Western Digital®

Xvisor: Embedded Hypervisor for RISC-V

Anup Patel <anup.patel@wdc.com>

Open Source Summit Europe 2019

Roadmap

- RISC-V H-Extension
- Xvisor Overview
- Xvisor RISC-V
- Status & Future Work
- Xvisor RISC-V Demo
- Questions

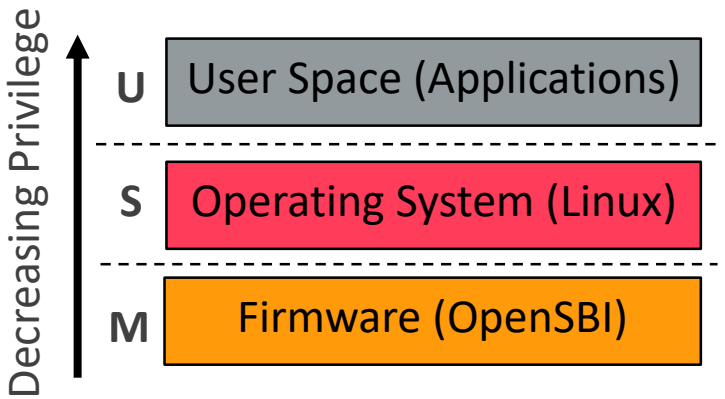
RISC-V H-Extension

The RISC-V Hypervisor Extension

What is RISC-V ?

Free and Open Instruction Set Architecture (ISA)

- **Clean-slate and Extensible ISA**
- **XLEN (machine word length)** can be 32 (RV32), 64 (RV64), and 128 (RV128)
- **32 general purpose registers**
- **Variable instruction length** (instruction compression)
- **Three privilege modes:** Machine (M-mode), Supervisor (S-mode), and User (U-mode)
- **Control and Status Registers (CSR)** for each privilege mode



General Purpose Registers	
zero	Hardwired-zero register
ra	Return address register
sp	Stack pointer register
gp	Global pointer register
tp	Thread pointer register
a0-a7	Function argument registers
t0-t6	Caller saved registers
s0-s11	Callee saved registers

S-mode CSRs (Used By Linux)	
sstatus	Status
sie	Interrupt Enable
sip	Interrupt Pending
stvec	Trap Handler Base
sepc	Trap Progam Counter
scause	Trap Cause
stval	Trap Value
satp	Address Translation
sscratch	Scratch
NOTE: **sedeleg**, **sideleg**, and **scounteren** not used currently	

RISC-V H-Extension: Spec Status

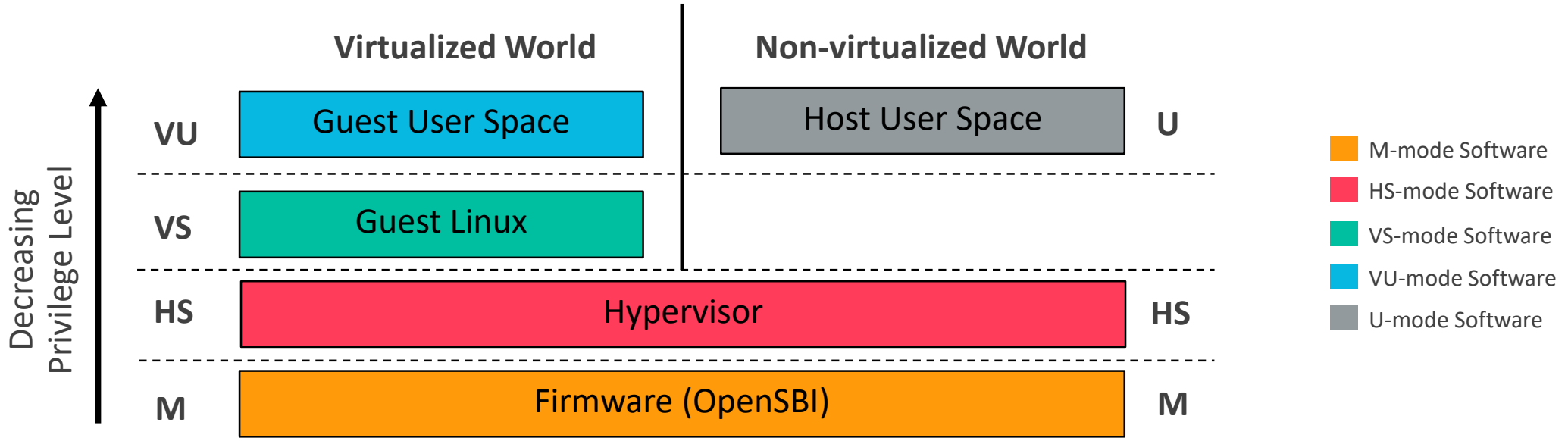
H-Extension spec close to freeze state

- Designed to suit both Type-1 (Baremetal) and Type-2 (Hosted) hypervisor
- v0.4-draft was released on 16th June 2019
- v0.5-draft will be released soon
- WDC's initial QEMU, Xvisor and KVM ports were based on v0.3
- They have all been updated to the new v0.4 spec
  - There were limited software changes required between v0.3 and v0.4
    - QEMU required more changes

RISC-V H-Extension: Privilege Mode Changes

New execution modes for guest execution

- HS-mode = S-mode with hypervisor capabilities and new CSRs
- Two additional modes:
  - VS-mode = Virtualized S-mode
  - VU-mode = Virtualized U-mode



