IntelのIA32向けハードウェア仮想化機構

Intel Vanderpool Technology for IA32 Processors(VT-x)
Preliminary Specification

2005/2/22 **清水 正明**

Order Number C97063-001
January 2005

目次

1.Vanderpool Technologyの背景
2.VMXの概要
3.仮想マシンコントロールストラクチャ (VMCS)
4.VMX命令セット

1.Vanderpool Technologyの背景

1.1 仮想マシンモニタ(VMM)再開発の背景
1.2 VMMの分類
1.3 Intel IA32アーキテクチャの仮想化
1.4 SilvervaleとVanderpool
1.5 VanderpoolによるVMM構築例

1.1 仮想マシンモニタ(VMM)再開発の背景

- 1960年代末から1970年代にかけて熱心に研究された
  IBMもVM/370で仮想マシンを採用
- ハードウェアとOSの間の薄皮インタフェース(Hypervisor)
- “直接実行”向きのプロセッサアーキテクチャを開発

↑

- ハードウェアが高価だった
- OSがシングルユーザだったので複数人で使用するために
  VMMで多重化した

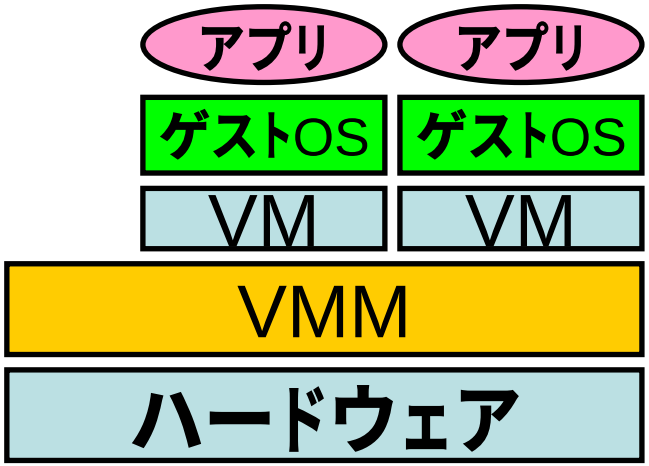
↓

ハードウェアのコストダウン＆ＯＳの機能向上で魅力低減

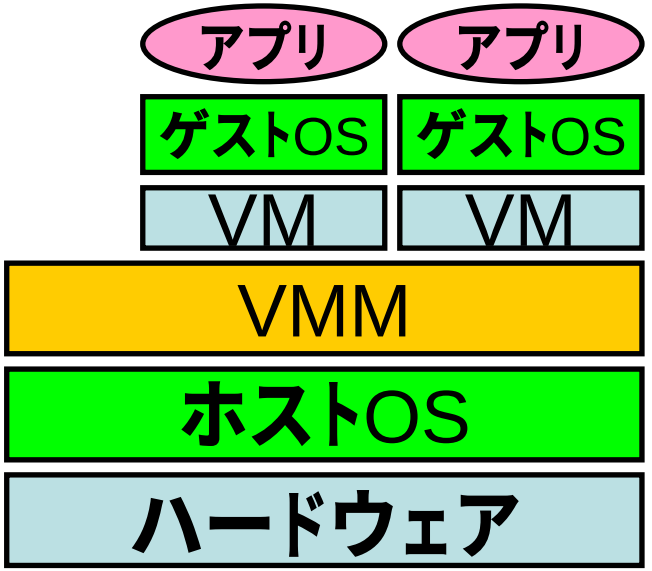
しかし近頃、フォールトトレランス、高セキュリティの実現、マルチコア・HMTのプロセッサを有効利用する基礎技術として再び脚光

1.2 VMMの分類

・type I VMM (ハードウェア上に直接VMMを実装)
・type II VMM (ユーザプロセスでVMMを実装)



Type I VMM
・メインフレーム
・Xen



Type II VMM
・VMware
・VirtualPC
・Bochs, PearPC

1.3 Intel IA32アーキテクチャの仮想化

・最大シェアのアーキテクチャであり、仮想化のニーズは最も高いが…
・8086からの改良アーキテクチャであり、仮想化に対する考慮が少ない
-スーパーバイザーモードを仮想化する際に問題となる命令が存在する(250命令中22命令)†

↓

VMwareでは命令のバイナリ変換を行って問題回避
(ただし性能面で不利、実CPUの50%程度の性能)

