Linaro connect
Bangkok 2016

sched-freq

integrating the scheduler and cpufreq

Presented by

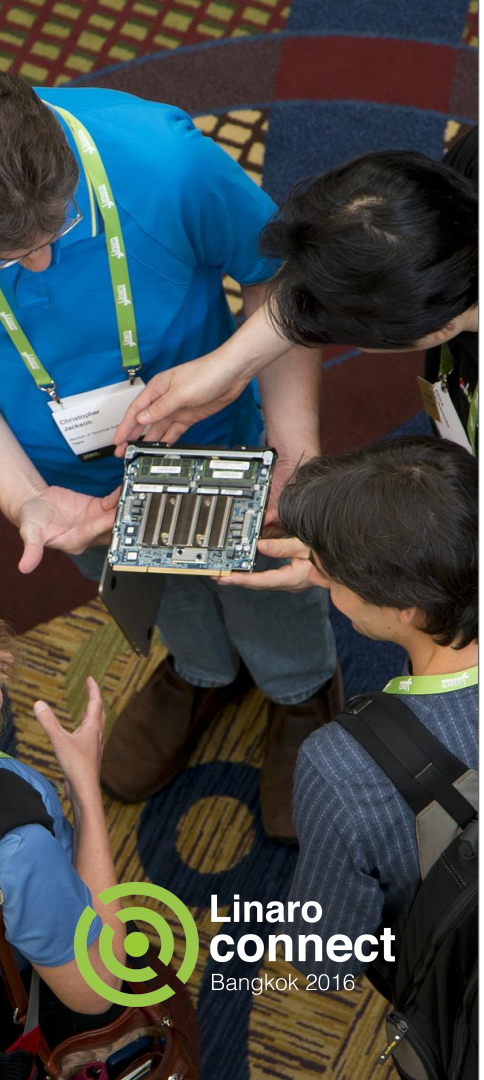
Steve Muckle

Date

BKK16-104 March 7, 2016

Event

Linaro Connect BKK16



outline

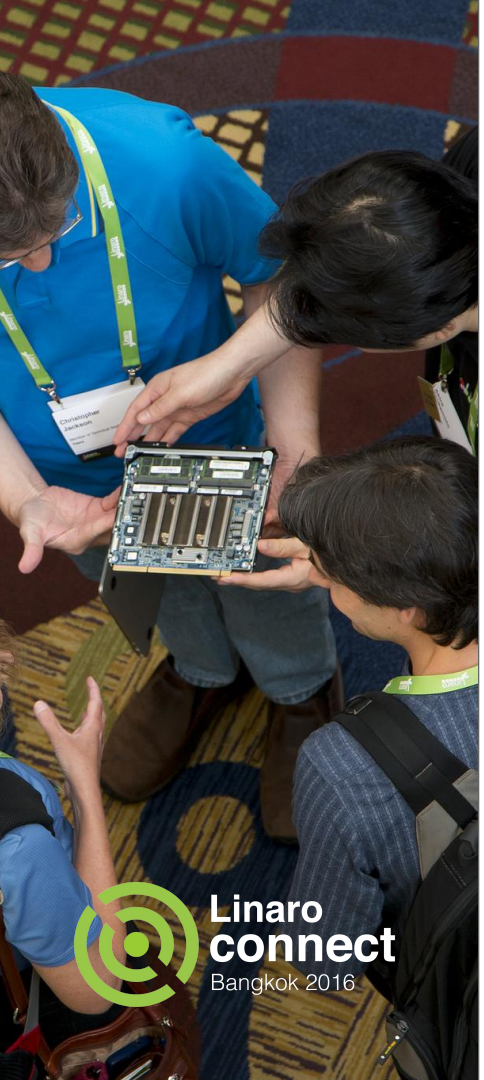
- how things (don’t) work today
- schedfreq design
- latest test results
- an upstream surprise
- next steps

clarifications/quick questions here

proposals and debate in hacking session

Tuesday 8th March, 15:00-15:50

HACKING-2 (lobby lounge - 23rd floor)



outline

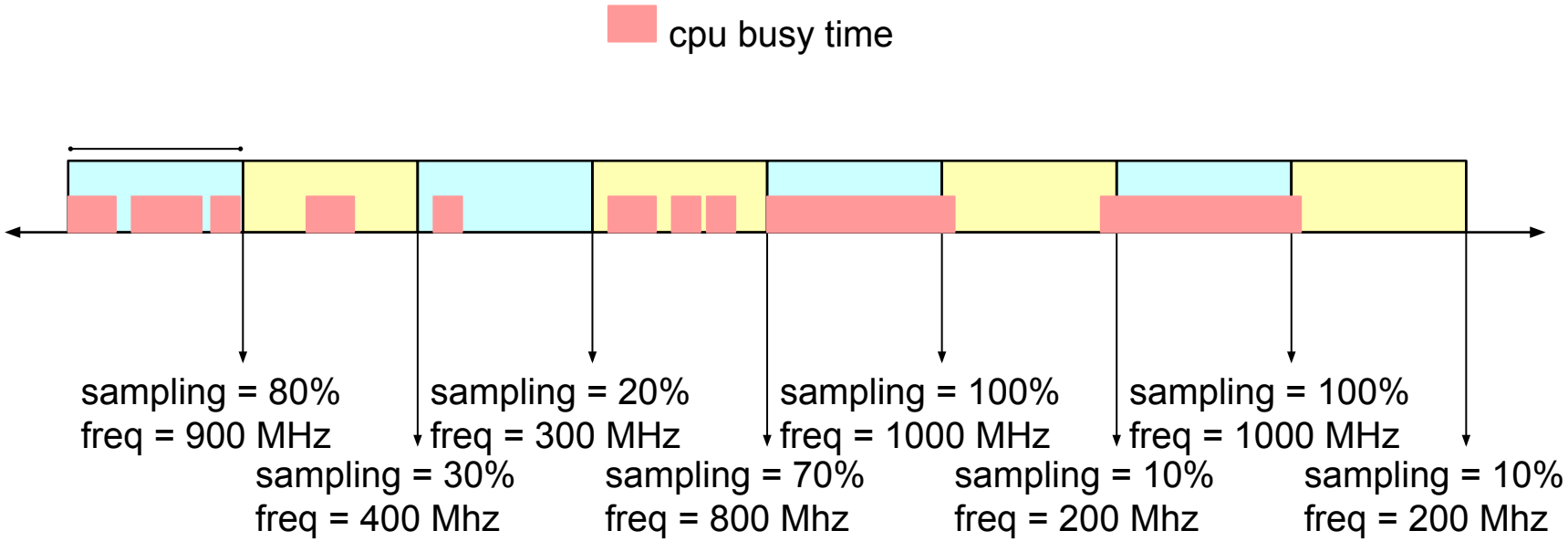
- how things (don’t) work today
- schedfreq design
- latest test results
- an upstream surprise
- next steps

how does cpufreq currently work?

