Operating System: Translation Lookaside Buffers

Sang Ho Choi (shchoi@kw.ac.kr)
School of Computer & Information Engineering
KwangWoon University

Temporal Overhead of Paging

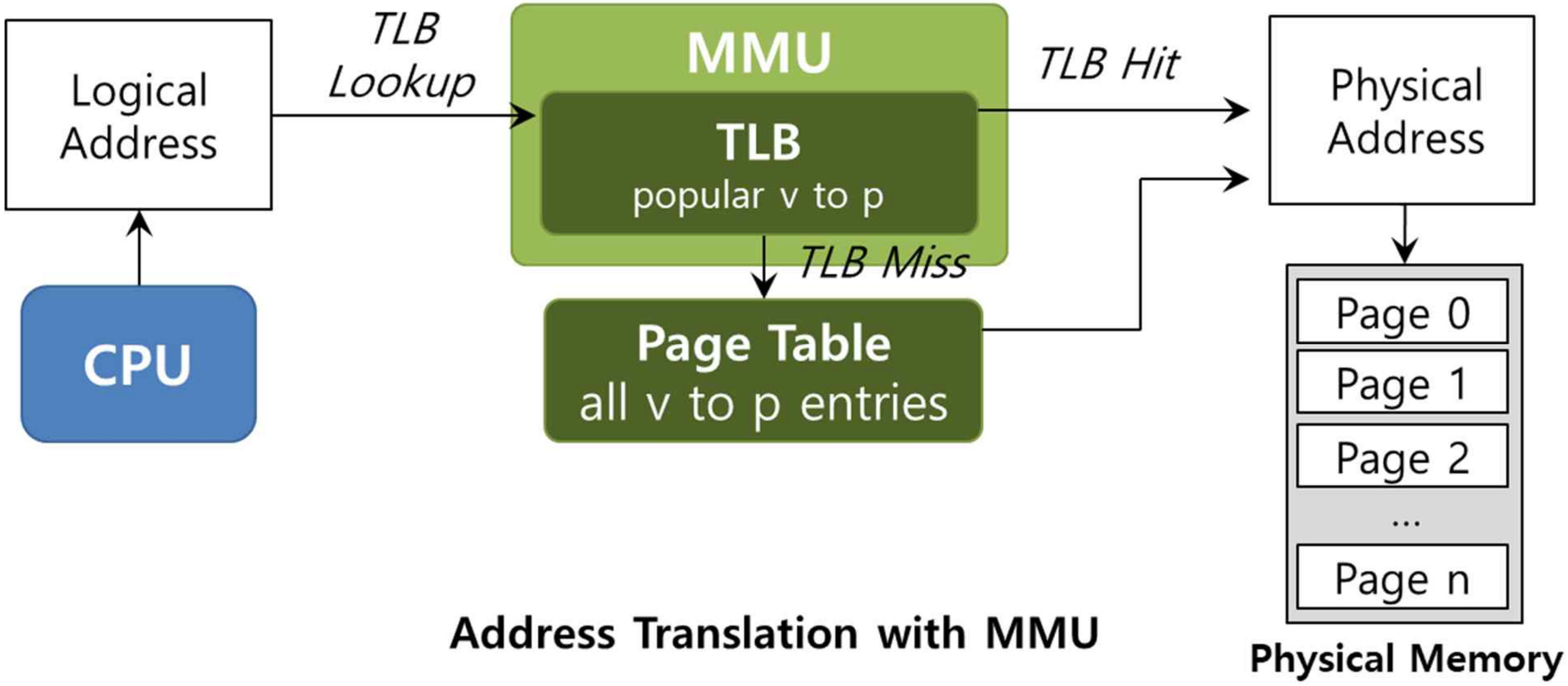
- Address translation is too slow
  - A simple linear page table doubles the cost of memory lookups
    - ➢One for the page table, another to fetch the data or instruction
  - Multi-level page tables increase the cost further (discussed later)

pagetable 읽고 P.A에 접근하니까.

- Goal: make address translation fast
  - Make fetching from a virtual address about as efficient as fetching from a physical address

TLB (Translation Lookaside Buffer)

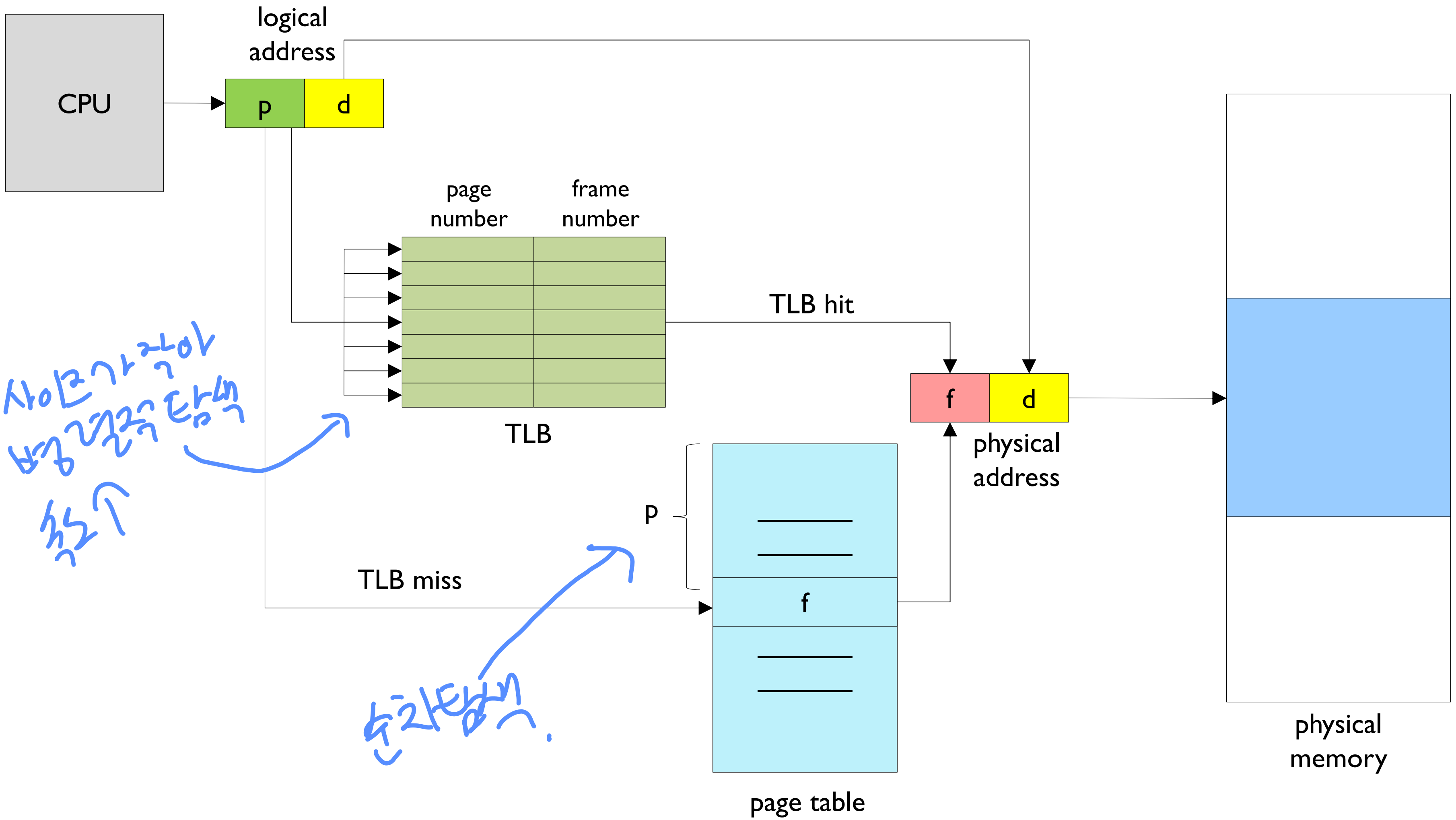
- Part of the chip's memory-management unit (MMU)
- A hardware cache of **popular** virtual-to-physical address translation 자주접근하는 주소들을 캐시처럼 저장.



Address Translation with MMU

TLB (Cont.)

- Address translation with TLB



사이즈가 작아 병렬적 탐색 속↑

순차탐색.

