MC714 - Sistemas Distribuídos

Consenso

Islene Calciolari Garcia

Instituto de Computação - Unicamp

Primeiro Semestre de 2015

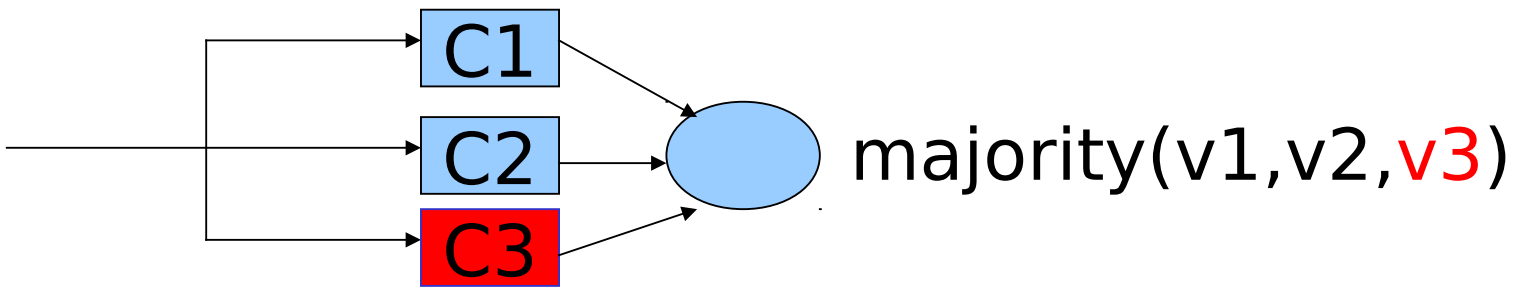
Sumário

Motivation

Build reliable systems in the presence of faulty components

Common approach:

- Have multiple (potentially faulty) components compute same function
- Perform majority vote on outputs to get “right” result



f faulty, f+1 good components ==> 2f+1 total

Fonte: Andrea C. Arpaci-Dusseau

Failures in Distributed Systems

Link failure: A link fails and remains inactive for some time; the network may get disconnected

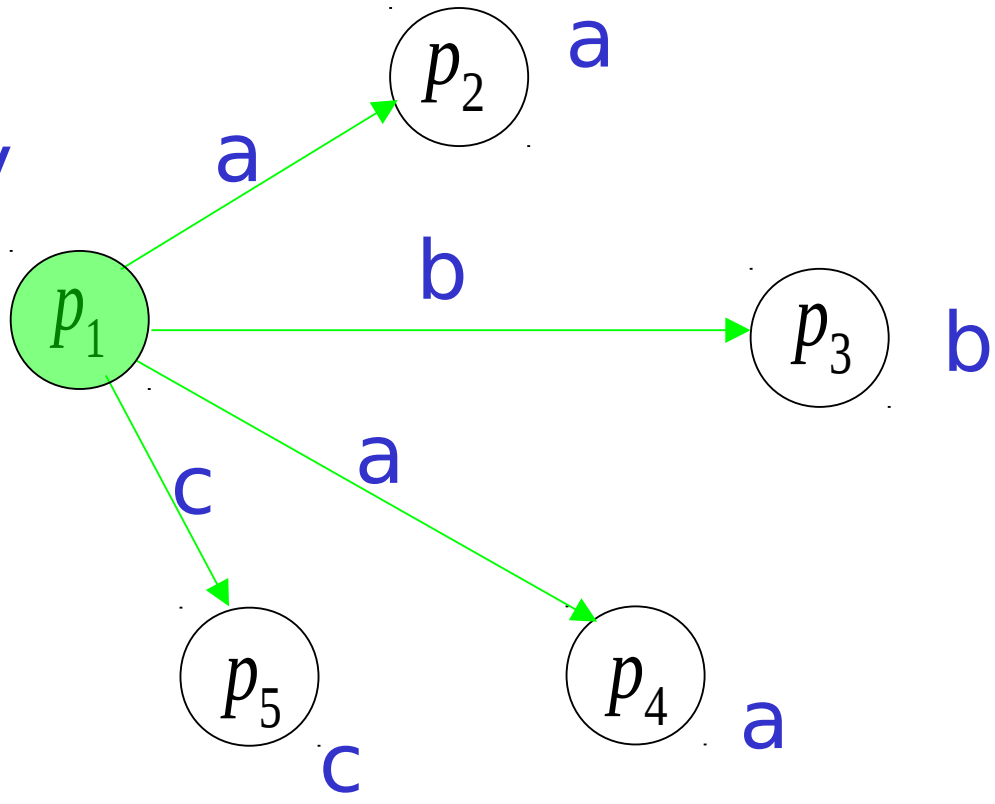
Processor crash failure: At some point, a processor stops forever taking steps; also in this case, the network may get disconnected

Processor Byzantine failure: during the execution, a processor changes state arbitrarily and sends messages with arbitrary content (name dates back to untrustable Byzantine Generals of Byzantine Empire, IV–XV century A.D.); also in this case, the network may get disconnected

Fonte: Prof. Guido Proietti

Normal operating

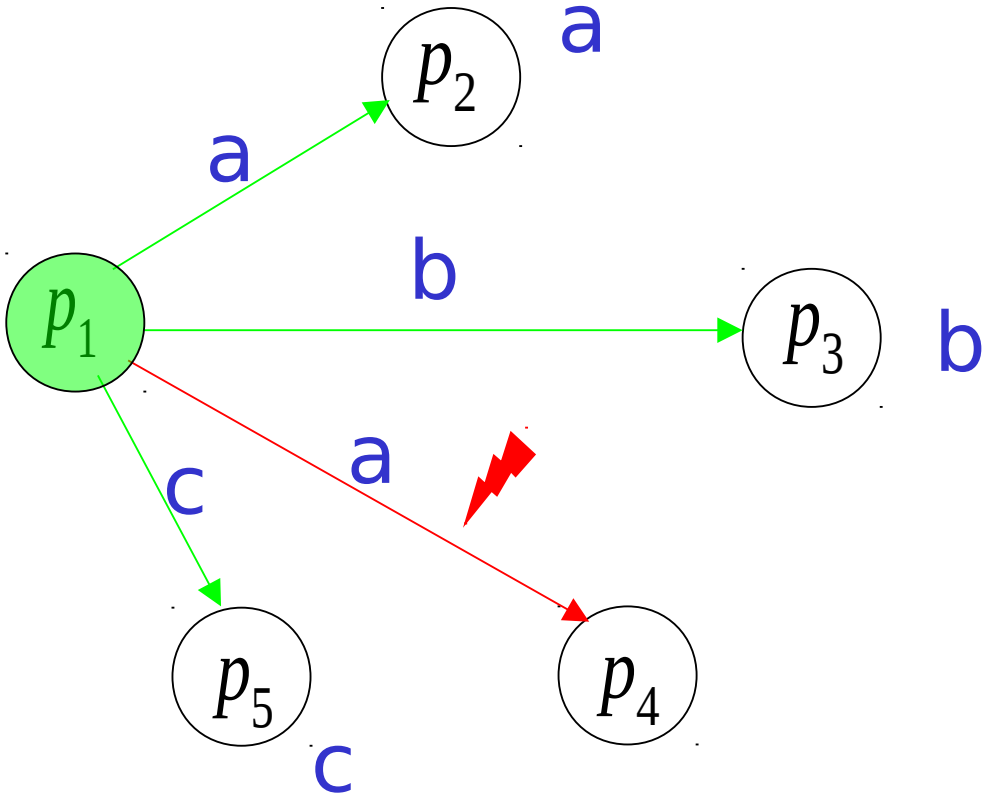
Non-faulty links and nodes



Fonte: Prof. Guido Proietti

Link Failures

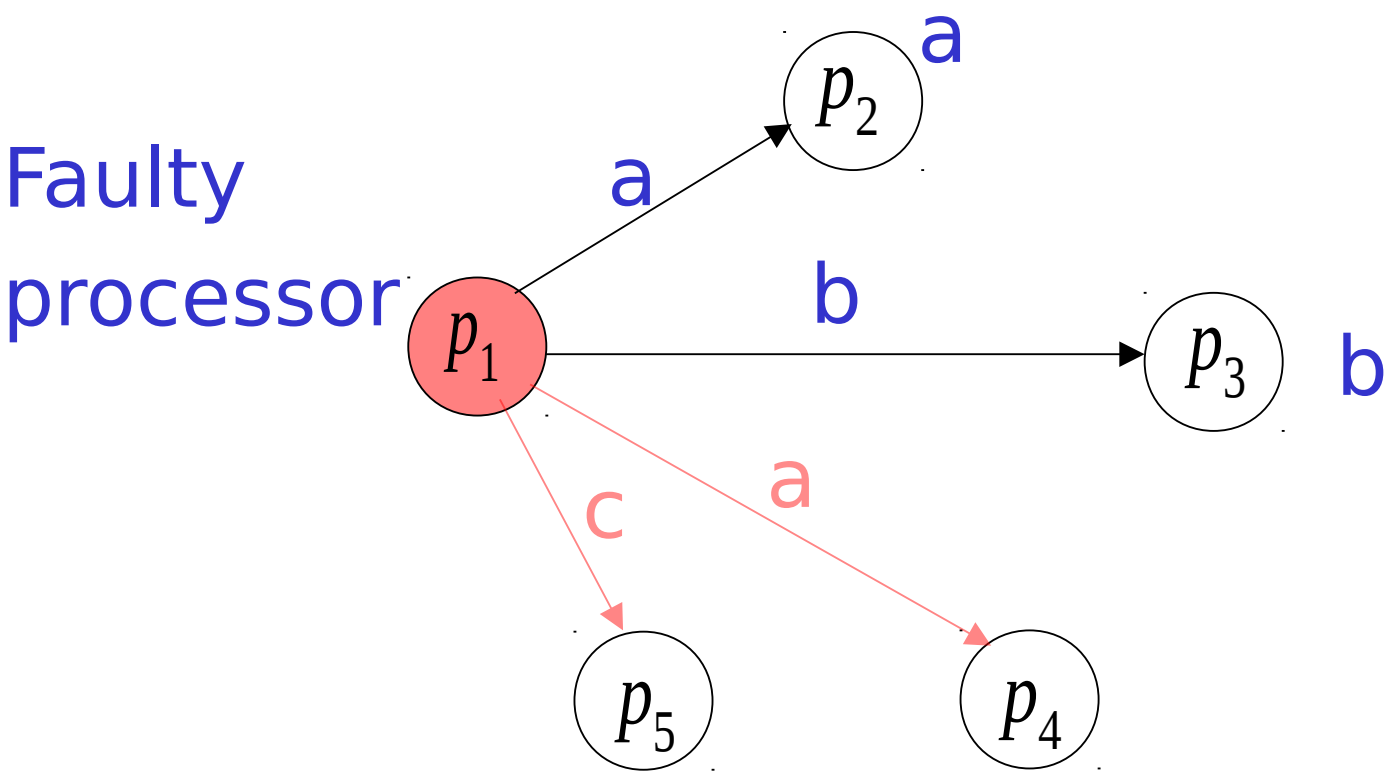
Faulty link



Messages sent on the failed link are not delivered

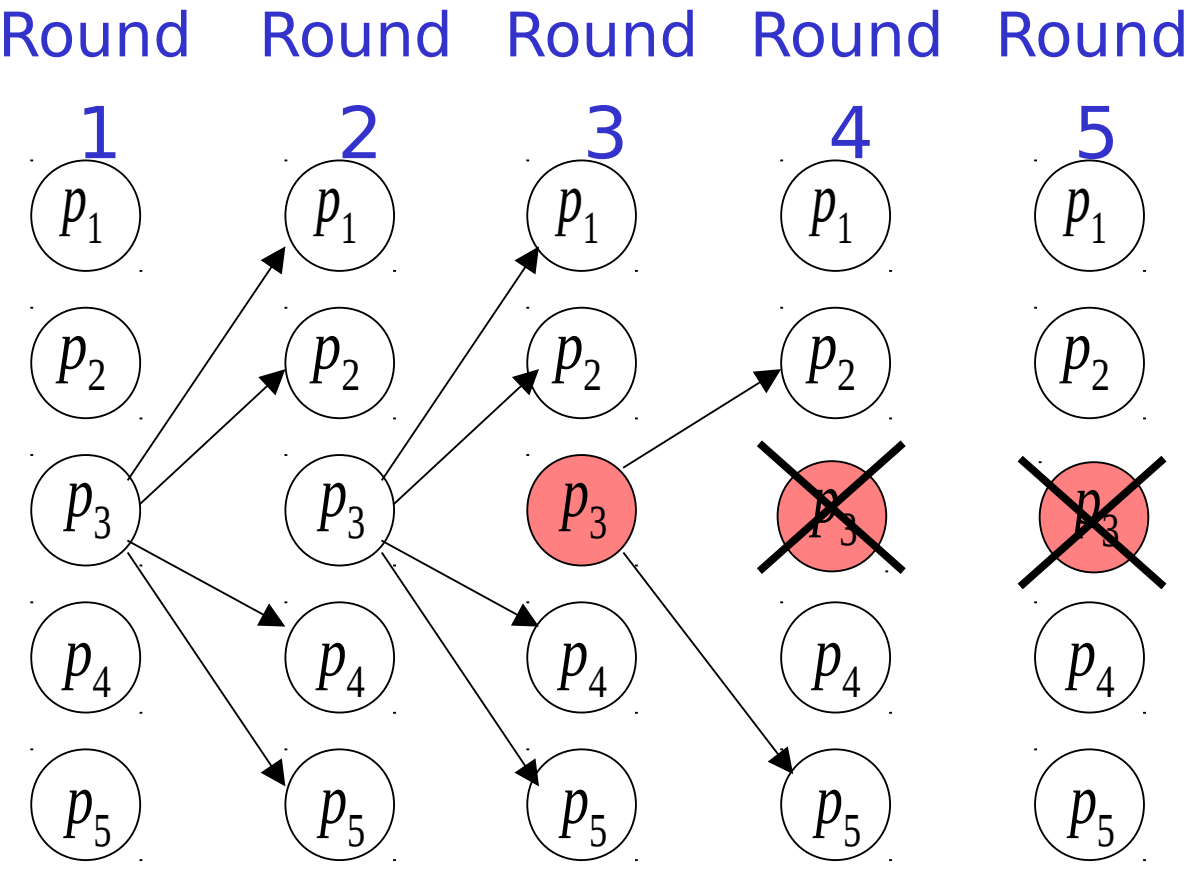
Fonte: Prof. Guido Proietti

Processor crash failure



Some of the messages are not sent

Fonte: Prof. Guido Proietti

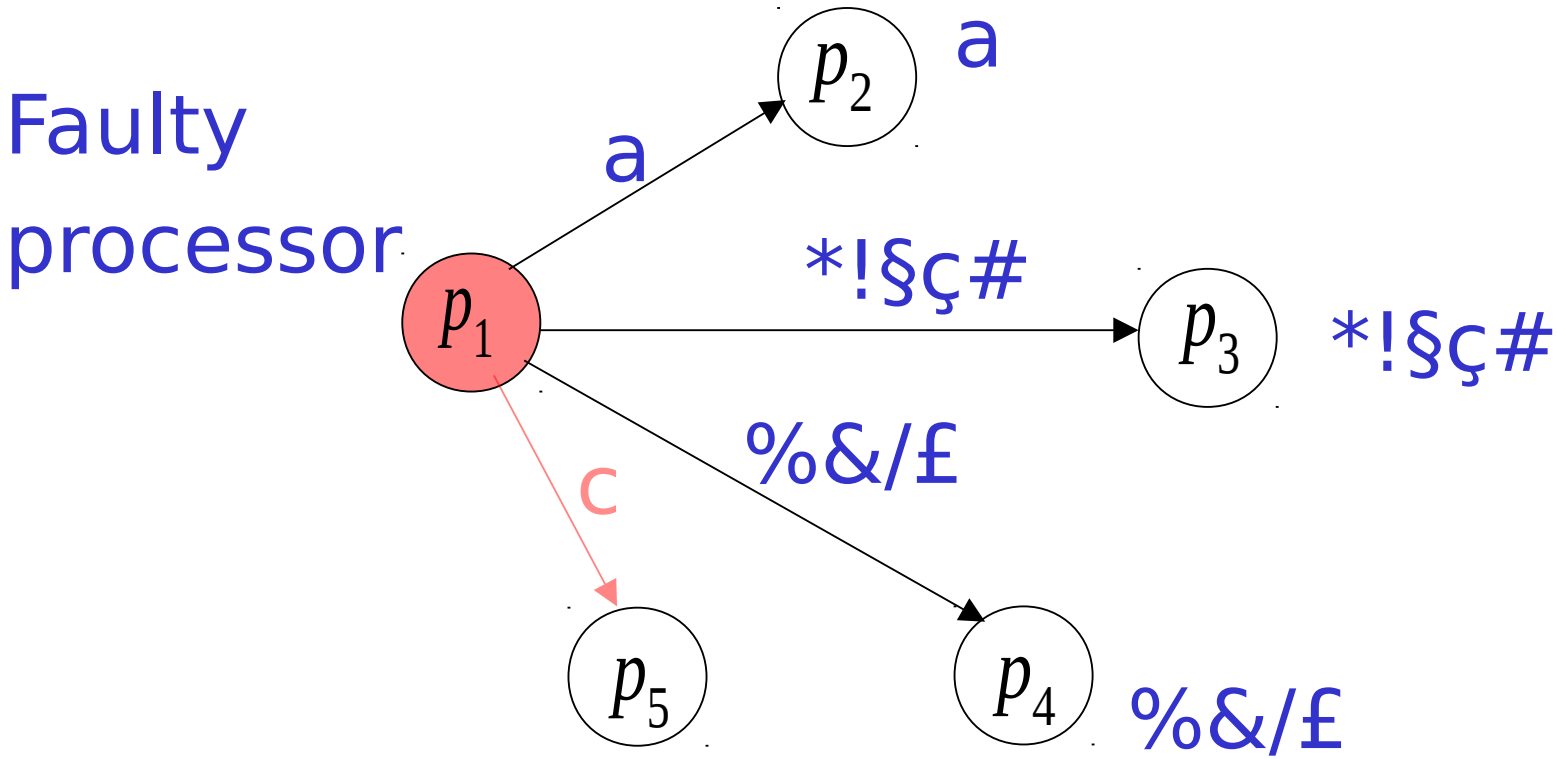


Failure in a synchronous MPS

After failure the processor disappears from the network

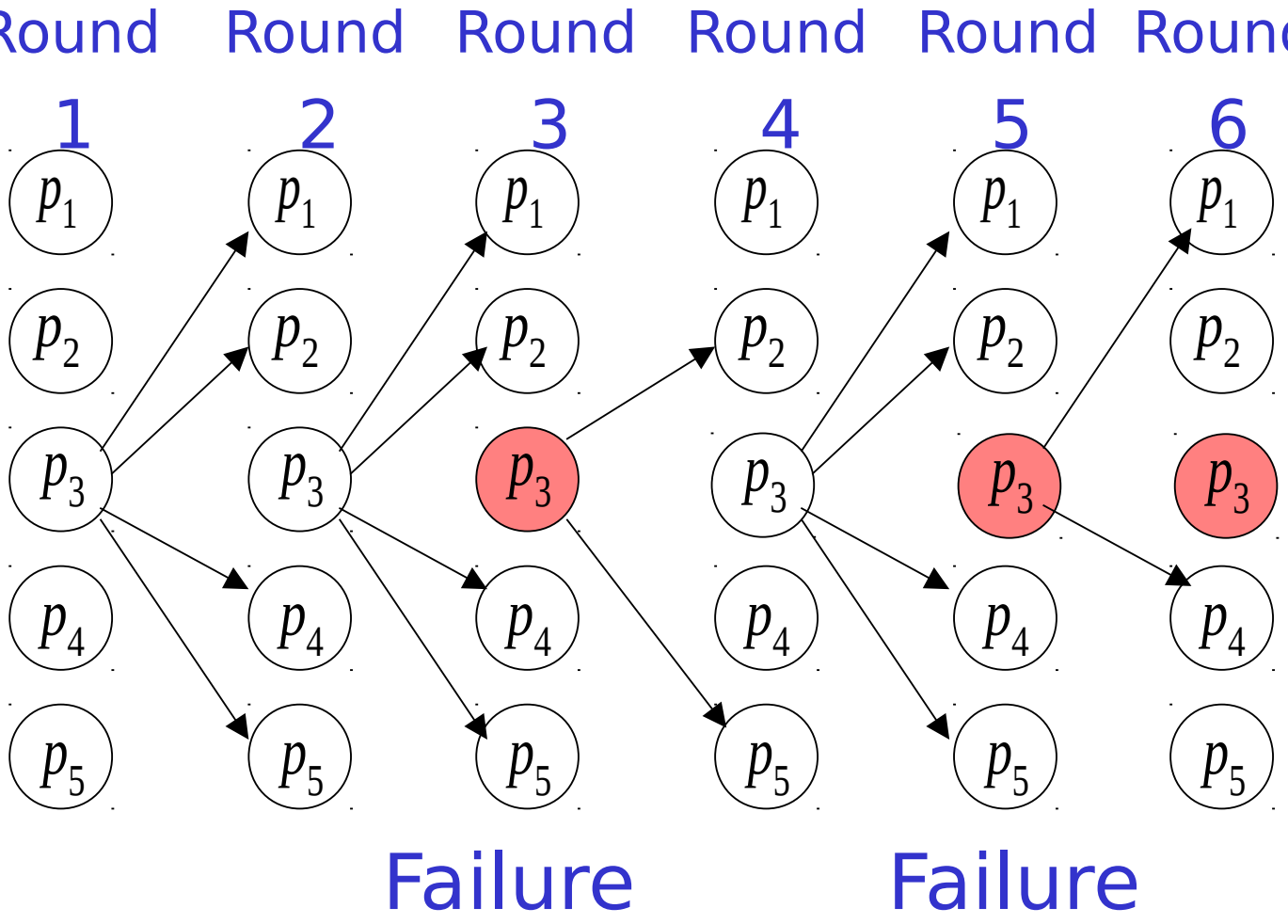
Fonte: Prof. Guido Proietti

Processor Byzantine failure



Processor sends arbitrary messages (i.e., they could be either correct or not), plus some messages may be not sent

Fonte: Prof. Guido Proietti



After failure the processor may continue functioning in the network

Fonte: Prof. Guido Proietti

Consensus Problem

Every processor has an input $x \epsilon X$ (this makes the system non-anonymous, in a sense), and must decide an output $y \epsilon Y$. Design an algorithm enjoying the following properties:

Termination: Eventually, every non-faulty processor decides on a value $y \epsilon Y$.

Agreement: All decisions by non-faulty processors must be the same.

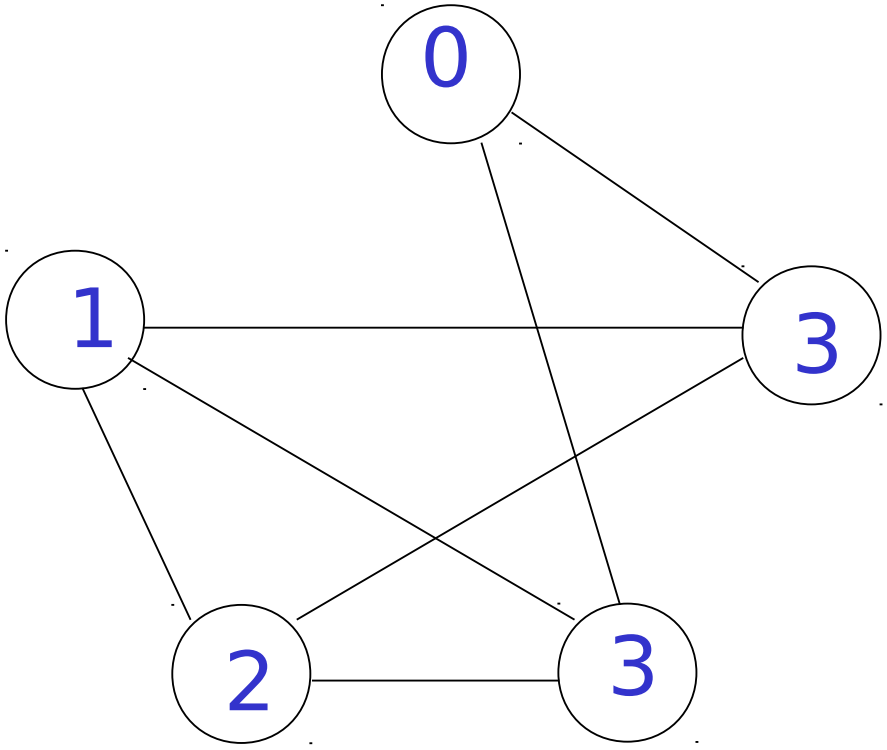
Validity: If all inputs are the same, then the decision of a non-faulty processor must equal the common input (this avoids trivial solutions).

