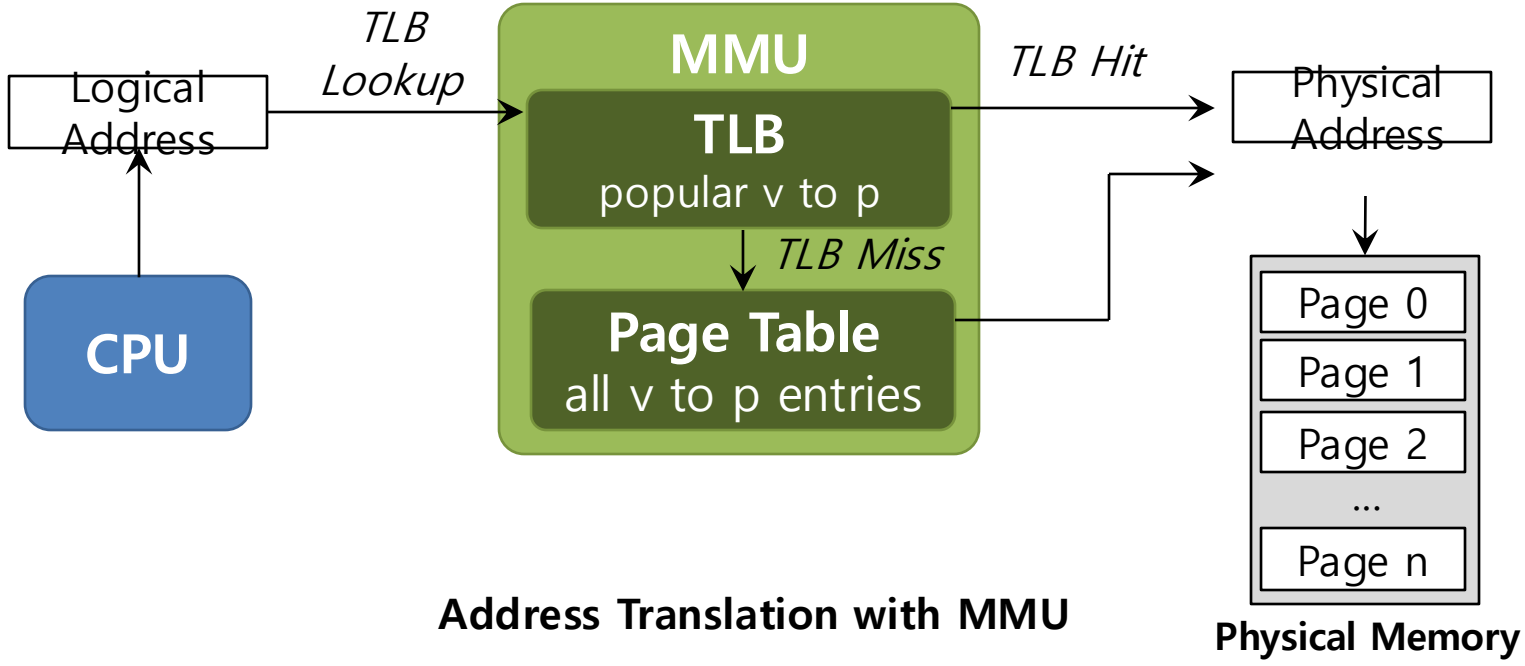
Locality and Caching

TLB (Translation Look-Aside Buffer)

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



Address Translation with MMU

TLB Basic Algorithms

```
1: VPN = (VirtualAddress & VPN_MASK ) >> SHIFT
2: (Success , TlbEntry) = TLB_Lookup(VPN)
3:    if(Success == True){ // TLB Hit
4:    if(CanAccess(TlbEntry.ProtectBit) == True ){
5:        offset = VirtualAddress & OFFSET_MASK
6:        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:        AccessMemory( PhysAddr )
8:    }else RaiseException(PROTECTION_ERROR)
```

