Power management

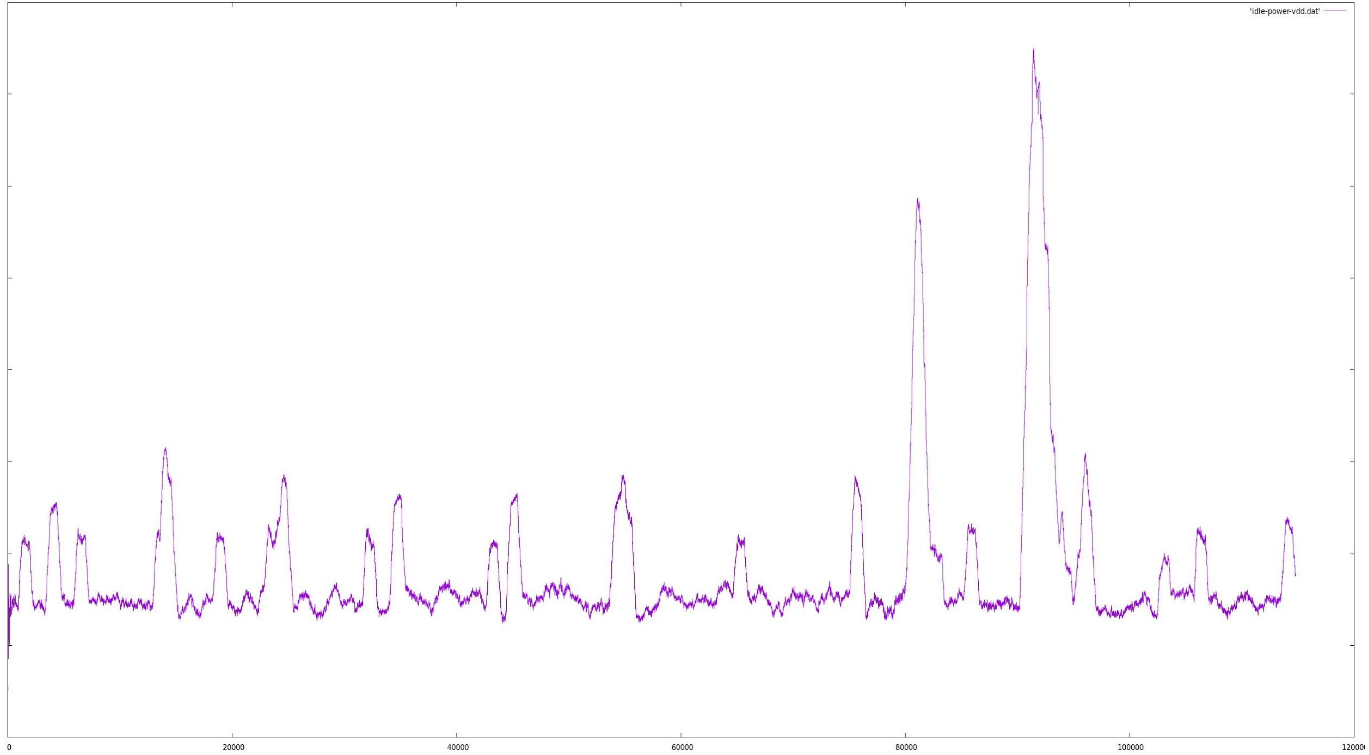
IRQ next event prediction - Where are we ?

Daniel Lezcano

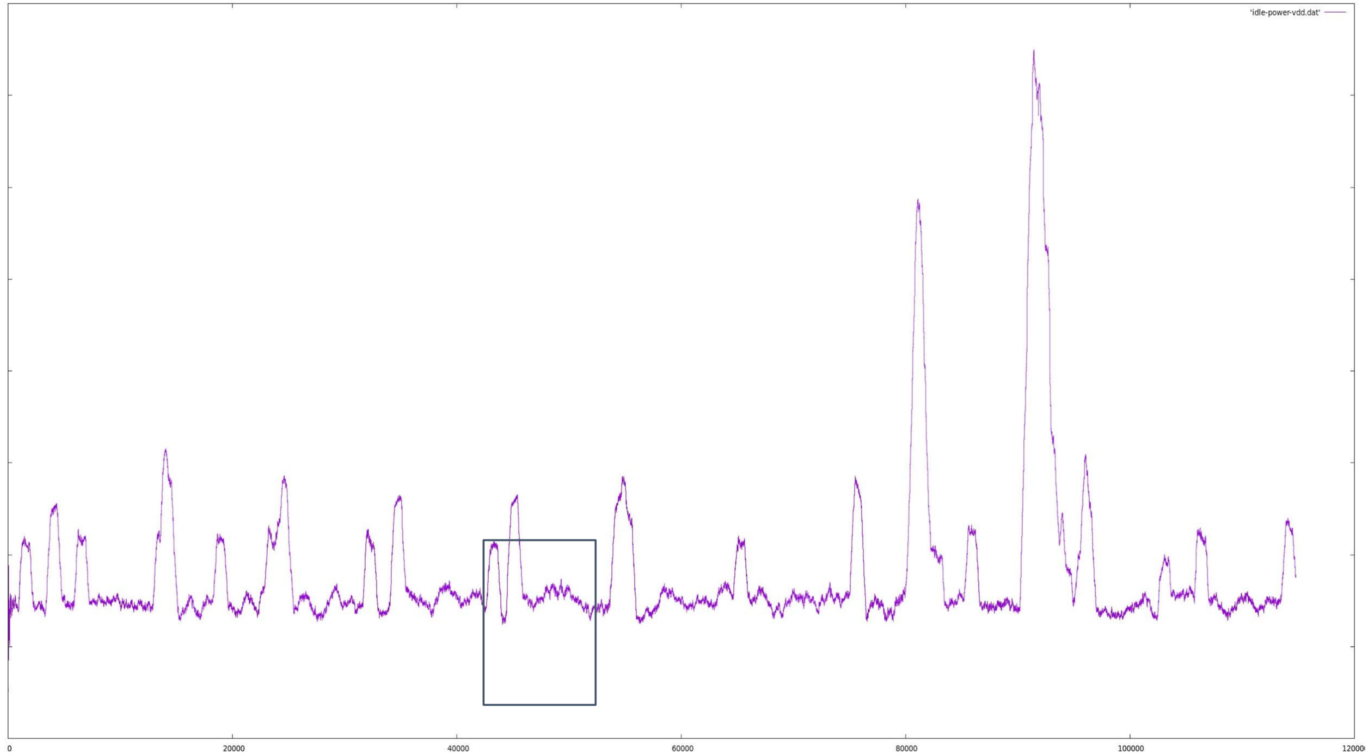
Agenda

- Introduction
- Energy / Idle states / Break even
- Finding the sleep duration
- The sources of wake up
- The governor and the heuristics
- Energy consumption and governors
- Changing the approach
- Measuring the events
- Predict the next event
- A dedicated embedded governor
- Comparisons
- Conclusion