In the following, we assume that $X=Y=\mathbf{N}$

Fonte: Prof. Guido Proietti

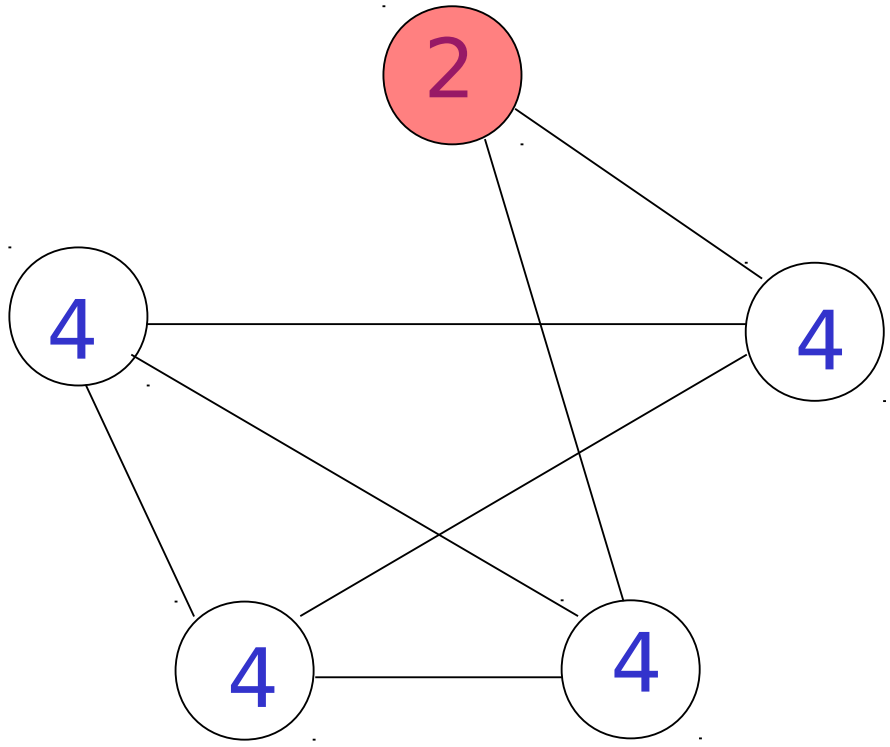
Agreement

Start



Everybody has an initial value

Finish



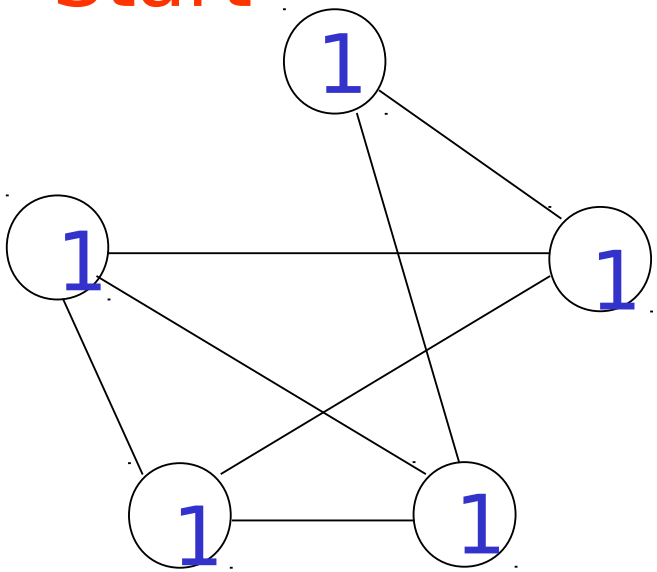
All non-faulty must decide the same value

Fonte: Prof. Guido Proietti

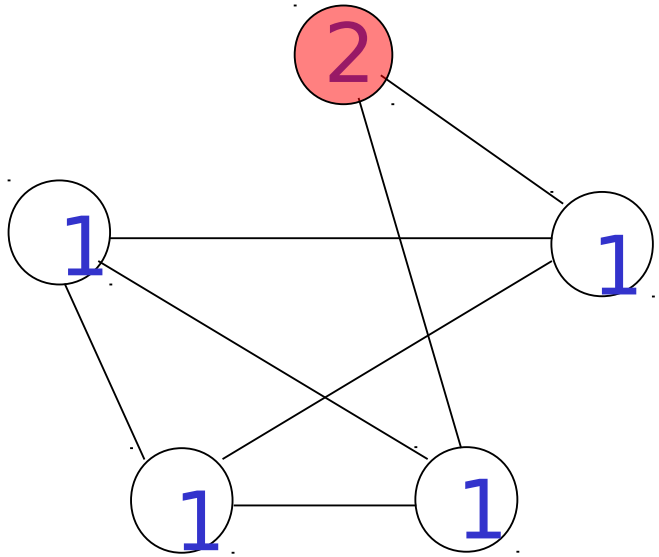
Validity

If everybody starts with the same value, then non-faulty must decide that value

Start



Finish



Fonte: Prof. Guido Proietti

Negative result for link failures

- Although this is the simplest fault a MPS may face, it is already enough to prevent consensus (just think to the fact that the link failure may be permanent and could disconnect the network!)

⇒ Thus, is general (i.e., there exist at least one instance of the problem such that) it is impossible to reach consensus in case of link failures, even in the synchronous case, and even if one only wants to tolerate a single link failure

- To illustrate this negative result, we present the very famous problem of the 2 generals

Fonte: Prof. Guido Proietti

Consensus under non-permanent link failures:the 2 generals problem

- There are two generals of the same army who have encamped a short distance apart.
- Their objective is to decide on whether to capture a hill, which is possible only if they both attack (i.e., if only one general attacks, he will be defeated, and so their common output should be either “not attack” or “attack”). However, they might have different opinion about what to do (i.e., their input).
- The two generals can only communicate (synchronously) by sending messengers, which could be captured, though.
- Is it possible for them to reach a common decision?

⇒ More formally, we are talking about consensus in the following MPS:

Fonte: Prof. Guido Proietti

Impossibility of consensus under link failures

- First of all, notice that it is needed to exchange messages to reach consensus (as we said, generals might have different opinions in mind!)
- Assume the problem can be solved, and let Π be the shortest protocol (i.e., with minimum number of messages) for a given input configuration
- Suppose now that the last message in Π does not reach the destination (i.e., a link failure takes place). Since Π is correct independent of link failures, consensus must be reached in any case. This means, the last message was useless, and then Π could not be shortest!

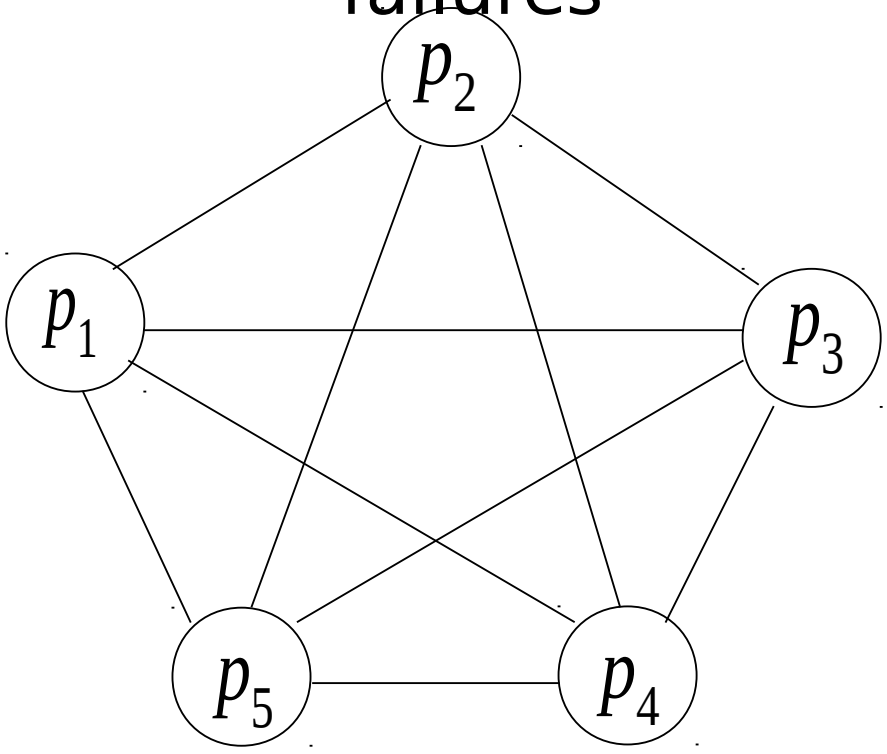
Fonte: Prof. Guido Proietti

Negative result for processor failures in asynchronous systems

- It is easy to see that a processor failure (both permanent crash and byzantine) is at least as difficult as a link failure, and then the negative result we just given holds also here
- But even worse, it is not hard to prove that in the asynchronous case, it is impossible to reach consensus for any system topology and already for a single crash failure!
- Notice that for the synchronous case it cannot be given a such general negative result ⇒ in search of some positive result, we focus on synchronous specific topologies

Fonte: Prof. Guido Proietti

Positive results: Assumption on the communication model for crash and byzantine failures



- Complete undirected graph (this implies non-uniformity)
- Synchronous network, synchronous start: w.l.o.g., we assume that messages are sent, delivered and read in the very same round

Fonte: Prof. Guido Proietti

Overview of Consensus Results

f-resilient consensus algorithms (i.e., algorithms solving consensus for f faulty processors)

	Crash failures	**Byzantine failures**
Number of rounds	f+1	2(f+1) f+1
Total number of processors	$n \geq f+1$	$n \geq 4f+1$ $n \geq 3f+1$
Message complexity	$O(n^3)$	$O(n^3)$ $O(n^{O(n)})$ (exponential)

Fonte: Prof. Guido Proietti

A simple algorithm for fault-free consensus

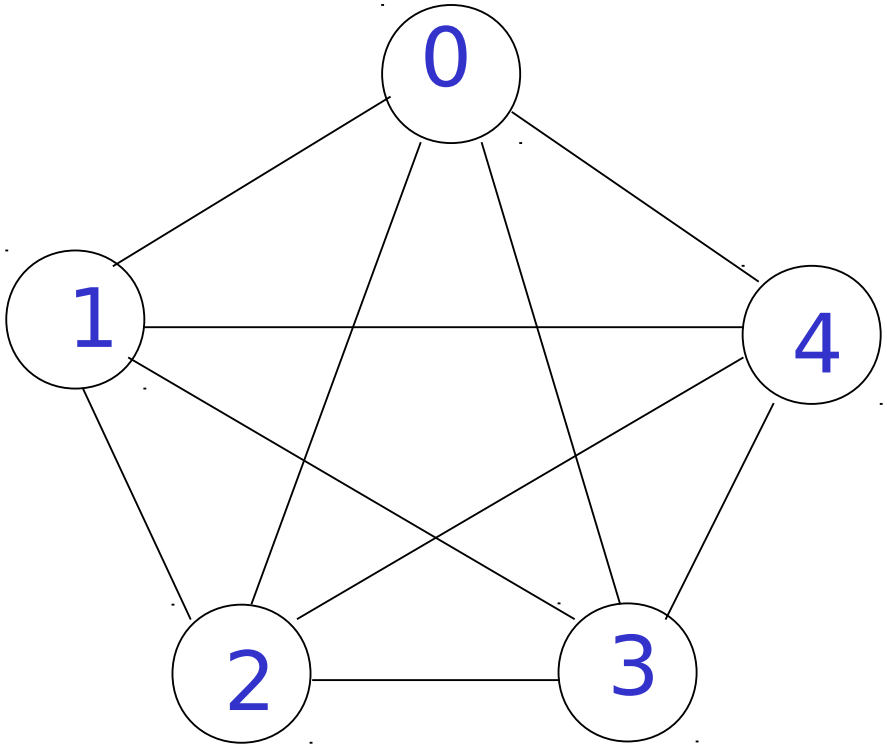
Each processor:

1. Broadcasts its input to all processors
2. Reads all the incoming messages
3. Decides on the minimum received value

(only one round is needed, since the graph is complete)

