THE UNIVERSITY of York

RTS York

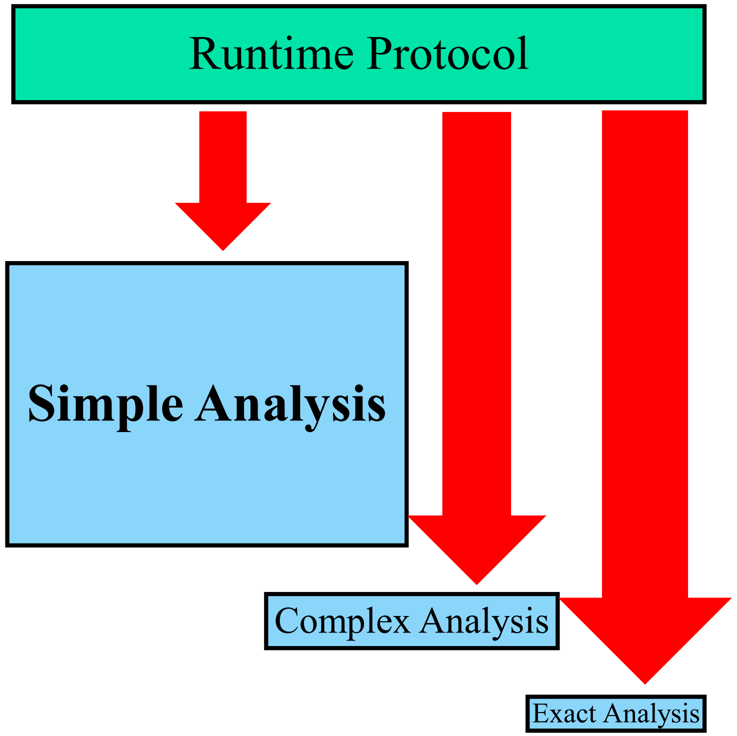
Analysis-Runtime Co-design for Adaptive Mixed Criticality Scheduling

Iain Bate, Alan Burns, Robert Davis,

Real-Time Systems Research Group, University of York, UK

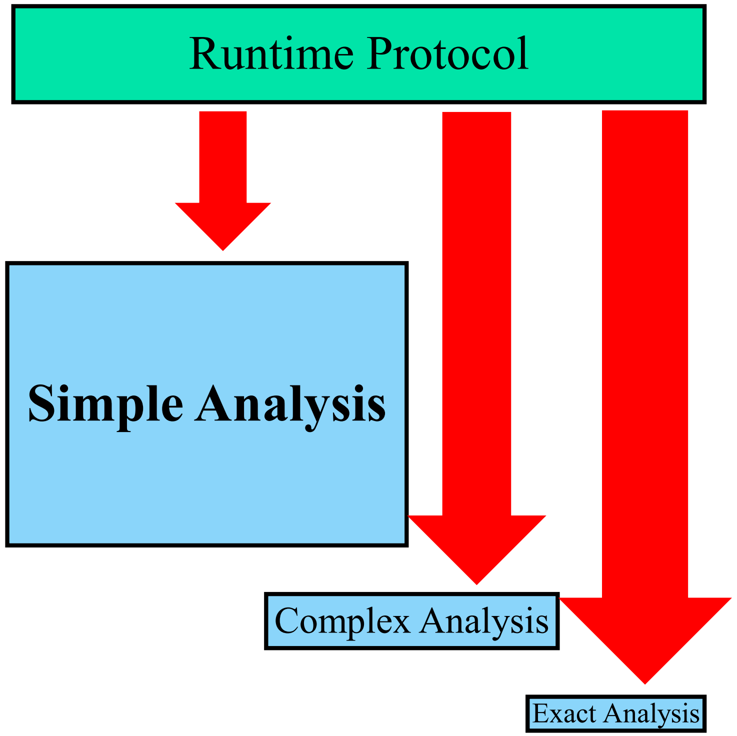
Background: Analysis-Runtime Co-design

- **Focus of this research**
  - Runtime scheduling protocols and their schedulability analysis
- **Traditional approach**
  - Runtime protocol designed first
  - Typically this is done without considering the difficulties involved in providing analysis for it
  - Schedulability analysis comes later, often in the form of a simple tractable test that is *sufficient*, but not exact
  - Subsequent work then tends to focus on ever more precise analysis, trading off complexity for greater precision
  - Finally, *exact* analysis, if it can be developed at all, is often intractable, and may also be quite difficult to understand



