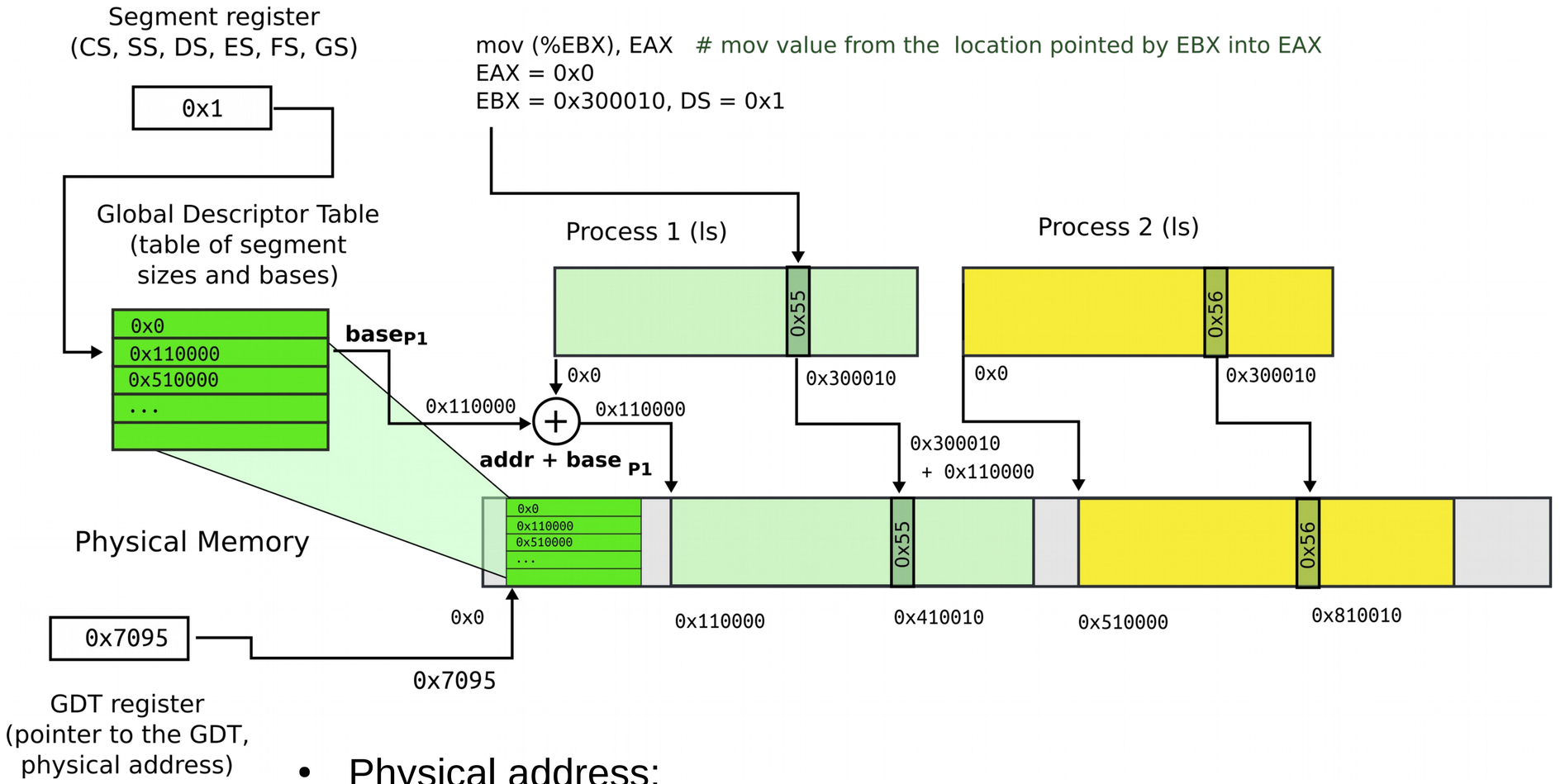
CS 238P
Operating Systems
Discussion 5

Based on slides from Saehanseul Yi

Today’s agenda

- Intro to homework 4
- Solving segmentation problems
- Solving paging problems

Segmentation: base + offset



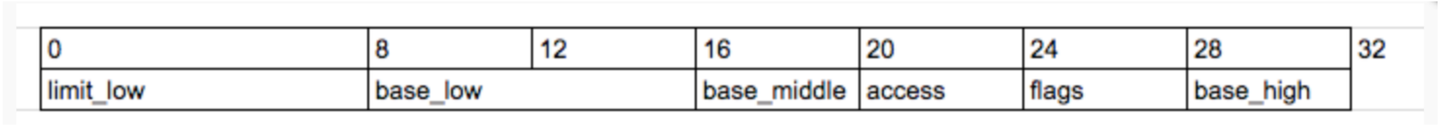
- Physical address:
  - 0x410010 = 0x300010 (offset) + 0x110000 (base)
  - Intel calls this address “linear”