- (1 lines) extract the virtual page number(VPN)
- (2 lines) check if the TLB holds the transalation 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

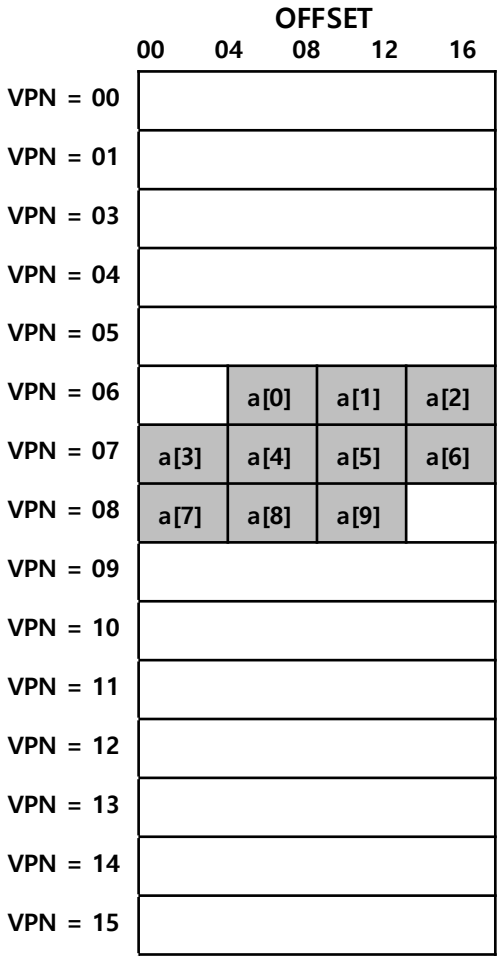
TLB Basic Algorithms

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

- (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



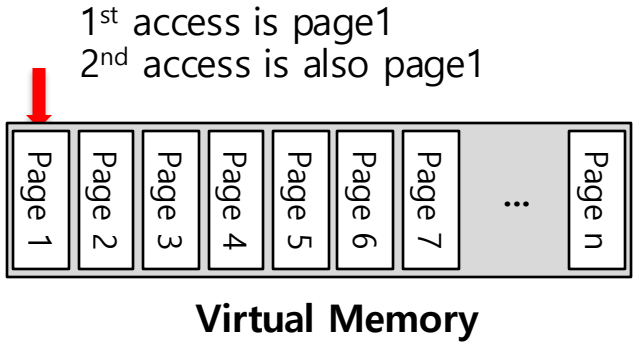

```
0:      int sum = 0 ;
1:      for( i=0; i<10; i++){
2:              sum+=a[i];
3:      }
```

The TLB improves performance due to spatial locality

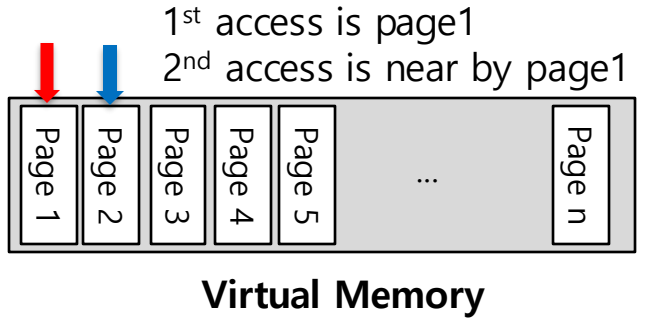
3 misses and 7 hits
Thus TLB hit rate is 70%

Locality

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



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



Who Handles The TLB Miss?

- Hardware handle the TLB miss entirely on CISC
  - The hardware has to know exactly where the page tables are located in memory
  - The hardware would "walk" the page table, find the correct page-table entry and extract the desired translation, update and retry instruction
  - **hardware-managed TLB**

Who Handles The TLB Miss?

