PROXIMA

Measurement-Based Timing Analysis of the AURIX Caches

1 

2 

Leonidas Kosmidis[1,2] , Davide Compagnin[3], David Morales[2],
Enrico Mezzetti[2], Eduardo Quinones[2], Jaume Abella[2],
Tullio Vardanega[3], Francisco J. Cazorla[2,4]

3

4

16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)
Toulouse, France, 5th July 2016

This project and the research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7 / 2007-2013] under grant agreement 611085

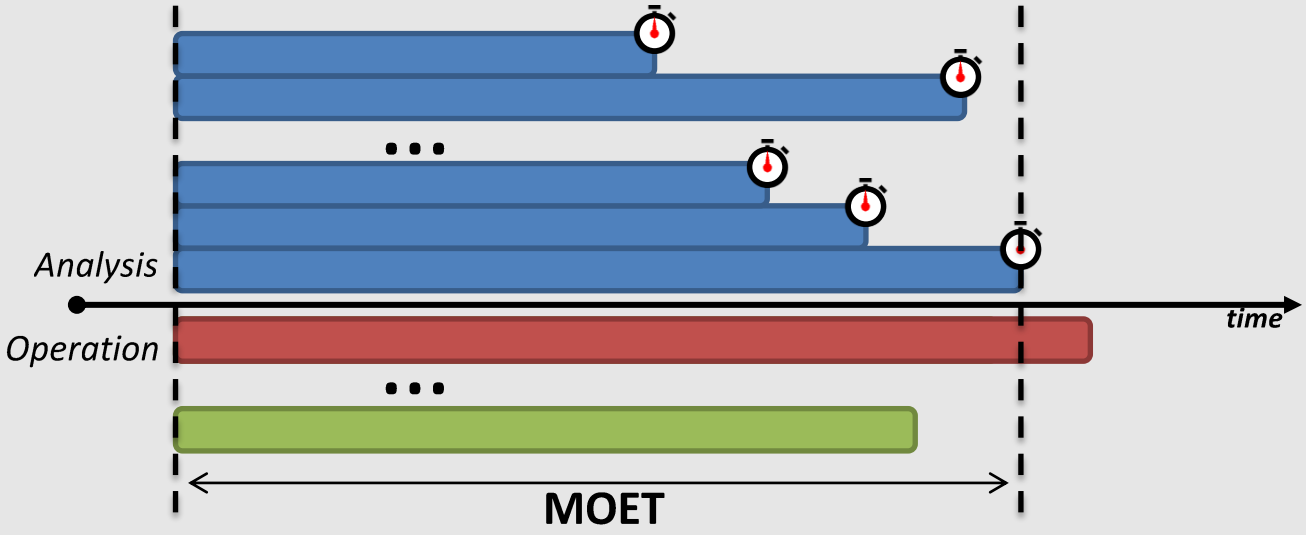
www.proxima-project.eu

Outline

- Motivation and scope
- Software-level Static Software Randomization
- Evaluation on Automotive case study
- Conclusions

Motivation

- Measurement-Based Timing Analysis of a **representative automotive application**
  - Based on system **analysis-time** measurements
  - Derive WCET estimates that hold at **system operation**
- *Representativeness* of observations
  - Guarantee that measurements taken at analysis time capture those events impacting execution time



MBTA representativeness

- Timing behavior is the result of a *complex interaction* of several relevant factors
  - Initial hardware state, input-data dependent execution, interference from underlying RTOS, memory and cache layout, etc.
  - Extremely difficult to control them all at analysis time
  - Equally difficult to design experiments to force “bad” events to simultaneously occur
- How to guarantee confidence on the results?
  - In a (industrially viable) cost-effective way
  - Without relying on
    - Strong assumptions on the quality and effectiveness of the test campaign
    - Provably unscientific safety margins

