現在までの実装に関する説明

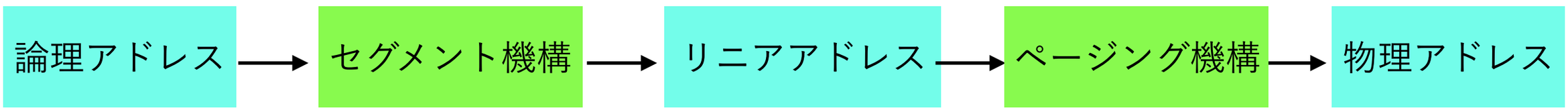
tatsu

前置き

- x86_64アーキテクチャにおいては，セグメント機構と，ページング機構がハードウェアで用意されている
  - Linuxでは，ページング機構のみを使用している．
- セグメント機構は無効にできないため，論理アドレスとリニアアドレスが等しくなるように設定されている(フラット)
- Linuxでは，仮想アドレス空間を実現するために，ページング機構が使用されている

前置き

- セグメント機構
  - 論理アドレスからリニアアドレスを算出する機構
- ページング機構
  - リニアアドレスから物理アドレスを算出する機構



前置き

- Linuxにおけるプロセスでは，アドレスの参照にリニアアドレスを用いている．

- しかし，実際にメモリ上にアクセス(読み書き)するときは，ページング機構を通し，物理アドレスを算出している．

- 詳しい仕組みについては，後述

本研究におけるアーキテクチャ

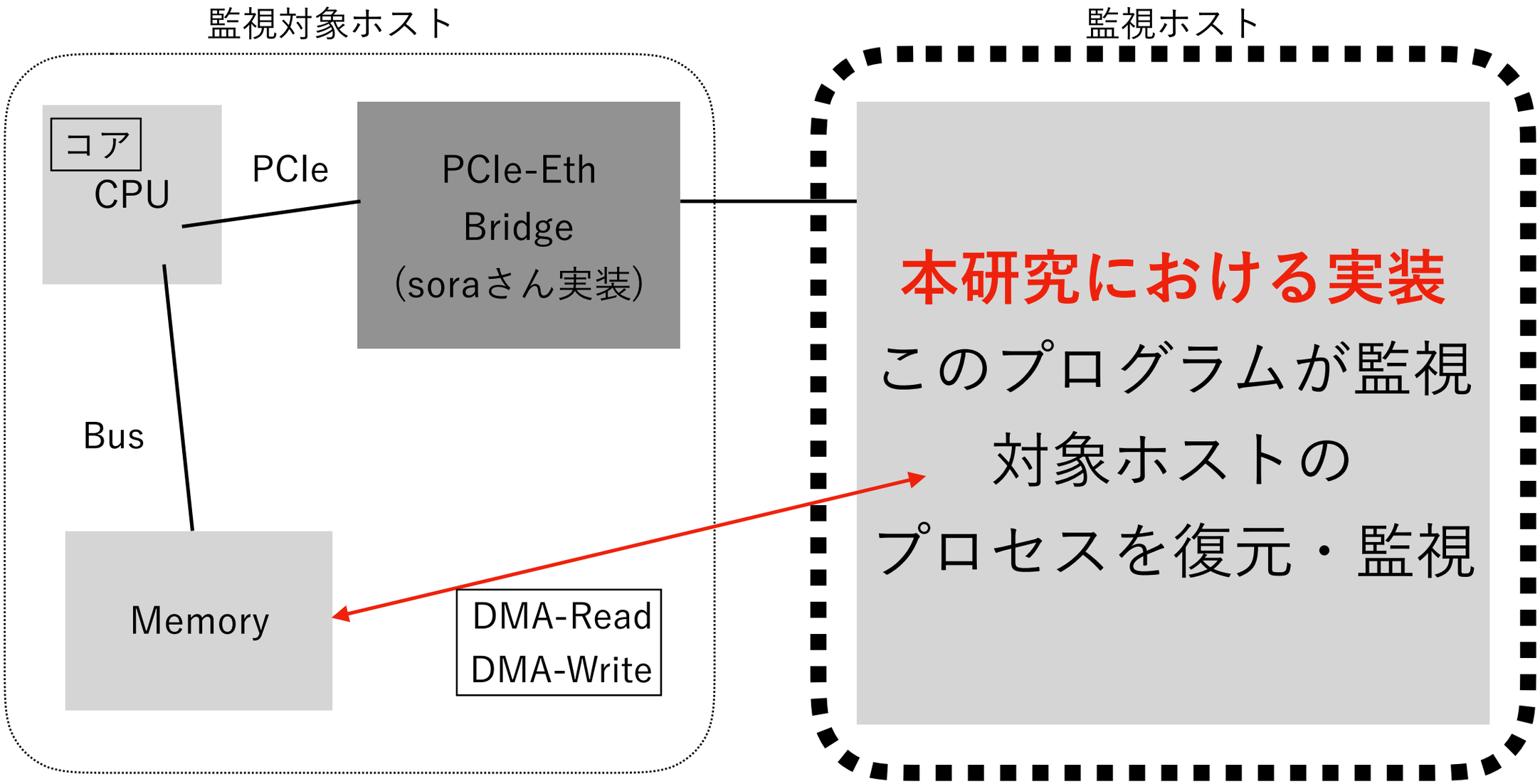
- PCIeバス上で動作するPCI-eth-bridghe(soraさん実装)を使用.
    - 物理アドレスを直接指定することで，そのアドレスから，4Byteの値を取得できるFPGAデバイス

