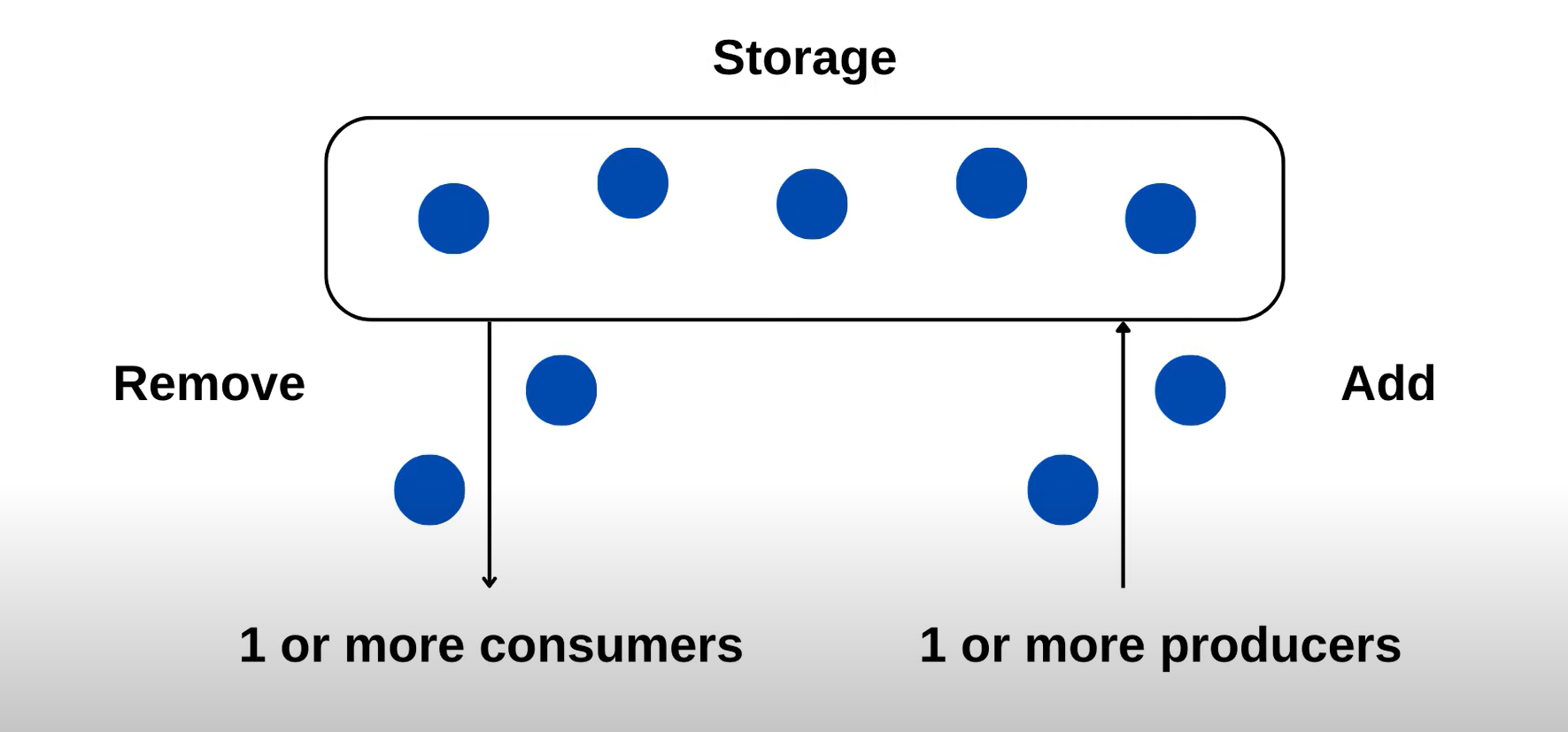
Produces Consumer Pattern Problems: -



Example: 1. Let us consider there is queue, and there is different type of threads

