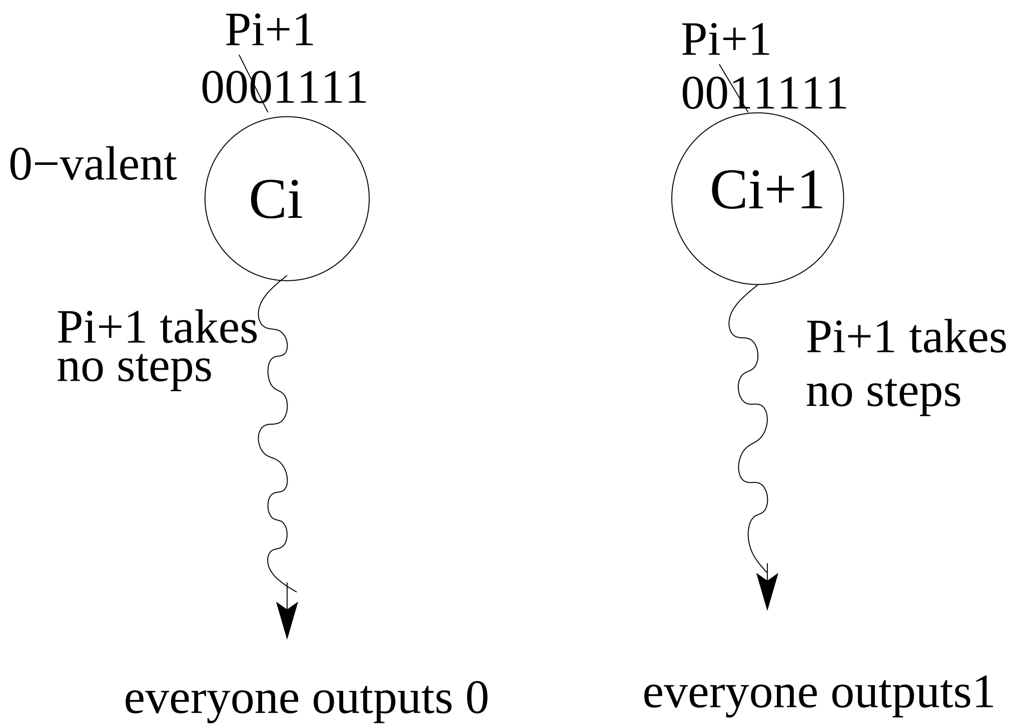


Figure 2:

Definition: A critical configuration is a multivalent configuration such that any step by any process will take the system into a 0-valent or 1-valent configuration. (See figure 3.)

By facts 1 and 2, there exists a critical configuration. If there wasn't, we could construct an execution that never terminates by starting from a multivalent initial configuration and always scheduling the next step to be the step that keeps the system in a multivalent configuration. No process would ever produce an output value in this infinite execution (by Fact 1).

**Intuition:** This is the moment where a decision must be made.

To prove the impossibility result, we show this cannot happen in system with only read/write registers. At least one of successors of $C$ is 0-valent and one is 1-valent (otherwise $C$ wouldn't be multivalent). Let $P$ and $Q$ be processes such that $P$'s step leads to 0-valent configuration and $Q$'s step leads to 1-valent configuration.

**case I**

If $P$ and $Q$'s operations are on different registers, then the configuration obtained from $C$ by having $P$ take 1 step and $Q$ take 1 step is same as configuration obtained by having $Q$ take 1 step and $P$ take 1 step. (Please see figure 4.)

