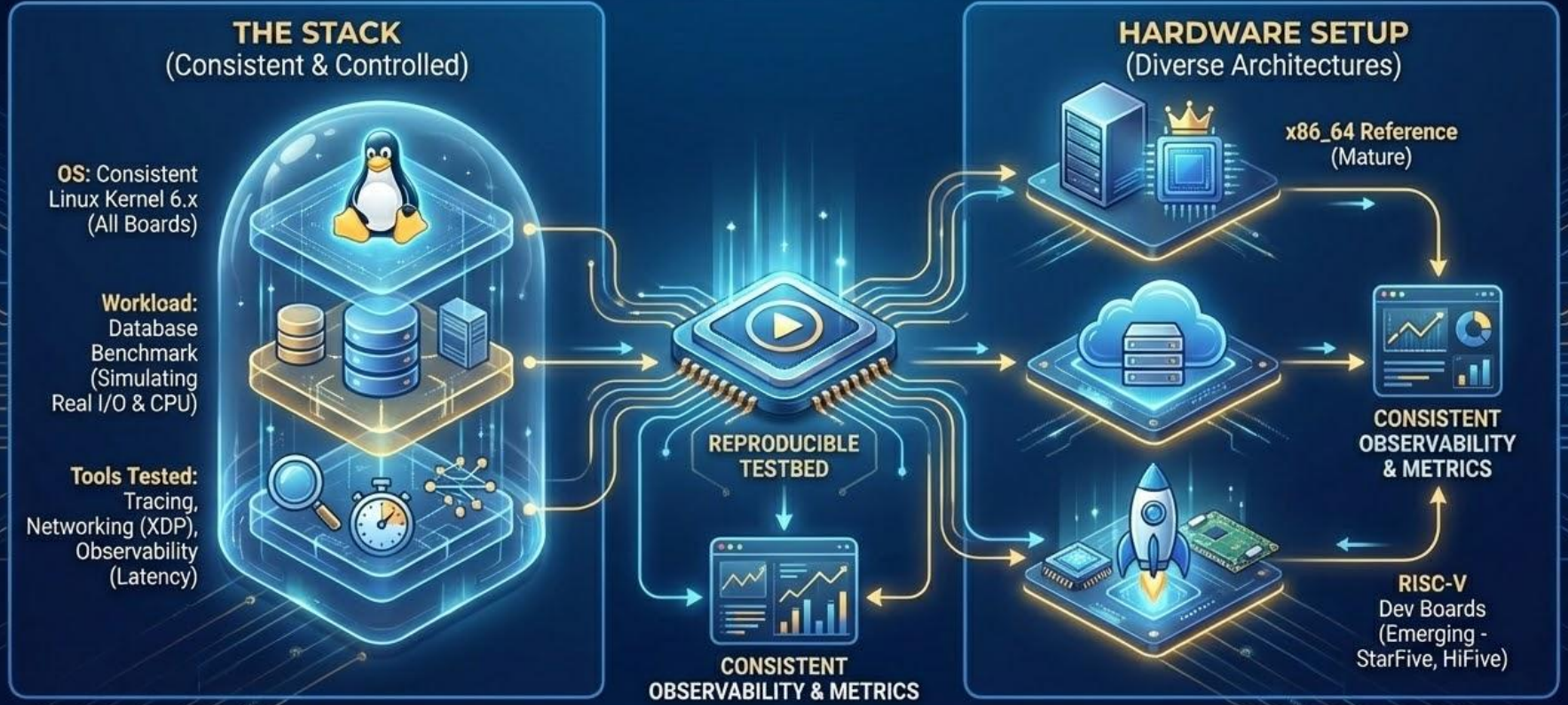
METHODOLOGY: A REPRODUCIBLE TESTBED

Objective: Eliminate environmental noise to test the architecture itself.



