Xen/IA64の実装と最適化手法

山幡 為佐久

2007.07.20

ヴィーエー・リナックス・システムズ・ジャパン株式会社
VA Linux Systems Japan K.K.

valinux.co.jp

アジェンダ

- IA64
- Xen/IA64
- **今後の開発**
- **質疑応答**

IA64

Itanium(IA64)とは

- IntelとHPが共同開発したアーキテクチャ
  - IA64は開発コード
- エンタープライズ用途やHPC向け
- 2001年に最初の製品が出荷された
  - 現在はItanium2(Montecito)が出荷されている

Itanium(IA64)とは(その2)

- EPIC(Explicitly Parallel Instruction Computing)を採用
  - x86とは全く違う命令体型
- 柔軟なメモリ管理機能
  - x86で採用されているようなページテーブルは無く、tlb insert/purgeを使用
  - ページ保護機構も柔軟
    - セグメント等複雑怪奇なものは無い

IA64と仮想化

- x86と比較して仮想化しやすい
- 仮想化出来ない命令は存在
- その他
  - 特権モードPL0-PL3(x86と同様)
  - アドレス変換モードが複雑
  - RSE(レジスタスタックエンジン)
- ファームウェア (EFI/SAL/PAL)も仮想化が必要(x86のBIOSに相当)

EFI:Extensible Firmware Interface
SAL:System Abstraction Layer
PAL:Processor Abstraction Layer

仮想化出来ない命令

- trap-and-emulate出来ない
  - thash, ttag メモリ管理
  - cover レジスタスタック操作
  - fc メモリアクセス
  - mov cpuid
  - mov pmd pmdレジスタからの読み出し
  - mov ar.rsc 特権レベルの読み出し

VT-i

- IA64向けVT
  - 命令セットが修正され仮想化可能に
  - psr.vmビット追加
  - vmsw命令追加
  - Virtualization fault追加
  - PALにインターフェース追加

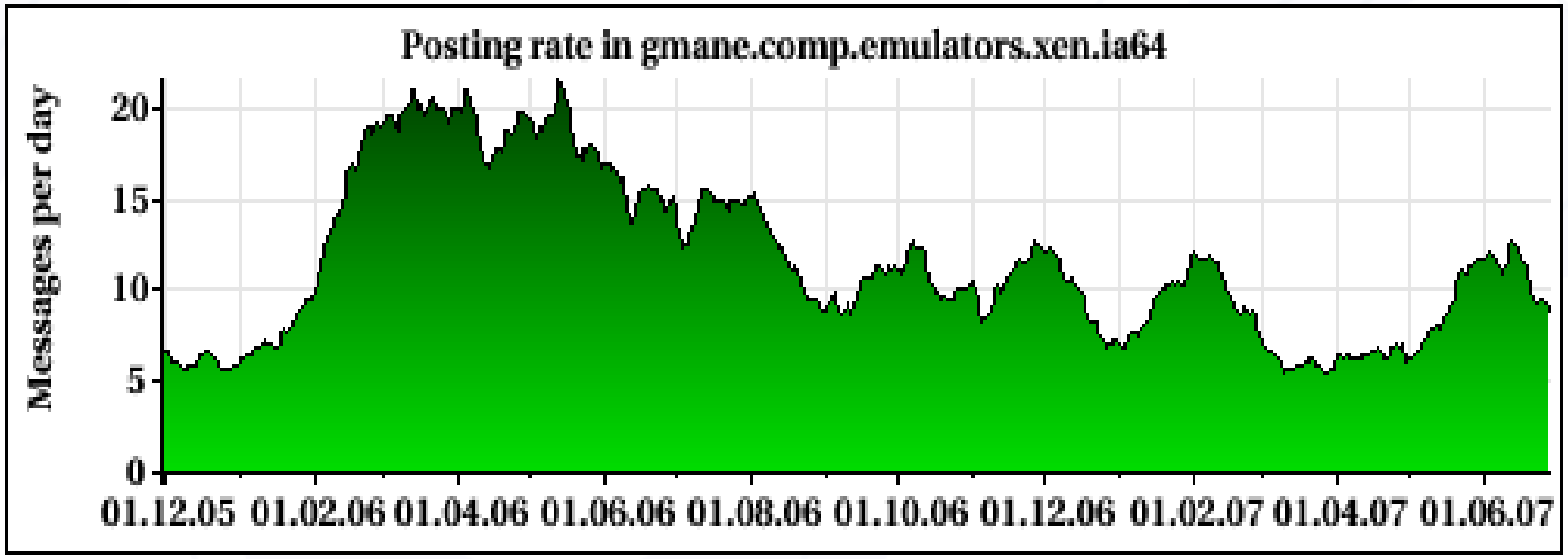
Xen/IA64

Xenアーキテクチャサポート

- Xenは複数のアーキテクチャをサポート
  - x86
    - x86_32,x86_32pae,x86_64
  - IA64
    - **X86以外で移植された最初のアーキテクチャ**
  - PPC
  - ARM

Xen/IA64の開発体制

- xen-ia64-develで活発に開発
  - 200人強が購読
  - うち日本人60人強と思われる



www.gmane.org

Xen/IA64の特徴(機能編)

- 機能的にx86と同等
  - 仮想デバイス
  - ドメインセーブ/レストア
  - ライブマイグレーション
  - RHEL5サポート
- 準仮想化/完全仮想化ドメインをサポート
  - 完全仮想化ドメインはVT-iを使用
    - ゲスト用ファームウェア(GFW)が必要
  - qemuを使用したデバイスエミュレーション
  - Windows/IA64も動作
  - PV-on-HVM

Xen/IA64の特徴(実装編)

- Linuxのコードを活用
  - そのままあるいは若干の修正で使用
  - Cまたはアセンブリ(.c/.S ia64配下137ファイル)
    - 修正なし 25ファイル
    - 修正あり 38ファイル
  - ヘッダファイル(.h asm-ia64配下271ファイル)
    - 修正なし 94ファイル
    - 修正あり 48ファイル
  - ダミー空ヘッダファイル 55ファイル
    - コンパイルを通す為に必要

Xen/IA64の特徴(実装編2)

- Transparent paravirtualization
  - 同一バイナリがXenと実環境の両方で動作(すべき)
- Optimized paravirtualization
  - 正しく動作させる為に準仮想化を行うだけでなく、性能向上の為にも準仮想化を行う
    - 仮想化できない命令だけでなく、性能上重要な特権命令も仮想化を行う

x86との相違

- アーキテクチャの違いに起因する仮想化の違い
  - レジスタ仮想化
  - メモリ管理
  - 割り込み
- Linuxへの変更(準仮想化)の違い
- ファームウェア(EFI/SAL/PAL)の仮想化

