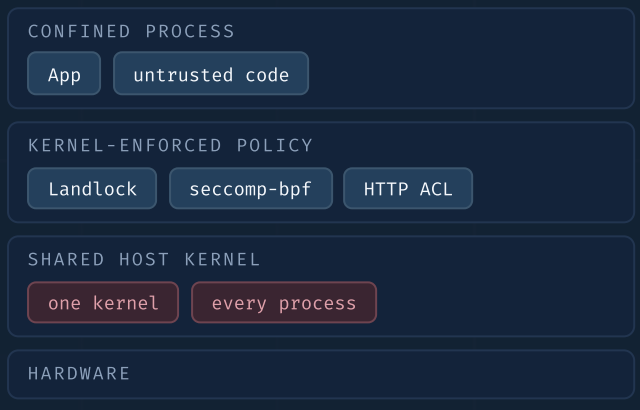
Four Ways to Isolate a Workload
Sandbox · Container · Multikernel · Virtual Machine

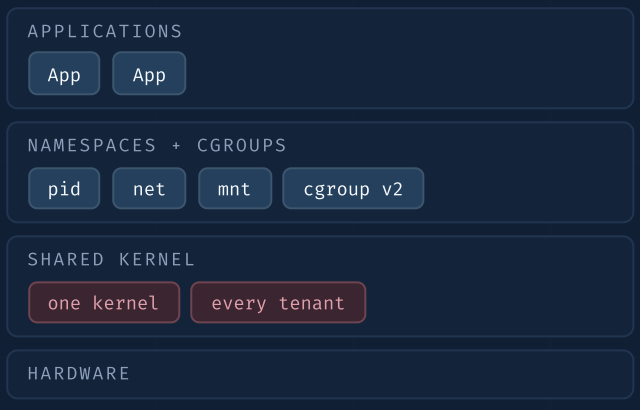
Every isolation technology draws its boundary at a different layer. Sandlock wraps kernel-enforced policy around a single process. Containers group processes in namespaces on a shared kernel. Multikernel gives each workload its own real kernel on dedicated cores. Virtual machines put a guest kernel on virtual hardware. **The question is not which one is best, but where you need the boundary and what you are willing to pay for it.**

PROCESS SANDBOX · SANDLOCK



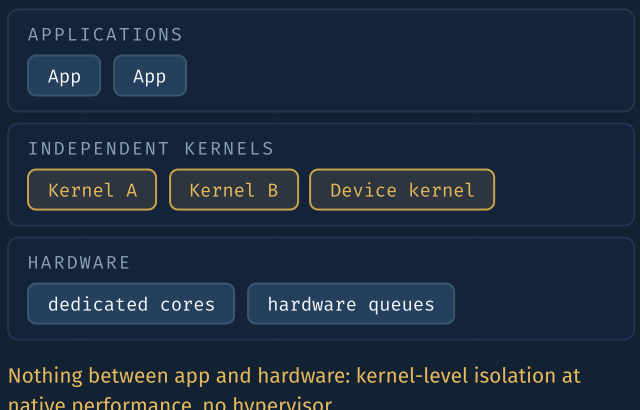
Policy around one process: no root, no images, millisecond startup. The kernel itself stays shared.

CONTAINERS



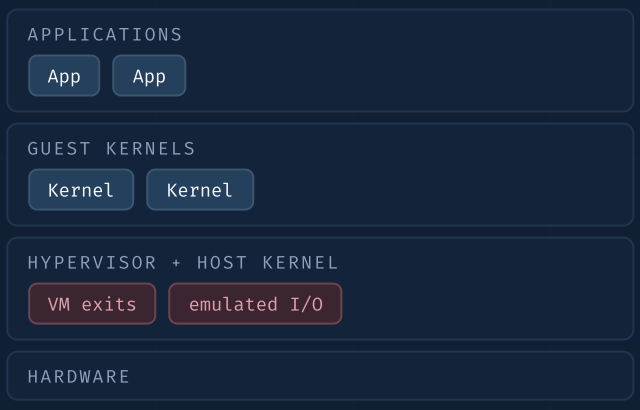
One kernel shared by every tenant: one kernel bug away from every other workload.

MULTIKERNEL · SPLIT-KERNEL



Nothing between app and hardware: kernel-level isolation at native performance, no hypervisor.

VIRTUAL MACHINES



Strong isolation, but two kernels and a hypervisor sit between the app and the hardware.

SANDLOCK	CONTAINERS	MULTIKERNEL	VIRTUAL MACHINES
Boundary: syscalls	**Boundary: namespaces**	**Boundary: the kernel itself**	**Boundary: virtual hardware**
No root, ~5 ms startup, shared kernel underneath	Minimal overhead, full kernel exposed to every tenant	Separate kernels, native performance, no hypervisor	Strong walls, 5-20% performance tax

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

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

CAPABILITY MATRIX

Side by Side

Sandlock and Multikernel solve different problems and the numbers below keep them apart: Sandlock confines a process on a shared kernel; Multikernel removes the shared kernel entirely. **Containers and VMs sit in between, trading isolation against overhead.**

CAPABILITY	SANDLOCK	CONTAINERS	MULTIKERNEL	VMS
Isolation boundary	Syscall policy (Landlock + seccomp)	Namespaces on a shared kernel	Separate kernels per workload	Hypervisor + guest kernel
Performance overhead	~3% (97% of bare metal)	Minimal	None	5-20%
Cold start time	~5 ms	~200 ms	~50 ms	~100 ms (microVM)
Privileges required	No root, no image build	Root or user namespaces	Owns the machine or VM	KVM access
Kernel customization	No (host kernel)	No	Yes, per workload	Yes
Attack surface per workload	Full kernel, syscalls filtered	Full kernel	Minimal (stripped kernel)	Reduced
Noisy neighbor effect	Yes (shared kernel)	Yes	Eliminated	Reduced
Zero-downtime kernel updates	N/A	N/A	Yes	Live migration
Fine-grained security policy	Files, network, syscalls, HTTP (method + host + path)	Coarse (seccomp / AppArmor profiles)	Compose with Sandlock	No
Works inside cloud VMs	Yes	Yes	Yes	Needs nested virtualization

Sandlock figures are from the Sandlock project benchmarks. VM and container figures are from the Multikernel technology overview.

GUIDANCE

When to Reach for Which

Sandlock

Untrusted code inside a trusted environment: AI agent tool calls, CI steps, generated code. Millisecond startup, no root, no images.

Containers

Packaging and orchestration for code you already trust. The ecosystem default; isolation is not the reason you choose them.

Multikernel

Kernel-level isolation at native speed: latency-sensitive services, mixed workloads on one machine, AI agents with full GPU access.

Virtual Machines

Established multi-tenant estates where a hypervisor is mandated, or guests that are not Linux at all.

DEFENSE IN DEPTH Boundaries compose. Multikernel gives every workload its own kernel; Sandlock confines individual processes inside it. An AI agent can run in its own app-kernel with direct GPU access while Sandlock gates its file, network and HTTP activity: **two independent enforcement layers, and still no hypervisor anywhere in the stack.**