Distributed algorithms: Lesson 2 on the consensus abstraction

Emmanuelle Anceaume

emmanuelle.anceaume@irisa.fr
http://people.irisa.fr/Emmanuelle.Anceaume/

Content of this lesson

- Focus on the Consensus abstraction in synchronous environments
  1. Specification of the consensus abstraction
  2. An algorithm that tolerates crash failures
  3. Lower bound on the number of rounds needed to tolerate f crash failures
  4. An algorithm that tolerates Byzantine failures
  5. Lower bound on the number of Byzantine processes

Byzantine generals metaphor

The Byzantine generals metaphor of the consensus problem

Several divisions of the Byzantine army are waiting outside an enemy city. Each division is commanded by a general. Generals can communicate with reliable but possibly slow messengers.

Each general should eventually decide on a plan, and this plan should be common : attack the city or not, and if the generals are unanimous in their initial opinion, then that opinion should be the decision.

Some of the generals may be traitors, and may try to prevent the loyal generals from agreeing

- Traitors send conflicting messages to different generals, falsely report on what they have heard from other generals, and even conspire and form a coalition

The Consensus abstraction [1]

Consensus definition

- Agreement : All non-faulty processes must agree on the same single value
- Validity : If all processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value
- Termination : Each non-faulty process must eventually decide on a value

1. Leslie Lamport, Robert E. Shostak and Marshall C. Pease, *The Byzantine generals problem*, ACM Trans. Program. Lang. Syst, Vol(4), 1982

Model of the system

- Set of n processes, and among them f are faulty
  - Recall that a faulty process is a process whose functionality is incorrect
  - We will consider two types of failures : crash and Byzantine
  - Recall that it is not known in advance which processes are faulty
- Message-passing model : processes communicate and synchronize by exchanging messages
- Communication graph is complete
- Synchronous system : existence and knowledge of temporal bounds on communication and processing step

A simple consensus algorithm in the crash model (1)

```
propose(v_i) {// algorithm run by process p_i
Initially V_i = {v_i}

   1:round k, 1 ≤ k ≤ f + 1
   2:   send S_i = {v ∈ V_i: p_i has not already sent v} to all
        processes
   3:   receive S_j from p_j
   4:   V_i := V_i ∪ ⋃_{j=1}^{n} S_j
   5:y= min(V_i)
   6:return decide(y)
}
```

Algorithm 1

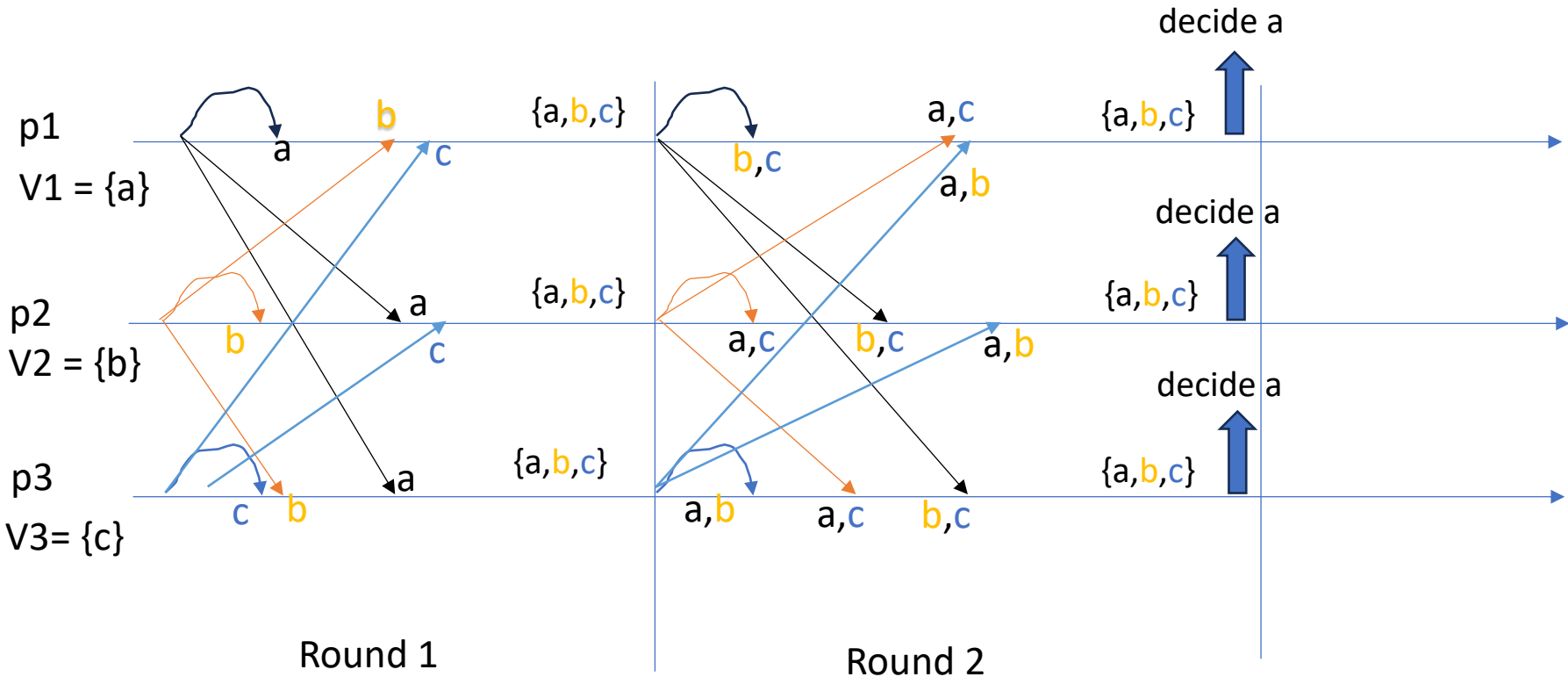
A simple consensus algorithm in the crash model (2)

```
propose(v_i) {// algorithm run by process p_i
Initially V_i = {v_i}

   1:round k, 1 ≤ k ≤ f + 1
   2:   send S_i = {v ∈ V_i: p_i has not already sent v} to all
        processes
   3:   receive S_j from p_j
   4:   V_i := V_i ∪ ⋃_{j=1}^{n} S_j
   5:y= min(V_i)
   6:return decide(y)
}
```

Algorithm 1

- Each process p_i maintains a set of values V_i. Initially, V_i contains the input value of p_i
- In later rounds, p_i updates V_i with the values received from the other processes, and broadcasts any value it has not already broadcast
- Once p_i has executed $f + 1$ rounds, it decides the smallest value in its set V_i



A failure-free scenario of Algorithm 1 with n=3, f = 1

Correctness proof of Algorithm 1 (1)

```
propose(v_i) {// algorithm run by process p_i
Initially V_i = {v_i}

   1:round k, 1 ≤ k ≤ f + 1
   2:   send S_i = {v ∈ V_i: p_i has not already sent v} to all
        processes
   3:   receive S_j from p_j
   4:   V_i := V_i ∪ ⋃_{j=1}^{n} S_j
   5:y= min(V_i)
   6:return decide(y)
}
```

Algorithm 1

- Termination : Each non-faulty process must eventually decide on a value
  - Proof : From the code, the algorithm requires $f + 1$ rounds
  - At the end of round $f + 1$ all processes decide some value

Correctness proof of Algorithm 1 (2)

```
propose(v_i) {// algorithm run by process p_i
Initially V_i = {v_i}

   1:round k, 1 ≤ k ≤ f + 1
   2:   send S_i = {v ∈ V_i: p_i has not already sent v} to all
        processes
   3:   receive S_j from p_j
   4:   V_i := V_i ∪ ⋃_{j=1}^{n} S_j
   5:y= min(V_i)
   6:return decide(y)
}
```

Algorithm 1

- Validity : If all processes have the same initial value v, then all the non-faulty processes must decide v

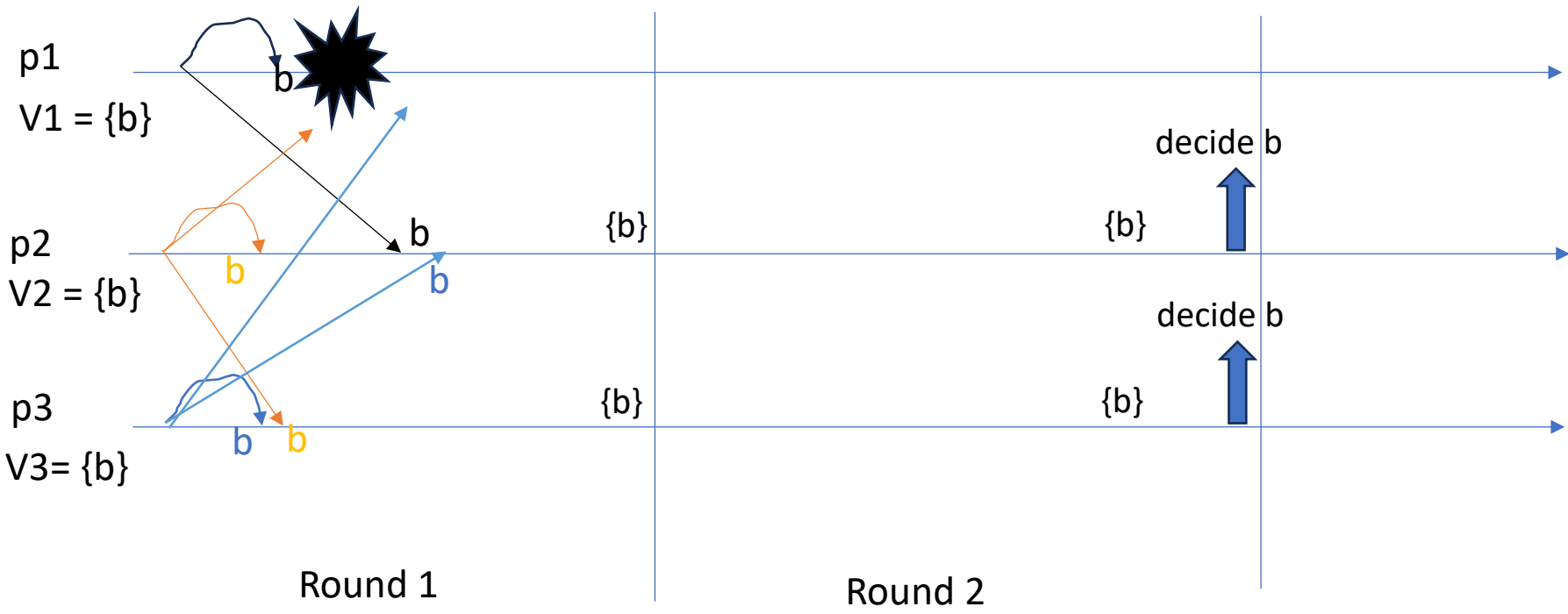
Correctness proof of Algorithm 1 (3)

```
propose(v_i) {// algorithm run by process p_i
Initially V_i = {v_i}

   1:round k, 1 ≤ k ≤ f + 1
   2:   send S_i = {v ∈ V_i: p_i has not already sent v} to all
        processes
   3:   receive S_j from p_j
   4:   V_i := V_i ∪ ⋃_{j=1}^{n} S_j
   5:y= min(V_i)
   6:return decide(y)
}
```

Algorithm 1

- Proof of Validity
  - Processes do not send fictitious values in this failure model
  - For all i, j, if the initial value v_i is identical to v_j then the only value that can be decided is $v_i = v_j$.



