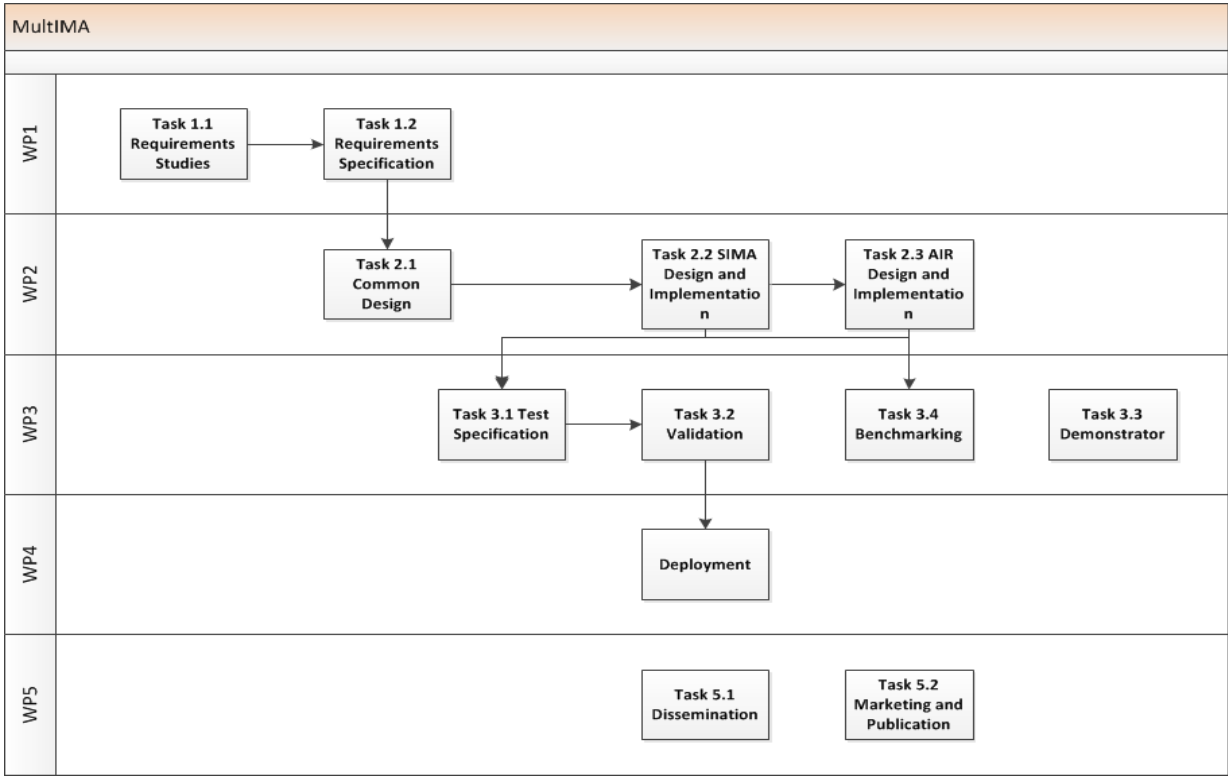
MultIMA

Multi-core in Integrated Modular Avionics

INNOVATING SOLUTIONS

STUDY LOGIC

- GTSP-AO activity / TO – Martin Hiller
- KO - December 2012 / AR – May 2014

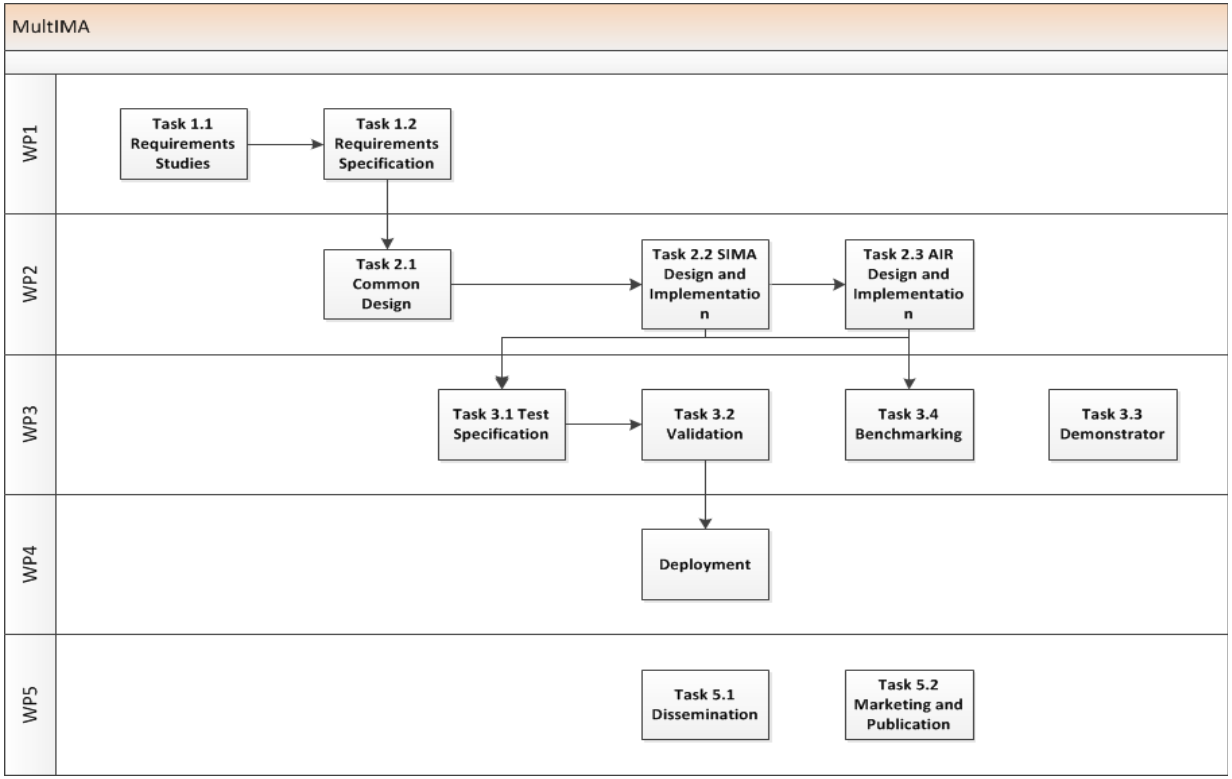


PROJECT GOALS

- To establish the feasibility of running an IMA system in parallel on different cores;
- To create multi-core versions of the AIR operating system and of the SIMA simulator;
- To determine and demonstrate the best mapping between the IMA-SP architecture and multiple cores, with the aim to address architectural problems of IMA-SP and multi-core.

STUDY LOGIC

- GTSP-AO activity / TO – Martin Hiller
- KO - December 2012 / AR – May 2014



OUTLINE

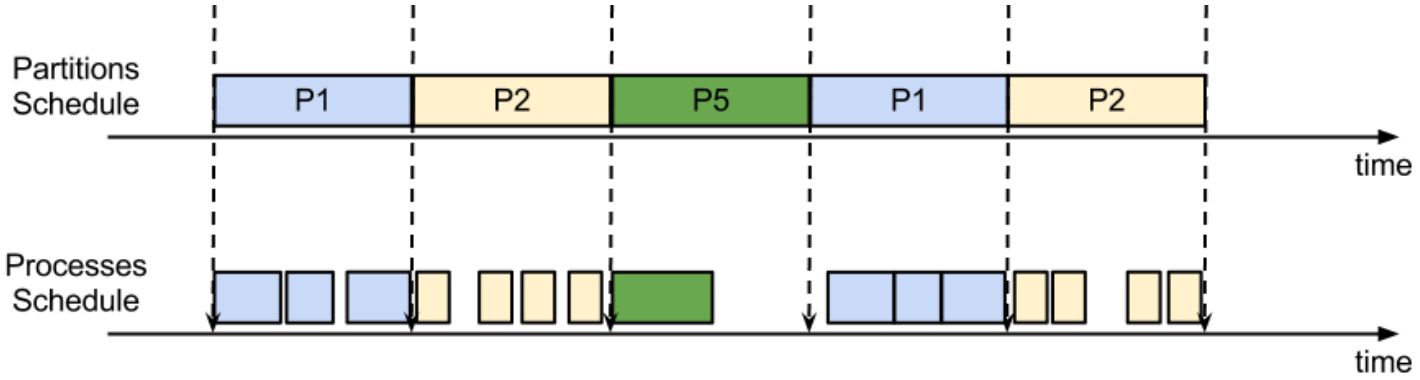
- MultIMA Activity
- IMA, SIMA & AIR
- Multi-Core
  - Motivation and Issues
- Design and Implementation
  - Common
  - SIMA
  - AIR
- Tests, Benchmarks and Demonstrator
- Conclusion

IMA, SIMA & AIR

INNOVATING SOLUTIONS

INTEGRATED MODULAR AVIONICS

- Enables multiple unrelated applications, with different criticalities, to share the same computing platform without interference;
- Based on Robust Partitioning – Time, Space and I/O segregation:
  - TDM partition schedule
  - Memory areas with strict access rights