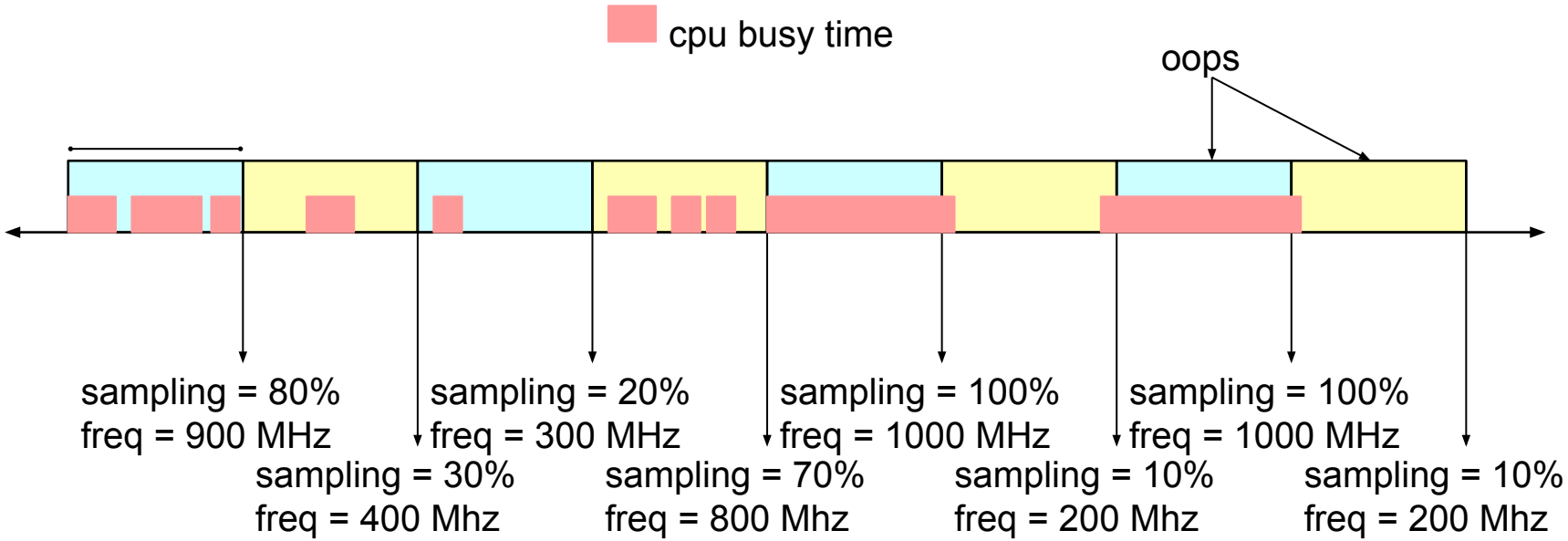
- plugin architecture (governors)
- popular governors are sampling-based

- let's assume:
  - fmin = 100MHz, fmax = 1000Mhz
  - a policy which goes to util*fmax + 100Mhz

