Problem of agreement
Consensus
with crashes

Janna Burman

janna.burman@lisn.fr

Consensus - motivation

- **Fundamental problem** in distributed computing
- The processes must **agree on a common data value**, which is then used in a computation/task in the presence of faults

Applications:

- leader election
- distributed pair-to-pier databases, based on blockchain or more conventional ones (to decide whether or not to record a transaction)
- clock synchronisation
- load balancing in parallel computing systems
- etc.

Assumptions

1. **complete undirected communication graph**
2. n processes 1, ... n, completely knowing the network
3. each process has an **initial value** belonging to a fixed set of values **U**
4. perfectly **synchronous** communication **model**
5. **communication links** are assumed to be **reliable** - every sent message is received
6. **any process can crash**: stop running at any time (but not in the middle of sending a message) and forever after; without this being observable or detectable by other processes
7. let **f** < n be the **maximum number of the crashed/faulty processes** (in every execution)

The specification (definition) of *consensus*

1. ***Agreement*:** two processes never decide different values.
2. ***Validity*:** if all processes have the same initial value v ($\in$ U), v is the only possible decision value.
3. ***Termination*:** every non-faulty process eventually decides a value.

The **decision** is made only once. It is irreversible.

Consensus – solution?

- The problem is simple to solve **in the absence of faults**
  - each process broadcasts its initial value, and when a process has received all the values, it decides, for example, on a maximum value (in one round)
- If even only one crash is possible, the previous solution no longer works (in one round).
- However, **a solution is possible in a synchronous network**.

What if the communication is purely asynchronous?

Theorem: In an **asynchronous network** (e.g., where the time to deliver a message is finite, but unbounded), even **with only one possible crash**, **there is no deterministic solution to the consensus problem.**

Flood-Set algorithm

- Each process maintains a variable W containing a subset of U. Initially, variable W of process p_i contains only the initial value of p_i.
- In each of the f+1 rounds, each process broadcasts W and adds all the received sets to W.
- At the end of round f+1, every process p_i applies the following decision rule: if W is a singleton, then p_i decides this singleton, otherwise p_i decides a default value v_0 ($\in$ U).

Flood-Set - pseudo-code (1)
for a process p_i

M (possible message set) = P(U) = 2^U
(powerset - set of all subsets of U)

states$_i$ + start$_i$ (possible states + initial states):

rounds: $\mathbb{N}^+$(positive integers); rounds := 1

decision: U ∪ {unknown}; decision := unknown (written only once)

W: P(U) (W ⊆ U); W := {v_i},
where v_i is the initial value of p_i

Flood-Set - pseudo-code (2)
for a process p_i

msgs$_i$ (message generation function, applied at the beginning of each round):

If (rounds $\leq$ f+1) then generate (send) W to every (neighbouring) process

Flood-Set - pseudo-code (3)
for a process p_i

trans$_i$ (transition function, applied at the end of each round):

Let Xj ($\in$ P(U)) be the message received from neighbour p_j

W := W$\cup\bigcup_{\text{over all pj}}$Xj (gather in W all the received values)

If (rounds = f+1) then

If (|W| = 1) then

decision := v such that W={v}

Else decision: = v_0 ($\in$ U, pre-defined value) EndIf

EndIf

If (rounds $\neq$ f+1) then rounds: = rounds + 1

Flood-Set – pseudo-code (3.a)
for a process p_i

With the stronger Validity condition: a process can only decide on an initial value of one of the processes:

trans$_i$ (transition function, applied at the end of each round):

Let Xj ($\in$ U) be the message received from neighbour p_j

W := W $\cup\bigcup_{\text{over all pj}}$Xj (gather in W all the received values)

If (rounds = f+1) then

decision: = min(W)

EndIf

If (rounds $\neq$ f+1) then rounds: = rounds + 1

Flood-Set – proof scheme

The correctness is based on 2 main points:

1. **At the end of a round with no faults**, **each non-faulty** (*correct*) **process** up to this round **obtains the same** set of values **W** and from this round onwards, W of any correct process does not change anymore.
   - as every value known to the correct processes is transmitted

2. **Among the f+1 rounds, at least one is without faults**
   - otherwise, in each round among f+1, there is at least one crashed process; then there are at least f+1 faulty processes, contradicting the assumption of an upper bound f on the number of faulty processes

Flood-Set – proof (1)

- Let **Wi(r)** be the value of the variable **W of process i (p_i) at the end of round r**
- A **process** is **active after r rounds** if it did not crashed during the first r rounds.
- Let **A(r)** be the **set of active processes at the end of round r**

Lemma 1: If no process crashes during a round r, $1 \leq r \leq f+1$, then Wi(r)= Wj(r) for every two active processes i and j, after r rounds.

Proof: $\forall\ p_i \in A(r)$: $Wi(r) = \cup Wx(r-1)$ for every $x \in A(r-1)$, thus Wi(r) has the same value for any process i at the end of round r

Flood-Set – proof (2)

Lemma 2: If Wi(r)= Wj(r) = w for every two active processes i and j after r rounds, then for any round r', r≤ r'≤ f+1, Wi(r')= Wj(r) = w for every two active processes i and j after r' rounds.

Proof: Let us consider any process p_i, and prove by induction on r'.
This is trivial for r'=r. Let us assume the correctness for an r' and prove for r'+1: for any neighbour p_j of p_i and the message Xj received by i from j, at r'+1: Xj is either empty or contains Wj(r') = Wi(r'),

thus $Wi(r'+1) = Wi(r') \cup \bigcup_j Xj = Wi(r')$.

Lemma 3: If processes i and j are active after f+1 rounds, then Wi(r)= Wj(r) at the end of round r = f+1.

Proof: Among f+1 rounds, at least one is without crashes. Otherwise, in each round among f+1, there is at least one crashed process; then there are at least f+1 faulty processes, contradicting the assumption of the upper bound f.
Then, by Lemmas 1 and 2, the lemma is correct.

Flood-Set – proof (3)

Lemma 4 (Validity): if all processes have the same initial value v, v is the only possible decision value.

Proof: If all processes have the same initial value v, then $\forall\ p_i \in A(0)$: $W_i(0) = \{v\}$ ➔ by Lemma 2, $W_i(f+1) = \{v\}$, for all $p_i \in A(f+1)$ ➔ $|W_i(f+1)| = 1$ and the decision is v.

Theorem: The Flood-Set algorithm solves consensus with crashes (with parameter f).

Proof:

Termination: Every active process decides in round f+1.

Validity: by Lemma 4.

Agreement: By Lemma 3, for every two active processes i and j at the end of f+1 rounds, $W_i(f+1) = W_j(f+1)$ ➔ thus they decide the same value.

Flood-Set bit complexity – exercise

Calculate the worst-case FloodSet complexity in terms of transmitted bits.

- Let us assume that each initial value in U can be encoded by at most b bits.

Problem of agreement, Consensus

with unreliable communication links

Assumptions

- **all processes are correct** but the **communication links can lose an arbitrary, even infinite, number of messages**
- the initial values of the processes belong to the set $U = \{0, 1\}$

- the Termination and Agreement conditions remain the same
- a new (less restricted) Validity condition:
  Validity':
  1. If all processes have the same initial value 0, 0 is the only possible decision value.
  2. If all processes have the same initial value 1 **and no messages are lost**, 1 is the only possible decision value.

Impossibility (1)

Theorem: In a network with only two processes (p1 and p2), the consensus problem (with **Validity'**) does not admit a solution, with the assumptions above.

Proof: By contradiction ➔ an algorithm A exists.

