

【TERM 最終発表】

RDMAを用いた, 遠隔ベアメタルマシン
デバッグのための仮想メモリの参照

Arch B3 tatsu

親: macchanさん,soraさん

背景

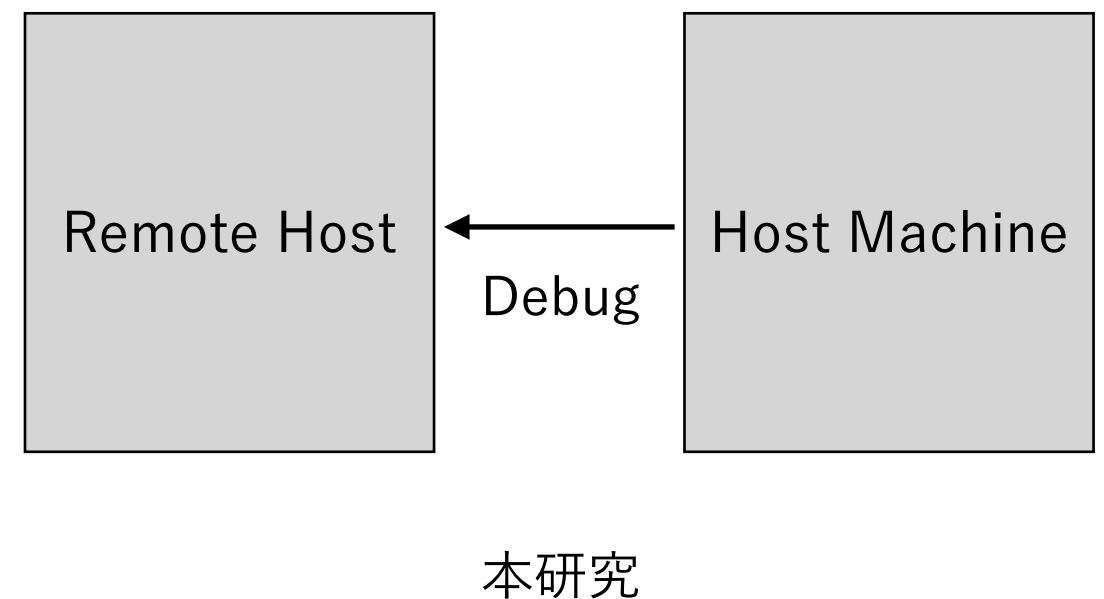
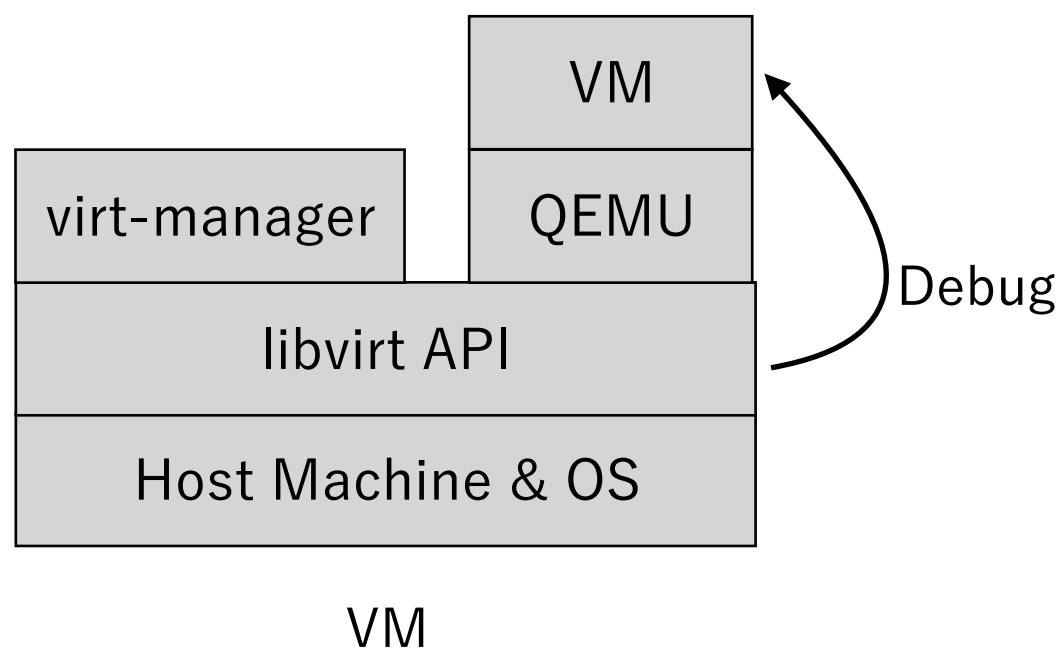
- OSデバッグやメモリフォレンジックをする際、物理マシン・仮想マシンを解析
 - 特に、カーネルパニック時の解析にはVMを利用
 - XenやQEMU + KVMなど
 - 関連ソフトウェア: libvmi, google/rekallなど

