

PCの仮想化支援機能と BitVisorコア技術詳細

2008年11月18日
セキュアVMワークショップ

筑波大学システム情報工学研究科研究員

榮樂 英樹

発表の流れ

1.PCについて

2.PCの仮想化 (一般的な話)

- DOS環境の仮想化、デバイスのエミュレーション
- プロセッサの仮想化支援機能

3.BitVisorコア

- 開発過程
- 難しかった点

PCについて

- IBM PC (1981)
- IBM PC/XT (1983)
- IBM PCjr (1983)
 - 日本語仕様:
IBM JX (1984)
- IBM PC/AT (1984)
- IBM PS/2 (1987)
 - 日本語仕様:
IBM PS/55 (1987)



IBM JX5 (1985)

1980年代当時のソフトウェア: DOS



画面ははめこみ合成
です

1980年代当時のソフトウェア: 表計算ソフト



画面ははめこみ合成
です

1980年代当時のソフトウェア: ゲーム



画面ははめこみ合成
です

PCの仮想化について

- 8086プロセッサ・DOS環境の仮想化
 - 仮想8086モード
 - WindowsやOS/2のDOSプロンプト、dosemuなど
- 386プロセッサ・WindowsやLinux環境の仮想化
 - 支援機能なし
 - VMware, Xen (paravirtualization) など
 - Intel VT/AMD-V
 - VMware, Xen (full virtualization), KVMなど

DOS環境の仮想化 (1990年代) の例: WindowsのDOSプロンプト



- ディスプレイをウインドウ内に表現
- キーボードはウインドウがアクティブな時だけ
- スピーカーも仮想化
- その他の**デバイスのエミュレーション**

```
コマンド プロンプト - STRIKER
(C)1985 by DEREK WILLIAMS          STRIKER BY DEREK WILLIAMS          game version: 1.2

This diskette contains an arcade game that runs on the IBM PC,XT,JR and
100% compatibles. FEEL FREE TO COPY THIS PROGRAM AND GIVE IT TO YOUR
FRIENDS. I also suggest if you enjoy this game that you send a donation
of $10 dollars to the address below. This will put you on a list of
registered users who will receive information on new releases of STRIKER

Type the files READ.ME and STRIKER.DOC for more information.
ENJOY STRIKER AND GIVE IT TO YOUR FRIENDS !!!
THIS PROGRAM MAY NOT BE DUPLICATED OR SOLD FOR COMMERCIAL PROFIT !

ADDRESS: DEREK WILLIAMS
10503 DOERING LN.
AUSTIN, TEXAS 78750

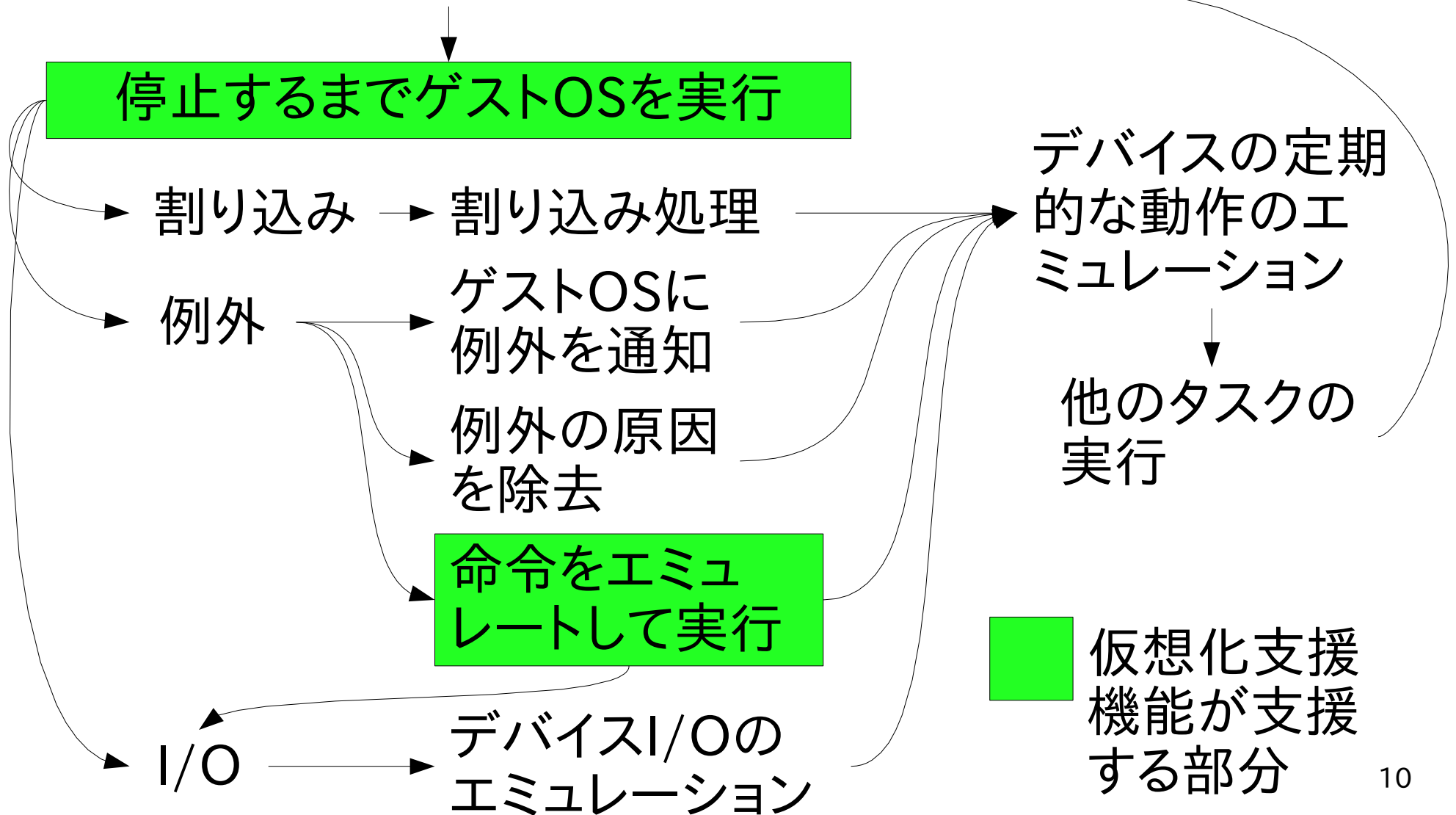
REQUIRES: PC,XT,Compatible with 192k, 1 DS Disk Drive, Color Graphics.
Jr with 256k, 1 DS Disk Drive, Color Graphics.
and DOS 2.00 or better on any system.
<< PRESS ANY KEY TO CONTINUE >>_
```