RISC-V H-Extension: CSR changes

More control registers for virtualising S-mode

- In HS-mode (V=0)
  - “s<xyz>” CSRs point to standard “s<xyz>” CSRs
  - “h<xyz>” CSRs for hypervisor capabilities
  - “vs<xyz>” CSRs contains VS-mode state
- In VS-mode (V=1)
  - “s<xyz>” CSRs point to virtual “vs<xyz>” CSRs

HS-mode CSRs for hypervisor capabilities	
hstatus	Hypervisor Status
hideleg	Hypervisor Interrupt Delegate
hedeleg	Hypervisor Trap/Exception Delegate
htimedelta	Hypervisor Guest Time Delta
hgatp	Hypervisor Guest Address Translation

HS-mode CSRs for accessing Guest/VM state	
vsstatus	Guest/VM Status
vsie	Guest/VM Interrupt Enable
vsip	Guest/VM Interrupt Pending
vstvec	Guest/VM Trap Handler Base
vsepc	Guest/VM Trap Progam Counter
vscause	Guest/VM Trap Cause
vstval	Guest/VM Trap Value
vsatp	Guest/VM Address Translation
vsscratch	Guest/VM Scratch

RISC-V H-Extension: Two-stage MMU

Hardware optimized guest memory management

- Two-Stage MMU for VS/VU-mode:
  - **VS-mode page table (Stage1):**
    - Translates Guest Virtual Address (GVA) to Guest Physical Address (GPA)
    - Programmed by Guest (same as before)
  - **HS-mode guest page table (Stage2):**
    - Translates Guest Physical Address (GPA) to Host Physical Address (HPA)
    - Programmed by Hypervisor
- In HS-mode, software can program two page tables:
  - **HS-mode page table:** Translate hypervisor Virtual Address (VA) to Host Physical Address (HPA)
  - **HS-mode guest page table:** Translate Guest Physical Address (GPA) to Host Physical Address (HPA)
- Format of VS-mode page table, HS-mode guest page table and HS-mode host page table is same (Sv32, Sv39, Sv48, ….)

RISC-V H-Extension: I/O & Interrupts

I/O and guest interrupts virtualization

- Virtual interrupts injected by updating VSIP CSR from HS-mode
- Software and Timer Interrupts:
  - Hypervisor will emulate SBI calls for Guest
- HS-mode guest page table can be used to trap-n-emulate MMIO accesses for:
  - Software emulated PLIC
  - VirtIO devices
  - Other software emulated peripherals

Xvisor Overview

The Extensible Versatile Hypervisor

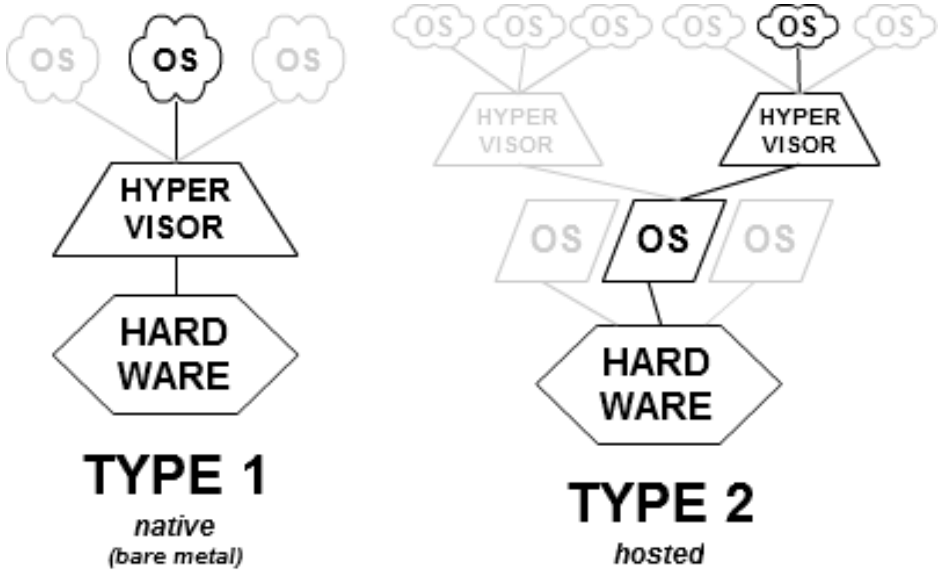
What is Xvisor ?

The Extensible Versatile Hypervisor

- **XVISOR** = e**X**tensible **V**ersatile hyperv**ISOR**
- Xvisor is an open-source GPLv2 Type-1 monolithic (i.e. Pure Type-1) hypervisor
- Community driven open source project (http://xhypervisor.org, xvisor-devel@googlegroups.com)
- 9+ years of development and hardening (since 2010)
- Supports variety of architectures: ARMv5, ARMv6, ARMv7, ARMv7ve, ARMv8, x86_64, and RISC-V (work-in-progress)
- **Primarily designed and developed for embedded virtualization**
- First paper in IEEE PDP 2015 titled "***Embedded Hypervisor Xvisor: A comparative analysis***"

Xvisor: Traditional Classification

How does Xvisor compare with other hypervisors ?



Type1 Examples: Xvisor, Xen, VMWare ESX Server, Microsoft HyperV, OKL4 Microvisor, etc

Type2 Examples: Linux KVM, FreeBSD Bhyve, VMWare Workstation, Oracle VirtualBox, etc

Xvisor: Features

Lots of features

- **Virtualization Infrastructure:**
  - Device tree based configuration
  - Soft real-time pluggable scheduler
  - Hugepage for Guest and Host
  - Tickless and high-resolution timekeeping
  - Host device driver framework
  - Threading framework
  - Runtime loadable modules
  - Management terminal
  - Light-weight virtual filesystem (VFS)
  - White-box testing framework
  - … Many More …

Xvisor: Features (Contd.)