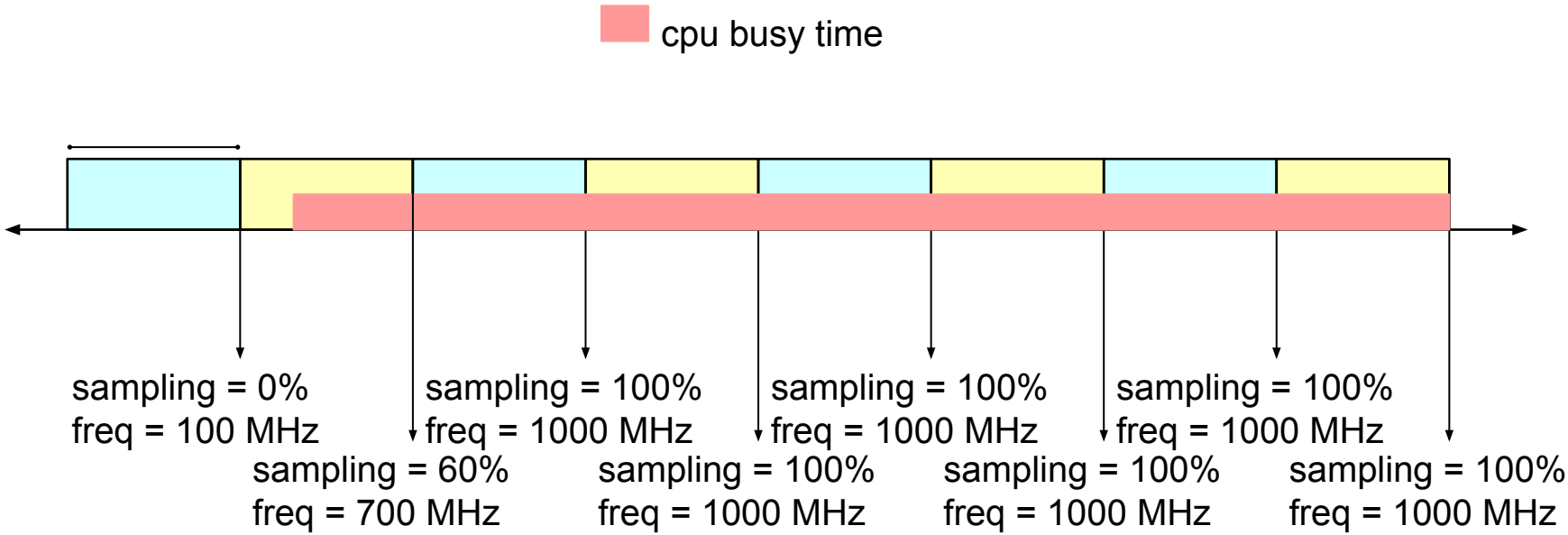
cpufreq governor sampling



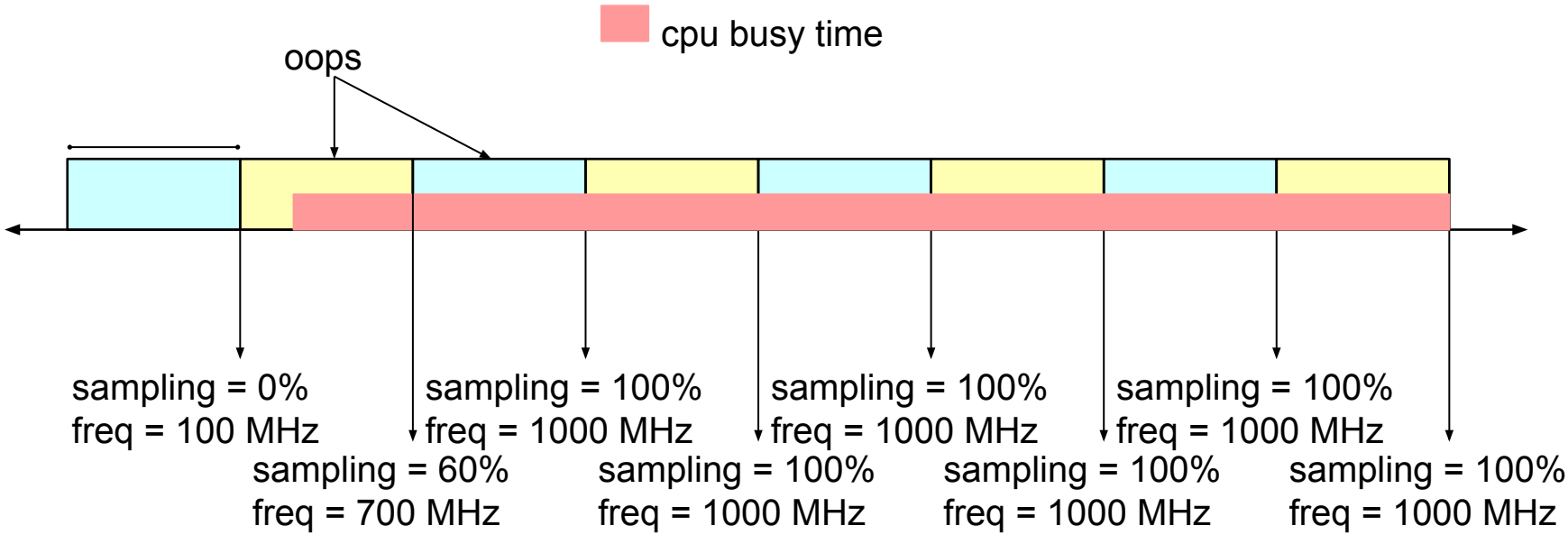
cpufreq governor sampling



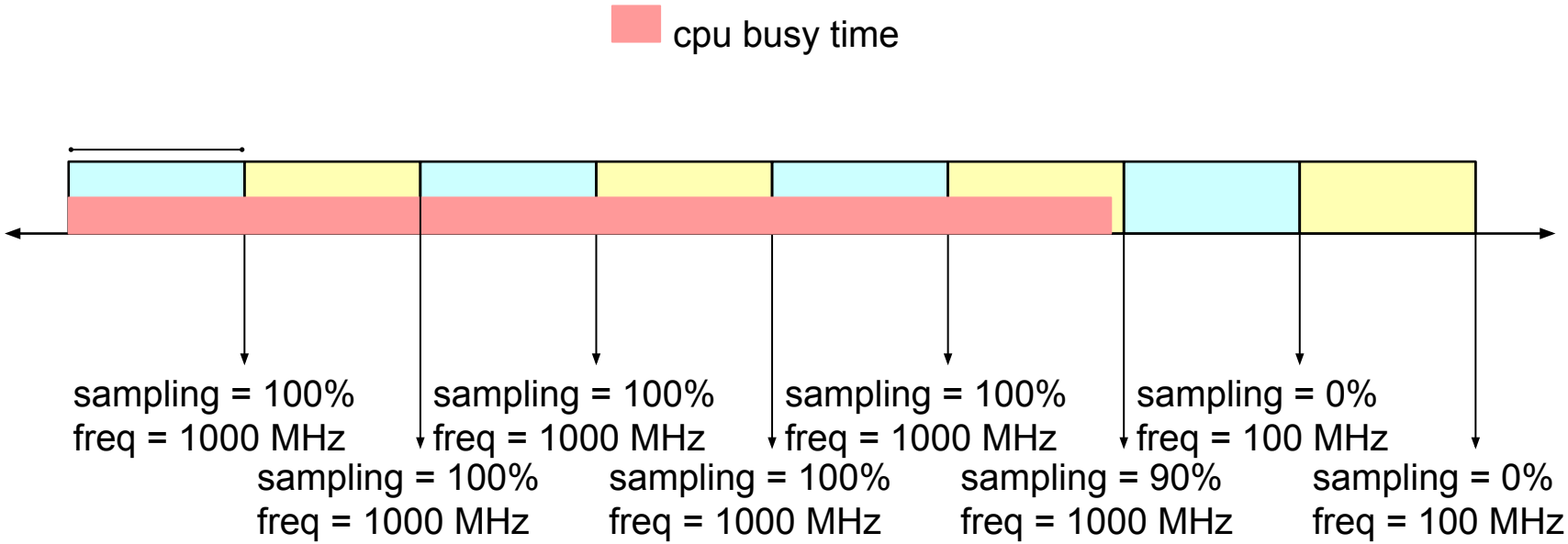
more problems: new tasks



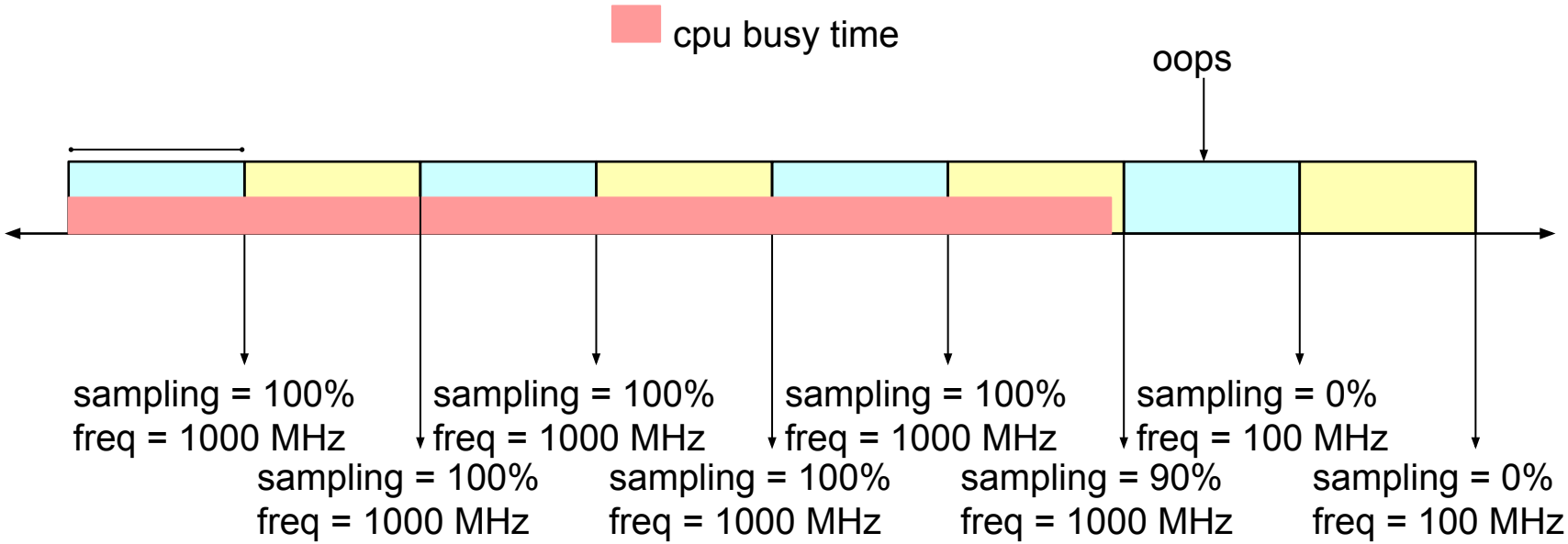
more problems: new tasks



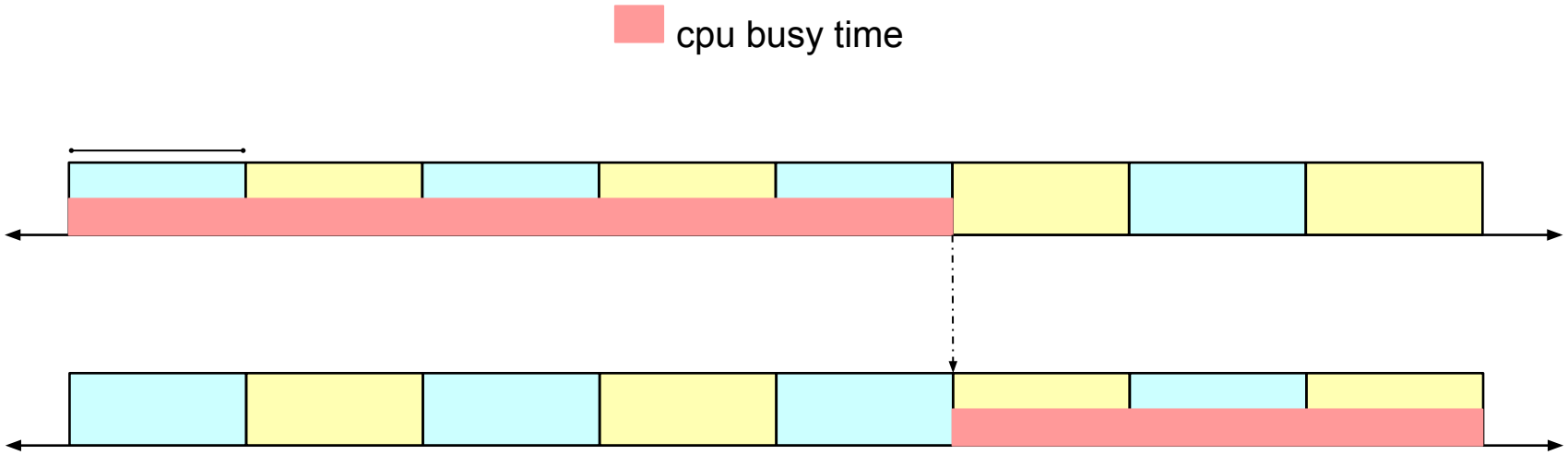
more problems: exiting tasks



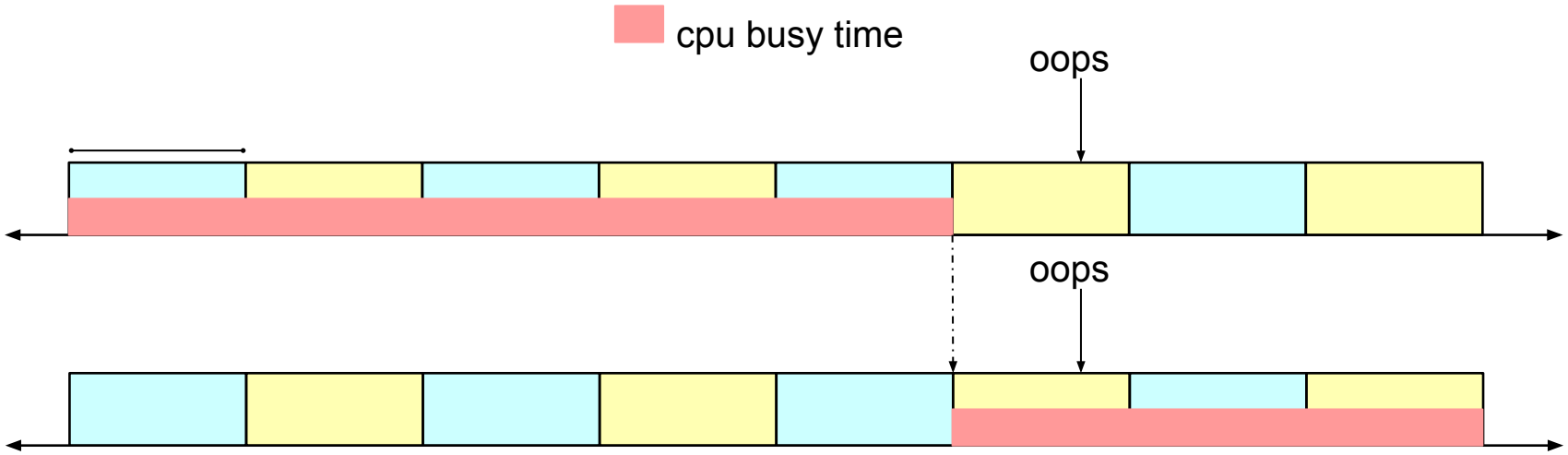
more problems: exiting tasks



