IRQ Coloring: Mitigating Interrupt-generated Interference on ARM Multicore Platforms

Diogo Costa

HiPEAC23 @ Toulouse

January 18th, 2023

Agenda

01 **Motivation**
Interrupt-generated Interference in MCSs

02 **IRQ Coloring**
Overview and Specification

03 **Evaluation**
Setup and Results

04 **Roadmap**
WiP and Future Steps

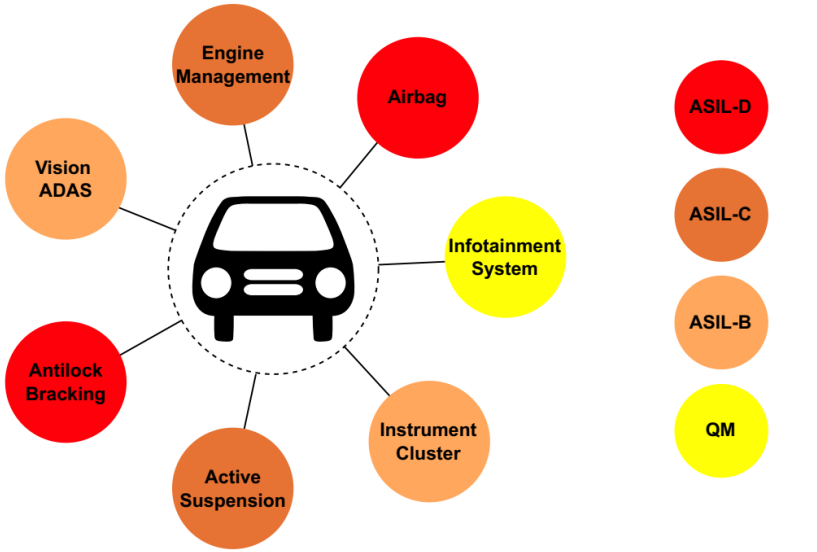
Motivation

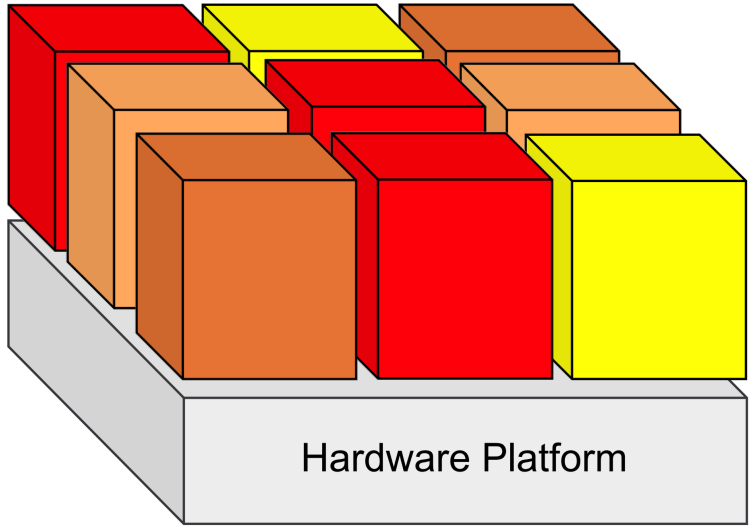
01

Consolidation of Mixed-Criticality Systems (MCS)

- **Challenges for the certification of MCS**
  - Increasing digitalization trend
  - Well-established trend toward multicore

- **Consolidation of different applications on the same hardware platform:**
  - Virtualization as a key-enabling technology
  - Challenges
    - Contention at shared hardware resources

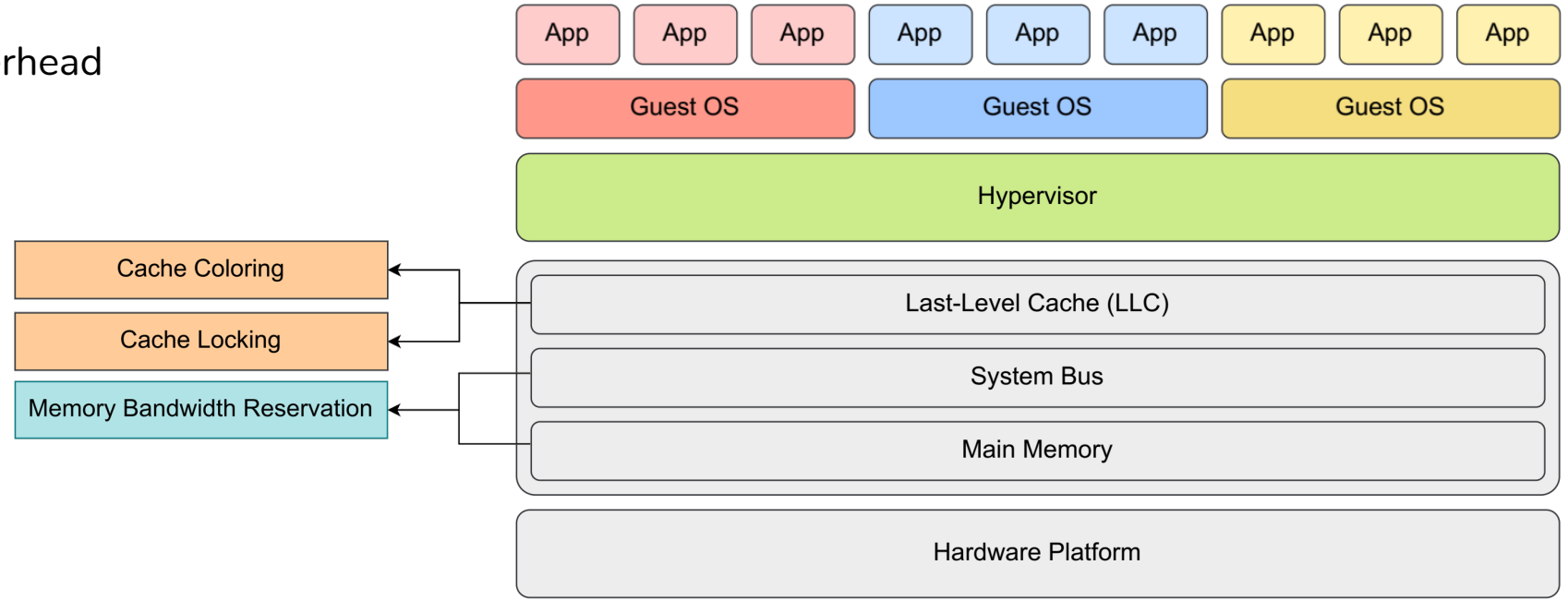




Motivation: The Problem

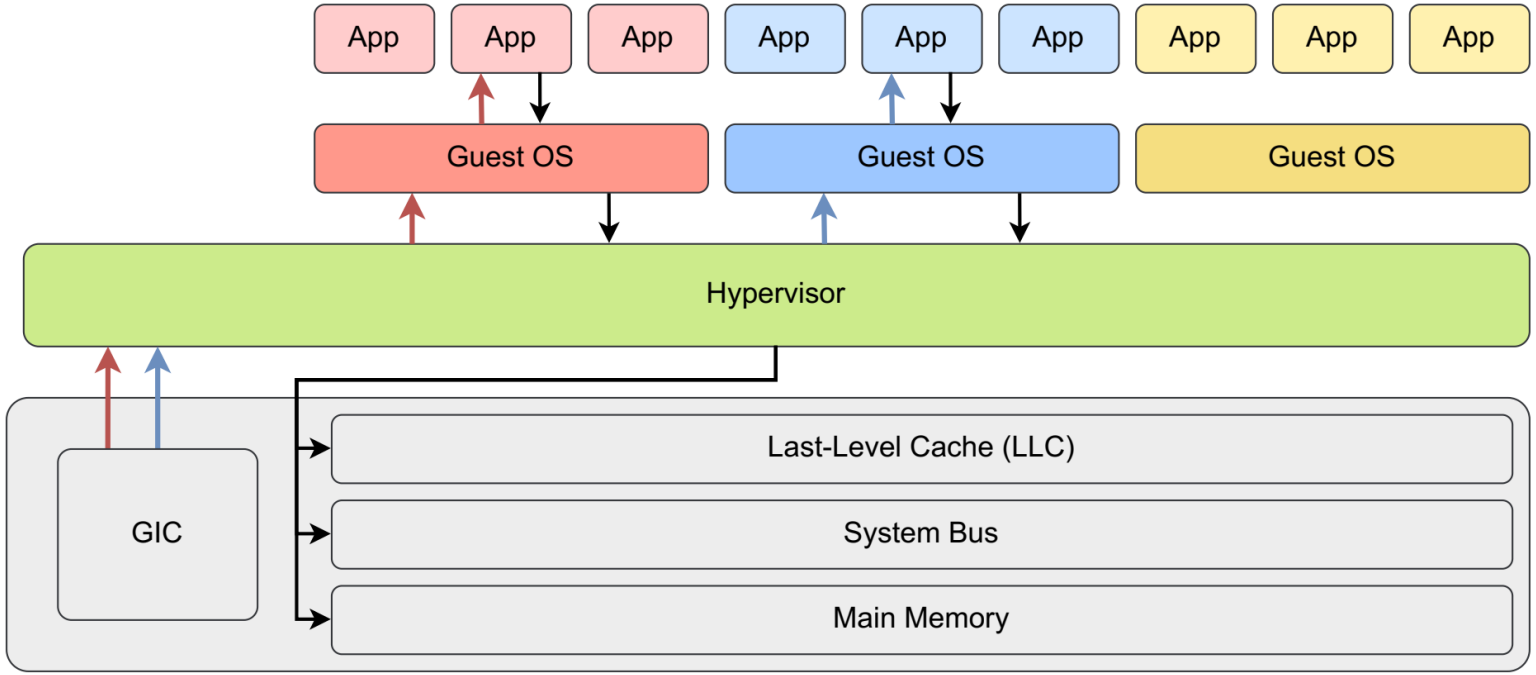
- **Cache partitioning limitations**:
  - **Cache Locking** - Hardware dependency
  - **Cache Coloring** - Virtual memory (MMU) available

- **Memory bandwidth reservation limitations:**
  - Hardware dependency
  - Significant runtime-overhead



