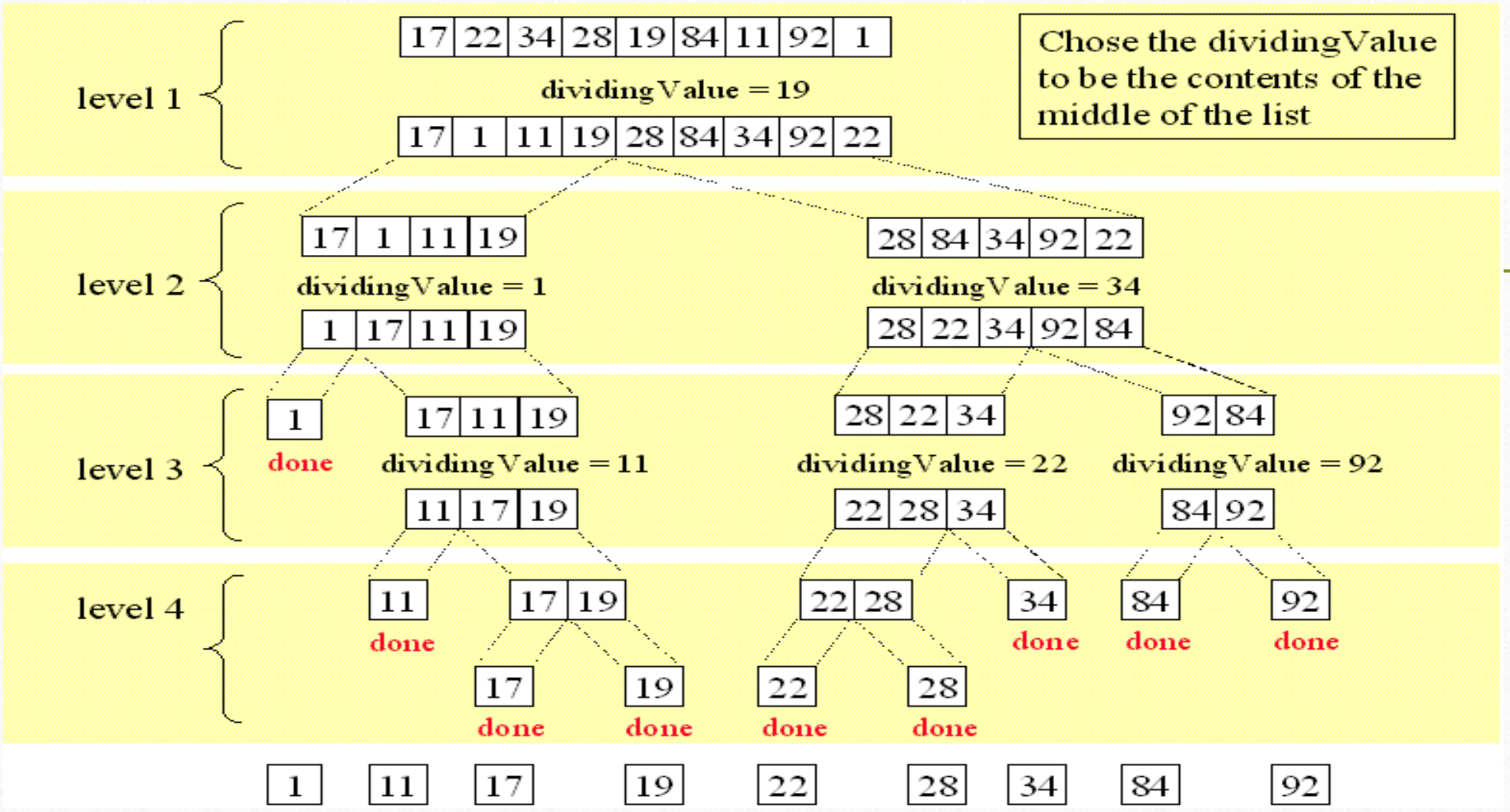
Memory

Part 3

Final Exam Review Session

- **Final Exam Review session**
- **A) Monday morning 10:30am Friday AUG 07**
- **B) Monday morning 10:30am Monday AUG 11th**
- **C) I cannot attend either due to Exams at Both Times**

Quicksort Algorithm



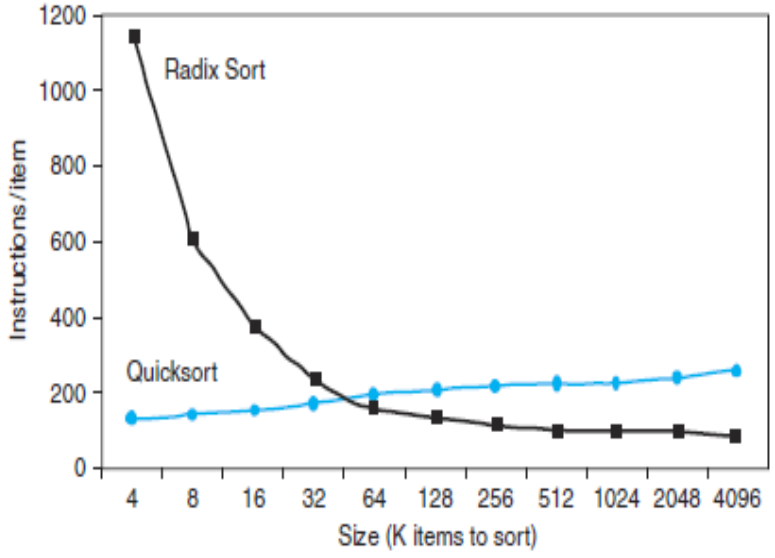
http://www.sethi.org/classes/ecet370/labs_and_projects/lab_2/quicksort/

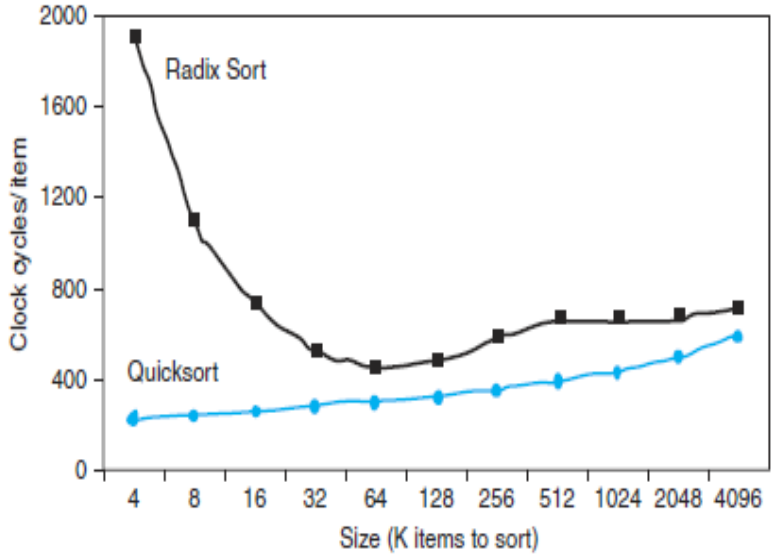
Radix Sort:

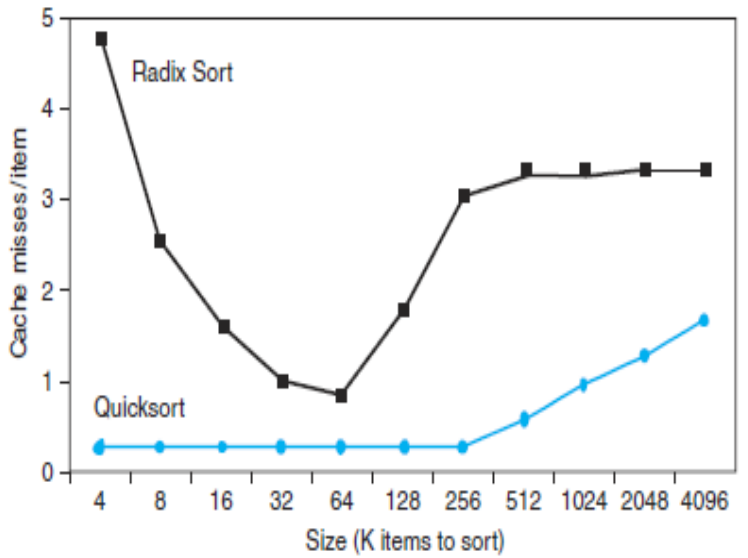
- https://www.youtube.com/watch?v=YXFI4osELGU
- Sorting A list of Numbers:
  - based on successively examining each digit.
  - Repeatedly need to examine the entire list

Sorting Example

Impact of cache on performance







Why does Quicksort have less cache misses than Radix sort

- A)Radix sort needs to pass the entire list repeatedly therefore a big list cannot be stored all in cache
- B) Quicksort needs only to work on smaller portions of the original list,

  Successively working on subsets of the original list.
- C) Quicksort we need to pass again and again through the entire list , similar to radix sort, however this is done recursively
- D) Radix sort only examines n times the number of digits therefore this makes cache misses rise
- E) both A and B

In Class Example : Total Number of Clock Cycles Including Memory Access

```
addi $1, $0, 100
sub $4, $2, $6
addi $2, $0, 0
lw $5, 0,($4)
add $2, $5, $2
addi $4, $4, 4
subi $1, $1, 5
bne $1,0, -5
sll $0,$0,0
```

VIRTUAL MEMORY

- Layer between RAM and DISK
- Every Program/Process on your Computer thinks that it has unlimited amount of memory space *(upper limit only based on address space, 32 bits or 64 bits)*

Virtual Address
Is word Addressed

- **NOTE:** Course notes examples ignore that fact that lowest 2 bits should always

Be zero. (Subsequent examples)

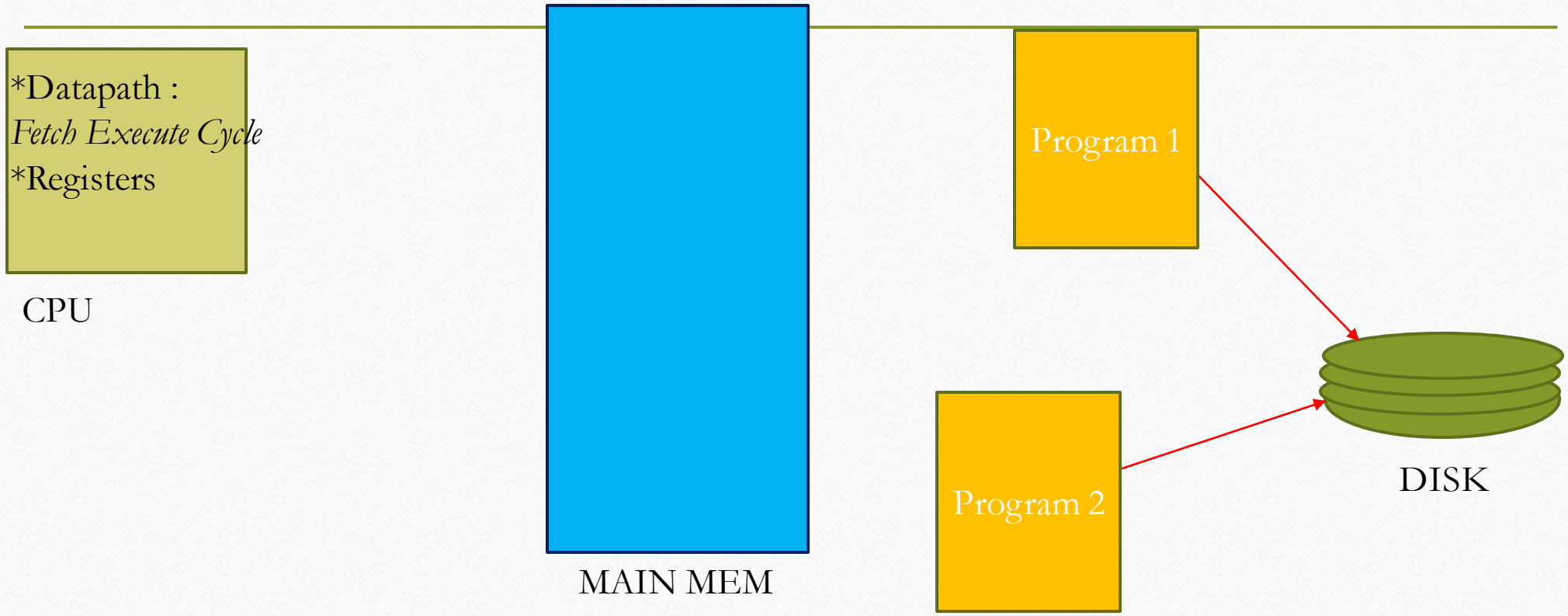
- **0000 1111 0101 0111 0000 0000 0000 1100**
- **Therefore :Physical 30 bit addresses should also be word addressed**

- As we saw previously in cache related Addresses: Lowest two bits are zero
- To make these word addressed. This is called the byte offset of the 32 bit address.