1. Let us take **an execution e** in which p1 and p2 have an initial value of 1, and no message is lost. By Termination, p1 and p2 eventually decide. By Validity', they decide 1. Let us suppose they decide after r rounds.
2. Let us take **an execution e1**, which is like e, but in which every message after round r is lost. In e1, p1 and p2 also decide 1.

Proof of Impossibility (2)

Proof (cont.) :

3. Let us consider **an execution e2**, which is like e1, but the last message (in round r) from p1 to p2 is lost. For p1, e1 and e2 are identical (the same state sequences and messages in both executionsS). Thus, p1 decides 1. ➔ By agreement, p2 also decides 1.

4. Let us consider **an execution e3**, which is like e2, but the last message (in round r) from p2 to p1 is also lost. For p2, e2 and e3 are identical. Thus, p2 decides 1. ➔ By agreement, p1 also decides 1.

5. Let us continue in the same way until we get **an execution e'** where all messages are lost. By the same arguments, p1 and p2 decide 1.

Proof of Impossibility (3)

Proof (cont.) :

6. Let us consider **an execution e''**, which is like e', but in which the initial value of p2 is 0. For p1, e' and e'' are identical. Thus, p1 decides 1. ➔ By agreement, p2 also decides 1.

7. Let us consider **an execution e'''**, which is like e'', but in which the initial value of p1 is 0. For p2, e'' and e''' are identical. Thus, p2 decides 1. ➔ By agreement, p1 also decides 1. Every initial value is 0, but processes decide 1 ➔ **contradiction to Validity'**

Questions

1. What if the Validity condition remains as before?

2. What if only a finite, but non-bounded number of messages can be lost?

3. …

Problem of agreement
Consensus
with Byzantine faults

Assumptions

1. **complete undirected communication graph**
2. n process 1, ... , n, completely knowing the network
3. each process has an **initial value** belonging to a fixed set of values **U**
4. perfectly **synchronous** communication **model**
5. **communication links** are assumed to be **reliable** - every sent message is received
6. a process can be faulty (incorrect), referred to as **Byzantine**
   - it can have a behaviour deviating from the protocol of the solution/algorithm (**arbitrary behaviour**): it can start in any state, send arbitrary messages and not follow the transition function
   - but cannot control/influence other entities in the system
7. let **f** < n be the **maximum number of faulty processes** (in any execution)
8. **n > 3f**

The specification (definition) of the *Byzantine agreement problem*

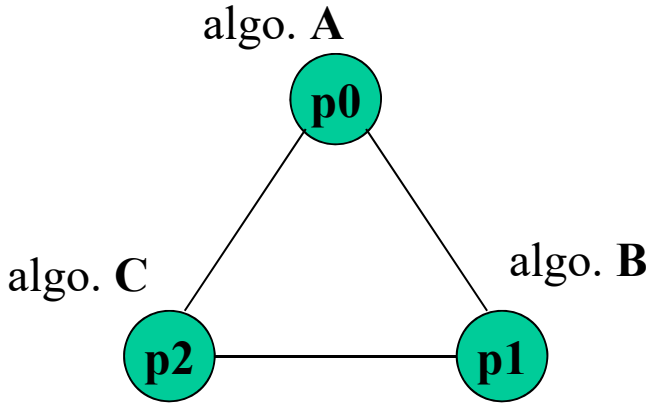
1. ***Agreement*:** two non-faulty/correct processes never decide on different values.
2. ***Validity*:** if all non-faulty/correct processes have the same initial value v ($\in$ U), v is the only possible decision value for the correct processes.
3. ***Termination*:** every non-faulty process will eventually decide on a value.

The **decision** is made only once. It is irreversible.

Impossible with n = 3 and f = 1 (1)

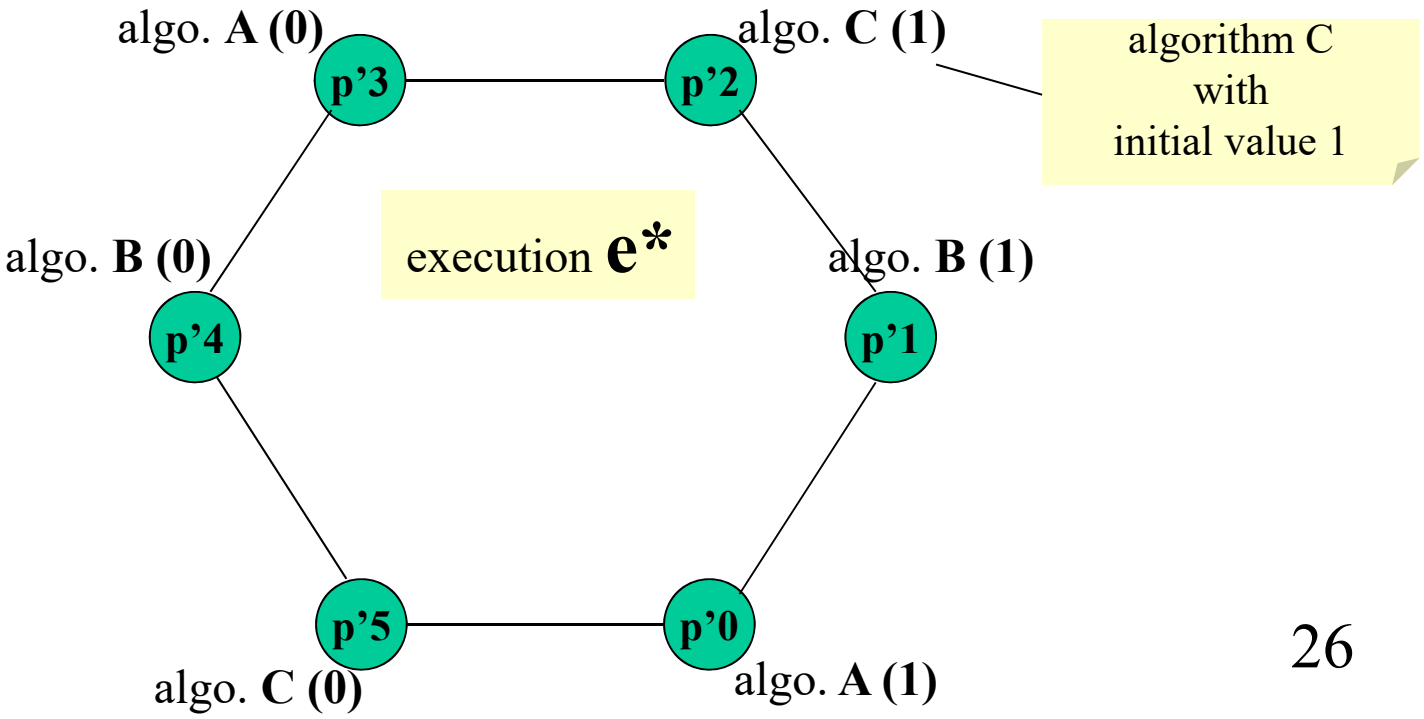
Theorem: In a network with 3 processes and one Byzantine, there is no solution to the consensus problem.

Proof: By contradiction, an algorithm exists in a system **with 3 processes p0, p1, p2 with local algorithms A, B, and C** respectively**.**

