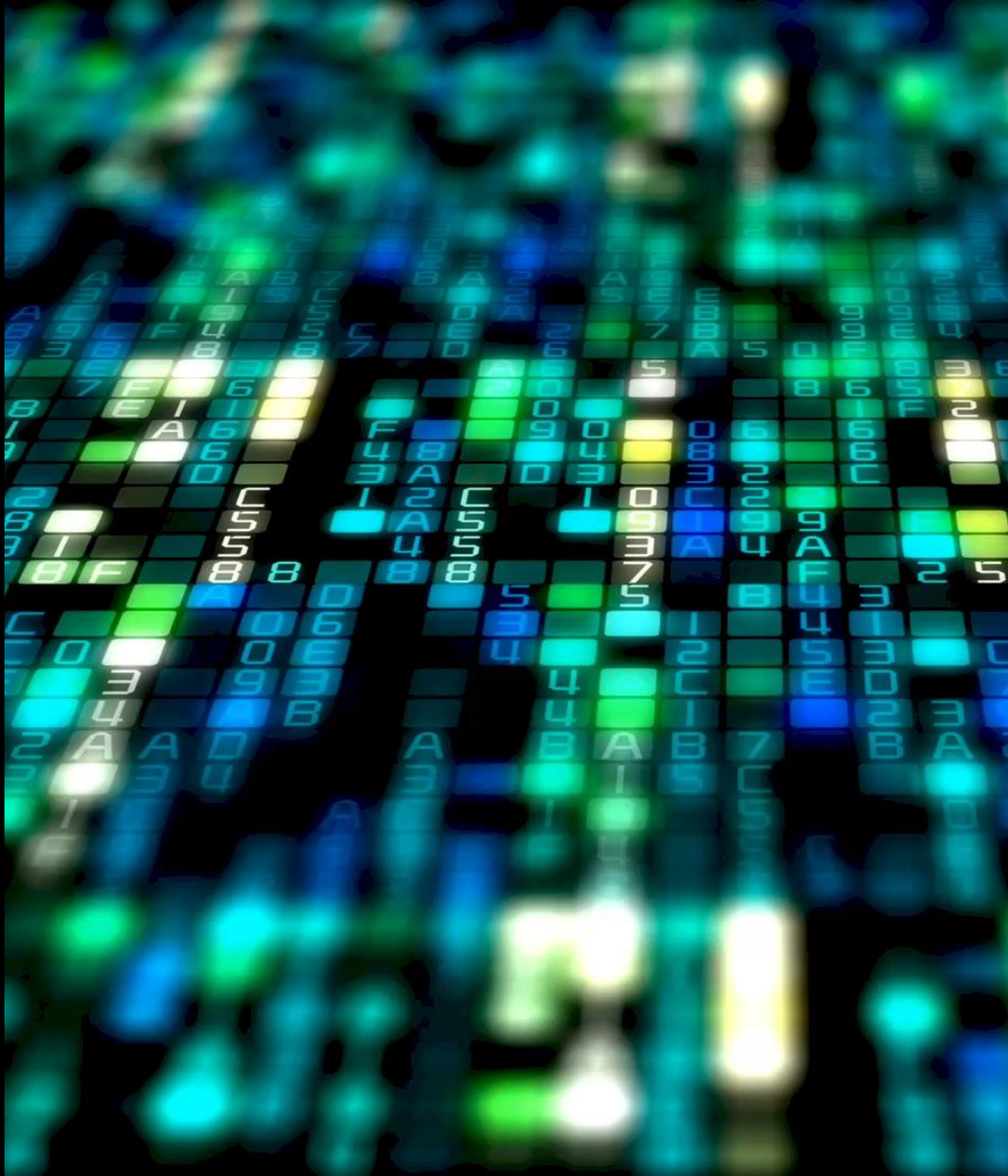
Running Certified Operating Systems under the seL4 Hypervisor

Chris Guikema

chris.guikema@dornerworks.com

DORNERWORKS Michigan, USA



Agenda

- What does a Hypervisor need to do?
- How is the CAmkES-VM (and VMM libraries) built around Linux?
- What is Deos?
- How can the CAmkES-VM Support Deos & Linux?
- How can we certify seL4 Hypervisor based systems?

Background

- What does a Hypervisor need to do?
- At a high level, it needs to:
  - Context switch Guest Operating Systems
  - Provide Stage 2 (ARM) or EPT (x86) Translations
  - Emulate necessary hardware resources that either seL4 owns or a VM may need to share
    - Interrupt controller, Serial, Timers, etc…
  - Handle guest faults and events
  - Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)

Background

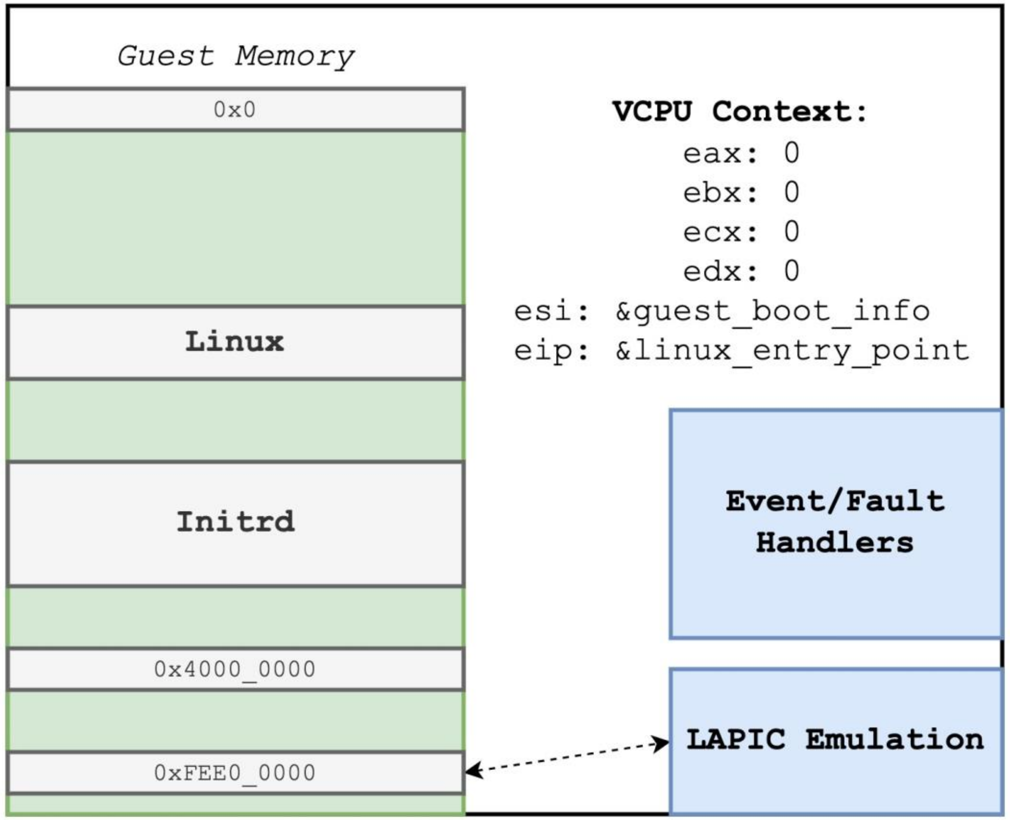
- What does a Hypervisor need to do?
- At a high level, it needs to:
  - Context switch Guest Operating Systems
  - Provide Stage 2 (ARM) or EPT (x86) Translations
  - Emulate necessary hardware resources that either seL4 owns or a VM may need to share
    - Interrupt controller, Serial, Timers, etc…
  - Handle guest faults and events
  - Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)

Background

NO seL4 KERNEL MODIFICATIONS WERE REQUIRED

- What does a Hypervisor need to do?
- At a high level, it needs to:
  - Context switch Guest Operating Systems
  - Provide Stage 2 (ARM) or EPT (x86) Translations
  - Emulate necessary hardware resources that either seL4 owns or a VM may need to share
    - Interrupt controller, Serial, Timers, etc...
  - Handle guest faults and events
  - Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)

Guest State Post VMM Initialization



What Operating Systems has the CAmkES-VM Run?