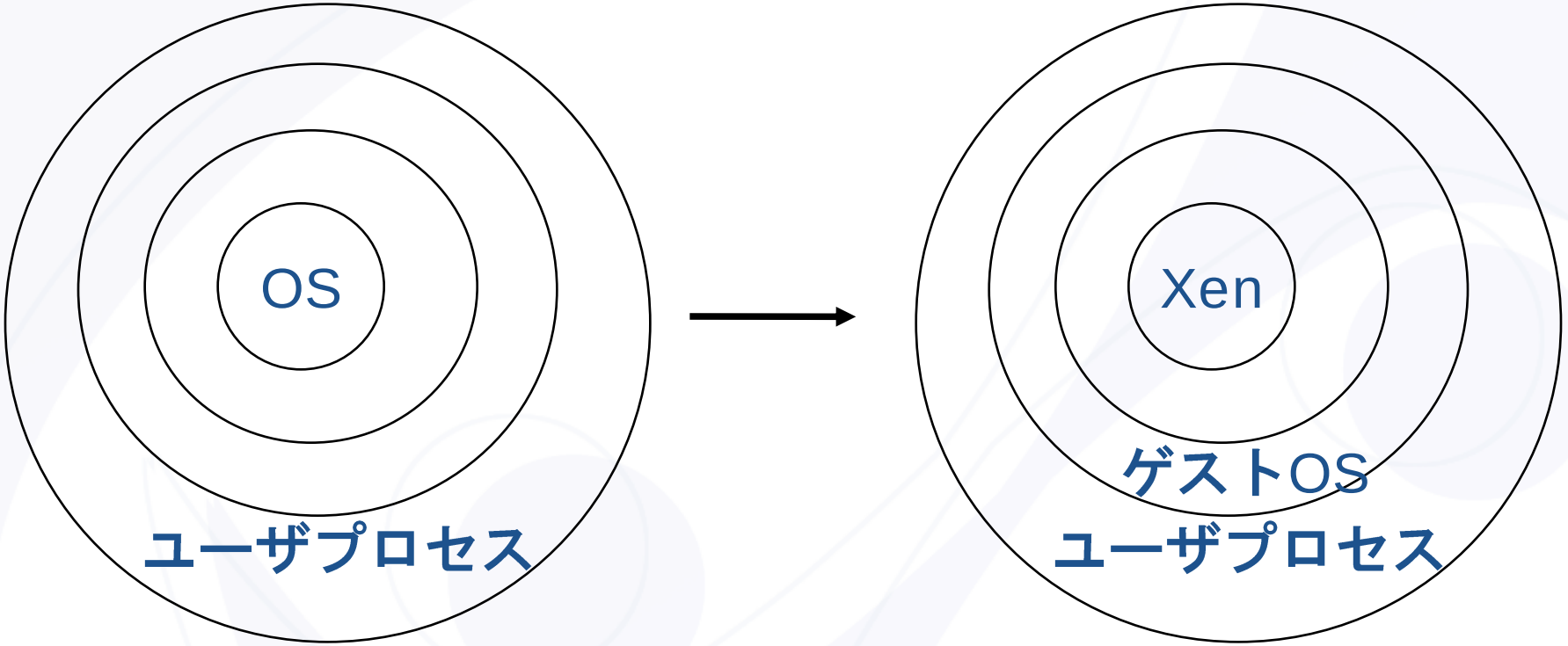
x86と共有

- ハイパーコール
  - 共通のハイパーコール
  - イベントチャネル
  - grant table
- 仮想I/Oデバイス
  - 仮想デバイス等は出来るだけそのまま使えるように
- 管理ツール
  - 共通の管理ツール(xmコマンド、Xen API)

Xen/IA64準仮想化手法

- ゲストOSを低い特権モードで動作
- Transparent Paravirtualization
- Memory Mapped Registers (a.k.a. HyperRegisters)
- HyperPrivops
- MMU完全仮想化

特権モード



ゲストOSを低い特権レベルで動作させる(x86と同様)
特権レベル=1を使っていないのは歴史的事情による

Transparent Paravirtualization

- 同一バイナリがXen環境及び実環境の両方で動作すべき
- x86ではXen専用カーネルになる
  - paravirt_opsで複数の実行環境(実環境、Xen環境、etc)対応に

Transparent Paravirtualization (その2)

- Xen Machine Vectorの導入
  - 初期化ルーチン、割り込み処理、IPI、I/Oアクセス、DMA処理等を環境毎に切り替える機構
  - Machine Vectorは元々は複数のIA64マシンで動作するように導入された
- is_running_on_xen()による処理の切り分け

is_running_on_xen()
1: Xen環境時
0: native環境時

```
if (is_running_on_xen()) {
Xenの時の処理
} else {
Nativeの時の処理
}
```

Memory Mapped Registers (a.k.a. Hyper Registers)

- 特権レジスタをメモリにマップ
  - Xen/x86でもイベントチャネル操作で使っている
  - 特定アドレスにマップする
  - 特に特権レジスタの値をメモリ参照で取得可能
- 特権レジスタ操作→メモリ操作
  - 特権レジスタ操作より早い
    - 例 割り込みのマスク/アンマスク(イベントチャネル操作)

HyperPrivops

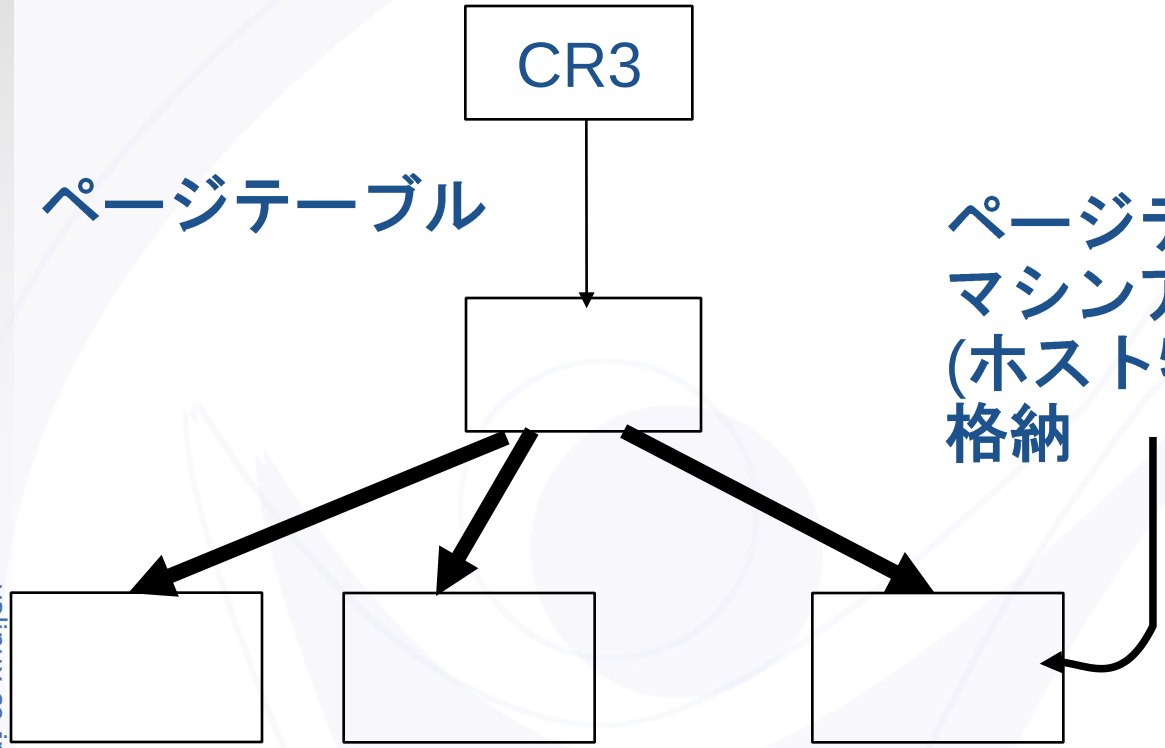
- 特権命令をハイパーコールに置き換え
  - 仮想化出来ない命令は必須
- Optimized Paravirtualization
  - 仮想化可能な命令も性能上重要なものは準仮想化を行う