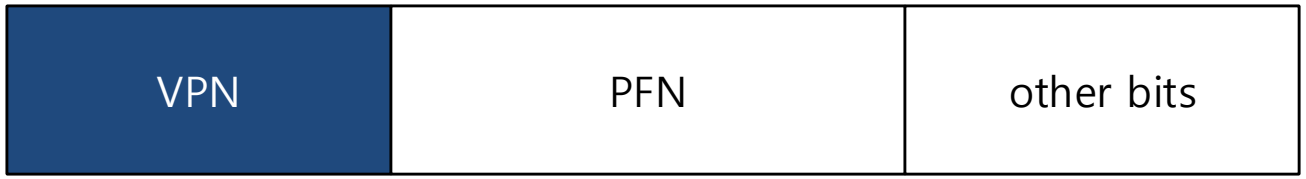
- RISC have what is known as a **software-managed TLB**
  - On a TLB miss, the hardware raises exception(trap handler)
    - **Trap handler is code** within the OS that is written with the express purpose of handling TLB miss

TLB Control Flow Algorithm (OS Handled)

```
1:        VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2:        (Success, TlbEntry) = TLB_Lookup(VPN)
3:        if (Success == True) // TLB Hit
4:                if (CanAccess(TlbEntry.ProtectBits) == True)
5:                        Offset = VirtualAddress & OFFSET_MASK
6:                        PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:                        Register = AccessMemory(PhysAddr)
8:                else
9:                        RaiseException(PROTECTION_FAULT)
10:       else // TLB Miss
11:                RaiseException(TLB_MISS)
```

