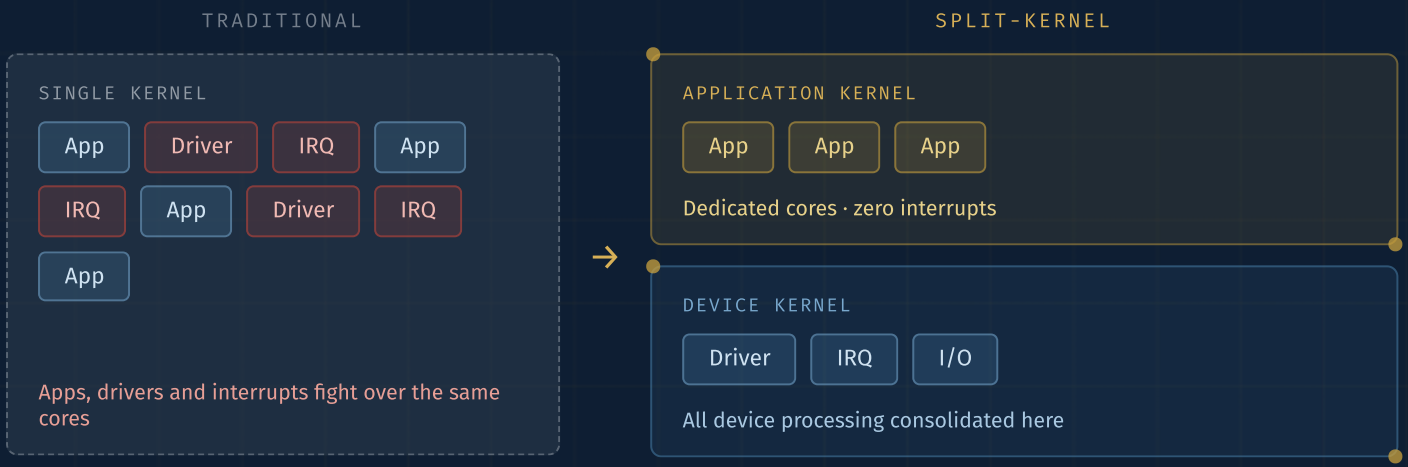
Bare-Metal Performance · Kernel-Level Isolation
Next-Generation Linux Infrastructure, No Hypervisor

Multikernel is built on split-kernel architecture: each application runs in its own Linux kernel with dedicated CPU cores, while device drivers and interrupt handling are consolidated in a dedicated device kernel. No hypervisor required. On the servers or cloud instances you already have, **get bare-metal performance and VM-grade isolation at the same time**.



0	0	<1 s
Interrupts on application cores	**Layers between app and hardware**	**Kernel failover**
Structural, not tuning	No virtualization layer, direct hardware access	Zero-downtime upgrades · instant rollback

Platform & Infrastructure Teams

Consolidate more workloads per machine: no hypervisor tax, no virtualization licenses, no noisy neighbors

AI Infrastructure Teams

Kernel-level sandboxes for untrusted agents and generated code: direct GPU access at container-like speed

SRE & Fleet Operations

Fleet-wide kernel patching with sub-second failover and instant rollback. No more maintenance windows

PRODUCTS · THREE OPEN-SOURCE PRODUCTS

Multikernel Private Cloud

PRIVATE CLOUD OS

Private cloud without the hypervisor: consolidate Linux workloads at high density on bare metal or cloud instances

- ✓ Native hardware performance
- ✓ Zero cross-workload interference

Multikernel Sandbox

AI AGENT SANDBOX RUNTIME

Each AI agent gets its own kernel, so strong isolation and full GPU power are no longer a trade-off

- ✓ Direct GPU access
- ✓ Fast snapshot & restore

Multikernel LiveUpdate

ZERO-DOWNTIME KERNEL UPGRADES

Kernel patches and upgrades applied live. Workloads never notice, and reboot windows never happen

- ✓ Sub-second failover
- ✓ Instant rollback

Remove the virtualization layer
Kernel-native performance, isolation and availability