From Anton Burtsev lecture 6: Address translation

GDT

GDT is a table, each entry has the following format:

```
struct gdt_entry_struct
{
   u16int limit_low;           // The lower 16 bits of the limit.
   u16int base_low;            // The lower 16 bits of the base.
   u8int  base_middle;         // The next 8 bits of the base.
   u8int  access;              // Access flags, determine what ring this segment can be used in.
   u8int  granularity;
   u8int  base_high;           // The last 8 bits of the base.
} __attribute__((packed));
```



0	8	12	16	20	24	28	32
limit_low	base_low		base_middle	access	flags	base_high	

GDTR

Register which stores address of GDT

48 bits size

|LIMIT(16)|— —BASE(32)— —|

Solving segmentation problem

encoded(base, limit) return a valid GDT entry

GDT:

0x0 -> encoded(0x100, 0x200)

0x1 -> encoded(0x300, 0x200)

0x2 -> encoded(0x200, 0x100)

0x3 -> encoded(0x200, 0x100)

0x4 -> encoded(0x800, 0x100)

Question 1: which address we would get is we want to access cs:eax when CS = 0x1, eax = 0x13?

Solving segmentation problem

encoded(base, limit) return a valid GDT entry

GDT:

0x0 -> encoded(0x100, 0x200)

0x1 -> encoded(0x300, 0x200)

0x2 -> encoded(0x200, 0x100)

0x3 -> encoded(0x200, 0x100)

0x4 -> encoded(0x800, 0x100)

Question 1: which address we would get is we want to access cs:eax when CS = 0x1, eax = 0x13?

Answer: 0x300

Solving segmentation problem

encoded(base, limit) return a valid GDT entry

GDT:

0x0 -> encoded(0x100, 0x200)

0x1 -> encoded(0x300, 0x200)

0x2 -> encoded(0x200, 0x100)

0x3 -> encoded(0x200, 0x100)

0x4 -> encoded(0x800, 0x100)

Question2: which address we would get is we want to access ds:eax when DS = 0x3, eax = 0x513?

Solving segmentation problem

encoded(base, limit) return a valid GDT entry

GDT:

0x0 -> encoded(0x100, 0x200)

0x1 -> encoded(0x300, 0x200)

0x2 -> encoded(0x200, 0x100)

0x3 -> encoded(0x200, 0x100)

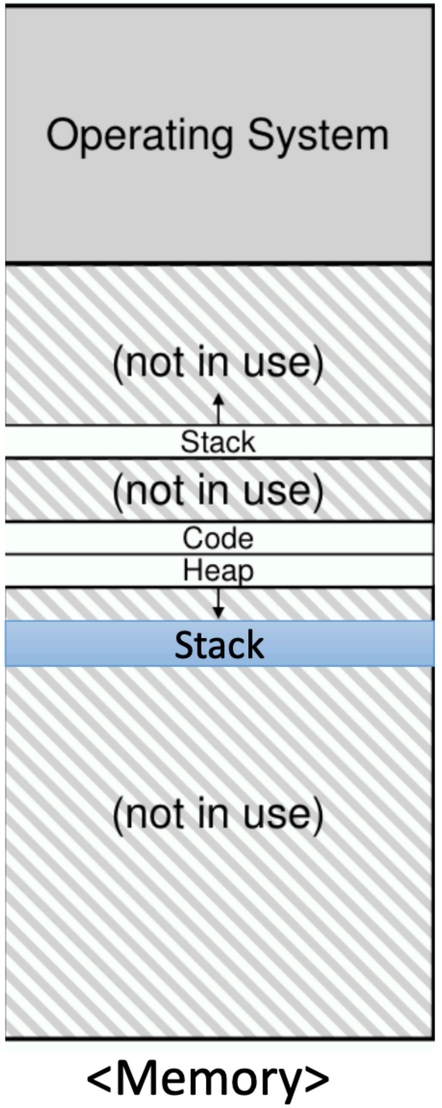
0x4 -> encoded(0x800, 0x100)

Question2: which address we would get is we want to access ds:eax when DS = 0x3, eax = 0x513?