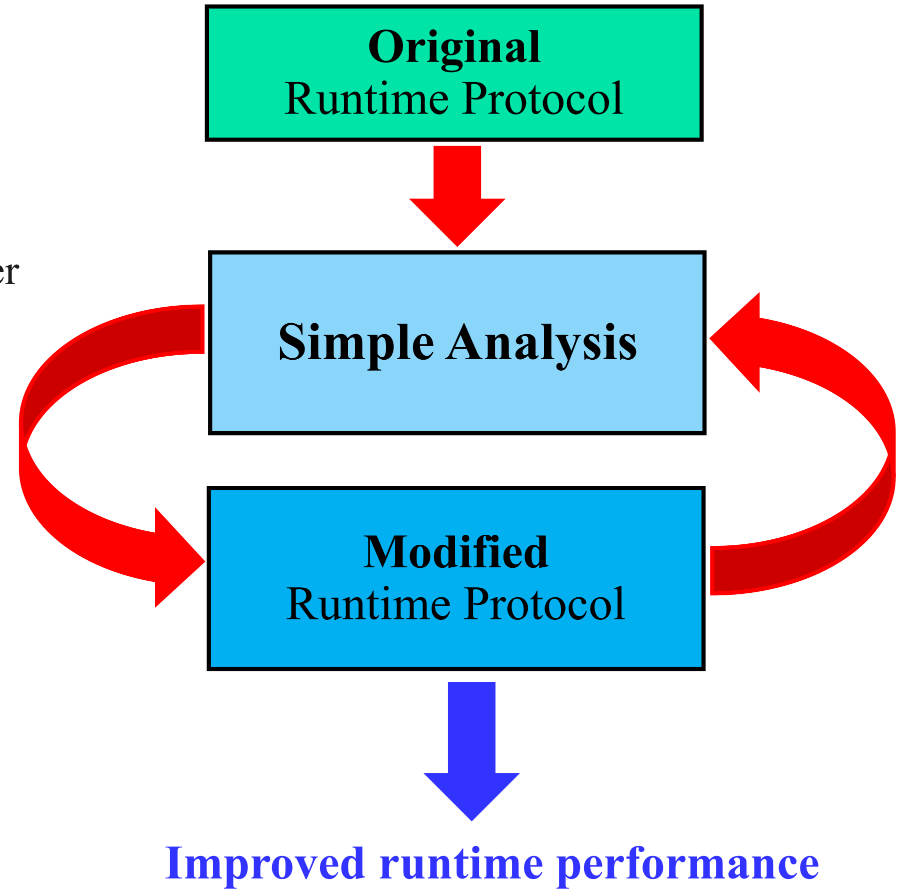
Background: Analysis-Runtime Co-design

- **Industrial perspective**
  - Industry has a strong preference for simple solutions
  - Simple analysis may well be: "*good enough for industrial use*"
  - Marginal gains of more complex analysis may not be worthwhile, given that it is usually much harder to understand and to build upon
- **Mantra: "*Don't let the perfect be the enemy of the good*"**
  - Often attributed to **Voltaire**
  - More likely attributable to Charles-Louis de Secondat, Baron de La Brède et de **Montesquieu**
  - "*Le mieux est le mortel ennemi du bien*" or "The better is the mortal enemy of the good"



Background: Analysis-Runtime Co-design

- **Basic idea**
  - **Retain the simple analysis** along with the schedulability guarantees that it provides
  - **Refine the runtime protocol** so that it has improved performance with respect to other important metrics while still complying with the assumptions of the analysis



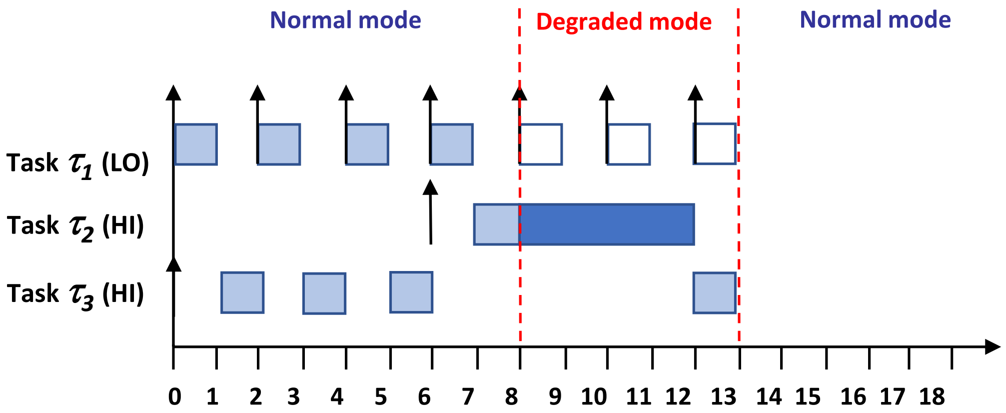
Mixed Criticality Systems

- **System model**
  - Tasks are characterized by their *criticality* level either HI or LO
  - LO-criticality tasks have a single LO-criticality estimate of their WCET, $C_i(LO)$
  - HI-criticality tasks have an additional HI-criticality estimate $C_i(HI)$

- **Timing Assurance Requirements**
  - **Requirement R1**: (**Normal behaviour**) If all jobs of the tasks comply with their LO-criticality WCET estimates $C_i(LO)$, then all jobs must be guaranteed to meet their deadlines.
  - **Requirement R2**: (**Abnormal behaviour**) If a job of a HI-criticality task executes for its LO-criticality WCET estimate $C_i(LO)$ without completing, then only jobs of HI-criticality tasks are required to meet their deadlines.

