Distributed Consensus

He Sun
School of Informatics
University of Edinburgh

THE UNIVERSITY of EDINBURGH

Outline

- Fault-tolerant consensus in synchronous systems
- Link failures:
  - The Two Generals Problem
- Process failures:
  - Stopping and Byzantine failure models
  - Algorithms for agreement with stopping and Byzantine failures
  - Exponential information gathering

Distributed Consensus

- Abstract problem of reaching agreement among processes in a distributed system, when they all start with their own “opinions”.
- Complications: Failures (process, link); timing uncertainties.
- Motivation:
  - Database transactions: Commit or abort
  - Aircraft control:
    - Agree on which plane should go up/down, in resolving encounters (TCAS)
  - Resource allocation: Agree on who gets priority for obtaining a resource, doing the next database update, etc.
- Fundamental problem
- We’ll revisit it several times:
  - With link failures, processor failures.
  - Algorithms, impossibility results.

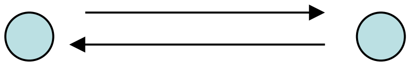
Formal Problem Statement

- $G = (V, E)$, undirected graph (bidirected edges)
- Synchronous model, n processes
- Each process has input 1 (attack) or 0 (don't attack).
- Any subset of the messages can be lost.
- All should eventually set decision output variables to 0 or 1.
- Correctness conditions:
  - ***Agreement:*** No two processes decide differently.
  - ***Validity:***
    - 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.

Alternatively…

- ***Stronger validity condition:***
  - *If anyone starts 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.
- Guidelines:
  - For designing algorithms, try to use stronger correctness conditions (***better algorithm***).
  - For impossibility results, use weaker conditions (***better impossibility result***).

Impossibility Result for 2-Vertex Graph

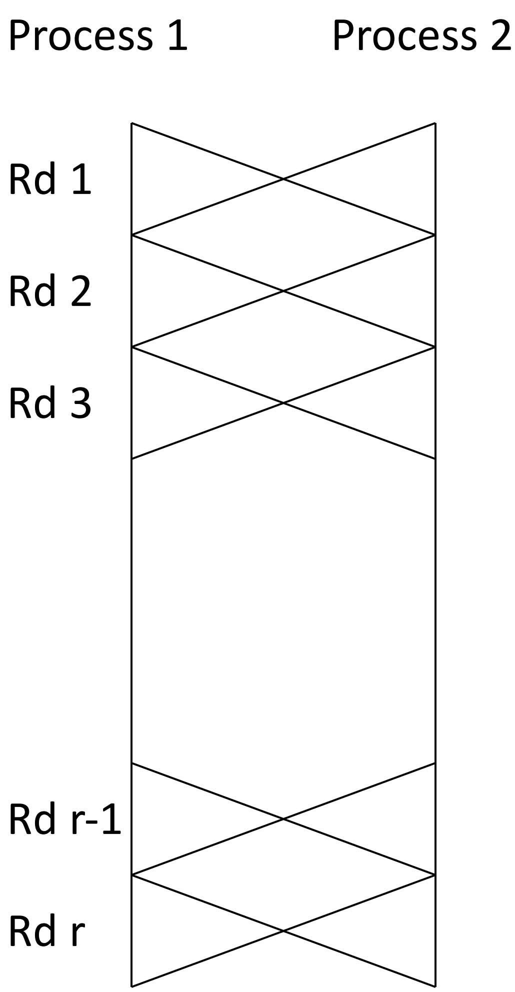


Proof: By contradiction.

- Suppose we have a solution---a process (states, transitions) for each index 1, 2.
- Assume WLOG that both processes send messages at every round. *Could add dummy messages.*
- Proof based on limitations of local knowledge.
- Start with α, the execution where both start with 1 and all messages are received.
  - By termination condition, both eventually decide.
  - Say, by r rounds.
  - By validity, both decide on 1.

