Xen Hypervisor:
Progress on safety certifiability

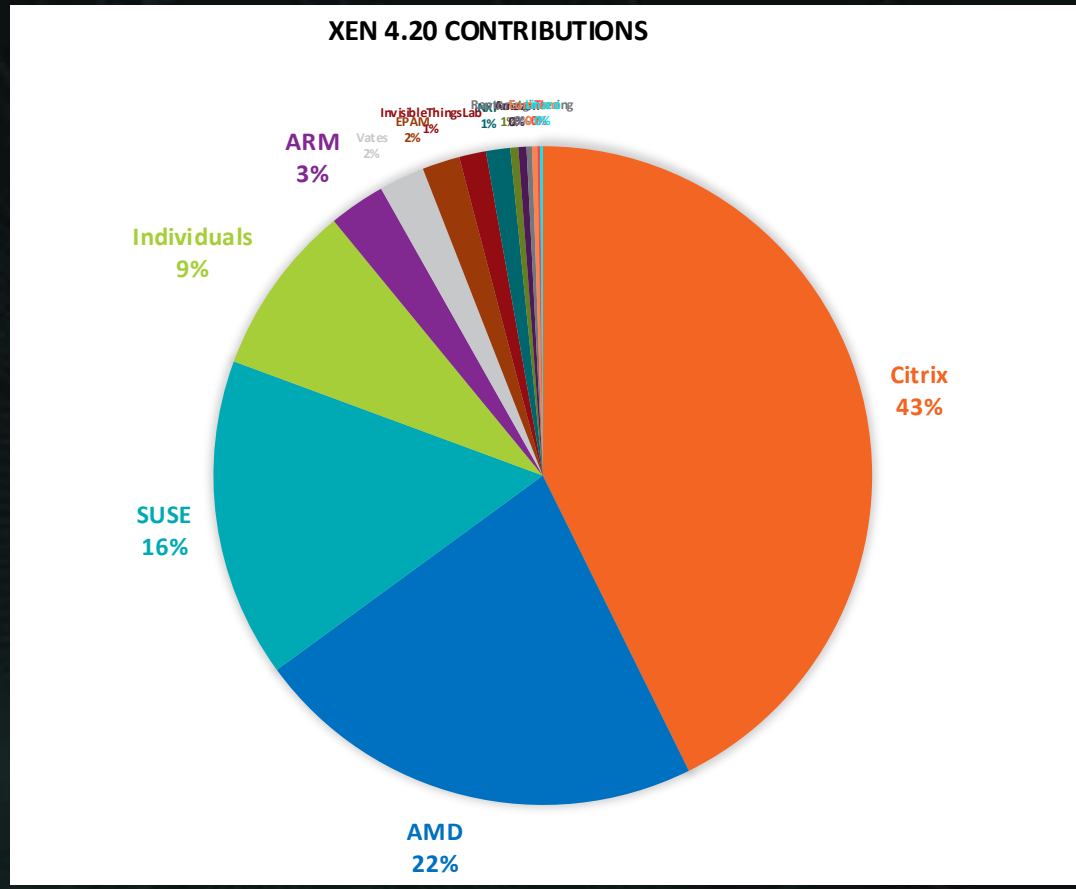
Stefano Stabellini & Edgar E. Iglesias

May 15th, 2025

AMD

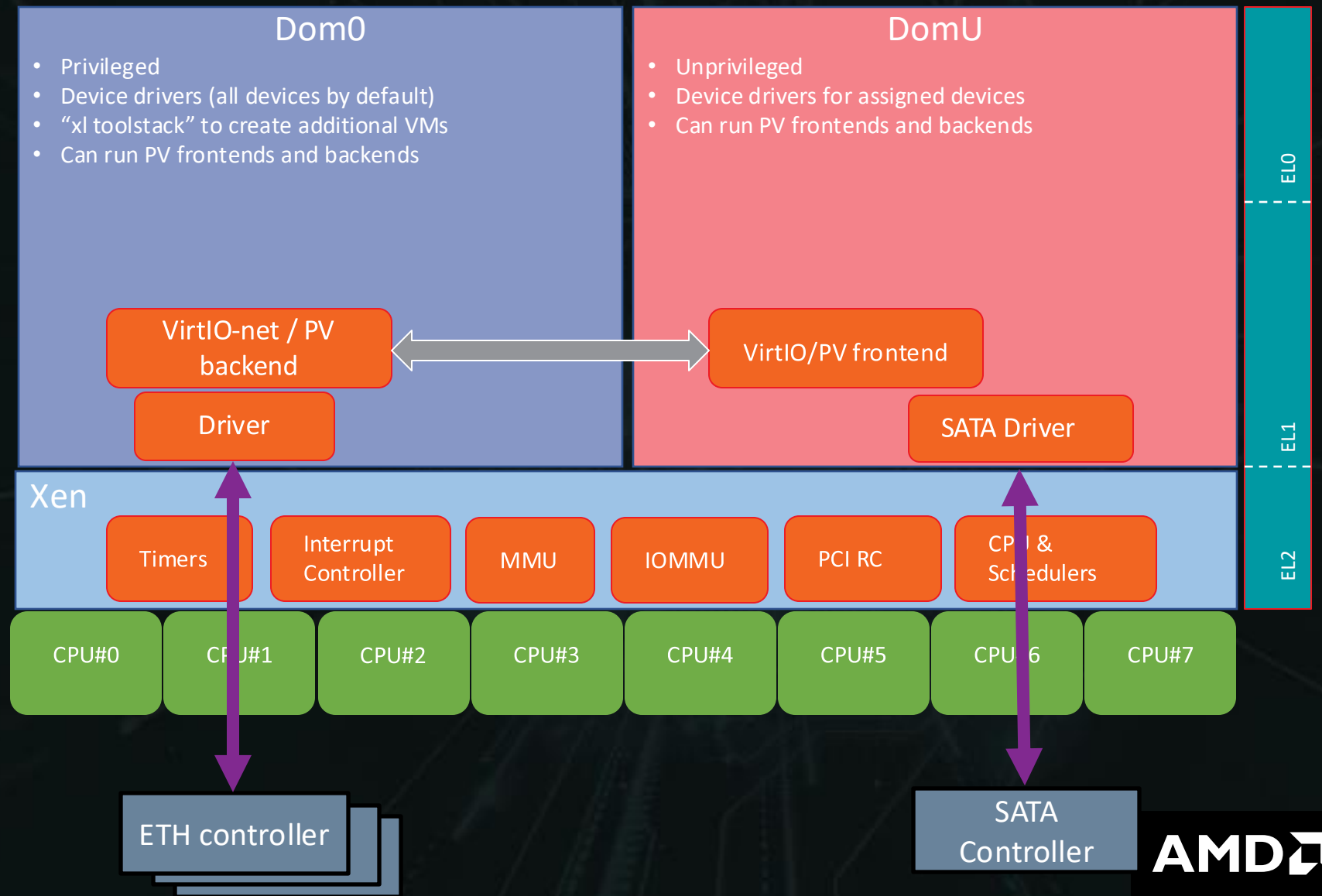
XEN: OPEN SOURCE COMMUNITY

- Xen Project is an **Open Source project under the Linux Foundation**
  - Well known and widely used in the industry
  - Extremely **strong review process and security process**
  - Reference Open Source hypervisor for Embedded and Automotive
- x86 and ARM supported
- RISC-V in progress by ResilTech, Vates, Microchip and RT-RK
- The Xen Community is a **diverse multi-vendor community**
  - Maintainers from Amazon, ARM, Citrix, AMD, SuSE, and more
  - Independent panel of experts
- AMD joined Xen Project as a member in 2022
  - AMD is the second contribution to Xen Project!
- **Honda joined Xen Project in Dec 2024!**
  - Ford and Daimler Truck joining the Xen development community



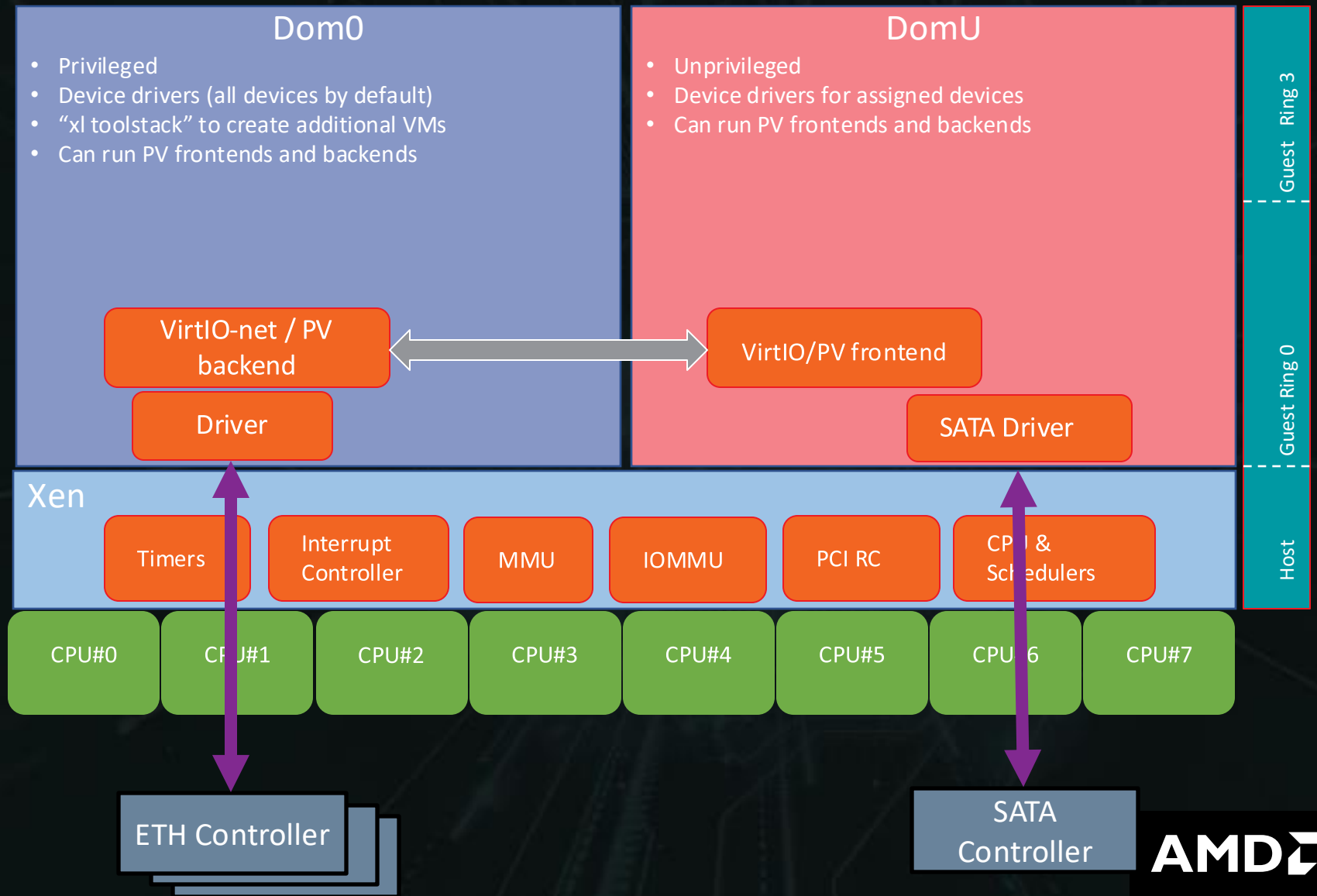
MODERN XEN ARCHITECTURE, ARMV8 (AND RISC-V)

- Xen runs alone at EL2, uses HW virtualization
- Xen only owns the key HW components:
  - CPUs, Generic Timers, GIC, MMU
  - If present: SMMU, PCI RC
- Devices are directly assigned to Domains
  - By default, to Dom0 (if present)
  - Fully configurable
- ARM Xen Domains
  - No need for QEMU for emulation
  - No need for ”PV guests”
- Dom0less
  - Dom0 becomes optional
- Devices can be shared with a PV frontend/backend architecture
  - Both Xen PV drivers and VirtIO
  - Multiple device sharing models supported



MODERN XEN ARCHITECTURE, AMD X86

- Xen runs in its own privilege level (host mode)
- Xen owns the key HW components:
  - CPUs, Timers, Interrupts, MMU, IOMMU, PCI RC
  - Uses AMD SVM
- Devices are directly assigned to Domains
  - By default, to Dom0
  - Fully configurable
- PVH only
  - No need for QEMU for emulation
  - No need for "PV guests"
  - QEMU only for VirtIO backends
- Dom0less (Hyperlaunch)
  - Dom0 becomes optional
- Devices can be shared with a PV frontend/backend architecture
  - Both Xen PV drivers and VirtIO
  - Multiple device sharing models supported



XEN: THE FULL SPECTRUM, FROM IVI TO VISION HUB

In-Vehicle Infotainment (IVI)

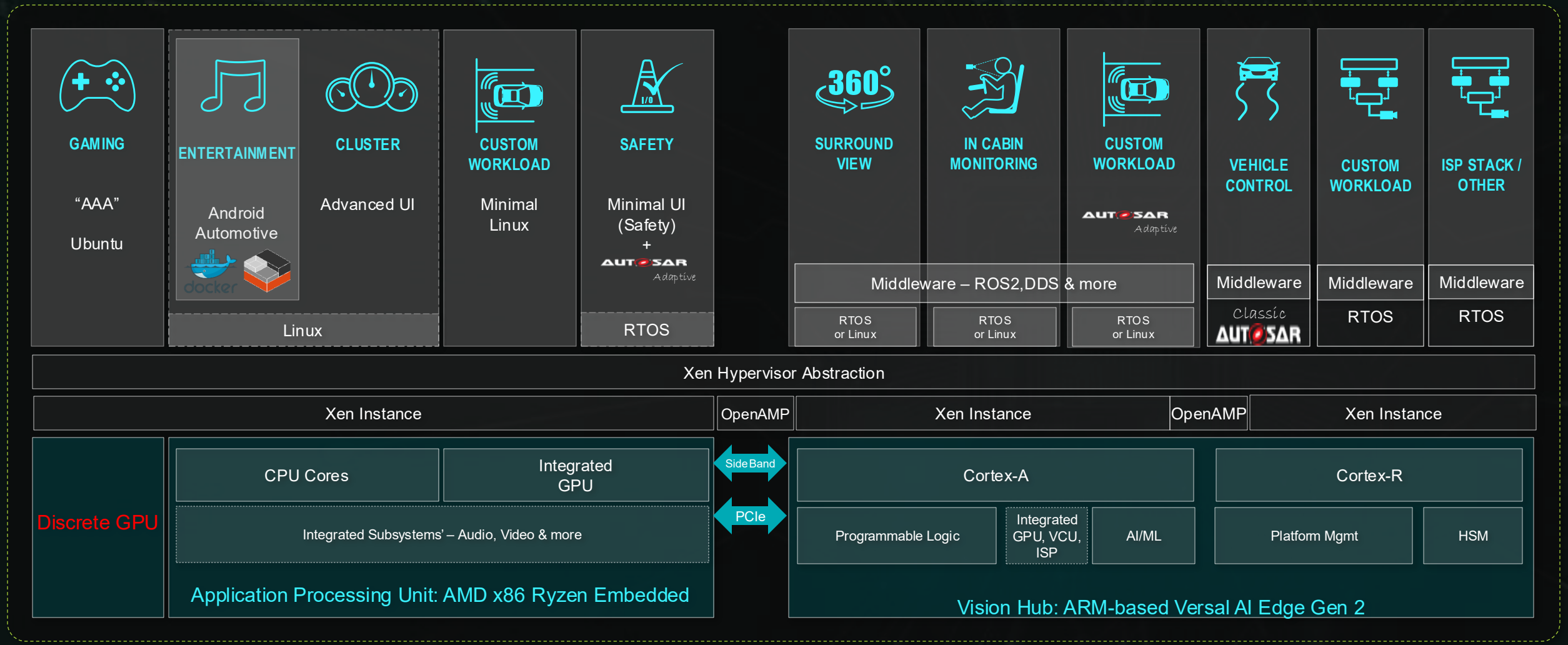
Graphics
Virtio & PV devices
Higher Complexity
Lower Safety Level

Real-time
Static Partitioning
Lower Complexity
Higher Safety Level

Vision Hub

AMD

XEN: FROM IVI TO VISION HUB, AN EXAMPLE



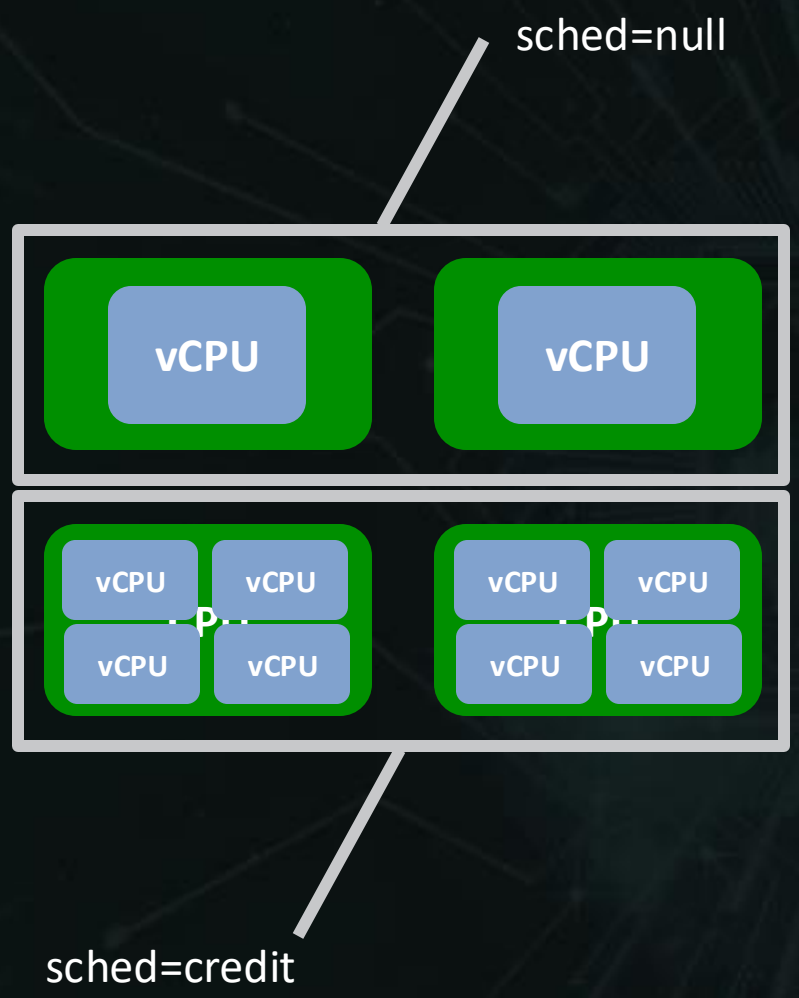
Static Partitioning

XEN SCHEDULERS

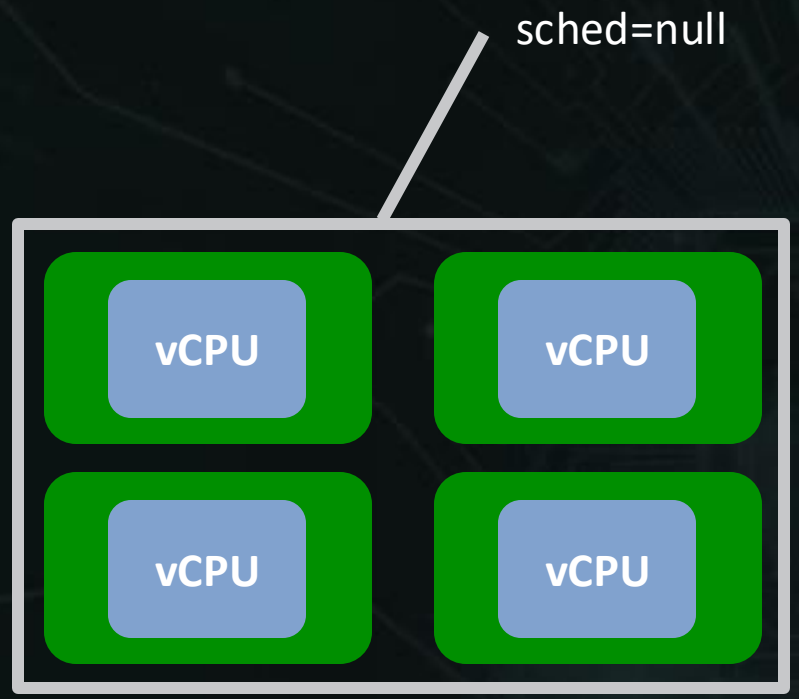
Xen allows you to divide your physical CPU cores into multiple CPU pools, each with its own scheduler.

For example, you could choose the null scheduler for one CPU pool to fully dedicate physical CPUs to virtual CPUs, optimizing for the best latency. Meanwhile, in another pool, you can use a general purpose scheduler.

This flexibility enables you to reach your consolidation targets while providing the best latency for real-time workloads.

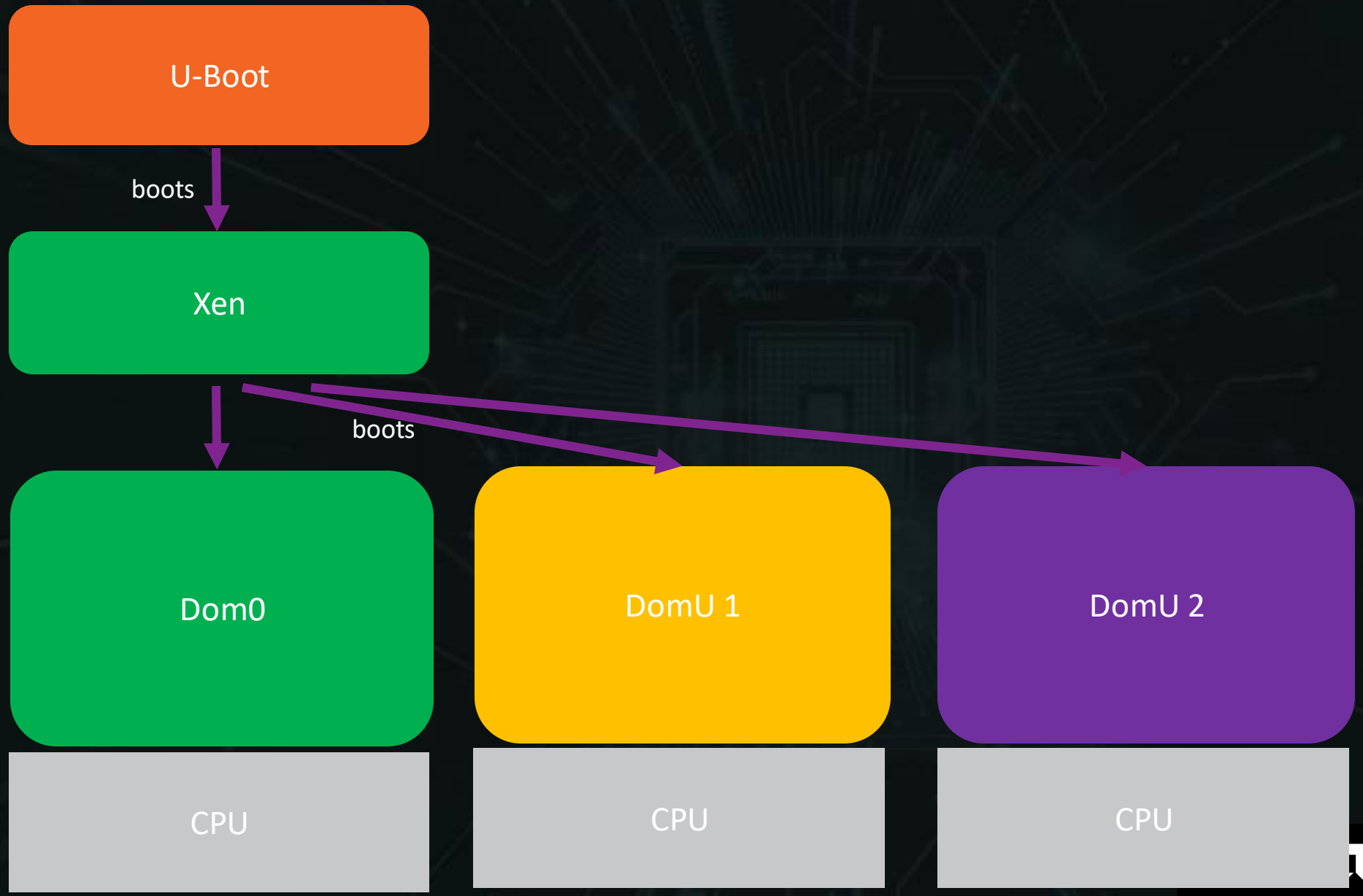


XEN SCHEDULERS: STATIC PARTITIONING CONFIGURATION



DOM0LESS BOOT

- Create all the VMs in parallel at boot
- Much faster boot times
- Dom0 is not needed anymore to create VMs
- Dom0 is optional and can be removed



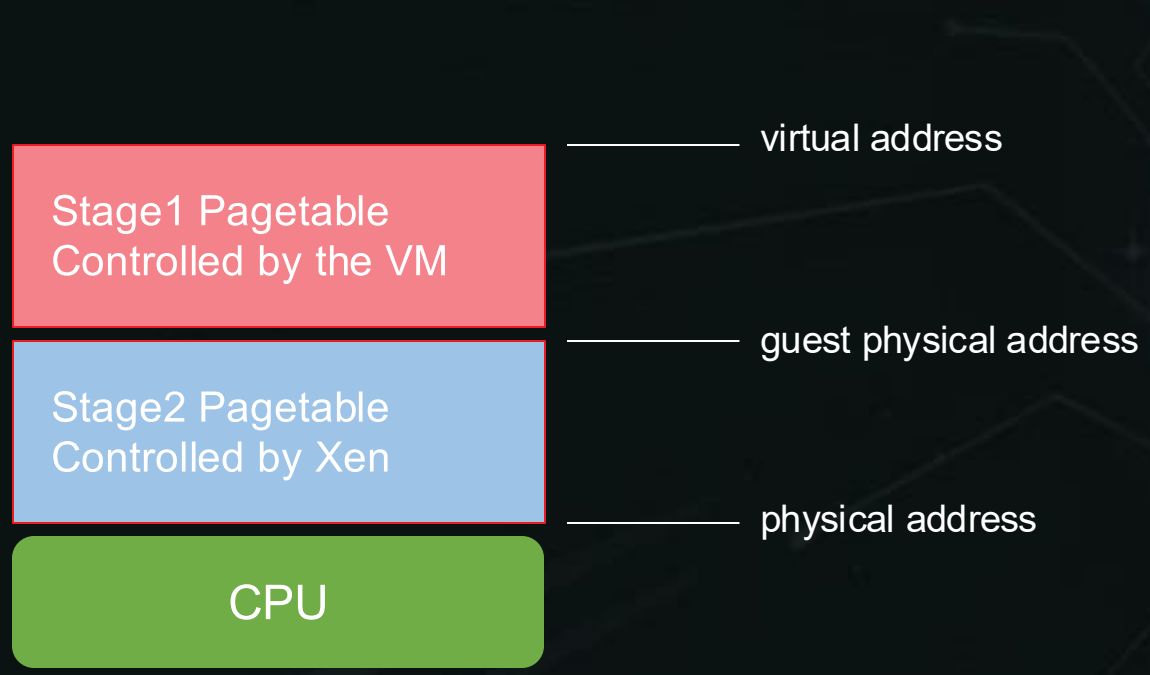
PURE STATIC PARTITIONING

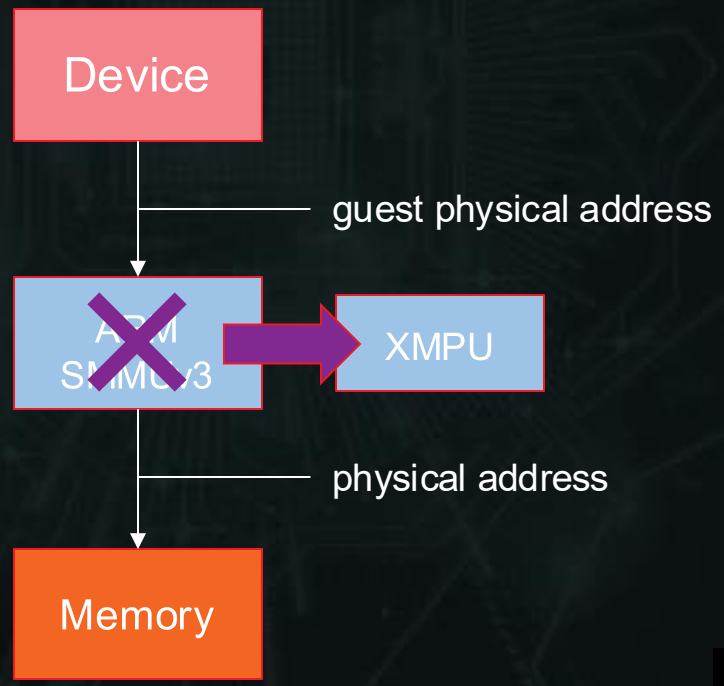
- **Dom0less**: everything defined statically before boot
  - VM memory, including memory ranges
  - Number of vCPUs
  - Device assignment
- **Null** scheduler: only static vCPU-pCPU assignment
- No Dom0, no privileged VMs
- No hypercalls
- Minimal Xen footprint at runtime
  - Only: virtual timer, virtual UART, virtual interrupt controller
- **IOMMU** is not a hard requirement
  - MMIO and memory can be 1:1 mapped (on ARM)
  - XMPU/AXI Filters/TMR for protection
- Hypervisor mode supported: Xen now running on Cortex-R52 and Cortex-R82 (no MMU!)


```
NUM_DOMUS=2
DOMU_KERNEL[0]="linuxrt"
DOMU_RAMDISK[0]="initrd.cpio"
DOMU_VCPUS[0]=2
DOMU_STATIC_MEM[0]="0x100000 0x60000000"
DOMU_PASSTHROUGH_PATHS[0]="/axi/ethernet@ff0e0000"
DOMU_KERNEL[1]="zephyr.bin"
DOMU_STATIC_MEM[1]="0x0 0x100000"
DOMU_PASSTHROUGH_PATHS[1]="/axi/serial@ff000000"
```

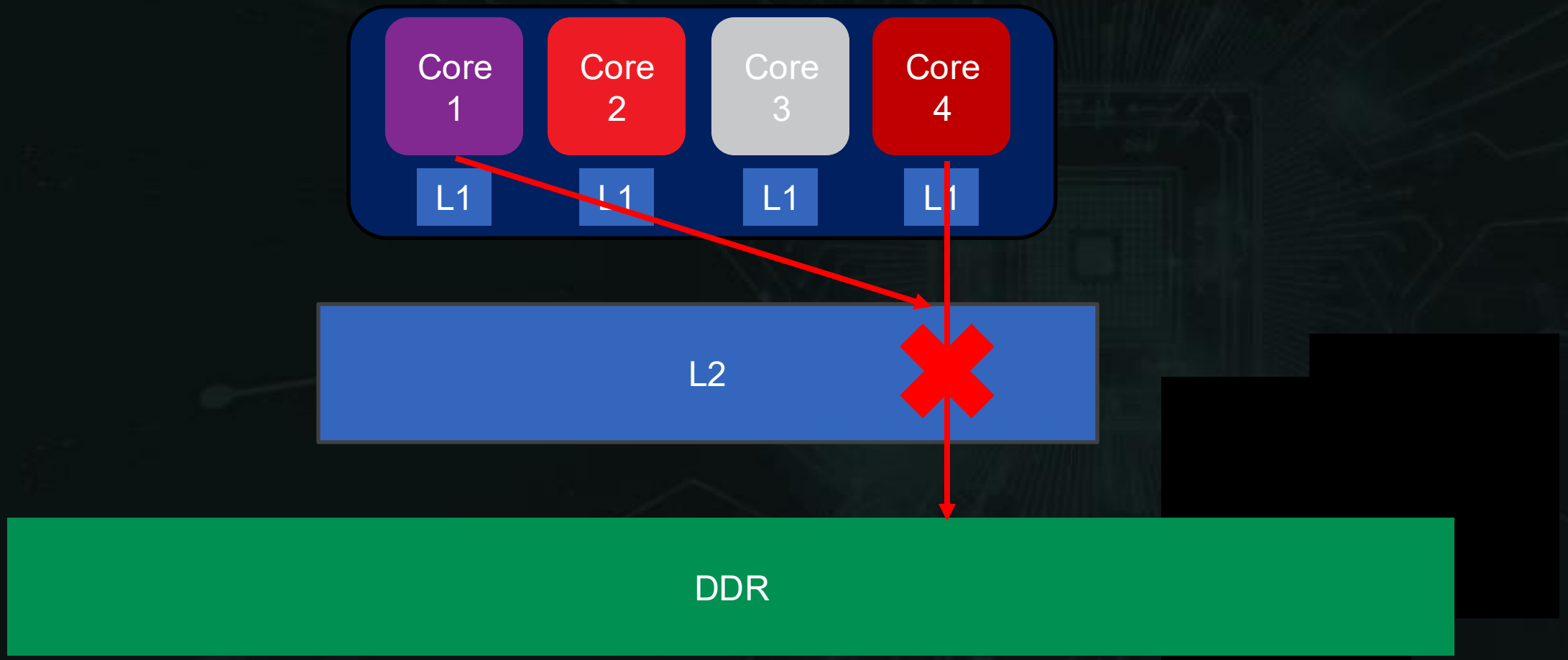
VM 1:1 MEMORY MAPPING

- Guest physical addresses == Physical addresses
- ARM SMMU is not required for DMA address translations any longer
- Another protection mechanism is required to ensure isolation between VMs, e.g. Xilinx XMPU and XPPU





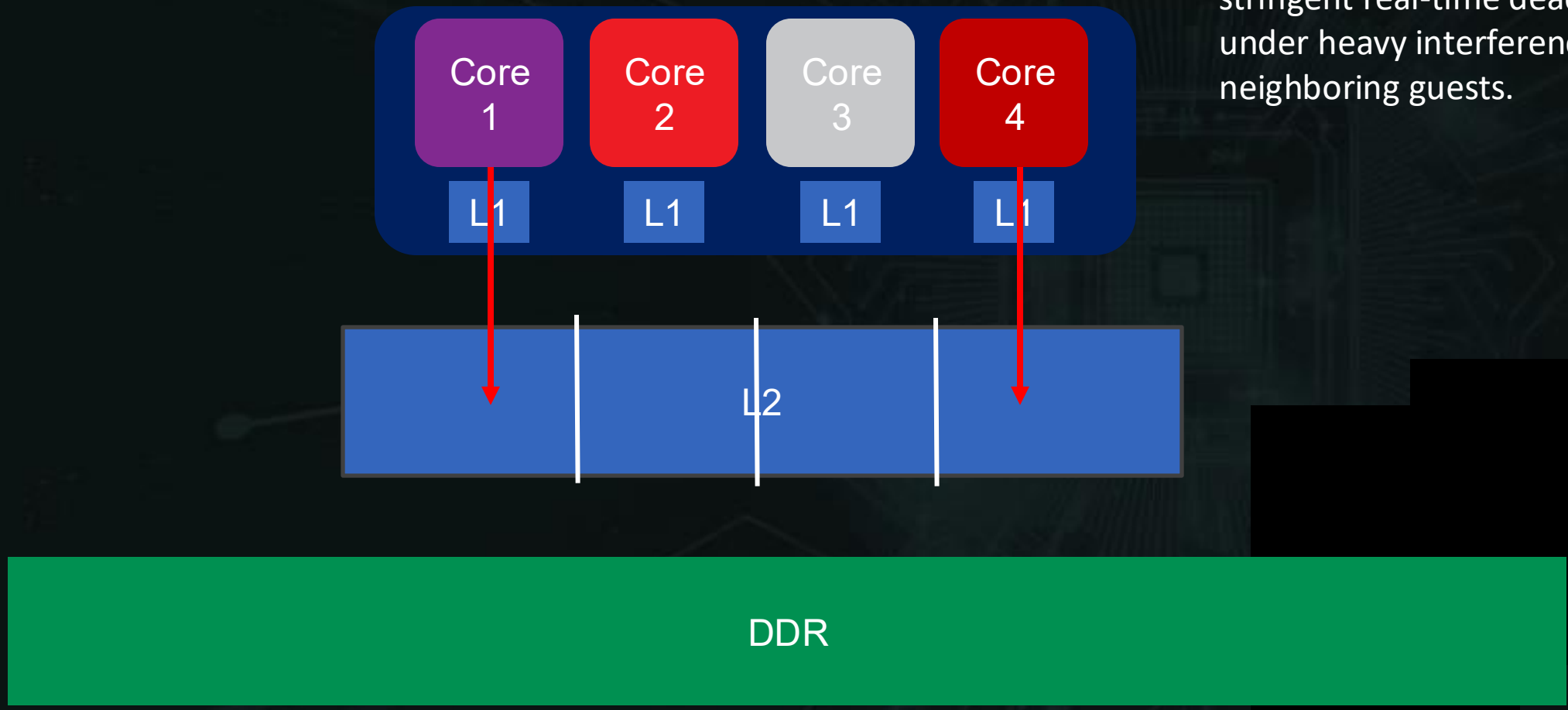
REAL-TIME – CACHE COLORING



REAL-TIME – CACHE COLORING

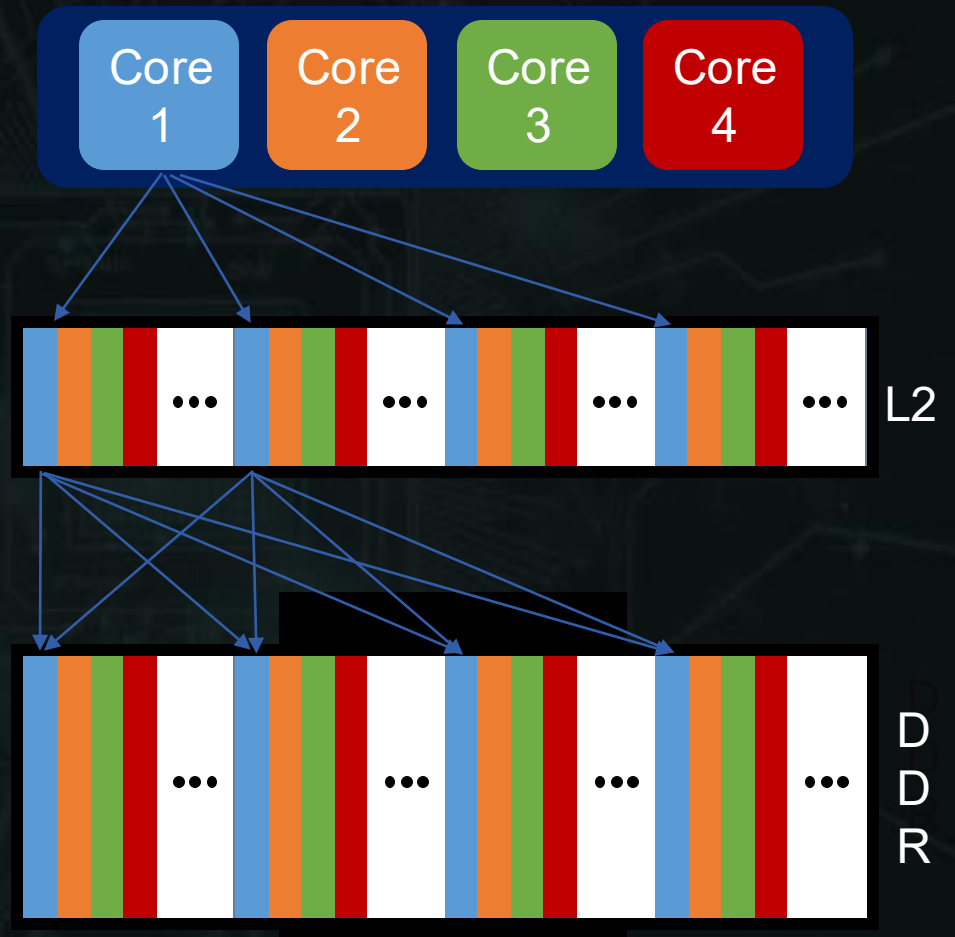
Cache Coloring: cache partitioning in software to **zero the effects of cache interference**.

With Cache Coloring you can meet stringent real-time deadlines even under heavy interference from neighboring guests.



REAL-TIME – CACHE COLORING

- CPUs clusters often share L2 cache
- Interference via L2 cache affects performance
  - App0 running on CPU0 can cause cache entries evictions, which affect App1 running on CPU1
  - App1 running on CPU1 could miss a deadline due to App0's behavior
  - It can happen between Apps running on the same OS & between VMs on the same hypervisor
- Hypervisor Solution: Cache Partitioning, AKA **Cache Coloring**
  - Each VM gets its own allocation of cache entries
  - No shared cache entries between VMs
  - Allows real-time apps to run with deterministic IRQ latency
  - **3us IRQ latency under heavy interference**



RESULTS

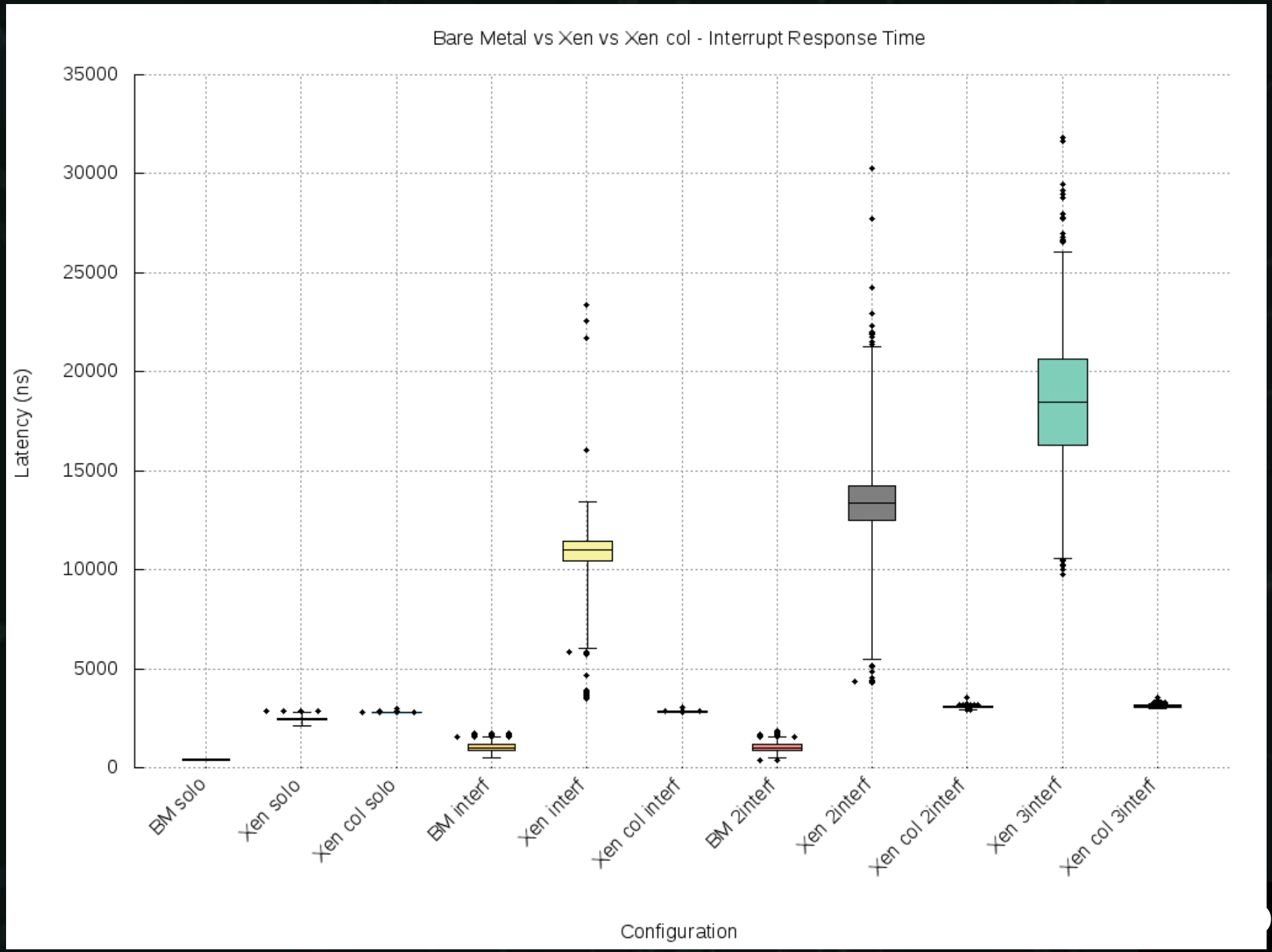
IRQ latency measurements

Xilinx Ultrascale+ Cortex-A53

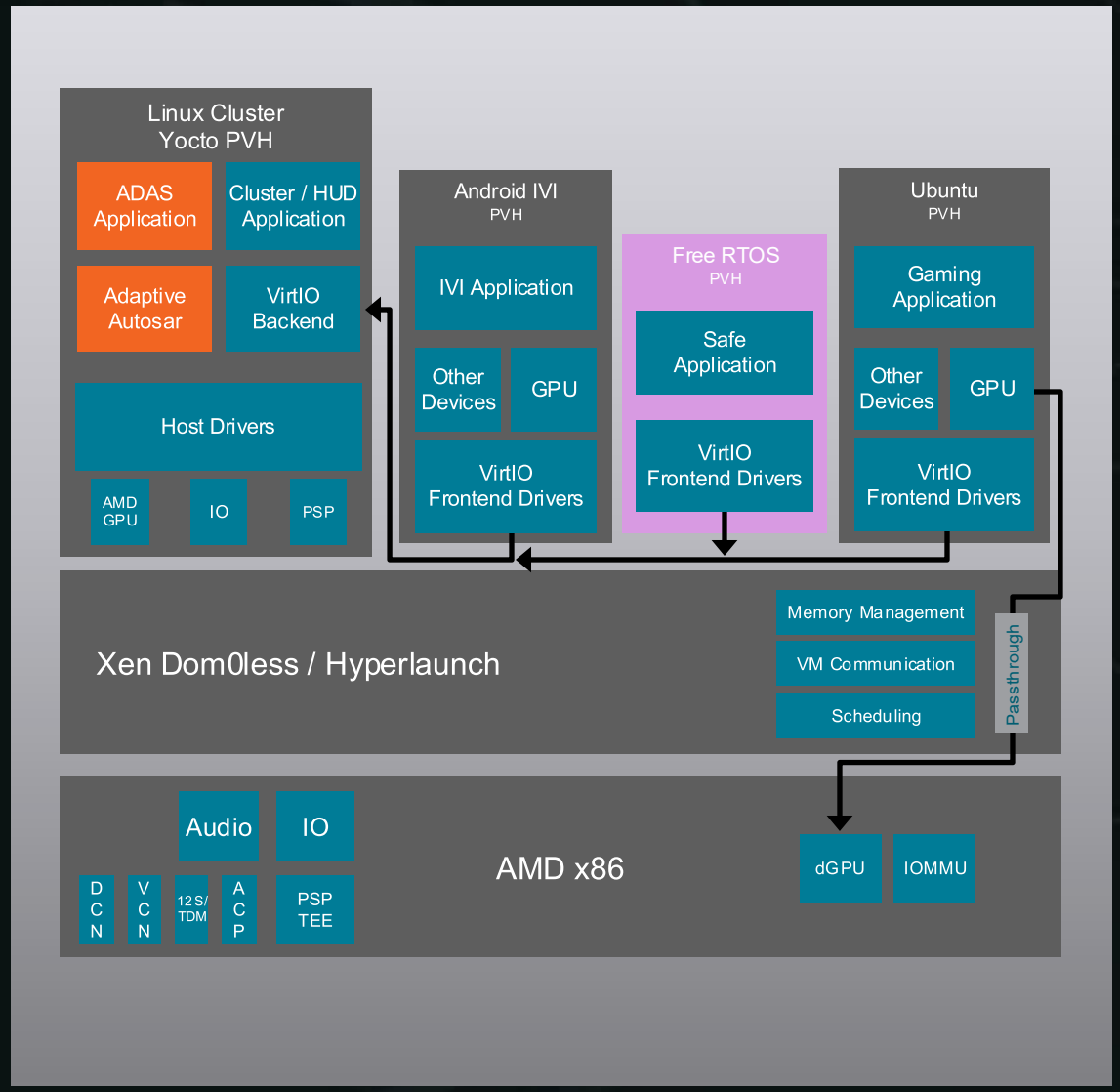
Xen Coloring provides stable IRQ latency even under severe levels of interference

BM: BareMetal
Xen: Xen without cache coloring
Xen col: Xen with cache coloring

solo: no interference
interf: 1 interference VM
2interf: 2 interference VMs
3interf: 3 interference VMs



XEN-POWERED IVI STACK



- Dom0less (Hyperlaunch) boot
  - Critical for short boot times
  - No Dom0, only a limited-privileged Linux hardware domain
- All Domains are PVH
  - Better VM boot times
  - Direct kernel boot possible
- Much smaller QEMU machine for VirtIO backends only
  - QEMU machine common between ARM and x86
- FreeRTOS and other RTOSes available to run critical apps
- Xen PV Drivers:
  - 1x PV audio device
- VirtIO:
  - 1x VirtIO-net
  - 1x VirtIO-block
  - 1x VirtIO-gpu device
  - 1x VirtIO-tee device
  - 1x VirtIO-gpio device
  - 4x VirtIO-i2c device
  - 1x VirtIO-input

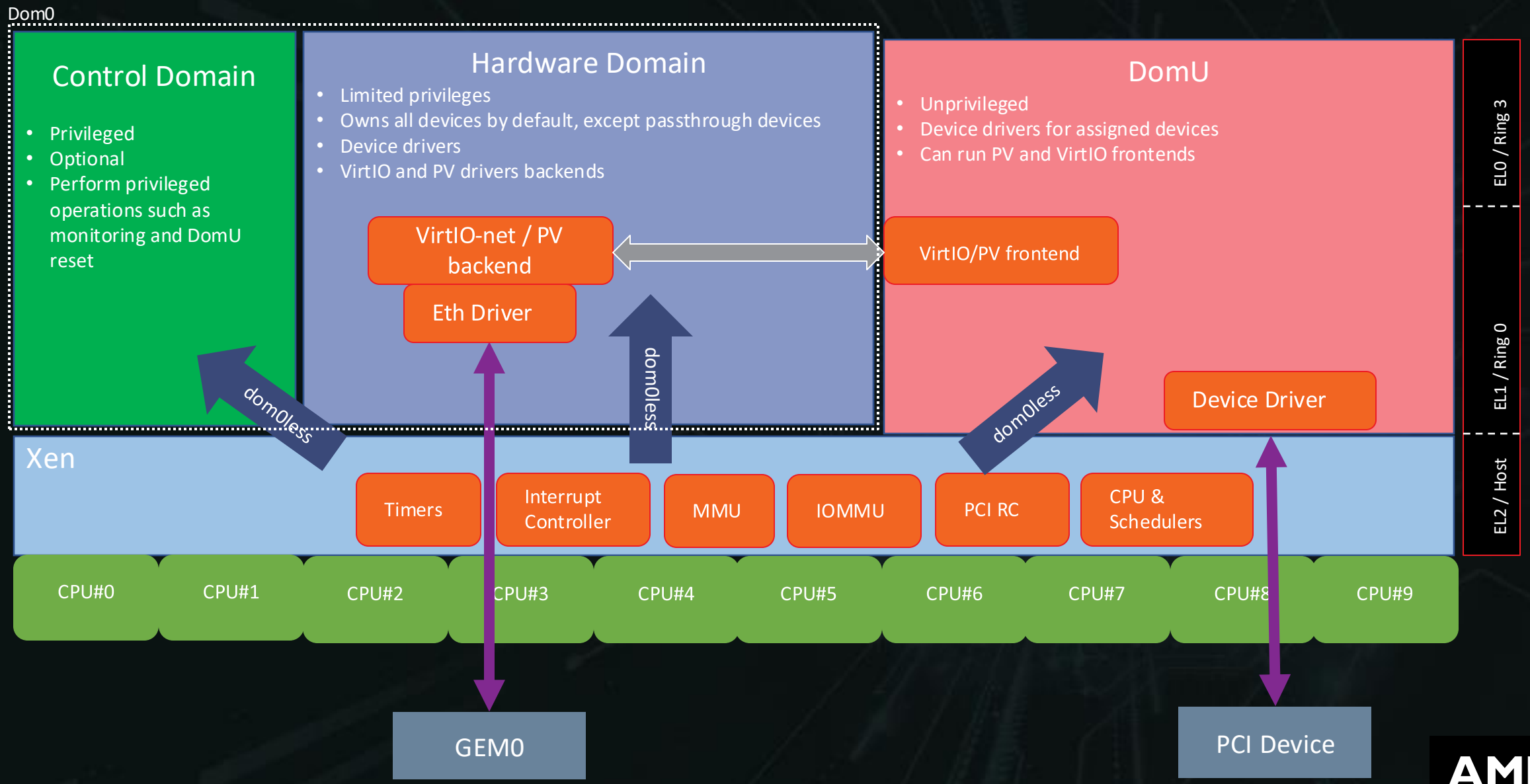
SAFETY CERTIFYING XEN HYPERVISOR

- **Xen is the Open-Source reference hypervisor for embedded and automotive at AMD**
  - AMD has an in-house engineering team to develop, enhance, and support Xen for embedded and automotive
    - Ayan Kumar Halder, Edgar Iglesias, Jason Andryuk, Luca Miccio, Michal Orzel, Stewart Hildebrand, Victor Lira, Xenia Ragiadakou
  - Xen is delivered to customers today as reference and is supported by Forum, Premium Technical support, and Engineering
  - Xen in production across multiple verticals

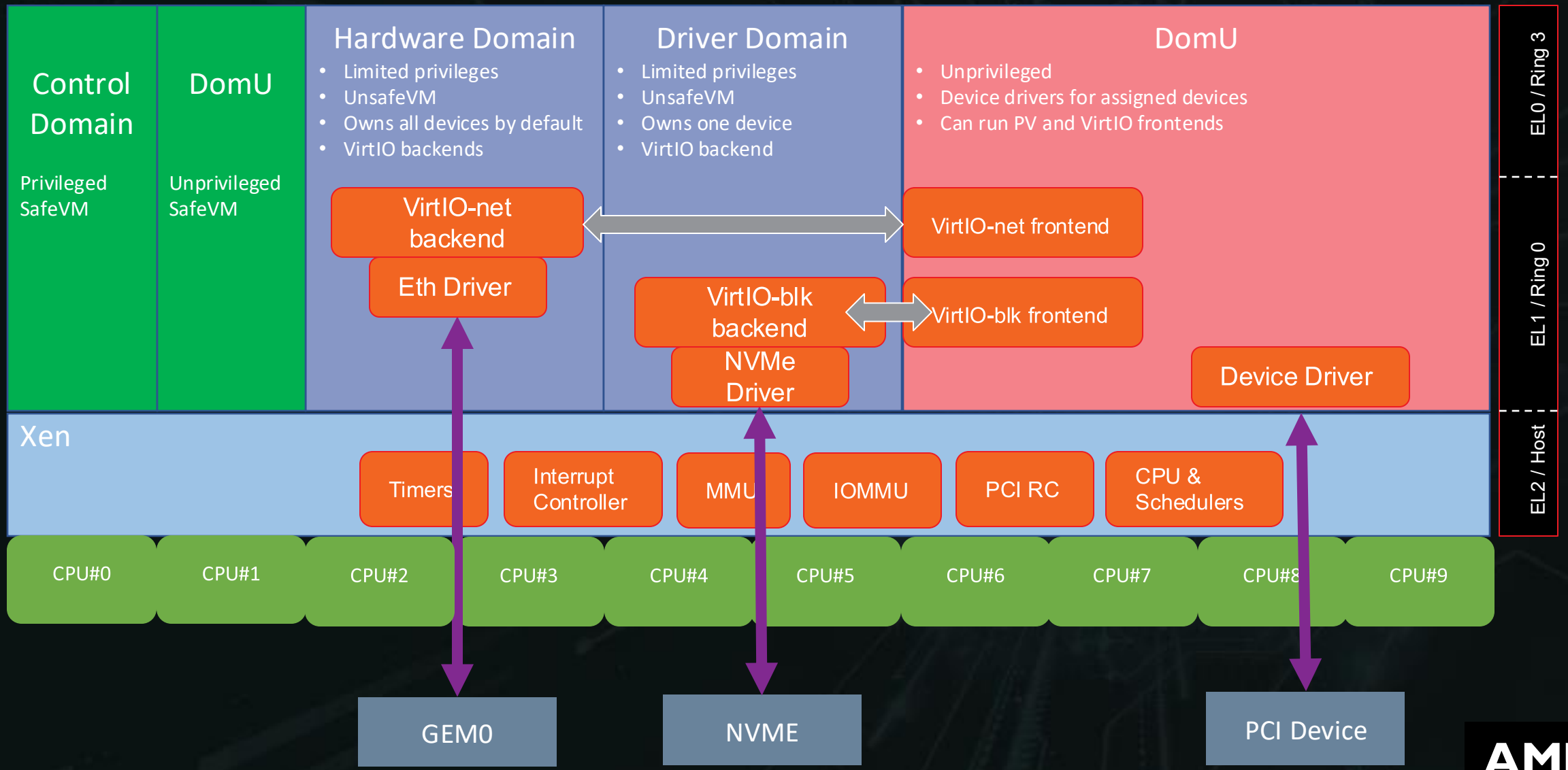
- **AMD is working on making Xen safety-certifiable**
  - **IEC 61508 SIL 3 & ISO 26262 ASIL D**
  - **Strictest level of safety for automotive**
  - **Certification based on Xen upstream community processes and upstream codebase**
  - AMD platforms both ARM and AMD x86
  - Not working with a private fork -- Ability to update the certification with limited efforts
  - Open to collaborations with other community members upstream

A new Xen Architecture for Safety

MODERN XEN ARCHITECTURE FOR SAFETY



ROLES AND PRIVILEGES



FEATURES IN SCOPE

- No OS/hypervisor dependencies: run (multiple) Safe VMs and QM VMs of your choice
- Common code and core components in Xen
  - UART, IOMMU, Interrupt Controller, Timer, PCI
- Static Configuration
  - Domain creation via Dom0less (no dynamic VMs)
  - Memory, vCPU, virtual devices, allocated at boot time
  - No dynamic memory management
  - Dom0less domain reboot (domain reset)
- Passthrough, both PCI and non-PCI
  - All DMA operations are protected by IOMMU
- CPUPools, null, and credit2 schedulers
- Domain Reset
  - Domain reboot, without memory reallocation
- Control Domain | Hardware Domain | DomU roles with different levels of privilege
  - Hardware Domain is Dom0 without the ability to cause damage
  - Hardware Domain can be QM
- VM-to-VM communication and Device Sharing
  - VirtIO and PV Drivers frontends/backends
  - Event channels: VM-to-VM event notifications
  - Static shared memory
  - Grant table: voluntary VM-to-VM memory sharing
  - Argo (vsock-like interface)

VIRTIO, PV DRIVERS, AND OTHER VM-TO-VM COMMUNICATION MECHANISMS

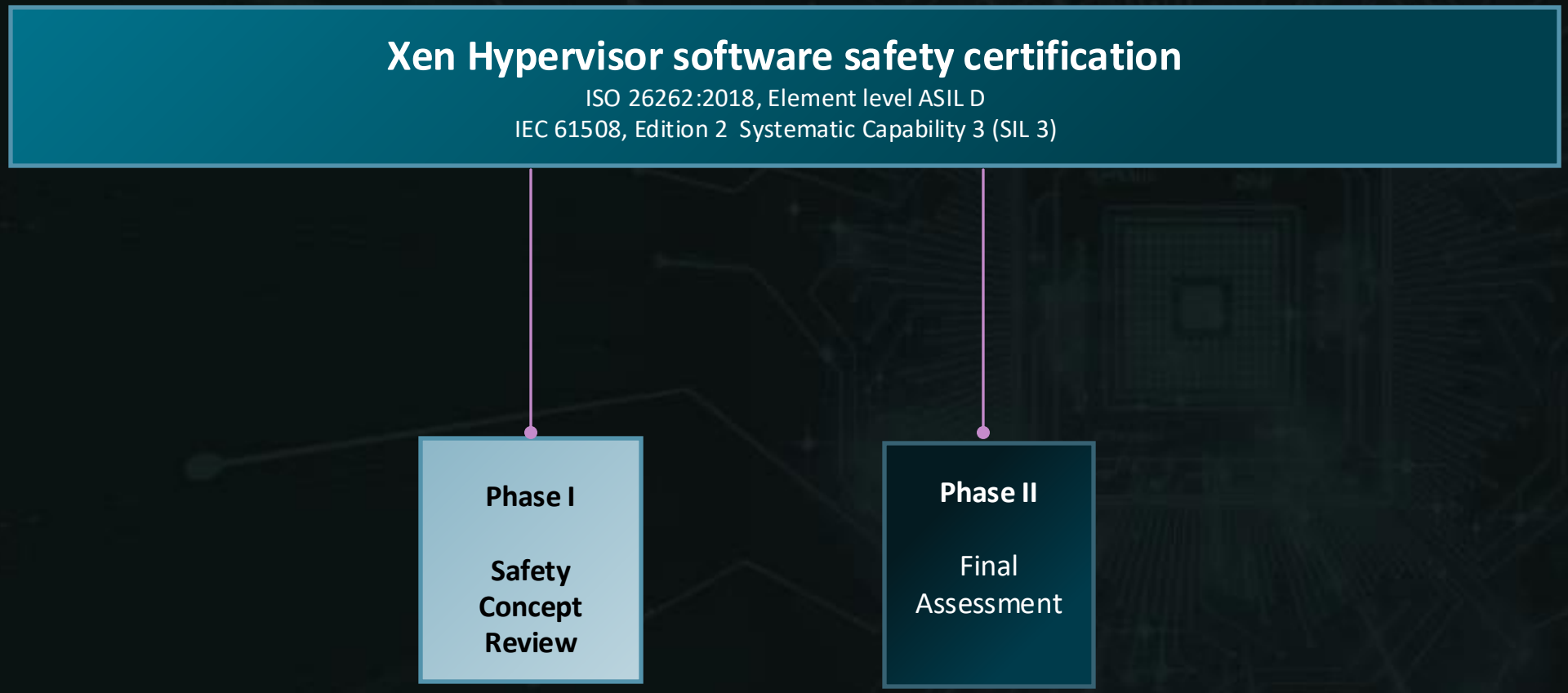
Simpler VM-to-VM communication mechanisms

- Static shared memory and event channels
- Grant table and event channels
- Argo – hypervisor-mediated memory copies
- All of these can work between Safe/Unsafe VMs

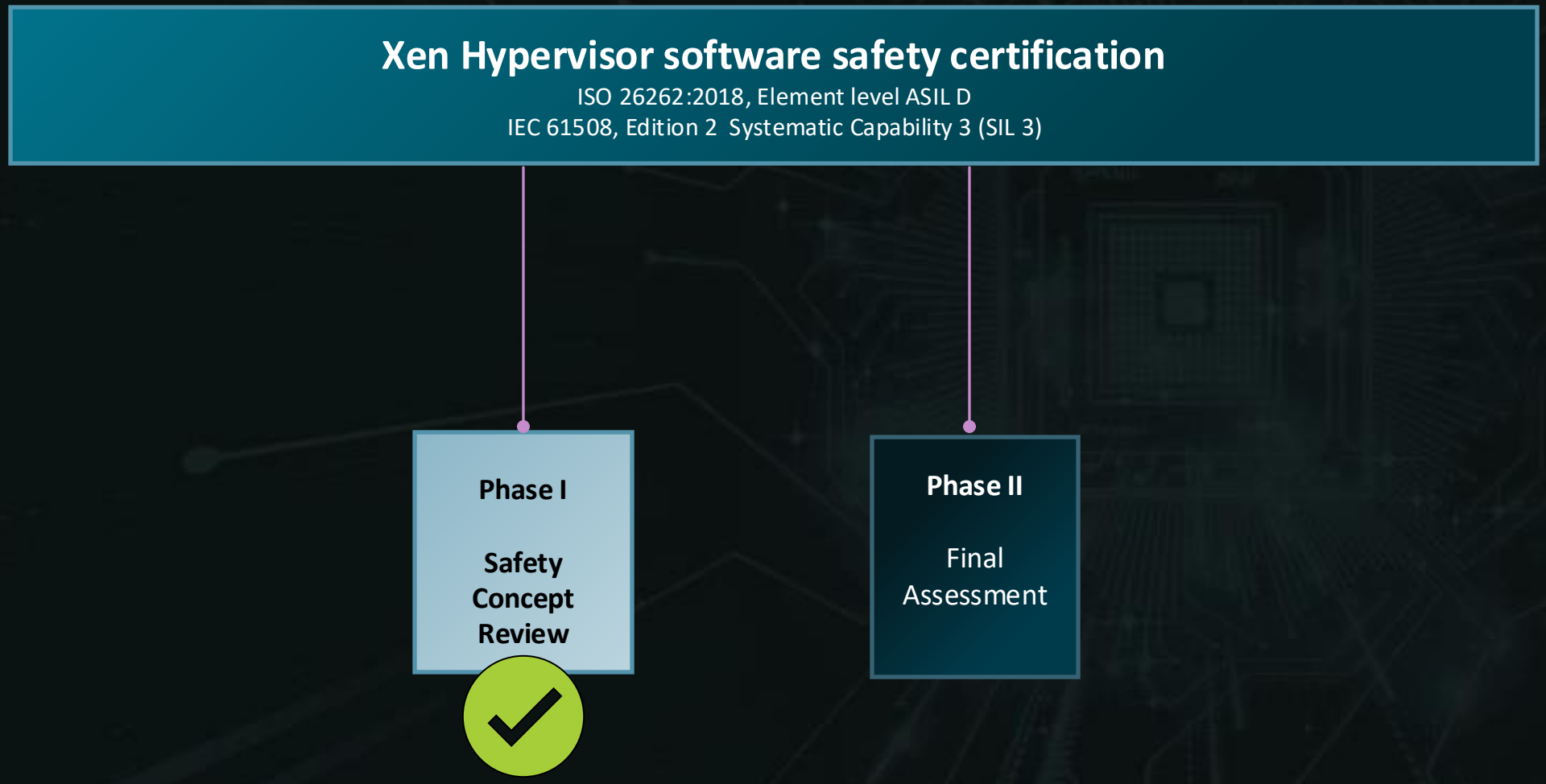
VirtIO and PV Drivers

- The building blocks in Xen are part of the safety certification scope
- Existing implementations are not Free-From-Interference
  - The ring protocol is not designed for safety
  - Potentially dangerous sync (waiting) operations
  - Potentially dangerous memory mappings
  - The frontends are particularly exposed to interference
  - It should be possible to write a Safe backend today
- Current recommendation:
  - Use VirtIO between QM VMs
  - The Hardware Domain can be QM
- We are working on two VirtIO protocol extensions that enable VirtIO between SafeVMs, including Safe frontends, but will require protocol changes

XEN HYPERVISOR SAFETY CERTIFICATION PLAN



XEN HYPERVISOR SAFETY CERTIFICATION PLAN



Completed November 2024!