Answer: Fault, because limit is set to 0x100

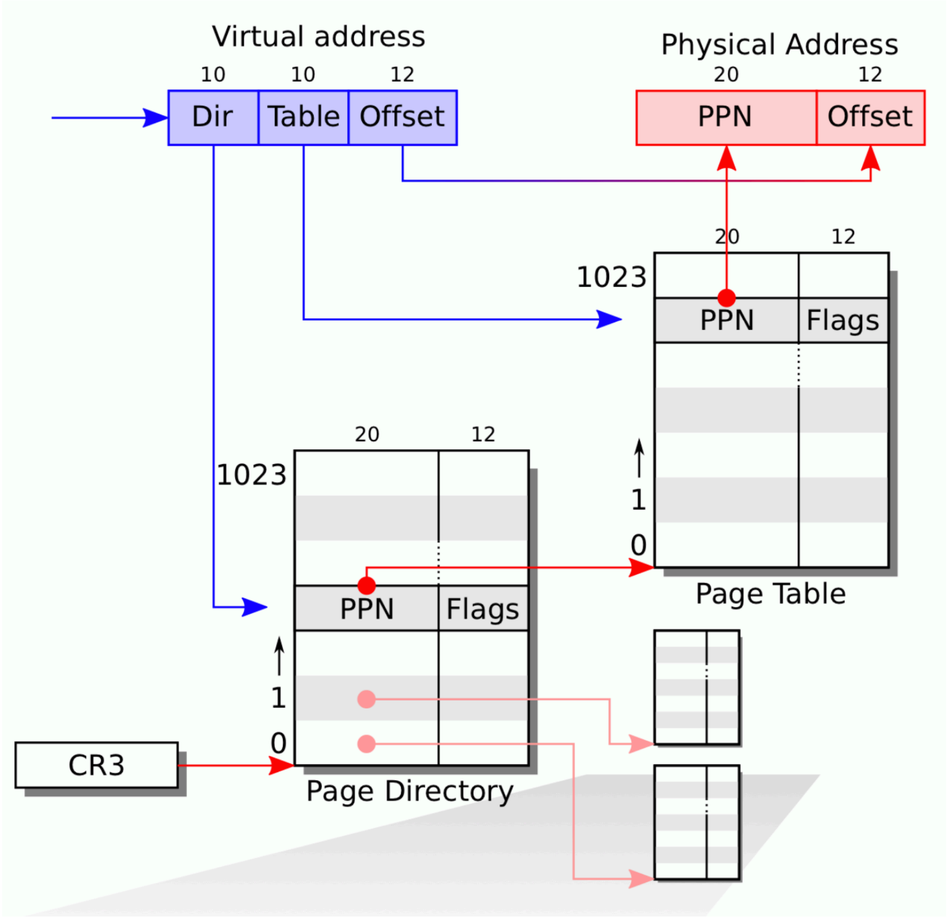
Paging

- What if segments are about to be overlapped?
  - OS should find a free contiguous memory region and move the segment
  - Fragmentation occurs
  - Sometimes the small space between segments is not large enough for a new segment -> wasted
  - Moving the segment costs a lot (lots of memory operations)
- Solution?
  - Divide memory into many fixed-size regions (pages) and allocate this to segment dynamically by paging



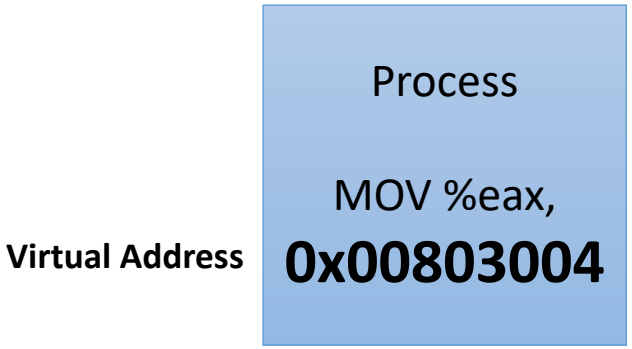
Paging

- x86 page table = an array of 2^20 Page Table Entries (PTEs)
- PTE: 20-bint Physical Page Number (PPN). Usually an address
- Top 20 bits of virtual address = index of page table
- Page Directory: contains reference to page table
- Page Fault: PTE_P(PAGE_PRESENT) is not set

