CantripOS: An OS for AmbientML Applications

June Tate-Gans (Google), Sam Leffler (Google), Kai Yick (Google)

Google Research

Agenda

01 Sparrow Project

02 Matcha System

03 CantripOS

04 Future Work

Google Research

01

Sparrow Project

Hardware/Software Co-design for ML

Machine Learning (ML) operates on a wide variety of inputs and at a wide range of scales

Co-design enables us to **speed up the iteration** loops on both hardware and software

Simulation is crucial here as it enables us to modify hardware at the speed of software

Unlocking the **entire system stack** allows better design for maximum security and efficiency gains

Privacy for ML, through security

Privacy, Security, and Comfort are critical for developing user trust, and user trust is a major motivating force behind ML adoption.

Protecting user privacy and security is a responsibility that comes first. This means always being thoughtful about what data we use, how we use it, and how we protect it

It is essential to consider and address the security of an AI system before it is widely relied upon in safety-critical applications.

In practice, researchers and developers must iterate to find an approach that appropriately balances privacy, security and utility for the task at hand.

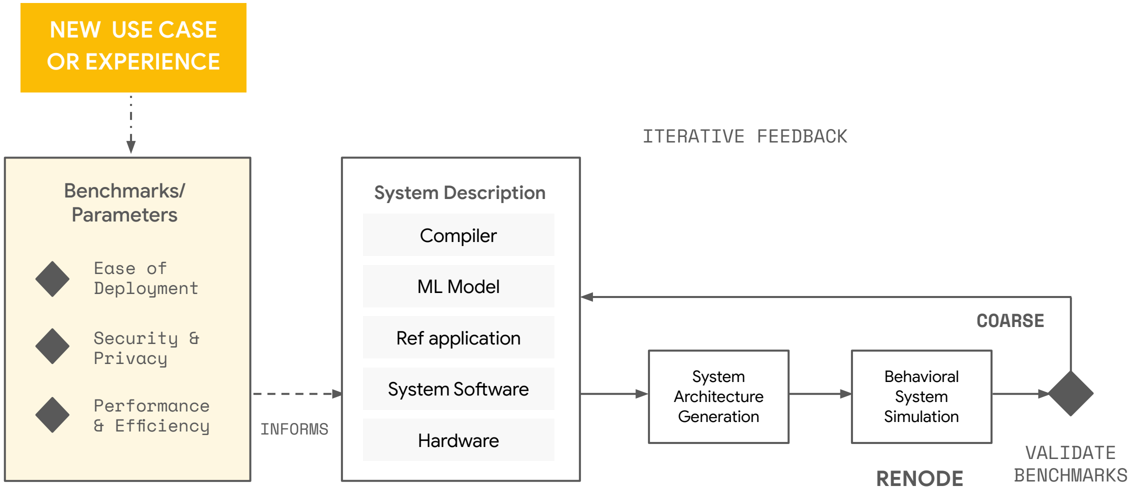
Sparrow

A set of open source tools and infrastructure for co-design of HW, SW & ML systems that enable secure scalable efficient neural compute.

North star goals

- **Enable new use cases** through increased security, transparency and system efficiency.
- **Reduce deployment time** from initial model to complete system.
- **Open Source transparency** is key to trust in security & privacy promises.

