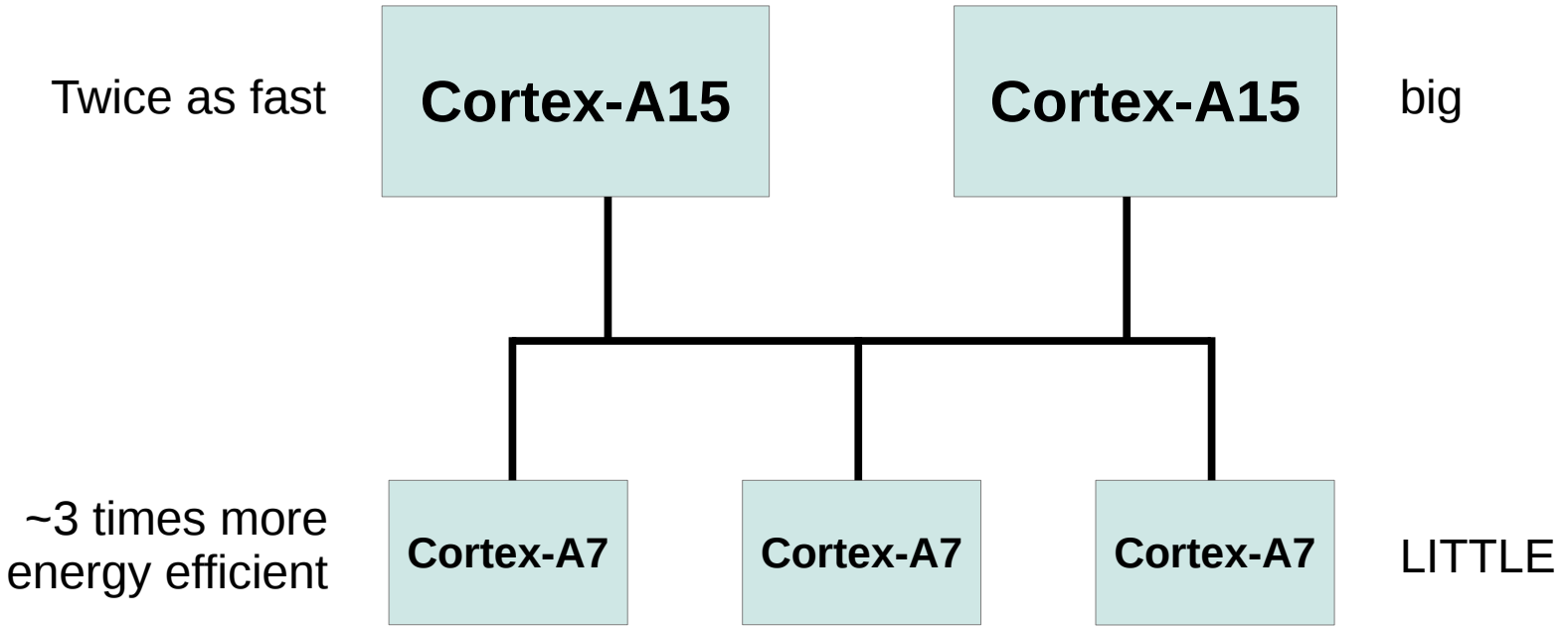
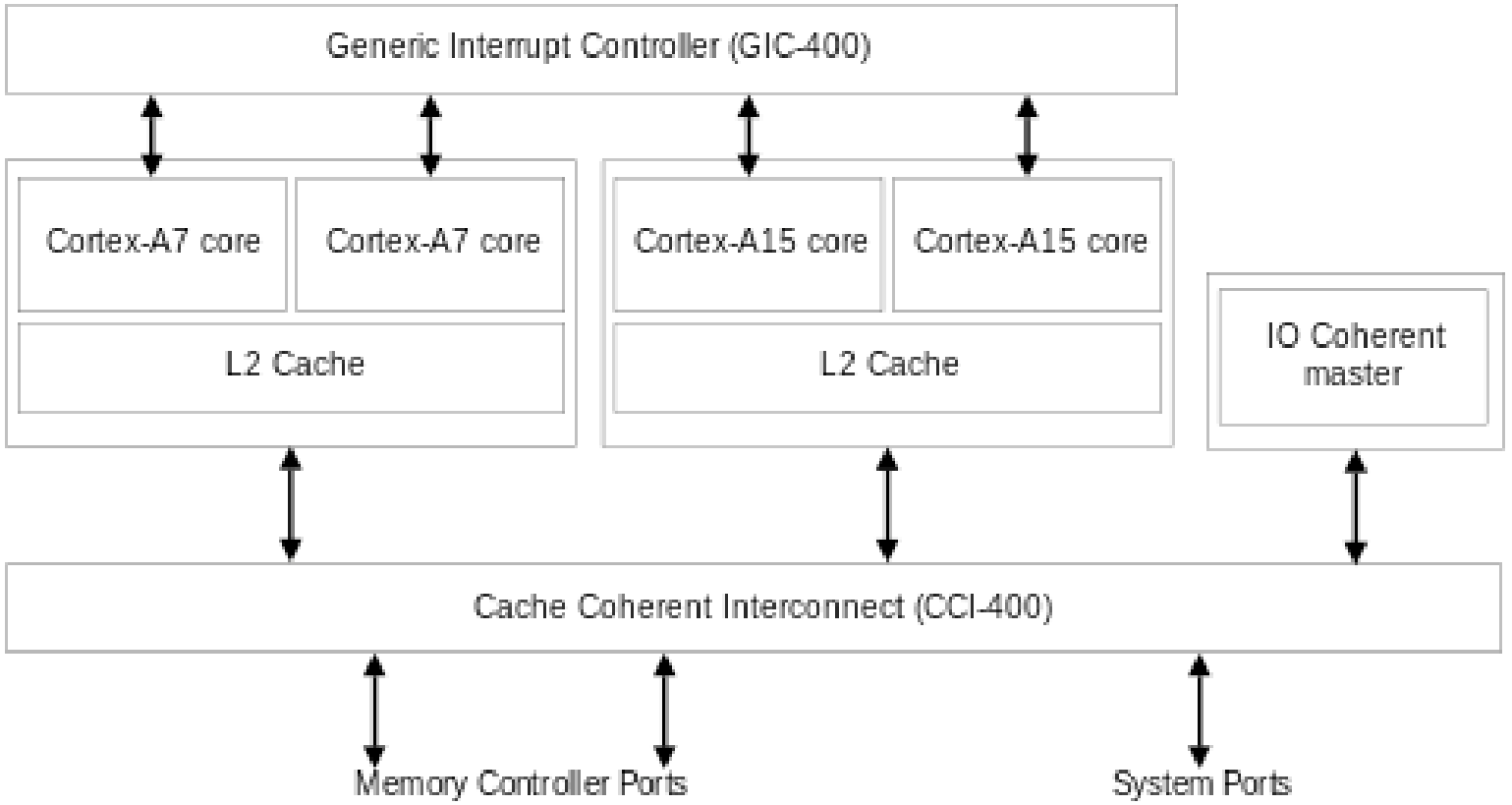
Improving Energy Efficiency On Asymmetric Multiprocessing Systems

