Memory Hierarchy

Virtual Memory, Address Translation

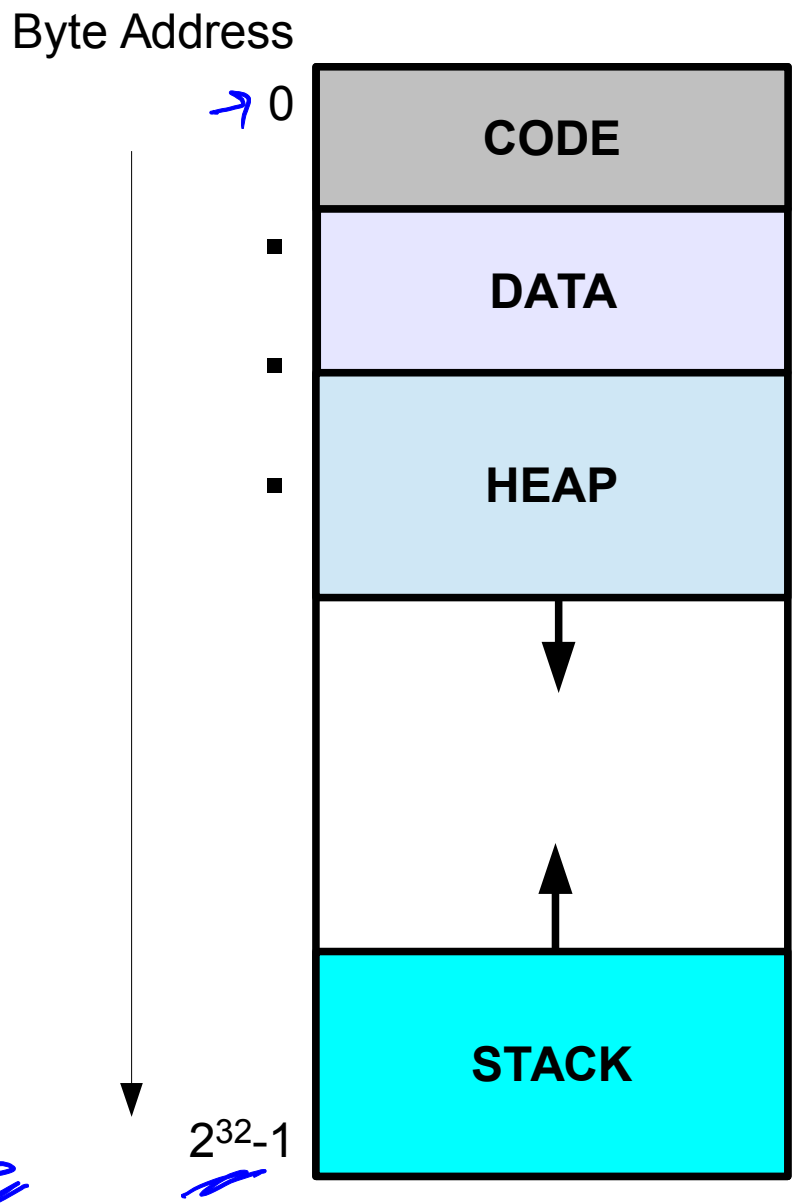
Slides contents from:

Hennessy & Patterson, 5ed. Appendix B and Chapter 2.

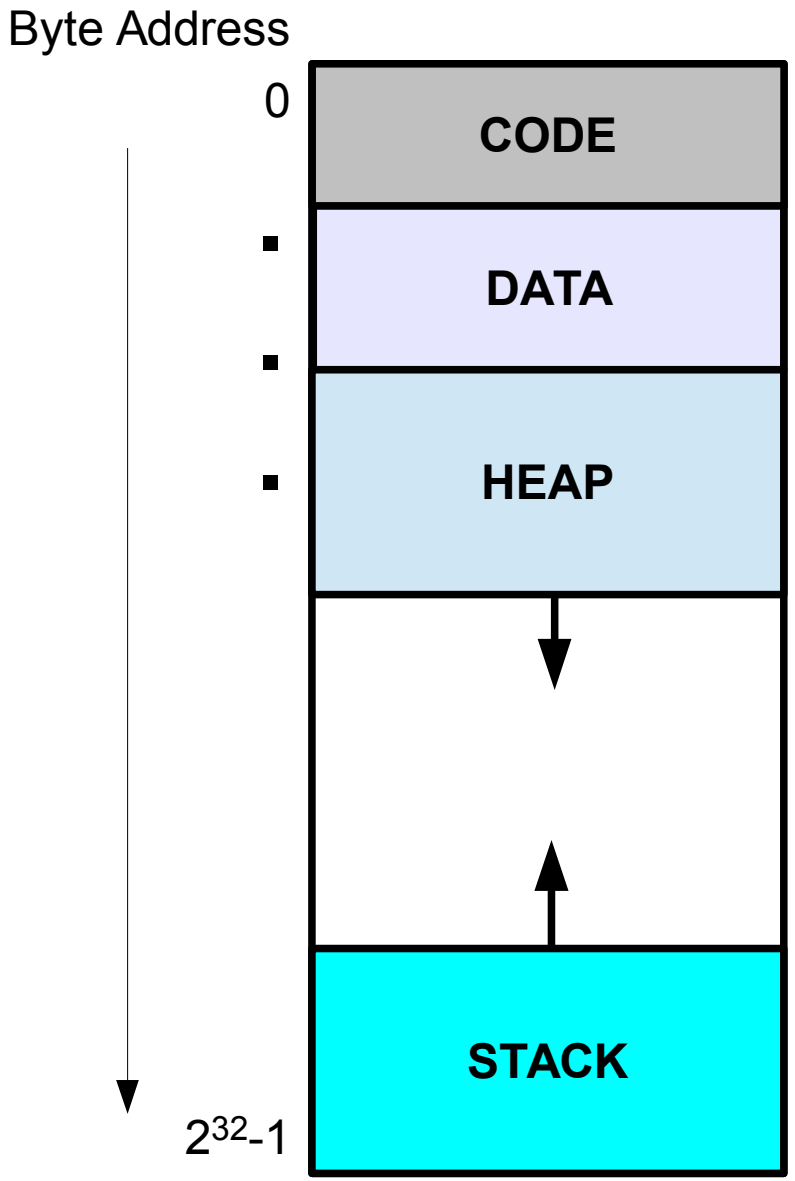
David Wentzlaff, ELE 475 – Computer Architecture.

MJT, High Performance Computing, NPTEL.

Process/Program Address Space

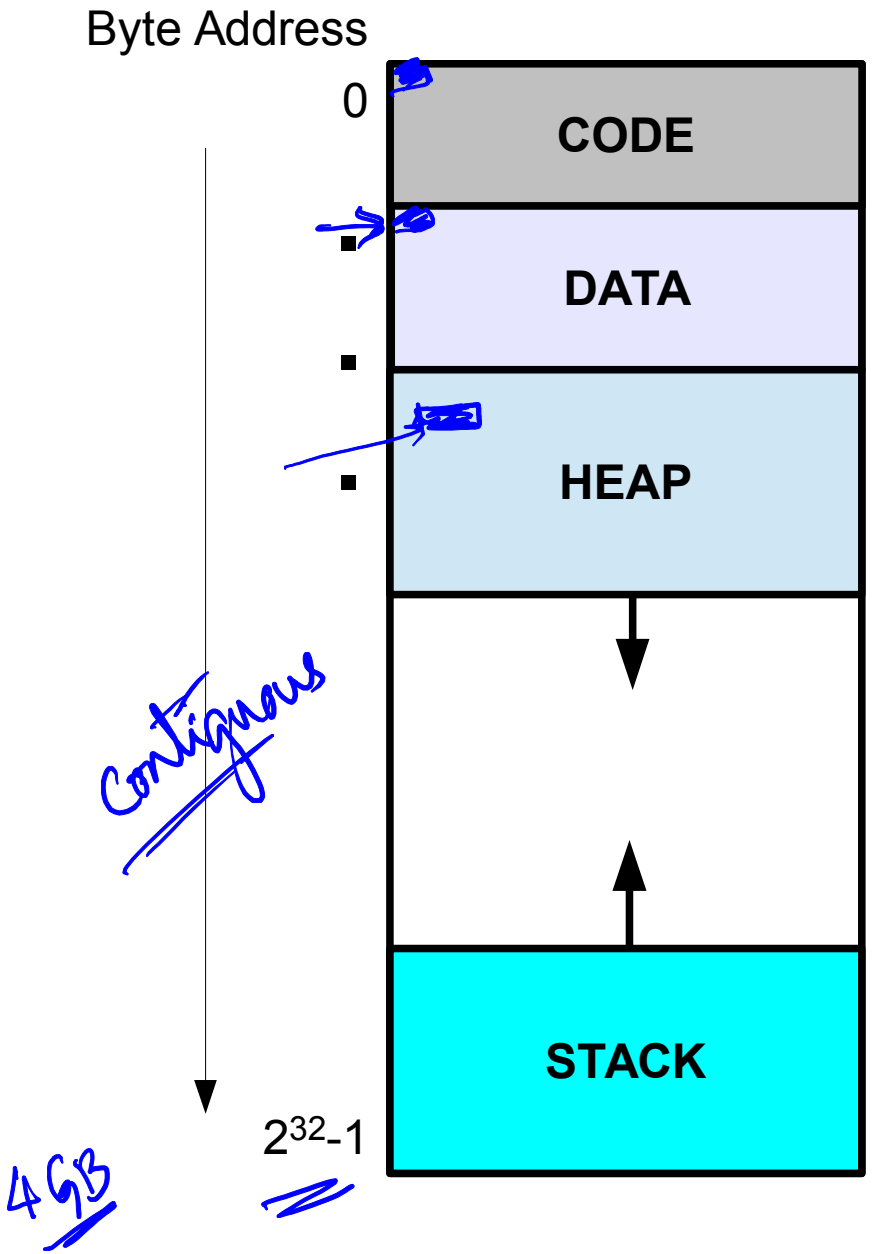


Process/Program Address Space



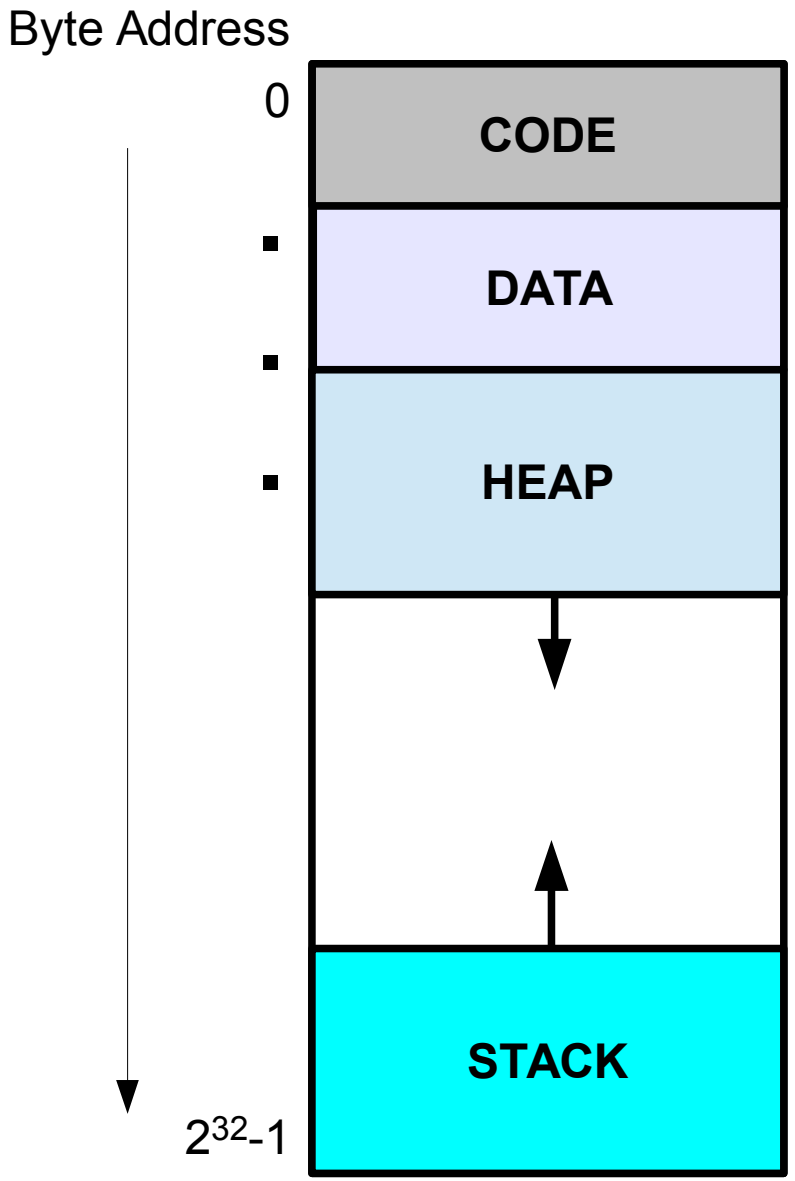
- Compiler assumes a linear address space
  - Byte 0 to Byte 2^{32}-1

Process/Program Address Space

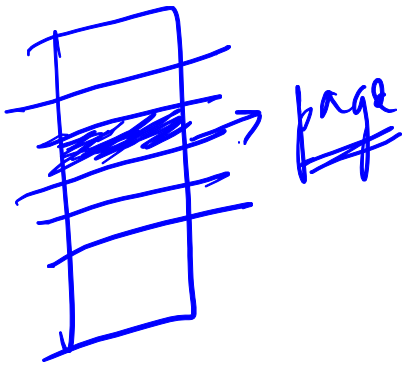


- Compiler assumes a linear address space
  - Byte 0 to Byte 2^{32}-1
- **Virtual Address space**

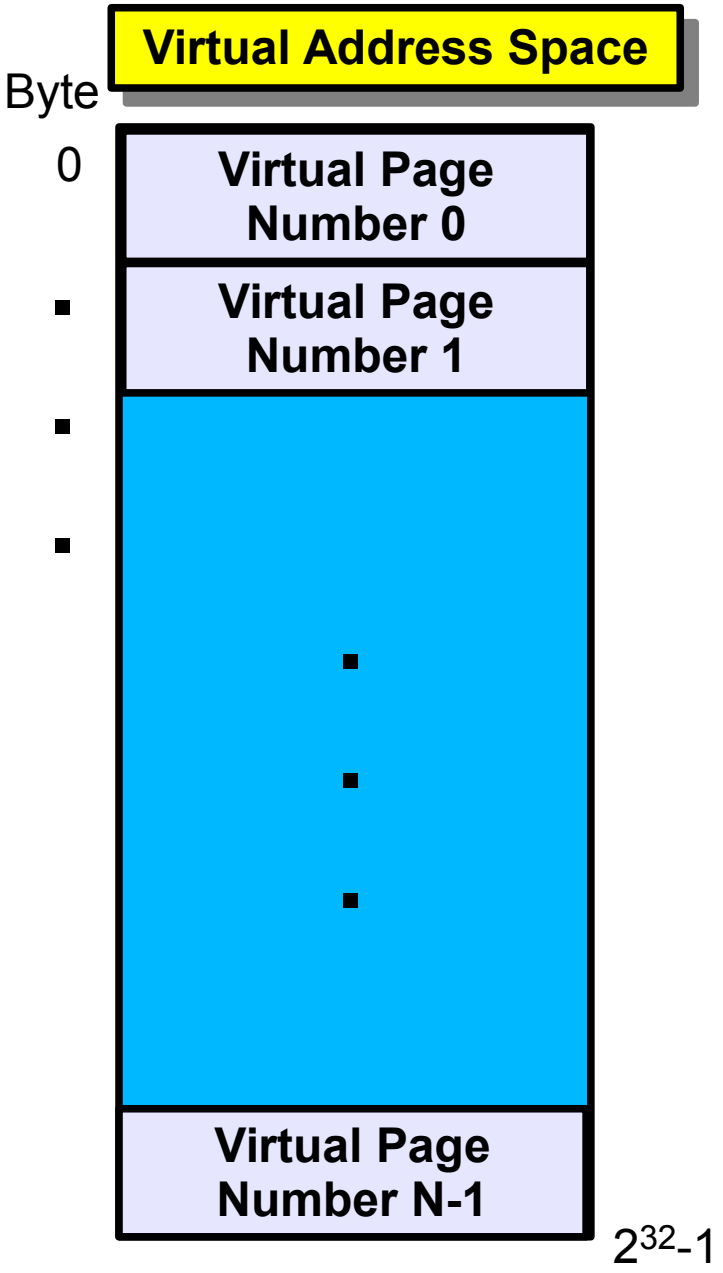
Process/Program Address Space



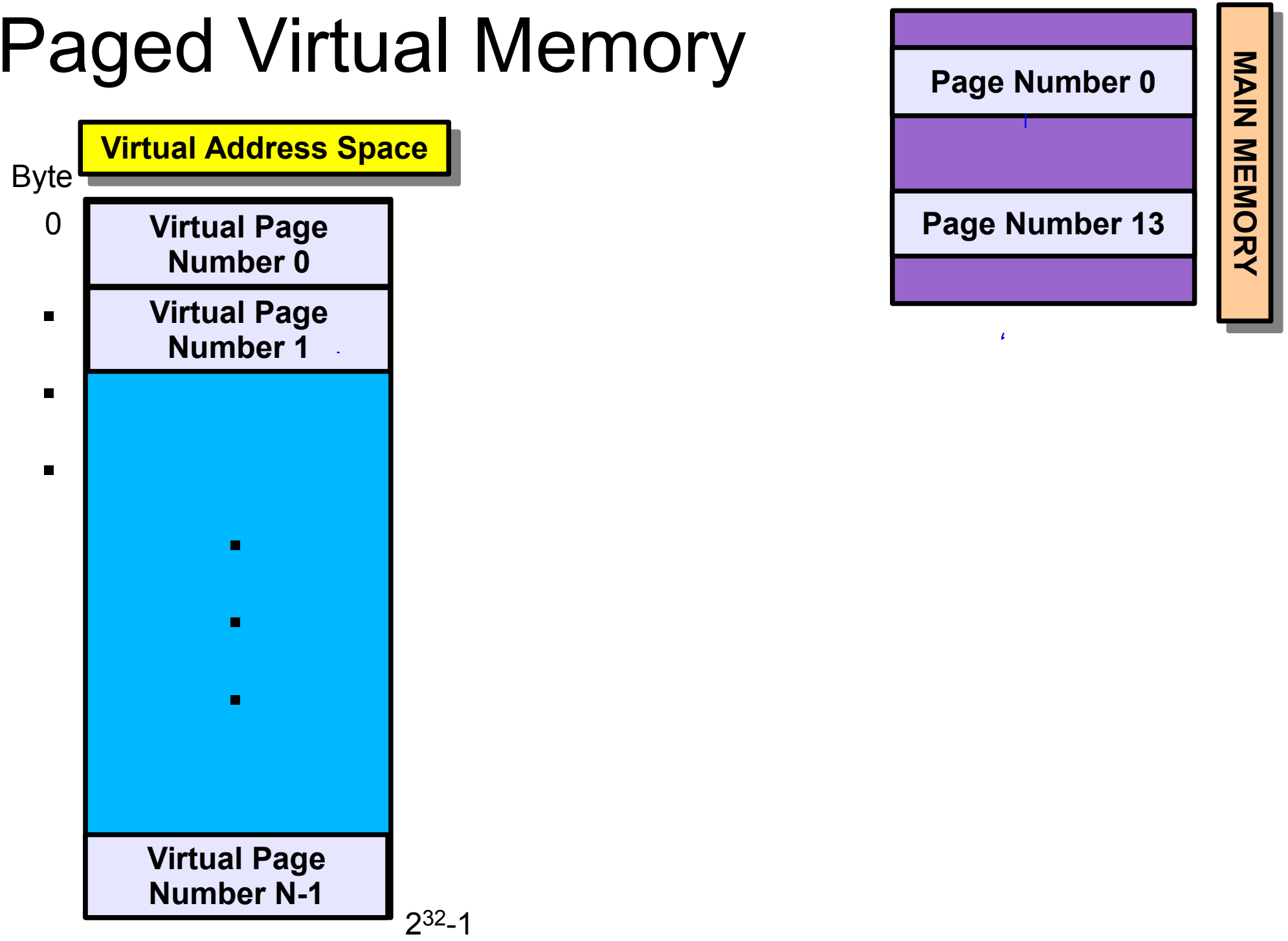
- Compiler assumes a linear address space
  - Byte 0 to Byte 2^{32}-1
- **Virtual Address space**
- The entire process data structure may not be present in MM at all times.



