Fault-Tolerant Consensus

Consensus

Assumptions

- Denote by f the maximum number of processes that may fail. We call the system f-resilient

Description of the Problem

✓ Each process starts with an individual input from a particular value set V. Processes may fail by crashing.

✓ All non-faulty processes are required to produce outputs from the value set V, subject to simple agreement and validity.

Correcteness Conditions

Agreement: No two processes decide on different values.
Validity: If all processes start with the same initial value $v \in V$, then v is the only decision value.
Termination: All non-faulty processes eventually decide.

Motivation

- Processes in a database system may need to agree whether a transaction should commit or abort.
- Processes in a communication system may need to agree on whether or not a message has been received.
- Processes in a control system may need to agree on whether or not a particular other process is faulty.

Synchronous Shared Memory System

- Is there an algorithm that solves consensus in a synchronous shared-memory system?

Assumptions

- The maximum number of processes that can fail is f, where f is some positive integer. We call the system f-resilient.
- The communication graph is a clique of n nodes.
- The communication channels are reliable; all messages sent are delivered.

A Simple Algorithm for Synchronous Message-Passing Systems

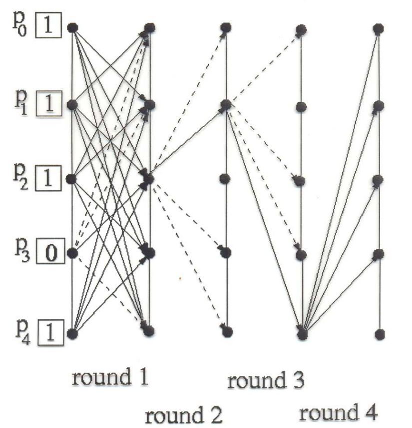
code for processor p_i, $0 \le i \le n-1$.

Initially $V = \{x\}$ // V contains p_i's input

round k, $1 \le k \le f+1$:
send $\{v \in V : p_i$ has not already sent $v\}$ to all processors
receive S_j from p_j, $0 \le j \le n-1$, $j \ne i$
$V := V \cup \bigcup_{j=0}^{n-1} S_j$
if $k = f+1$ then $y := \min(V)$ // decide

- Each process maintains a set of the values it knows to exist in the system; initially, this set contains only its own input.
- At the first round, each process broadcasts its own input to all processes.
- For the subsequent f rounds, each process takes the following actions:
  - updates its set by joining it with the sets received from other processes, and
  - broadcasts any new additions to the set to all processes.
- After f+1 rounds, the process decides on the smallest value in its set.

A Simple Algorithm for Synchronous Message-Passing Systems



f = 3, n = 5

A Simple Algorithm for Synchronous Message-Passing Systems

Termination?

Validity?

Intuition for Agreement:

- Assume that a process p_i decides on a value x smaller than that decided by some other process p_j.
- Then, x has remained "hidden" from p_j for (f+1) rounds.
- We have at most f faulty processes. A contradiction!!!

Number of processes? $n > f$

Round complexity? (f+1) rounds

Message Complexity?

- n^2 * |V| messages, where V is the set of input values.

Exponential Information Gathering Algorithms

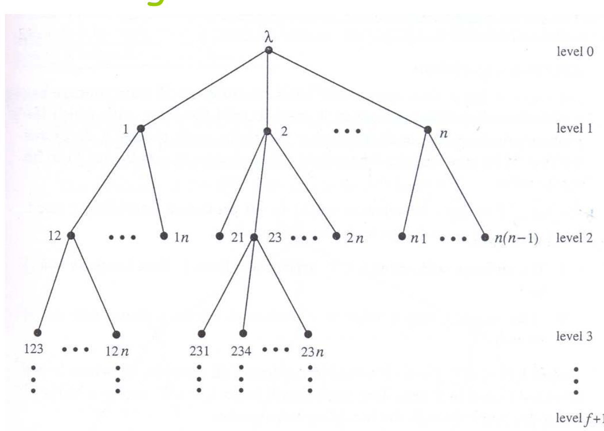
Main Idea

- Processes send and relay initial values for several 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 based on the values recorded in their trees.