Paul E. McKenney (IBM Linux Technology Center assigned to Linaro)
Dietmar Eggeman (ARM Ltd. Cambridge)
Robin Randhawa (ARM Ltd. Cambridge)

ARM big.LITTLE Architecture



ARM big.LITTLE Schematic

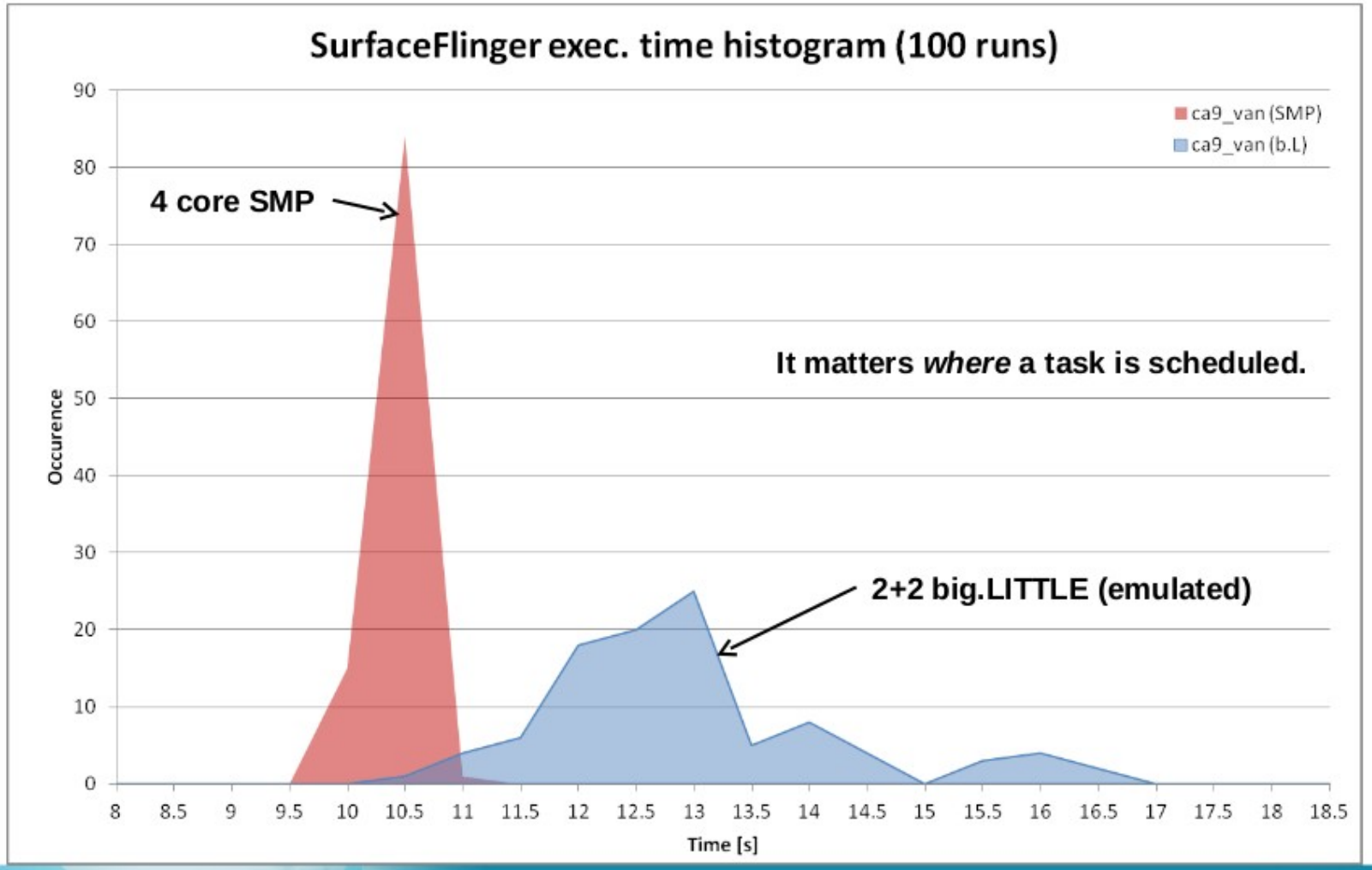


big.LITTLE Architecture: Strategy

- Run on the LITTLE by default
- Run on big if heavy processing power is required
- In other words, if feasible, run on LITTLE for efficiency, but run on big if necessary to preserve user experience
  - Use big CPUs for media processing, rendering, etc.