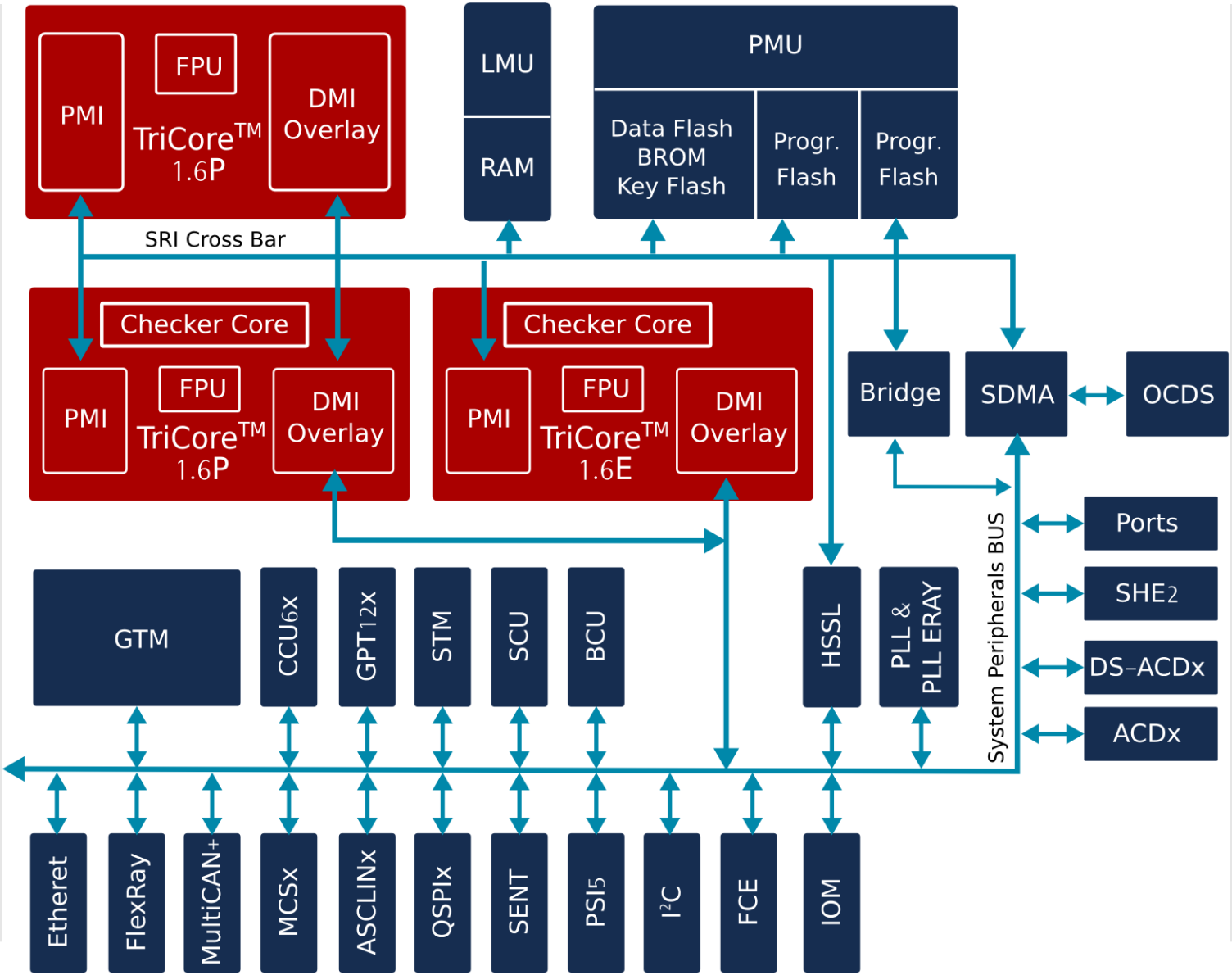
Focus on cache-induced variability

- Cache effects and induced variability
  - Memory mapping → cache layouts → cache conflicts → execution time
  - The effect of conflict misses in particular are typically difficult to assess and analyze
    - Very sensitive to small changes in the layout (recompilation, integration, etc.)
    - Difficult to capture and control in measurements
  - We are not after “optimal” memory and cache layouts
    - Optimal code and data placement NP-Hard
    - We are addressing “critical” scenarios
- Cache randomization and Meas.-based Probabilistic TA
  - Allows transparently exposing cache-induced variability
  - User can derive the probability of the observed mappings to be observed at operation
  - And the probability of unobserved mappings