What Operating Systems has the CAmkES-VM Run?

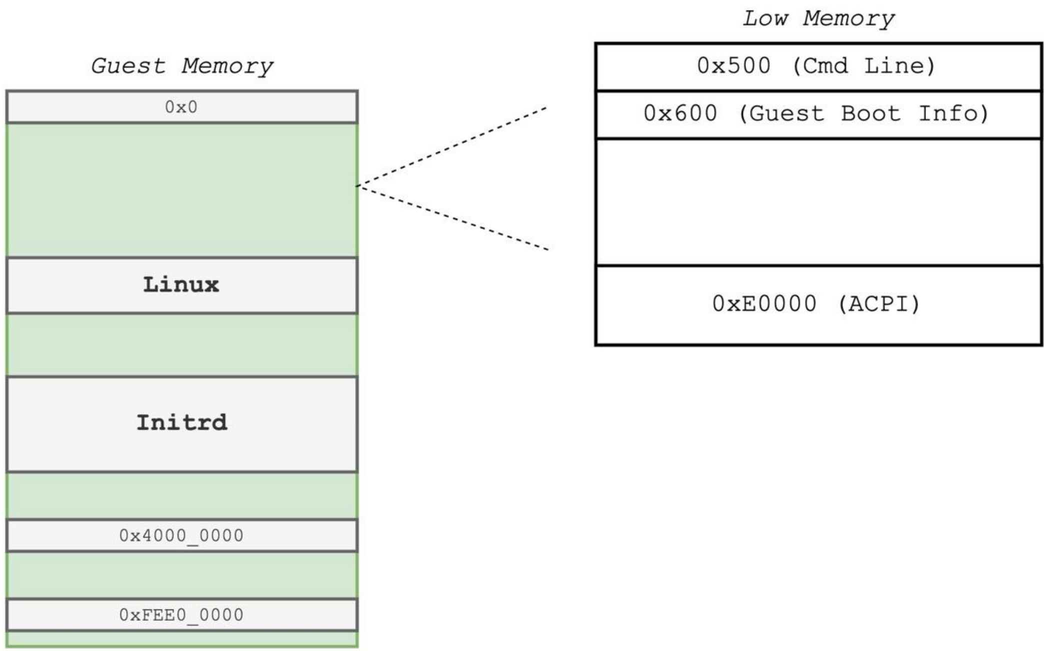
Linux™

Checklist

- Provide Stage 2 (ARM) or EPT (x86) Translations
- Emulate necessary hardware resources
- Handle guest faults and events
- Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)

Linux CAmkES-VM "Dependencies"

- **Hypervisor Requirement:** Provide Stage 2 (ARM) or EPT (x86) Translations



e820

0 -> 0x500 (2)
0x500 -> 0x9fc00 (1)
0x9fc00 -> 0x10003000 (2)
0x10003000 -> 0x30003000 (1)
0x30003000 -> 0x100000000 (2)
0x100000000 -> 0x160000000 (1)

Linux CAmkES-VM “Dependencies”

- **Hypervisor Requirement:** Provide Stage 2 (ARM) or EPT (x86) Translations

```c
static memory_range_t guest_ram_regions[] = {
    /* Allocate all the standard low memory areas */
    /* On x86 the BIOS loads the MBR to 0x7c00. But for this VMM,
     * we don't use MBR, so there is no need to exclude the MBR
     * bootstrap code region */
    {0x500, 0x80000 - 0x500},
    {0x80000, 0x9fc00 - 0x80000},
};

static memory_range_t guest_fake_devices[] = {
    {0xf0000, 0x10000}, // DMI
    {0xc0000, 0xc8000 - 0xc0000}, // VIDEO BIOS
    {0xc8000, 0xe0000 - 0xc8000}, // Mapped hardware and MISC
};
```

```c
for (i = 0; i < ARRAY_SIZE(guest_fake_devices); i++) {
    vm_memory_reservation_t *reservation = vm_reserve_memory_at(&vm, guest_fake_devices[i].base, guest_fake_devices[i].size,
                                                                default_error_fault_callback, (void *)NULL);
    ZF_LOGF_IF(!reservation, "Failed to create guest device reservation at %p", (void *)guest_fake_devices[i].base);
    error = map_frame_alloc_reservation(&vm, reservation);
    ZF_LOGF_IF(error, "Failed to map guest device reservation at %p", (void *)guest_fake_devices[i].base);
}
```

```c
/* Do we need to do any early reservations of guest address space? */
for (i = 0; i < ARRAY_SIZE(guest_ram_regions); i++) {
    error = vm_ram_register_at(&vm, guest_ram_regions[i].base, guest_ram_regions[i].size, false);
    ZF_LOGF_IF(error, "Failed to alloc guest ram at %p", (void *)guest_ram_regions[i].base);
}
```

Linux CAmkES-VM "Dependencies"

- **Hypervisor Requirement:** Provide Stage 2 (ARM) or EPT (x86) Translations

```
/* Allocate guest ram. This is the main memory that the guest will actually get
 * told exists. Other memory may get allocated and mapped into the guest */
bool paddr_is_vaddr;
paddr_is_vaddr = false;
// allocate guest ram in 512MiB chunks. This prevents extreme fragmentation of the
// physical address space when a large amount of guest RAM has been reuqested.
// An important side affect is that if the requested RAM is large, and there are
// devices or other regions in the lower 4GiB of the guest address space then we will
// still allocate some RAM in the lower 4GiB, which a guest may require to run correctly.
size_t remaining = MiB_TO_BYTES(guest_ram_mb);
while (remaining > 0) {
    size_t allocate = MIN(remaining, MiB_TO_BYTES(512));
    uintptr_t res_addr = vm_ram_register(&vm, allocate);
    ZF_LOGF_IF(!res_addr, "Failed to allocate %lu bytes of guest ram. Already allocated %lu.",
               (long)allocate, (long)(MiB_TO_BYTES(guest_ram_mb) - remaining));
    remaining -= allocate;
}
```

Linux CAmkES-VM "Dependencies"

- **Hypervisor Requirement:** Provide Stage 2 (ARM) or EPT (x86) Translations

```
[    0.000000] BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x00000000000004ff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000000500-0x000000000009fbff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009fc00-0x0000000010002fff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000010003000-0x0000000030002fff] usable
[    0.000000] BIOS-e820: [mem 0x0000000030003000-0x00000000ffffffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000100000000-0x000000015fffffff] usable
```

