# Multi-Processor Systems

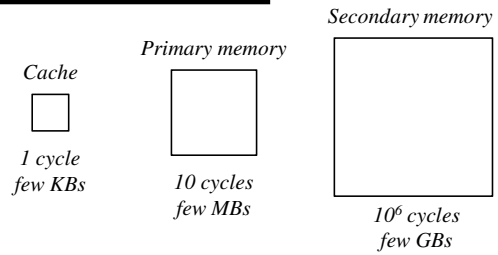
*Pirouz Bazargan Sabet*

Sorbonne Université - LIP6

Pirouz.Bazargan-Sabet@lip6.fr



## Memory Hierarchy




- The organization of the different levels in the hierarchy is always based on the same principles
- The Primary Memory can be seen as a ***cache*** regarding the Secondary Memory

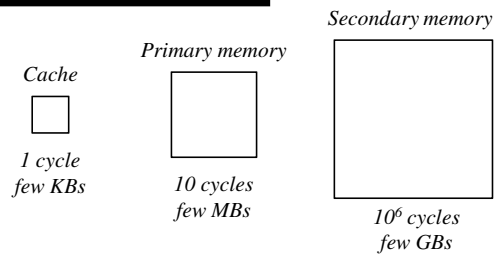


## Outline

- Memory Hierarchy
- Virtual Memory

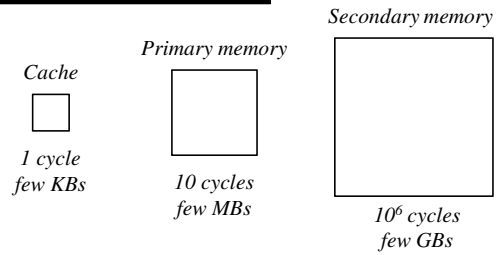


## Memory Hierarchy




- The exploitation of Spatial and Temporal localities allow to reduce drastically the Miss rate in the cache ($< 10\%$)
- What Miss rate should we expect for the Primary Memory ? $\simeq 10^{-5}$

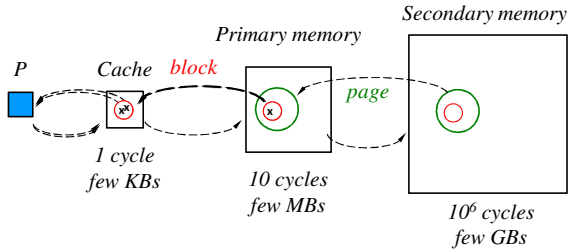
## Memory Hierarchy




- How such a Miss rate can be reached ?
  - Spatial locality
  - Temporal locality



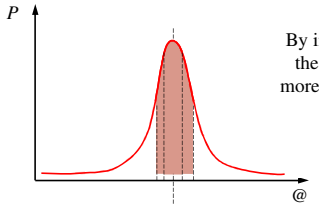
## Memory Hierarchy






## Memory Hierarchy

Spatial locality




By increasing the size of the block, we can take more advantage of Spatial locality



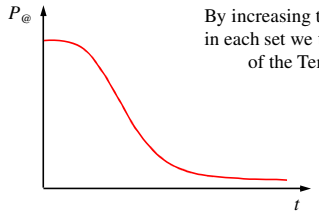
## Memory Hierarchy

Spatial locality

| architecture | Page size |
|---|---|
| IBM 370 | 4 KBytes |
| VAX | 1/2 KBytes |
| IBM 400 | 1/2 KBytes |
| DecAlpha | 8 KBytes |
| Mips | 4 KBytes to 16 MBytes |
| UltraSparc | 8 KBytes to 4 MBytes |
| Pentium | 8 KBytes to 4 MBytes |
| Power | 4 KBytes |
| Itanium | 4 KBytes to 256 MBytes |

## Memory Hierarchy

Temporal locality




By increasing the number of places in each set we take more advantage of the Temporal locality



## Memory Hierarchy

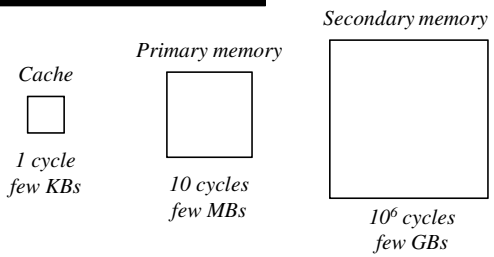
Write strategy ?

