Sistemas Operativos: VM Paging

TLB

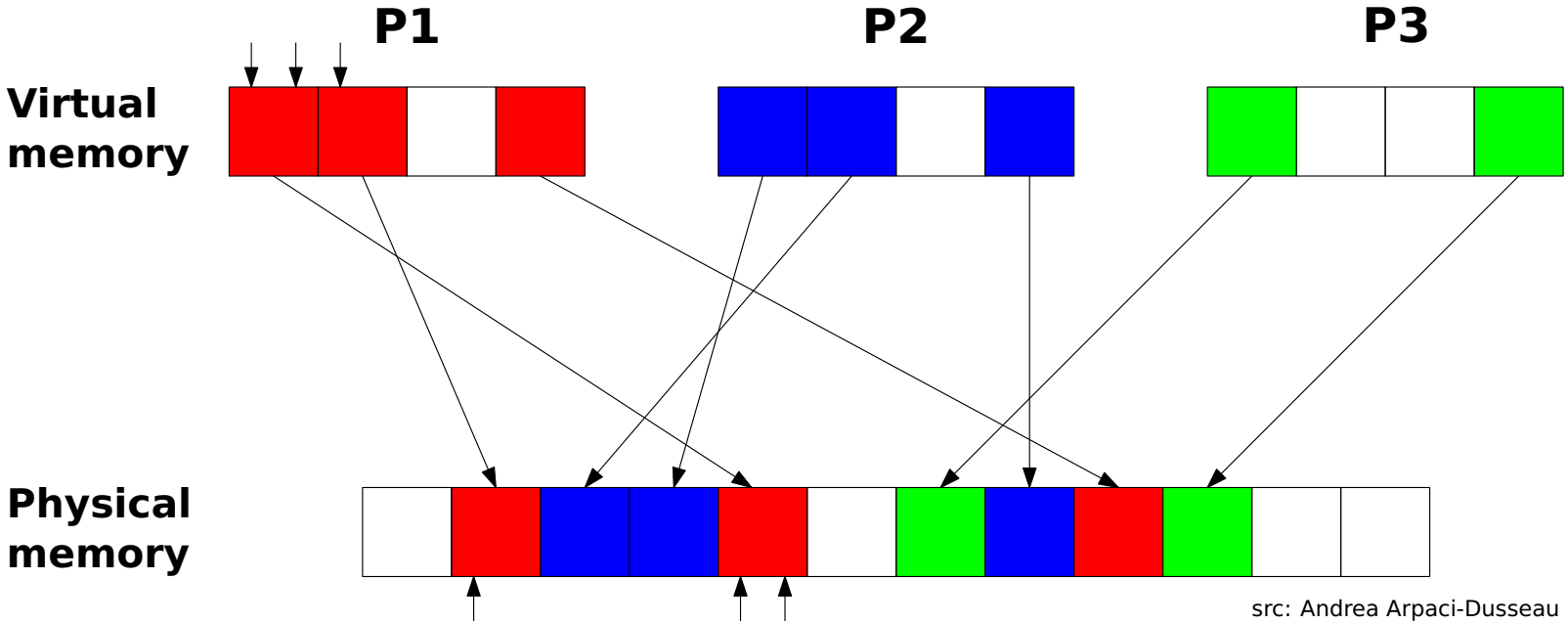
Pedro F. Souto (pfs@fe.up.pt)

May 6, 2020

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

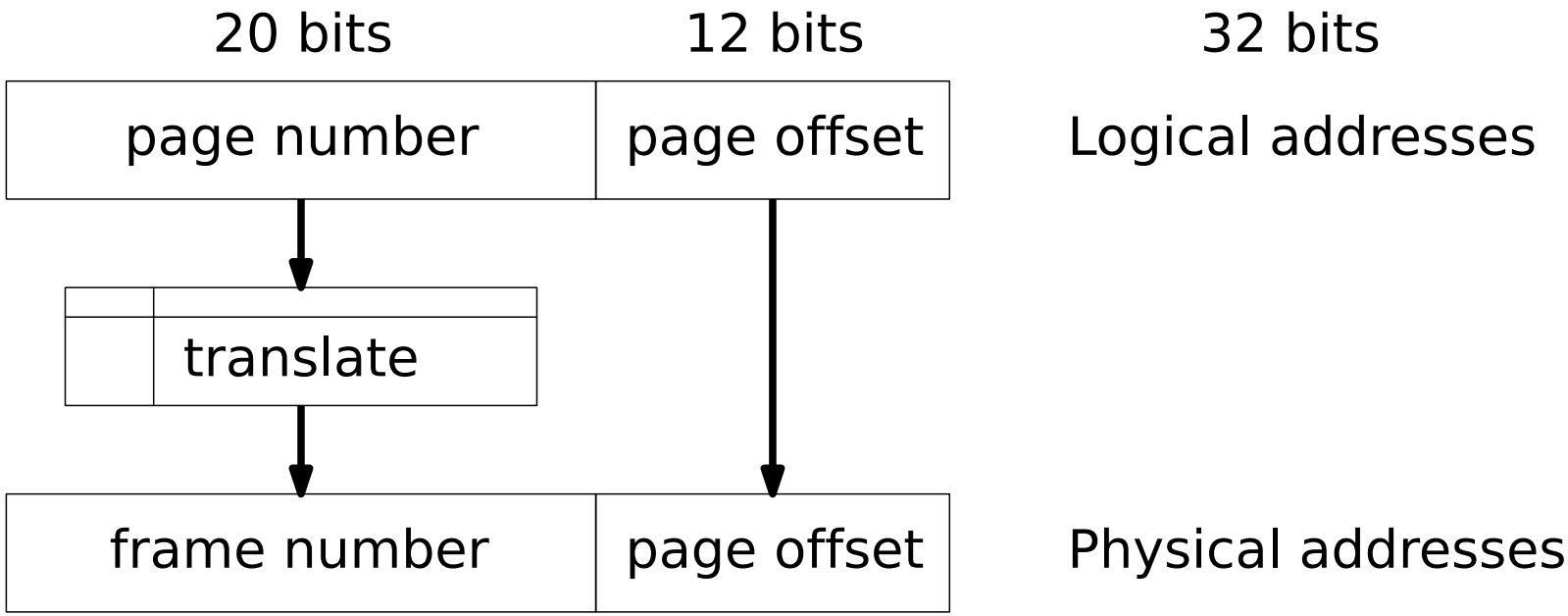
Question How to translate virtual address to physical address?

