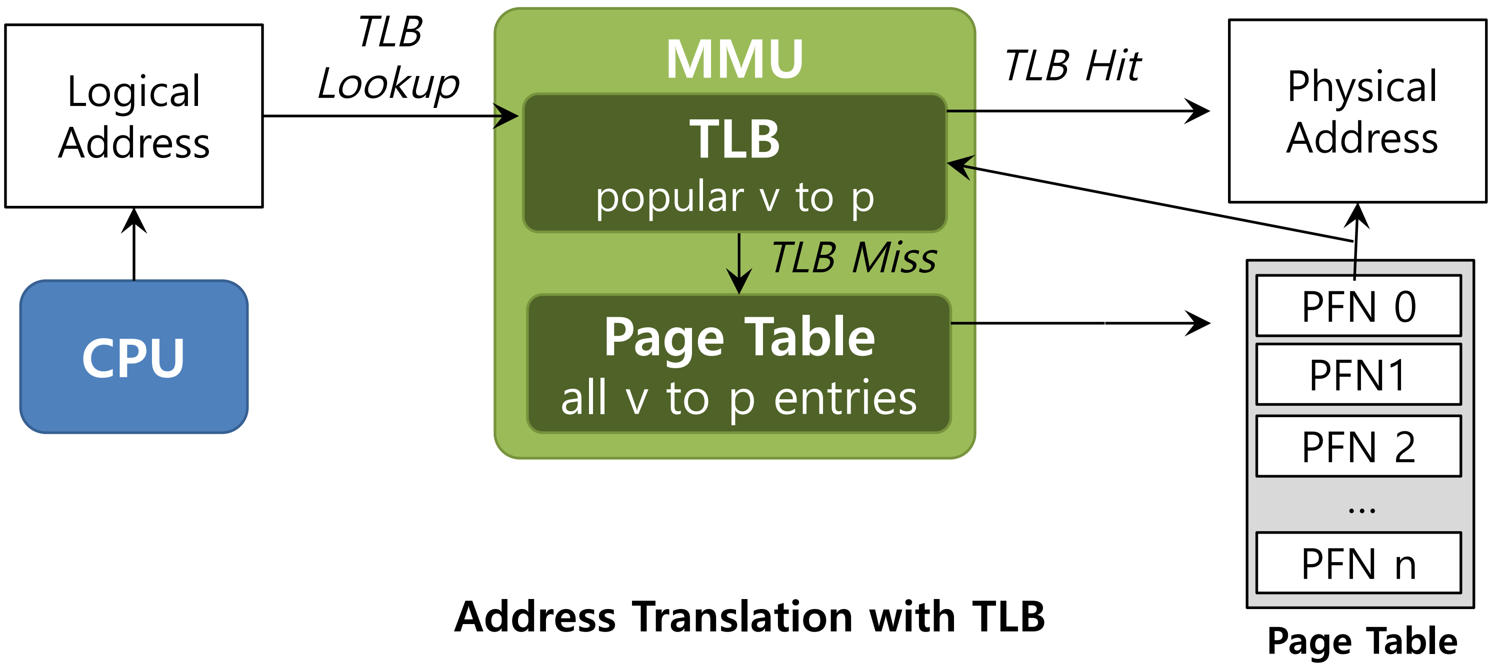
19. Translation Lookaside Buffers

Operating System: Three Easy Pieces

TLB

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



Address Translation with TLB

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 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.