HyperPrivops(その2)

- 特権操作はgcc_intrin.h, intel_intrin.hに纏められているのでそれらの関数/マクロを置き換える。
- ハンドコードされたアセンブリコードは手で書き換え
  - 割り込みハンドラ、レジスタ待避/復元コードなど
    - hyperregistersを活用
  - 少数のファイルかつ性能上重要なもの
    - ivt.S
    - entry.S

MMU準仮想化(x86)

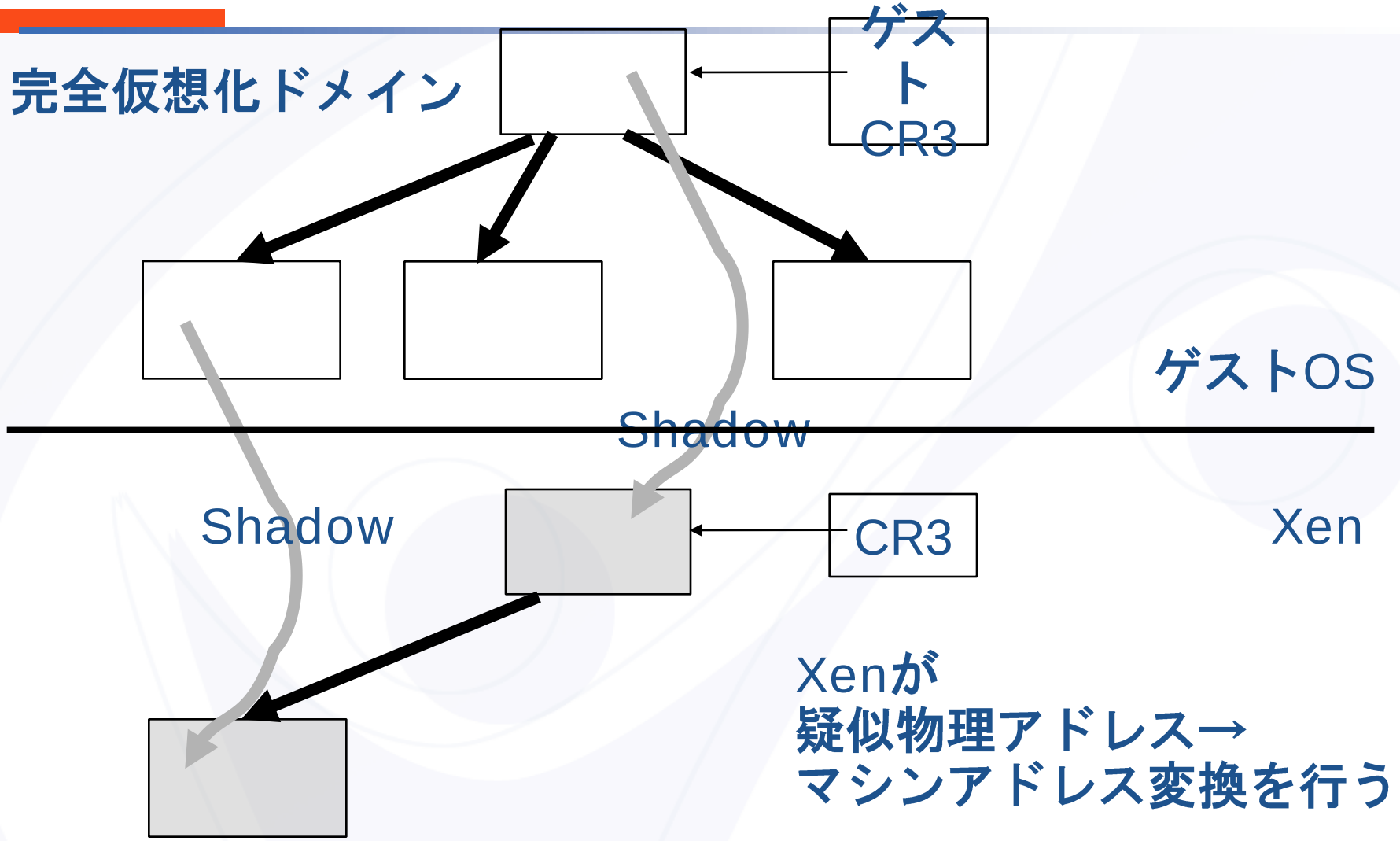
準仮想化ドメイン



ページテーブルエントリには
マシンアドレス
(ホスト物理アドレス)を
格納

ゲストOSが
直接書き換え
疑似物理アドレス
→マシンアドレス
変換をゲストOSが行う

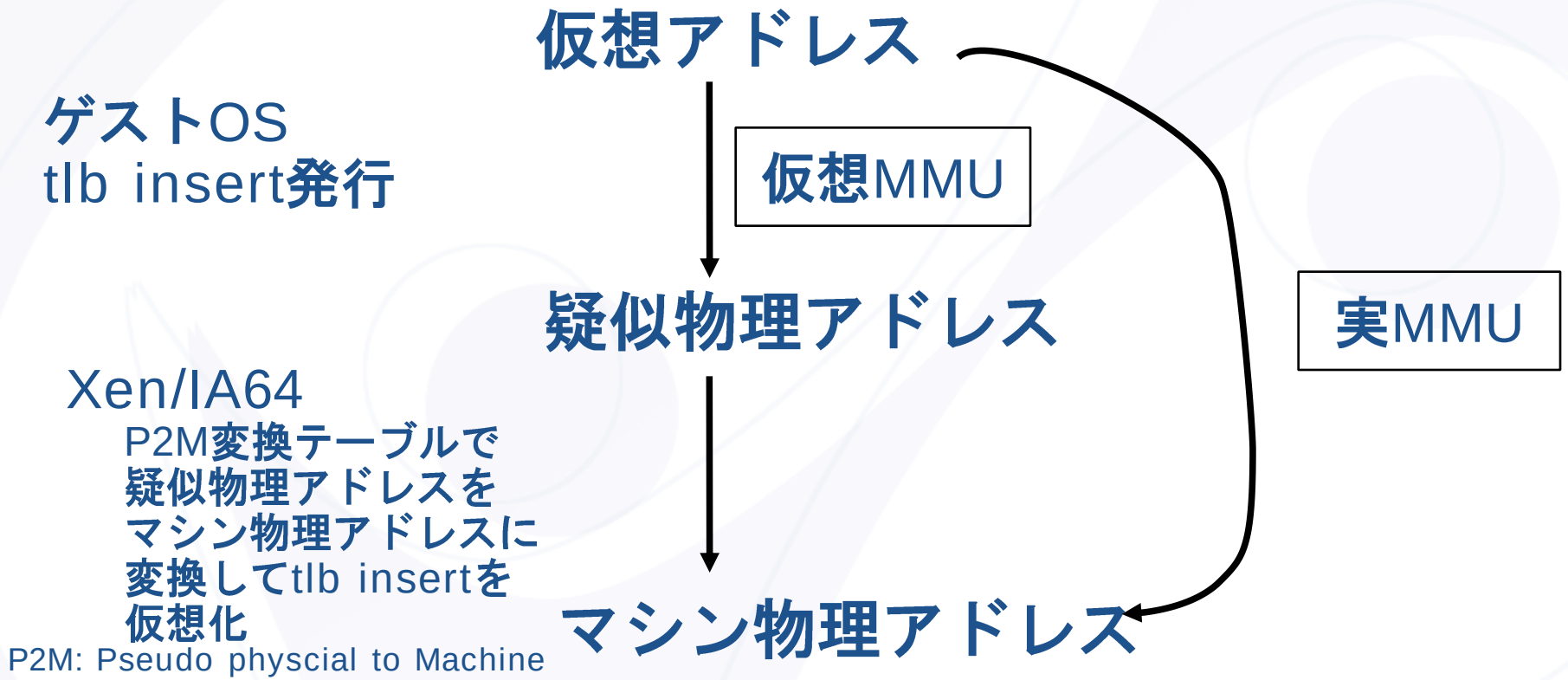
MMU仮想化(x86)