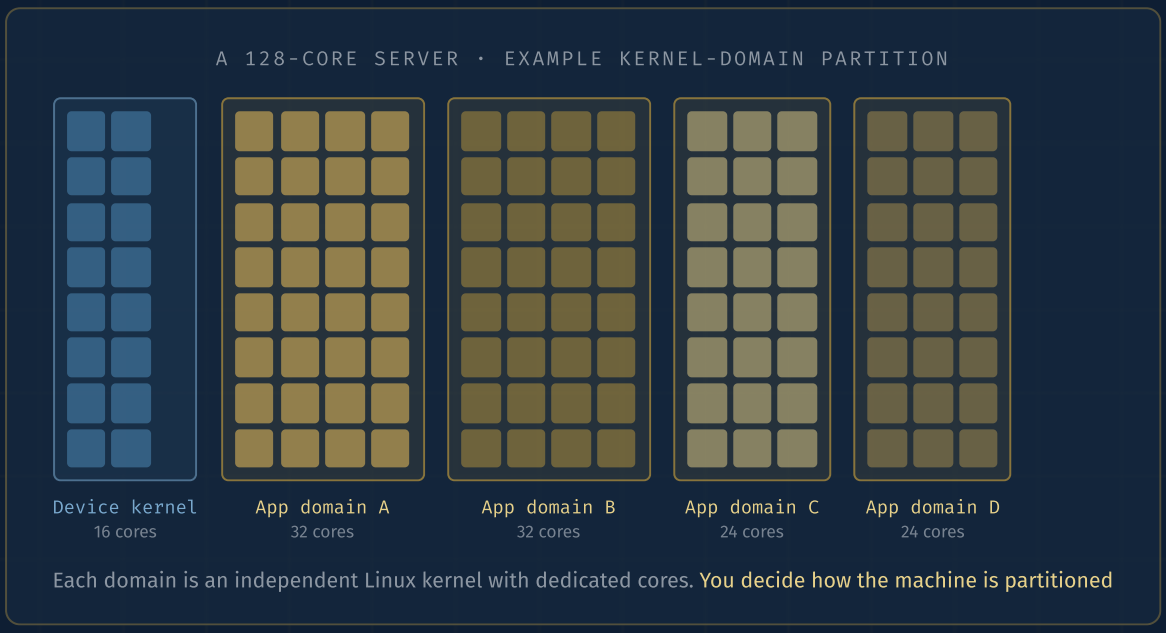
Built on upstream Linux · 100% open source · Docker-compatible · Runs on your own hardware

multikernel.io · github.com/multikernel
cwang@multikernel.io

SCALABILITY

More Cores, More Advantage

On a traditional single kernel, every core shares one kernel: the more cores, the heavier the global lock contention and cross-core cache-coherence traffic, and scale becomes a liability. Multikernel partitions large many-core servers into share-nothing kernel domains: **each kernel manages only its own subset of cores, so scalability is guaranteed by the architecture itself**.



Lock contention doesn't grow with scale

Kernels share nothing: 128 cores stay as calm as 8, with no re-tuning for bigger machines

Faults and interference stay in their domain

An overloaded or crashed domain never takes down the machine; everything else keeps running

Full value from every core

More cores means more independent domains per machine: higher density, higher utilization

ARM PLATFORM · ADAPTATION & PARTNERSHIP

128-Core ARM Servers: The Ideal Stage for Split-Kernel

High-core-count ARM platforms offer abundant cores and leading energy efficiency, a natural match for split-kernel architecture whose advantages grow with every added core. We propose a three-step path to partnership:

01

Platform Adaptation

Joint bring-up on your target machines: kernel boot, device kernel, and driver enablement covering network, storage and GPU

02

Proof of Concept

Validate performance, isolation and operational gains on your real workloads, side by side with your current virtualization stack

03

Upstream & Ecosystem

Push adaptation work into upstream Linux, publish joint results, and build the high-core-count ARM software ecosystem together

Remove the virtualization layer
Kernel-native performance, isolation and availability

Built on upstream Linux · 100% open source · Docker-compatible · Runs on your own hardware

multikernel.io · github.com/multikernel
cwang@multikernel.io

RISC-V PLATFORM · HETEROGENEOUS CORES

Heterogeneous RISC-V Cores: One Machine, No Compromise

RISC-V servers increasingly mix core types on one die: scalar cores for general compute, wide-vector cores for AI. Their vector contexts are incompatible (different VLEN, RVV 0.7.1 vs 1.0), so a single kernel can never migrate a task between them. Today's answer is to fuse off vector units or hide cores in firmware, throwing silicon away. **Multikernel never migrates tasks across kernels, so the problem disappears by construction.**

ONE KERNEL FOR ALL CORES

- ✗ The scheduler assumes identical cores; a task holding 512-bit vector state cannot resume on a 128-bit core
- ✗ The kernel is built for the lowest common denominator of every cluster
- ✗ Vendors disable vector units or partition cores in firmware, and paid-for silicon sits idle

ONE KERNEL PER CORE CLUSTER

- ✓ Each kernel sees a homogeneous machine and uses the full vector width of its cores
- ✓ Per-domain kernel builds: exact -march flags, vendor errata, even RVV 0.7.1 and 1.0 side by side
- ✓ Explicit placement: vector-heavy services on wide-VLEN cores, everything else on scalar cores

A HETEROGENEOUS RISC-V SERVER · ONE KERNEL PER CLUSTER

Device kernel	General domain	General domain	AI domain
8 scalar cores	16 cores · RVV VLEN 128	16 cores · RVV VLEN 128	16 cores · RVV VLEN 512

Wider cells, wider vector registers. Each cluster runs a kernel built for its exact ISA: nothing gets fused off

No migration bug class
Tasks never cross kernel boundaries, so incompatible vector contexts can never meet

Full silicon utilization
Every core runs at its full vector width; no lowest common denominator, nothing disabled

One box, two worlds
RVV 0.7.1 and 1.0 userspaces coexist on one machine, each in its own kernel domain

PARTNERSHIP The same three-step path as ARM applies to RISC-V: platform adaptation on your target silicon, proof of concept on real workloads, then upstream and ecosystem work together.

Remove the virtualization layer
Kernel-native performance, isolation and availability

Built on upstream Linux · 100% open source · Docker-compatible · Runs on your own hardware

multikernel.io · github.com/multikernel
cwang@multikernel.io