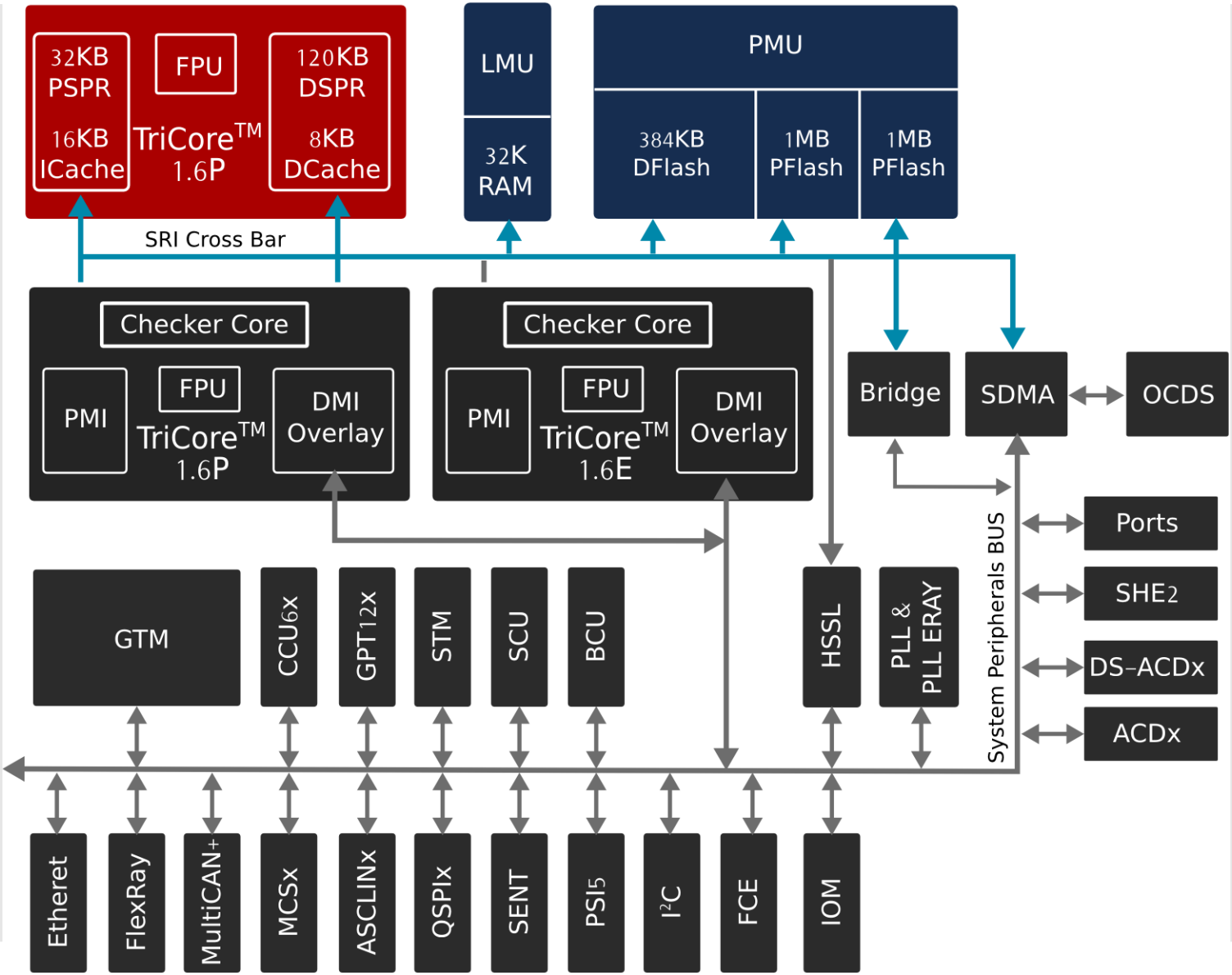
Focus on automotive reference platform

- AURIX TC27x has been designed to be deterministic (large scratchpads, optional cache usage, …)
  - Pretty predictable if me make ***diligent*** use of the platform
  - Industrial practice, however, can be extremely ***casual!***
    - Exhibit sources of variability that need to be coped with
- **Caveat**
  - We do not aim to provide an analysis method for the TC27x
  - We show instead how sources of variability can be characterized (e.g. caches, interconnects)
  - This may require deploying 'non standard' software setups, for instance to place code/data in caches even when the best practice could suggest not do it
    - We are not after "optimal" setups from an application perspective, but rather any setup that helps us to expose cache variability