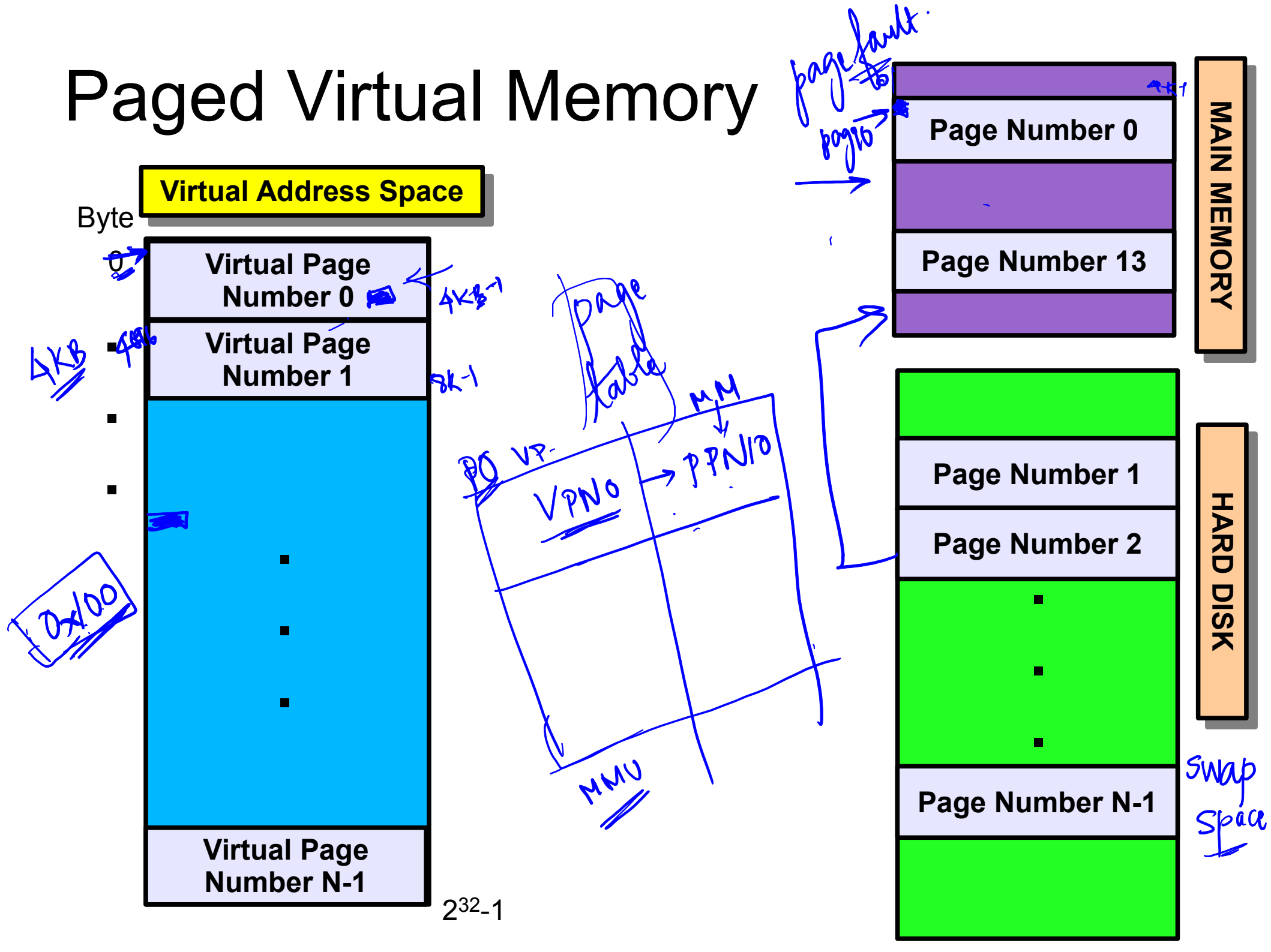
Paged Virtual Memory



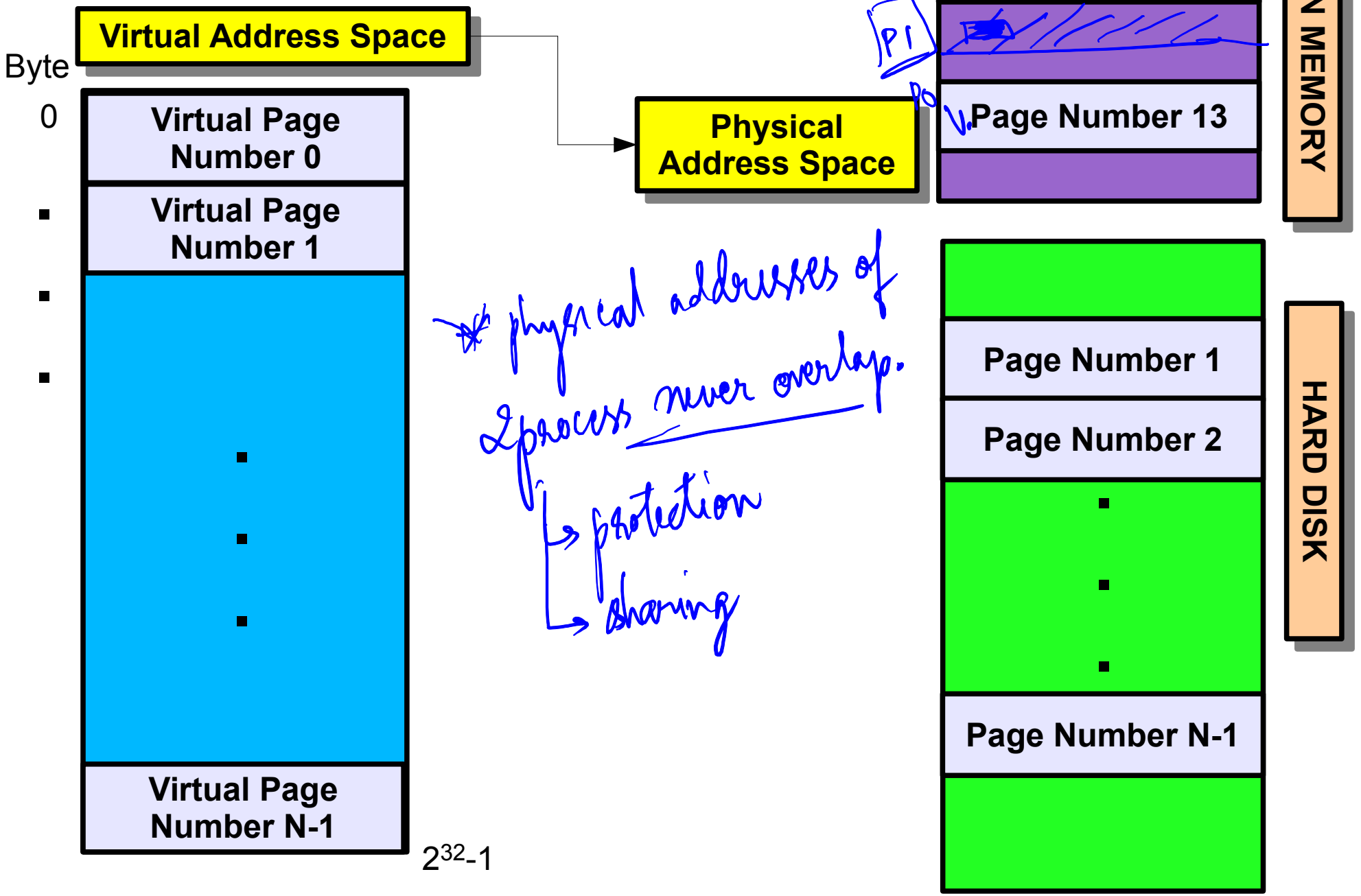
Paged Virtual Memory

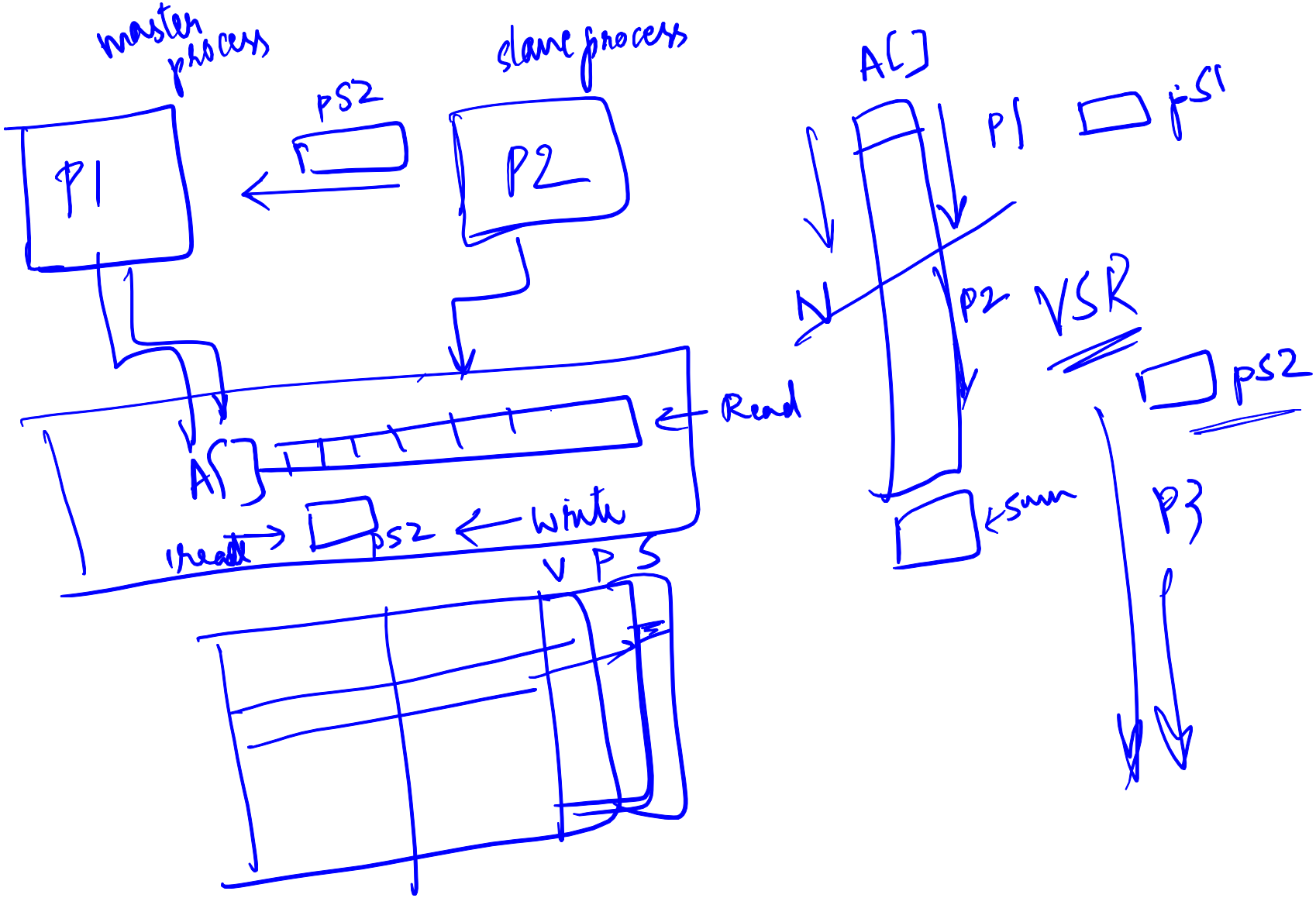


Paged Virtual Memory

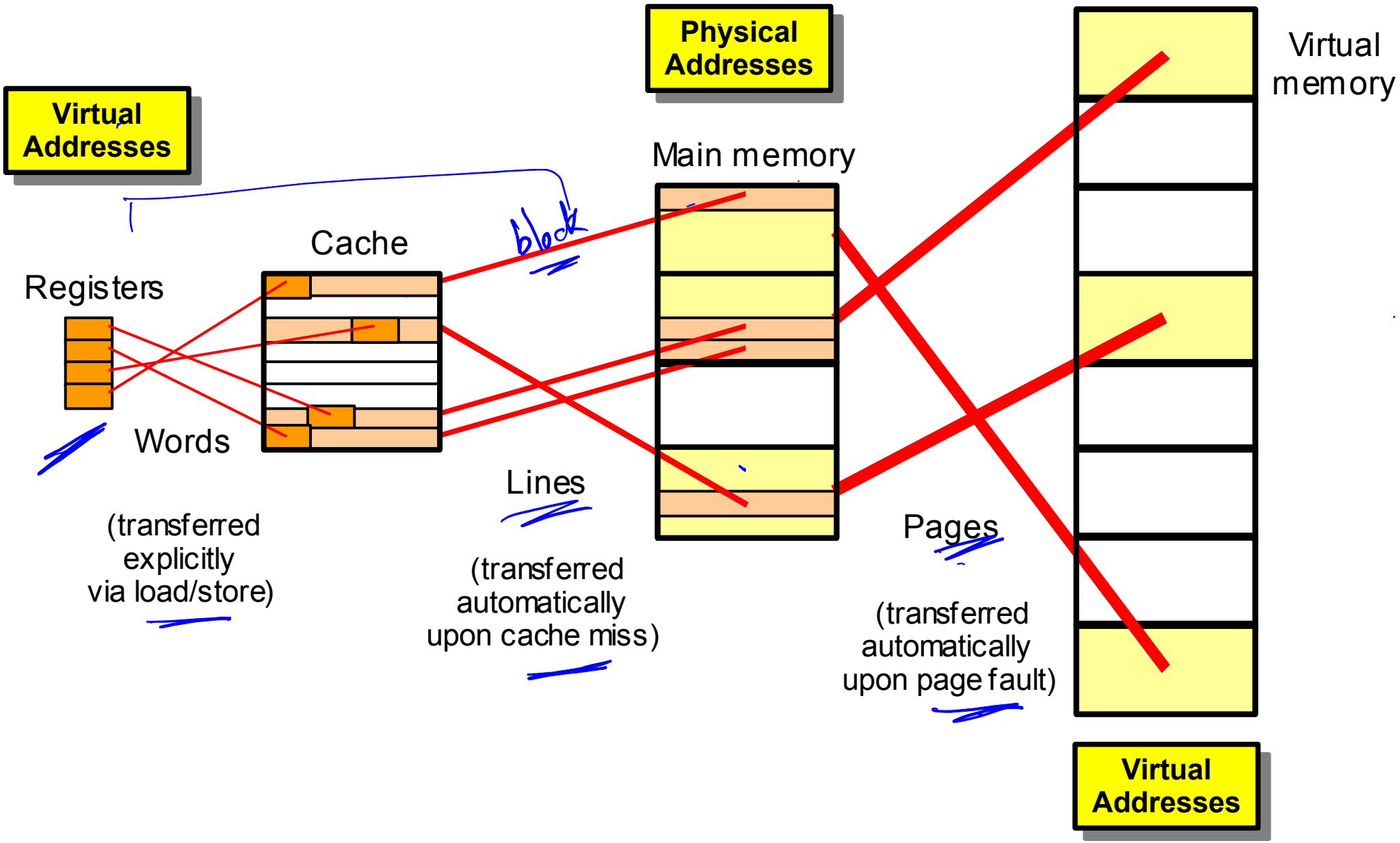


Paged Virtual Memory

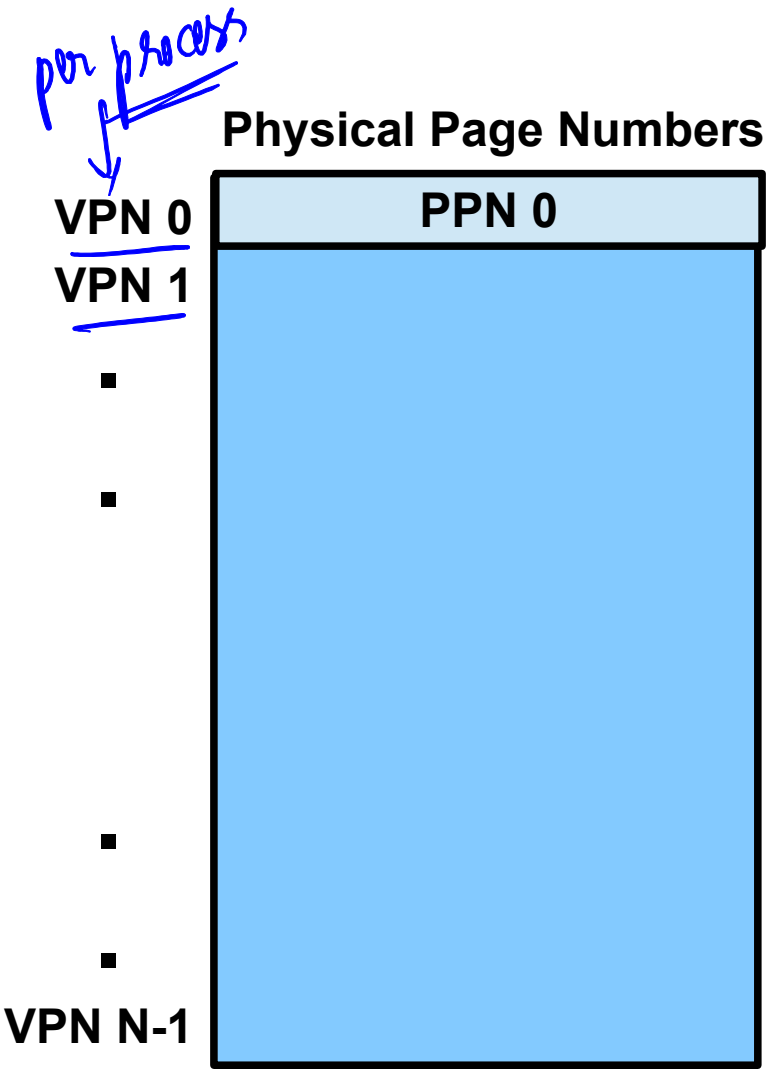



master process
slave process
P1
PS2
P2
A[]
Read
Write
V P S
A[]
P1
j
S1
N
P2
VSR
PS2
Sum
P3