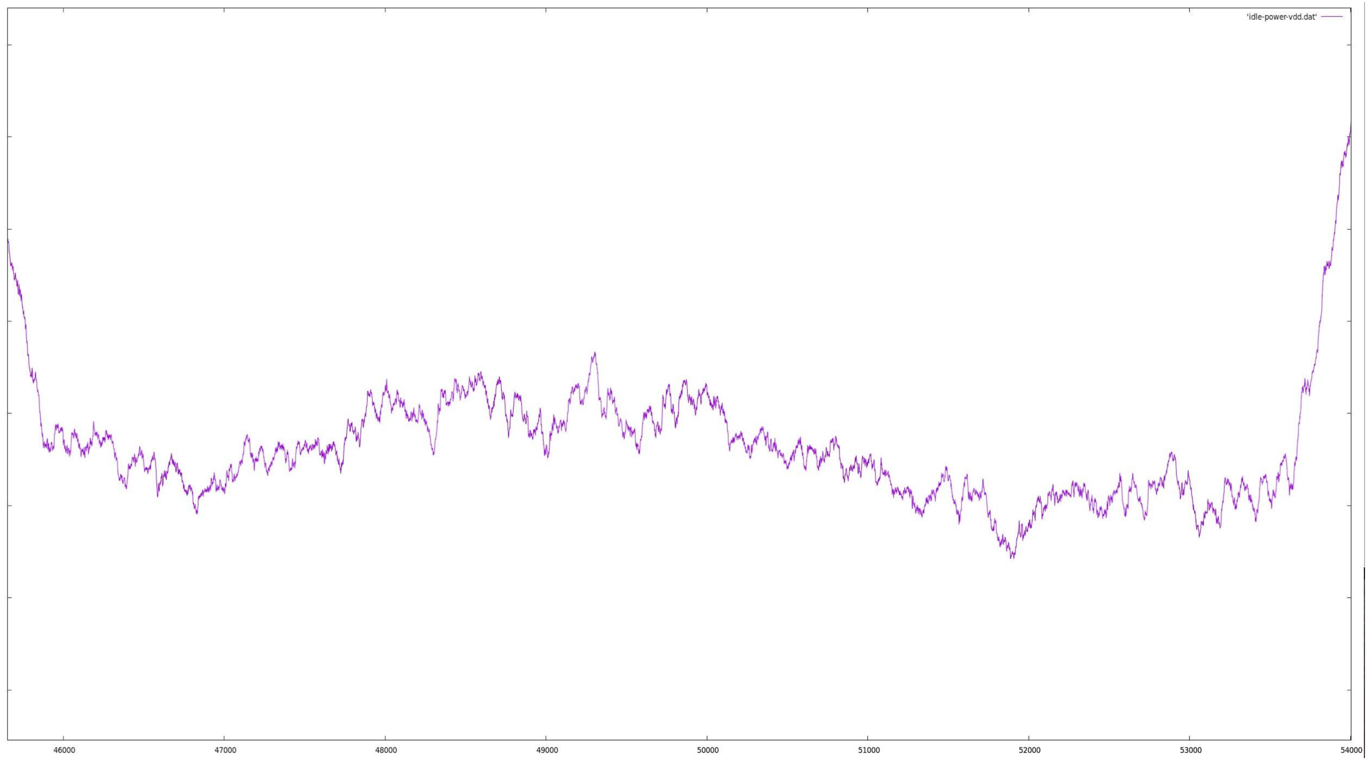
Consumption vs idle states



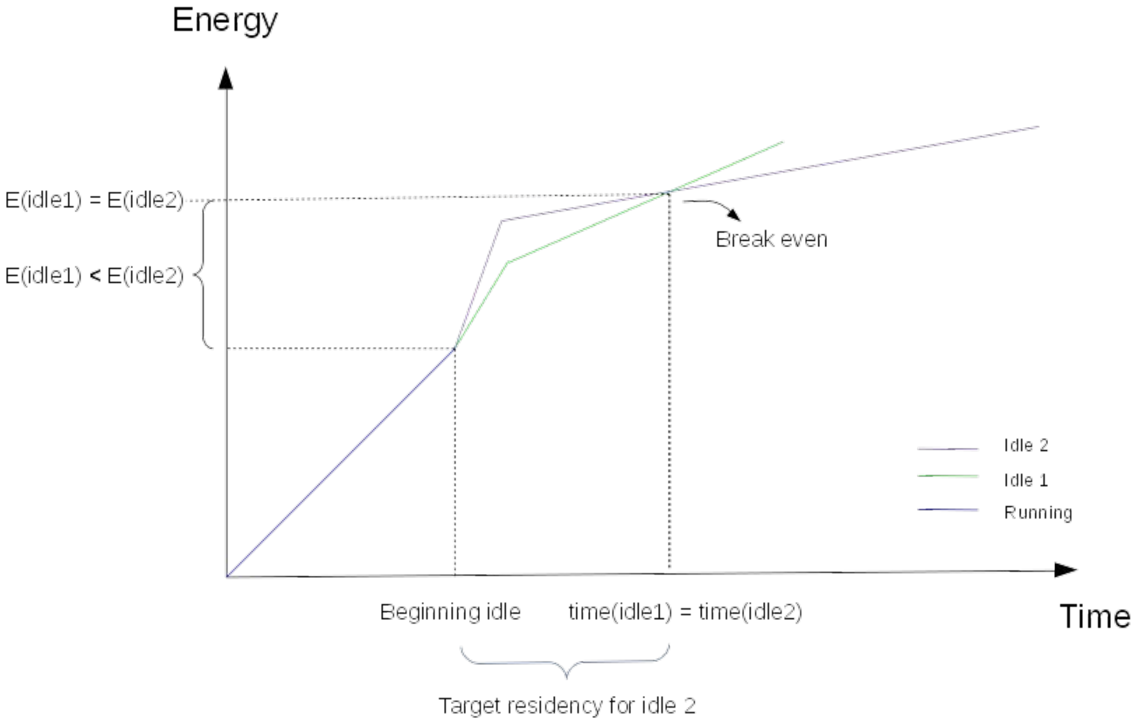
Consumption vs idle states



Consumption vs idle states



Consumption vs idle states



Computing the target residency

- Formula to compute the minimum residency time

$$time = \frac{Widle2 - Widle1}{Pidle1 - Pidle2}$$

- Demonstration available on the PMWG wiki page
- Alternatively, empiric approach presented at HKG18

Idle states characteristics

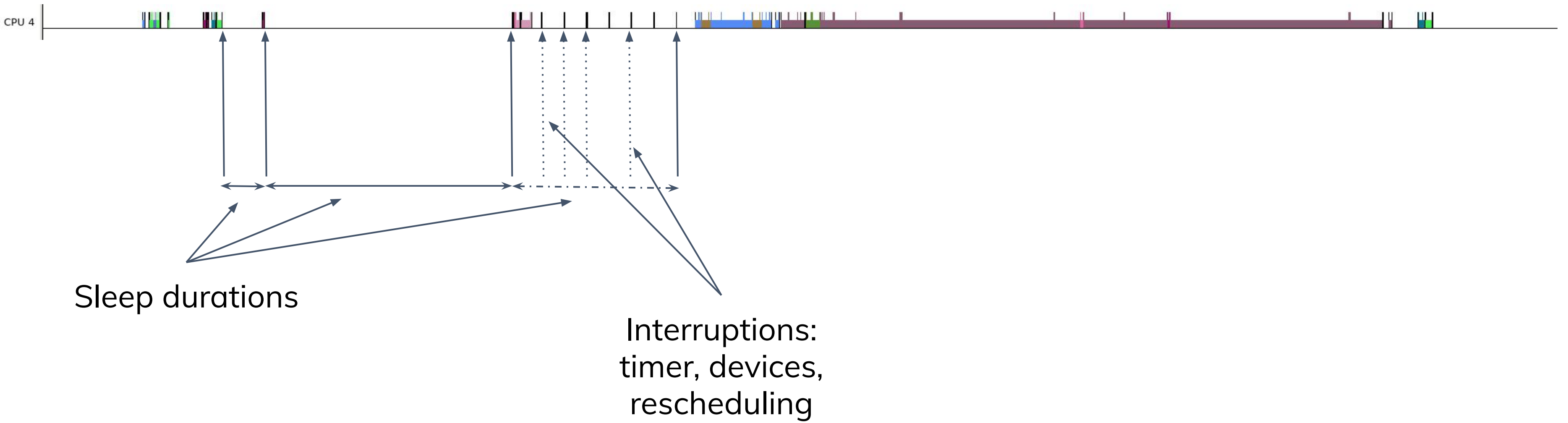
- Idle states must be described accurately
    - Target residencies
        - Usually very approximate values
    - Exit latencies per OPP
        - Only worst case is provided
    - Power at the idle state per OPP
        - These are not available

Choosing the idle state

- Take decision on which idle state to choose
- Based on past events
- Try to predict the future
- Algorithm must be simple

Sleep durations

- Origin of the wake up source
- Statistics on the sleep durations



Problematic

- As we read the sleep duration, the source of wake up can be anything
  - How do we sort out this ?
  - We try to predict the scheduler behavior
  - We try to predict the interruption with the noise of the scheduling + timers

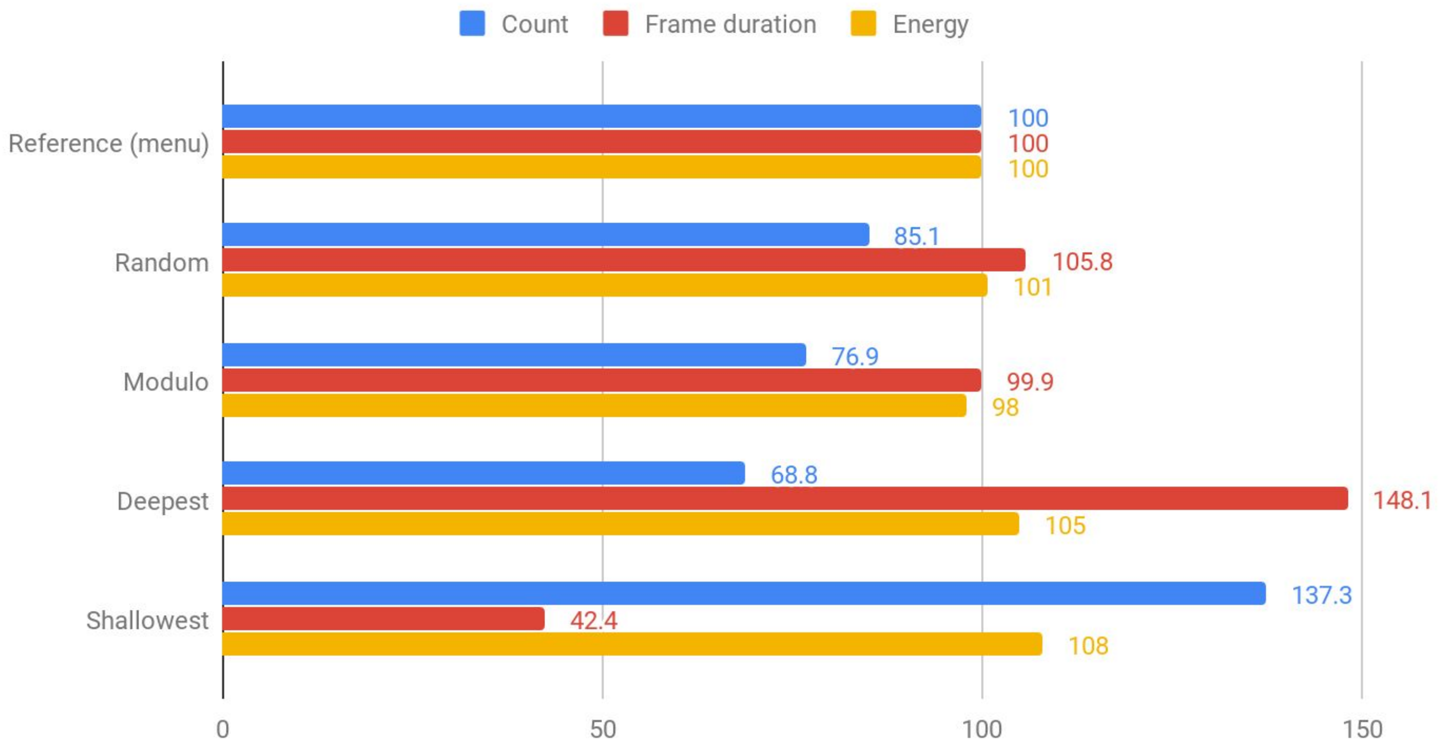
- That can work only if there are periodic wakes up
  - Specific workload, especially IOs

Experiments with governors

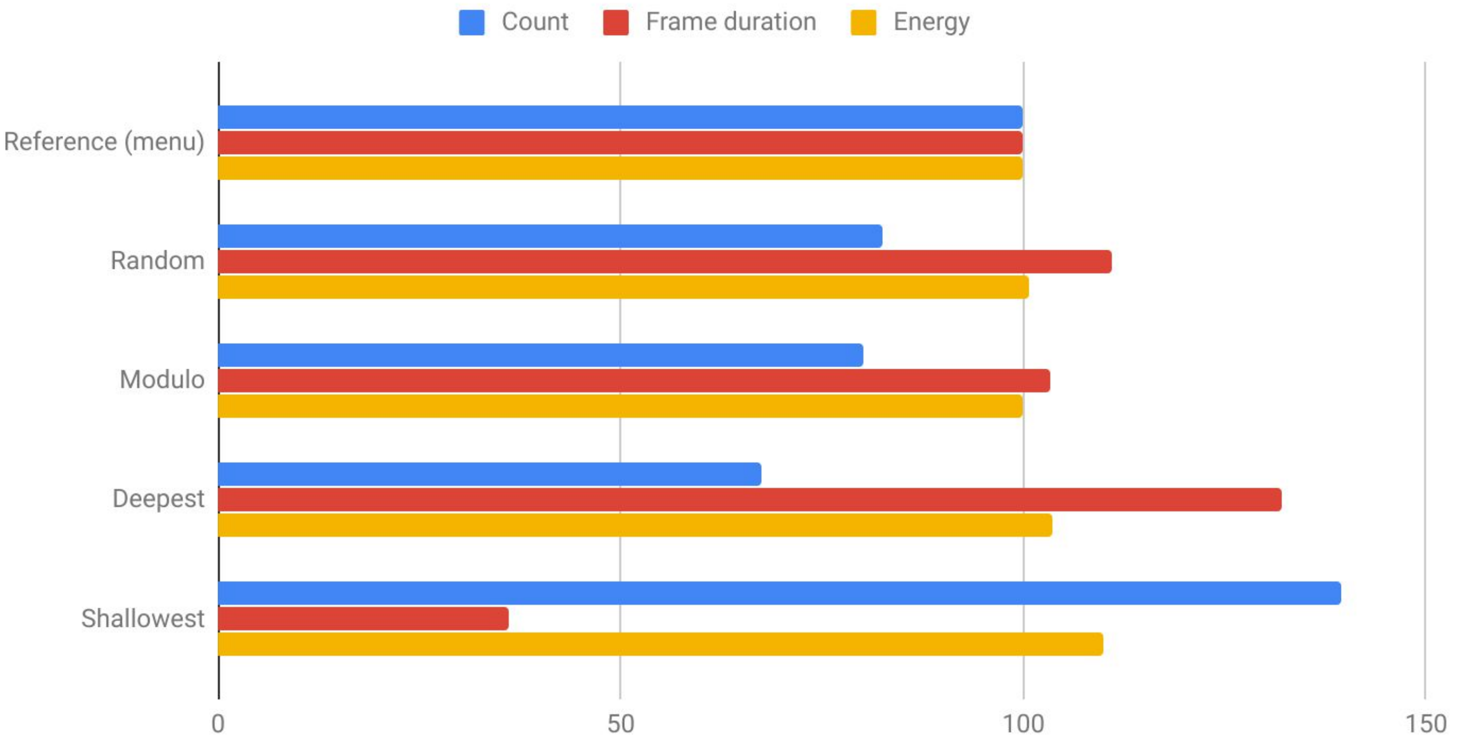
Let's create dummy governors and compare them to the reference: the menu governor

- Random governor: Randomly choose an idle state
- Modulo governor: Always +1 on the selected state modulo number of states
- Deepest governor: Always choose the deepest idle state
- Shallowest governor: Always choose the shallowest idle state

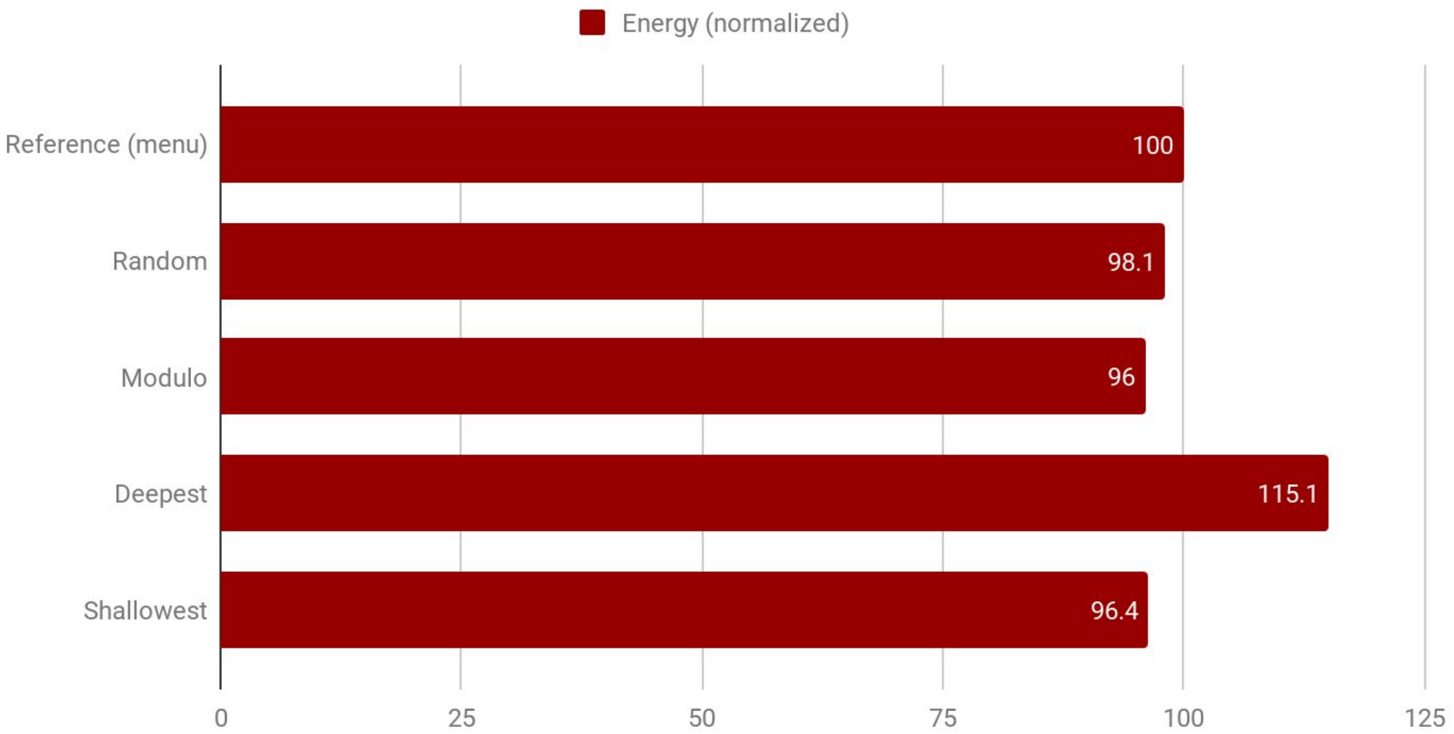
Jankbench / image list vs governors



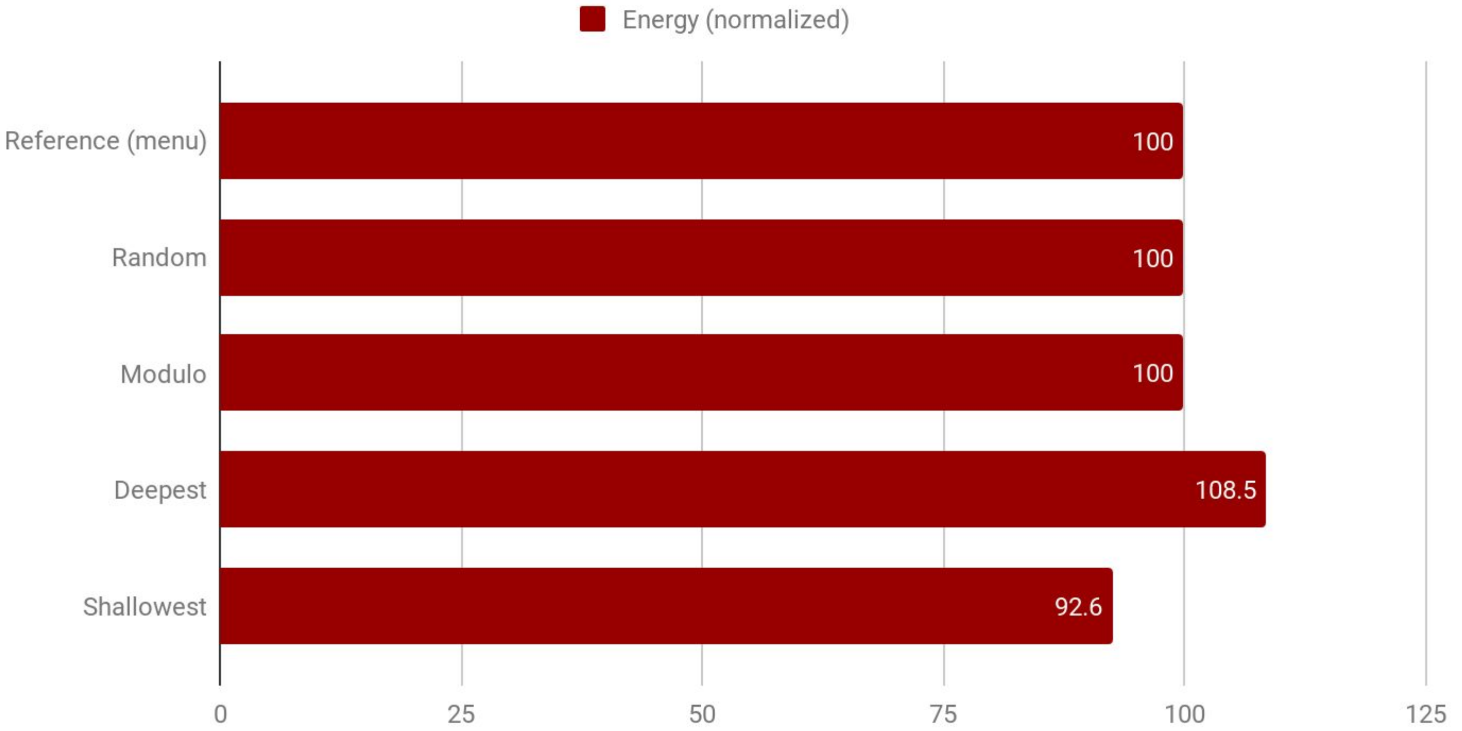
Jankbench / edit text vs governors



Exoplayer audio vs governors (no frame dropped)



Exoplayer video vs governors (no frame dropped)

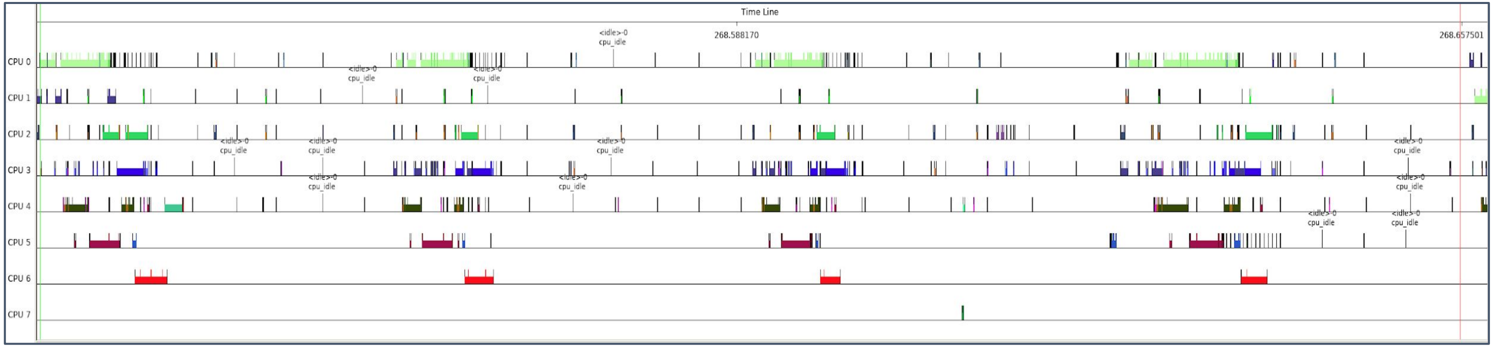


Observations

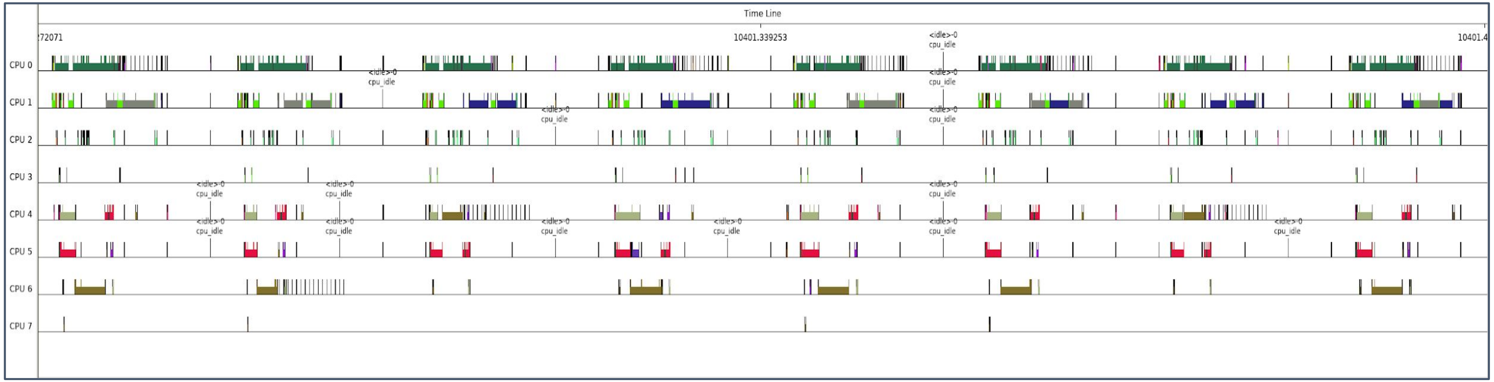
- Going always for the deepest idle state kill performance and consume more energy
- “Randomly” choosing idle state gives same or better results than the menu governor
- Using the shallowest idle state saves up to 8% of energy with audio and video
- Using the shallowest idle state reduces the frame rendering duration up to 58% with an energy drop of 8%
- What is going on ?

