Lecture 11
Introduction to Synchronization

EN 600.320/420

Instructor: Randal Burns

24 March 2014

Department of Computer Science, *Johns Hopkins University*

Synchronization

A look inside the critical section

Two common goals for synchronization

- Contention:
  - How to resolve the conflicts that result from multiple processes trying to access shared resources?
- Cooperation:
  - An action by one process may enable another action by another process
  - In such cases, processes should coordinate their actions

Why is synchronization hard?

- Design an algorithm for purchasing milk between two roomates Alice and Bob
- Steps:
  - Arrive home
  - Look in fridge for milk
  - Leave for grocery
  - Buy milk
  - Arrive home with purchased milk

Alice

- Arrive home
- Look in fridge for milk
- Leave for grocery
- Buy milk
- Arrive home with purchased milk

Bob

- Arrive home
- Look in fridge for milk
- Leave for grocery
- Buy milk
- Arrive home with purchased milk

Why is synchronization hard?

- Design an algorithm for purchasing milk between two roomates Alice and Bob
- Steps:
  - Arrive home
  - Look in fridge for milk
  - Leave for grocery
  - Buy milk
  - Arrive home with purchased milk
- Too much milk!
- Problem is impossible without communication between parties

Let's Try Using Notes

- *Algorithm #1:* If you find that there is no milk in fridge, leave a note on the door, go to store and purchase milk, on return home remove note

```
if (no note) then
  if (no milk) then
    leave note
    buy milk
    remove note
  fi
fi
```

They can't see each other

Alice

```
if (no note) then
  if (no milk) then
    leave note
    buy milk
    remove note
  fi
fi
```

Bob

```
if (no note) then
  if (no milk) then
    leave note
    buy milk
    remove note
  fi
fi
```

Let’s Try Using Notes

- *Algorithm #2:* Based on leaving a note (with one’s name) before checking fridge

```
leave note A
if (no note B) then
  if (no milk) then
    buy milk
  fi
fi
remove note
```

Alice

```
leave note A
if (no note B) then
  if (no milk) then
    buy milk
  fi
fi
remove note
```