AURIX TC27xT architecture



AURIX TC27xT architecture



Exploiting memory randomization

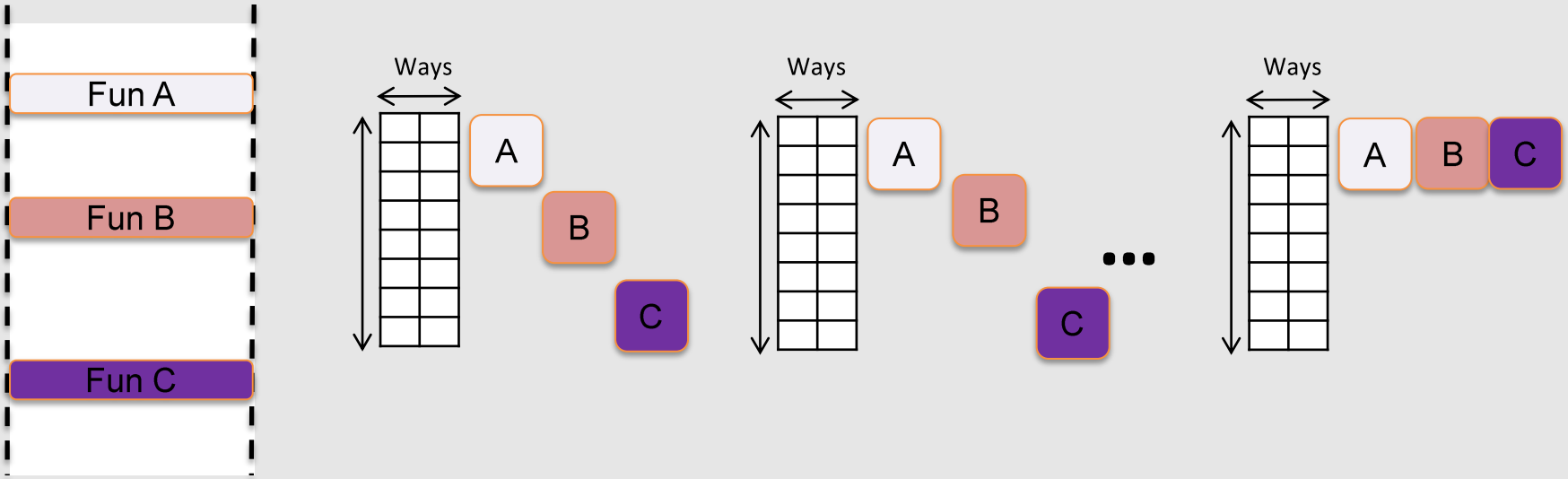
- Randomization to better characterize the behavior and take it into account in WCET analysis
  - Exposes jitter effects caused by cache memories
    - For a comprehensive characterization of cache-induced variability
  - Makes the system better amenable to MBPTA
    - Supports MBPTA requirements and improves representativeness
- How cache variability is exposed
  - Placing memory objects (functions, stacks, globals) in random memory locations, **across distinct executions**
  - **Program objects are allocated to random sets across runs**
- This may happen
  - By Hardware (random placement and replacement policies)
  - Dynamically: at program load time and during execution
  - Statically: at compilation time, before the program runs

Static Software Randomization

- For the AURIX domain of application
  - No HW support to random policies
  - Memory layout of the program cannot be changed at run time
  - Randomization can only be static
- Source-code level static SW randomization (SL-SSR)
  - Only flavour of SW randomization compatible with the AURIX
    - Memory relocations are **not** allowed at run time
  - The location of memory objects is randomized by moving their declaration in the program source
- SL-SSR in practice
  - Source-to-source compiler
  - No changes to the system
  - Portable across different platforms/tool chains

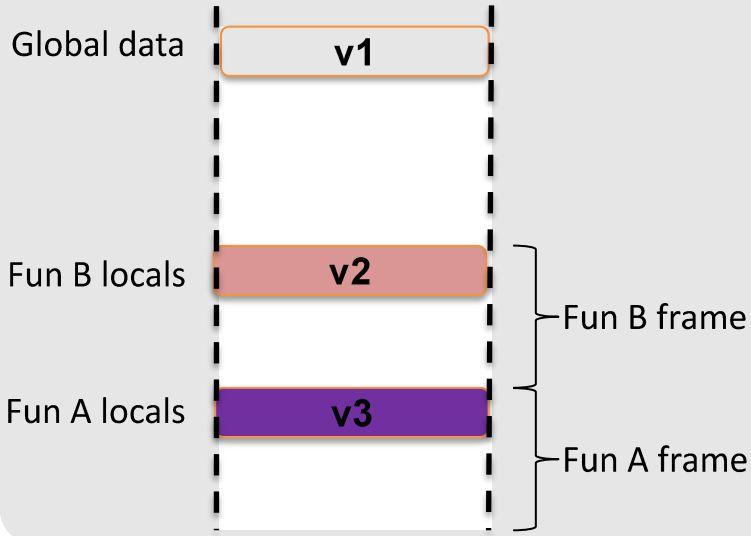
Randomization of code and data

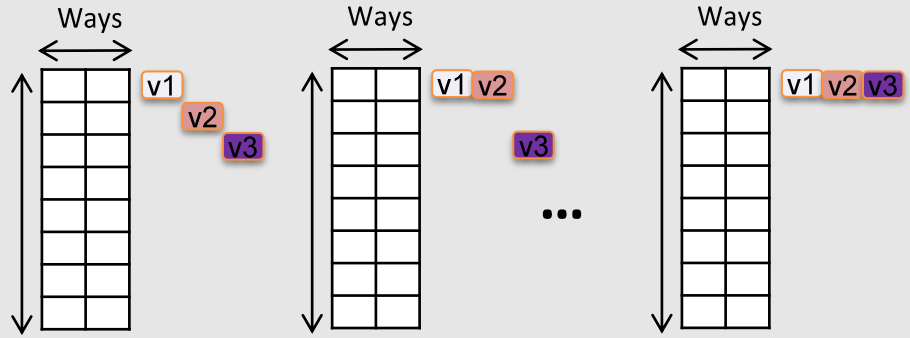
- Randomly reordering functions and global data
  - Affects memory layout
  - Induces different patterns for conflict misses



