EE 382C/361C: Multicore Computing **Fall 2016**

Lecture 18: October 27

Lecturer: Vijay Garg *Scribe: Yu Sun*

18.1 Atomic Scan

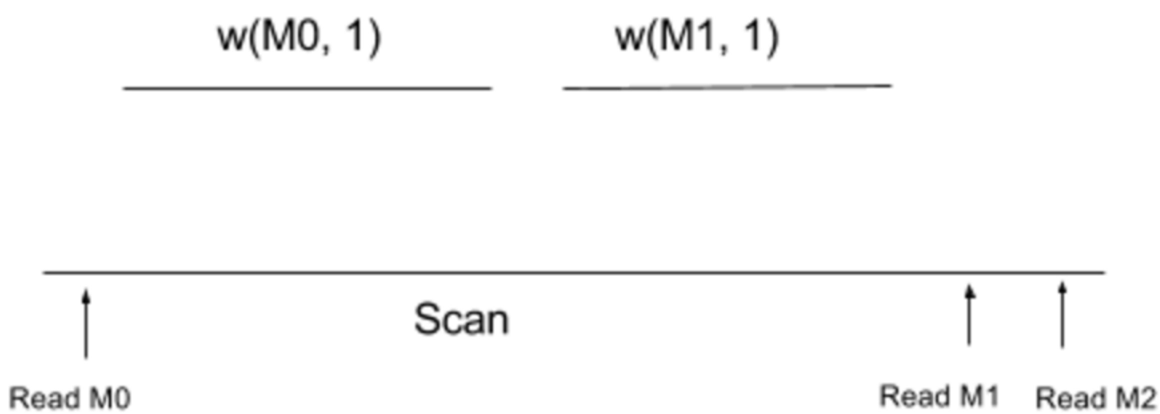
Scan → Atomic read of multiple locations

Update → Atomic write of a single location
Assume the memory state is as following:

M0	0
M1	0
M2	0

We now do two write operations: write (M0, 1) and write (M1, 1)

It is possible that the scan return 010 which is not an atomic execution. Because read(M0) may happen before write(M0, 1) but read(M1) can happen after write(M1, 1).



Non-atomic Scan

Solution

So how to solve this problem? We add a timestamp to each value so that each write operation will update the corresponding timestamp and read operation can guarantee the consistency by compare the timestamps.

Update: Write the new value with updated timestamp.

Scan: Collect the entire array in W.

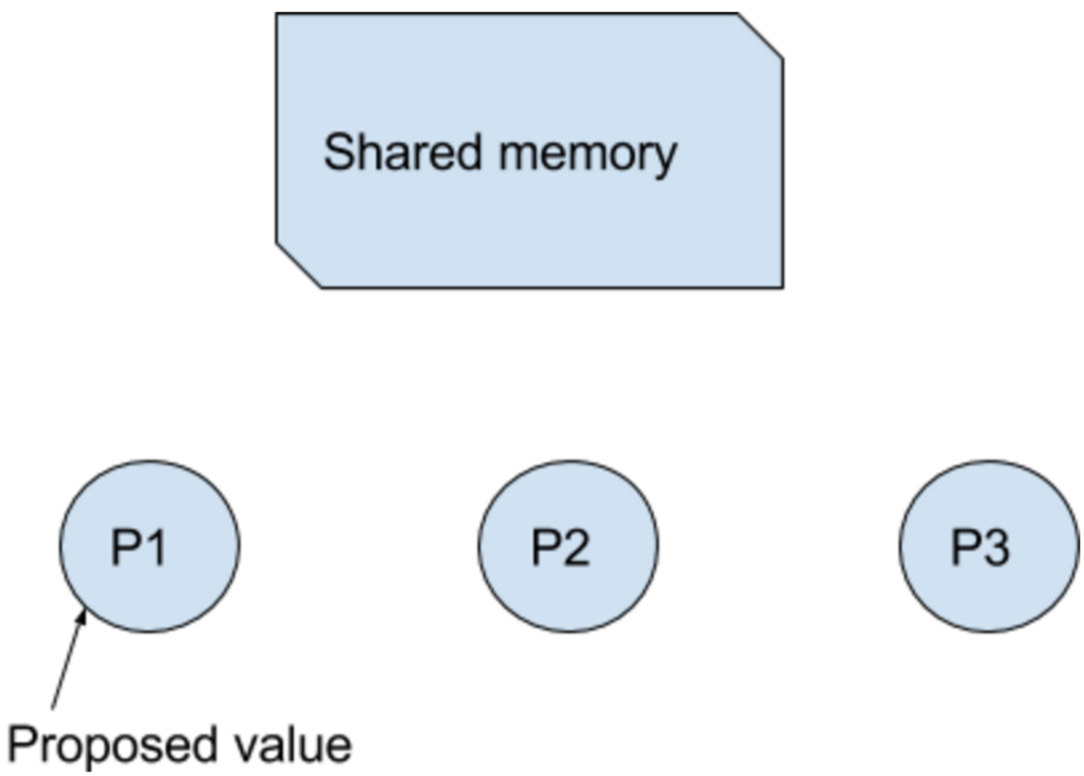
Loop: Read the array again to make sure that there is no change in any timestamp.

