

# Valency Arguments

Faith Ellen

Department of Computer Science

University of Toronto

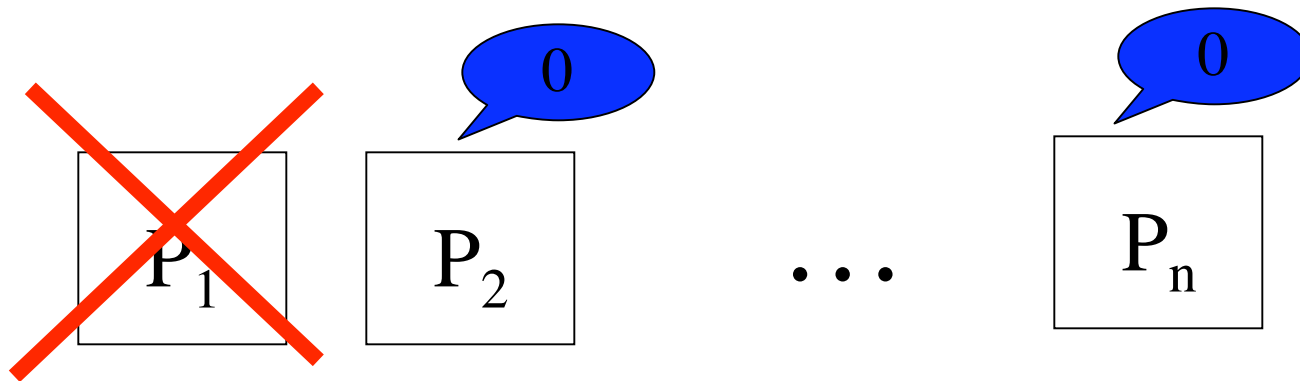
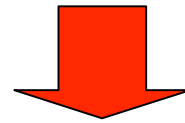
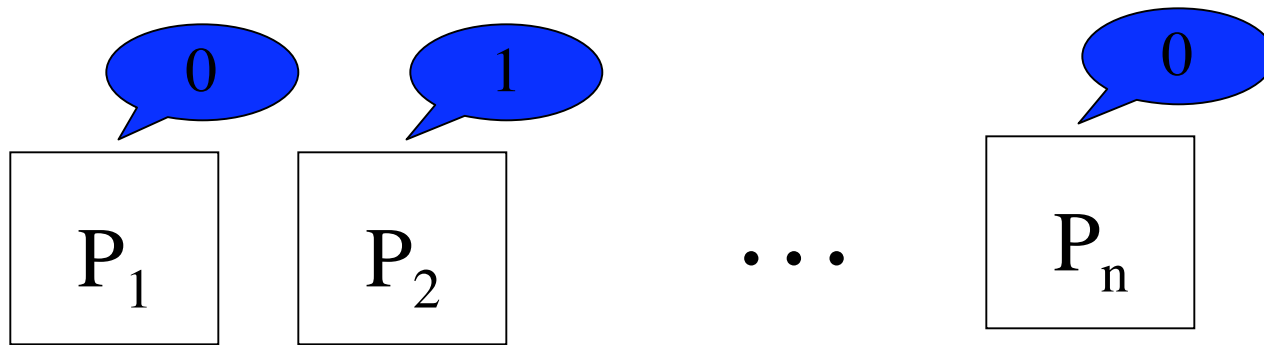
# Valency Arguments

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson, *Impossibility of distributed consensus with one faulty process*, JACM, volume 32, April 1988, pages 374–382.

# OUTLINE

- consensus problem
- valency argument
- proof that fault tolerant consensus is unsolvable in an asynchronous system
- impact of this result
- some other results proved using valency arguments and its variants

# Consensus



# Consensus

Each process has a private input.

- **Termination:** Every nonfaulty process outputs a value after finitely many steps.
- **Agreement:** All output values are the same.
- **Validity:** The output value of each process is the input value of some process.