Adaptive Mixed Criticality (AMC)

- **Runtime protocol for AMC**
  - Based on Fixed Priority Pre-emptive Scheduling
  - System starts in **normal mode** where all tasks can release jobs
  - If a HI-criticality job executes for its $C_i(LO)$ without completing then the system enters **degraded mode**
  - In **degraded mode**, HI-criticality tasks can release new jobs, whereas new jobs of LO-criticality tasks are abandoned and do not execute
  - On an idle instant, the system returns to **normal mode**



Original AMC scheme

Schedulability Analysis: Concepts

- **Key concept in the analysis of fixed priority pre-emptive scheduling**
- **Priority level-*i* busy period**
  - This is a contiguous interval of time during which jobs of tasks of priority i or higher execute
  - It starts at a time $s[i]$ when a job of a task of priority i or higher is released and there are no jobs of tasks of priority i or higher that currently have any execution pending
  - It ends at the earliest time $t[i]$ after its start $s[i]$ when there are no jobs of tasks of priority i or higher that have execution remaining that were released strictly before the end time $t[i]$

- **Useful properties**
  - Longest priority level-i busy period upper bounds the worst-case response time of the task at priority i

Analysis for AMC: AMC-rtb test

- **Normal behaviour**

$$R_i(LO) = C_i(LO) + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO)$$

1. All interfering jobs of higher priority HI-criticality tasks can execute for their $C_j(HI)$

- **Abnormal behaviour**

$$R_i(HI) = C_i(HI) + \sum_{j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI)$$

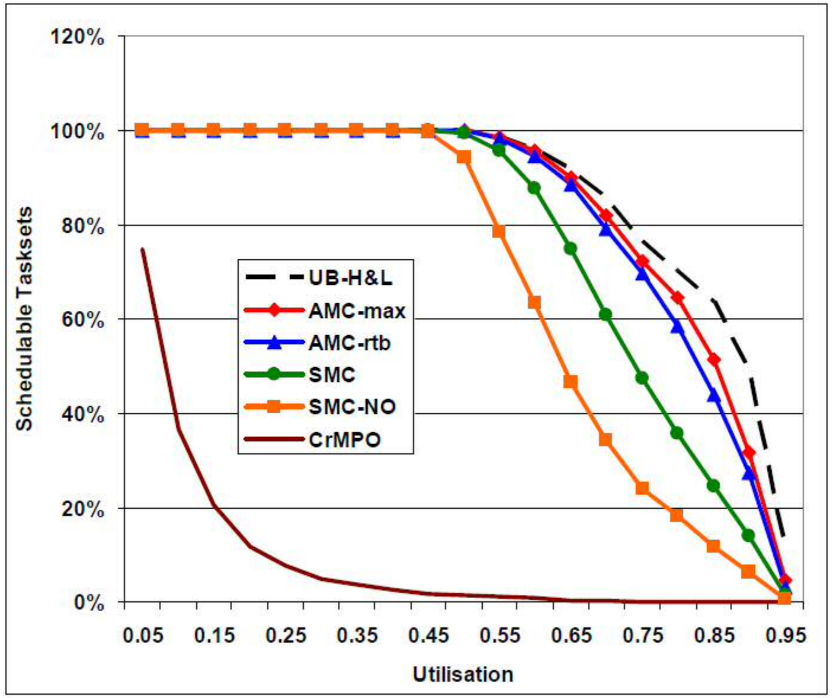
$$+ \sum_{k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO)$$

2. Jobs of higher priority LO-criticality tasks that are released by $R_i(LO)$ as measured from the start of the busy period can cause interference

3. Pessimistic: Once a single HI-criticality job executes for its $C_j(LO)$ without completing, then no more jobs of LO-criticality tasks can be released

