



EagleARM

Porting EagleEye to the ARM Architecture

Dr. Mattias Holm (Terma)

Vicent Brocal (FentlSS)

Alberto Ferrazzi (Terma)

TERMA[®]
ALLIES IN INNOVATION

Project Overview



- Schedule: 1 y (actual 1.5y)
- Purpose
 - Evaluate the ARM Architecture
 - Port the EagleEye software to ARM
- Consortium:
 - Terma (prime): SVF + CSW
 - FentISS: XtratuM hypervisor
 - Enea Romania: ARM study
- Work Performed
 - ARM processor study
 - Integration of TEMU in the EagleEye SVF
 - Porting of EagleEye DMS and payload partitions from Ada to C
 - Replacing OBOSS with libpus
 - Porting the EagleEye flight software to ARM

Work Logic



- Kept existing architecture -> Low technical risk
- Approach (stages, only change one major thing):
 - Integrate SPARC version of TEMU on SVF, rerun tests with old EagleEye
 - Only emulator changed
 - Port Ada code to C, rerun tests on SPARC in TEMU
 - Only the DMS and payload code changed to C
 - Port EagleEye (C-version) to ARM, rerun tests in TEMU (but with ARM model)
 - Only the CPU architecture changed

Porting Eagle Eye to ARM



- Challenges
 - Select hypervisor
 - XtratuM ARM version to minimise changes, existing EagleEye used XtratuM for LEON3
 - No Ada compilers for the hypervisors
 - Port to C
 - OBOSS Ada PUS library
 - Replacement is libPUS
- Integrate new emulator:
 - TEMU SPARC in SVF
 - Switch CPU to ARM in configuration file

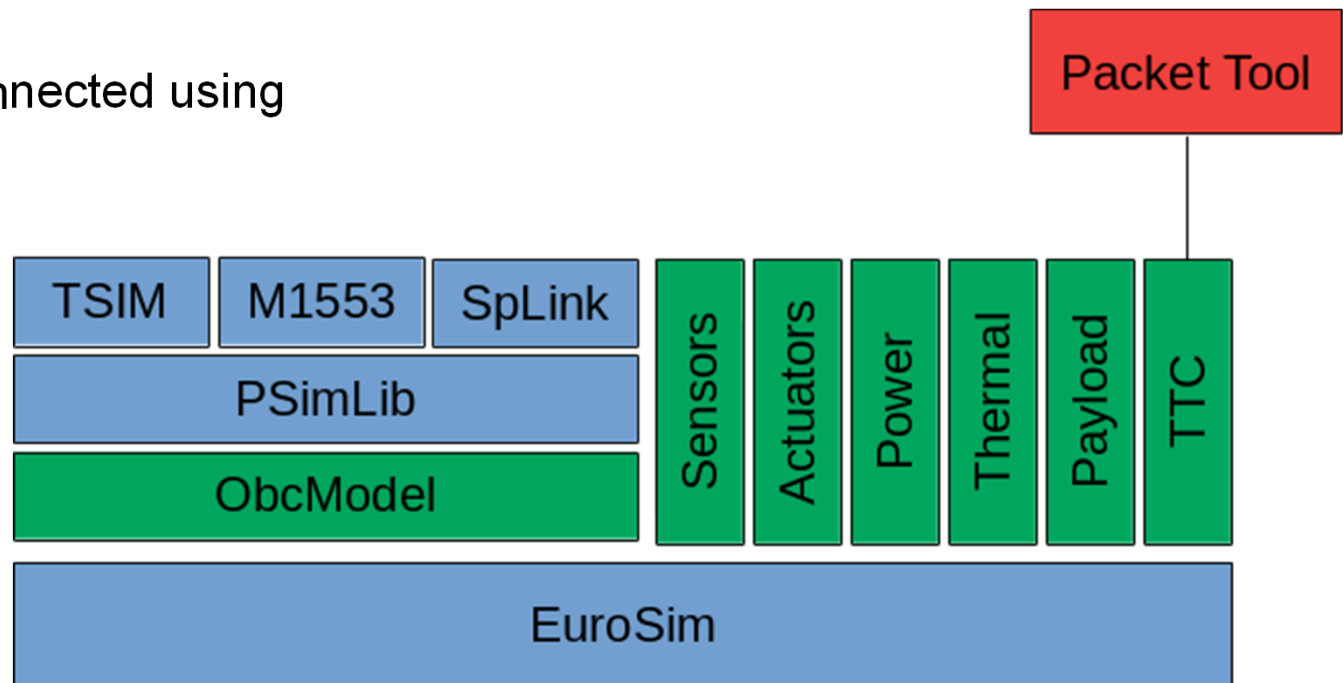


- High performance emulator
- Supports SPARCV8, ARMv7 and PowerPC instruction set simulation
- Supports full system simulation
 - Events
 - Publication
 - Checkpointing (“simulator breakpoint”)
 - Easy to use modelling APIs
- Built in bus models to ease the development and interoperability of RTU models:
 - SpaceWire
 - Milbus
 - Serial
 - CAN
 - Etc...

ATB SVF (Starting Point)



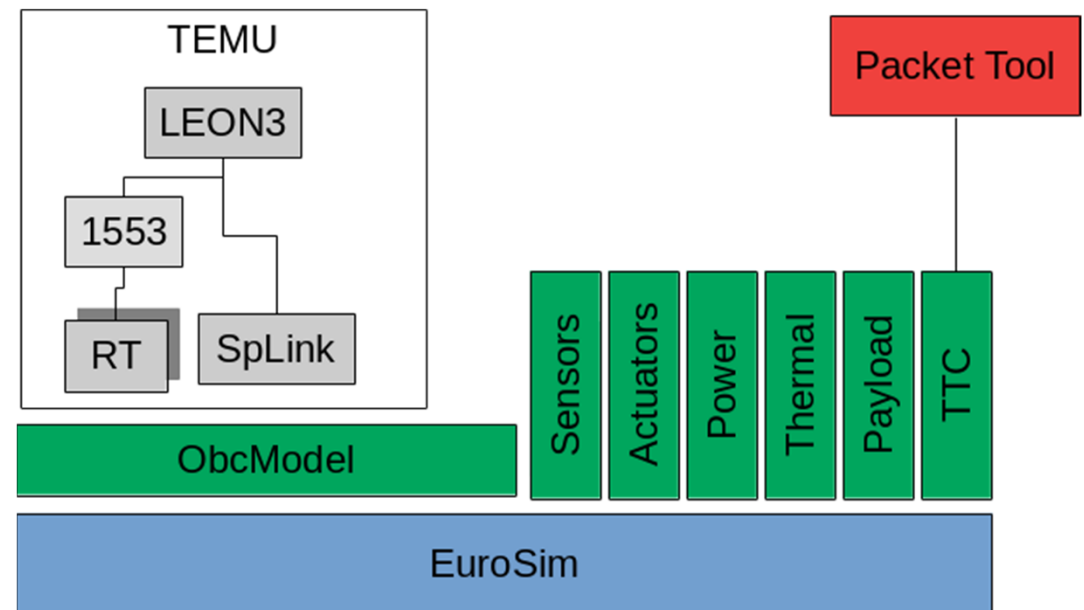
- SpaceCraft simulator for the EagleEye mission
- Based on EuroSim + SMP2
- Used TSIM as emulator
- PacketTool PUS management library
- Python based SVF test environment using PacketTool.
- SMP2 models (green) connected using SMP2 dataflow



TEMU in SVF



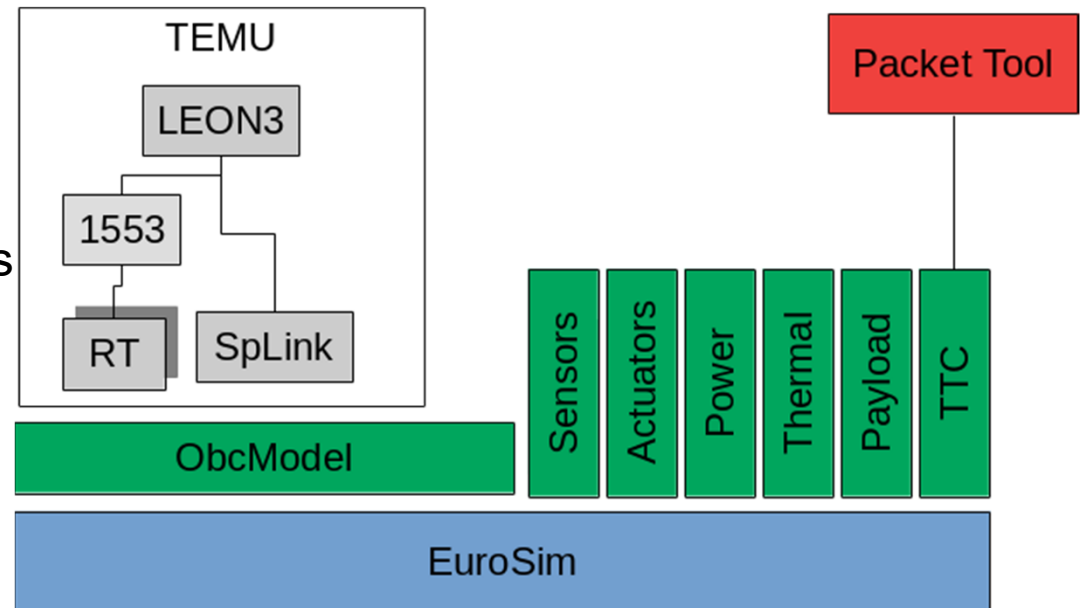
- Replaced TSIM with the Terma Emulator (TEMU).
 - Can reconfigure using scripts whether a LEON3 or ARM or any other processor is used.
- Replaced PSIMLib + custom 1553/splink with TEMU models.
- 1553 remote terminals, one per sensor/actuator (exchanging data with ObcModel).



TEMU in the SVF



- All HW configuration done in TEMU scripts.
- Stand-alone mode (without EuroSim):
 - RTU models are then simply not getting data from the environment (returns zeroes).
 - Model for injecting TCs added to be able to use TEMU debugging capabilities for TCs without dealing with the packet tool.
- 1553 traffic can be dumped using the TEMU 1553 bus traffic capture models => verified that bus utilisation was only 5% at runtime.



ARM Architecture



- Justification
 - Ecosystem size
 - User base
- Very complex instruction set
 - Load-store architecture, standard ISA (A32) is 32 bit
 - Thumb2 is variable length 16 or 32 bit
 - Instruction set is hard to decode => you cannot write an emulator for it in a day.
- Space versions
 - At the start of the project, no dedicated space qualified ARM processors existed
 - At the end of the project, products have been announced by Microchip (Atmel) and the DAHLIA SoC involving Airbus, ST, et.al.
 - Atmel SAM V71 is available
- Reliability and fault tolerance
 - Widely deployed in automotive systems
 - DCLS: Run lock stepped cores to detect failures, on failure reboot
 - No offerings with rad tolerance found
- Other features:
 - No SoCs with ARM and 1553 or SpW identified (except DAHLIA)

ARM Emulators



- All known ARM emulators support full system simulation (i.e. no need for separate simulation framework).
 - OVPSim
 - Commercial (free version for non-commercial use)
 - QEMU
 - GPL
 - Free of charge
 - Simics
 - Commercial
 - TEMU
 - Commercial
- Common Features
 - Modeling framework
 - Library based design
 - GDB server
 - Command line interface
- Built in bus models:
 - Simics
 - TEMU
- OS-aware debugging:
 - Simics
 - TEMU (planned)
- Reverse execution:
 - Simics
- Performance:
 - TEMU2 (interpreter)
 - OVPSim + QEMU + Simics + TEMU3 (binary translator)

RTOS/TSP Support



Many RTOS and TSP kernels support ARM, infact it is now a standard architecture for supporting tools:

- RTEMS: Commonly used in European space applications.
- PikeOS
- XtratuM
- Wind River VxWorks + Hypervisor
- ENEA OSE + Hypervisor
- Etc

Main problem:

- The number of ARM platforms available implies that there is probably no BSP for your system yet.

EagleEye CSW



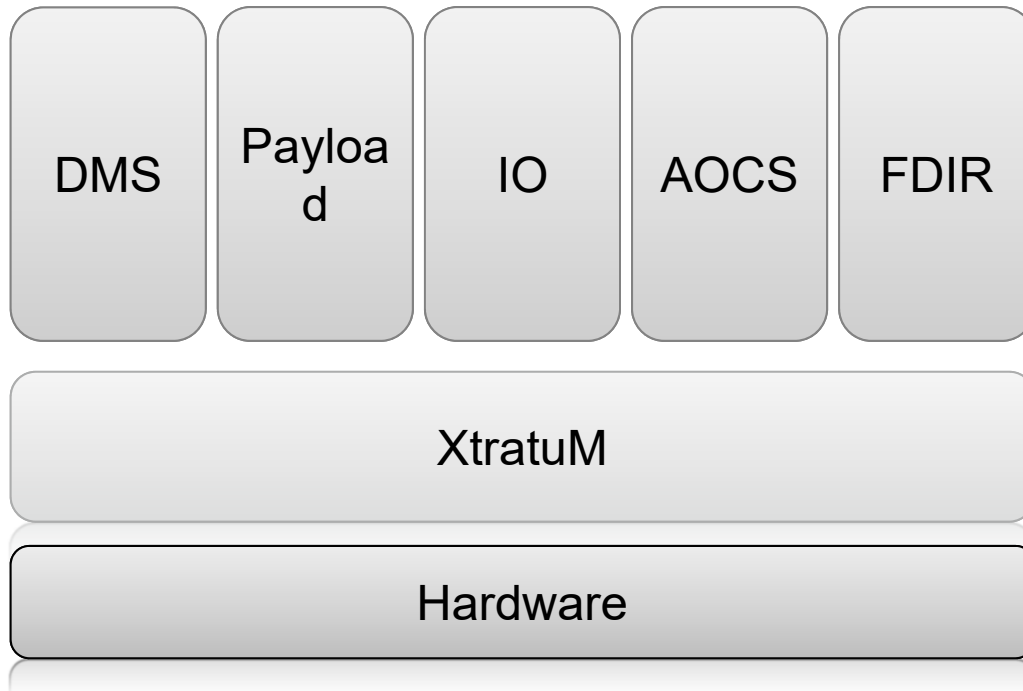
- Flight software for a virtual EO mission
 - Has existed for over 10 years
 - Auto-coded GNC/AOCS
 - Payload: GoldenEye camera
- Used to be:
 - Monolithic Ada software on RTEMS
 - OBOSS for PUS handling
- Since 2016 using TSP:
 - Partitioned into DMS, Payload, AOCS, IO and FDIR partitions.
 - DMS & Payload in Ada (still using OBOSS)

Porting to ARM



- New XtratuM configuration file
- New makefile configuration file
- Conditional compilation:
 - Interrupt handler registration (different RTEMS API)
 - I/O addresses (some I/O blocks moved due to MPU restrictions with power of 2 sizes)
- Compatibility library that remaps some XM/SPARC 3 functions to XM/ARM 1.5 functions.

TSP in the context of EagleEye

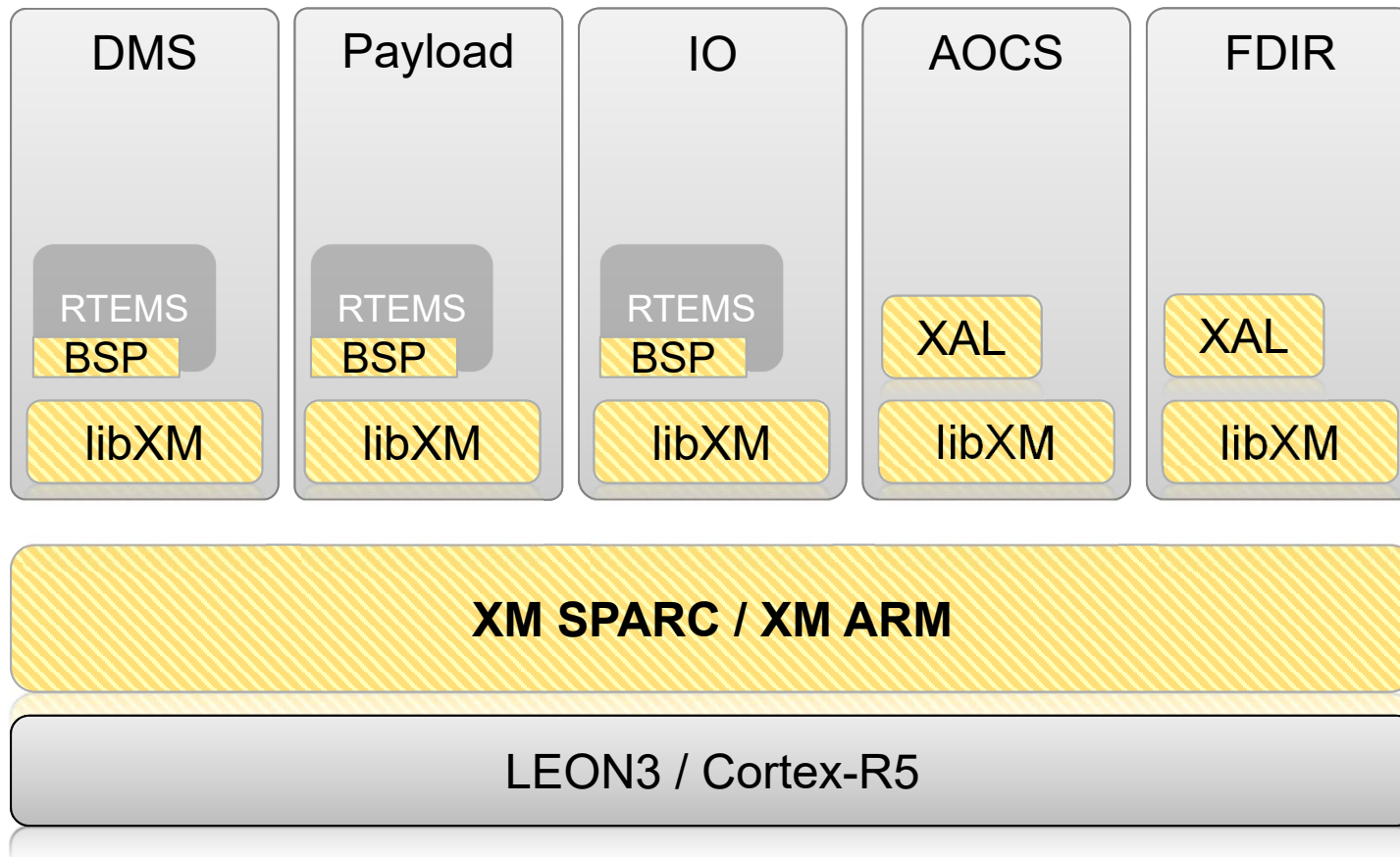


EE TSP (LEON 3)

↳ **EE HW in the loop**
(LEON3 and LEON4)

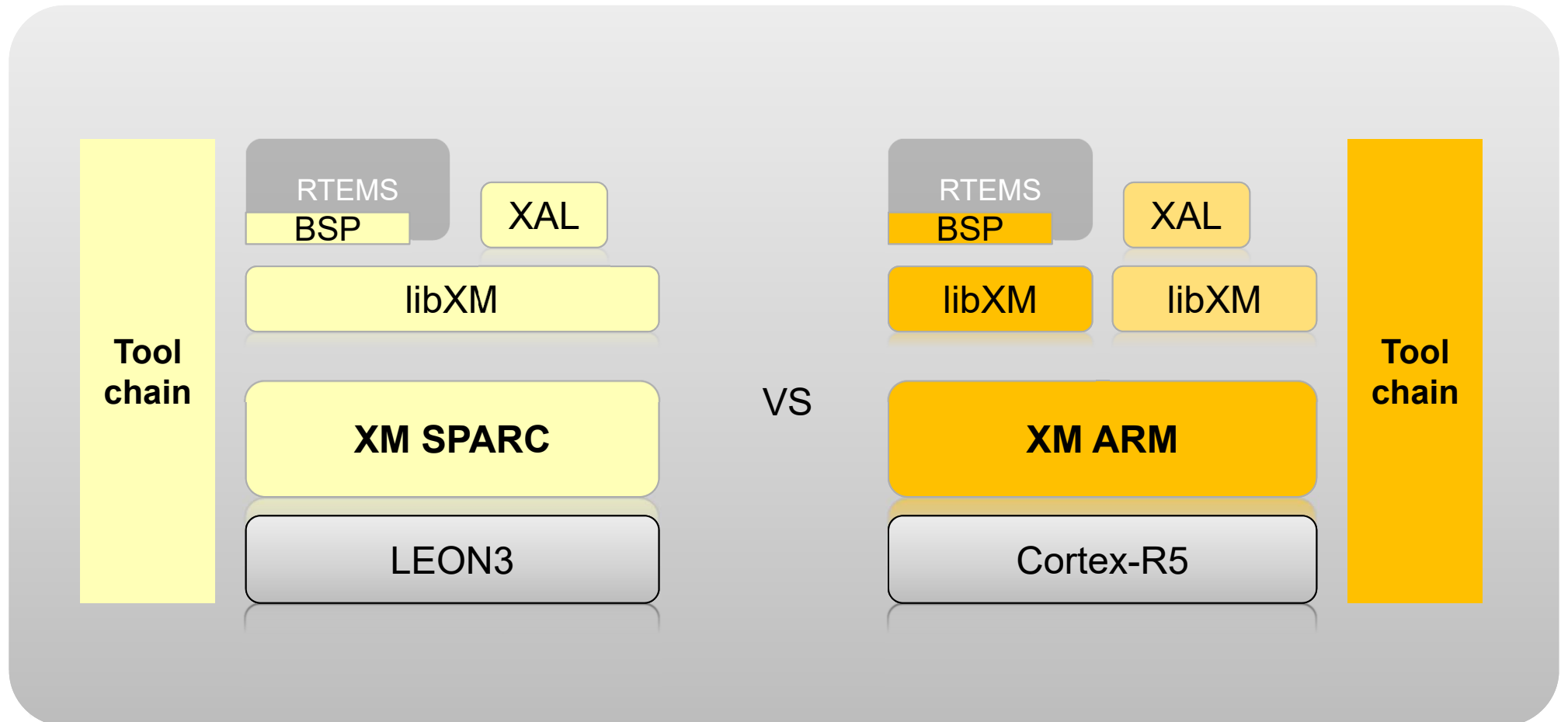
↳ **EE ARM**
(LEON3 and ARM Cortex-R5)