- *Write Through*
- *Write Back*
  - Mean # cycles to access the SM ($\simeq 10^6$)
  - A big page size means a high probability to have multiple writes in the same page

→ *Write Back*



## Memory Hierarchy

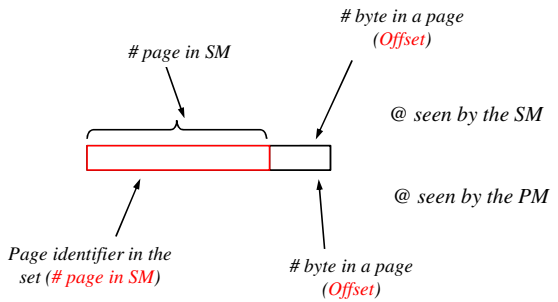



- How such a Miss rate can be reached ?
  - Spatial locality → *page size : 4 KBs*
  - Temporal locality → *Full associative*



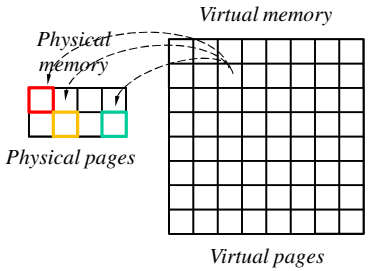
## Memory Hierarchy

Primary Memory
Secondary Memory



## Memory Hierarchy

Full Associative



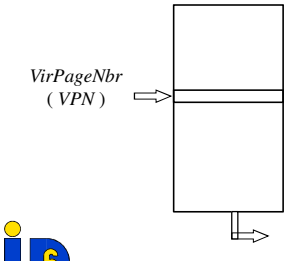



## Virtual Memory

Finding the Physical Page Number (PPN) knowing the Virtual Page Number (PPN) ?

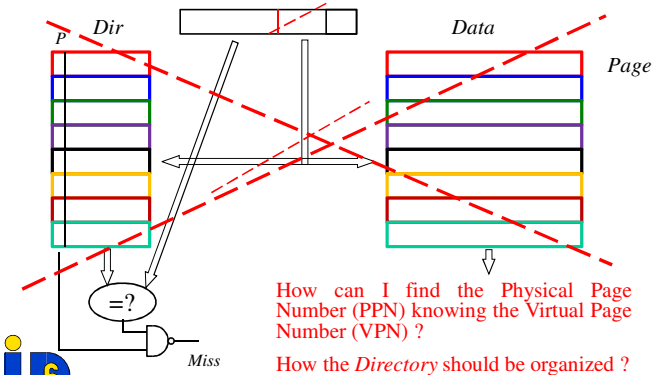
I need to set up a function *f* such as : $f(VirPageNbr) = PhyPageNbr$

@ Translation






## Virtual Memory




How can I find the Physical Page Number (PPN) knowing the Virtual Page Number (VPN) ?

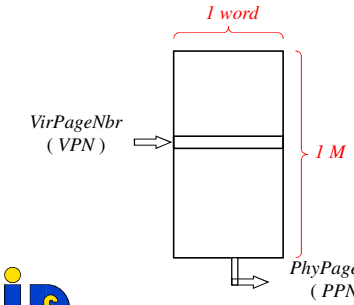
How the *Directory* should be organized ?



## Virtual Memory

*Page Table*

Page = 4 KBytes
VM = 4 GBytes
1 M Virtual pages
PM = 1 GBytes
256 K Physical pages
*PPN* : 18 bits
*+ P, LRU, D, RWX, etc*
32 bits




*Size of Page table : 4 MB*
*... per process*
*... regardless the size of physical memory*