Sparrow Co-design Process - Simulation

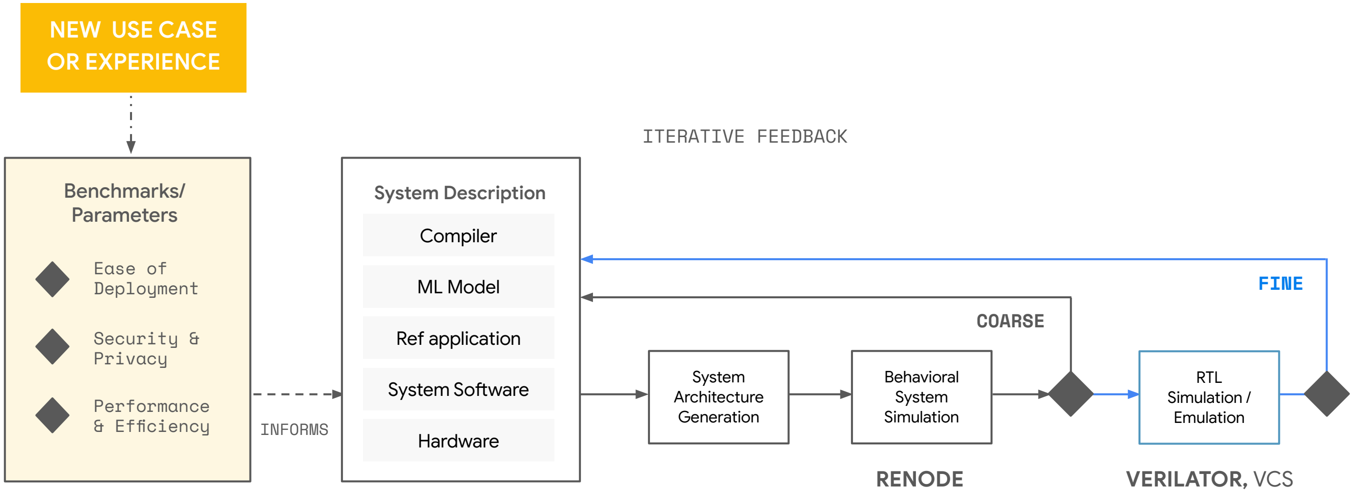


NEW USE CASE OR EXPERIENCE

Benchmarks/ Parameters

System Description

Sparrow Co-design Process - Simulation



NEW USE CASE OR EXPERIENCE

Benchmarks/ Parameters

- Ease of Deployment
- Security & Privacy
- Performance & Efficiency

INFORMS

System Description

- Compiler
- ML Model
- Ref application
- System Software
- Hardware

ITERATIVE FEEDBACK

FINE

COARSE

System Architecture Generation

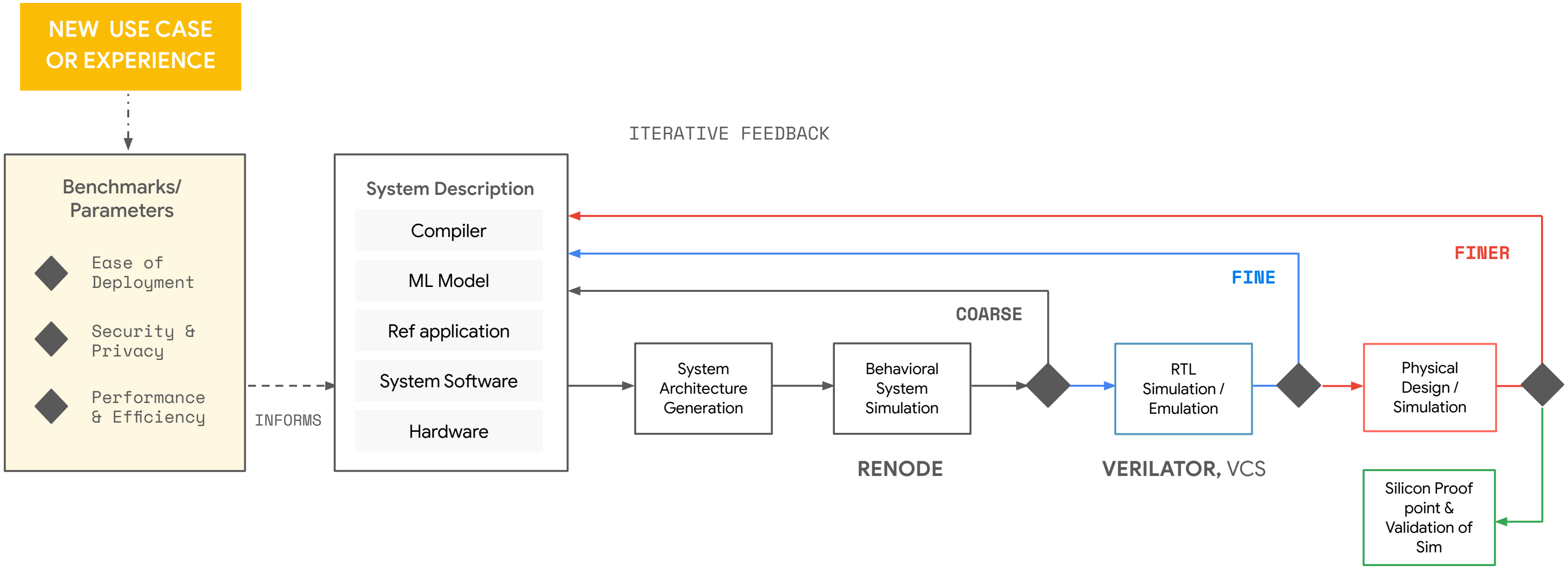
Behavioral System Simulation

RTL Simulation / Emulation

RENODE

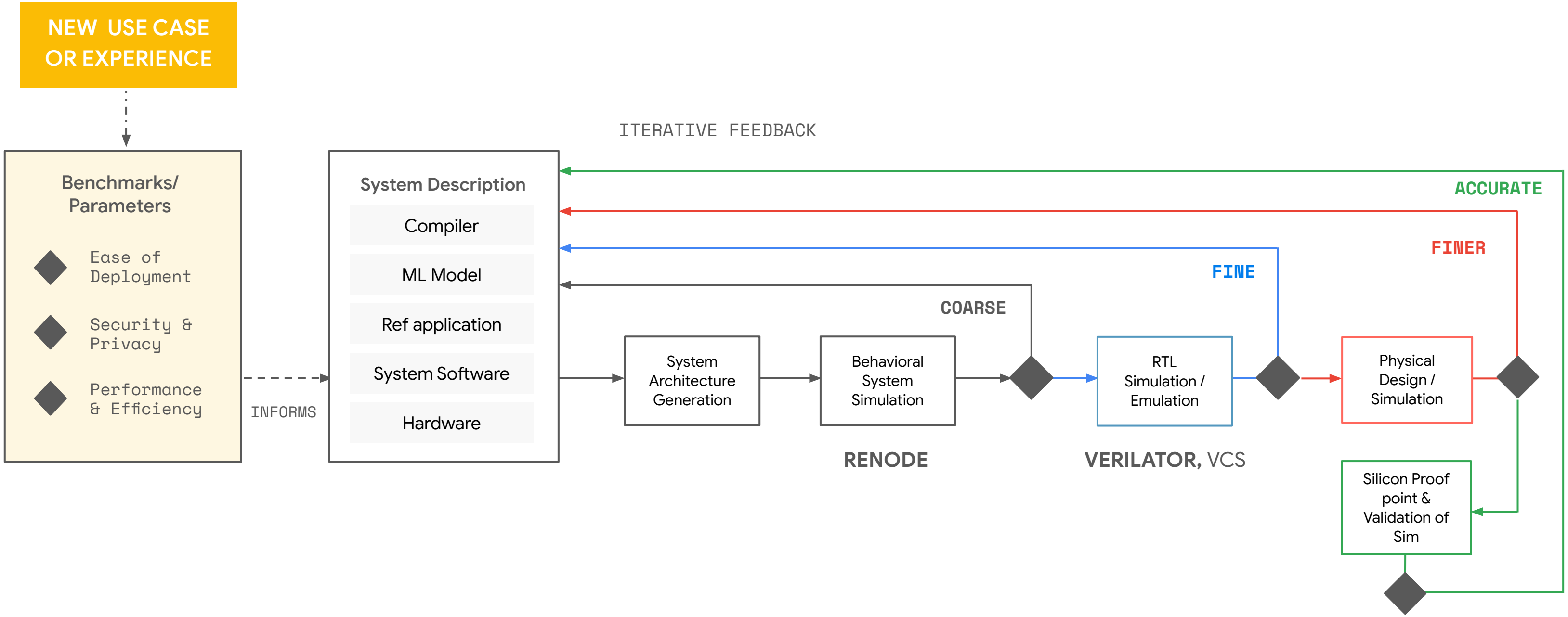
VERILATOR, VCS

Sparrow Co-design Process - Simulation



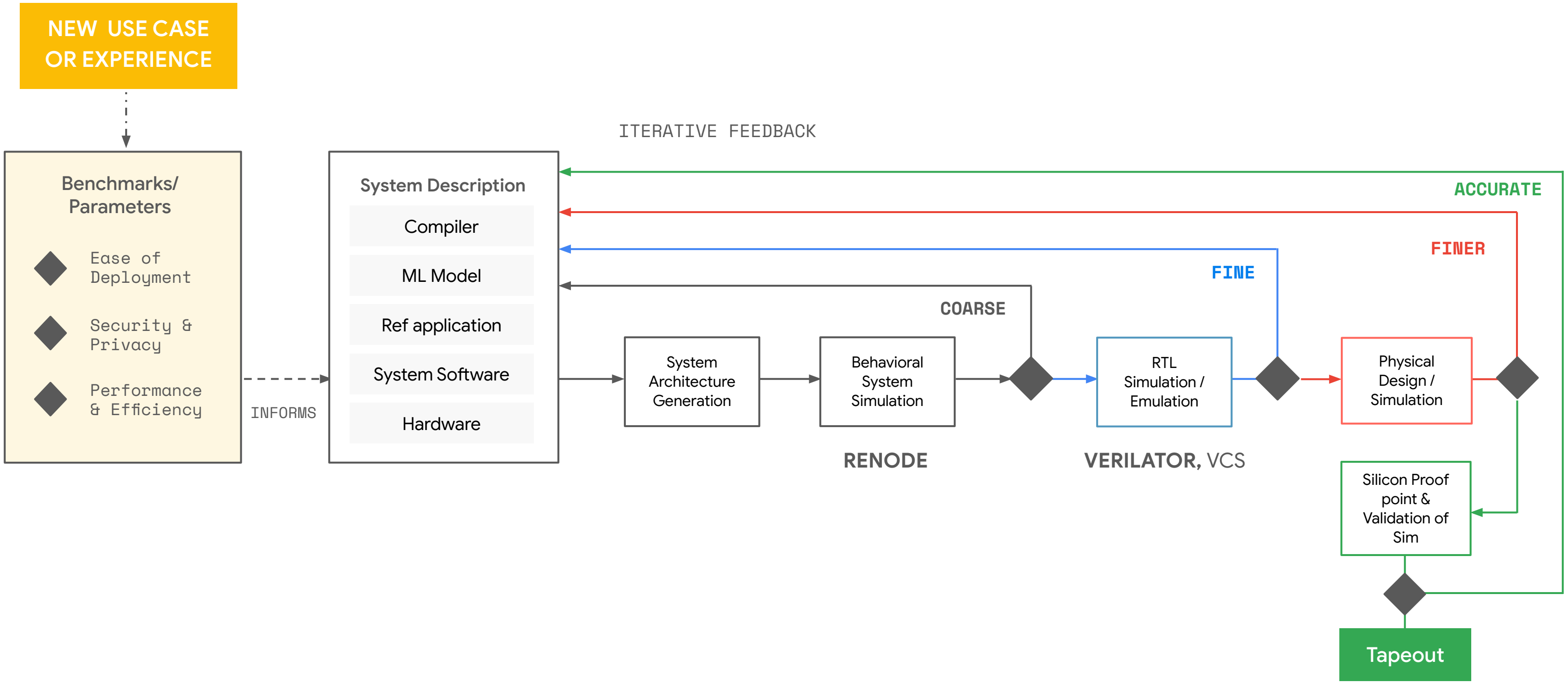
Google Research

Sparrow Co-design Process - Build



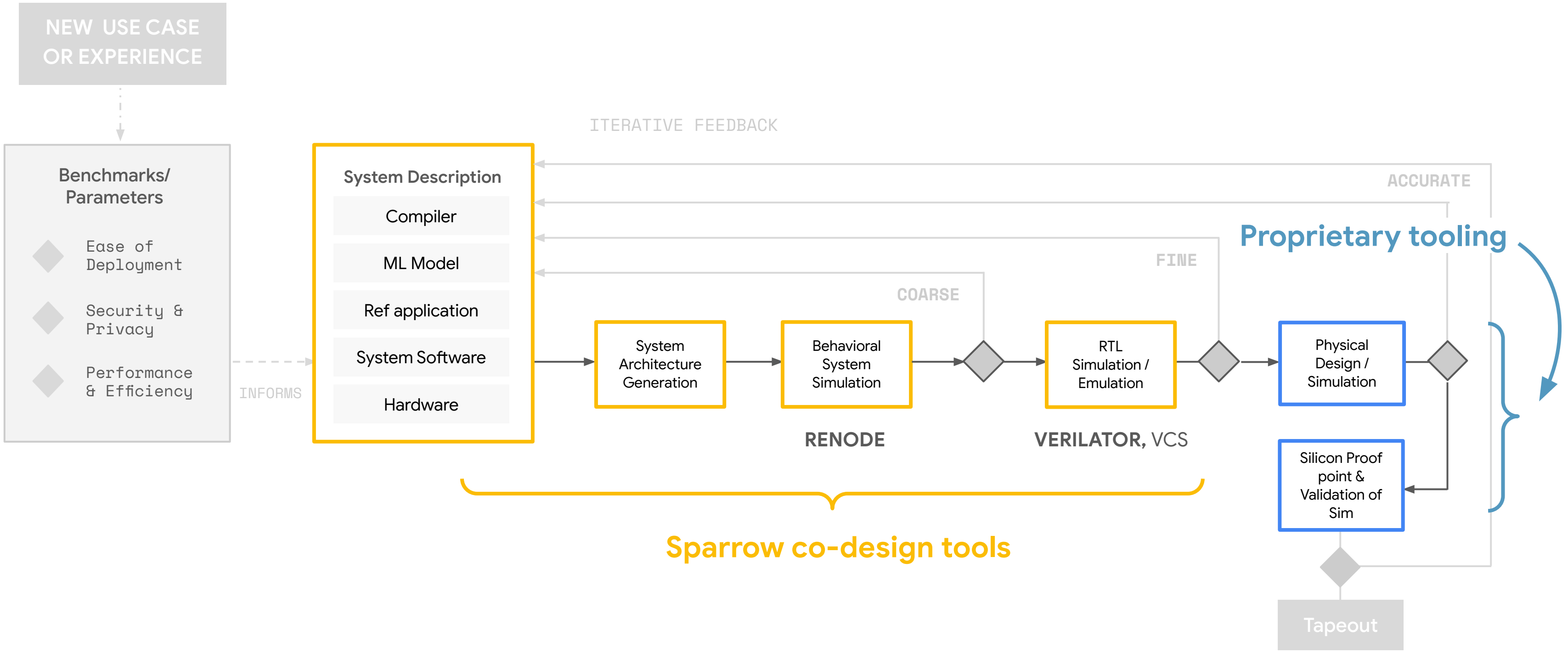
Google Research

Sparrow Co-design Process - Build



Google Research

Sparrow Co-design Process - Toolbox



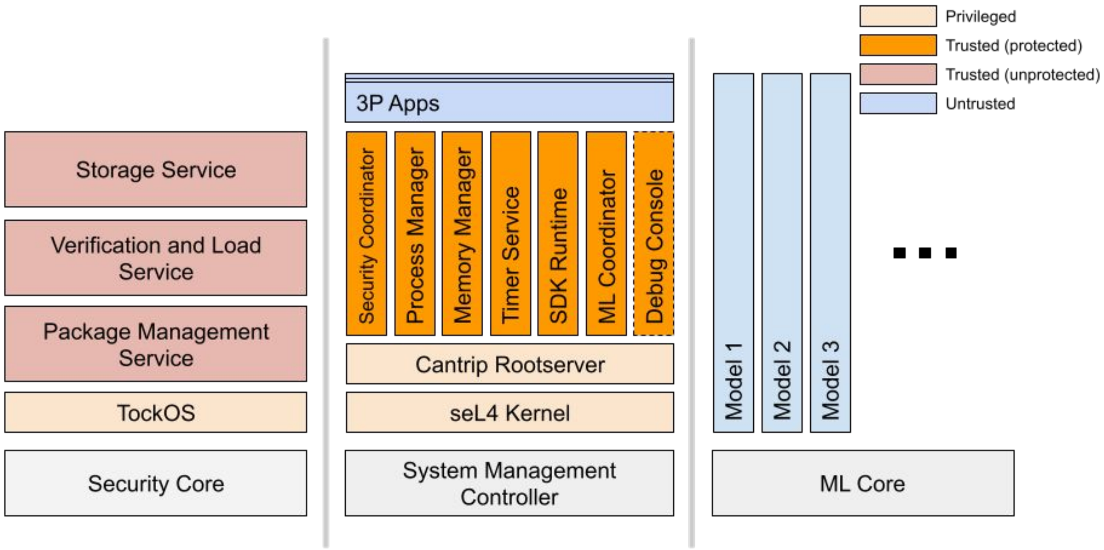
Google Research

02

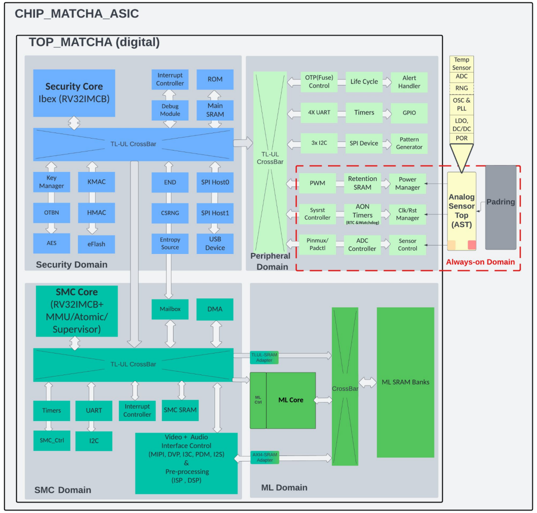
Matcha System

Matcha: Preliminary Sparrow reference implementation

Objective: Build a secure, low power ambient perception and sensing system

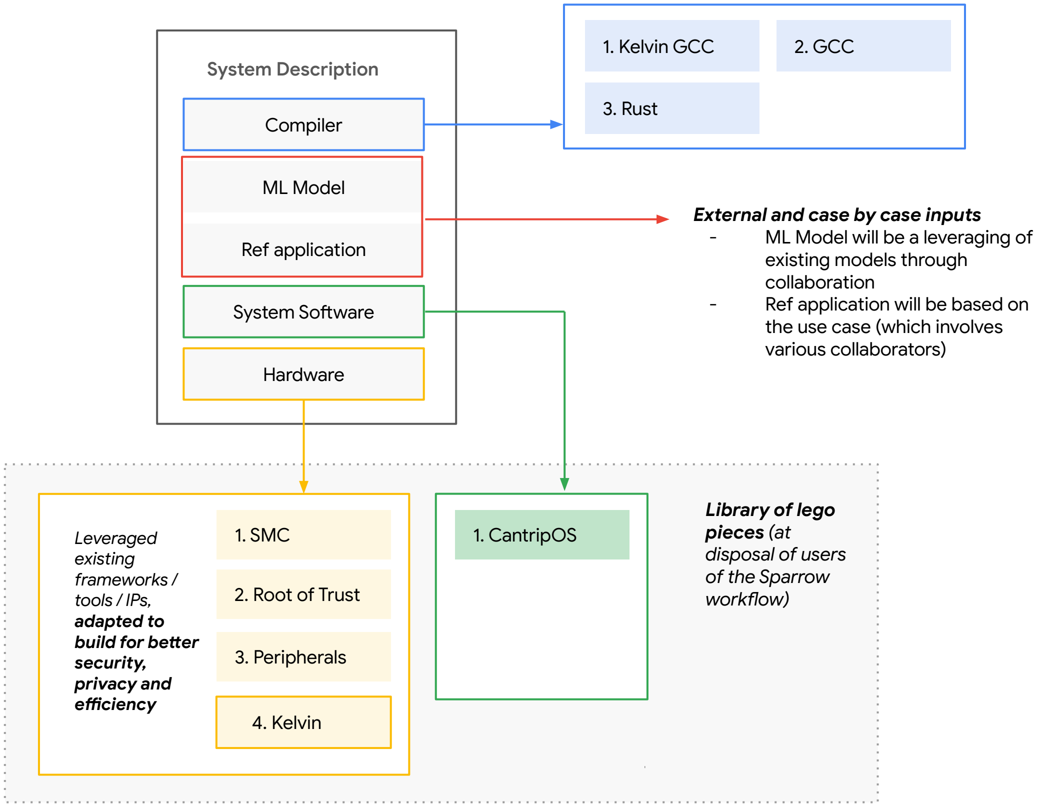


High Level System Architecture



Block Diagram

Matcha Artifacts (and zoo) (System Reference Design)