Data Structure

- Each process maintains an EIG Tree ($T = T_{n,f}$), each node of which is labeled by a string of process indices.
- Each path in the tree from the root represents a chain of processes along which initial values are propagated.
- The tree T has f+2 levels, 0, ..., f+1.
- Each node of level k has exactly n-k children, where $0 \leq k \leq f$.

Exponential Information Gathering Algorithms



Node Labeling

The root is labeled by the empty string λ.

Each node with label $i_1 \ldots i_k$ has exactly n-k children with labels $i_1 \ldots i_k j$, where $j \in \{1, \ldots, n\} - \{i_1, \ldots, i_k\}$.

Exponential Information Gathering Algorithms

- The computation proceeds for exactly f+1 rounds.
- In the course of the computation, the processes decorate the nodes of their trees with values in V or null, decorating all those at level k, at the end of round k.

Decoration of the EIG Tree of process p_i

- The root of process p_i tree gets decorated with p_i's input value.
- At each round, if the node labeled by the string $i_1 \ldots i_k$, $1 \le k \le f+1$, is decorated by a value $v \in V \Rightarrow i_k$ has told i at round k that i_{k-1} has told i_k at round k-1 that ... that i_1 has told i_2 at round 1 that i_1's initial value is v.
- If the node labeled by the string $i_1 \ldots i_k$ is decorated with null $\Rightarrow$ the chain of communication $i_1, \ldots, i_k, i$ has been broken by a failure.

Assumption

- Each process is able to send messages to itself in addition to the other processes.

EIGStop Algorithm

- For every string x that occurs as a label of a node of T, p_i has a variable val(x).
  - val(x) holds the value with which the process decorates the node labeled x.
  - Initially, val(λ) = initial value of p_i.

Round 1: Process p_i broadcasts val(λ) to all processes, including i itself.

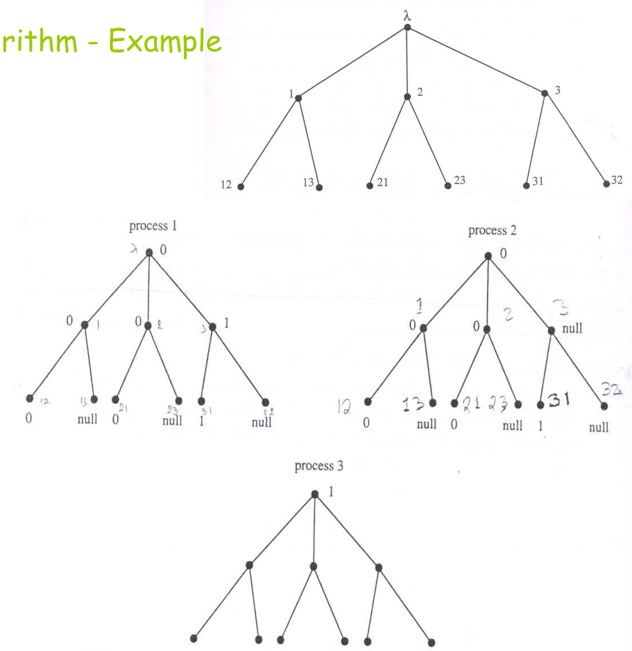
- Then, p_i records the incoming information:
  - If a message with value v arrives at p_i from $p_j \Rightarrow$ val(j) = v.
  - If no message arrives at p_i from $p_j \Rightarrow$ val(j) = null.

Round k, $2 \le k \le f+1$: Process p_i broadcasts all pairs (x,val(x)), where x is a level k-1 label in T that does not contain index i.

- Then, p_i records the incoming information:
  - If xj is a level k node label in T, where x is a string of process indices and j is a single index, and a message saying that val(x) = v arrives at p_i from p_j, then p_i sets val(xj) to v.
  - If xj is a level k node label in T, and no message with a value in V for val(x) arrives at p_i from p_j, then p_i sets val(xj) to null.
- At the end of f+1 rounds, process p_i applies a decision rule:
  - Let W_i be the set of non-null values that decorate nodes of p_i's tree. Process p_i decides its output to be the smallest element of W_i.

EIGStop Algorithm - Example

- n=3
- f = 1
- 2 rounds
- 3 tree levels
- initial values of p_1, p_2, p_3: 0, 0, and 1, respectively.