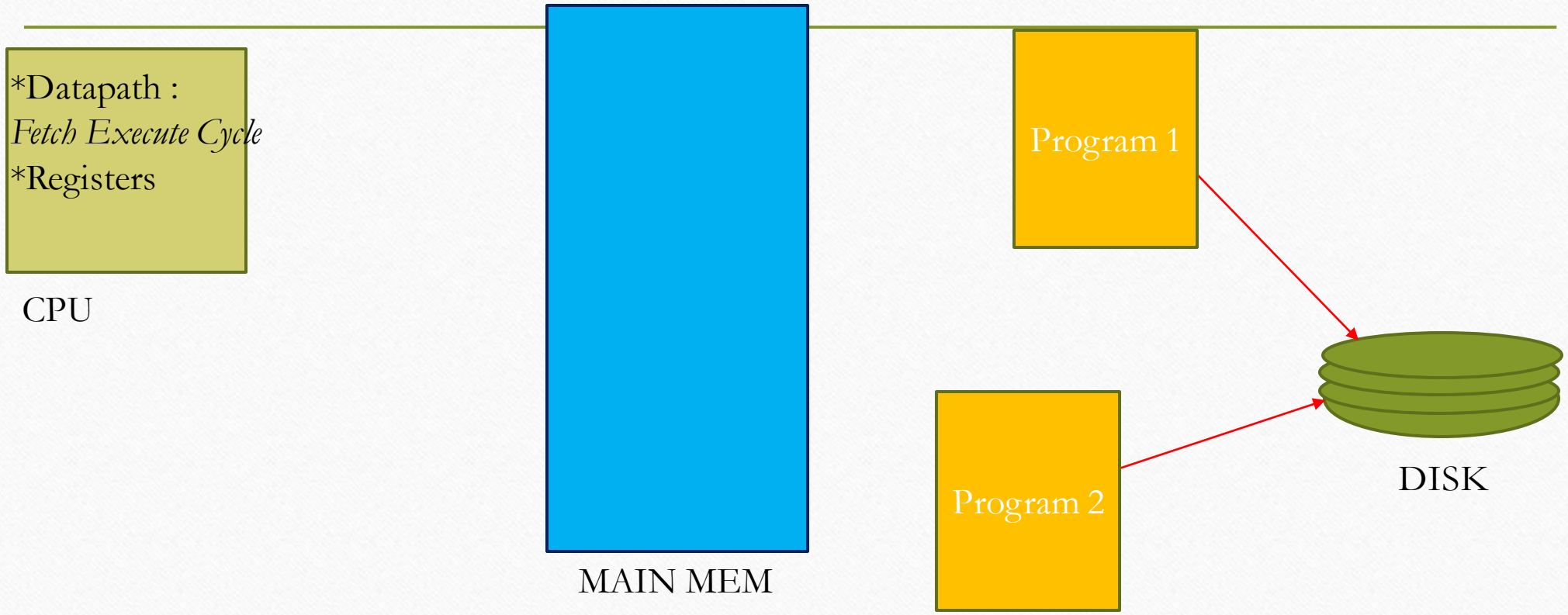
Virtual Address
Is word Addressed

- **Therefore :Physical 30 bit address space in our examples**
- Physical Address space may be less than the Address space that a program thinks it has.
  - 32 bit address space available to all programs 4GB
  - RAM may only be 1 GB ➔ 30 bits
  - This is the examples we use in class

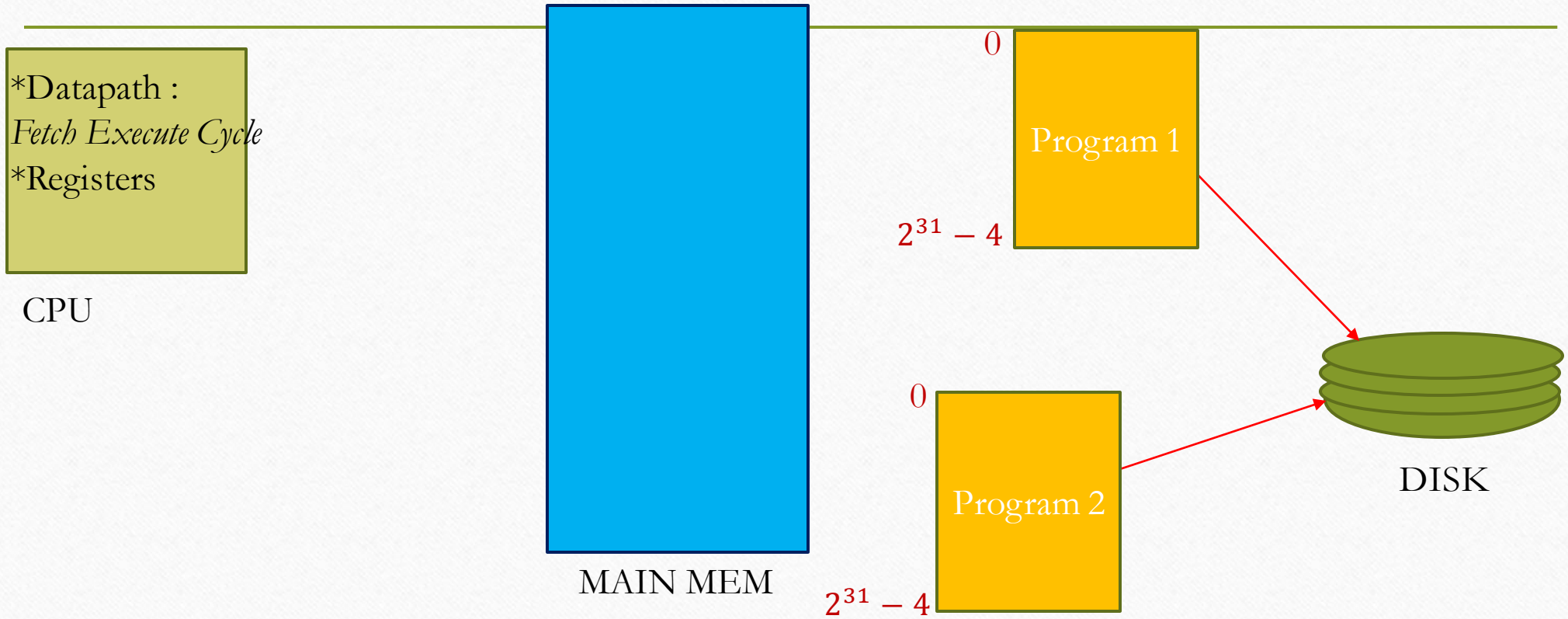
Initially Program 1, and Program 2 loaded on disk. Actually Disk has many different programs



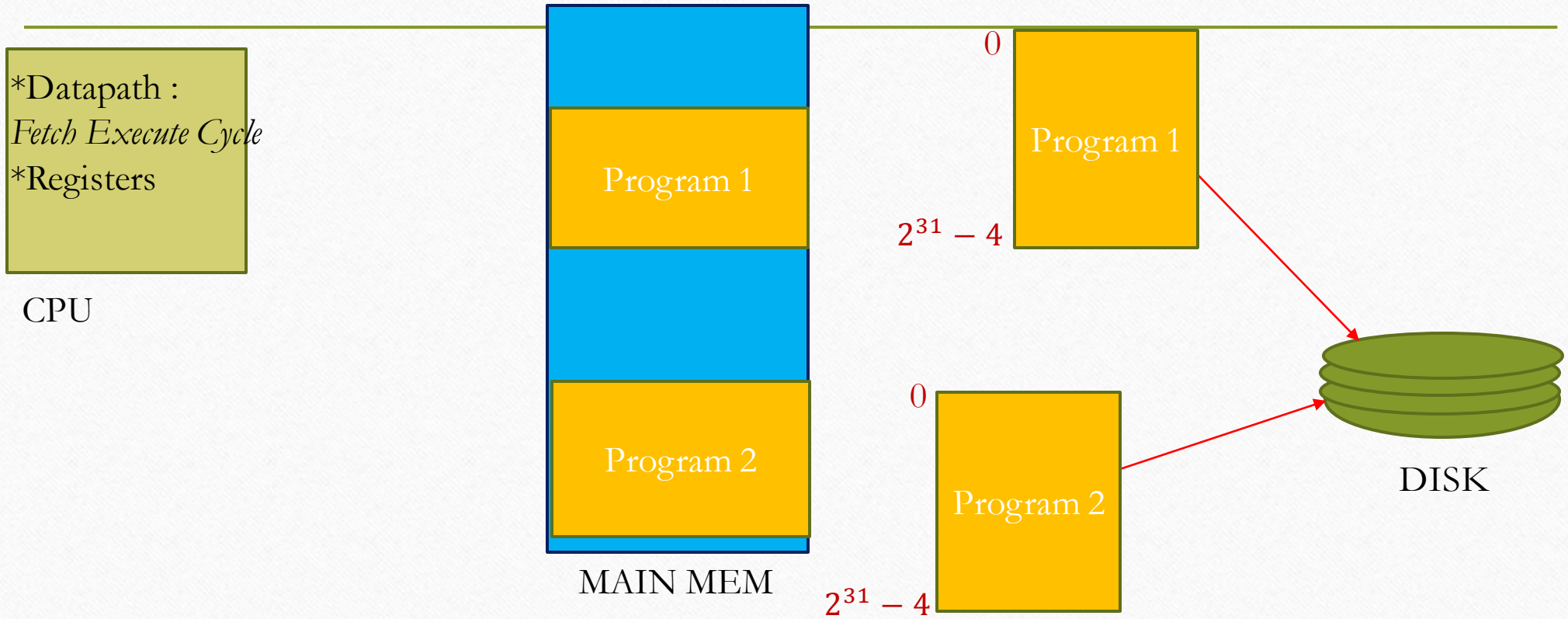
When we click on a Program that gets loaded into Main Memory (some of it)



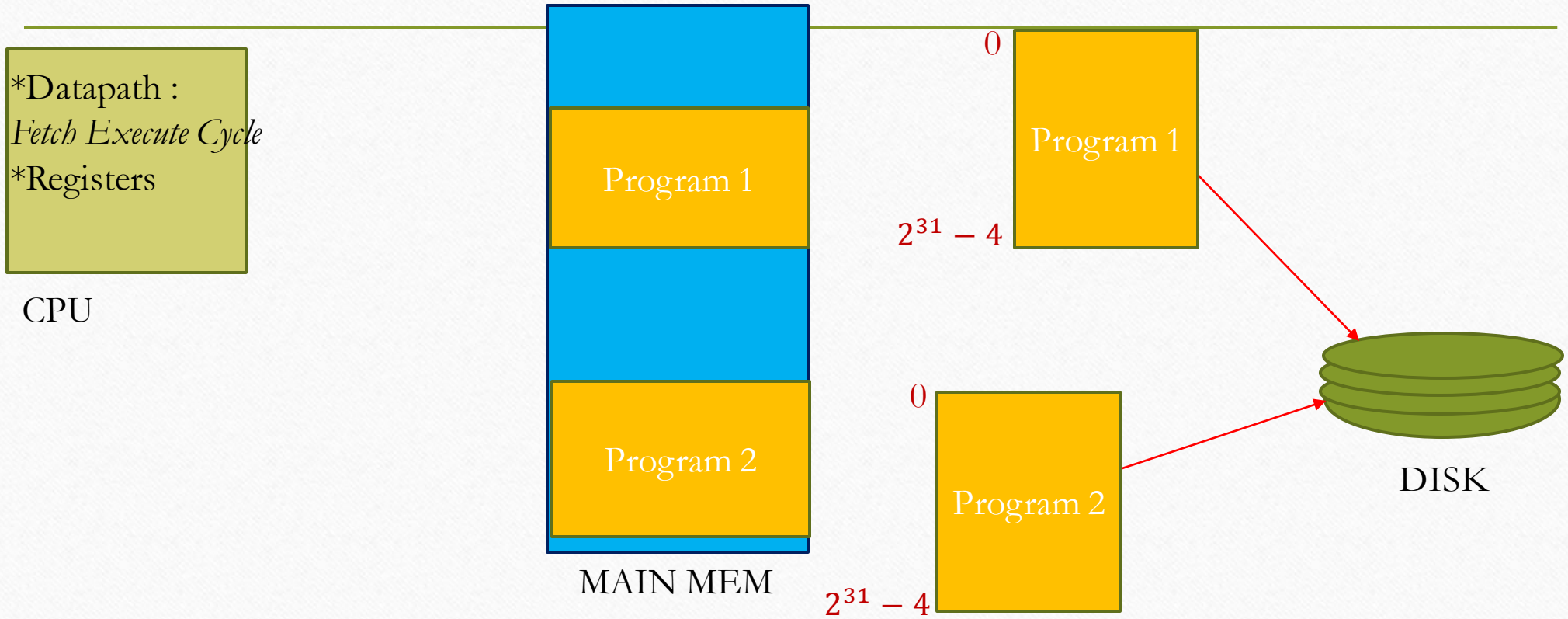
Every program has an address space.
Begins at 0 ends somewhere $< = 2^{31} - 4$