エミュレーションが必要な PC内部のデバイスの例 (参考)

- CPU・ROM・RAM
- 割り込みコントローラ
- プログラマブル・インターバル・タイマー
- カレンダー時計
- キーボードコントローラ
- ビデオコントローラ
- スピーカー
- 補助記憶装置 (HDD, FDD, DVDドライブなど)
- etc.

VMMの基本動作

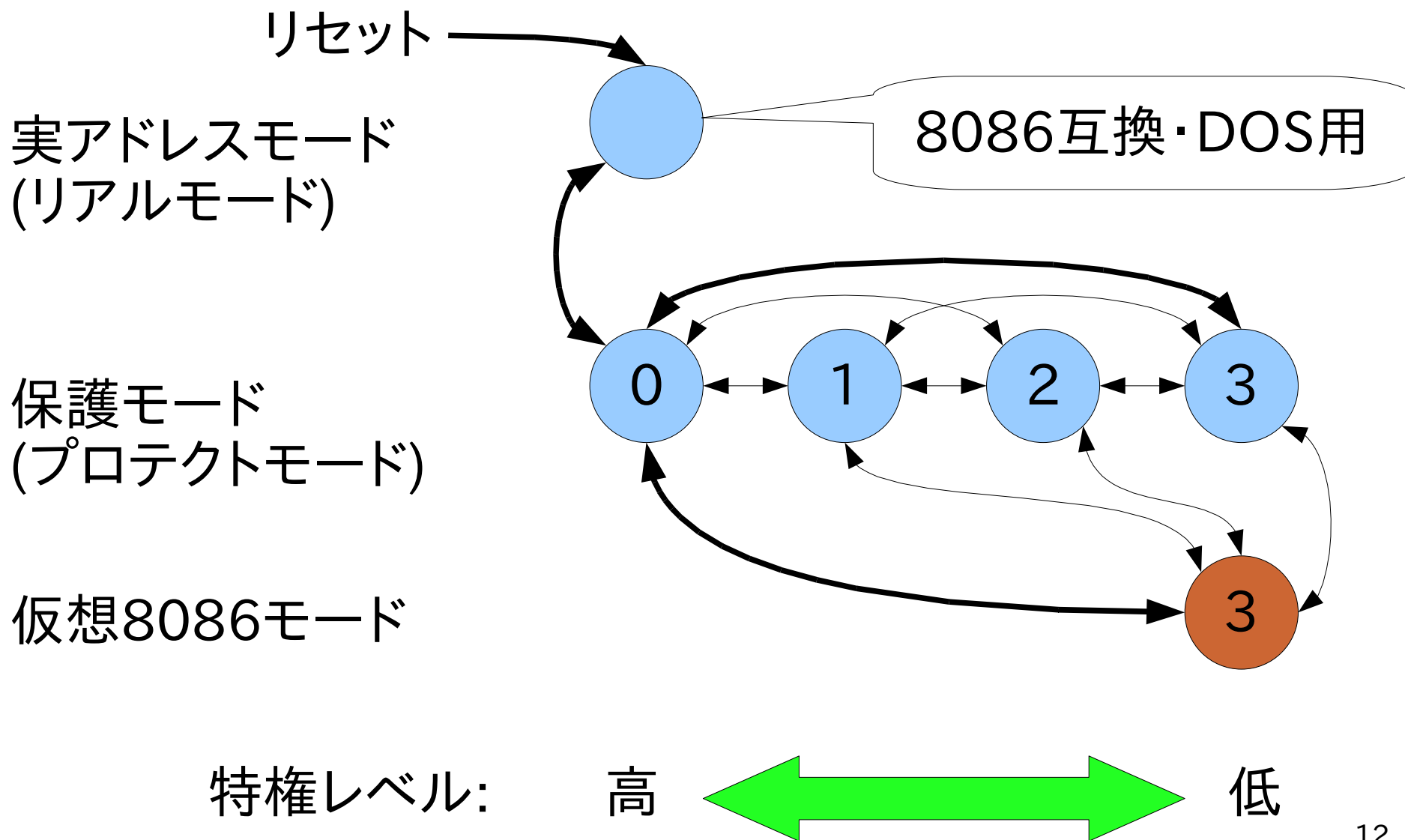
ゲストOSに対する割り込みの処理



プロセッサによる仮想化支援機能 について

- 仮想8086モード
 - 386プロセッサで初めて登場 (1986)
 - IA-32およびAMD64 (Intel 64) のLegacy Modeで利用可能
- Intel Virtualization Technology (Intel VT)
 - 2005年頃に初めて登場 (Pentium 4の一部)
- AMD Virtualization (AMD-V)
 - 2006年頃に初めて登場 (Athlon 64の一部)