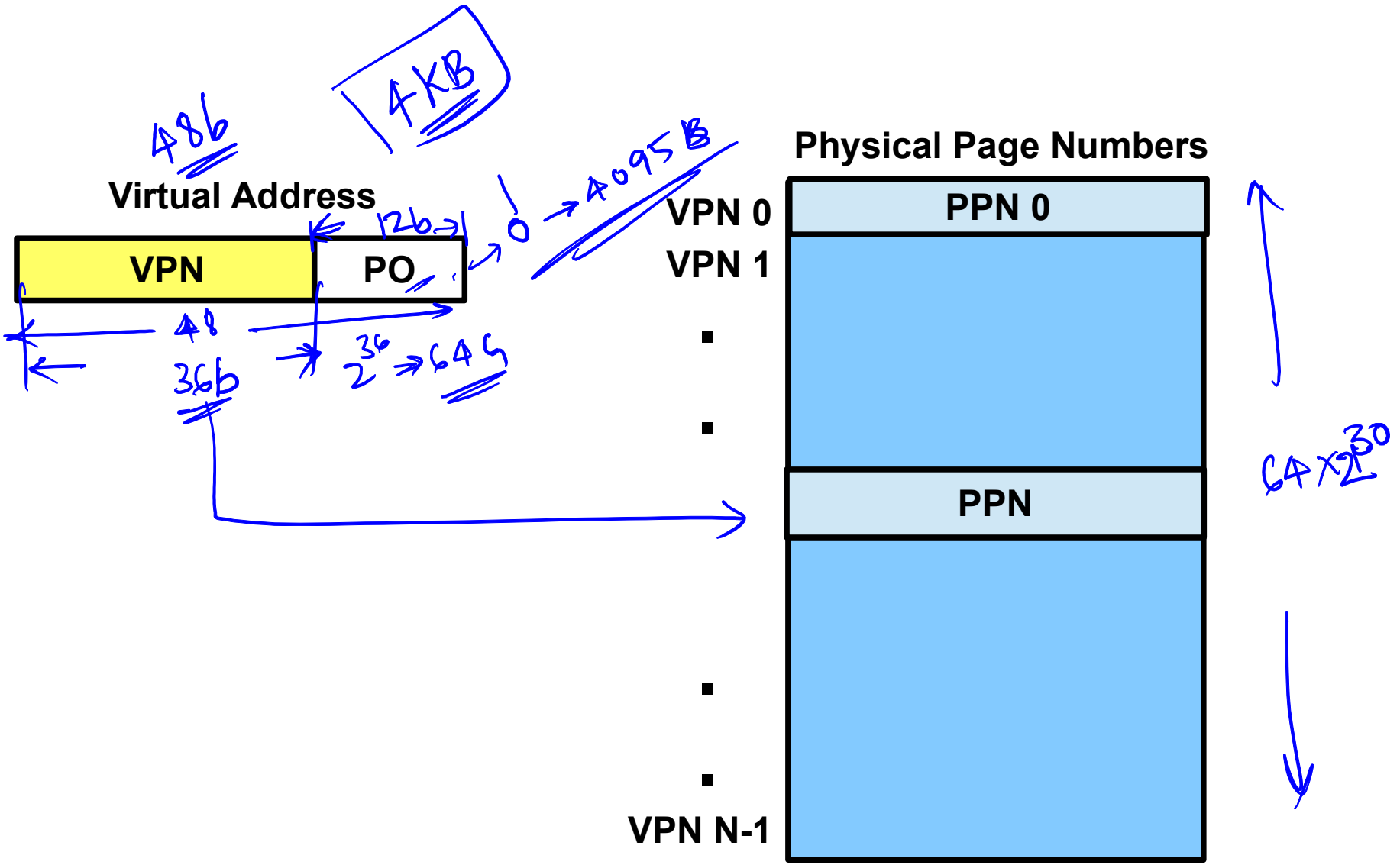
The Memory Hierarchy



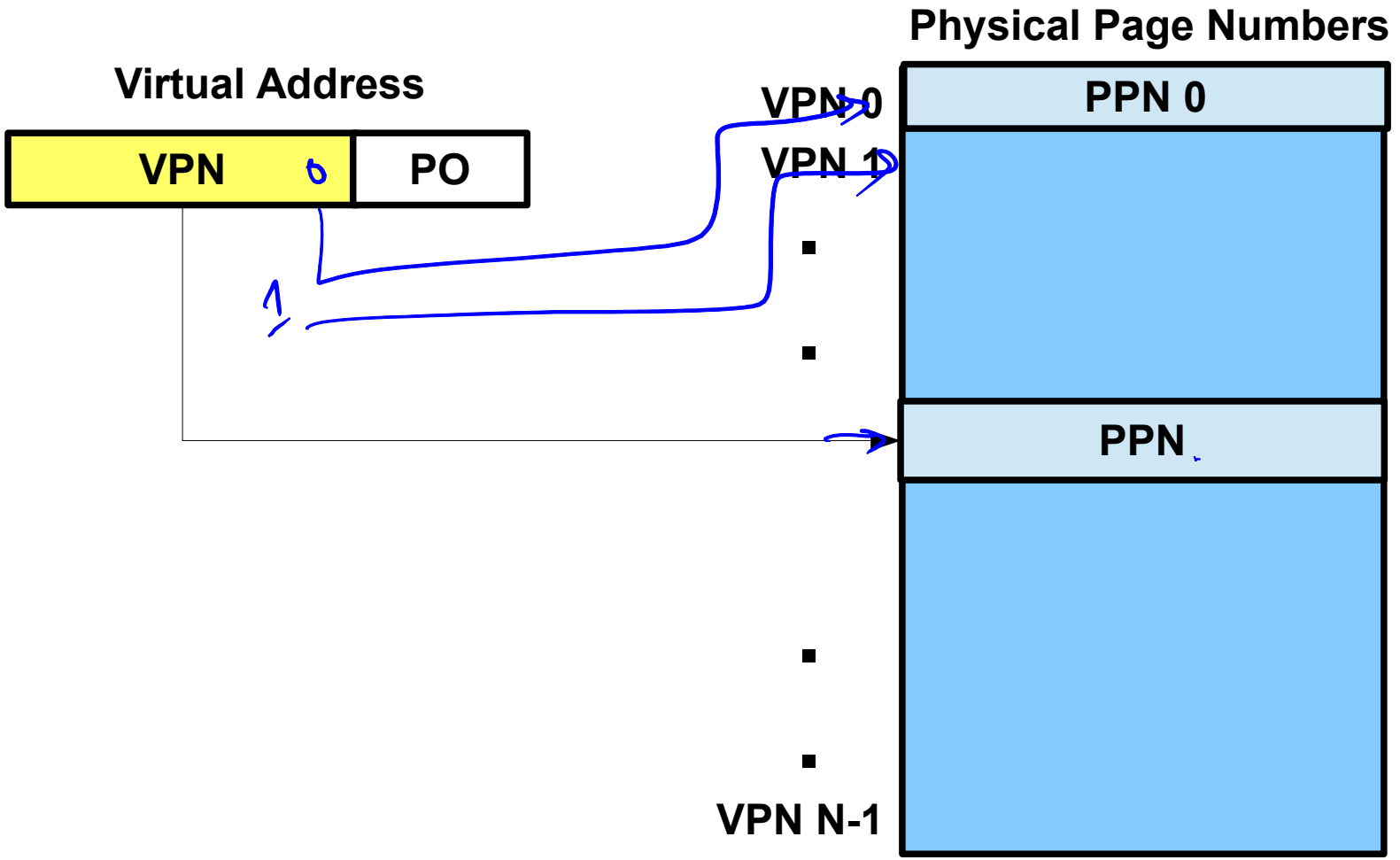
Address Translation Table



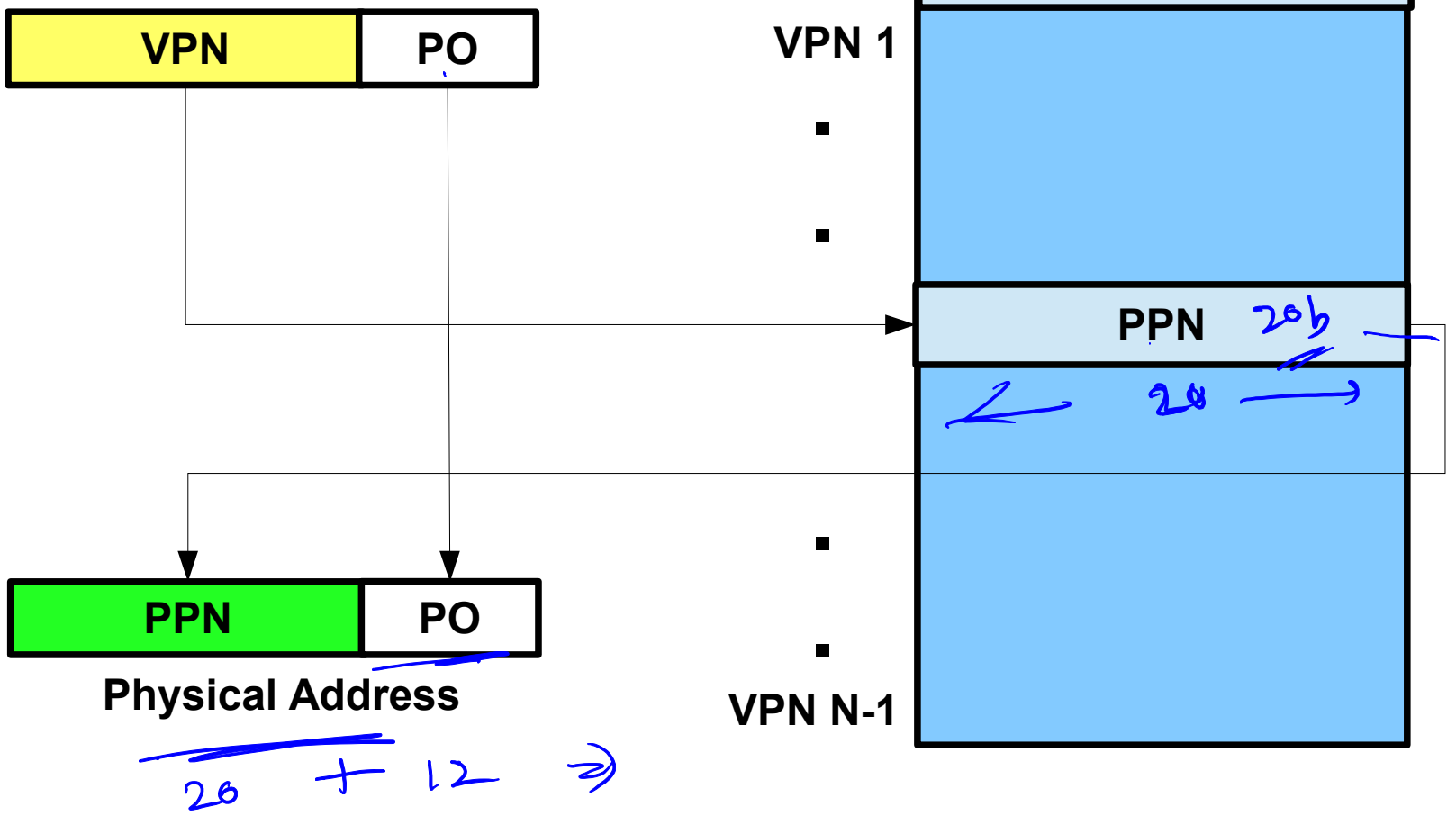
Address Translation Table



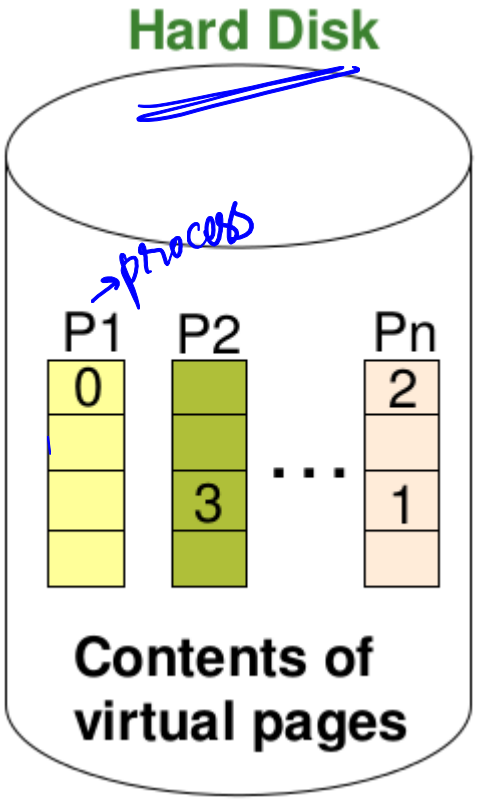
Address Translation Table



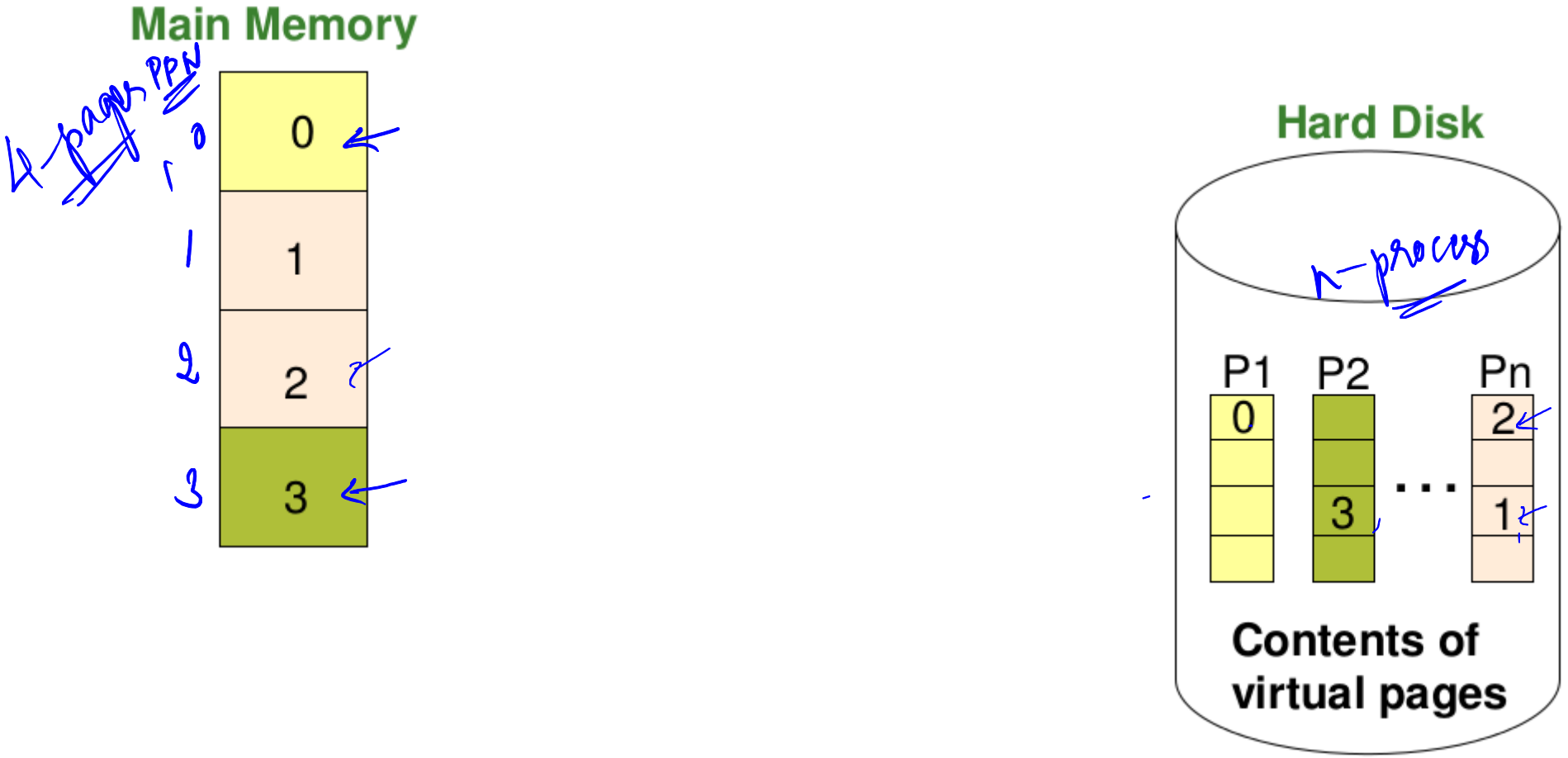
Address Translation Table



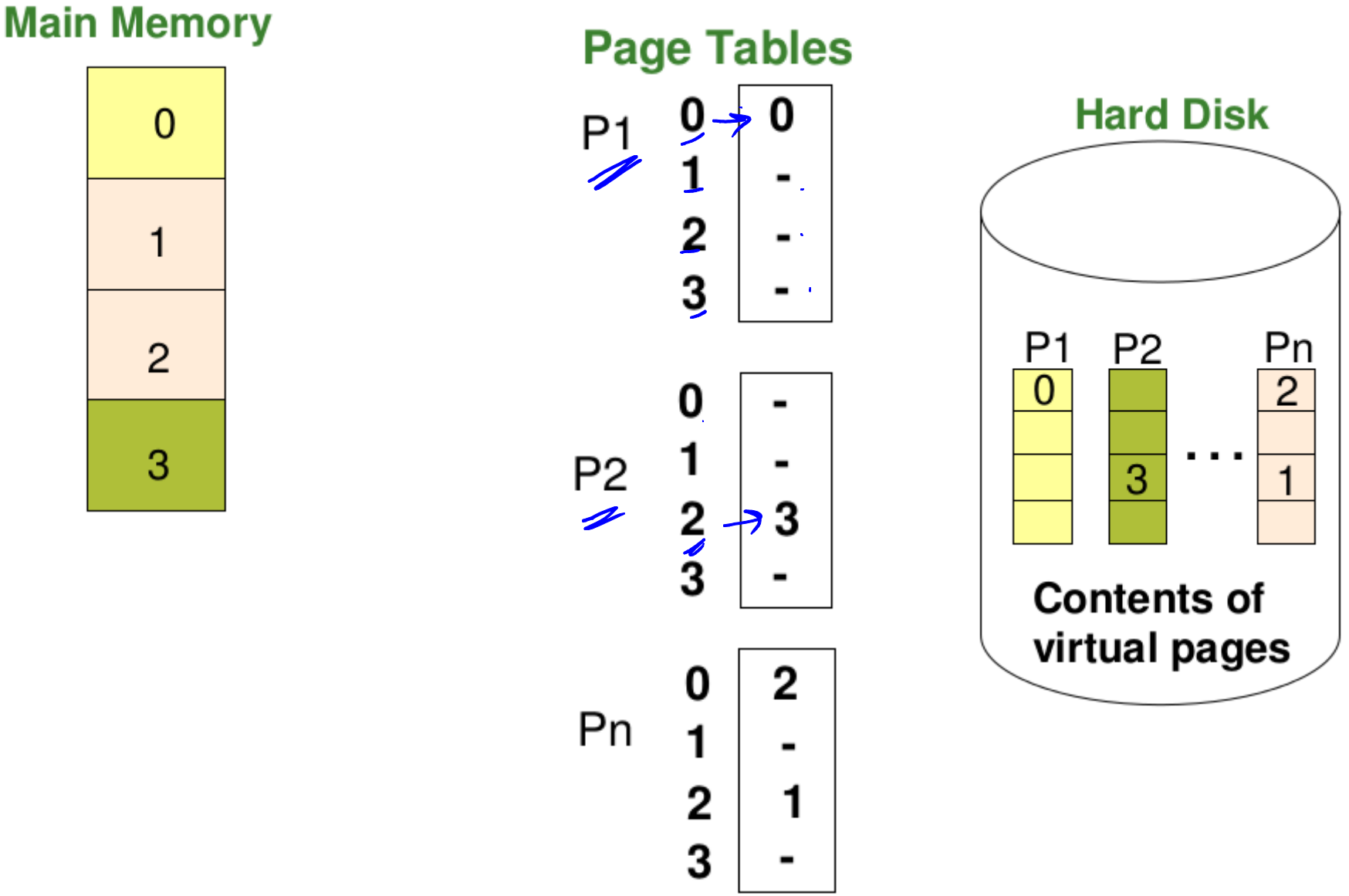
Virtual Memory



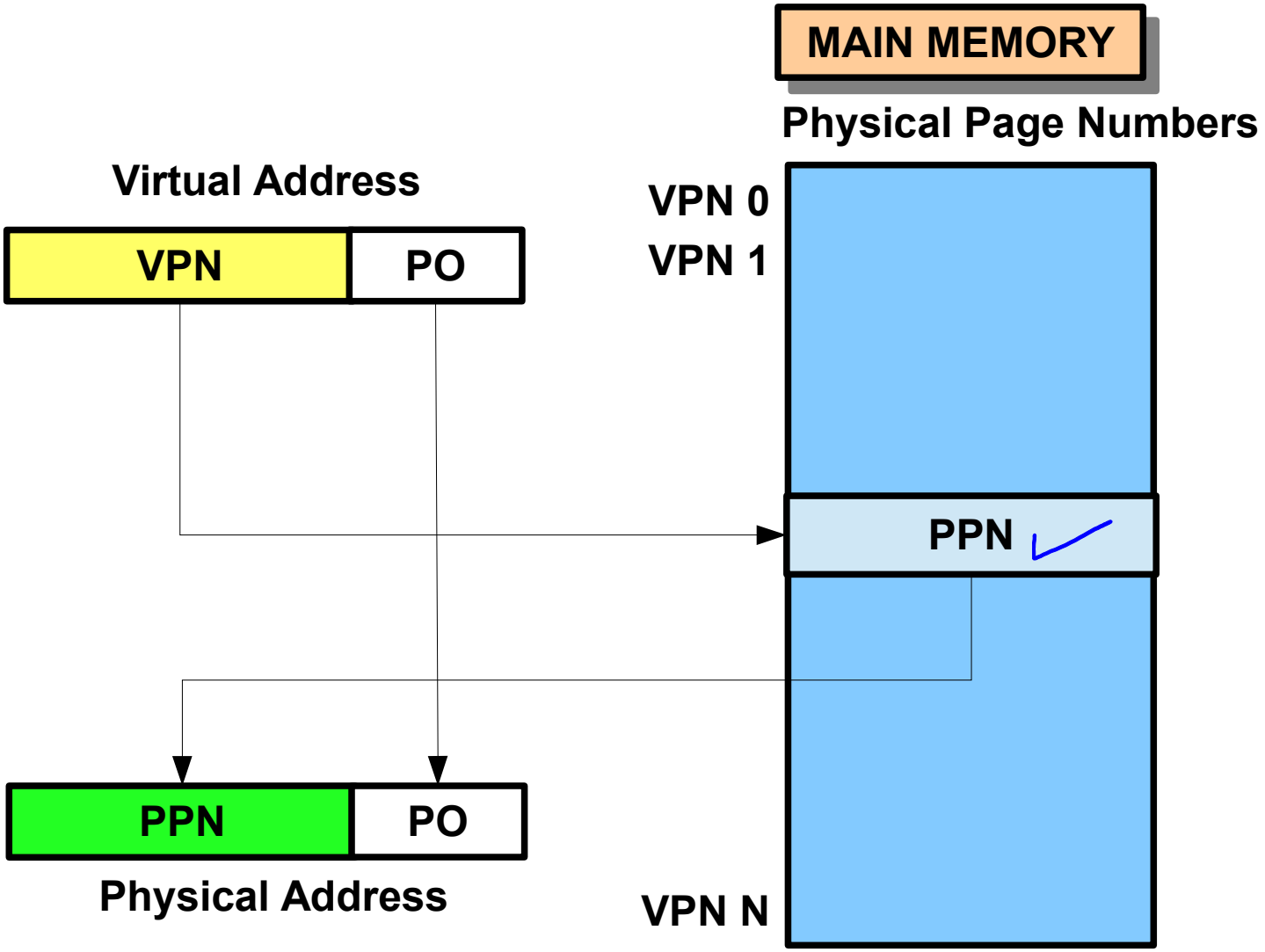
Virtual Memory



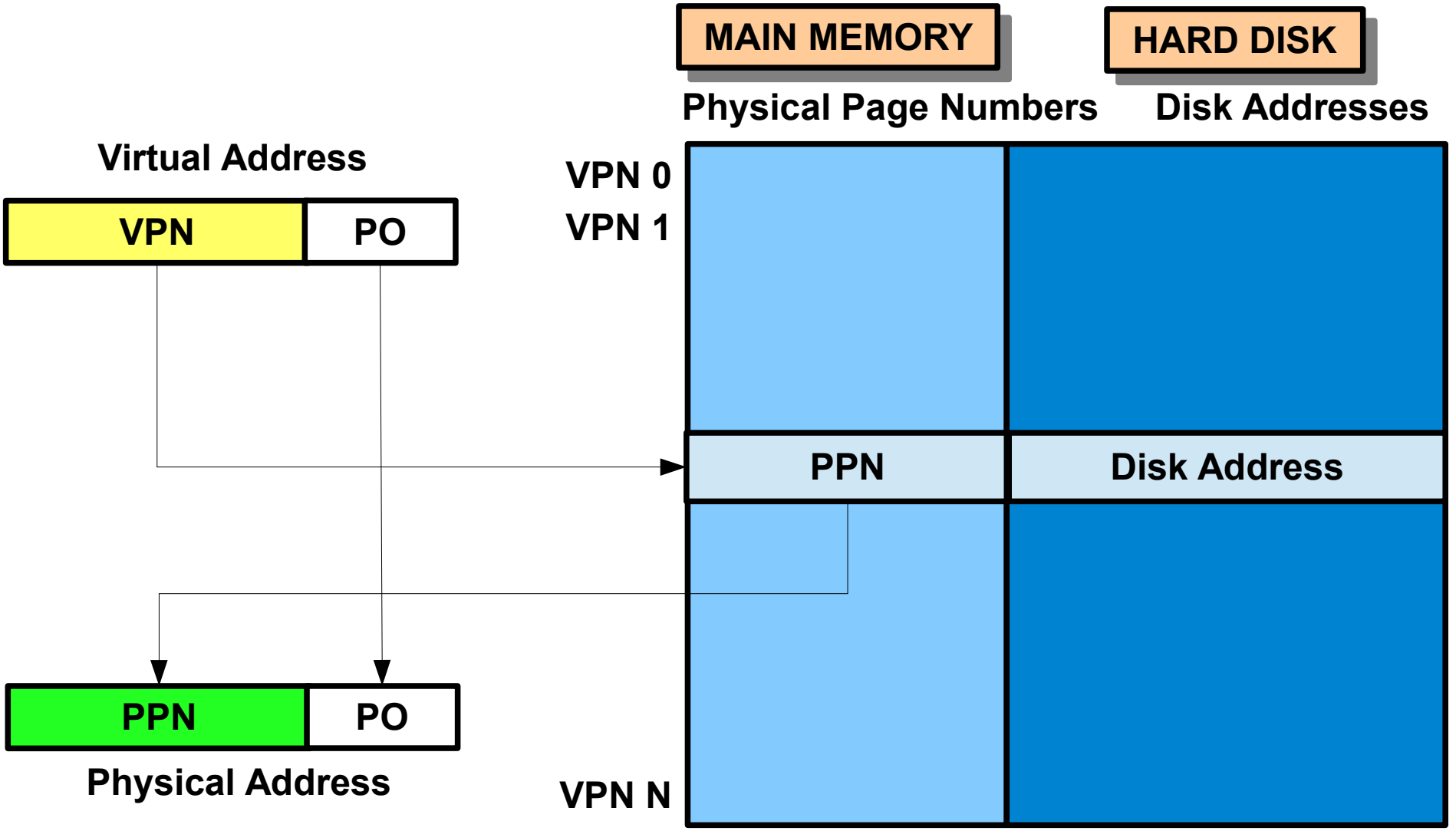
Virtual Memory



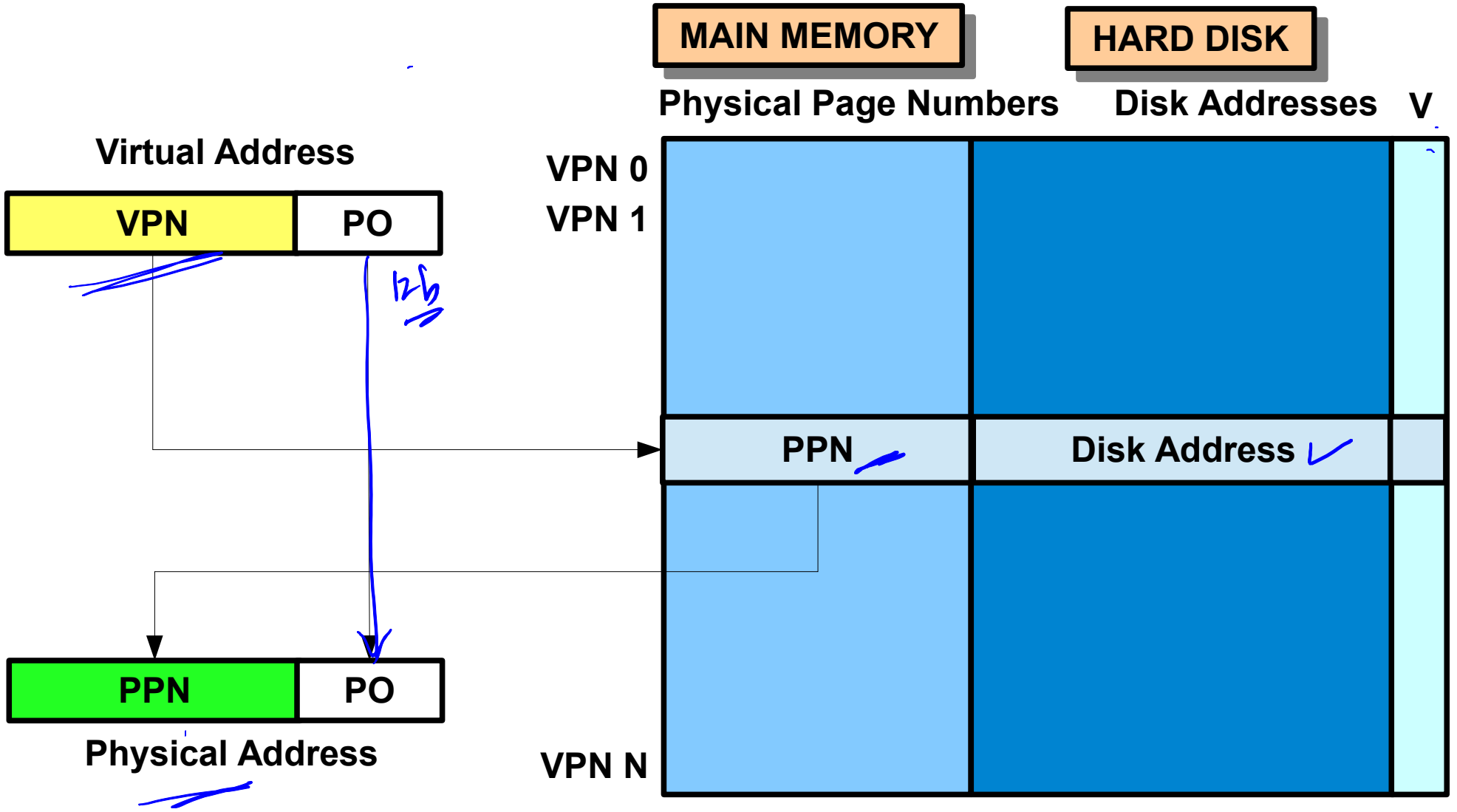
Address Translation Table



Address Translation Table



Address Translation Table



Implementation of Address Translation

- Process always uses virtual addresses

Implementation of Address Translation

- Process always uses virtual addresses
- **Memory Management Unit (MMU)**: part of CPU; hardware that does address translation

Implementation of Address Translation

- Process always uses virtual addresses
- **Memory Management Unit (MMU)**: part of CPU; hardware that does address translation
- The page tables are (at best) present in the MM (OS virtual address space)
  - One main memory reference per address translation!
- load R1, 4(R2) → VA
  - * all addresses are VA
  - * addr translation → 1 MM access
  - * 1 MM access to get the data

4GB VA → PO: 12b

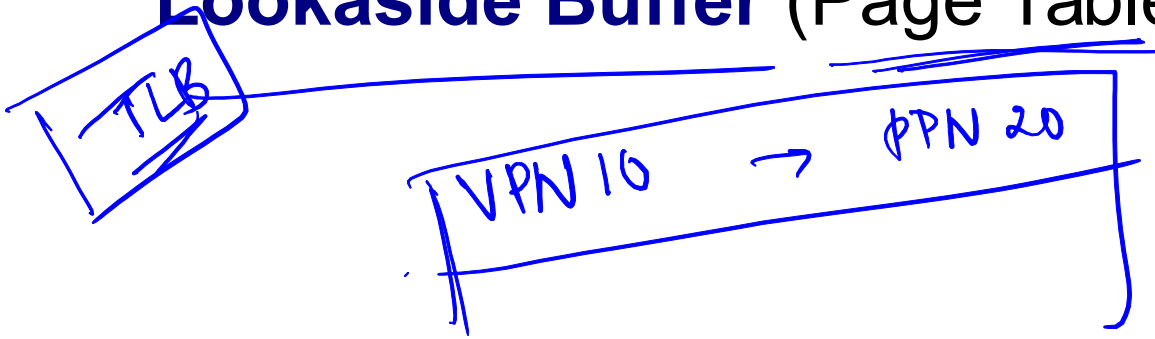
2^{36} entries in PageTable

Implementation of Address Translation