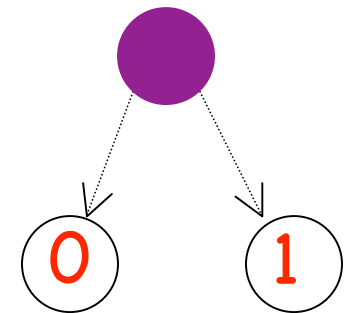
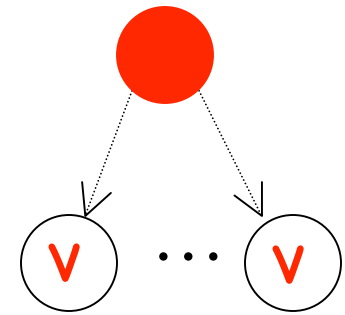
# Valency of Configurations

A configuration  $C$  is:

**v-valent** if all executions starting from  $C$  produce output  $v$ .

**univalent** if it is **v-valent** for some output value  $v$ , and

**multivalent** if there are two executions starting from  $C$  that produce different outputs.

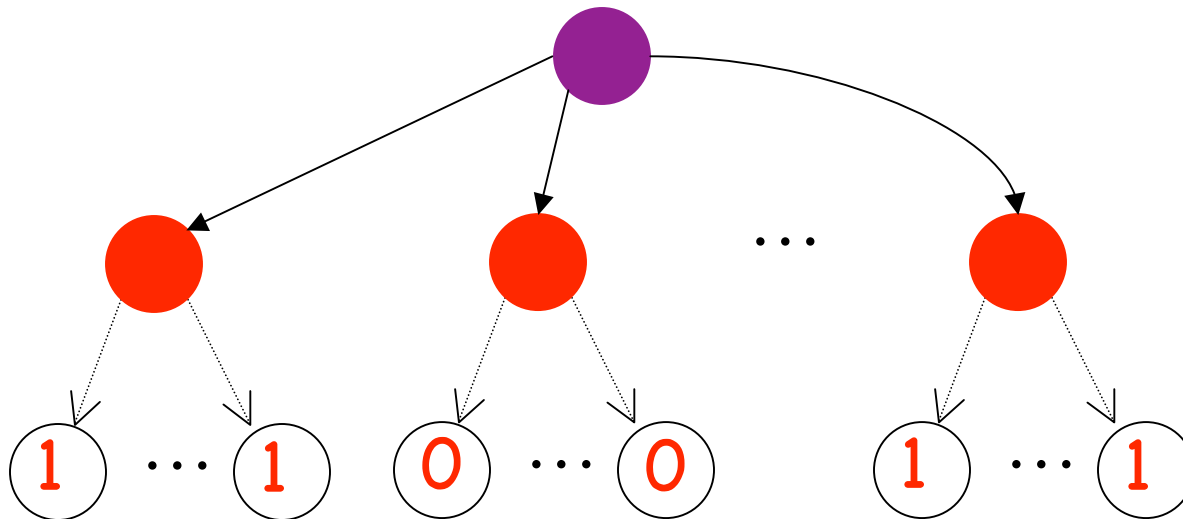


In every consensus algorithm, every configuration in which some process has decided what value to output is **univalent**.

In particular, **every final configuration** is **univalent**.

# Critical configuration

A critical configuration is a multivalent configuration where one step by any process will move the system into a univalent configuration.