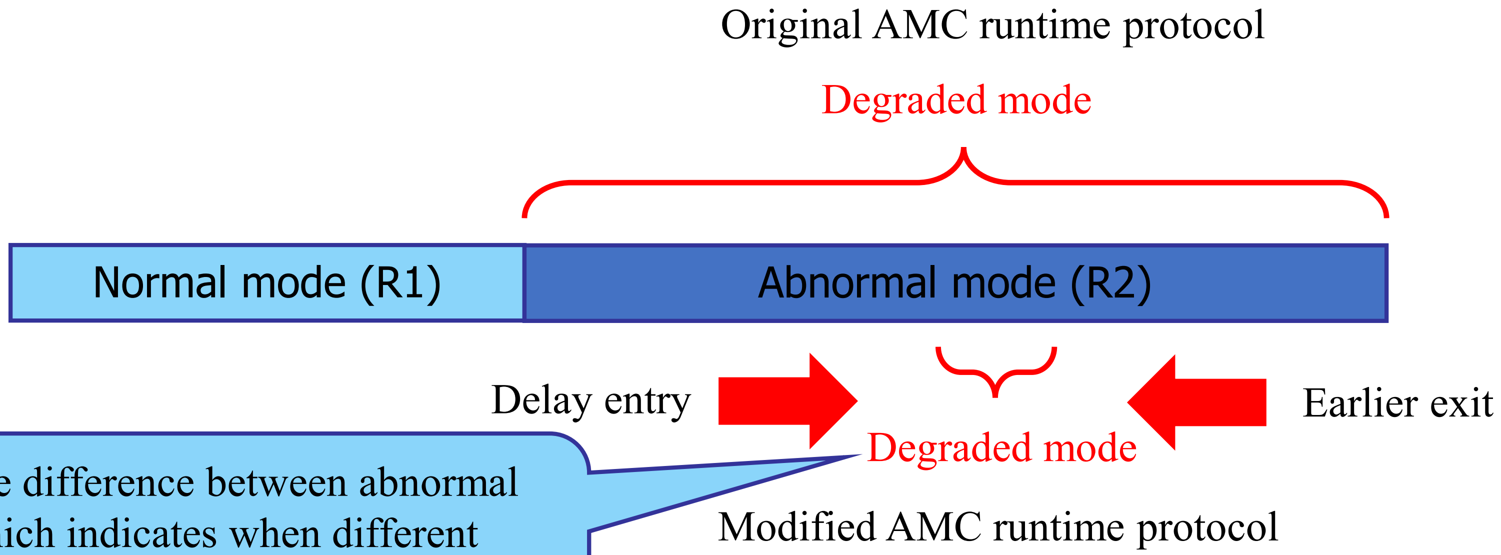
Analysis for AMC: AMC-rtb test

- **De-facto standard test for AMC**
  - Published in RTSS 2011
  - Built upon by many subsequent papers, which extended the original work
  - Performance of the AMC-rtb test is close to that of more complex tests, such as AMC-max
  - AMC-rtb test is however more suitable for industrial use due to its simplicity and effectiveness
  - Studies into the use of AMC (based on the AMC-rtb test) have been done by a major aerospace company: Rolls Royce Control Systems on a Full Authority Digital Engine Controller (FADEC)



What can we do to improve upon the AMC runtime protocol?

- **Aim is to reduce the time spent in degraded mode**
  - Reducing how often degraded mode is entered
  - Waiting longer before entering degraded mode in the first place
  - Exiting degraded mode quicker
- **To achieve the main goal: abandon far fewer LO-criticality jobs**



Notice the difference between abnormal mode, which indicates when different requirements apply, and degraded mode, which indicates a runtime behaviour

How can we improve upon the original AMC runtime protocol?

- **Modify the runtime protocol to closely follow the analysis**
  - Allow jobs of HI-criticality tasks to execute for their $C_i(HI)$, and also permit LO-criticality tasks to release jobs until some job of a HI-criticality task τ_i reaches a time $R_i(LO)$ since the start of the priority level-i busy period in which it was released

$$R_i(HI) = C_i(HI) + \sum_{j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) + \sum_{k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO)$$

- **Key point: Trigger on response times rather than execution times**
  - The system starts in **normal mode** where all tasks can release jobs
  - If an active job of a HI-criticality task τ_i reaches a time equal to its $R_i(LO)$ after the start of the priority level-i busy period in which it was released then the system enters **degraded mode** where only HI-criticality tasks can release jobs
  - When a job of some HI-criticality task τ_j completes and there is no active job of any other HI-criticality task τ_k that has reached a time equal to its $R_k(LO)$ after the start of the priority level-k busy period in which it was released then the system returns to **normal mode**

Proof in the paper that the AMC-rtb test holds for this modified AMC runtime protocol

Pros and Cons of the modified AMC runtime protocol

- **Advantages**
  - Compatible with the AMC-rtb schedulability test and retains its guarantees
  - For any given scenario, entry into degraded mode cannot be earlier with the modified protocol, since $R_i(LO)$ is the latest that the transition could occur when triggering a change to degraded mode based on execution times
  - Typically, entry into degraded mode is much later and is often not required at all
  - Further, exit from degraded mode is typically much earlier than waiting for an idle instant
  - Automatically takes advantage of any gain time produced when interfering jobs execute for less than their worst-case execution time
  - Also automatically takes advantage of non-worst-case patterns of job arrivals from higher priority tasks (for example sporadic behaviours and periodic releases that are not synchronized, i.e. not at a critical instant)
- **Disadvantages**
  - Exact schedulability is dominated by (i.e. worse than) that for the original AMC runtime protocol
  - Not compatible with the improved but still sufficient AMC-max schedulability test

Enhancements to AMC scheduling schemes

- **Static Slack**
  - Increasing $C_i(LO)$ as far as possible for each HI-criticality task, which delays entry into degraded mode for the original runtime protocol and also for the modified runtime protocol via increased $R_i(LO)$ values
- **Gain Time**
  - Gain time occurs when a job executes for less than its execution time budget
  - Explicitly accounting for gain time and transferring it to the next lower priority task can improve the performance of the original runtime protocol
  - Gain time is automatically taken care of by the modified protocol, since it triggers on response times
- **Lazy Execution**
  - Last chance opportunity for LO-criticality jobs that would otherwise be abandoned in degraded mode to run via a separate background priority queue
  - Not appropriate for all systems as it can increase blocking effects and impacts mutual exclusion primitives that are based on priorities

Scenario-based Evaluation

- **Configuration**
  - Generated 500 synthetic task sets with utilization 0.8 that were schedulable according to AMC-rtb, but not schedulable under FPPS as a single-criticality system
  - Task periods used were either semi-harmonic (typical of automotive and avionics systems) or non-harmonic
  - $C_i(HI) = 2\ C_i(LO)$
  - At runtime, jobs had variable execution times with a probability of exceeding $C_i(LO)$ of 0.01% (i.e. approx. 1 in 10,000 jobs of HI-criticality tasks exceed their $C_i(LO)$)
  - Simulation run for each task set was 10^{13} time units, enough for 10^6 periods of the longest task

- **Performance metrics**
  - **HDM** Number of HI-criticality task deadline misses – this was always zero, so is not shown on the graphs
  - **NiD** Number of times degraded mode was entered
  - **TiD** Total time spent in degraded mode
  - **JNE+LDM** Number of LO-criticality jobs that were either not executed or missed their deadlines

