



Enhanced Virtual Machine Manager (eVMM)

SOFTWARE AND SERVICES

Outline

- Design motivations
- VTx Overview
- eVMM highlights
- VTx capabilities needed
- Design and architectural details of eVMM
- Possible enhancements
- Compiling, running and debugging
- Discussion

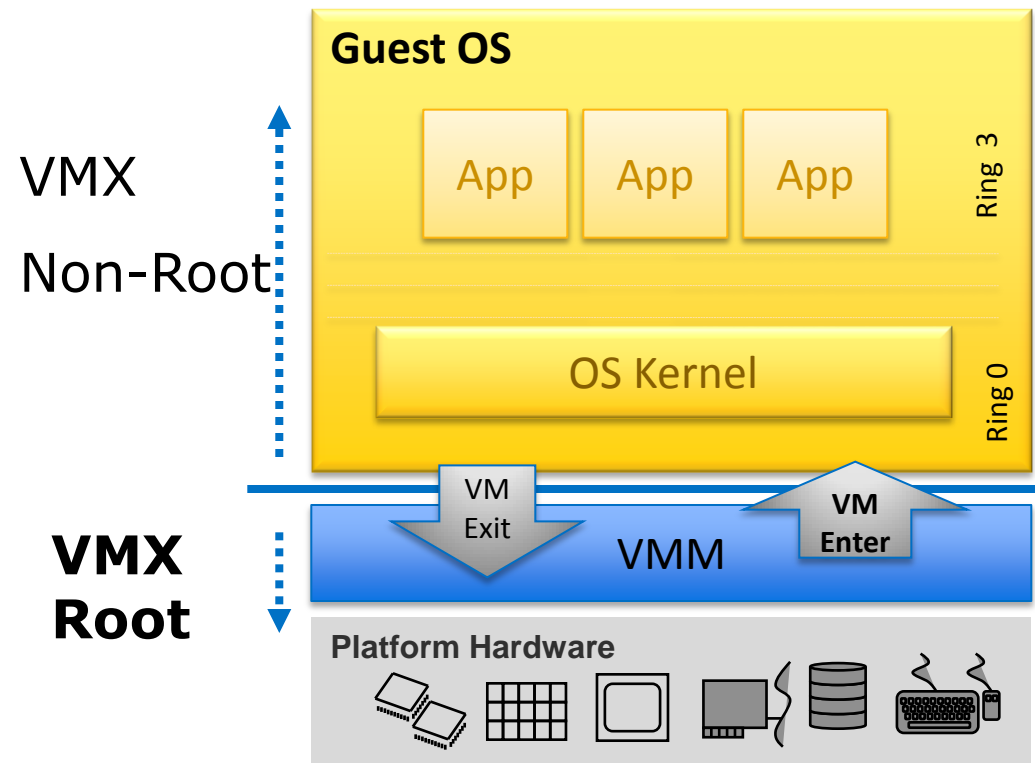
Design Motivations

- Create a VTx Monitor with support for one Primary guest that is the platform owner
- Lowest TCB possible
- Minimal degradation of primary guest
- Main purpose of this VMM is to de-privilege the guest
 - Provides CPU and Memory Virtualization
 - Supports EPT's per Guest
- Maximize the TAM
 - Only depend on CPU VT feature and has no platform dependency
 - No Interrupt or device virtualization
 - Devices are Pass-through to Primary guest OS

eVMM Highlights

- Type 1 VTx Monitor
- Lightweight
 - ~40K lines of code
 - Binary size – 200KB (zipped)
 - Low performance overhead (less than 1-2%)
 - Platform/BIOS independent - requires only VT and XD in BIOS setting
- Guest OS Support
 - Windows, Linux and Android guests (both 32 and 64 bit)
- Supported on Core i & Atom processors
- SMP Support – Supports up to 80 logical processors

