5. Virtual Memory

5.1 Introduction

Motivation

① Efficient memory sharing among several virtual machines (processes)

② The MM has limited size => burden

MM need contain only the active portions of many Virtual Machines

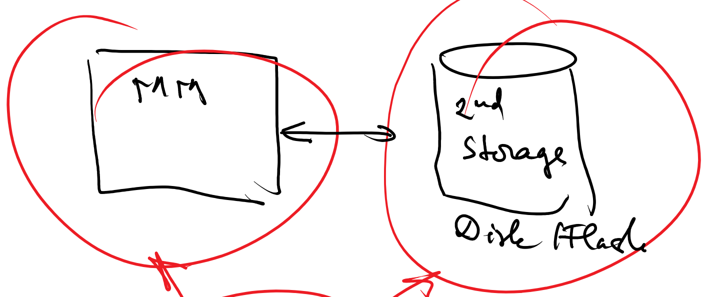
VM - allows us to efficiently share CPU & MM among virtual machines according to locality principle

- each program has its own address space
- programs dynamically interact among them

Virtual Addresses $\xmapsto{\text{Translation process}}$ Physical Address

MMU - Memory Management Unit → Translation, → Protection

Historically → Overlay



VM block → page

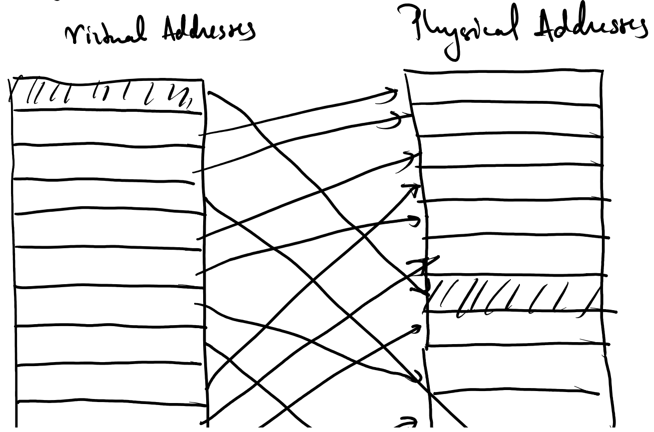
VM miss → page fault

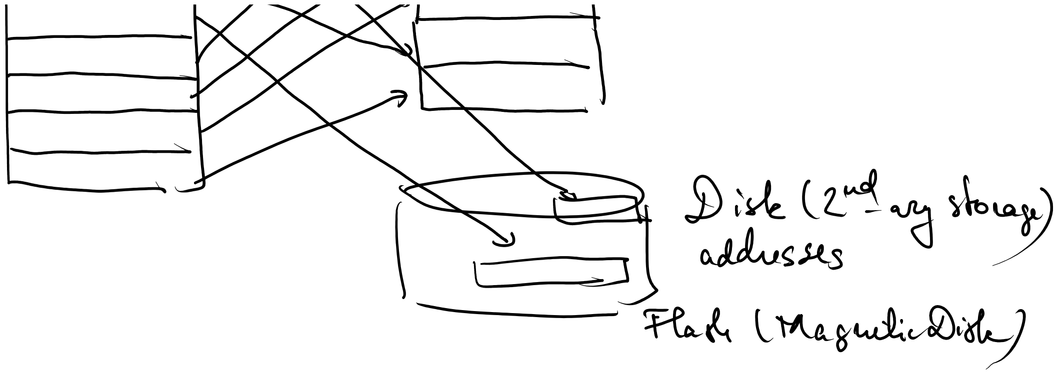
CPU $\xrightarrow{\text{virtual address}}$ MMU → physical address