本研究におけるアーキテクチャ

- 適切な物理アドレスを指定し値を本研究の実装(tatsu実装)で解釈

- 仮想アドレス空間(テキストセグメント，データセグメント)を復元する

本研究におけるアーキテクチャ



PCIe-Eth Bridgeのデータ構造

仮想アドレス→物理アドレス

- ページング機構の起点となるのは，cr3の値

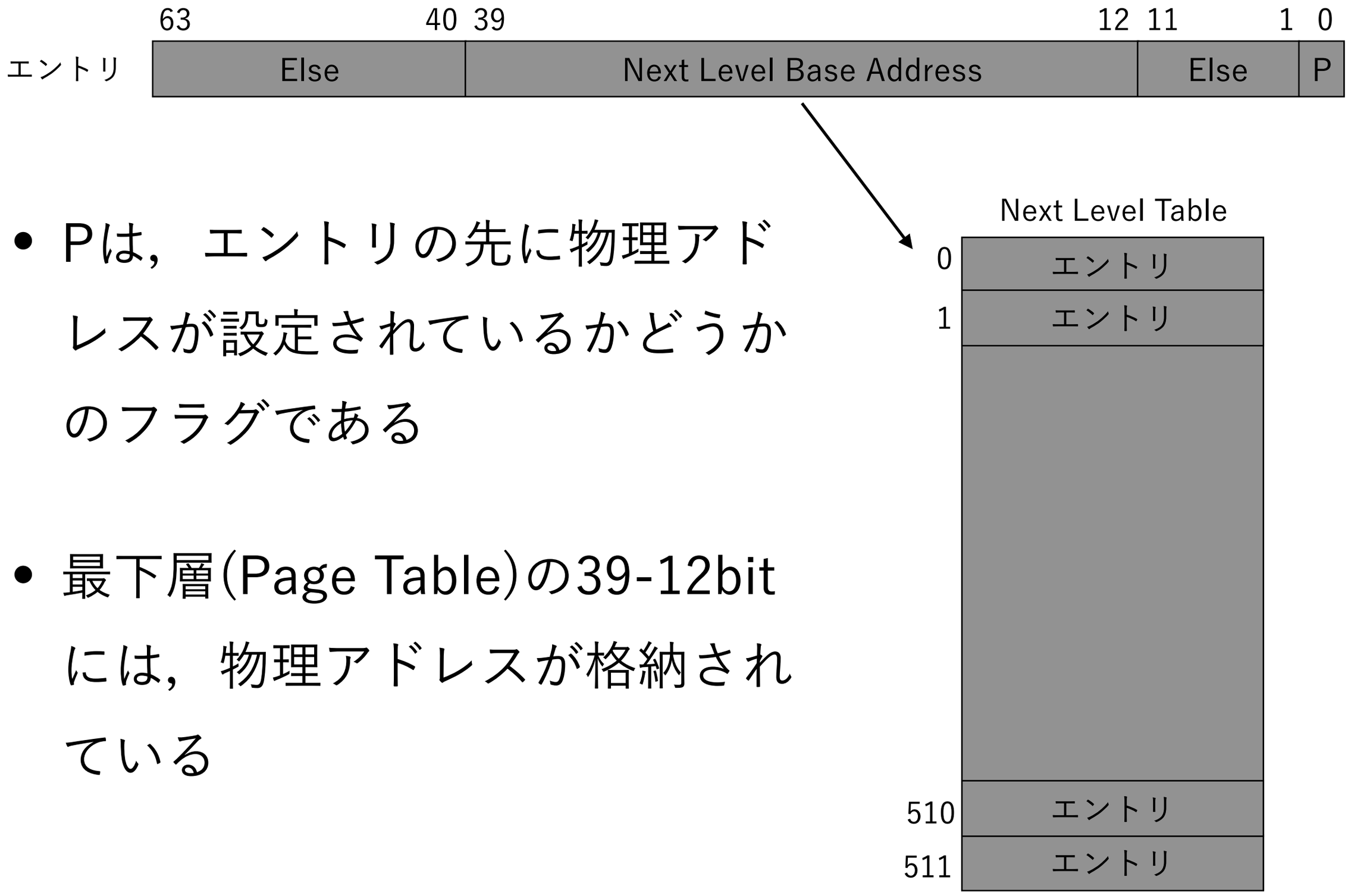
- cr3はレジスタであり，ページング機構の層の，最上位テーブルのベースアドレスが格納されている.

