CHERITech'26
CONFERENCE

CHERI

12-13 Nov 2026
Munich, DE

SignetOS

Designing a Secure and Fast Single-Address-Space Microkernel to Support Modern Workloads

John Alistair Kressel
Google



Located at semi | SEMICON EUROPA

Goals

Fine-Grained Compartments

Low compartment switching overhead

- **Isolate components:** drivers, libraries, and OS services (FS, scheduler)
- **Isolate software at fine granularity**
- **Clean fault recovery:** compartment local fault handling

Sandboxing Untrusted Code

Explicit capabilities for tools & AI agents

- **Per-task compartments:** run each tool or agent isolated
- **No ambient names:** sealed handles for files & sockets
- **Bounded budgets:** memory, CPU, disk, and network quotas

Template-Based Cloning

Fork-less COW in one address space (lazy mem copy)

- **Beyond Zygote fork:** no duplicate address spaces
- **Relative addressing:** clone language runtime templates
- **Instant page sharing:** copy-on-write without pointer fixups

Eventual goal to run on CHERI-enabled Coral silicon

Core Design

Access — Access to objects is via **sealed, opaque handles**.

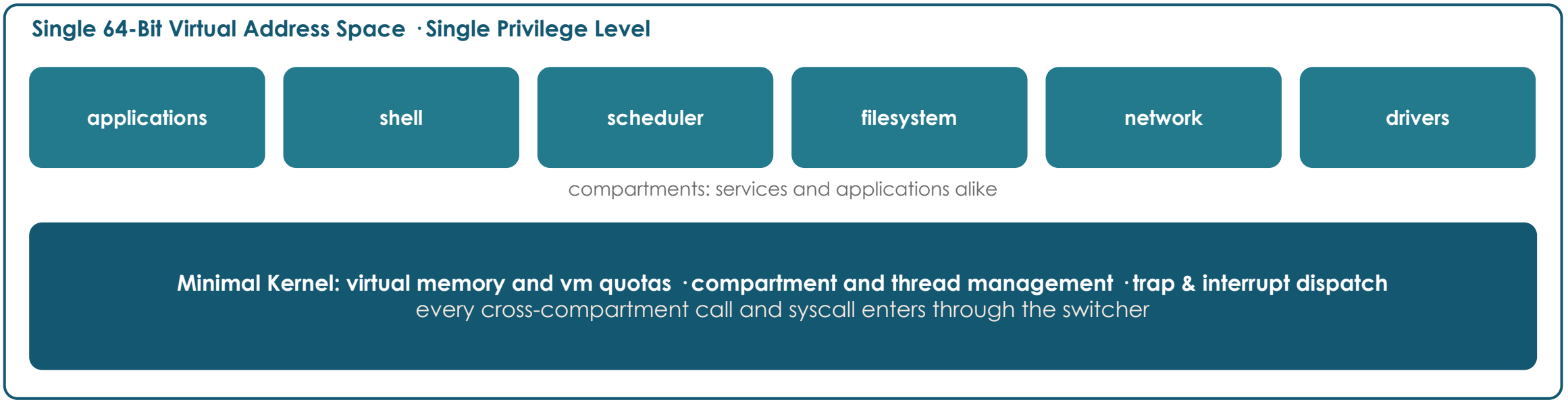
↓

Funding — Objects refer to resources which are funded by **hierarchical quotas**.

↓

Management — Resources are managed by **resource manager compartments**.

