

Lecture 05 Intel VMX (part 3): Chapters 31-32

Why EPT and APICv work the way they do. A design-oriented guide to Chapters 31 and 32 of the Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3C. Order Number 325384-092US.

Contents

- 01** Two bottlenecks
- 1** bottlenecks
- 02** Three translation identities
- 03** Hypervisor-managed linear translation
- 04** Extended Page Tables
- 05** Translation caches and invalidation
- 06** The APIC before virtualization
- 6**
- 07** Virtualizing the local APIC
- 08** Posted interrupts and VT-d
- 09** Putting Chapters 31 and 32 together

01

Two bottlenecks

Preserve the abstraction, avoid the exit

Chapters 31 and 32 of the Intel SDM explain two families of VMX features that remove some of the most expensive reasons for entering a hypervisor.

CHAPTER 31

Virtualizes address translation with VPIDs, HLAT, EPT, translation caches, and explicit invalidation.

CHAPTER 32

Virtualizes the local APIC and interrupt-delivery fast path with a virtual-APIC page, virtual interrupt state, IPI virtualization, and posted interrupts.

The common idea is preserve the abstraction while avoiding exits on the common path.

Without these features

Without these features, a VMM can still virtualize memory and interrupts, but it must intervene far more often. It may need to:

- maintain shadow page tables;
- flush translations on VM transitions;
- trap APIC-register accesses;
- re-enter the guest to inject each interrupt.

EPT and APIC virtualization move those repetitive, mechanically checkable operations into the processor while preserving explicit points at which the VMM can enforce policy.

A guest wants to own memory and an interrupt controller

MEMORY

The guest translates a linear address using its own page tables:

```
guest linear address → guest physical address
```

The machine must then translate that guest physical address into actual host memory:

```
guest physical address → machine physical address
```

INTERRUPTS

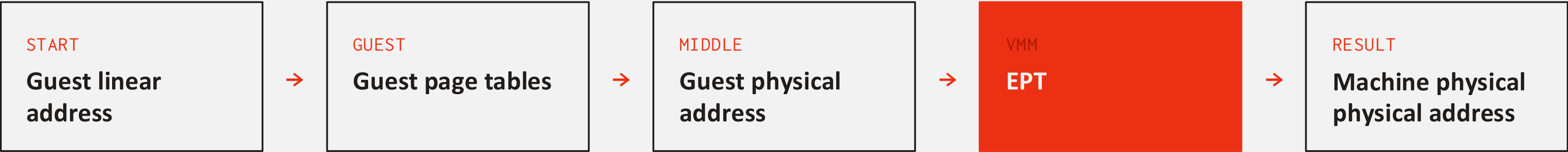
The guest wants to program an APIC, queue interrupt vectors, mask priority classes, acknowledge completed interrupts, and send IPIs.

The VMM must make all of that appear private to the virtual CPU even though the real APIC belongs to the host processor.

The slow but conceptually simple solution is to exit whenever either subsystem is touched. The scalable solution is to let the processor execute the common cases against VMM-provided data structures.

Hardware performs the repeated work

CHAPTER 31 · MEMORY



CHAPTER 32 · INTERRUPTS



The VMM still creates the mappings and routing policy. Hardware performs the repeated walks, lookups, and state transitions.

02

Three translation identities

Three identities answer three cache questions

IDENTITY	ROUGH QUESTION IT ANSWERS	TYPICAL OWNER
PCID	Which guest process or linear address space does this translation belong to?	Guest OS
VPID	Which VMX linear context or virtual processor does it belong to?	VMM
EPTP / EPT root	Which guest-physical-to-machine-physical mapping was used?	VMM

A cached combined translation can therefore be thought of as carrying a tuple resembling:

(VPID, PCID, EPT context, guest linear page) → machine physical page

- This is not the literal internal TLB format, which is implementation-specific. It is the architectural identity needed to reason about when a cached result is reusable.

Why both VPID and PCID?

PCID distinguishes address spaces managed by an operating system. Two processes in one guest may have different PCIDs but still belong to the same virtual CPU context.

VPID distinguishes VMX contexts. Without it, a translation cached while a guest was running could be mistaken for a host translation or for a translation belonging to another guest after a VM transition. Original VMX avoided that mistake by flushing TLB and paging-structure caches on transitions. That was safe, but it destroyed useful work.

With VPID enabled, a nonzero 16-bit VPID from the VMCS tags the guest's relevant cached mappings. VM entry and exit can change the current translation identity rather than flushing everything.

Why VPID zero is special

VPID 0 represents root operation, operation outside VMX, and non-root operation when VPID is disabled.

VM entry therefore rejects a zero VPID when VPID support is enabled. Reserving zero gives the processor a defined untagged/root identity and prevents an enabled guest from colliding with it.

WHY THIS MAKES SENSE

A transition does not prove that a cached translation is stale. It proves only that the processor is entering a different context.

Tagging preserves valid translations; invalidation is reserved for actual mapping changes or identifier reuse.

Who chooses VPID and PCID?

The VMM directly chooses the VPID. The guest OS normally chooses the PCID. There is no hardware instruction that allocates a VPID: the VMM selects a nonzero number from 1 through 65535 , writes it into the VMCS, and takes responsibility for invalidating stale mappings before changing what that number means.

A simple VMM commonly gives each vCPU or VMCS its own VPID:

EXECUTION CONTEXT	VPID	PCID	EPT CONTEXT
Host process	0	5	None
VM A, vCPU 0, process A	17	5	EPT-A
VM A, vCPU 0, process B	17	6	EPT-A
VM A, vCPU 1, process A	18	5	EPT-A
VM B, vCPU 0, process A	19	5	EPT-B

PCIDs subdivide a VPID

If the VMM exposes PCID support and the guest enables `CR4.PCIDE`, `guest CR3[11:0]` contains the current 12-bit PCID. The guest can use PCID 5 for one process and PCID 6 for another without knowing that both operate inside VPID 17.

The host can independently use PCID 5 because `(VPID 0, PCID 5)` and `(VPID 17, PCID 5)` are different translation contexts.

This division avoids allocating one VPID per guest process:

VMM:

assigns a VPID to the virtual execution context

Guest OS:

subdivides that context with PCIDs for its processes

VPIDs are optional

A VMM can run correctly without allocating any nonzero VPID. If the “enable VPID” execution control is 0, non-root execution uses current VPID 0. Because root and guest execution then share the default identity, VM entry and VM exit invalidate the relevant VPID-zero linear and combined mappings for all PCIDs.

This is safe but discards useful guest translations on transitions. EPT can still be enabled, and EPT-only guest-physical mappings are not required to be discarded merely because VPID is disabled.

