Evaluate implementation options of KVM-based Type1 (or 1.5) hypervisor

Jun Nakajima

Contributors:
Chuanxiao Dong, Anthony Xu

Legal Disclaimers

- Intel provides these materials as-is, with no express or implied warranties.
- All products, dates, and figures specified are preliminary, based on current expectations, and are subject to change without notice.
- Intel processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at http://intel.com.
- Some results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.
- Intel and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.

THE LINUX FOUNDATION

Agenda

- Motivation
- Implementation Options
- PoC
- Performance Data
- Our Conclusion
- Next Step

Security Risks of Linux/KVM Guests

- KVM piggybacks on Linux
  - More attack surfaces, making guests more exposed…
- Full access by user-space VMM
- Full access by KVM/Linux Kernel
  - To any guest VM memory, vCPU states, etc.

*: From presentation last KVM Forum: “Manage Session Enhancing KVM for Guest Protection and Security”
https://kvmforum2019.sched.com/event/Tmvt/enhancing-kvm-for-guest-protection-and-security-jun-nakajima-intel-corp

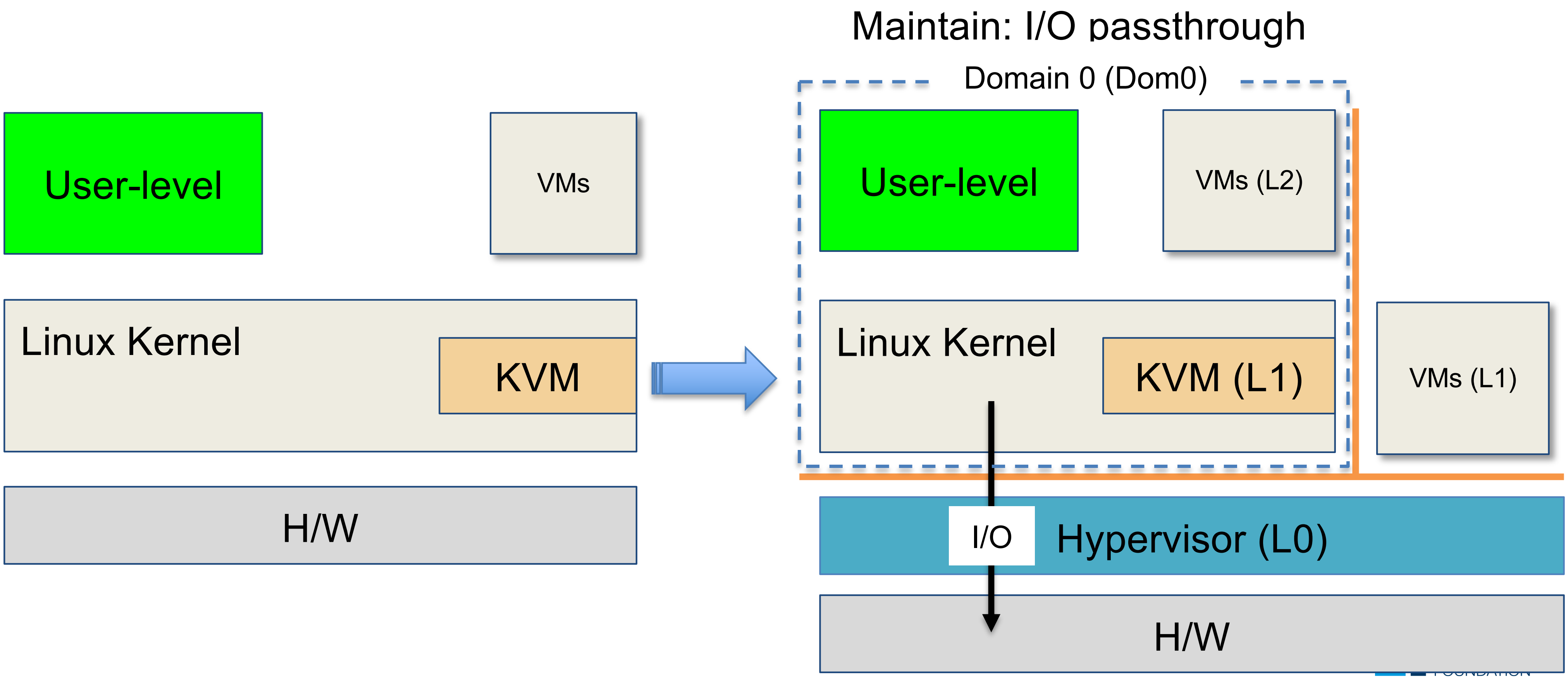
THE LINUX FOUNDATION

Motivation of Type 1.5 Hypervisor

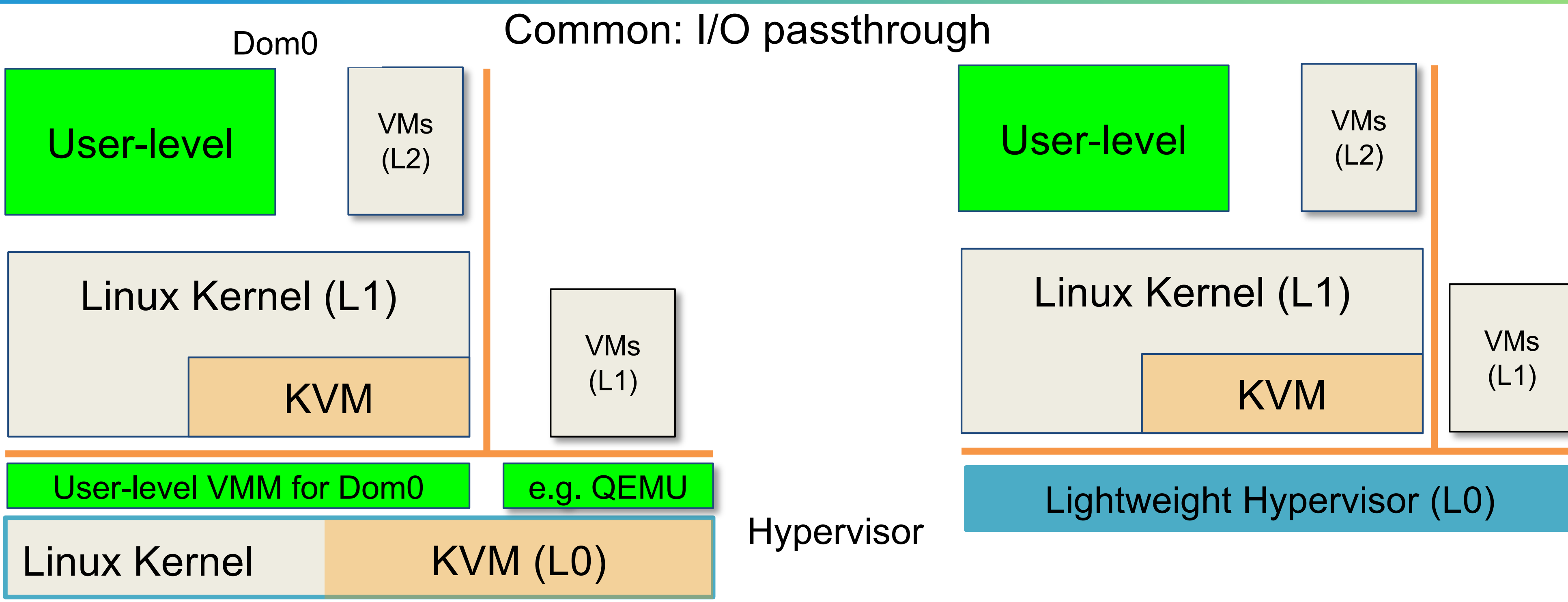
- Separate Hypervisor functionality from Linux
  - Linux handles I/O and user processes
  - Hypervisor is responsible for isolation
    - Thus needs to be trusted
- If trusted, hypervisor can create secure environment
  - TEE (Trusted Execution Environment)
  - Trusted VMs

THE LINUX FOUNDATION

Converting KVM to Type 1.5



Two Extremes

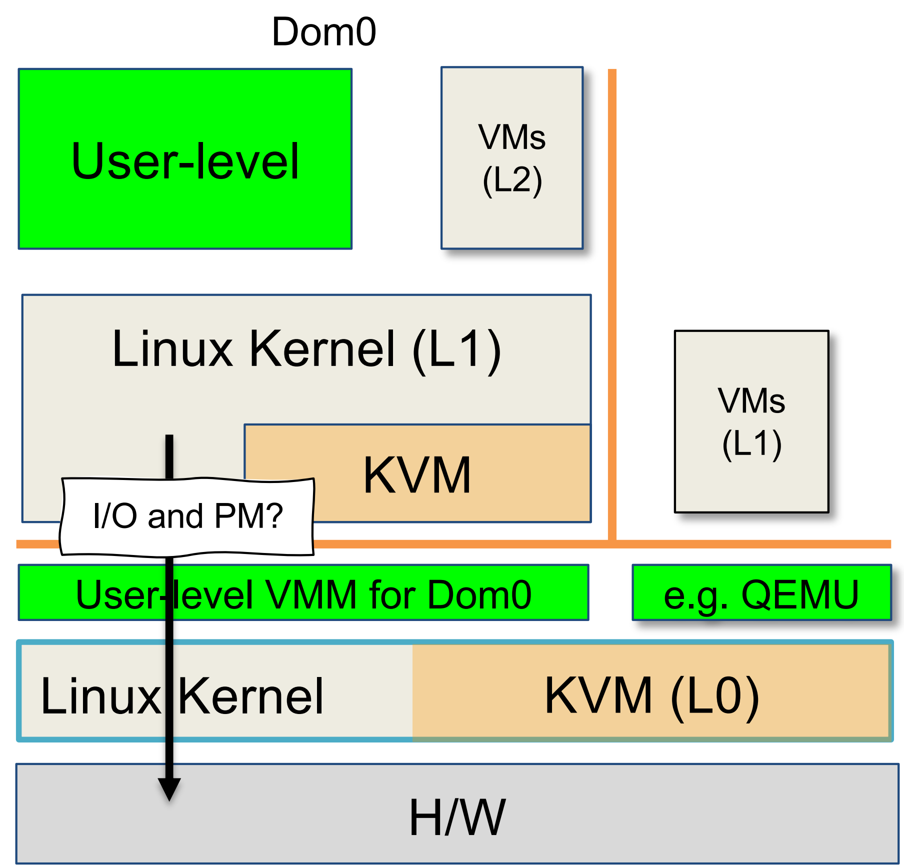


OS functionality (scheduler, memory mgmt, etc.)
Mini-conf Linux/KVM
Boots first and separately

Deprivilege Linux for isolation
Reactive (no scheduler)
Can be loaded by Linux

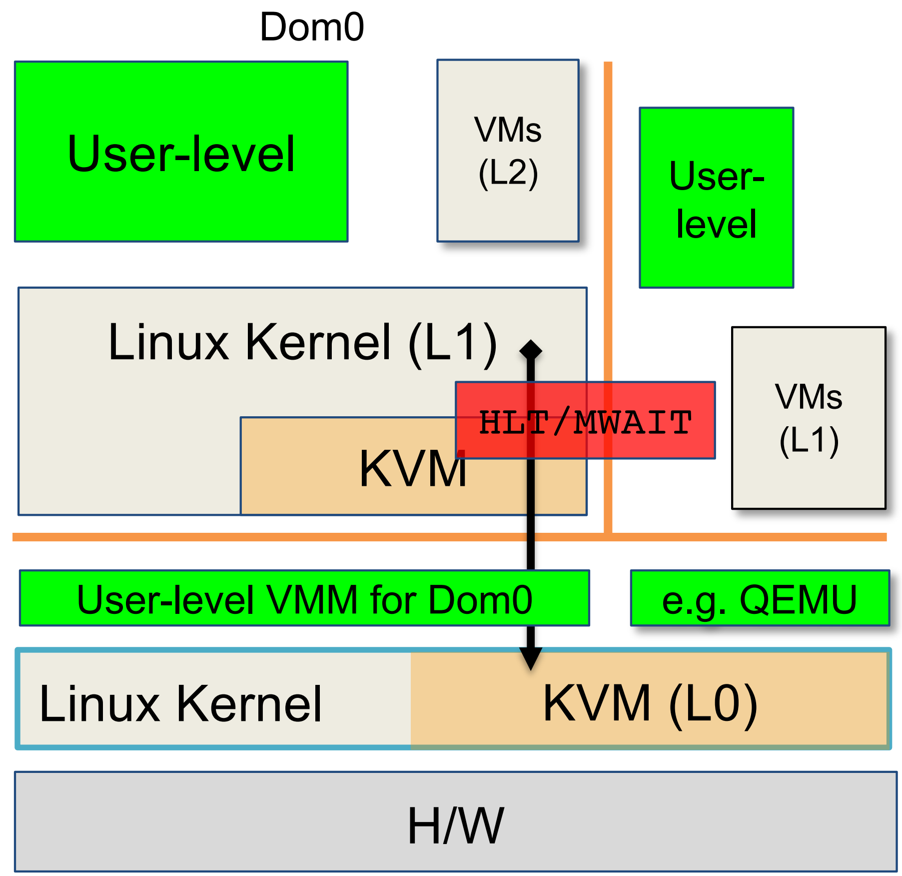
Linux/KVM Hypervisor

- Pros
  - Unmodified guests on L1
  - Benefits from Linux/KVM
- Cons
  - Higher latency to Dom0
    - Scheduling, VM exits
  - Still big (e.g. TCB)
    - Maybe we can deal with it…
  - Virtio for guests
  - Power management (PM)?
    - Who should manage power for CPUs and platform