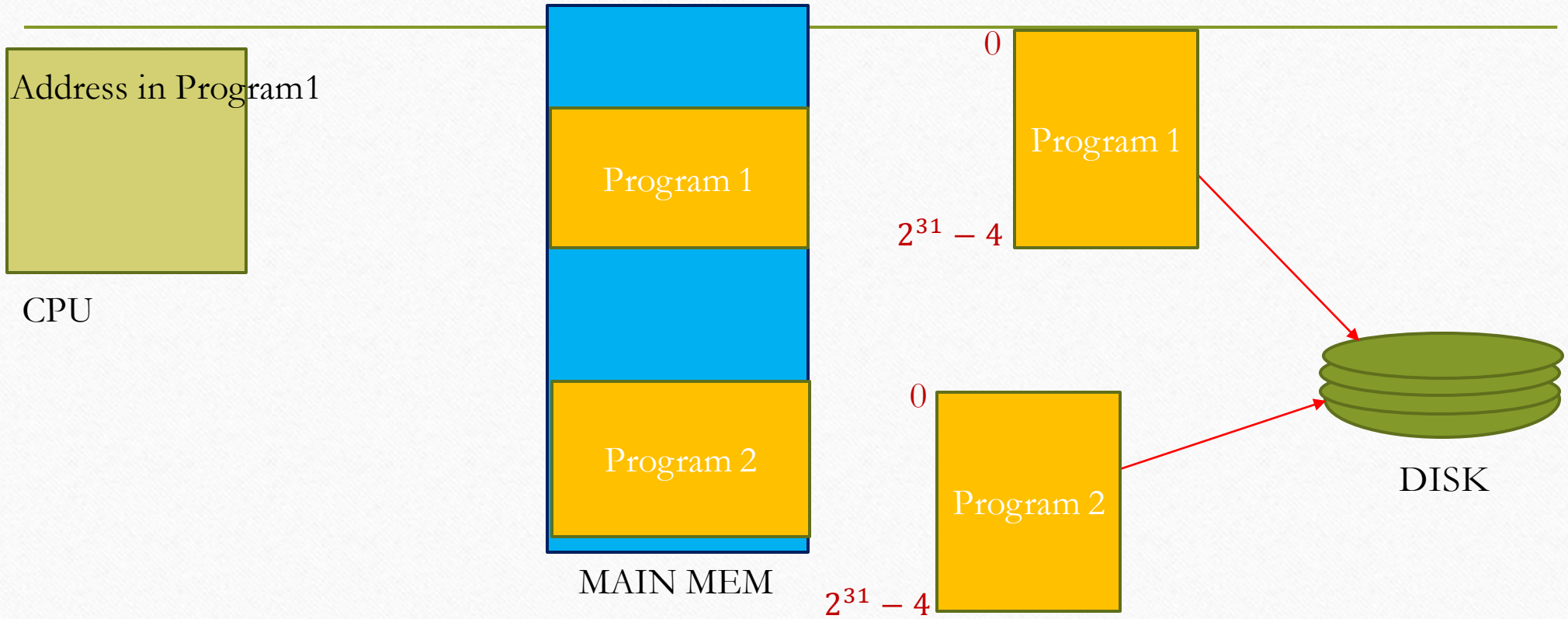
The Processor will swap between multiple programs.
Therefore both Prgm1 and Prgm2 will have some portions loaded into RAM



The Processor will swap between multiple programs.
Therefore both Prgm1 and Prgm2 will have some portions loaded into RAM

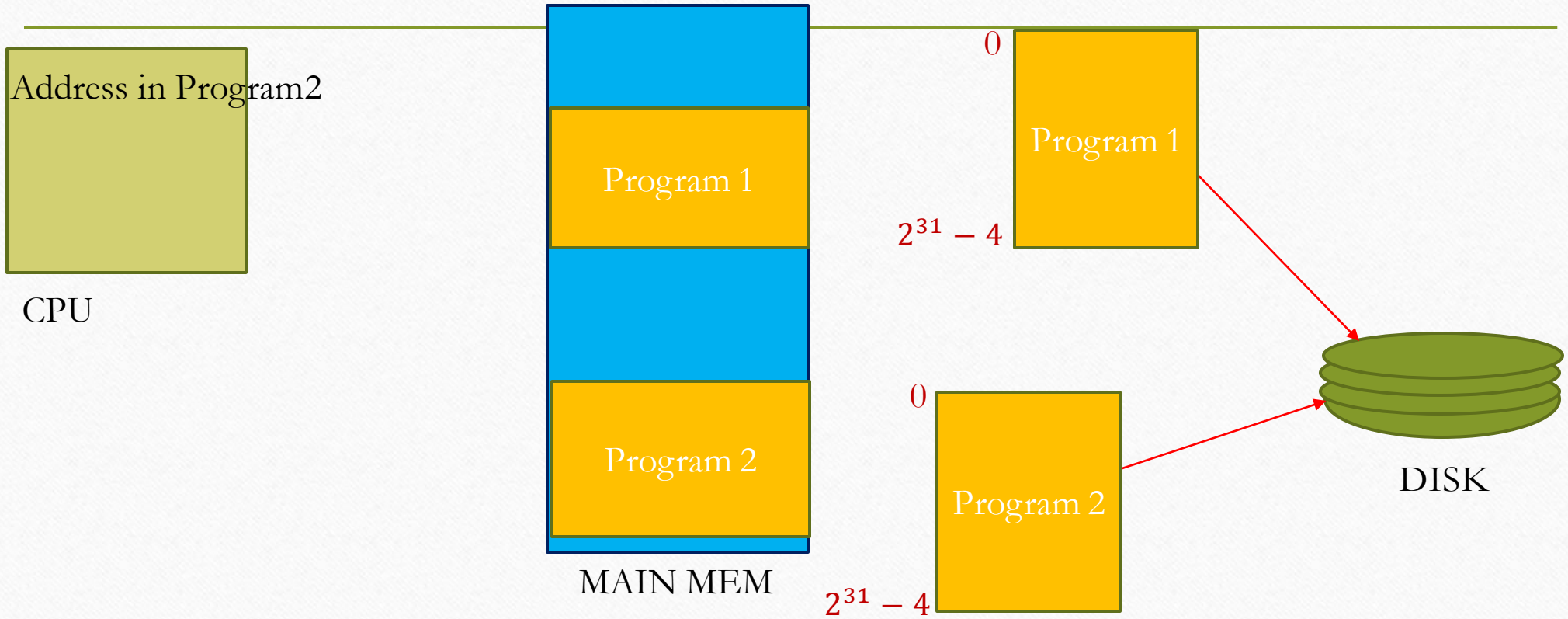


The Processor will swap between multiple programs.
Therefore both Prgm1 and Prgm2 will have some portions loaded into RAM

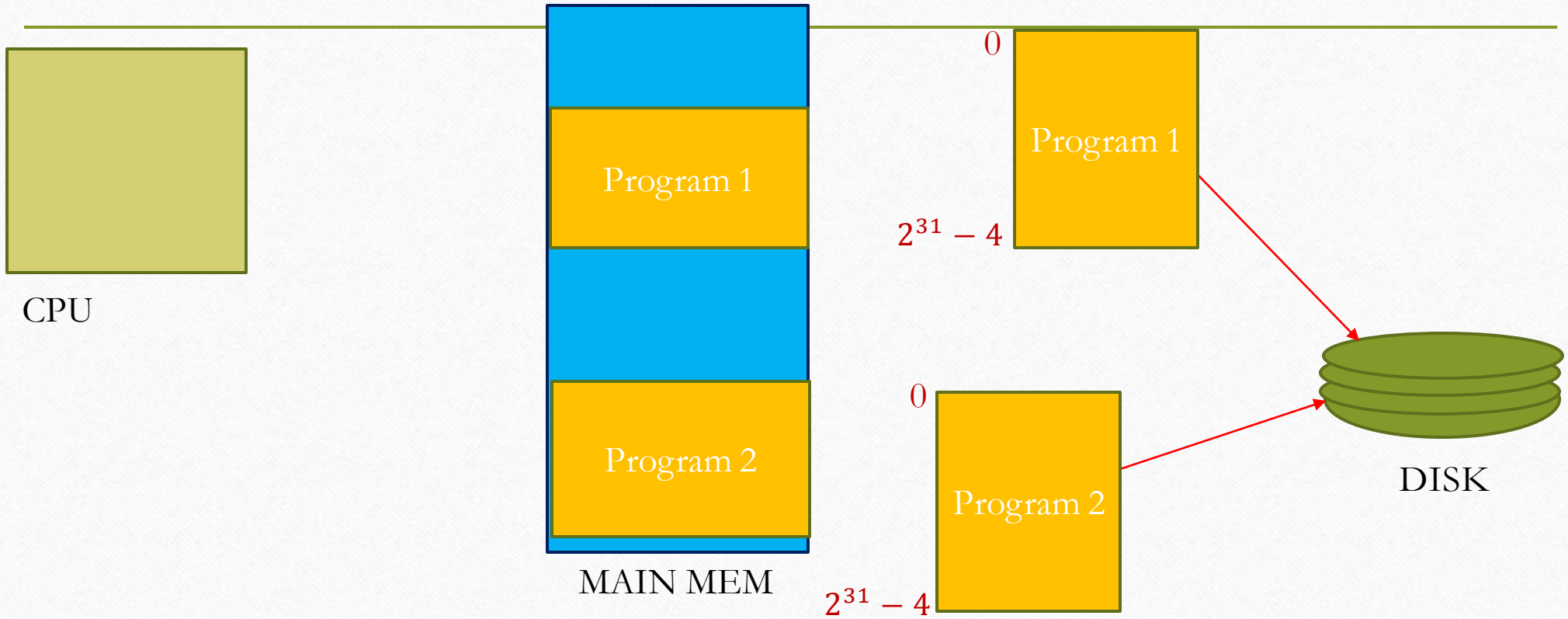


How will processor know where to get data for Program1 and Program2

Also, both programs are using the same address space.