Lots of features

- **Domain Isolation:**
  - VCPU and Host Interrupt Affinity
  - Spatial and Temporal Memory Isolation
- **Device Virtualization:**
  - Pass-through device support (**Not available for RISC-V due to lack of IOMMU**)
  - Block device virtualization
  - Network device virtualization
  - Input device virtualization
  - Display device virtualization
  - VirtIO v0.9.5 for para-virtualization
- **Domain Messaging:**
  - Sharing On-chip Coprocessor
  - Zero-copy Inter-Guest Communication

Xvisor: Key Aspects

Everything is a VCPU in Xvisor

- Scheduling entity is VCPU
- Two types of VCPUs:
  1. **Normal VCPU:** A VCPU belonging to Guest/VM
  2. **Orphan VCPU:** A VCPU belonging to Hypervisor for background processing
- Orphan VCPUs are very light-weight compared to Normal VCPUs
- Scheduler supports **pluggable scheduling policy**, available policies:
  - Fixed priority round-robin
  - Fixed priority rate monotonic
- Scheduling policies are soft real-time

Xvisor: Key Aspects (Contd.)

Three Execution Context

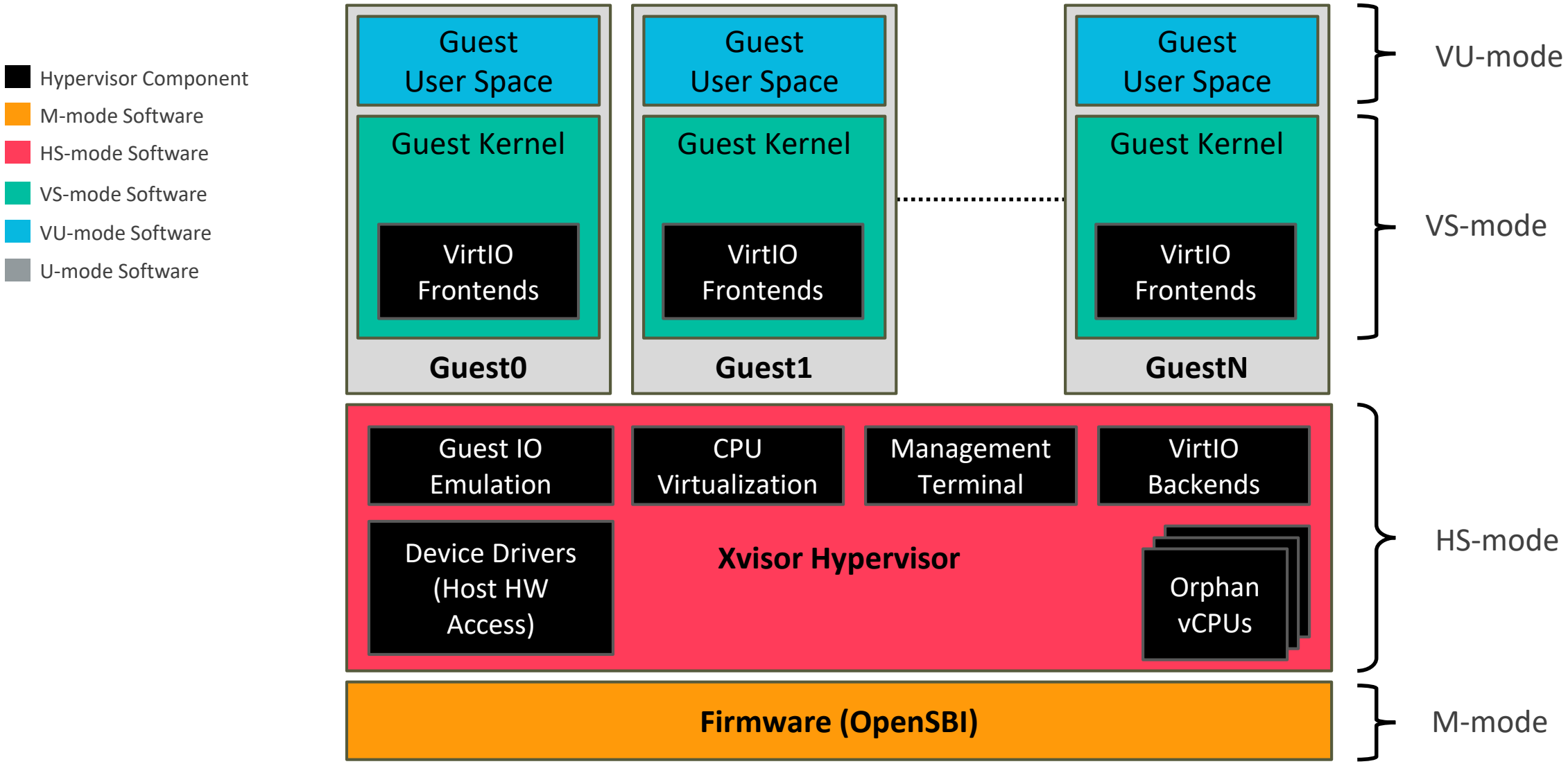
- Linux kernel runs in two possible contexts: **process context** OR **interrupt context**
- Xvisor runs in three possible contexts:
  1. **Normal Context:** Handling trap for a Normal VCPU
  2. **Orphan Context:** Running an Orphan VCPU
  3. **Interrupt Context:** Processing Host interrupt
- **Allowed to sleep in Orphan Context only**
- This ensures:
  - Host interrupt handlers can never blocks
  - Trap handling (Stage2 traps, MMIO traps, etc) for Guest can never block
- **Xvisor provides predictable delay in MMIO emulation and Trap handling for Guest**