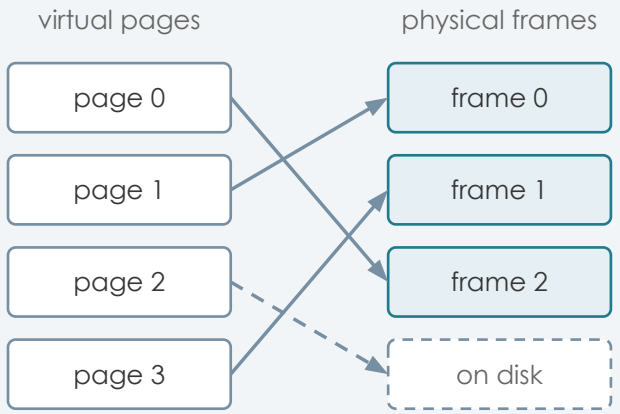
Architecture Overview



1. Single Address Space
2. Compartments, Threads & Switcher
3. Sealed Handles
4. Hierarchical Quotas
5. Minimal Kernel

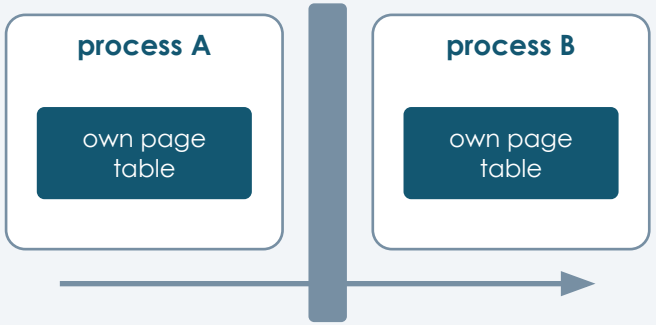
The MMU Conflates Translation and Protection

MMU Role 1: Address Translation



Keep MMU for virtual memory and paging

MMU Role 2: Isolation

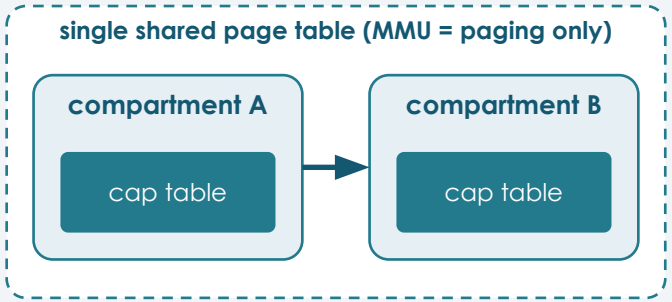


address-space boundary

trap · switch page tables · flush TLB · copy

Per-process page tables make crossings costly

SignetOS: Decouple the Two



capability switcher crossing

no page-table switch

MMU translates pages; CHERI capabilities isolate

Conventional OSes turn every protection boundary into an address-space boundary. SignetOS decouples protection and translation.

