Concurrency

Threading problems

Threading problems

- Race condition.
- Deadlock.
- Livelock.
- Starvation.

Race Condition

- A *race condition* occurs when two or more threads gain access to a shared resource at the same time.

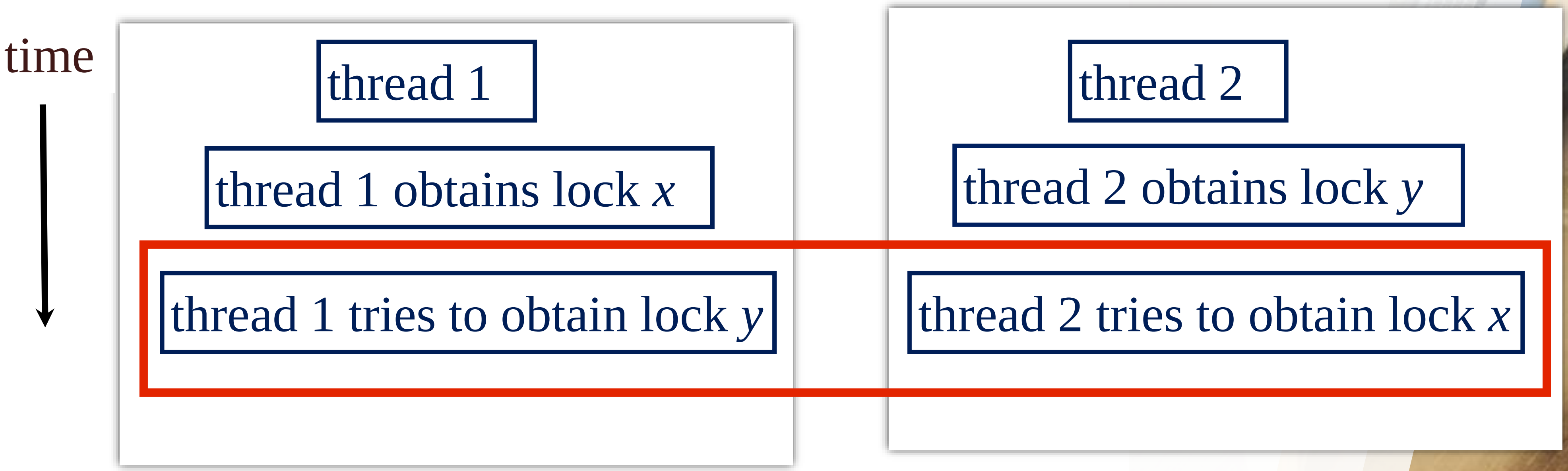
- This shared resource should be accessed sequentially.

- Creation of a car registration number…
  - two cars with the same registration number
  - both cars refused
  - one with a registration number and one without

- RaceCondition.java

Deadlock

- A *deadlock* occurs when locking threads are waiting on each other to free locks that they themselves hold.