課題

- 物理マシンに対して、カーネルパニック時におけるメモリ解析手段が存在しない
- OSが提供していた機能が使えなくなるため
 - Memory Management Unit
 - システムコール
 - カーネルシンボル

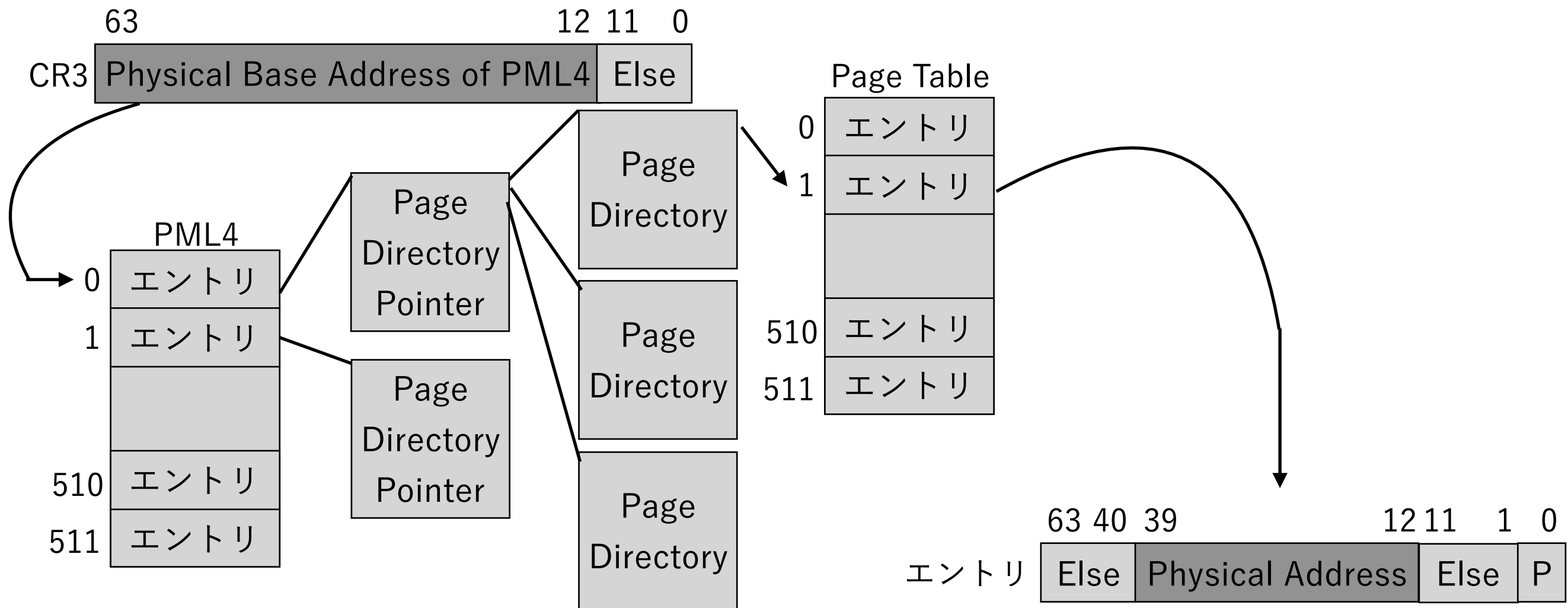
目的

- 遠隔ベアメタルマシンの物理メモリの監視アーキテクチャを確立
- 外部電源・外部プロセッサで動作
- **カーネルパニック時に特定のプロセスの仮想アドレス空間を復元**



