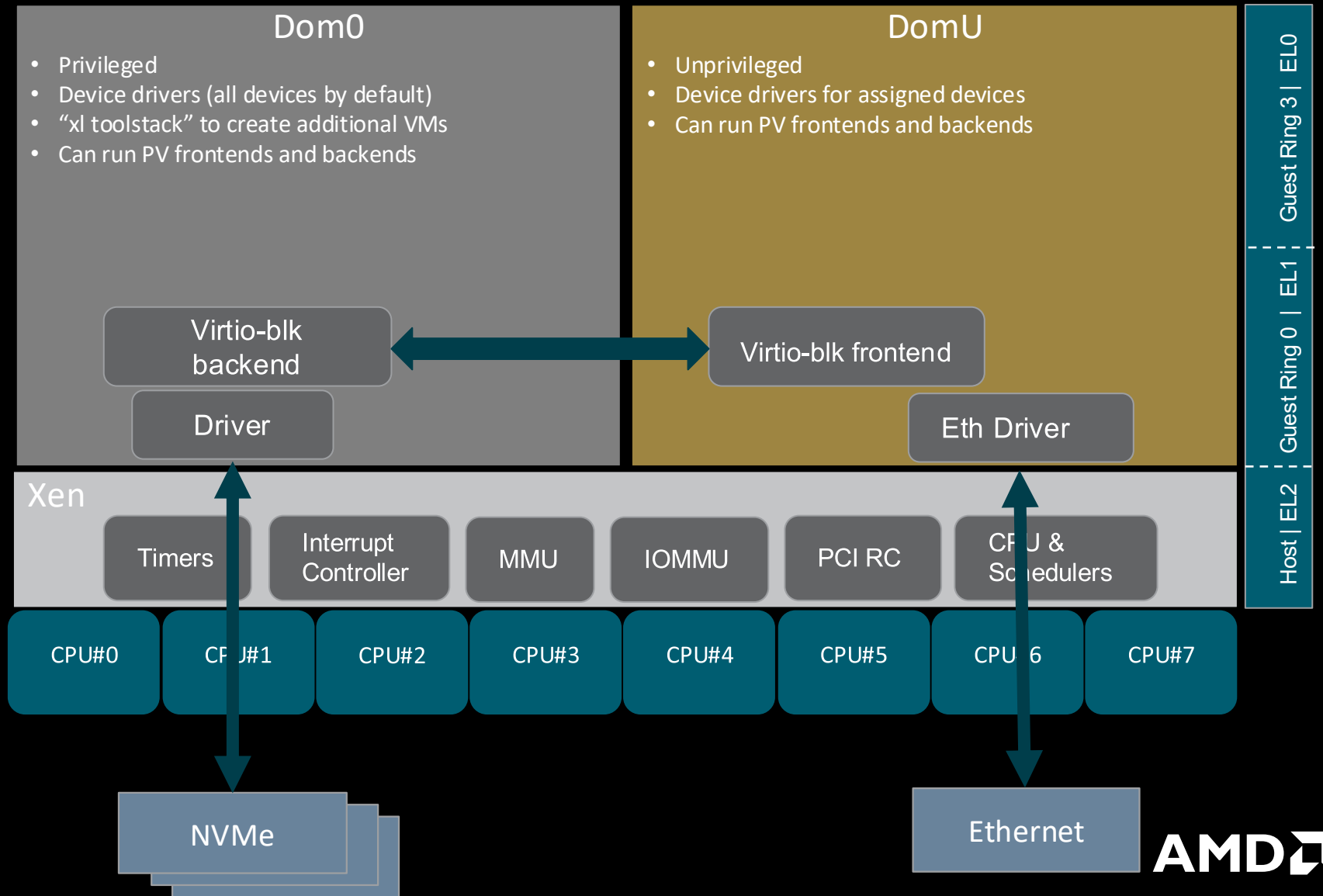
The Final Phase of Xen Safety: Solving Code Coverage and Residual Gaps

Stefano Stabellini
Fellow at AMD
Xen Hypervisor & Linux Kernel Maintainer

20 May 2026
Minneapolis, Minnesota

Modern Xen Architecture, AMD x86 / ARM / RISC-V

- Xen runs in its own privilege level (host mode / EL2); uses hardware virt support
- Xen owns the key HW components:
  - CPUs, Timers, Interrupts, MMU, IOMMU, PCI RC
- Devices are directly assigned to Domains
  - By default, to Dom0
  - Fully configurable
- PVH domains
  - No need for QEMU for emulation
  - No need for ”PV guests”
  - QEMU only for Virtio backends
- Dom0less (Hyperlaunch)
  - Parallel boot of VMs
  - Dom0 becomes optional
- Devices can be shared with a PV frontend/backend architecture
  - Both Xen PV drivers and Virtio
  - Multiple device sharing models supported



[Public]

Slide from Xen Summit 2025 hosted for the first time here at the AMD office in San Jose last September.

11/2024

AMD
achieves Xen Safety Concept Approval (ISO 26262 ASIL-D)

12/2024

HONDA
joins Xen Project

3/2025

RENESAS
joins Xen Project

5/2025

AUTOMOTIVE GRADE LINUX
announces a new Xen-based architecture

6/2025

Ford
joins Xen Project

May 20 2026

Boeing joins Xen Project

as new Advisory Board member!

Xen® Project

BOEING

Xen: the full spectrum, from IVI to Vision Hub & Industrial

In-Vehicle Infotainment (IVI)

Graphics

Virtio & PV devices

Richer configurations

Real-time

Static Partitioning

Simpler configurations

Vision Hub & Industrial

Xen: the full spectrum, from IVI to Vision Hub & Industrial

In-Vehicle Infotainment (IVI)

Graphics
Virtio & PV devices
Richer configurations

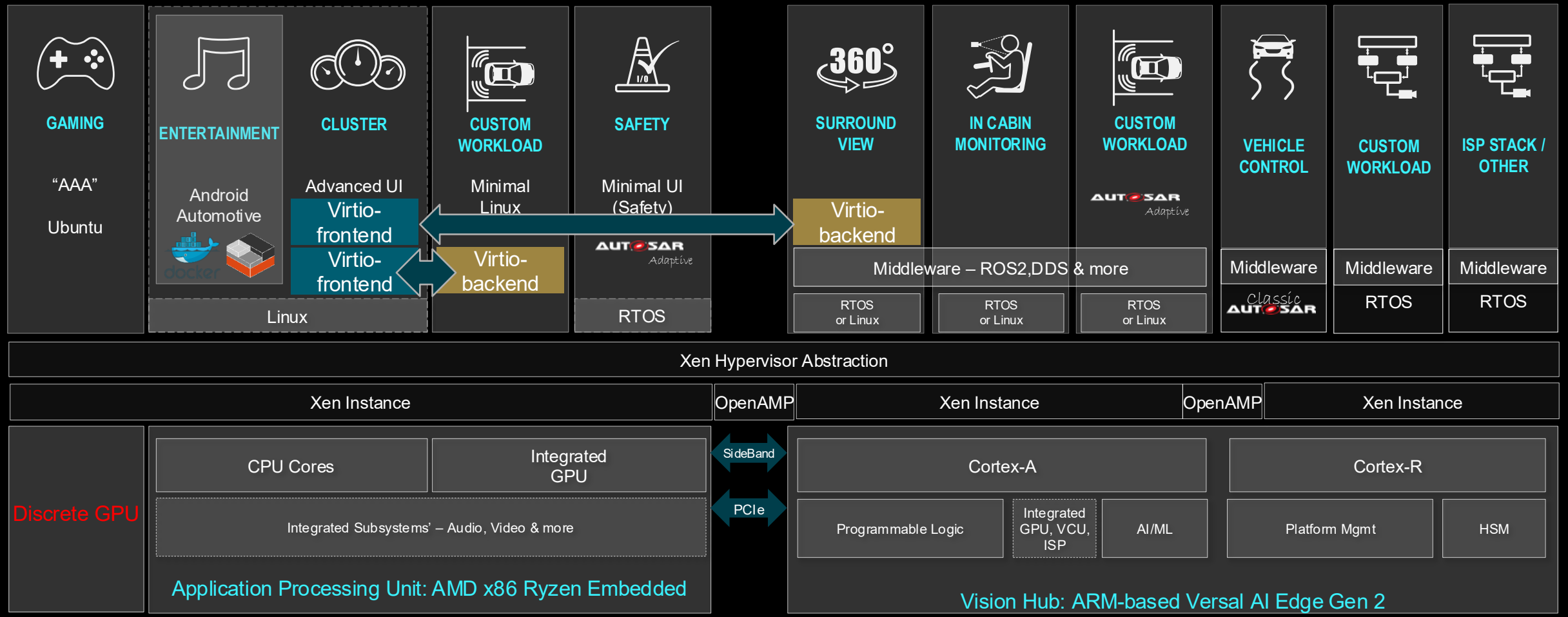
Real-time
Static Partitioning
Simpler configurations