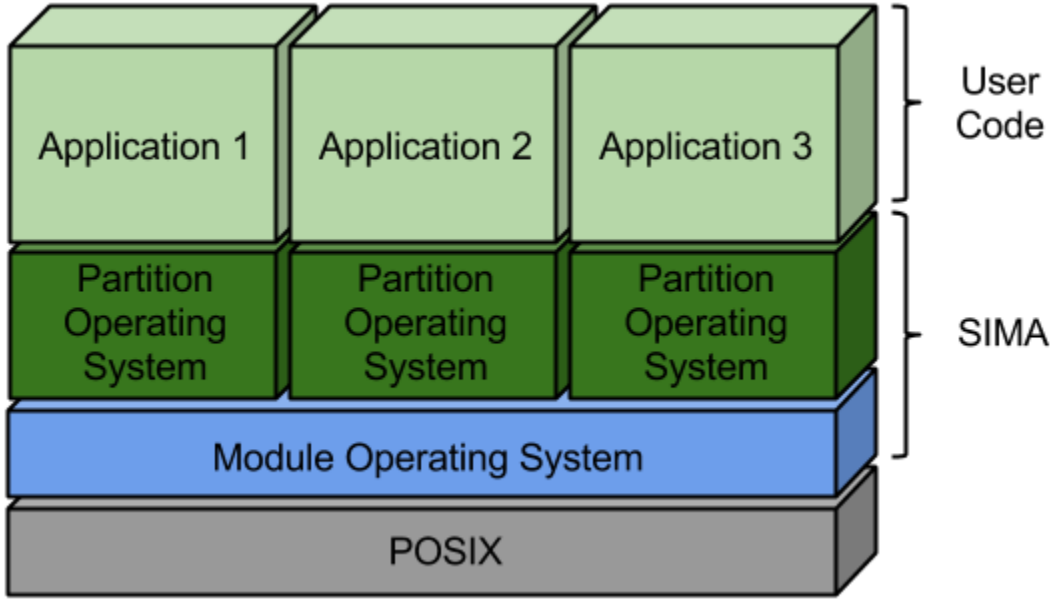


SIMA

- Simulation of Integrated Modular Avionics;
- Origin: ESA AMOBA project – ARINC 653 Simulator for Modular Space Based Applications;
- Functional simulation of ARINC 653 – Application Executive (APEX);
- Assessment of ARINC 653 transposition to the Space domain;
- Support application development and training;

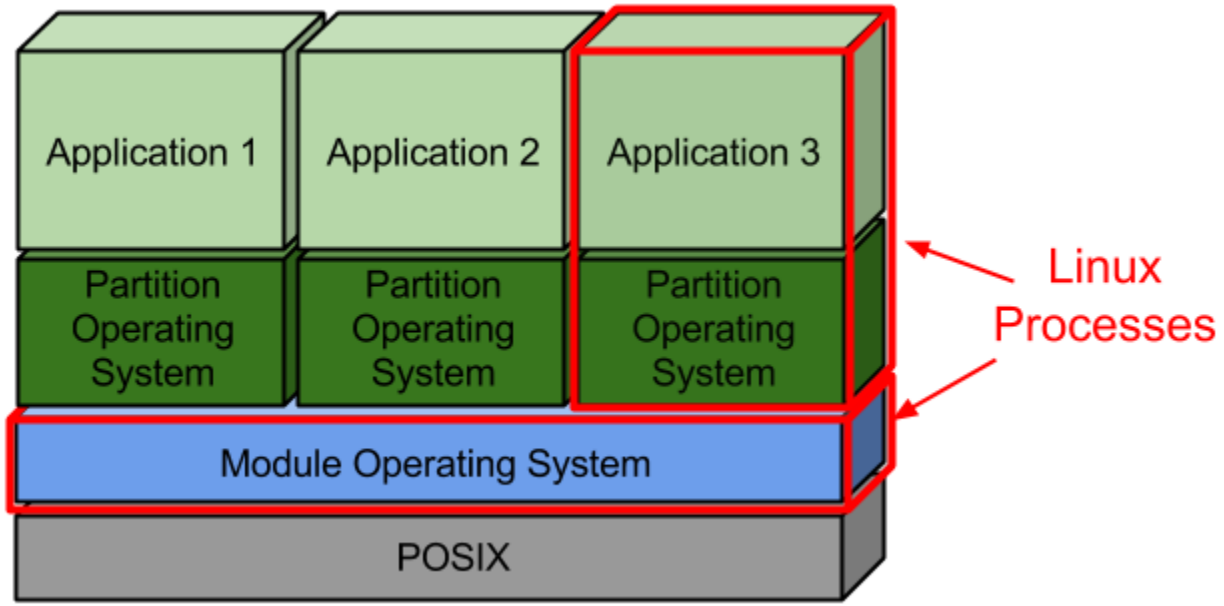
SIMA

- Designed to address portability across operating systems supporting POSIX;
- Shield applications from POSIX implementation specificities;
- Share configuration format with ARINC 653 operating systems;
- Isolate specific configuration into another file.



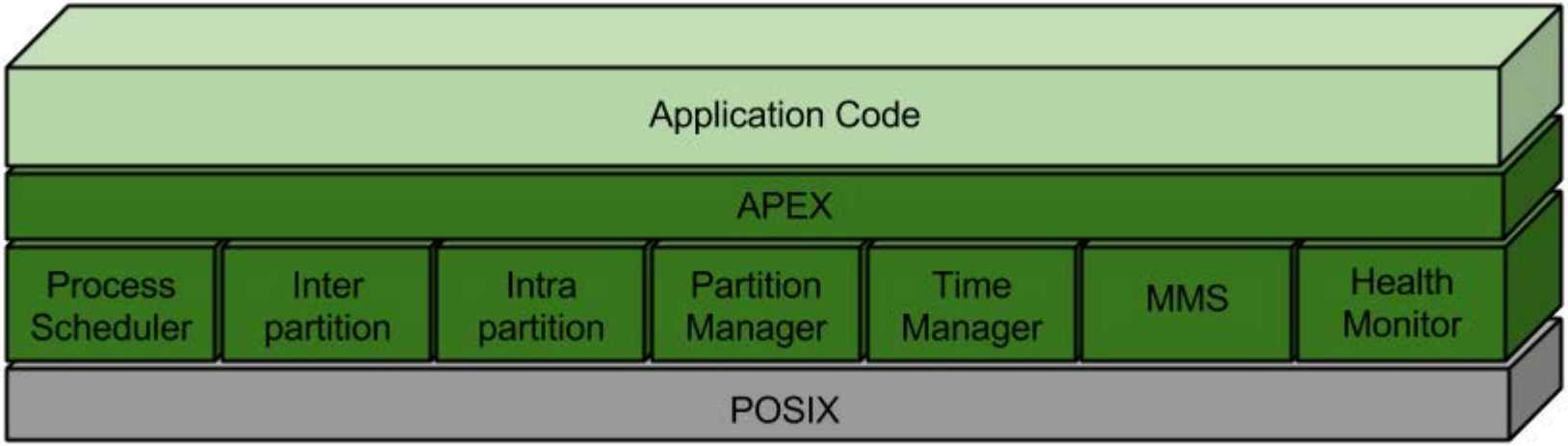
SIMA

- Each SIMA partition is comprised by a process;
- The SIMA Module Operating System is another process;
- The partition scheduling is accomplished through signals exchanged between those two processes;



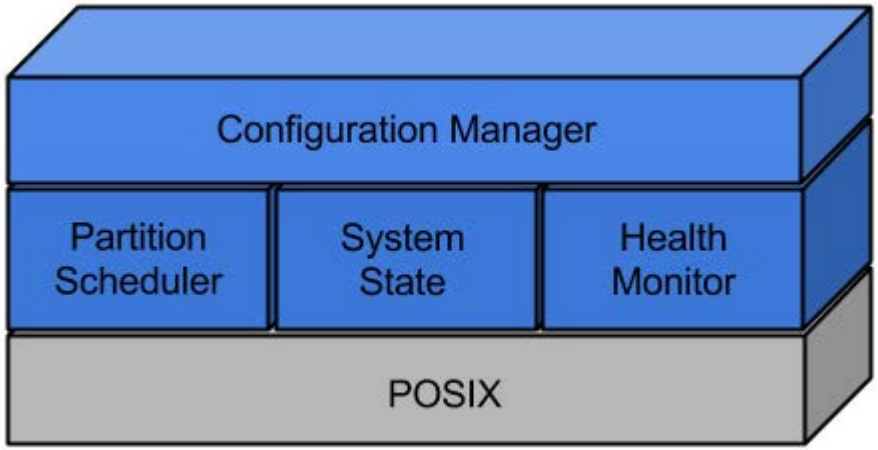
SIMA PARTITION

- Different software modules implementing APEX services;
- Process scheduler supports processes queues visibility for users;
- A SIMA partition can be executed in standalone mode;
  - With inter partition communication;
  - No MMS;
  - No HM;



SIMA MOS

- Manages system state and module configuration data;
- Controls the module schedule (SET_MODULE_SCHEDULE);
- Controls partitions execution;
- Host the Health Monitor;



AIR

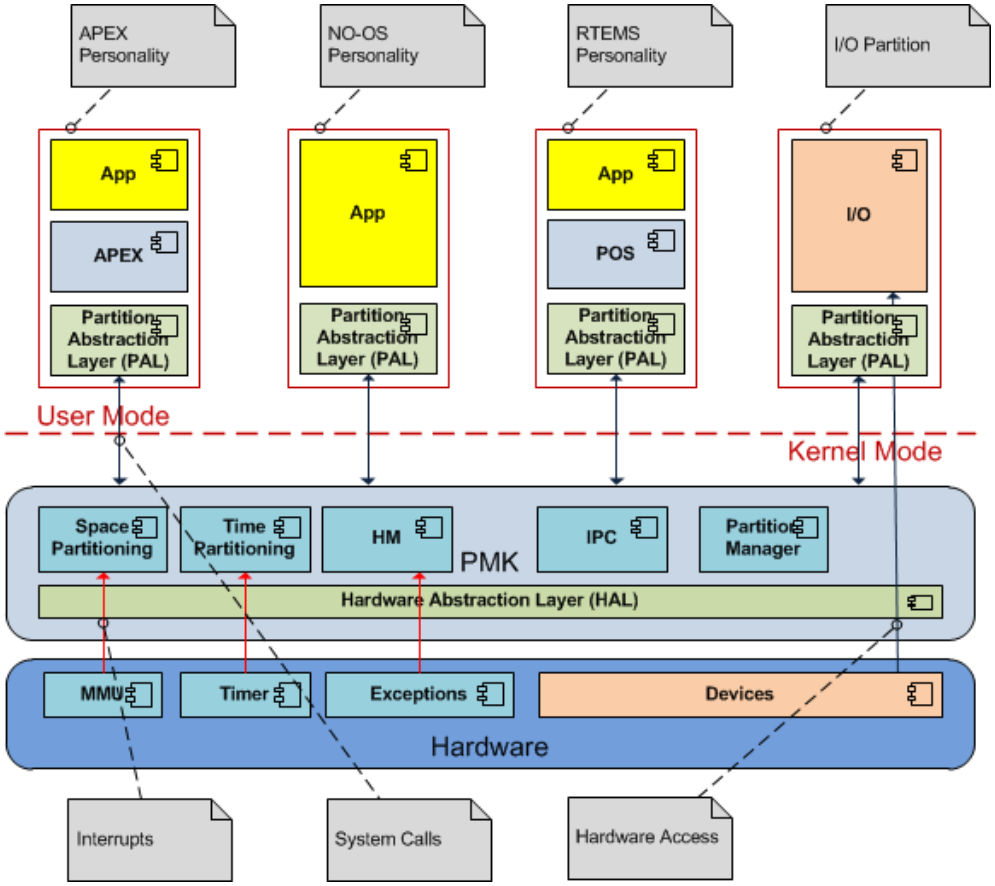
air

REAL-TIME HYPERVISOR COMPLIANT WITH ARINC 653 and IMA-SP

- AIR is a full fledged real-time operating system that guarantees time and space partitioning
- AIR hardware abstraction layer is rooted in RTEMS
- Currently TRL 5
- AIR results from several research and development efforts made in cooperation with ESA
- Executes in SPARC/Leon 2/3/4
- AIR is open source (GPL v2 with linking exception) – RTEMS license