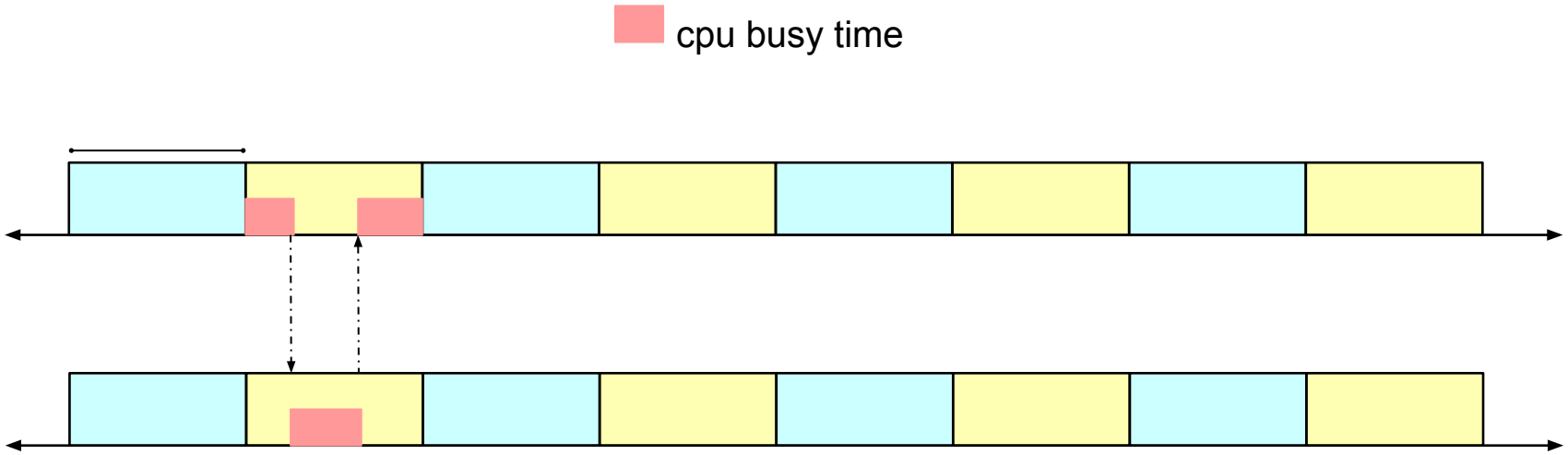
more problems: task migration



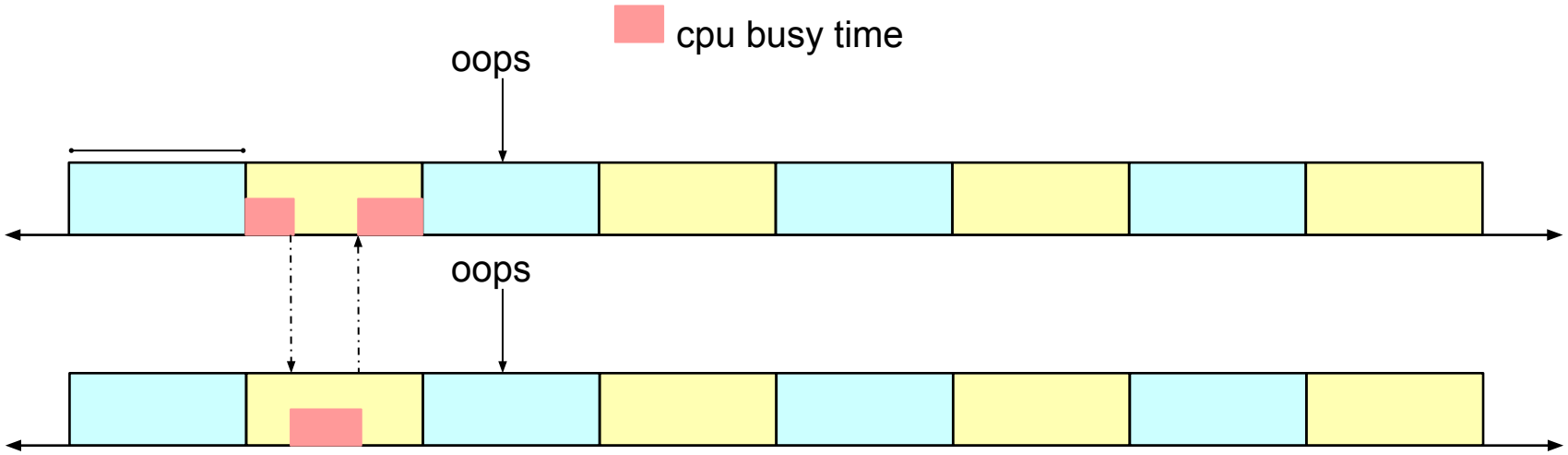
more problems: task migration



more problems: task migration



more problems: task migration



more problems: tuning

- ondemand has 7 knobs
- interactive has 11 knobs

- ...and more get added by OEMs
  - along with hacks

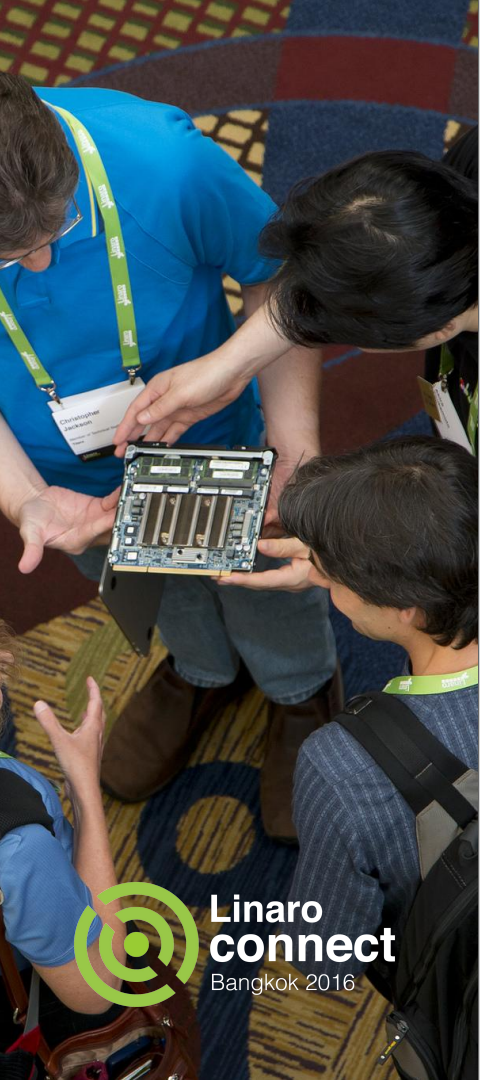
the line in the sand

Ingo Molnar, May 31 2013