VPID disabled:

host runs with VPID 0

→ VM entry invalidates

relevant VPID-0 mappings

guest runs with VPID 0

→ VM exit invalidates

relevant VPID-0 mappings

host resumes

Production VMMs generally enable VPIDs when supported. A small or educational VMM can initially leave them disabled and add the allocation and invalidation bookkeeping later.

Reusing identifiers safely

An identifier is not permanent ownership. If a VMM gives a VPID previously used by guest A to guest B, stale linear or combined mappings may still carry that VPID. The VMM must invalidate the old context before reuse unless a distinct EPT context makes the retained combined mappings unambiguous and the architecture's detailed rules permit that use.

VPID state is meaningful in each logical processor's translation caches; it is not a globally registered vCPU number. When a vCPU migrates to another physical CPU, the VMM must ensure that the destination has no incompatible stale mappings with the same VPID.

PER-CPU	UNIQUE	INVALIDATE	GENERATION
Allocate VPIDs per physical CPU. CPU.	Use unique active values.	Invalidate the selected VPID on the destination.	Keep a generation number and flush when the allocator wraps.

A tag avoids routine flushes, but reassigning the meaning of a tag requires invalidation.

03

Hypervisor-managed linear translation

Why EPT alone cannot protect a linear address

EPT permissions attach to guest-physical pages. A VMM can make the page holding the guest kernel's code or system-call table read-only, and any write to it exits. But an attacker with guest-kernel write access does not need to write that page:

```
protected:  GVA sys_call_table → GPA 0x1000    (EPT: read-only)
```

```
attacker edits a guest page-table entry:
```

```
      GVA sys_call_table → GPA 0x7000    (attacker data, writable)
```

```
EPT is never violated: GPA 0x1000 is untouched
```

EPT protects the contents of a page. It cannot protect which page a linear address refers to, because that translation lives in guest page tables the guest can rewrite.

Before HLAT: trap every guest page-table write

WITHOUT HLAT

The VMM write-protects, in EPT, every guest-physical page that holds guest paging structures, and checks each update in an EPT-violation handler.

- Page tables change constantly: page faults, mmap, fork , process exit. Each change becomes an exit.
- The VMM must track which GPAs are page tables as the guest allocates and frees them.
- Every guest process has its own hierarchy under a different CR3.

WITH HLAT

The VMM builds a small tree only for the protected ranges and points HLATP at it.

- Guest CR3 and its page tables change freely, without exits.
- Restart entries hand every other region back to ordinary guest paging.
- The HLAT tables live in guest-physical pages the VMM does not map writable in EPT.

The protected mappings become VMM-owned, so the guest cannot redirect them, and the guest's own paging keeps running at native speed.

HLAT: a hypervisor-selected linear root

Ordinary four- or five-level paging begins at the root selected by guest CR3 . HLAT paging instead begins at a root selected by the VMCS field HLATP.

Depending on the configured HLAT prefix size:

- all linear addresses may use HLAT; or
- addresses with bit 63 set may use HLAT while addresses with bit 63 clear use ordinary guest paging.

An HLAT paging-structure entry can also set its restart bit.

Translation then starts over using ordinary paging and the guest's CR3.

linear address

- selected for HLAT?
 - no: walk from guest CR3
 - yes: walk from VMCS.HLATP
 - restart entry encountered?
 - no: use HLAT result
 - yes: restart complete walk from guest CR3

Why a separate root, and why restart?

WHY NOT REWRITE GUEST CR3?

CR3 is architecturally owned by the guest and changes as the guest switches processes. A VMCS-owned root lets the hypervisor supply selected linear mappings without pretending that the guest chose them and without chasing every guest CR3 change.

- HLAT is limited to four- and five-level paging because it is a modification of those ordinary paging algorithms, not a third not a third independent physical-memory translation layer. EPT may still translate the guest-physical addresses used during addresses used during either the HLAT or ordinary guest walk.

WHY THE RESTART MECHANISM?

A separate top-level tree would otherwise need to duplicate every ordinary guest mapping that it does not intend to override. Restart entries let the HLAT hierarchy say, in effect, “use the guest's normal translation for this region.”

04

Extended Page Tables

EPT: two-dimensional translation

With EPT enabled, addresses produced by guest paging are guest physical addresses, not final machine addresses. The processor walks VMM-owned EPT structures to obtain the machine physical address and EPT permissions. If guest paging is disabled, the guest linear address is treated directly as a guest physical address and is then translated through EPT.

If guest paging is enabled, a single guest memory reference can require several EPT translations:

- 1 Translate the guest physical address of a guest page-table entry through EPT so the processor can read that entry.
 - 2 Repeat for each level of the guest page-table walk.
 - 3 Translate the final guest physical page through EPT to access the requested data or instruction.
- Paging-structure caches and TLBs can retain intermediate and combined results so the full nested walk is not repeated for every access.

Why EPT is better than shadow page tables

Before hardware second-level translation, a VMM commonly maintained a shadow page table whose entries already mapped guest linear addresses to machine physical pages. That forces the VMM to observe guest page-table changes and keep a derived structure synchronized.

EPT separates ownership cleanly:

GUEST	VMM	PROCESSOR
Owns linear-to-guest-physical policy.	Owns guest-physical-to-machine-physical policy.	Composes them.

The guest can change its own page tables directly. The VMM normally becomes involved only when its EPT policy denies an access or when it intentionally changes the second-stage mapping.

EPTP selects the root and the walk's shape

The VMCS's EPT pointer, EPTP, includes:

- the physical address of the EPT root;
- the page-walk length, currently four or five levels where supported;
- the memory type used to read EPT paging structures;
- an enable for EPT accessed and dirty flags;
- newer feature controls such as supervisor shadow-stack access.

The root address identifies the address-space context for cached EPT-derived mappings. The other bits describe how that root must be interpreted.

Reusing the same root with incompatible walk settings without the required invalidation could cause old cached interpretation to survive, which is why the manual gives explicit INVEPT rules for such changes.

Permissions are checked through the whole walk

EPT entries provide independent read, write, and execute permissions. Effective permission is constrained by every entry used in the walk, not only the leaf. This has two useful consequences:

-
- 1 A VMM can deny access to a large region at an upper level without rewriting every leaf.
 - 2 A leaf cannot re-enable a permission removed by an ancestor.
-

With mode-based execute control, EPT can distinguish execution reached through supervisor-mode and user-mode linear addresses: “this page may contain guest-kernel code but must not be executable through a user mapping.”

- EPT can map 4 KiB, 2 MiB, and, where supported, 1 GiB pages. Large pages reduce walk depth and TLB pressure. Small pages provide finer placement, protection, dirty tracking, and copy-on-write control.