- 64bitにおいては，Page Map Level 4(PML4)というテーブルのベースアドレス

仮想アドレス→物理アドレス

- Page Map Level 4
    - Page Directory Pointer
        - Page Directory
            - Page Table
- それぞれ，512個のエントリがある
- 1エントリのサイズは8Byte
- テーブル全体のサイズは，4KB

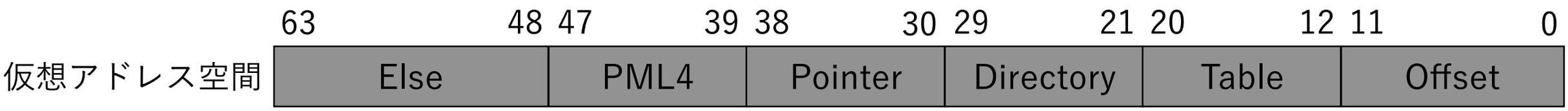
各テーブルの構造



- Pは，エントリの先に物理アドレスが設定されているかどうかのフラグである

- 最下層(Page Table)の39-12bitには，物理アドレスが格納されている

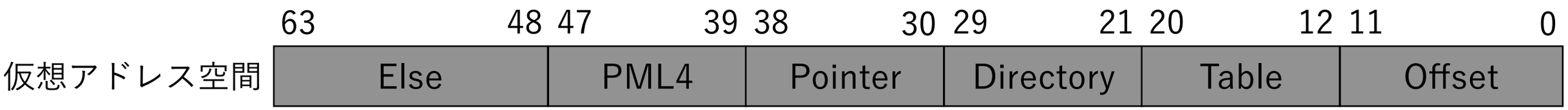
仮想アドレス→物理アドレス



- PML4~Tableには各階層のエントリのインデックス
- 9bitなので，0~511まで
  - エントリの数と一致
- 各テーブルの指定されたインデックスをたどる

- ここから，本研究での実装

仮想アドレス→物理アドレス



- 仮想アドレスが，0x7ffffffe358の場合
- PML4 → 255
- Pointer → 511
- Directory → 511
- Table → 510
- Offset → 856
- Tableの39-12bitの値(4KB境界)にOffsetの値を足す
- →物理アドレス算出

物理アドレス→仮想アドレス

- 現在の環境では，cr3の値を監視対象ホストから通知
- この値から4KB境界のアドレスを算出
- まずは，PML4のエントリを一つ一つ呼んでいく
- 愚直にやると，4階層で512 ** 4個のエントリを見る必要がある
  - 数を減らすためPフラグが0のものは，次の階層では除外

物理アドレス→仮想アドレス

- PML4, Pointer, Directory, Tableの各エントリから，次の階層のアドレスを取得
- Tableまで到達したら，物理アドレスの保持.
- 以下の構造体に格納

```
typedef struct _BINARY
{
    uint64_t pml4Index;
    uint64_t pdpIndex;
    uint64_t pdIndex;
    uint64_t ptIndex;
    uint64_t offset;
    uint64_t addr;
    uint64_t virtAddr;
    uint64_t value;
} BINARY;
```

物理アドレス→仮想アドレス

```
(ptIndex << 12) + (pdIndex << 21) +
(pdpIndex << 30) + (pml4Index << 39) +
offset;
```