Compartments, Threads and Switching

Compartments

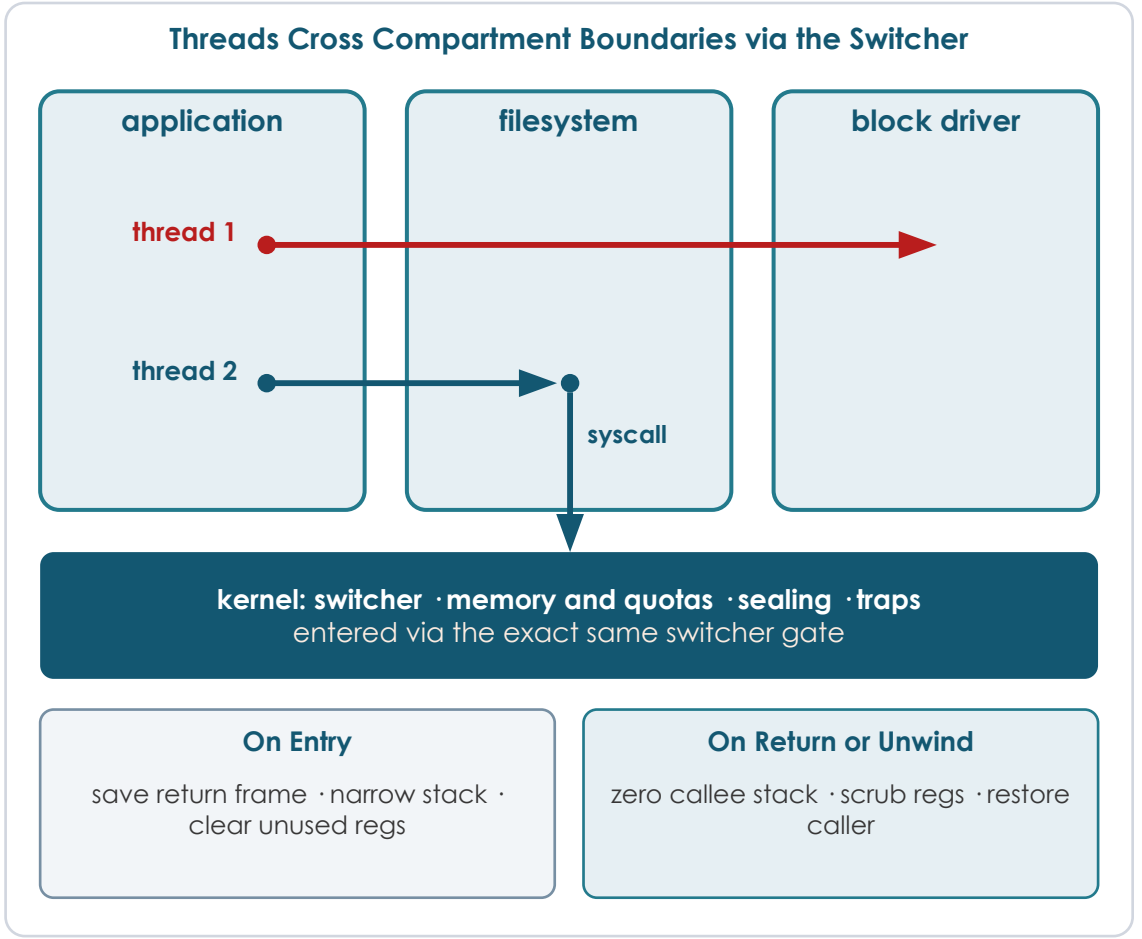
- Created and destroyed dynamically by kernel.
- Contain code, data and capabilities, entered via switcher through sealed entry points.

Threads

- Managed by kernel, scheduled by scheduler compartment.
- Registers, call stack, and trusted stack.
- Cross compartments boundaries through switcher.

Switching

- Invokes sealed entry handle; narrows and shifts stack and clears regs.
- Zeroes callee stack on return; syscalls use the same gate.



Sealed Handles

Why Data Sealing Is Essential

- **Unforgeable token:** sealed capabilities point directly at objects such as file or thread, only resource manager can unseal.
- **Sealing:** hardware sealing types to identify object types.
- **Zyseal (RVY):** critical to enable sealing beyond code sentries!

Hardware Types

Type in capability metadata (otype)

