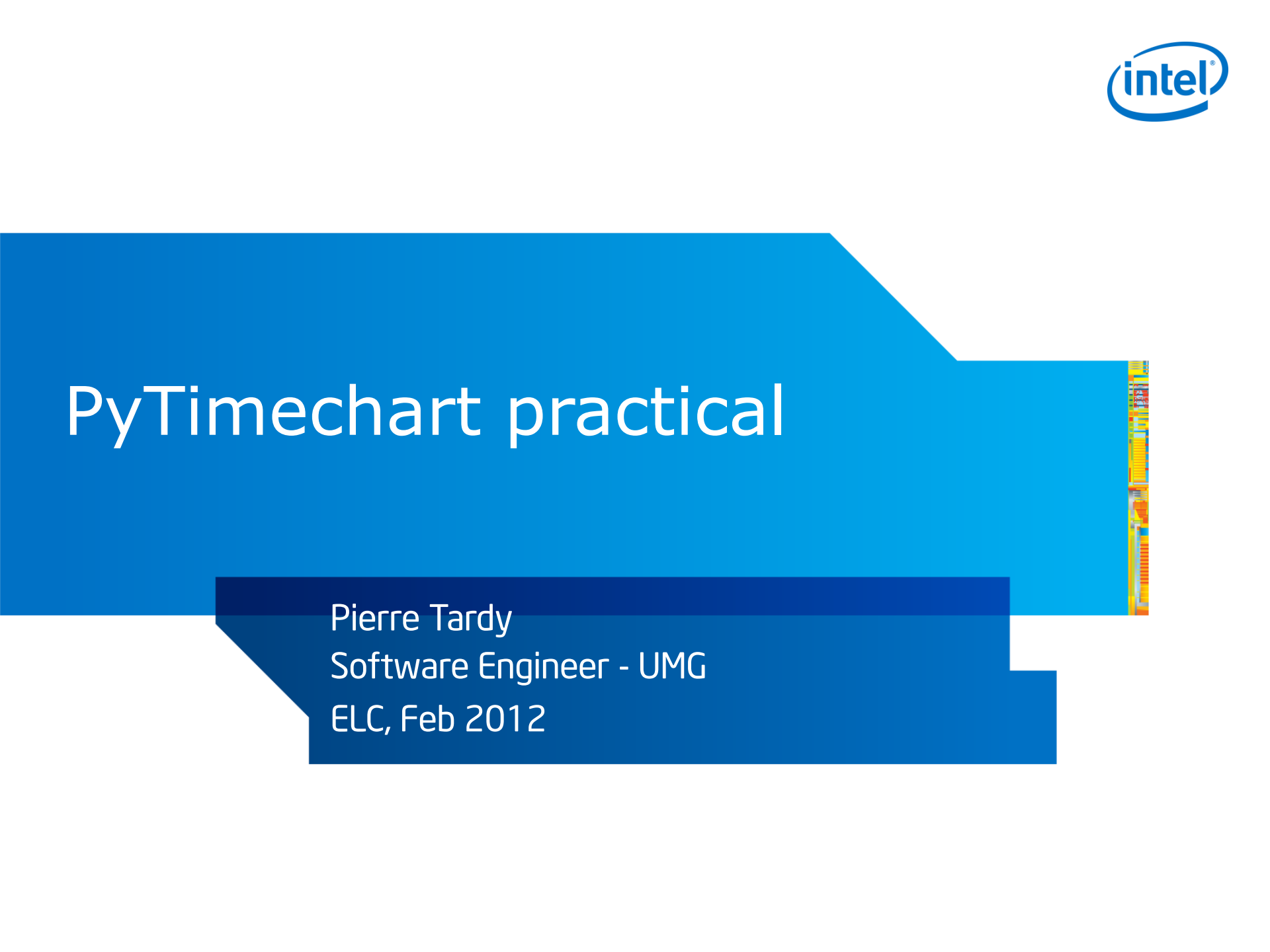


PyTimechart practical

Pierre Tardy
Software Engineer - UMG
ELC, Feb 2012

About the author

Intel Employee since 2009

Working on Intel's phone platforms

- Meego
- Android
- Power Management
- Tools (pytimechart, buildbot)
- Open-Source expertise

Formerly Freescale Employee

DVBH, LTE (gpe, oe, poky, cairo)

Agenda

I will not rephrase the documentation, you can access it following this link:

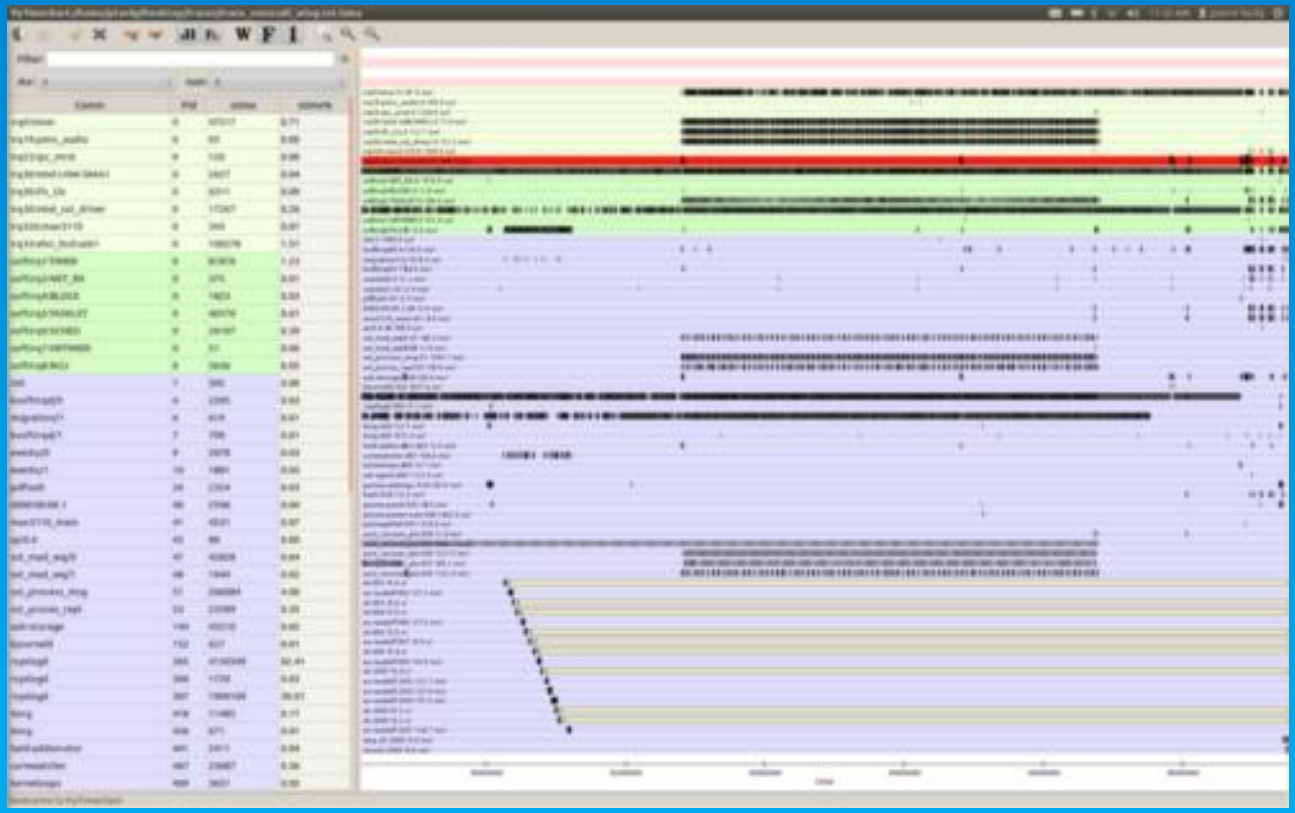
http://packages.python.org/pytimechart/index.html

Goal is to present the tool using real-world traces. See what you can get, what you won't get.

1. PyTimechart Overview
2. Audio Player usecase – Hunting the wakes
3. Bootcharting Ubuntu – Filtering with pytimechart
4. Modem driver traces - Hacking PyTimechart to decode your own traces

Questions?

PyTimechart Overview



PyTimechart Overview

Ftrace/perf are linux kernel tracing system.

- Very low overhead
- Already a lot of tracepoints available
- Can trace all function calls without kernel compile
- Hard to dive into trace
- Most people need visual representation or get lost

PyTimechart aims to let you quickly find what you are looking for

- Based on chaco, the browsing is very fast
- Based on python, parsing is very simple

PyTimechart Overview

One line per Process. Process in pytimechart can be:

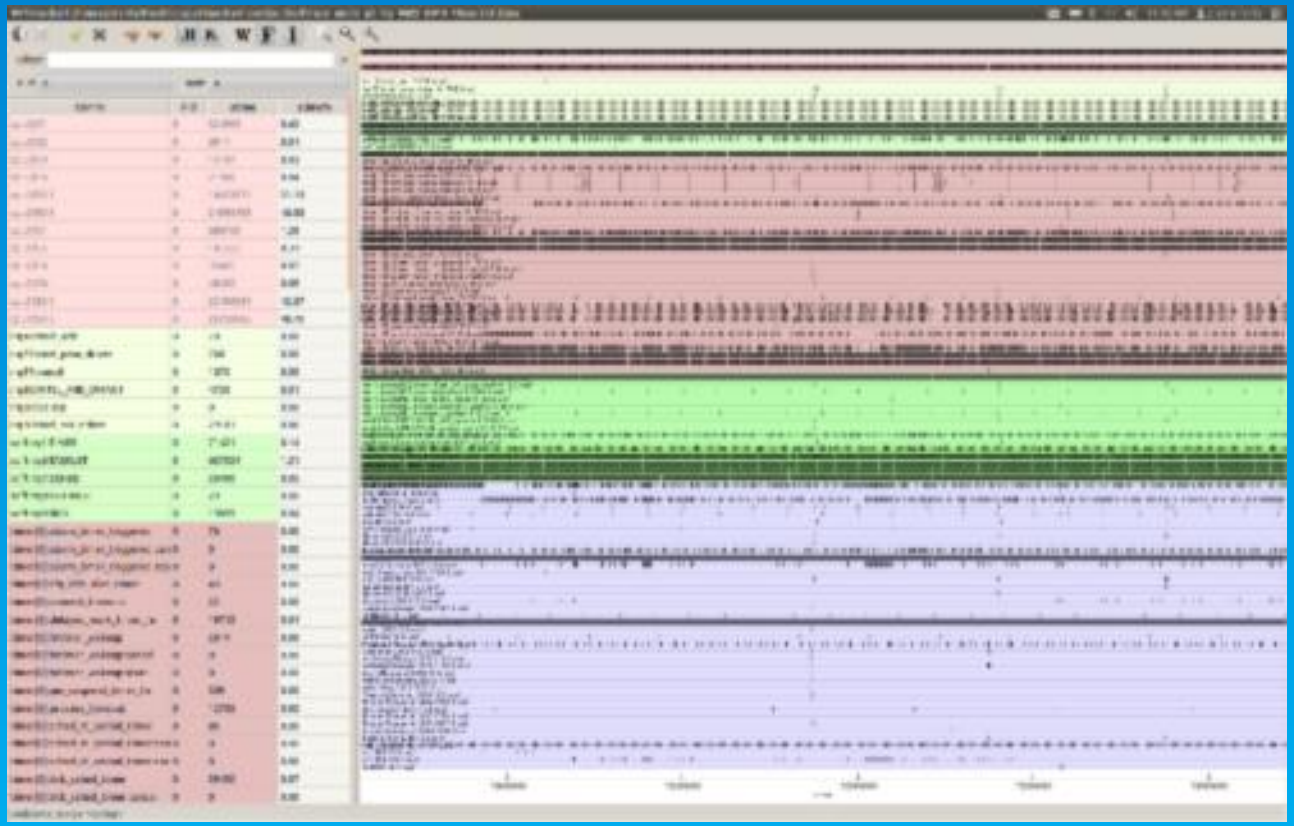
- An IRQ callback
- A workqueue
- A tasklet
- A linux process, identified by pid, and comm
- A power state (cpuidle, runtime_pm)

Simple browsing

- Zoom with mouse wheel
- Select a part of the chart
  - Show time range
  - Processor share
  - Original trace as text

Audio Player usecase

hunting the wakes



LowPower MP3 usecase

Playing mp3 is easy. Playing your whole music library in one battery charge is much harder.

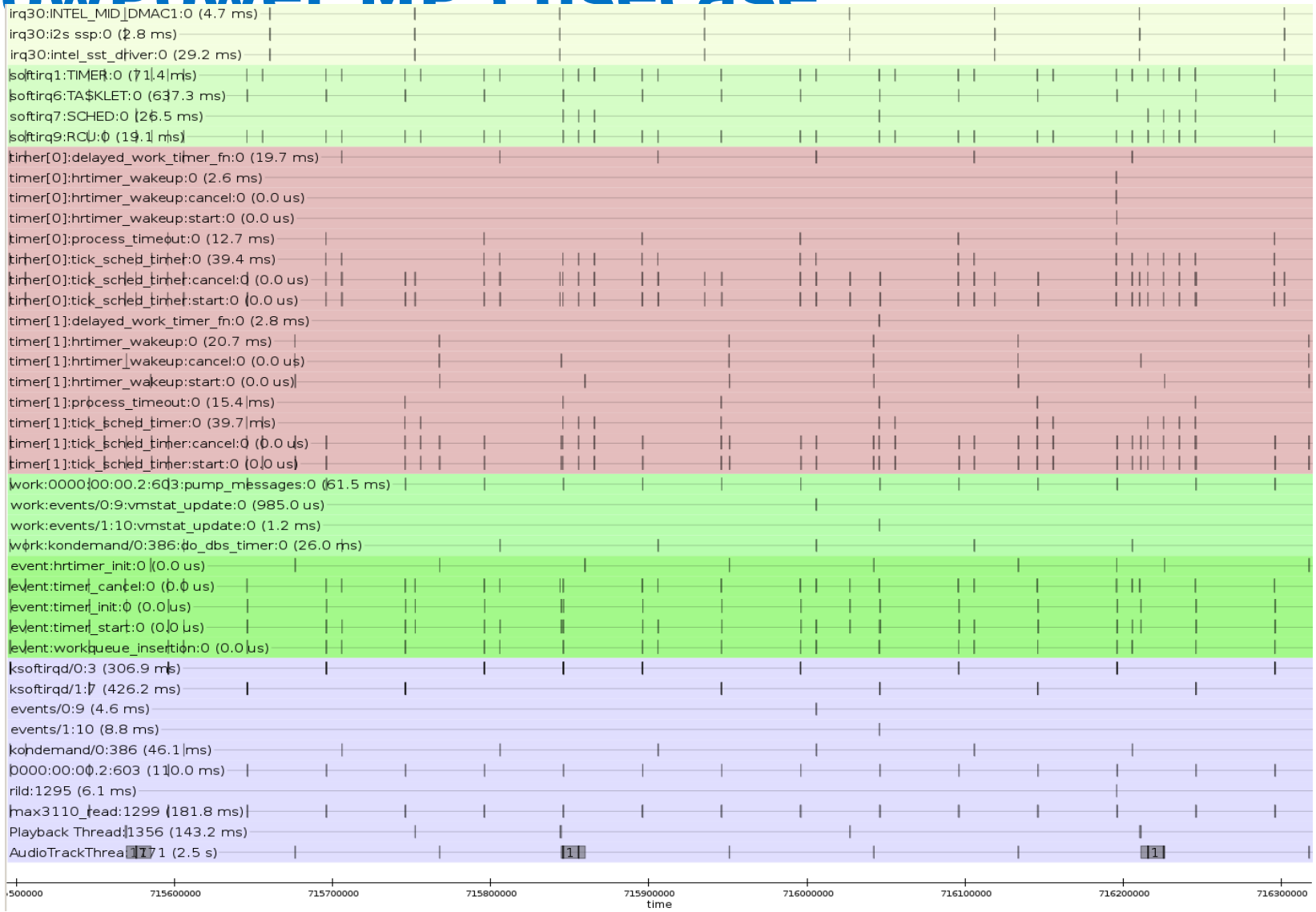
Need to improve wake up count. Each time the cpu wakes up, energy is spent (cache, PLL, etc)

Audio HW is generating IRQ when a new buffer is needed. This goes all the way through user space, where the mp3 decode is done.