System Description

Compiler

ML Model

Ref application

System Software

Hardware

1. Kelvin GCC

2. GCC

3. Rust

External and case by case inputs

- ML Model will be a leveraging of existing models through collaboration
- Ref application will be based on the use case (which involves various collaborators)

*Leveraged existing frameworks / tools / IPs, **adapted to build for better security, privacy and efficiency***

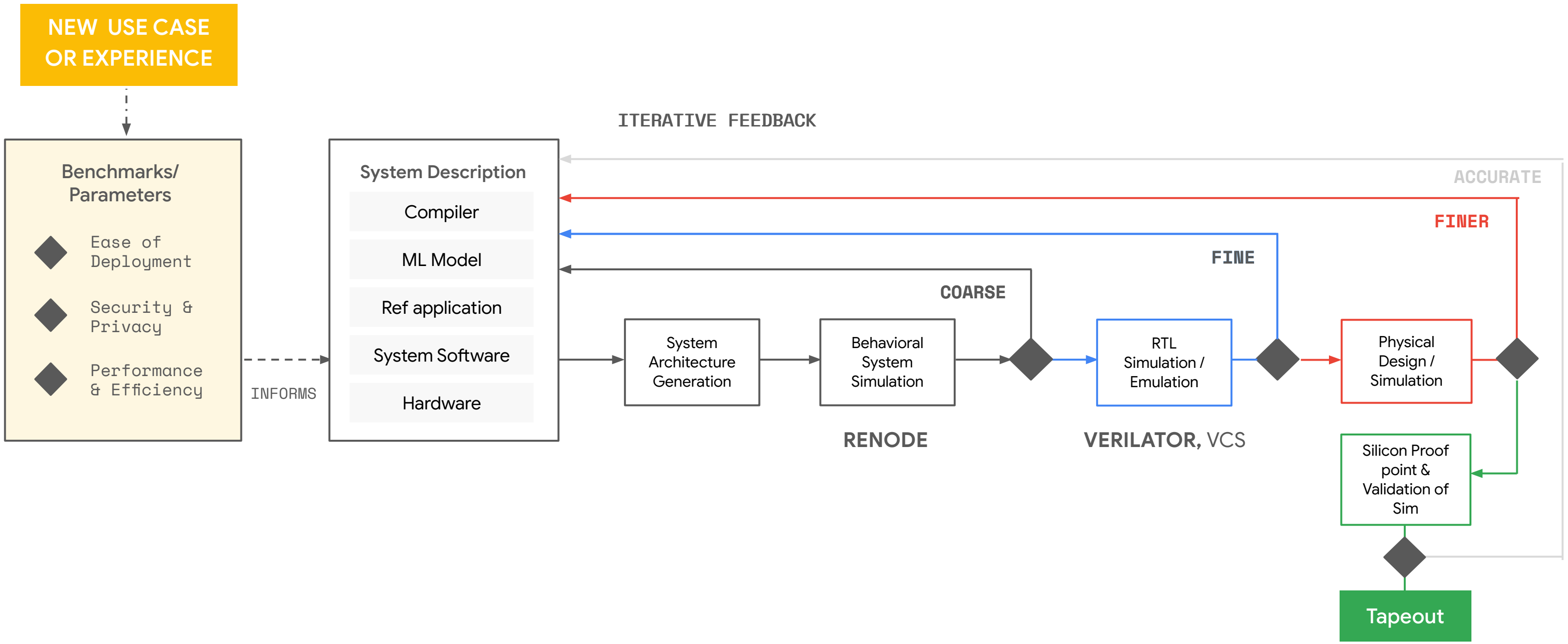
1. SMC
2. Root of Trust
3. Peripherals
4. Kelvin

1. CantripOS

***Library of lego pieces** (at disposal of users of the Sparrow workflow)*

Matcha Reference Implementation Timeline (H2 2023)

RTL Freeze & Tapeout



Google Research

Matcha Status & Demo

- Achieve system PPA through physical synthesis & simulation.
- RTL Freeze in Q3’23 & Tapeout in Q4’23.

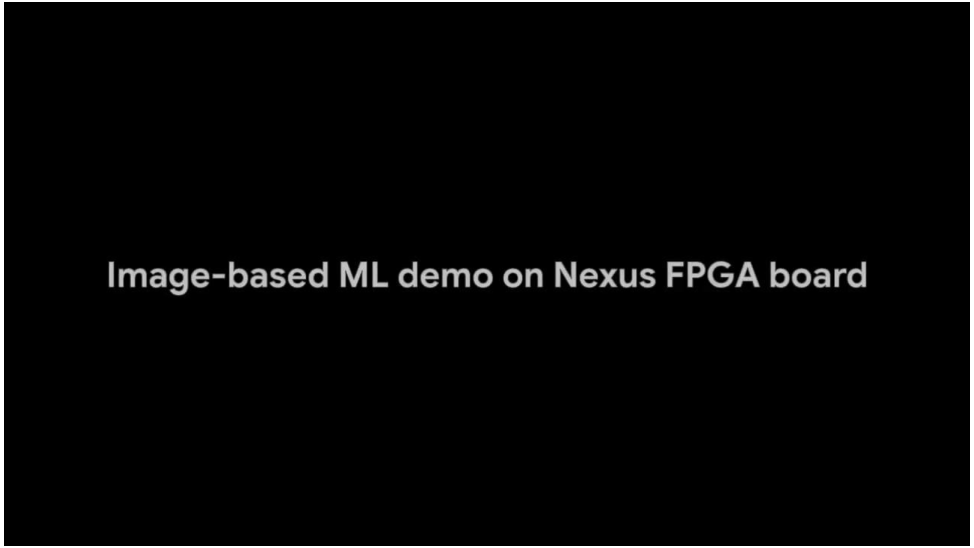


Image-based ML demo on Nexus FPGA board

03

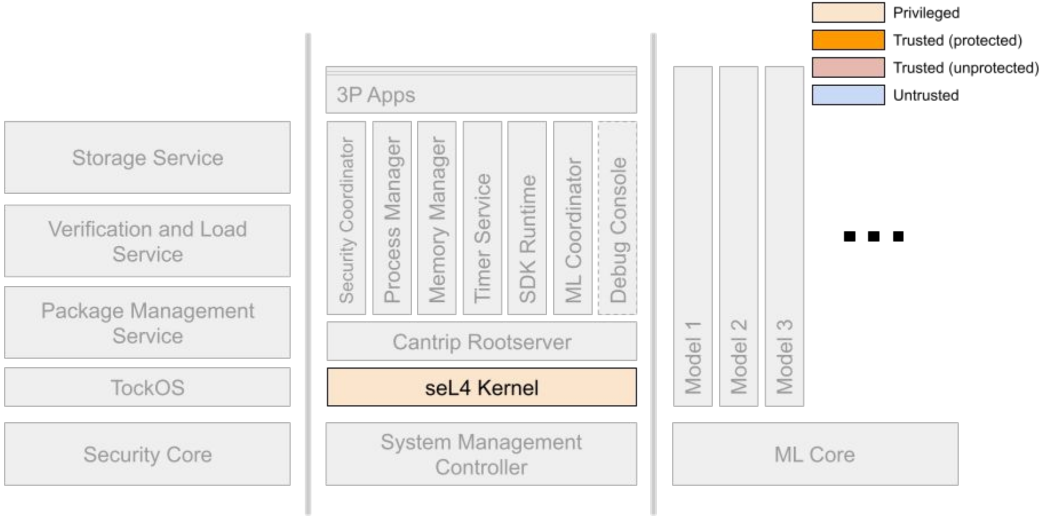
CantripOS

CantripOS

seL4 + Rust

- Hybrid static/dynamic system.
- seL4 kernel.
- 100% Rust runtime.
- Uses CAmkES' static componentization framework to specify & assemble trusted system services.
- Rust rootserver (aka capdl-loader) implementation.
- Provides dynamic memory management.
- Provides dynamic untrusted application support.
- Supports ML workloads built with IREE.
- Uses Security Core (based on OpenTitan) for secure boot and verification of application bundles.
- Provides a variety of developer tools.
- Open Source: available at github.com/AmbiML/sparrow-manifest.

CantripOS: seL4 kernel



seL4:

- Uses a reduced subset of C for proofs written in Isabelle/HOL.
- Microkernel design
- Real-time scheduler

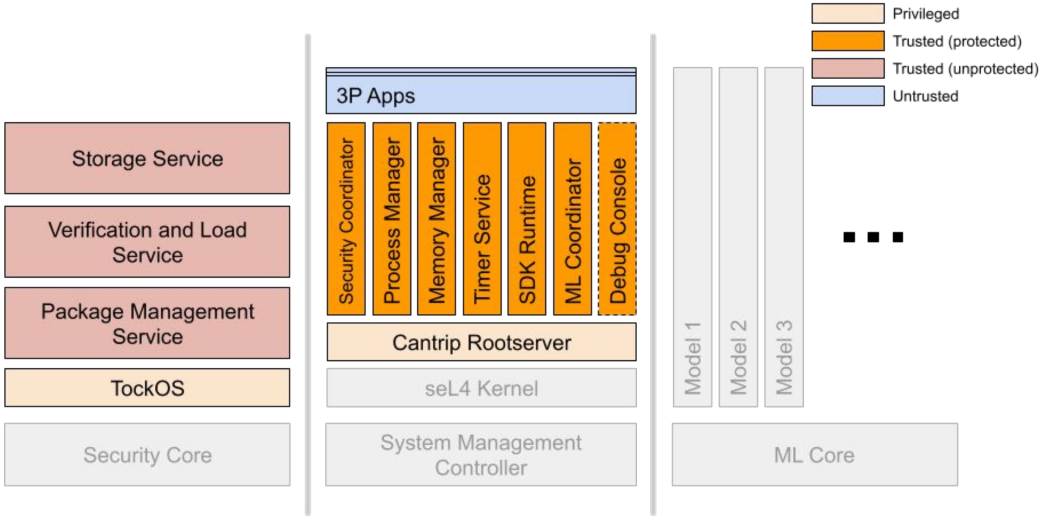
CantripOS changes:

- Allocate rootserver objects from `UntypedMemory`.
- Tweak `UntypedMemory` objects for MemoryManager.
- Development tools; e.g. capscan to dump `CNode` contents.
- Platform support: drivers, bootstrap, configuration, etc.

The Ugly:

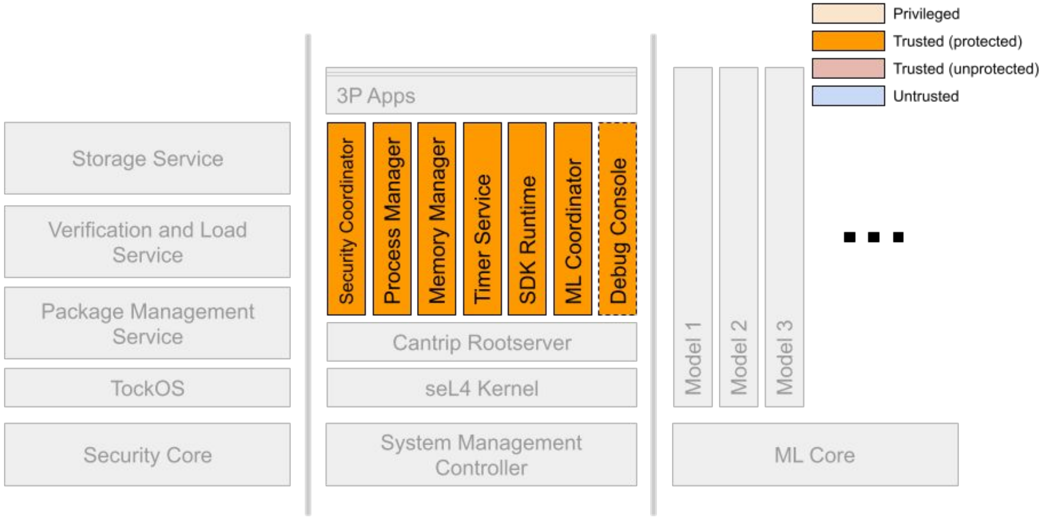
- Kernel forked mid-2021
- Not verified (yet)

CantripOS: 100% Rust



- Entire runtime except seL4 kernel is written in Rust.
- Expected Rust properties: memory safety, undefined behaviour guardrails, explicit unsafe code, potential formal verification.
- Compiler can optimize entire component; no FFI boundaries.
- `sel4-sys` crate replaces `libsel4`.
- `sel4-config` crate syncs kernel configuration to Rust.

CantripOS: CAmkES in Rust



- Uses `camkes-tool` & `capdl` + Rust templates & runtime.
- No interface specifications.
- No `sel4runtime`, `musllibc`, etc.
- Additions:
  - Capability passing
  - CSpace headroom
  - CopyRegion
  - Thread suppression
  - IRQ consolidation
  - Component startup
  - Semaphores
  - RPC implementations use serde & postcard for parameters

Guiding principle: minimize generated code.

Google Research

CantripOS: RPC support in Rust

- Multiple flavors: *basic*, *shared*, *sdkruntime*, *security core*.
- All RPC implementations use serde + postcard for parameter passing; main difference is the transport mechanism.
- Basic & shared RPC use CAmkES resources bound by the assembly: threads, notifications, badging, etc.
- Basic & shared RPC support attaching a capability to a request and/or reply.
- Sdkruntime RPC is used by sandboxed applications to talk to the SDKRuntime service.
- Security core (SEC) RPC is for communication with TockOS.

```
// Connect the DebugConsole to the OpenTitanUARTDriver
connection cantripRPCCall write_call(
    from debug_console.uart_write, to uart_driver.write);

// Connect the MemoryManager to each component that needs to allocate global memory.
connection cantripRPCOverMultiSharedData multi_memory(
    from debug_console.memory,
    from process_manager.memory,
    from security_coordinator.memory,
    from sdk_runtime.memory,
    from ml_coordinator.memory,
    to memory_manager.memory)
```

```
struct WriteInterfaceThread;
impl CamkesThreadInterface for WriteInterfaceThread {
    fn run() {
        rpc_basic_recv!(write, WRITE_REQUEST_DATA_SIZE, UartDriverError::Success);
    }
}
```

Overview
Shared
Basic
SDKRuntime
SEC

CantripOS: Shared RPC support

- Like CAmkES' seL4RPCOverMultiSharedData uses a shared 4KiB page for each connection.
- Has no CAmkES IDL; the protocol is serde-annotated Rust with wire-encoding done by postcard.
- Supports attaching a capability to a request and/or reply.

```
#[derive(Clone, Debug, Serialize, Deserialize)]
pub enum MemoryManagerRequest<'a> {
    Alloc {
        bundle: Cow<'a, ObjDescBundle>,
        lifetime: MemoryLifetime,
    },
    Free(Cow<'a, ObjDescBundle>),
    Stats, // -> MemoryResponseData
    Debug,
    Capscan,
}
```

```
type MemoryManagerResult = Result<Option<seL4_CPtr>, MemoryManagerError>;
impl CamkesThreadInterface for MemoryInterfaceThread {
    fn run() {
        rpc_shared_recv_with_caps!(
            memory,
            MEMORY_RECV_CNODE,
            MEMORY_REQUEST_DATA_SIZE,
            MemoryManagerError::Success
        );
    }
}
impl MemoryInterfaceThread {
    fn dispatch(
        _client_badge: usize,
        request_buffer: &[u8],
        reply_buffer: &mut [u8],
    ) -> MemoryManagerResult {
        let _cleanup = Camkes::cleanup_request_cap();
        let request = match postcard::from_bytes::<MemoryManagerRequest>(request_buffer) {
            Ok(request) => request,
            Err(_) => return Err(MemoryManagerError::DeserializeFailed),
        };
        match request {
            MemoryManagerRequest::Alloc {
                mut bundle,
                lifetime,
            } => Self::alloc_request(bundle.to_mut(), lifetime),
            MemoryManagerRequest::Free(mut bundle) => Self::free_request(bundle.to_mut()),
    …
}
```

CantripOS: Basic RPC support

- Like CAmkES' seL4RPCCall uses the TLS-allocated seL4_IPCBuffer.
- Has no CAmkES IDL; the protocol is serde-annotated Rust with wire-encoding done by postcard.
- Supports attaching a capability to a request and/or reply.
- Has potential ***Undefined Behaviour*** due to re-use of seL4_IPCBuffer: may require copying received data.

```
impl WriteInterfaceThread {
    fn dispatch(
        _client_badge: usize,
        request_buffer: &[u8],
        reply_buffer: &mut [u8],
    ) -> Result<usize, UartDriverError> {
        let request = match postcard::from_bytes::<WriteRequest>(request_buffer) {
            Ok(request) => request,
            Err(_) => return Err(UartDriverError::DeserializeFailed),
        };
        match request {
            WriteRequest::Write(available) => Self::write_request(available,
reply_buffer),
            WriteRequest::Flush => Self::flush_request(),
        }
    }
    …
}
```

```
fn uart_write_request<T: DeserializeOwned>(
    request: &WriteRequest
) -> Result<T, UartDriverError> {
    let (request_buffer, reply_slice) = rpc_basic_buffer!().split_at_mut(WRITE_REQUEST_DATA_SIZE);
    let request_slice =
        postcard::to_slice(request, request_buffer).or(Err(UartDriverError::SerializeFailed))?;
    match rpc_basic_send!(uart_write, request_slice.len()).0.into() {
        UartDriverError::Success => {
            let reply =
                postcard::from_bytes(reply_slice).or(Err(UartDriverError::DeserializeFailed))?;
            Ok(reply)
        }
        err => Err(err)
    }
}
```

Google Research

CantripOS: SDKRuntime RPC support

- Used by sandboxed applications to talk to the SDKRuntime service.
- Has no CAmkES IDL; the protocol is serde-annotated Rust with wire-encoding done by postcard.
- No CAmkES; uses a per-application 4KiB page that is “loaned” to the SDKRuntime.
- Could “pre-map” RPC pages but want to minimize app resource visibility and flexibility (may need multiple pages for moving lots of data).
- Proof-of-concept; e.g. haven’t measured cost to do map & unmap per-RPC.

```
/// SDKRequest token sent over the seL4 IPC interface. We need repr(seL4_Word)
/// but cannot use that so use the implied usize type instead.
///
/// Note that this enum starts off at 64. This is to avoid collisions with the
/// seL4_Fault enumeration used by the kernel, as the SDK runtime is also used
/// as the application's fault handler.
#[repr(usize)]
#[derive(Debug, Clone, Copy, Eq, PartialEq, IntoPrimitive, TryFromPrimitive)]
pub enum SDKRuntimeRequest {
    Ping = 64, // Check runtime is alive
    Log,       // Log message: [msg: &str]

    ReadKey,   // Read key: [key: &str, &mut [u8]] -> value: &[u8]
    WriteKey,  // Write key: [key: &str, value: &KeyValueData]
    DeleteKey, // Delete key: [key: &str]

    OneshotTimer,  // One-shot timer: [id: TimerId, duration_ms: TimerDuration]
    PeriodicTimer, // Periodic timer: [id: TimerId, duration_ms: TimerDuration]
    CancelTimer,   // Cancel timer: [id: TimerId]
    WaitForTimers, // Wait for timers to expire: [] -> TimerMask
    PollForTimers, // Poll for timers to expire: [] -> TimerMask

    OneshotModel,   // One-shot model execution: [model_id: &str] -> id: ModelId
    PeriodicModel, // Periodic model execution: [model_id: &str, duration_ms:
TimerDuration] -> ModelId
    CancelModel,   // Cancel running model: [id: ModelId]
    WaitForModel,  // Wait for any running model to complete: [] -> ModelMask
    PollForModels, // Poll for running models to complete: [] -> ModelMask
    GetModelOutput, // Return output data from most recent run: [id: ModelId, clear: bool]
-> ModelOutput
}
```

CantripOS: Security Core RPC support

- For communication with the TockOS StorageManager (so far).
- Buried in the mailbox-driver with two flavors: IRQ-driven (SecurityCoordinator) and synchronous (cantrip-os-rootserver).
- The protocol is serde-annotated Rust with wire-encoding done by postcard
- RPC messages are passed through the mailbox hardware FIFO.
- Bulk data is moved with memory pages loaned by the SMC to the SEC.