- **Core objects:** compartments, threads, entries, quotas

Virtual Types

Sealed key in object header

- **Unlimited types:** minted dynamically by any compartment

Hierarchical Quotas

Derive Sub-Quotas

Can be used directly or derive smaller child quotas

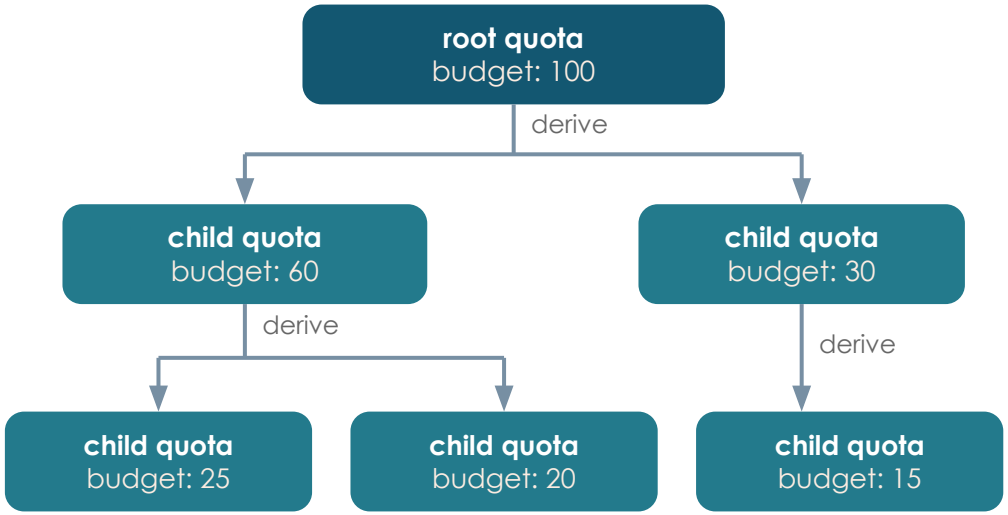
Subtree Reclamation

Destroying a node closes its subtree and refunds unused quota to parent

Caller-Funded Nodes

Billed to caller memory quota

Monotonic Quota Derivation Tree (sum of children ≤ parent)



Quota	Manager	Resource Dimension
quota_vm	kernel	virtual memory
quota_thread_mem	kernel	thread metadata and stacks
quota_sched	scheduler	CPU bandwidth
quota_disk	filesystem	disk space
quota_net	network	bandwidth

scheduler, filesystem, and network run as unprivileged compartments

Minimal Kernel

Monolithic POSIX Kernel

All OS Subsystems in Privileged Kernel Space

- **Full privilege:** scheduler, VFS, network, drivers, IPC
- **Global state:** kernel heaps and ID lookup tables
- **Large TCB:** any kernel bug compromises the machine

SignetOS Kernel

Few Primitives in Kernel

- **Minimal kernel:** memory/paging/quotas, sealing, traps/interrupts, compartment and thread creation and destruction
- **Minimal State:** kernel does not track compartments or threads
- **Non-Kernel Compartments:** scheduler, filesystem, drivers, network etc

Moving mechanisms into compartments leaves a minimal kernel small enough for future formal verification.

Prototype Implementation Status

Mechanisms

Compartments, Threads & Switcher
Dynamic domains & single gate

Sealed Handles
Hardware & virtual types

Quotas
Caller-funded hierarchy

Faults & Unwinding
Per-compartment handlers

Paging & Revocation
Quarantine & capability sweeps

Multi-Core Initialization
SMP boot on CHERI RISC-V

Isolated Compartments

Scheduler

Filesystem

Device Drivers

Network

Shell

Everything runs on CHERI RISC-V QEMU.

Related Systems

CheriBSD	CHERIoT RTOS	CheriOS
CHERI for existing UNIX software	Compartments for low-cost embedded devices	Mutual distrust, including the OS

SignetOS

Fine-grained isolation and resource control for dynamic workloads

Contained

Untrusted code can use only the handles it holds, and consume only what its quotas fund

Every resource budgeted

Quotas for memory, allocator, CPU, disk and network, split and reclaimed at run time

One call path

Every call, syscalls included, runs on the caller's thread through one switcher

Small trusted base

Only the kernel and resource managers are trusted, kept small enough to verify

Summary

Decouple Translation and Protection

- **MMU for virtual memory only**
- CHERI capabilities enforce all compartment isolation

Fast Domain Switching

- **Threads cross compartments via the switcher**
- No page-table switch, TLB flush, or IPC copy

Hierarchical Quotas

- **Sealed handles subdivide memory, CPU, disk, network**
- Caller-funded tree nodes with no central kernel ledger

Minimal Kernel

- **Only switching, memory/sealing primitives, and traps**
- All services and drivers run as ordinary compartments

Find Out More

Find SignetOS on GitHub

github.com/OpenSeCura/SignetOS

Learn about Coral

coral.ai

CHERITech'26 CONFERENCE

12-13 Nov 2026
Munich, DE

Located at semi | SEMICON EUROPA

CHERI

THANK YOU

Questions

CHERI™ and the CHERI logo are protected trademarks. Use of these marks is subject to the CHERI Alliance Trademark Policy – https://cheri-alliance.org/legal-and-policies/trademark/

This work © 2026 by CHERI Alliance is licensed under CC BY-SA 4.0 (Creative Commons Attribution-ShareAlike 4.0 International) – https://creativecommons.org/licenses/by-sa/4.0/