AIR ARCHITECTURE OVERVIEW

- Modular Design:
  - PMK
  - HAL
  - PAL
- Partition Management Kernel (PMK) is designed using a microvisor architecture;
- PAL is a generic abstraction layer to which the Partition operating system attaches to;
- "System" Partitions:
  - Higher Privileges



AIR ARCHITECTURE OVERVIEW II

- Space partitioning
  - Uses MMU;
  - Reduces the number of MMU pages used for each partition (ideally only one) to ease WCET analysis;
  - Uses virtual addresses to guarantee partition independence;

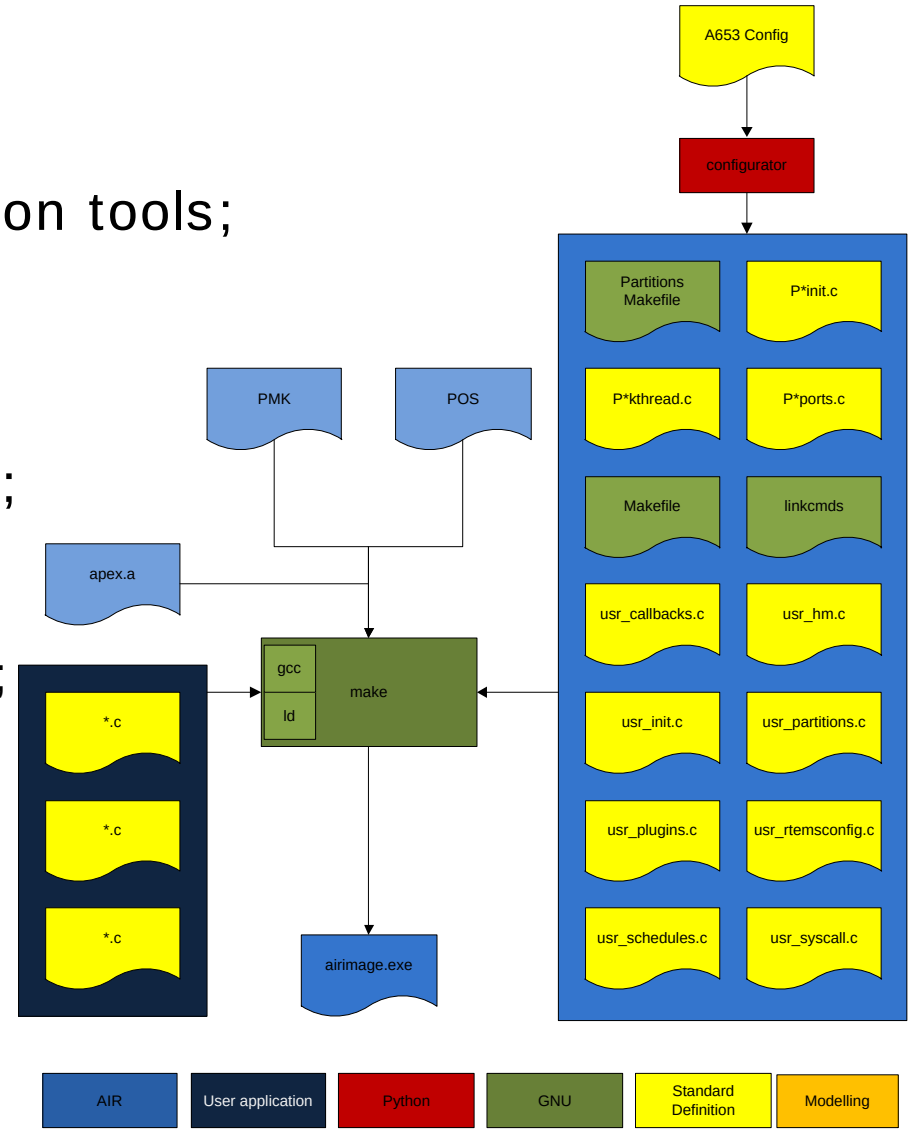
- Time Partitioning
  - Based on a monotonic timer;
  - Partitions can set and use other timers through a specific API;
  - “Big” system calls are preemptable to reduce the effects of the partitioning kernel presence on the partitions;

AIR FEATURES

- Supported “personalities”:
  - ARINC 653 APEX (P1, P2)
  - IMA-SP
  - RTEMS (-impr, -4.x)
  - Bare C Executive
- AIR provides ARINC 653 compatible:
  - Inter-partition communication: queuing and sampling ports;
  - Health Monitoring (HM);
  - Multiple Module Schedules (MMS);

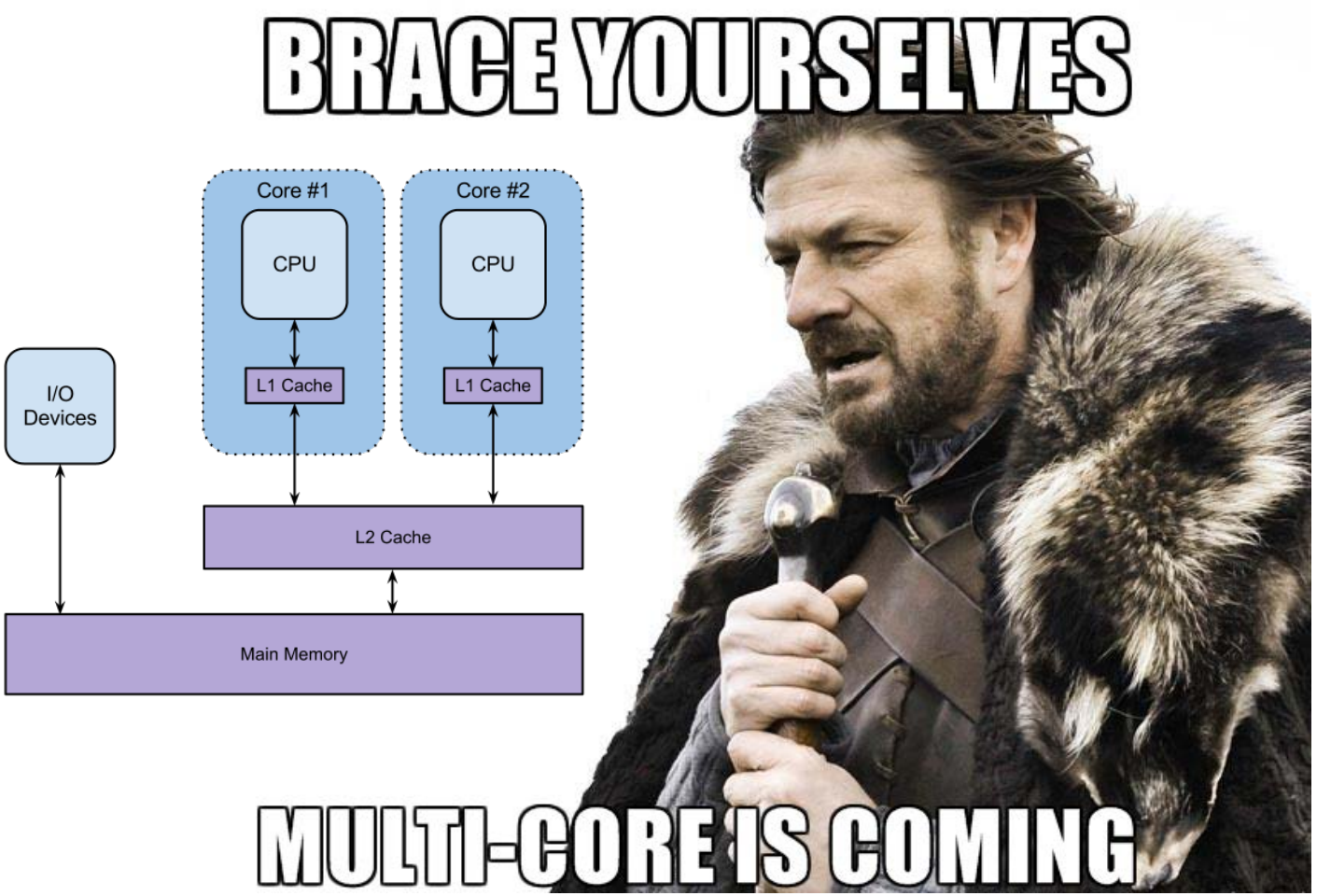
AIR TOOLCHAIN

- Kernel is small and simple;
- The intelligence is in configuration tools;
- Tool chain is hence complex;
- Starting point is an ARINC 653 XML specifically tailored for AIR;
- It defines:
  - Partition attributes (e.g. FPU use);
  - Scheduling;
  - I/O device allocation



MULTI-CORE

MULTI-CORE AND REAL-TIME



MOTIVATION FOR MULTI-CORE

- Address the Memory/ILP/Power wall hit by single-core processors;
- Enables the integration of onboard systems that were, up to now, split in multiple computational units;
- Enables development of more autonomous and sophisticated missions by exploring the increased processing power provided by multi-core. E.g.
  - Multi-core for payload processing: use case from the Euclid mission (ADCSS 2011)
  - GNC application cases needing multi-core processors (ADCSS 2011)

ISSUES WITH MULTI-CORE

- Most software components are inherently sequential and, thus, not prone to parallelization;
- Parallelization of software artefacts creates complexities at software design level (requirements concerning synchronization, deadlock and starvation avoidance, etc.) that are difficult to address;
- Shared resources open channels of interference that can have a huge impact on the software behavior and can break the independence between different software modules (key point in the IMA concept).