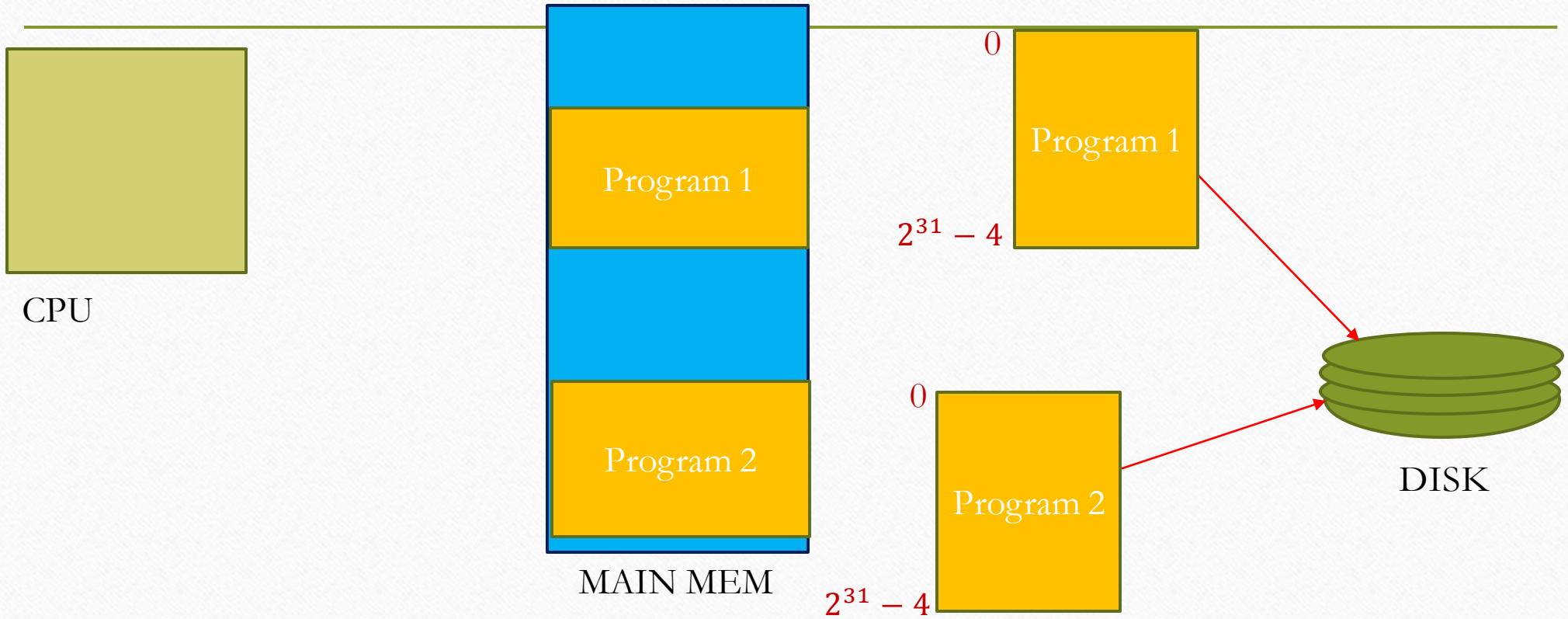


Virtual Memory: Let the two programs use the same Virtual Address Space $\mathbf{0 \ldots 2^{31} - 4}$



Virtual Memory: Let the two programs use the same Virtual Address Space $\mathbf{0 \ldots 2^{31} - 4}$

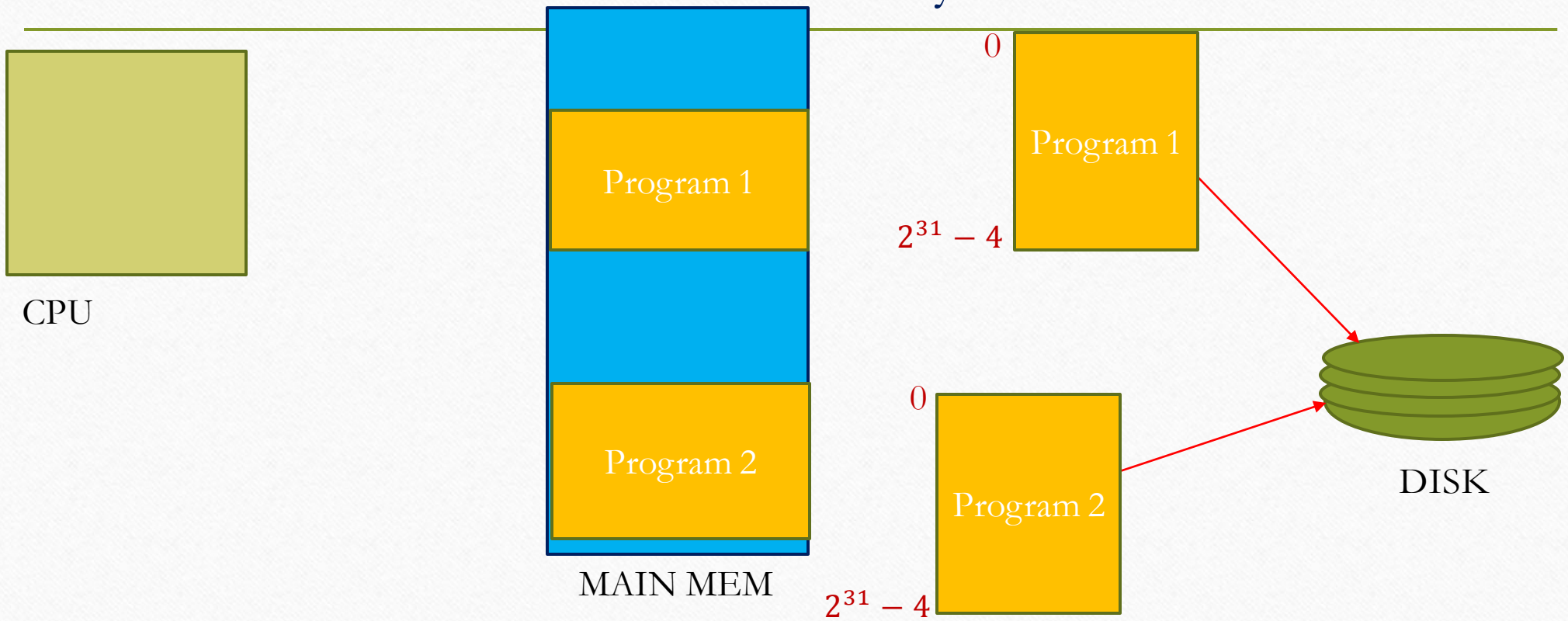
We will need a Virtual Address To Physical Address Mapping



We will need a Virtual Address To Physical Address Mapping

Page Table For Each Program.

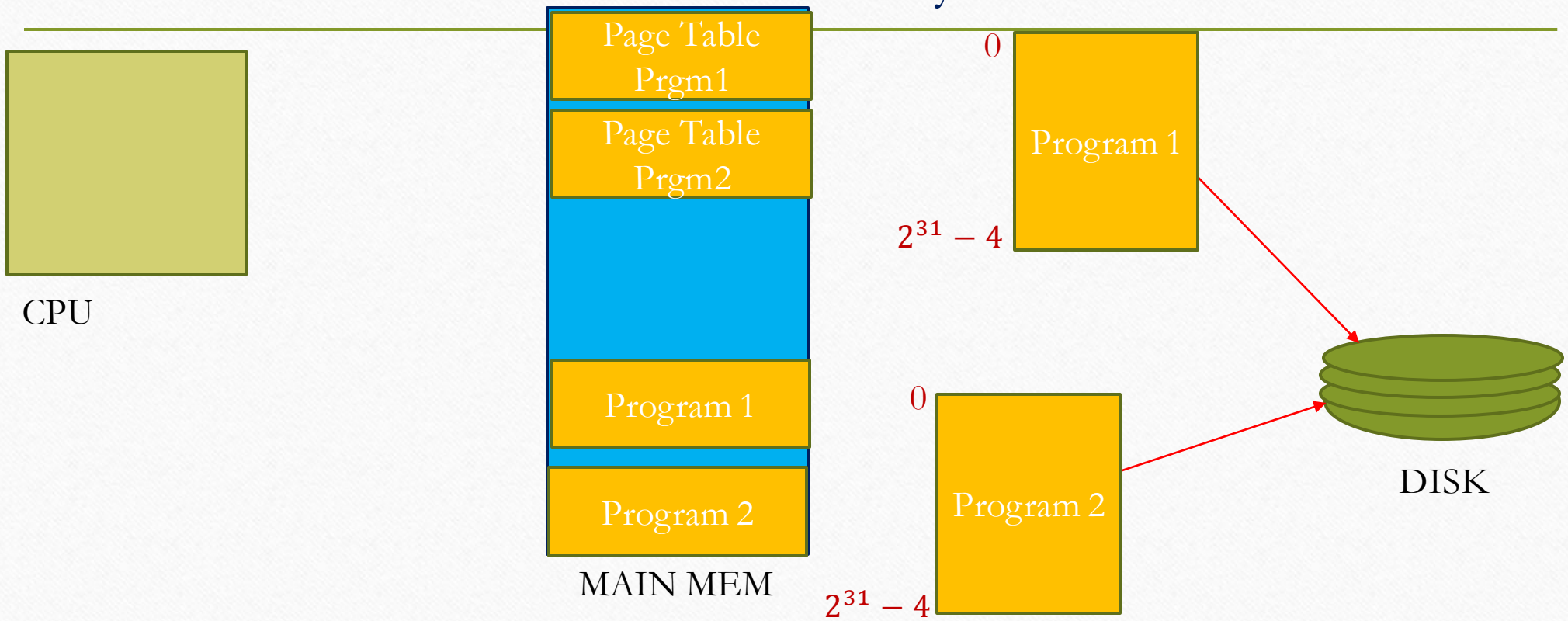
Page Table will map program1 to where it is physically located in Main Memory



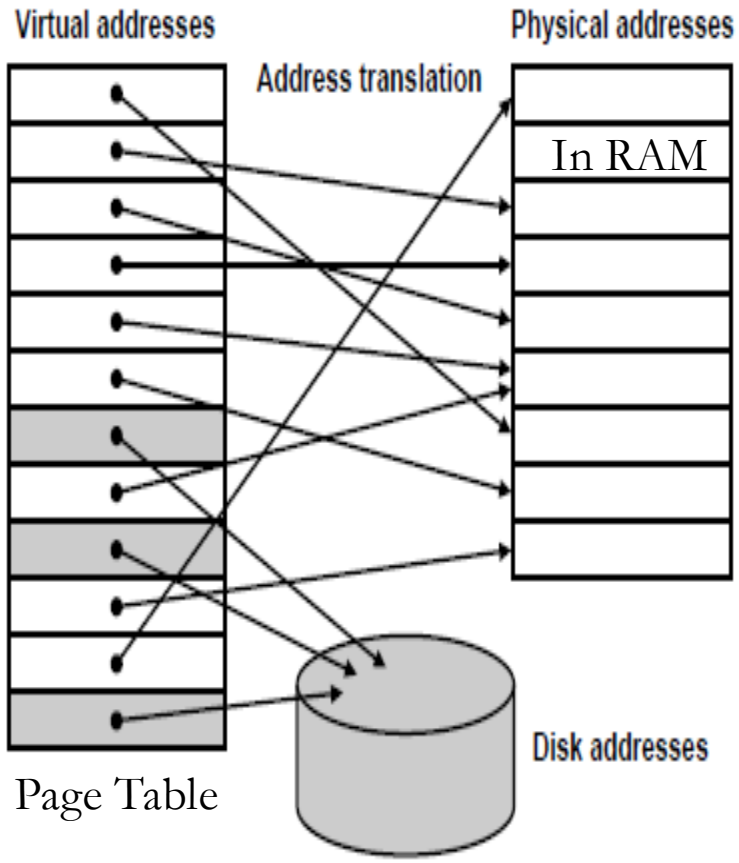
We will need a Virtual Address To Physical Address Mapping

Page Table For Each Program.

Page Table will map program1 to where it is physically located in Main Memory



Virtual Memory Mappings



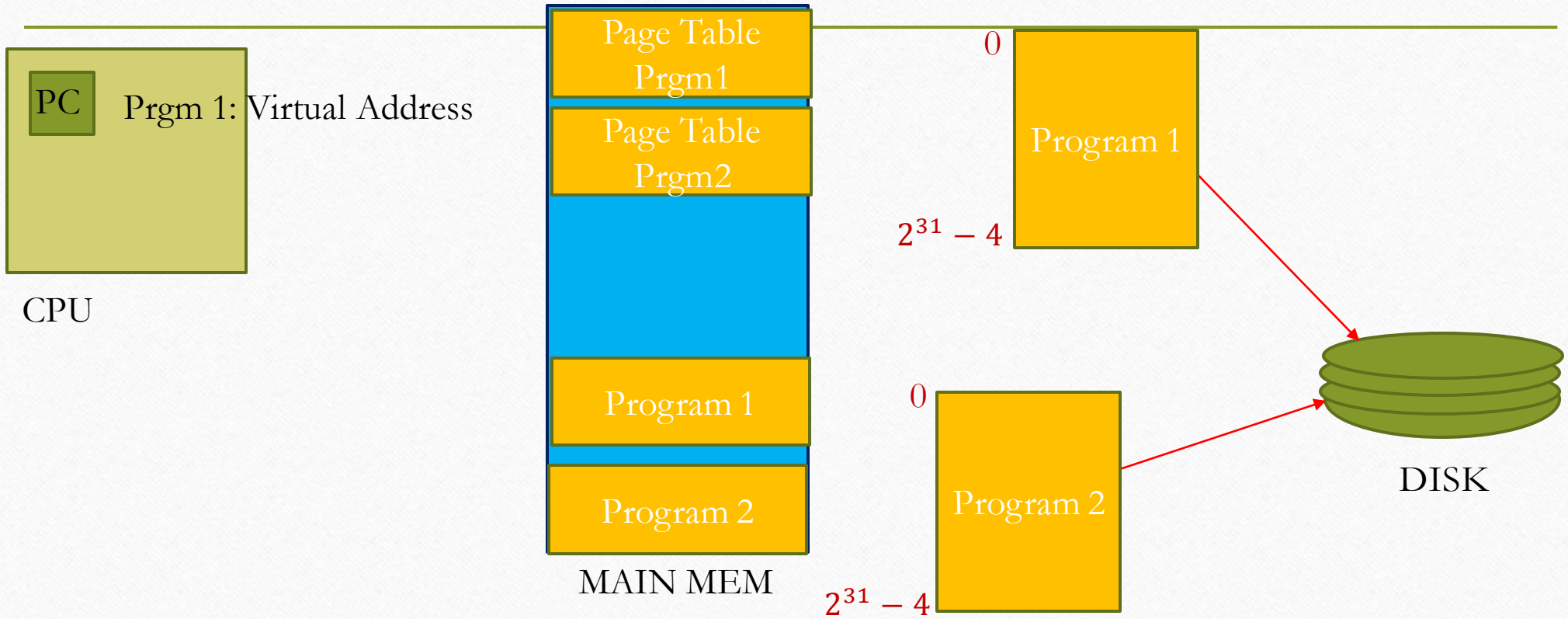
Every Program will have its own Page Table
To Map virtual addresses of that program to Physical locations
In Main Memory – RAM