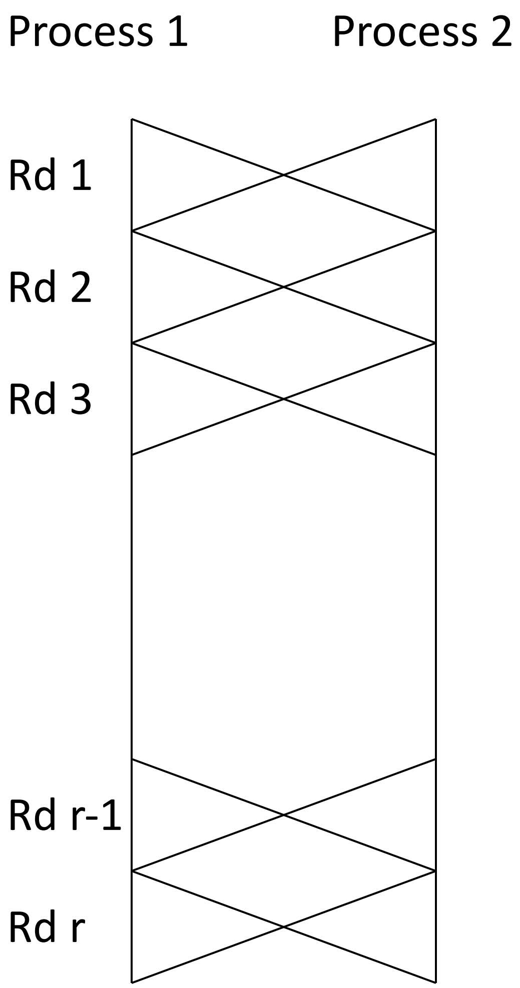
Impossibility Result for 2-Vertex Graph

- α_1: Same as α, but lose all messages after round r.
  - Doesn't matter, since they've already decided by round r.
  - So, both decide 1 in α_1.
- α_2: Same as α_1, but lose the last message from process 1 to process 2.
  - Claim α_1 indistinguishable from α_2 by process 1, denoted by $\boldsymbol{\alpha_1 \sim^1 \alpha_2}$.
  - Formally, 1 sees the same sequence of states, incoming and outgoing messages.
  - So process 1 also decides 1 in α_2.
  - By agreement, process 2 decides 1 in α_2.



Continuing

- α_3: Same as α_2, but lose the last message from process 2 to process 1.
  - Then $\alpha_2 \sim^2 \alpha_3$.
  - So process 2 decides 1 in α_3.
  - By agreement, process 1 decides 1 in α_3.
- α_4 : Same as α_3, but lose the last message from process 1 to process 2.
  - Then $\alpha_3 \sim^1 \alpha_4$.
  - So process 1 decides 1 in α_4.
  - So process 2 decides 1 in α_4.
- Keep removing edges, get to:



The Contradiction

- α_{2r+1} : Both start with 1, no messages received.
  - Still both must eventually decide 1.
- α_{2r+2} : process 1 starts with 1, process 2 starts with 0, no messages received.
  - Then $\alpha_{2r+1} \sim^1 \alpha_{2r+2}$.
  - So process 1 decides 1 in α_{2r+2}.
  - So process 2 decides 1 in α_{2r+2}.
- α_{2r+3} : Both start with 0, no messages received.
  - Then $\alpha_{2r+2} \sim^2 \alpha_{2r+3}$.
  - So process 2 decides 1 in α_{2r+3}.
  - So process 1 decides 1 in α_{2r+3}.
- But α_{2r+3} contradicts weak validity!

Consensus with Process Failure

- Stopping failures (crashes) and Byzantine failures (arbitrary processor malfunction, possibly malicious)
- Agreement problem:
  - n-node connected, undirected graph, known to all processes.
  - Input v from a set V, in some state variable.
  - Output v from V, by setting decision $:= v$.
  - Bounded number $\leq f$ of processors may fail.
- Bounded number of failures:
  - A typical way of describing limited amounts of failure.
  - Alternatives: Bounded rate of failure; probabilistic.