Bob

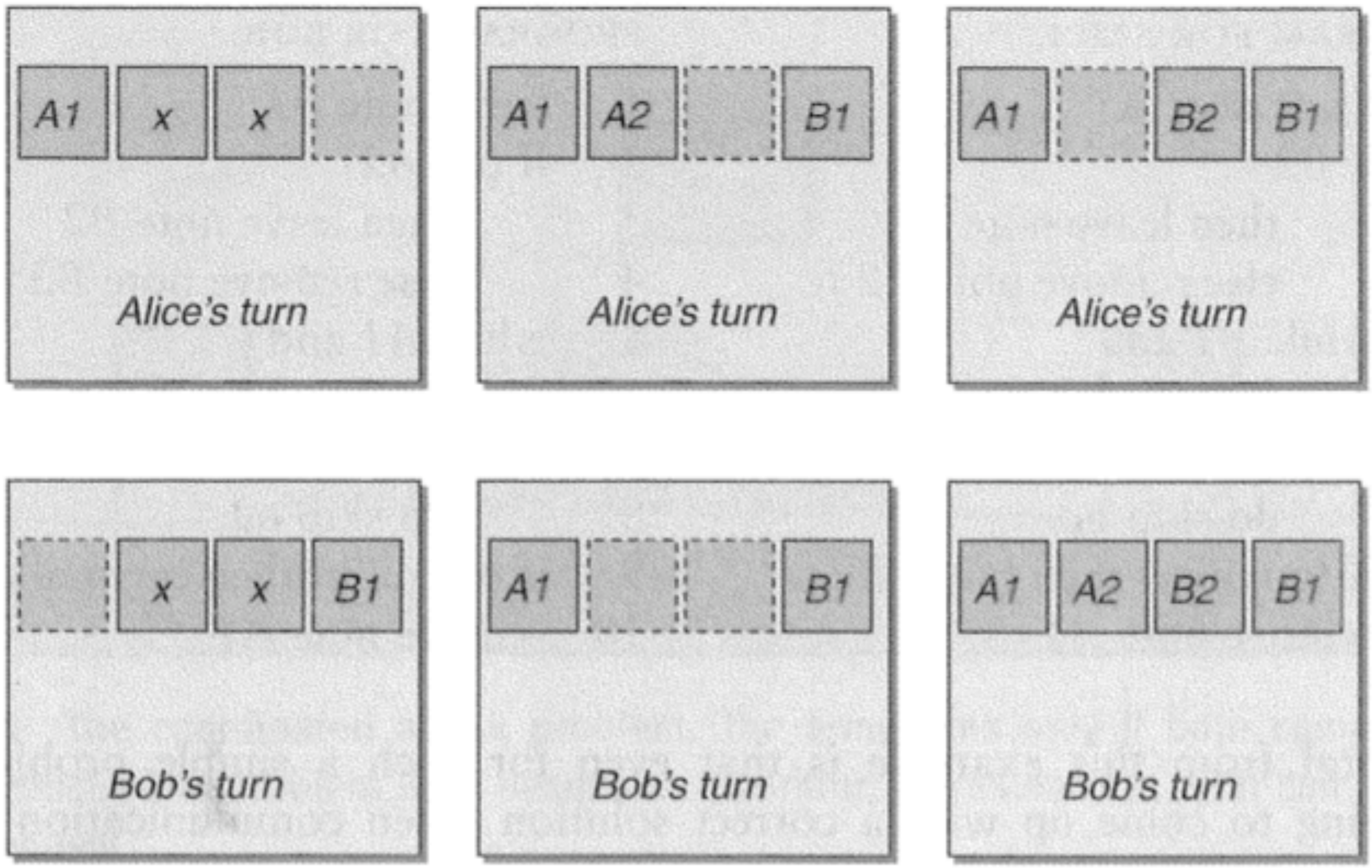
```
leave note B
if (no note A) then
  if (no milk) then
    buy milk
  fi
fi
remove note
```

A Correct Algorithm

```
leave note A1
if (B2)
  then leave note A2
  else remove note A2 fi
while B1 and
  ((A2 and B2) or
   (no A2 and no B2))
  do skip do
if (no milk)
  then buy milk fi
remove note A1
```

```
leave note B1
if (no A2)
  then leave note B2
  else remove note B2 fi
while A1 and
  ((A2 and no B2) or
   (no A2 and B2))
  do skip do
if (no milk)
  then buy milk fi
remove note A1
```

Possible Configurations



Two Notes

- First one to identify contention
  - Are two parties vying for this resource
- Second one to break ties during contention
  - Essentially even and odd configurations

- These notes are the analogies of atomic shared registers in computing
  - Essentially a volatile variable of basic type