As multiple programs are loaded in RAM,
Their respective Page Tables are created and stored in RAM

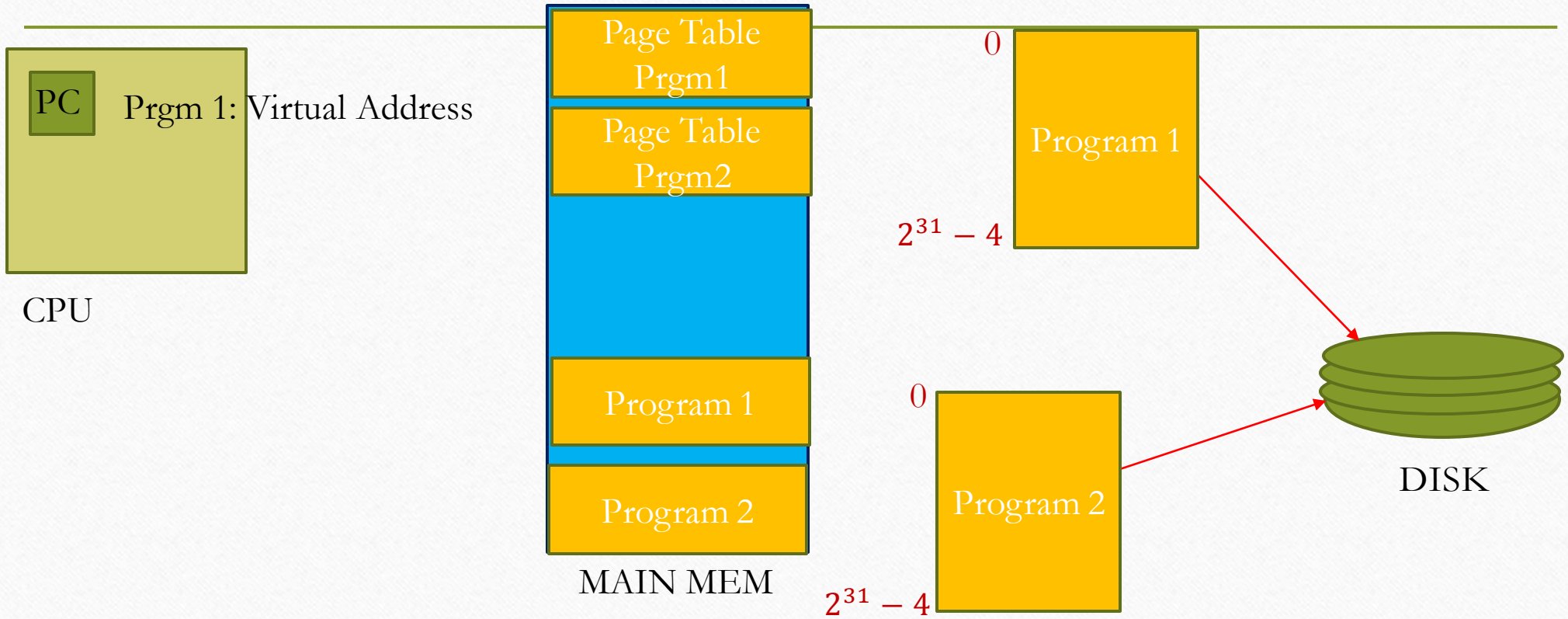
Processor works on each program, using Virtual Addresses
And translating these into Physical Addresses
Via the Page Table.

Number of Bits in Physical Address depends on the size of RAM

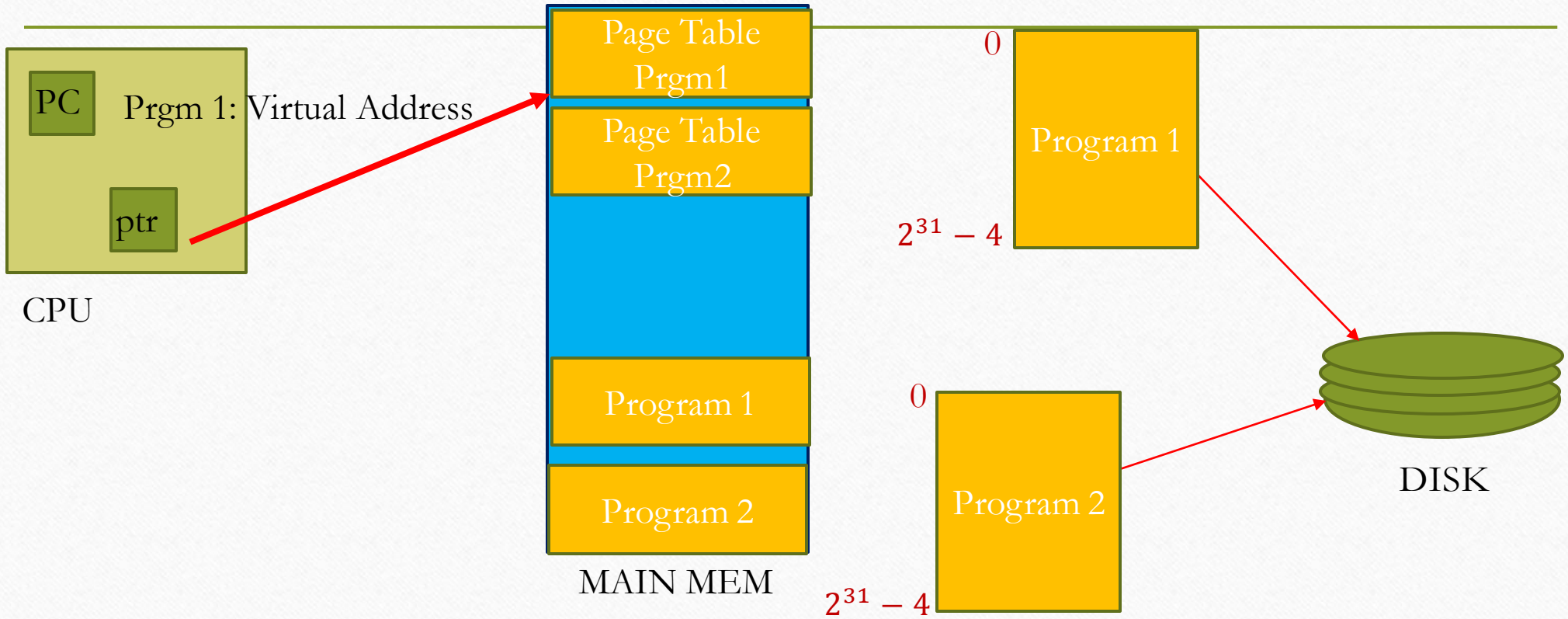
Program 1 is loaded. CPU works with Virtual Addresses.



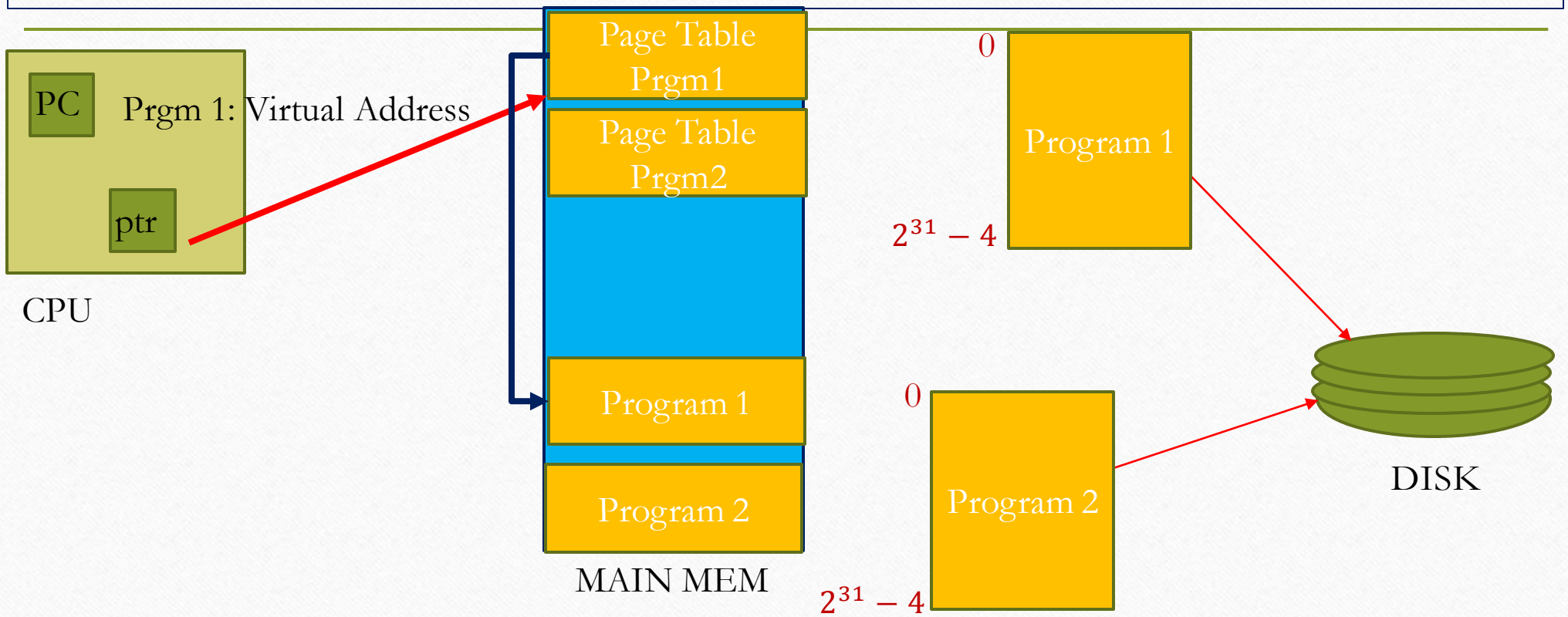
Step1: Translate This Virtual Address to a Physical Address via Page Table for Prgm1



