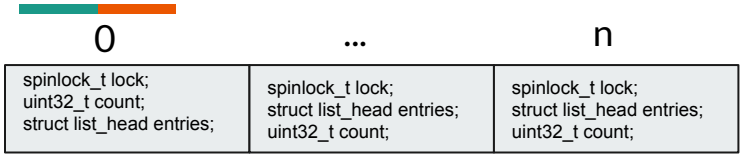
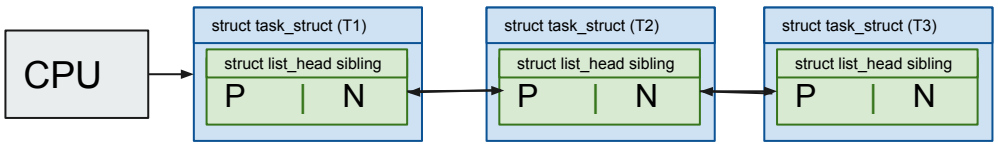
Wait Queues

Fridge with Blocking Get: How does it work?

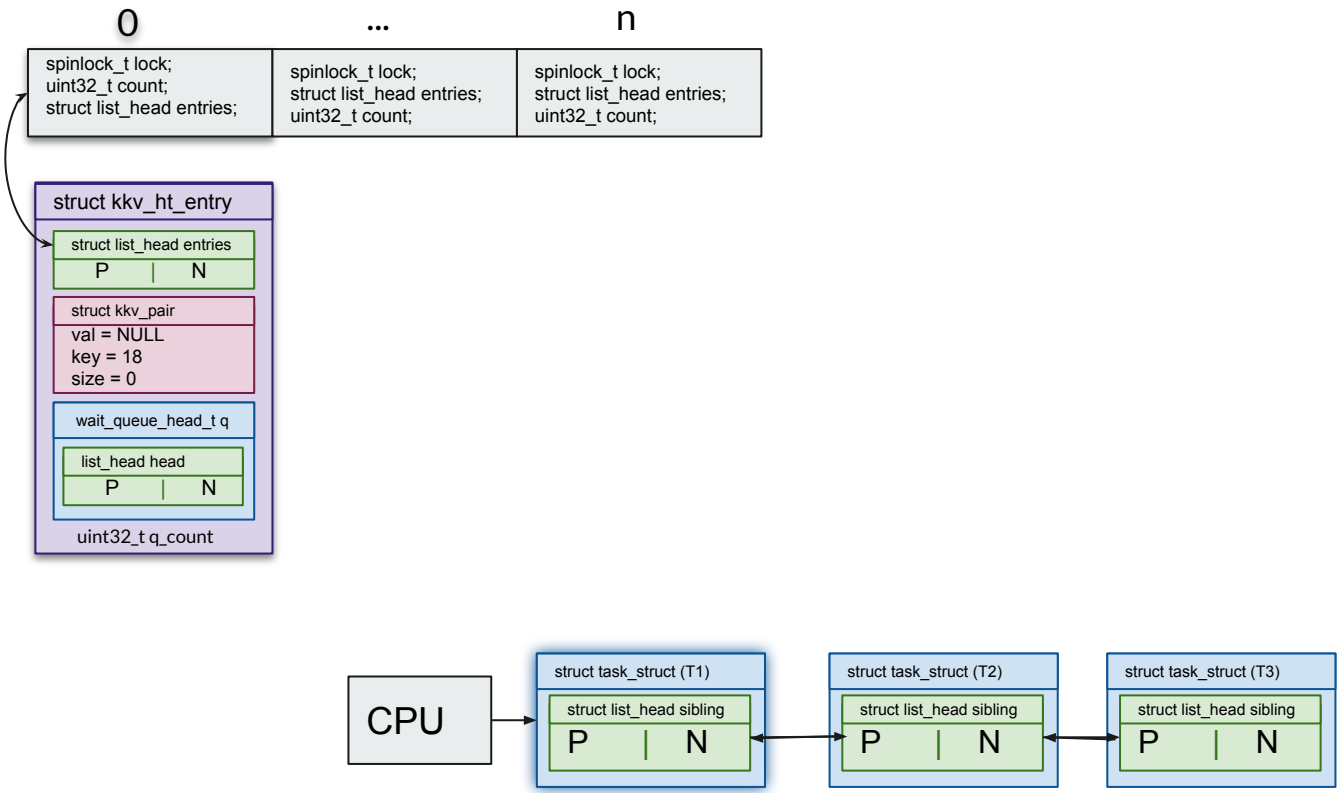
Written by Dave Dirnfeld for COMS 4118 at Columbia University

