RAPITA Systems Ltd
A DANLAW Company

Multi-core Timing Analysis

An *industrial approach* for multi-core airborne software timing verification using DO-178C/CAST-32A

Ian Broster, Christos Evripidou (Rapita)
Francisco Cazorla, Enrico Mezetti, Suzana Milutinovic (BSC)

Multi-core processors (MCP) are (really) here

- They are available and lots of people trying to use them in critical systems

- Big issue:
  - "How much does an application on one core affect another?"
  - Software timing/*contention*

- Our solution: new MCP timing analysis process and tooling
  1. Methodology
     - V-model with t*raceability to objectives in CAST-32A*
  2. Technologies for timing tests
     - *Microbenchmarks, contenders, RapiTime, Automation*
  3. Certification evidence
     - *Tool qualification and traceable process*

Industrialization of research in numbers

3 Technologies

RapiTime

RapiTest

4+ Research Projects

ASC (NATEP, UK)

SDESI – (Catalan, Spain)

SECT-AIR – (ATI UK)

PROXIMA – (EU FP7)

(and a few older ones)

6 Industrial stakeholders

RAPITA SYSTEMS LTD A DANLAW COMPANY

BSC

ALTRAN

ROLLS ROYCE

BAE SYSTEMS

3 real projects ongoing

X

Y

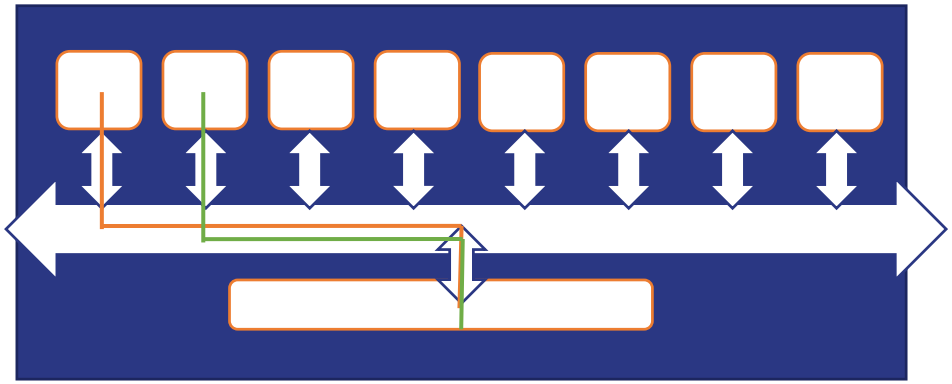
Z

Challenges with multi-core Processors

What's so hard with multi-cores?

Each core can influence other cores

⇒ Timing/delays/contention



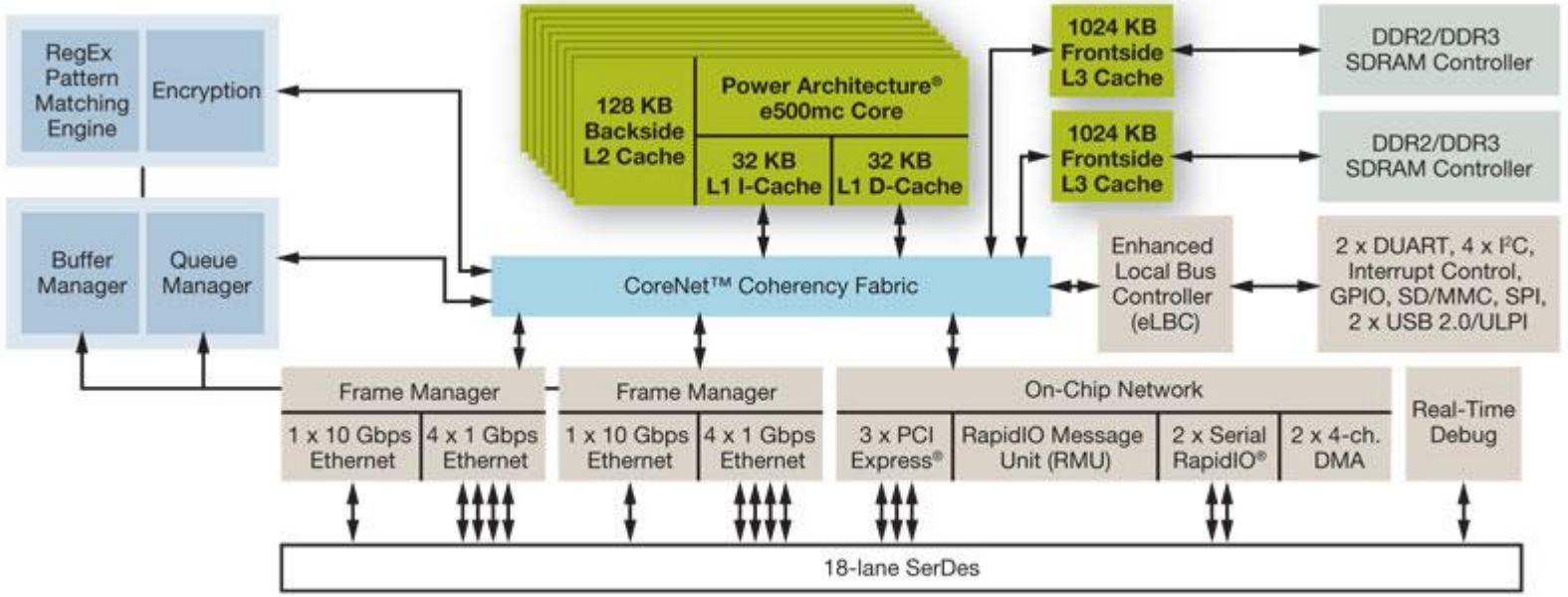
FAA/CAA rationale for using multi-cores [CAST-32A]

- Potential performance improvement
- Obsolescence of single-core

FAA/CAA: CAST-32A

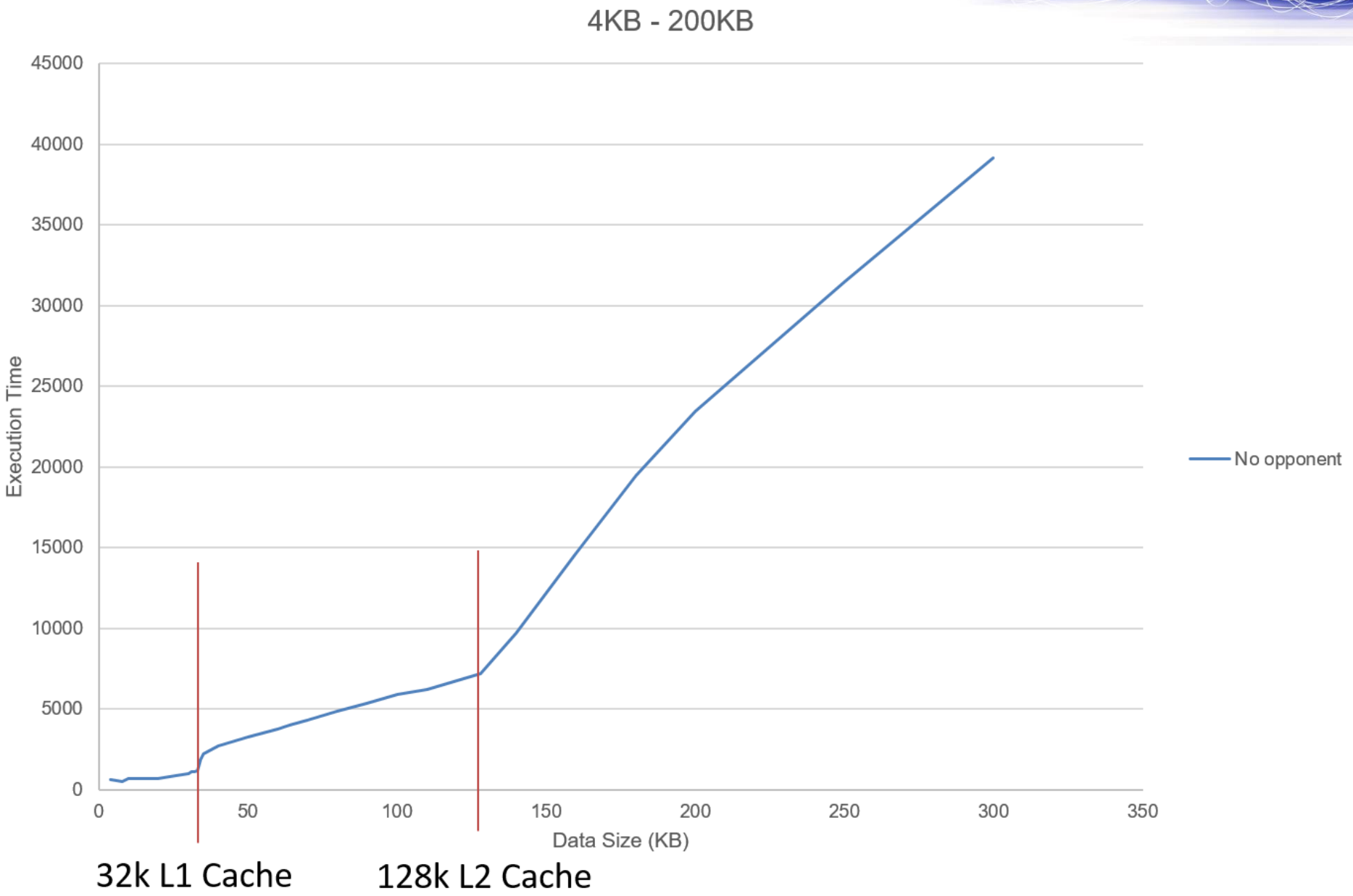
- From CAST-32A (paper from FAA & others)
  - *"several applications may therefore attempt to access the same shared resources of the MCP (such as memory, cache and external interfaces) at the same time* ***causing contention for those resources****"*
  - *"..interconnects to handle and arbitrate the demands for MCP resources, but the contention for shared resources between applications usually* ***causes delays in access to resources****…"*
  - *"There* ***could be functional interference*** *between applications via MCP mechanisms"*
  - *"…to behave in a non-deterministic or unsafe manner, or* ***could prevent them having sufficient time to complete the execution*** *of their safety-critical functionality".*

