Exploiting AMBA AXI protocol for Denial-of-Service attacks of shared resources

UC San Diego

Francesco Restuccia - hardwear.io 06/09/2022

Who am I

Currently: Postdoctoral researcher @ UCSD

Ph.D. computer engineering @ Retis Lab, Italy (2021)

Hardware security - access control systems

Timing predictability, safety, and security for FPGA SoC platforms

Time-predictable DNN acceleration for FPGA SoC platforms

Francesco Restuccia

Used to have longer beard

In a nutshell

Explore, analyze, and address safety and security concerns in a popular on-chip communications standard

Detected threats

Circular dependencies among hardware modules able to threaten the availability of the shared resources

Unfair bandwidth distribution

Denial-of-service of shared resources

Lesson learned, solutions, and guidelines

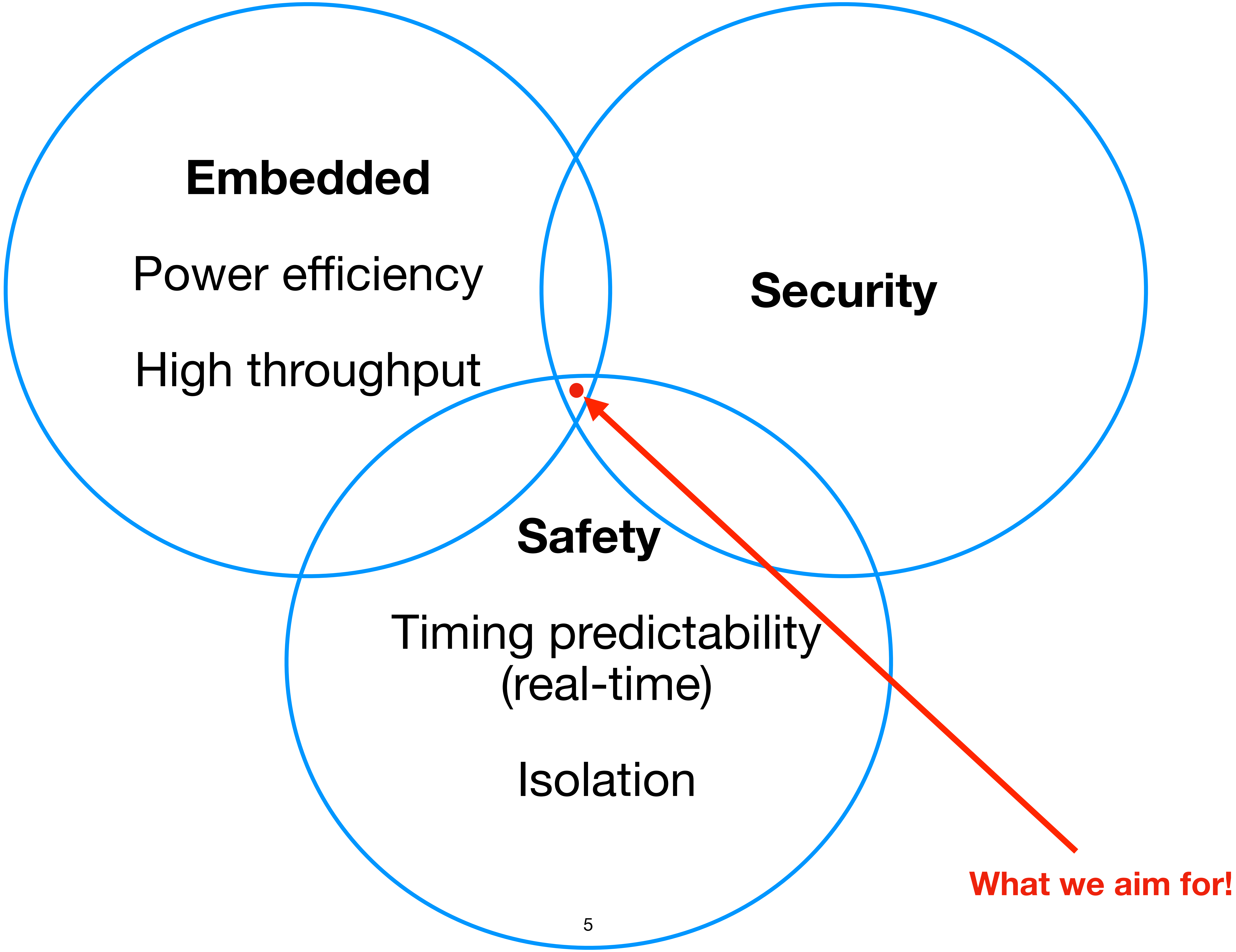
Critical computation systems

“Safety-critical systems are **those systems whose failure could result in loss of life, significant property damage, or damage to the environment.**”

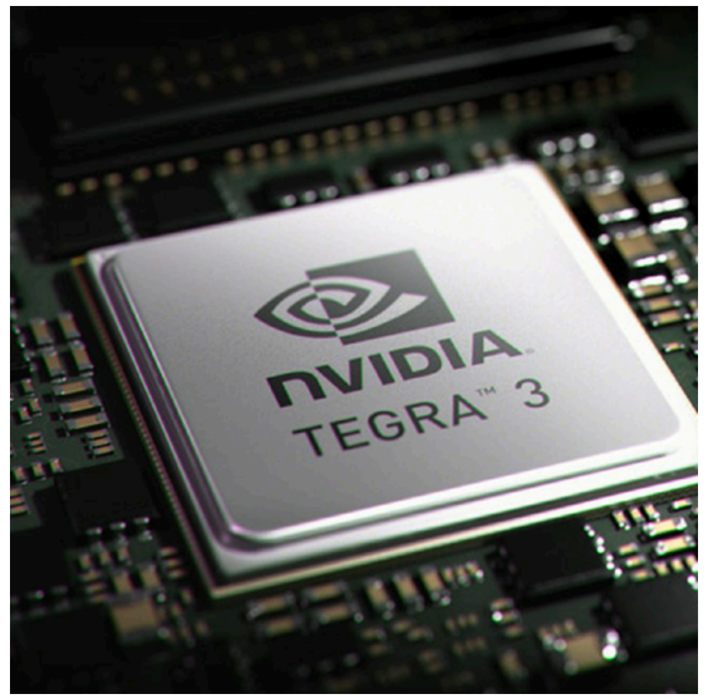
Credits: https://www.bostondynamics.com/

Avionics, space applications, cars (autonomous), robots, medical devices

Typical requirements



Popular heterogenous platforms



GPU SoCs

Credits: NVIDIA corporation

FPGA SoCs

Credits: Xilinx Inc

Custom SoCs

Specialization

Performance/Power Ratio

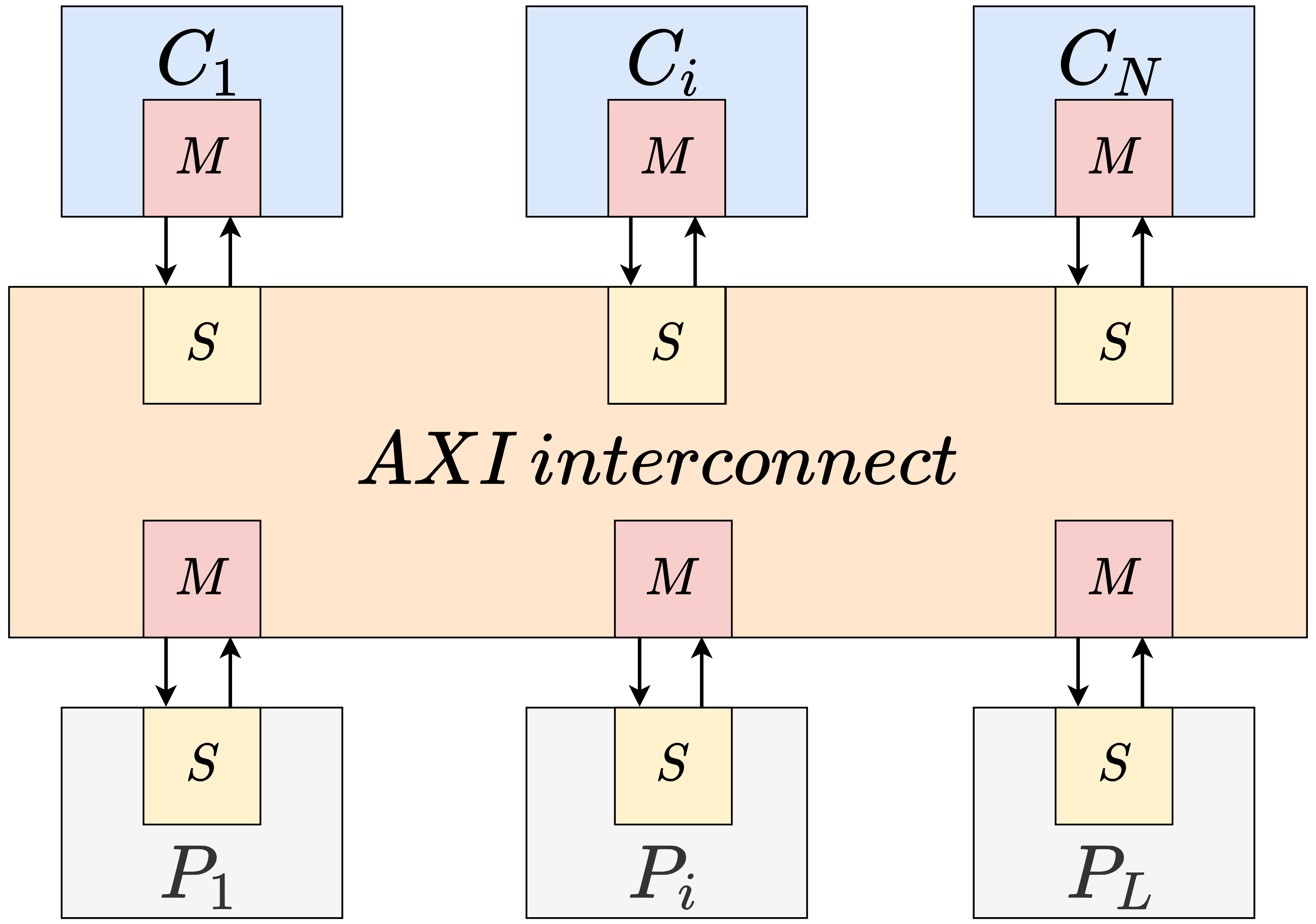
Anatomy of a typical heterogenous platform

Multiple heterogenous modules (controllers + peripherals)

Controllers -> active (processors, DMAs, hardware accelerators, etc.)

System interconnect

Peripherals -> passive (Memories, IO, etc.) (memory-mapped)



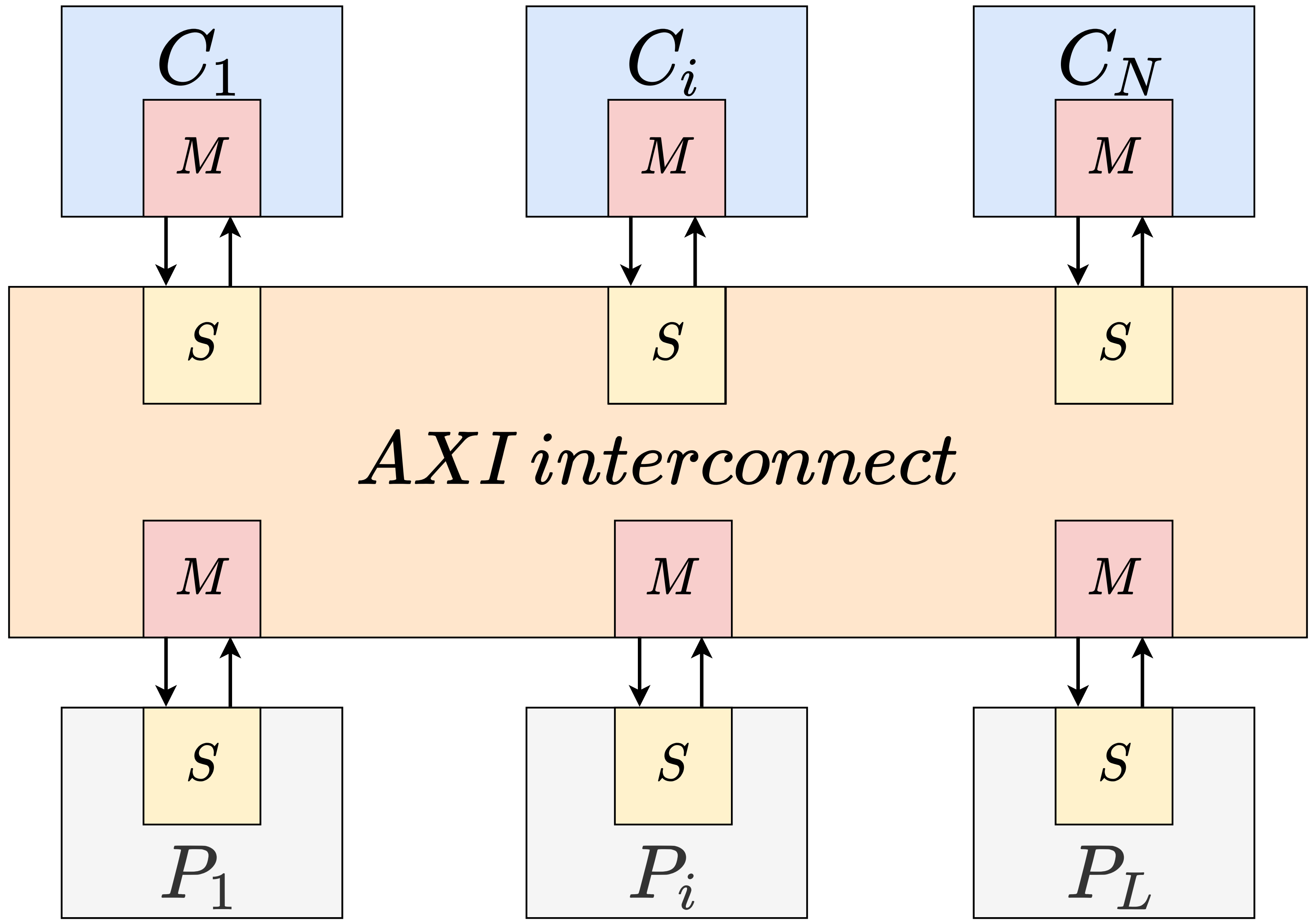
Interactions among modules

Peripherals are typically shared among controllers

Interconnect arbitrates the interactions among controllers and peripherals

Multiple controllers generate **interference** on the **shared resources**

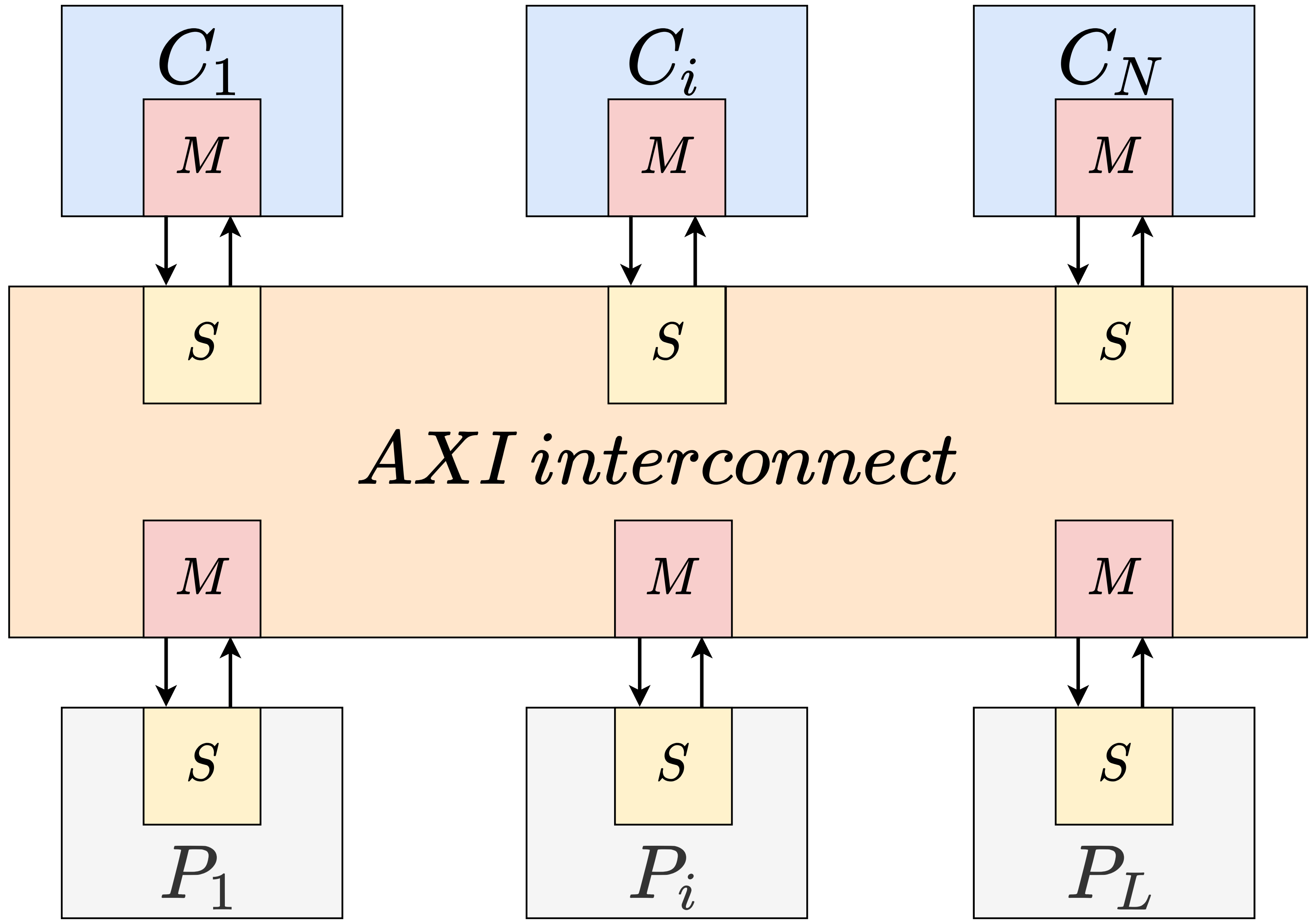
Heterogenous modules -> heterogenous interactions



The AMBA AXI standard

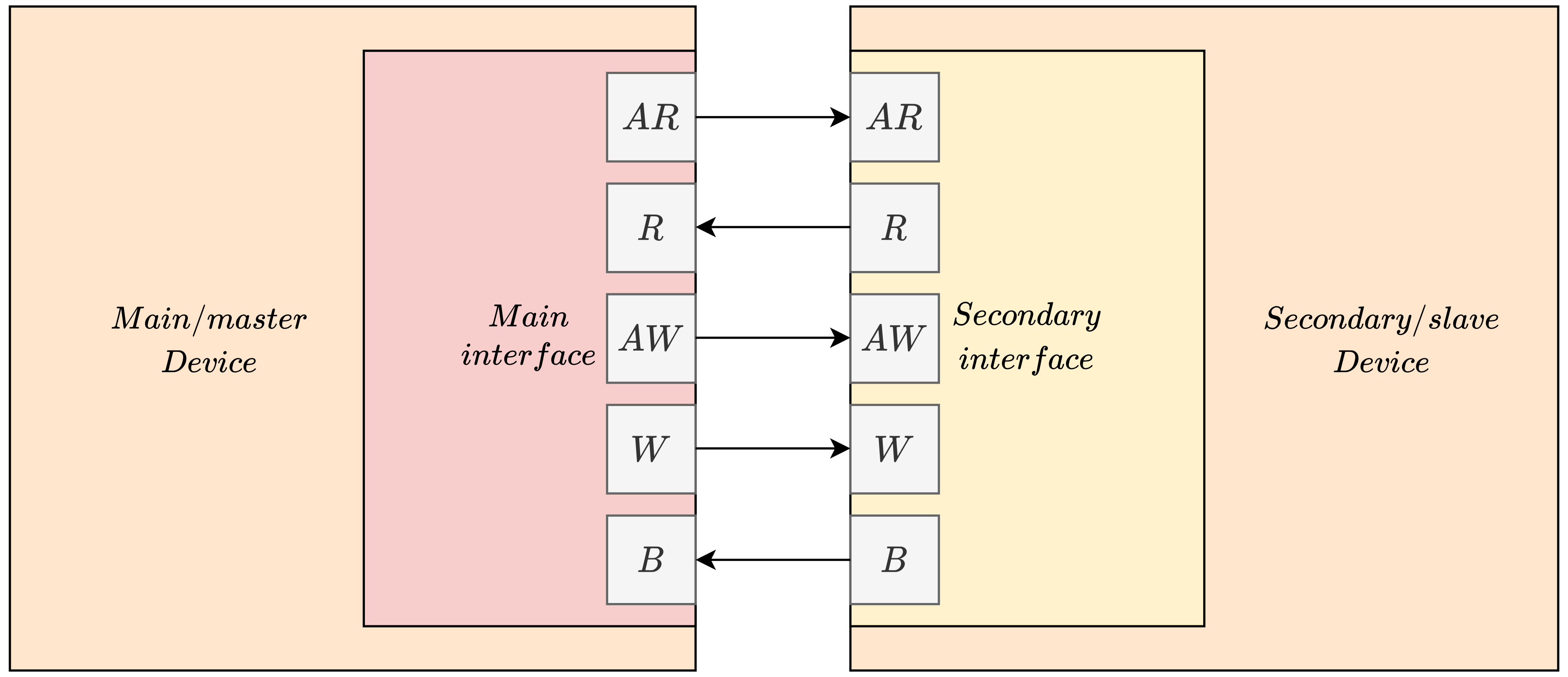
Popular standard for communication on modern heterogeneous SoCs

Each controller has a **separated** communication interface -> **isolation** (electrical)

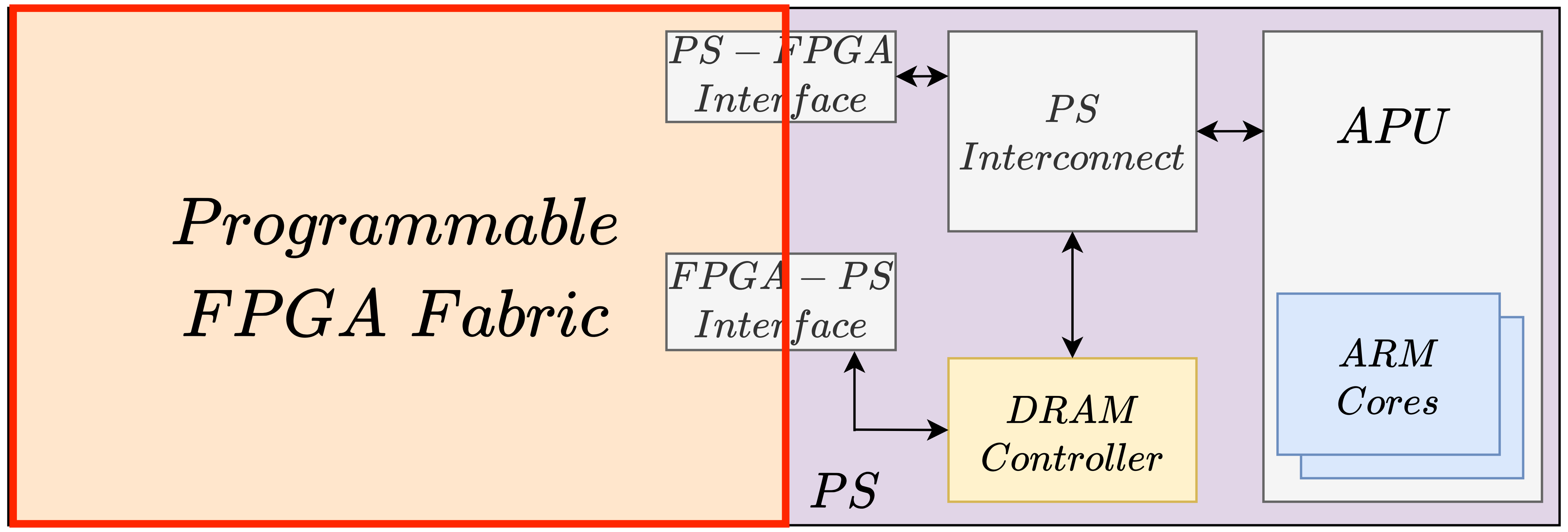


AXI defines a manager/subordinate interface

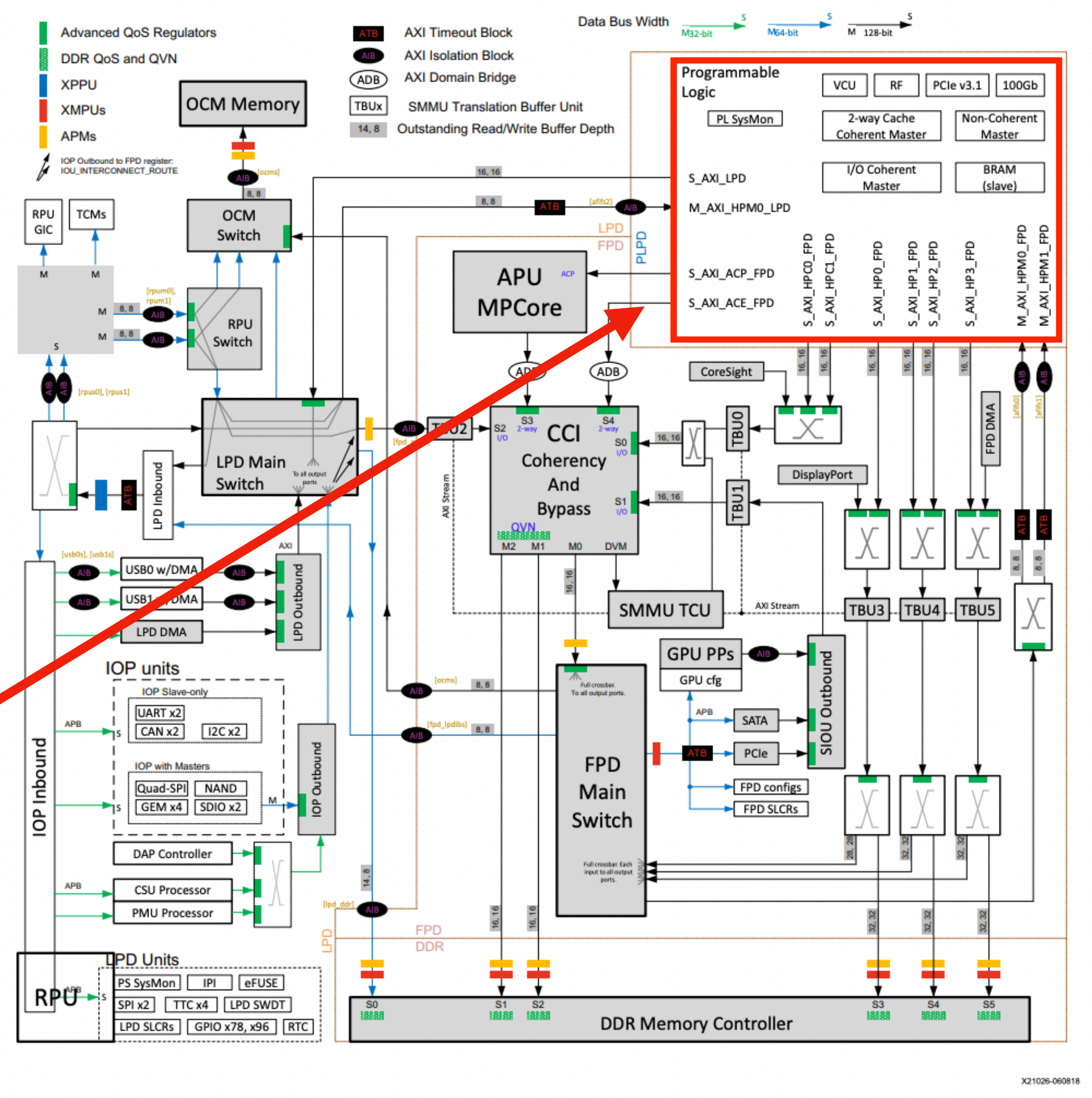
5 channels, handled independently



A real platform example



Credits: Zynq Ultrascale trm



Hardware accelerators deployed in the FPGA fabric as our test **controller managers**

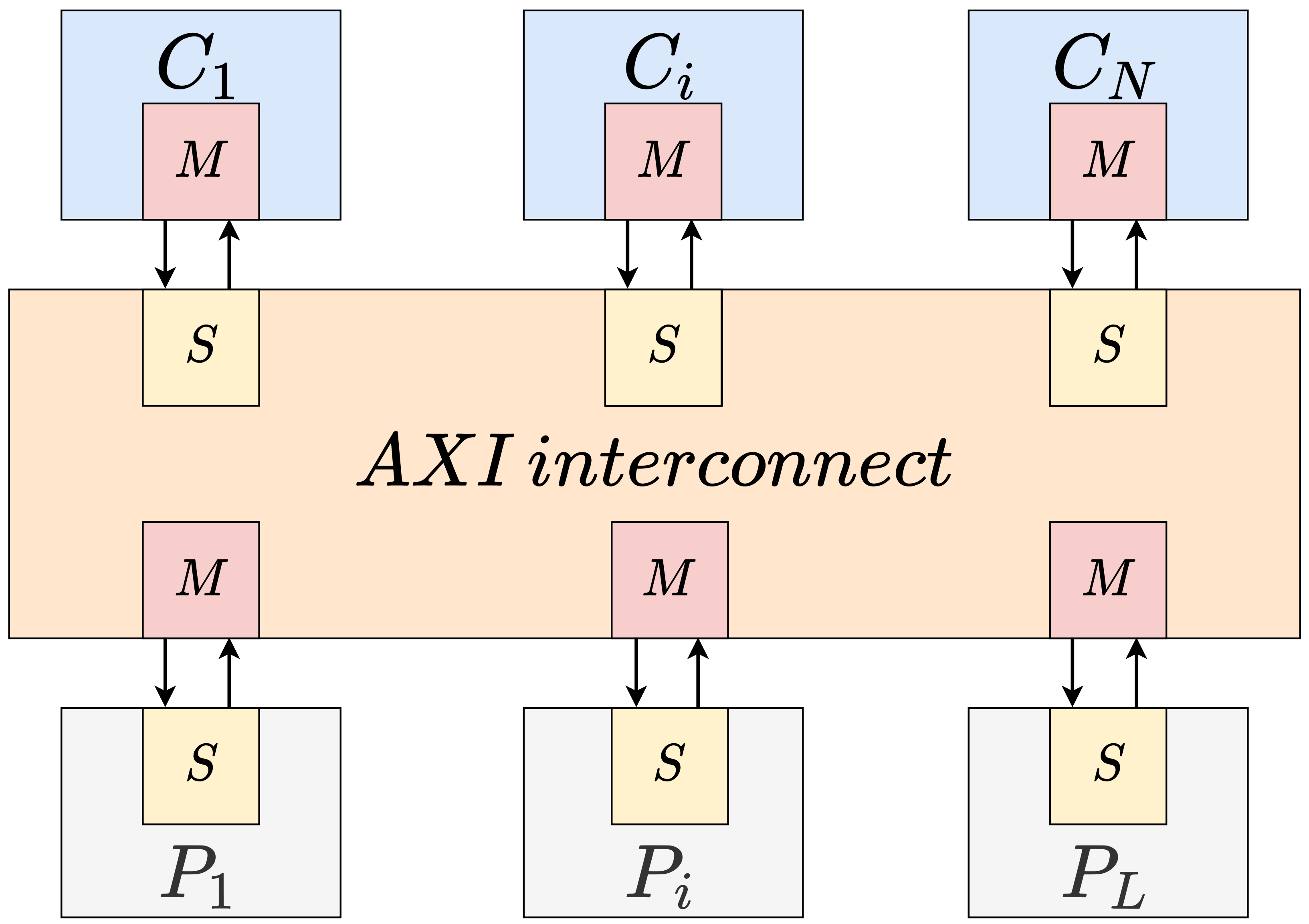
Threat model

Trusted

Interconnect and peripherals

Untrusted

Controllers (Third-party IPs, affected by bugs, superficial security verification, etc.)



