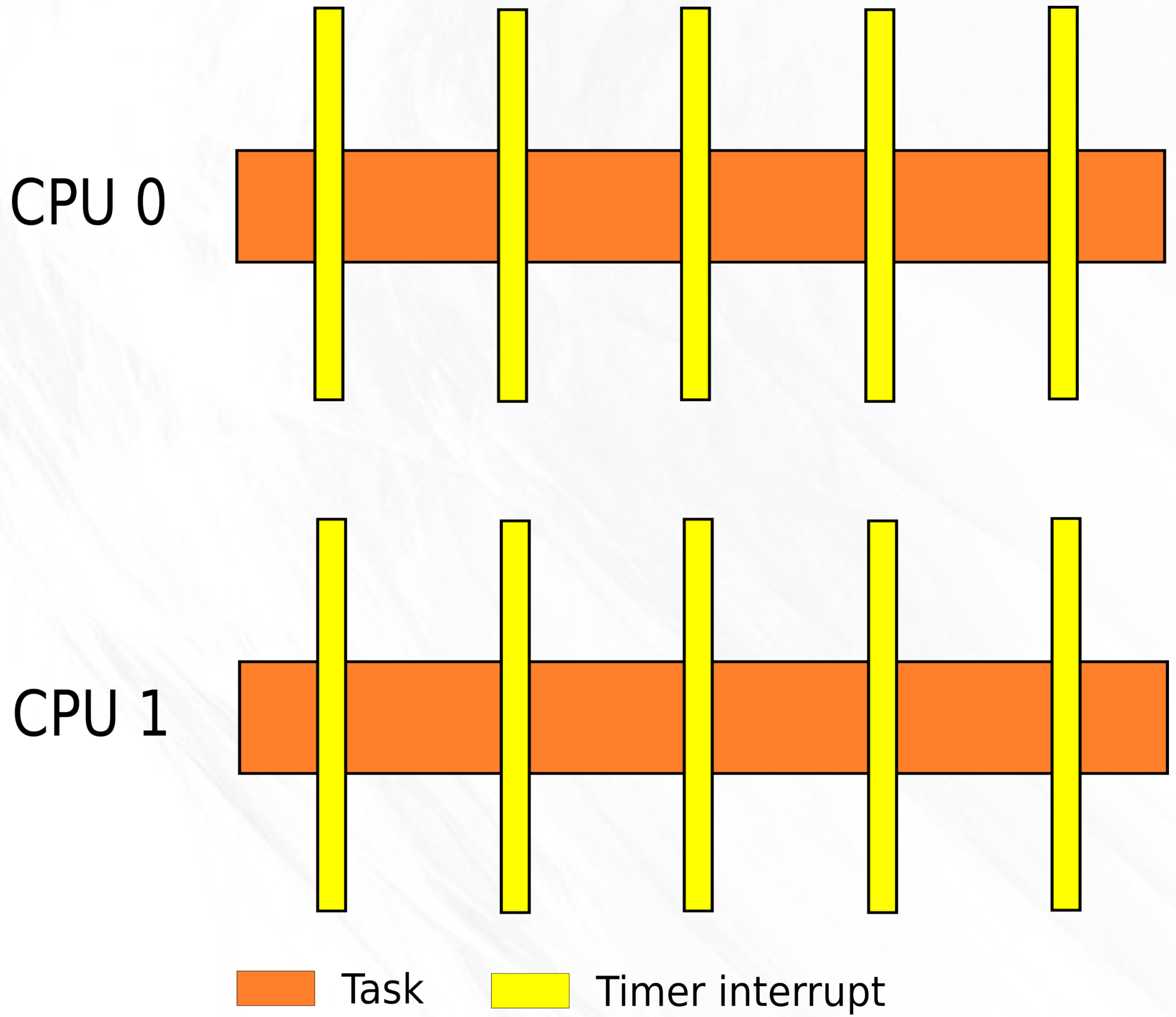
Status of Linux dynticks

Frederic Weisbecker <fweisbec@gmail.com>

What is the tick?

What is the tick?

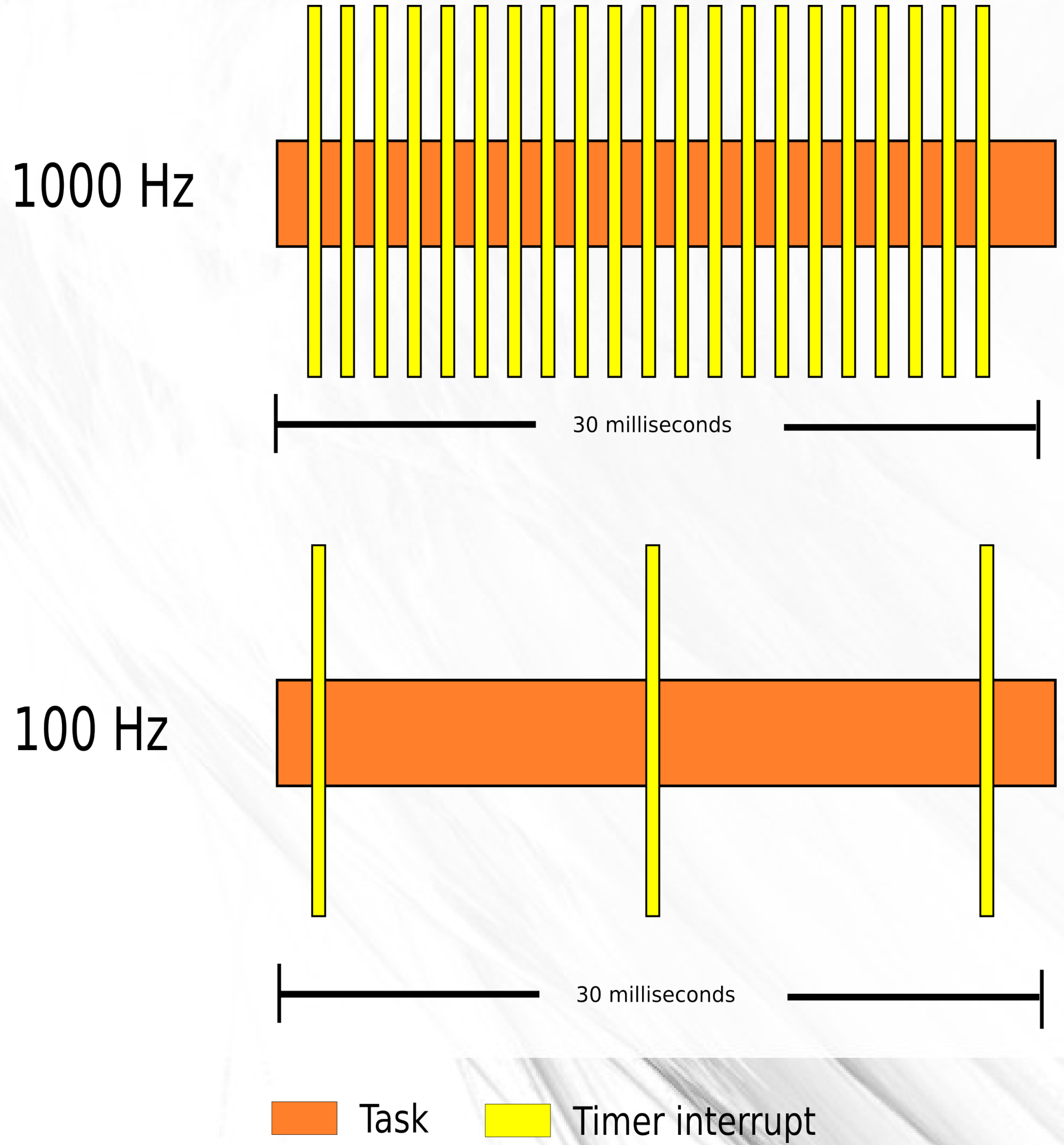
- A periodic interrupt


CPU 0
CPU 1
Task
Timer interrupt

What is the tick?

