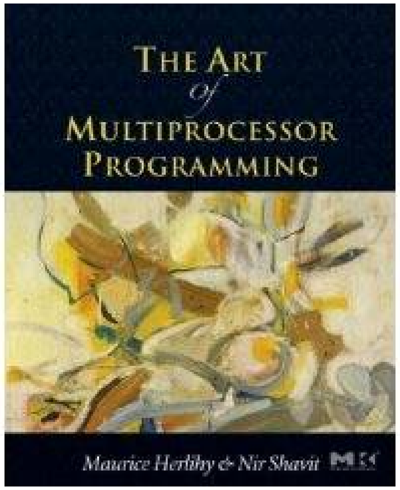
Mutual Exclusion


The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Companion slides for

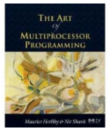
The Art of Multiprocessor Programming

by Maurice Herlihy & Nir Shavit

With some very minor changes by APS

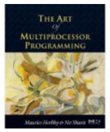
Deep Philosophical Question

- The Bakery Algorithm is
  - Succinct, Elegant,
  - Mutual Exclusion, Deadlock-Free, Starvation-Free
  - Fair.
- Q: So why isn't it practical?
- A: Well, you have to read **N** distinct variables



Shared Memory

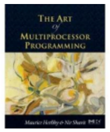
- Shared read/write memory locations called Registers (historical reasons)
  - Assume Reads and Writes are atomic
- Come in different flavors
  - Multi-Reader-Single-Writer (`Flag[]`)
  - Multi-Reader-Multi-Writer (Victim`[]`)
  - Not that interesting: SRMW and SRSW



Theorem

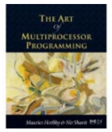
At least N MRSW (multi-reader/single-writer) registers are needed to solve N-thread deadlock-free mutual exclusion.

N registers like Flag[]…



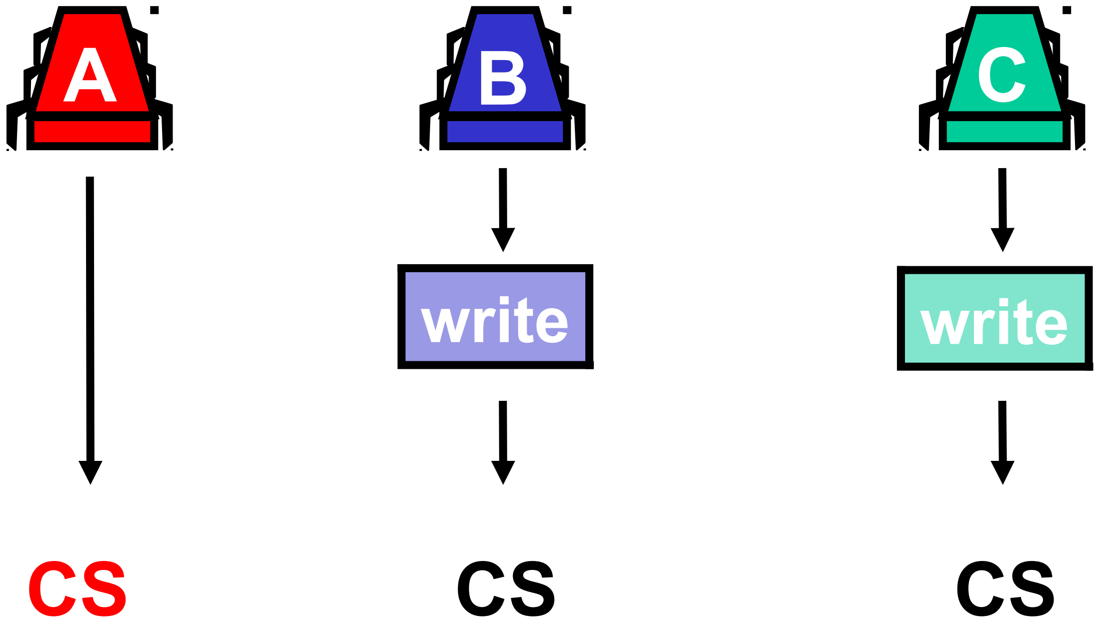
Proving Algorithmic Impossibility

- To show no algorithm exists:
  - assume by way of contradiction one does,
  - show a bad execution that violates properties:
  - in our case assume an alg for deadlock free mutual exclusion using < N registers

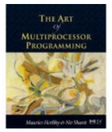


Proof: Need N-MRSW Registers

Each thread must write to some register

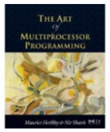


…can't tell whether A is in critical section



Upper Bound

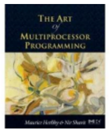
- Bakery algorithm
  - Uses 2N MRSW registers
- So the bound is (pretty) tight
- But what if we use MRMW registers?
  - Like victim[] ?



