Mutual Exclusion I

```
Boolean S0, S1;
S0=false, S1=false;
```

```
//Thread T0
while (true) {
    while (S0 == S1);
    //Critical section
    S0 = S1;
}
```

```
//Thread T1
while (true) {
    while (S0 != S1);
    //Critical section
    S1 = !S0;
}
```

- Does it achieve one of more of the correctness properties of a concurrent program:
  - Mutual exclusion: Only one thread in critical section at a time
  - Progress (deadlock-free): If several simultaneous requests, must allow one to proceed
  - Bounded waiting (starvation-free): Must eventually allow each waiting thread to enter
- Does it need the TestAndSet() instruction for atomic execution like the previous slide “Locks: Loads/Stores”?
- What is its major flaw?
- ANS:

Mutual Exclusion II

```
Boolean flag[2];
flag[0]=false, flag[1]=false;
```

```
//Thread T0
while (true) {
    flag[0] = true;
    while (flag[1]==true);
    /* Critical Section */
    flag[0] = false;
}
```

```
//Thread T1
while (true) {
    flag[1] = true;
    while (flag[0]==true);
    /* Critical Section */
    flag[1] = false;
}
```

- Does it achieve one of more of the correctness properties of a concurrent program:
  - Mutual exclusion: Only one thread in critical section at a time
  - Progress (deadlock-free): If several simultaneous requests, must allow one to proceed
  - Bounded waiting (starvation-free): Must eventually allow each waiting thread to enter
- ANS:

Mutual Exclusion III (Peterson's Solution)

```
Boolean flag[2];
flag[0]=false, flag[1]=false;
int turn = 0;
```

```
//Thread T0
while (true) {
    flag[0] = true;
    turn = 1;
    while (flag[1]==true && turn==1);
    /* Critical Section */
    flag[0] = false;
}
```

```
//Thread T1
while (true) {
    flag[1] = true;
    turn = 0;
    while (flag[0]==true && turn==0);
    /* Critical Section */
    flag[1] = false;
}
```

- Does it achieve one of more of the correctness properties of a concurrent program:
  - Mutual exclusion: Only one thread in critical section at a time
  - Progress (deadlock-free): If several simultaneous requests, must allow one to proceed
  - Bounded waiting (starvation-free): Must eventually allow each waiting thread to enter
- ANS:

Mutual Exclusion III (Peterson's Solution Variation)

```
Boolean flag[2];
flag[0]=false, flag[1]=false;
int turn = 0;
```

```
//Thread T0
while (true) {
    flag[0] = true;
    turn = 0;
    while (flag[1]==true && turn==1);
    /* Critical Section */
    flag[0] = false;
}
```

```
//Thread T1
while (true) {
    flag[1] = true;
    turn = 1;
    while (flag[0]==true && turn==0);
    /* Critical Section */
    flag[1] = false;
}
```

- Does it achieve one of more of the correctness properties of a concurrent program:
  - Mutual exclusion: Only one thread in critical section at a time
  - Progress (deadlock-free): If several simultaneous requests, must allow one to proceed
  - Bounded waiting (starvation-free): Must eventually allow each waiting thread to enter
- ANS: