Exponential Information Gathering

For reaching consensus - part 1

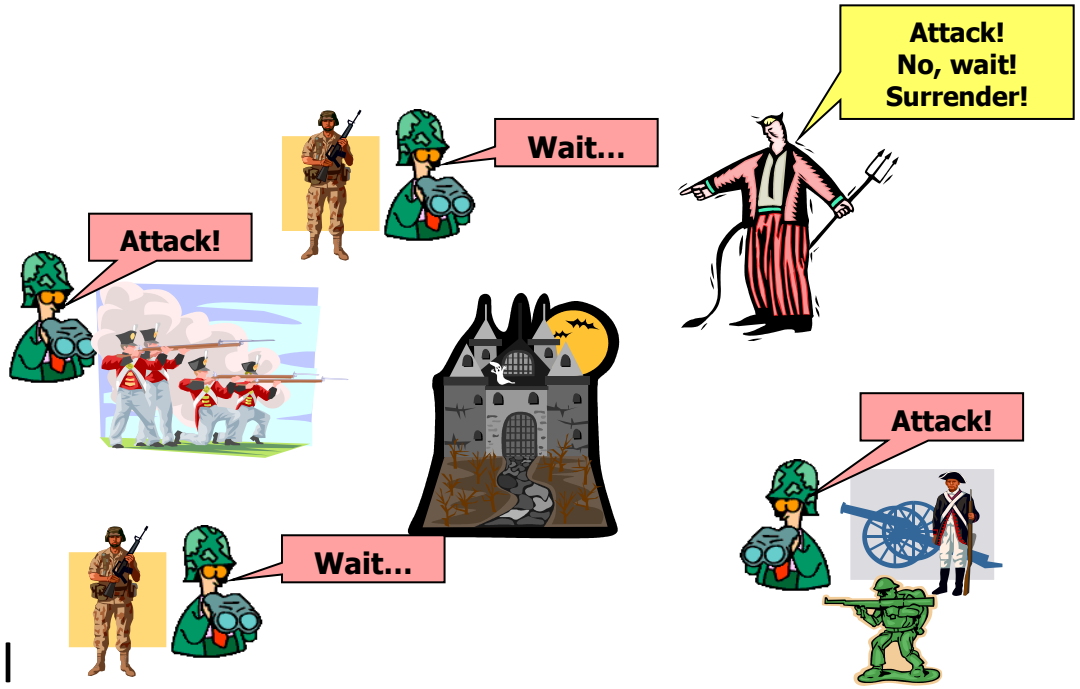
By: Pourandokht Behrouz

Exponential Information Gathering

- A strategy for consensus algorithms, which works for Byzantine agreement
- Based on EIG tree data structure
- By Byzantine we mean: “Byzantine Generals Problem” [Lamport, Pease, Shostak]
- EIG was introduced by Bar-Noy, D. Dolev (1987)
- The full name of their algorithm as they stated in their paper:
  - Exponential Information Gathering with Recursive Majority Voting
  - In short: Exponential Algorithm
- The algorithm works for both scenarios of the problem:
  - When there is a Leader who sends the initial values (Commander)
  - When each Processor has its own initial value (0 or 1)

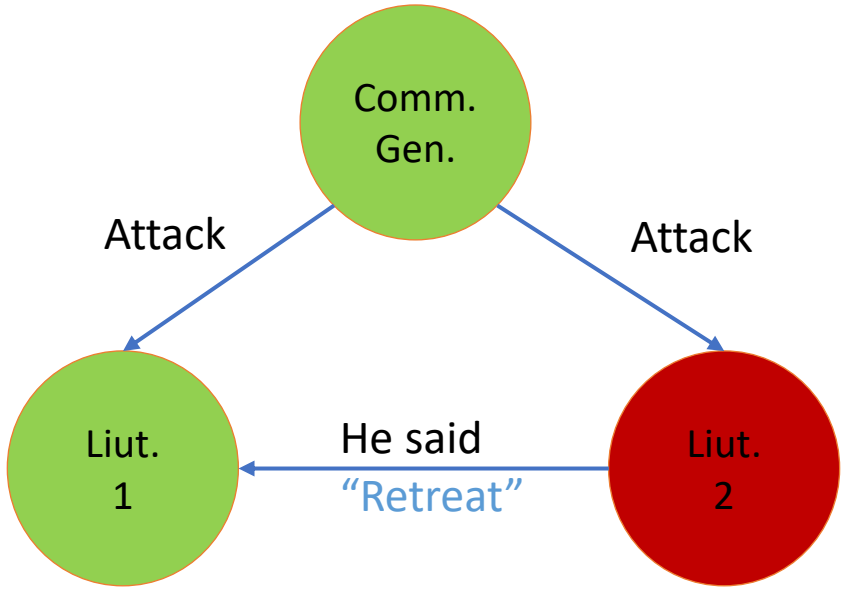
Byzantine Generals Problem

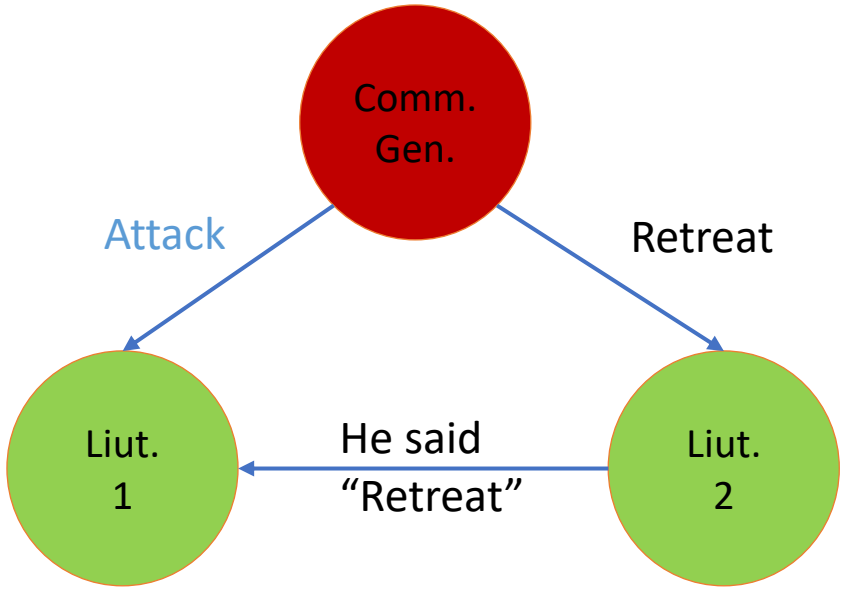
- Several division of the Byzantine army are camped outside an enemy city, commanded by a general
  - General = “processor”
  - Generals communicate by messenger
  - Use a network, i.e., distributed systems
- Some may be traitors
  - Fail, but in the worst possible ways
- All loyal generals agree on the same plan (attack or retreat)
- A small number of traitors cannot cause the loyal generals to adopt a bad plan



Byzantine Generals Problem

- Impossibility results :





- They proved: total number of processors n should be $n \geq 3t + 1$ where t is the number of faulty processors

Variations of the Problem

- Byzantine agreement problem: also referred to as broadcast problem, is when one out of n nodes (players) is the Commander general. And if he is non faulty every non-faulty(correct) node, should agree on his decision (value). [BDDS][MW]
- Distributed Consensus: when every node has their own initial value (decision) and all the correct player should come to consensus on the same decision which should be originated from one of the non faulty procesors. [BG1]
- EIG tree was originally introduced for Byzantine agreement (BA) variation [BDDS], but later used for Distributed Consensus as well.[BG1 , GM and more]

