Lecture 10.1
Safety and Liveness

EN 600.320/420

Instructor: Randal Burns

28 February 2018

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

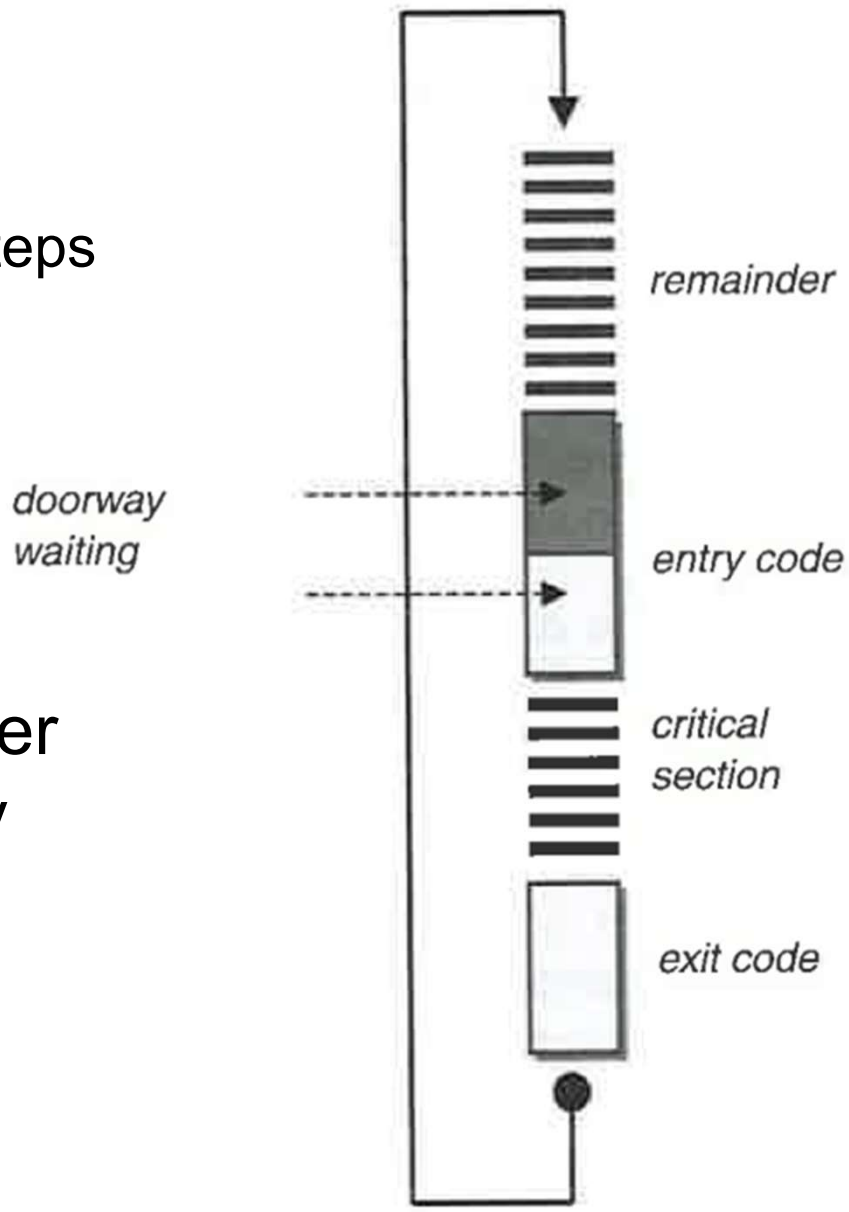
- FIFO: all operations execute serially
  - Concurrent execution does not change the semantics of computing