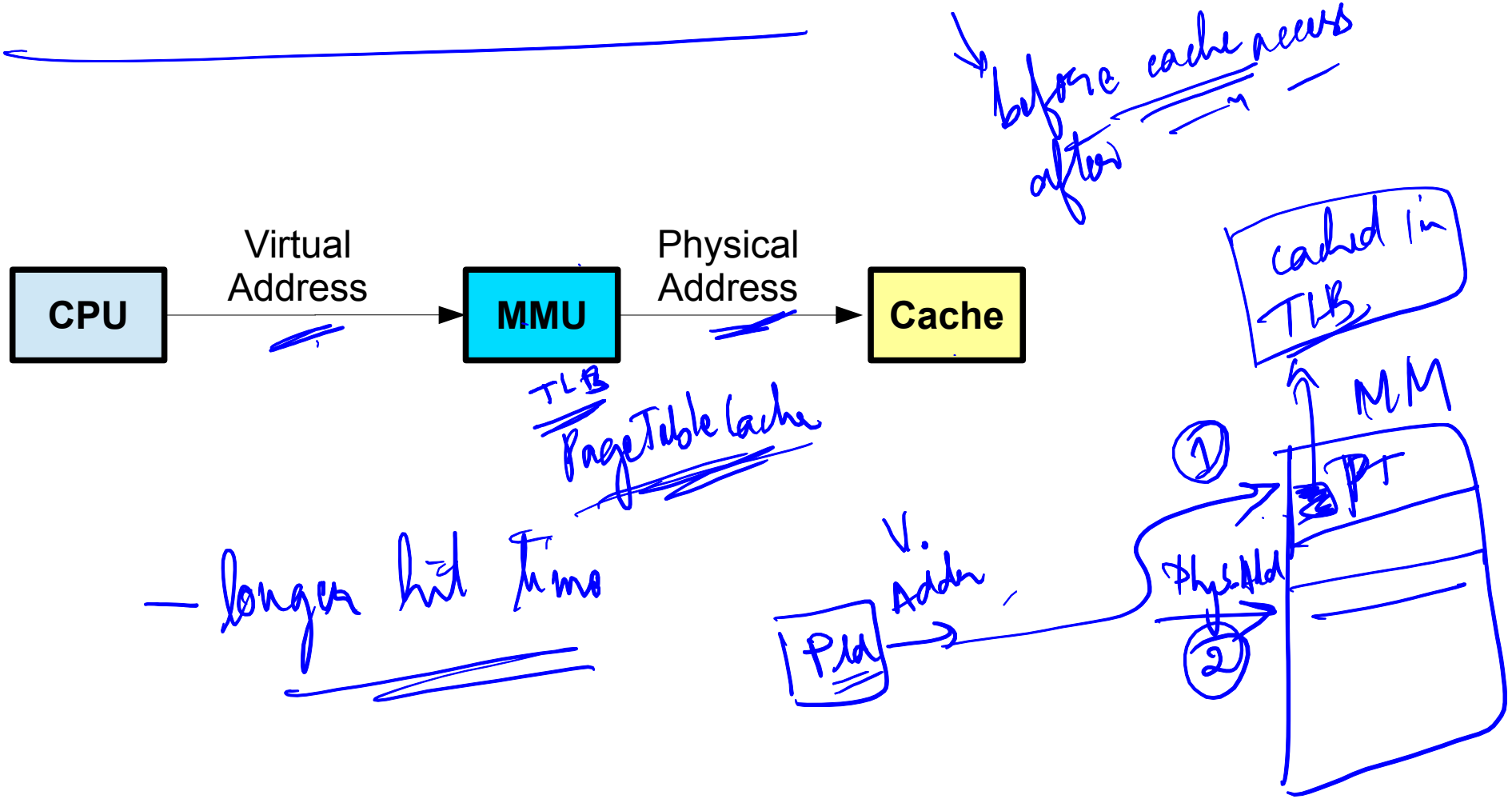
- Process always uses virtual addresses
- **Memory Management Unit (MMU)**: part of CPU; hardware that does address translation
- To translate a virtual memory address, the MMU has to read the relevant page table entry out of memory
  -

Implementation of Address Translation

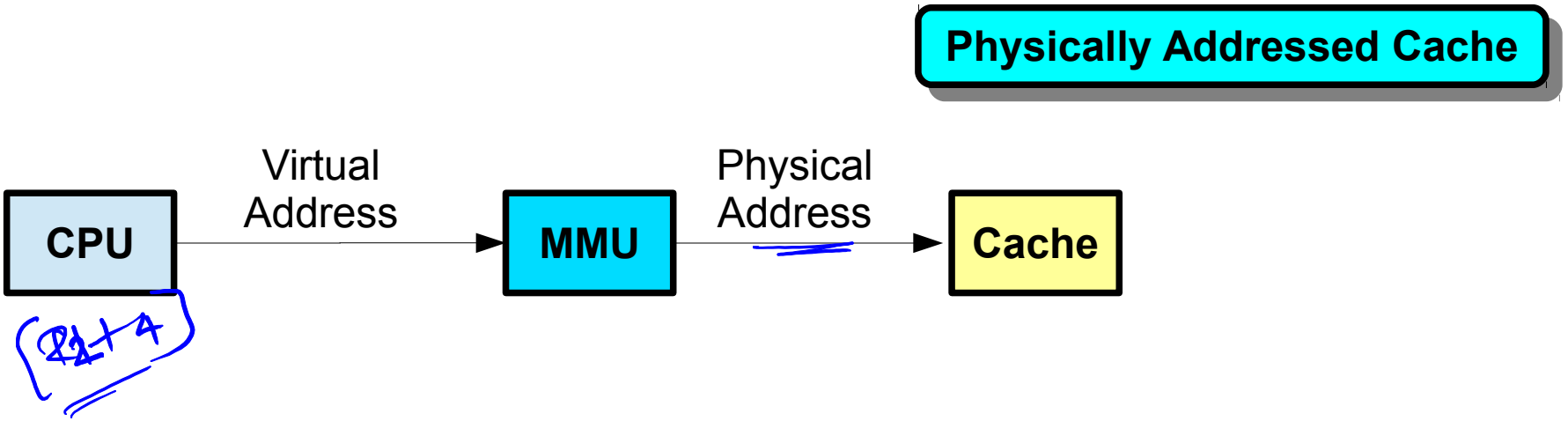
- Process always uses virtual addresses
- **Memory Management Unit (MMU)**: part of CPU; hardware that does address translation
- To translate a virtual memory address, the MMU has to read the relevant page table entry out of memory
  - Caches recently used translations in a **Translation Lookaside Buffer** (Page Table Cache)



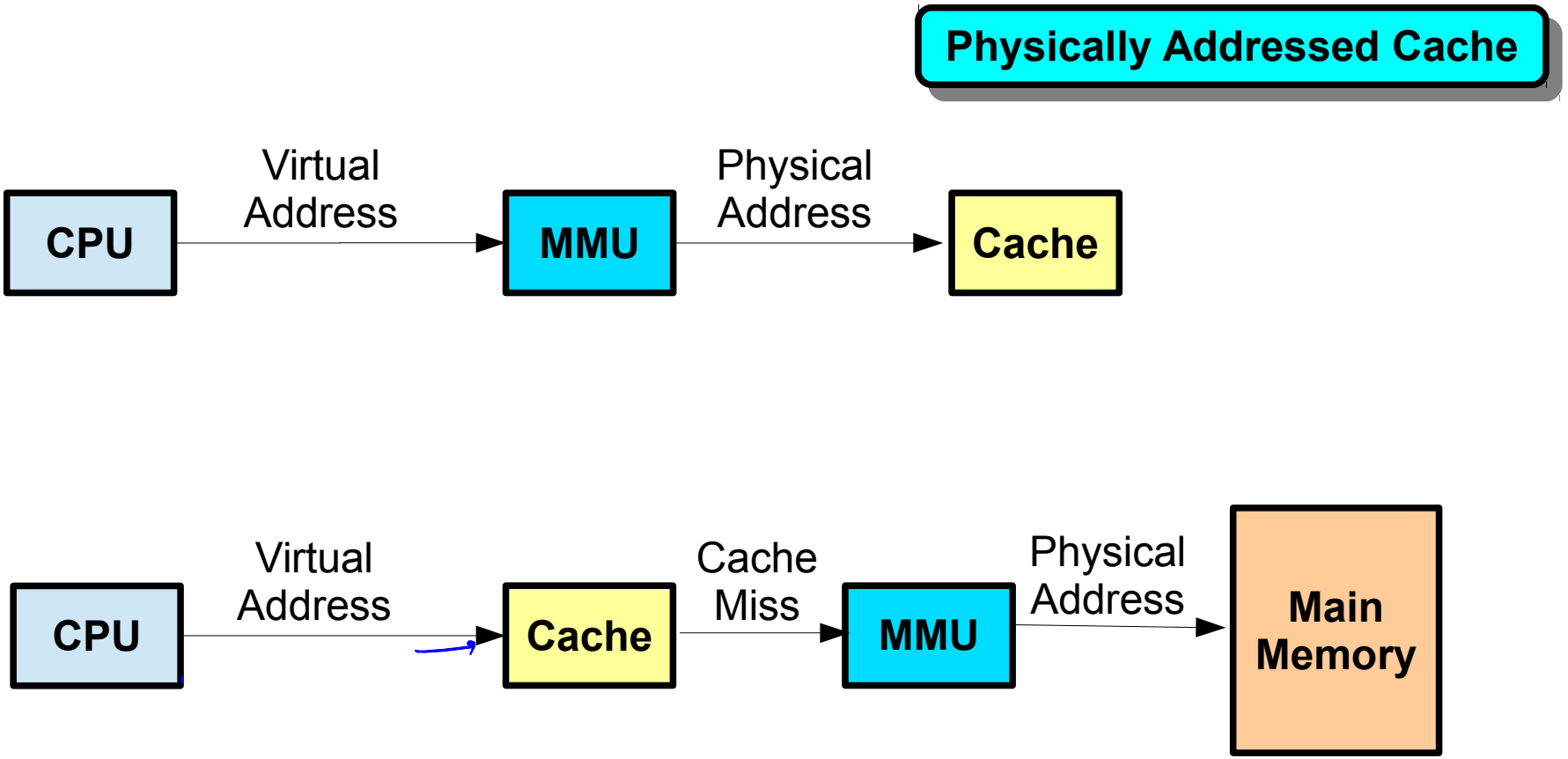
Caches and Address Translation



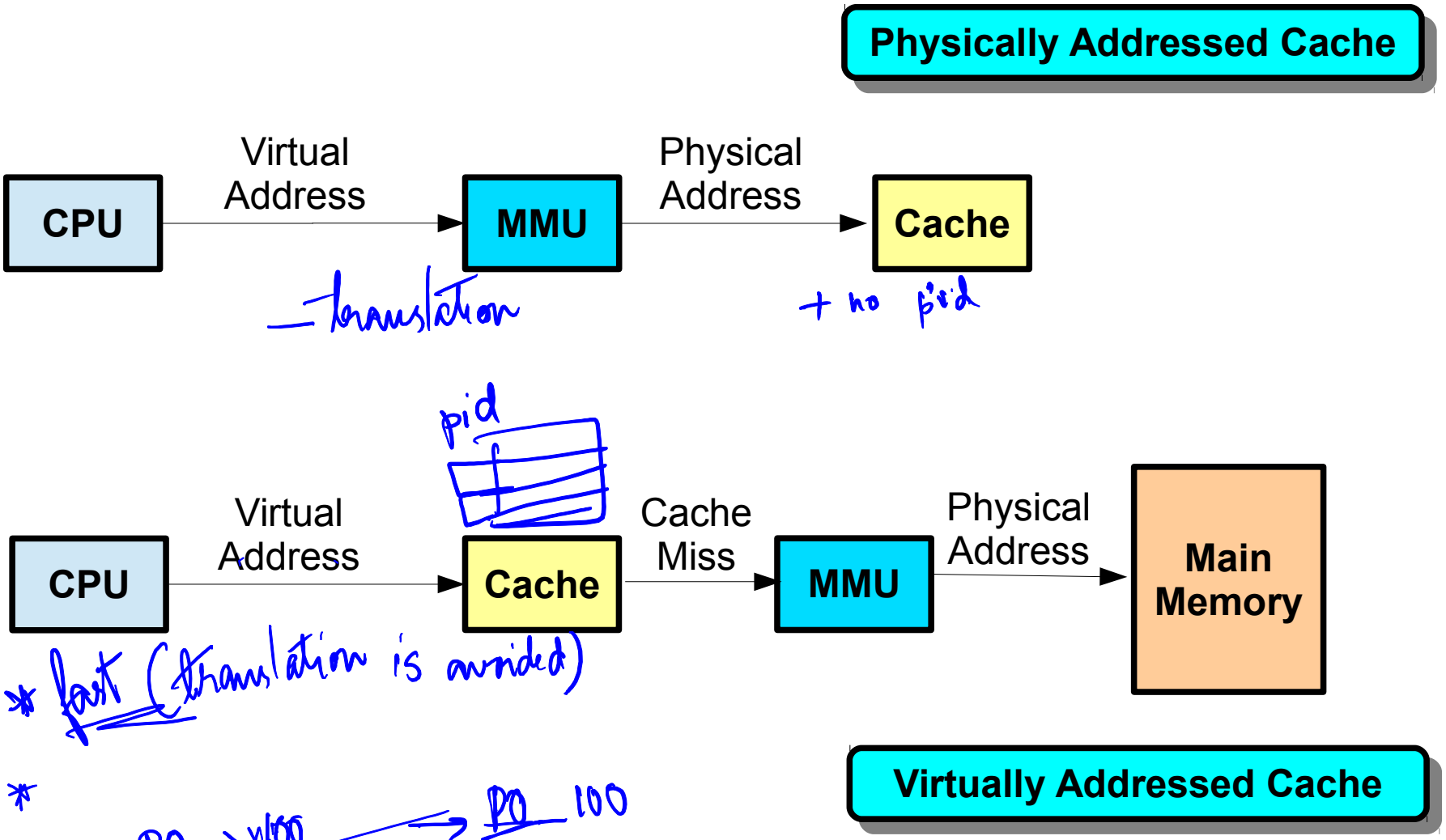
Caches and Address Translation



Caches and Address Translation



Caches and Address Translation



Which is less preferable?

- **Physical addressed cache**
  -
- **Virtual addressed cache**
  -

Which is less preferable?

- **Physical addressed cache**
  - Hit time higher (cache access after translation)
- **Virtual addressed cache**
  -

Which is less preferable?

- **Physical addressed cache**
  - Hit time higher (cache access after translation)
- **Virtual addressed cache**
  - Data/instruction of different processes with same virtual address in cache at the same time ...
    -

Which is less preferable?