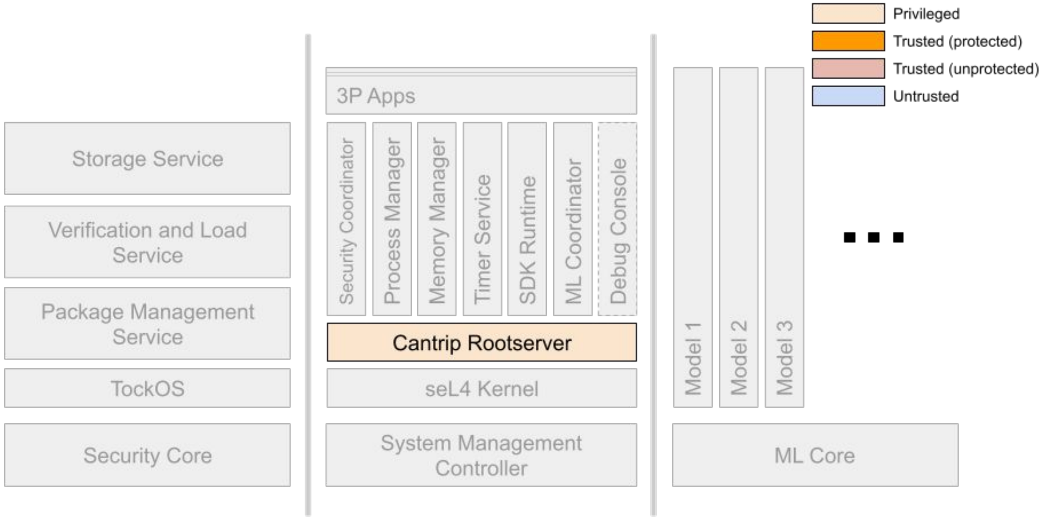
```
#[derive(Debug, Serialize, Deserialize)]
pub enum SECRequest<'a> {
    FindFile(&'a str),     // Find file by name -> (/*fid*/ u32, /*size*/ u32)
    GetFilePage(u32, u32), // Get page of file data -> <attached page>

    #[cfg(feature = "alloc")]
    GetBuiltins, // Get package names -> Vec(String)
}
...
#[cfg(feature = "alloc")]
pub fn mbox_get_builtins() -> Result<cantrip_security_interface::BundleIdArray,
SECRequestError> {
    sec_request(&SECRequest::GetBuiltins, None).map(|reply: GetBuiltinsResponse|
reply.names)
}

pub fn mbox_find_file(name: &str) -> Result<(u32, u32), SECRequestError> {
    sec_request(&SECRequest::FindFile(name), None)
        .map(|reply: FindFileResponse| (reply.fid, reply.size_bytes))
}

pub fn mbox_get_file_page(fid: u32, offset: u32, frame: seL4_CPtr) -> Result<(),
SECRequestError> {
    sec_request(&SECRequest::GetFilePage(fid, offset), Some(frame))?;
    Ok(())
}
```

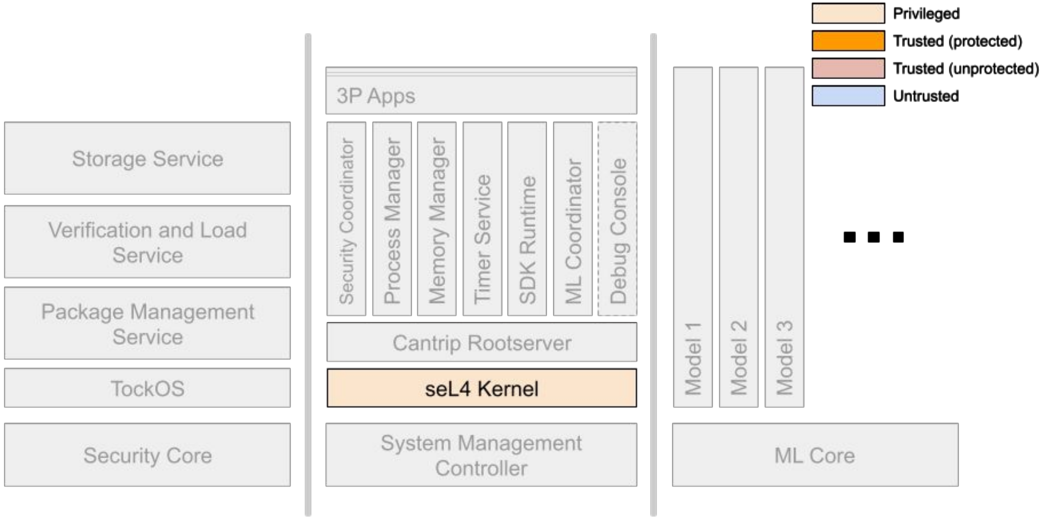
CantripOS: cantrip-os-rootserver



- Rust implementation of rootserver.
- Intentionally mimics flow of `capdl-loader-app` (was first foray into CAmkES internals).
- Additions:
  - Participates in rootserver memory reclamation
  - Dup-on-error for shared pages
  - MemoryManager handoff
  - ProcessManager handoff
  - Fill file data from flash

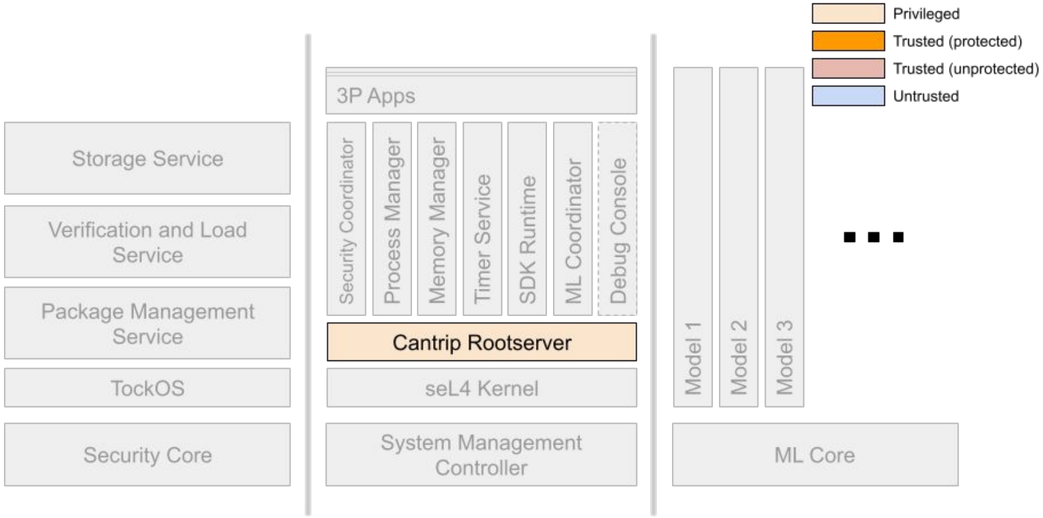
Meant to reuse code for ProcessManager but did not happen.

CantripOS: cantrip-os-rootserver memory reclamation



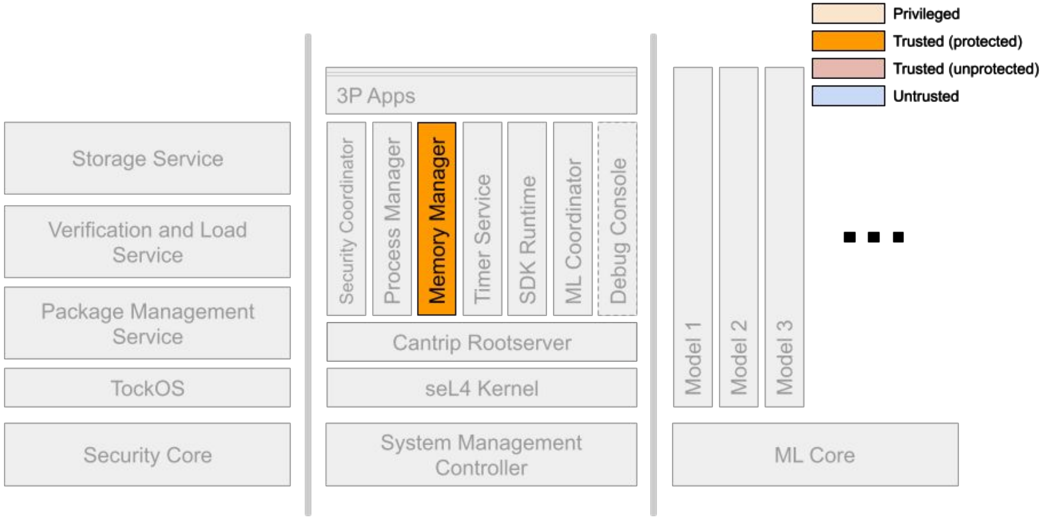
1. **Kernel** forms `UntypedMemory` objects for allocating initial thread state (e.g. `TCB`, `CNode`) instead of anonymous memory.
2. **Kernel** allocates objects & marks the associated `UntypedMemory` as “*tainted*”.

CantripOS: cantrip-os-rootserver memory reclamation



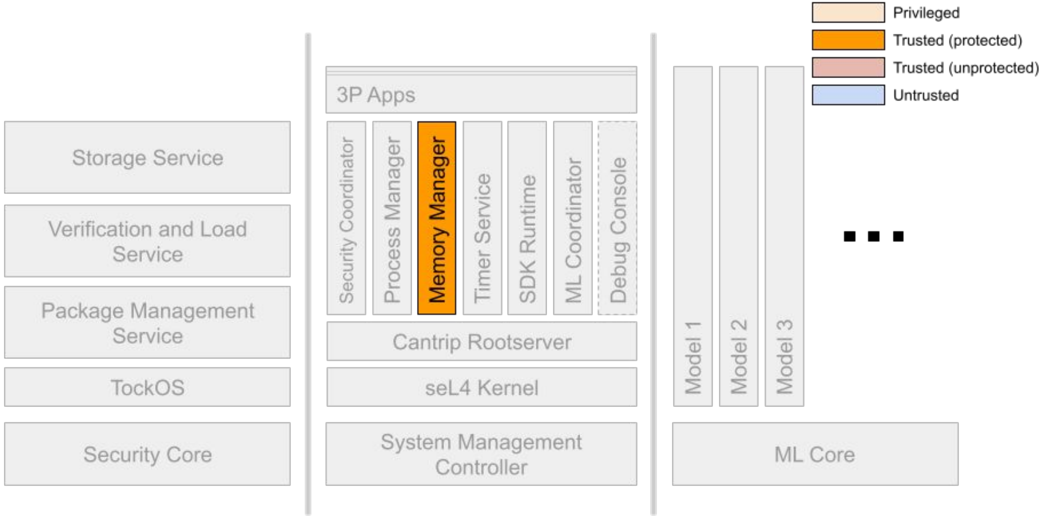
1. **Kernel** forms `UntypedMemory` objects for allocating initial thread state (e.g. `TCB`, `CNode`) instead of anonymous memory.
2. **Kernel** allocates objects & marks the associated `UntypedMemory` as “*tainted*”.
3. **Rootserver** runs and sets up CAmkES assembly <u>*using untainted memory*</u> (to not mix with kernel-allocated objects).
4. **Rootserver** collects CAmkES objects in a “Holding `CNode`”.
5. **Rootserver** passes Holding `CNode`, `UntypedMemory` and `BootInfo` state to MemoryManager.
6. **Rootserver** suspends itself.

CantripOS: cantrip-os-rootserver memory reclamation



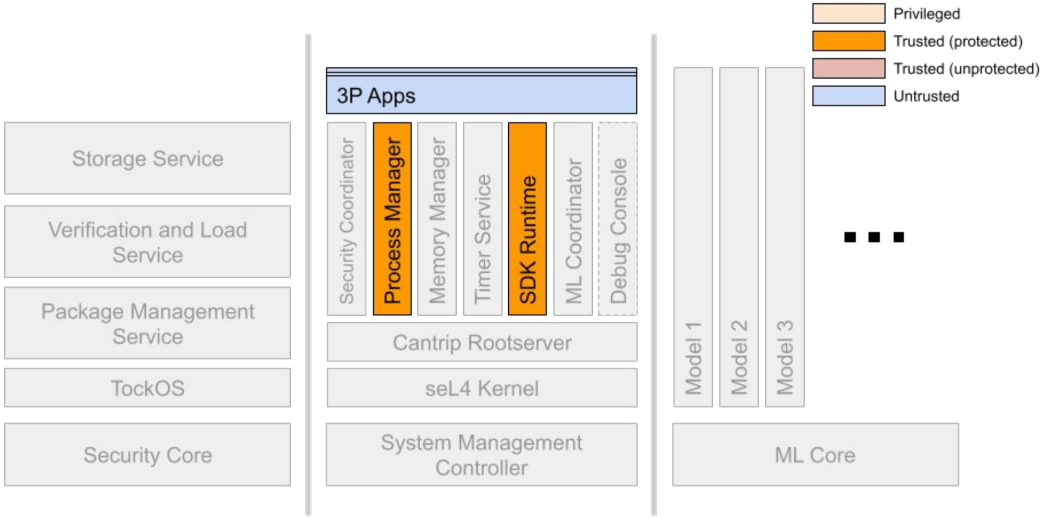
1. **Kernel** forms `UntypedMemory` objects for allocating initial thread state (e.g. `TCB`, `CNode`) instead of anonymous memory.
2. **Kernel** allocates objects & marks the associated `UntypedMemory` as "*tainted*".
3. **Rootserver** runs and sets up CAmkES assembly *using untainted memory* (to not mix with kernel-allocated objects).
4. **Rootserver** collects CAmkES objects in a "Holding `CNode`".
5. **Rootserver** passes Holding `CNode`, `UntypedMemory` and `BootInfo` state to MemoryManager.
6. **Rootserver** suspends itself.
7. **MemoryManager** runs and revokes *tainted* `UntypedMemory` objects to reclaim rootserver memory.