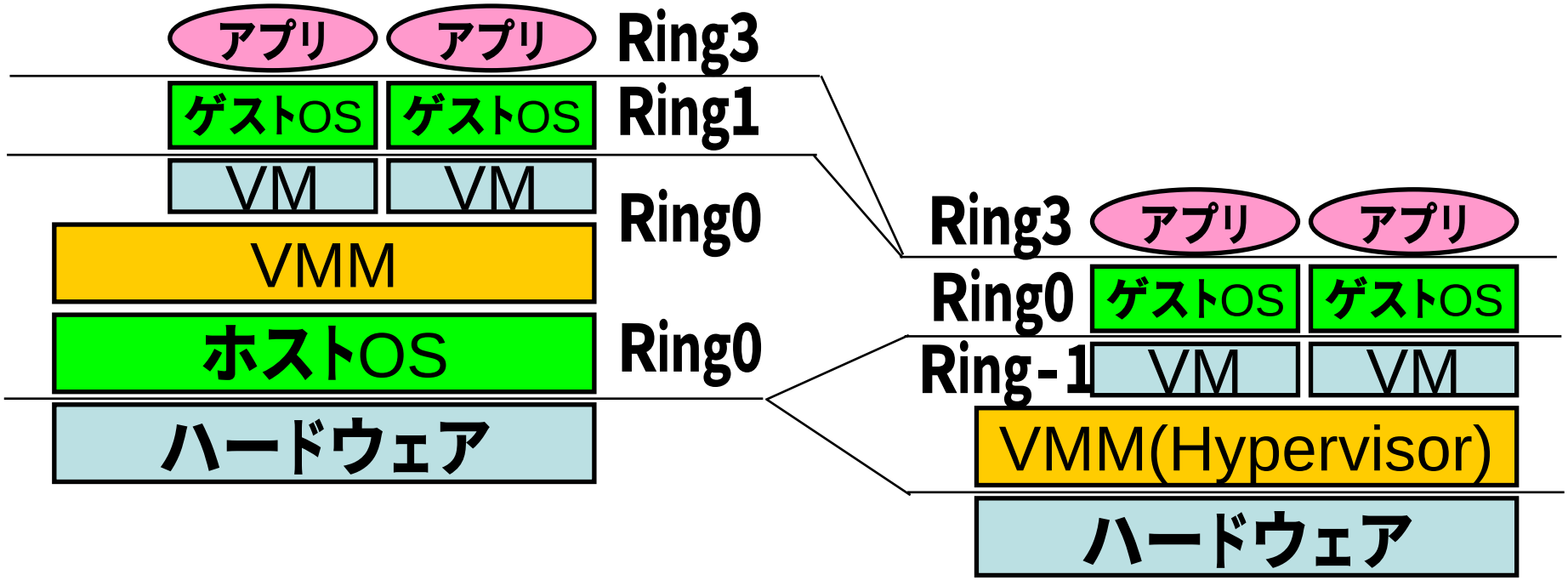
† Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor, John Scott Robin,etc, U.S. AirForce Naval Postgraduate School, USENIX Security 2000.

1.4 SilvervaleとVanderpool

・サーバ向け仮想化機能Silvervale
・PC向け仮想化機能Vanderpool
　　↓
アーキテクチャと名称をVanderpoolに統一？
2005/1/20にPreliminary Spec.を公開

Vanderpool概要
・仮想計算機モニタ実装のためのハードウェアアシスト
-Ring0より特権レベルの高いRing-1でVMMを動作
-VMモードのRing0の動作はVMにトラップされる
　　↓
単純な機構で堅牢な仮想マシンを提供可能

1.5 VanderpoolによるVMM構築例



VMware方式

- ゲストOSをRing1で動かす工夫が必要 (Ring情報隠蔽)

Vanderpool(Xen)方式

- ゲストOSはＶＭなしと同じ特権で動作
- ただしHypervisorにtrapされる

2.VMXの概要

2.1 仮想マシンアーキテクチャ
2.2 VMXオペレーション
2.3 VMMソフトウェアのライフサイクル
2.4 仮想マシンコントロールストラクチャ
2.5 VMXオペレーションモードのenableとenter
2.6 VMXオペレーションの制限

2.VMXの概要

Vanderpool Technology(VT-x)のためにIA32に拡張された機能がVMX(Virtual Machine Extensions)。

VMXは次の機能をサポート

・プロセッサハードウェアの仮想化
・複数のソフトウェア（OS）環境を提供

2.1 仮想マシンアーキテクチャ

VMXが定義するIA32プロセッサの仮想マシン

・仮想マシンモニタ(VMM)
  -プロセッサリソース (物理メモリ、割り込み、I/O)管理
  -プロセッサのフルコントロール可能

・ゲストソフトウェア
  -各仮想マシン(VM)はゲストソフトウェア環境を提供
    (OSとアプリケーションスタックを含む)
  -他の仮想マシンとは独立に動作
  -VMMがない場合のプロセッサと同じ環境
  -VMMよりも低い特権で動作

2.2 VMXオペレーション

二種類のVMXオペレーションモード
・VMX rootオペレーション (VMMが動作)
・VMX non-rootオペレーション(ゲストソフトウェアが動作)

-VMM(in VMX rootオペレーション)はVM-entry命令を発行することでVMX non-rootオペレーションモードに移行可能
-VMX rootオペレーションモードに戻る際にはVM exit命令

