Soon after the FLP impossibility result appeared, people started to find a way to circumvent it. In 1983, Ben Or gave the first *randomized algorithm* that solve the consensus problem in asynchronous message-passing system with $f < n/2$ crashes with probability 1 ($n$ is the number of processes and $f$ is the maximum number of processes that can crash). The protocol runs forever, but every non-faulty process makes a decision in finitely many steps. With high probability this number of steps is huge ($O(2^{2n})$). In spite of the fact that Ben Or's protocol is not practical the result is important because it indicates that by using randomization the impossibility of consensus can be circumvented.

The model of the distributed system is almost the same: we have a set of $n$ processes, exchange messages in an asynchronous environment. Communication channels are reliable. Each process has access to a *perfect coin*: when a process toss its coin, it obtains 0 or 1 with probability 1/2. The consensus protocol must satisfy the usual conditions, except for the Termination Property, that is slightly relaxed:

- **Integrity**: At most one decision per process.
- **Agreement**: All correct processes must choose the same decision value.
- **Validity**: The value decided by a process must have been initially proposed.
- **(New) Termination**: *With probability 1*, every correct process eventually decides some value.

Every process $p_i$ executes the **randomized consensus algorithm**, as reported in Algorithm 1.

The algorithm works under the assumption that a majority of processes are correct (i.e., $n > 2f$). It is easy to see that this requirement is necessary for any algorithm that solves the consensus problem in asynchronous systems with crash failures, even if all processes have access to a random number generator.

In the algorithm, every message contains a tag (`R` or `P`), a round number $k$, and a value which is either 0 or 1 (for messages tagged `P`, it could also be ?). Messages tagged `R` are called *reports* and those tagged with `P` are called *proposals*. When $p_i$ sends `(R,` $k$, $v$`)` or `(P,` $k$, $v$`)` we say that $p_i$ reports or proposes $v$ in round $k$, respectively.

Each execution of the **while** loop is called a *round*, and each round consists of two asynchronous *phases*. During round $k$ only messages which are timestamped with round number $k$ are processed.

In the first phase, processes report to each other their current estimate (0 or 1) for a decision value.

In the second phase, if a process receives a majority of reports for the same value then it proposes that value to all processes, otherwise it proposes "?". Note that it is impossible for one process to propose 0 and another process to propose 1 in the same round. At the end of the second phase, if a process receives $f + 1$ proposals for the same value different than ?, then it decides that

**Algorithm 1:** The *randomized consensus* algorithm

```
   Input: The initial value v_i of process p_i
   RandomizedConsensus(v_i):
1      x ← v_i                          // p_i's current estimate of the decision value
2      k ← 0                                          // k is the current round number
3      while true do
4          k ← k + 1
5          Send (R, k, x) to all processes
6          Wait for (R, k, *) messages from n − f processes (∗ ∈ {0, 1})
7          if received more than n/2 (R, k, v) messages with the same v then
8              Send (P, k, v) to all processes
           else
9              Send (P, k, ?) to all processes
10         Wait for (P, k, ∗) messages from n − f processes (∗ ∈ {0, 1, ?})
11         if received at least f + 1 (P, k, v) messages with the same v ≠ ? then
12             decide v
13         if received at least one (P, k, v) message with v ≠ ? then
14             x ← v
           else
15             x ← 0 or 1 randomly                                  // toss coin
```

value. If it receives at least one value different than ?, then it adopts that value as its new estimate, otherwise it adopts a random value for its estimate.

The algorithm does not include a halt statement. Moreover, once a correct process decides a value, it will keep deciding the same value in all subsequent phases. However, it is easy to modify the algorithm so that every process decides at most once, and halts at most one phase after deciding.

The Integrity Property is guarantee by the semantics of the **decide** primitive at line 12. We must show the this algorithm satisfy the remaining three conditions.

We now give some definitions.

- We say that *process $p_i$ starts round $k$* if process $p_i$ completes at least $k-1$ iterations of the **while** loop.
- We say that *process $p_i$ reaches line $n$ in round $k$* if process $p_i$ starts round $k$ and $p_i$ executes past line $n-1$ in that round.
- We say that *value $v$ is $k$-locked* if every process that starts round $k$ does so with its variable $x$ set to $v$.

When ambiguities may arise, a local variable of a process $p_i$ is subscripted by $p_i$, e.g., $x_{p_i}$ is the local variable $x$ of process $p_i$.