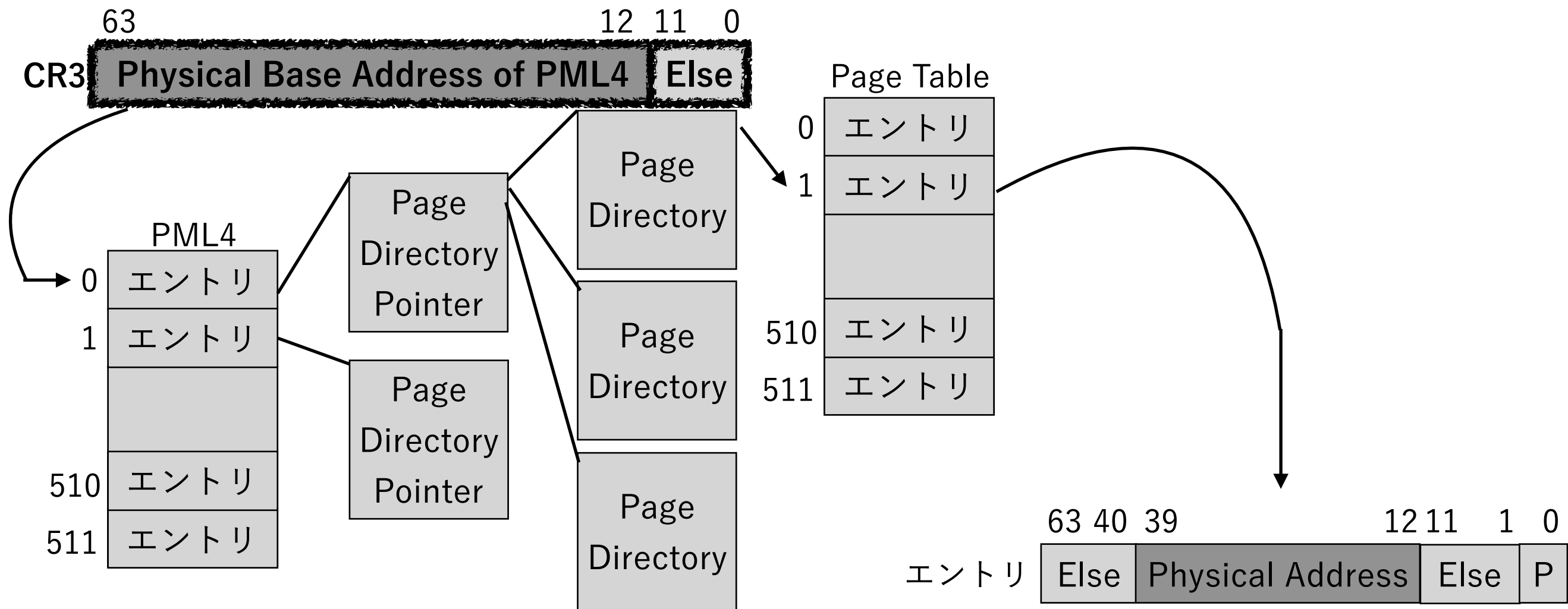
アプローチ

- ページウォークのロジックを解析
 - CR3**の値をもとに、ページウォークをし仮想アドレスを物理アドレスに変換



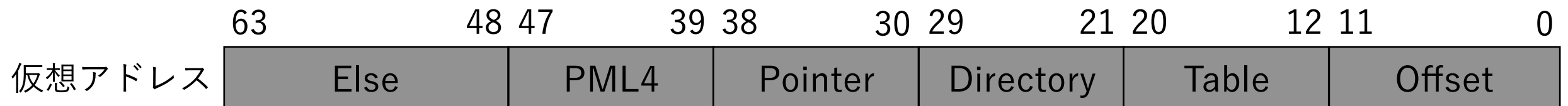
アプローチ

- **CR3**とは、ページウォークの起点であるPML4の物理アドレスを保持するレジスタ
- プロセスごとに設定されている



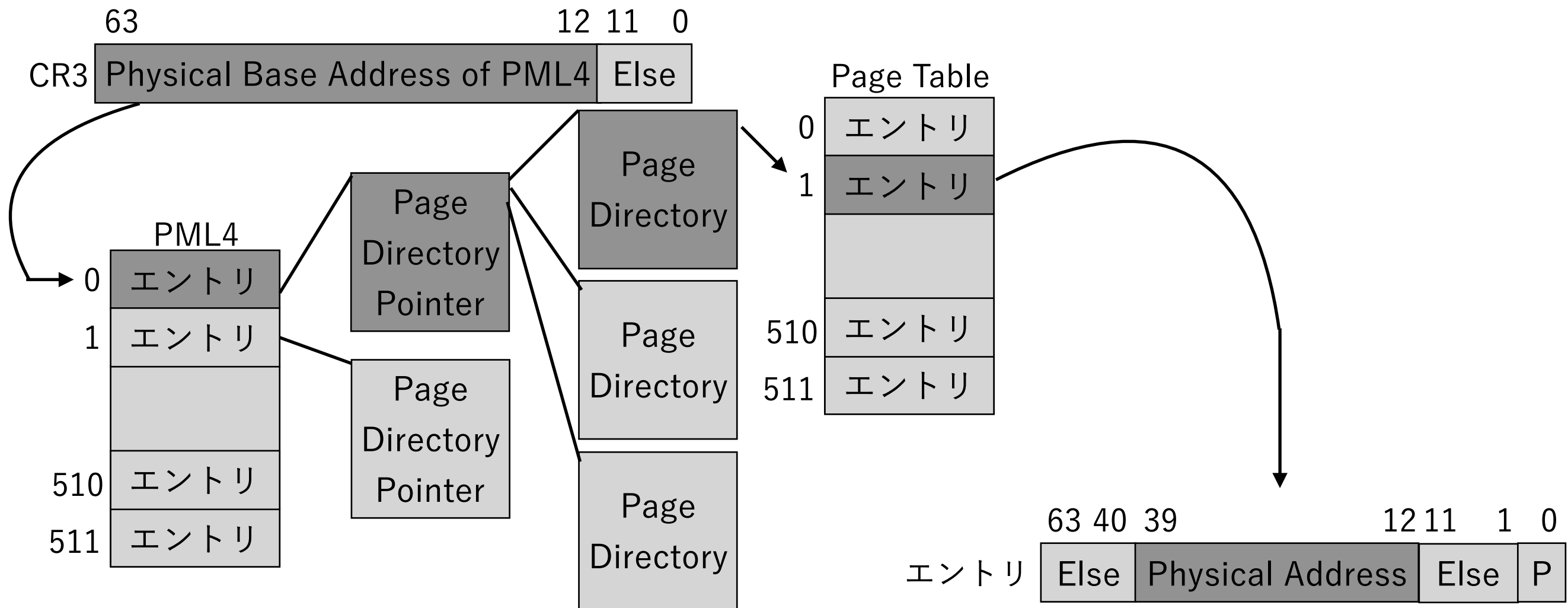