  - This suggests that RCU callbacks should run on LITTLE CPUs, possibly also for timers and other low-priority asynchronous events
  - Key point: Goal of big.LITTLE scheduling is to distribute tasks *unevenly* to handle different energy-efficiency and performance goals
  - Unlike traditional SMP, it now matters where a task is scheduled

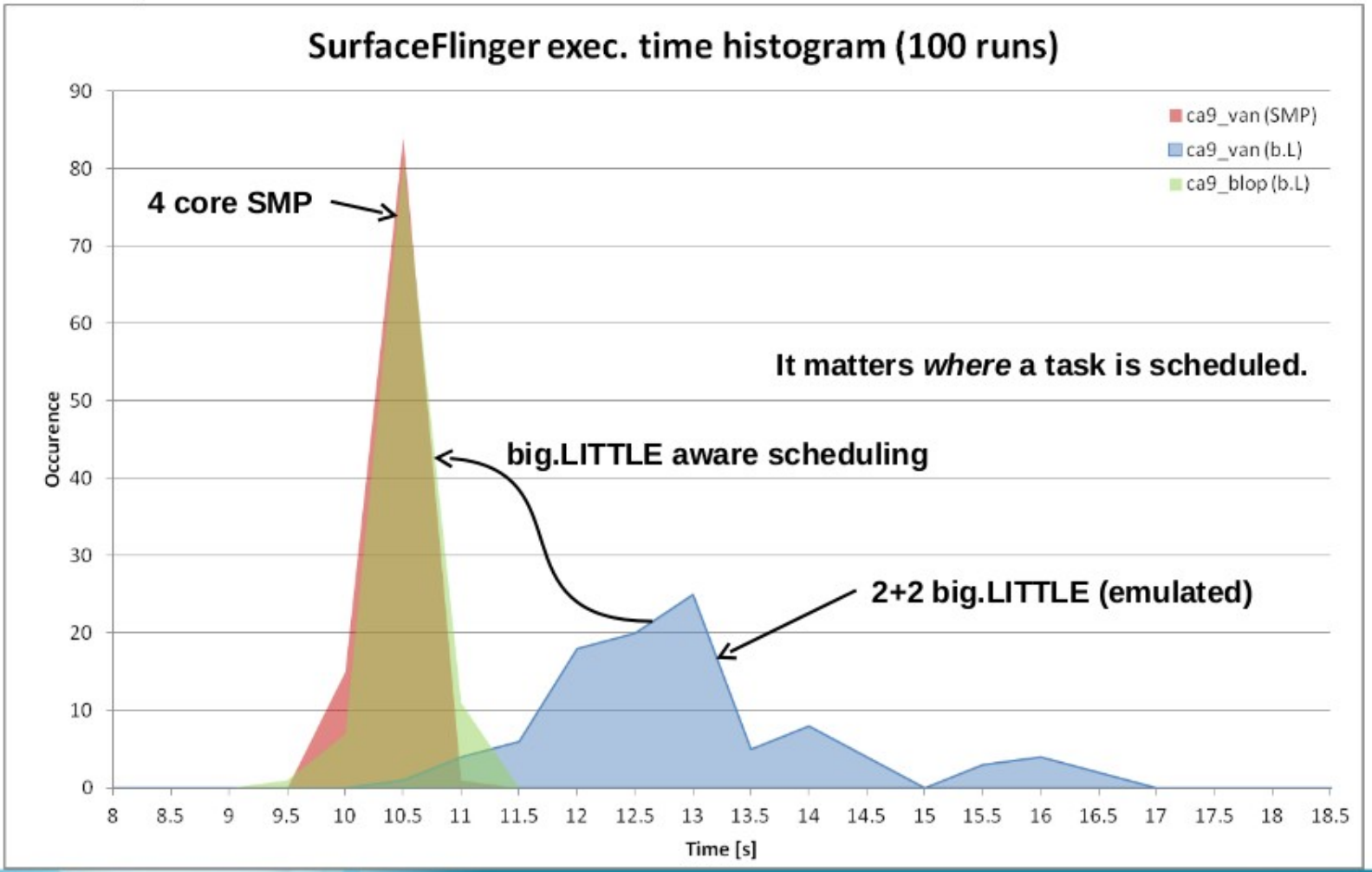
SMP OS (Morten Rasmussen)

- Example: Android UI render thread execution time.



big.LITTLE OS (Morten Rasmussen)

- Example: Android UI render thread execution time.