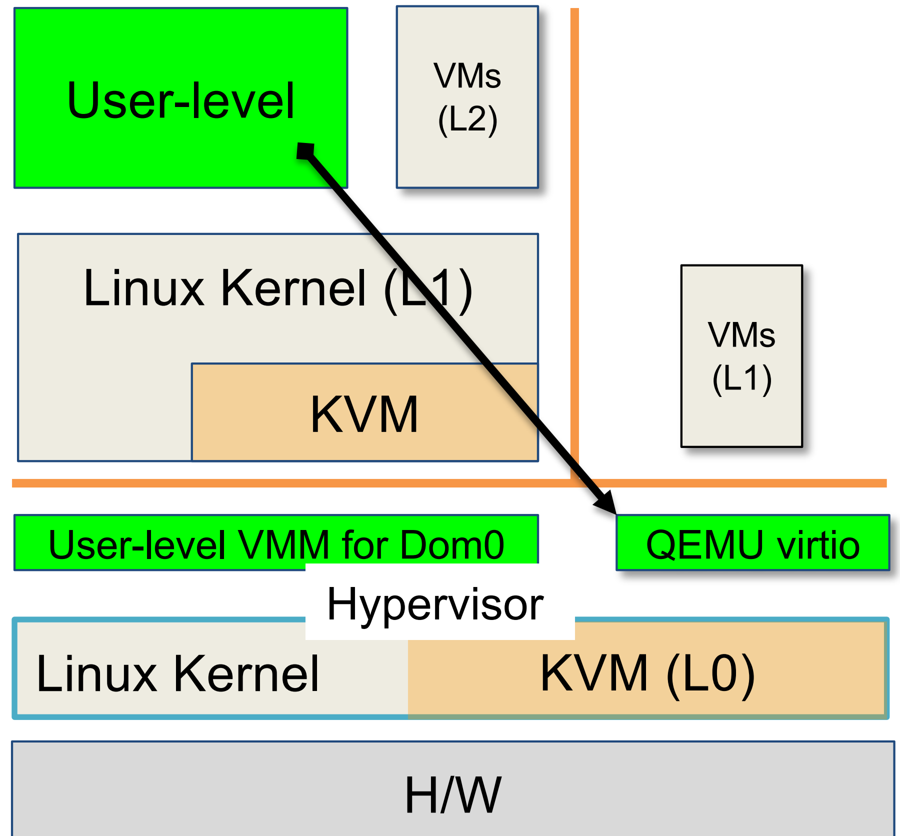
Dom0: Scheduling and PM Issues

- Hypervisor needs to own VM scheduling
  - Intercept `HLT/MWAIT` in Dom0
- Inefficient for clients:
  - Two-level scheduling
    - VM-level and process-level (within VM)
  - Unexpected latencies in VMs, especially Dom0