A failure-prone scenario of Algorithm 1 with n=3, f = 1

Correctness proof of Algorithm 1 (4)

```
propose(v_i) {// algorithm run by process p_i
Initially V_i = {v_i}

   1:round k, 1 ≤ k ≤ f + 1
   2:   send S_i = {v ∈ V_i: p_i has not already sent v} to all
        processes
   3:   receive S_j from p_j
   4:   V_i := V_i ∪ ⋃_{j=1}^{n} S_j
   5:y= min(V_i)
   6:return decide(y)
}
```

Algorithm 1

- Agreement :All non-faulty processes must agree on the same value

Correctness proof of Algorithm 1 (5)

Lemma (1)

In every execution, at the end of round $f + 1$, $V_i = V_j$ for every nonfaulty p_i and p_j

Correctness proof of Algorithm 1 (6)

Proof of Lemma (1) : It suffices to show that if $x \in V_i$ by the end of round $f+1$ then $x \in V_j$ by the end of round $f+1$, for all nonfaulty p_i and p_j. Let r be the first round at which $x \in V_i$ for some nonfaulty p_i (if x is already in V_i, $r = 0$)

case 1 : $r \leq f$

- At round $r+1 \leq f+1$, p_i broadcasts x to each process p_j (L2)
- p_j adds x to V_j (L4)
- Thus $x \in V_j$ by round $r+1 \leq f+1$

Correctness proof of Algorithm 1 (7)

case 2 : $r = f + 1$

- r is the first round at which $x \in V_i$ for some non-faulty p_i
- We must have a chain of faulty processes that transfered x to p_i during the previous rounds such that
  - $p_{i_{f+1}}$ sends x to p_i in round $f + 1$ (assumption of the case)
  - p_{i_f} sends x to $p_{i_{f+1}}$ in round f
  - ...
  - p_{i_2} sends x to p_{i_3} in round 2 and
  - p_{i_1} sends x to p_{i_2} in round 1 (x is the initial value of p_{i_1})

Correctness proof of Algorithm 1 (8)

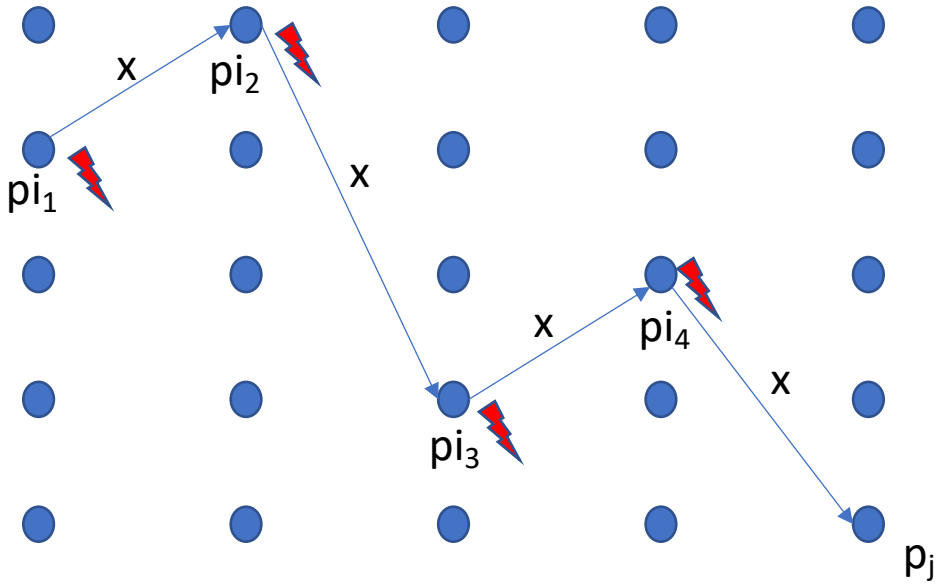


Illustration of the case r = f+1 with f=3

Correctness proof of Algorithm 1 (9)

- By construction each process sends x at most once (being faulty or not)
- Thus $p_{i_1}, p_{i_2}, \ldots, p_{i_{f+1}}$ are $f+1$ distinct processes
- Thus there must be at least one correct process among them
- Thus this process must have stored x in a round r' with $r' \leq f < r$
- Which contradicts the assumption that r is minimal

As a consequence, nonfaulty processes have the same set V by the end of round $f+1$ and thus decide on the same value

Performance of Algorithm 1

- Number of processes $n > f$
- $f + 1$ rounds
- $n^2 \mid V \mid$ messages, each of size $\lceil \log(V) \rceil$ bits, where V is the input set

Lower bound on rounds

- We present a lower bound of $f + 1$ on the number of rounds required in a synchronous system for reaching consensus in the presence of f crash failures
  - This means that there is no algorithm that always solves consensus in at most f rounds

Lower bound on rounds

Assumptions

- We focus on binary consensus
  - In a binary consensus, processes can only propose 0 or 1
- The proof assumes that every correct process is supposed to send a message to every other process in every round
- The proof assumes that there is one crash per round
- $f < n - 1$

State of a process

- Each p_i has a one-bit input register x_{p_i}, and output register y_{p_i} with values in $\{0, 1, \perp\}$

Definition (State of a process)

The state of p_i comprises the value of x_{p_i}, the value of y_{p_i} and all the messages it has received in the previous rounds

- Initial state of p_i : $x_{p_i} = 0$ or $x_{p_i} = 1$ and $y_{p_i} = \perp$
- Decision states : $y_{p_i} = 0$ or $y_{p_i} = 1$ (y_{p_i} is writable only once)

Configuration

Definition (Configuration)

The configuration at the end of round r is made up of the state of each process at the end of round r

- Configuration at the end of a round is the same at the one at the beginning of the next round
- Given an initial configuration and a pattern of failures, the execution of the algorithm gives rise to a sequence of configurations

Valence of a configuration

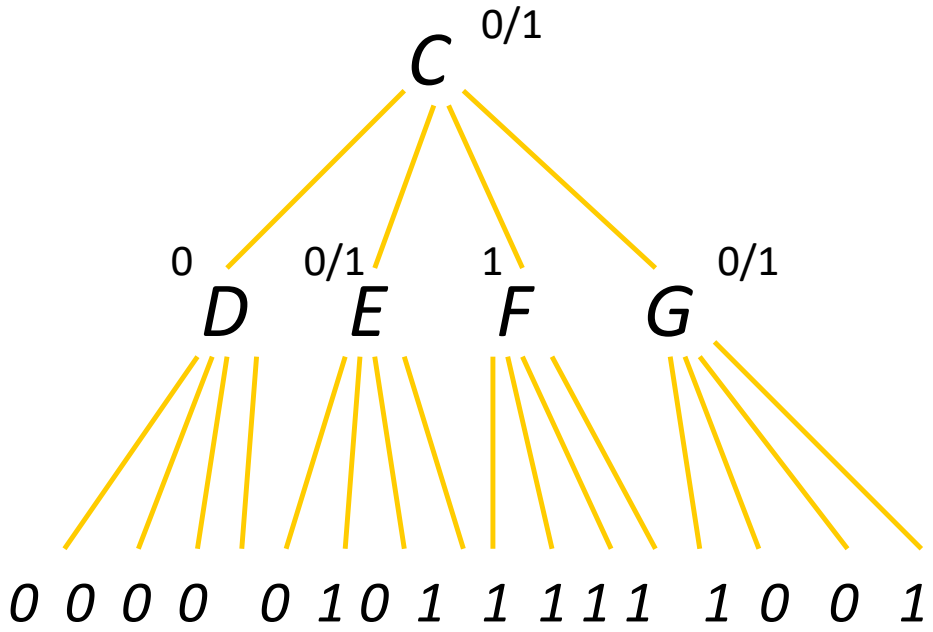
A key notion in this proof is the set of decisions that can be reached from a particular configuration

Definition (Valence of a configuration)

The valence of a configuration C during an execution is the set of values that are decided by a correct process in some configuration that is reachable from C

- By the termination property of Consensus, the set of decided values cannot be empty
- C is univalent if the set contains one value
  - C is 0-valent if the value is 0
  - C is 1-valent if the value is 1
- C is bivalent if the set contains two values

Valence of a configuration



<= decisions

0/1 : bivalent
1 : 1-valent
0 : 0-valent

Valence of a configuration

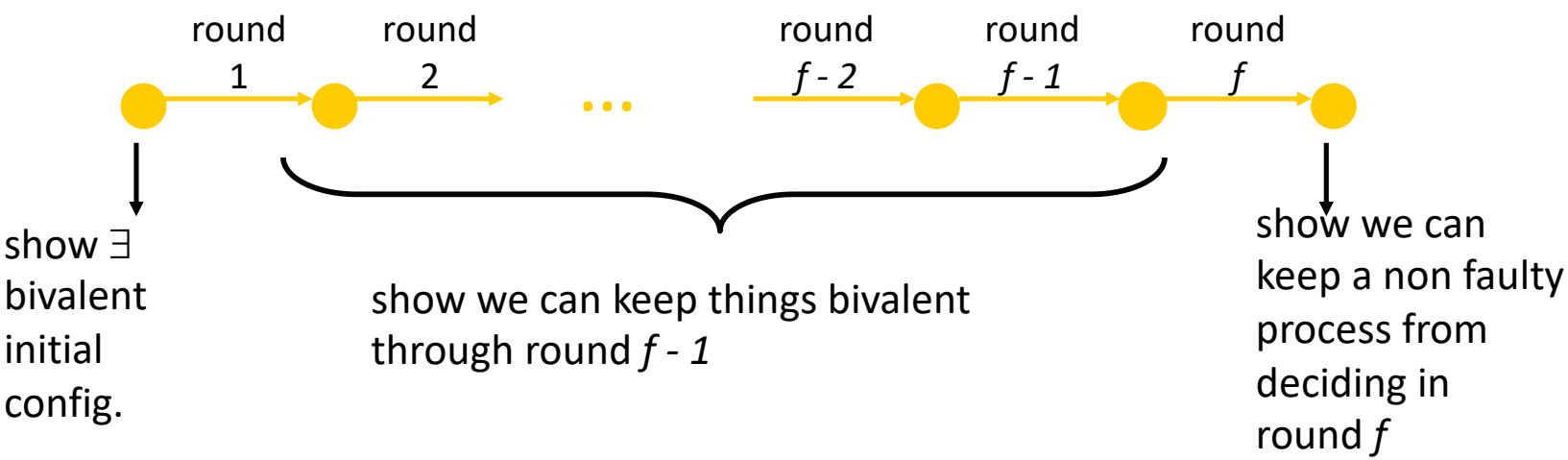
- By the agreement property, if some process has decided in a configuration, then the configuration is univalent
- In a bivalent configuration, which value is going to be decided depends on future events

Proof of the lower bound

Theorem

Any consensus algorithm $\mathcal{A}$ for n processes, resilient to f crash failures with $n > f + 1$, requires at least $f + 1$ rounds in some execution

Proof strategy



Existence of bivalent initial configuration

Lemma (1)

Algorithm $\mathcal{A}$ has a bivalent initial configuration.