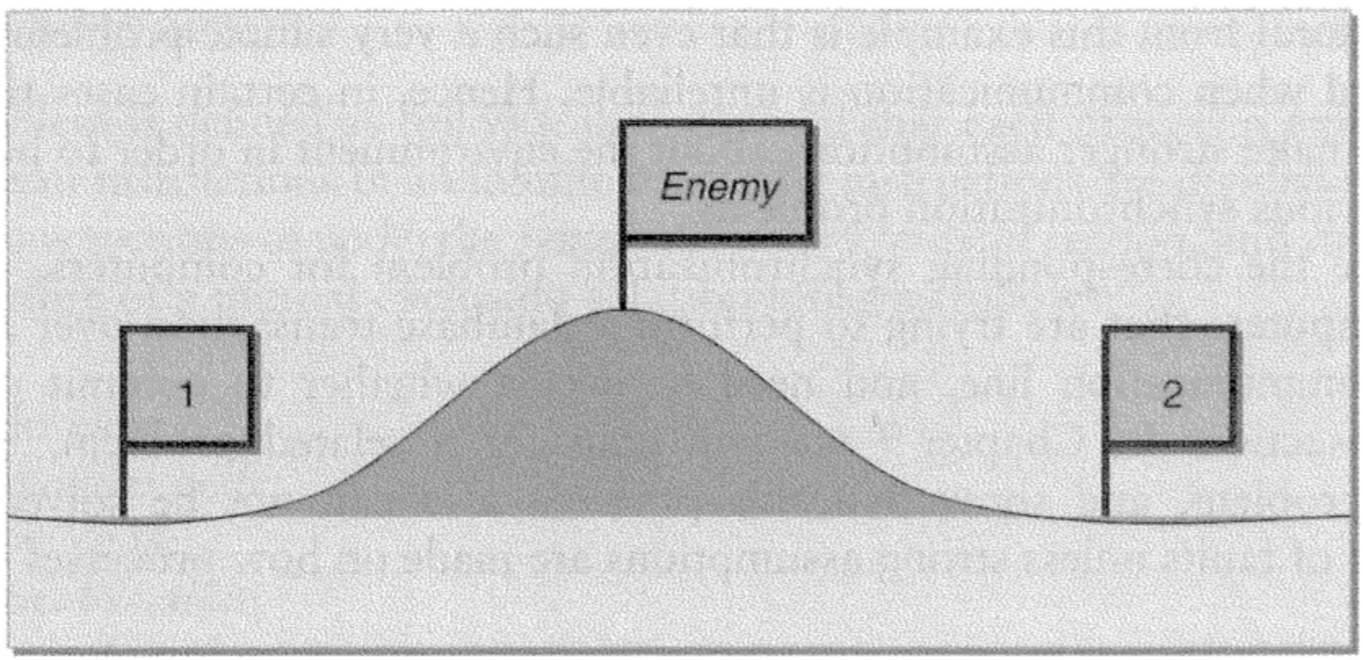
Some Properties

- Correct
- Asynchronous: doesn't depend on timing
- Symmetric: equal chance of A/B buying milk
  - Notably steps aren't symmetric
- Two parties