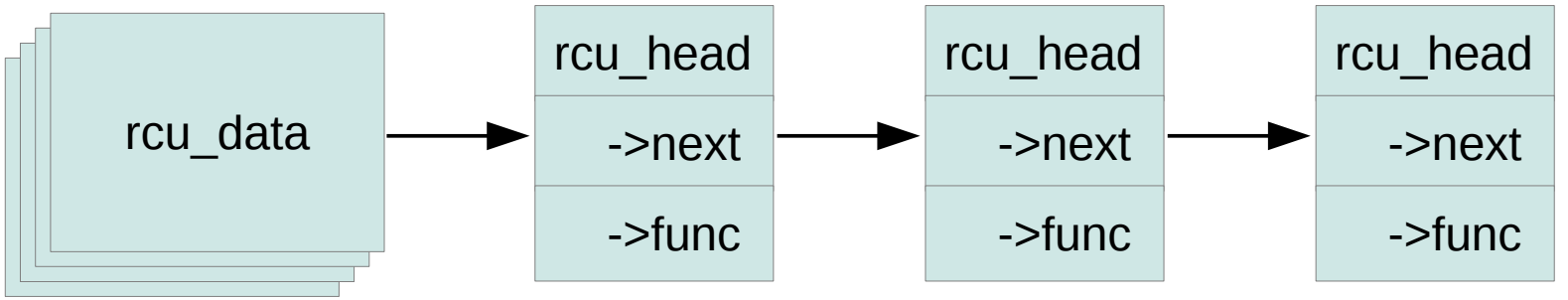


Morten Rasmussen Approach

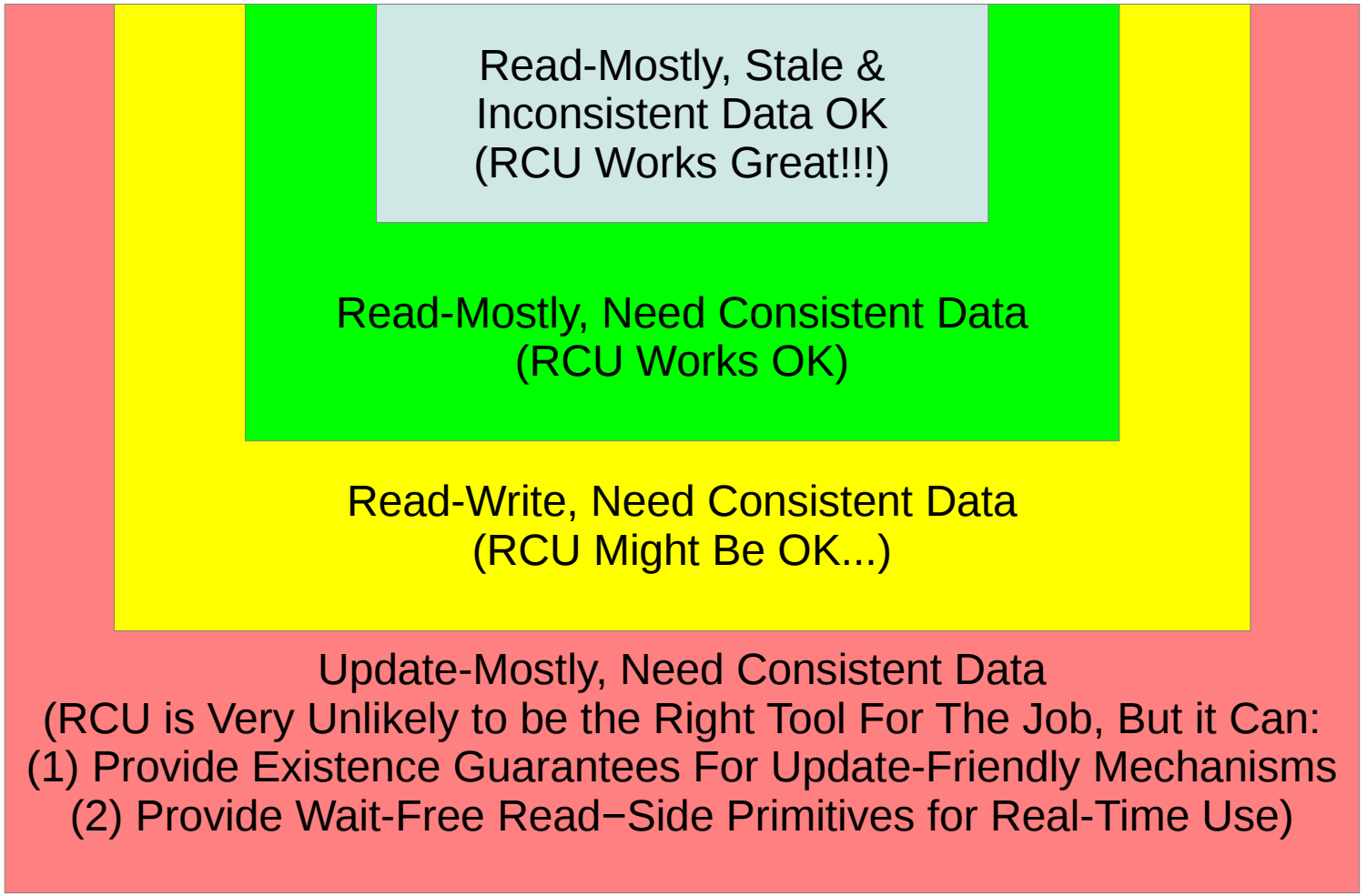
- Leverage Paul Turner's Entity Load Tracking work
- Policy: Keep all tasks on LITTLE cores unless:
  - The task load is above a fixed threshold, and
  - The task priority is default or higher