What is going on? (Jankbench test1)

menu

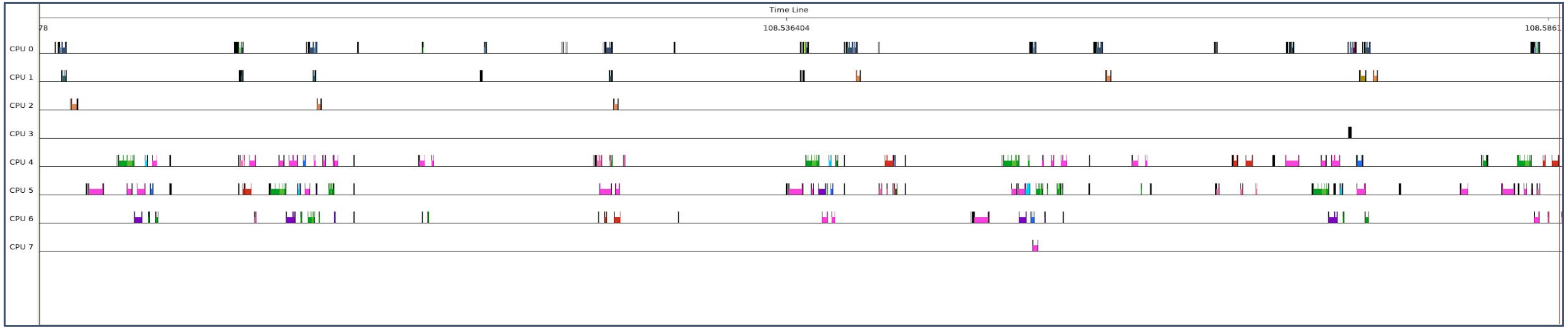


wfi

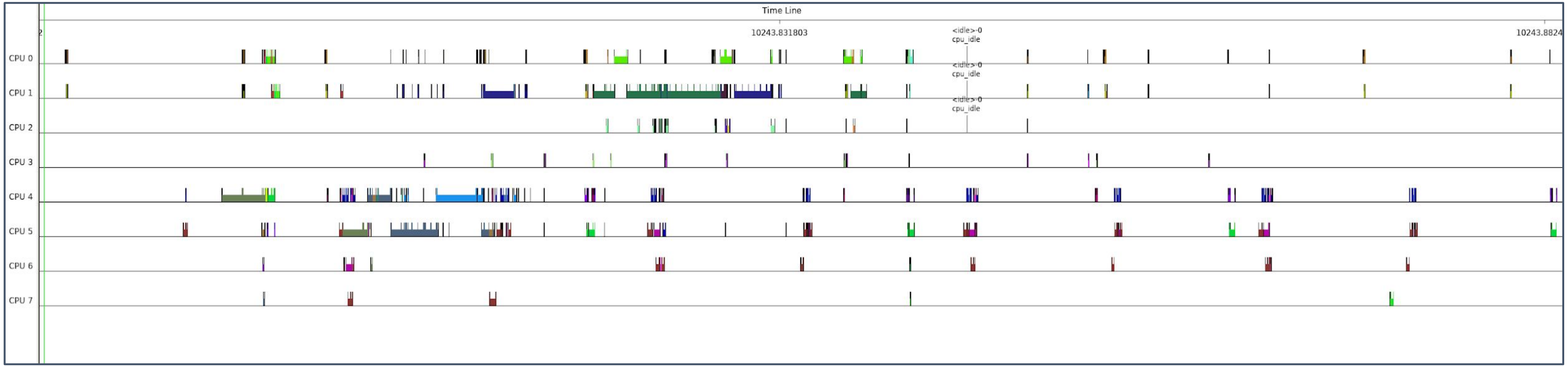


What is going on? (exoplayer ogg)

menu



wfi

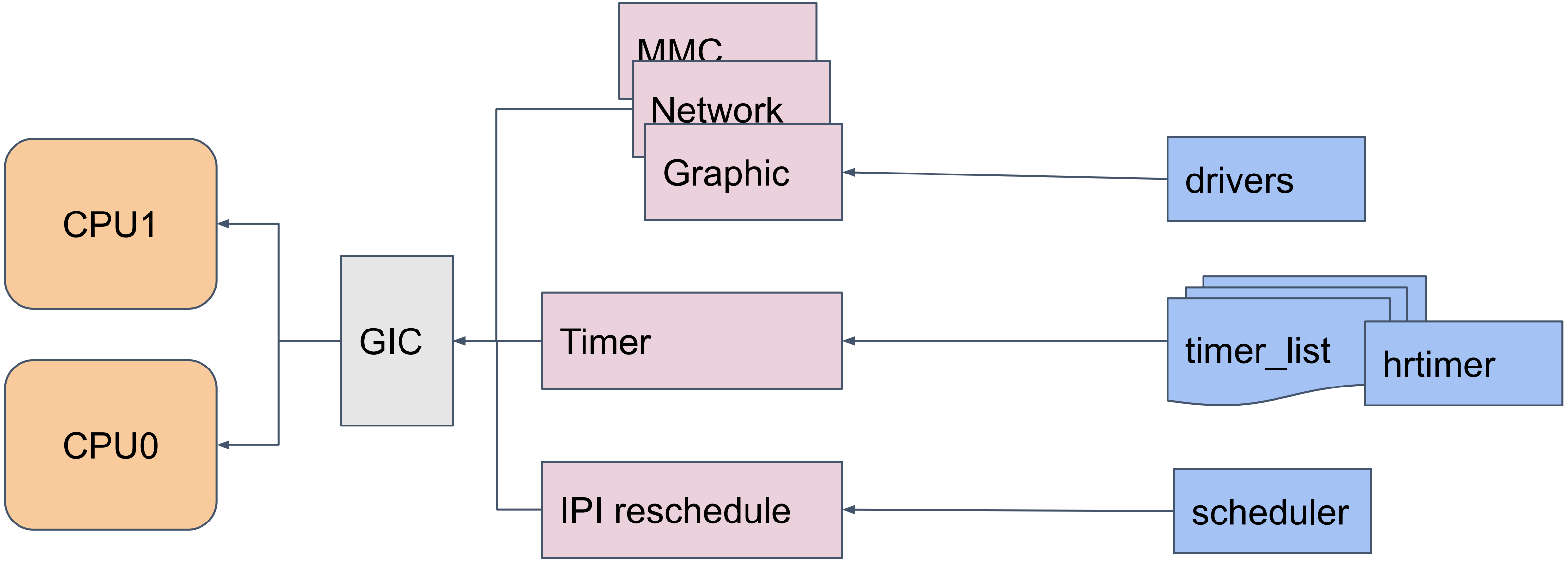


What is going on ?

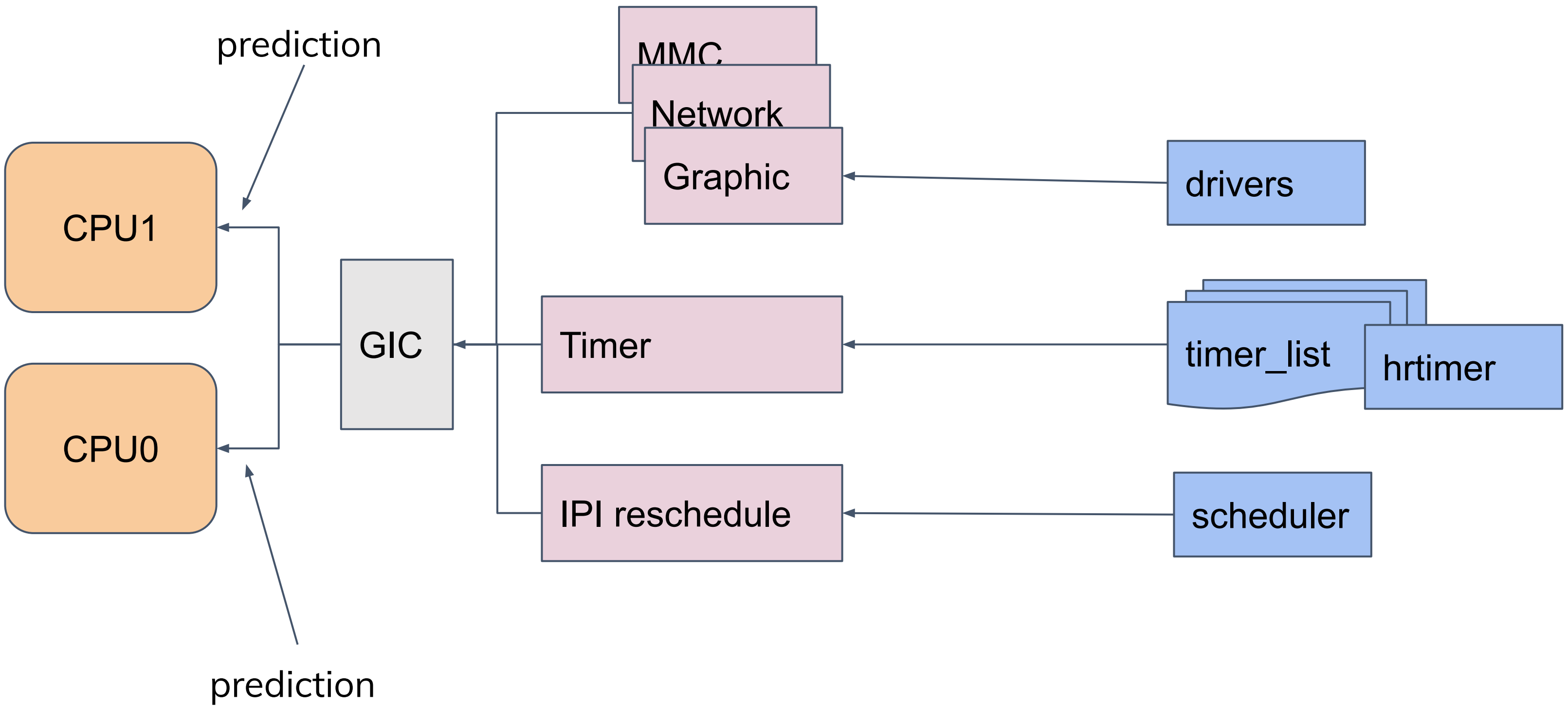
- EAS scheduler behaves differently regarding the idle states:
  - Race to idle
  - Tasks are packed