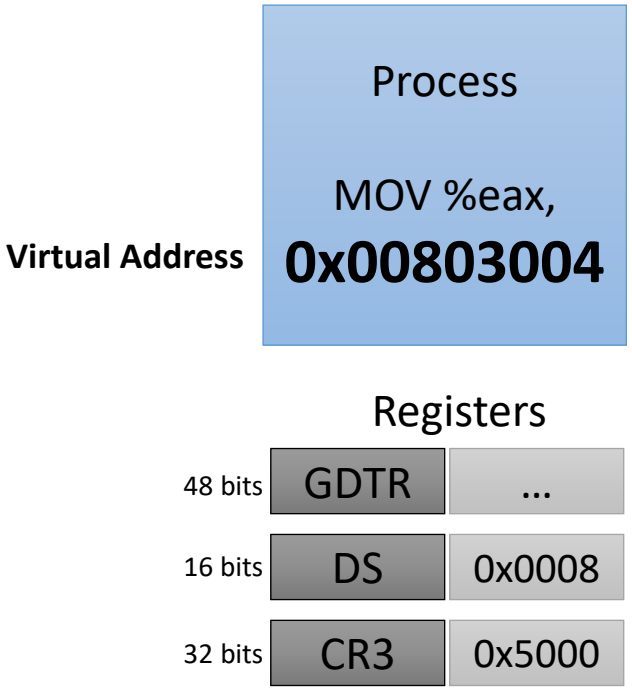


A Simple Example

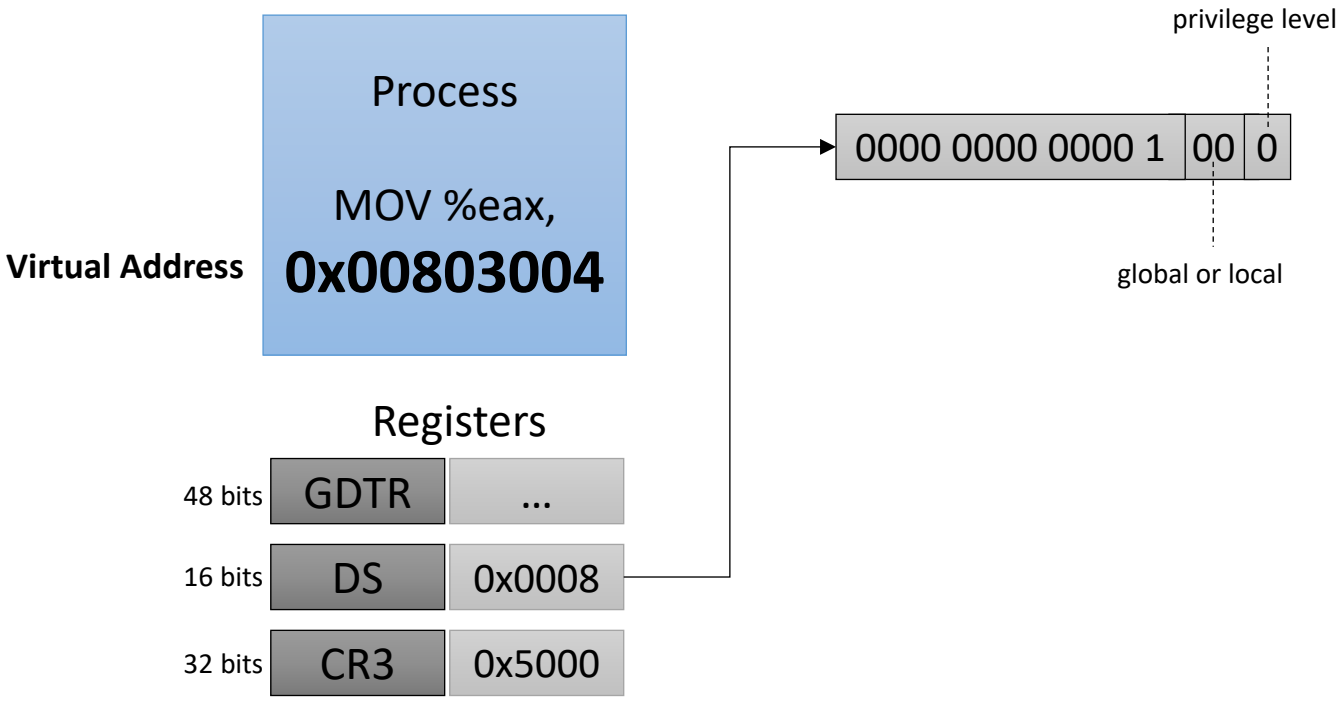
- x86, 4k page
- Logical address 0x803004 → Physical address 0x8004
- Physical address of Page Directory: 0x5000
- Physical address of the page table involved: 0x8000
- entry[1] in Global Descriptor Table: 0x1000000, 2GB
- DS register: 0x8
- **Draw the diagram of process translation**



