Department of

Computer Science

Taming Hosted Hypervisors with (Mostly) Deprivileged Execution

Chiachih Wu[†], Zhi Wang[*], Xuxian Jiang[†]

[†]North Carolina State University, [*]Florida State University

NC STATE UNIVERSITY

Virtualization is Widely Used

- □ "There are now hundreds of thousands of companies around the world using AWS to run all their business, or at least a portion of it. They are located across 190 countries, which is just about all of them on Earth."

Werner Vogels, CTO at Amazon

AWS Summit '12

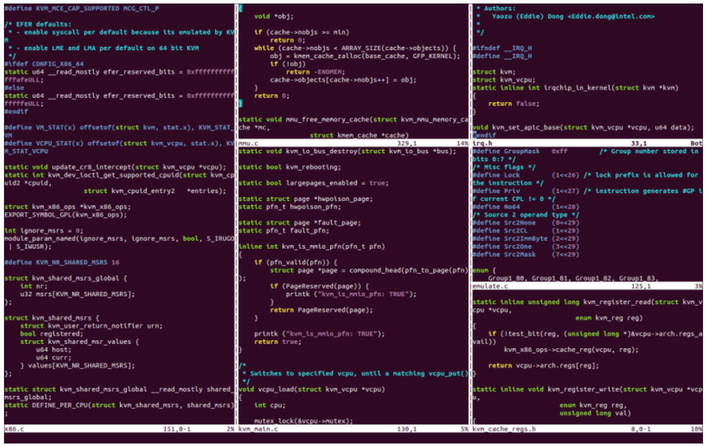
- □ "Virtualization penetration has surpassed 50% of all server workloads, and continues to grow."

Gartner

Magic Quadrant for x86 Server Virtualization Infrastructure
June '12

Threats to Hypervisors

- □ Large Code Bases



```
#define KVM_MCE_CAP_SUPPORTED MCG_CTL_P

/* EFER defaults:
 * - enable syscall per default because its emulated by KV
M
 * - enable LME and LMA per default on 64 bit KVM
 */
#ifdef CONFIG_X86_64
static u64 __read_mostly efer_reserved_bits = 0xffffffffff
fffafeULL;
#else
static u64 __read_mostly efer_reserved_bits = 0xffffffffff
fffffeULL;
#endif

#define VM_STAT(x) offsetof(struct kvm, stat.x), KVM_STAT_
VM
#define VCPU_STAT(x) offsetof(struct kvm_vcpu, stat.x), KV
M_STAT_VCPU

static void update_cr8_intercept(struct kvm_vcpu *vcpu);
static int kvm_dev_ioctl_get_supported_cpuid(struct kvm_cp
uid2 *cpuid,
                    struct kvm_cpuid_entry2   *entries);

struct kvm_x86_ops *kvm_x86_ops;
EXPORT_SYMBOL_GPL(kvm_x86_ops);

int ignore_msrs = 0;
module_param_named(ignore_msrs, ignore_msrs, bool, S_IRUGO
 | S_IWUSR);

#define KVM_NR_SHARED_MSRS 16

struct kvm_shared_msrs_global {
    int nr;
    u32 msrs[KVM_NR_SHARED_MSRS];
};

struct kvm_shared_msrs {
    struct kvm_user_return_notifier urn;
    bool registered;
    struct kvm_shared_msr_values {
        u64 host;
        u64 curr;
    } values[KVM_NR_SHARED_MSRS];
};

static struct kvm_shared_msrs_global __read_mostly shared_
msrs_global;
static DEFINE_PER_CPU(struct kvm_shared_msrs, shared_msrs)
;
x86.c                                        151,0-1        2%
```

```
{
    void *obj;

    if (cache->nobjs >= min)
        return 0;
    while (cache->nobjs < ARRAY_SIZE(cache->objects)) {
        obj = kmem_cache_zalloc(base_cache, GFP_KERNEL);
        if (!obj)
            return -ENOMEM;
        cache->objects[cache->nobjs++] = obj;
    }
    return 0;
}

static void mmu_free_memory_cache(struct kvm_mmu_memory_ca
che *mc,
                  struct kmem_cache *cache)
mmu.c                                        329,1         14%
```

```
static void kvm_io_bus_destroy(struct kvm_io_bus *bus);

static bool kvm_rebooting;

static bool largepages_enabled = true;

static struct page *hwpoison_page;
static pfn_t hwpoison_pfn;

static struct page *fault_page;
static pfn_t fault_pfn;

inline int kvm_is_mmio_pfn(pfn_t pfn)
{
    if (pfn_valid(pfn)) {
        struct page *page = compound_head(pfn_to_page(pfn)
);
        if (PageReserved(page)) {
            printk ("kvm_is_mmio_pfn: TRUE");
        }
        return PageReserved(page);
    }

    printk ("kvm_is_mmio_pfn: TRUE");
    return true;
}

/*
 * Switches to specified vcpu, until a matching vcpu_put()
 */
void vcpu_load(struct kvm_vcpu *vcpu)
{
    int cpu;

    mutex_lock(&vcpu->mutex);
kvm_main.c                                   130,1          5%
```

```
 * Authors:
 *   Yaozu (Eddie) Dong <Eddie.dong@intel.com>
 *
 */

#ifndef __IRQ_H
#define __IRQ_H

struct kvm;
struct kvm_vcpu;
static inline int irqchip_in_kernel(struct kvm *kvm)
{
    return false;
}

void kvm_set_apic_base(struct kvm_vcpu *vcpu, u64 data);
#endif
irq.h                                        33,1          Bot
```

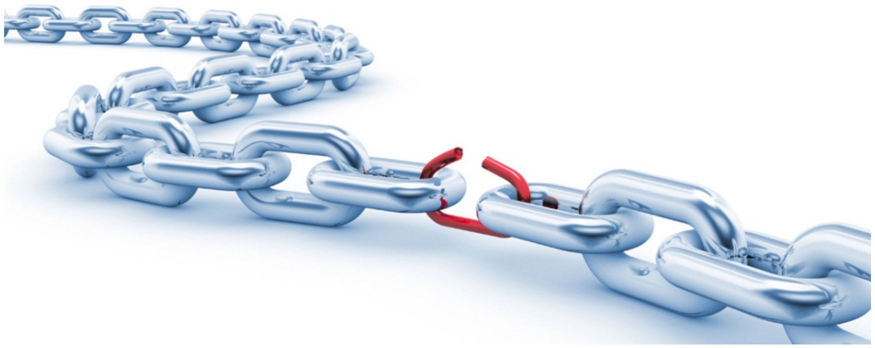
```
#define GroupMask   0xff        /* Group number stored in
bits 0:7 */
/* Misc flags */
#define Lock        (1<<26) /* lock prefix is allowed for
the instruction */
#define Priv        (1<<27) /* instruction generates #GP i
f current CPL != 0 */
#define No64        (1<<28)
/* Source 2 operand type */
#define Src2None    (0<<29)
#define Src2CL      (1<<29)
#define Src2ImmByte (2<<29)
#define Src2One     (3<<29)
#define Src2Mask    (7<<29)

enum {
    Group1_80, Group1_81, Group1_82, Group1_83,
emulate.c                                    125,1          3%
```

```

static inline unsigned long kvm_register_read(struct kvm_v
cpu *vcpu,
                          enum kvm_reg reg)
{
    if (!test_bit(reg, (unsigned long *)&vcpu->arch.regs_a
vail))
        kvm_x86_ops->cache_reg(vcpu, reg);

    return vcpu->arch.regs[reg];
}

static inline void kvm_register_write(struct kvm_vcpu *vcp
u,
                      enum kvm_reg reg,
                      unsigned long val)
{
kvm_cache_regs.h                             8,0-1         10%
```

Hypervisor	SLOC
Xen (4.0)	194K
VMware ESXi[1]	200K
Hyper-V[1]	100K
KVM (2.6.32.28)	33.6K