- The menu governor is doing a lot of mispredictions

Wake up sources

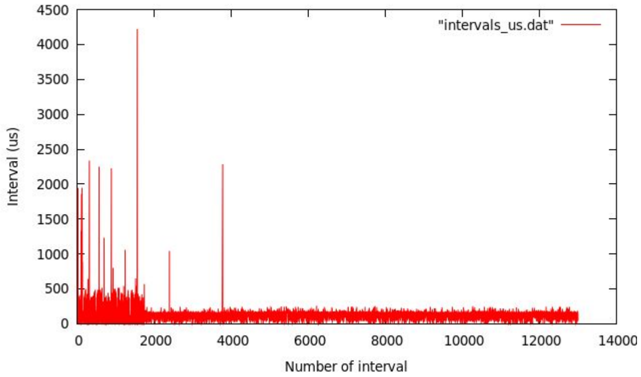


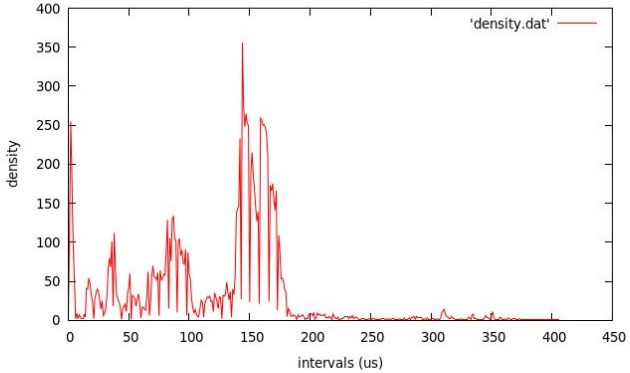
Wake up sources



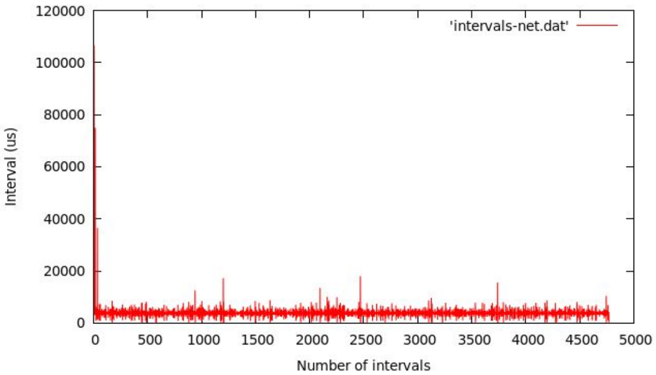
How behave devices?

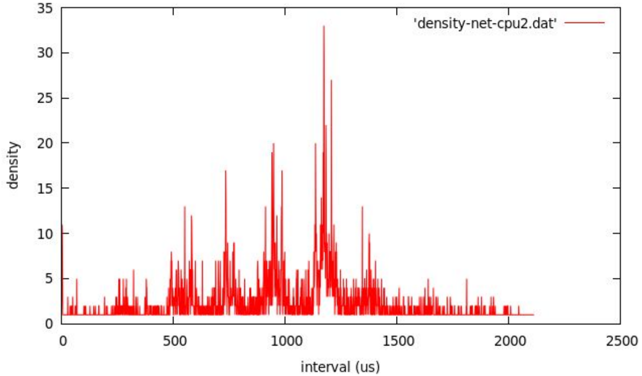
- SSD





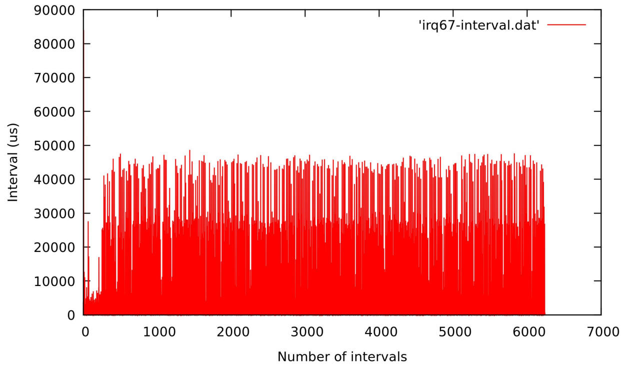
- Network

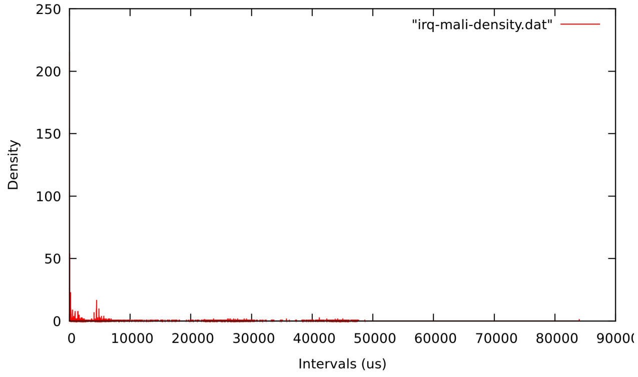




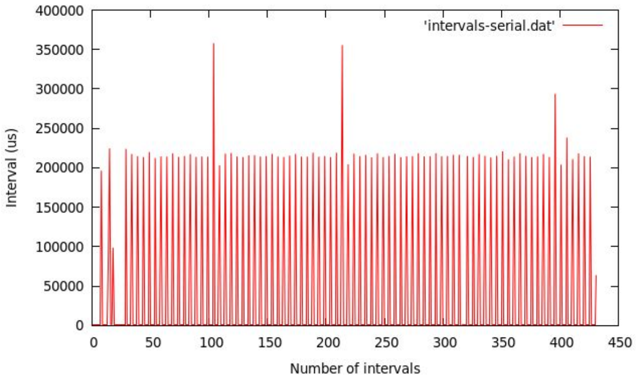
How behave devices?

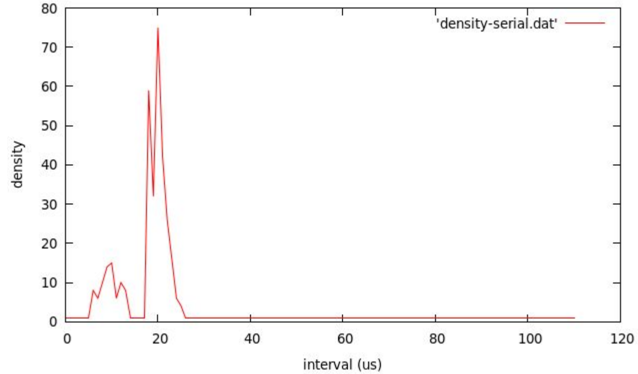
- Graphics



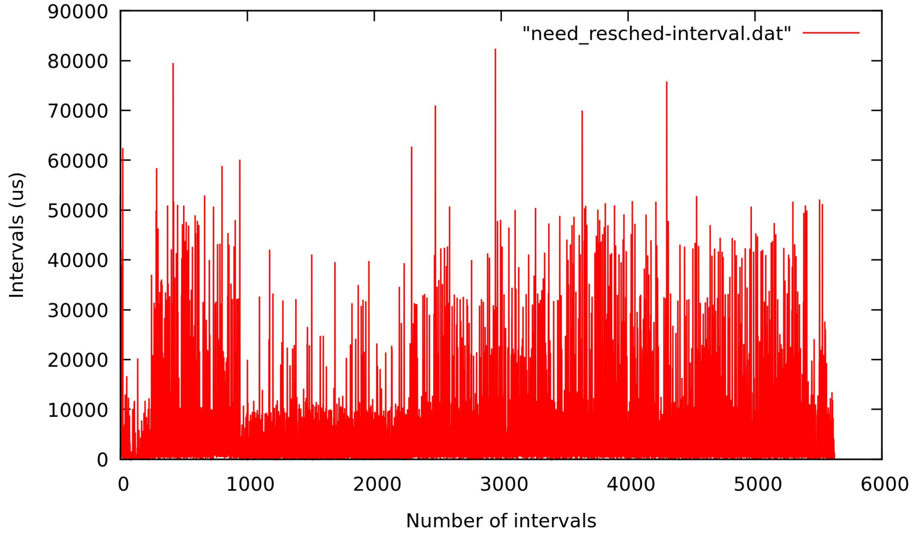


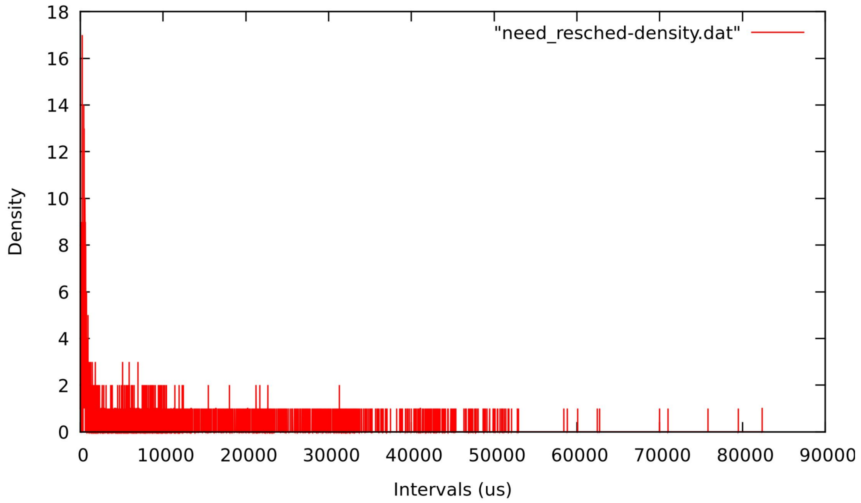
- Console





How behaves the idle task rescheduling





How behave the timers?

That’s a good question, the answer is “as expected”

We always know the next event for the timer

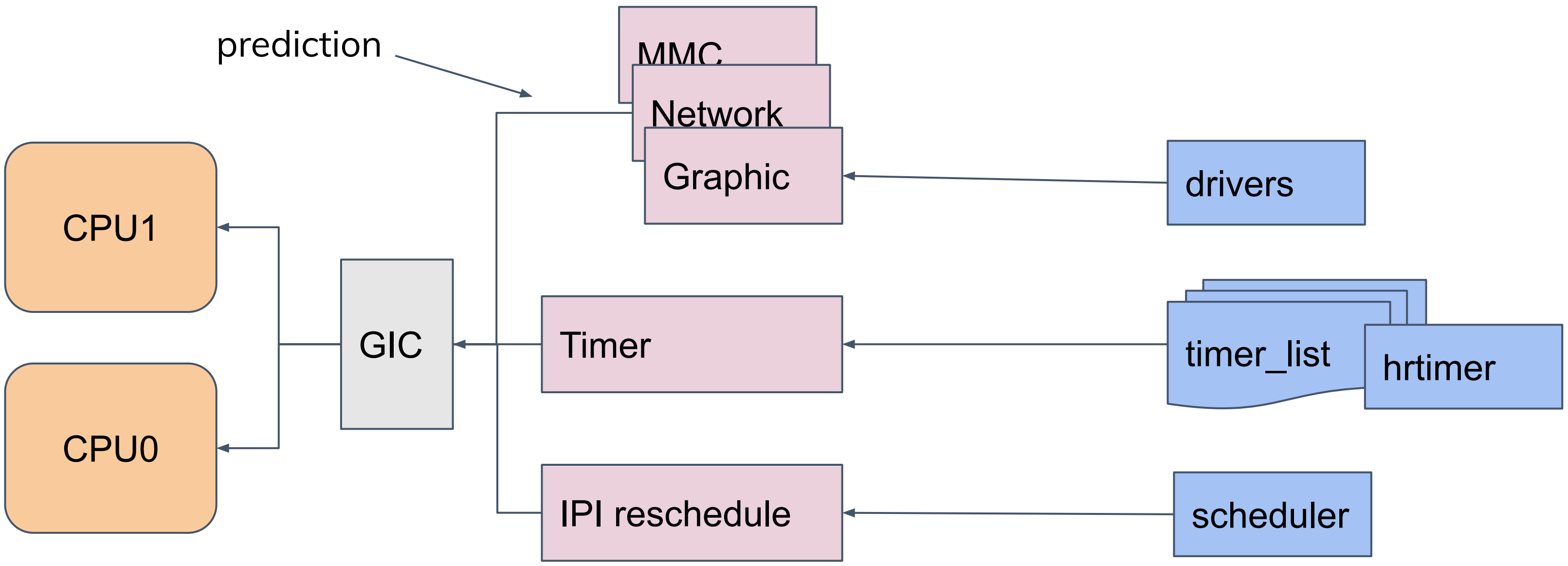
Observations

- Devices can have periodic interrupt
  - Periodicity in the intervals
  - Periodicity of a group of intervals
- Idle task rescheduling is almost random
  - Based on scheduled work
  - Tasks taking locks
  - Tasks blocked on IO
- Timers give an accurate information for the next wakeup
- Side note: On mobile, interrupts are usually pinned on CPU0