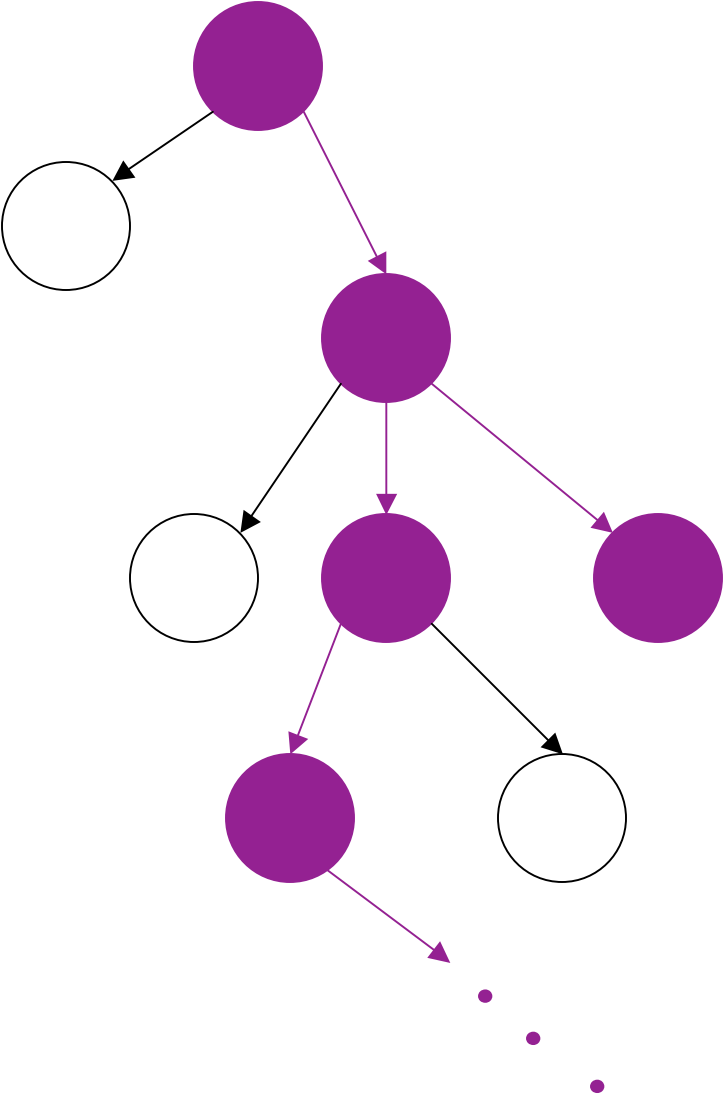
# Valency Arguments

Consider any problem (such as consensus) in which **all executions are finite** and **all final configurations are univalent**.

To prove that this problem is unsolvable, it suffices to show any algorithm which solves the problem has:

1. a multivalent initial configuration, and
2. no critical configurations,

because this implies there is an infinite execution (with only multivalent configurations).



**CLAIM:** Every consensus algorithm has a multivalent initial configuration.

**Proof:** by contradiction. Suppose all initial configurations are univalent.

Let  $C_i$  denote the initial configuration where process  $P_j$  has input value

1 if  $j \leq i$

0 if  $j > i$ .

00...0

$C_0$

10...0

$C_1$

1...10

$C_{n-1}$

1...11

$C_n$

00...0	10...0	1...10	1...11
$C_0$	$C_1$	$C_{n-1}$	$C_n$

By validity, all executions from  $C_0$  output 0, and all executions from  $C_n$  output 1. Hence, there exists  $i$  such that all executions from  $C_{i-1}$  output 0, and all executions from  $C_i$  output 1.

Consider any execution  $\alpha$  starting from  $C_{i-1}$  in which  $P_i$  crashes before taking any steps. To all other processes,  $C_{i-1}$  and  $C_i$  are indistinguishable. Hence they have output the same values when  $\alpha$  starts from  $C_{i-1}$  and when  $\alpha$  starts from  $C_i$ .

To prove the impossibility of consensus,  
it suffices to prove that no consensus  
algorithm has a critical configuration.

To prove the impossibility of consensus,  
it suffices to prove that no consensus  
algorithm has a critical configuration.

In some models, consensus can be solved.

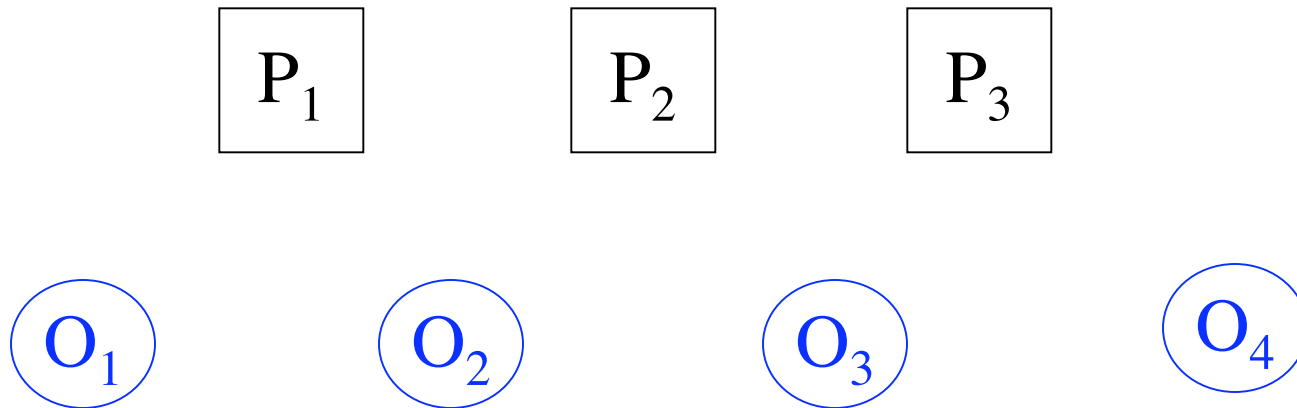
To prove the impossibility of consensus, it suffices to prove that no consensus algorithm has a critical configuration.

