Reaching Agreement in the Presence of Faults

M. Pease, R. Shostak, and L. Lamport

Journal of the ACM, Vol. 27, No. 2, April 1980, pp. 228-234.

Problem

Obtaining interactive consistency is quite fundamental for the design of fault-tolerant distributed systems.

In SIFT, the problem arises in three parts:

- synchronization of clocks,
- stabilization of input from sensors, and
- agreement on results of diagnostic tests.

Interactive Consistency

Consider a set of *n* processors, of which it is known that no more than *m* are faulty. However, it is not known which processors are faulty.

Communication

- Fully connected network
- Reliable links of negligible delay

Interactive Consistency

Each processor p has some private value of information v_p

For given m and n, an algorithm should allow each nonfaulty processor p to compute a vector of values with an element for each of the n processors, such that

1. the nonfaulty processors compute exactly the same vector;
2. the element of this vector corresponding to a given nonfaulty processor is the private value of that processor.

Interactive Consistency

Once interactive consistency has been achieved, each nonfaulty processor can apply an averaging or filtering function to the IC vector.

Since each nonfaulty processor applies this function to the same vector of values, an exact agreement is necessarily reached.

Formal Demonstrations

Algorithms can be devised to guarantee interactive consistency iff $n \geq 3m+1$.

Interactive consistency can be assured for arbitrary $n \geq m \geq 0$ using message authentication.

The Single-Fault Case

The procedure consists of an exchange of messages, followed by the computation of the IC vector on the basis of the results of the exchange.

Two rounds of information exchange are required.

1. The processors exchange their private values.
2. They exchange the results obtained in the first round.

$m=1, n=4$

The Single-Fault Case

The faulty processor (if there is one) may "lie," of course, or refuse to send messages.

If a nonfaulty processor *p* fails to receive a message it expects from some other processor, *p* simply chooses a value at random and acts as if that value had been sent.

The Single-Fault Case

Each nonfaulty processor *p* records its private value v_p for the element of the interactive consistency vector corresponding to *p* itself.

The element corresponding to every other processor *q* is obtained by examining the three received reports of *q's* value (one obtained directly from *q* in the first round, and the others from the remaining two processors in the second round). If at least two of the reports agree, the majority value is used. Otherwise, a default value is used (NIL).

Proof of Existance

The results presented in this paper demonstrate existence of solutions.

The construction of simpler and efficient algorithms is a topic for future research.

The Byzantine Generals Problem

L. Lamport, R. Shostak, and M. Pease

ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982, pp. 382-401.

Solvable if more than two-thirds of the generals are loyal

Abstract problem

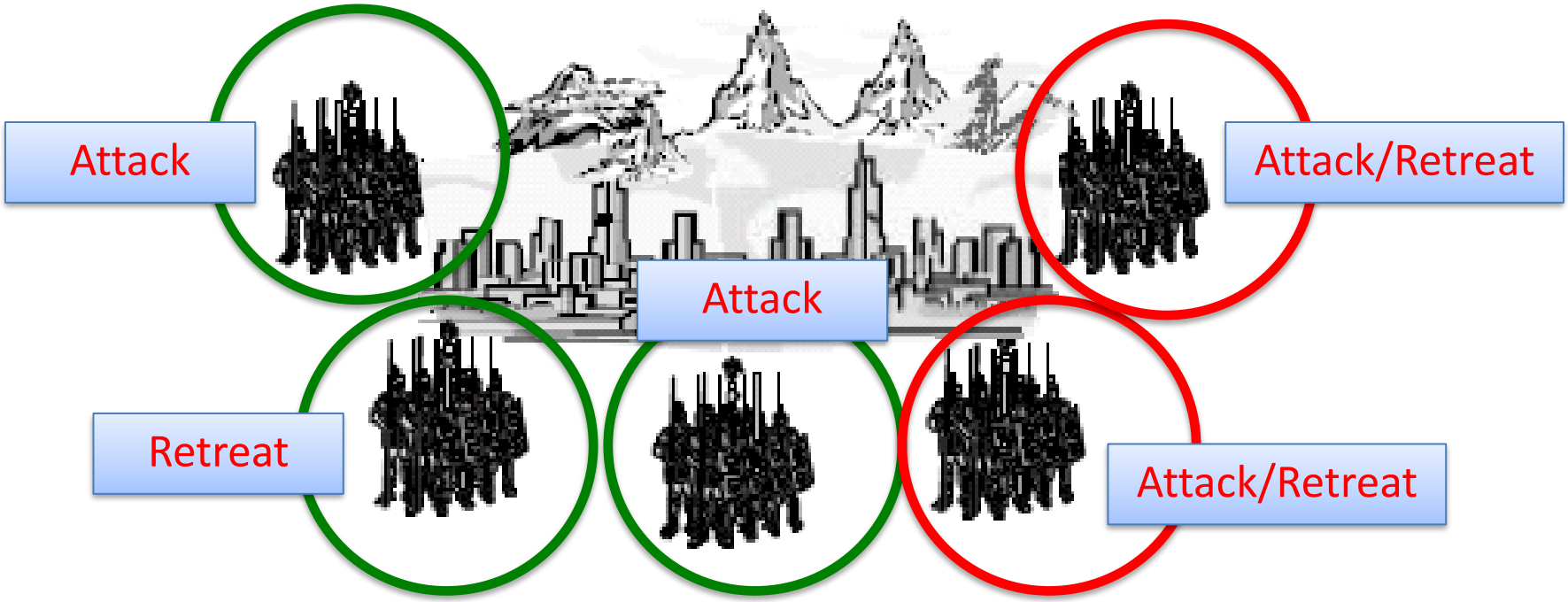
n divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general.

The generals can communicate with one another only by messenger.

After observing the enemy, they must decide upon a common plan of action.

m of the generals may be traitors, trying to prevent the loyal generals from reaching agreement.

Byzantine Generals



Abstract problem

The generals must have an algorithm to guarantee that

A. All loyal generals decide upon the same plan of action
B. A small number of traitors cannot cause the loyal generals to adopt a bad plan

General - processor
Traitor - faulty processor

Bizantine Generals

Each general observes the enemy and communicates his observations to the others.

Let v(i) be the information communicated by the ith general.

Byzantine Generals

For condition A to be satisfied:

1. Every loyal general must obtain the same information v(1), v(2), …, v(n).
2. If the ith general is loyal, then the value that he sends must be used by every loyal general as the value of v(i).

We can rewrite condition 1 as:

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

Reduction of the Problem

We can restrict ourselves to the problem of how a single general sends his value to the others.

Byzantine Generals Problem (BGP). 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 commander is loyal, then every loyal lieutenant obeys the order he sends.

Interactive Consistency Conditions

Solving the Original Problem

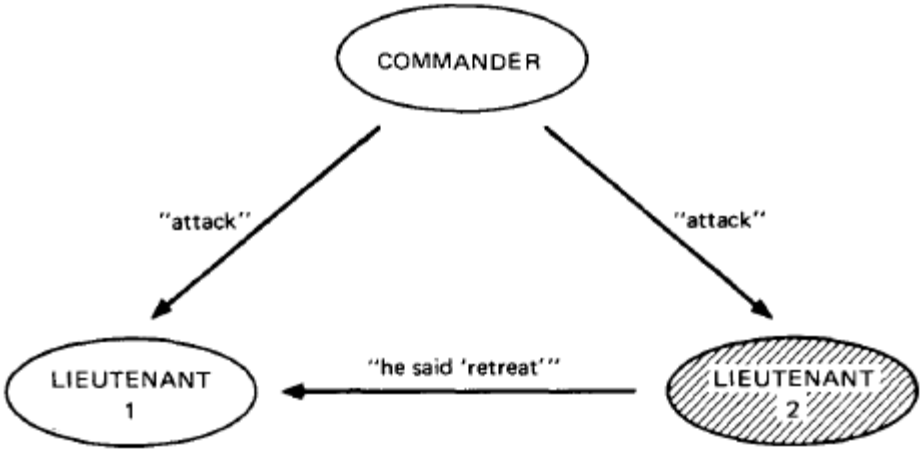
To solve the original problem, each general i employs a solution to the Bizantine Generals Problem to send the order “use v(i) as my value”, with the other generals acting as the lieutenants.