Formal problem statement

- We study the distributed consensus problem with the following assumptions:
  - Synchronous model, n processors
  - G = (V,E), undirected graph (bidirected edges)
  - Each processor has input 1 (attack) or 0 (don’t attack)
  - Some messages can be lost or altered.
  - All should eventually set decision output variables to 0 or 1

Correctness Conditions

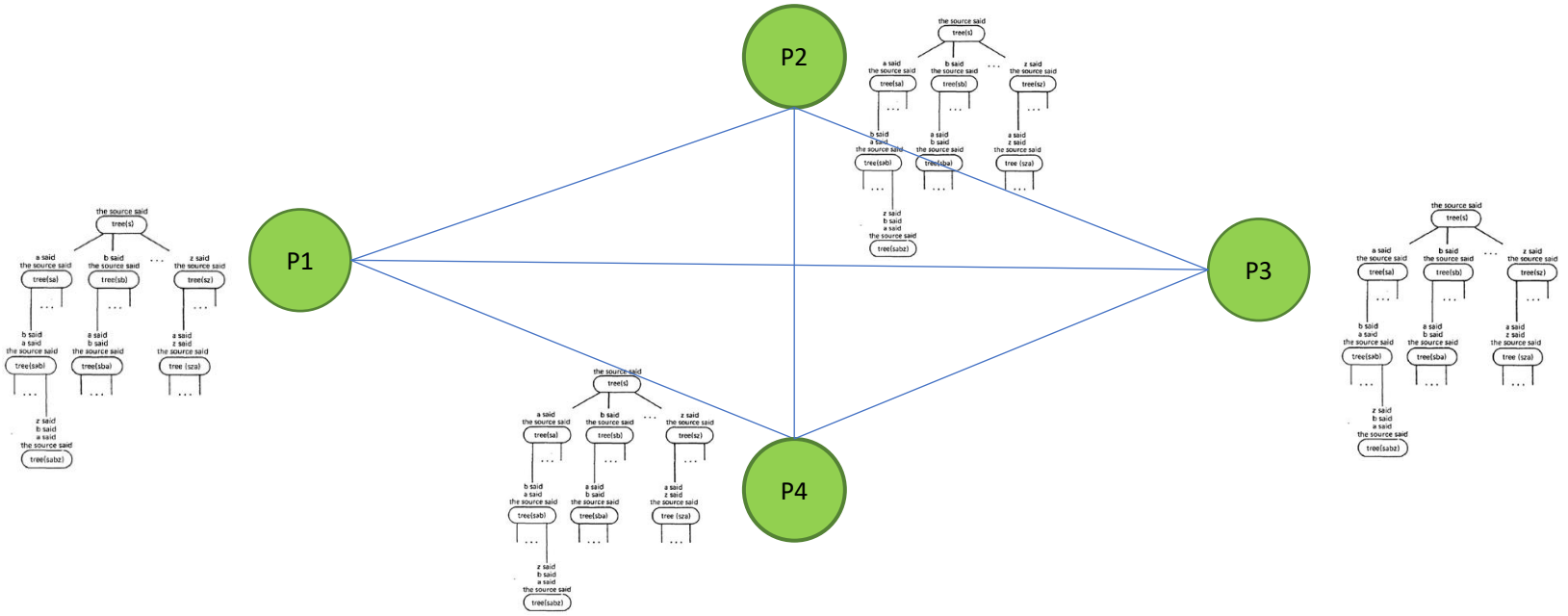
- Termination: every non faulty processor decides on an output (All nonfaulty processors eventually decide)
- Agreement: All non faulty processors decide on the same output (no two processor decide differently)
- Validity: if all processors begin with the same input, then the output is the same value:
    - If all start with 0, then 0 is the only allowed decision.
    - If all start with 1 and all messages are successfully delivered, then 1 is the only allowed decision
- Integrity: if a non faulty processor decides on y then y must have been proposed by some non-faulty processor (this makes sure the consensus result is not originated from an adversary)

Some Points to Consider

- Every algorithm used for BFT consensus should meet the correctness conditions stated
- EIG strategy with the recursive majority votes on the tree data structures at each node has been proved to meet the correctness conditions
- The optimal round for the algorithm is t + 1 (t is the number of faulty processors)
- Total number of players: $n \geq 3t + 1$
- The Exponential Algorithm reaches Byzantine agreement in t + 1 rounds tolerating $t < n / 3$ faults (the same for consensus)
- Time complexity of the Exponential Algorithm has been enhanced to polynomial, based on EIG but not using recursive majority voting (out of scope of this presentation)

The Algorithm in brief

- Processors send and relay values for number of rounds
- Recording the values they receive along various communication paths in a data structure called an EIG tree
- At the end, they use a commonly agreed-upon decision rule and decide based on the values recorded in their trees



About Exponential Algorithm

- Each processor maintains a large tree of height at most t (BA problem) and t+1 (Consensus) [BG1][GM].
- This difference is because in EIG protocol for BA, at first round commander sends his initial value to n - 1 nodes and does nothing else so in the rest of the rounds (t rounds) every other nodes communicate the values. The height of the tree at each node is t and the commander has a tree of height 1.
- In EIG for consensus case, all n nodes have their initial value and they communicate for t + 1 rounds so the height of the tree is t + 1
- In both cases we only need t + 1 rounds of exchanges

About Exponential Algorithm

- As we said each processor incrementally constructs a tree-based data structure which we will call it EIG tree.
- Root of the tree is of dept 0
- Inductively, the depth of a node is greater by one than the depth of its parent
- We are only interested in EIG trees of depth at most t + 1
- Each path from root to leaf contains t + 1 vertices
- In EIG tree, a node of depth d has n – d children. The root has n children

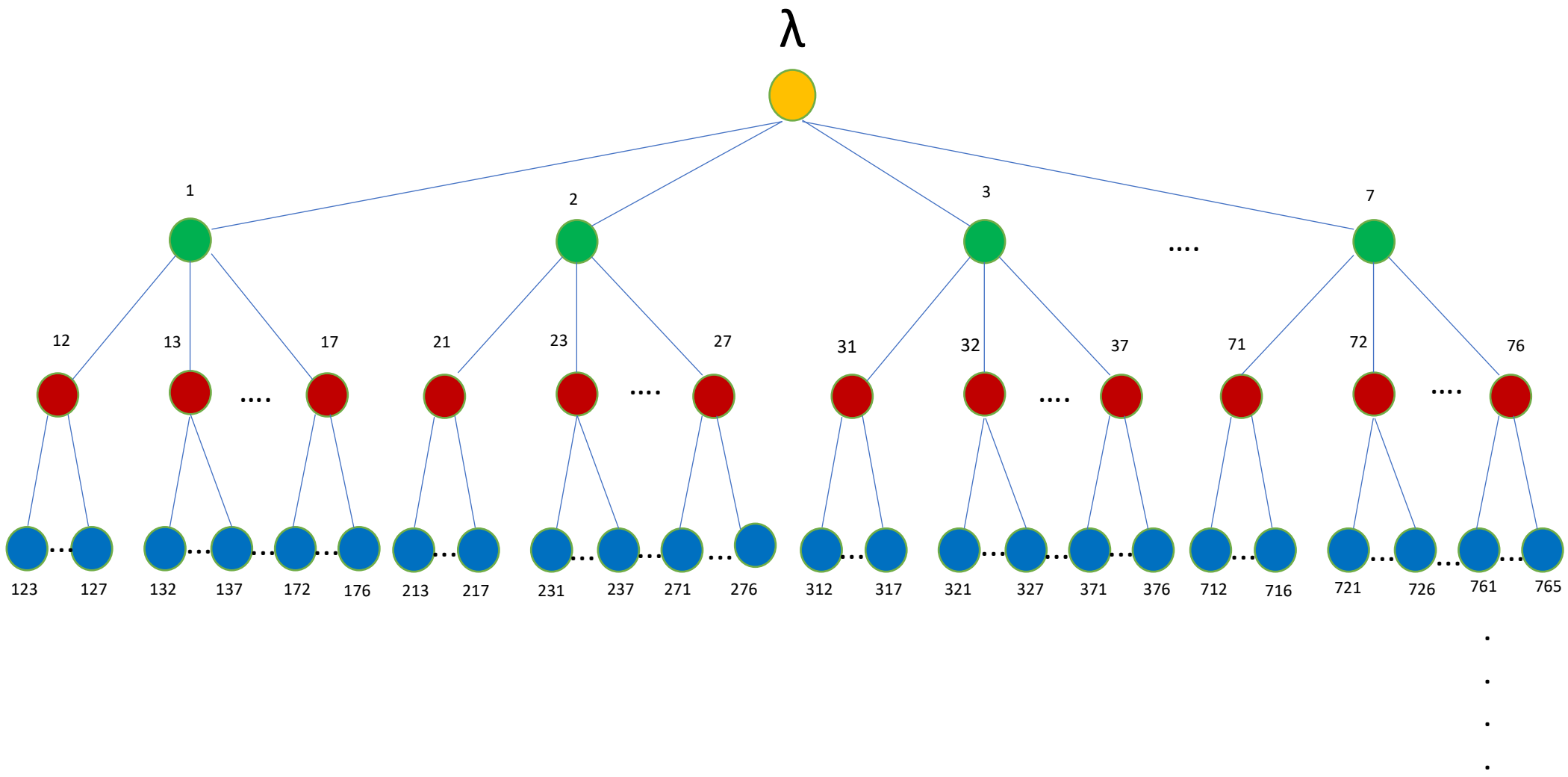
EIG tree node's Labeling

- The outgoing edges from the root are labelled 1 … n
- A node of depth d ≥ 1 has an edge labeled i for every processor name i that does not appear on the path leading from the root to the node
- Thus, no label appears twice on a path from the root to a leaf
- The sequence of labels on the path from the root to a given node uniquely determines this node or we can say:
  - Let σ be an internal node: for every processor p not labeling an ancestor of σ, σ has exactly one child labeled p thus no label appears twice in any path from root to leaf (tree is without repetition)

EIG tree node's Labeling

- For each internal node, the sequence of the labels encountered traversing from the root to that node, forms the node's label
- There is 1 – 1 correspondence between string σ of up to t + 1 distinct processors names and the nodes in an EIG tree of dept t + 1. So this is the name of the corresponding node σ
- Root is labeled λ for an empty string
- The size of this label (size of the sequence) is equal to the level of the tree which the node is at or
- The length of the string σ which will be denoted at |σ|, coincide with the depth of σ
- There are two values associated with each node σ in processors i's tree:
  - A stored value denoted by $tree(\sigma)$ and a resolved value denoted by $resolve(\sigma)$ or $res(\sigma)$.
  - To show this is the value of σ in tree i: $tree_i$ (σ) and $resolve_i$ (σ).
  - Stored values are assigned during t + 1 round of information exchange, resolve values are computed at the end in order to determine the decision value.

EIG tree example



Exponential Algorithm

- In case of consensus problem: At first round each processor sends his value to all other $n - 1 \geq 3t$ processors and each processor starts building its first level of the tree.
- In case of BA problem: At first round commander sends his initial value to $n - 1$ other nodes and sits back. Other nodes receive this value and build their tree storing this value at the root of their tree.
- At each remaining round, all processors add one level to their tree with the message they have received
- At each round all processors broadcast values they have in the current level of their tree
- A node is created for the tree in the round in which a value is stored in the node

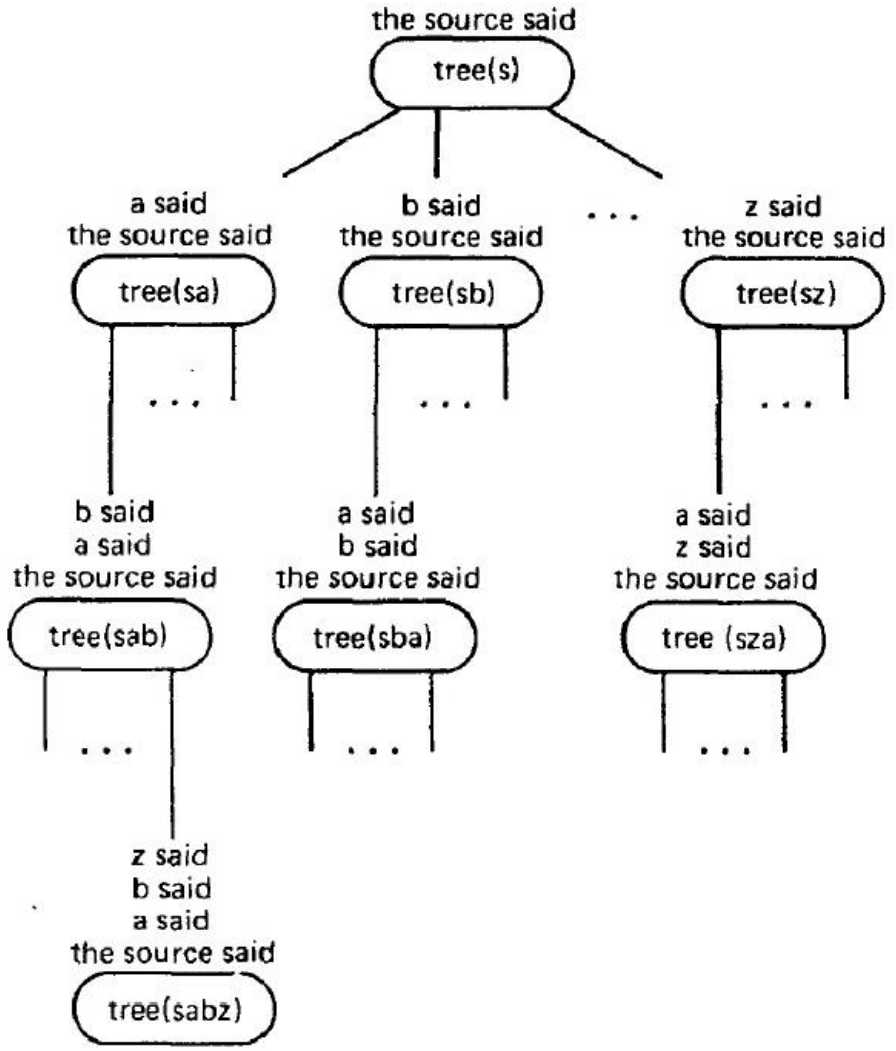
Exponential Algorithm

- The messages received are broken up to form a new level in the tree
- Notice that a single message is received by i from j in a given round and this reports on many values in j's tree and is used to update many different nodes in $tree_i$ and this message will not all be stored in a single node of the tree [GM]
- The information is gathered for t + 1 rounds until all nodes of dept t + 1 are created
- At that point each processor applies to the tree a recursive data reduction function, called resolve to obtain a new preferred value (this is done by preferring the majority value of the children at each node) and at last step resolve the root of the tree.