↑仮想アドレスの復元が完了

物理アドレス→仮想アドレス

- 動作確認(テキストセグメントの復元)

```
0000000000400772 <main>:
  400772: 55                              push   %rbp
  400773: 48 89 e5                        mov    %rsp,%rbp # ①

  400776: 48 83 ec 60                     sub    $0x60,%rsp
  40077a: 89 7d ac                        mov    %edi,-0x54(%rbp)
  40077d: 48 89 75 a0                     mov    %rsi,-0x60(%rbp)
  400781: 64 48 8b 04 25 28 00  mov    %fs:0x28,%rax # ②
```

物理アドレス→仮想アドレス

- 動作確認(テキストセグメントの復元)

```
pml4BaseAddress :0x273e0a000
prease enter i1start,i1end
0,0 # indexを入力
pml4[0]:0x273e0a000: 0x8000000270c54067
PML4 Done
prease enter i2start,i2end
0,0 # indexを入力
pdpe[0][0]:0x270c54000: 0x27245d067
Page directory pointer Done
prease enter i3start,i3end
2,2 # indexを入力
pde[0][0][2]:0x27245d010: 0x273a21067
Page directory Done
prease enter i4start,i4end
0,0 # indexを入力
pte[0][0][2][0]:0x273a21000: 0x268253865
Page table Done
Phys Addr Read from Page Table Entry
virt:0x400640, phys:0x268253640: 0x66e0ff00601060bf
virt:0x400648, phys:0x268253648: 0x841f0f
virt:0x400650, phys:0x268253650: 0x2e6600401f0fc35d
.
.
.
.
virt:0x400940, phys:0x268253940: 0x2074612064253a61
virt:0x400948, phys:0x268253948: 0x64255243000a7025
virt:0x400950, phys:0x268253950: 0xa786c257830203a
virt:0x400958, phys:0x268253958: 0x3b031b0100000000
dmaRead done
```

```
-------------RESULT-------------
0x400640(phys[0x268253640]) :
0x66e0ff00601060bf
0x400648(phys[0x268253648]) : 0x841f0f
.
.
.
.
0x400758(phys[0x268253758]) :
0xc68948f08949f989
0x400760(phys[0x268253760]) : 0xb800400914bf
0x400768(phys[0x268253768]) :
0x90ffffe11e80000
0x400770(phys[0x268253770]) :
0x8348e5894855c3c9 # ①
0x400778(phys[0x268253778]) :
0x758948ac7d8960ec
0x400780(phys[0x268253780]) : 0x2825048b4864a0
# ②
.
.
.
0x400938(phys[0x268253938]) : 0xa7325206e6570
0x400940(phys[0x268253940]) :
0x2074612064253a61
0x400948(phys[0x268253948]) :
0x64255243000a7025
0x400950(phys[0x268253950]) :
0xa786c257830203a
0x400958(phys[0x268253958]) :
0x3b031b0100000000
```

物理アドレス→仮想アドレス

- 動作確認(テキストセグメントの復元)

```
a:456 at 0x7ffd382b7588 # 456 = 1C8
255 500 449 183 1416
CR0: 0x80050033
CR2: 0x7ffd82381f98
CR3: 0x77e50000
CR4: 0x160670
```