VTx Overview

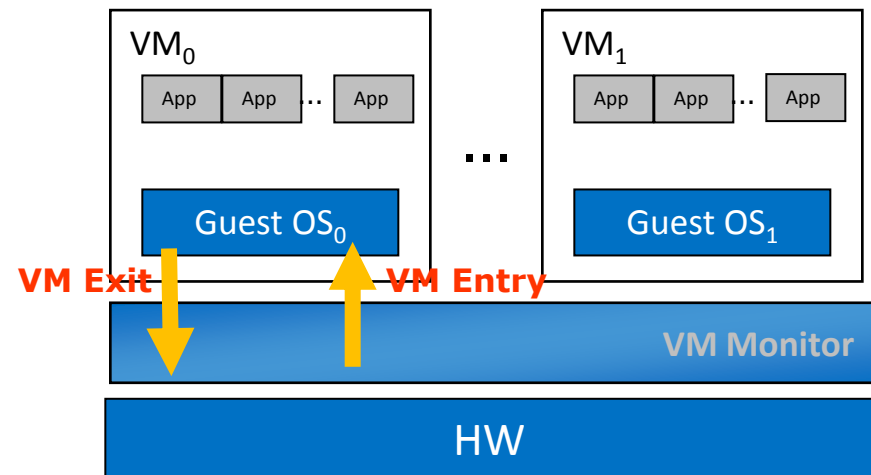


- **Additional Mode - VMX Root**
 - Privileged mode
 - Below Ring 0
- **Hardware Assisted Transitions**
 - **Guest To VMX Root**
 - VM Exit
 - On CPU actions
 - Configurable via VMCS
 - VM Call
 - Triggered by software
 - Synchronous
 - **VMX Root To Guest**
 - VM Enter
 - *VMLaunch*
 - *VMResume*

