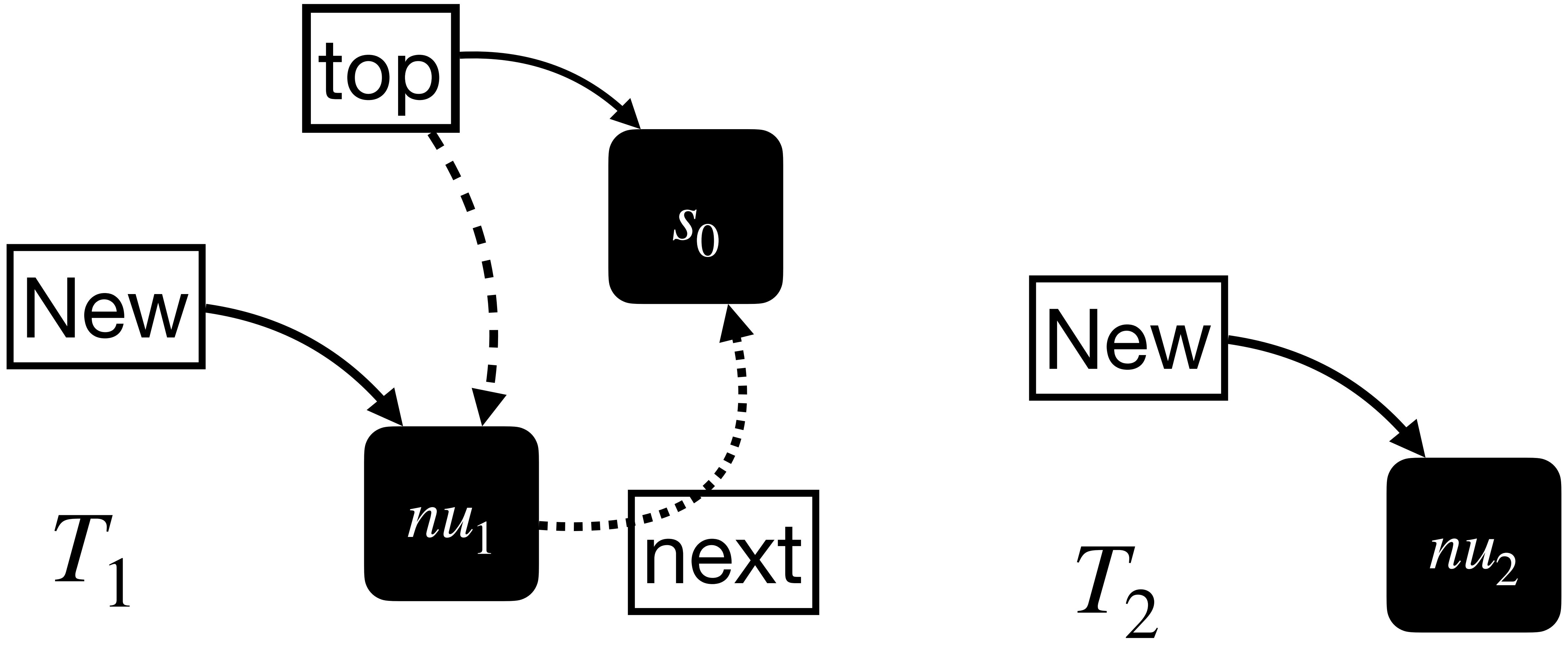
Introduction to Parallel & Distributed Programming

Lec 12— Mutual Exclusion

Subodh Sharma | Feb 09, 2026

RECAP: Implementing Lock-free Stack: FAS & CAS




```
class LockFreeStack{

…

  void  push (T value){

    Node ∗ new = new Node (value);

    Node ∗ old_top = top.load();

    do{

      new->next = old_top;

    } while(!top.compare_and_exchange(old_top,
    new))

    size.fetch_add(1);

  }
```

RECAP: Synchronisation

Types of Synchronisation Tools

- **Memory fences** (eg: `# pragma omp flush`, h/w memory fences like `mfence`)
- **Atomic Operations:** event should happen uninterrupted
  - **Test & set, Fetch & add, Compare & swap**
- **Critical sections, Lock, Mutexes:** Events should **NOT** happen together
- **Barriers:** Events should happen together
- **Wait, Condition variables:** event A should happen before event B

Peterson Mutual Exclusion Algorithm

- Thread i: announces its interest
- Defer to another
- Wait while other is interested and thread i is interested
- Unset the flag when no longer interested

```
public void lock() {
  flag[i] = true;
  victim  = i;
  while (flag[j] && victim == i) {};
 }
 public void unlock() {
  flag[i] = false;
 }
```

Peterson Mutual Exclusion Algorithm

Is it correct?