Stopping Agreement

- Assume process may stop working at any point:
  - Between rounds.
  - While sending messages at a round; any subset of intended messages may be delivered.
- Correctness conditions:
  - ***Agreement***: No two processes decide on different values.
    - "Uniform agreement"
  - ***Validity***: If all processes start with the same v, then v is the only allowable decision.
  - ***Termination***: All nonfaulty processes eventually decide.
- Alternatively:
  - ***Stronger validity condition***: Every decision value must be some process' initial value.

Byzantine Agreement

- "Byzantine Generals Problem"
  - Originally "Albanian Generals"
- Faulty processes may exhibit "arbitrary behavior":
  - Can start in arbitrary states, send arbitrary messages, perform arbitrary transitions.
  - But can't affect anyone else's state or outgoing messages.
  - Often called "malicious" (but they aren't necessarily).
- Correctness conditions:
  - ***Agreement***: No two nonfaulty processes decide on different values.
  - ***Validity***: If all nonfaulty processes start with the same v, then v is the only allowable decision for nonfaulty processes.
  - ***Termination***: All nonfaulty processes eventually decide.

Technicality about stopping vs. Byzantine agreement

- A Byzantine agreement algorithm doesn’t necessarily solve stopping agreement.
- For stopping, all processes that decide, even ones that later fail, must agree (uniformity condition).
- Too strong for Byzantine setting.

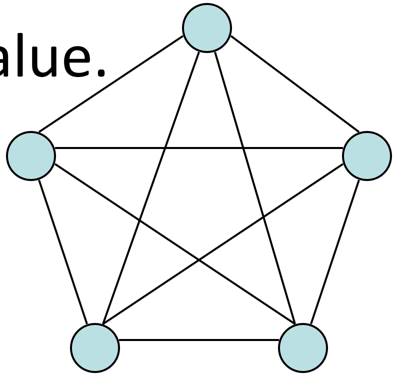
Complexity Measures

- **Time**: Number of rounds until all nonfaulty processes decide.

- **Communication**: Number of messages, or number of bits.
  - For Byzantine case, just count those sent by nonfaulty processes.

Simple Algorithm for Stopping Agreement

- Assume complete n-node graph.
- Idea:
  - Processes keep sending all V values they've ever seen.
  - Use simple decision rule at the end.
- In more detail:
  - Process i maintains $\mathrm{W} \subseteq V$, initially containing just i's initial value.
  - Repeatedly: Broadcast W, and add received elements to W.
  - After k rounds:
    - If $|W| = 1$ then decide on the unique value.
    - Else decide on default value $v_0 \in V$.
- **Question:** How many rounds?



How many rounds?

- Depends on number f of failures to be tolerated.
- $f = 0$:
  - $k = 1$ is enough.
  - All get same W.
- $f = 1$:
  - $k = 1$ doesn't work:
    - Say process 1 has initial value u, others have initial value v.
    - Process 1 fails during round 1, sends to some and not others.
    - So some have $W = \{v\}$, others $\{u, v\}$, may decide differently.
  - $k = 2$ does work:
    - If someone fails in round 1, then no one does in round 2.
- General f:
    - $k = f + 1$

Correctness Proof (for $k = f + 1$)

- **Claim 1**: Suppose $1 \leq r \leq f + 1$ and no process fails during round r. Let i and j be two processes that haven't failed by the end of round r. Then $W_i = W_j$ right after round r.
- Proof: Each gets exactly the union of all the W's of the non-failed processes at the beginning of round r.
- "Clean round"---allows everyone to resolve their differences.

- **Claim 2**: Suppose W sets are identical just after round r, for all processes that are still non-failed. Then the same is true for any $r' > r$.
- Proof: Obvious.

Checking Correctness Conditions

- ***Agreement***:
  - $\exists$ round r, $1 \leq r \leq f + 1$, at which no process fails (since $\leq f$ failures).
  - Claim 1 says all that haven't yet failed have same W after round r.
  - Claim 2 implies that all have same W after round $f + 1$.
  - So nonfaulty processes pick the same value.
- ***Validity:***
  - If everyone starts with v, then v is the only value that anyone ever gets, so $|W| = 1$ and v will be chosen.
- ***Termination:***
  - Obvious from decision rule.