If there is some change then go to Loop;

18.2 Consensus

The **consensus** problem requires a given set of processes to agree on an input value.

We abstract the consensus problem as follows: Each process has a value input to it that it can propose. For simplicity, we will restrict the range of input values to a single bit. The processes are required to run a protocol so that they decide on a common value.



The **requirements** on any object implementing consensus are as follows:

- Agreement: No two correct processes decide on different values.
- Validity: The value decided must be proposed by some process.
- Wait-freedom: Decides in a finite number of steps.

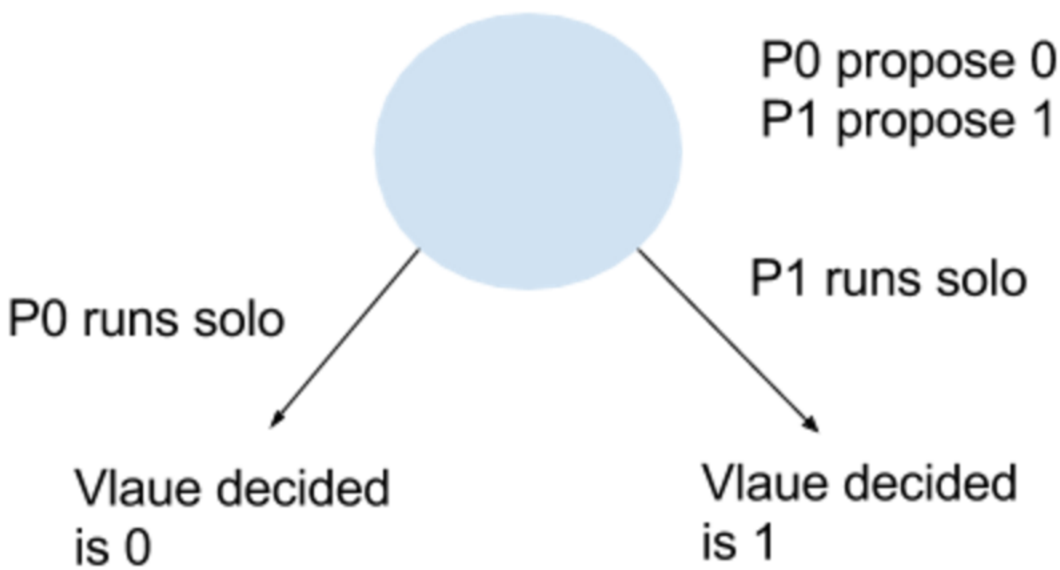
A protocol is in a bivalent state if both the values are possible as decision values starting from that global state.

A bivalent state is a critical state if all possible moves from that state result in nonbivalent states.