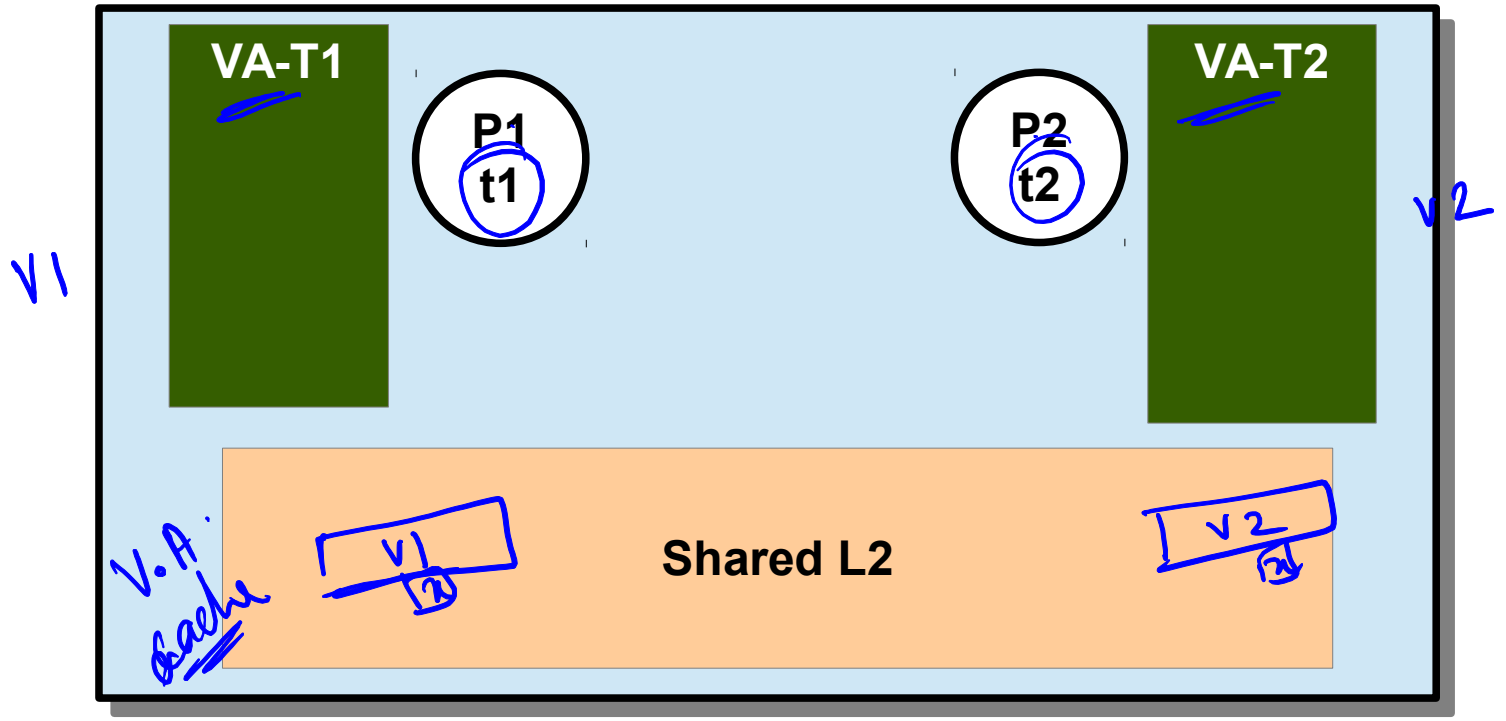
- **Physical addressed cache**
  - Hit time higher (cache access after translation)
- **Virtual addressed cache**
  - Data/instruction of different processes with same virtual address in cache at the same time ...
    - Flush cache on context switch, or
    - Include Process id as part of each cache directory entry

process state → page table addr

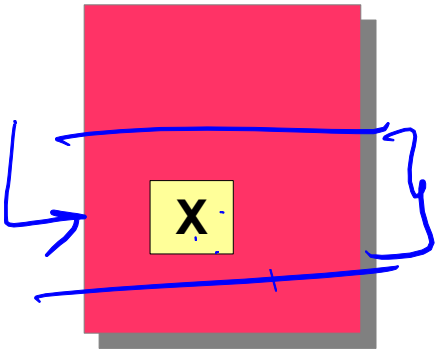
Which is less preferable?

- **Physical addressed cache**
  - Hit time higher (cache access after translation)
- **Virtual addressed cache**
  - Data/instruction of different processes with same virtual address in cache at the same time ...
    - Flush cache on context switch, or
    - Include Process id as part of each cache directory entry
  - Synonyms

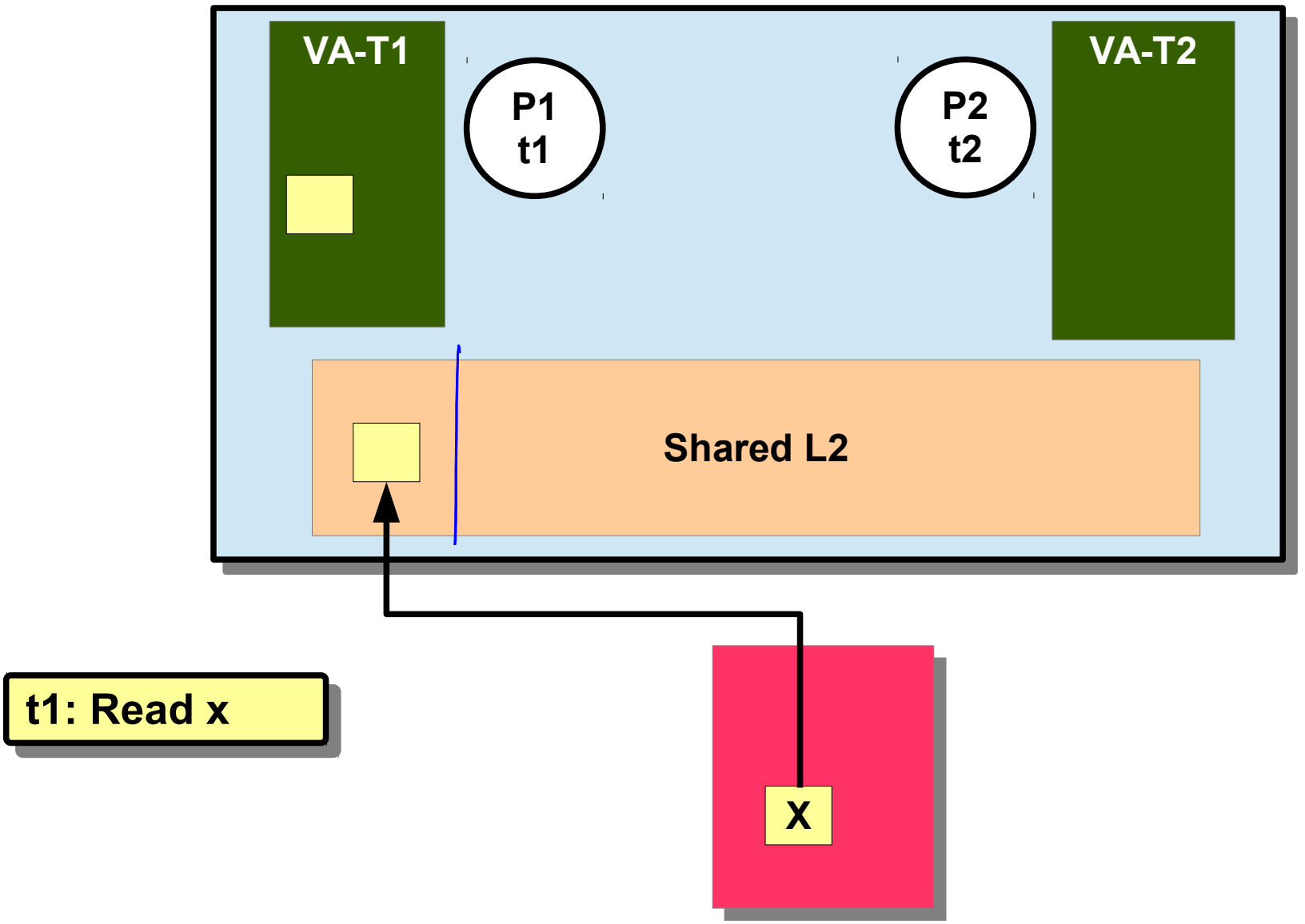
Synonyms (Aliases)



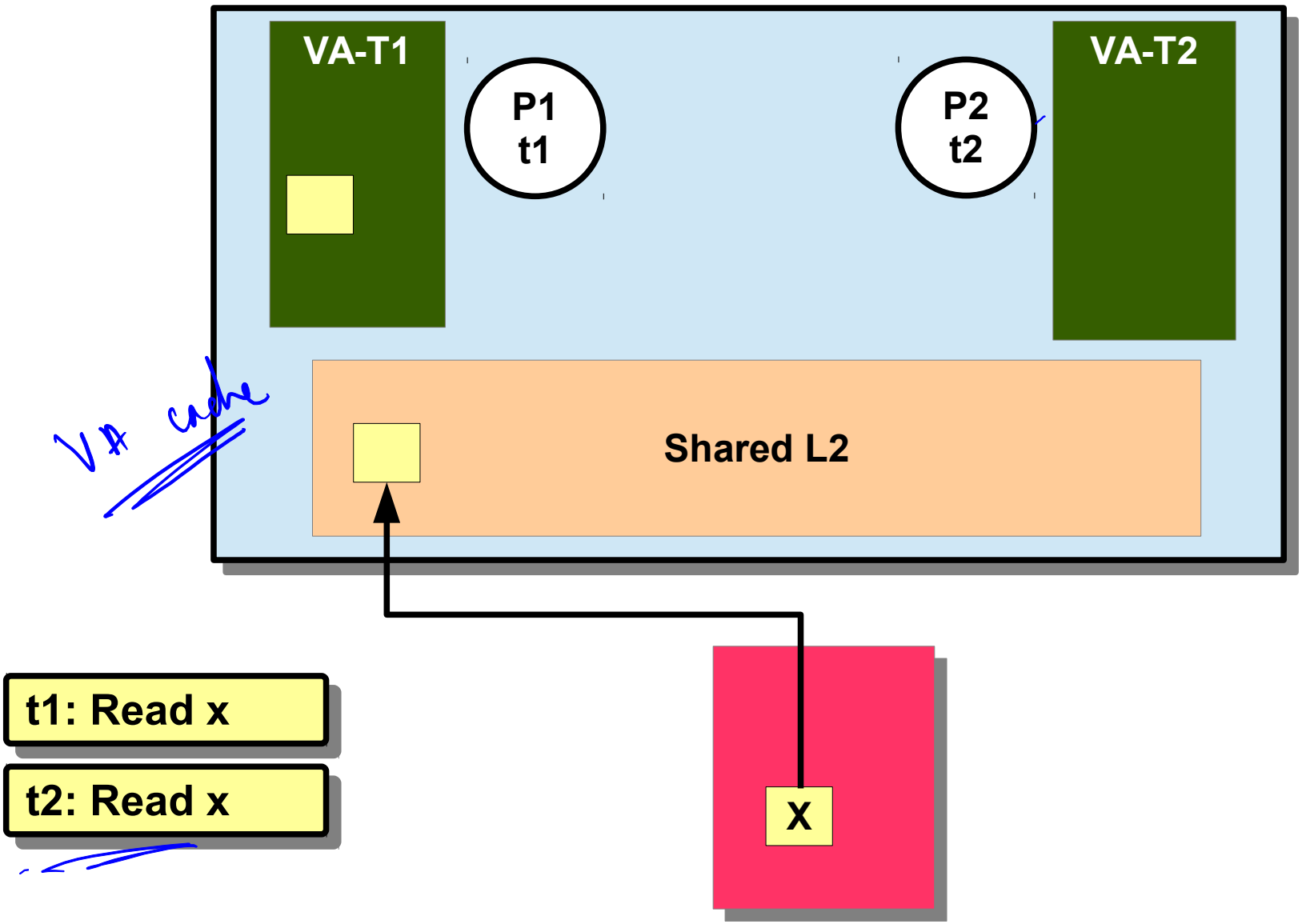
t1: Read x



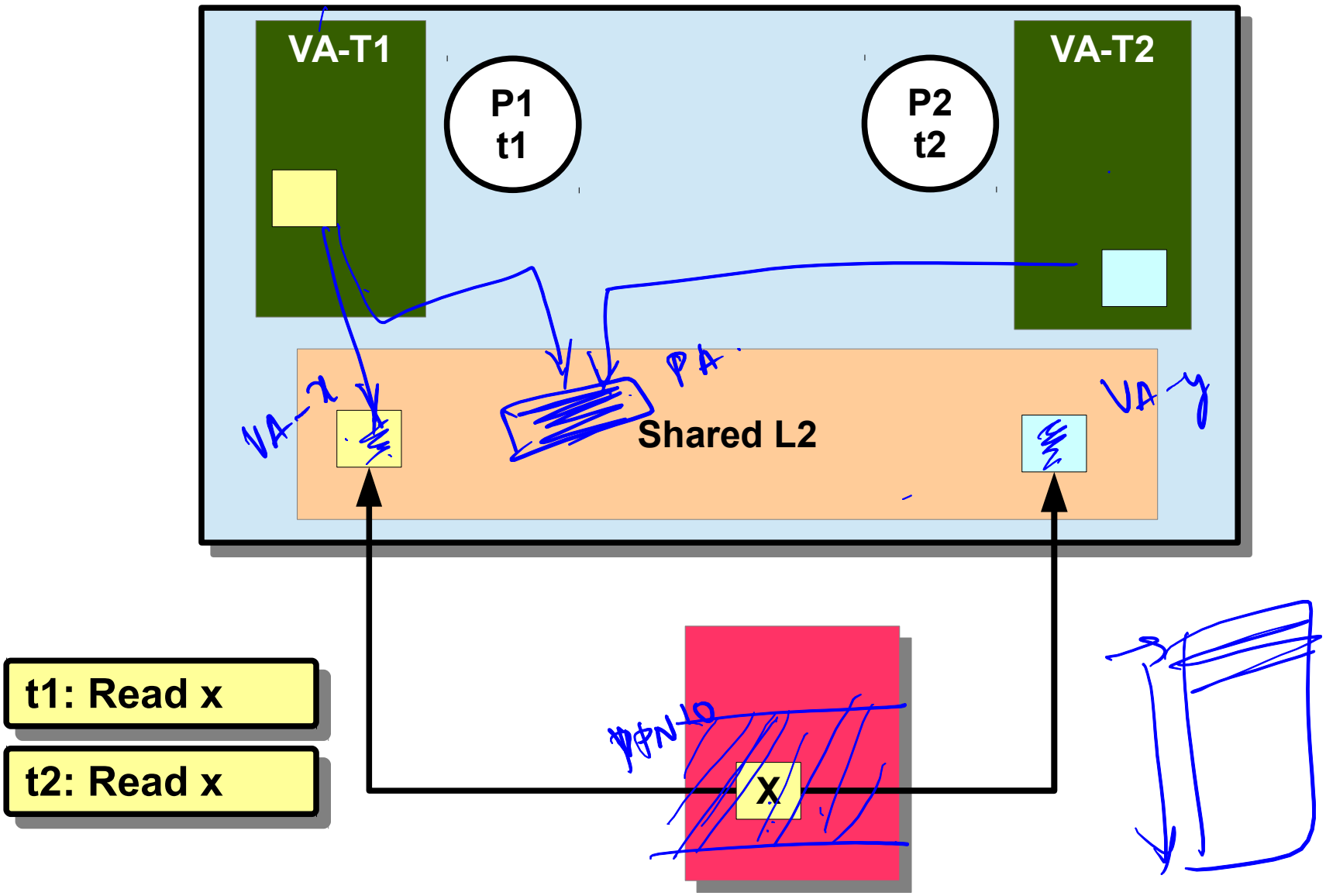
Synonyms (Aliases)



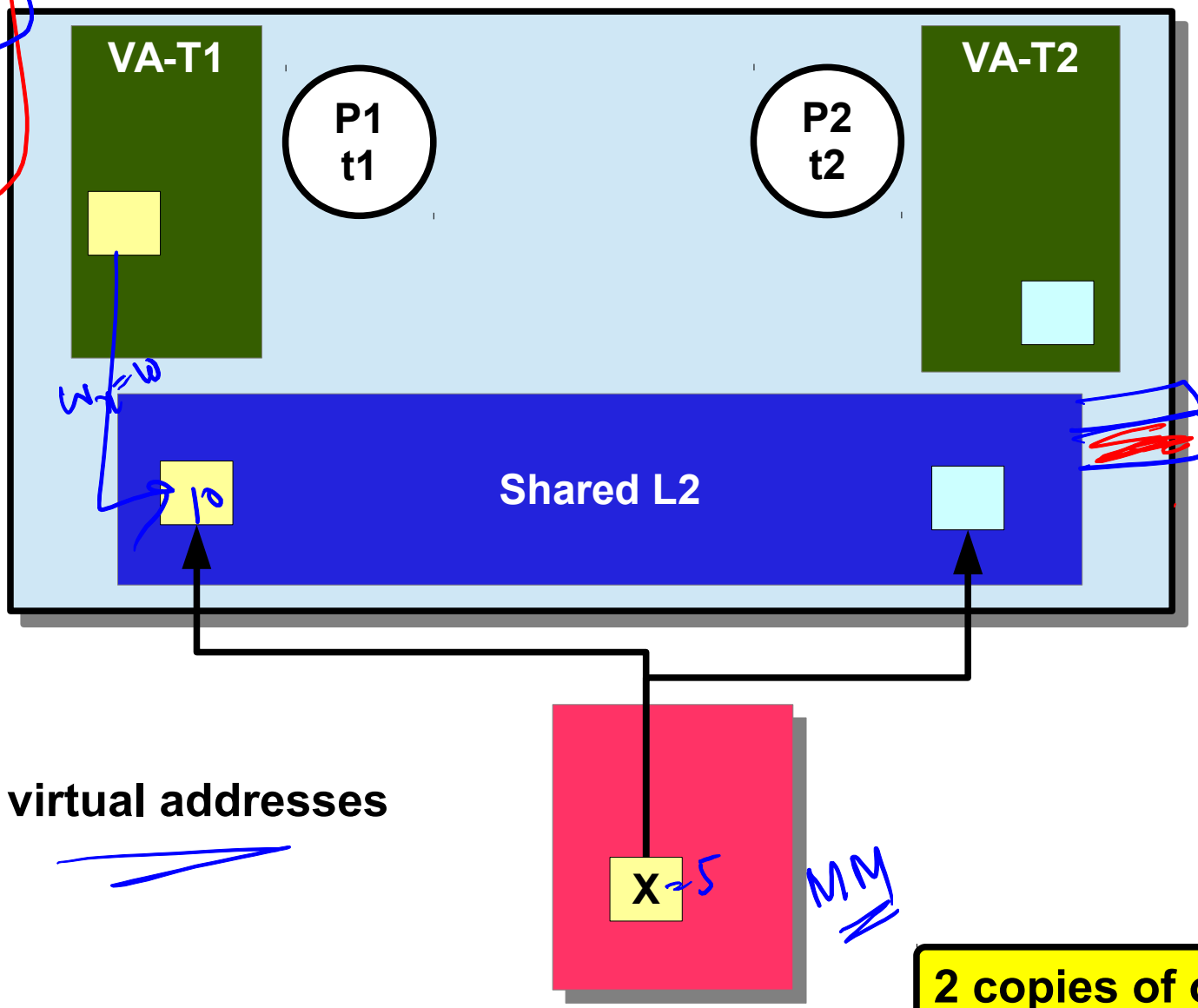
Synonyms (Aliases)



Synonyms (Aliases)

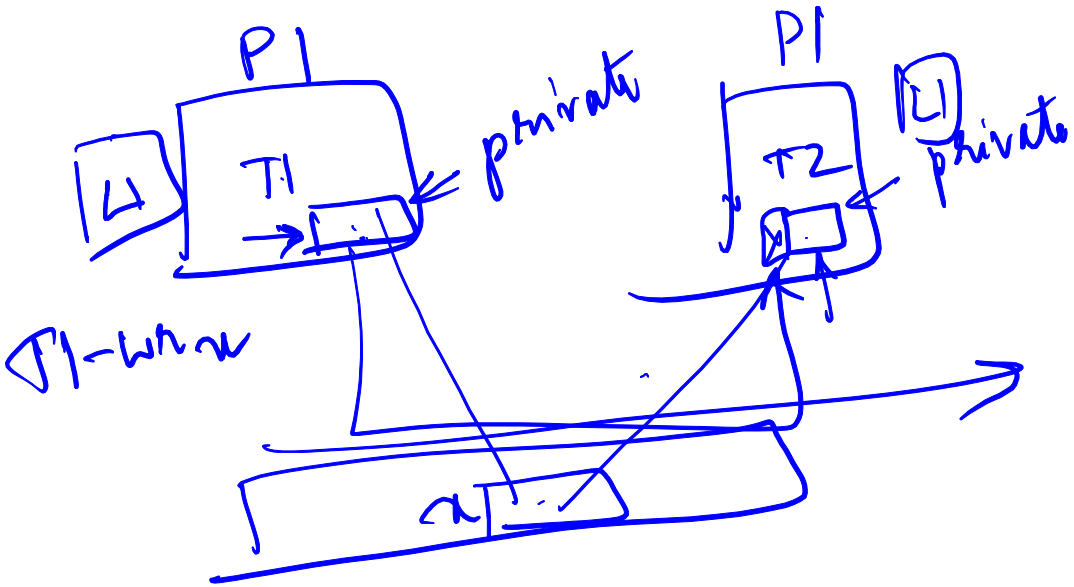


Synonyms (Aliases)



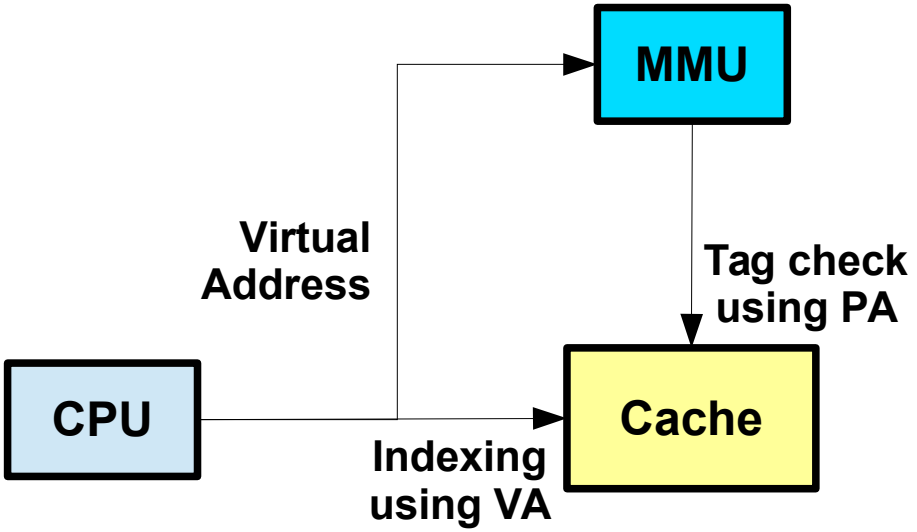
- L2 uses virtual addresses

2 copies of one physical page in the cache!