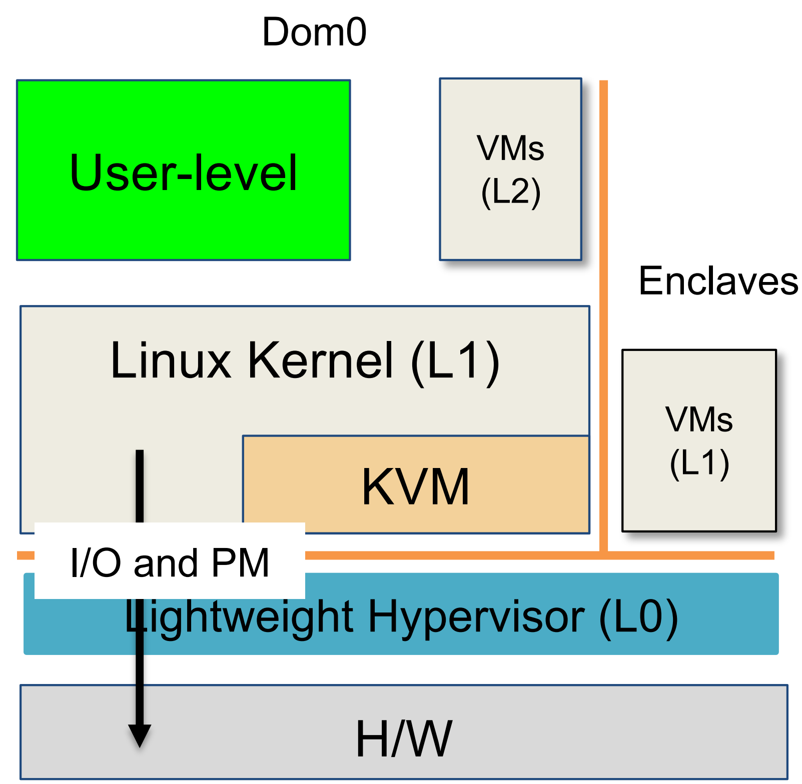
Impacts of Linux/KVM Hypervisor

- How to create VMs?
  - Need to invoke QEMU process on the host from Dom0 (a guest)
    - E.g. Nitro Enclaves driver
- Virtio
  - No I/O devices are available in hypervisor
  - Only memory filesystem



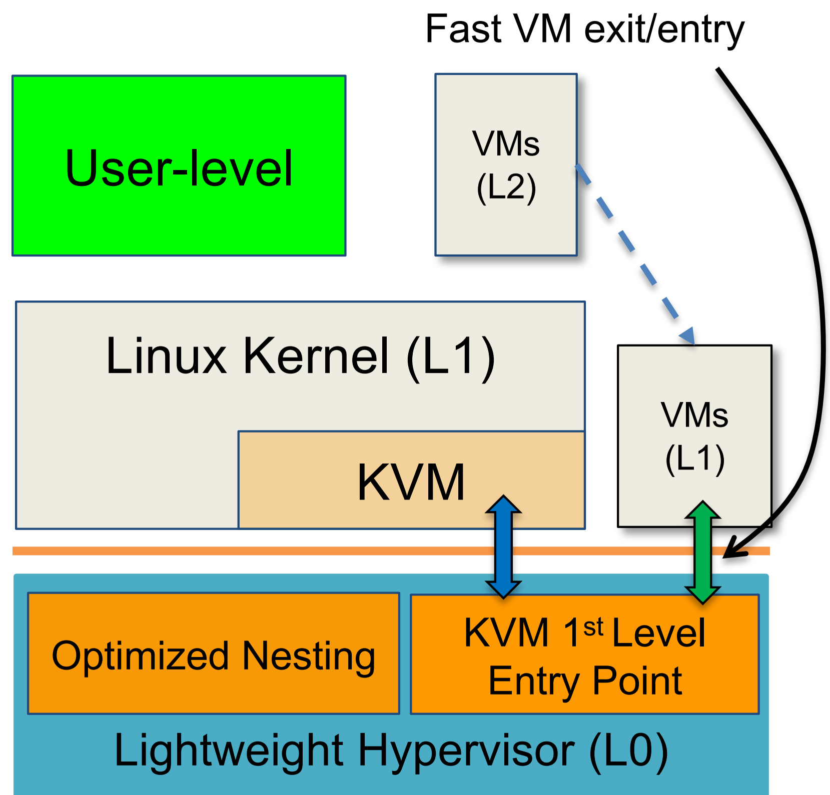
Lightweight Hypervisor

- Pros
  - Same code path of bare-metal Linux/KVM
  - Low latency & overhead
    - No VM exits if dom0 behaves legitimately
  - Small TCB
- Cons
  - Limited L1 VM types
    - E.g. no virtual devices support
  - L2 for Unmodified guests
    - Overhead compared with L1?