- A periodic interrupt

- Frequency depends on arch and hardware
  x86: 100 Hz, 250 Hz, 1000 Hz



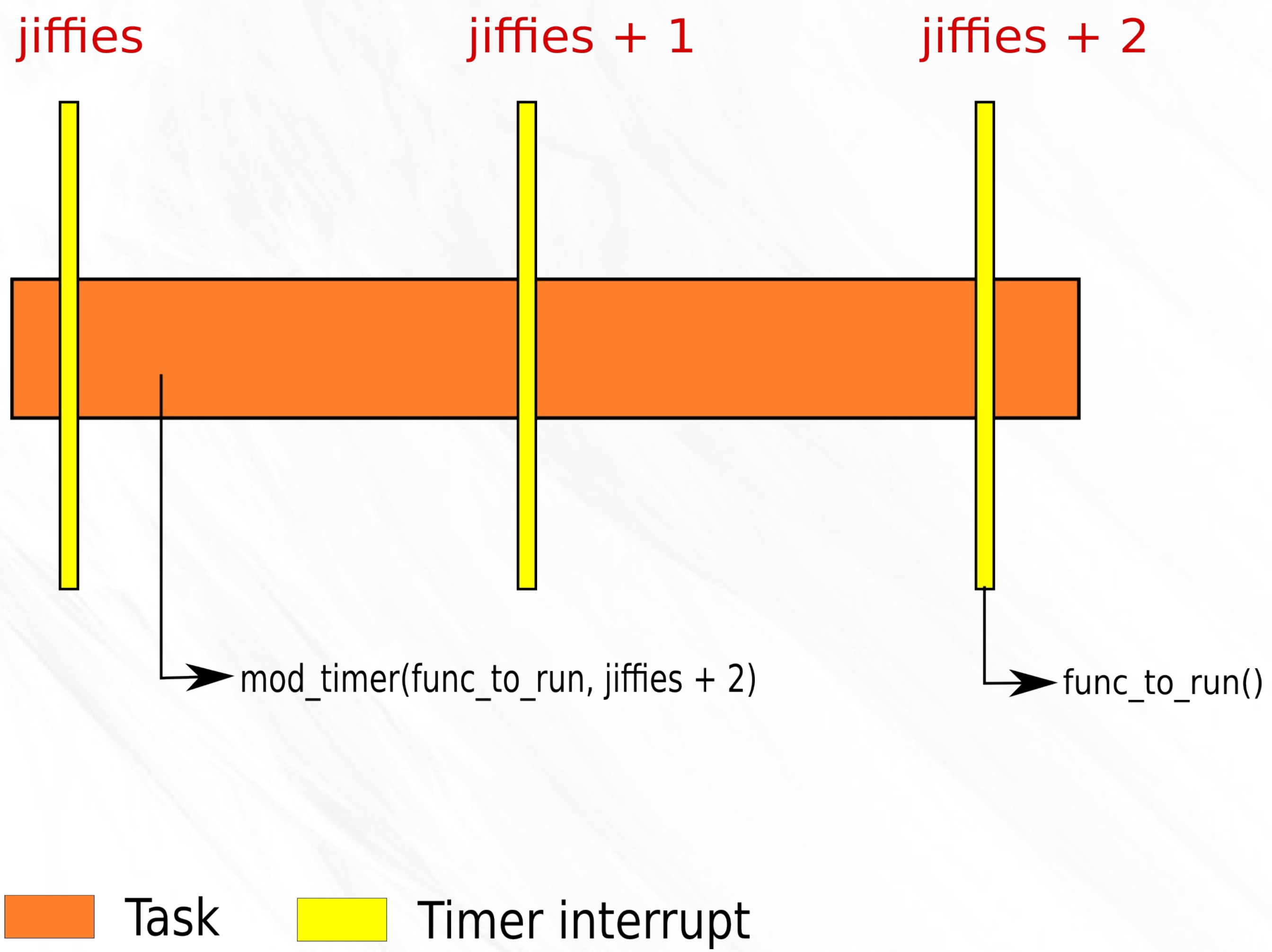
1000 Hz
30 milliseconds
100 Hz
30 milliseconds
Task
Timer interrupt

What is the tick?

- Low frequency (100 Hz): throughput, less interrupts, CPU less stolen, less cache trashed, ...
- High frequency (1000 Hz): latency, timer and scheduler granularity, cputime precision
- Hrtimer reduce low freq drawback (poll(), epoll(), ...)

What is the tick?

- Timekeeping (walltime, xtime, gettimeofday())
- Jiffies: relative, internal clock
- timer wheel: struct timer_list