Vision Hub & Industrial

- Dom0less/Hyperlaunch for faster boot times
- Several Virtio protocol supported: Virtio-NPU, Virtio-net, Virtio-block, Virtio-gpu, Virtio-sound, Virtio-media, Virtio-tee, Virtio-gpio, Virtio-i2c device, Virtio-input
- Passthrough available

- Dom0less/Hyperlaunch for static partitioning
- Hardware passthrough to domains
- Fully dedicated pCPUs to vCPUs (no scheduling)
- 3us interrupt latency under interference
- Minimal Xen footprint at runtime
- Xen on microcontrollers without MMU (Cortex-R52, R82)
- Virtio available

Xen: from IVI to Vision Hub



A Common Hypervisor Layer across Clusters

AMD

Safety-Certifying the Xen Hypervisor

- **AMD safety certification work near completion**
  - **IEC 61508 SIL 3 & ISO 26262 ASIL D**
  - Both ARM and AMD x86
  - Phase I done, Phase II nearing completion
  - Certification based on Xen upstream community processes and upstream codebase
  - Already fully public and Open Source:
    - MISRA C rules adopted and documented
    - All Xen code changes are Open Source and public
    - Safety requirements being upstreamed
    - Test harness and examples
      - Requirement-based, QEMU fault-injection, fuzzing
    - Xen Project GitLab CI open to any community member
      - AMD hosts 11 runners; partners can plug in their own hardware

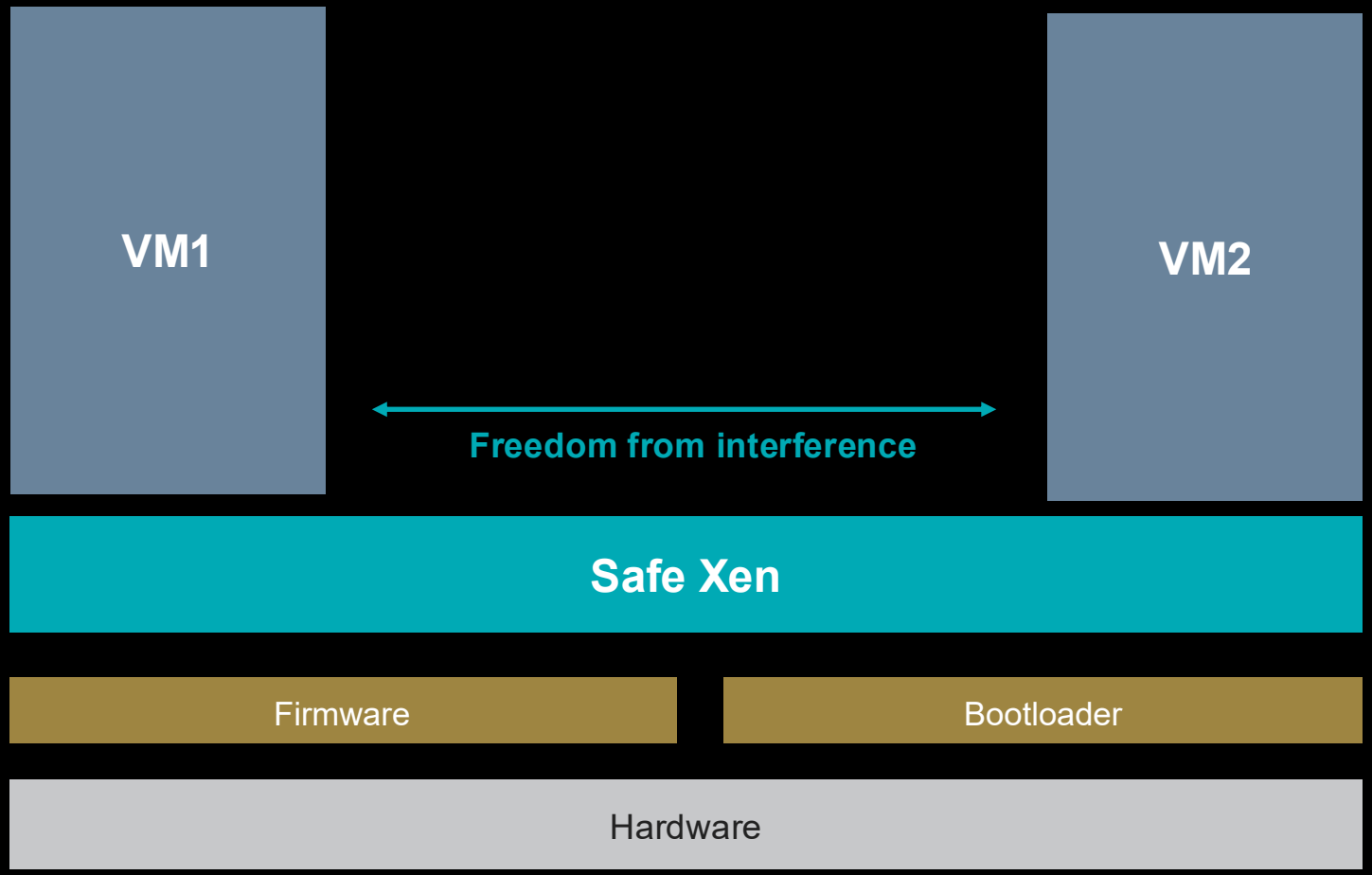
AMD Phase I — Completed (Nov 2024)

- Configuration, change, and document mgmt plans accepted
- Custom V-model lifecycle established
- Software safety requirements (concept, examples, traceability)
- Software architecture spec (concept, examples, traceability)
- Verification & validation plan (high level)
- Software validation & verification plans
- Software tools classification
- **Safety Concept Approval achieved**