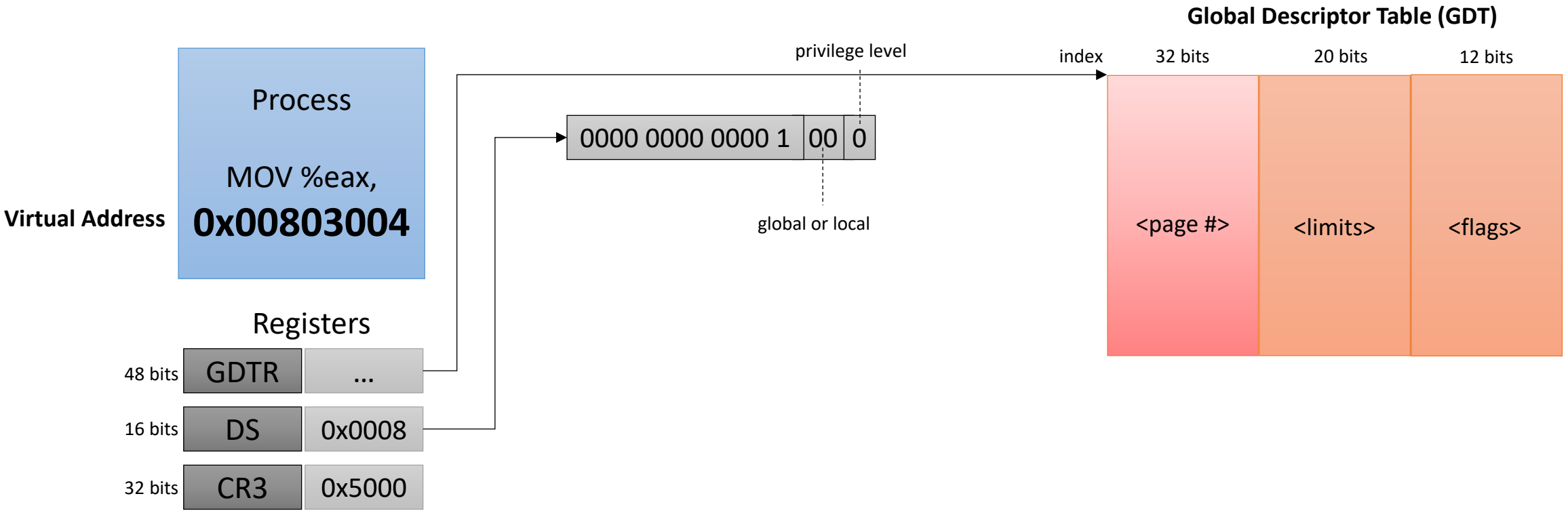
saehansy@uci.edu



saehansy@uci.edu



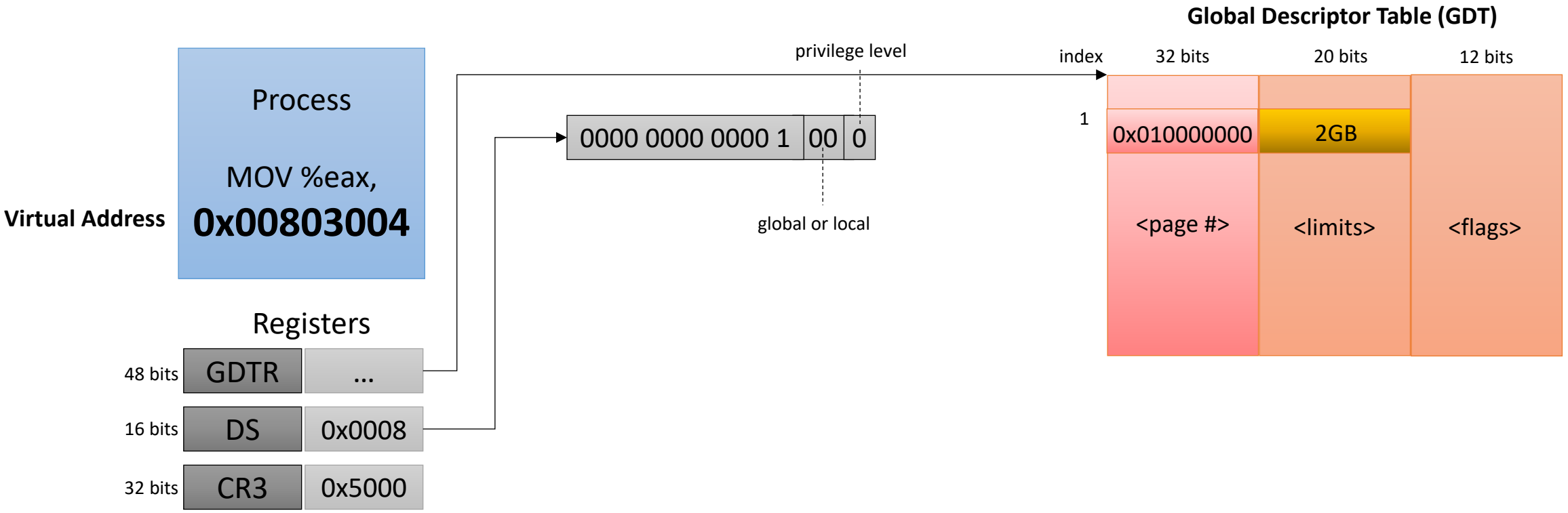
saehansy@uci.edu



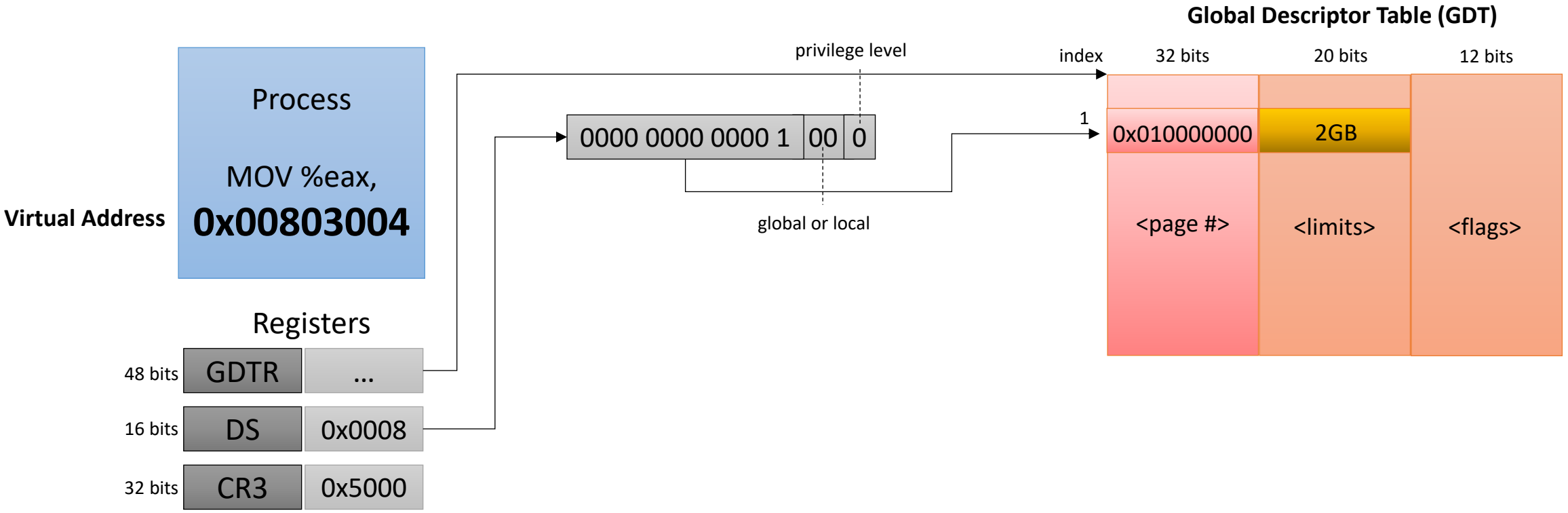