Xenが
疑似物理アドレス→
マシンアドレス変換を行う

MMU仮想化(IA64)

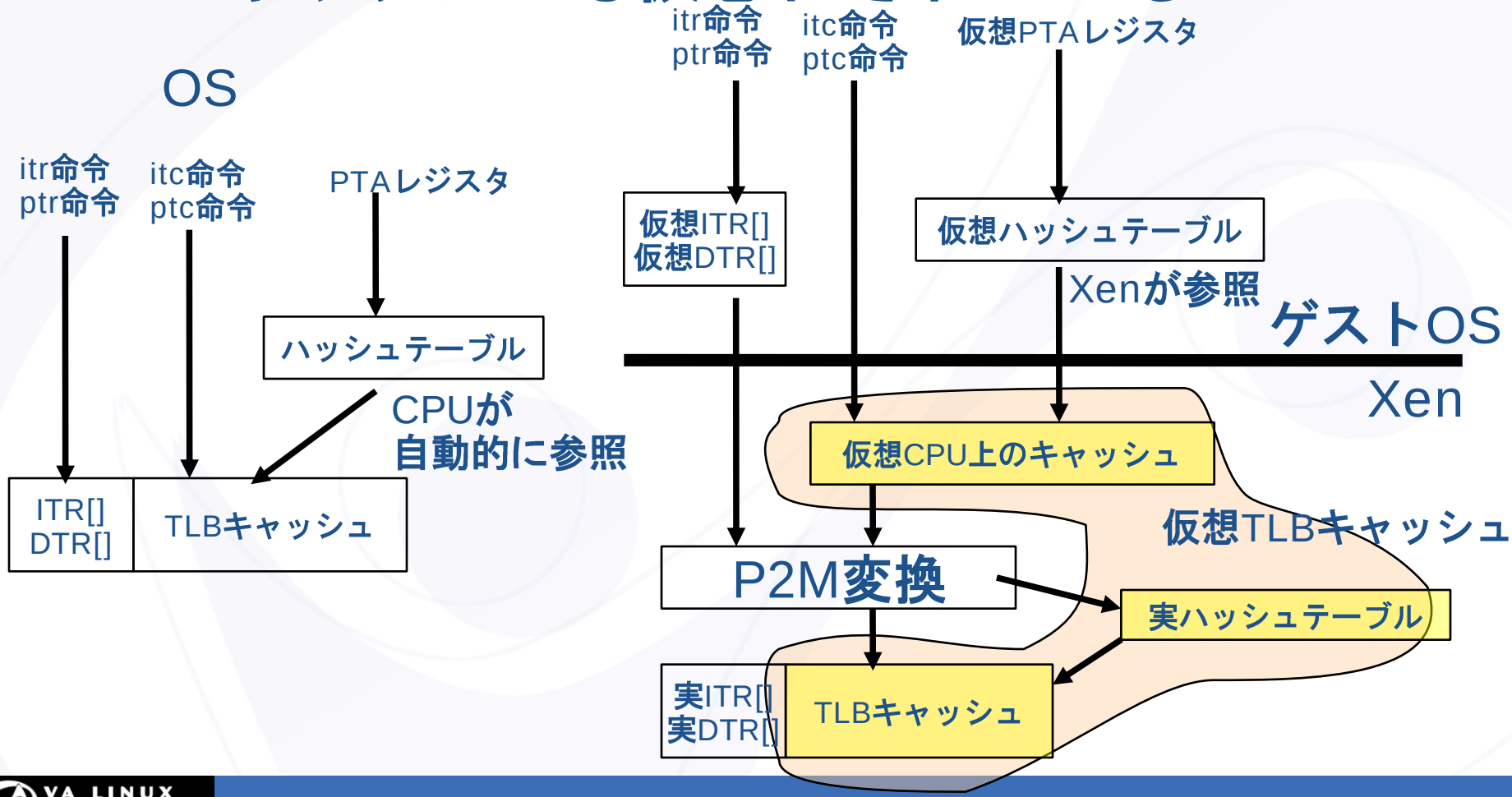
- MMUは完全仮想化されている
- “Shadow TLB”