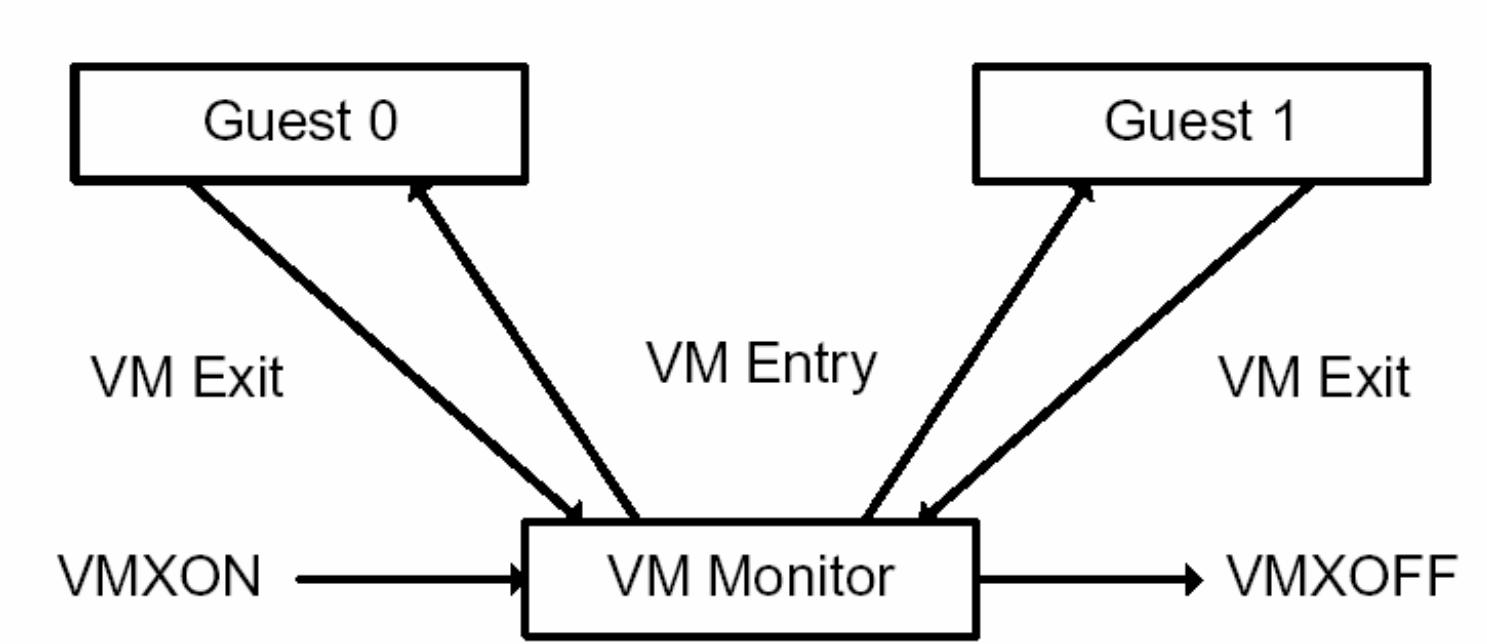
2.2 VMXオペレーション

・VMX rootオペレーション (VMMが動作)
 -Nativeプロセッサとほぼ同じ機能
 (違いはVMX命令の追加とプロセッサモードの制限)

・VMX non-rootオペレーション(ゲストソフトウェアが動作)
 -特定の命令やイベントでVM exitが発生しVMMに戻る
 (VMCALL命令もVM exitを起こす命令の一つ)
 -VMX non-rootオペレーションモードのソフトウェアが
 自身の実行モード(non-root)を知ることはできない
 -CPL0で動作するソフトウェアも仮想化される
 (CPL0で実行されるためVMMの負担が少ない)

2.3 VMMソフトウェアのライフサイクル

- VMXON命令実行でVMM rootモードに入る
- VM-entry命令実行でゲスト(non-root)モードに入る
- VM ExitでVMM rootモードに戻る
- VMXOFF命令で通常プロセッサモードに戻る



2.4 仮想マシンコントロールストラクチャ(VMCS)

VMCS(Virtual Machine Control Structure):

・VMX non-rootオペレーションモードと
VMXモード遷移を制御するデータ構造
-プロセッサはVM entry時にVMCSからVMの
コンテキストと設定を読み込む
-VM exit時のVMMのentry pointもVMCSに保持

・VMCSはVMCS pointerが指す(VMPTRLDで設定)
・VMCSの内容はVMREAD,VMWRITE,VMCLEARで操作

2.5 VMXオペレーションモードのenableとenter

・VMXオペレーションモードのENABLE
 -bit13 in CR4(CR4.VMXE)=1

・VMXオペレーションモードへのENTER
 -VMXON命令の実行
  (VMXON時に4KBのワークメモリを指定する)

VMXON中はCR4.VMXEのクリアは不可能
・VMXOFF後CR4.VMXEをクリアする

2.6 VMXオペレーションの制限

VMXON実行時に、次の条件を満たしている必要がある
・CR0.PE(Protect mode enable)
・CR0.NE(FPU error enable)
・CR0.PG(Paging enable)
・CR4.VMXE(VMX enable)
VMXオペレーションモードで上記bitをクリアしようとすると一般保護例外が発生する。VM-entry/VM exit命令でもクリアできない。

