KVM Address Space Isolation (ASI)

Alexandre Chartre – Oracle

Ofir Weisse, Junaid Shahid, Oleg Rombakh, and Paul Turner – Google

KVM Summit – October 2020

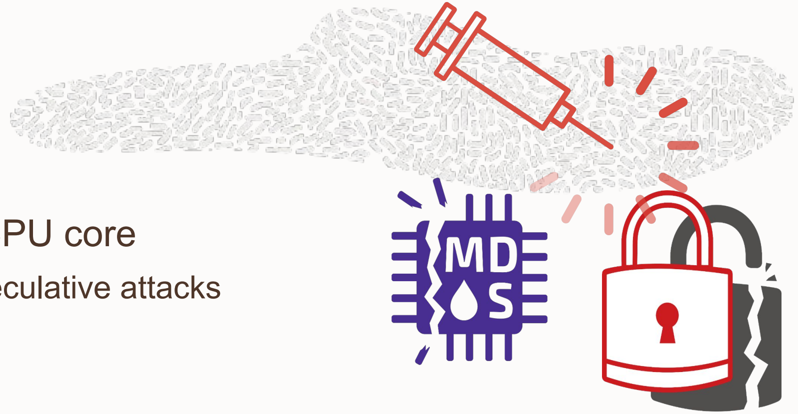
Safe harbor statement

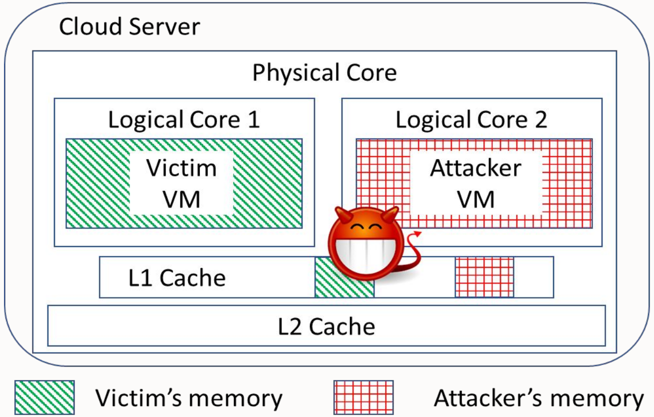
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Why ASI?

Context

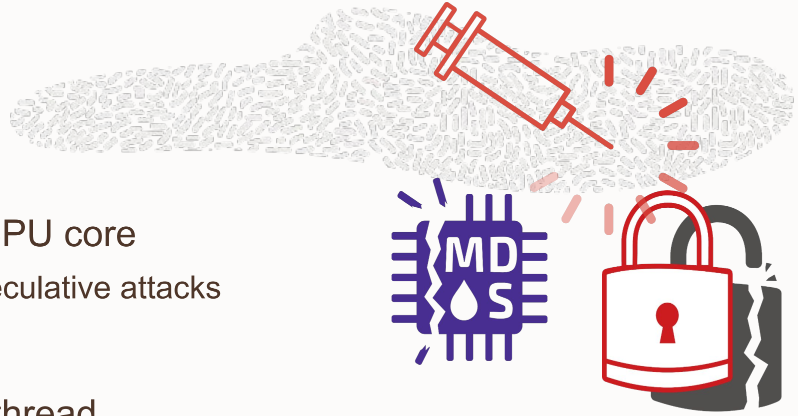
- Data can leak between CPU threads from the same CPU core
  - Leak through shared hardware (micro)architecture via speculative attacks
  - Example: L1TF and MDS speculative attacks

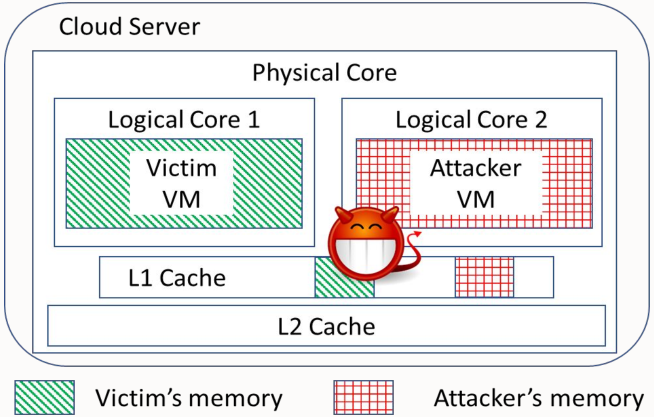


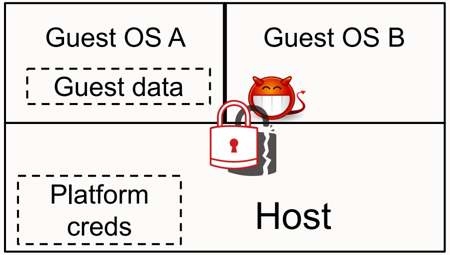


Context

- Data can leak between CPU threads from the same CPU core
  - Leak through shared hardware (micro)architecture via speculative attacks
  - Example: L1TF and MDS speculative attacks
- A VM can control the leak and spy on its sibling CPU thread
  - Guest can spy on another guest running on the same CPU core
  - Guest can spy on the host running on the same CPU core
- Major issue for virtual machines and cloud providers
  - Allow Guest-to-Guest attacks and Guest-to-Host attacks

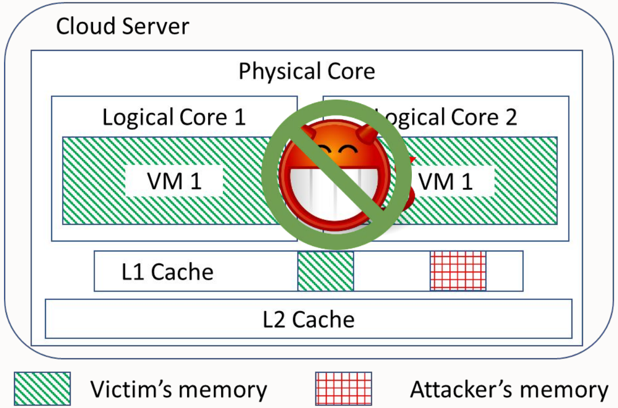






Mitigations

- Basic mitigation: disable CPU hyper-threading
  - Most complete and reliable solution
  - Significant impact on performances and capacity
- Mitigation for Guest-to-Guest attacks
  - Pin VMs to different dedicated CPU cores
  - Core scheduling
- Mitigation for Guest-to-Host attacks
  - Synchronize VM entry and VM exit for all CPU threads on a CPU Core, i.e., halt sibling core when running host code
  - Flush L1D cache before every VMENTER - not cheap
  - **KVM Address Space Isolation (ASI)**