- segment
- page ✓

} VM systems

Address mapping





VM allows to load the program anywhere in MM

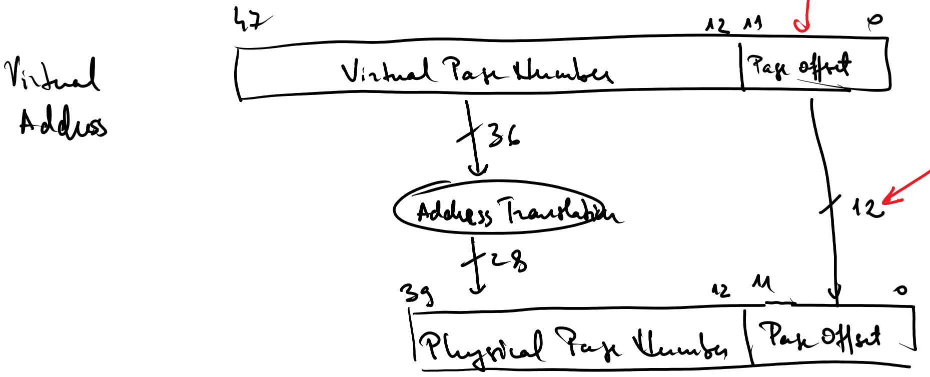
- Realocation of fixed-size pages

ARM v.8 Address word – 64 bit (First 16 bits are not used)

1 page = 4 KiB = $2^2 \cdot 2^{10}$ B = 2^{12} B

$2^{10} = 1$ ki
$2^{20} = 1$ Mi
$2^{30} = 1$ Gi
$2^{40} = 1$ Ti

2^{48} B = 256 TiB — Virtual Address Space Size



2^{40} = 1TiB Physical Address Space Size

48 − 12 = 36   40 − 12 = 28

2^{36} pages in VM = 64 G pages
2^{28} pages in PM = 256 M pages

Design choises – motivated by the large cost of a page fault (millions of clock cycles)
– MM is 100 000 quicker than the disk => enormous Miss Penalty !

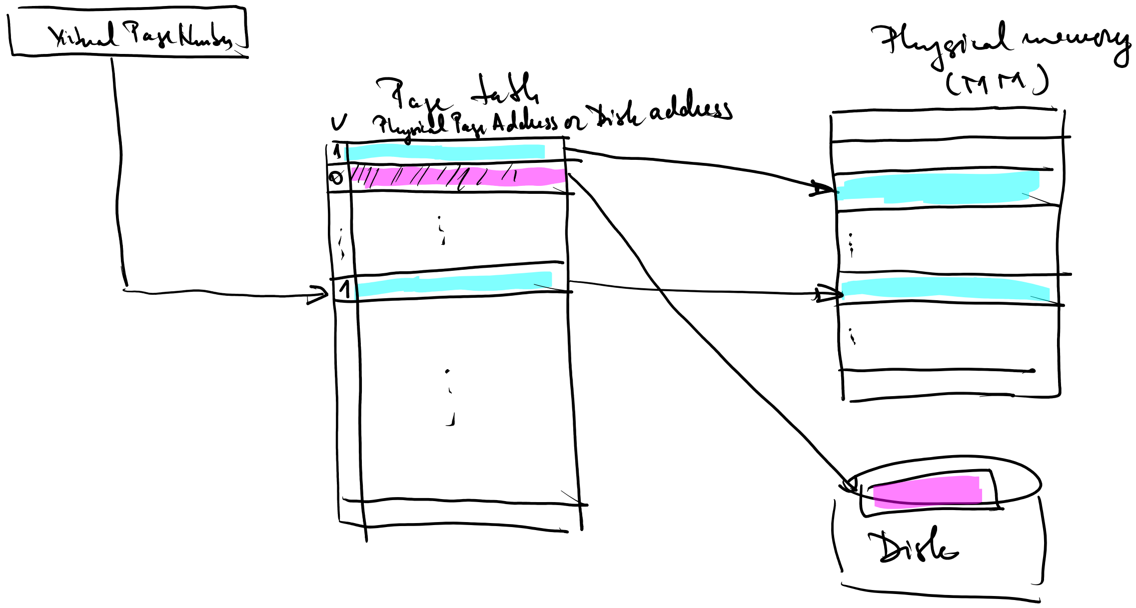
Solutions
- pages large enough to amortize the high access time
  4 KiB — 64 KiB
  – desktops/servers : 32 KiB, 64 KiB
  – embedded system : 1 KiB
- Mappings/organizations that reduce page fault rate are attractive
  => Full Associative mapping of pages in the MM
- Page faults handled in SW (more sophisticated algorithms for placement)
- Write through is banned! Only Write Back + Write Allocate