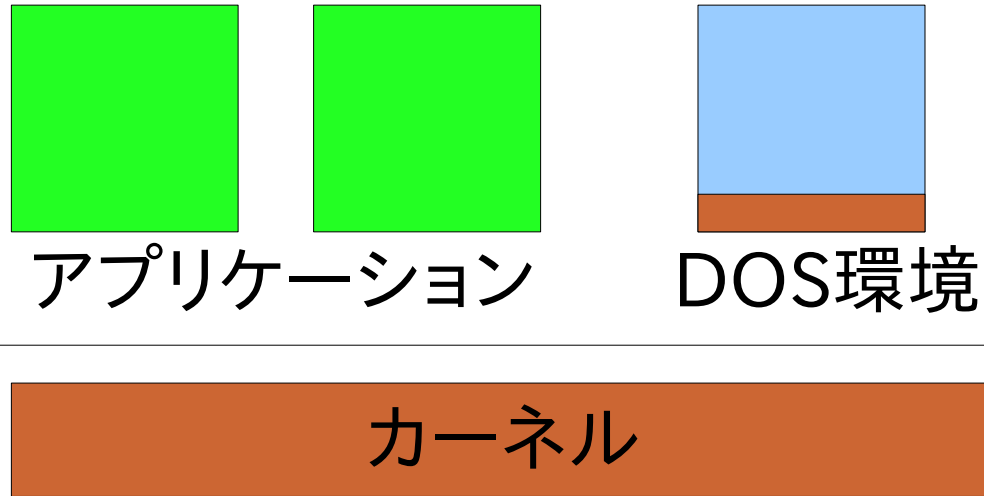
仮想8086モードへの状態遷移



仮想8086モードの特徴

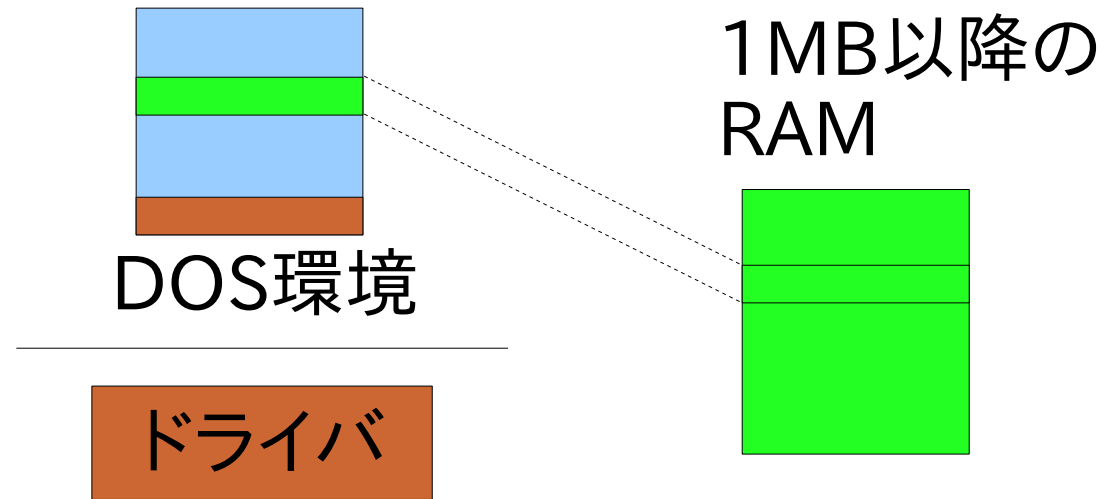
- 保護モードと違うところ
 - 常に特権レベル3で、セグメント機能が8086互換になる
 - IOPL (I/O特権レベル) を3未満にすると、割り込み許可フラグを含むフラグレジスタの読み書きを禁止できる
 - IOPLに関係無く常にI/O許可ビットマップが使われる
- 仮想メモリ (ページング) や割り込みは保護モードとほぼ同じ
 - I/O命令はI/O許可ビットマップにより実行を禁止することができる
 - 禁止されたI/O命令や特権命令を実行しようとする、例外が発生する