x86-64 Linuxのページウォーク

- テーブルは4段階存在(Page Map Level 4, Page Pointer, Page Directory, Page Table)
- 仮想アドレスは以下のフィールドに分割
- それぞれのフィールドには各テーブルのインデックス(0~511)が格納されている



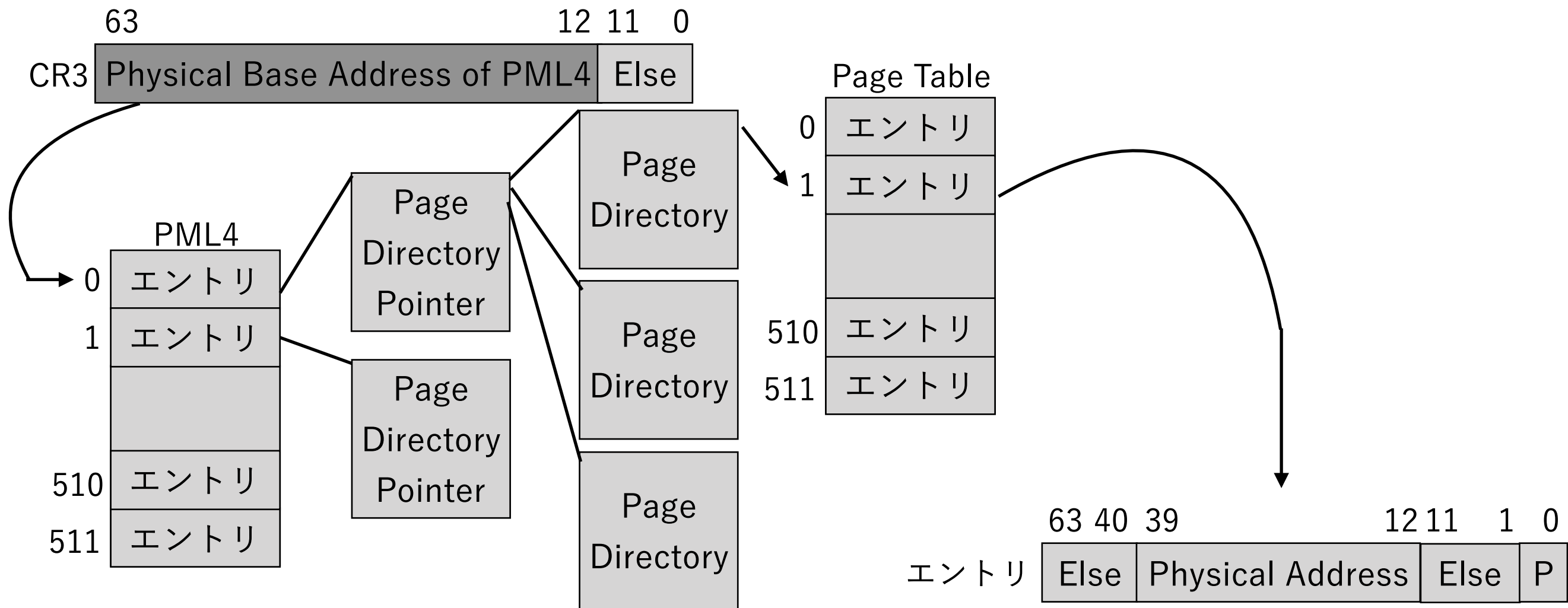
x86-64 Linuxのページウォーク(発表練習の時間次第で消すスライド)