EagleEye support for ARM Cortex-R5



**Development
and build
toolchain**

EagleEye support for ARM Cortex-R5





SPARCV8:

LEON2

LEON3 (qualification end of year - ESA)

LEON4

ARM:

Cortex R4/R5 (ARMv7, Monocore)

Cortex A9 (ARMv7, SMP)

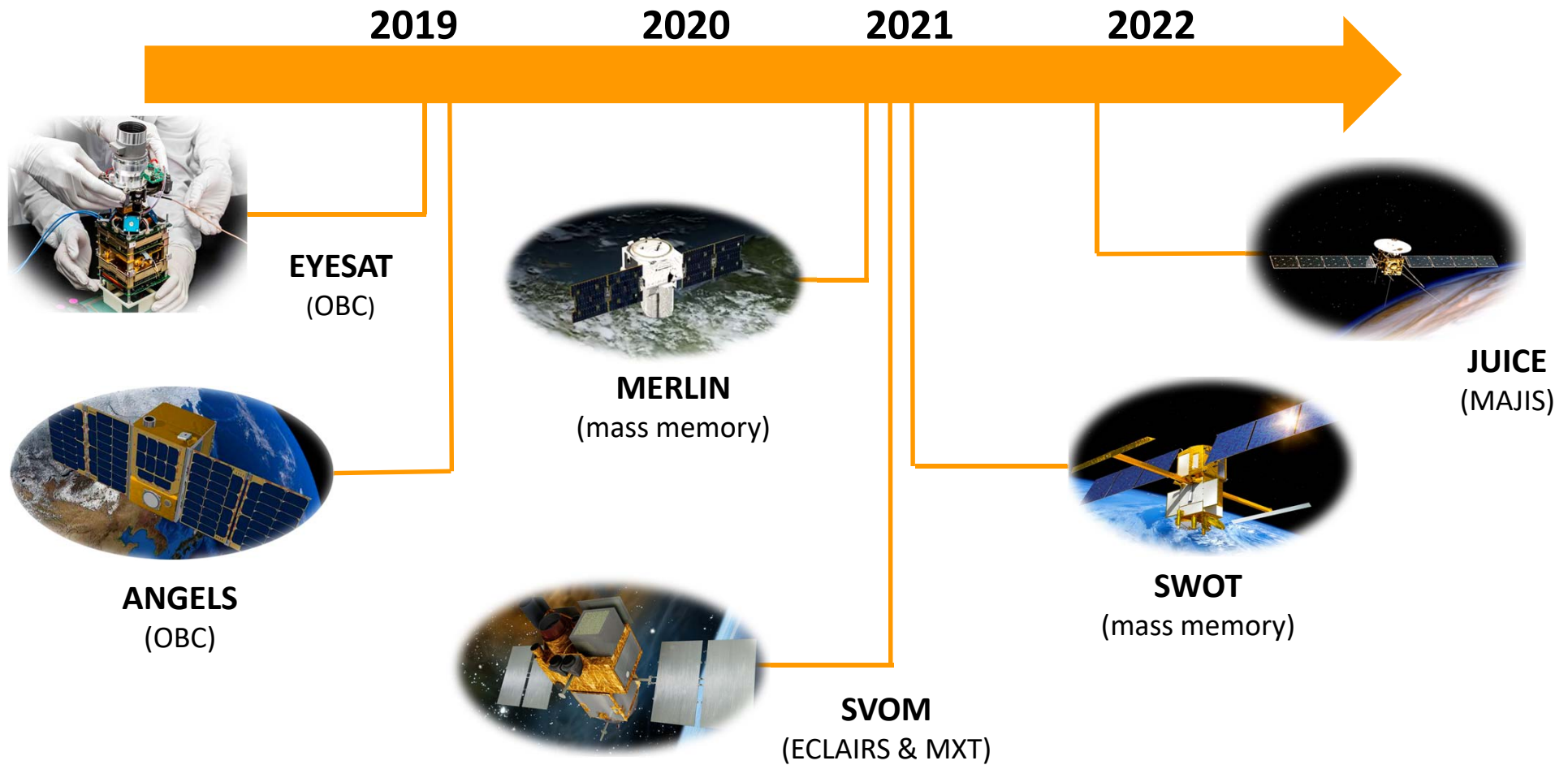
Cortex A9 (ARMv7, monocore, qualification end of year - CNES)

Zynq UltraScale+ (planned 2019)

Corex R5 (AMP)

Cortex A53 (SMP)

