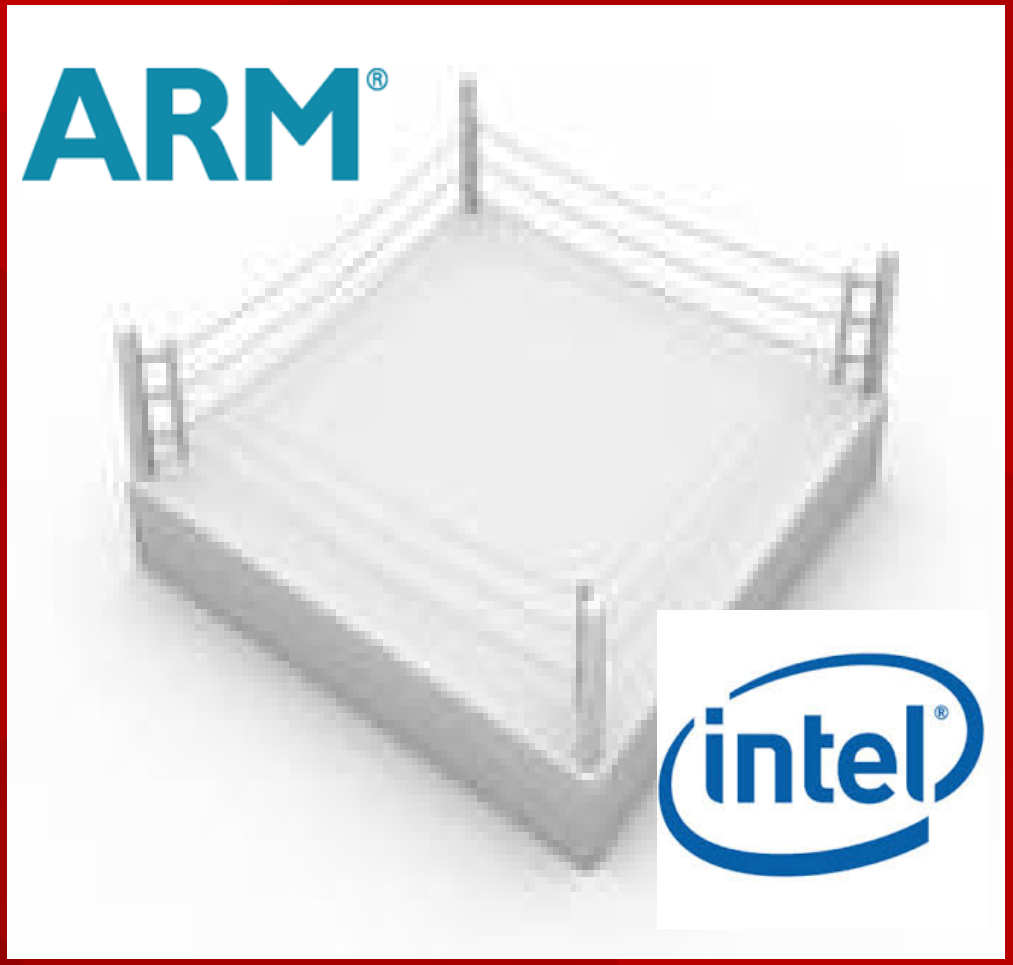
KVM: Has ARM done it better?



Brno devconf 2015
Drew Jones

Overview

- Introduction
- Scope of comparison
- ARM Virt. vs. VMX
- Conclusion
- Q/A

Introduction: Terminology

- Operating system
  - Software that manages/provides applications with resources and services
  - That thing you boot in order to get to Facebook

Introduction: Terminology

- Operating system
  - Software that expects **full** access to hardware
- Virtualization
  - The means to run an unmodified OS with **restricted** access to hardware
- Hypervisor
  - Software that manages/provides isolated OSes with resources and services
  - That thing you boot, in order to boot that other thing, that gets you to Facebook

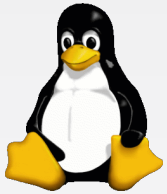
Introduction: Terminology

- Virtualization extensions
  - Processor support for virtualization, a.k.a hardware-assisted virtualization
  - How well do they assist?

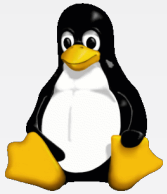
Introduction: Hypervisor

- KVM
  - Leave the resource management to OS software

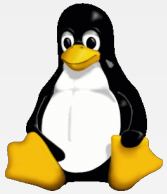
Introduction: Hypervisor

- KVM
  - Leave the resource management to OS software
    - Linux 

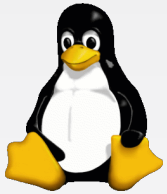
Introduction: Hypervisor

- KVM
  - Leave the resource management to OS software
    - Linux 
    - But, need to change the way those resources are provided – the guest OS expects register poking to work

Introduction: Hypervisor

- KVM
  - Leave the resource management to OS software
    - Linux 
    - But, need to change the way those resources are provided – the guest OS expects register poking to work
  - Leave any necessary emulation to emulators

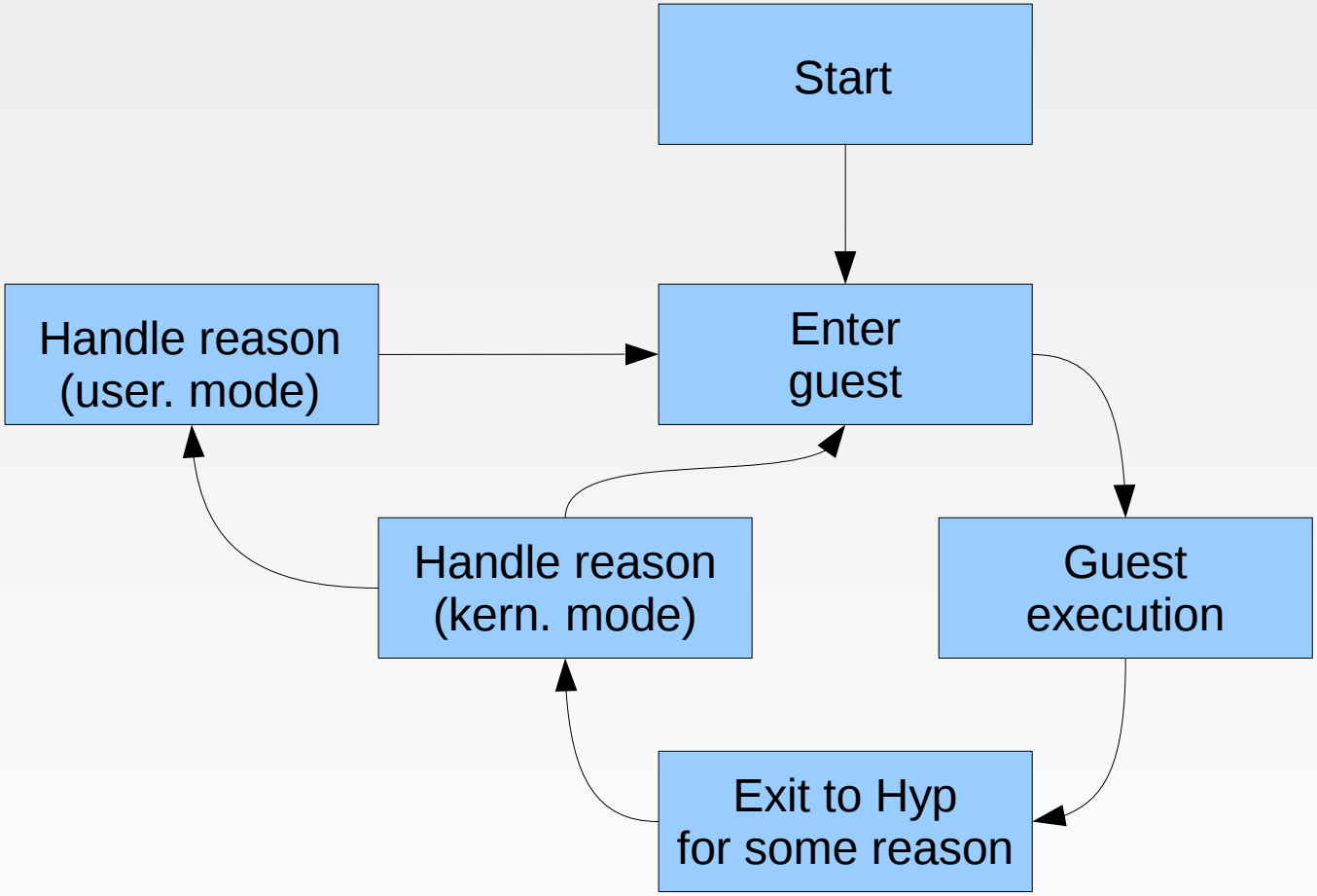
Introduction: Hypervisor

- KVM
  - Leave the resource management to OS software
    - Linux 
    - But, need to change the way those resources are provided – the guest OS expects register poking to work
  - Leave any necessary emulation to emulators
    - QEMU 
  - Connect everything and fill in the gaps

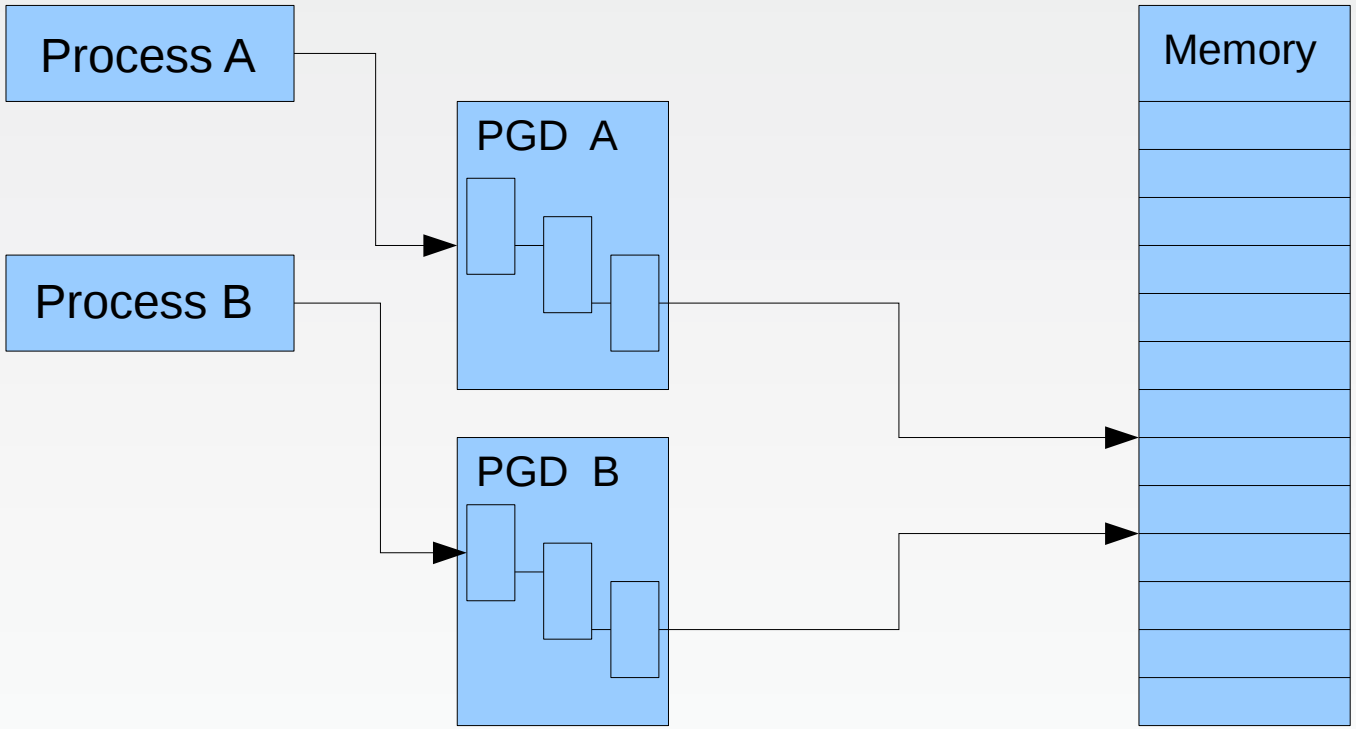
Introduction: Virt. extensions

- Intel's Virt. extensions (VT-x “VMX”)
  - 2005 – initial release (CPU virt.)
  - 2008 – EPT (memory)
  - 2010 – real mode (MMU off) guest launching
  - 2014 – APICv (reduce interrupt virt. overhead)
- ARM's Virt. extensions (ARM v7 & v8)
  - 2013 (v7) / 2015 (v8) – 2nd stage translation, VGIC
  - Soon (v8.1) – VHE