What Is RCU?
(AKA Read-Copy Update)

- For an overview, see http://lwn.net/Articles/262464/ or http://doi.acm.org/10.1145/2488364.2488549
- For the purposes of this presentation, think of RCU as something that defers work, with one work item per callback
  - Each callback has a function pointer and an argument
  - Callbacks are queued on per-CPU lists, invoked after “grace period”
  - Deferring the work a bit longer than needed is OK, deferring too long is bad (splat after 20 seconds) – but failing to defer long enough is fatal
  - RCU allows extremely fast & scalable read-side access to shared data

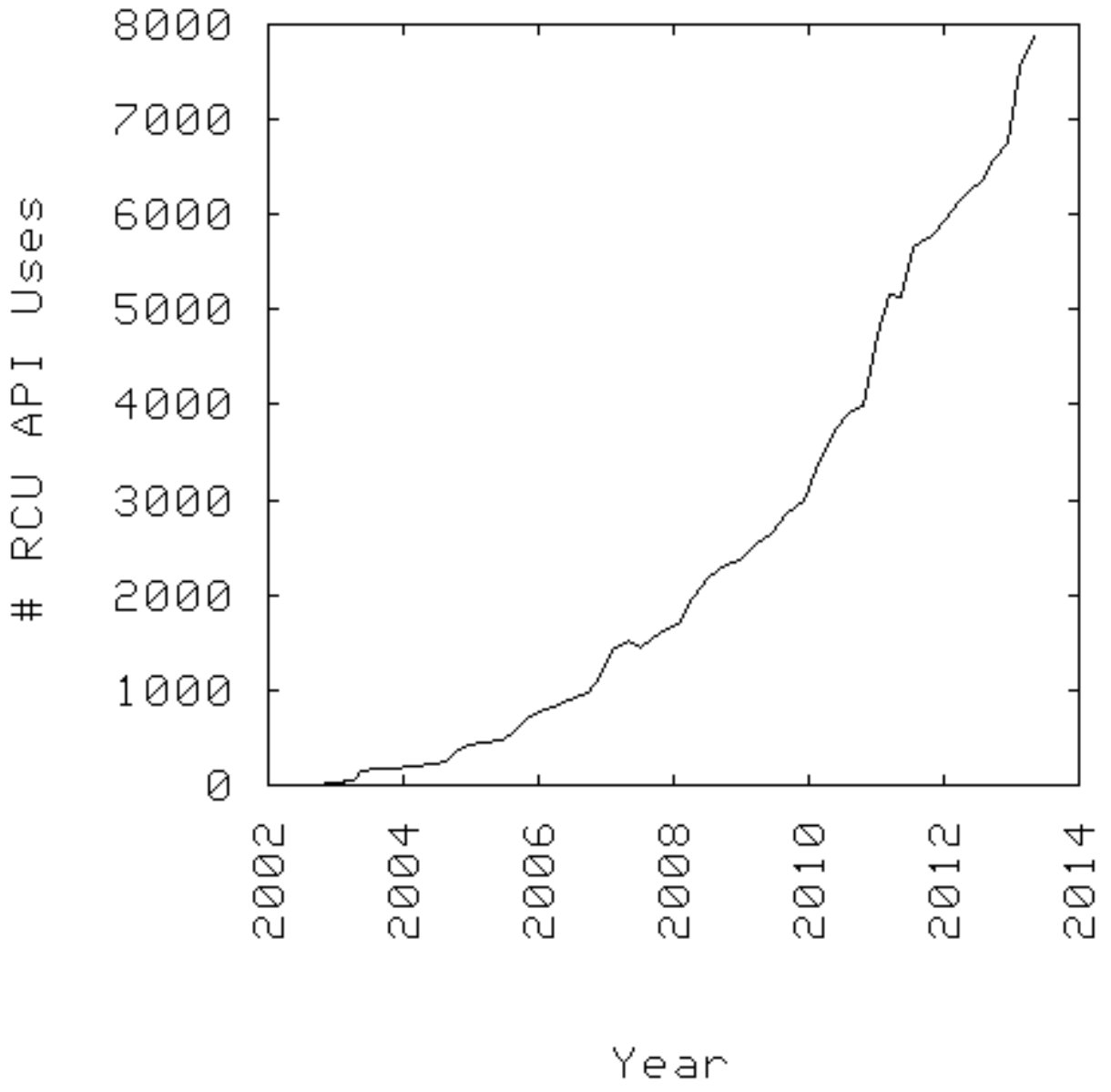


RCU Area of Applicability

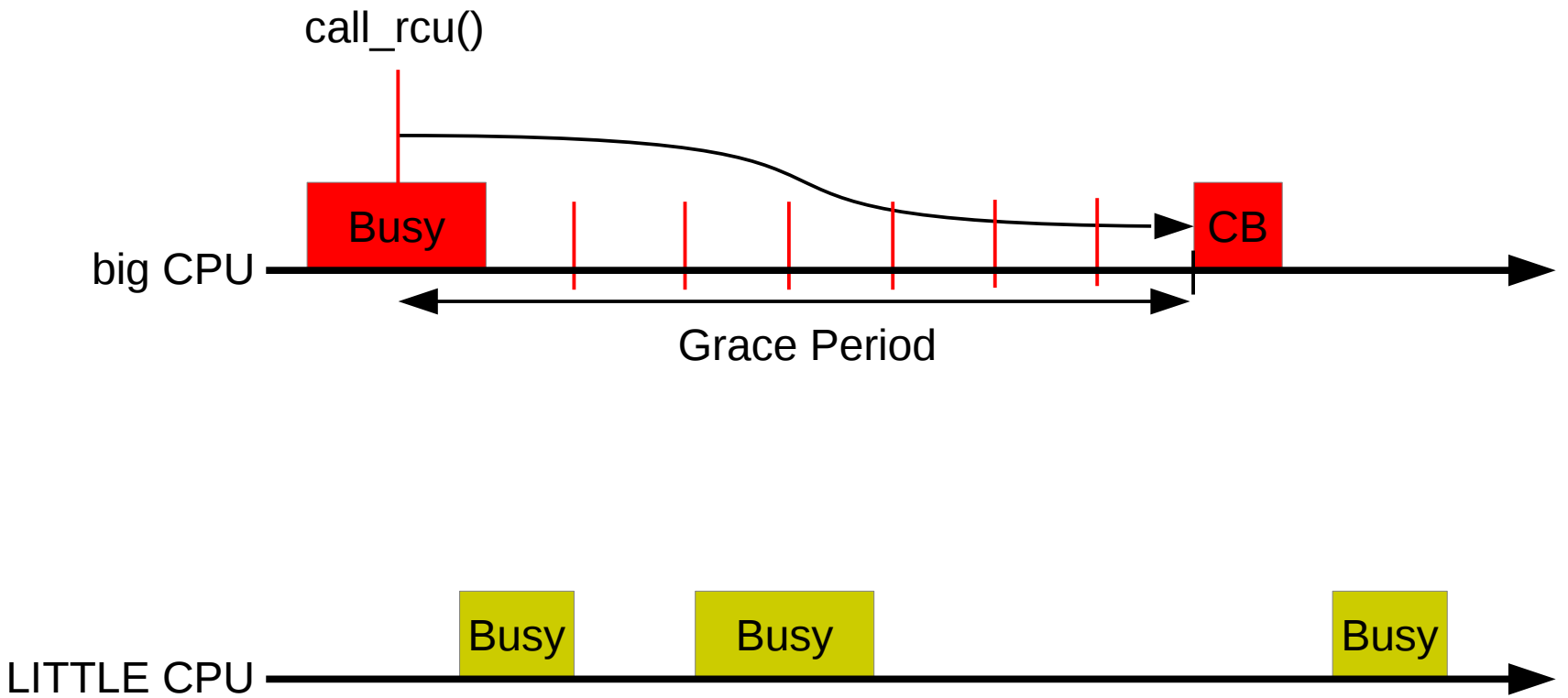


Use the right tool for the job!!!

