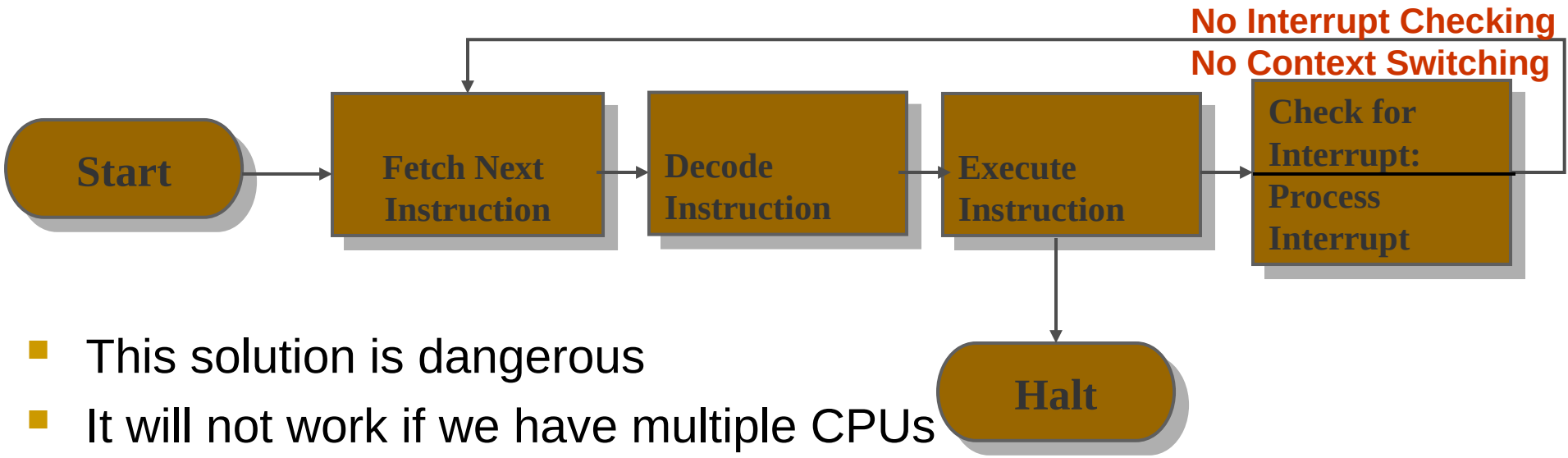
TSL & Priority Inversion

Hardware Solutions

- Disabling/Enabling Interrupts



- This solution is dangerous
- It will not work if we have multiple CPUs
- Since, the Disable Interrupt instruction will execute on only one CPU
- Rest of the CPUs, will continue checking the interrupts

FLAG = FALSE

ware Lock

The solution will work, if Testing and Setting the Flag are atomic

Process 1

```
1.while (FLAG == FALSE);
2.FLAG = FALSE;
3.critical_section();
4.FLAG = TRUE;
5.noncritical_section();
```

Process 2

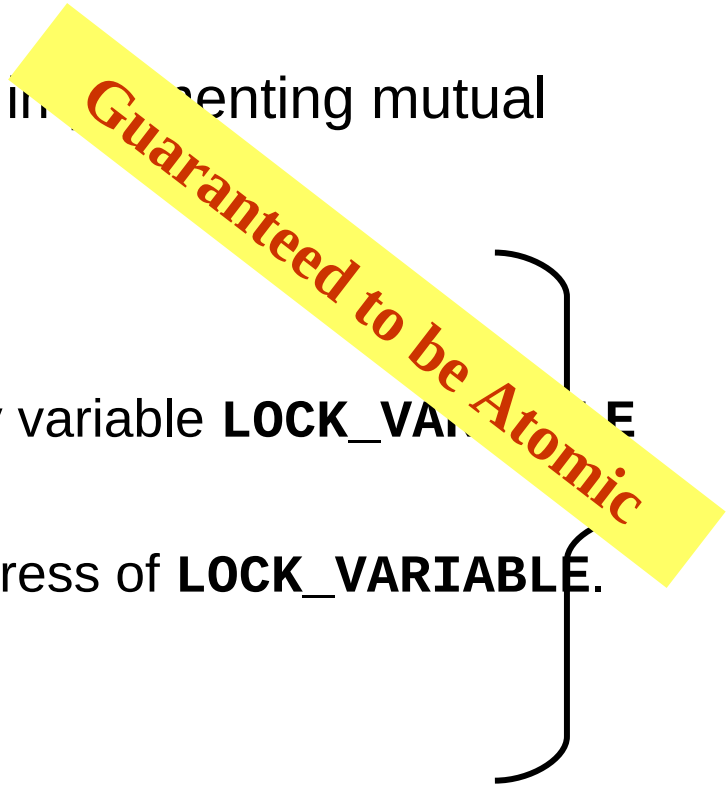
```
1.while (FLAG == FALSE);
2.FLAG = FALSE;
3.critical_section();
```

Timeout

No two processes may be simultaneously inside their critical sections

Hardware Lock

- Some instruction sets assist us in implementing mutual exclusion on multiple processors
- Test and Set Lock (TSL)
- **`TSL Rx, LOCK_VARIABLE`**
  - Reads the contents of the memory variable **`LOCK_VARIABLE`**
  - Stores it in the **`Rx`** register
  - Stores back the value **`1`** at the address of **`LOCK_VARIABLE`**.



TSL is “Atomic”

- Atomic => Uninterruptible
- No other process can access the memory until TSL is complete
- The CPU executing the TSL instruction locks the memory bus
- Thus, prohibits others CPUs from accessing the memory

Mutual Exclusion Using TSL

```
1.TSL          Rx, LOCK;   //copy LOCK to register
                           //and LOCK = 1
2.cmp          Rx,0        //Check if Old value of
                           //LOCK was 0. (Rx == 0)
3.jne          Line1       //Jump if not equal to
                           //Line 1
4.Critical_Section();

5.mov          LOCK,0      //Lock == 0;
```

```
LOCK =0;

1.TSL           Rx, LOCK;     //copy LOCK to register
                              //and LOCK = 1
2.cmp           Rx,0          //Check if Old value of
                              //LOCK was 0. (Rx == 0)
3.jne           Line1         //Jump if not equal to
                              //Line 1
4.Critical_Section();
1.TSL           Rx, LOCK;     //copy LOCK to register
                              //and LOCK = 1
2.cmp           Rx,0          //Check if Old value of
                              //LOCK was 0. (Rx == 0)
3.jne           Line1         //Jump if not equal to
                              //Line 1

5.mov           LOCK,0 //Lock == 0;


4.Critical_Section();

5.mov           LOCK,0//Lock == 0;
```

Busy Waiting and Priority Inversion

- Peterson's Solution and TSL both solve the mutual exclusion problem
- However, both of these solutions sit in a tight loop waiting for a condition to be met (busy waiting).
- Wasteful of CPU resources

Busy Waiting and Priority Inversion

- Suppose we have two processes
  - One of high priority ***H***
  - One of low priority ***L***
- The scheduling algorithm runs ***H*** whenever it is in ready state
- Suppose
  - ***L*** is in its critical section
  - ***H*** becomes ready
  - ***L*** will be placed in a ready state
  - ***H*** will be in the run state

Busy Waiting and Priority Inversion

- But, the ***H*** will not be able to run
- Even ***L*** cannot run again to release ***H***
- This is sometimes known as the
  - **Priority inversion problem.**