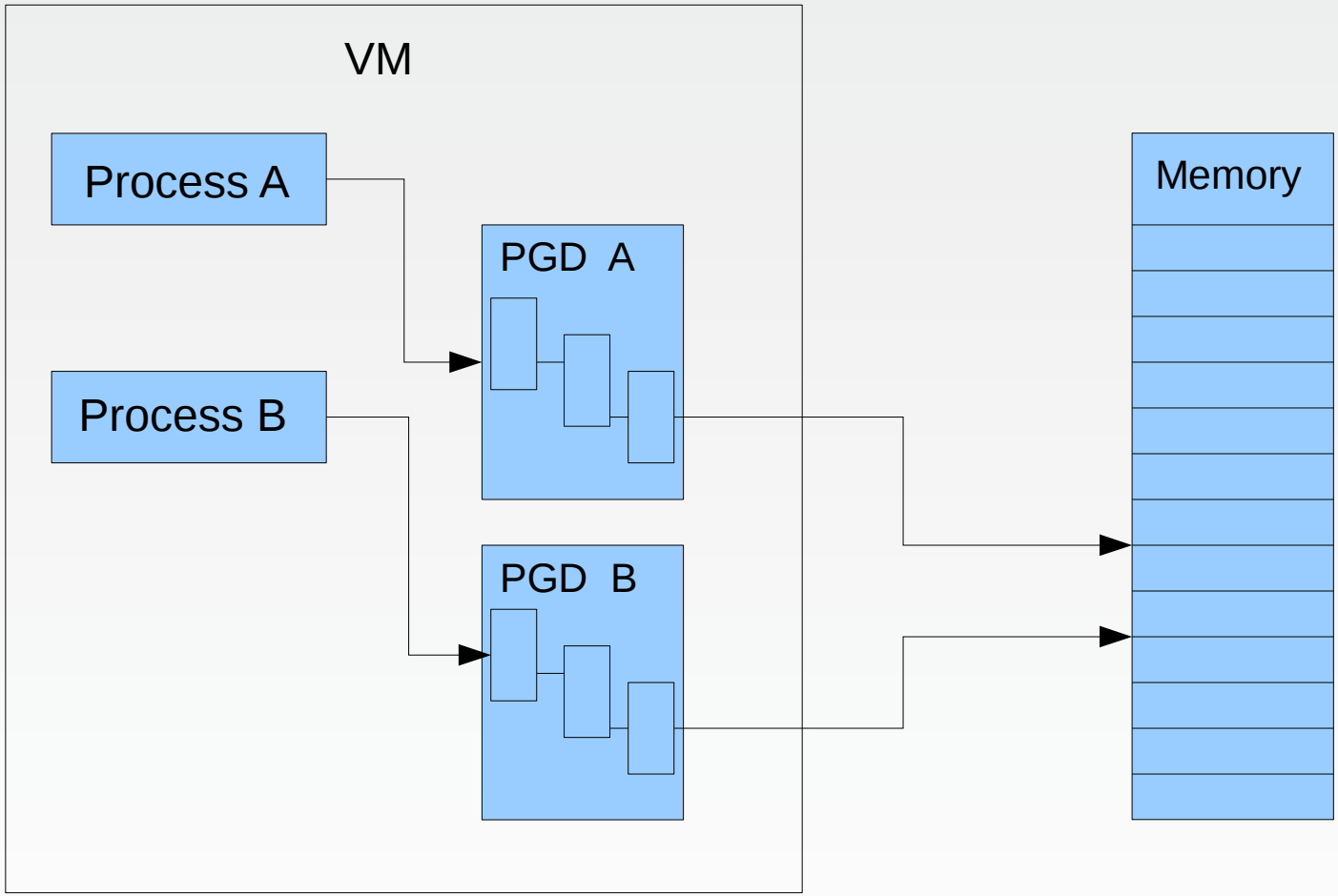
Intro: Virt. extensions: CPU



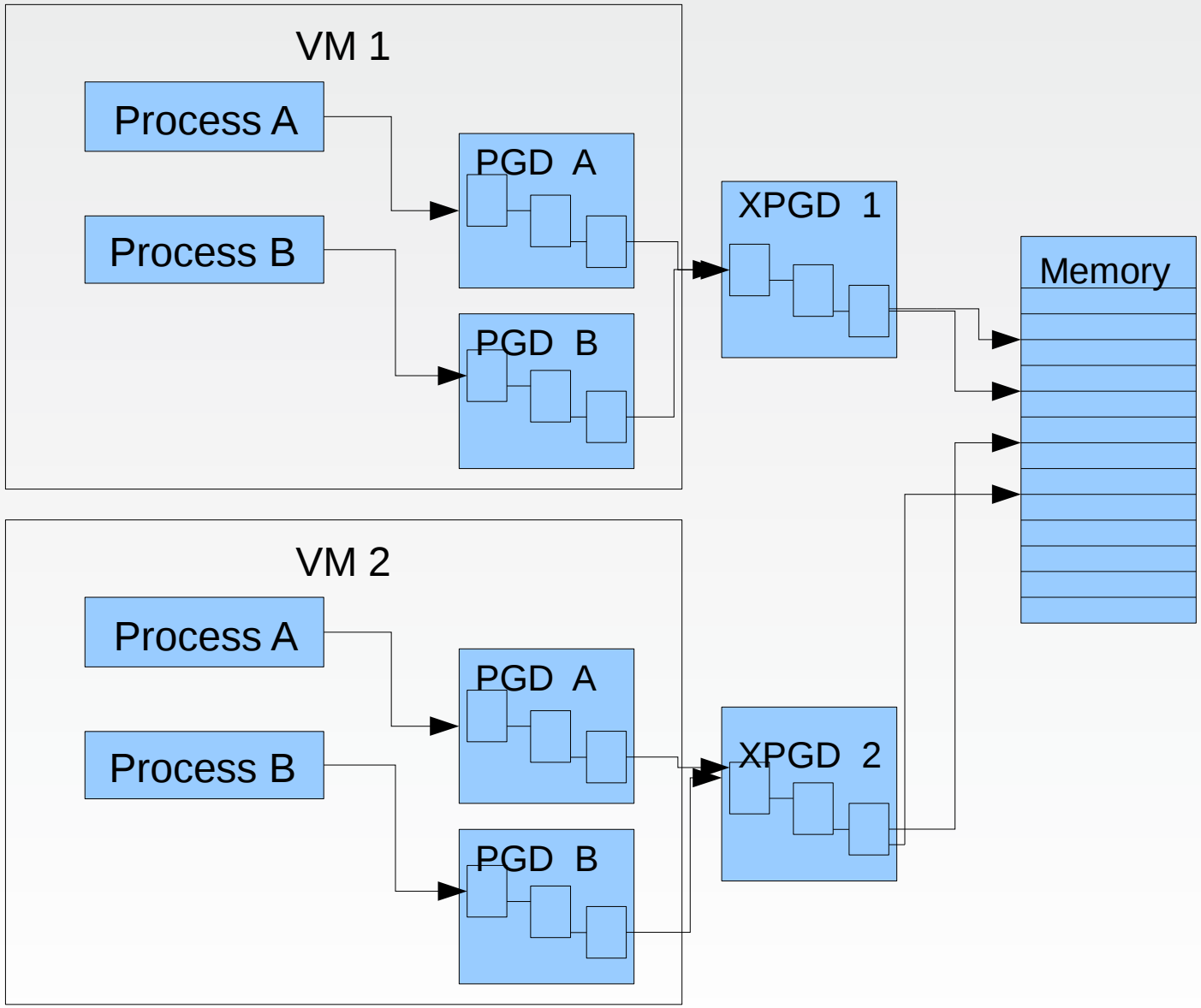
Intro: Virt. extensions: memory



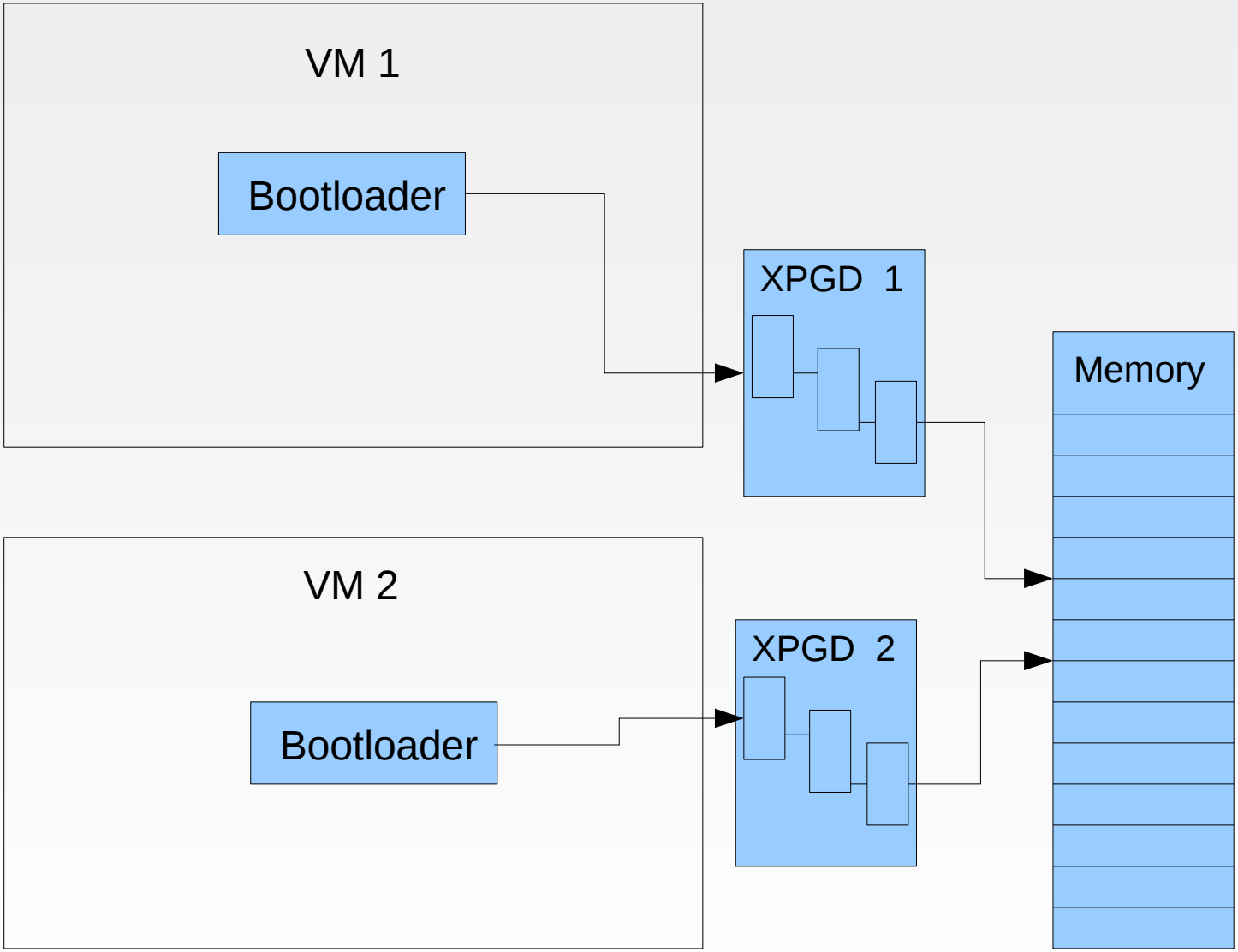
Intro: Virt. extensions: memory



Intro: Virt. extensions: memory



Intro: Virt. extensions: memory



Scope of comparison

- What it's not a
  - Power comparison
  - Perf comparison
  - Perf*Power comparison
- It's not even a
  - Functionality comparison
    - We've already seen there's parity for core extensions

Scope of comparison cont.