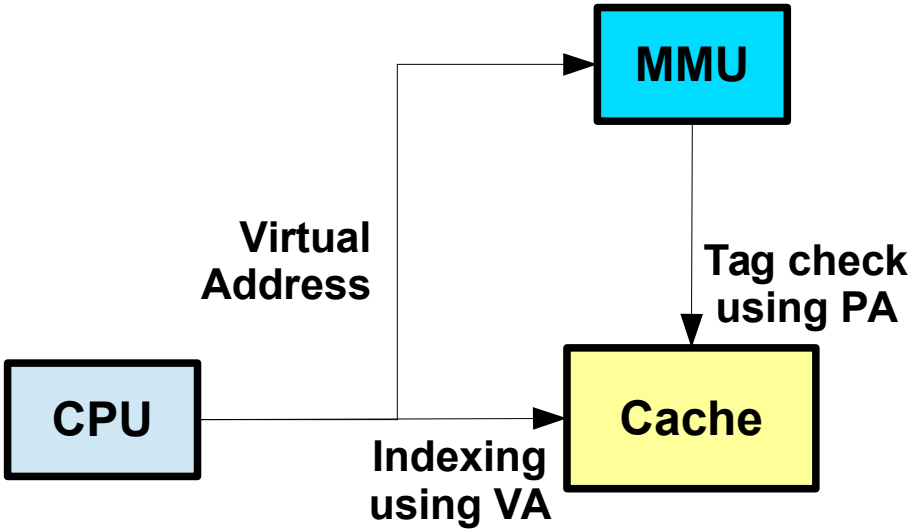


coherence

Overlapped Operation



Overlapped Operation



Virtually Indexed Physically Tagged Cache (VIPT)

Other options: PIPT, VIVT

Recall – Cache Access

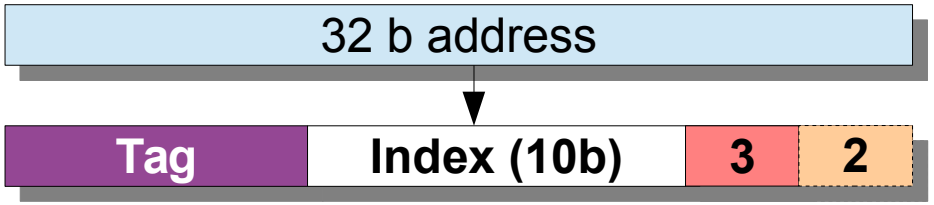
Direct mapped, 32 KB, 32B block, 32b main memory address

Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address

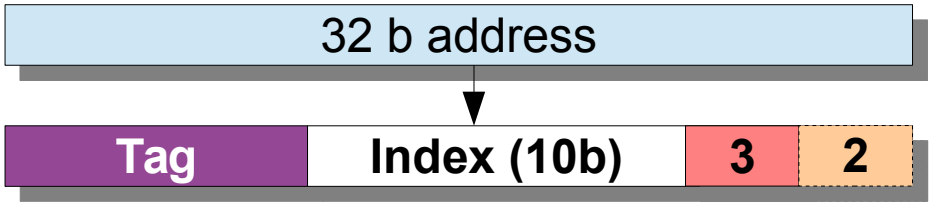
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address



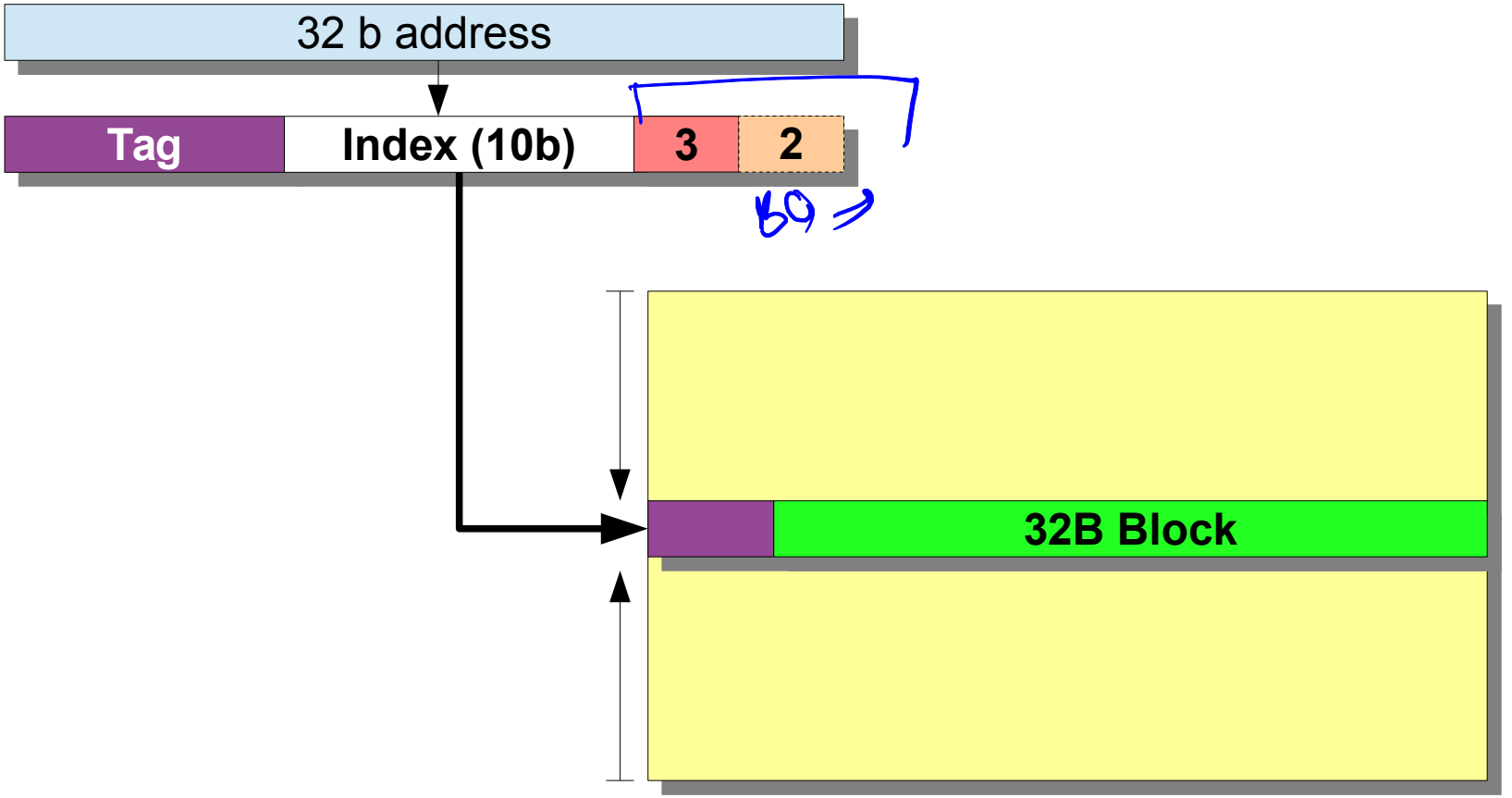
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address



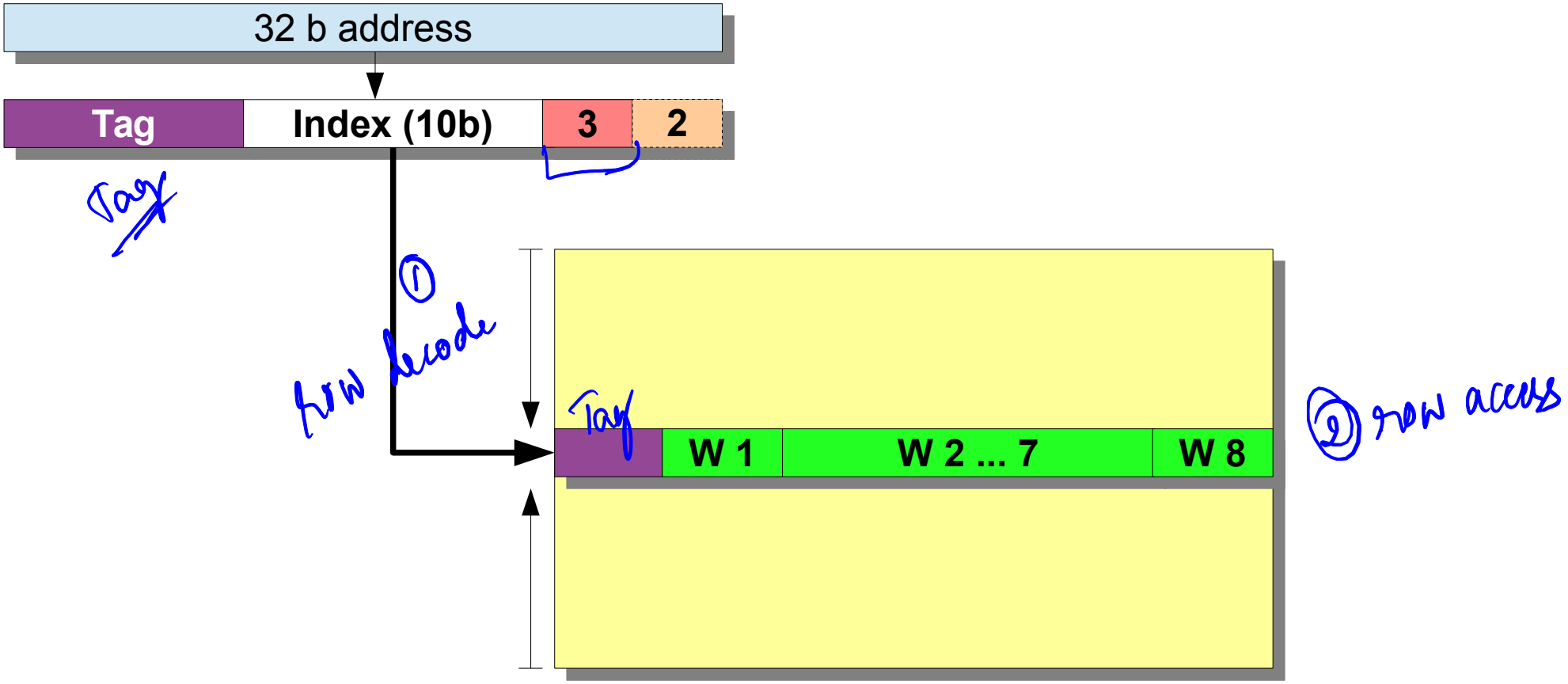
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address



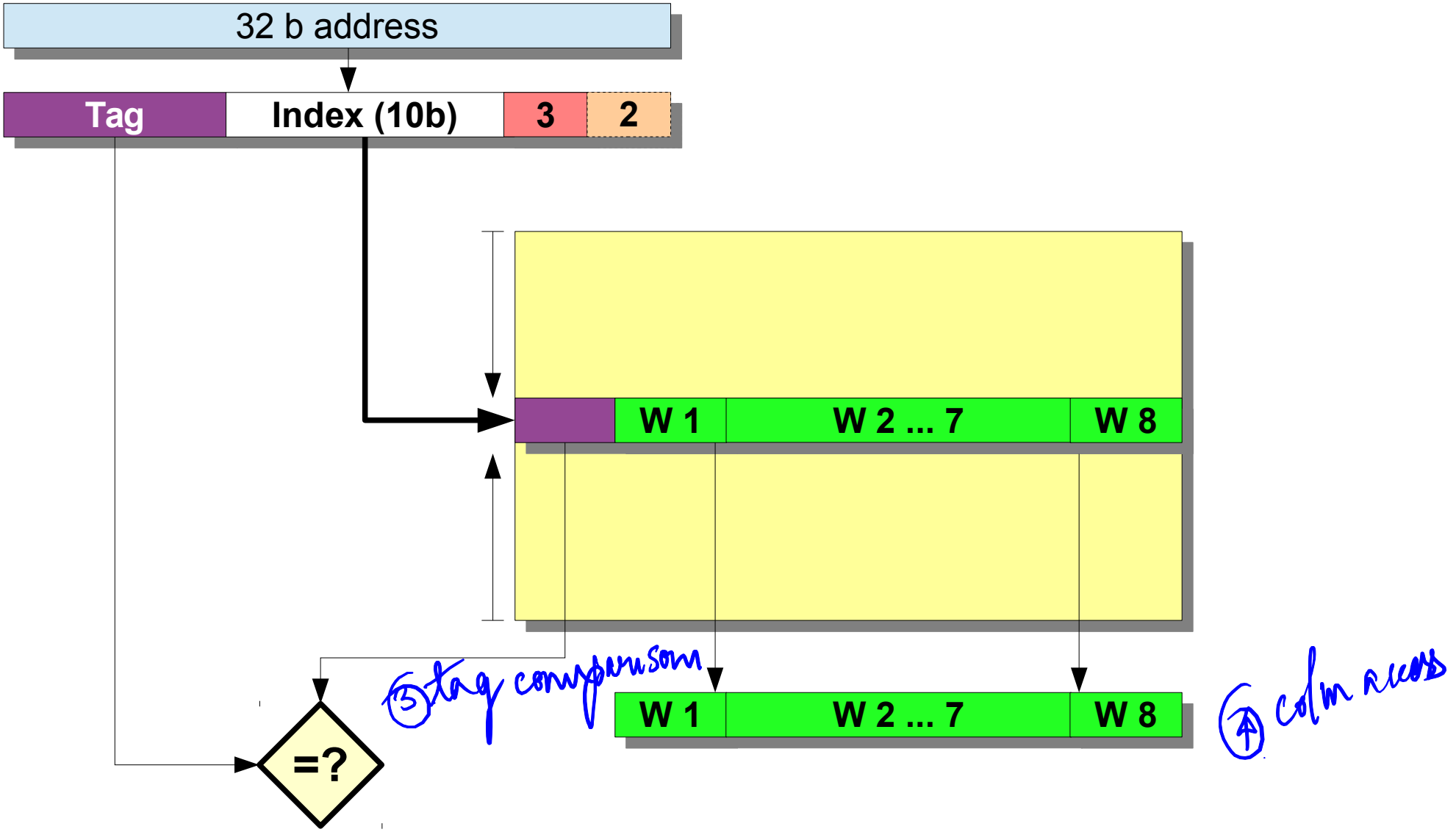
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address



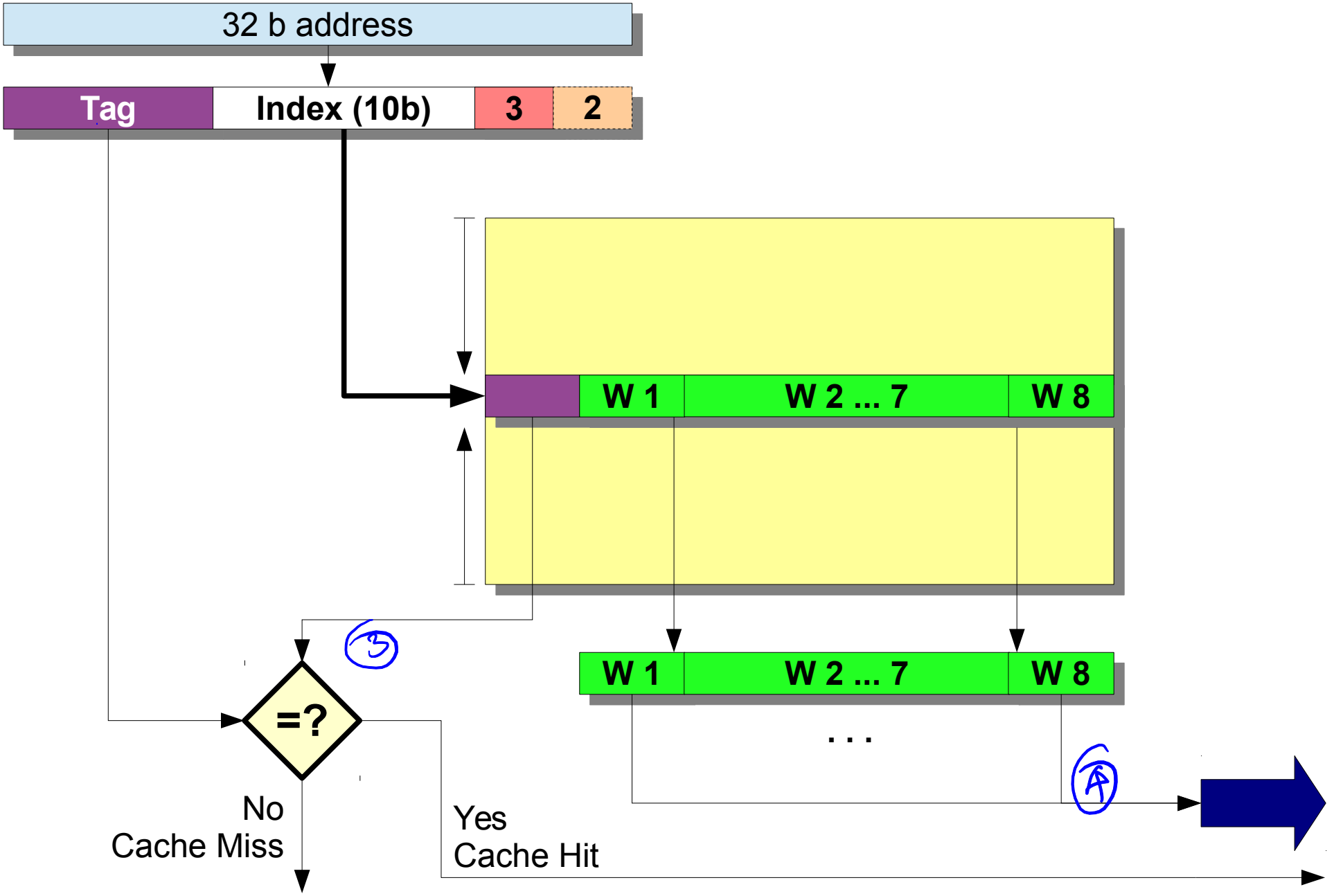
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address



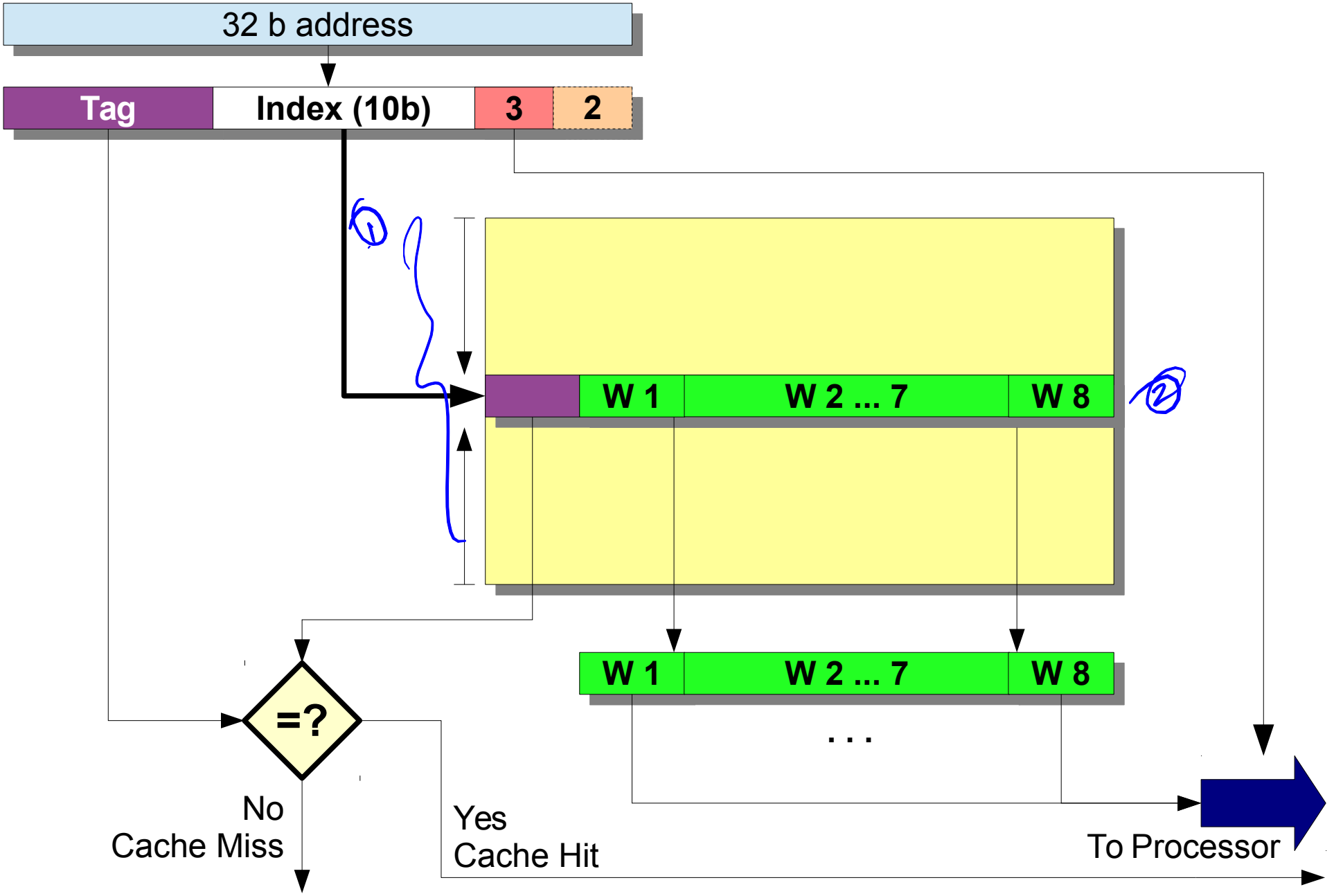
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address