- 仮想アドレスが**0x1000**の場合
- PML4 = **0**, Page Directory Pointer = **0**,
Page Directory = **0**, Page Table = **1**



アプローチ

- テーブルの全エントリを本研究のプログラムで保持
- ロジックを逆算し仮想アドレス空間を復元し，仮想メモリを参照する



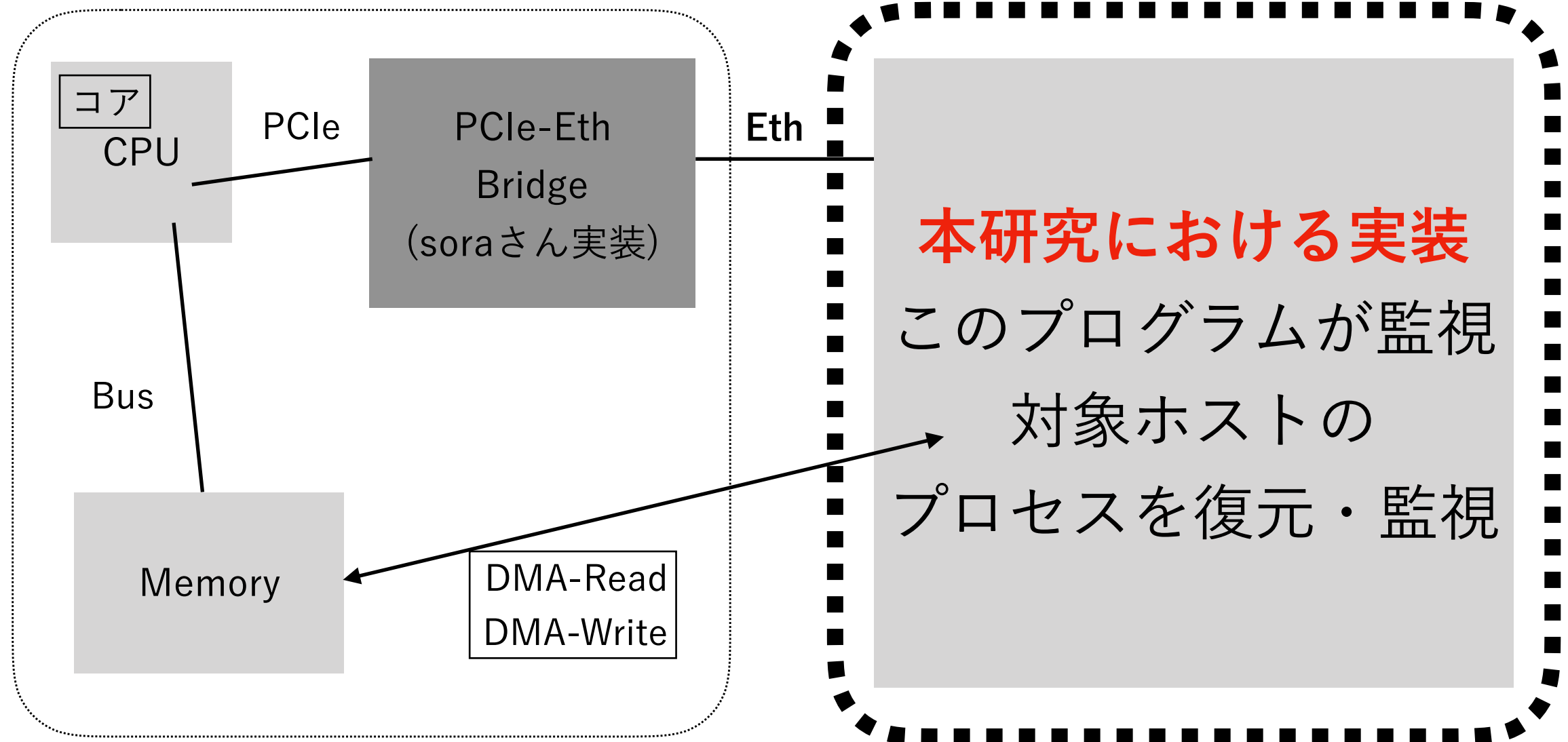
環境

- 監視プロセスが動くホスト
 - x86-64 Linux