- Developer consumption
  - How easy are the virt. extensions to use?
- Limited to
  - CPU and virtual memory
  - KVM (kernel space)
- Limited code paths
  - Traced exits while booting a simple guest
  - Then looked at the relevant KVM handlers

MMU enable

```
kvm_exit: reason EPT_VIOLATION rip 0x40015e info 184 0
kvm_exit: reason CR_ACCESS rip 0x400196 info 4 0
kvm_exit: reason CR_ACCESS rip 0x40019e info 3 0
kvm_exit: reason MSR_READ rip 0x4001a6 info 0 0
kvm_exit: reason MSR_WRITE rip 0x4001ac info 0 0
kvm_exit: reason CR_ACCESS rip 0x4001b9 info 0 0
kvm_exit: reason EPT_VIOLATION rip 0x4002b5 info 181 0
```

```
kvm_guest_fault:  ipa 0x40080000, hsr 0x82000007, hxfar 0x40080000, pc 0x40080000
kvm_guest_fault:  ipa 0x400af000, hsr 0x92000047, hxfar 0x400affe0, pc 0x4008001c
kvm_guest_fault:  ipa 0x44000000, hsr 0x93830007, hxfar 0x44000004, pc 0x400812fc
kvm_guest_fault:  ipa 0x400b0000, hsr 0x93040047, hxfar 0x400b0000, pc 0x4008263c
kvm_guest_fault:  ipa 0x40093000, hsr 0x93d48047, hxfar 0x400935c0, pc 0x400832e4
kvm_guest_fault:  ipa 0x400c0000, hsr 0x92000047, hxfar 0x400c0000, pc 0x4008259c
kvm_guest_fault:  ipa 0x400d0000, hsr 0x92000047, hxfar 0x400d0000, pc 0x4008259c
kvm_toggle_cache: VM op at 0x00000000400800a0 (cache was off, now off)
kvm_toggle_cache: VM op at 0x00000000400800a8 (cache was off, now off)
kvm_toggle_cache: VM op at 0x00000000400800ac (cache was off, now off)
kvm_toggle_cache: VM op at 0x00000000400800c4 (cache was off, now on)
```

VMX handlers

```
handle_ept_violation()
{
    exit_qualification = vmcs_read(EXIT_QUALIFICATION)
    gpa = vmcs_read(GUEST_PHYSICAL_ADDRESS)
    is_mmio = handle_page_fault(gpa, fault_type(exit_qualification))
    if (is_mmio)
        emul_mmio(gpa)
}

handle_cr()
{
    exit_qualification = vmcs_read(EXIT_QUALIFICATION)
    (cr, reg) = decode(exit_qualification)
    switch (cr) {
    case 0: return init_kvm_mmu(reg)
    case 3: return switch_pgdir(reg)
    case 4: return set_cpu_feature(reg)
    }
}

handle_rdmsr(msr_index)
{
    if (in_vmcs(msr_index))
        return vmcs_read(msr_index)
    return emul_rdmsr(msr_index)
}
```

VMCS

- Guest State
- Host State
- Trap controls
- VMExit controls
- VMEntry controls
- VMExit information

Instruction emulation

- Much simpler with ARM than with x86
  - arch/x86/kvm/emulate.c 10x larger than arch/arm64/kvm/emulate.c + arch/arm/kvm/emulate.c
- But, ldm/stm & ldp/stp are a challenge nobody has taken on

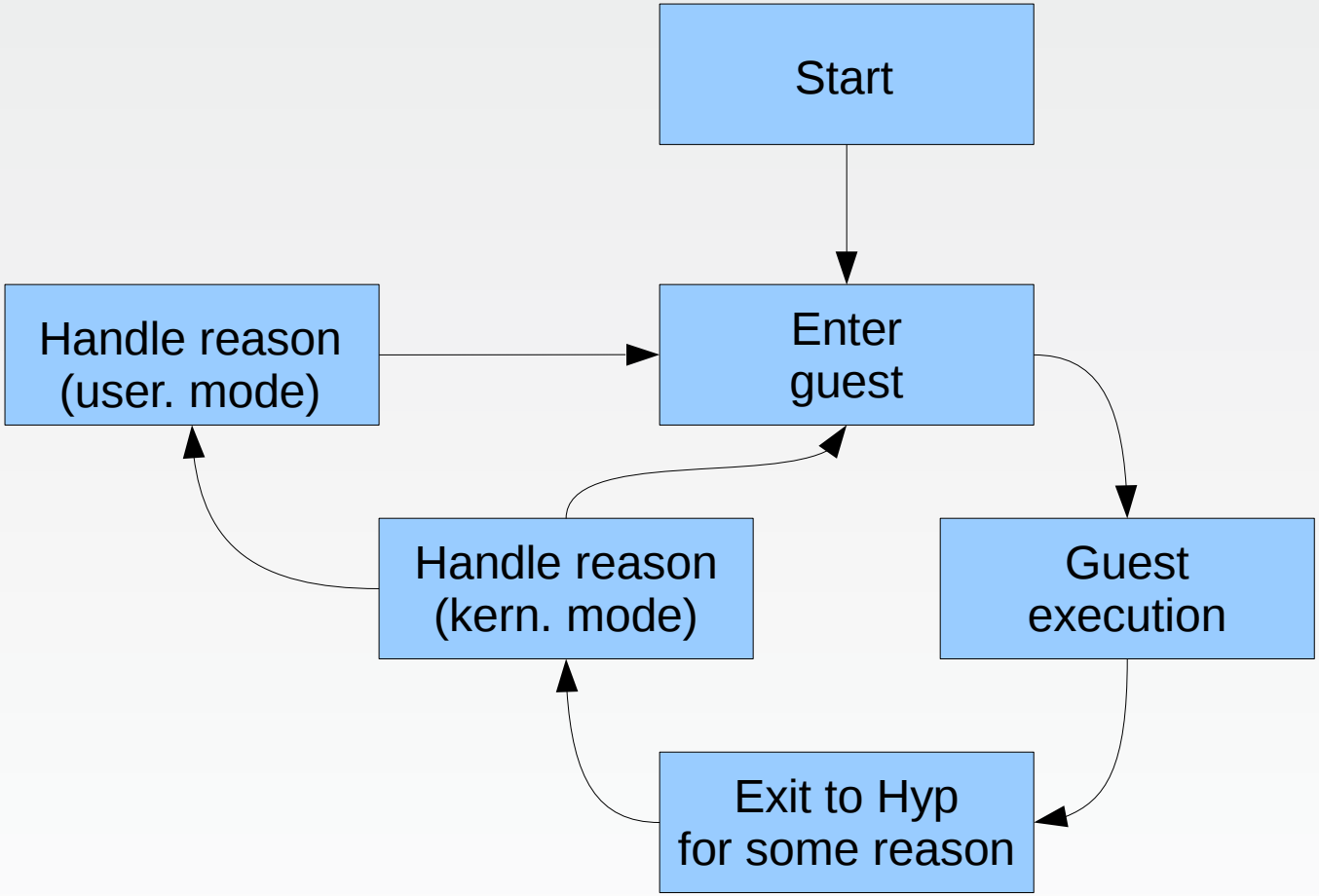
ARM handlers

```
handle_guest_fault(vcpu)
{
    type = fault_type(vcpu)
    gpa = fault_ipa(vcpu)

    if (mmio_address(gpa))
        return emul_mmio(gpa)

    if (!cache_enabled(vcpu) || uncached(gpa))
        flush_dcache(gpa)
    stage2_table_update(gpa)
}

access_vm_reg(vcpu)
{
    was_enabled = cache_enabled(vcpu)

    update_vm_reg(vcpu)

    if (cache_enabled(vcpu) != was_enabled)
        stage2_flush_vm(vcpu->kvm)
    if (cache_enabled(vcpu))
        vcpu_hcr_clear(vcpu, HCR_TVM)
}
```

ARM v8 stage1&2 cache behavior

```
MMU_CombineS1S2Noncached(s2type, s1type)
{
    if (s2type == NONCACHED || s1type == NONCACHED)
        return NONCACHED
}
```

Handlers: not so fast...

