Universidade Estadual de Campinas
Instituto de Computação

MC504 Sistemas Operacionais

T12 Translation Lookaside Buffers

Referência principal
Ch.19 of *Operating Systems: Three Easy Pieces* by Remzi and Andrea Arpaci-Dusseau (pages.cs.wisc.edu/~remzi/OSTEP/)

Discutido em classe em 05 de setembro de 2018

Arthur João Catto, PhD

2º semestre de 2018

Background

- The requirement that instructions must be in physical memory to be executed may limit the size of a program to the size of main memory.
- In many cases, parts of the program are not needed or, at least, not at the same time, e.g.
  - Code to handle unusual error conditions.
  - Data structures that are dimensioned in excess of actual need.
  - Rarely used options and features.

Background

- The ability to execute a program which is only partially loaded in memory has many benefits
  - The size of the program would no longer be constrained by the physical memory that is available.
  - Since programs would require less memory, the degree of multiprogramming could be increased.
  - Loading or swapping user programs into memory would require less I/O.
- Being able to run a program that is not entirely in memory would benefit both user and system.

Background

- Virtual memory allows us to program using virtual addresses that will only be translated to physical addresses when accessed.
- To implement the virtual memory model, we can view a program as a collection of segments, e.g. user code, stack, heap, libraries, etc.
  - On their turn, each segment will be mapped onto a number of pages.

Background

- Additional benefits of virtual memory are
  - System libraries can be shared by several processes through mapping of the shared object onto a virtual address space.
  - Processes can share part of their virtual address spaces, by mapping them onto actually shared memory pages.
  - Parent and child processes may share pages, thus speeding up process creation.

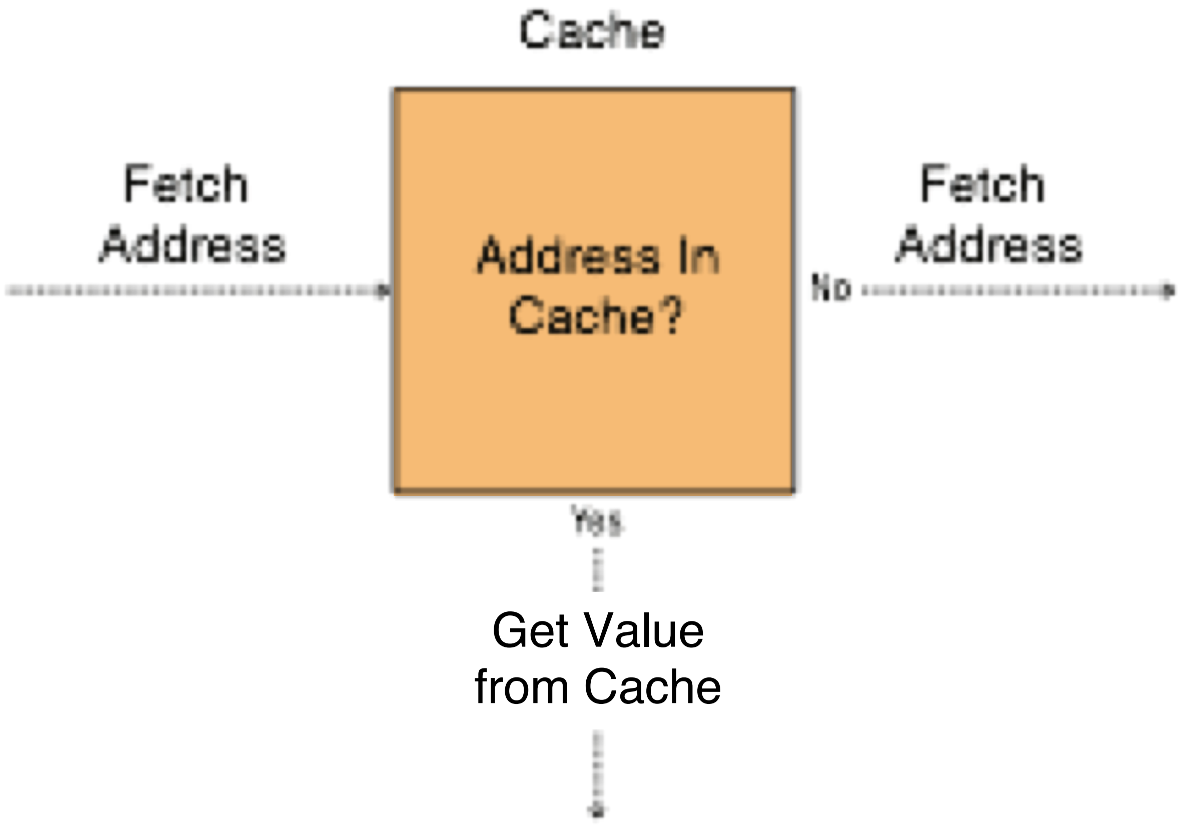
Background

- To enable access to a virtual address during execution, the page that contains it must be loaded in a frame in physical memory.
- Since the number of pages in a program may exceed the number of frames in the target machine's memory, pages may have to be loaded and unloaded on-the-fly, as execution progresses.
- To avoid the slowdown caused by dynamically loading and unloading pages, due to references to addresses that are not present in memory when needed, we make extensive use of caches.