Bad News Theorem

At least N MRMW multi-reader/multi-writer registers are needed to solve deadlock-free mutual exclusion.

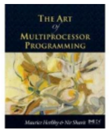
(So multiple writers don’t help)



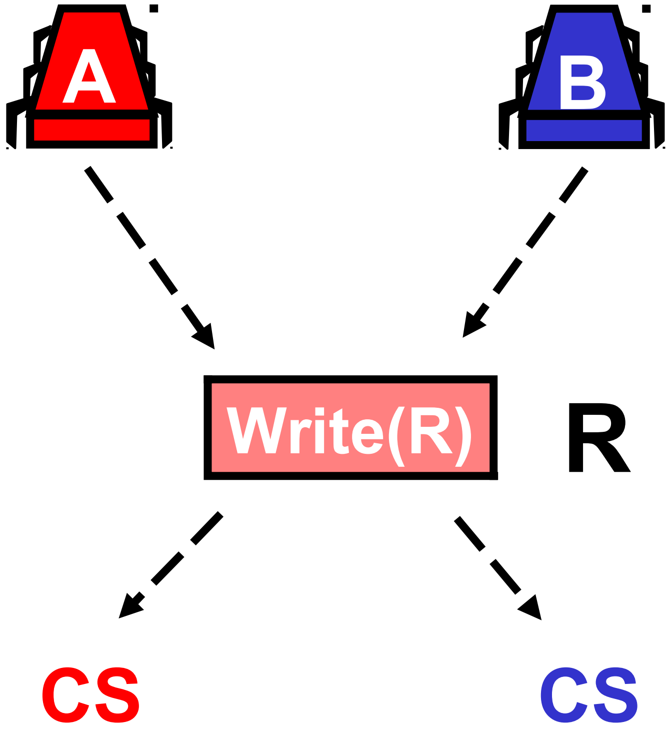
Theorem (For 2 Threads)

Theorem: Deadlock-free mutual exclusion for 2 threads requires at least 2 multi-reader multi-writer registers

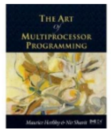
Proof: assume one register suffices and derive a contradiction



Two Thread Execution

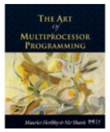


- Threads run, reading and writing R
- Deadlock free so at least one gets in

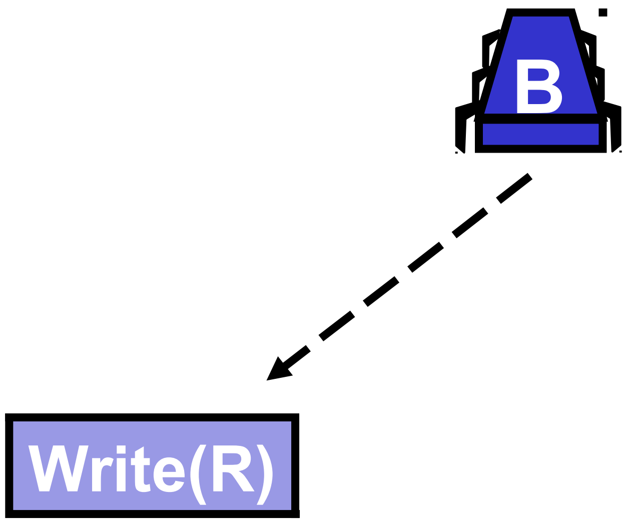


Covering State

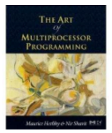
- One thread each about to write to each shared location on next instruction
- The shared locations "look" like the is no thread currently in, or waiting to enter, the Critical Section