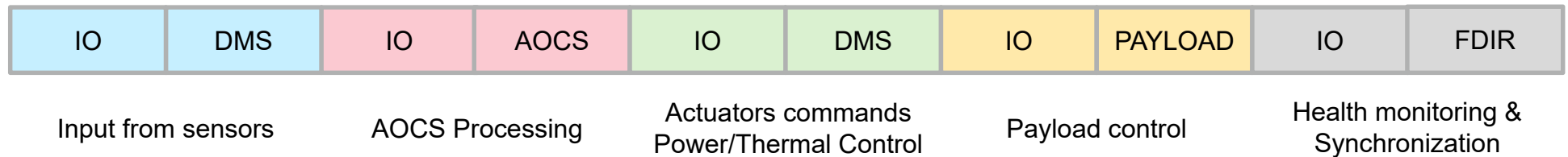
TSP is not a promise anymore



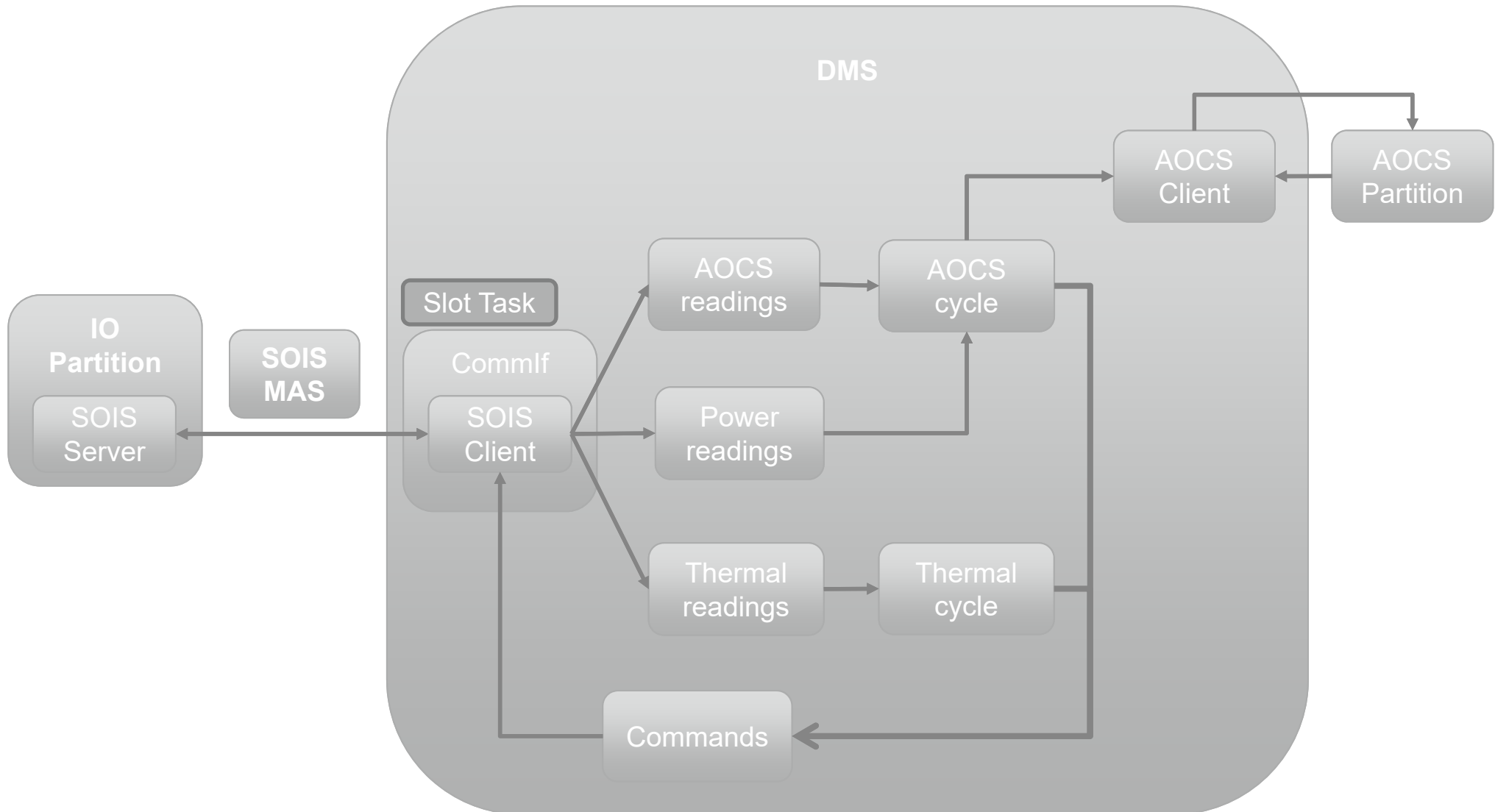
CSW – Partition schedule



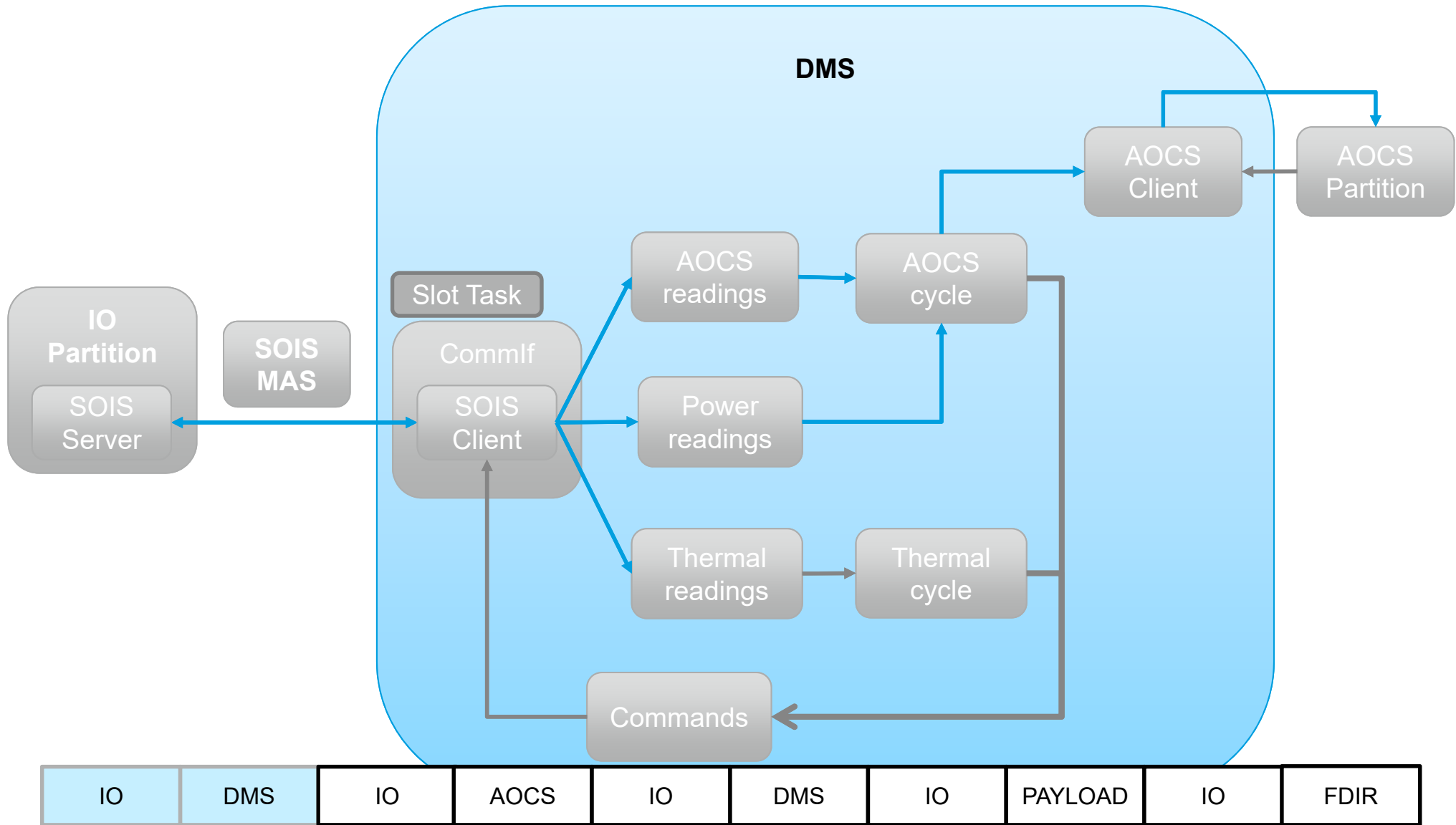
- Tailored around AOCS
 - Cyclic (250ms)
 - Fixed time slots