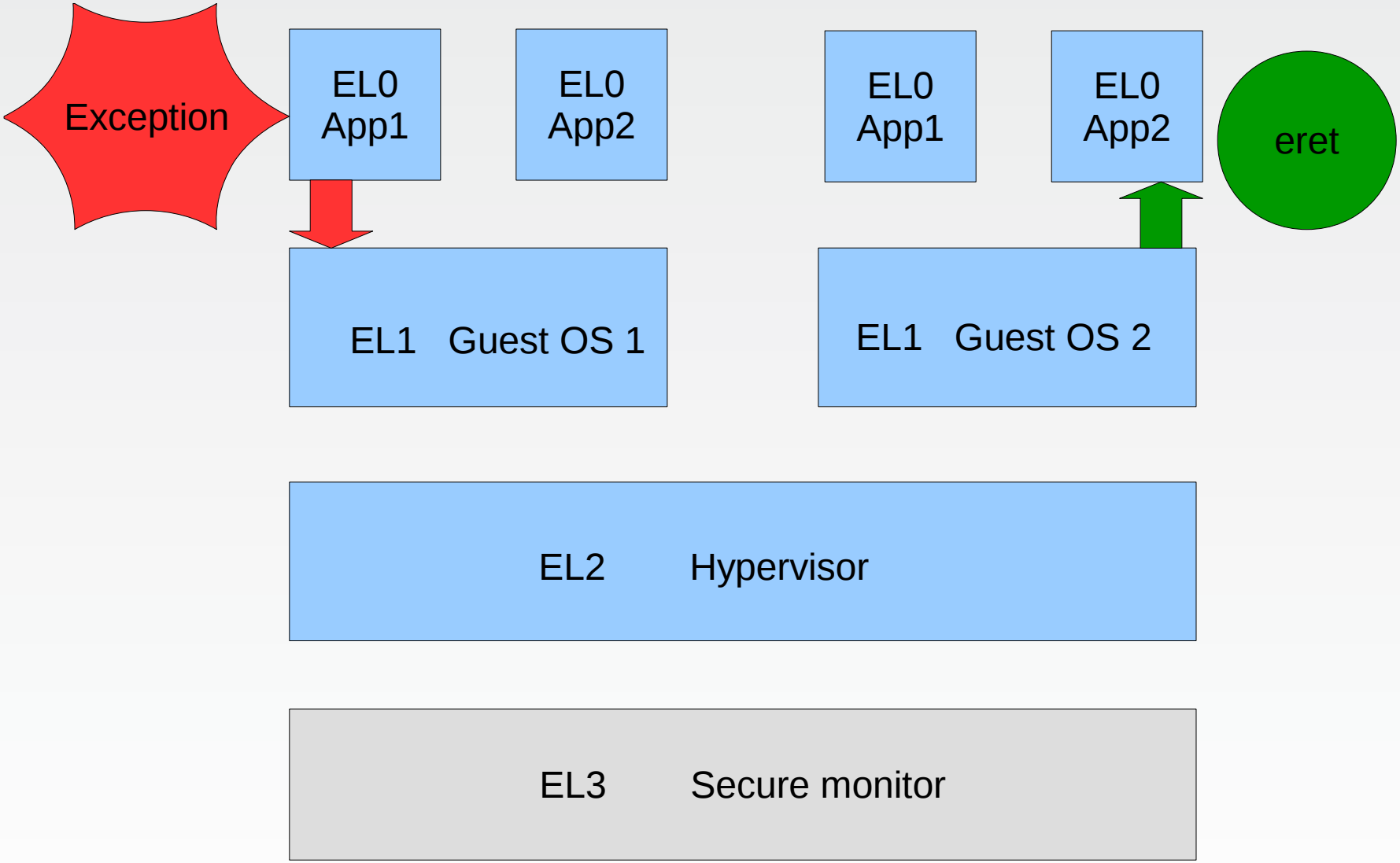


Handlers: not so fast...

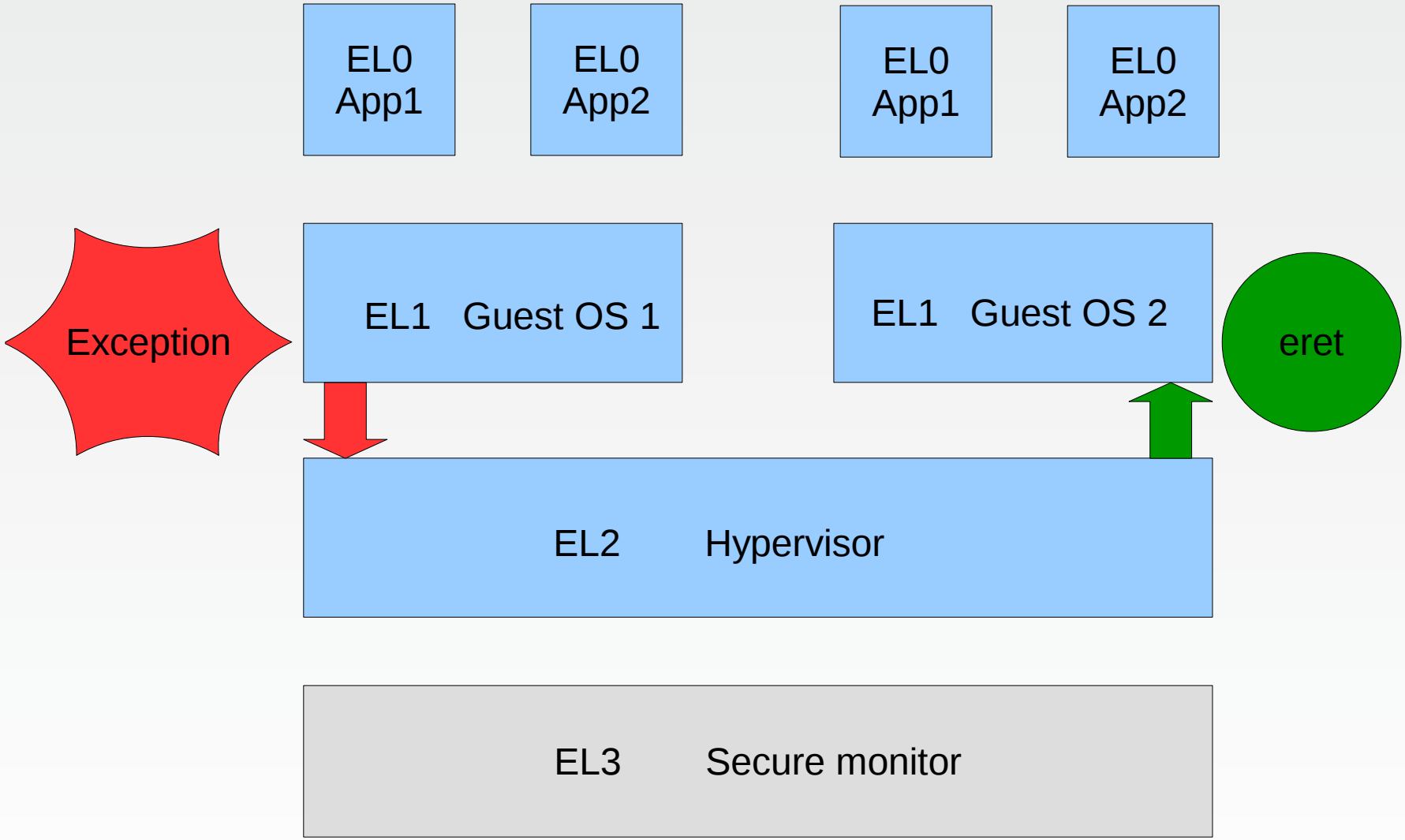
```
vcpu_enter_guest()
{
    vcpu_update()
    save_host_state()
    asm
    {
        save host registers
        restore guest registers
        VMLAUNCH/VMRESUME
        save guest registers
        restore host registers
    }
    handle_exit()
}
```

```
vcpu_enter_guest()
{
    vcpu_update()
    save_host_state()
    call_hyp(vcpu_run)
    handle_exit()
}

vcpu_run:
    save host registers
    restore guest registers
    eret
vcpu_return:
    save guest registers
    restore host registers
    ret

call_hyp:
    hvc
    ret
```

ARMv8 Exception levels



ARMv8 Exception levels



Handlers: not so fast... continued...

```
vcpu_enter_guest()
{
    vcpu_update()
    save_host_state()
    call_hyp(vcpu_run)
    handle_exit()
}

vcpu_run:
    save host registers
    restore guest registers
    eret
vcpu_return:
    save guest registers
    restore host registers
    ret

call_hyp:
    hvc
    ret
```

```
el2_vector_handler:
    determine if guest exited
    cbnz    guest_exited,el1_trap
    blr     x0
    eret

el1_trap:
    save vcpu exit state
    b       vcpu_return
```

Guest execution

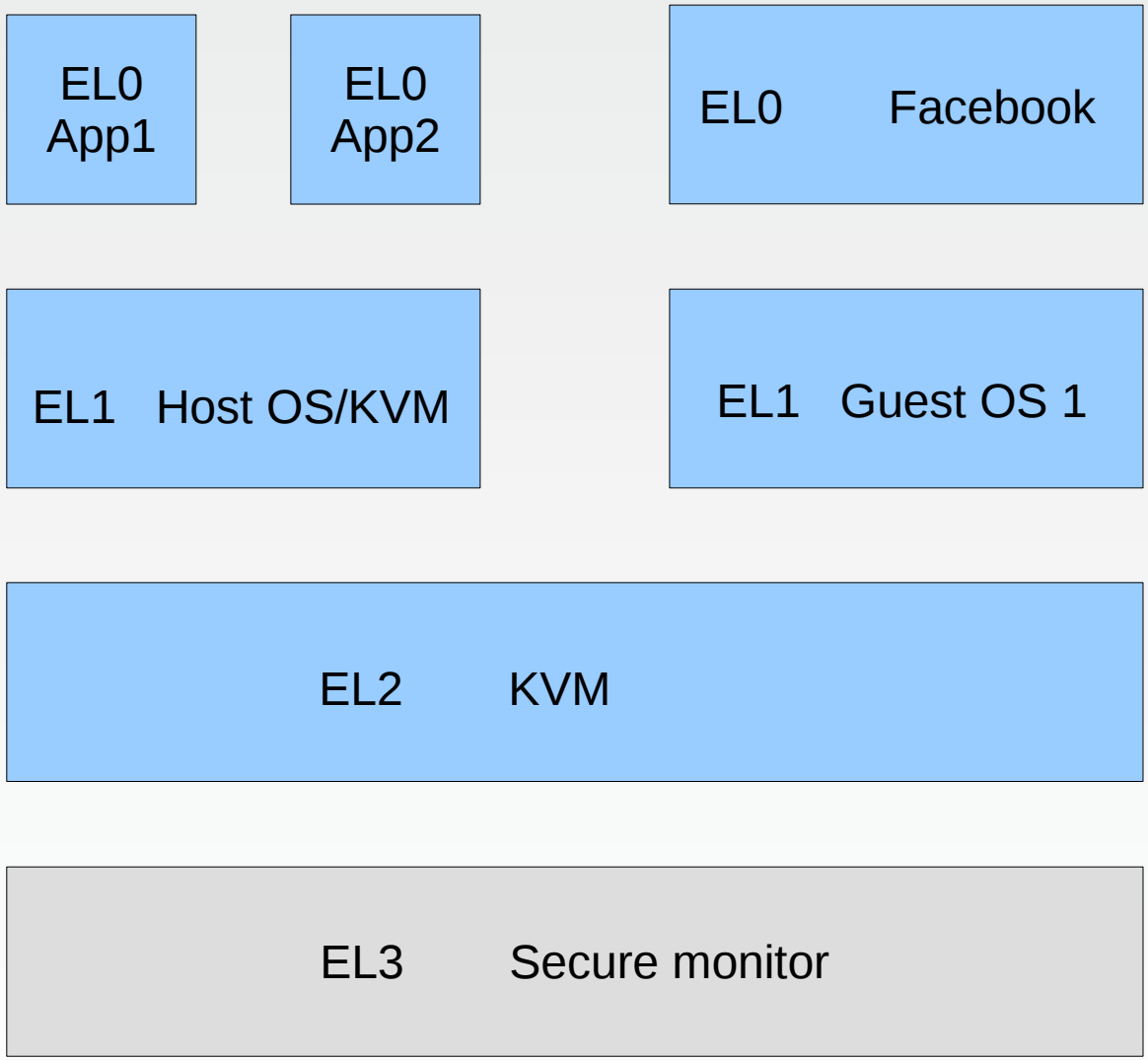
Handlers: not so fast... continued...

```
vcpu_enter_guest()
{
    vcpu_update()
    save_host_state()
    call_hyp(vcpu_run)
    handle_exit()
}

vcpu_run:
    save host registers
    restore guest registers
    eret
vcpu_return:
    save guest registers
    restore host registers
    ret

call_hyp:
    hvc
    ret
```

