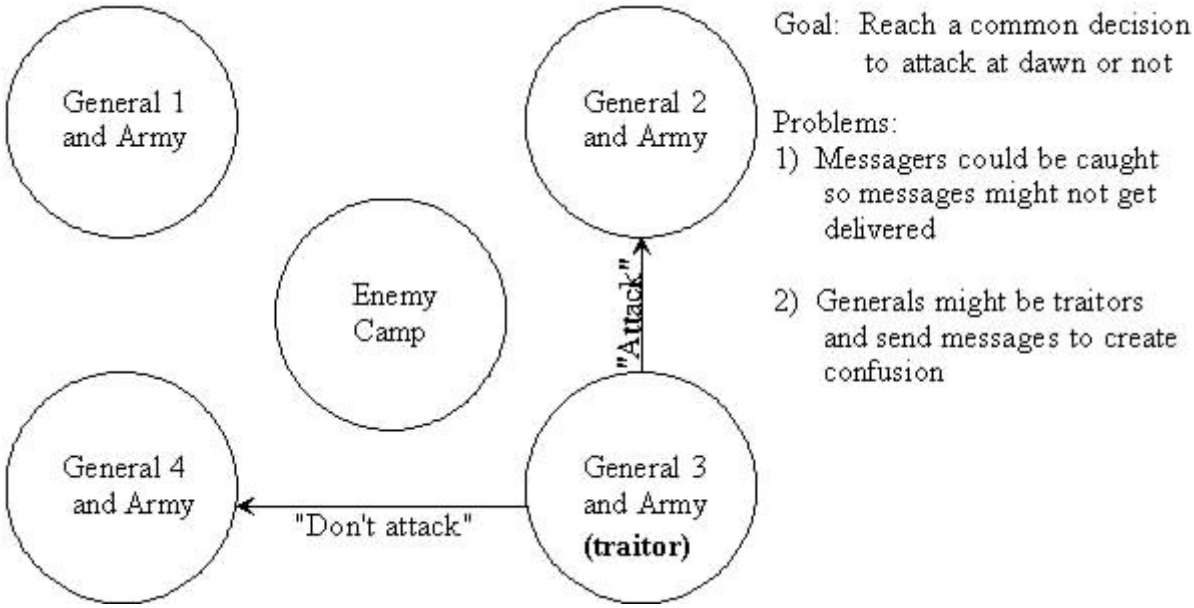
Reaching Agreement ("Byzantine Generals Problem") - agreeing on a common value

(NOTE: the value agreed upon is not necessarily the majority value)

Faulty processes might send incorrect messages making "reaching agreement" on a common value difficulty. In the worst case, the faulty processes might collaborate together to force agreement on an incorrect value.