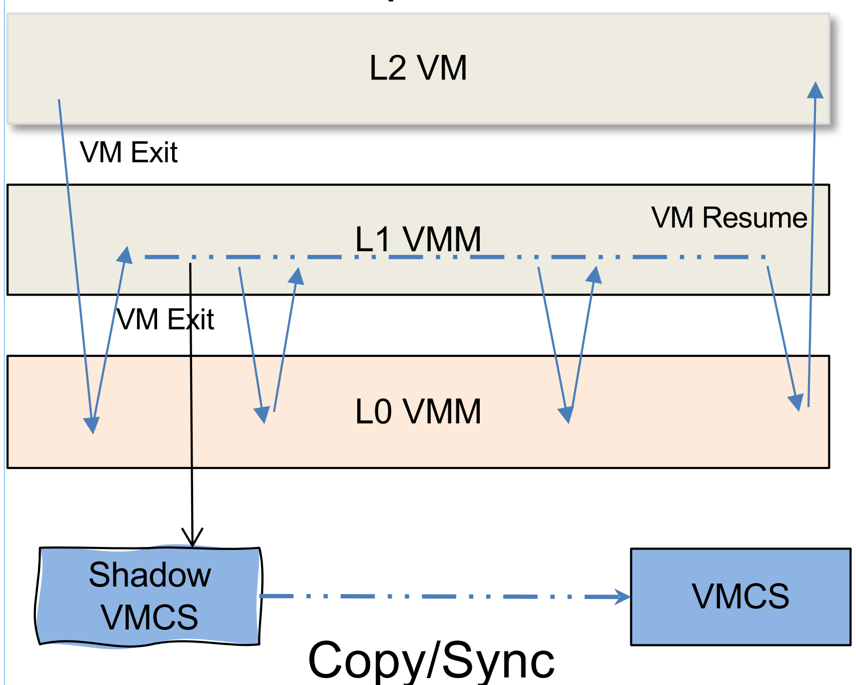
Optimization for KVM Guests

- Optimized nested virtualization using VMCS shadowing
  - Passthrough shadow VMCS (for most fields) in L1
  - Convert shadow VMCS to real VMCS quickly (flip one bit)
- KVM 1st level Entry Point
  - Fast VM entry/exit if exit handling doesn't require Linux services
  - Allow KVM VMs to run as L1