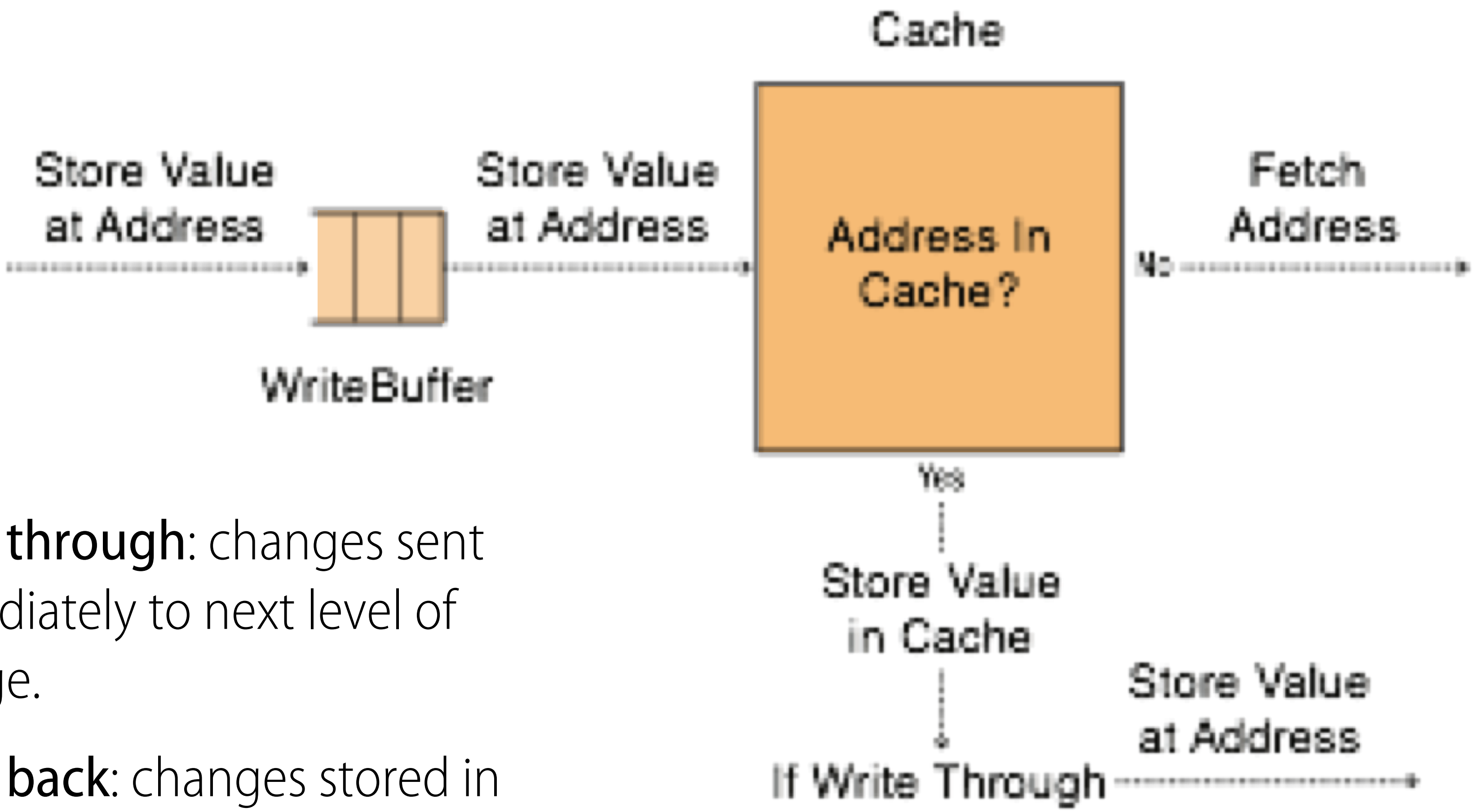
What is a cache?

- A cache is a copy of data or code that is faster to access than the original.
  - Cache memory is allocated in units called *blocks*, comprising multiple memory locations.
  - The result of a search for an address in a cache will be
    - a *hit*, if the cache has a copy of that address, or
    - a *miss*, if the cache does not have a copy of that address.
- Cache efficiency is measured by its *hit rate* and is heavily dependent on
  - Temporal locality
    - Programs tend to reference the same memory locations multiple times, e.g. by executing instructions in a loop.
  - Spatial locality
    - Programs tend to reference nearby locations, e.g. examining array data in a loop

Cache Concept (Read)



Cache Concept (Write)



Write through: changes sent immediately to next level of storage.

Write back: changes stored in cache until cache block is replaced.

Memory hierarchy example

1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 µs	100 TB
Local non-volatile memory	100 µs	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 EB