Windowsにおける 仮想8086モードの利用



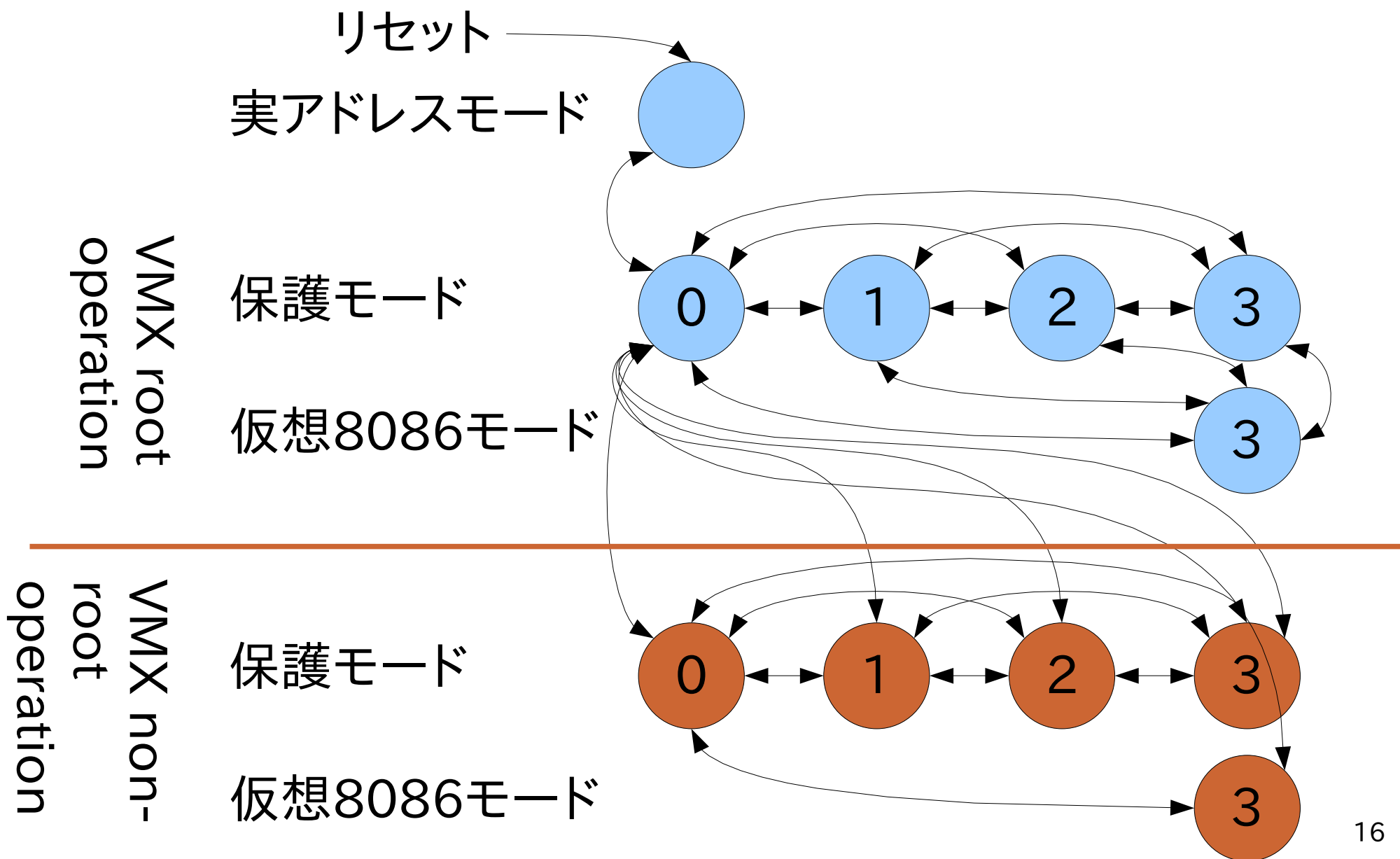
- I/O命令はI/O許可ビットマップで禁止され、OSによってエミュレートされる
- 特権命令は一部を除きエミュレートされない
- 特権命令を実行して保護モードに移行するDOSアプリケーションは実行できない

DOSにおける 仮想8086モードの利用

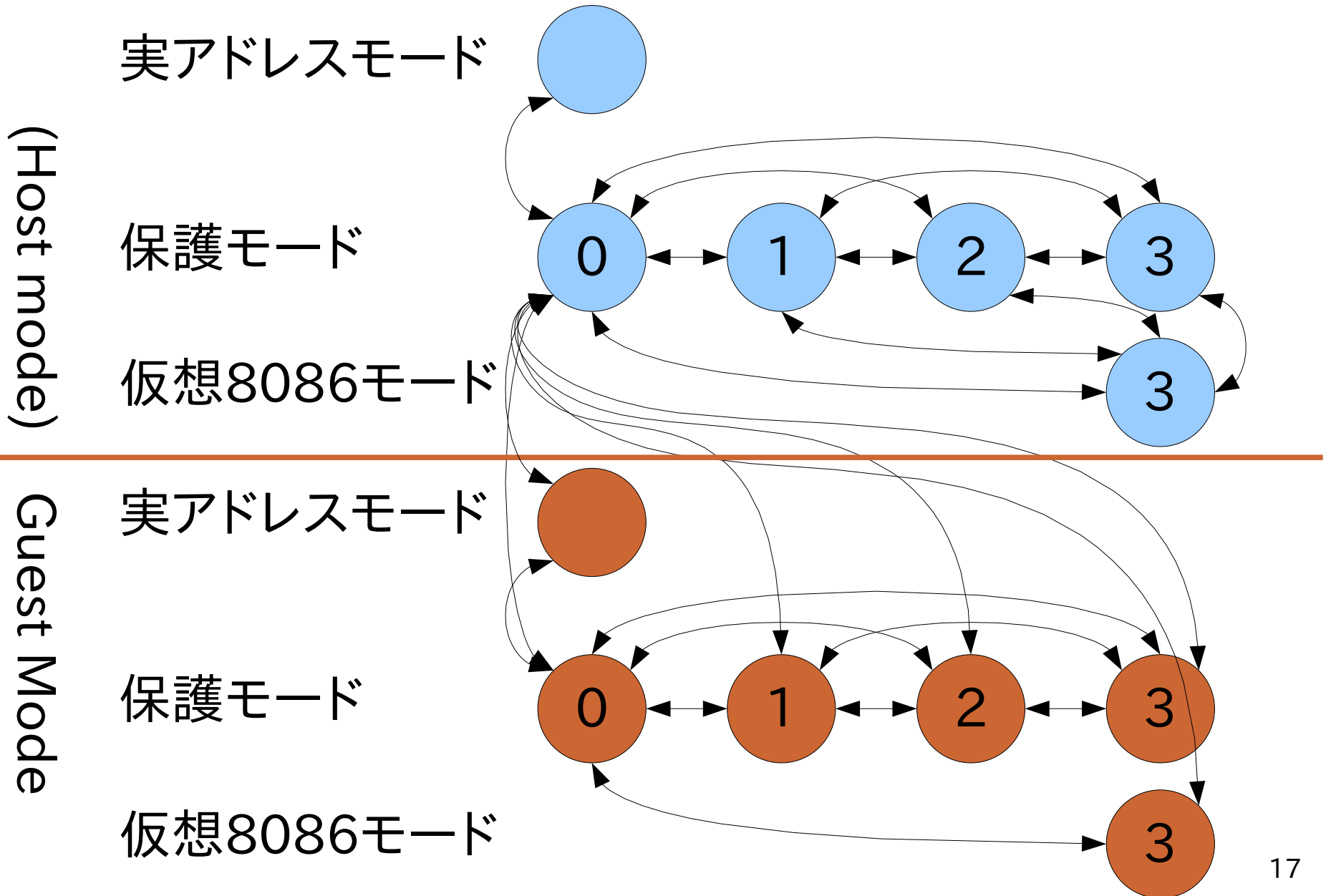


- 仮想8086モードを利用した、ソフトウェアEMS (バンク切り替えによるメモリ拡張をソフトウェアで実現したもの) が普及していた
 - 割り込みはpass through。バンク切り替えの部分とUMBを除いてアドレス変換無し
 - 基本的にI/Oエミュレーションはしないが、DMA転送はエミュレーションをしていたらしい