ASI Intuition - Can’t Speculate Through a Page Fault

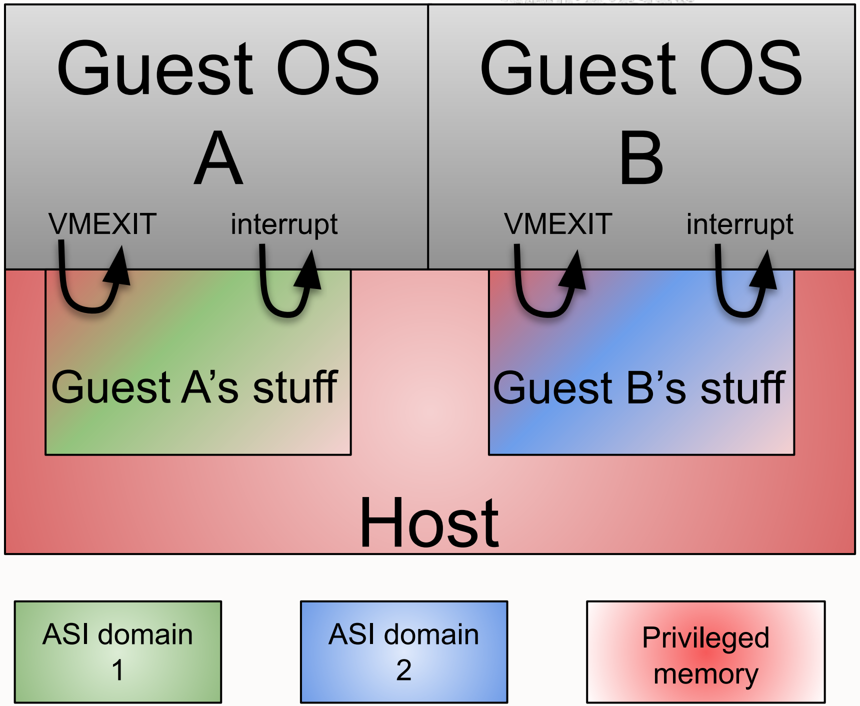
Trivial example: Spectre V1 (bounds check bypass)

```
int foo(u8 *arr, int size, int index) {
        if (index < size) {
                // Should lfence
                return global_array[ arr[index] * 64 ];
        }
        // ...
}
```

If index is out of bounds, “`arr`” might speculatively still be accessed.
If `&arr[index]` is not mapped in the page-table → page-fault

ASI Overview

- Define a restricted address space
  - Define a page table with limited data
  - Contain no sensitive or secret data
- Prevent a sub-system from accessing the entire memory or unrelated data
- Sub-system explicitly enters/exits ASI
- ASI can be interrupted/resumed on interrupt, exception, context switch
- ASI can be suspended/exited on page-fault
- **Accessing secrets → page fault**



ASI Applications

- KVM ASI
  - Protect against guest-to-host attack
  - Major challenge: what data to include in the ASI domain?
- User ASI
  - Implement kernel/user page-table switch with ASI
  - Refactor Kernel Page-Table Isolation (KPTI) to use ASI
- Userland ASI?
  - Provide multiple user address spaces to a user process
  - Isolate user virtual environment (JVM, containers…)
  - To be investigated

KVM ASI

KVM ASI

- Address space with limited kernel and VM mappings
  - Only has mappings required to enter VM and handle VM exit
- Goal: run VM and handle (most) VM exits without exiting ASI
- Only map data from a single VM in the same ASI
  - Prevent VM running on same CPU core to steal data from host kernel or from another VM
- Synchronize on VM entry only if sibling CPU thread is not running ASI
  - No need to synchronize if all CPU threads are running with ASI
  - Core scheduling helps having the same VM ASI run on all CPU threads

ASI Lifecycle

- Create an ASI
  - Each ASI has its own page-table
- Populate the ASI page-table
- Enter ASI = switch to the ASI page-table
- Handle interrupt/exception/fault/context switch
  - ASI can be interrupted and resumed, or exited
- Exit ASI = switch to the kernel page-table
- Destroy the ASI page-table

KVM ASI Usage

- Create one KVM ASI per VM (or per VCPU)
- Populate the KVM ASI page-table
- Use KVM ASI when running a guest VCPU
  - KVM_RUN ioctl
  - Enter KVM ASI
  - Ensure KVM ASI is used on siblings
    - VMEnter
    - VMExit

    KVM ASI also used on siblings
  - Stop KVM ASI enforcement on siblings
  - Run (most) VMExit handlers with KVM ASI
  - vcpu_run() loop

ASI Page-Table Filling

- Basic solution: manually add each required mapping
  - asi_map(asi, address, length);
  - asi_unmap(asi, address);
- Similar to memory allocation/free
- Works okay for pre-allocated well-known buffers
- More cumbersome for frequently re-allocated buffers
- Need mechanisms for adding mappings more automatically

ASI Page-Table Filling - Statically Allocated Buffers

- Put non-sensitive statically allocated data into a dedicated object file section
- Just require to add a compiler section attribute to the data definition

```
#define __asi_not_sensitive __attribute__((__section__(ASI_NON_SENSITIVE_SECTION_NAME)))
```

```
static struct clocksource clocksource_tsc __asi_not_sensitive = {
        .name                   = "tsc",
        .rating                 = 300,
        .read                   = read_tsc,
L  arch/x86/kernel/tsc.c                                          67%
```