SOFTWARE AND SERVICES

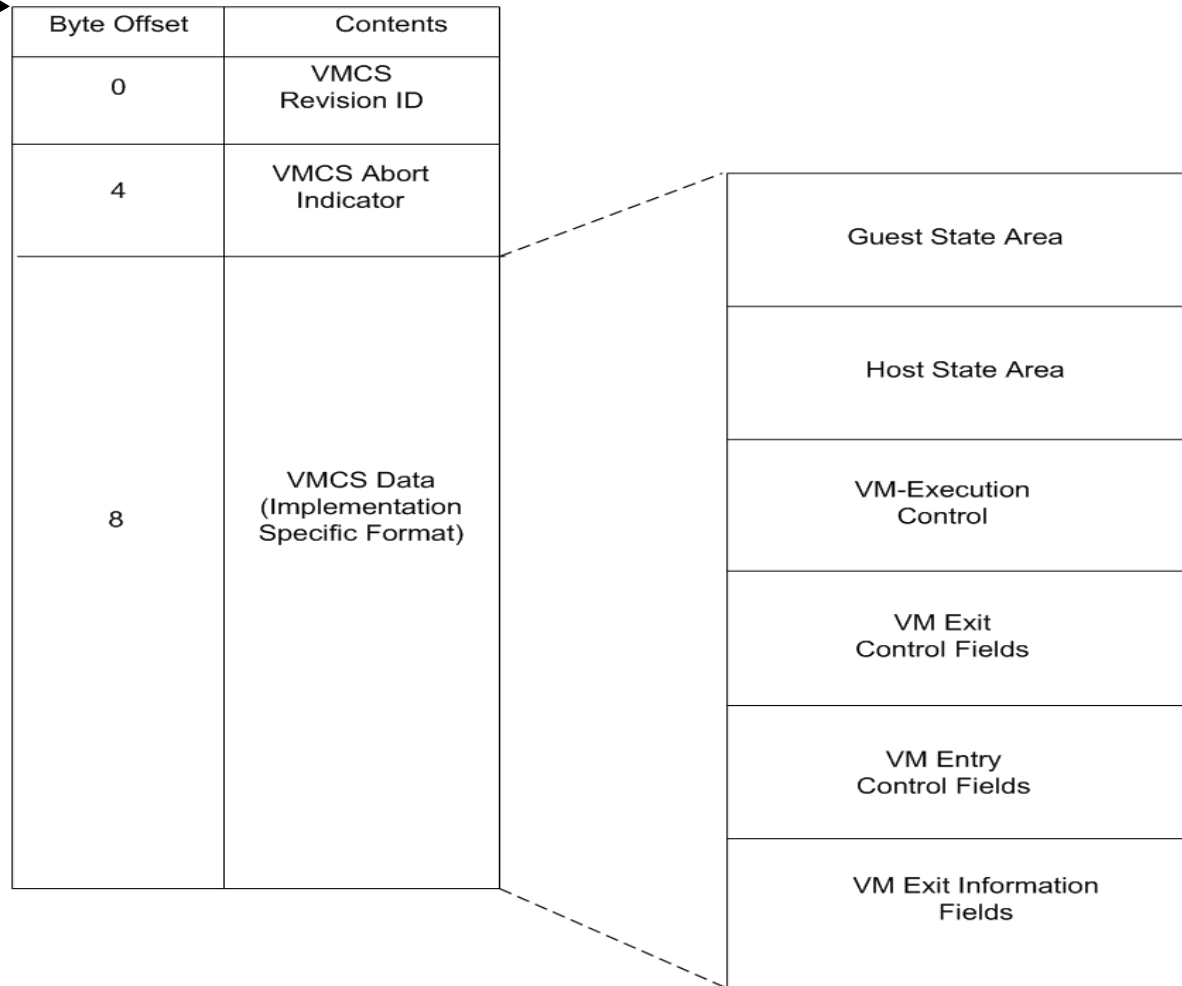
VTx Transitions

- VM Entry
 - Transition from VMM to Guest
 - Enters VMX non-root operation
 - Loads Guest state and Exit criteria from VMCS
 - VMLAUNCH instruction used on initial entry
 - VMRESUME instruction used on subsequent entries
- VM Exit
 - VMEXIT instruction used on transition from Guest to VMM
 - Enters VMX root operation
 - Saves Guest state in VMCS
 - Loads VMM state from VMCS



VTx Control - VMCS

VMCS Pointer →



VM Exit

- **Causes of VM Exit**

VM Exits occur as a result of certain instructions and events in VMX non-root operation

- **Instructions:**

Un-Conditional: CPUID, GETSEC,3 INVD, VMCALL,5 VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON

Conditional: HLT, IN, OUT, INLVP, LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR,

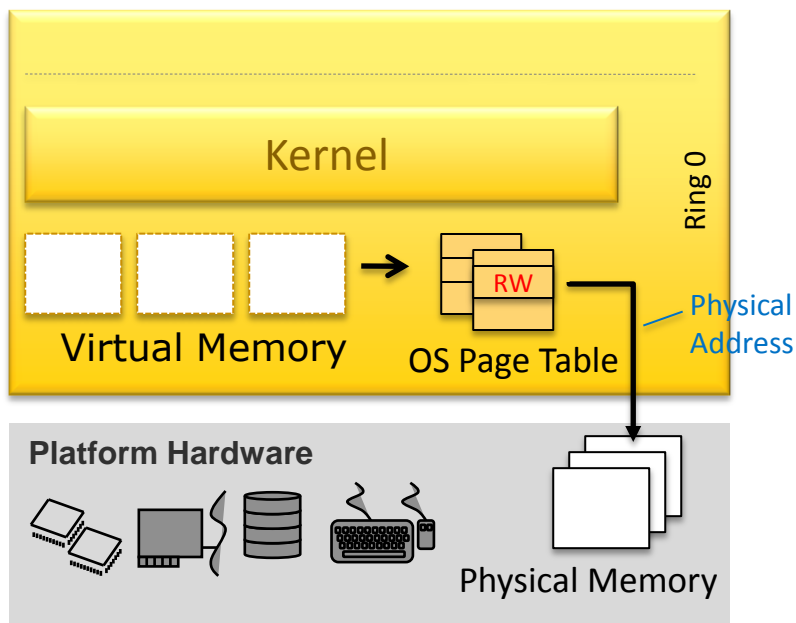
- **Other:**

- Exceptions
- Triple Fault
- External Interrupt
- NMI
- INIT Signals
- Start-up IPIs
- Task Switches
- SMIs
- VMX pre-emption timer