Hypothesis

- Why not predict for each wake up source ?
    - Per interrupt
    - Per need_resched duration
    - Make scheduler idle wise
    - Timers are predictable

Wake up sources



Predicting the interrupts from devices

- Store the interrupts <irq,timestamp> when they happen
- At idle time, look at the interrupt history and compute intervals
- Store the interrupt intervals in a log2 array
- Use a fast algorithm based on array suffix
- Use the exponential moving average for similar past events

At runtime

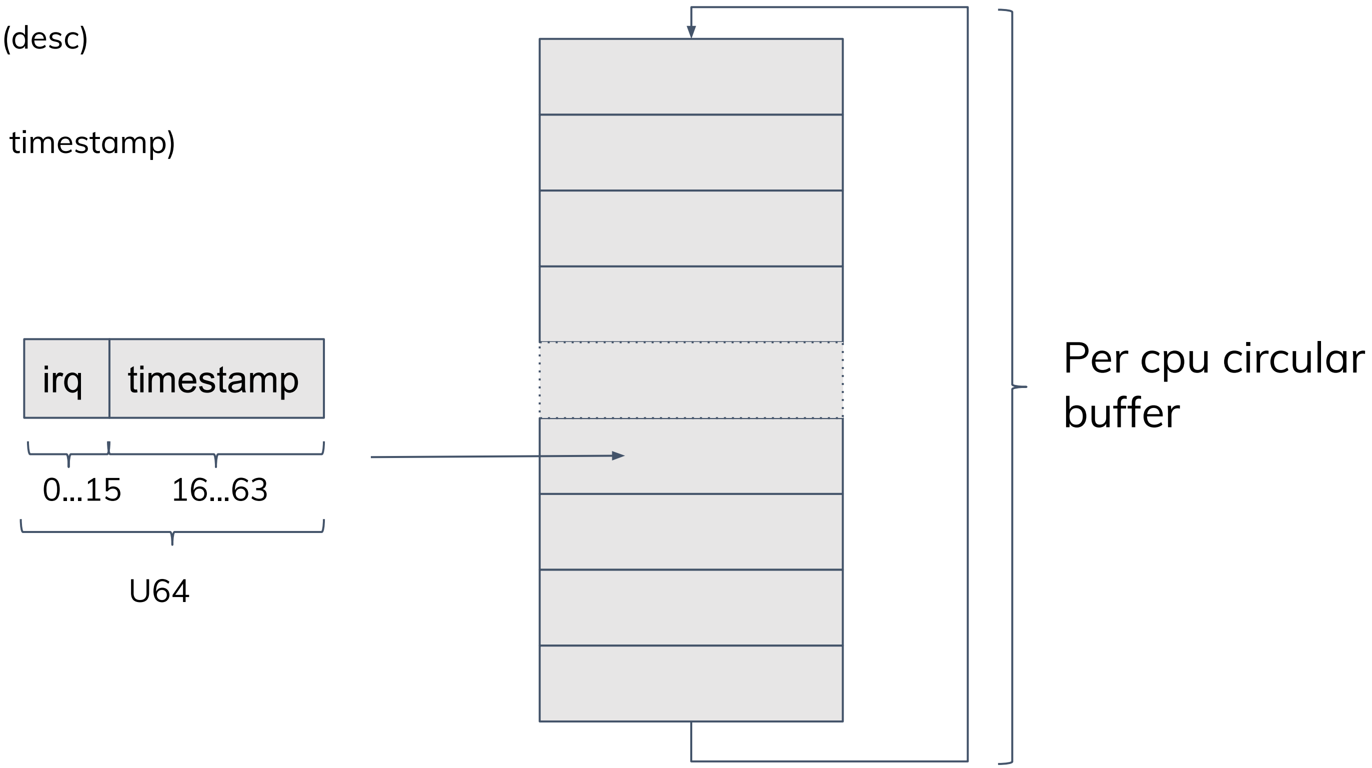
Store the interrupts and timestamp

__handle_irq_event_percpu(desc)

⇒ record_irq_time(desc)

⇒ irq_timings_encode(irq, timestamp)

⇒ irq_timings_store()



At idle time

Discretization of intervals

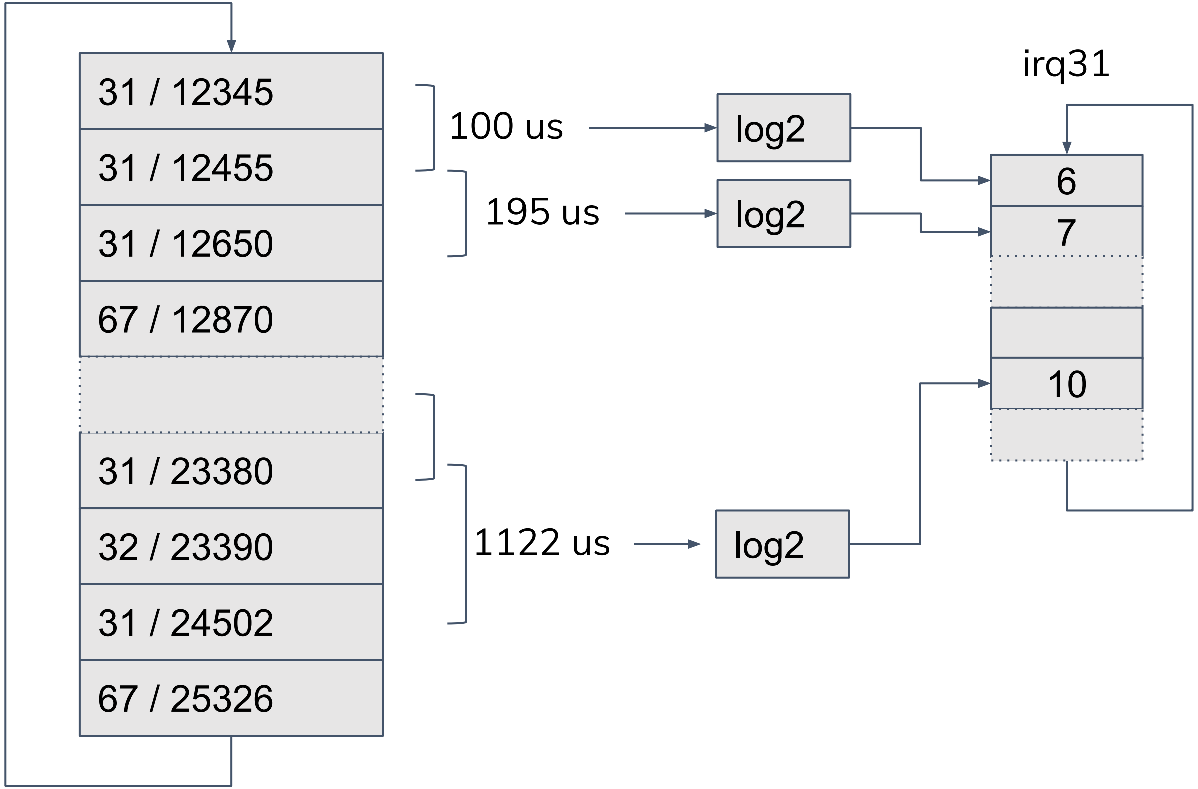
- High number of different values

- Time events: the higher the interval, the lower the precision

- Group the intervals per range
  - [0 , 2[[2 , 4[[4 , 8[[8 , 16[[16 , 32[... [2^{31}, ∞ [
  - An array of 32 values

- Log2 is fast and has dedicated ASM function

Compute intervals on log2 basis

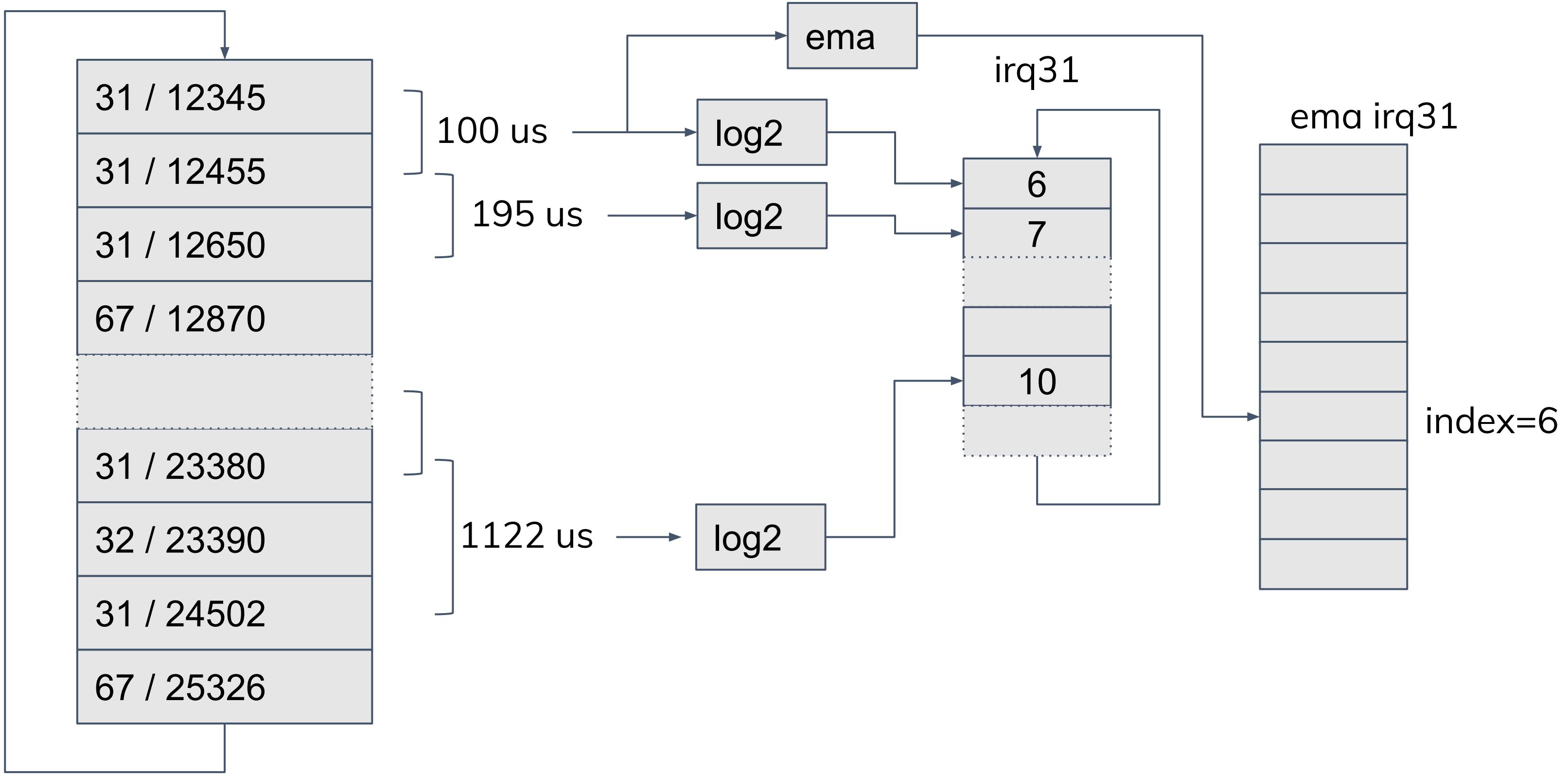


Tracking signals with EMA

- Each intervals is separately tracked with exponential moving average

- Exponential moving average:
  - Stock value tracking
  - Very fast
  - Tweakable via alphas