AMD Phase II — Nearing Completion

- MISRA C: 99% complete and upstream
- Requirements, architecture specs
- Validation and Verification tests
- 100% Code Coverage
- Safety Case, Safety Manual
- Final assessment and certification

The role of Xen in Functional Safety systems



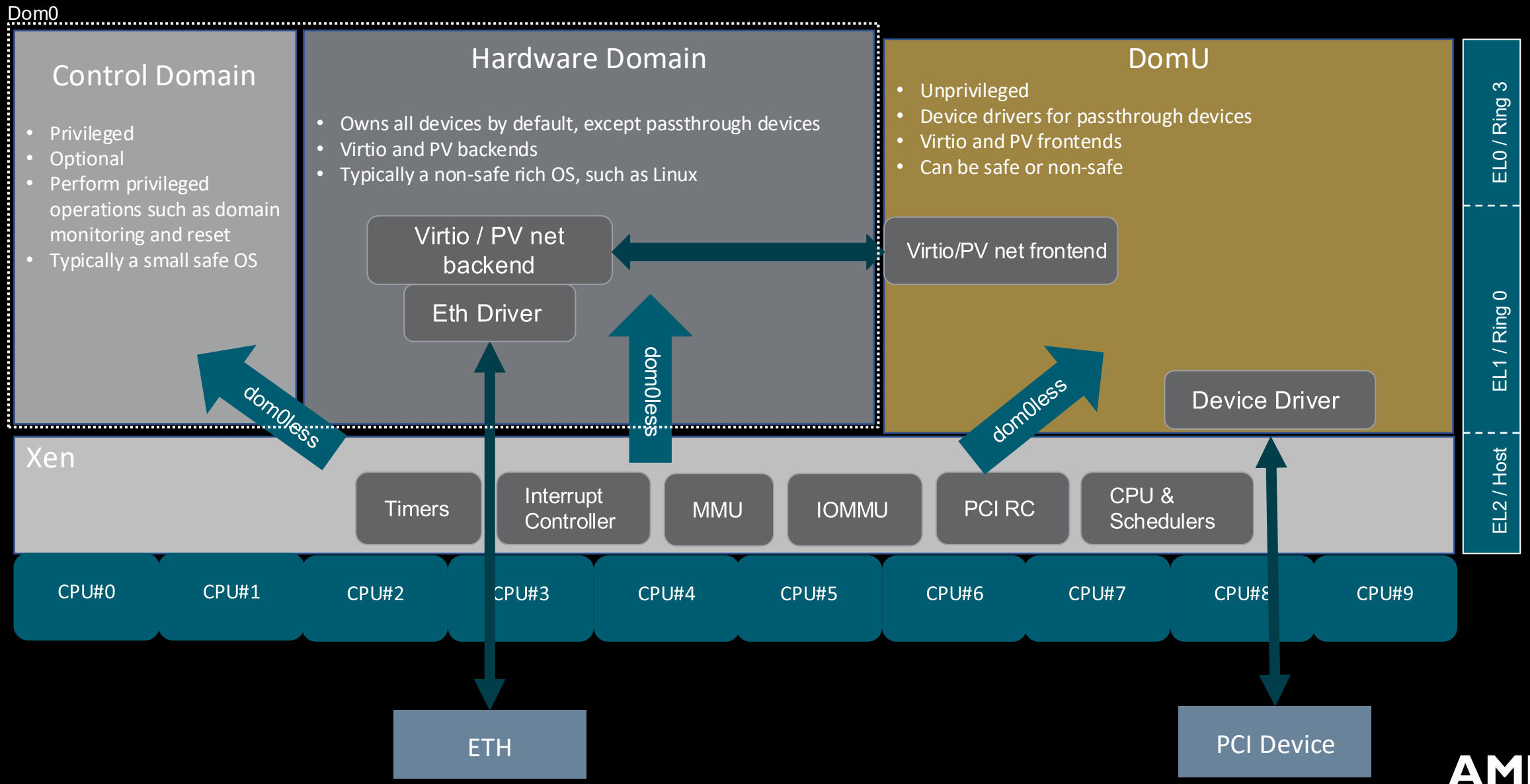
Type-1 Hypervisor with x86 & Arm Support

Static Partitioning
- Memory and HW resources are partitioned across VMs
- A VM cannot access another VM's resources
- Real-time isolation

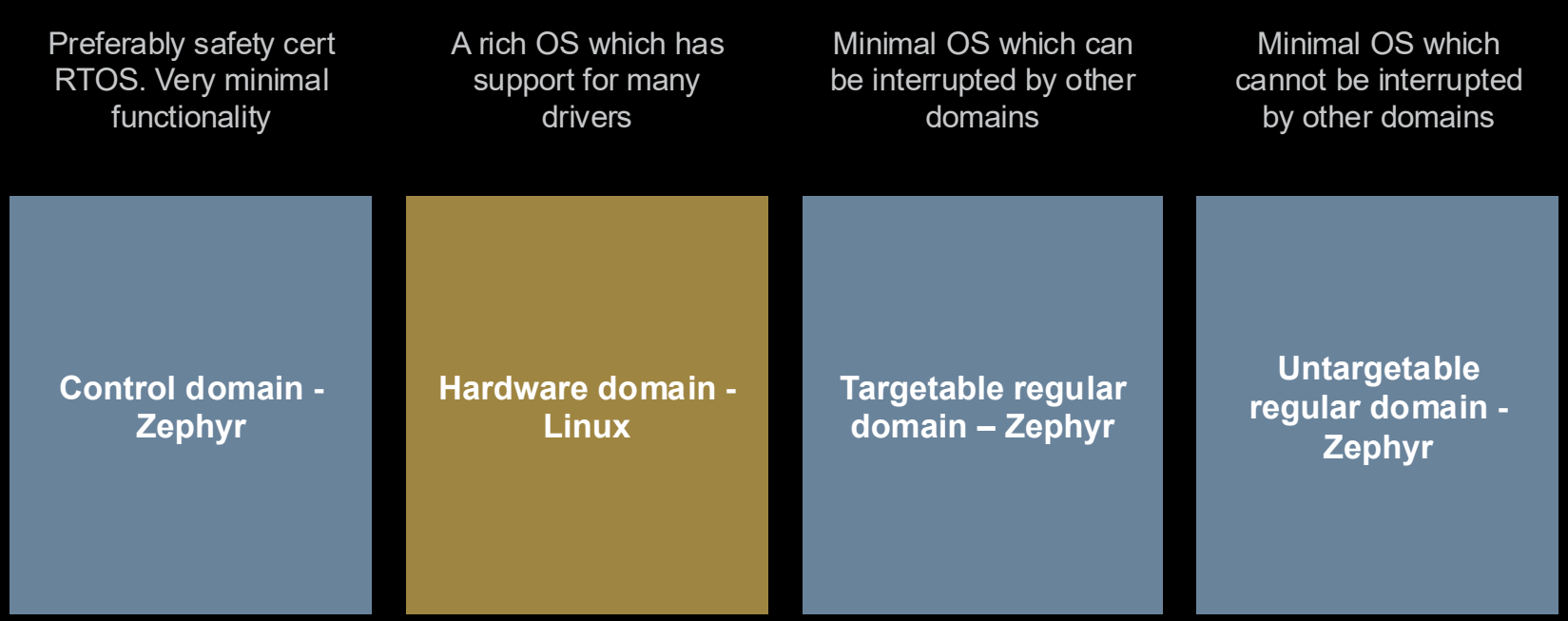
Domain Creation
- Xen can create and boot domains in parallel (Hyperlaunch/Dom0less)
- Dom0 is not required to create domains

Assumptions
- Firmware can have a safety integrity level ≥ that of Xen
- VM1, VM2 can have a safety integrity level ≤ that of Xen
- VM1 and VM2 can be of different safety integrity levels
- VM1 and VM2 can be Linux, Zephyr, etc.

Modern Xen Architecture for Safety



Xen with Zephyr and Linux as VMs



Per system:
- 0–1 control domain
- 0–1 hardware domain
- 0–N targetable / untargetable domains

Domain creation
- Static (boot-time) domains.
- Dynamic (run-time) domains.

AMD

Domain Roles and Capabilities

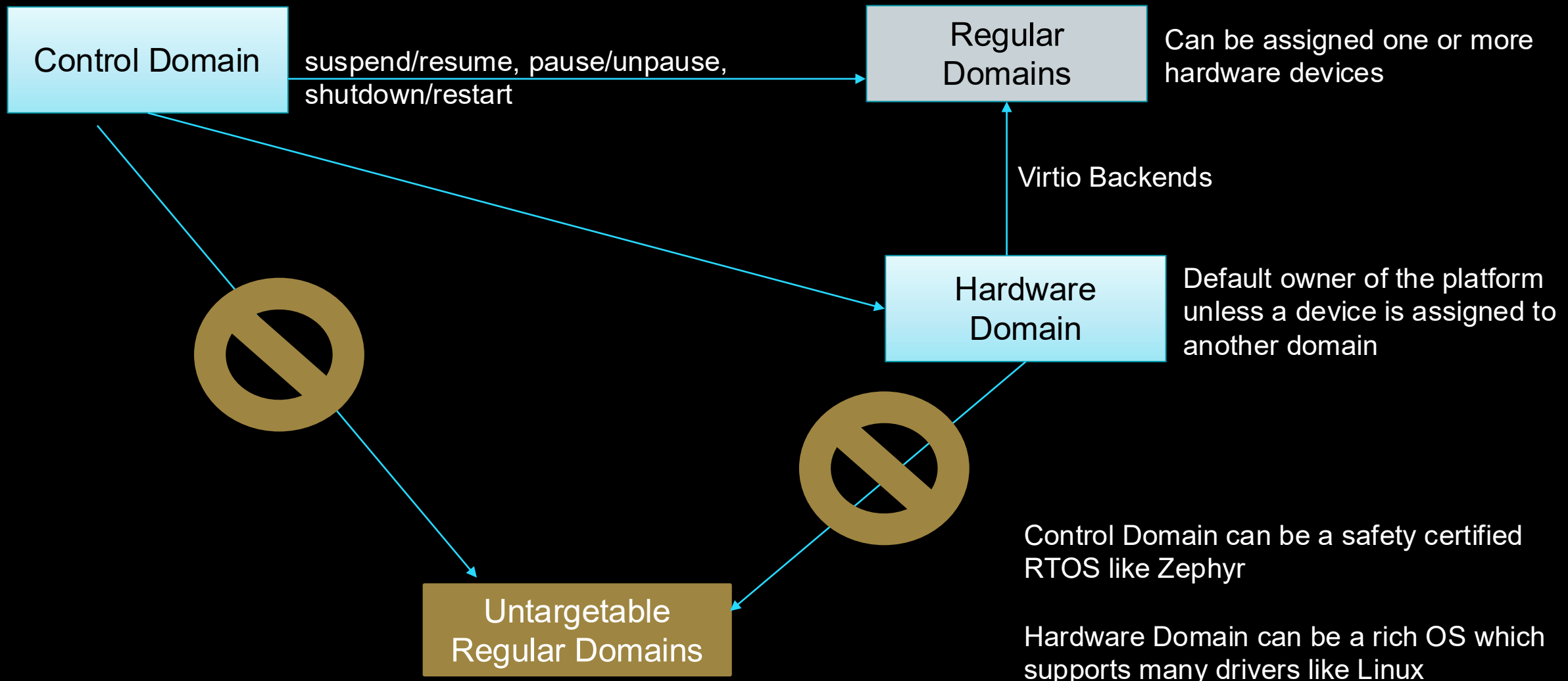
The Implementation Details

- Domain capabilities:
  - DOMAIN_CAPS_CONTROL
    - Control Domain
  - DOMAIN_CAPS_HARDWARE
    - Hardware Domain
  - DOMAIN_CAPS_XENSTORE
    - Run Xenstored
  - DOMAIN_CAPS_DEVICE_MODEL
    - Subset of hypercalls needed to run Virtio backends, expected to be given to the Hardware Domain
- Configured via Dom0less/Hyperlaunch:
  - capabilities = <0xf>

- New domain flag: XEN_DOMCTL_CDF_not_hypercall_target
- Configured via Dom0less/Hyperlaunch:
  - hypercall-untargetable = <1>;
- Used in XSM checks. If set, XSM checks will fail when an untargetable domain is the hypercall target

- “Safe Virtio” extensions:
  - Virtio + grant-table
    - By Juergen. Already upstream in Linux and QEMU.
  - Non-blocking Virtio
    - By Edgar, upstreaming pending. Based on 2 extra virtio registers, runtime discoverable

Domain Roles and Capabilities

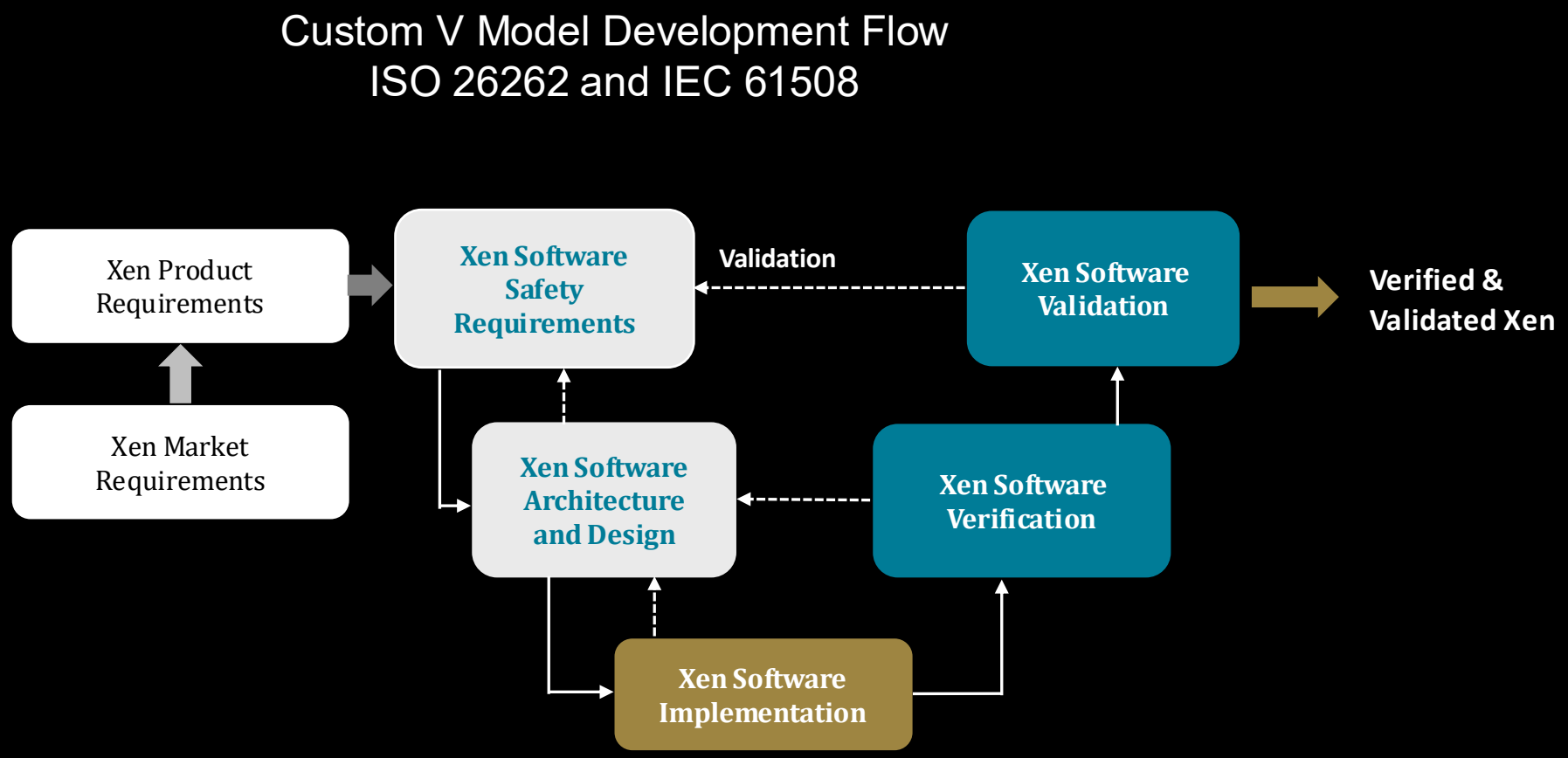


14

AMD

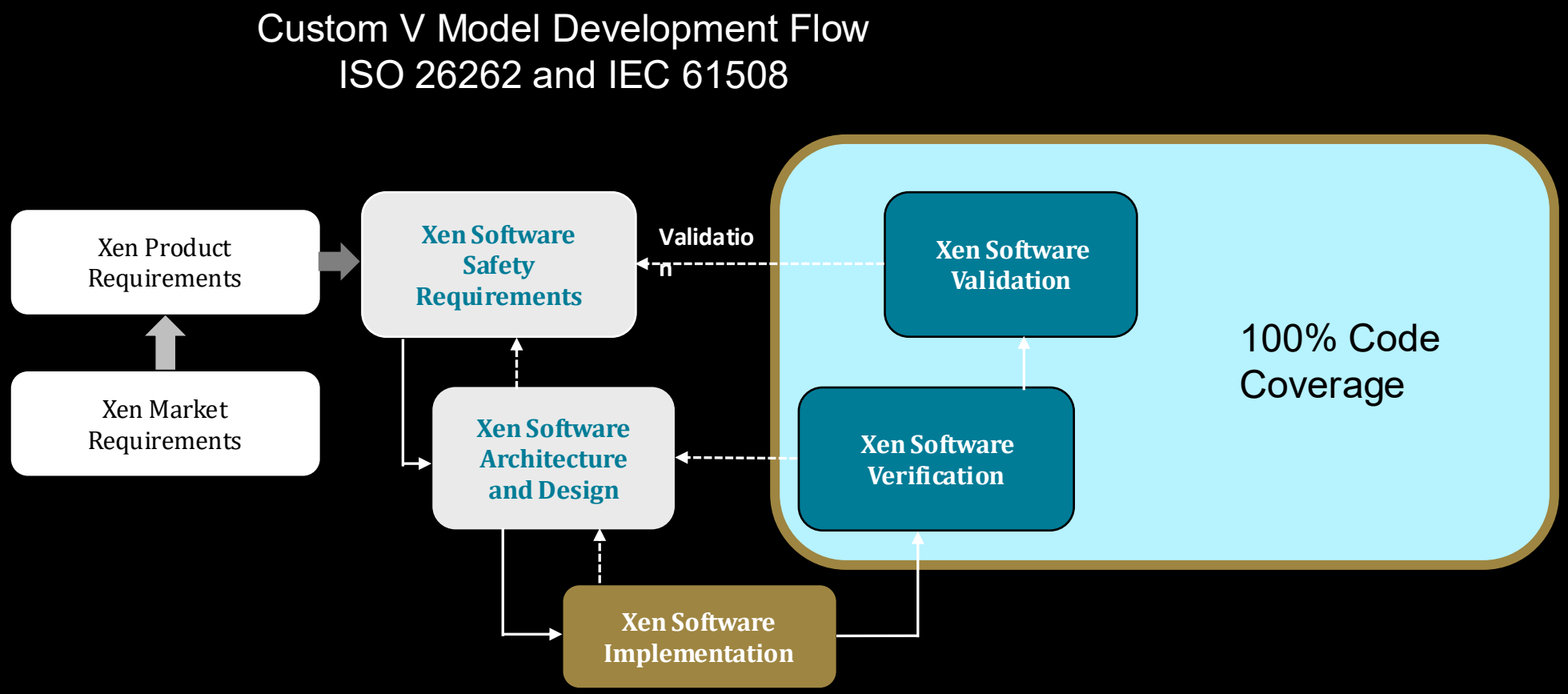
Code Coverage & Dead Code Elimination for Fun and Profit

Custom Safety Life-cycle Process



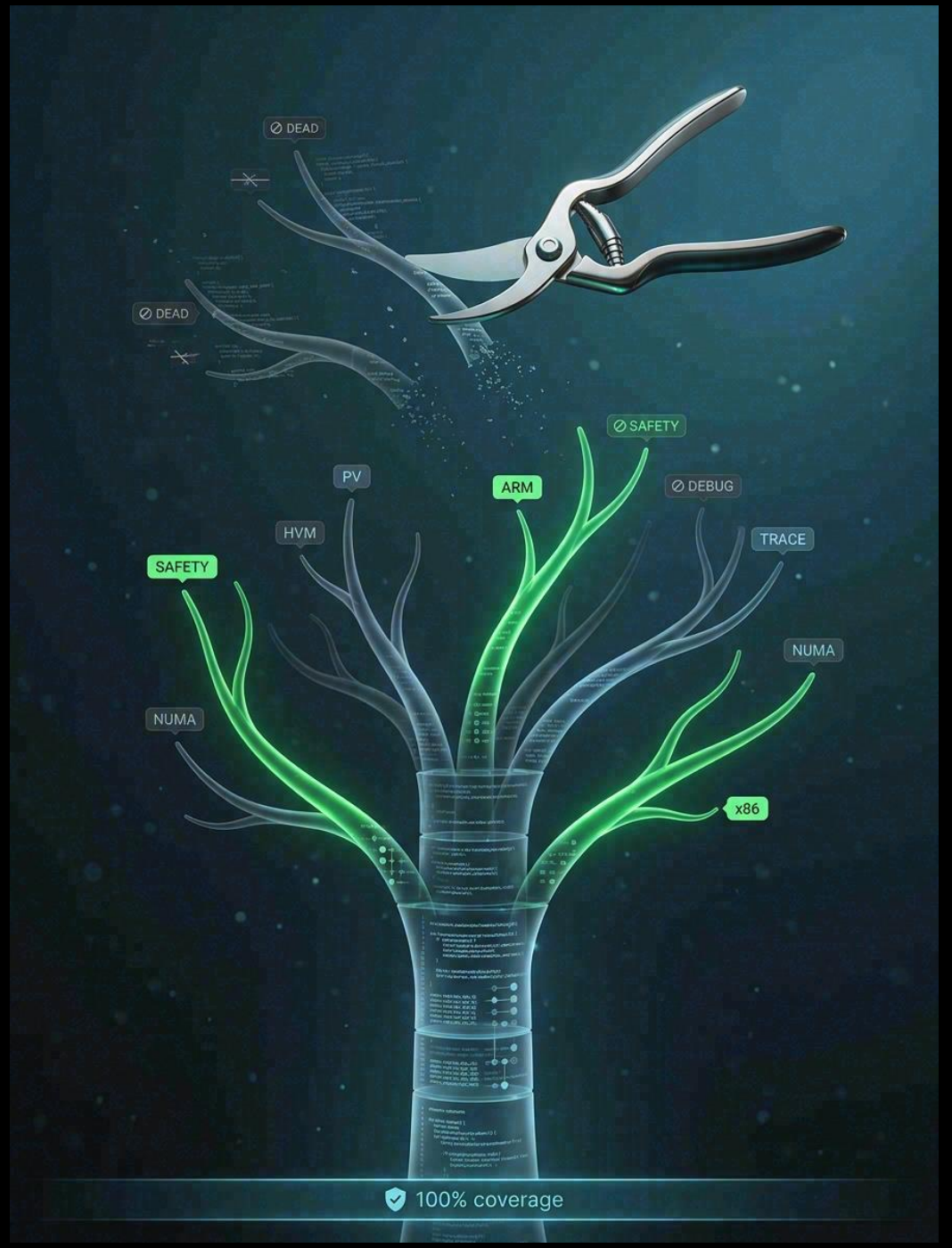
16

Custom Safety Life-cycle Process



Code Coverage: the Problem

- Open Source projects support many configurations, not just the safety-critical one
- This breadth is a strength: the code is more flexible, more widely used, better reviewed, and better tested
- However, in any given configuration, large portions of the code are unused
  - effectively dead code at build time
- Safety certification requires 100% coverage of the code that is built
  - the dead code must be removed
- The question is: how?



Classic dead-code elimination — works, but invasive

Three established techniques. *All require touching the source. All scatter conditionals through the tree.*

Makefile gating

Coarse — drops whole files

```
obj-$(CONFIG_FEATURE) +=
    feature.o
```

Limits

- Whole-file granularity
- Needs extern stubs in callers

#ifdef CONFIG_X

Surgical — invisible to the compiler

```
#ifdef CONFIG_FEATURE
    do_feature();
#endif
```

Limits

- Disabled branch never type-checked
- Guards proliferate at every site

IS_ENABLED(CONFIG_X)

Type-safe — both branches compile

```
if(IS_ENABLED(CONFIG_FEAT))
    do_feature();
```

Limits

- Compiler sees both sides
- Still: a guard at every use site

Doesn't scale to thousands of switch cases. Next: let the compiler do the work.

19

Value Range Propagation + bitmaps: cpu_vendor()

What if the value itself told the compiler what is reachable?

Before: `#ifdef everywhere`

```
switch ( boot_cpu_data.vendor )
{
#ifdef CONFIG_AMD
  case X86_VENDOR_AMD:
    setup_amd();    break;
#endif
#ifdef CONFIG_INTEL
  case X86_VENDOR_INTEL:
    setup_intel();  break;
#endif
}
```

After: `cpu_vendor()`

```
switch ( cpu_vendor() )
{
  case X86_VENDOR_AMD:
    setup_amd();    break;
  case X86_VENDOR_INTEL:
    setup_intel();  break;
}
```

The trick *(asm/cpufeature.h)*

```
#define X86_ENABLED_VENDORS \
 ((IS_ENABLED(CONFIG_INTEL) ? \
     X86_VENDOR_INTEL : 0) | \
  (IS_ENABLED(CONFIG_AMD)   ? \
     X86_VENDOR_AMD   : 0) | …)

static always_inline unsigned int
cpu_vendor(void)
{
  return boot_cpu_data.vendor
         & X86_ENABLED_VENDORS;
}
```

Kconfig builds the bitmap. Each enabled vendor sets one bit; mask drops impossible vendors before the switch
AMD-only build: X86_ENABLED_VENDORS is 1 bit → `cpu_vendor()` collapses to a constant → all other vendor cases vanish via DCE
Net effect: vendor-conditional paths across `xen/arch/x86/cpu/` compile out — no #ifdef required.

Value Range Propagration + sets: CONST_FILTER_OR_JUMP()

VRP handles ranges. Hypercall switches over hundreds of opcodes are not ranges.

Wrap the switch expression once:

```
switch ( CONST_FILTER_OR_JUMP(op->cmd, return -EINVAL,
                              CMD_0x002, CMD_0x034, CMD_0x134) )
{
  case CMD_0x001:  handle_0x001(); break; /* ELIMINATED */
  case CMD_0x002:  handle_0x002(); break; /* KEPT */
  …
  case CMD_0x034:  handle_0x034(); break; /* KEPT */
  case CMD_0x035:  handle_0x035(); break; /* ELIMINATED */
  case CMD_0x036:  handle_0x036(); break; /* ELIMINATED */
  …
  case CMD_0x134:  handle_0x134(); break; /* KEPT */
  case CMD_0x135:  handle_0x135(); break; /* ELIMINATED */
  default: handle_default();       break;  /* ELIMINATED */
}
```

The trick *(xen/macros.h)*

```
/* CONST_FILTER_OR_JUMP(x, or, c1, c2, c3)
 * expands to: */
({
  typeof(x) __x = x, __y;
  if      (__x == c1) __y = c1;
  else if (__x == c2) __y = c2;
  else if (__x == c3) __y = c3;
  else or;    /* must not return */
  __y;
})

switch (CONST_FILTER_OR_JUMP(
          var, goto skip, 1, 3, 5))
{
  case 1: ...
  case 2: /* ELIMINATED */
  case 3: ...
  case 4: /* ELIMINATED */
  case 5: ...
  default: /* ELIMINATED */
}
skip:
```

Value Range Propagration + sets: CONST_FILTER_OR_JUMP()

VRP handles ranges. Hypercall switches over hundreds of opcodes are not ranges.

Wrap the switch expression once:

```
switch ( CONST_FILTER_OR_JUMP(op->cmd, return -EINVAL,
                              CMD_0x002, CMD_0x034, CMD_0x134) )
{
  case CMD_0x001:  handle_0x001(); break; /* ELIMINATED */
  case CMD_0x002:  handle_0x002(); break; /* KEPT */
  …
  case CMD_0x034:  handle_0x034(); break; /* KEPT */
  case CMD_0x035:  handle_0x035(); break; /* ELIMINATED */
  case CMD_0x036:  handle_0x036(); break; /* ELIMINATED */
  …
  case CMD_0x134:  handle_0x134(); break; /* KEPT */
  case CMD_0x135:  handle_0x135(); break; /* ELIMINATED */
  default: handle_default();       break;  /* ELIMINATED */
}
```

Real impact, upstream

8 hypercall dispatch sites filtered

`via xen/include/xen/amd-hypercall.h`
`VCPUOP, XENMEM, GNTTAB, EVTCHN,`
`HVMOP, HVM_PARAM (get/set), DMOP`

~3 000 LoC removed

x86 instruction emulator — thousands of switch cases, only ~20 needed, scattered randomly

1 edit updates every site

Allowlists live in one header — no #ifdef sprawl across .c files

22

AMD

Link-time GC: --gc-sections *(zero source change)*

What if you didn't have to touch the source at all?