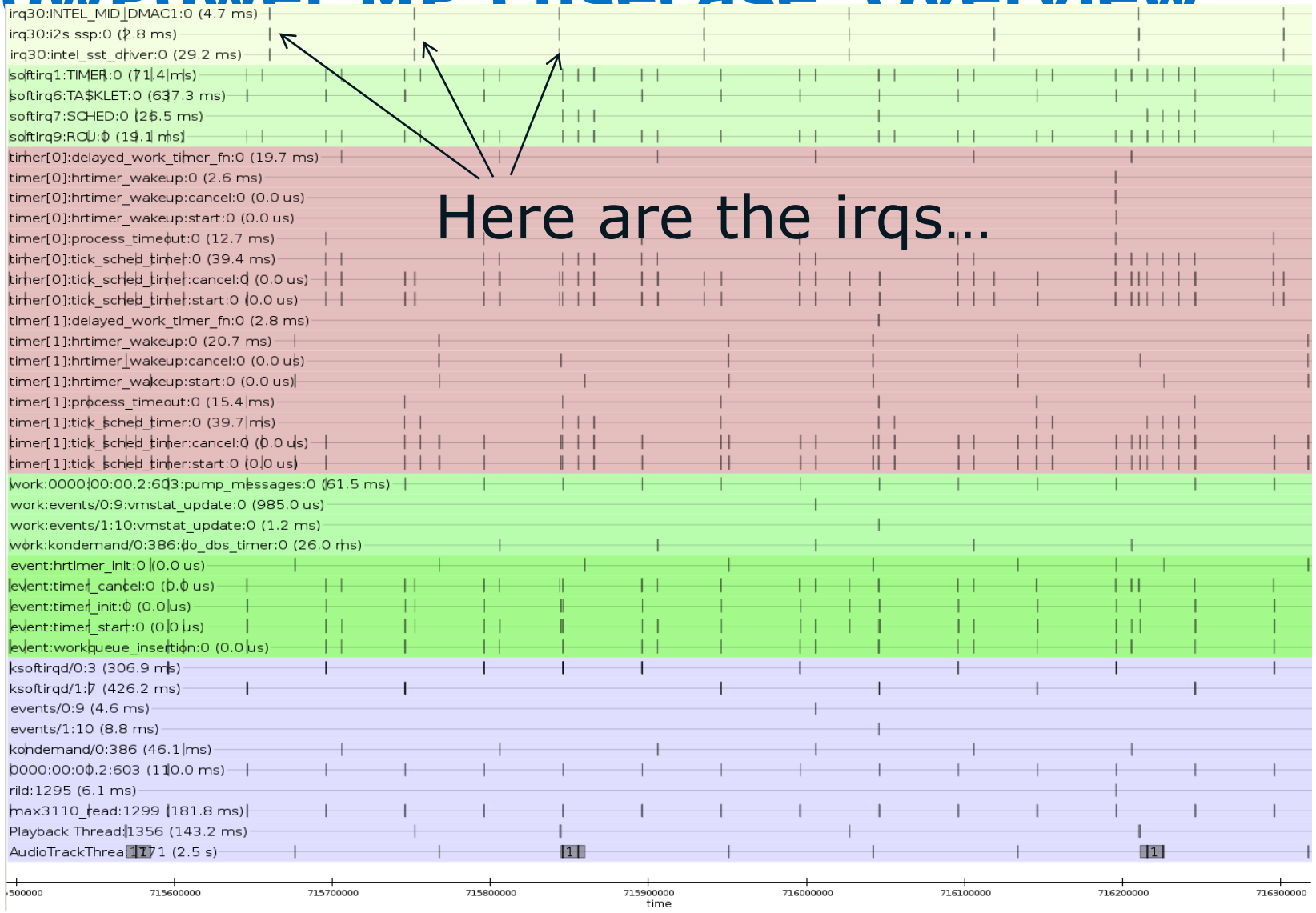
We are waiting for:

- Audio IRQ
- MP3Decode
- Sleep
- Audio IRQ
- MP3Decode
- Sleep
- Etc.

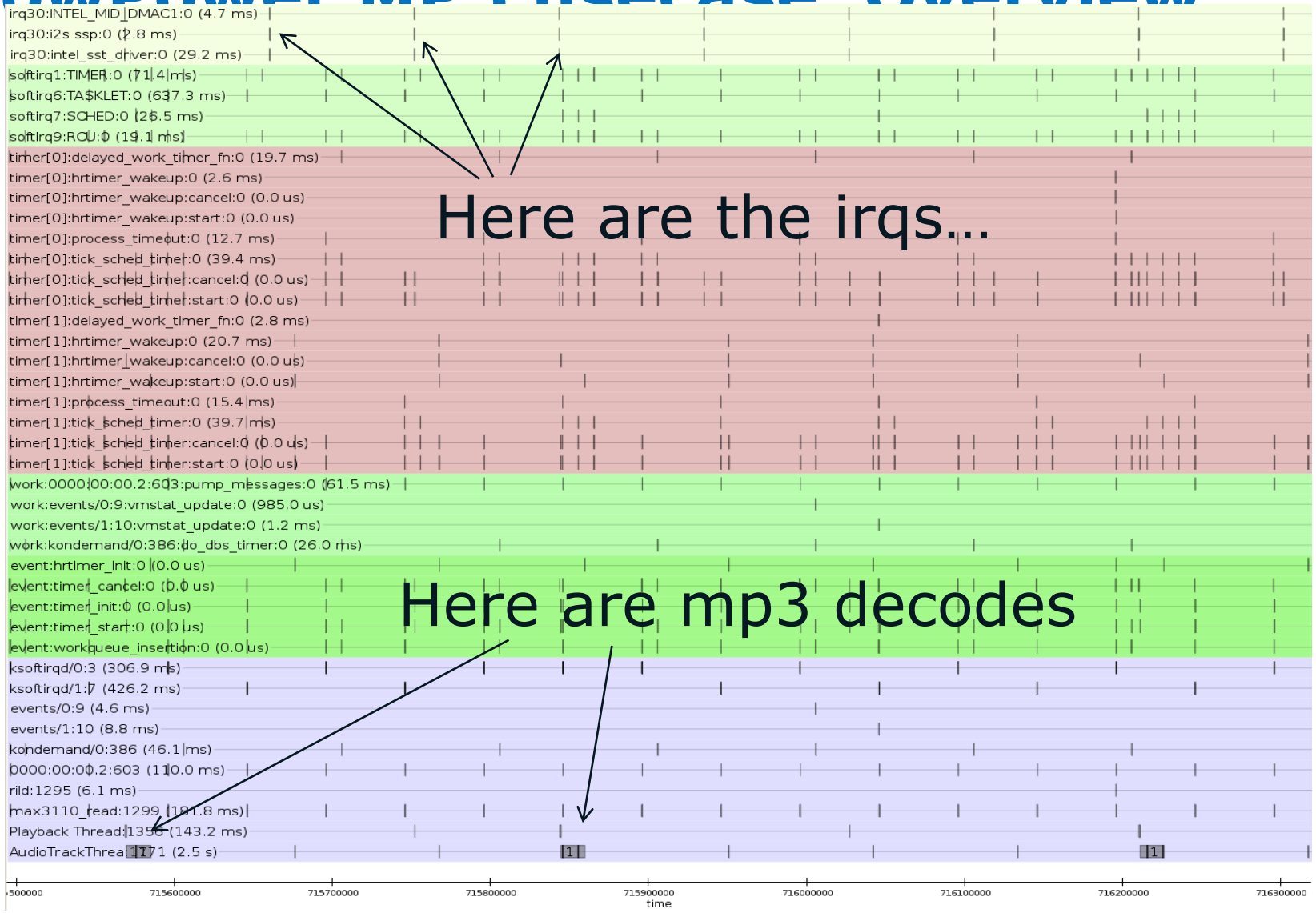
LowPower MP3 usecase



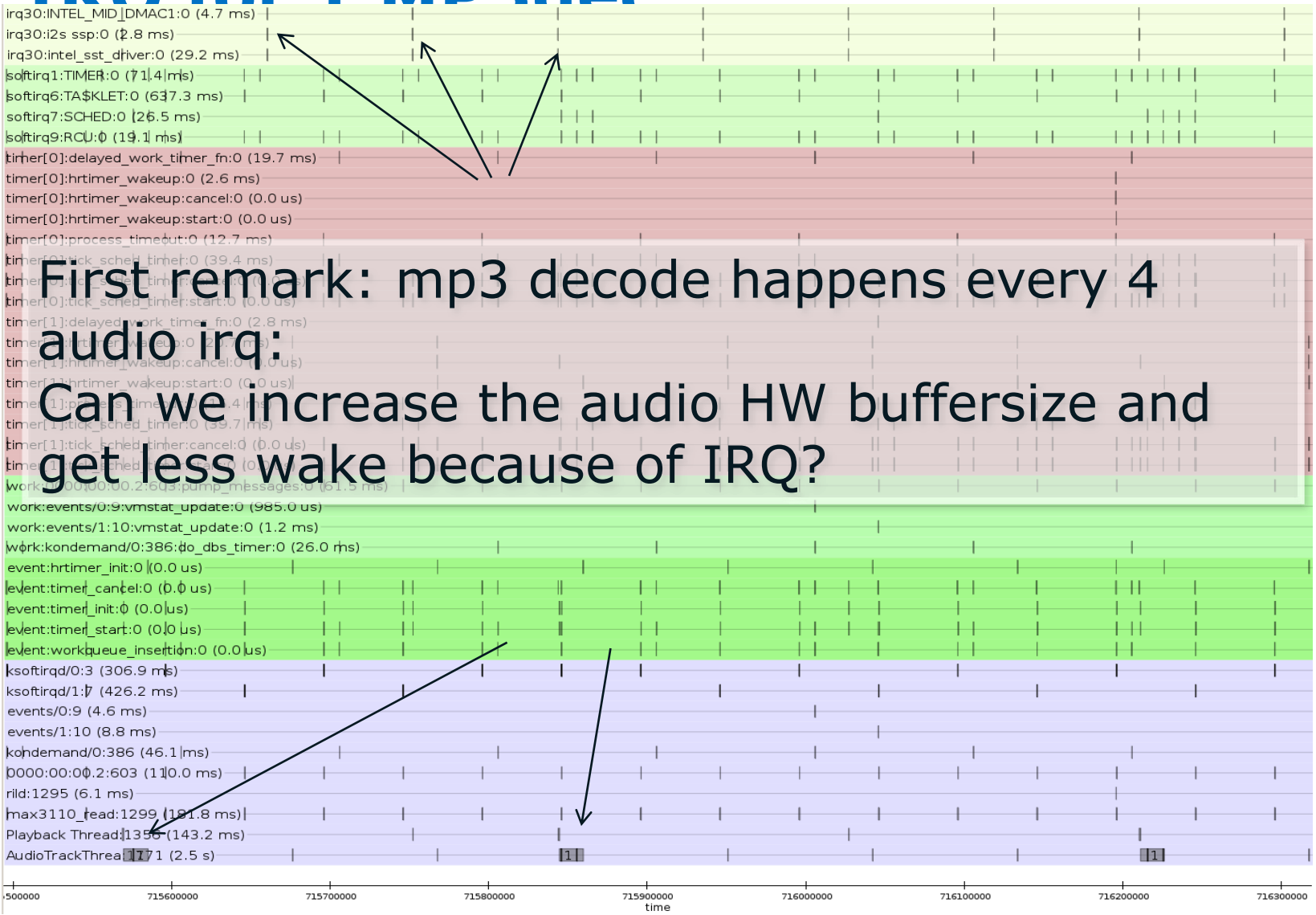
LowPower MP3 usecase Overview



LowPower MP3 usecase Overview



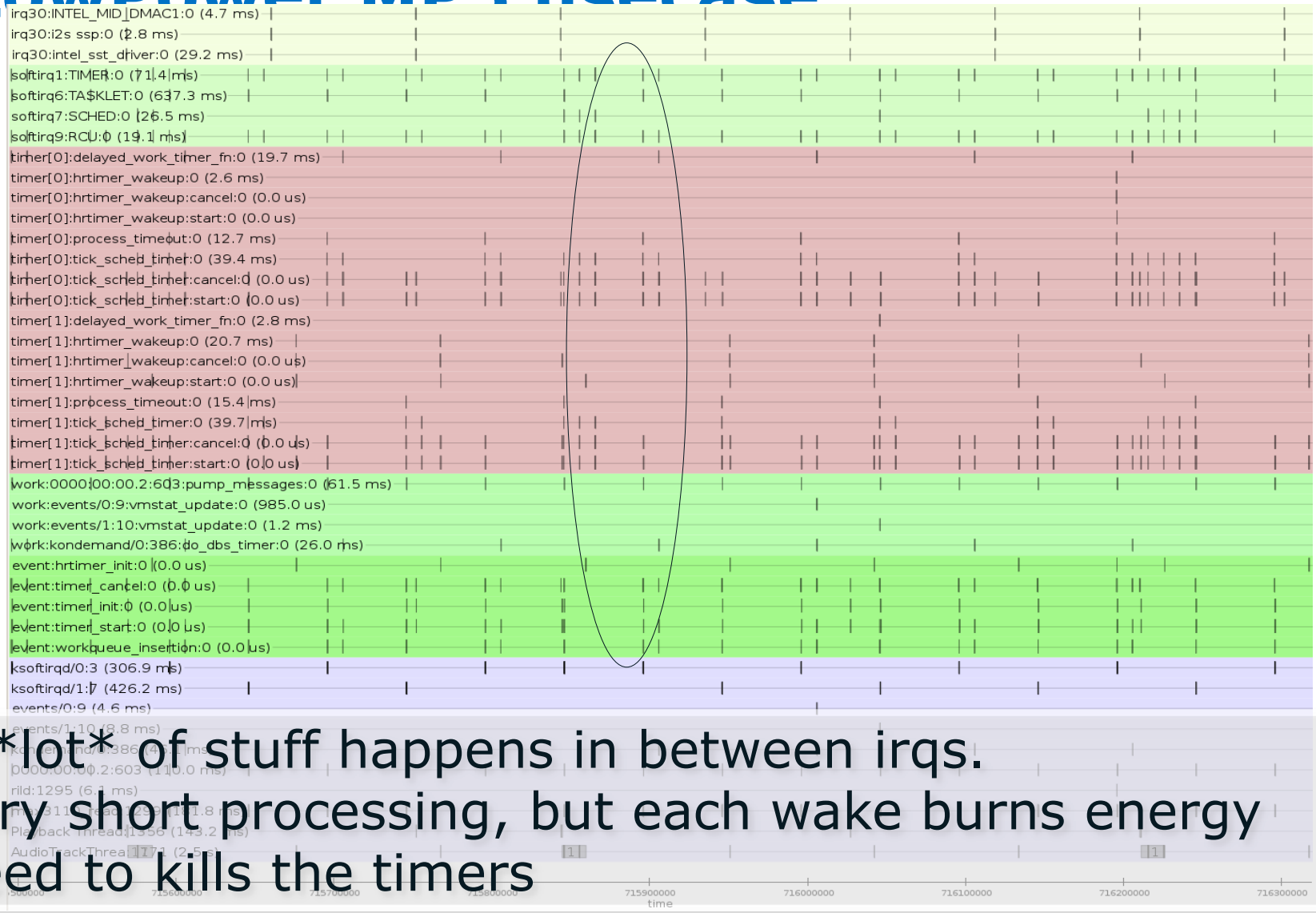
4 IRQ for 1 MP3dec



First remark: mp3 decode happens every 4 audio irq:

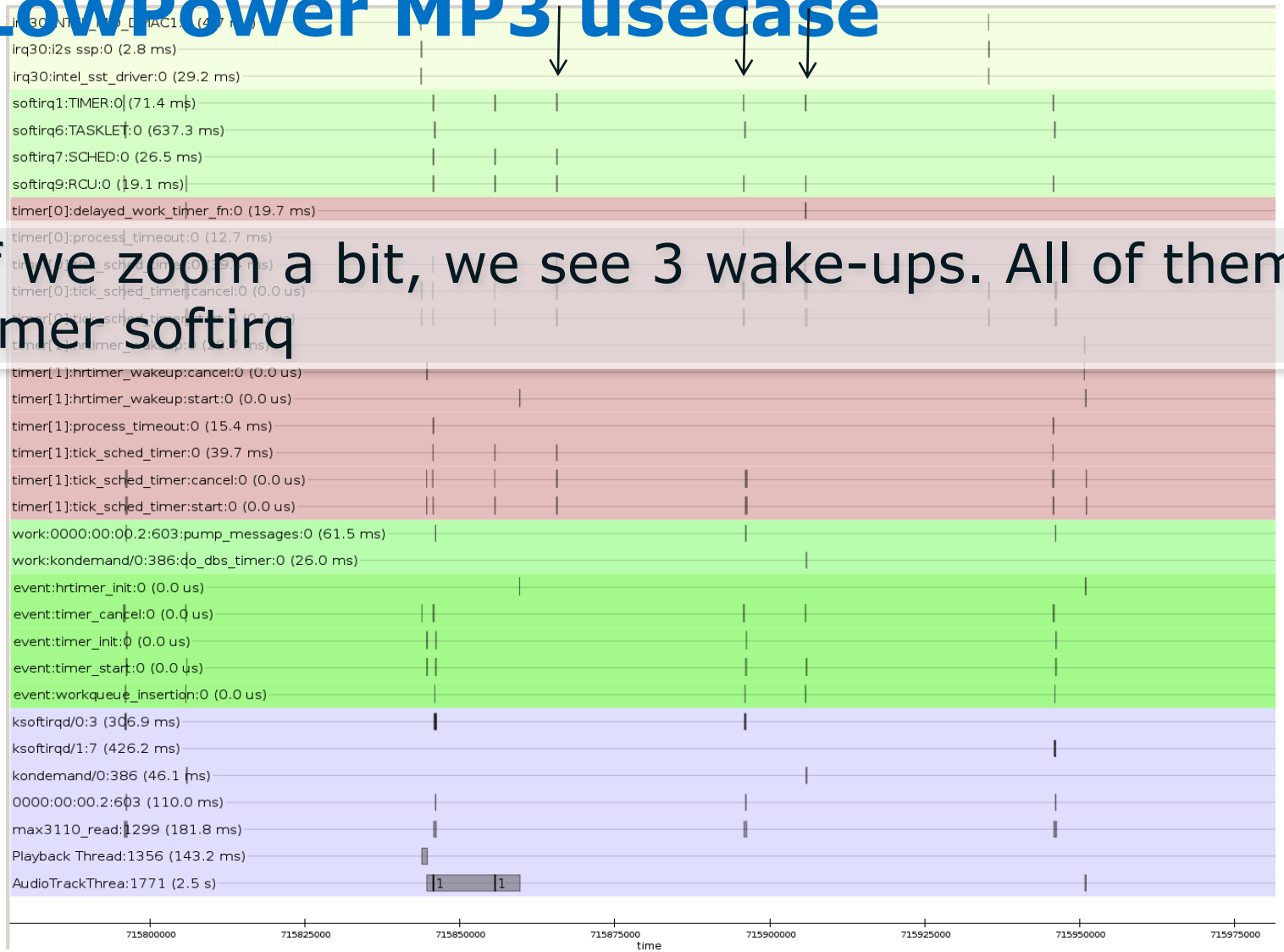
Can we increase the audio HW buffersize and get less wake because of IRQ?

LowPower MP3 usecase



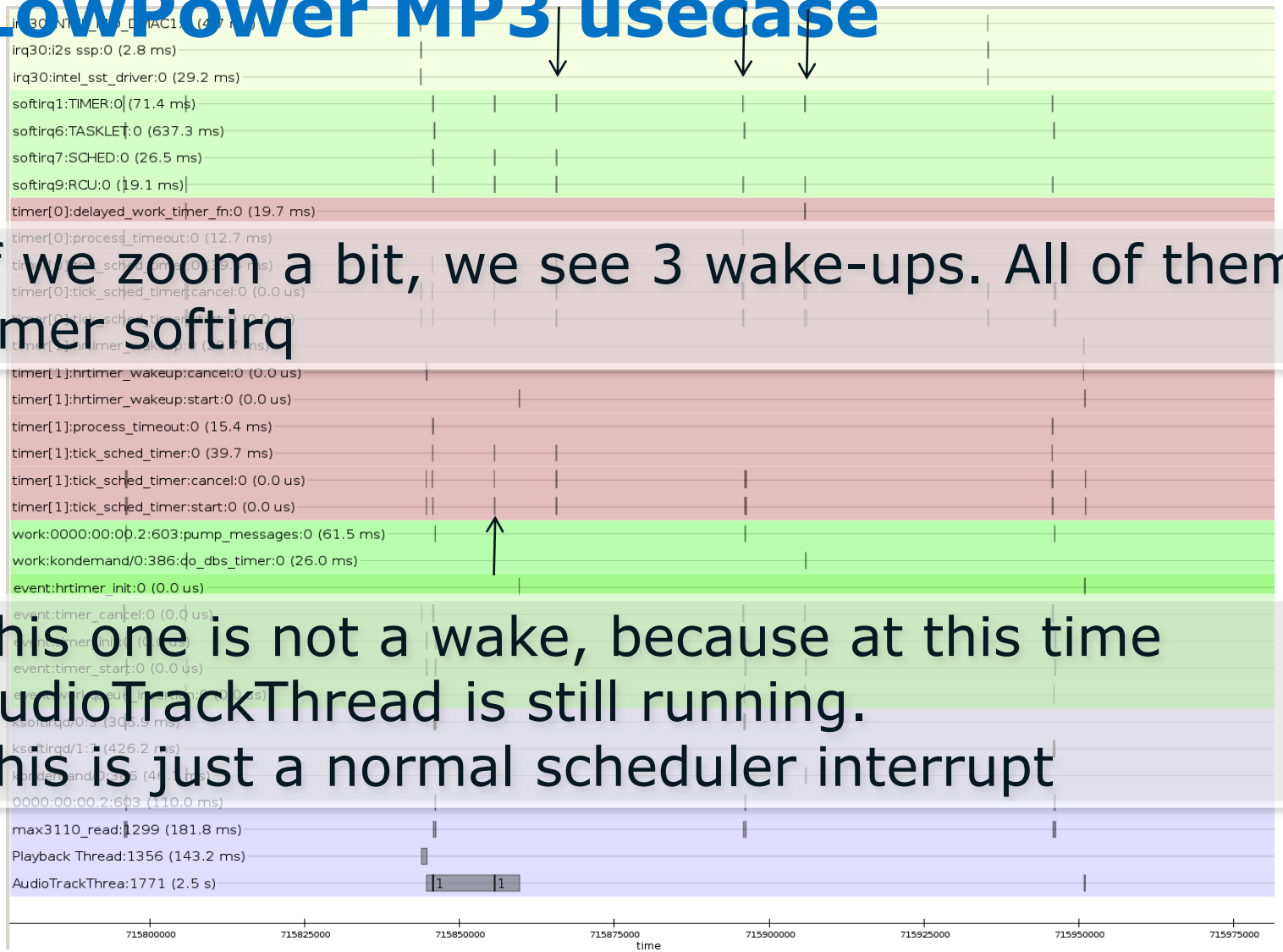
A *lot* of stuff happens in between irqs.
Very short processing, but each wake burns energy
Need to kills the timers

LowPower MP3 usecase



If we zoom a bit, we see 3 wake-ups. All of them show timer softirq

LowPower MP3 usecase

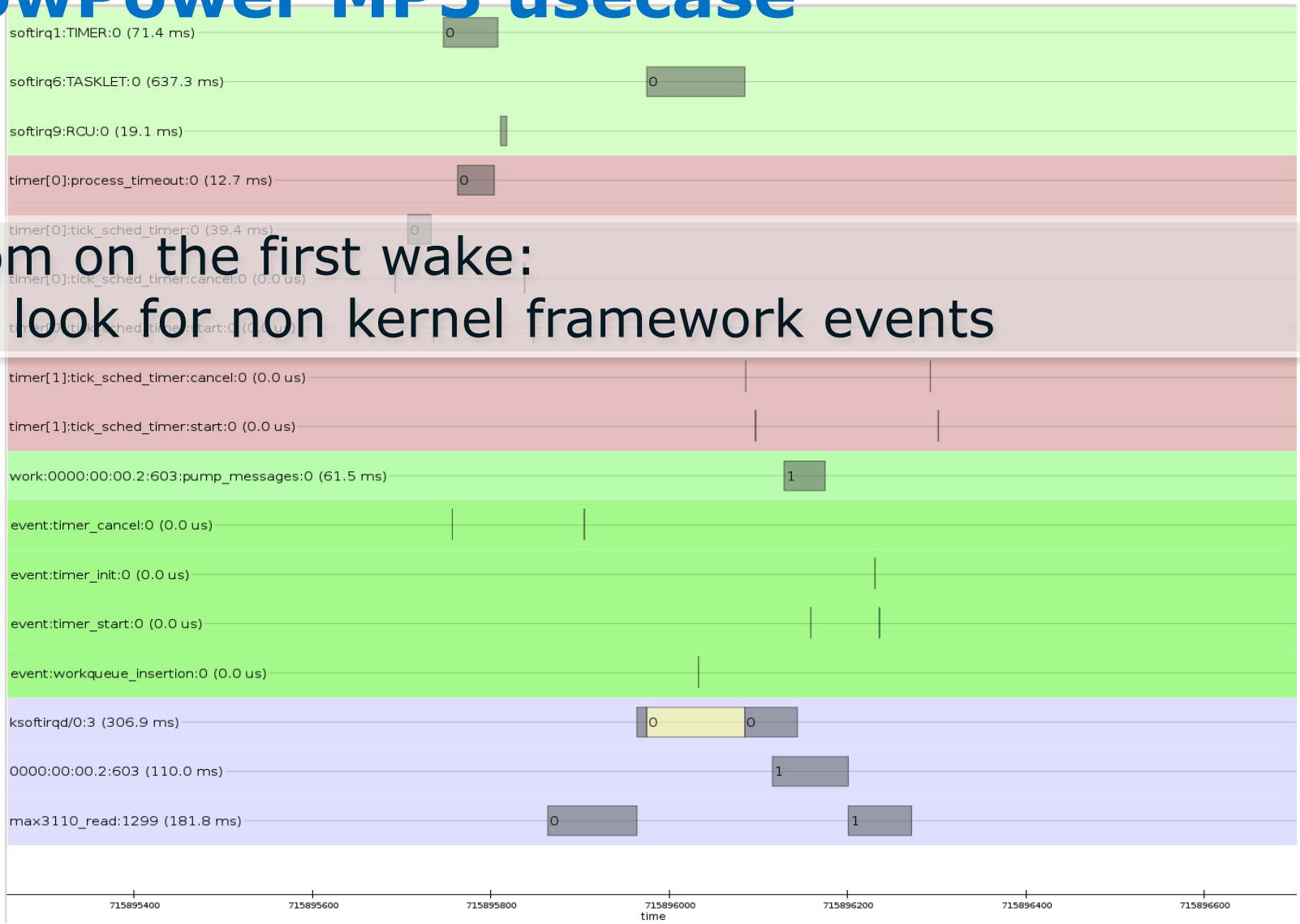


If we zoom a bit, we see 3 wake-ups. All of them show timer softirq

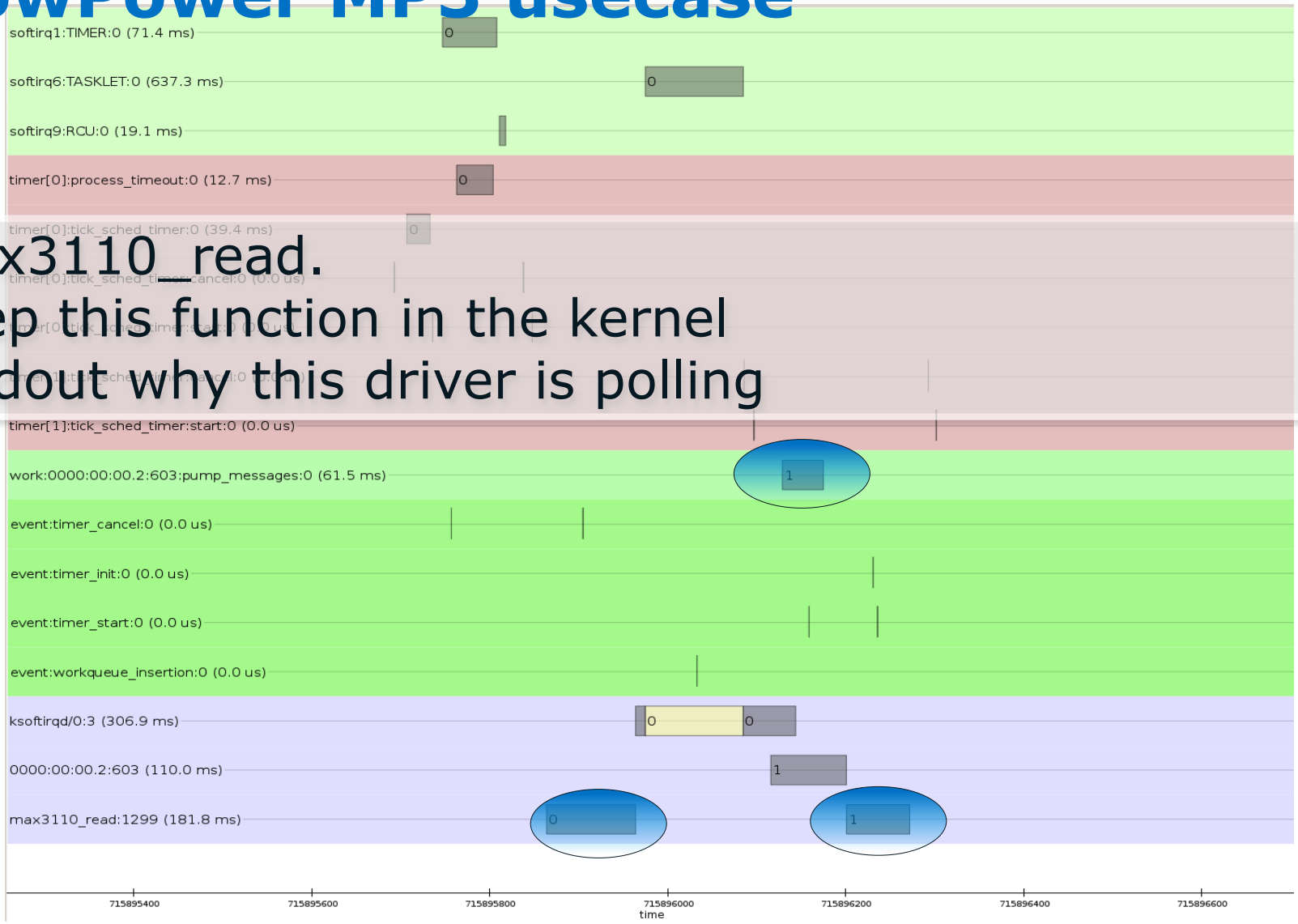
This one is not a wake, because at this time AudioTrackThread is still running.
This is just a normal scheduler interrupt