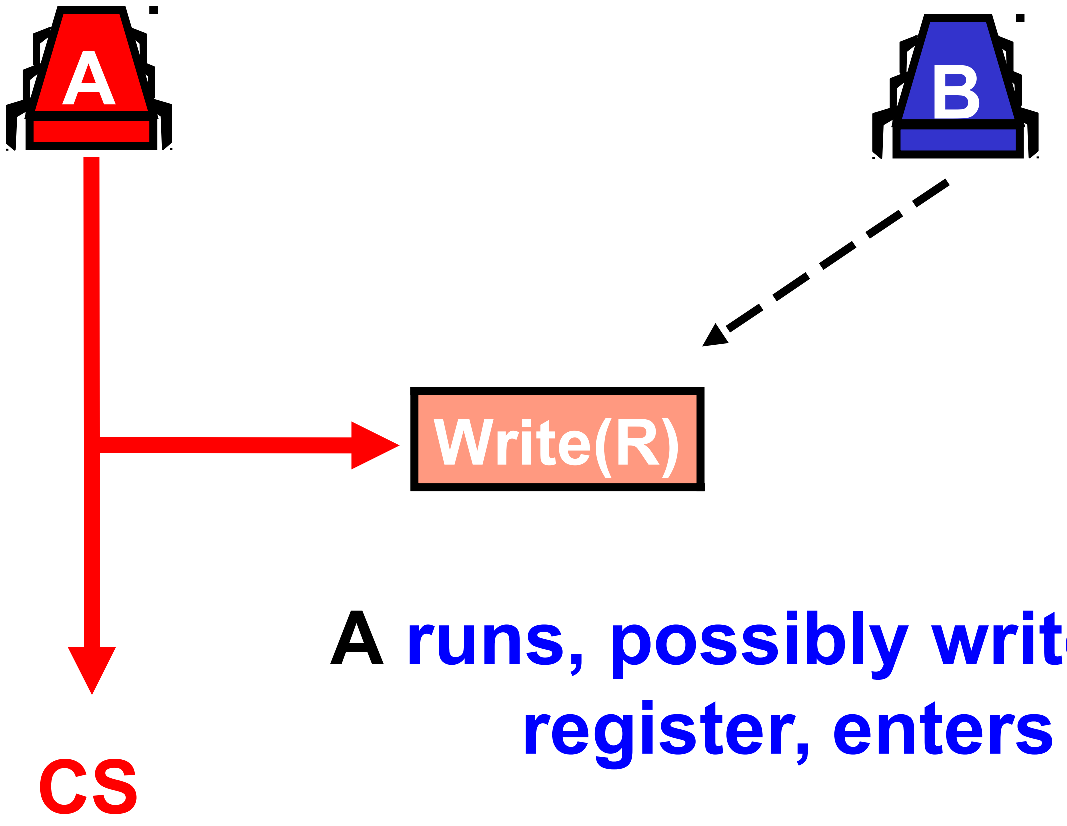
Covering State for One Register Always Exists



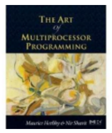
In any protocol B has to write to the register before entering CS, so stop it just before



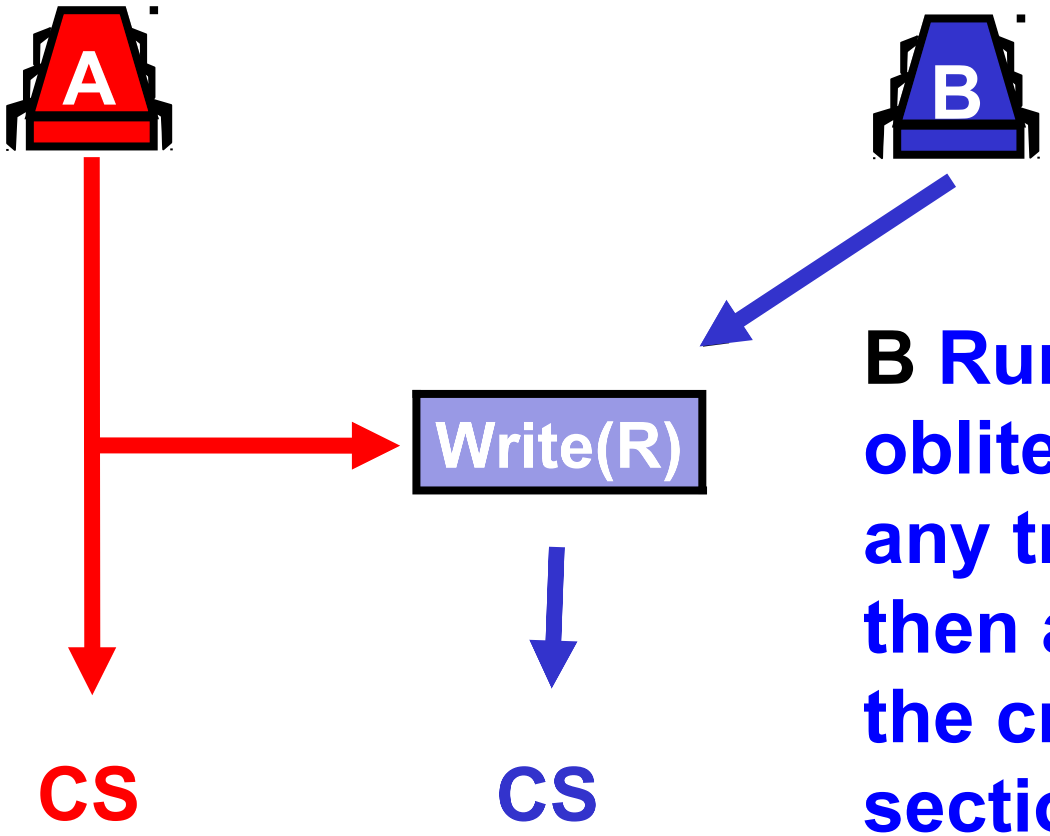
Proof: Assume Cover of 1



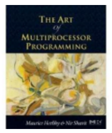
A runs, possibly writes to the register, enters CS