Existence of bivalent initial configuration

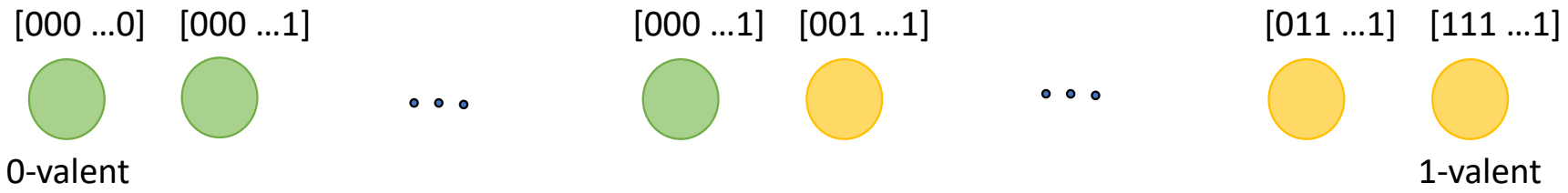
Proof : By contradiction. Suppose that all the initial configurations are univalent. By the validity property,

- initial configuration in which all inputs are 0 is 0-valent
- initial configuration in which all inputs are 1 is 1-valent

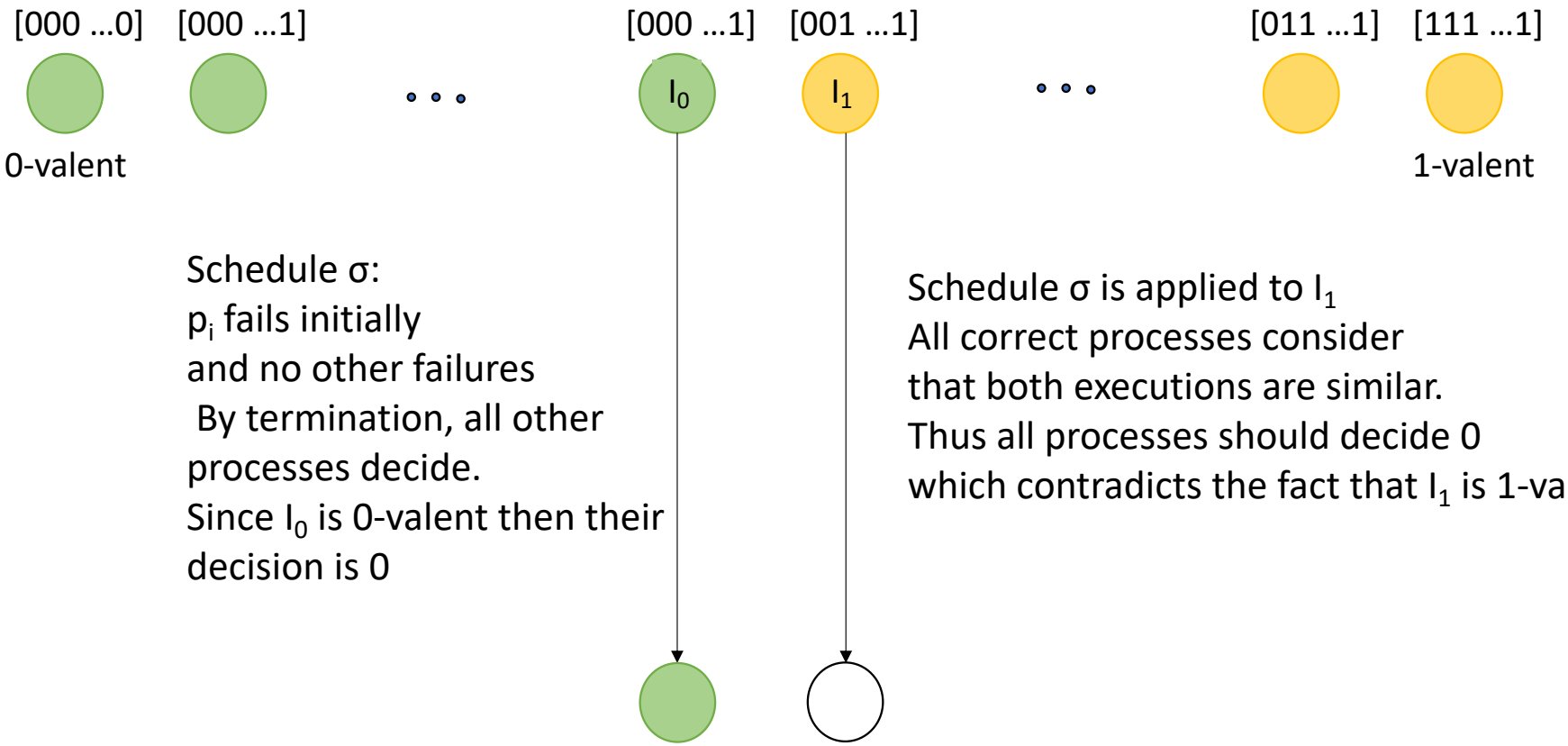
We can order the initial configurations in a chain of configurations, where two configurations are next to each other if they differ by only one value

$\rightarrow$ the difference between two adjacent configurations is the starting value of one process

Existence of bivalent initial configuration



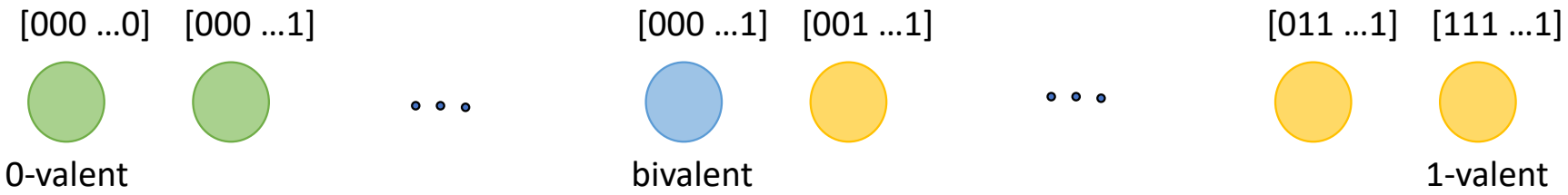
Existence of bivalent initial configuration



Schedule σ:
p_i fails initially
and no other failures
By termination, all other processes decide.
Since I_0 is 0-valent then their decision is 0

Schedule σ is applied to I_1
All correct processes consider that both executions are similar.
Thus all processes should decide 0
which contradicts the fact that I_1 is 1-valen

Existence of bivalent initial configuration



Existence of bivalent initial configuration

- So this result contradicts the fact that the outcome of the consensus algorithm is uniquely predetermined by the initial configurations

Keeping things bivalent

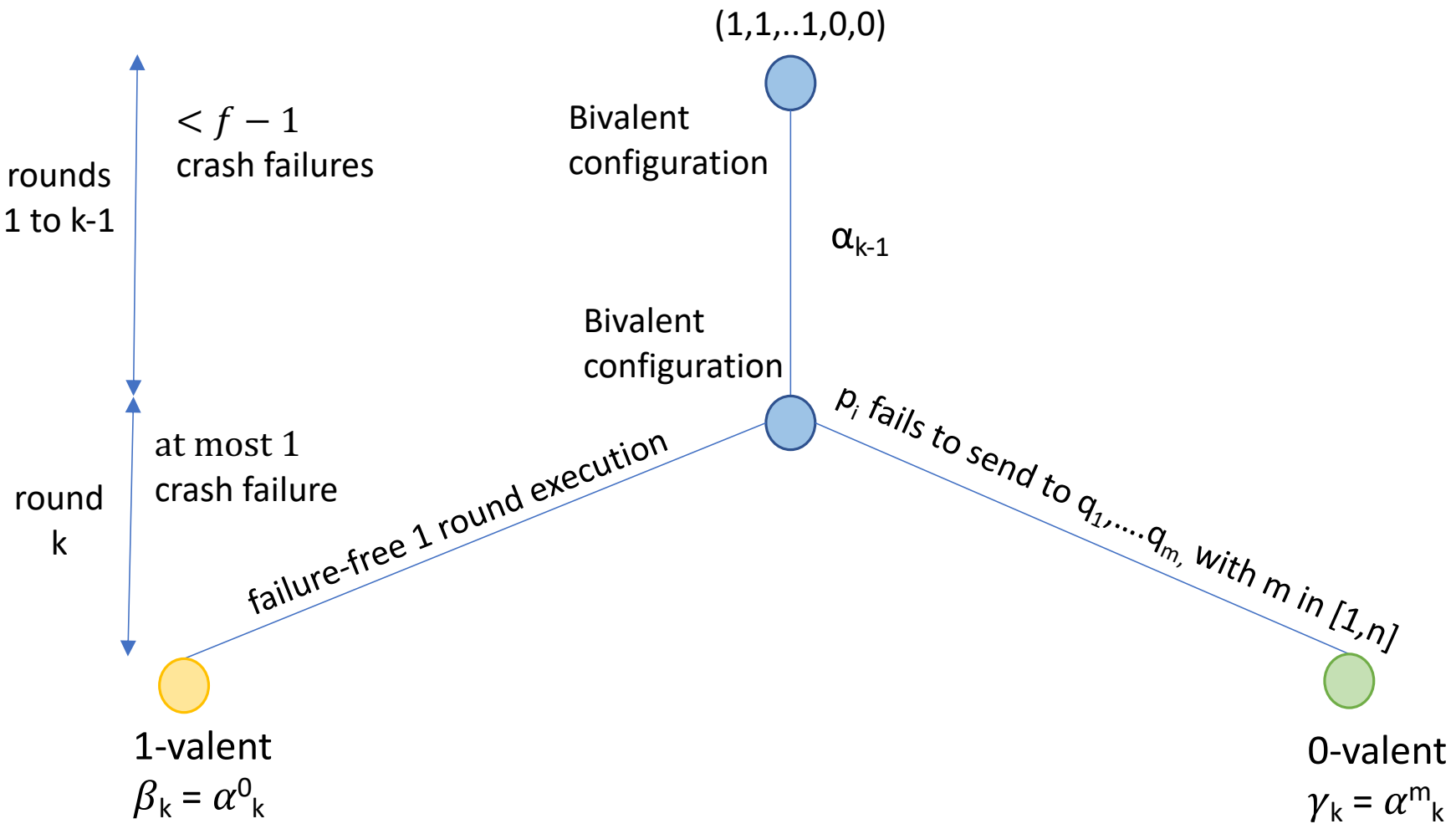
Lemma (2)

For each k, $0 \leq k \leq f-1$, there is a k-round execution of $\mathcal{A}$ that ends in a bivalent configuration