LowPower MP3 usecase

zoom on the first wake:
We look for non kernel framework events

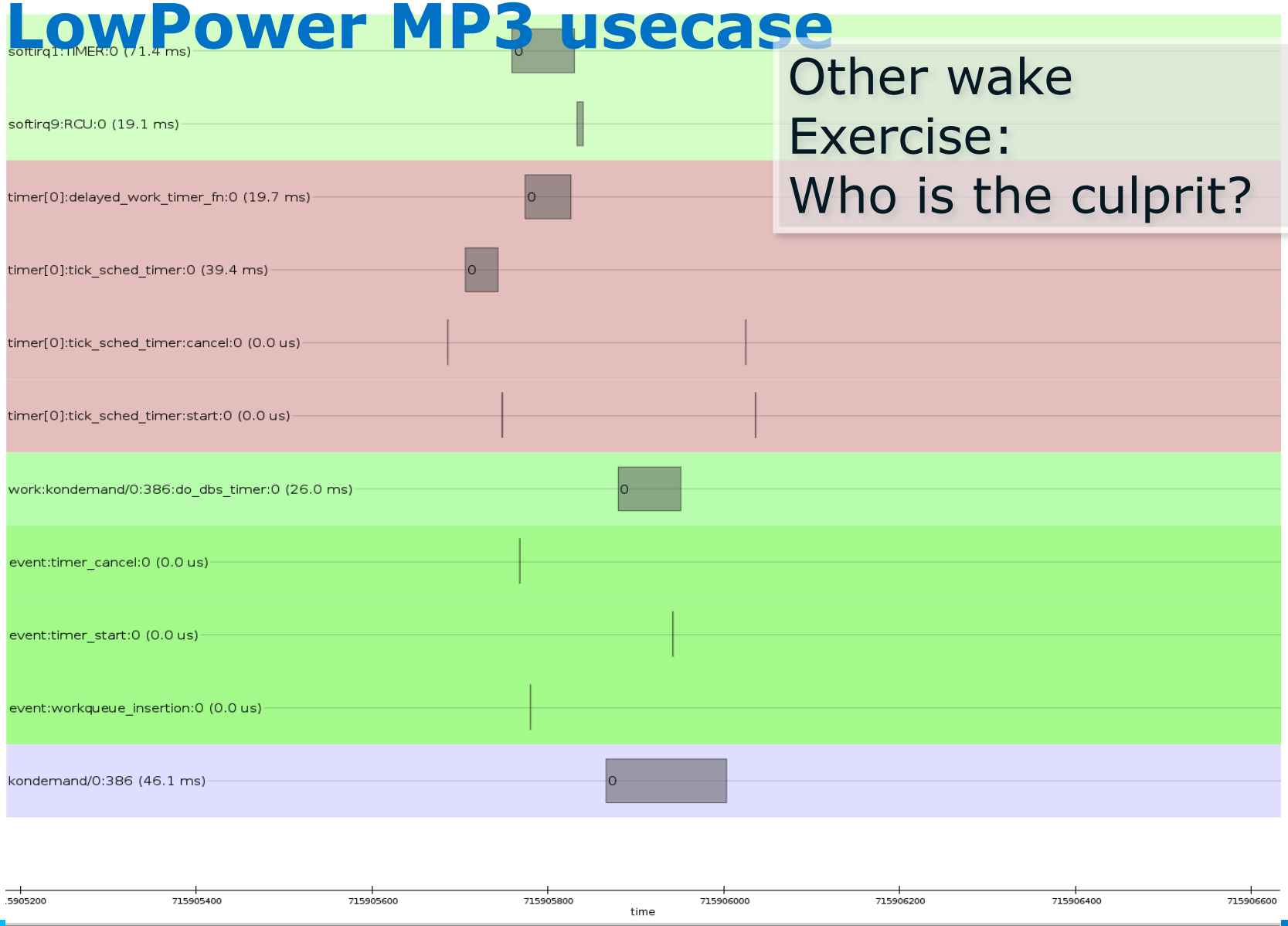


LowPower MP3 usecase



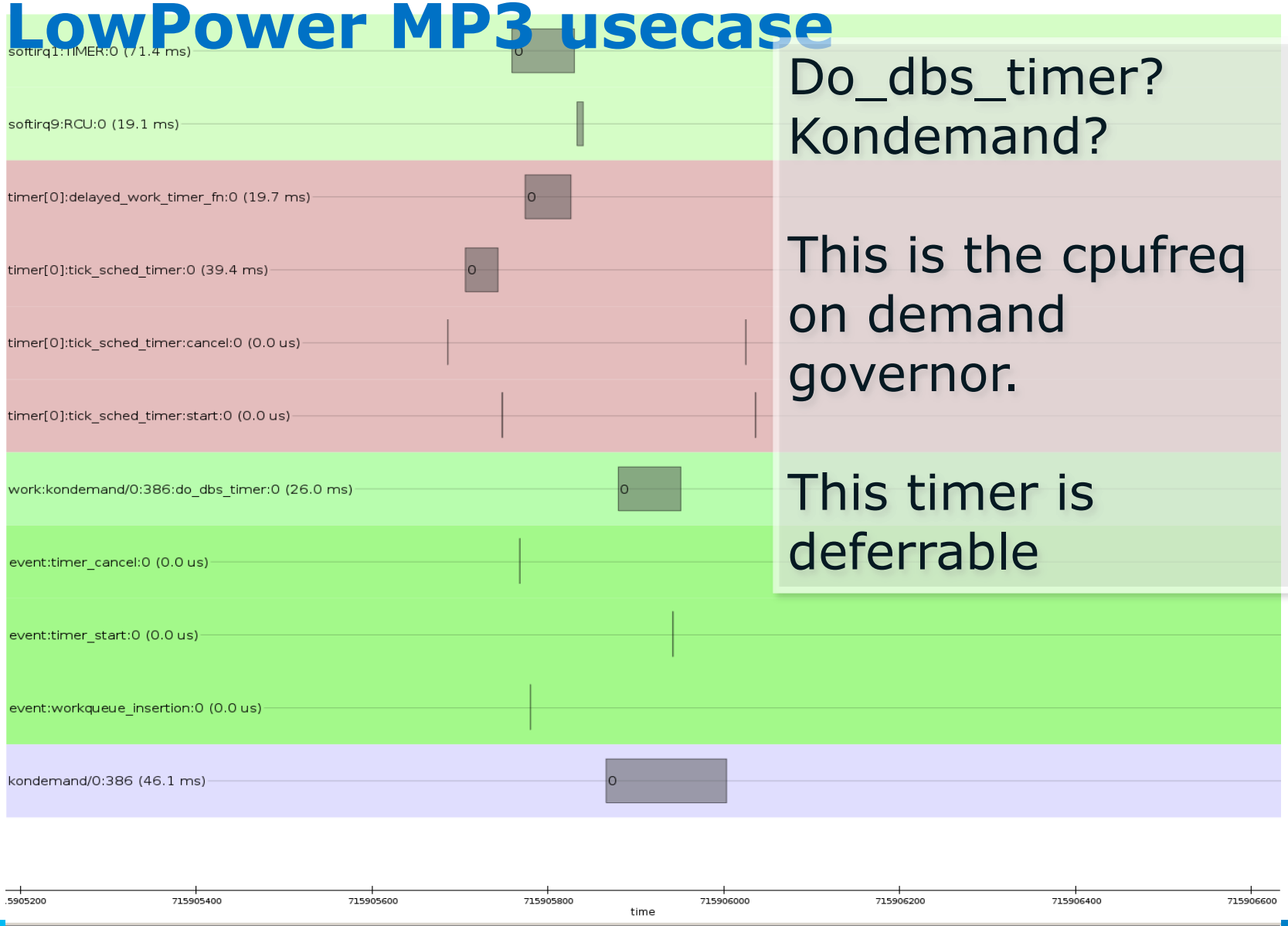
max3110_read.
Grep this function in the kernel
Findout why this driver is polling

LowPower MP3 usecase



Other wake
Exercise:
Who is the culprit?

LowPower MP3 usecase



Do_dbs_timer?
Kondemand?

This is the cpufreq on demand governor.

This timer is deferrable

http://www.lesswatts.org/projects/tickless/deferrable.php

Tickless Idle

Deferrable timers

In the Linux kernel, timers provide a way for any driver or kernel routine to register an event that needs to be handled at some point in the future.

Some timers, used inside the kernel, are important to service in a sub-second interval when the system is busy, but are not critical when the system is idle. A prime example, is the timer usage in ondemand driver, which does a periodic sampling of CPU use.

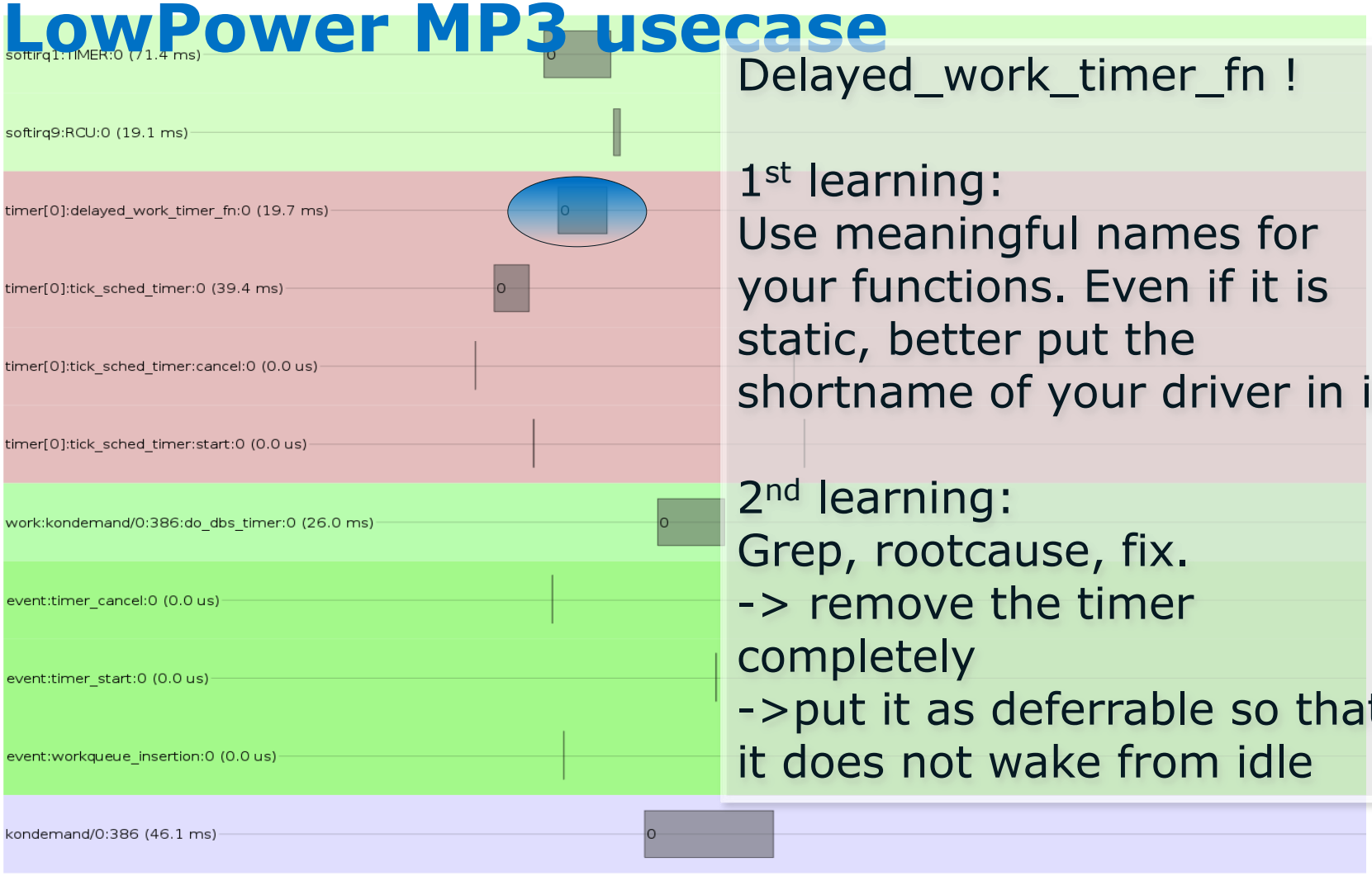
Accuracy is important for these timers when the CPU is busy because they are used to increase the frequency of a CPU to meet increased demand. But, while system is idle, ondemand governor can tolerate some delay and let the CPU remain idle for a longer time.

To effectively handle such timers, the "deferrable timer" infrastructure was introduced in the Linux kernel. These deferrable timers are handled normally when the CPU is busy. However, when the CPU is idle, deferrable timers are queued until there is a non-deferrable timer or an interrupt that wakes the CPU from idle.

Using deferrable timers significantly reduces the number of times a CPU is woken up from idle. The data in the table below is based on an Intel® Core™ 2 Duo-based test platform. The table shows the number of interrupts per second and the average time the CPU spends in idle state (in uS). Note that the kernel used during the measurement had tickless idle enabled with HZ being 1000.

	# interrupts	#events	Avg CPU idle residency (uS)
Ondemand	118	60.60	10161
Ondemand + deferrable timer	89	17.17	20312

LowPower MP3 usecase

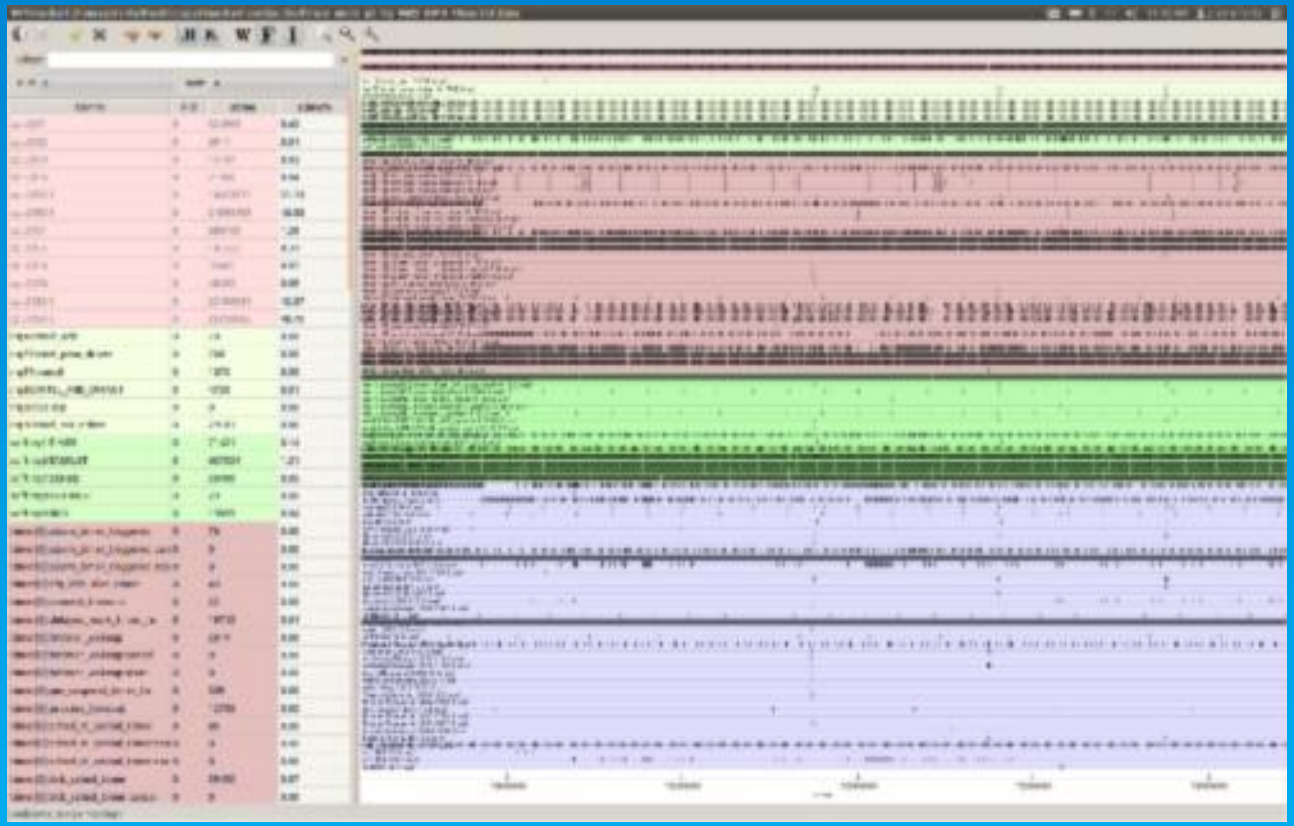


Delayed_work_timer_fn !

1st learning:
Use meaningful names for your functions. Even if it is static, better put the shortname of your driver in it.

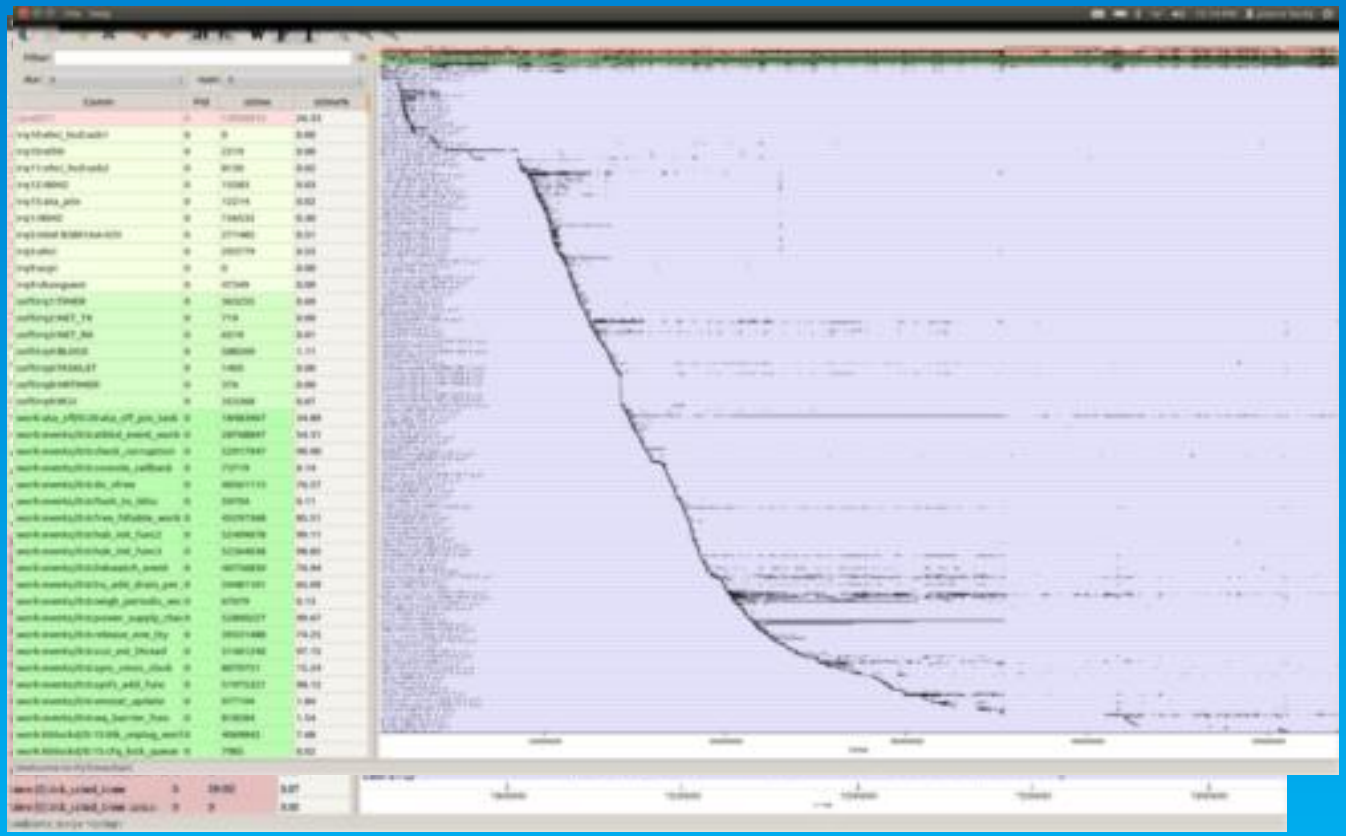
2nd learning:
Grep, rootcause, fix.
-> remove the timer completely
->put it as deferrable so that it does not wake from idle

Quick Demo!

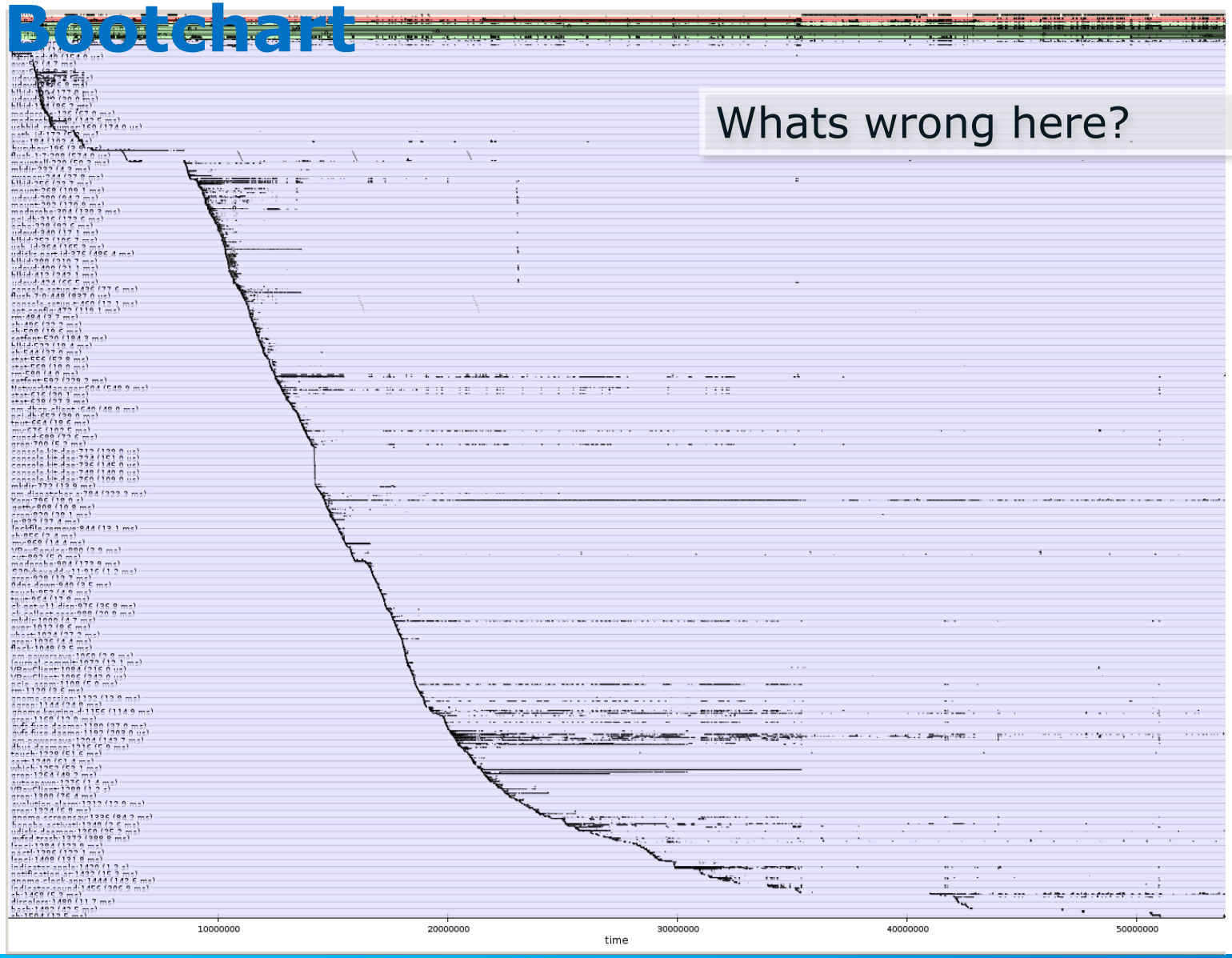


Bootcharting Ubuntu

filtering in pytimechart

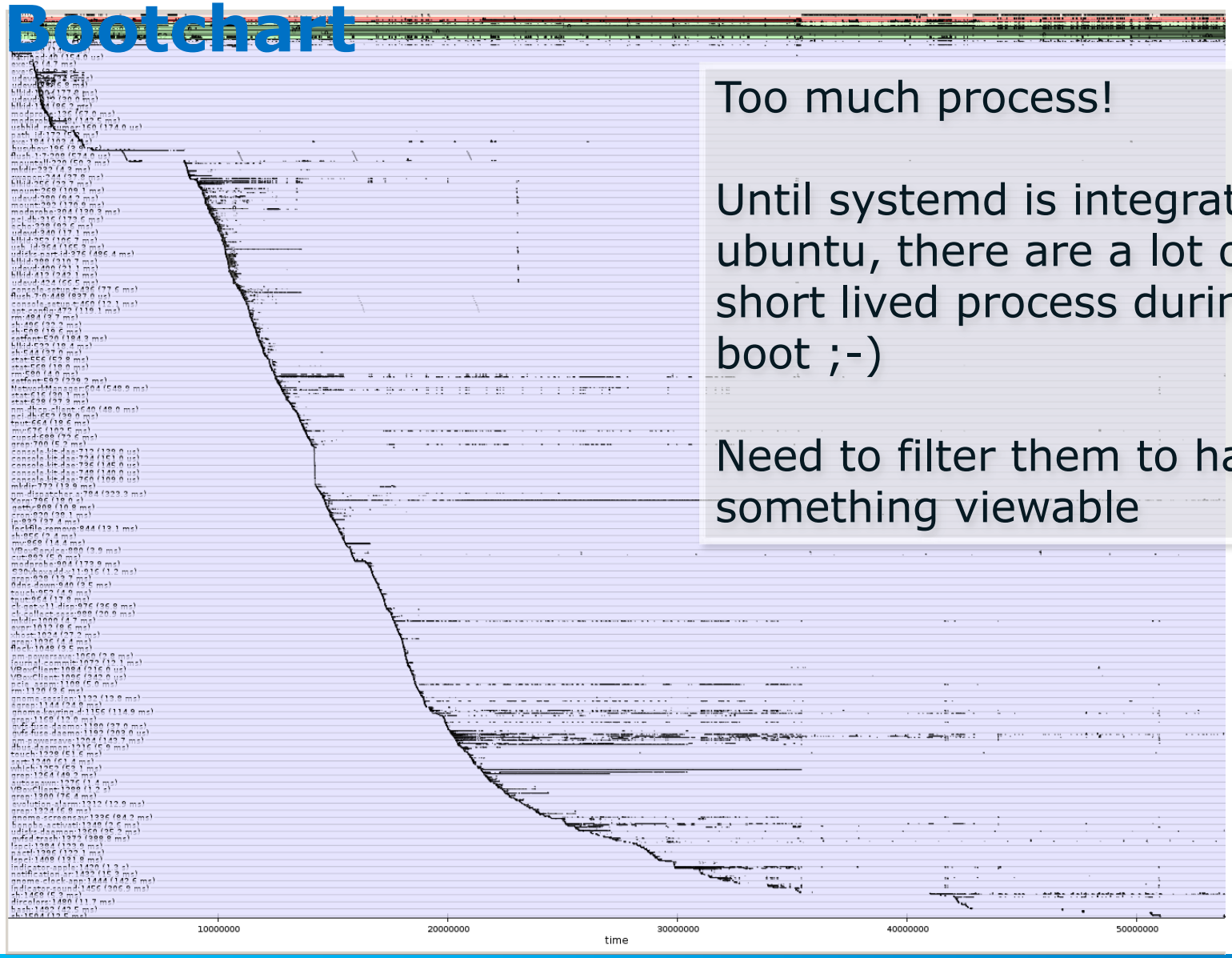


Bootchart



Whats wrong here?

Bootchart

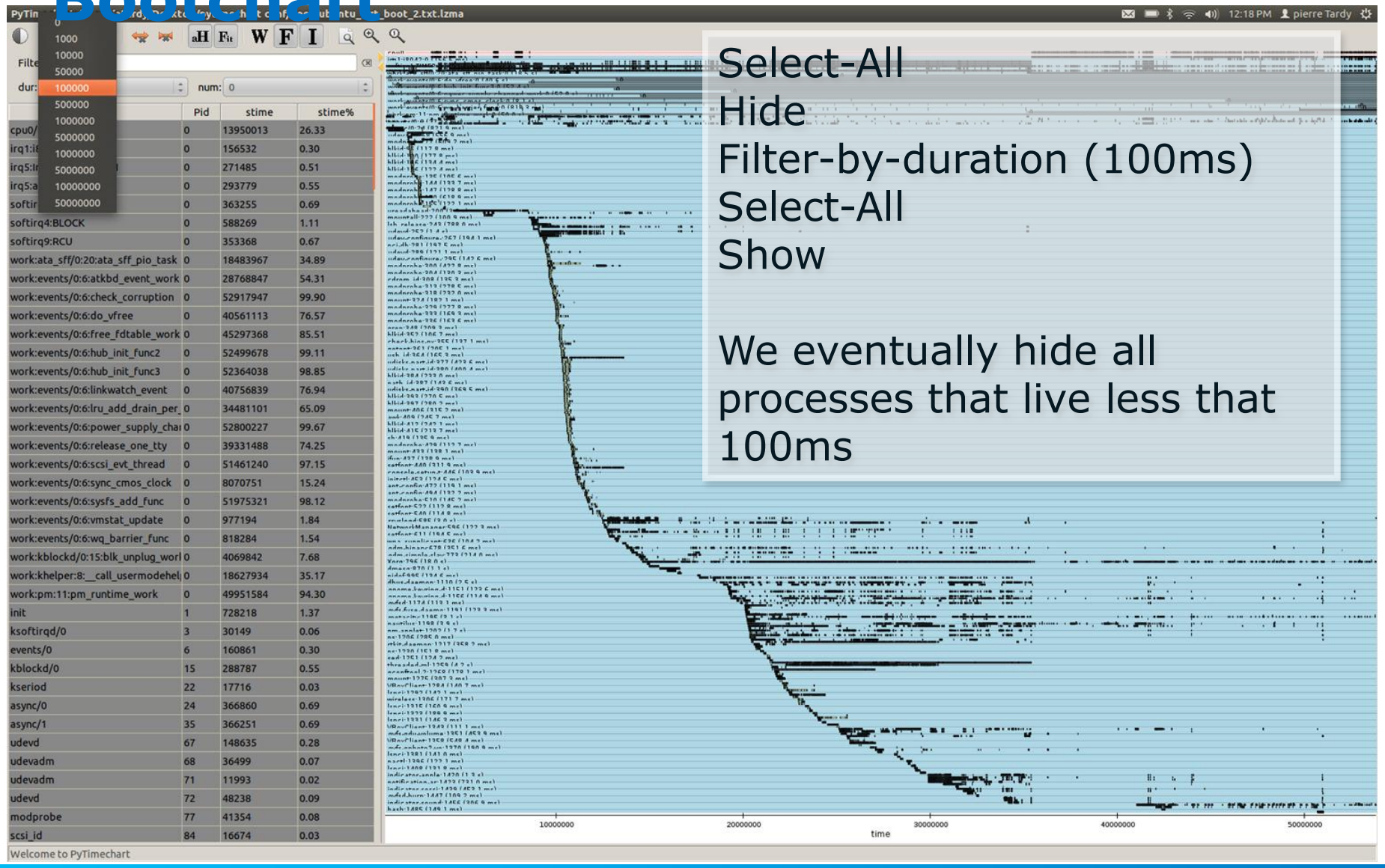


Too much process!

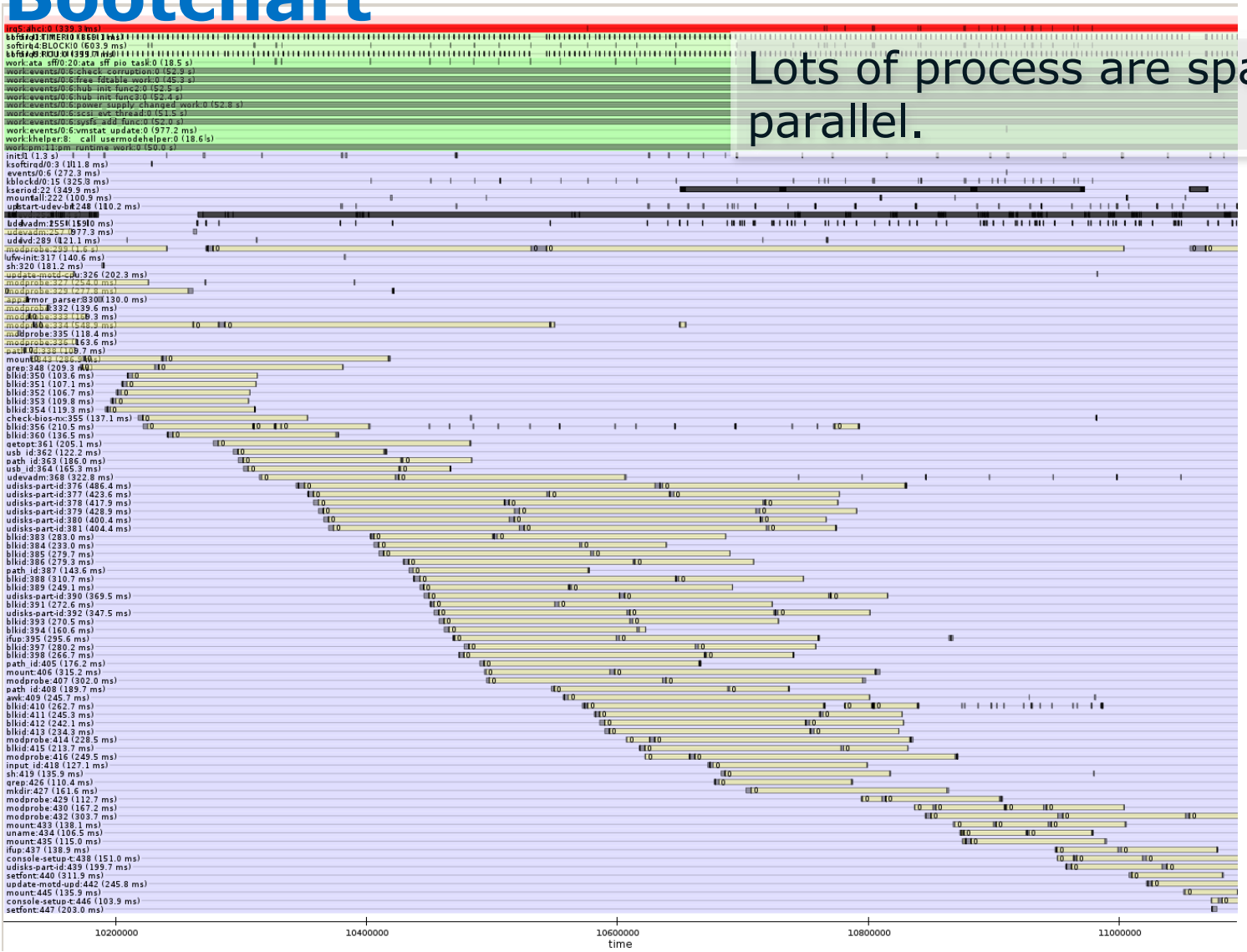
Until systemd is integrated in ubuntu, there are a lot of short lived process during boot ;-)

Need to filter them to have something viewable

Bootchart

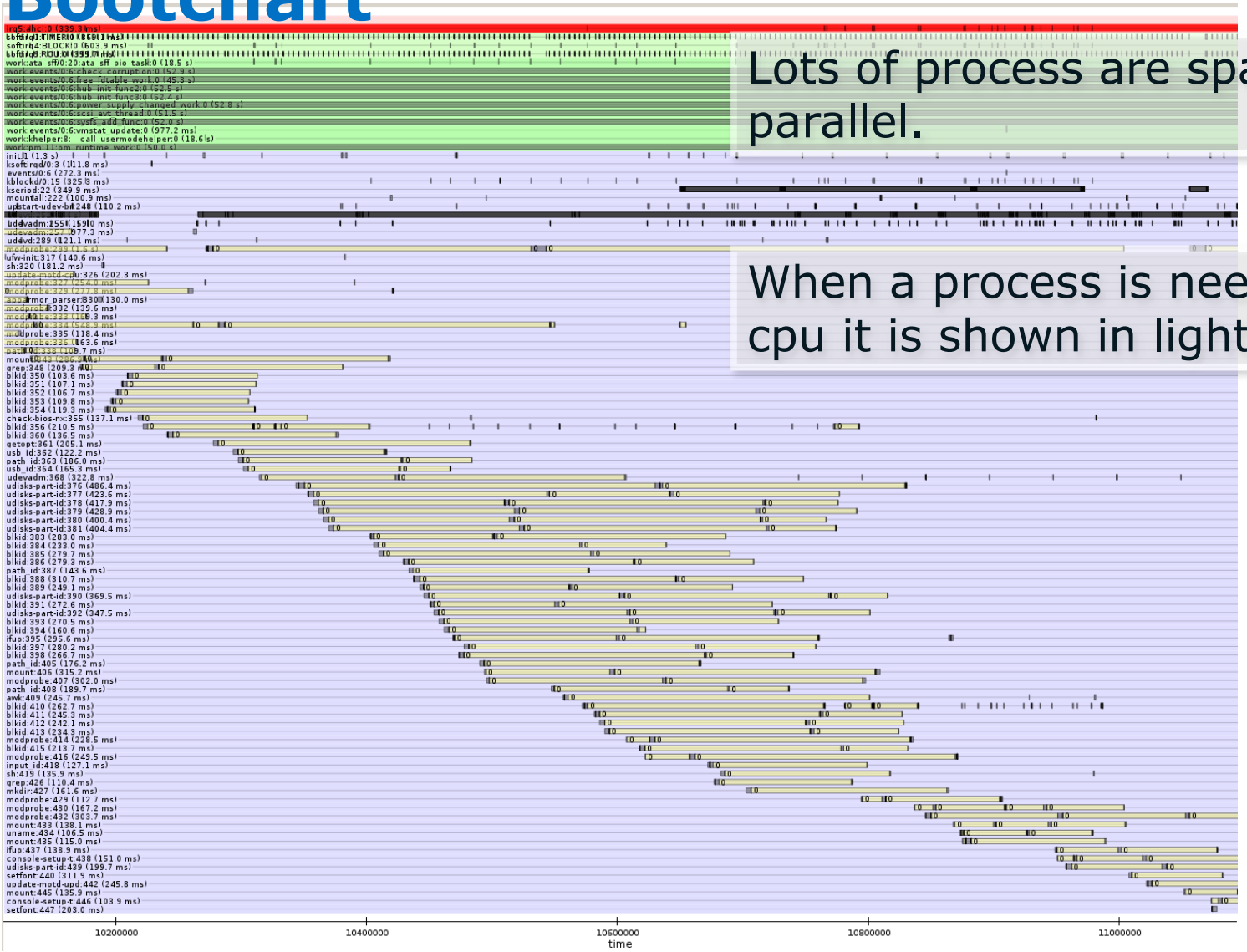

Select-All
Hide
Filter-by-duration (100ms)
Select-All
Show
We eventually hide all processes that live less that 100ms
time
Welcome to PyTimechart

Bootchart



Lots of process are spawn in parallel.

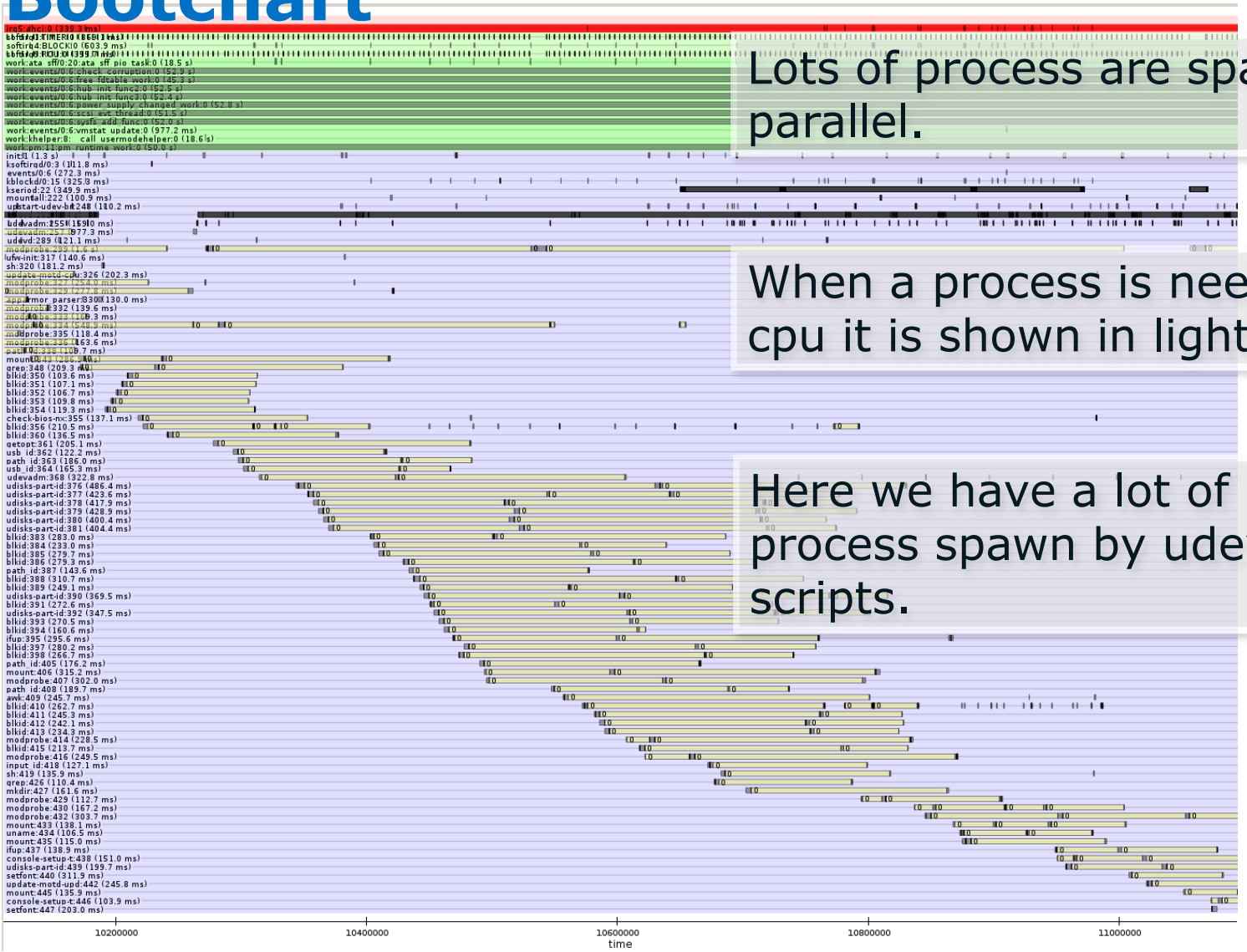
Bootchart



Lots of process are spawn in parallel.

When a process is needing cpu it is shown in light yellow.

Bootchart



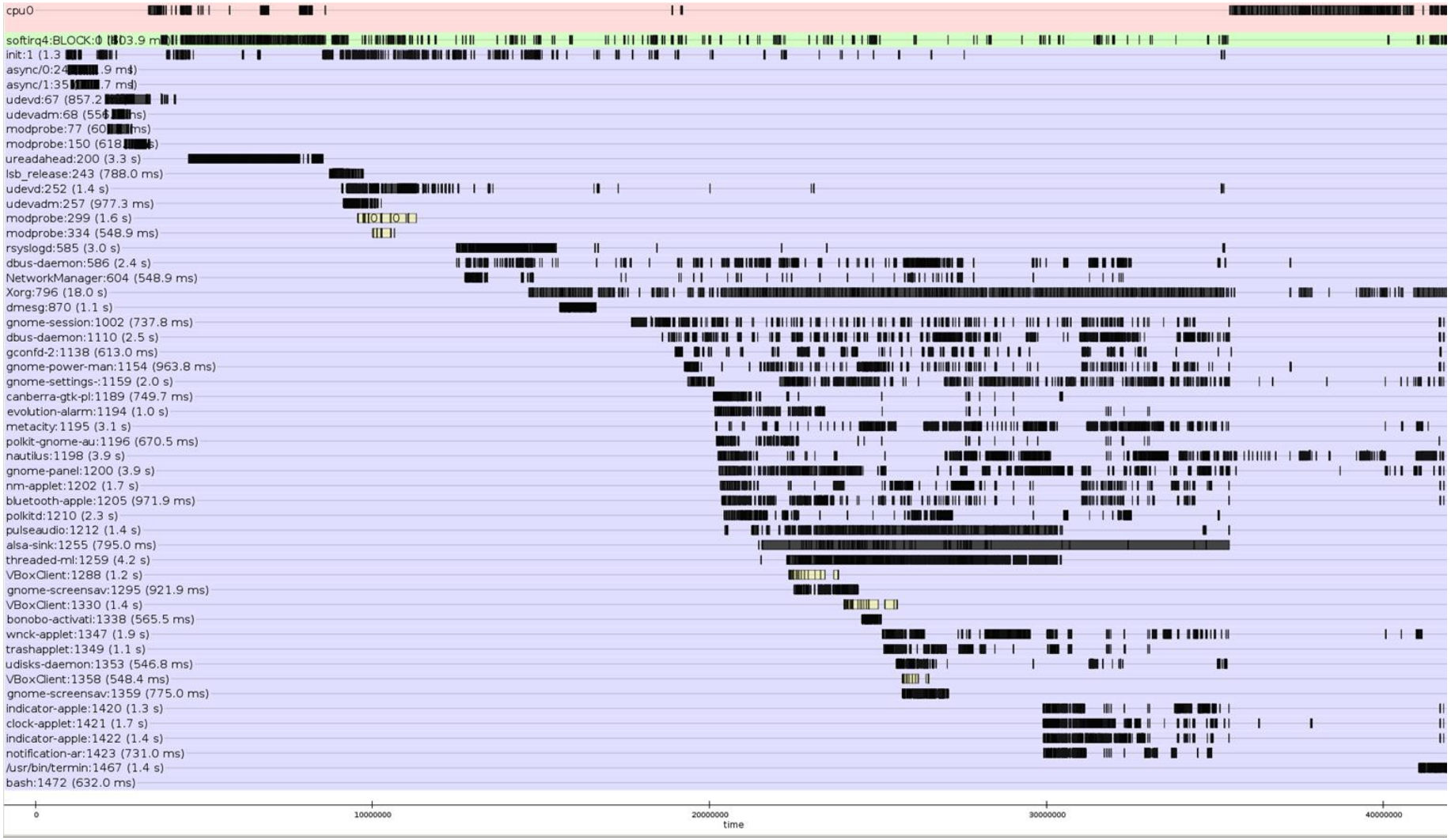
Lots of process are spawn in parallel.

When a process is needing cpu it is shown in light yellow.

Here we have a lot of blkid process spawn by udev scripts.

