Towards a Time-Composable Operating System

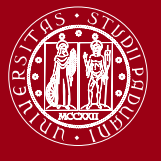
Andrea Baldovin, Enrico Mezzetti, Tullio Vardanega

18th International Conference on Reliable Software Technologies
(Ada-Europe 2013)
Berlin, June 13th, 2013

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n.249100.

Università degli Studi di Padova

Outline



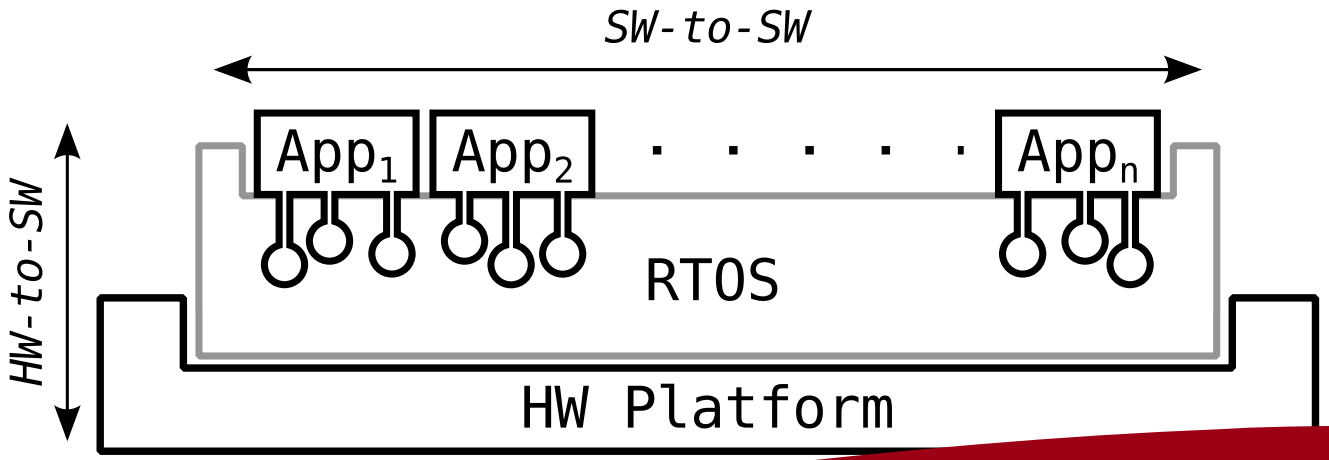
Motivation

- Incremental qualification of Hard Real-Time systems must be performed on both their functional and timing dimension
- The process is facilitated if the system and its elementary components can be carachterized in terms of *compositionality* and *composability*
  - Compositionality: the properties of interest at system level can be determined from relevant properties of its constituent components
  - Composability: the properties determined for individual components in isolation should hold in the face of composition with other components
- Functional composition is comparatively easy to achieve
- Time composition is much harder and threatens compositionality

Time composability

- The timing properties calculated for individual components in isolation may be invalidated at any level of the execution stack by disrupting effects in HW/SW behaviour and their interactions

- Some of them pertain to the HW-to-SW axis of composition (e.g. inter-task communication), others arise from the interaction between HW and SW (e.g. cache state pollution)

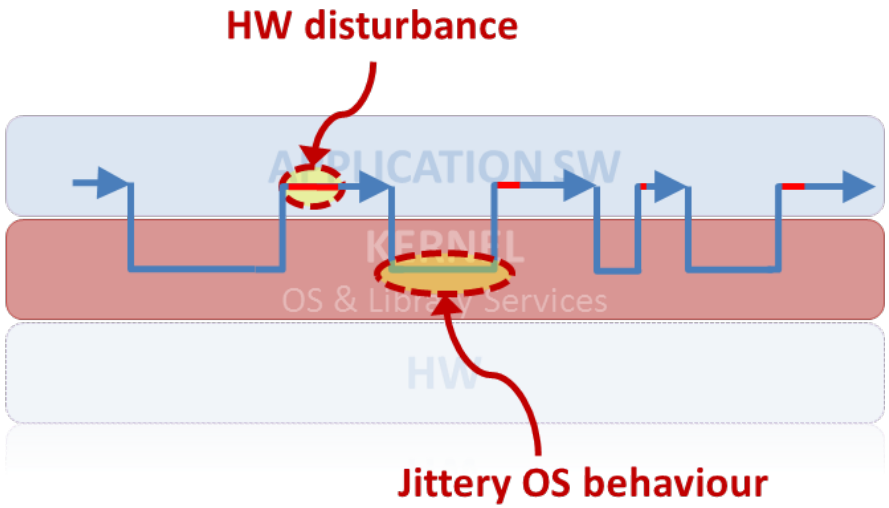


Time composability

- Most of existing work focuses on the HW-to-SW axis of composition, attacking issues in the HW platform
  - Providing specialized HW can be unaffordable
  - Selectively disabling HW features may be more convenient
- What can be achieved at software level depends on the designed SW architecture and task model
  - Find a tradeoff between analyzability and expressiveness
- The RTOS is arguably the best place to shield the application from HW effects and to inject composability in the SW layer
  - No special attention is given to the OS in traditional analysis
  - Its contribution is usually accounted for as part of application timing

Disturbance and jitter

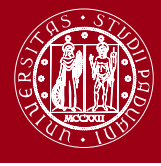
- In previous work[1] we introduced two principles to be enforced in the RTOS to enable time composability, namely *zero-disturbance* and *steady timing behaviour*
  - Zero-disturbance: RTOS execution may affect the response time of applications by polluting the inner state of history-sensitive resources
  - Steady timing behaviour: the jittery behaviour of RTOS primitives impairs time composition with the application



[1] Baldovin, Mezzetti, Vardanega in WCET'12

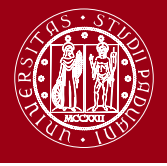
- We applied those principles to a very benevolent ARINC-653 setting; we now investigate a more expressive task model
- Lightweight RTOS kernel implementing the Ravenscar profile on a LEON2 architecture, gently provided by UPM
- Sporadic task model, under fixed-priority scheduling
- Some restrictions apply to task behaviour and interaction
  - No task termination allowed
  - No synchronous communication
  - Deadlock-free synchronization on protected resources regulated through the ICP protocol
  - At each time, no more than one task queuing on a resource

Areas of intervention



- Non-intrusive time managament: progression of time should interfere as least as possible with application execution
- Constant-time scheduling primitives: avoid execution time variability of kernel routines depending on system state
- Flexible task model: synchronous events should be serviced with minimum latency but preserving time composability
- Composable inter-task communication: task communication and synchronization via shared resources should be permitted in a time-composable way
- Selective HW isolation: RTOS-induced perturbations on history-dependent HW should be avoided

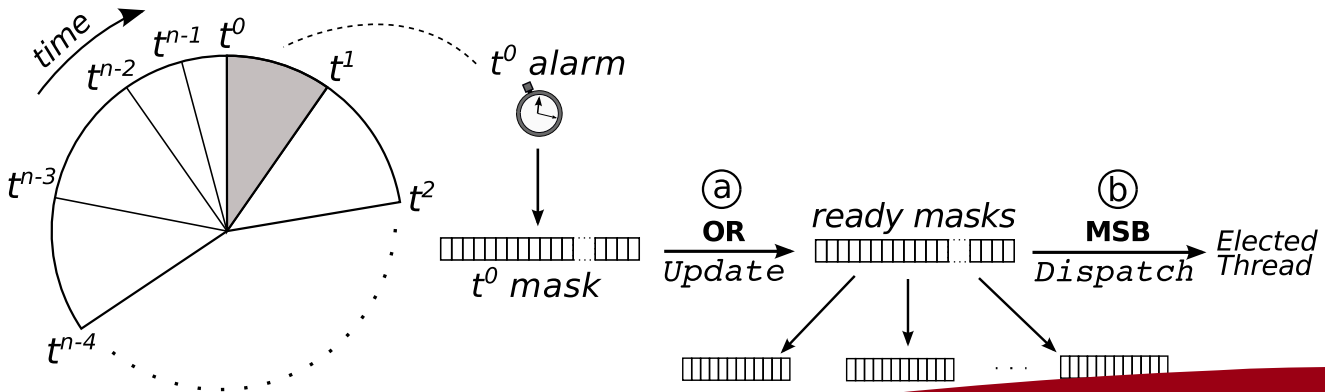
Non-intrusive time management



- No tick counting which causes great disturbance to application execution
- The range of representable time values shall be sufficient to represent 50 years with a tick precision of 1 millisecond (Ada RM, Annex D)
- LEON2 processor only provides 2x 24-bit periodic/one-shot time registers requiring some additional software management
- Porting to PPC 750 (64-bit time base + 32-bit one-shot timer)

Constant-time scheduling primitives

- Assuming FPPS and no shared resource, scheduling and dispatching occur at task release and completion events
- For scheduling and dispatching, ORK+ keeps two lists:
  - Alarms to be fired at task activations are sorted by expiry time
  - Ready tasks are sorted by decreasing priority and activation
- Our redesigned scheduler takes benefit of constant-time operations on bit-masks



Flexible task model

- By admitting *sporadic task* and timing events, activations do not follow a predictable pattern anymore

- Limited preemptive scheduling can help mitigate the negative effects of intrusive, uncontrolled preemption on composability
  - Activation acknowledgement and scheduling are kept separate
  - The number of incurred preemptions is minimized
  - Schedulability is preserved and in some cases even improved
  - *Timing events* should be processed at the time of occurence

Flexible task model

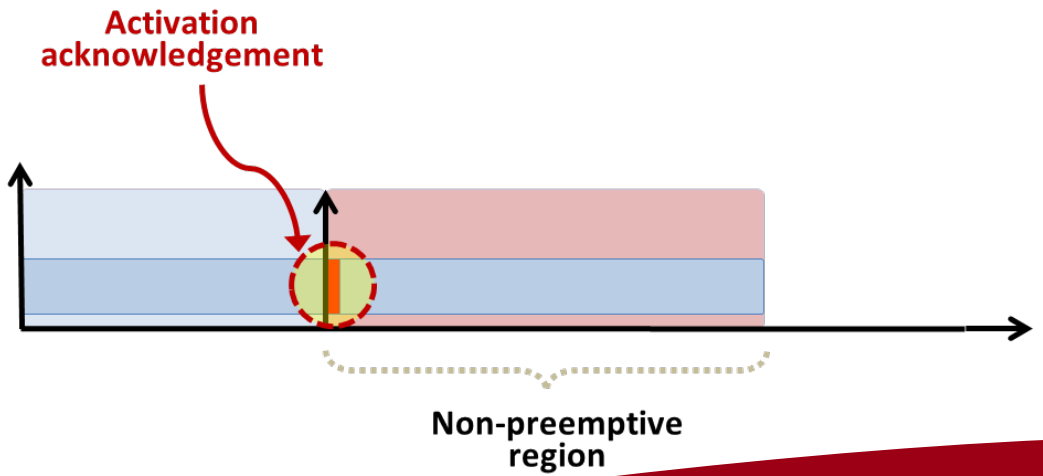
- By admitting *sporadic task* and timing events, activations do not follow a predictable pattern anymore

- Limited preemptive scheduling can help mitigate the negative effects of intrusive, uncontrolled preemption on composability
  - Activation acknowledgement and scheduling are kept separate
  - The number of incurred preemptions is minimized
  - Schedulability is preserved and in some cases even improved
  - *Timing events* should be processed at the time of occurence

Flexible task model

- By admitting *sporadic task* and timing events, activations do not follow a predictable pattern anymore

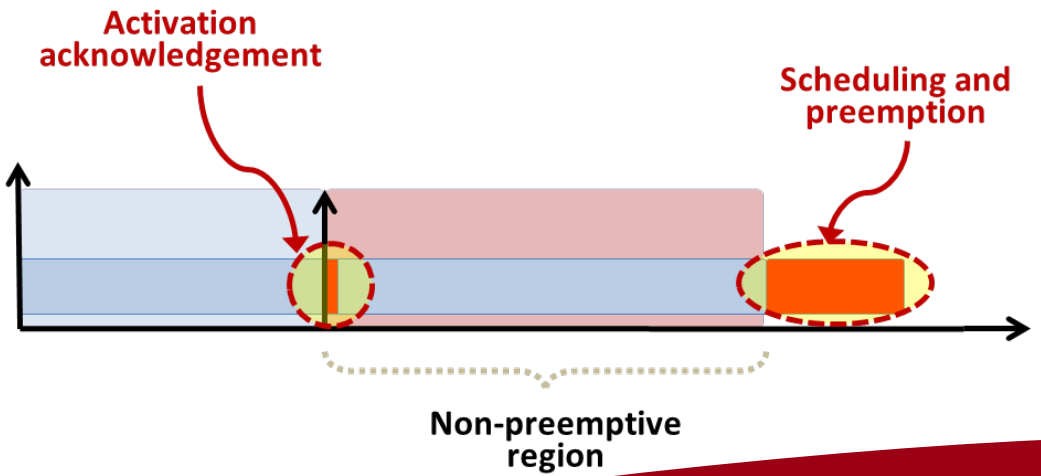
- Limited preemptive scheduling can help mitigate the negative effects of intrusive, uncontrolled preemption on composability
  - Activation acknowledgement and scheduling are kept separate
  - The number of incurred preemptions is minimized
  - Schedulability is preserved and in some cases even improved
  - *Timing events* should be processed at the time of occurence



Flexible task model

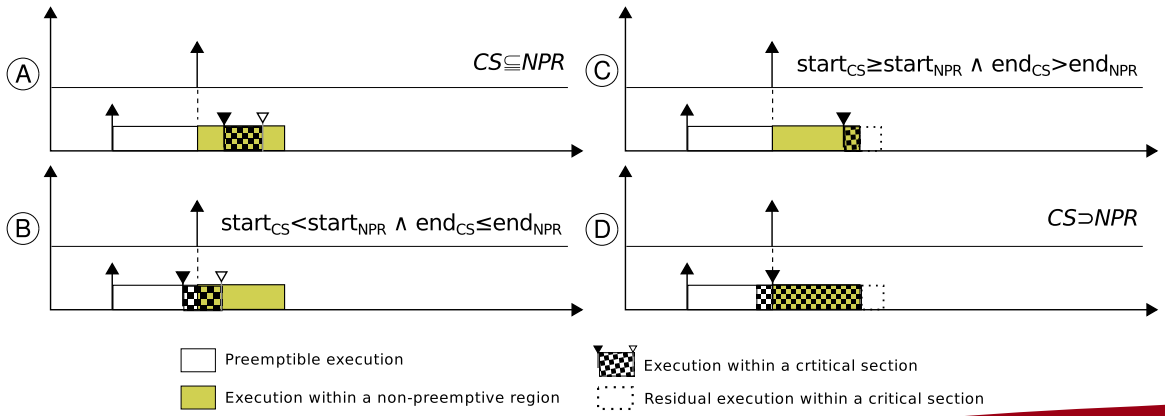
- By admitting *sporadic task* and timing events, activations do not follow a predictable pattern anymore

- Limited preemptive scheduling can help mitigate the negative effects of intrusive, uncontrolled preemption on composability
  - Activation acknowledgement and scheduling are kept separate
  - The number of incurred preemptions is minimized
  - Schedulability is preserved and in some cases even improved
  - *Timing events* should be processed at the time of occurence



Composable inter-task communication

- Applies to both SW and HW resources
- The ceiling locking protocol included in Ravenscar and implemented by ORK+ avoids deadlock and provides a strict bound on the blocking suffered due to resource sharing
- Challenge: make blocking from resource sharing and blocking from deferred preemption coexist (submission to EMSOFT'13)

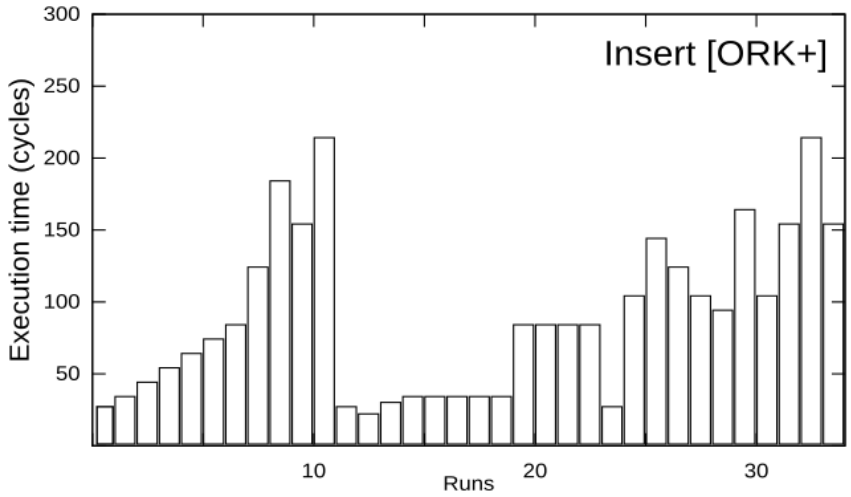


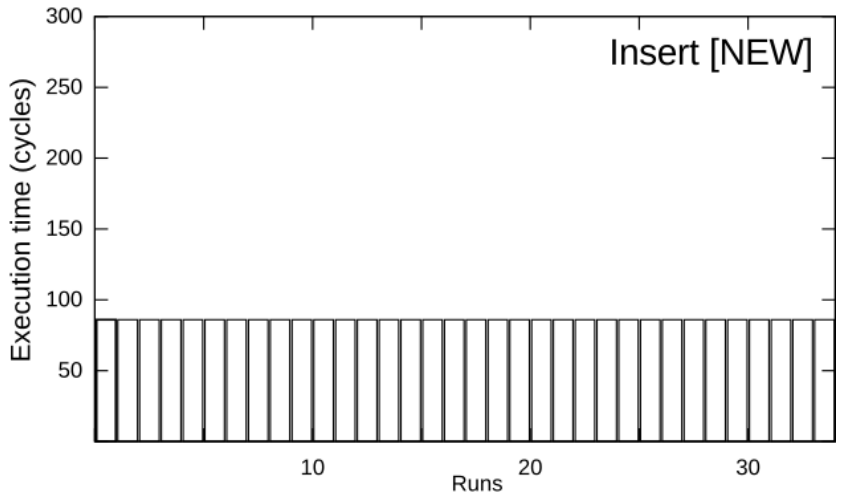
Selective HW isolation

- The principle of zero disturbance requires a mechanism to prevent the RTOS execution from polluting the HW state of the user application
- Example: cache partitioning for RTOS and application data
- Simulated in our experimental setting by disabling caches in kernel mode
- LEON2 cache API implemented by ORK+ could be ported to our PPC target but was ouf of the scope of this work

Experimental results (kernel)

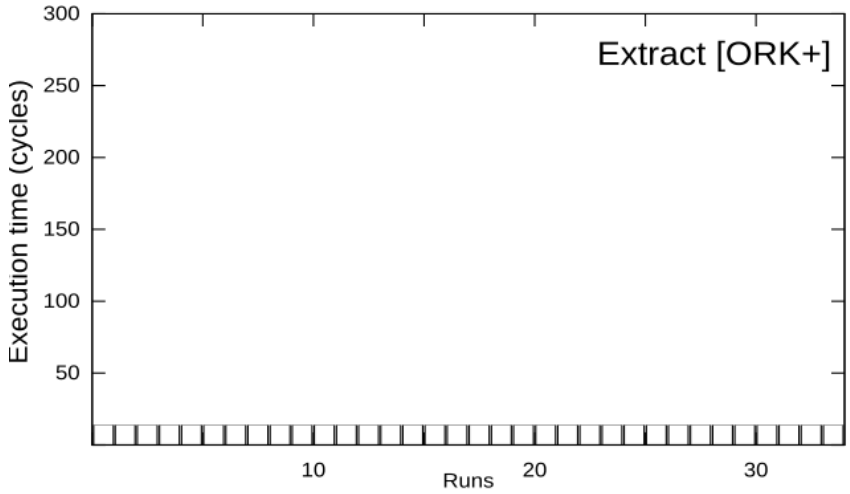
- Task insertion in ready queue invoked at task activation showed jittery behaviour according to the queue state
- Constant-time behaviour is achieved using bit masks and bitwise operations

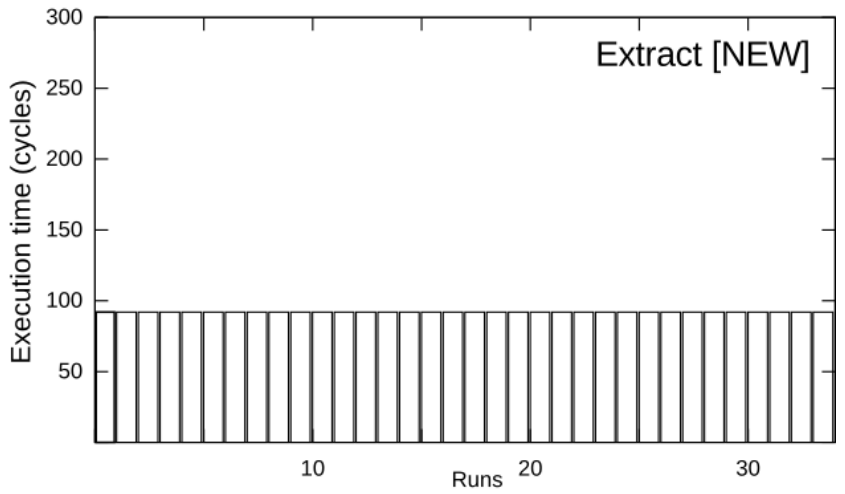




Experimental results (kernel)

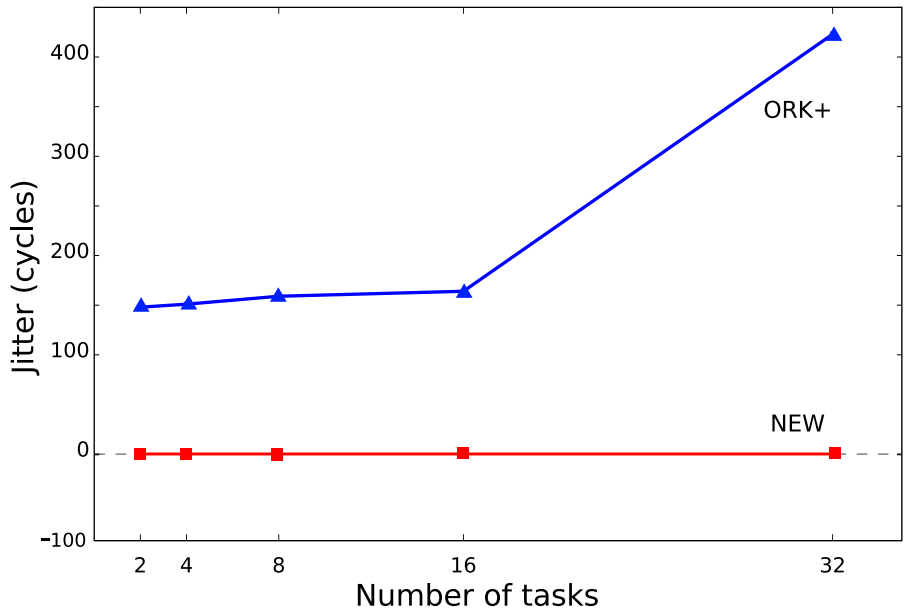
- Task extraction from ready queue invoked at task self-suspension required just dereferencing a pointer
- Our implementation needs to perform some bitwise operations, though the additional overhead is negligible





