Sistemas Operativos: VM Paging
Page Tables

Pedro F. Souto (pfs@fe.up.pt)

May 6, 2020

Memory Virtualization

Idea

Goal Illusion that each process has its own memory (address space)

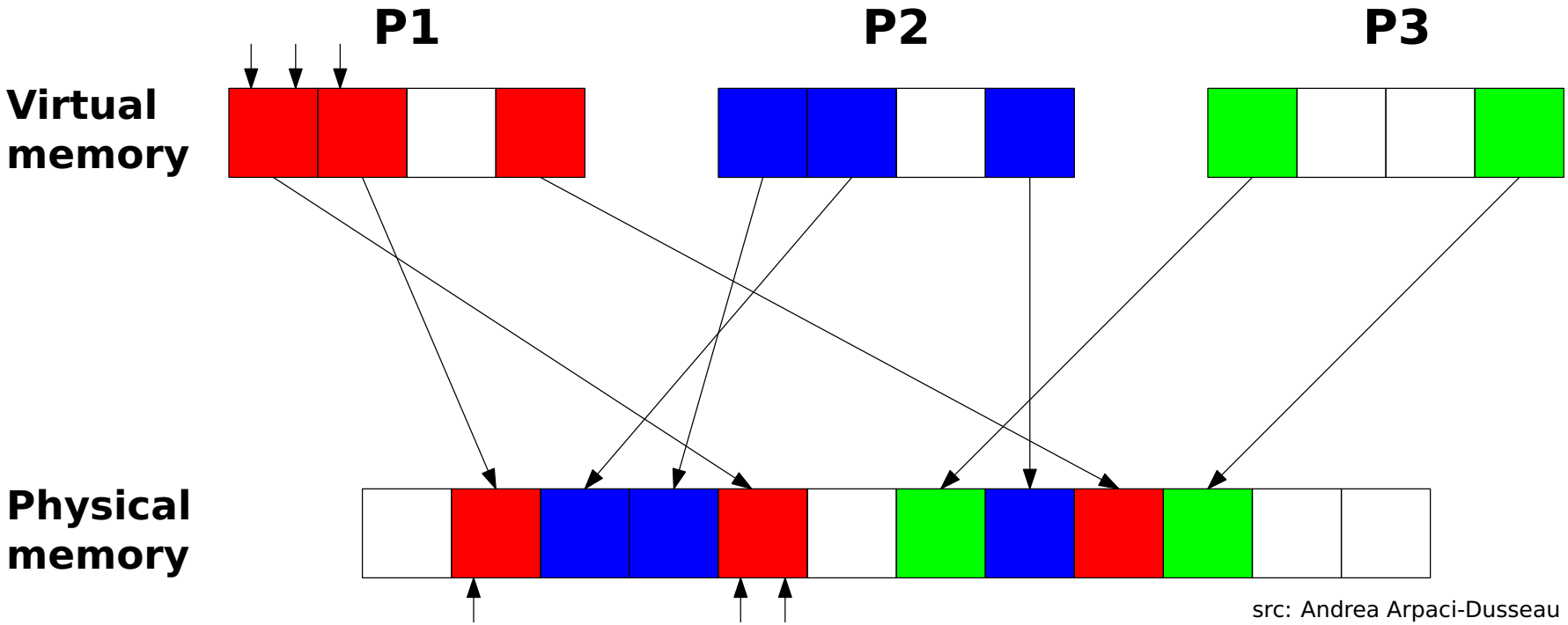
Mechanism Address translation

- On every memory access, the VM subsystem maps the virtual address to a physical address

Paging

Idea

1. Divide address space into *fixed-size* (2^n) units: *pages*
   - Typically 4KiB, but also 8KiB or even 1 MiB (super-pages)
2. Divide physical memory in same size units: *physical pages* or *page frames*
3. Map virtual pages to page frames
   - Relocate each page independently in memory.



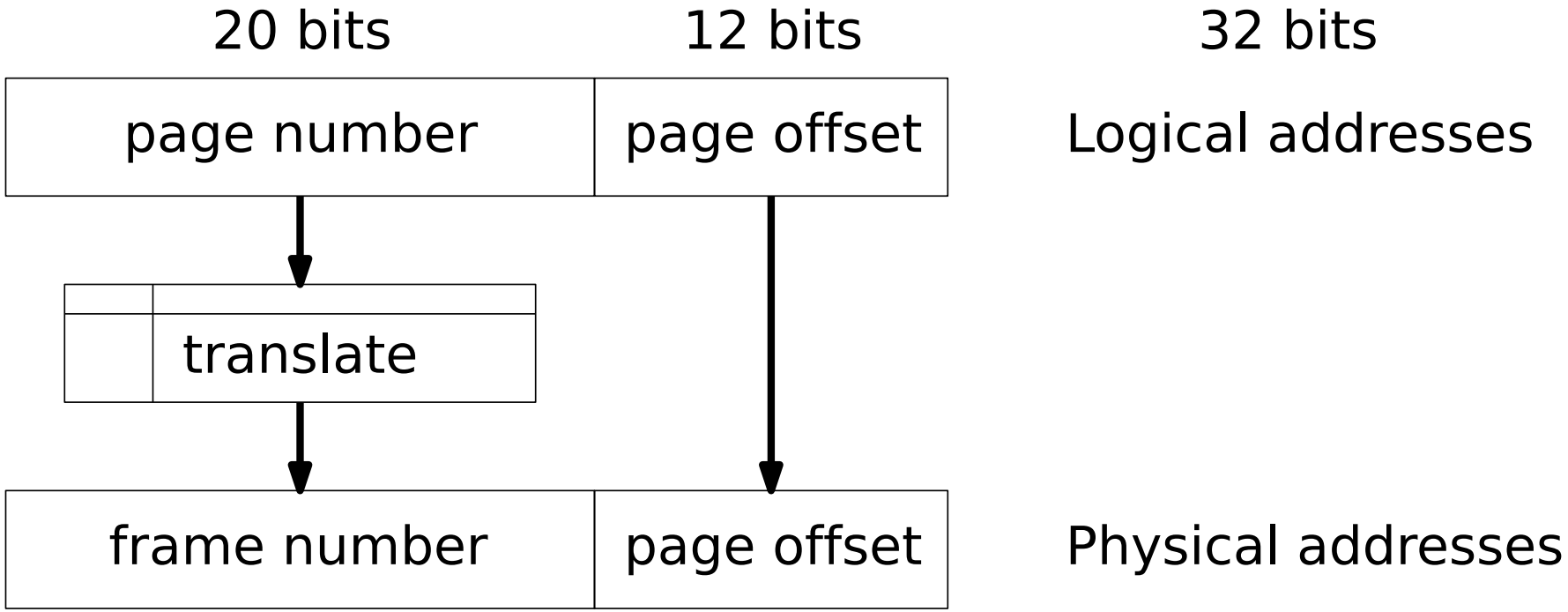
src: Andrea Arpaci-Dusseau

Paging: Address Translation

Question How to translate virtual address to physical address?

Answer Map the virtual page number (VPN) to the page frame number (PFN)

- Virtual pages and page frames have the same size



No addition needed; just append bits correctly

src: Andrea Arpaci-Dusseau

Question What data structure should we use for translating VPN to PPN?

Page Tables: How much space?

Assume

- a **32-bit** address space
- a 4KiB page
- a 4 byte page table entry (PTE)

Answer

- Page table size = Num entries * size of PTE
- Num entries = num virtual pages = $2^{\text{bits for VPN}}$
- Bits for VPN = 32 - number of bits for page offset
  $= 32 - \log_2(4\text{KiB}) = 32 - 12 = 20$
- Num entries = 2^{20} = 1Mi
- Page table size = Num entries * 4 bytes = 4 MiB

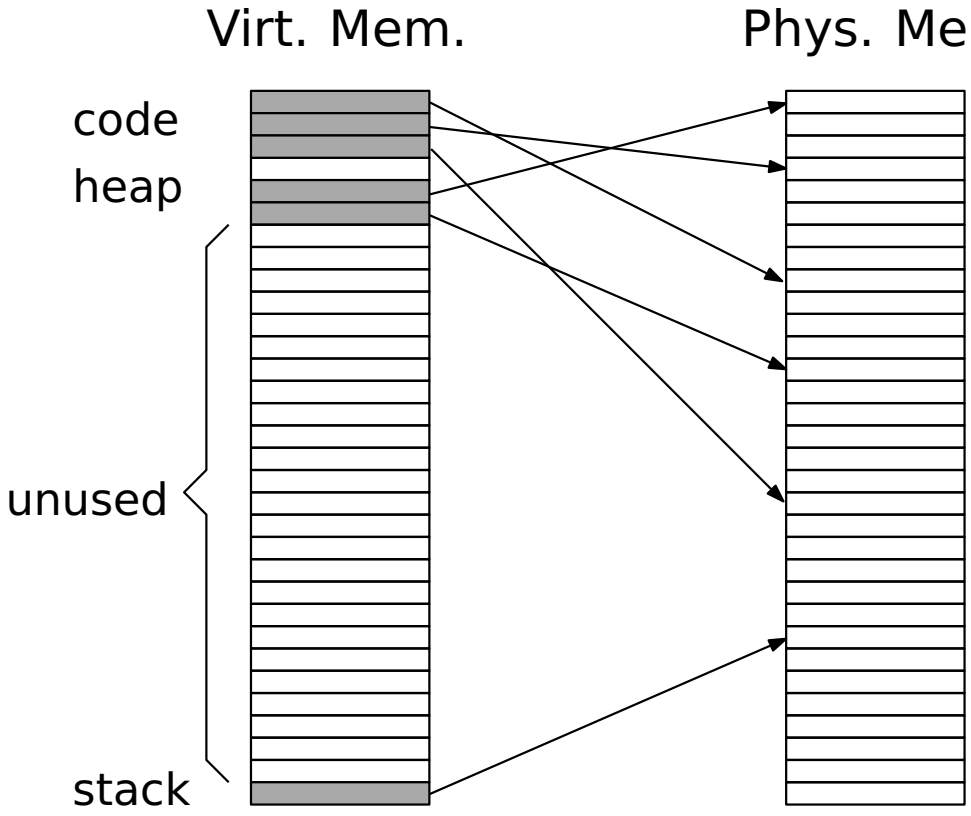
Each process has its own PT

Assume about 200 processes (`ps -ax | wc -l`)

$200 \times 4\ \text{MiB} = 800\ \text{MiB}$!!!

Why are page tables so large?

Answer Page tables are full of invalid PTEs



Page Table

PFN	valid	prot
8	1	r-x
3	1	r-x
18	1	r-x
-	0	-
0	1	rw-
12	1	rw-
-	0	-
-	0	-
many more invalid PTEs		
-	0	-
-	0	-
24	1	rw

Question How to avoid storing invalid PTE?

How to reduce PTs size?

Multi-level Page Tables (PT) i.e. page the PTs

Hybrid segmentation and paging i.e. page the segments

Inverted PTs store one PTE per page frame

- Storewise, cannot do better than this
- Typically, structured as a hash table
- Requires OS-managed TLB

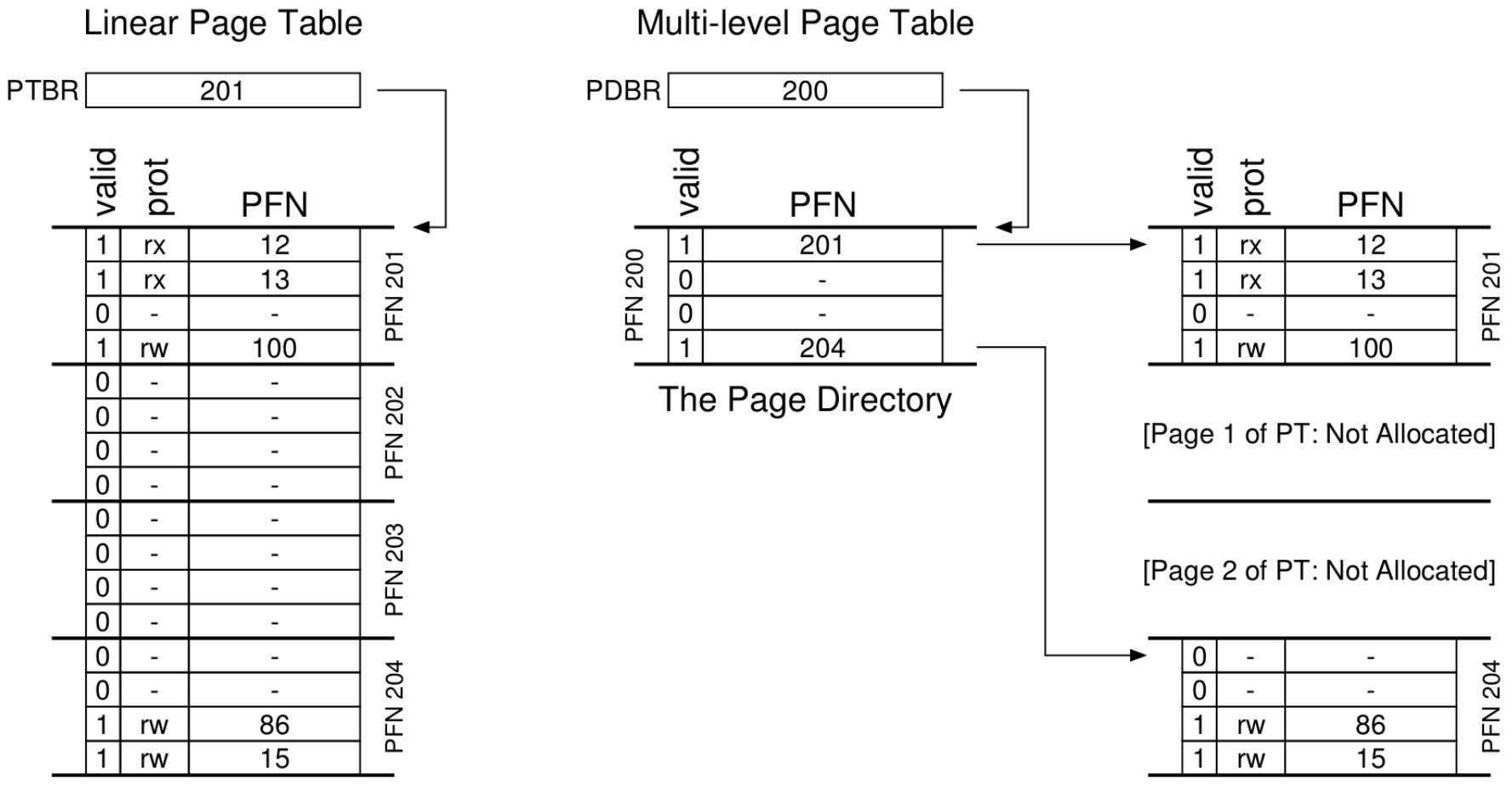
Swap PTs move PTs from main memory to disk and vice-versa