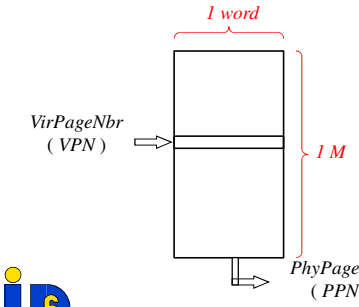
## Virtual Memory

*Page Table*

Page = 4 KBytes
VM = 4 GBytes
1 M Virtual pages
PM = 1 GBytes
256 K Physical pages
*PPN* : 18 bits
*+ P, LRU, D, RWX, etc*
32 bits

*1/4 Present bit = 1*

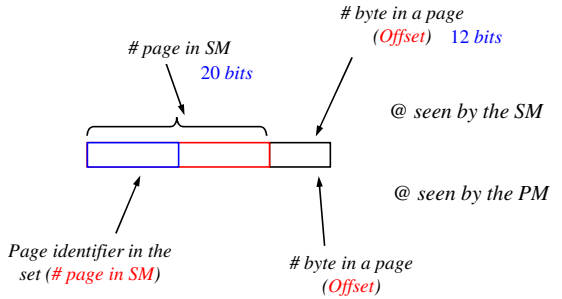
*The major part of Present bits are 0 (the Page Table is almost empty)*






## Memory Hierarchy

Primary Memory
Secondary Memory

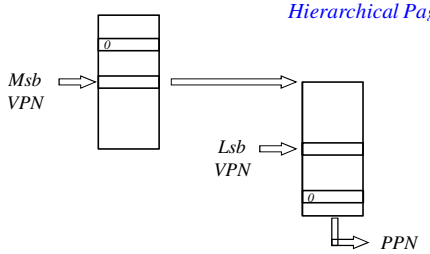





## Virtual Memory

Finding the Physical Page Number (PPN) knowing the Virtual Page Number (PPN) ?

*Hierarchical Page Table*

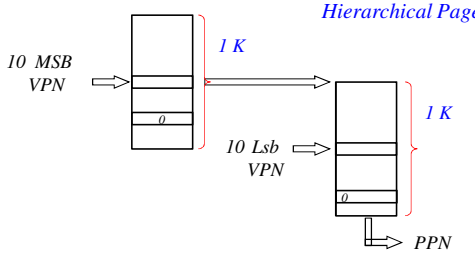





## Virtual Memory

Finding the Physical Page Number (PPN) knowing the Virtual Page Number (PPN) ?

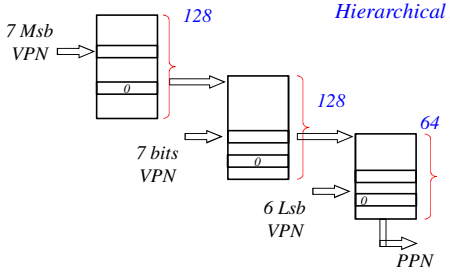
*Hierarchical Page Table*



## Virtual Memory

Finding the Physical Page Number (PPN) knowing the Virtual Page Number (PPN) ?

*Hierarchical Page Table*



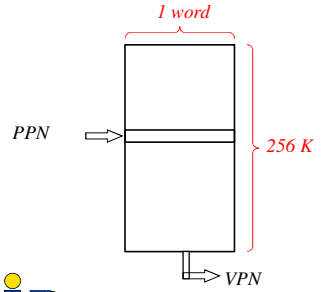

- An @ translation requires 2 or 3 memory accesses
- 1 table per process
- The space required for the table does not depend on the size of the PM



## Virtual Memory

*Inverted Page Table*

In the Page table there is only relevant information for those entries that correspond to the physical pages




*Do not represent those table entries that do not contain useful information*

*But the PPN is not known*

- *Scan the Table seeking for the VPN*
- *The index of the entry would be the PPN*
- *Translation latency*

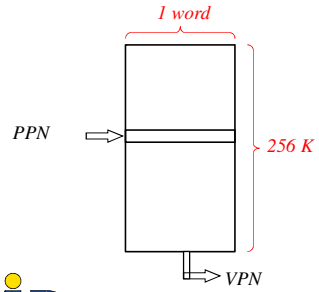
*Associative Array*



## Virtual Memory

*Inverted Page Table*

In the Page table there is only relevant information for those entries that correspond to the physical pages




