2-Level Page Tables

Virtual Address (VA): 32 bits

Offset or Displacement field in VA: 12 bits

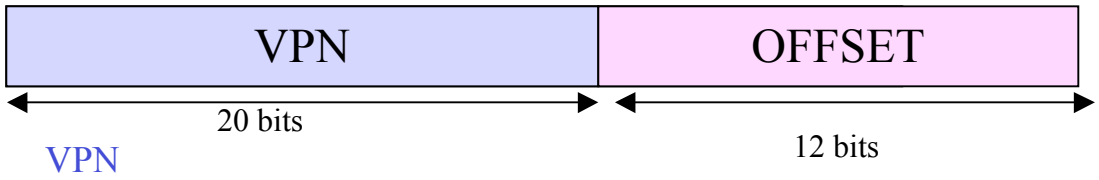
Virtual Address Space: 2^{32} bytes

Page Size: 2^{12} bytes = 4KB

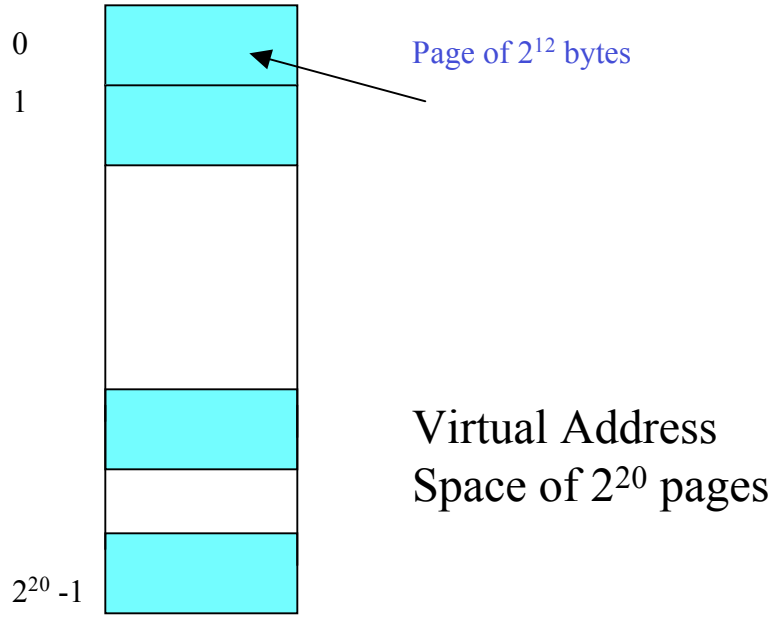
Virtual Page Number field in VA: 32 - 12 = 20 bits

Number of Virtual Pages: $2^{32} / 2^{12} = 2^{20}$

VA:



VPN



2-Level Page Tables

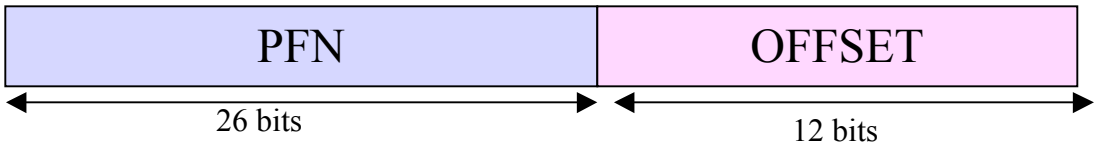
PA:

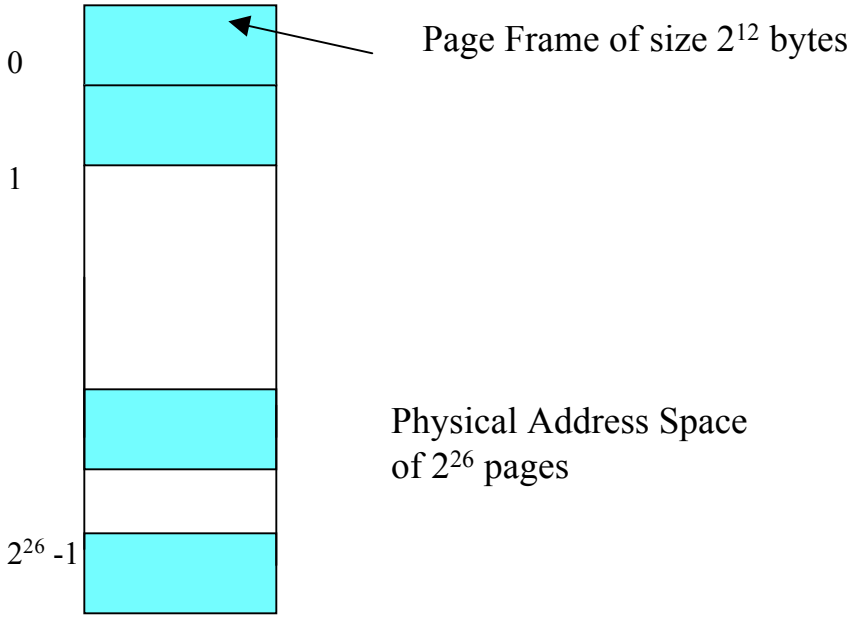
Physical Address (PA): 38 bits
Offset or Displacement field in PA: 12 bits

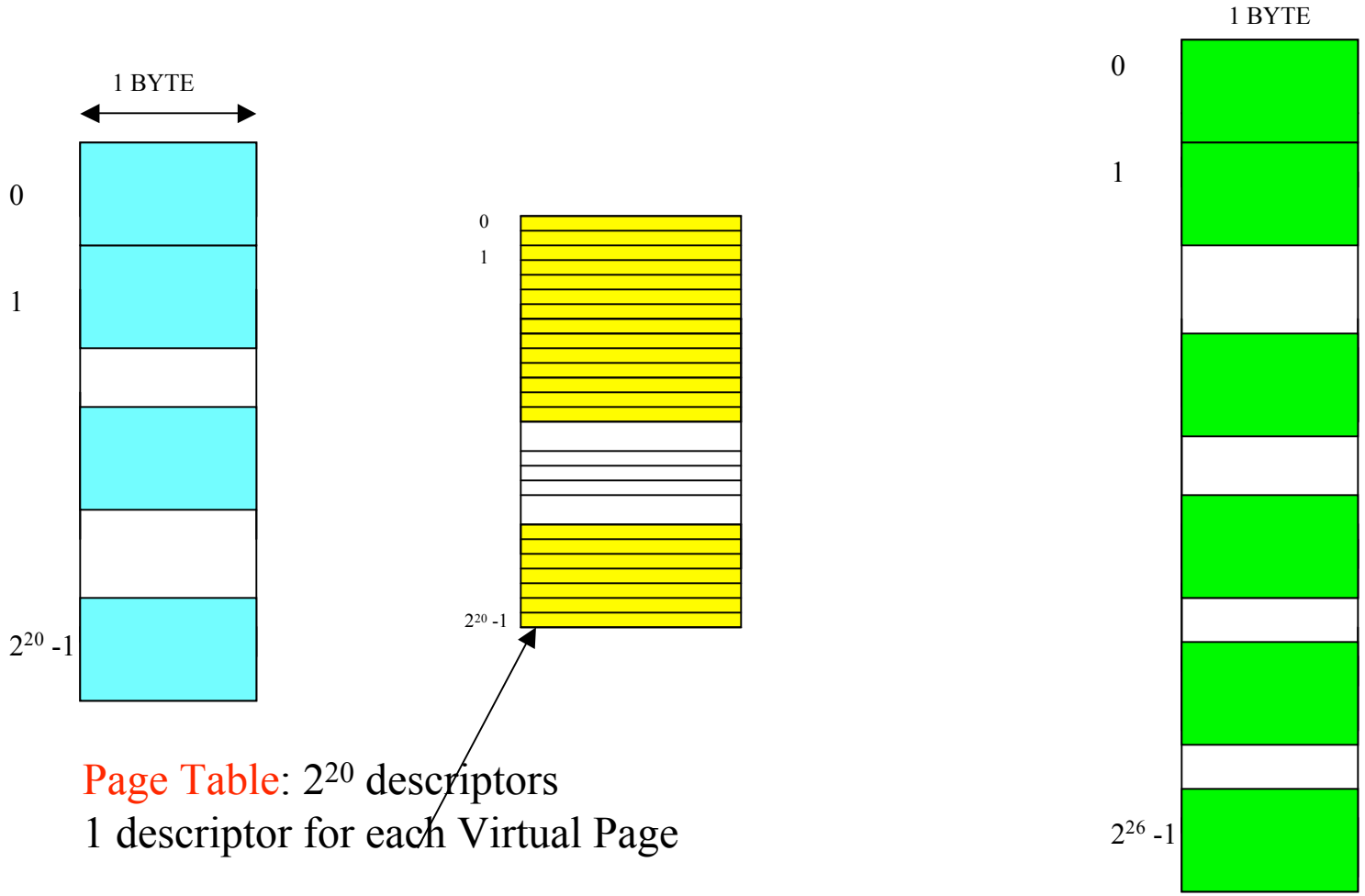
Physical Address Space: 2^{38} bytes
Page Size: 2^{12} bytes = 4KB