CantripOS: Dynamic Memory Management



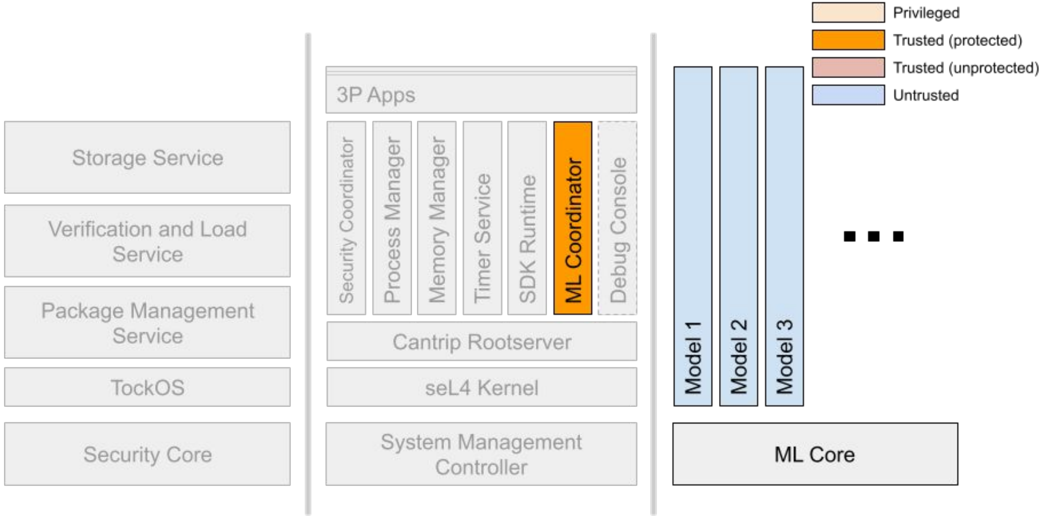
- System-wide dynamic memory management service (MemoryManager): clients `allocate` & `free` seL4 objects.
- Supports batching to minimize RPC calls and memory fragmentation.
- Starts with unallocated `UntypedMemory` objects sent by rootserver after it finishes its work.
- Holds refs to objects created by rootserver for CAmkES components.
- Depends on `free` calls to reclaim resources (ProcessManager cleans up applications).

CantripOS: Dynamic Applications



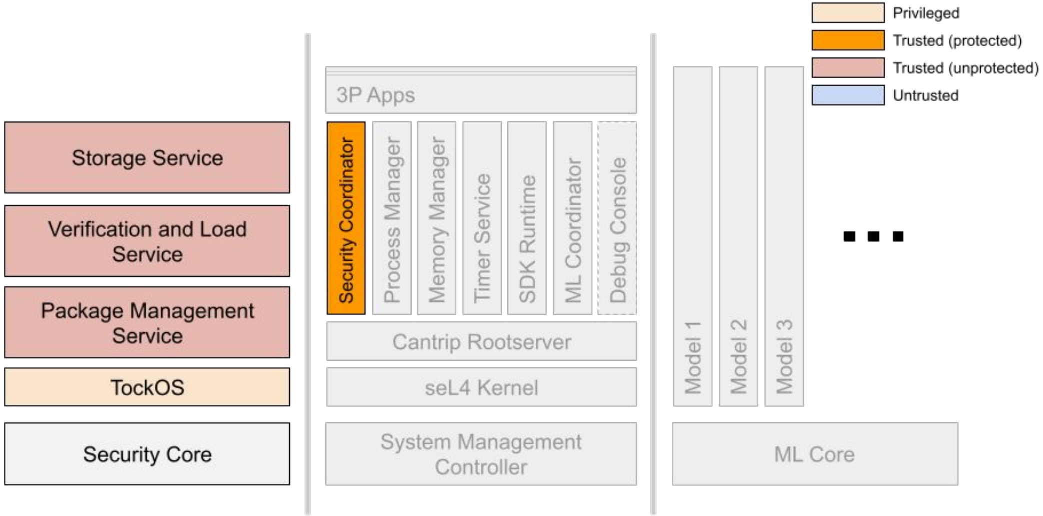
- Untrusted applications run with strong sandboxing.
- Language-agnostic; mostly Rust samples to start.
- Single seL4 thread: intent is to leverage language-specific multi-threading (e.g. Rust async)
- Application-specific runtime intended (current implementation is a POC).
- ProcessManager creates & manages applications.
- Applications loaded from FLASH (builtin) or optionally side-loaded.
- Still a WIP–e.g signed executables, support for multiple runtimes.

CantripOS: Machine Learning (ML) Coordinator



- Manages ML accelerator hardware:
  - manage resources
  - load model
  - handle interrupts
  - return results
- Schedules ML workloads.
- Single & periodic runs.
- Simplistic multi-tenant support for initial Springbok device.

CantripOS: Security Coordinator



- Manages communication with the Security Core (SEC).
- SEC:
  - OpenTitan-based (TockOS)
  - Root of Trust
  - FLASH storage
  - SEC↔SMC Mailbox hardware
  - Crypto functionality*
  - Secure* seL4 bootstrap

* TBD

CantripOS: Developer Tools

sel4gdb	seL4-thread-aware debugging support
logging	Rust log crate integration
kmem	kmem (+ bloaty): tools for inspecting memory footprint
capscan	facility for inspecting CNode contents
sel4-sys	C-callable wrappers for integrating Rust syscall stubs with sel4test

CantripOS: sel4gdb

- seL4-thread-aware debugging support for gdb + Renode
- Understands CAmkES components & threads
- Uses Renode to intercept thread & kernel context switches (can auto-switch symbol tables)

```
sel4 break [<thread>:<symbol> | user | kernel]
sel4 tbreak [...]
sel4 thread
sel4 threads
sel4 wait-for-thread [<thread>]
```

```
(gdb) # Wait until rootserver thread is known to seL4 extensions
(gdb) sel4 wait-for-thread rootserver
(gdb) # Switch symbols to rootserver's
(gdb) sel4 switch-symbols rootserver
(gdb) # Create temporary breakpoint on main function in rootserver thread
(gdb) sel4 tbreak rootserver main
(gdb) continue
(gdb) # We can confirm, that we are indeed in rootserver thread
(gdb) sel4 thread
rootserver
```

```
(gdb) # Create temporary breakpoint on context-switch to kernel
(gdb) sel4 tbreak kernel
(gdb) continue
(gdb) sel4 thread
kernel
```

CantripOS: Logging

- Uses familiar Rust log crate.
- Logging requests passed to DebugConsole component using IPCs.
- Development logging (`debug!`, `trace!`) typically compiled out in release builds.
- Log level settable through console.

The Ugly: clients can block,
requires a console

```
CANTRIP> loglevel trace
TRACE
CANTRIP> start timer
cantrip_proc_manager::proc_manager::start bundle_id timer
cantrip_proc_manager::proc_manager::builtin auto-install
cantrip_proc_manager::ProcessManagerInterface::start Bundle { app_id: "timer",
app_memory_size: 0 }
cantrip_security_interface::cantrip_security_request LoadApplication("timer") cap
None
cantrip_security_coordinator::LOAD APPLICATION bundle_id timer
cantrip_memory_interface::cantrip_object_alloc { MEMORY_RECV, [ObjDesc { type_:
seL4_RISCV_4K_Page, count: 1, cptr: 0 }] }
cantrip_memory_interface::cantrip_memory_request Alloc { bundle: ObjDescBundle {
cnode: 3, depth: 8, objs: [ObjDesc { type_: seL4_RISCV_4K_Page, count: 1, cptr: 0 }]
}, lifetime: Medium } cap Some(3)
…
cantrip_proc_manager::sel4bundle::map ipcbuffer slot 28 vaddr 0x74000 seL4_CapRights
{ words: [3] }
cantrip_proc_manager::sel4bundle::arch::map page ObjDesc { type_:
seL4_RISCV_4K_Page, count: 1, cptr: 28 } root ObjDesc { type_:
seL4_RISCV_PageTableObject, count: 1, cptr: 128 } at 0x74000
cantrip_proc_manager::sel4bundle::map sdk_runtime slot 29 vaddr 0x75000
seL4_CapRights { words: [3] }
cantrip_proc_manager::sel4bundle::arch::map page ObjDesc { type_:
seL4_RISCV_4K_Page, count: 1, cptr: 29 } root ObjDesc { type_:
seL4_RISCV_PageTableObject, count: 1, cptr: 128 } at 0x75000
Cantrip_proc_manager::sel4bundle::init_tcb
Cantrip_proc_manager::sel4bundle::init_cspace
Bundle "timer" started.
```

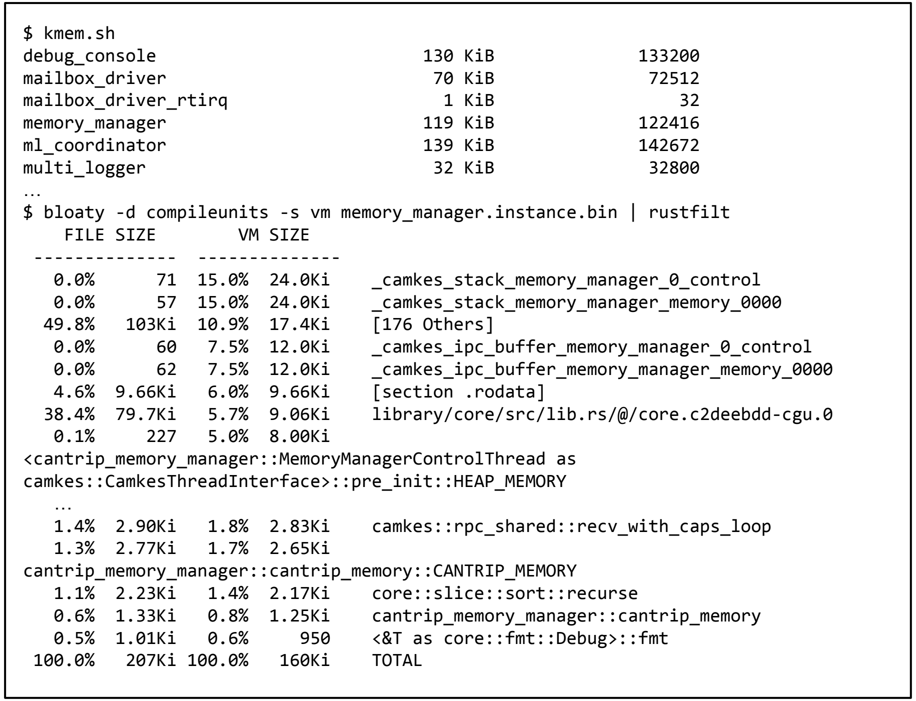
sel4gdb logging kmem capscan sel4-sys

Google Research

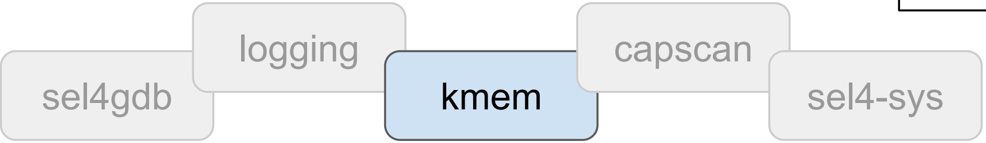
CantripOS: Memory Footprint Tools

- Memory footprint = $$$ + power.
- Target platform has 4MiB of memory but we want to work in 2MiB (or less).
- kmem.sh analyzes capdl-generated AST to identify system usage.
- bloaty analyzes a single ELF to identify component usage.