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

- Virtual pages and page frames have the same size

  High order bits of VA specify the page number

  Low order bits of VA specify the offset within page



No addition needed; just append bits correctly

src: Andrea Arpaci-Dusseau

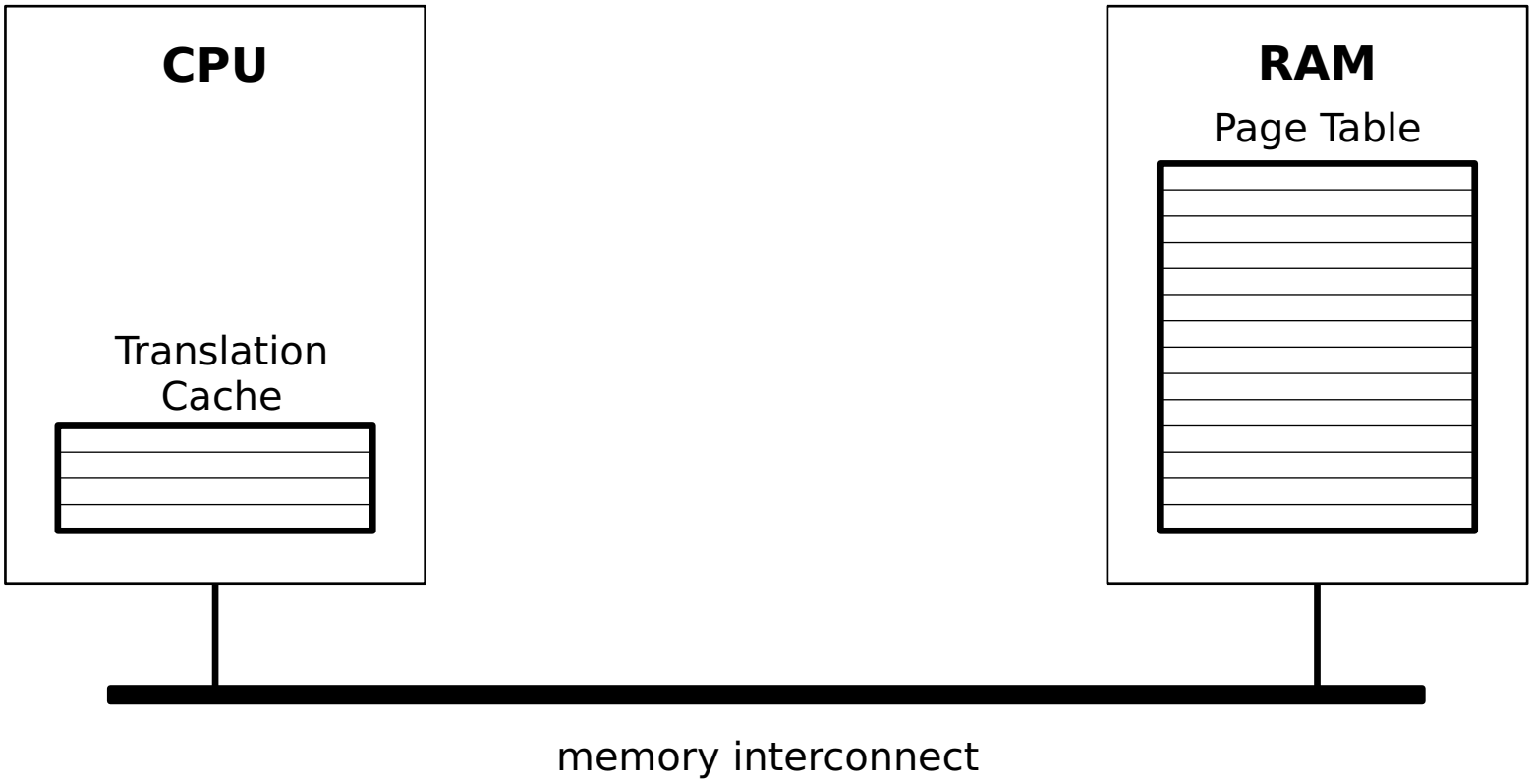
Virtual Addresss to Physical Address Translation: HW Implementation

1. Extract VPN from virtual address
   VPN = VA >> PAGE_SHIFT
2. Compute address of PTE
   PTBR + VPN
3. Read PTE from PT (in memory)
4. Extract PFN (if PTE valid)
5. Extract offset from virtual address
   OFFSET = VA & OFFSET_MASK
6. Concatenate offset to PFN
   PA = PFN | OFFSET

Issue Address translation requires one memory access (to the PTE) – step 3

How to avoid it?