We focus on the availability (to the controllers) of the shared peripherals during the system execution

The beginning of our journey

FPGA SoCs for next-generation cyber-physical systems

Challenge: Timing predictability of bus/memory interactions

Our main aim: bound the response times of bus/memory interactions

Credits: Xilinx

Credits: Xilinx

We ended up facing safety and security issues at first

(We eventually published two papers on timing predictability afterward)

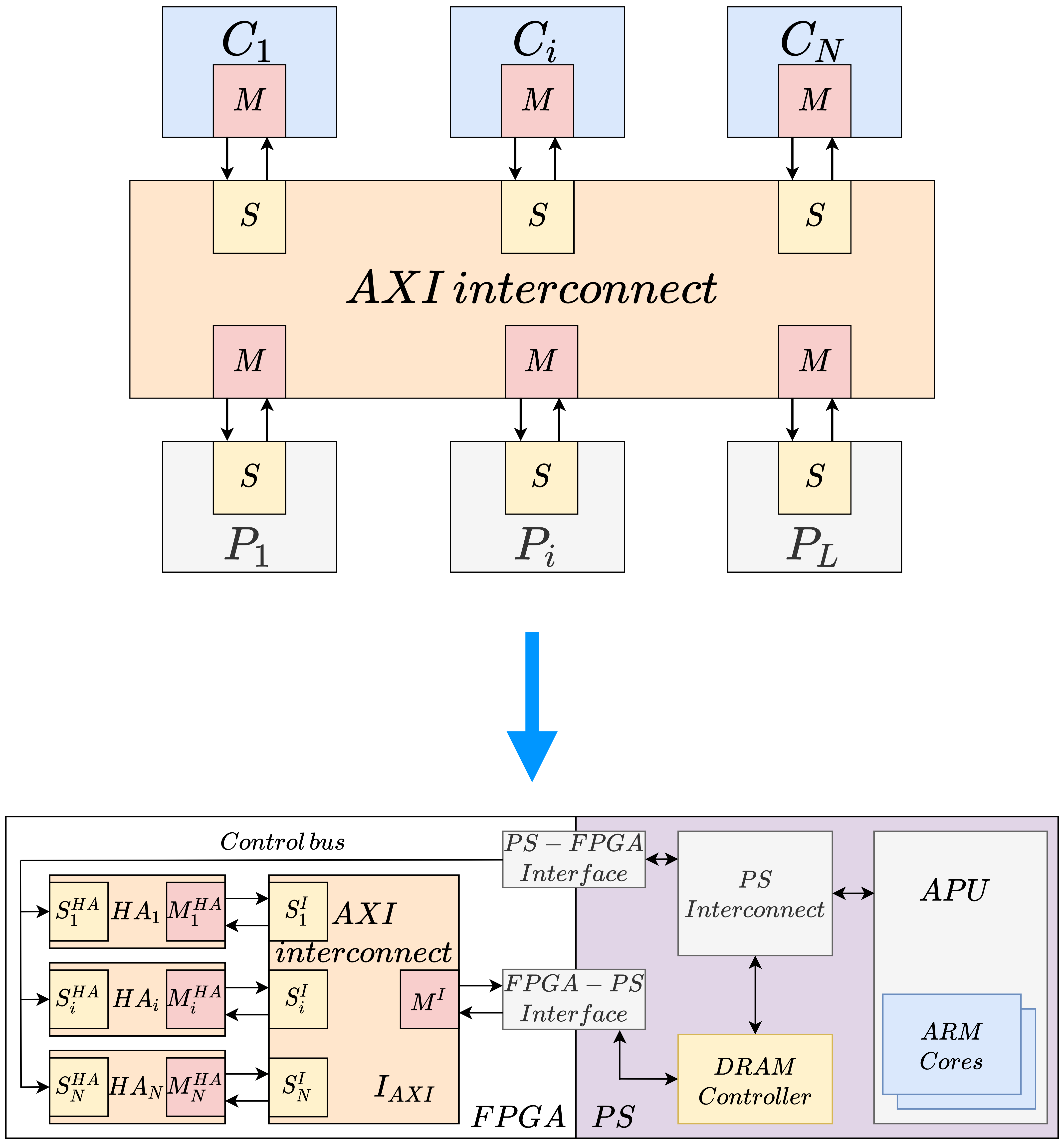
Test architecture

Three equal Xilinx DMA HAs on the FPGA

Stock AXI interconnect from the vendor

Test the assigned bandwidth to the HAs by the interconnect in accessing the shared resources (memory)

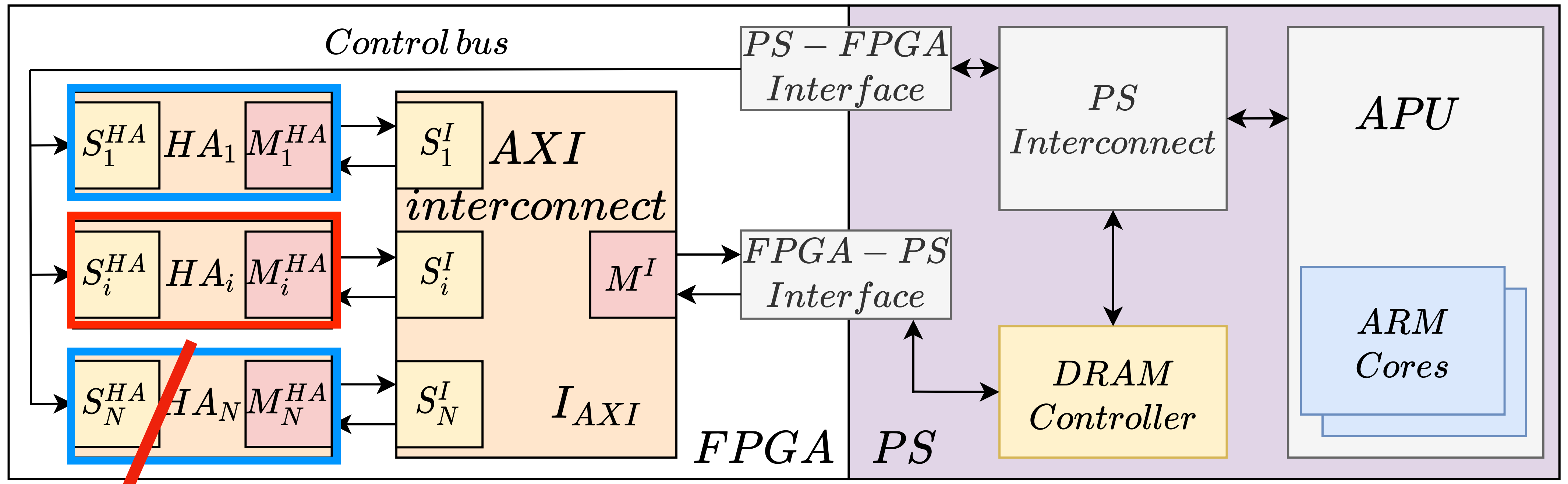
Round-robin arbitration in the interconnect: expected fair bandwidth distribution



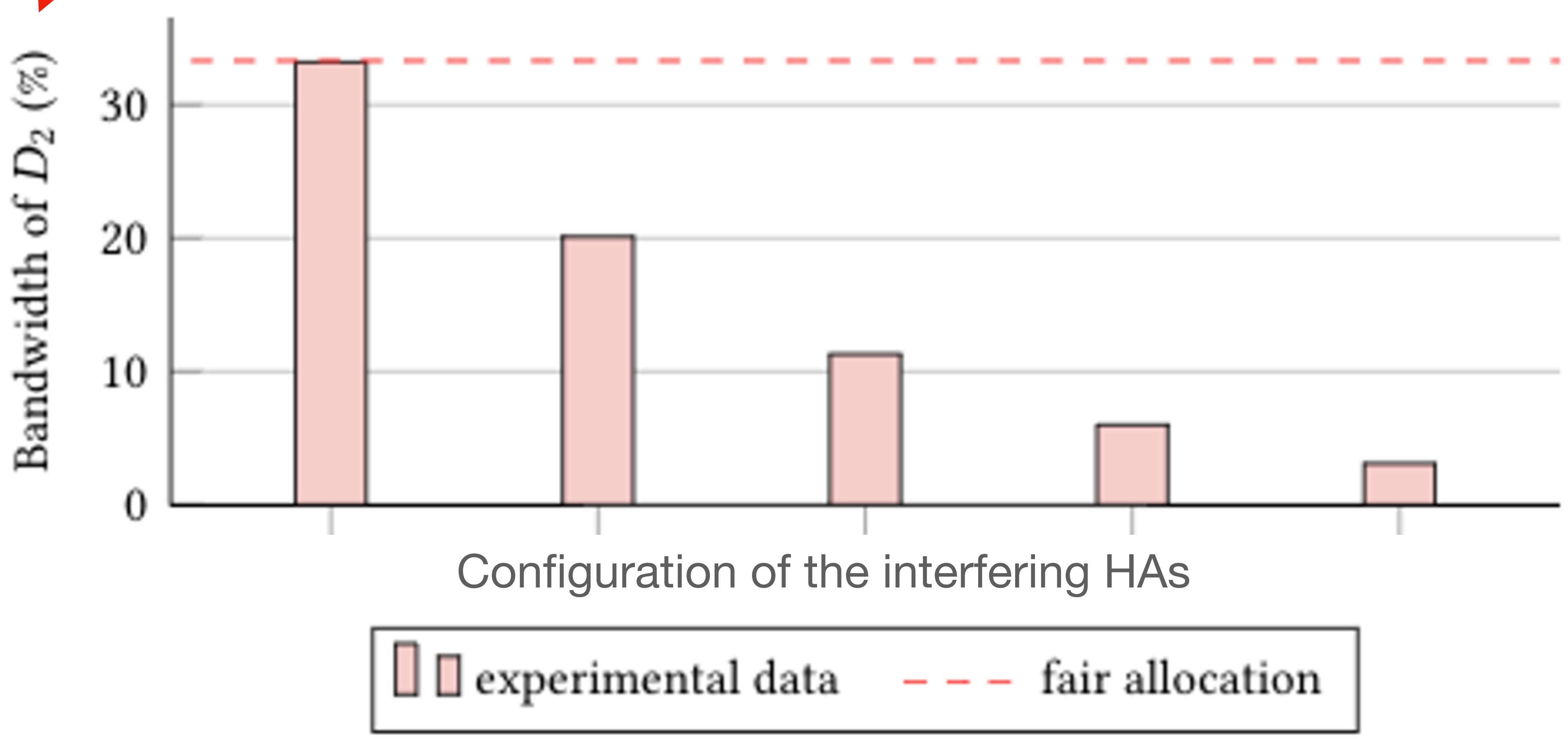
Measured bandwidth

Set one HA as the device under analysis

Set the other two HAs to generate maximum interference

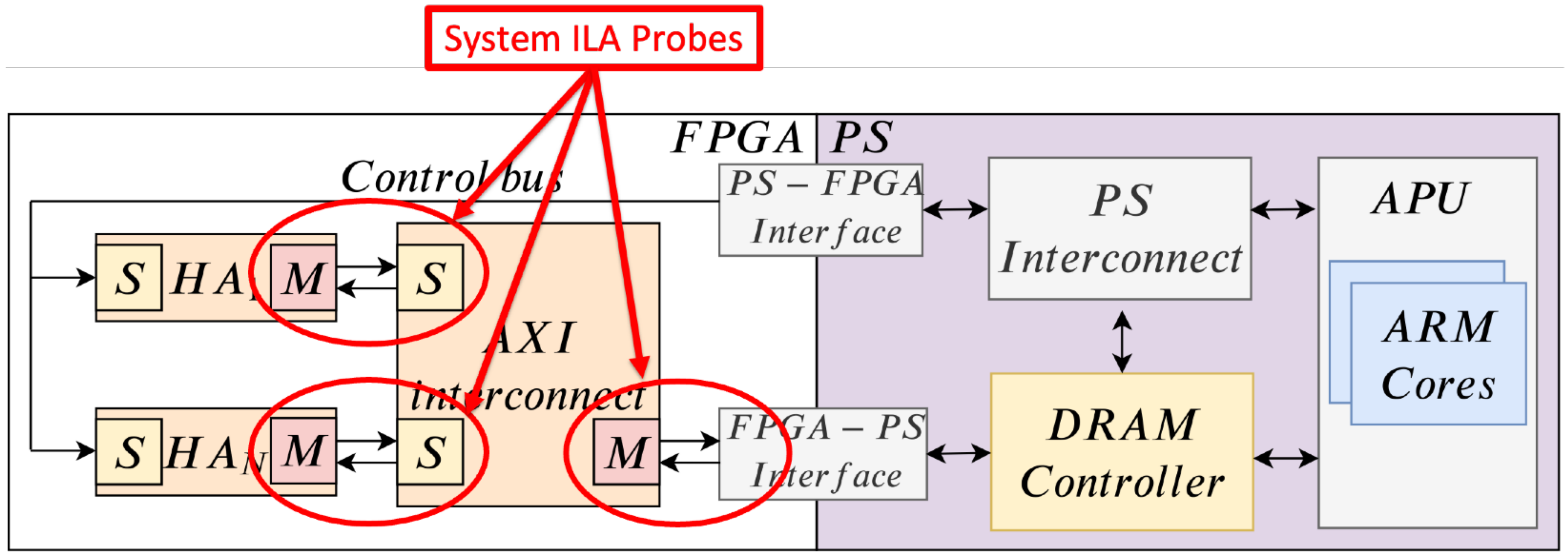


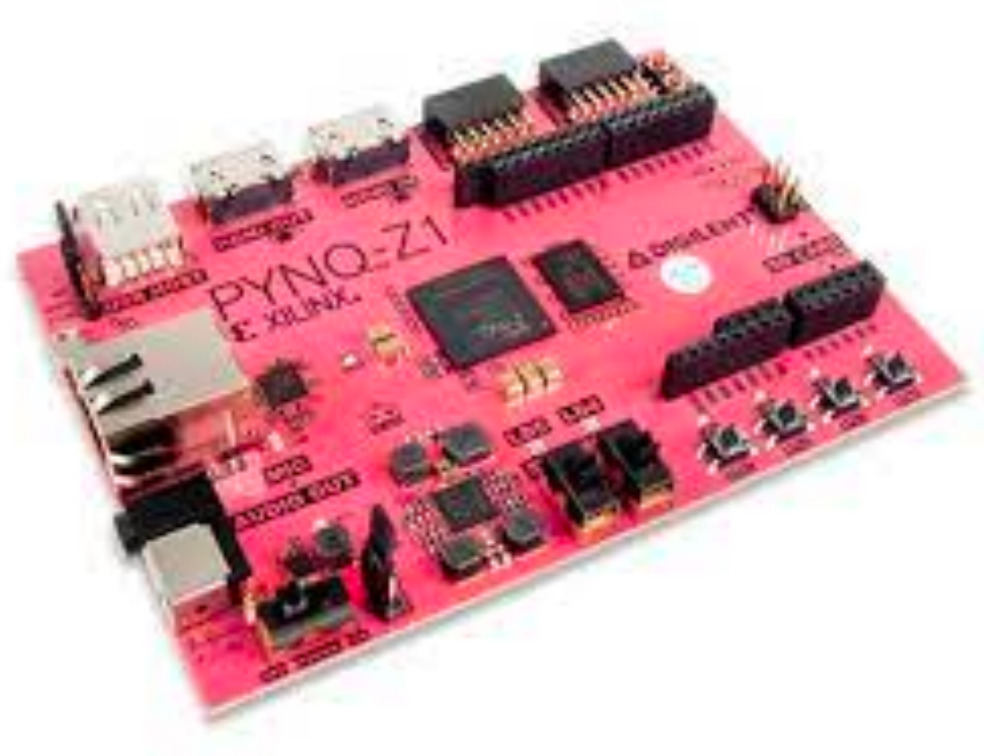
The bandwidth of a HA under analysis drops changing the configuration of the interfering HAs



Instead of being a property defined by the interconnect, the assigned bandwidth depends on interfering HAs

Investigation - hardware track





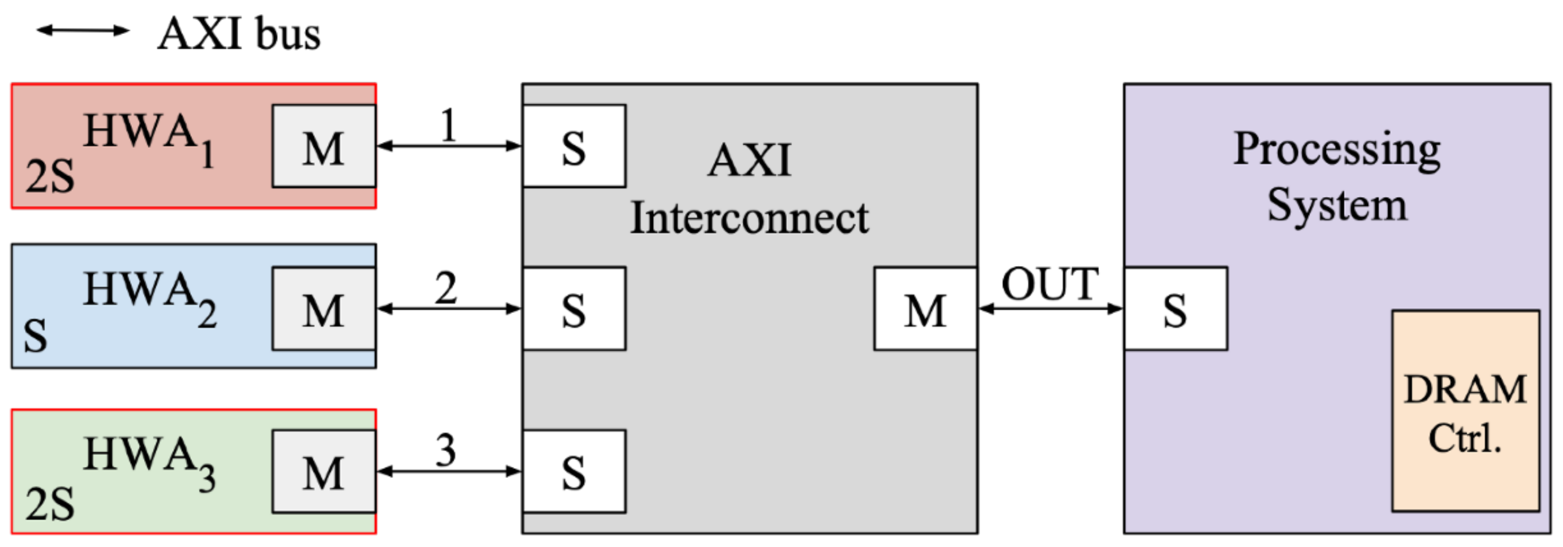
Xilinx PYNQ (ZYNQ 7000 SoC)

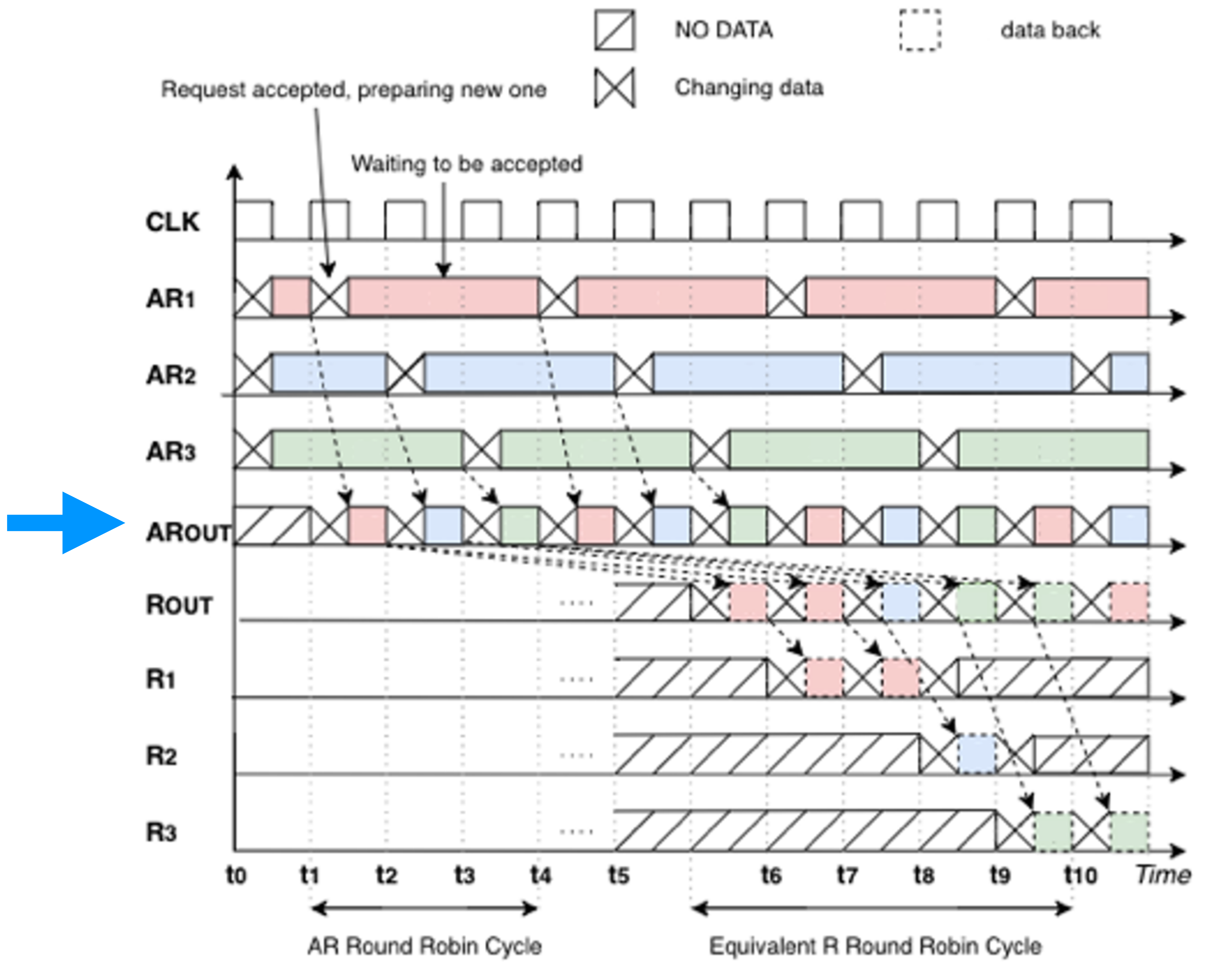
DEMO

Xilinx ZCU102 (Ultrascale+ MPSoC)

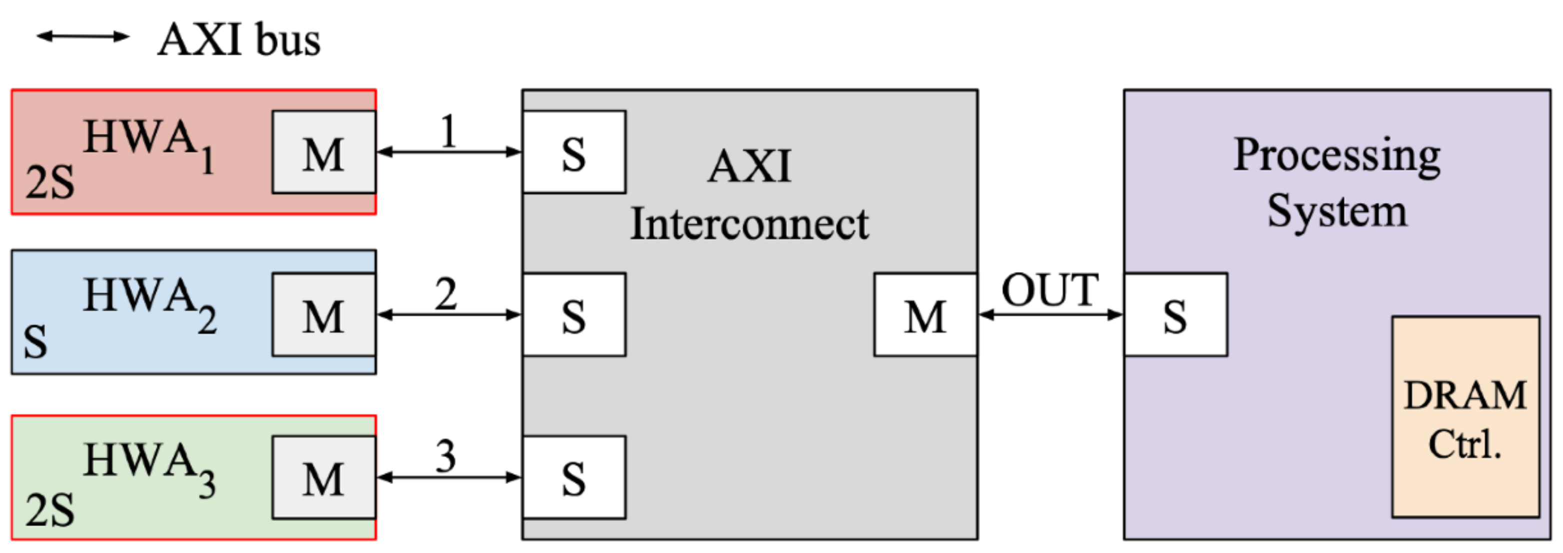
Analysis - Simplified diagram

Interconnect serves the HAs following a round-robin schema

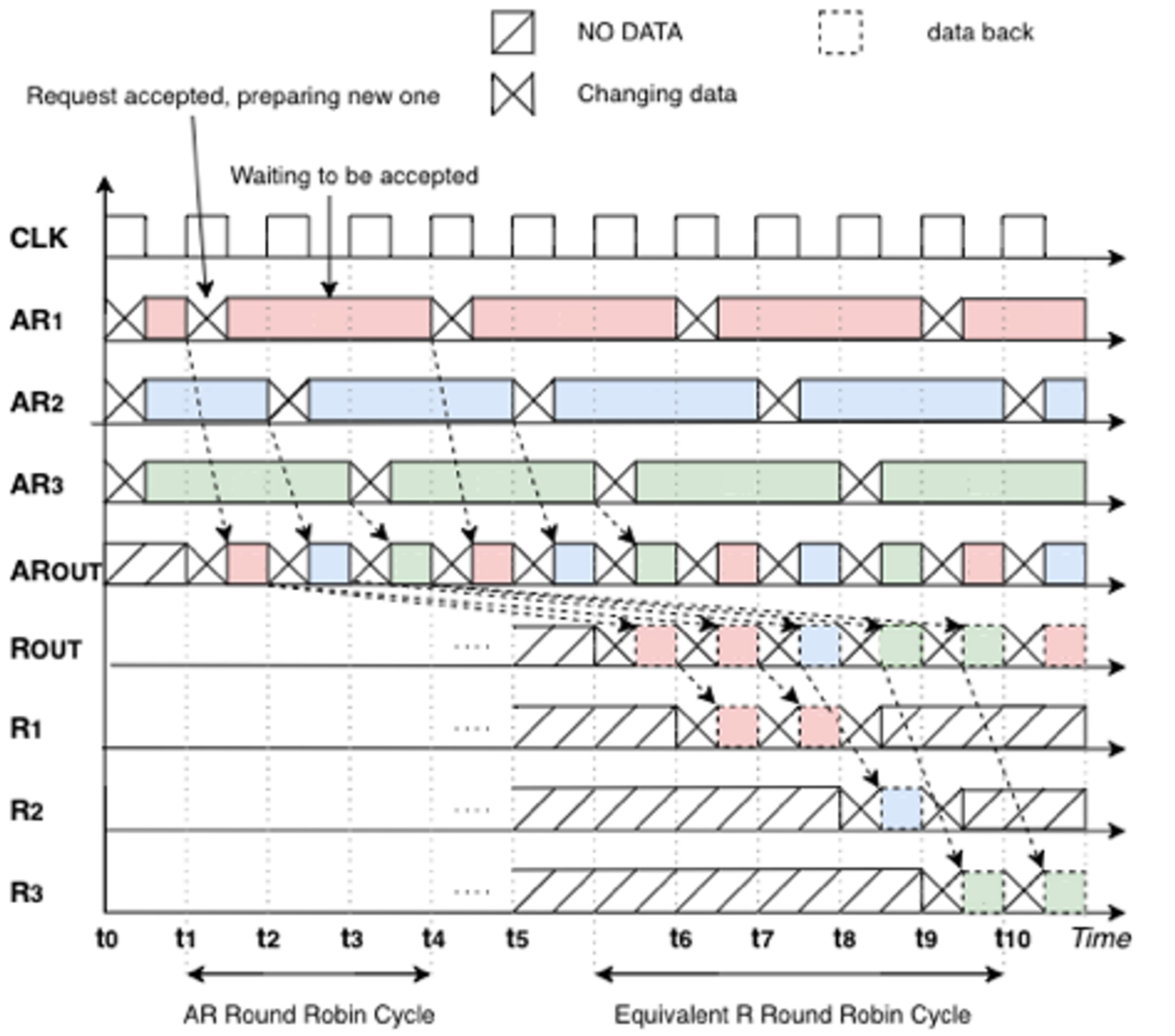




Analysis - Simplified diagram



Interconnect serves the HAs following a round-robin schema



The granularity on data depends on the bus structure of the transactions

Such structure is decided by the HA!