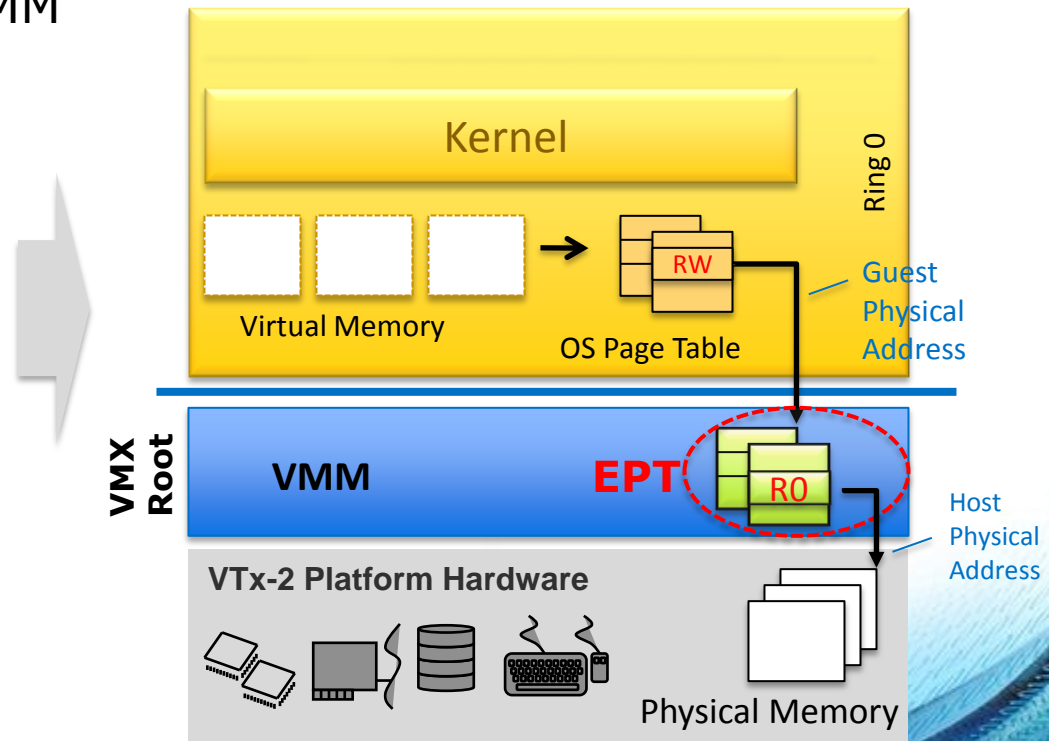
VTx-2 - EPT

• Extended Page Table (EPT)

- Additional Page Permissions outside OS
- Maintained by VMM
- Enforced by hardware
 - Violations cause VMExit
 - Transition control to VMM