Still need: per-thread stack analysis,
`system.cdl` rootserver sizing



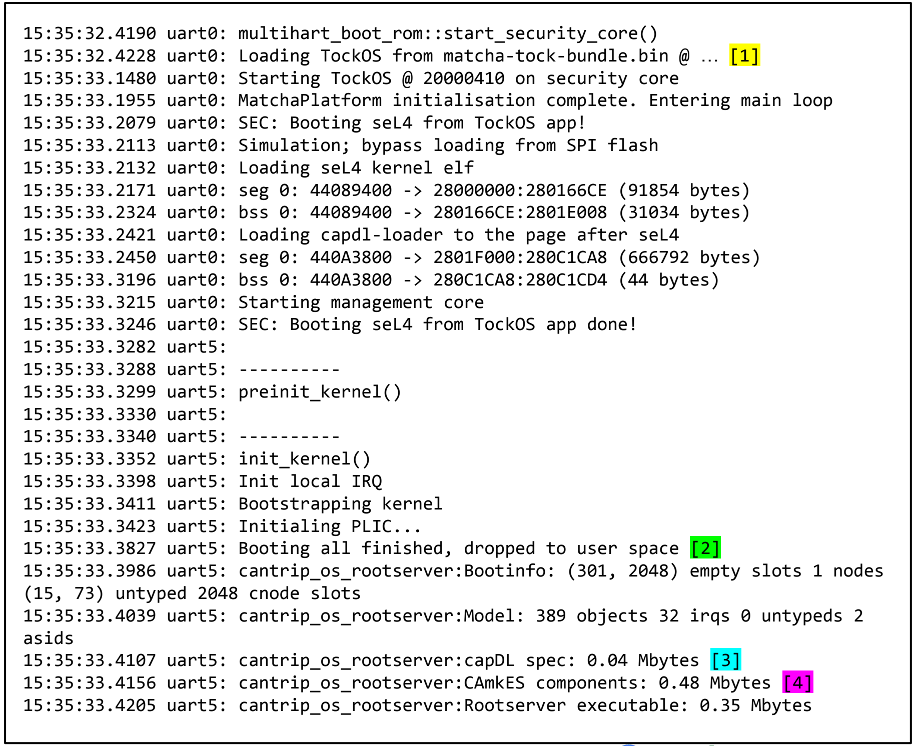
```
$ kmem.sh
debug_console                          130 KiB              133200
mailbox_driver                          70 KiB               72512
mailbox_driver_rtirq                     1 KiB                  32
memory_manager                         119 KiB              122416
ml_coordinator                         139 KiB              142672
multi_logger                            32 KiB               32800
…
$ bloaty -d compileunits -s vm memory_manager.instance.bin | rustfilt
    FILE SIZE        VM SIZE
 --------------  --------------
   0.0%      71  15.0%  24.0Ki    _camkes_stack_memory_manager_0_control
   0.0%      57  15.0%  24.0Ki    _camkes_stack_memory_manager_memory_0000
  49.8%   103Ki  10.9%  17.4Ki    [176 Others]
   0.0%      60   7.5%  12.0Ki    _camkes_ipc_buffer_memory_manager_0_control
   0.0%      62   7.5%  12.0Ki    _camkes_ipc_buffer_memory_manager_memory_0000
   4.6%  9.66Ki   6.0%  9.66Ki    [section .rodata]
  38.4%  79.7Ki   5.7%  9.06Ki    library/core/src/lib.rs/@/core.c2deebdd-cgu.0
   0.1%     227   5.0%  8.00Ki
<cantrip_memory_manager::MemoryManagerControlThread as
camkes::CamkesThreadInterface>::pre_init::HEAP_MEMORY
   ...
   1.4%  2.90Ki   1.8%  2.83Ki    camkes::rpc_shared::recv_with_caps_loop
   1.3%  2.77Ki   1.7%  2.65Ki
cantrip_memory_manager::cantrip_memory::CANTRIP_MEMORY
   1.1%  2.23Ki   1.4%  2.17Ki    core::slice::sort::recurse
   0.6%  1.33Ki   0.8%  1.25Ki    cantrip_memory_manager::cantrip_memory
   0.5%  1.01Ki   0.6%     950    <&T as core::fmt::Debug>::fmt
 100.0%   207Ki 100.0%   160Ki    TOTAL
```



sel4gdb
logging
kmem
capscan
sel4-sys

CantripOS: Memory Footprint Intro

- We define the memory footprint as the peak memory used to reach the DebugConsole `CANTRIP>` prompt.
- Calculating the memory footprint is complicated by the use of CAmkES/capDL (and implicitly the rootserver).
- First we must understand the boot process:
  - boot TockOS on the SEC [1]
  - TockOS loads the seL4 boot image (kernel + capdl-loader-app) and starts it
  - seL4 kernel sets up the rootserver state and starts it [2]
  - rootserver instantiates the capDL specification [3] and starts all TCBs marked *runnable*; this includes loading executable images from a cpio archive embedded in the boot image [4]
  - rootserver memory is reclaimed

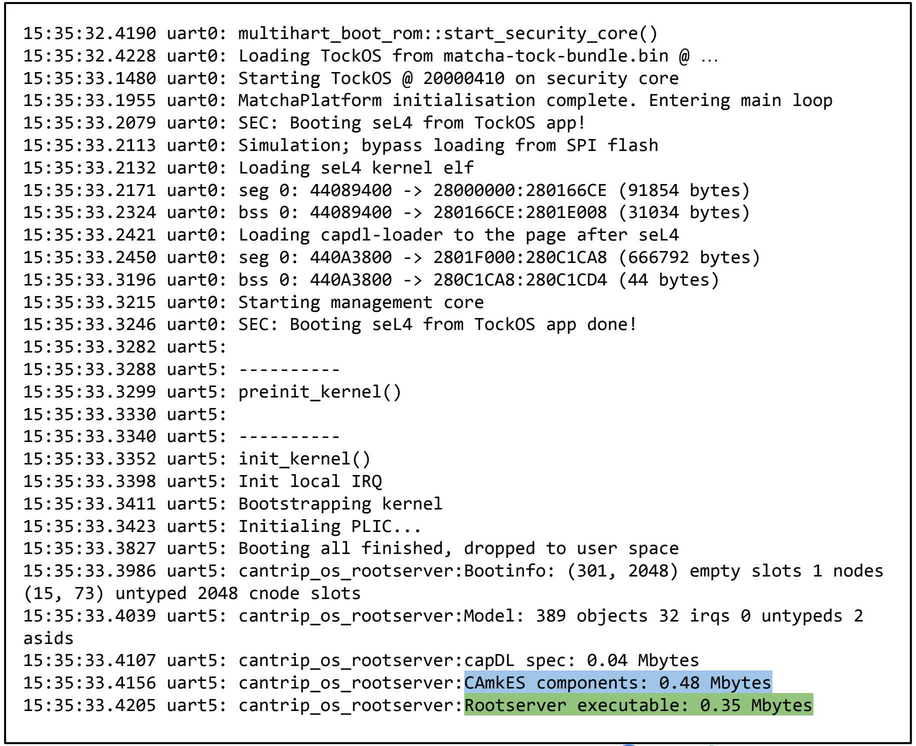


```
15:35:32.4190 uart0: multihart_boot_rom::start_security_core()
15:35:32.4228 uart0: Loading TockOS from matcha-tock-bundle.bin @ … [1]
15:35:33.1480 uart0: Starting TockOS @ 20000410 on security core
15:35:33.1955 uart0: MatchaPlatform initialisation complete. Entering main loop
15:35:33.2079 uart0: SEC: Booting seL4 from TockOS app!
15:35:33.2113 uart0: Simulation; bypass loading from SPI flash
15:35:33.2132 uart0: Loading seL4 kernel elf
15:35:33.2171 uart0: seg 0: 44089400 -> 28000000:280166CE (91854 bytes)
15:35:33.2324 uart0: bss 0: 44089400 -> 280166CE:2801E008 (31034 bytes)
15:35:33.2421 uart0: Loading capdl-loader to the page after seL4
15:35:33.2450 uart0: seg 0: 440A3800 -> 2801F000:280C1CA8 (666792 bytes)
15:35:33.3196 uart0: bss 0: 440A3800 -> 280C1CA8:280C1CD4 (44 bytes)
15:35:33.3215 uart0: Starting management core
15:35:33.3246 uart0: SEC: Booting seL4 from TockOS app done!
15:35:33.3282 uart5:
15:35:33.3288 uart5: ----------
15:35:33.3299 uart5: preinit_kernel()
15:35:33.3330 uart5:
15:35:33.3340 uart5: ----------
15:35:33.3352 uart5: init_kernel()
15:35:33.3398 uart5: Init local IRQ
15:35:33.3411 uart5: Bootstrapping kernel
15:35:33.3423 uart5: Initialing PLIC...
15:35:33.3827 uart5: Booting all finished, dropped to user space [2]
15:35:33.3986 uart5: cantrip_os_rootserver:Bootinfo: (301, 2048) empty slots 1 nodes
(15, 73) untyped 2048 cnode slots
15:35:33.4039 uart5: cantrip_os_rootserver:Model: 389 objects 32 irqs 0 untypeds 2
asids
15:35:33.4107 uart5: cantrip_os_rootserver:capDL spec: 0.04 Mbytes [3]
15:35:33.4156 uart5: cantrip_os_rootserver:CAmkES components: 0.48 Mbytes [4]
15:35:33.4205 uart5: cantrip_os_rootserver:Rootserver executable: 0.35 Mbytes
```

CantripOS: Memory Footprint Estimation

- When the rootserver runs both the CAmkES components embedded in capdl-loader and the instantiated components are in memory at the same time. On a memory-tuned system this requires ~1.5-1.7x the memory used once we reach the prompt: sizeof(CAmkES components) + sizeof(rootserver-user-mode) + sizeof(rootserver-kernel-mode) + sizeof(capdl_archive) = sizeof(CAmkES components) + 0.35M + sizeof(rootserver-kernel-mode) + 0.48M.