上記は、VMXの仮想化がページング＆プロテクトモードのみであることを意味している。ページングなしプロテクトモードorリアルアドレスモードの実現はエミュレーション要

3.仮想マシンコントロールストラクチャ (VMCS)

3.1 VMCSの基本構造
3.2 Guest-state area
3.2 Host-state area
3.3 VM-execution control fields
3.4 VM-exit control fields
3.5 VM-entry control fields
3.6 VM-exit information fields
3.7 VMCSのlaunch state

3.仮想マシンコントロールストラクチャ(VMCS)

VMCS：64bit、4KBアライン、物理アドレス

・VMPTRLDでアドレスを指定するとVMCSがactive
・複数のVMCSをactiveにできる
・最近にVMPTRLDしたVMCSが(通常)active
・VMPTRSTで現在activeなVMCSのアドレスを取得可

VMCS内部フォーマットは実装依存であり
アーキテクチャ上は定義されていない
　→VMCSアクセス命令を使用する

3.1 VMCSの基本構造

VMCSに含まれる6つの論理構造

・Guest-state area：
　　-VM-entry時に読まれ、VM-exit時に保存される
・Host-state area：
　　-VM-exit時にプロセッサに読み込まれる
・VM-execution control fields：
　　-VM-exitを起こすイベントの条件等の設定
・VM-exit control fields：
・VM-entry control fields：
　　-VM-exit,entryの制御
・VM-exit information fields
　　-VM-exit時の詳細情報を保存しているfield

3.2 Guest-state area

ゲスト(VM)の状態を保持している領域
(VM-entry時にプロセッサにロードされる情報)

・ゲストregister状態
  -CR0,CR3,CR4(paging, protect-mode, PAE,VME…)
  -RSP,RIP,RFLAGS(64bit, stack-pointer, instruction pointer,…)
  -CS,SS,DS,ES,FS,GS,LDTR,TR
  (セレクタ以外にbase, segment-limit, ARbyte等の hidden partも保存している)
・割り込み許可状態
  -外部割込み,NMIをブロックする/しない

3.2 Host-state area

ホストの状態を保持している領域
（VM-exit時にプロセッサにロードされる情報）

・ホストregister状態
- CR0,CR3,CR4(paging, protect-mode, PAE,VME…)
- RSP,RIP(64bit, stack-pointer, instruction pointer,…)
- CS,SS,DS,ES,FS,GS,TRのセレクタフィールド
- FS,GS,TR,GDTR,IDTRのベースアドレスフィールド

上記以外にVM exit時に固定設定されるregister状態もある。

3.3 VM-execution control fields

VMX non-rootオペレーションモードにおいて、
VM exitを起こす条件を設定するフィールド

- Pin-Based VM-Execution Controls
  - 外部割込み、NMIでVM exitする/しない
- Processor-Based VM-Execution Controls
  - 内部要因の同期イベントでVM exitする/しない
  (HLT,LNVLPG,MWAIT,RDTSC,CR8load,等の命令..)
- Exception Bitmap
  - IA32の各例外に1bit割り当て、VM exitの可否設定
- Page-Fault Controls
  - PageFault　error-codeについて詳細なマスク設定

3.3 VM-execution control fields

- I/O-Bitmap Address
  - 0x0000-0x7fff,0x8000-0xffffのI/O領域
    アクセスに対するVM exitの可否
- Time-Stamp Counter Offset
  - タイムスタンプレジスタにOffsetを付加する
- Guest/Host Masks and Read Shadows for CR0/CR4
  - CR0/CR4をHostが持ち、guestのwriteでVM exit,
    ReadでShadowレジスタを読ませることができる
- CR3-Target Controls
  - ページディレクトリを保持するCR3を4セット持つ
    既知のCR3のロードではVM exitを起こさない
- Controls for CR8 Accesses
  - Local APICのTPRの仮想化の可否

3.4 VM-exit control fields

VM exit時に行なう処理を制御するフィールド

- VM-Exit Controls
  - Hostに戻る際のmode, 32bit/64bit
  - 割り込みでVM exitする場合に割り込みにack を返すか否か
- MSRの保存
  - Guest指定のMSRを保存する
- MSRの読み込み
  - Host指定のMSRをロードする

3.5 VM-entry control fields

VM entry時に行なう処理を制御するフィールド

- VM-entry Controls
  - -Guestに入る際にIA32eモードにするか否か
    (前述のとおり、Pagingは常にenable)
- MSRの読み込み
  - -Guest指定のMSRをロードする
- VM-entry Controls for Event Injection
  - -VM entry時に指定の外部割込みを発生させる
    (割り込みベクタor NMI)

3.6 VM-exit information fields

VM-exit時の詳細情報を保持しているフィールド

- Basic VM-Exit Information
  - Exit reason：例外、HLT,INVPG,VMCALL,etc..
- Exit qualification
  - コントロールレジスタアクセスでexitした場合の詳細情報 (アクセスデータ、レジスタ)
  - I/O命令でexitした場合の詳細情報 (アクセスサイズ、R/W、ポート番号)
- Information for VM Exits Due to Vectoring Events
- Info. for VM Exits that Occur During Event Delivery
- Info. For VM Exits Due to Instruction Execution