THE XEN COMMUNITY AND FUNCTIONAL SAFETY

Upstreaming Activities Nearing Completion

- Code Changes to Xen & Implementation of New Features relevant to the Safety Architecture
  - PVH completion
  - Hyperlaunch/Dom0less and static configurations
  - PCI Passthrough (vPCI)
- MISRA C Compliance
  - Goal: improve the Xen codebase
  - MISRA C compliance is never at the expense of quality
    - Adopt MISRA C rules as part of the Xen coding guidelines
    - Address or deviate MISRA C violations in Xen
  - BUGSENG ECLAIR MISRA C checker integrated in the Gitlab CI-loop

Ongoing Upstreaming Activities

- Hardware Domain / Control Domain separation
- Assumptions of Use
  - Assumptions Xen relies on regarding other components
- Software Safety Requirements
  - All functional requirements of the software
  - Define scope and requirements structure
  - “market”, “product” and “software safety” requirements
  - Traceability
- Software Architecture Specification
  - All interfaces and designs aligned with the functional safety requirements

XEN SAFETY PROGRESS: MISRA C

- Preliminary tailoring resulted in the selection of **143** MISRA C rule candidates
- MISRA C rules adoption in progress:
  - **120** rules adopted and added to **`docs/misra/rules.rst`**
  - **0** rules left to discuss among maintainers and BUGSENG experts!
- Xen 4.18 release: 148 commits to fix MISRA C violations by bugseng
- ECLAIR MISRA C scanner integrated in the upstream Xen Gitlab CI-loop
  - **87** rules checked with **zero** unjustified violations (“clean” and checked against regressions)
  - **17** additional rules monitored at each commit (only few violations left, checked against regressions)

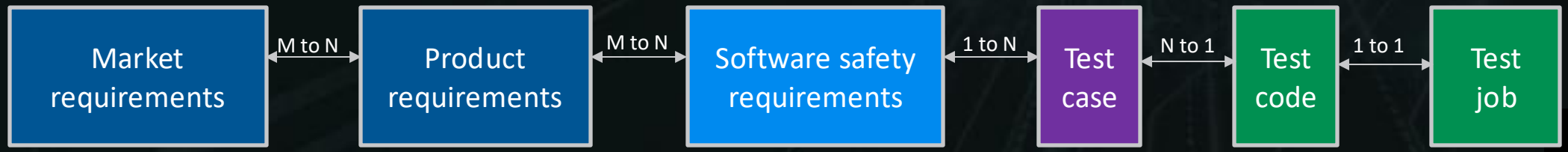
XEN SAFETY REQUIREMENTS

Requirements

- Detailed docs of all expected software behaviours
- Written in plan English from the perspective of what Xen is expected to fulfil
- The software safety requirements (SSR) are the most granular form
- SSR should be unambiguous, complete, consistent, correct
- Engineers are expected to refer to a SSR (and architecture spec) to write tests to validate it
- Each SSR should be tested independently

Traceability

- Define the Market Requirements first
- Market Requirements are linked to Product Requirements. Product Requirements explain how the hypervisor implements Market Requirements
- Product Requirements are further split into numerous Software Safety Requirements, which are individually testable
- Market Requirements, Product Requirements and SSR should be independently baselined
- SSR should be traceable all the way to market requirements



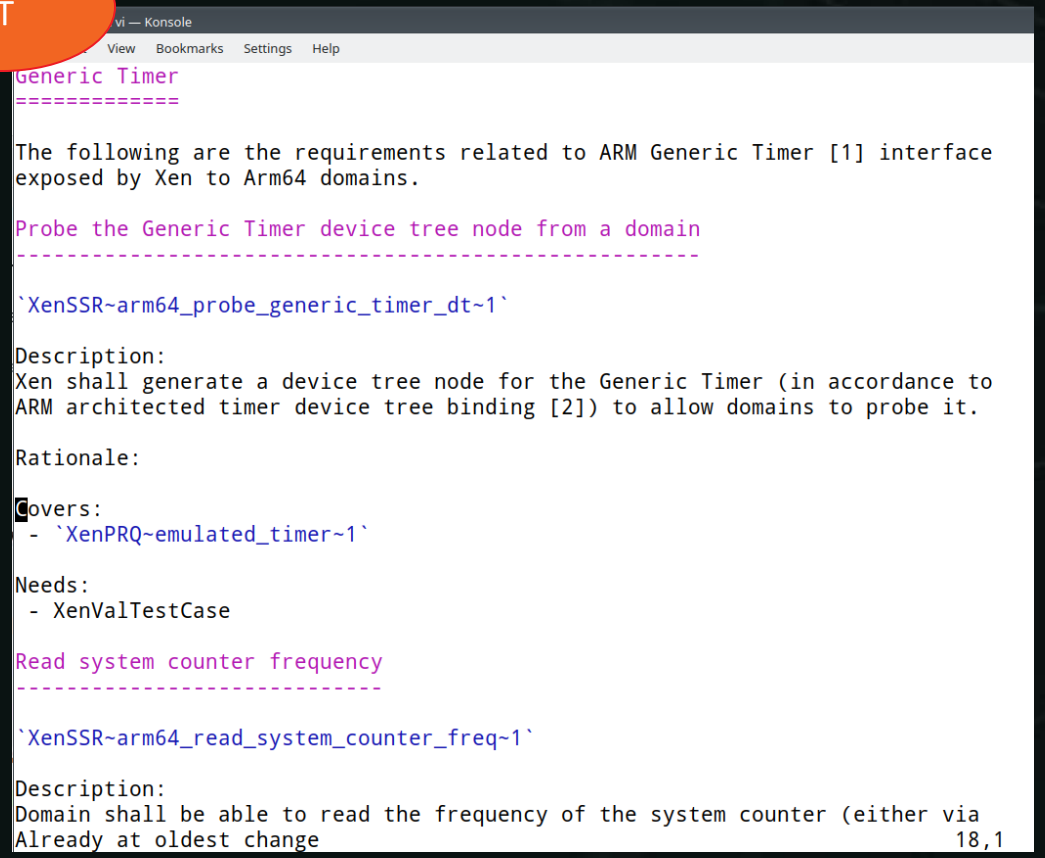
REQUIREMENTS-AS-CODE

- Traceability typically handled with complex proprietary solutions
  - They do not work well in an Open Source community environment
- Requirements are documents → Reuse the same processes we already have in Xen also for Requirements
- Benefits:
  - The Xen Community is already familiar with it
  - Zero ramp up time, high speed of development and review
  - Alignment with Zephyr, ELISA, and other Open Source software projects

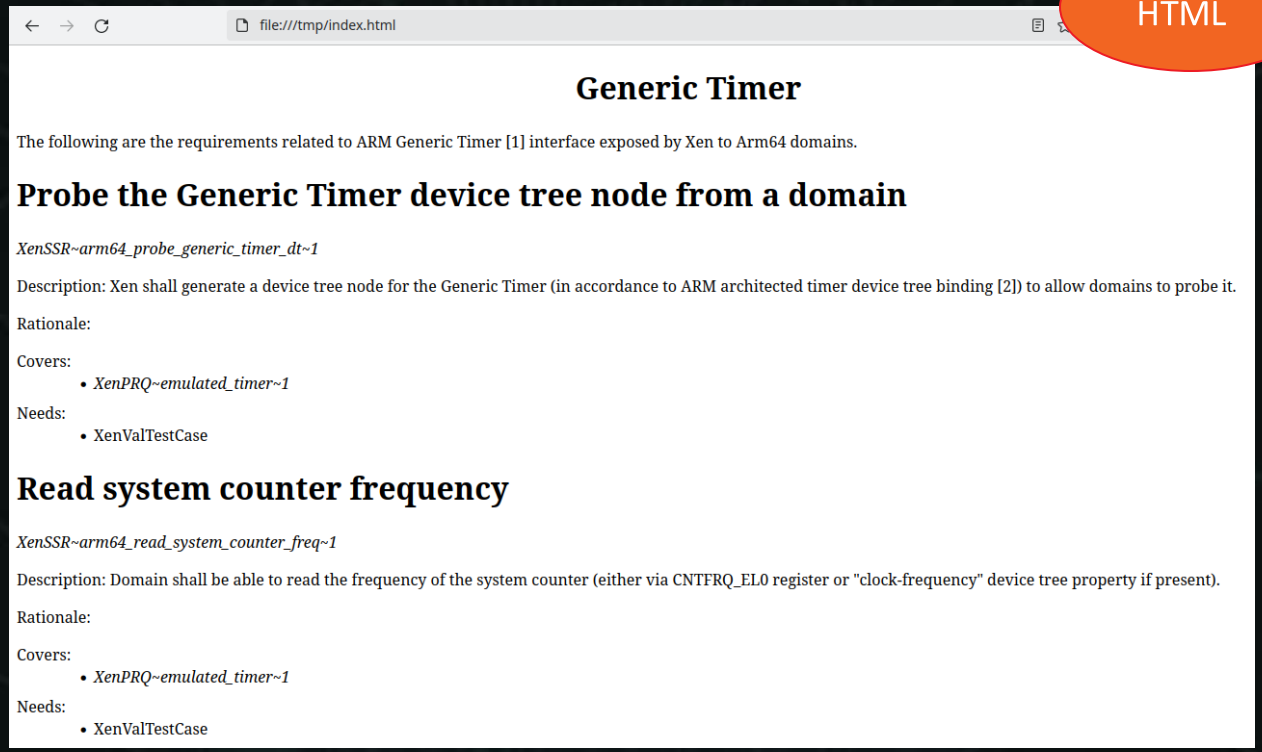
- For years now, the larger Open Source Community has been re-using the same powerful tools and processes for both code and documentation
- Write plain text documents using formats like RST and Markdown
  - Easily readable and modifiable in source format
  - They can also be rendered to PDF and HTML
- Same submission and review process as code; same version control as code

REQUIREMENTS-AS-CODE

RST



HTML



REQUIREMENTS-AS-CODE: TRACEABILITY

Linking and Traceability for Requirements-as-Code

- Open Source projects started to address this need:
  - OpenFastTrace
  - StrictDoc
  - Basil

We are using OpenFastTrace for linking and traceability reports

Requirements-as-Code is a great fit for Open Source software projects

No need for proprietary tools

OpenFastTrace: OSS requirement tracing tool

- Handles requirements written in markdown, RST
- Link requirements all the way to code
- Independent versions for each requirement
- Detect missing dependencies (missing links)
- Detect obsolete requirements (old versions)
- Generate reports in html and xml
- Extremely lightweight and fast
- Very mature (~7 years old), actively Maintained, proven in use
- https://github.com/itsallcode/openfasttrace/blob/main/README.md