$$\mathbf{res}_i(\sigma) = \begin{cases} \mathbf{tree}_i(\sigma) & \text{if } \sigma \text{ is a leaf;} \\ 1 & \text{if majority of } \mathbf{res}_i(\sigma j) \text{ are 1;} \\ 0 & \text{otherwise.} \end{cases}$$

EIG Tree

- In the original paper by (Bar - Noy et al), the tree built at each processor looks like this:

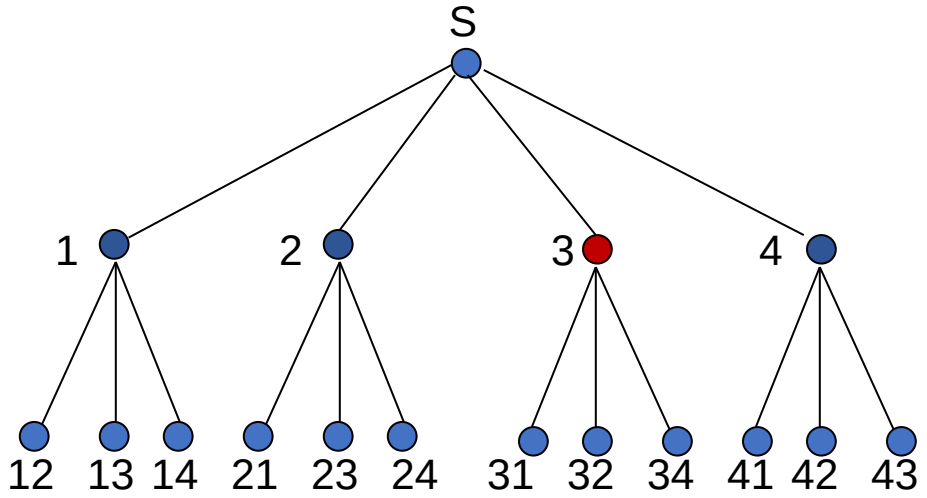


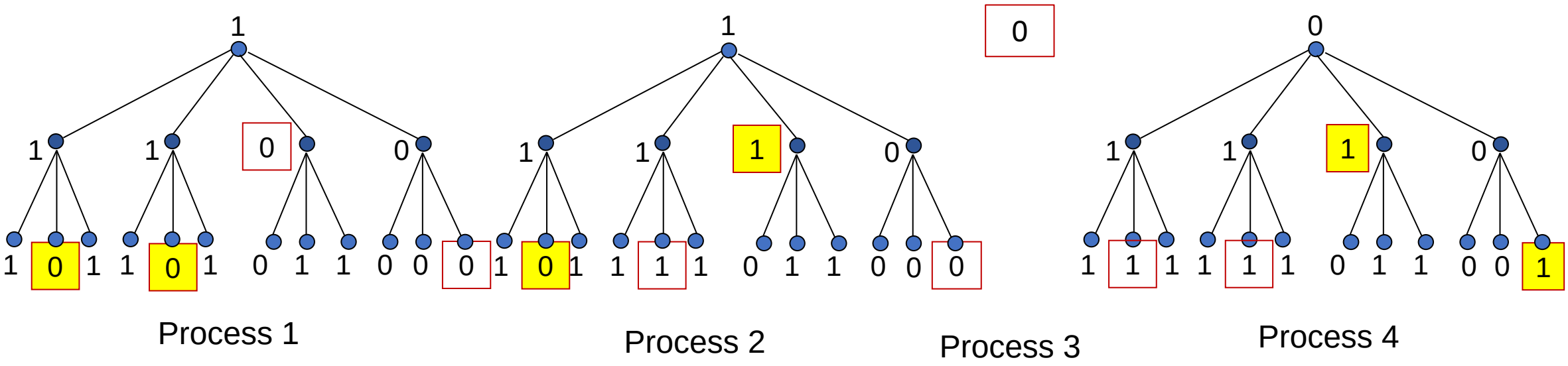
EIG algorithm for Byzantine agreement pseudo code

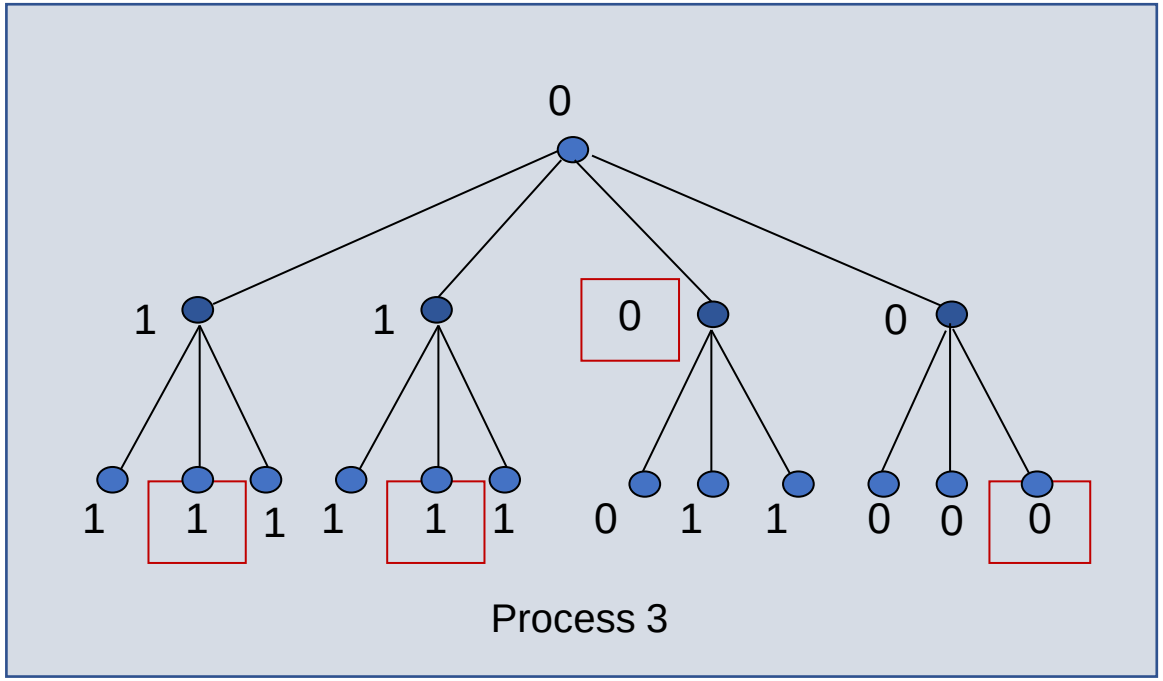
- Use EIG tree, construct the tree at each node
- Assume $n > 3t + 1$
- Relay messages for t+1 rounds
- Decorate the tree with values from V, replacing any garbage messages with default value v_0
- New decision rule:
  - Call the decorations val(x), where x is a node label
  - Redecorate the tree, defining newval(x)
    - Proceed bottom-up
    - Leaf: newval(x) = val(x)
    - Non-leaf: newval(x) =
      - newval of strict majority of children in the tree, if majority exists,
      - v_0 otherwise
  - Final decision: newval(S) (newval at root)