IMA AND MULTI-CORE SYNERGIES

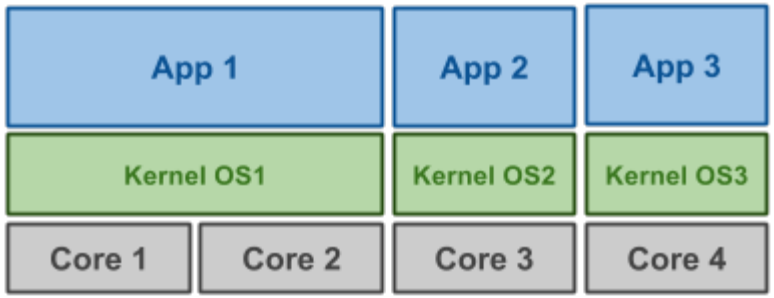
- IMA can potentiate the greater level of integration promoted by multi-core architectures:
  - Without partitioning, integrating a high number of components can become prohibitively complex;
- TSP can help manage the additional complexity brought by multi-core operating systems;
- Ease the transposition of application from single-core to multi-core platforms;

DESIGN

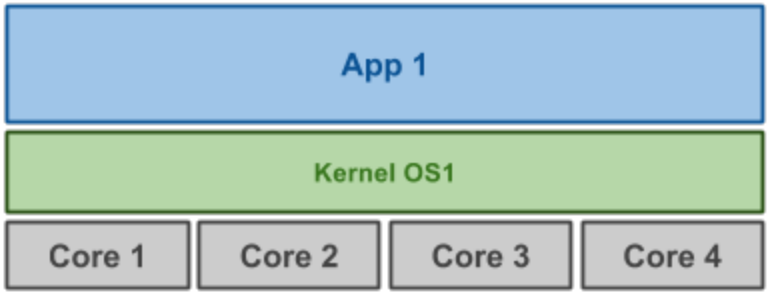
INNOVATING SOLUTIONS

SOFTWARE MAPPING

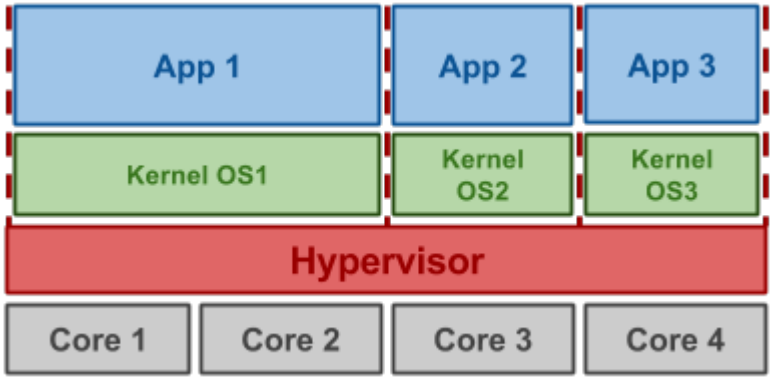
- AMP



- SMP

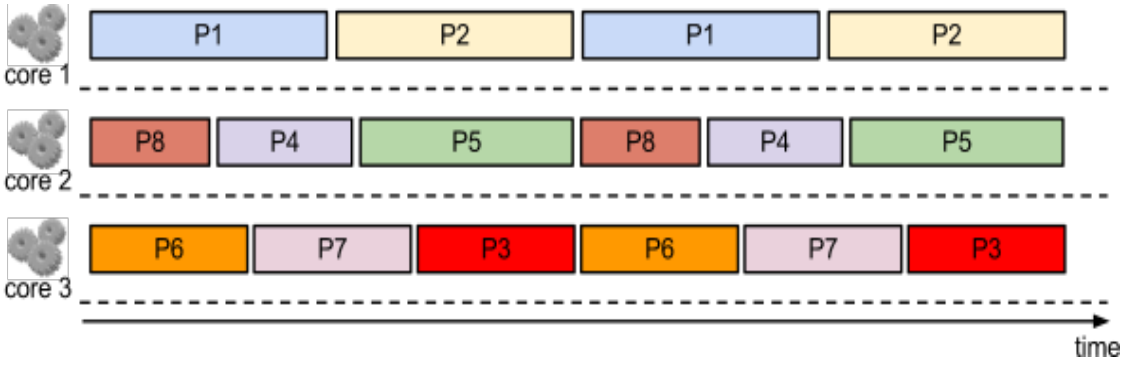


- Supervised AMP



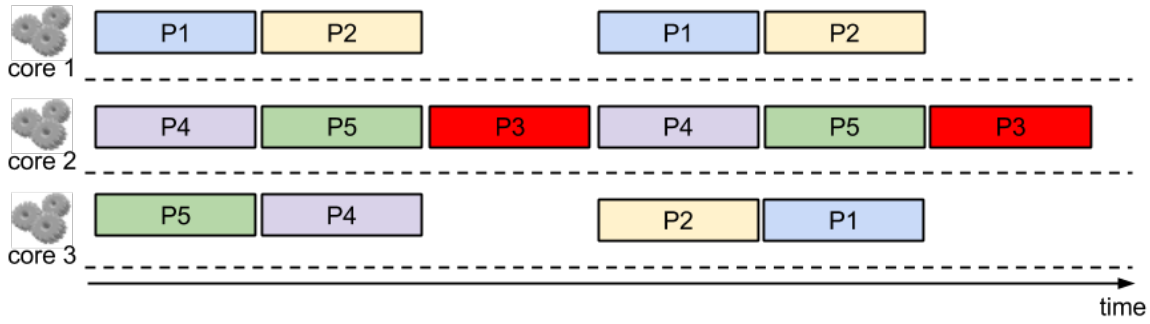
IMA SCHEDULING

- The most direct way to handle scheduling is to extend ARINC 653 partition scheduling to multiple cores;
- Different partitions run concurrently in different cores;
- Partitions don’t see a multi-core platform:
  - Partition operating system doesn’t need to be multi-core aware;
  - Applications don’t require parallelization;



IMA SCHEDULING - II

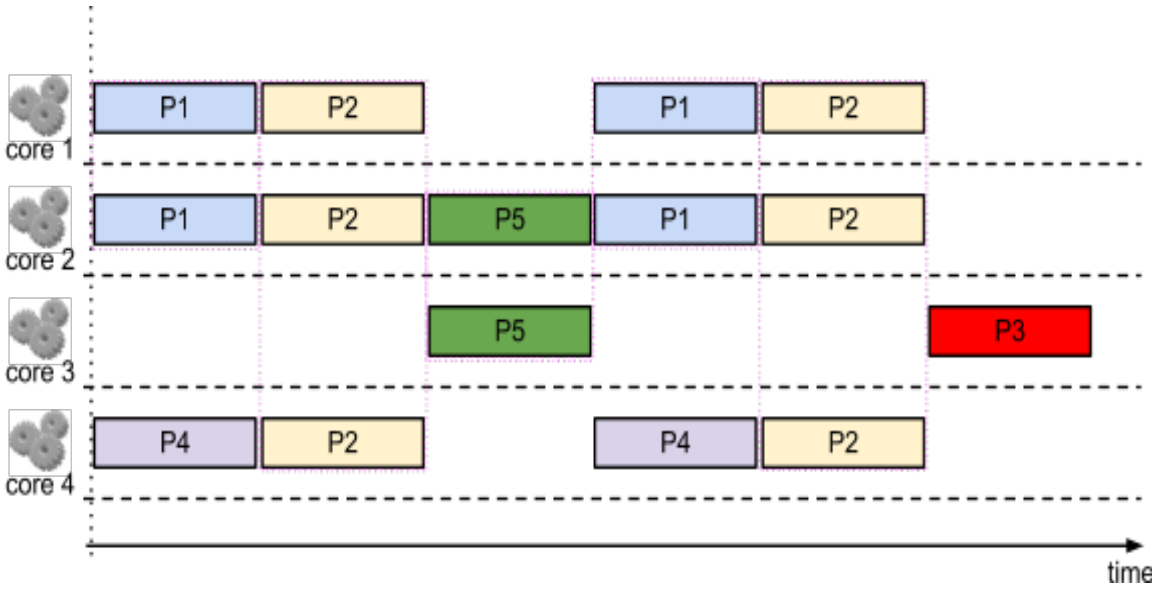
- Concurrently executing partitions create interference, therefore breaking independence;
- Need to manage interference for Hard Real-time partitions;
- Solution: run HRT partitions isolated;



- Disadvantage: Decreases CPU utilization;

IMA SCHEDULING - III

- Some applications are parallelizable and therefore require access to more than one-core:



MULTICORE CONFIGURATION

```
<Partition_Schedule PartitionIdentifier="10" PartitionName="p0"
    PeriodSeconds="2.000" PeriodDurationSeconds="2.000">
    <Window_Schedule WindowIdentifier="101" WindowStartSeconds="0.0"
    WindowDurationSeconds="0.500" PartitionPeriodStart="true" />
    <Window_Schedule WindowIdentifier="102" WindowStartSeconds="0.5"
    WindowDurationSeconds="0.500" PartitionPeriodStart="false" />
    <Window_Schedule WindowIdentifier="103" WindowStartSeconds="1.0"
    WindowDurationSeconds="0.500" PartitionPeriodStart="false" />
    <WinExt xmlns="Window_Sched_Ext" WindowIdentifier="101" Cores="0" />
    <WinExt xmlns="Window_Sched_Ext" WindowIdentifier="102" Cores="1" />
    <WinExt xmlns="Window_Sched_Ext" WindowIdentifier="103" Cores="2;3" />
</Partition_Schedule>
```