Proof: Assume Cover of 1

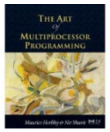


B Runs, first obliterating any trace of A, then also enters the critical section

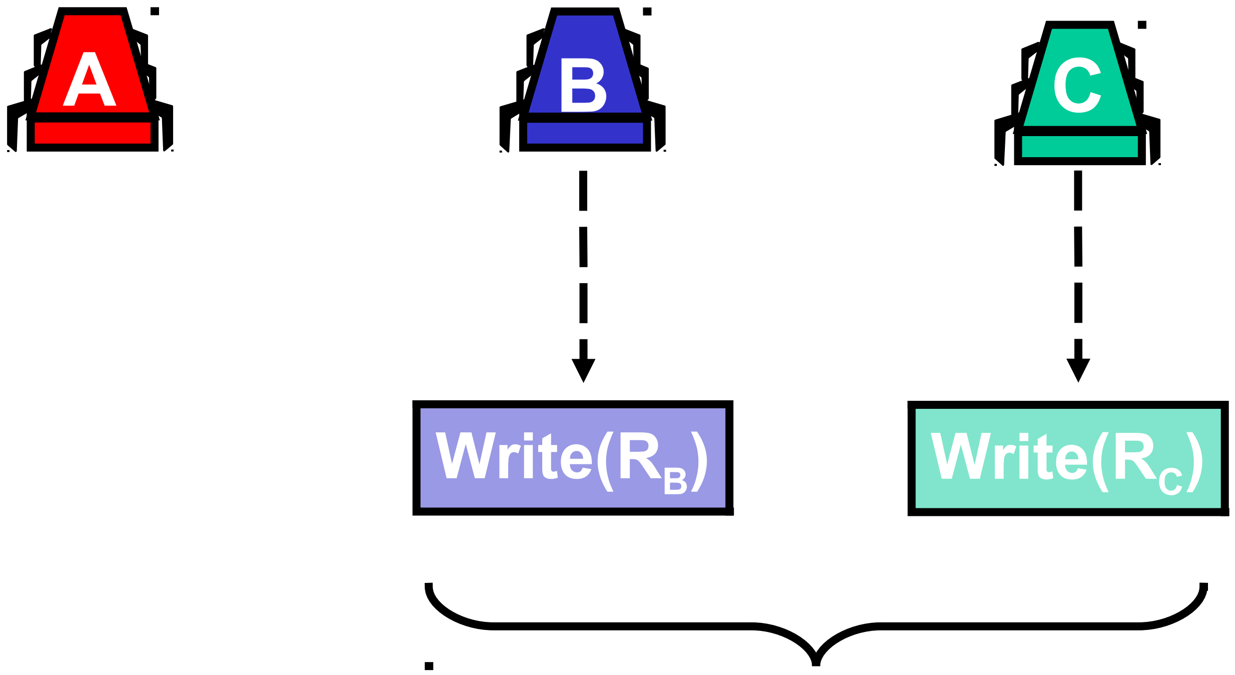


Theorem

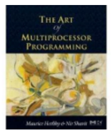
Deadlock-free mutual exclusion for 3 threads requires at least 3 multi-reader multi-writer registers