[1]: Data source: NOVA (Steinberg *et al.*, EuroSys ’10)

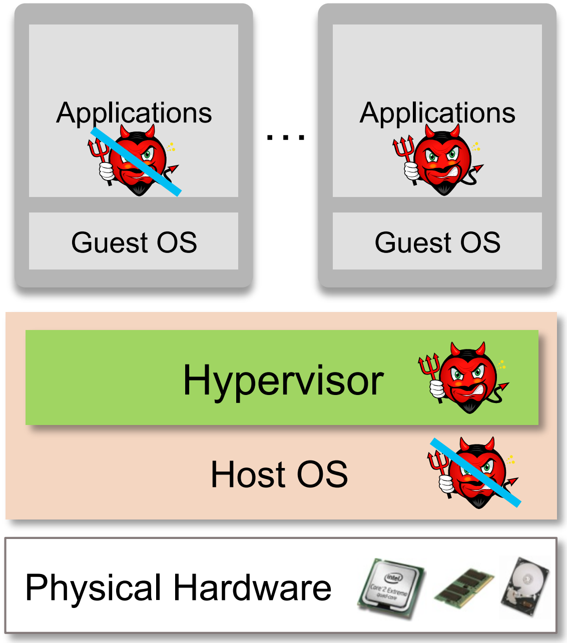
- □ Vulnerabilities



Hypervisor	Vulnerabilities
Xen	41
KVM	24
VMware ESXi	43
VMware Workstation	49

Data source: National Vulnerability Database (‘09~’12)

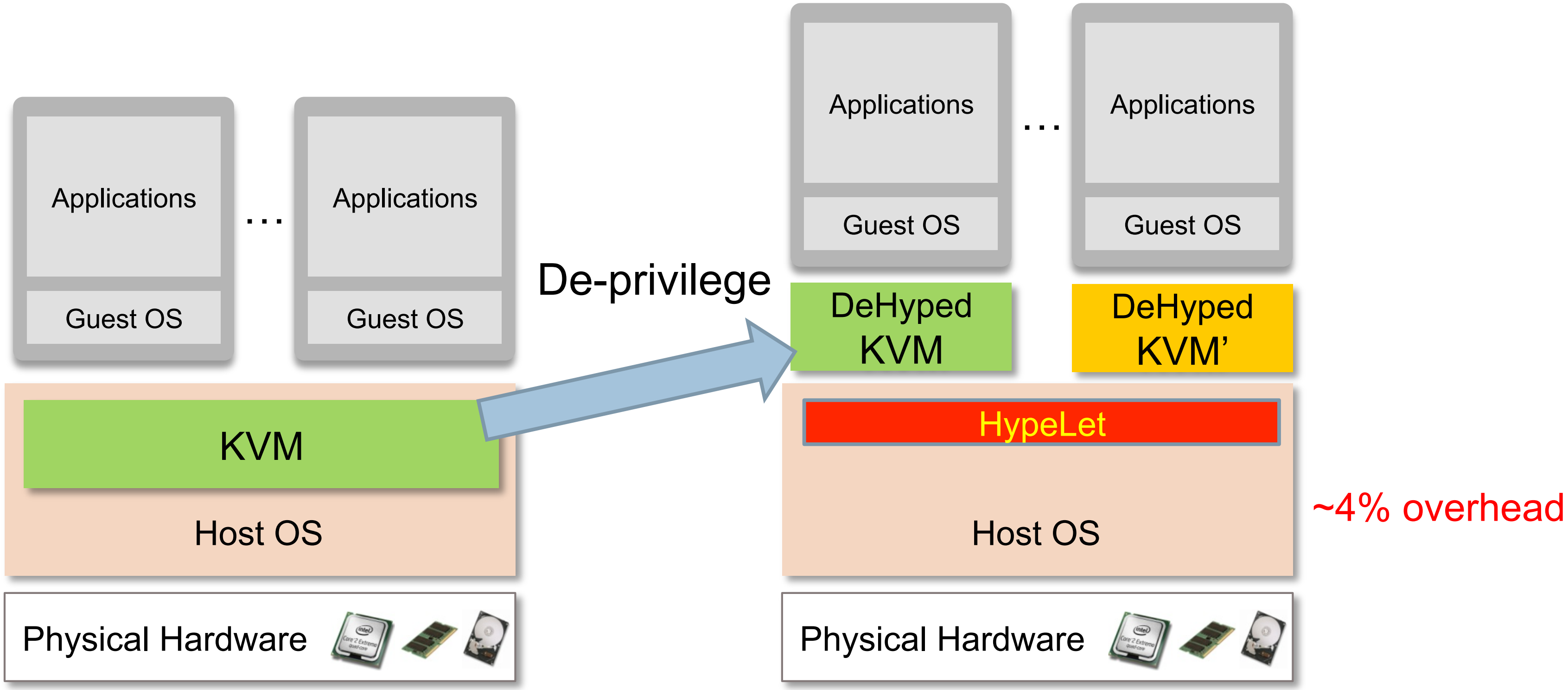
Threats to Hosted Hypervisors



Can we prevent the compromised hypervisor from attacking the rest of the system?

DeHype

- Decomposing the KVM hypervisor codebase
  - De-privileged part → user-level (93.2% codebase)
  - Privileged part → small kernel module (2.3 KSLOC)