- $(flag[B] := true) \rightarrow (victim := B)$
- $(victim := A) \rightarrow Rd_A(flag[B]) \rightarrow Rd_A(victim)$
- WLOG:
  $(victim := B) \rightarrow (victim := A)$
- Combining the observations, we observe
  $(flag[B] = True) \wedge (victim = A)$
  - $CS_B \rightarrow CS_A$

```
public void lock() {
  flag[i] = true;
  victim  = i;
  while (flag[j] && victim == i) {};
}
public void unlock() {
  flag[i] = false;
}
```

Peterson Mutual Exclusion Algorithm

Is it deadlock-free?

- **Deadlock-freedom:** The system as a whole progresses
- At any point one or the other is **NOT** the victim
    - At least one thread is progressing at all times

```
public void lock() {
  flag[i] = true;
  victim  = i;
  while (flag[j] && victim == i) {};
 }
 public void unlock() {
  flag[i] = false;
 }
```

Peterson Mutual Exclusion Algorithm

Is it starvation-free?

- **Starvation-freedom**: Each thread eventually makes progress
- Thread A is **starved** only if B repeatedly enters CS_B
- But can thread B repeatedly enter?
  - **WHY NOT?**

```
public void lock() {
  flag[i] = true;
  victim  = i;
  while (flag[j] && victim == i) {};
 }
 public void unlock() {
  flag[i] = false;
 }
```

Peterson Mutual Exclusion Algorithm

Does it have bounded-waiting?

- **Bounded-waiting**: Each thread makes progress in finite time
- If thread A **starts** before thread B then thread A enters CS before thread B
    - But what does start mean?

```
public void lock() {
  flag[i] = true;
  victim  = i;
  while (flag[j] && victim == i) {};
 }
 public void unlock() {
  flag[i] = false;
 }
```

Peterson Mutual Exclusion Algorithm

Does it have bounded-waiting?

- `lock()` is divided into two parts
  - **Doorway**: Always finish in finite steps
  - **Waiting:** May take unbounded steps
- For threads A and B:
  - If $D_A^k \to D_B^j$ (ie, A's kth doorway precede B's 9th doorway)
  - Then, $CS_A^k \to CS_B^{(}j + r)$ (ie, B cannot overtake A by more than r times)

```
public void lock() {
  flag[i] = true;
  victim  = i;
  while (flag[j] && victim == i) {};
}
public void unlock() {
  flag[i] = false;
}
```

Peterson Mutual Exclusion Algorithm

Does it have bounded-waiting?

- What is the r value for Peterson's ME Alg?
- Can Peterson's ME Alg work for more than 2 threads?
- Can it work under weak memory models?

```
public void lock() {
  flag[i] = true;
  victim  = i;
  while (flag[j] && victim == i) {};
}
public void unlock() {
  flag[i] = false;
}
```

Bakery Algorithm

An n thread ME Solution

- Doorway:
  - $flag[i] := true$ — I'am interested
  - $label[i]\ldots$ — Take increasing labels
- Bounded Wait:
  - Someone is interested **AND** has earlier lex-ordered label
  - $D_A \rightarrow D_B$ then $A's$ label is smaller
  - B is locked out while $flag[A]$ is true!

```
class Bakery implements Lock {
  boolean[] flag;
  Label[] label;
 public Bakery (int n) {
   flag  = new boolean[n];
   label = new Label[n];
   for (int i = 0; i < n; i++) {
      flag[i] = false; label[i] = 0;
   }
 }
```

```
public void lock() {
 flag[i]  = true;
 label[i] = max(label[0], …,label[n-1])+1;
 while (for some k:
   flag[k] && (label[i],i) > (label[k],k));
}
 public void unlock() {
   flag[i] = false;
 }
```

Prove that Bakery Alg is ME-safe!