time

thread 1

thread 1 obtains lock *x*

thread 1 tries to obtain lock *y*

thread 2

thread 2 obtains lock *y*

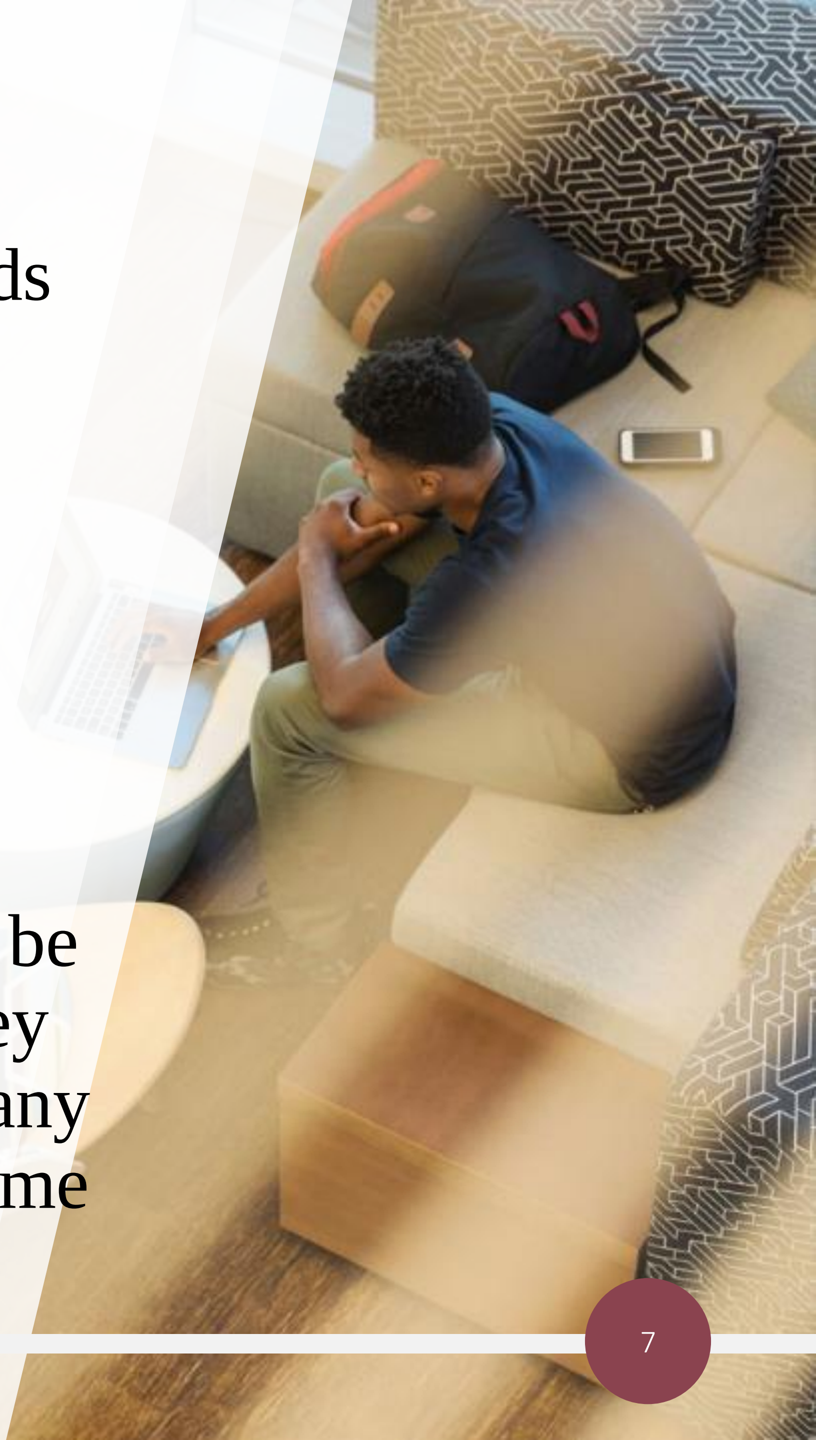
thread 2 tries to obtain lock *x*

- Deadlock.java

Livelock

- A *livelock* is similar to a deadlock in that the threads involved are stuck, making no progress. However, with deadlock, the threads are doing nothing. With livelock, the threads are busy but their actions are repeatedly triggering the same conditions.