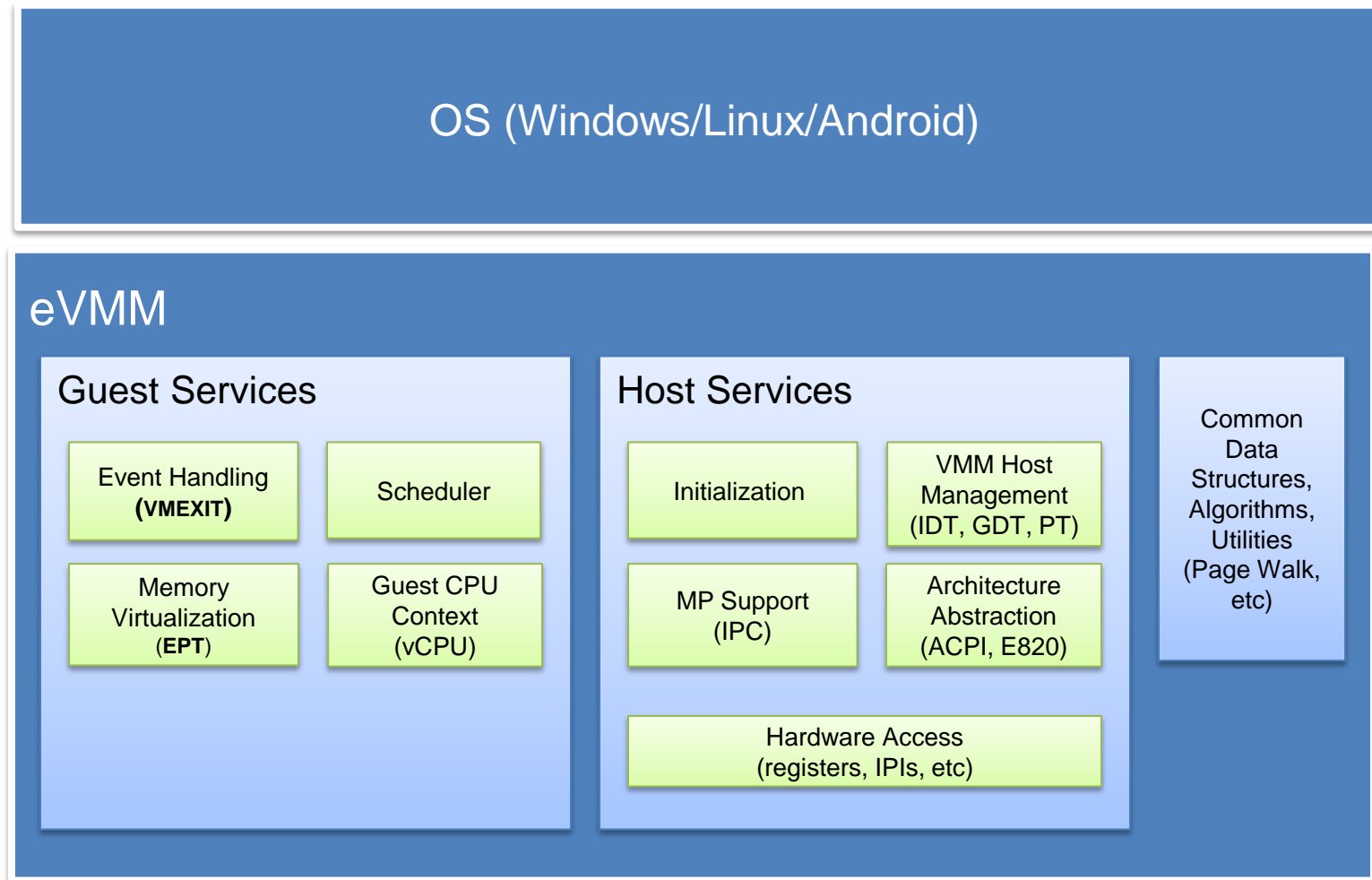
SOFTWARE AND SERVICES



VTx Capabilities Needed

- EPT
 - Has 2MB page size support for EPT
- Unrestricted Guest (UG)
 - UG allows the guest to start in Real Mode. Available on all latest Intel CPUs