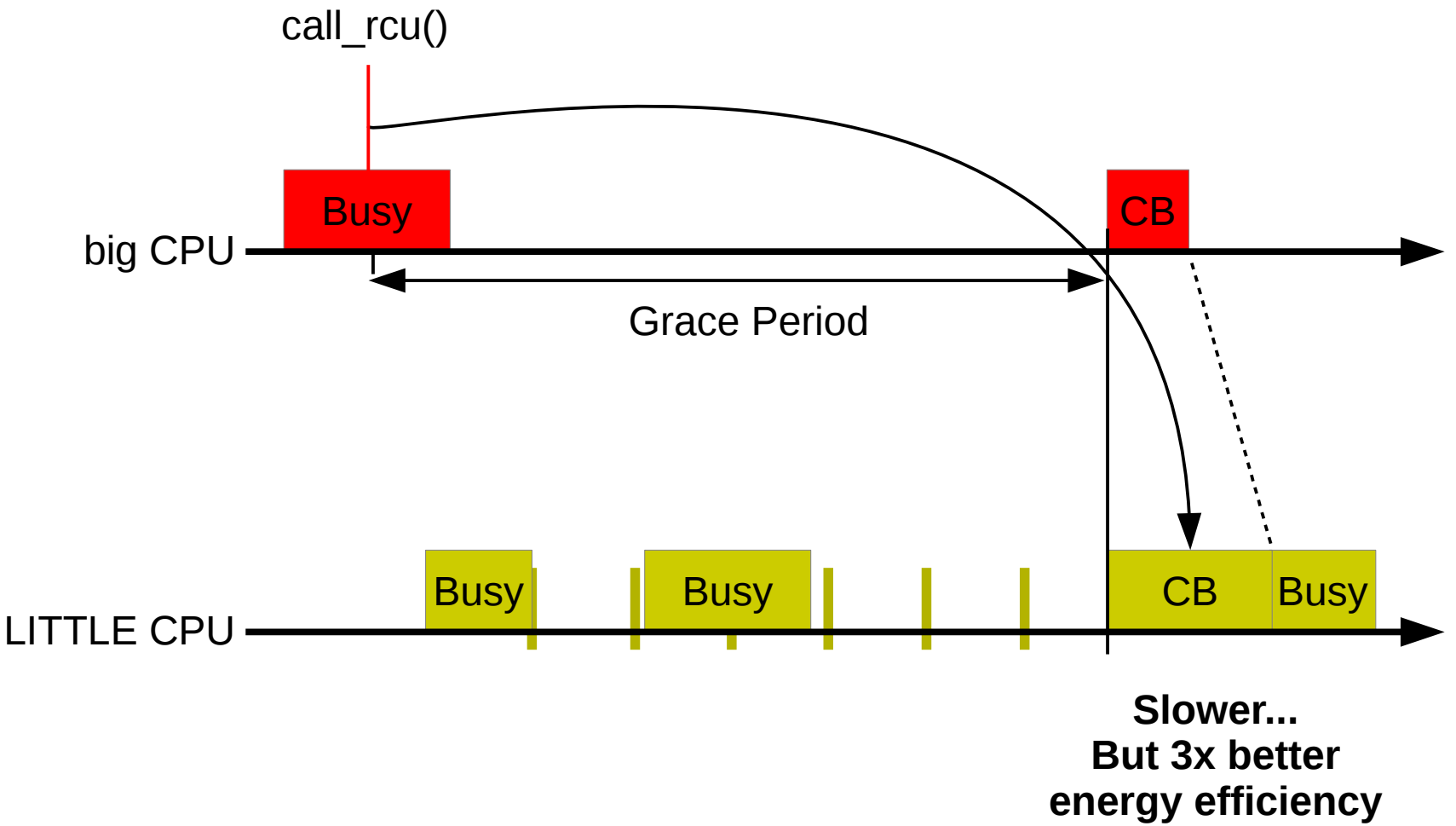
RCU Applicability to Linux Kernel



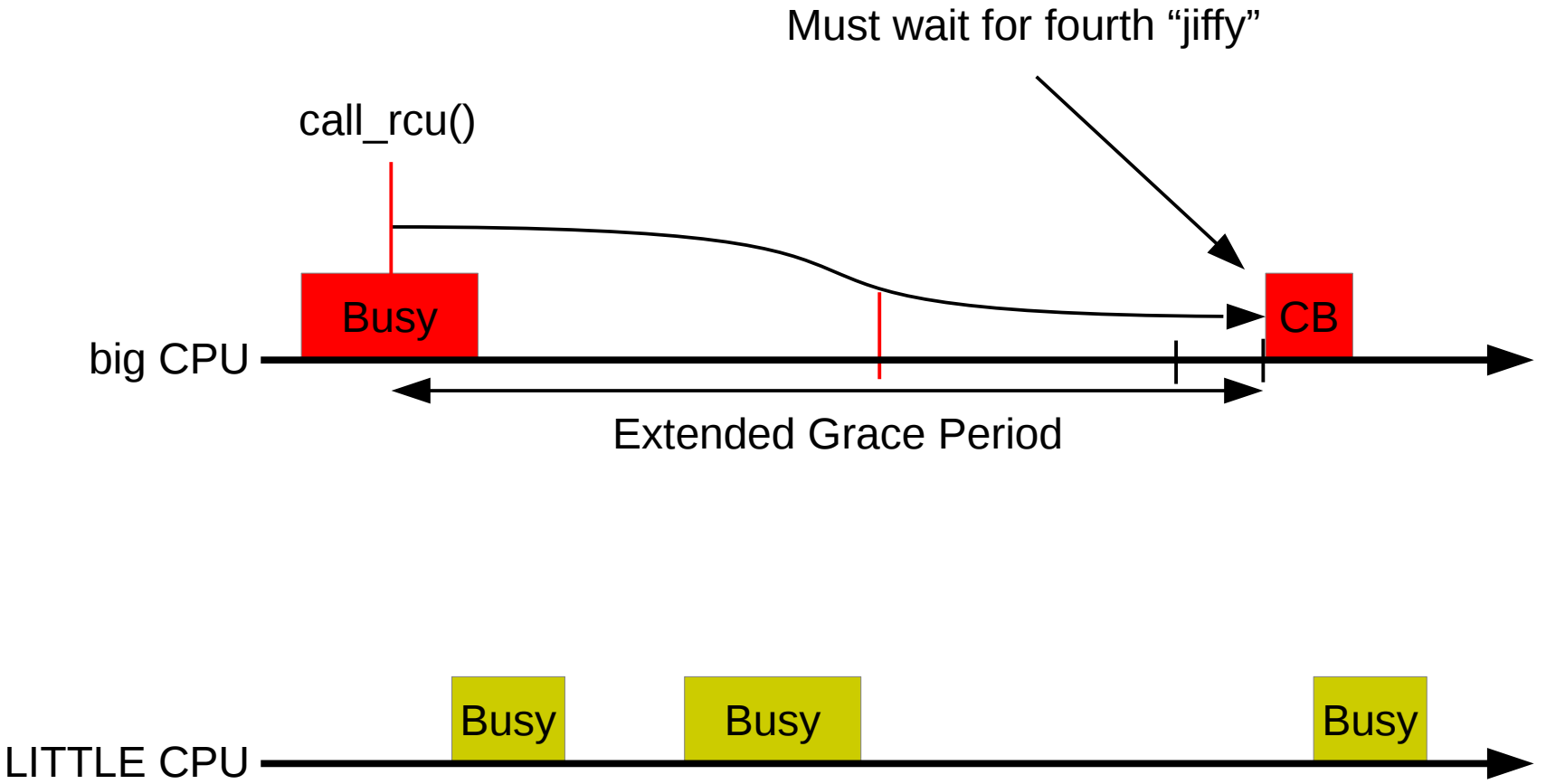
ARM big.LITTLE Without RCU Callback Offloading