Impossibility Results

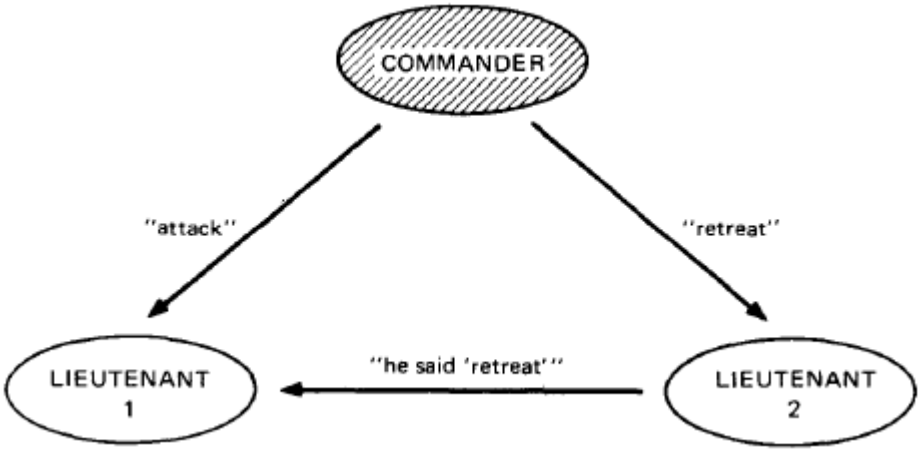
3 generals, 1 traitor among them.

L1 sees {“attack”, “retreat”}.

Who is the traitor? C or L2?



L1 has to attack to satisfy IC2.



L1 attacks, L2 retreats. IC1 violated.

No solution to the BGP exists with 3 generals and 1 traitor.

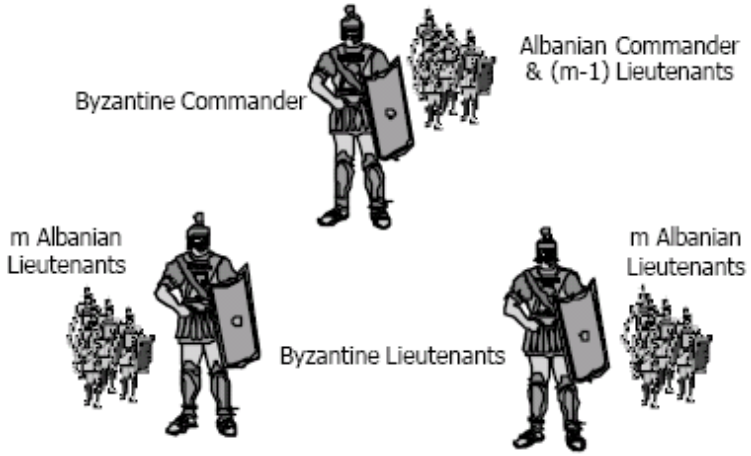
Impossibility Results

No solution with fewer than 3*m*+1 generals can cope with *m* traitors.

Proof by contradiction.

Assume there is a solution for 3*m* *Albanian* generals with *m* traitors.

Reduce to 3-BGP.



Solution to 3*m* Albanian generals problem → solution to 3-BGP!

Solutions Principle

A principle at work in a Byzantine setting:

“we can trust the group but not the individuals”

Barbara Liskov

Solution I – Oral Messages

An oral message is one whose contents are completely under the control of the sender.

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.

Solution I – Oral Messages

A traitorous commander may decide not to send any order.

Since the lieutenants must obey some order, they need some default order to obey in this case.

We let RETREAT be this default order.

Solution I – Oral Messages

The algorithm assumes a function *majority* returning

- the majority value among the v_i if it exists, otherwise the value RETREAT, or
- the median of the v_i if they come from an ordered set.

Solution I – Oral Messages

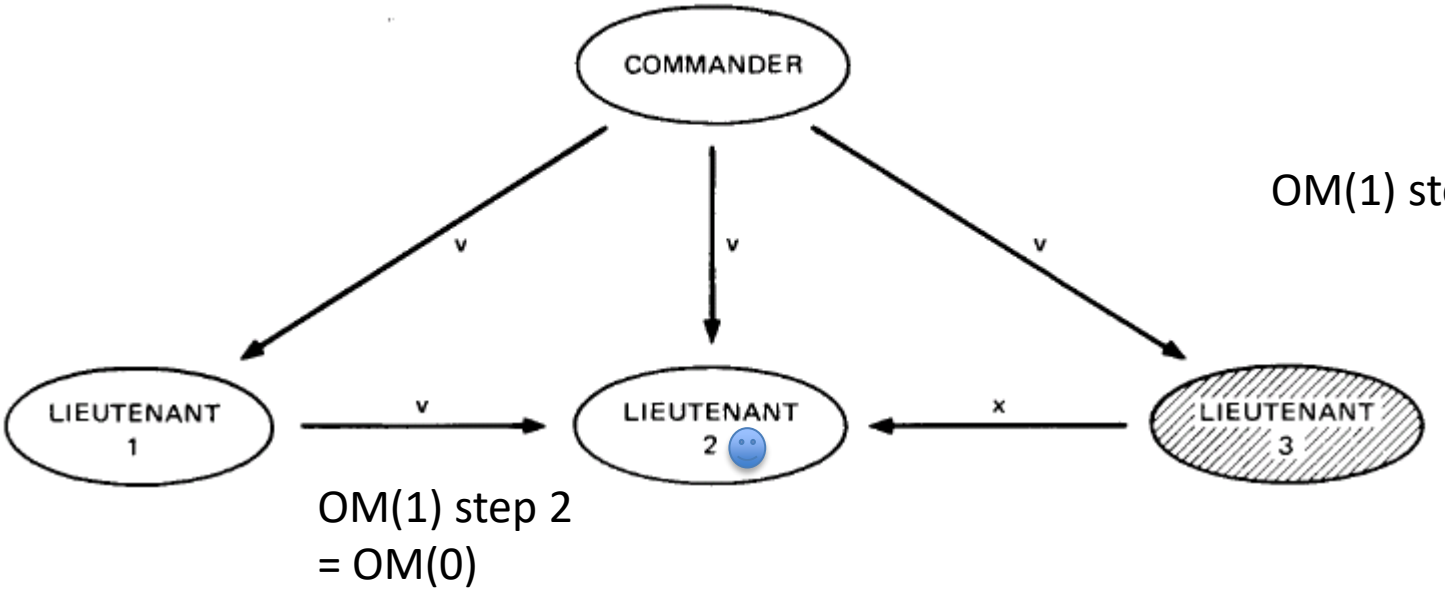
Algorithm OM(0)

1. The commander sends his value to every lieutenant.
2. Each lieutenant uses the value received from the commander.

Algorithm OM(m), m>0

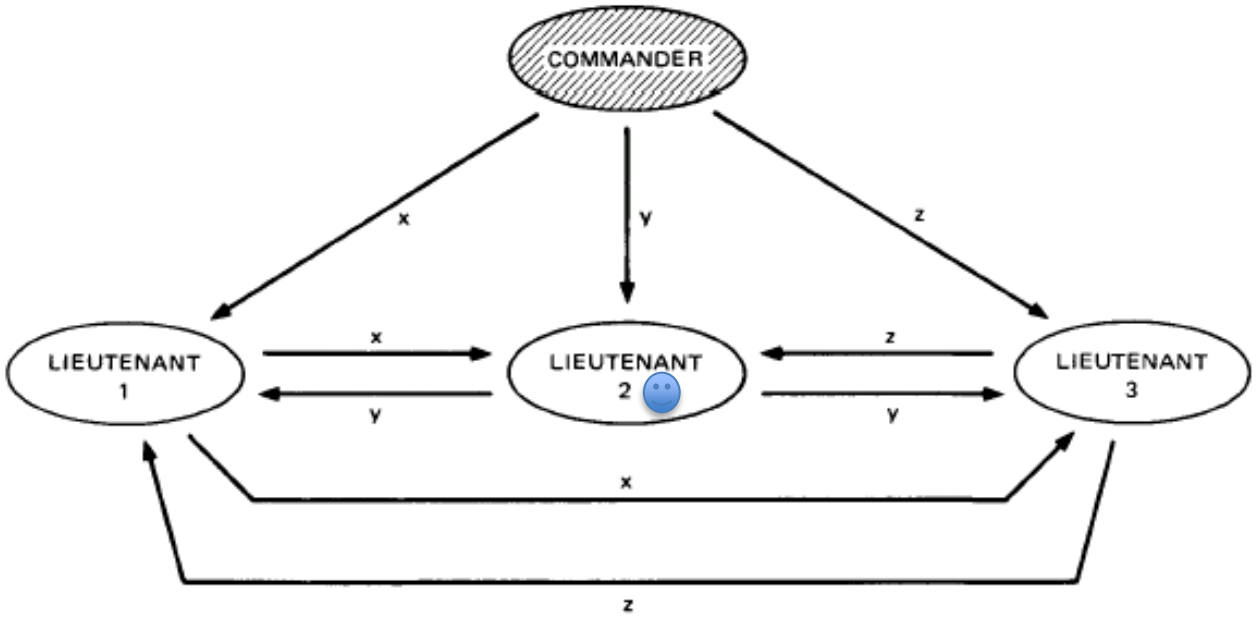
1. The commander sends his value to every lieutenant (v_i).
2. For each *i*, let v_i be the value lieutenant *i* receives from the commander. Lieutenant *i* acts as the commander in OM(m-1) to send the value v_i to each of the n-2 other lieutenants.
3. For each *i*, and each $j \neq i$, let v_j be the value lieutenant *i* received from lieutenant *j* in step 2 (using OM(m-1)). Lieutenant *i* uses the value *majority*($v_1, v_2,\ldots,v_{n-1}$).