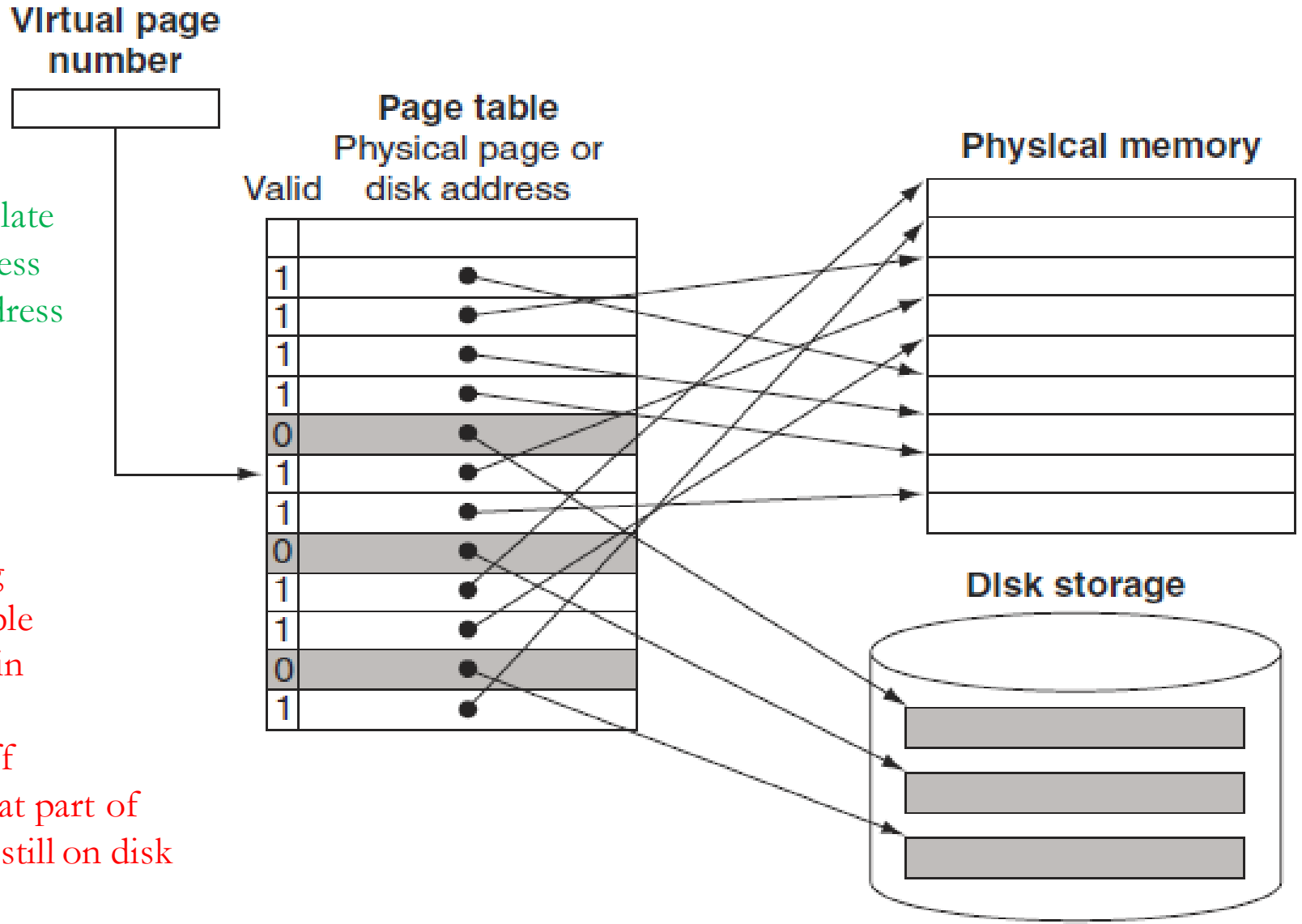
Step1: Page Table Register points to the Location of Page Table for Current Program: Prgm1



Step2: Page Table says where the Physical location in Main Memory is for that Virtual Address

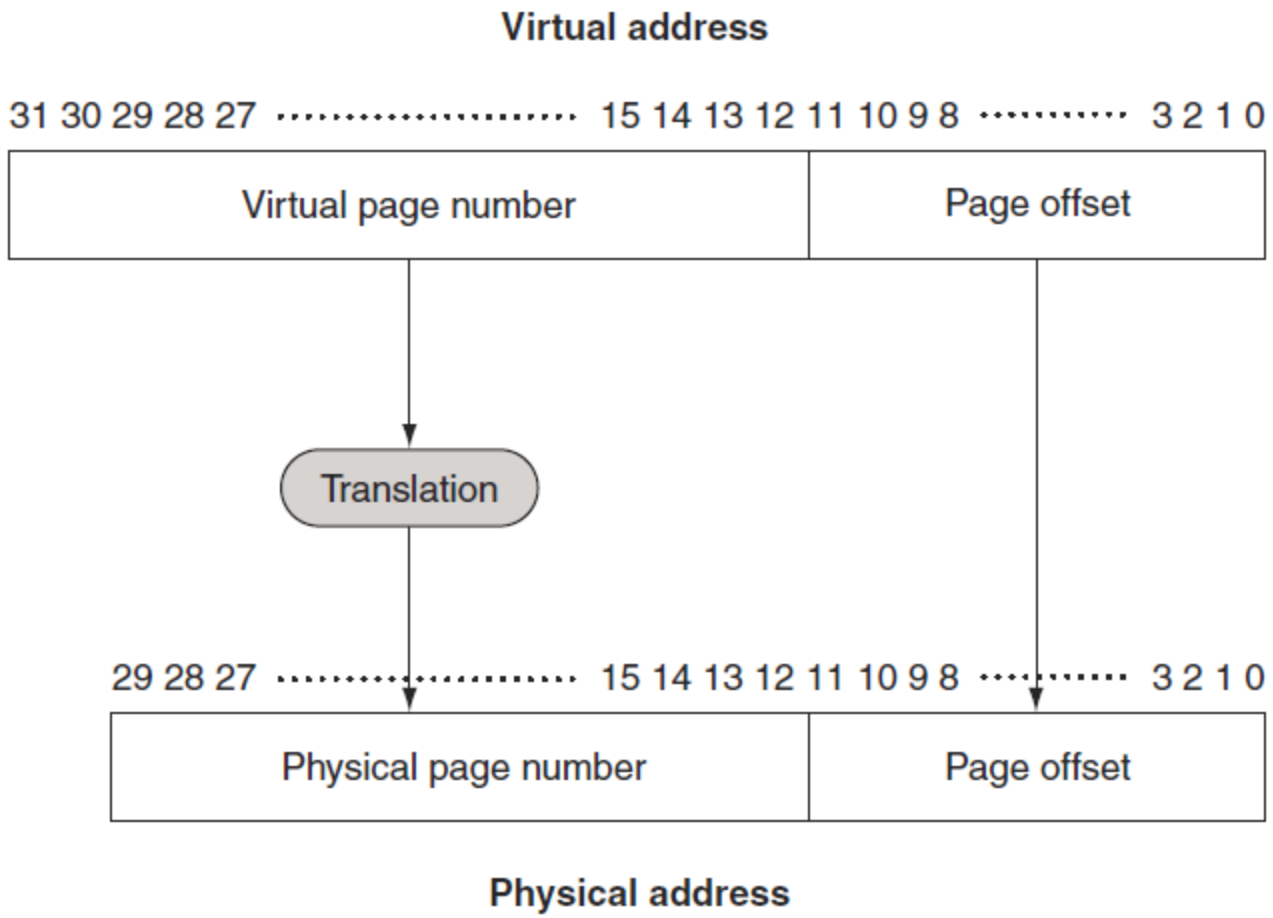


CPU
Needs the
Data at this
Virtual Address



Step1: First Translate
The Virtual Address
To a Physical Address

Page Table will
Have a mapping
For every possible
Virtual address in
Prgm1.
If Valid bit is off
This means =that part of
The program is still on disk



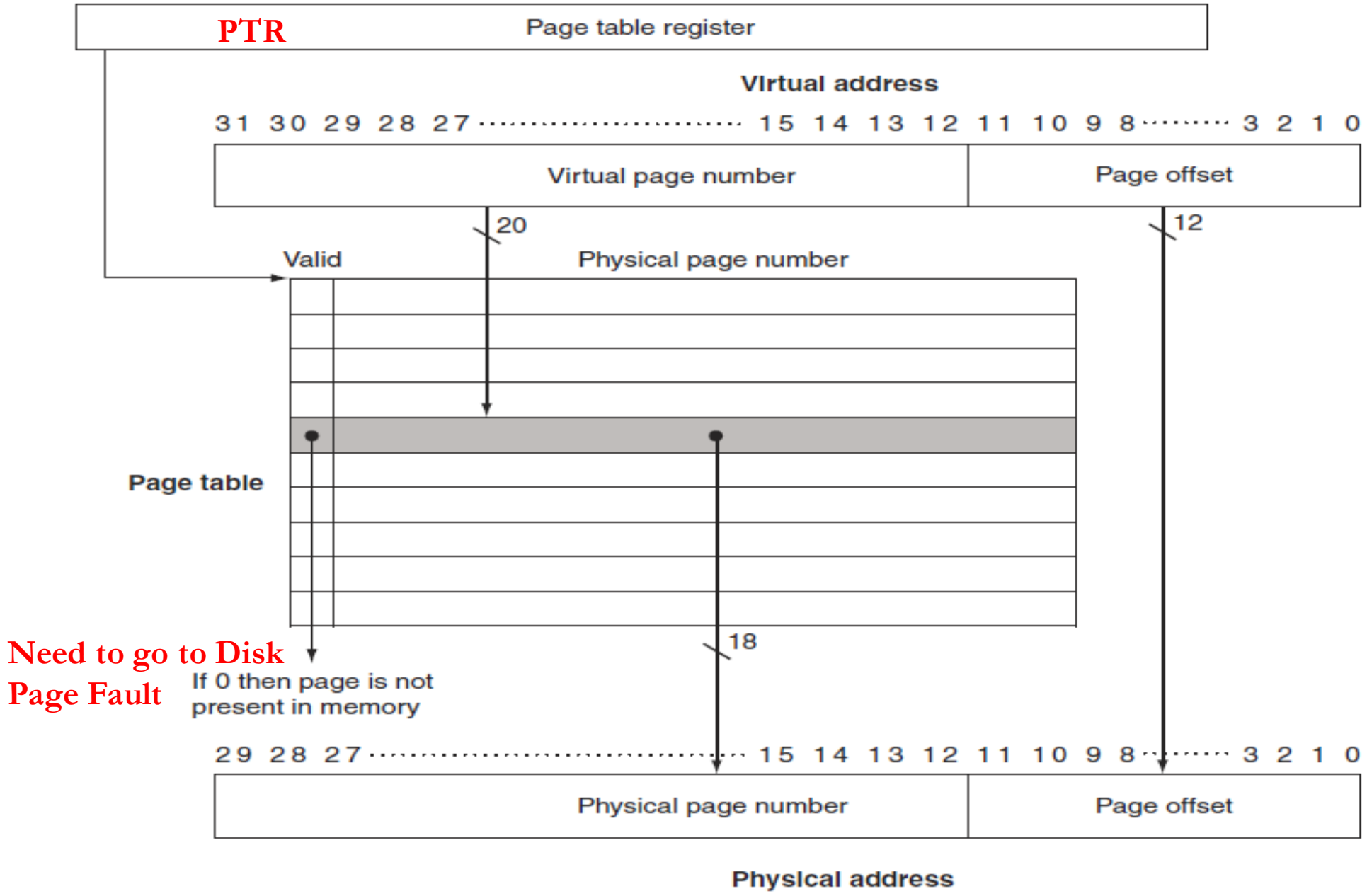
Data for Program 1 will be
Paged
1 block of data is equivalent
To a Page of Data.

Break up the 32 address bits
Into the Page Portion
And the Offset Portion.

Offset: How far down to go
In the page.