Byzantine Generals Problem

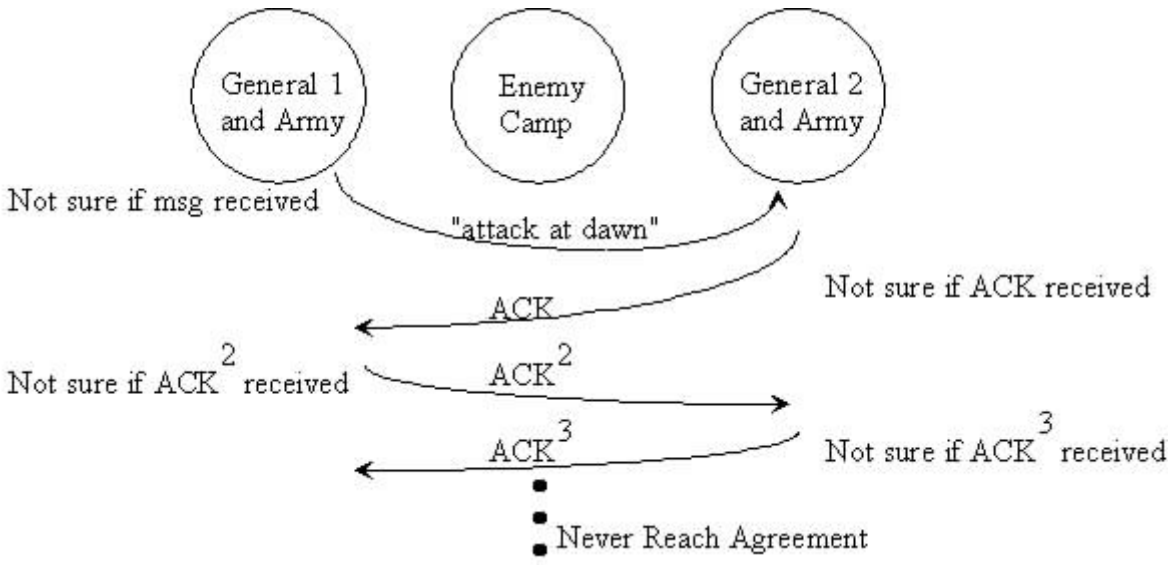


All loyal generals should agree on the same decision. If all loyal generals had the same initial decision about whether to attack or not, then they should decide on this same value

Unreliable Communication - messages might be lost

Assume nonfaulty processes (e.g., no traitor generals).

Generals want to agree to attack or not by exchanging messages. Consider two-general case.



Faulty Processes - assume reliable communication, but processes can fail and send messages unpredictably. All nonfaulty processes must agree on the same value.

Let

n = the total (faulty and nonfaulty) number of processes

m = the total number of faulty processes

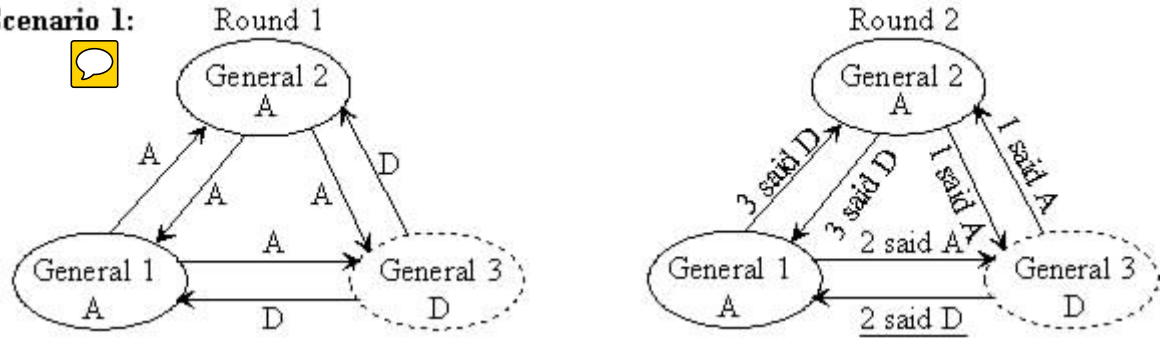
m + 1 rounds of messages must be exchanged to allow a processes to have complete information.

Lamport ('82) showed that agreement can be reached with m faulty processes only if the total number of processes is such that n $\geq$ 2m + 1. THIS ASSUMES that messages cannot be corrupted and identity of senders can be authenticated.

Lamport ('82) showed that agreement can be reached with m faulty processes only if the total number of processes is such that n $\geq$ 3m + 1. THIS DOES NOT rely on any assumptions that messages cannot be corrupted and identity of senders can be authenticated.

