Timing Channels in OS Design

Comparing S3K and seL4 Time Protection

Operating Systems Comparison

February 12, 2026

Outline

1. S3K: Hardware-Assisted Isolation
2. seL4: Time Protection Abstraction
3. Comparison: S3K vs. seL4

S3K: Overview

- **Target:** Embedded RISC-V systems.
- **Goal:** A capability-based multicore partitioning kernel for Real-Time Systems.
- **Core Philosophy:**
  - Uses RISC-V hardware features (specifically `fence.t`) to enforce temporal isolation.
  - Minimal Trusted Computing Base (TCB) acting as a bare-metal hypervisor.
- **Key Attributes:**
  - Capability model with bounded Worst-Case Execution Time (WCET).
  - Time-driven scheduler.
  - Predictable constant-time system calls

S3K: Timing Isolation Mechanisms

S3K employs a hardware-centric approach to prevent timing interference:

The `fence.t` Instruction

- Used at context switches to flush core-local microarchitectural state.
- Prevents state (like branch predictors or L1 cache) from leaking data to the next process.

Scheduling & Padding

- **Major/Minor Frames:** Time is strictly partitioned.
- **Padding:** The scheduler uses worst-case padding to ensure strict deterministic process dispatch.
- **Execution:** Constant-time kernel operations prevent the kernel itself from becoming a timing channel.

S3K: Memory Hierarchy & Limitations

Scratchpad Memory (SPM) Focus:

- S3K is designed to reside entirely within the SPM (backed by L1).
- **Benefit:** Enables efficient flushing of the kernel's footprint via `fence.t`.

Scope of Protection:

- **In-Kernel:** Protected. All syscalls are non-interfering and constant-time.
- **User-Level:** Not fully protected by the kernel.
- Unlike seL4, S3K does not enforce cache coloring for user space. Processes must protect their own user-level state from side-channels in higher-level caches (L2/DRAM)

seL4: The Missing OS Abstraction

- **Problem:** Standard memory protection (spatial) is insufficient. Shared microarchitectural state (caches, TLB, branch predictors) creates timing channels.
- **Solution: Time Protection** as a first-class OS abstraction.
- **Definition:** Preventing interference between security domains such that execution speed in one domain is independent of another

Threat Model

Addresses both **Side Channels** (victim leaks to attacker) and **Covert Channels** (colluding sender/receiver).

seL4: Implementation Mechanisms

seL4 introduces software mechanisms to overcome hardware limitations:

1. **Kernel Cloning:**
   - Replaces static partitioning with dynamic kernel clones.
   - Each domain gets a private copy of OS text/stack to prevent kernel state leakage.
2. **Deterministic Flushing:**
   - On domain switch: flush L1, TLB, BP.
   - **Crucial:** Operations are padded to worst-case latency to hide the cost of the flush itself.
3. **Cache Coloring:**
   - Partitions the Last Level Cache (LLC) by locking pages to specific cache sets.
   - Prevents cross-domain interference in shared memory hierarchies.

Comparison: State Clearing Strategy

S3K (Hardware-Centric)

- Relies on RISC-V `fence.t`.
- **Pros:** Efficient hardware implementation; single instruction handles local state.
- **Cons:** Requires specific hardware support (RISC-V extensions); focuses mostly on core-local state.

seL4 (Software-Centric)

- Manual software flushing sequences (e.g., iterating cache lines) + Padding.
- **Pros:** Works on commodity hardware (x86/Arm) by abstracting the lack of flush instructions.
- **Cons:** Higher overhead due to manual flushing and padding requirements.

Comparison: Memory Hierarchy Protection

How they handle the cache hierarchy (L2/LLC) differs significantly:

S3K	**seL4**
Optimized for **SPM/L1** locality.	Optimized for **Deep Hierarchies** (L2/LLC).
Does **not** provide inherent protection for shared upper-level caches (DRAM/L2).	Uses **Cache Coloring** to spatially partition shared caches.
"User processes are responsible for protecting their own user-level state."	*"OS enforces isolation via coloring to prevent contention."*

Comparison: Kernel Architecture

S3K: The Hypervisor Model

- Single kernel instance acting as a bare-metal hypervisor.
- Relies on constant-time implementation of syscalls to prevent leakage.
- **Focus:** Determinism via scheduling and instruction padding.

seL4: The Kernel Clone Model

- **Kernel Cloning:** Creates separate kernel images per domain.
- Spatially separates global kernel data (stacks, variables) to eliminate shared state channels.
- **Focus:** Spatial separation of the kernel itself to aid temporal isolation

Summary of Differences

Feature	S3K	seL4
Primary Mechanism	`fence.t` (RISC-V)	Software Flush + Padding
Kernel State	Single Constant-Time Kernel	Cloned Kernel Images
Shared Cache	User Responsibility	Cache Coloring (OS Managed)
Isolation Scope	Core-local (L1/SPM)	Full Hierarchy (L1-LLC)
Target	Embedded Real-Time	General Purpose / Cloud