Multithreading

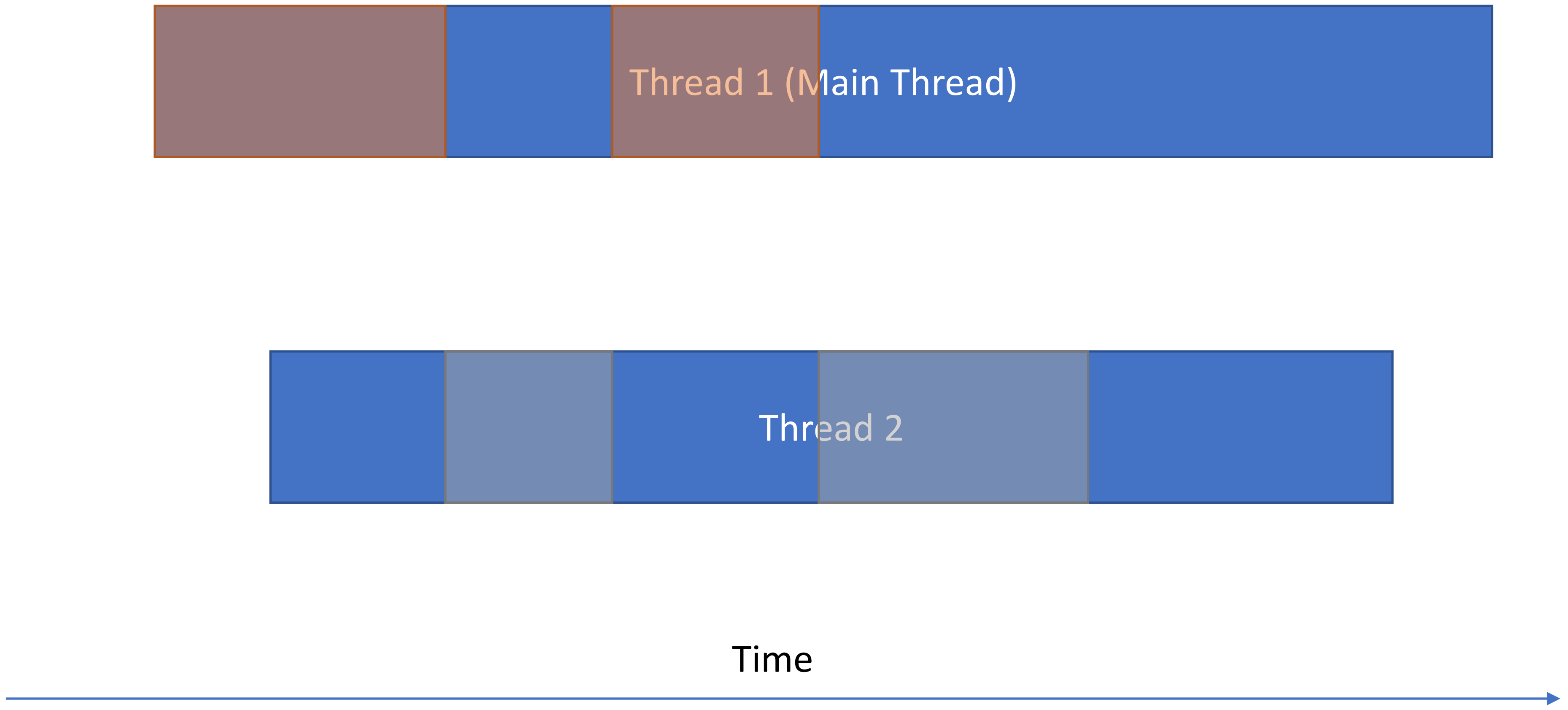
Thread

- Definition: A single sequential flow of execution
  - If we have a multi-threaded application, we essentially have multiple sequential flows of execution
  - This means we can run multiple instructions “simultaneously”
  - In a multi-threaded application, we have multiple “readers”
  - Every single thread still just executes one instruction (one line) after another.

