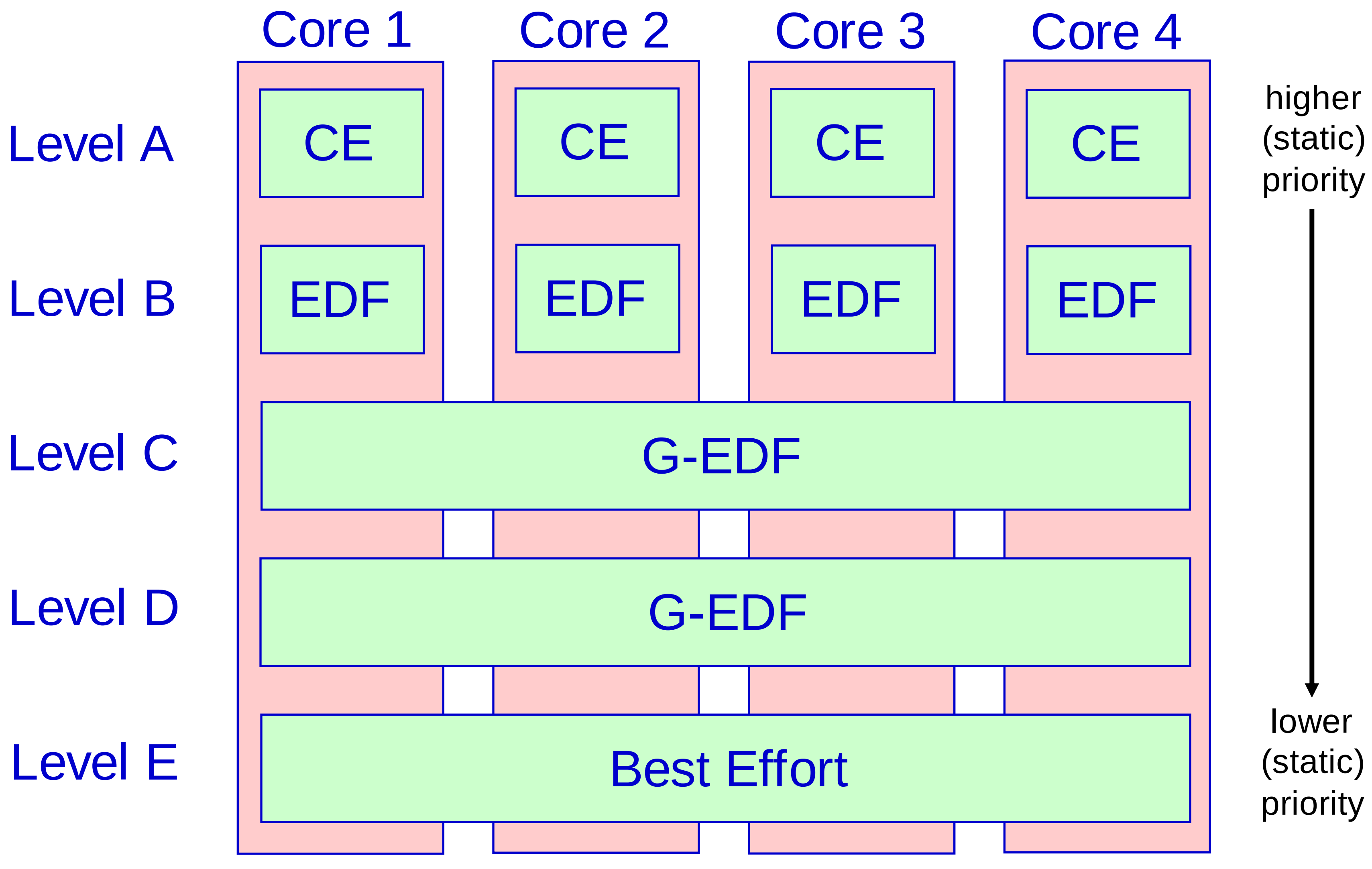
Mixed Criticality Plugin Discussion

Mac Mollison

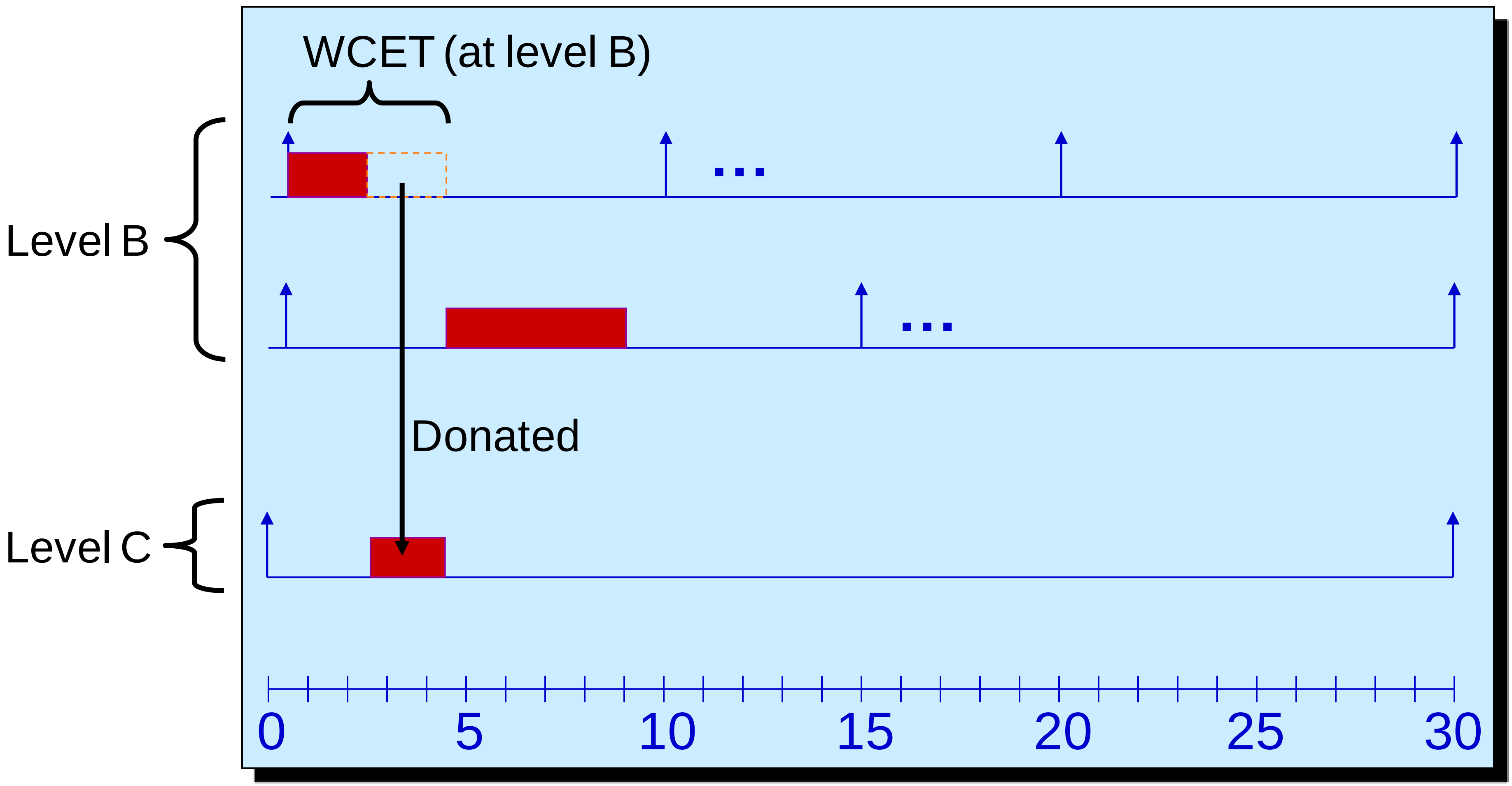
Outline

1. Review of mixed criticality (2 slides)
2. Current implementation (2 slides)
3. Future implementation: Adding slack shifting (10 slides)

Full Architecture



A (Very) Simple Example



Status of current implementation

- Levels A through E working
- Caveats
  - » Level A is currently P-EDF, and not table driven (yet)
  - » No “slack shifting” (yet)
  - » Minor variations btw. Mac’s code and Jeremy’s code; going to merge them together after this meeting.