Store in EMA array



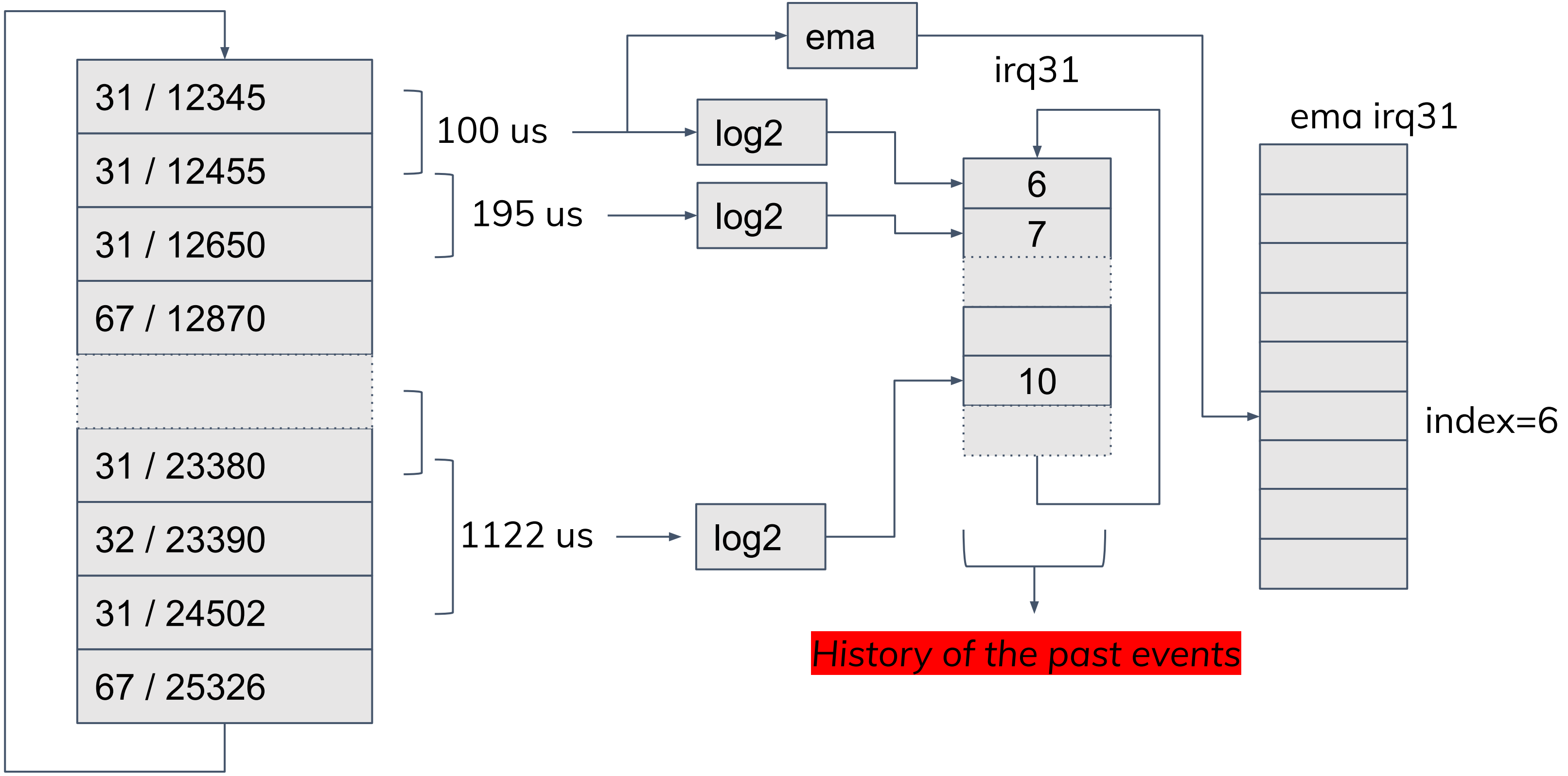
Array suffix

- Data structure for full text indices search, data compression algorithm, bibliometrics, combinatorics on words, bioinformatics

- Build an array of suffix of the terms:
  - Eg. banana has the suffixes : banana, anana, nana, ana, na, a

- Per irq tables have suite of numbers between <1, 32> resulting from log2

Store in EMA array



Array suffix

- An interrupt is predictable if there is a **repetition**
  - **We need to find the period of this repetition**
- Experiment showed a max period of 5 for repeating patterns
  - We assume pattern repeating 3 times has a strong period
  - We take the last 3 x 5 = 15 events
- Example with MMC:

Interval	1385	212240	1240	1386	1386	1386	214415	1236	1384	1386	1387	214276	1234	1384	1388
log2	10	15	10	10	10	10	15	10	10	10	10	15	10	10	10

Max period = 5

Last 3x5 =15 events

Search with array suffix

- Other example with console

Interval	4	5	112	4	6	4	110	4	4	5	112	4	7	4	110
log2	2	2	7	2	2	2	7	2	2	2	7	2	2	2	7

Period					
5	2	2	7	2	2
4	2	2	7	2	
3	2	2	7		
2	2	2			

Search with array suffix

Interval	4	5	112	4	6	4	110	4	4	5	112	4	7	4	110
log2	2	2	7	2	2	2	7	2	2	2	7	2	2	2	7
p=5	2	2	7	2	2	2	2								
p=4	2	2	7	2	2	2	7	2	2	2	7	2	2	2	7
p=3	2	2	7	2	2	7									
p=2	2	2	2												

Search with array suffix

Interval	4	5	112	4	6	4	110	4	4	5	112	4	7	4	110
log2	2	2	7	2	2	2	7	2	2	2	7	2	2	2	7
p=4	2	2	7	2	2	2	7	2	2	2	7	2	2	2	7

period

last pattern length

Next event index = last pattern length % period
Next event index = 3 % 4 = 3

Search with array suffix

- Other example with console

Interval	4	5	112	4	6	4	110	4	4	5	112	4	7	4	110
log2	2	2	7	2	2	2	7	2	2	2	7	2	2	2	7
p=4	2	2	7	2	2	2	7	2	2	2	7	2	2	2	7

period

last pattern length

Next event index = last pattern length % period
Next event index = 3 % 4 = 3

suffix p=4	2	2	7	2

Search with array suffix

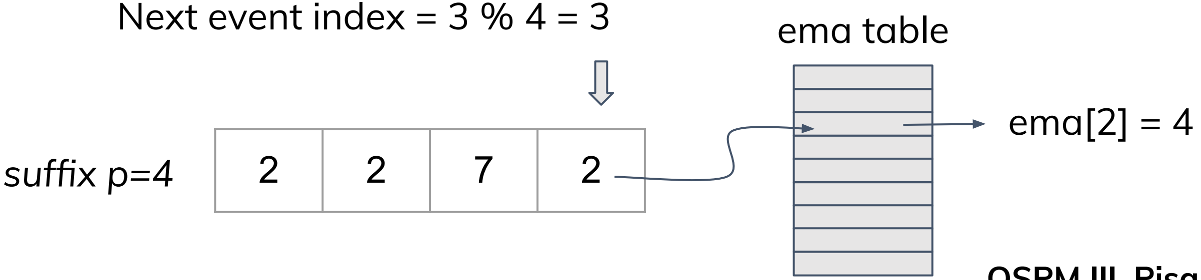
- Other example with console

Interval	4	5	112	4	6	4	110	4	4	5	112	4	7	4	110
log2	2	2	7	2	2	2	7	2	2	2	7	2	2	2	7
p=4	2	2	7	2	2	2	7	2	2	2	7	2	2	2	7

period

last pattern length

Next event index = last pattern length % period
Next event index = 3 % 4 = 3



Embedded cpuidle governor

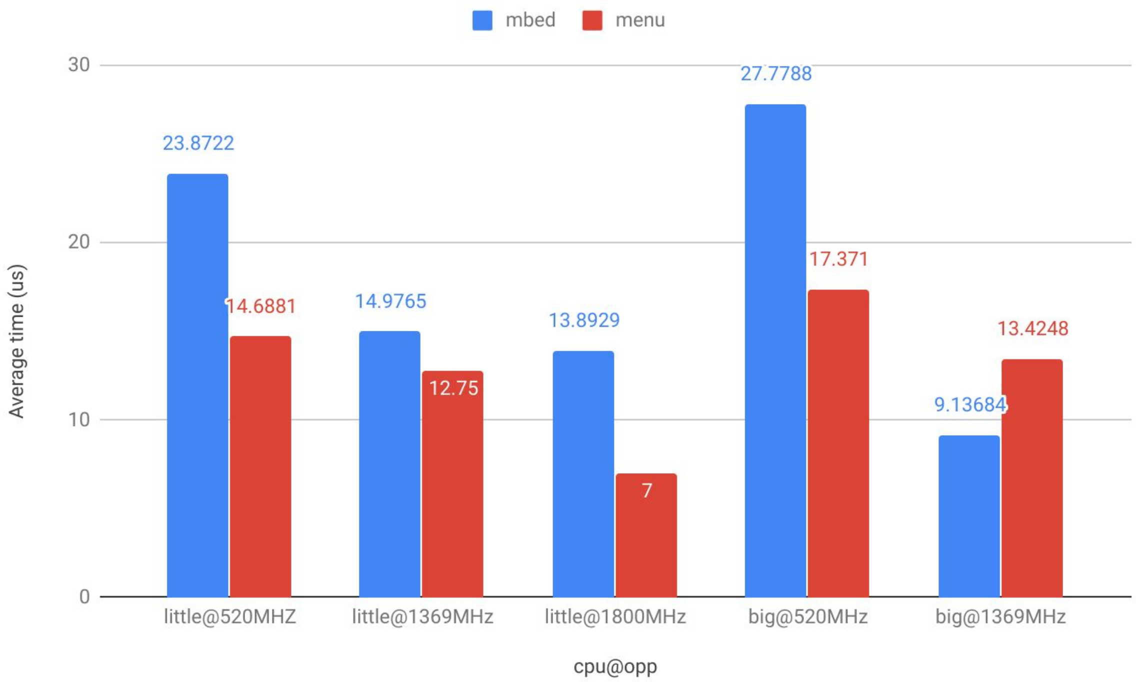
- Makes use of the interrupt prediction

- Clearly identifies the source of wake up in the prediction path

- Designed to work with the embedded systems, especially mobile
  - Tweaked for mobile workload (video, audio, benchmarks)
  - Iteratively improved with non-regression testing for existing and defined workloads
  - Avoids to use biased heuristics

- How does it compare with the existing ?