Example (n=4, m=1)



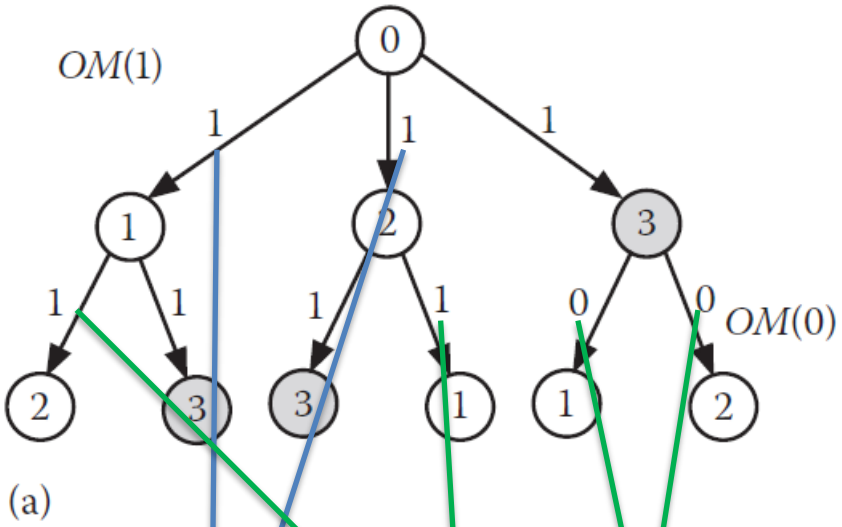
- Algorithm OM(1): Lieutenant 3 is a traitor.
- L2 and L1 both receive {v,v,x}. (IC1 is met)
- IC2 is met because L1 and L2 obeys C. (v=majority(v,v,x))

Example (n=4, m=1)

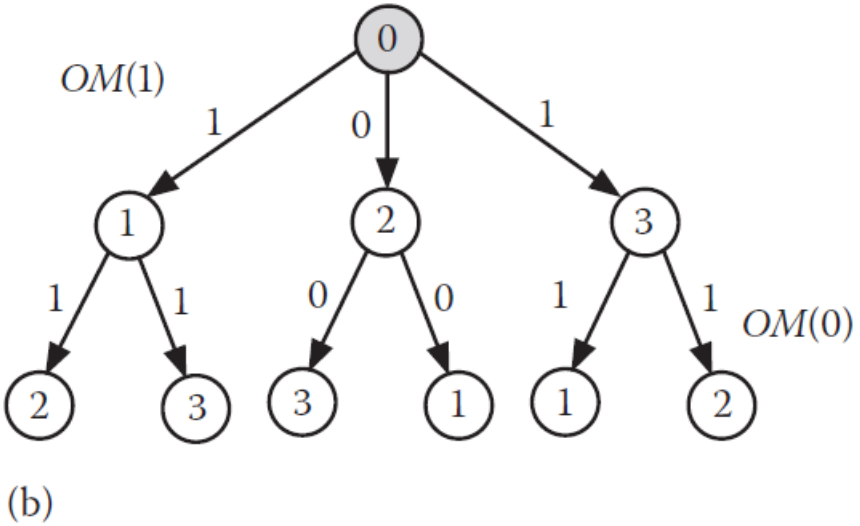


- Algorithm OM(1): Commander is a traitor.
- All lieutenants receive {x,y,z}. (IC1 is met)
- IC2 is irrelevant since commander is a traitor.

Example (n=4, m=1)



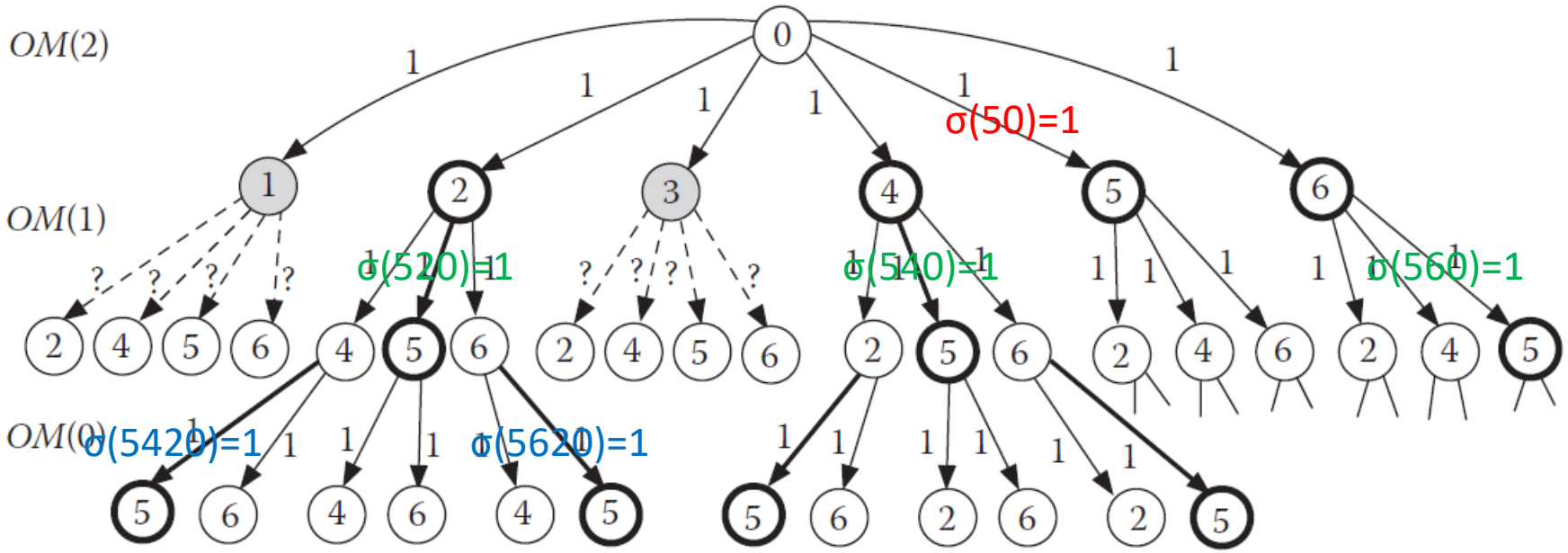
L1 = {σ(10)} + {σ(120), σ(130)}
L2 = {σ(20)} + {σ(210), σ(230)}
L1 = {1} + {1,0} = {1,1,0}
L2 = {1} + {1,0} = {1,1,0}
Majority({1,1,0}) = 1



L1 = {1} + {0,1} = {1,0,1}
L2 = {0} + {1,1} = {0,1,1}
L3 = {1} + {1,0} = {1,1,0}
Majority() = 1

Example (n=7, m=2)