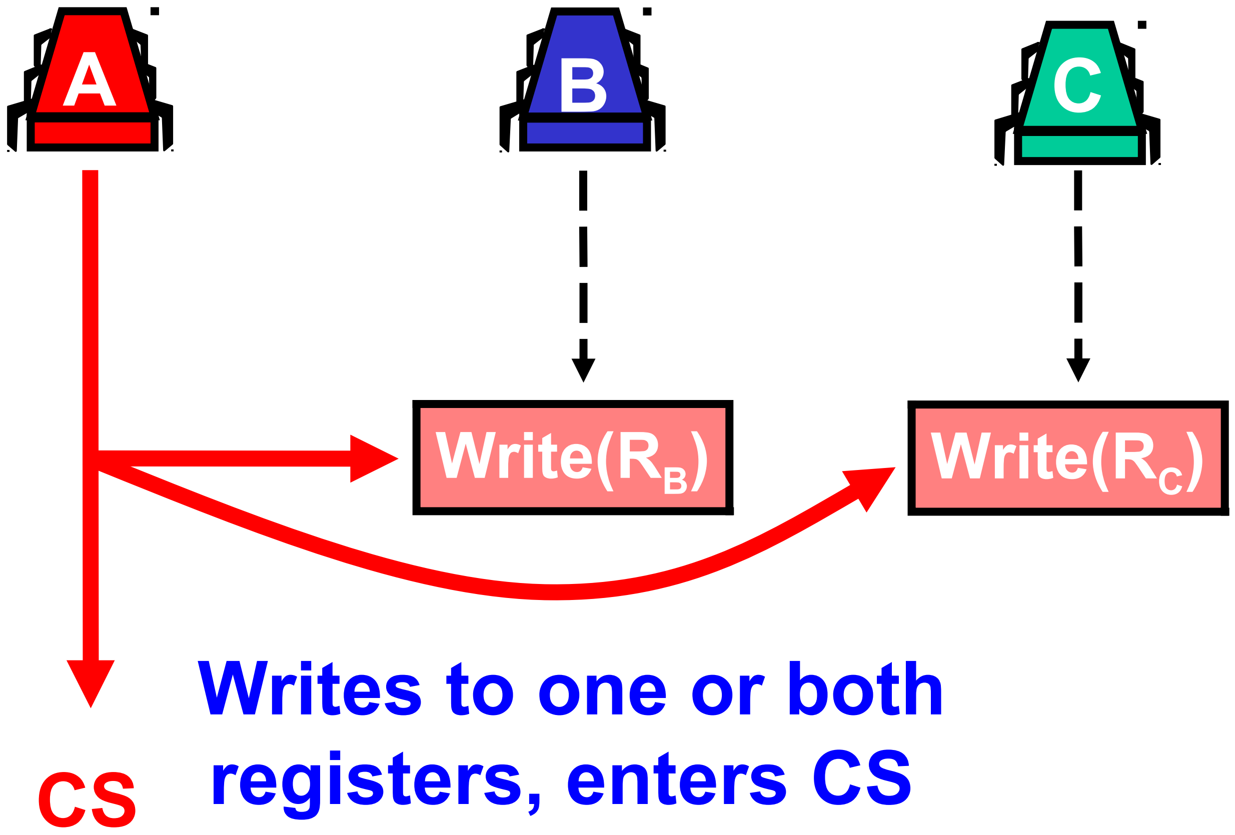
Proof: Assume Cover of 2



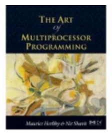
Only 2 registers



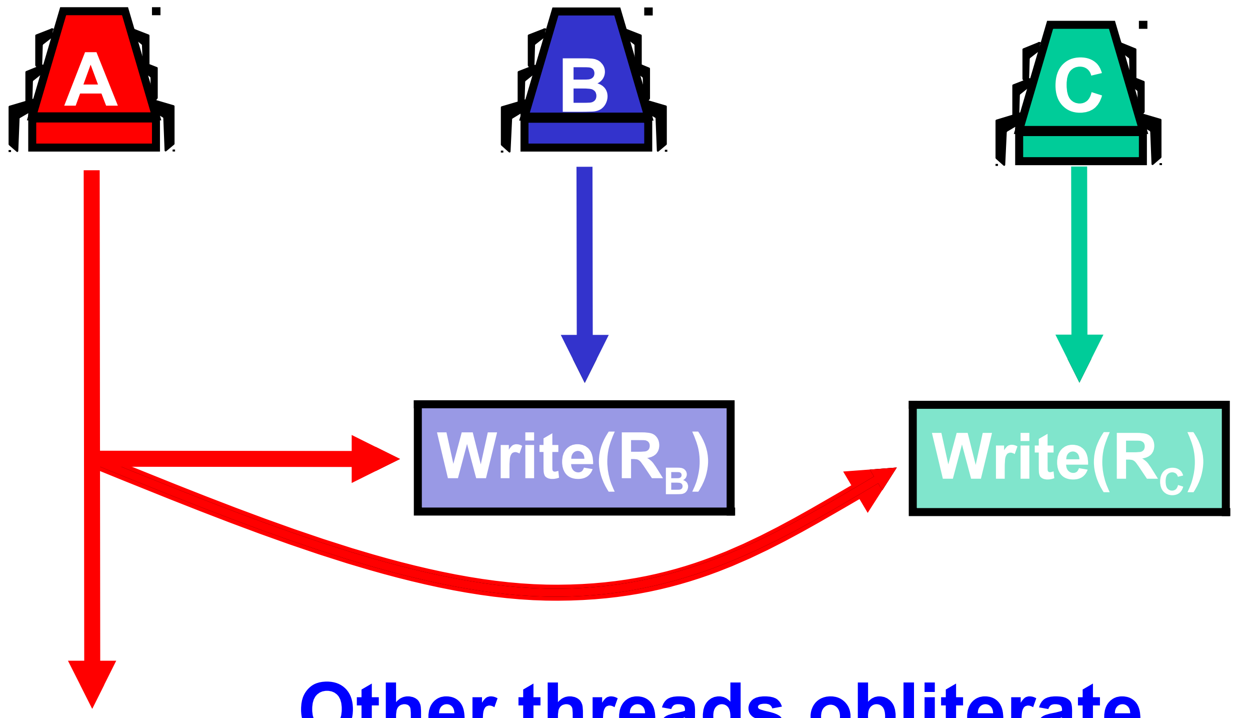
Run A Solo



Writes to one or both registers, enters CS