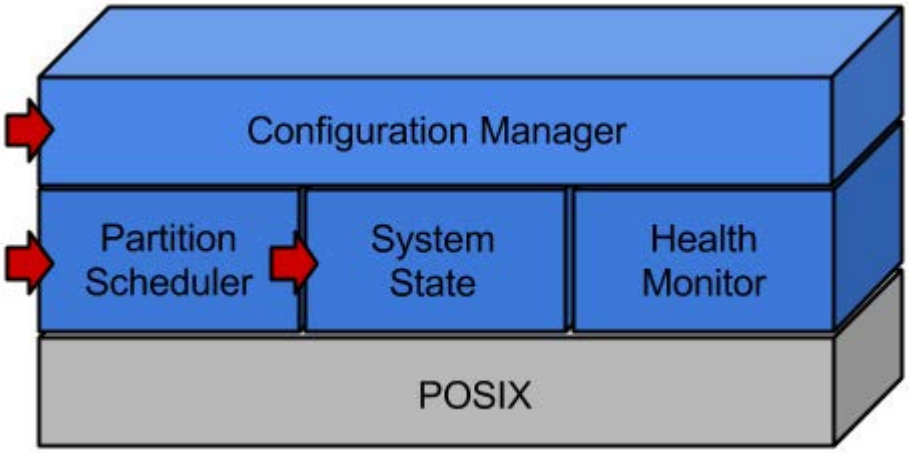
XML Multi-Core Configuration

SIMA MULTI-CORE

INNOVATING SOLUTIONS

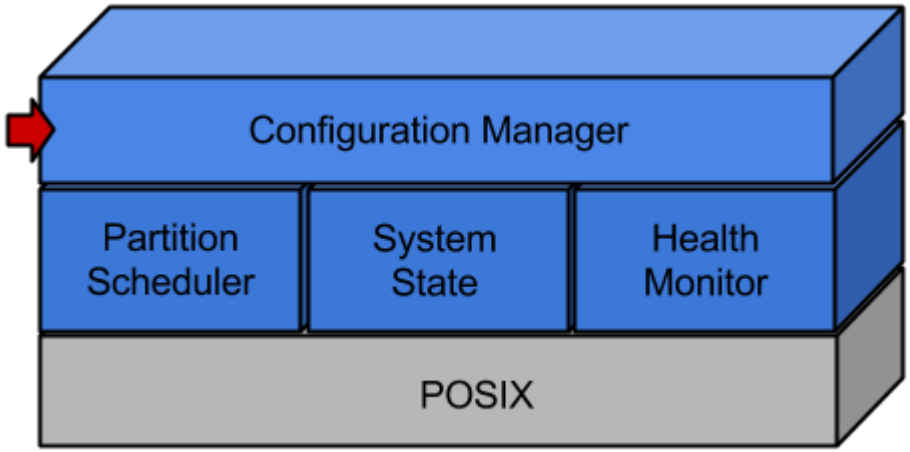
MULTICORE SIMA

- MOS/SEP Executive hosts a set of four core schedulers within the partition scheduler module;
- MOS uses Linux timers for controlling partitions schedule;
- Core schedulers are synchronized by barriers;
- Configuration manager and system state modules were also modified to acknowledge multicore aspects;



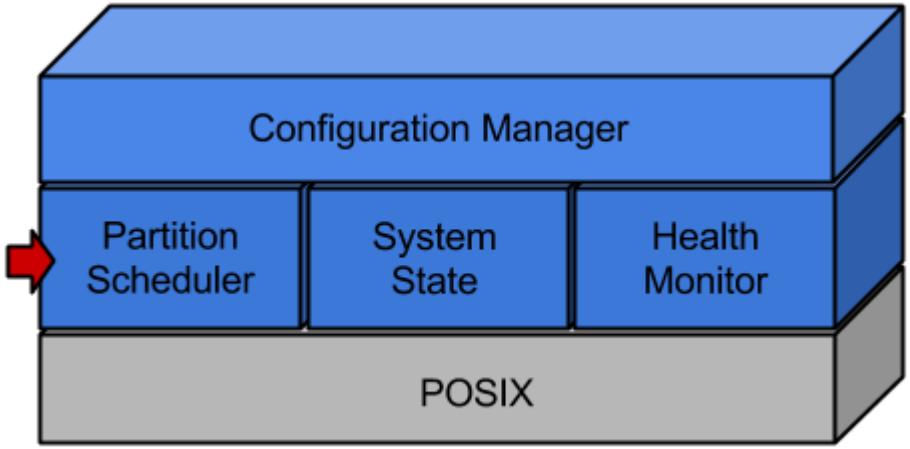
MULTICORE SIMA

- Configuration Manager:
  - Addresses cores schedulers and the mapping of cores to the host available cores;
  - Map the partition schedules to each core;
  - Verifies the partitions permissions for changing the module schedule and setting partitions operating modes;



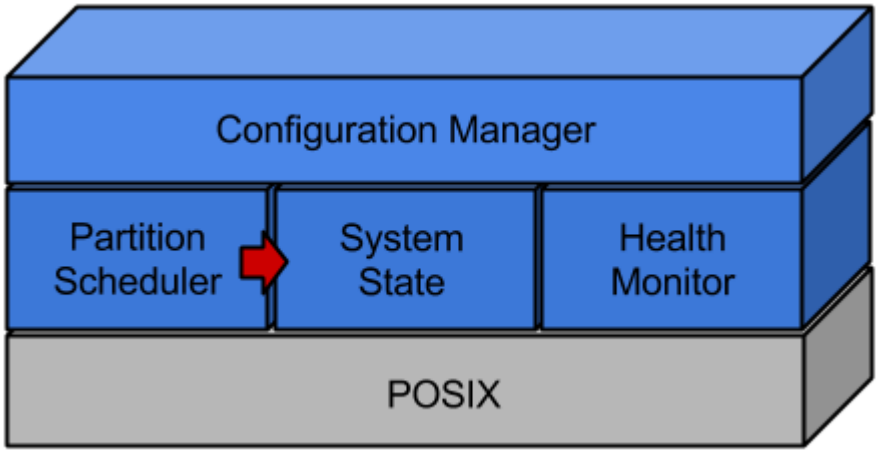
MULTICORE SIMA

- Partition Scheduler:
  - Starts and stops core schedulers;
  - Initializes, destroys and reinitializes the barriers synchronizing partitions schedulers (according to the module schedules);
  - Consolidate the partial system state based on its own set of partitions;



MULTICORE SIMA

- System State:
  - Consolidate the system state based on core schedulers states;
  - Determine the Health Monitor response action;

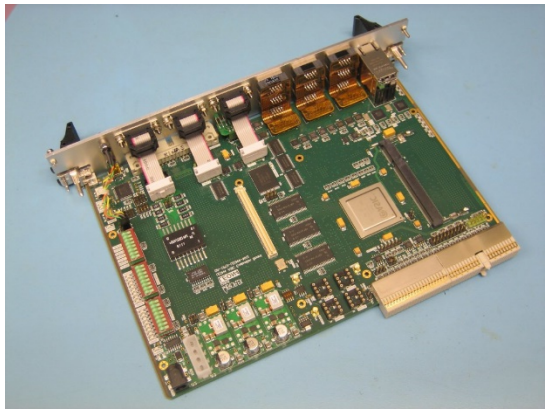


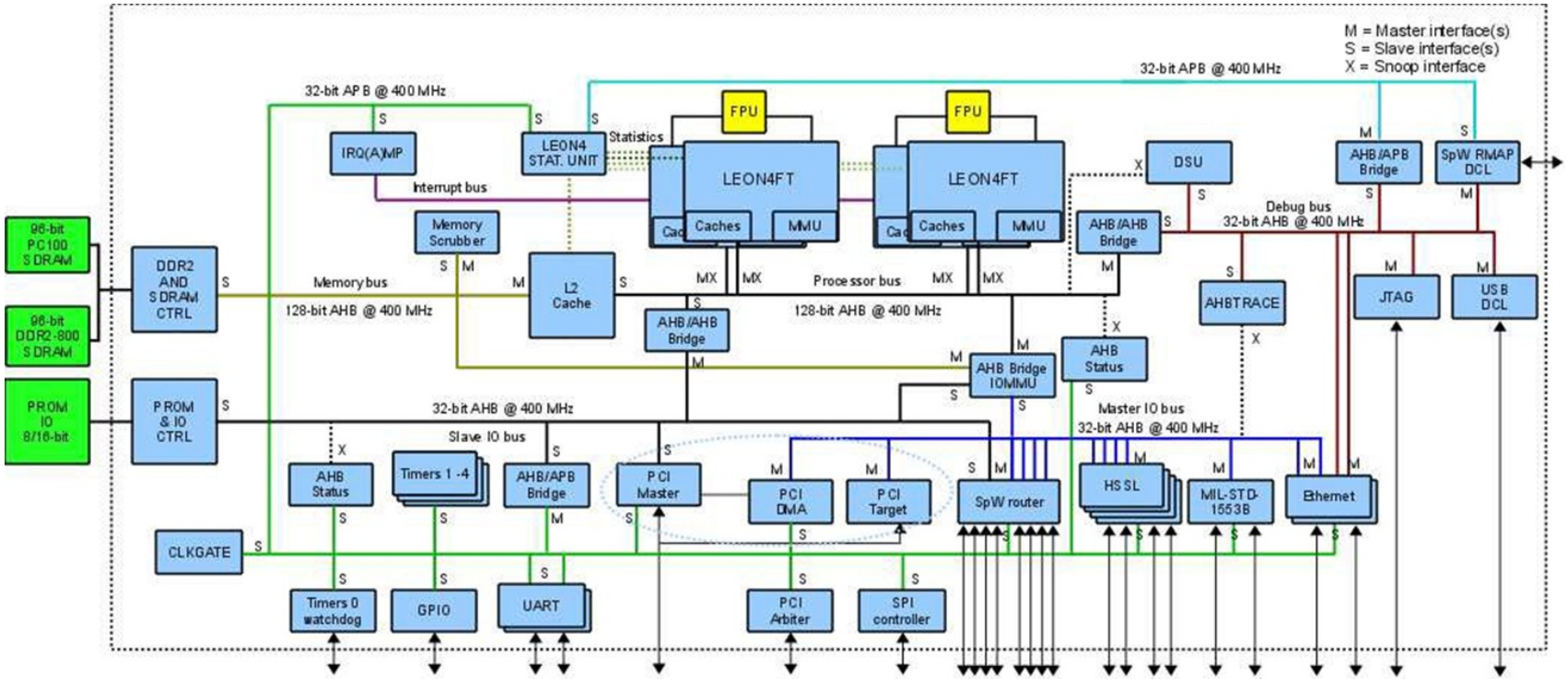
AIR MULTI-CORE

INNOVATING SOLUTIONS

AIR TARGET

- NGMP – Next Generation Microprocessor
- Quad-Core LEON 4





FIRST STEP: MAKE IT RUN!