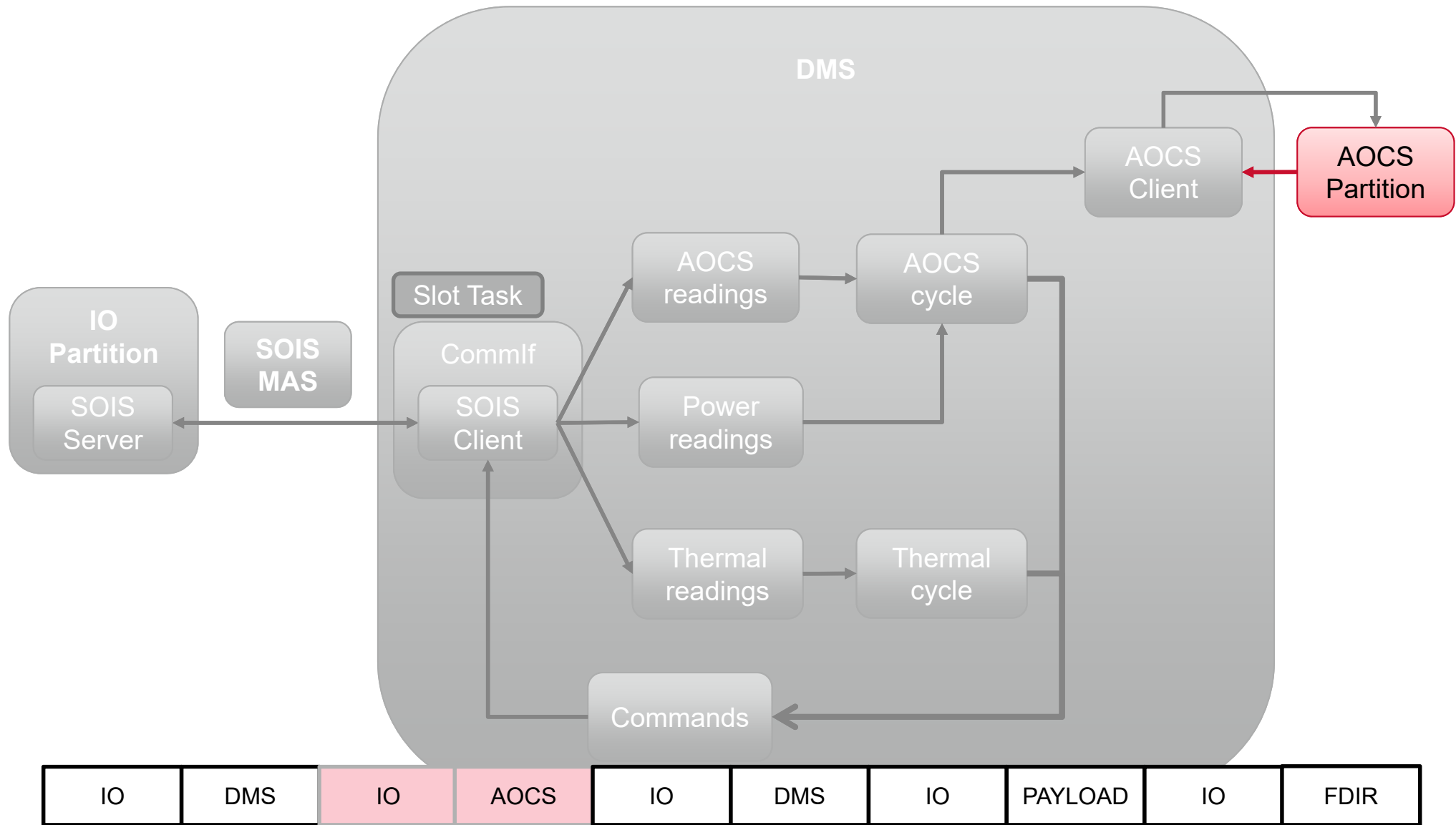
CSW – DMS Sensor/Actuators



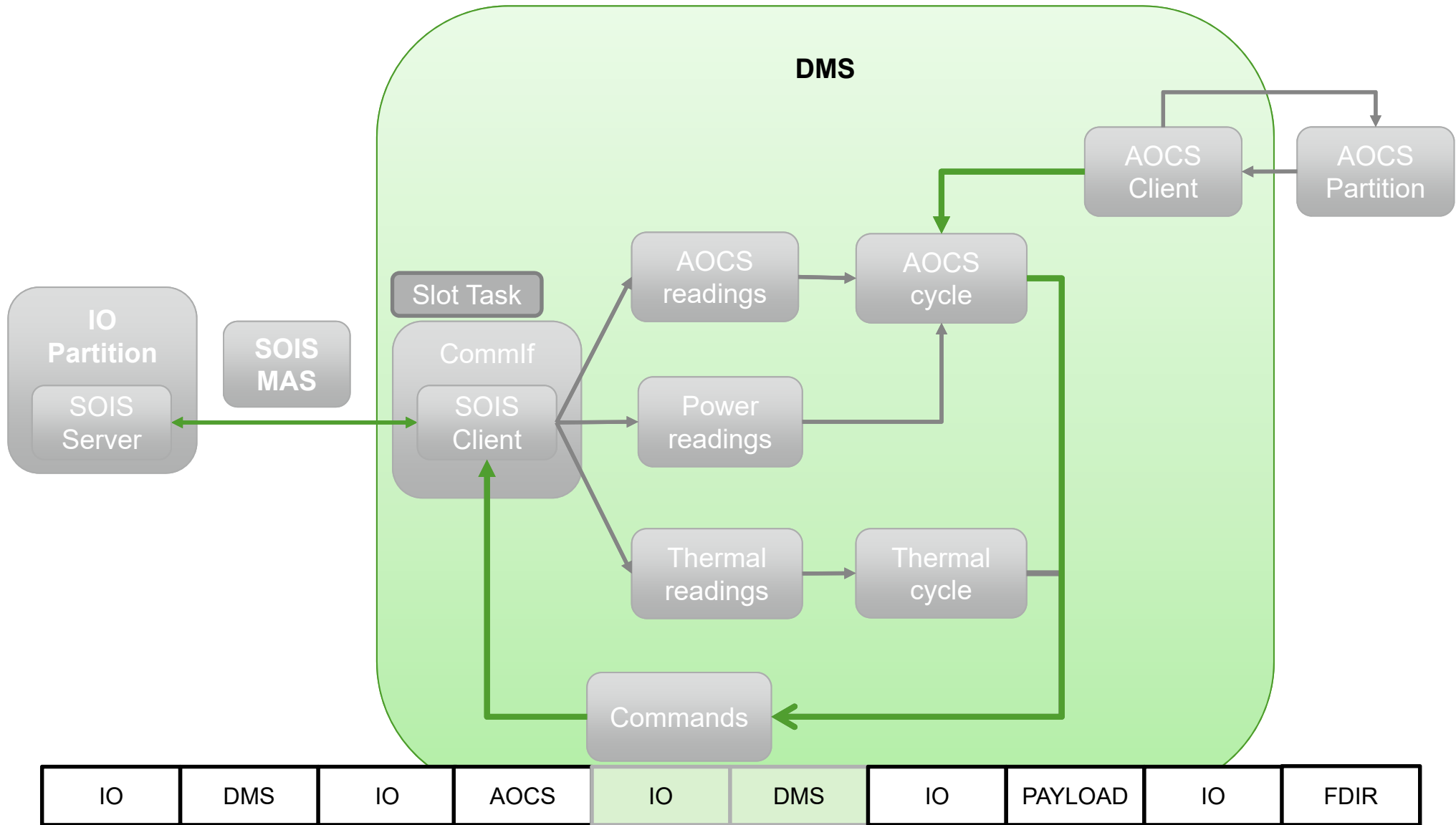
CSW – DMS Sensor/Actuators



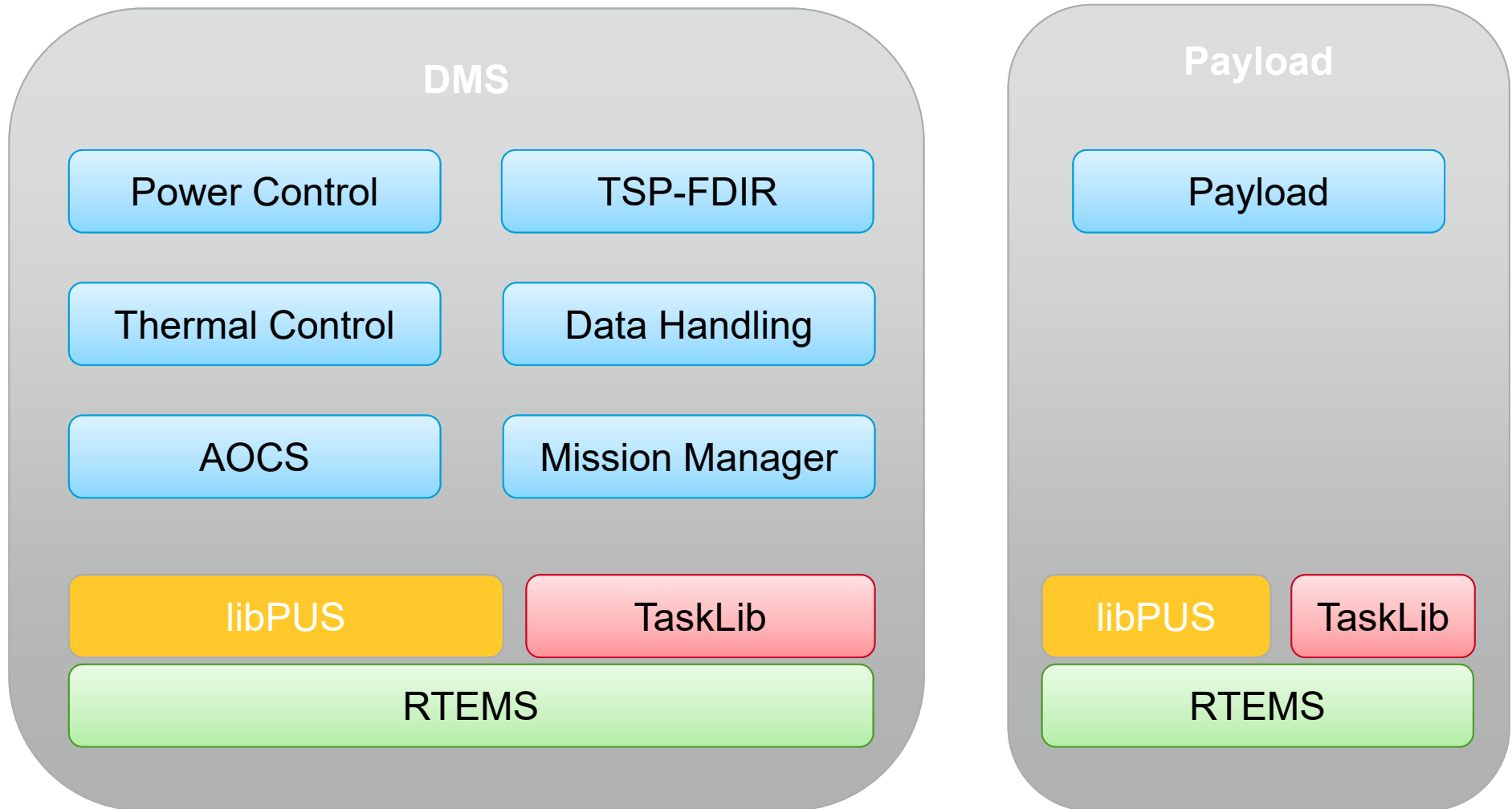
CSW – DMS Sensor/Actuators



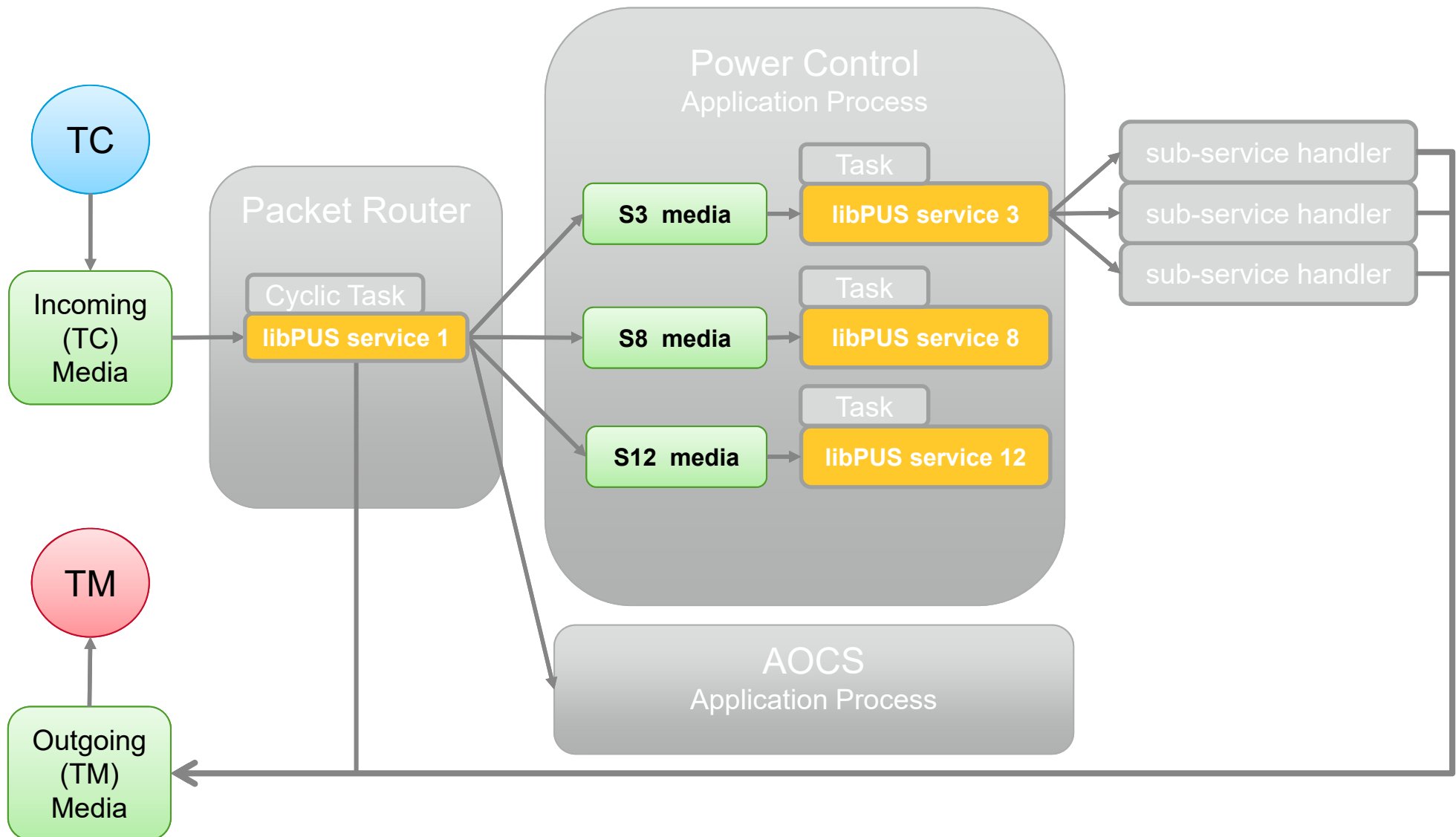
CSW – DMS Sensor/Actuators



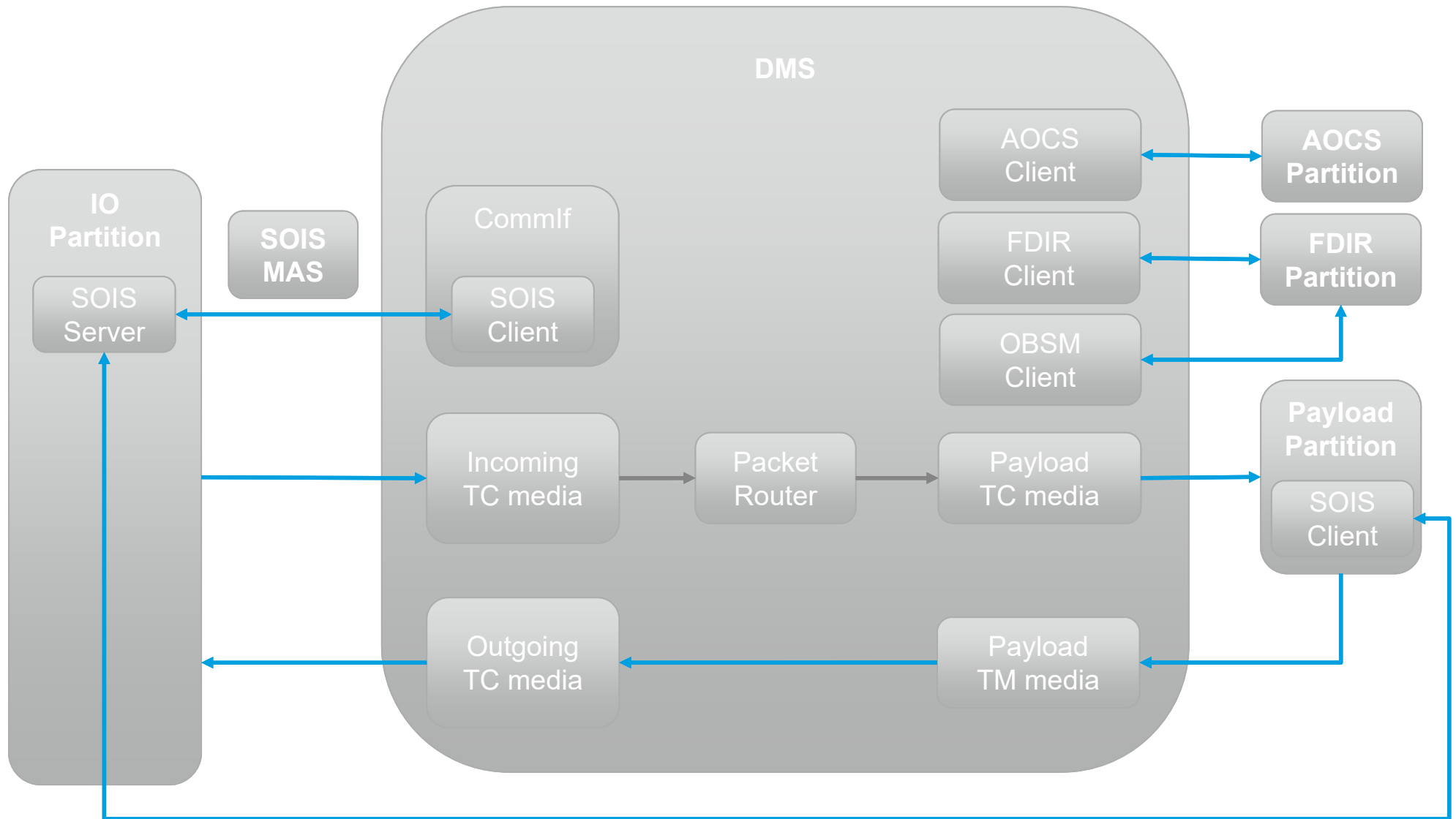
CSW – Architecture



CSW – Command handling



CSW - IPC



Validation of CSW



- Re-ran existing validation suite
- Modifications done due to libPUS limitations:
 - Define / clear HK report definitions not supported out of the box, removed, report definitions hardcoded in SW instead of the MIB).
 - Progress reports not supported out of the box, progress reports removed from MIB and checks in the test-suite.
- Validation suite (confusingly called an “integration test”) is written in Python and is a systems test of the CSW, it verifies TC/TM responses and expected HK changes.

Lessons Learned and Some Final Results



- LibPUS has a steep learning curve.
 - Limited documentation, could be improved a lot (examples, tutorials etc).
 - Easy to work with when one understands it.