P2M: Pseudo physcial to Machine

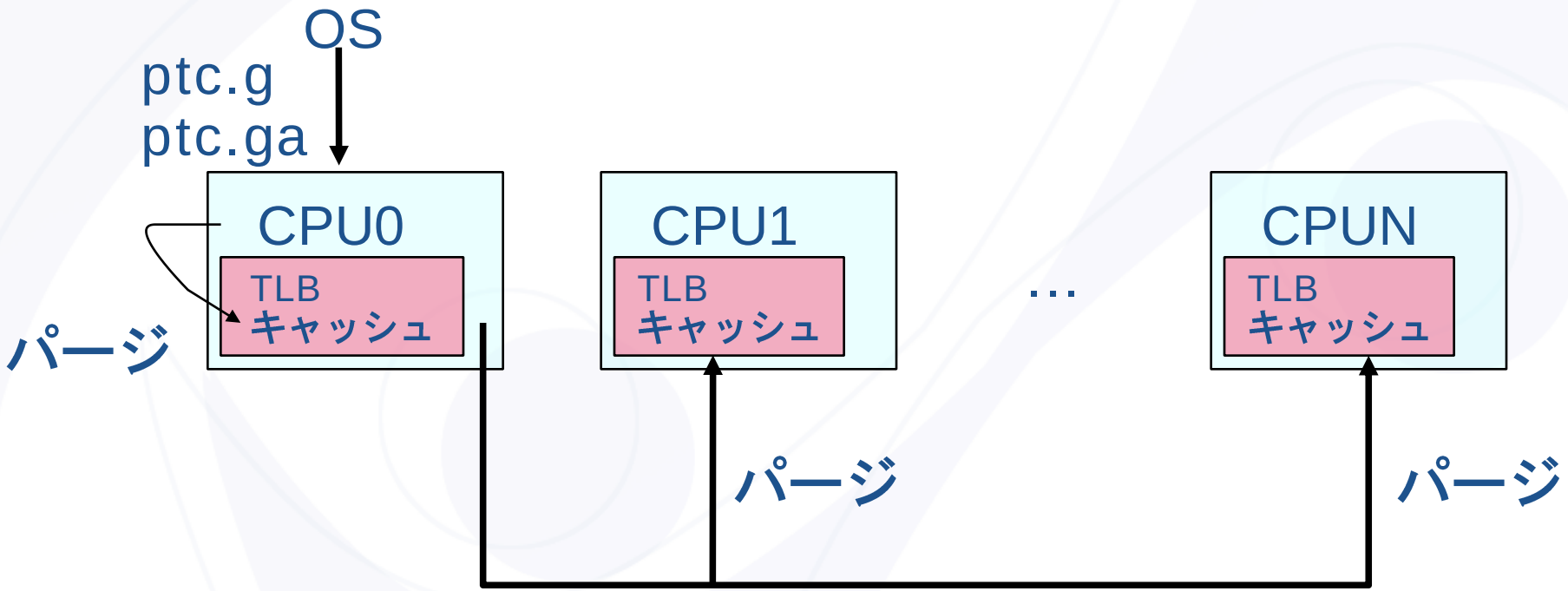
MMU仮想化(IA64その2)