TLB Basic Algorithms

```
1: VPN = (VirtualAddress & VPN_MASK ) >> SHIFT             VPN만 추출
2: (Success , TlbEntry) = TLB_Lookup(VPN)                  TLB 보고
3: if(Success == True){ // TLB Hit
4:    if(CanAccess(TlbEntry.ProtectBit) == True ){         권한 있으면
5:        offset = VirtualAddress & OFFSET_MASK            오프셋 추출하고
6:        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset      PFN 쉬프트한거랑 오프셋
                                                           OR연산
7:        AccessMemory( PhysAddr )                         접근.
8:    }else
9:        RaiseException(PROTECTION_ERROR)                 권한에러
```

- (1 lines) extract the virtual page number (VPN)
- (2 lines) check if the TLB holds the translation for this VPN
- (5-8 lines) extract the page frame number from the relevant TLB entry, and form the desired physical address and access memory

TLB Basic Algorithms (Cont.)

Page table Base Register (P.T의 시작점)

Page table entry

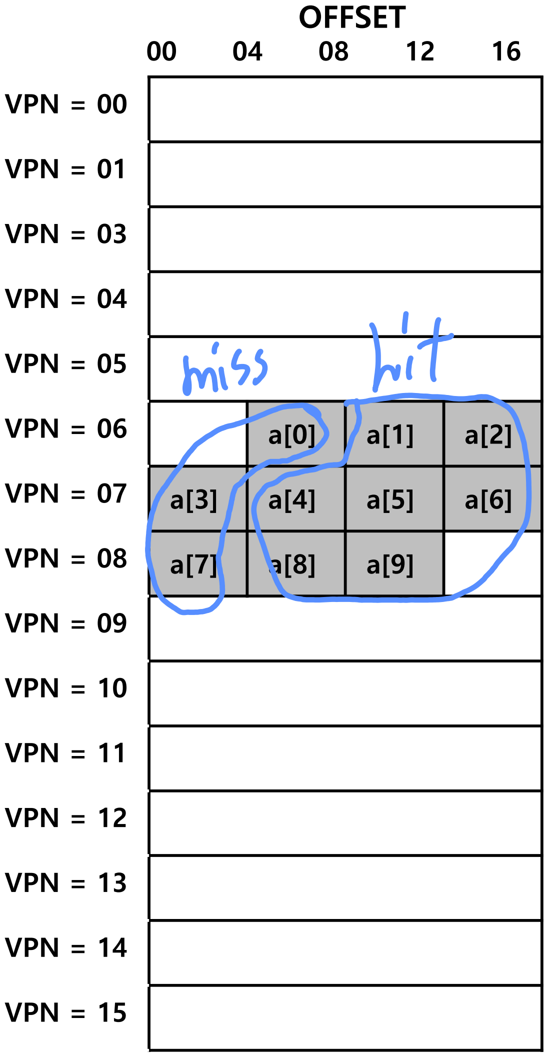
```
11:else //TLB Miss
12:         PTEAddr = PTBR + (VPN * sizeof(PTE))
13:         PTE = AccessMemory(PTEAddr)
14:         (…)
15:
16:         TLB_Insert( VPN , PTE.PFN , PTE.ProtectBits)
17:         RetryInstruction()
18:     }
19:}
```

사용했던거니 TLB에 업데이트.

- (11-12 lines) The hardware accesses the page table to find the translation
- (16 lines) updates the TLB with the translation

Example: Accessing An Array

- How a TLB can improve its performance



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

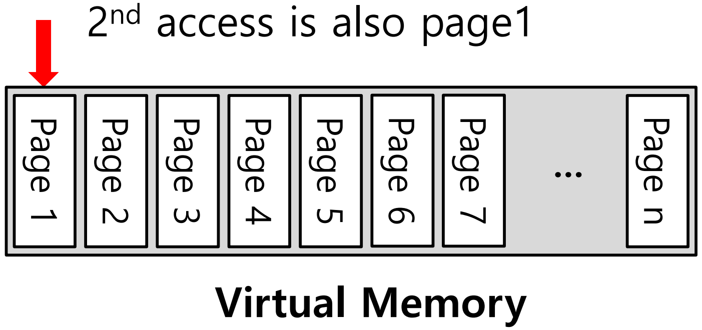
The TLB improves performance due to spatial locality

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

Locality

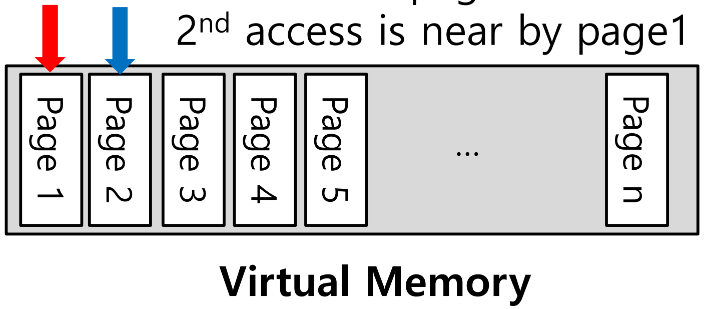
- Temporal Locality
  - An instruction or data item that has been recently accessed will likely be re-accessed soon in the future

1st access is page1
2nd access is also page1



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

1st access is page1
2nd access is near by page1



TLB Performance

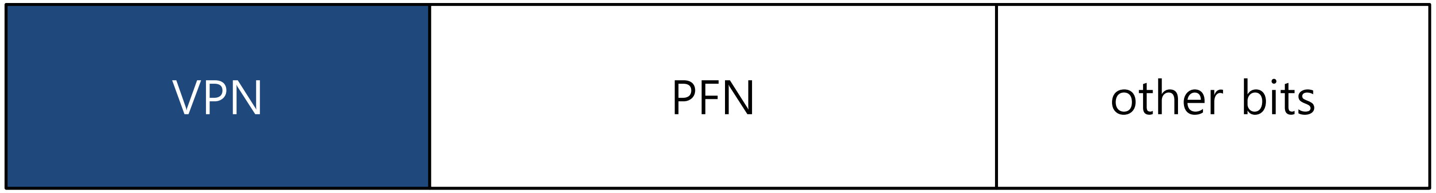
- Effective Access Time with TLB
  - TLB lookup time: ε
  - Memory access time: 1
  - Hit ratio: α
    - Percentage that is found in TLB
  - Effective Access Time

    = α x Hit memory time + (1–α) x miss memory time

    = α (1 + ε) + (1 - α)(2 + ε) = 2 + ε - α

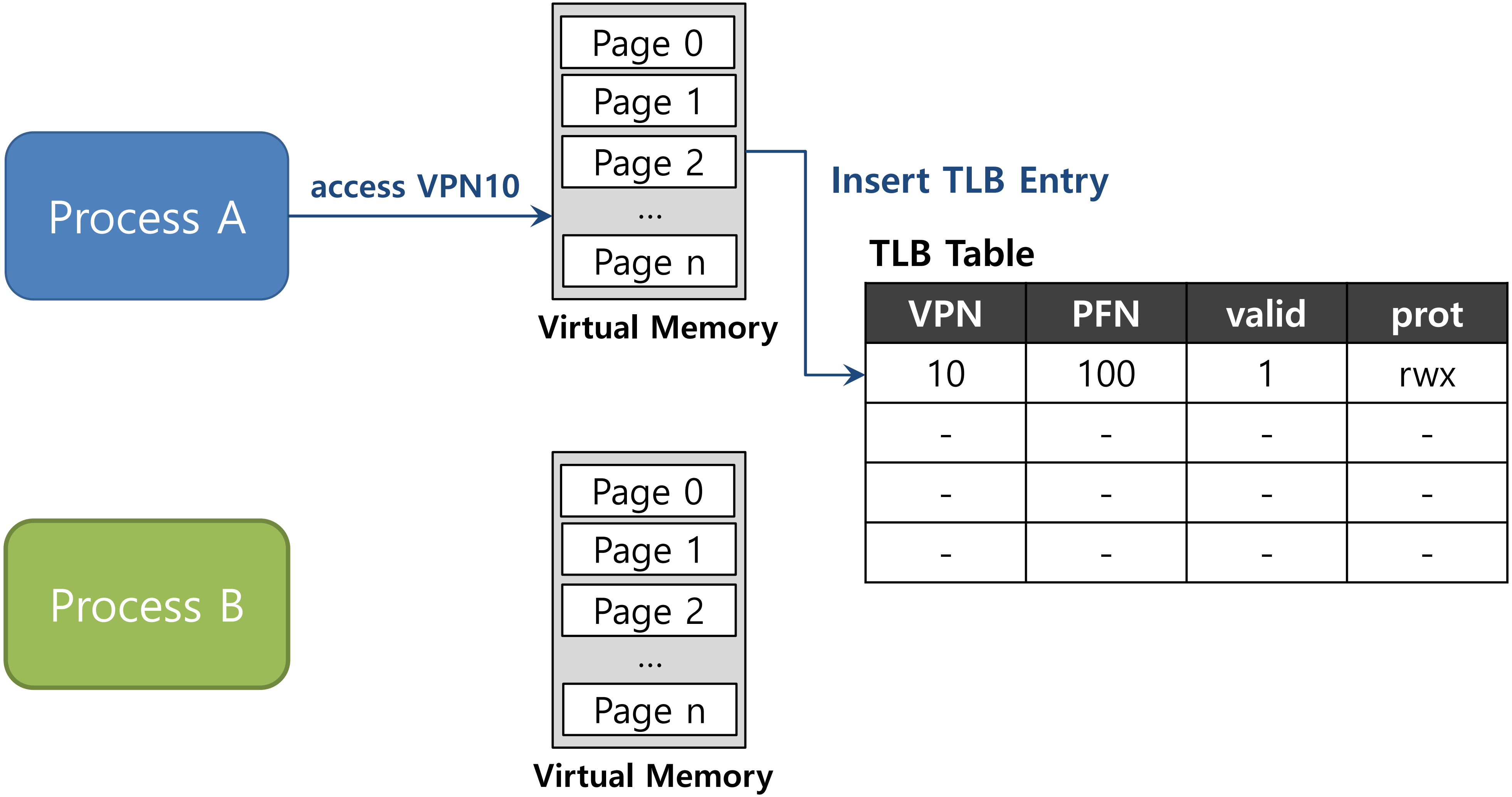
TLB entry

- TLB is managed by **Full Associative** method
  - A typical TLB might have 32,64, or 128 entries
  - Hardware search the entire TLB in parallel to find the desired translation
  - other bits: valid bits , protection bits, address-space identifier, dirty bit



Typical TLB entry look like this

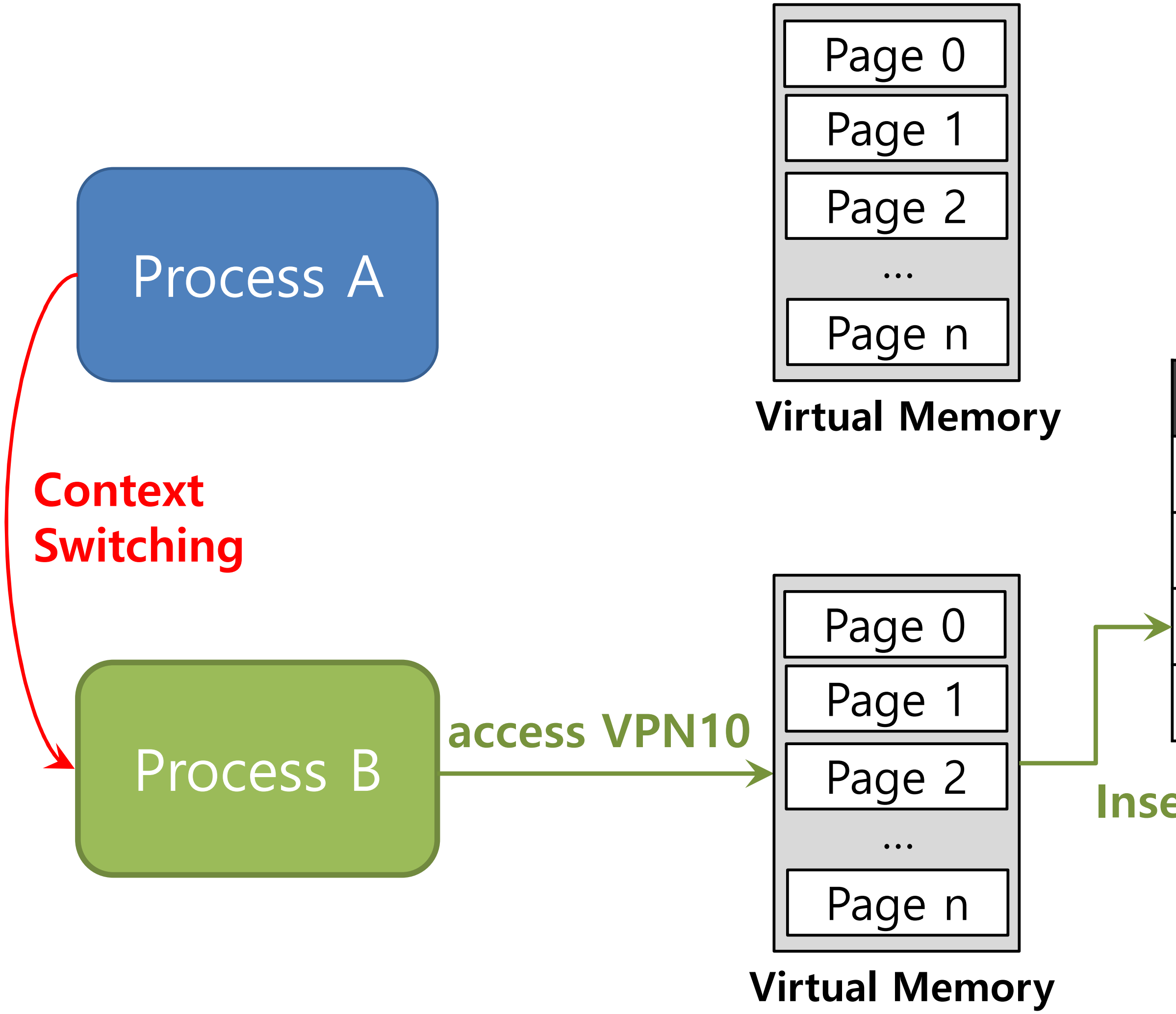
TLB Issue: Context Switching



TLB Table

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

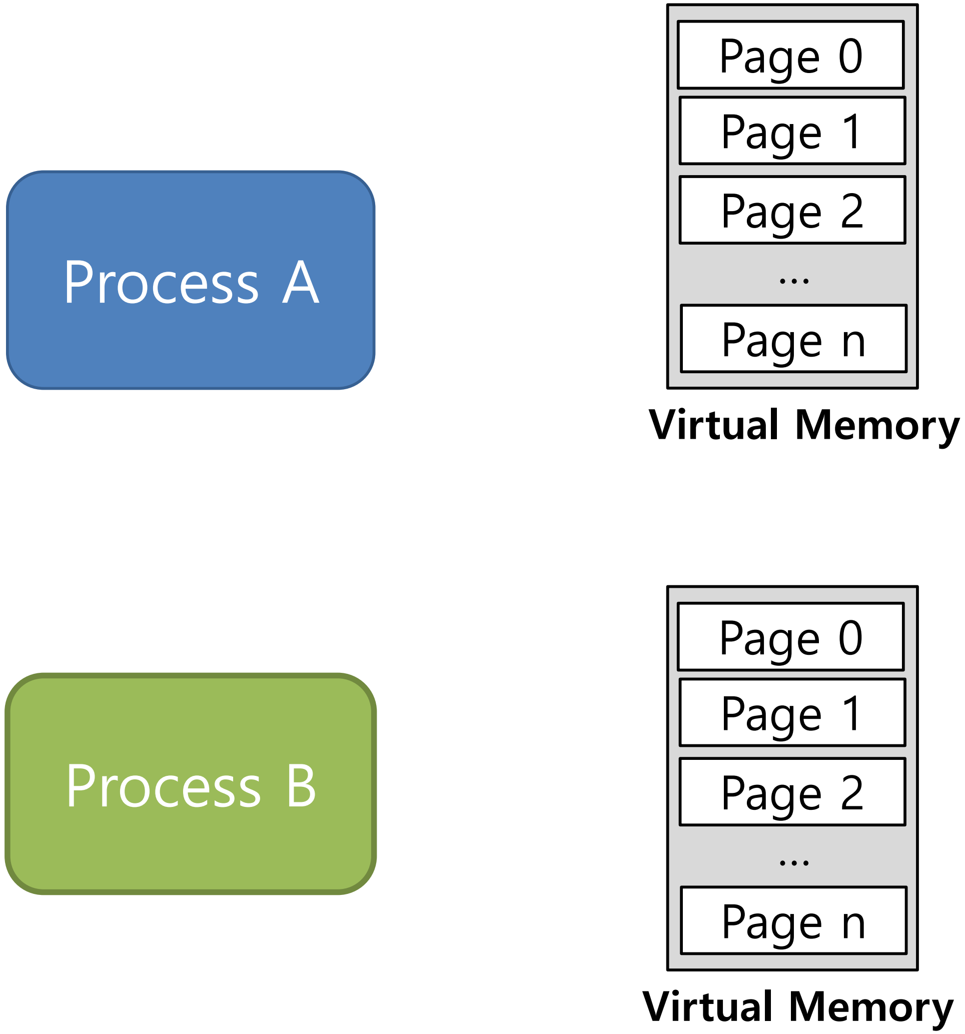
TLB Issue: Context Switching



TLB Table

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

TLB Issue: Context Switching



TLB Table

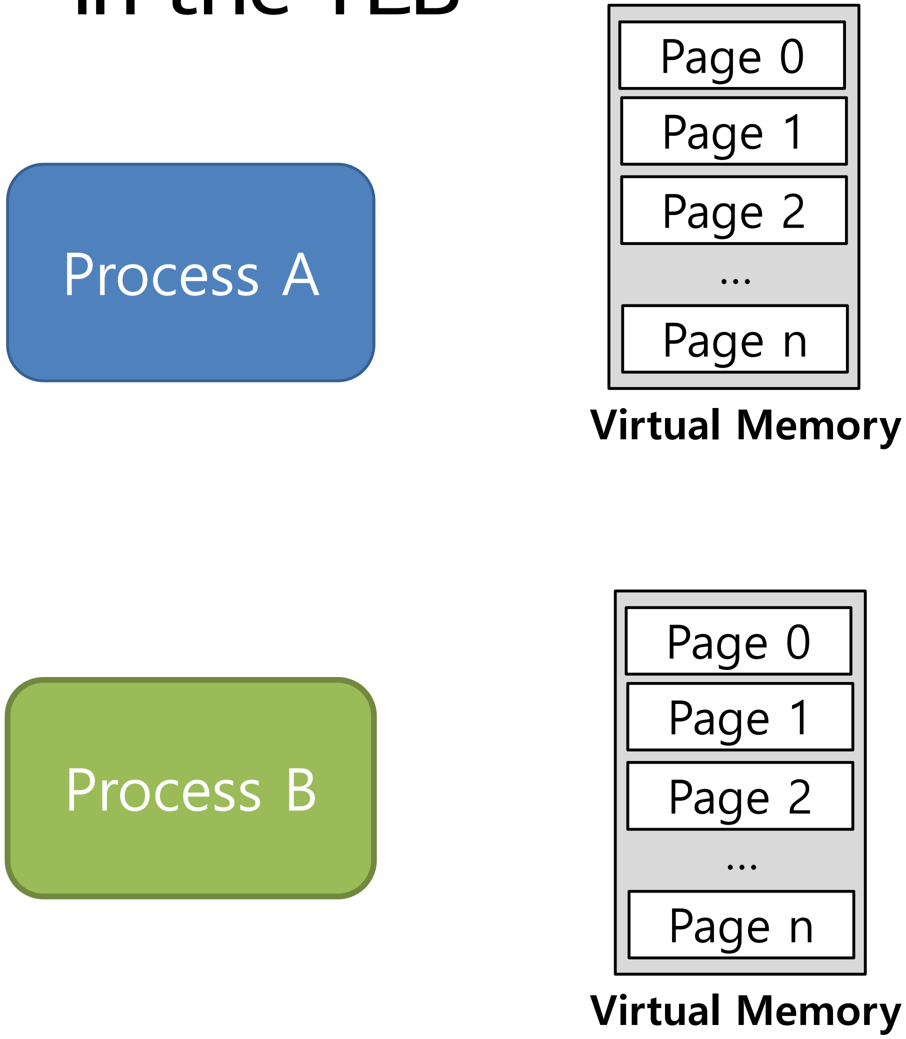
VPN	PFN	valid	prot
10	100	1	rwx
-	-	-	-
10	170	1	rwx
-	-	-	-