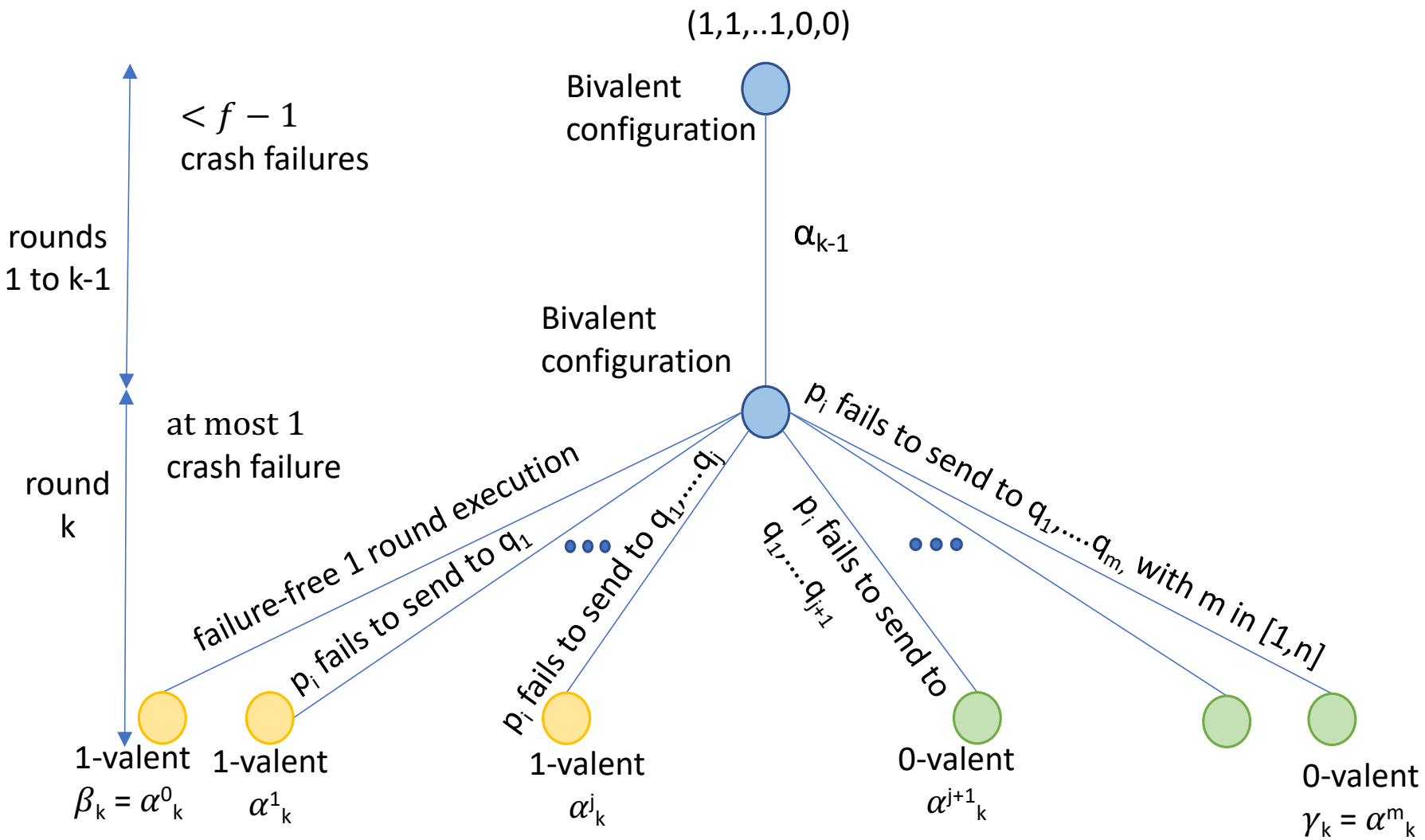
Proof : by induction on the round number

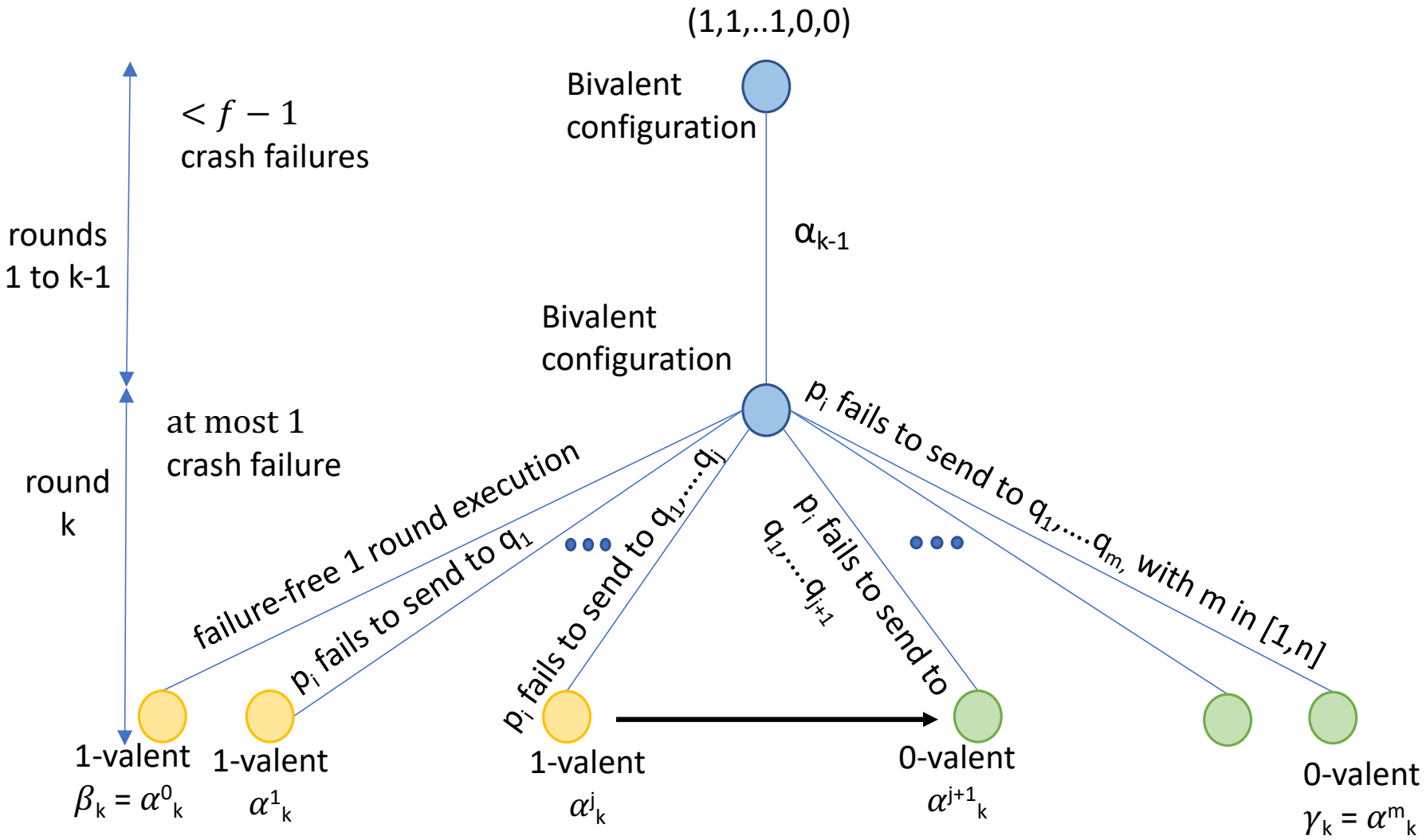
- The base case, $k = 0$, follows from Lemma (1)
- Assume that the lemma is true for rounds $0 \leq k \leq f-2$, and show that it is true for $0 \leq k \leq f-1$
- Let α_{k-1} be the $(k-1)$-round execution ending in a bivalent configuration (exists by the inductive assumption)
- Show that there is a one-round extension of α_{k-1} ending in a bivalent configuration
- Assume by contradiction that every one-round extensions of α_{k-1} with at most one crash failure ends in a one valent configuration

Keeping things bivalent



Keeping things bivalent





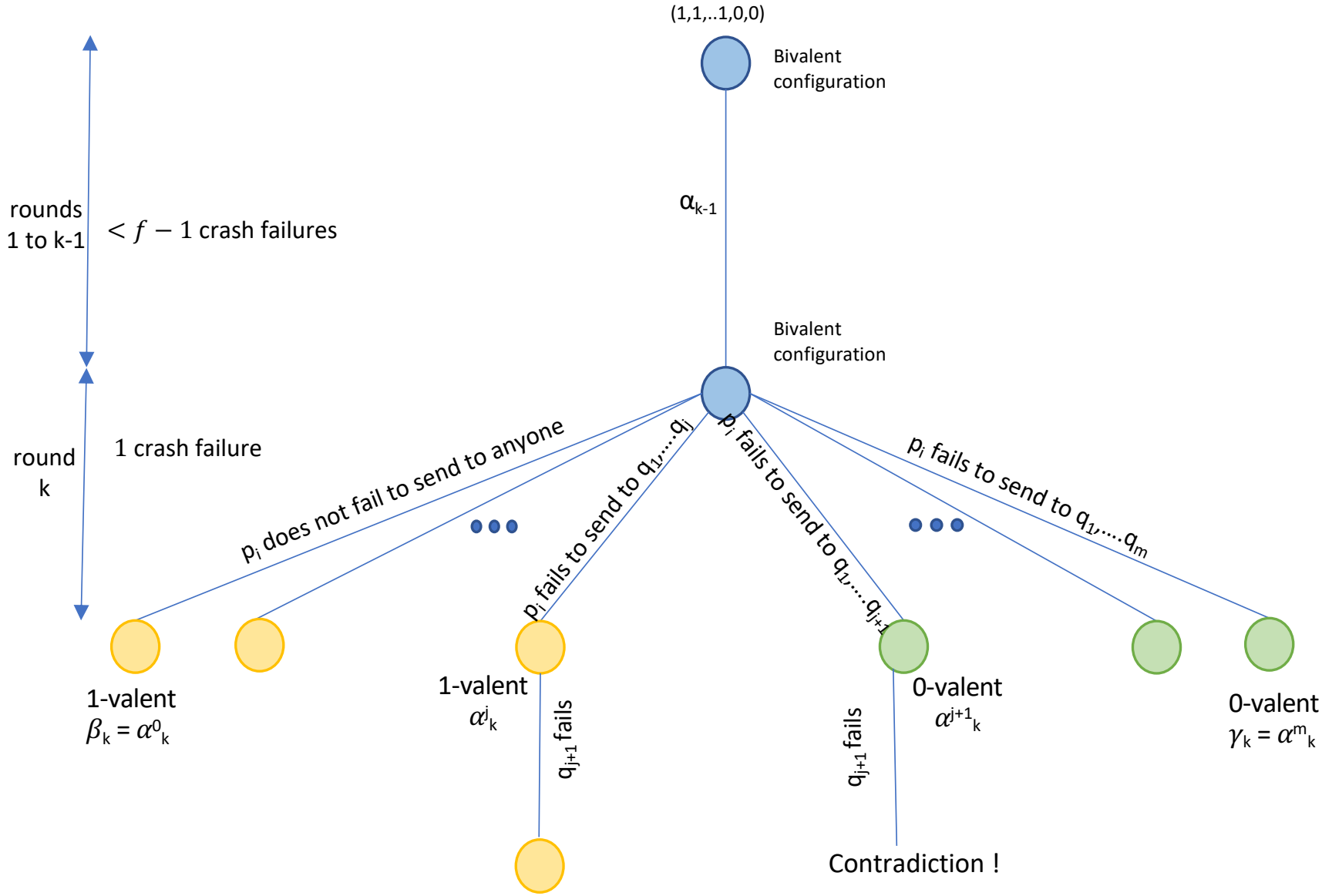
Switch from a 1-valent to a 0-valent conf.
in α^j_k : p_i sends a value to q_{j+1}
in α^{j+1}_k : p_i does not send a value to q_{j+1}