物理アドレス→仮想アドレス

- 動作確認(データセグメントの復元)

```
pml4BaseAddress :0x77e50000
prease enter i1start,i1end
254,256
pml4[254]:0x77e507f0: 0x8000000078cc3067
pml4[255]:0x77e507f8: 0x8000000077e70067
pml4[256]:0x77e50800: 0x0
PML4 Done
prease enter i2start,i2end
499,501
pdpe[254][499]:0x78cc3f98: 0x0
pdpe[254][500]:0x78cc3fa0: 0x0
pdpe[254][501]:0x78cc3fa8: 0x0
pdpe[255][499]:0x77e70f98: 0x0
pdpe[255][500]:0x77e70fa0: 0x7f1fd067
pdpe[255][501]:0x77e70fa8: 0x0
Page directory pointer Done
prease enter i3start,i3end
448,450
pde[254][0][448]:0xe00: 0x0
pde[254][0][449]:0xe08: 0x0
pde[254][0][450]:0xe10: 0x0
pde[255][500][448]:0x7f1fde00: 0x0
pde[255][500][449]:0x7f1fde08: 0x7f236067
pde[255][500][450]:0x7f1fde10: 0x0
Page directory Done
prease enter i4start,i4end
182,184
pte[254][0][0][182]:0x5b0: 0x0
pte[254][0][0][183]:0x5b8: 0x0
pte[254][0][0][184]:0x5c0: 0x0
pte[255][500][449][182]:0x7f2365b0: 0x8000000033837867
pte[255][500][449][183]:0x7f2365b8: 0x8000000034399867
pte[255][500][449][184]:0x7f2365c0: 0x80000000344f3865
Page table Done
Phys Addr Read from Page Table Entry
Skip0x33837000
prease offset_start,offset_range
150,200
virt:0x7ffd382b74b0, phys:0x343994b0: 0x0
virt:0x7ffd382b74b8, phys:0x343994b8: 0x4
.
.
.
.
virt:0x7ffd382b7628, phys:0x34399628: 0x0
virt:0x7ffd382b7630, phys:0x34399630: 0x7c08cde3bafacbcf
virt:0x7ffd382b7638, phys:0x34399638: 0x7d758bda3eeacbcf
Skip0x344f3000
dmaRead done
```

```
-------------RESULT-------------
.
.
.
.
0x7ffd382b7570(phys[0x34399570]) : 0x7ffd382b76b8
0x7ffd382b7578(phys[0x34399578]) : 0x100000000
0x7ffd382b7580(phys[0x34399580]) :
0x5f5f00656d697474
0x7ffd382b7588(phys[0x34399588]) : 0x5000001c8
0x7ffd382b7590(phys[0x34399590]) : 0x2800000003
0x7ffd382b7598(phys[0x34399598]) : 0x7ffd382b7588
0x7ffd382b75a0(phys[0x343995a0]) : 0x80050033
0x7ffd382b75a8(phys[0x343995a8]) :
0xffff880274e48f00
0x7ffd382b75b0(phys[0x343995b0]) : 0x7ff77b43ce50
0x7ffd382b75b8(phys[0x343995b8]) : 0x77e50000
0x7ffd382b75c0(phys[0x343995c0]) : 0x160670
0x7ffd382b75c8(phys[0x343995c8]) :
0xee14269bab0ed100
0x7ffd382b75d0(phys[0x343995d0]) : 0x400890
0x7ffd382b75d8(phys[0x343995d8]) : 0x7f439b37b830
0x7ffd382b75e0(phys[0x343995e0]) : 0x0
0x7ffd382b75e8(phys[0x343995e8]) : 0x7ffd382b76b8
0x7ffd382b75f0(phys[0x343995f0]) : 0x100000000
0x7ffd382b75f8(phys[0x343995f8]) : 0x400772
0x7ffd382b7600(phys[0x34399600]) : 0x0
0x7ffd382b7608(phys[0x34399608]) :
0x83f2bd35401acbcf
0x7ffd382b7610(phys[0x34399610]) : 0x4005f0
0x7ffd382b7618(phys[0x34399618]) : 0x7ffd382b76b0
0x7ffd382b7620(phys[0x34399620]) : 0x0
0x7ffd382b7628(phys[0x34399628]) : 0x0
0x7ffd382b7630(phys[0x34399630]) :
0x7c08cde3bafacbcf
0x7ffd382b7638(phys[0x34399638]) :
0x7d758bda3eeacbcf
```

物理アドレス→仮想アドレス

- 動作確認(テキストセグメントの復元)

$ sudo ~/apps/dma/dam-write **0x34399588** 123

実行すると↓
(~/getcr3/user in azkaban)
a:**456** at 0x7ffd382b7588
255 500 449 183 1416
CR0: 0x80050033
CR2: 0x6b6090
CR3: 0x77e50000
CR4: 0x160670
a:**291** at 0x7ffd382b7588
255 500 449 183 1416
CR0: 0x80050033
CR2: 0x6b6090
CR3: 0x77e50000
CR4: 0x160670

値が変化した

まとめ

- cr3の値がわかる状態であれば，復元ができる状態
- 今は環境により，全てのアドレスを網羅することはできない
- 各階層のインデックスを通知するなどすることで，限定的に仮想アドレス空間を復元

今後

- Linuxのカーネル空間に存在するtask_struct[]を探す
- task_struct->mm_struct-> pgdの値を取得
- pgdがcr3にロードされるため,
  - 各プロセスのpgdの値を取得することで，全てのプロセスの仮想アドレス空間の状態を復元できる