i7 has 8MB as shared 3rd level cache; 2nd level cache is per-core

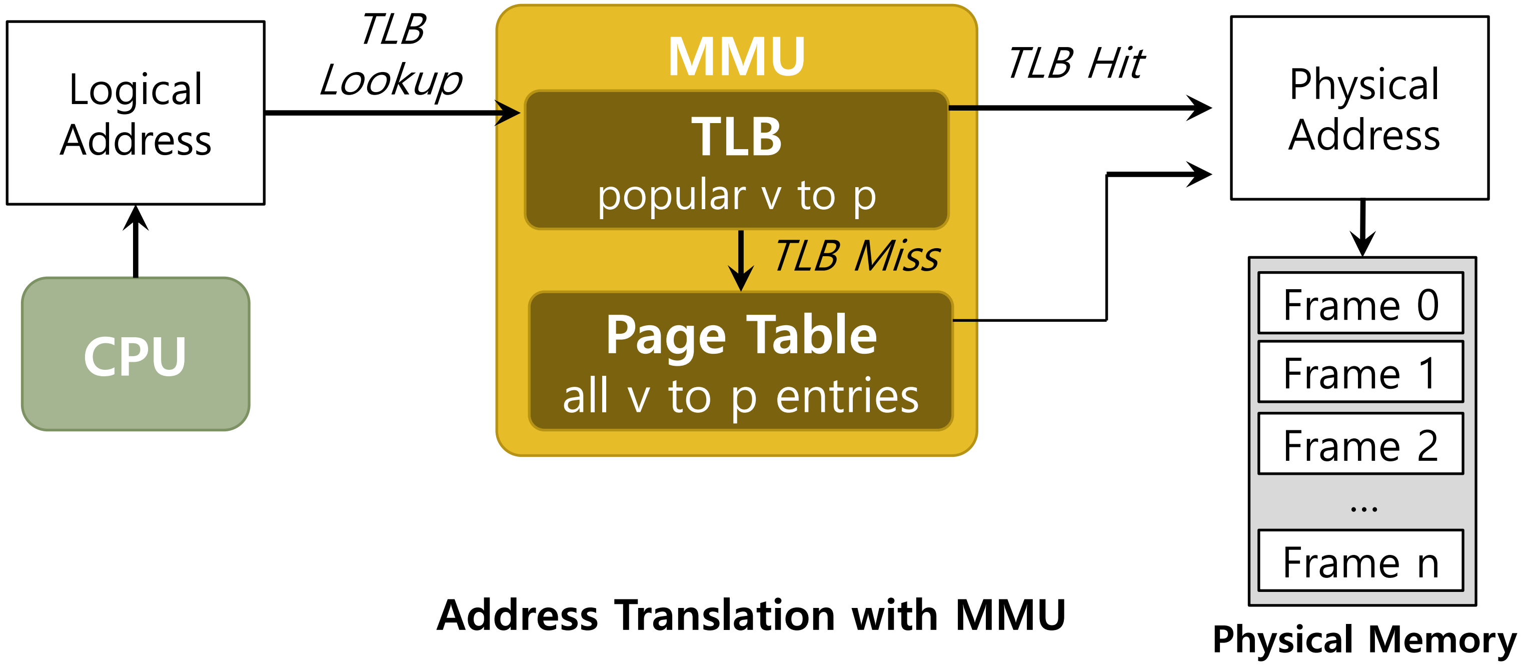
Memory hierarchy example

Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 μs	100 TB
Local non-volatile memory	100 μs	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 EB