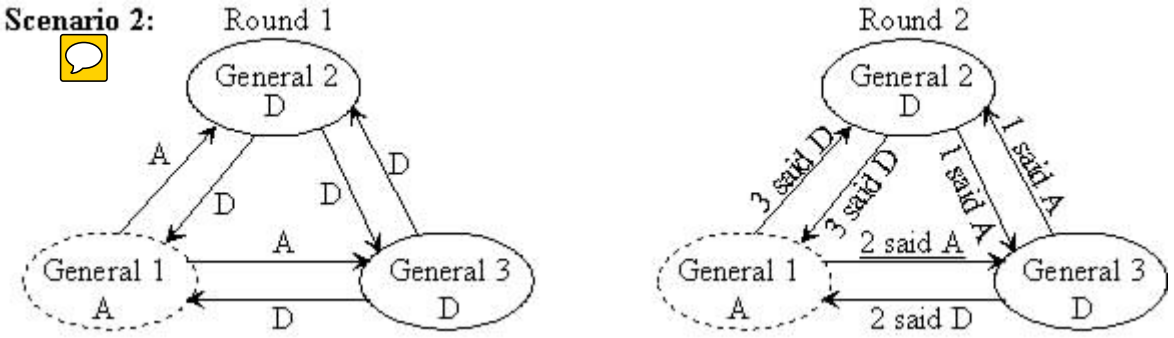
If n < 3m + 1, then agreement cannot be reached. To help see this, consider the following three scenarios

Scenario 1:



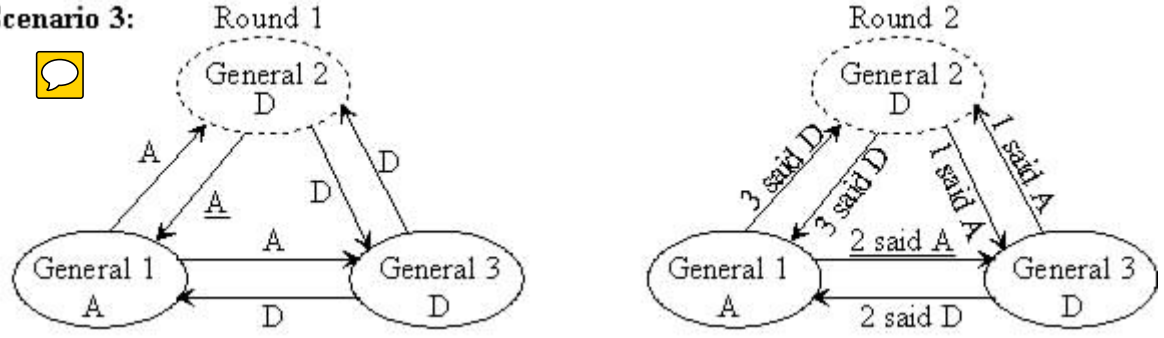
In scenario 1, generals 1 and 2 should decide to attack (A) since they both initially agreed.

Scenario 2:



In scenario 1, generals 2 and 3 should decide don't attack (D) since they both initially agreed.

Scenario 3:



In this situations, generals 1 and 3 conflict in their initial values so what should they decide?

What is true about General 1 in scenarios 1 and 3?

What is true about General 3 in scenarios 2 and 3?

In scenario 3 if generals 1 and 3 decided to Attack, then how could General 3 in scenario 2 decide to don't attack?

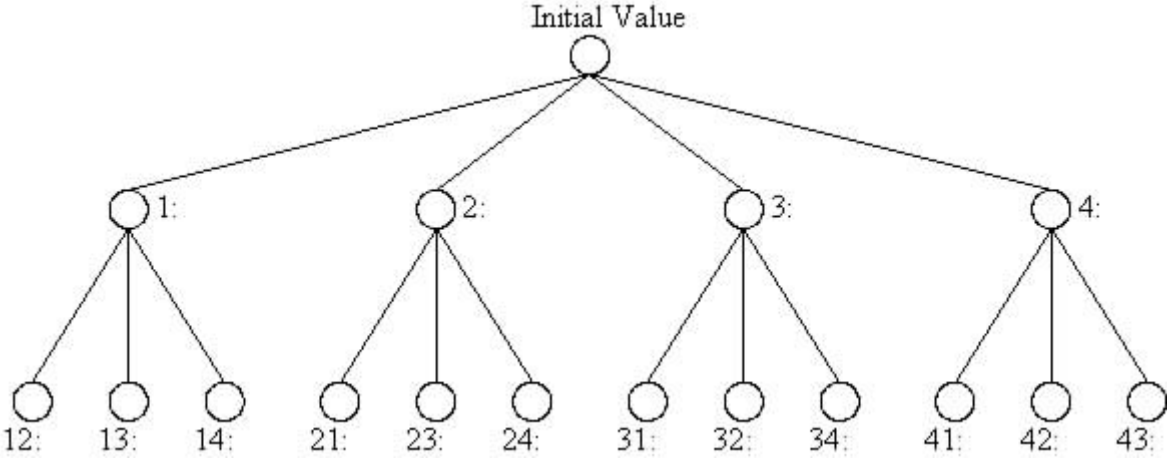
In scenario 3 if generals 1 and 3 decided to don't attack, then how could General 1 in scenario 1 decide to attack?