QorIQ™ P4080 Block Diagram

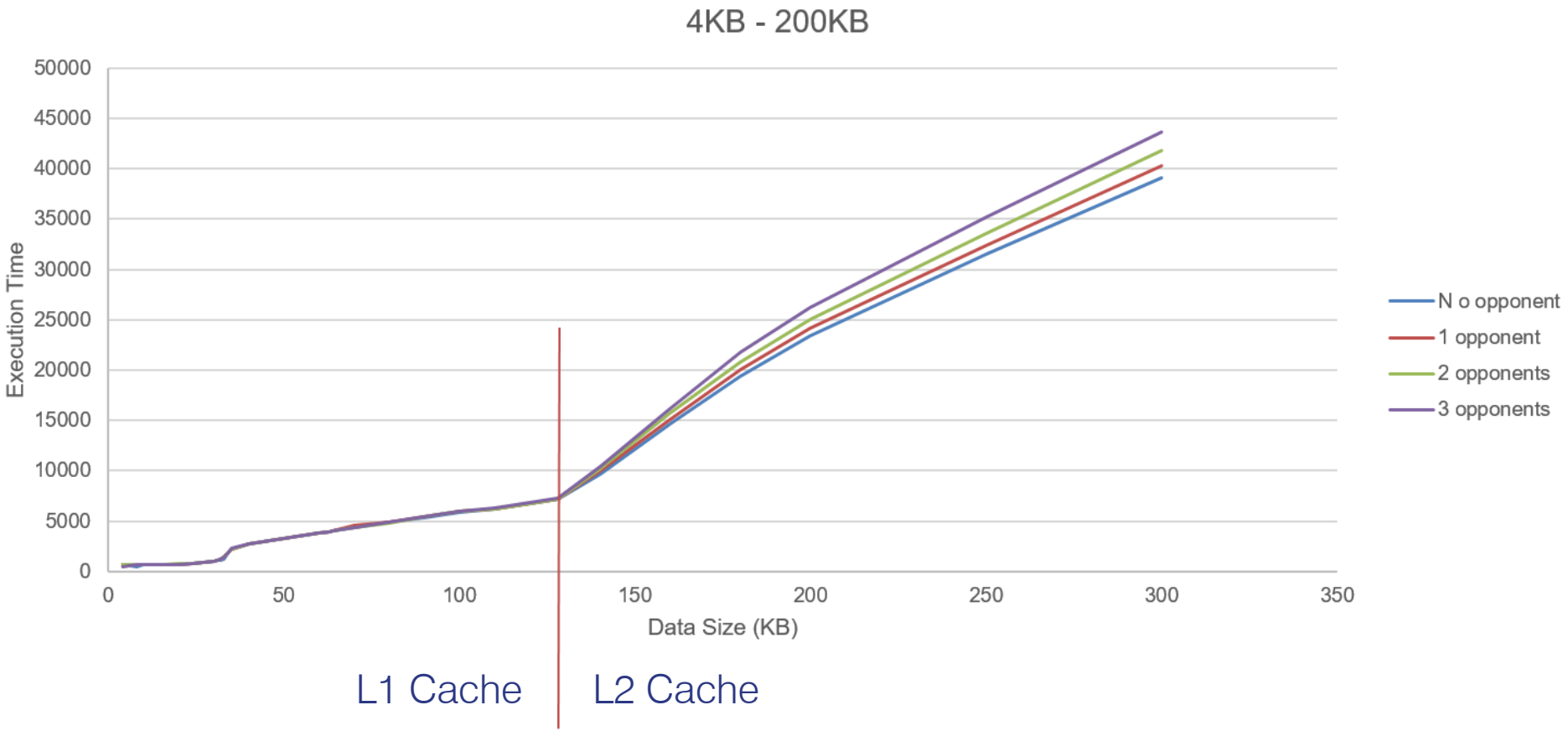


An experiment on Cache Impact

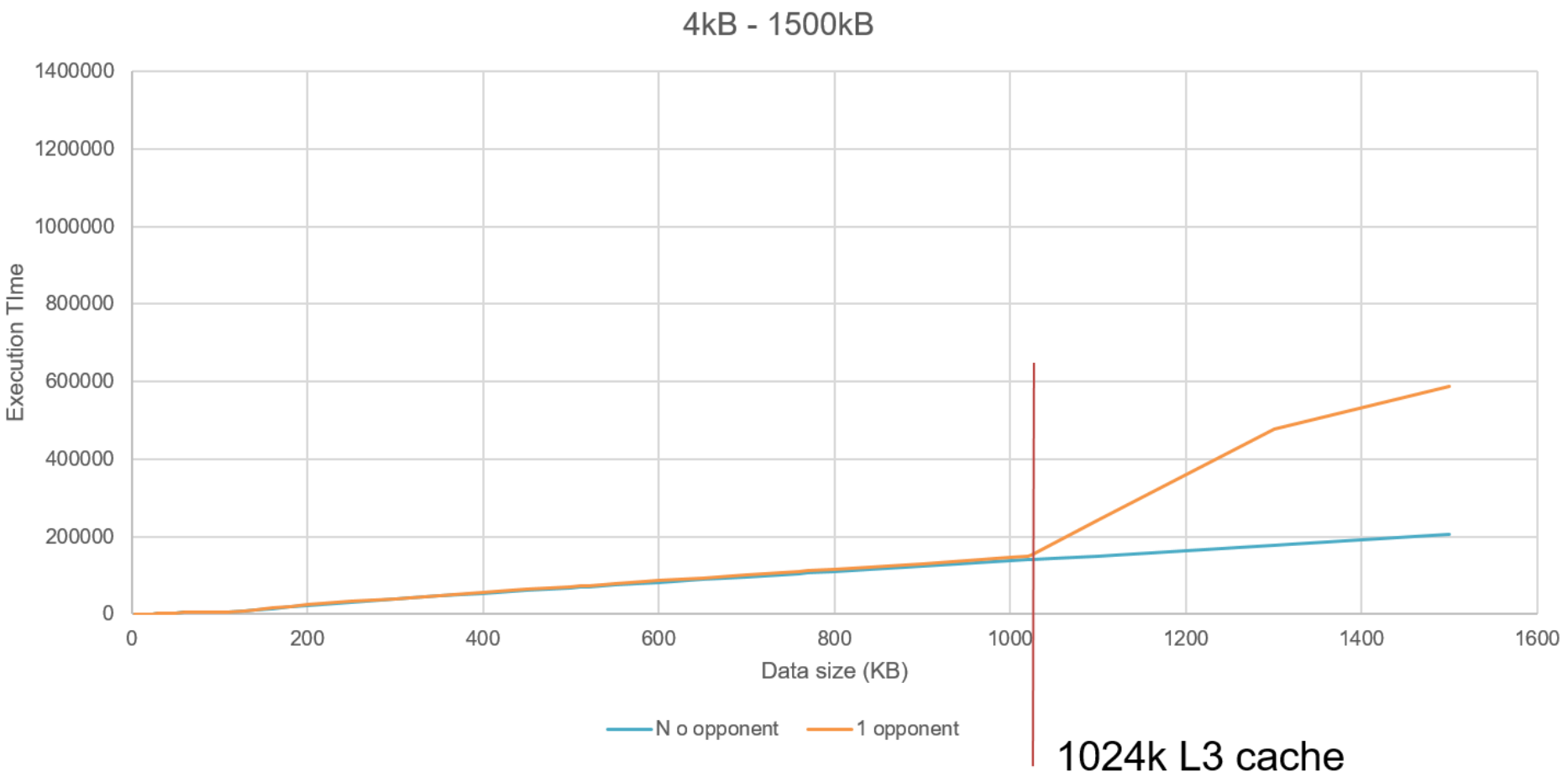
- Application reading x Kb of data from memory on one core
- Opponents also reading data from memory on some other cores
- Used RVS for timing measurements
  - Integration with PikeOS PROXIMA tracing mechanism
- Parameters:
  - Size of data
  - Number of **opponents** enabled
- P4080 Memory architecture