i7 has 8MB as shared 3rd level cache; 2nd level cache is per-core

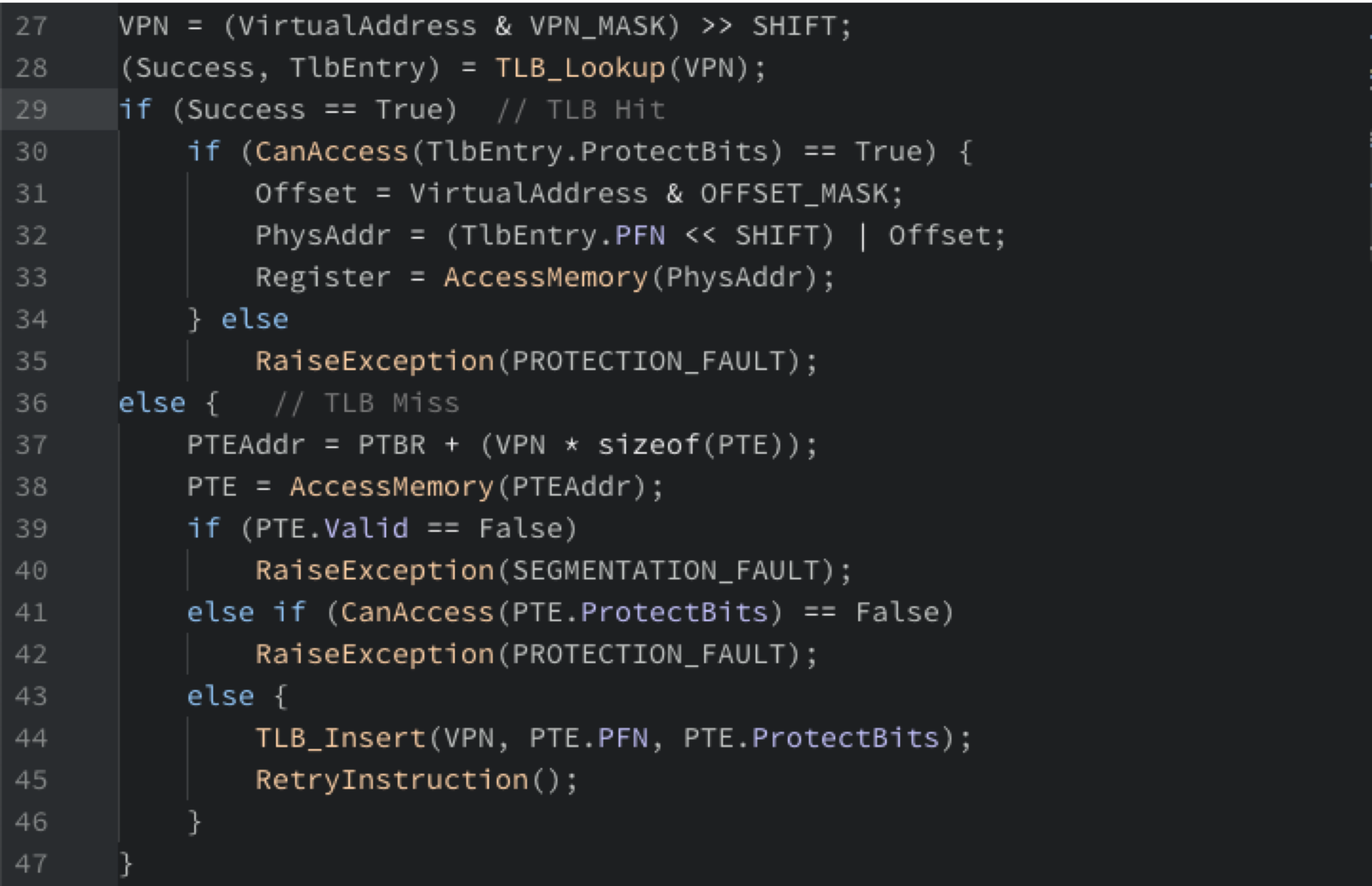
TLB

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



Address Translation with MMU

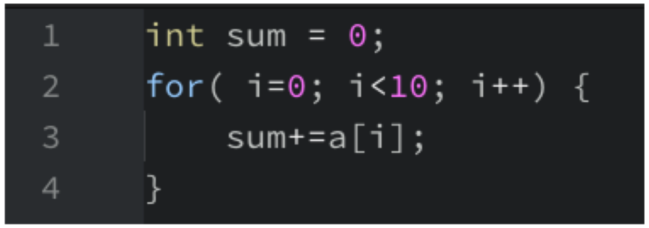
TLB Basic Algorithms



```
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();
    }
}
```