*Do not represent those table entries that do not contain useful information*

Page = 4 KBytes
MV = 4 GBytes
MP = 1 GBytes
256 K Physical Pages



## Virtual Memory

*Inverted Page Table*

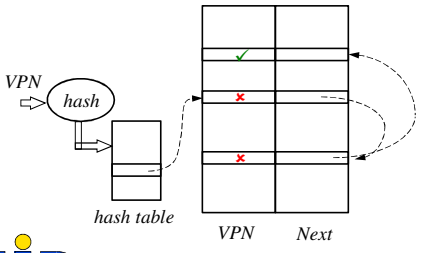
*Associative Array* : search by content

- Split the table into a set of lists
- Each list element contains a VPN and the pointer (the index) of the next element
- The elements of a list are equivalent regarding an equivalence relation (of VPN)
- The equivalence relation is a *Hash Function* thet makes a uniform repratition of VPNs

## Virtual Memory

*Inverted Page Table*

*Enhanced Associative Array* : table of lists

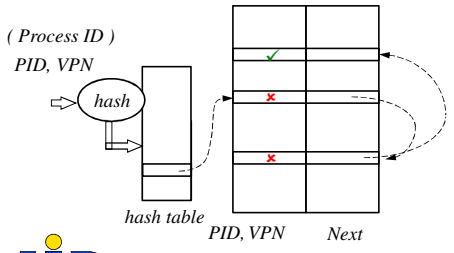





## Virtual Memory

*Inverted Page Table*

*Enhanced Associative Array* : table of lists



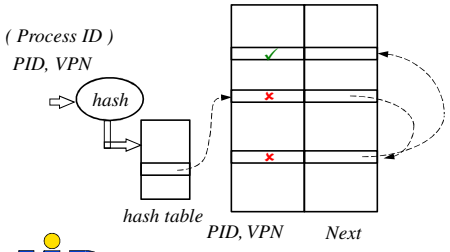

- The search latency depends on the size of the *Hash Table* (space-time tradoff)
- Increasing the size of the *Hash Table* reduces the number of elements in each list, reducing the search latency



## Virtual Memory

*Inverted Page Table*

*Enhanced Associative Array* : table of lists




- Time costly translation
- A single Page Table for all the processes
- The space required for the table is proportional to the size of the PM



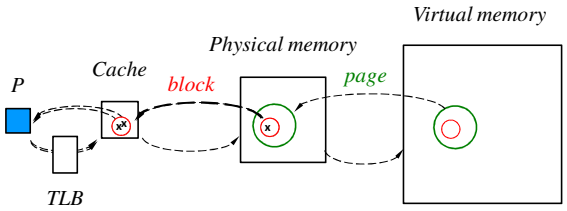
## Virtual Memory

- The *Page Table* has the same function as the *Directory*
- The address translation latency is time consuming regardless of the type of the table
- Untractable considering the requirement imposed by the processor : one access every cycle
- The address translation process should be accelerated

Introduce a hardware Cache of the Page Table :
*TLB ( Translation Lookaside Buffer )*

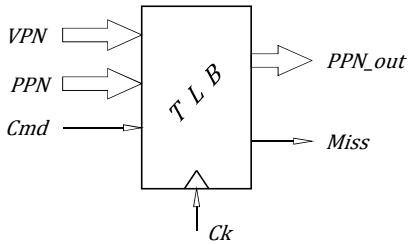
## Virtual Memory

- The *TLB* is a cache that saves the recent address translations (VPN to PPN)






## Virtual Memory




*Cmd : Write , Read , Remove, Invalidate*



## Virtual Memory

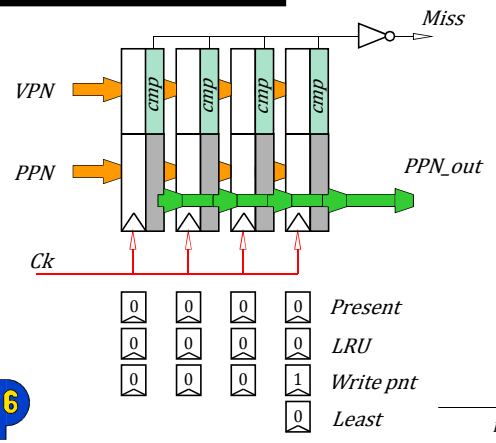
- The organization of the *TLB* is based on *Temporal locality*