- Even simple synchronization is hard and subtle

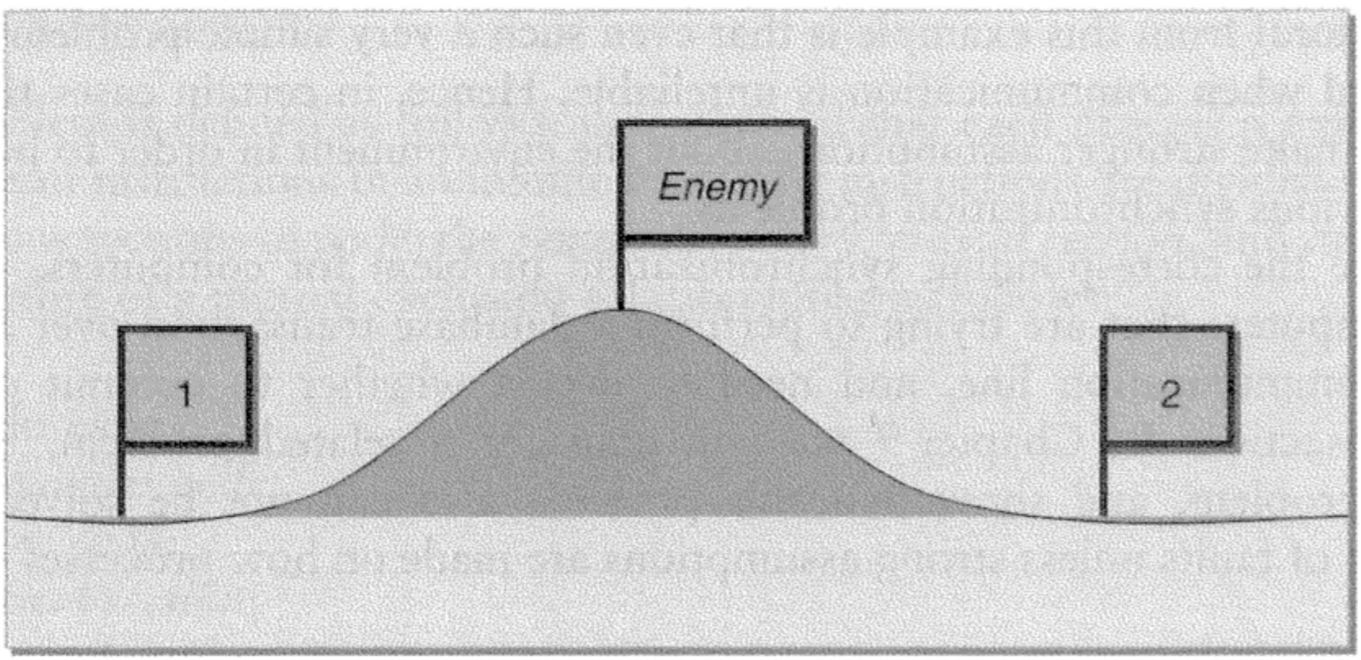
The Coordinated Attack Problem

- Jim Gray, 1978.
  - Armies only defeat the enemy if both parties attack simultaneously
  - Can only communicate with unreliable runners



The Coordinated Attack Problem

- Impossible:
  - Difficulty of distributed computing

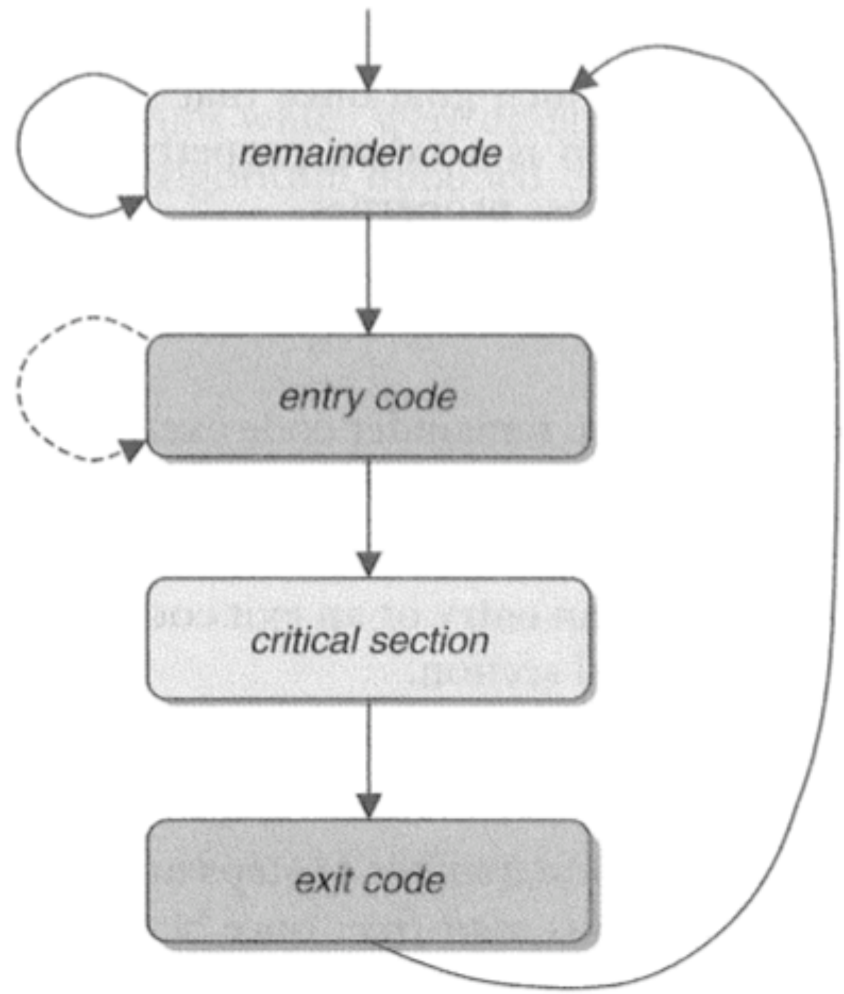


The Point Again

- Going to take a somewhat more formal look at synchronization
  - Not just present the constructs
- Synchronization issues are the major bug in parallel programs
  - Deadlock
  - Incorrect results
- The constructs/algorithms underlying critical sections, locks, atomic variables are complex
  - Understanding them will help you use them well