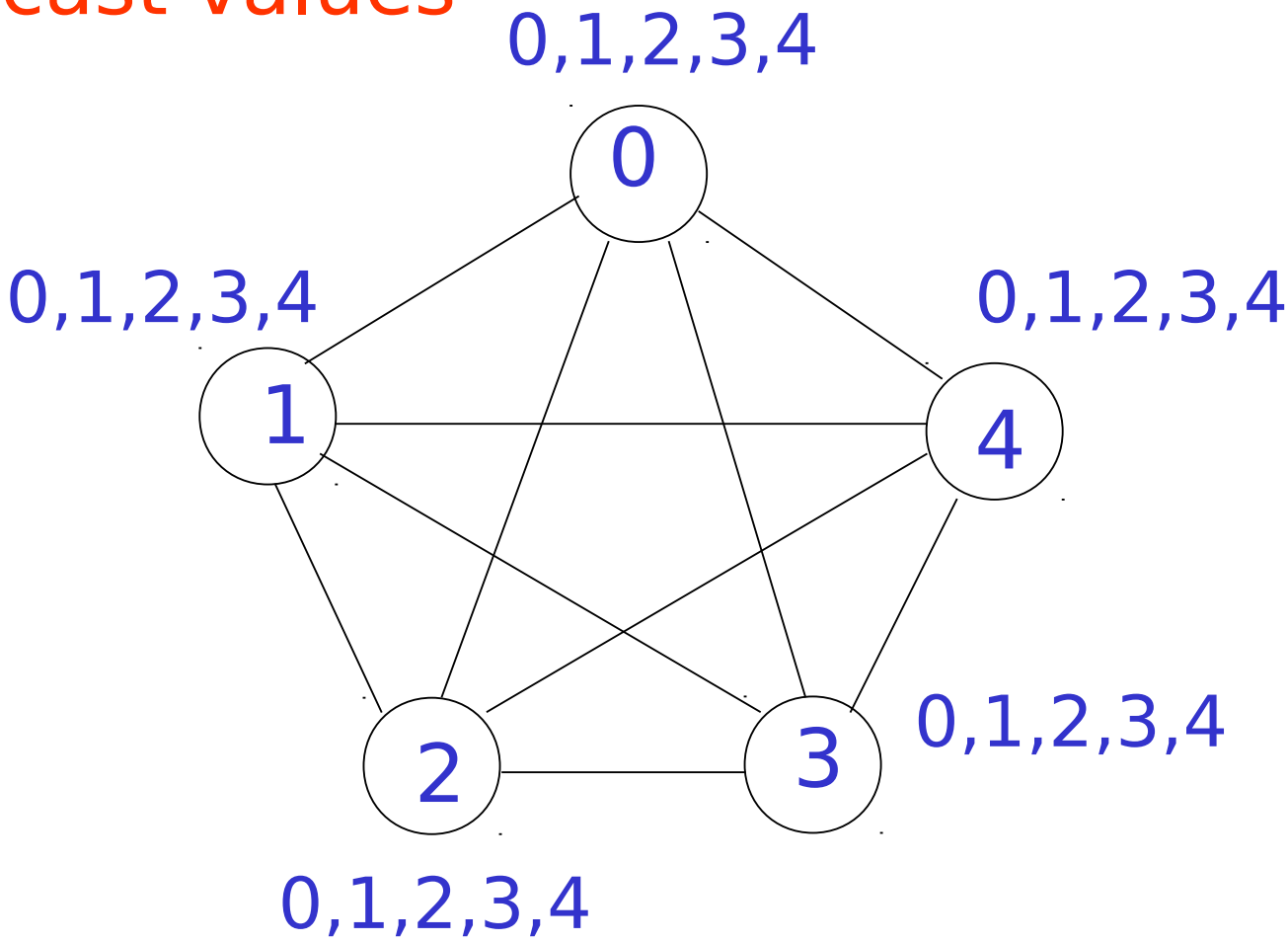
Fonte: Prof. Guido Proietti

Start



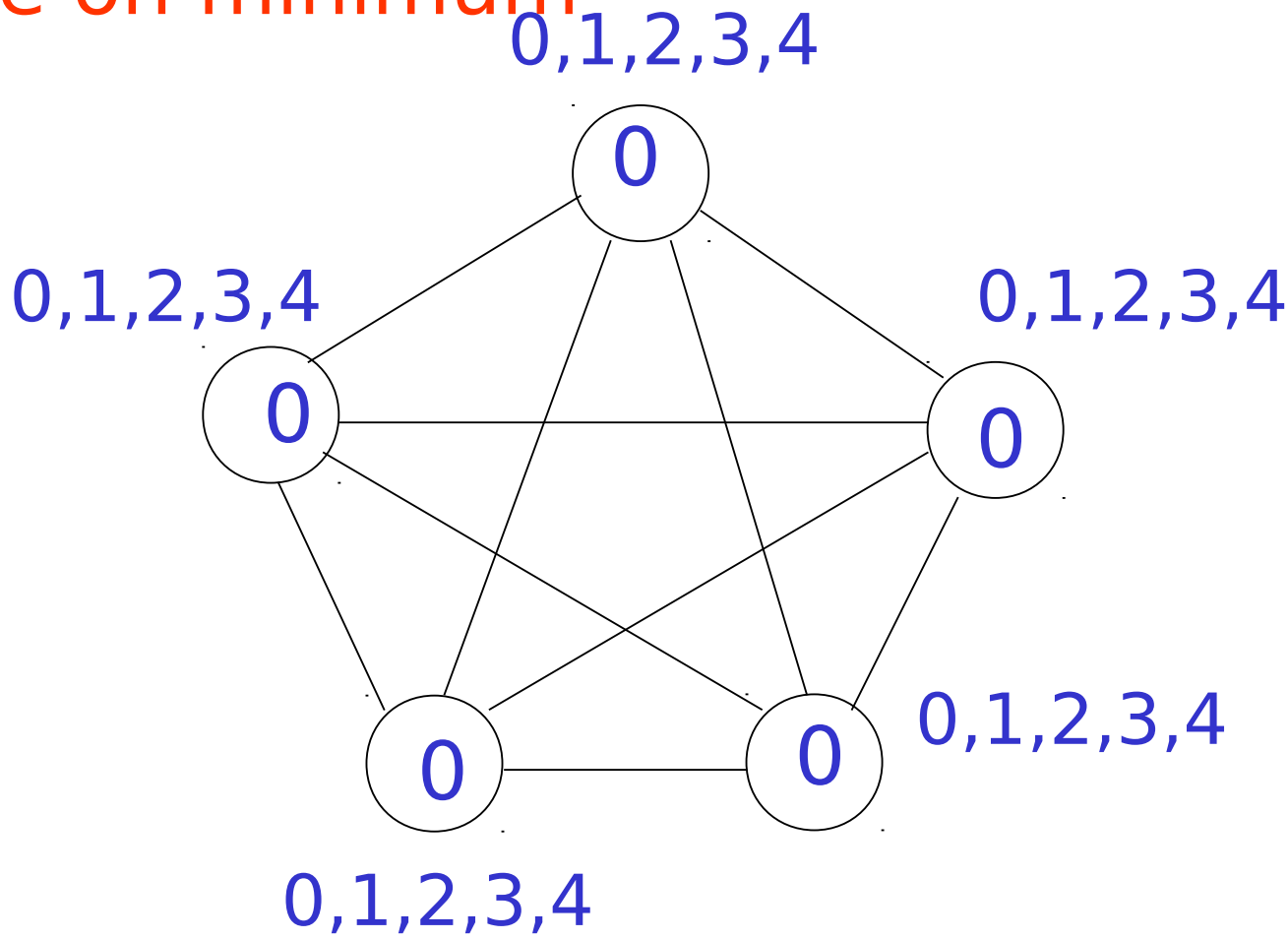
Fonte: Prof. Guido Proietti

Broadcast values



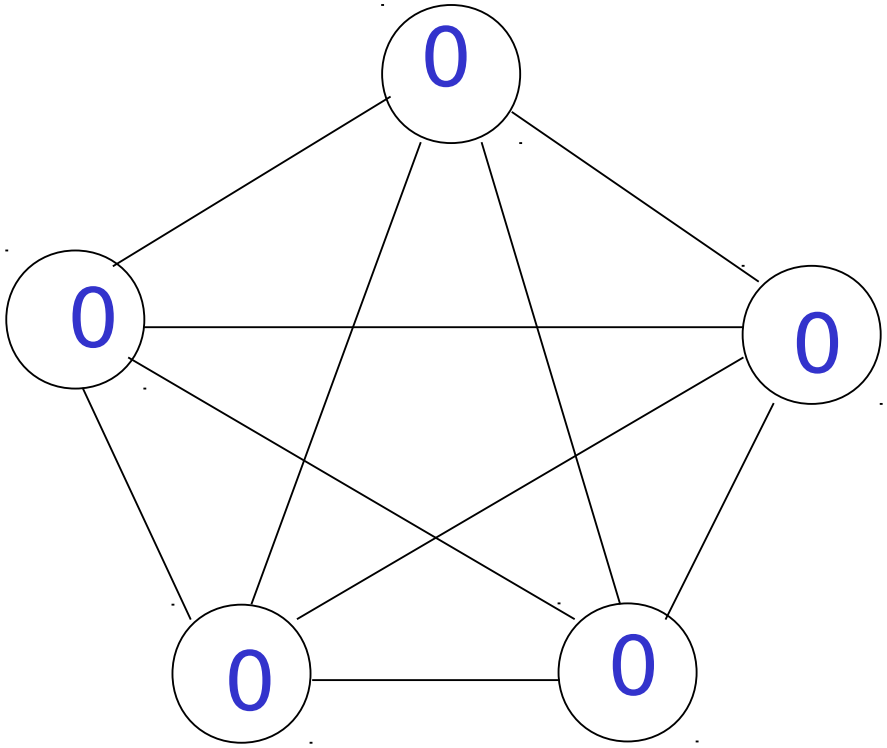
Fonte: Prof. Guido Proietti

Decide on minimum



Fonte: Prof. Guido Proietti

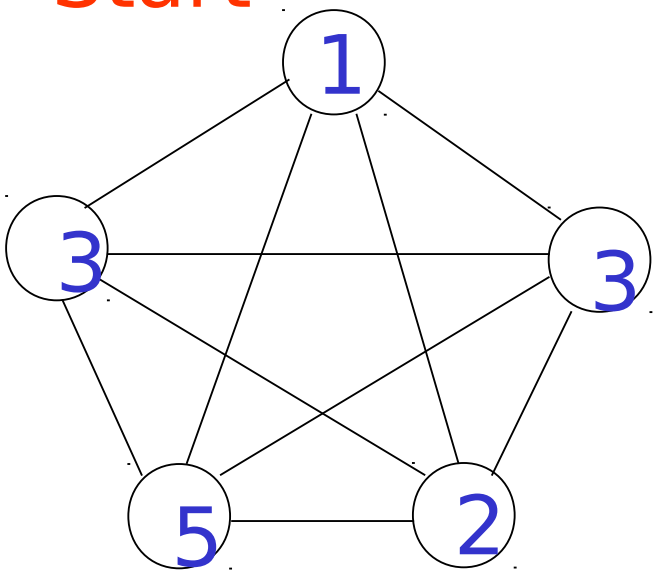
Finish



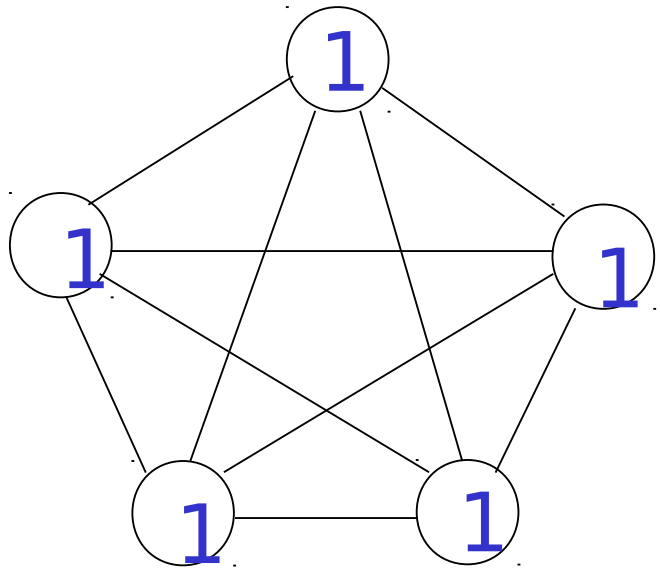
Fonte: Prof. Guido Proietti

This algorithm satisfies the agreement

Start



Finish

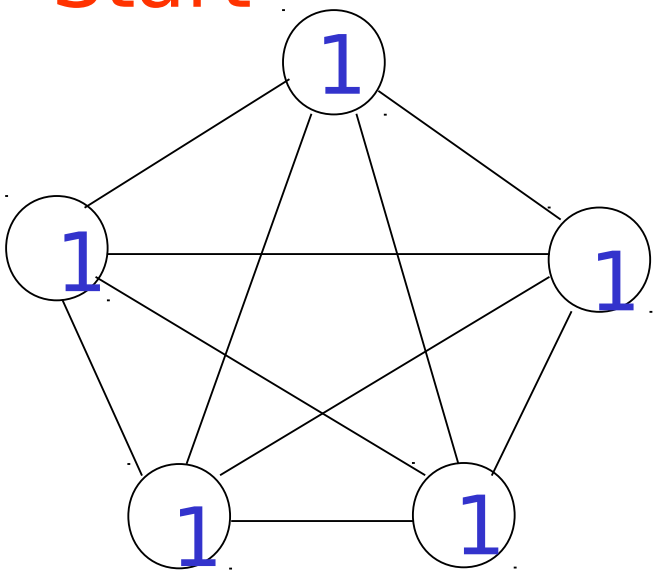


All the processors decide the minimum exactly over the same set of values

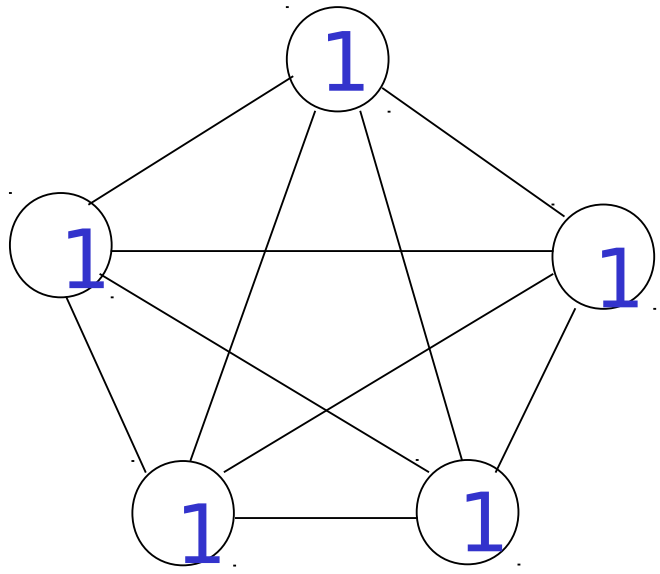
Fonte: Prof. Guido Proietti

This algorithm satisfies the validity conditio

Start



Finish

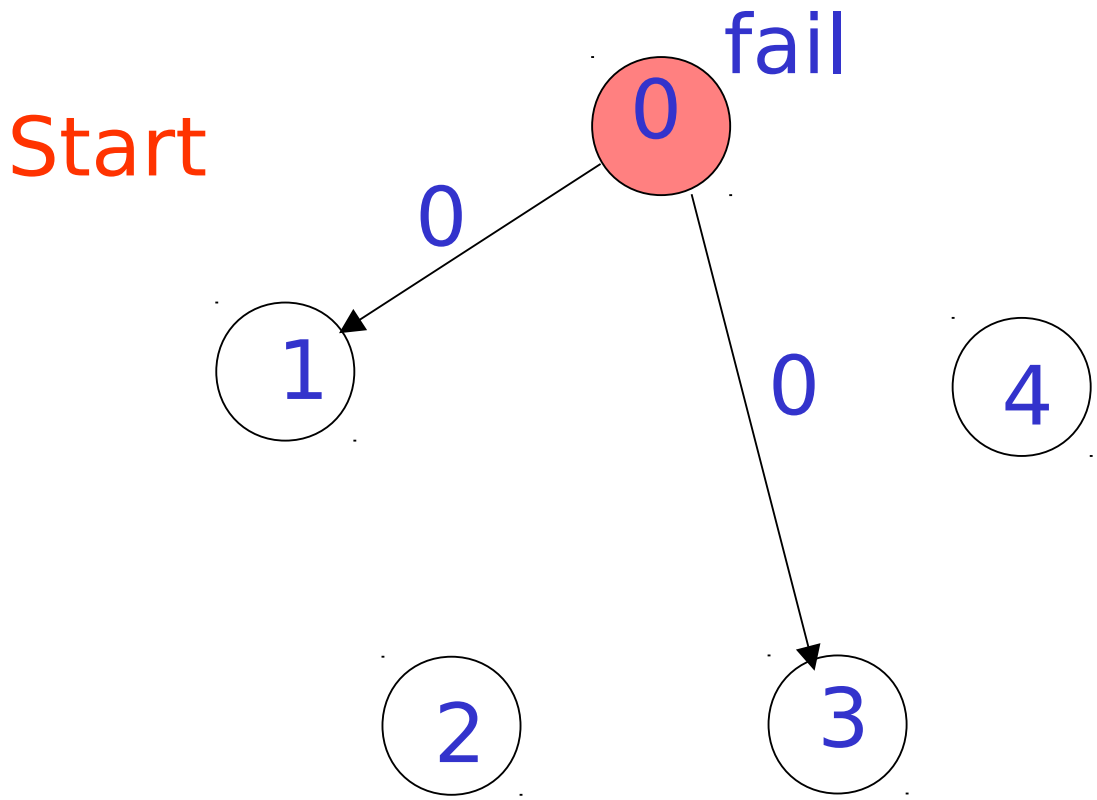


If everybody starts with the same initial valu
everybody decides on that value (minimum)

Fonte: Prof. Guido Proietti

Consensus with Crash Failures

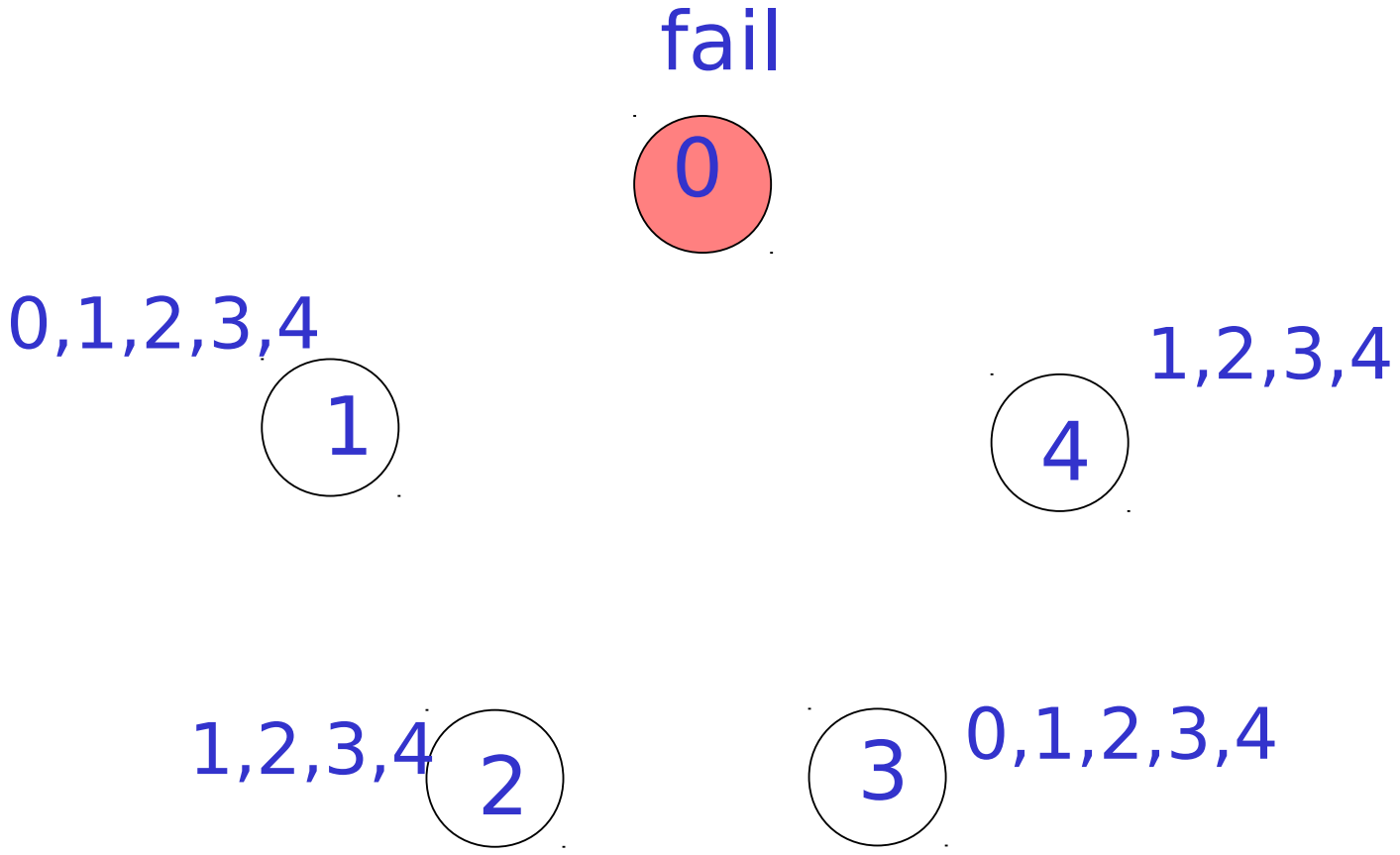
The simple algorithm doesn't work



The failed processor doesn't broadcast its value to all processors

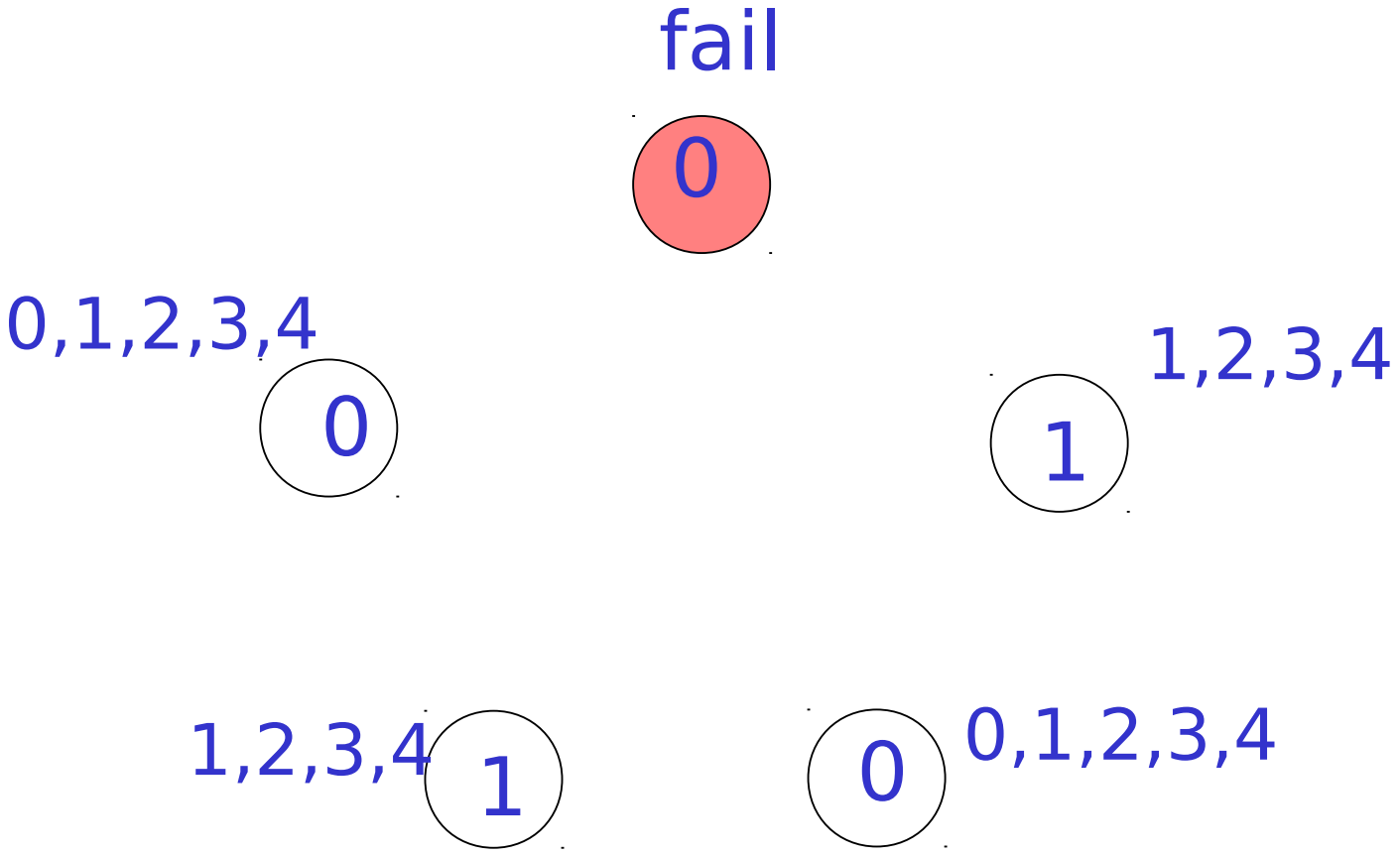
Fonte: Prof. Guido Proietti

Broadcasted values



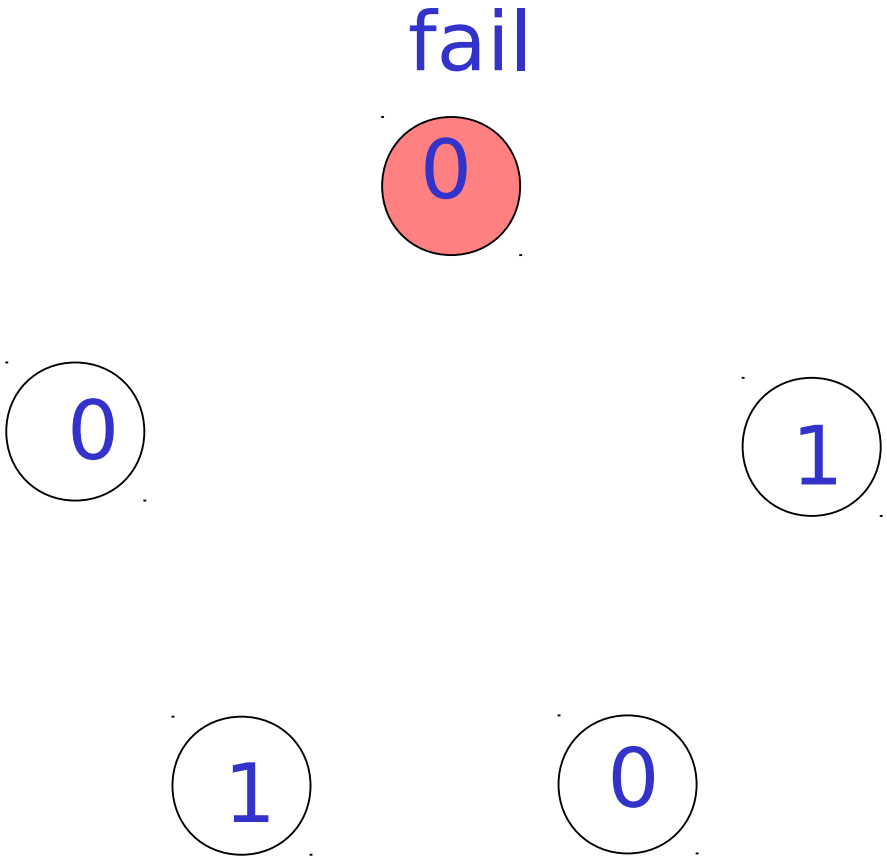
Fonte: Prof. Guido Proietti

Decide on minimum



Fonte: Prof. Guido Proietti

Finish



No agreement!!!

Fonte: Prof. Guido Proietti

An f-resilient to crash failures algorithm

Each processor:

Round 1:

Broadcast to all (including myself) my value;

Read all the incoming values;

Round 2 to round f+1:

Broadcast to all (including myself) any new received values:

Read all the incoming values;

End of round f+1:

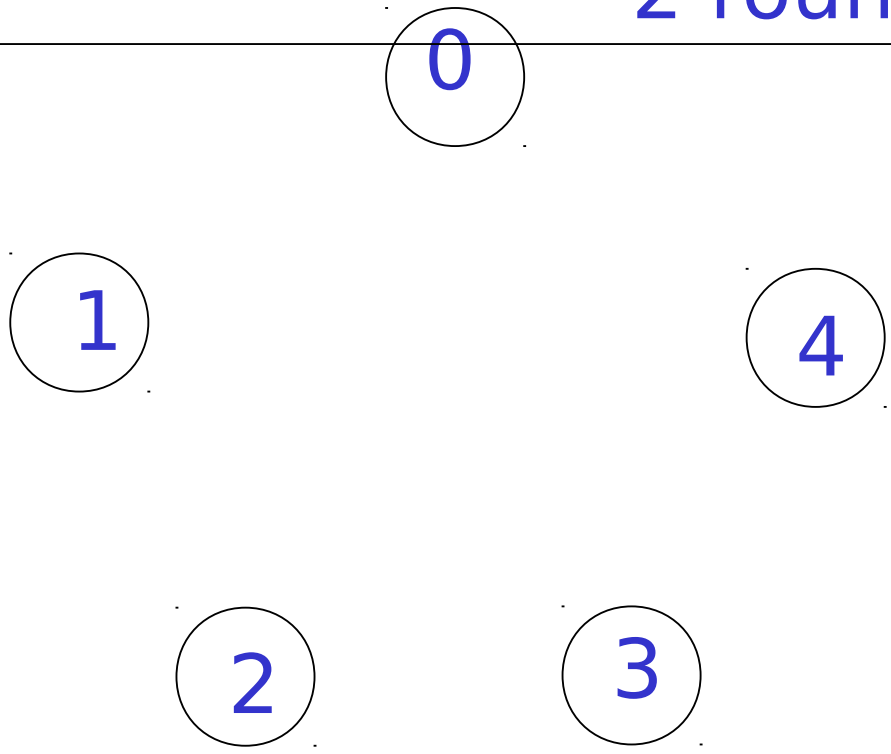
Decide on the minimum value ever received.