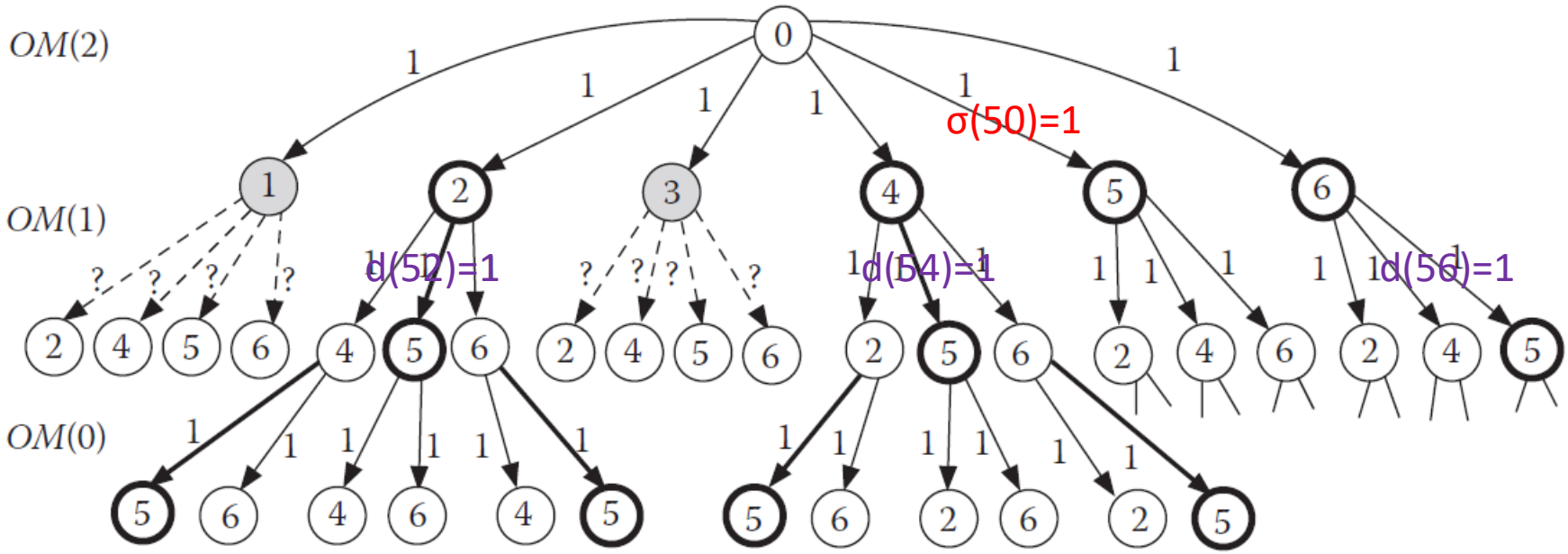
Partial unfolding of the recursion



Loyal comander

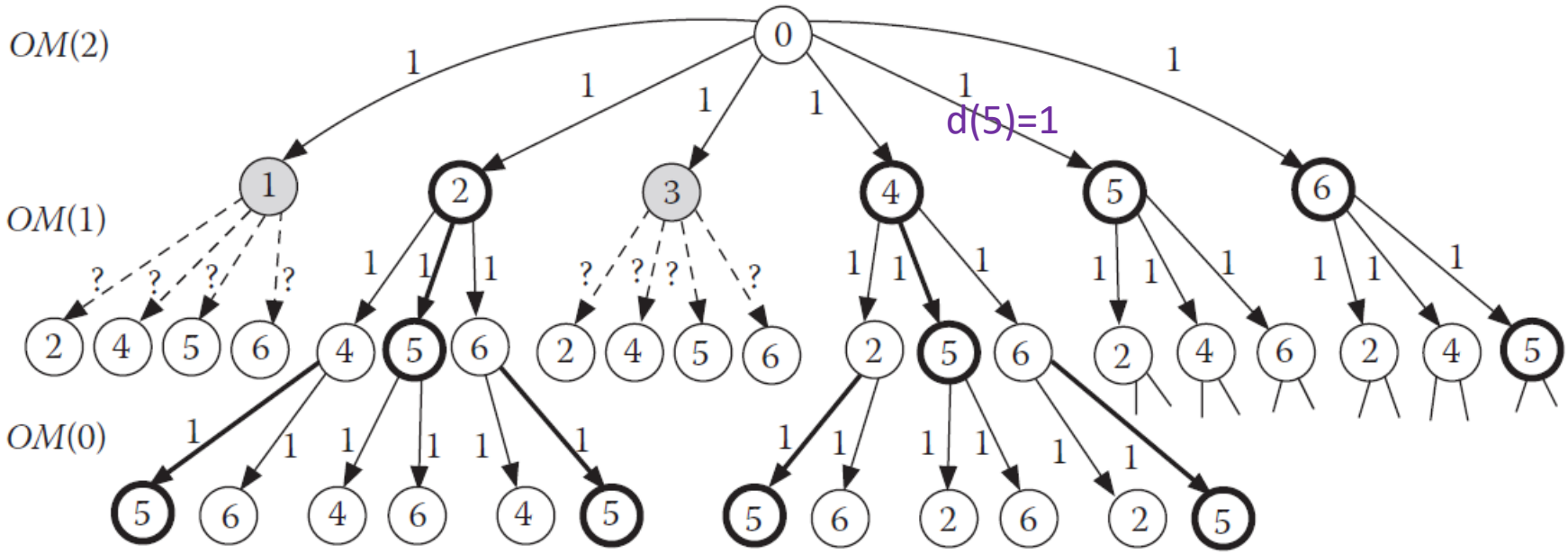
Example (n=7, m=2)

Partial folding of the recursion



Example (n=7, m=2)

Partial folding of the recursion



m+1 synchronous rounds

Expensive Communication

- OM(m) invokes n-1 OM(m-1)
- OM(m-1) invokes n-2 OM(m-2)
- …
- OM(m-k) will be called (n-1)…(n-k) times
- …
- OM(1) invokes n-m OM(0)
- Each OM(0) will send n-(m-1) messages

- $O(n^{m+1})$ expensive!

There are solutions with polynomial communication cost

Solution II - Signed messages

Previous algorithm allows a traitor to lie about the commander's orders.

If we add one more assumption, we can make the problem a lot easier:

A4: Messages are signed.

a) A loyal general's signature cannot be forged, and any alteration of the contents of his signed messages can be detected.

b) Anyone can verify the authenticity of a general´s signature.

Solution II - Signed messages

Let *x*:*i* denote the value *x* signed by General *i.* Thus, *v:j:i* denotes the value *v* signed by *j,* and then that value *v:j* signed by *i.*

Each lieutenant maintains a set V of properly signed orders received so far.

When a lieutenant will receive no more messages, he obeys the order *choice*(V)

- *choice*(V) = v if v is the only element in V
- *choice*(V) = RETREAT if V is empty

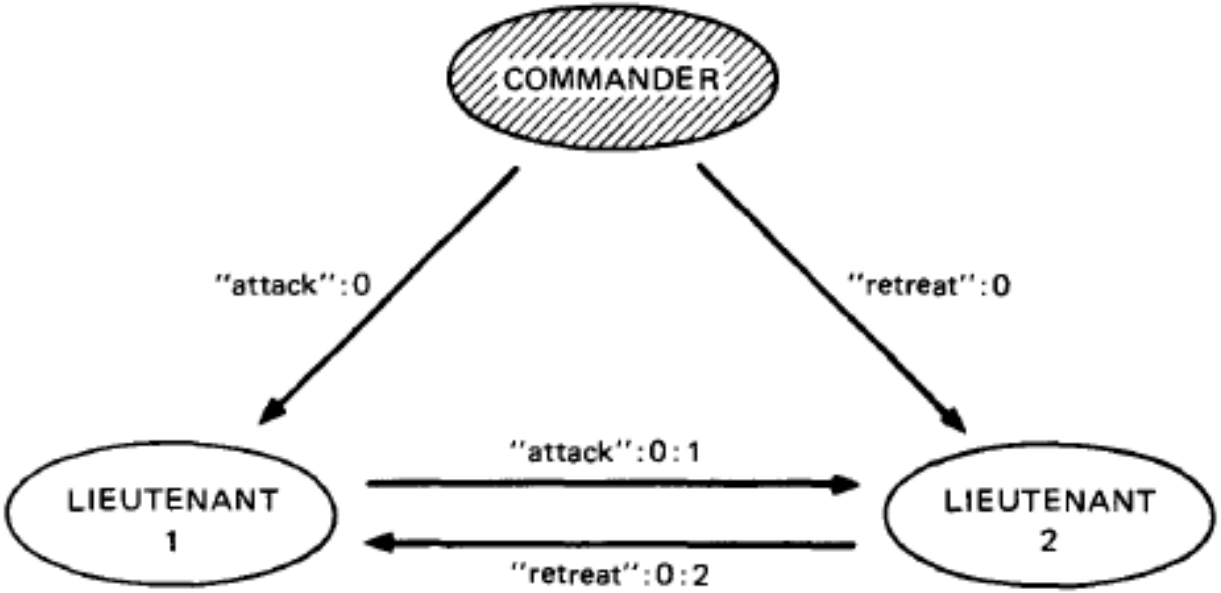
Solution II - Signed messages

Algorithm SM(m)

Initially $V_i = \{\}$.