- Then map the entire section into the ASI

ASI Page-Table Filling - Dynamically Allocated Buffers

- Use new flag when allocating a non-sensitive buffer:
  - `GFP_GLOBAL_NONSENSITIVE`- non-sensitive data for any ASI

```
cluster_hotplug_mask = kzalloc_node(sizeof(*cluster_hotplug_mask),
                                    GFP_KERNEL | GFP_GLOBAL_NONSENSITIVE, node);
/x86/kernel/apic/x2apic_cluster.c                                  46% ☰ 1(
```

  - `GFP_LOCAL_NONSENSITIVE`- non-sensitive data for the local process

```
static struct vmcs *alloc_vmcs_cpu(bool shadow, int cpu)
{
        int node = cpu_to_node(cpu);
        struct page *pages;
        struct vmcs *vmcs;

        pages = __alloc_pages_node(node,
                                   GFP_KERNEL_ACCOUNT | GFP_LOCAL_NONSENSITIVE,
                                   vmcs_config.order);
 arch/x86/kvm/vmx.c                                                29%
```

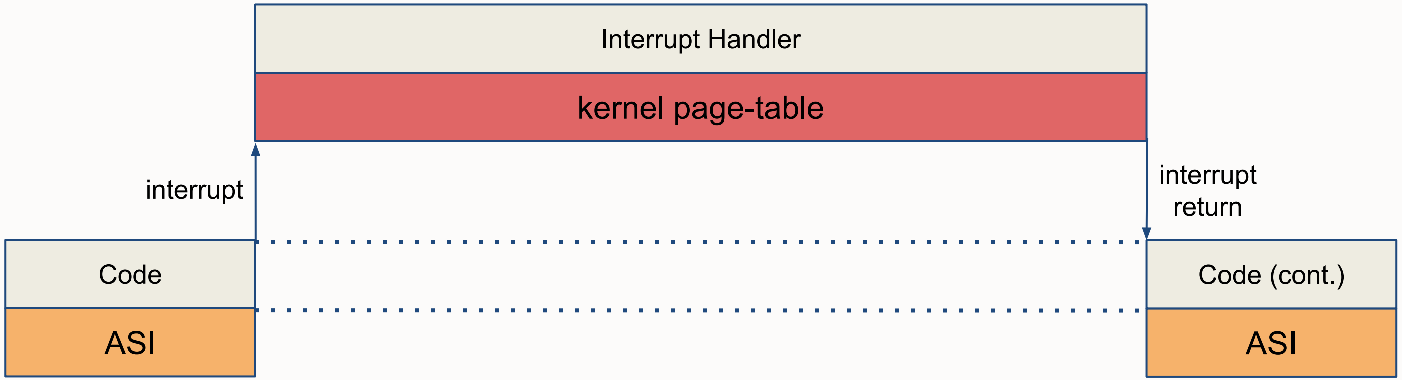
- Buffer is automatically [un]mapped into the current ASI on alloc/free

ASI Page-Table Switching

- Simple on x86: just update the CR3 control register
  - But inefficient if you don't consider TLB
- TLB = Translation Lookaside Buffer
  - Caches VA-to-PA translations, avoid MMU to go through the page-table
  - Switching require flushing the TLB → impact on performances
- Optimization: use Process-Context Identifiers (PCIDs)
  - Facility to associate a TLB entry with a page-table
  - Avoid flushing the entire TLB
  - Flush only TLB entries for a specified PCID

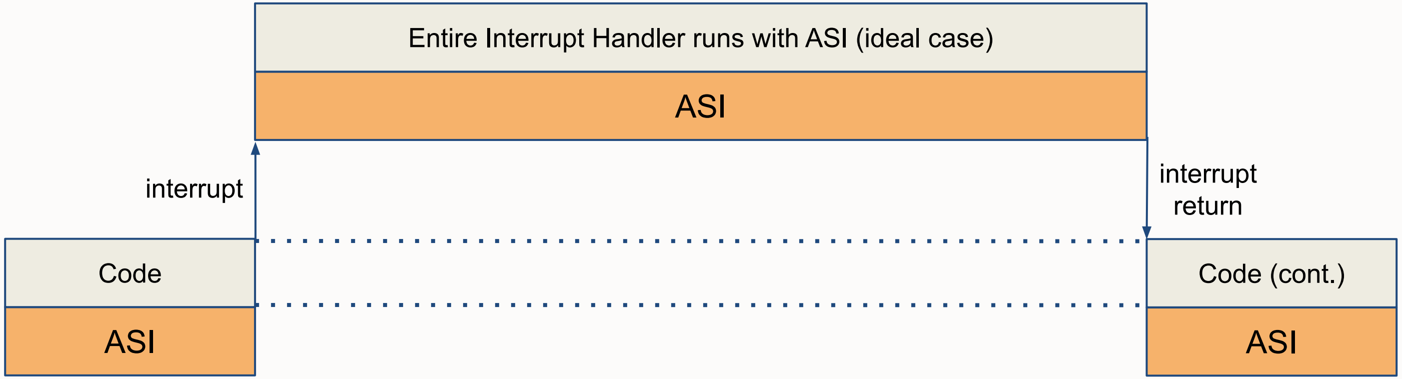
ASI and Interrupts/Exceptions

- Should be able to handle interrupts/exceptions while running with ASI
  - But ASI doesn't necessarily have all mappings to run the interrupt/exception handler
- Suspend/Resume ASI while running interrupt/exception handler
  - Exit ASI at the beginning of the handler
  - Re-enter ASI at the end of the handler
  - Solution when we know the handler needs the kernel page-table