EPT violation versus EPT misconfiguration

EVENT	MEANING	TYPICAL HANDLER INTERPRETATION
EPT violation	The EPT structures are valid, but their current policy does not allow this access	Expected policy event: demand mapping, MMIO, write protection, execute protection, copy-on-write
EPT misconfiguration	The walk encountered an architecturally invalid or reserved encoding	VMM bug, corrupted EPT state, or unsupported configuration

VIOLATIONS

- an EPT entry is not present;
- a read, write, or instruction fetch encounters an entry without the applicable permission.

MISCONFIGURATIONS

- a reserved bit or reserved memory type;
- write permission without read permission;
- an address beyond the supported physical-address width.

A violation is a meaningful result of a valid policy. A misconfiguration means the policy itself cannot be interpreted safely. Keeping separate exit reasons prevents a normal memory-management slow path from hiding a broken page table.

Worked example: demand allocation

Suppose guest physical page 0x4000 has no EPT mapping.

```
guest load from a linear address
→ guest paging produces GPA 0x4xxx
→ EPT walk finds a non-present entry
→ EPT-violation VM exit
→ VMM allocates a machine page and
  installs a valid EPT entry
→ VMM invalidates if required
  and resumes
→ guest retries the memory access
```

The denied memory access has not completed.

The handler repairs policy and resumes at the instruction that needs the data. The instruction may then run normally.

The exit reports access and translation

The exit qualification tells the VMM whether the attempted access was a read, write, or instruction fetch and reports relevant effective EPT permissions. The VMCS also reports the guest physical address, and, when valid for the event, the guest linear address.

The VMM's policy is usually indexed by guest physical address, but emulating the guest-visible fault or decoding the instruction may require the linear address.

One linear address can map to different guest physical pages, and one guest physical page can be reachable through several through several linear aliases.

Exit or #VE?

With EPT-violation virtualization exceptions enabled, selected violations can be delivered to guest software as #VE instead of causing a VM exit. A leaf entry's suppress- #VE bit can force violations for that page to continue exiting.

This creates a second fast path: a cooperating guest can handle selected second-level translation events without a round trip to the outer VMM, while sensitive pages still force an exit.

Sub-page write permissions

Ordinary EPT write permission has page granularity. 4 KiB may still be too coarse when the VMM wants to permit updates to a small structure while protecting neighboring bytes.

Sub-page write permissions, or SPP, add write permission at naturally aligned 128-byte granularity within a 4 KiB page. They apply only to otherwise write-protected 4 KiB EPT mappings whose leaf enables SPP.

$$4 \text{ KiB page} / 128 \text{ bytes} = 32 \text{ sub-pages}$$

Each page has a 64-bit SPP vector. Address bits 11:7 select one of the page's 32 sub-pages. Bit $2S$ controls sub-page S ; odd vector bits are reserved.

The vectors are found through VMM-owned sub-page-permission tables rooted at SPPTP.

Why 128-byte regions exist

WHY NOT 32 BITS IN EVERY EPT LEAF?

An EPT entry has limited architectural space and most pages do not need sub-page policy. A separate, opt-in hierarchy keeps ordinary EPT leaves compact and charges extra memory and walk cost only to pages that use the feature.

- SPP does not override a missing mapping, does not apply to large EPT pages, and is excluded from guest page-table accessed/dirty updates and transactional-region writes. These restrictions keep the permission decision precise when an operation could span pages, create multiple writes, or require special atomic semantics.

WHY 128 BYTES RATHER THAN BYTES?

Fine granularity increases metadata, lookup bandwidth, and complexity. A 128-byte region gives meaningfully finer control than a page while keeping one page's policy in a compact vector.

Accessed and dirty flags

When enabled in EPTP, hardware maintains accessed and dirty flags in EPT entries:

- an entry's accessed bit is set when that entry participates in an EPT translation;
- a leaf's dirty bit is set when the mapped guest physical page is written.

The bits are sticky: hardware sets them but software clears them.

CACHE SUBTLETY

If software clears a bit while a translation remains cached, a later access may use the cached translation and not set the in-memory bit again. Starting a new observation epoch requires the prescribed invalidation.

GUEST PAGE-TABLE WALKS

The processor may update the guest page tables' own A/D bits while walking them. To EPT, those are writes to guest physical memory, so a walk can cause an EPT violation even when the instruction only reads data.

Page-modification logging

PML builds on EPT dirty bits. When a leaf dirty bit changes from 0 to 1, hardware records the page-aligned guest physical address in a 4 KiB log containing 512 entries and decrements the VMCS PML index.

When the index is outside 0..511, the log is full. The next event that needs a log entry causes a PML-full VM exit before setting the bit or performing the triggering guest physical access. The VMM drains or replaces the log, resets the index, and resumes.

Why log only the 0 → 1 transition? Once a page is known to be dirty for the current epoch, recording every later write adds no information.

Live migration with PML

- 1 Copy the guest's memory while it runs.
 - 2 Clear dirty state for the next collection epoch and perform the required EPT invalidation.
 - 3 Let hardware log pages whose dirty bit changes from zero to one.
 - 4 On a PML-full exit or periodic collection, drain the logged guest physical page numbers.
 - 5 Recopy those pages and repeat until the dirty set is small enough for a short final pause.
- Software still owns epoch management, log draining, dirty-bit clearing, concurrency, and the final consistency point.

Page tables and target data have different memory types

- EPTP specifies how the processor accesses the EPT hierarchy itself, normally write-back memory.
- The final EPT entry supplies the EPT memory type for the mapped page.
- Unless the leaf says to ignore PAT, the final effective type combines the EPT type with the guest's PAT-selected type according to the architectural rules.

Page-table memory is ordinary metadata and should normally be cacheable even when the page being described is device memory.

- Memory typing controls caching, ordering, and the behavior expected by memory-mapped devices. Changing a cached EPT mapping's memory-type interpretation requires invalidation just as changing its physical destination does.

05

Translation caches and invalidation

What the processor may cache

CACHED FAMILY	MAPS	DEPENDS ON
Linear mapping	Linear address to a final physical result when EPT is not part of the mapping	Ordinary paging, VPID, PCID
Guest-physical mapping	Guest physical address toward a machine physical result	EPT context
Combined mapping	Guest linear address directly to the final machine physical result	Guest paging and EPT; VPID, PCID, EPT context

Hardware need not replay both page walks for every access. The same classification explains why invalidation is split into two instructions.

Two invalidation instructions

INVVPID · LINEAR SIDE

Invalidates linear and combined mappings selected by VPID and, depending on the type, linear address. Not required to invalidate pure guest-physical mappings, because those do not depend on a guest linear context.