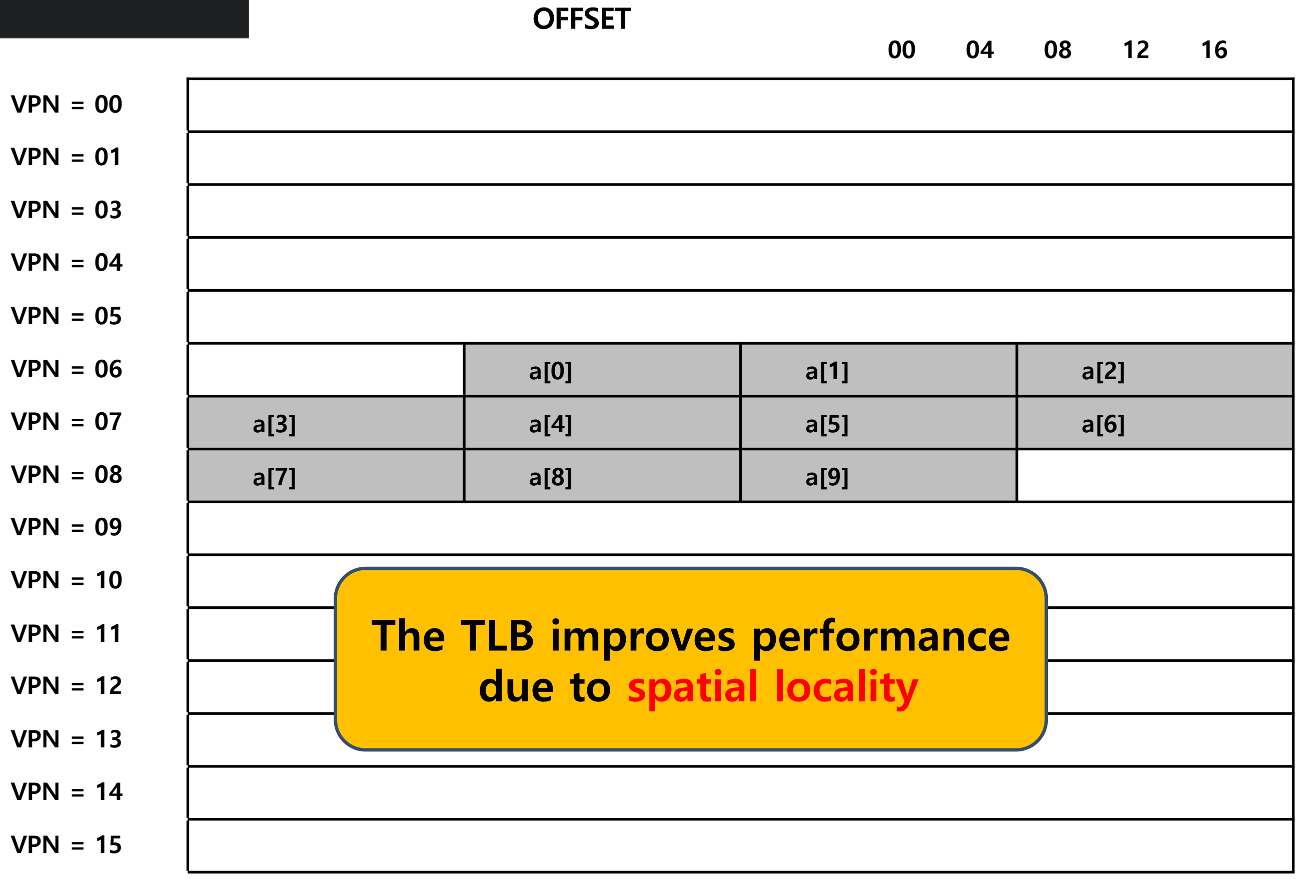
Example: Accessing An Array

- How a TLB improves paging performance…



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

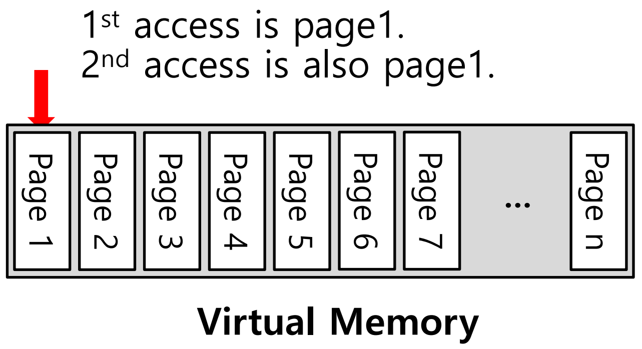
3 misses and 7 hits.
Thus TLB hit rate is 70%.



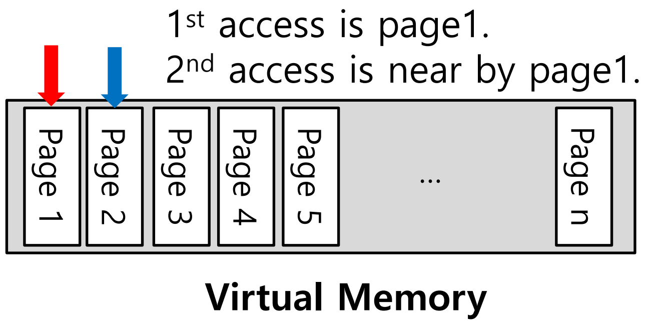
The TLB improves performance due to spatial locality

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`**.



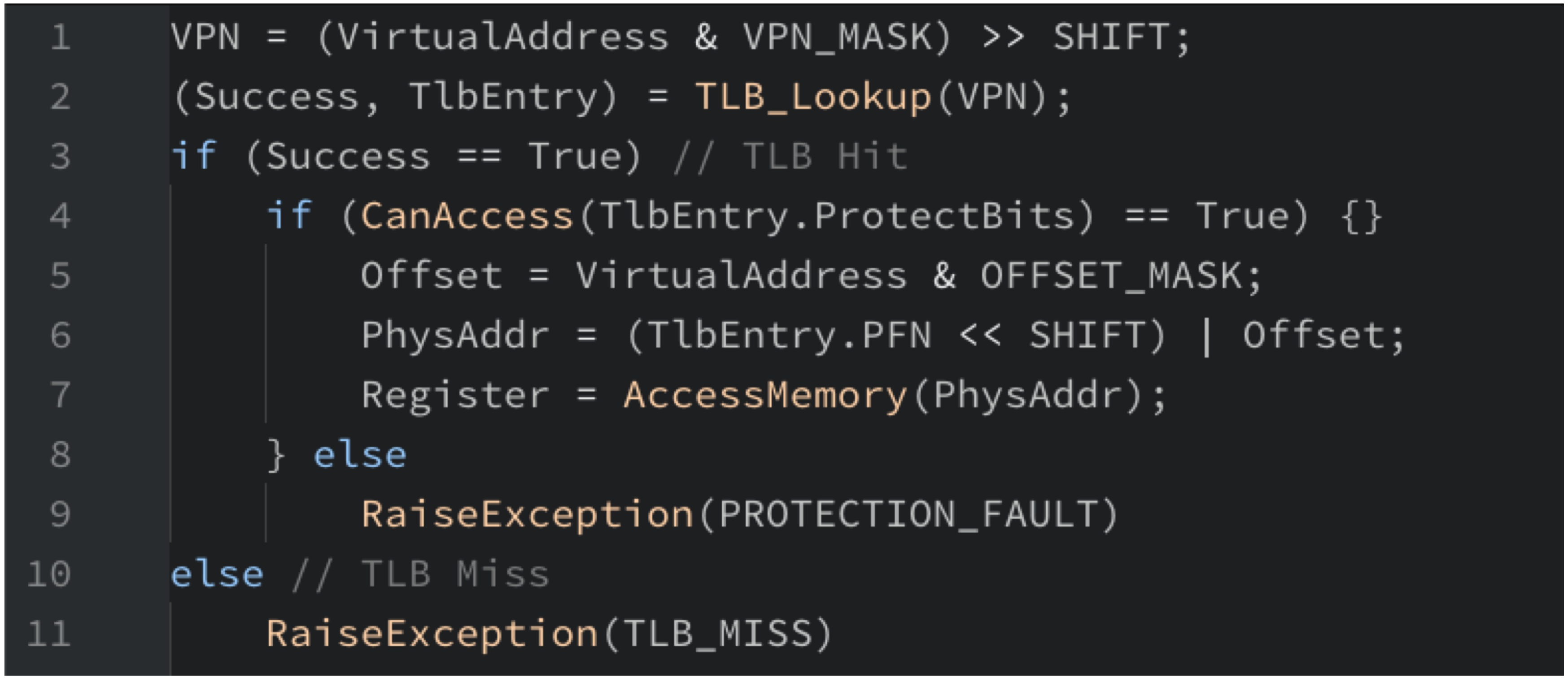
Who Handles The TLB Miss?