- 監視対象ホスト
 - x86-64 Linux

システム概要

監視対象ホスト

監視ホスト



Ethernetフレーム

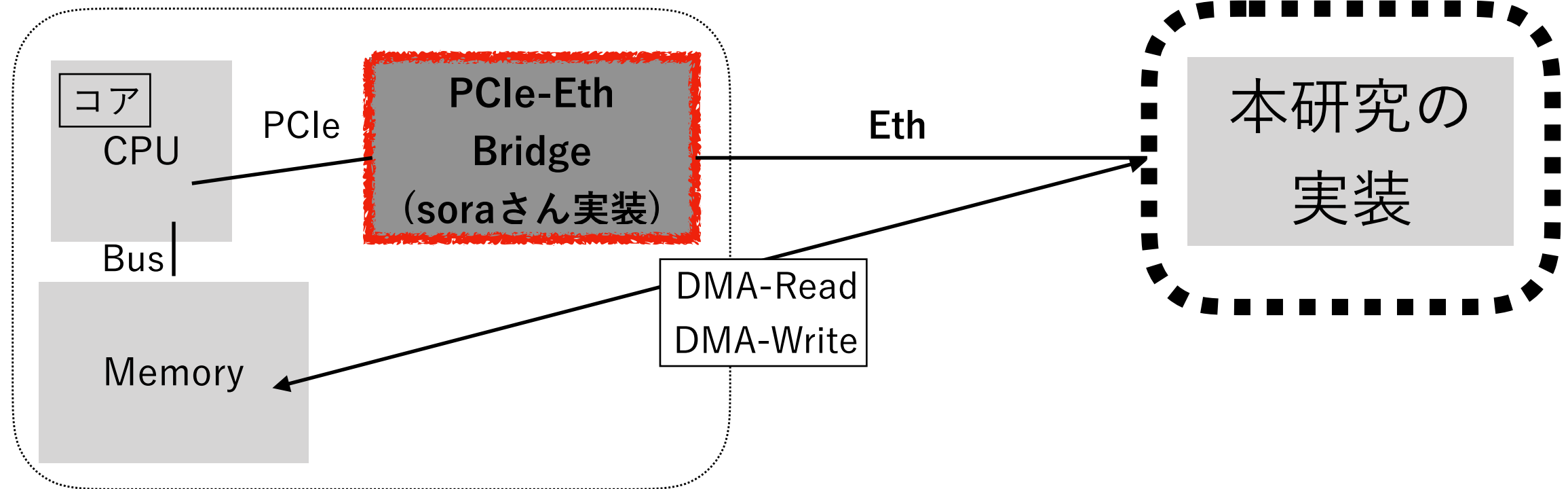
PCleメッセージ

PCle-Eth Bridgeのデータ構造

システム概要

監視対象ホスト

監視ホスト

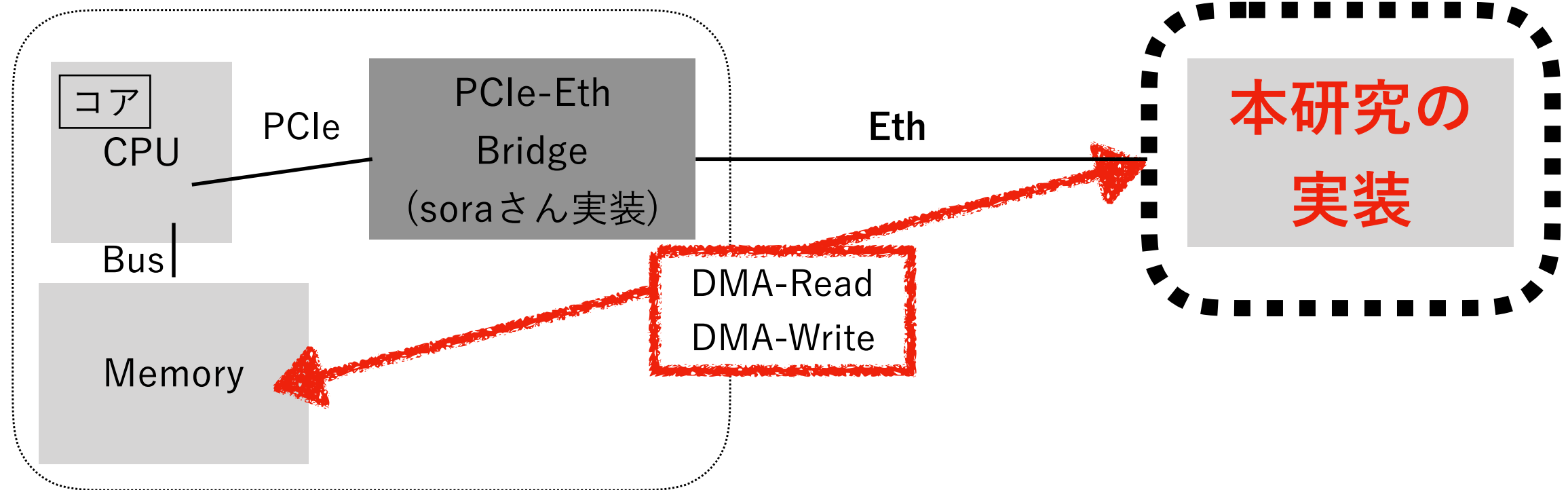


- PCI MessageとEthernetフレームを変換するFPGAデバイス
- 物理的に設置する

システム概要