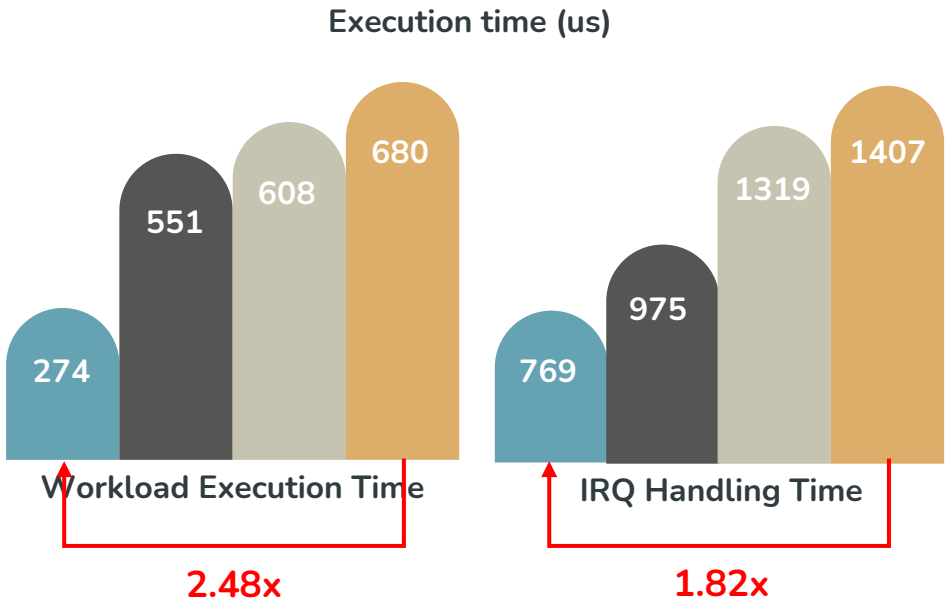
Motivation: The Problem

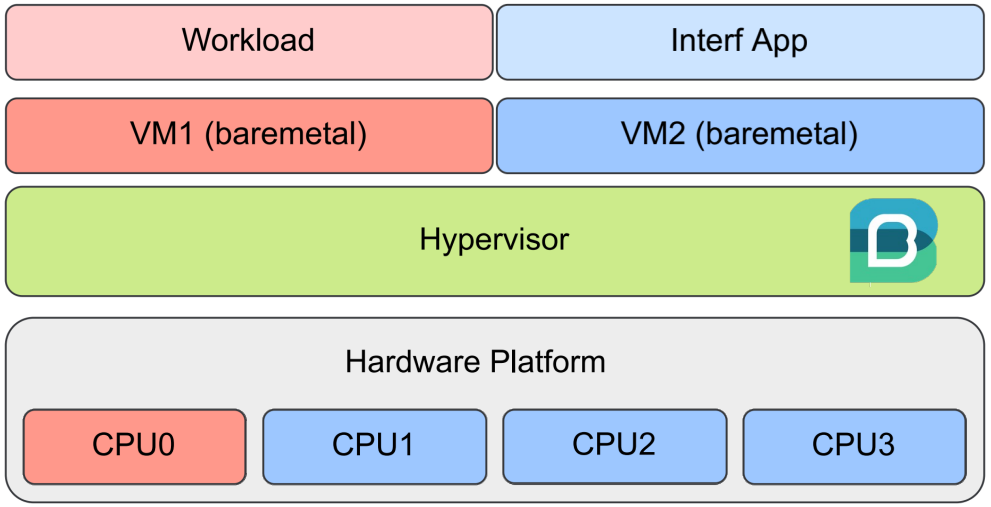
- **Interrupt-driven workloads**:
  - Unpredictable diversions in the overall computational execution flow
  - Stresses the microarchitectural shared components
  - A storm of interrupts can create a DoS attack



Motivation: The Evidence

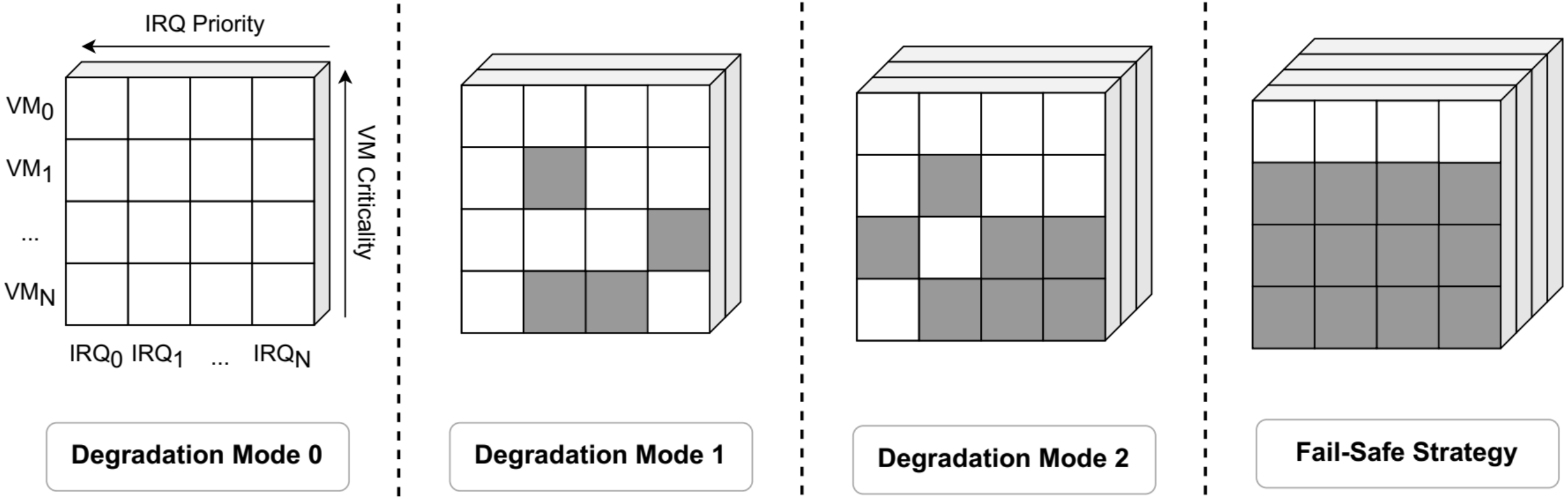
- **Empirical Evidences:**
  - **Solo**: Only ASIL-D VM running using 1 CPU:
  - **1 Interf VM**: [Solo] + QM VM running with 1 CPU
  - **2 Interf VM**: [Solo] + QM VM running with 2 CPU
  - **3 Interf VM**: [Solo] + QM VM running with 3 CPU



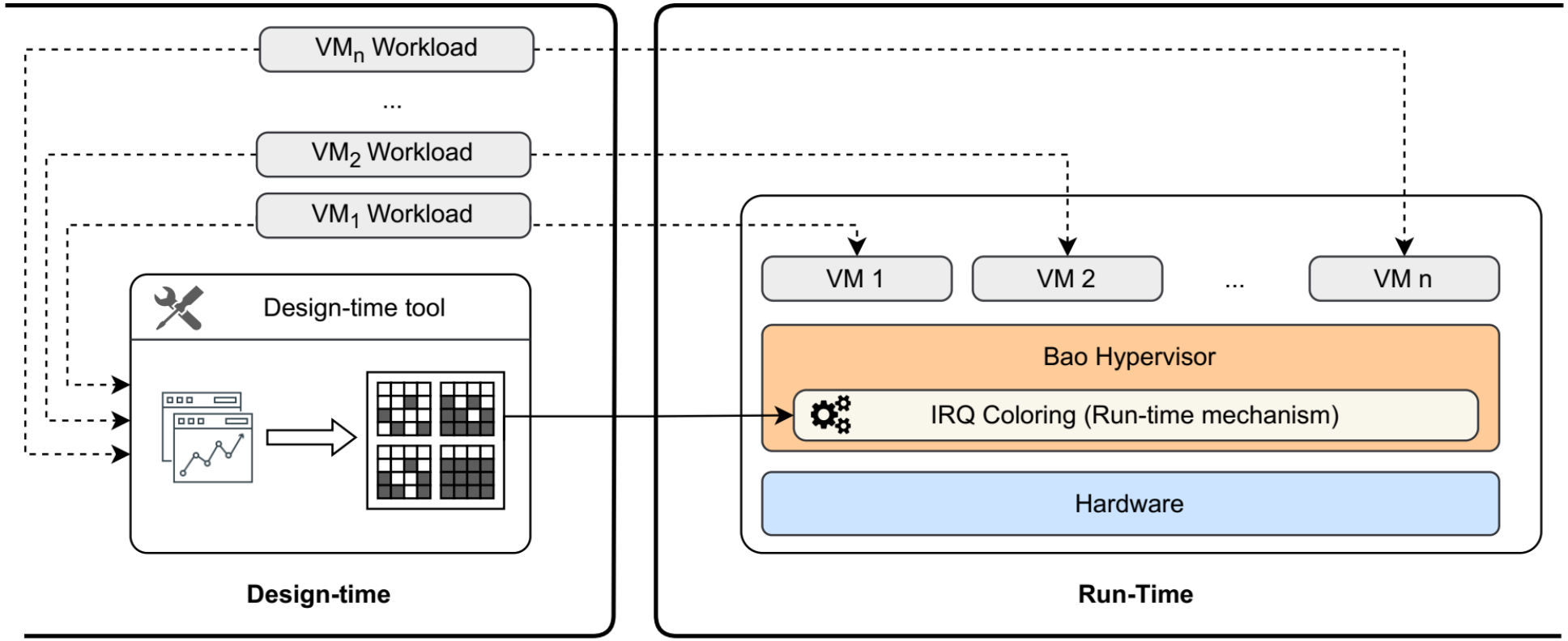


IRQ Coloring

Conceptual Design



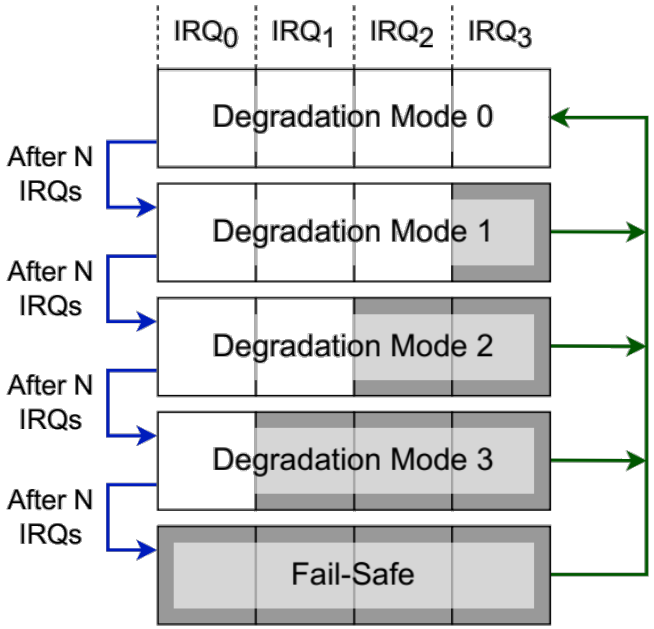
System Overview



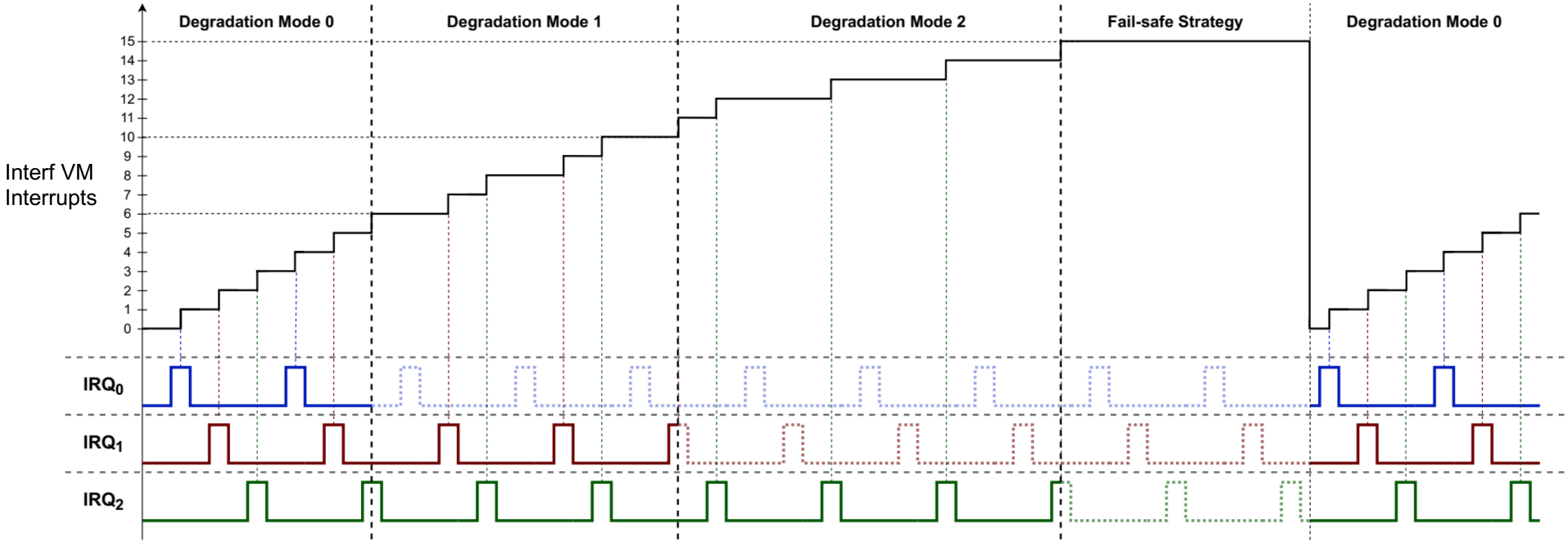
Run-Time Mechanism

Algorithm 1 IRQ Coloring RTM - Implementation

```
1: funtion periodic_timer_handler;
2: begin
3: for VM = 1, 2, ... do
4:     VM_Degradation_mode = 0
5:     Set PMU to generate overflow interrupt at VM_B0
6:     for IRQ = 1, 2, ... do
7:         Unmask IRQ
8:     end for
9: end for
10: Re-schedule timer period
11: funtion pmu_overflow_interrupt_handler;
12: begin
13: for IRQ = 1, 2, ... ∈ IRQ_MAP_Degradation_mode do
14:     Mask IRQ
15: end for
16: VM_Degradation_mode ← VM_Degradation_mode + 1
17: if VM_Degradation_mode < Max_Degradation_modes then
18:     Set PMU to generate overflow interrupt at B_VM,Degradation_mode
19: end if
```



Run-Time Mechanism



System Architecture

Distributor (GICD)

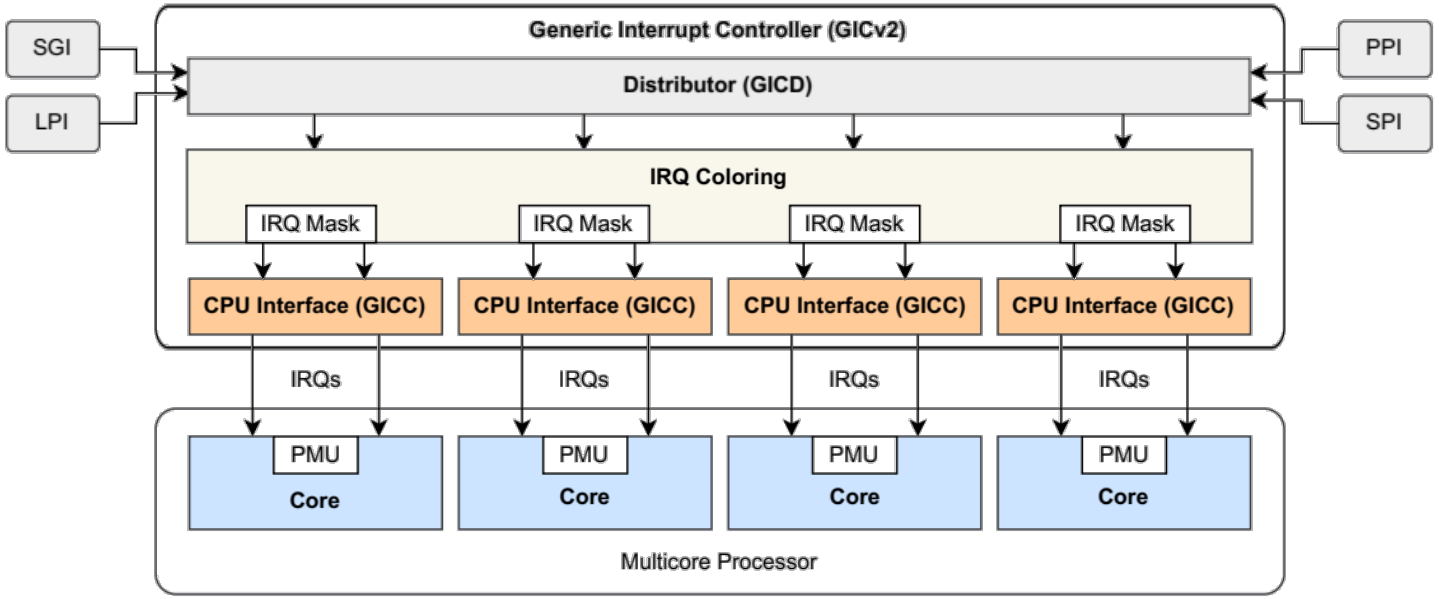
- Distributes IRQs to the appropriate CPU interface;
- Enables efficient handling of multiple interrupts in a multi-core system.

IRQ Coloring (IRQC)

- Selectively deactivate/defer interrupts;
- Reduce interference from non-critical VMs on critical VMs without fully suspending non-critical workloads.

Interface (GICC)

- Connects a CPU to the GICD, allowing the CPU to IRQs from the distributor;
- Enables the communication with the GICD to acknowledge and control interrupts handling.

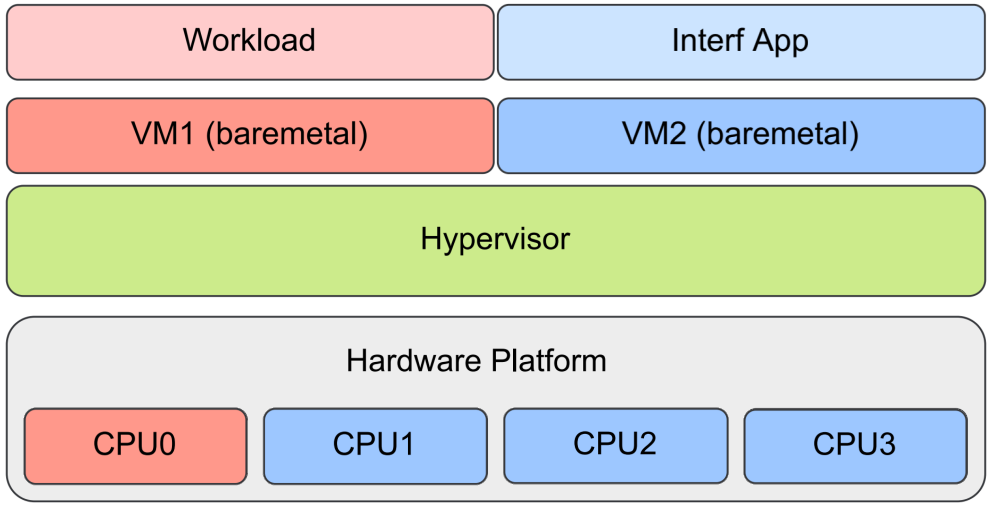


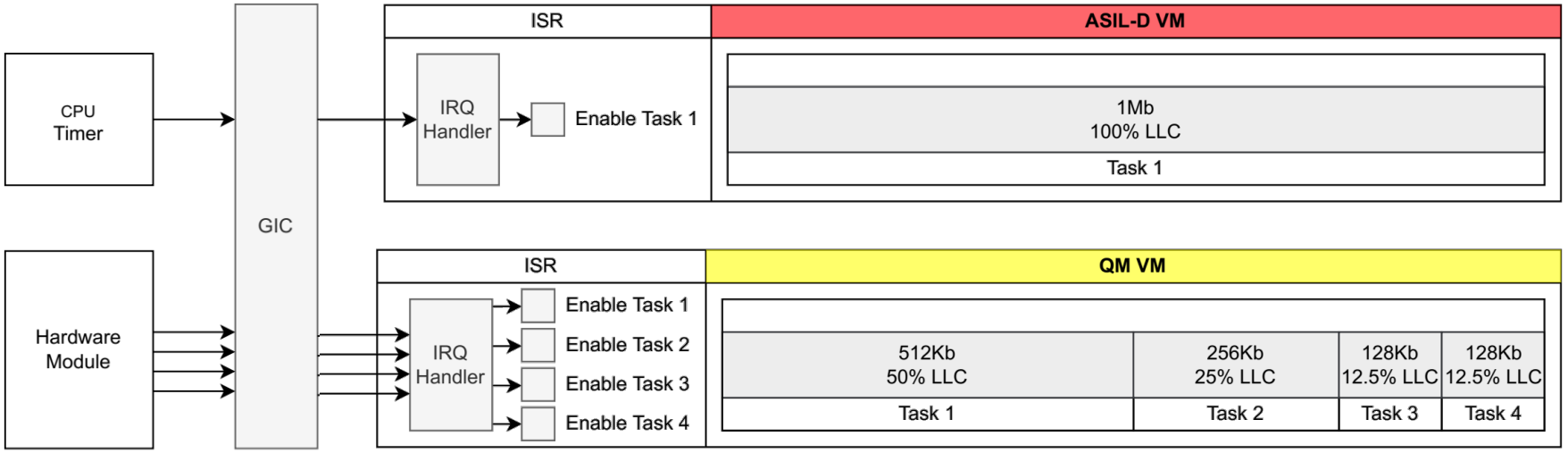
Evaluation

03

Evaluation – Proof-of-Concept

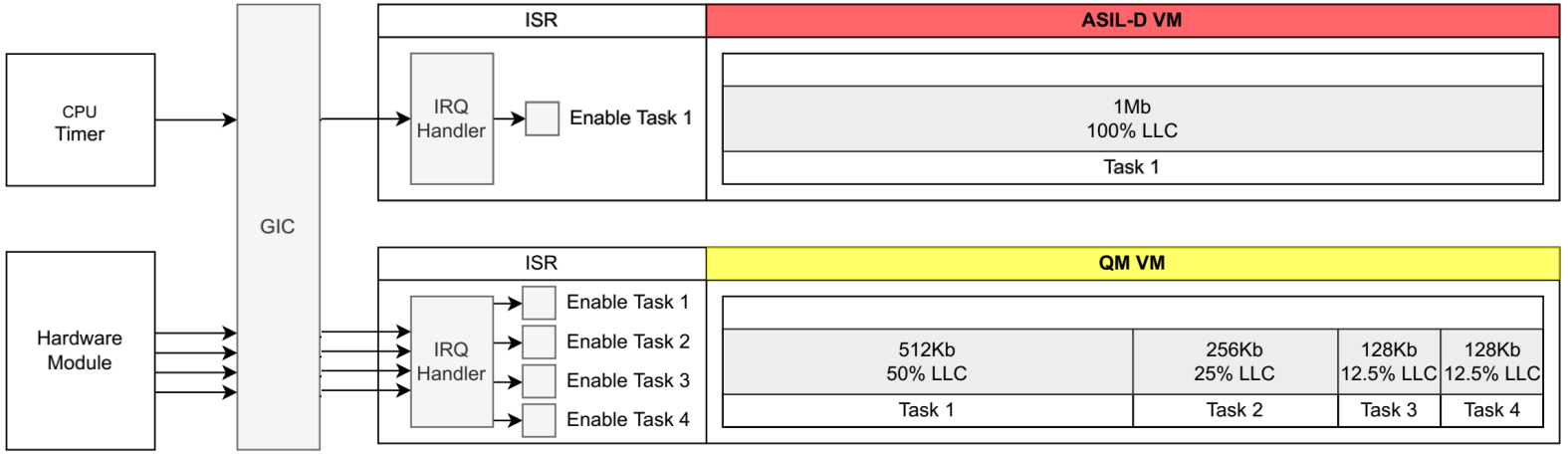
- **Experimental Setup**
  - VM1 - Critical VM (ASIL-D)
    - 1 CPU
  - VM2 - Interf VM (QM);
    - 3 CPUs
    - Budgets: DM0 (1000), DM1 (4000), DM2 (5000), DM3 (5000)
    - Sampling period: 150ms

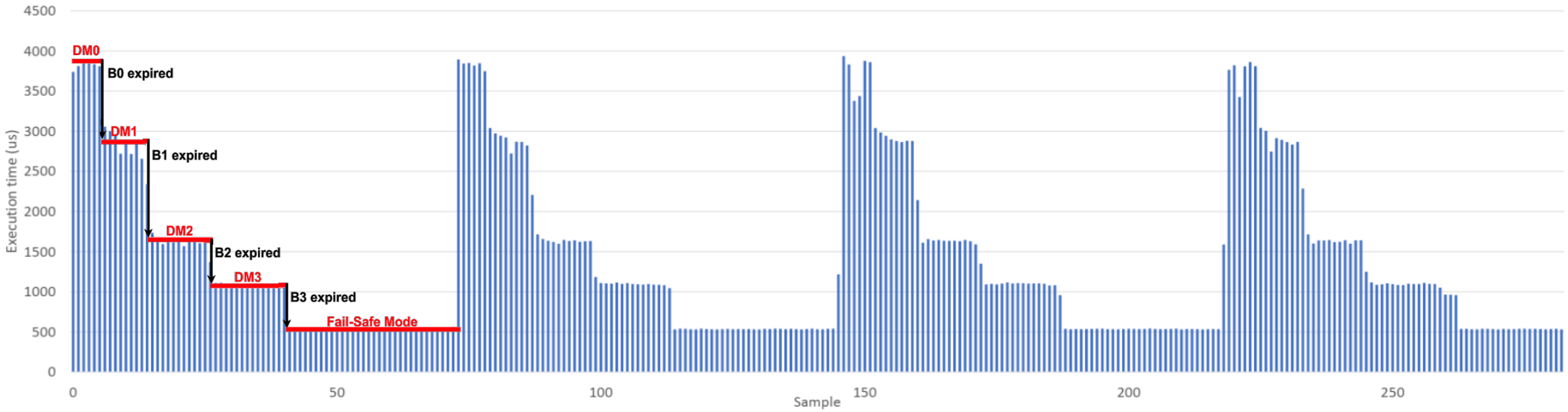




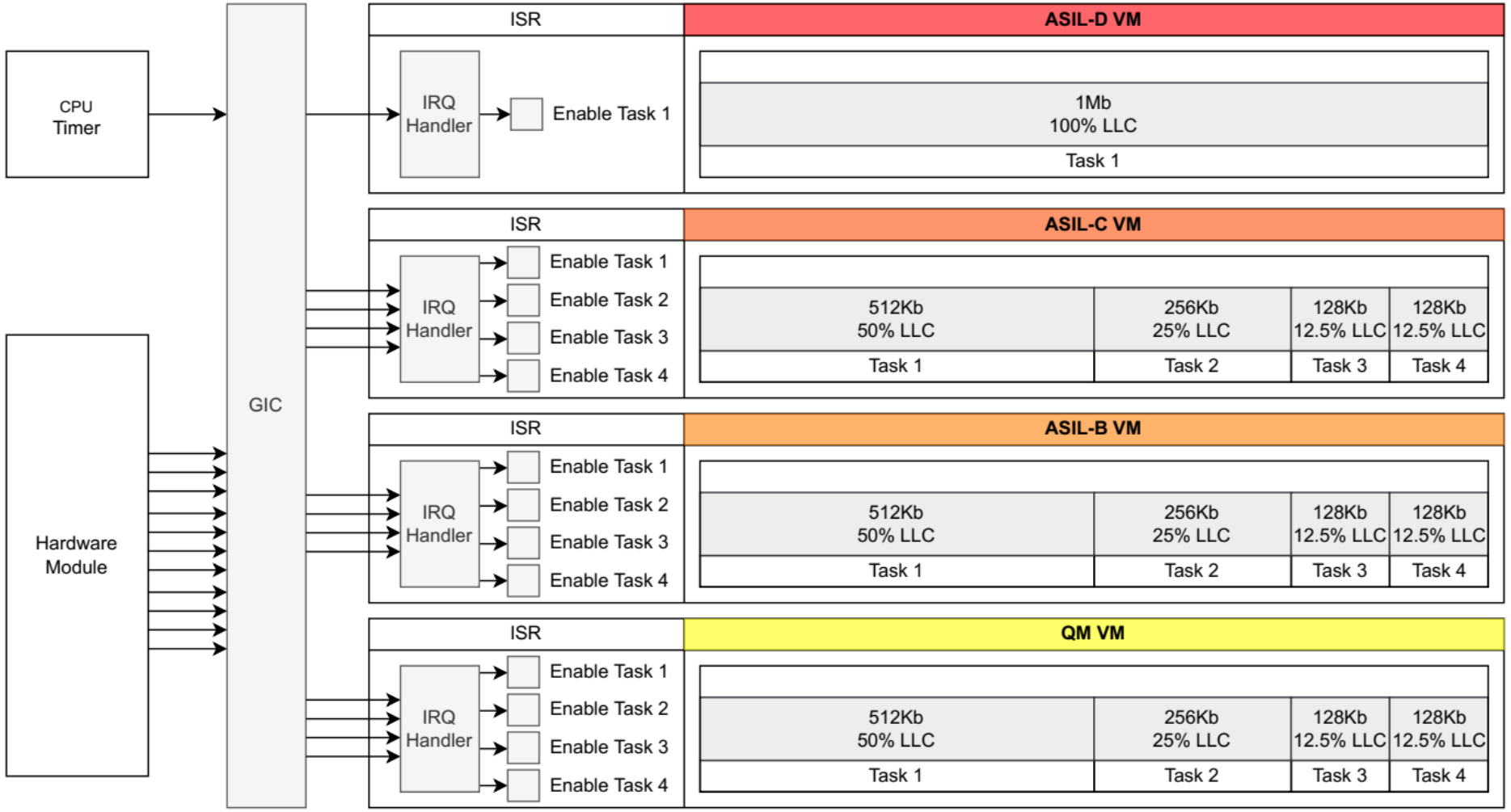
Evaluation – Proof-of-Concept

- **Experimental Setup**
  - Budgets
    - DM0 (1000)
    - DM1 (4000)
    - DM2 (5000)
    - DM3 (5000)
  - Sampling Period:
    - 150ms