Can’t Distinguish which entry is meant for which process

Flush TLB on each context switch → Cost is high

To Solve Problem

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



Process A

Page 0
Page 1
Page 2
…
Page n

Virtual Memory

Process B

Page 0
Page 1
Page 2
…
Page n

Virtual Memory

TLB Table

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

Another Case

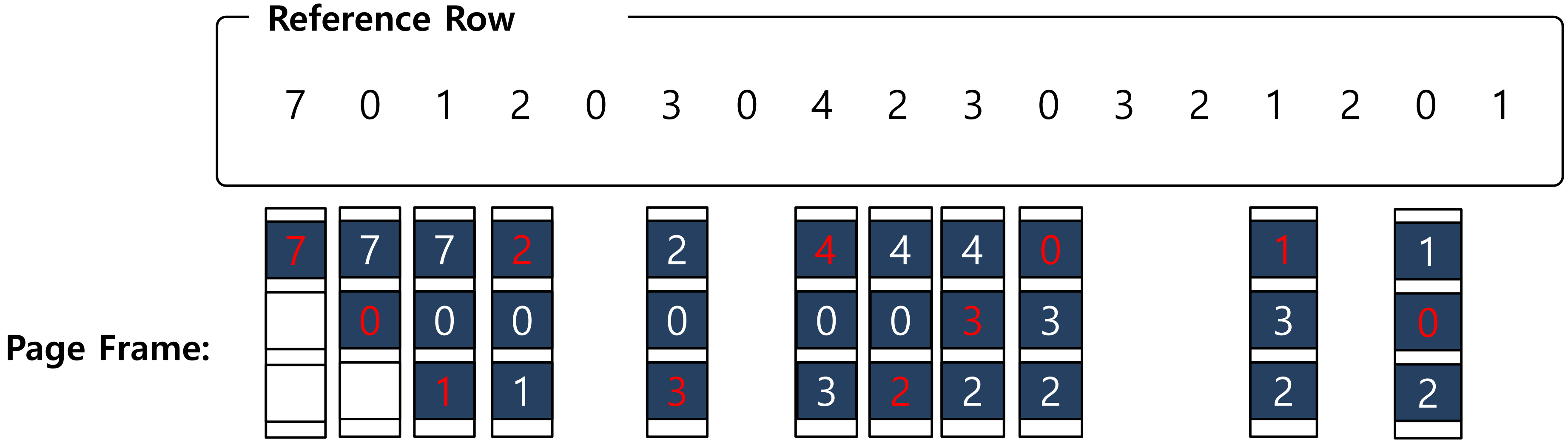
- Two processes share a page
  - Process 1 is sharing physical page 101 with Process2
  - P1 maps this page into the 10th page of its address space
  - P2 maps this page to the 50th page of its address space

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

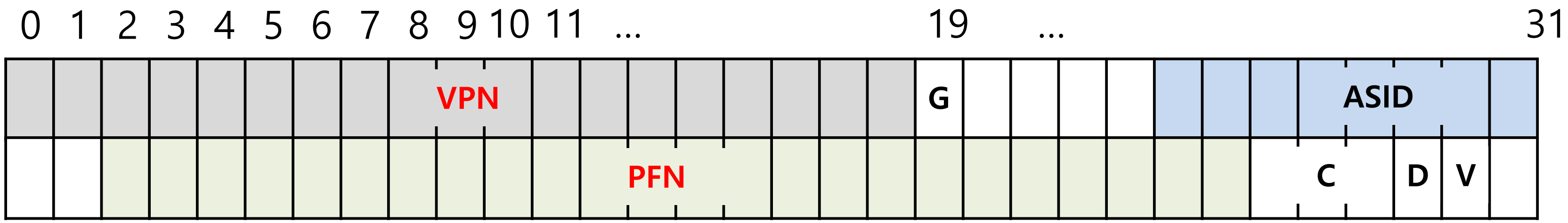


Reference Row	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1
Page Frame:	7	7	7	2		2		4	4	4	0			1		1	
		0	0	0		0		0	0	3	3			3		0	
			1	1		3		3	2	2	2			2		2	

Total 11 TLB miss

A Real TLB Entry

All 64 bits of this TLB entry (example of MIPS R4000)



0 1 2 3 4 5 6 7 8 9 10 11 … 19 … 31

VPN G ASID

PFN C D V

Flag	Content
19-bit VPN	The rest reserved for the kernel.
24-bit PFN	Systems can support with up to 64GB of main memory($2^{24} * 4KB$ 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.