Conditions for a Solution

- The execution of related critical sections must be mutually exclusive, that is, no two processes may access related critical sections simultaneously.
- One process, A, outside of its critical section, cannot prevent another process, B, from entering any of B's critical sections.
- When two processes want to enter their critical sections at the same time, the decision as to which one gets to go first cannot be postponed indefinitely.

Software Non-Solution #1

P_i and P_j share data, are in infinite loops, and are identical except for `i`'s and `j`'s

P_i:

```
while(1) {
    while(turn != i);
    Critical Section
    turn = j;
    Non-Critical Section
}
```

P_j:

```
while(1) {
    while(turn != j);
    Critical Section
    turn = i;
    Non-Critical Section
}
```

Violates condition 2

Software Non-Solution #2

P_i:

```
while(1) {
    while(flag[j]);
    flag[i] = true;
    Critical Section
    flag[i] = false;
    Non-critical Section
}
```

P_j:

```
while(1) {
    while(flag[i]);
    flag[j] = true;
    Critical Section
    flag[j] = false;
    Non-critical Section
}
```

Violates condition 1

Software Non-Solution #3

P_i:

```
while(1) {
    flag[i] = true;
    while(flag[j]);
    Critical Section
    flag[i] = false;
    Non-critical Section
}
```

P_j:

```
while(1) {
    flag[j] = true;
    while(flag[i]);
    Critical Section
    flag[j] = false;
    Non-critical Section
}
```

Violates condition 3

A Software Solution

```
while(1) {
    flag[i] = true;
    while(flag[j]) {
        if (turn == j) {
            flag[i] = false;
            while (turn == j);
            flag[i] = true;
            }
        }
    Critical Section
    turn = j;
    flag[i] = false;
    Non-critical Section
    }
```

```
while(1) {
    flag[j] = true;
    while(flag[i]) {
        if (turn == i) {
            flag[j] = false;
            while (turn == i);
            flag[j] = true;
            }
        }
    Critical Section
    turn = i;
    flag[j] = false;
    Non-critical Section
    }
```

A Hardware Solution

- Most modern processors provide an instruction called "Test & Set" or "Exchange."
  - Test & Set: tests an item in storage and sets it to a given value
  - atomic operation: no chance for interruption (either all of it happens or none of it does
  - Exchange: swaps two items

A Hardware Solution

```
BOOLEAN test_and_set(BOOLEAN *location) {
    BOOLEAN x;
    x = *location;
    *location = TRUE;
    return(x);
}

while(1) {
    while (test_and_set(&lock)); /* busy wait */
    Critical Section
      lock = FALSE;
    Non-Critical Section
}
```

Test & Set works like this test_and_set software.

Solution Evaluation

- The software solution given for two processes is difficult to extend to n processes
- The hardware Test & Set solution will work for any number of processes
- Problems:
  - busy waits are a waste of resources (e.g CPU)
  - not easily generalized to more complex synchronization problems