4KB - 200KB
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
Execution Time
0
50
100
150
200
250
300
350
Data Size (KB)
No opponent
32k L1 Cache
128k L2 Cache

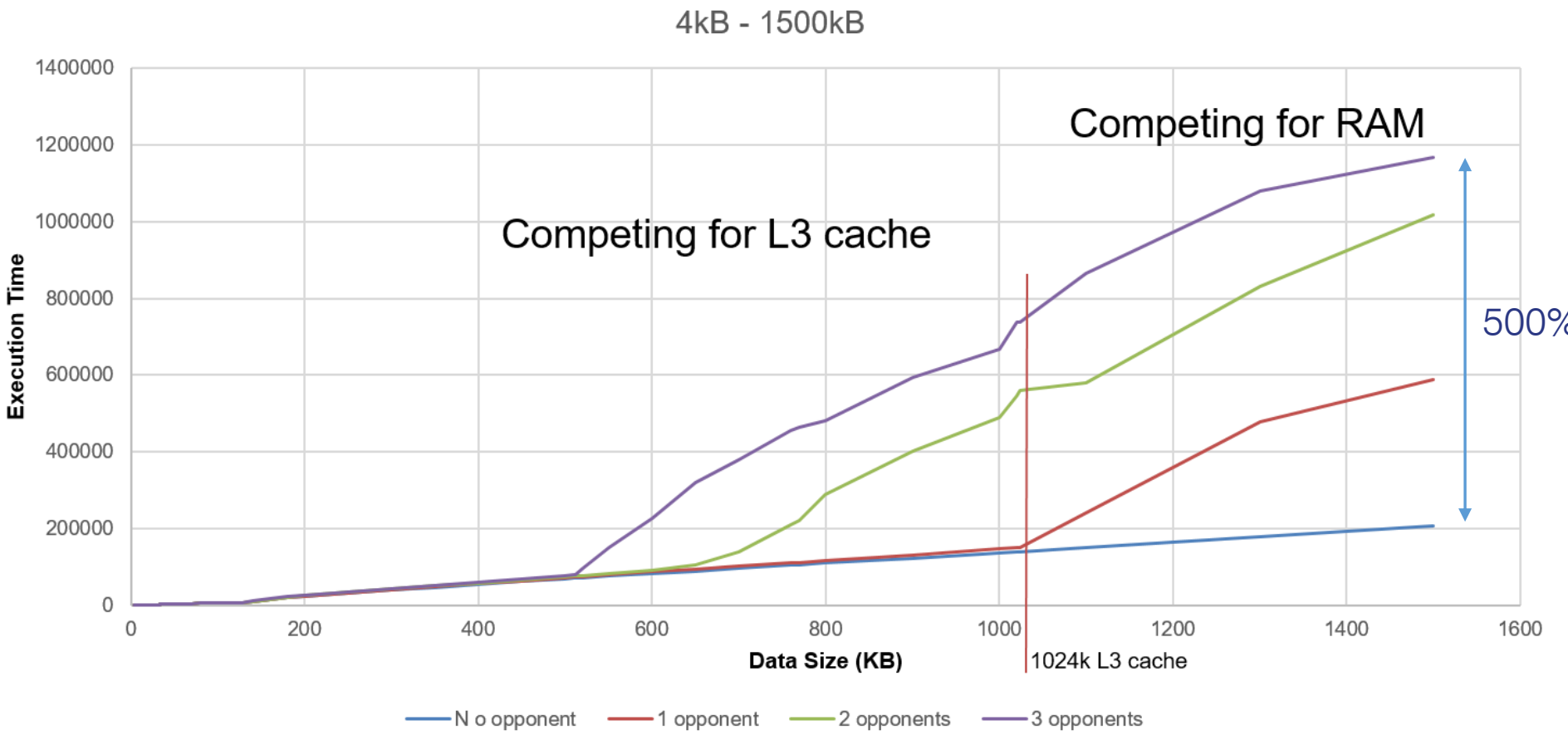
Adding **Opponents on other cores**



Larger Data size with an opponent



Big data read with 1, 2, 3 Opponents

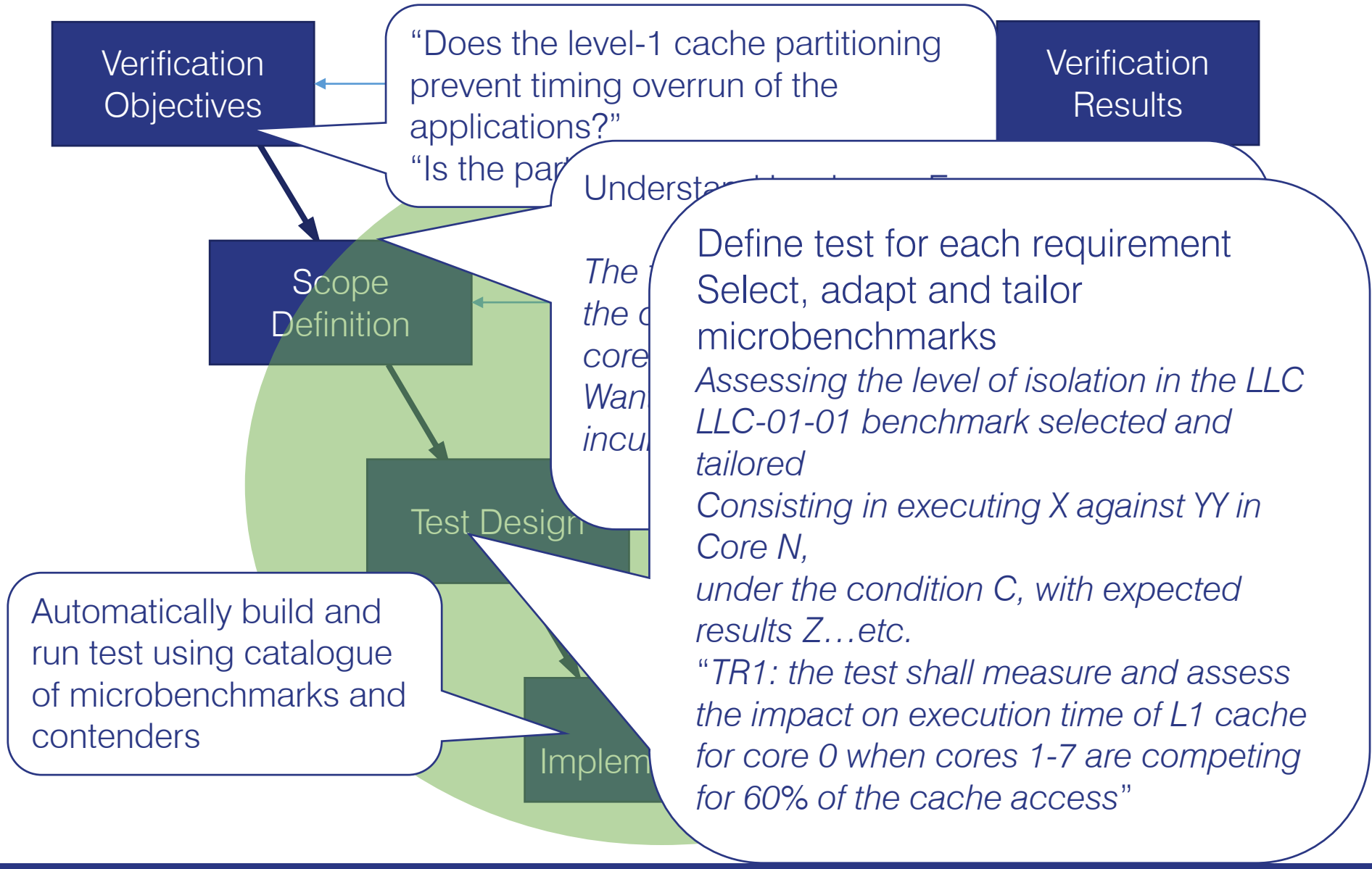


Multi-Core Methodology

Principles for MCP/WCET timing analysis

- P1: "The best model of a processor is the processor itself"!
  - No cost-effective static model of the processor
- P2: “Measurements, measurements and more measurement (and analysis)”.
  - Need to have “built-in” observability channels
  - Define minimum set of what are you looking for
  - Combine measurements with static analysis
- P3: “You shall trust no-one”.
  - Verify all key assumptions by evidence based analysis
  - More detailed for “key” contributors
- P4: “No tool (one-button) solution exists”.
  - Engineering wisdom for problem formulation and analysis of results
  - Embedded experts on experimental design and running tests
  - Automation of the evidence generation phase