Byzantine Agreement Algorithm for n processes and up to m faulty processes

1) Information exchange stage - m + 1 rounds of information exchange with each round consisting of sending "trace-labeled" messages received in the previous round to processes that have not seen the message.

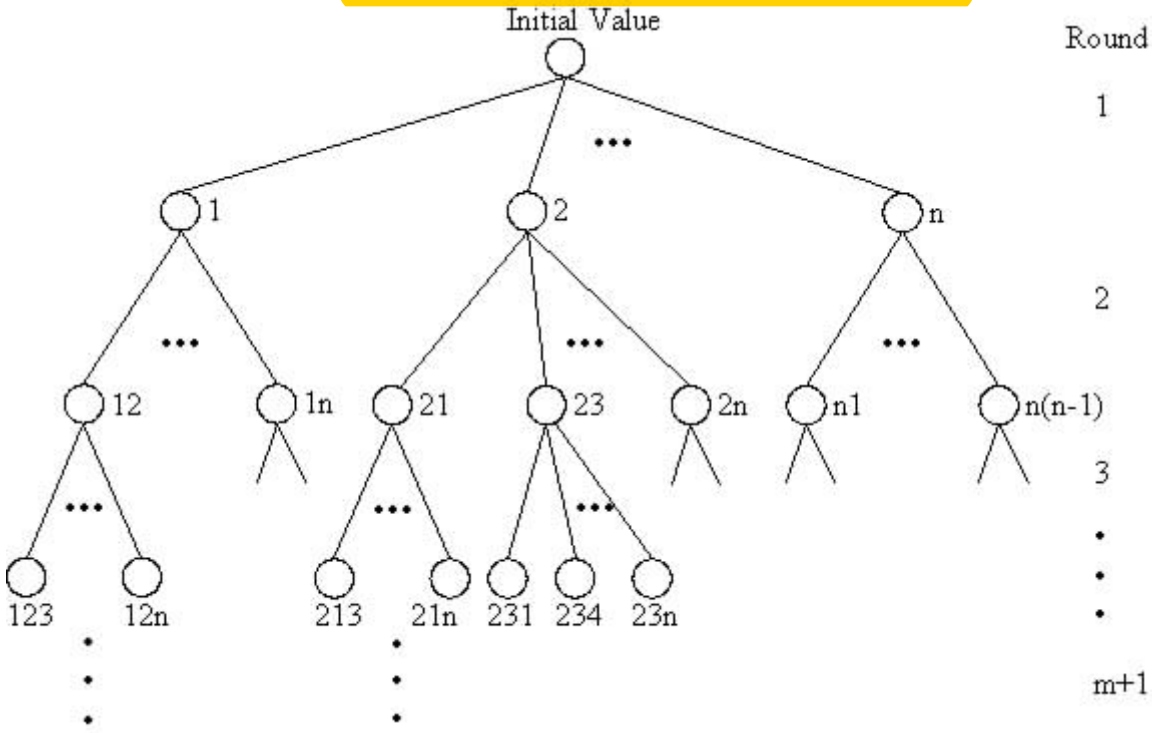
Messages between processes contain a label $i_1 i_2 ... i_k$ that traces the route of a value, e.g., during round 3 of message passing if process 4 receives message 275:Attack, then this message means that process 5 told process 4 in round 3 that process 7 told process 5 in round 2 that process 2 told process 7 in round 1 that process 2's initial value was Attack. After process 4 records this message, Process 4 will append its process number to the label and forward this message to all processes that are not already in the label.