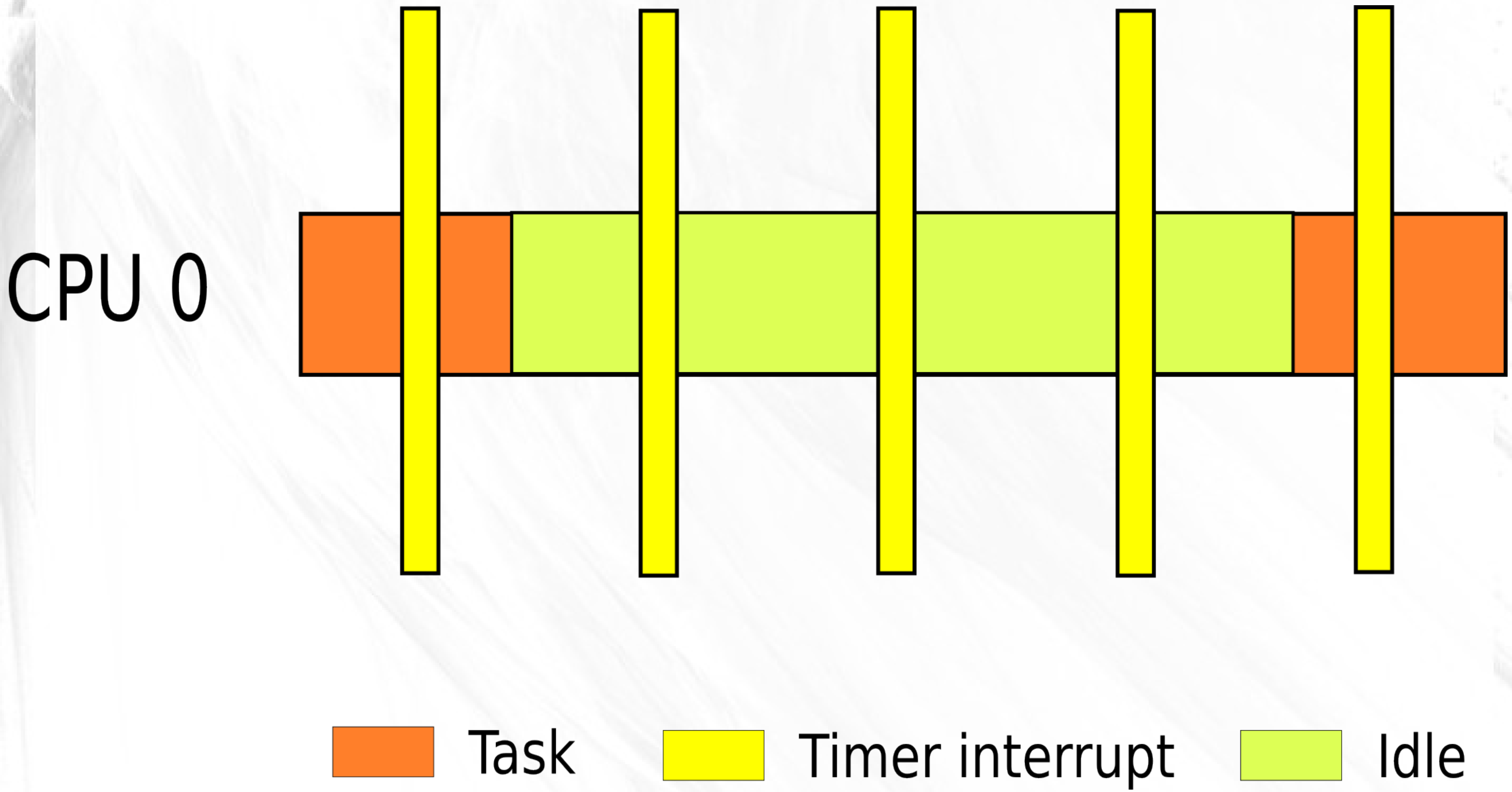


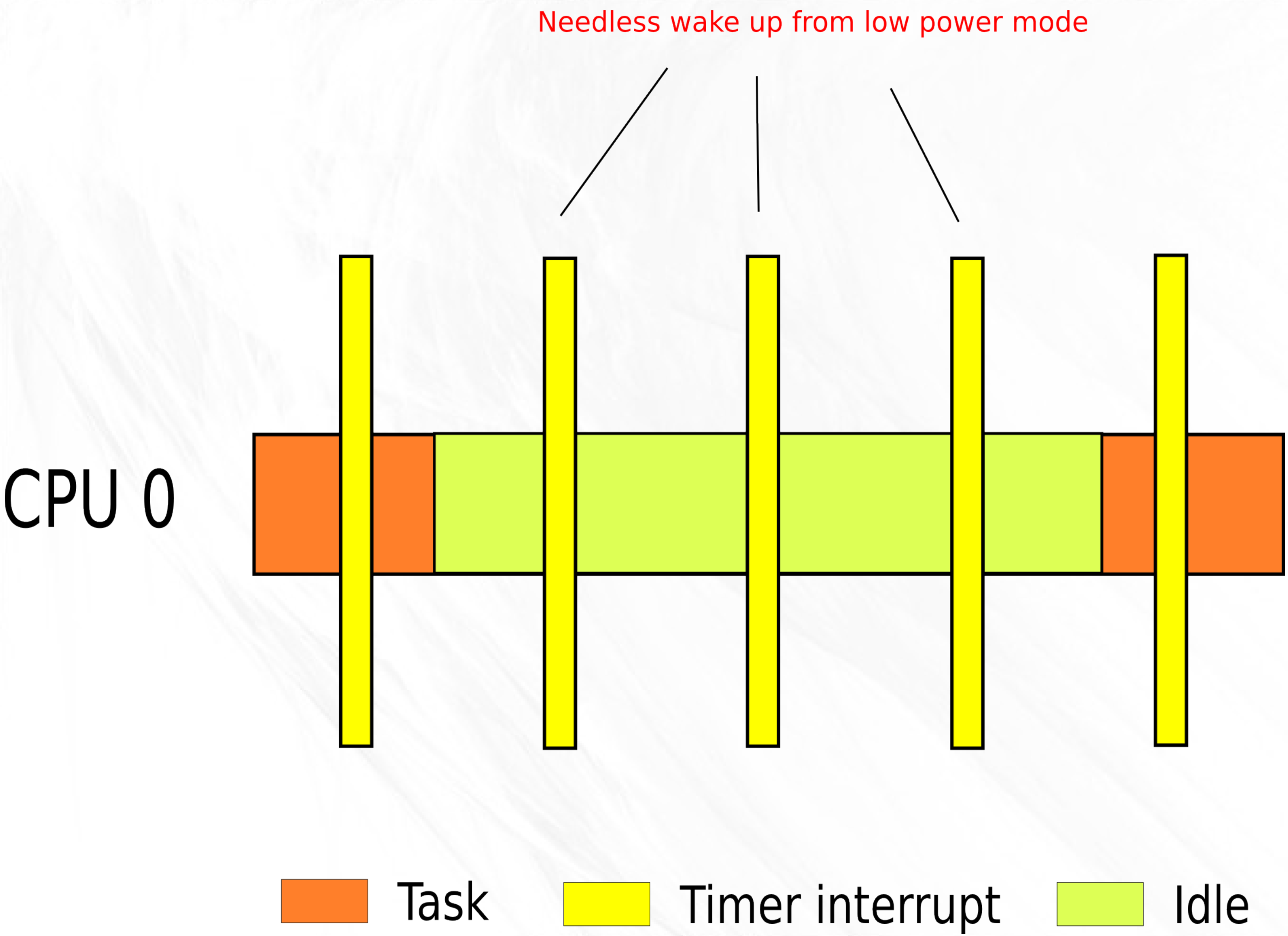
jiffies
jiffies + 1
jiffies + 2
mod_timer(func_to_run, jiffies + 2)
func_to_run()
Task
Timer interrupt

What is the tick?

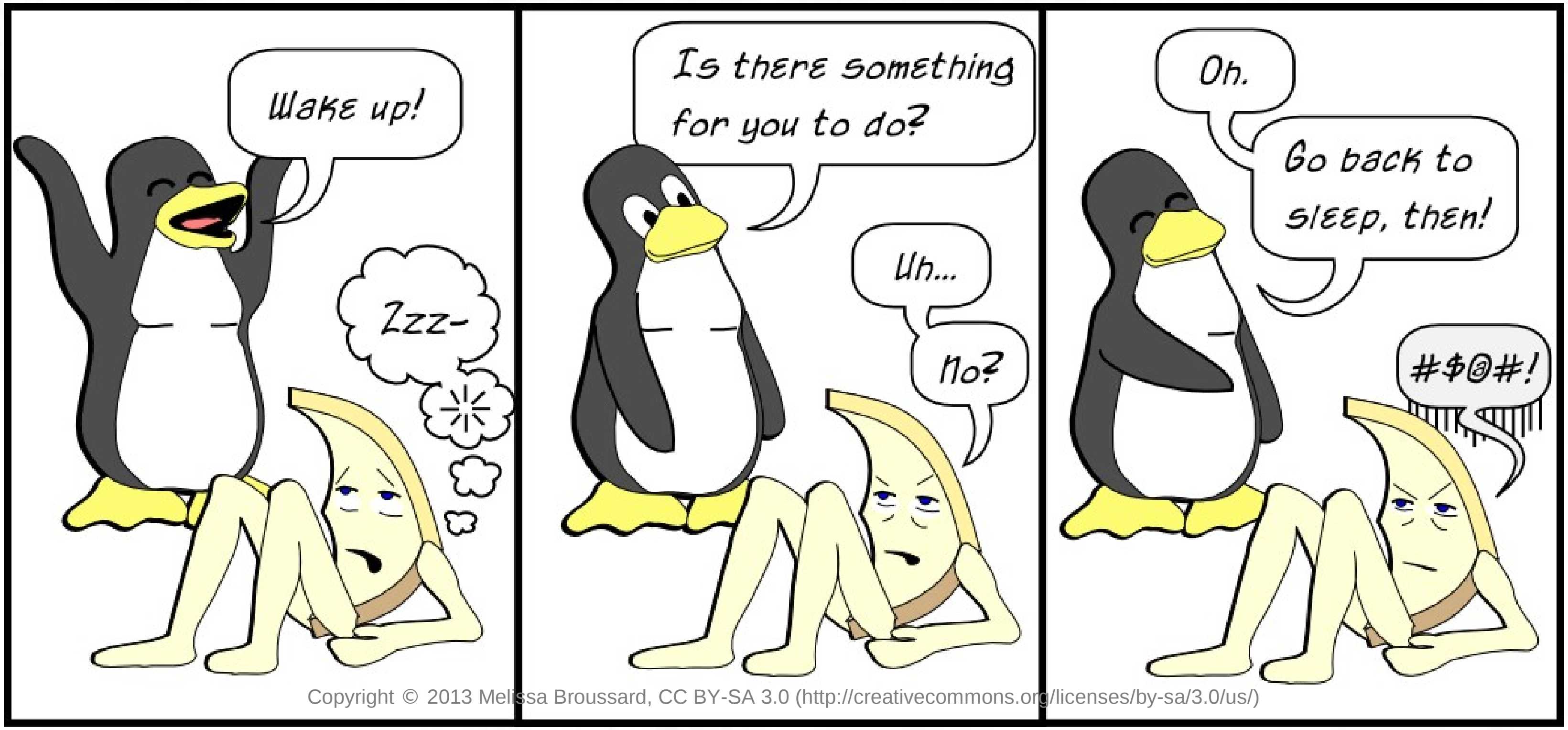
- Posix CPU timers (itimer, timer_settime, RLIM_CPU, ...)
- Cputime
- Scheduler (local and global fairness, bandwidth, load/time accounting...)
- RCU