Multi-core timing analysis for DO-178C



MuBT

- Multi-core Microbenchmarks
  - Characterise Multicore timing
  - Identify interference
  - Verify HW/RTOS/SW assumptions

- Developed by BSC + Rapita
- Result of EU project collaboration over 8 years

Multi-cores in real-time systems: opportunities and challenges

Multi-core processors are becoming the baseline computing solution in critical embedded systems. While multi-cores allow high software integration levels, hence reducing hardware procurement and SWaP (Space, Weight and Power) costs, use of multi-cores challenges current practices in timing analysis.

When planning to use multi-cores, critical real-time system practitioners face the following challenges:

> How to limit the impact that contention in multi-cores' shared resource may have on the application's execution time.

> How to provide evidence of correct multi-core timing behaviour for certification/qualification purposes.

BSC's microbenchmark technology

BSC's mμBT speeds up multi-core adoption. mμBT consists of a set of specialized user-level benchmarks that put high load on multi-cores' shared resources. By running BSC's microbenchmarks against your reference application under analysis, you will get an accurate measure of the impact that resource contention may have on your application's timing behaviour.

mμBT comprises a validation loop that works with Performance Monitoring Counters (PMCs) to provide evidence that the microbenchmarks achieve their intended goal in stressing different processor resources.

mμBT can be tailored for a wide set of multi-core processors. BSC also offers over ten years of experience in multi-core contention analysis to help you to achieve the requirements of certification authorities for the use of multi-cores. For avionics, BSC provides its experience to accomplish CAST32A recommendations in identifying contention (interference) channels, establishing hardware setups that limit contention and providing evidence of the degree of isolation.

BSC's microbenchmark technology can be used not only for performance but also for energy/power analysis. Architectures analysed using BSC's mμBT include: the IBM POWERX family of processors; ORACLE Niagara T2; Cobham Gaisler's NGMP; and the AURIX TX277x family.

Contact

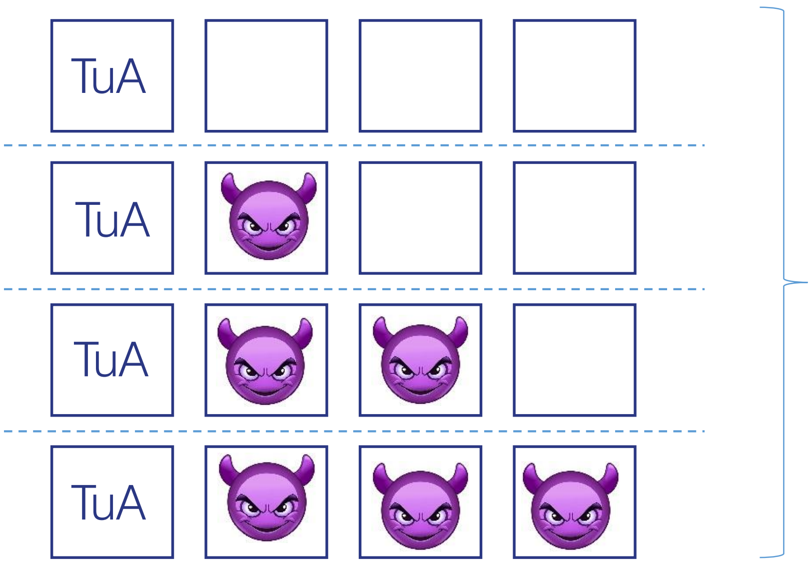
For more information on microbenchmark technology and consultancy service, contact:

Francisco J. Cazorla
francisco.cazorla@bsc.es
Director of the CAOS group

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

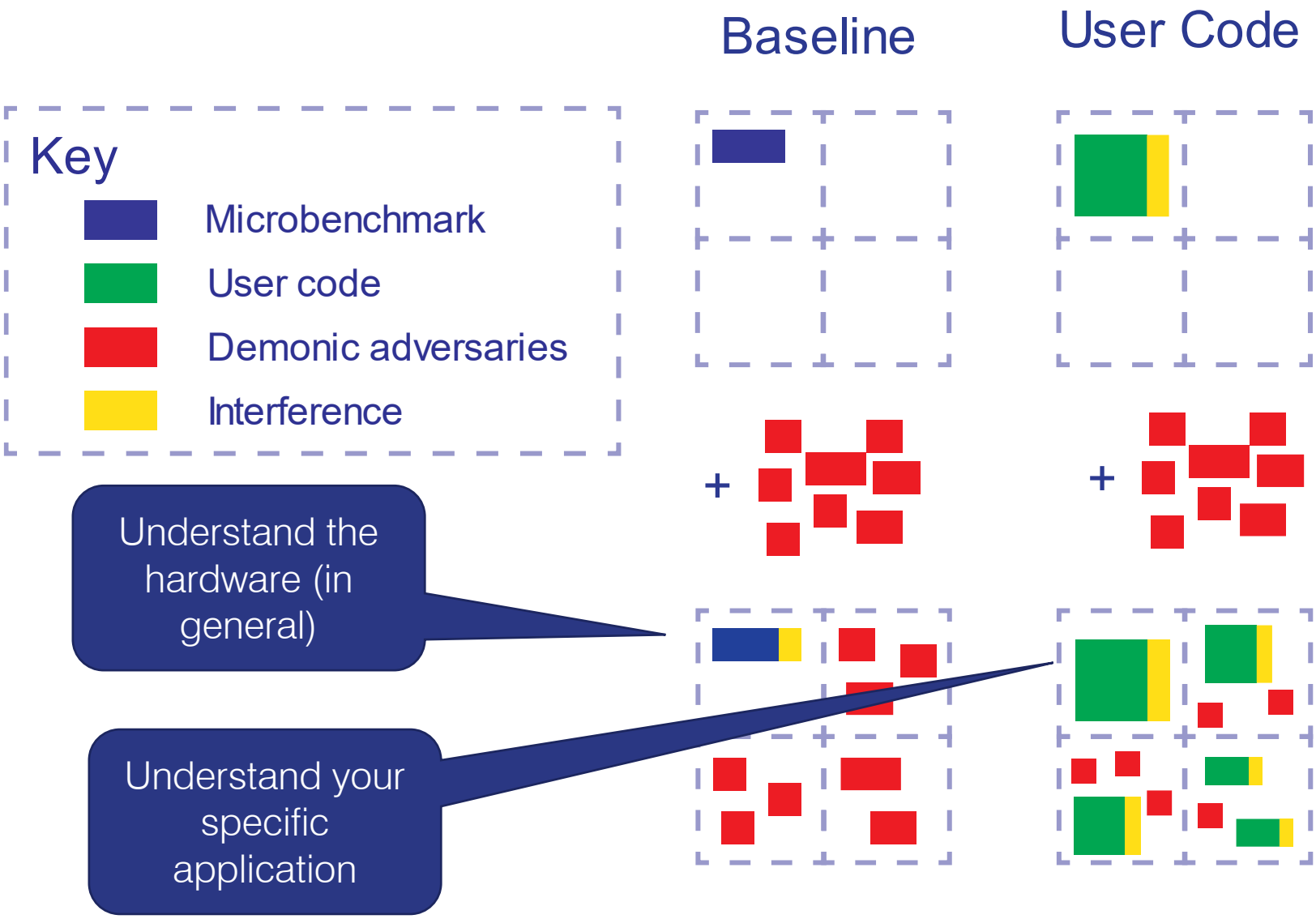
Microbenchmarks and contenders

- Each test is implemented as a set of microbenchmark configurations.
  - Use of performance monitoring counters (PMC)
    - Have a library of PMCs and tests (and ways to measure)
  - **Opponents, Contenders**, “demonic adversaries”
    - Have a library of contenders to exercise specific parts of the MCP
  - Automatic configuration of combinations and execution of tests