- 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 the instruction.
  - This is called 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.

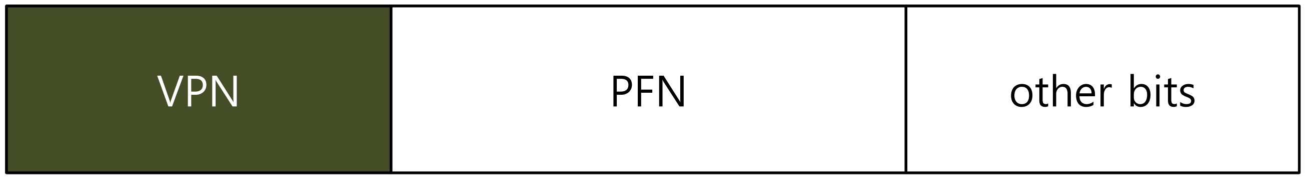
TLB Control Flow algorithm (OS Handled)



```
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)
```

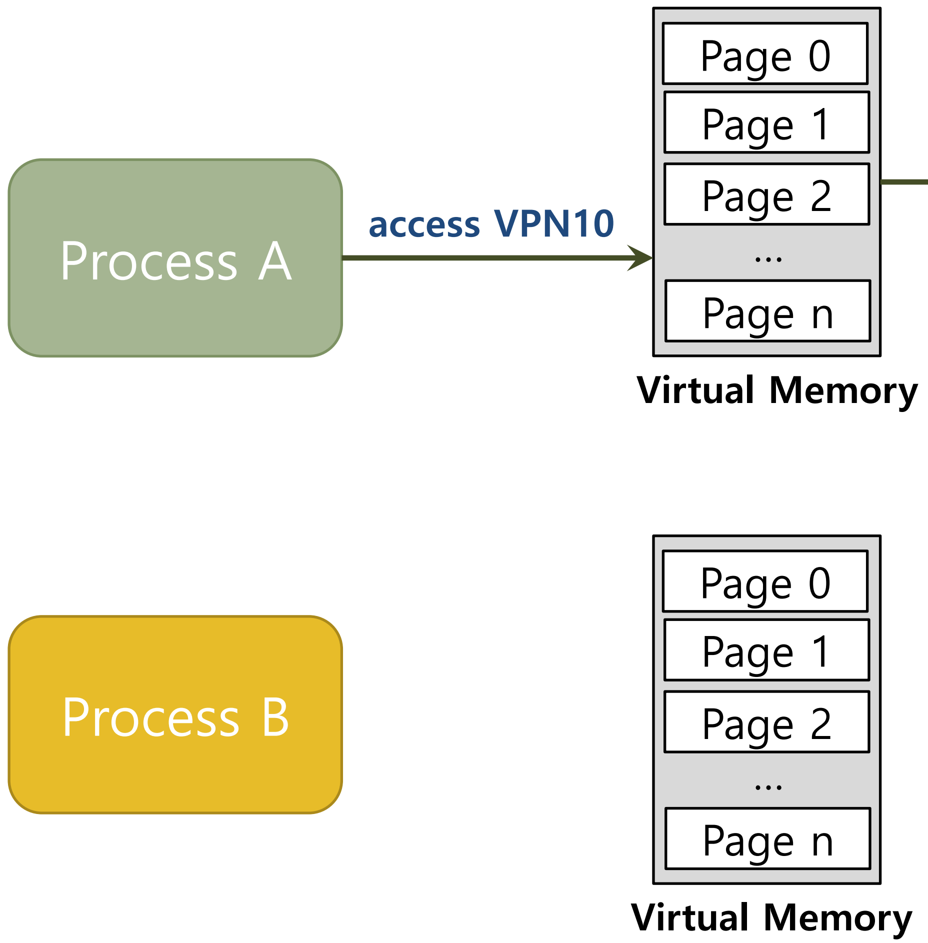
TLB entry

- TLB is managed by Full Associative method.
  - A typical TLB might have 32, 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