Randomization of stack frame

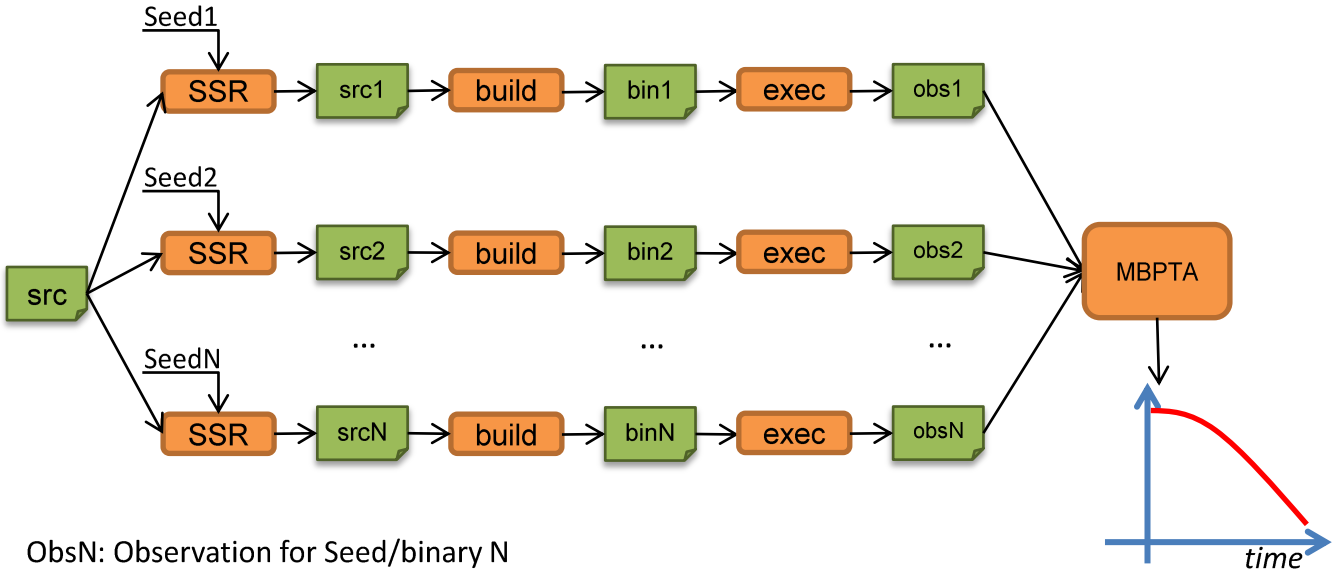
- Size of stack frame depends on size of local variables
  - Compiler-enforced alignment paddings
    - Variables (and stack) needs to be aligned according to their size
- SL-SSR uses two complimentary solutions
  - Inter-stack randomization (between different stack frames)
    - Artificially increase the stack frame size, by a random amount
  - Intra-stack randomisation (between local variables)
    - Reorder local variable declaration order





Evaluation

- Evaluated our approach on a representative setting
  - AURIX TC277T Board
  - Excerpt from automotive application
  - Erika Enterprise OSEK/VDX Compliant RTOS <http://www. http://erika.tuxfamily.org//>
- Generated and collected observations over a large set of binaries