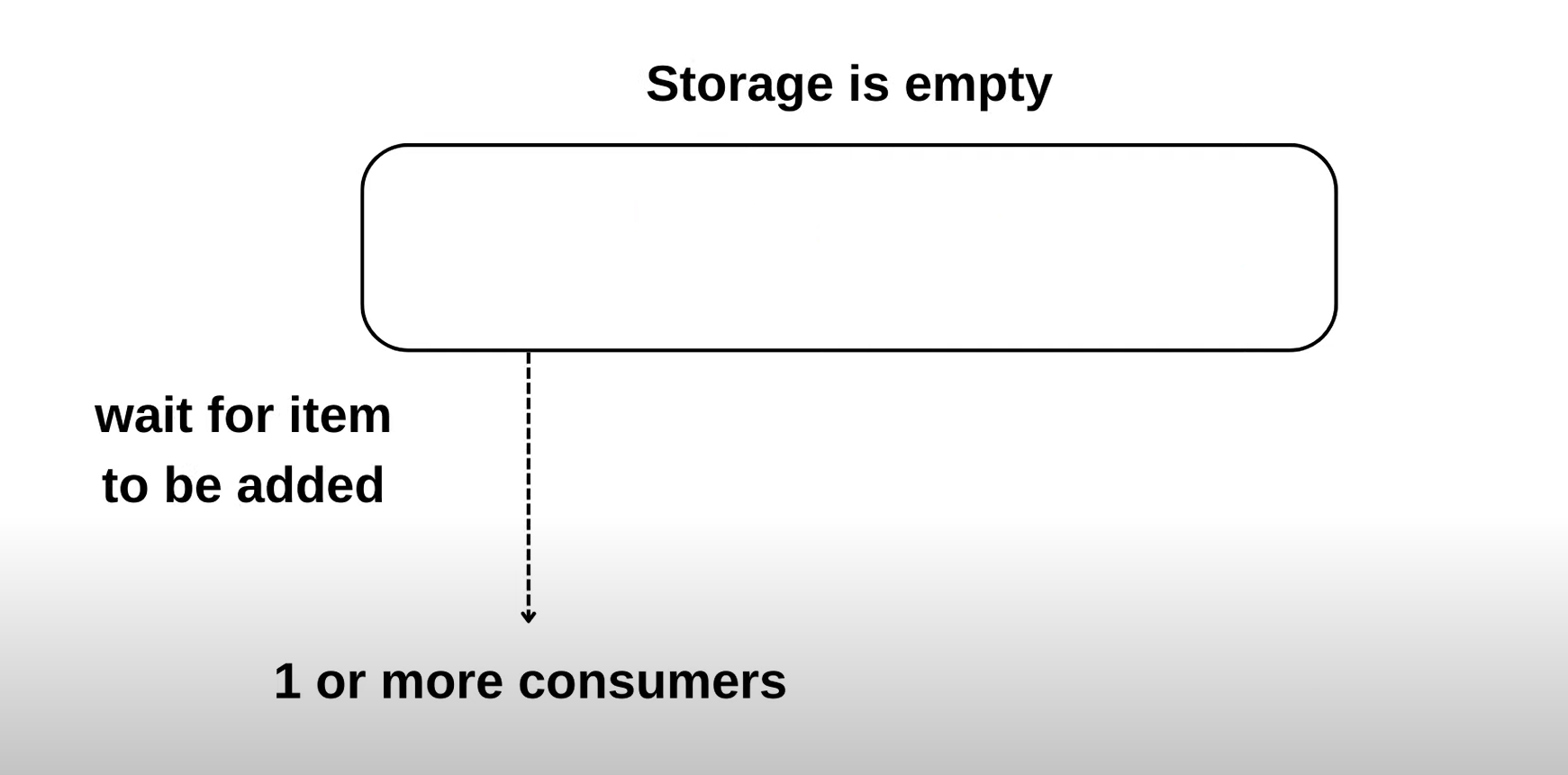
2.let me divide the threads into two groups