- Both change no page tables and flush no ordinary data cache lines. They only invalidate translation metadata on the logical processor that executes them.

INVEPT · EPT SIDE

Invalidates guest-physical and combined mappings selected by EPT context. Not required to invalidate pure linear mappings, because those contain no EPT information.

INVVPID invalidation scopes

TYPE	NAME	REQUIRED SCOPE	TYPICAL USE
0	Individual address	One VPID and one linear address, across all PCIDs and all EPT contexts	Emulating an INVLPG -like operation
1	Single context	All linear and combined mappings for one VPID, across all PCIDs and EPT contexts	Before reusing a VPID
2	All contexts	All linear and combined mappings for all nonzero VPIDs	Allocator rollover, or stale set unknown
3	Single context retaining globals	One VPID, but mappings used for global translations need not be invalidated	Modeling a guest MOV CR3

■ The processor may invalidate more than the minimum required, so software can depend only on the scope guaranteed by the selected type.

Three invalidation families

INSTRUCTION	SELECTION IDENTITY	MAIN USE
INVPCID	PCID and optionally a linear address	Process-address-space invalidation inside the current VPID context
INVVPID	VPID and optionally a linear address	VMM invalidation of a virtual linear context
INVEPT	EPT context	Invalidation after changing GPA-to-machine mappings or EPT EPT permissions

Changing a guest PTE from one GPA to another is a linear-side event handled by the guest's ordinary invalidation protocol. Changing an EPT leaf from one machine page to another is an EPT-side event and requires the applicable INVEPT; INVVPID alone is not the right architectural guarantee.

Tightening versus relaxing permission

1 → 0: permission is tightened

0 → 1: permission is relaxed

TIGHTENING · 1 → 0

A stale cached writable translation would violate the new policy, so the VMM must perform the required INVEPT before allowing affected execution to continue.

RELAXING · 0 → 1

A stale translation can only be too restrictive. If the VMM does not invalidate, the guest may take one unnecessary EPT violation. That violation invalidates the relevant cached mapping; retrying can then observe the relaxed entry.

- No invalidation is needed when repairing an entry that was previously non-present or misconfigured: the processor does not cache a usable mapping derived from such an entry.

Stale permissive state threatens correctness; stale restrictive state usually threatens only performance.
performance.

Why TLB shutdowns still exist

INVEPT and INVVPID act on the logical processor that executes them. If the same EPT context has run on several physical CPUs, those CPUs may each retain translations.

Tightening a mapping therefore requires the VMM to coordinate invalidation across all relevant logical processors: a TLB shutdown.

Tags reduce how often software flushes. They do not make shared cached state coherent with arbitrary software page-table writes.

06

The APIC before virtualization

The machine's interrupt switchboard

The APIC routes an interrupt to a logical processor, assigns it a vector and priority, and tracks it until software finishes servicing it. Two components traditionally divide the work:

COMPONENT	ROLE
I/O APIC	Converts platform interrupt inputs into interrupt messages and routes them to selected processors
Local APIC	One per logical processor; receives messages, prioritizes pending vectors, and presents an eligible interrupt to its core

Modern PCIe devices usually use MSI or MSI-X instead of a physical interrupt line. An MSI is a specially formatted write-like message whose address and data identify an interrupt destination and vector.

From device event to handler

device event

- I/O APIC or device-generated MSI/MSI-X
- destination local APIC
- vector becomes pending
- processor enters IDT[vector]
- operating-system handler

services the source

The message reaches the destination local APIC, which applies masking and priority rules before the core vectors through its IDT.

An operating system writes the local APIC's interrupt-command register to send an inter-processor interrupt (IPI) for scheduling, cross-CPU function calls, stopping CPUs, or TLB shutdowns.

Request, acknowledgement, and EOI

- 1 Requested:** its vector is recorded in the Interrupt Request Register (IRR).
 - 2 Acknowledged and accepted:** the APIC supplies the vector to the processor, which moves it from pending state into the In-Service Register (ISR) and enters the vector's handler.
 - 3 Completed:** after servicing the underlying source, software writes End Of Interrupt (EOI), allowing the APIC to remove the current in-service indication and reconsider lower-priority requests.
-
- EOI does not clear the device's condition: the driver must drain a completion queue or clear a device status bit as appropriate.

A vector is a prompt to inspect device state

Acknowledgement means that the CPU has accepted the interrupt and acquired its vector. EOI means that software has finished the corresponding interrupt-service interval.

For a level-triggered input, the device keeps the condition asserted until software clears it. If the handler sends EOI while the condition remains asserted, the I/O APIC may request the interrupt again.

MSI and MSI-X are message-signaled rather than level wires, but the same principle remains: the vector is a prompt to inspect device state, not a complete description of the work.

A vector identifies a handler, not a unique event

SHARED VECTORS

Several interrupt sources can use the same vector: legacy shared lines, or multiple device queues when vectors are scarce. The common handler checks the possible sources to determine which ones have work.

COALESCING

APIC pending state is fundamentally a bitmap. Repeated requests may coalesce into “ vector 0xD1 is pending.” The device records the real work in status registers, descriptor rings, or completion queues.

PER-CPU STATE

The same vector number can be pending or in service independently on different CPUs, because each logical processor has its own local APIC state.

| A vector supplies a priority class and an IDT index; it is not a globally unique event identifier.

Baseline VMX: who receives a physical external interrupt?

The VMCS pin-based **external-interrupt exiting** control makes the coarse first decision for ordinary maskable physical interrupts:

CONTROL	PHYSICAL INTERRUPT RECOGNIZED DURING GUEST EXECUTION
0	Delivered according to guest interruptibility through the guest's IDT
1	Causes a VM exit regardless of guest RFLAGS.IF

The host needs reliable access to its scheduler timer, device interrupts, IPIs, and other physical events. If all physical interrupts were delivered through the guest IDT, a guest with IF=0 , an elevated TPR, or an unrelated handler at that vector could delay or mishandle an event that belongs to the host.

Acknowledge interrupt on exit

The separate **acknowledge interrupt on exit** control determines how the exiting interrupt is acquired:

SET TO 1

The processor acknowledges the physical interrupt controller during the VM exit and records the acquired vector in vector in VM-exit interruption information. The VMM's exit path exit path dispatches the already-acknowledged vector.

SET TO 0

The interrupt remains pending at the physical controller. VM exit begins with host interrupts disabled; when the host later enables them, the interrupt is acknowledged and vectored normally through the host IDT.

- The all-or-nothing policy is refined by APICv. Ordinary host interrupts still exit, while one configured posted-interrupt notification vector receives special hardware treatment and can deliver approved virtual interrupts without an ordinary VM exit.