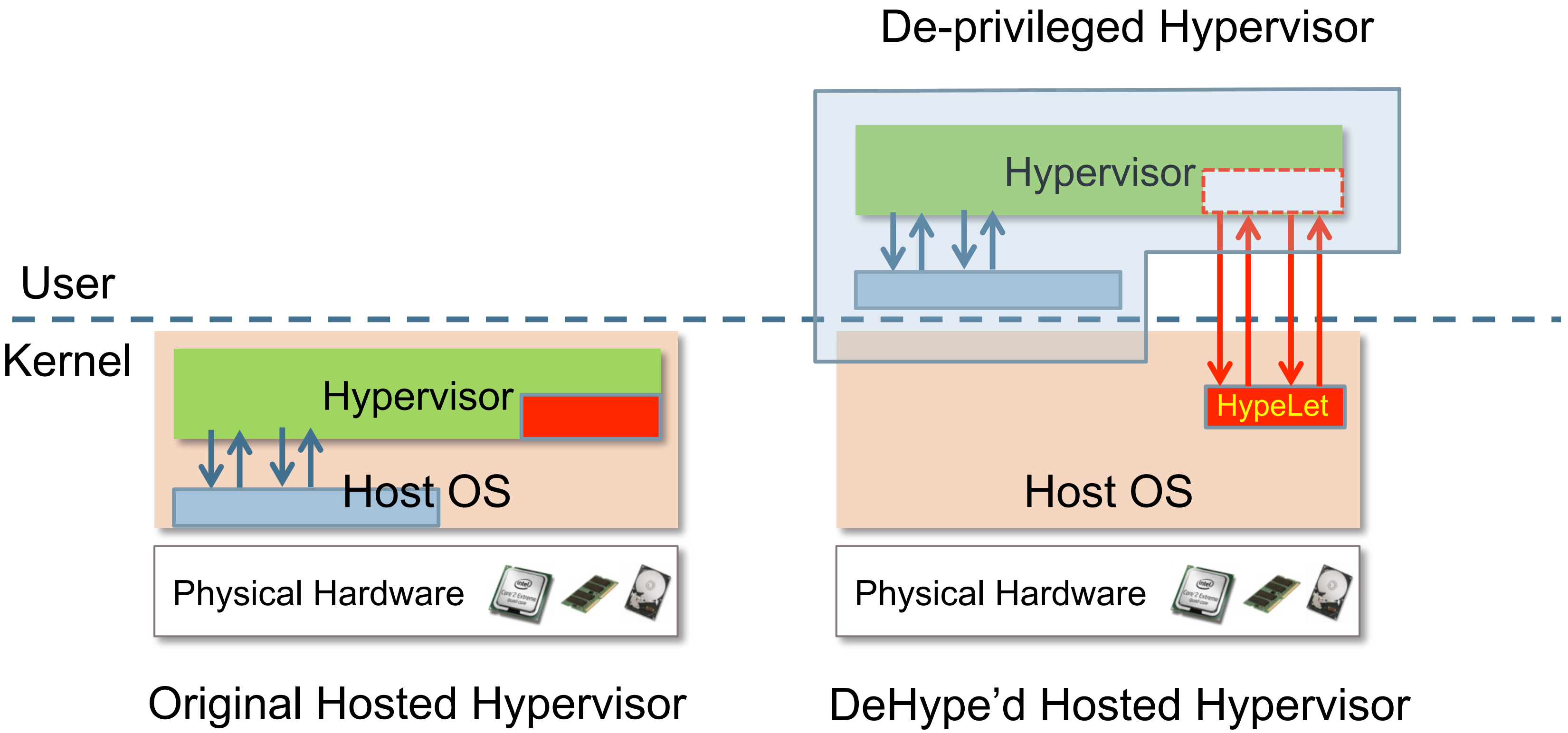


Challenges

- Providing the OS services in user mode
- Minimizing performance overhead
- Supporting hardware-assisted memory virtualization at user-level

Challenge I

- Providing the OS services in user mode



Dependency Decoupling

- Abstracting the host OS interface and providing OS functionalities in user mode

- For example
  - Memory allocator: kmalloc/kfree, alloc_page, etc.
  - Kernel APIs for memory access: virt_to_page, etc.
  - Scheduling, signal handling, invoking system calls
    - Leveraging GLIBC

Dependency Decoupling

10 privileged services provided by HypeLet

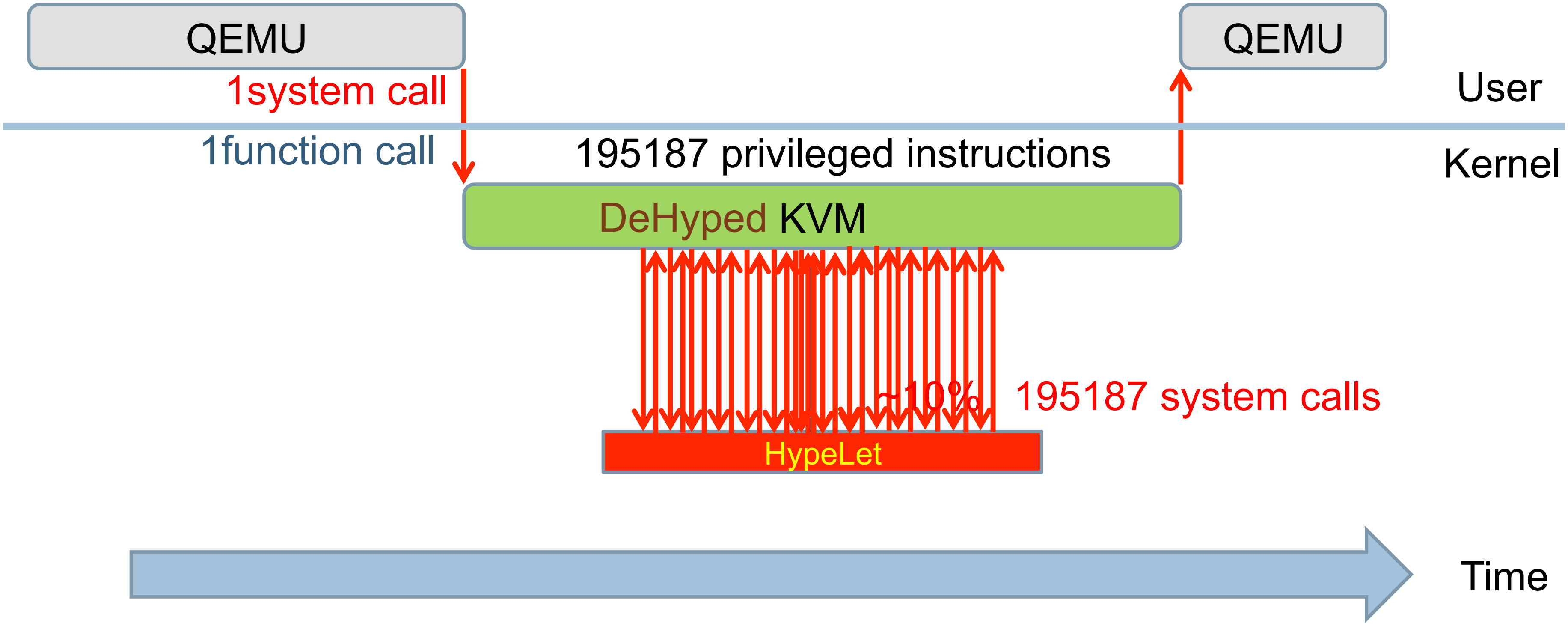
Name	Function
VMREAD	Read VMCS fields
VMWRITE	Write VMCS fields
GUEST_RUN	Perform host-to-guest world switches
GUEST_RUN_POST	Perform guest-to-host world switches
RDMSR	Read MSR registers
WRMSR	Write MSR registers
INVVPID	Invalidate TLB mappings based on VPID
INVEPT	Invalidate EPT mappings
INIT_VCPU	Initialize vCPU
MAP_HVA_TO_PFH	Translate host virtual address to physical frame

Privileged instrustions

Service routines

Challenge II

- Minimizing performance overhead



Optimization: Caching VMCS

- VMCS (Virtual Machine Control Structure)
  - ~90% of the privileged instructions issued by the hypervisor are for accessing VMCS
  - Accessed by the hypervisor for monitoring or controlling the behavior of the guest VM
  - Indirectly affected by the guest VM throughout the running period in guest mode

Optimization: Caching VMCS

- Maintaining cached copy of VMCS in user-level
- Caching only the most frequently accessed fields

- Caching 8 VMWRITE’d fields: 98.28% VMWRITE system calls reduced

Top 8 Most Frequently VMWRITE’d VMCS Fields			
CPU_BASED_VM_EXEC_CONTROL	EPT_POINTER_HIGH	EPT_POINTER	GUEST_RIP
VM_ENTRY_INTR_INFO_FIELD	GUEST_RFLAGS	GUEST_CR3	GUEST_RSP