eVMM Components



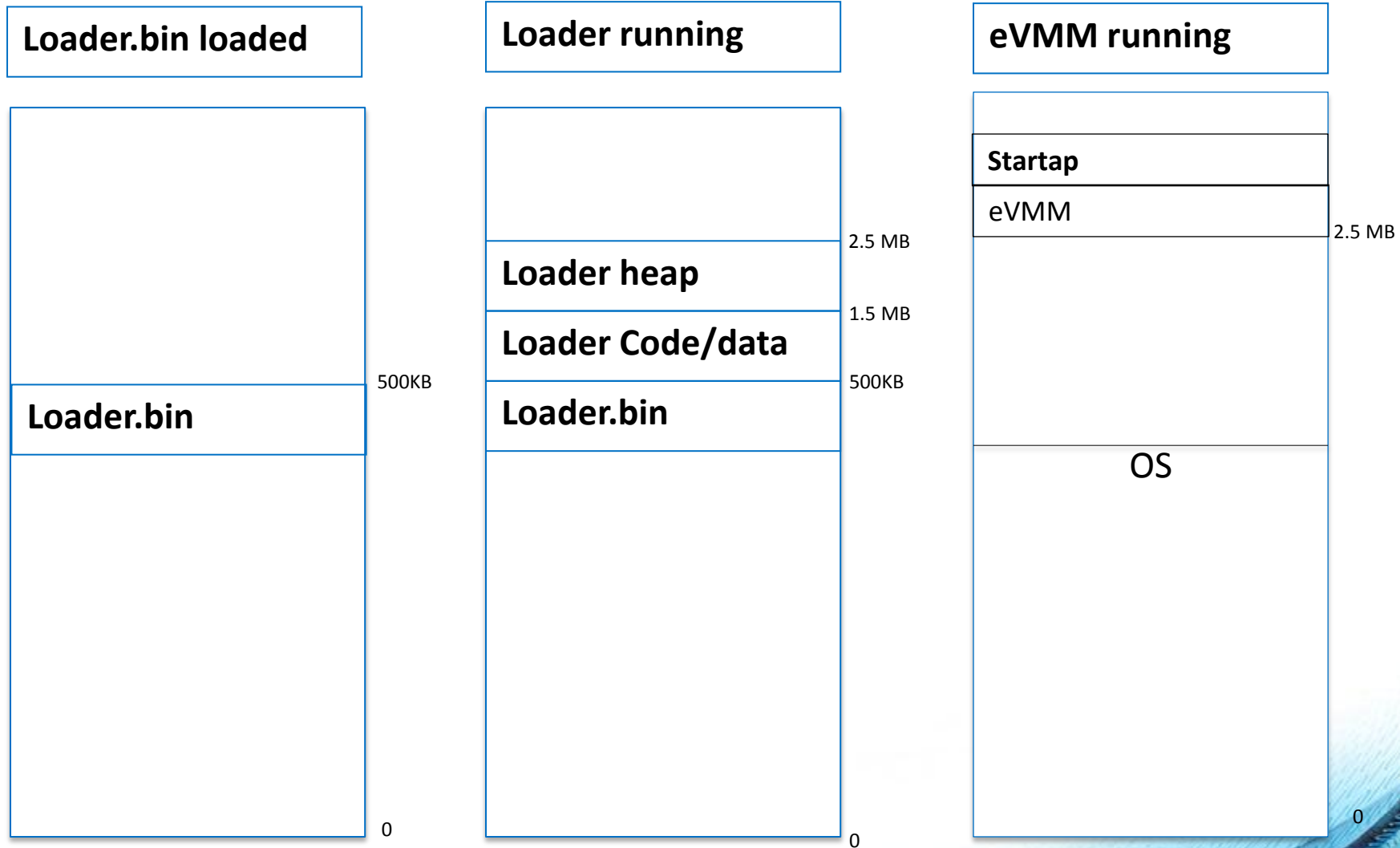
SOFTWARE AND SERVICES

Pre-OS Loader

- Simple, generic and self-contained binary (Loader.bin)
- Runs from (and within) a block of memory
- Works with Grub, EFI
- Loader.bin is packaged together with three other binaries
 - PE-format eVmmh – Actual Loader
 - PE-format StartAp – For CPU Initialization
 - PE-format eVMM - Hypervisor
- Loader.bin provides unzip and PE loader utilities
- Loader is position independent code

Code link: [loader\pre_os](#)

Run-Time Memory Layout



SOFTWARE AND SERVICES

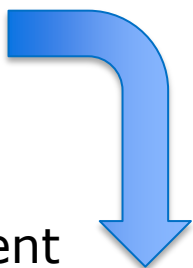
CPU Initialization -- StartAp

- BSP:
 - Sets up AP real-mode code in memory
 - Sends SIPI to all AP's
 - Switches to 64-bit page protected mode
 - Passes control to eVMM (next slide)
- AP:
 - Receives SIPI
 - Executes real-mode code, switches to 64-bit page protected mode
 - Passes control to eVMM (next slide)

Code link: [loader\startap](#)

