Recall the simple case of n processes attempting to decide on a Boolean value by voting. The problem is easy if all processes are reliable, but theoretically beyond the capability of deterministic processes if even one can fail. We use a "quorum method" due to David Gifford (Stanford, Xerox Parc) 1979. We show that the following simple consensus protocol can tolerate f failures and might reach consensus if $n = 3 \cdot f + 1$ processes vote on values, and if the environment delivers messages in a sufficiently random manner. (The Ben-Or protocol uses enforced randomness to guarantee that consensus can be achieved with high probability.)

We call our protocol *simple consensus* (or the 2/3 protocol). In the general case it is used among clients and n replicas of a resource to guarantee executing a stream of client requests (possibly conflicting) in a specific order. We imagine that an unbounded number of clients are issuing commands *asynchronously* to a resource (imagine a replicated data bases, e.g. Amazon orders, the Google file system, etc.). They want all replicas to see the same execution log, say for command $\mathtt{cmd}_i$ we want each replica to execute them in the same order $\mathtt{cmd}_1$, $\mathtt{cmd}_2$, ..., $\mathtt{cmd}_n$, ... For simplicity, we assume each $\mathtt{cmd}_i$, is a *Boolean value*, 0 or 1. The general case is a simple extension. We also assume that replicas are voting for a *single instance*, the nth command.

For interested students, Mark Bickford and Vincent Rahli have implemented this protocol in Event ML and have provided a closely related version correct (safe) in Nuprl.

G is a group of participating processes, called *replicas*, P_i. Each P_i is identified by its address in the type `Addr`. Thus G can be given by a list of addresses, `(Addr)List`. The protocol is designed to tolerate f failures. Simple consensus requires $3 \cdot f + 1$ processes and relies on a *quorum* of $2 \cdot f + 1$ processes. The quorum allows voting for the consensus value.

Clients *propose* values to processes in G. The proposal has the format $\langle n, c\rangle$ where $n \in \mathbb{N}^+$ and c is a command. The client is proposing that c is the n^{th} command. The *value* is the pair $\langle n, c\rangle$.

The `SC` protocol will consider multiple proposals, but any P_i in G will accept only one client proposal $\langle n, c\rangle$ as the n^{th} command. When P_i receives this proposal, it will ask the group G to *vote* on it. It will collect a quorum of $2 \cdot f + 1$ votes (possible since we assume at most f process can fail). It will see if the votes are *unanimous*, and if so decide that value. Otherwise it starts another round of voting, considering its first proposal as the first round. So its *proposals* have the form $\langle r, v, i\rangle$ where $v = \langle n, c\rangle$, $r \in \mathbb{N}$ is the round number, and i is the address of P_i.

```
@ Replica (i,G) i:Addr, n:  ℕ+, b:  𝔹, f:ℕ+, votes:(ℕ+ x 𝔹) List
newproposal:  rcv(<n,b>) effect initialize(r,v); Voter(r,v)
                where  initialize == r:=0, v:=<n,b>
                       Voter(r,v) = NewRound(r,v,i)
                where  NewRound(<r,v,i>)== SendVote(<r,v,i>); Quorum
                where  SendVote(<r,v,i>) = broadcast(G)(<r,v,i>)
                       Quorum(<r,v,i>) =
                       for j:G do voted(j):= false od; count:=0; votes:=nil
                       while count< 2·f +1 do
                       rcv(<r', v', j>) effect
                           if r'>r then NewRound(<r', v', i>)
                           if r'<r then skip (goto od)
                           else  if voted(j) then skip (goto od)
                                 else  voted(j):= true;
                                       cons(v'; votes)
                                       count:=count+1
                       od
                       if unanimous(Votes) then Notify(value(Votes))
                       else NewRound(<r+1, majority(Votes), i>)
                where  Notify(v) = broadcast(G)(decided(v))
```

start voting at round 0 with value <n,b>

Note, we assume that v′ is a vote for the nth command.

If P$_j$ already voted for nth command ignore this vote.

Note cons(a;L) adds a to the head of list L.

NewRound is called on round r+1, P$_i$ votes for the majority which exists for $2 \cdot f + 1$ an odd number, $b \in \mathbb{B}$.

Note votes is ($\mathbb{N}^+$ x $\mathbb{B}$) List and value is the unanimous value of the list.

Clients will propose a pair $\langle n, \texttt{cmd}\rangle$, a *proposal* that command cmd be number n. The proposal is made to a group G of replicas P_i located at some address $\texttt{loc}_i$. The P_i will vote on which command is the n^{th}. They might be voting simultaneously on several proposals.

We require the *agreement property* that if G decides on the n^{th} command, then all processes that have not failed reach the same decision. We also prove a *liveness property*, that for any state of the protocol, it is possible to reach agreement by some choice of the delivery of messages, e.g. some action of the environment.

Suppose SC decides at two or more locations, consider two of them P_i, P_j. Suppose P_i decides in the lower round if $r \neq r'$.

P_i decides v in round r at event d_i.

So P_i sees $2 \cdot f + 1$ unanimous (think 3 of 4) in round r, so at least $2 \cdot f + 1$ voted for v in this round.

Say at a later or equal round $r' \geq r$ P_j votes for $v' \neq v$, then (3 votes for v')

If P_j participates in this round r and sees a unanimous value of $2 \cdot f + 1$ votes (say 3 votes) then one of these must be the same as the vote at P_i thus $v = v'$.

picking two sets of $2 \cdot f + 1$ values from $3 \cdot f + 1$ values, they must over lap, would need	$2 \cdot f + 1$ $2 \cdot f + 1$ $- - - -$ $4 \cdot f + 2$	values to have disjoint unanimous. (are $f + 1$ short!)

Any process participating at round r will eventually collect at least $f + 1$ votes from this group, a majority, so it will vote for v as well and thus in a higher round r' if one occurs.

1 Liveness

If f processes fail or are very slow, then only $2 \cdot f + 1$ participate, an odd number, so they can't tie (in binary case). So the environment can arrange a decision.

More generally,

FLP is a way to get stuck, likewise if all $3 \cdot f + 1$ replicas receive a different command, then it is possible that no decision is possible.