Race Conditions

When 2 ore more threads use the same data and try to manipulate it at the same time. The used data is concurrently changed by the threads and they both race which changes will be committed. The changes by the thread which finishes last will remain

Example:

Thread A and B both want to use variable x at the same time to assign something to y

x = 2

{ y = x * x }

- y should be 4, but if a thread changes x before another one does the calculation, y could be anything
- that is why we use locks before the critical part

lock x

x = 2

{ y = x * x }

release lock for x