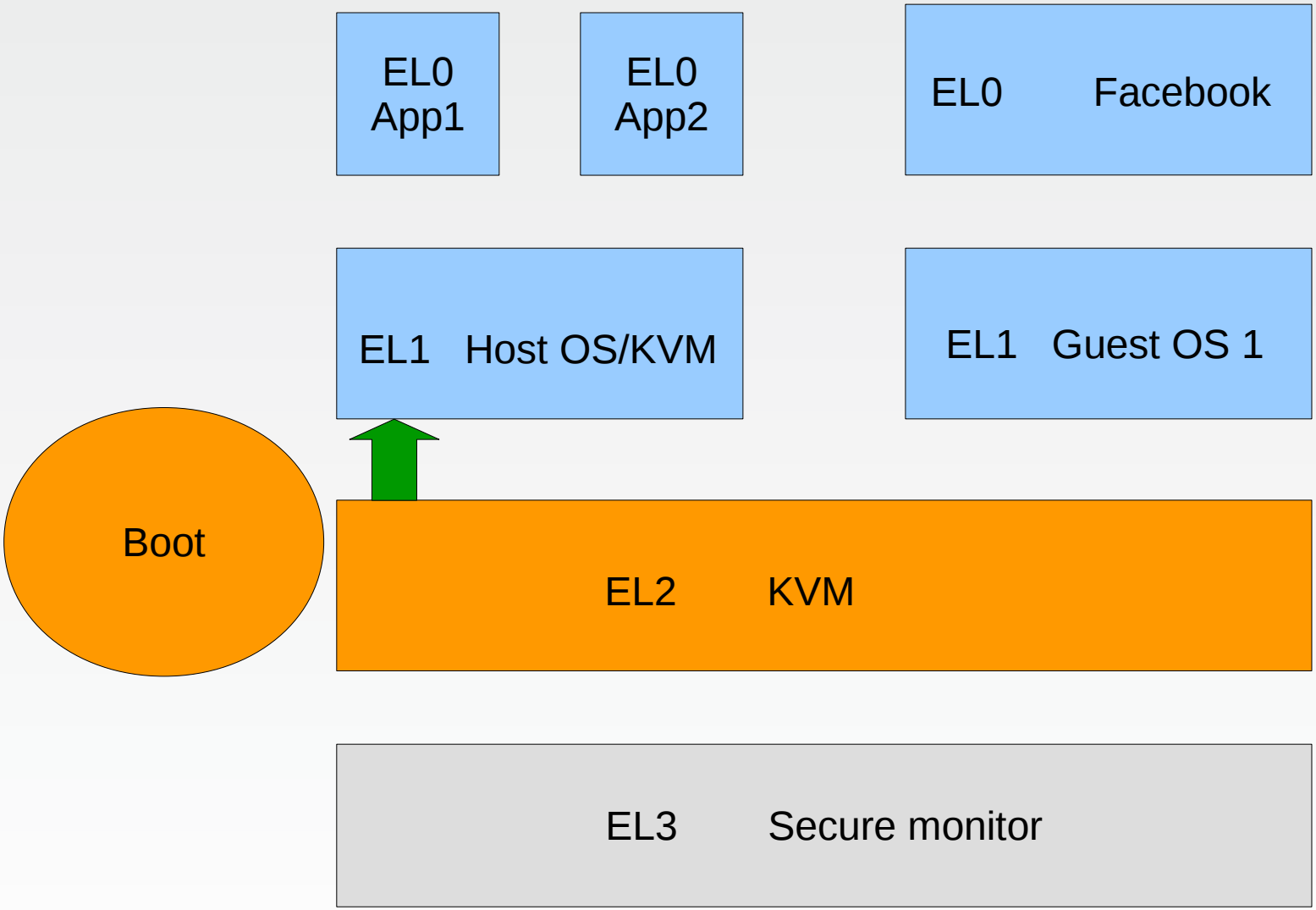
```
el2_vector_handler:
    determine if guest exited
    cbnz    guest_exited,el1_trap
    blr     x0
    eret

el1_trap:
    save vcpu exit state
    b       vcpu_return
```