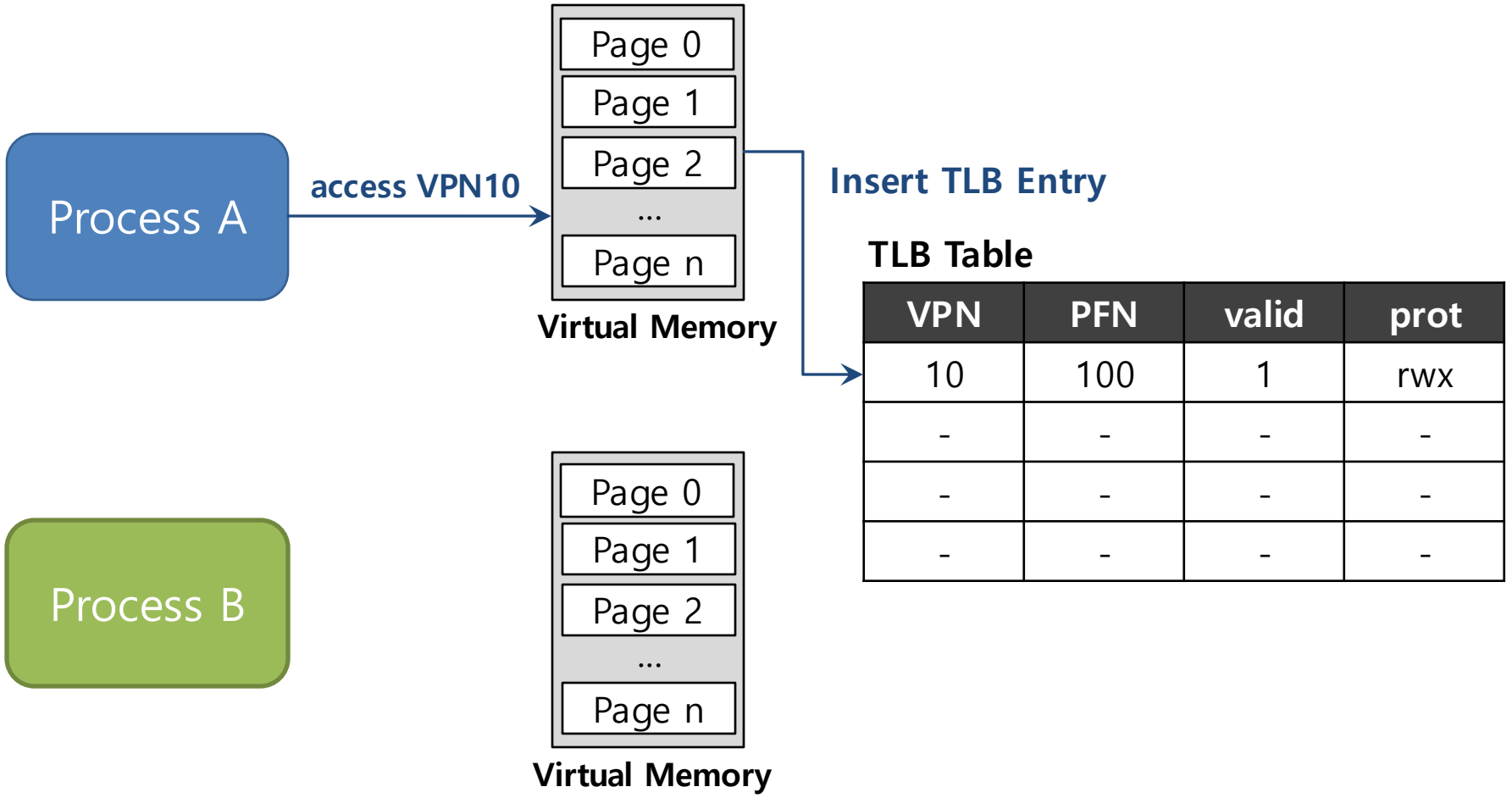
TLB Entry

- TLB is managed by **Fully-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