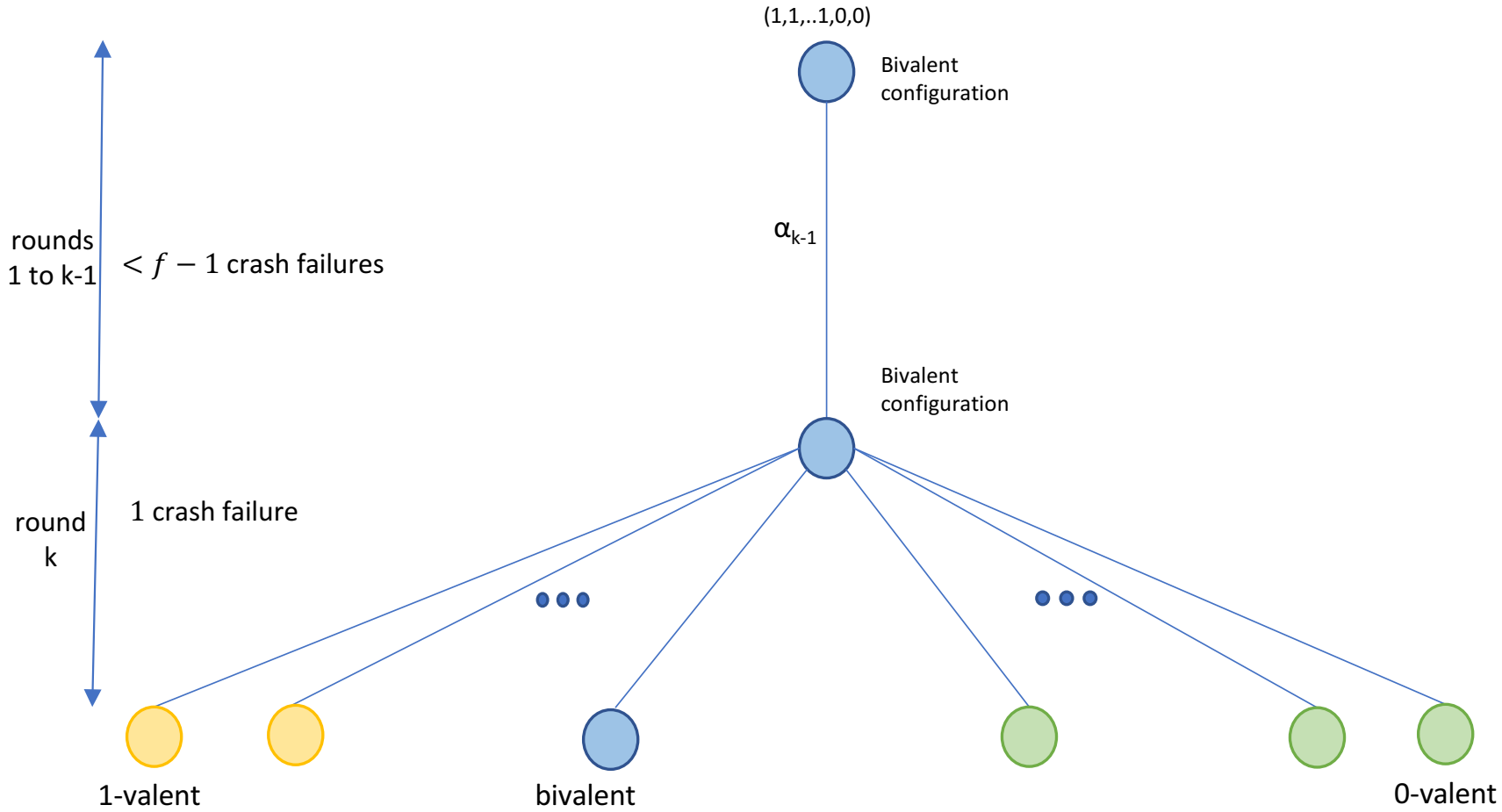
Keeping things bivalent

- The number of crashes in α_k^j and in α_k^{j+1} is $\leq f-1$ (since at most $k-1 \leq f-2$ processes crash in α_{k-1} and p_i crashes in round k)
- Thus there is still one more process that can crash without violating the bound f

Keeping things bivalent

- Consider both extensions
  - δ_{k+1}^{j} of α_k^j
  - δ_{k+1}^{j+1} of α_k^{j+1}
- In both extensions, q_{j+1} fails before sending any message in round $k+1$
- Thus q_{j+1} did not get the opportunity of revealing whether or not it received a message from p_i
- Thus both extensions are similar with respect to every nonfaulty process (since the only difference between them is that p_i sends its information to q_{j+1} in δ_{k+1}^{j} but not in δ_{k+1}^{j+1}, but q_{j+1} failed at the beginning of round $k+1$)


(1,1,..1,0,0)
Bivalent configuration
rounds 1 to k-1
$< f - 1$ crash failures
α_{k-1}
Bivalent configuration
round k
1 crash failure
p_i does not fail to send to anyone
p_i fails to send to $q_1,\ldots q_j$
p_i fails to send to $q_1,\ldots q_{j+1}$
p_i fails to send to $q_1,\ldots q_m$
1-valent
$\beta_k = \alpha^0_k$
1-valent
α^j_k
q_{j+1} fails
q_{j+1} fails
0-valent
α^{j+1}_k
0-valent
$\gamma_k = \alpha^m_k$
Contradiction !



Thus there must exist a one-round extension of α_{k-1} ending in a bivalent configuration

Keeping things bivalent

- We have shown that we necessarily have a $(f-1)$ round execution ending in a bivalent configuration
- We now show that the f-th round execution does not necessarily preserve bivalence, but nonfaulty processes cannot determine yet what decision to make, and thus an additional round is necessary

Cannot decide in round f

Lemma

Let α_{f-1} be an $(f-1)$-round execution of $\mathcal{A}$ that ends in a bivalent configuration, then there exists a one round extension of α_{f-1} in which some non-faulty process cannot decide

Cannot decide in round f

Proof

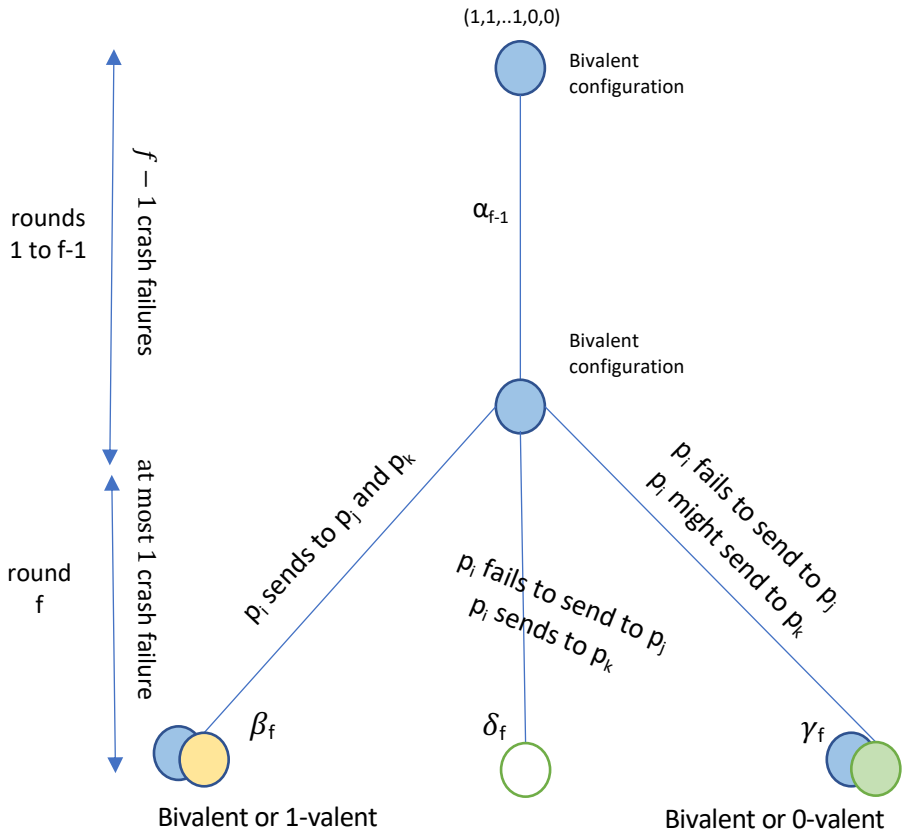
- Let β_f be the one-round univalent extension of α_{f-1} in which no failure occurs
- If β_f ends in a bivalent configuration we are done
- Suppose that β_f ends in an univalent configuration (say 1-valent)
- There must exist another 1-round execution γ_f of α_{f-1} which ends either in a bivalent configuration (in which case we are done) or in a 0-valent configuration
- Remember that one more process p_i that can crash, and by assumption at least 2 processes never crash (i.e., $f < n - 1$)

Lower bound on the number of rounds

β_f and δ_f are similar with respect to p_k. Thus p_k is either undecided or decide 1 at the end of round f.

γ_f and δ_f are similar with respect to p_j. Thus p_j is either undecided or decide 0 at the end of round f.