Multilevel Page Tables (PT)

Idea Store the PT in non-contiguous pages, i.e. page the PTs

- Leads to multiple levels of PTs

  Page directory the outer level PT
- Only allocate PT pages with valid PTEs



Address Translation with Multi-level Page Tables (PT)

PD index (4 bits)	PT index (4 bits)	page offset (12 bits)

page directory

PPN	valid
0x03	1
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
0x92	1

page of PT (@PPN 0x03)

PPN	valid
0x10	1
0x23	1
-	0
-	0
0x80	1
0x59	1
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

page of PT (@PPN 0x92)

PPN	valid
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
0x55	1
0x45	1

0x01ABC → 0x23ABC

0x04000 → 0x80000

0XFEED0 → 0x55ED0

Two-level Paging Address Format

30-bit virtual address

page directory index	page table index	page offset (12 bits)

Question: How to structure the virtual address?

- How many bits to use for each paging level?

Remember Multilevel paging is about fitting the PT into pages

Inner level should use as few pages as needed:

sizeof(PTE) $\times$ no_of_PTE_per_page = sizeof(page)

assume sizeof(PTE) = 4 B (bytes)

sizeof(page) = 2^{12} = 4096 B

maximum number of PTEs per page: $4096/4 = 1024$

# of bits for selecting inner page: $log_2(1024) = 10$

Outer level Use remaining bits

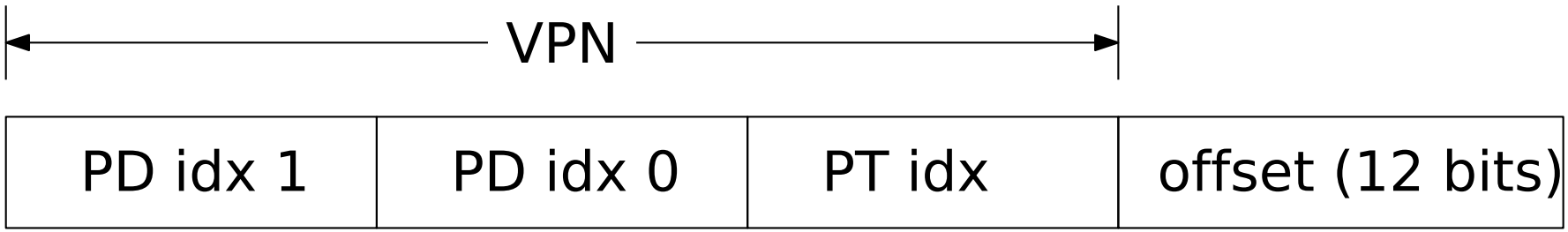
$$30 - 10 - 12 = 8$$

Two levels may not be enough

Problem: Page directory (outer level) may not fit into one page

64-bit virtual address

Solution Page the intermediate levels of the PT



Question How large is virtual address space with 4 KiB pages, 4 B PTE, each page table fits in one page:

4KiB/4 bytes = 1 Ki entries per page

1 level PT : $1Ki \times 4Ki = 2^{22}B = 4MiB$

2 level PT : $1Ki \times 1Ki \times 4Ki = 2^{32}B = 4GiB$

3 level PT : $1Ki \times 1Ki \times 1Ki \times 4Ki = 2^{42}B = 4TiB$

Multilevel PTs and TLBs

Problem: on a TLB miss, each PT level adds one memory access

Assume:

3-level PT

256-bytes pages 8-bit offset

16-bit addresses 8-bit VPN

TLB

ASID	VPN	PFN	Valid
211	0xBB	0x91	1
211	0xFF	0x23	1
122	0x05	0x91	1
211	0x05	0x12	0

How many memory accesses by process with ASID 211?

```
0xAA10:  movl 0x1111, %edi
0xBB13:  addl $0x3, %edi
0x0519:  movl %edi, 0xFF10
```