Complexity Bounds

- Time: $f + 1$ rounds
- Communication:
  - Messages: $\leq (f \ + \ 1)\, n^2$
  - Message bits: Multiply by $n\ b$

Number of values sent in a message

A fixed bound on number of bits to represent a value in V.

- Can improve communication:
  - Messages: $\leq 2\, n^2$
  - Message bits: Multiply by b

Improved algorithm (Opt)

- Each process broadcasts its own value in round 1.
- May broadcast at one other round, just after it first hears of some value different from its own.
- In that case, it chooses just one such value to rebroadcast.
- After $f + 1$ rounds:
  - If $|W| = 1$ then decide on the unique value.
  - Else decide on default value v_0.

Correctness

- Relate behavior of Opt to that of the original algorithm.
- Specifically, relate executions of both algorithms with the same inputs and same failure pattern.
- Let O denote the W set in the optimized algorithm.
- Relation between states of the two algorithms:
  - For every vertex i:
    - $O_i \subseteq W_i$.
    - If $|W_i| = 1$ then $O_i = W_i$.
    - If $|W_i| > 1$ then $|O_i| > 1$.

Not necessarily the same set, but both > 1.

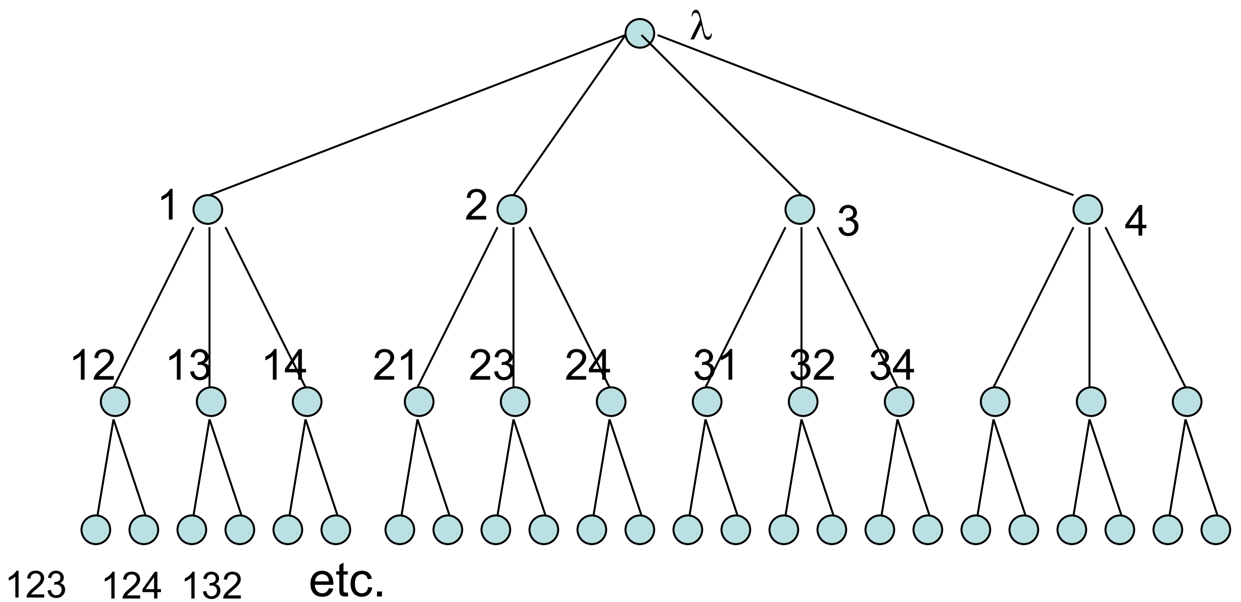
- Relation after $f + 1$ rounds implies same decisions.