saehansy@uci.edu

A Simple Example

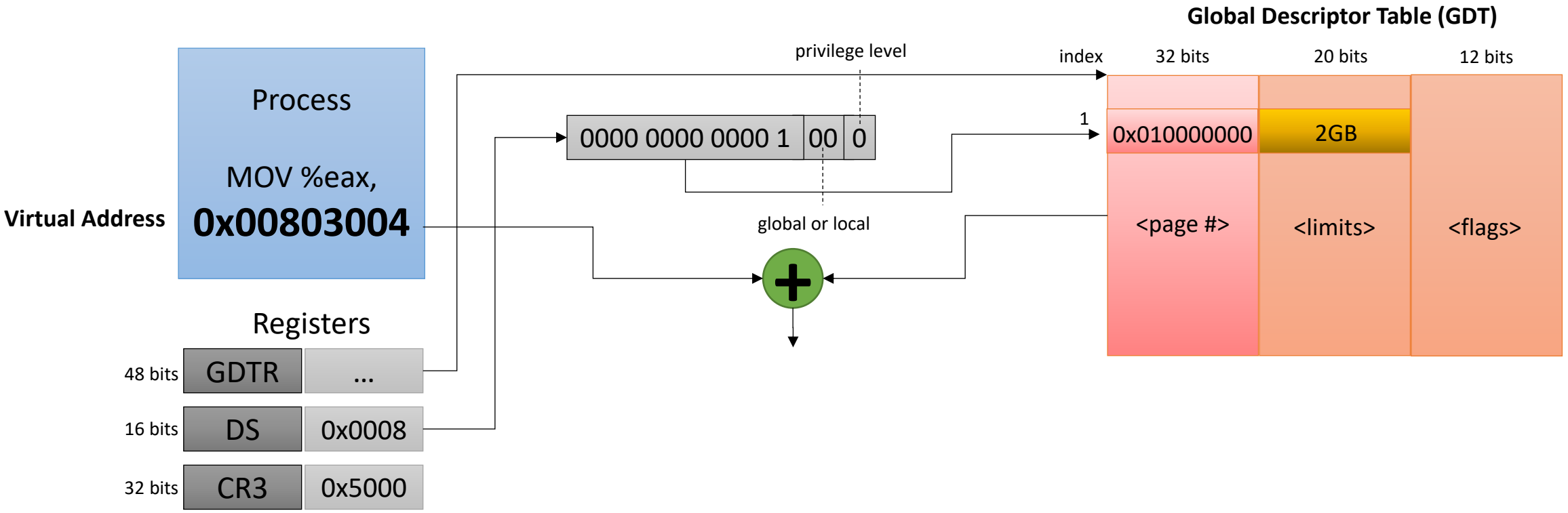
- x86, 4k page
- Logical address 0x803004 → Physical address 0x8004
- Physical address of Page Directory: 0x5000
- Physical address of the page table involved: 0x8000
- entry[1] in Global Descriptor Table: 0x1000000, 2GB
- DS register: 0x8
- **Draw the diagram of process translation**