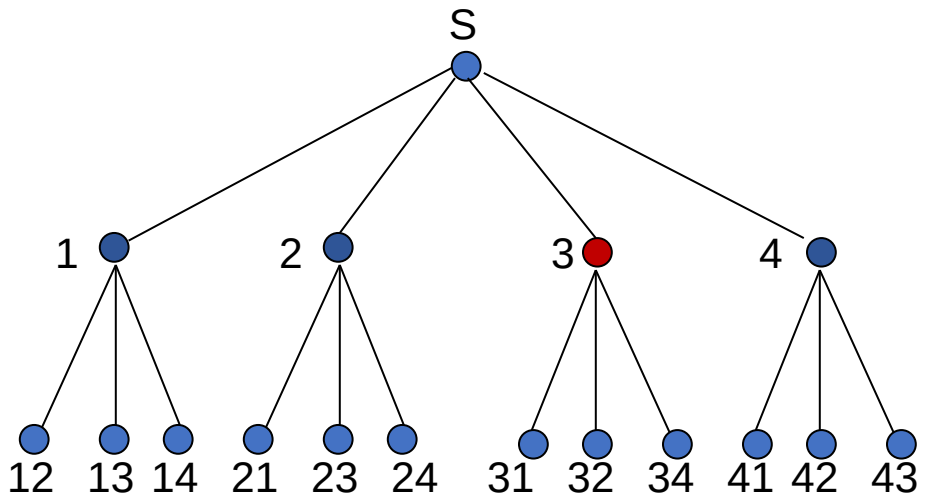
Example: n = 4 , t = 1

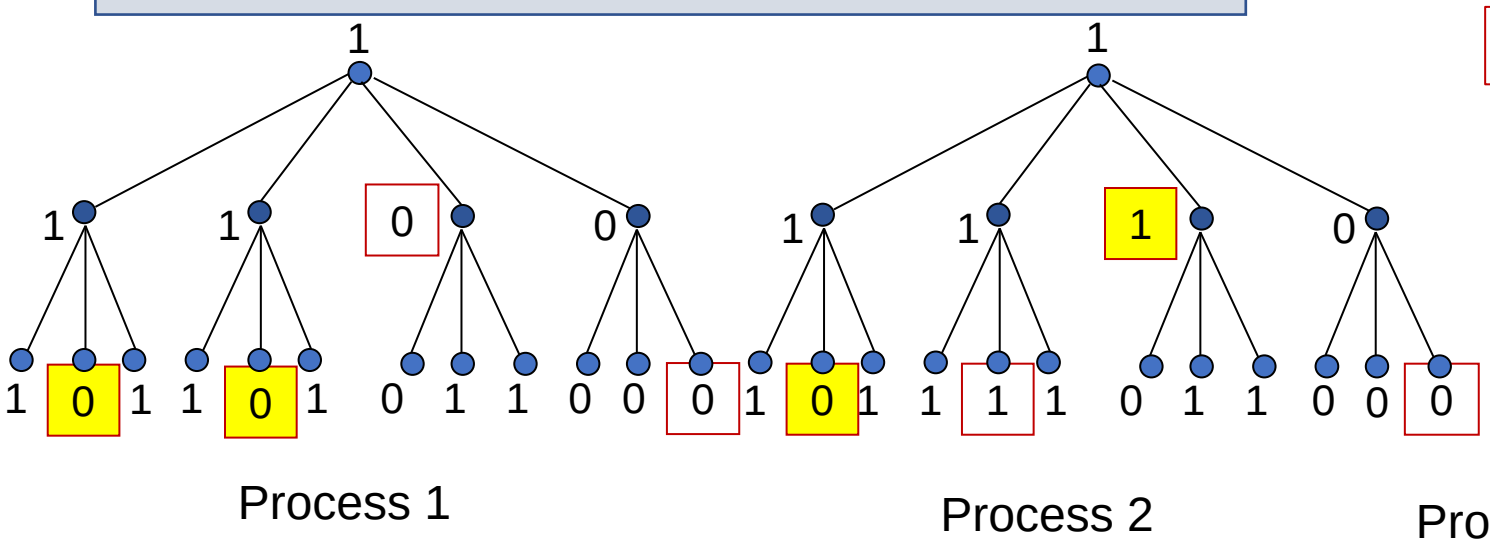
- Four non faulty processors and P_3 is faulty
- Initial values: $P_1 = 1$, $P_2 = 1$, $P_3 = 0$, $P_4 = 0$
- Round one: P_3 lies to P_2 and P_4
- Round two
- P_3: Lies





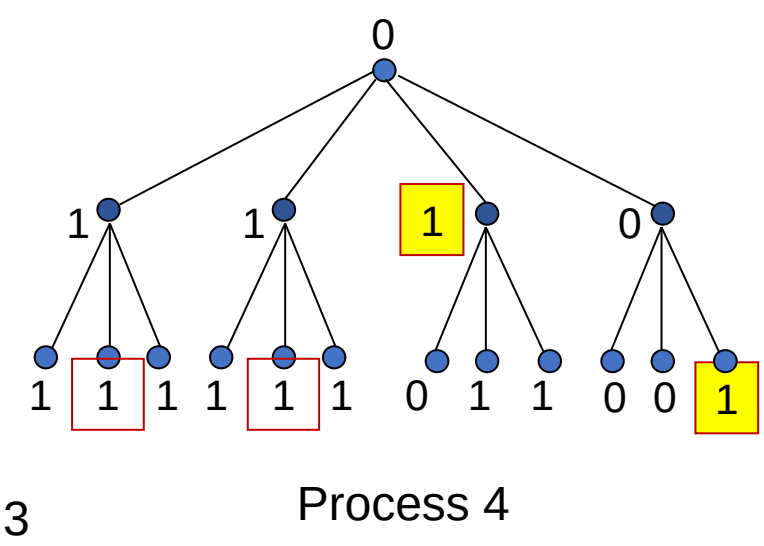

0
1 1 0 0
1 1 1 1 1 1 0 1 1 0 0 0
Process 3


S
1 2 3 4
12 13 14 21 23 24 31 32 34 41 42 43


1
1 1 0 0
1 0 1 1 0 1 0 1 1 0 0 0
Process 1

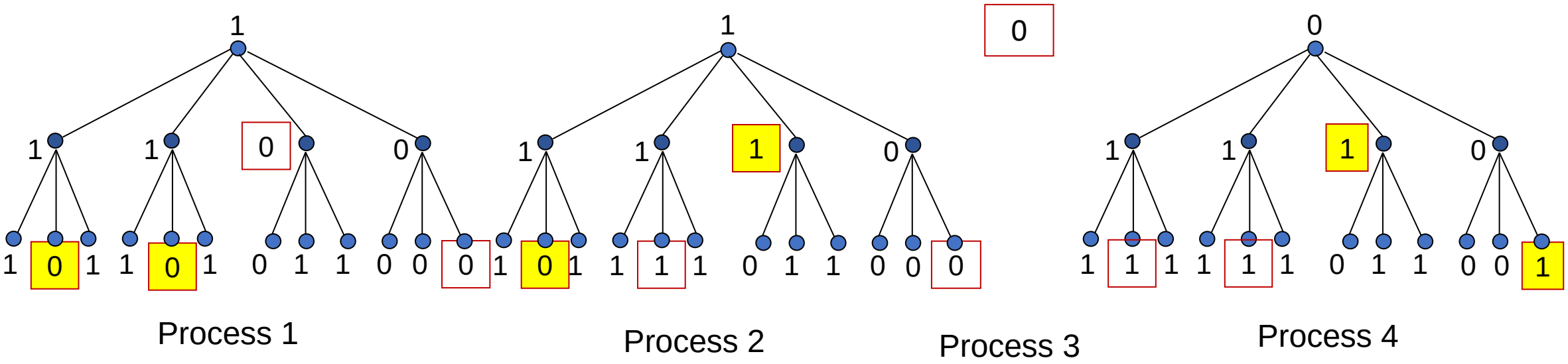
1
1 1 1 0
1 0 1 1 1 1 0 1 1 0 0 0
Process 2