Is it free?


CPU 0
Task
Timer interrupt
Idle

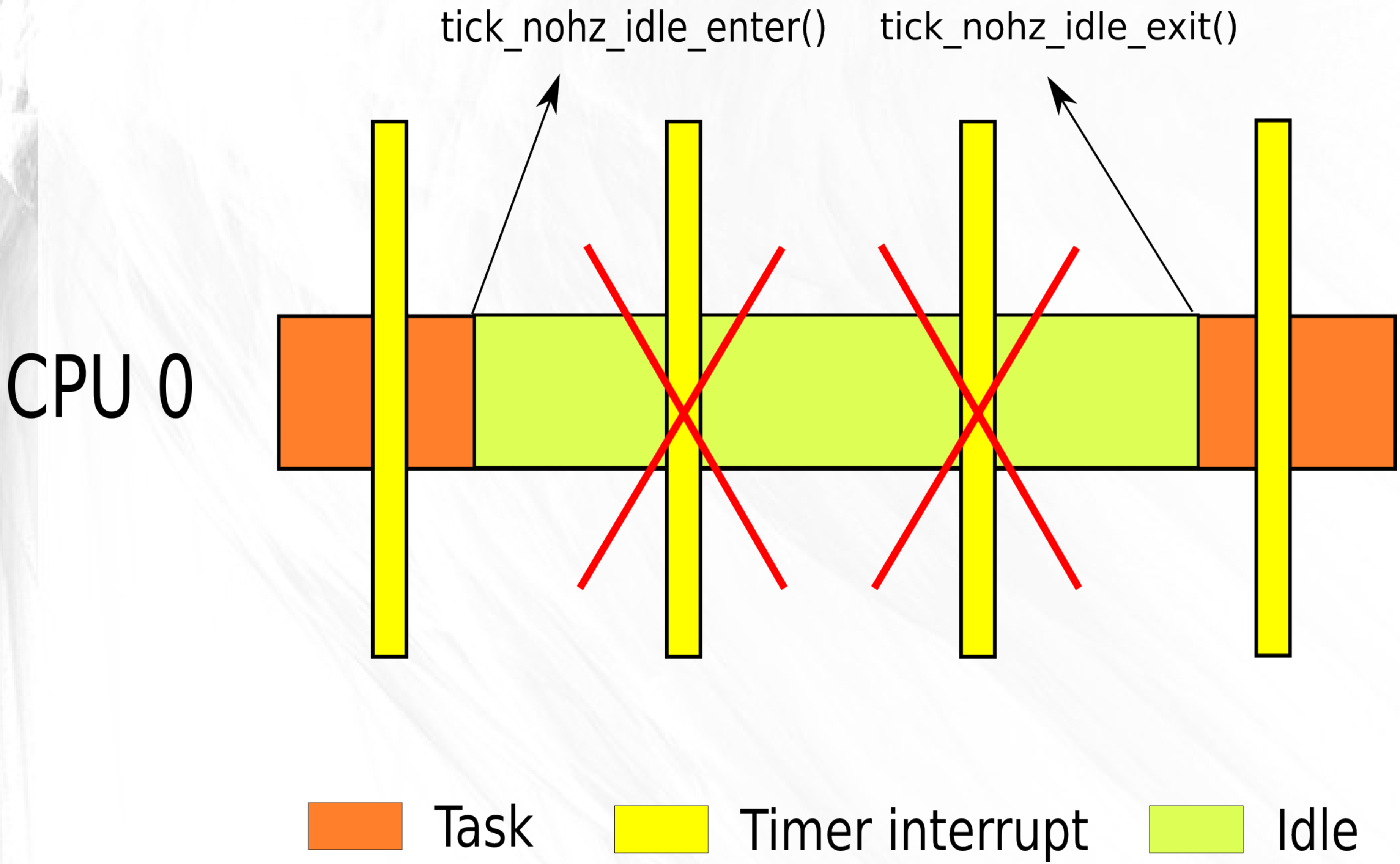

Needless wake up from low power mode
CPU 0
Task
Timer interrupt
Idle

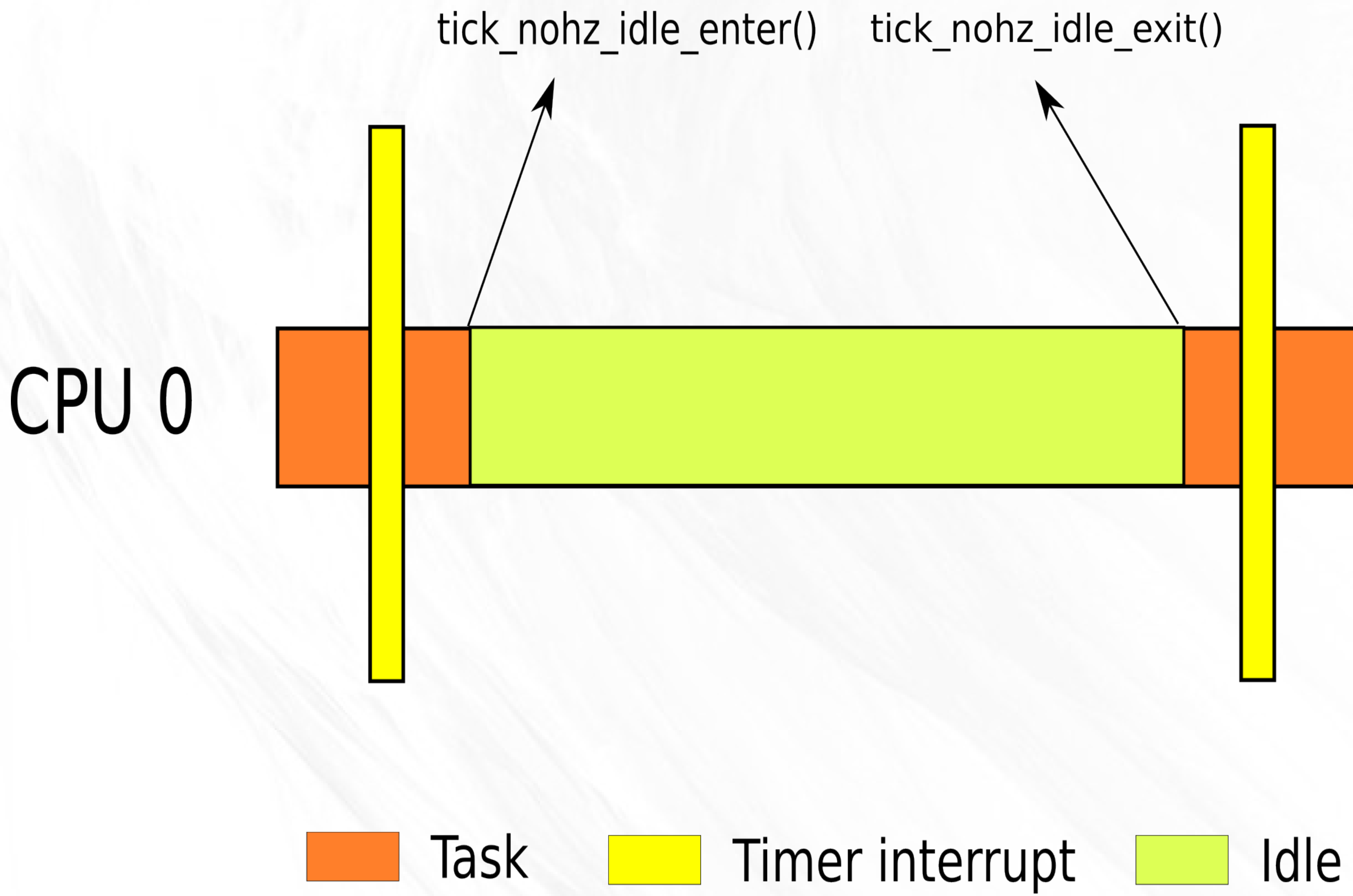
Is it free?