3.7 VMCSのlaunch state

VMCSの使用状況を”launch state”と呼んでいる

- 最初にVMCSを使う前にVMCLEARを呼ぶ
  - -launch state = clear
- 使用する際にVM enter(VMLAUNCH)を呼ぶ
  - -launch state = launched
- 二回目(state=launched)のVMCSをLAUNCHする際にはVMRESUMEを使う

4.VMX命令セット

- VMPTRLD,VMPTRST
- VMCLEAR
- VMREAD,VMWRITE
- VMCALL
- VMLAUNCH,VMRESUME
- VMXON,VMXOFF

関連研究

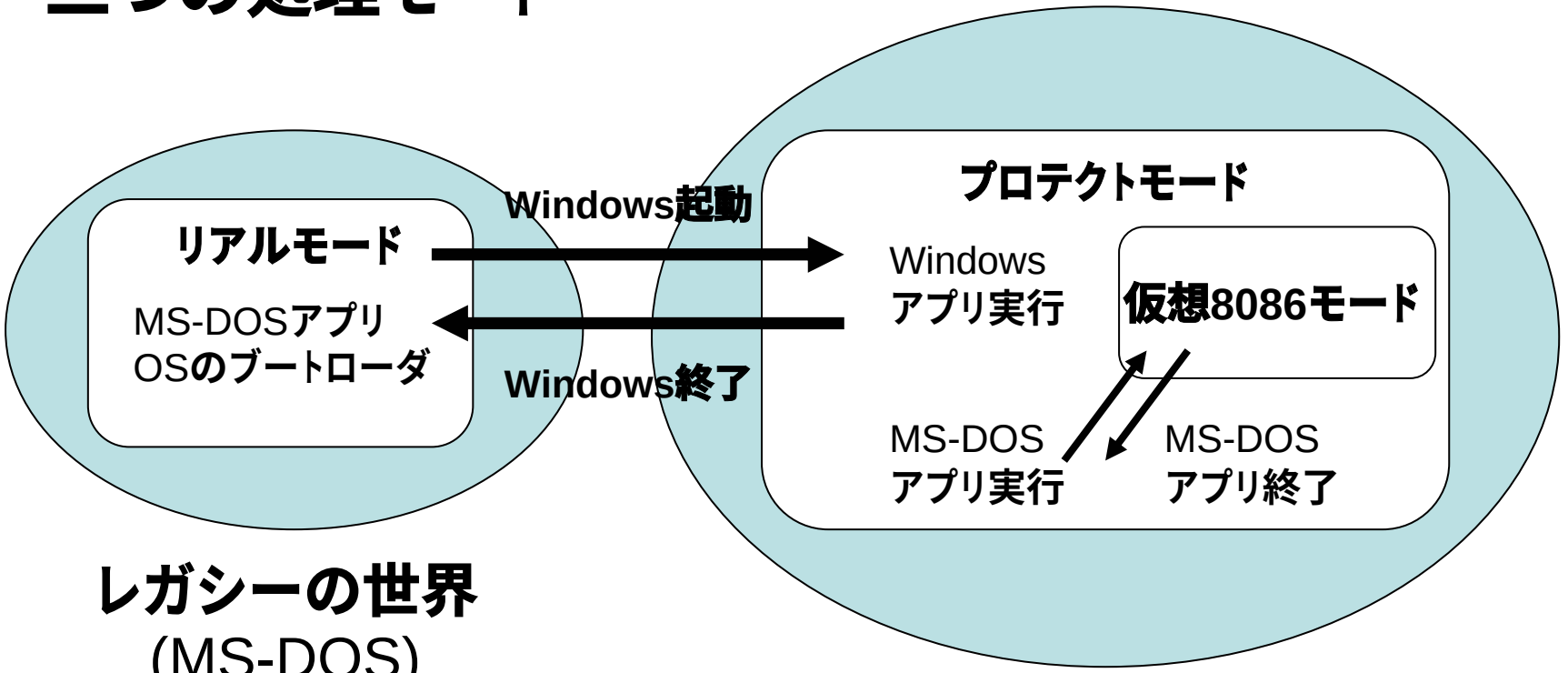
・Xen
  - Hypervisorモードで動作するVMM
  - Vanderpool対応、多くのベンダーから支持

・ＡＭＤ Pacifica
  - Intel同様VMMハードアシスト機能？

・IBM POWER4,POWER5,PowerPC970,CELL
  - メインフレーム譲りの仮想化機能
  - 仕組みはVanderpoolと同様
  - pSeriesサーバのファームウェアがVMMを実装
    (CPUの時分割利用、デバイスの多重化、etc..)