This allows a HA to affect the bandwidth assigned to another HA!

Impact on assigned bandwidth

Proposed a simple mathematical model (paper)

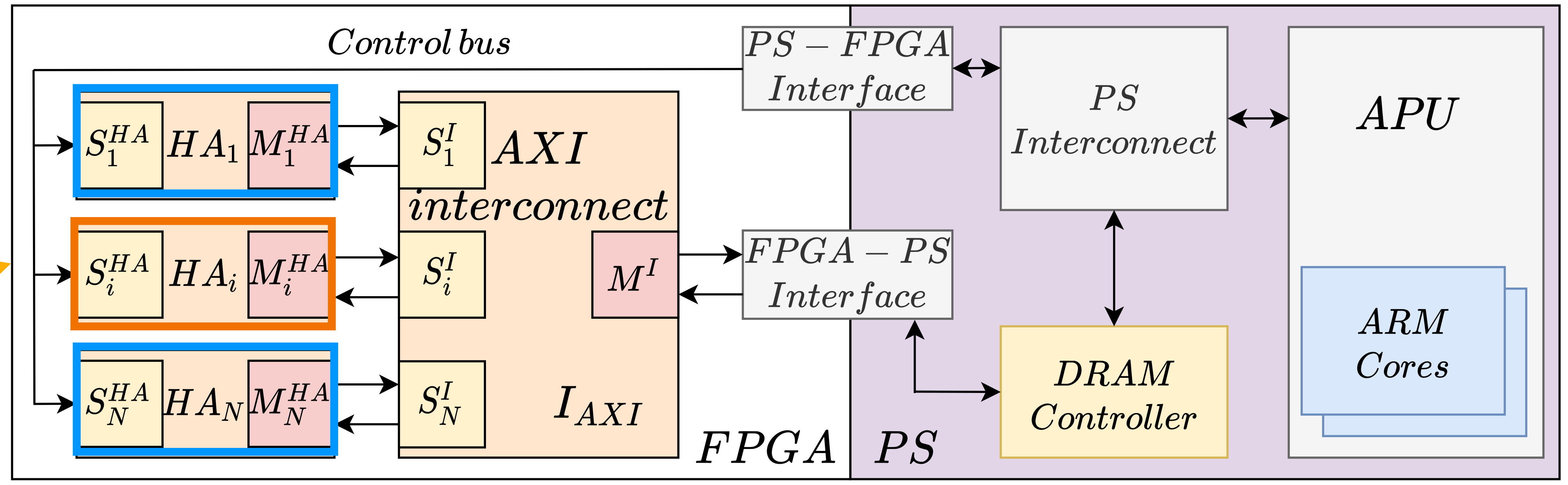
Example: burst length of the HA under analysis set to 16 words

Burst length of interfering modules

Interfering accel.		16	32	64	128	256
Num.	1	50.0%	33.34%	20.0%	11.11%	5.88%
	2	33.4%	20.0%	11.11%	5.88%	3.03%
	3	25.0%	14.29%	7.69%	4.0%	2.04%
	4	20.0%	11.11%	5.88%	3.03%	1.54%
	5	16.67%	9.09%	4.76%	2.44%	1.24%
	6	14.29%	7.69%	4.00%	2.04%	1.03%
	7	12.50%	6.67%	3.45%	1.75%	0.89%

Number of interfering modules

Bandwidth associated with the HA under analysis

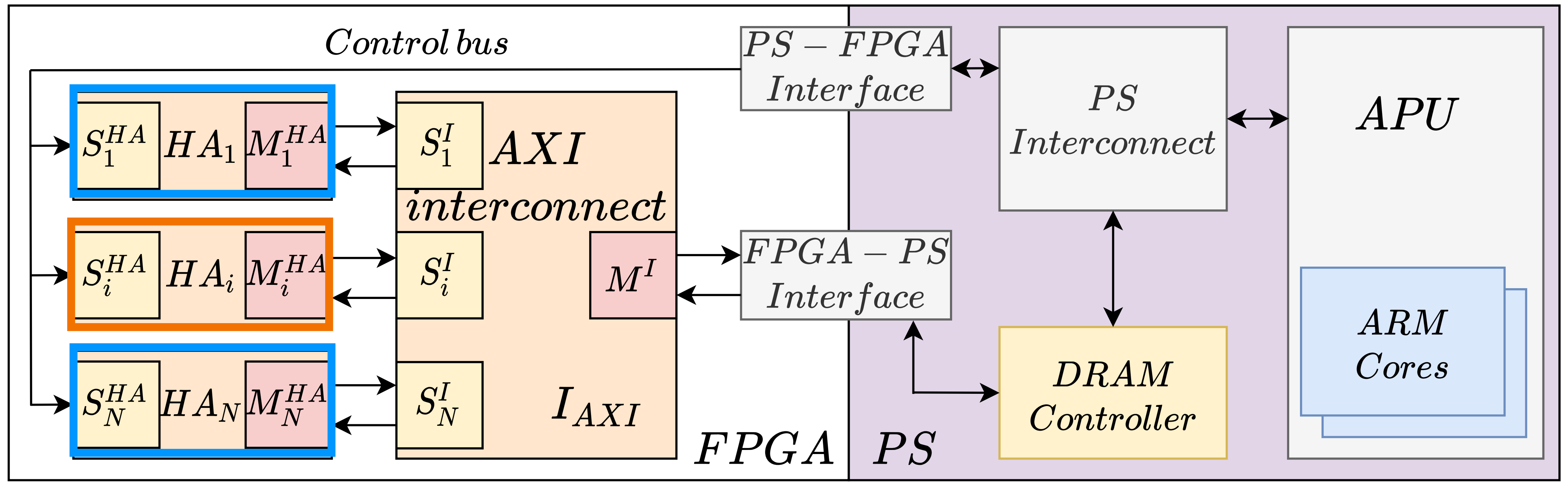


Impact on assigned bandwidth

Burst length of interfering modules

Interfering accel.		16	32	64	128	256
Num.	1	50.0%	33.34%	20.0%	11.11%	5.88%
	2	33.4%	20.0%	11.11%	5.88%	3.03%
	3	25.0%	14.29%	7.69%	4.0%	2.04%
	4	20.0%	11.11%	5.88%	3.03%	1.54%
	5	16.67%	9.09%	4.76%	2.44%	1.24%
	6	14.29%	7.69%	4.00%	2.04%	1.03%
	7	12.50%	6.67%	3.45%	1.75%	0.89%

Number of interfering modules



88% drop with respect to the expected bandwidth

Fair bandwidth distribution

Should we care about this issue?

How likely is that to happen?

Where does the controllers come from

Different sources

In-house development (expensive)

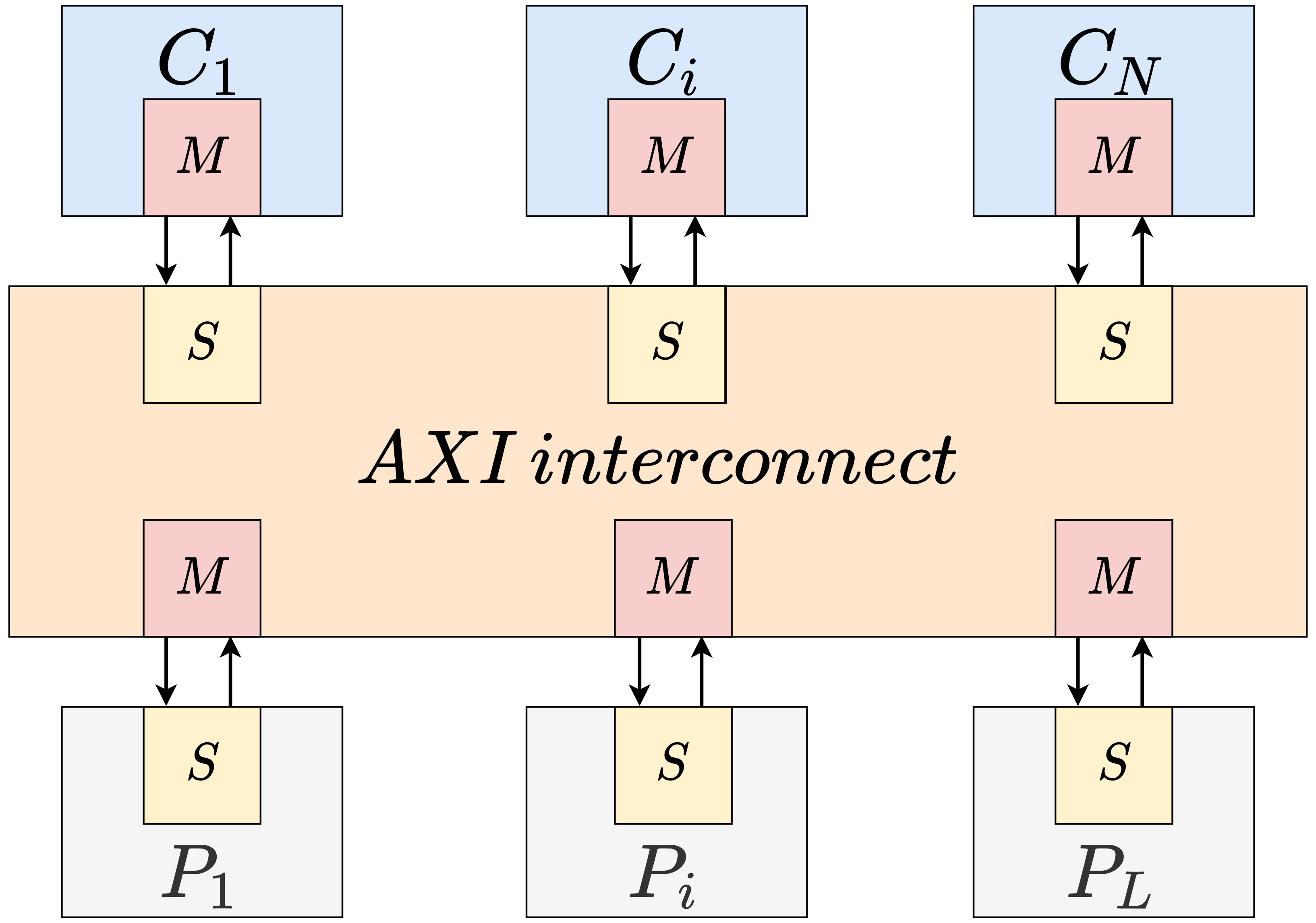
Third-party outsourced modules (popular)

Different development

Register Transfer Level (RTL) (Verilog, VHDL, SystemVerilog, etc.)

High-level synthesis (HLS) (Catapult, Vivado HLS, Intel HLS, etc.)

Hardware Construction Languages (Chisel)



Module integration challenges

Verification

Complexity of the IP modules

HLS-generated code

Encrypted third-party IP modules

Dependencies

Among multiple modules

Modules may be software configurable

Different versions of the standard - different allowable burst lengths

HLS compilers may choose by default the structure of the transactions

- Different from compiler to compiler
- Low-level detail that may be hidden (abstracted)

Summarizing - Lesson learned

Leaving the controllers defining the structure of their transactions may strongly affect the available bandwidth of other modules

Create circular dependencies among modules

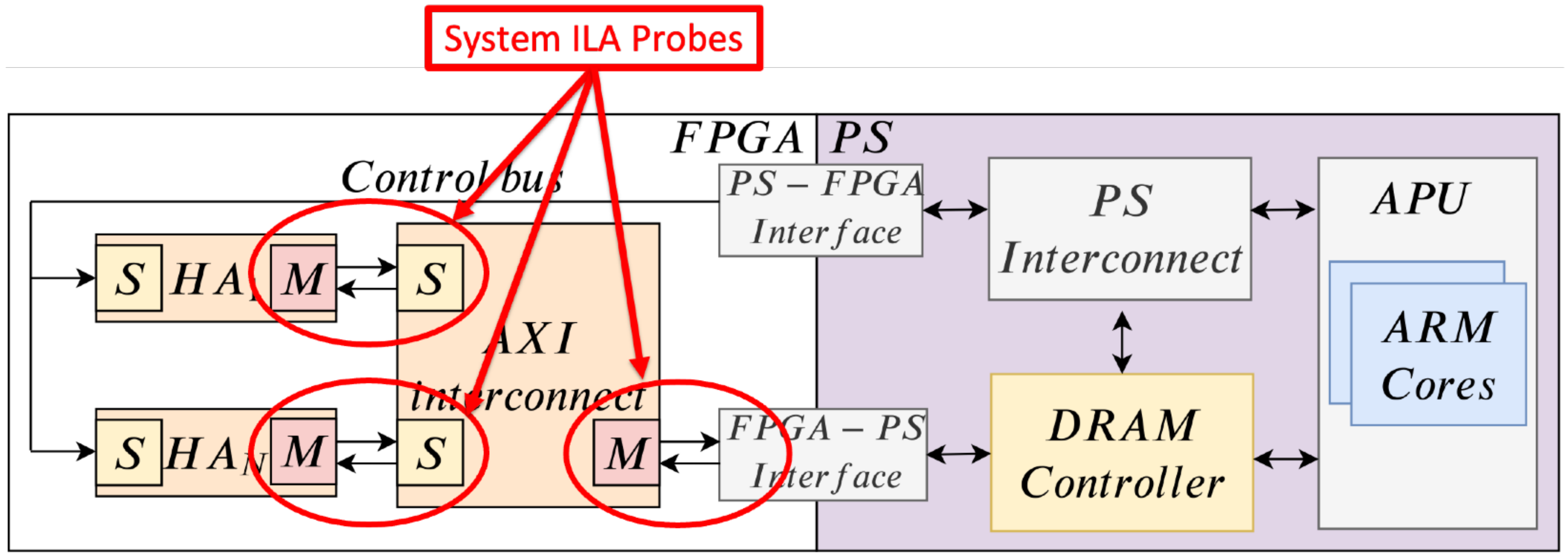
- Unexpected bandwidth distribution
- Affect the availability of shared resources

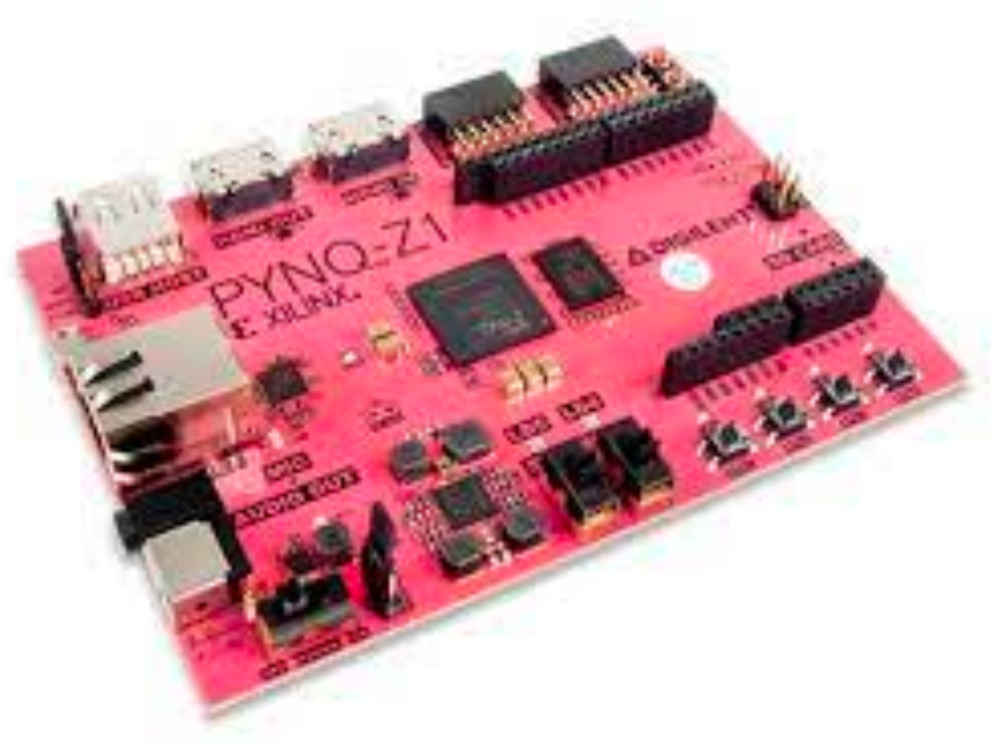
Proposed solution and more experimental results later in the presentation

On dependencies of write transactions

We developed our own AXI-compliant controller for cycle-accurate profiling

Found an interesting behavior during development



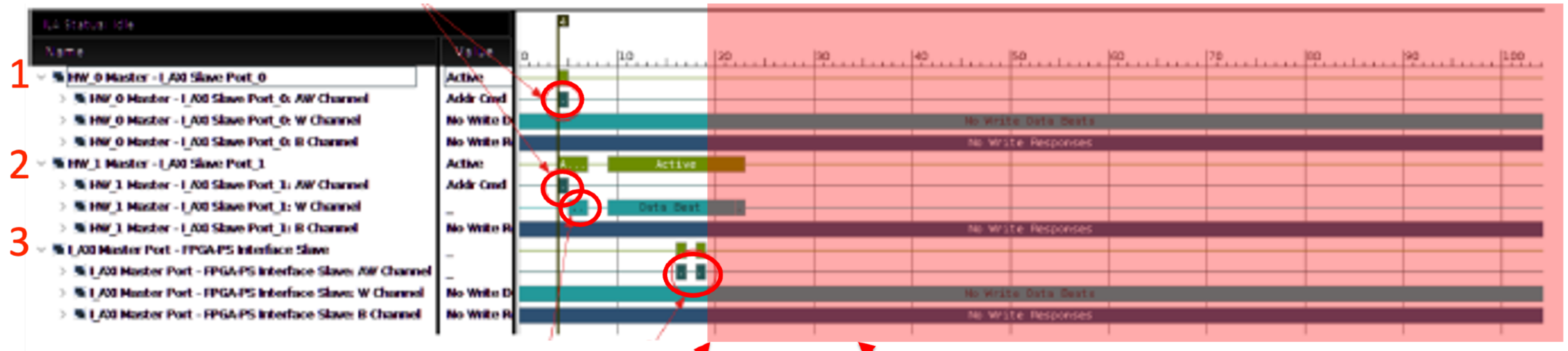


Xilinx PYNQ (ZYNQ 7000 SoC)

DEMO

Xilinx ZCU102 (Ultrascale+ MPSoC)

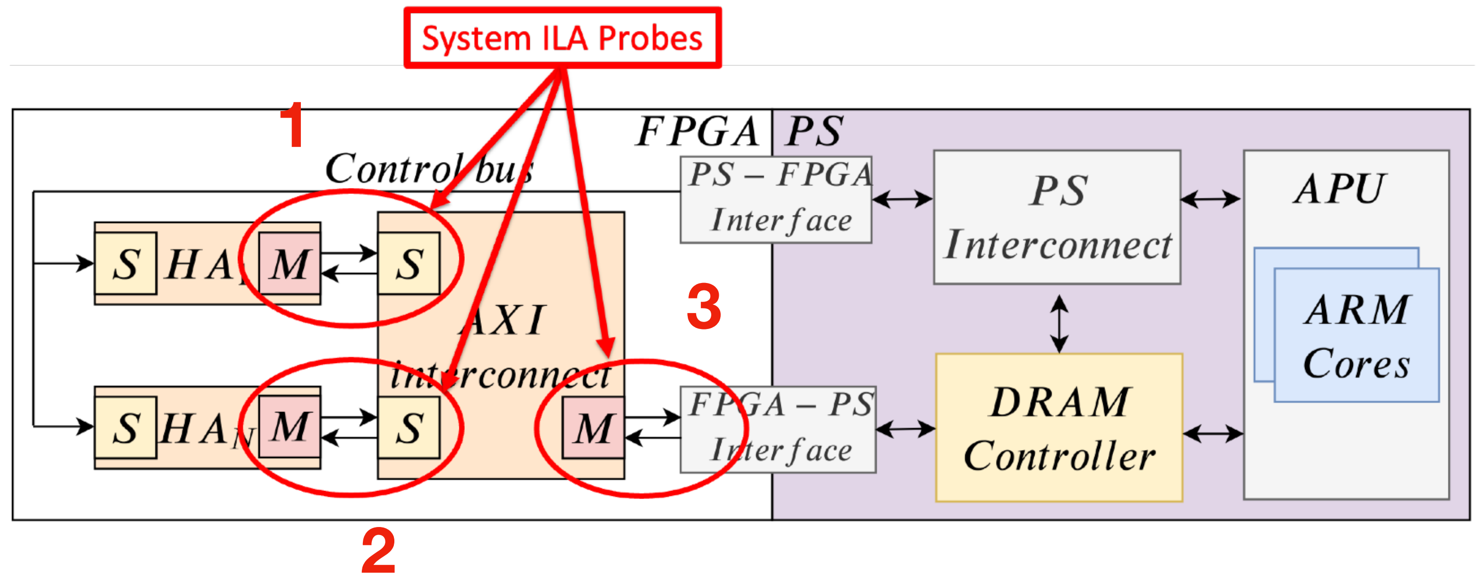
AXI bus stalls - analysis



Actual System ILA track recorded on a Zynq Ultrascale+ plaftform (ZCU102) using Xilinx Vivado 2018.2

Waiting for data

The Bus is stalled!



a) HA_1 and HA_2 send a request for transaction

b) The round-robin arbiter transmits the transaction of HA_1 first

c) HA_2 is ready to propagate data, but cannot access the shared bus because booked by HA_1

d) As long as HA_1 does not provide whole data words, the bus is stalled.

From AMBA® AXI and ACE Protocol Specification

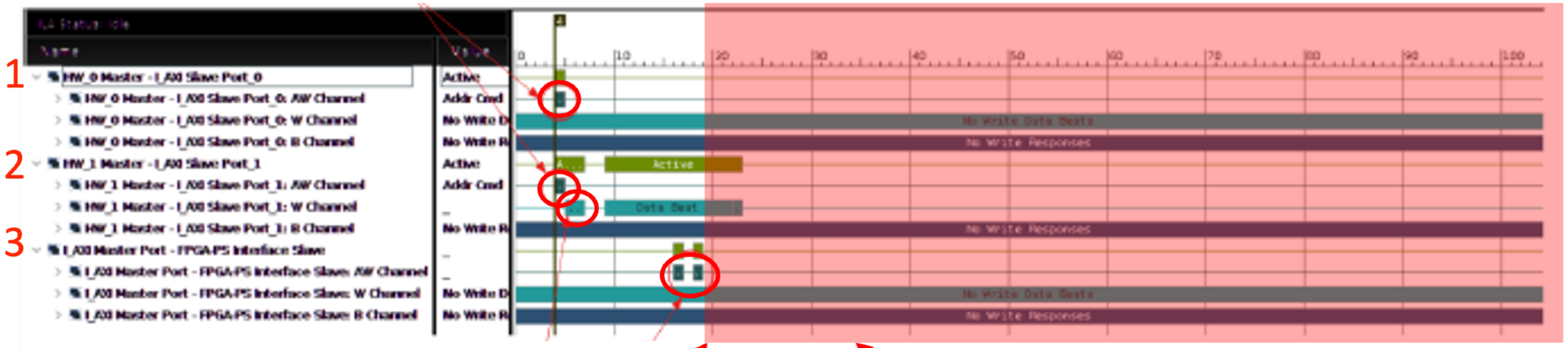
A5.2.2 Write data ordering

A Manager must issue write data in the same order that it issues the transaction addresses.

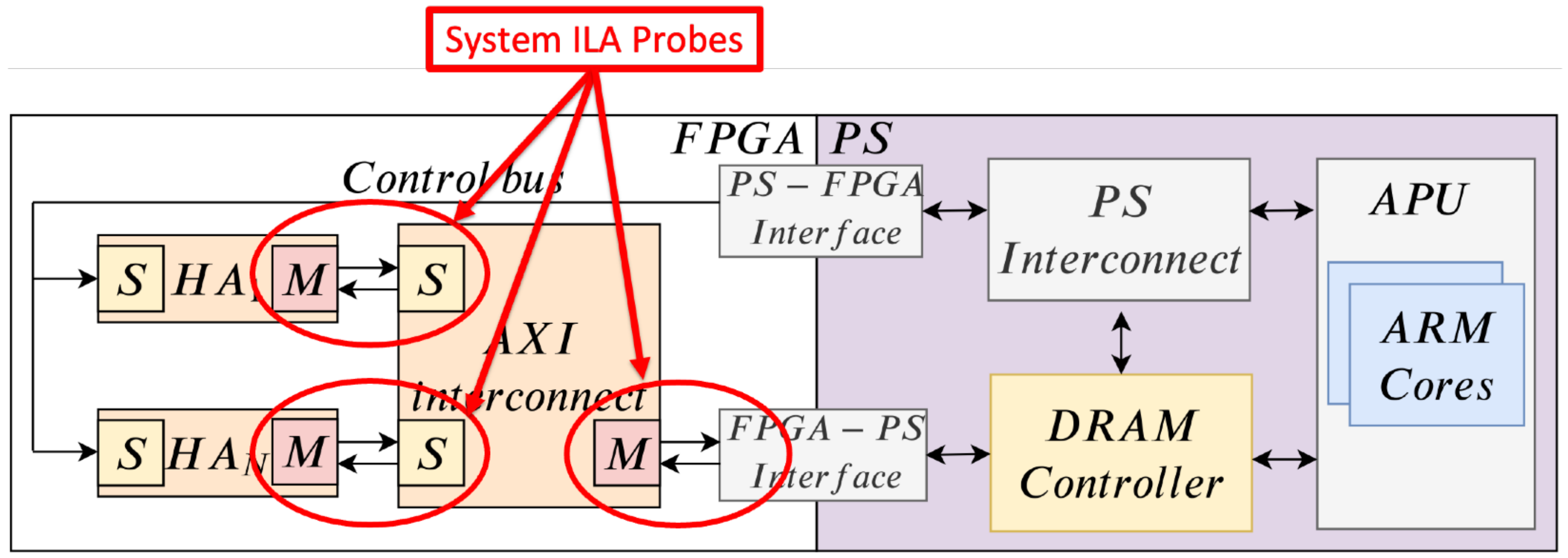
An interconnect that combines write transactions from different Managers must ensure that it forwards the write data in address order.

The interleaving of write data with different IDs was permitted in AXI3, but is deprecated in AXI4 and later. See the AMBA AXI and ACE Protocol Specification issue F specification for more details on write data interleaving.