Wake up!
Zzz-
Is there something for you to do?
Uh...
No?
Oh.
Go back to sleep, then!
#$@#!

Is it free?

- Dynticks idle merged in 2.6.21 (2007)


tick_nohz_idle_enter()
tick_nohz_idle_exit()
CPU 0
Task
Timer interrupt
Idle


tick_nohz_idle_enter()
tick_nohz_idle_exit()
CPU 0
Task
Timer interrupt
Idle

Is it free?

- Dynticks idle merged in 2.6.21 (2007)
- Dynticks != Tick-free
- Fix power issue side of the tick
- CPU can enter deep C-states

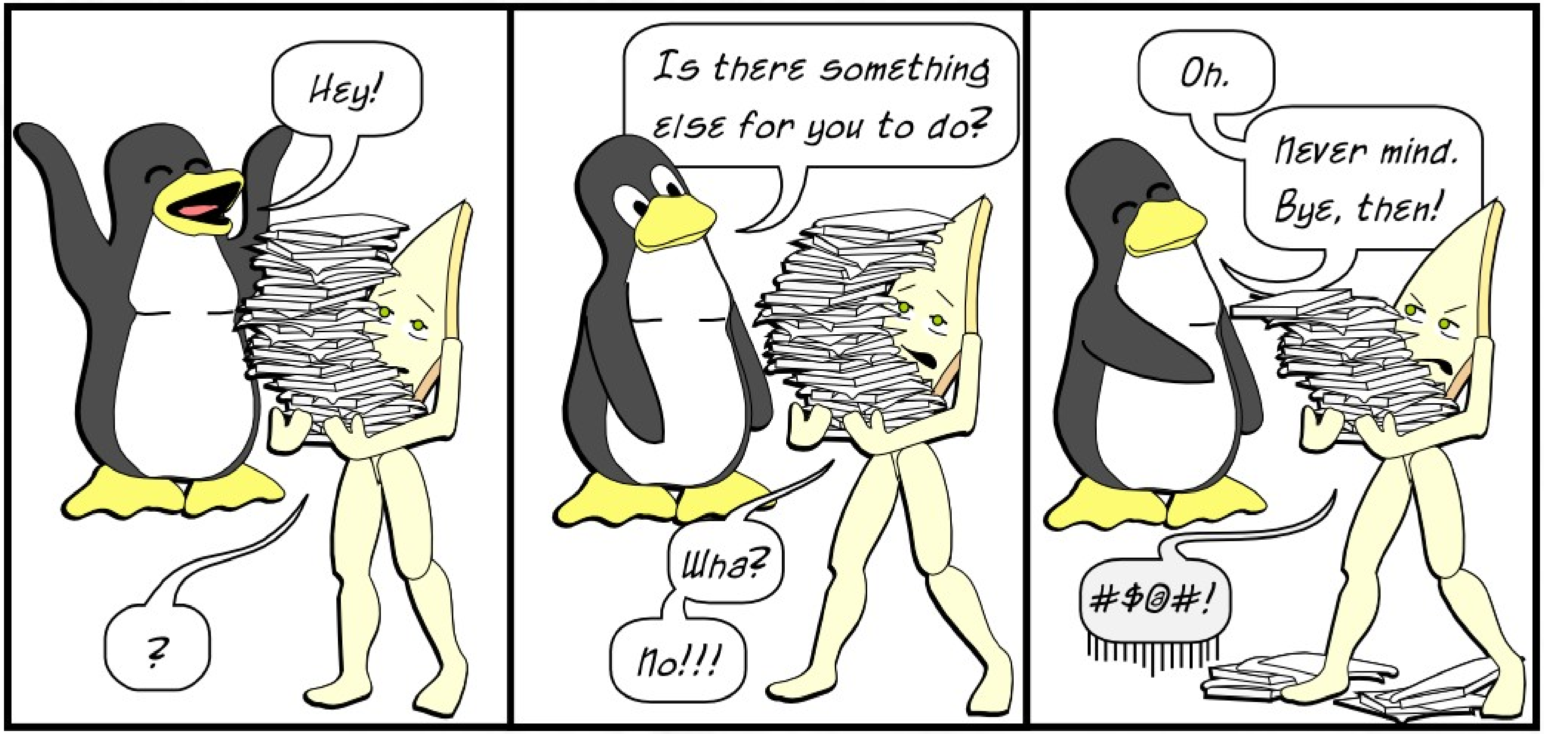
Is it free?

- So now it's free, right?

Is it free?

- So now it's free, right? No!
- Tick steals CPU 100 to 1000 times per secs
- Icache, dcache periodically trashed

Is it free?


Hey!
?
Is there something else for you to do?
Wha?
No!!!
Oh.
Never mind. Bye, then!
#$@#!

Is it free?

- Who complains?
- HPC: extreme throughput
- Real time: extreme latency

Is it free?