CONFIG_CC_SPLIT_SECTIONS
`-ffunction-sections  -fdata-sections`
Each function / object gets its own ELF section

CONFIG_GC_SECTIONS
`ld --gc-sections`
Linker drops sections it can't reach

CONFIG_PRINT_GC_SECTIONS
`ld --print-gc-sections`
Diagnostic — see what got dropped

Source: no change required *(callers vanished via cpu_vendor() / CONST_FILTER_OR_JUMP())*

```
void do_feature(void)        { … }
void do_feature_b(void)      { … }
void do_feature_c(void)      { … }
bool feature_flag;

/* No callers remain anywhere */
```

Build output: `ld --print-gc-sections`

```
ld: removing unused section '.text.do_feature'
ld: removing unused section '.text.do_feature_b'
ld: removing unused section '.text.do_feature_c'
ld: removing unused section '.data.feature_flag'
```

Linker walks reachability from entry points and drops orphans.
Pairs perfectly with the prior techniques: they kill use-sites, gc-sections sweeps up the orphan callees — for free.

23

Action at a distance — when to use what

Three techniques. Different jobs. They compose.

Technique	When to reach for it	Source change	Granularity
`gc-sections`	Functions become unreachable (callers removed by DCE)	None — pure build flag	Whole functions / data objects
`cpu_vendor()`	Runtime value drawn from a small enum (e.g. CPU vendor)	Replace boot_cpu_data.vendor with cpu_vendor()	Switch arms / branches
`CONST_FILTER_OR_JUMP()`	Switch over an arbitrary opcode allowlist (hypercall, instruction, …)	One line change in the switch statement	Switch arms / case bodies

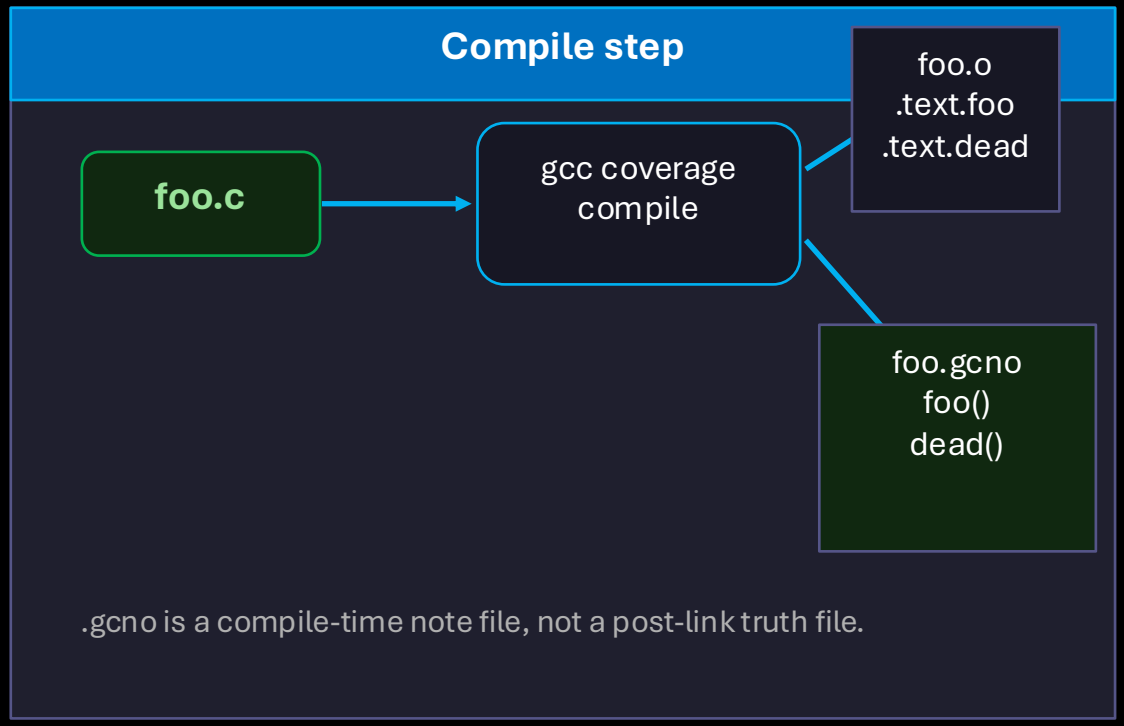
Why this matters for safety certification

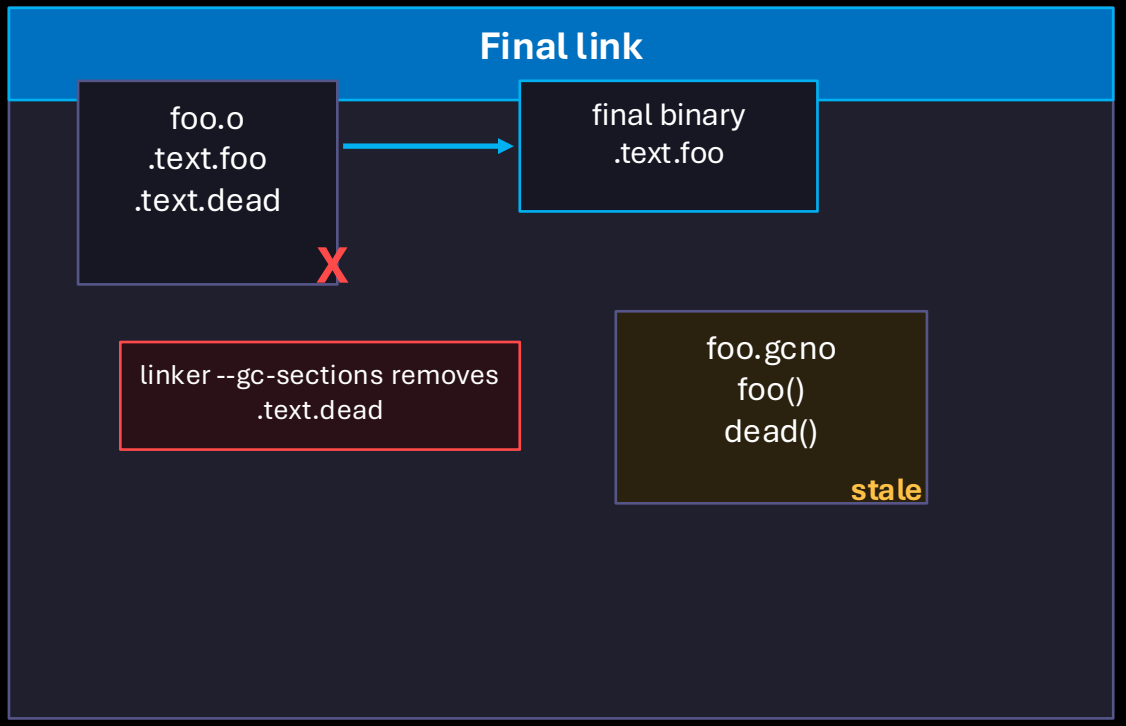
- Impossible to meet MISRA / coverage targets when unreachable code is part of the safety build
- These techniques remove unreachable code in the safety build with minimal upstream impact
- No fork-per-build needed

24

Coverage *notes*: written before the final link

GCC records what it compiled. The linker later decides what actually survives.