Page Frame Number field in PA: 38 - 12 = 26 bits

Number of Physical Pages: $2^{38} / 2^{12} = 2^{26}$







Page Table: 2^{20} descriptors
1 descriptor for each Virtual Page

VPN can be used as an index into Page Table to find the descriptor for that page

Single Level Page Table

- Descriptor holds the Page Frame Number (PFN) of the virtual page if it is in memory

- A presence bit (P) indicates if it is in memory or on the backing device
- Descriptor also contains other administrative and protection bits
  - e.g. D (Dirty), U (Used), R (Read), W (Write), E(Execute) etc.

In the example: PFN requires 26 bits
Assume exactly the 6 administrative bits mentioned above

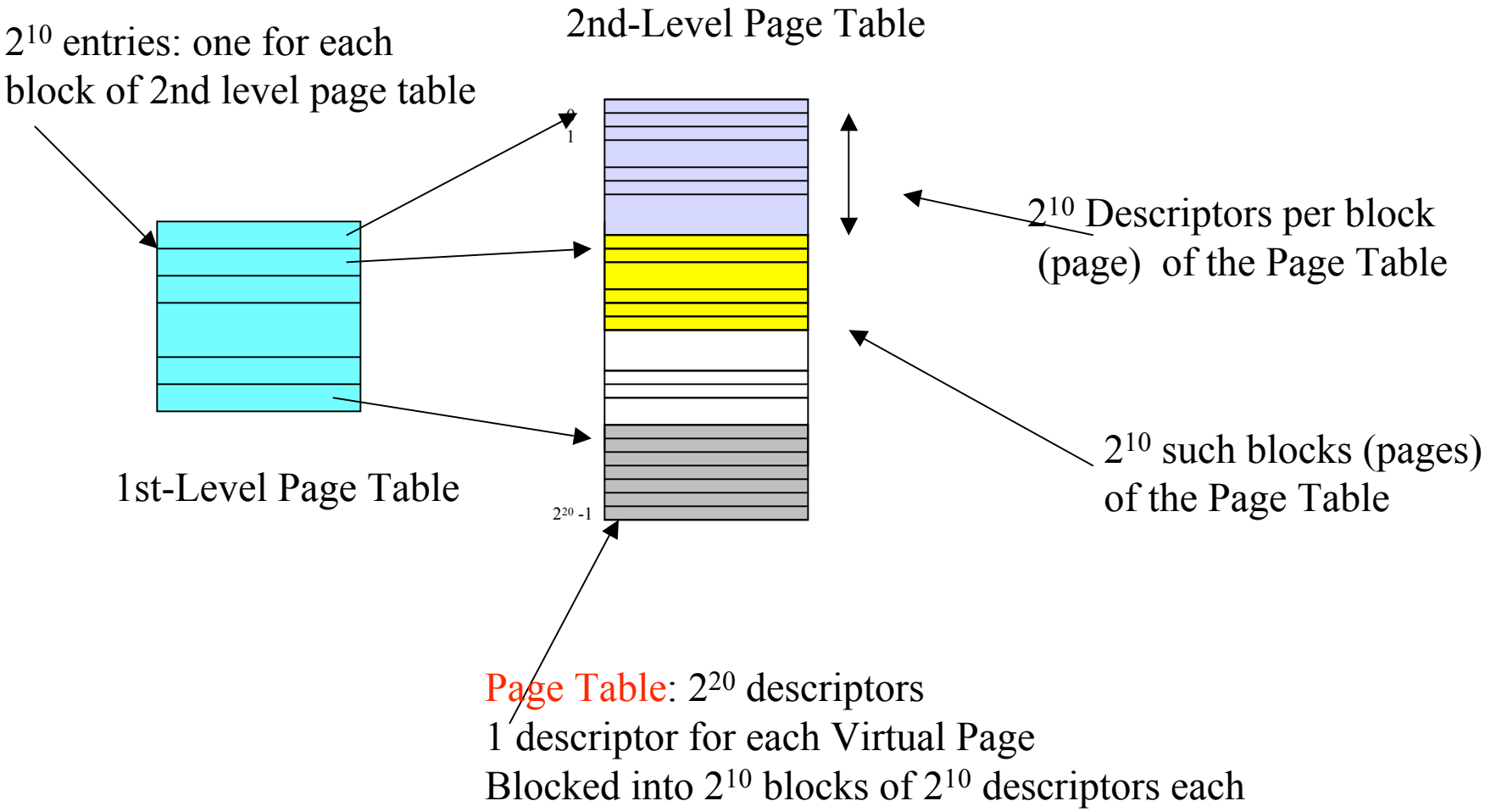
Descriptor is 26+6 = 32 bits or 4 bytes wide.