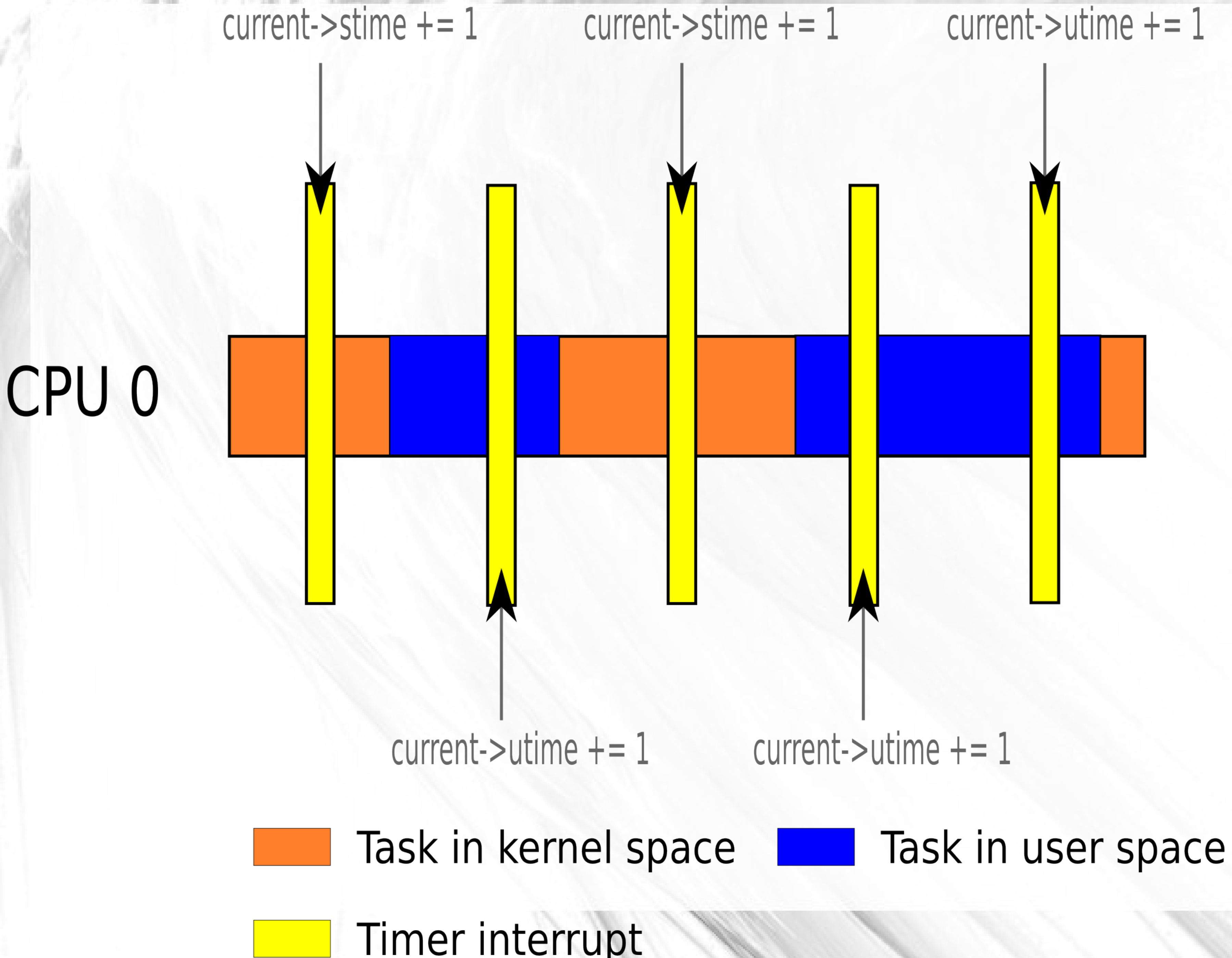
- Need to stop tick even on busy CPUs
- Full dynticks merged in 3.10
- But to stop the tick comes at various costs and requirements
- Poll driven -> Event driven

Cputime accounting

- Poll on task execution

- Account tick to interrupted ring:
  - Userspace (task->utime)
  - Kernelspace (task->stime)
  - Fine grained details (guest, hardirq, softirq, ...)

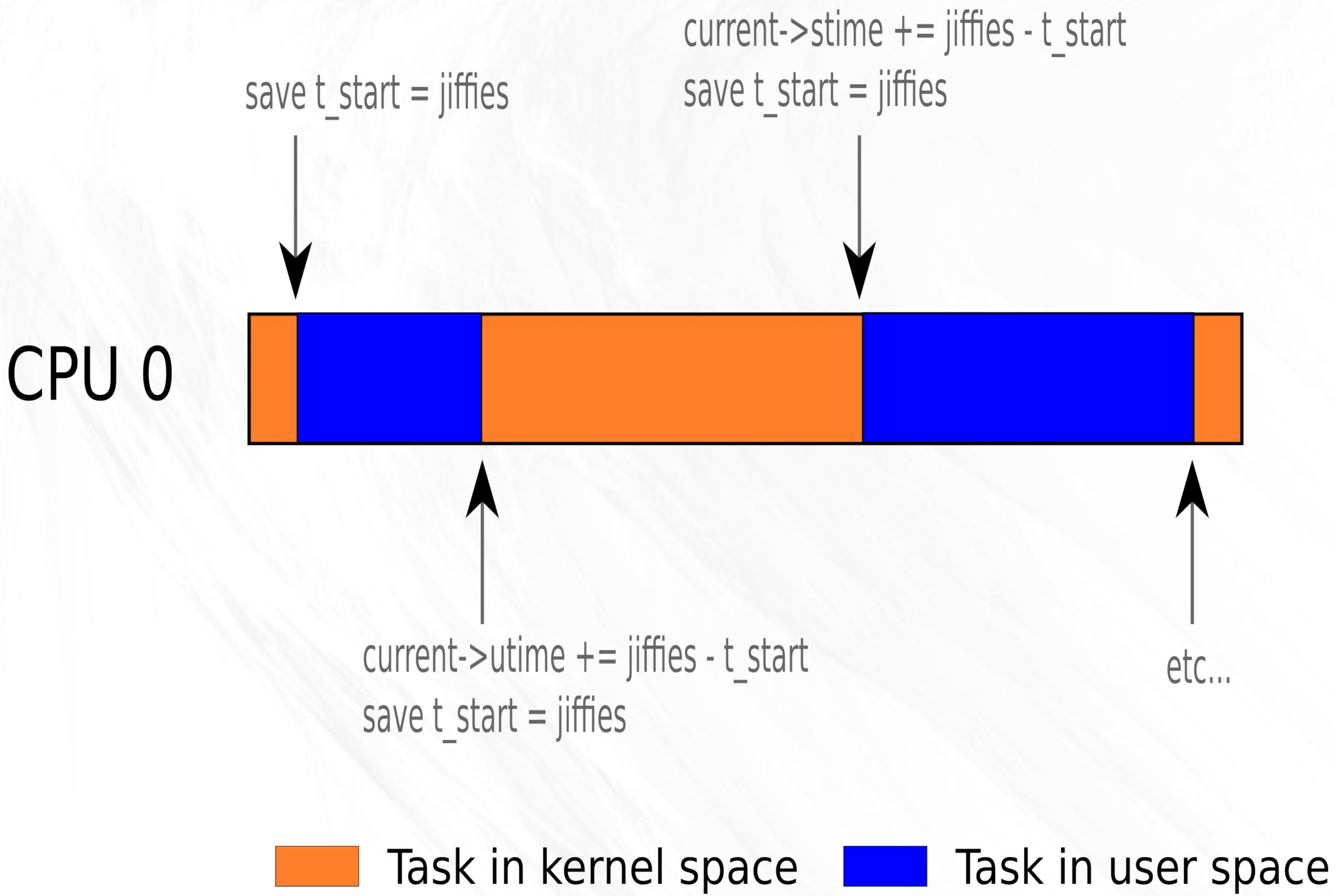
- CONFIG_TICK_CPU_ACCOUNTING


current->stime += 1
current->stime += 1
current->utime += 1
CPU 0
current->utime += 1
current->utime += 1
Task in kernel space
Task in user space
Timer interrupt

Full dynticks: Cputime accounting

- Convert to event driven accounting

- Listen to ring boundaries, account delta:
  - Syscall entry/exit
  - Exception entry/exit (traps, faults, ...)
  - Irqs entry/exit


current->stime += jiffies - t_start
save t_start = jiffies
save t_start = jiffies
CPU 0
current->utime += jiffies - t_start
save t_start = jiffies
etc...
Task in kernel space
Task in user space

Full dynticks: Cputime accounting

- Convert to event driven accounting
- Listen to ring boundaries, account delta:
  - Syscall entry/exit
  - Exception entry/exit (traps, faults, ...)
  - Irqs entry/exit
- Hooks overhead