監視対象ホスト

監視ホスト



- PCIe Eth Bridgeの手続き(read/write)を連続で呼び出す(4 Bytesごと)
- 連続した領域の値を取得

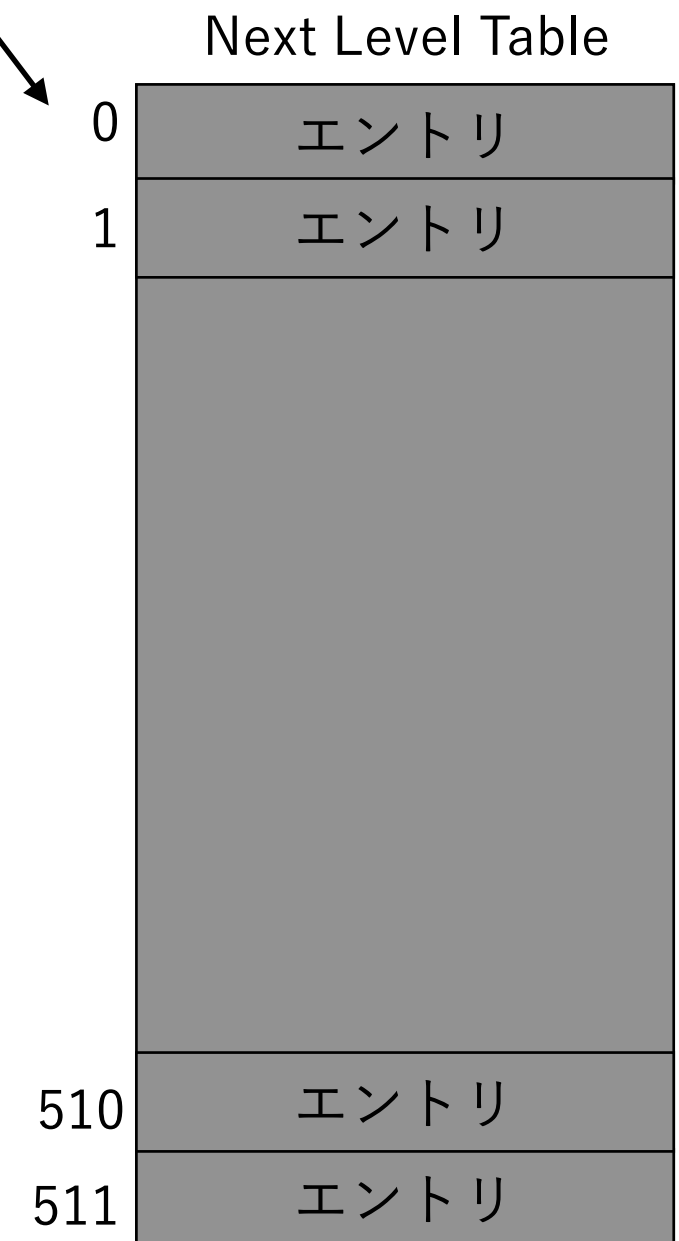
実装

- CR3の値を監視対象ホストから通知
 - CR3はレジスタであるため現在の値をメモリから参照できないため
- <https://github.com/doooooooooingggggg/dmaLinux/blob/master/src/dmaLinux.c>