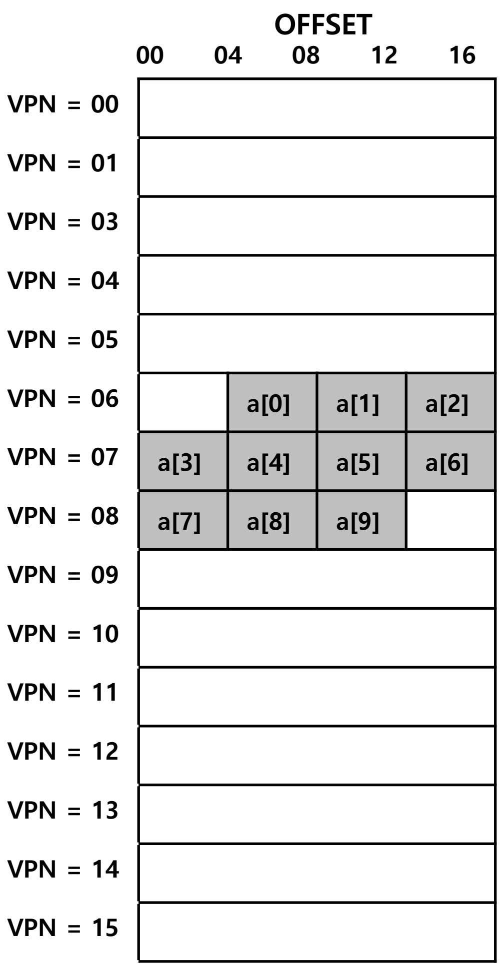
TLB Basic Algorithms (Cont.)

```
11:     }else{ //TLB Miss
12:         PTEAddr = PTBR + (VPN * sizeof(PTE))
13:         PTE = AccessMemory(PTEAddr)
14:         if (PTE.Valid == False) {
15:             RaiseException(SEGMENTATION_FAULT)
16:         }else if (CanAccess(PTE.ProtectBits) == False){
17:             RaiseException(PROTECTION_FAULT)
18:         }else{
19:             TLB_Insert( VPN , PTE.PFN , PTE.ProtectBits)
20:             RetryInstruction()}
21:     }
22:}
```

- (11-13 lines) Accesses the page table to find the translation.
- (13-17 lines) Check if PTE is ok
- (19 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];
}
```

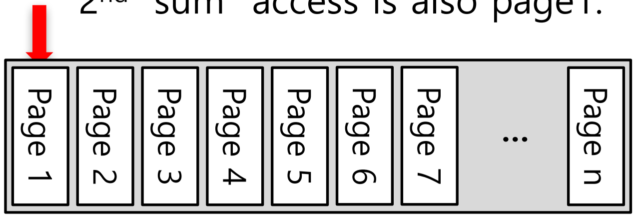
The TLB improves performance due to spatial locality

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.

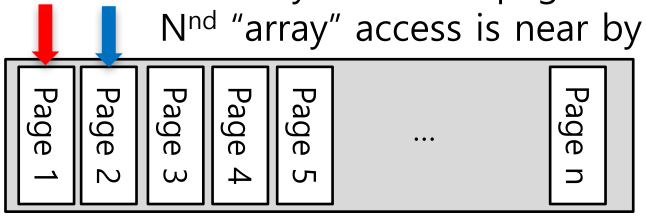
1st "sum" access is page1.
2nd "sum" access is also page1.



Virtual Memory

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

1st "array" access is page1.
Nnd "array" access is near by page1.



Virtual Memory

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 (TLB walker), find the correct page-table entry and **extract** the desired translation, **update** and **retry** instruction. This is the previous algorithm.
  - **hardware-managed TLB.**

Who Handles The TLB Miss? (Cont.)