  A (virtual) page recently used has a great probability to be accessed in near future

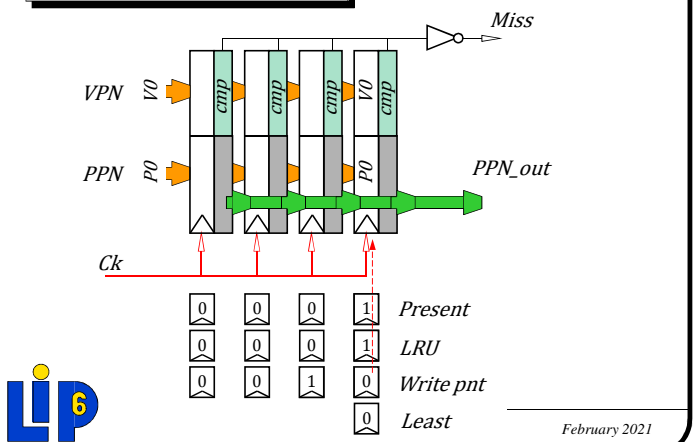
- A *TLB* is an *associative array* (search by content) that associates a *VPN* to its translation in the physical space, the *PPN*



## Virtual Memory

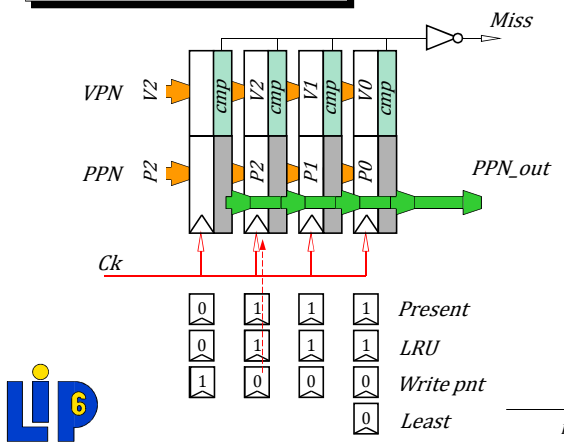


## Virtual Memory — TLB : Write



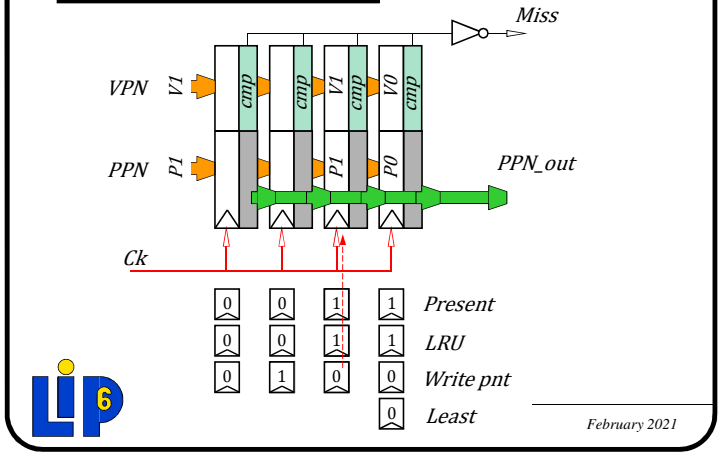

February 2021

## Virtual Memory — TLB : Write



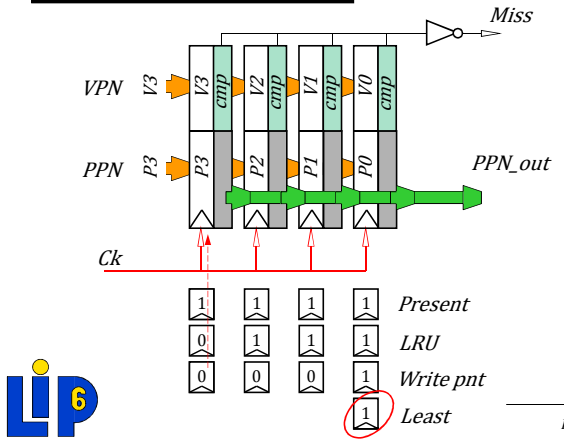

February 2021

## Virtual Memory — TLB : Write



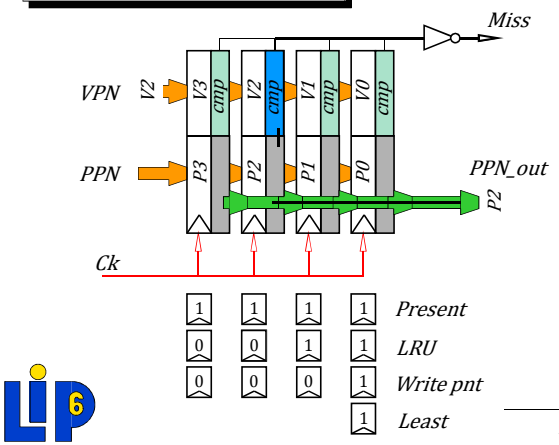

February 2021

