EAS – Energy Aware Scheduler

An unbiased look

Vitaly Wool, Konsulko Group

- **Introduction**
- Energy Aware Scheduler
- Qualcomm HMP scheduler
- Comparisons and outcome
- Way forward
- Wrap-up

Completely Fair Scheduler

- The main idea is to maintain balance (fairness) in providing processor time to tasks
- CFS maintains the amount of time provided to a given task to determine if it needs balancing
  - the smaller amount of time a task has been permitted access to the processor — the higher its need for the processor is
- CFS maintains a time-ordered red-black tree
  - Instead of run queues as did predecessors
  - Guarantees O(log(*N*))

CFS operation principles

- Sorts tasks in ascending order by CPU bandwidth received
  - This is where red-black tree comes into play
- The leftmost task off the rbtree is picked up next
  - It has the least spent execution time
  - So that task gets the CPU to restore balance (fairness)
- Considers all CPUs to be the same
  - Works very well in SMP systems
  - Does not work in more complicated cases

big.LITTLE

- big.LITTLE technology is a *heterogeneous* processor architecture which uses two types of cores
  - "LITTLE" cores are designed for maximum power efficiency
  - "big" cores should provide maximal computing power
  - big.LITTLE CPU may have arbitrary number of big / little cores
- big.LITTLE operation
  - Each task may be scheduled for execution either on big or on LITTLE core
    - Depending on task's demand for computing power
  - The aim is for high **peak** performance with low **mean** power

big.LITTLE in a nutshell

Konsulko Group

- The key is **task placement**
  - Wrong task-core distribution kills big.LITTLE advantages
- big.LITTLE puts high requirements on scheduler
  - It should be aware of 2 types of cores
  - It should be *energy aware*
  - it should communicate with the DVFS subsystem
- big.LITTLE scheduling implies heuristics
  - The task placement decision should ideally be made basing on the task's **future** activity

Scheduler for big.LITTLE?

Konsulko Group

- CFS is a good scheduler
  - But it's not really a perfect fit for big.LITTLE
- Extend CFS to be applicable to non-SMP architectures
  - Work started back in 2013
- 2 competing implementations were developed
  - Qualcomm/Codeaurora (HMP scheduler, QHMP)
  - Linaro/ARM (**EAS**)
- We'll concentrate more on EAS

- Introduction
- **Energy Aware Scheduler**
- Qualcomm HMP scheduler
- Comparisons and outcome
- Way forward
- Wrap-up

EAS: basic principles

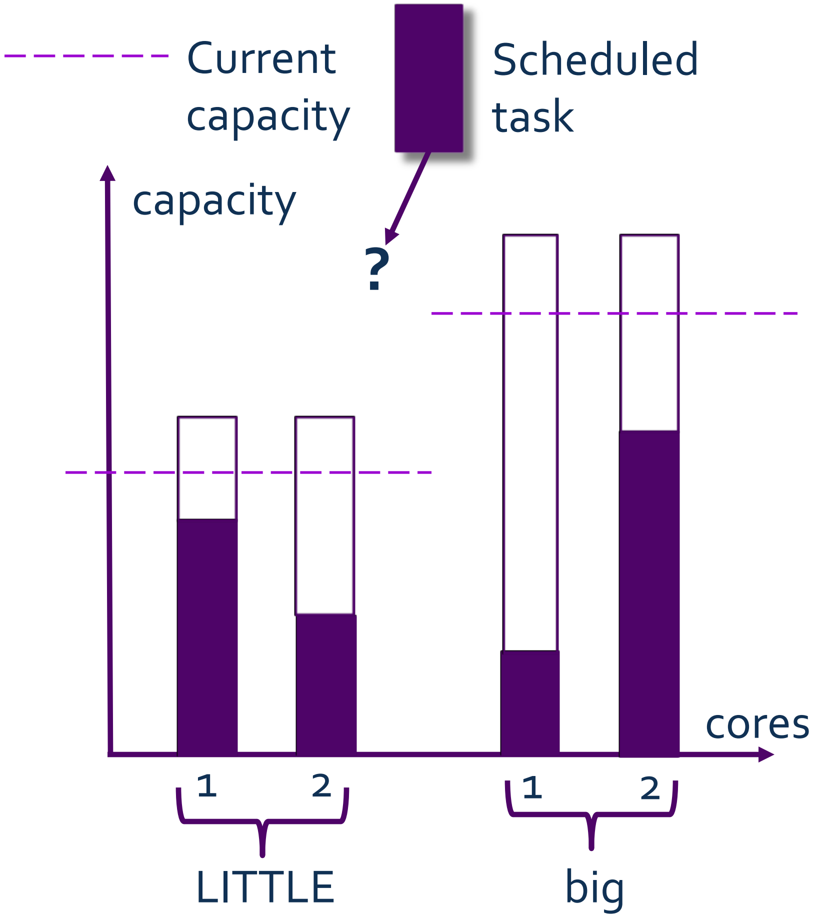
- Task scheduling that considers energy implications
- Decision should be made basing on:
  - System topology
    - E. g. SMP or HMP
  - Power management features
    - CPU Idle states, DVFS
  - Workload for each core
- Work load calculation is basically independent
  - Separate module providing results to EAS

PELT: Per-Entity Load Tracking

- In mainline already, merged in 3.8
  - used by mainline CFS
- The main idea is that process can contribute to load even if it is not actually running at the moment
- PELT tracks load on a per-entity basis
- Let L_i designate the entity's load contribution in period p_i
  - Then the total load is $L = L_0 + L_1 q + L_2 q^2 + L_3 q^3 + \cdots$
    - q is the decay factor

EAS/PELT operation

- Estimate energy
  - $E = P_{idle}t_{idle} + P_{busy}t_{busy}$
- Pick CPU with sufficient spare capacity and smallest energy impact
  - Here both LITTLE and big #2 cores have sufficient capacity
  - the energy impact is smaller with the former



Introduction

Energy Aware Scheduler

Qualcomm HMP scheduler

Comparisons and outcome

Way forward

Wrap-up

Qualcomm HMP scheduler

Konsulko Group

- Tasks are divided into groups
  - By importance
    - Depending on *nice* priority
  - By "size"
    - Depending on the calculated load
    - Task may be "*big*", "*little*" or *other*
    - Thresholds are parametrized
- Scheduling a task should depend on its properties
- Task "size" should be defined somehow
  - It's done basing on *task demand* calculation

HMP scheduler: task demand

- *Task demand* D_{task} is the contribution of a task's running time to a window
  - $D_{task} = \frac{delta\_time \times cur\_freq}{\max\_possible\_freq}$
    - *delta_time* - time of task running on a core in a period of time
    - *cur_freq* - the current frequency of the core this task is running on
    - *max_possible_freq* is the maximum possible frequency across **all** cores
- Calculated over *N* sliding windows (*N* is a parameter)
  - E. g. the average demand $D_{avg} = (D_1 + \cdots + D_N)/N$
  - The best result is achieved with $\mathrm{D} = \max\{D_{avg}, D_1\}$

Task demand scaling

- We already account for difference in maximum frequency
  - D_{task} is calculated in regard to maximum frequency across all cores
- We also need to account for higher performance of big cores
  - $D_{task,scaled} = D_{task} \cdot \frac{rq \rightarrow efficiency}{\max possible\ efficiency}$
    - *Efficiency* is a per-runqueue parameter
    - Usually big cores are considered 2x more effective

“big” and “small” tasks in HMP

- Small task
  - A periodic task with short execution time
  - Can be easily identified using task average demand
- Big task
  - Task producing high CPU load (parametrized, 90%+)
  - Some heavy tasks HMP doesn’t want to count as *big*
    - e.g. background threads in Android
- Some tasks are neither big nor small
- Tasks can change their “size” over time

HMP scheduler and DVFS

Konsulko Group

- HMP scheduler calculates loads anyway
  - It sort of has to, for QoS reasons
    - Take too long to wait for a load increase notification from governor
- CPUFreq governor either runs within a cluster or should be aware of HMP architecture
  - So a truly "standalone" CPUFreq governor will end up duplicating HMP functions
- As a result, HMP scheduler used to come with heavily patched 'performance' governor
  - Which is itself out-of-tree

Introduction

Energy Aware Scheduler

Qualcomm HMP scheduler

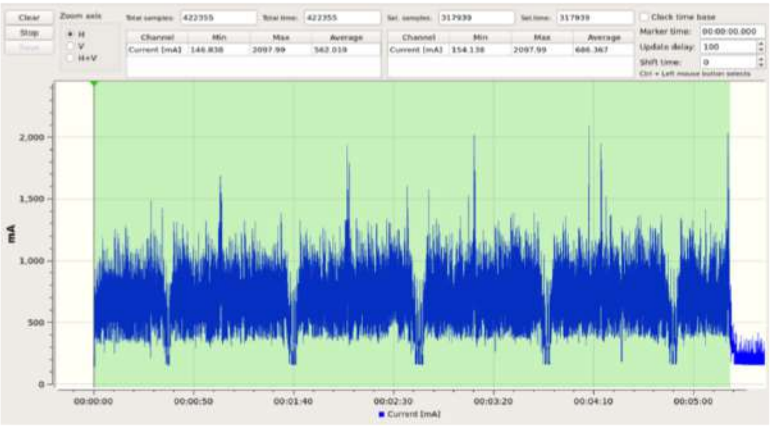
Comparisons and outcome

Way forward

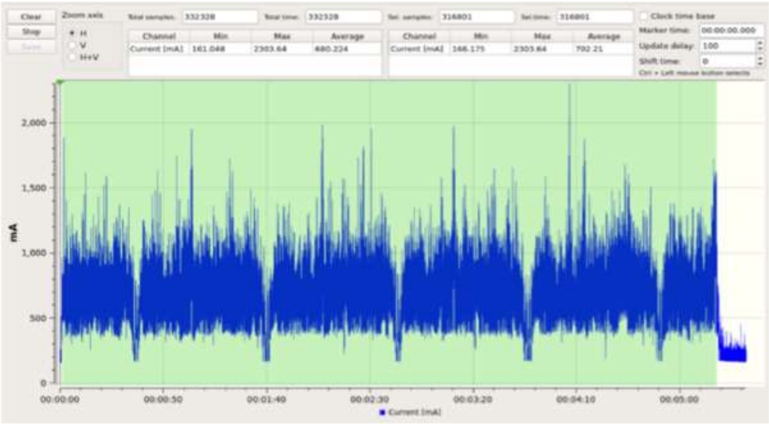
Wrap-up

Test: Youtube playback / power

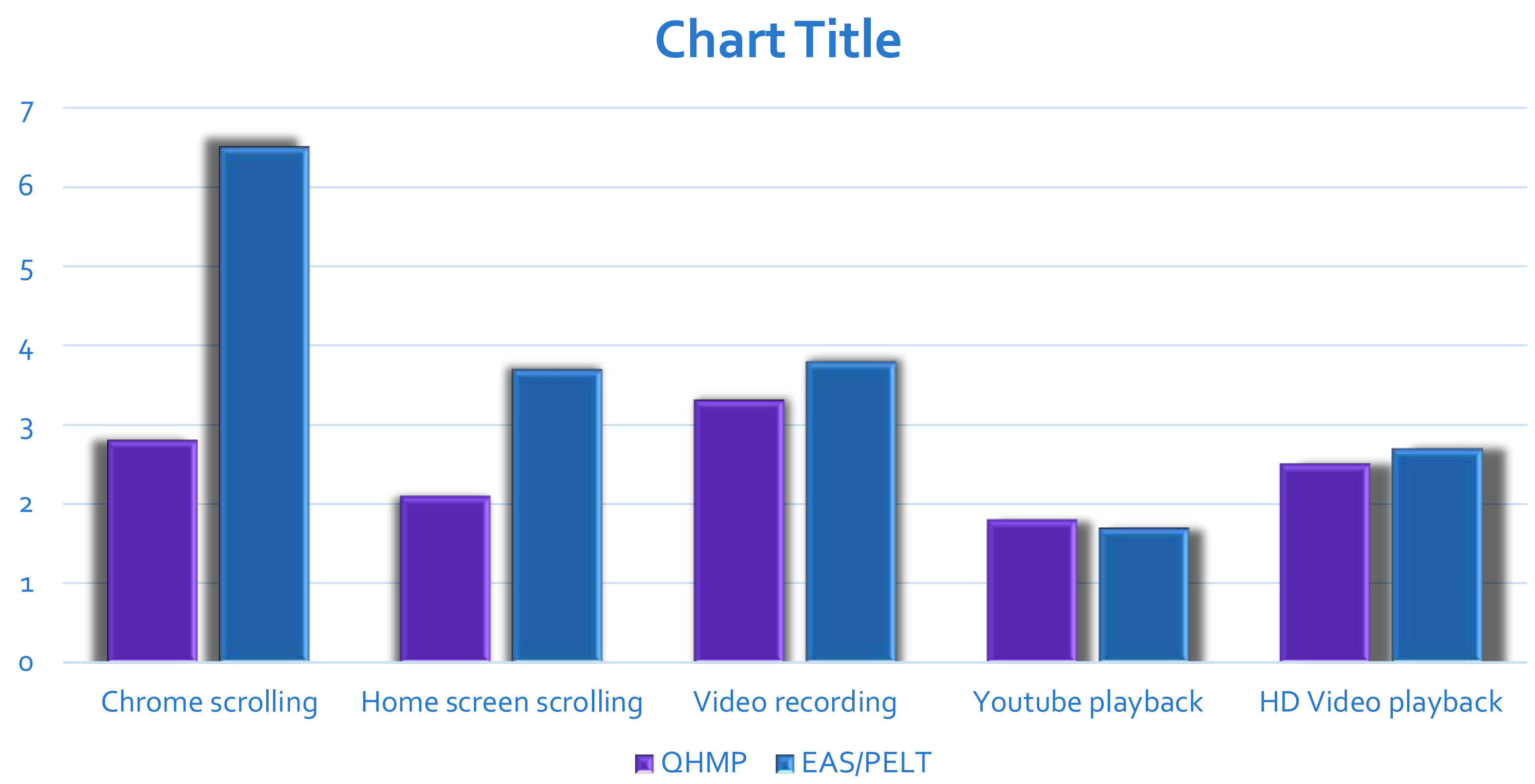
EAS/PELT: 561 mA



QHMP: 680 mA



Test: frame drops per sec.



Result interpretation

- EAS works best with a steady load
  - Excellent power consumption results
  - Good QoS
- EAS doesn't cope well with bursts
  - QoS is lacking
  - Need for frequency boost
    - But then power increases too

QHMP vs EAS/PELT side-by-side

- QHMP has a strong focus on performance
- QHMP is complex and its code is obfuscated
- QHMP is flexible but basically not maintainable
- QHMP doesn't stand a chance of being mainlined

- EAS/PELT is more focused on power conservation
- EAS is based on simple enough principles
- EAS is more predictable and maintainable
- EAS has a chance of being merged into mainline

Introduction

Energy Aware Scheduler

Qualcomm HMP scheduler

Comparisons and outcome

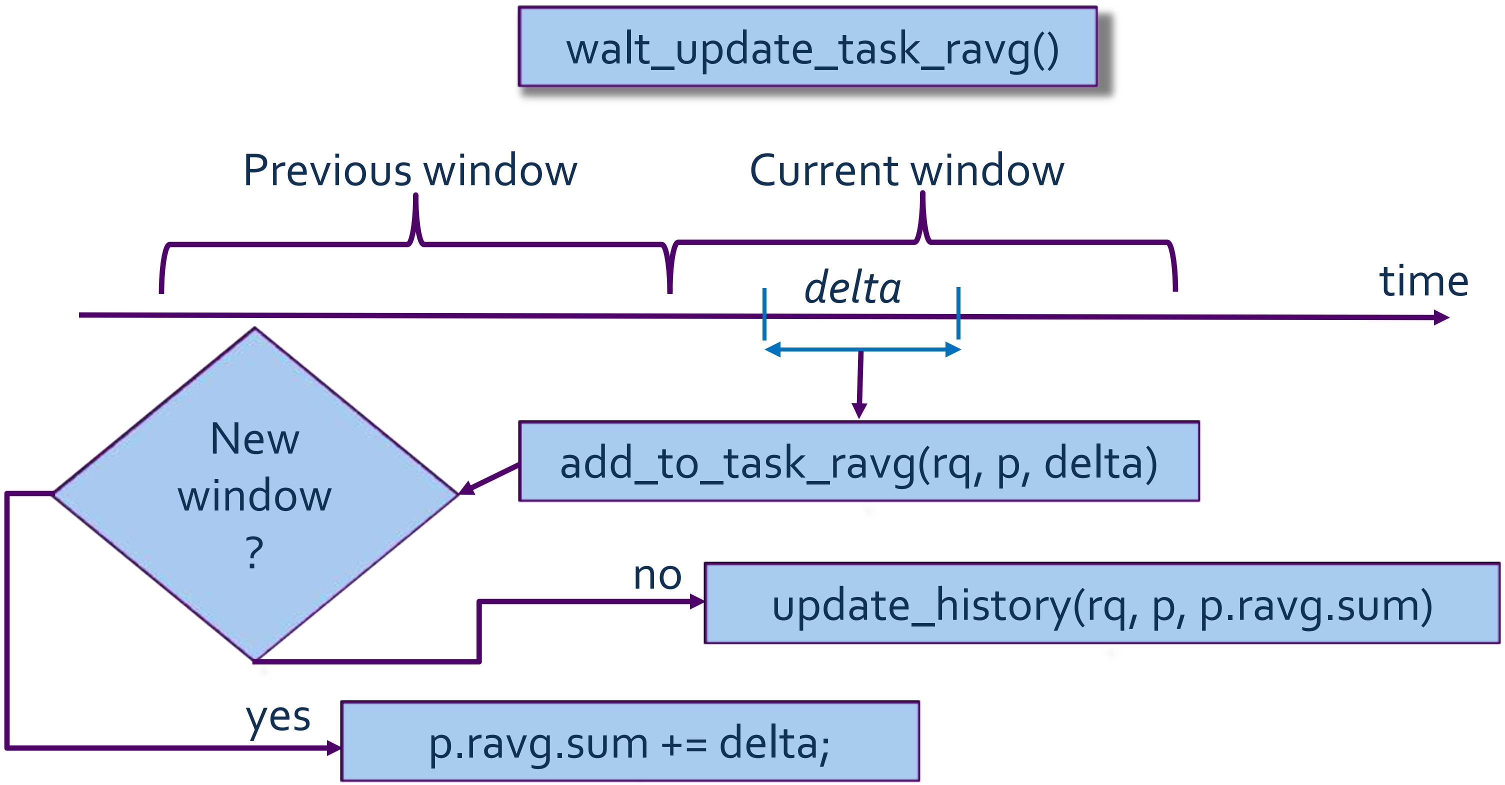
Way forward

Wrap-up

EAS: way forward

- It still made sense to move forward with EAS
  - But turning a blind eye to its deficiencies wouldn't be smart
  - Something had to be done with performance issues
- Use task demand calculation from QHMP for EAS
  - Modularize it and take off the QHMP
- WALT: Window Assisted Load Tracking
  - Retains PELT "per-entity" tracking pattern
  - Implements *N-window* demand calculation from QHMP

WALT: demand contribution calculation



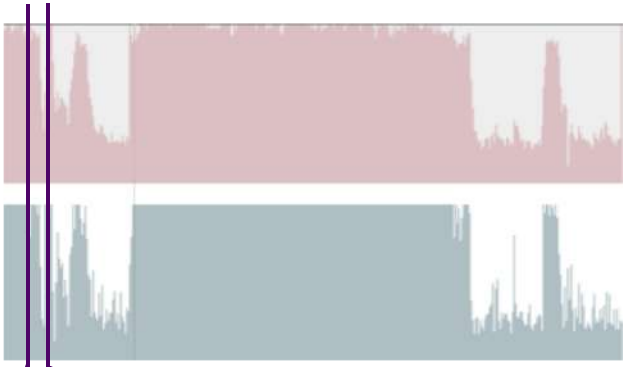
WALT: CPU utilization

- WALT estimates the utilization of CPU by considering the sample measured during the last window.
  - *prev_runnable_sum*
  - So everything happening in the current window's time frame is not affecting the view of utilization
- WALT provides CPU utilization data to CPUFreq governor on demand
- WALT notifies governor about inter-cluster migrations
  - CPUFreq operates on cluster
  - Governor recalculates frequencies for clusters

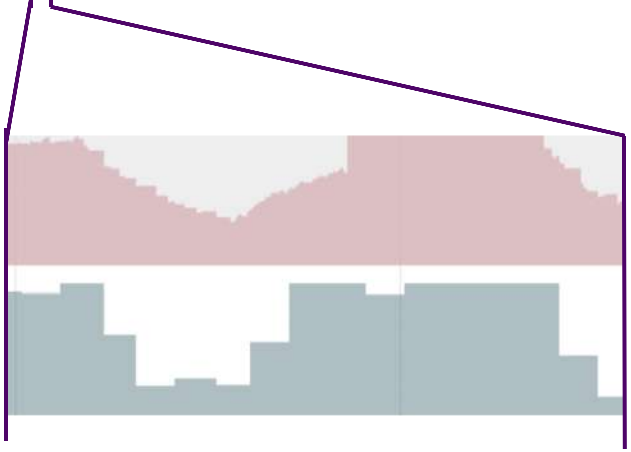
CPU load tracking: PELT vs WALT

EAS/PELT (*util_avg*)