ObsN: Observation for Seed/binary N

The CONCERTO application

- Cruise Control System
  - Automatically generated from a Simulink model

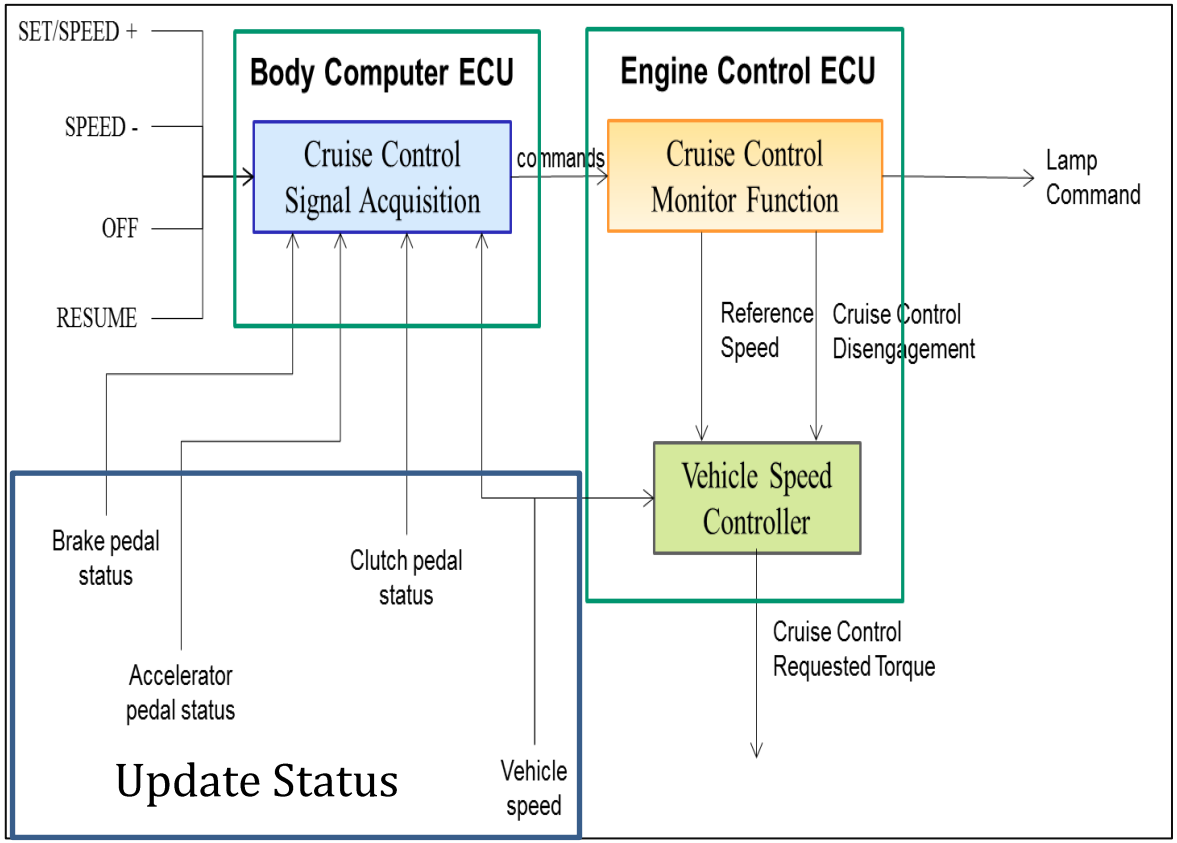
CONCERTO
ARTEMIS JU FP7 Project
<http://www.concerto-project.org/>

The application includes 4 tasks, with precedence relation

1. SignalAcq
2. Monitor
3. SpeedCtrl
4. UpdateStatus

The application is assigned to Core 1

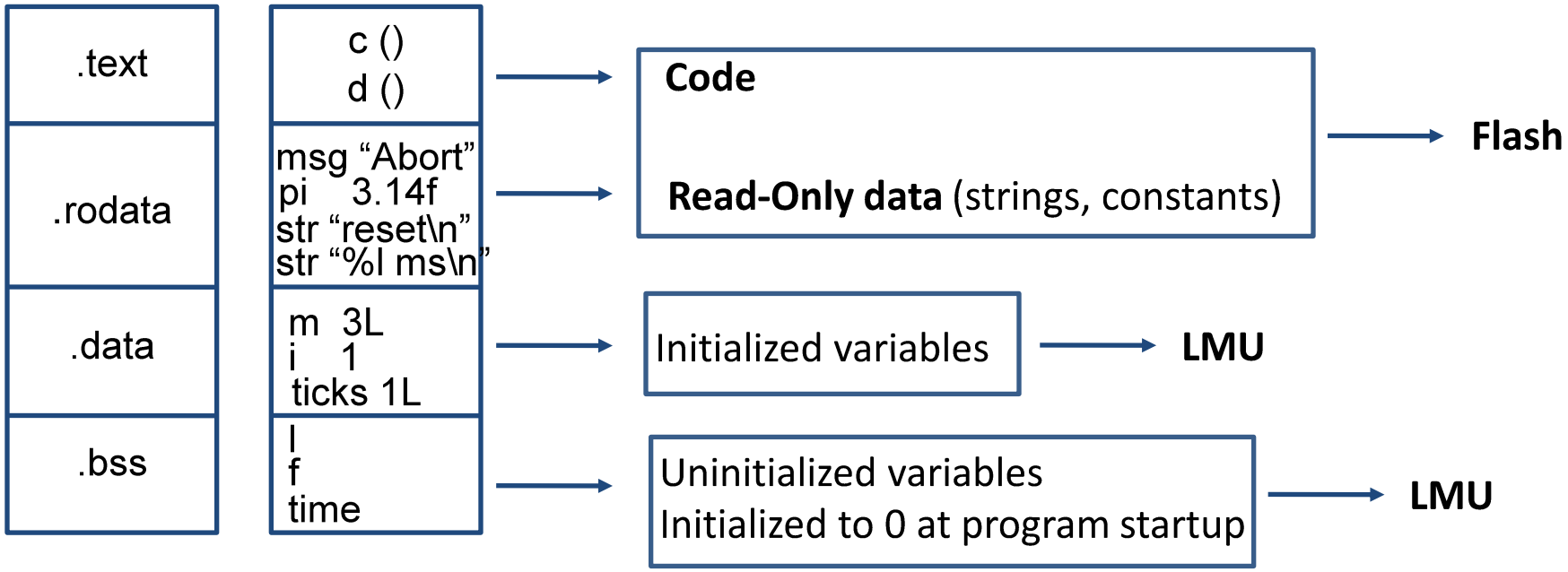
Code and constants mapped to PMU
Data mapped to SRAM