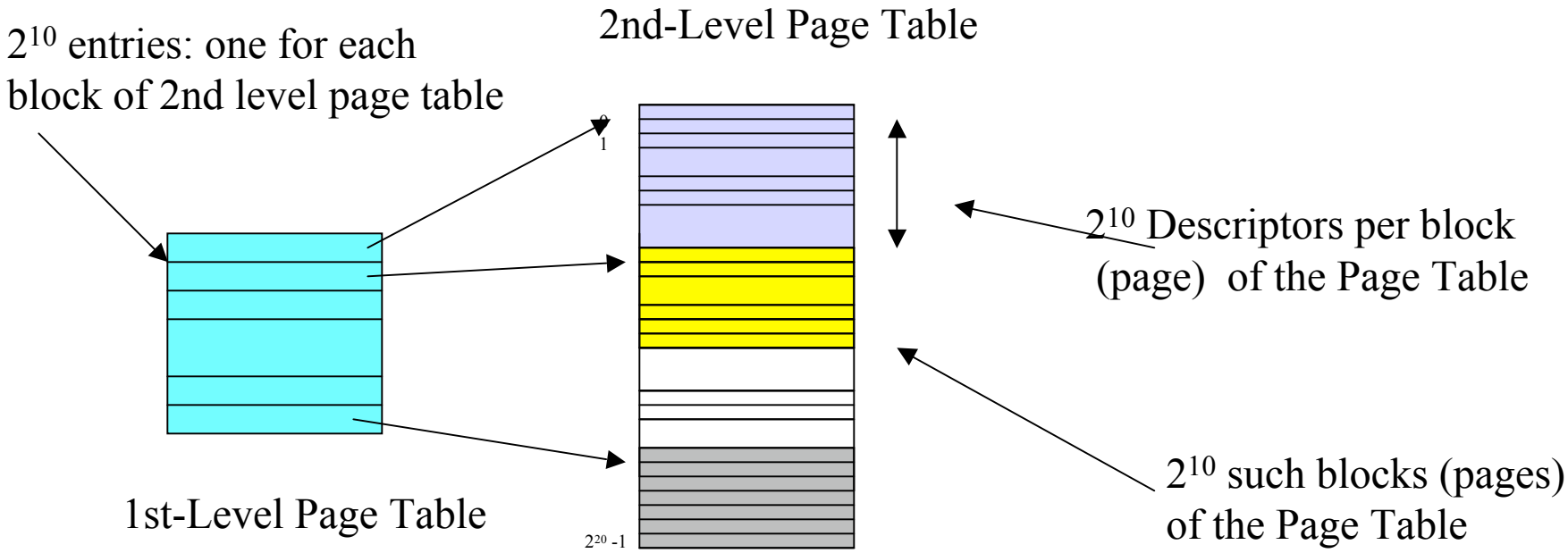
Size of Page Table = Number of descriptors x Size of descriptor
= 2^{20} x 4 bytes = 4MB

Two-Level Page Tables

- Break up Page Table into fixed-size blocks of the same size as a page
- In example: Each page is 4KB and Page Table is 4MB
- So we will have 4MB/4KB = 2^{10} = 1024 such blocks
  - This collection of blocks that make up the Page Table will be called the 2nd-level Page Table
  - The 1st-Level Page Table will have entries pointing to each block of the 2nd level Page Table.
  - In example: 1024 entries in the 1st-level Page Table
- How many descriptors in each block?
  - Each block (or page of the Page Table) will hold:

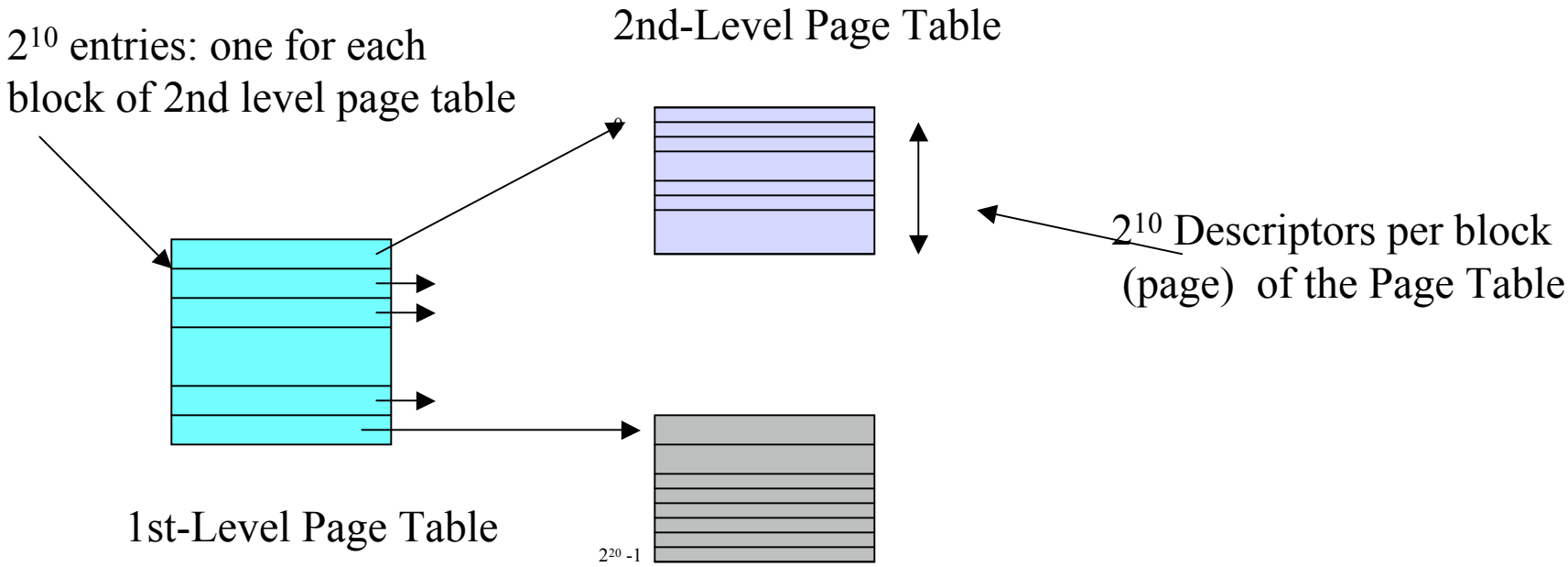
    4KB/4bytes = 1024 descriptors


2nd-Level Page Table
2^{10} entries: one for each block of 2nd level page table
0
1
2^{10} Descriptors per block (page) of the Page Table
2^{10} such blocks (pages) of the Page Table
1st-Level Page Table
2^{20} -1
Page Table: 2^{20} descriptors
1 descriptor for each Virtual Page
Blocked into 2^{10} blocks of 2^{10} descriptors each



Do not need to store the entire 2nd level Page Table as a contiguous array

Do not allocate blocks that have no descriptors
Keep blocks on secondary store and bring in when needed
(mini virtual memory system for the Page Table management)



Do not need to store the entire 2nd level Page Table as a contiguous array

Do not allocate blocks that have no descriptors
Keep blocks on secondary store and bring in when needed
(mini virtual memory system for the Page Table management)

Q: What is the actual size of virtual address space being used by the above process?

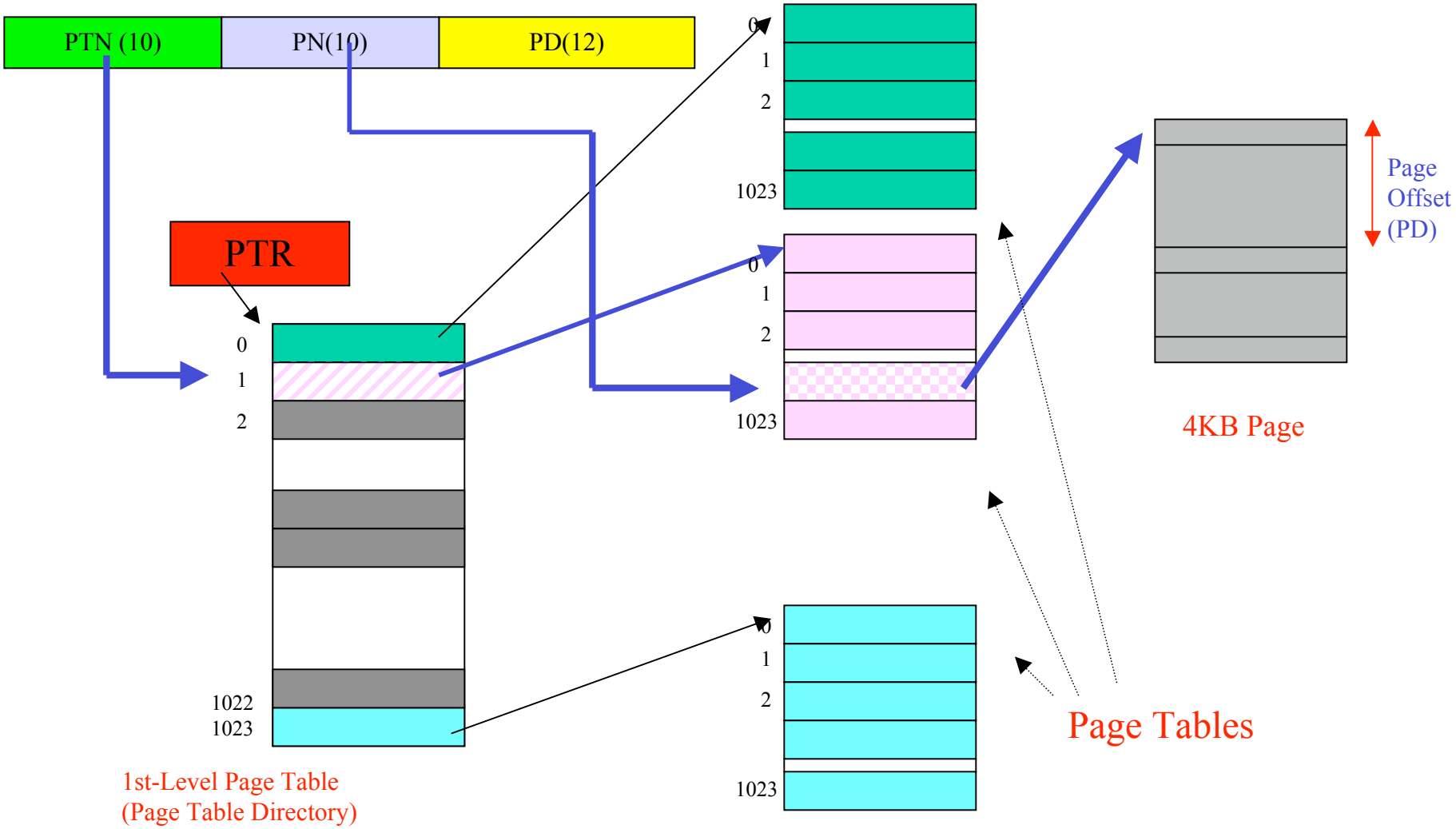
Each descriptor represents to a 2^{12} = 4KB portion of the address space

2 blocks = 2 x 1024 descriptors imply : 2 x 1024 x 4KB = 8MB address space

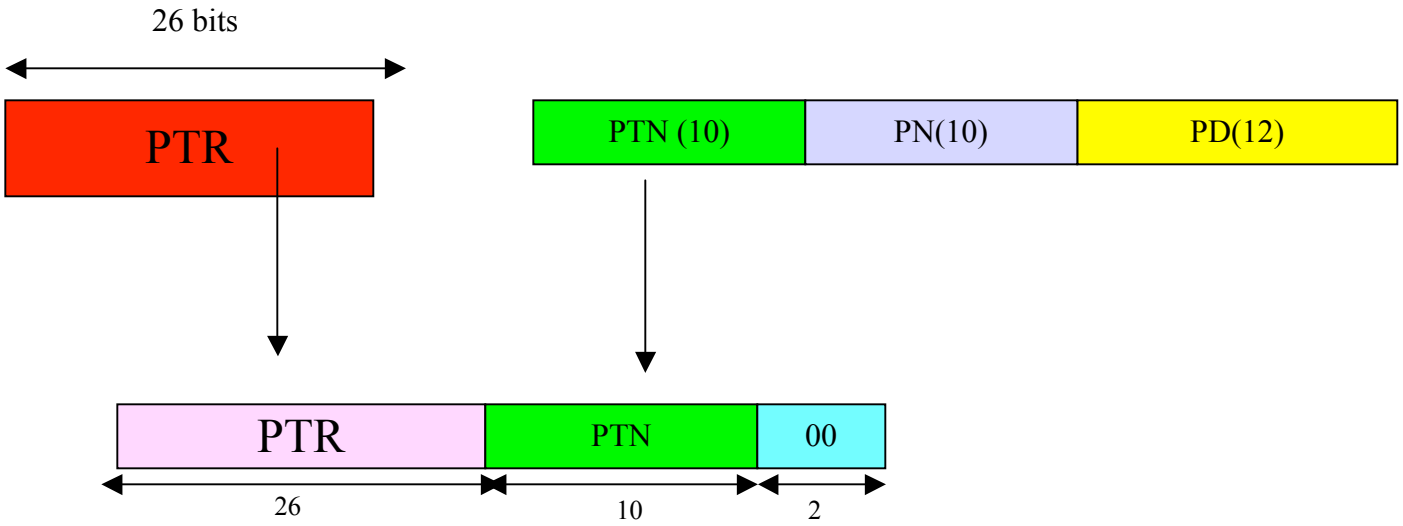
Virtual Memory: 2-level Page Table

10 MSBs (bits 22..31) of the virtual address (PTN) are used to index into the Page Table Directory

Next 10 bits (12 ..21) are used to index the chosen Page Table.