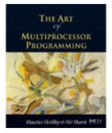


Obliterate Traces of A

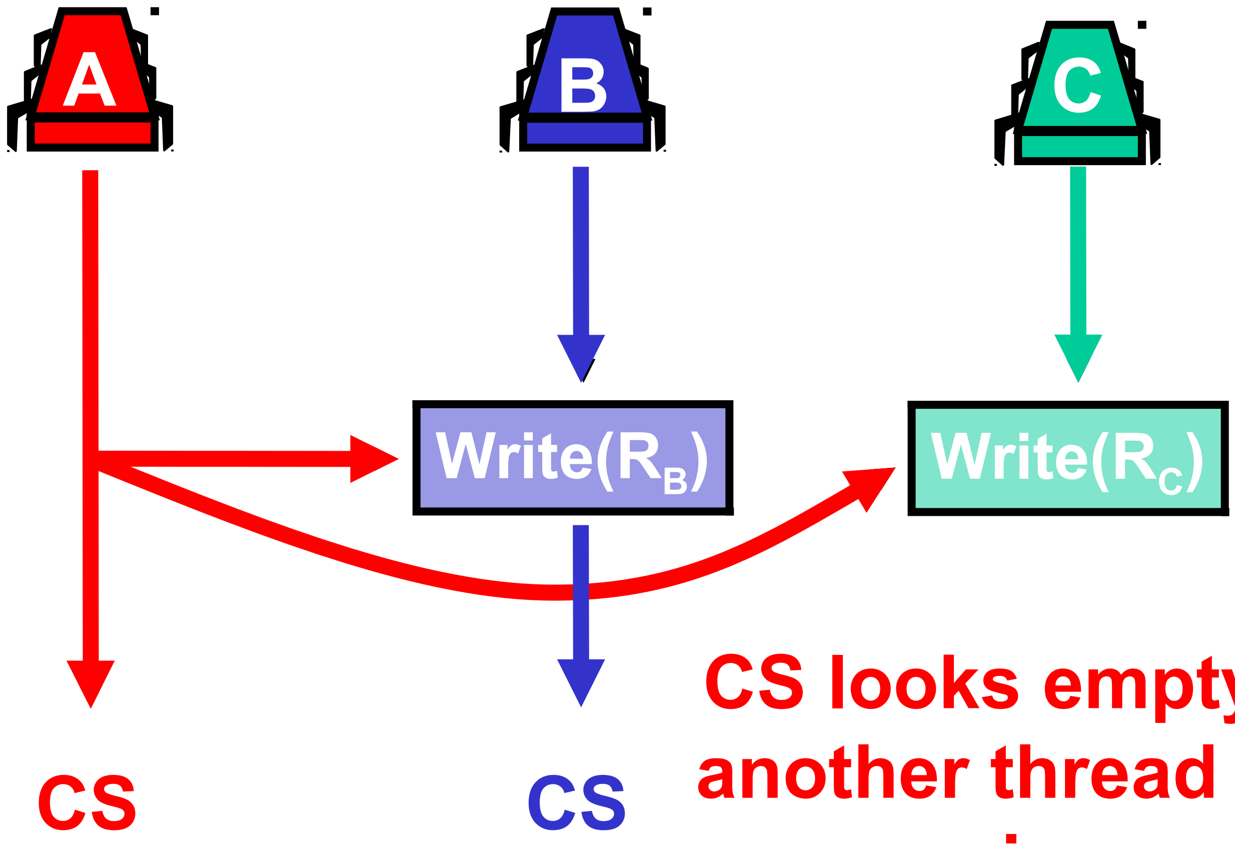


CS

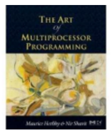
Other threads obliterate evidence that A entered CS