07

Virtualizing the local APIC

Why the APIC is a virtualization hot path

An operating system changes task priority, acknowledges interrupts with EOI, programs local timers, reads APIC state, and sends IPIs for scheduling and TLB shootdowns. If every APIC access caused a VM exit, workloads with many interrupts or vCPUs would spend substantial time crossing the VM boundary.

- 1 Trap APIC accesses and let the VMM emulate them.
 - 2 Redirect selected accesses to a virtual-APIC page.
 - 3 Let hardware update interrupt priority and pending/in-service state.
 - 4 Let hardware deliver eligible virtual interrupts inside non-root operation.
 - 5 Let devices or other CPUs post interrupts directly to a vCPU and notify the physical CPU running it.
- The features are separate controls because a VMM may want only some layers, and older processors may implement only a subset.

Four objects with different jobs

OBJECT	WHAT IT REPRESENTS	WHO PRIMARILY USES IT
Real local APIC	The physical CPU's interrupt controller	Host and processor
APIC-access page	The physical page recognized as the guest's memory-mapped APIC target	Address-recognition/interception mechanism
Virtual-APIC page	4 KiB backing state for the guest's emulated APIC registers	Processor and VMM
Posted-interrupt descriptor (PID)	Concurrent queue and notification information for interrupts destined to one vCPU	Devices, CPUs, processor, and VMM

The APIC-access page is not the register file

APIC-ACCESS PAGE

The page whose address causes special treatment when the guest uses xAPIC-style memory-mapped accesses. Selected operations are redirected to the virtual-APIC page; unsupported or policy-sensitive operations exit.

VIRTUAL-APIC PAGE

Stores virtual TPR, PPR, EOI, IRR, ISR, and ICR state. Lets hardware perform many APIC state transitions without touching the host's real APIC registers.

PID

Separate because it must be safely updated by agents outside the running guest, such as another processor or an interrupt-remapping unit. Its atomic protocol is designed for concurrent producers.

Virtual APIC state

VIRTUAL-APIC PAGE

- VTPR: virtual task-priority register
- VPPR: virtual processor-priority register
- VIRR: 256 pending bits, one per vector
- VISR: 256 in-service bits, one per vector
- VEOI: virtual EOI register
- VICR_LO / VICR_HI: virtual interrupt-command register

VMCS GUEST INTERRUPT STATUS

- RVI , the request-vector value: the highest-priority pending pending virtual vector known to hardware
- SVI , the service-vector value: the highest-priority virtual currently in service

VIRR and VISR are full sets. RVI and SVI are cached summaries that let the processor make the hot priority comparison without scanning 256 bits at every instruction boundary.

TPR, PPR, and vector priority

```
vector 0x52 → priority class 5  
vector 0xD1 → priority class 13
```

For APIC interrupts, a vector's high nibble is its priority class. VTPR expresses the priority threshold requested by guest software. SVI represents the highest virtual interrupt already in service. Hardware derives VPPR from the higher of those two priority classes.

This prevents a pending interrupt of the same or a lower class from nesting merely because the guest set a low TPR while a higher-priority handler is already active.

WHY ONLY THE CLASS

The low nibble selects a vector within a class but does not make it preempt another vector in the same class. This keeps the comparison small and predictable while leaving 16 vector identities in each class.

When a pending virtual interrupt is recognized

With virtual-interrupt delivery enabled, evaluation is triggered by VM entry, TPR virtualization, EOI virtualization, self-IPI virtualization, IPI virtualization, and posted-interrupt processing. A pending interrupt is recognized when:

```
RVI priority class > VPPR priority class
and interrupt-window exiting is disabled
```

Delivery occurs at an instruction boundary only when:

IF	STI	MOV/POP SS	WINDOW
Guest RFLAGS.IF is 1.	There is no STI blocking.	There is no MOV/POP-SS blocking.	Interrupt-window exiting is disabled.

Virtual-interrupt delivery, step by step

- | | |
|---|--|
| 1 | Select RVI as the vector. |
| 2 | Set the corresponding VISR bit. |
| 3 | Record the vector in SVI. |
| 4 | Update VPPR to the vector's priority class. |
| 5 | Clear the corresponding VIRR bit. |
| 6 | Recompute RVI from the remaining pending bits. |
| 7 | Deliver the vector through the guest's ordinary interrupt-delivery architecture. |

The guest reaches its IDT or FRED handler without a VM exit and without VM-entry event injection.

Evaluation and delivery are separate

Priority state can say that an interrupt should be next while architectural interruptibility says that it cannot be delivered at this particular boundary. For example, $IF=0$ or the one-boundary shadow after STI may temporarily block it.

The interrupt remains pending. Hardware does not force a maskable virtual interrupt through $IF=0$, STI blocking, MOV/POP-SS blocking, or an insufficient priority threshold.

OLDER VM-ENTRY INJECTION

The VMM must not request an illegal injection. It can enable interrupt-window exiting so the CPU exits at the first boundary where the guest can accept a maskable interrupt, then inject it on the next VM entry.

WITH APICV

Leave the vector in virtual pending state and let hardware deliver it when both priority and interruptibility allow.

TPR virtualization and the threshold exit

In 64-bit mode the guest may access APIC task priority through CR8. With TPR shadowing enabled:

- MOV from CR8 reads the priority class from VTPR;
- MOV to CR8 updates VTPR and triggers TPR virtualization.

WITHOUT VIRTUAL-INTERRUPT DELIVERY

Hardware compares the new virtual priority with the VMCS TPR threshold. If it fell below, a **TPR-below-threshold VM exit** occurs after the instruction completes. It is trap-like: lowering the threshold may make queued virtual work deliverable.

WITH VIRTUAL-INTERRUPT DELIVERY

Hardware instead recomputes VPPR and evaluates pending interrupts itself.

EOI virtualization

- | | |
|---|---|
| 1 | Take the vector from SVI. |
| 2 | Clear its bit in VISR. |
| 3 | Find the next-highest in-service vector and update SVI. |
| 4 | Recompute VPPR. |
| 5 | Either evaluate pending virtual interrupts or cause an EOI-induced VM exit. |

THE EOI-EXIT BITMAP · ONE BIT PER VECTOR

Some virtual interrupts need no software action when their handler finishes. Others may be associated with emulated level-triggered devices whose source must be unmasked or whose remote-IRR state must be updated. The bitmap lets the VMM pay only for vectors with an external consequence.