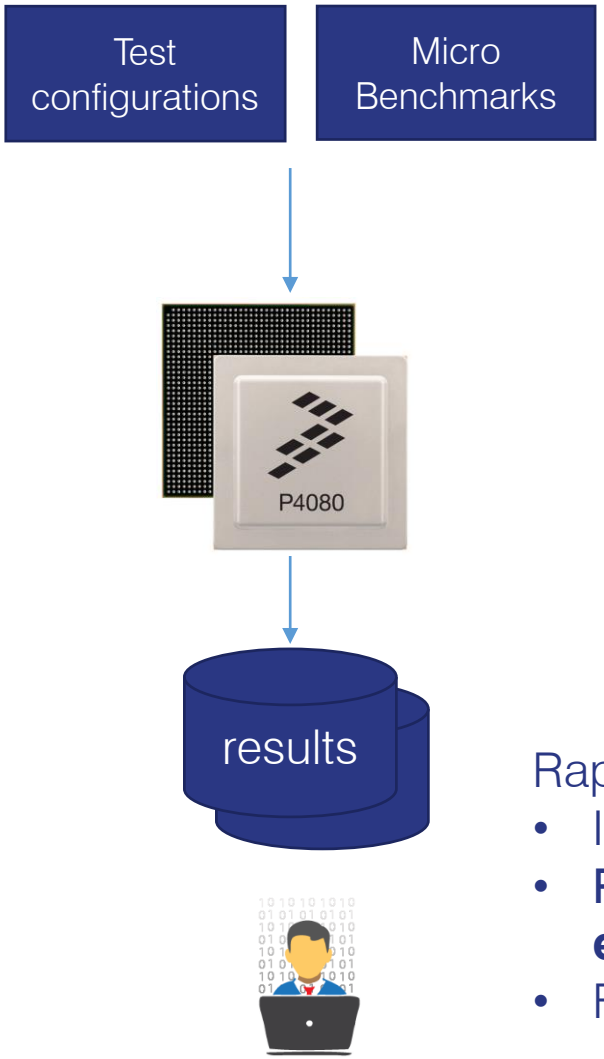



```
X        0                              \\ Entering
HEADER
0        123                            \\ µB UID
0        1                              \\ µB
configuration UID, End of HEADER
10       1315645   \\ IPoint Timestamp
0        1                              \\ PMC UID
0        560                            \\ PMC read
11       1329650   \\ IPoint Timestamp
0        1                              \\ PMC UID
0        16944                          \\ PMC read
X        0                              \\ Entering
HEADER
0        123                            \\ µB UID
0        1                              \\ µB
configuration UID, End of HEADER
10       1315645   \\ IPoint Timestamp
0        1                              \\ PMC UID
0        560                            \\ PMC read
11       1329650   \\ IPoint Timestamp
0        1                              \\ PMC UID
0        16944                          \\ PMC read
X        0                              \\ Entering
HEADER
0        123                            \\ µB UID
0        1                              \\ µB
configuration UID, End of HEADER
10       1315645   \\ IPoint Timestamp
0        1                              \\ PMC UID
0        560                            \\ PMC read
11       1329650   \\ IPoint Timestamp
0        1                              \\ PMC UID
0        16944                          \\ PMC read
```

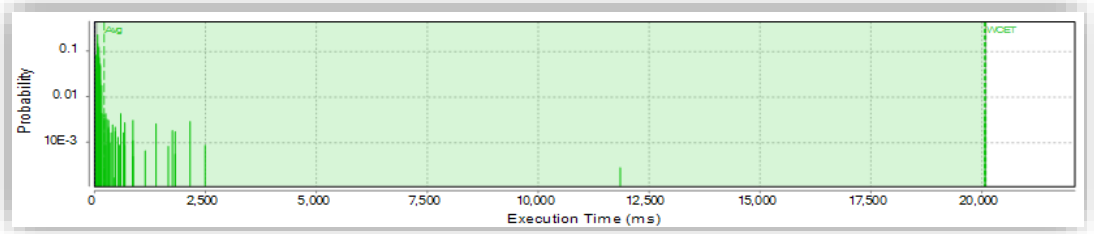
Test Design


Baseline
User Code
Key
Microbenchmark
User code
Demonic adversaries
Interference
+
+
Understand the hardware (in general)
Understand your specific application

Evidence gathering using RapiTime



Function	Min-ET		Avg-ET		H-ET		Max-ET		W-ET		
Name	Min-Freq	Min-SelfET	A-Freq	A-SelfET	H-Freq	H-SelfET	Max-Freq	Max-SelfET	W-Freq	W-SelfET	#Tests
nav_compute_baseleg	1	15,039	1.000	15K	1	15,040	1	15,040	1	16,721	255
nav_compute_final_from_glide	1	14,788	1.000	15K	1	14,789	1	14,789	1	15,149	255
nav_circle_XY	4	8,620	4.000	8,911	4	10,934	4	10,934	4	11,860	1,020



RapiTime:

- Is an **automated timing** analysis tool
- Provides **High watermark (HWM) & worst-case execution time (WCET) analysis**
- Finds **timing** problems and helps **optimize** your code

Multi[illegible]-178C

Generate multicore analysis report
Supporting certification arguments
Traceability to requirements and needs
Generated structured document with traceability info
Summarizing experiment plans design, implementation and results

Analy
Draw
Chec
Conc

Verification Results

Analysis

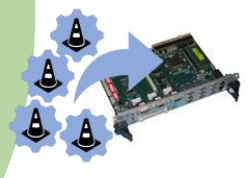
Execute tests and collect timing data (PMCs, timing)