PROXIMA

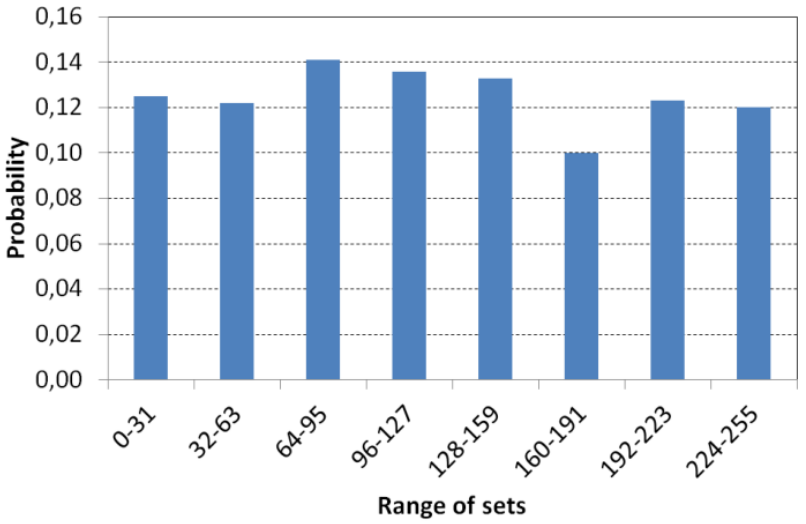
SL-SSR on CONCERTO ELF binary

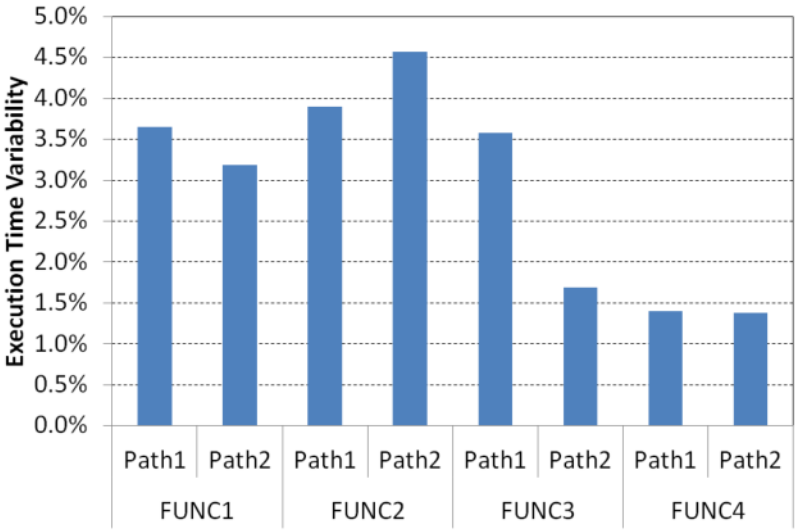
- ELF widespread binary format
  - SL-SSR can be used with any other format
- Main sections: text, .rodata, .data,.bss
  - Loaded in memory before program start up
- Specific configuration enforced through RTOS