- TLBキャッシュも仮想化されている

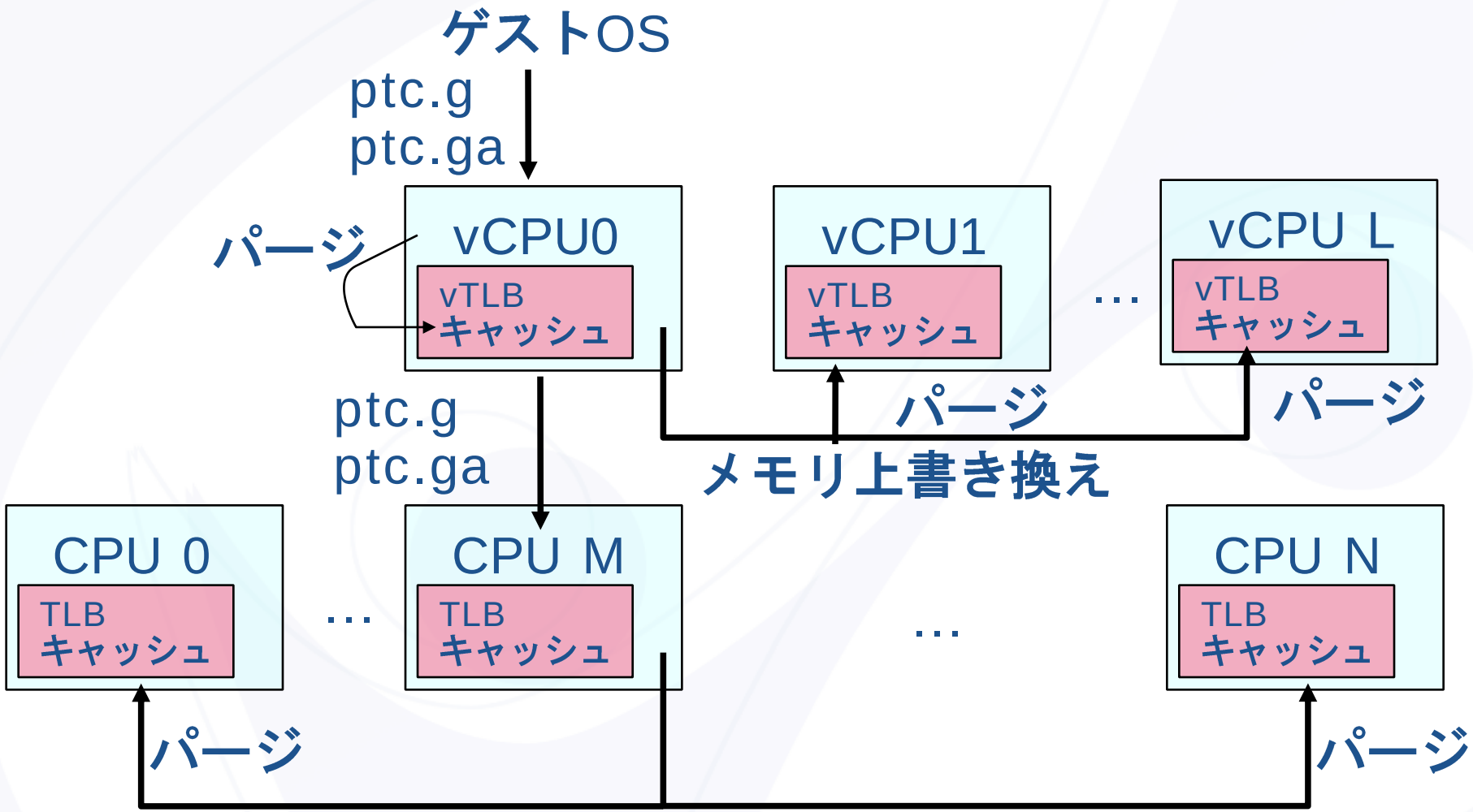


グローバルTLBパージ

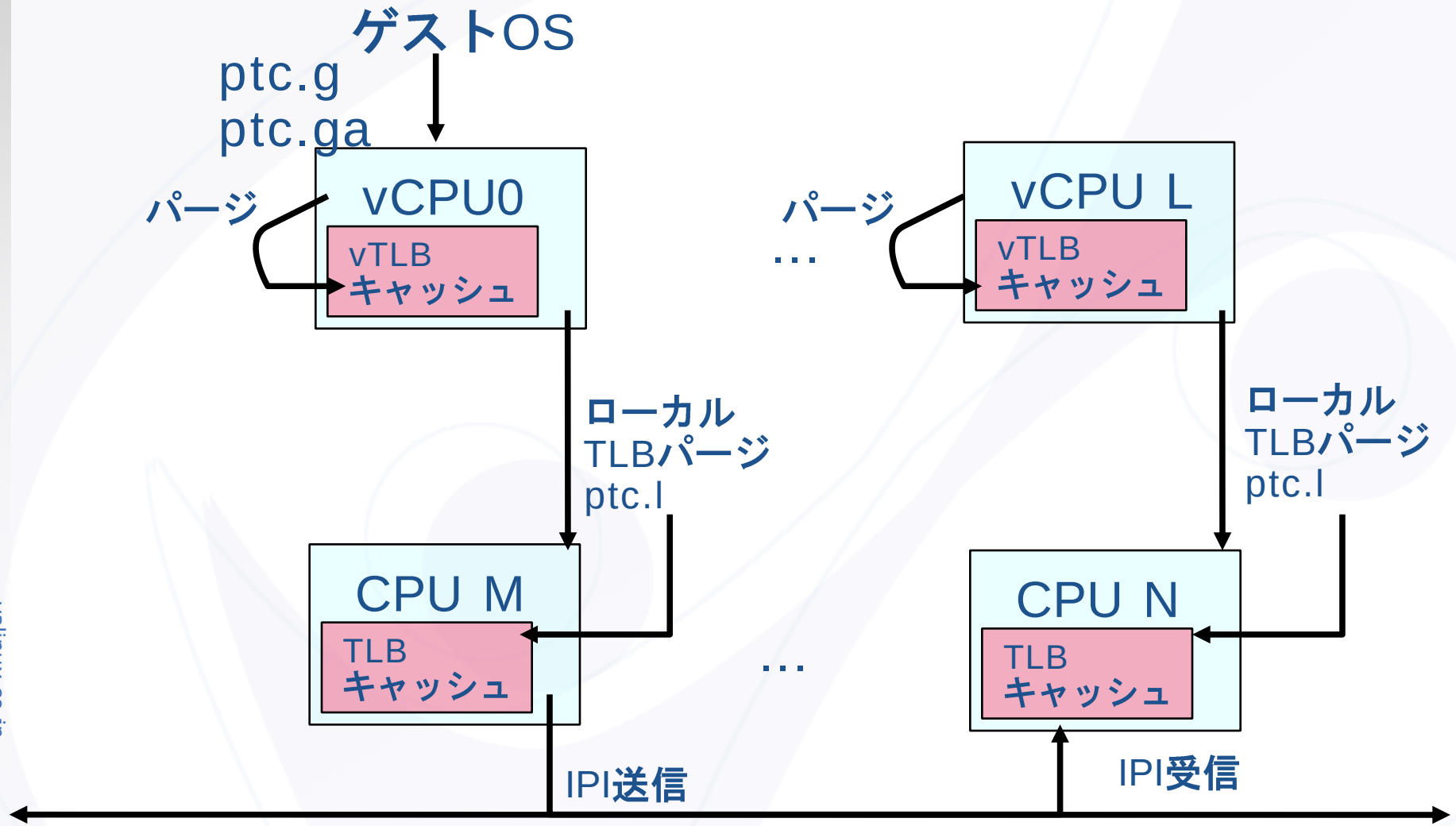
- ptc.g, ptc.ga命令はグローバルにTLBキャッシュをパージする



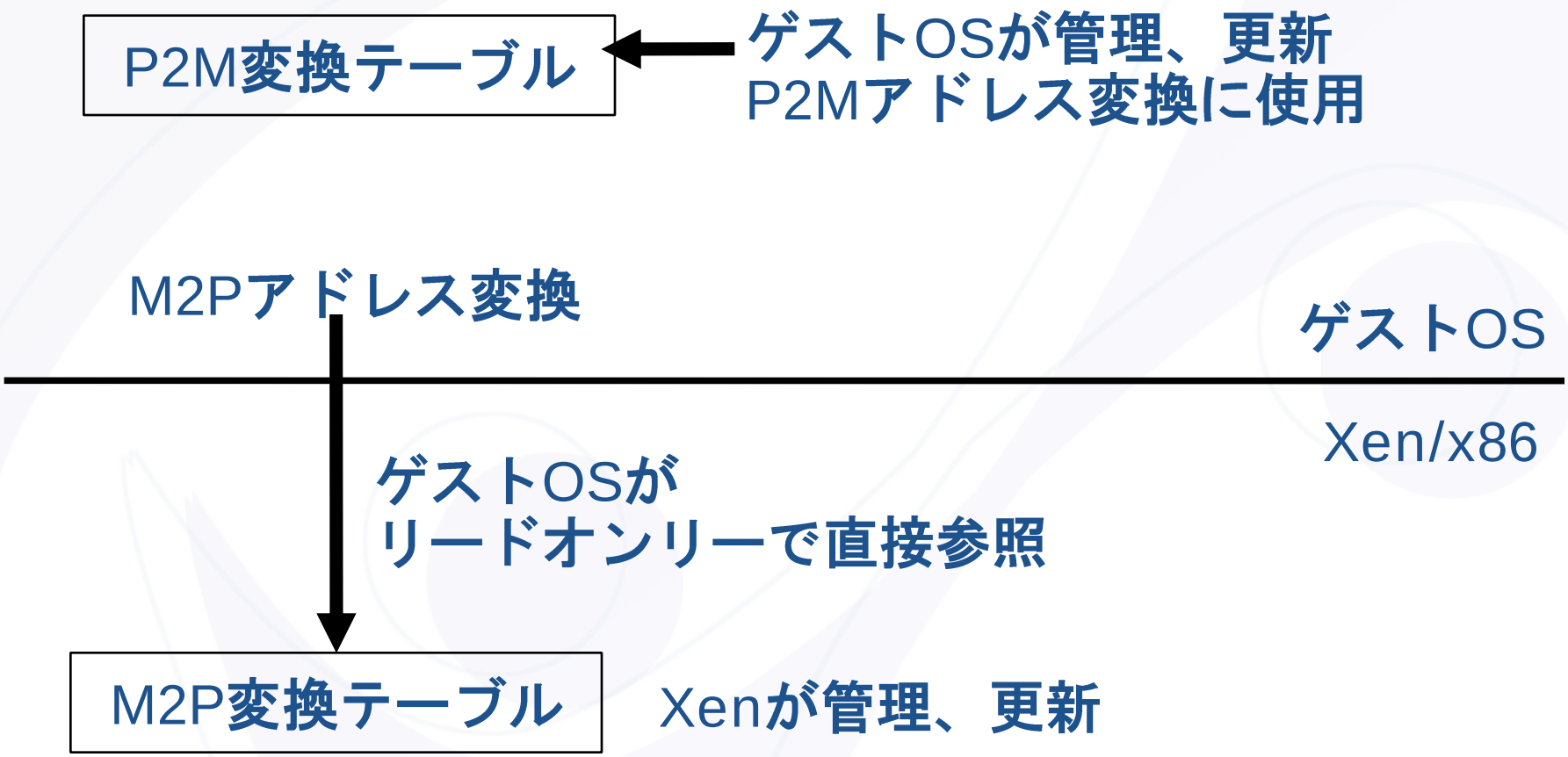
グローバルTLBパージ (準仮想化ドメイン)