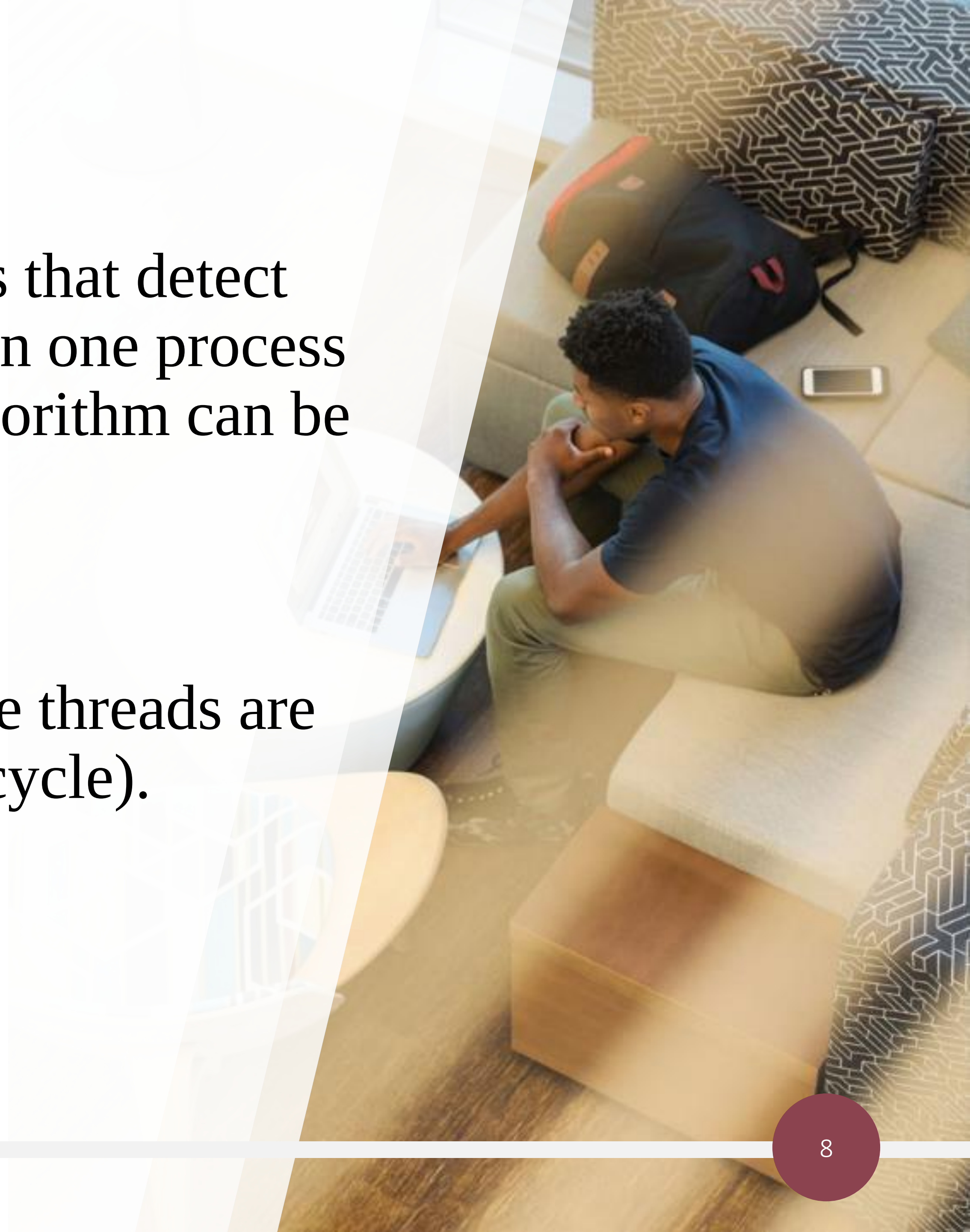
- A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time.



Livelock

- Livelock is a risk with some algorithms that detect and recover from deadlock. If more than one process takes action, the deadlock detection algorithm can be repeatedly triggered.

- Livelock can be difficult to detect as the threads are active (they are stuck in an an endless cycle).



Starvation

- *Starvation* occurs when a thread is unable to gain access to a required resource.

- This can happen to low-priority threads if the resource in question is in high demand by higher-priority threads.

- This can affect the liveness of you application as, even if it is a low-priority thread, it must get it’s work done.