AXI bus stalls - analysis



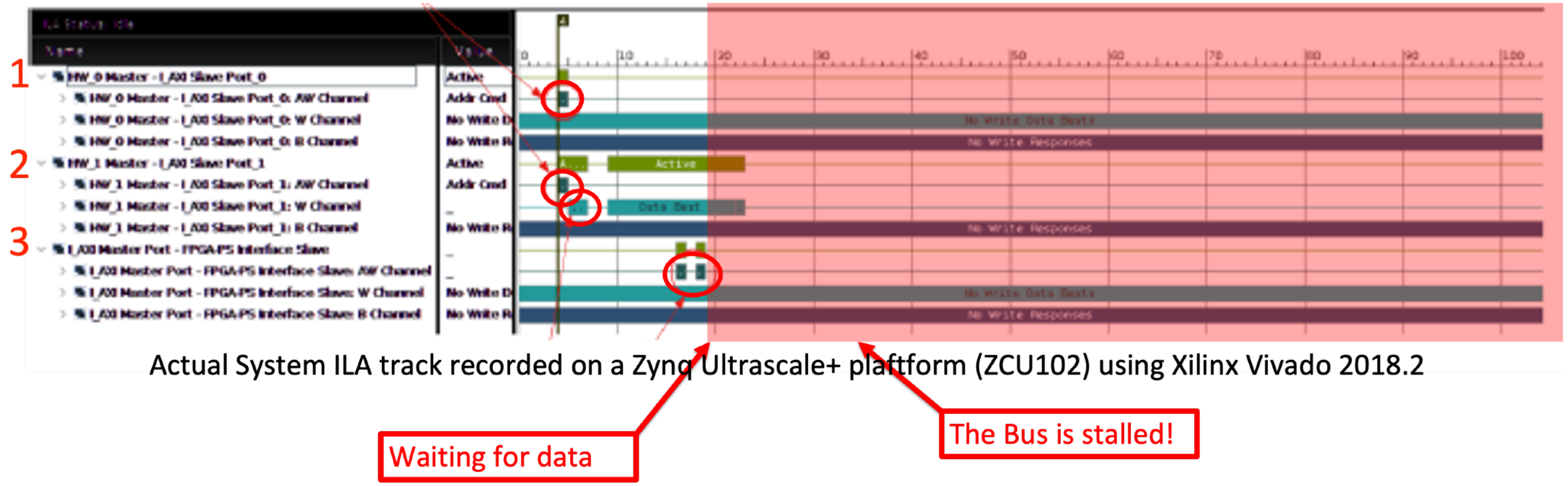
Actual System ILA track recorded on a Zynq Ultrascale+ plaftform (ZCU102) using Xilinx Vivado 2018.2



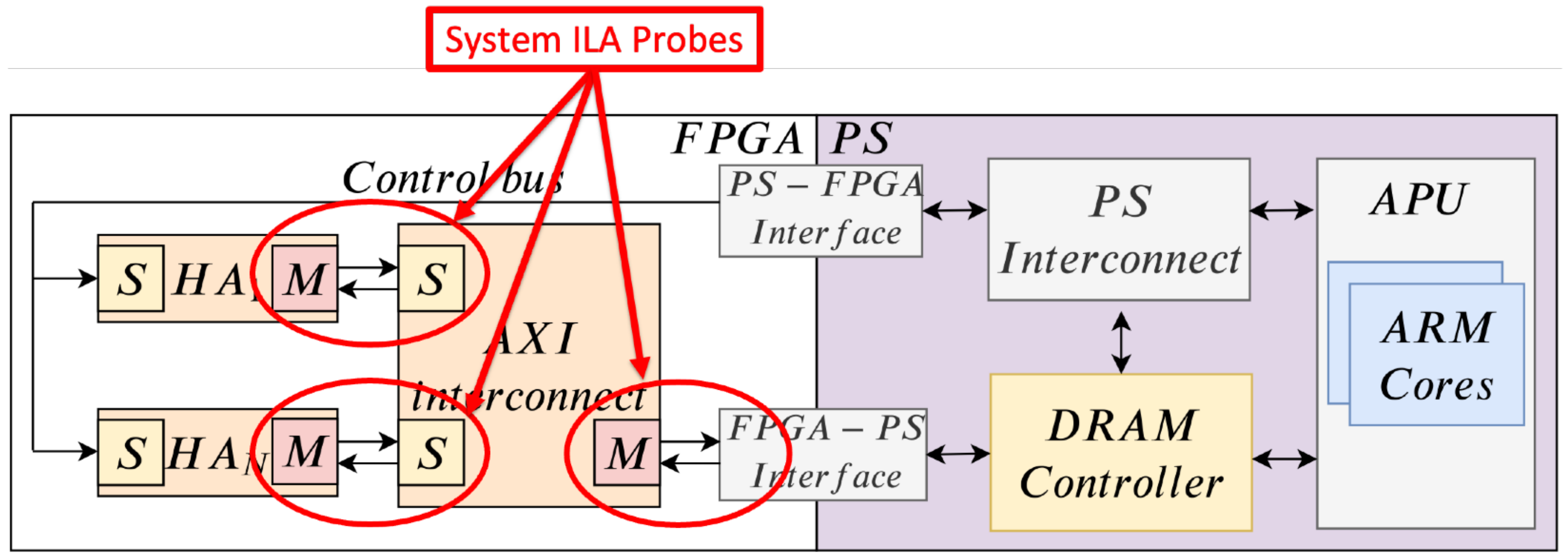
When the transaction generated by HA_0 is propagated, the write data channel in the shared output bus is assigned to HA_0

The interconnect is trusting HA_0 that it will fulfil the transaction and leave the bus to others as soon as possible

AXI bus stalls - analysis



Actual System ILA track recorded on a Zynq Ultrascale+ plaftform (ZCU102) using Xilinx Vivado 2018.2



Delaying its data provisioning, a **single** module can **deny** the access to **all of the peripherals** from the other controllers

The protocol is not broken! No maximum delay is defined in the standard

Consequences

As long as a transaction is kept pending by an HA, no other controllers can write data

The timings of the other HAs are affected

The availability of the shared resources is compromised

The network is left in an **inconsistent** state - a **system reset** may be required

A single controller can exploit this lack of specification to denial the access to the shared resources from all of the other controllers

Should we care about this issue?

How likely is that to happen?

Potential source of bus stalls

Malicious behaviour

Misbehaviour/Fault

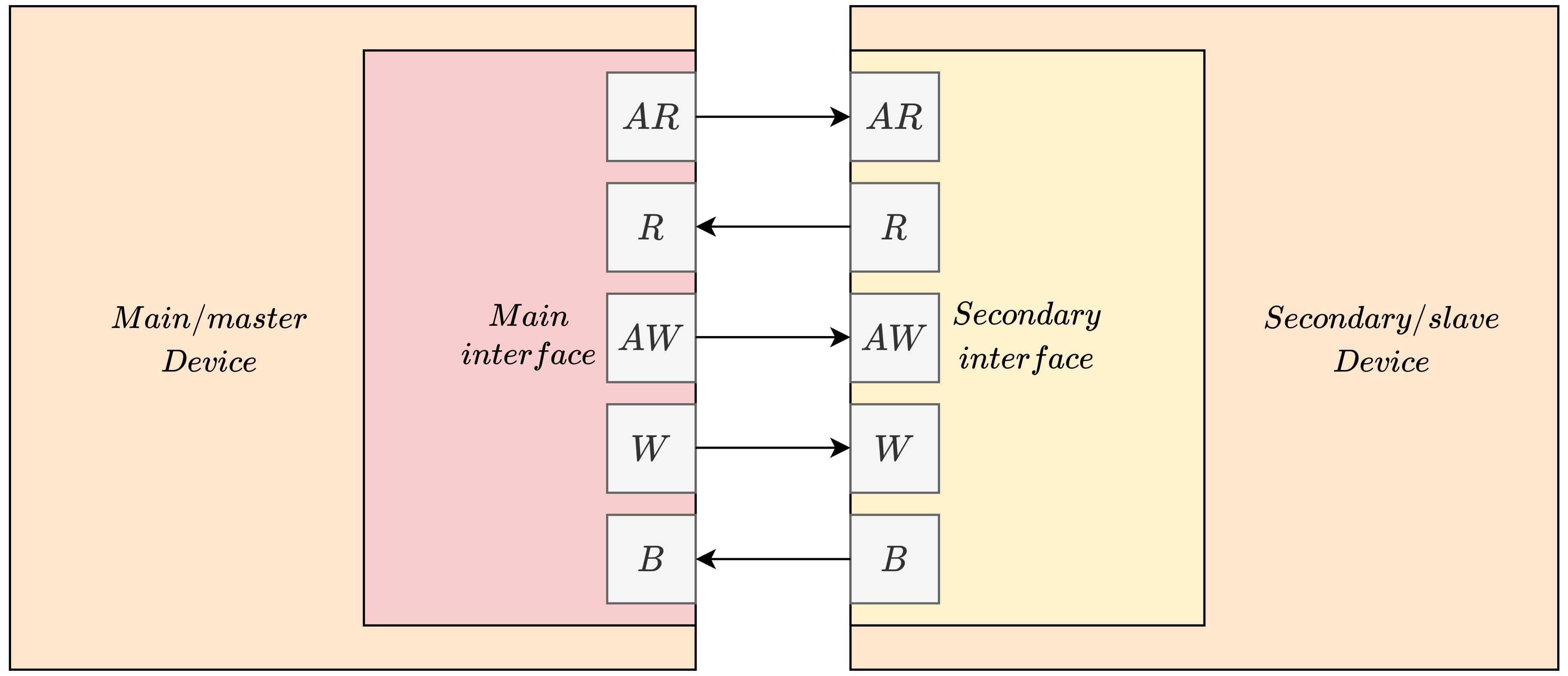
Bugs in development/ testing

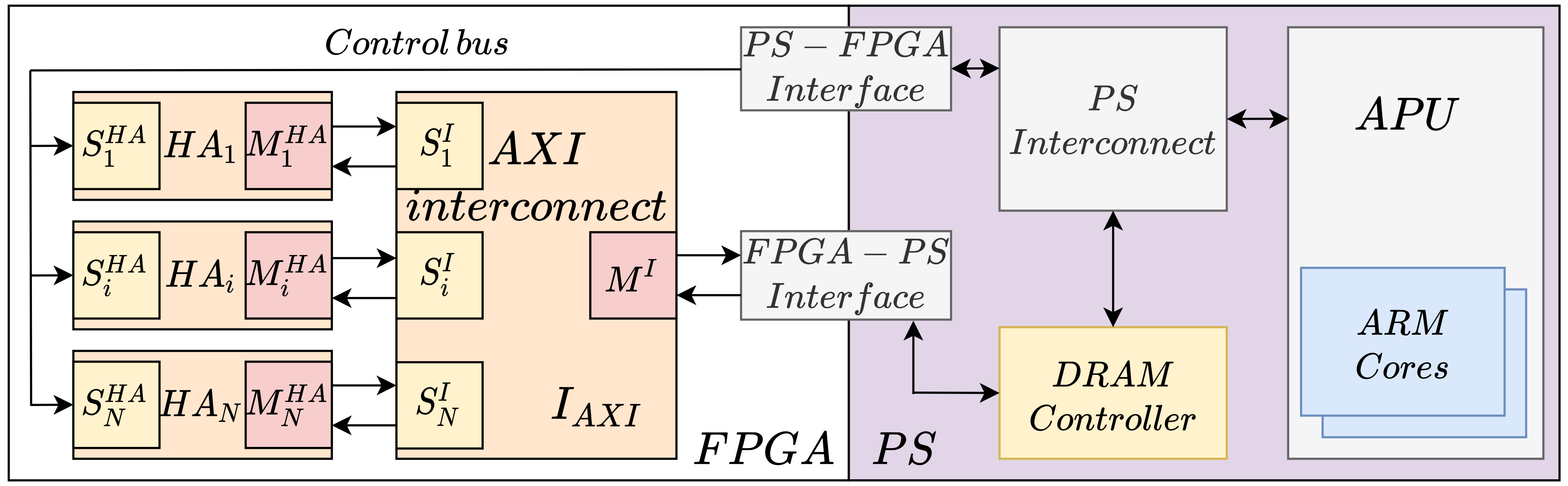
Optimizations

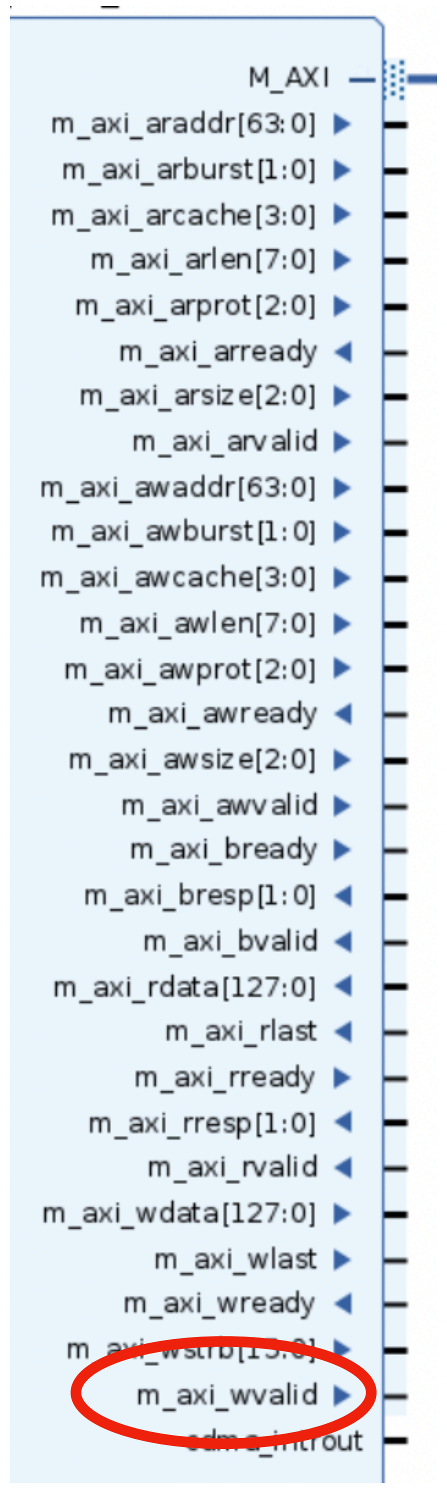
Speculative bus access

Delays in data provisioning

How this lack of specification can be exploited







Malicious/misbehaving hardware module

Directly acting on the valid line of the W channel (wvalid)

Delaying the write data production after a write request is issued

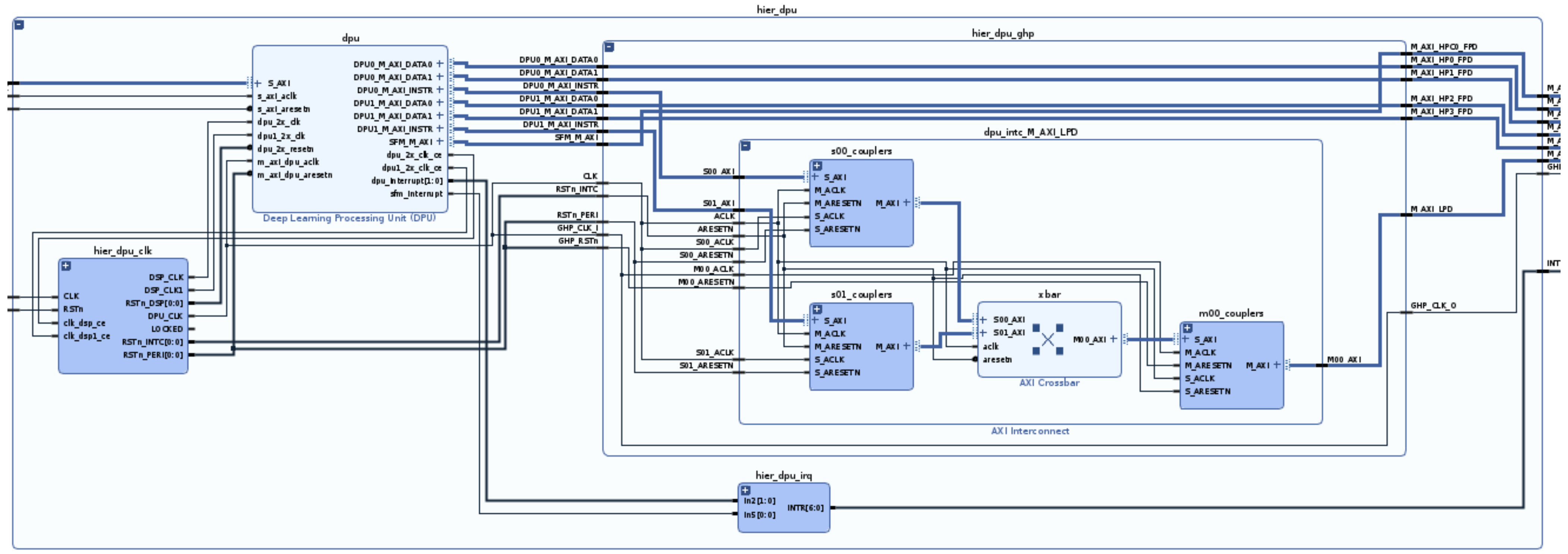
Delaying data read operation (in speculative-access modules)

Example of speculative bus access

Xilinx Deep-Learning Processing Unit (DPU)

Most recent DNN hardware accelerator proposed by Xilinx

Part of the Xilinx Vitis AI framework



Credits: xilinx.com

Xilinx ZCU102 (Ultrascale+ MPSoC)

We customized the Vitis AI hardware design integrating a System ILA to analyze the DPU execution (+ other custom profilers)

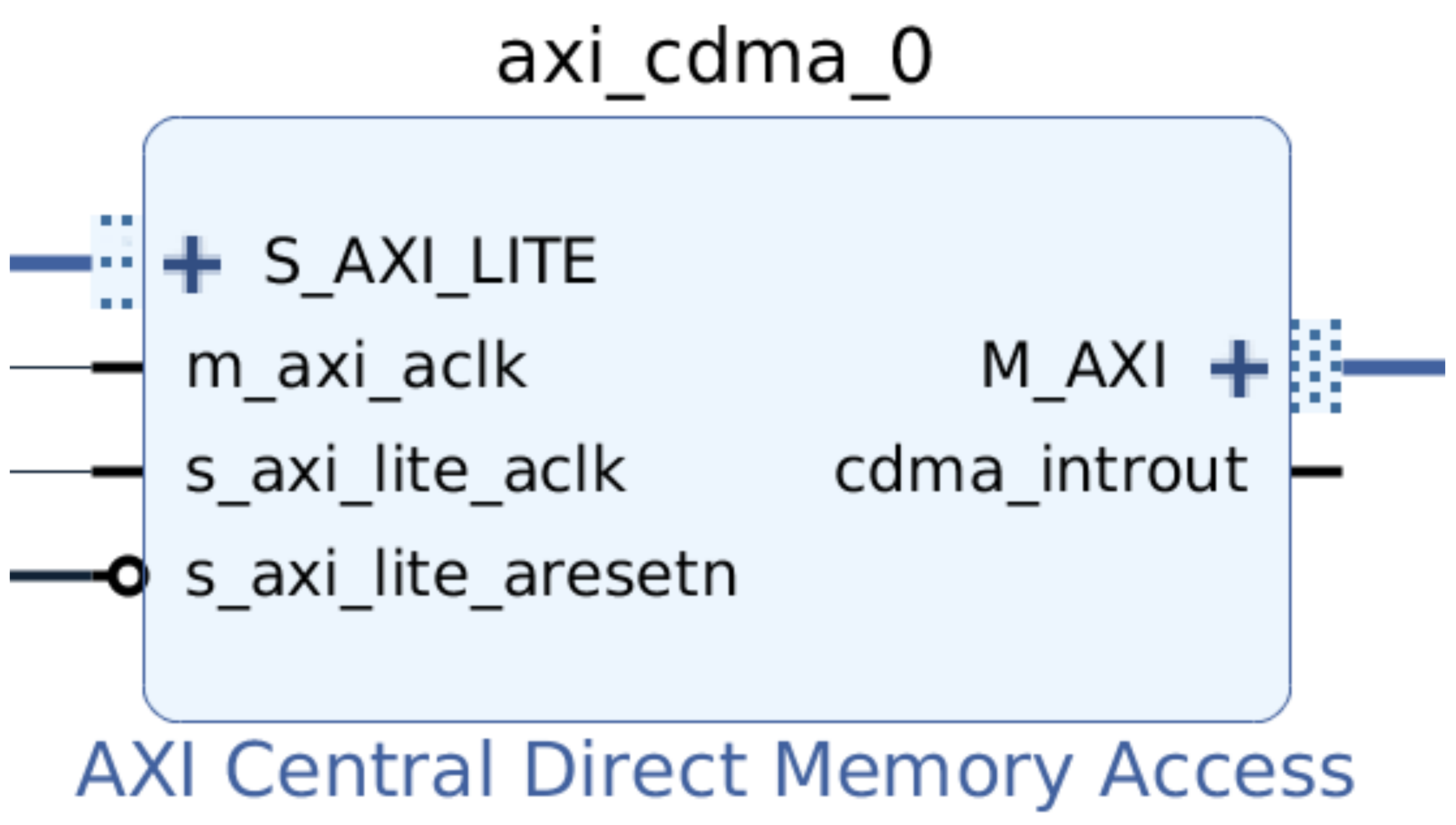
DPU execution hardware track

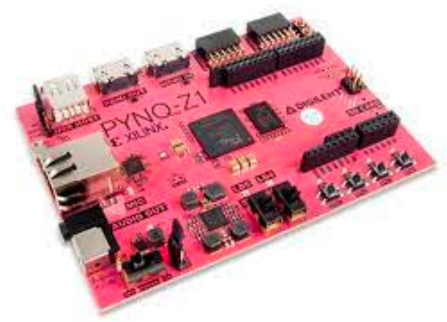
Write channel booked by the DPU but no data are propagated

Another example of speculative bus access

Xilinx Central Direct Memory Access (CDMA)

Direct Memory Access between a source buffer and a destination buffer

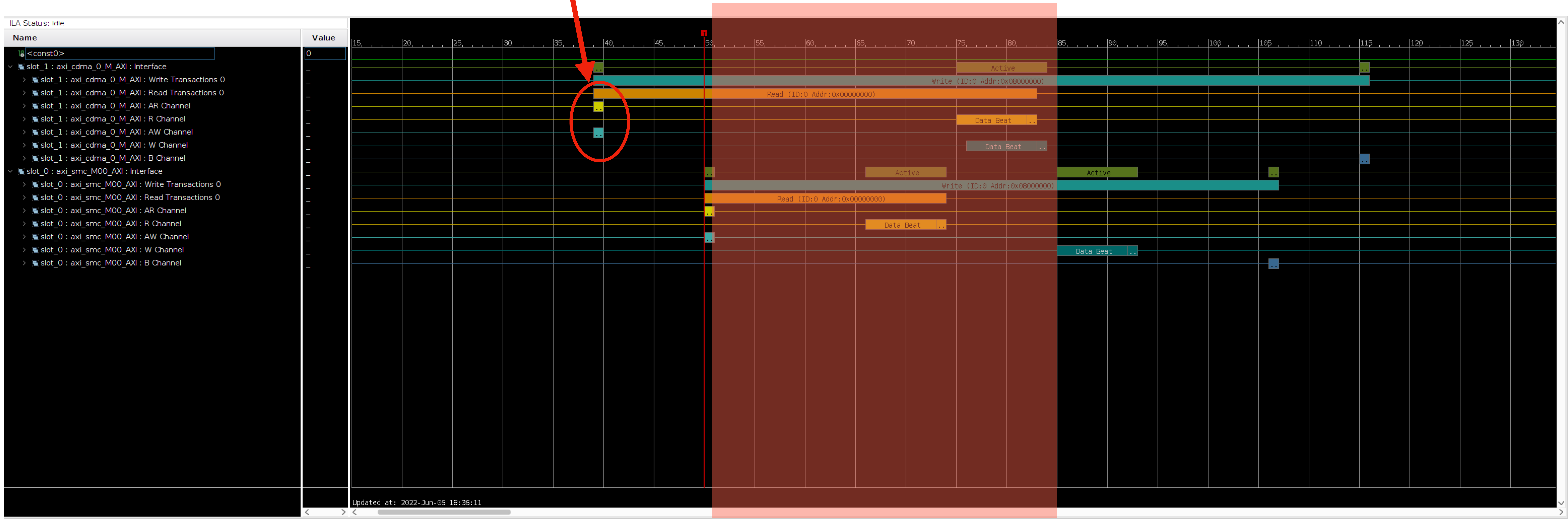




Created a hardware design to monitor the CDMA execution

CDMA execution hardware track

The write request is issued before receiving the data to be written



The shared write bus is booked (stalled) and cannot be used by other controllers

Stall on write depends on the delay for receiving the read data

The impact

Security

Endanger the availability of shared resources

Exploitable for denial-of-service attacks of shared resources

Safety

Create circular dependencies among controllers

Broken isolation among controllers

Average performance

Lower than expected

Waste cycles on data channel

Test on a realistic mixed-critical scenario

Target platform: Xilinx Ultrascale+ MPSoC

Inspired by common functionalities required in modern autonomous vehicles

High-critical:

Deep learning hardware acceleration (CHaiDNN)

Critical sensor/actuation (real-time constraints)

Low-critical:

Generic data mover (possibly injecting stalls)

Xilinx ZCU102 (Ultrascale+ MPSoC)

Xilinx/**CHaiDNN**

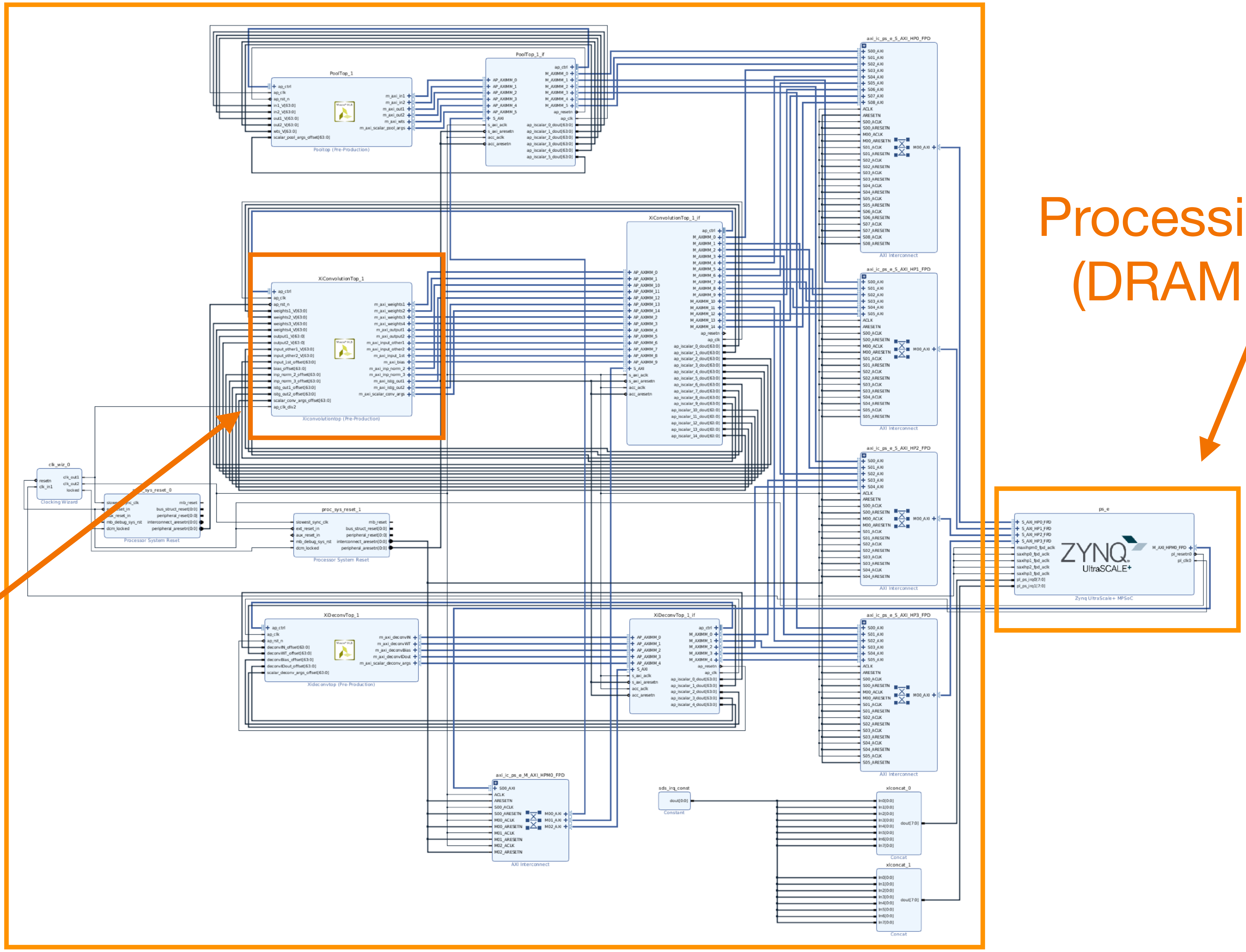
HLS based Deep Neural Network Accelerator Library for Xilinx Ultrascale+ MPSoCs

