Lecture 12
Safety and Liveness

EN 600.320/420

Instructor: Randal Burns

31 March 2014

Department of Computer Science, *Johns Hopkins University*

Characterizing Concurrency

- Safety (correctness): what are the semantic guarantees (invariants) expressed by a locking protocol under concurrent execution
  - Typically related to some notion of serial execution

- Liveness (progress): how can the execution of one thread be delayed by other threads
  - **Starvation Freedom:** *If a process is trying to enter its critical section, then this process must eventually enter its critical section*
  - Many algorithms ignore starvation freedom on the premise that contention is rare

Ideal Properties

Safety

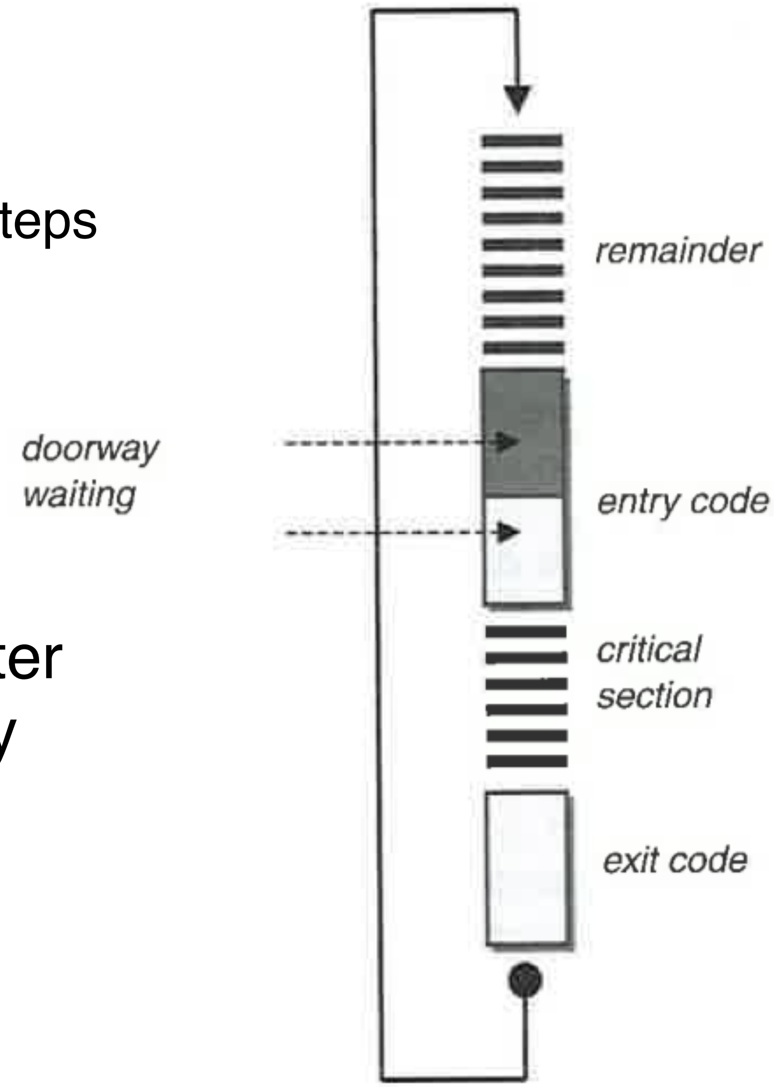
- FIFO: all operations execute serially
  - Concurrent execution does not change the semantics of computing

Liveness

- Non-blocking

Waiting

- *Doorway* is *wait free*
  - Bounded number of atomic steps
- Waiting is *busy waiting* on some condition
- Notions of waiting:



r-bounded: a process will enter its critical section before every other process executes r+1 critical sections