- 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)

```
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)}
```

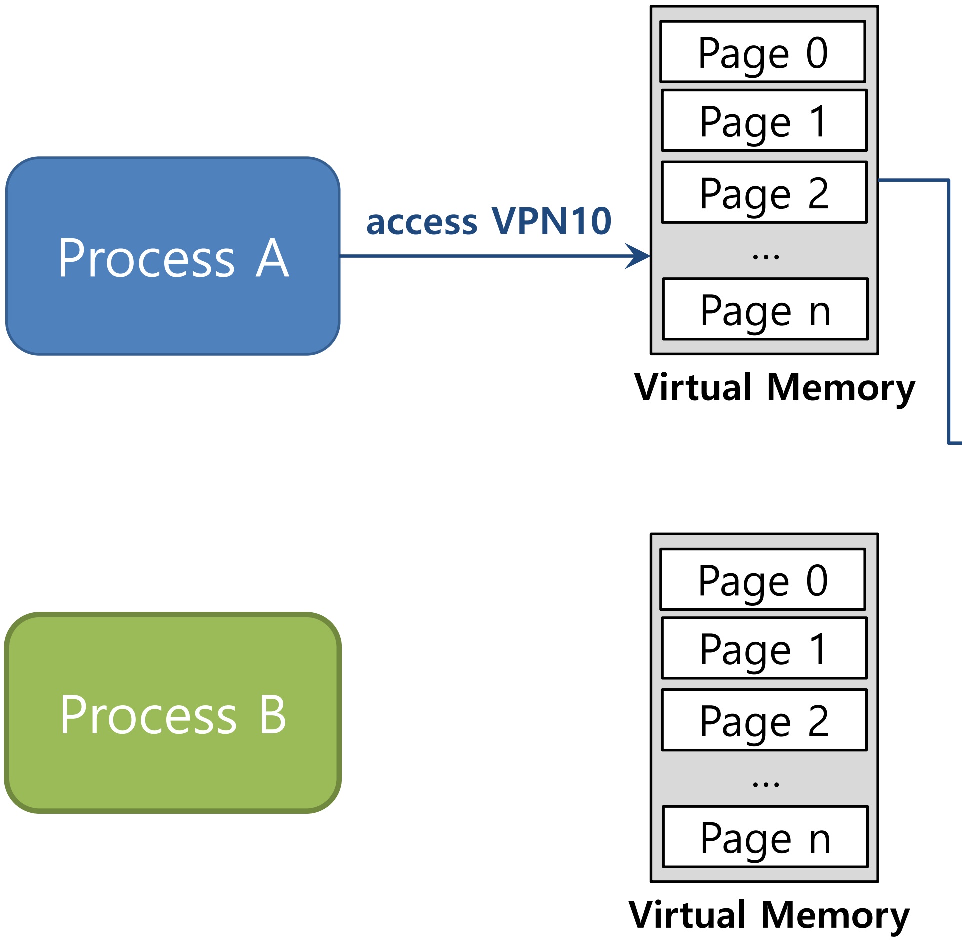
TLB entry

- TLB is managed by **Full Associative** method.
  - A typical TLB might have 32 to 128 entries.
  - Hardware search the entire TLB in parallel to find the desired translation (i.e. a cache without index bit, i.e. tag=VPN)
  - other bits: valid bits , protection bits, address-space identifier, dirty bit

VPN	PFN	other bits

Typical TLB entry look like this

TLB Issue: Context Switching

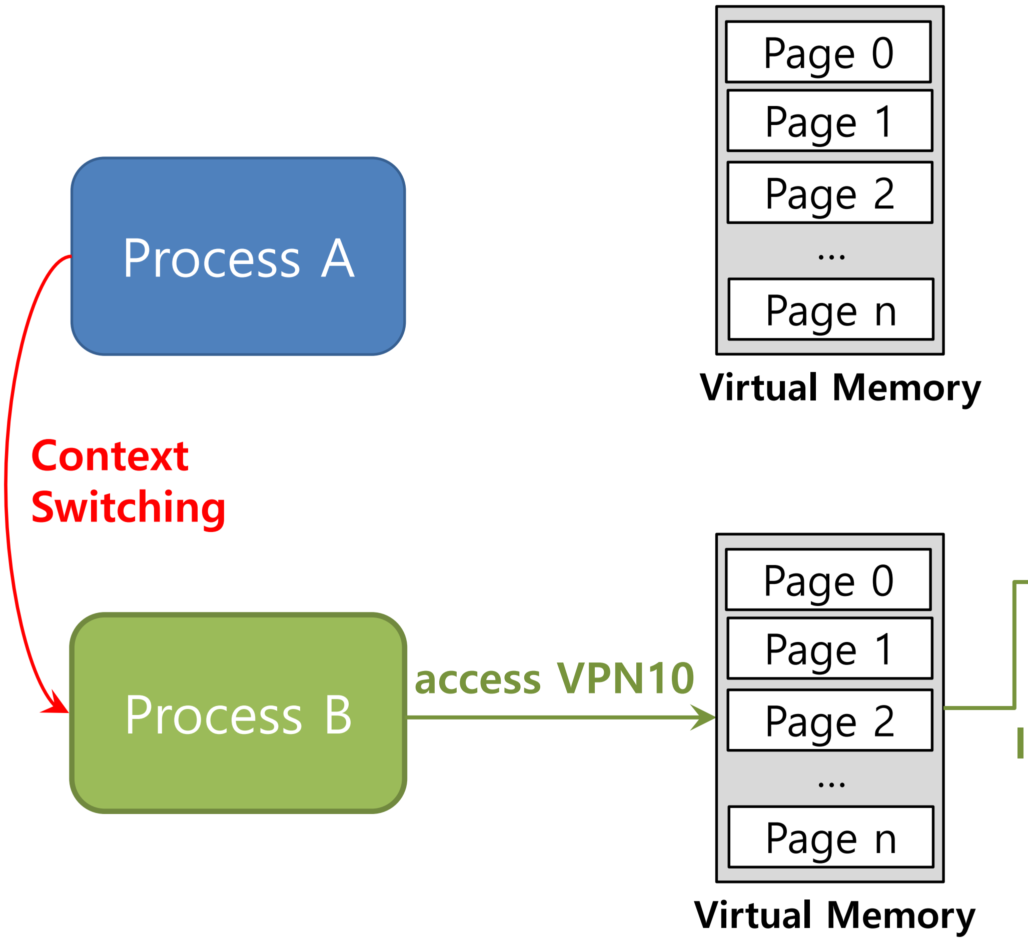


Insert TLB Entry

TLB Table

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

TLB Issue: Context Switching