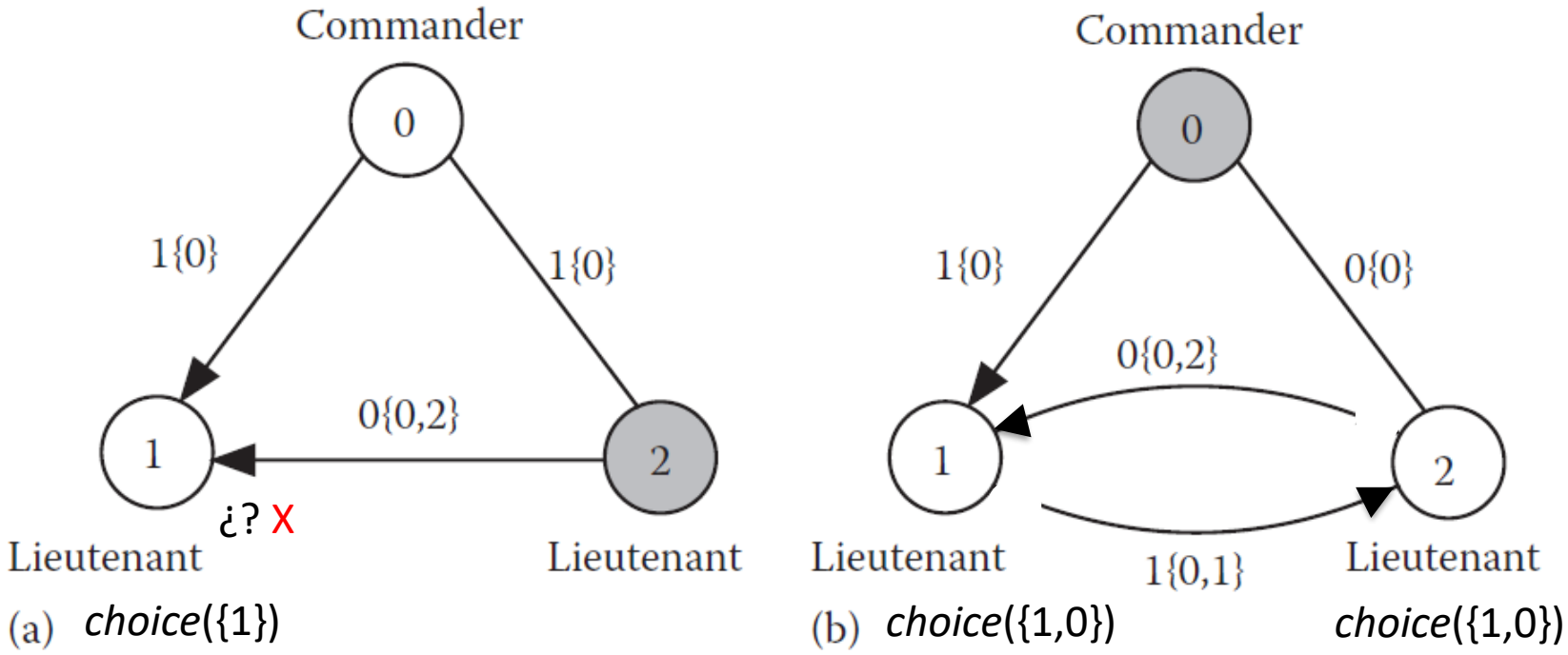
1. The commander signs and sends his value to every lieutenant.
2. For each *i:*
   A. If Lieutenant *i* receives a message of the form *v:*0 from the commander and he has not yet received any order, then
      i. he lets V_i equal *{v};*
      ii. he sends the message *v:*0:*i* to every other lieutenant.
   B. If Lieutenant *i* receives a message of the form *v*:0:j_1: ... : j_k and *v* is not in the set V_i, then
      i. he adds *v* to V_i;
      ii. if *k*<*m,* then he sends the message *v:*0:j_1: ... :j_k:*i* to every lieutenant other than j_1, ... ,j_k
3. For each *i:* When Lieutenant *i* will receive no more messages, he obeys the order *choice(*V_i *).*

Example (n=3, m=1)



choice({"attack", "retreat"})

Example (n=3, m=1)



Algorithm SM(1)

Solution II - Signed messages

By using signed messages:

- We can cope with any number of traitors as long as their number *m* is known.
- All loyal generals will agree on a common result after *m*+1 rounds.

Problem Specifications

Byzantine Agreement

(single source process has an initial value)

- Agreement: All non-faulty processes must agree on the same value.
- Validity: If the source process is non-faulty, then the agreed upon value by all the non-faulty processes must be the same as the initial value of the source.
- Termination: Each non-faulty process must eventually decide on a value.

Problem Specifications

Interactive Consistency

(each process has an initial value)

- Agreement: All non-faulty processes must agree on the same array of values $A[v_1 \ldots v_n]$.
- Validity: If process i is non-faulty and its initial value is v_i , then all non-faulty processes agree on v_i as the ith element of the array A. If process j is faulty, then the non-faulty processes can agree on any value for A[j].
- Termination: Each non-faulty process must eventually decide on the array A.

Problem Specifications

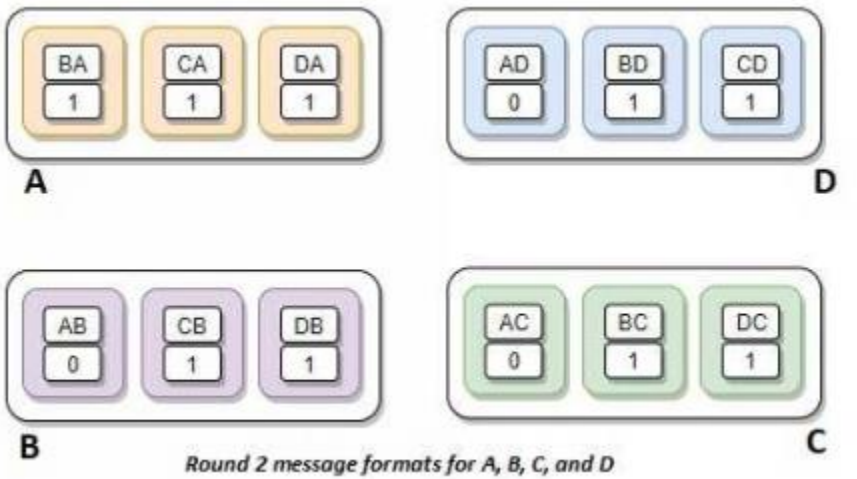
Consensus

(each process has an initial value)

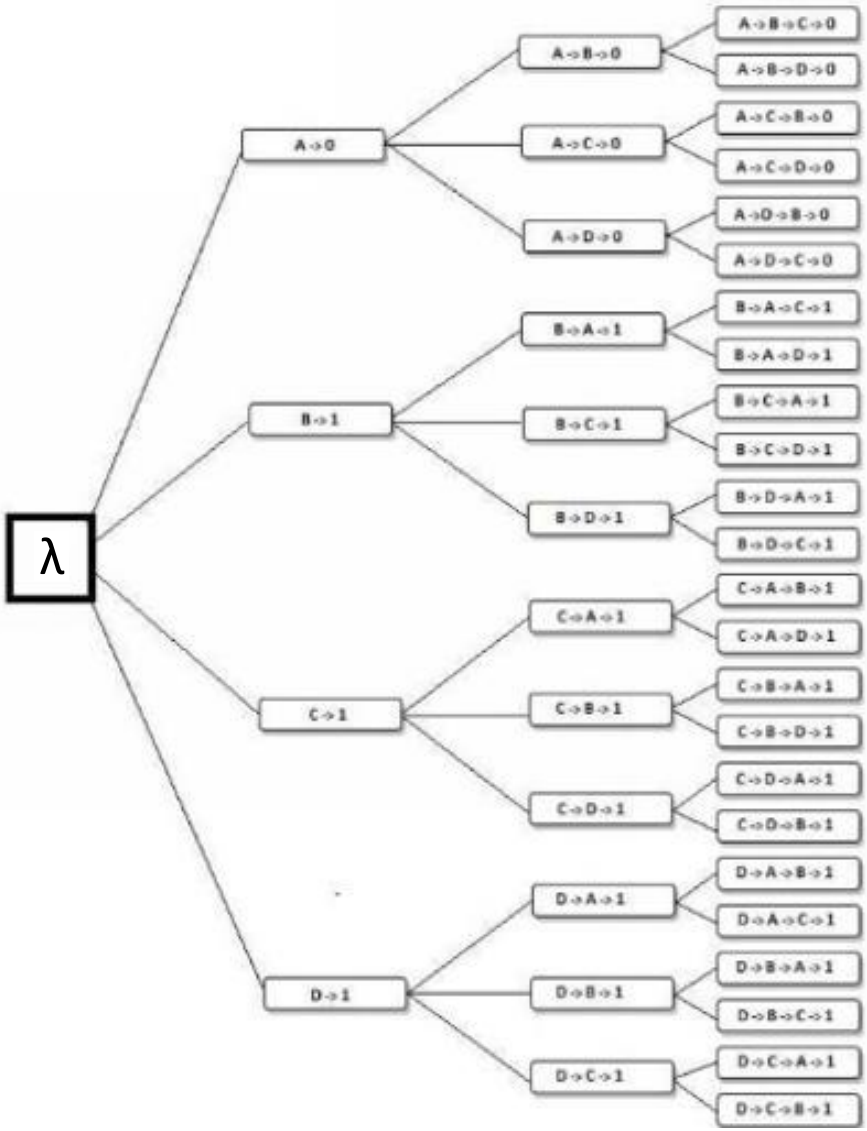
- Agreement: All non-faulty processes must agree on the same (single) value.
- Validity: If all the non-faulty processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value.
- Termination: Each non-faulty process must eventually decide on a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

EIG

Tree



Round 2 message formats for A, B, C, and D



Correctnes

Safety

The algorithm satisfies agreement and validity

Liveness

The algorithm satisfies termination

The Byzantine Generals Problem

L. Lamport, R. Shostak, and M. Pease

ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982, pp. 382-401.

