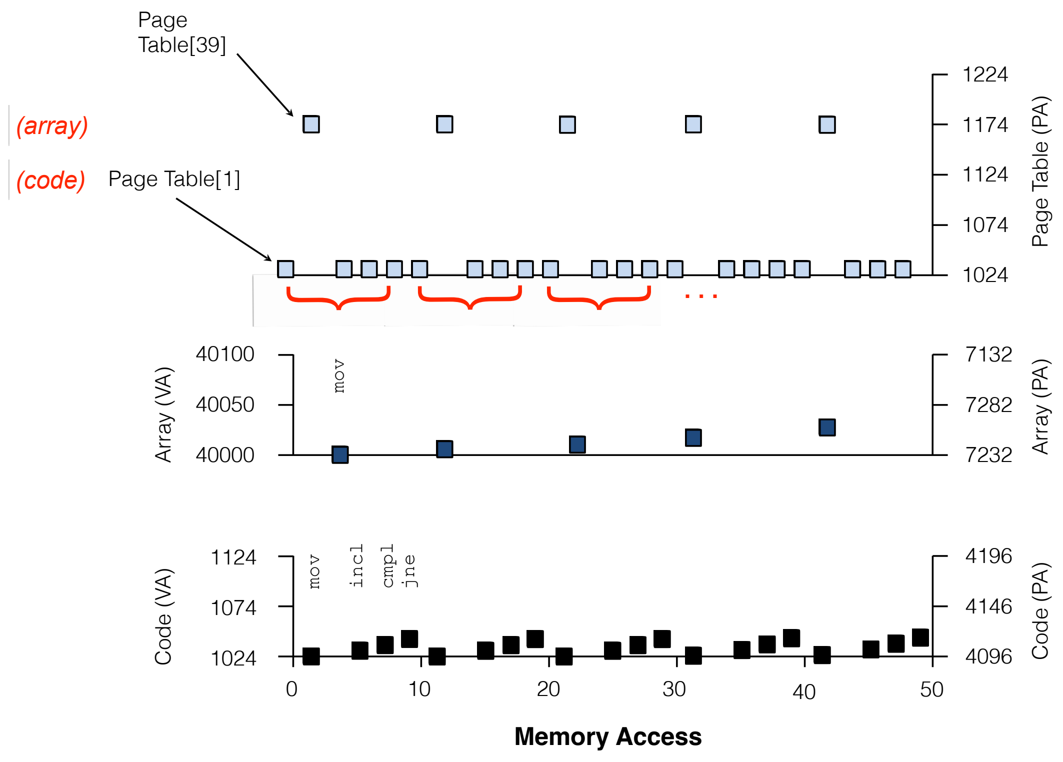
Virtual Memory

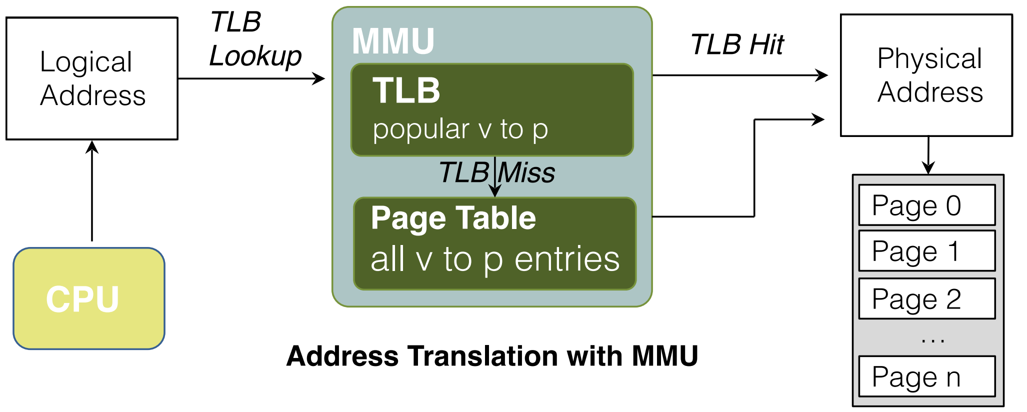
- 13 - Address Spaces
- 14 - Memory API
- 15 - Address Translation
- 16 - Segmentation
- 17 - Free Space Management
- 18 - Paging
- 19 - Translation Lookaside Buffers
- 20 - Advanced Paging
- 21 - Swapping
- 22 - Swapping Policy

A Virtual(And Physical) Memory Trace



TLB

- Part of the chip’s memory-management unit (MMU).
- A hardware cache of **popular** virtual-to-physical address translation.



Address Translation with MMU

Physical Memory

Basic TLB Algorithm

- extract the virtual page number (VPN)
- check for hit in the the TLB
- extract page frame number from relevant TLB entry, form desired physical address, and access memory

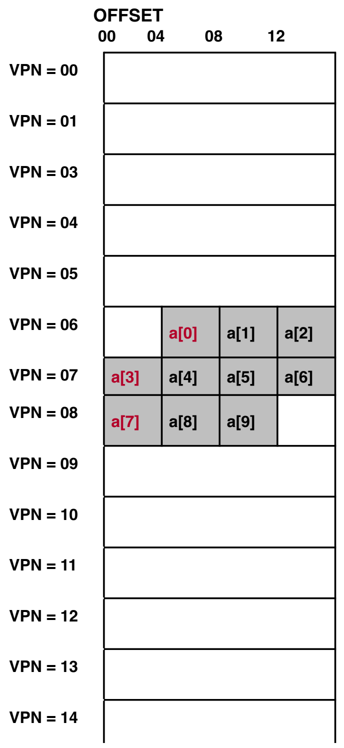
Basic TLB Algorithm

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True)    // TLB Hit
    if (CanAccess(TlbEntry.ProtectBits) == True)
        Offset   = VirtualAddress & OFFSET_MASK
        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
        Register = AccessMemory(PhysAddr)
    else
        RaiseException(PROTECTION_FAULT)
else                    // TLB Miss
    PTEAddr = PTBR + (VPN * sizeof(PTE))
    PTE = AccessMemory(PTEAddr)
    if (PTE.Valid == False)
        RaiseException(SEGMENTATION_FAULT)
    else if (CanAccess(PTE.ProtectBits) == False)
        RaiseException(PROTECTION_FAULT)
    else
        TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
        RetryInstruction()
```

Figure 19.1: **TLB Control Flow Algorithm**

Example: Accessing An Array

- How a TLB can improve its performance (only data accesses shown)



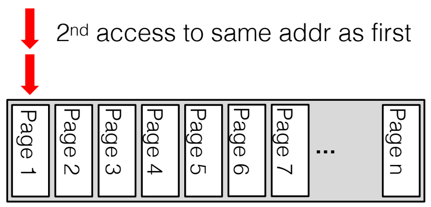
The TLB improves performance due to spatial locality

```
int sum = 0 ;
for( i=0; i<10; i++){
        sum += a[i];
}
```

3 TLB misses and 7 hits.
Thus TLB hit rate is 70%.

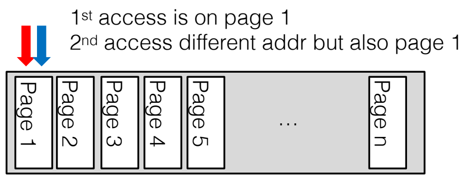
Locality

- Temporal Locality
  - An instruction or data item that has been recently accessed will likely be re-accessed soon in the future.



Virtual Memory

- Spatial Locality
  - If a program accesses memory at address **`x`**, it will likely soon access memory near **`x`.**



Virtual Memory

Who Handles The TLB Miss?

- Hardware handles the TLB miss entirely on CISC processors.
  - The hardware know where the page tables are located
  - … “walks” the page table, finding the correct entry and extracting the desired translation, update and retry instruction.
  - this is a hardware-managed TLB.
- RISC processors often manage TLBs in software.
  - On a TLB miss, the hardware raises an exception
    - Trap handler is code within the OS that is written with the express purpose of handling TLB misses.

TLB Control Flow algorithm (OS Handled)

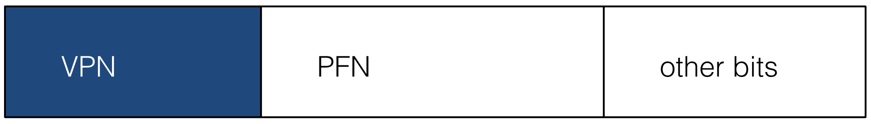
- The hardware would do the following:

```
VPN = (VirtualAddress & VPN_MASK) >> SHIFT
(Success, TlbEntry) = TLB_Lookup(VPN)
if (Success == True) // TLB Hit
    if (CanAccess(TlbEntry.ProtectBits) == True)
        Offset = VirtualAddress & OFFSET_MASK
        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
        Register = AccessMemory(PhysAddr)
    else
        RaiseException(PROTECTION_FAULT)
else // TLB Miss
    RaiseException(TLB_MISS)
```

- But might be slow, why not just use the hardware approach?

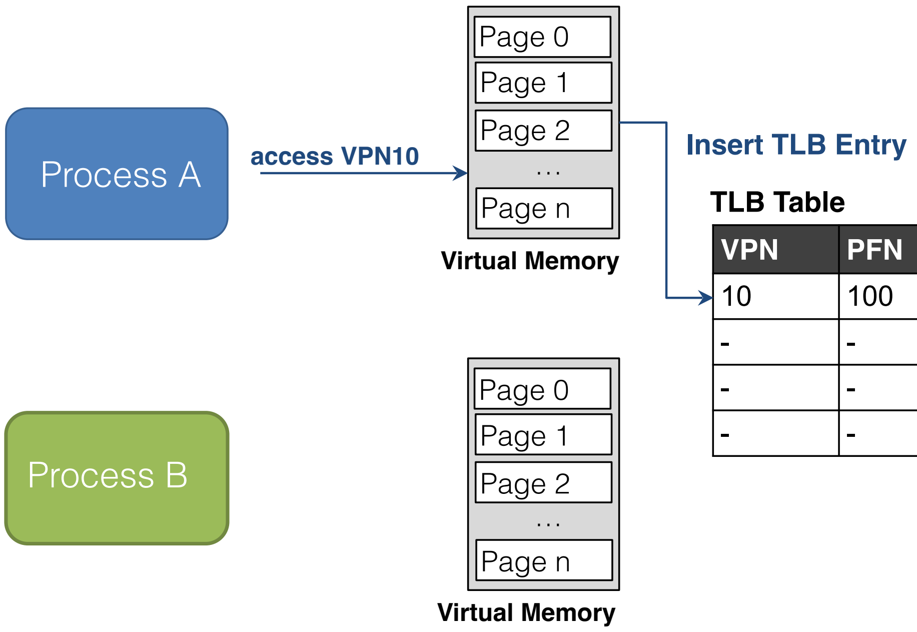
TLB entry

- TLB entries are often *fully associative* (any entry for any mapping)
  - A typical TLB might have 32, 64, or 128 entries.
  - Hardware searches the TLB in parallel to find the translation.
  - other bits: valid, protection, address-space identifier, dirty bit



Typical TLB entry

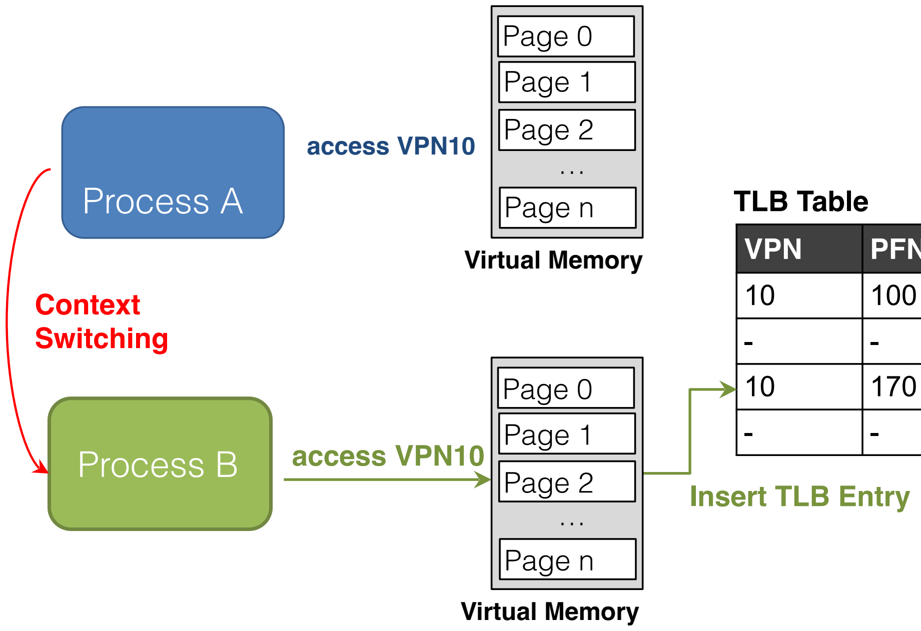
TLB Issue: Context Switching



TLB Table

VPN	PFN	valid	prot
10	100	1	rwx
-	-	-	-
-	-	-	-
-	-	-	-

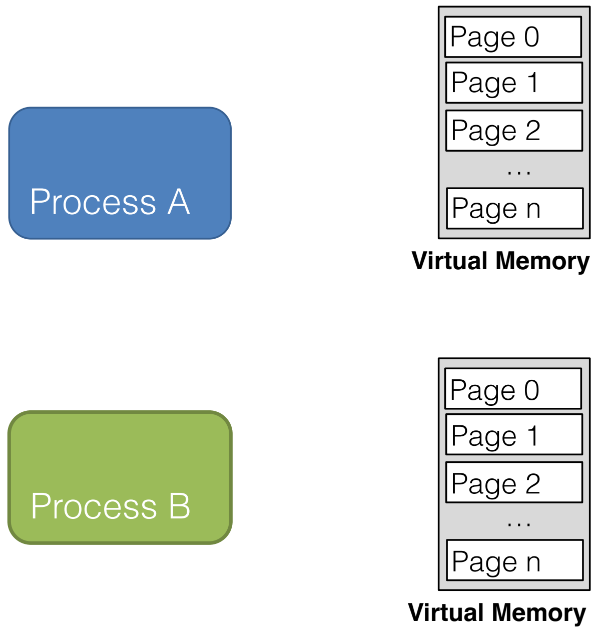
TLB Issue: Context Switching



TLB Table

VPN	PFN	valid	prot
10	100	1	rwx
-	-	-	-
10	170	1	rwx
-	-	-	-

TLB Issue: Context Switching



TLB Table

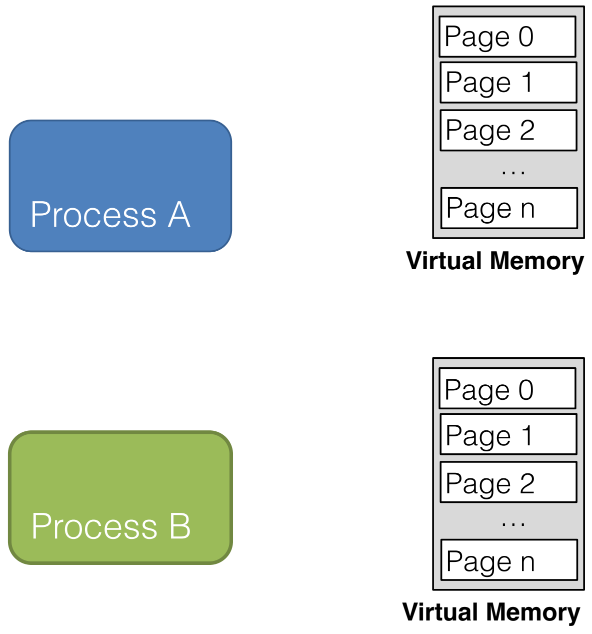
VPN	PFN	valid	prot
10	100	1	rwx
-	-	-	-
10	170	1	rwx
-	-	-	-

Can't Distinguish which entry is meant for which process

Could just flush the TLB on every context switch…

Disambiguating Address Spaces

- Provide an address space identifier(ASID) field in the TLB.



TLB Table

VPN	PFN	valid	prot	ASID
10	100	1	rwx	1
-	-	-	-	-
10	170	1	rwx	2
-	-	-	-	-

Another Case

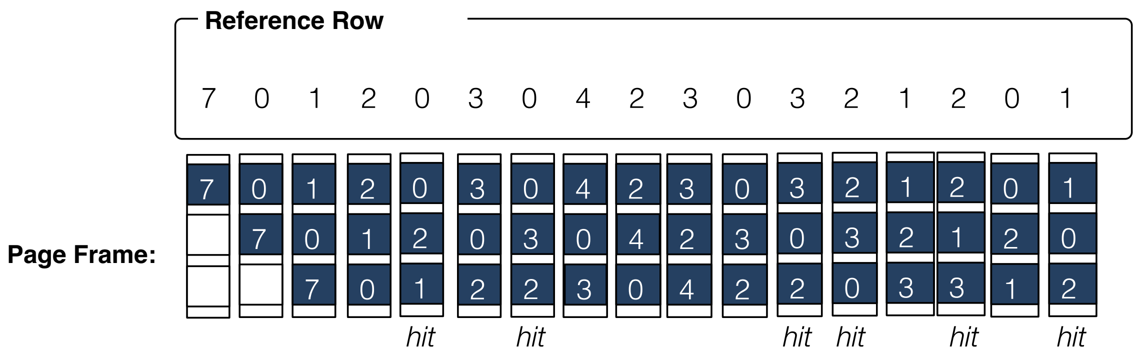
- Two processes share a page.
  - Process 1 is sharing physical page 101 with Process2.
  - P1 maps this page into the 10th page of its address space.
  - P2 maps this page to the 50th page of its address space.

VPN	PFN	valid	prot	ASID
10	101	1	rwx	1
-	-	-	-	-
50	101	1	rwx	2
-	-	-	-	-

Sharing of pages is useful as it reduces the number of physical pages in use.

TLB Replacement Policy

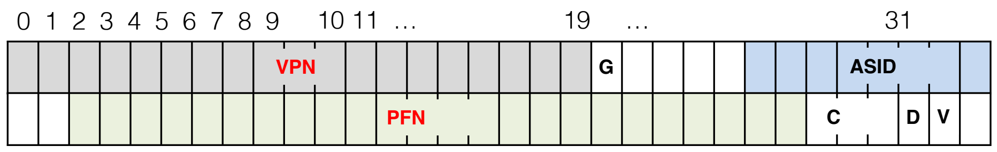
- LRU (Least Recently Used)
  - Evict an entry that has not recently been used.
  - Take advantage of *locality* in the memory-reference stream.



- 6 hits, 11 misses

A Real TLB Entry

64-bit MIPS R4000 TLB entry



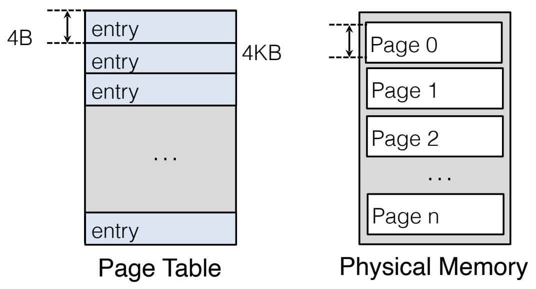
Flag	Content
19-bit VPN	The rest reserved for the kernel.
24-bit PFN	Systems can support with up to 64GB of main memory(pages).
Global bit(G)	Used for pages that are globally-shared among processes.
ASID	OS can use to distinguish between address spaces.
Coherence bit(C)	determine how a page is cached by the hardware.
Dirty bit(D)	marking when the page has been written.
Valid bit(V)	tells the hardware if there is a valid translation present in the entry.

Virtual Memory

- 13 - Address Spaces
- 14 - Memory API
- 15 - Address Translation
- 16 - Segmentation
- 17 - Free Space Management
- 18 - Paging
- 19 - Translation Lookaside Buffers
- 20 - Advanced Paging
- 21 - Swapping
- 22 - Swapping Policy

Paging: Linear (Single-Level) Tables

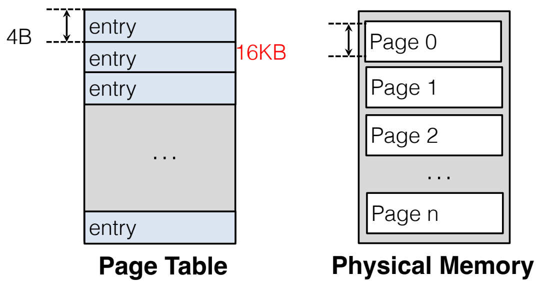
- Usually one page table for every process in the system.
- Example:
  - 32-bit address space, 4KB pages, 4-byte page-table entries



Page table size = $\frac{2^{32}}{2^{12}} * 4Byte = 4MByte$

Paging: Smaller Tables

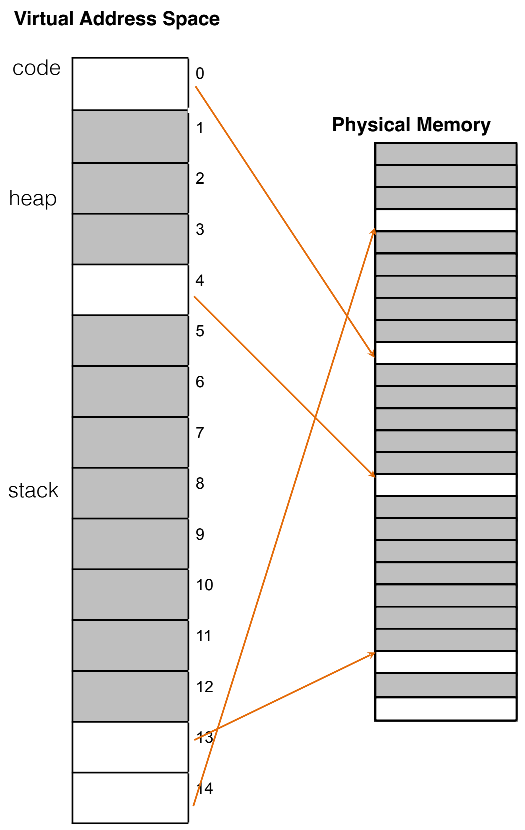
- Larger pages mean fewer entries
  - 32-bit address space, 16KB pages, 4-byte entries.



$\frac{2^{32}}{2^{16}} * 4 = 1MB$ **per page table**

Big pages lead to internal fragmentation.

Problem

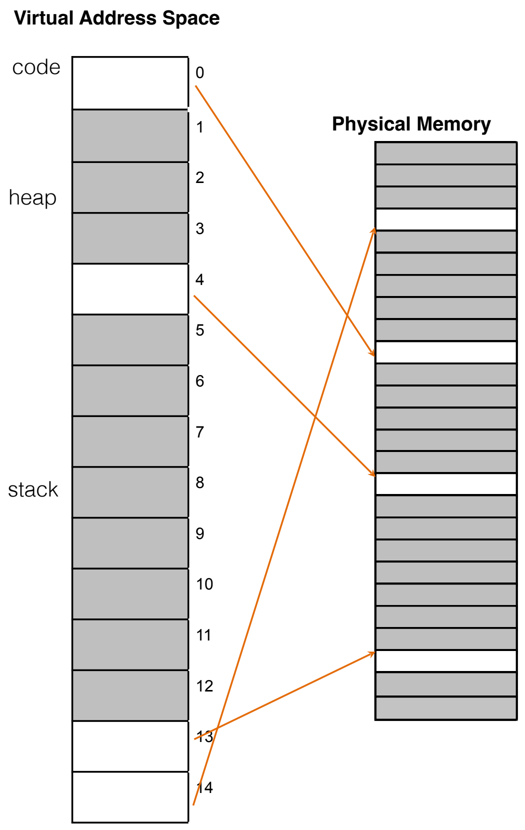


PFN	valid	prot	present	dirty
10	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
…	…	…	…	…
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

A Page Table For 16KB Address Space

Problem

Most of the page table is **unused**

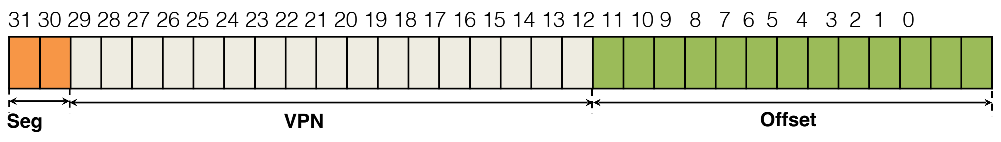


PFN	valid	prot	present	dirty
9	1	r-x	1	0
-	0	-	-	-
-	0	-	-	-
-	0	-	-	-
15	1	rw-	1	1
…	…	…	…	…
-	0	-	-	-
3	1	rw-	1	1
23	1	rw-	1	1

A Page Table For 16KB Address Space

Hybrid: Page Table Per Segment

- Each process has three page tables associated with it.
  - Base register for each segment is physical address of its page table.



32-bit Virtual address space with 4KB pages

Seg value	Content
00	unused segment
01	code
10	heap
11	stack

GeekOS!

TLB miss on Hybrid Approach

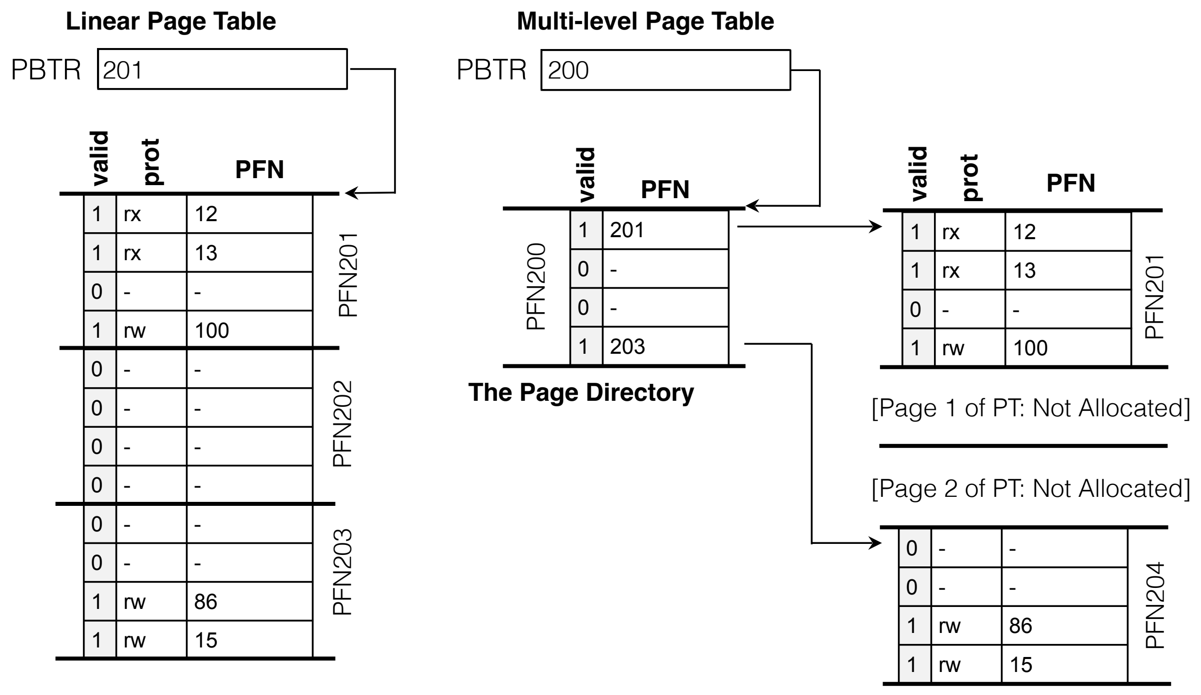
- Need physical address of entry from page table.
  - Segment bits (SN) determine which base and bounds pair
  - Hardware combines physical address therein and the VPN to form the address of the page table entry (PTE) .

```
SN = (VirtualAddress & SEG_MASK) >> SN_SHIFT
VPN = (VirtualAddress & VPN_MASK) >> VPN_SHIFT
AddressOfPTE = Base[SN] + (VPN * sizeof(PTE))
```

Multi-level Page Tables

- Hybrid Approach is not without problems
  - Assumes specific segment layout
  - Sparsely-used heap still leads to external fragmentation
- So turn the linear page table into something like a tree
  - Page the page table
  - Allocate page-table pages as needed
  - Track valid page table pages with *page directory*

Multi-level Page Tables: Page directory

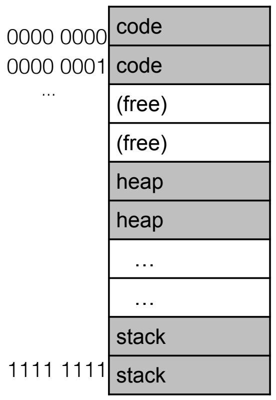


Linear (Left) And Multi-Level (Right) Page Tables

Multi-level Page Tables

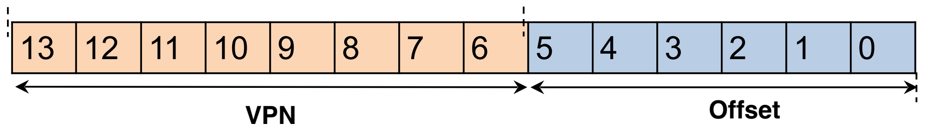
- Page directory has:
  - one page directory entry (PDE) per page of the page table
  - Valid bit and page frame number (PFN)
- Advantages
  - Page-table space in proportion to used address space
  - OS can lazily allocate new pages as need
  - *Indirection* can disperse page-table pages through memory
- Disadvantages
  - Time and space tradeoff
  - Complexity

A Detailed Multi-Level Example



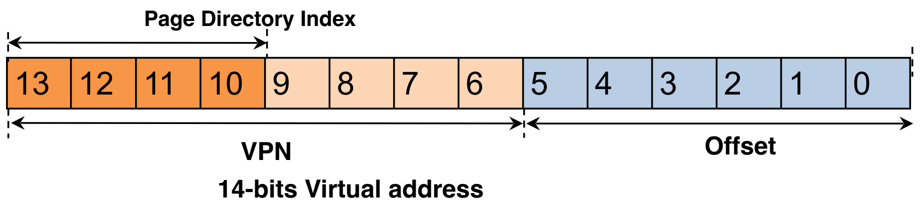
Flag	Detail
Address space	16 KB
Page size	64 byte
Virtual address	14 bit
Num pages	$2^{14}/2^6 = 2^8 = 256$ pages
VPN	8 bit
Offset	6 bit
Page table entry	4 bytes

A 16-KB Address Space With 64-byte Pages



Detailed Example

- Page directory has one entry per page of the page table
  - 256 pages, 4 bytes for PTE, 64-byte pages
    - $\frac{256 \times 4}{64} = 16$ pages for directory —> 4 bits for PDI
    - each page can hold 64/4=16 entries —> 4 bits for PTI
- Accessing invalid page-directory entry raises exception



- Page-table index (PTI) is used to index into the page table page