Xvisor RISC-V

The RISC-V port of Xvisor

Xvisor RISC-V

World's first Type-1 RISC-V hypervisor



Xvisor RISC-V: VCPU Context

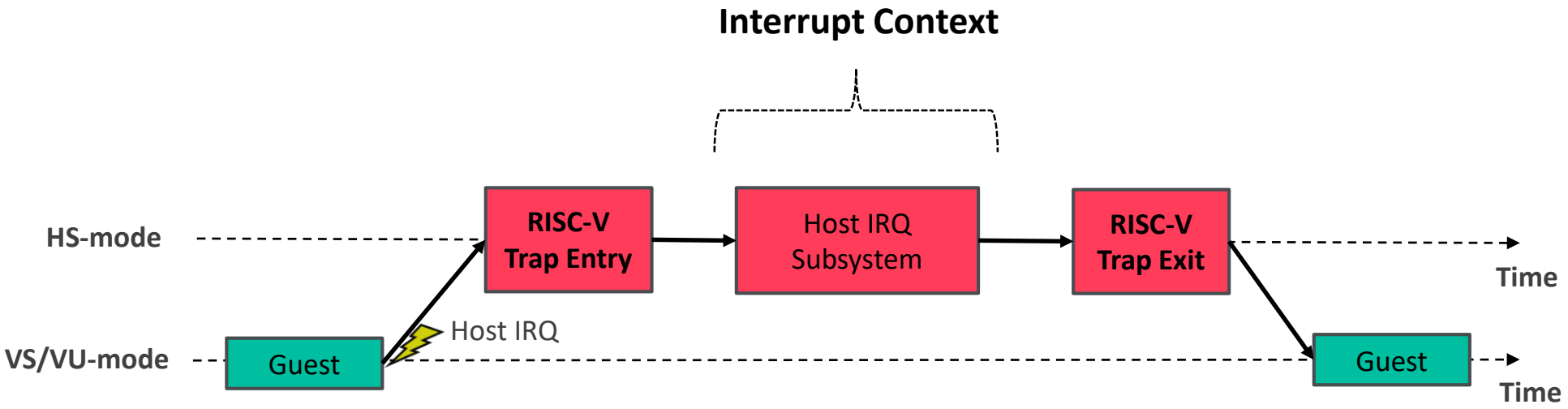
What things are saved/restored for a VCPU ?

struct arch_regs (Orphan and Normal VCPUs)		
Field	**Description**	**Save/Restore**
zero	Zero register	---
ra	Return address register	Trap Entry/Exit
sp	Stack pointer register	Trap Entry/Exit
gp	Global pointer register	Trap Entry/Exit
tp	Thread pointer register	Trap Entry/Exit
a0-a7	Function argument registers	Trap Entry/Exit
t0-t6	Caller saved registers	Trap Entry/Exit
s0-s11	Callee saved register	Trap Entry/Exit
sepc	Program counter	Trap Entry/Exit
sstatus	Shadow SSTATUS CSR	Trap Entry/Exit
hstatus	Shadow HSTATUS CSR	Trap Entry/Exit
sp_exec	Stack pointer for traps	Trap Entry/Exit

struct riscv_priv (Normal VCPUs Only)		
Field	**Description**	**Save/Restore**
xlen	Register width	---
isa	Feature bitmap	---
vsstatus	SSTATUS CSR	Context Switch
vsie	SIE CSR	Context Switch
vstvec	STVEC CSR	Context Switch
vsscratch	SSCRATCH CSR	Context Switch
vsepc	SEPC CSR	Context Switch
vscause	SCAUSE CSR	Context Switch
vstval	STVAL CSR	Context Switch
vsip	SIP CSR	Context Switch
vsatp	SATP CSR	Context Switch
fp	Floating-point Registers	Context Switch

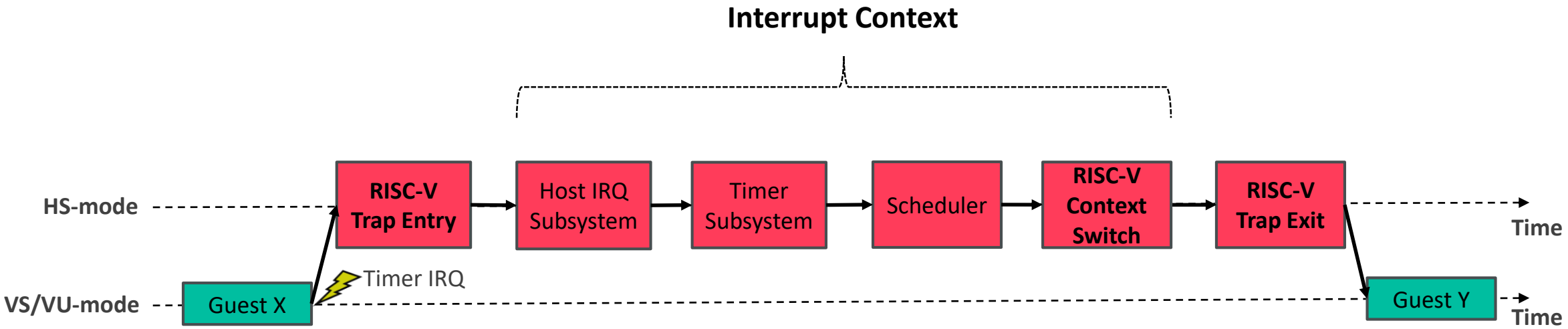
Xvisor RISC-V: Host Interrupts

How are host interrupts handled ?