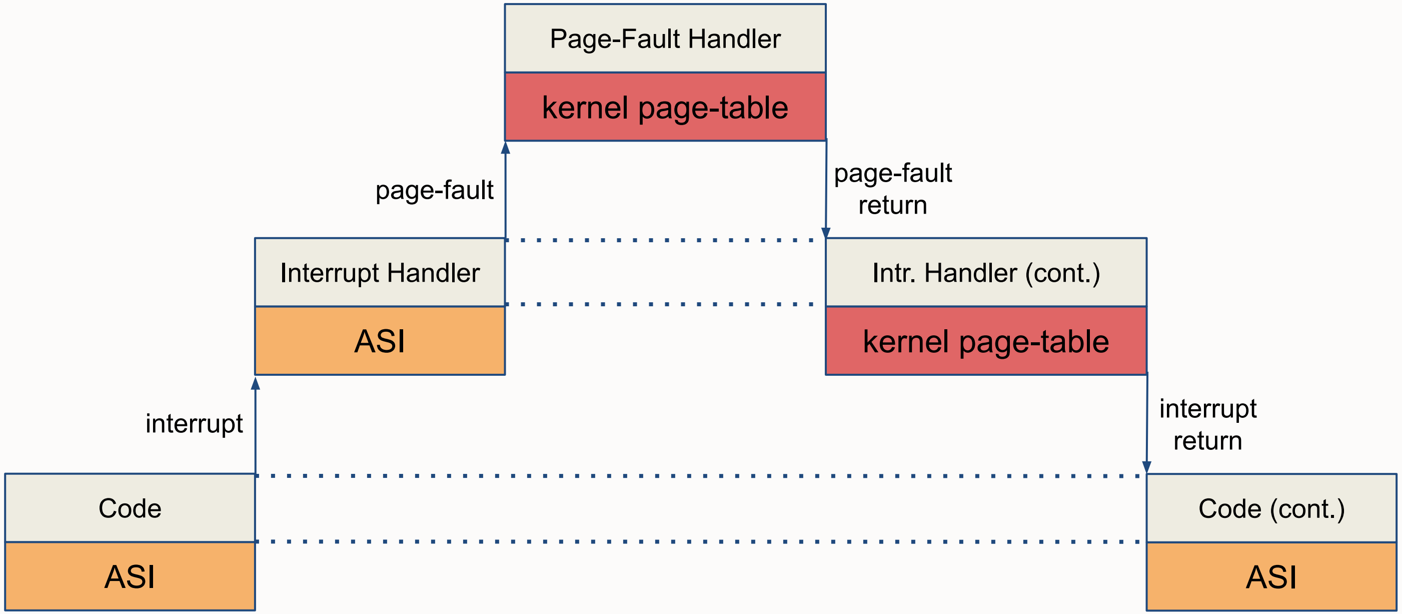


ASI and Interrupts/Exceptions

- Run the interrupt/exception handler with the ASI page-table
- Handler can fault because of the ASI restricted page-table
- Page-fault handler will then exit ASI and resume handler execution
- ASI is re-entered after the handler is done (if there was a fault)
- Solution when we expect the handler to be able to run with ASI



ASI and Interrupts/Exceptions



ASI and Page Fault

- Basic behavior:
  - Exit ASI and retry with kernel page-table
  - Log fault to identify missing mapping in ASI page-table
  - Eventually manually augment the ASI page-table to prevent this fault
- Can we do better?
  - Automatically add missing mapping to the ASI page-table
  - Only if mapping does not provide access to sensitive data
  - Allow to return with ASI and avoid the same fault to happen again
  - Require to flag kernel memory which has no sensitive data (e.g. `GFP_GLOBAL_NONSENSITIVE`)

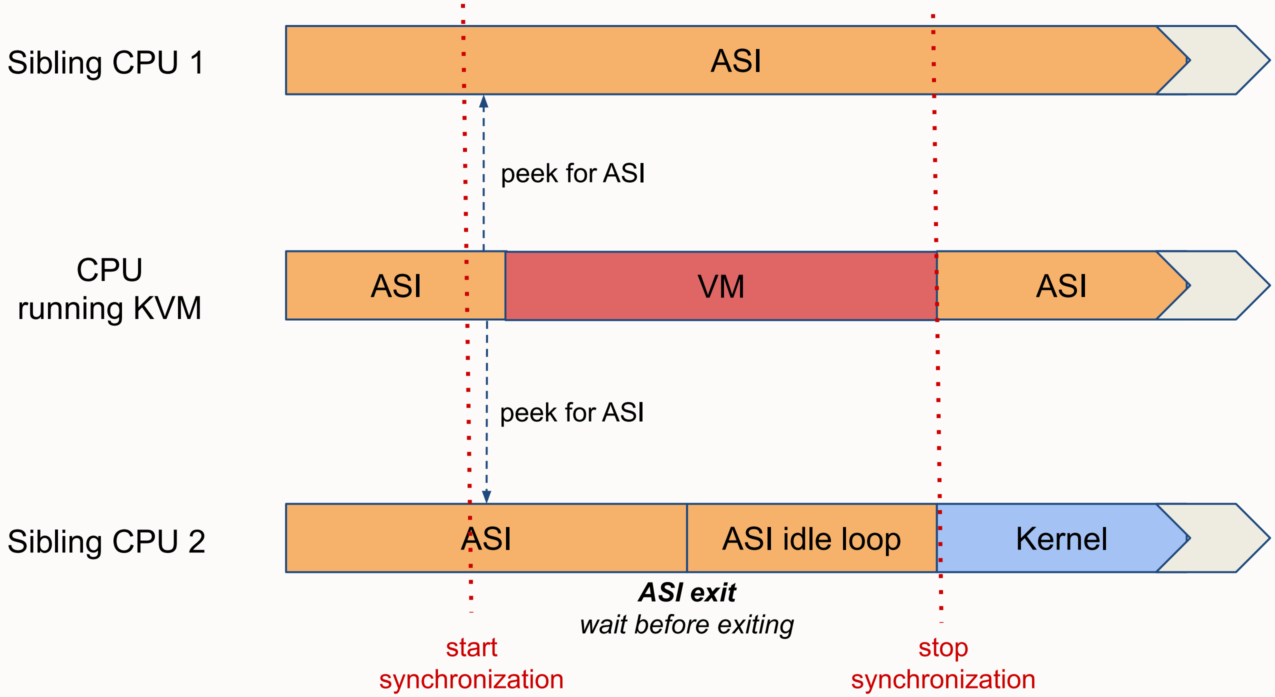
ASI and Context Switch

- Task using ASI is scheduled out → interrupt ASI
  - Exit ASI and save ASI information for this task
- Task using ASI is scheduled in → resume ASI
  - Enter ASI with the saved ASI information for this task
- Additional complexity if switch occurs during interrupt/exception handler
  - Interrupt/exception might already have exited ASI
  - Need to save/restore ASI state information

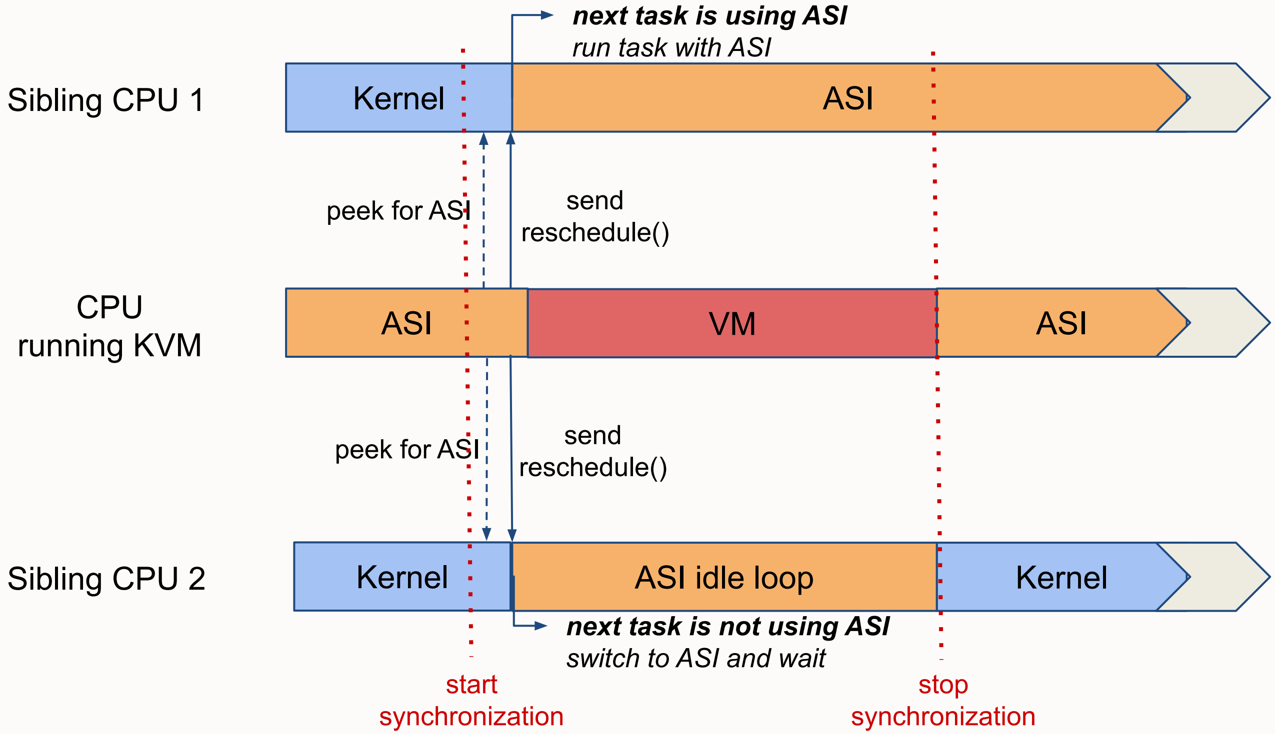
ASI Synchronization Across CPU Threads

- Mechanism to force all CPU threads from a CPU core to use a specified ASI
- If sibling CPU thread is running ASI
  - CPU thread continues to run uninterrupted
  - If CPU thread tries to exit ASI when it waits in idle loop
- If sibling CPU thread is not running ASI
  - CPU thread is requested to reschedule
  - If next task is using ASI then enter ASI and run the task
  - If next task is not using ASI then enter ASI and wait in idle loop
- KVM ASI uses ASI synchronization while running a VM
  - Synchronization is started before VMEnter
  - Synchronization is stopped after VMExit

KVM ASI Synchronization with Siblings Running ASI



KVM ASI Synchronization with Siblings not Running ASI



ASI Synchronization and Interrupt/Exception

- If Interrupt/Exception needs to exit ASI
- Then this will cause **all** sibling CPU threads to interrupt
- One CPU thread receives an interrupt/exception, touches secret data → PF → ASI-exit
- That CPU thread forces all sibling CPU threads to interrupt
  - All CPU threads exit ASI, wait for interrupt to be processed, return to ASI
  - Behavior is synchronized across all CPU threads

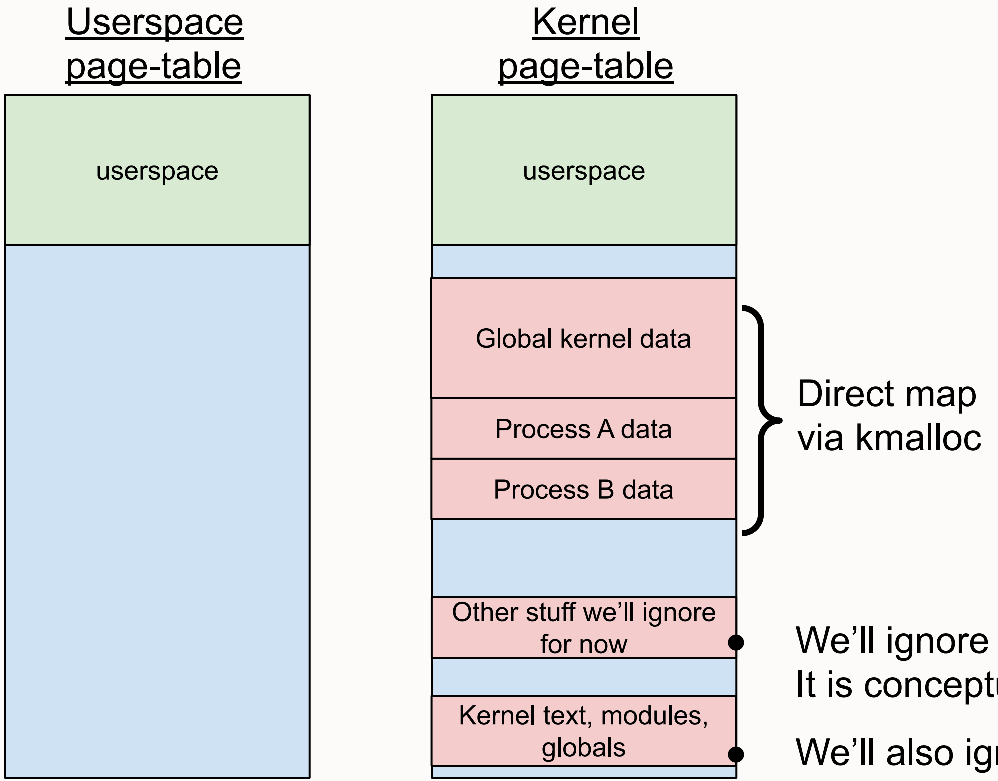
- Core Scheduling has a somewhat similar mechanism
  - To prevent interrupt/exception handler to run with tagged process