Guest execution

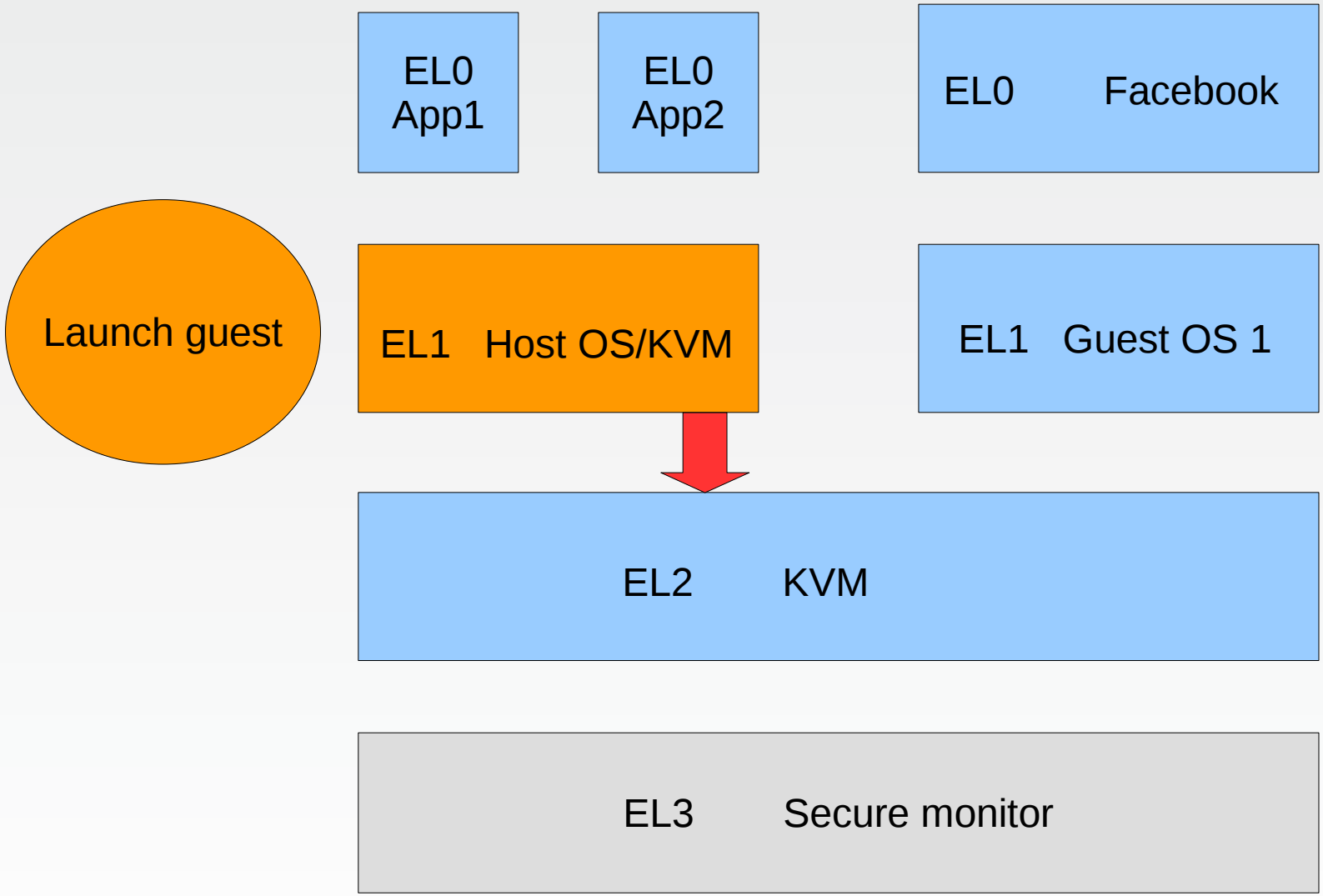
ARMv8 KVM split



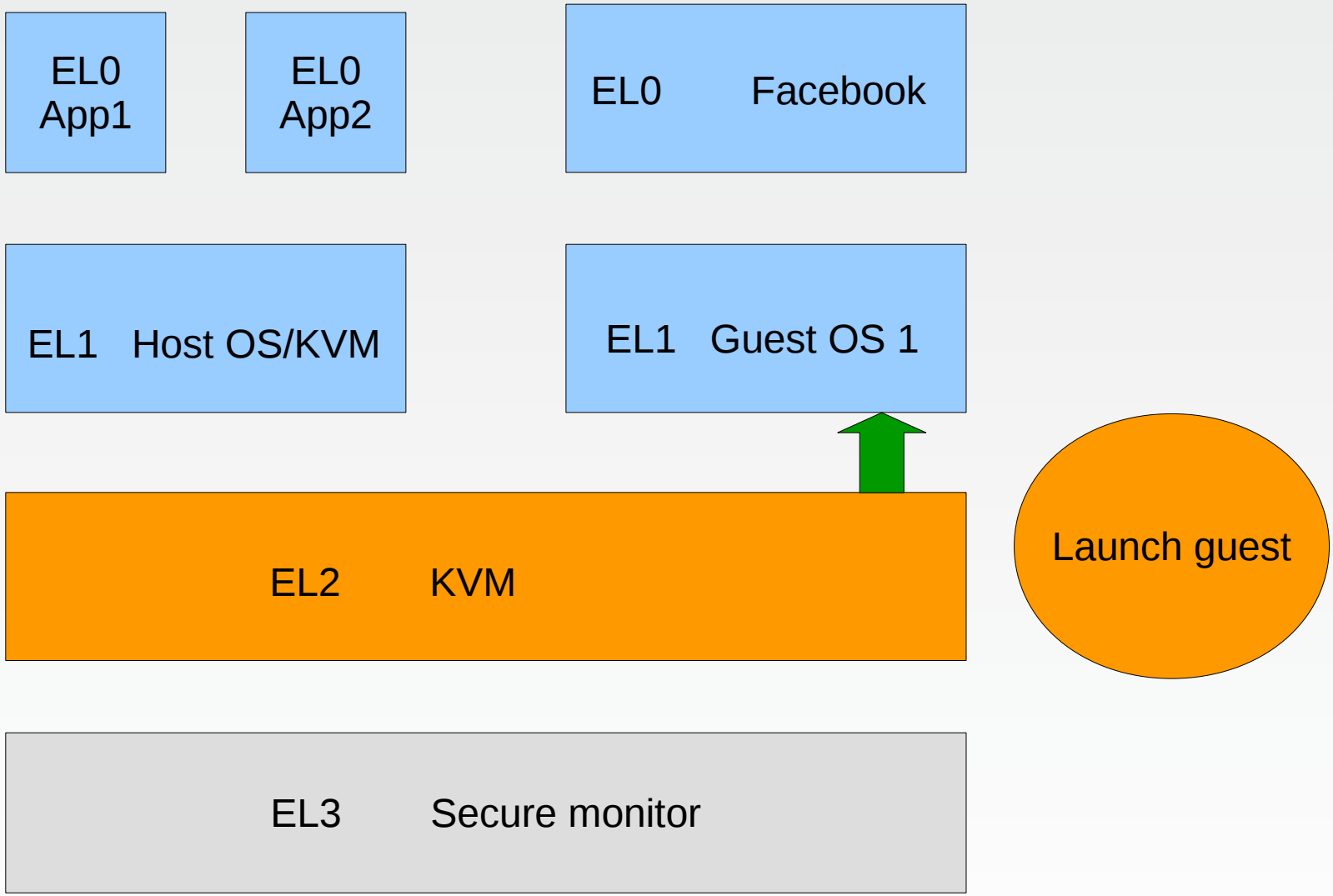
ARMv8 KVM split



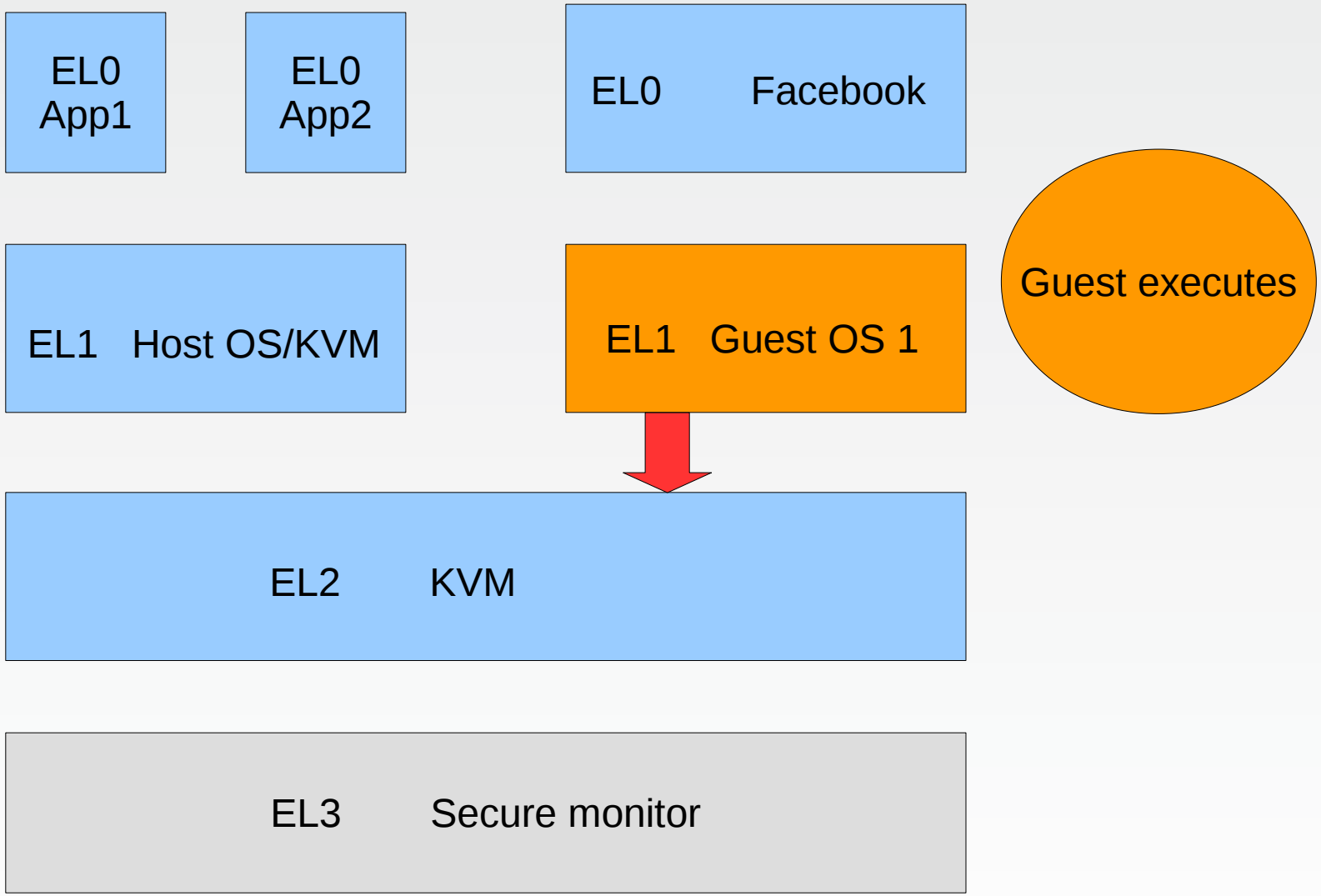
ARMv8 KVM split



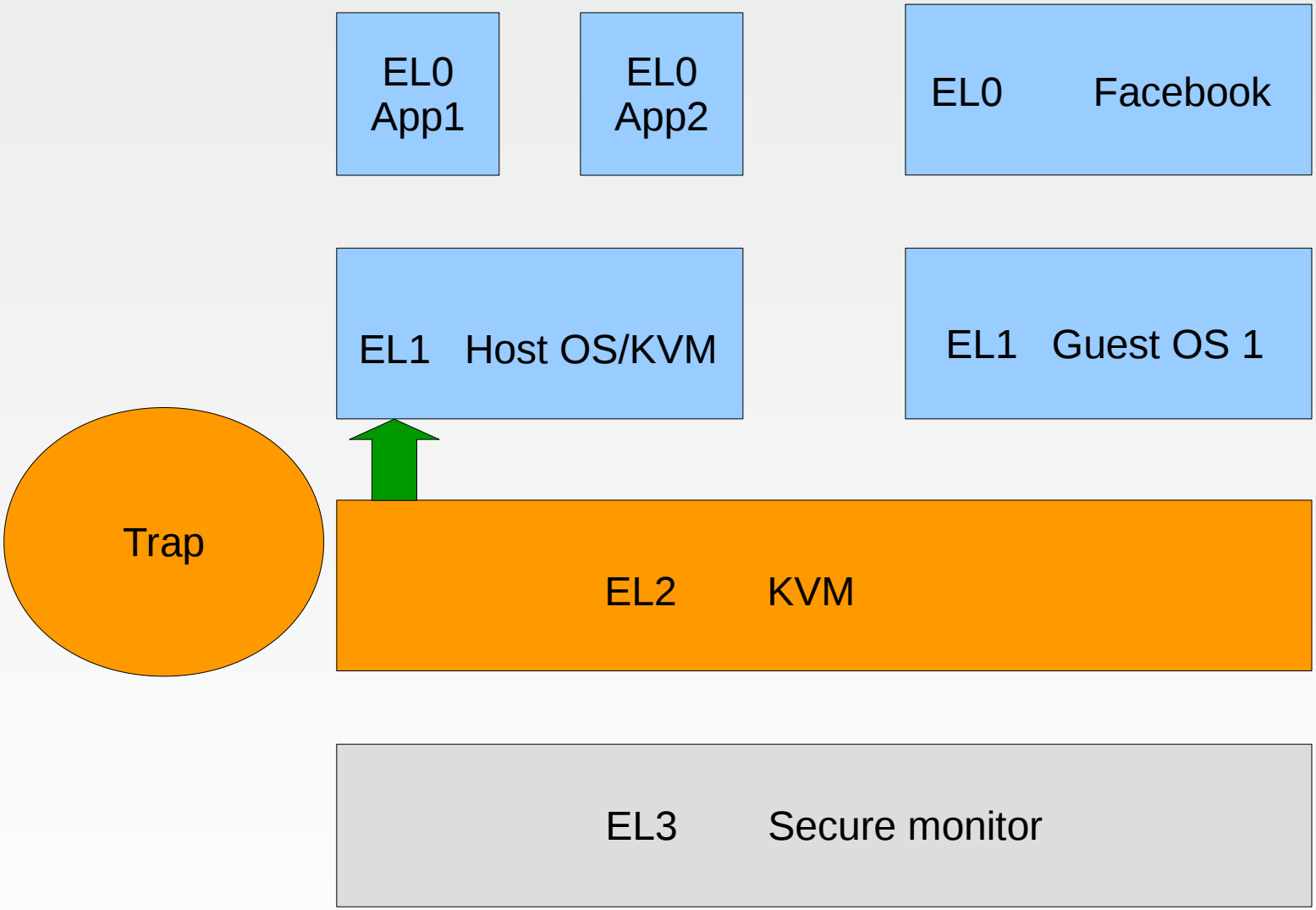
ARMv8 KVM split



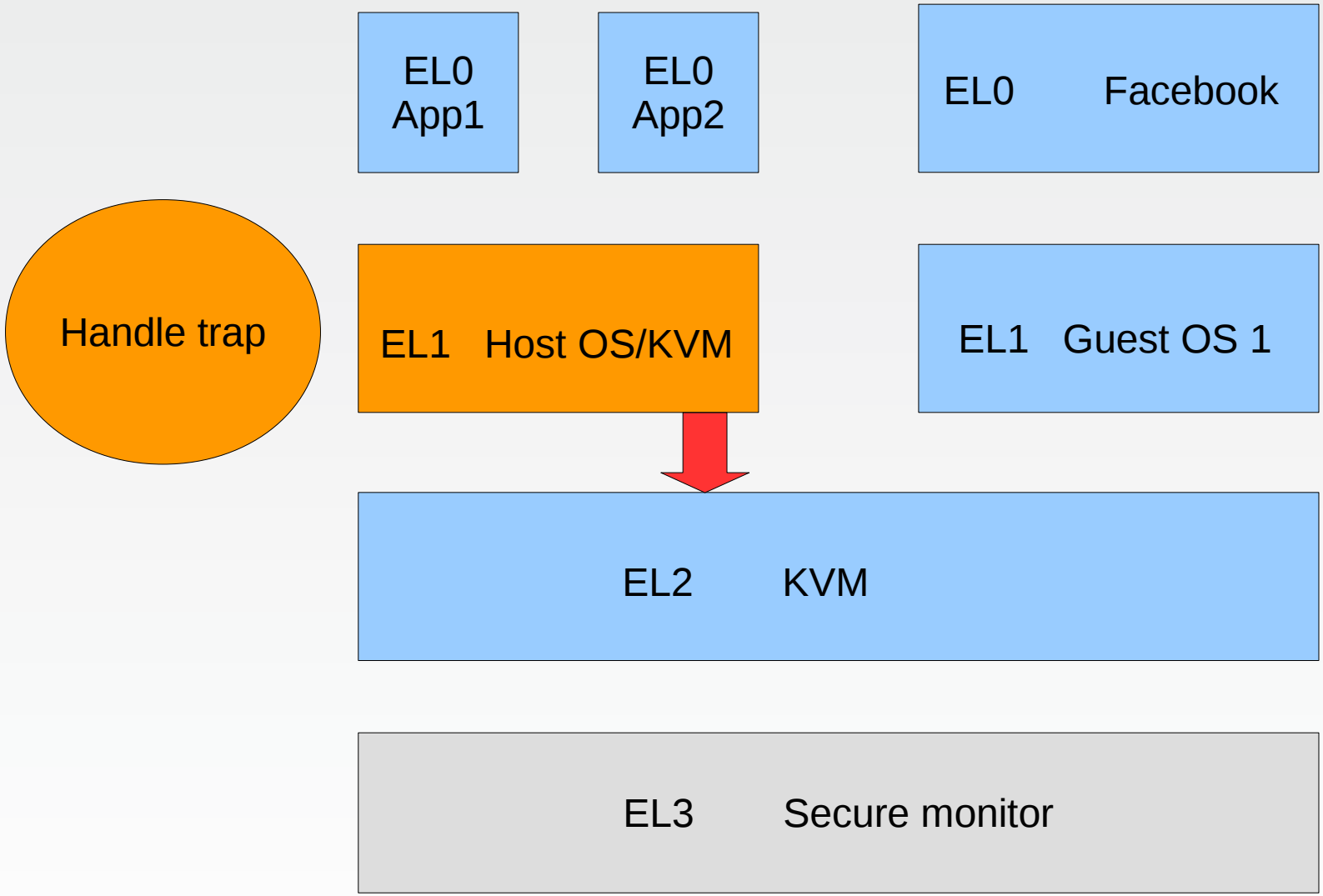
ARMv8 KVM split



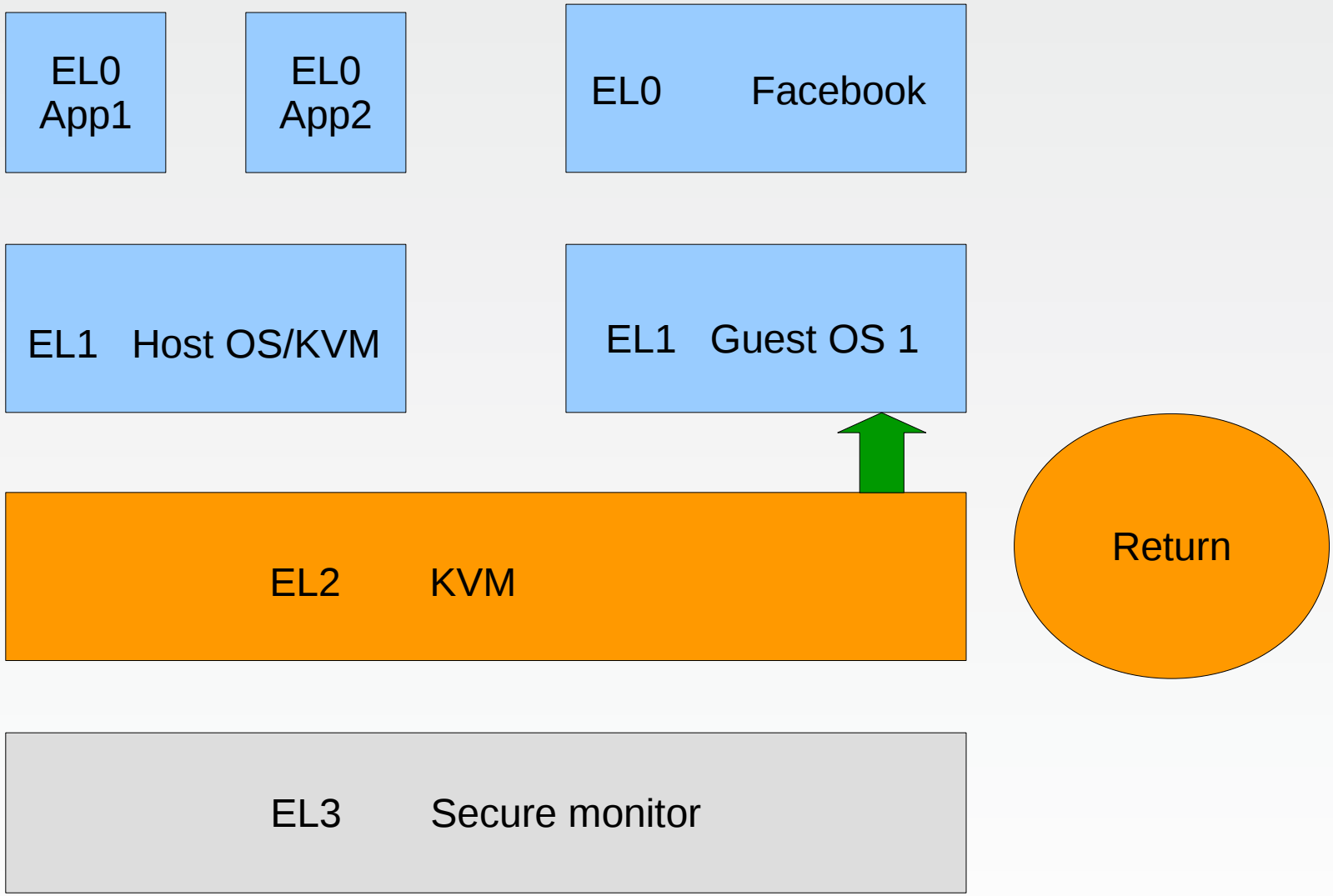
ARMv8 KVM split