RCU

- Kernel lockless synchronization
- Read sides can run concurrently with the writer
- Synchronize object lifecycles

RCU

READ SIDE

```
rcu_read_lock()
p = rcu_dereference(rcu_object)
if (p != NULL)
        do_something_with(p)
rcu_read_unlock()
```

RCU

WRITE SIDE

```
p = rcu_object
rcu_assign_pointer(rcu_object, new_object)
synchronize_rcu()
kfree(p)
```

RCU

1) Writer updates the object pointer

=> rcu_assign_pointer()

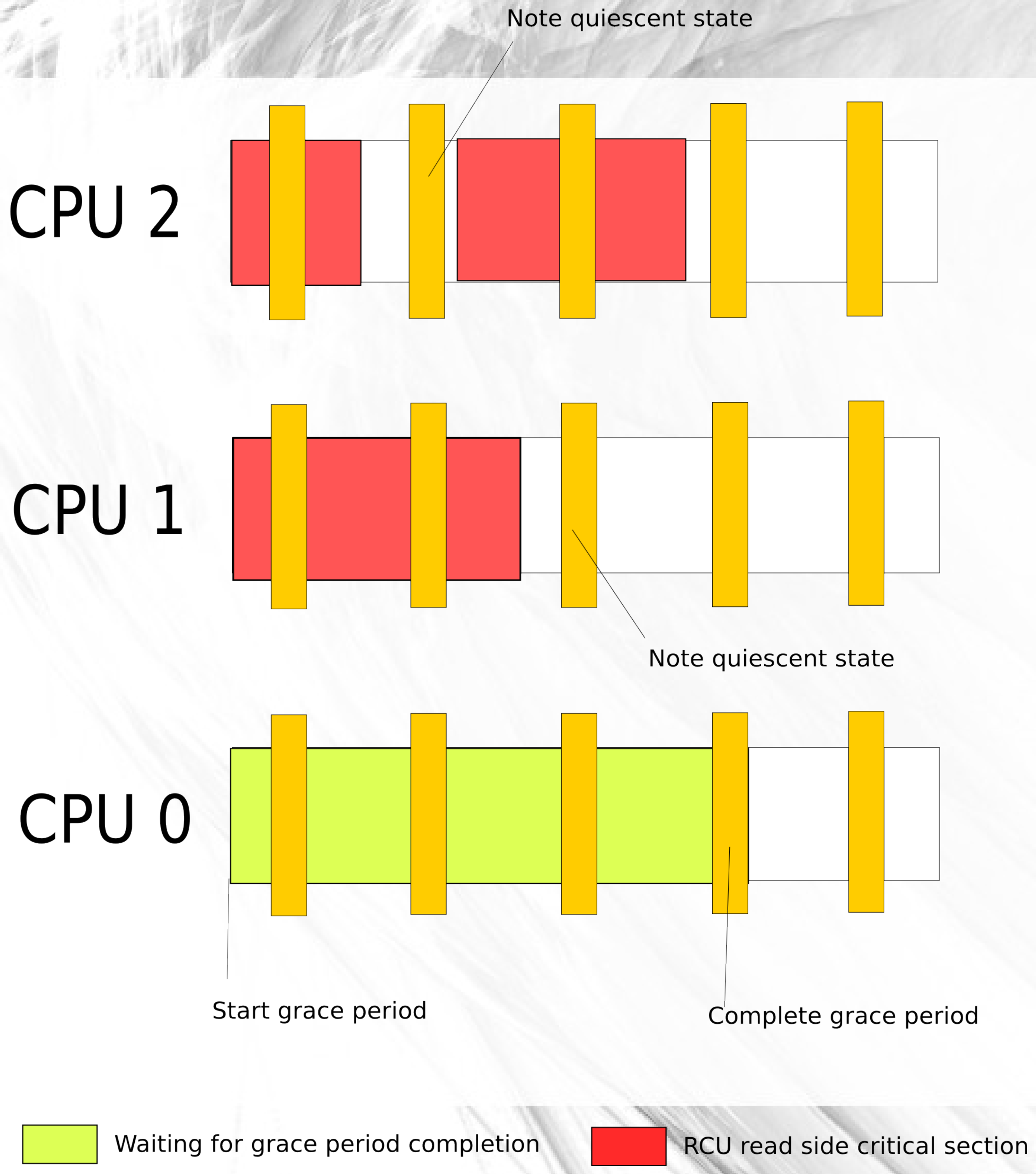
2) Starts grace period (finish when no more reader is using or can access old value)

=> synchronize_rcu(), call_rcu(...)

3) Guarantee old value not visible anymore: remove old object

RCU

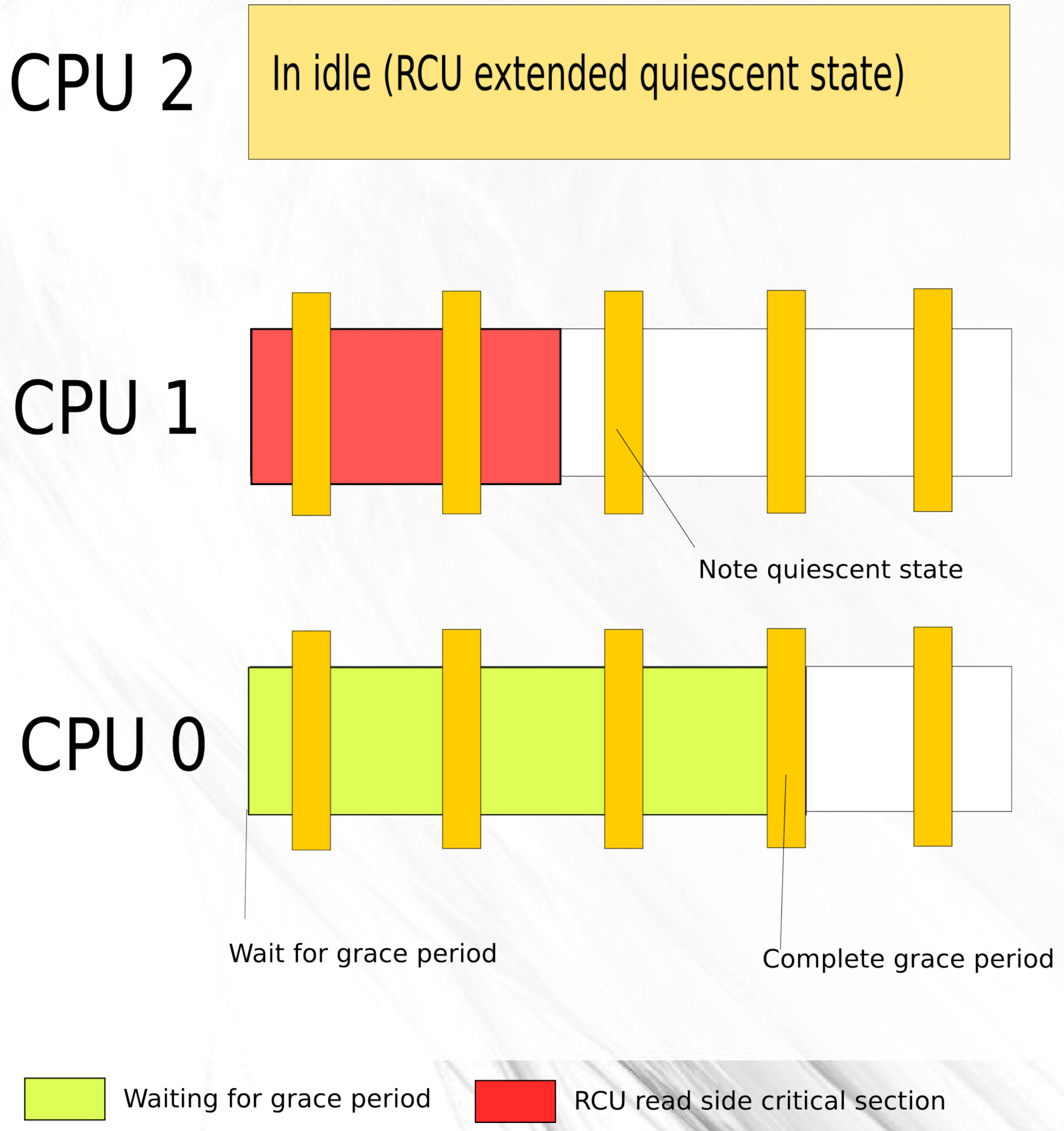
- Quiescent state = CPU not using RCU
- All CPU report a quiescent state: grace period end.
- Grace period => global state machine, all CPU participate
- Poll on quiescent states through tick