Note that I still disagree with the whole design notion of having an "idle back-end" (and a 'cpufreq back end') separate from scheduler power saving policy…

This is a "line in the sand", a 'must have' design property for any scheduler power saving patches to be acceptable…

https://lwn.net/Articles/552889/

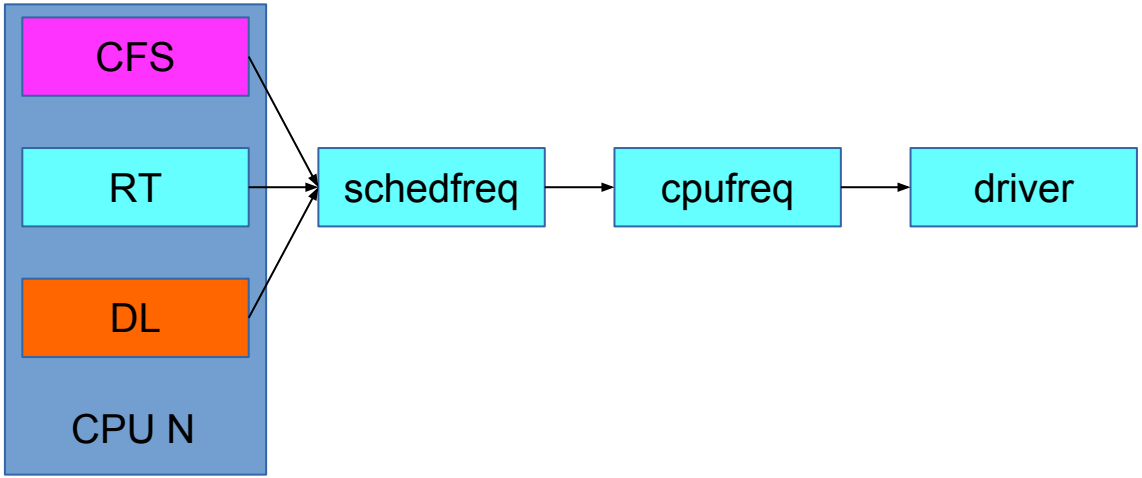


outline

- how things (don’t) work today
- schedfreq design
- latest test results
- an upstream surprise
- next steps