Fonte: Prof. Guido Proietti

Example: f=1 failures, f+1 =

Start

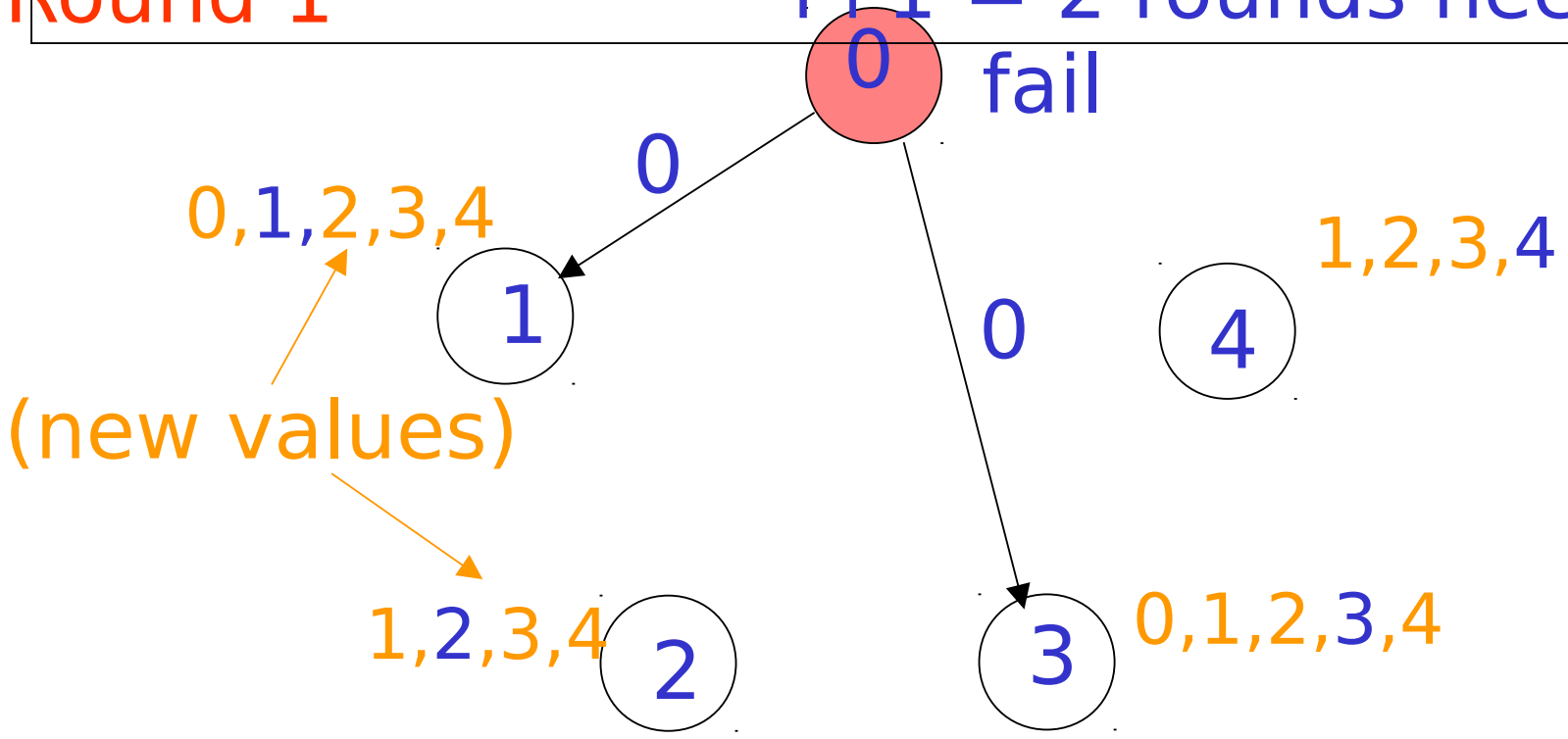
2 rounds needed



Fonte: Prof. Guido Proietti

Example: f=1 failures,

Round 1 f+1 = 2 rounds needed

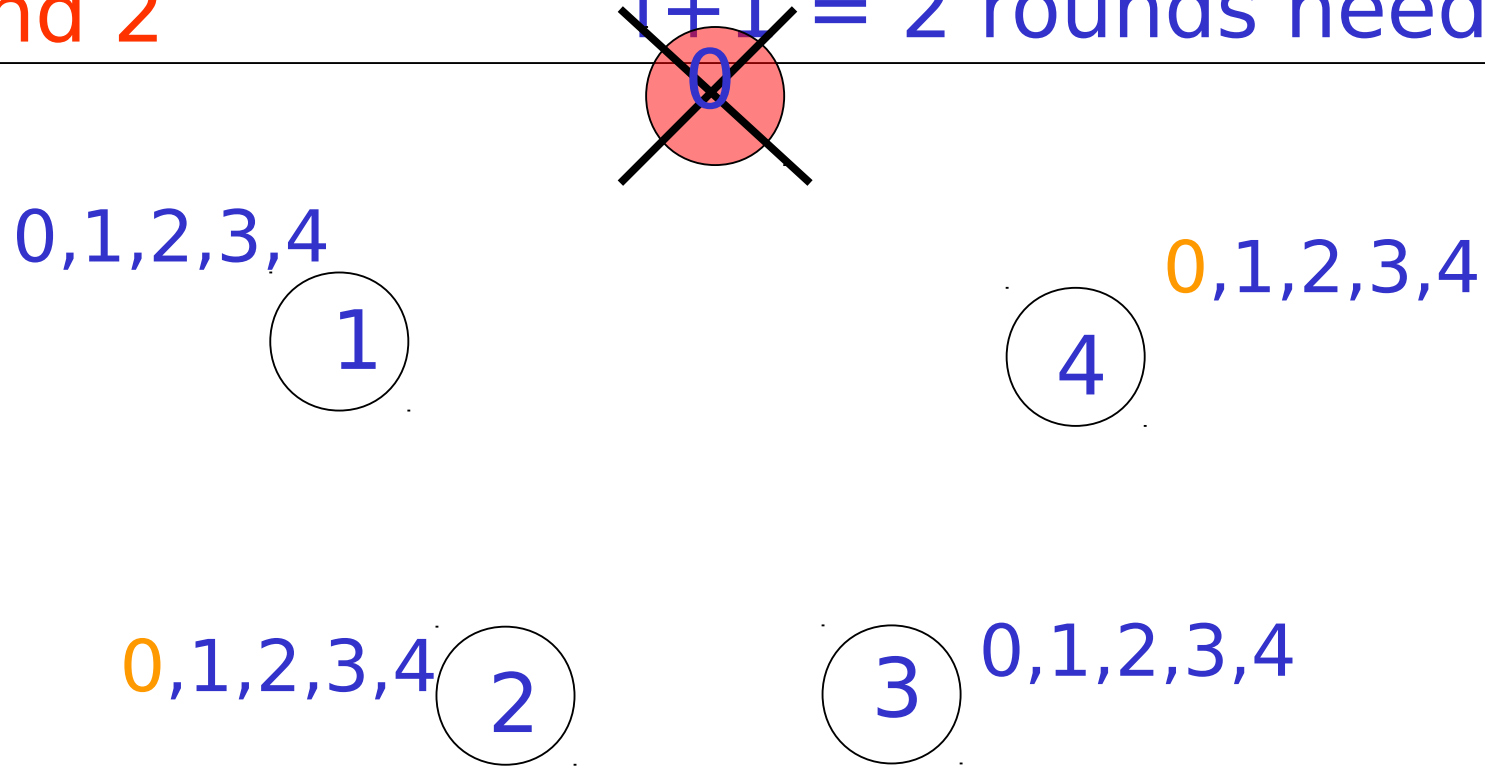


Broadcast all values to everybody

Fonte: Prof. Guido Proietti

Example: f=1 failures,

Round 2 f+1 = 2 rounds needed

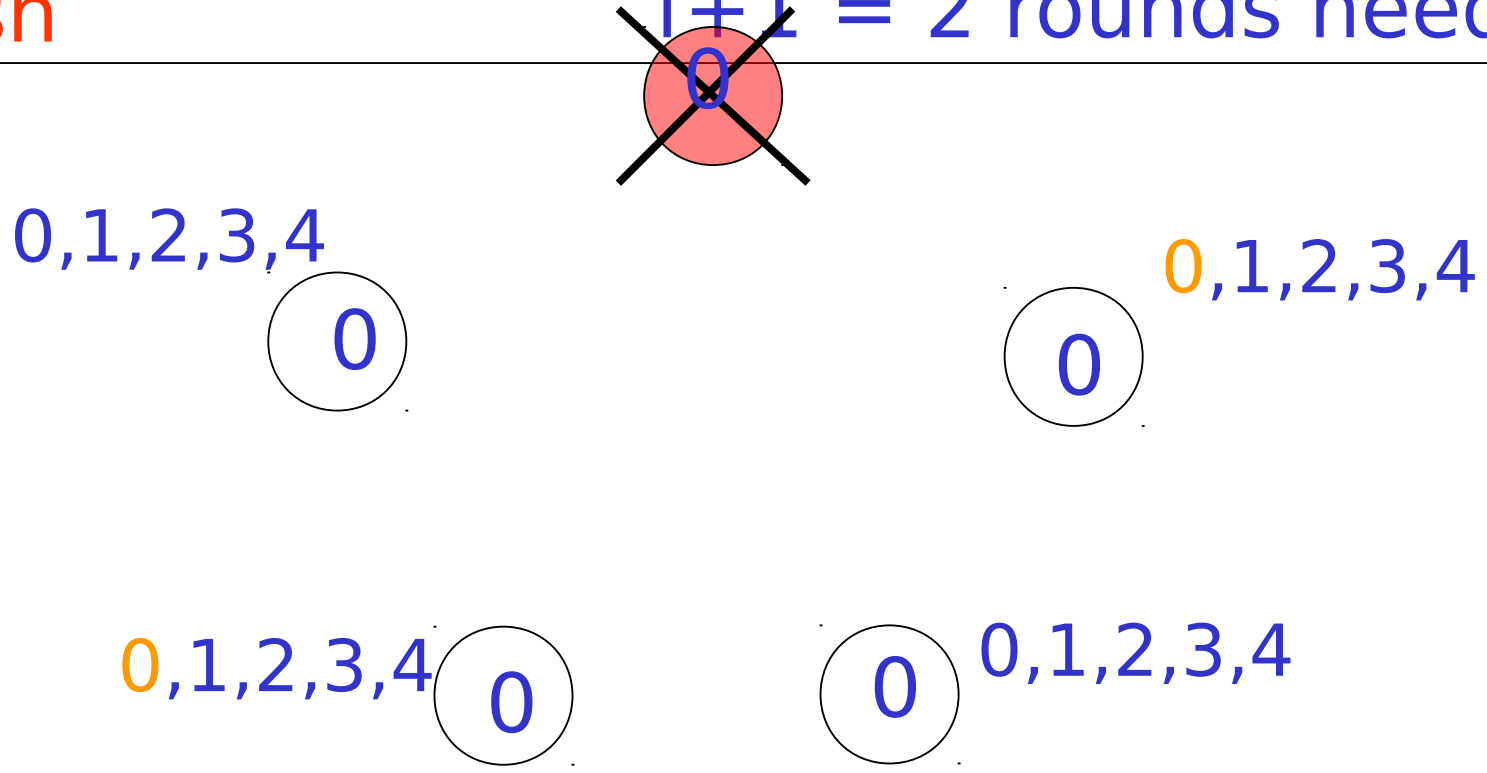


Broadcast all new values to everybody

Fonte: Prof. Guido Proietti

Example: f=1 failures,

Finish f+1 = 2 rounds needed

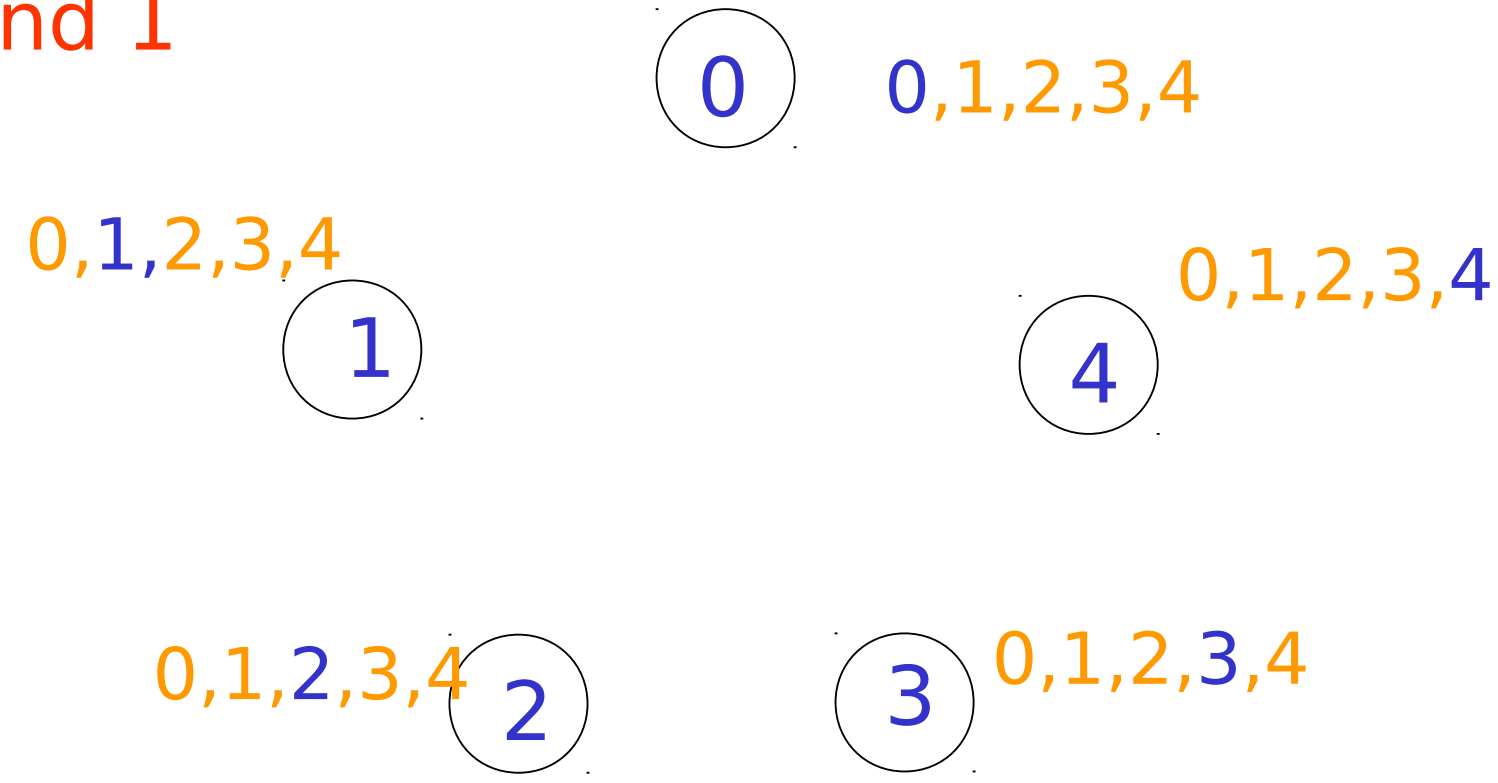


Decide on minimum value

Fonte: Prof. Guido Proietti

Example: f=1 failures, f+1 = 2 rounds need

Round 1

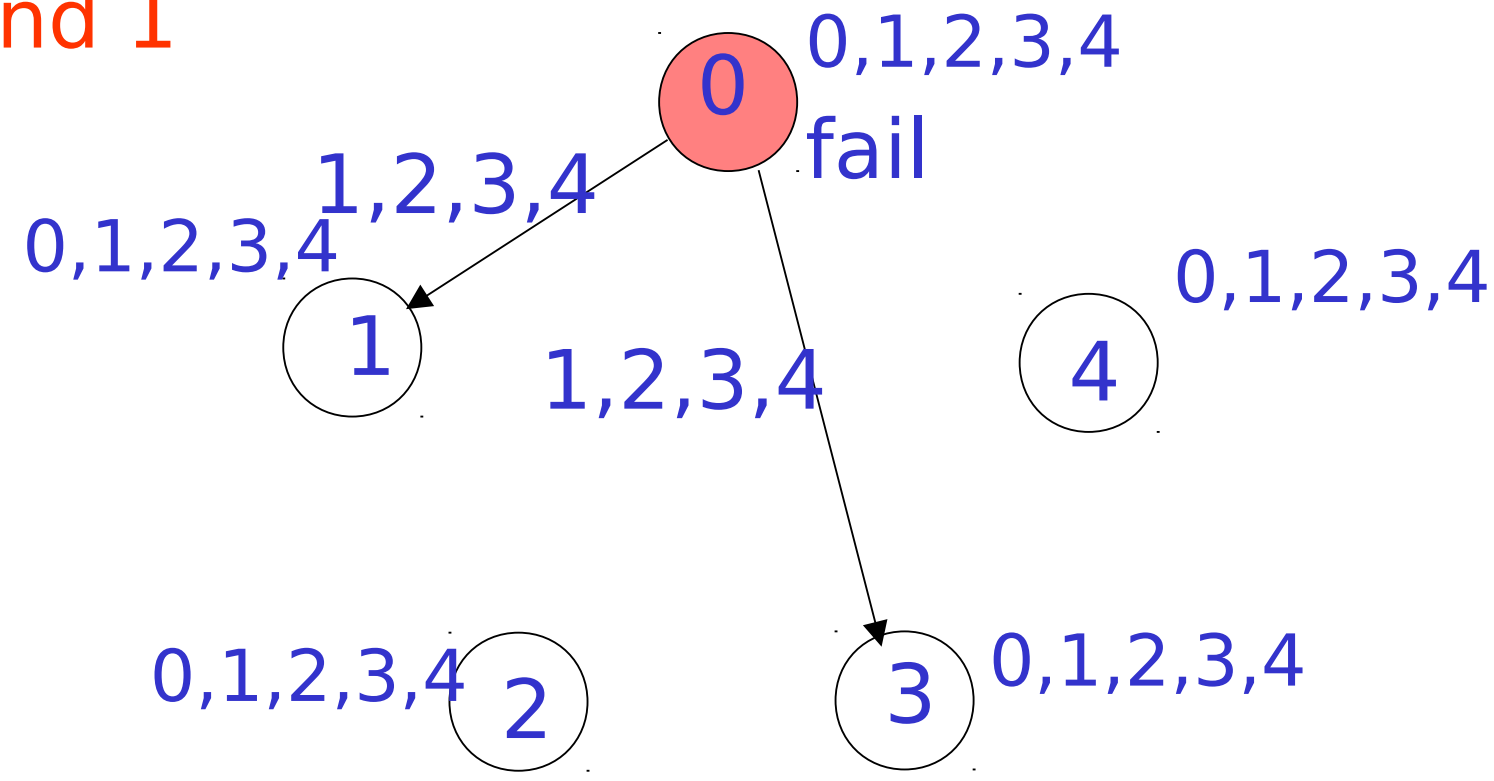


No failures: all values are broadcasted to all

Fonte: Prof. Guido Proietti

Example: f=1 failures, f+1 = 2 rounds neec

Round 1

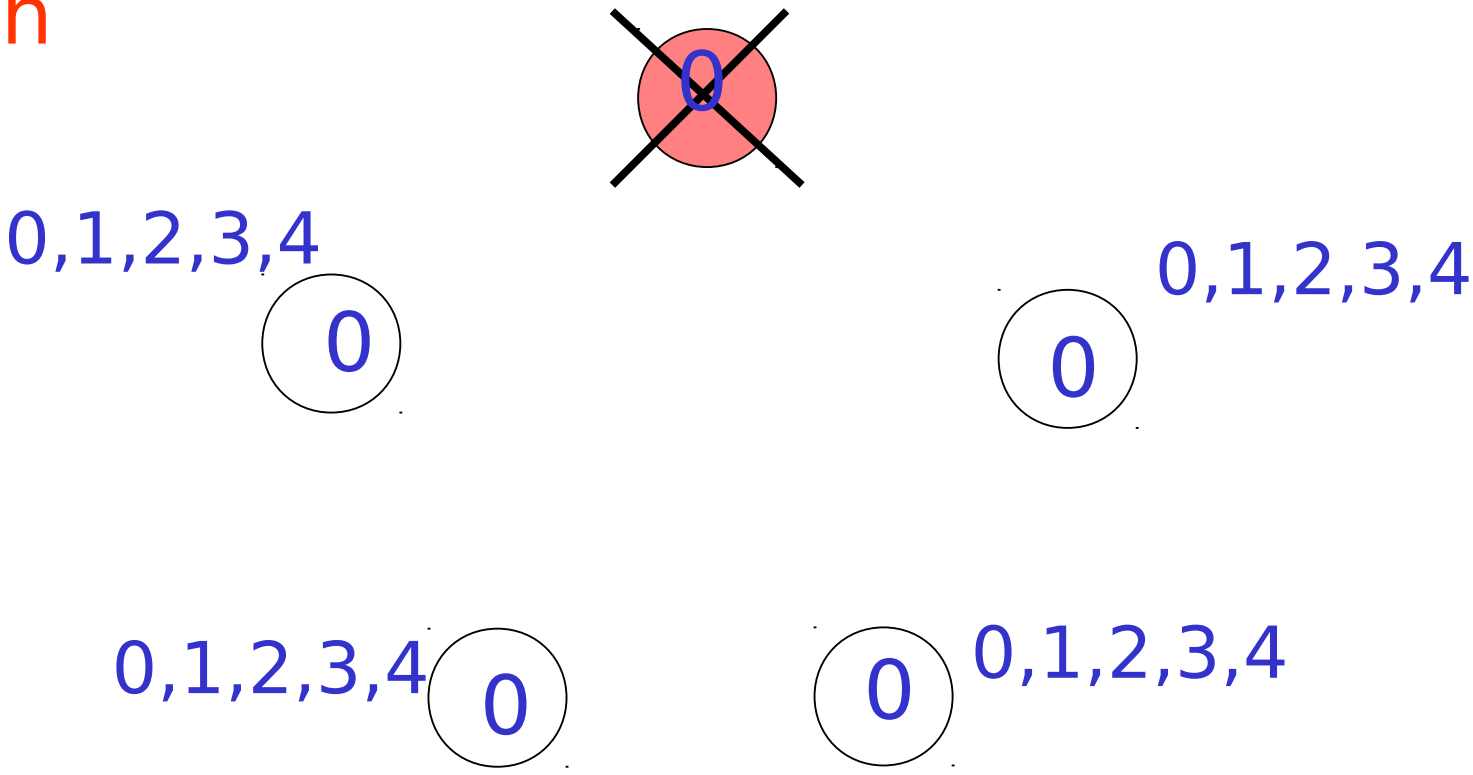


No problems: processors “2” and “4” have already seen 1,2,3 and 4 in the previous round