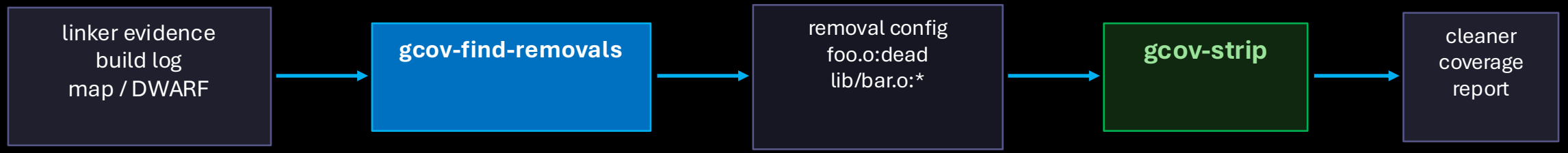
Small mismatch, noisy reports

Coverage tools read the stale .gcno, so removed functions can still look like uncovered code.

25

Let the build tidy up gcov notes

Use the linker's own evidence, then trim only the matching .gcno records
New gcov scripts: gcov-find-removals and gcov-strip



Relaxed, but still precise

Object-qualified rules keep edits local: foo.o:dead touches foo.gcno, not every helper named dead.

26

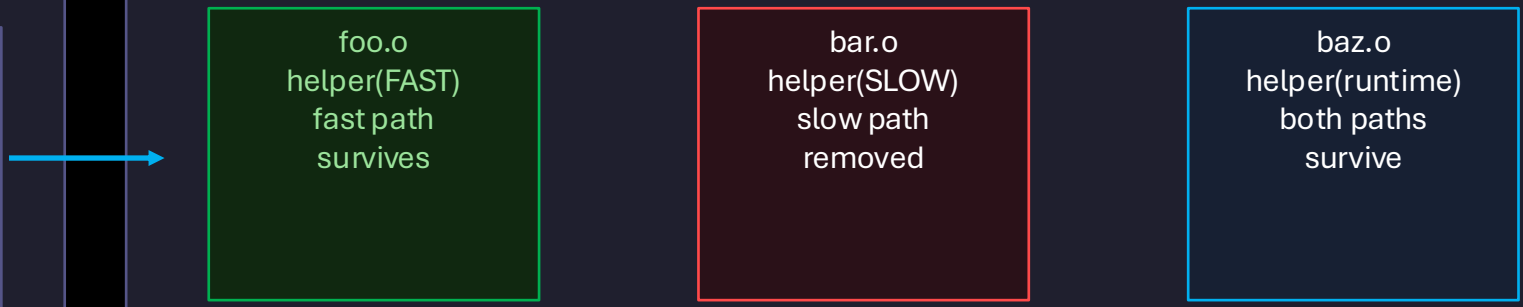
Static inline functions with shared names

Each object gets its own copy, and optimization can keep different pieces in each copy.

Helper.h

```
static inline helper(mode)
{
   if (mode == FAST)
      fast_path();
   else
      slow_path();
}
```

After compilation and optimization



The same source helper now has object-local coverage records.

wrong: helper — would erase valid copies

right: bar.o:helper — only trims the stale copy

Why this gets subtle

A header helper may produce different code in each .o, so we trim only the removed copy.

27

AMD

Fault-injection

Fault injection in QEMU

Programmable, reproducible hardware manipulation in CI

Exercise Xen error paths that are hard, rare, or unsafe to trigger on real hardware.

Real hardware	QEMU + Python
- Rare timing windows - Needs platform RAS or safety support - Difficult to repeat exactly - Limited internal visibility	- Scripted fault setup - Deterministic CI jobs - Direct emulated hardware access - Observe Xen through symbols and logs

The test becomes the hardware lab: configure, perturb, observe, verify.

Python framework and APIs

A single test script coordinates QEMU machine control, device control, and Xen introspection

Python test

QMP API

- Inject NMI and MCE events
- Add/remove PCI devices
- Reset the machine
- Run monitor commands
- Query QOM/device state
- Read or dump physical memory

Qtest API

- Read/write MMIO and RAM
- Read/write x86 I/O ports
- Override future MMIO and port reads
  - *"next read to return value 0x1"*
- Monitor and drive IRQs and wires

GDB API

- Breakpoints and watchpoints
- Read/write C variables
- Evaluate C expressions
- Call functions (API testing)
- Single-step

Same Python harness runs locally and in GitLab CI; failures are reproducible from python scripts.

30

Example fault-injection

Combine APIs to steer Xen into specific error paths

x86 NMI / MCE

NMI tests
- QTest overrides port 0x61 reads // set NMI *reason*
- QMP inject-nmi enters Xen NMI path
- Xen reads port 0x61, classify error as PCI SERR

MCE tests
- QMP mce injects CPU machine-check state
- Xen MCE/RAS path runs
- Test validates panic or recovery behavior

ARM64 Serror / other errors

SError
- Qtest writes to FMU_SMINJERR_ADDR to trigger SError
- Test validates panic or recovery behavior

GICv3 init error
- QTest overrides GICR_WAKER reads // replace with sleep
- Test validate gicv3_cpu_init returning timeout error

GDB unit tests
- GDB sets breakpoint inside Xen
- On break, GDB calls aarch64_set_branch_offset
- Test validates immediate field extraction on various INSNs

Stop Xen, perturb emulated hardware, continue execution, and verify the expected state transition.

31

Xen Safety: a Community-wide Collaboration

Automotive industry adopting Xen

- **Honda** and **Ford** joined the Xen Project
- Xen Project collaborating with Automotive Grade Linux

Active Contributors

- AMD · EPAM · Renesas · Bugseng · Resiltech
- **Boeing** expanding its role in Xen safety work

Safety Pilot — AMD · Renesas · EPAM

- Active joint effort to accelerate safety work
- Precursor to a broader Xen Safety Initiative, coming soon
- A large share of the safety artifacts is in the pilot:
  - Safety requirements
  - Architecture specifications
  - Fault injection tests, Fuzzing, and more
  - FMEA, tool qualification, and more

Open Collaboration

- Certification tracks the same code everyone uses
  - No private safety fork to maintain
- Updating the certification stays a bounded effort
  - Upstream changes flow back into the safety baseline
- Community review increases assurance
  - Independent panel of maintainers across vendors
- Lowers the barrier for adoption
  - Start from a public, reviewed, MISRA-clean codebase
- Keeps the safety effort evergreen
- Shared effort keeps a single, current source of truth
  - Lower per-company cost, predictable timelines

Xen: the Open Source hypervisor for safety critical systems

- A record-breaking year in automotive
  - Honda, Ford, Renesas, Automotive Grande Linux, Boeing
  - More companies investing in Xen safety certification

- AMD safety certification work near completion
- Closing the last gaps
  - Domain roles: Hardware / Control domain split, Untargetable flag
  - 100% code coverage, DCE and gc-sections
  - Fault injection in QEMU: Python framework using QMP / QTest / GDB

- Join us at Xen Summit 2026!

AMD

AMD