Checklist

- ~~Provide Stage 2 (ARM) or EPT (x86) Translations~~
- Emulate necessary hardware resources
- Handle guest faults and events
- Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)

Linux CAmkES-VM "Dependencies"

- **Hypervisor Requirement:** Emulate necessary hardware resources
- On x86, there are (usually) 4 available timers:
  - PIT
  - HPET
  - LAPIC Timer
  - TSC

Linux CAmkES-VM "Dependencies"

- **Hypervisor Requirement:** Emulate necessary hardware resources
- On x86, there are (usually) 4 available timers:
  - PIT
  - HPET
  - LAPIC Timer
  - TSC

Linux CAmkES-VM "Dependencies"

- **Hypervisor Requirement:** Emulate necessary hardware resources
- On x86, there are (usually) 4 available timers:
  - PIT
  - HPET
  - LAPIC Timer
  - TSC
- Userspace VMM connects to TimeServer component to set absolute timeouts
- Timeserver is backed by either the physical PIT or the HPET hardware
- MMIO/IOPort Emulation sets timeouts for Linux
- Linux uses the PIT/HPET to calibrate TSC, and for system tick IRQ

```
[    1.855732] clocksource: Switched to clocksource tsc
```

Checklist

- ~~Provide Stage 2 (ARM) or EPT (x86) Translations~~
- ~~Emulate necessary hardware resources~~
- Handle guest faults and events
- Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)

Linux CAmkES-VM "Dependencies"

- **Hypervisor Requirement:** Handle guest faults and events
- Lots of expected faults the VMM handles:
  - EPT Violations
  - CPUID Calls
  - MSR Reads/Writes
- Let's follow the EPT Violation Path:
  1. Get EPT Violation Physical Address and R/W
  2. Read the faulting instruction (e.g 0x44 0x8b 0x20 == mov r12d, [rax])
  3. Decode the instruction to determine which register to read from/write to
  4. Call a fault handler (LAPIC Emulation)
  5. Get/Set the Fault Data (decodes the Instruction *again*)
  6. Set the Instruction Pointer to the next instruction

```
static vm_exit_handler_fn_t x86_exit_handlers[VM_EXIT_REASON_NUM] = {
    [EXIT_REASON_PENDING_INTERRUPT] = vm_pending_interrupt_handler,
    [EXIT_REASON_CPUID] = vm_cpuid_handler,
    [EXIT_REASON_MSR_READ] = vm_rdmsr_handler,
    [EXIT_REASON_MSR_WRITE] = vm_wrmsr_handler,
    [EXIT_REASON_EPT_VIOLATION] = vm_ept_violation_handler,
    [EXIT_REASON_CR_ACCESS] = vm_cr_access_handler,
    [EXIT_REASON_IO_INSTRUCTION] = vm_io_instruction_handler,
    [EXIT_REASON_HLT] = vm_hlt_handler,
    [EXIT_REASON_VMX_TIMER] = vm_vmx_timer_handler,
    [EXIT_REASON_VMCALL] = vm_vmcall_handler,
};
```

Linux CAmkES-VM "Dependencies"

- **Hypervisor Requirement:** Handle guest faults and events

```
static const struct decode_table decode_table_1op[] = {
    [0 ... MAX_INSTR_OPCODES] = {DECODE_INSTR_INVALID, decode_invalid_op},
    [0x88] = {DECODE_INSTR_MOV, decode_modrm_reg_op},
    [0x89] = {DECODE_INSTR_MOV, decode_modrm_reg_op},
    [0x8a] = {DECODE_INSTR_MOV, decode_modrm_reg_op},
    [0x8b] = {DECODE_INSTR_MOV, decode_modrm_reg_op},
    [0x8c] = {DECODE_INSTR_MOV, decode_modrm_reg_op},
    [0xc6] = {DECODE_INSTR_MOV, decode_imm_op},
    [0xc7] = {DECODE_INSTR_MOV, decode_imm_op}
};

static const struct decode_table decode_table_2op[] = {
    [0 ... MAX_INSTR_OPCODES] = {DECODE_INSTR_INVALID, decode_invalid_op},
    [0x6f] = {DECODE_INSTR_MOVQ, decode_modrm_reg_op}
};
```

Checklist

- ~~Provide Stage 2 (ARM) or EPT (x86) Translations~~
- ~~Emulate necessary hardware resources~~
- ~~Handle guest faults and events~~
- Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)

Linux CAmkES-VM “Dependencies”

- **Hypervisor Requirement:** Create interfaces between VMs or a VM and an seL4 Thread
- Example: Virtio-Net
  1. seL4 emulates the PCI Bus
  2. seL4 places a virtio-net device on the PCI bus, accessed via *IOPorts*
  3. IO Port handlers read virtio-net descriptors and route packet to destination via virtqueues
- Linux is really flexible!
  - Supports legacy and modern VirtIO interfaces
  - Reads access information from a PCI scan
    - Can support either IO Ports or MMIO access

Summary

- The CAmkES VM for x86 was specifically built around running Linux as a VM
- The Memory Configuration, VCPU Initialization, Hardware Emulation, and VirtIO all assume a Linux guest
- However, Linux isn't the only Operating System out there
- The CAmkES-VM should also support other Operating Systems
  - VxWorks
  - RTEMS
  - Deos

What are DO-178C & Design Assurance Levels?

- **DO-178C, Software Considerations in Airborne Systems and Equipment Certification** is the primary document by which the certification authorities such as the FAA approve all commercial software-based aerospace systems.
- Design Assurance Levels (DAL)
  - Determined from safety assessment and hazard analysis
- What about formal methods?
  - DO-333 discusses using formal methods to certify against DO-178C