Intel VTの状態遷移



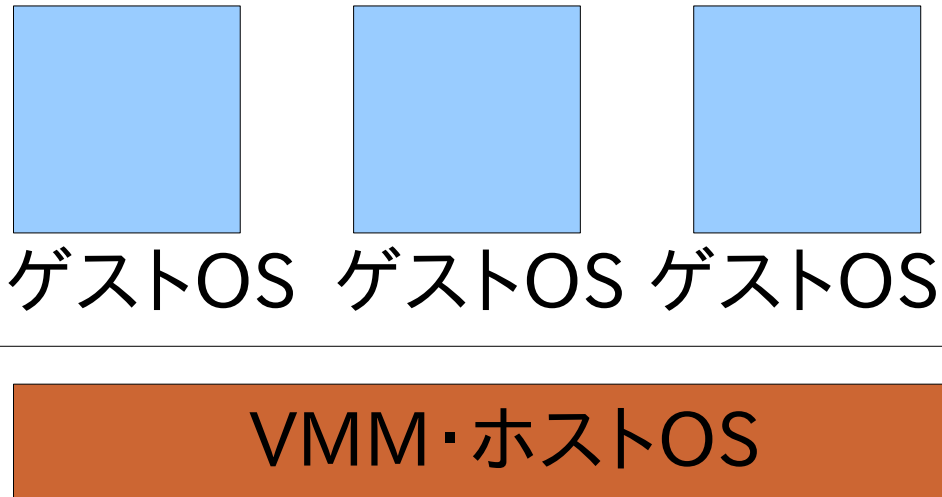
AMD-Vの状態遷移



Intel VT/AMD-Vの特徴

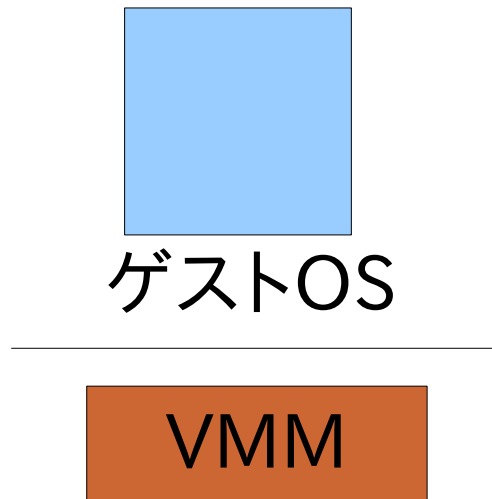
- 制御レジスタ、セグメントレジスタ、GDTR, IDTRやTRなどのレジスタの切り替えをしてくれる
 - セグメントレジスタのシャドー部分を含む
 - 例えばゲストOSのGDTRをそのまま使えば、**SGDT**命令 (非特権命令) はそのままが良い
- 特権レベル0でも、特定の命令の実行を禁止することができる
 - I/O命令や、制御レジスタの書き込みを禁止することで、VMMが管理できる
 - 特権レベル0が使えるので、セグメントレジスタの読み出し (非特権命令) もそのままが良い

KVMにおける Intel VT/AMD-Vの利用



- I/O命令は禁止され、VMM (ホストOS上で動くプログラム) によってエミュレートされる
- 特権命令は、命令によって異なる
 - 例: 制御レジスタのアクセスはエミュレートされるが、**LGDT**命令はそのまま実行される

BitVisorにおける Intel VT/AMD-Vの利用



- DOSにおける仮想8086モードの利用方法に似ている
 - 割り込みはpass through。VMM部分を除いてアドレス変換無し
- VMMがI/Oの監視、暗号化等の追加処理を行う

例: I/O命令の処理

OUT 60H, AL ; I/Oポート60HにALを出力
の実行時に停止して、以下の情報を渡してくれる

Intel VT

Exit reason
30 (I/O instruction)

Exit qualification
00600040H

VM-exit instruction
length
命令のバイト数

AMD-V

EXITCODE
7BH (VMEXIT_IOIO)

EXITINFO1
00600090H

EXITINFO2
次の命令のプログラム
カウンタ

何が難しいか？

- 一般話
 - 実アドレスモード
 - 機械語の解釈
 - ページング (仮想メモリ)
- BitVisorの話

実アドレスモード

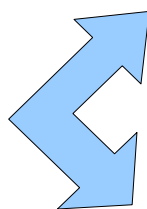
- Intel VTではゲストを実アドレスモードに設定できない
 - 仮想8086モードを使って再現する
 - 特権命令は例外をとらえて機械語を解釈しエミュレートする (一部命令はとらえることができない)
 - 実アドレスモードと仮想8086モードの違いが問題
- AMD-Vには“Paged Real Mode”がある
 - Guest Modeで設定できる
 - ページングを有効にしたまま実アドレスモードとして動作する

実アドレスモードと仮想8086モードの違い

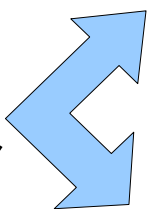
セグメントレジスタ	CS, DS, ES, FS, GS, SS	セクタ値 16ビット	ATTR	LIMIT 32ビット	BASE 32ビット
-----------	------------------------	---------------	------	----------------	---------------