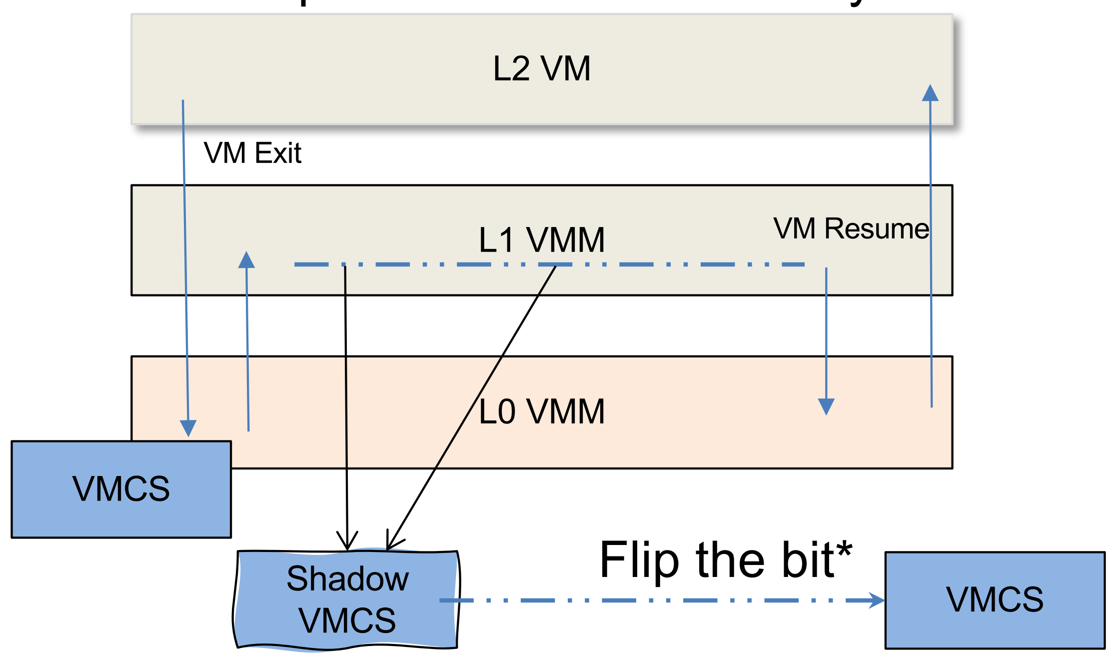


Optimized Nested Virtualization

Current Implementation



Optimized L2 VM exit/entry



*: Bit 31: shadow-VMCS indicator in VMCS region

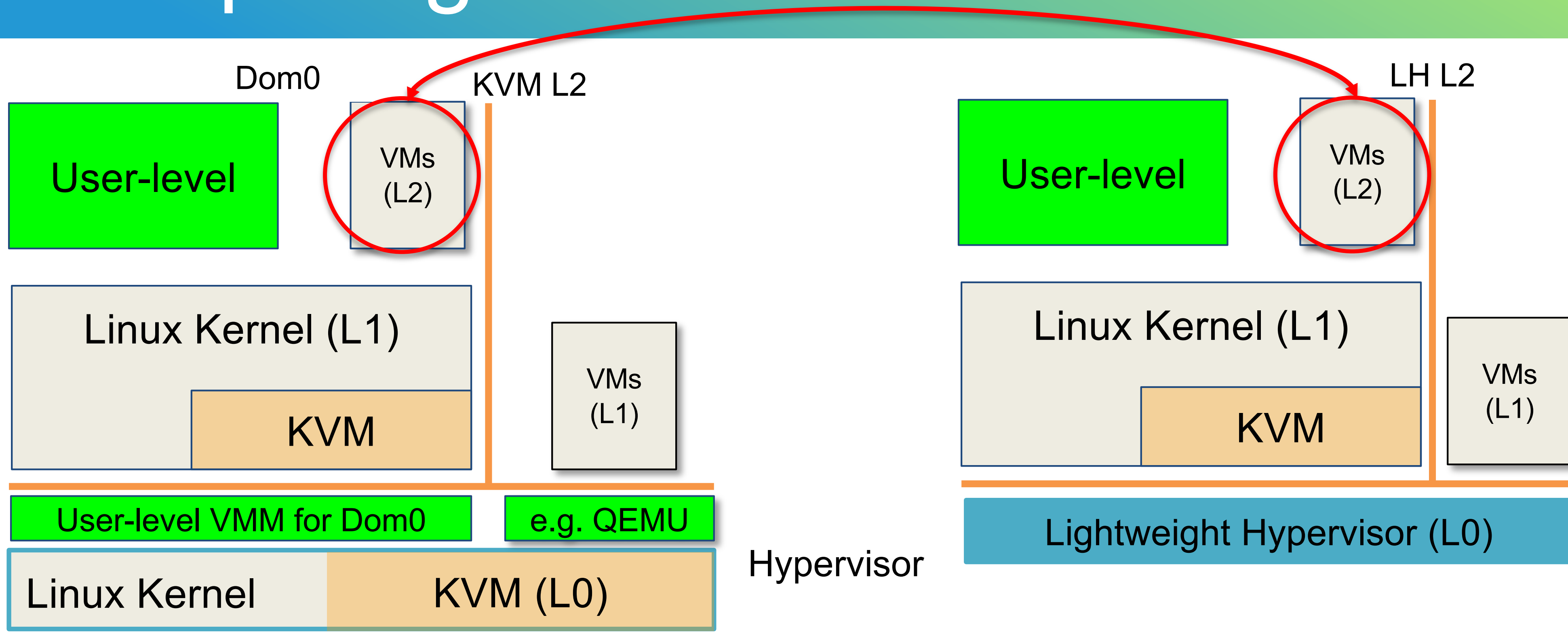
PoC: Lightweight Hypervisor by extending VBH

- Original VBH* (Virtualization Based Hardening)
  - Deprivileges Linux kernel to harden the kernel (Dom0)
    - With all I/O and APIC passthrough
- Added simple nested virtualization to run KVM guests (L2)
  - Only for L1 VM (bare-metal VM, where GPA = HPA)
  - Implemented optimized VMCS shadowing, virtual EPT for isolation
- Added a feature to run a simple L1 VM in TEE
  - E.g. OP-TEE OS**
- Working on virtual IOMMU

*: From presentation KVM Forum 2019:
"Manage Session Virtualization Based Hardening: Securing Container Workloads and Beyond"