ASI Page Tables

The KPTI Model - Control & Data Privilege

Not mapped

Privileged data



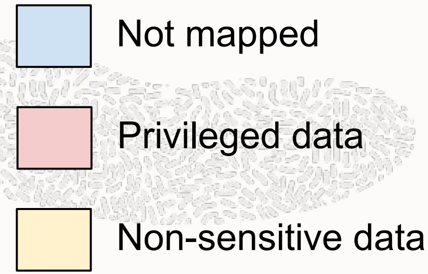
To mitigate Meltdown attacks, KPTI differentiates between privileged/unprivileged execution level.

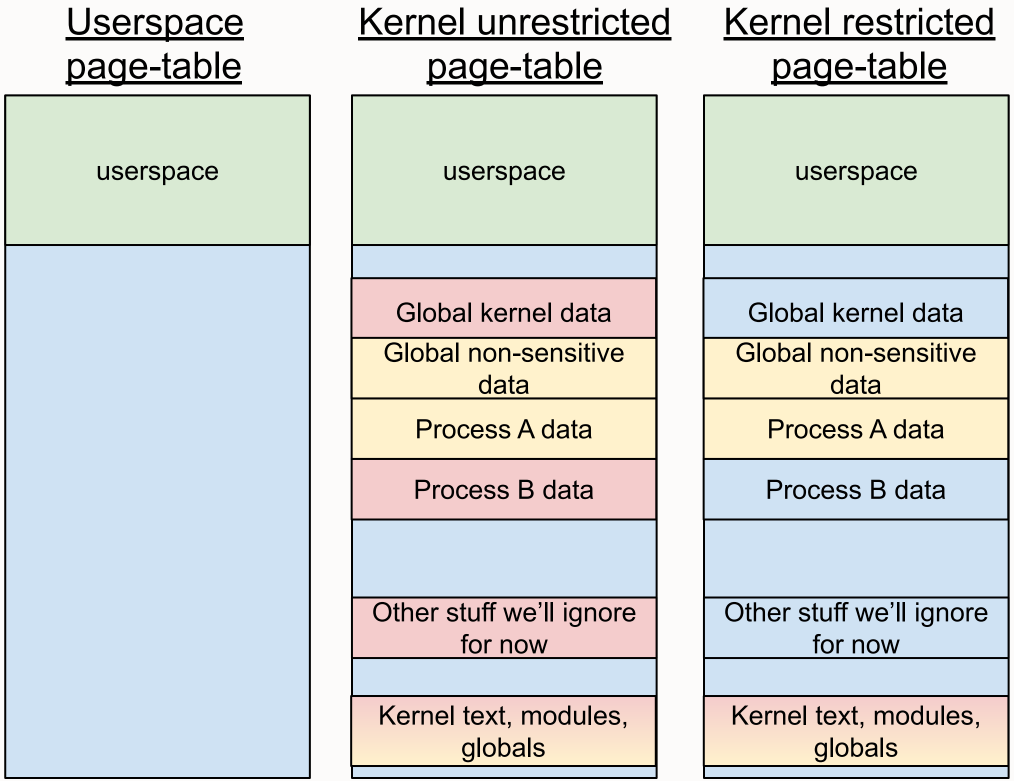
The methodology - using two page tables to separate between user space memory and kernel privileged memory.

We'll ignore vmalloc space for now.
It is conceptually similar to direct map

We'll also ignore global vars

The ASI Model - Data Privilege





In ASI, we define privilege based on data access, not execution-level. We add another “restricted” page-table which only maps kernel **non-sensitive** data.

Data is deemed **non-sensitive** if, when stolen by a malicious VM, does not pose a security threat to <u>other VMs</u> or <u>cloud infrastructure</u>.