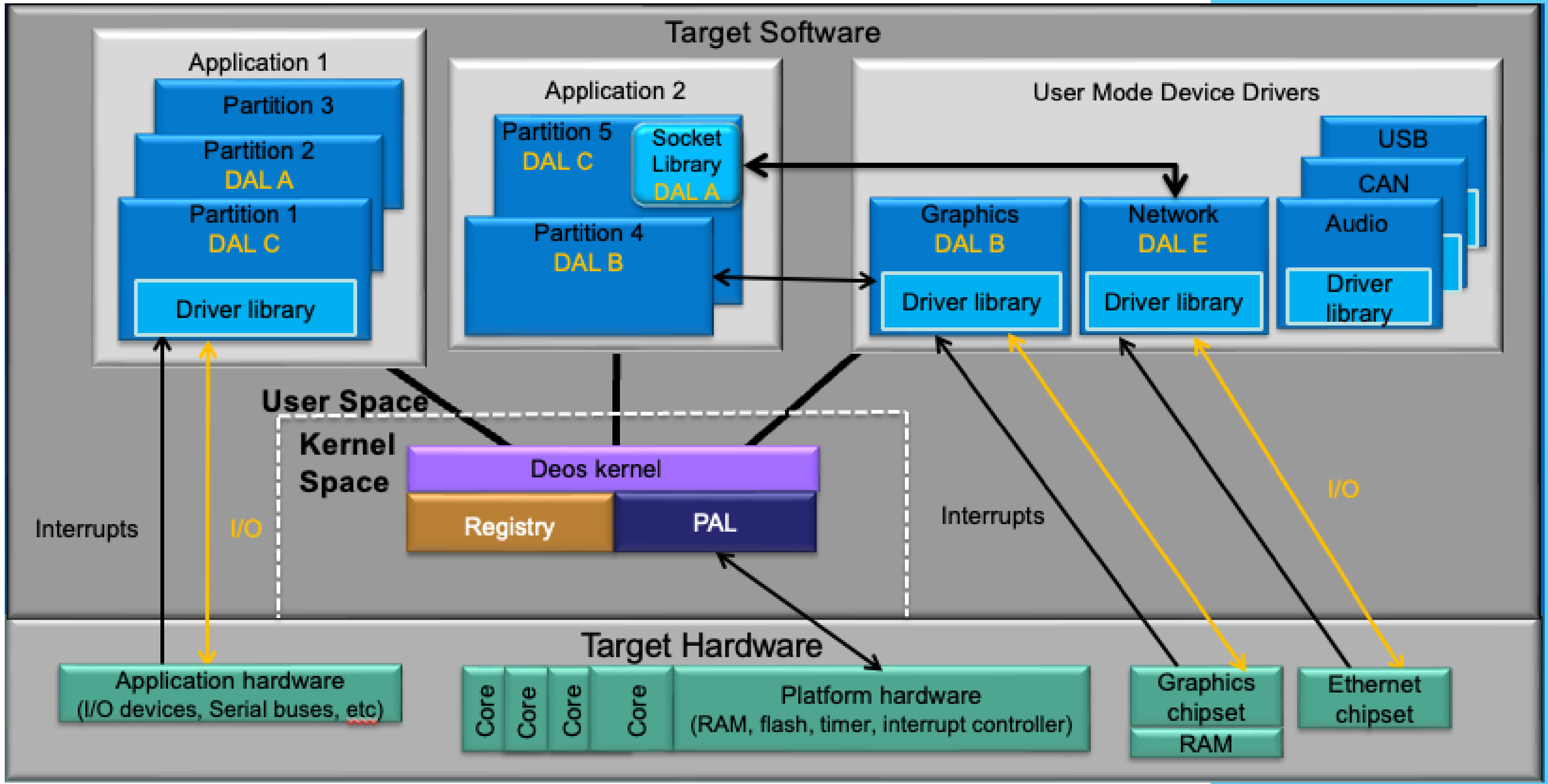
Design Assurance Level (DAL)	Description	Failure Rate	Example
Level A - Catastrophic	Failure causes crash, deaths	< 1 x 10^9 / flight-hr	Flight Controls
Level B - Hazardous	Failure may cause crash, deaths	< 1 x 10^7 / flight-hr	Braking System
Level C - Major	Failure may cause stress, injuries	< 1 x 10^5 / flight-hr	Backup Systems
Level D - Minor	Failure may cause inconvenience	No safety metric	Ground Navigation Systems
Level E - No Effect	No safety effect on passenger/crew	No safety metric	Passenger Entertainment

What is Deos?

- Certified, Safety-Critical RTOS developed by DDC-I
  - High performance, Multicore
  - Supports ARM, x86, PPC
  - Conforms to FACE Technical Standard v3.1
- Verifiable to DO-178C Design Assurance Level (DAL) A since 1998
- Enables time, space, and resource partitions
  - Like seL4, it uses user mode drivers, making it easy to build a driver to the DAL required
    - All I/O is not required to be DAL A
  - DAL-A Linker/Loader for Binary Modularity
    - Enables software reuse & certification
    - Each application and library has a DAL with a full certification package

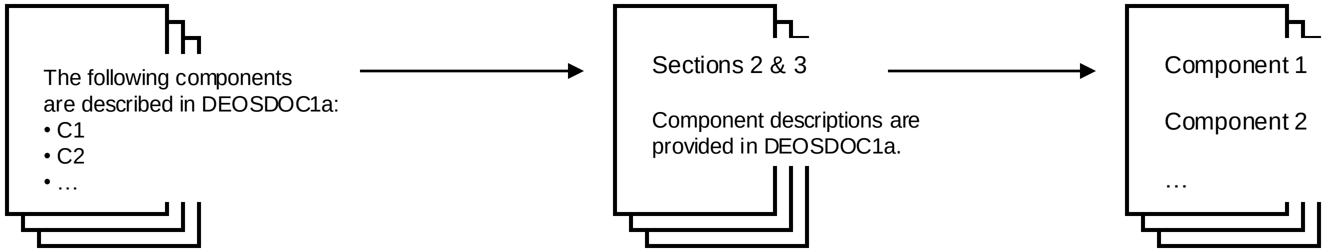
What is Deos?

- Unmatched record of deployment, support, and certification
  - > 10,000 aircraft
  - > 10,000,000 flight hours
  - > 40 aircraft types
  - > 100 certifications
- Performance
  - Multicore – Safe Scheduling, Cache Partitioning
  - Quick boot up times