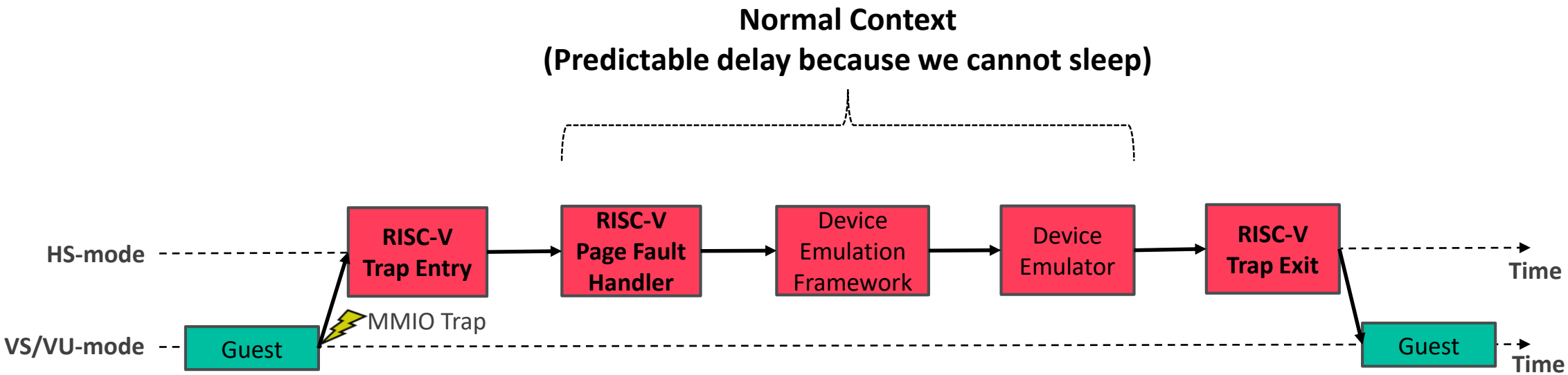
Xvisor RISC-V: Context Switch

How is a Guest VCPU preempted ?



Xvisor RISC-V: Guest MMIO Emulation

How is a Guest VCPU MMIO trap handled ?



Xvisor RISC-V: Guest RAM handling

How is Guest RAM accessed from Xvisor ?

- **Guest RAM is not pre-mapped in HS-mode page table** to:
  - Reduce memory consumed by HS-mode page table
  - Avoid cache aliasing by not having two different virtual addresses for Guest RAM pages
- **Guest RAM is accessed 4K (Page Size) at a time iteratively**, as follow:
  - Map 4K page of Guest RAM in HS-mode page table
  - Access 4K page of Guest RAM using hypervisor virtual address
  - Unmap 4K page of Guest RAM from HS-mode page table
- **RISC-V unprivileged load/store can be used improve Guest RAM accesses**
- **Host hugepages** to make Xvisor memory access faster
- **Guest hugepages** to make Guest OS memory access faster
- **RISC-V hugepage size is:** 2M or 1G for RV64 and 4M for RV32

Xvisor RISC-V: SBI Interface

Syscall style interface between Host and Guest

- **SBI = Supervisor Binary interface**
- SBI v0.1 in-use by Linux kernel
  (Refer, https://github.com/riscv/riscv-sbi-doc/blob/v0.1.0/riscv-sbi.md)
- SBI v0.2 in draft stage