Solution: Cache Previous Translations



src: Andrea Arpaci-Dusseau

Translation Lookaside Buffer cache of previous translations

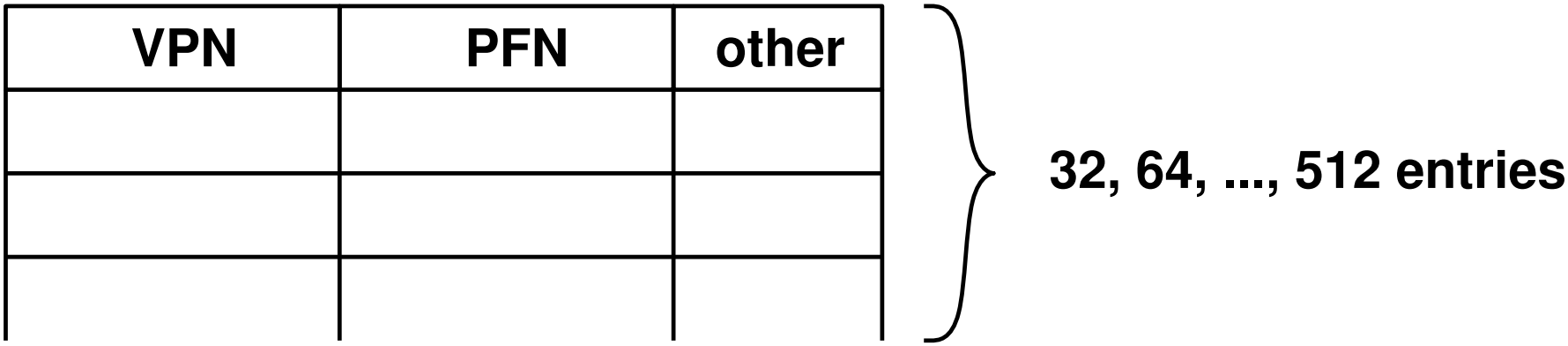
- Poor name for the address translation cache

Rationale memory access locality

Spatial future accesses will be to nearby addresses

Temporal in the near future the same address will be used

TLB



Other

Supervisor can be accessed only in privileged/supervisor mode
Valid is the entry valid
Protection for quick protection check

- This is similar to main memory cache
  - It is content addressable:
    - via the VPN field
  - It is fast
    - Register-like speed
    - Parallel search
  - Usually fully-associative
    - Translation for every VPN can be anywhere

Virtual Addresss Translation: HW Implementation

1. Extract VPN from virtual address
   VPN = VA >> PAGE_SHIFT
2. Check TLB

   If TLB hit
   - 2.1 Extract PFN from TLB entry

   Else (TLB miss)
   - 2.1 Compute address of PTE: PTBR + VPN
   - 2.2 Read PTE from PT (in memory)
   - 2.3 Extract PFN (if PTE valid)
3. Extract offset from virtual address
   OFFSET = VA & OFFSET_MASK
4. Concatenate offset to PFN
   PA = PFN | OFFSET

Requirement Most programs exhibit memory access locality so that on average address translation costs close to 0 memory accesses

Who handles TLB misses?

HW CPU must know where the page-tables are

- CR3 register on x86
- Page-table structure is determined by the HW
- HW "walks" the page-table and fills TLB

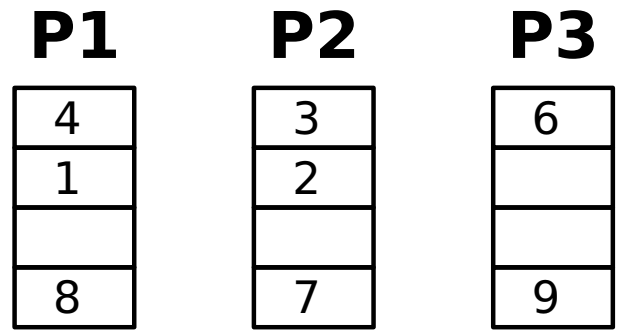
OS CPU traps into OS upon TLB miss

- "Software-managed TLB"
- OS may use a more convenient page-table structure
- OS updates TLB (with privileged instruction)
  - Which replacement policy to use?
- Upon return from trap, the HW retries the instruction

TLB & Context Switch

Issue Each process has its own address space

- Same VPN of different processes are most likely mapped to different PFNs
- The TLB contents stops to be valid upon context switch



Solution at least two

Flush the TLB, i.e. invalidate all entries

Track which entries are for which process

- Use an Address Space Identifier (ASID), e.g. 8 bits
- Tag each TLB entry with the ASID

ASID	VPN	PFN	other

- ASID and VPN are both used to access the TLB