Concurrency v. Parallelism

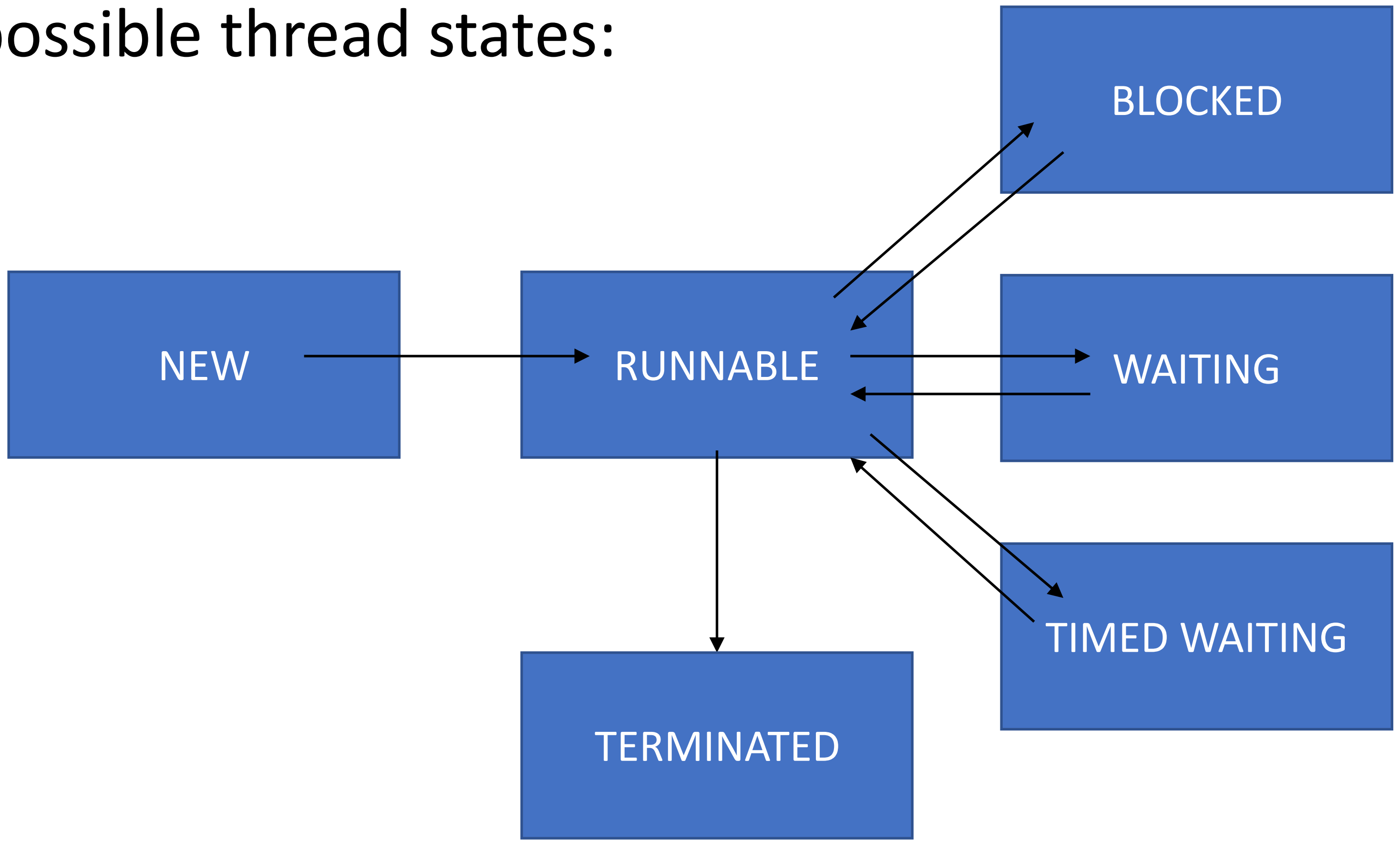
- Concurrency: Two or more tasks can be done in overlapping time periods
    - Without parallelism: similar to a person who is multitasking. They’re not actually doing two things at the same time, they’re just switching between different tasks
    - Thread Scheduler: This is a component of the operating system that gives processing time to each thread, giving the illusion
- Parallelism: Things are really happening at the same time
    - This can happen if our CPU has multiple cores
    - So, if we have, for example, 4 cores, we can actually process 4 threads at the same time.
    - IF we have 4 cores and 60 threads, we can still only process 4 threads at the same time, but our thread scheduler will try its best to give each thread an even amount of processing time
        - This gives the illusion that everything is happening at the same time