Oral Messages

(variables)
boolean: v ⟵ initial value;
integer: f ⟵ maximum number of malicious processes, $\leq \lfloor (n-1)/3 \rfloor$;
(message type)
OralMsg(v, *Dests*, *List*, *faulty*), where
v is a boolean,
Dests is a set of destination process ids to which the message is sent,
List is a list of process ids traversed by this message, ordered from most recent to earliest,
faulty is an integer indicating the number of malicious processes to be tolerated.

OralMsg(f), where $f > 0$:

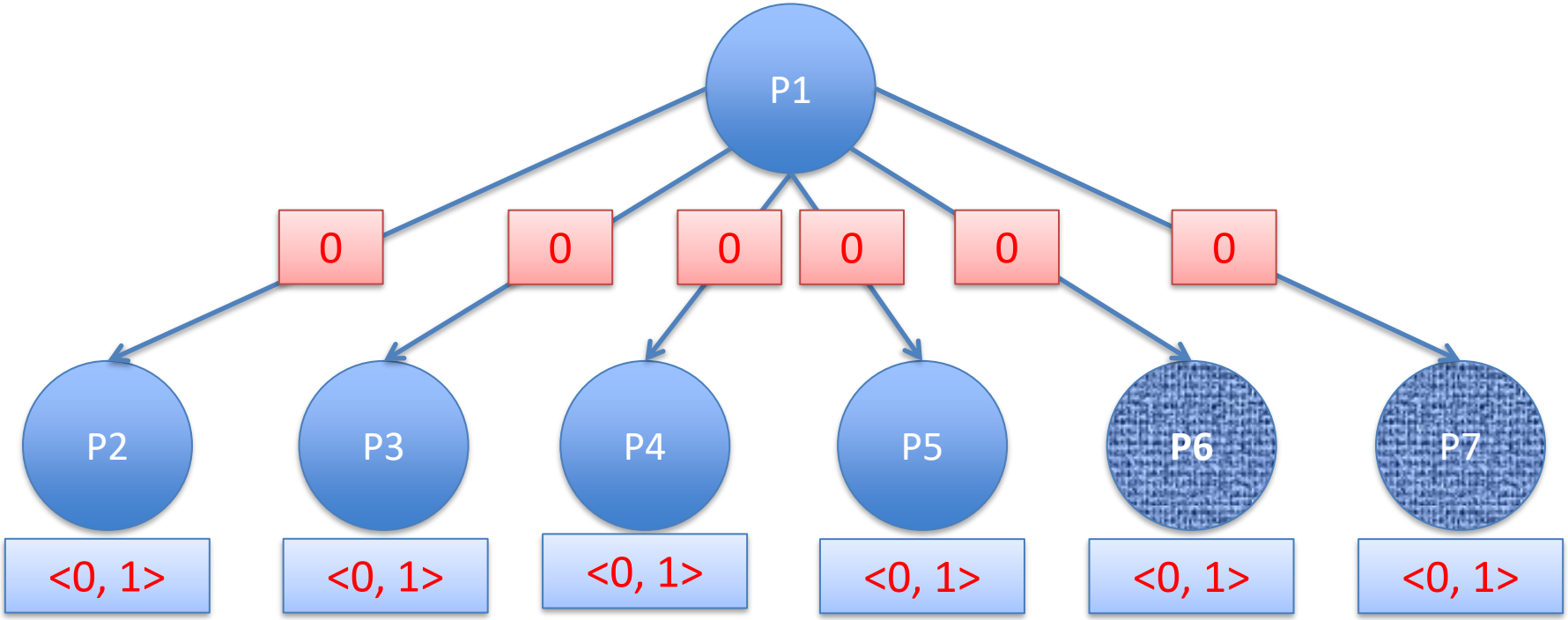
1. The algorithm is initiated by the Commander, who sends his source value v to all other processes using a $OM(v, N, \langle i \rangle, f)$ message. The commander returns his own value v and terminates.
2. **[Recursion unfolding:]** For each message of the form $OM(v_j, Dests, List, f')$ received in this round from some process j, the process i uses the value v_j it receives from the source, and using that value, acts as a *new* source. (If no value is received, a default value is assumed.)
   To act as a new source, the process i initiates $OralMsg(f' - 1)$, wherein it sends
   $OM(v_j, Dests - \{i\}, concat(\langle i \rangle, L), (f' - 1))$
   to destinations not in $concat(\langle i \rangle, L)$
   in the next round.
3. **[Recursion folding:]** For each message of the form $OM(v_j, Dests, List, f')$ received in Step 2, each process i has computed the agreement value v_k, for each k not in *List* and $k \neq i$,corresponding to the value received from P_k after traversing the nodes in *List*, at one level lower in the recursion. If it receives no value in this round, it uses a default value. Process i then uses the value $majority_{k \notin List, k \neq i}(v_j, v_k)$ as the agreement value and returns it to the next higher level in the recursive invocation.

OralMsg(0):

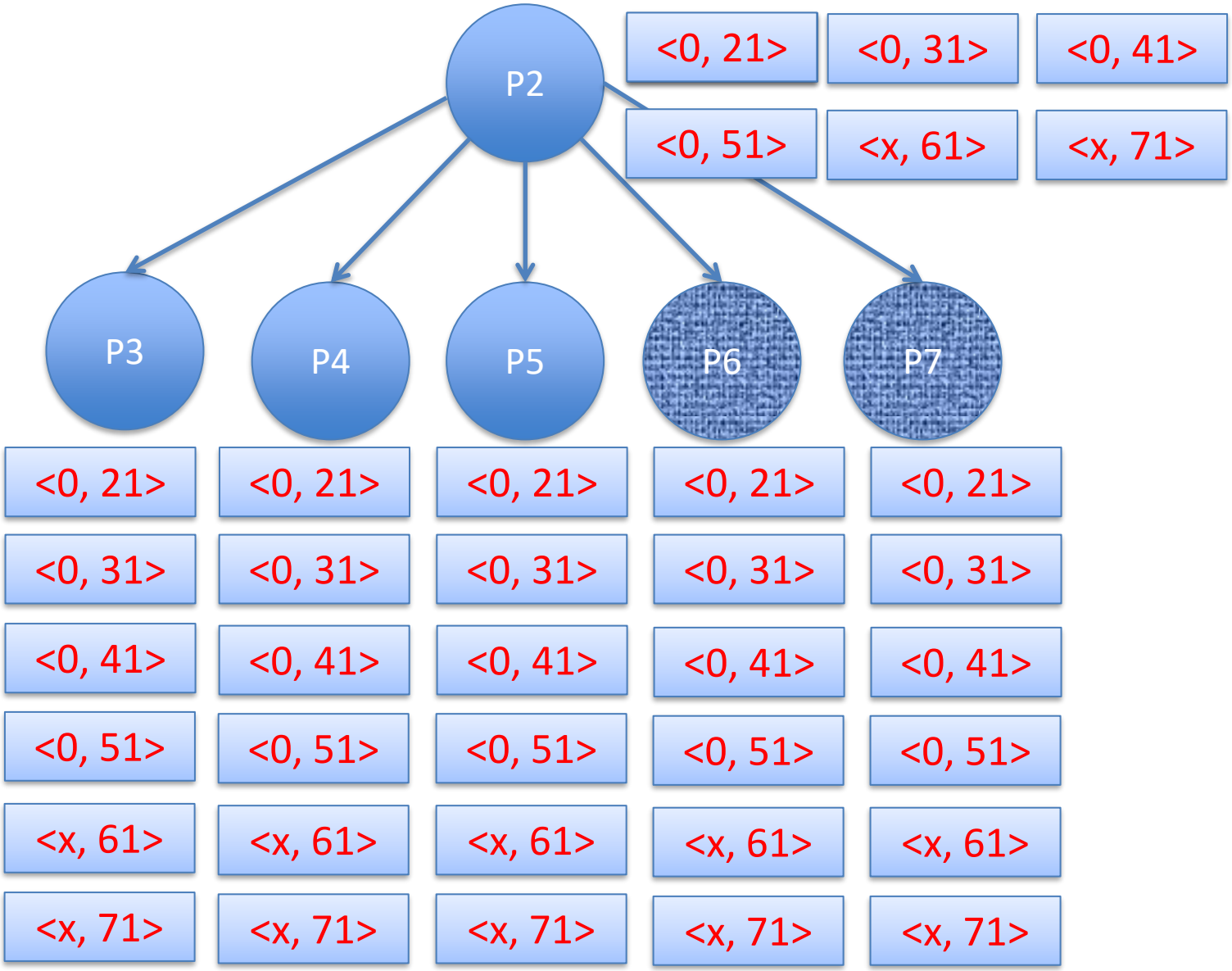
1. **[Recursion unfolding:]** Process acts as a source and sends its value to each other process.
2. **[Recursion folding:]** Each process uses the value it receives from the other sources, and uses that value as the agreement value. If no value is received, a default value is assumed.