schedfreq design

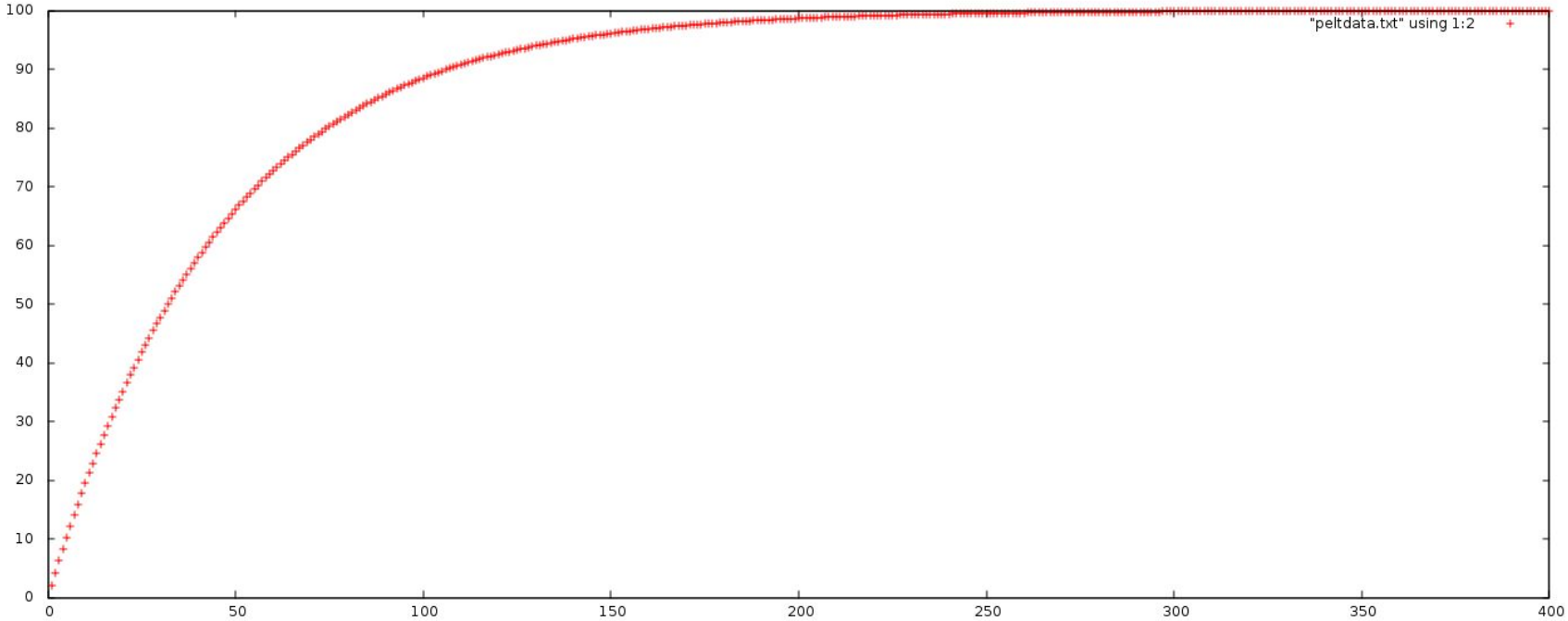
- schedfreq is a cpufreq governor
- can cpufreq be removed?



estimating CFS capacity

- per-entity load tracking (PELT)
  - introduced in 3.8
- exponential moving average
  - sum = is_running() + sum*y
- frequency invariance required
  - partially merged (core bits in, ARM support not)
- microarch invariance required
  - partially merged (core bits in, ARM support not)

estimating CFS capacity - PELT



estimating CFS capacity - PELT

- initial task load
  - was 0 when task is fork-balanced to different CPU due to bug
  - fix is on mainline/4.5
  - http://thread.gmane.org/gmane.linux.kernel/2106780/
- blocked load is included in util_avg

estimating DL capacity

- runtime utilization tracking not strictly required
- DL tasks have runtime, deadline, and period parameters
  - this describes the task's bandwidth reservation
- util = runtime/period
- track DL bandwidth admitted into the system

estimating DL capacity

- util = runtime/period has drawbacks
  - it’s worst case
  - it’s always there
- better solution - track active utilization
  - related to bandwidth reclaiming
- both solutions under discussion

estimating RT capacity

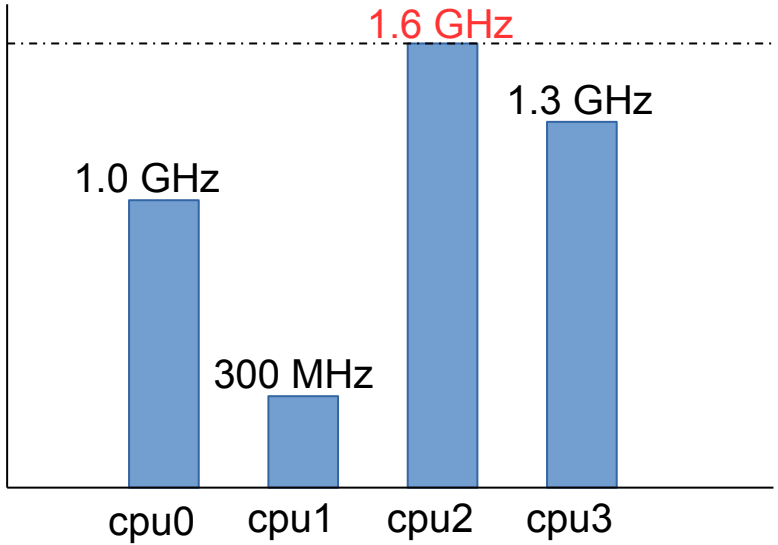
- task priority but no constraints
- monitor RT utilization
  - use rt_avg
  - no way to react to short latency constraints
  - focus on long term constraints and soft real time
  - may not be optimal but it already exists
- be sure to budget capacity for RT
  - do not steal from other class's cap requests

aggregation of sched classes

- sched class capacities are summed
- headroom added to CFS and RT
  - (CFS + RT) * 1.25
- no headroom for DL tasks
  - DL tasks have precise capacity parameters
- total capacity converted to frequency
  - scale using policy->max

aggregation of CPUs

- CPU with max request in freq domain drives frequency



setting the frequency

- tricky to do from hot scheduler paths
  - locking
  - performance implications
  - varying CPU frequency driver behavior
    - does driver sleep?
    - is driver slow?

setting the frequency - fast path

- target freq can always be calculated
- if…
  - driver isn't slow or sleeps AND
  - schedfreq isn't throttled AND
  - a freq transition isn't underway AND
  - the slow path isn't active

  then we can set freq in the fast path

setting the frequency - slow path

- kthread spawned by schedfreq
- safe to sleep
- safe to do more work

setting the frequency - slow path

- kthread spawned by schedfreq
- safe to sleep
- safe to do more work

but

setting the frequency - slow path

- kthread spawned by schedfreq
- safe to sleep
- safe to do more work

but

- task wake overhead ($$$)

locking

- lots of cleanup going on in cpufreq
  - ongoing work from several people
- no blocking locking issues seen

...for now

locking

- sched hooks hold rq lock
  - protect per-CPU data
- avoid accessing policy->rwsem
  - freq_table
  - min/max
  - transition_ongoing
  - not required to initiate freq transitions

locking

- schedfreq has 3 internal locks
- gov_enable_lock (mutex)
  - GOV_START/STOP for static key control
- fastpath_lock (spinlock)
  - solve race to re-evaluate frequency domain
- slowpath_lock (mutex)
  - solve race between slow path, fast path, GOV_START/STOP

scheduler hooks

- enqueue_task_fair, dequeue_task_fair*
  - set CFS capacity
- CFS load balance paths
  - set CFS capacity at src, dest
- scheduler_tick()
  - jump to fmax if headroom is impacted
- pick_next_task_rt, task_tick_rt
  - set RT capacity

scheduler hooks - todo

- DL
- migration paths in kernel/sched/core.c
  - changing task affinity
  - hotplug
  - balance on exec()
  - NUMA balancing

policy summary

- re-evaluate and set freq when tasks
  - wake
  - block (except when CPU goes idle)
  - migrate
- event driven - too many events?