Implementation Technique

- Based upon GSN-EDF
- Each 'container' gets its own rt_domain
  - » Levels A and B are added to cpu_entry_t. Levels C and D are global.
  - » Minor changes to various functions to deal with this
- Treat partitioned tasks basically like global tasks, except they only run on their partition ☺

Slack Scheduling

- Algorithm is based on the “ghost job” metaphor presented in the paper.

Slack Scheduling – Ghost jobs

- When a job at level X finishes, we convert it to a ghost job
  - We set a parameter `is_ghost` to `1`.
  - It is assigned *budget* starting at the difference between the level-X WCET and the actual execution time of the task
- We place this ghost job on the level-X run queue. (If level X is partitioned, we use the run queue for the CPU from which the job originally ran.)

Scheduling Ghost Jobs (Overview)

- A level-X ghost job is treated as a normal job from the perspective of the level-X scheduler.
  - It can be selected from the run queue as the job to schedule on a CPU.
  - A ghost job can preempt a normal job if its deadline is shorter.
- From the perspective of a scheduler below level X, a ghost job can be completely ignored.
- Schedulers at higher levels are covered on the (future) slide discussing preemptions.

Change to Support Ghost Jobs

- We will expand the `cpu_entry_t` struct.
  - We will add an array to track which *ghost jobs* are "executing" (consuming budget) on the same CPU – one entry per criticality level.

When a Ghost Job is Scheduled

- When a ghost job is scheduled, the `cpu_entry_t` will be updated and the starting time of the job fragment will be recorded.
- We also set a watchdog timer that will go off at the earliest time the budget could expire – the time at which it would expire given no preemptions.
- We then continue making scheduling decisions for lower levels as if no job had been scheduled.

Preempting Ghost Jobs

- We say a ghost job is *preempted* if a different job at the same or higher criticality is scheduled. It is *not* preempted if a job of lower criticality is scheduled.
- On preemption, the ghost job's budget must be updated based on how long the fragment actually ran, and the job is returned to the ready queue.
- To achieve this, whenever any task is linked to a CPU, we run this action on all ghost jobs of *lower* criticality on that CPU.

Watchdog Timers

- When a watchdog timer goes off, we update and check the budgets of all ghost jobs on the relevant CPU.
- Any ghost job which has finished is removed from the system, and we perform normal “job finished” tasks (i.e. checking for new tasks to schedule.)
- This code would also be executed on preemption in case a ghost job happens to finish just as it is being preempted for a different reason.

Global Scheduling – Added Complexity

- Currently, a single heap of available `cpu_entry_t` objects is used, and preemptions are checked on the CPU of lowest priority.
  - This is correct with no slack scheduling, because we statically prioritize level C over D.
- This is not correct with slack scheduling!

Global Scheduling – Added Complexity (contd.)

- Consider the following 2 CPU system:
  - On CPU 1, D_1 with a deadline of 1000 ms
  - Also on CPU 1, ghost job C_1
  - On CPU 2, D_2 with a deadline of 10 ms
- A new job C_2 should preempt D_2 on CPU 2
- However, a new job D_3 with deadline before 1000 ms should preempt D_1 on CPU 1!
- **No consistent “lowest priority” CPU!**

Global Scheduling – Added Complexity (contd.)

- We plan to solve this by having separate CPU heaps (referencing the *same* `cpu_entry_t` objects) for levels C and D.
- The priority function will be changed such that:
  - At level C, level-C ghost jobs are considered as normal level-C jobs. (The treatment of level-D ghost jobs doesn't matter.)
  - At level D, level D ghost jobs are considered as normal level D jobs, but level C ghost jobs are considered as if they were *not running*.