Intel x86アーキテクチャと仮想化における問題

x86アーキテクチャの概要
・三つの処理モード

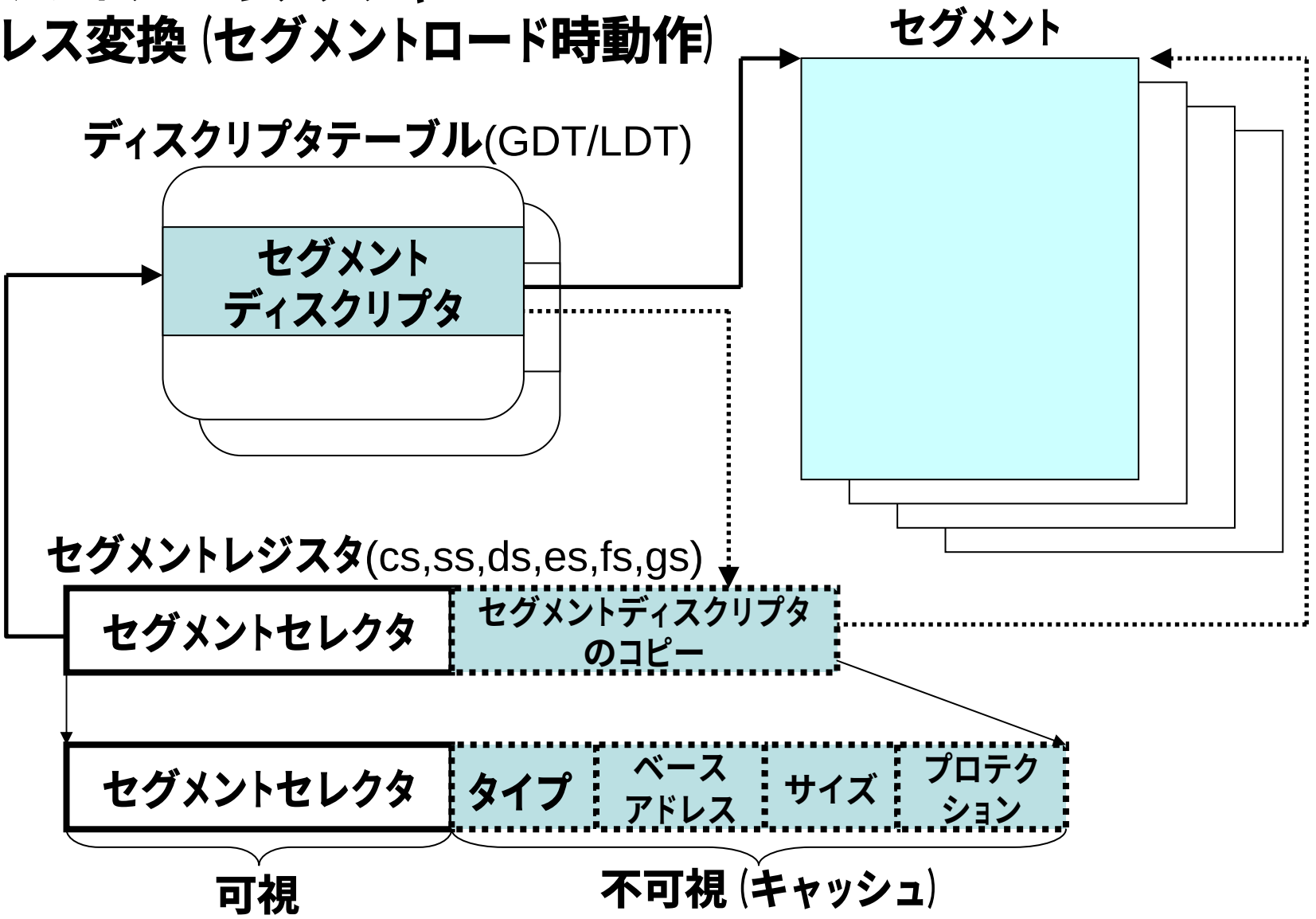


レガシーの世界
(MS-DOS)

現代の世界
(Windows, Linux)