Taxonomy ARM page → granule

page fault → MMU exception

5.2 Placing a page and finding it again

- Tags are excluded
- Page tables! (table of indexes into MM)

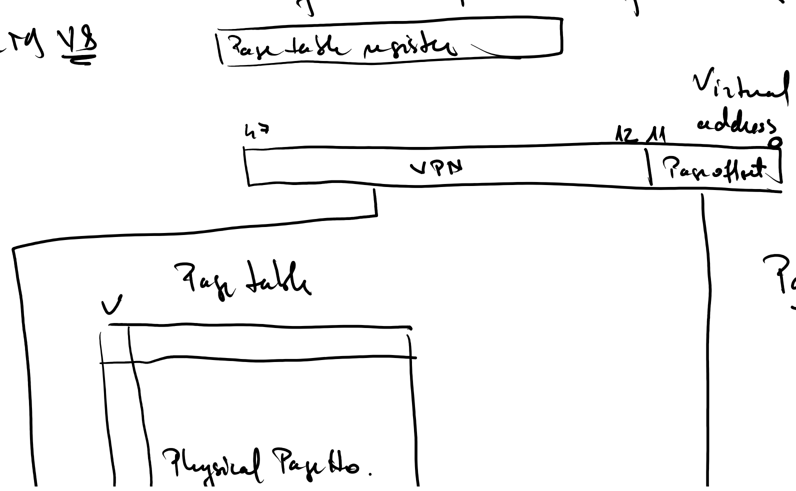


- Page table resides in MM
- Each program has its own Page Table
- HW provides a page table register

$$\text{Page Table} + \underbrace{\text{PC}}_{\text{Program Counter}} + \text{internal registers} = \text{state of process (virtual machine)}$$

- Process is active if is in possession of CPU
  inactive if not in possession of CPU (the state of inactive process is saved)

ARM V8

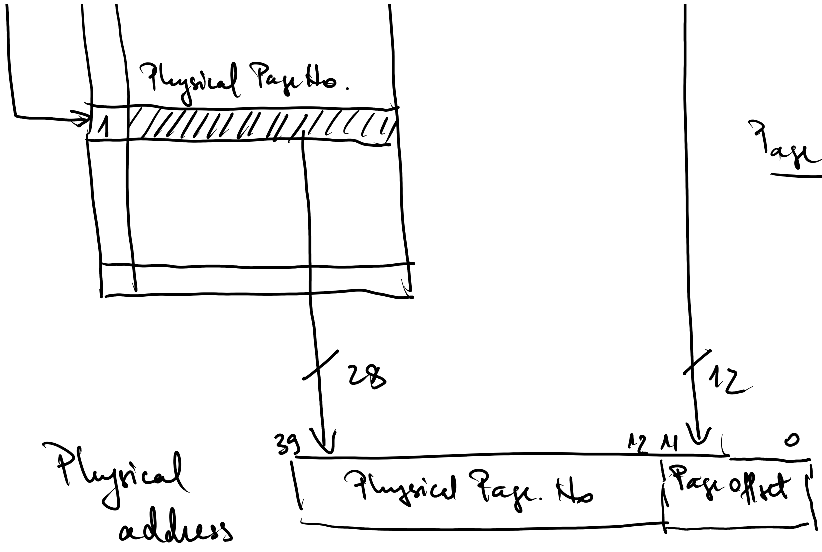


Page fault

→ exception

→ swap space on secondary storage

→ swap space on secondary storage



Page replacement

- approximate LRU
- reference, use, access bits