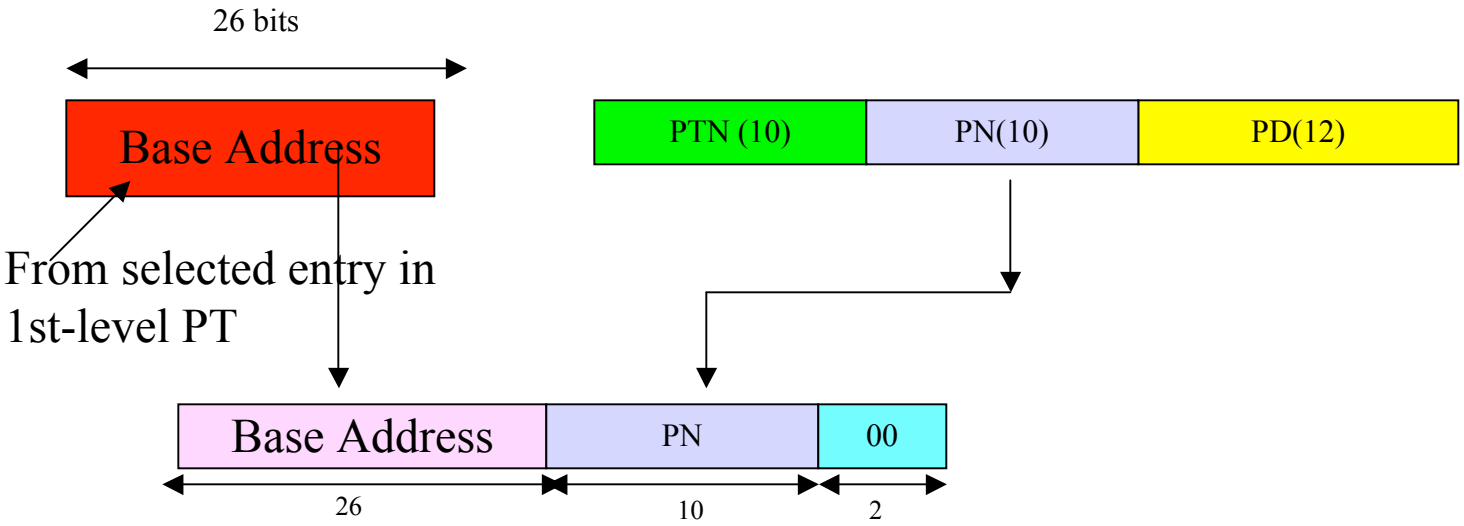
More Details on Page Table Lookup



38 bit physical address of desired1st-level Page Table entry

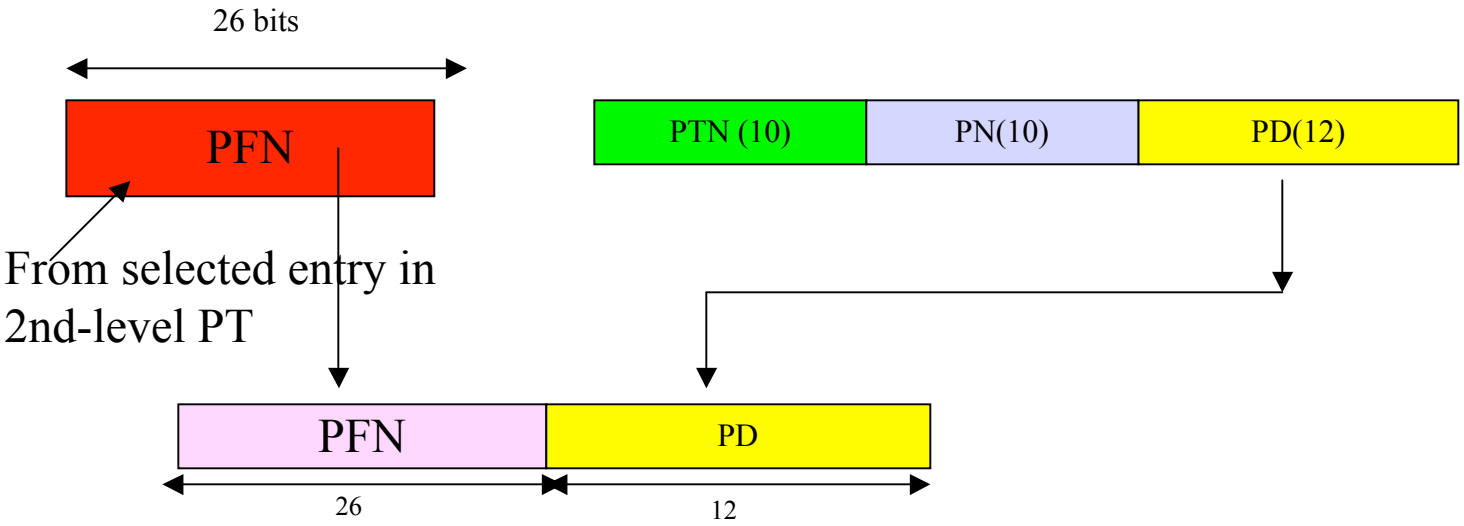
1st Level Page Table and all blocks of the 2nd-level Page Table are stored at Page Aligned Boundaries i.e. 12 LSBs are zero