Effectiveness of SL-SSR on CCS

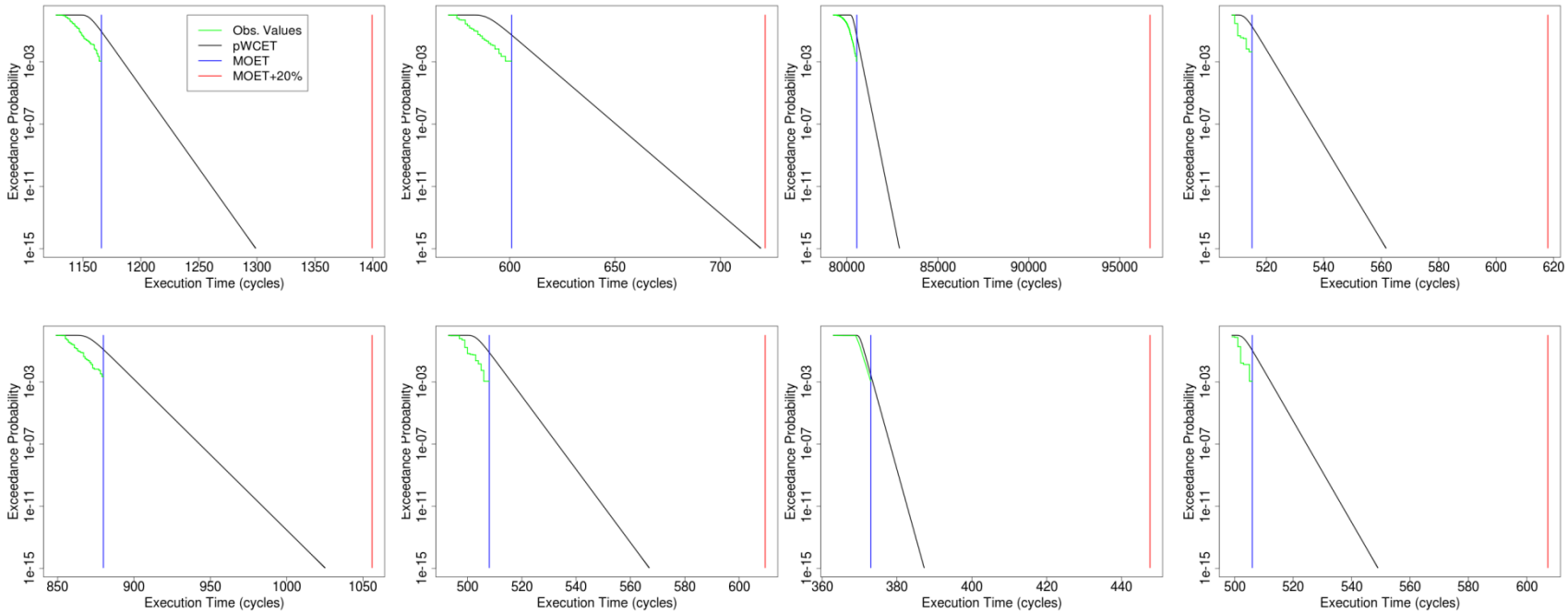
- Effectiveness of SL-SRR
  - How often the fist address of one of the main functions is mapped to a group of cache sets
  - Converges to homogeneous distribution (~1/8 = 0.125)
- Induced cache-related variability
  - Focus on longest and shortest paths
  - Variability does come from memory randomization
    - Almost no variability when SSR is not in place





Capturing cache variability with MBPTA

- User does not need to explicitly control the cache layout
  - Analysis-time variability upperbounds that at operation
    - Attaching probabilistic guarantees to WCET figures
  - Execution time variability achieved (i.i.d tests passed)



Capturing cache variability with MBPTA

- Tightness of results on CCS
  - As compared to MOET values
  - At pWCET at 10^{-12} exceedance risk

	Cut-off Probability 10^{-12}	
	Path1	*Path2*
Function 1	11%	9%
Function 2	**6%**	10%
Function 3	15%	**17%**
Function 4	12%	14%

- Very close to observed values
- Always below the typical 20% industrial safety margin

Conclusions

- SL-SSR as a means to effectively characterize how cache may affect the timing behavior of a program
- Benefits on sensitivity to cache behavior
  - Programs may be more or less robust to changes in the cache layout
  - This may happen even in those architecture that have been designed with predictability in mind
    - As long as the user is allowed to diverge from the intended usage
- Enabled an effective application of MBPTA to automotive application
  - Improved representativeness
  - Provided tight results in the analyzed application
    - pWCET estimates strictly comparable to observed values
    - Largely below the typical 20% industrial safety margin
- Next steps
  - Comprehensive evaluation over a larger class of programs
  - On different hardware architecture

More on SL-SSR

- Leonidas Kosmidis, Roberto Vargas, David Morales, Eduardo Quiñones, Jaume Abella, Francisco J. Cazorla
  - "*TASA: Toolchain-Agnostic Static Software Randomisation for Critical Real-Time Systems*"

    To appear in International Conference On Computer Aided Design (ICCAD) 2016, November 7-10, 2016, Austin, TX