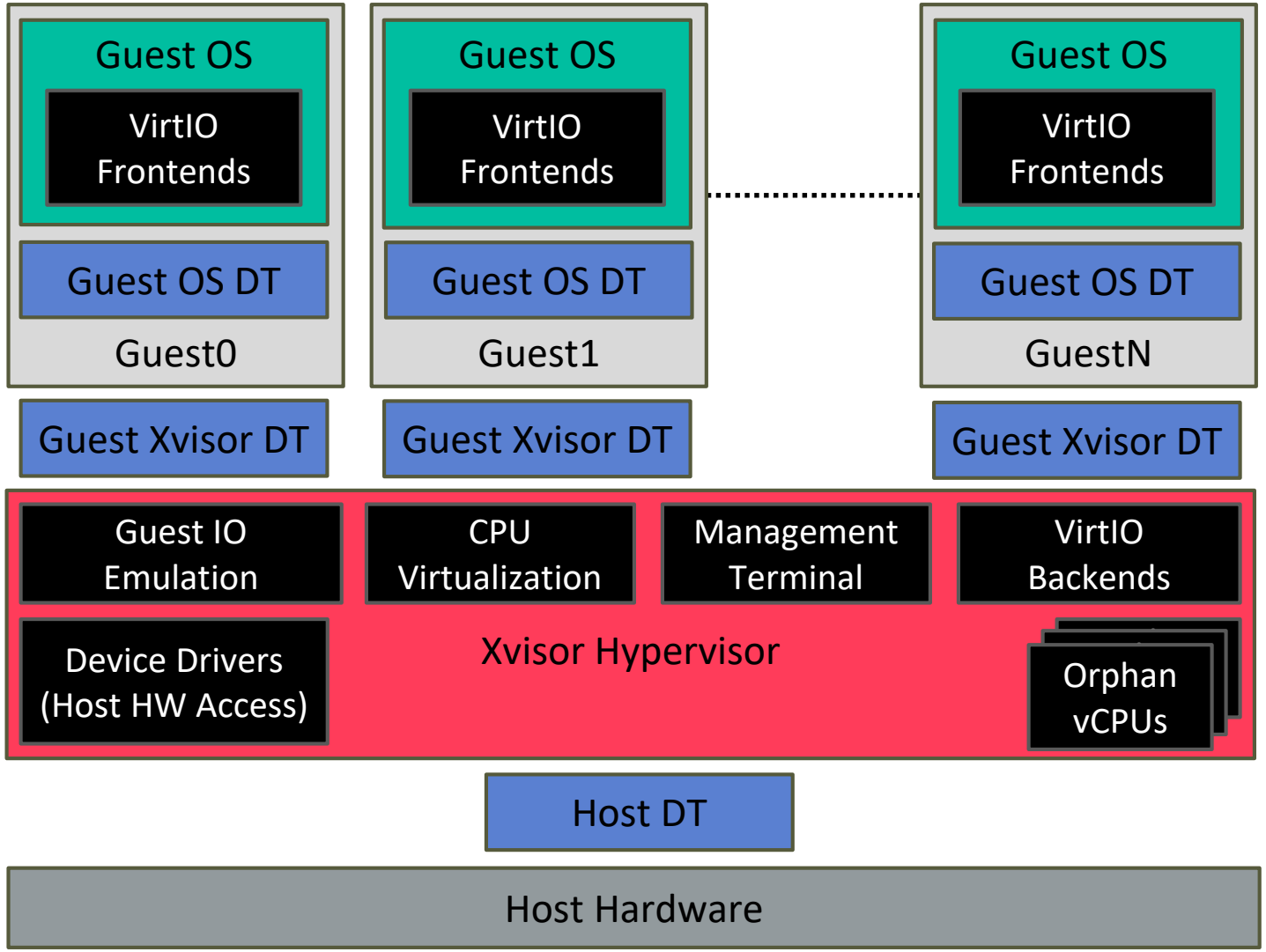
Type	Function	Function ID
Timer	sbi_set_timer	0
IPI	sbi_clear_ipi sbi_send_ipi	3 4
Memory Model	sbi_remote_fence_i sbi_remote_sfence_vma sbi_remote_sfence_vma_asid	5 6 7
Console	sbi_console_putchar sbi_console_getchar	1 2
Shutdown	sbi_shutdown	8

Xvisor RISC-V: Device tree based configuration

Using DT for both Host and Guest

Three types of device tree (DT):

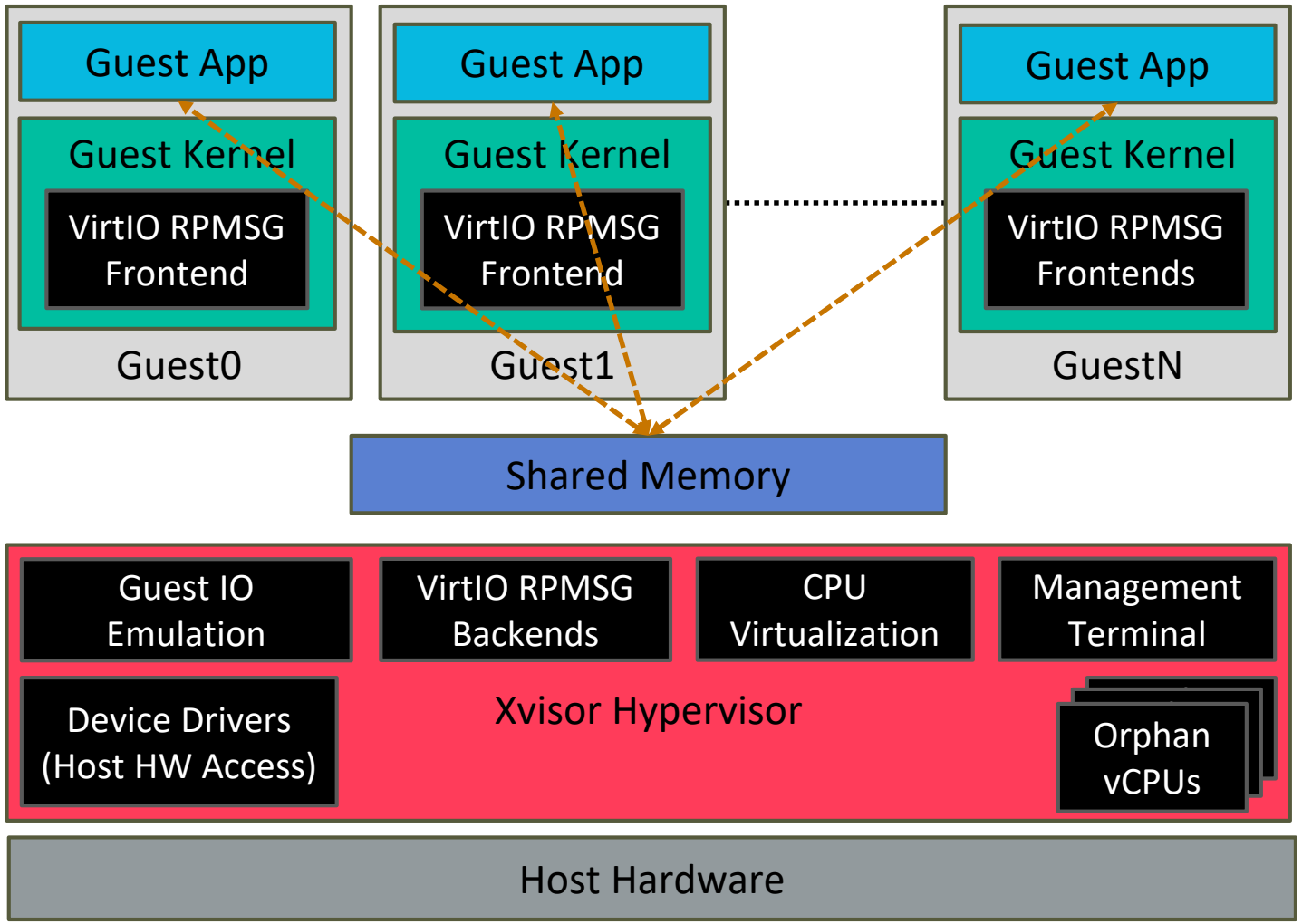
1. **Host DT:**
   - Device tree which describes underlying host HW to Xvisor
   - Used by Xvisor at boot-time
2. **Guest Xvisor DT:**
   - Device tree which describes Guest virtual HW to Xvisor
   - Used by Xvisor to create Guest
3. **Guest OS DT:**
   - Device tree which describes Guest virtual HW to Guest OS
   - Used by Guest OS at boot-time