EIGStop Algorithm - Correctness

Lemma 1: After f+1 rounds of the EIGStop Algorithm, the following hold:

1. $val(\lambda)_i$ is the input value of p_i
2. If xj is a node label and $val(xj)_i = v$, then $val(x)_j = v$.
3. If xj is a node label and $val(xj)_i = null$, then either $val(x)_j = null$ or else p_j fails to send a message to p_i at round $|x|+1$.

Lemma 2: After f+1 rounds of the EIGStop Algorithm, the following hold:

1. If y is a node label, $val(y)_i = v$ and xj is a prefix of y, then $val(x)_j = v$.
2. If v appears in the set of vals at any process, then $v = val(\lambda)_i$, for some i.
3. If v appears in the set of vals of process p_i, then there is some label y that does not contain i s.t. $v = val(y)_i$.

Proof: Part 1 follows from repeated use of Lemma 1 (part 2).

Part 2: Suppose $v = val(y)_i$. If $y = \lambda$ we are done. Otherwise, let j be the first index in y. Part 1 then implies the claim.

For part 3, suppose to the contrary that v only appears as the val for labels containing i. Let y be a shortest label s.t. $v = val(y)_i$. Then y has a prefix of the form xi. But then part 1 $\Rightarrow$ $val(x)_i = v$, which contradicts the choice of y.

EIGStop Algorithm - Correctness

Lemma 3: If processes p_i and p_j are both non-faulty, then $W_i = W_j$.

Proof: We may assume that $i \neq j$. We show that $W_i \subseteq W_j$ and $W_j \subseteq W_i$.

1. $W_i \subseteq W_j$

   Suppose $v \in W_i$. Then, Lemma 2 implies that $v = val(x)_i$, for some label x that does not contain i.

   i. $|x| < f+1 \Rightarrow |xi| \leq f+1$. Since string x does not contain i, (non-faulty) process p_i relays value v to process p_j at round $|xi| \Rightarrow val(xi)_j = v \Rightarrow v \in W_j$.

   ii. $|x| = f+1$. Because there are at most f faulty processes and all indices in x are distinct, there must be some non-faulty process p_l whose index appears in x

   $\Rightarrow x$ has a prefix of the form yl. Lemma 2 implies that $val(y)_l = v$. Since process p_l is non-faulty, it relays v to p_j at round $|yl| \Rightarrow val(yl)_j = v \Rightarrow v \in W_j$.

3 $W_j \subseteq W_i$. Symmetric to the previous case.

EIGStop Algorithm – Correctness & Complexity

Theorem

- EIGStop solves the consensus problem for stopping failures.

Proof

- ❑ Termination is obvious by the decision rule.

Validity

- ❑ Assume that all the initial values are equal to v. Then, each W_i must be exactly equal to {v}. Thus, all processes output v.

Agreement

- ❑ Let p_i and p_j be any two processes that decide $\Rightarrow p_i, p_j$ are non-faulty. Lemma 3 implies that $W_i = W_j$. Thus, p_i, p_j decide the same output value.

Complexities

- *Round complexity?*
- *Communication Complexity?*

Algorithms for Byzantine Failures

Algorithm EIGByz – Code for process p_i

- The processes (we assume that $n > 3f$) propagate values for $f+1$ rounds in the same way as in EIGStop with the following exceptions:
- If p_i receives a message from p_j that is not of the specified form, then p_i ignores the message.
- At the end of $f+1$ rounds, p_i works from the leaves up in its decorated tree, decorating each node with an additional newval, as follows:
  - For each leaf labeled x, newval(x) = val(x).
  - For each non-leaf node labeled x, newval(x) is defined to be the newval held by a strict majority of the children of node x
    - It takes the value v, s.t. newval(xj) = v for a majority of the nodes with label of the form xj, provided that such a majority exists).
    - If no such majority exists, newval(x) = null.
- The output value of p_i is newval(λ).

Algorithm EIGByz - Correctness

Lemma 1: After $f+1$ rounds of the EIGByz algorithm, the following holds. If p_i, p_j and p_k are non-faulty processes, with $i \neq j$, then $val(x)_i = val(x)_j = val(y)_k$ for each label x ending in k.

Proof: Since k is non-faulty, it sends the same message $val(y)_k$ to p_i and p_j at round |x|.