1: ARM big.LITTLE With RCU Callback Offloading



2: ARM big.LITTLE With Reduced Wakeups



Results Summary

	Reference		RCU Processing Offload				Enforced Idle			
Benchmark	E (J)	T (s)	Energy	Benefit	Time	Benefit	Energy	Benefit	Time	Benefit
cyclictest	1.75	3.13	1.47	15.98%	3.23	**-3.38%**	1.47	**16.13%**	3.13	**-0.07%**
sysbench	32.68	9.14	31.61	3.29%	8.99	**1.68%**	31.12	4.77%	8.99	**1.70%**
andebench2	59.51	20.54	57.91	2.70%	20.37	**0.83%**	57.99	**2.57%**	21.59	**-5.09%**
andebench8	174.04	45.87	170.37	**2.11%**	46.19	**-0.70%**	166.91	4.09%	46.60	**-1.59%**
audio	7.87	30.01	7.39	**6.05%**	30.03	**-0.04%**	6.28	20.16%	30.02	**-0.04%**
audio+bbench	99.05	156.29	93.02	6.09%	155.58	**0.45%**	86.95	12.21%	160.75	**-2.85%**

Benchmark	Reference		RCU Processing Offload		Enforced Idle	
	Energy	Time	Energy	Time	Energy	Time
cyclictest	1.62	3.12	1.42	3.12	1.40	3.12
	1.68	3.12	1.42	3.12	1.42	3.13
	1.74	3.12	1.46	3.12	1.44	3.13
	1.80	3.12	1.48	3.13	1.45	3.13
	1.94	3.15	1.59	3.67	1.65	3.14
sysbench	32.48	8.98	31.44	8.98	31.06	8.97
	32.54	9.00	31.50	8.99	31.10	8.98
	32.79	9.03	31.65	8.99	31.12	8.99
	32.80	9.03	31.70	8.99	31.16	9.00
	32.81	9.68	31.74	9.00	31.17	9.01
andebench2	59.36	20.29	57.67	20.27	56.75	20.31
	59.38	20.32	57.86	20.29	56.81	20.31
	59.52	20.33	57.90	20.39	56.84	20.41
	59.64	20.36	58.01	20.41	59.73	23.39
	59.69	21.39	58.11	20.49	59.84	23.51
andebench8	171.40	45.27	167.72	45.29	164.92	45.62
	173.45	45.46	168.11	45.52	165.61	45.65
	173.77	45.48	171.42	46.64	167.01	46.89
	175.08	46.45	171.72	46.73	167.40	47.09
	176.49	46.68	172.86	46.77	169.64	47.74
audio	7.73	30.00	6.84	30.02	5.50	30.02
	7.85	30.01	7.35	30.02	5.85	30.03
	7.85	30.01	7.46	30.03	6.04	30.03
	7.94	30.01	7.57	30.03	6.77	30.03
	7.97	30.02	7.73	30.03	7.24	30.03
audio+bbench	96.15	152.60	89.94	152.69	83.96	151.79
	98.67	155.18	92.03	152.80	85.63	155.76
	99.11	157.14	93.35	154.93	86.69	156.77
	99.86	157.82	93.75	158.71	87.56	160.54
	101.43	158.74	96.01	158.80	90.93	178.89

Which is Better?

- Both produce real benefits
  - Offloading gives slightly better wall-clock time
  - Enforced idle gives slightly better energy efficiency
  - Combining them does not help
- Both are needed
  - Offloading for real time and reduced OS jitter
  - Enforced idle for SMP energy efficiency