**: https://github.com/OP-TEE/optee_os

Comparing Performance 1/2



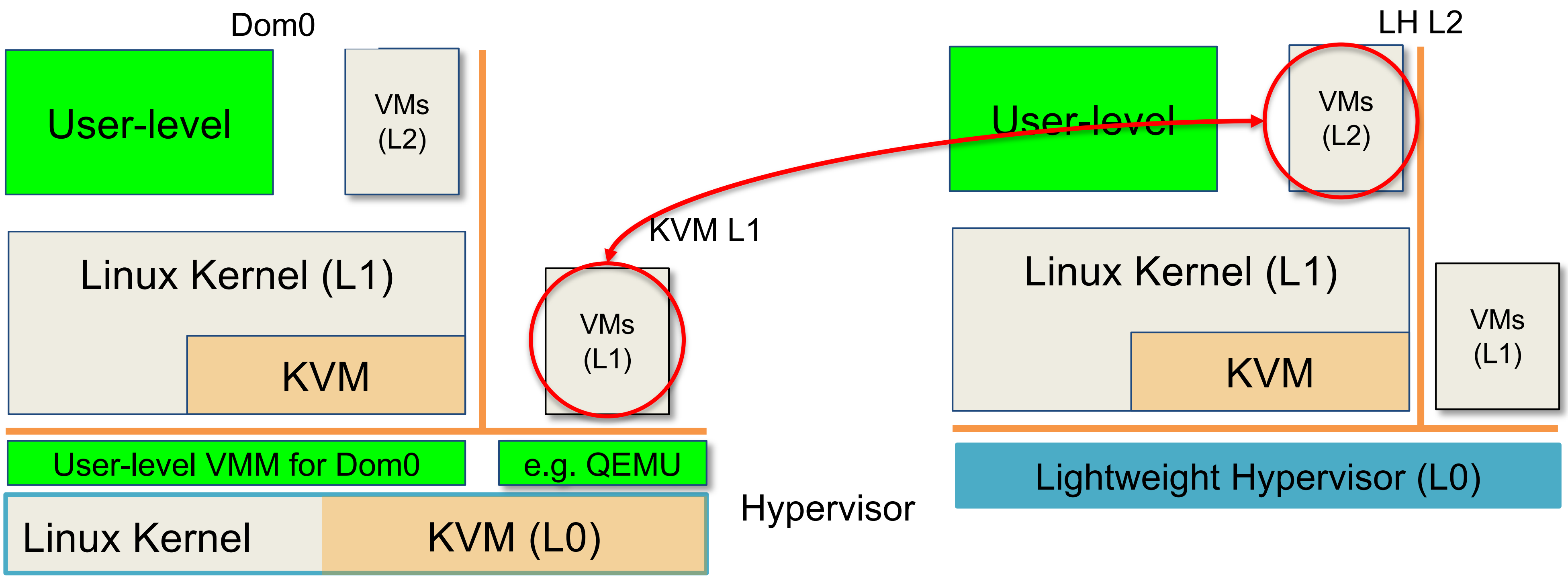
KVM L2 and LH L2

Kernel Build L2 VM Exit Breakdown

Improvements
from flipping shadow-VMCS indicator

	Hypervisor	VM Exit: L2->L1	L1 Handler	VM Entry :L1->L2	Total	Improvement Contribution
External Interrupt	Linux/KVM	27172	147345	12058	186575	Total Improvement:40% L1<->L2 Switch:47% Handler:53%
	LH	**2418**	108332	**1087**	111838	
IO_INSTRUCTION	Linux/KVM	17362	483317	26788	526570	Total Improvement:87% L1<->L2 Switch:9% Handler:91%
	LH	**540**	63745	**925**	65211	
MSR WRITE	Linux/KVM	18175	67198	17674	103047	Total Improvement:77% L1<->L2 Switch:43% Handler:57%
	LH	**956**	21358	**721**	23036	
PREEMPTION_TIMER	Linux/KVM	46058	215206	27750	289015	Total Improvement:91% L1<->L2 Switch:28% Handler:72%
	LH	**1768**	27610	**744**	30123	

Comparing Performance 2/2



Comparing KVM L1 and LH L2 Guest

(without KVM 1st Level Entry Point)

LH L2 and KVM L1 is almost equivalent

Benchmark	KVM L1	LH L2	LH L2 vs. KVM L1
Kernel compiling Unit: second	348.3	353	**98.67%**
Iperf Unit: Gb/sec (Between VM and underlining VMM)	41.2	37.16	**90.19%**
FIO seq read Unit: MB/s	515.8	471.2	**91.35%**
FIO seq write Unit: MB/s	279.2	232.4	**83.24%**
FIO rand read Unit: MB/s	256.8	226.6	**88.24%**
FIO rand write Unit: MB/s	219	182	**83.11%**
Sysbench CPU Unit: events per second	4623.66	4609.03	**99.68%**
Sysbench CPU Unit: MiB/sec	8218.38	8207.89	**99.87%**

Findings from PoCs

- Linux/KVM Hypervisor has structural impacts:
  - Large structural changes to resource management
    - Scheduling, power management, VM management
  - Virtio implementation
  - It would require different efforts to optimize/tune
    - Beyond current Linux/KVM
- Lightweight Hypervisor
  - LH L2 and KVM L1 is almost equivalent
    - I/O needs more optimization

Our conclusion

- Lightweight (reactive) Hypervisor approach is more suitable for the current Linux/KVM to make it more secure (Type 1.5 VMM)
  - Same code path as bare-metal Linux/KVM, including scheduling and power management, etc.
  - Low latency & overhead
- VBH-based Hypervisor can harden Dom0 kernel and guests additionally
- KVM guests run with minimal overhead
- Advantage when implementing TEE because of small TCB

THE LINUX FOUNDATION

Next Step

- Finish VBH-based PoC
  - Complete IOMMU virtualization
    - For direct I/O support in secure environment
  - Optimize KVM guest performance more
    - I/O performance (e.g. write operations)
    - KVM 1st Level Entry Point in VBH
- Share the code
  - github


KVM
FORUM