Test Design

Test Results

Test Implementation

RapiTime

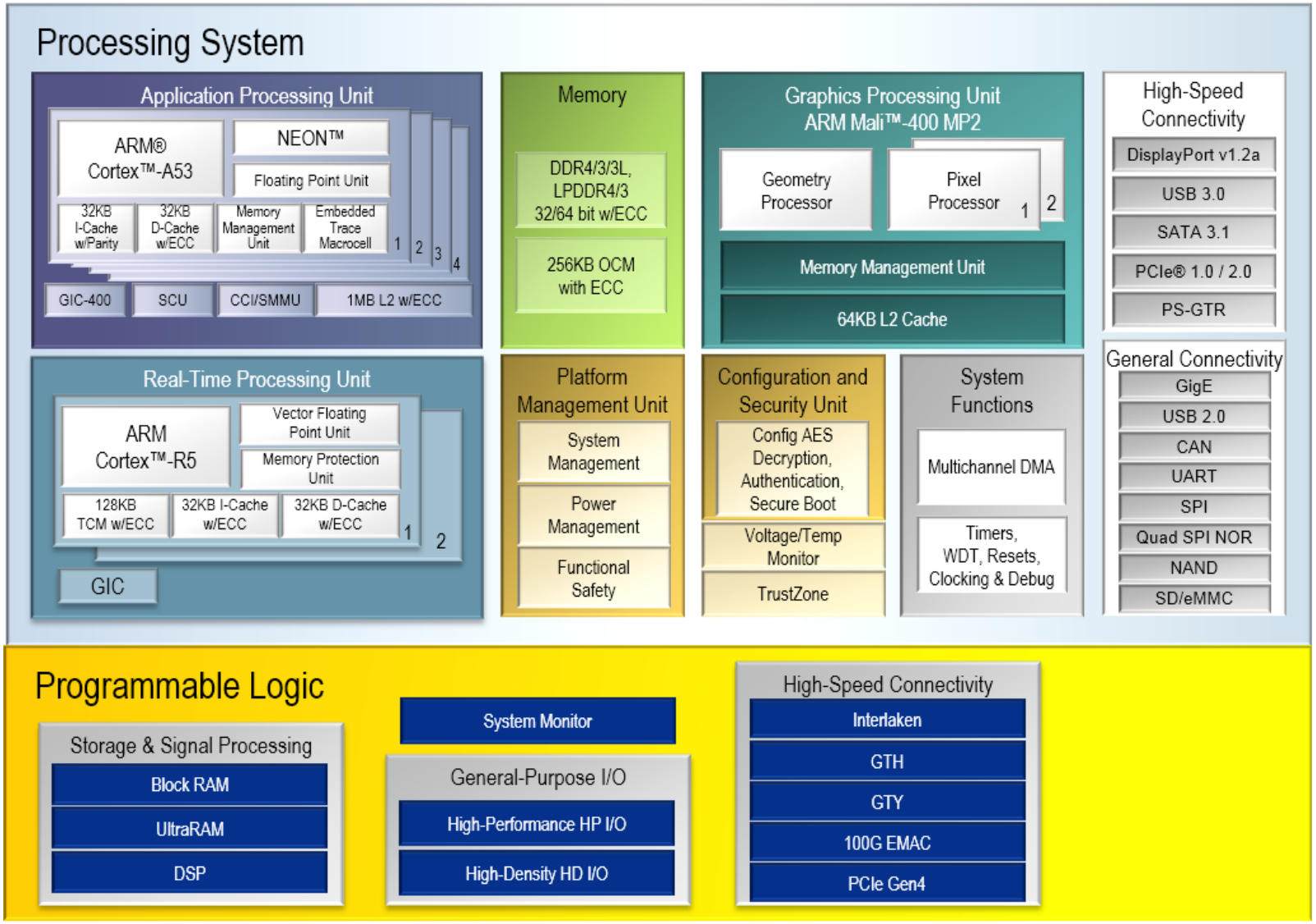


RapiTime

Tools, automation, experience

Multi-core Test Examples

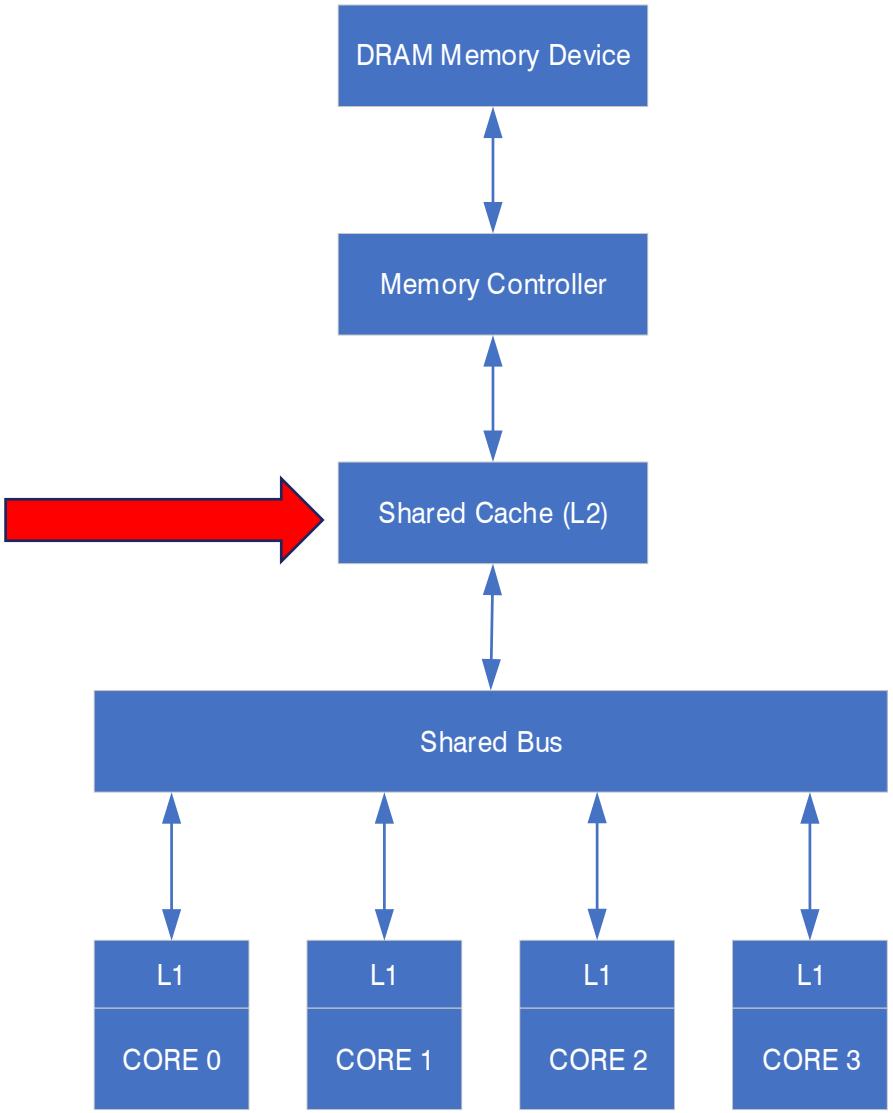
Zynq UltraScale+



Requirements and Understanding

- Requirement:
  - REQ-1234: The function muB_L2A_01 shall complete its execution in $6*10^{-8}$ cycles.
- Understanding:
  - The board is Zynq UltraScale+.
  - Platform support package: bare metal
  - muB_L2A_01 is a memory-intensive function.
  - muB_L2A_01 fits in L2 cache, but not L1 cache.
  - muB_L2A_01 is statically allocated to CORE 0.
- Interference channels
  - L2 Cache evictions