Filtering more process…

Bootchart



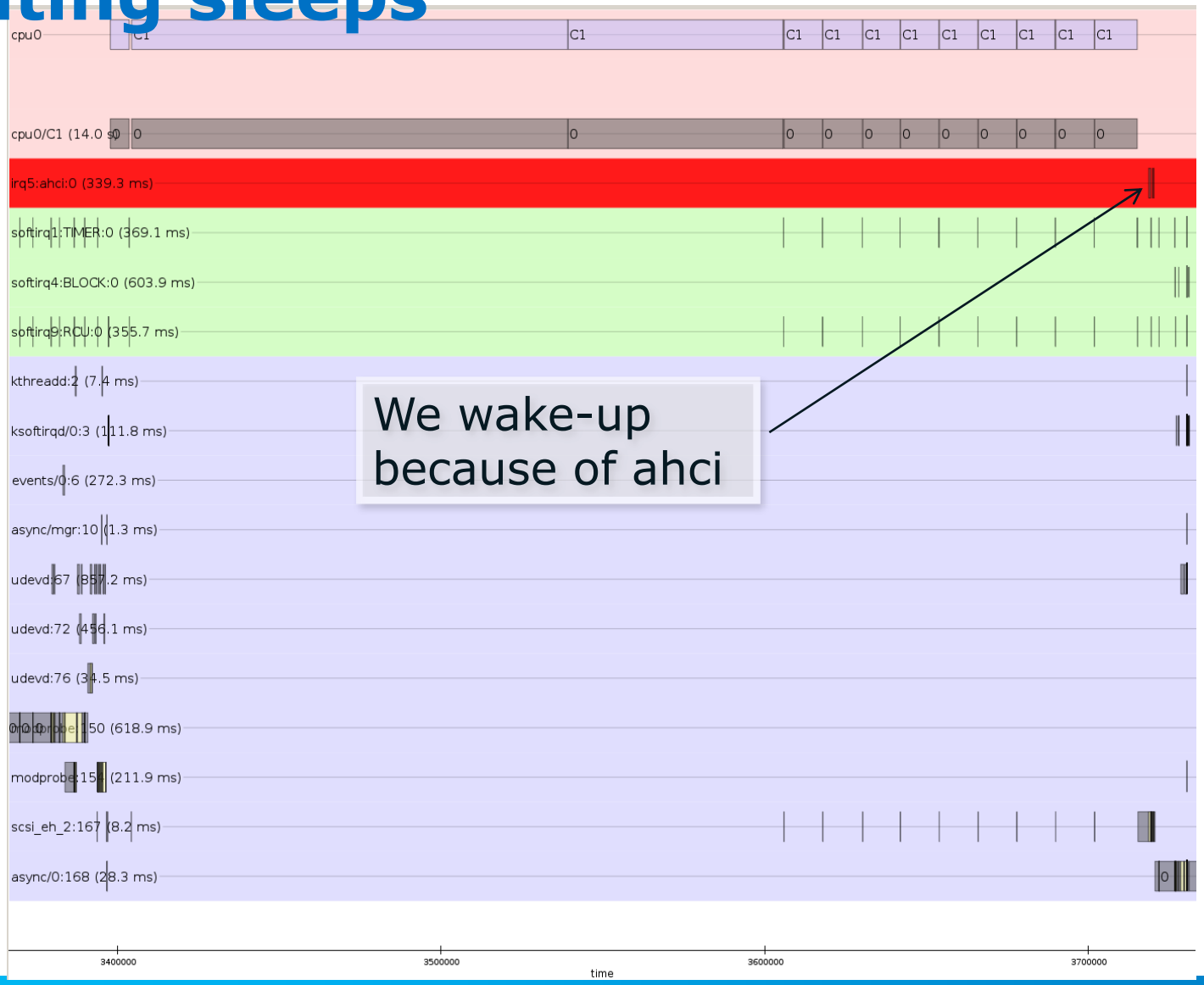
Filtering more process…

Bootchart

There are some part were we sleep during boot…

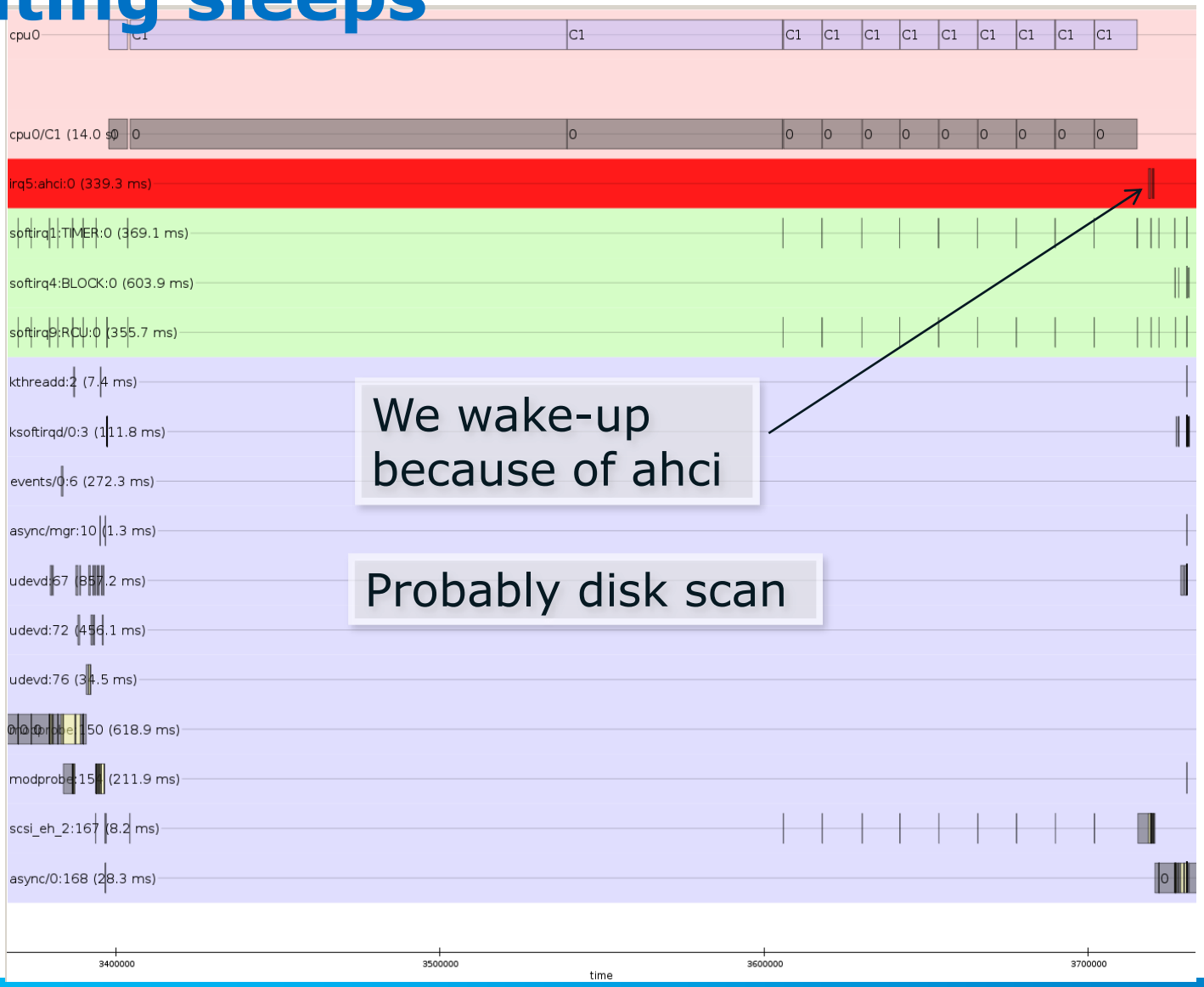
Hunting sleeps

Zoom to first sleep



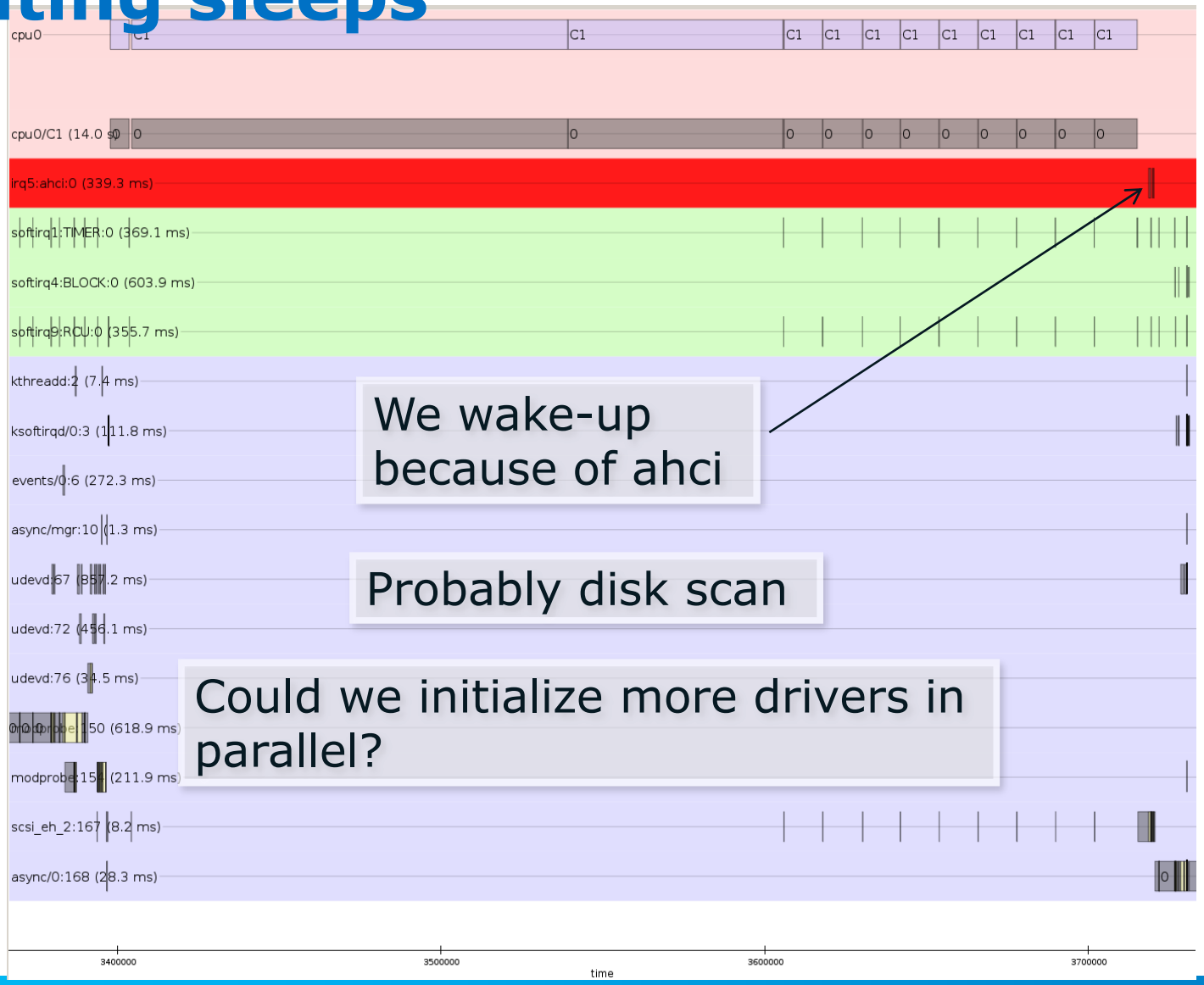
Hunting sleeps

Zoom to first sleep



Hunting sleeps

Zoom to first sleep



We wake-up because of ahci

Probably disk scan

Could we initialize more drivers in parallel?

http://lwn.net/Articles/314808/

An asynchronous function call infrastructure

By **Jonathan Corbet**
January 13, 2009

Arjan van de Ven's fast boot project will be familiar to most LWN readers by now. Most of Arjan's work has not yet found its way into the mainline, though, so most of us still have to wait for our systems to boot the slow way. That said, the 2.6.29 kernel will contain one piece of the fast boot work, in the form of the asynchronous function call infrastructure. Users will need to know where to find it, though, before making use of it.

There are many aspects to the job of making a system boot quickly. Some of the lowest-hanging fruit can be found in the area of device probing. Figuring out what hardware exists on the system tends to be a slow task at best; if it involves physical actions (such as spinning up a disk) it gets even worse. Kernel developers have long understood that they could gain a lot of time if this device probing could, at least, be done in a parallel manner: while the kernel is waiting for one device to respond, it can be talking to another. Attempts at parallelizing this work over the years have foundered, though. Problems with device ordering, concurrent access, and more have adversely affected system stability, with the inevitable result that the parallel code is taken back out. So early system initialization remains almost entirely sequential.

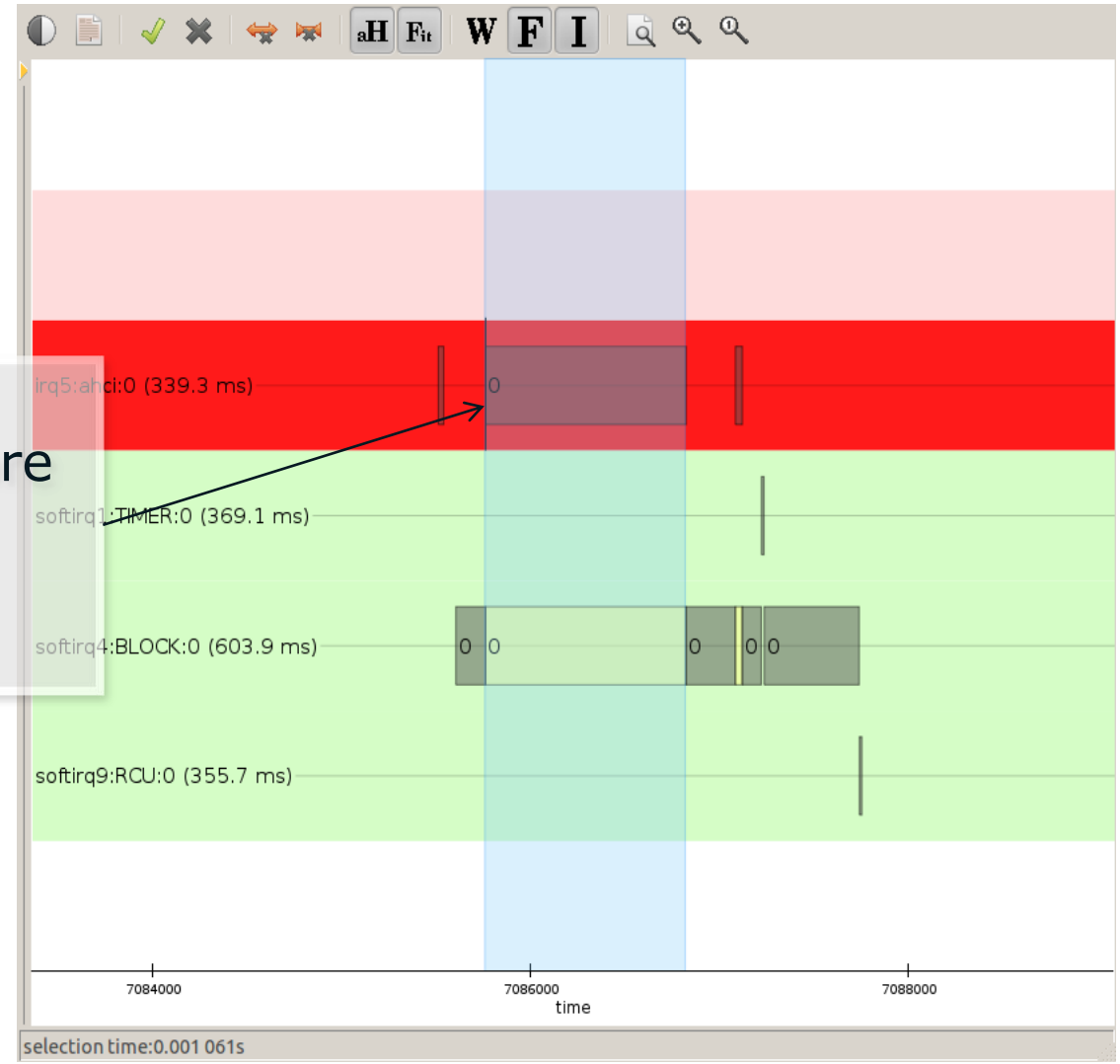
Although this is far to be new stuff, we don't see a lot of drivers using this, AFAIK.

Disgression on red irqs

What does those IRQs in red means?

Disgression on red irqs

IRQ handler that lasts more than 1ms
Bad for RT latency!