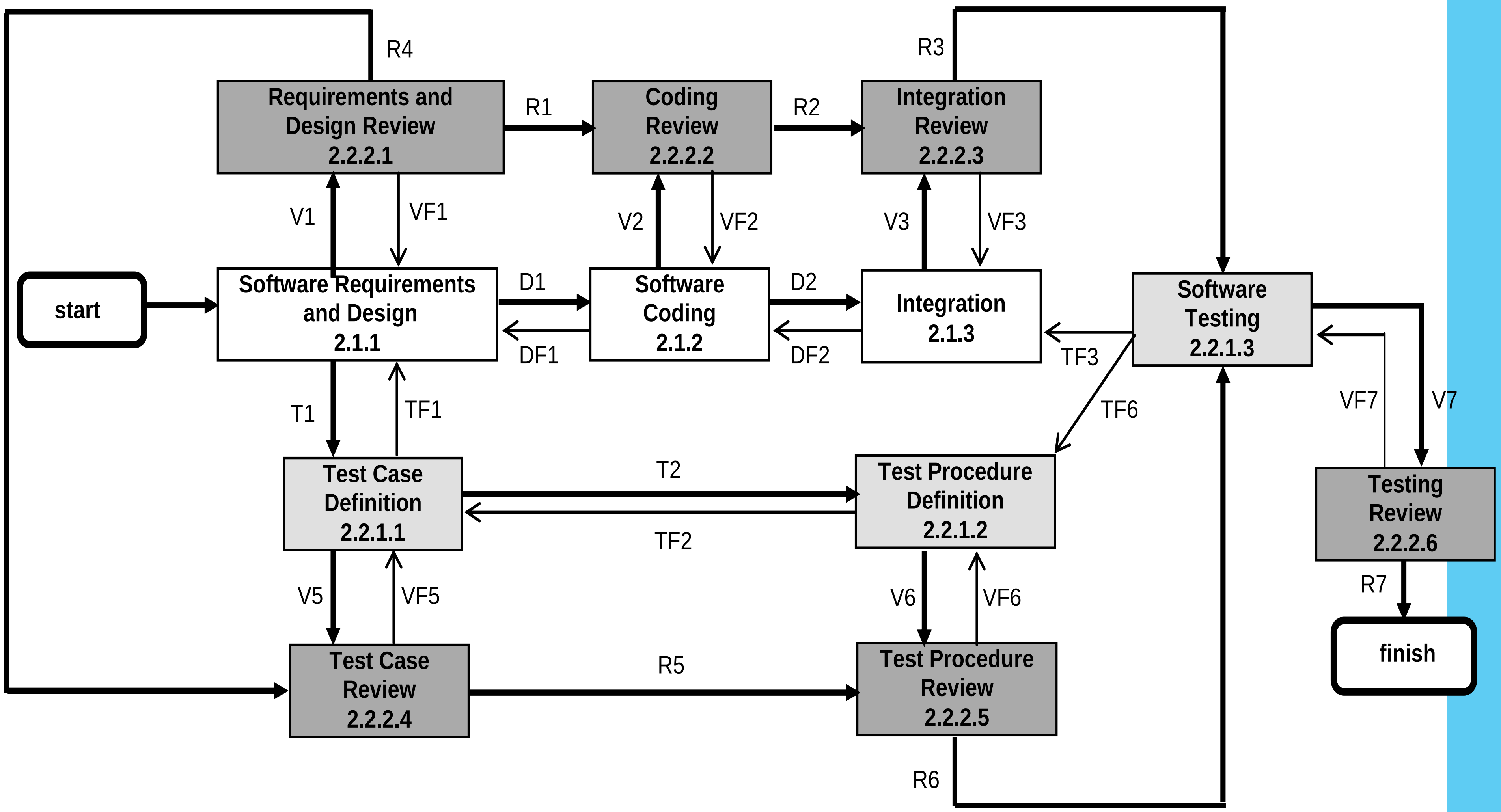


Deos Certification Process

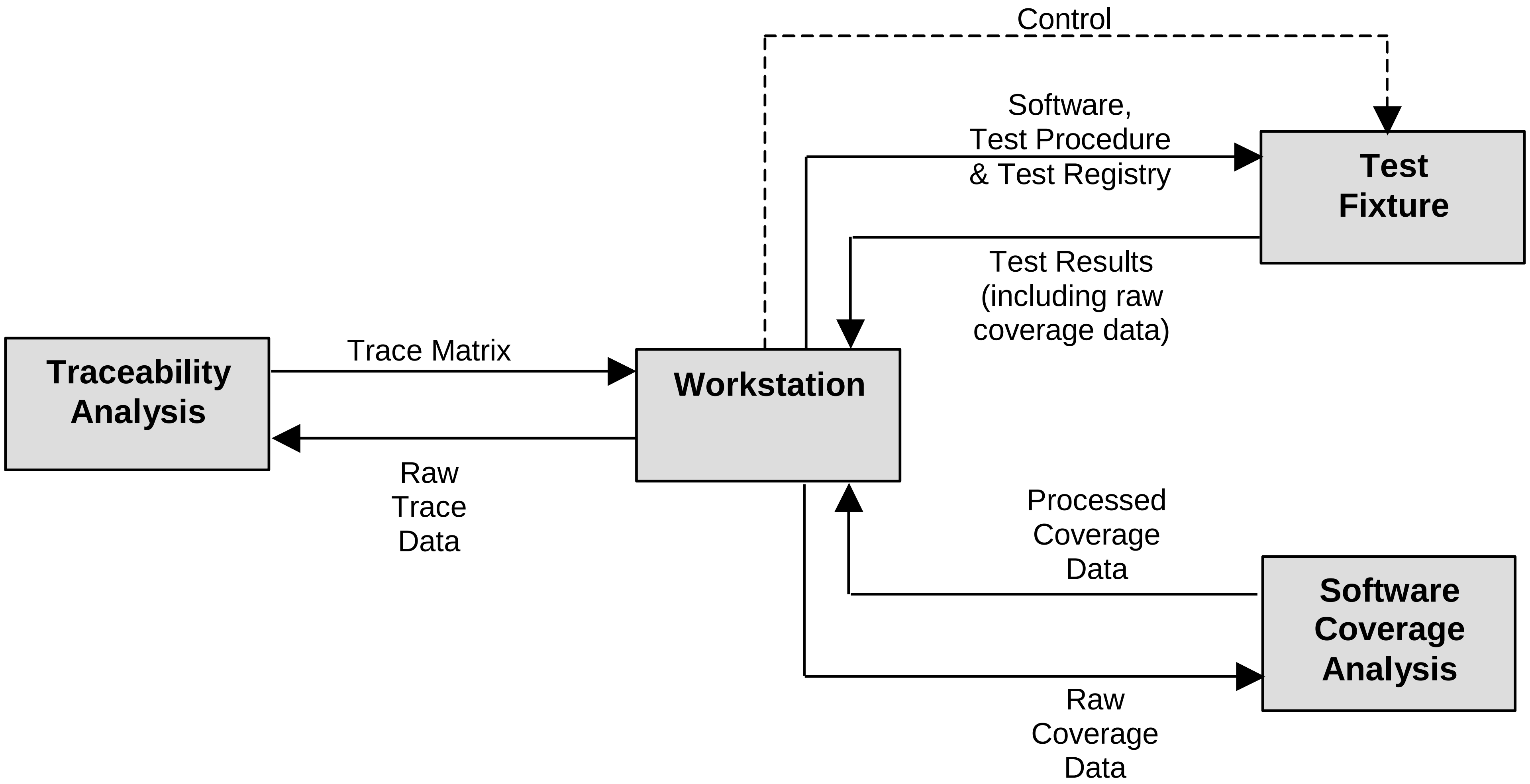
Title	Doc Number
"Plan for Software Aspects of Certification for DDCI Software" (PSAC)	DDCIDOC1
"Deos Software Component Descriptions" (DEOSDOC1a)	DEOSDOC1a
"DDCI Additional Considerations Document" (DDCIDOC1b)	DDCIDOC1b
"Software Development and Verification Plan for Software" (SDVP)	DDCIDOC2
"Software Configuration Management Plan for DDCI Software" (SCMP)	DDCIDOC3
"Software Quality Assurance Plan for DDCI Software" (SQAP)	DDCIDOC4



Deos Software Life Cycle (DSLC)