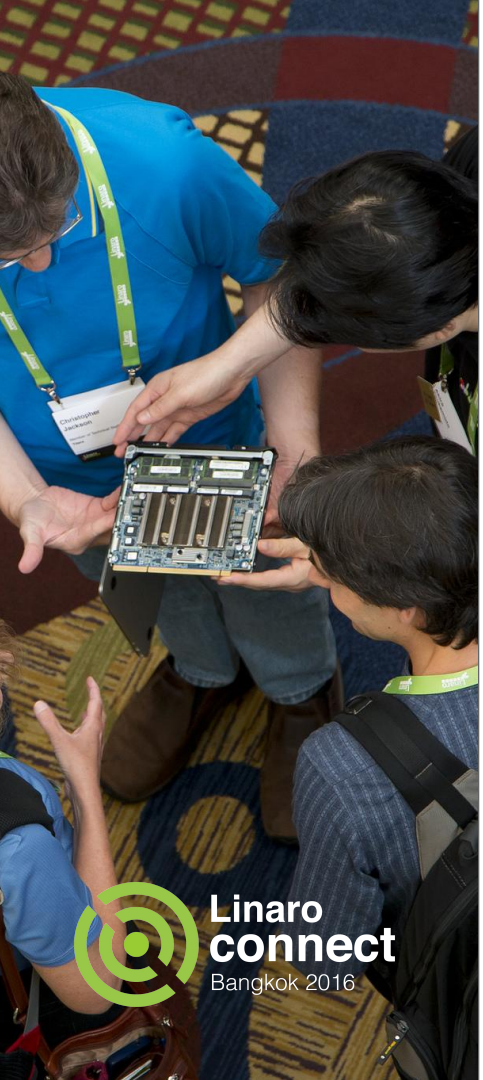
- go to fmax at tick if headroom is impacted

policy summary

- when CPU goes idle, clear vote but don’t re-evaluate/set domain freq
  - don’t initiate more work when going idle
  - right thing to do?

policy summary

- PELT is very important, will need work
- Patrick Bellasi's util_est
  - buffer utilization value to yield more stable estimate
- Vincent Guittot's invariance improvements
  - same amount of work over same period at different freqs => different utilizations
- tuning via schedtune



outline

- how things (don’t) work today
- schedfreq design
- latest test results
- an upstream surprise
- next steps

rt-app cpufreq test case

- simple periodic workload
- each test case uses a different duty cycle
- 16 different test cases/duty cycles
  - either 10, 100 or 1000 loops
  - varies from ~1% to ~43% busy

rt-app cpufreq test case

test case #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
busy (ms)	1	10	1	10	100	6	66	4	40	400	5	50	500	9	90	900
idle (ms)	100	1000	10	100	1000	33	333	10	100	1000	9	90	900	12	120	1200
loops	100	10	1000	100	10	300	30	1000	100	10	1000	100	10	1000	100	10
duration (s)	10.1	10.1	11	11	11	11.7	11.97	14	14	14	14	14	14	21	21	21
busy%	0.99%	0.99%	9.09%	9.09%	9.09%	15.38%	16.54%	28.57%	28.57%	28.57%	35.71%	35.71%	35.71%	42.86%	42.86%	42.86%

rt-app cpufreq test case

- for each loop, record
  - time to execute busy work
  - whether busy work overran period
- for each test case, report
  - average time to complete busy work
  - number of overruns

rt-app cpufreq test case

- define overhead as ...
  (avg_time_test_gov - avg_time_perf_gov) /
  (avg_time_pwrsv_gov - avg_time_perf_gov)

- 0% = completes as fast as perf gov
- 100% = completes as fast as powersave gov

Samsung Chromebook 2

- “Peach Pi”
- Exynos 5800
  - CPUs 0-3: 200 MHz - 2000 MHz A15
  - CPUs 4-7: 200 MHz - 1300 MHz A7
- A15 fmax 1800 MHz with most recent clock support
- no power numbers yet

Samsung Chromebook 2
SCHED_OTHER (CFS) Perf

			ondemand		interactive		sched	
run (ms)	idle (ms)	loops	OR	OH	OR	OH	OR	OH
1	100	100	0	62.07%	0	100.02%	0	78.49%
10	1000	10	0	21.80%	0	22.74%	0	72.56%
1	10	1000	0	21.72%	0	63.08%	0	52.40%
10	100	100	0	8.09%	0	15.53%	0	17.33%
100	1000	10	0	1.83%	0	1.77%	0	0.29%
6	33	300	0	15.32%	0	8.60%	0	17.34%
66	333	30	0	0.79%	0	3.18%	0	12.26%
4	10	1000	0	5.87%	0	10.21%	0	6.15%
40	100	100	0	0.41%	0	0.04%	0	2.68%
400	1000	10	0	0.42%	0	0.50%	0	1.22%
5	9	1000	2	3.82%	1	6.10%	0	2.51%
50	90	100	0	0.19%	0	0.05%	0	1.71%
500	900	10	0	0.37%	0	0.38%	0	1.82%
9	12	1000	6	1.79%	1	0.77%	0	0.26%
90	120	100	0	0.16%	1	0.05%	0	0.49%
900	1200	10	0	0.09%	0	0.26%	0	0.62%

Looks mostly good…

Samsung Chromebook 2
SCHED_FIFO (RT) Perf

			ondemand		interactive		sched	
run (ms)	idle (ms)	loops	OR	OH	OR	OH	OR	OH
1	100	100	0	39.61%	0	100.49%	0	99.57%
10	1000	10	0	73.51%	0	21.09%	0	96.66%
1	10	1000	0	18.01%	0	61.46%	0	67.68%
10	100	100	0	31.31%	0	18.62%	0	77.01%
100	1000	10	0	58.80%	0	1.90%	0	15.40%
6	33	300	251	85.99%	0	9.20%	1	30.09%
66	333	30	24	84.03%	0	3.38%	0	33.23%
4	10	1000	0	6.23%	0	12.21%	10	11.54%
40	100	100	100	62.08%	0	0.11%	1	11.85%
400	1000	10	10	62.09%	0	0.51%	0	7.00%
5	9	1000	999	12.29%	1	6.03%	0	0.04%
50	90	100	99	61.47%	0	0.05%	2	6.53%
500	900	10	10	43.37%	0	0.39%	0	6.30%
9	12	1000	999	9.83%	0	0.01%	14	1.69%
90	120	100	99	61.47%	0	0.01%	28	2.29%
900	1200	10	10	43.31%	0	0.22%	0	2.15%

RTavg mechanism not reacting fast enough.
sched_time_avg_ms = 50

MediaTek 8173 EVB

- CPUs 0-1: 507 MHz - 1573 MHz A53
- CPUs 2-3: 507 MHz - 1989 MHz A72
- power measured via onboard TI INA219s

- thanks to Freedom Tan for this data

MediaTek 8173 EVB

SCHED_OTHER (CFS) Perf

			ondemand		interactive		sched	
run (ms)	idle (ms)	loops	OR	OH	OR	OH	OR	OH
1	100	100	0	98.04%	0	100.41%	0	98.04%
10	1000	10	0	34.00%	0	67.68%	0	99.95%
1	10	1000	0	56.32%	0	101.11%	0	100.69%
10	100	100	0	18.31%	0	31.57%	0	100.02%
100	1000	10	0	2.77%	0	6.79%	0	7.72%
6	33	300	0	41.29%	0	100.27%	0	100.28%
66	333	30	0	1.27%	0	10.38%	0	39.31%
4	10	1000	0	21.45%	2	18.90%	6	65.66%
40	100	100	0	1.35%	0	8.16%	0	24.30%
400	1000	10	0	1.02%	0	1.74%	0	6.55%
5	9	1000	0	13.43%	2	14.14%	5	52.51%
50	90	100	0	1.31%	0	1.39%	1	12.62%
500	900	10	0	0.54%	0	1.32%	0	5.21%
9	12	1000	1	7.19%	1	8.59%	3	27.47%
90	120	100	0	0.88%	0	0.75%	1	3.80%
900	1200	10	0	0.16%	0	0.79%	0	2.83%

Trouble with most workloads with run < 100ms.