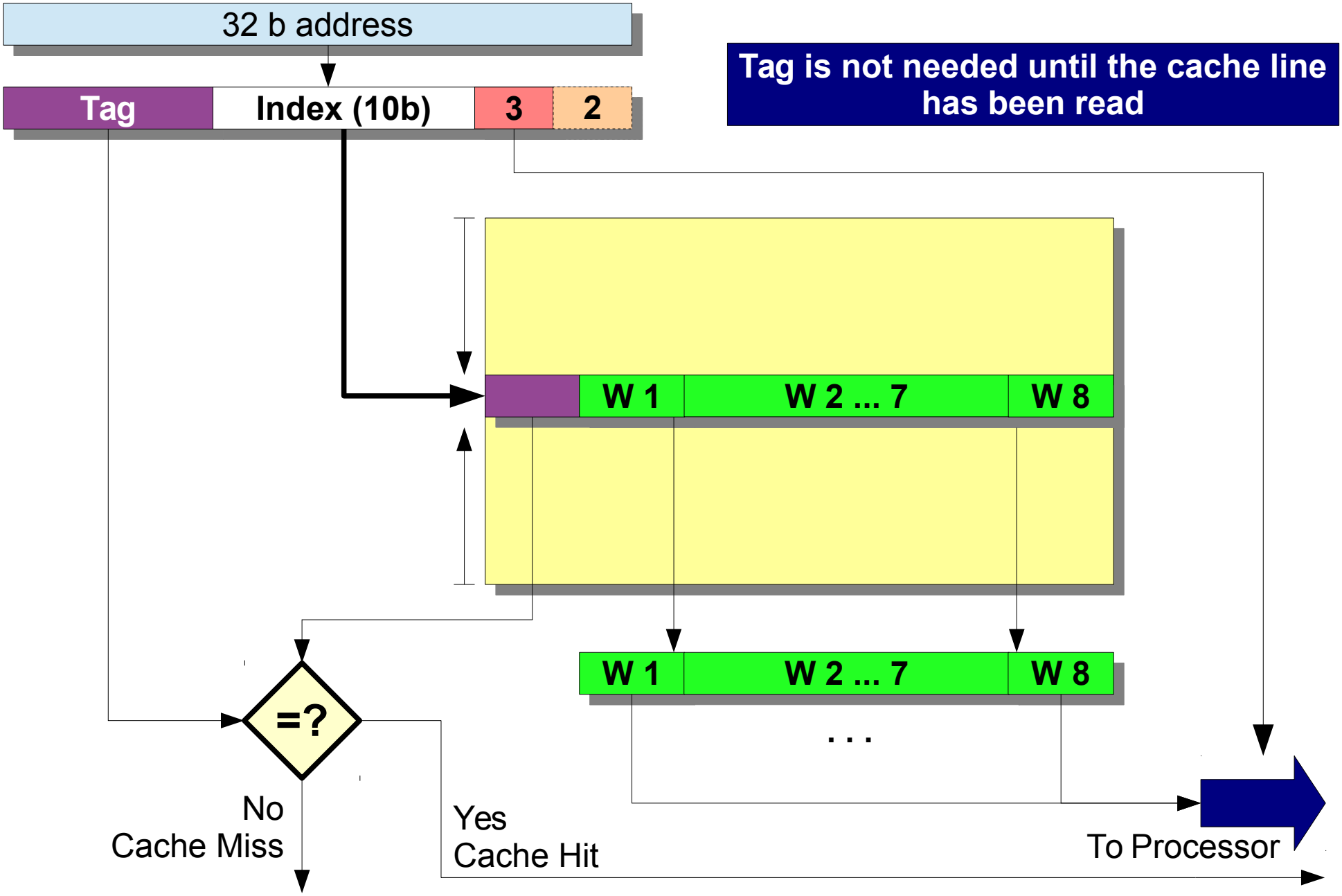
Recall – Cache Access

Direct mapped, 32 KB, 32B block, 32b main memory address



Recall – Cache Access

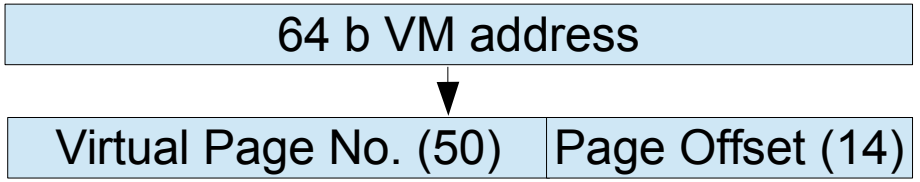
Direct mapped, 32 KB, 32B block, 32b main memory address



64 b VM address

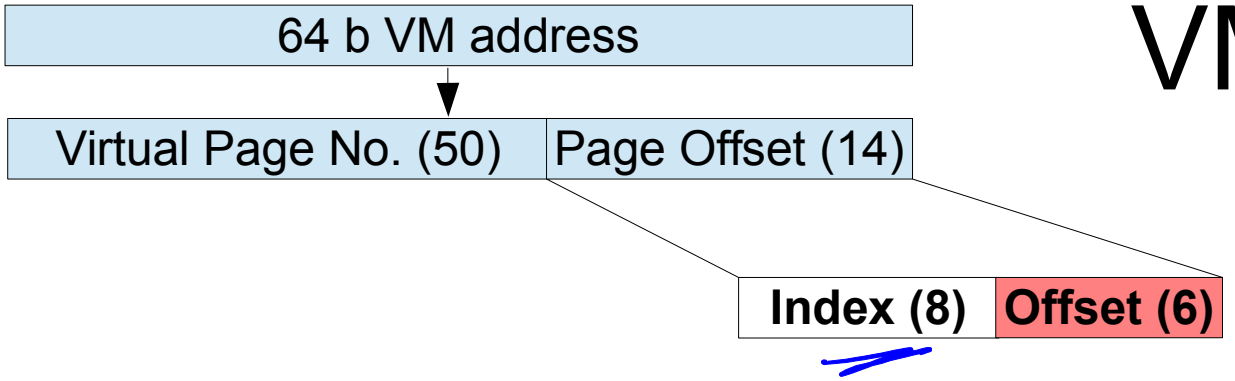
VM Example

VM Example

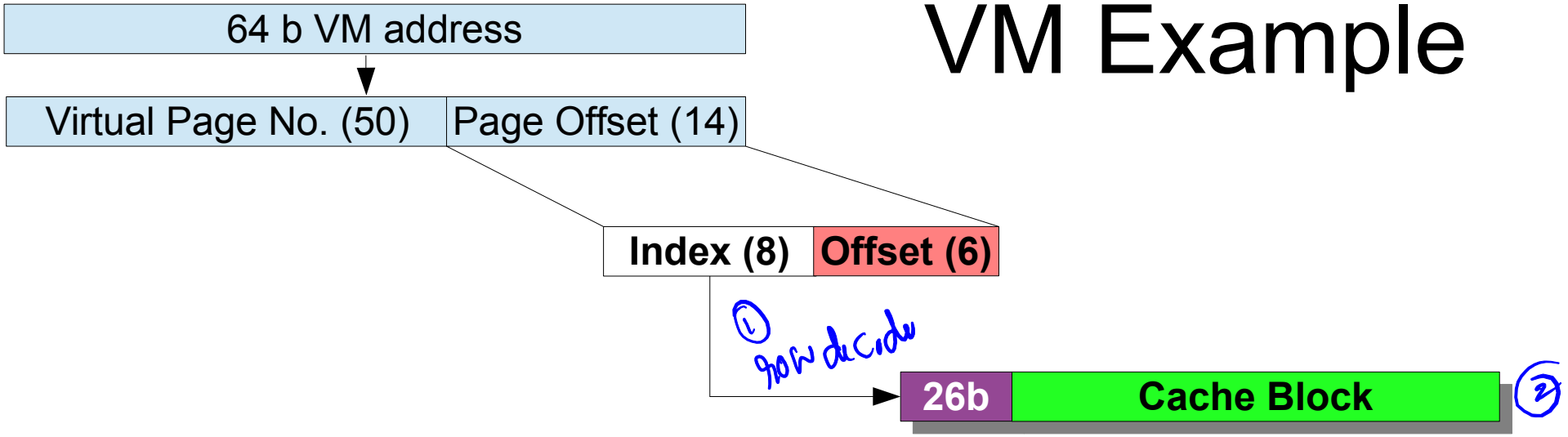


Page = 16KB

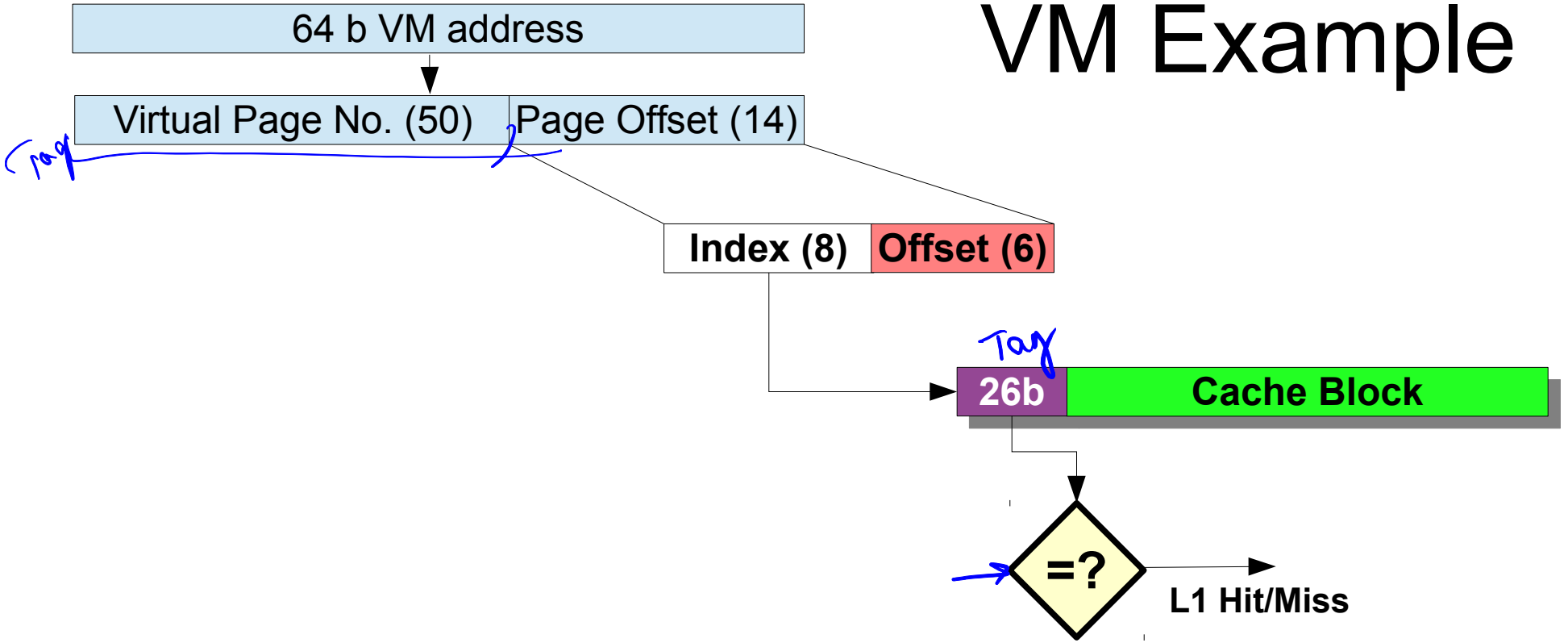
VM Example


64 b VM address
Virtual Page No. (50)
Page Offset (14)
Index (8)
Offset (6)

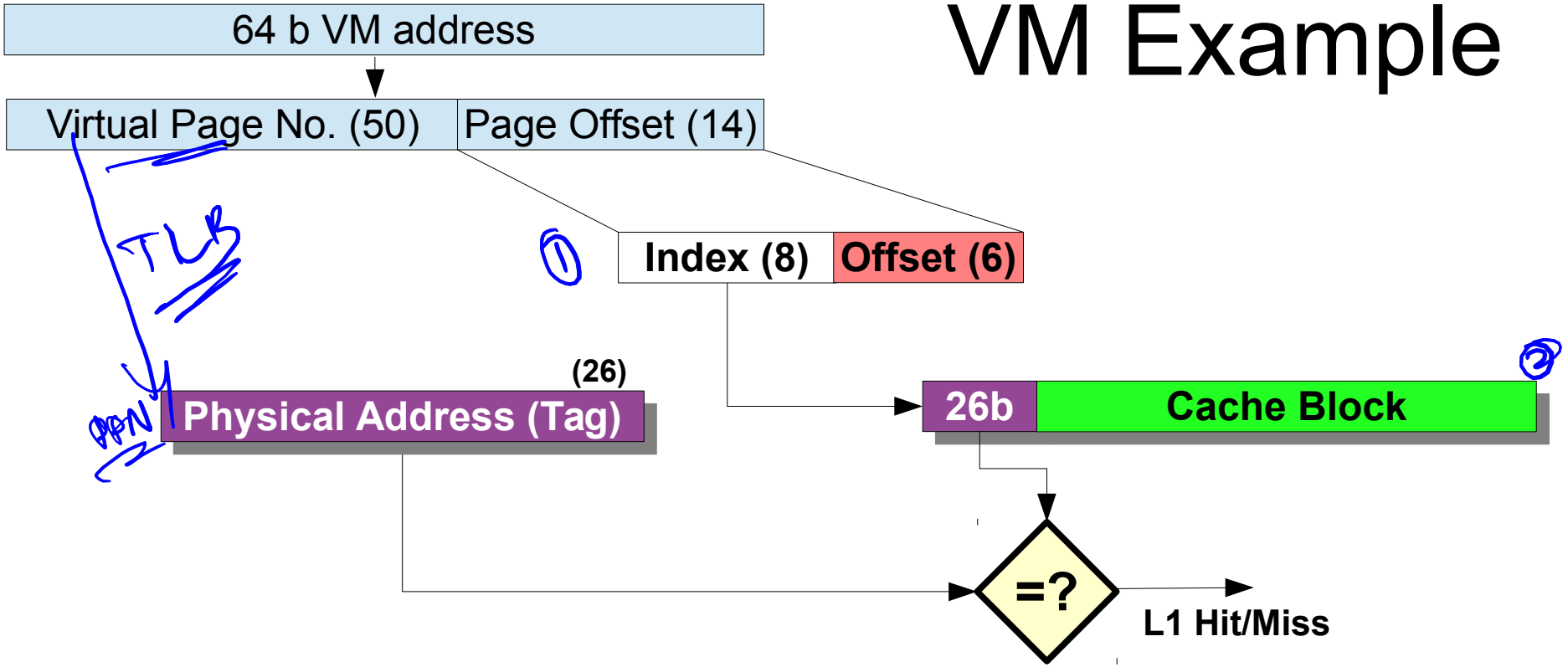
VM Example


64 b VM address
Virtual Page No. (50)
Page Offset (14)
Index (8)
Offset (6)
1
now decide
26b
Cache Block
2

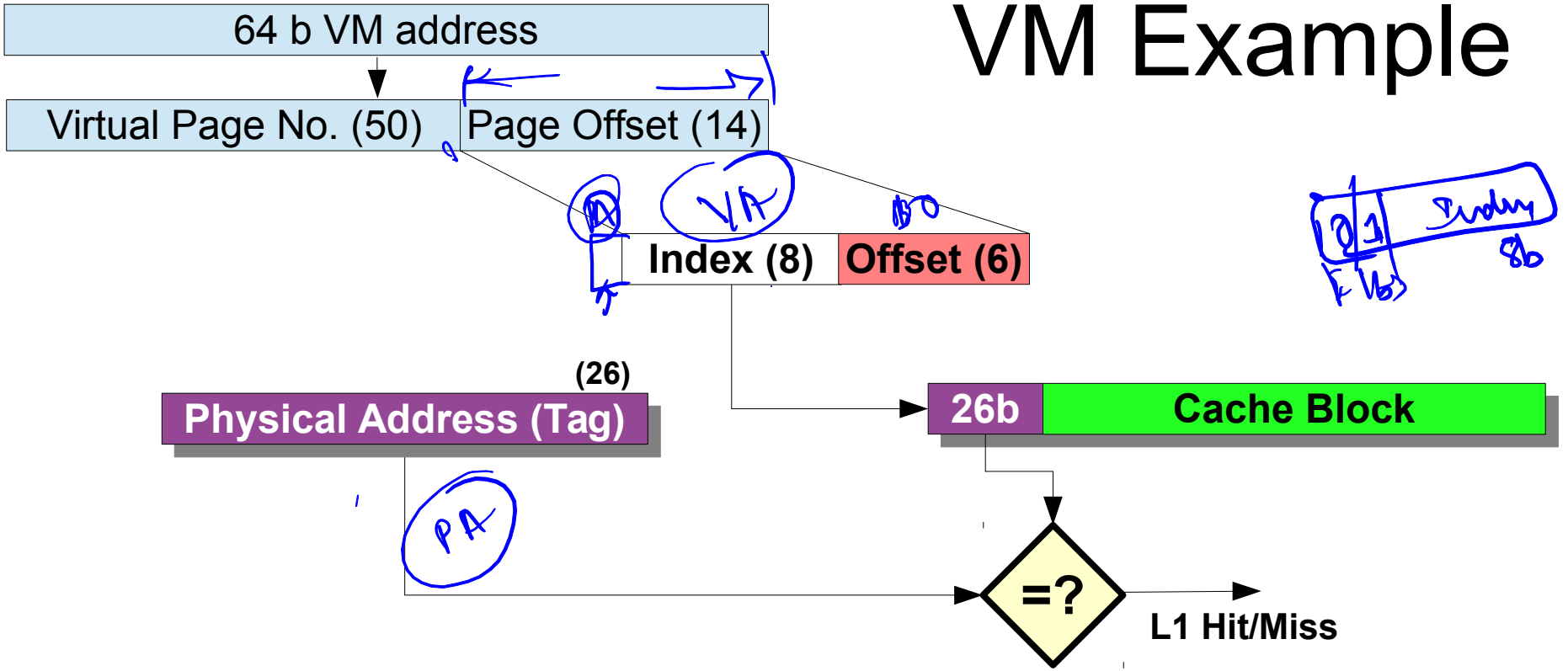
VM Example


64 b VM address
Virtual Page No. (50)
Page Offset (14)
Tag
Index (8)
Offset (6)
Tag
26b
Cache Block
=?
L1 Hit/Miss

VM Example


64 b VM address
Virtual Page No. (50)
Page Offset (14)
TLB
PPN
Index (8)
Offset (6)
(26)
Physical Address (Tag)
26b
Cache Block
=?
L1 Hit/Miss