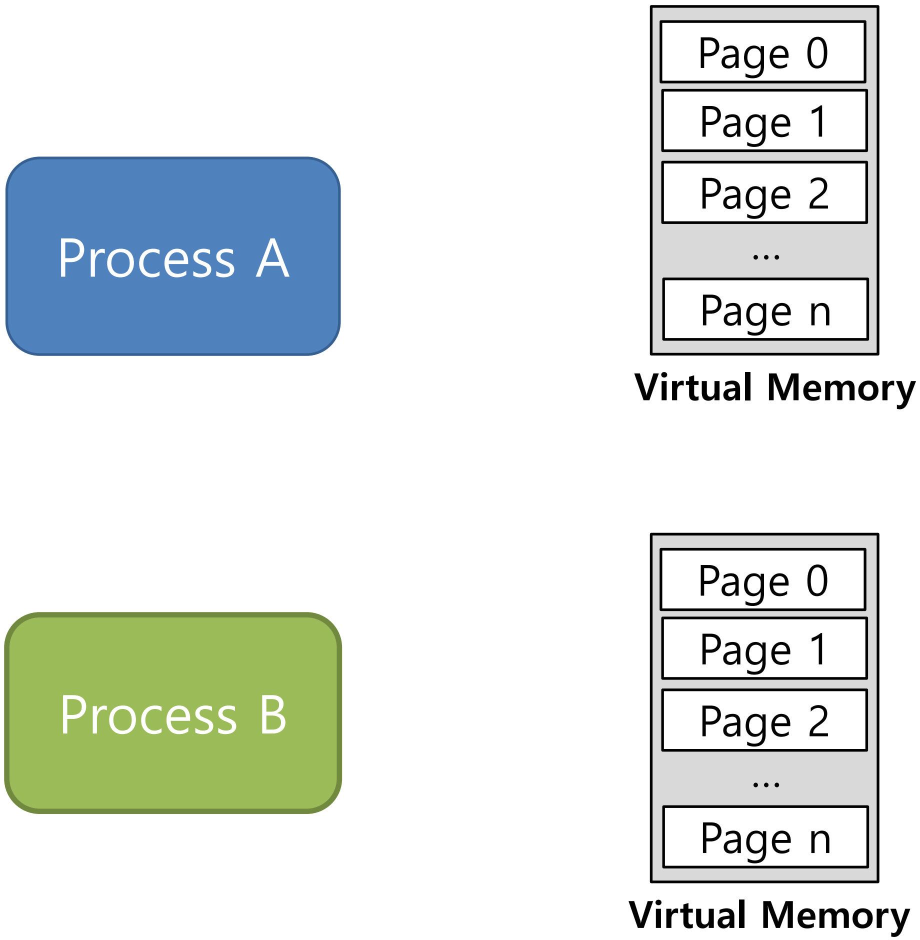


Insert TLB Entry

TLB Table

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

TLB Issue: Context Switching

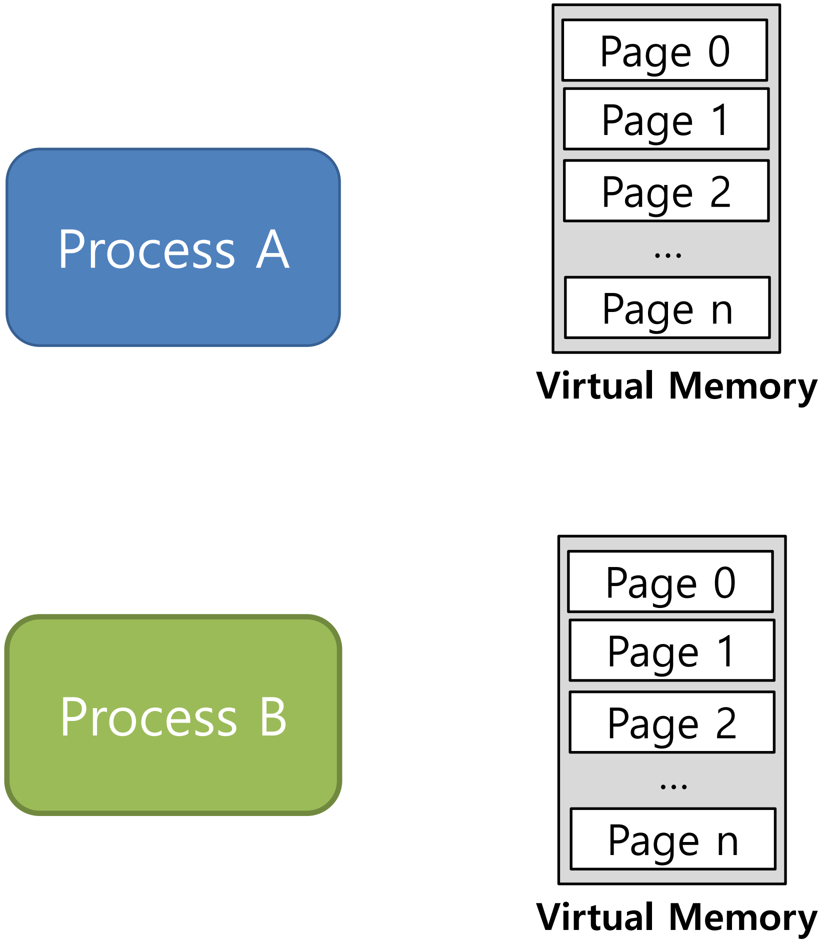


TLB Table

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

Can't Distinguish which entry is meant for which process

- 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

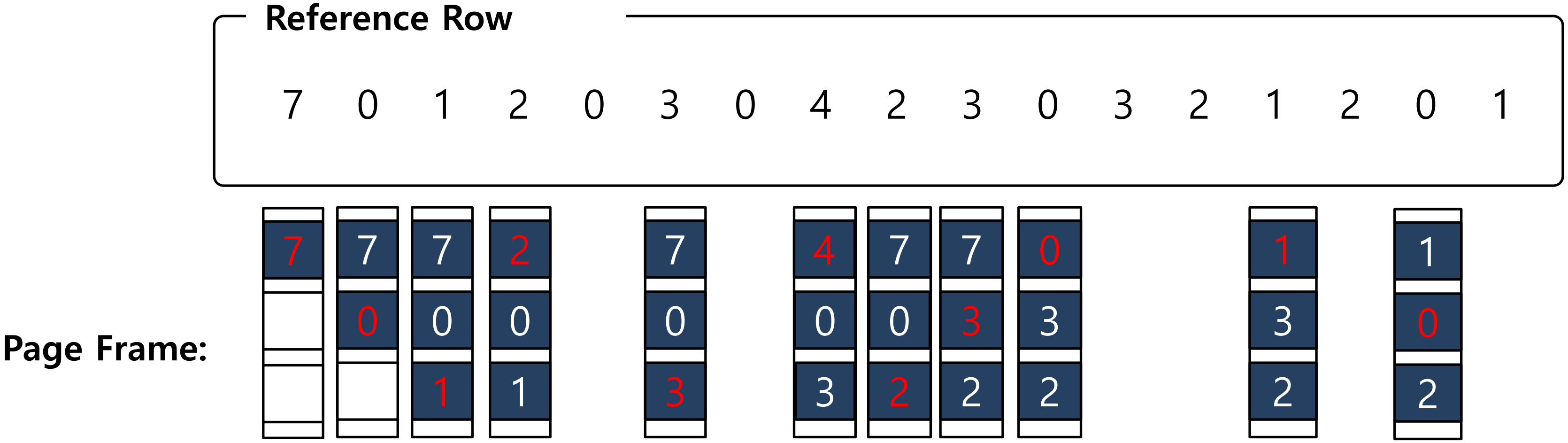
- Two processes share a page.
  - Process 1 is sharing physical page 101 with Process2.
  - P1 maps this page into the 10^{th} page of its address space.
  - P2 maps this page to the 50^{th} page of its address space.

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