Lemma 2: After $f+1$ rounds of the EIGByz algorithm, the following holds. Suppose that x = yk is a label such that p_k is a non-faulty process. Then, $newval(x)_i = val(x)_i = val(y)_k$ for all non-faulty processes i.

Proof: By induction on the tree labels, working from the leaves up.

- **Induction Base:** Suppose x is the label of a leaf node ($|x| = f+1$).
  - Due to the way that values are assigned to the newval variables of leaf nodes $\Rightarrow newval(x)_i = val(x)_i$
  - By Lemma 1 $\Rightarrow$ for all non-faulty processes p_i, it holds that $val(x)_i = val(y)_k$.

Algorithm EIGByz - Correctness

- **Inductive Hypothesis:** Fix any r, $1 \le r \le f$ and assume that the claim holds for all labels $x' = y'k'$ s.t. $|x'| = r+1$ (where $p_{k'}$ is a non-faulty process).
- **Inductive Step:** We prove that the claim holds for all labels $x = yk$ with $|x|=r$ (where p_k is a non-faulty process).
  - Lemma 1 $\Rightarrow$ all non-faulty-processes p_i have $val(x)_i = val(y)_k = v \Rightarrow$
  - Every non-faulty process p_j sends the same value v for x to all processes at round r+1 $\Rightarrow val(xj)_i = v$ for all non-faulty processes p_i and p_j.
  - By inductive hypothesis $\Rightarrow newval(xj)_i = val(xj)_i = v$, for all non-faulty processes p_i and p_j.
  - The majority of labels of children of node x end in non-faulty process indices:
  - # of children of $x = n-r \ge n-f > 3f - f = 2f \Rightarrow$ since at most f of the children have labels ending in indices of faulty processes, we have the needed majority.

    $\Rightarrow$ For any non-faulty process p_i, $newval(xj)_i = v$ for a majority of children xj of node x.

    $\Rightarrow newval(x)_i = v$.

Algorithm EIGByz - Correctness

Lemma 3: If all non-faulty processes begin with the same initial value v, then v is the only possible decision value for a non-faulty process.

Proof: All non-faulty processes broadcast v at the 1st round $\Rightarrow val(j)_i = v$ for all non-faulty processes p_i and p_j.

- Lemma 2 $\Rightarrow newval(j)_i = val(j)_i = v$.
- By the majority rule: $newval(\lambda)_i = v$.

Definitions

1. We say that a subset C of the nodes of a rooted tree is a path covering provided that every path from the root to a leaf contains at least one node in C.
2. A tree node x is said to be **common** in a provided that at the end of f+1 rounds in a, all the non-faulty processes p_i have the same $newval(x)_i$.
3. A set of tree nodes is said to be **common** in a if all the nodes in the set are common in a.

Algorithm EIGByz - Correctness

Lemma 4: After f+1 rounds of the EIGByz algorithm, the following holds. Let x be any node label in the EIG tree. If there is a common path covering of the subtree rooted at x, then x is common.

Proof: By induction on tree labels working from the leaves up.

- **Base Case:** Suppose that x is a leaf. If there is a common path covering of the subtree rooted at x, then it contains only x. Thus, x is common.
- **Inductive Hypothesis:** Fix any r, $1 \leq r \leq f$ and assume that the claim holds for each node with label x' s.t. $|x'| = r+1$.
- **Inductive Step:** We prove that the claim holds for each node with label x s.t. $|x|=r$.
  - Suppose that there is a common path covering C of x's subtree. If x itself is in C, then it is common. So assume that x is not in C.
  - Consider any child xl of x. Since $x \notin C$, C induces a common path covering for the subtree rooted at xl.
  - By the inductive hypothesis $\Rightarrow$ xl is common. Since xl was chosen to be an arbitrary child of x, all the children of x are common.
  - Then by the definition of newval(x), x is common.

Algorithm EIGByz - Correctness

Lemma 5: After f+1 rounds of any execution of the EIGByz algorithm, there exists a path covering that is common in a.

Proof: Let C be the set of nodes with labels of the form xi where i is the index of a non-faulty process.

- All nodes in C are common.
- Consider any path from the root to a leaf. It contains exactly f+1 non-root nodes, and the label of each such node ends with a distinct process index.
- Since there are f faulty processes, there is some node of the path whose label ends in a non-faulty process index.
- This node must be in C.