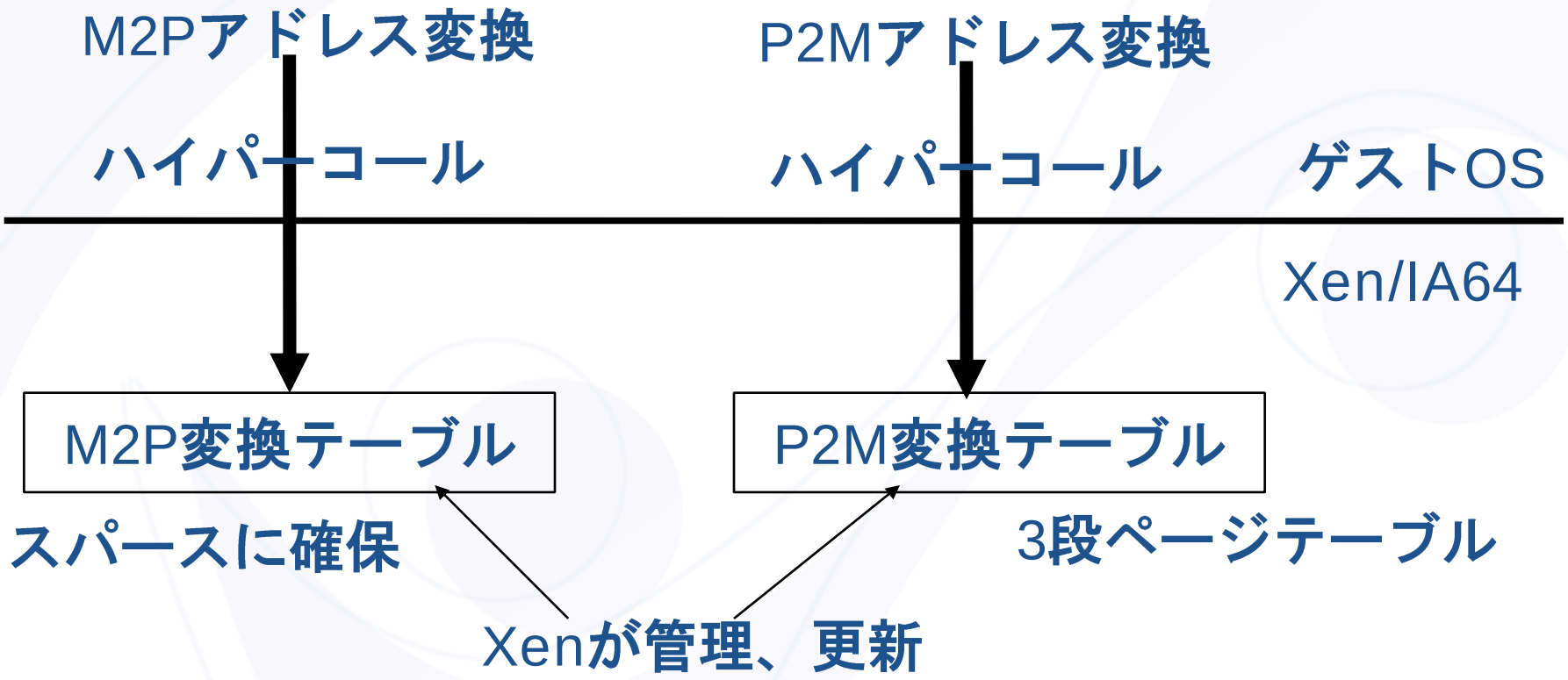
グローバルTLBパージ (完全仮想化ドメイン)



P2M/M2Pアドレス変換(x86)



P2M/M2Pアドレス変換(IA64)

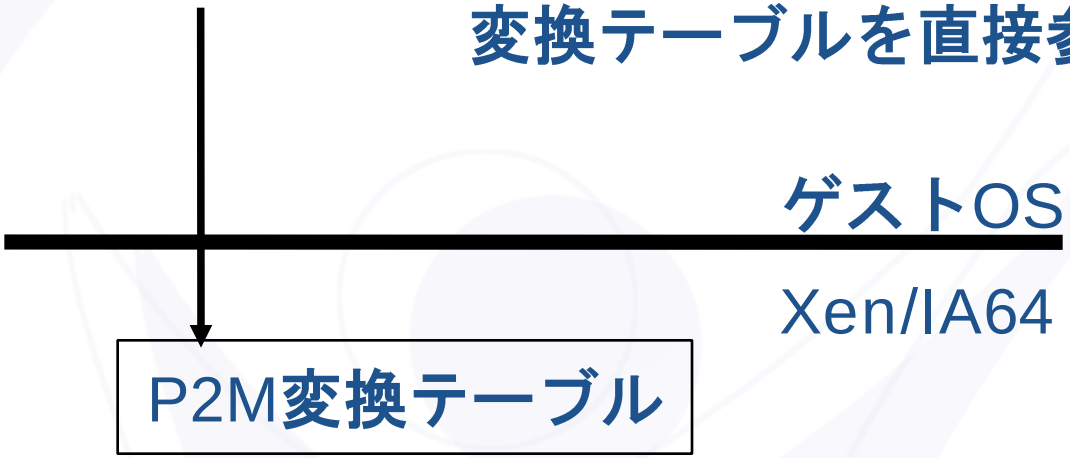


P2M table exposure

- 疑似物理アドレス→マシン物理アドレス変換にハイパーコールを省略

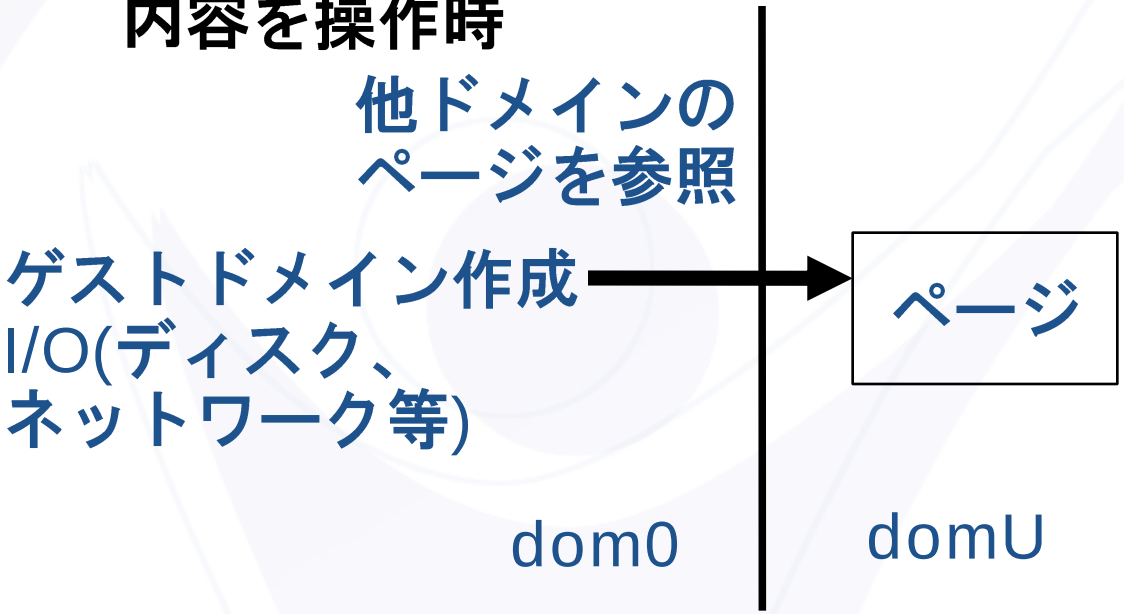
直接参照

変換テーブルを疑似物理アドレス空間にマップしてやることにより,ゲストOSが変換テーブルを直接参照



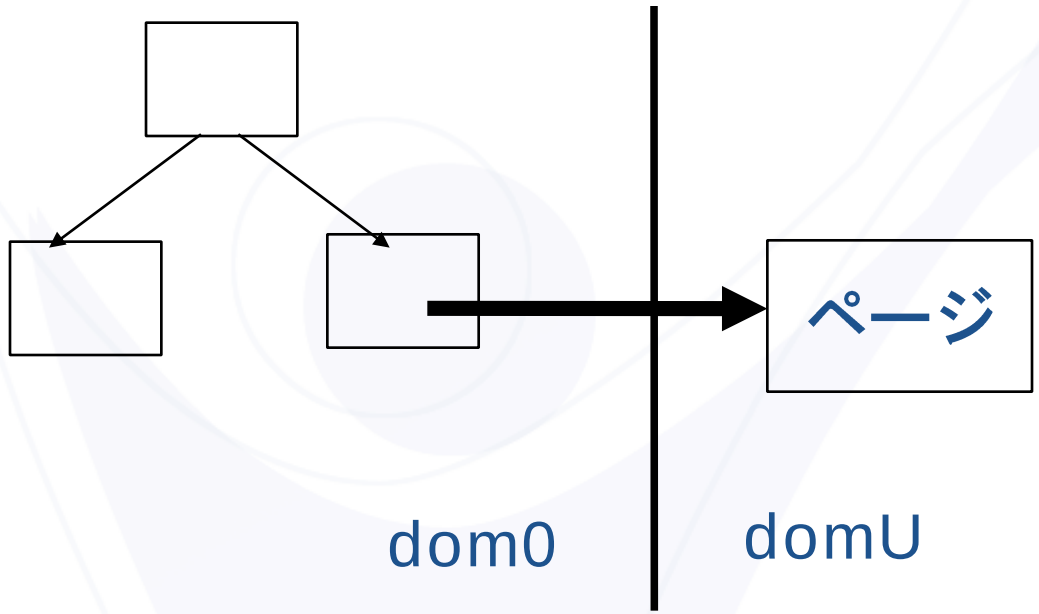
Foreign domain page mapping/grant table

- あるドメインが他ドメインのページを参照する仕組み
  - ゲストドメインを作成時
  - 仮想デバイスバックエンドがフロントエンドのページ内容を操作時



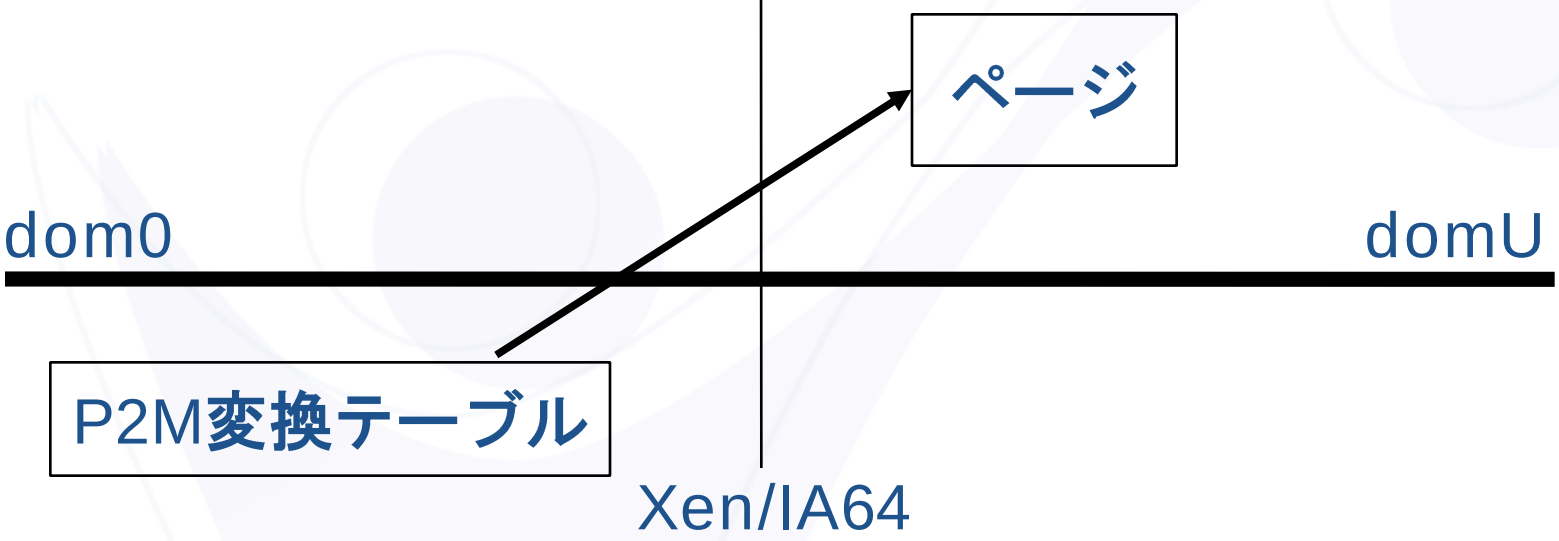
Foreign domain page mapping/grant table(x86)

- x86では仮想アドレスを使用してページテーブルが該当ページを指す事により実装されている



Foreign domain page mapping/grant table(IA64)

- IA64ではMMUが仮想化されている為に同様な実装ができない
  - 疑似物理アドレス領域を確保
  - ページを疑似物理アドレス空間にマップする。
  - P2Mテーブルの書き換え



TLB insert tracking

- ドメイン間でページを共有後、アンマップ時にはTLBフラッシュが必要
- IA64の場合フラッシュする為の仮想アドレスが不明
  - x86は仮想アドレスベース
  - TLBキャッシュが仮想化されているのでフルTLBフラッシュはとても重たい
    - ハッシュテーブルを全て無効化する必要がある
  - ハッシュテーブルの大きさを考えるとメモリコピーのコストの方が小さい

TLB insert tracking(その2)

- マップしたページに関するTLB insertを追跡してTLBフラッシュのコストを削減
  - 通常使用される仮想アドレスは1アドレスのみ
- P2Mテーブルに追跡状況を記録
  - 追跡を行っている/行っていない
  - tlb insertされた/されていない
    - 仮想アドレスは別途記録
  - 複数の仮想アドレスに対してtlb insertされた

TLB insert tracking(その3)

- 追跡していないページの場合
  - 追跡ビットを見てそれ以上の処理を行わない
- Disk I/Oの場合
  - Dom0はDMAを行うのみでTLBフラッシュ不要
  - 追跡ビットが立っているがtlb insertビットが立っていないことで確認する
- ネットワークI/Oの場合
  - パケットのヘッダを参照する必要がある。この場合1cpuのみで参照されることが多く、グローバルtlbパージでなくローカルtlbパージで済むことが多い。
  - tlb insertビットが立っているがtlb insertした物理CPUも追跡していて処理をグローバルtlbパージを省く

今後の開発

今後の開発

- 上流へのマージ
- kexec/kdump
- スケーラビリティー
- NUMA
- IOMMU
- スーパーページ

今後の開発(その2)

- ゲストドメインデバッガ
- Fast Hypercall
- パフォーマンスカウンタ(PMU/PMD)仮想化
- 完全仮想化ドメインでのドライバドメイン
  - IOMMU無しでも可能かも