- An EOI-induced exit is trap-like. The guest's EOI write has completed and the virtual in-service state has already been updated.

xAPIC MMIO versus x2APIC MSRs

xAPIC · MEMORY-MAPPED

The processor recognizes translations to the APIC-access page. Depending on the access and controls, it may:

- satisfy a read from the virtual-APIC page;
- write the corresponding virtual-APIC field and perform hardware emulation;
- complete the write and then cause an APIC-write exit;
- cause an APIC-access exit before the access completes.

x2APIC · MSRs 0x800–0x8FF

With x2APIC virtualization and the supporting controls, selected RDMSR and WRMSR operations are redirected to virtual-APIC fields. TPR, EOI, self-IPI, and qualifying ICR writes invoke the same virtual state machines used by the MMIO interface. Unsupported encodings or operations that require richer routing policy still fault or exit.

APIC-write exits versus APIC-access exits

Basic aligned accesses to supported APIC fields get the fast path. Odd sizes, cross-boundary accesses, unsupported registers, or complex instruction forms may exit so software can preserve exact architectural behavior.

This split is another hardware design compromise: accelerate common compiler-generated loads and stores without building a general-purpose instruction emulator into the page walker.

APIC-WRITE EXIT · TRAP-LIKE

Generated by post-operation APIC-write emulation. Saved guest RIP points after the operation.

APIC-ACCESS EXIT · INTERCEPTION

The handler must not assume that the access and its architectural effects have already completed.

Self-IPIs and IPIs

A self-IPI is simple: set the vector's bit in VIRR, raise RVI if necessary, and evaluate pending interrupts. No destination lookup is required.

An IPI to another vCPU must translate a guest-visible virtual APIC ID into the target vCPU's interrupt queue. IPI virtualization uses a VMM-provided PID-pointer table indexed by virtual APIC ID; a valid entry points to that vCPU's posted-interrupt descriptor.

The fast path requires a legal vector and a straightforward fixed, physical-destination delivery. Complex destination modes, shorthands, or invalid table entries cause an APIC-write exit.

Fixed physical-destination IPIs reduce to a table lookup, an atomic bit set, and possibly a notification. Useful for reschedule and virtual TLB-shutdown IPIs on large guests.

08

Posted interrupts and VT-d

Posted interrupts: direct delivery without a VM exit

CONVENTIONAL PATH

```
physical interrupt → VM exit → VMM records virtual interrupt  
→ VM entry with injection → guest handler
```

With posted interrupts, a producer records the virtual vector in the target vCPU's PID and, when needed, sends an ordinary physical notification interrupt to the physical CPU running that vCPU.

- VM-entry injection says “deliver this particular event on the next transition into the guest.” Posting says “queue this maskable vector, notify its physical CPU if necessary, and let hardware deliver it when eligible.”

Posting is not delivery

POSTING · PRODUCER SIDE

Place a vector into a vCPU's durable pending bitmap and notify the physical CPU if necessary.

VIRTUAL-INTERRUPT DELIVERY · CONSUMER SIDE

Compare virtual APIC priority and guest interruptibility, then enter the guest's handler when the posted vector becomes eligible.

Posting does not mean that the handler has already run. A vector can be posted, transferred into VIRR , eligible by priority, and finally delivered at four different moments.

Posted-interrupt descriptor fields

FIELD	MEANING
PIR[255:0]	One posted-request bit per virtual vector
ON	Outstanding-notification bit
SN	Suppress-notification bit used by virtual-interrupt posting
NV	Physical notification vector
NDST	Physical APIC destination for that notification

The producer atomically sets PIR[vector]. If both ON and SN were clear, it atomically changes ON from zero to one and sends the notification described by NV and NDST.

Why ON and SN exist

ON · COALESCING

Without ON , ten interrupts posted close together could produce ten physical notifications even though one notification can make the target collect all ten PIR bits.

```
first producer:
  ON 0 → 1, send notification
later producers:
  ON already 1, set PIR bits,
  send nothing
```

The target clears ON atomically, then collects the pending bits.

SN · SUPPRESSION

The VMM may know that notifications should not be sent, for example while a vCPU is not running and its PID routing information is being managed. Setting SN tells producers to record requests without sending the physical notification.

SN suppresses the notification, not the virtual PIR bit remains the durable pending record.

What the physical CPU does with a notification

Posted-interrupt processing is enabled together with external-interrupt exiting. Normally a physical external interrupt would cause a VM would cause a VM exit. The posted-notification vector is the special case:

- 1** The real local APIC supplies a physical vector.
- 2** If it is not the configured posted-interrupt notification vector, the ordinary external-interrupt exit occurs.
- 3** If it matches, hardware clears `ON` atomically and EOIs the physical notification.
- 4** Hardware ORs `PIR` into the virtual APIC's `VIRR` and clears the collected `PIR` bits atomically.
- 5** It updates `RVI` and evaluates pending virtual interrupts.
- 6** If the guest is interruptible and the virtual vector outranks `VPPR`, the interrupt may be delivered immediately inside non-root operation.

From a posted bit to the guest handler

producer atomically sets PIR[0xD2]

- ON changes 0 → 1
- physical notification NV is sent to the pCPU running vCPU 3
- CPU recognizes NV as a posted-interrupt notification
- PIR[0xD2] moves into virtual IRR; ON is cleared
- virtual priority and IF/blocking conditions are checked
- guest receives vector 0xD2 without a VM exit

If the guest has IF=0, is in an interrupt shadow, or has a priority that blocks class 0xD , the vector remains pending in virtual APIC state. It is not lost and is not forced through an illegal boundary.

VT-d: securely connecting a device to a vCPU

A PCIe device normally signals MSI or MSI-X with a write-like transaction whose address and data would traditionally encode a physical APIC destination, vector, and delivery information. Giving a guest unrestricted control of those values would let an assigned device target host CPUs or vectors chosen by the guest.

VT-D FACILITY	QUESTION IT ANSWERS
DMA remapping	Which host-physical memory may this device read or write?
Interrupt remapping	Which interrupt destination and vector may this device request?

The interrupt remapping table

The VMM builds an Interrupt Remapping Table in protected host memory. Each entry (IRTE) contains trusted routing and validation state. A device is programmed with a remappable-format MSI that selects an entry,

conceptually a table handle.

Source validation binds an entry to an expected PCI requester ID.