実装

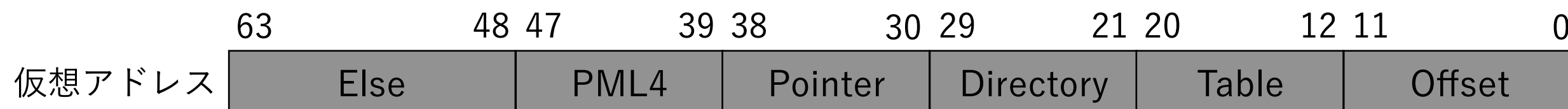


- CR3を起点に全てのエントリの情報を取得し**配列に保持**



実装

- 最終的にPage Tableから物理アドレス，値を取得
- 以下の処理に通すことで仮想アドレスの復元が完了
 $(\text{pml4Index} \ll 39) + (\text{pdpIndex} \ll 30) + (\text{pdIndex} \ll 21) + (\text{ptIndex} \ll 12) + \text{offset};$



評価

- カーネルパニック時の対象プロセスの仮想メモリ参照の可否
 - 状態を正しく取得できるか？
 - 手段として、現在の時間を変数として保持するプロセスを監視する
- 手順として、監視プロセスが動いている最中に、別コンソールから、意図的にカーネルパニックを起こす

```
$ sudo su -  
root$ echo c > /proc/sysrq-trigger
```

評価

- 監視するプロセスの詳細
- <https://github.com/doooooooooingggggg/getcr3/blob/master/user.c>
- CR3の値をカーネルモジュール経由でprintf()
 - macchanさん実装のカーネルモジュールを拡張
- 現在のUNIX TIMEをprintf()

- 出力例

```
# a:1548316892 (0x5c4970dc) at 0x7fff6509aa90
# CR3: 0x274be2000
```

評価

- 出力結果

```
# 0x7fff65099000 (phys[0x2697c9000]) : 0x0
# 0x7fff65099008 (phys[0x2697c9008]) : 0x0
# 0x7fff65099010 (phys[0x2697c9010]) : 0x0
# 0x7fff65099018 (phys[0x2697c9018]) : 0x0
# 0x7fff65099020 (phys[0x2697c9020]) : 0x0
# 0x7fff6509aa8c (phys[0x2697e5a8c]) : 0x5c4970de000000028
# 0x7fff6509aa94 (phys[0x2697e5a94]) : 0x6509aa9000000000
# 0x7fff6509aa9c (phys[0x2697e5a9c]) : 0x8005003300007fff
# 0x7fff6509aaa4 (phys[0x2697e5aa4]) : 0x75cf3b0000000000
# 0x7fff6509aaac (phys[0x2697e5aac]) : 0x309c3230ffff8802
# 0x7fff6509b000 (phys[0x269cba000]) : 0x0
# 0x7fff6509b008 (phys[0x269cba008]) : 0x0
# 0x7fff6509b010 (phys[0x269cba010]) : 0x0
# 0x7fff6509b018 (phys[0x269cba018]) : 0x0
# 0x7fff6509b020 (phys[0x269cba020]) : 0x0
```

評価

- 出力結果

```
# 0x7fff65099000 (phys[0x2697c9000]) : 0x0
# 0x7fff65099008 (phys[0x2697c9008]) : 0x0
# 0x7fff65099010 (phys[0x2697c9010]) : 0x0
# 0x7fff65099018 (phys[0x2697c9018]) : 0x0
# 0x7fff65099020 (phys[0x2697c9020]) : 0x0
# 0x7fff6509aa8c (phys[0x2697e5a8c]) : 0x5c4970de00000028
# 仮想アドレス (p物理アドレス4) : 0x変数の値00000000
# 0x7fff6509aa9c (phys[0x2697e5a9c]) : 0x000000000007fff
# a:1548316892 (0x5c4970dc) at 0x7fff6509aa90
# CR3: 0x274be2000
# 0x7fff6509b008 (phys[0x269cba008]) : 0x0
# 0x7fff6509b010 (phys[0x269cba010]) : 0x0
# 0x7fff6509b018 (phys[0x269cba018]) : 0x0
# 0x7fff6509b020 (phys[0x269cba020]) : 0x0
```

- 0x7fff6509aa90に変数の値が見える
- 5c4970de(= 1548316894 (UNIX TIME))にカーネルパニックが発生

結論

- CR3が通知された状態において、仮想メモリの参照が可能なことが実証
- 参照したい変数の値を確認できた

今後の展望(卒論に向けて)

- カーネル空間にあるtask_structをメモリのみから探す
 - CR3の値はtask_struct->mm_struct->pgdに格納
 - 通知なしで全プロセスの復元が可能になる
- PCIe Eth Bridgeを拡張