OPENFASTTRACE – WRITING A REQUIREMENT

```
A Requirement Title With an Underline
-------------------------------------
`req~this-is-the-id~1`

Description:
Each specification id consist of 3 parts
- Artifact type – 'req’
- Name - 'this-is-the-id’
- Revision number – 1

Needs:
- subreq
```

This is the unique specification id

The artifact type of its sub requirement (i.e. child requirement)

TRACEABILITY REPORT:

✗ 541 total 102 defects | XenMRQ · XenPRQ · XenSSR · XenValTestCase · XenValTestCoc · XenValTestJob

XenMRQ

- ✗ **Xen hypervisor shall boot on Arm64 and AMD-x86 hardware**, rev. 1, XenMRQ
- ✗ **code_execution**, rev. 1, XenMRQ
- ✗ **freedom_from_interference**, rev. 1, XenMRQ
- ✗ **memory_access**, rev. 1, XenMRQ
- ✗ **memory_ballooning**, rev. 1, XenMRQ
- ✗ **non_pv_vms_support**, rev. 1, XenMRQ
- ✗ **run_arm64_x86_vms**, rev. 1, XenMRQ
- ✗ **security_features_vms**, rev. 1, XenMRQ
- ✗ **static_vm_definition**, rev. 1, XenMRQ
- ✗ **vm_device_assignment**, rev. 1, XenMRQ
- ✗ **vm_to_vm_communication**, rev. 1, XenMRQ
- ✗ **vms_resource_sharing**, rev. 1, XenMRQ

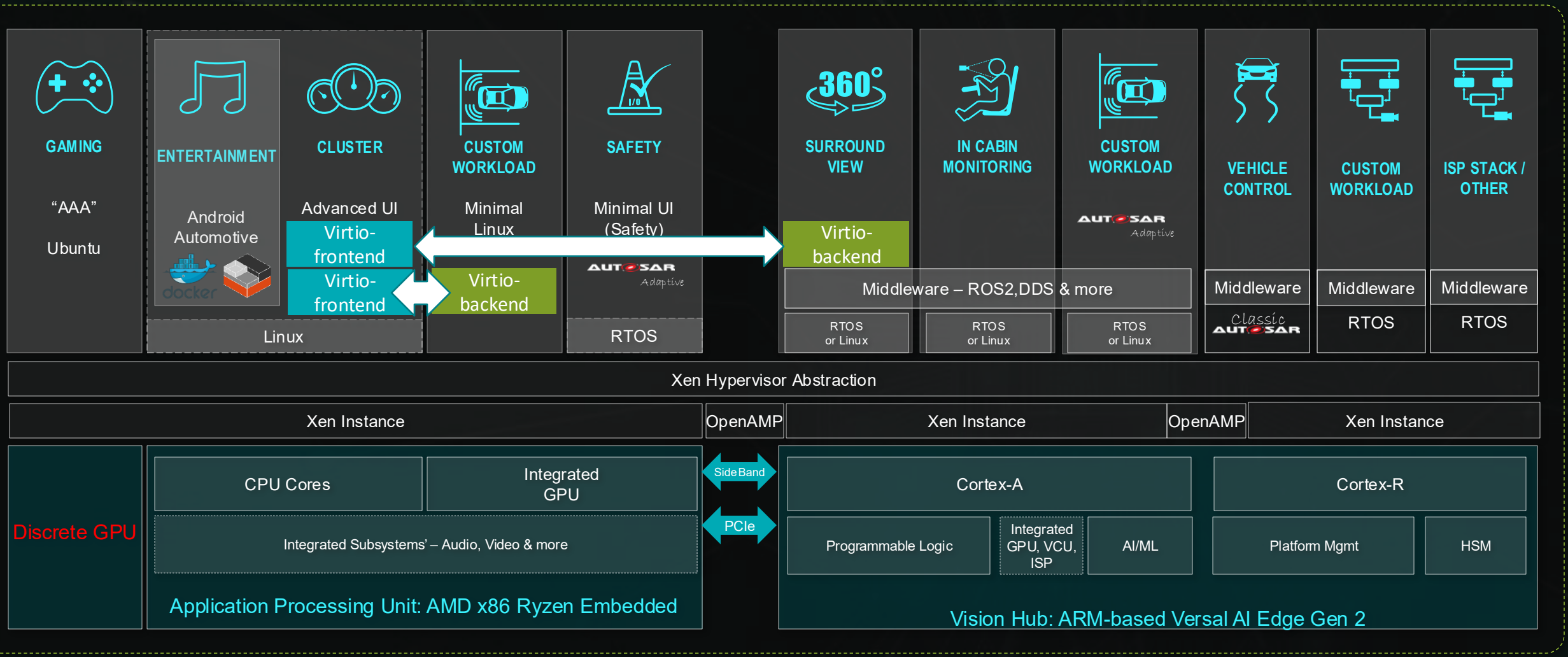
XenPRQ

- ✗ **Hardware Privilege**, rev. 1, XenPRQ
- ✓ **console_io_hypercall**, rev. 1, XenPRQ
- ✗ **device_passthrough**, rev. 1, XenPRQ
- ✗ **dm_hypercall**, rev. 1, XenPRQ
- ✓ **emulated_timer**, rev. 1, XenPRQ
- ✓ **emulated_uart**, rev. 1, XenPRQ
- ✗ **event_channel_hypercall**, rev. 1, XenPRQ
- ✗ **grant_table_hypercall**, rev. 1, XenPRQ
- ✗ **hvm_hypercall**, rev. 1, XenPRQ
- ✗ **hypervisor_bootloader**, rev. 1, XenPRQ
- ✗ **hypervisor_bsp_interface**, rev. 1, XenPRQ
- ✗ **instruction_emulation**, rev. 1, XenPRQ
- ✗ **memory_hypercall**, rev. 1, XenPRQ
- ✗ **physdev_hypercall**, rev. 1, XenPRQ
- ✗ **sched_hypercall**, rev. 1, XenPRQ
- ✓ **static_vms_configuration**, rev. 1, XenPRQ
- ✓ **vcpu_op_hypercall**, rev. 1, XenPRQ
- ✓ **version_hypercall**, rev. 1, XenPRQ
- ✗ **virtual_interrupt_controller**, rev. 1, XenPRQ
- ✗ **vms_memory_sharing**, rev. 1, XenPRQ

XenSSR

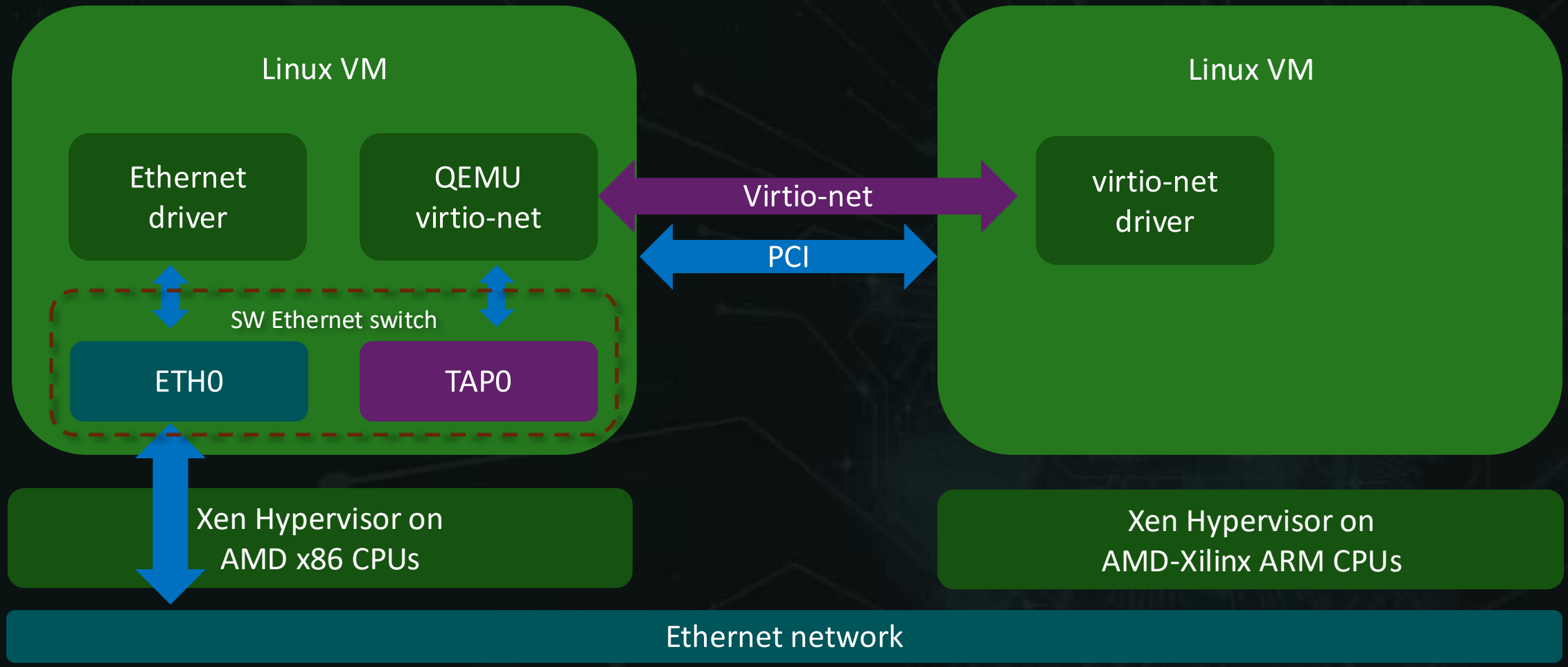
- ✓ **Access hypercall __HYPERVISOR_console_io for writing to the Xen console**, rev. 1, XenSSR
- ✓ **Access hypercall __HYPERVISOR_xen_version for getting xen version**, rev. 1, XenSSR
- ✗ **Access hypercall hvm_op for getting Xenstore shared page**, rev. 1, XenSSR
- ✗ **Access hypercall memory_op for mapping grant table page**, rev. 1, XenSSR
- ✓ **Access hypercall memory_op for mapping shared_info page**, rev. 1, XenSSR
- ✗ **Access hypercall memory_op for querying current memory reservation**, rev. 1, XenSSR
- ✓ **Access hypercall vcpu_op for registering vcpu_info structure**, rev. 1, XenSSR

USING XEN AS A UNIFIED RUNTIME MANAGER



A Common Hypervisor Layer across Clusters

WORK-IN-PROGRESS: HETEROGENEOUS VIRTIO-NET ACROSS PCIE



stefano.stabellini @ amd.com
xen-devel @ lists.xenproject.org

CONCLUSIONS

- **The Xen community is working on making Xen safety-certifiable ISO 26262 ASIL-D**
  - Certification based on Xen upstream community processes and code
  - Broad industry collaboration on Xen Safety; bi-weekly meetings:
    - Resiltech, Renesas, Ford, Boeing, EPAM, ARM, BUGSENG, Tenstorrent, RSB
    - Let's work together!
  - MISRA C and Software Safety Requirements
- Xen Architecture for Safety
  - From Vision Hub to IVI
  - Mixed-criticality, Real-time, VirtIO and PV Drivers
  - Multiple configurations possible
    - e.g. number of Safe VMs, presence of Control Domain, criticality of Hardware Domain
- Accelerating momentum in the Xen Project
  - Honda and Ford becoming part of the Xen community

linaro Connect 2025

Thank You!

AMD