Mutual Exclusion Fails

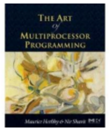


CS looks empty, so another thread gets in

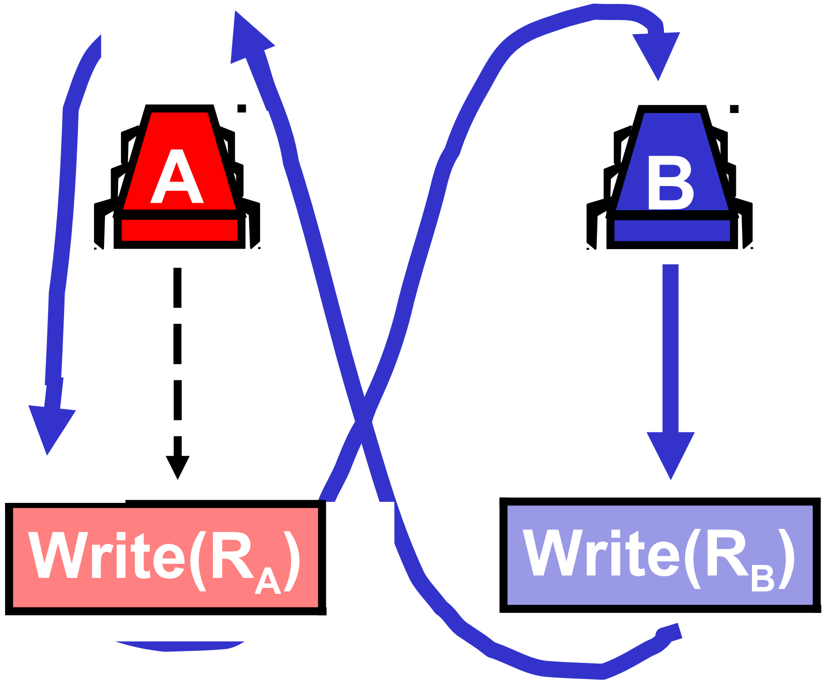


Proof Strategy

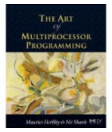
- **Proved**: a contradiction starting from a covering state for 2 registers
- **Claim**: a covering state for 2 registers is reachable from any state where CS is empty



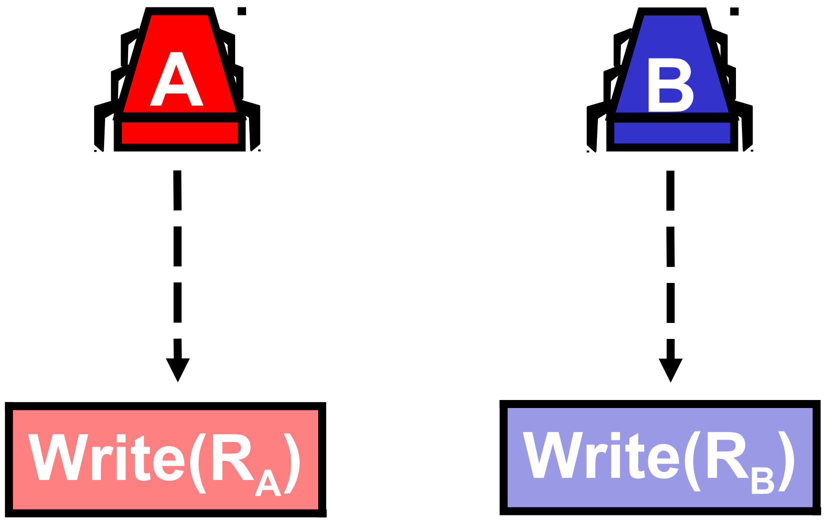
Covering State for Two



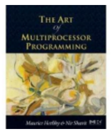
- If we run B through CS 3 times, B must return twice to cover some register, say R_B



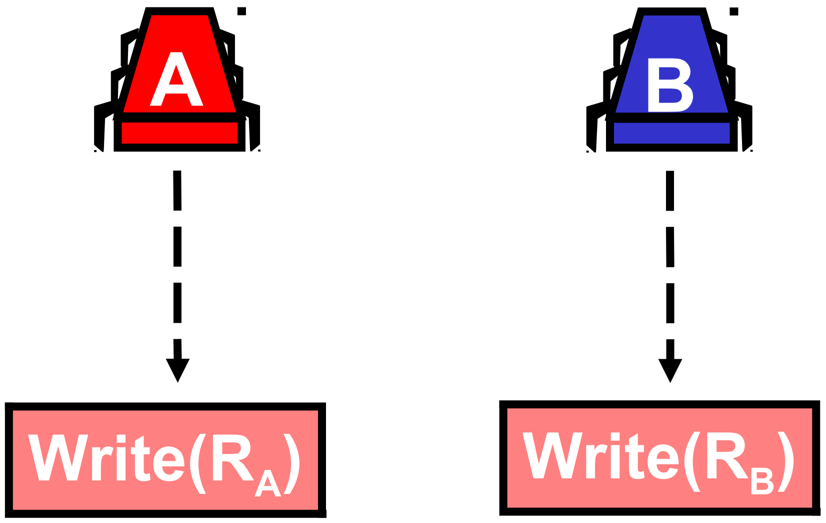
Covering State for Two



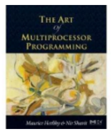
- Start with B covering register R_B for the 1st time
- Run A until it is about to write to uncovered R_A
- Are we done?



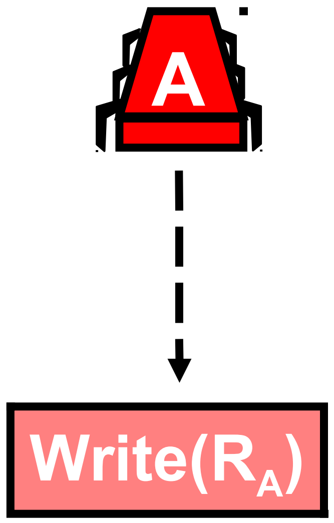
Covering State for Two

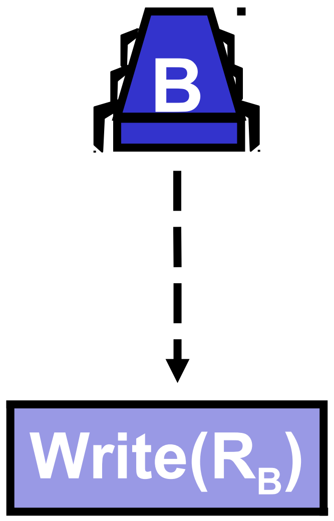


- NO! A could have written to R_B
- So CS no longer looks empty to thread C

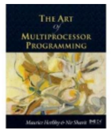


Covering State for Two

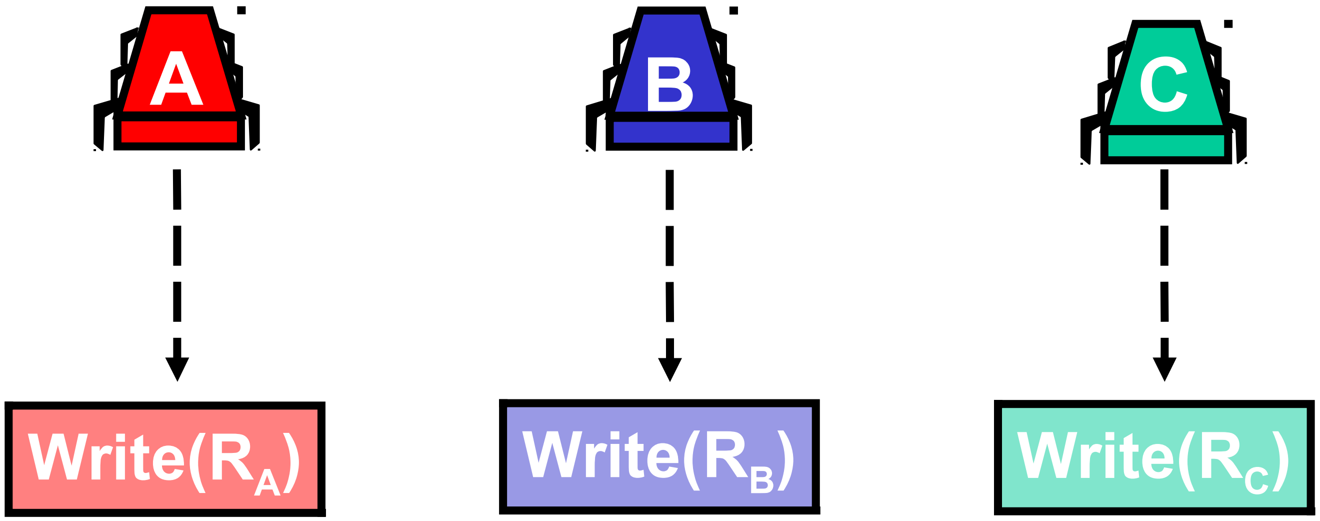




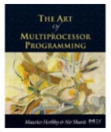
- Run B obliterating traces of A in R_B
- Run B again until it is about to write to R_B
- Now we are done



Inductively We Can Show

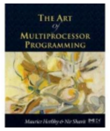


- There is a covering state
  - Where k threads not in CS cover k distinct registers
  - Proof follows when k = N-1



Summary of Lecture

- In the 1960's several incorrect solutions to starvation-free mutual exclusion using RW-registers were published…
- Today we know how to solve FIFO N thread mutual exclusion using 2N RW-Registers



Summary of Lecture

- N RW-Registers inefficient
  - Because writes "cover" older writes
- Need stronger hardware operations
  - that do not have the "covering problem"