More Details on Page Table Lookup



38 bit physical address of desired 2nd-level Page Table entry

More Details on Page Table Lookup



38 bit physical address of desired memory byte

Virtual Memory and Caches

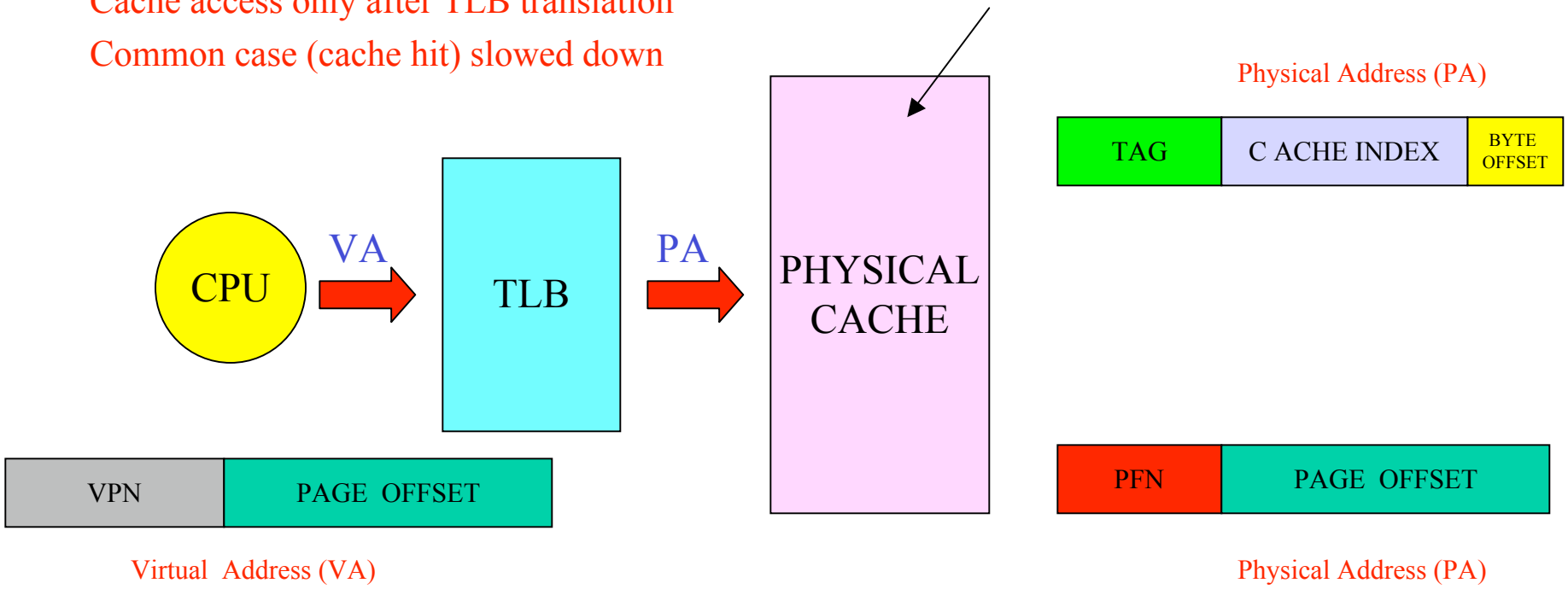
Physical cache

Accessed using translated physical address

Cache access only after TLB translation

Common case (cache hit) slowed down

Cache locations addressed using physical memory addresses

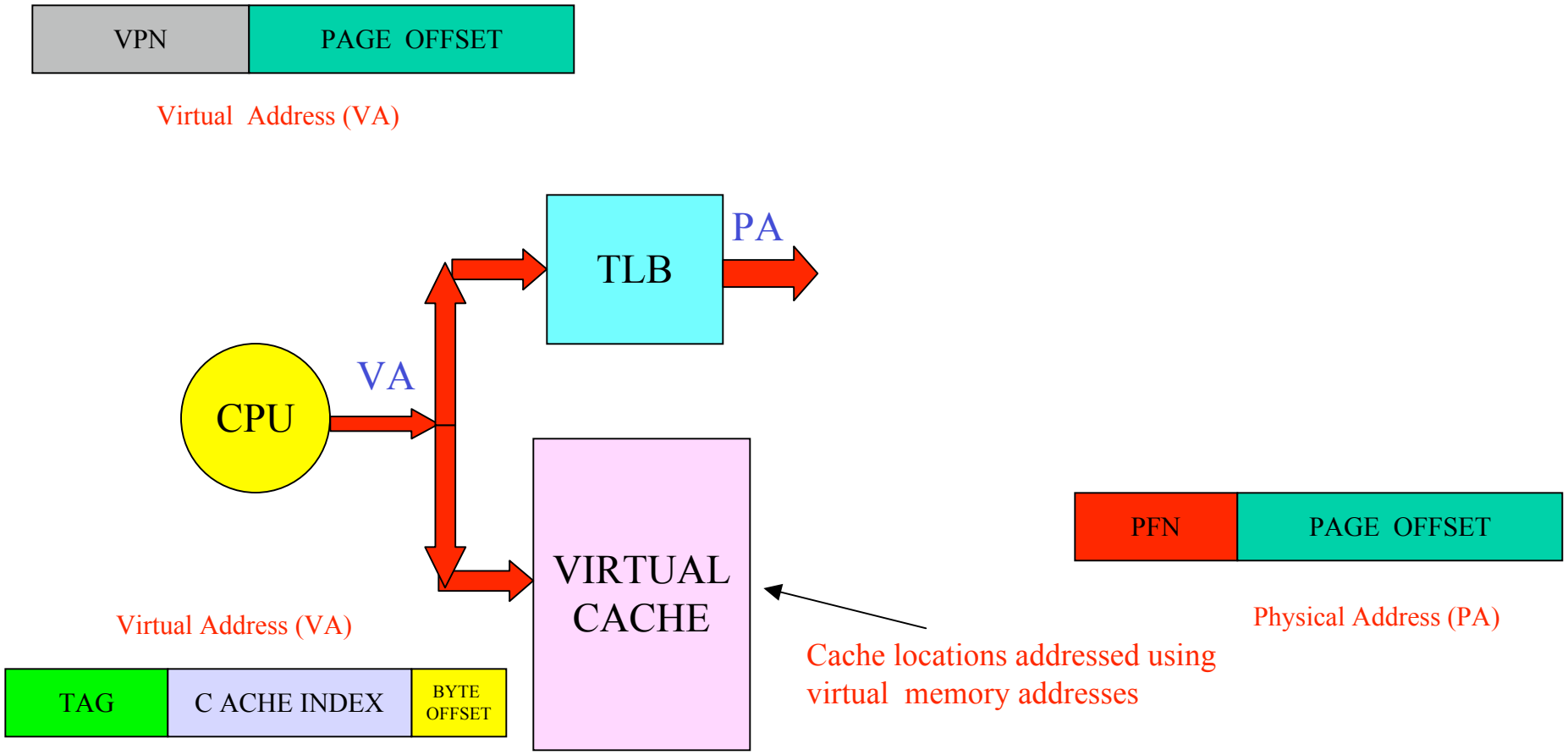


Can we avoid latency of translation every memory access?

Virtual Memory and Caches

Virtual cache

Accessed using the virtual address directly



Virtual Memory and Caches

Virtual Cache

Accessed using virtual addresses

(+) Address translation (TLB lookup) in parallel with cache lookup

Access TLB for protection information unless information replicated in cache

(-) Context switch must invalidate all cache entries

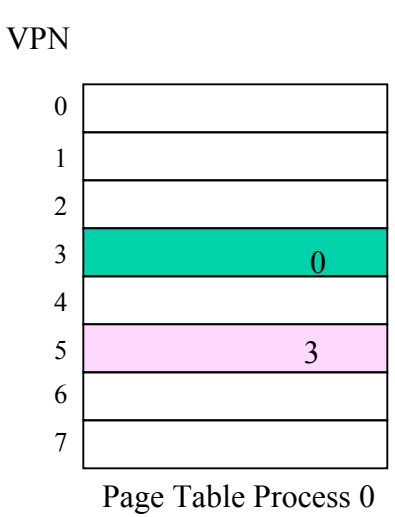
Every process has the same virtual address space 0 … 2^n -1

How do you distinguish a virtual address of some process from the same virtual address of a different process ?