Corollary: After f+1 rounds of the EIGByz algorithm, the root node λ of the tree of each non-faulty process is common.

Theorem: EIGByz solves the Byzantine agreement problem for n processes with f failures, if $n > 3f$.

Number of Rounds with Stopping Failures – Special Case where f = 1

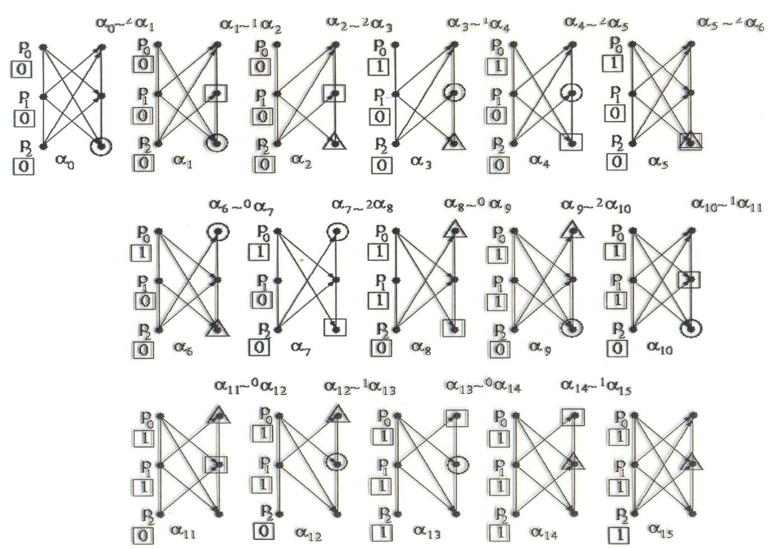
Theorem

- Suppose that n ≥ 3. Then there is no n-process stopping agreement algorithm that tolerates one fault, in which all non-faulty processes always decide by the end of round 1.

Proof: By contradiction. Let A be any such algorithm.

- We construct a chain of executions of A, each with at most one faulty process:
  - o the first execution in the chain contains 0 as its unique decision value,
  - o the last execution in the chain contains 1 as its unique decision value
  - o any two consecutive executions in the chain are indistinguishable to some process that is non-faulty in both.
- ⇒ Every execution in the chain must have the same unique decision value. A contradiction!!!!

Number of Rounds with Stopping Failures – Special Case where f = 1



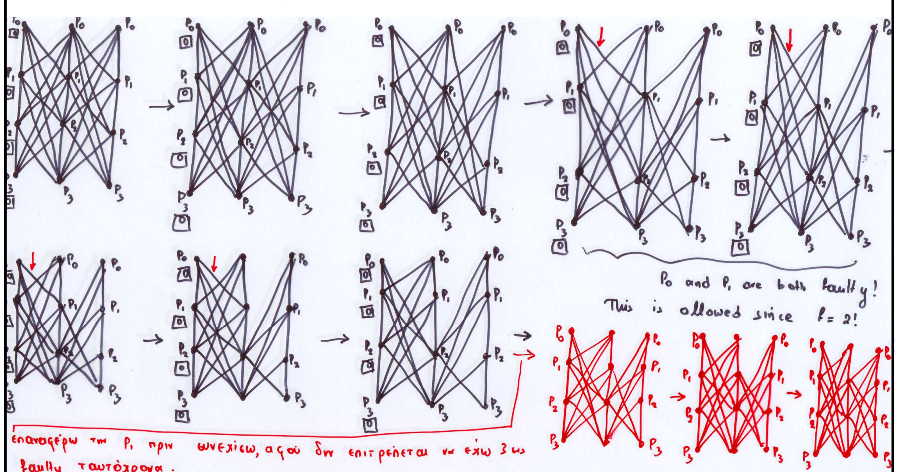
Example n = 3

Number of Rounds with Stopping Failures – Special Case where f = 2

Theorem

- Suppose that $n \geq 4$. Then there is no n-process stopping agreement algorithm that tolerates two faults, in which all non-faulty processes always decide by the end of round 2.
- **Proof:** By contradiction. Let A be any such algorithm.
- We follow similar arguments as that for case f=1. However:
- *Now we have two rounds!!! What problem may result from this?*
- How can we remove a message from p_0 to p_1?
  - o we will remove one-by-one the messages sent by p_1 during the 2nd round
  - o then we will remove the 1st round message from p_0 to p_1
  - o we will recover one-by-one the messages 2nd round messages sent by p_1

Number of Rounds with Stopping Failures – Special Case where f = 2



At this point only process p_0 is faulty.