Since algorithm A satisfies the agreement Property, it cannot be the case that in δ_f both p_k and p_j have decided (p_j would have decided 0 and p_k would have decided 1)



Lower bounds on the number of rounds

Thus $f + 1$ rounds are necessary for all correct processes to decide

Theorem

Any consensus algorithm $\mathcal{A}$ for n processes, resilient to f crash failures with $n \geq f + 2$, requires at least $f + 1$ rounds in some execution

Consensus problem with Byzantine failures in a synchronous system

Definition (Synchronous Byzantine consensus)

- Agreement : All non-faulty processes must agree on the same single value
- Validity : If all the non-faulty processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value
- Termination : Each non-faulty process must decide on a value

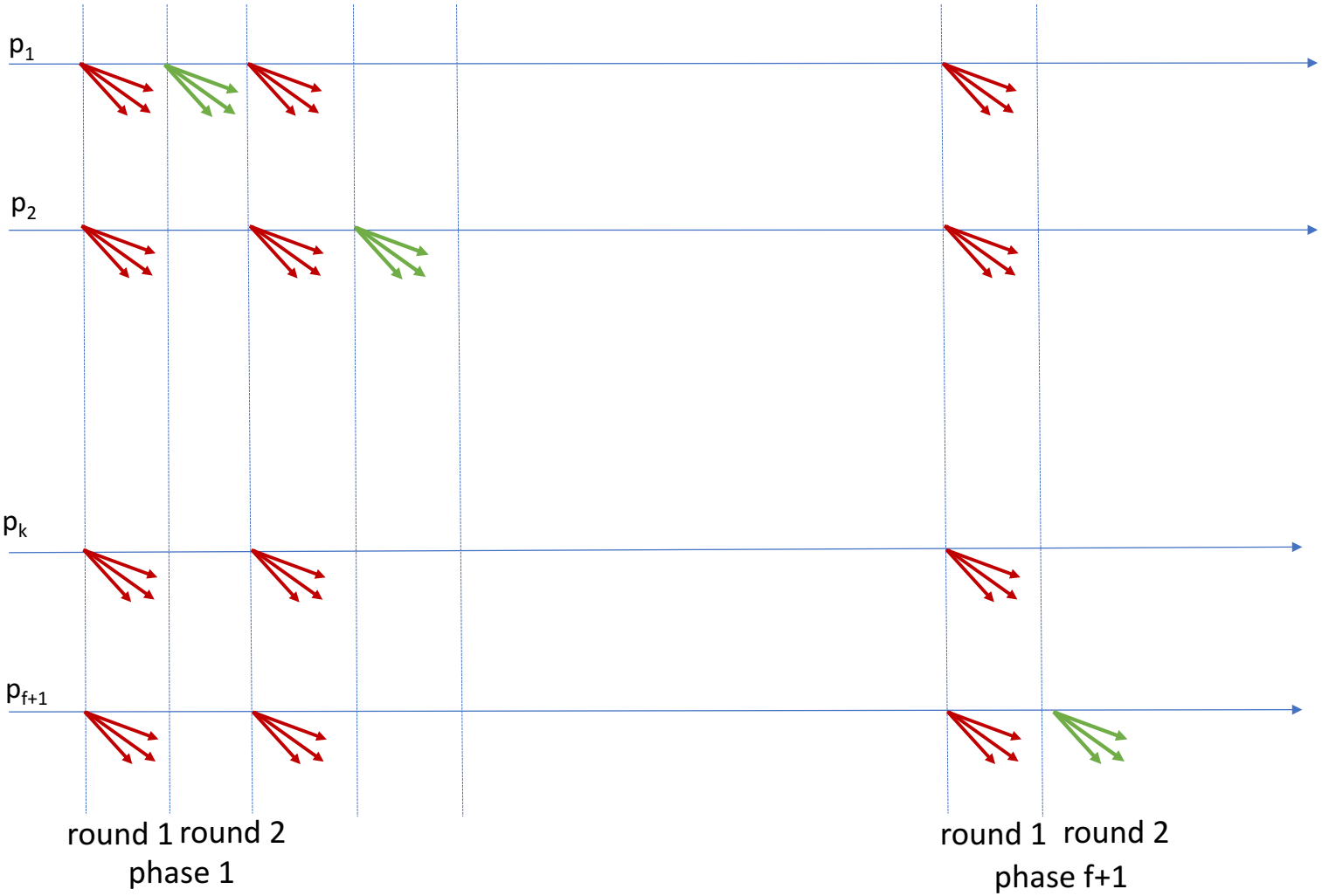
An algorithm to solve consensus with Byzantine processes (Berman and Garay, 1993)

The phase-king algorithm solves the consensus problem in a synchronous model

- $n > 4f$ processes, f Byzantine processes
- The algorithm requires $f + 1$ phases, each phase made of 2 rounds
- At each phase, a unique process plays the leader role

The phase-king algorithm : a rotating coordinator algorithm

- Rotating coordinator : allows us to break the symmetry among processes
- Each phase is partially under the control of a coordinating process
- The identity of the coordinator is given by the round number (thus each process knows who is the current coordinator)
- Each process p_i maintains an estimate v_i of the decision value


p_1
p_2
p_k
p_{f+1}
round 1
round 2
phase 1
round 1
round 2
phase f+1

The phase-king algorithm : a rotating coordinator algorithm

- Two principles :

case 1 If the occurrence number of the most frequent estimate value passes some majority threshold, this value will be the decided value

case 2 Otherwise, the current coordinator will force an estimate value to be adopted by enough processes, so that case 1 applies

```
// algorithm run by process p_i
Initially est_i ← v_i , ∀j ≠ i  est_j ← v_⊥


1: Round 2k − 1, 1 ≤ k ≤ f + 1
2:     send est_i to all
3:     receive  v_j from p_j ; est_j ← v_j for all responses
4:     maj_i ← the majority value of est_1,...,est_n (v_⊥ if none)
5:     mult_i ← number of times maj occurs

6: Round 2k, 1 ≤ k ≤ f + 1
7:     if i = k then send maj_k to all
8:     receive king_maj from p_k (v_⊥ if none)
9:     if mult_i > ⌊n/2⌋ + f then
10:        est_i ← maj_i
11:    else
           est_i ← king_maj
12:    if k = f + 1 then y_i = est_i
}
```

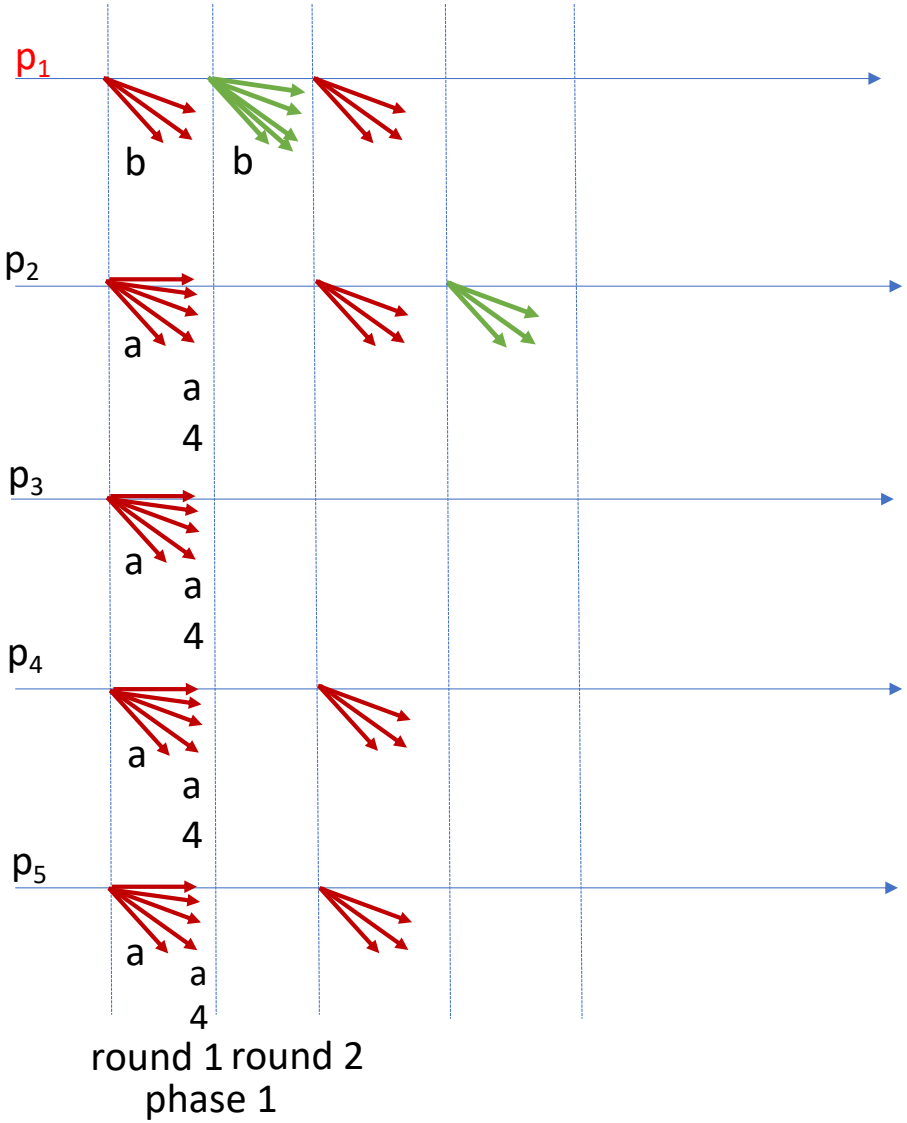
Proof of correctness of the phase-king consensus algorithm

Lemma (1)

If all non faulty processors have v as estimate at the beginning of phase k, then they keep v at the end of phase k, $1 \leq k \leq f+1$

Proof :

- Suppose all correct processes have v as estimate at the beginning of phase k
- Each process receives at least $n-f$ copies of v (including its own value) at the end of the first round of phase k
- By assumption, $n \geq 4f+1$, thus $n-f \geq 3f+1$
- We have $\lfloor n/2 \rfloor + f \leq \lfloor (4f+1)/2 \rfloor + f = 3f$
- Thus these $n-f$ messages exceed the $\lfloor n/2 \rfloor + f$ threshold to accept v
- All correct proceses will all set est_i to v by the end of round 2 of phase k



For all correct p_i, maj_i = a

and $mult_i$ = 4

By definition, n-f = 4

We have floor(n/2) = 2

So $mult_i$ > floor(n/2) + 1 (Line 9)

So Line 10 applies

Proof of correctness of the phase-king consensus algorithm

Lemma(1) immediately implies Validity

Validity property

- If all non faulty processes start with the same value v, they continue to prefer v throughout the phases since the value at the beginning of the next phase is equal to the value at the end of the current phase
- Thus at phase $f + 1$ they decide v

Termination property

- At the end of phase $f + 1$ all correct processes decide

Proof of correctness of the phase-king consensus algorithm

Agreement

- Because there are at most f failures, at least one phase will be coordinated by a non faulty coordinator
- We ignore all phases up to the first phase with a non faulty coordinator
- Let g be such a phase
- We assume that all the estimates are set arbitrarily at the start of this phase
- We will argue that at the end of the phase, all non-faulty processes will have the same estimate

Proof of correctness of the phase-king consensus algorithm

Lemma (2)

Let g be a phase whose coordinator p_g is nonfaulty. Then all nonfaulty processes finish phase g with the same estimate

Proof :

- case 1. Suppose that all processes use the coordinator value (Line 11)

  - By assumption, p_g is nonfaulty, thus it sends the same value to all processes
  - By Lemma (1) they will keep the same value

Proof of correctness of the phase-king consensus algorithm

Proof (cont'd) :

Case 2. Suppose that some correct process p_i uses its own majority value v (Line 10)

- Thus p_i must have received more than $\lfloor n/2 \rfloor + f$ messages for v in round 1 of phase g
- Of these values, more than $\lfloor n/2 \rfloor$ were sent by non-faulty processes
- Thus every process (including the coordinator p_g) received more than $\lfloor n/2 \rfloor$ messages for value v during round 1 of phase g
- Thus all of them set their majority value equal to v
- If any other process ℓ observes a majority of $\lfloor n/2 \rfloor$ messages for value v', the two super majority overlap and thus $v = v'$

Thus whether Line 10 or Line 11 are executed, each non faulty process sets its estimate to v

Proof of correctness of the phase-king consensus algorithm

Theorem (Agreement)

All nonfaulty processes agree on the same value

Proof :

- By Lemma (2), if coordinator p_g of phase $g \leq f + 1$ is nonfaulty then all nonfaulty processes finish phase g with the same estimate.
- By Lemma (1) all non-faulty processes keep the same value until the end of phase $f + 1$. Thus they all decide the same value at the end of phase $f + 1$

Lower bound on the ratio of Byzantine processes

Lower bound on the ratio of Byzantine processes

Theorem (Lower bound on the ratio of Byzantine processes)

If a third or more of the processes can be Byzantine, then consensus cannot be reached

Informally this lower bound captures the following scenario : if there are only three parties a, b, c and parties b and c accuse each other for lying and provide no proof-of-malice to party a, then a has no way to decide between b and c. Party a has no way to know who to trust and agree with.