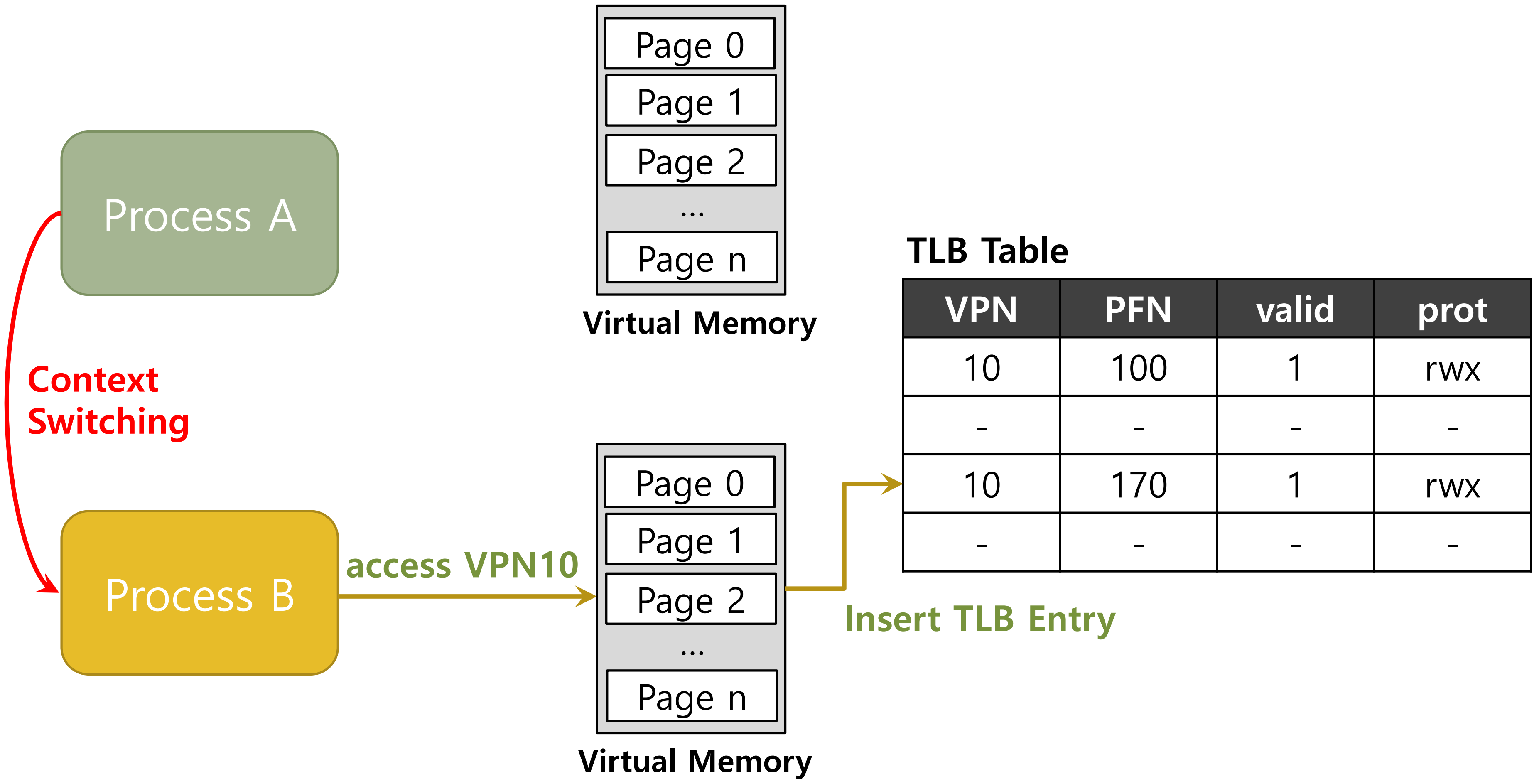
TLB Issue: Context Switching



TLB Table

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

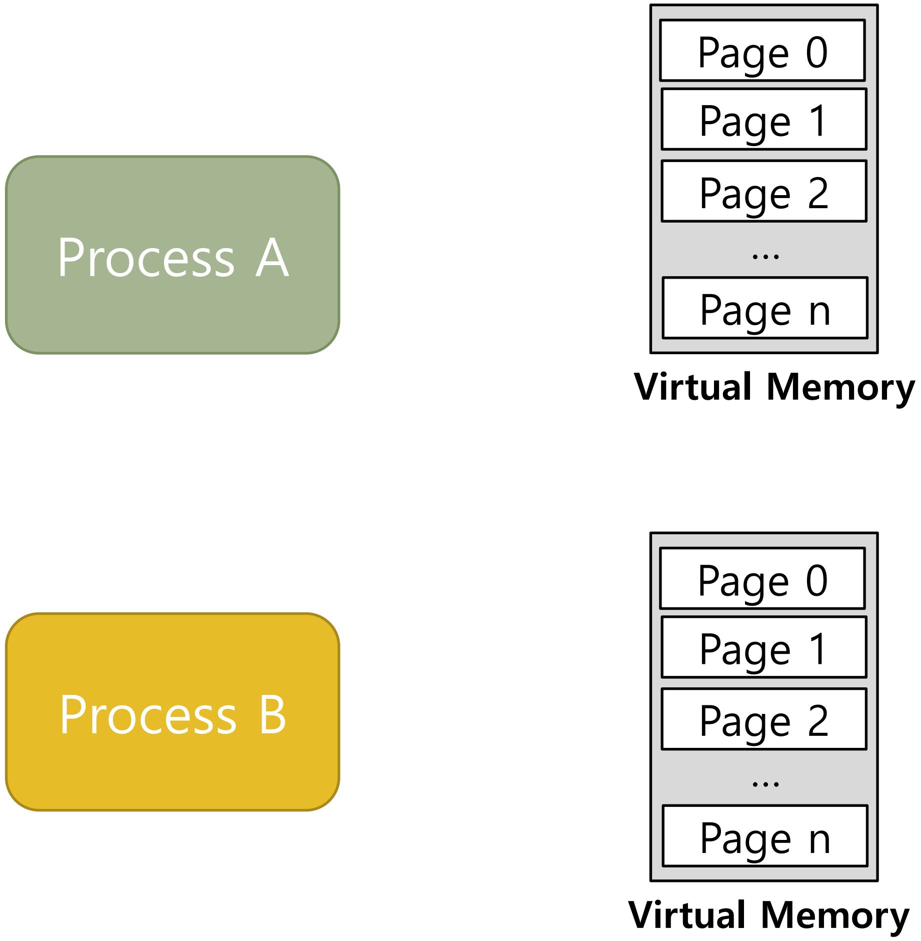
TLB Issue: Context Switching



TLB Table

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

TLB Issue: Context Switching



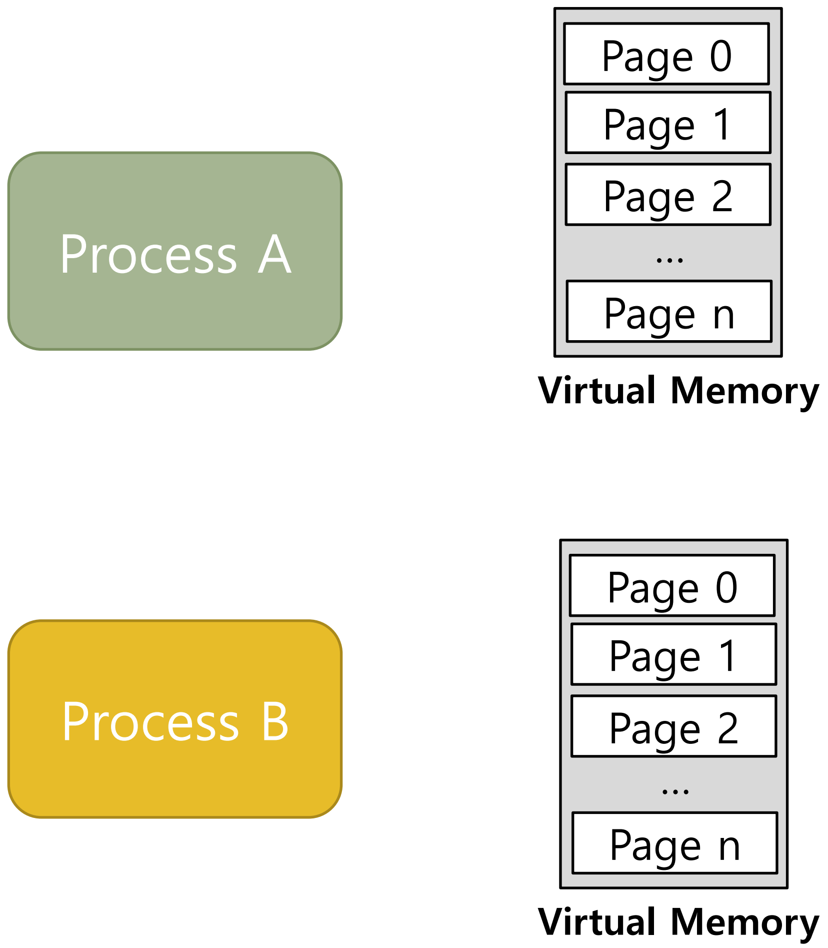
TLB Table

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