Let’s imagine a simple system with one CPU with 3 processes on the run queue, and an initialized KKV Hash Table that has no key/value pair in it. In this example n=17

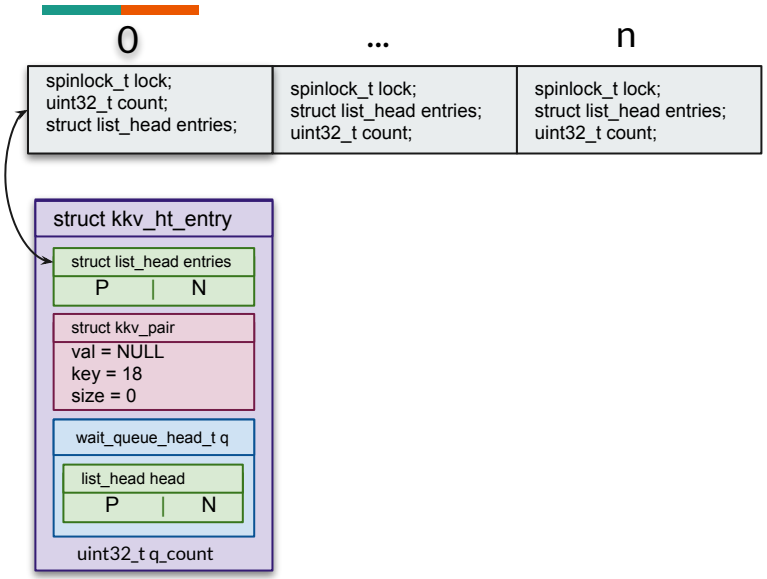




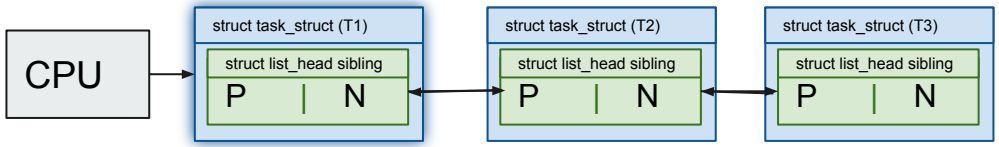
Now, let's say T1 calls kvv_get(18). Since key = 18 is not there (i.e. bucket 0 has no entry for key 18), T1 will create an entry for the key it is waiting for, and will go through the process of putting itself on the wait_queue until another process adds a value for key 18.



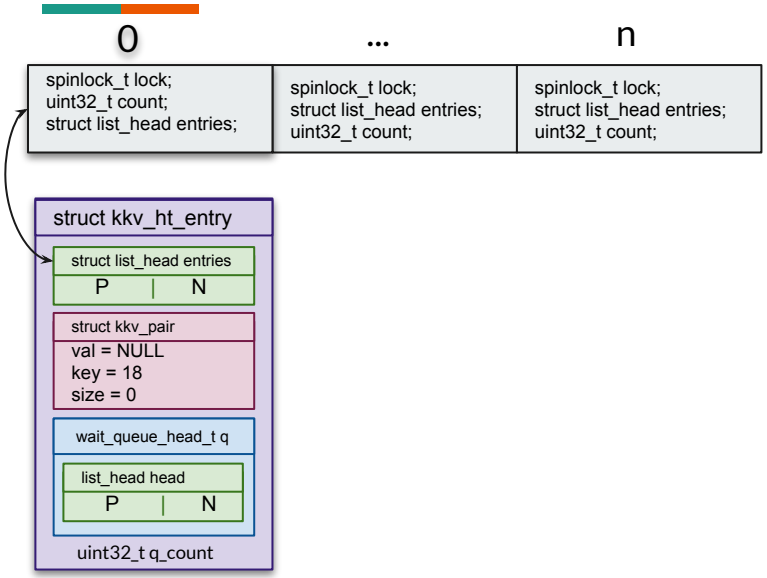
The Process of putting yourself to sleep



- T1 will call the macro DEFINE_WAIT()
- T1 will call prepare_to_wait()
- T1 will call schedule()
- Finally T1 will call finish_wait()



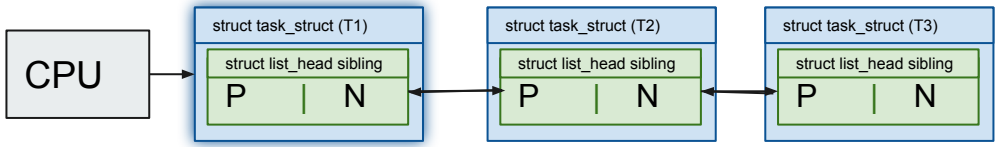
The Process of putting yourself to sleep



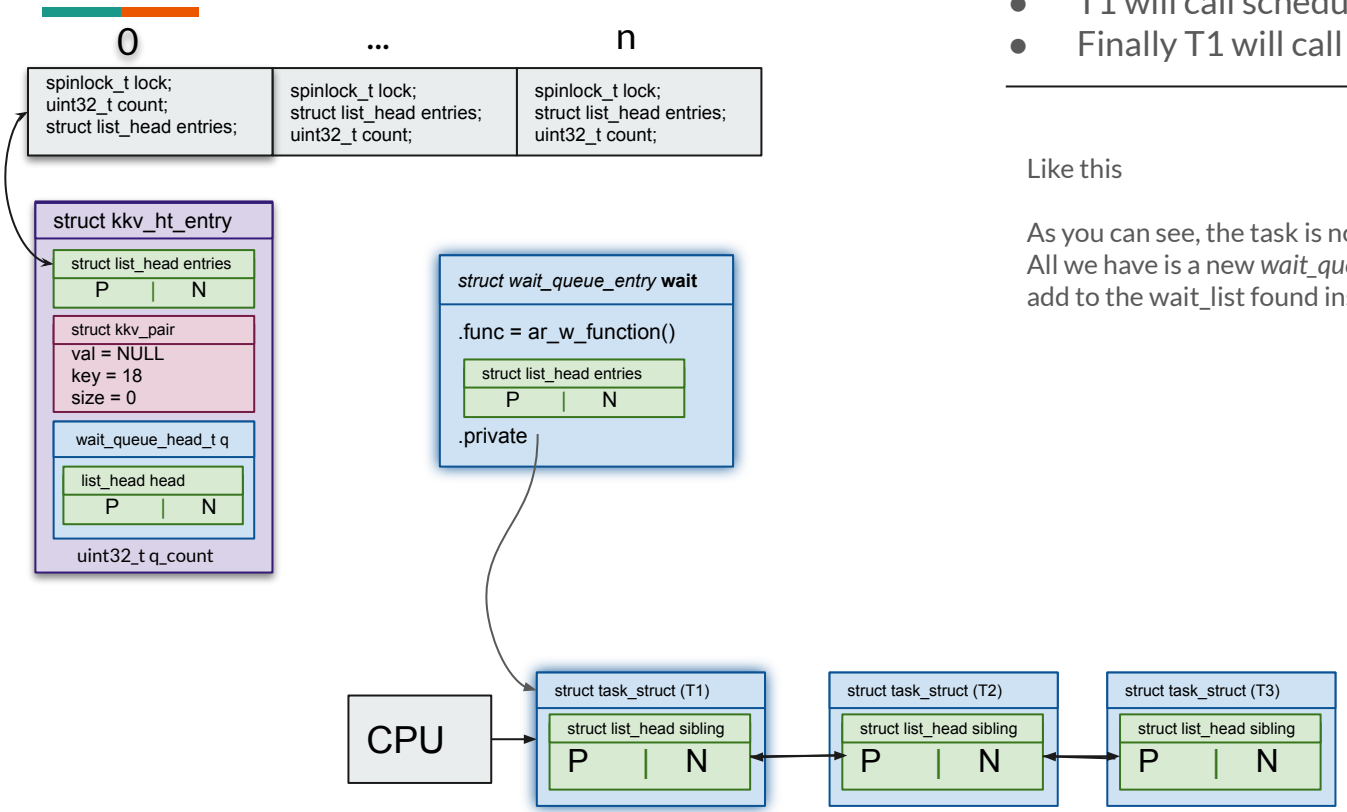
- **T1 will call the macro DEFINE_WAIT()**
- T1 will call prepare_to_wait()
- T1 will call schedule()
- Finally T1 will call finish_wait()

When a process calls DEFINE_WAIT(wait), the OS will create a new *struct wait_queue_entry* for that process. The OS fills in the required fields as follows:

- .private = current (this points to the current process)
- .func = autoremove_wake_function (ignore for now)
- .entry = LIST_HEAD_INIT((name).entry) (next and previous pointers point to itself)



The Process of putting yourself to sleep

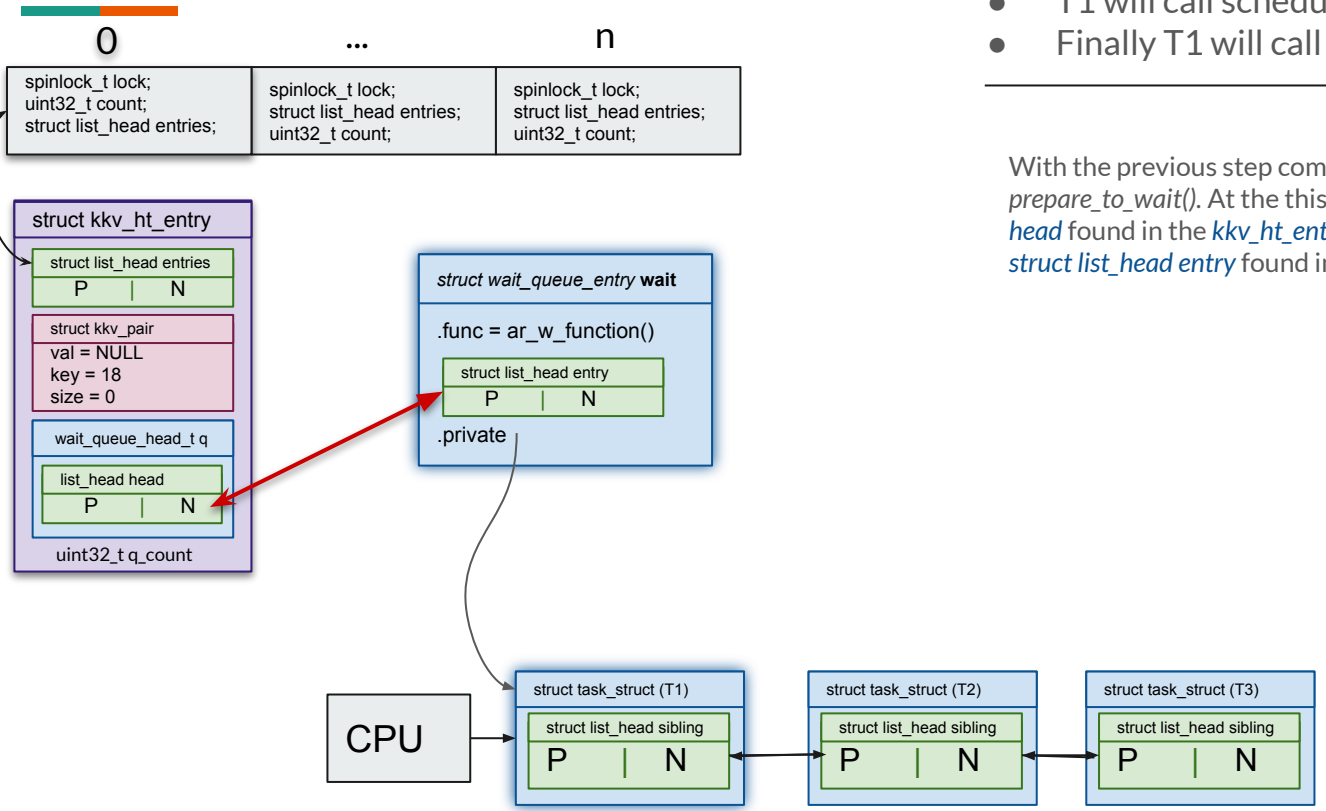


- **T1 will call the macro DEFINE_WAIT()**
- T1 will call prepare_to_wait()
- T1 will call schedule()
- Finally T1 will call finish_wait()

Like this

As you can see, the task is not on any wait_queue at this stage. All we have is a new *wait_queue_entry* that we will use later to add to the wait_list found inside the *kkv_ht_entry*.

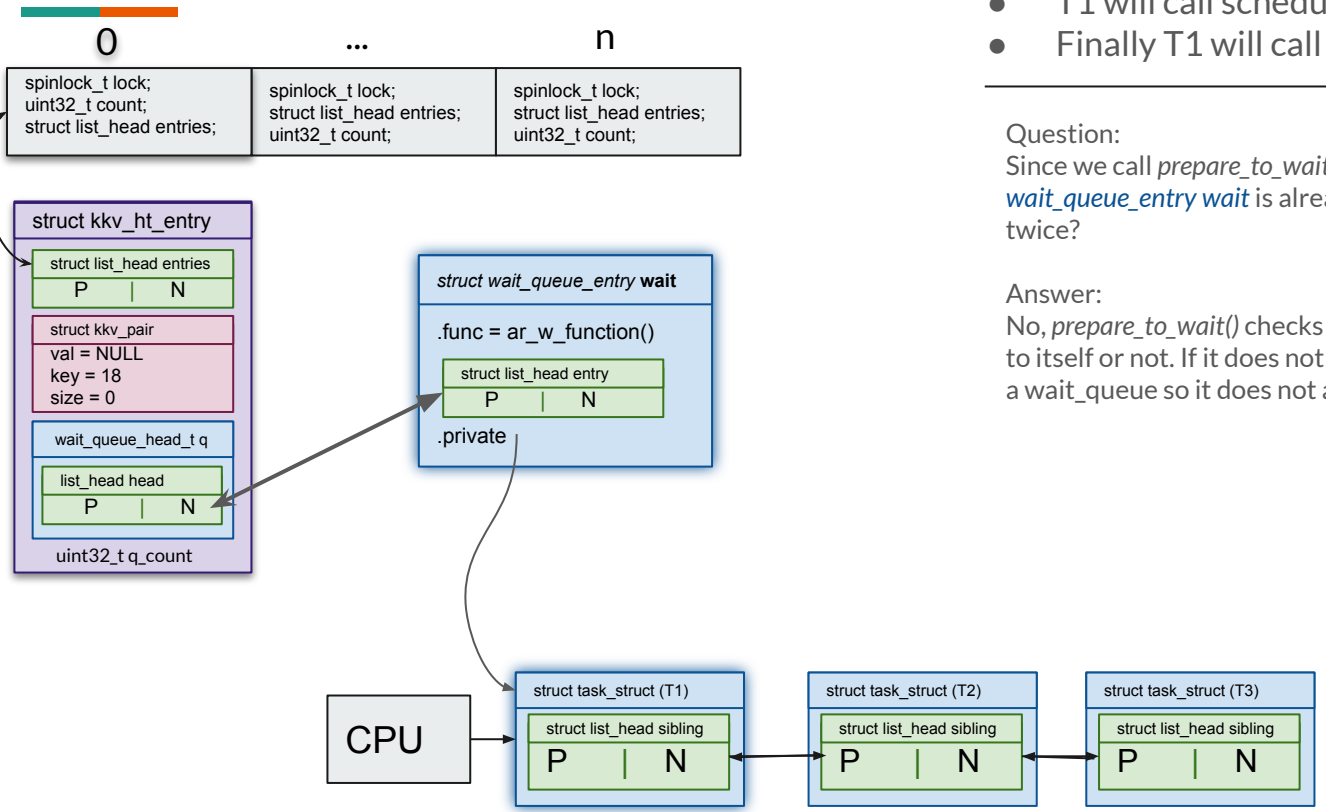
The Process of putting yourself to sleep



- T1 will call the macro DEFINE_WAIT()
- **T1 will call prepare_to_wait()**
- T1 will call schedule()
- Finally T1 will call finish_wait()

With the previous step completed, we can now call *prepare_to_wait()*. At the this stage the OS will link the *list_head head* found in the *kkv_ht_entry->wait_queue_head_t q* with the *struct list_head entry* found in *wait_queue_entry wait*.

The Process of putting yourself to sleep

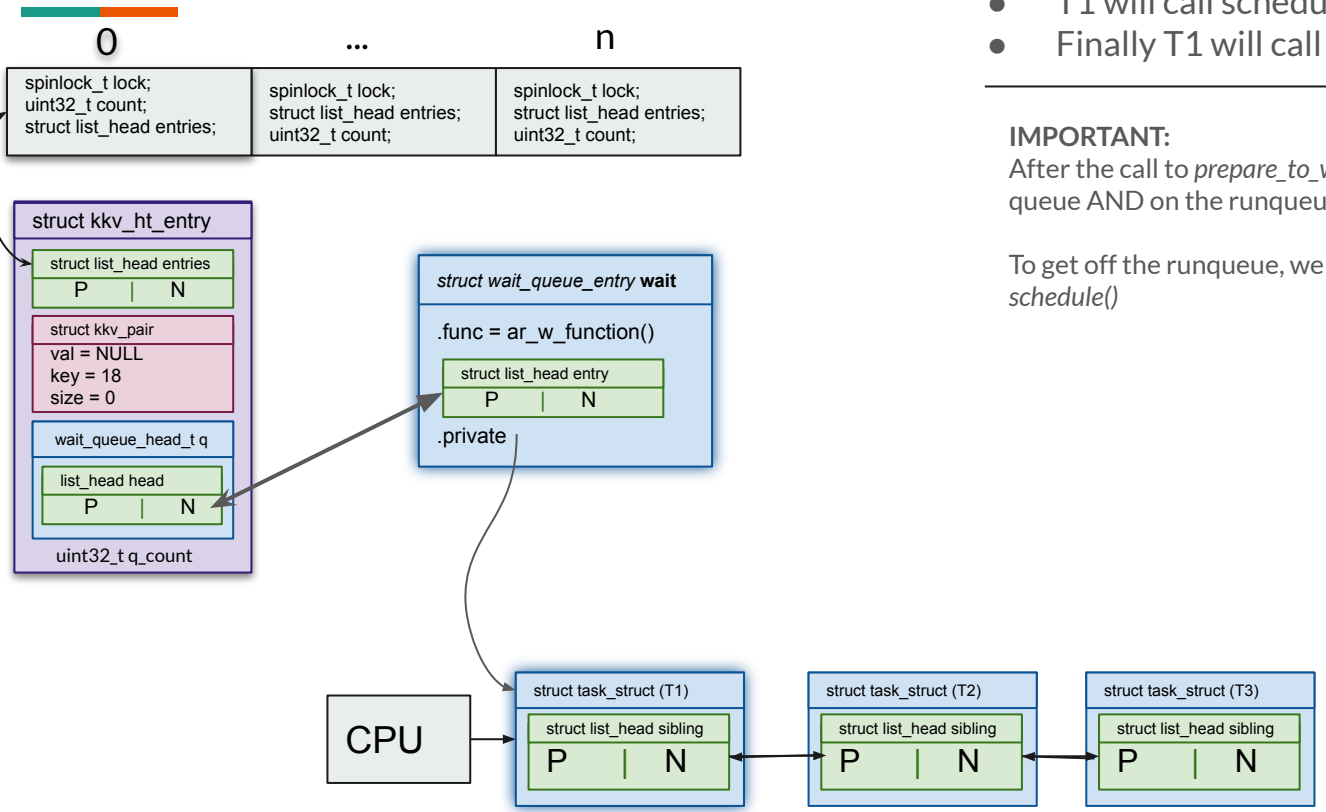


- T1 will call the macro DEFINE_WAIT()
- **T1 will call prepare_to_wait()**
- T1 will call schedule()
- Finally T1 will call finish_wait()

Question:
Since we call *prepare_to_wait()* in a loop, what happens if the *wait_queue_entry wait* is already on the list. Will it not be added twice?

Answer:
No, *prepare_to_wait()* checks if the *wait_queue_entry wait* points to itself or not. If it does not point to itself, than it is already on a wait_queue so it does not add it.

The Process of putting yourself to sleep

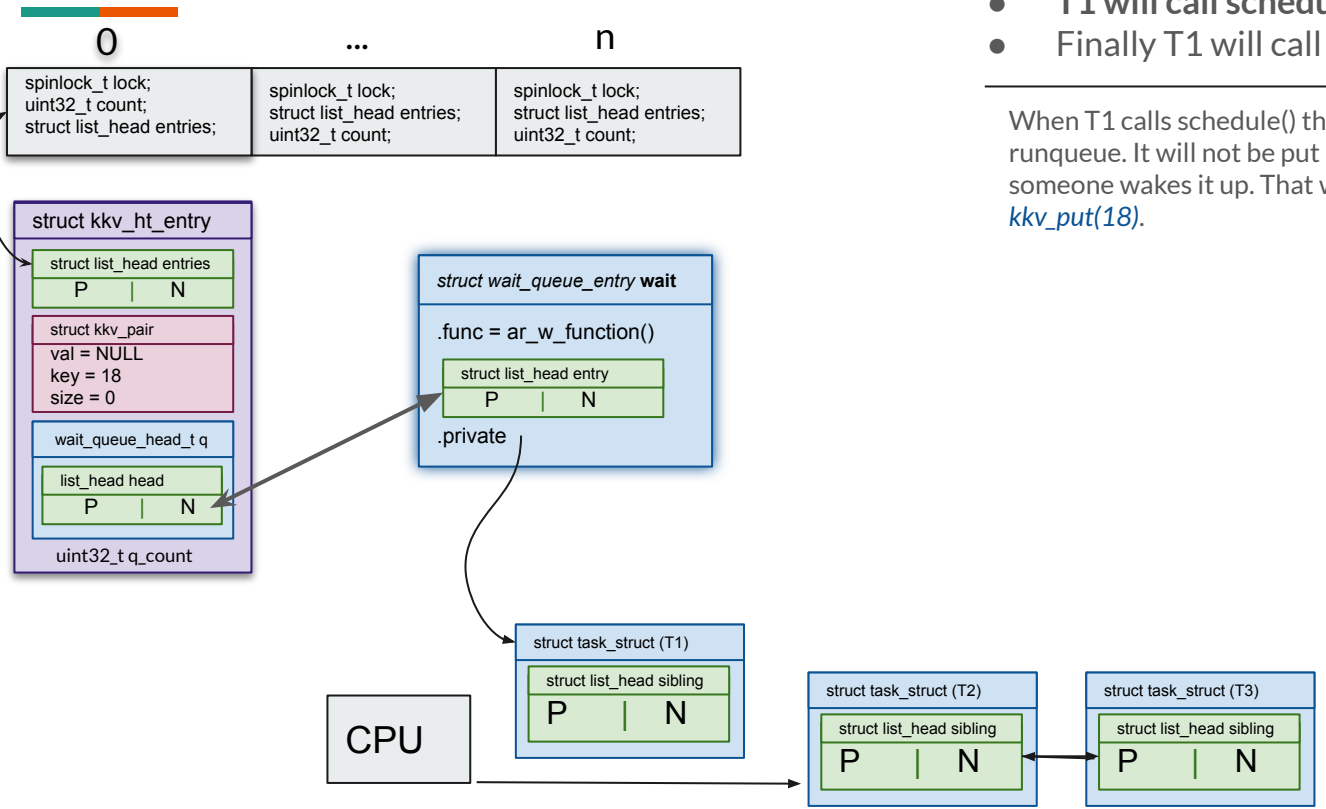


- T1 will call the macro DEFINE_WAIT()
- **T1 will call prepare_to_wait()**
- T1 will call schedule()
- Finally T1 will call finish_wait()

IMPORTANT:
After the call to *prepare_to_wait()* the task is both on a wait queue AND on the runqueue.

To get off the runqueue, we need to do the next step, call *schedule()*

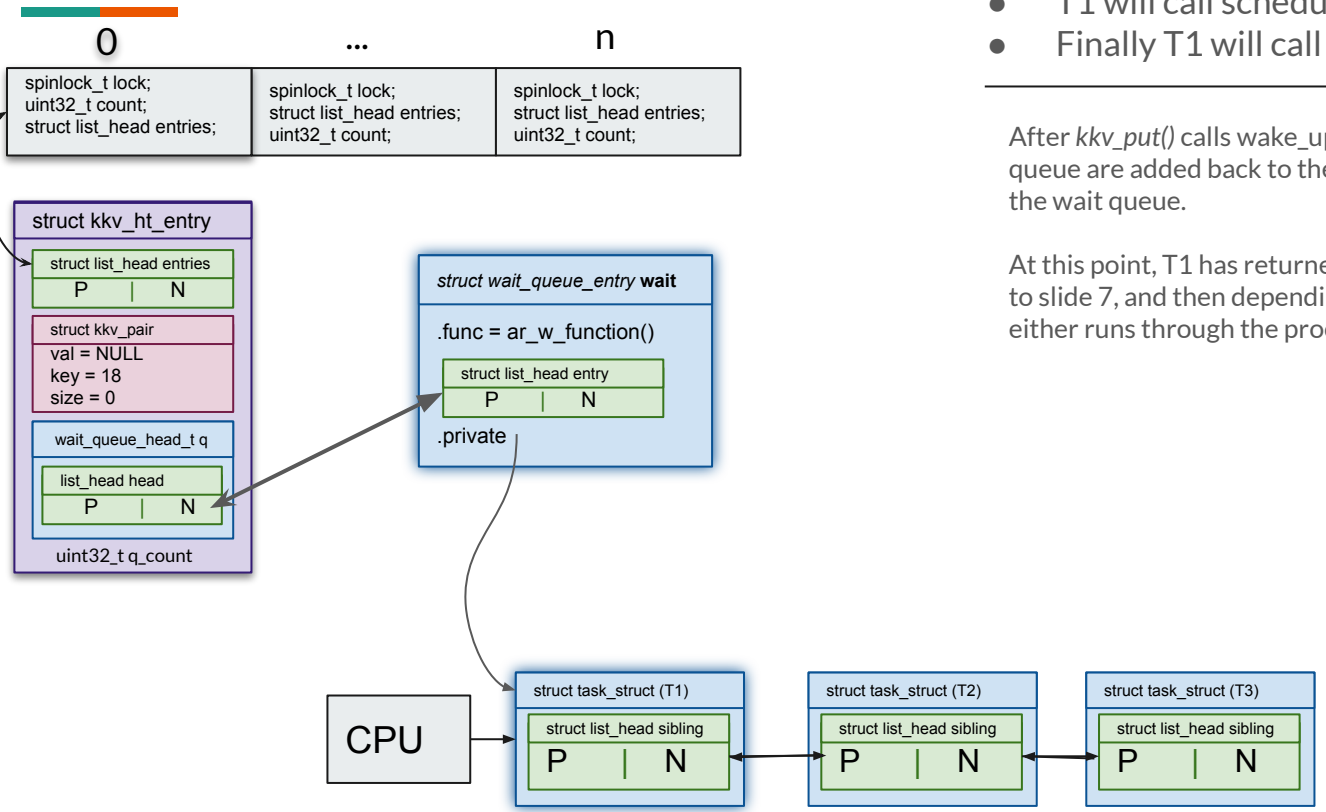
The Process of putting yourself to sleep



- T1 will call the macro DEFINE_WAIT()
- T1 will call prepare_to_wait()
- **T1 will call schedule()**
- Finally T1 will call finish_wait()

When T1 calls schedule() the OS will remove the task from the runqueue. It will not be put back onto the queue until someone wakes it up. That will happen when a process calls *kkv_put(18)*.

The Process of putting yourself to sleep

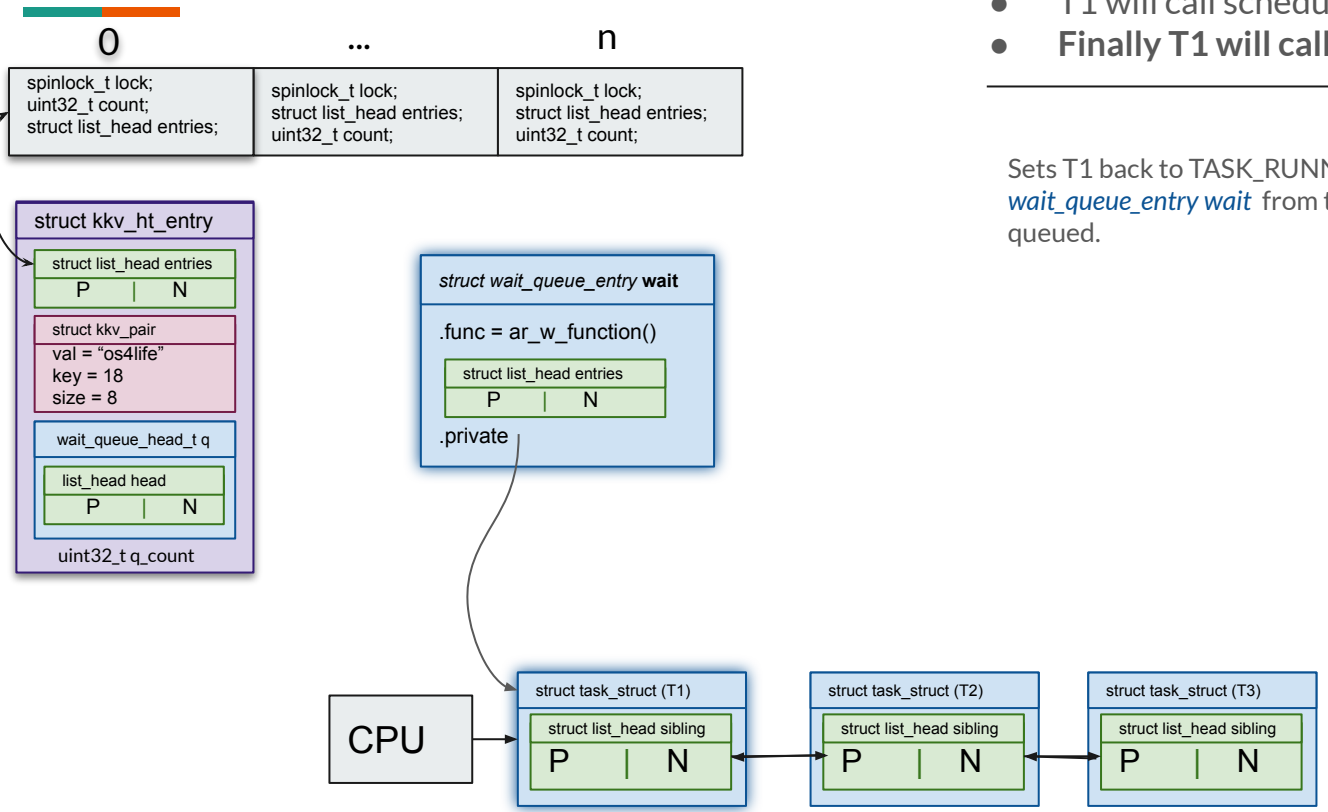


- T1 will call the macro DEFINE_WAIT()
- T1 will call prepare_to_wait()
- T1 will call schedule()
- Finally T1 will call finish_wait()

After *kkv_put()* calls wake_up(), all the processes on the wait queue are added back to the run queue, while remaining on the wait queue.

At this point, T1 has returned from schedule(). It then returns to slide 7, and then depending on the value of the condition, either runs through the process again, or exits the loop.

The Process of putting yourself to sleep



- T1 will call the macro DEFINE_WAIT()
- T1 will call prepare_to_wait()
- T1 will call schedule()
- **Finally T1 will call finish_wait()**

Sets T1 back to TASK_RUNNING and removes struct *wait_queue_entry wait* from the *wait_queue_head_t q* if still queued.