- **FIFO** is 0-bounded waiting

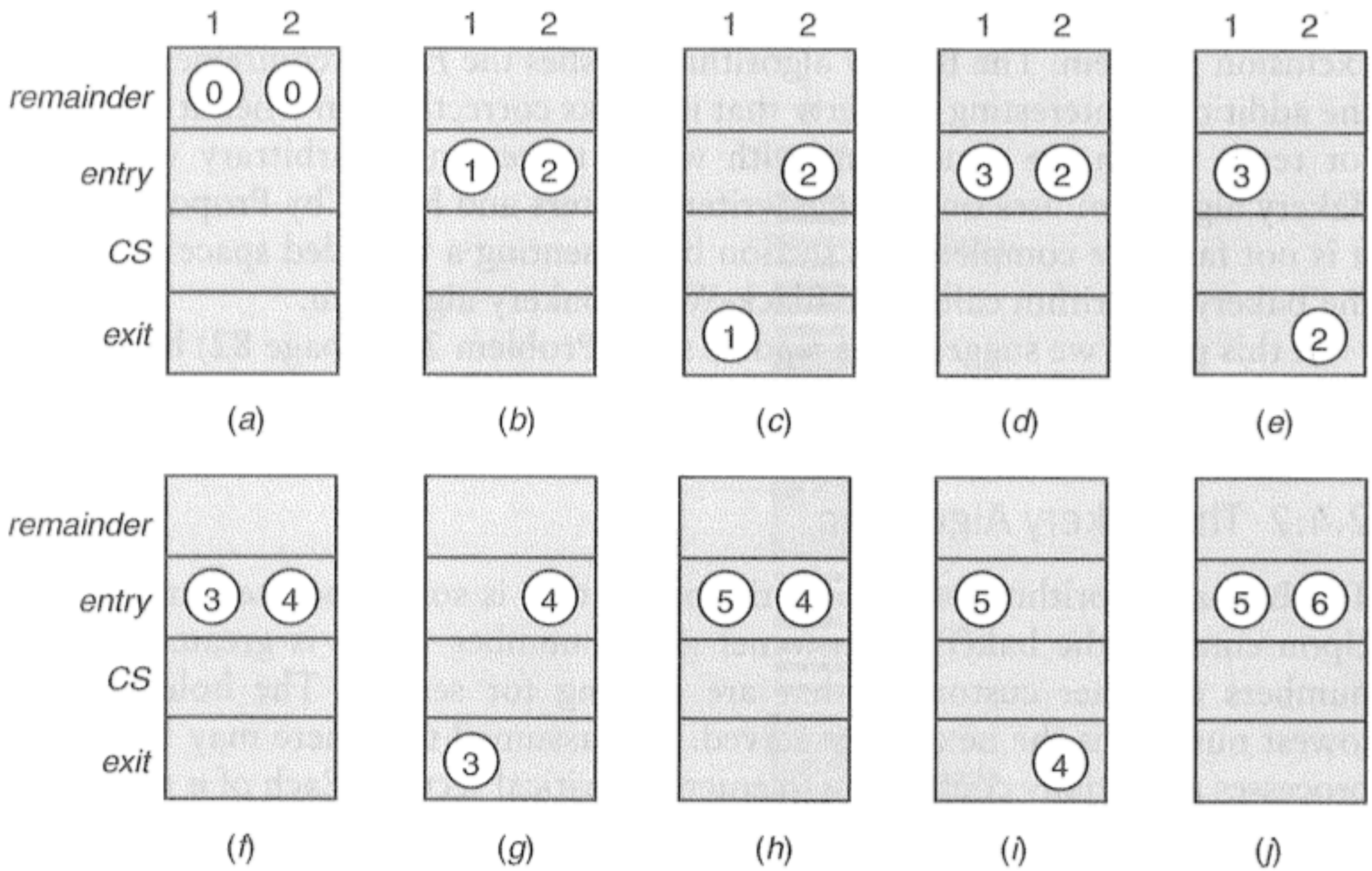
The Bakery Algorithm

- FIFO processing for Mutual Exclusion

Initially: all entries in *choosing* and *number* are *false* and 0, respectively.

```
choosing[i] := true;
number[i] := 1 + maximum(number[1], . . . , number[n]);
choosing[i] := false;
for j = 1 to n do
        await choosing[j] = false;
        await (number[j] = 0 or (number[j], j) ≥ (number[i], i))
od;
critical section;
number[i] := 0;
```