Use processor identifiers (PIDs) as additional field to tag cache blocks

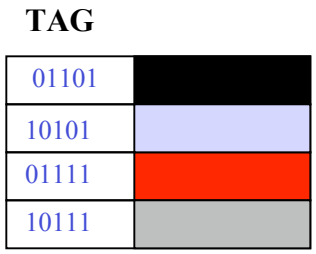
Virtual Memory and Caches

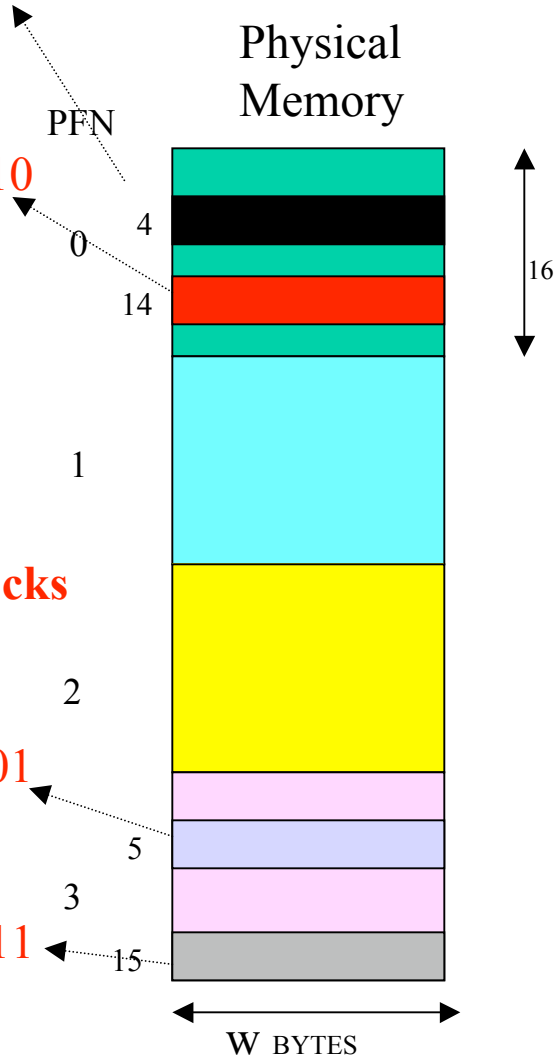
Virtual Cache



VA = 011 0100 PA = 00 0100

VA = 011 1110 PA = 00 1110





Virtual Cache
Direct Mapped size 4 Blocks

Assumes cache block size of w bytes

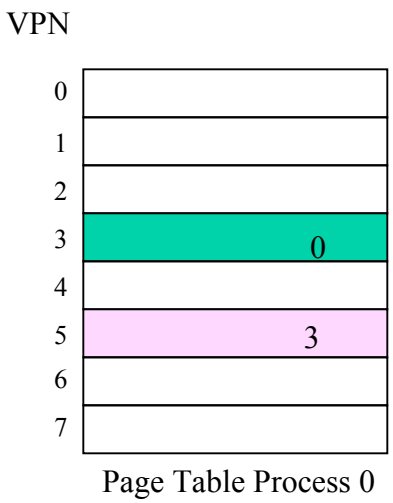
VA = 101 0101 PA = 11 0101

VA = 101 1111 PA = 11 1111

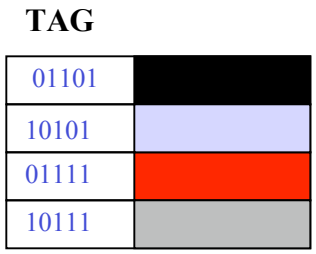
Virtual Memory and Caches

Virtual Cache

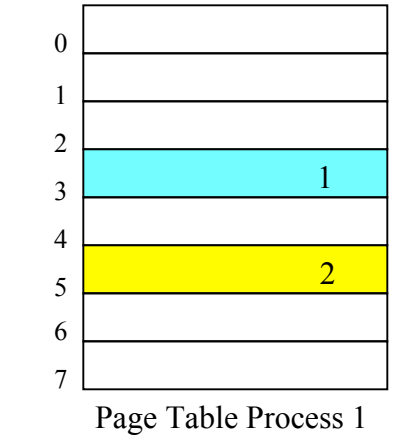
VA = 011 0100



VA = 011 1110

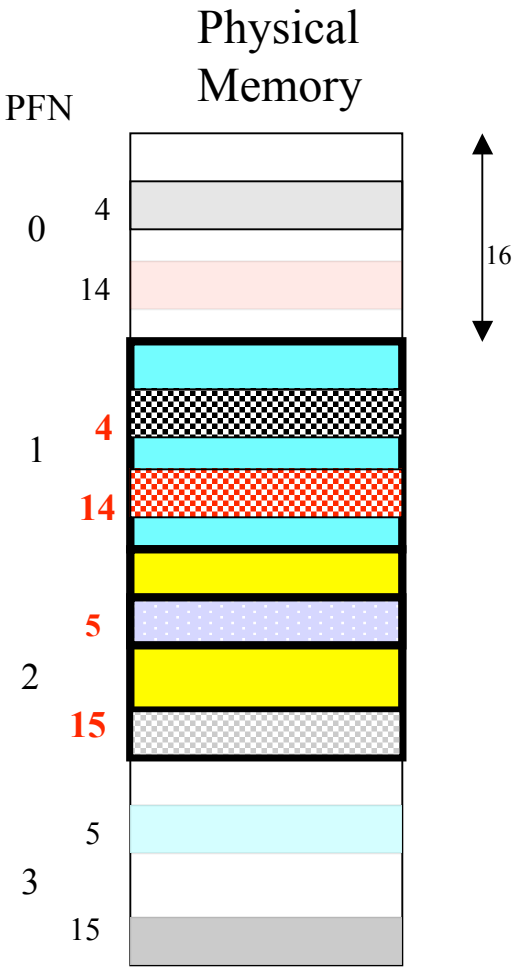


Virtual Cache



VA = 101 0101

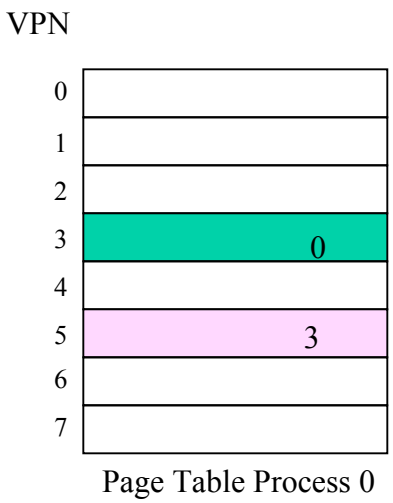
VA = 101 1111

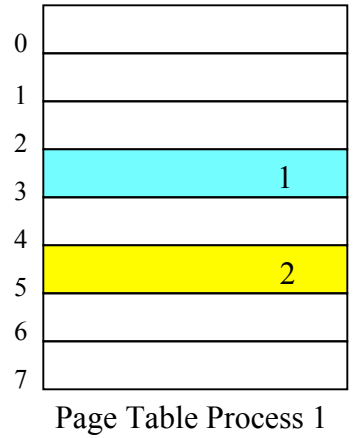


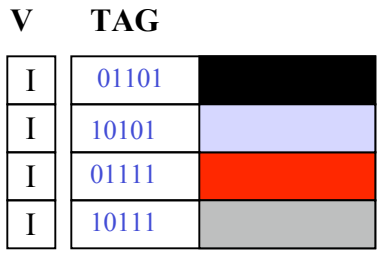
Process 1 will access cached data of Process 0

Virtual Memory and Caches

Virtual Cache





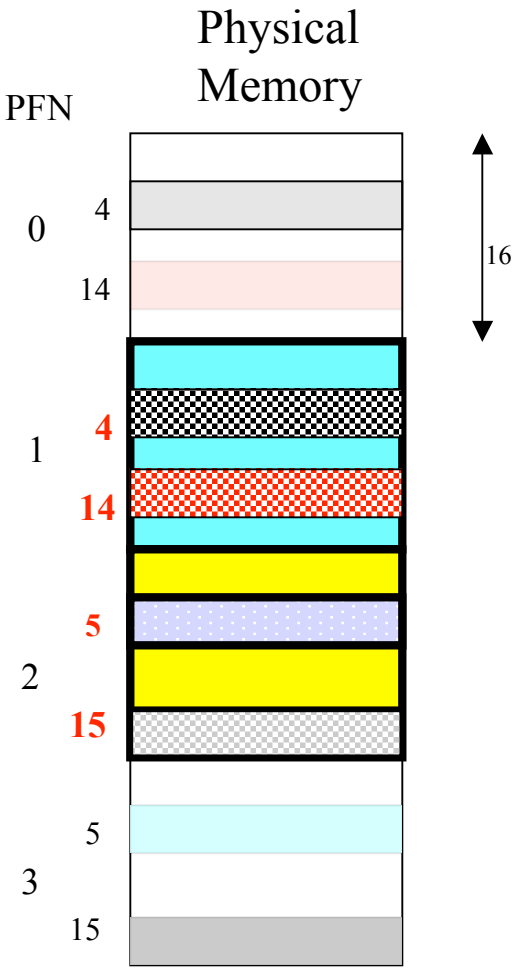


Virtual Cache

Solution 1: Invalidate all cache blocks on a context switch

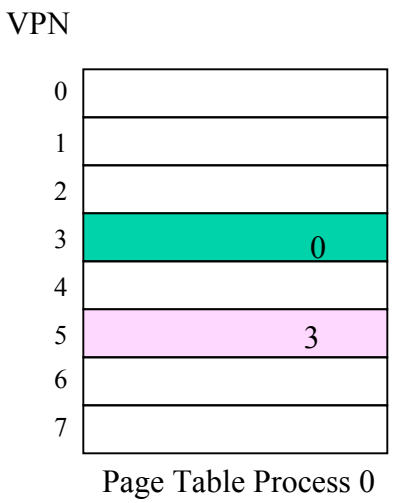
Cache blocks that may have survived (i.e were not evicted) by swapped-in process are wastefully invalidated.