- Caching 28 VMREAD’d fields: 99.86% VMREAD system calls reduced

Challenge III

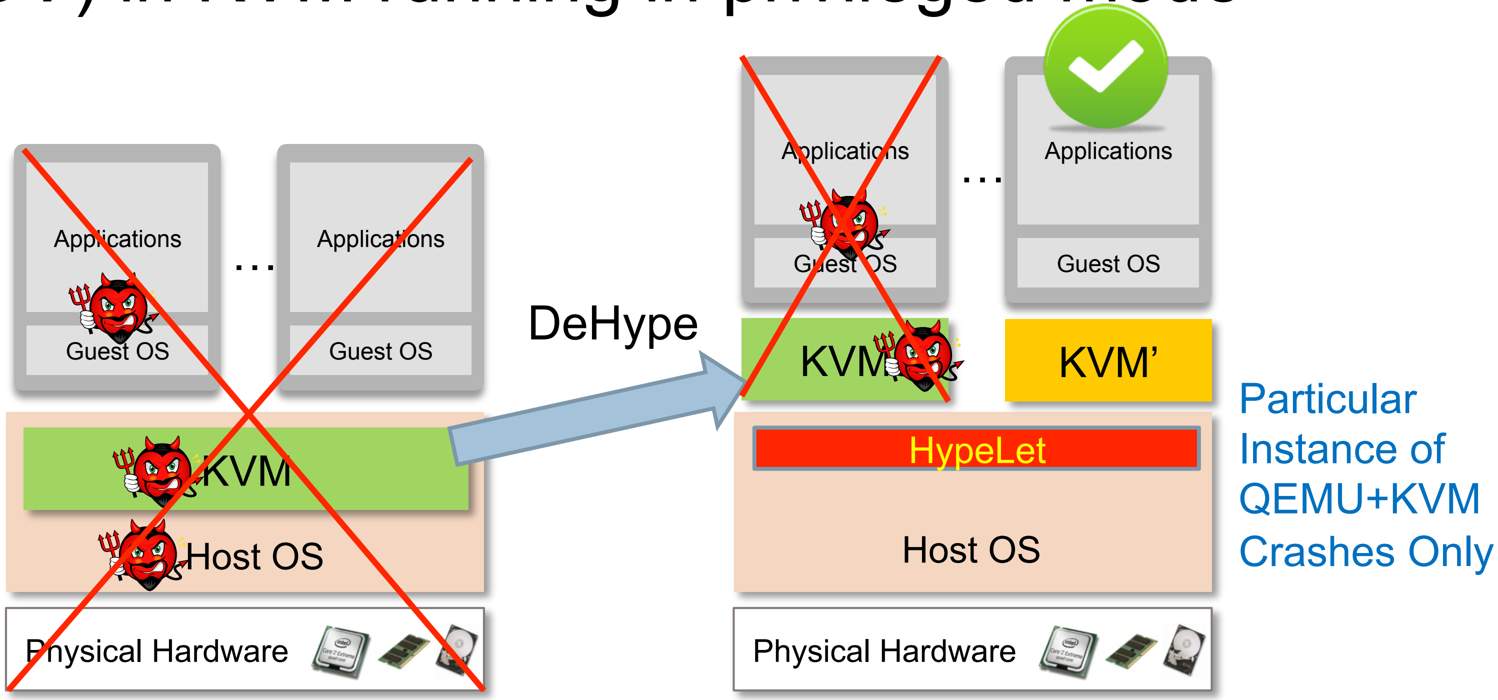
- Supporting hardware-assisted memory virtualization at user-level
  - Maintaining nested page tables which translate guest-physical to host-physical addresses
    - Memory may be paged out
    - Virtual-physical mapping information is unknown
  - Preventing the untrusted hypervisors from accessing memory areas not belonged to them
    - Bactch-processing NPT updates with sanity checks in HypeLet

Implementation and Evaluation

- Prototype
  - KVM 2.6.32.28 with qemu-kvm-0.14.0
  - ~93.2% of KVM codebase is de-privileged
  - 2.3K SLOC small kernel module (HypeLet)
- Evaluation
  - Security benefits
  - Non-security benefits
  - Performance

Testing real-world vulnerabilities

- CVE-2010-0435
  - Guest OS causing a NULL pointer dereference (*accessing debug registers with MOV*) in KVM running in privileged mode

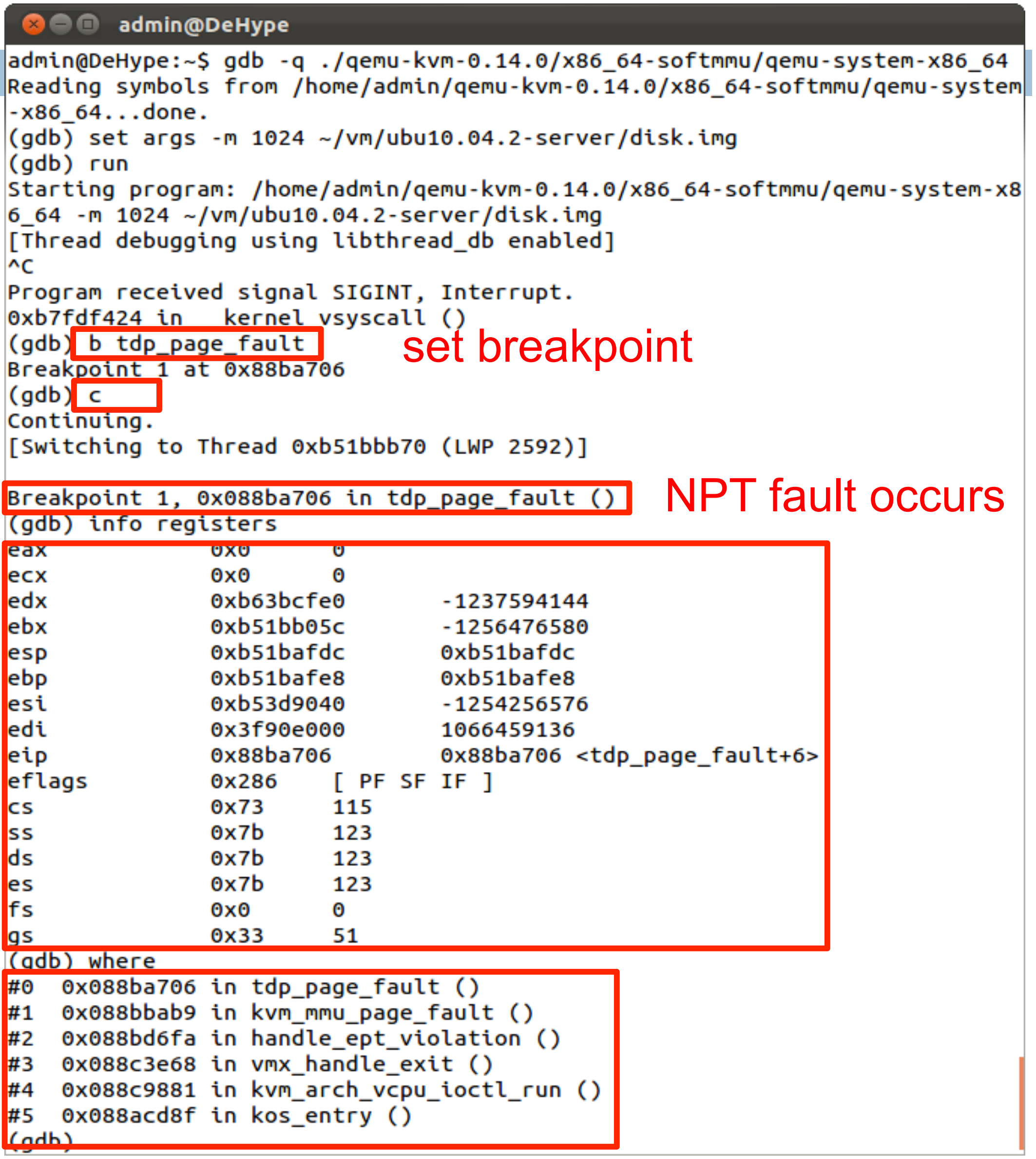


Whole System Crashes

Particular Instance of QEMU+KVM Crashes Only

Facilitating hypervisor development

- e.g., debugging the NPT fault handler with GDB



```
admin@DeHype:~$ gdb -q ./qemu-kvm-0.14.0/x86_64-softmmu/qemu-system-x86_64
Reading symbols from /home/admin/qemu-kvm-0.14.0/x86_64-softmmu/qemu-system
-x86_64...done.
(gdb) set args -m 1024 ~/vm/ubu10.04.2-server/disk.img
(gdb) run
Starting program: /home/admin/qemu-kvm-0.14.0/x86_64-softmmu/qemu-system-x8
6_64 -m 1024 ~/vm/ubu10.04.2-server/disk.img
[Thread debugging using libthread_db enabled]
^C
Program received signal SIGINT, Interrupt.
0xb7fdf424 in __kernel_vsyscall ()
(gdb) b tdp_page_fault
Breakpoint 1 at 0x88ba706
(gdb) c
Continuing.
[Switching to Thread 0xb51bbb70 (LWP 2592)]

Breakpoint 1, 0x088ba706 in tdp_page_fault ()
(gdb) info registers
eax            0x0      0
ecx            0x0      0
edx            0xb63bcfe0       -1237594144
ebx            0xb51bb05c       -1256476580
esp            0xb51bafdc       0xb51bafdc
ebp            0xb51bafe8       0xb51bafe8
esi            0xb53d9040       -1254256576
edi            0x3f90e000       1066459136
eip            0x88ba706        0x88ba706 <tdp_page_fault+6>
eflags         0x286    [ PF SF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb) where
#0  0x088ba706 in tdp_page_fault ()
#1  0x088bbab9 in kvm_mmu_page_fault ()
#2  0x088bd6fa in handle_ept_violation ()
#3  0x088c3e68 in vmx_handle_exit ()
#4  0x088c9881 in kvm_arch_vcpu_ioctl_run ()
#5  0x088acd8f in kos_entry ()
(gdb)
```

set breakpoint

continue the program

NPT fault occurs

register dump

call trace

Running multiple hypervisors

- Running each hypervisor in a different security level
  - Suspicious guests: running on VMI-enabled hypervisors
  - Others: running on normal hypervisors