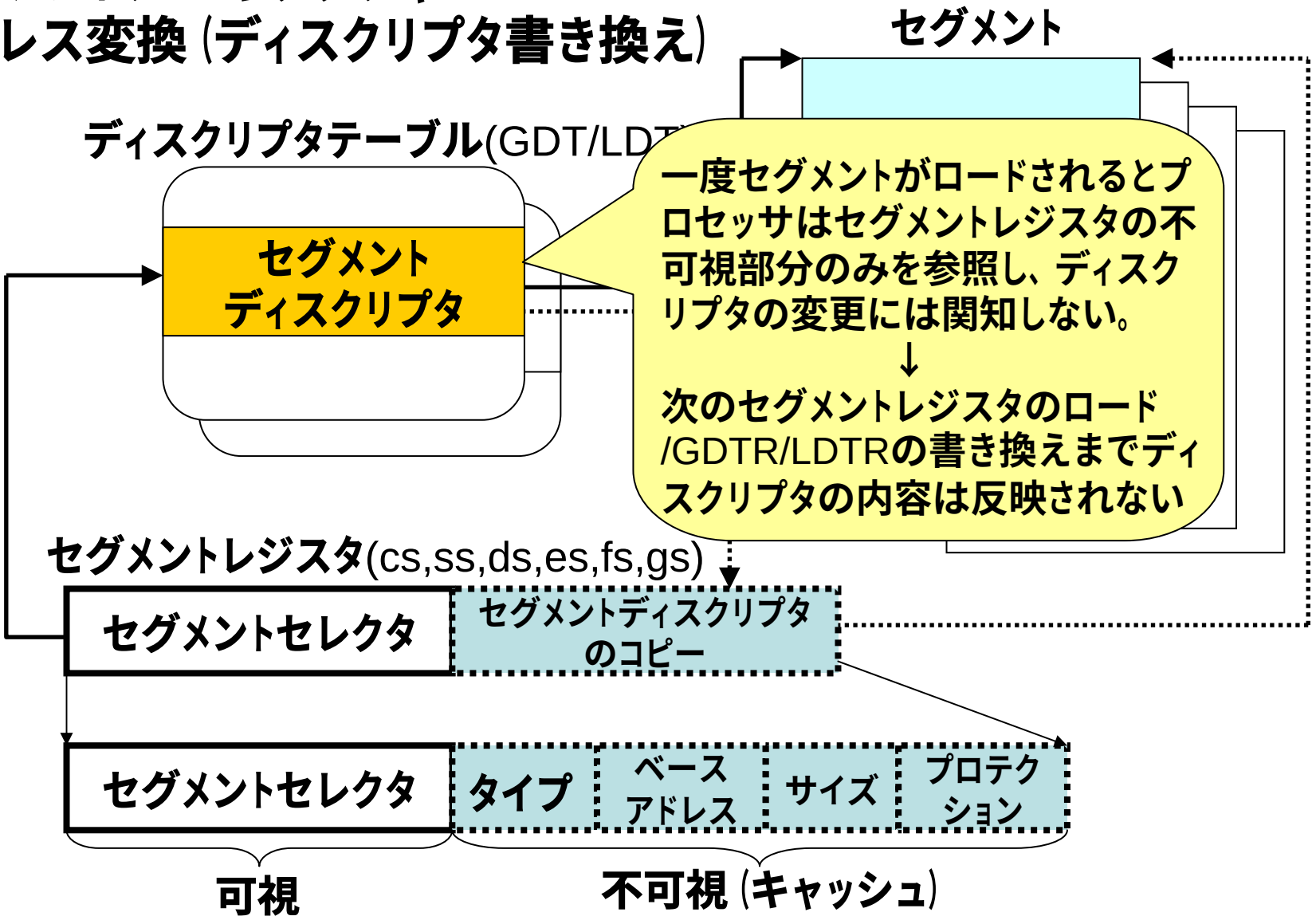
Intel x86アーキテクチャと仮想化における問題

セグメントアーキテクチャ
アドレス変換 (セグメントロード時動作)



Intel x86アーキテクチャと仮想化における問題

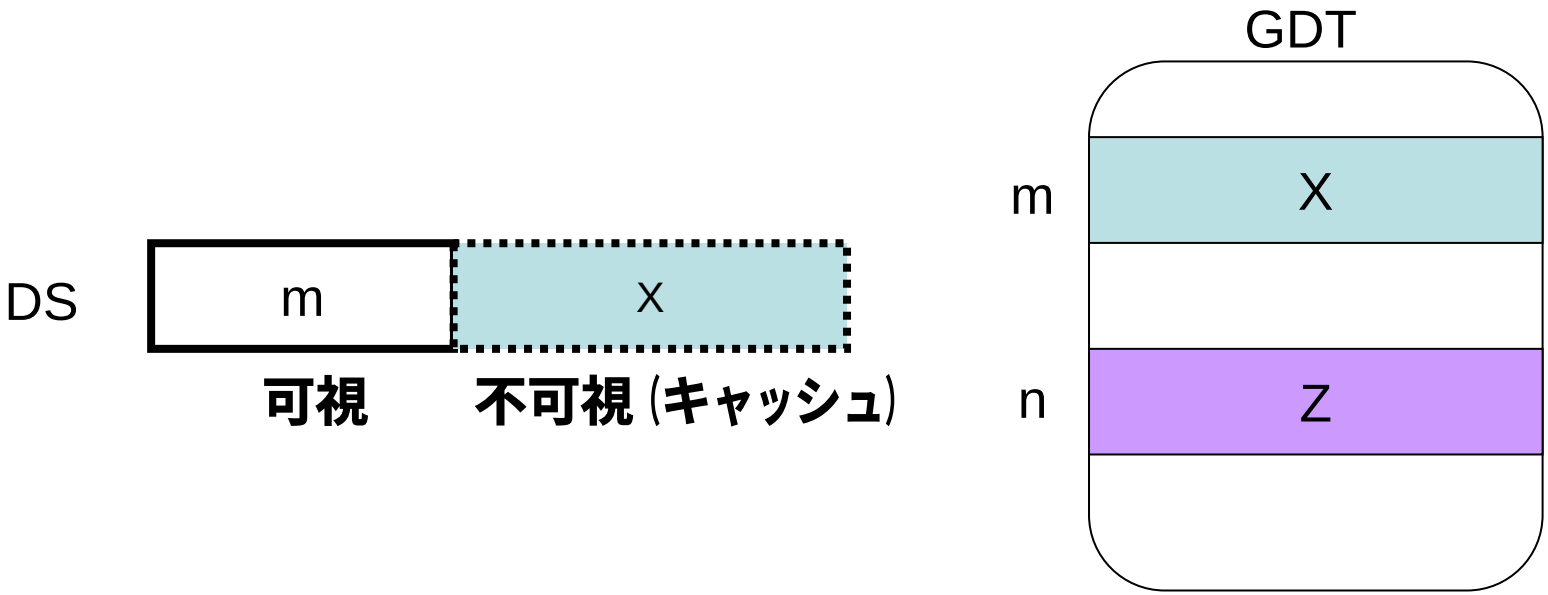
セグメントアーキテクチャ
アドレス変換 (ディスクリプタ書き換え)