In some models, consensus can be solved.

To prove the impossibility of consensus in a particular model of computation, it suffices to prove that no consensus algorithm for that model has a critical configuration.

# Shared Memory Models

Processes communicate through shared objects.



# Register

Set of possible values:  $N$

Initial value:  $0$

**READ( $R$ )**: Returns the value of register  $R$ , doesn't change its value.

**WRITE( $R, v$ )**: Sets the value of  $R$  to  $v$ . Returns **ack**.

# Compare & Swap Object

Set of possible values:  $N$

Initial value:  $\perp$

$C\&S(R,u,v)$ : If the value of  $R$  is  $u$ , sets it to  $v$  and returns **true**; otherwise, leaves the value of  $R$  unchanged and returns **false**.

Each **object** has:

- a set of possible **values**,
- an **initial value**, and
- a set of **operations** that can be applied to it.

A step consists of a **process** that applies **an operation** to **an object**. The process updates its state depending on its current state and the response from the object.

Processes perform steps **atomically**.

In **synchronous** models, all processes take steps at the same speed.

In each round, every process that has not failed takes exactly one step.

In **asynchronous** models, the speed of each process may vary arbitrarily. Only one step occurs at a time.

$n$  processes can solve binary consensus using one Compare & Swap object  $R$ .

algorithm for process  $p_i$  with input  $x_i$ :