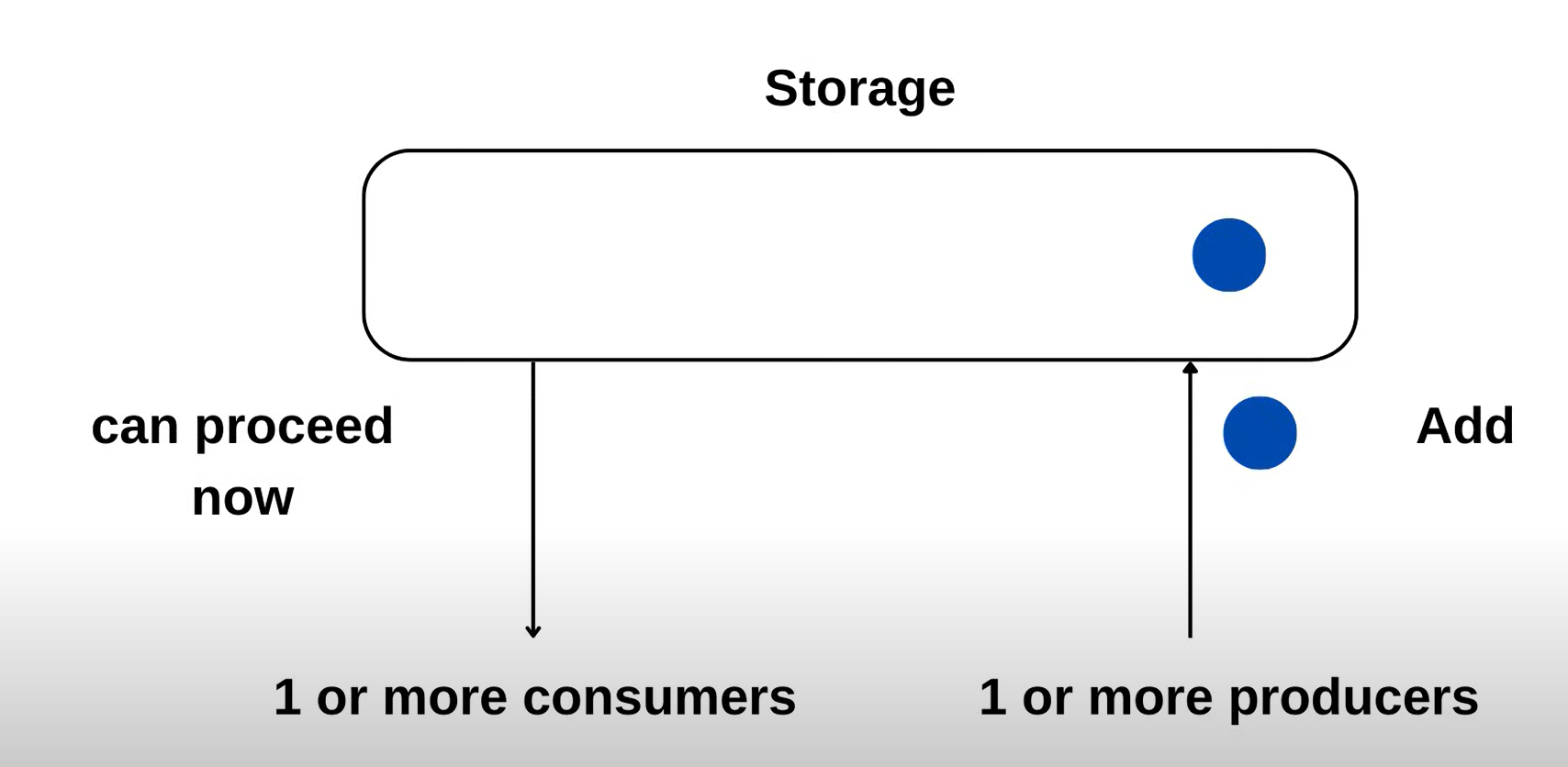
i.one thread is trying to push items into queue.

ii. another group of threads are trying to remove items from the queue.

3. if we can only push items to the queue, if it doesn’t increase the capacity.

4.we can only pull-out items from the queue, if there is item that is left in the queue.