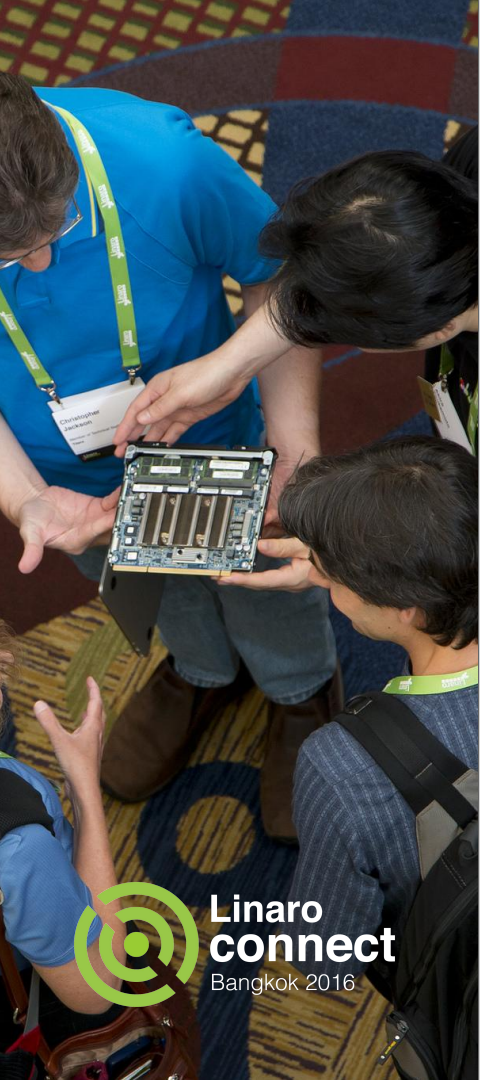
MediaTek 8173 EVB

SCHED_OTHER (CFS) Power/Perf

			power delta		perf (from last slide)
run (ms)	idle (ms)	loops	delta w/ondemand	delta w/interactive	sched overhead
1	100	100	-7.56%	0.01%	98.04%
10	1000	10	-0.41%	1.21%	99.95%
1	10	1000	12.97%	3.92%	100.69%
10	100	100	-3.19%	-4.90%	100.02%
100	1000	10	-7.00%	1.37%	7.72%
6	33	300	-11.84%	-9.95%	100.28%
66	333	30	-8.15%	-2.09%	39.31%
4	10	1000	-0.93%	-8.59%	65.66%
40	100	100	-3.18%	-9.88%	24.30%
400	1000	10	-2.99%	0.07%	6.55%
5	9	1000	1.67%	-9.33%	52.51%
50	90	100	-5.97%	-10.89%	12.62%
500	900	10	-2.29%	1.73%	5.21%
9	12	1000	-5.90%	-9.75%	27.47%
90	120	100	-6.88%	-5.12%	3.80%
900	1200	10	5.23%	6.57%	2.83%

Power savings seen, but not meaningful with observed perf losses.

avg delta w/interactive: -3.48%
avg delta w/ondemand: -2.9%



outline

- how things (don’t) work today
- schedfreq design
- latest test results
- an upstream surprise
- next steps

an upstream surprise

- scheduler - cpufreq hooks posted by Rafael Wysocki
  - Jan 29th 2016
  - now in linux-next
- sched utilization driven gov also posted by Rafael
  - Feb 21st 2016

important differences

- ondemand-like freq algorithm
  - possible to get stuck due to freq invariance
  - weird semantics w.r.t. headroom
- no aggregation of sched class capacities
  - currently goes to fmax for RT, DL
- uses workqueue rather than kthread

what's it mean?

- more engagement from upstream

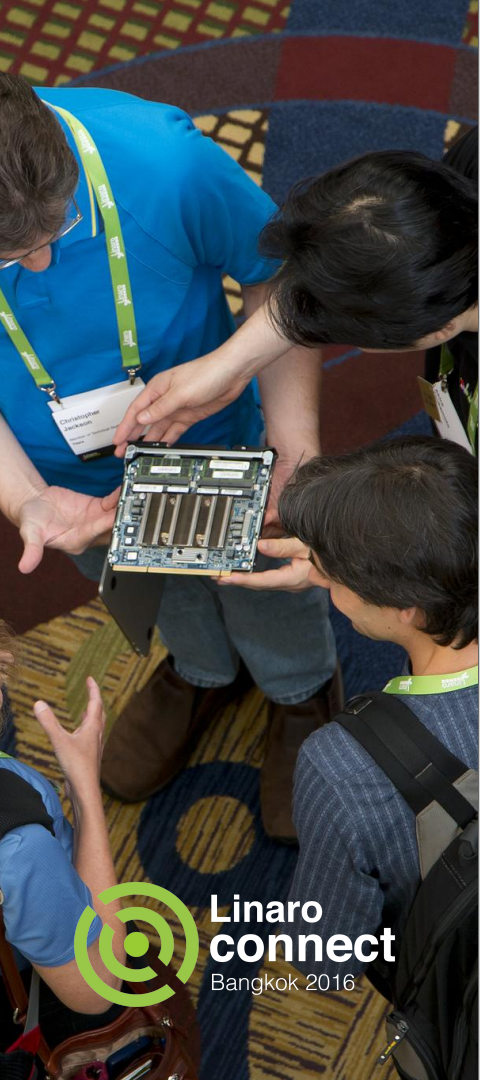
- what's the value of schedfreq?

words of encouragement

“What I'd like to see from a scheduler metrics usage POV is a single central place, kernel/sched/cpufreq.c, where all the high level ('governor') decisions are made.
This is the approach Steve's series takes.”

Ingo Molnar, 03/03/2016

Note: Mike Turquette and Juri Lelli conceived and authored much of the schedfreq series.



outline

- how things (don’t) work today
- schedfreq design
- latest test results
- an upstream surprise
- next steps

next steps

- address shortcomings in schedutil
  - freq algorithm
  - scheduler hooks
  - better RT, DL response
- more in-depth testing and analysis
- experiments with real-world usecases
  - Android UI, games, benchmarks, etc.
- merging with EAS
- integration and testing with schedtune
- window-based load tracking

the end

backup

ondemand

- sample every `sampling_rate` usec
- cpu usage = busy% of last `sample_rate` usec
- if busy% > `up_threshold`, go to fmax
- otherwise scale with load
  - freq_next = fmin + busy% * (fmax - fmin)
- stay at fmax longer with `sampling_down_factor`

interactive

- sample every `timer_rate` usec
- cpu usage = busy% of last `timer_rate` usec
- if busy% > `go_hispeed_load` go to `hispeed_freq`
- otherwise scale CPU according to `target_loads`
  - 85 1000000:90 1700000:95
  - XXX put a note in here to explain this
- can prevent slowdown with `min_sample_time`
- delay speedups past `hispeed_freq` with `above_hispeed_delay`