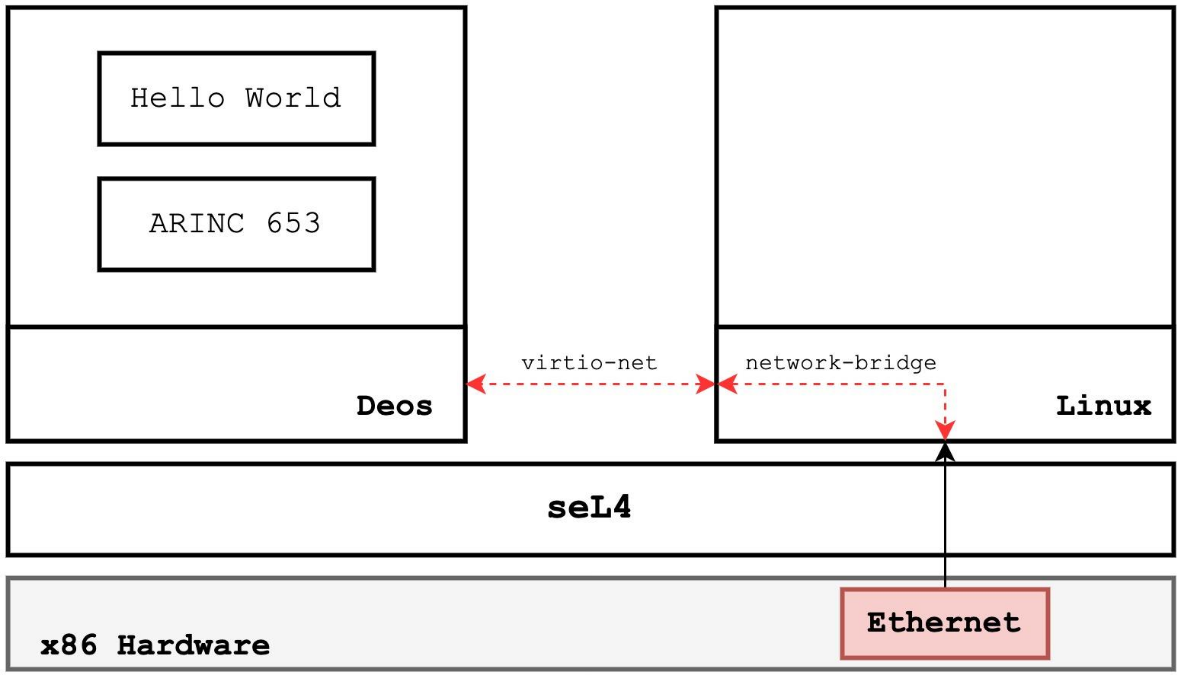


Deos Test Environment



Goal:

- Two VM Configuration
- Deos + Linux
  - Deos uses QEMU-x86_64 Platform
- VirtIO Network Channel between VMs
  - Network bridge ensures Deos has access to external network
- Any changes we make to the CAmkES-VM need to be backwards compatible!
  - And should be expandable for other Operating Systems
- Allows for general purpose applications to run in Linux alongside DAL certified applications running in Deos



Checklist

- Provide Stage 2 (ARM) or EPT (x86) Translations
- Emulate necessary hardware resources
- Handle guest faults and events
- Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)

Adding Deos Support to CAmkES-VM

- **Hypervisor Requirement:** Provide Stage 2 (ARM) or EPT (x86) Translations
- **Deos Requirement**: Static Memory Map from 0x0 -> 0x4000_0000

Adding Deos Support to CAmkES-VM

- **Hypervisor Requirement:** Provide Stage 2 (ARM) or EPT (x86) Translations
- **Deos Requirement**: Static Memory Map from 0x0 -> 0x4000_0000
- Problem: x86 CAmkES-VM uses anonymous memory regions for guest RAM
    - e820 Map allows memory to be “anywhere”
- Solution: ARM CAmkES-VM already provides static memory maps using the “vm_ram_register_at” functions
- Still need to provide a method for the VMM to specify a memory range, instead of a “guest_ram_mb” parameter

Adding Deos Support to CAmkES-VM

```
vm0.vm_address_config = {
    "ram_base" : VAR_STRINGIZE(VM_RAM_BASE),
    "ram_paddr_base" : VAR_STRINGIZE(VM_RAM_BASE),
    "ram_size" : VAR_STRINGIZE(VM_RAM_SIZE),
    "high_ram_size" : VAR_STRINGIZE(0x1000),
    "kernel_addr" : VAR_STRINGIZE(VM_KERNEL_ADDR),
    "initrd_addr" : VAR_STRINGIZE(VM_INITRD_ADDR),
};

vm0.vm_image_config = {
    "kernel_name" : "deosBoot.exe",
    "kernel_relocs_name" : "",
    "initrd_name" : "composite.darc",
    "kernel_cmdline" : DEOS_CMDLINE,
    "map_one_to_one" : false,
    "provide_initrd": true,
    "is_linux": false,
    "is_deos": true,
};
```

```
typedef struct {

    vm_ram_t low_ram;
    vm_ram_t high_ram;

    uintptr_t kernel_addr;
    uintptr_t kernel_align;
    uintptr_t initrd_addr;
    uintptr_t guest_ram_mb;

    bool provide_initrd;
    bool map_one_to_one;
```

Adding Deos Support to CAmkES-VM

```
IF VM == Linux:
        vm_ram_register       // Pulls from anon regions
        find_large_region(&addr)
ELSE IF VM == Deos:
        vm_ram_register_at    // Specifies region to map
        addr = vm_config.kernel_addr

vm_load_guest_kernel(addr)
```

Adding Deos Support to CAmkES-VM

- Linux uses the boot_info struct
  - Tells Linux crucial boot information, including kernel, ramdisk, memory, and command line
- Deos expects to be booted from a Multiboot compliant bootloader
  - Therefore, Deos needs a multiboot struct in its initial memory
  - Contains much the same information as Linux boot_info struct
- Libraries modified to search for multiboot header in first 2048 bytes of kernel image

```
context.eax = MULTIBOOT_BOOTLOADER_MAGIC;
context.ebx = guest_boot_info_addr;
context.ecx = 0;
context.edx = 0;
context.esi = 0;
```

Checklist

- ~~Provide Stage 2 (ARM) or EPT (x86) Translations~~
- Emulate necessary hardware resources
- Handle guest faults and events
- Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)

Adding Deos Support to CAmkES-VM

- **Hypervisor Requirement:** Emulate necessary hardware resources
- **Deos Requirement**: QEMU-x86_64 Platform needs the LAPIC Timer

Adding Deos Support to CAmkES-VM

- **Hypervisor Requirement:** Emulate necessary hardware resources
- **Deos Requirement**: QEMU-x86_64 Platform needs the LAPIC Timer
- Problem: seL4 does NOT support the LAPIC timer
- Solution: Leverage the Open-source Community and pull in a LAPIC Timer PR
- Initialize timers based on VM Configuration

Adding Deos Support to CAmkES-VM

```
typedef struct vm_timers {
    bool use_hpet;
    bool use_pit;
    bool use_lapic;
} vm_timers_t;
```

```
vm0.vm_timer_config = {
    "use_pit": false,
    "use_hpet": false,
    "use_lapic": true,
};
```

```
/*- if vm_timer_config -*/

    .timers = {
        .use_pit = /*? vm_timer_config.get('use_pit') ?*/,
        .use_hpet = /*? vm_timer_config.get('use_hpet') ?*/,
        .use_lapic = /*? vm_timer_config.get('use_lapic') ?*/,
    },

/*- else -*/

    .timers = {
        .use_pit = true,
#ifdef CONFIG_VMM_USE_HPET
        .use_hpet = true,
#else
        .use_hpet = false,
#endif
        .use_lapic = false,
    },

/*- endif -*/
```

Adding Deos Support to CAmkES-VM

```
IF VM_Config.PIT:
       pit_init()
IF VM_Config.HPET:
       hpet_init()
IF VM_Config.LAPIC:
       lapic_init()
```

Checklist

- ~~Provide Stage 2 (ARM) or EPT (x86) Translations~~
- ~~Emulate necessary hardware resources~~
- Handle guest faults and events
- Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)

Adding Deos Support to CAmkES-VM

- **Hypervisor Requirement:** Handle guest faults and events
- **Deos Requirement:** Handle (extra) guest faults and events
- Deos's EPT violations required adding support for 2 extra MOV instructions, and fixing the MOV_IMM emulation
  - EPT Violations were ignoring the Immediate value, so the LAPIC wasn't properly initialized