A request from the wrong device is blocked and reported as an interrupt-remapping fault, which prevents one assigned device from borrowing another device's handle.

device MSI containing

remapping handle

→ VT-d selects IRTE[handle]

→ validate PCI requester
identity

→ use trusted routing
information in the IRTE

→ emit an ordinary physical
interrupt or post a

virtual one

Inside the interrupt remapping table

A VT-d register (IRTA) gives the table's host-physical base and size: up to 65,536 entries of 16 bytes each. A remappable MSI carries a handle that indexes one entry. The IRTE mode bit selects one of two formats:

REMAPPED FORMAT · IM = 0

Present	Entry is valid; otherwise the request faults
Vector	Host physical vector to deliver
Destination ID	Physical APIC ID of the target CPU
Delivery / trigger	Fixed, lowest-priority, NMI...; edge or level
Destination mode	Physical or logical addressing

POSTED FORMAT · IM = 1

Present	Entry is valid; otherwise the request faults
Vector	Guest vector, set in the PID's PIR
PID address	Host-physical address of the target vCPU's posted-interrupt descriptor
Urgent	Notify even if the PID's SN is set

COMMON · SOURCE VALIDATION AND FAULTS

Source ID	Expected PCI requester ID (bus, device, function)
SVT / SQ	How strictly to compare the requester ID: exact, masked, or by bus range
FPD	Suppress fault reporting for this entry

Field layout from the Intel VT-d specification rather than the SDM; see it for exact bit positions.

Remapped versus posted-mode entries

CONVENTIONAL REMAPPED

The IRTE supplies a host physical vector and physical APIC destination. If that CPU is running a guest with external-interrupt exiting enabled, the result is still a VM exit.

```
device → VT-d validation
  → host vector → VM exit
  → host/VMM handler
  → virtual interrupt injection
```

POSTED MODE

The IRTE identifies the target vCPU's PID and a guest-visible vector. The remapping unit sets that PIR bit and follows the PID

```
assigned device → remappable MSI-X
  → VT-d validates, reads IRTE
  → set PID.PIR[guest vector]
  → if ON=0 and not suppressed,
      set ON and notify NDST
  → VMX moves PIR into virtual IRR
  → guest IDT[guest vector]
      when eligible
```

Two vectors participate

GUEST VECTOR · e.g. 0xD1

Posted in PIR and eventually indexes the guest IDT.

PHYSICAL NOTIFICATION VECTOR · e.g. 0xF2

A VMM-selected doorbell sent to the real local APIC. VMX consumes it specially; it is not delivered as 0xF2 to the guest.

The fast path is not “give the guest every physical interrupt.” It is “let VT-d validate this device request, map it to request, map it to this vCPU and virtual vector, and let APICv perform delivery.”

Passed-through NIC: configuration

Assume NIC function `0000:5e:00.1` is assigned to a VM, receive queue 4 belongs to guest vCPU 2, and the guest expects vector `0xD1`. The VMM configures an IRTE with:

```
expected requester  = 0000:5e:00.1
delivery form       = posted
guest vector        = 0xD1
posted descriptor   = PID for vCPU 2
```

If vCPU 2 is running on physical CPU 12, its PID might specify physical destination 12 and notification vector `0xF2`.

Passed-through NIC: a packet arrives

- 1 The NIC emits the queue's remappable MSI-X request.
 - 2 VT-d selects the IRTE and validates requester `0000:5e:00.1`.
 - 3 VT-d sets `PIR[0xD1]` in vCPU 2's PID.
 - 4 If no notification is outstanding, VT-d sets `ON` and sends physical vector `0xF2` to CPU 12.
 - 5 VMX recognizes `0xF2` as the configured notification, rather than taking the ordinary external-interrupt exit.
 - 6 Hardware moves `0xD1` into virtual pending state and applies guest IF, interrupt-shadow, VTPR, and VPPR rules.
 - 7 When eligible, the guest enters `IDT[0xD1]` and its NIC driver drains queue 4.
- If queue 4 produces more work while `0xD1` is already pending, requests may coalesce into the same bit.

Scheduling, migration, and remapping caches

MIGRATION

Several IRTEs can point to the same vCPU PID. Its NDST identifies the physical CPU to notify. When the scheduler moves the vCPU, the VMM updates this one destination instead of rewriting every device entry.

DESCHEDULED vCPU

The VMM can suppress notifications while leaving requests recorded in PIR, and must arrange a wakeup or process the pending state before or during the next run.

IRTE CACHING

VT-d may cache IRTEs. After changing routing or permissions, the VMM must use the VT-d invalidation protocol and observe its completion ordering, just as with page tables.

Virtualizing user interrupts and SENDUIPI

Intel user interrupts let user-mode software receive and send selected notifications without the traditional kernel interrupt path. That machinery contains its own notification vectors and posted-interrupt descriptors, so a guest's use of it must compose with APIC virtualization.

When a delivered virtual APIC vector is the guest's user-interrupt notification vector, hardware can perform the virtual EOI-like bookkeeping and continue into user-interrupt notification processing without exposing the host APIC.

For SENDUIPI , VMX transforms the request into the corresponding virtual IPI operation: write virtual ICR state, post through the PID-pointer table, or exit when software must handle the case.

Accelerating user interrupts while trapping every underlying APIC notification would move the bottleneck rather than remove it.

09

Putting Chapters 31 and 32 together

APIC-access recognition depends on translation caches

The processor recognizes the APIC-access page after address translation. If the VMM changes the APIC-access address, enables APIC-access virtualization, or changes a mapping so that an address now reaches that page, an old cached translation can bypass the newly intended treatment.

The required invalidation depends on which translation dimension changed:

INVVPID

For affected linear/combined mappings when VPID-side state is relevant.

INVEPT

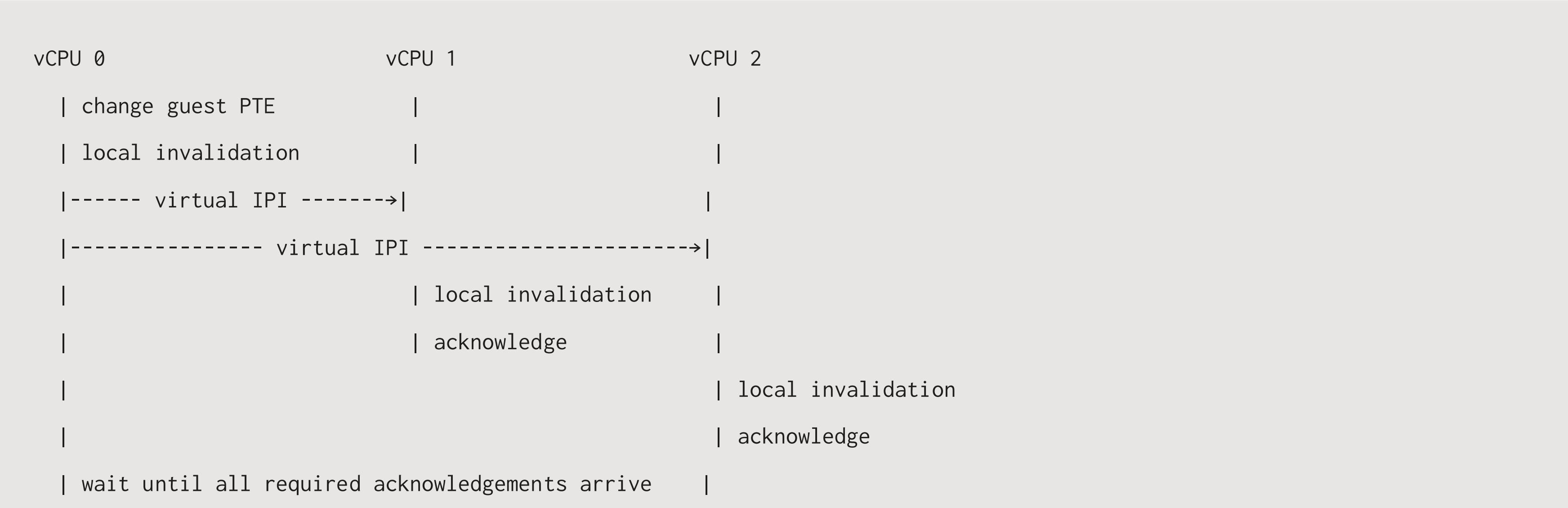
For affected guest-physical/combined mappings when EPT state is relevant.