9 Contributors · 58 Issues · 280 Stars · 147 Forks

The CHaiDNN hardware accelerator

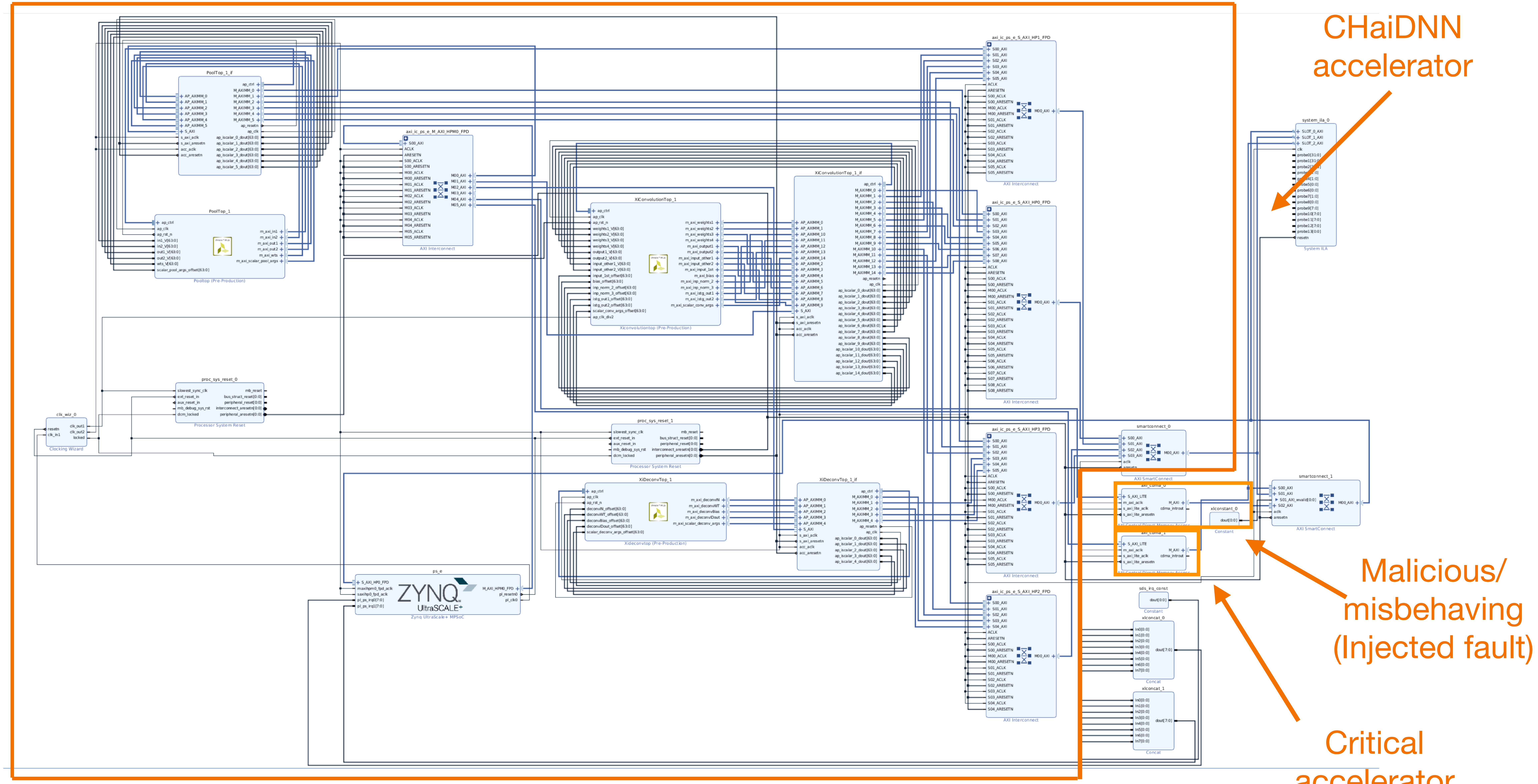
CHaiDNN hardware accelerator

Processing System (DRAM memory)



Convolutional accelerator

The mixed-critical architecture



Nominal behaviour

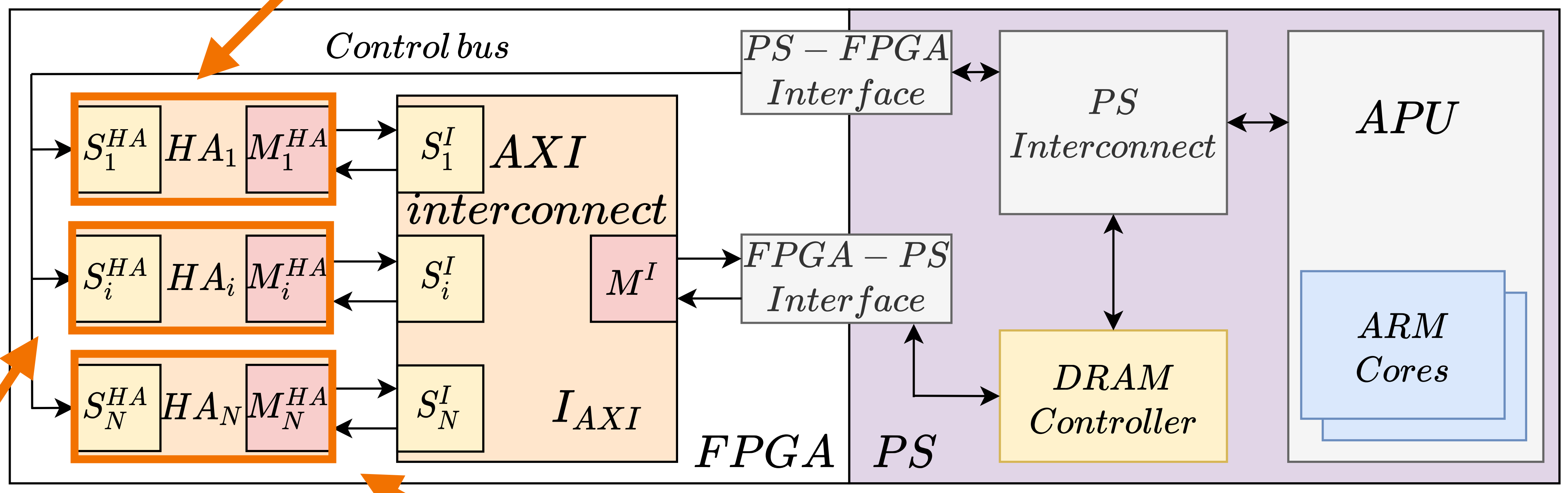
PS: Xilinx CHaiDNN stock Petalinux

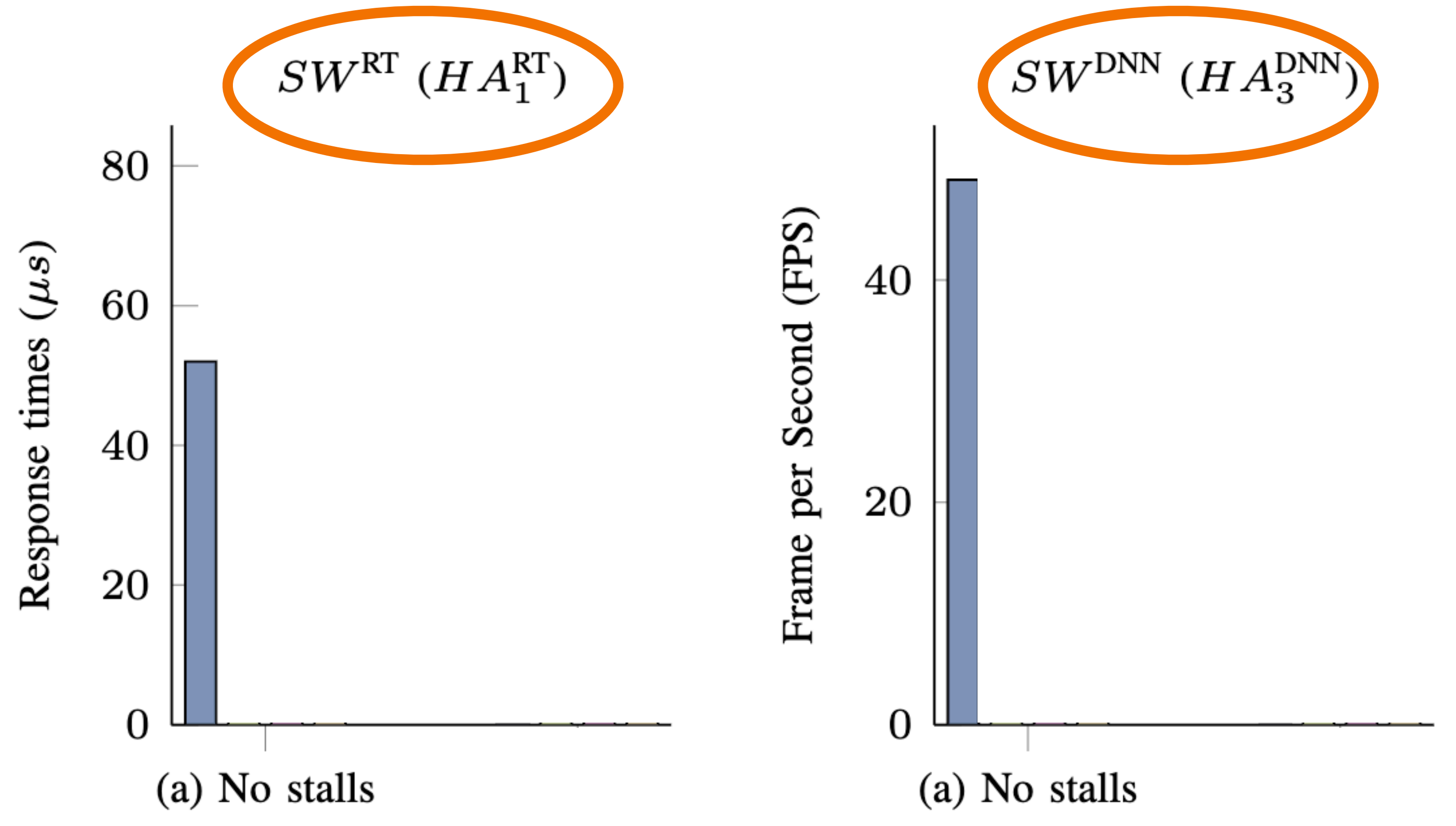
FPGA: mixed-critical design (CHaiDNN + critical module + misbehaving)

Accelerators leveraged by SW-tasks running on Linux

Requirements:

CHaiDNN (SW^{DNN}): minimum FPS
Critical module (SW^{RT}): execute before deadline





Good execution

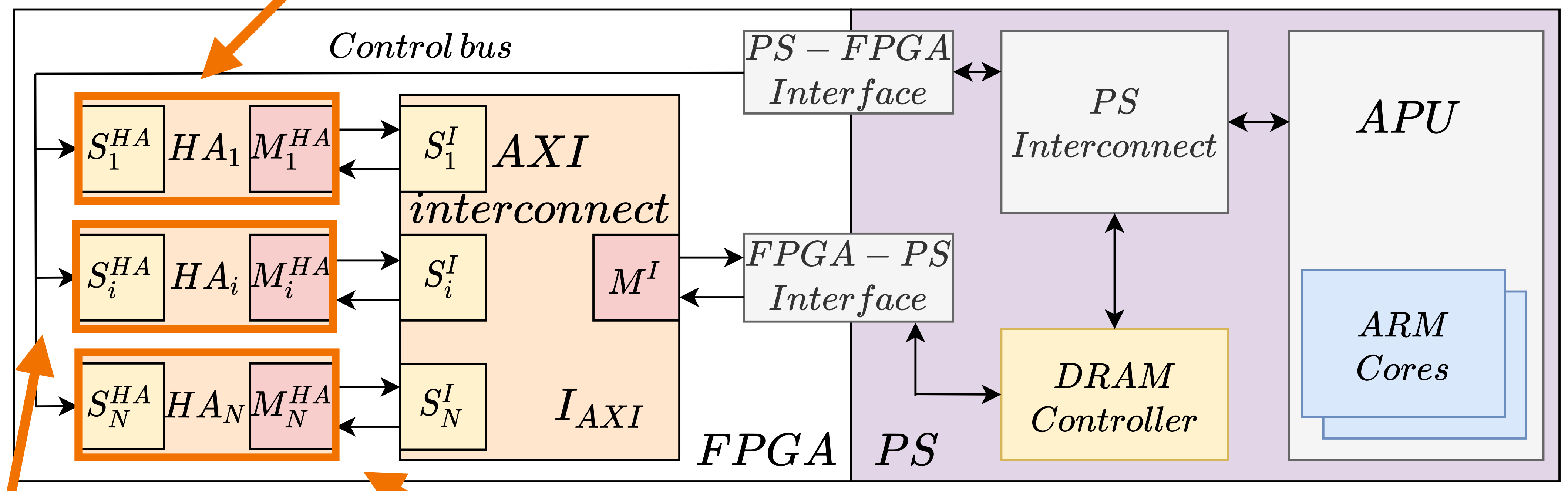
Injecting AXI stalls

SW-task running in PS requests a data movement from HA_2

The stall introduced by HA_2 denies memory access from the CHaiDNN and the critical HAs

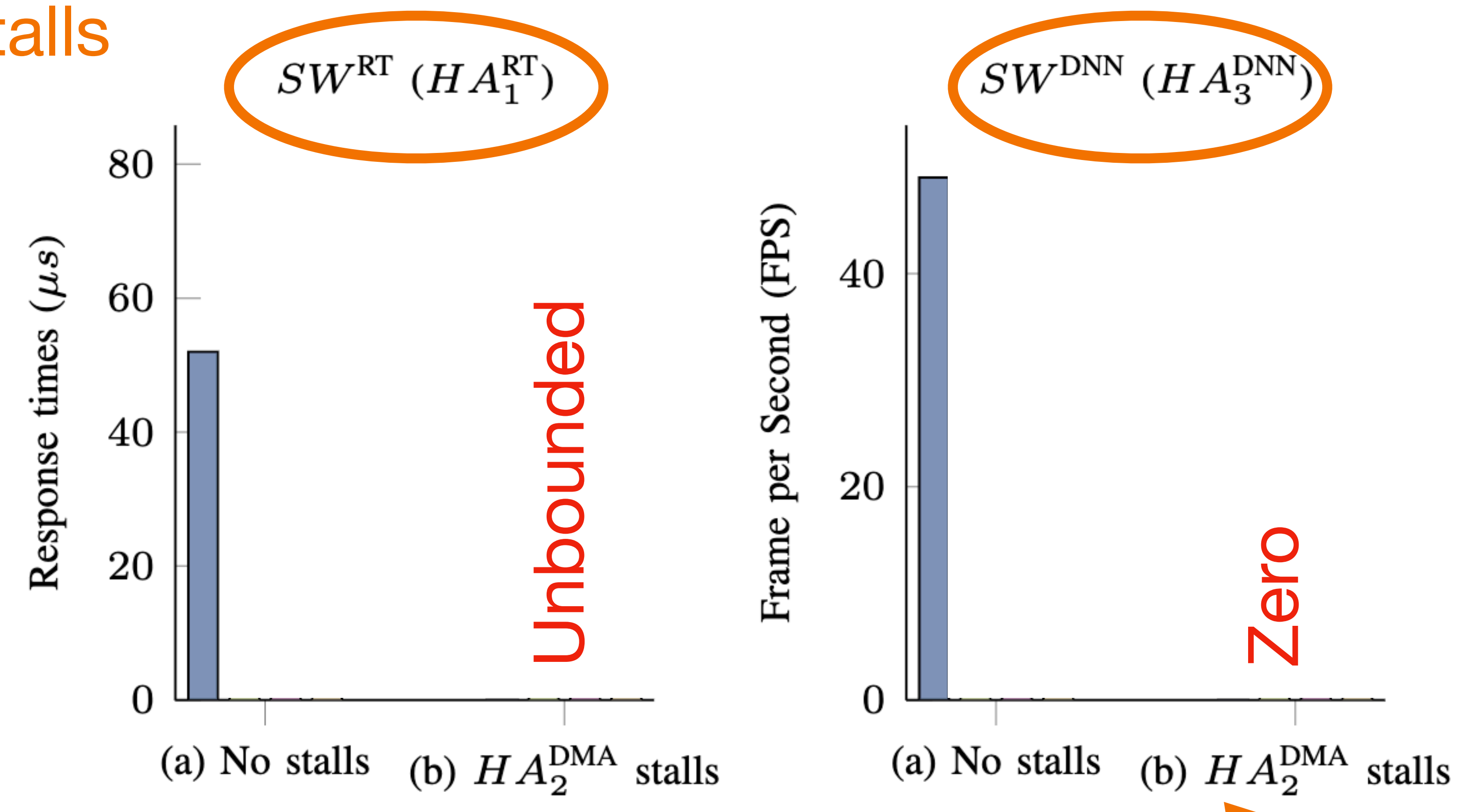
Any timing performance requirement is broken a-priori

Critical accelerator



Malicious/ misbehaving injecting stalls

CHaiDNN hardware accelerator



HA_2 inject stalls

Broken execution

Summarizing criticalities

1. **The protocol is not broken!** Can be difficult to detect in a superficial functional verification

2. No **default recovery** mechanisms provided - AXI transactions cannot be aborted

3. No maximum time for stall defined - can be **unbounded**

4. **Circular dependencies** among modules - isolation broken

Summarizing - Lesson learned

Leaving the controllers the freedom of delaying their data provisioning can affect the availability of the shared resources

…this can be exploited to introduce a denial-of-service of shared resources!

Proposed solution and more experimental results later in the presentation

Proposed solutions

Solving unfair bandwidth distribution

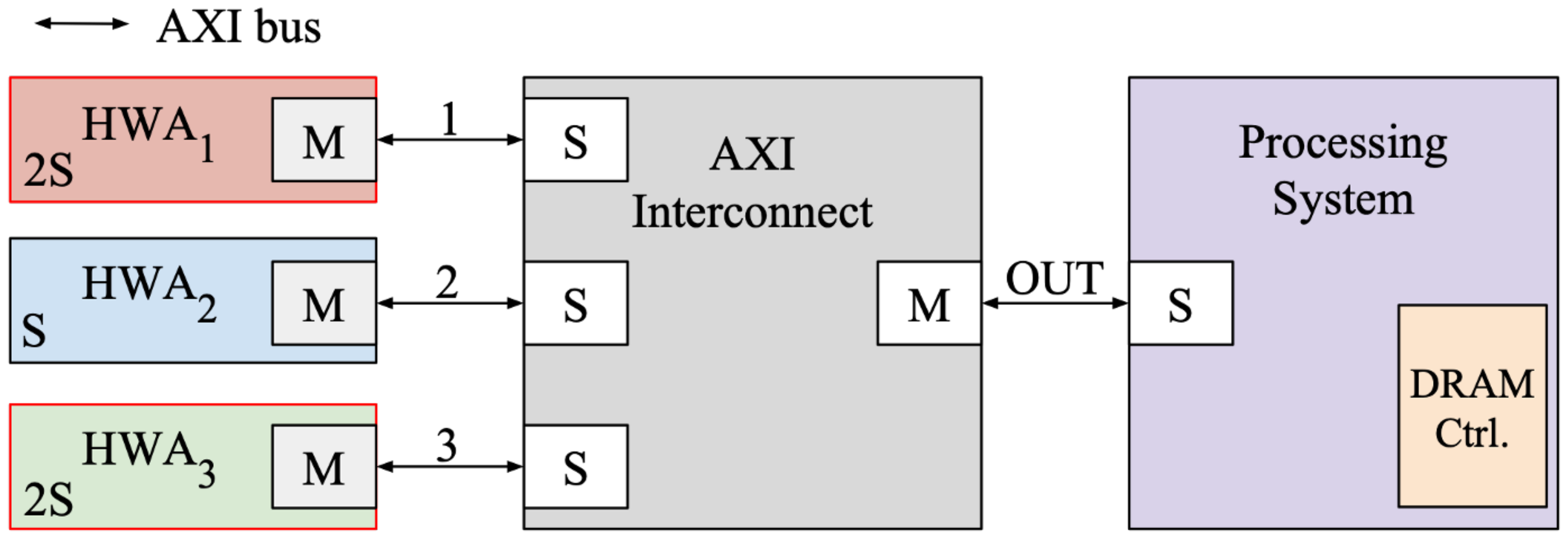
Source of the issue

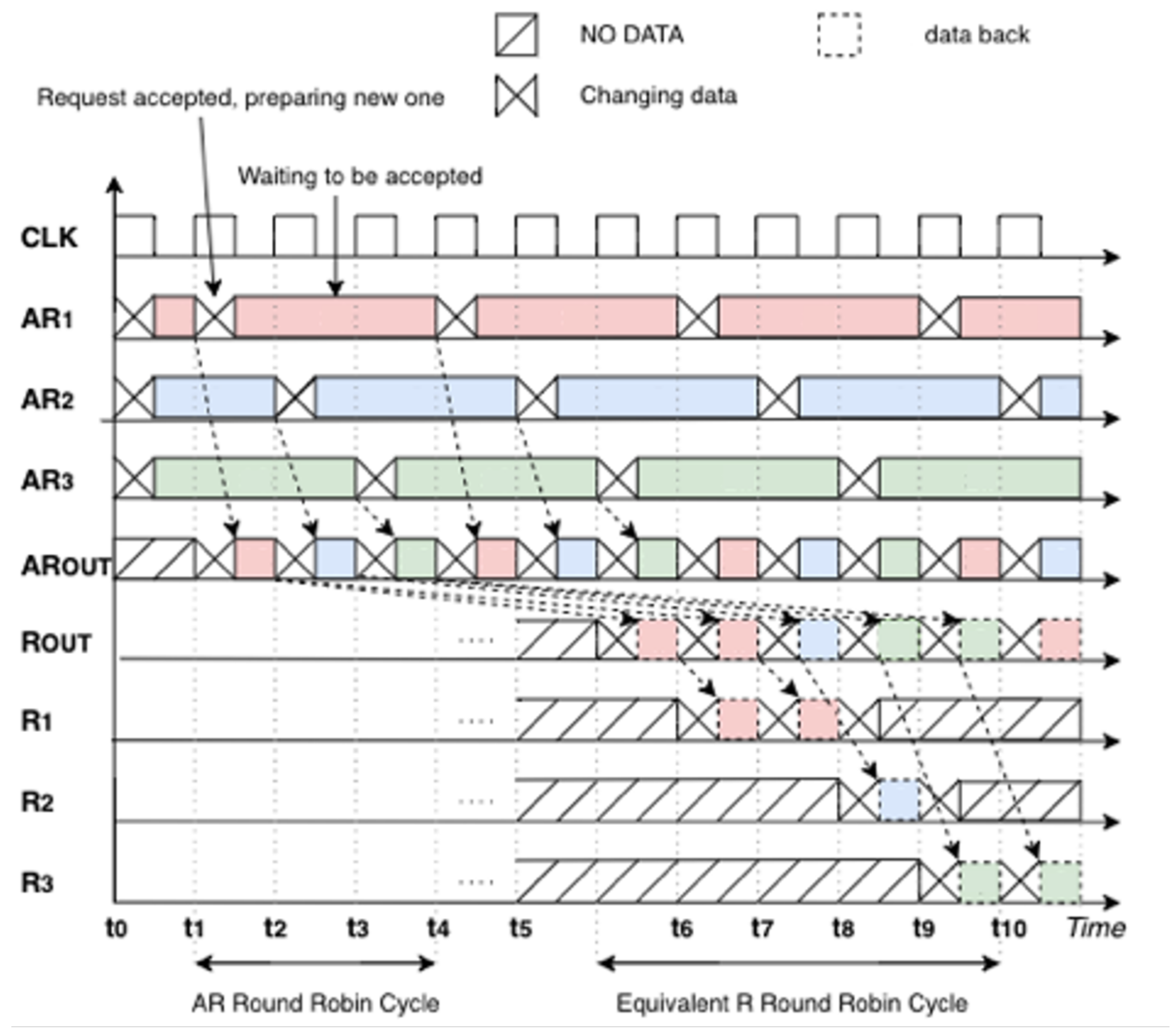
Data structure of transactions is **left to be decided** by the controller itself

This can even change **dynamically** during execution

Need a solution to **equalize** the structure of the transactions issued by the HAs

Cannot just stick a constant to burst length signal





The AXI burst equalizer (ABE)

Essential module to be placed between controllers and the interconnect

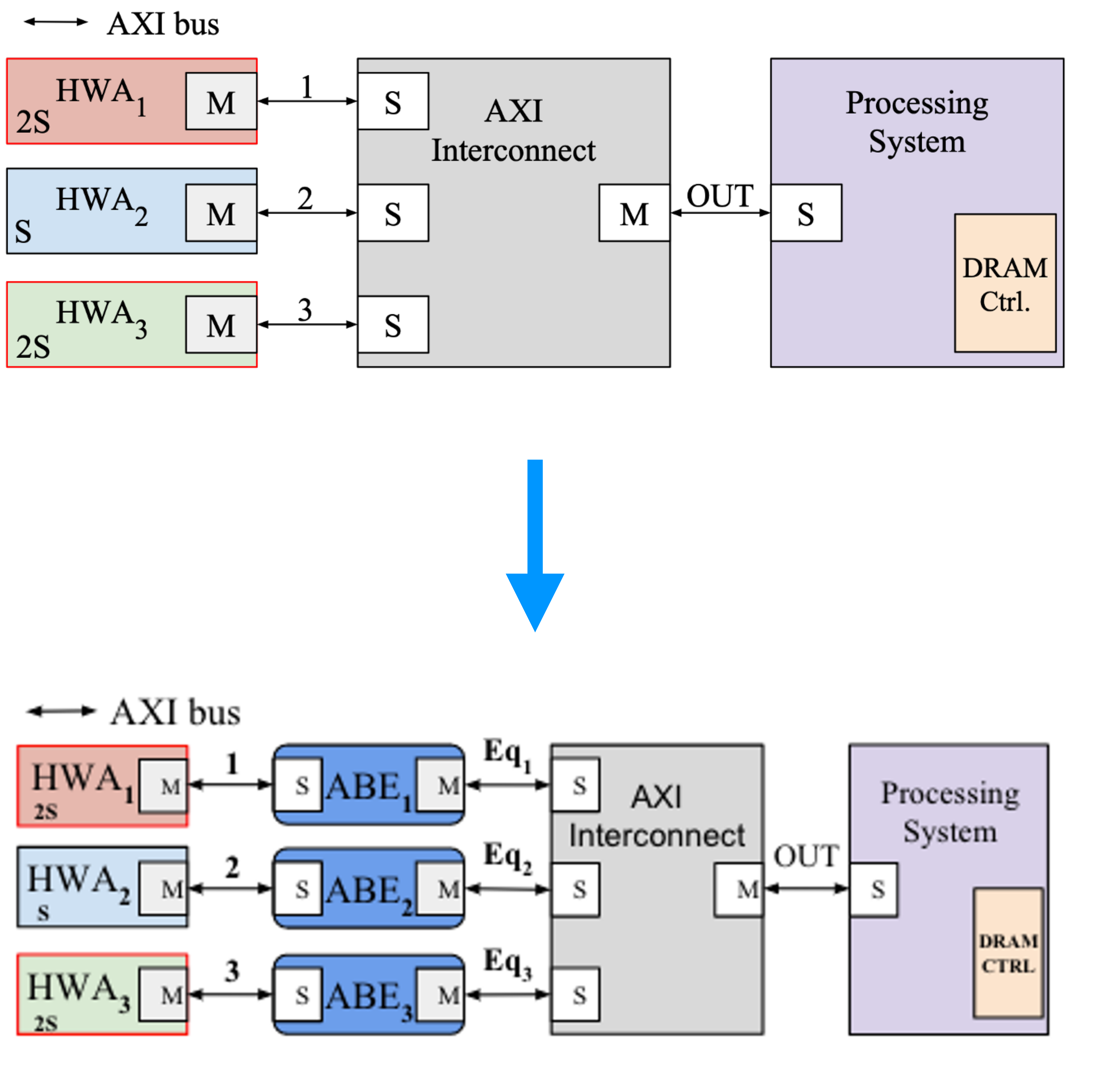
Enforce a nominal bus configuration of the HAs