Intel x86アーキテクチャと仮想化における問題

セグメントの不可逆性　ー何が問題なのか？

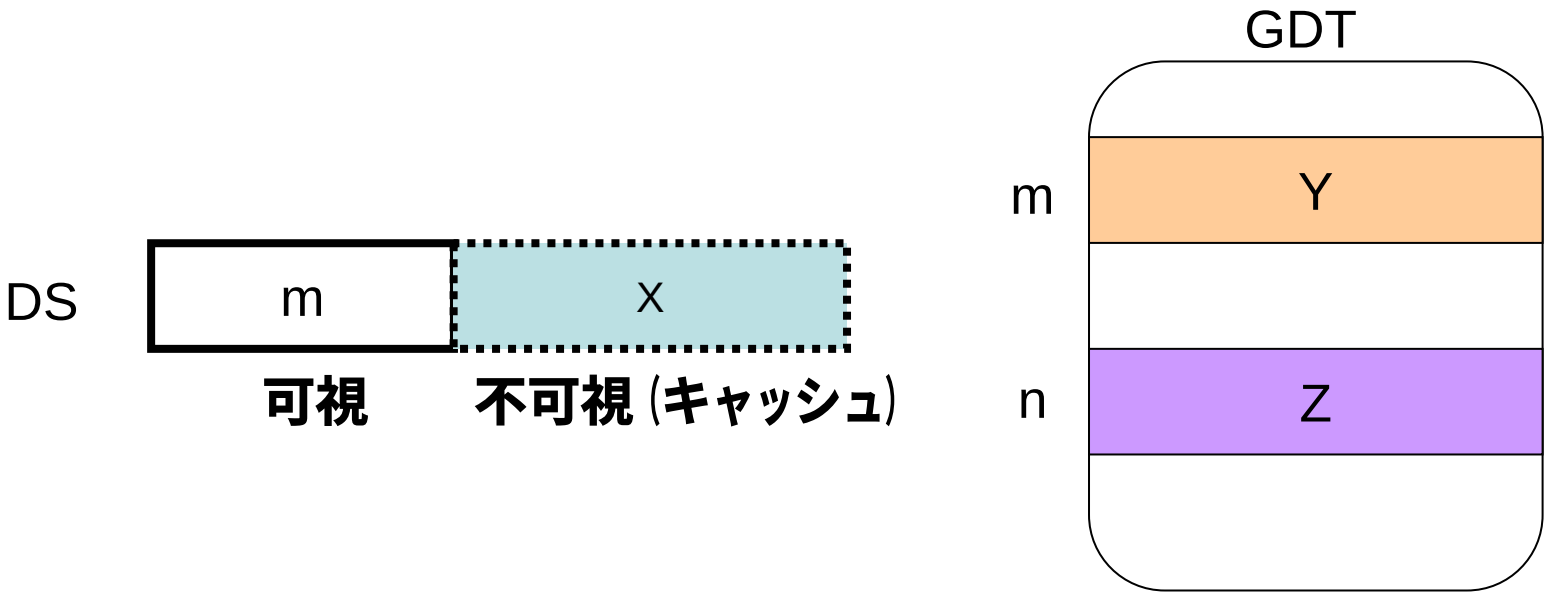
1. VMが”mov m→DS”を実行
   GDT上のオフセットmのセグメントディスクリプタの内容がセグメントレジスタDSの不可視部分にロードされる



Intel x86アーキテクチャと仮想化における問題

セグメントの不可逆性　ー何が問題なのか？

2. VMがGDTのオフセットXの内容をYに変える
   この変更はセグメントレジスタDSには反映されない



Intel x86アーキテクチャと仮想化における問題

セグメントの不可逆性　ー何が問題なのか？

3. VMMが一時的に(trapの処理や自身の処理で)DSを使う
   VMMが”mov n→DS”を実行
   GDT上のオフセットnのセグメントディスクリプタの内容がセグメントレジスタDSの不可視部分にロードされる
   ↓
   GDTにXの内容がないので
   DSレジスタ(X)を復元不能