The critical section problem - two or more processes try to access a shared variable concurrently

Critical section is that section of code in a program where multiple threads are trying to access some shared data concurrently and the outcome depends on the order in which access takes place. If these threads are not synchronized using some mechanism, this can lead to race condition.

Let's look at some examples to understand it.

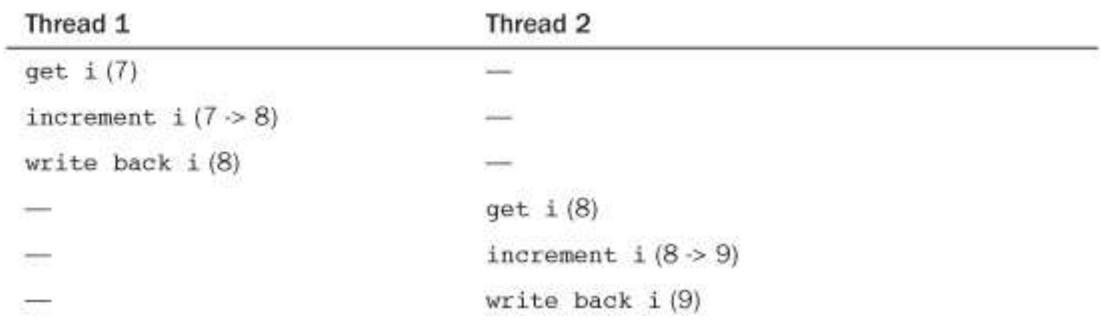
```
int i = 7; //global variable or shared resource
void increment() {
   i++; // critical section of code
}
int main() {
thread T1,T2;
T1(increment);
T2(increment);
}
```

i++ may translate into machine instructions to the computer's processor that resemble the following:

1. get the current value of i and copy it into a register
2. add one to the value stored in the register
3. write back to memory the new value of i

What will be the final value of i after both threads have completed execution?

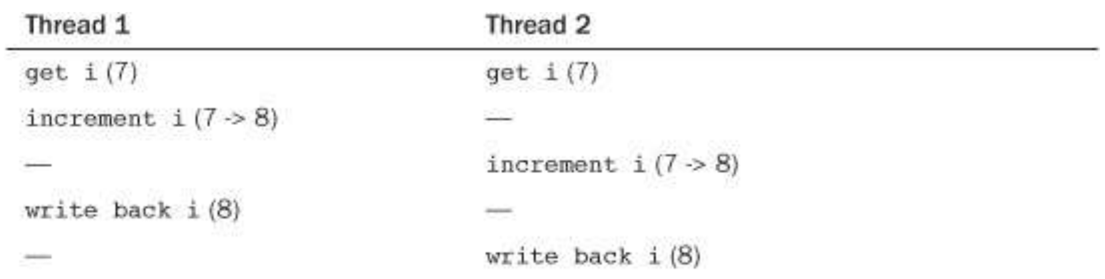
Case 1: The desired outcome (9) is similar to the following (with each row representing a unit of time):



Thread 1	Thread 2
get i (7)	—
increment i (7 -> 8)	—
write back i (8)	—
—	get i (8)
—	increment i (8 -> 9)
—	write back i (9)

As expected, the final value of i is 9. But there are other possible outcomes too.

Case 2: The output can also be 8 if the below event happens.



Thread 1	Thread 2
`get i (7)`	`get i (7)`
`increment i (7 -> 8)`	—
—	`increment i (7 -> 8)`
`write back i (8)`	—
—	`write back i (8)`

We can see from example that the final value of i depends on the order in which both threads access the variable. Different order of execution can result in different outputs.

This situation is called **race condition** since each thread is racing to access the shared data first.