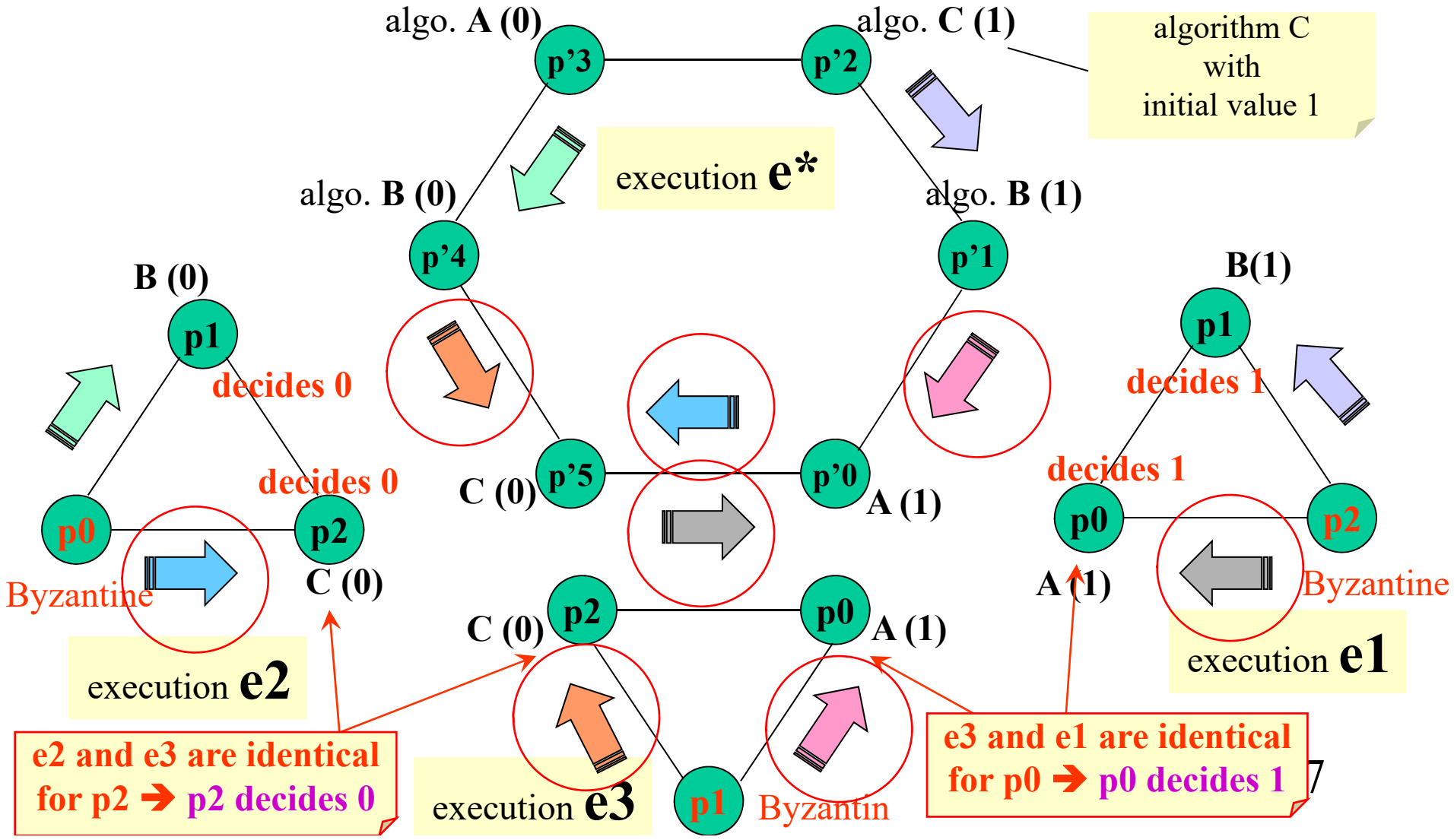


Proof of impossibility (2) with n = 3 and f = 1

Let us take **a synchronous ring system with 6 correct processes:** p'0 and p'3 with local algorithm A, p'1 and p'4 with local algorithm B, and p'2 and p'5 with local algorithm C. Let us consider an **execution e*** with the initial values shown in the figure.



Proof of impossibility (2) with n = 3 and f = 1



Impossible with $\mathbf{n \leq 3f}$ (1)

Theorem: In a network with n processes and f Byzantine processes, there is no solution to the consensus problem if $\mathbf{n \leq 3f}$.