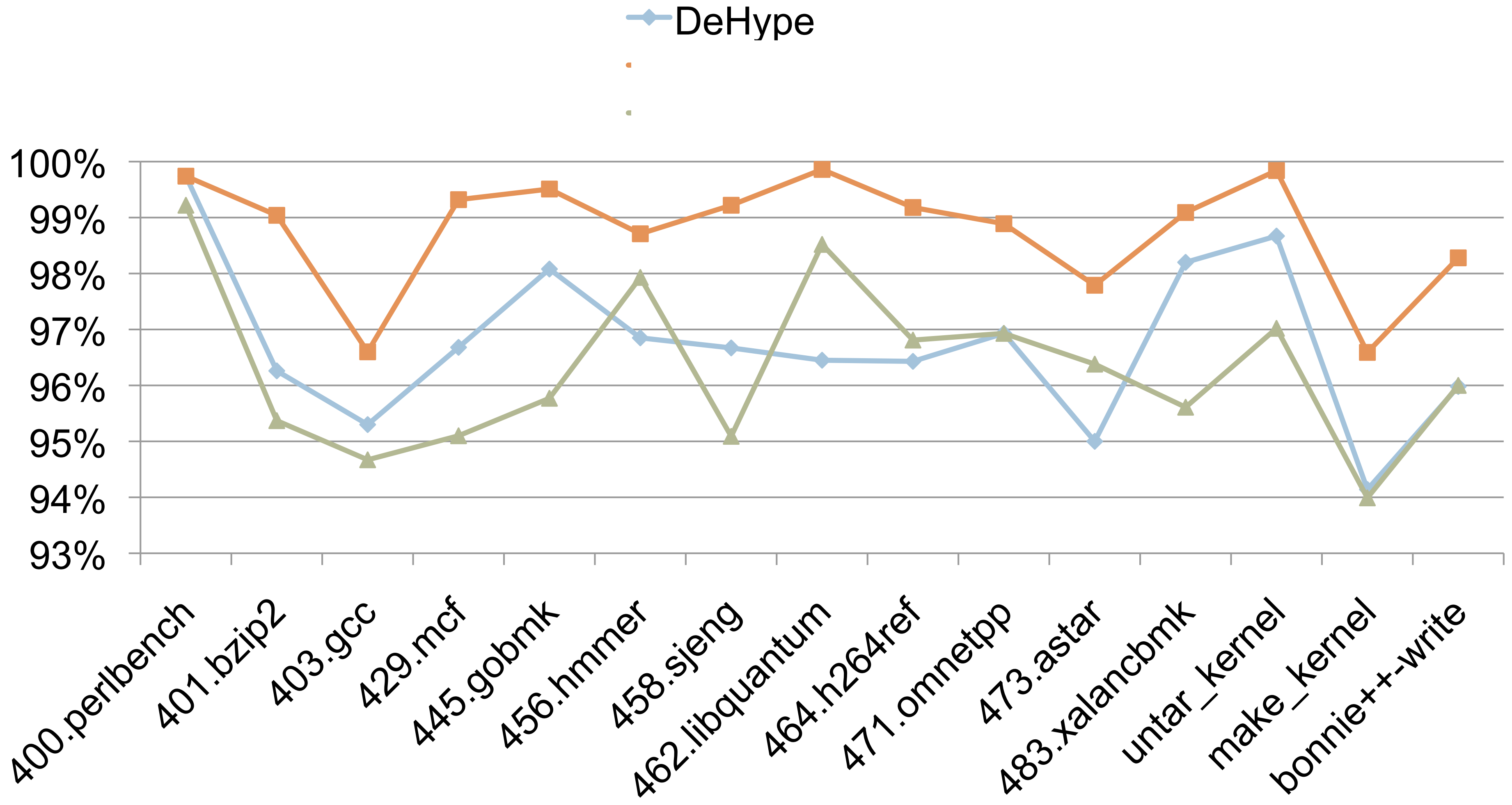
- Live-migrating guests to another hypervisor in the same host computer
  1. New vulnerability reported and fixed
  2. Starting a patched hypervisor
  3. Live-migrating all guests one-by-one

Performance Evaluation

- Test platform
  - Dell OptiPlex 980: Intel Core i7 860 + 3G RAM
  - Host: Ubuntu 11.10 desktop + Linux kernel 2.6.32.28
  - Guests: Ubuntu 10.04.2 LTS server
- Benchmarks

Software Package	Version	Configuration
SPEC CPU2006	v1.0.1	Reportable int
Bonnie++	1.03e	bonnie++ -f -n 256
Linux kernel	2.6.39.2	untar_kernel: tar zfx <KERNEL-TARBALL> make_kernel: make defconfig vmlinux

Relative Performance



Discussion

- HypeLet and the host OS are a part of the TCB
  - HypeLet is the main attack surface in the cloud environment
  - HypeLet is highly constrained (2.3 KSLOC, 10 services)

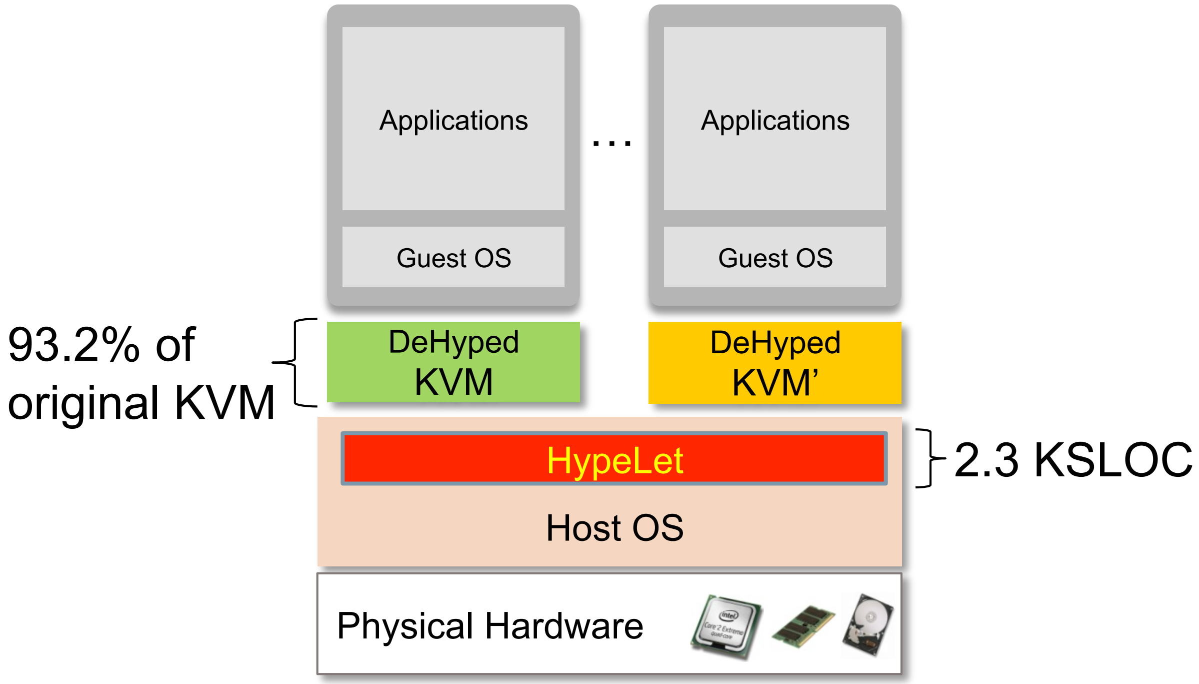
- Prototype limitations
  - Pinning guest memory
    - Could be extended with Linux MMU notifier
  - Not supporting all KVM features
    - SMP
    - Para-virtualized I/O