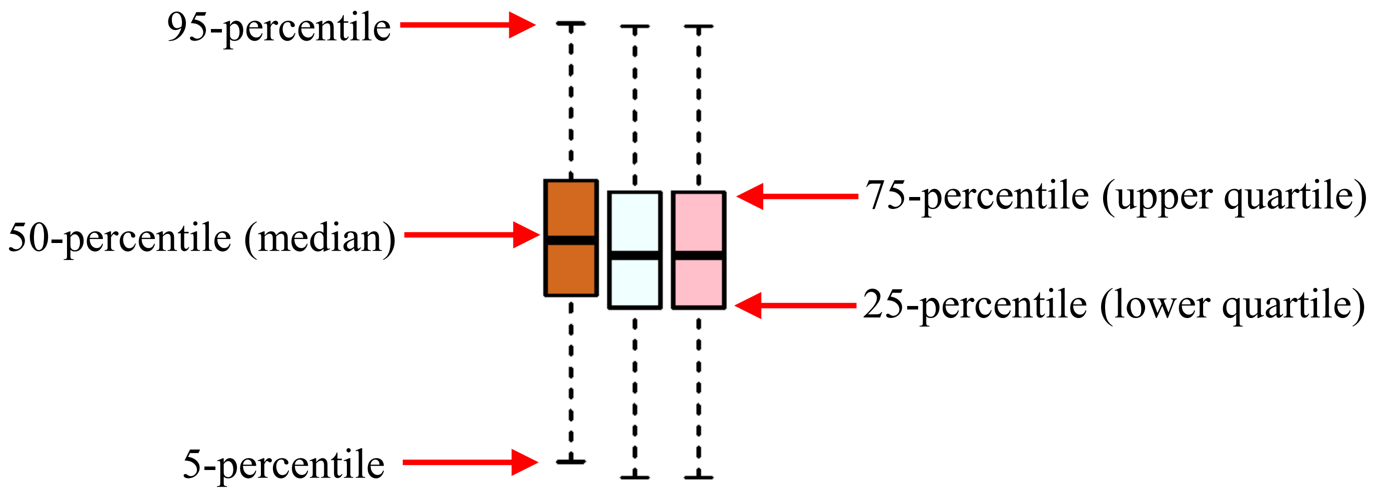
Scheduling Schemes

- **Comparison between different families of scheduling schemes**
  - **AMC-RA** modified runtime protocol with exit from degraded mode on an idle instant
  - **AMC-RH** modified runtime protocol with fast exit from degraded mode
  - **AMC+** original runtime protocol with exit from degraded mode on an idle instant
  - **BP** Bailout Protocol – based on AMC+ with a faster return to normal mode

- **Variants**
  - **S** (Static Slack) e.g. **AMC-RAS**, **AMC-RHS**, **AMC+S**, **BPS**
  - **G** (Gain time) e.g. **AMC+SG**, **BPSG**
  - **L** (Lazy execution) **AMC-RASL**, **AMC-RHSL**, **AMC+SGL**, **BPSGL**