Problem is when two adding threads threads and one remove thread

1.If adder 1 Thread get access.it will wait because the capacity is full

2.Then it removes thread start executing, and removes the item from the stack

3.notifies all threads that elements are removed

4.Now 1st adder thread will execute add elements

5.now capacity is capacity full again, second adder thread will wait until an element get removed (this is the problem)

6.for this we use while loop that problem gets sloved

Now Execution:- 1.when adder one wins the lock

lets say both threads are awakened they’re removed from the wait set and both trying to acquire the lock again

Remember when the particular thread calls wait() method,it relinquishes the lock,it doesn’t have the lock with it anymore

And when next time it is awakened it,it again fight for the lock as,it was other threads were doing

When it gets the lock, noe it resumes its execution from while

After while line if there is a while loop,it again go and check back this condition and first thread checks only one room left so,it can go and add item and notify it and return true

Now adder to gets the lock to execute,again check the capacity of the queue becaue of while, check the condition no capacity is full,again it goes to the wait set

Here we kind of maintain consistency or we don’t allow more items to added then the capacity of the queue

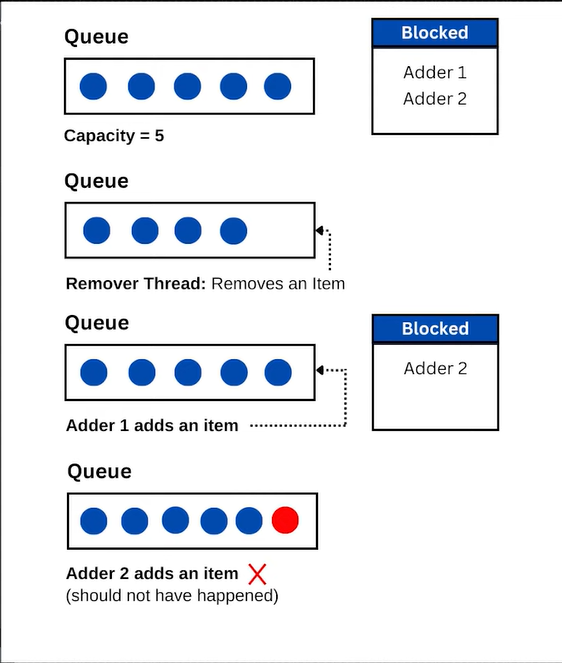
Note we have to use while only not if, if we use if condition we have no write if condition for total number of threads

The problem arises when there are two adding threads and one removing thread.

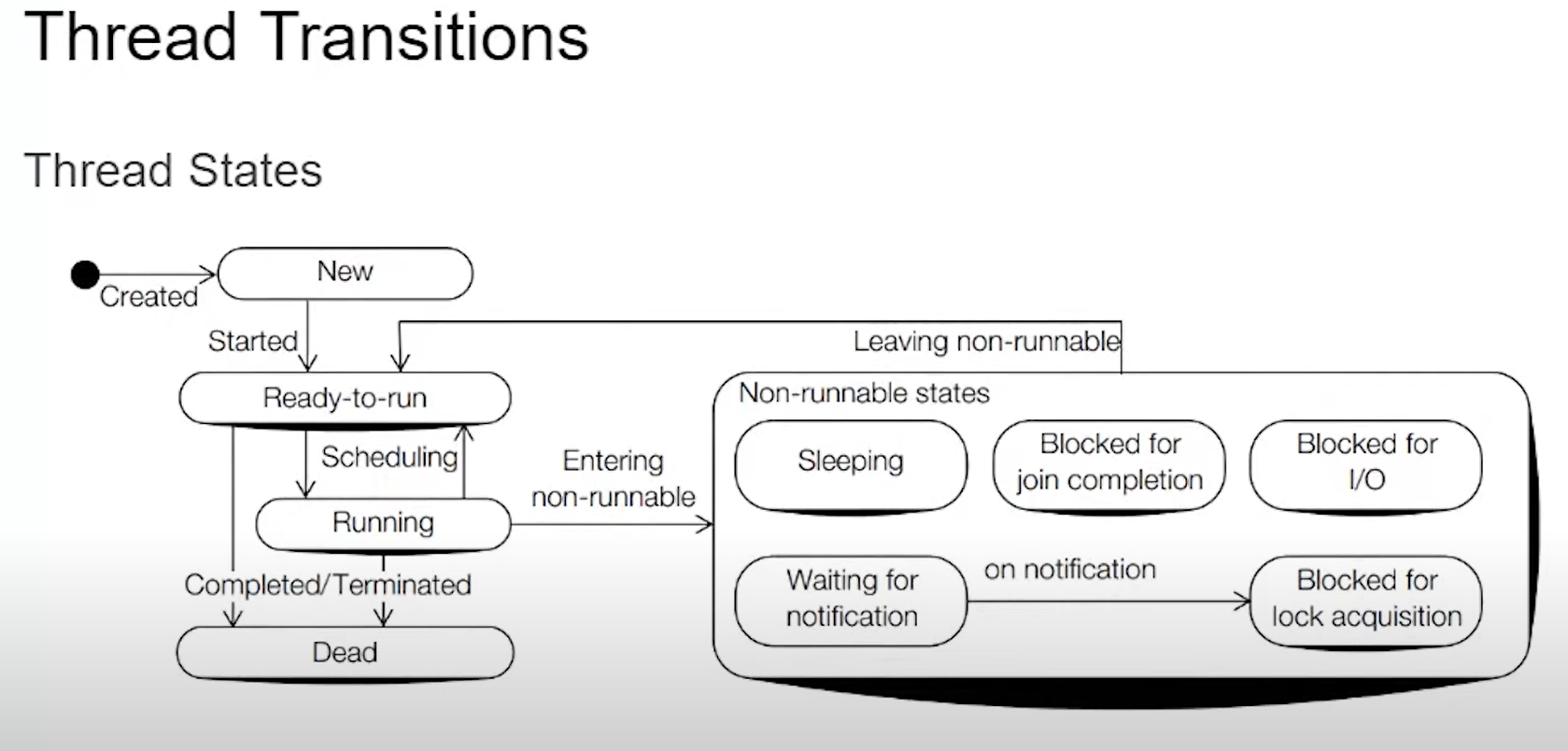
1. If Adder Thread 1 gets access, it will wait because the capacity is full.
2. Then the removing thread starts executing and removes an item from the stack.
3. It notifies all threads that elements have been removed.
4. Now, the first Adder thread can execute and add elements.
5. However, since the capacity is full again, the second Adder thread will wait until an element is removed. This is the problem.
6. To solve this problem, we use a while loop.