OM(2) broadcast

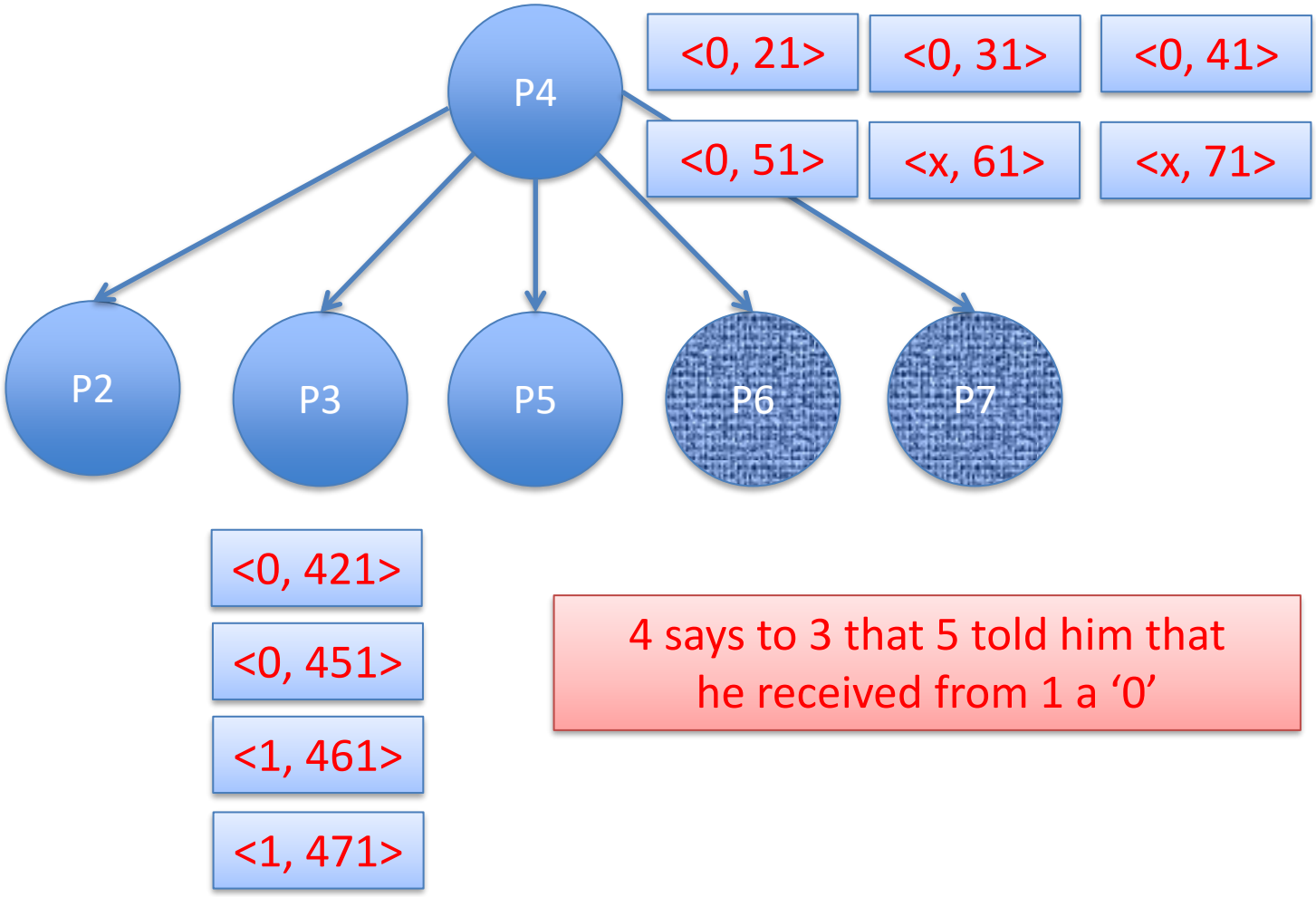
- Let, m = 2. Therefore, n = 3m + 1 = 7
- Commander sends order to all the lieutenants