The Bakery Algorithm



Observations about Bakery

- Code parts
  - Lines 1-3 are doorway
  - Lines 4-7 are waiting
  - Line 9 is exit
- Boolean array *choosing[j]* and integer array *number[j]*
  - Read by all processes
  - Written only by process j
- Uses lexicographic ordering of nodes as well as ticket numbers
  - (a,b)<(c,d): a<c or (a=c and b<d)

Properties of Bakery

- FIFO: no process that enters the *doorway* gets ahead of a process that has already started *waiting*
- Satisfies mutual exclusion
- Is not fast!!!
  - 3(n-1) memory accesses when there is no contention
  - This is why Fast Mutual Exclusion is so cool
    - Exchange FIFO for constant overhead
    - Don’t read other processes state if there is not contention

The Next Layer of Concepts

- Variants on number of processes
  - Infinitely many processes
  - Sparse process id address space (symmetric algs.)
- Spinning on local registers only
- For different memory models
  - CC, DSM

Building on Primitives

- Common atomic operations (building blocks):
  - Read
  - Write
  - Test-and-set
  - Swap
  - Fetch and add (fetch and increment)
  - Read-modify-write
  - Compare-and-swap

Test-and-Set Bit

- Two operations
  - Reset: write 0
  - Test and set: write 1 and return old value
- Trivial deadlock free synchronization

```
await (test-and-set(x) = 0);
critical section
reset(x);
```

- This is called a *spin lock*
  - Mutual exclusion, deadlock free
  - Not starvation resistant

Test-and-Test-and-Set Bit

- Test-and-set alg. writes bit every iteration
  - Invalidates caches even when data don't change
- Test-and-test-and-set
  - Supports test w/out set
- Produces fewer cache misses
  - What's the miss pattern during contention

```
await (x=0);
while (test-and-set(x) = 1) do
  await (x=0) od;
critical section
reset(x);
```

What’s wrong with Spin Locks?

- Every process spins on shared state
- When lock is freed, all processes attempt to acquire
- Performance varies with contention:
  - Low contention good (simple algorithms)
  - High contention bad (burst of activity: messages and cache invalidations)
- Can be addressed with backoff policies
  - Like exponential backoff in TPC
- But, queuing is better

Ticket Algorithm

- Bakery algorithms using read-modify-write
  - < and > indicate RMW boundaries

THE TICKET ALGORITHM: process i's program.

constant $N = \{0, 1, \ldots, n-1\}$
shared $(ticket, valid)$ a read-modify-write register ranges over $N \times N$;
  initially $ticket = valid$
local $(ticket_i, valid_i)$ ranges over $N \times N$

$\langle (ticket_i, valid_i) := (ticket, valid);$
  $ticket := (ticket + 1) \bmod n \rangle;$
while $ticket_i \neq valid_i$ **do**
    $\langle valid_i := valid \rangle$ **od**;
critical section;
$\langle valid := (valid + 1) \bmod n \rangle;$

Properties of RMW Ticket Alg.

- FIFO: in the order of successful RMW
- Mutual exclusion and deadlock freedom
- Uses one shared register that holds n^2 values