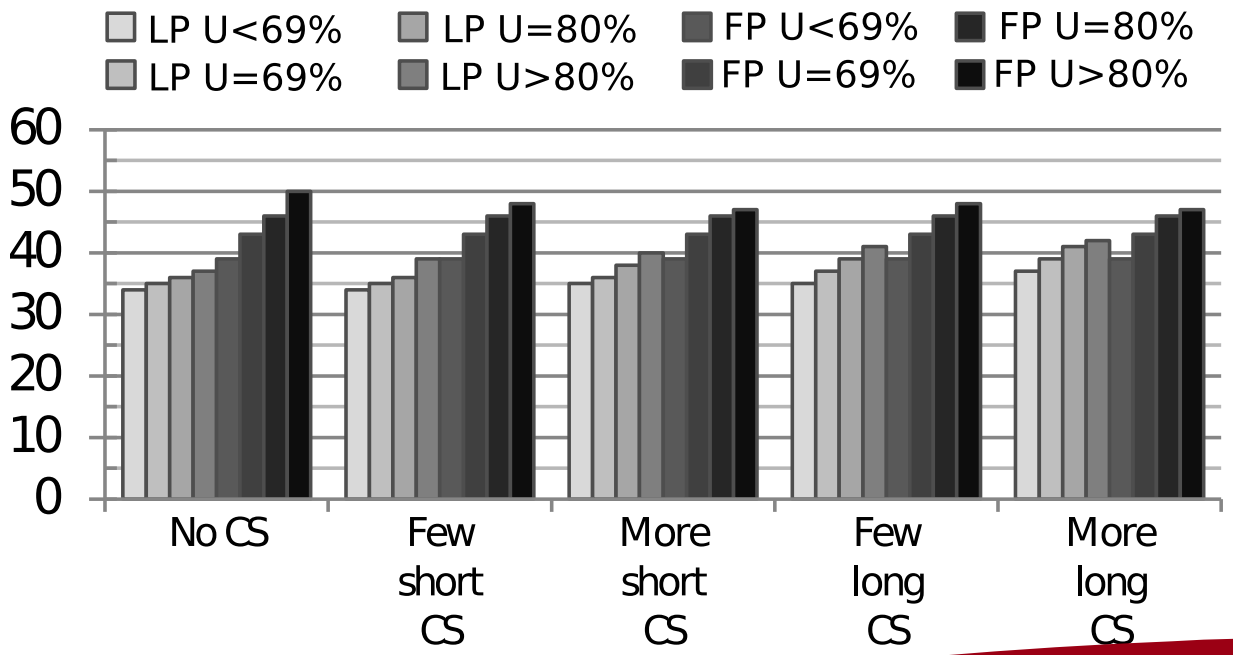
Experimental results (application)

- Synthetic application measured under different workloads, including end-to-end job and scheduling but no context switch overhead
- Constant-time execution is observed, as expected



Experimental results (limited preemption)

- The number of preemptions grows with higher utilization
- The number of preemptions is smaller with LP scheduling
- More preemptions with longer critical sections



Conclusions

- Application analyzability benefits from task model restrictions
- Careful RTOS design is mandatory to attenuate or eliminate the sources of unpredictability which eventually compromise timing analysis
- Open issue: limited-preemptive scheduling effectiveness depends on the particular task set
- Open issue: in a multicore setting composability is possibly harder to guarantee