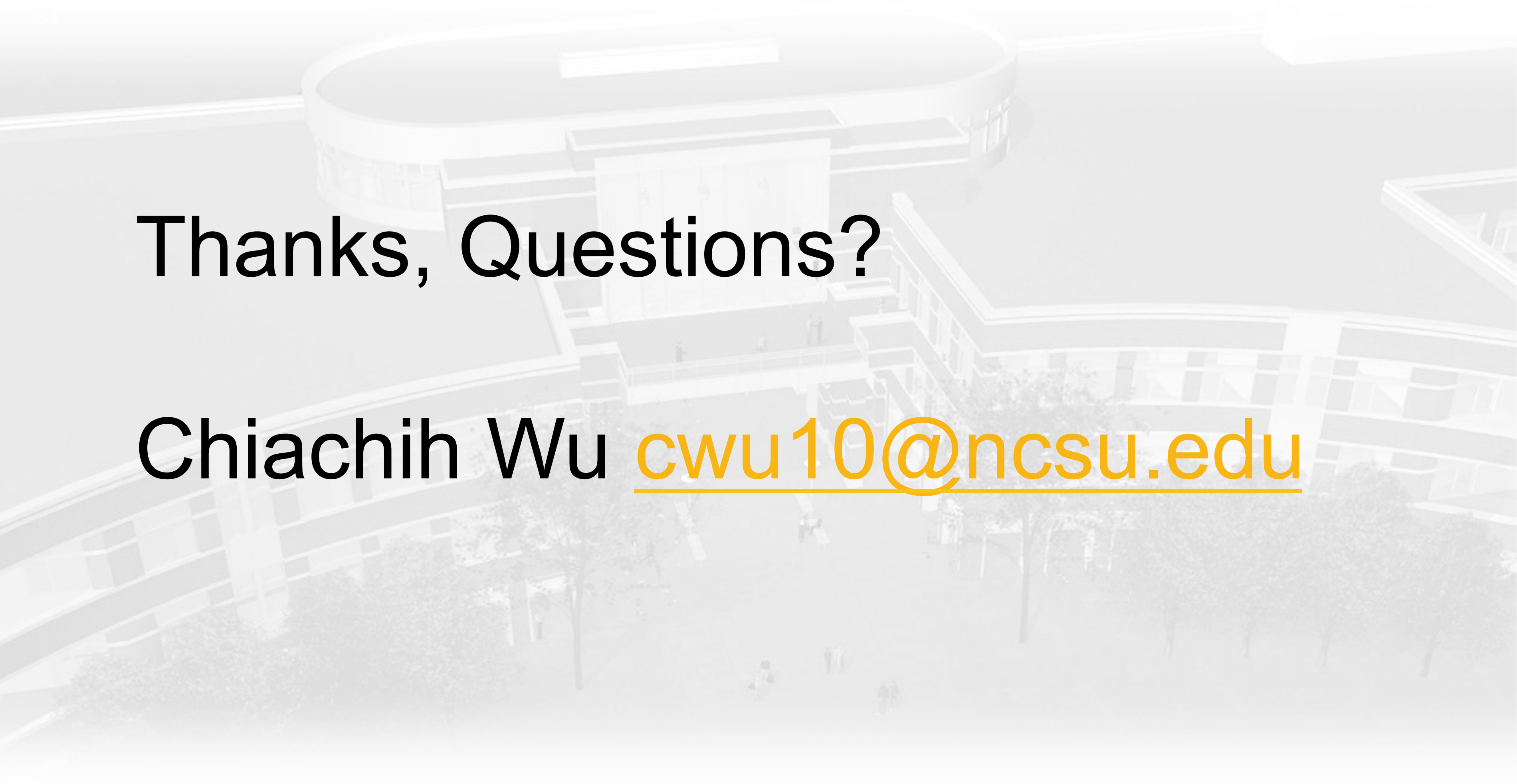
Related Work

- Improving hypervisor security
  - seL4 (Klein *et al.*, SOSP '09), NOVA (Steinberg *et al.*, EuroSys '10), HyperLock (Wang *et al.*, EuroSys '12) …
- Isolating untrusted device drivers
  - Nooks (Swift *et al.*, SOSP '03), Microdrivers (Ganapathy *et al.*, ASPLOS '08) …
- Applying virtualization to host security
  - HookSafe (Wang *et al.*, CCS '09), Lockdown (Vasudevan *et al.*, TRUST '12) …

Conclusion

- DeHype substantially reduces hosted hypervisor's attack surface and brings additional benefits
  - Better development and debugging
  - Concurrent execution of multiple hypervisors



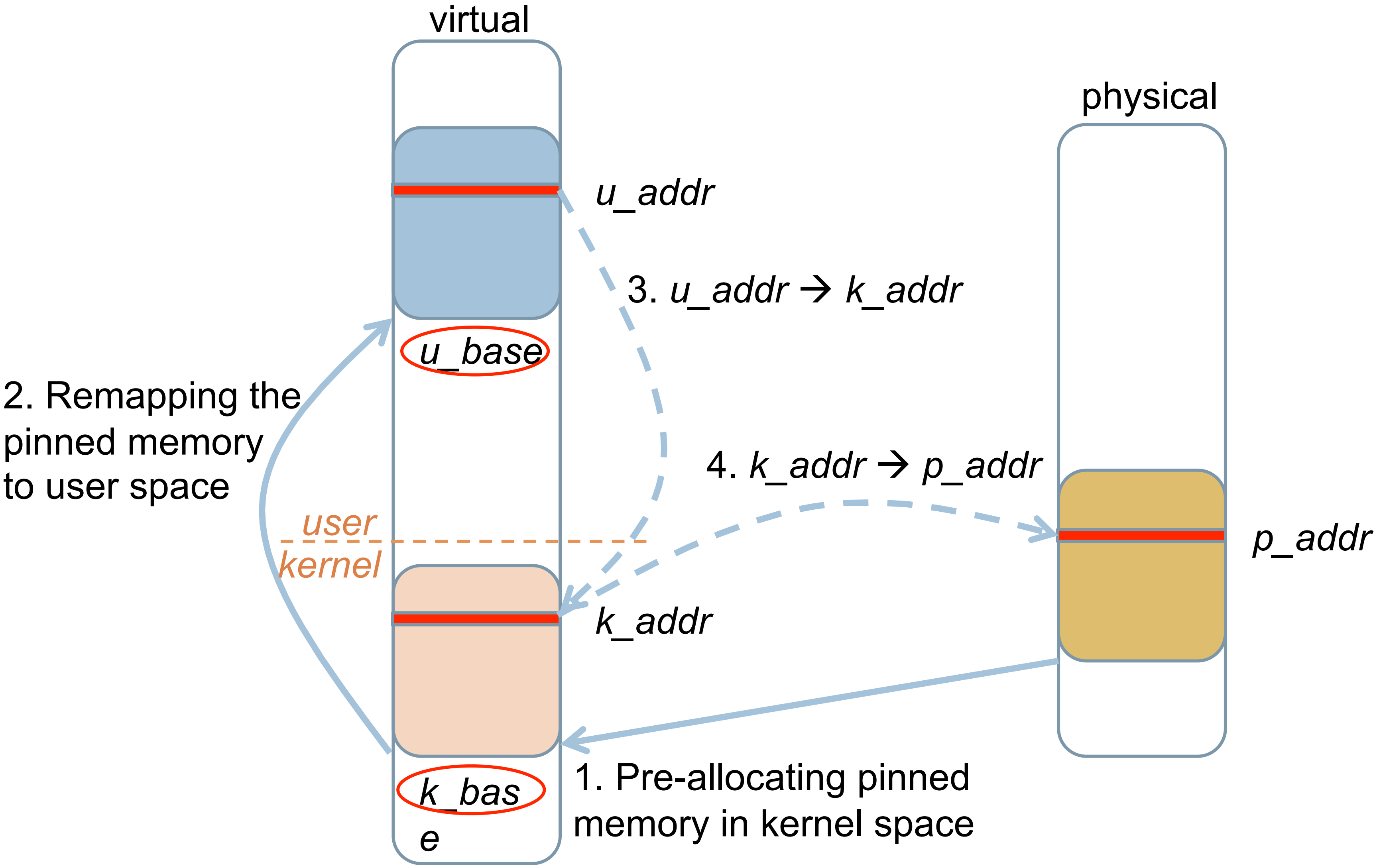


Thanks, Questions?