NOTE: *the numbers shown here are after major reductions*



```
15:35:32.4190 uart0: multihart_boot_rom::start_security_core()
15:35:32.4228 uart0: Loading TockOS from matcha-tock-bundle.bin @ …
15:35:33.1480 uart0: Starting TockOS @ 20000410 on security core
15:35:33.1955 uart0: MatchaPlatform initialisation complete. Entering main loop
15:35:33.2079 uart0: SEC: Booting seL4 from TockOS app!
15:35:33.2113 uart0: Simulation; bypass loading from SPI flash
15:35:33.2132 uart0: Loading seL4 kernel elf
15:35:33.2171 uart0: seg 0: 44089400 -> 28000000:280166CE (91854 bytes)
15:35:33.2324 uart0: bss 0: 44089400 -> 280166CE:2801E008 (31034 bytes)
15:35:33.2421 uart0: Loading capdl-loader to the page after seL4
15:35:33.2450 uart0: seg 0: 440A3800 -> 2801F000:280C1CA8 (666792 bytes)
15:35:33.3196 uart0: bss 0: 440A3800 -> 280C1CA8:280C1CD4 (44 bytes)
15:35:33.3215 uart0: Starting management core
15:35:33.3246 uart0: SEC: Booting seL4 from TockOS app done!
15:35:33.3282 uart5:
15:35:33.3288 uart5: ----------
15:35:33.3299 uart5: preinit_kernel()
15:35:33.3330 uart5:
15:35:33.3340 uart5: ----------
15:35:33.3352 uart5: init_kernel()
15:35:33.3398 uart5: Init local IRQ
15:35:33.3411 uart5: Bootstrapping kernel
15:35:33.3423 uart5: Initialing PLIC...
15:35:33.3827 uart5: Booting all finished, dropped to user space
15:35:33.3986 uart5: cantrip_os_rootserver:Bootinfo: (301, 2048) empty slots 1 nodes
(15, 73) untyped 2048 cnode slots
15:35:33.4039 uart5: cantrip_os_rootserver:Model: 389 objects 32 irqs 0 untypeds 2
asids
15:35:33.4107 uart5: cantrip_os_rootserver:capDL spec: 0.04 Mbytes
15:35:33.4156 uart5: cantrip_os_rootserver:CAmkES components: 0.48 Mbytes
15:35:33.4205 uart5: cantrip_os_rootserver:Rootserver executable: 0.35 Mbytes
```

CantripOS: Memory Footprint Reduction

How to reduce memory footprint (w/ simple changes):

- Remove unused components; e.g. DebugConsole and UART Driver (no serial console)
- Remove unused RPC connections; e.g. shared memory connections cost 4KiB
- Remove unnecessary threads; e.g. re-use the control thread, consolidate IRQ handling
- Tune thread stack sizes
- Compile out debug/devel logging (`release_max_level_off`)
- Disable unneeded kernel options; e.g. CONFIG_PRINTING
- Tune rootserver resource configuration

More advanced/involved:

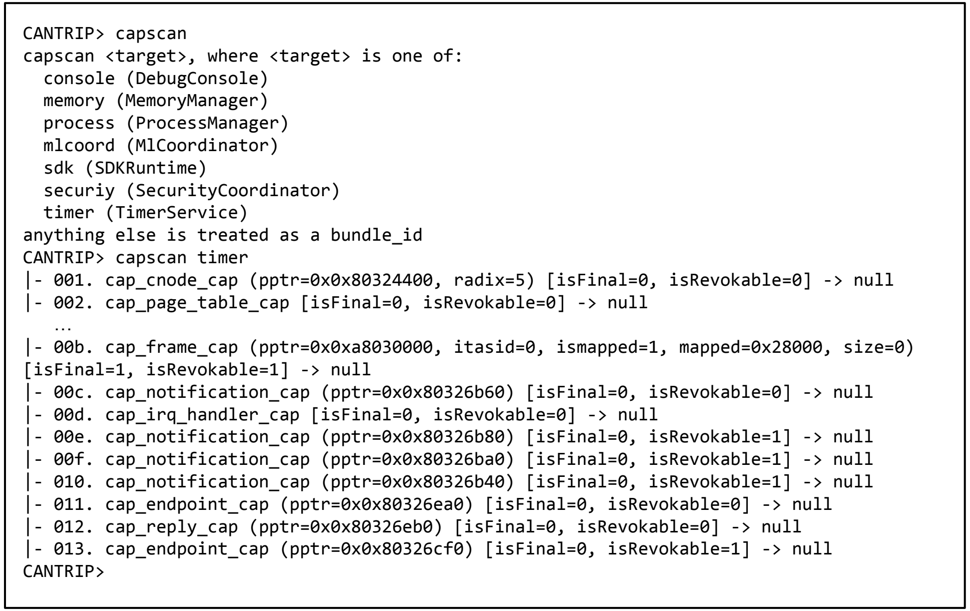
- Move data to flash; e.g. builtins & capdl_archive

```
15:35:32.4190 uart0: multihart_boot_rom::start_security_core()
15:35:32.4228 uart0: Loading TockOS from matcha-tock-bundle.bin @ …
15:35:33.1480 uart0: Starting TockOS @ 20000410 on security core
15:35:33.1955 uart0: MatchaPlatform initialisation complete. Entering main loop
15:35:33.2079 uart0: SEC: Booting seL4 from TockOS app!
15:35:33.2113 uart0: Simulation; bypass loading from SPI flash
15:35:33.2132 uart0: Loading seL4 kernel elf
15:35:33.2171 uart0: seg 0: 44089400 -> 28000000:280166CE (91854 bytes)
15:35:33.2324 uart0: bss 0: 44089400 -> 280166CE:2801E008 (31034 bytes)
15:35:33.2421 uart0: Loading capdl-loader to the page after seL4
15:35:33.2450 uart0: seg 0: 440A3800 -> 2801F000:280C1CA8 (666792 bytes)
15:35:33.3196 uart0: bss 0: 440A3800 -> 280C1CA8:280C1CD4 (44 bytes)
15:35:33.3215 uart0: Starting management core
15:35:33.3246 uart0: SEC: Booting seL4 from TockOS app done!
15:35:33.3282 uart5:
15:35:33.3288 uart5: ----------
15:35:33.3299 uart5: preinit_kernel()
15:35:33.3330 uart5:
15:35:33.3340 uart5: ----------
15:35:33.3352 uart5: init_kernel()
15:35:33.3398 uart5: Init local IRQ
15:35:33.3411 uart5: Bootstrapping kernel
15:35:33.3423 uart5: Initialing PLIC...
15:35:33.3827 uart5: Booting all finished, dropped to user space
15:35:33.3986 uart5: cantrip_os_rootserver:Bootinfo: (301, 2048) empty slots 1 nodes
(15, 73) untyped 2048 cnode slots
15:35:33.4039 uart5: cantrip_os_rootserver:Model: 389 objects 32 irqs 0 untypeds 2
asids
15:35:33.4107 uart5: cantrip_os_rootserver:capDL spec: 0.04 Mbytes
15:35:33.4156 uart5: cantrip_os_rootserver:CAmkES components: 0.48 Mbytes
15:35:33.4205 uart5: cantrip_os_rootserver:Rootserver executable: 0.35 Mbytes
```

CantripOS: Capscan

- Tool for inspecting `CNode` contents.
- Especially useful for debugging capability passing (e.g. leaks).

```
CANTRIP> capscan <component>
CANTRIP> capscan <application>
```



```
CANTRIP> capscan
capscan <target>, where <target> is one of:
  console (DebugConsole)
  memory (MemoryManager)
  process (ProcessManager)
  mlcoord (MlCoordinator)
  sdk (SDKRuntime)
  securiy (SecurityCoordinator)
  timer (TimerService)
anything else is treated as a bundle_id
CANTRIP> capscan timer
|- 001. cap_cnode_cap (pptr=0x0x80324400, radix=5) [isFinal=0, isRevokable=0] -> null
|- 002. cap_page_table_cap [isFinal=0, isRevokable=0] -> null
   …
|- 00b. cap_frame_cap (pptr=0x0xa8030000, itasid=0, ismapped=1, mapped=0x28000, size=0)
[isFinal=1, isRevokable=1] -> null
|- 00c. cap_notification_cap (pptr=0x0x80326b60) [isFinal=0, isRevokable=0] -> null
|- 00d. cap_irq_handler_cap [isFinal=0, isRevokable=0] -> null
|- 00e. cap_notification_cap (pptr=0x0x80326b80) [isFinal=0, isRevokable=1] -> null
|- 00f. cap_notification_cap (pptr=0x0x80326ba0) [isFinal=0, isRevokable=1] -> null
|- 010. cap_notification_cap (pptr=0x0x80326b40) [isFinal=0, isRevokable=1] -> null
|- 011. cap_endpoint_cap (pptr=0x0x80326ea0) [isFinal=0, isRevokable=0] -> null
|- 012. cap_reply_cap (pptr=0x0x80326eb0) [isFinal=0, isRevokable=0] -> null
|- 013. cap_endpoint_cap (pptr=0x0x80326cf0) [isFinal=0, isRevokable=1] -> null
CANTRIP>
```

sel4gdb logging kmem **capscan** sel4-sys

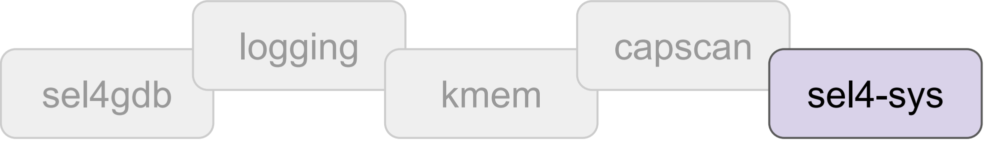
Google Research

CantripOS: Syscall Wrappers

- C-callable wrappers for Rust syscall stubs for `sel4test` integration.
- Validate stubs by comparing stock `sel4test` to `sel4test+wrapper`.

```
fn derive_test_wrapper(func: syn::ItemFn) -> TokenStream {
    let attributes = &func.attrs;
    let signature = &func.sig;
    let name = &signature.ident;
    let export_name = format_ident!("test_export_{}", name);
    let arguments = &signature.inputs;
…
```

```
$ m sel4test+wrapper
…
11:56:50.3653 [INFO] uart5: [output] Booting all finished, dropped to user space
11:56:50.3938 [INFO] uart5: [output] Node 0 of 1
11:56:50.3945 [INFO] uart5: [output] IOPT levels:     0
11:56:50.3956 [INFO] uart5: [output] IPC buffer:      0x646000
11:56:50.3965 [INFO] uart5: [output] Empty slots:     [1677 --> 8192)
11:56:50.3974 [INFO] uart5: [output] sharedFrames:    [0 --> 0)
11:56:50.3983 [INFO] uart5: [output] userImageFrames: [87 --> 1677)
11:56:50.3993 [INFO] uart5: [output] userImagePaging: [83 --> 85)
11:56:50.4001 [INFO] uart5: [output] untypeds:        [15 --> 83)
11:56:50.4008 [INFO] uart5: [output] Initial thread domain: 0
11:56:50.4025 [INFO] uart5: [output] Initial thread cnode size: 13
…
11:56:51.7405 [INFO] uart5: [output] Starting test suite sel4test
11:56:51.7421 [INFO] uart5: [output] Starting test 0: Test that there are tests
11:56:51.7435 [INFO] uart5: [output] Starting test 1: SYSCALL0000
11:56:51.7475 [INFO] uart5: [output] Starting test 2: SYSCALL0001
…
```



sel4gdb

logging

kmem

capscan

sel4-sys

04

Future Work

Google Research

Open source

Complete Hardware IP, Simulation/Infrastructure Tooling, System software, etc. to be available by EOY

Open up master repository (currently copied to GitHub).

Platform Support

Matcha SMC is riscv32imac, but seL4 kernel support was hacked and needs re-integration to support non-Matcha RISC-V devices.

aarch64 (rpi3/bcm2837) works on qemu but there are reports of problems running on real hardware.

Non-Matcha platforms need owners!

Functionality

SensorManager for managing sensor hardware.

Integrate more Security Core functionality.

Off-device communication: I2C or SPI, or possibly networking hardware & software.

ML connection to the Data Center (e.g. Project Oak).