Some preliminary definitions

Definition (View of a process)

Let α be an execution and let p_i be a process.
The view of p_i, denoted by $\alpha|p_i$, is the subsequence of computation and message delivery events that occur in α at p_i together with p_i's initial value.

Definition (Similar executions)

Let α_1 and α_2 be two executions and let p_i be a process that is correct in both α_1 and α_2.
α_1 is similar to α_2 with respect to p_i, denoted by $\alpha_1 \overset{p_i}{\sim} \alpha_2$, if $\alpha_1|p_i = \alpha_2|p_i$

Idea of the lower bound proof

- We consider executions that start from different carefully chosen initial configurations
- An adversary chooses these executions so that each process finds certain pairs of executions indistinguishable

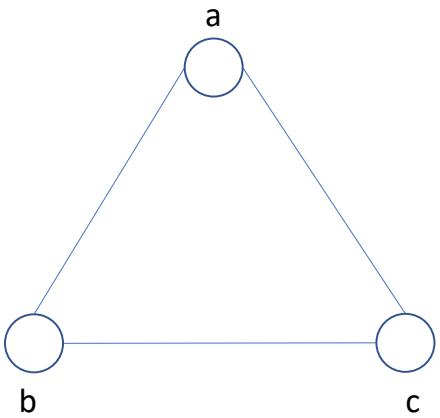
Lower bound on the ratio of Byzantine processes

Proof.

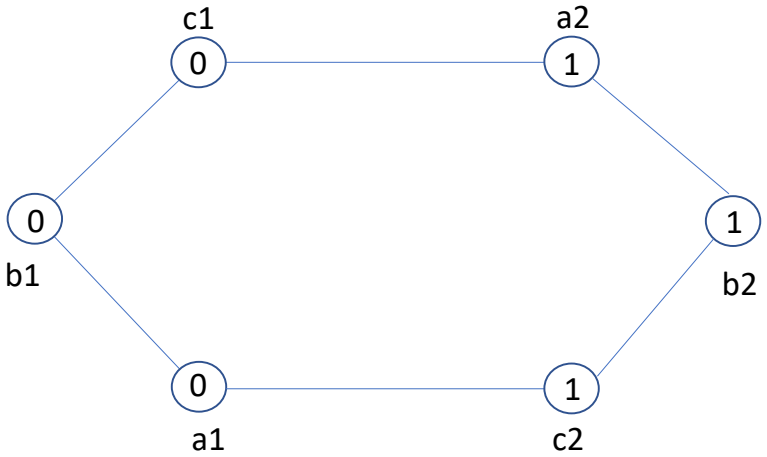
- We suppose by contradiction that there exists a protocol P solving consensus in a system R
- R consists of 3 processes a, b and c, and one of them is Byzantine
- We construct a new system $\overline{R}$
  - $\overline{R}$ is a synchronous ring composed of 6 entities $(a_1, b_1, c_1, a_2, b_2, c_2)$
  - $(a_1, b_1, c_1, a_2, b_2, c_2)$ are well behaved
  - We do not assume that $(a_1, b_1, c_1, a_2, b_2, c_2)$ solve consensus
  - We just claim that processes in the triangle solve it
- We will use $\overline{R}$ to get the contradiction

Lower bound on the ratio of Byzantine processes

- inputs :
  - $Input(a_1) = Input(b_1) = Input(c_1) = 0$
  - $Input(a_2) = Input(b_2) = Input(c_2) = 1$



R



$\bar{R}$

Lower bound on the ratio of Byzantine processes

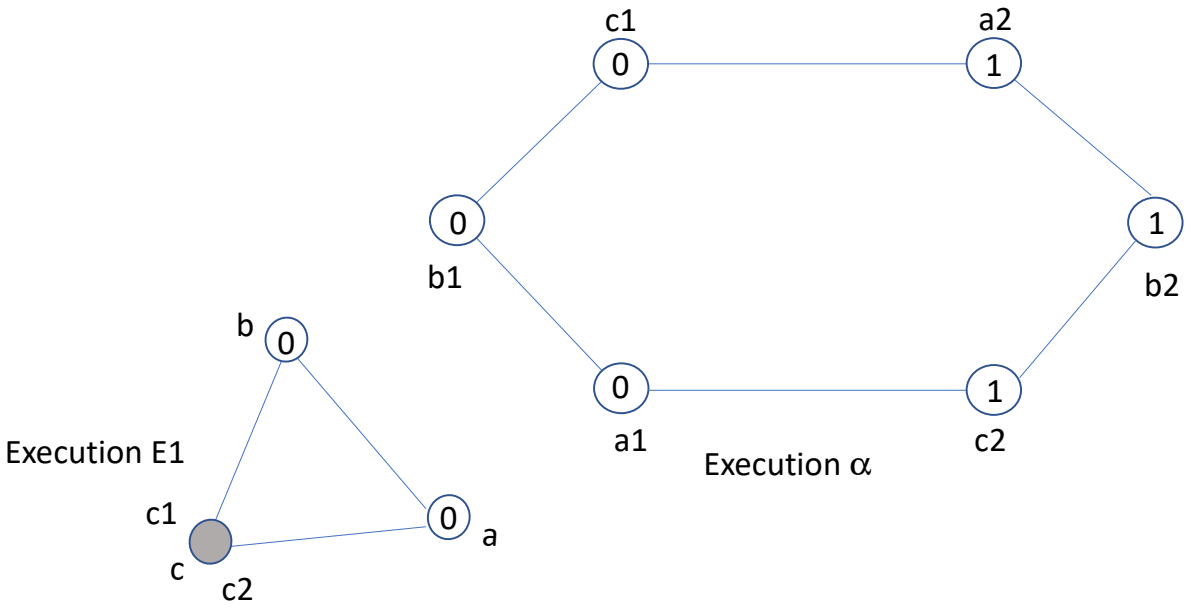
- Both a_1 and a_2 run the local protocol run by a
- Both b_1 and b_2 run the local protocol run by b
- Both c_1 and c_2 run the local protocol run by c ;

- All these entities execute without any faults
- Their execution is called α
- Execution α is used to specify the behavior of faulty processes in some triangles

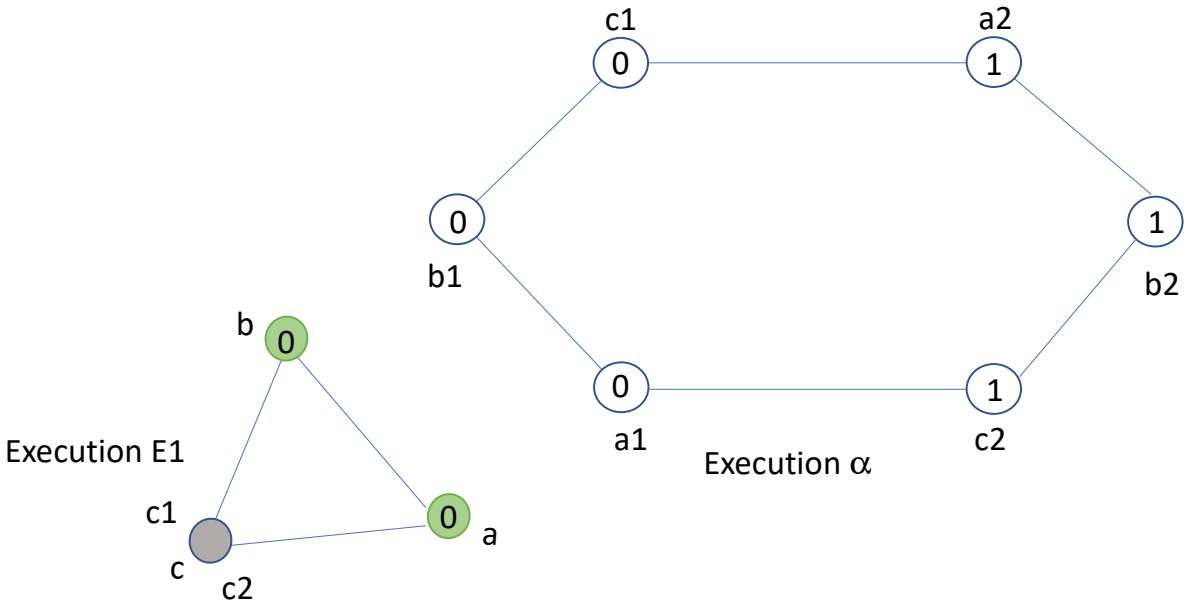
Lower bound on the ratio of Byzantine processes

- We now consider the original ring R and focus on three different executions of protocol P
- In each of these executions, 2 entities are nonfaulty and one is Byzantine
- The behavior of the nonfaulty entities is determined by P
- For the Byzantine entity we choose a behavior which is connected to ring $\overline{R}$

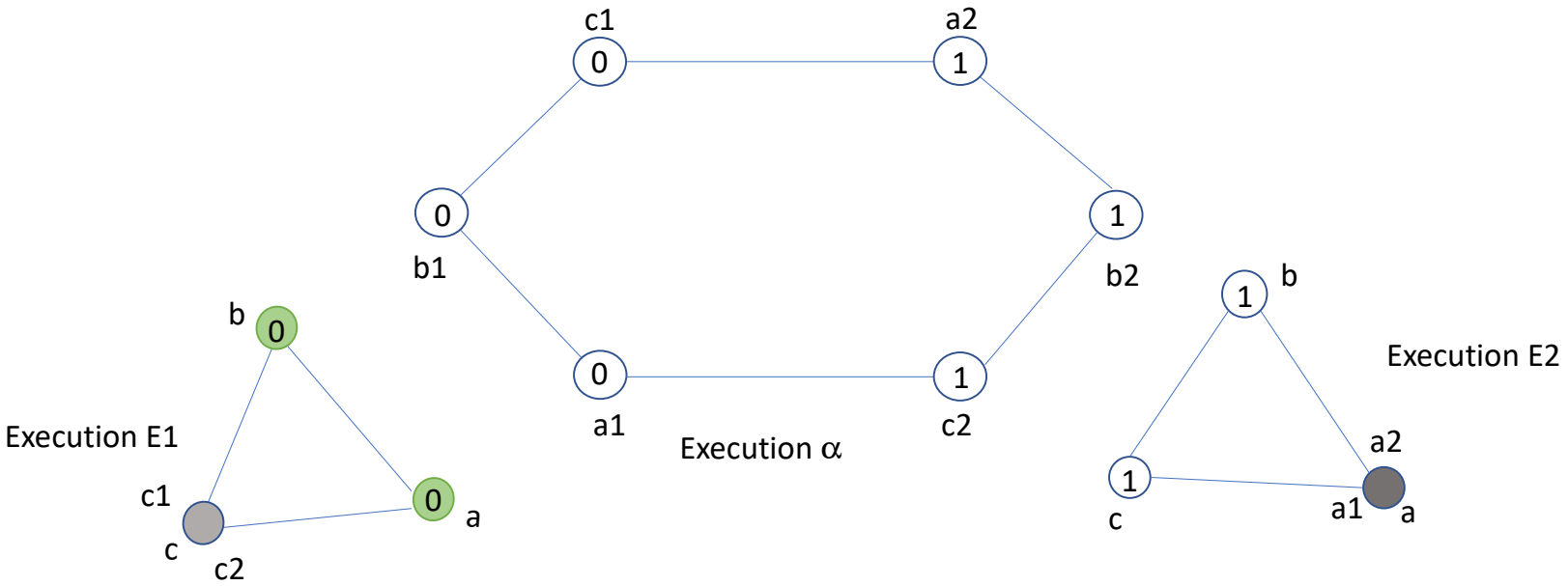
- Execution E_1 :
  - Entities a and b are nonfaulty and have initial value 0
  - Entity c is Byzantine and behaves toward a as c_2 toward a_1 and behaves toward b as c_1 toward b_1 in $\overline{R}$
  - $E_1|a = \alpha|a_1$ (i.e., a has the same view in E_1 as a_1 has in α)
  - $E_1|b = \alpha|b_1$ (i.e., b has the same view in E_1 as b_1 has in α)