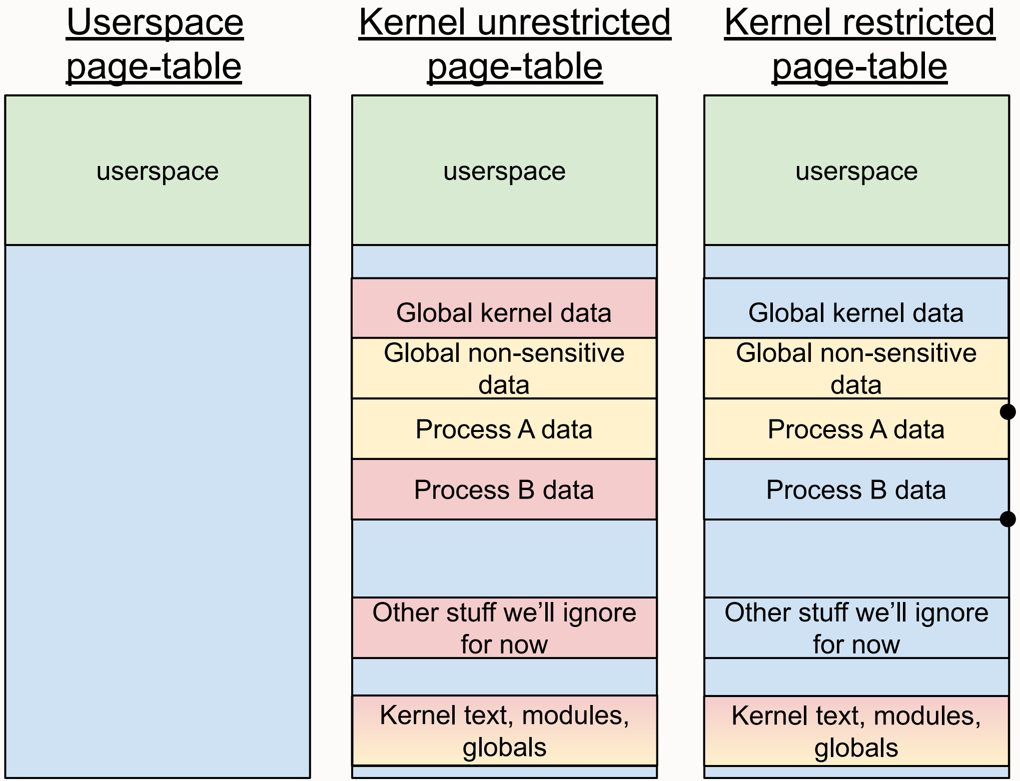
For performance reasons, we’re interested in memory that is accessed frequently by the kernel, when operating a VM between VMEXIT and VMENTER.

The ASI Model - Data Privilege

Not mapped

Privileged data

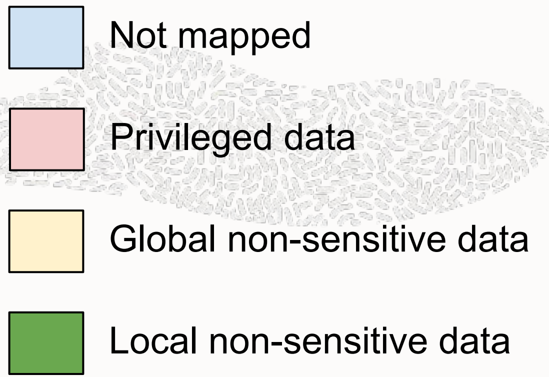
Non-sensitive data

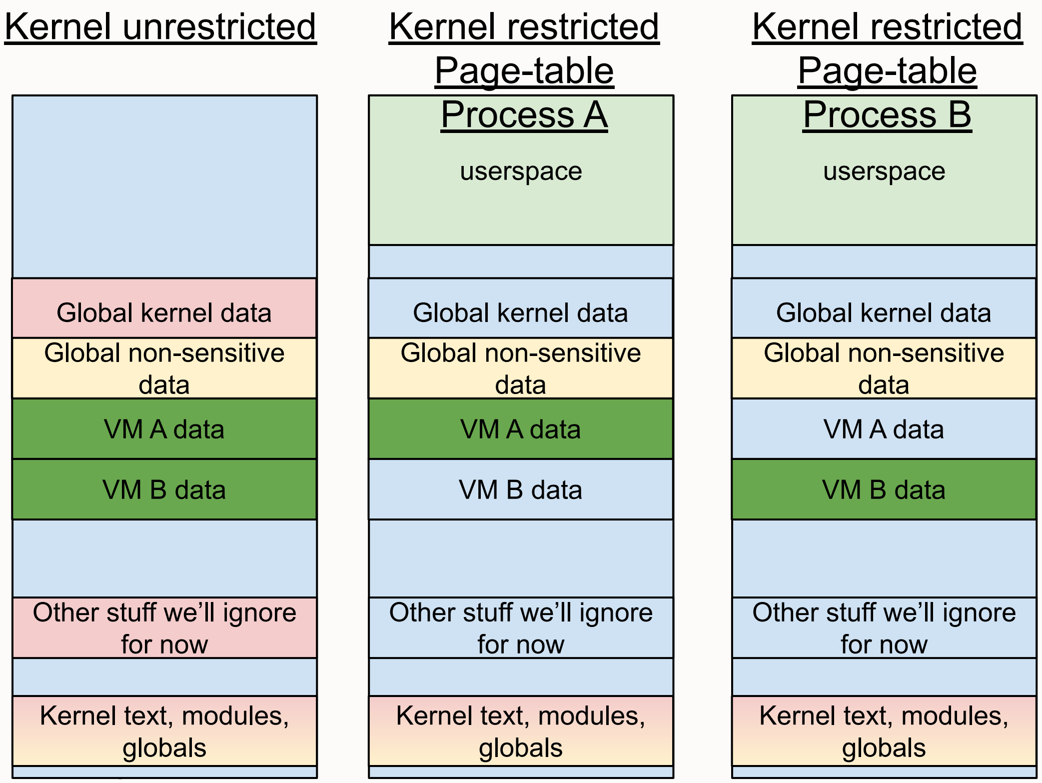


Non-sensitive data can be accessed freely, without the need for any L1TF mitigations

Access to "unmapped" area will cause a PF, which will switch to the unrestricted page-table. Use L1TF mitigation when switching (stunning/L1D-flush)

Partitioning Global/Local Data





Is data considered non-sensitive locally in a process or globally in the entire system?

Examples:

1. Local data: VMCS, vcpu, file-descriptor-table
2. Global data: sk_buffs

All non-sensitive data in ASI can be read by a guest VM via an L1TF attack

While we want VM-1 to access its VMCS freely we don't want VM-1 to read the VMCS of VM-2!!