$C\&S(R, \perp, x_i)$

if  $C\&S(R, 0, 0)$  then return 0

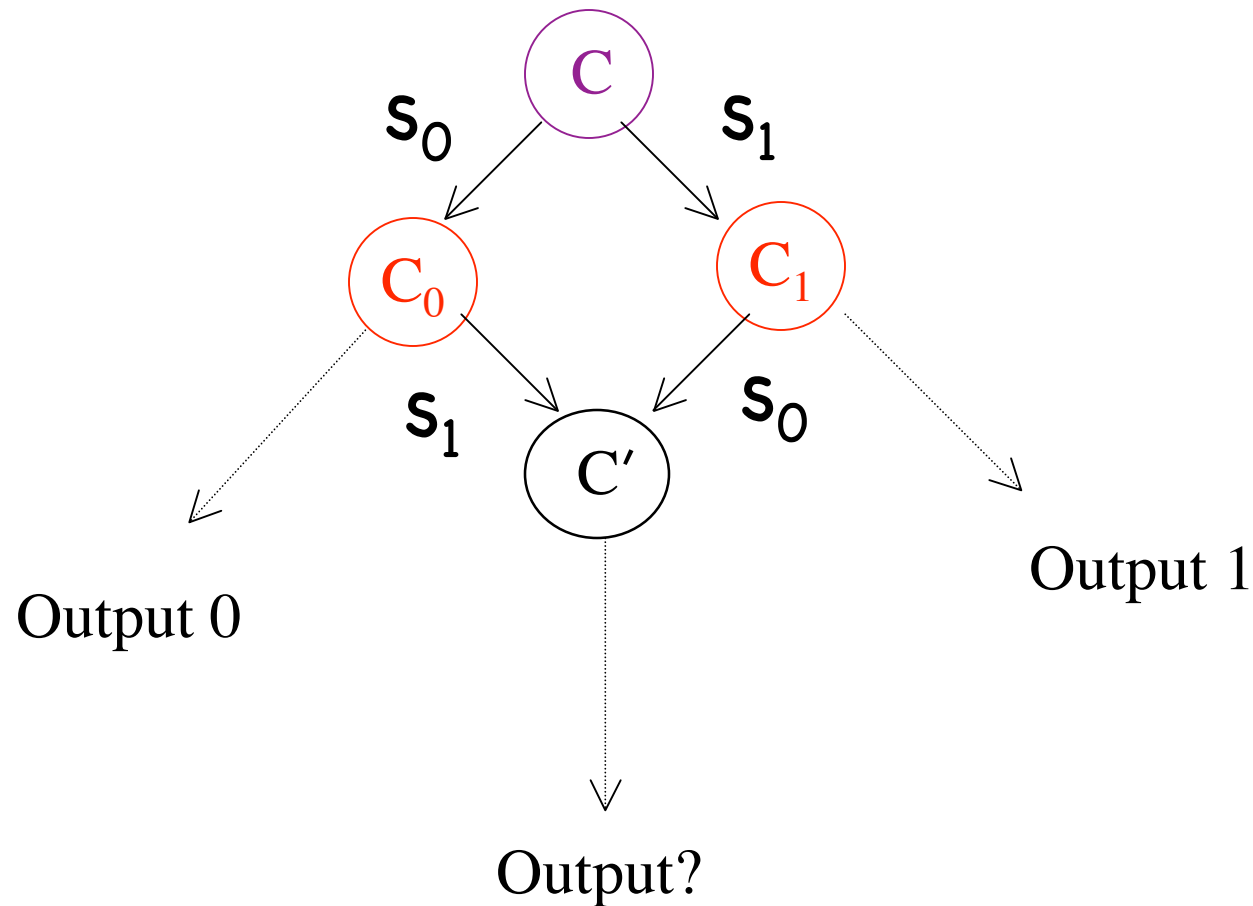
if  $C\&S(R, 1, 1)$  then return 1

**THEOREM:** 2 processes cannot solve binary consensus using only **Registers** in an **asynchronous** system