BPF Observability on RISC: What Works, What Breaks, and How to Test It

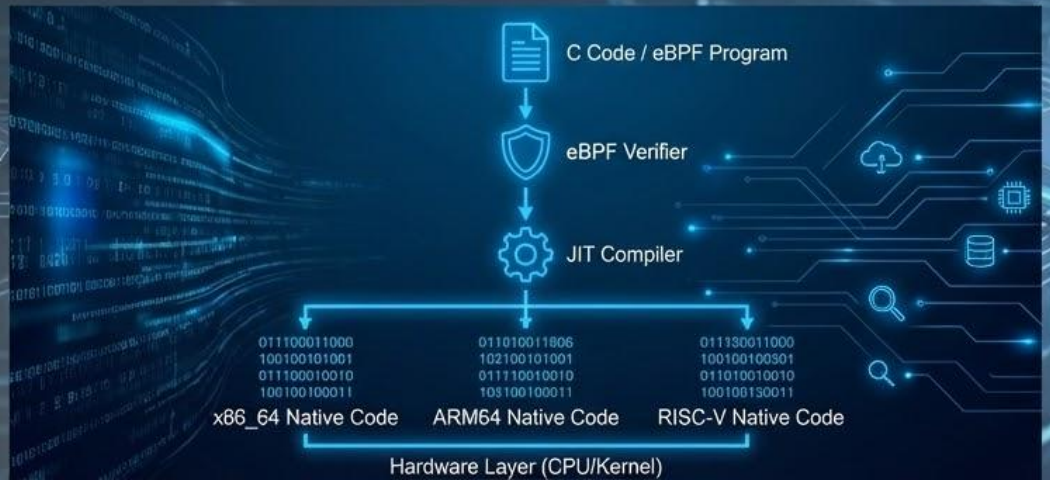
A Comparative Analysis of x86_64, ARM64, and RISC-V

Bruce Gain | Analyst
ReveCom | www.revecom.io

Yuning Liang | Founder and CEO
DeepComputing | deepcomputing.io

The Context: eBPF is Everywhere

- **The Promise:** eBPF has become the standard for modern observability, networking, and security.
- **The Assumption:** We often assume "Write Once, Run Anywhere" (WORA) because of the bytecode abstraction.
- **The Reality:** eBPF bytecode is architecture-independent, but the **JIT (Just-In-Time) compiler, helpers**, and **kernel support** are deeply tied to the underlying hardware (ISA).
- **Visual:** A diagram illustrating the translation path from high-level C code down to the specific native machine code for each architecture:



The Challenge: RISC vs. CISC in the Kernel

A Comparative Analysis of Mature and Emerging Architectures

x86_64 (CISC)

Mature JIT, full feature parity, the "gold standard" for eBPF.

ARM64 (RISC)

Widely used in cloud (AWS Graviton, etc.), mature but occasionally diverges in verifier constraints.

RISC-V (RISC)

The emerging frontier. Rapidly evolving, but JIT support is newer and fragmentation is a risk.

Key Question: Can we reliably use standard observability tools (BCC, bpftrace, Cilium) on these RISC architectures today?

Findings: ARM64 – Ready for Prime Time?

STATUS

Mostly stable. Parity with x86 is high.

NUANCES

JIT Maturity: Excellent.

Verifier Constraints: Some register usage differences can cause complex programs to be rejected on ARM64 that pass on x86.

Performance: Overhead is comparable to x86.

VERDICT

PRODUCTION-READY

Production-ready for most observability use cases.

Findings: RISC-V – The Wild West

STATUS & WHAT WORKS

Functional but incomplete. Basic socket filtering, simple tracing.

WHAT BREAKS: Helpers & Trampolines

Missing Helpers: Specific kernel helper functions may not be implemented yet.

Trampolines: fentry/fexit support has historically lagged.

JIT LIMITATIONS

Tail calls or complex loops may trigger fallback to the interpreter (huge performance hit).

Case Study: Database Benchmarking

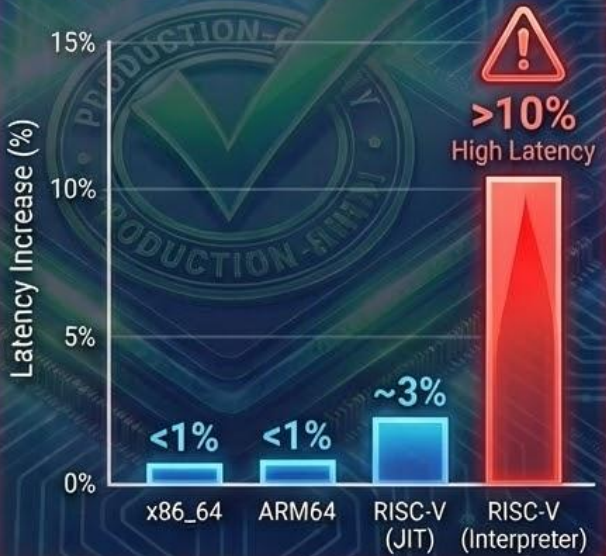
SCENARIO & INSTRUMENTATION

- High-throughput database (e.g., Postgres/MySQL) under load.
- Attaching eBPF probes to measure query latency.