2 threads, 1 core

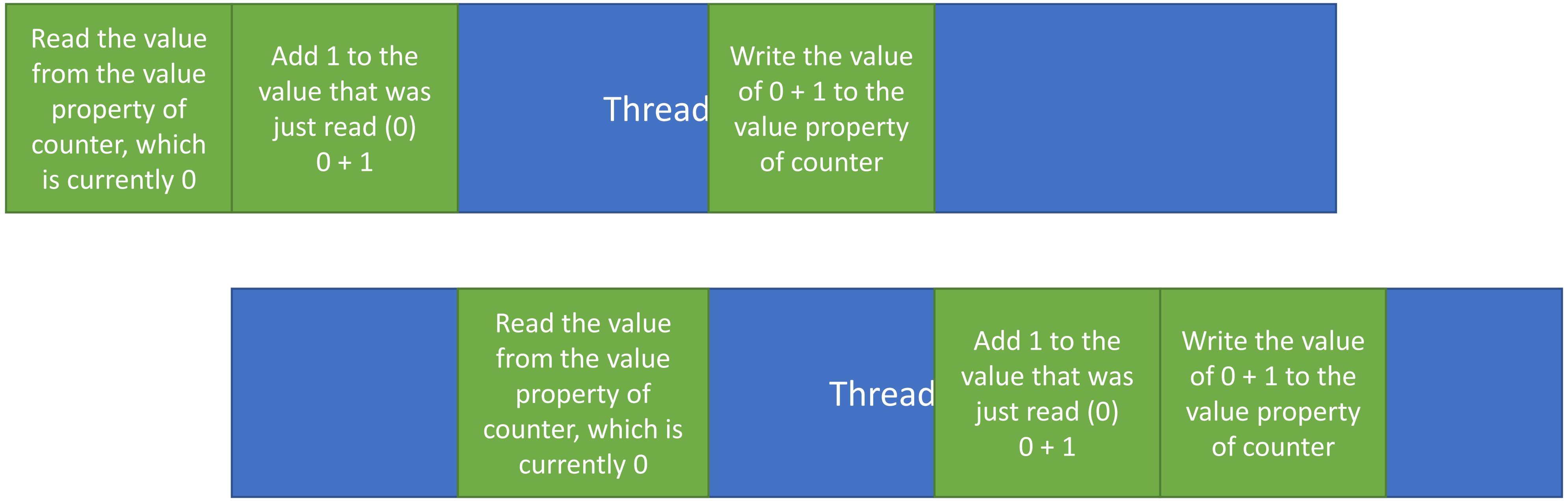


Thread States

- This is associated with the Thread class and Thread objects in Java
- We have 6 different possible thread states:
  - NEW
  - RUNNABLE
  - BLOCKED
  - WAITING
  - TIMED WAITING
  - TERMINATED



Race Conditions


Read the value from the value property of counter, which is currently 0
Add 1 to the value that was just read (0) 0 + 1
Thread
Write the value of 0 + 1 to the value property of counter
Read the value from the value property of counter, which is currently 0
Thread
Add 1 to the value that was just read (0) 0 + 1
Write the value of 0 + 1 to the value property of counter

Deadlock

- Deadlock: This situation arises if we have two threads in which 1 thread is depending on another lock in order to release the lock it currently has, but the other thread is also depending on the lock of the first thread in order to release its lock
  - Therefore, both threads are stuck and can't continue
  - “If we have two people who want to switch rooms, and we have one guy in one room waiting for the other guy in another room to leave before he leaves, and vice versa, neither of them leave and are stuck in their current room”

Producer-Consumer Problem

- Producer: a thread that produces values (or grabs data from somewhere and populates it into some sort of data structure)
- Consumer: a thread that takes values out of some data structure to do something with
- The problem:
  - If there is no data available in the data structure, the consumer should go to sleep (enter the WAITING state)
  - If the data structure is full, the producer should go to sleep (enter the WAITING state)
- Once a thread enters WAITING, it needs to be informed to “wake up” later.