18.2.1 Consensus Claims

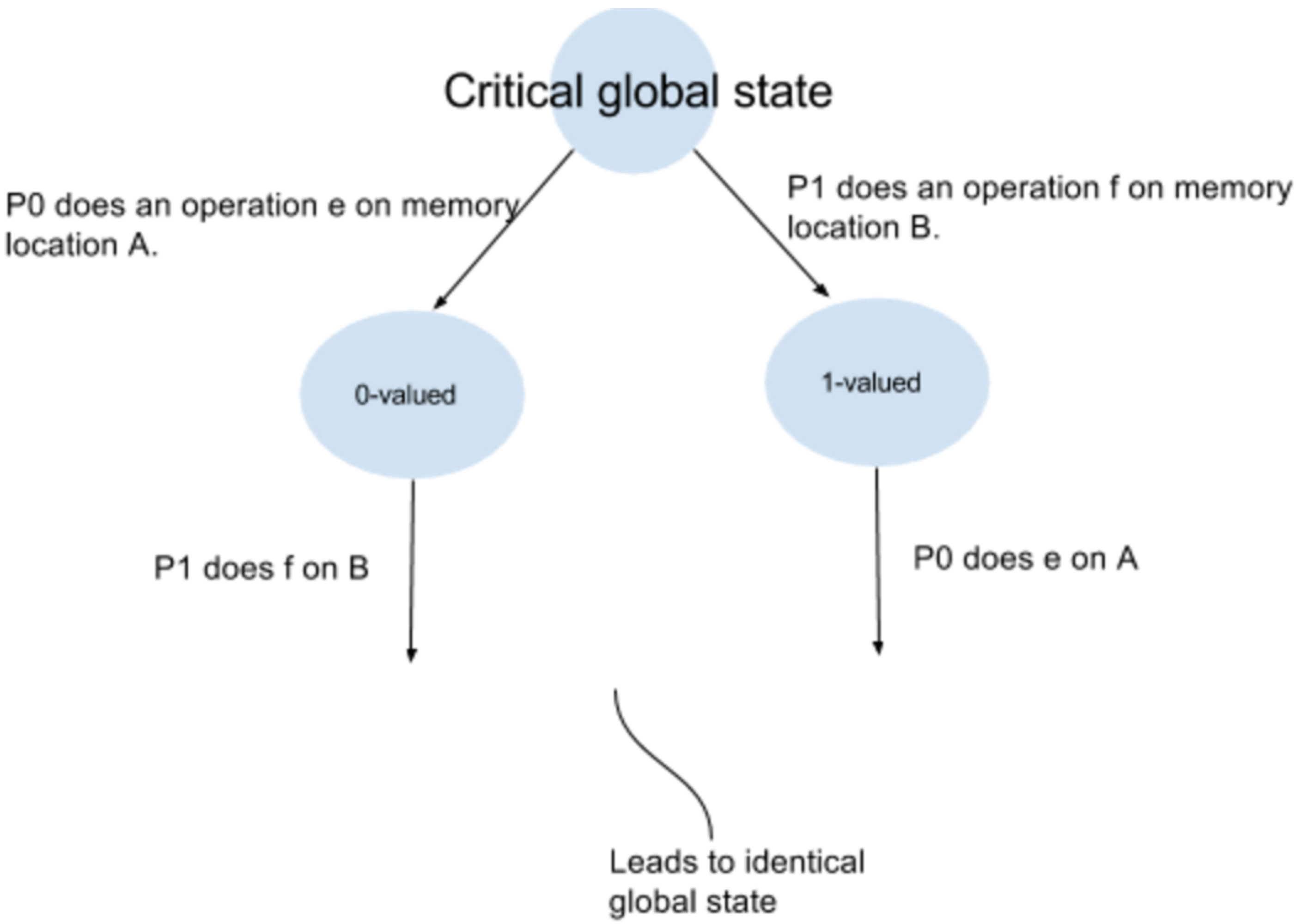
Claim 1: There exists an initial bivalent global state for any consensus protocol.

Proof: Because there exist at least two runs from that state that result in dierent decision values. In the rst run, the process with input 0 gets to execute and all other processes are very slow. Because of wait freedom, this process must decide, and it can decide only on 0 to ensure validity. A similar run exists for a process with its input as 1.

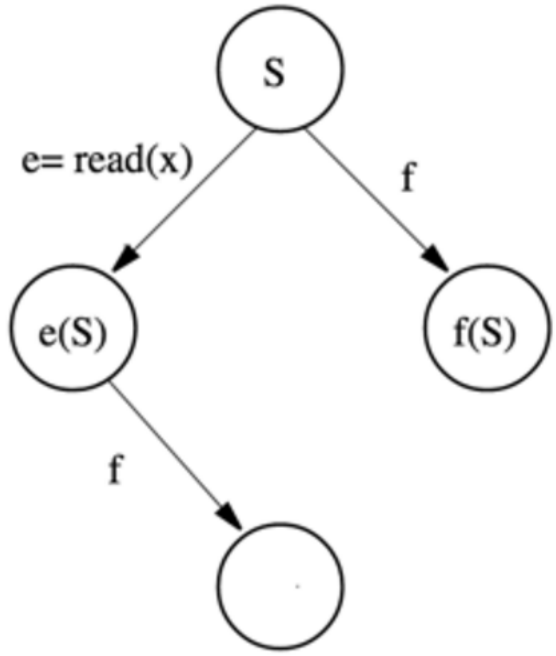


Claim 2: There exists a critical global state for every consensus protocol.

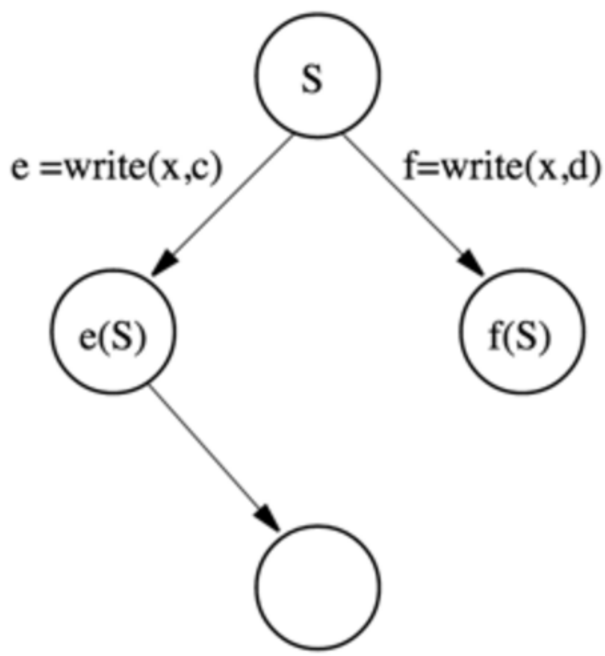
Claim 3: There does not exist any protocol to solve the consensus using atomic registers. **Proof:** We show that even in a two-process system, atomic registers cannot be used to go to non-bivalent states in a consistent manner. We perform a case analysis of events that can be done by two processes, say, P and Q in a critical state S. Let e be the event at P and event f be at Q be such that e(S) has a decision value dierent from that of f(S). We now do a case analysis:



Case 1: e and f are on dierent registers. In this case, both ef and fe are possible in the critical state S. Further, the state ef(S) is identical to fe(S) and therefore cannot have dierent decision values. But we assumed that f(S) and e(S) have dierent decision values, which implies that e(f(S)) and f(e(S)) have dierent decision values because decision values cannot change.



Case 2: Either e or f is a read. Assume that e is a read. Then the state of Q does not change when P does e. Therefore, the decision value for Q from f(S) and e(S), if it ran alone, would be the same; a contradiction.



Case 3: Both e and f are writes on the same register. Again the states f(S) and f(e(S)) are identical for Q and should result in the same decision value.

18.3 SPSC

Assume we have a queue storing values of 'win' and 'lose'.

Pi:

Write my proposal in the prop array.

Deq from Queue

If I win choose my proposal, otherwise choose other guys proposal.

Theorem: There is no wait-free algorithm to build SPMC Queue using atomic read write registers.

Proof: Consensus number of a shared object class O is the maximum number of processes that can use objects from class O to solve consensus.

Consensus number of a shared is the maximum number of processes that can use that object to solve consensus. The following is the operation with its corresponding consensus number.

Operation: Consensus Number

R/W Register: 1

Test And Set: 1

Get And Increment: 2

Swap: 2

CAS: ∞

CAS Operation

Initialize to -1

Pi:

Write my proposal in the prop array

Do R.CAS(-1, my_pid):

Fail: decide prop[R];

Succeed: decide prop[my_id];

References

[1] V. K. Garg, Introduction to Multicore Computing