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

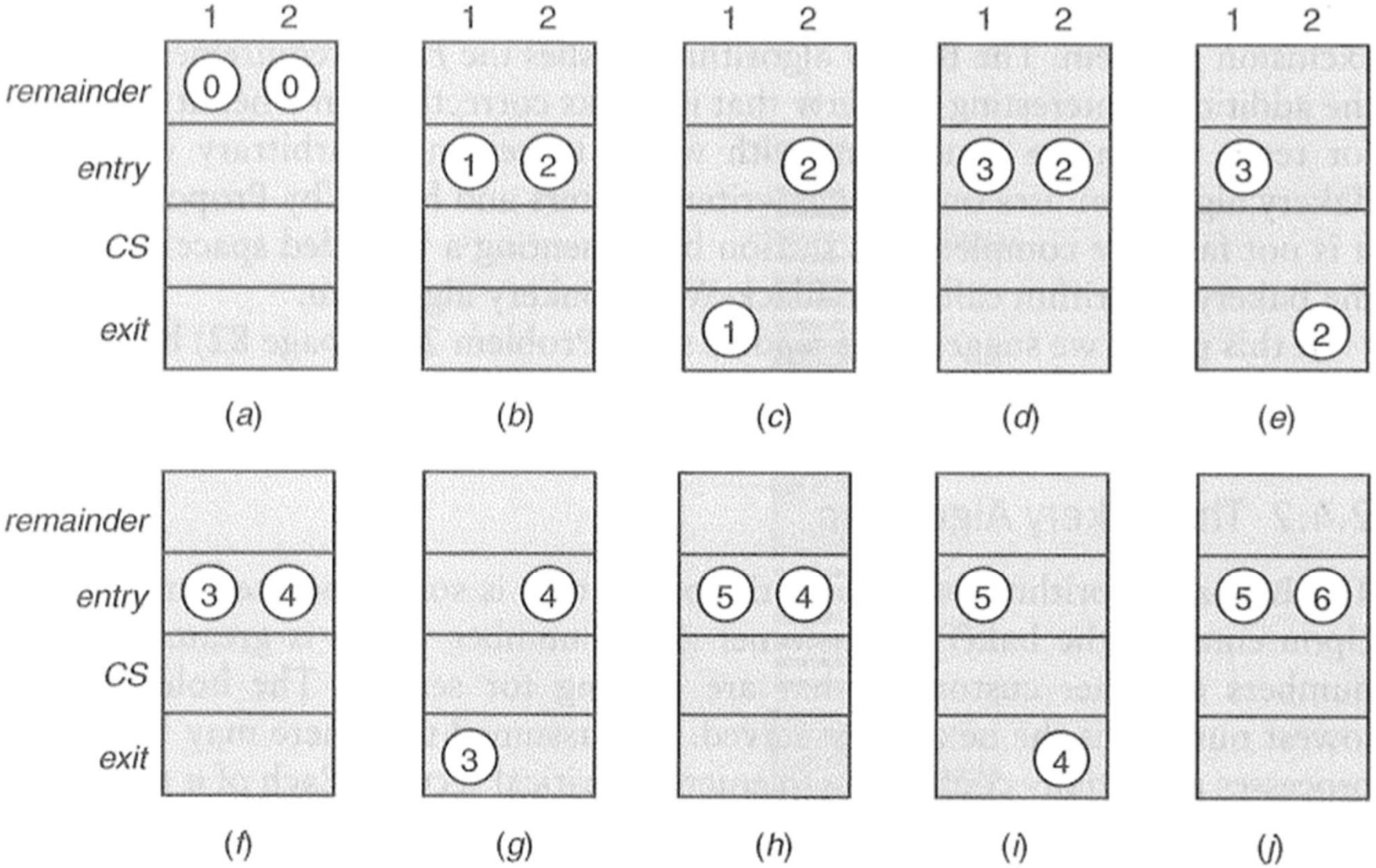
The Bakery Algorithm

- FIFO processing for Mutual Exclusion

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

```
1    choosing[i] := true;
2    number[i] := 1 + maximum(number[1], ..., number[n]);
3    choosing[i] := false;
4    for j = 1 to n do
5            await choosing[j] = false;
6            await (number[j] = 0 or (number[j], j) ≥ (number[i], i))
7    od;
8    critical section;
9    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
- Fast mutual exclusion is *optimistic* that there won't be any contention and sacrifices fairness for efficiency