Makes transactions homogenous

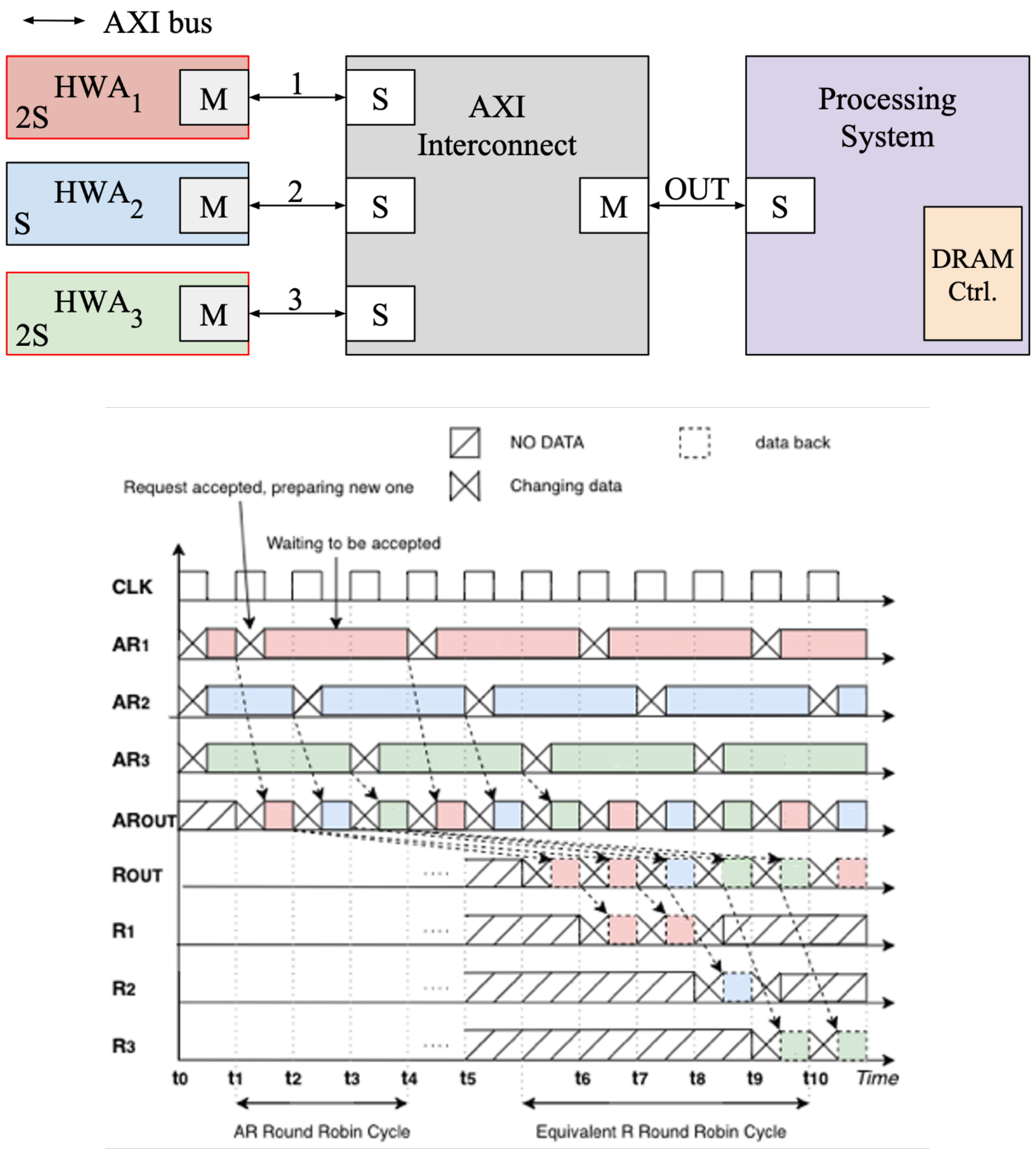
AXI compliant - transparent

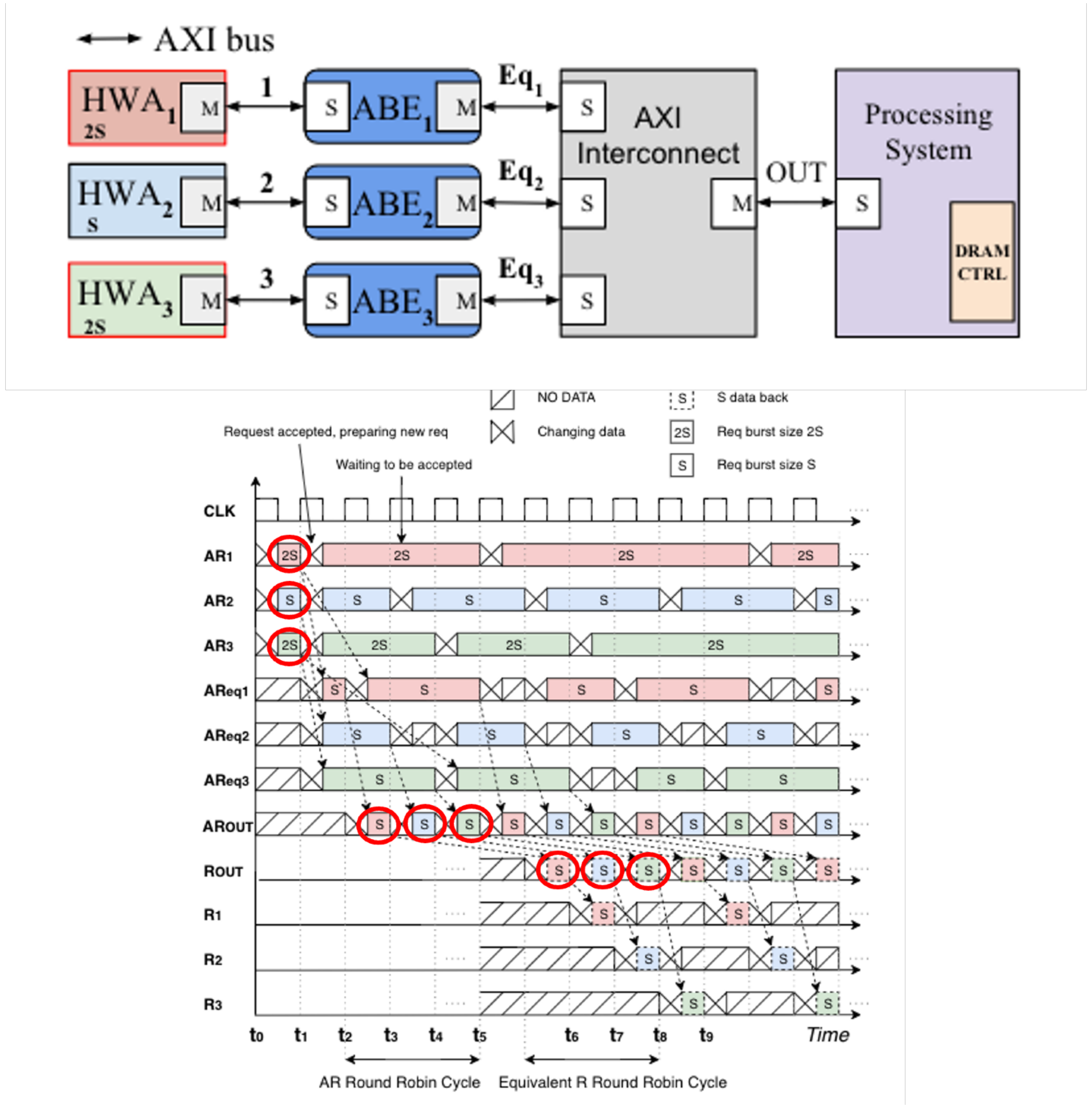
The interconnect arbitrates homogenous transactions

Enforce a fair per-manager granularity on data

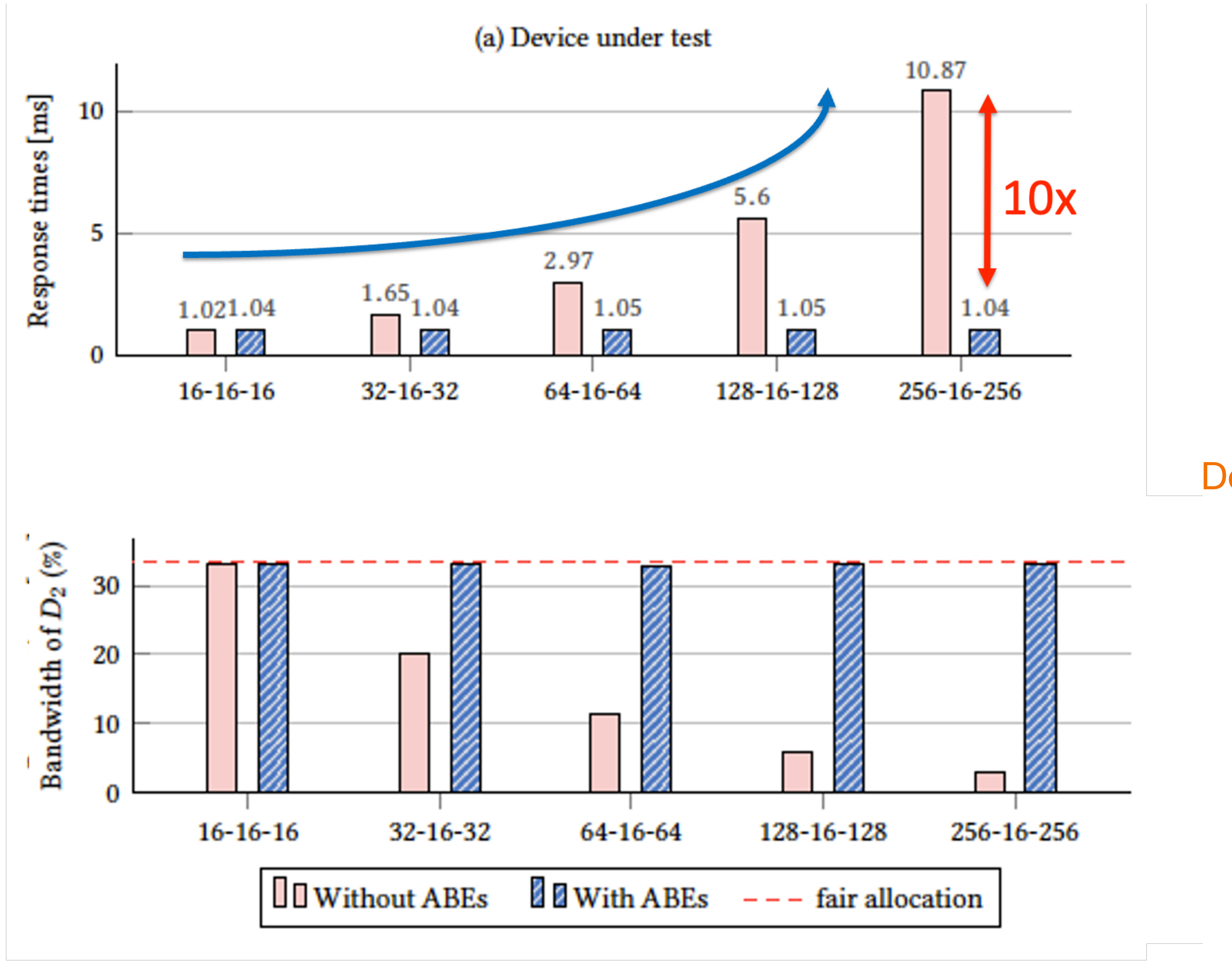


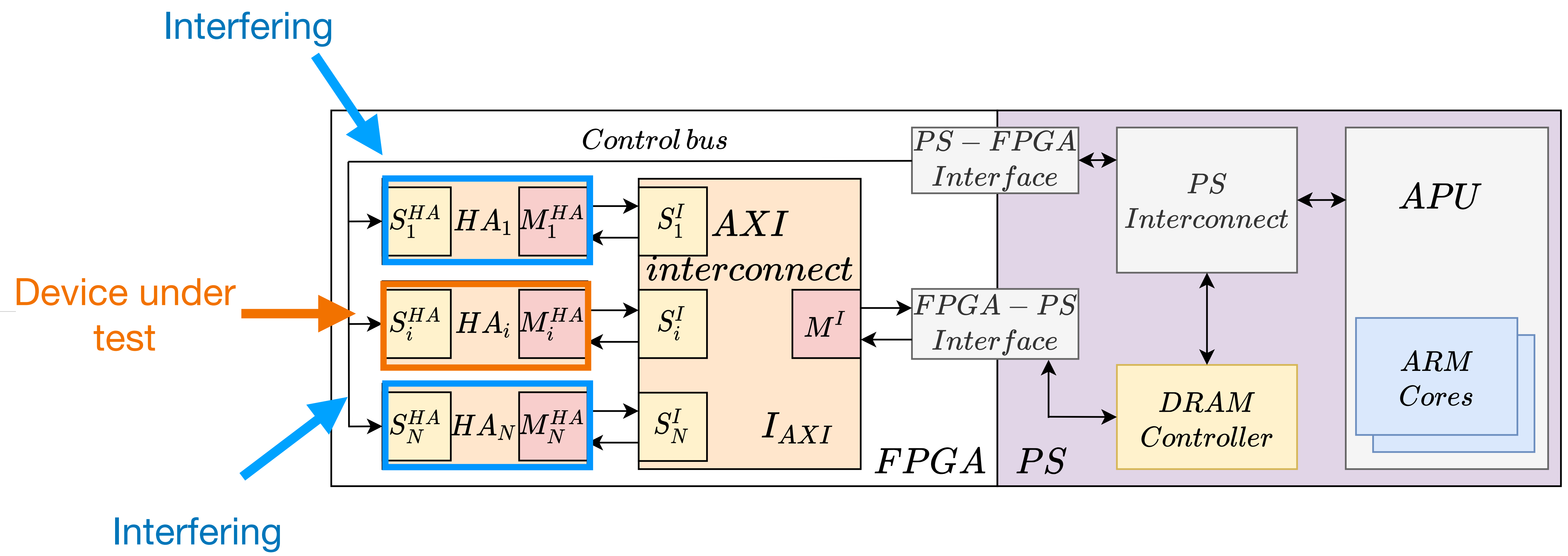
The AXI burst equalizer in action





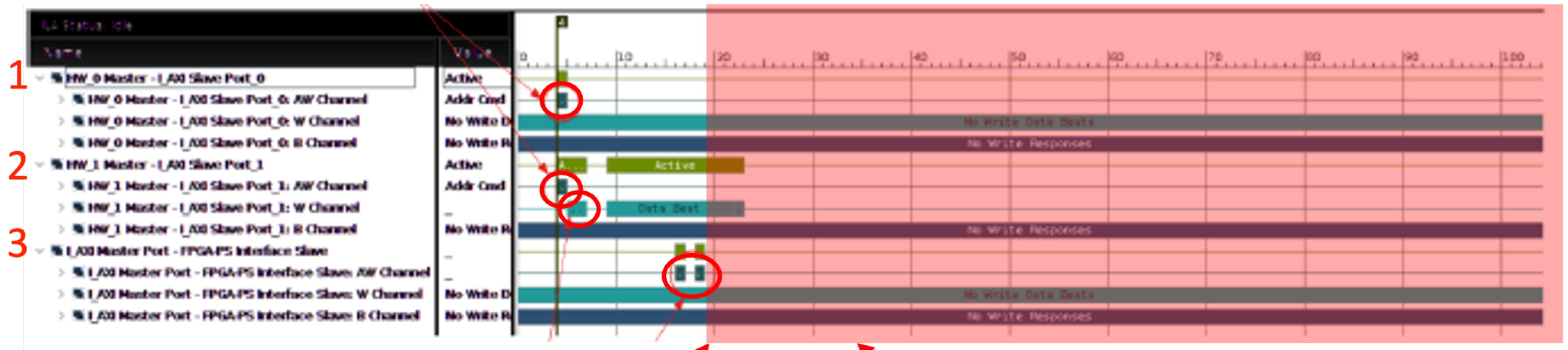
The AXI burst equalizer in action





Associated bandwidth is fair, predictable, and independent of the structure of the transactions

Preventing denial of service of shared resources



Actual System ILA track recorded on a Zynq Ultrascale+ plaftform (ZCU102) using Xilinx Vivado 2018.2

Source of the issue: controllers are trusted to complete (rapidly) their initiated write transactions and release the bus

The solution should be able to:

1) Recognize when a stall is **endangering** the system execution

2) Restore a **safe** condition of the bus - **guarantee** access from the other HAs

How do we know when a stall is dangerous?

Stalls may be introduced by managers during normal execution

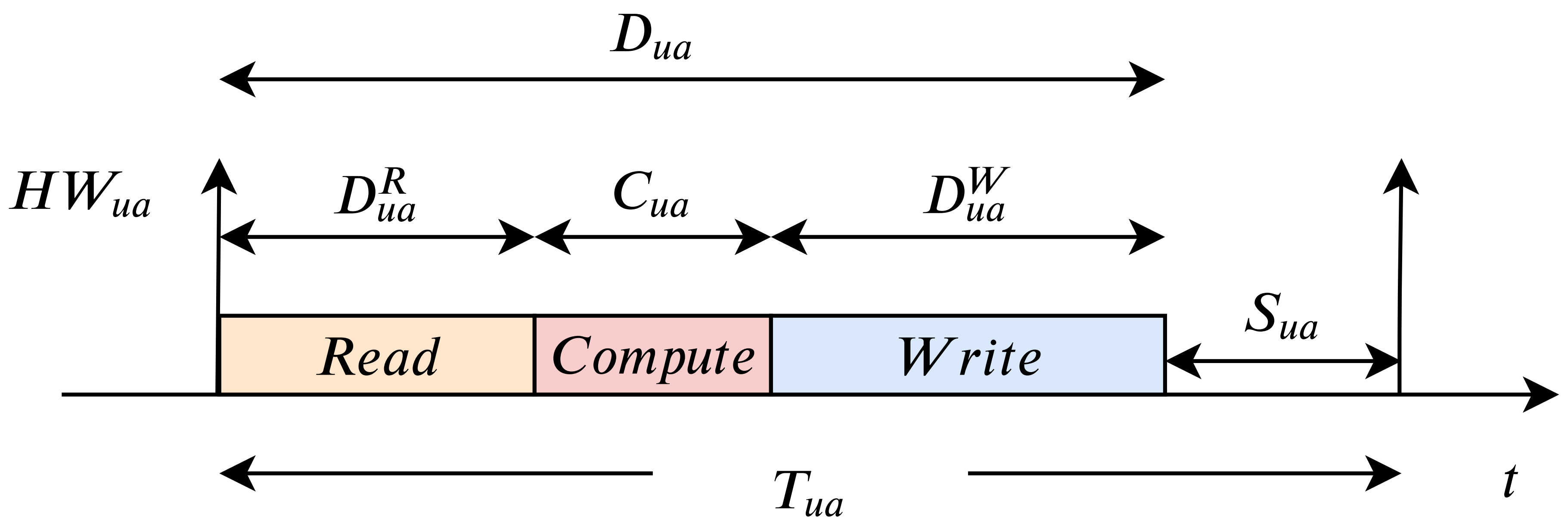
1. When does a stall become a threat to the system execution?

a) Defined the model for the HAs, interconnect, and peripherals

b) Propose a worst-case response time analysis for the HAs

c) Find the maximum acceptable time for stalling the bus (correlated to the slack)

Full mathematical analysis in the paper

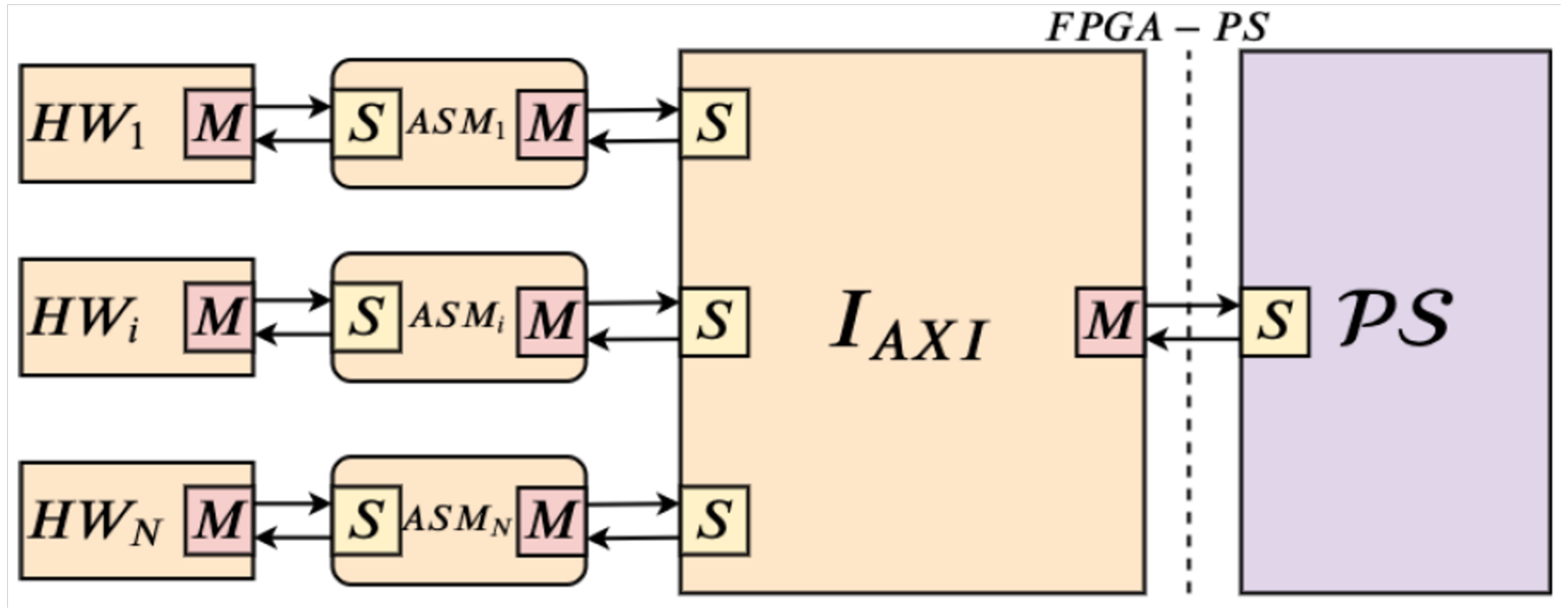


The AXI Stall Monitor (ASM)

2. How to take back control of the bus when stalled?

Monitor the HAs and intervene when system schedulability is endangered

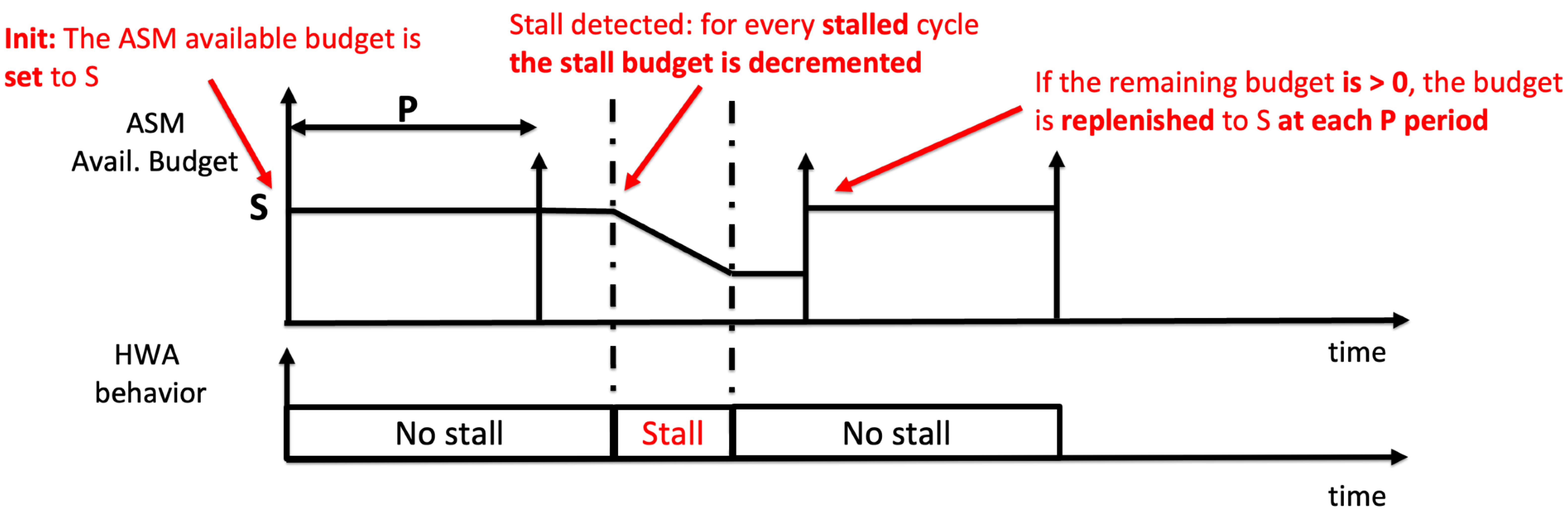
Configured with a stall budget found with the worst-case analysis

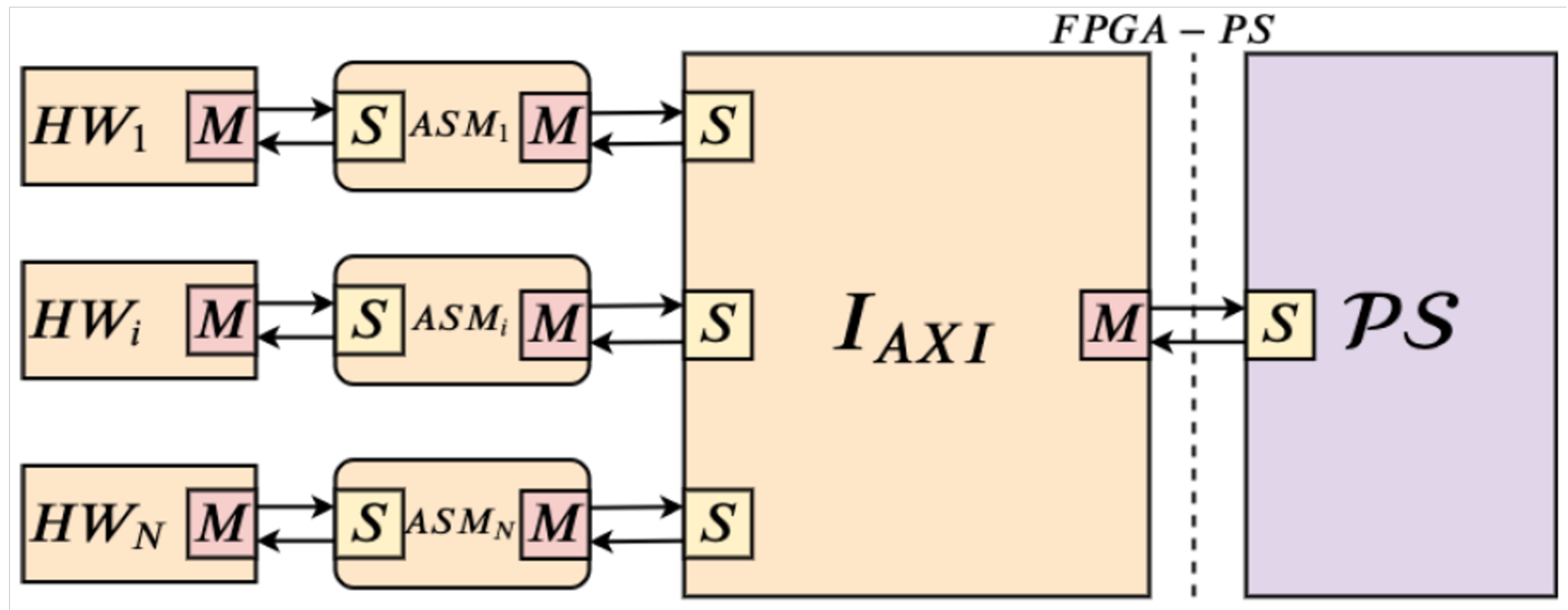


Takes back the control of the bus completing the pending stalled transactions when the system execution (schedulability) is endangered