**Lemma 3.** *If is impossible for process $p_i$ to propose 0 and for process $p_j$ to proposed 1 in the same round $k > 0$.*

*Proof.* We proceed by contradiction. Suppose that process $p_i$ proposes 0 and process $p_j$ proposes 1 in round $k$. Thus $p_i$ receives more than $n/2$ reports equal to 0 and $p_j$ receives more than $n/2$ reports equal to 1 in round $k$. Thus it exists a process $p_k$ that reports 0 to $p_i$ and 1 to $p_j$ in round $k$, but this is impossible. □

**Lemma 4.** *If some process $p_i$ decides $v$ in round $k > 0$, then all the processes $p_j$ that start round $k + 1$ do so with $x_{p_j} = v$, i.e., $v$ is $k + 1$-locked.*

*Proof.* Suppose that some process $p_i$ decides $v$ in round $k$. Process $p_i$ must have received at least $f + 1$ proposals for $v$ in round $k$. Let $p_j$ be any process that starts round $k + 1$: then $p_j$ received $n - f$ proposals in round $k$. Hence $p_j$ receives at least one $v$ in round $k$ since $n - f > f + 1$. By the previous Lemma 3, $p_j$ did not receive $\overline{v}$ in round $k$, hence $p_j$ sets its $x = v$ in round $k$ and $p_j$ starts round $k + 1$ with $x_{p_j} = v$. □

**Lemma 5.** *If a value $v$ is $k$-locked for some $k > 0$, then every process that reaches line 13 in round $k$ decides $v$ in round $k$.*

*Proof.* Suppose $v$ is $k$-locked for some $k > 0$. Then, all reports sent in line 6 of round $k$ are for $v$. Since $n - f > n/2$, every process that proposes some value in round $k$ proposes $v$ in line 8. Consider a process $p$ that reaches line 13 in round $k$. Clearly, $p$ receives $n - f$ proposals for $v$ in round $k$. Since $n - f \geq f + 1$, $p$ decides $v$ in round $k$.

□

**Corollary 1.** *If some process decides $v$ in round $k > 0$, then every processes that executes line 13 in round $k + 1$ decides $v$ in round $k + 1$.*

*Proof.* From Lemma 4 and Lemma 5. □

**Corollary 2** (Agreement)**.** *If some processes $p_i$ and $p_j$ decide $v$ and $v'$ in round $k > 0$ and $k' > 0$, respectively, then $v = v'$.*

*Proof.* For $k = k'$ the result follows from Lemma 3 and the fact that a process can decide a value in a round only if that value was proposed in the same round. Assume that $k < k'$. Since $p'$ decides in round $k'$ then $p'$ reaches line 13 in every round $r$, $k < r \leq k'$. Since $p$ decides $v$ in round $k$, by Corollary 1 $p'$ decides $v$ in round $k + 1 \leq k'$. By additional applications of Corollary 1, we conclude that $p'$ decides $v$ in round $k'$. Each process can decide at most once per round, so $v = v'$.

□

**Corollary 3** (Validity)**.** *If some process $p$ decides $v$, then $v$ is the initial value of some process.*

*Proof.* Note that $v \in \{0, 1\}$. If the initial values of all processes are not identical, then $v$ is clearly the initial value of some process. Now, suppose all processes have the same initial value $w$. Thus, $w$ is 1-locked. From Lemma 5, $p$ decides $w$, and from Corollary 2, $w = v$. □

**Theorem** ((New) Termination)**.**

$$\lim_{k\to\infty} \mathrm{P}[v \text{ is } k\text{-locked during the first } k \text{ rounds}] = 1.$$

*Proof.* By Lemma 5, if some value $v$ is $k$-locked, then $v$ is decided in round $k$. At round 1,

$$\mathrm{P}[v \text{ is 1-locked during the first round}] = \frac{1}{2^n}.$$

At round $k$, the probability that some value $v$ is $k$-locked is *at least* $\frac{1}{2^n}$; indeed some process $p_i$ can set $x_i = v$ not necessarily by flipping a coin. Hence, for any round $k$,

$$\mathrm{P}[\text{no value is } k\text{-locked}] < 1 - \frac{1}{2^n}.$$

Since coin flips are independent,

$$\mathrm{P}[\text{no value is } k\text{-locked for the first } k \text{ rounds}] < \left(1 - \frac{1}{2^n}\right)^k.$$

Thus the probability that $v$ is $k$-locked during the first $k$ rounds is

$$\mathrm{P}[v \text{ is } k\text{-locked for the first } k \text{ rounds}] \geq 1 - \left(1 - \frac{1}{2^n}\right)^k.$$

□