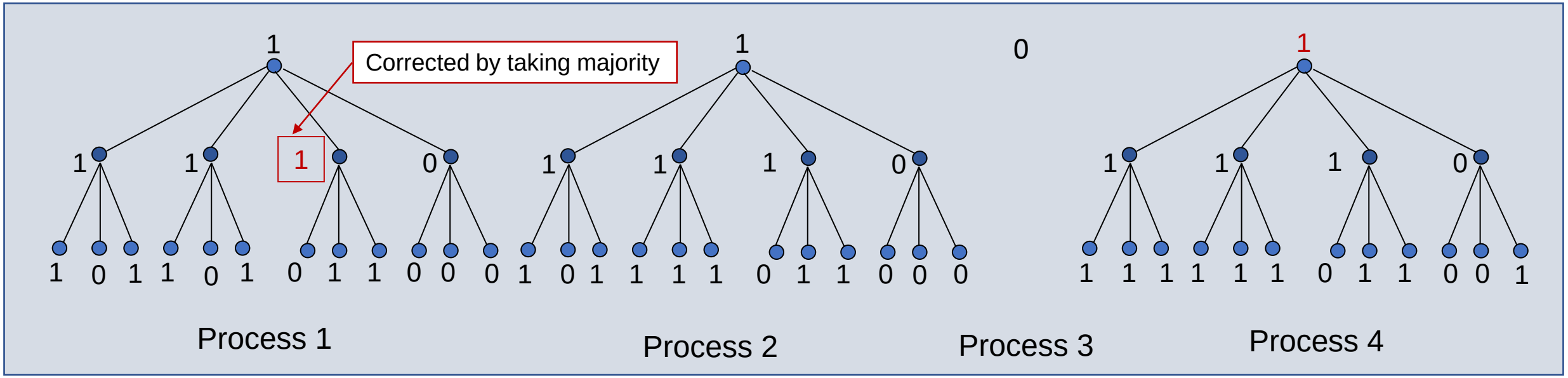

0
Process 3

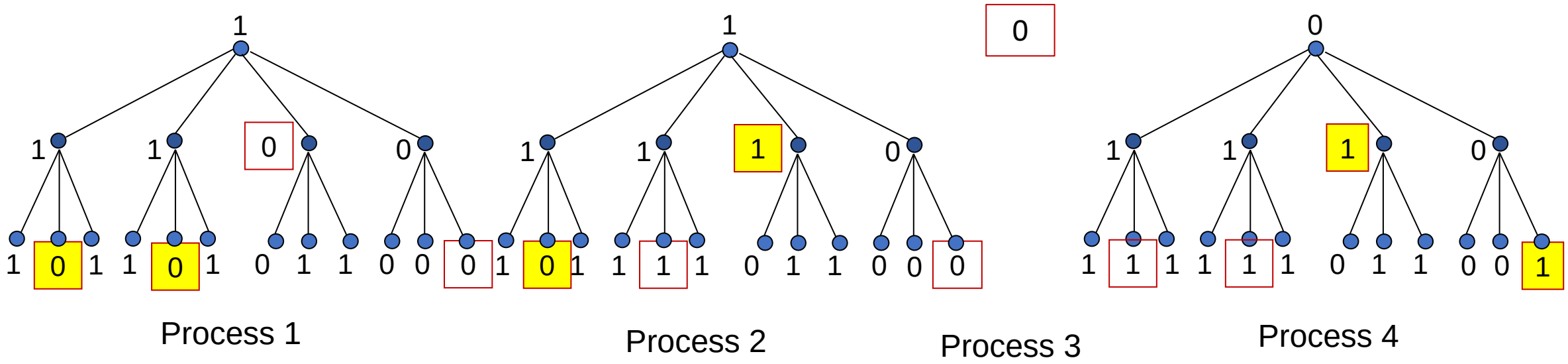

0
1 1 1 0
1 1 1 1 1 1 0 1 1 0 0 1
Process 4

Example: n = 4, t = 1

- We saw t + 1 rounds (two rounds) of information exchange
- Each processor has built its own tree
- Now calculate newvals, bottom-up, choosing majority values (v_0 = 0 if no majority)




Corrected by taking majority
Process 1
Process 2
Process 3
Process 4

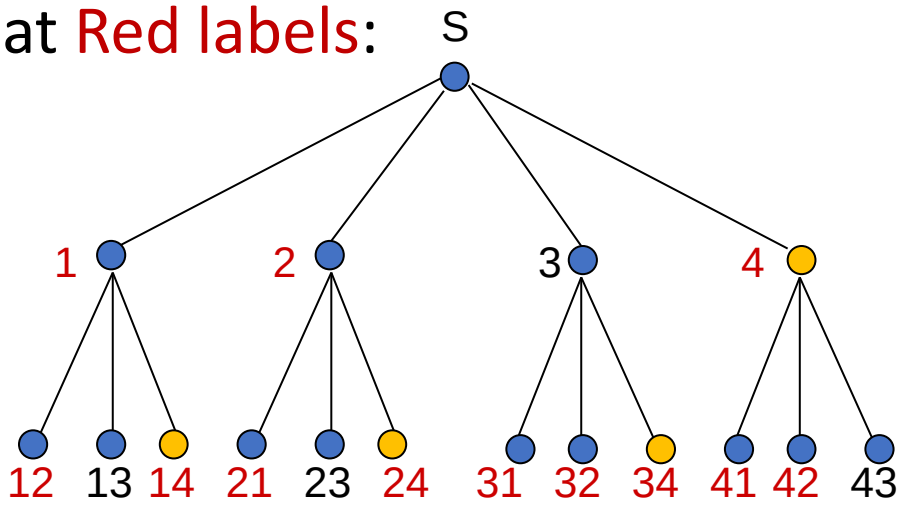

Process 1
Process 2
Process 3
Process 4

Proof of correctness for the Algorithm

- We first prove two Lemmas to be used in the next steps
- Then we check on the correctness conditions given for any BFT algorithm
- We prove that these conditions hold (using the two lemmas)