Xvisor RISC-V: Zero-copy Inter-Guest Transfer

Transferring data between Guests with minimum overhead

- Achieved using:
  1. **VirtIO RPMSP:**
     - Used for control messages
     - Name-service notifications
  2. **Shared Memory:**
     - Used for actual data transfers
     - Very fast zero-copy
- Linux applications can transfer data across Guests using **RPMSG character device**
- **VirtIO Network** can also be used in-place of VirtIO RPMSG



Xvisor RISC-V: Code Size and Memory Usage

Lines of code, binary size and runtime memory usage

Lines of Code	Comments	Code
arch/riscv/	2099	6357
core/	9151	35989
commands/	1031	10409
daemons/	147	524
drivers/	19094*	62395*
emulators/	4117*	25375*
libs/	6023	18296
TOTAL	**41662**	**159345**

***** Can be further decreased or increased based on compile-time configuration

BLOB	Size
.text	580 KB*
.data	50 KB
.rodata	205 KB*
.bss	141 KB
vmm.bin	**849 KB**

Runtime Memory	Size
Text memory freed at boot-time	136 KB
Typical memory usage	22.6 MB*
Max VAPOOL limit	32 MB*

NOTE: Stats gathered from Xvisor-next on 21st October 2019 for RV64GC

Xvisor RISC-V: Ideal for Embedded Systems

Why is Xvisor RISC-V ideal for Embedded Systems ?

- No dependency on any Guest OS for running management tools
- Single software providing complete virtualization solution
- Guest types described using device tree instead of fixed Guest types
- Soft real-time and pluggable scheduler
- Predictable delay in MMIO emulation and Trap handling for Guest
- Para-virtualization complying open-standards (such as VirtIO)
- Zero-copy inter-guest communication
- Low memory footprint with reasonable code size
- Playground for academic research

Status & Future Work

Where are we ? and What next ?

Xvisor RISC-V: Current State

Where are the patches ?

- Initial Xvisor RISC-V port was released last year (21st October 2018) in Xvisor v0.2.11
  - Just capable of booting on QEMU without using any HS-mode CSRs
  - QEMU with H-Extension was not available at that time
- Currently, Xvisor RISC-V is in very good shape (comparable to Xvisor ARM64):
  - Able to boot Guests with multiple VCPUs on SMP Host
  - Able to boot Open Embedded and Fedora as Guest OS
- **Xvisor v0.3.0 release will have a feature complete RISC-V port** but it is delayed due to:
  - Few pending RISC-V spec changes
  - Few pending QEMU H-Extension fixes
- **To play with Xvisor RISC-V on QEMU refer:**
  - **docs/riscv/riscv-virt-qemu.txt** in latest Xvisor-next sources
  - **Work-in-progress QEMU with H-Extension** at https://github.com/kvm-riscv/qemu
- **To follow Xvisor RISC-V development,** please join **xvisor-devel@googlegroups.com**

Xvisor RISC-V: TODO List

What next ?

- Get Xvisor RISC-V 32-bit working
- SBI v0.2 base and replacement extensions support
- SBI v0.2 para-virtualized time accounting
- Access Guest RAM using RISC-V unprivileged load/store
- Loadable module support
- Virtualize vector extensions
- Libvirt support
- Allow 32bit Guest on 64bit Host (**Defined in RISC-V spec**)
- Allow big-endian Guest on little-endian Host and vice-versa (**Defined in RISC-V spec**)
- **..... and more .....**

Xvisor RISC-V Demo

Xvisor running on QEMU RISC-V

Questions ???

Western Digital®

Backup

RISC-V H-Extension: Compare ARM64

How is RISC-V H-Extension compared to ARM64 virtualization?