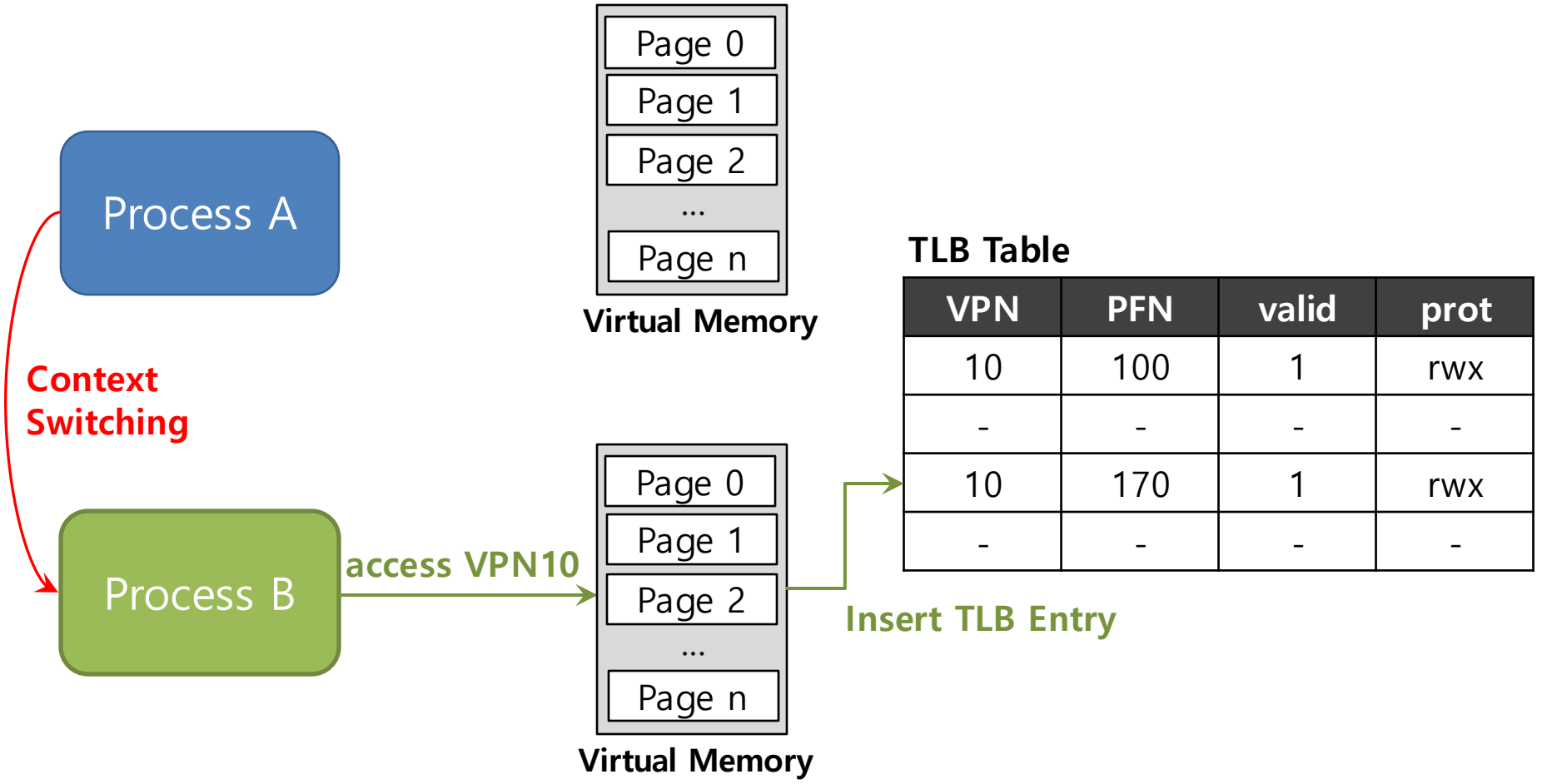
Typical TLB entry look like this

TLB Issue: Context Switching



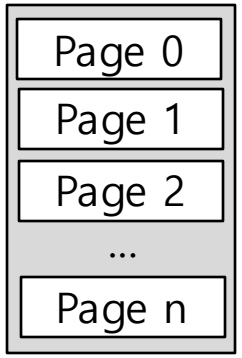
VPN	PFN	valid	prot
10	100	1	rwx
-	-	-	-
-	-	-	-
-	-	-	-

TLB Issue: Context Switching



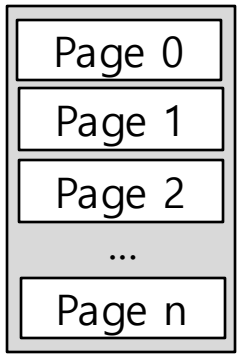
TLB Issue: Context Switching

Process A



Virtual Memory

Process B



Virtual Memory

TLB Table

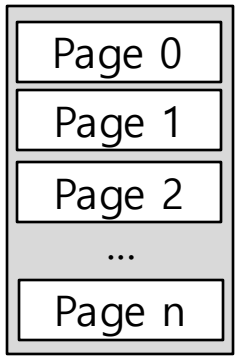
VPN	PFN	valid	prot
10	100	1	rwx
-	-	-	-
10	170	1	rwx
-	-	-	-

Can't Distinguish which entry is meant for which process

To Solve Problem

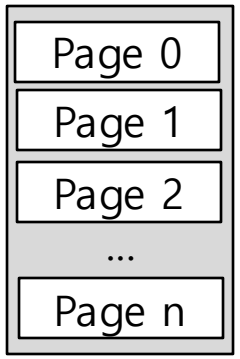
- Provide an address space identifier(ASID) field in the TLB