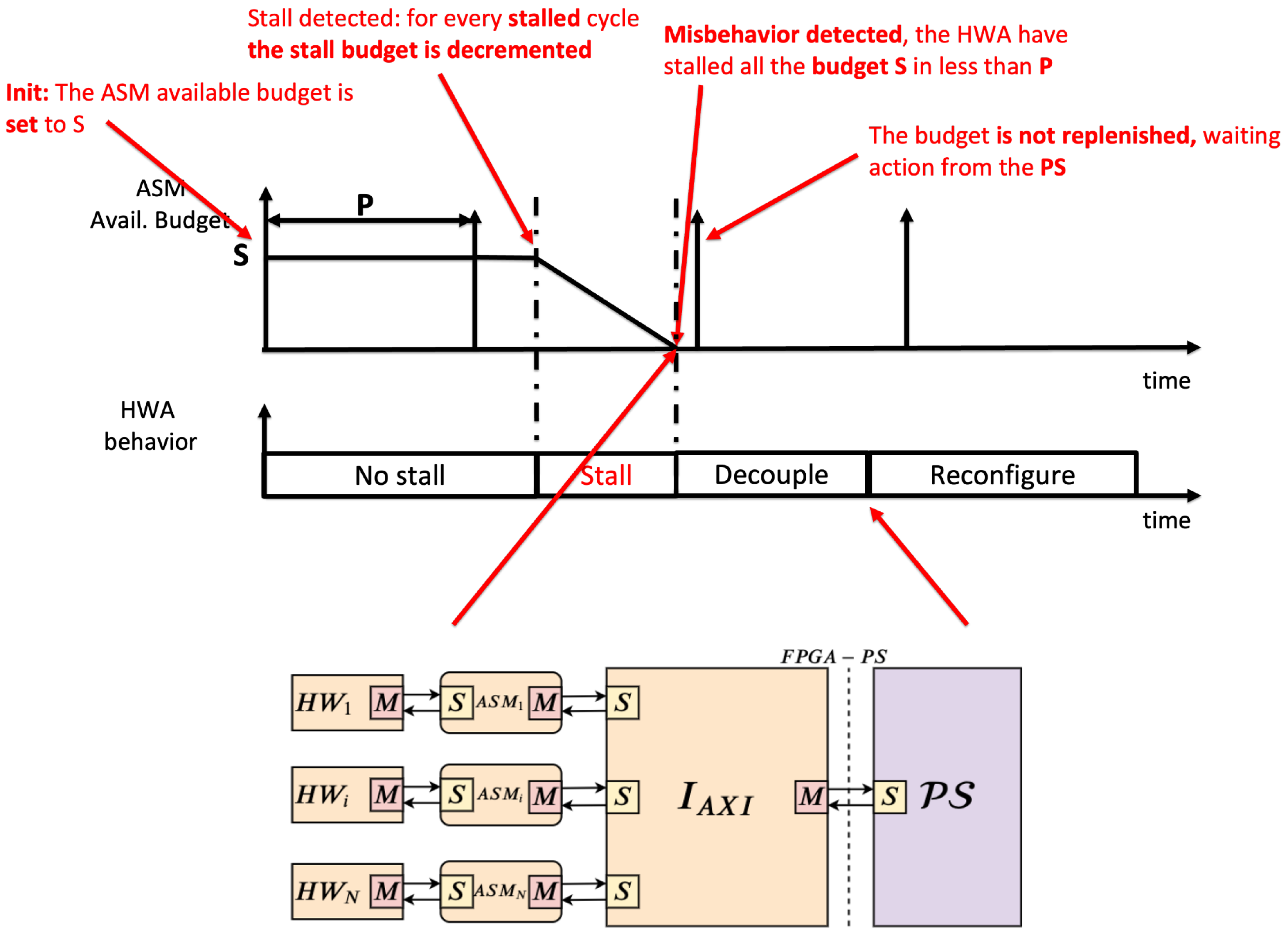
Leave the other controllers access to the shared bus

The AXI Stall Monitor in action





The AXI Stall Monitor in action

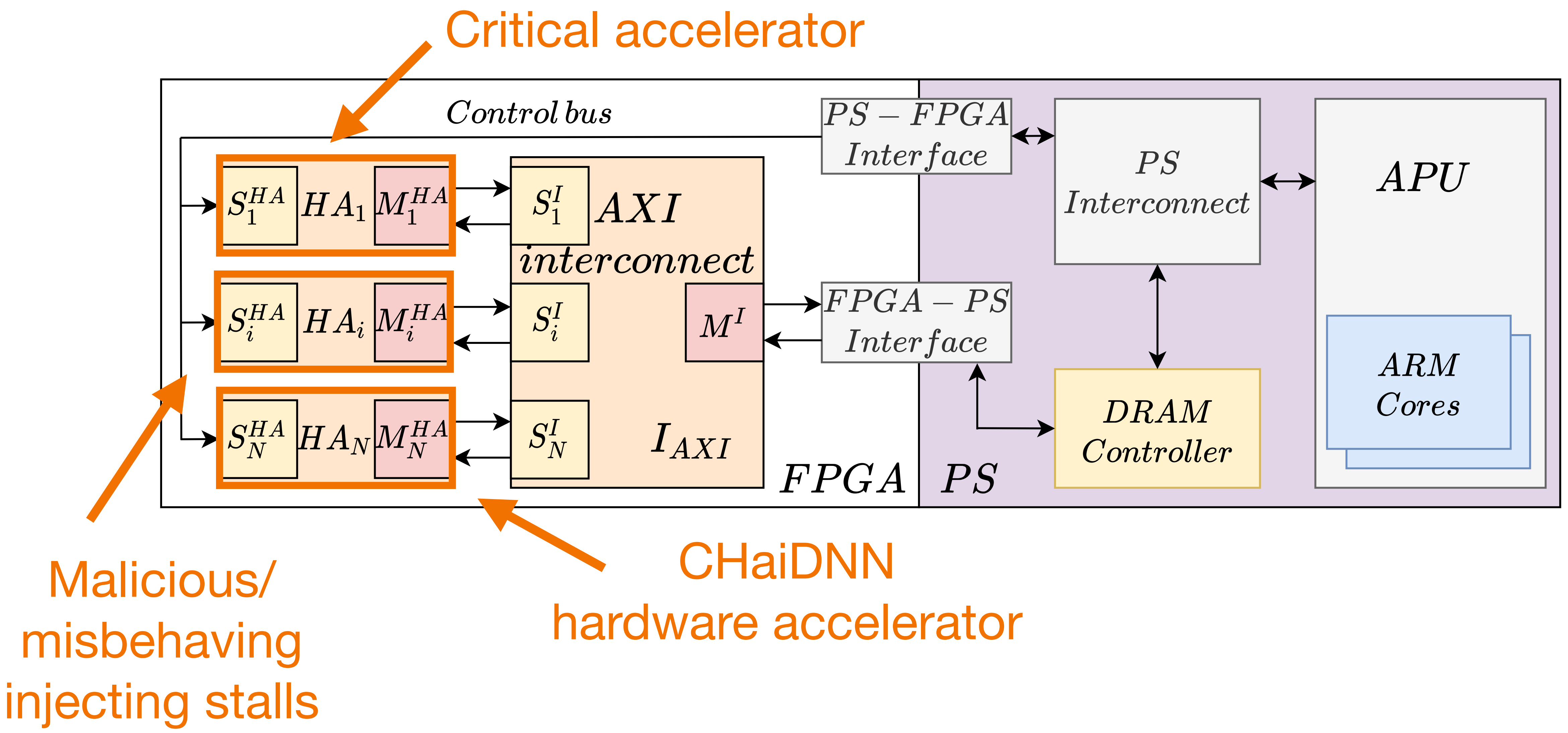


Limitations of ASM

Limitations

Need to fully know the bus workload (periodic)
Need to apply a worst-case analysis

A great solution for real-time systems



Example:
ASM cannot be applied to mixed-critical scenarios (like the CHaiDNN one)

Developed a more versatile and elegant solution

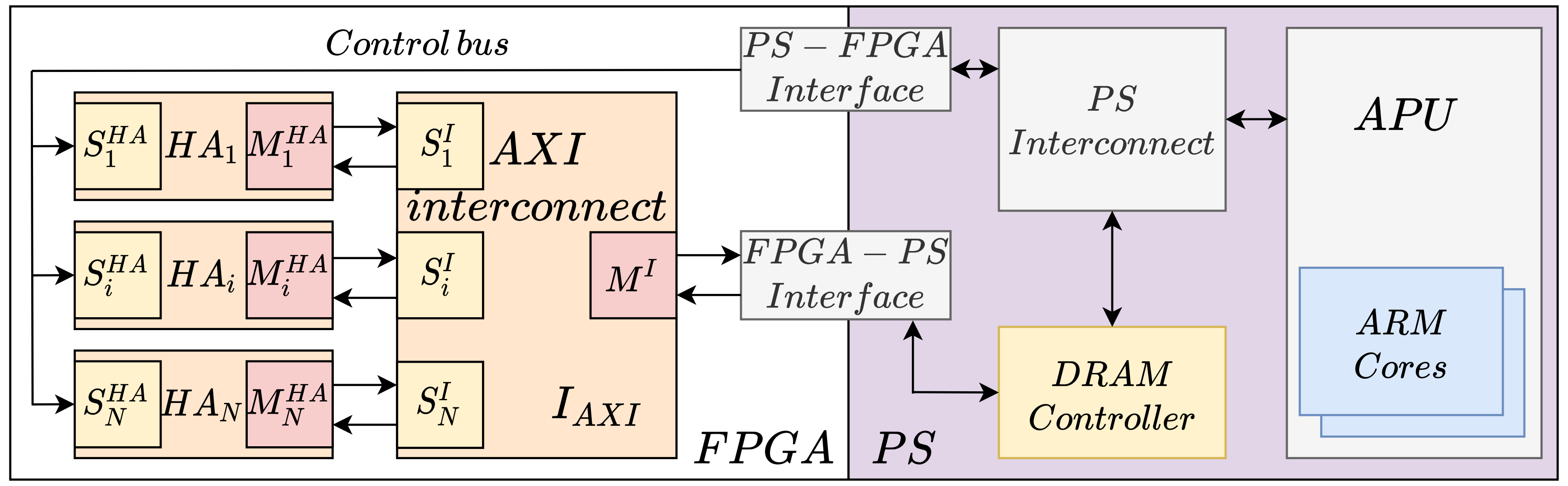
Paper currently under peer review process - stay tuned!

The criticality of the access control system

The access control plays a crucial role in the security of a system

Defines who (controllers) access what (peripherals)

Best approach: give minimum access to the controllers



Challenges

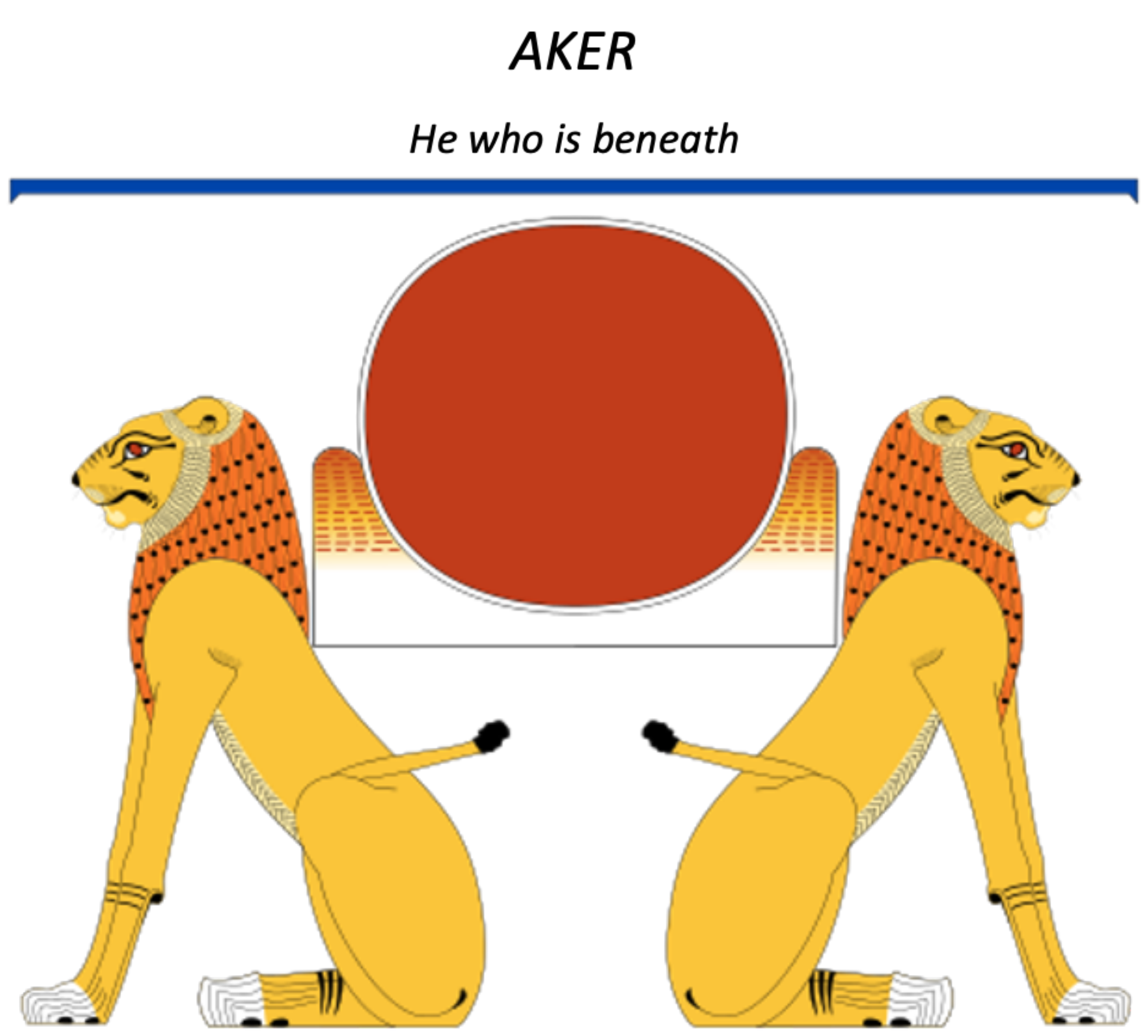
The access control system deployed in commercial platforms may show limited functionalities

Access control systems are sadly known to be a common source of bugs/weaknesses

Top 12 CWEs for 2021 - related to access control

CWE-1189	Improper Isolation of Shared Resources on System-on-a-Chip (SoC)
CWE-1191	On-Chip Debug and Test Interface With Improper Access Control
CWE-1231	Improper Prevention of Lock Bit Modification
CWE-1233	Security-Sensitive Hardware Controls with Missing Lock Bit Protection
CWE-1240	Use of a Cryptographic Primitive with a Risky Implementation
CWE-1244	Internal Asset Exposed to Unsafe Debug Access Level or State
CWE-1256	Improper Restriction of Software Interfaces to Hardware Features
CWE-1260	Improper Handling of Overlap Between Protected Memory Ranges
CWE-1272	Sensitive Information Uncleared Before Debug/Power State Transition
CWE-1274	Improper Access Control for Volatile Memory Containing Boot Code
CWE-1277	Firmware Not Updateable
CWE-1300	Improper Protection of Physical Side Channels

The AKER framework



AKER is a framework for building safe and secure access control systems

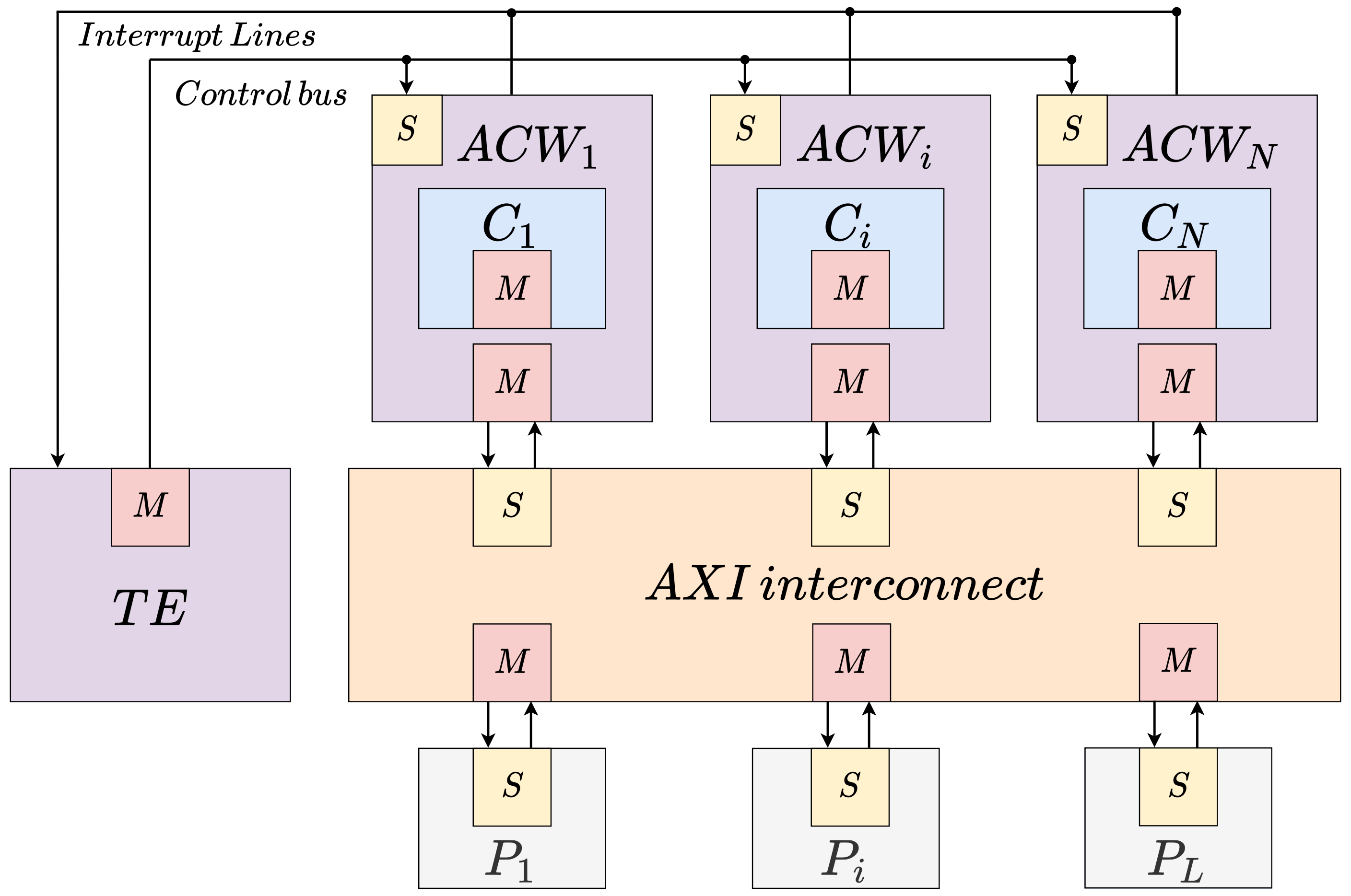
AKER is based on two pillars

The access control wrapper (ACW)

Universal building block for AKER-based access control systems

AKER security verification

Extensive property-based addressing the MITRE CWEs



Concluding remarks

Guidelines for secure integration of controllers

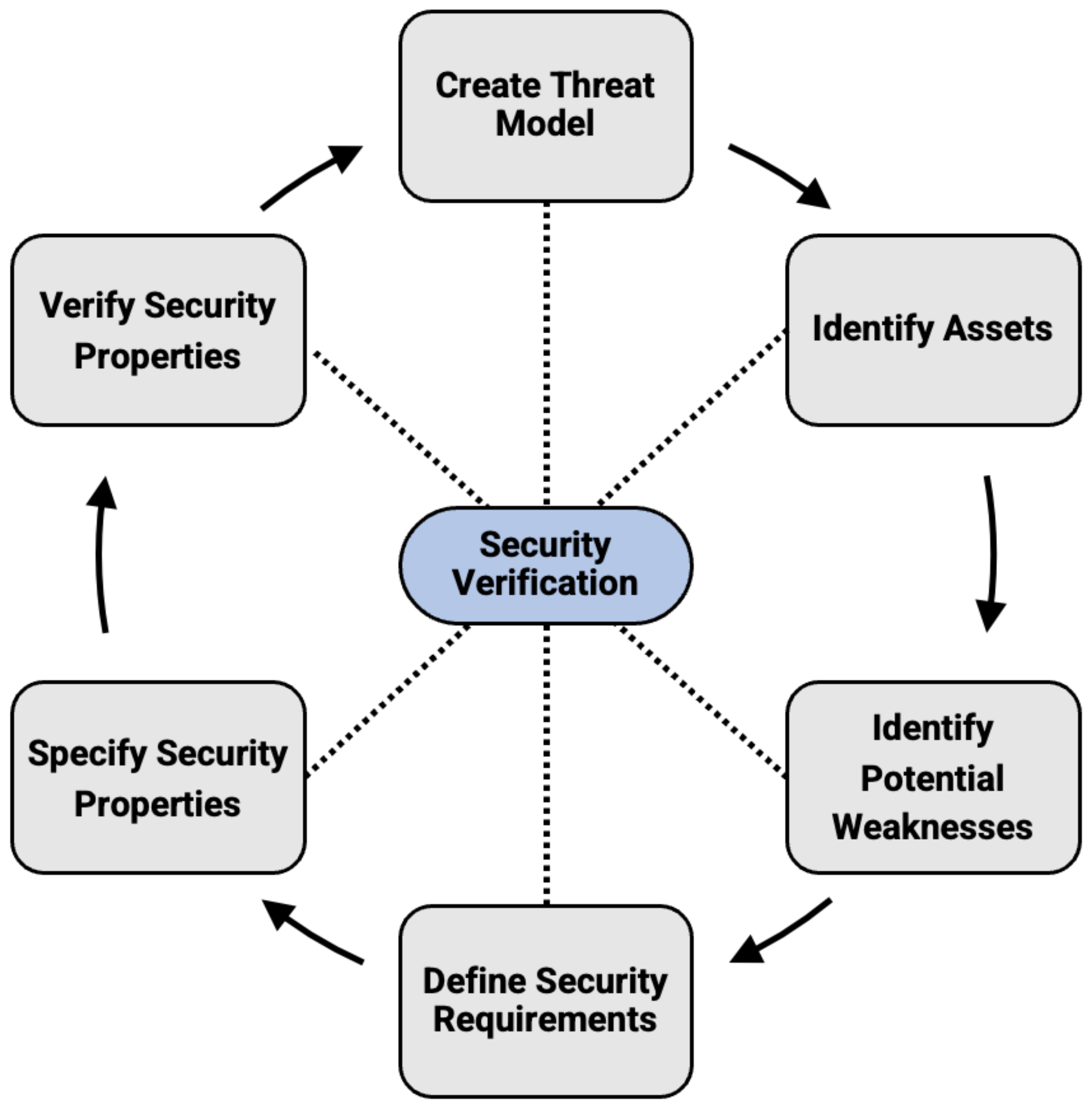
Perform an extensive safety/security verification of the bus interaction of the controllers

Proposal

Leveraged Information Flow Tracking (IFT) to verify the safety of bus interactions among on-chip hardware resources.

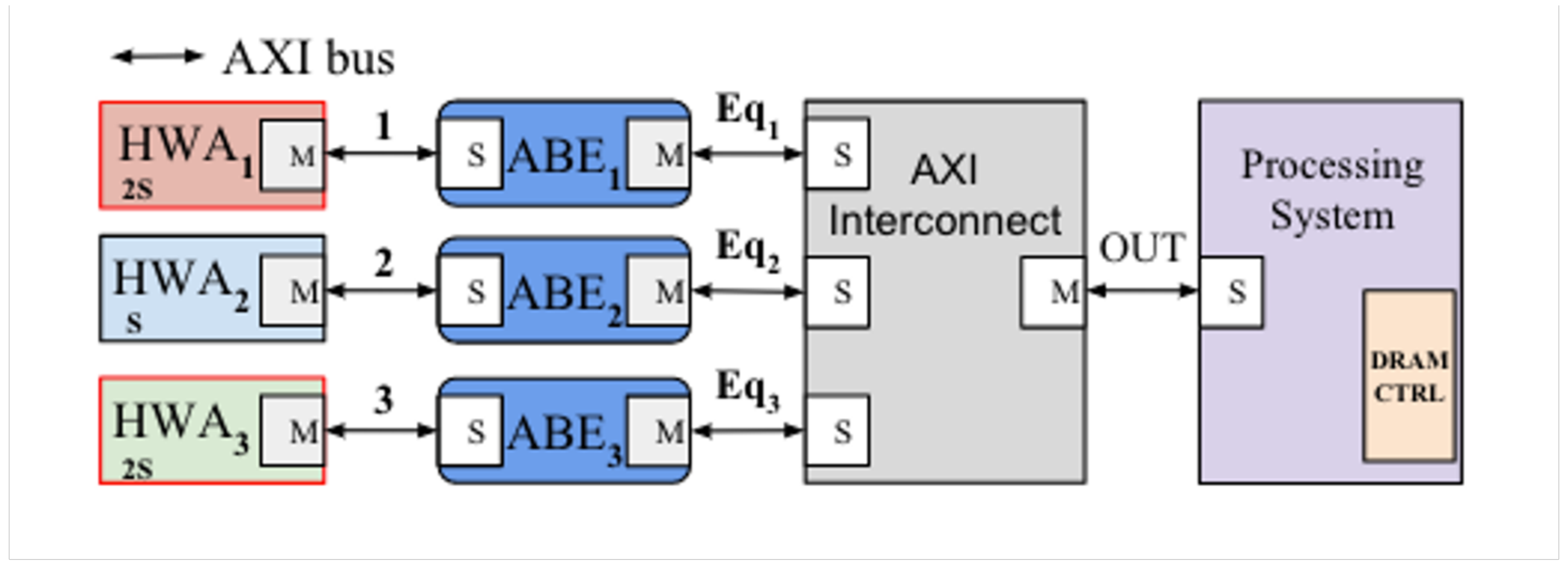
Tortuga Logic Radix-S IFT tool

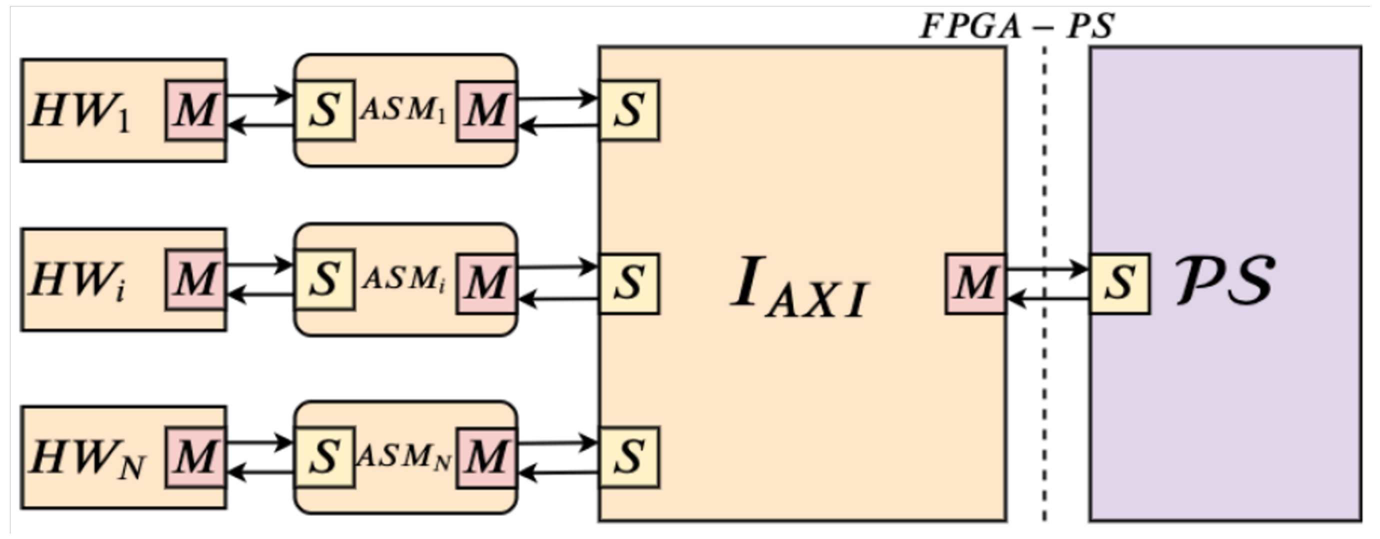
Tortuga Logic™



Meza, A., Restuccia, F, Kastner, R, and Oberg, J (2022, July). ***Safety Verification of Third-Party Hardware Modules via Information Flow Tracking***. In 2022 Real-time And intelliGent Edge computing workshop @ Design and Automation Conference (DAC). To appear.