- NGMP has a different hierarchical AMBA structure;
  - Our AMBA scanning functions required adaptation to support it.
- NGMP contains several “new” peripherals, that weren’t completely supported by AIR:
  - L2Cache;
  - IOMMU;
  - IRQASMP.

L2 CACHE

- 256 Kb, 4 Ways;
- Several possible configurations and replacement policies:
  - LRU;
  - LRR;
  - Partitioned;
  - Write Through;
  - Copy Back;
- New hardware abstraction interface for AIR;
- Default Configuration for AIR:
  - Partitioned with write through write policy.

IOMMU

- IOMMU stands between DMA capable I/O devices and main memory; It mediates DMA transfers from I/O devices.
- Several possible configurations:
  - With memory translation and memory protection;
  - Only with memory protection;
  - Variable granularity (page size)
- Default configuration for AIR:
  - Access protection vector (memory protection only) with configurable page size;
- Devices are allocated as part of the configuration to partitions;
  - Allocated devices are able to access the partition memory space;
  - Partition can refine the accessible memory area.

INITIALIZATION

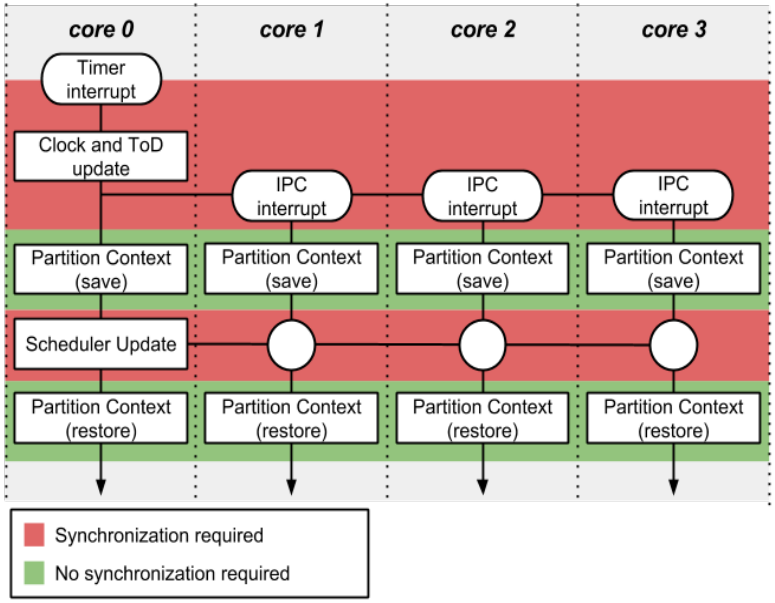
- Initialization is mainly sequential;
  - A single core (core 0) is used to handle the bulk of the initialization;
- New peripherals are initialized:
  - L2 Cache;
  - IOMMU;
- If included in the schedule, secondary cores are initialized:
  - The initialization routine prepares the CPU registers and sets up an initial MMU context;
  - The secondary cores signal the end of their initialization and enter the idle mode;
  - Initialization routine in core 0 waits for the other cores;
  - Cores are awaken when the MTF starts through a Inter-processor interrupt.

NEW INTERNAL PRIMITIVES

- Classical Spin lock
- Mellor-Crummey and Scott (MCS) spin lock
- Phase-Fair Read-Write lock
  - Used on the ARINC 653 ports
- Synchronization Barriers
  - Sense-reversing centralized barriers

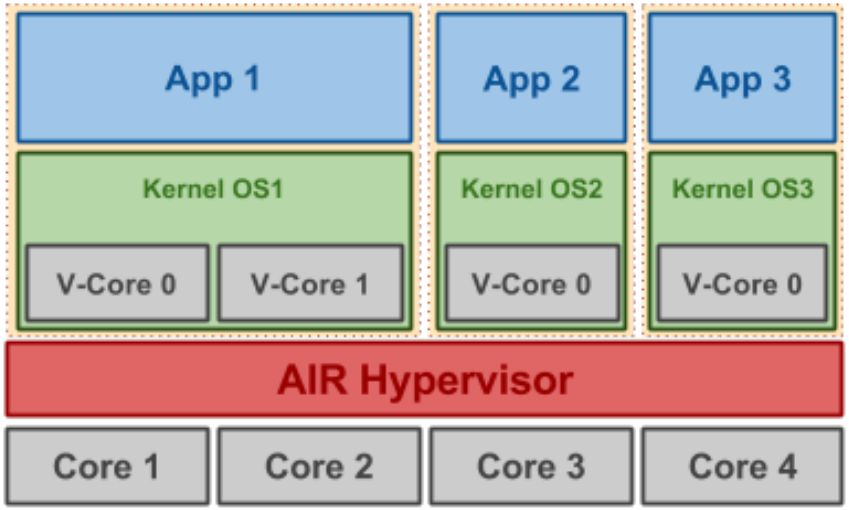
SCHEDULING

- AIR's partition scheduler was extended to support multiple cores;
- A timer interrupt is used to drive the partition scheduler;
- When a context switch is necessary in another core, the core that received the timer interrupt sends a inter-processor interrupt;
- Several synchronization points are required:
  - Partition migrating core;
  - MTF end;
  - MMS change;
  - End of SMP interrupt;



MULTI-CORE PARTITIONS: VIRTUAL CPU

- Virtual CPU concept extended to support several vCPUs per partition;
- Each vCPU contains the status of virtualized registers that are different for each core and partition;
- vCPU are accessed through system calls;
- Their values are filtered and written to the real hardware registers



MULTI-CORE PARTITIONS: INTERFACE

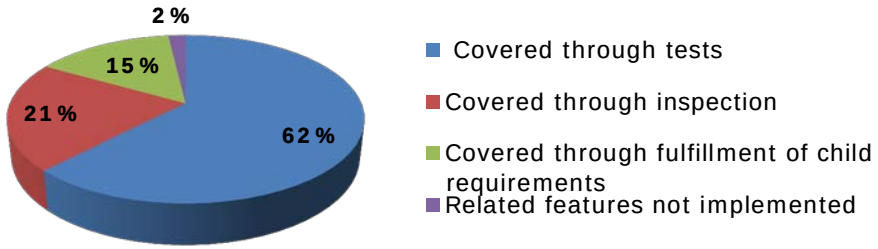
- A new set of system calls was defined that enables the same operations over virtual CPUs that could be done over hardware cores in SPARC:
- A partition may:
  - Boot a secondary virtual CPU
  - Send an Inter processor interrupt to a virtual CPU
  - Broadcast an Inter-processor interrupt to all virtual CPUs

TESTS AND BENCHMARKS

gmv®
INNOVATING SOLUTIONS

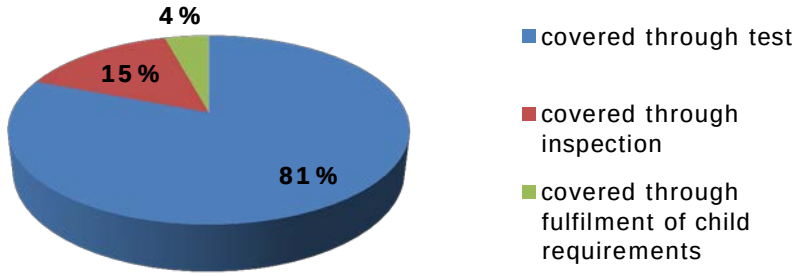
AIR AND SIMA TEST RESULTS

AIR



Execution Statistics	
Tests Ran:	133/139
Tests Passed:	133/133
Success Rate:	100%
Elapsed time:	~24 mins

SIMA

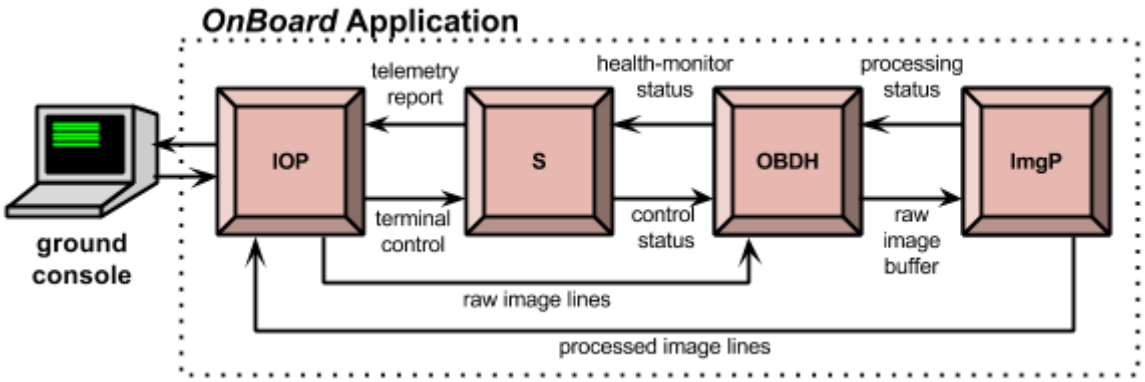


Execution Statistics	
Tests Ran:	53/53
Tests Passed:	53/53
Success Rate:	100%
Elapsed time:	~8 mins