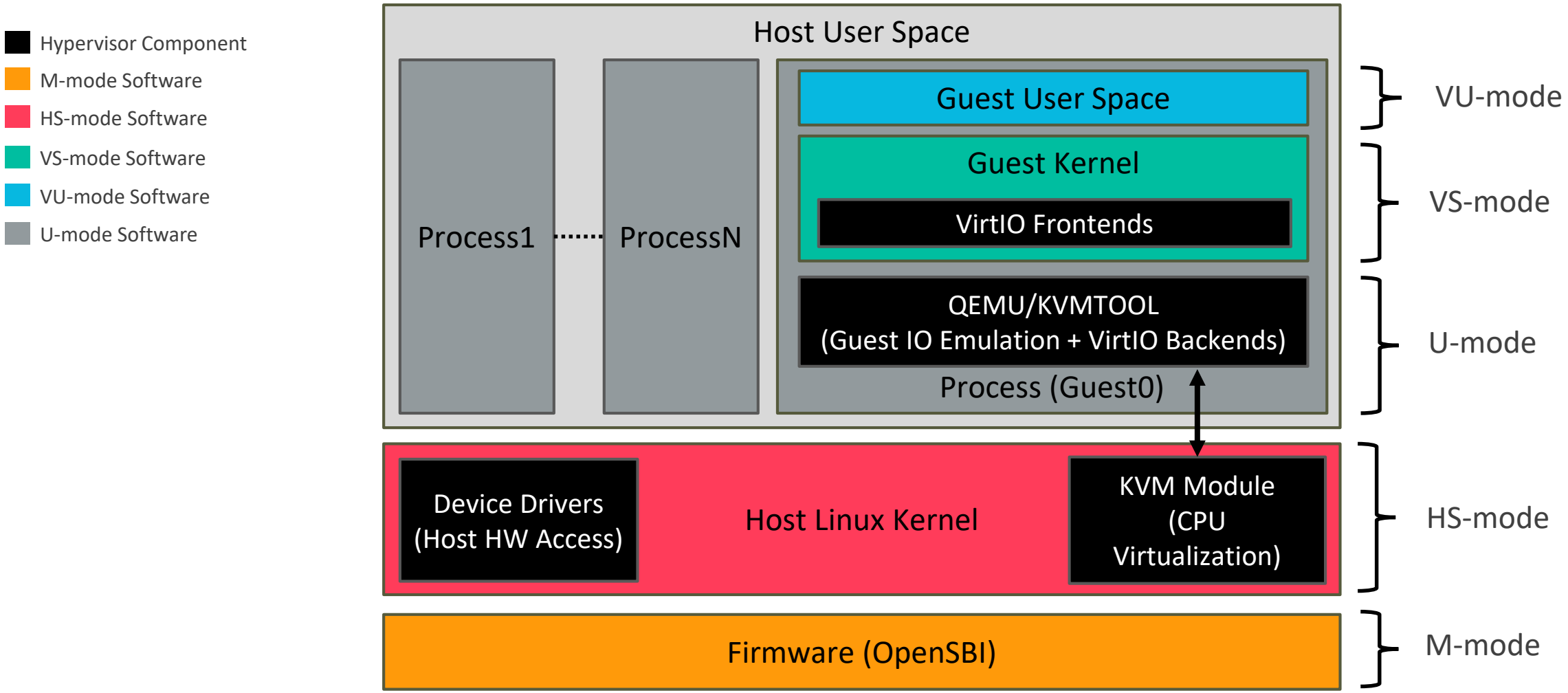
RISC-V H-Extension v0.4 draft	ARM64 (ARMv8.x) Virtualization
No separate privilege mode for hypervisors. Extends S-mode with hypervisor capabilities (HS-mode) and Guest/VM run in virtualized S-mode/U-mode (VS-mode or VU-mode).	**Separate EL2 exception-level for hypervisors** with it's own <xyz>_EL2 MSRs. The Guest/VM will run in EL1/EL0 exception levels.
Well suited for both Type-1 (baremetal) and Type-2 (hosted) hypervisors. The S<xyz> CSRs access from VS-mode map to special VS<xyz> CSRs which are only accessible to HS-mode and M-mode.	**Special ARMv8.1-VHE Virtualization Host Extension for better performance of Type-2 (hosted) hypervisor.** Allows Host kernel (meant for EL1) to run in EL2 by mapping <xyz>_EL1 MSRs to <abc>_EL2 MSRs in Host mode.
Virtual interrupts for Guest/VM injected using VSIP CSR. The hypervisor does not require any special save/restore but it will emulate entire PLIC in software.	**Virtual interrupts for Guest/VM injected using LR registers of GICv2/GICv3 with virtualization extension.** The hypervisor will save/restore LR registers and emulate all GIC registers in software except GIC CPU registers.

RISC-V H-Extension: Compare ARM64 (Contd.)

How is RISC-V H-Extension compared to ARM64 virtualization?

RISC-V H-Extension v0.4 draft	ARM64 (ARMv8.x) Virtualization
Virtual timer events for Guest/VM using SBI calls emulated by hypervisor. The SBI calls trap to hypervisor so save/restore of virtual timer state not required.	**Virtual timer events for Guest/VM using ARM generic timers with virtualization support.** The hypervisor will save/restore virtual timer state and manage virtual timer interrupts.
Virtual inter-processor interrupts for Guest/VM using SBI calls emulated by hypervisor. The hypervisor does not require any special save/restore.	**Virtual inter-processor interrupts for Guest/VM by emulating ICC_SGI1R_EL1 (virtual GICv3) or GICD_SGIR (virtual GICv2).** The save/restore will be handled as part of LR registers save/restore.
Nested virtualization supported using HSTATUS.TVM and HSTATUS.TSR bits. The hypervisor will trap-n-emulate Guest hypervisor capabilities.	**Special ARMv8.3-NV for supporting nested virtualization on ARMv8.** The hypervisor will trap-n-emulate Guest hypervisor capabilities. The ARMv8.4-NV further enhances nested virtualization support.

Linux KVM RISC-V



Xen RISC-V (Work-in-progress)