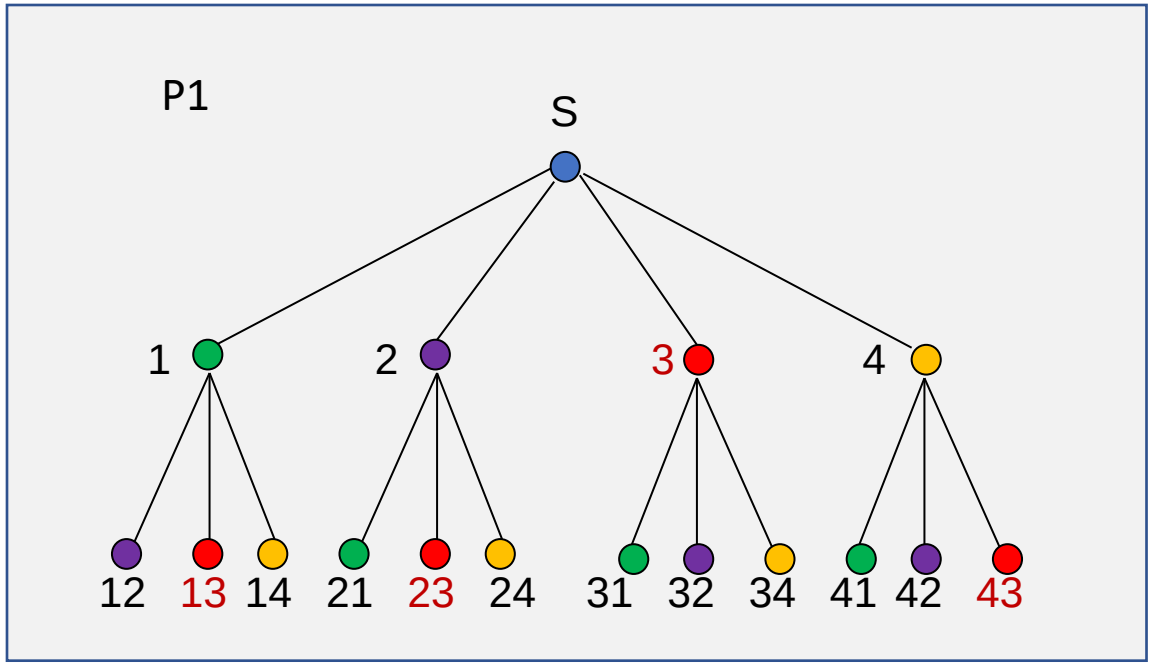
Proof of correctness

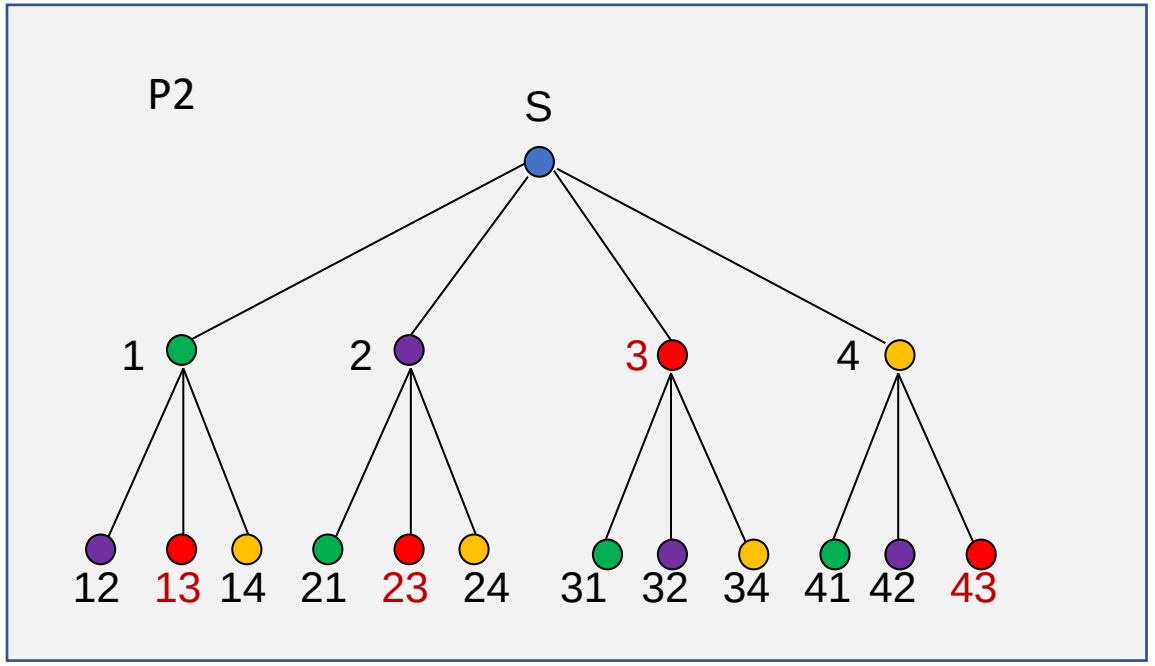
- Lemma 1: If i, j, k are nonfaulty, then $val(x)_i = val(x)_j$ for every node label x ending with k
- Tree of the previous example (P3 is Faulty) look at Red labels:
- The tree at nonfaulty processor, P1 (i = 1):
- The same values are found in the tree of P2 (j = 2) for the yellow nodes.
- Here P4 sends to P1,P2,P3 (k = 4)



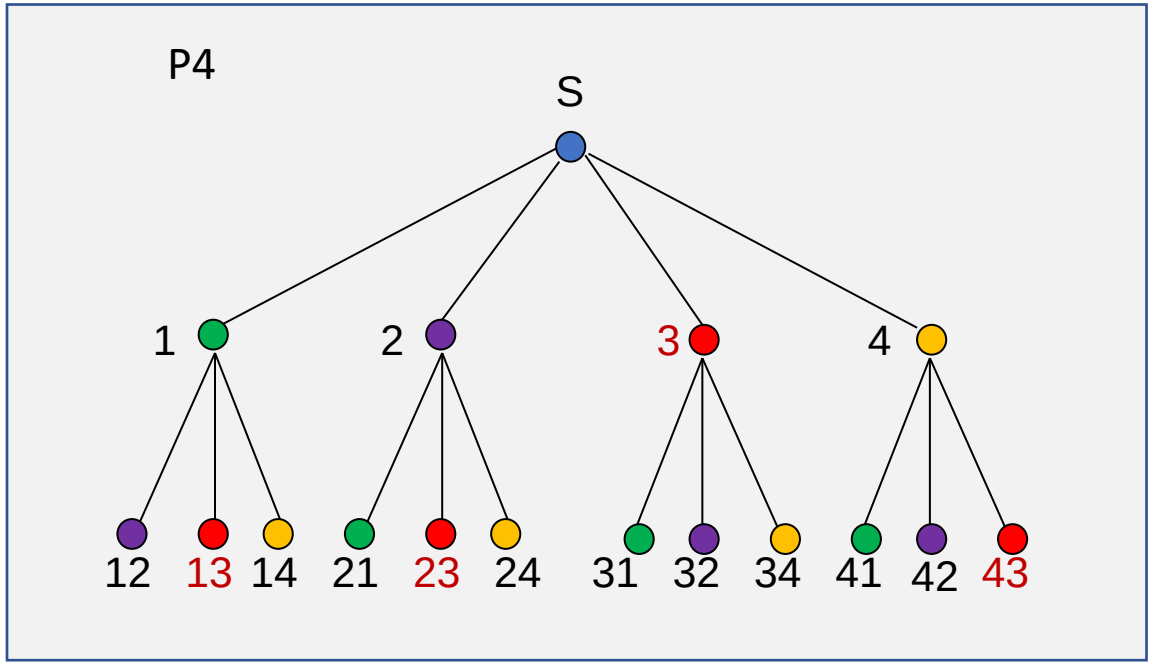
P1

- Proof: k sends same messages to i and j and they fill their trees accordingly





- All yellow nodes have the same value in all P1, P2, P4 trees
- All Green nodes have the same value in all P1, P2, P4 trees
- All purple nodes have the same value in all P1, P2, P4 trees
- The Red nodes belongs to the faulty P3 so the values depends on his lies.