in which one of them can crash [Chor, Israeli & Li, Loui & Abu-Amara, Abrahamson, Herlihy]

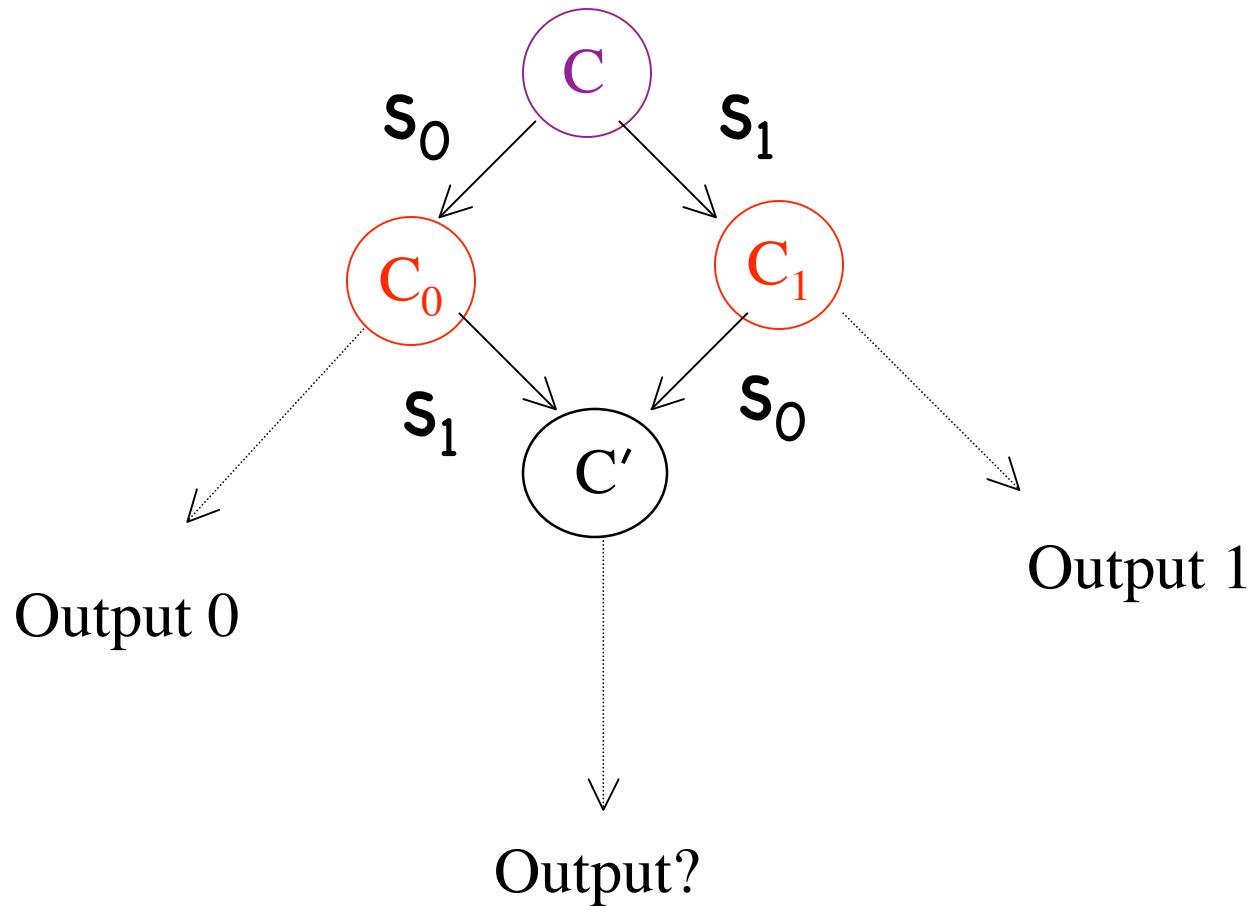
It suffices to prove that no binary consensus algorithm **for this model** has a **critical configuration**.

To obtain a contradiction, suppose there is an algorithm for this model that has a **critical configuration**.

$s_0$  and  $s_1$  are steps (by different processes)  
that access different registers

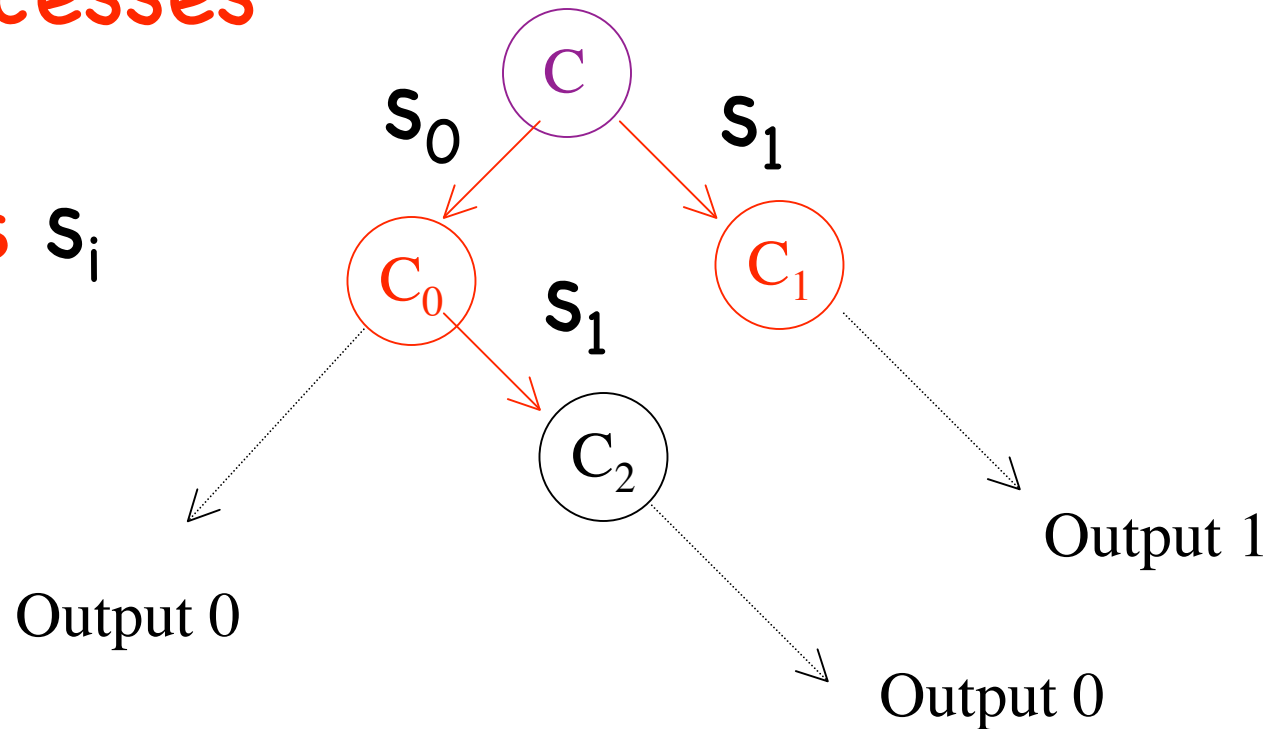


$s_0$  and  $s_1$  are steps that read the same register



$s_1$  is a write to the same register that step  $s_0$  accesses

$p_i$  performs  $s_i$



$C_1$  and  $C_2$  are indistinguishable to  $p_1$ , so the executions from  $C_1$  in which  $p_0$  takes no steps behave the same when started from  $C_2$ .

A similar valency argument [Herlihy] shows that **3** processes cannot solve consensus using only **registers** and **queues** in an asynchronous system.

Therefore, asynchronous shared memory systems with only **registers** and **queues** are less powerful than asynchronous shared memory systems with compare&swap objects.

This is important information for a computer architect to guide or justify design decisions.

# Consensus Number

Objects can be classified by their ability to solve consensus.

$\text{Cons}(T) = \max \{ n \mid n \text{ process consensus can be solved using copies of } T \text{ and registers } \}$

If  $\text{Cons}(T') > \text{Cons}(T)$  then  $T'$  cannot be implemented from copies of  $T$  and registers.

## The impossibility of consensus motivated the study of:

- stronger models of computation
  - partial synchrony [Dolev, Dwork, and Stockmeyer]
  - broadcasts (sending multiple messages at once) [DDS]
  - operations on multiple or overlapping words [Ha, Tsigas, & Anshus, Ruppert]

## The impossibility of consensus motivated the study of:

- weaker kinds of agreement
  - k-set agreement
  - approximate agreement
  - known weak dependencies on the input values of different processes
  - randomized agreement

Valency arguments can be used to prove lower bounds on space complexity.

$\Omega(\sqrt{n})$  registers are necessary to solve randomized consensus among  $n$  processes.  
[Fich, Herlihy, & Shavit]

The proof shows that, if an algorithm uses too few registers (as compared with the number of processes), then it has an execution that decides both 0 and 1.

A register is **covered** if some process is about to write to it.

If there is a multivalent configuration in which **all registers are covered**, it is easy to construct an execution that decides both **0** and **1**.

Such a configuration can be constructed using a **valency argument** by showing that any multivalent configuration with **uncovered registers** leads to another multivalent configuration with **fewer uncovered registers**.

Valency arguments can also be used to obtain lower bounds on the number of rounds to solve consensus in synchronous models [Moses & Rajsbaum, Aguilera & Toueg].

At each round, to ensure multivalence, the adversary crashes one process.

This can continue only so long as there remain processes for the adversary to crash.

Any algorithm for randomized consensus among  $n$  processes using only registers requires  $\Omega(n^2)$  expected total steps. [Aspnes, Attiya & Censor]

The proof considers a restricted set of executions, consisting of a sequence of layers in which each process takes at most 1 step and only a small number of processes take no steps.

The configurations at the end of each layer are classified according to the decisions that can be reached from them via such executions:

An **adversary strategy** decides which process takes the next step in any execution.

What steps that process will subsequently perform may depend on its local coin tosses.

A configuration **C** is **v-potent** if there is an adversary strategy which, with high probability, starting from **C** causes an execution that decides **v**.

A configuration is:

**0-valent** if it is **0-potent**, but not **1-potent**,

**1-valent** if it is **1-potent**, but not **0-potent**,

**bivalent** if it is both **0-potent** and **1-potent**, and

**nullvalent** if it is neither **0-potent** nor **1-potent**.

Faith Ellen Fich and Eric Ruppert,  
**Hundreds of Impossibility Results for  
Distributed Computing,**  
*Distributed Computing*, volume 16,  
2003, pages 121-163.