Guest TLB shutdowns exercise both features

Suppose vCPU 0 removes the guest mapping GVA 0x4000 -> GPA 0x9000 , which other vCPUs may have cached. The guest OS typically:

- 1 Modifies the guest page-table entry.
- 2 Invalidates the local translation on vCPU 0.
- 3 Sends a virtual IPI to every other vCPU that may have used the address space.
- 4 Each target runs the guest's TLB-shutdown interrupt handler.
- 5 Each handler executes INVLPG, INVPCID, a flushing MOV CR3, or another local invalidation.
- 6 Each target acknowledges completion in guest memory.
- 7 The initiating vCPU waits for the required acknowledgements before reusing the old page.

The shutdown protocol



| continue safely |

If vCPU 0 uses VPID 101 and vCPU 1 uses VPID 102, vCPU 0's local INVLPG cannot remove vCPU 1's VPID-102 entry. The guest neither knows the VPID values nor normally executes INVVPID.

Two paths for the virtual IPI

WITHOUT IPI VIRTUALIZATION

The guest ICR write exits. The VMM identifies the target vCPUs, records the virtual vector as pending, and kicks or schedules each target. A running target may first exit to the VMM and then re-enter with the interrupt injected.

- A target vCPU that is not running cannot consume the stale translation. The VMM retains the pending virtual IPI and makes it runnable. If guest interrupts are disabled, the IPI remains pending and the sender keeps waiting, just as on physical hardware.

WITH APICV AND IPI VIRTUALIZATION

Hardware locates the target PID, sets the shutdown vector in its vector in its posted-interrupt requests, sends a notification when when needed, and delivers the vector through the target's virtual virtual APIC without an ordinary VM exit.

When does the VMM need INVVPID here?

The VMM usually does not need to execute INVVPID for this ordinary guest protocol when EPT is enabled and guest invalidation instructions execute natively. It may need INVVPID if it:

INTERCEPTS	SHADOWS	REUSES	PARAVIRTUALIZES
Intercepts and emulates those operations.	Maintains shadow page tables.	Reuses a VPID.	Provides a paravirtualized shutdown interface.

A paravirtualized design may replace many virtual IPIs with a hypercall and per-vCPU TLB generation numbers, marking a stopped vCPU to flush before it next continues.

Guest shutdown versus VMM EPT shutdown

MAPPING CHANGED	INITIATOR	CROSS-CPU MECHANISM	LOCAL INVALIDATION
Guest linear to guest physical	Guest OS	Virtual IPIs to affected vCPUs	Guest INVLPG, INVPCID , or equivalent
Guest physical to machine physical, or EPT permission	VMM	Host IPIs to affected physical CPUs	Host INVEPT

If the VMM changes GPA 0x9000 from machine page A to machine page B, guest INVLPG is not the required EPT invalidation. The VMM must execute the appropriate INVEPT on every logical processor that may retain mappings for that EPT context. A normal guest page-table change does not require the guest to know about EPT or INVEPT.

The guest keeps using the real architectural multiprocessor protocol, while VPID, EPT, APICv, and posted interrupts confine and accelerate its effects.

A handler-oriented summary

CHAPTER 31 EVENT

- 1 Which address exists at this stage: guest linear, guest physical, or machine physical?
- 2 Is this a valid policy denial or an invalid EPT encoding?
- 3 Did the denied access complete, or should the instruction retry after repair?
- 4 Which cached family could now be stale: linear, guest physical, or combined?
- 5 Is permission becoming more restrictive or merely more permissive?
- 6 Could another physical CPU retain the same context?

CHAPTER 32 EVENT

- 1 Is this a physical host interrupt, a virtual guest vector, or a posted-notification doorbell?
- 2 Is the vector in PIR, VIRR, or VISR: posted, pending, or in service?
- 3 Does its priority class exceed VPPR?
- 4 Is delivery blocked by IF , STI, MOV/POP SS, or interrupt-window exiting ?
- 5 Did hardware complete the APIC write before exiting?
- 6 Does this vector's EOI require device-model work?

These prevent the common mistakes: treating every address as the same address, every interrupt bit as the same state, and every VM exit as though the triggering operation had not run.

The design lesson

Chapter 31 gives hardware enough identity and permission metadata to cache the composed result safely. Chapter 32 gives hardware enough virtual APIC and notification state to execute the common state transitions safely. Both use the same three-part design:

-
- 1 Software defines policy** through VMCS controls and memory-resident structures.
 - 2 Hardware executes the common case** through bounded walks, bitmaps, priority comparisons, and atomic state transitions.
 - 3 Exceptional or ambiguous cases exit** with enough information for software to finish the job.
-

These features look more complex than a simple “trap or do not trap” switch. Their purpose is to make the frequent path native while keeping every cached identity, denied access, priority transition, and notification race architecturally accountable.

SOURCE AND SCOPE

Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3C

Order Number 325384-092US.

PRIMARY READING

- 31** **Chapter 31** : VMX support for address translation: VPID, HLAT, EPT, EPT violations and misconfigurations, SPP, accessed and dirty flags, PML, EPT memory types, cached translation information, INVVPID, INVEPT
 - 32** **Chapter 32** : APIC virtualization and virtual interrupts: virtual-APIC page, TPR and EOI virtualization, virtual-interrupt delivery, APIC-access and x2APIC virtualization, IPI virtualization, posted-interrupt processing, user-interrupt virtualization
-
- Production hypervisor code must follow the manual's exact pseudocode, priority rules, capability checks, reserved-bit requirements, errata, and processor-specific documentation.