OM(1) broadcasts

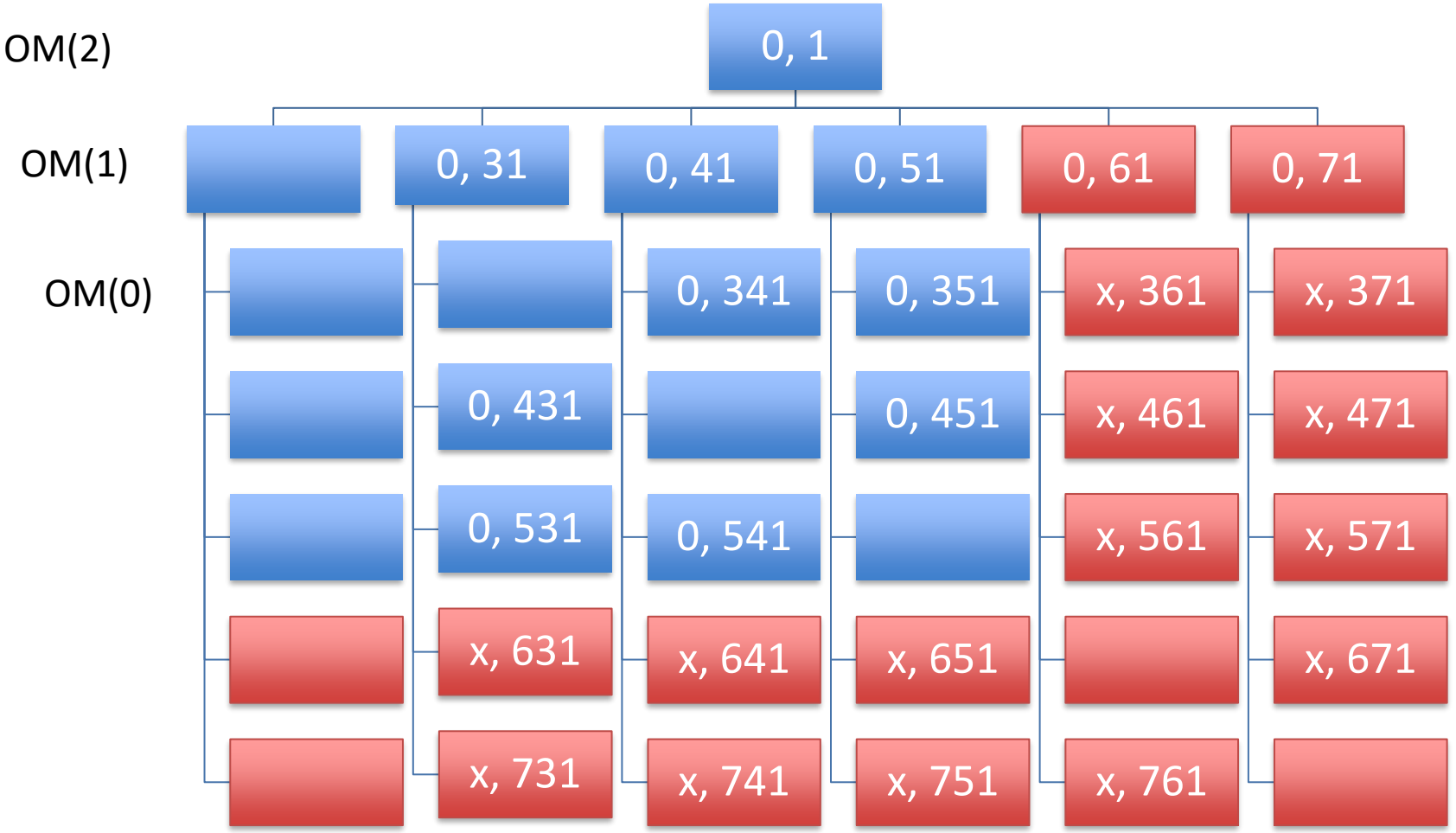


OM(0) broadcasts



Received Messages

P2



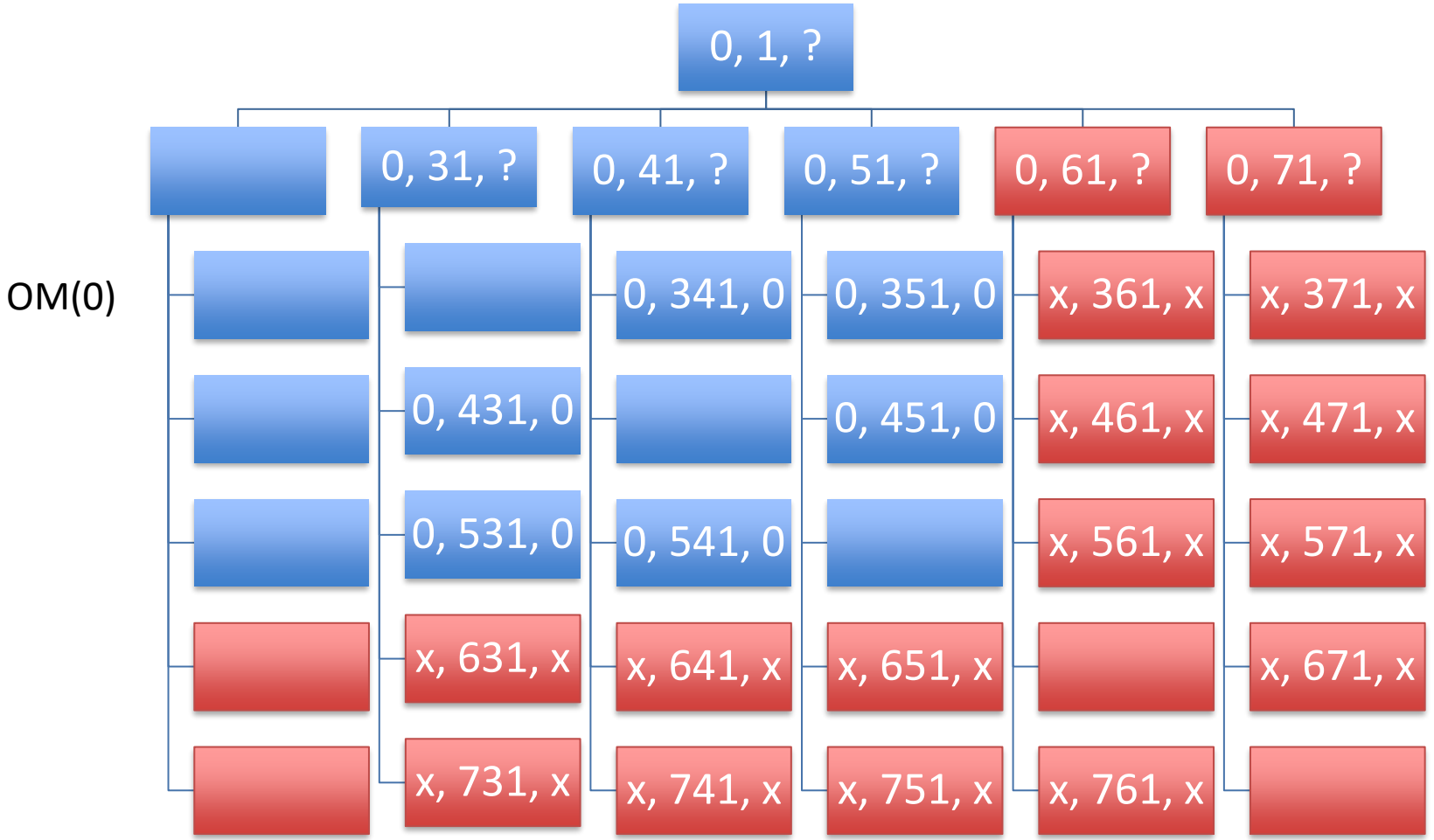
Use value

P2


0, 1, ?
0, 31, ?
0, 41, ?
0, 51, ?
0, 61, ?
0, 71, ?
0, 341, ?
0, 351, ?
x, 361, ?
x, 371, ?
0, 431, ?
0, 451, ?
x, 461, ?
x, 471, ?
0, 531, ?
0, 541, ?
x, 561, ?
x, 571, ?
x, 631, ?
x, 641, ?
x, 651, ?
x, 671, ?
x, 731, ?
x, 741, ?
x, 751, ?
x, 761, ?

OM(0) - *received*

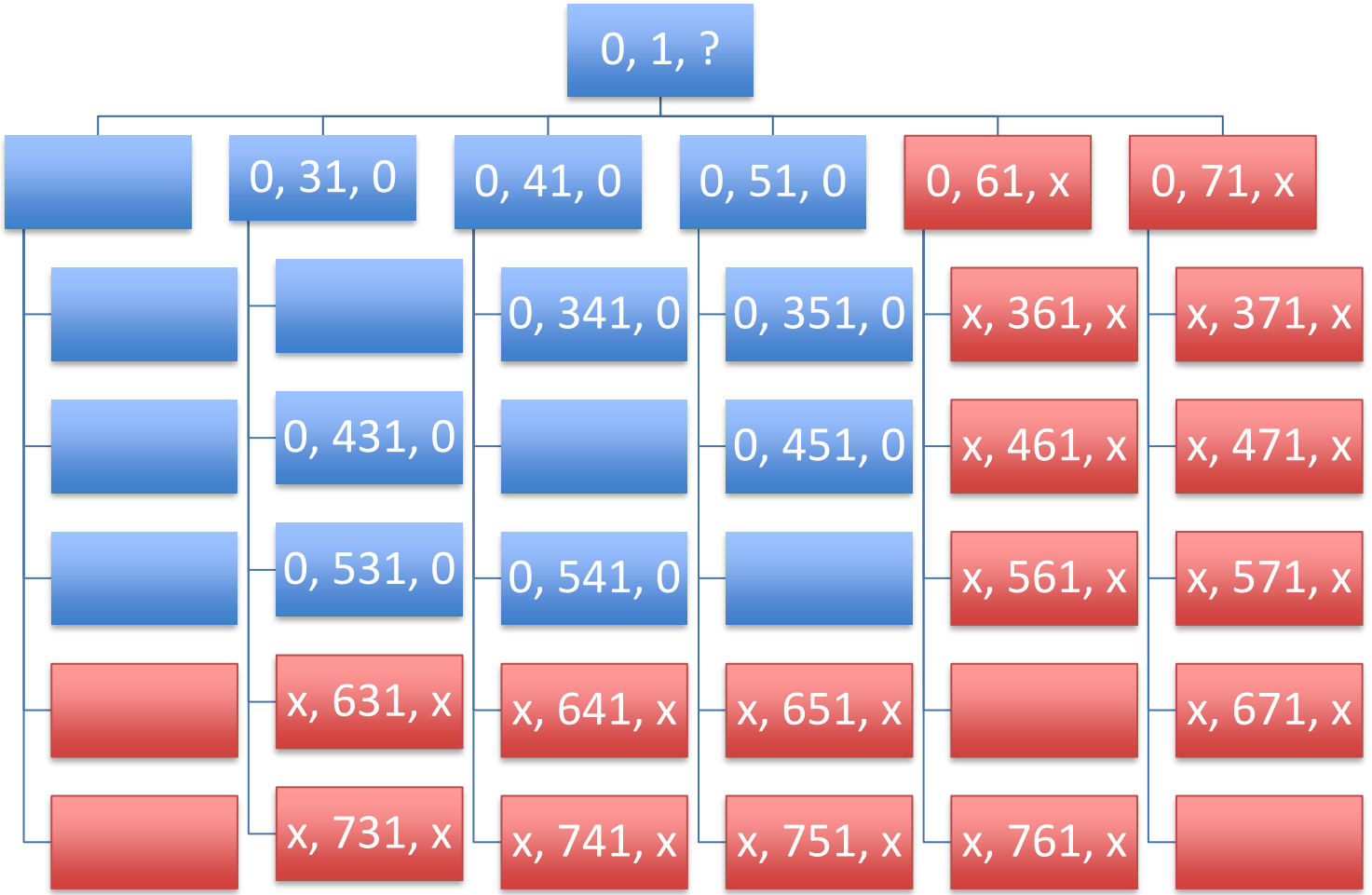
P2



OM(1) - *majority*

P2

OM(1)



OM(2) - *majority*

P2

OM(2)