SOFTWARE AND SERVICES

eVMM Initialization

- BSP:
 - Gets data passed in by loader: number of CPU's, stack size, E820 table, debug setting, guest CPU states, etc. (`vmm_bsp_proc_main()`)
 - Initializes debugging output
 - Sets up Vmm host environment (GDT, IDT, CR's)
 - Prepares E820 map for the guest OS (next slide)
 - Sets up host memory manager and EPT
 - Waits for AP's to complete?
 - Launches guest
 - AP:
 - Sets up Vmm host environment
 - Launches guest in "virtual" wait-for-SIPI state.
- 

Code link: [vmm\vmm.c](#)

SOFTWARE AND SERVICES

E820 Memory Map

Loader:

- Converts Grub/EFI memory map to E820 table
- Passes E820 table to eVMM.

eVMM:

- Reserves its own footprint from E820 table
- Intercepts/Emulates BIOS E820 calls from guest OS

Code link: [vmm\arch\e820_abstraction.c](#)

SOFTWARE AND SERVICES

eVMM Host Memory Management

- eVMM maintains its own page table
- All guest pages are mapped 1:1
- eVMM code/data/stack/Heap resides in physically continuous memory. This is also mapped 1:1.
 - Few exceptions – Virtual 0 page, stack boundaries, extended heap
- Permissions
 - By default all memory pages are mapped with R/W permissions
 - Guest pages are mapped with R/W permissions
 - eVMM code pages are mapped with R/X permissions
 - eVMM read only data pages are mapped with R permissions

eVMM Host Memory Management

- eVMM always maps all memory below 4G in its own page table
- All memory above 4G is mapped based on the available memory on the platform
- eVMM can access any guest memory

Code link: [vmm\memory\memory_manager\host_memory_manager.c](#)

Guest Memory Management (GPM)

- eVMM manages guest-owned memory with GPM
- For each guest there are forward and reverse mappings (collectively called GPM).
- GPM can perform two-way translation of HPA & GPA.
- Guest memory isolation, memory hiding, and memory re-mapping can be easily accomplished
 - eVMM-owned code and data pages are not present in Guest GPM
- EPT is created from GPM

Code link: [vm\memory\memory_manager\gpm.c](#)

MAM: Memory Address Mapper

- A set of functions to handle address mapping from one address to another one at 4K granularity.
- Address mapping is being handled in a very similar way as page table.
- Perform conversion from existing address mapping to hardware-compliant page tables, EPT tables, or even VT-d tables.
- Being used by both GPM and Host Memory Manager.

Code link: [vmm\memory\memory_manager\memory_address_manager.c](#)

eVMM Heap Management

- Maintains its own heap
- Handles dynamic host memory allocation
 - Alloc and free are supported for any size
- eVMM also supports extended heap which can be non-physically continuous. (Not being used currently)

Code link: [vmm\utils\heap.c](#)

IPC - Inter Processor Communication

- NMI interrupt is used for IPC message signaling.
- Inter Processor Interrupts (IPI's) are generally used for Inter Processor Communication
 - Utilization of this mechanism by eVMM will require virtualization of interrupt controller and interrupt modeling.
- For NMI's received in Guest Mode eVMM registers for NMI VMEXIT
- For NMI's received Host mode uses NMI Window VMEXIT to inject NMI into guest
- The main problem here is distinguishing between platform-generated NMI's and IPC-generated NMI's on a receiving side.

Event Handling

- There are handlers in eVMM for most VMEXITs. (Next Slide)
- Most VMEXITs are not enabled by default
 - Can be enabled optionally as required.
 - Some VMEXIT's are always enabled – For Example NMI and NMI window, some control registers access, CPUID
- VMEXIT's are hooked to provide callbacks for any additional processing

Code link: [vm\vmexit](#)

Event Handling

ID	Exit Reason	Description
18	VMCALL	Calls registered handler for vmcall ID passed in RCX
10	CPUID	Handlers are registered for virtualizing
28	CR access	Used to monitor reads and writes to CR0, CR3 and CR4
46	GDTR IDTR access	Monitor read/write to GDT/IDT
47	LDTR TR access	Monitor read/write to LDT/TR
29	DR access	Monitor read/write to DR