DEMONSTRATOR

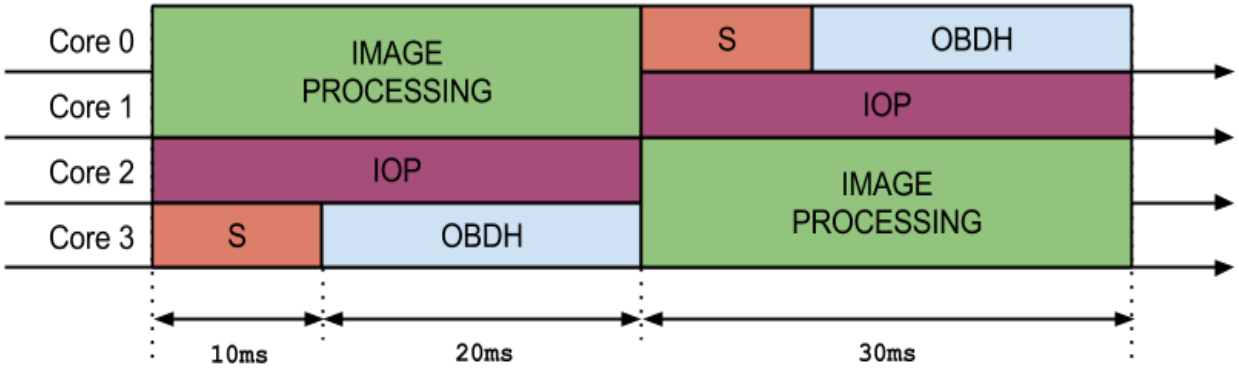
MULTICORE DEMONSTRATOR

- Demonstrator is based on an image processing application
- Independent set of partitions
- Two scheduling configurations:
  - AMP Scheduling
  - SMP Scheduling
- I/O communication with a ground console

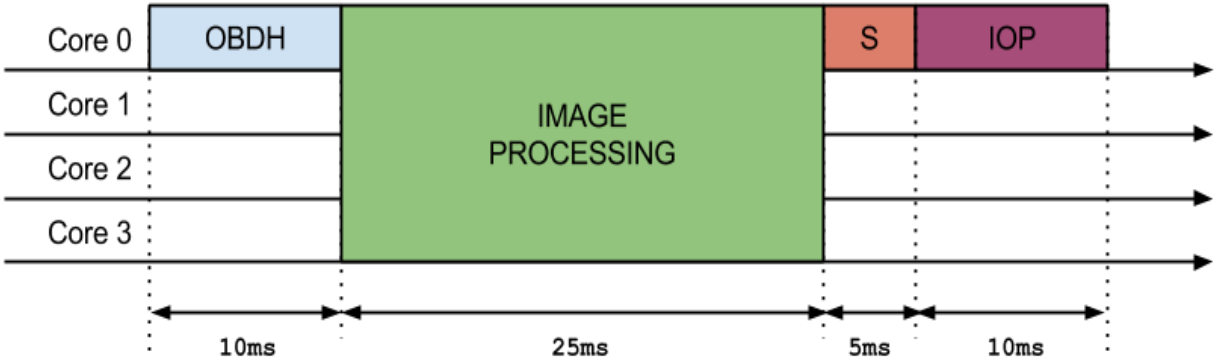


DEMONSTRATOR SCHEDULE CONFIGURATIONS

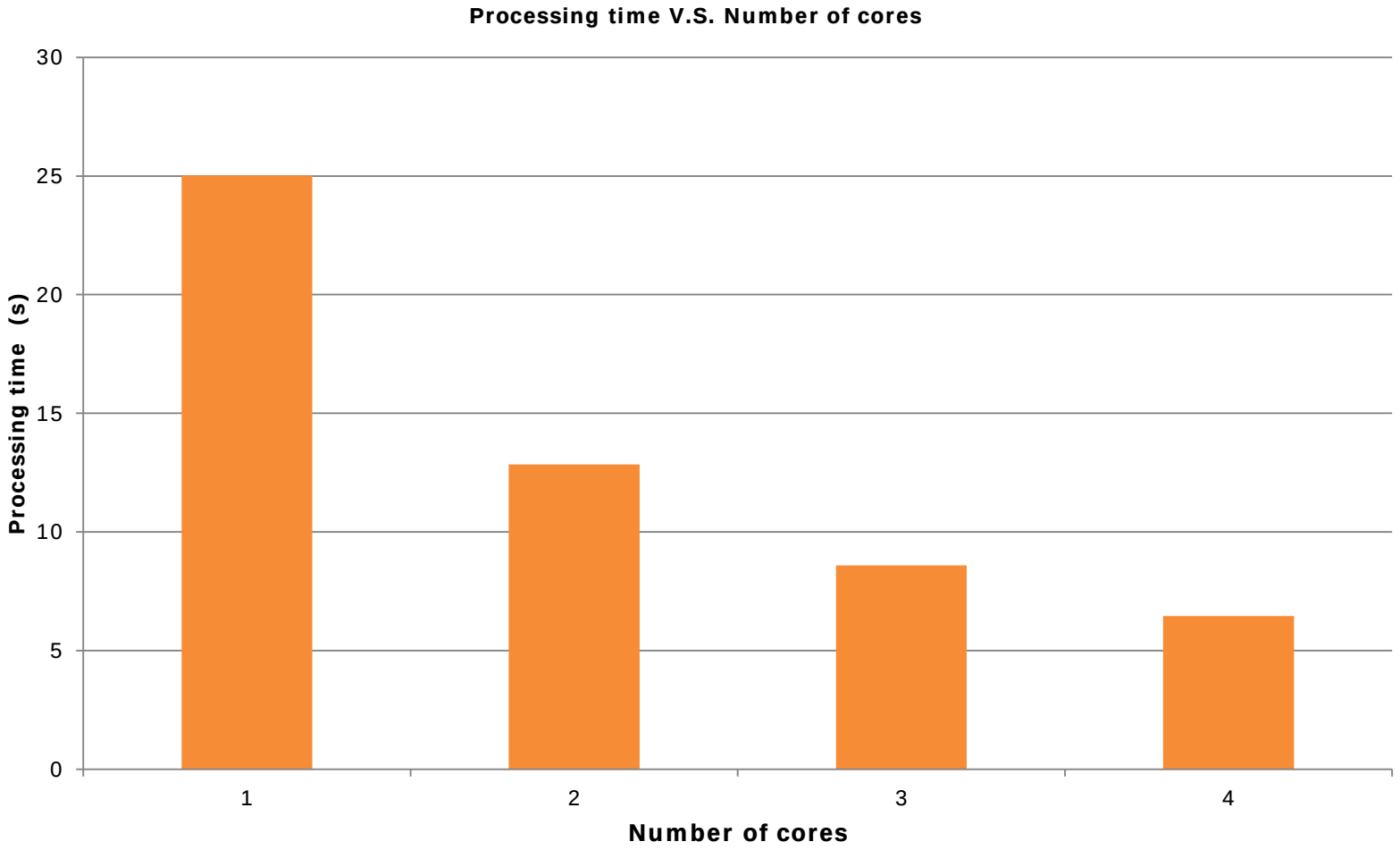
AMP Schedule



SMP Schedule



DEMONSTRATOR RESULTS



BENCHMARKS

COREMARK

- Bare RTEMS Result

Execution Time	Iterations per second	CoreMark
12.07	166	1.105

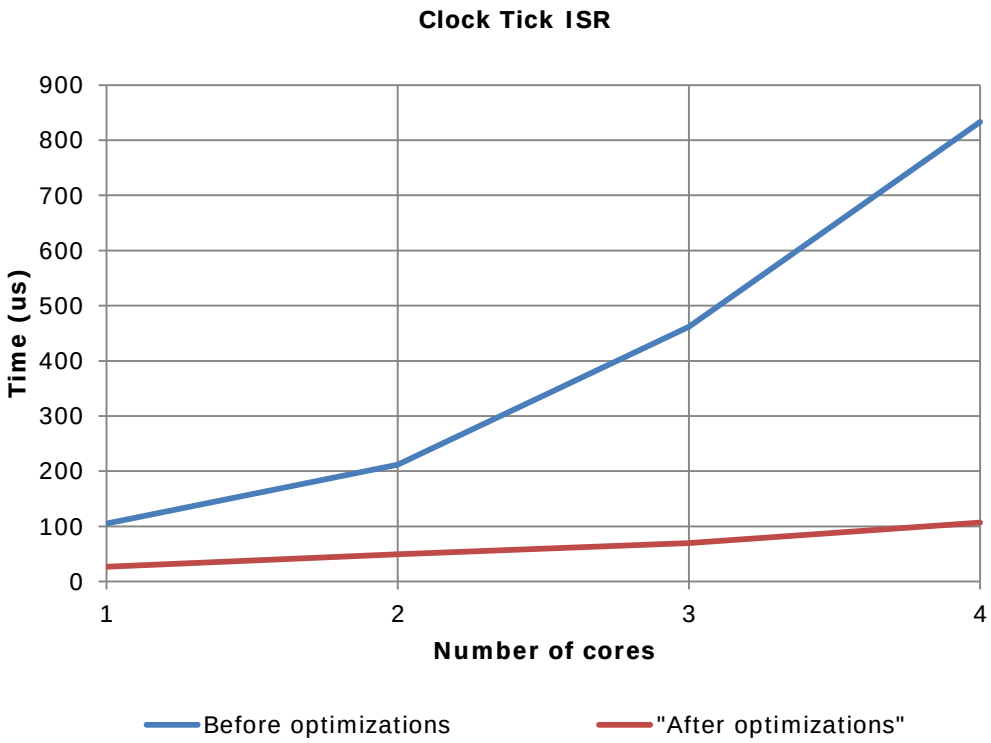
- AIR Multi-core for variable number of cores:

Number Cores	Execution Time	Iterations per second	CoreMark	Performance Gain
1	12.280	163	1.086	-1.71%
2	10.0100	200	1.332	20.58%
3	9.9000	202	1.345	21.80%