## Virtual Memory — TLB : Write



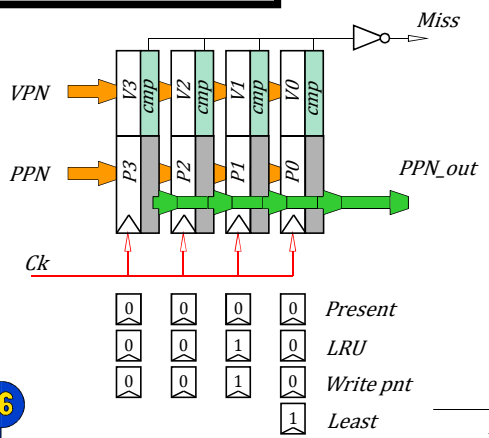

February 2021

## Virtual Memory — TLB : Read



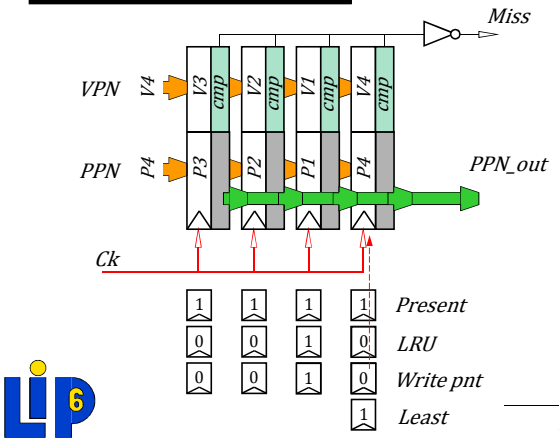

February 2021

## Virtual Memory — TLB : Invalidate



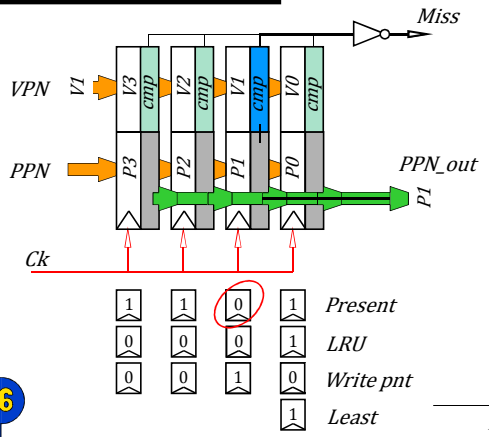

February 2021

## Virtual Memory — TLB : Write




February 2021

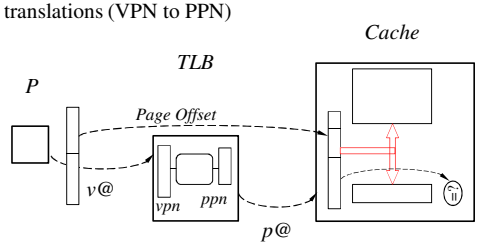
## Virtual Memory — TLB : Remove




February 2021

## Virtual Memory

- The *TLB* is a cache that saves the recent address translations (VPN to PPN)




Page Offset : 12 bits

restricts the size of the cache : $\#Set \times BlocSize$



## Virtual Memory

Read access