Adding Deos Support to CAmkES-VM

- **Hypervisor Requirement:** Handle guest faults and events
- **Deos Requirement:** Handle (extra) guest faults and events
- Deos's EPT violations required adding support for 2 extra MOV instructions, and fixing the MOV_IMM emulation
  - EPT Violations were ignoring the Immediate value, so the LAPIC wasn't properly initialized

```
/* Partial support to decode an instruction for a memory access
   This is very crude. It can break in many ways. */
```

Adding Deos Support to CAmkES-VM

- QEMU to the rescue!
- QEMU has an x86 decoder and emulator pulled in from the Veertu Hypervisor
- Supports decoding all x86 OPCodes
- Ported the decoder and emulator for use in the seL4 VMM Libraries
  - Emulator required a bit more porting to integrate with EPT Violation path
  - EPT Violations just require the register to read from / write to, and *sometimes* an immediate value

Adding Deos Support to CAmkES-VM

```
/* For an EPT Read Violation, the value at the faulting address is stored in the
 * register, so we need to check op[0]. For an EPT Write Violation, the destination
 * is the faulting address, and the source is either a register or an immediate value.
 * Either way, we need to check op[1].
 */
if (is_vcpu_read_fault(env)) {
    assert(decode->op[0].type == X86_VAR_REG);
    decode->reg = decode->op[0].reg;
} else {
    if (decode->op[1].type == X86_VAR_IMMEDIATE) {
        decode->value = decode->op[1].val;
        decode->use_value = true;
    } else if (decode->op[1].type == X86_VAR_REG) {
        decode->reg = decode->op[1].reg;
    } else {
        ZF_LOGF("Handle type %d", decode->op[1].type);
    }
}
```

Checklist

- ~~Provide Stage 2 (ARM) or EPT (x86) Translations~~
- ~~Emulate necessary hardware resources~~
- ~~Handle guest faults and events~~
- Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)

Adding Deos Support to CAmkES-VM

- **Hypervisor Requirement:** Create interfaces between VMs or a VM and an seL4 Thread (VirtIO)
- **Deos Requirement:** The QEMU-x86_64 Deos Target needs a Virtio Ethernet Device

Adding Deos Support to CAmkES-VM

- **Hypervisor Requirement:** Create interfaces between VMs or a VM and an seL4 Thread (VirtIO)
- **Deos Requirement:** The QEMU-x86_64 Deos Target needs a Virtio Ethernet Device
- Deos has a virtio-net library, and VirtIO is a standard, so it should drop into place
- Two Problems:
  1. Deos assumes a Modern VirtIO Backend
  2. Deos uses MMIO regions to access the VirtIO Backend

Adding Deos Support to CAmkES-VM

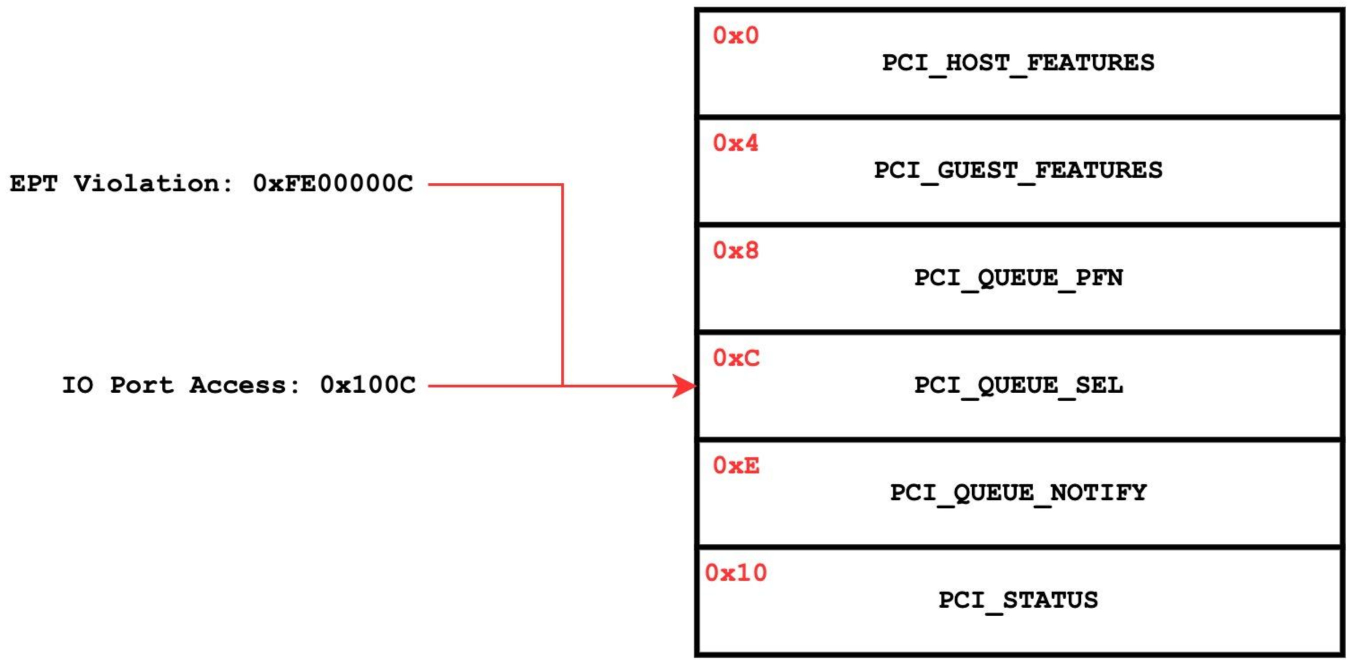
- **Solution #1:** The Deos Library can be configured to use Legacy VirtIO

virtio-net/code/virtio_ethernetif.cpp

```diff
@@ -457,7 +457,7 @@ uint8_t lwip_driver_init(void *netifp)
      * Ref: lwIP's virtio-net driver, deos-virtio.cpp::deos_virtio_net_open()
      * TODO: Justify magic number offsets.
      */
-    virtio_net_sc.vio_dev.is_modern = 1;
+    virtio_net_sc.vio_dev.is_modern = 0;
```

Adding Deos Support to CAmkES-VM

- **Solution #2:** We can modify seL4 VirtIO backend to support MMIO access
- common_make_virtio_net_mmio
  - Does all VirtIO initialization
  - Create EPT Fault handler for specified MMIO region
  - Uses same VirtIO offsets