Chiachih Wu cwu10@ncsu.edu

Backup Slides

Memory Rebasing



1. Pre-allocating pinned memory in kernel space

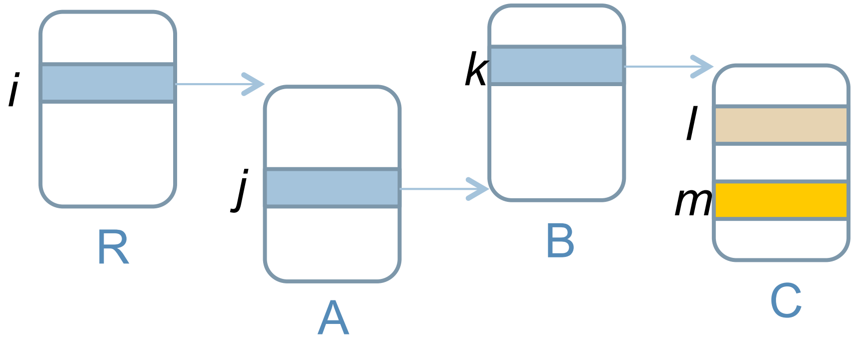
2. Remapping the pinned memory to user space

3. *u_addr* → *k_addr*

4. *k_addr* → *p_addr*

Securely Update NPT Entries

- Preventing the untrusted hypervisor from updating the NPT tables directly
  - Recording the update operations into buffer
  - Batch-processing the updates in next host-to-guest switch with sanity check (by HypeLet)
  - Issue: the hypervisor needs the actual NPTs to traverse the layer-based NPTs



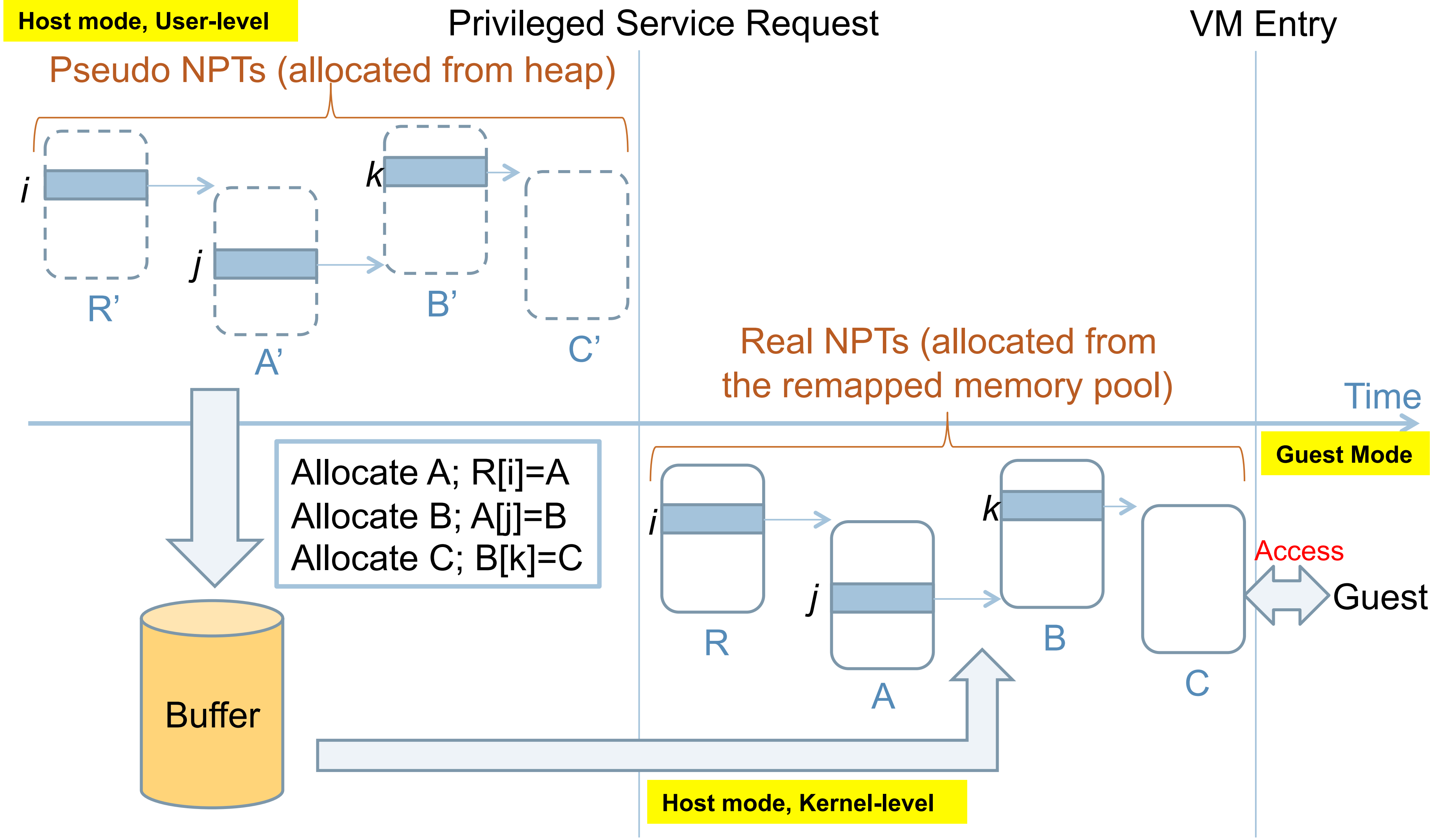
Update entry l
1. Allocate A; R[i]=A
2. Allocate B; A[j]=B
3. Allocate C; B[k]=C
4. Update C[l]

Recording only

Update entry m
1. A=R[i]
2. B=A[j]
3. C=B[k]
4. Update C[m]

Cannot traverse

Pseudo NPT



Host mode, User-level

Privileged Service Request

VM Entry

Pseudo NPTs (allocated from heap)

Allocate A; R[i]=A
Allocate B; A[j]=B
Allocate C; B[k]=C

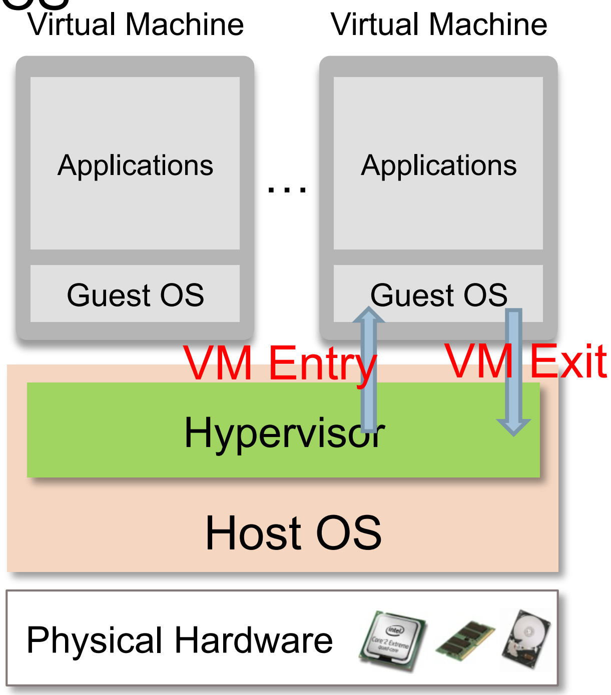
Real NPTs (allocated from the remapped memory pool)