- Software budgets changed with respect to code size
 - ARM version showed a 38% reduction in code size:
 - Reason: Variable length Thumb2 mode
- Data sizes remained roughly the same:
 - 32 bit architectures
 - Same optimal alignment rules

General Remarks




- Simulator design approaches:
 - Simulation centric:
 - Two simulation engines (sim framework + emulator)
 - Sim is the simulation clock on high level
 - CPU provides separate sim clock, CPU is scheduled by sim framework.
 - Bug discovered: SVF v4 was running clocks at different rates.
 - Software centric:
 - One simulation engine (emulator)
 - CPU is the simulation clock
 - Software debugging greatly simplified, simulator is easy to halt at any time, fully deterministic.
- SVF: Software Validation Facility
 - Needs effective software debugging support.
 - Problem: GDB server wants to control simulation time. Emulator provides one, but it controls the emulator, which the simulator framework wants to control at the same time.

Recommendations and Future Work



- General
 - Provide EagleEye requirements specification in machine readable format!
 - Drop 1553 and memory mapped SpaceLink models in favour of SpaceWire or Ethernet (SpW terminal model can talk spacelink on the other end if needed).
 - Improved packet tool. Currently has issues with TMTC verification (e.g. verification chains expecting a 1,8, but getting a 1,7 -> tests hang in many cases).
 - Would be better with a better maintained TMTC based test environment (e.g. TSC).
- CSW
 - Build system for CSW
 - Migrate from Makefiles to CMake
 - TMTC Definition Language
 - Current system uses XML files, which are hard to modify and maintain.
- SVF
 - Stand Alone Mode
 - Useful for flight software debugging to be able to run OBSW without EuroSim (or environmental) models in the loop.
 - SVF Build System
 - Currently a mixture of shell scripts and Makefiles, the OBC model use CMake.
 - Beneficial to move everything to CMake.

A horizontal bar with an orange segment on the left and a white segment on the right.

Meet us at

www.terma.com

www.terma.com/press/newsletter

www.linkedin.com/company/terma-a-s

http://instagram.com/terma_group

www.twitter.com/terma_global

www.youtube.com/user/TermaTV

TERMA^T
ALLIES IN INNOVATION