レジスタ名 命令で読み書きする値 ソフトウェアから見えない部分

切替前の
値を保持



すべての
セグメント
をロード



	セグメントのロード	Intel VT での制約
実アドレスモード	BASE ← sel × 10H LIMIT は不変	禁止
保護モード	GDT/LDT からロード	特権レベル等が 条件を満たすこと
仮想 8086 モード	BASE ← sel × 10H LIMIT は 0FFFFH 固定	BASE = sel × 10H LIMIT = 0FFFFH ATTR = 0F3H

機械語の解釈

なぜ機械語の解釈が必要か

- Intel VT: 実アドレスモードのかわりに仮想8086モードを使うため。特権命令等のエミュレーションに必要
- AMD-V: EXITCODE, EXITINFO1およびEXITINFO2に返される情報では不足するため。以下に例を示す

MOV CR4, EDI ; EDIを制御レジスタCR4に転送

Intel VT

Exit reason 28
制御レジスタアクセス

Exit qualification
00000704H

AMD-V

EXITCODE 14H
VMEXIT_CR4_WRITE

これだけ!

ページング

- ページングに関してIntel VT/AMD-Vがやってくれること
 - ゲストOSとVMMの間で制御レジスタを切り替える
 - 制御レジスタへのアクセスを止める (interceptする)
- 仮想化の必要性

仮想番地	物理番地
0000	1000
0001	1001
0002	2000

ゲストOSの
ページテーブル

ゲストの 物理番地	実際の 物理番地
1000	E000
1001	E001
2000	F000

VMMのメモリ管理

仮想番地	物理番地
0000	E000
0001	E001
0002	F000

実際にCPUが見る
べきページテーブル

VMMが作るシャドウページテーブル

ページングのプロセッサによる支援

- Intel VT: Extended Page Tables (EPT)
2008年登場
- AMD-V: Nested Page Tables (NPT)
2007年登場
- TLBミス時のMMUの動作

1. ゲストOSのページ
テーブルを見る
2. Nested ページ
テーブルを見る
3. TLBエントリを追加する

仮想番地	物理番地
0000	1000
0001	1001
0002	2000

ゲストOSの
ページテーブル

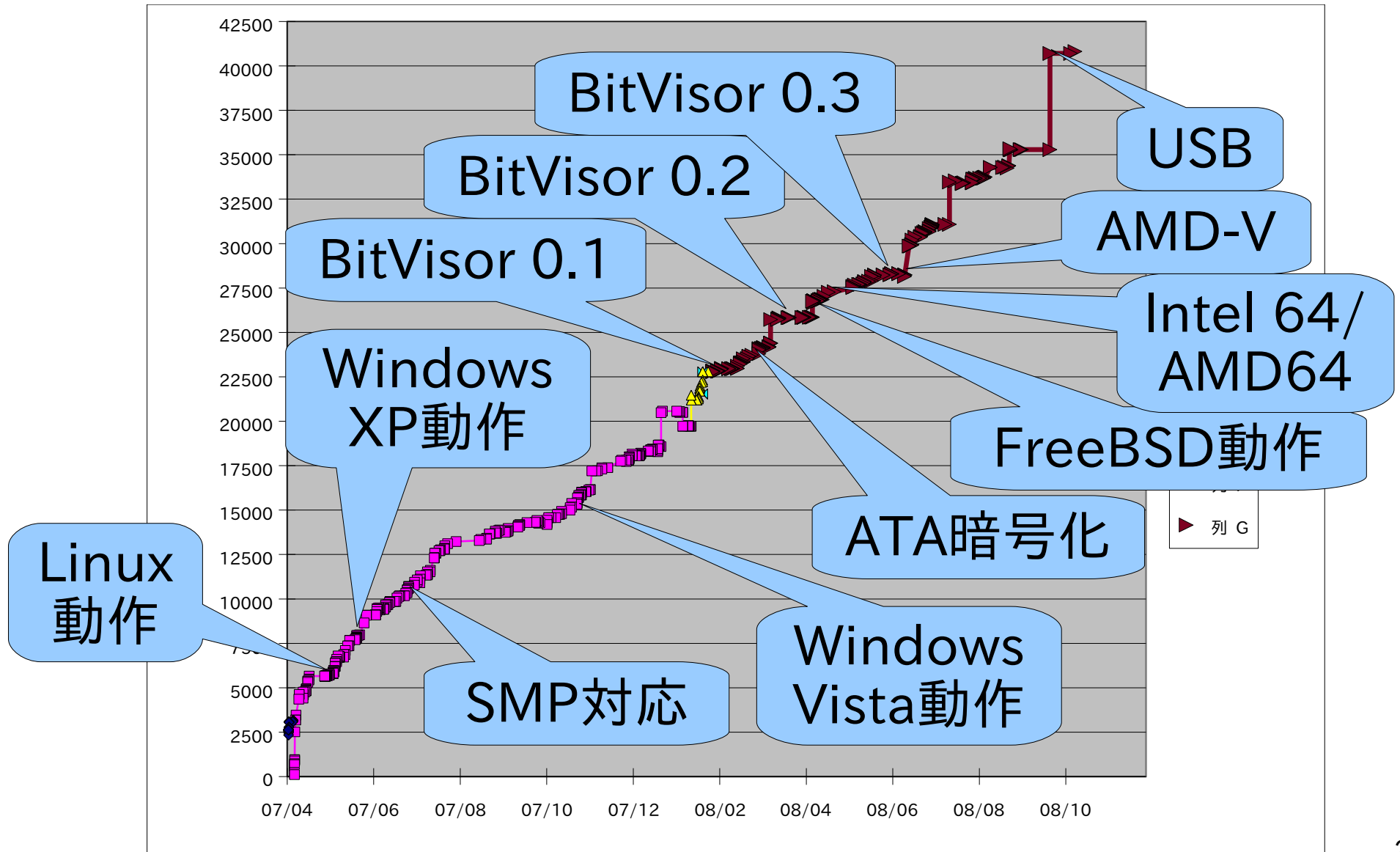
ゲストの 物理番地	実際の 物理番地
1000	E000
1001	E001
2000	F000

Nested
ページテーブル

BitVisor開発の過程

0. Intel VTの調査。FreeBSDで動くミニVMM開発
1. Linux用のローダブルカーネルモジュール。DOSのソフトウェアEMSの、Linux+Intel VT版のようなもの
2. GRUBからの起動部分開発、実アドレスモード対応
3. FreeDOSゲストでデバッグ
4. Linuxゲストでデバッグ
5. Windows XPゲストでデバッグ
6. マルチプロセッサ・マルチコア対応
7. メモリ管理機能、デバッグ機能、性能改善、整理
8. PAE対応、64bit対応、Windows Vista対応、etc.