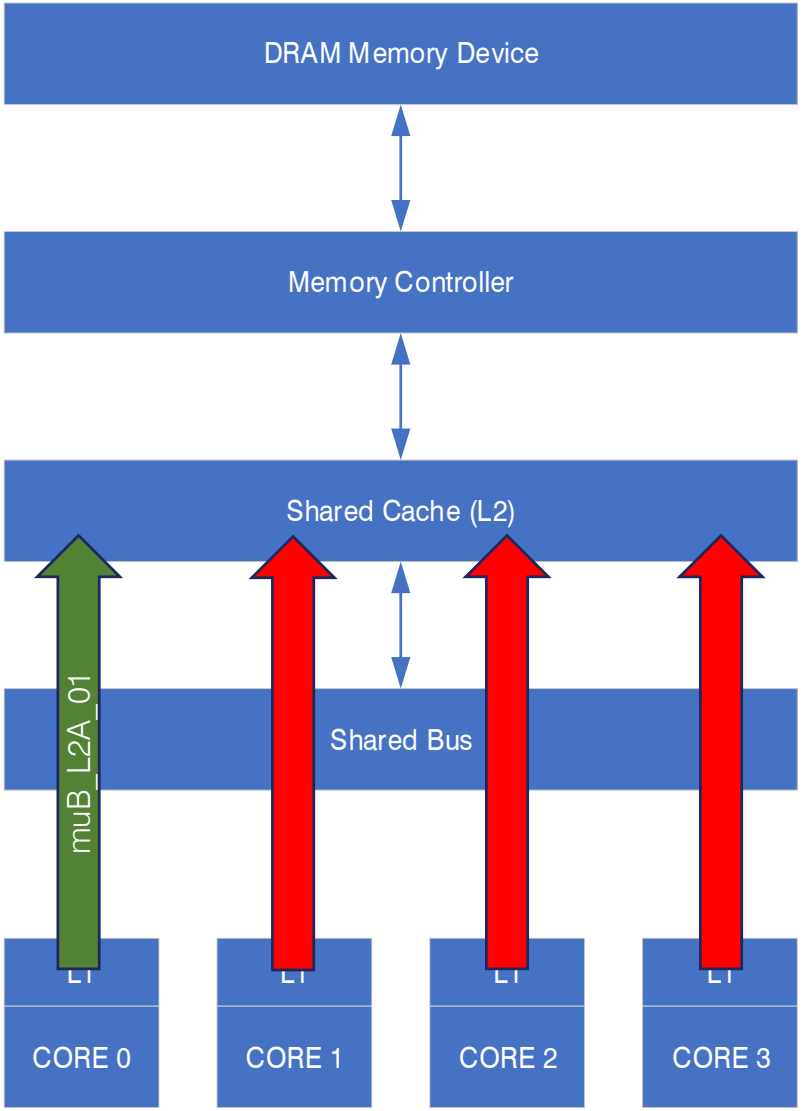
Simplified UltraScale+ Memory Hierarchy



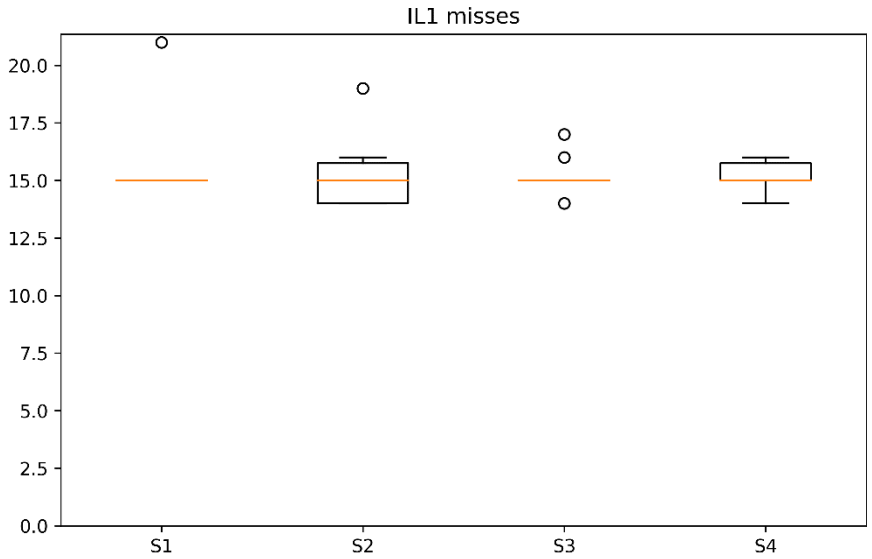
Test Design

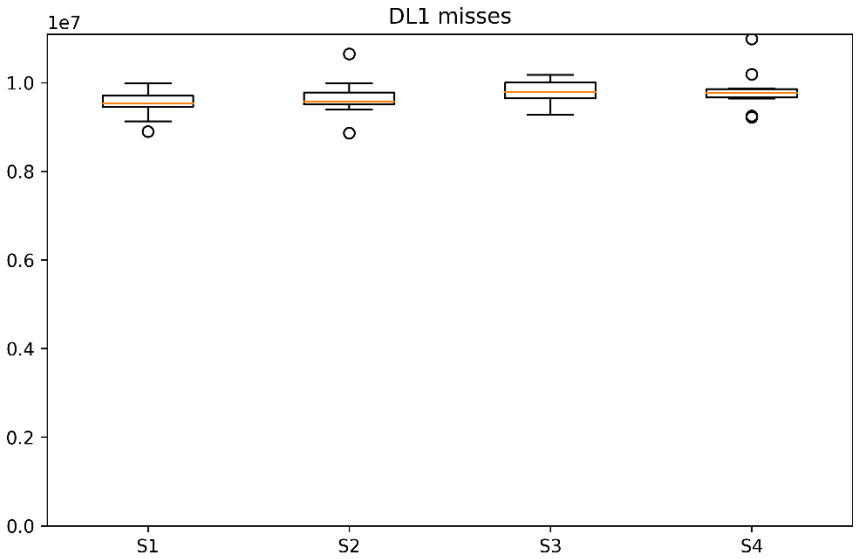
- Objective:
  - Identify the maximum L2 cache contention that ensures that the Task under Analysis (TuA) finishes within $6*10^{-8}$ cycles.
- Task Under Analysis: muB_L2A_01
- Required metrics:
  - L2 accesses
  - L2 misses
  - CPU cycles
- Added metrics: IL1 misses, DL1 misses, External memory accesses
- Repeat 10 times.
- Scenarios:
  - S1: Execute muB_L2A_01 with no resource contenders.
  - S2: Execute muB_L2A_01 with one L2 cache resource contender.
  - S3: Execute muB_L2A_01 with two L2 cache resource contenders.
  - S4: Execute muB_L2A_01 with three L2 cache resource contenders.