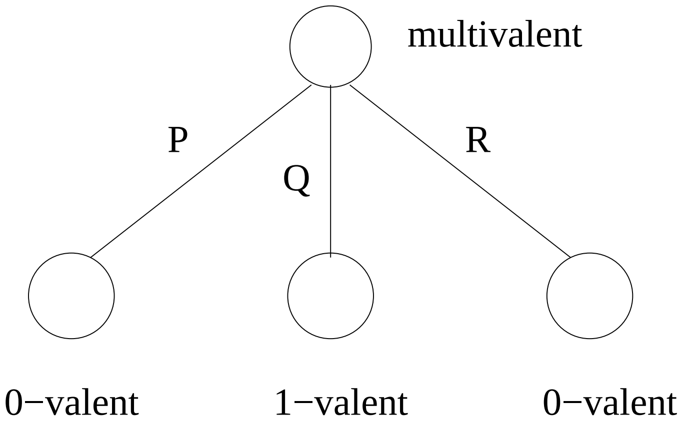


Figure 3: The critical configuration

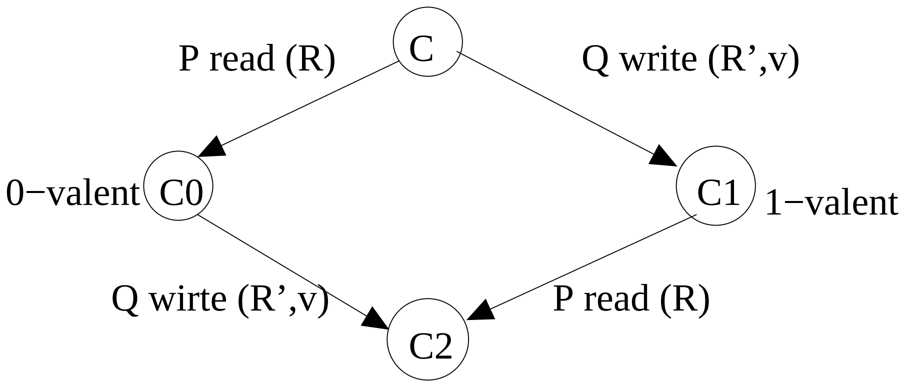


Figure 4: One example of Case I (if $P$ is writing or $Q$ is reading, the diagram would be similar)

$C'$ is 0-valent, since $C0$ is; $C'$ is 1-valent, since $C1$ is. This is a contradiction.
So $P, Q$'s step must be operations on same register.

**case II**

$P$ and $Q$'s operations are on same register and $P$'s step is a read. Please see figure 5.

These two executions are indistinguishable to $Q$ (since all registers, including $R$, have the same state in $C$ and $C0$). $Q$ must output 0 in the left execution. Therefore $Q$ must output 0 in the right execution, contradicting the fact that $C1$ is 1-valent.

**case III**

$P$ and $Q$'s operations are on the same register and $Q$ reads.
Symmetric to Case II.

**case IV**

$P$ and $Q$'s operations are both writes to same register $R$. Please see figure 6

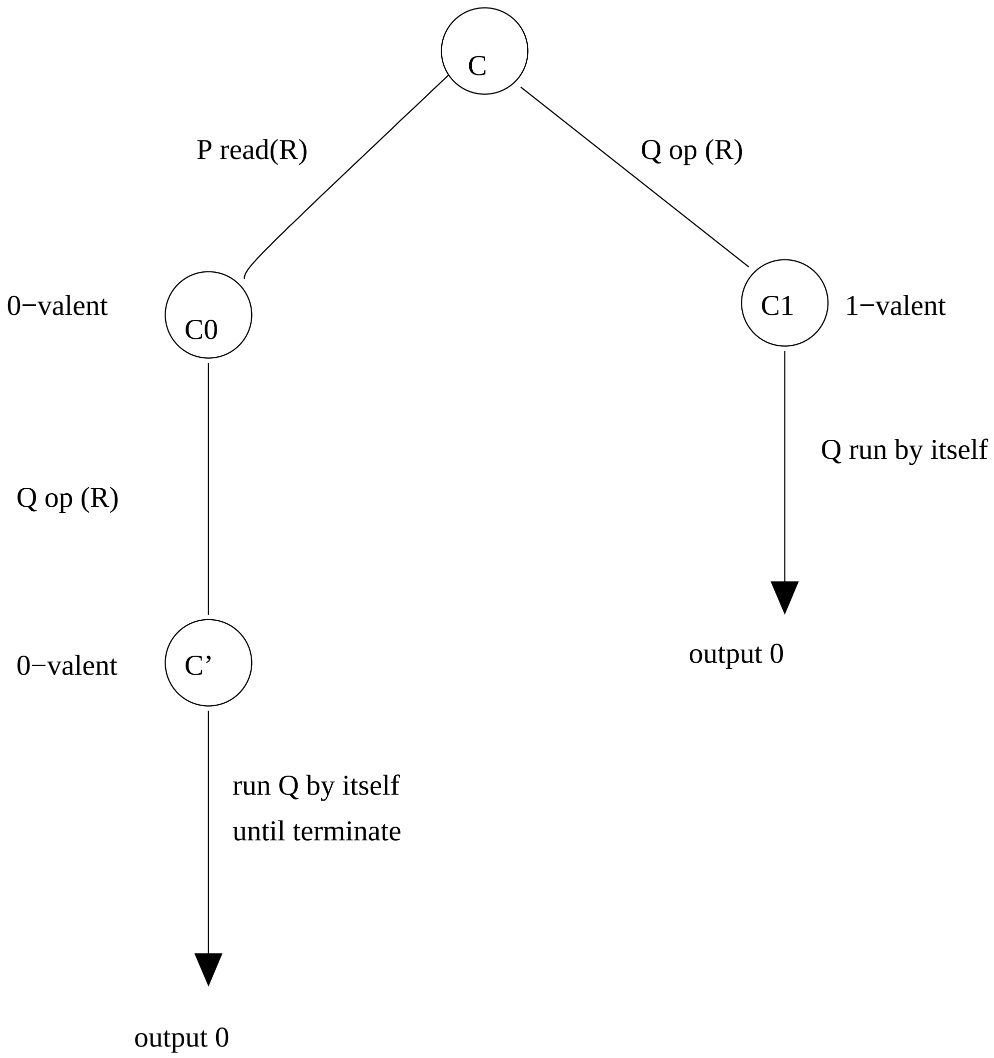


Figure 5: Case II

All registers, including $R$, are in the same state in $C1$ and $C'$. Thus $Q$ cannot distinguish the two executions shown. Yet $Q$ must output different results in the two executions. A contradiction.