BitVisor コード行数の変遷



BitVisorで何が難しかったか？

- System Management Mode (SMM)
- シャドウページテーブルのデバッグ
- タイマー
- INT 15Hのフック
- マルチプロセッサ (マルチコア) 対応
- Memory Mapped I/O (MMIO) のフック
- 電源管理 (ACPI)
- Intel 64/AMD64対応
- Intel VTとAMD-Vの違い

System Management Mode (SMM)

- 電源管理等を目的として386SLから導入された動作モード。System Management Interrupt (SMI) によって起動される
- SMIハンドラは通常は見えないところにあるファームウェア (BIOS) に入っていて、独立した動作環境で実行される (用途はマシン依存)

OUT 0B2H, AL ; I/Oポート0B2HにALを出力

- このI/O命令を実行するとSMIが発行される
- この命令をVMMが代行するとおかしくなるので、フックしないようにする

シャドウページテーブルのデバッグ

- もっとも基本的なアルゴリズム: シャドウページテーブルを仮想TLBとみる (CPU_MMU_SPT_1)
 - 性能は悪いが、コード量が少ないためデバッグも簡単
- 性能重視のアルゴリズム (CPU_MMU_SPT_3)
 - 仮想TLBフラッシュの際にシャドウページテーブルをなるべくクリアせずに保持する
 - ゲストのページテーブルを読み取り専用にすることで、ページテーブルの書き換えを検出してなんとかする
 - 複雑でデバッグが難しい
 - たまに固まる、たまに落ちる、たまに一部のプロセスが無限ループに陥るがOSは動いている、などの症状

タイマー

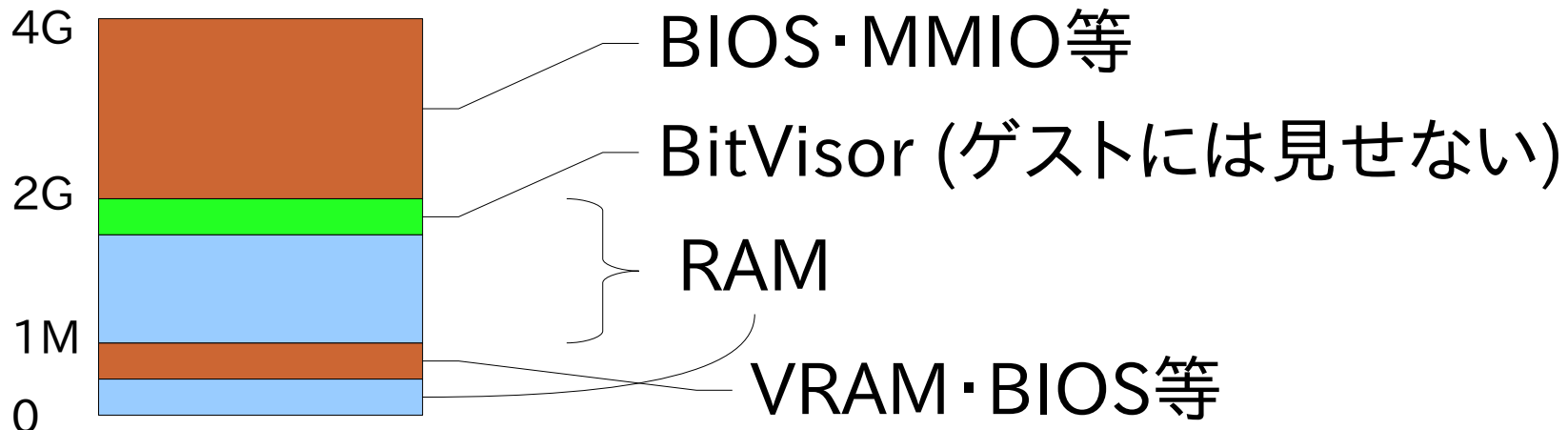
- VPNやUSBのドライバでタイマーが欲しいという
- タイマー一覽
 - 初代IBM PCからある8253PIT: 16bit, 1.1932MHz
 - ACPIのPower Management Timer (PM_TMR): 24bitまたは32bit, 3.5795MHz
 - プロセッサのTime Stamp Counter (TSC): 64bit, プロセッサのクロック
 - Local APICのタイマー: 32bit, APICバスのクロック
- ゲストに依存せずに使えるもの
 - TSCとPM_TMR。但しTSCは一部ノートPCで変動する

タイマー余談

- 1.1932MHz, 3.5795MHz, 4.77MHz, 7.2MHz...
の元は?
 - IBM PCの基本周波数**14.31818MHz**
 - $14.31818/2 = 7.15909$ (IBM JX5 CPUクロック)
 - $14.31818/3 = 4.772726$ (IBM PC CPUクロック)
 - $14.31818/4 = 3.579545$ (カラーTVのバースト信号)
 - $14.31818/12 = 1.1931816$ (IBM PC PITクロック)
- DOSのタイマー割り込み**18.2Hz**の正体は?
 - PITのカウンタを65535にしたときの割り込み頻度
 $1.1932\text{MHz}/65535 = 18.206785\text{Hz}$