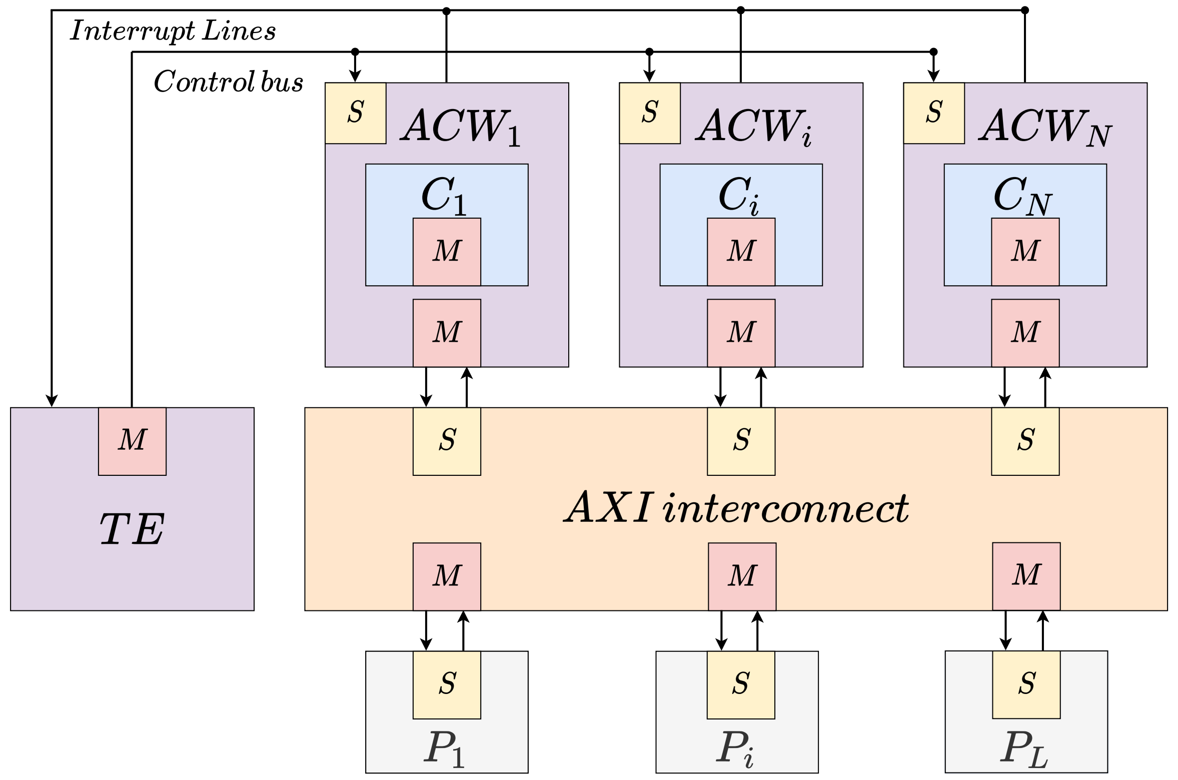
The AXI HyperConnect





+

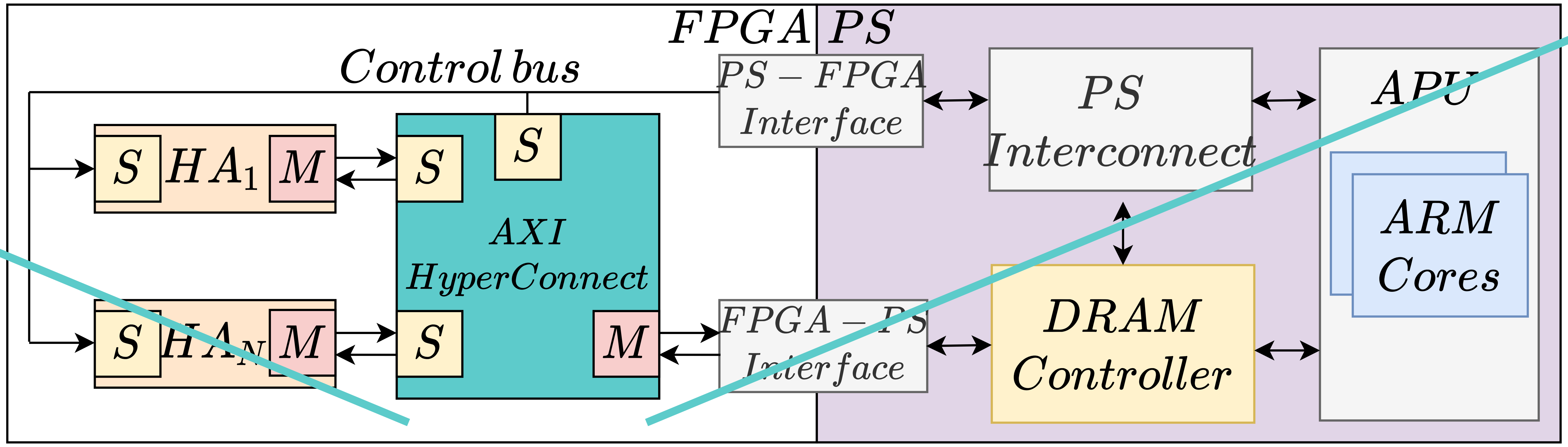
new solution



Enforce fair and predictable bus access

Prevent denial of service of shared resources

Safe and secure access control system



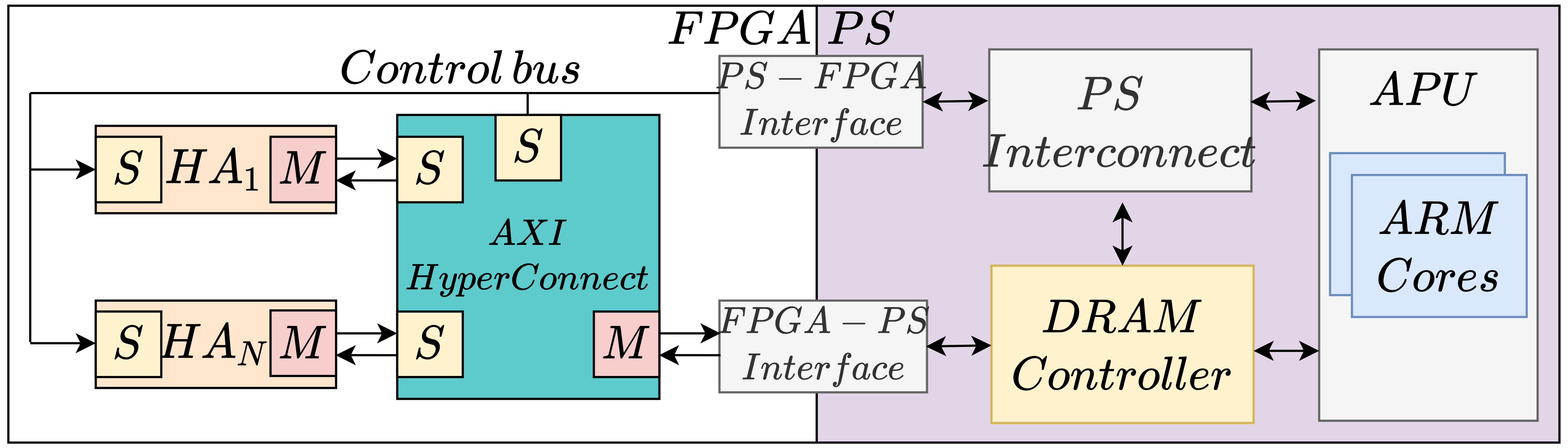
The AXI HyperConnect + clare accelerat

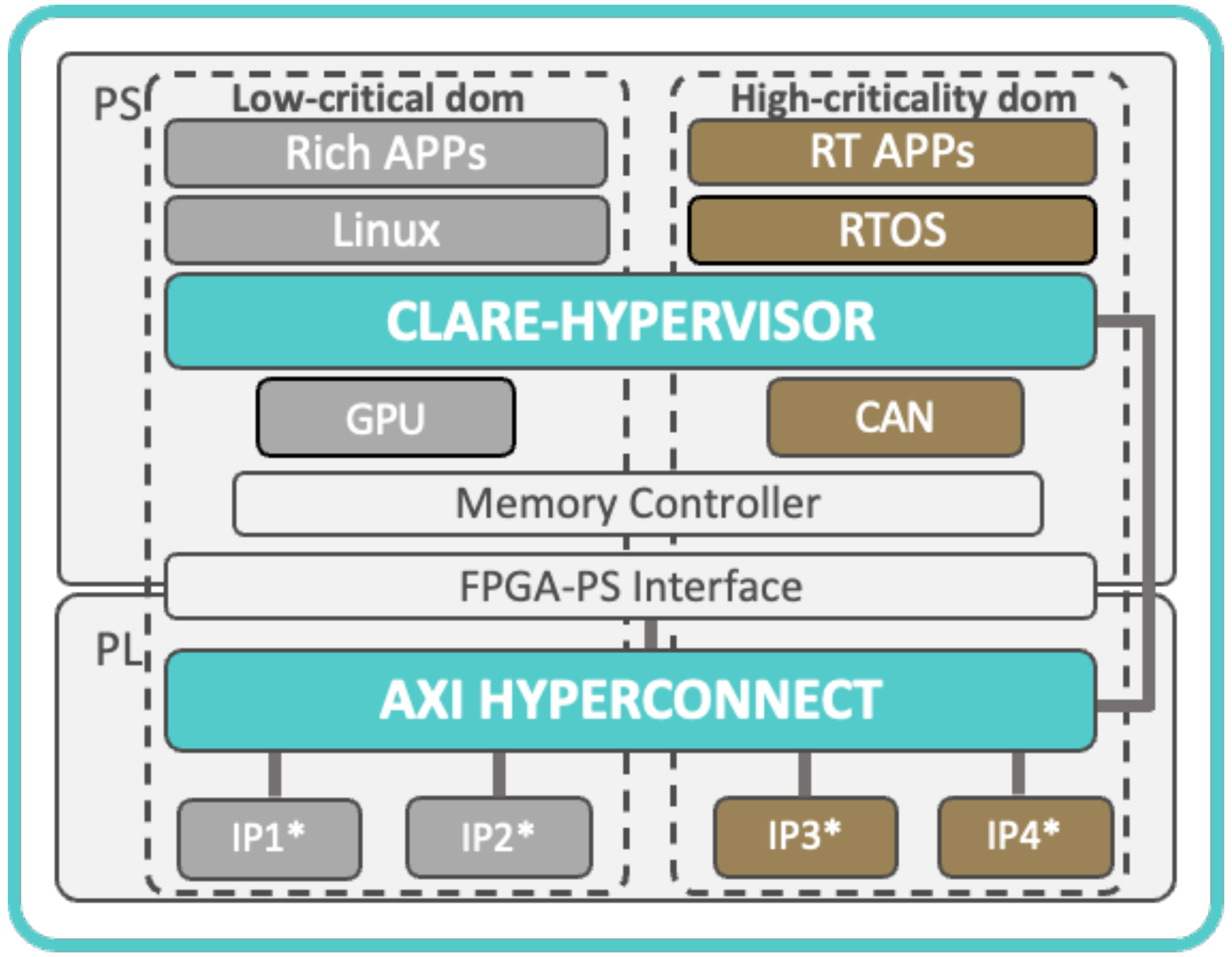
Research interconnect enforcing secure and safe bus interactions

For **standalone** use
or
Integrated with CLARE - **Hypervisor extension**

CLARE is a **hypervisor-centric** software stack for secure, safe, and time-predictable Cyber-Physical Systems

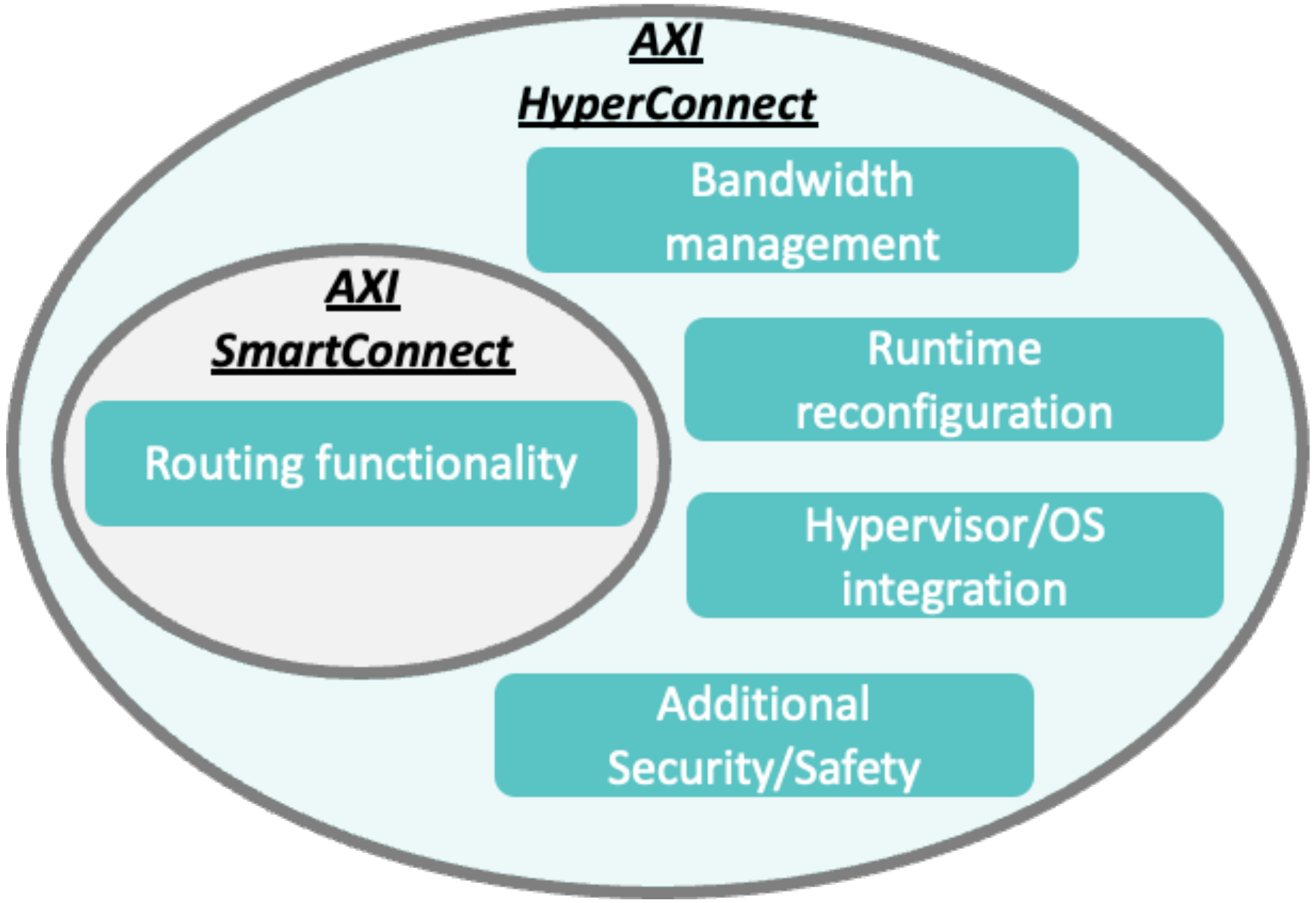
https://accelerat.eu/

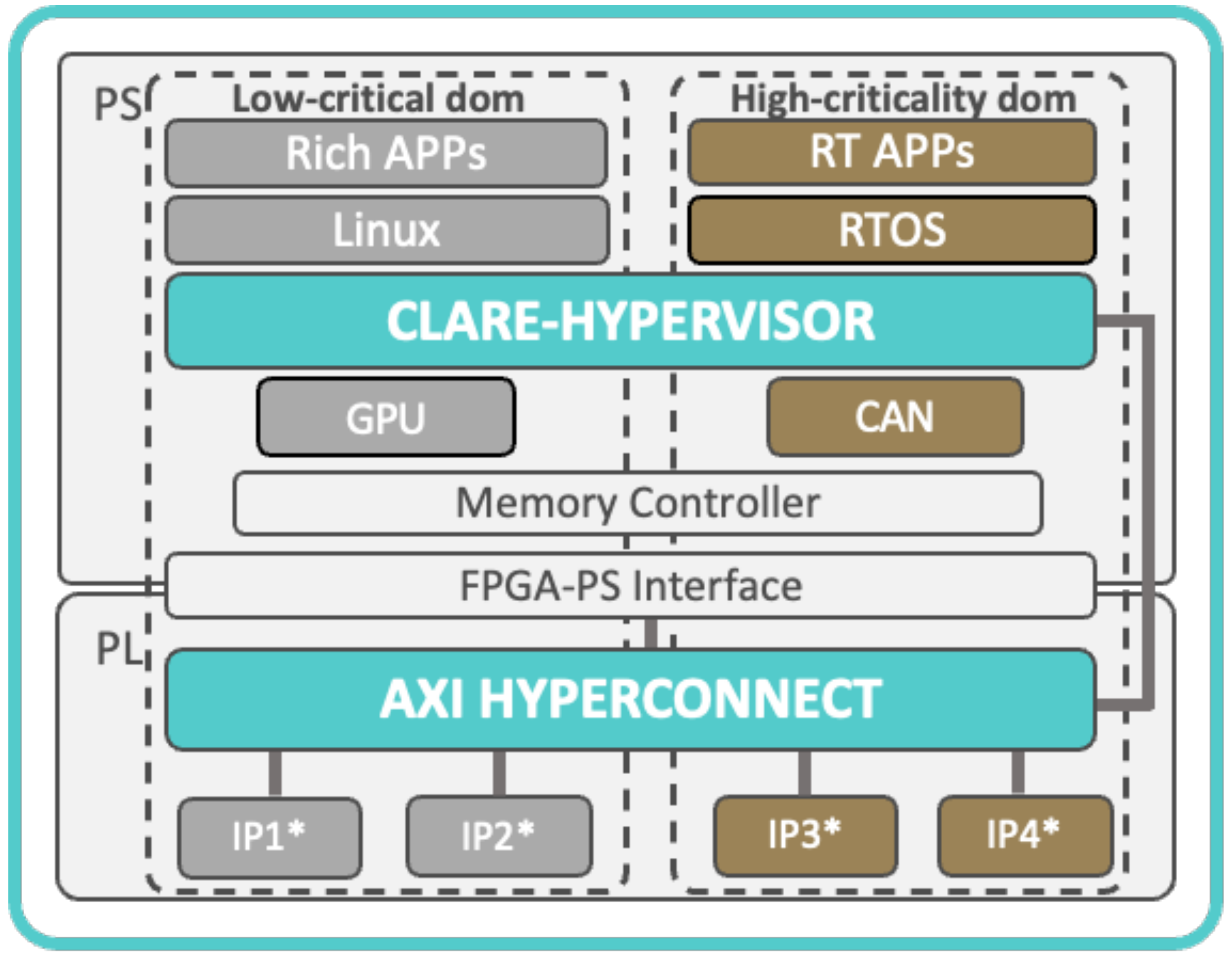




* AXI Manager IPs

AXI HyperConnect features





* AXI Manager IPs

Next steps on the HyperConnect:

Extensive safety/security verification

Automatic firmware management and configuration

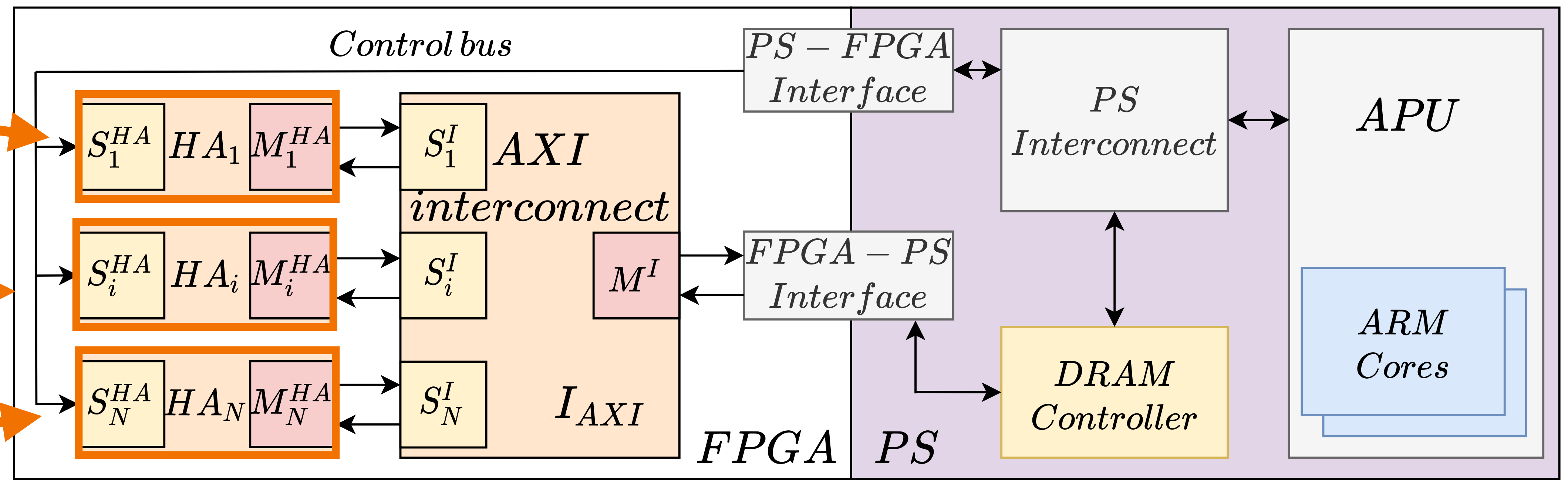
Integration example HyperConnect - mixed-critical application

Injecting AXI stalls - previous example

Critical accelerator

Malicious/ misbehaving injecting stalls

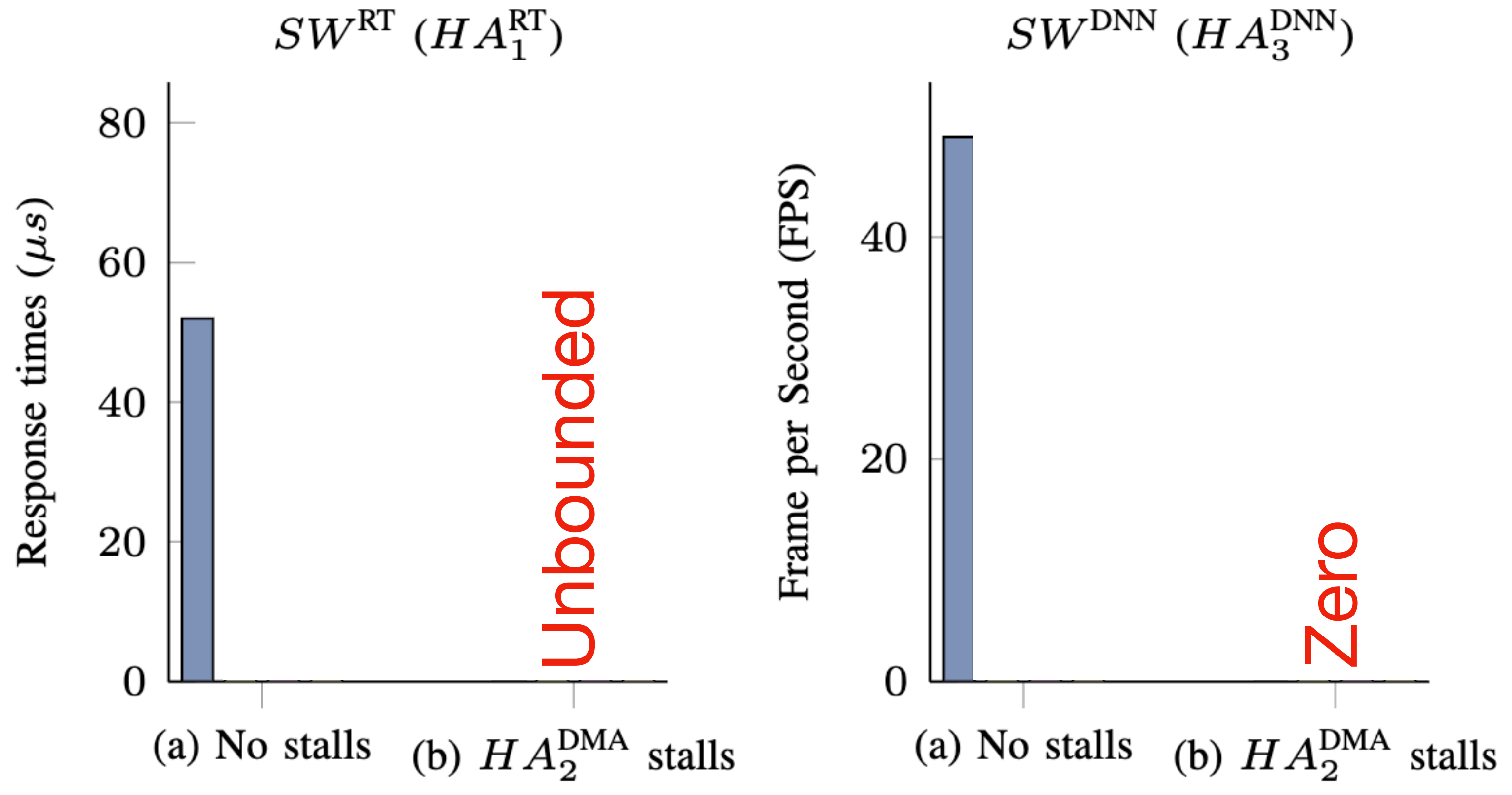
CHaiDNN hardware accelerator



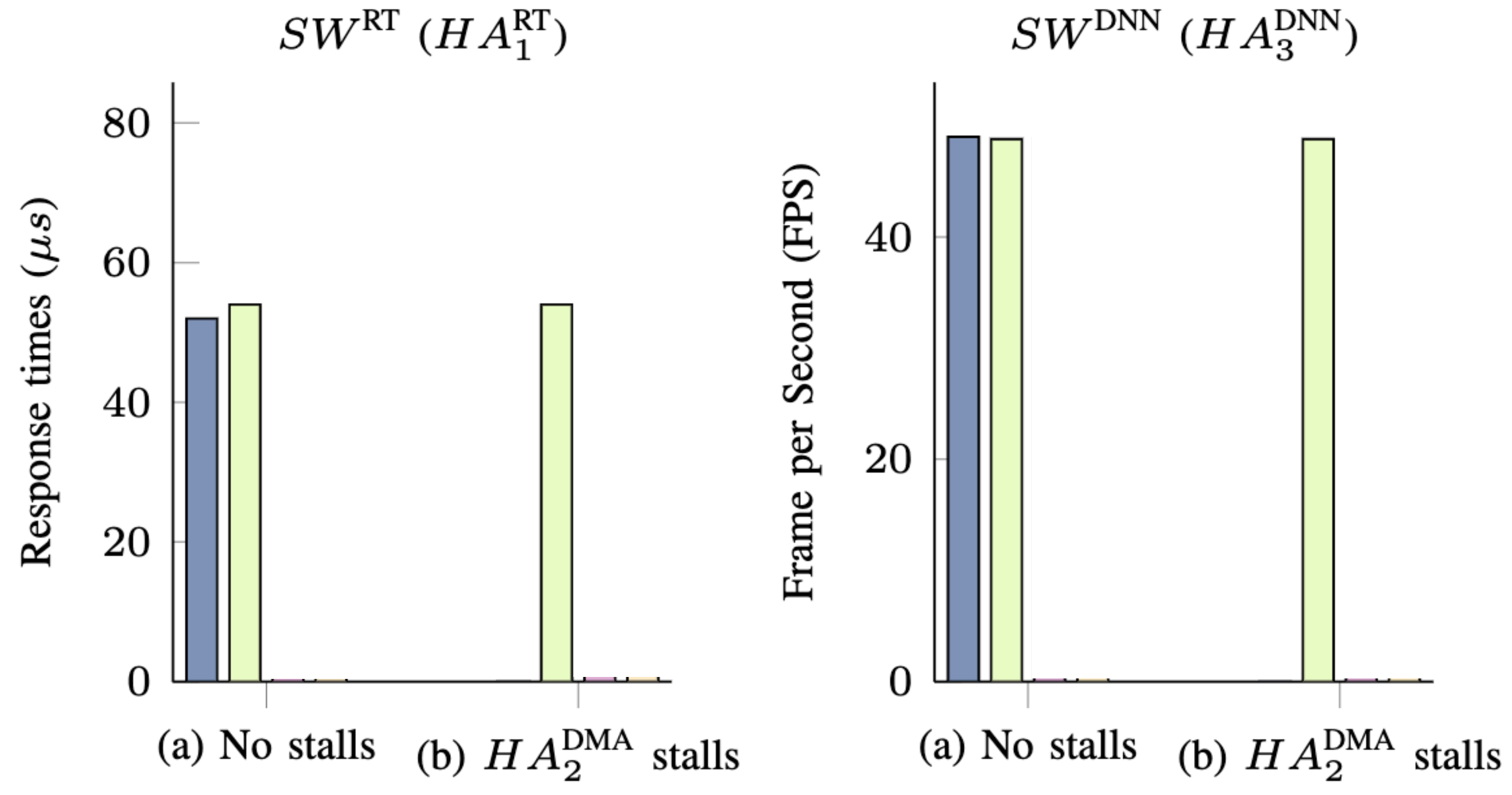
AXI interconnect

AXI HyperConnect

AXI interconnect



AXI HyperConnect



HA_2 inject stalls

Broken execution

HA_2 inject stalls

System is kept operational

Collaborators

Ryan Kastner

Andres Meza

Jason Oberg

UC San Diego

Tortuga Logic

Alessandro Biondi

Marco Pagani

Giorgiomaria Cicero

Mauro Marinoni

Giorgio Buttazzo

Retis Real-Time Systems Laboratory

accelerat

Industrial collaborators: Intel corporation, Tortuga Logic, Leidos.

Contacts and references

frestuccia@ucsd.edu

Linkedin

Restuccia, F., Pagani, M., Biondi, A., Marinoni, M., and Buttazzo, G. (2019). ***Is your bus arbiter really fair? restoring fairness in axi interconnects for fpga socs***. *ACM Transactions on Embedded Computing Systems (TECS)*, Presented at ESWEEK - CASES 2019, New York, USA.

Restuccia, F., Biondi, A., Marinoni, M., and Buttazzo, G. (2020, May). ***Safely preventing unbounded delays during bus transactions in FPGA-based SoC***. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

Restuccia, F., Biondi, A., Marinoni, M., Cicero, G., and Buttazzo, G. (2020, July). ***AXI hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in FPGA SoC***. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*

Restuccia, F., Meza, A., and Kastner, R. (2021, November). ***Aker: A design and verification framework for safe and secure SoC access control***. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*

Meza, A., Restuccia, F, Kastner, R, and Oberg, J (2022, July). ***Safety Verification of Third-Party Hardware Modules via Information Flow Tracking***. In 2022 Real-time And intelliGent Edge computing workshop @ Design and Automation Conference (DAC). To appear.

On timing predictability for bus interactions:

Restuccia, F., and Biondi, A. (2021, December). ***Time-Predictable Acceleration of Deep Neural Networks on FPGA SoC Platforms***. In *2021 IEEE Real-Time Systems Symposium (RTSS).*

Restuccia, F., Pagani, M., Biondi, A., Marinoni, M., and Buttazzo, G. (2020). ***Modeling and analysis of bus contention for hardware accelerators in FPGA SoCs.*** In *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*.