Number of Rounds with Stopping Failures – Special Case where f = 2

- I repeat this process with p_2 playing the role of p_1, in order to remove the message from p_0 to p_2. I do the same for all other processes.
- Then:
  - o I change the input value of p_0 to 1.
  - o I apply the reverse procedure to recover the 1^{st} round messages of p_0
- I repeat the above procedure with each process p_j, other than p_0, playing the role of p0!

Number of Rounds with Stopping Failures – General Case

Theorem

- Suppose that $n \geq f+2$. Then there is no n-process stopping agreement algorithm that tolerates f faults, in which all non-faulty processes always decide by the end of round f.

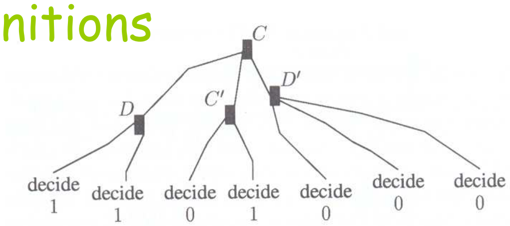
Sketch of Proof

- The main ideas have already been presented!
- The chain of execution that is created is much longer and in order to make it we have to kill f processes.

Impossibility of Consensus in Asynchronous Shared-Memory Systems

Theorem 1:. For $n \geq 2$, there is no algorithm in the read/write shared memory model that solves the agreement problem and guarantees wait-free termination.

Useful Definitions



- The valence of a configuration C is the set of all values decided upon in any configuration reachable from C.
- C is univalent if this set contains one value; it is 0-valent if this value is 0 and 1-valent if this value is 1.
- If the set contains two values then C is bivalent.
- If C is bivalent and the configuration resulting by letting some process p take a step is univalent, we say that p is critical in C.
- **Recall that:** Two configurations C_1 and C_2 are similar to a process p, denoted $C_1 \sim^p C_2$, if the values of all shared variables and the state of p are the same in C_1 and C_2.

Impossibility of Consensus - Proof

Assume, by the way of contradiction, that A is a wait-free consensus algorithm.

Main Ideas of the Proof

- **We construct an infinite execution in which:**
  - **every process takes an infinite number of steps,**
  - **yet every configuration is bivalent,**
  - **and thus no process can decide.**
- **This contradicts the fact that the algorithm is wait-free.**

Impossibility of Consensus

Lemma 2: Let C_1 and C_2 be two univalent configurations. If $C_1 \sim^p C_2$, for some process p, then C_1 is v-valent, if C_2 is also v-valent, where $v \in \{0,1\}$.

Proof: Suppose C_1 is v-valent.

- Consider an infinite execution α from C_1 in which only p takes steps.
- Since the algorithm is supposed to be wait-free ⇒ α is admissible and eventually p must decide in α.
- Since C_1 is v-valent ⇒ p must decide v in α.
- The schedule of α can be applied from C_2
- Since $C_1 \sim^p C_2$ and only p takes steps, it follows that p decides v in this execution as well.
- Thus, C_2 is v-valent, as needed.

Impossibility of Consensus

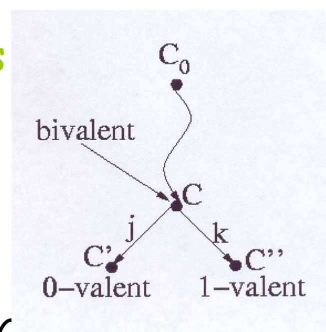
Lemma 3: There exists a bivalent initial configuration.

Proof: By contradiction.

- Let I_0 be the initial configuration in which all processes start with $0 \Rightarrow I_0$ is 0-valent.
- Let I_1 be initial configuration in which all processes start with $1 \Rightarrow I_1$ is 1-valent.
- Let I_{01} be the initial configuration in which p_0 starts with 0 and the remaining processes start with 1.
- $I_{01} \sim^{p0} I_0 \Rightarrow$ (by Lemma 2) I_{01} is 0-valent
- $I_{01} \sim^{p1} I_1 \Rightarrow$ (by Lemma 2) I_{01} cannot be 0-valent.