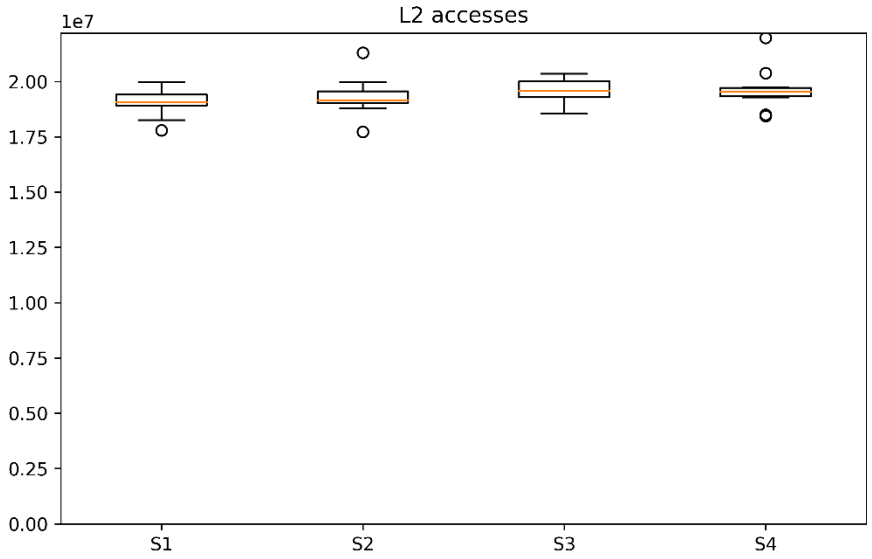
Simplified UltraScale+ Memory Hierarchy

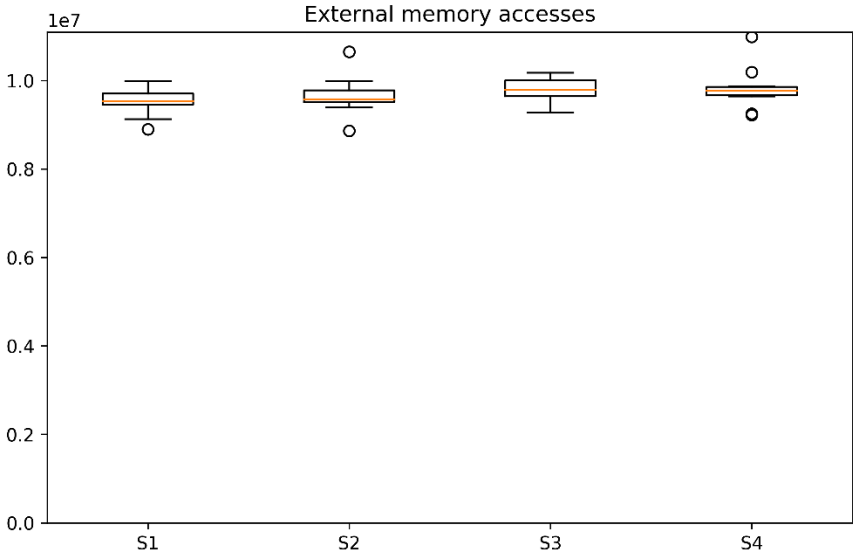


Task Under Analysis Characterisation

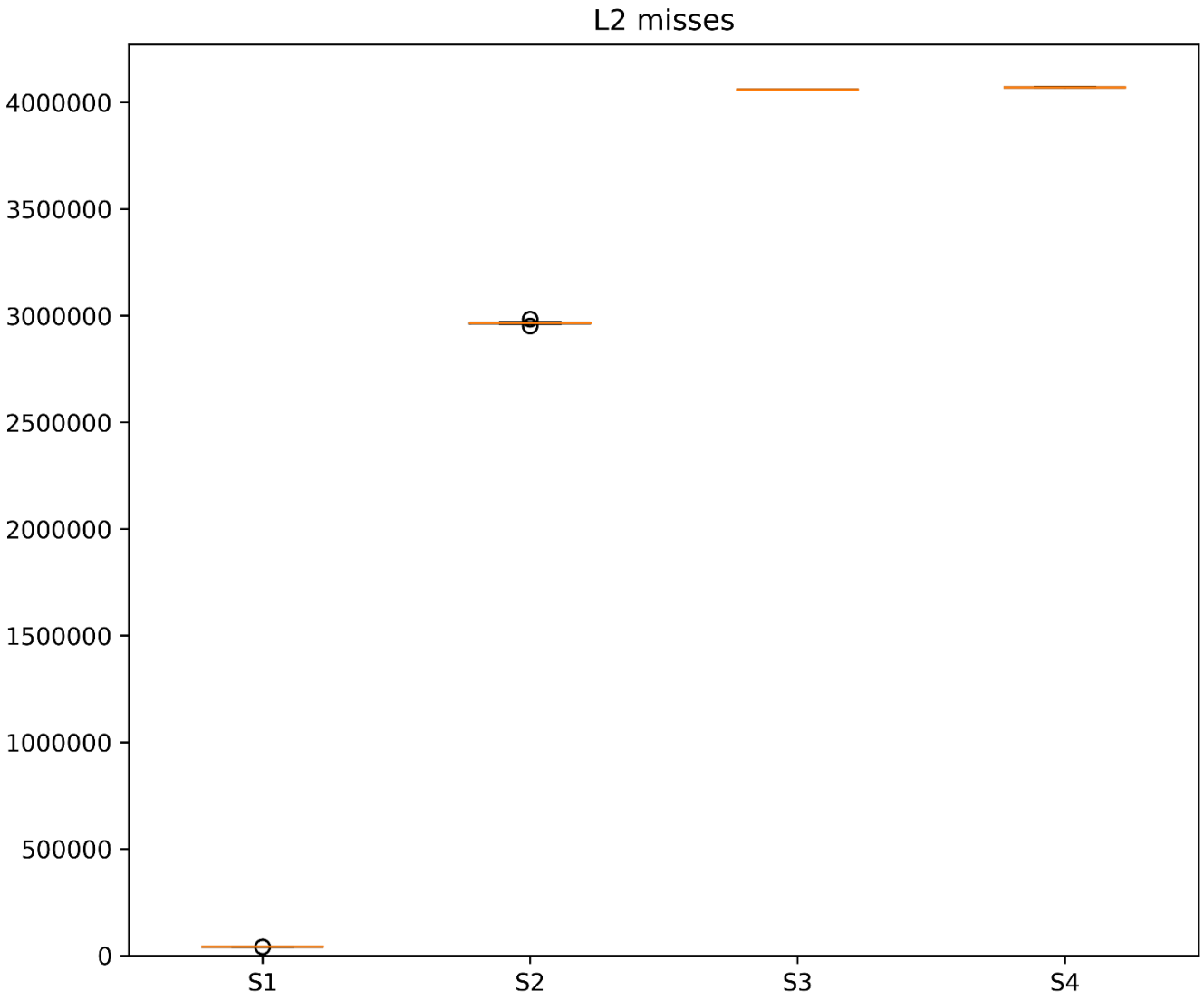




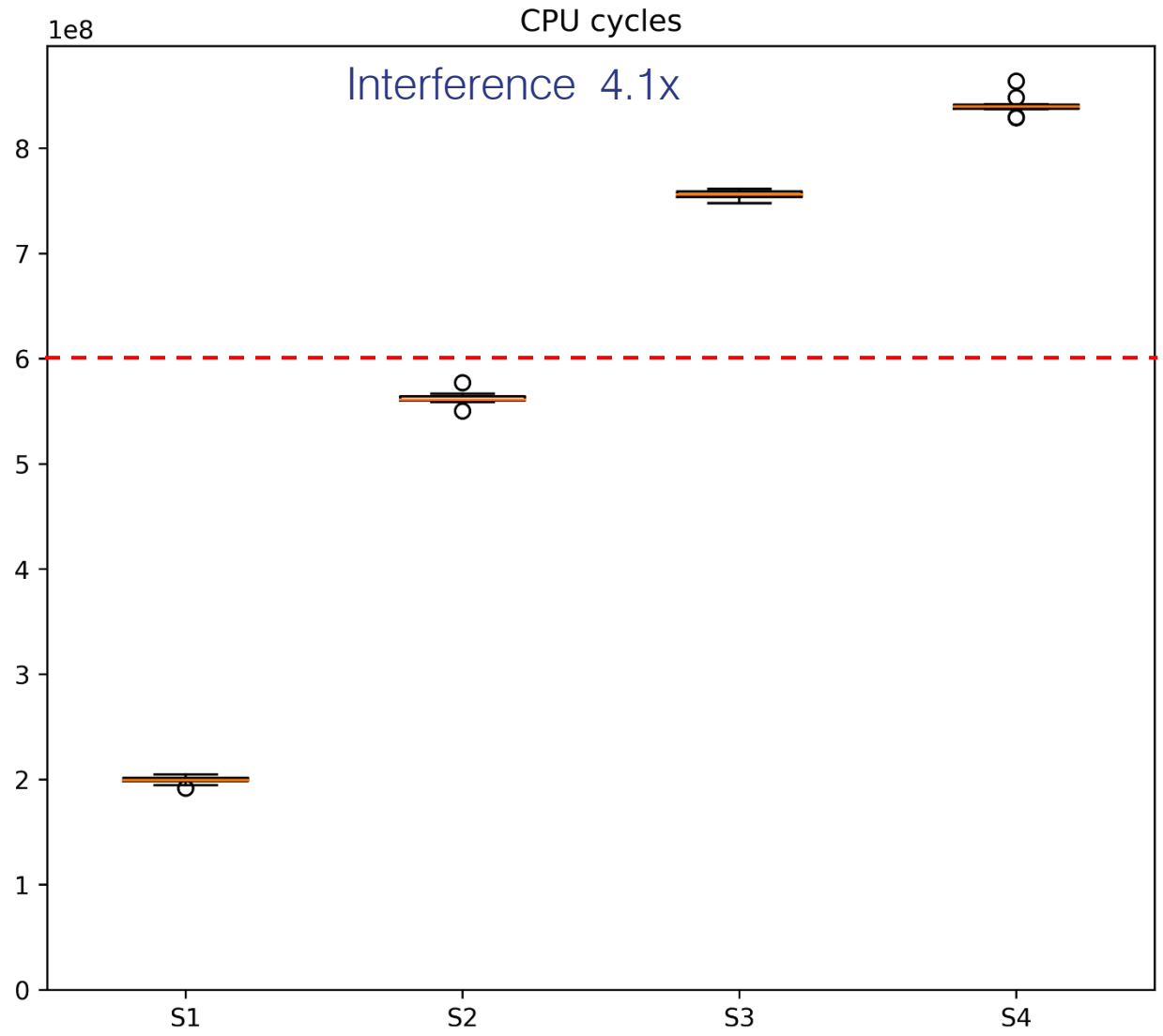




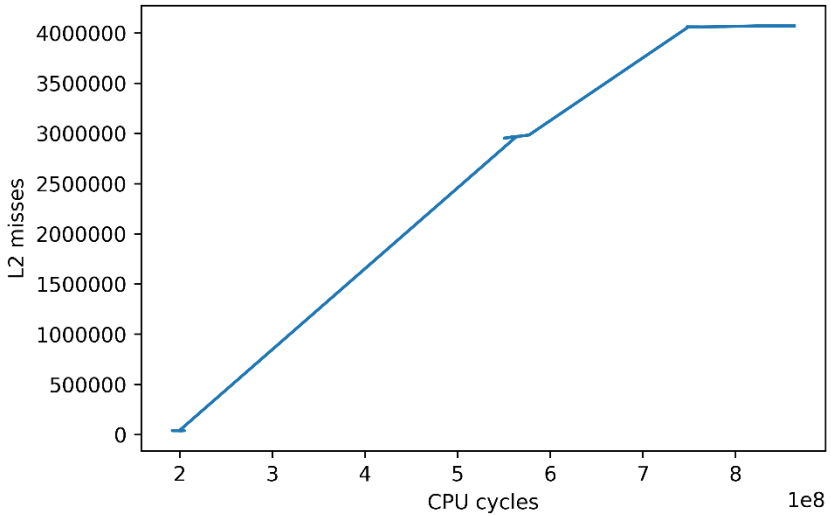
Contender Behaviour



Test Results



Test Results



- Strong linear correlation (up to L2 misses cut-off)
  - r=0.9844
  - p=2.79E-30
- Regression analysis:
  - Every L2 miss adds 147 CPU cycles as interference.
- New interference channel:
  - Contention on the bus

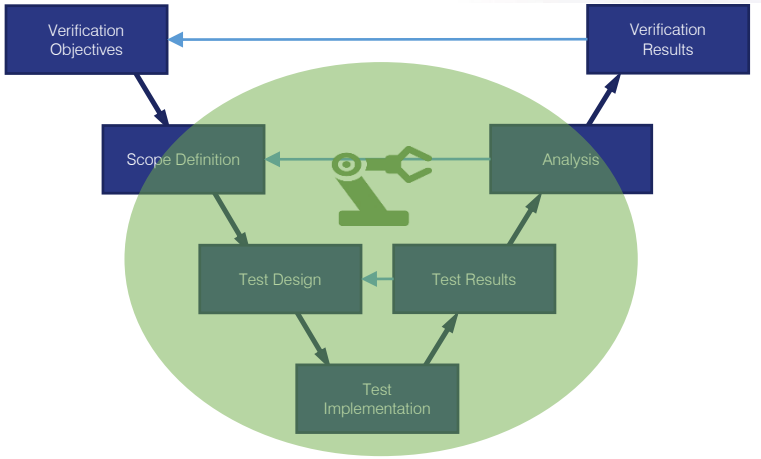
Industrial Solution and Direction

- Rapita and BSC offer this as a service
  - Currently doing the method for 3 customers
    - 2 are DO-178C

- What makes this an economic industrial solution:
  - Efficient tooling
  - Process and traceability
  - Corpus of library code, contenders and experiments
  - Expertise and lessons learned

Summary

1. Methodology (V-model)
   - *Traceability to objectives in CAST-32A*
2. Technologies for timing tests
   - *Microbenchmarks, contenders, RapiTime, Automation*
3. Certification evidence
   - *Tool qualification and traceable process*



3 Technologies

RapiTime

RapiTest