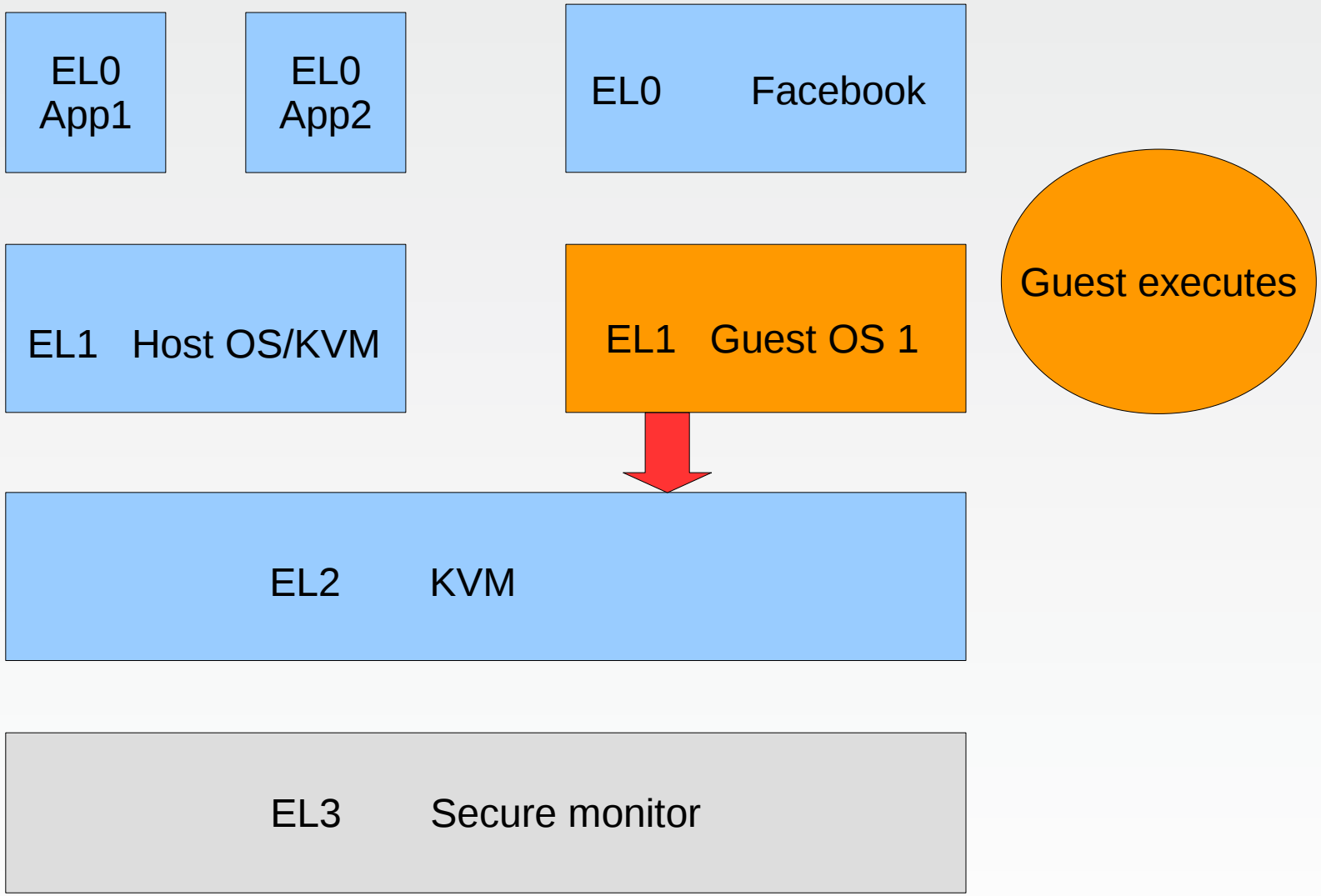
ARMv8 KVM split



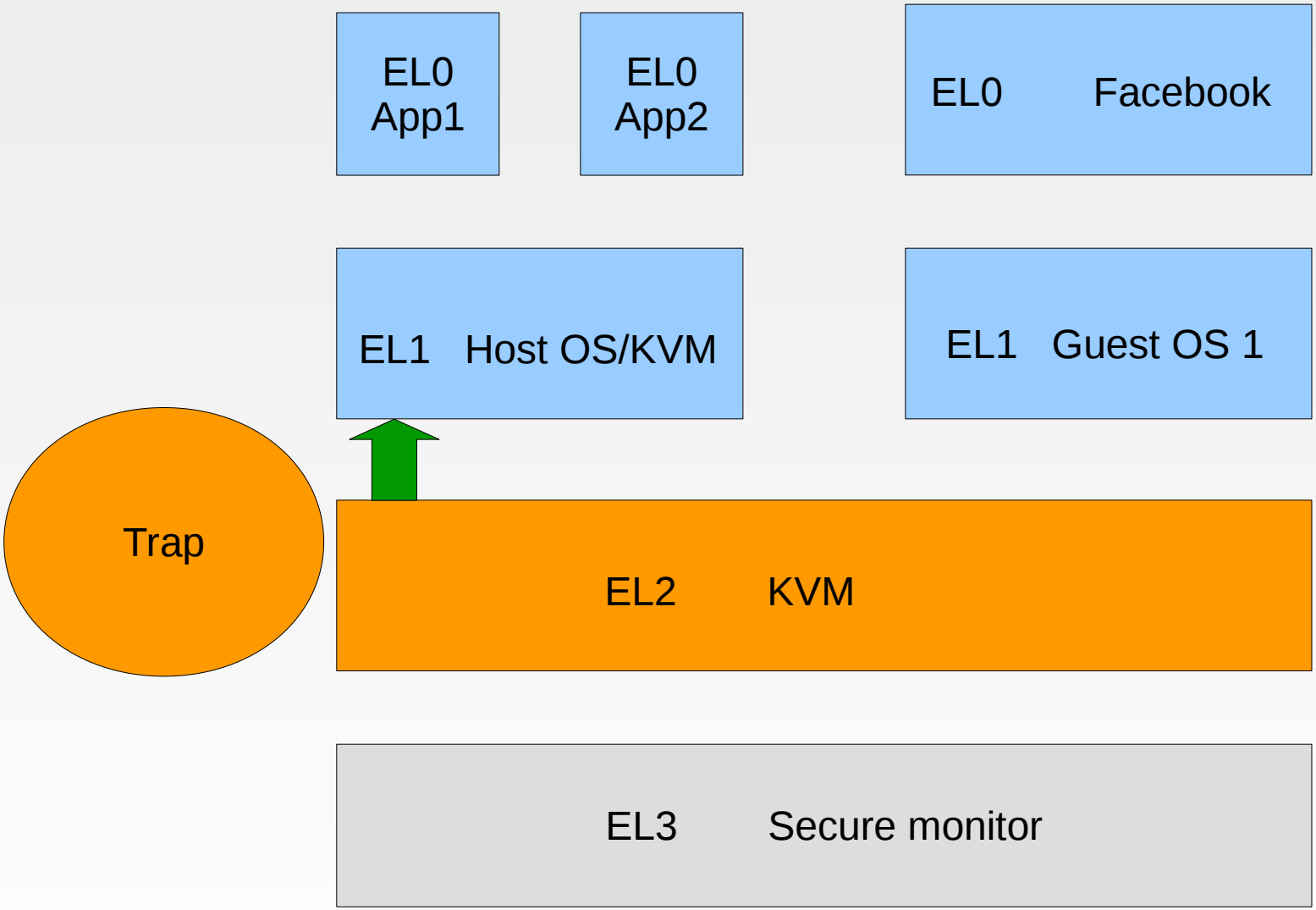
ARMv8 KVM split



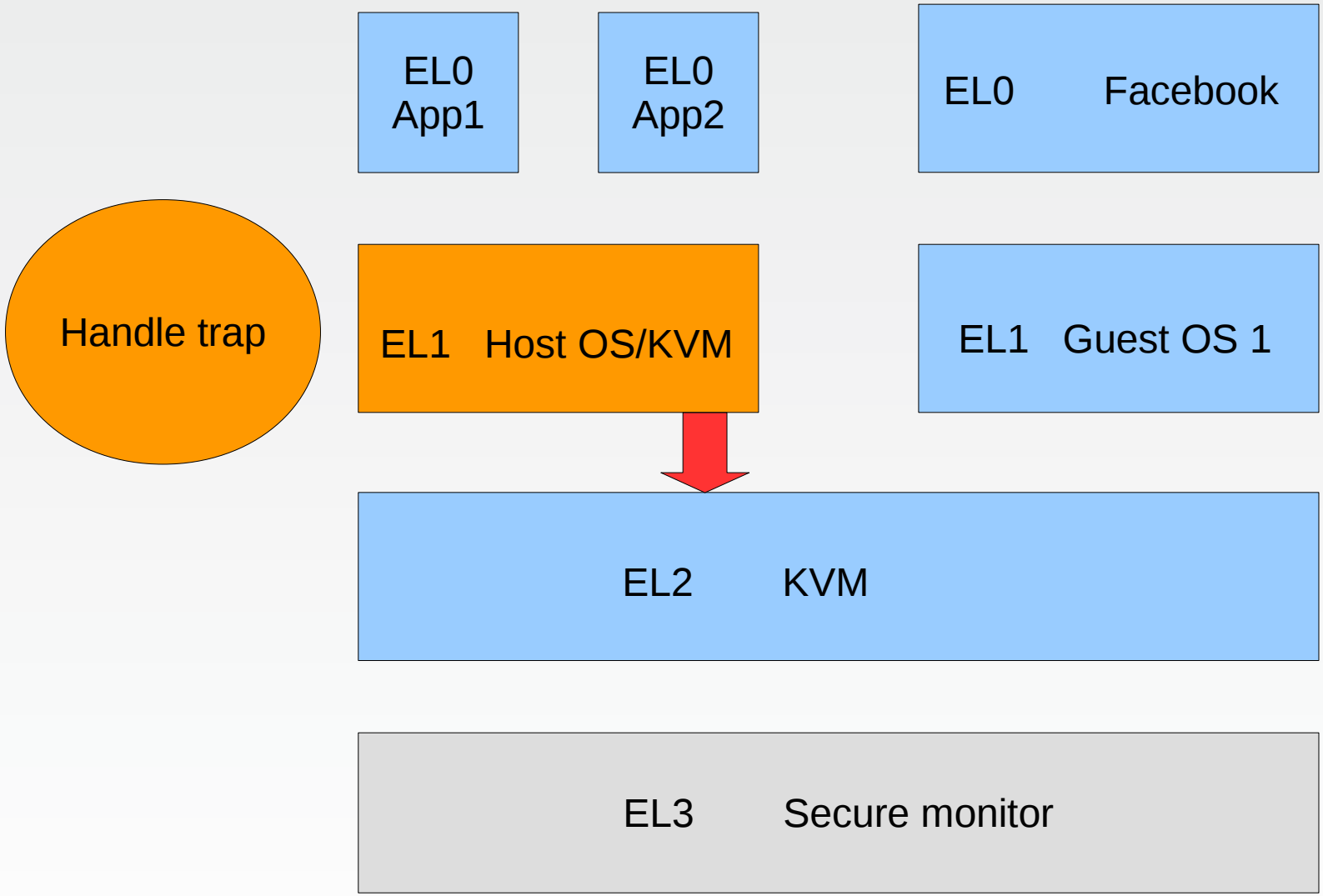
ARMv8 KVM split



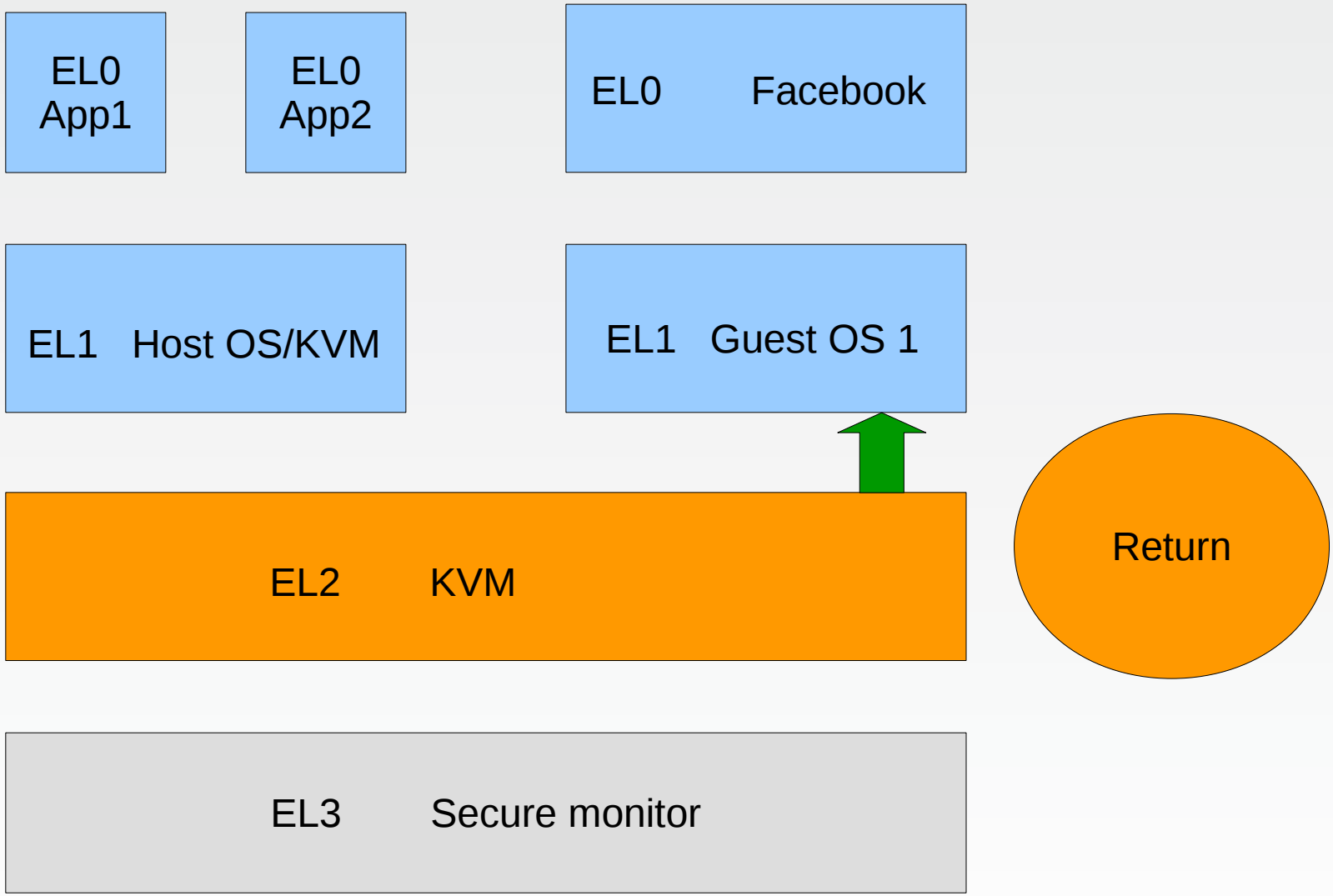
ARMv8 KVM split



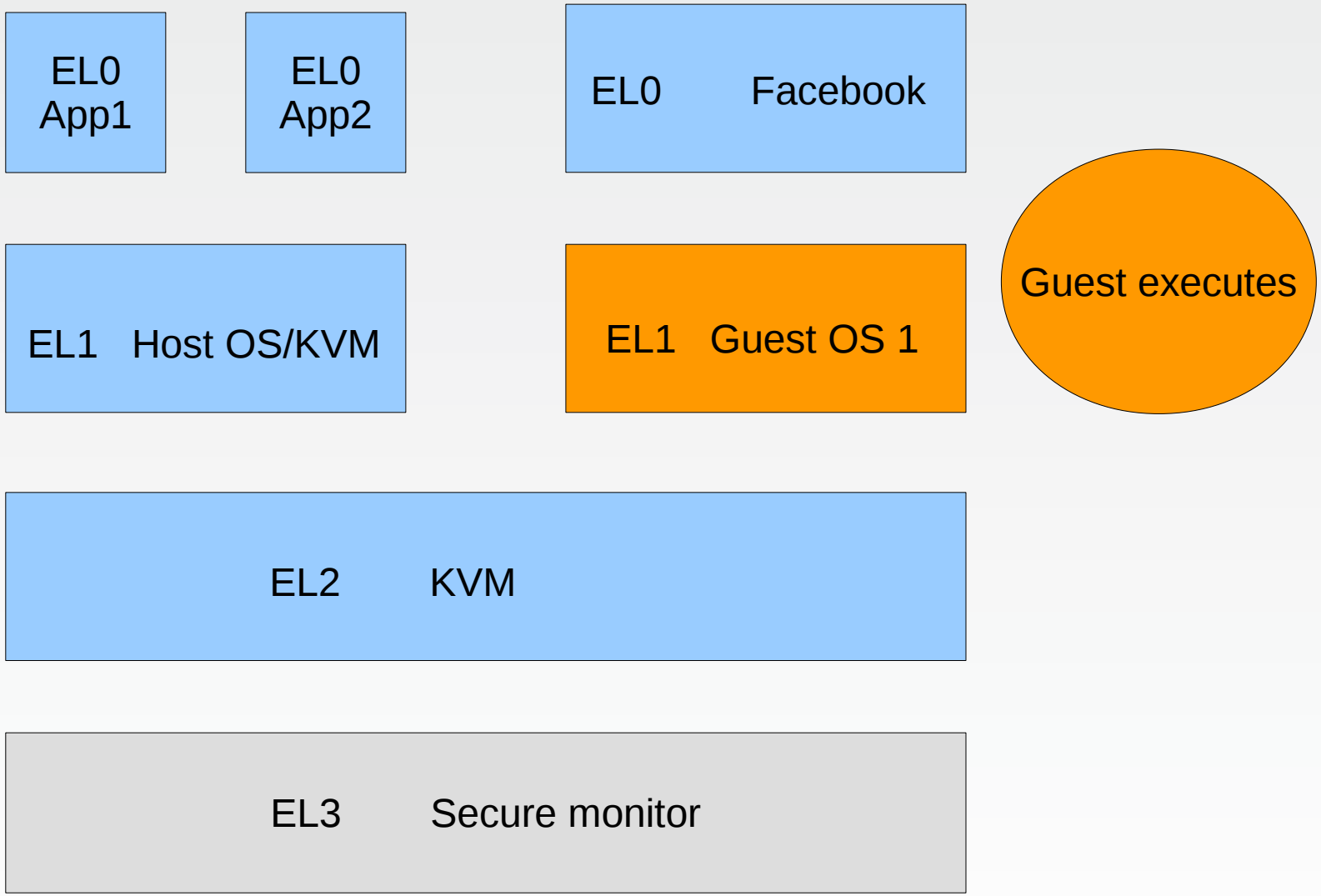
ARMv8 KVM split



ARMv8 KVM split



ARMv8 KVM split



Conclusion

- Intel has been evolving its virt. extensions for a decade
- ARM's initial release has feature parity
- Handling vmexits like normal exceptions is nice, but has some pitfalls
  - Not sure what it implies for nested-virt
- Managing the cache with stage1 precedence is a pain
- Looking forward to ARM v8.1 (VHE) for KVM!

Q/A

Thanks!
Questions?

Yup, this is the last slide.

- Feeback: http://devconf.cz/f/47
- Contact: drew – drjones@redhat.com