Can't Distinguish which entry is meant for which process

To Solve Problem

- 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

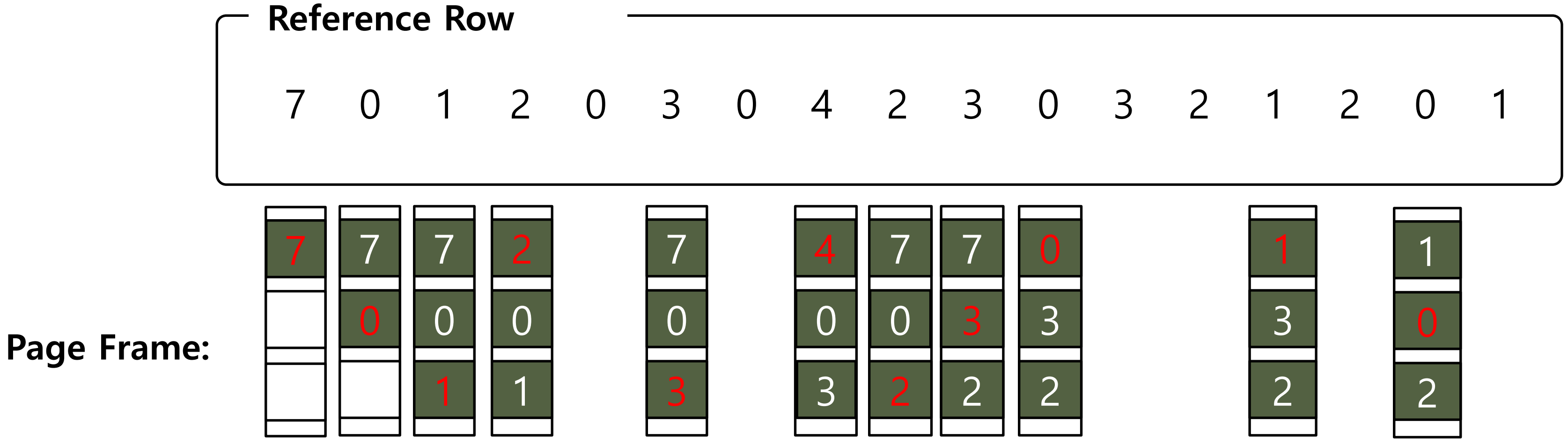
- Two processes share a page.
  - Process 1 is sharing physical page 101 with Process2.
  - P1 maps this page into the 10th page of its address space.
  - P2 maps this page to the 50th page of its address space.

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.



Total 11 TLB misses