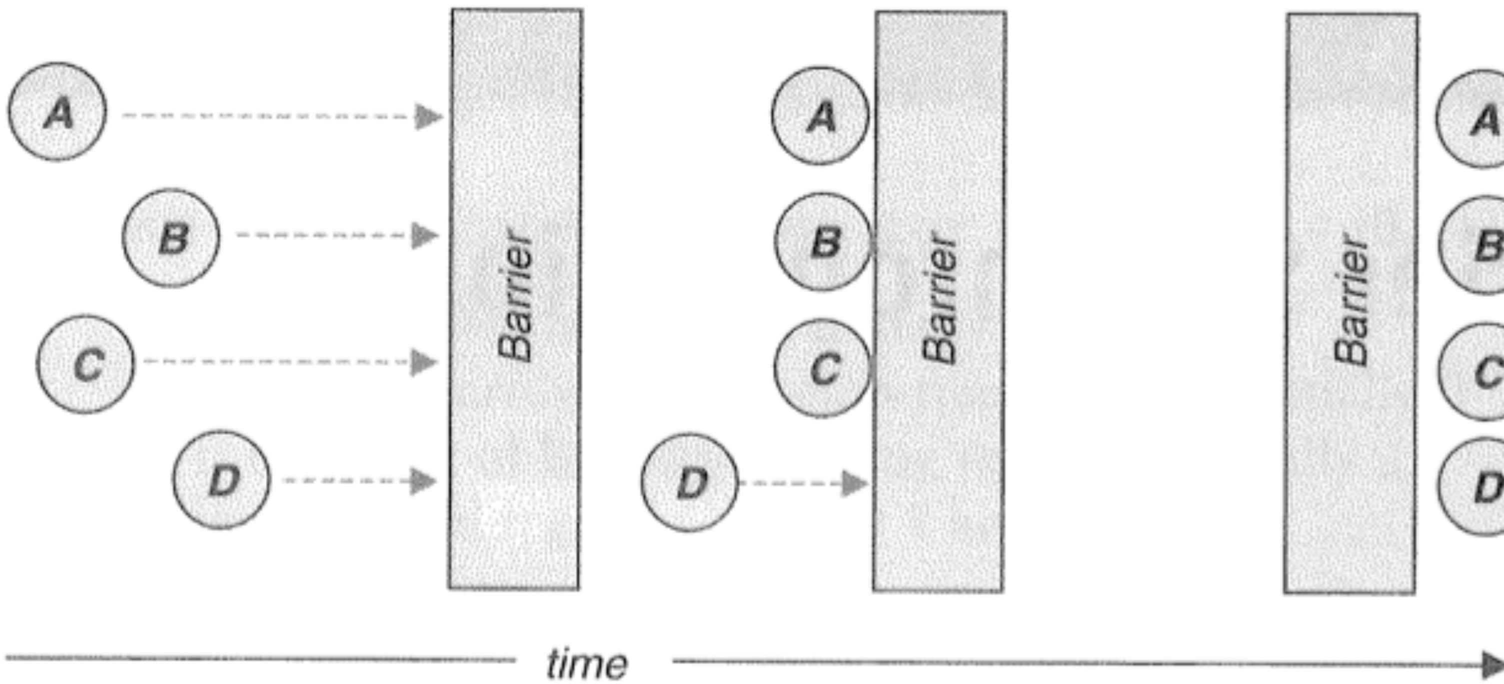
- This is the power of H/W support
  - Modern processors provide some variant of RMW

Waiting w/out the Busy Wait

- The Semaphore *S*
  - up(*S*) increase the value of *S*
  - down(*S*) decrease the value of *S*
  - Binary semaphore takes values 0 and 1
- Using the semaphore
  - down(*S*); critical section; up(*S*);
  - To realize deadlock-free, mutual exclusion
- Where does the busy wait go?
  - Nowhere: implement semaphores with test-and-set
  - Into the kernel: one process does all the busy waiting
  - Into hardware: use interrupts

Barriers

- Allows a “synchronous” algorithm to run on asynchronous hardware



Simple Barrier

- Built on an atomic counter and atomic bits

shared *counter*: atomic counter ranges over $\{0, \ldots, n\}$, initially 0
go: atomic bit, initial value is immaterial
local *local.go*: a bit, initial value is immaterial

```
local.go := go                          /* remembers current value */
counter := counter + 1
                          /* atomically increment the counter */
if counter = n then          /* last to arrive to the barrier */
    counter := 0                       /* reset the barrier */
    go := 1 − go                              /* notify all */
else await(local.go ≠ go) fi     /* not the last to arrive */
```

Multiple Resources

- To now, we have talked about deadlock freedom for mutual exclusive access to a single resource

- With multiple resources, we get deadlock even with deadlock-free access to each resource