Proof of correctness

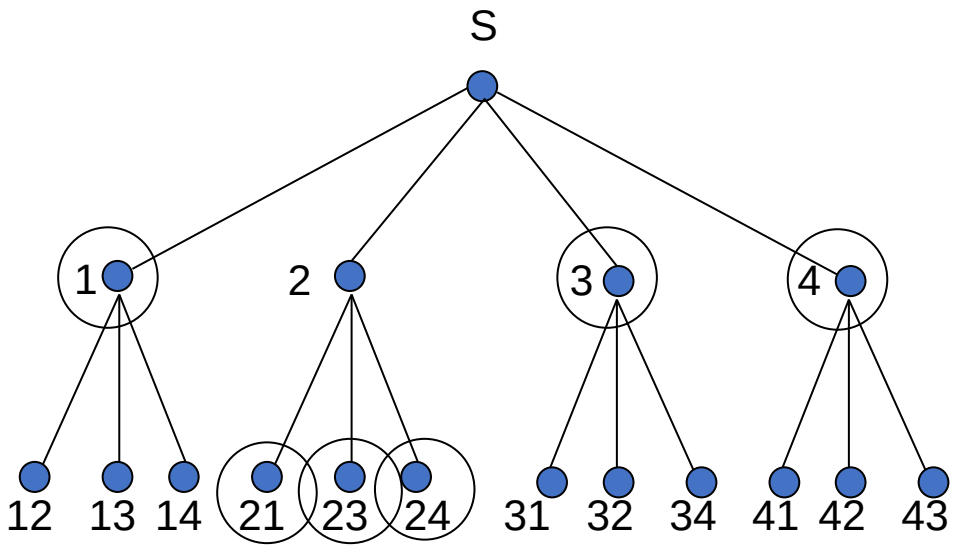
- Lemma 2: If x ends with nonfaulty process index then $\exists v \in V$ such that $val(x)_i = newval(x)_i = v$ for every nonfaulty i.
- Proof: by Induction on lengths of labels, bottom up
  - Basis: Leaf
    - Lemma 1 implies that all nonfaulty processes have same val(x) (say x= 21, all processors has same value at this node)
    - newval = val for each leaf
  - Inductive step: $|x| = r \leq t$ ($|x| = t+1$ at leaves, $|21| = 2$ so its at level 2 or the leaf level)
    - Lemma 1 implies that all nonfaulty processes have same val(x), say v.
    - We need newval(x) = v everywhere also
    - Every nonfaulty process j broadcasts same v for x at round r+1, so $val(xj)_i = v$ for every nonfaulty j and I
    - By inductive hypothesis, also $newval(xj)_l = v$ for every nonfaulty j and I
    - A majority of labels of x's children end with nonfaulty process indices:
      - Number of children of node x is $\geq n - t > 3t - t = 2t$
      - At most t are faulty
    - So, majority rule applied by i leads to $newval(x)_i = v$, for all nonfaulty i

Correctness Conditions

- Validity: if all processors begin with the same input, then the output is the same value
  - If all nonfaulty processes begin with v, then all nonfaulty processes broadcast v at round 1, so $val(j)_i = v$ for all nonfaulty i, j
  - By Lemma 2, also $newval(j)_i = v$ for all nonfaulty i,j
  - Majority rule implies $newval(S)_i = v$ for all nonfaulty i
  - So all nonfaulty i decide v
- Termination: All nonfaulty processors eventually decide
  - Obvious
- Agreement: All non faulty processors decide on the same output
  - Needs the concept of path covering and common node (next slides)

Agreement condition

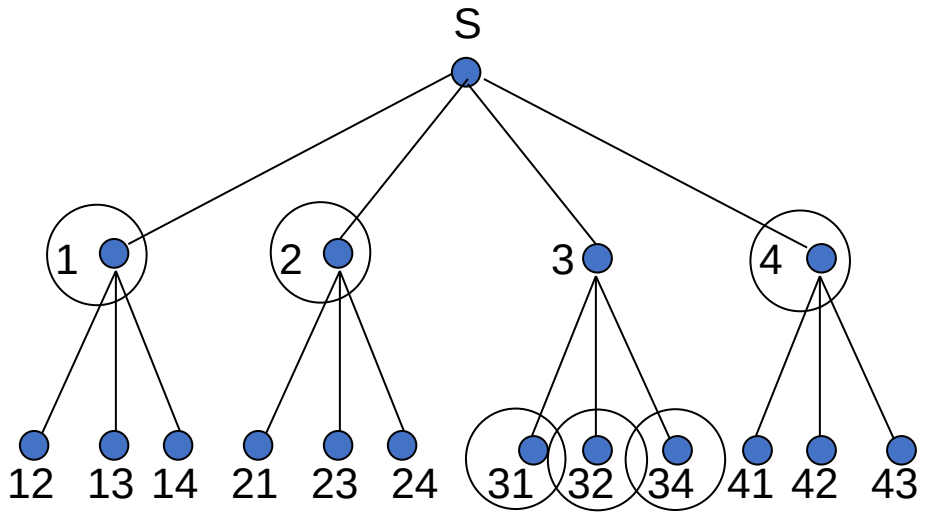
- Path covering: Subset of nodes containing at least one node on each path from root to leaf



- Common node: One for which all nonfaulty processes have the same newval
  - If label ends in nonfaulty process index, Lemma 2 implies it's common
  - Might be others too

Agreement condition

- Lemma 3: There exists a path covering all of whose nodes are common.



- Proof:
  - Let C = nodes with labels of the form xj, j nonfaulty.
  - By Lemma 2, all of these are common.
  - Claim these form a path covering:
    - There are at most t faulty processes.
    - Each path contains t+1 labels ending with t+1 distinct indices.
    - So at least one of these labels ends with a nonfaulty process index

Agreement condition

- Lemma 4: If there's a common path covering of the subtree rooted at any node x, then x is common
- Proof:
  - By induction, from the leaves up,"Common-ness" propagates upward
- Lemmas 3 and 4 together imply that the root is common So:
  - All nonfaulty processors get the same newval(S)
  - Results in Agreement

Complexity of the Algorithm

- Using EIG Algorithm we have:
  - Time: t + 1 (number of rounds)
  - Communication: $O(n^{t+1})$ which is exponential
    - Despite the simplicity of the algorithm, the message size and the amount of local computation required grow exponentially with t
  - using other methods it has been improved in later algorithms to polynomial
- It also requires n > 3t processors

References

- [PSL] Pease, M., Shostak, R., and Lamport, L., “Reaching Agreement in the presence of fault” JACM 27(2) (1980), pp. 228-234.
- [BDDS] A.Bar-Noy, D. Dolev, C. Dwork, and H. R. Strong, Shifting Gears: Changing Algorithms on the Fly to Expedite Byzantine Agreement, “Proceedings of the sixth Annual ACM Symposium on Principles of Distributed Computing, 1987,” pp. 42-51.
- [MW] Y. Moses, O. Waarts, “coordinated traversal: (t+1) Round Byzantine Agreement in polynomial Time” Journal of Algorithms 17, 1994, pp.110-156 (received 1988, Revised 1992)
- [BG1] P. Berman, A. Garay, “Cloture Vote: n/4 Resilient Distributed Consensus in t+1 rounds. Preliminary version appeared as part of “Toward optimal Distributed Consensus” in Proc. 30th IEEE symposium. On Foundation of computer science [BGP] and it is a part of Garay’s P.h.D dissertation.
- [GM] J. A. Garay, Y. Moses, “Fully Polynomial Byzantine Agreement in t + 1 rounds – Extended Abstract” Proc. 25th ACM Symposium on Theory of Computing (STOC 1993), acm press, pp. 31-41, 1993.

Thank You