RESULTS SUMMARY

- **x86/ARM64**: negligible overhead (<1%).
- **RISC-V**: Variable overhead.
  - **Scenario A**: JIT works -> Low overhead.
  - **Scenario B**: Feature missing (Interpreter fallback) -> High latency spikes.

Database Latency Impact of eBPF Instrumentation



How to Build a Multi-Arch Testbed

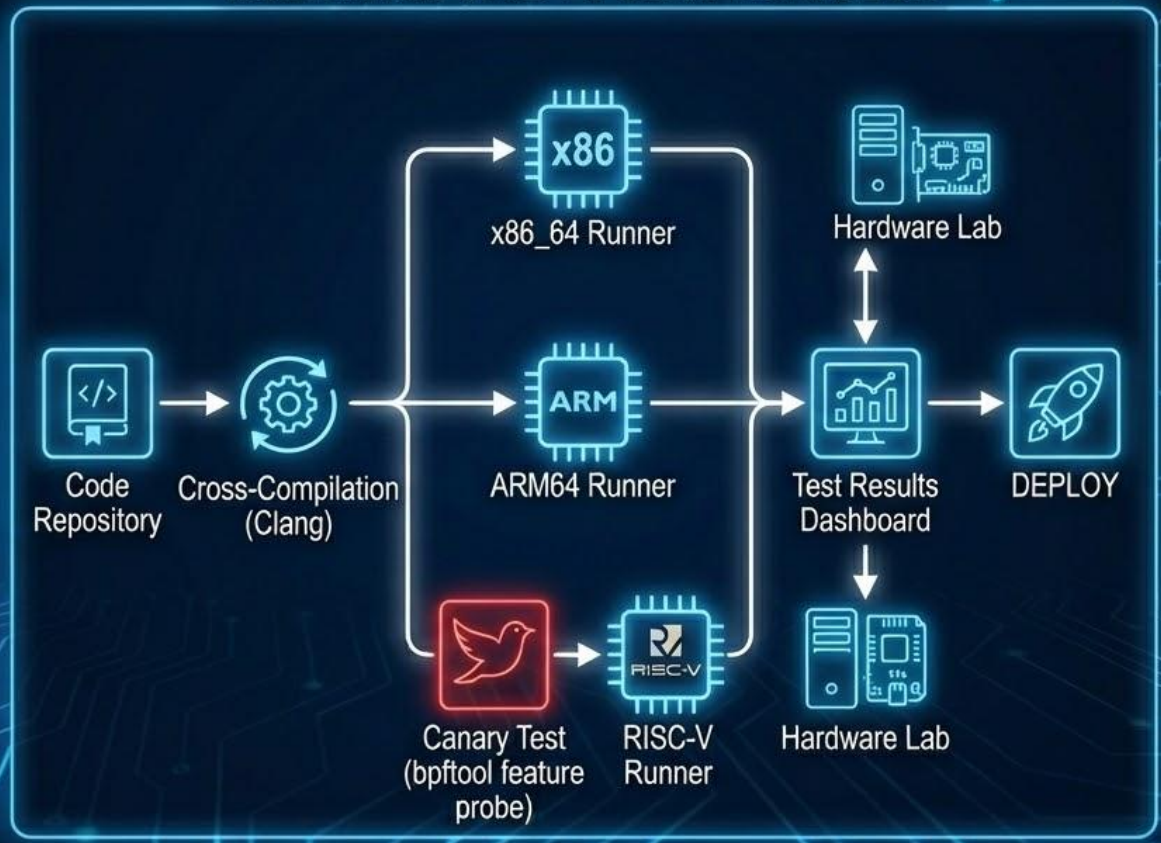
BEST PRACTICES & INTEGRATION

Don't rely on QEMU
- Emulation often hides JIT/Verifier bugs specific to real silicon.

CI/CD Integration
- Cross-compilation is easy (Clang is a cross-compiler!).
- Testing requires native runners.

The 'Canary' Test
- Always run a capability probe (e.g., bpftool feature probe) before deploying agents to RISC-V nodes.

MULTI-ARCH CI/CD PIPELINE WORKFLOW



Conclusion & Takeaways

ARM64

1st Class Citizen

- ARM64 is a first-class citizen in the eBPF world.

RISC-V

Feature-Flagging Required

- RISC-V is exciting but requires rigorous feature-flagging; do not assume feature parity.

Testing

Test on Real Hardware

- **Testing:** You must test eBPF artifacts on actual hardware to catch JIT and Verifier edge cases.

Future: The gap is closing, but observability teams need to be aware of the underlying ISA.