Fonte: Prof. Guido Proietti

Example: f=1 failures, f+1 = 2 rounds neec

Finish

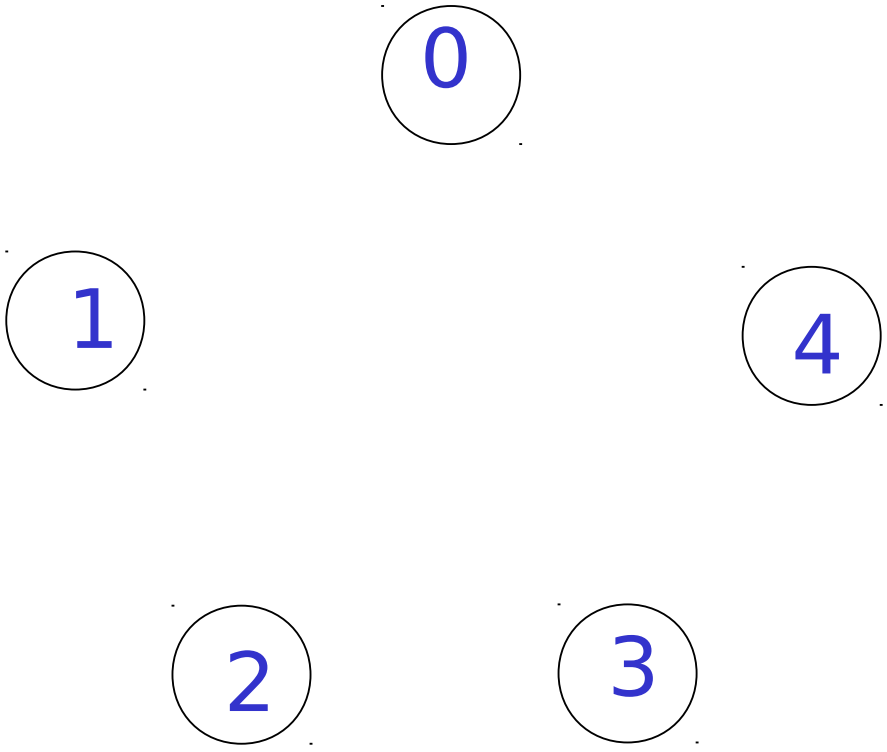


Decide on minimum value

Fonte: Prof. Guido Proietti

Example: f=2 failures, f+1 = 3 rounds need

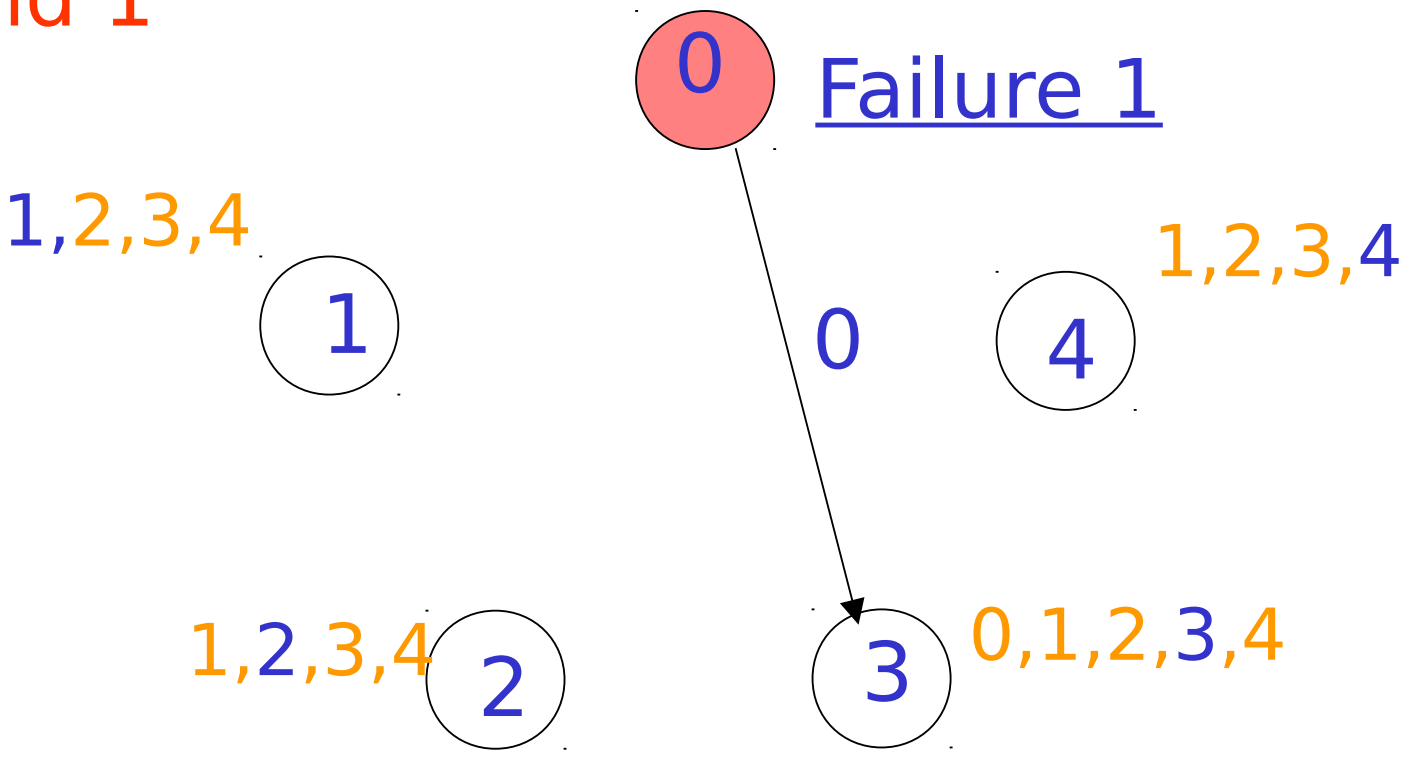
Start



Fonte: Prof. Guido Proietti

Example: f=2 failures, f+1 = 3 rounds need

Round 1

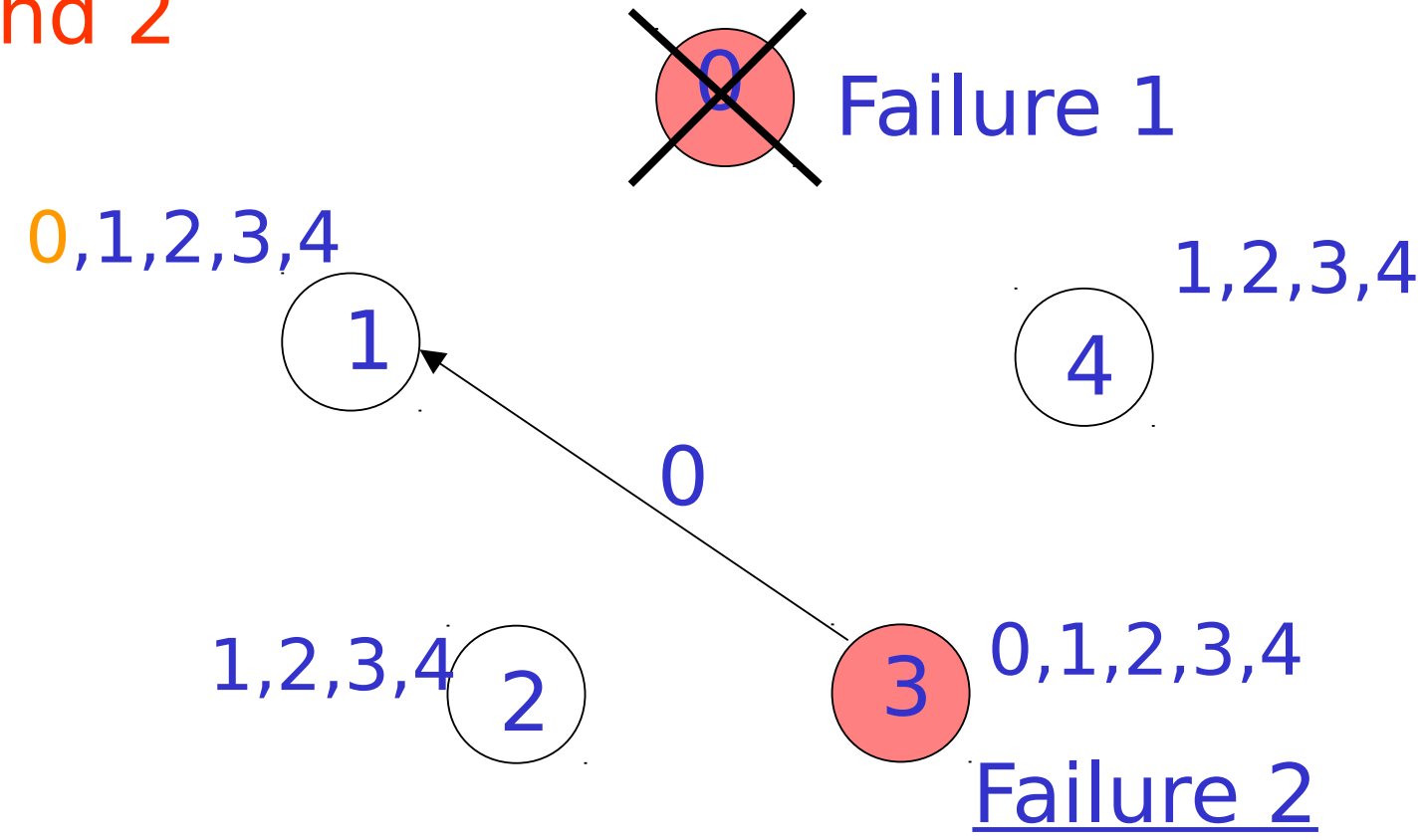


Broadcast all values to everybody

Fonte: Prof. Guido Proietti

Example: f=2 failures, f+1 = 3 rounds need

Round 2

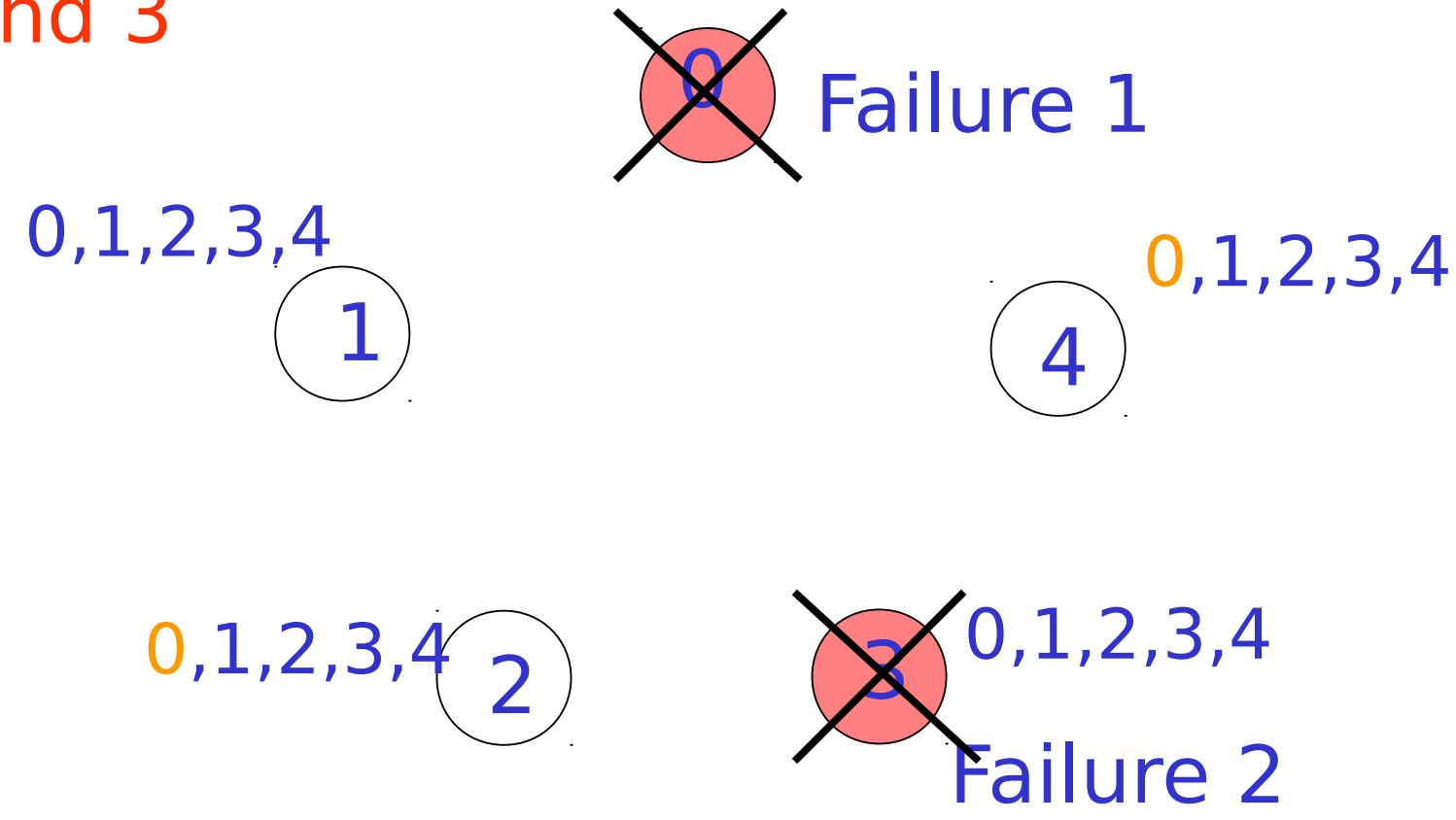


Broadcast new values to everybody

Fonte: Prof. Guido Proietti

Example: f=2 failures, f+1 = 3 rounds need

Round 3

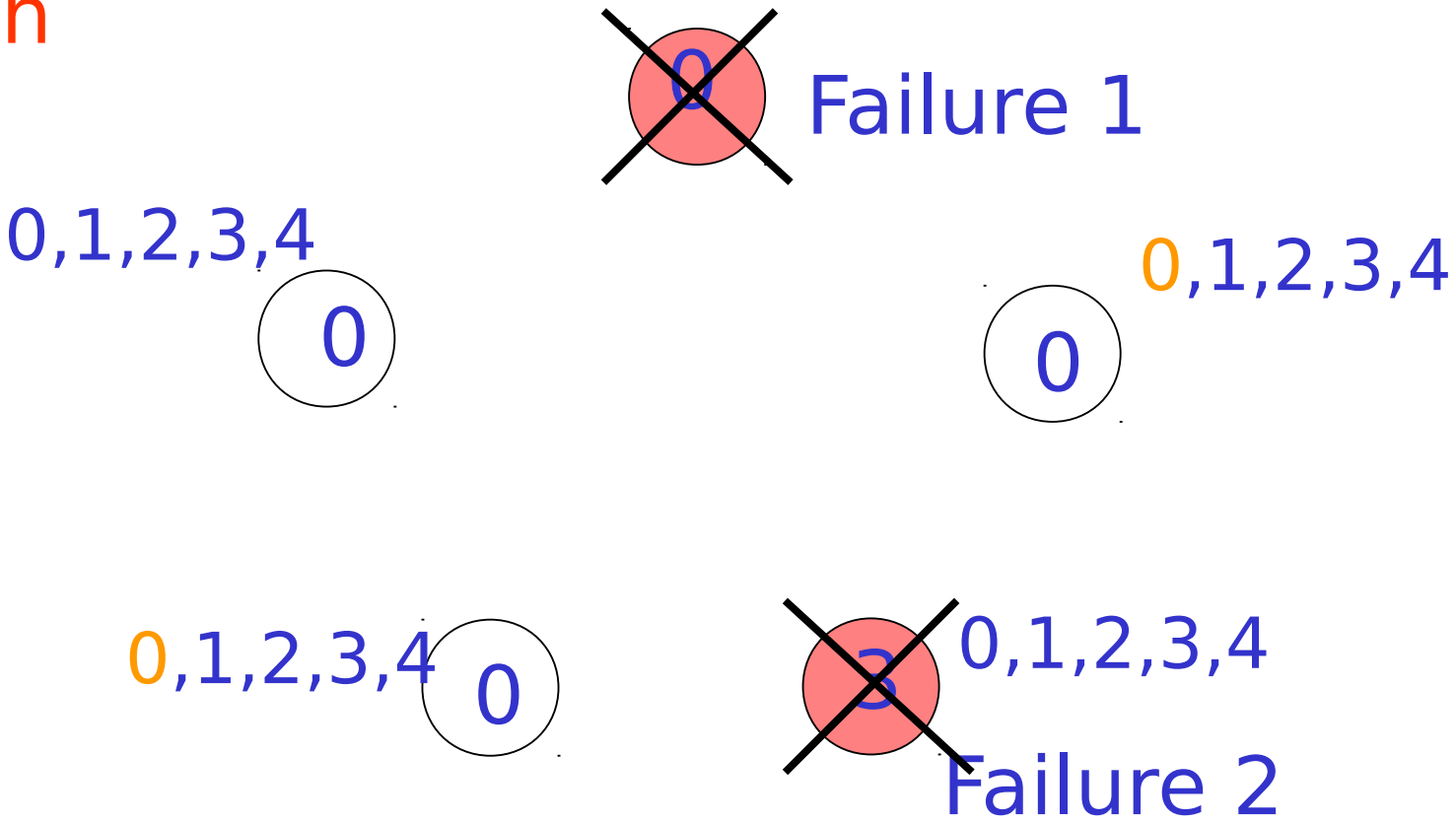


Broadcast new values to everybody

Fonte: Prof. Guido Proietti

Example: f=2 failures, f+1 = 3 rounds need

Finish



Decide on the minimum value

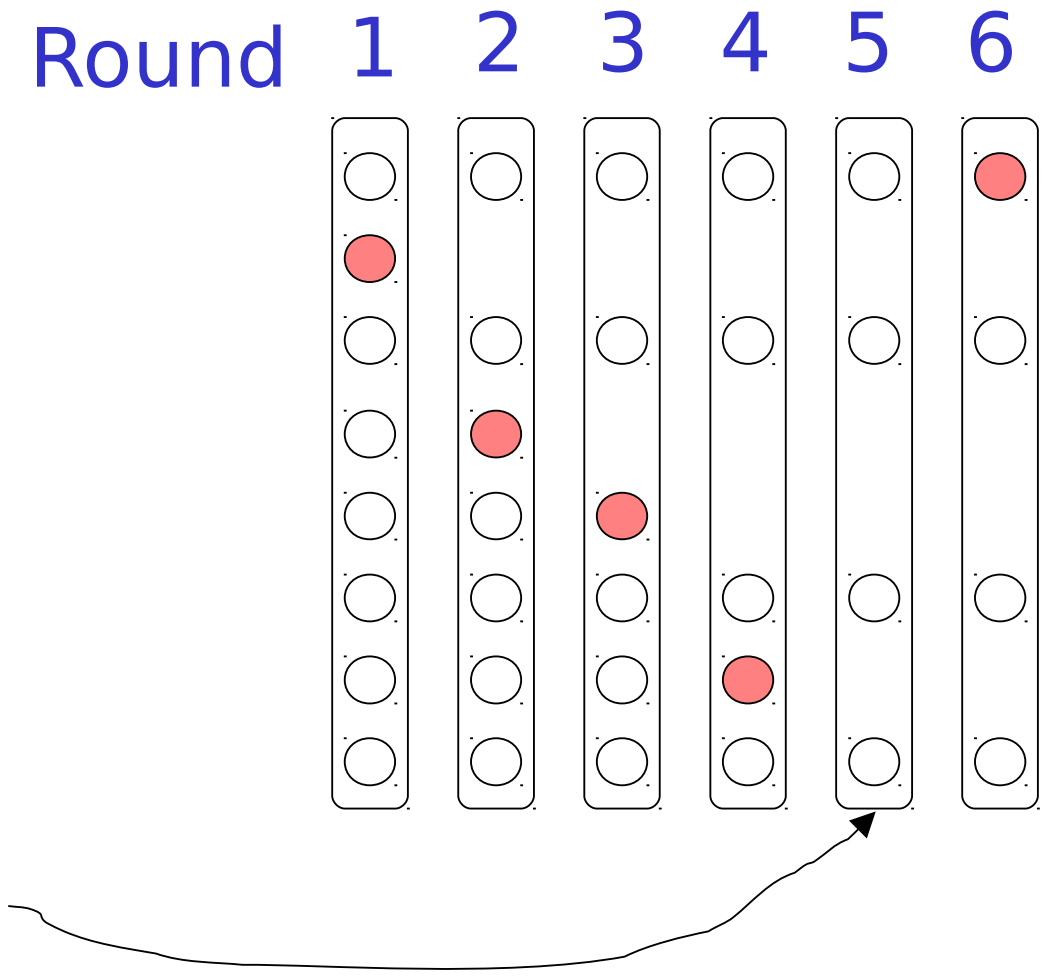
Fonte: Prof. Guido Proietti

In general, since there are f failures and f+1 rounds, then there is at least a round with no failed processors:

Example:

5 failures,

6 rounds



Fonte: Prof. Guido Proietti

Correctness (1/2)

Lemma: In the algorithm, at the end of the round with no failures, all the non-faulty processors know the same set of values.

Proof: For the sake of contradiction, assume the claim is false. Let x be a value which is known only to a subset of non-faulty processors at the end of the round with no failures. Observe that any such processors cannot have known x for the first time in a previous round, since otherwise it had broadcasted it to all. So, the only possibility is that it received it right in this round, otherwise all the others should know x as well. But in this round there are no failures, and so x must be received and known by all, a contradiction.

QED

Fonte: Prof. Guido Proietti

Correctness (2/2)

Agreement: this holds, since at the end of the round with no failures, every (non-faulty) processor has the same knowledge, and this doesn't change until the end of the algorithm (no new values can be introduced) $\Rightarrow$ eventually, everybody will decide the same value!

Remark: we don't know the exact position of the free-of-failures round, so we have to let the algorithm execute for f+1 rounds

Validity: this holds, since the value decided from each processor is some input value (no exogenous values are introduced)

Fonte: Prof. Guido Proietti

Performance of Crash Consensus Algorithm

- Number of processors: $n > f$
- $f+1$ rounds
- $O(n^2 \bullet k)=O(n^3)$ messages, where $k=O(n)$ is the number of different inputs. Indeed, each node sends $O(n)$ messages containing a given value in X

Fonte: Prof. Guido Proietti

A Lower Bound

Theorem: Any f-resilient consensus algorithm with crash failures requires at least f+1 rounds

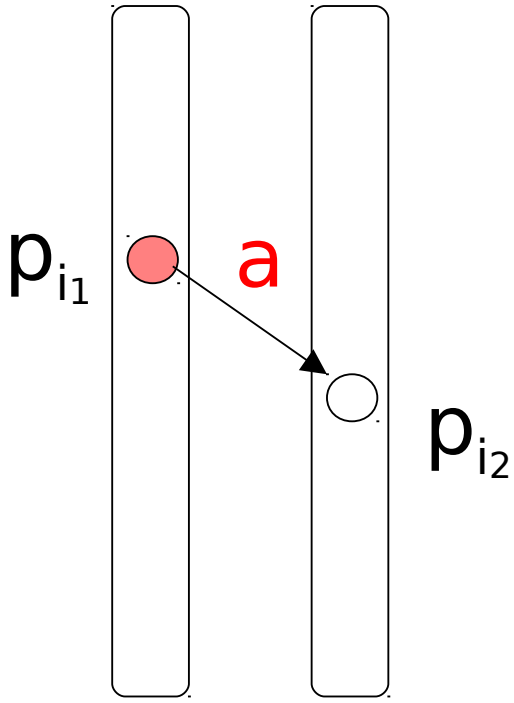
Proof sketch: Assume by contradiction that f or less rounds are enough. Clearly, every algorithm which solves consensus requires that eventually non-faulty processors have the very same knowledge

Worst case scenario:
There is a processor that fails in each round

Fonte: Prof. Guido Proietti

Worst case scenario

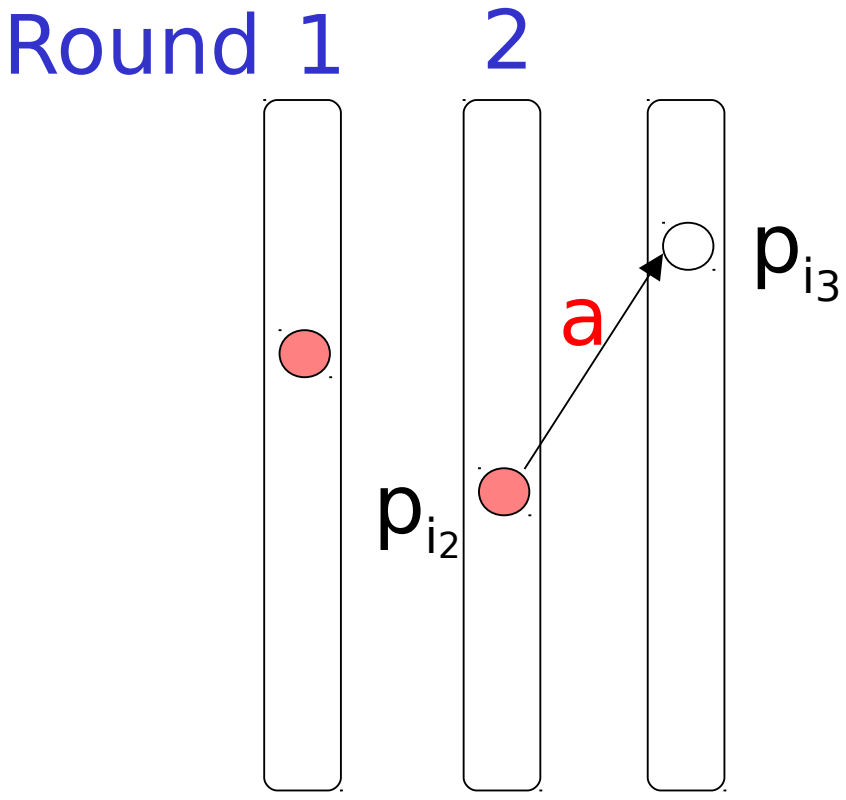
Round 1



before processor p_{i_1} fails, it sends its value a to only one processor p_{i_2}

Fonte: Prof. Guido Proietti

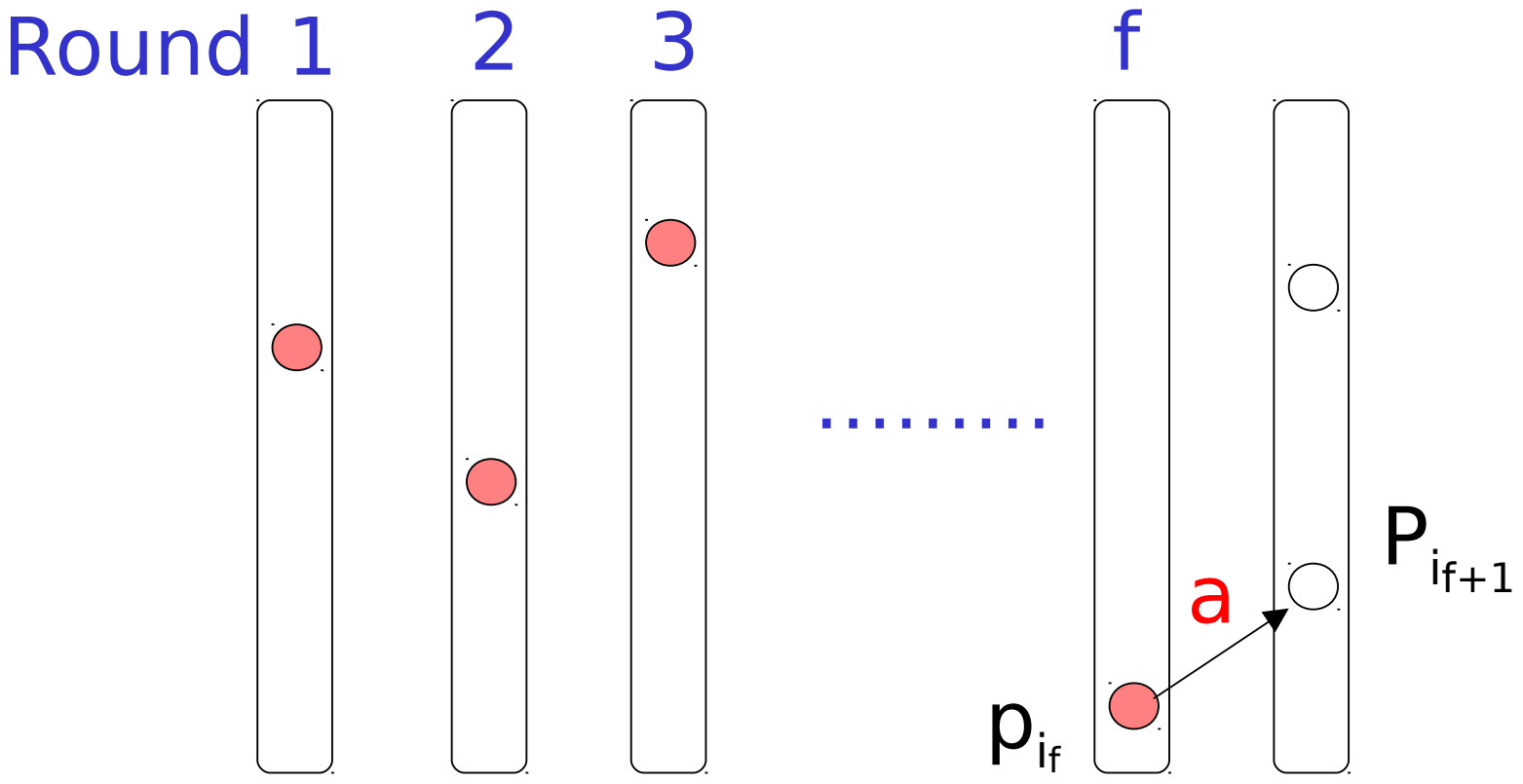
Worst case scenario



before processor p_{i_2} fails, it sends its value a to only one processor p_{i_3}

Fonte: Prof. Guido Proietti

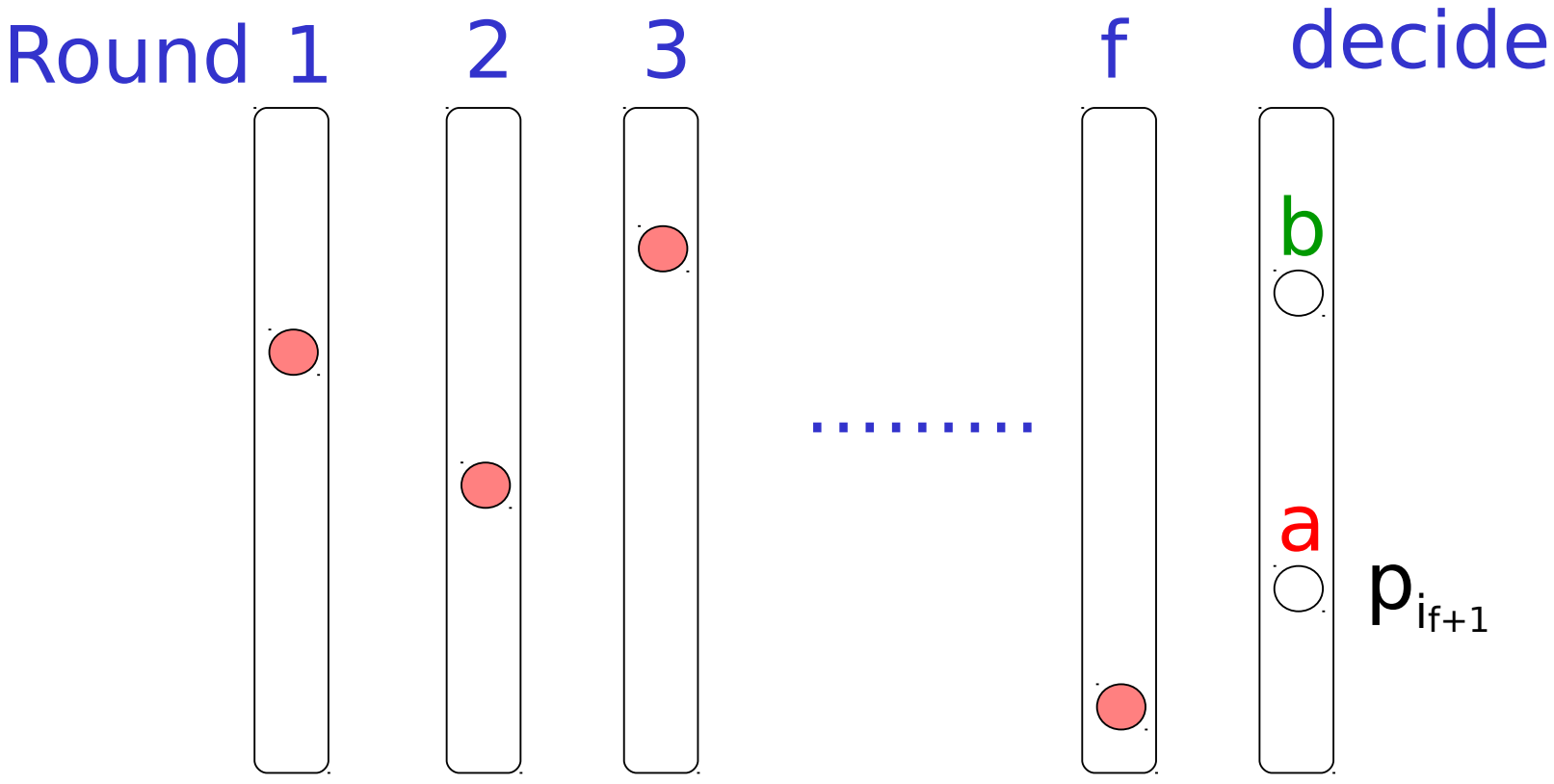
Worst case scenario



Before processor p_{i_f} fails, it sends its value a to only one processor $p_{i_{f+1}}$. Thus, at the end of round f only one processor knows

Fonte: Prof. Guido Proietti

Worst case scenario



No agreement: Processor $p_{i_{f+1}}$ has a different knowledge, i.e., it may decide a, and all other processors may decide another value, say b>a

⇒ contradiction, f rounds are not enough. **QED**

Fonte: Prof. Guido Proietti

Consensus with Byzantine Failures

f-resilient to byzantine failures consensus algorithm:

solves consensus for

f byzantine processors

Fonte: Prof. Guido Proietti

Lower bound on number of rounds

Theorem: Any f-resilient consensus algorithm with byzantine failures requires at least f+1 rounds

Proof:

follows from the crash failure lower bound

Fonte: Prof. Guido Proietti

An f-resilient to byzantine failures algorithm

The **King** algorithm

solves consensus in 2(f+1) rounds with:

n processors and f failures, where $f < \frac{n}{4}$ (i.e., n≥4f+1)

Assumption: Processors have (distinct) ids, and these are in {1,...,n} (and so the system is non anonymous), and we denote by p_i the processor with id i; this is common

Fonte: Prof. Guido Proietti

The **King** algorithm

There are f+1 phases; each phase has 2 rounds, used to update in each processor p_i a preferred value v_i. In the beginning, the preferred value is set to the input value

In each phase there is a different king

$\Rightarrow$ There is a king that is non-faulty!