Process A



Virtual Memory

Process B



Virtual Memory

TLB Table

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

Another Case

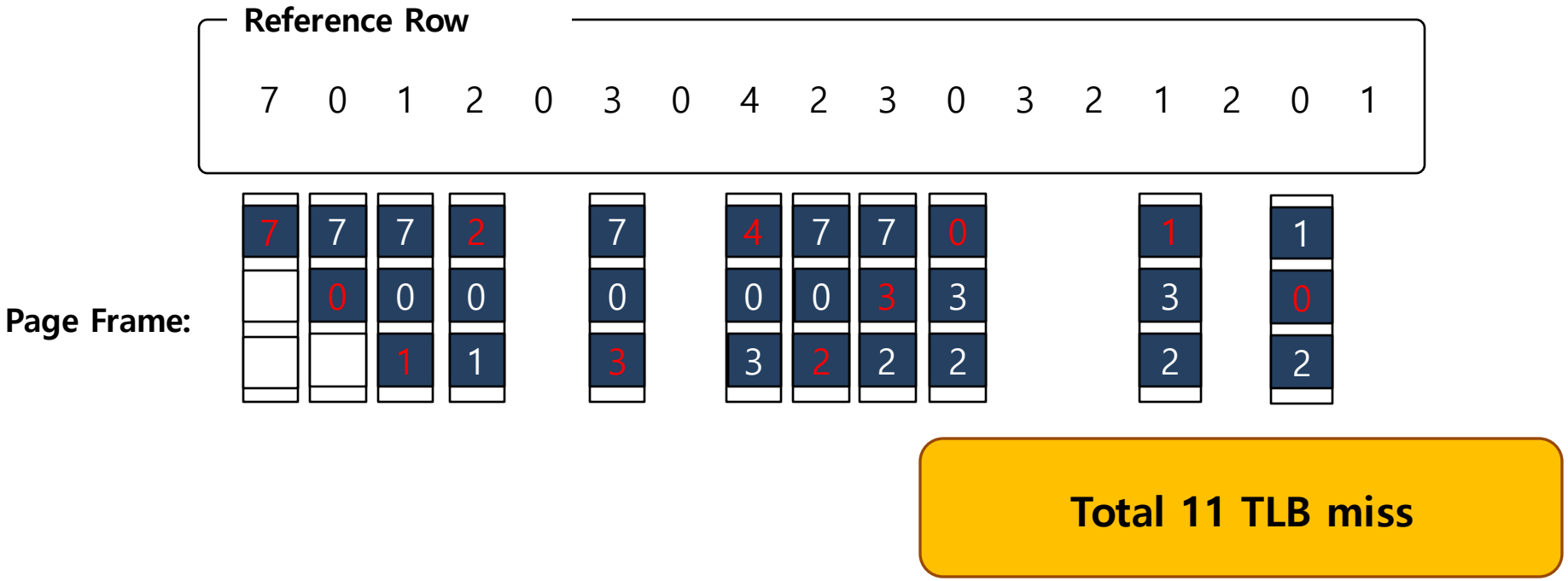
- Two processes share a page
  - Process 1 is sharing physical page 101 with Process2
  - P1 maps this page into the 10th page of its address space
  - P2 maps this page to the 50th page of its address space

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

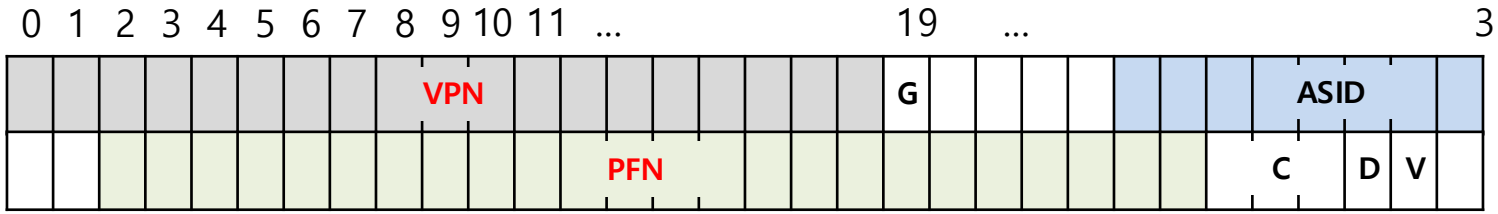
TLB Replacement Policy

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



A Real TLB Entry

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



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.