Deadlock

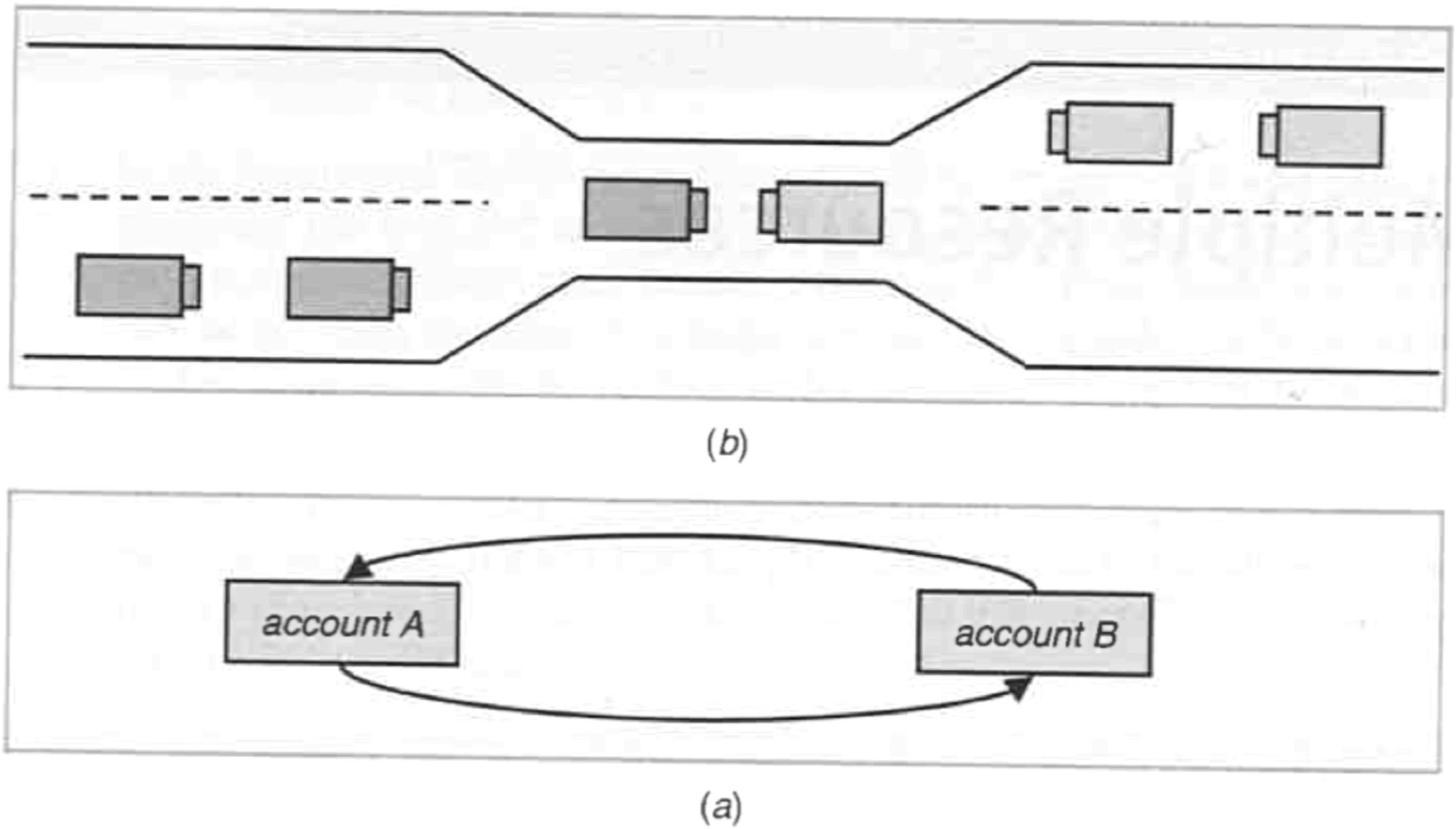


Figure 7.1 Examples of deadlocks. (*a*) Transferring money between two accounts. (*b*) Bridge crossing: the resources are the two entrances.

Deadlock

- *Def'n:* A set of processes are deadlocked if each process in the set is waiting for an event that only another process in the set can cause
- Requirements (all four must hold simultaneously)
  - Mutual exclusion
  - Hold and wait
  - No preemption
  - Circular wait
- We'll do more on deadlock in MPI