Guest Mode

Host mode, Kernel-level

Intel VT-x: World Switches

- VM Entry
  - Transition from VMM to Guest (VMLAUNCH/VMRESUME)
  - Enters VMX non-root operation (guest mode)
  - Saves VMM state in VMCS
  - Loads Guest state and exit criteria from VMCS
- VM Exit
  - Transition from Guest to VMM (VMEXIT)
  - Enters VMX root operation (host mode)
  - Saves Guest state in VMCS
  - Loads VMM state from VMCS

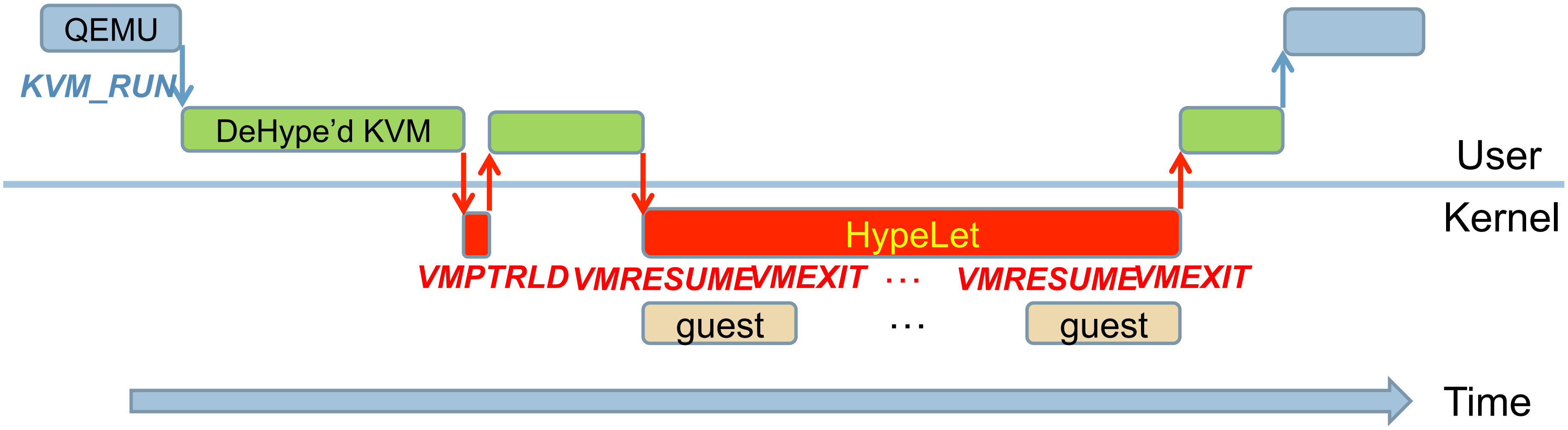


Optimization: Caching VMCS

Top 28 Most Frequently VMREAD'ed VMCS Fields			
GUEST_INTERRUPTIBILITY_INFO	EXIT_QUALIFICATION	GUEST_CS_BASE	GUEST_RSP
IDT_VECTORING_INO_FIELD	GUEST_CS_SELECTOR	GUEST_DS_BASE	GUEST_RIP
GUEST_PHYSICAL_ADDRESS_HIGH	GUEST_CS_AR_BYTES	GUEST_ES_BASE	GUEST_CR0
GUEST_PHYSICAL_ADDRESS	GUEST_PDPTR0_HIGH	GUEST_PDPTR0	GUEST_CR3
VM_EXIT_INTR_INFO	GUEST_PDPTR1_HIGH	GUEST_PDPTR1	GUEST_CR4
VM_EXIT_INSTRUCTION_LEN	GUEST_PDPTR2_HIGH	GUEST_PDPTR2	GUEST_RFLAGS
CPU_BASED_VM_EXEC_CONTROL	GUEST_PDPTR3_HIGH	GUEST_PDPTR3	VM_EXIT_REASON

Combining privileged instructions

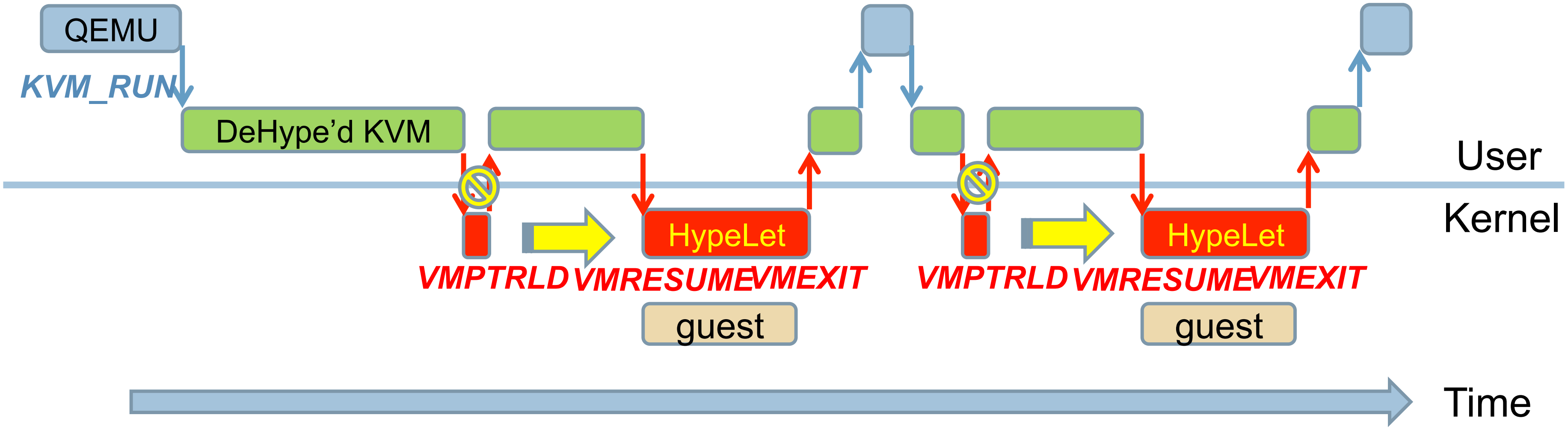
- VMPTRLD: a privileged instruction to load guest states before switching to guest mode



- CPU intensive workload
  - KVM handles most VM Exits
  - One VMPTRLD is followed by multiple runs of (VMRESUME, VMEXIT)
  - The latency of VMPTRLD is not significant

Combining privileged instructions

- IO intensive workload
  - QEMU handles most VM exits for issuing IO instructions
  - One VMPTRLD is followed by one run of (VMRESUME, VMEXIT)
  - VMPTRLD introduces significant latency



  - Postponing the VMPTRLD instruction until the first VMRESUME instruction

Testing real-world vulnerabilities

- CVE-2009-4031
  - KVM attempting to interpret wrong-size (too long) instructions
  - Being exploited
    - Causing large latencies in non-preempt hosts
  - With DeHype
    - Instruction emulation is done in user-level where preemption is natively enabled

Testing real-world vulnerabilities

- CVE-2010-3881
  - KVM copying certain data structures to user program without clearing the padding
  - Being exploited
    - QEMU processes potentially obtaining sensitive information from kernel stack
  - With DeHype
    - QEMU process obtaining information from the stack of the hypervisor paired with it, not from the kernel stack