EAS/WALT (*prev_runnable_sum*)



With strong magnification



Result interpretation

- WALT ramps up and down faster
  - Better accuracy for CPUFreq
  - Power consumption may be a concern
- Less need for frequency boosting
  - So in fact power consumption doesn't increase compared to PELT

Introduction

Energy Aware Scheduler

Qualcomm HMP scheduler

Comparisons and outcome

Way forward

Wrap-up

PELT vs WALT summary

Konsulko Group

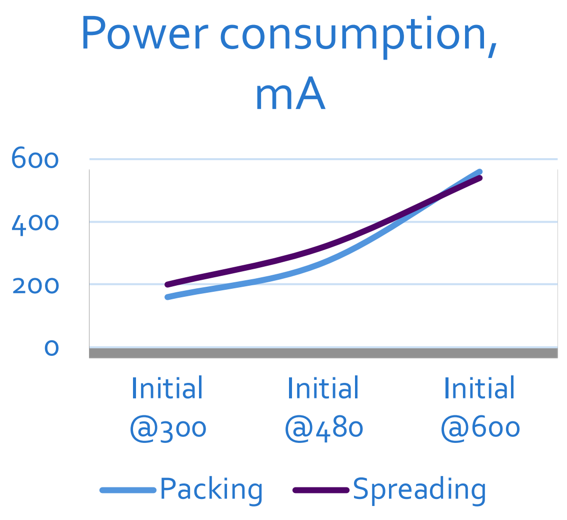
	PELT	WALT
Load tracking	Load is accounted using a geometric series	Load is accounted with a policy that observes past *N* windows
Blocked load/utilization tracking	Load is decayed as part of a runqueue statistic when the task is blocked	Blocked load contribution is removed from runqueue sum/average statistics.
Blocked load restoration	Runqueue statistics include blocked load/utilization at all times	Load contribution is restored to RQ statistics when the task becomes runnable again.

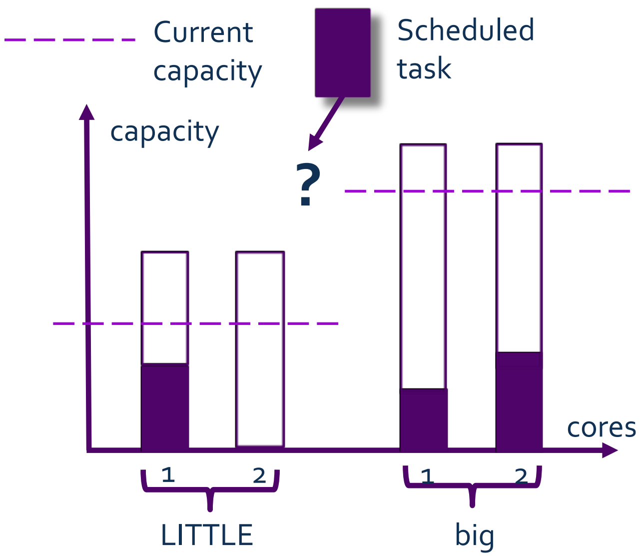
EAS: current status

- WALT became the first choice for EAS
  - Better QoS
- EAS/WALT is effectively EAS + accounting from QHMP
- And that's a *mostly* good thing
  - Convergence
  - *Most* of the good stuff from QHMP got into EAS/WALT
    - E.g. accounting (WALT) got in
    - But: the notion of "small" and "big" task was lost

EAS and task packing

- EAS won’t pack a task if that would mean raising CPU frequency
  - For a small task, keeping an extra CPU awake **may cost more**
- EAS will pack a task even if it would be considered “big”
  - A big task may have to be migrated soon





Conclusions

- big.LITTLE architecture puts high demands on the system software
  - Scheduler has to account for multiple metrics
    - Capacity, power impact
  - DVFS becomes tightly couples with scheduler
- EAS is the most used scheduler for big.LITTLE as of now
- What would the unbiased view on EAS be?
  - it is the best we've got for big.LITTLE scheduling
  - it still has significant shortcomings

Credits

- Uladzislau “Vlad” Rezki <urezki@gmail.com>
  - Help with EAS/QHMP internals
- Anton Ugarov <anton.ugarov@cicknet.pro>
  - Help with testing / measurements
- Tatyana Nekludova
  - Pictures and inspiration
- Maria Wool
  - Inspiration and patience

Questions?

Vitaly.Wool@konsulko.com