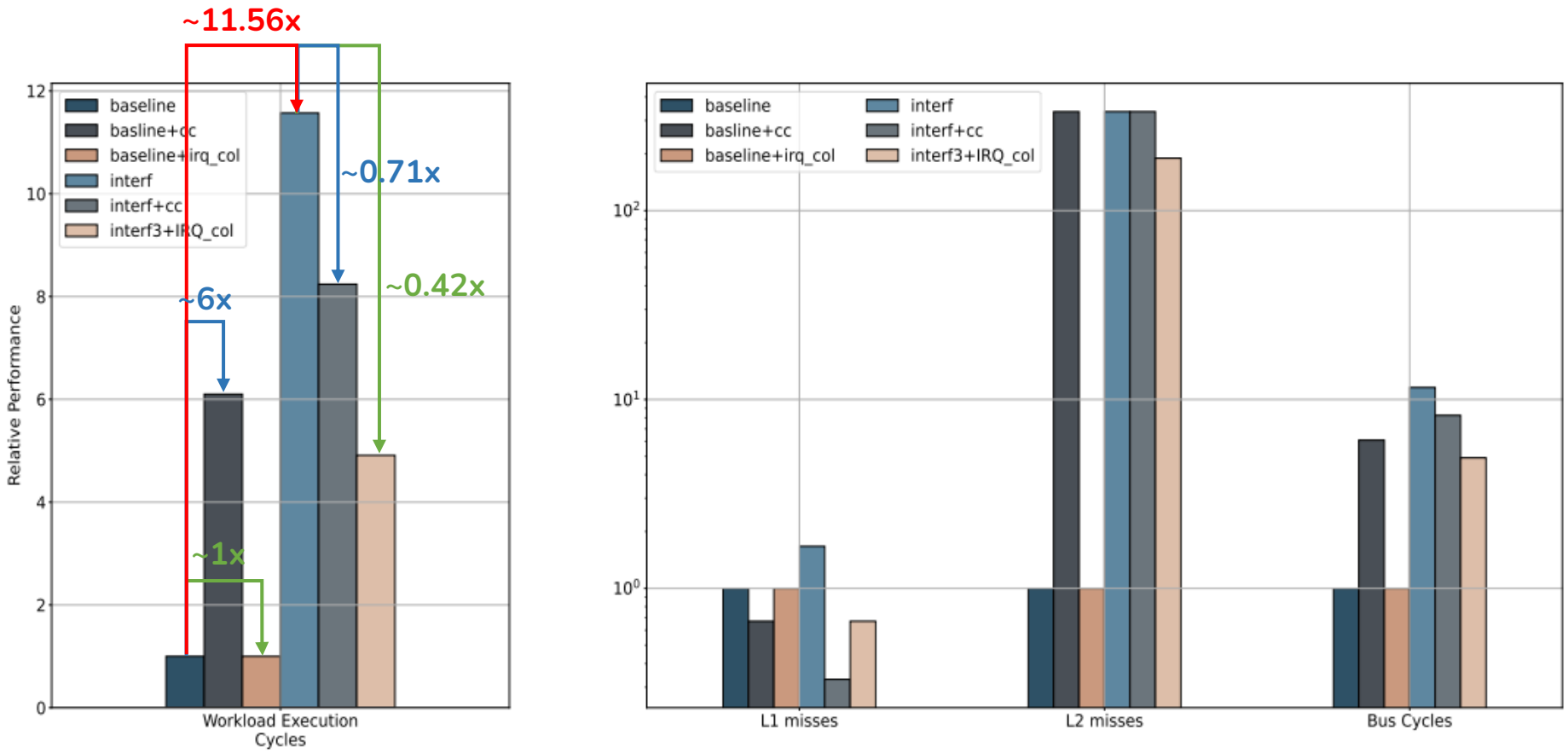




Evaluation – Use-Case Setup

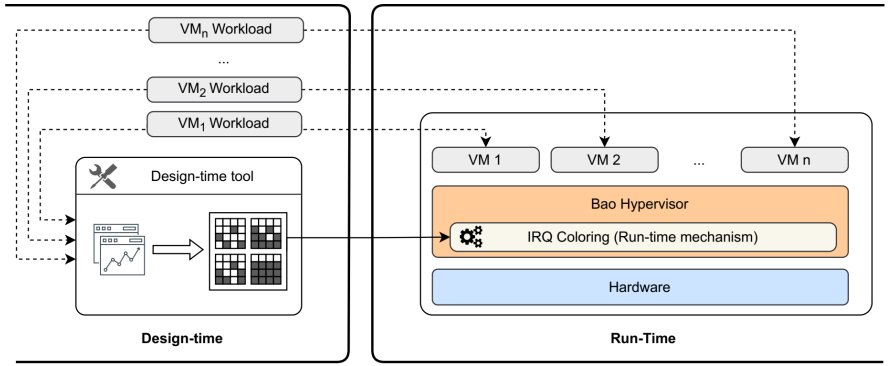


Evaluation – Use-Case Results



IRQ Coloring Roadmap

Roadmap



01

Improve Run-Time Mechanism

- Add QoS awareness (global system performance)
  - Evaluate different ASILs QoS
  - Prioritize higher-criticality ASILs

02

Development of the Design-Time Tool

03

More Compreensive Evaluation

- Consider state-of-the-art interference mitigation techniques evaluation process
- Extend the evaluation use case to more realistic scenarios

THANK YOU!

ANY QUESTIONS?

Diogo Costa
diogocostaes21@gmail.com