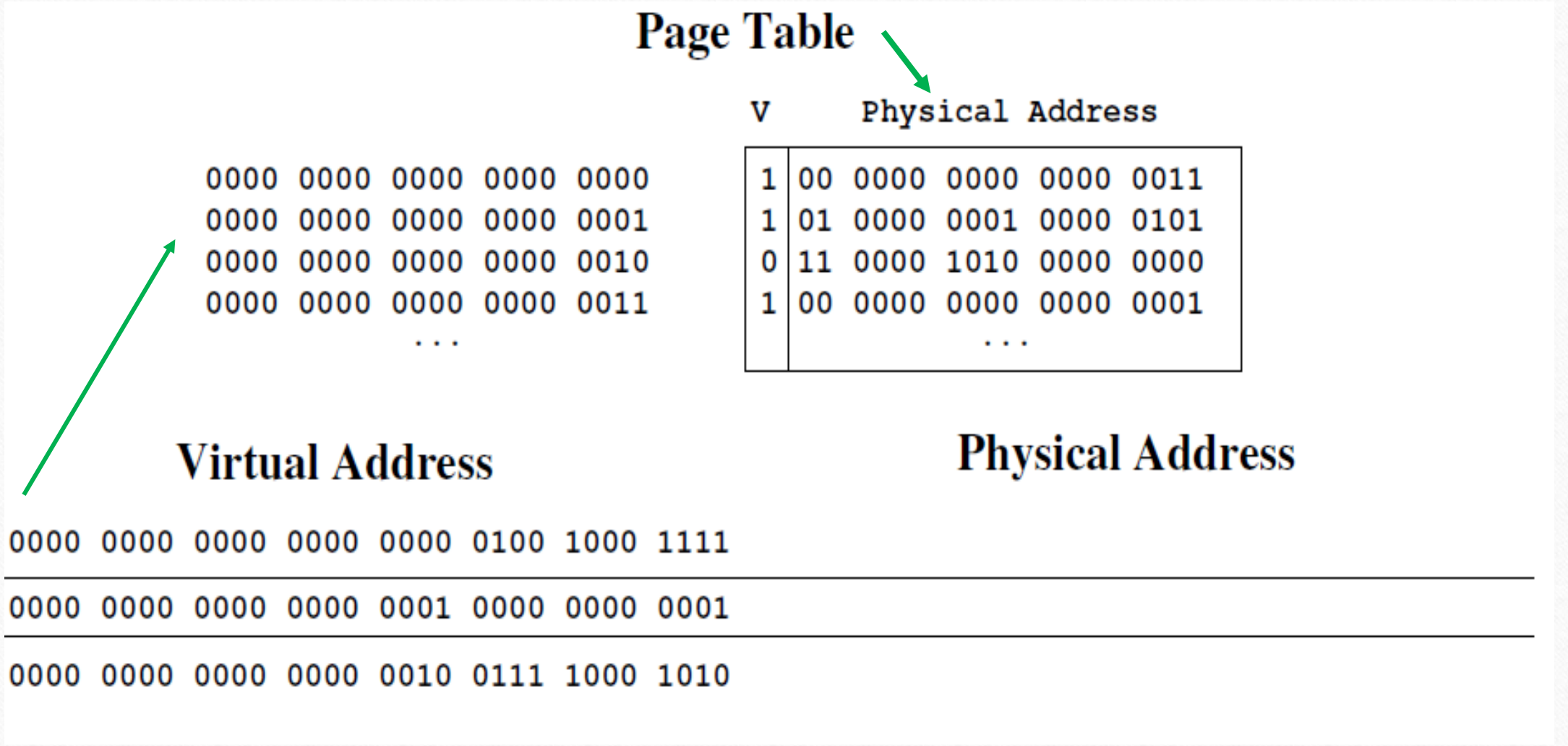
Therefore the only translation
Needed is the
Virtual to Physical Page number

Offset bits being 12 bits indicates
The Page size is 2^12
Or 4K page Size


PTR
Page table register
Virtual address
31 30 29 28 27 ..... 15 14 13 12 11 10 9 8 ..... 3 2 1 0
Virtual page number
Page offset
20
12
Valid
Physical page number
Page table
18
Need to go to Disk
Page Fault
If 0 then page is not present in memory
29 28 27 ..... 15 14 13 12 11 10 9 8 ..... 3 2 1 0
Physical page number
Page offset
Physical address

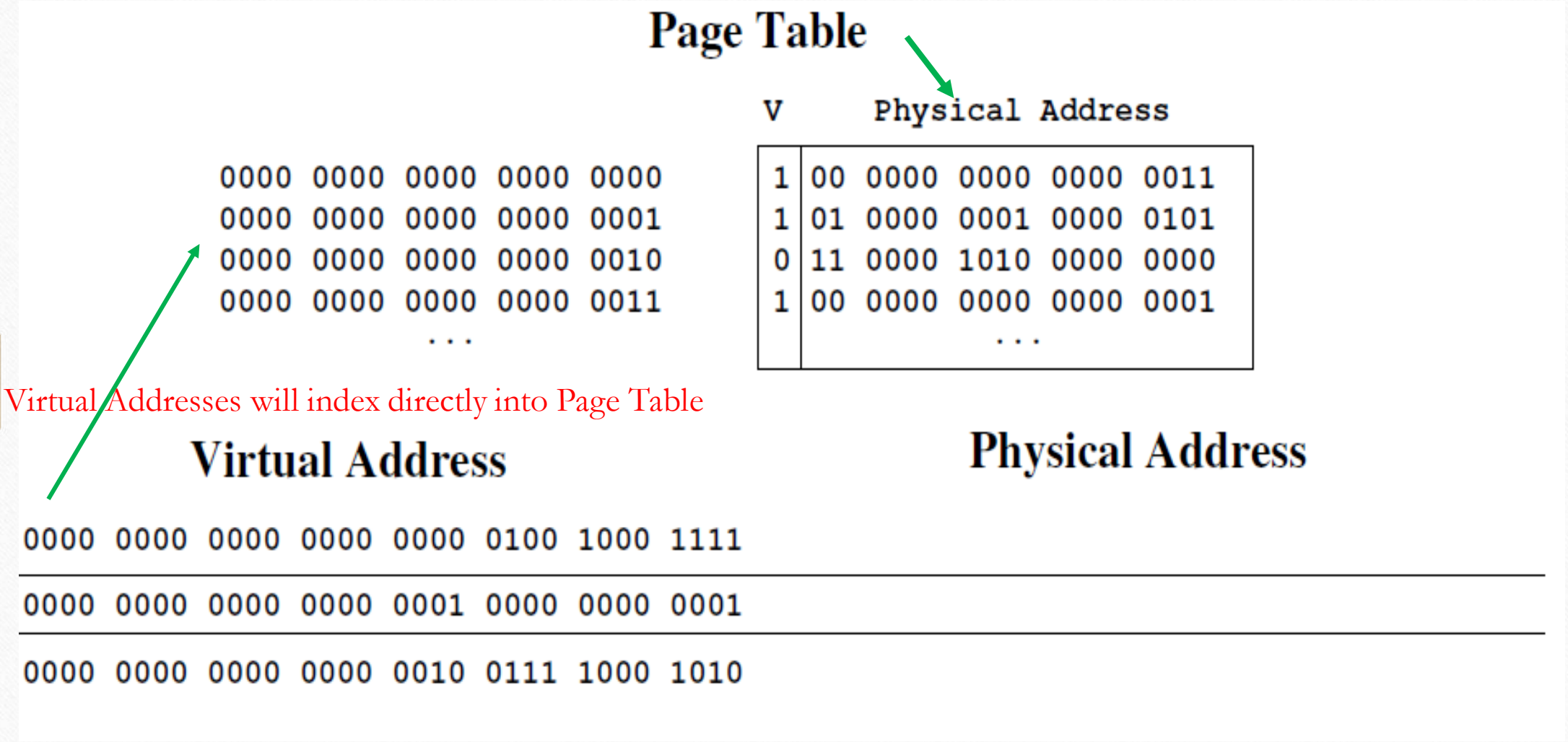
30 bits for Physical Address: Therefore 1GB Memory Space Available.

Virtual Address space : 2^32 therefore 4GB address space.



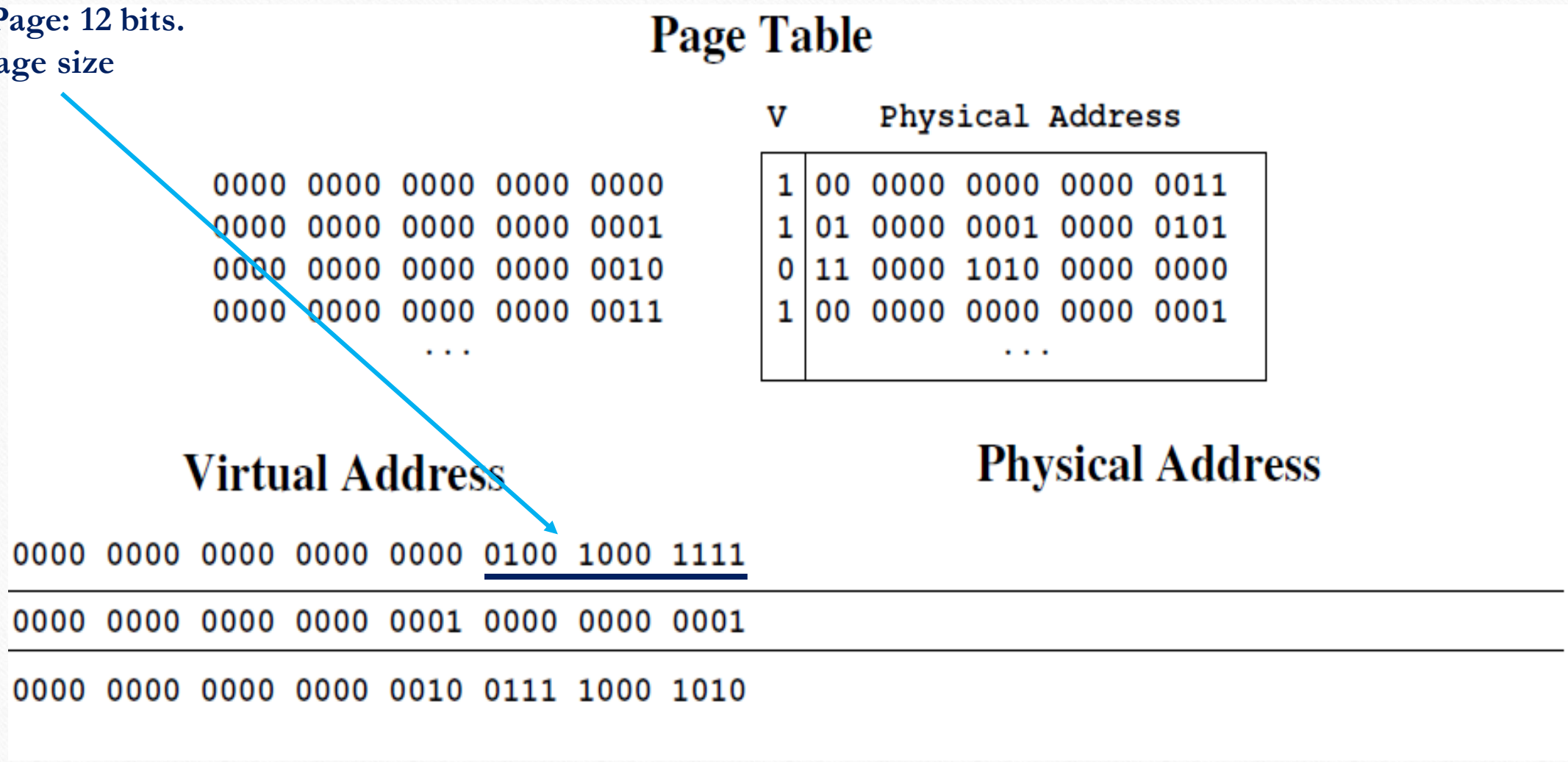
30 bits for Physical Address: Therefore 1GB Memory Space Available.

Virtual Address space : 2^32 therefore 4GB address space.



If your program uses all of its address space, then it will be paged in from Disk,
Not ALL of it will ever reside in Memory

Each Page: 12 bits.
4KB page size



Page Table

	V	Physical Address
0000 0000 0000 0000 0000	1	00 0000 0000 0000 0011
0000 0000 0000 0000 0001	1	01 0000 0001 0000 0101
0000 0000 0000 0000 0010	0	11 0000 1010 0000 0000
0000 0000 0000 0000 0011	1	00 0000 0000 0000 0001
...		...

Virtual Address

0000 0000 0000 0000 0000 0100 1000 1111

0000 0000 0000 0000 0001 0000 0000 0001

0000 0000 0000 0000 0010 0111 1000 1010

Physical Address

00 0000 0000 0000 0011 0100 1000 1111

Page Table

	V	Physical Address
0000 0000 0000 0000 0000	1	00 0000 0000 0000 0011
0000 0000 0000 0000 0001	1	01 0000 0001 0000 0101
0000 0000 0000 0000 0010	0	11 0000 1010 0000 0000
0000 0000 0000 0000 0011	1	00 0000 0000 0000 0001
...		...