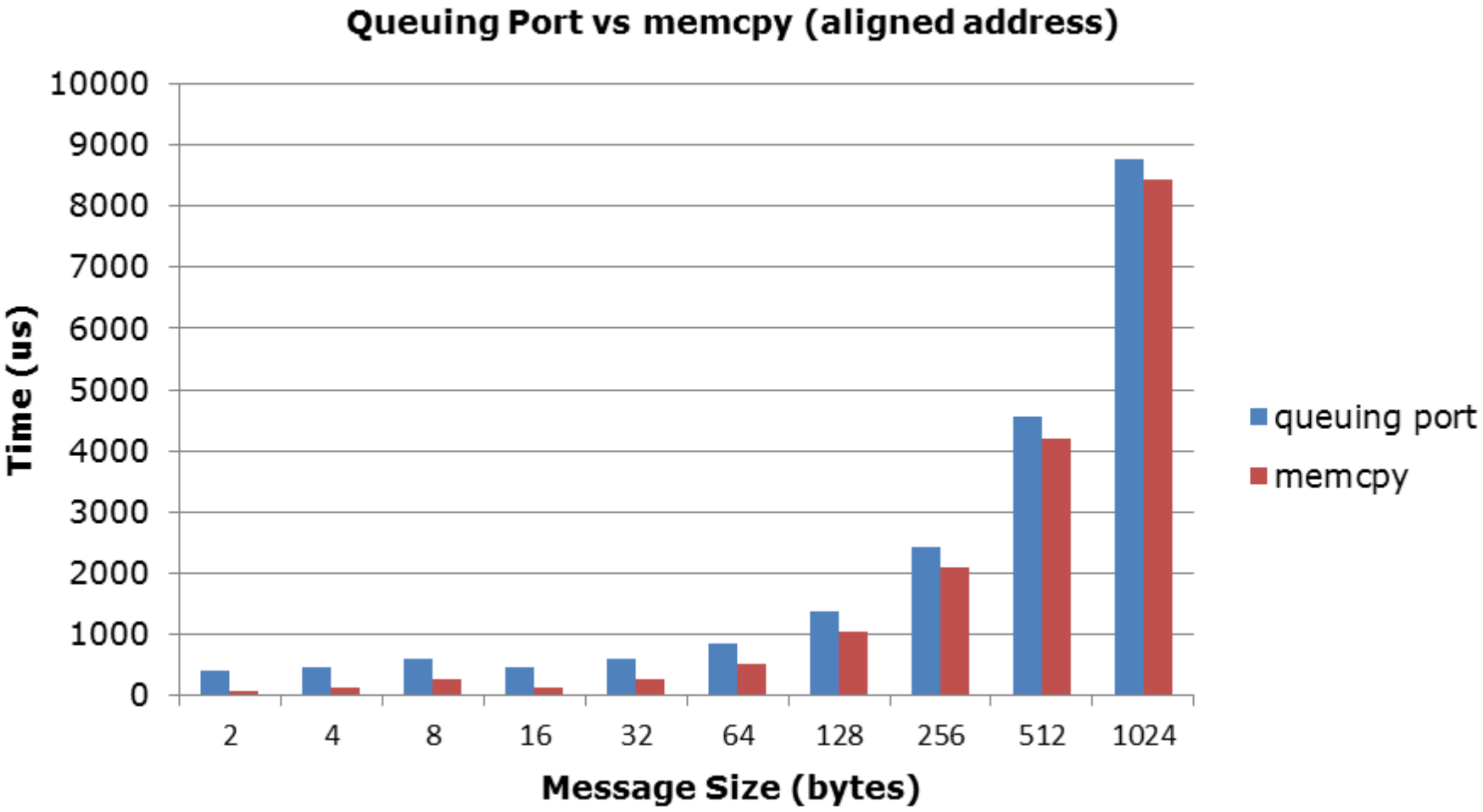
- With variable window size:

Window Slot	Execution Time	Iterations per second	CoreMark	Performance Loss
30 s	12.220	164	1.091	1.23%
1 s	12.240	163	1.089	1.38%
0.5 s	12.240	163	1.088	1.39%
0.1 s	12.280	163	1.086	1.71%
0.01 s	12.450	161	1.074	2.77%

CLOCK TICK OPTIMIZATION



BENCHMARKS – PORTS



CONCLUSION

INNOVATING SOLUTIONS

CONCLUSION

- We were able to successfully develop and demonstrate a multi-core operating system and simulator;
- IMA and Multi-core are two technologies that should be exploited in tandem due to their positive synergies;
- It is possible to mitigate some multi-core issues with the choice of system configuration;
- Inter-core interference exists; it breaks application independence;
- Interference must be:
  - Addressed immediately since the processor development stages (e.g. T-CREST and MERASA projects);
  - Considered and characterized through new WCET estimation methodologies (e.g. probabilistic WCET - Proartis);

CONCLUSION

- Developing software for multi-core is more complex than for single-core:
  - Simply put: bugs become worse, software becomes less intuitive
  - There is a more pressing need for characterizing software;
  - Tooling support is key;
- Complexity should be managed with a raise in the abstraction level:
  - Use high level programing languages or libraries that enforce concurrency patterns (e.g. Go-lang, MPI, etc);
- ESA should monitor and govern external research efforts from domains with similar predictability requirements;

Thank You

Cláudio Silva
claudio.silva@gmv.com

Av. D. João II, Lote 1.17.02
Torre Fernão Magalhães, 7º
1998-025 Lisboa
Portugal

Tel. +351 21 382 93 66
Fax +351 21 386 64 93

www.gmv.com

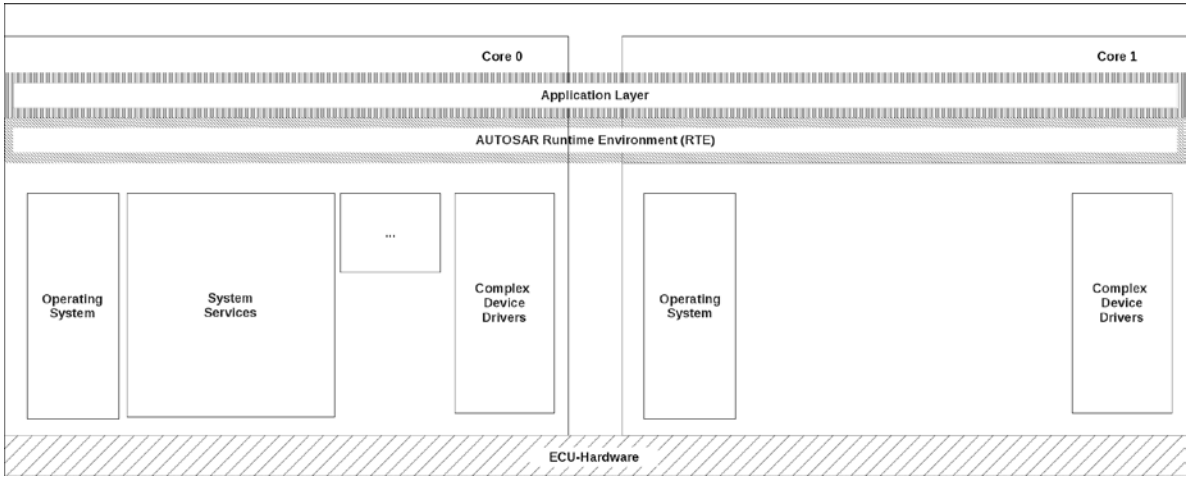
gmv®
INNOVATING SOLUTIONS

Multi-core

ADOPTION IN OTHER DOMAINS

AUTOSAR MULTI-CORE

- SMP configurations only with tasks scheduled statically to a core;
- One main core is defined – system support services run on that core;
- Inter-core communication through shared memory;
- Mutual exclusion through spin locks with the multiprocessor priority ceiling protocol;



AERONAUTICS

- ARINC 653 will include a multi-core supplement in part 5:
  - Very political discussion underway…
  - Several competing implementations from different OS providers (VxWorks, DEOS, Integrity);
- EASA is suggesting some guidelines regarding multi-core processors:
  - Currently only considering dual-core processors;
  - Integration of software from different systems not permitted, therefore ensuring that erroneous behavior of any of the hosted applications is contained within that one system.

Multi-core

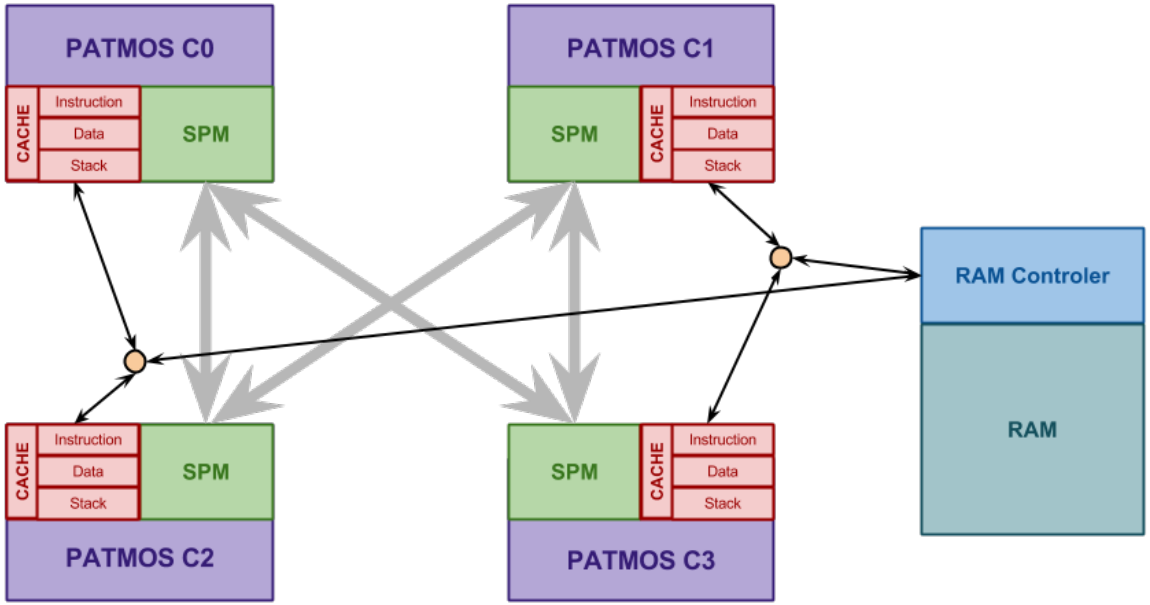
R&D PROJECTS

MERASA & PARMERASA (FP7)

- New multi-core processor designs for hard real-time embedded systems to guarantee analyzability and predictability;
- Creates the concept of hard real-time (HRT) and non-hard real-time (NHRT) threads in hardware;
- The HRT thread is completely isolated from the other NHRT threads running in the same core and is prioritized;

T-CREST

- Objective is to create a multi-processor system-on-chip (SoC) architecture designed since its inception to fully time-predictable;
- Explores WCET-aware compilation, local memory, core-to-core communication, stack caches, etc



PROARTIS & PROXIMA (FP7)

- These projects introduce probabilistically time analyzable (PTA) techniques for multicore platforms;
- Treat the system as a statistical system that has timing behavior that forms a statistical distribution;
- Hardware architecture needs to be randomized (Random caches policy, etc)