The processes record these message values in an exponential information gathering (EIG) tree. If no value or a garbage value is received, then a *NULL* value is recorded. The shape of the tree depends on the number of processes and the degree of failure. For n = 4 and m = 1, the EIG tree would look like:

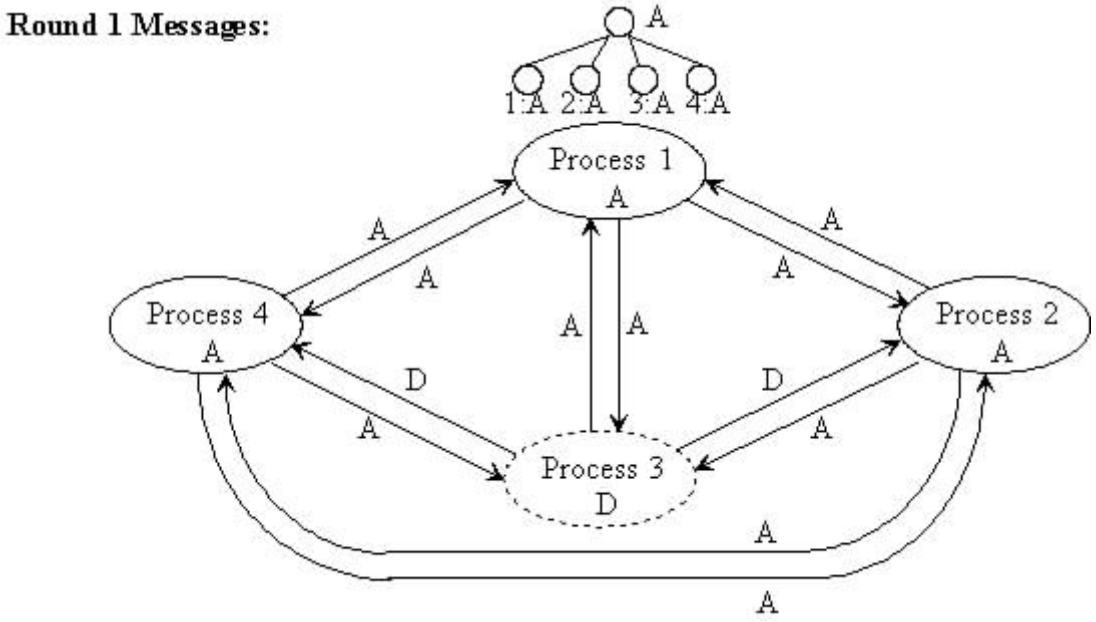


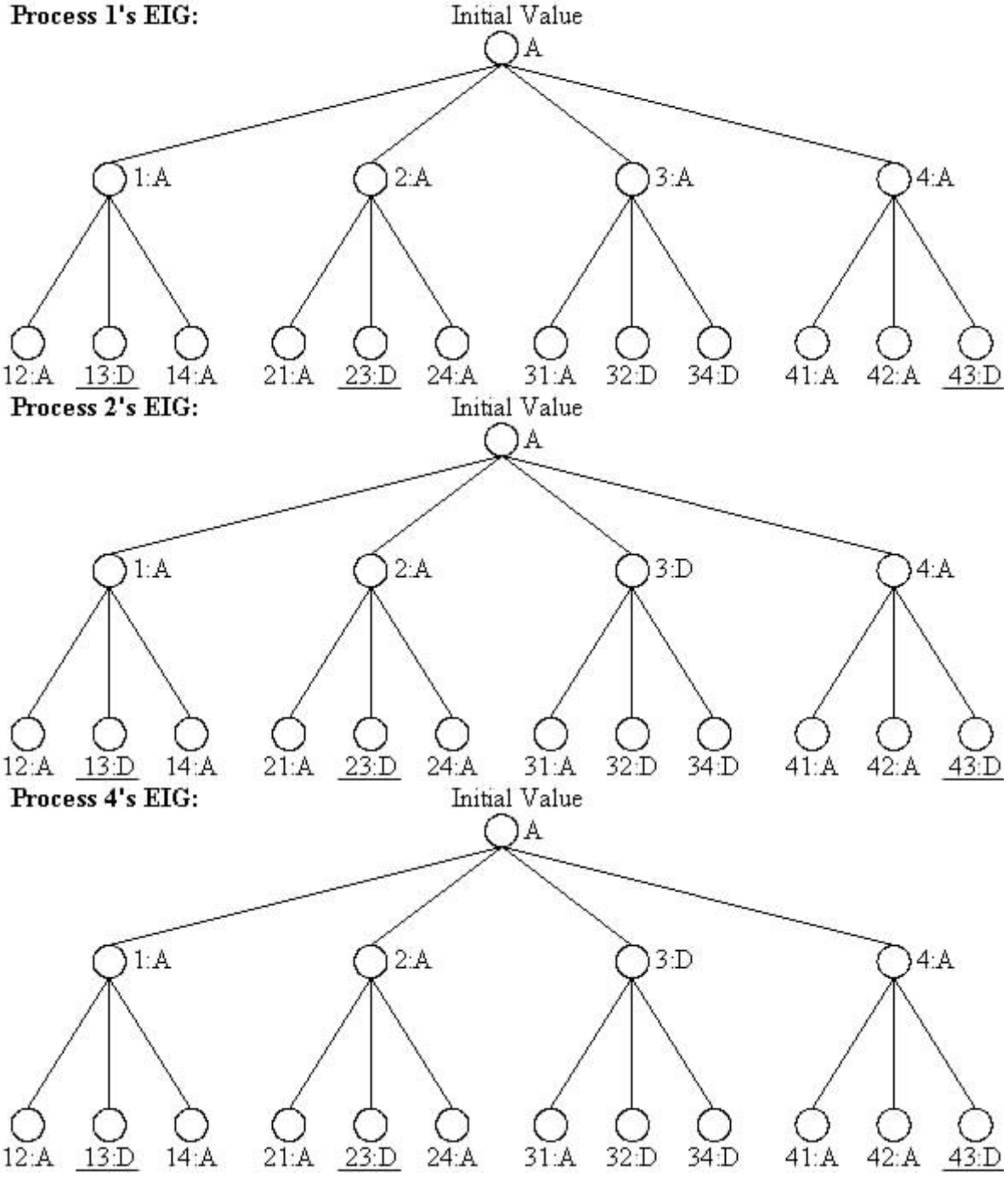
2) Processor Decision Stage - Each processor uses its EIG tree to make it decision.

In general the EIG tree for n processes and m failures will have m+1 levels.



At the beginning of round r, each processor sends the r^{th} level of its tree





Algorithm for Filling the EIG Tree:

Initially each process i assigns the root of its EIG tree with its own initial value.

Round 1:

Sending round 1 messages: Process i sends its initial value to all processes (including itself).

Receiving round 1 messages: When process i receives a message from process j during round 1, it either:

a) assigns the node labeled j in level 1 of the EIG with the value received in the message, or

b) assigns the node labeled j in level 1 of the EIG with *NULL* if the message received contains garbage.

If no message from process j was received during round 1, it assigns the *NULL*value.

Round k (for k between 2 and m+1):

Sending round k messages: Process i sends/forwards each values received during round k-1 to all processes that have not seen this value before, i.e., the value v at a EIG node labeled $p_1p_2p_3...p_{k-1}$ will be sent as the message $p_1p_2p_3...p_{k-1} : v$ to process j if j $\notin \{p_1, p_2, p_3, ..., p_{k-1}\}$.

Receiving round k messages: When process j receives the message $p_1p_2p_3...p_{k-1} : v$ from process i during round k, it either:

a) assigns the node labeled $p_1p_2p_3...p_{k-1}$ i in level k of the EIG with the value v, or

b) assigns the node labeled $p_1p_2p_3...p_{k-1}$ i in level k of the EIG with *NULL* if the message received contains garbage.