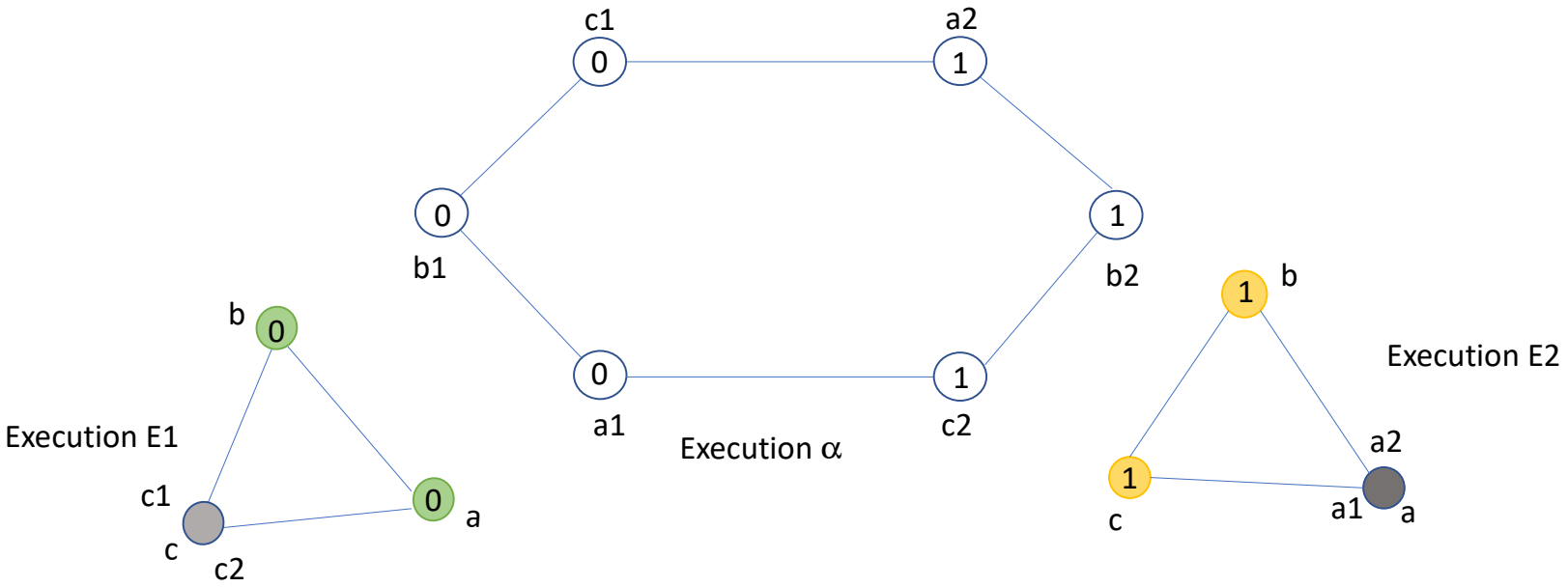
- Execution E_1 :
  - We have assumed that P is correct thus both entities a and b must at some point decide
  - By the validity property, if all nonfaulty have the same initial value, they must decide that value. Thus both a and b must decide 0.



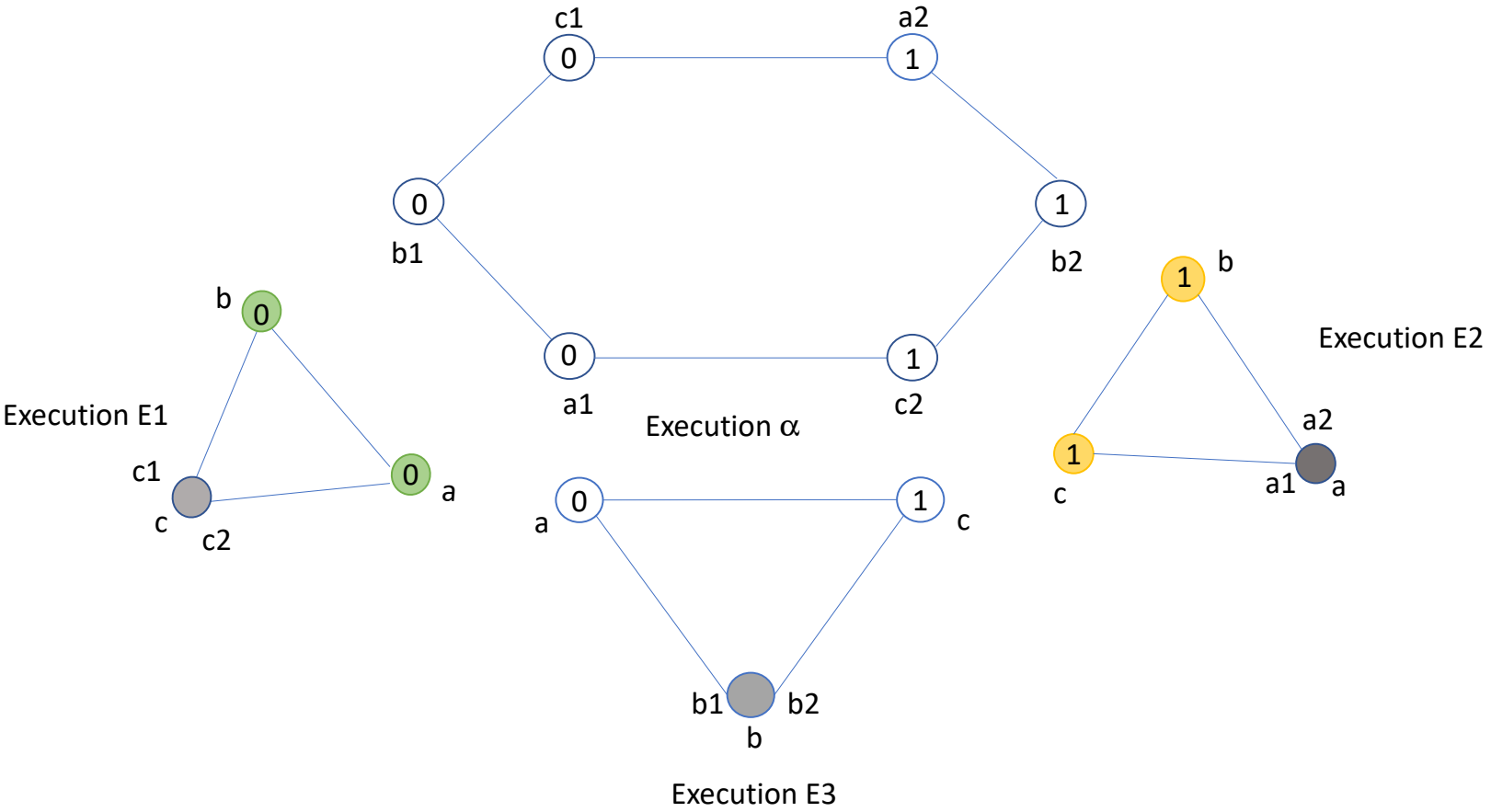
- Execution E_2 :
  - Entities b and c are nonfaulty and have initial value 1
  - Entity a is Byzantine and behaves toward b as a_2 toward b_2 and behaves toward c as a_1 toward c_2 in $\overline{R}$
  - $E_2|b = \alpha|b_2$ (i.e. b has the same view in E_2 as b_2 has in α)
  - $E_2|c = \alpha|c_2$ (i.e., c has the same view in E_2 as c_2 has in α)



- Execution E_2 :
  - We have assumed that P is correct thus both entities b and c must at some point decide
  - By the validity property, both b and c must decide 1.



- Execution E_3 :
  - Entities a and c are nonfaulty and have initial value 0 and 1
  - Entity b is Byzantine and behaves toward a as b_1 toward a_1 and behaves toward c as b_2 toward c_2 in $\overline{R}$
  - $E_3|a = \alpha|a_1$ and $E_3|c = \alpha|c_2$



Lower bound on the ratio of Byzantine processes

- We argue that $E_1 \overset{a}{\sim} E_3$

Recall

Definition (Similar executions)

Let α_1 and α_2 be two executions and let p_i be a process that is correct in both α_1 and α_2.
α_1 is similar to α_2 with respect to p_i, denoted by $\alpha_1 \overset{p_i}{\sim} \alpha_2$, if $\alpha_1|p_i = \alpha_2|p_i$

Lower bound on the ratio of Byzantine processes

- We have seen that $E_1|a = \alpha|a_1$ and $E_3|a = \alpha|a_1$
- Process a is non faulty in both E_1 and E_3
- Thus $E_1 \overset{a}{\sim} E_3$
- And so a decides 0 in E3

Lower bound on the ratio of Byzantine processes

- We argue that $E_2 \overset{c}{\sim} E_3$

Lower bound on the ratio of Byzantine processes

- We have seen that $E_2|c = \alpha|c_2$ and $E_3|c = \alpha|c_2$
- Process c is non faulty in both E_2 and E_3
- Thus $E_2 \overset{c}{\sim} E_3$
- And so c decides 1 in E3

- We have that a (which is correct) decides 0 in E3 and that c (which is also correct) decides 1 in E3
- This violates the agreement property
- A contradiction, which ends the proof of the Theorem

Lower bound on the ratio of Byzantine processes

Theorem

If a third or more of the processes are Byzantine, then consensus cannot be reached

Bibliography

- H. Attiya and J. Welch, Distributed Computing, Fundamentals, simulations, and advanced topics. Wiley Series.
- N. Lynch, Distributed Algorithms, Morgan Kaufmann Publishers