Presentation of results

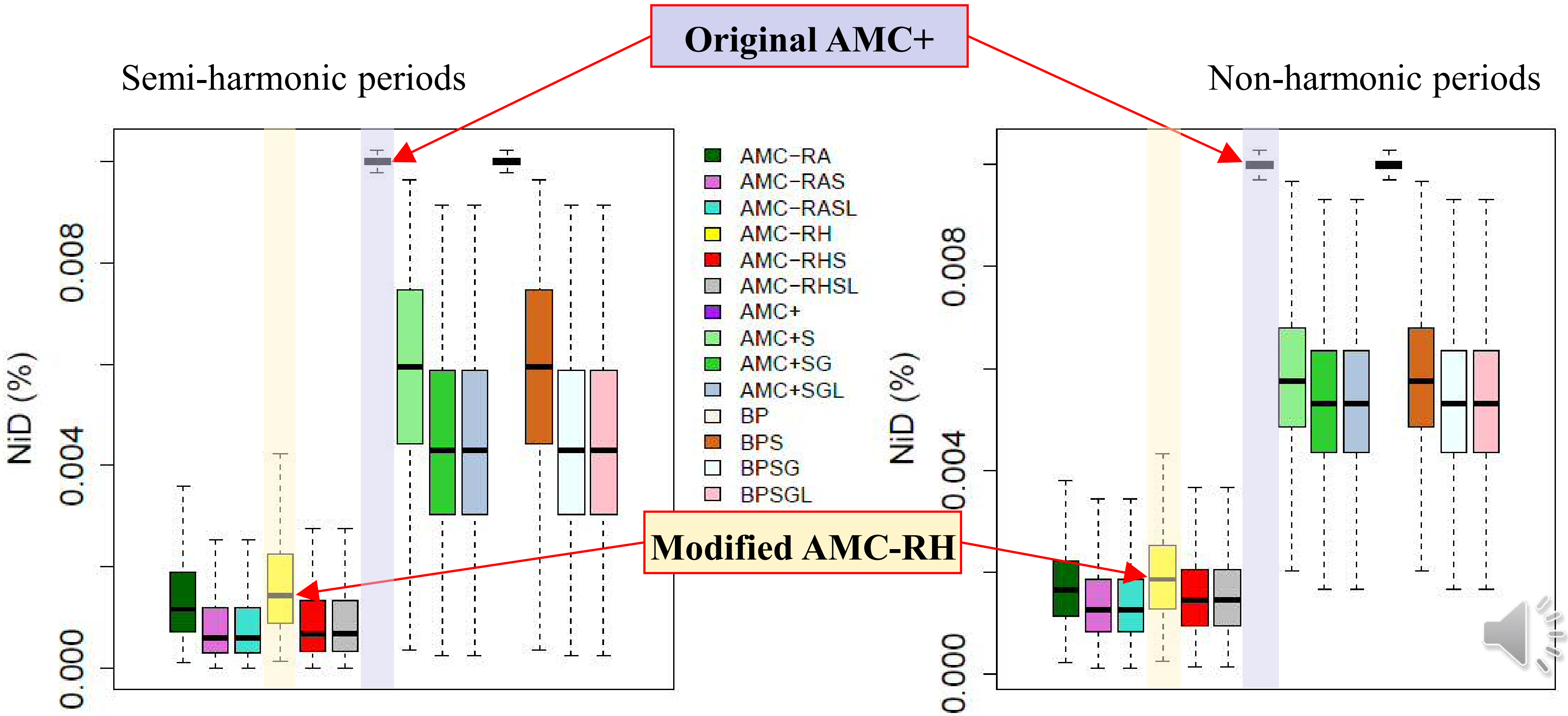
- **Box and whisker plots**



Results

1. Number of times degraded mode entered reduced to 16.8% and 19.9% respectively of the mean values for the original protocol

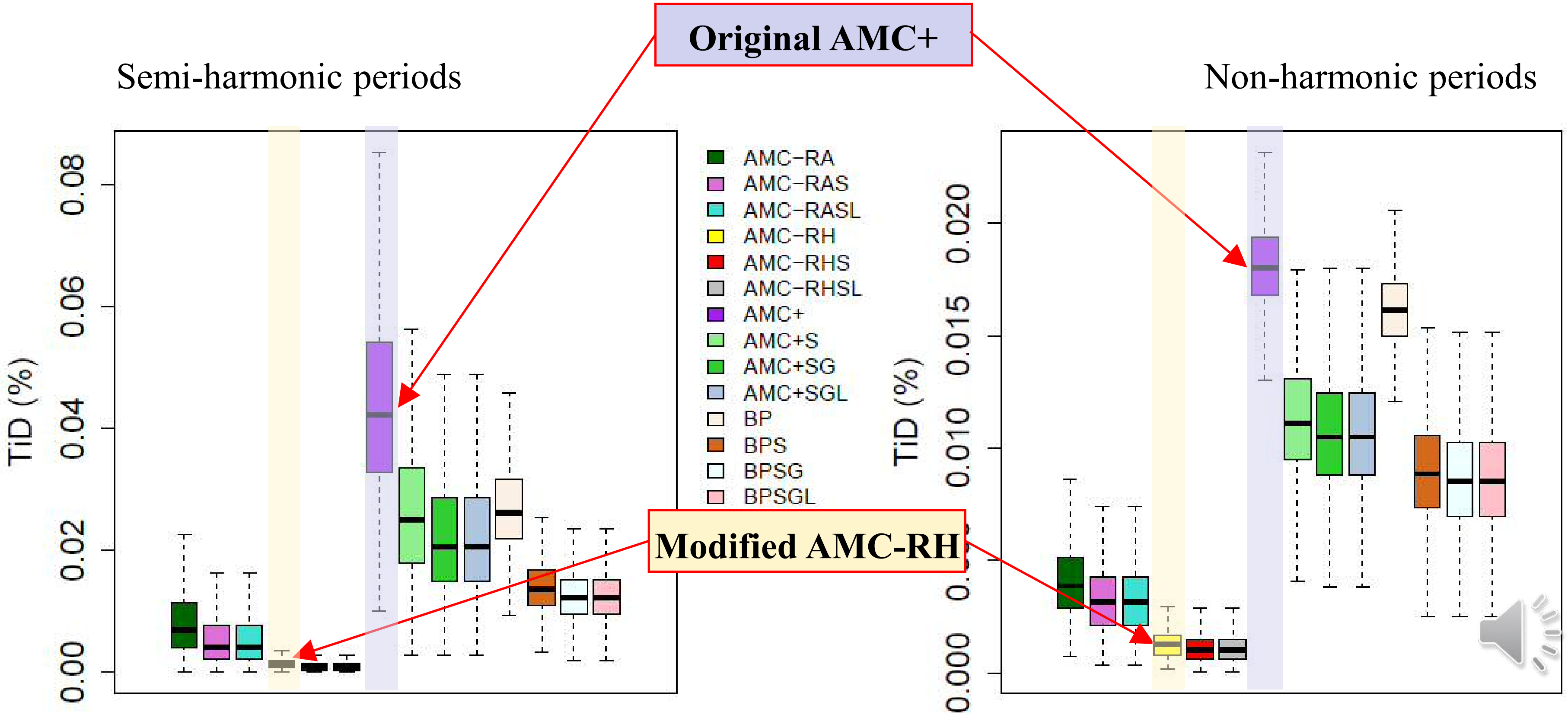
- **NiD% (Number of times degraded mode entered)**



Results

2. Total time in degraded mode reduced to 1.7% and 4.1% respectively of the mean values for the original protocol

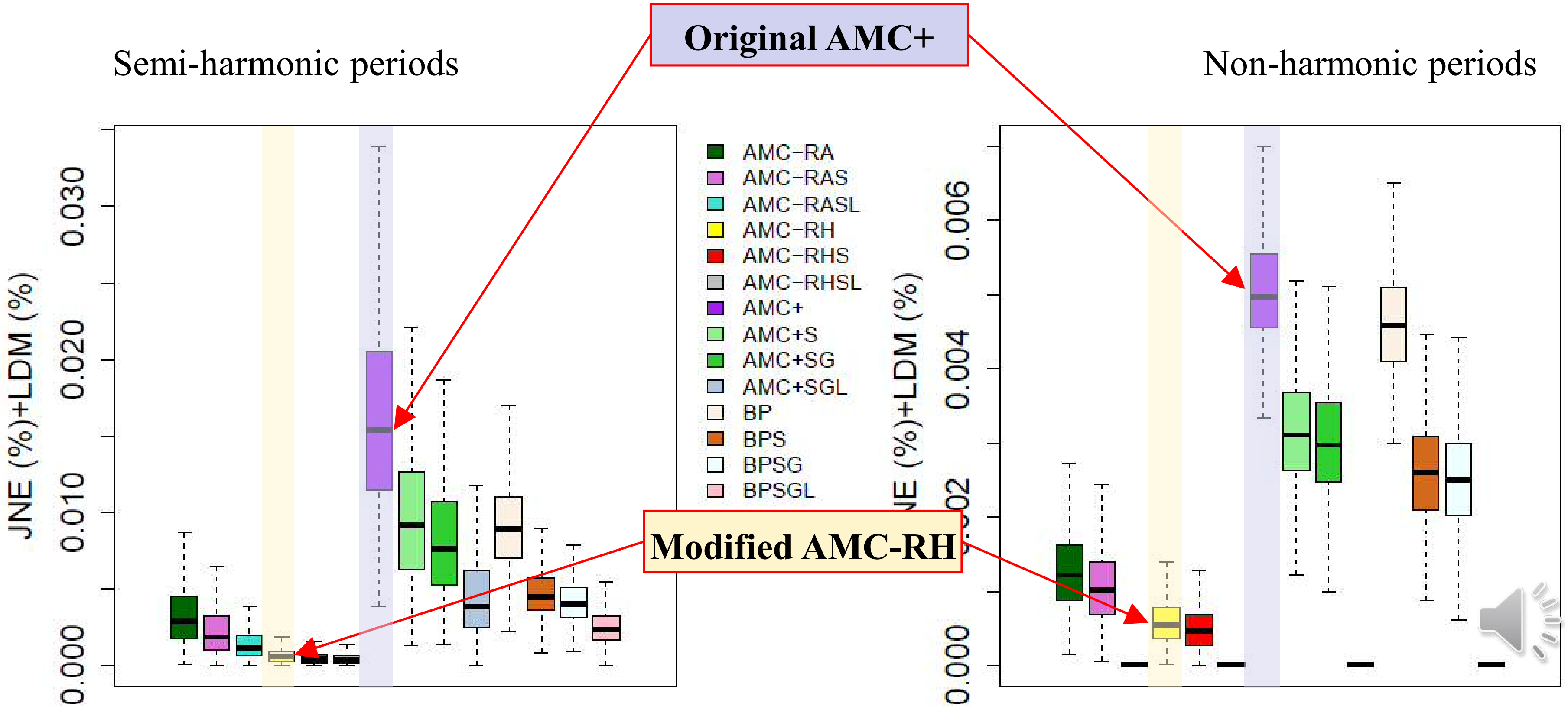
- **TiD% (Total time in degraded mode)**



Results

3. LO-criticality jobs not executed or missed deadline reduced to 2.5% and 8.7% respectively of the mean values for the original protocol

- **JNE%+LDM% (LO-criticality jobs not executed or missed deadline)**



Implementation of the modified AMC runtime protocol

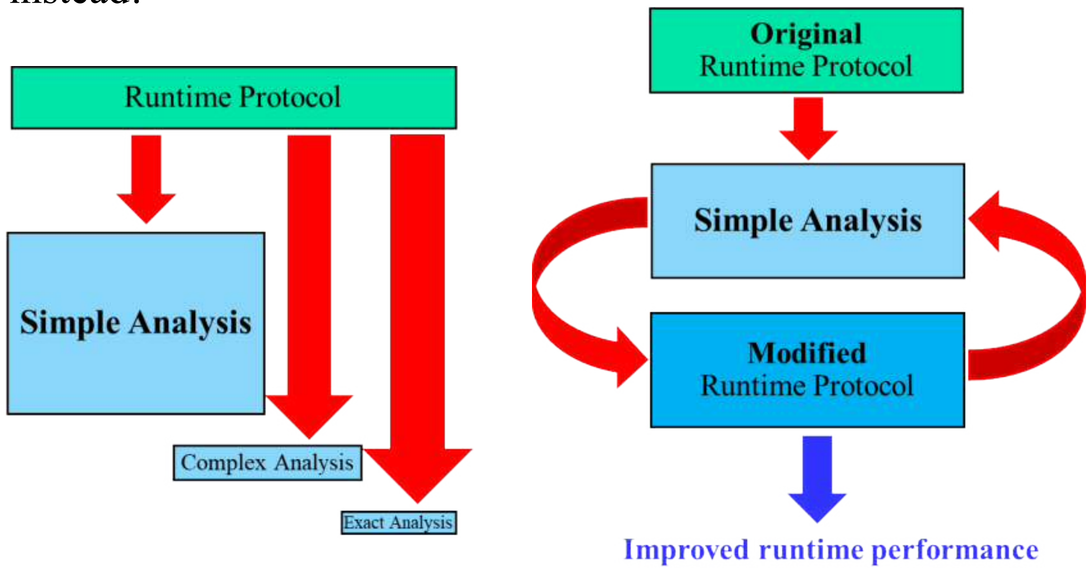
- **RTOS track busy period start times**
  - Need the start time $s[i]$ of each currently active priority level-i busy period for all priority levels corresponding to HI-criticality tasks
  - Track these start times via O(1) operations at each job release
  - When a new job of task τ_i is inserted into the run queue then if it is inserted at the head of the run queue $s[i]$ = *current time* (i.e. the release time of the job) otherwise the busy period start time is inherited, $s[i] = s[k]$, from that of the task τ_k that is immediately ahead of task τ_i in the run queue (i.e. next higher priority active task)
- **RTOS track response time expiries**
  - Require monitoring of response time expiry for all active jobs of HI-criticality tasks
  - **Similar to monitoring deadline expiry** and can be integrated with it
  - It can be implemented using a single timer interrupt and an expiry queue
  - O(log n) operations at each job release (for queue insertion)
  - O(1) operations to handle response time expiry (e.g. to switch to degraded mode)
  - O(1) operations at job completion (e.g. to switch back to normal mode)

