

Multikernel vs. Containers

Same Cores, Separate Kernels, Measured

A container is a set of namespaces and cgroups on a kernel it shares with every other container on the host. Every lock, every reclaim pass and every panic in that kernel belongs to all of them. Multikernel gives each workload its own Linux kernel on dedicated cores and memory, with no hypervisor underneath. We held the hardware constant and changed only that boundary.

Here is what the shared kernel was costing.

2.6x

THROUGHPUT

unlink rate on the same 24 cores,
split across two kernels

153x

LOWER P99

beside a memory-hungry
neighbor: 35.6 s down to 232 ms

0 bytes

LOST TO PANICS

4 kernel panics mid-write;
memory intact after every one

~50 ms

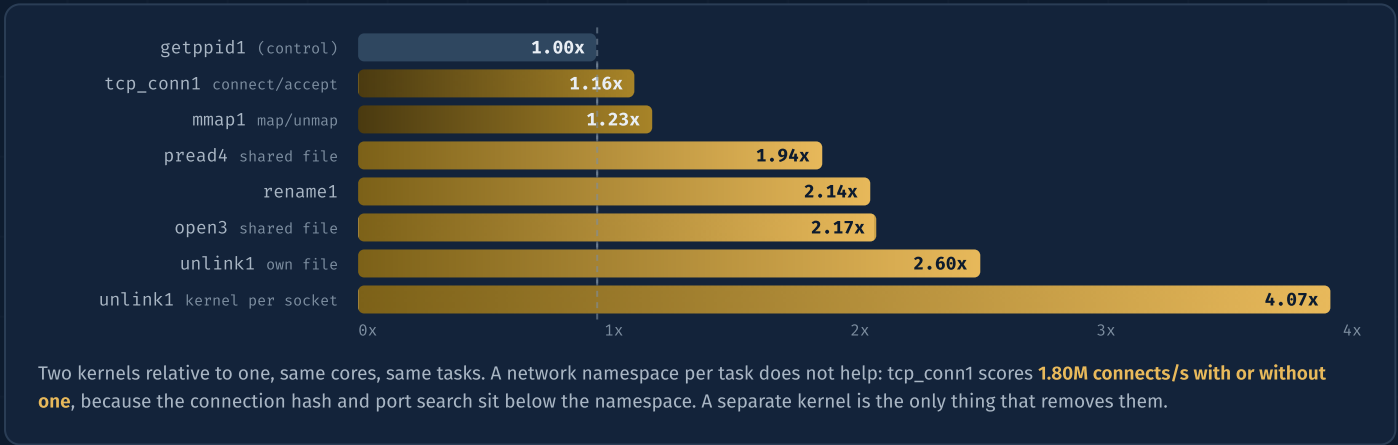
COLD START

a full kernel boot, against ~200 ms
for a container

THROUGHPUT · WILL-IT-SCALE

The Locks Stop Being Shared

Namespaces change what a process can see, not the kernel data structures underneath it. On a 48-core Xeon, 48 processes each creating and deleting *their own* file sustain 188,000 ops/s in total: 40% of what one core does alone. Below, the same 24 cores run the same tasks, as 24 on one kernel or 12 + 12 on two. **The control tests stay at 1.00x, so the gain is the kernel boundary and nothing else.**



TAIL LATENCY · NOISY NEIGHBOR

Containers Pick a Victim. Multikernel Runs Both.

A latency-sensitive task on 4 dedicated cores beside a batch job filling memory on 20 others. They share no cores and no memory, only the kernel's reclaim. The container answer, memory.high on the neighbor's cgroup, rescues the task by **throttling the neighbor about 3,000x**. Neighbor work counts full passes over its memory in 300 s.

CONFIGURATION	P50	P99	NEIGHBOR WORK
Shared kernel, idle machine	56 ms	92 ms	n/a
Shared kernel, beside neighbor CONTAINERS	824 ms	35.6 s	29,719
Shared kernel, cgroup memory.high CONTAINERS	58 ms	91 ms	10
Two kernels, beside neighbor MULTIKERNEL	208 ms	232 ms	27,501
Two kernels, neighbor on other node MULTIKERNEL	59 ms	67 ms	30,493

FAILURE DOMAIN • DAXFS

A Kernel Panic Is Not a Host Outage

On a shared kernel, a panic is a memory-loss event for every tenant at once: each cache, index and working set starts over cold. Multikernel shrinks the failure domain to one instance, and lets memory outlive the kernel that wrote it.

CONTAINERS

One kernel, one blast radius

A bug in any tenant's syscall path, driver or filesystem panics the host kernel. **Every container on the machine goes down together**, and everything they held in RAM is gone. Recovery means rebooting the host and re-warming every cache from scratch.

MULTIKERNEL

The kernel crashed, the memory didn't

An app-kernel wrote a file on DAXFS through a plain mmap, then panicked mid-write, **four times in a row**. Each time a fresh kernel booted on the same memory and found every byte where the dead one left it, down to a single half-written page. Other instances kept running. **No log to replay, nothing to reload: recovery was a boot.**

CAPABILITY MATRIX

Side by Side

CAPABILITY	CONTAINERS	MULTIKERNEL
Kernel	One, shared by every tenant	One per workload, on dedicated cores and memory
Shared-lock syscalls (unlink, rename, stat, open)	Contend machine-wide; 48 cores deliver 40% of one on unlink	1.9x to 2.6x more throughput on the same cores
Connection setup	Hash and port locks survive a netns per task	1.16x, nothing to tune
Neighbor under memory pressure	p99 35.6 s, or throttle the neighbor 3,000x	p99 232 ms (67 ms with NUMA placement), neighbor at full speed
Kernel panic blast radius	Every container on the host	One instance; DAXFS memory intact
Cold start time	~200 ms	~50 ms
Steady-state overhead	Minimal	None, native execution, no hypervisor
Kernel version and tuning	Host kernel only	Chosen per workload
Kernel updates	Reboot the host and every tenant	Zero-downtime
Packaging	OCI images	Docker-compatible

FOR INFERENCE SERVING

Where It Shows Up in an Inference Stack

API and routing tier

Short-lived connections at high rates hit the connection hash and port locks that namespaces cannot shard. Separate kernels can.

Serving beside batch

Evals, checkpoint loading and data prep share the box with latency-critical serving, without making either one the victim.

Warm state across crashes

Host-memory caches, such as tiered prefix caches spilled to DRAM, outlive an app-kernel panic. Recovery is a boot, not a cold refill.

Direct device access

Each instance owns its cores, memory and devices outright. No hypervisor and no virtual NIC between the workload and the hardware.

Throughput and tail-latency figures: dual-socket Intel Xeon Gold 5418Y, 24 cores per socket, same kernel build in every configuration; only the kernel boundary changes. Sources on [multikernel.io/blog](#): "The Fastest Lock Is One You Don't Share" (2026-08-17), "The Noisy Neighbor Is the Kernel" (2026-08-25), "The Kernel Crashed, but the Memory Didn't" (2026-09-30). DAXFS memory is DRAM and does not survive power loss. Cold start figures from the [Multikernel technology overview](#).