VM Example

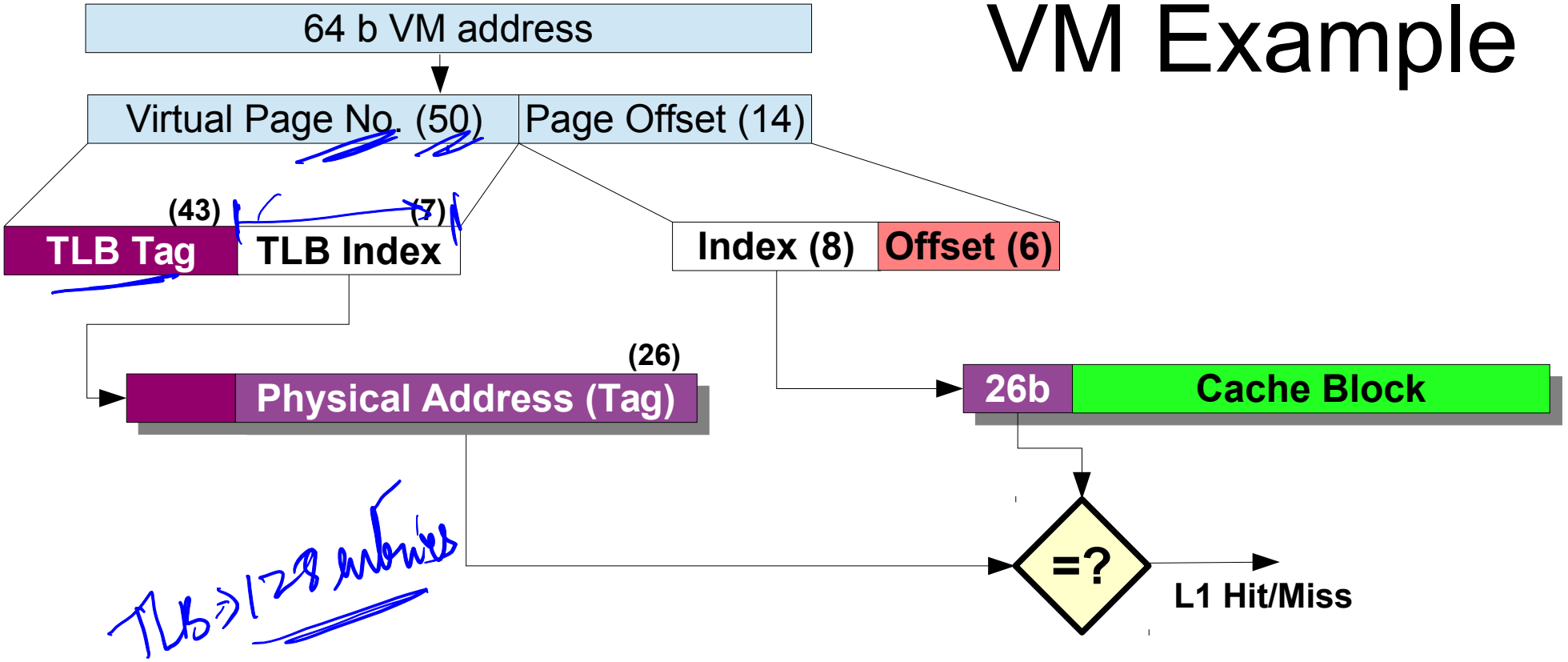


Index comes from the Virtual Address (Virtually Indexed)

Tag comes from the Physical Address (Physically tagged)

VIPT

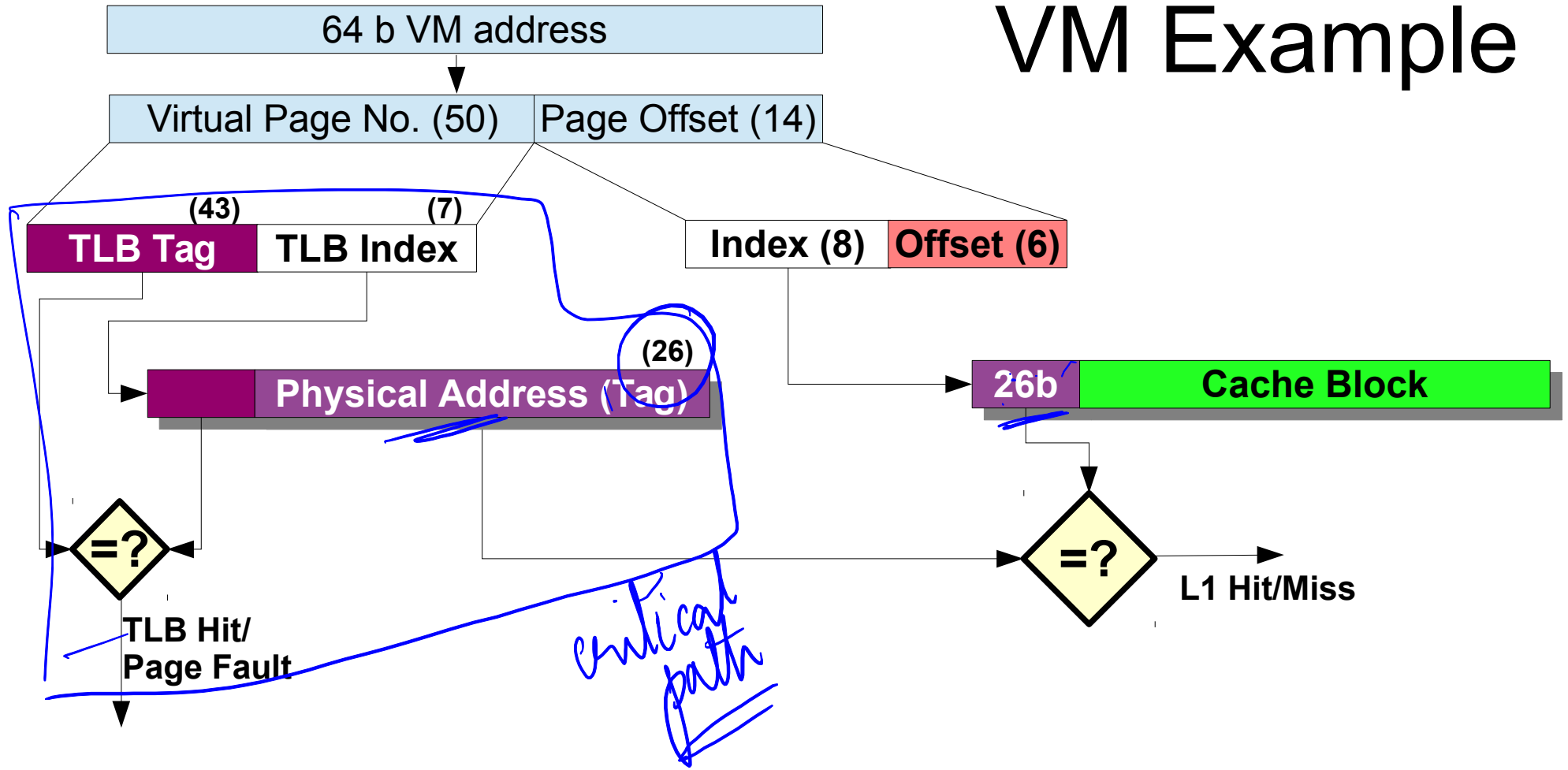
VM Example



Index comes from the Virtual Address (Virtually Indexed)

Tag comes from the Physical Address (Physically tagged)

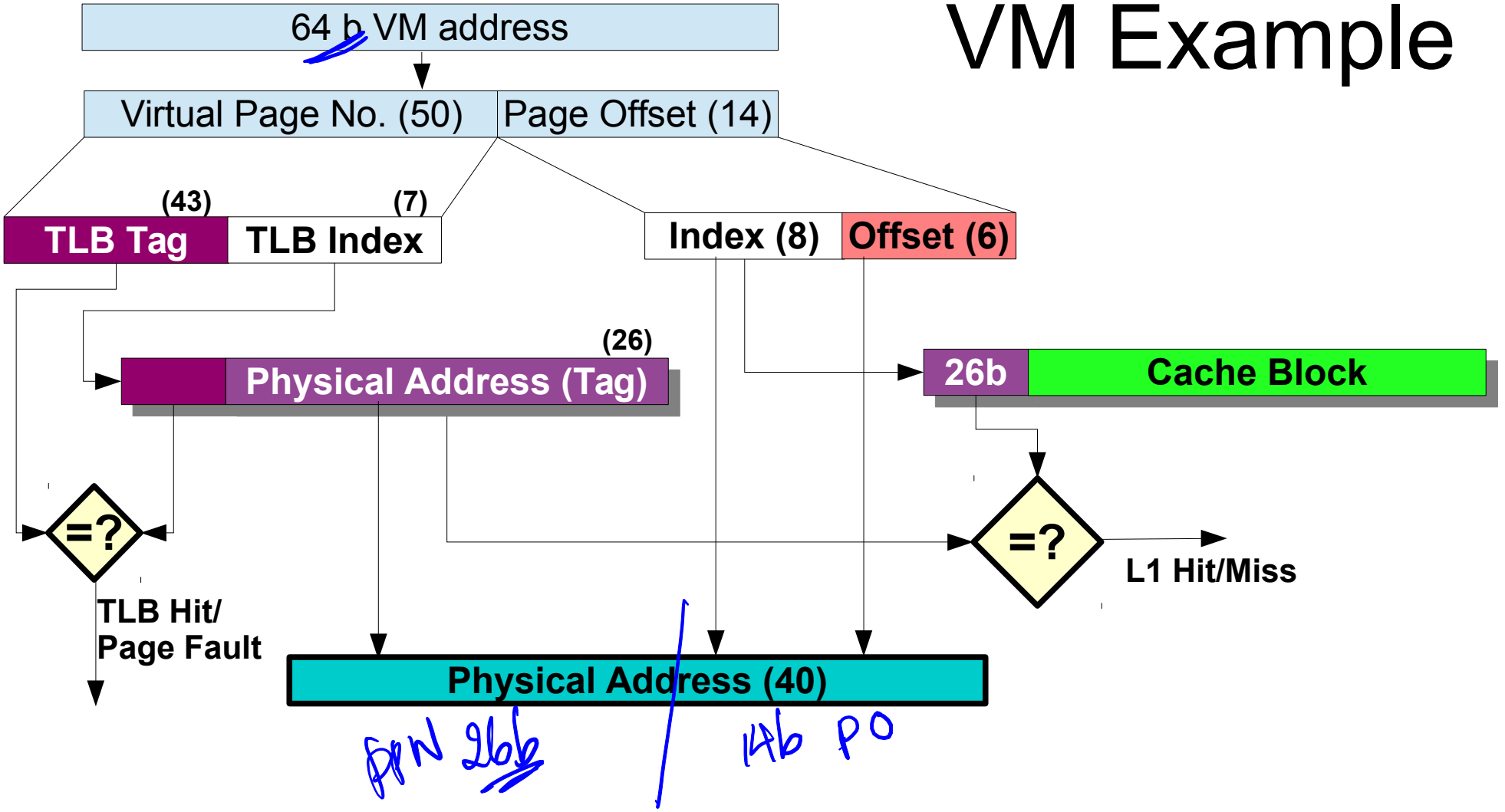
VM Example



Index comes from the Virtual Address (Virtually Indexed)

Tag comes from the Physical Address (Physically tagged)

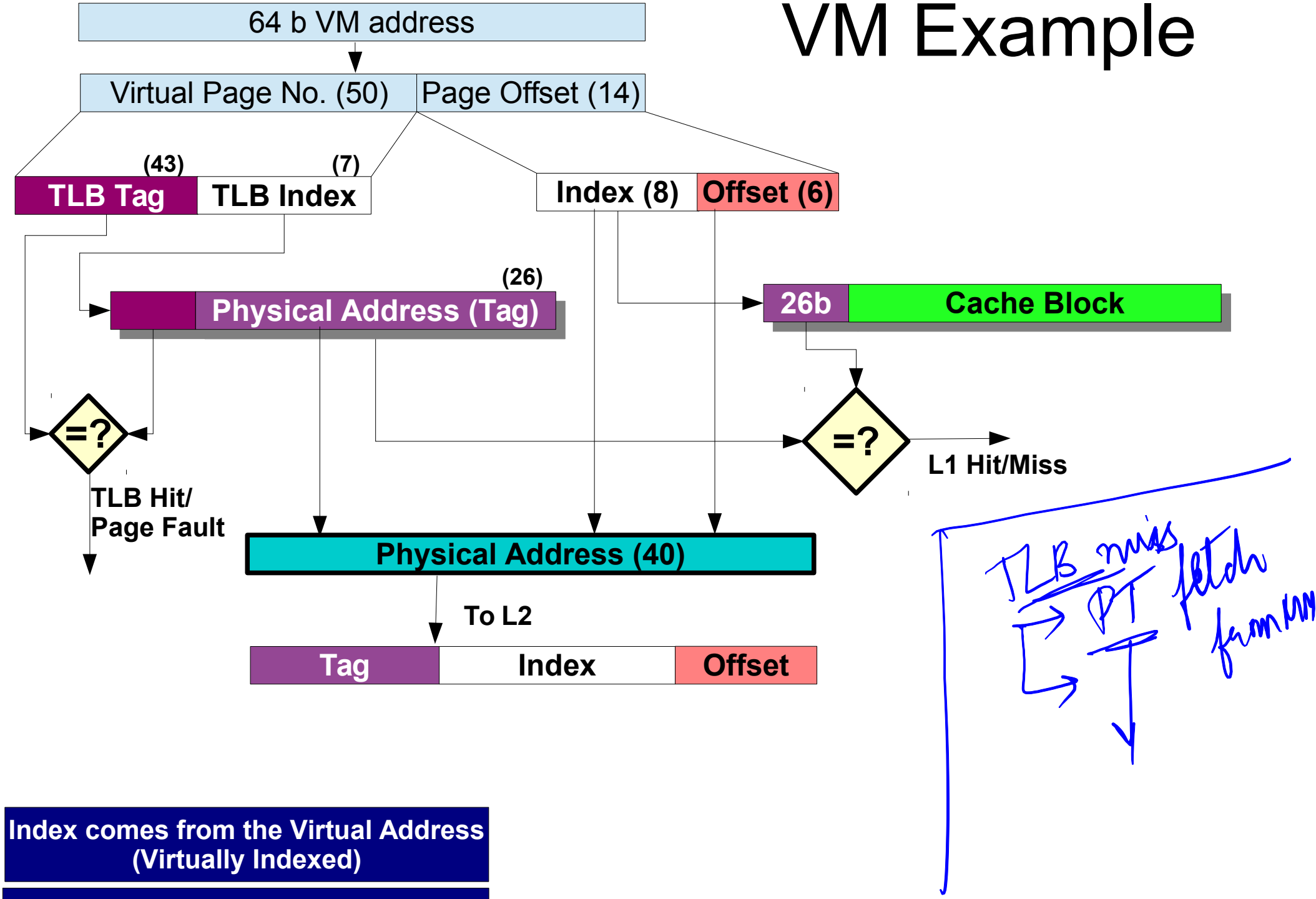
VM Example



Index comes from the Virtual Address (Virtually Indexed)

Tag comes from the Physical Address (Physically tagged)

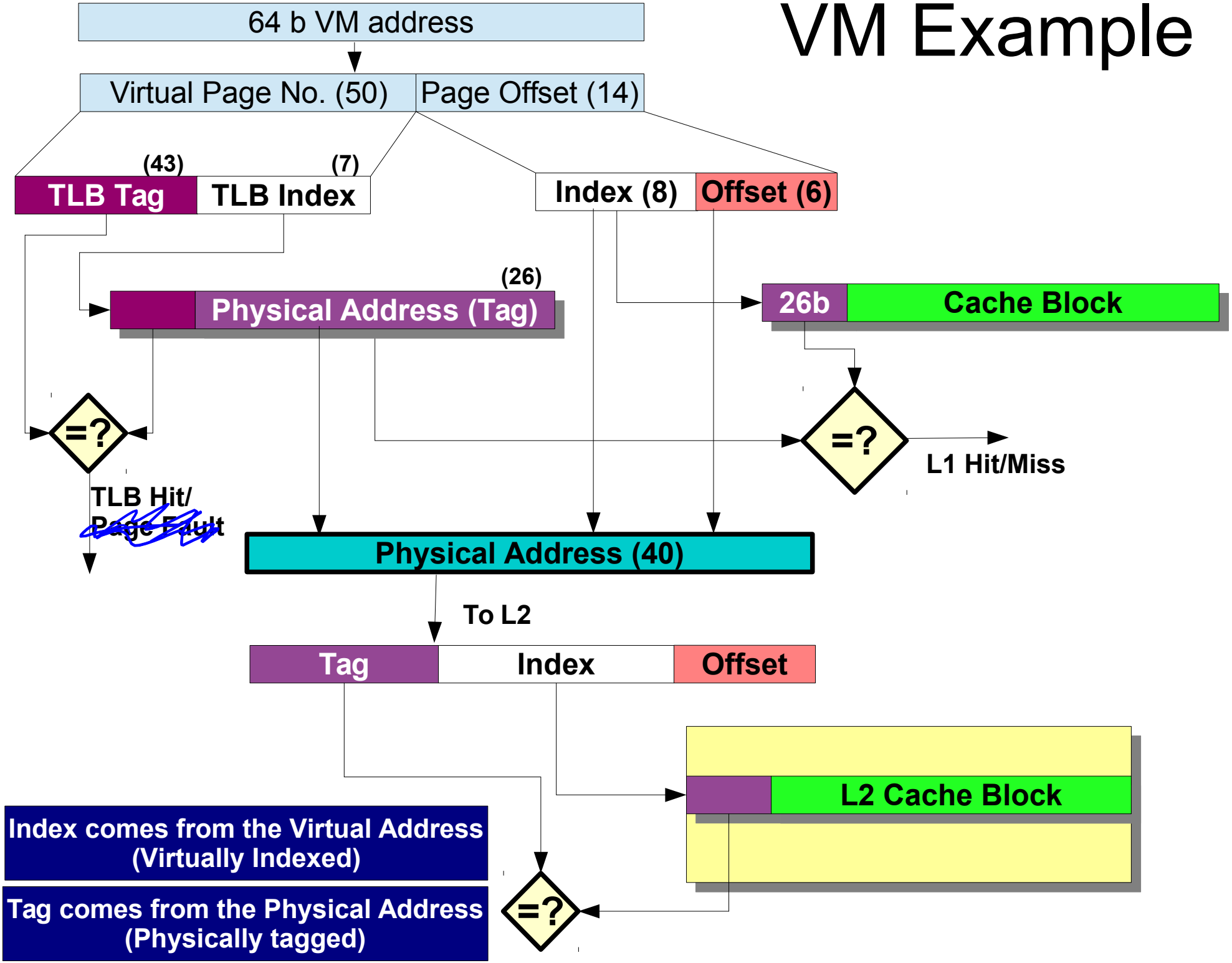
VM Example



Index comes from the Virtual Address (Virtually Indexed)

Tag comes from the Physical Address (Physically tagged)

VM Example


64 b VM address
Virtual Page No. (50)
Page Offset (14)
(43)
(7)
TLB Tag
TLB Index
Index (8)
Offset (6)
(26)
Physical Address (Tag)
26b
Cache Block
=?
=?
L1 Hit/Miss
TLB Hit/
Page Fault
Physical Address (40)
To L2
Tag
Index
Offset
L2 Cache Block
=?
Index comes from the Virtual Address
(Virtually Indexed)
Tag comes from the Physical Address
(Physically tagged)

Translation Lookaside Buffer

- Cache of page table mappings
- 32 – 4096 entries long
  - SA, FA, or DM
- Dirty flag – use during page write back
- Ref – used for LRU

VPN (tag)	PPN (data)	Valid	Ref	Dirty	Access Rights

Page Fault

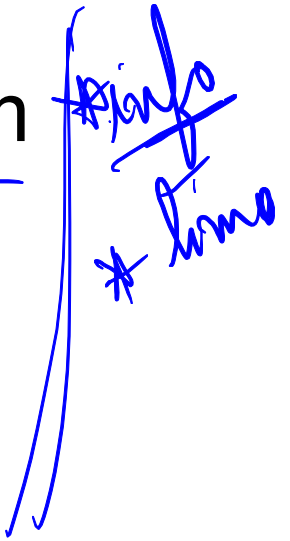
- Virtual address generated by processor is not available in main memory

Page Fault

- Virtual address generated by processor is not available in main memory
- Detected on attempt to translate address
  - Page Table entry is invalid

Page Fault

- Virtual address generated by processor is not available in main memory
- Detected on attempt to translate address
  - Page Table entry is invalid
- Must be `handled' by operating system
  - Identify slot in main memory to be used
  - Get page contents from disk
  - Update page table entry
-

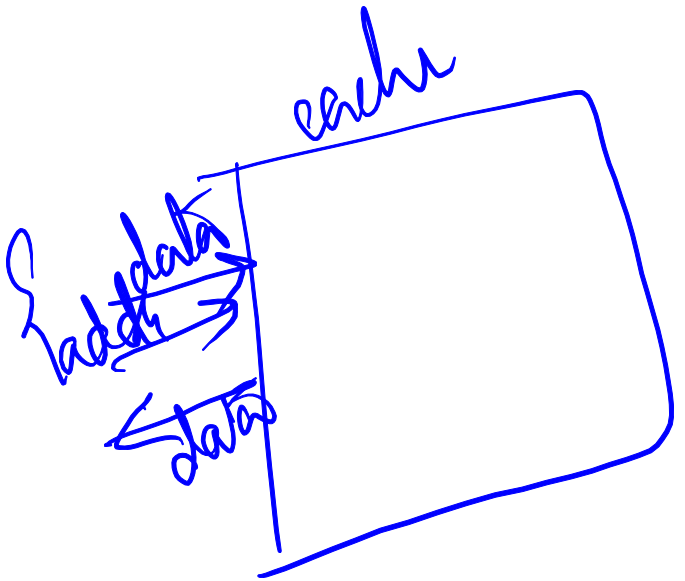


Page Fault

- Virtual address generated by processor is not available in main memory
- Detected on attempt to translate address
  - Page Table entry is invalid
- Must be `handled' by operating system
  - Identify slot in main memory to be used
  - Get page contents from disk
  - Update page table entry
- Provide data to the processor

Abstraction: Virtual vs. Physical Memory

- Programmer sees virtual memory
  - Can assume the memory is “infinite”



Abstraction: Virtual vs. Physical Memory

- Programmer sees virtual memory
  - Can assume the memory is “infinite”
- Reality: Physical memory size is much smaller than what the programmer assumes

Abstraction: Virtual vs. Physical Memory

- Programmer sees virtual memory
  - Can assume the memory is “infinite”
- Reality: Physical memory size is much smaller than what the programmer assumes
- The system (system software + hardware, cooperatively) maps virtual memory addresses are to physical memory
  - The system automatically manages the physical memory space transparently to the programmer

Abstraction: Virtual vs. Physical Memory

+ Programmer does not need to know the physical size of memory nor manage it → A small physical memory can appear as a huge one to the programmer → Life is easier for the programmer

-- More complex system software and architecture

A classic example of the programmer/(micro)architect tradeoff

Virtual Memory

- What is the size of the Page Table?
- Where is it stored?
- What factors decide the size of a page?
  - What are its side effects?
- Page size is constant/variable?
-