Conclusions: Modified AMC runtime protocol

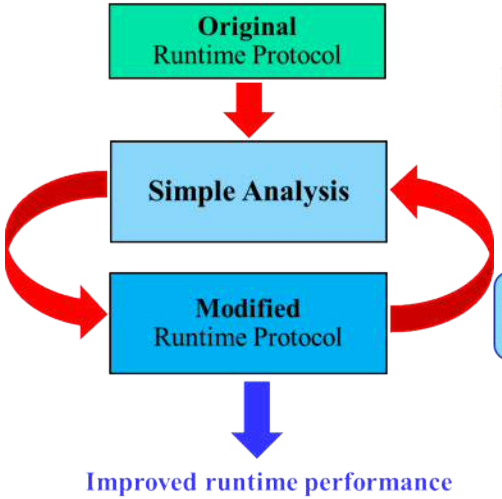
- **Retains the schedulability guarantees afforded by the AMC-rtb test**
  - For systems passing the AMC-rtb test, all tasks meet their deadlines according to the requirements R1 and R2 placed on Mixed Criticality Systems
- **Substantial improvements in runtime metrics vs original protocol**
  - Reduces the number of times that degraded mode is entered (5-6 fold reduction)
  - Reduces the total time spent in degraded mode (24-60 fold reduction)
  - Reduces the number of LO-criticality jobs that are abandoned or miss their deadlines (11-40 fold reduction)
  - Larger of these improvements were observed with semi-harmonic periods typical of automotive and avionics systems
  - Automatically benefits from gain time and non worst-case job release patterns
- **Suitable for use by industry**
  - Based on the simple yet effective AMC-rtb test and its guarantees
  - Substantial improvements in runtime performance, specifically a large reduction in the number of abandoned LO-criticality jobs
  - Similar implementation overheads and complexity to policing task deadlines

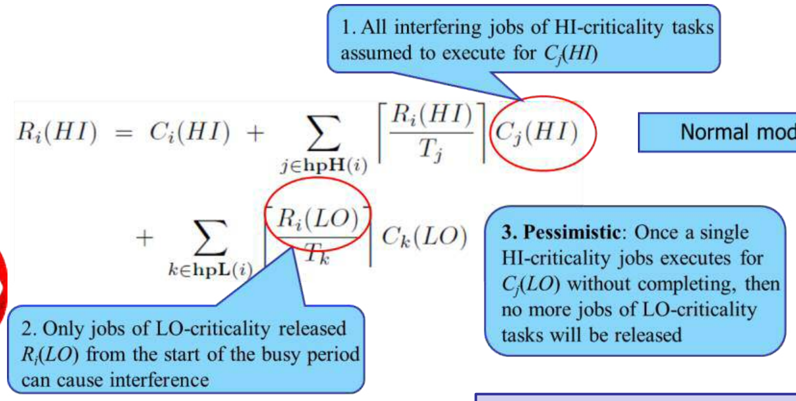
And finally…

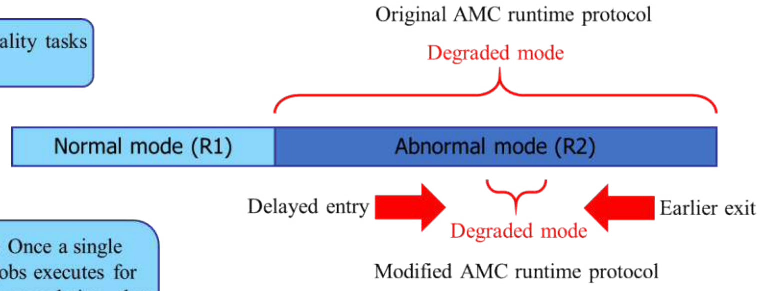
- **Encourage other researchers to explore the idea of Analysis-Runtime Co-design**
  - Significant research effort typically goes into deriving improved schedulability tests often for marginal gains
  - Let’s not forget that other aspects are also important to industry
  - It can be worthwhile using simple analysis and improving the runtime protocol instead!



Discussion and Questions?







rob.davis@york.ac.uk