Selection latency



Selection latency

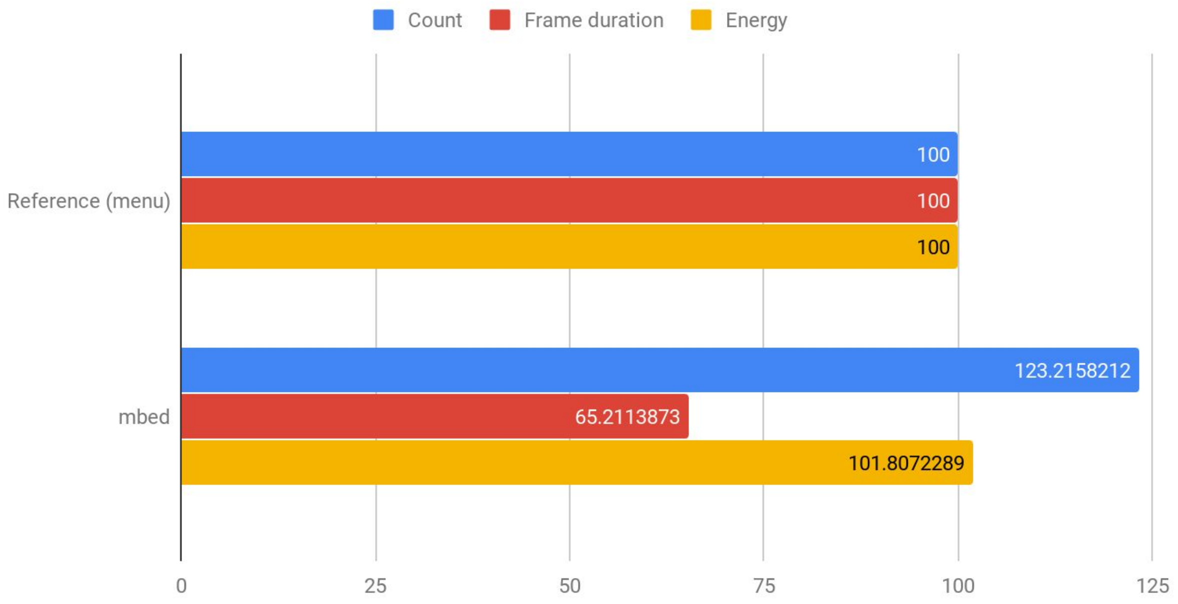
- Higher latency on the CPU with the interrupts
  - Usually CPU0

- Other CPUs have a negligible latency

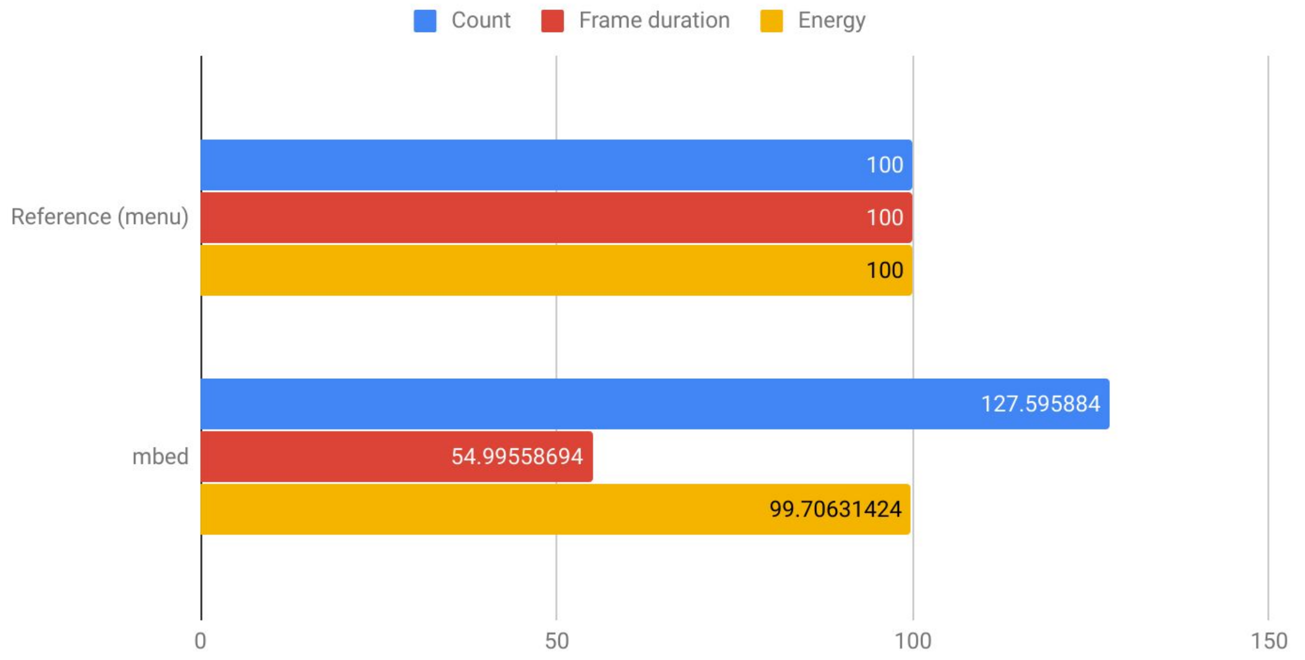
- The higher the interrupts number, the higher the load, the lower the idle duration
  - Do we really care about these latencies?

- Some part of the prediction can be still optimized
  - Suffixes on the fly, unpredictable interrupts discarded from the prediction, etc ...

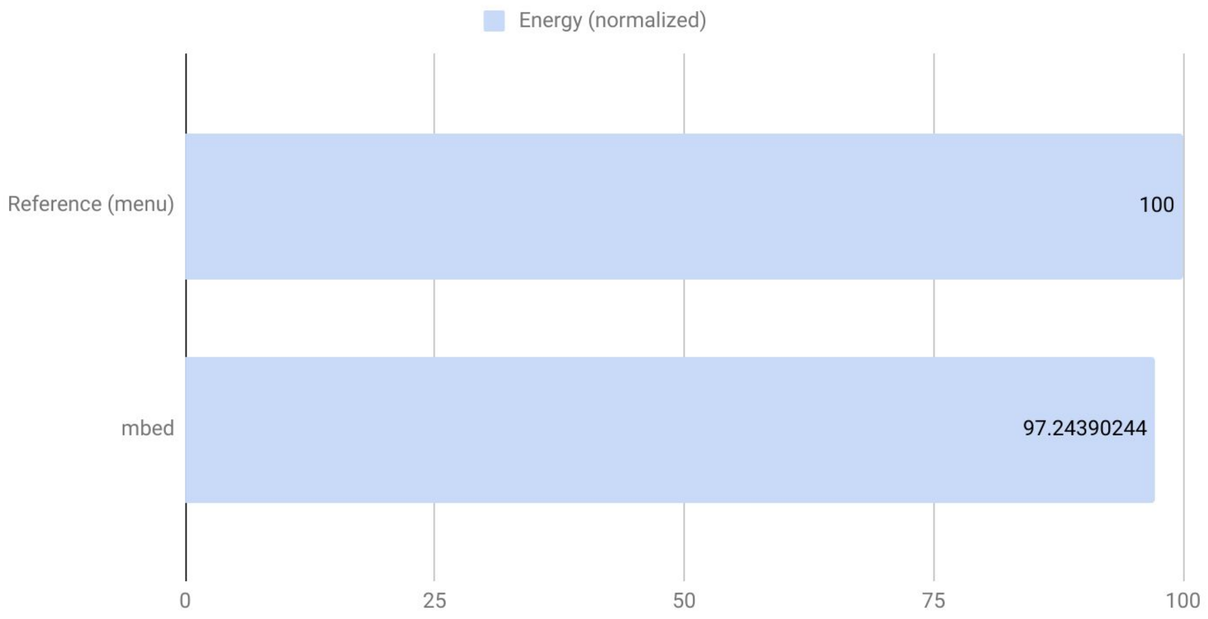
Measurements - Jankbench test1



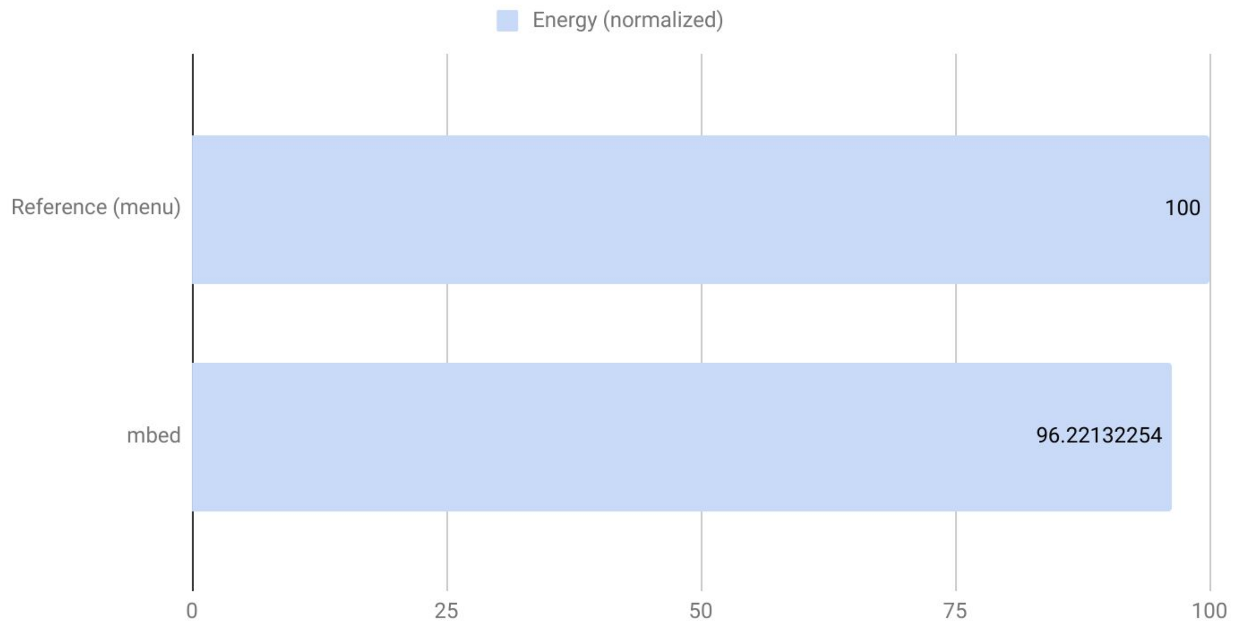
Measurements - Jankbench test2



Measurements - exoplayer (ogg)



Measurements - exoplayer (mov)



Conclusion

- Splitting different wake sources signals to predict works
  - Despite the simplicity of the actual governor we do better predictions
  - Better performances for better energy

- There is still room for more improvements on the mbed governor
  - Identified workload (expecting more than 8% energy improvement for ogg/video)
  - Identified weaknesses in the prediction (need_resched)
  - Scheduler interactions (idle wise)

- Next steps
  - Put noisy wakeup sources apart
  - Offer an API to drivers to register their next interrupt event

Thank you


96Boards.org
Develop & Prototype on the Latest Arm Technology
96Boards is a range of specifications with boards and peripherals offering different performance levels and features in a standard footprint.