Virtual Address

0000 0000 0000 0000 0000 0100 1000 1111

0000 0000 0000 0000 0001 0000 0000 0001

0000 0000 0000 0000 0010 0111 1000 1010

Physical Address

00 0000 0000 0000 0011 0100 1000 1111

Offset bits
Remain the same

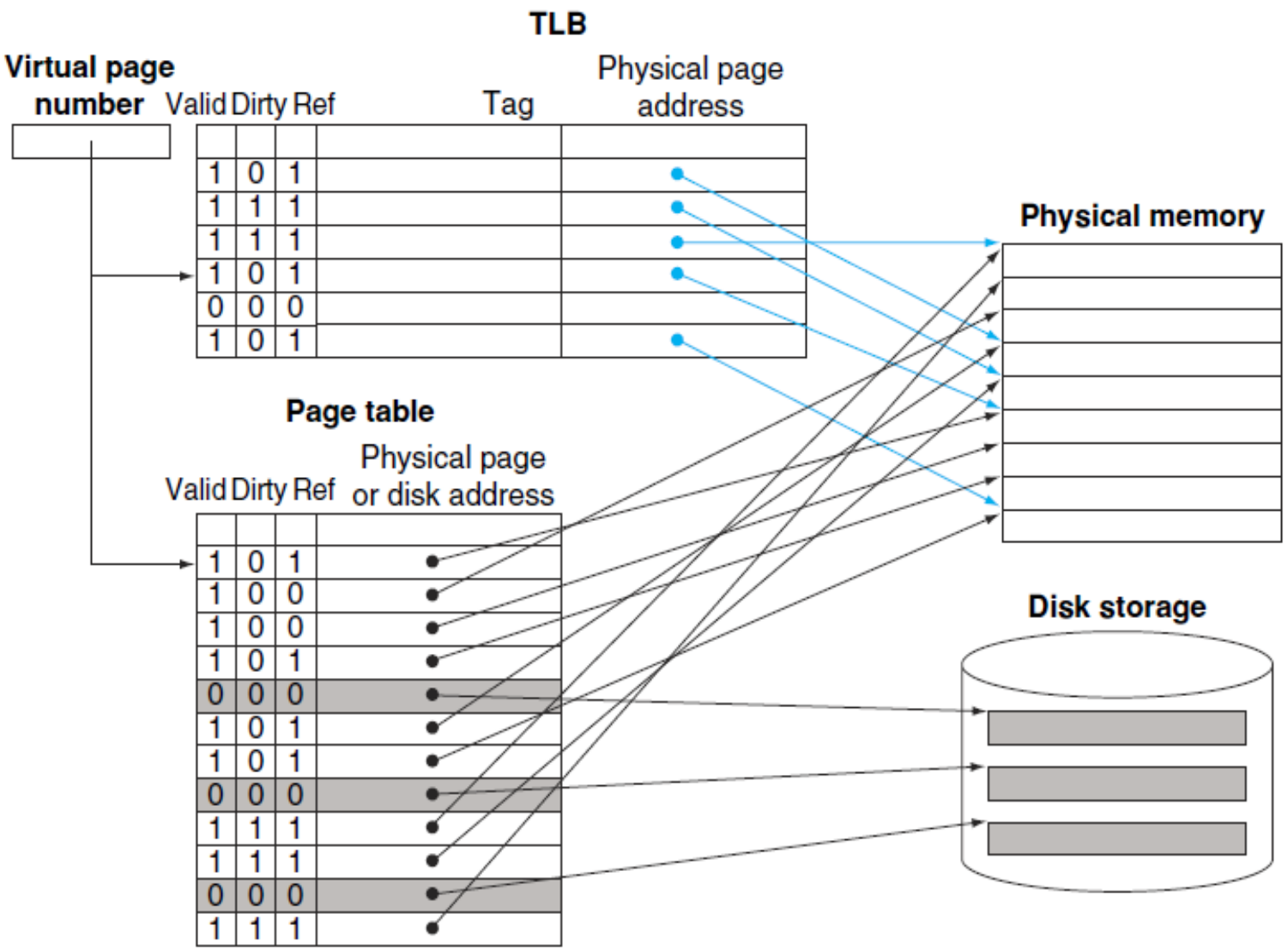
Page Table

	V	Physical Address
0000 0000 0000 0000 0000	1	00 0000 0000 0000 0011
0000 0000 0000 0000 0001	1	01 0000 0001 0000 0101
0000 0000 0000 0000 0010	0	11 0000 1010 0000 0000
0000 0000 0000 0000 0011	1	00 0000 0000 0000 0001
...		...

Virtual Address	Physical Address
0000 0000 0000 0000 0000 0100 1000 1111	00 0000 0000 0000 0011 0100 1000 1111
0000 0000 0000 0000 0001 0000 0000 0001	01 0000 0001 0000 0101 0000 0000 0001
0000 0000 0000 0000 0010 0111 1000 1010	**Not Valid : GO TO DISK**

Translation Look Aside Buffer: TLB

- In order to translate between virtual and physical addresses,
- Takes time to access Page Table in RAM
- Instead use a small (cache) for address translations only: TLB
- CPU gets a Virtual Address:
  - First Check to Translate this using the TLB
  - If Not in TLB- Check Page Table in Main Memory

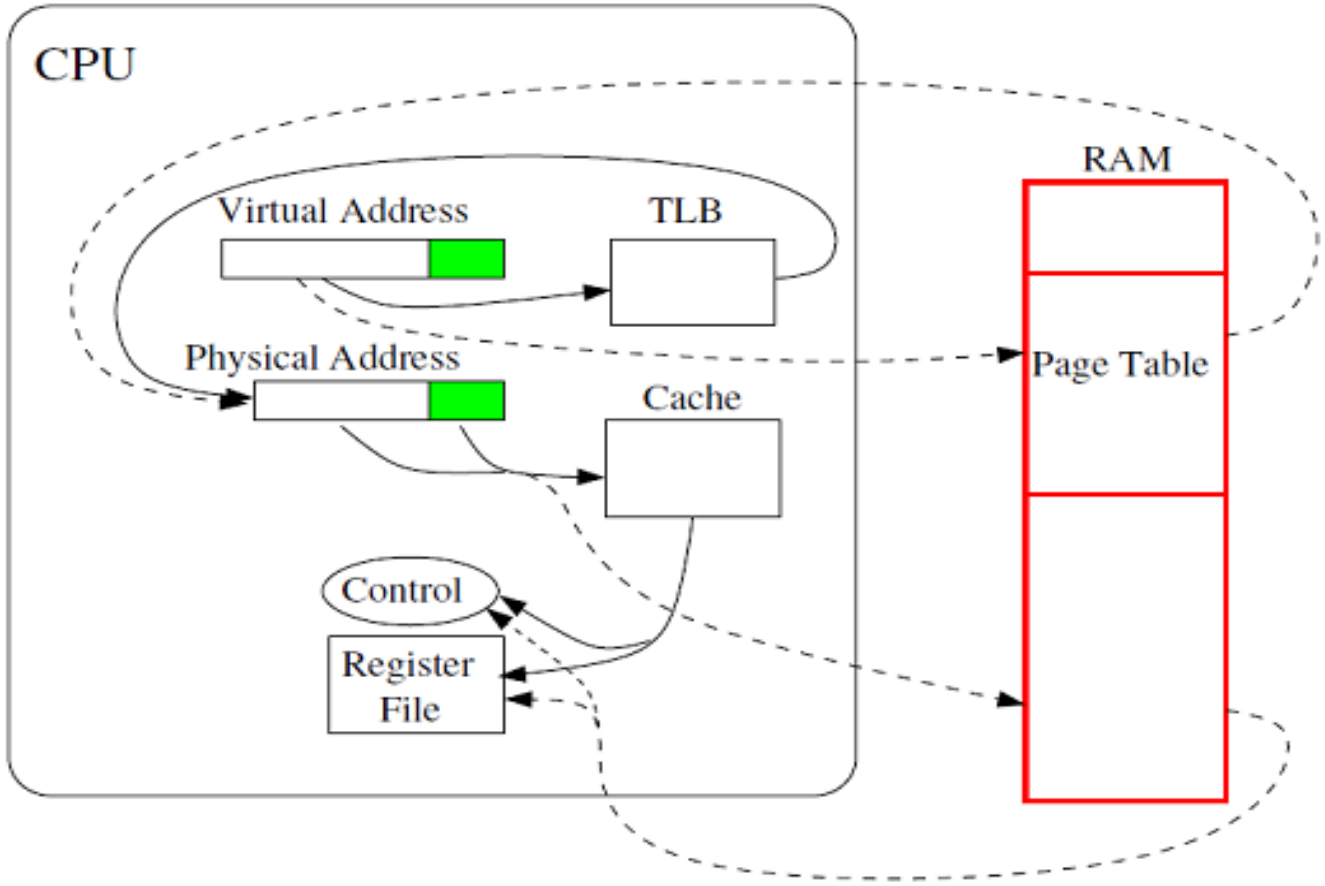


TLB: IDEA to CACHE
Frequent Page Table Entries.

This is because looking up page table
In Memory (even with PTR)
Requires one memory access.

TLB: Will store
Page Table Information relevant to
Both **instructions** and **data**

Virtual Memory: The CPU



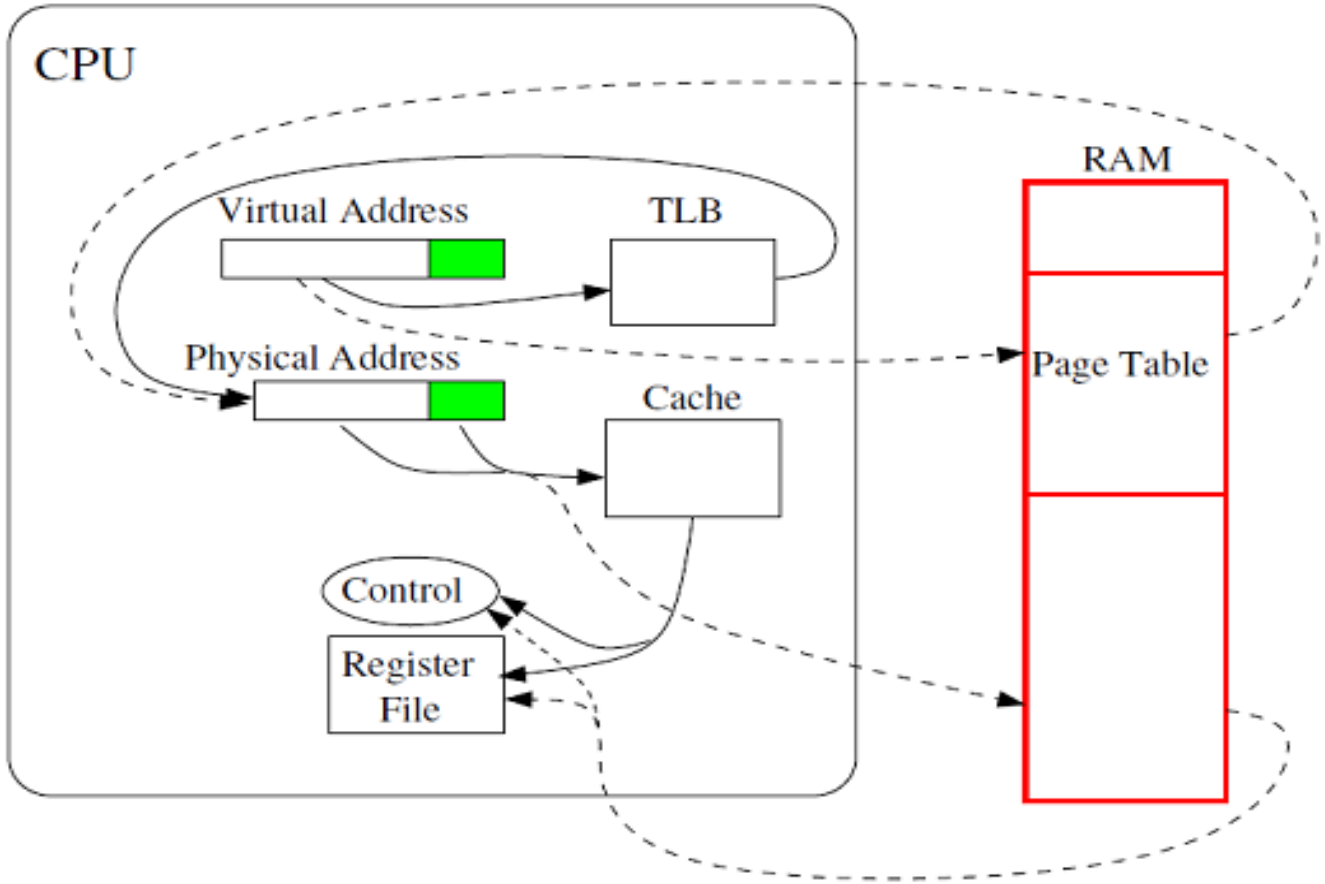
The Processor Is working on each Instruction, one at at time
Based on the virtual address space of Each program. Processor is also accessing Data based on the virtual addresses Within each program

Page Table: Virtual to Physical Translation
Entire Page Table in RAM, however Parts of this page table in TLB.

TLB is close (like a cache)
Look here first.

Virtual Address is translated to Physical Address via TLB (if TLB miss then we Need to do translation from RAM)

Virtual Memory: The CPU



ONCE we have the PHYSICAL Address:

First Check Cache if it is there

Check Cache in the same way:

Use Index Bits to tell us
Where in the cache we can look for
This address: Use the Physical address
Bits as we did with
Set- Associative Caches.
Blocks of one word, 4 words etc.

All of this applies to caches.