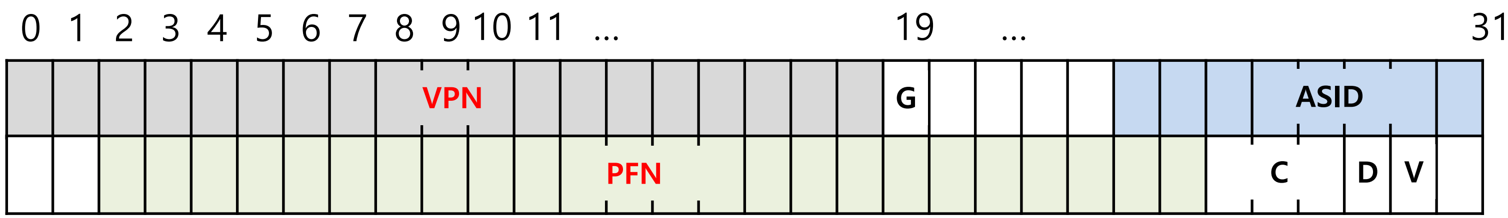
- LRU(Least Recently Used)
  - Evict an entry that has not recently been used.
  - Take advantage of *locality* in the memory-reference stream.



Total 11 TLB miss

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 (ignore ASID).
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.

TLB today

- Applications with very large memory footprint might be heavily impacted by TLB size (i.e. the locality of the code/data is too large)
  - Data/Instruction Separate TLBs
  - Multiple level TLB
  - Mega pages
  - No TLB → no pages for a fraction of the address space

- TLB is an itchy issue for virtualization
  - Complex hardware support required to minimize performance overheads
  - Unikernels

Recommended Reading (beyond class)

- Unikernels
  - http://unikernel.org

- Specialized single-address-space OS
  - One application
  - Convert the OS back in a set of library routines
  - Designed to be run in virtual environments

- Improved security and performance over conventional OS
  - Smaller attack surface
  - No virtual memory overhead
  - No VMM redirections in syscalls

- Disclaimer: Disclaimer: This lecture slide set is used in AOS course at University of Cantabria. Was initially developed for Operating System course in Computer Science Dept. at Hanyang University. This lecture slide set is for OSTEP book written by Remzi and Andrea Arpaci-Dusseau (at University of Wisconsin)