Modem driver traces

Hacking PyTimechart to decode your own traces

```
from timechart.plugin import *
from timechart import colors
from timechart.model import tcProcess

class hsi(plugin):
    additional_colors = """
hsi_bg                #80ff80
"""
    additional_ftrace_parsers = [
        ]
    additional_process_types = {
            "hsi":(tcProcess, MISC_TRACES_CLASS),
        }
    @staticmethod
    def do_function_ffl_start_rx(proj,event):
        process = proj.generic_find_process(0,"hsiffl","hsi")
        caller = proj.generic_find_process(0,event.caller,"hsi")
        proj.generic_process_start(process,event,build_p_stack=False)

        proj.generic_process_single_event(caller,event)
        proj.generic_add_wake(caller, process,event)

    @staticmethod
    def do_function_ffl_stop_rx(proj,event):
        process = proj.generic_find_process(0,"hsiffl","hsi")
        proj.generic_process_end(process,event,build_p_stack=False)
    @staticmethod
    def do_function_hsi_start_rx(proj,event):
        process = proj.generic_find_process(0,"hsi","hsi")
        caller = proj.generic_find_process(0,event.caller,"hsi")
        pidcaller = proj.generic_find_process(event.common_pid,event.common_comm,"hsi")
        proj.generic_process_start(process,event,build_p_stack=False)

        proj.generic_process_single_event(caller,event)
        proj.generic_add_wake(caller, process,event)

    @staticmethod
    def do_function_hsi_stop_rx(proj,event):
        process = proj.generic_find_process(0,"hsi","hsi")
        proj.generic_process_end(process,event,build_p_stack=False)

plugin_register(hsi)
```

Hacking PyTimechart to decode your own traces

Pytimechart is written in python

- Easy to decode its own tracepoints
- Even if you don't add tracepoints, you can take advantage of function tracing to make more sense of your traces

The trace I took for this is a simple "receive SMS" trace

- I want to see when the hsi driver is in receive mode
- First, I trace hsi* and ffl* which are the low level, and protocol driver of our modem interface
- I can see 4 interesting functions in the trace:
  - hsi_start_rx()
  - hsi_stop_rx()
  - ffl_start_rx()
  - ffl_stop_rx()

timechart/plugins/template.py

Provided for convenience as a good starting point.

See doc for more detailed info

```
from timechart.plugin import *
from timechart import colors
from timechart.model import tcProcess

# to use with start_spi.sh
class template(plugin):
    additional_colors = """
template_bg                  #80ff80
"""
    additional_ftrace_parsers = [
        ]
    additional_process_types = {
            "template":(tcProcess, MISC_TRACES_CLASS),
        }
    @staticmethod
    def do_function_my_start_function(proj,event):
        """This method will be called when the function "my_start_function" appears in the trace
        in this example, we start a process, and mark its caller as waked it
        """
        process = proj.generic_find_process(0,"template","template")
        proj.generic_process_start(process,event,build_p_stack=False)

        proj.generic_process_single_event(caller,event)

        # the function caller
        caller = proj.generic_find_process(0,event.caller,"template")
        proj.generic_add_wake(caller, process,event)

        # the calling process
        pidcaller = proj.generic_find_process(event.common_pid,event.common_comm,"hsi")
        proj.generic_add_wake(pidcaller, process,event)

    @staticmethod
    def do_function_my_stop_function(proj,event):
        """This method will be called when the function "my_stop_function" appears in the trace
        in this example, we stop the "template" process
        """
        process = proj.generic_find_process(0,"template","template")
        proj.generic_process_end(prev,event,build_p_stack=False)

# this plugin is disabled... uncomment to enable it.
#plugin_register(template)
```

timechart/plugins/hsi.py

```
sed s/template/hsi/g template.py > hsi.py
```

```
from timechart.plugin import *
from timechart import colors
from timechart.model import tcProcess

class hsi(plugin):
    additional_colors = """
hsi_bg                #80ff80
"""
    additional_ftrace_parsers = [
        ]
    additional_process_types = {
            "hsi":(tcProcess, MISC_TRACES_CLASS),
        }
    @staticmethod
    def do_function_ffl_start_rx(proj,event):
        process = proj.generic_find_process(0,"hsiffl","hsi")
        caller = proj.generic_find_process(0,event.caller,"hsi")
        proj.generic_process_start(process,event,build_p_stack=False)

        proj.generic_process_single_event(caller,event)
        proj.generic_add_wake(caller, process,event)

    @staticmethod
    def do_function_ffl_stop_rx(proj,event):
        process = proj.generic_find_process(0,"hsiffl","hsi")
        proj.generic_process_end(process,event,build_p_stack=False)
    @staticmethod
    def do_function_hsi_start_rx(proj,event):
        process = proj.generic_find_process(0,"hsi","hsi")
        caller = proj.generic_find_process(0,event.caller,"hsi")
        pidcaller = proj.generic_find_process(event.common_pid,event.common_comm,"hsi")
        proj.generic_process_start(process,event,build_p_stack=False)

        proj.generic_process_single_event(caller,event)
        proj.generic_add_wake(caller, process,event)

    @staticmethod
    def do_function_hsi_stop_rx(proj,event):
        process = proj.generic_find_process(0,"hsi","hsi")
        proj.generic_process_end(process,event,build_p_stack=False)

plugin_register(hsi)
```

timechart/plugins/hsi.py

sed s/template/hsi/g template.py > hsi.py

I want that the start functions begins a process event

```
from timechart.plugin import *
from timechart import colors
from timechart.model import tcProcess

class hsi(plugin):
    additional_colors = """
hsi_bg                #80ff80
"""
    additional_ftrace_parsers = [
        ]
    additional_process_types = {
            "hsi":(tcProcess, MISC_TRACES_CLASS),
        }
    @staticmethod
    def do_function_ffl_start_rx(proj,event):
        process = proj.generic_find_process(0,"hsiffl","hsi")
        caller = proj.generic_find_process(0,event.caller,"hsi")
        proj.generic_process_start(process,event,build_p_stack=False)

        proj.generic_process_single_event(caller,event)
        proj.generic_add_wake(caller, process,event)

    @staticmethod
    def do_function_ffl_stop_rx(proj,event):
        process = proj.generic_find_process(0,"hsiffl","hsi")
        proj.generic_process_end(process,event,build_p_stack=False)
    @staticmethod
    def do_function_hsi_start_rx(proj,event):
        process = proj.generic_find_process(0,"hsi","hsi")
        caller = proj.generic_find_process(0,event.caller,"hsi")
        pidcaller = proj.generic_find_process(event.common_pid,event.common_comm,"hsi")
        proj.generic_process_start(process,event,build_p_stack=False)

        proj.generic_process_single_event(caller,event)
        proj.generic_add_wake(caller, process,event)

    @staticmethod
    def do_function_hsi_stop_rx(proj,event):
        process = proj.generic_find_process(0,"hsi","hsi")
        proj.generic_process_end(process,event,build_p_stack=False)

plugin_register(hsi)
```

timechart/plugins/hsi.py

sed s/template/hsi/g template.py > hsi.py

I want that the start functions begins a process event

and the stop function ends it

```
from timechart.plugin import *
from timechart import colors
from timechart.model import tcProcess

class hsi(plugin):
    additional_colors = """
hsi_bg                #80ff80
"""
    additional_ftrace_parsers = [
        ]
    additional_process_types = {
            "hsi":(tcProcess, MISC_TRACES_CLASS),
        }
    @staticmethod
    def do_function_ffl_start_rx(proj,event):
        process = proj.generic_find_process(0,"hsiffl","hsi")
        caller = proj.generic_find_process(0,event.caller,"hsi")
        proj.generic_process_start(process,event,build_p_stack=False)

        proj.generic_process_single_event(caller,event)
        proj.generic_add_wake(caller, process,event)

    @staticmethod
    def do_function_ffl_stop_rx(proj,event):
        process = proj.generic_find_process(0,"hsiffl","hsi")
        proj.generic_process_end(process,event,build_p_stack=False)
    @staticmethod
    def do_function_hsi_start_rx(proj,event):
        process = proj.generic_find_process(0,"hsi","hsi")
        caller = proj.generic_find_process(0,event.caller,"hsi")
        pidcaller = proj.generic_find_process(event.common_pid,event.common_comm,"hsi")
        proj.generic_process_start(process,event,build_p_stack=False)

        proj.generic_process_single_event(caller,event)
        proj.generic_add_wake(caller, process,event)

    @staticmethod
    def do_function_hsi_stop_rx(proj,event):
        process = proj.generic_find_process(0,"hsi","hsi")
        proj.generic_process_end(process,event,build_p_stack=False)

plugin_register(hsi)
```

timechart/plugins/hsi.py

sed s/template/hsi/g template.py > hsi.py

I want that the start functions begins a process event

and the stop function ends it

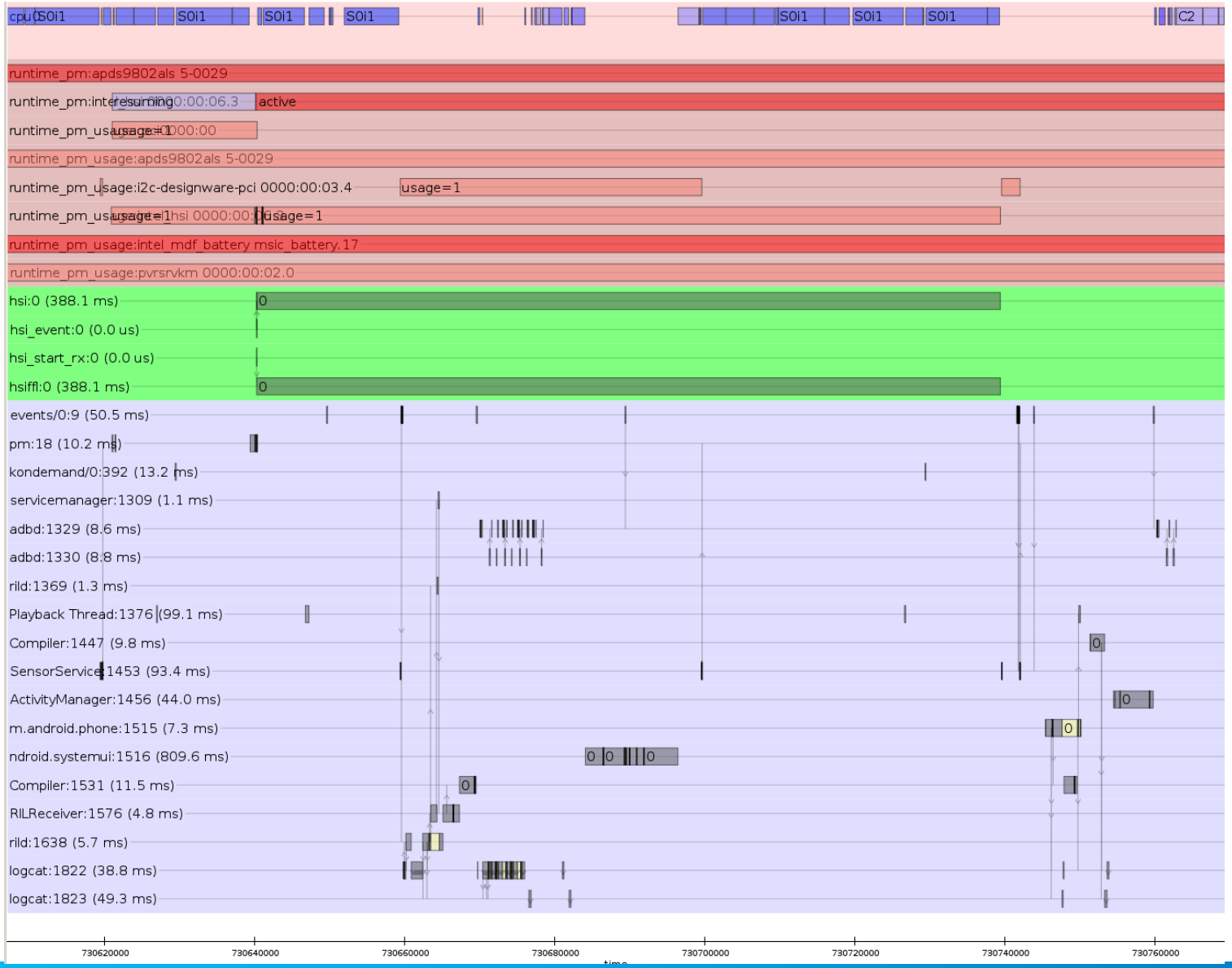
I also add a wake event to see where I come from

```
from timechart.plugin import *
from timechart import colors
from timechart.model import tcProcess

class hsi(plugin):
    additional_colors = """
hsi_bg                #80ff80
"""
    additional_ftrace_parsers = [
        ]
    additional_process_types = {
            "hsi":(tcProcess, MISC_TRACES_CLASS),
        }
    @staticmethod
    def do_function_ffl_start_rx(proj,event):
        process = proj.generic_find_process(0,"hsiffl","hsi")
        caller = proj.generic_find_process(0,event.caller,"hsi")
        proj.generic_process_start(process,event,build_p_stack=False)

        proj.generic_process_single_event(caller,event)
        proj.generic_add_wake(caller, process,event)

    @staticmethod
    def do_function_ffl_stop_rx(proj,event):
        process = proj.generic_find_process(0,"hsiffl","hsi")
        proj.generic_process_end(process,event,build_p_stack=False)
    @staticmethod
    def do_function_hsi_start_rx(proj,event):
        process = proj.generic_find_process(0,"hsi","hsi")
        caller = proj.generic_find_process(0,event.caller,"hsi")
        pidcaller = proj.generic_find_process(event.common_pid,event.common_comm,"hsi")
        proj.generic_process_start(process,event,build_p_stack=False)

        proj.generic_process_single_event(caller,event)
        proj.generic_add_wake(caller, process,event)

    @staticmethod
    def do_function_hsi_stop_rx(proj,event):
        process = proj.generic_find_process(0,"hsi","hsi")
        proj.generic_process_end(process,event,build_p_stack=False)

plugin_register(hsi)
```

Results

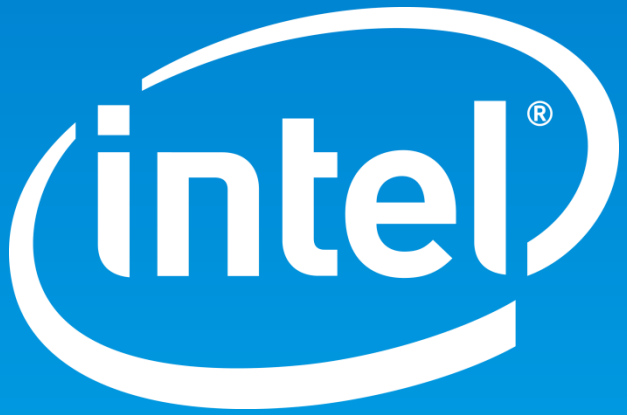

runtime_pm:apds9802als 5-0029
active
runtime_pm:intel_hsi 0000:00:06.3
active
runtime_pm_usage: pci0000:00
usage=1
runtime_pm_usage:apds9802als 5-0029
usage=1
runtime_pm_usage:i2c i2c-5
runtime_pm_usage:i2c-designware-pci 0000:00:0
usage=1
runtime_pm_usage:intel_hsi 0000:00:06.3
usage=1
usage=1
runtime_pm_usage:intel_mdf_battery msic_battery.17
runtime_pm_usage:lis3lv02d_i2c 5-0018
usage=1
usage=3
usage=3
usage=3
usage=3
hsi:0 (388.1 ms)
0
0
hsi_event:0 (0.0 us)
hsi_start_rx:0 (0.0 us)
hsiffl:0 (388.1 ms)
0
0
events/0:9 (50.5 ms)
pm:18 (10.2 ms)
kondemand/0:392 (13.2 ms)
file-storage:849 (12.0 ms)
servicemanager:1309 (1.1 ms)
adbd:1326 (20.8 ms)
adbd:1329 (8.6 ms)
adbd:1330 (8.8 ms)
rild:1369 (1.3 ms)
Playback Thread:1376 (99.1 ms)
system_server:1442 (14.6 ms)
Compiler:1447 (9.8 ms)
SurfaceFlinger:1450 (47.5 ms)
SensorService:1453 (93.4 ms)
er.ServerThread:1454 (19.1 ms)
ActivityManager:1456 (44.0 ms)
m.android.phone:1515 (7.3 ms)
ndroid.systemui:1516 (809.6 ms)
Compiler:1531 (11.5 ms)
RILReceiver:1576 (4.8 ms)
rild:1637 (751.0 us)
rild:1638 (5.7 ms)
com.android.mms:1655 (23.9 ms)
Compiler:1660 (15.0 ms)
AsyncQueryWorke:1762 (10.7 ms)
logcat:1822 (38.8 ms)
logcat:1823 (49.3 ms)
SmsReceiverServ:2524 (2.7 s)
730500000
730600000
730700000
730800000
730900000
731000000
731100000
time

Results Zoomed



Questions?

Thank You


intel