Fonte: Prof. Guido Proietti

The **King** algorithm Phase k

Round 1, processor p_i :

- Broadcast to all (including myself) preferred value v_i
- Let a be the majority of received values (including v_i)
  (in case of tie pick an arbitrary value)
- Set $v_i = a$

Fonte: Prof. Guido Proietti

The **King** algorithm Phase k

Round 2, king p_k :

Broadcast (to the others) new preferred value v_k

Round 2, processor p_i :

If v_i had majority of less than $\frac{n}{2}+f+1$

then set $v_i = v_k$

Fonte: Prof. Guido Proietti

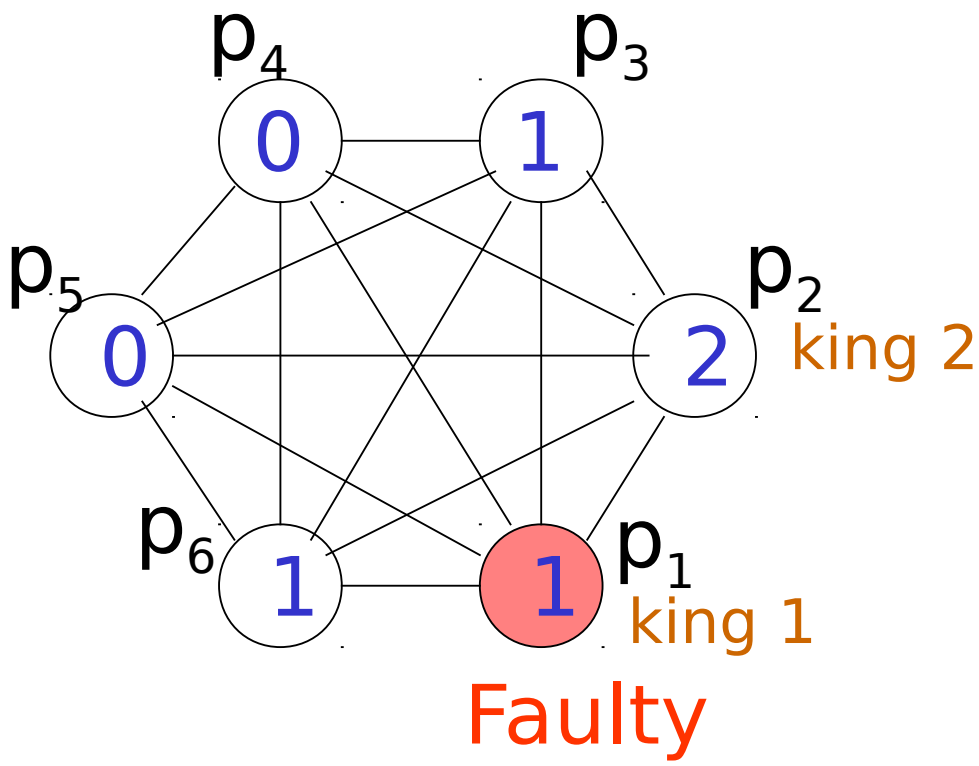
The **King** algorithm

End of Phase f+1:

Each processor decides on preferred value

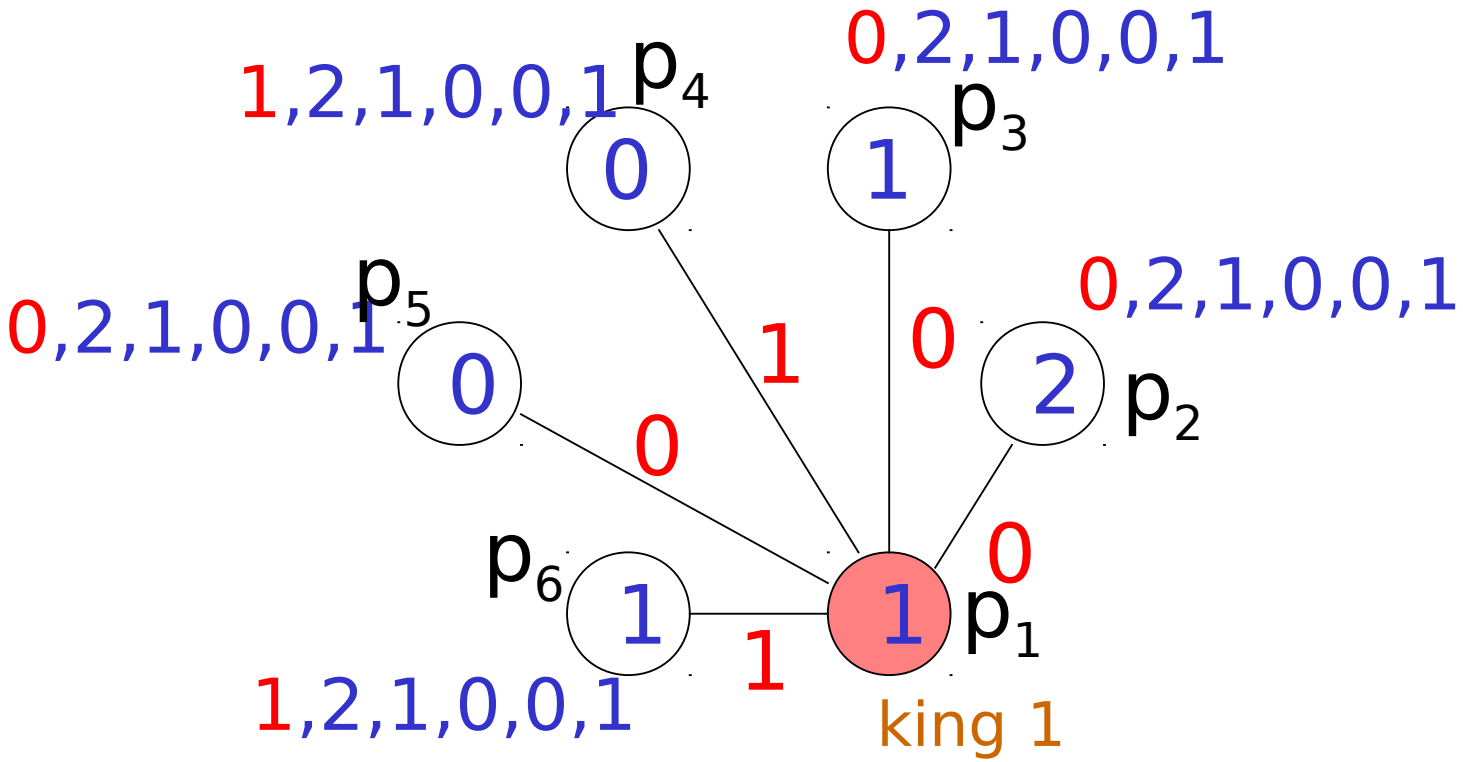
Fonte: Prof. Guido Proietti

Example: 6 processors, 1 fault, 2 phases



Fonte: Prof. Guido Proietti

Phase 1, Round 1

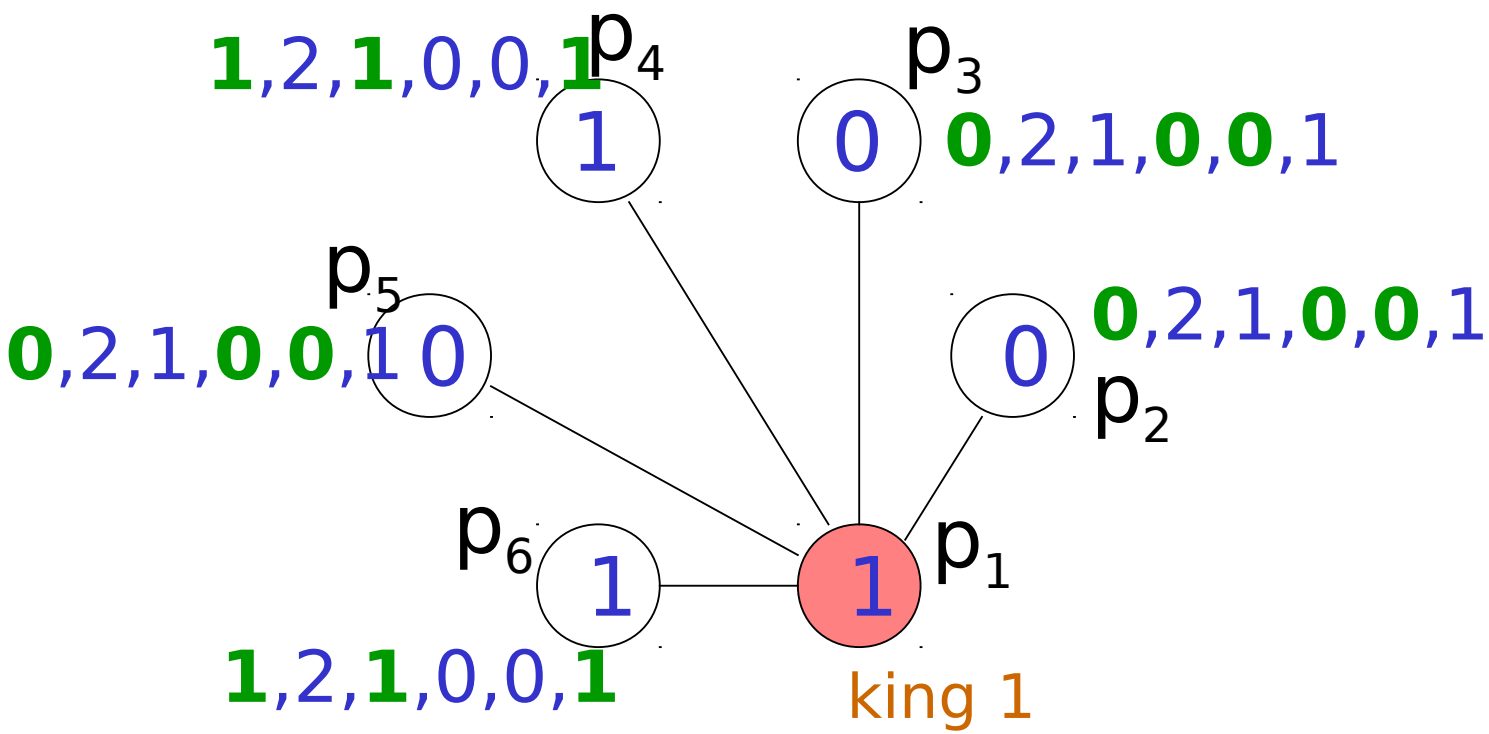


Everybody broadcasts

Fonte: Prof. Guido Proietti

Phase 1, Round 1

Choose the majority

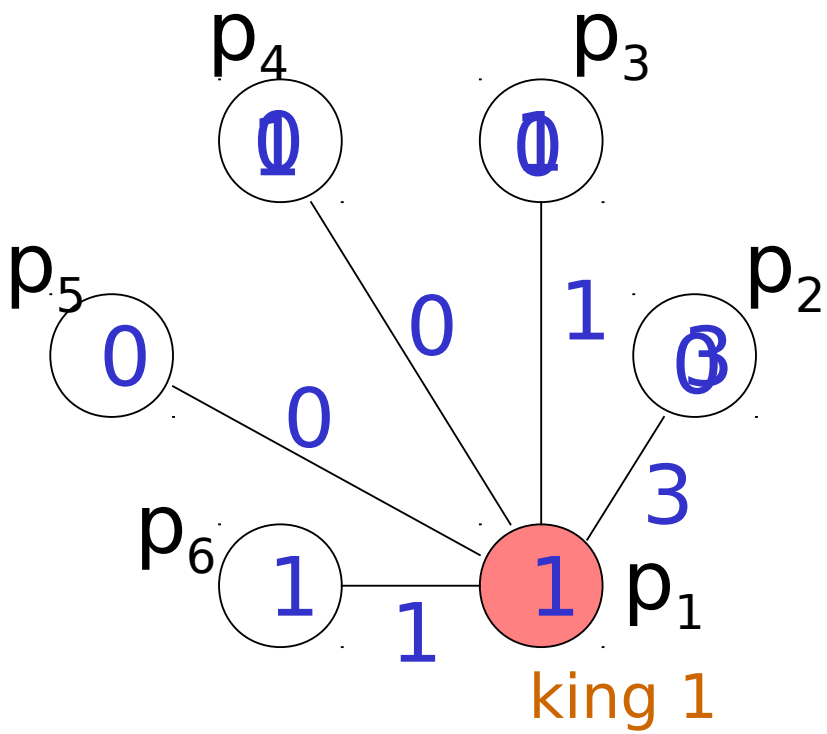


Each majority is equal to $3 < \frac{n}{2} + f + 1 = 5$

⇒ On round 2, everybody will choose the king's value

Fonte: Prof. Guido Proietti

Phase 1, Round 2

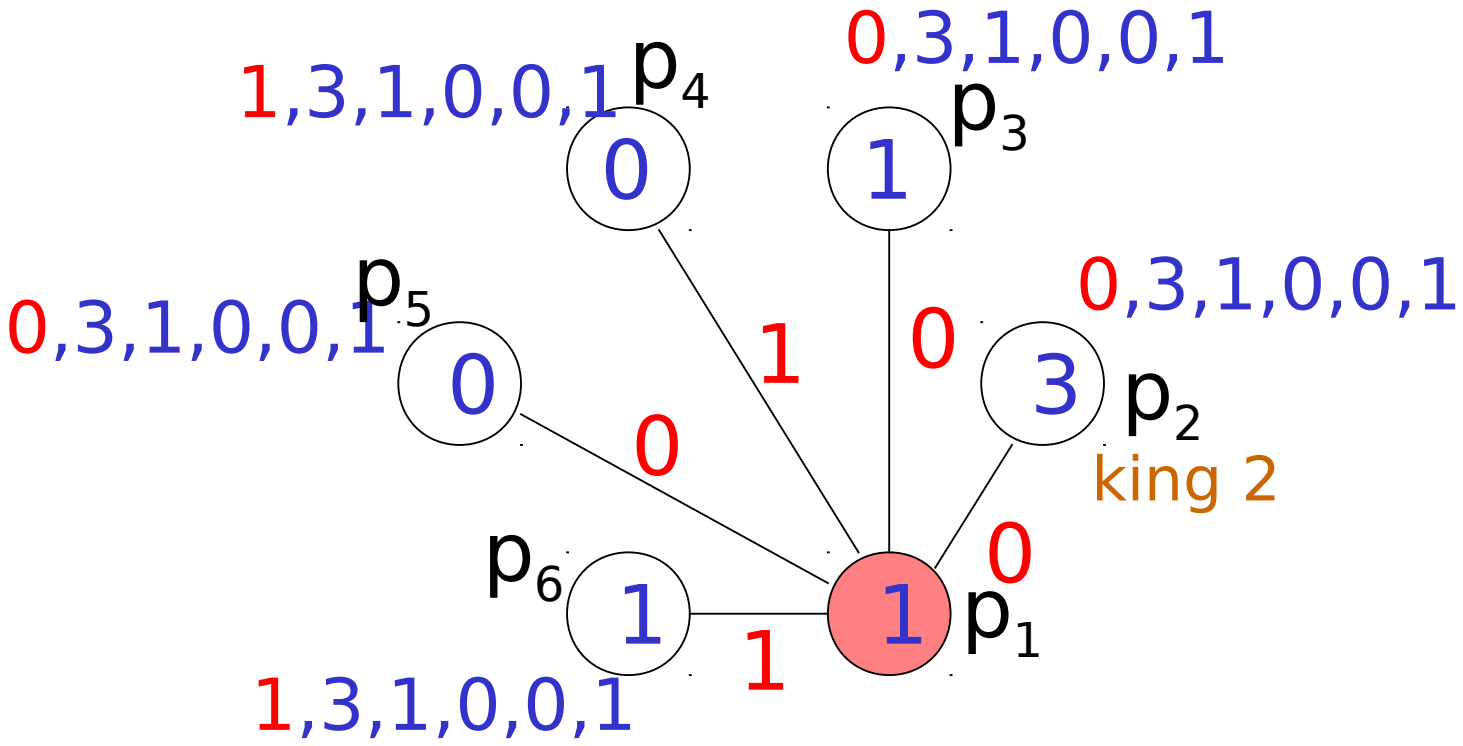


The king broadcasts

Everybody chooses the king's value

Fonte: Prof. Guido Proietti

Phase 2, Round 1

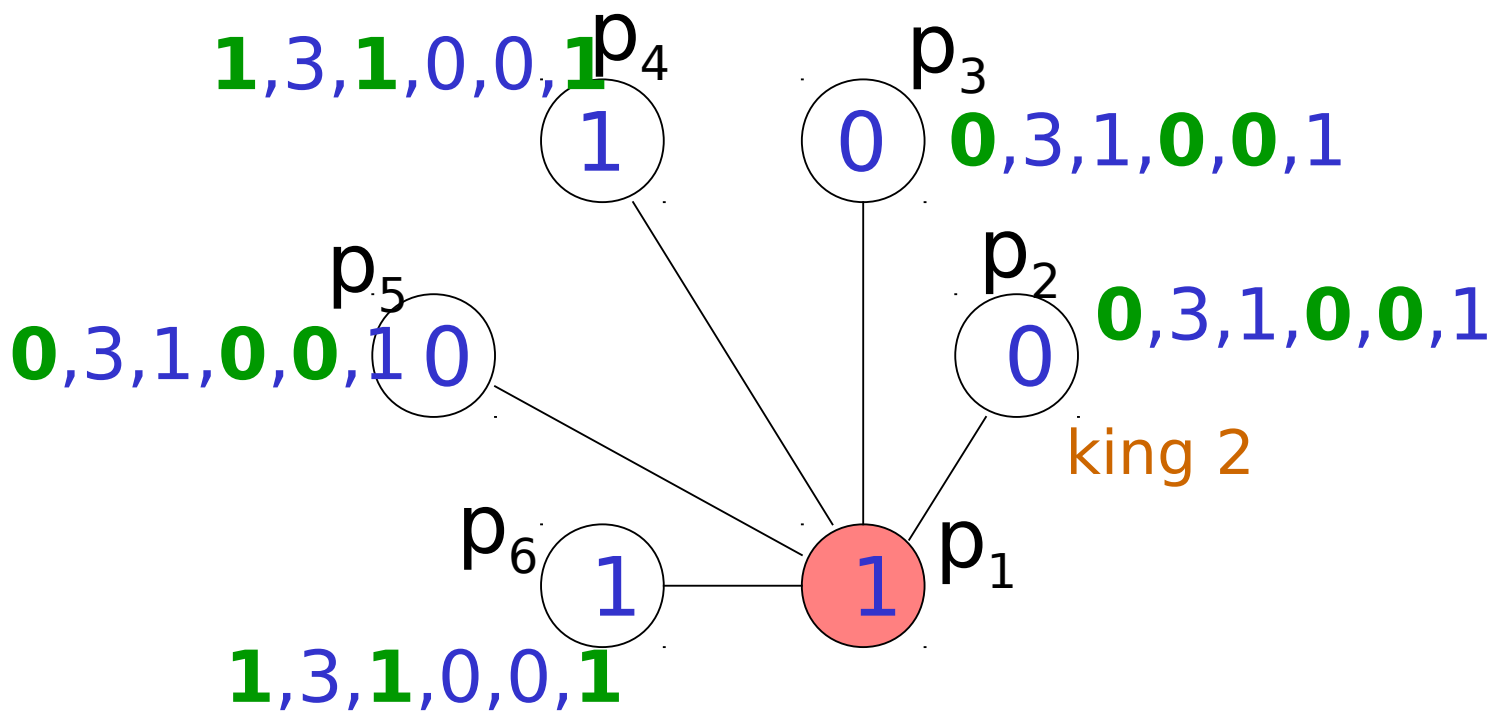


Everybody broadcasts

Fonte: Prof. Guido Proietti

Phase 2, Round 1

Choose the majority

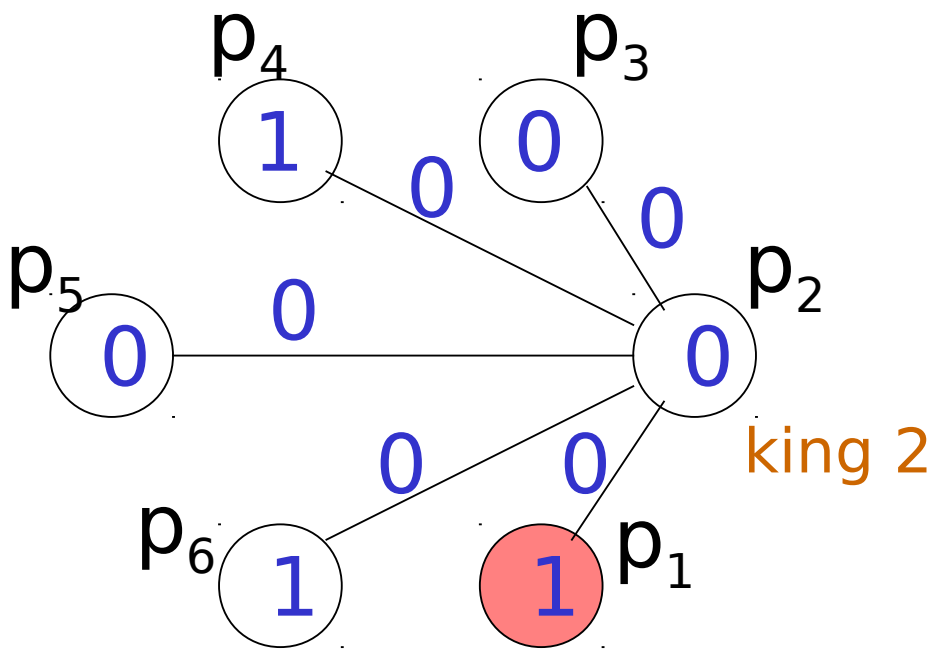


Each majority is equal to $3 < \frac{n}{2} + f + 1 = 5$

⇒ On round 2, everybody will choose the king's val

Fonte: Prof. Guido Proietti

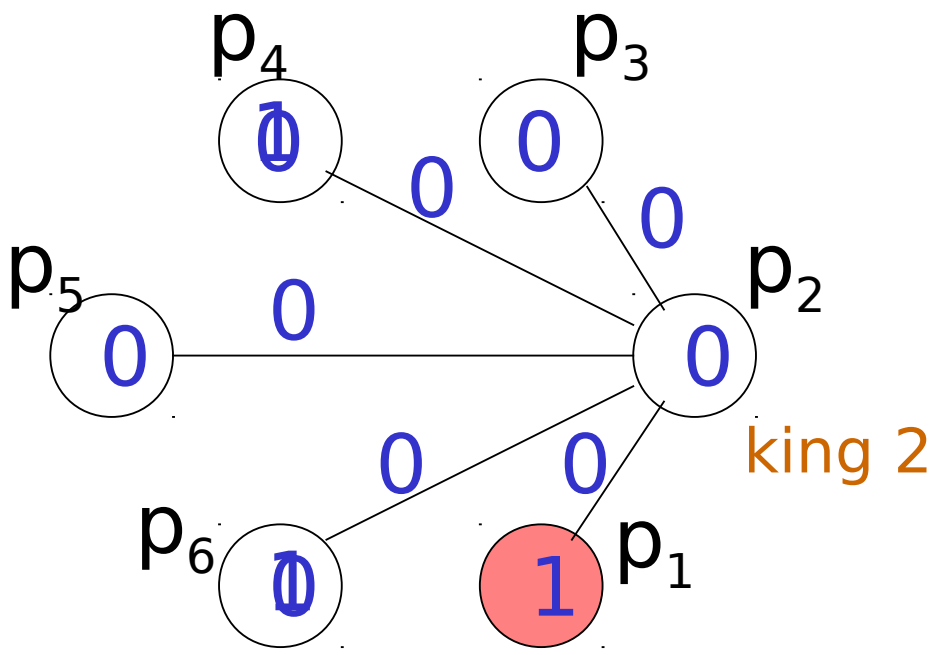
Phase 2, Round 2



The king broadcasts

Fonte: Prof. Guido Proietti

Phase 2, Round 2



The king broadcasts

Everybody chooses the king's value

Final decision and agreement!