INT 15Hのフック

- INT 15H: 古くはカセット制御、現在は電源管理やメモリマップの取得等に使われるBIOS呼び出し
- BitVisorではメモリマップの取得に使用するとともに、メモリ容量の縮小のためにフックする
 - フックハンドラをどこに入れるか?
 - 仮想8086モードで呼び出されることもあるので注意



マルチプロセッサ対応 (1/2)

- 起動時: 1つのプロセッサだけが動作、他プロセッサは停止
- BitVisorが他プロセッサを開始
 - Local APICを叩いて、“interprocessor interrupt” (IPI) を送信する
 - APICの初期化等は不要。自分以外のプロセッサに対して、INITと、Start-up IPIを送信すれば良い
- AMDのマシンで他プロセッサが開始しないことが多いという問題が未解決

マルチプロセッサ対応 (2/2)

- Intel VT

- プロセッサを停止状態に設定できる

- Activity State: Active, HLT, Shutdown or Wait-for-SIPI

- INITやStart-up IPIを処理できる

- Exit reason: 3 (INIT signal)

- Exit reason: 4 (Start-up IPI)

- Exit qualification: SIPI vector information

- BitVisorではこれを利用している

- AMD-V

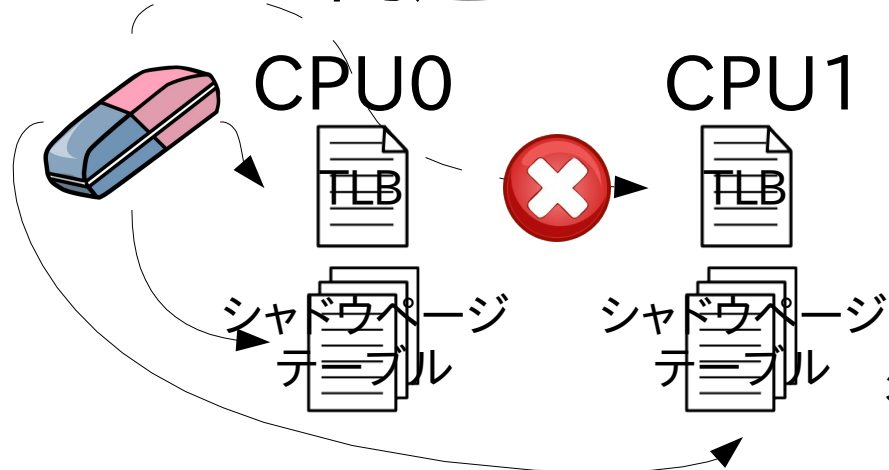
- 停止状態にできない・Start-up IPIを処理できない

- Local APICへのアクセスをフックするしかない?

MMIOのフック

- シャドウページテーブルで、フックするMMIOの範囲のページを読み書き禁止に (存在ビットをクリア) しておき、ページフォールトによりアクセスを検出する
- 検出後は、機械語の解釈を行い、アクセス内容を特定する
- マルチプロセッサ環境で、他プロセッサのTLBに入っている場合にどうするかという問題

- 他プロセッサのTLBを直接フラッシュすることはできない



電源管理 (ACPI)

- スリープの検出
 - Fixed ACPI Description Table (FADT) に書かれているI/Oポートアドレスへのアクセスをフックして検出する
 - Differentiated System Description Table (DSDT) に、書き込まれる値についての情報がある
- Windowsゲストでシリアルポートが止まる問題
- DSDTの解釈が難しい
 - ACPI Machine Language (AML) の文法は仕様書に書かれているが、何か怪しい
 - iaslコマンドでコンパイル・逆コンパイルが可能だが、やっぱり怪しい

AMLの困ったところの例: 引数の終わりが識別できない

iaslコマンドによる逆コンパイル結果の一部:

```
If (\_SB.PCI0.PEX0.XPM1 (0x00)) {  
    Notify (\_SB.PCI0.PEX0, 0x02)  
}
```

2行目をちょっと編集してコンパイル:

```
Notify (\_SB.PCI0.PEX0.XPM1 (0x01), 0x02)
```

そして逆コンパイル:

```
Notify (\_SB.PCI0.PEX0.XPM1, One)  
0x02
```

Intel 64/AMD64対応

- 1.VMMでのPAE有効化
- 2.VMMの64ビット対応
- 3.シャドウページテーブルのPAE有効化
- 4.ゲストのPAEに対応
- 5.ゲストの64ビットに対応

ポイント

- ページテーブルの読み書きを抽象化して、VMM、ゲスト、およびシャドウページテーブルを共通の方法で読み書きできるようにすることで、実装をシンプルにした
- 機械語解釈ルーチンも対応が必要

Intel VT/AMD-Vの違い

- マニュアルについて
 - Intelのマニュアルは説明が多すぎてうんざりする
 - AMDのマニュアルはところどころ説明が欠けている
 - host state-save areaに何バイト用意すれば良いのか不明
- 感想
 - Intel VTは強制される設定やチェック項目が多く、実アドレスモードの扱いにも困るが、CR0・CR4のシャドウの設定ができたたり、細かい情報を渡してくれるため機械語解釈処理の負担は小さい
 - AMD-Vのほうがシンプルでわかりやすいが、機械語解釈処理の負担は大きい

まとめ

- PCの仮想化支援機能
 - 仮想8086モード
 - Intel VT/AMD-V
 - 概要
 - 難しい点: 実アドレスモード、機械語の解釈、ページング
- BitVisorコア技術詳細
 - 開発過程
 - 難しかったところ: SMM、デバッグ、タイマー、INT 15Hのフック、マルチプロセッサ対応、MMIOのフック、ACPI、Intel 64/AMD64対応、Intel VT/AMD-Vの違い