Proof: By contradiction, such an algorithm A exists.

We show that such an algorithm (assumed by contradiction) can be used to solve consensus in a 3-process system p0, p1 and p2, with a Byzantine (to obtain the contradiction to the previous theorem).

- Let us divide n processes into three sets P0, P1 and P2, each with at most n/3 processes.
- We design an algorithm A' for p0, p1 and p2. In A', p0 simulates all processes in P0 according to algorithm A, p1 simulates all processes in P1 and p2 all processes in P2.
- If any process pi among p0, p1 and p2 is Byzantine, then because $n/3 \leq f$, at most f processes are faulty in the simulated system. Hence, A solves consensus in the simulated system and thus also in the 3-process system. The designed algorithm A' solves consensus in a 3-process system with a Byzantine one ➔ contradiction to the previous theorem!

Reliable broadcast
formal definition in 3 conditions

Used to ***broadcast/disseminate*** (send to all) reliably (consistently) a message (**m**, i, r) by a process p_i in round r, with payload m, and which will be ***accepted*** by processes in subsequent rounds.

Formally:

I. If a correct (non-faulty) process disseminates a message (m,i,r) in round r, the message is accepted by any correct process until round r+1

II. If a process p_i does not disseminate a message (m,i,r) in round r, (m,i,r) is never accepted by a correct process.

III. If a message (m,i,r) is accepted by a correct process p_j in round r', it is accepted by every other correct process (at least in round r'+1 or earlier)

Reliable Broadcast
algorithm

1. **To disseminate a message (m,i,r)** in round r, process p_i sends a message ("init", m, i, r) to every process in round r.

2. If a process p_j receives a message ("init", m, i, r) **from process p_i in round r** then it sends ("echo", m, i, r) to every process (including itself) in round r+1.

3. If before any round r' ≥ r+2, a process p_j receives a message ("echo", m, i, r) from **at least f+1 processes**, it sends ("echo", m, i, r) to every process (including itself) in round r' (if it has not already sent before).

4. If until round r' ≥ r+1, a p_j receives ("echo", m, i, r) from **at least n-f processes**, p_j **accepts** the message (m,i,r) in round r' (if it has not already done so).

Reliable Broadcast
proof of condition I

I. If a correct process disseminates a message (m,i,r), then **let us prove that** the message is accepted by every correct process until round r+1:

To disseminate (m,i,r), *p_i sends ("init", m, i, r) to every process in round r* → *every correct process among at least n-f correct processes sends ("echo", m, i, r) to every process in round r+1. Then, at the end of round r+1, every correct process receives ("echo", m, i, r) from at least n-f processes and thus accepts the message (m,i,r).*

Reliable Broadcast
proof of condition II

II. If a process p_i does not disseminate a message (m,i,r) in round r,
then **let us prove that** (m,i,r) is never accepted by a correct process:

If p_i does not disseminate (m,i,r), p_i does not send a message ("init", m, i, r). Thus, no correct process sends ("echo", m, i, r) in return → a correct process can receive ("echo", m, i, r) from f Byzantine processes.
However, $f < f+1 < n-f$ (since $n > 3f$), no correct process sends ("echo", m, i, r) in return to such f messages, and thus never accepts (m,i,r).

Reliable Broadcast
proof of condition III

III. If a message (m,i,r) is accepted by a correct process p_j in round r', then **let us prove that** it is accepted by any correct process (at least in round r'+1):

If (m,i,r) is accepted by p_j in r', p_j has received ("echo", m, i, r) from at least n-f processes until round r'. Among these n-f processes, there are at least n-2f ≥ f+1 correct processes. Each such process sends ("echo", m, i, r) to every process → thus, each correct process receives at least ("echo", m, i, r) from f+1 processes up to r' and then sends ("echo", m, i, r) to any process in round r'+1 → any correct process thus receives ("echo", m, i, r) from at least n-f processes up to round r'+1 and then accepts (m,i,r)