Note quiescent state
CPU 2
CPU 1
Note quiescent state
CPU 0
Start grace period
Complete grace period
Waiting for grace period completion
RCU read side critical section

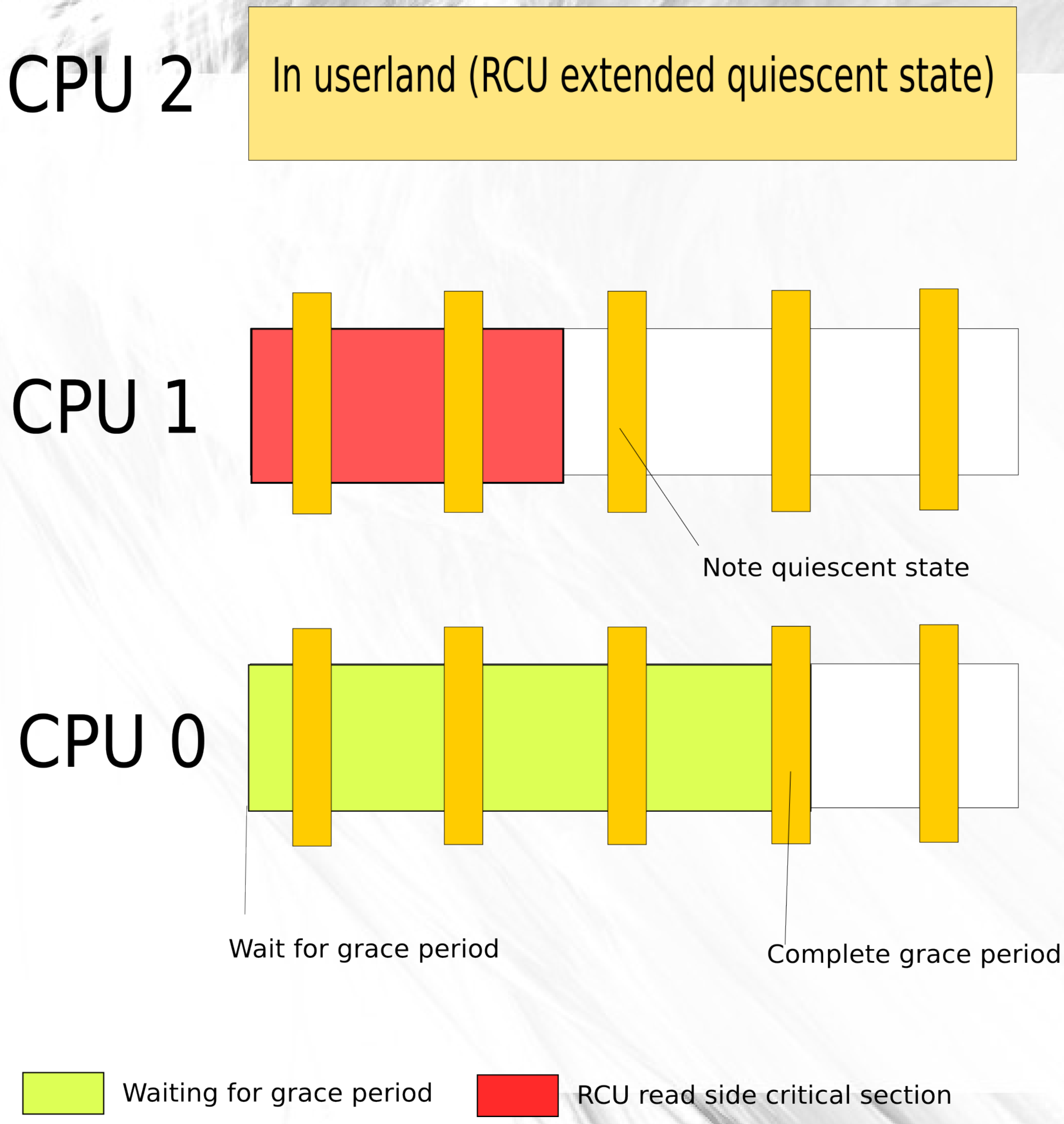
RCU

- Extended quiescent state = CPU not using RCU and no polling on quiescent states
- Passive part of global state machine, no quiescent state request (ie: no need for tick)
- Useful for dynticks
- Idle = extended quiescent state, to enforce powersaving


CPU 2
In idle (RCU extended quiescent state)
CPU 1
Note quiescent state
CPU 0
Wait for grace period
Complete grace period
Waiting for grace period
RCU read side critical section

Full dynticks: RCU

- Userspace don't use RCU
- Userspace = extended quiescent (CONFIG_RCU_USER_QS)
- Dynticks possible in userspace


CPU 2
In userland (RCU extended quiescent state)
CPU 1
Note quiescent state
CPU 0
Wait for grace period
Complete grace period
Waiting for grace period
RCU read side critical section

Full dynticks: Timekeeping

- Tickless busy CPU can use jiffies/walltime
- Unlike dyntick idle, need maintained timekeeping
- Need a periodic timekeeper (boot CPU)
- Big powersaving issue right now (solution from Paul Mckenney in the way)

Full dynticks: single task

- Need local fairness if more than a task runs (preemption)
- Only stop tick if single task on CPU
- Future: hrtick ?

Full dynticks: 1 Hz hack

- Still some work needed on scheduler
- Load balancing, various accounting stats, load average, etc...
- Keep 1 Hz at most until it gets solved

References

- Documentation/timers/NO_HZ.txt
- (Nearly) full tickless operation in 3.10
  - http://lwn.net/Articles/549580/
- “The 2012 realtime minisummit” (LWN, CPU isolation discussion)
  - http://lwn.net/Articles/520704/
- “NoHZ tasks” (LWN)
  - http://lwn.net/Articles/420544/
- Linux kernel development, Robert Love
- Bare metal multicore performance in a general purpose Operating System, Paul McKenney

Thanks!

- Josh Triplett: First prototype (LPC 2009)
- Steven Rostedt: Lots of code review and comments, tracing upgrades
- Christoph Lameter: Early adopter feedback
- Li Zhong: Power port
- Geoff Levand, Kevin Hilman: ARM port
- Peter Zijlstra: Scheduler-related review, comments, and work
- Paul E. McKenney: Read-copy update (RCU) work (fun with “Hotel California” interrupts!)
- Thomas Gleixner, Paul E. McKenney: “Godfathers”
- Ingo Molnar: Maintainer
- Other contributors:
  - Avi Kivity, Chris Metcalf, Geoff Levand, Gilad Ben Yossef, Hakan Akkan, Lai Jiangshan, Max Krasnyansky, Namhyung Kim, Paul Gortmaker, Paul Mackerras, Peter Zijlstra, Steven Rostedt, Zen Lin (and probably many more)

Questions?