Cold cache on resumption

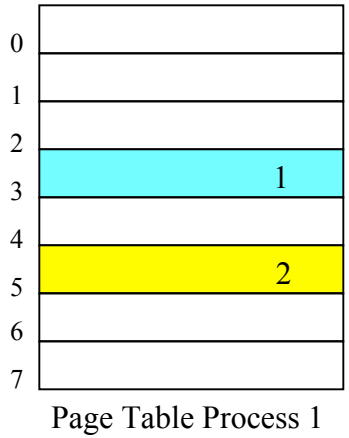


Virtual Memory and Caches

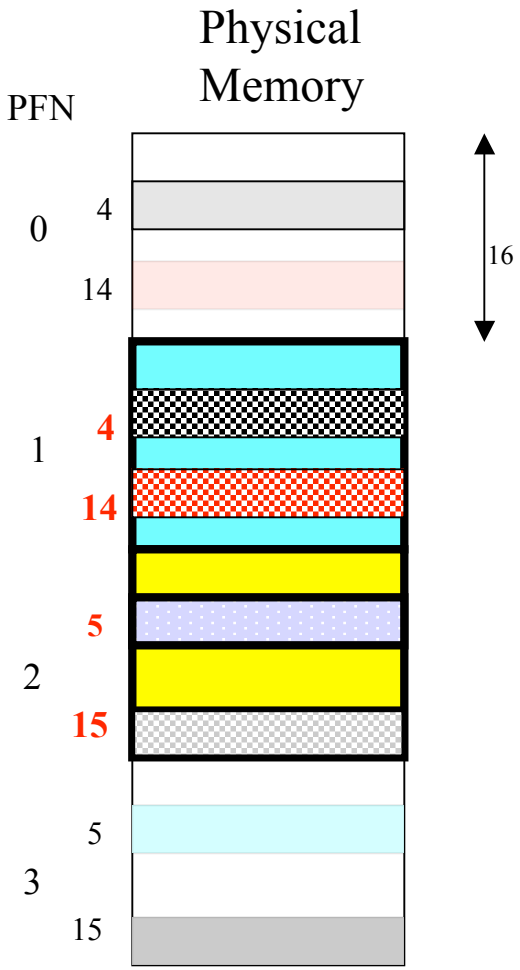
Virtual Cache



Virtual Cache



Solution 2: Add a process id field as a part of the tag to identify the process whose blocks are in cache



Virtual Memory and Caches

Virtual Cache : Accessed using virtual addresses

(-) Aliases: Different names for the same physical object

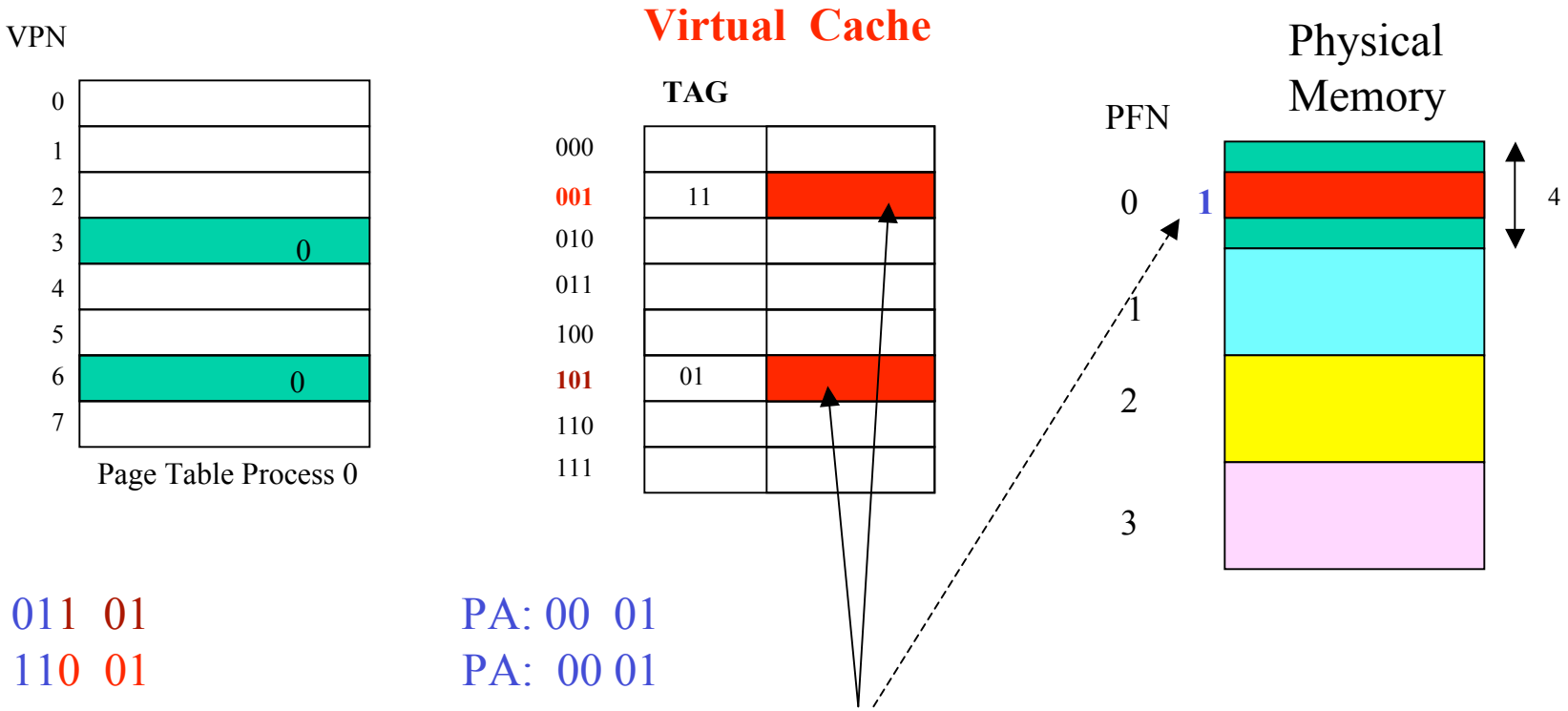
Different virtual address but same physical address

May result in multiple inconsistent copies of the data in the cache

:

Virtual Memory and Caches

Virtual Cache



VA: 011 01
VA: 110 01

PA: 00 01
PA: 00 01

Note: This can only happen if the number of blocks in any way of the cache is greater than the number of blocks in a virtual page.

2 cached copies of same physical location (in different cache locations)

For aliasing to occur: The cache page size must exceed the virtual page size