Proof of Correctness

- Induction on number of rounds
- Key ideas:
  - $O_i \subseteq W_i$
    - Obvious, since Opt just suppresses sending of some messages from Unopt.
  - If $|W_i| = 1$ then $O_i = W_i$.
    - Nothing suppressed in this case.
    - Actually, follows from the first property and the fact that O_i is always nonempty.
  - If $|W_i| > 1$ then $|O_i| > 1$.
    - Inductive step, for some round r:
    - If in Unopt, i receives messages only from processes with $|W| = 1$, then in Opt, it receives the same sets. So after round r, $O_i = W_i$
    - Otherwise, in Unopt, i receives a message from some process j with $|W_j| > 1$. Then after round r, $|W_i| > 1$ and $|O_i| > 1$.

Exponential Information Gathering (EIG)

- A strategy for consensus algorithms, which works for Byzantine agreement as well as stopping agreement.
- Based on EIG tree data structure.
- EIG tree $T_{n,f}$, for n processes, f failures:
  - $f+2$ levels
  - Paths from root to leaf correspond to strings of $f+1$ distinct process names.
- Example: $T_{4,2}$

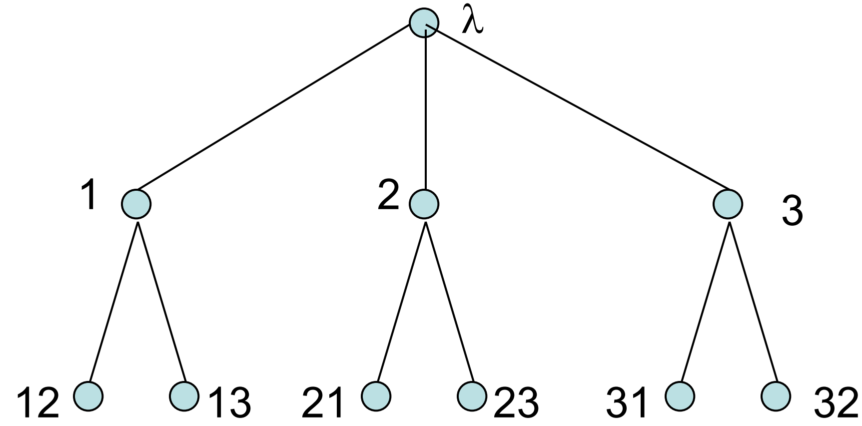


EIG Stopping Agreement Algorithm

- Each process i uses the same EIG tree, $T_{n,f}$.
- Decorates nodes of the tree with values in V, level by level.
- Initially: Decorate root with i's input value.
- Round $r \geq 1$:
  - Send all level $r-1$ decorations for nodes to everyone.
    - Including yourself---simulate locally.
  - Use received messages to decorate level r nodes---to determine label, append sender's id at the end.
  - If no message received, use $\perp$.
- The decoration for node $(i_1, i_2, i_3, \ldots, i_k)$ in i's tree is the value v such that (i_k told i) that (i_{k-1} told i_k) that …that (i_1 told i_2) that **i_1**'s initial value was v.
- Decision rule for stopping case:
  - Trivial
  - Let W = set of all values decorating the local EIG tree.
  - If $|W| = 1$ decide that value, else default v_0.

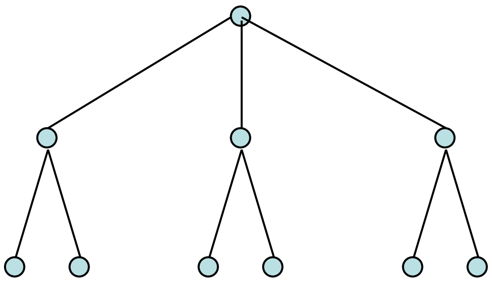
Example

- 3 processes, 1 failure
- Use $T_{3,1}$:



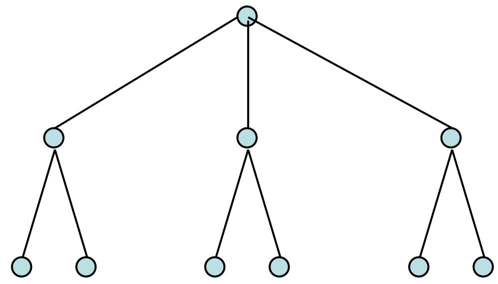
Initial values:

1



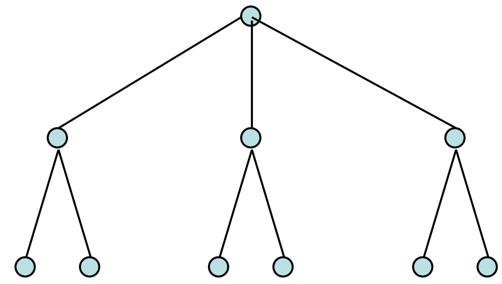
Process 1

0



Process 2

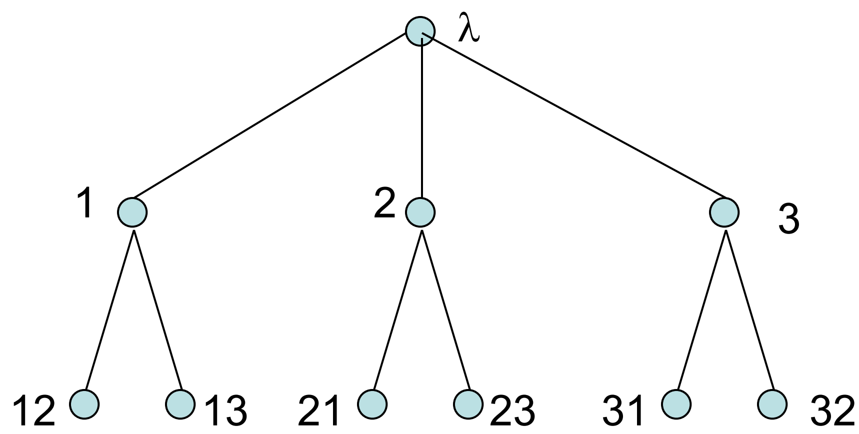
1

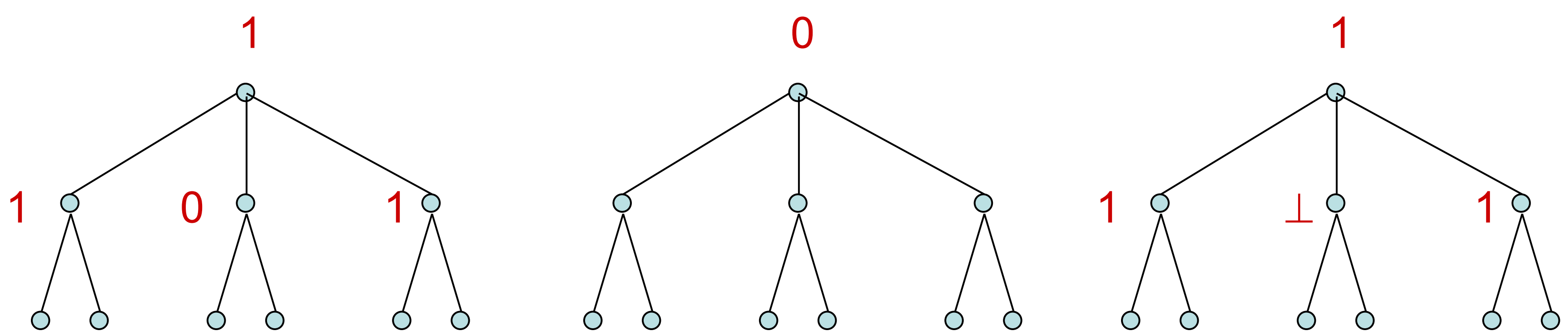


Process 3

Example

- Process 2 is faulty, fails after sending to process 1 at round 1.
- After round 1:

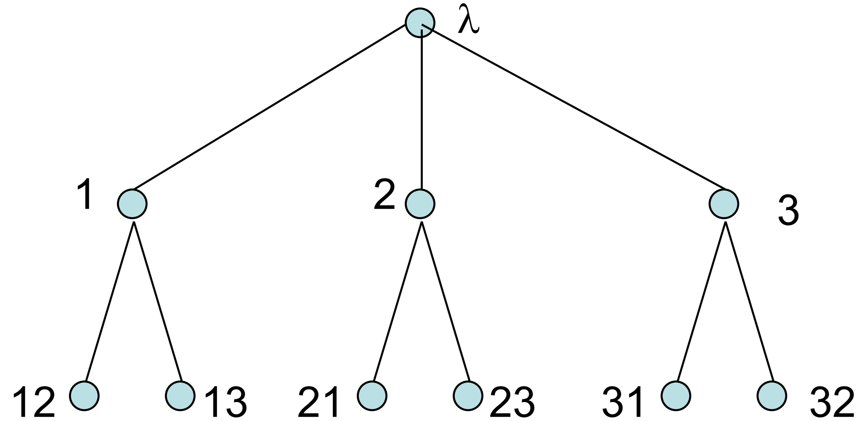


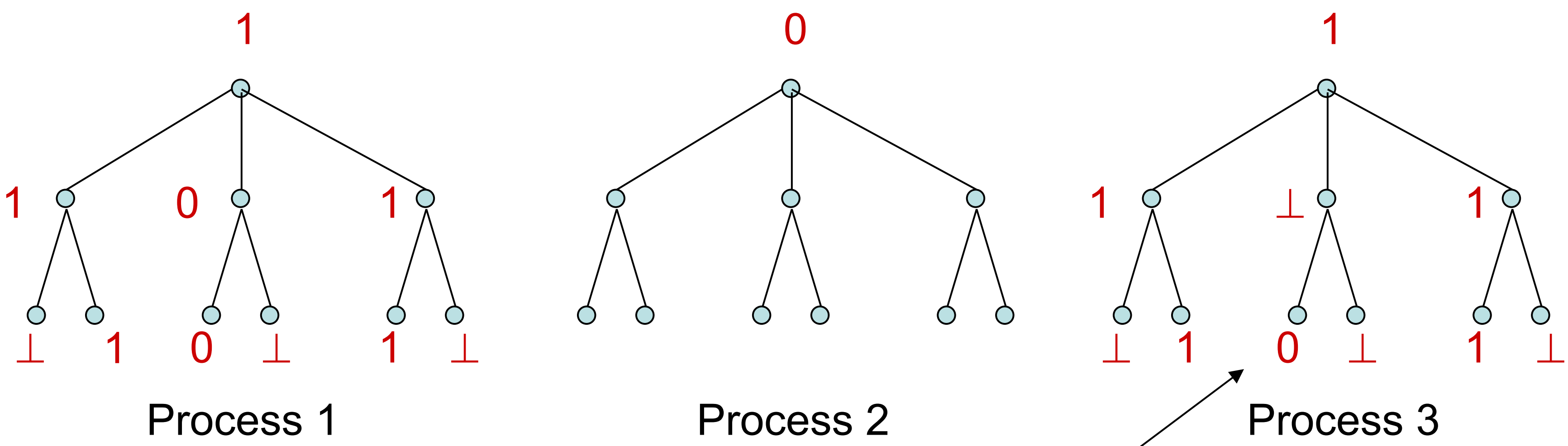


Process 1 Process 2 Process 3

Example

- After round 2:





p3 discovers that p2’s value is 0 after round 2, by hearing it from p1.

Correctness and Complexity

- Correctness similar to previous algorithms.
- Time: $f+1$ rounds, as before.
- Messages: $\leq (f+1)\, n^2$
- Bits: Exponential in number of failures, $O(n^{f+1}\, b)$
- Can improve as before by only relaying the first two messages with distinct values.
- Extension:
  - The simple EIG stopping algorithm, and its optimized variant, can be used to tolerate worse types of failures.
  - Not full Byzantine model---that will require more work…
  - Rather, a restricted version of the Byzantine model, in which processes can authenticate messages.
  - Removes ability of process to relay false information about what other processes said.