Early Results

Initial Results - Aeropspike YCSB

Ratio of ASI-exits/VM-exits

```
KVM/VCPU 0xffffc9003f575000/0: Time 139.64 seconds, asi/vm exits = 2504149 / 4816824 = 51.99 %
KVM/VCPU 0xffffc9003f575000/1: Time 139.51 seconds, asi/vm exits = 355404 / 657455 = 54.06 %
KVM/VCPU 0xffffc9003f575000/2: Time 139.51 seconds, asi/vm exits = 121572 / 481107 = 25.27 %
KVM/VCPU 0xffffc9003f575000/3: Time 139.51 seconds, asi/vm exits = 214304 / 439711 = 48.74 %
KVM/VCPU 0xffffc9003f575000/4: Time 139.51 seconds, asi/vm exits = 136400 / 351173 = 38.84 %
KVM/VCPU 0xffffc9003f575000/5: Time 139.51 seconds, asi/vm exits = 163717 / 445313 = 36.76 %
KVM/VCPU 0xffffc9003f575000/6: Time 139.51 seconds, asi/vm exits = 110505 / 360235 = 30.68 %
KVM/VCPU 0xffffc9003f575000/7: Time 139.51 seconds, asi/vm exits = 135873 / 393257 = 34.55 %
total_asi_exits = 3741924
KVM/VCPU 0xffffc90019e44000/0: Time 233.39 seconds, asi/vm exits = 2524400 / 4920423 = 51.30 %
KVM/VCPU 0xffffc90019e44000/1: Time 233.26 seconds, asi/vm exits = 368409 / 686433 = 53.67 %
KVM/VCPU 0xffffc90019e44000/2: Time 233.26 seconds, asi/vm exits = 114840 / 392239 = 29.28 %
KVM/VCPU 0xffffc90019e44000/3: Time 233.26 seconds, asi/vm exits = 194565 / 461824 = 42.13 %
KVM/VCPU 0xffffc90019e44000/4: Time 233.26 seconds, asi/vm exits = 108116 / 384302 = 28.13 %
KVM/VCPU 0xffffc90019e44000/5: Time 233.26 seconds, asi/vm exits = 163987 / 457225 = 35.87 %
KVM/VCPU 0xffffc90019e44000/6: Time 233.26 seconds, asi/vm exits = 100777 / 460626 = 21.88 %
KVM/VCPU 0xffffc90019e44000/7: Time 214.20 seconds, asi/vm exits = 90452 / 335486 = 26.96 %
total_asi_exits = 3665546
```

Initial Results - Aeropspike YCSB

Exit details

	RIP	address	accessor	allocator	count	CDF
0	0xffffffff811e33b9	0xffffc90019e55ca8	kernel/rcu/srcutree.c:410	PO: ././arch/x86/kvm/vmx.c:11916	80677	1.000000
1	0xffffffff81ade77d	0xffffe8fffe3c8730	././lib/timerqueue.c:52	././kernel/events/core.c:10523	51266	0.926417
2	0xffffffff81ade77d	0xffffe8fffe348730	././lib/timerqueue.c:52	././kernel/events/core.c:10523	45283	0.879659
3	0xffffffff81ade77d	0xffffe8fffe408730	././lib/timerqueue.c:52	././kernel/events/core.c:10523	31756	0.838358
4	0xffffffff81ade77d	0xffff88b6e494a7a0	././lib/timerqueue.c:52	PO: ./././include/acpi/platform/aclinuxex.h:86	31113	0.809394
5	0xffffffff811e33b9	0xffffc9001f3b3ca8	kernel/rcu/srcutree.c:410	././arch/x86/kvm/vmx.c:11916	30228	0.781017
6	0xffffffff81ade77d	0xffffe8fffe388730	././lib/timerqueue.c:52	././kernel/events/core.c:10523	28256	0.753447
7	0xffffffff81ade77d	0xffffa8b699dddde8	././lib/timerqueue.c:52	<Unknown alloc>	27816	0.727675
8	0xffffffff81ade77d	0xffffe8a000148730	././lib/timerqueue.c:52	././kernel/events/core.c:10523	23678	0.702305
9	0xffffffff81ade77d	0xffffe8a0002c8730	././lib/timerqueue.c:52	././kernel/events/core.c:10523	20716	0.680709
10	0xffffffff81ade77d	0xffffe89fffdc8730	././lib/timerqueue.c:52	././kernel/events/core.c:10523	15481	0.661815

Initial Results - Aeropspike YCSB

Exit details by allocation site

variable	count	CDF
<Unknown alloc>	428458	1.000000
././kernel/events/core.c:10523	419826	0.609217
PO: ././arch/x86/kvm/vmx.c:11916	80677	0.226306
sd_llc_size	39383	0.152723
PO: ./././include/acpi/platform/aclinuxex.h:86	31113	0.116803
././arch/x86/kvm/vmx.c:11916	30228	0.088426
././arch/x86/kvm/../../../virt/kvm/eventfd.c:1016	14221	0.060856
PO: ././fs/exec.c:1761	9620	0.047885
PO: ././kernel/fork.c:890	9263	0.039111
././fs/timerfd.c:391	6797	0.030663
PO: ././arch/x86/events/intel/uncore.c:321	4412	0.024464
././security/scudo/lsm.c:310	2896	0.020439
././arch/x86/kvm/../../../virt/kvm/irqchip.c:212	2578	0.017798
PO: ././fs/binfmt_elf.c:466	2239	0.015447
intel_pmu_ops	1955	0.013405
PO: ././kernel/fork.c:1659	1639	0.011622
softnet_data	1519	0.010127
numa_node	1294	0.008741

ASI Status

History

- Idea suggested after L1TF speculative attack discovery
  - Initially introduced by Liran Alon in KVM Forum 2018 BoF
  - Inspired from Microsoft HyperV HyperClear L1TF mitigation
  - More discussions at Linux Plumbers Conference 2019
- Several RFCs submitted by Oracle
  - v1 (KVM ASI), v2 (Kernel ASI), v3 (+ integration with PTI), v4 (+ page-table management)
  - ~~v5 (+ ASI synchronization)~~
- Different implementation recently proposed by Google
  - Presented at Linux Plumbers Conference 2020

Status and Future

- Collaboration between Oracle and Google
- Define a common solution and converge implementation
  - Page-table management
  - Interrupt/exception/page-fault handling
  - ASI synchronization across CPU threads (shared component with Core Scheduling?)
  - PTI integration
- Goal: upstream a common implementation
- Preliminary work: Defer PTI CR3 Switch to C Code
  - Simplify ASI support and integration with Page-Table Isolation (PTI)

Thank You

alexandre.chartre@oracle.com
oweisse@google.com