Event Handling (continued)

ID	Exit Reason	Description
48	EPT Violation	If a page with restricted permission was accessed
49	EPT Misconfiguration	Internal EPT setup error
3	INIT signal	Used to virtualize guest INIT
4	SIPI	Used to virtualize guest SIPI
0	Exception or NMI	Exceptions are injected to OS. NMI is used for IPC. If not IPC pending then is injected back to guest.
8	NMI Window	Used to virtualize NMI
2	Triple Fault	Injected to guest
12	HALT instruction	Used to monitor HALT instruction

Event Handling (continued)

ID	Exit Reason	Description
31,32	RDMSR/WRMSR	Monitor MSR read/write
19-27	VMX instructions	Used to virtualize VMX instructions
13	INVD	Used to monitor guest INVD instruction
14	INVLPG	Used to monitor guest INVLPG instruction
55	XSETBV instruction	Used to monitor XSETBV instruction
59	Invalid VMFUNC	Guest executed invalid VMFUNC instruction
9	Task Switch	Task Switch is emulated in eVMM
37	MONITOR TRAP FLAG	Used to be notified after one instruction is executed

eVMM Scheduler

- Includes a simple scheduler
- Saves/Loads guest CPU status
- Assumes only one guest per host CPU
 - Implementation not complete to support multiple guests

Code link: [vmm\guest\scheduler](#)

SOFTWARE AND SERVICES

Common Utilities

- Locking mechanism for synchronization
- Hashing utilities
- Printing functions (for debugging)
- Algorithms like guest page table walk

Power Management

- eVMM supports S3, S4 states
- ACPI table is used for retrieving Power management capabilities, Supported ACPI sleep states, OS wakeup vector
- Retrieving ACPI tables is through the loader
 - Loader scans and retrieves FADT from ACPI memory
 - FADT is passed from loader as parameter to eVmm initialization
 - Currently not implemented in loader

Building eVMM

- Build environment
 - Windows
 - Visual Studio 2010
 - ActivePerl
 - Cygwin
- Build instructions
 - Please refer to `eVmm\docs\readme.txt` for details of building instructions.

eVMM Debugging

- Serial Port Debugging
 - Use macro `VMM_LOG(MASK, LEVEL, ...)`
 - Update `eVmm\loader\pre_os\build_loader.sh`
 - `debug_port=0x3f8` (for COM1)
 - `Files=..."bin/ms/debug/startup.bin" "bin/ms/debug/$1"...`
 - Re-build debug version of eVMM binaries
 - Follow `Readme.txt`
 - Serial port setting
 - Baud rate 115200 N 8 1
- Command Line Interface
 - Not supported

Extending eVMM

- For extending VMEXIT handling for different usages framework is provided
- eVMM provides API's for working with the VMCS
 - `vmm_get_vmcs_guest_state`
 - `vmm_set_vmcs_guest_state`
 - `vmm_get_vmcs_control_state`
 - `vmm_set_vmcs_control_state`
- eVMM provides callback on VMEXIT's
 - `report_uvmm_event`

Code link: [vmm\startup\vmm_extensions.c](#)

Future Enhancements

- VT Nesting
 - Partial support for VT Nesting is available (layering code)
- Multiple Guests
 - Partial support for Multiple Guests is available
 - Scheduler
- Power Management

Backup

SOFTWARE AND SERVICES

Loader.bin Content

1. A file header;
2. A Multi-boot Header for Grub support;
3. Real-mode Chain Loader;
4. Position-independent Starter;
5. PE-format Evmmh;
6. PE-format StartAp;
7. PE-format Evmm;
8. EVmm startup structure; and
9. Guest CPU state structure.

Loader Execution Flow

1. Grub/EFI/SFI loads “loader.bin” to base address, and runs the Starter.
2. Starter saves CPU states (or creates real-mode execution environment), moves Chain_load to address 0x2000.
3. Starter passes control EVmmh.
4. EVmmh moves StartAp and EVmm to run-time location, creates EVmm startup structure, and then passes control StartAp.