Mutual Exclusion (2 processes)

- Guarantees
  - Exclusive access to a shared resource among competing processes
  - No deadlocks
  - Starvation resistance (must make progress eventually)
- Core problem of synchronization



Peterson's Algorithm

PROGRAM FOR PROCESS 0:

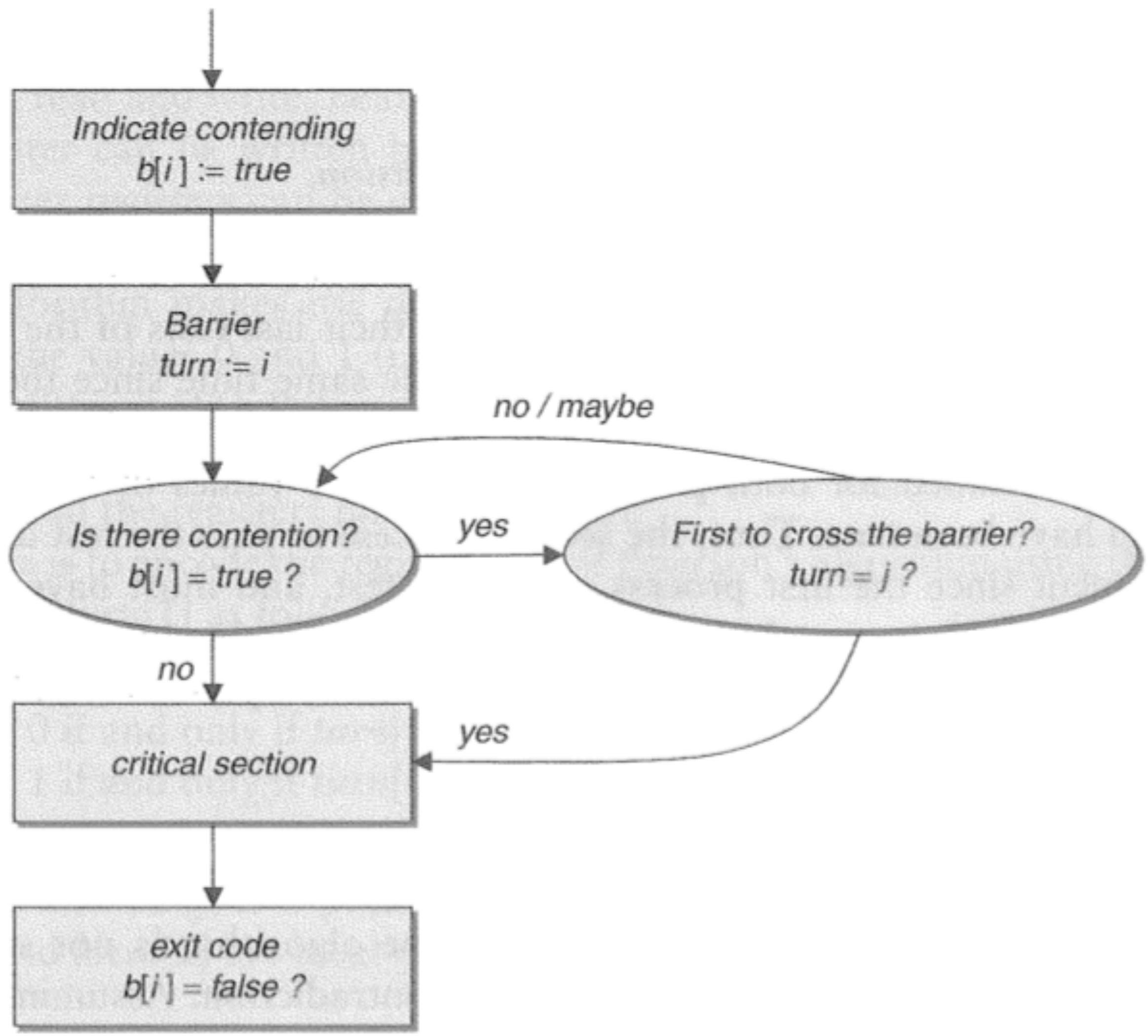
```
b[0] := true;
turn := 0;
await (b[1] = false or turn = 1);
critical section;
b[0] := false;
```