Each of four cases gives a contradiction, so initial assumption that there exists a consensus algorithm is false. (Note that, in the proof, we never had to kill off more than one process, so the impossibility result holds even if $f = 1$.)

## Asynchronous message passing with halting failures.

Consensus unsolvable. Prove it by showing that shared registers are at least as powerful as message passing.

Simulate each message channel by a register. If $P$ wants to send a message to $Q$, this is simulated by having $P$ append the message to end of register $R_{PQ}$. To simulate receiving a message from $P$, $Q$ reads register $R_{PQ}$.

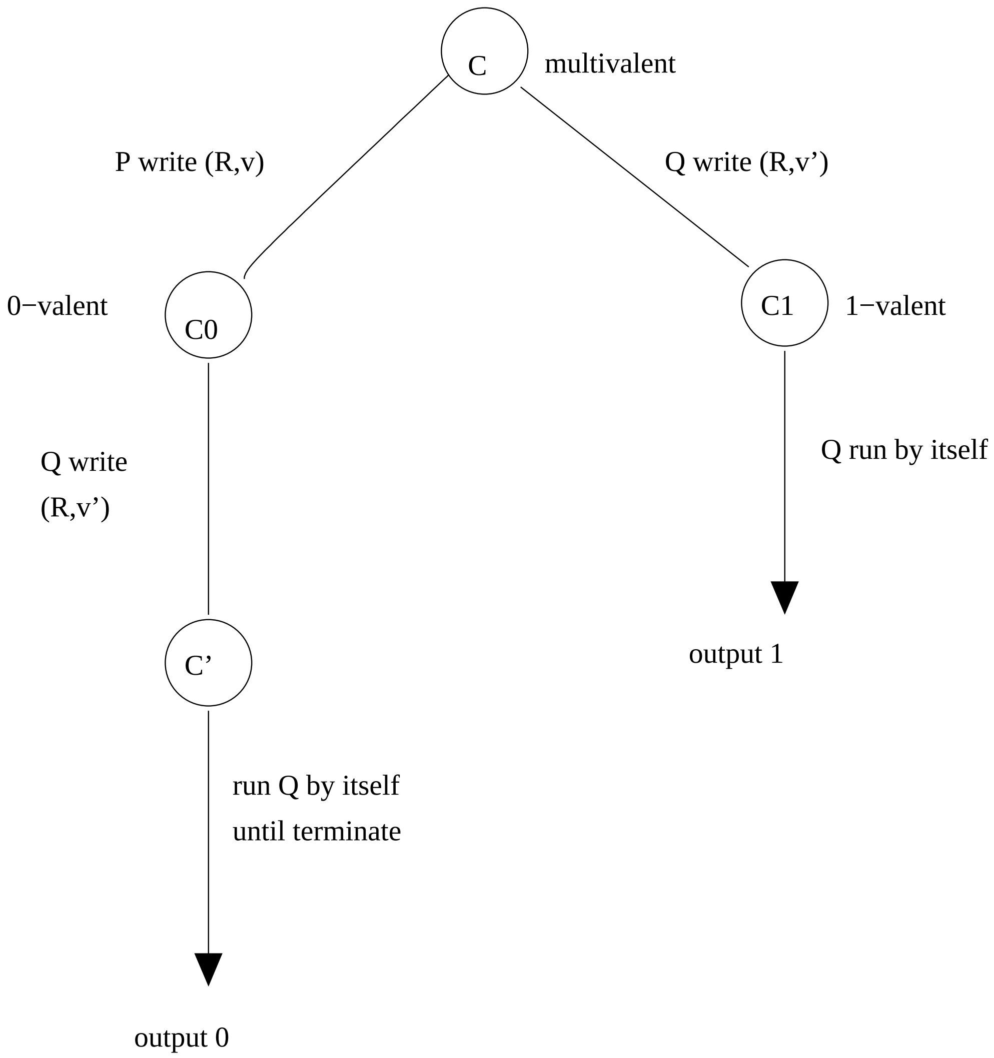


Figure 6: Case IV

So there is no message passing algorithm for asynchronous with halting failure. (If there were one, we could simulate it using registers, which is impossible.)

Hence, it also follows that there is no asynchronous message-passing algorithm for consensus if Byzantine failures can occur, since halting failures are a special case of Byzantine failures.