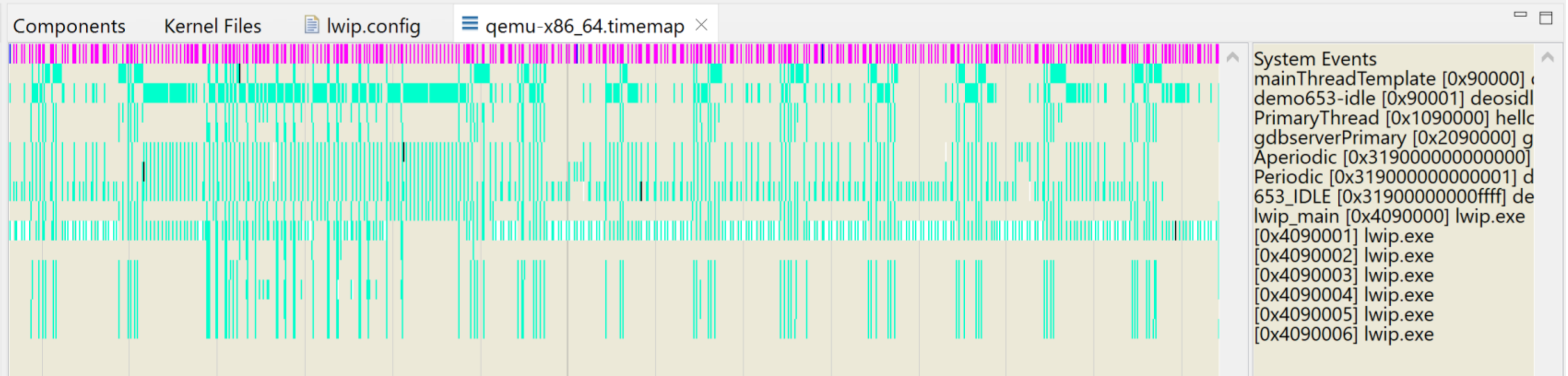
Checklist

- ~~Provide Stage 2 (ARM) or EPT (x86) Translations~~
- ~~Emulate necessary hardware resources~~
- ~~Handle guest faults and events~~
- ~~Optionally, create interfaces between VMs or a VM and an seL4 Thread (VirtIO)~~

Deos Output

```
Hello, World! System Tick = 18740
Partition restart counter : 12

   Periodic Loop Count : 37
   Aperiodic Loop Count: 125
   653 Time            : 368A7EDD60











0ACT:4E9 rx:223 tx:2F2 drop:0                                  TICK:   4934
if:lo:127.0.0.1/8               if:v1:192.168.19.100/24        Deos   1 core
```

Deos Output



653 | Processes | Threads | Exceptions | Schedulers | ProcessEvents

653Partitions | 653Processes | 653Objects | 653HealthMonitor

Select Quotas

○ Initial Quotas ○ Remaining Quotas ◉ Both Values

#	Partition Qu...	Partition Ha...	Partition Name	.EXE Name	Identifier	Lock Level	Operating ...	Start Conditi...	653 Processes	Stack Space	Sampling
1	Initial	0x3070000	demo653	demo653.exe	1	0	NORMAL	PARTITION_...	2	32768 B	0
2	Remaining	0x3070000	demo653	demo653.exe	1	0	NORMAL	PARTITION_...	0	24576 B	0

How would this system be certified?

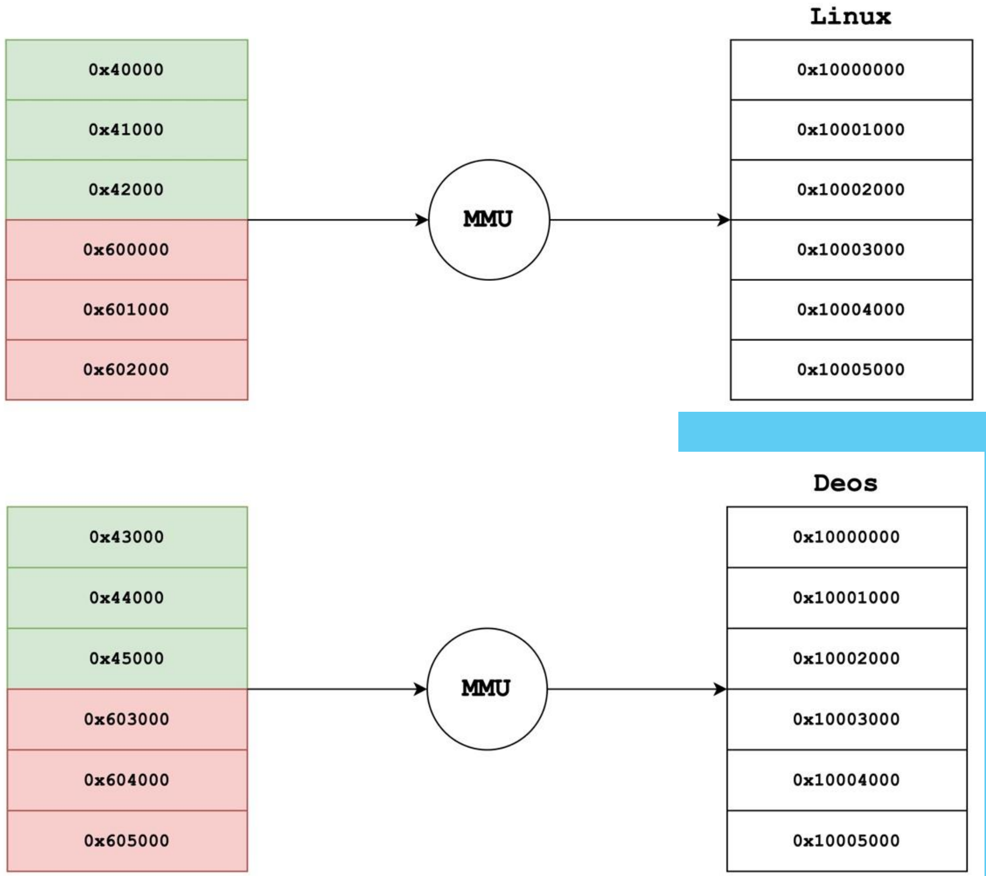
- Goal: Reuse **existing** Deos certification artifacts
- First consideration: The Hypervisor itself would need to be certified
  - RTCA DO-333 Formal Methods Supplement to DO-178C and DO-278A provides guidance to software developers wishing to use formal methods in the certification of airborne systems [1]
  - For seL4, Hypervisor Configurations would need to be certified
    - AARCH64 has Hypervisor Mode verified
    - x86, RISC-V do not
  - Assuming seL4 has the proper verified configurations, *the VMM must also have certification artifacts*

How would this system be certified?

- Second consideration: The Hypervisor itself needs to provide as close to an identical execution environment as possible
  - MCS Configuration would be a must!
    - Can give VM 100% of Core Execution. No round-robin scheduling tick
  - Any servers would need to exist on secondary cores
    - This would include VirtIO processing
  - Need to consider the effects of the cache
    - Deos has a patented cache coloring method to prevent interference in multicore environments

How would this system be certified?

- Guest memory is backed with “random” untyped objects
- MMU is used to give each VM a standard address space
  - VMs are free to virtualize their own memory
- VM’s running simultaneously on different cores can still share a cache line
  - This can force the processor to walk the MMU tables, effecting the Worst Case Execution Time (WCET) of VM applications
- Solution: provide known untyped objects to back guest memory
  - Allows VMs to use their own cache coloring mechanisms



System Certification Summary

- Start with formally verified seL4 Hypervisor configuration
  - Use DO-333 framework to provide certification arguments for the seL4 Kernel
- Update userspace VMM and provide certification documentation
- Provide testing to prove minimal execution environment differences for Deos guest
- Reuse Deos certification artifacts

What Conclusions Can We Draw?

- The CAmkES-VM is capable of running non-Linux guests
- Certification of seL4 Hypervisor based systems is possible with the right amount of funding
- Current gaps:
  - Verified Hypervisor configurations for x86_64 and RISC-V
  - Verified configurations for Multicore & MCS
  - Certification artifacts for userspace VMM
- This setup would allow certified guests like Deos to re-use their certification artifacts when running underneath the seL4 Hypervisor

Questions?