PROGRAM FOR PROCESS 1:

```
b[1] := true;
turn := 1;
await (b[0] = false or turn = 0);
critical section;
b[1] := false;
```

- b[x] indicates process b's desire for resource x
- Write to turn indicates who got there first
- Wait for other party to either
  - Give priority (through turn)
  - Not desire

Peterson's Algorithm



Properties of Petersons's Alg.

- Mutual exclusion
- Starvation resistant
- Contention free overhead = 4 accesses
- Arbitrary waits (non-preemptive)
- Uses three shared registers

- Requires *atomic* registers
  - Doesn't work in message passing environments
  - Need a simple modification

On busy waiting

- The *await* construct in Peterson's algorithm *busy waiting* aka *spinning*
  - Use an active processor to poll the state of a memory location
- This is a good construct when:
  - There are many processors
  - There is no other useful work to do
  - Wait periods are very short
- The alternative is to sleep/restart
  - Typically implemented by hardware interrupts
  - More overhead to start/stop,
  - Frees hardware for processing of other tasks
- *Do power constraints change this?*

Fast Mutual Exclusion

- Now let's scale this to *n* processes

Leslie Lamport Receives Turing Award

Microsoft Research
March 18, 2014 6:00 AM PT

Like 1.4k Tweet 1,285

Leslie Lamport first began dabbling in computers while he was still in high school. Nothing too unusual about that—until you consider that this was in the mid-1950s. Lamport was attending the Bronx High School of Science in New York, and he and a friend used to scrounge around, looking for discarded vacuum tubes to build a digital circuit.

The path to greatness begins with baby steps, and for Lamport, a principal researcher with Microsoft Research, that teenage curiosity has yet to be quenched. Over the ensuing decades, he has become a veritable legend in computing circles. His work in the theory of distributed computing is foundational. His 1978 paper *Time, Clocks, and the Ordering of Events in a Distributed System*

Publications

- *Time, Clocks, and the Ordering of Events in a Distributed System*
- *The Maintenance of Duplicate Databases*
- *A New Solution of Dijkstra's Concurrent Programming Problem*
- *The Byzantine Generals Problem*
- *The Part-Time Parliament*
- *Paxos Made Simple*
- *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*