This is a contradiction!

Impossibility of Consensus



Lemma 4: If C is a bivalent configuration, then at least one processor is not critical in C.

- Proof: By the way of contradiction. Assume that all processes are critical in C.
- Since C is bivalent and all processes are critical in $C \Rightarrow$ there exists two process p_j and p_k such that:
  - if p_j takes a step from C, then the resulting configuration C' is 0-valent, and
  - if p_k takes a step from C the resulting configuration C'' is 1-valent.

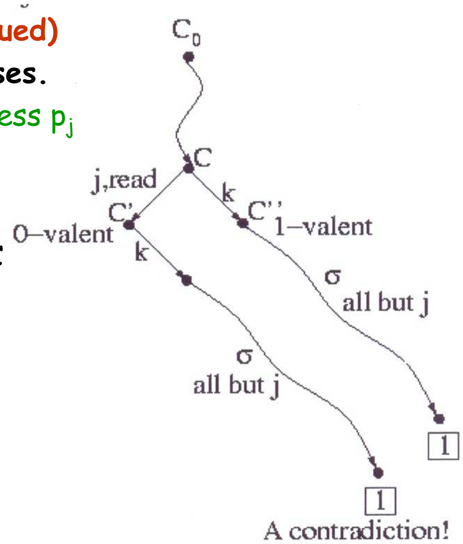
Impossibility of Consensus

Proof of Lemma 4 (continued)

Consider the following cases.

1. The first step of process p_j from C is a read.

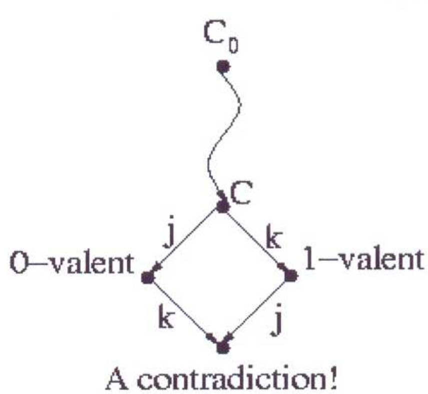
   The case where the first step of p_k from C is a read is symmetric.



Impossibility of Consensus

Proof of Lemma 4 (continued)

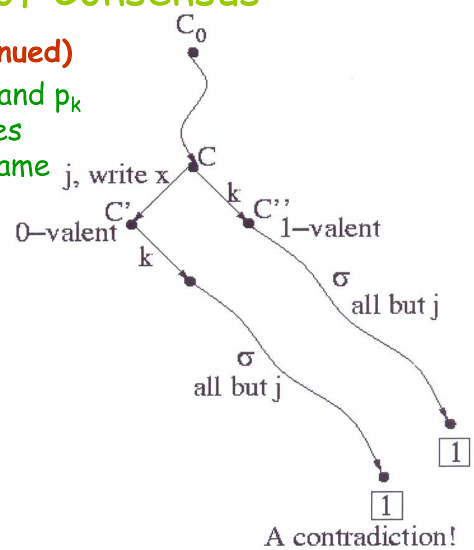
2. The first steps of p_j and p_k from C are both writes and they are to different variables.



Impossibility of Consensus

Proof of Lemma 4 (continued)

2. The first steps of p_j and p_k from C are both writes and they are to the same variable.



Impossibility of Consensus

Proof of Theorem 1

- We inductively create an admissible execution C_0 i_1 C_1 i_2 … in which the configurations remain bivalent forever.
  - By Lemma 3, there is an initial bivalent configuration; let it be C_0.
  - Suppose the execution has been created up to bivalent configuration C_k.
  - By Lemma 4, some process is not critical in C_k; denote this process by p_{ik}.
  - Then, p_{ik} can take a step without resulting in a univalent configuration.
  - We apply the event i_k to C_k to obtain C_{k+1} which is also bivalent.
- If we repeat this procedure forever, we will construct an execution in which all the configurations are bivalent. Thus, no process ever decides, contradicting the termination property of the algorithm and implying Theorem 1.