Now, during execution:

1. When Adder Thread 1 wins the lock, let's assume both threads are awakened. They are removed from the wait set and both try to acquire the lock again.
2. Remember, when a particular thread calls the wait() method, it relinquishes the lock. It no longer holds the lock.
3. When it is awakened again, it fights for the lock just like other threads were doing.
4. Once it acquires the lock, it resumes its execution from the while loop.
5. If there is another while loop after the first one, it checks the condition again. The first thread checks if only one room is left, so it can add an item, notify, and return true.
6. Now, Adder Thread 2 gets the lock to execute and checks the capacity of the queue again due to the while loop. If the capacity is full, it goes back to the wait set.
7. Here, we maintain consistency by not allowing more items to be added than the capacity of the queue.
8. Note that we must use a while loop instead of an if condition. If we use an if condition, we won't have the correct condition for the total number of threads.



Thread states:-



1.when a thread is created it will be in the new state,then it gets into “Readu to run” state

2.once the thread get scheduled,it will gets into run state,after the “run()” method get fully executed,it will go to dead state. Once the thread is dead we cant start that thread again

3.Now what we can do is. Let it is in running state what are the things can happen

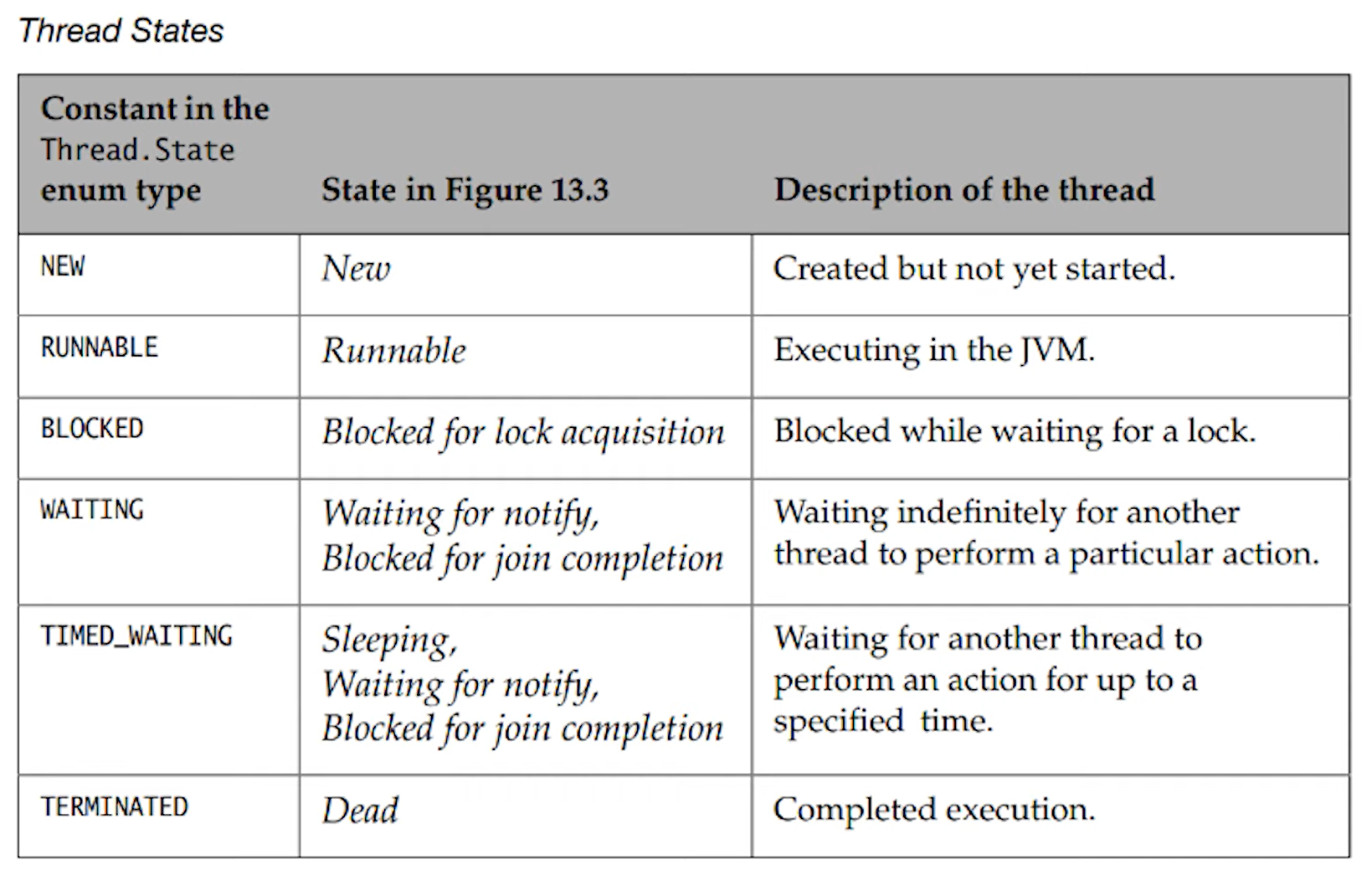
1.if we call the thread.sleep()-> it will enter into “Non-runnable” state in that it will go sleeping state,there are multiple non-runnable state sleeping state is the one of amoung them, in sleep state it wont execute for specifited time

2.Blocked for I/O:- it also non-runnable state,it will block the execution utill it gets some inputs from the user or imp:- waiting to for the lock it will wait until get gets the lock after getting the lock it leave the non runnable state

3.here one question is why,while sleeping the thread does not relinquish any lock,it will still hold the lock, it wont give the lock to other threads

1. When a thread is created, it starts in the new state and then transitions to the "Ready to run" state.
2. Once the thread is scheduled, it enters the run state. After the execution of the "run()" method is complete, it moves to the terminated state. Once a thread is terminated, it cannot be restarted.
3. Now, let's consider the possibilities when a thread is in the running state:
   * If we call the thread.sleep() method, it enters a "Non-runnable" state, specifically the sleeping state. In this state, it does not execute for a specified time.
   * Blocked for I/O is another example of a non-runnable state. It blocks the execution until it receives input from the user or, for example, waits for a lock. Once it acquires the lock, it leaves the non-runnable state.

Regarding the question of why a sleeping thread does not relinquish any lock and continues to hold it without giving it to other threads:



Running and yielding:

