Story of a tick

A story about the events happening in pintos kernel, during a context switch.

In the story I assumed we have thread1, thread2 and thread3 that get scheduled in a round robin fashion:
thread1->thread2->thread3->thread1->....
Otherwise we could have other threads on PREV and NEXT in switch_threads

Also, it's just a story! So not everything is accurate.

CPU was running happily under thread1 context, playing around with the registers, jumping on the branches, and modifying memory bytes. Then it came out of nowhere...

TICK....

The timer had signaled the CPU to show 1/TIMER_FREQ (defined in timer.h:8) has passed. thread1 wanted to continue playing with memory values, or maybe do some calculations, but it was too late, the quantum was over...

CPU urged to the "intr_handler" function (interrupt.c:345), there she was lost in front of all the interrupt vectors. There she had a monolith as her only clue, a intr_frame. She looked at it, it was an external interrupt, and there it was... she found the address 0x20. Excited, she looked at the interrupt vectors and took a dive into the mystical void function (interrupt.c:367)

It was the right one, it was written on it: "timer_interrupt". She remembered it from the first initialization (timer.c:39). She looked at thread1, and doubted... "I want to give this one another chance", but the timer_interrupt was of type "intr_handler_func" (timer.c:27) and she had no choice but to run the OS code. She quickly saved her registers onto the interrupt stack frame, and jumped to "timer_interrupt" function (timer.c:171)

There she increased the "ticks", a global variable defined in timer.c:21, and jumped into "thread_tick" (thread.c:123), where she was supposed to update thread1 (currently running) kernel_ticks, and there it was, the inevitable....

```
  /* Enforce preemption. */
  if (++thread_ticks >= TIME_SLICE)
    intr_yield_on_return ();
```

At this point (interrupt.c:222), she knew she would not get back to thread1 execution flow once returning from the interrupt. After shedding a couple of tears, she turned on the "yield_on_return". She was quickly pulled out of the void function, returning from each and every call. There she was at intr_handler again (interrupt.c:377) and had to finish what she had started. She closed her eyes, took a deep deep breath, and jumped into the thread_yield function...

She opened her eyes in thread_yield (thread.c:302), where she pushed the thread1 to the ready_list, and changed its status to THREAD_READY, and stepped into the well-known schedule function (thread.c:553), She would return from this function under thread1 context like a normal function call... However, that return could take quite some time...

There she found 3 cards, called PREV, CUR, and NEXT. She knew that these will play the main role in the magic that was about to happen.

Using a thread-selecting algorithm implemented in "next_thread_to_run" (thread.c:491) she picked thread2 from ready_list, wrote it on NEXT card. Holding thread1 in CUR card and a null PREV card, she stood in front of the mysterious mirror of "switch_threads" stub (switch.S).

There she could see herself standing in "switch_threads", but holding thread3 on CUR card and thread1 on NEXT...

How was it possible?? was it the same "switch_threads"? How come that all other threads were right in this exact function?

For a moment she felt a Deja Vu, but there was no thinking involved, this step was in assembly, she closed her eyes and just let it go....

It only took several machine instructions, during which she felt lost, out of place, losing memories, remembering the undone, deja vu's.... (switch.S)

She did not know how, but once she opened her eyes (thread.c:565), she was now on the "other side", now she was running under thread2 context! and had a thread1 written on PREV and had a very old NEXT card with thread3 writen on it, from her last switch from thread2.

She took a deep breath to find herself in the moment. As simple as that, yet surprisingly, it was over. She had switched...

Now it was thread2's turn. She had to take care of a couple of things in thread_schedule_tail (thread.c:516), with the interrupts still off, she activated thread2 by changing its status to THREAD_RUNNING, looked back at thread1 (written on her PREV card) to see if she had to "finish it" or not. thread1's status was THREAD_READY and not THREAD_DYING. She gave a smile, and let it live...

Once again she was pulled back, one function after another....

"thread_schedule_tail", then "schedule", then "thread_yield" (that had been called way long ago, more specifically two whole time quantums!), the unmerciful "intr_handler", and she found herself back in thread2, right where she had left off, at the exact line that she was frozen, two time quantums ago, when she had switched to thread3 context...