Awards

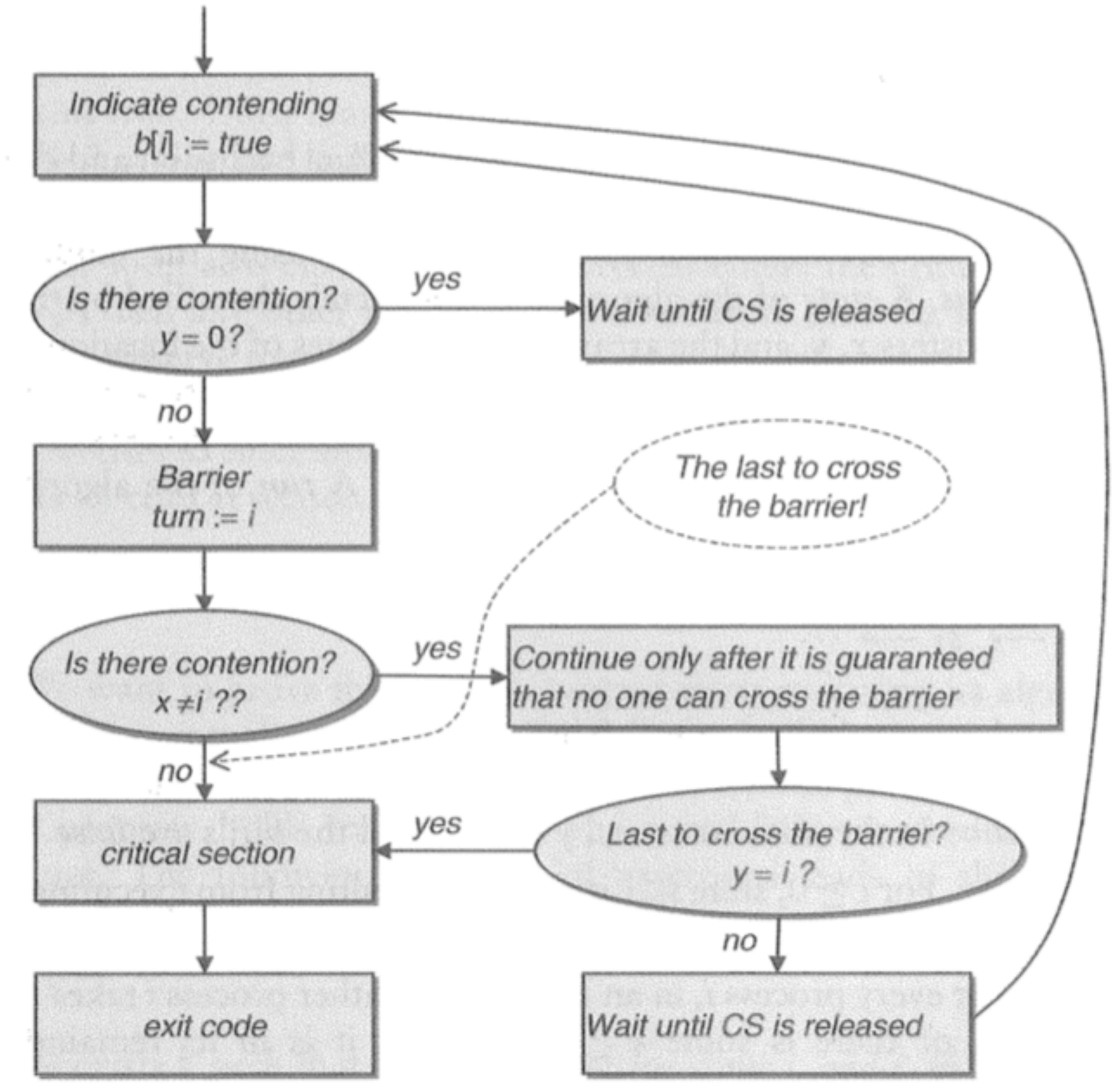
- A.M. Turing Award

Fast Mutual Exclusion

- Lamport 1987

```
start:  b[i] := true;
        x := i;
        if y ≠ 0 then b[i] := false;
                      await y = 0;
                      goto start fi;
        y := i;
        if x ≠ i then b[i] := false;
                      for j := 1 to n do await ¬b[j] od;
                      if y ≠ i then await y = 0;
                                    goto start fi fi;
       critical section;
       y := 0;
       b[i] := false
```

Fast Mutual Exclusion



Fast MutEx Properties

- Mutual exclusion/deadlock freedom
- Contention free overhead= 7 accesses
- **Starvation is possible!** (of any process)
  - Unbounded wait times

Practical Concerns

- None of the algorithms are used in practice
  - Too simple
  - H/W support
- But demonstrate fundamental tradeoffs
  - Space (# shared registers), speed, fairness (bounded waiting)
- All of these algorithms rely on *atomic* registers
  - Not available in distributed memory machines, which leads to whole new families of protocols