If no message from process j was received during round k, it assigns the *NULL* value.

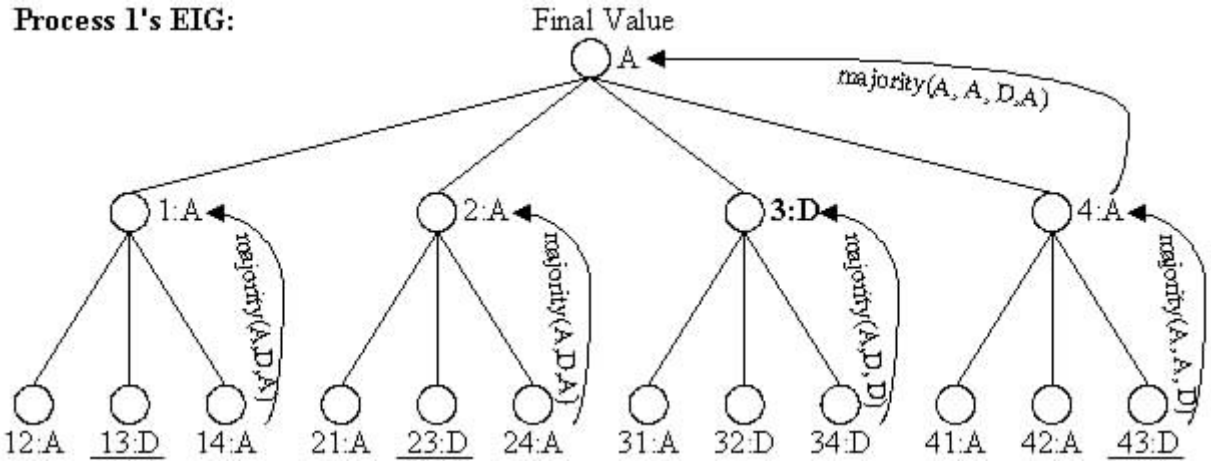
Determining the Decision from the Filled EIG:

At the end of round m+1, process i replaces any *Null* values in the tree with a default value $v_{default}$.

Then, process i changes the non-leaf values of the EIG working from level k up the tree to level 0 (the root) as follows: a non-leaf node value is set to the value held by a strict majority of all of its children nodes' values. If no strict majority value exists, then the node's value is set to the default value $v_{default}$.

Finally, when the root node's value is changed, it is the decision value for process i.

After running the decision algorithm on the filled EIG tree, process 1 decides to agree to attack.



The above algorithm solves the Byzantine-generals problem, but....

- it is very costly in terms of the number of messages -- $O(?)$ With signed messages the number of rounds in still m + 1, but number of messages is only $O(n^2)$.
- it relies on the system being "synchronous" so messages can take place in rounds so processes are entitled to timeout and assume that a faculty process has not sent a message within a round. Fischer et al ('85) proved that no system can "guarantee" consensus in an asynchronous system. (In an asynchronous system a crashed process is indistinguishable from a slow one since a process can be arbitrarily slow in responding.) This is an "impossibility result" meaning that consensus can be reached with a probability greater than zero.
- "synchronous" systems are not too practical.

Ways to handle this problem:

- consider "partially synchronous" systems having weaker assumptions than "synchronous systems". "Partially synchronous" systems (Dwork '88) are useful model for practical systems
- mask faults - processes store sufficient information in "stable storage" as necessary to be continue correctly after a crash and restart (e.g., transaction processsing systems)
- Consensus using "failure detectors" - failure detector signals processes if a process fails so it can be ignored. For example, processes might deem a process that has not responded for a long enough period of time as having failed and ignore subsequent messages from it. Chandra and Toueg ('96) worked on property of weak failure detectors.
- Consensus using "randomization" - introduction an element of chance in the correct processes' behavior to lessen the probability that adversarial (faulty) processes can defeat consensus. Processes cannot always reach consensus using this approach, but consensus can general be reached in finite expected time. (Canetti and Rabin '93)