Fonte: Prof. Guido Proietti

Correctness of the King algorithm

Lemma 1: At the end of a phase φ where the king is non-faulty, every non-faulty processor decides the same value

Proof: Consider the end of round 1 of phase

There are two cases:

Case 1: some node has chosen its preferred value with strong majority ($¿ \frac{n}{2} + f + 1$ votes)

Case 2: No node has chosen its preferred value with strong majority

Fonte: Prof. Guido Proietti

Case 1: suppose node i has chosen its preferred a value with strong majority ($¿ \frac{n}{2}+f+1$ votes)

⇒ This implies that at least n/2+1 non-faulty nodes must have broadcasted *a* at start of round 1 of phase φ, and then at the end of that round, every other non-faulty node must have preferred value *a* (including the king)

Fonte: Prof. Guido Proietti

At end of round 2:

If a node keeps its own value:

then decides a

If a node selects the value of the non-faulty king:

then it decides a ,

since the king has decided a

Therefore: Every non-faulty node decides a

Fonte: Prof. Guido Proietti

Case 2: No node has chosen its preferred value with strong majority (p_5 votes)

Every non-faulty node will adopt the value of the king, thus all decide on the same value

END of PROOF

Fonte: Prof. Guido Proietti

Lemma 2: Let a be a common value decided by non-faulty processors at the end of a phase φ. Then, a will be preferred until the end.

Proof: After φ, a will always be preferred with strong majority (i.e., >n/2+f), since there are at least n-f non-faulty processors and:

$$f < \frac{n}{4} \Rightarrow 2f < \frac{n}{2} \Rightarrow 2f < n - \frac{n}{2} \Rightarrow n - 2f > \frac{n}{2} \Rightarrow n - f > \frac{n}{2} + f$$

Thus, until the end of phase f+1, every non-faulty processor decides a. **QED**

Fonte: Prof. Guido Proietti

Agreement in the King algorithm

Follows from Lemma 1 and 2, observing that since there are f+1 phases and at most f failures, there is al least one phase in which the king is non-faulty (and thus from Lemma 1 at the end of that phase all non-faulty processors decide the same, and from Lemma 2 this decision will be maintained until the end).

Fonte: Prof. Guido Proietti

Validity in the King algorithm

Follows from the fact that if all (non-faulty) processors have a as input, then in round 1 of phase 1 each non-faulty processor will receive a with strong majority, since:

$$p_1$$

and so in round 2 of phase 1 this will be the preferred value of non-faulty processors, independently of the king's broadcasted value. From Lemma 2, this will be maintained until the end, and will be exactly the decided output!

QED

Fonte: Prof. Guido Proietti

Performance of King Algorithm

- Number of processors: $n > 4f$
- $2(f+1)$ rounds
- $\Theta(n^2 \bullet f) = O(n^3)$ messages. Indeed, each non-faulty node sends n messages in the first round of each phase, each containing a given preference value, and each non-faulty king sends n-1 messages in the second round of each phase. Notice that we are not considering the fact that a byzantine processor could generate an unbounded number of messages!

Fonte: Prof. Guido Proietti

Byzantine Generals Problem

Anthony Soo Kaim
Ryan Chu
Stephen Wu

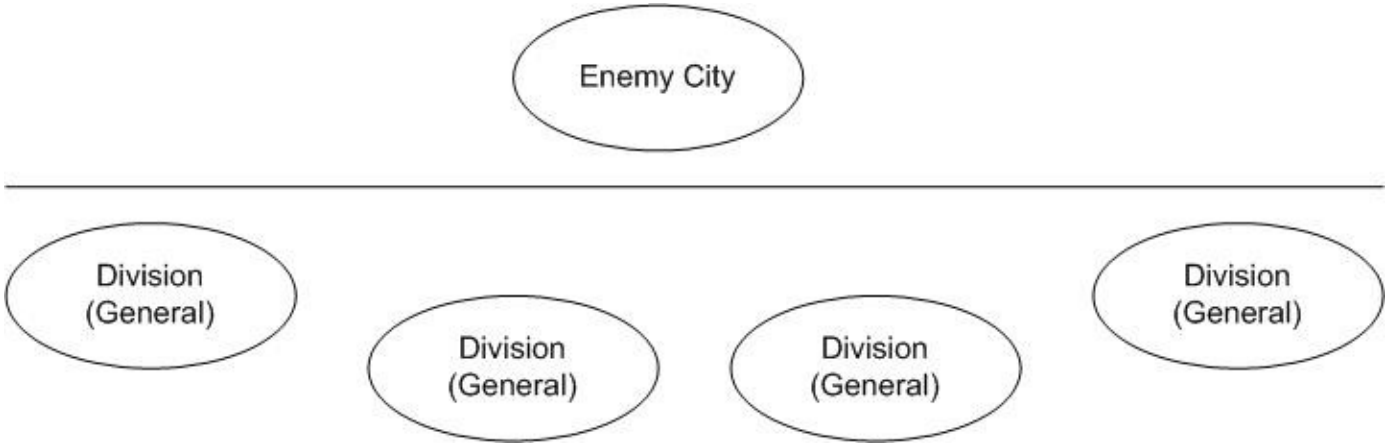
Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Mais uma contribuição de Leslie Lamport et al.

"The Byzantine Generals Problem", by Leslie Lamport, Robert Shostak, Marshall Pease, In ACM Transactions on Programing Languages and Systems, July 1982

> *We know of no area in computer science or mathematics in which informal reasoning is more likely to lead to errors than in the study of this type of algorithm.*

Problem



- Divisions of the Byzantine army camped outside the walls of an enemy city.
- Each division is led by a general.
- Generals decide on a common plan of action.

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Problem – Types of Generals

- There are two types of generals
  1. Loyal Generals
  2. Traitor Generals

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Problem – Conditions

- Two conditions must be met:
  1. All loyal generals decide upon the same plan of action.
  2. A small number of traitors cannot cause the loyal generals to adopt a bad plan.

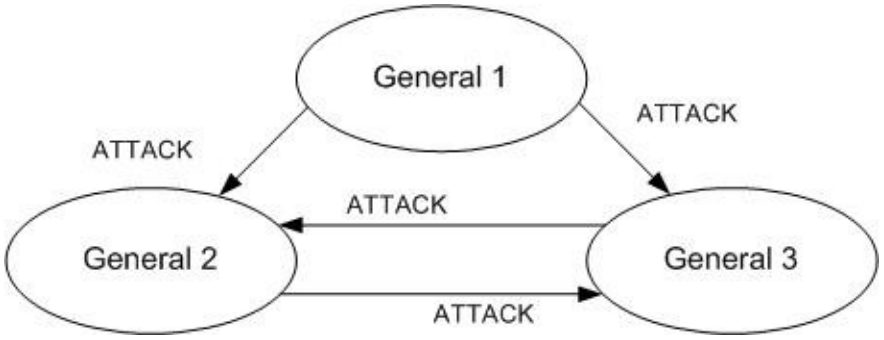
Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Problem – Not a Bad Plan

- A plan that is not bad is defined in the following way:
  - Each general sends his observation to all other generals.
  - Let v(i) be the message communicated by the ith general.
  - The combination of the v(i) for i = 1, …, n messages received determine a plan that is not bad.

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Problem – Example Not a Bad Plan



- General 2 receives ATTACK, ATTACK.
- General 3 receives ATTACK, ATTACK.
  - So Not a Bad Plan is to ATTACK

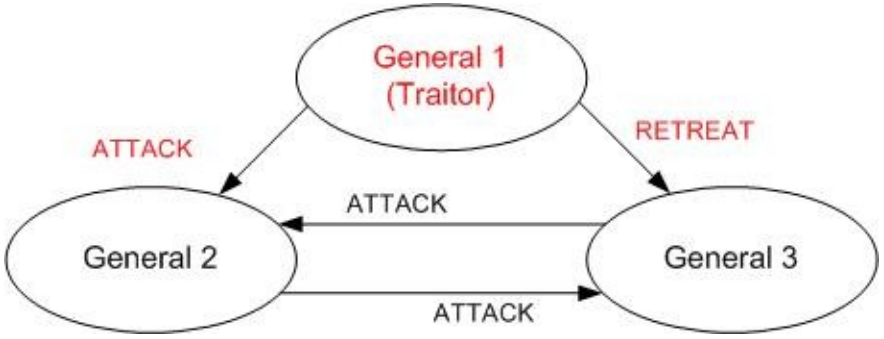
Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Problem – Not a Bad Plan Flaw

- Assumed that every general communicates the same v(i) to every other general.
- A traitor general can send different v(i) messages to different generals.

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Problem – Example Flaw



- General 2 receives ATTACK, ATTACK.
- General 3 receives RETREAT, ATTACK.
  - Is Not a Bad Plan to ATTACK or RETREAT?

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Problem – New Conditions

- The new conditions are:
  - Any two loyal generals use the same value of v(i).
  - If the i^{th} general is loyal, then the value that he sends must be used by every loyal general as the value of v(i).

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Byzantine Generals Problem

- A commander general giving orders to his lieutenant generals.
- ***Byzantine Generals Problem*** – A commanding general must send an order to his n-1 lieutenant generals such that:
  - **IC1**. All loyal lieutenants obey the same order.
  - **IC2**. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.
  - These are called the ***interactive consistency*** conditions.

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Key Step: Agree on inputs

Generals communicate v(i) values to one another:

1) Every loyal general must obtain same v(1)..v(n)

1') Any two loyal generals use same value of v(i)

- Traitor i will try to loyal generals into using different v(i)'s

2) If ith general is loyal, then the value he sends must be used by every other general as v(i)

Problem: How can each general send his value to n-1 others?

A commanding general must send an order to his n-1 lieutenants such that:

IC1) All loyal lieutenants obey same order

IC2) If commanding general is loyal, every loyal lieutenant obeys the order he sends

Interactive Consistency conditions

Fonte: Andrea C. Arpaci-Dusseau

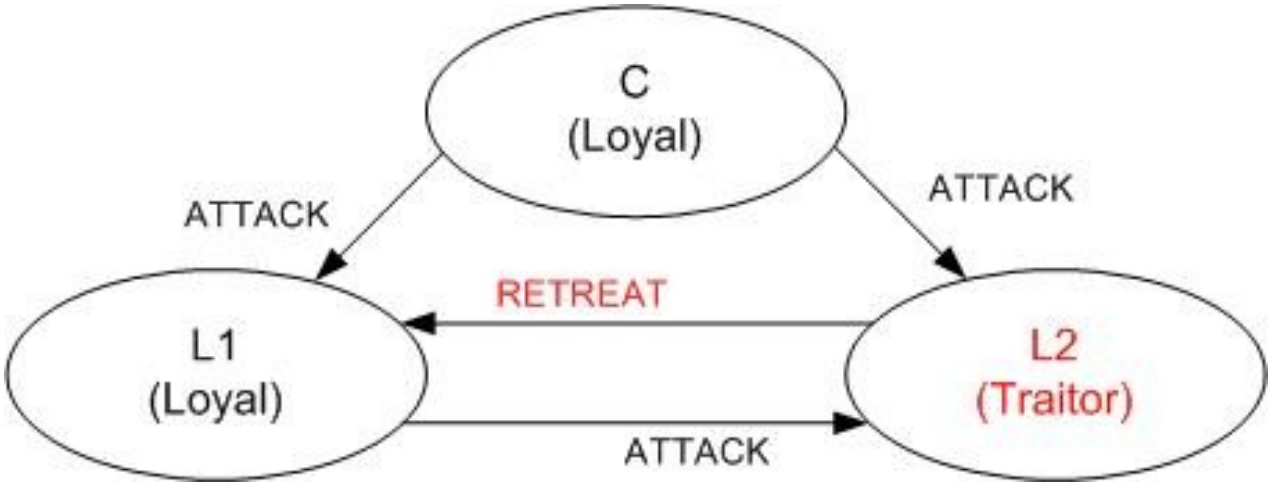
Impossibility Results

- When will the Byzantine Generals Problem fail?

- The problem will fail if 1/3 or more of the generals are traitors.

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

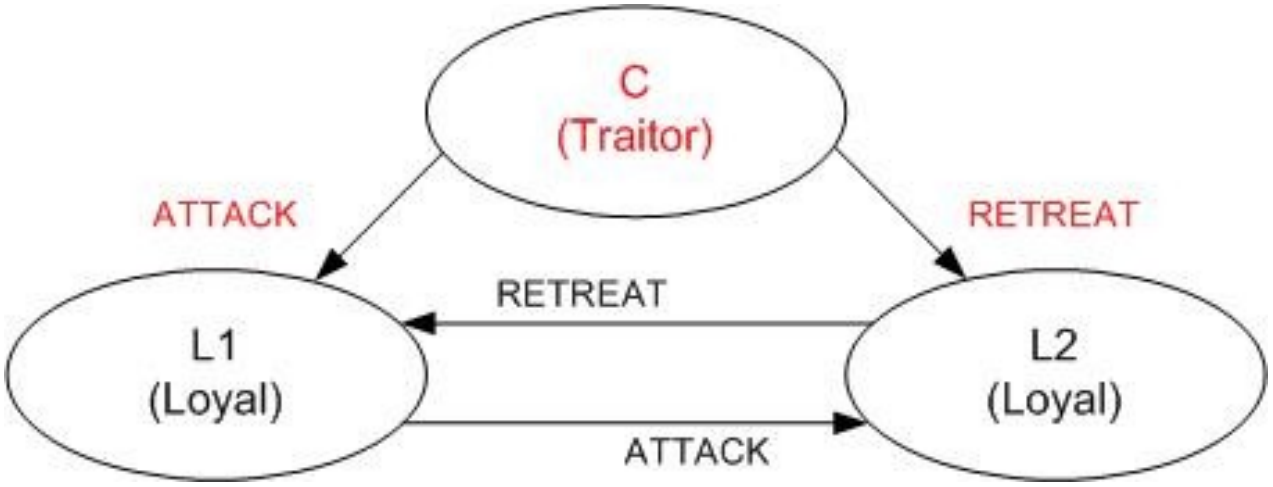
Impossibility Results – Example



- L1 received the commands ATTACK, RETREAT
- L1 doesn’t know which general is a traitor.

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Impossibility Results – Example 2



- L1 again received the commands ATTACK, RETREAT
- L1 doesn’t know which general is a traitor.

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Impossibility Results Generalization

- No solution when:
  - Fewer than $3m + 1$ generals;
    - m = number of traitor generals

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

A Solution with Oral Messages

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Solution with Oral Messages

- Assumptions:
  - **A1**: Every message that is sent is delivered correctly.
  - **A2**: The receiver of a message knows who sent it.
  - **A3**: The absence of a message can be detected.

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Solution with OM – Definition

- majority(v_1, …, v_{n-1})
  - If the majority of the values v_i equal v, then majority(v_1, …, v_{n-1}) is v.
  - If a majority doesn't exist, then the function evaluates to RETREAT.

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Solution with OM – Algorithm

Case where m = 0 (No traitors)

Algorithm OM(0)

1. The commander sends his value to every lieutenant.
2. Each lieutenant uses the value he receives from the commander, or uses the value RETREAT if he receives no value.

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

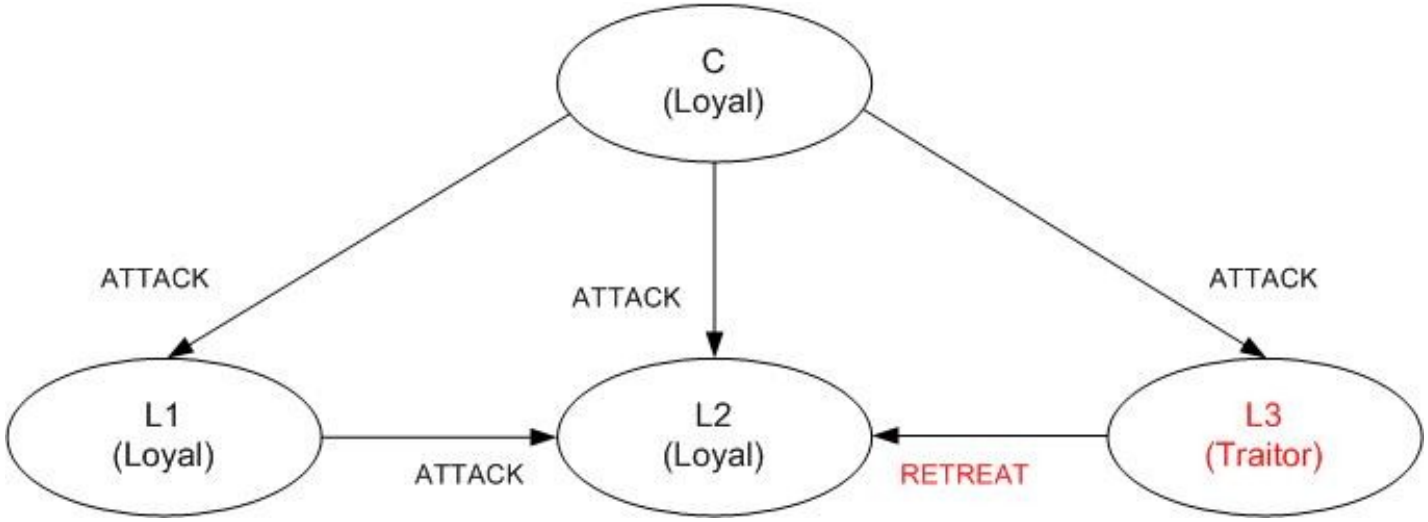
Solution with OM – Algorithm

Algorithm OM(m), $m > 0$

1. The commander sends his value to every lieutenant.
2. For each i, let v_i be the value lieutenant i receives from the commander, or else be RETREAT if he receives no value. Lieutenant i acts as the commander in Algorithm OM(m-1) to send the value v_i to each of the $n - 2$ other lieutenants.
3. For each i, and each $j \neq i$, let v_j be the value lieutenant i received from lieutenant j in step 2 (using OM(m-1)), or else RETREAT if he received no value. Lieutenant i uses the value majority(v_1, …, v_{n-1}).

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

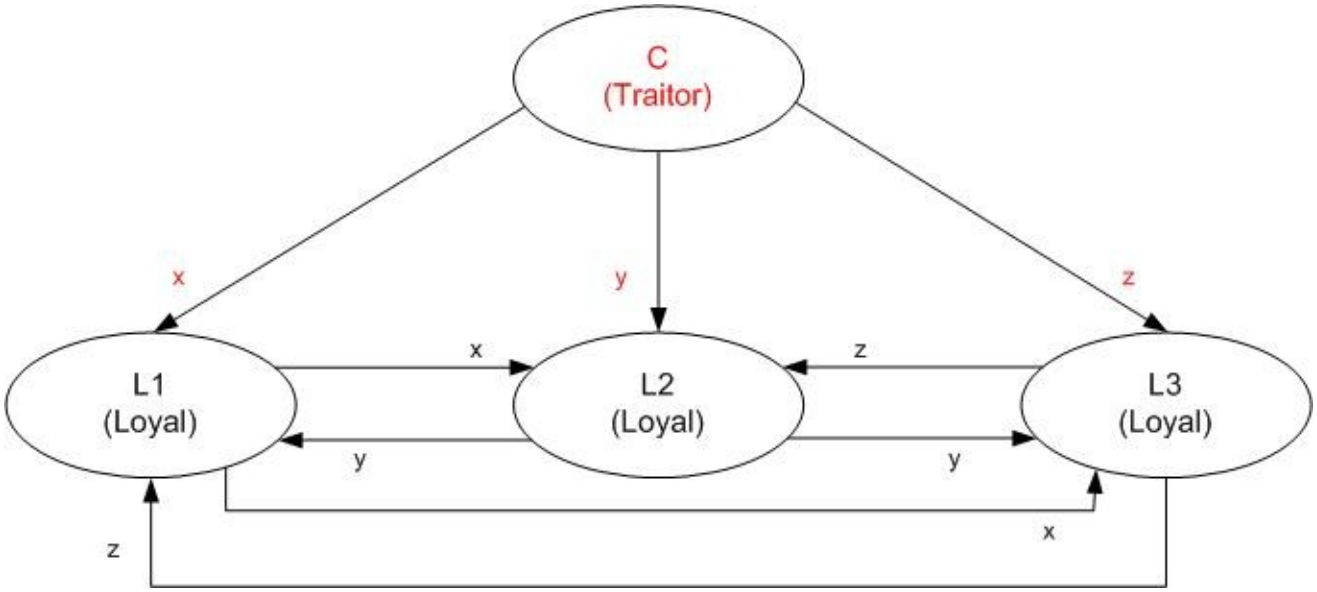
Solution with OM – Example



- n=4 generals; m=1 traitors
- L2 calculates majority(ATTACK, ATTACK, RETREAT) = ATTACK

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Solution with OM – Example

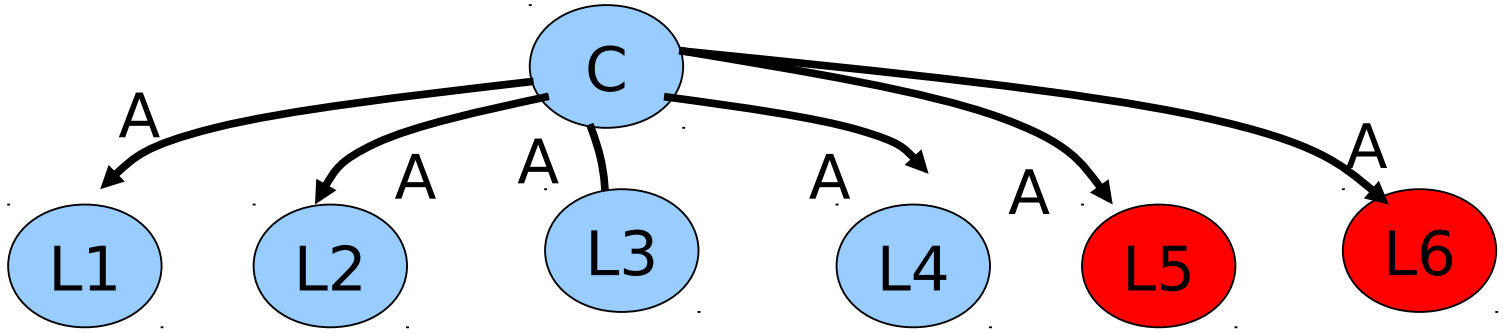


- n=4 generals; m=1 traitors
- L1, L2, L3 calculate majority(x, y, z)

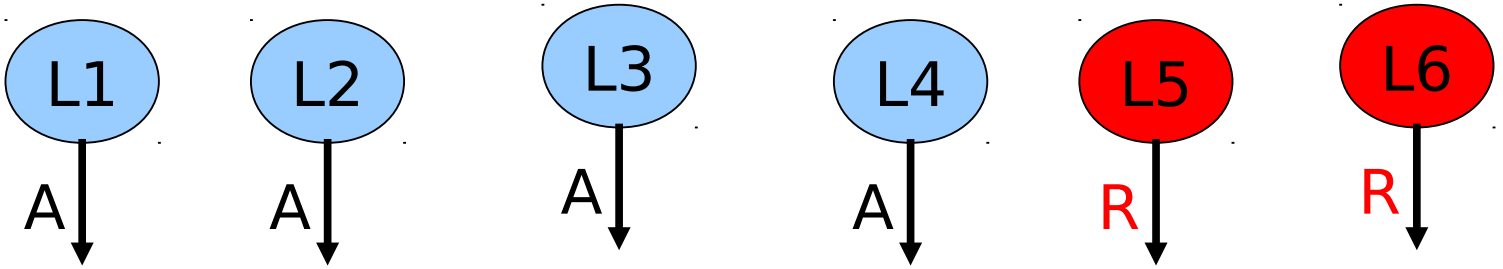
Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Bigger Example: Bad Lieutenants

Scenario: m=2, n=7, traitors=L5, L6



Messages?

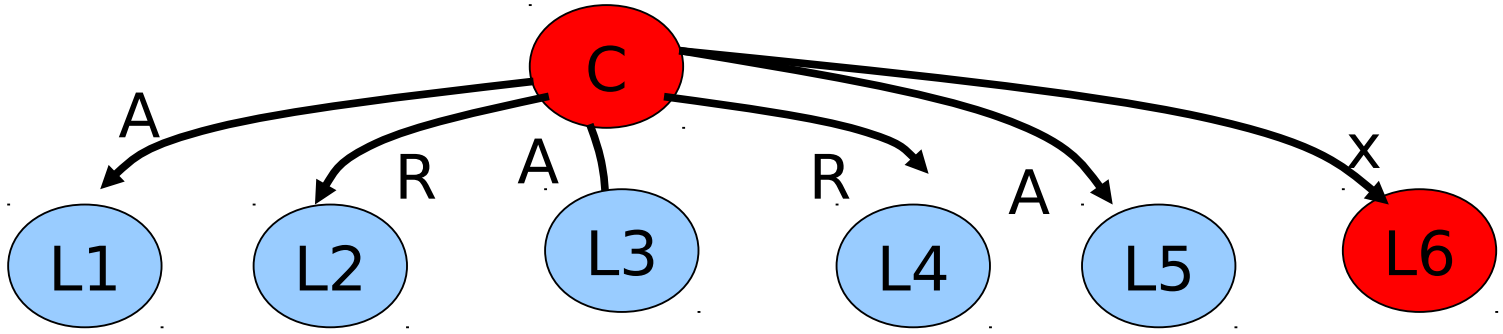


Decision??? m(A,A,A,A,R,R) ==> All loyal lieutenants atta

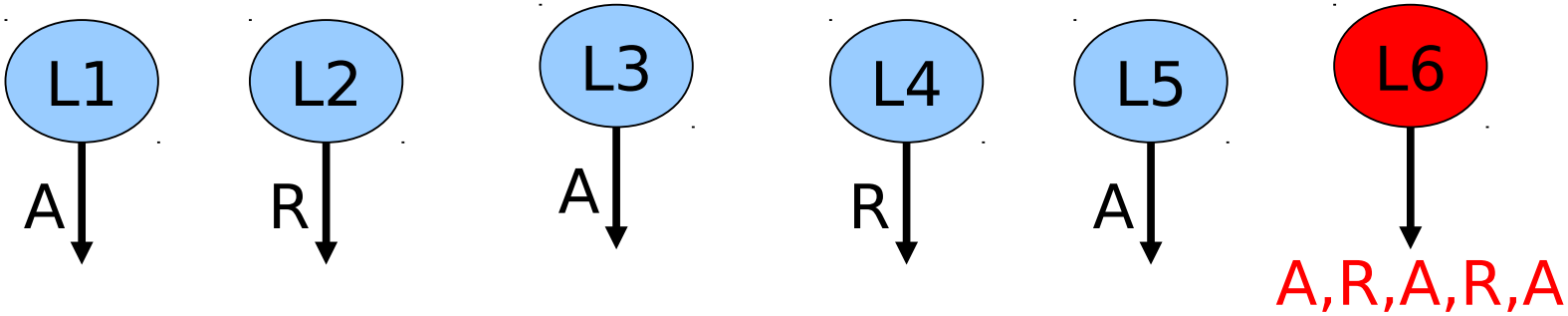
Fonte: Andrea C. Arpaci-Dusseau

Bigger Example: Bad Commander+

Scenario: m=2, n=7, traitors=C, L6



Messages?



Decision???

Fonte: Andrea C. Arpaci-Dusseau

Decision with Bad Commander+

L1: m(A,R,A,R,A,A) ==> Attack

L2: m(R,R,A,R,A,R) ==> Retreat

L3: m(A,R,A,R,A,A) ==> Attack

L4: m(R,R,A,R,A,R) ==> Retreat

L5: m(A,R,A,R,A,A) ==> Attack

Problem: All loyal lieutenants do NOT choose same action

Fonte: Andrea C. Arpaci-Dusseau

Next Step of Algorithm

Verify that lieutenants tell each other the same thing

- Requires rounds = m+1
- OM(0): Msg from Lieut i of form: “L0 said v0, L1 said v1, etc...”

What messages does L1 receive in this example?

- OM(2): A
- OM(1): 2R, 3A, 4R, 5A, 6A
- OM(0): 2{ 3A, 4R, 5A, 6R}
- 3{2R, 4R, 5A, 6A}
- 4{2R, 3A, 5A, 6R}
- 5{2R, 3A, 4R, 6A}
- 6{ total confusion }

All see same messages in OM(0) from L1,2,3,4, and 5

m(A,R,A,R,A,-) ==> All attack

Fonte: Andrea C. Arpaci-Dusseau

Proof of algorithm OM(m)

- **Lemma 1**. For any m and k, OM(m) satisfies IC2 if there are more than 2k + m generals and at most k traitors

- **Proof** by induction on m:
  - Step 1: loyal commander sends v to all n – 1 lieutenants.
  - Step 2: each loyal lieutenant applies OM(m – 1) with n – 1 generals.
  - By hypothesis, we have $n - 1 > 2k + (m - 1) \geq 2k$.
  - k traitors at most, so a majority of the n – 1 lieutenants are loyal. Each loyal lieutenant has $v_i = v$ for a majority of the n – 1 values, and therefore majority(…) = v

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Proof of algorithm OM(m)

- **Theorem 1**. For any m, OM(m) satisfies conditions IC1 and IC2 if there are more than 3m generals and at most m traitors

- **Proof** by induction on m:
  - For no traitors, OM(0) satisfies IC1 and IC2. Assume validity for OM(m – 1) and prove OM(m) for m > 0.
  - Loyal commander: k = m from Lemma 1, so OM(m) satisfies IC2.
  - Traitorous commander: must also show IC1 is met:
  - m – 1 lieutenants will be traitors. There are more than 3m generals and 3m – 1 lieutenants, and 3m – 1 > 3(m – 1), so OM(m – 1) satisfies IC1

Fonte: Anthony Soo Kaim, Rayn Chu e Stephen Wu

Exponential Tree Algorithm

- This algorithm uses
  - $n=3f+1$ processors (optimal)
  - $f+1$ rounds (optimal)
  - exponential number of messages (sub-optimal, the King algorithm was using only $O(n^3)$ msgs)
- Each processor keeps a tree data structure in its local state
- Topologically, the tree has height $f+1$, and all the leaves are at the same level
- Values are filled top-down in the tree during the $f+1$ rounds; more precisely, during round i, level i of the tree is filled
- At the end of round $f+1$, the values in the tree are used to compute bottom-up the decision.

Fonte: Prof. Guido Proietti

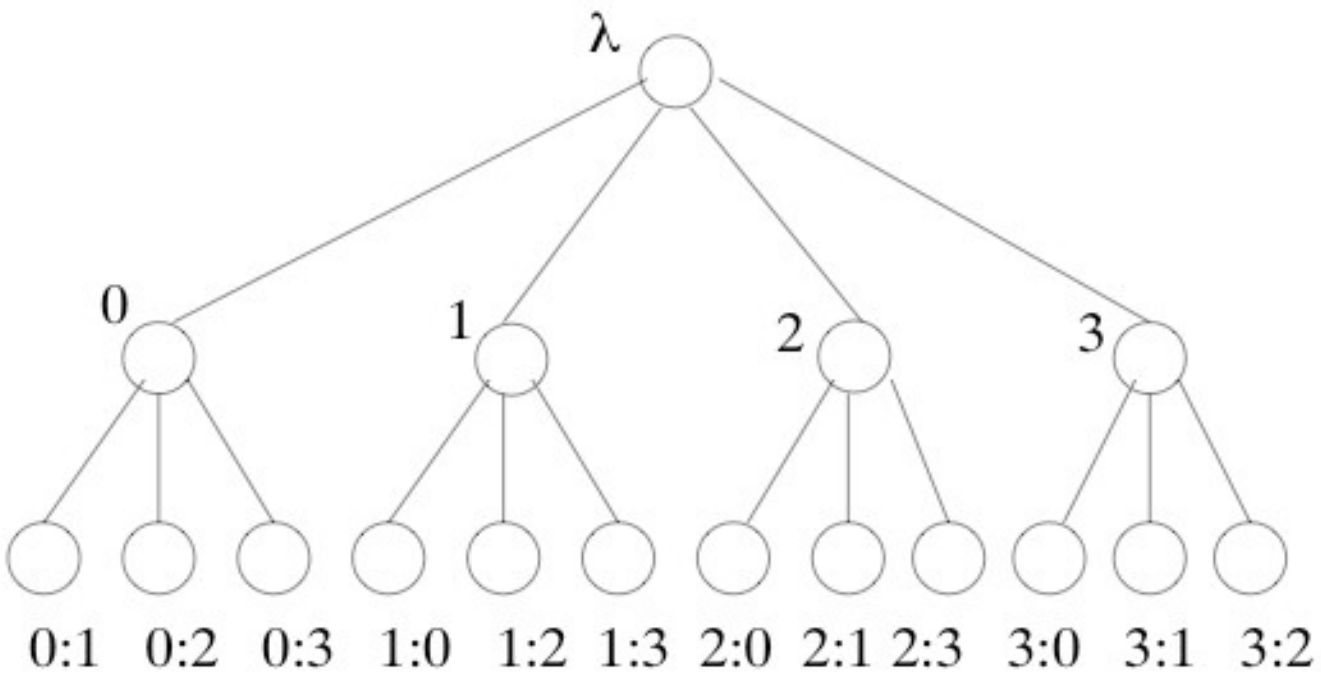
Local Tree Data Structure

- **Assumption**: Similarly to the King algorithm, processors have (distinct) ids (now in {0,1,...,n-1}), and we denote by p_i the processor with id i; this is common knowledge, i.e., processors cannot cheat about their ids;
- Each tree node is labeled with a sequence of unique processor ids in 0,1,...,n-1:
  - Root's label is empty sequence λ (the root has level 0 and height f+1);
  - Root has n children, labeled 0 through n-1
  - Child node of the root (level 1) labeled i has n-1 children, labeled i:0 through i:n-1 (skipping i:i);
  - Node at level d>1 labeled $i_1:i_2:\ldots:i_d$ has n-d children, labeled $i_1:i_2:\ldots:i_d:0$ through $i_1:i_2:\ldots:i_d:n-1$ (skipping any index $i_1,i_2,\ldots,i_d$);
  - Nodes at level f+1 are leaves and have height 0.

Fonte: Prof. Guido Proietti

Example of Local Tree

The tree when n=4 and f=1:



Fonte: Prof. Guido Proietti

Filling-in the Tree Nodes

- **Round 1:**
  - Initially store your input in the root (level 0)
  - send level 0 of your tree (i.e., your input) to all (including yourself)
  - store value x received from p_j, j=0,…,n-1 in tree node labeled j (level 1); use a default value "*" (known to all!) if necessary (i.e., in case a value is not received or it is unfeasible)
  - node labeled j in the tree associated with p_i now contains what "p_j told to p_i" about its input;
- **Round 2:**
  - send level 1 of your tree to all, including yourself (this means, send n messages to each processor)
  - let $\{x_0,\ldots,x_{n-1}\}$ be the set of values that p_i receives from p_j; then, p_i discards x_j, and stores each remaining x_k in level-2 node labeled k:j (and use default value "*" if necessary)
  - node k:j in the tree associated with p_i now contains "p_j told to p_i that "p_k told to p_j that its input was x_k""

Fonte: Prof. Guido Proietti

Filling-in the Tree Nodes (2)

- Round d>2:
  - send level d-1 of your tree to all, including yourself (this means, send n(n-1)…(n-(d-2)) messages to each processor, one for each node on level d-1)
  - Let x be the value received from p_j for node of level d-1 labeled i_1:i_2:…:i_{d-1}, with $i_1,i_2,\ldots,i_{d-1} \neq j$; then, store x in tree node labeled i_1:i_2:…:i_{d-1}:j (level d); use default value "*" if necessary
- Continue for f+1 rounds

Fonte: Prof. Guido Proietti

Calculating the Decision

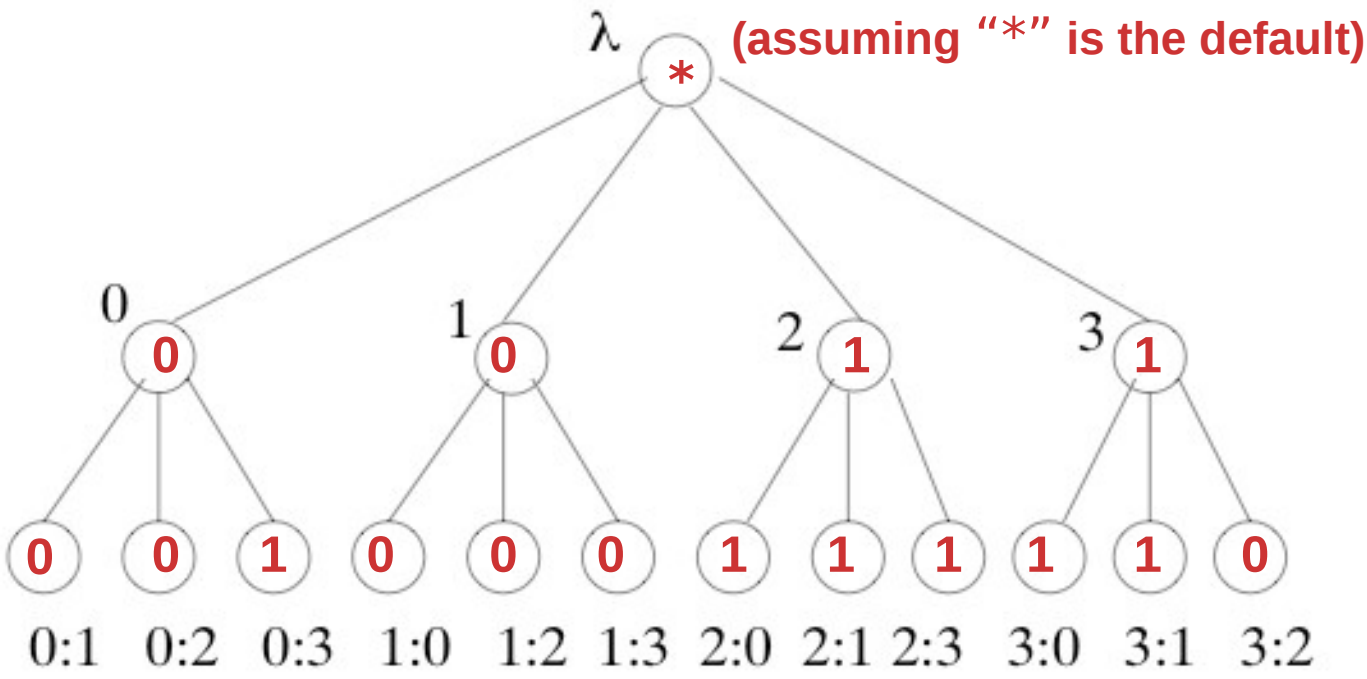
- In round f+1, each processor uses the values in its tree to compute its decision.
- Recursively compute the "resolved" value for the root of the tree, **`resolve`**(λ), based on the "resolved" values for the other tree nodes:

`resolve`(π) = {
- value in tree node labeled π if it is a leaf
- majority{**`resolve`**(π') : π' is a child of π}
- otherwise (use default "*" if tied)

Fonte: Prof. Guido Proietti

Example of Resolving Values

The tree when n=4 and f=1:



Fonte: Prof. Guido Proietti

Resolved Values are consistent

Lemma 1: If p_i and p_j are non-faulty, then p_i's resolved value for tree node labeled π=π'j is equal to what p_j stores in its node π‘ during the filling-up of the tree (and so the value stored and resolved in π by p_i is the same, i.e., it is consistent). (Notice this lemma does not hold for the root)

Proof: By induction on the height of the tree node.

- **Basis**: height=0 (leaf level). Then, p_i stores in node π what p_j sends to it for π’ in the last round. By definition, this is the resolved value by p_i for π

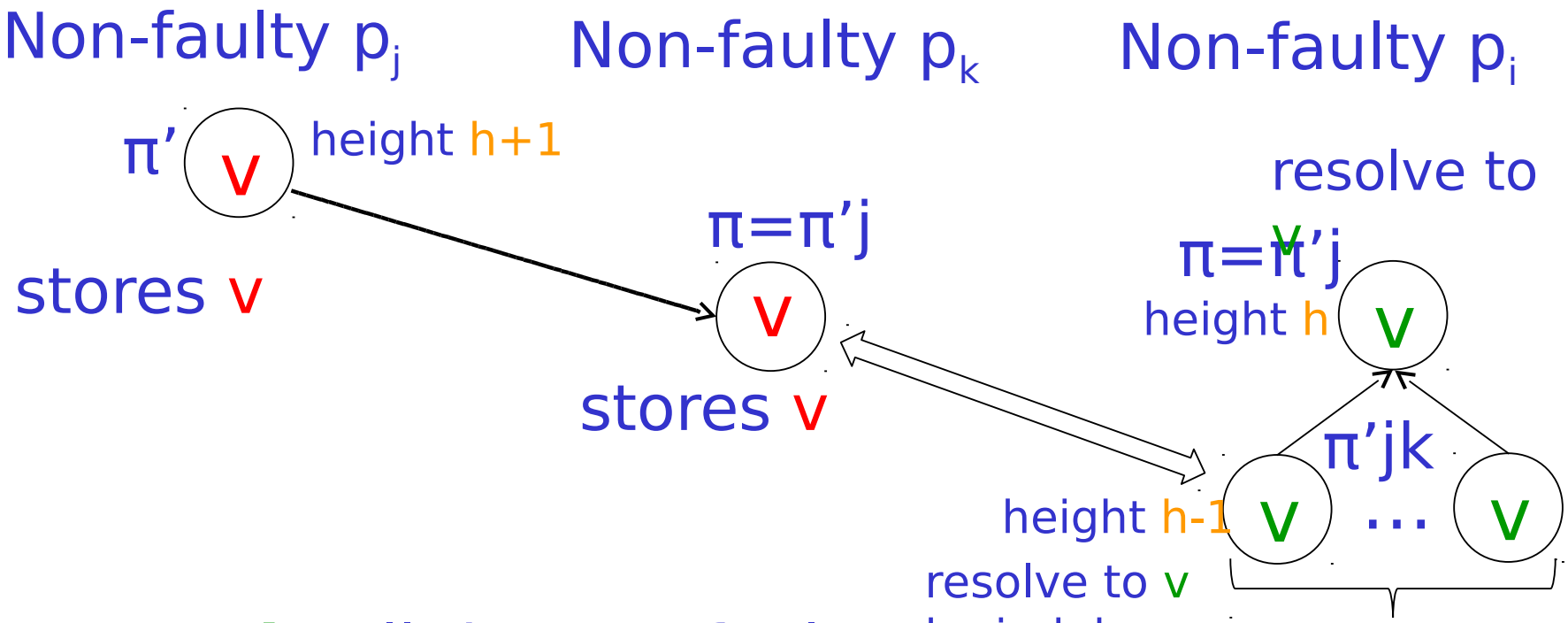
Fonte: Prof. Guido Proietti

- **Induction**: π is not a leaf, i.e., has height h>0;
  - By definition, π has at least n-f children, and since n>3f, this implies n-f>2f, i.e., it has a majority of non-faulty children (i.e., whose last digit of the label corresponds to a non-faulty processor)
  - Let πk=π'jk be a child of π of height h-1 such that p_k is non-faulty.
  - Since p_j is non-faulty, it correctly reports a value v stored in its π' node; thus, p_k stores it in its π=π'j node.
  - By induction, p_i's resolved value for πk equals the value v that p_k stored in its π node.
  - So, all of π's non-faulty children resolve to v in p_i's tree, and thus π resolves to v in p_i's tree.

END of PROOF

Fonte: Prof. Guido Proietti

Inductive step by a picture



Remark: all the non-faulty processors will resolve the very same value in π=π'j, namely v, and the node is said to be common

Fonte: Prof. Guido Proietti

Validity

- Suppose all inputs of (non-faulty) processors are v
- Non-faulty processor p_i decides **`resolve`**(λ), which is the majority among **`resolve`**(j), $0 \leq j \leq n-1$, based on p_i's tree.
- Since by Lemma 1 resolved values are consistent, if p_j is non-faulty, then p_i's resolved value for tree node labeled j, i.e., **`resolve`**(j), is equal to what p_j stores in its root, namely p_j's input value, i.e., v.
- Since there are a majority of non-faulty processors, p_i decides v.

Fonte: Prof. Guido Proietti

Agreement: Common Nodes and Frontiers

- Recall that a tree node π is **common** if all non-faulty processors compute the same value of `resolve`(π).

$\Rightarrow$To prove agreement, we have to show that the root is common

- A tree node π has a **common frontier** if every path from π to a leaf contains at least a common node.

Fonte: Prof. Guido Proietti

Lemma 2: If π has a common frontier, then π is common.

Proof: By induction on height of π:

- **Basis** (π is a leaf): then, since the only path from π to a leaf consists solely of π, the common node of such a path can only be π, and so π is common;
- **Induction** (π is not a leaf): By contradiction, assume π is not common; then:
  - Every child π' of π has a common frontier (this is not true, in general, if π would be common);
  - By inductive hypothesis, every child π' of π is common;
  - Then, all non-faulty processors resolve the same value for every child π' of π, and thus all non-faulty processors resolve the same value for π, i.e., π is common

END of PROOF

Fonte: Prof. Guido Proietti

Agreement: the root has a common frontier

- There are f+2 nodes on any root-leaf path
- The label of each non-root node on a root-leaf path ends in a distinct processor index: $i_1,i_2,\ldots,i_{f+1}$
- Since there are at most f faulty processors, at least one such nodes has a label ending with a non-faulty processor index
- This node, say $i_1:i_2:,\ldots,i_{k-1}:i_k$, by Lemma 1 is **common** (more precisely, in all the trees associated with non-faulty processors, the resolved value in $i_1:i_2:,\ldots,i_{k-1}:i_k$ equals the value stored by the non-faulty processor p_{i_k} in node $i_1:i_2:,\ldots,:i_{k-1}$)

⇒ Thus, the root has a common frontier, since on any root-leaf path there is at least a common node, and so the root is common (by previous lemma)

⇒ Therefore, agreement is guaranteed!

Fonte: Prof. Guido Proietti

Complexity

- Exponential tree algorithm uses $f+1$ rounds, and $n=3f+1$ processors are enough to guarantee correctness
- Exponential number of messages:
  - In round 1, each (non-faulty) processor sends n messages $\Rightarrow O(n^2)$ total messages
  - In round $d \geq 2$, each of the $O(n)$ (non-faulty) processors broadcasts to all (i.e., n processors) the level $d-1$ of its local tree, which contains $n(n-1)(n-2)\ldots(n-(d-2))$ nodes $\Rightarrow$ this means, for round d, a total of

    $O(n \cdot n \cdot n(n-1)(n-2)\ldots(n-(d-2)))=O(n^{d+1})$ messages
  - This means a total of $O(n^2)+O(n^3)+\ldots+ O(n^{f+1})= O(n^{f+1})$ messages, and since $f=O(n)$, this number is exponential in n if f is more than a constant relative to n

Fonte: Prof. Guido Proietti

Exercise 1: Show an execution with n=4 processors and f=1 for which the King algorithm fails.

Exercise 2: Show an execution with n=3 processors and f=1 for which the exp-tree algorithm fails.

Fonte: Prof. Guido Proietti