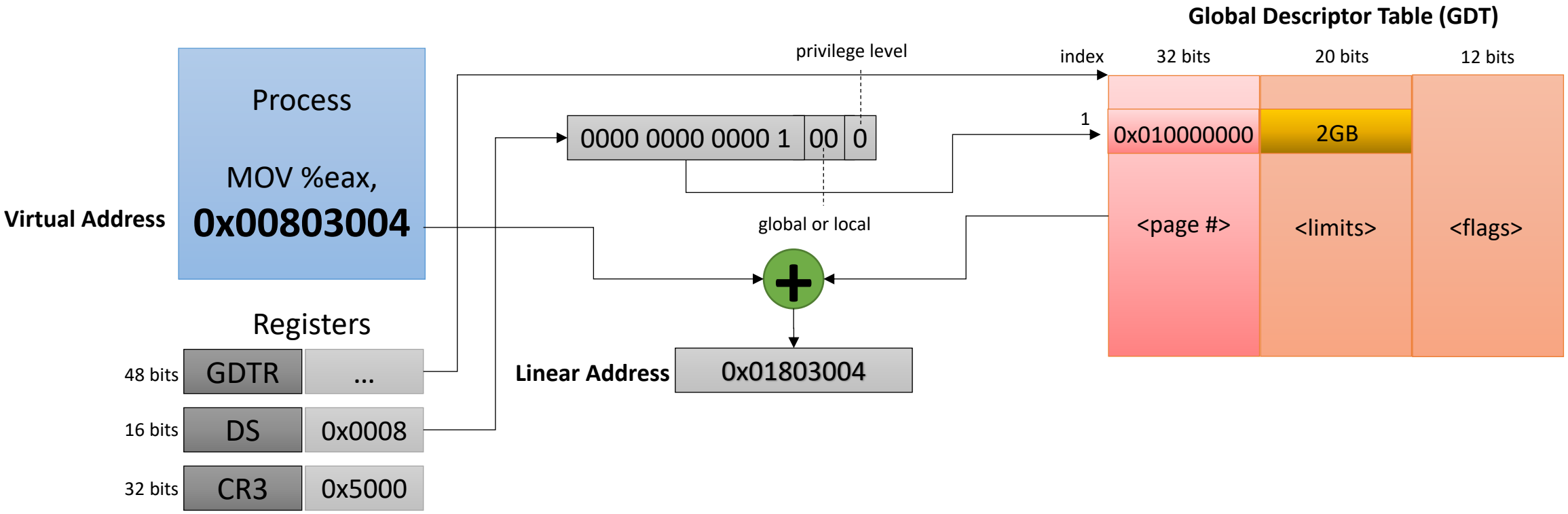
saehansy@uci.edu



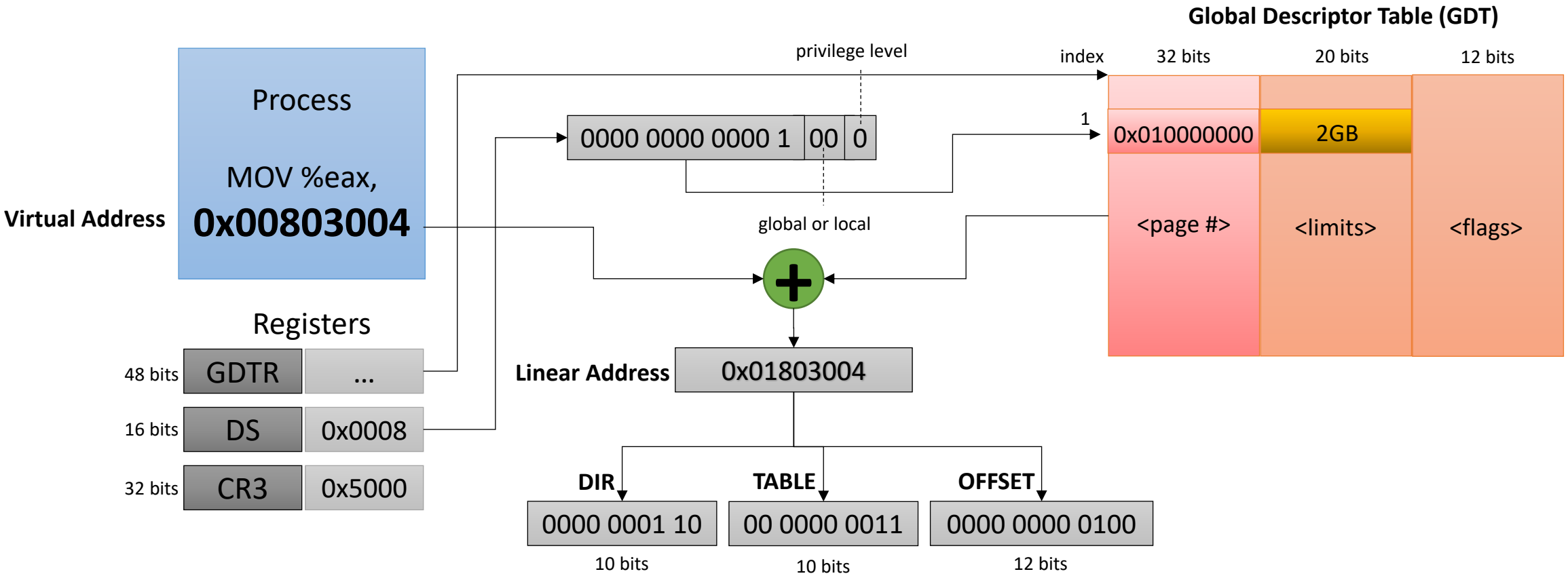
saehansy@uci.edu



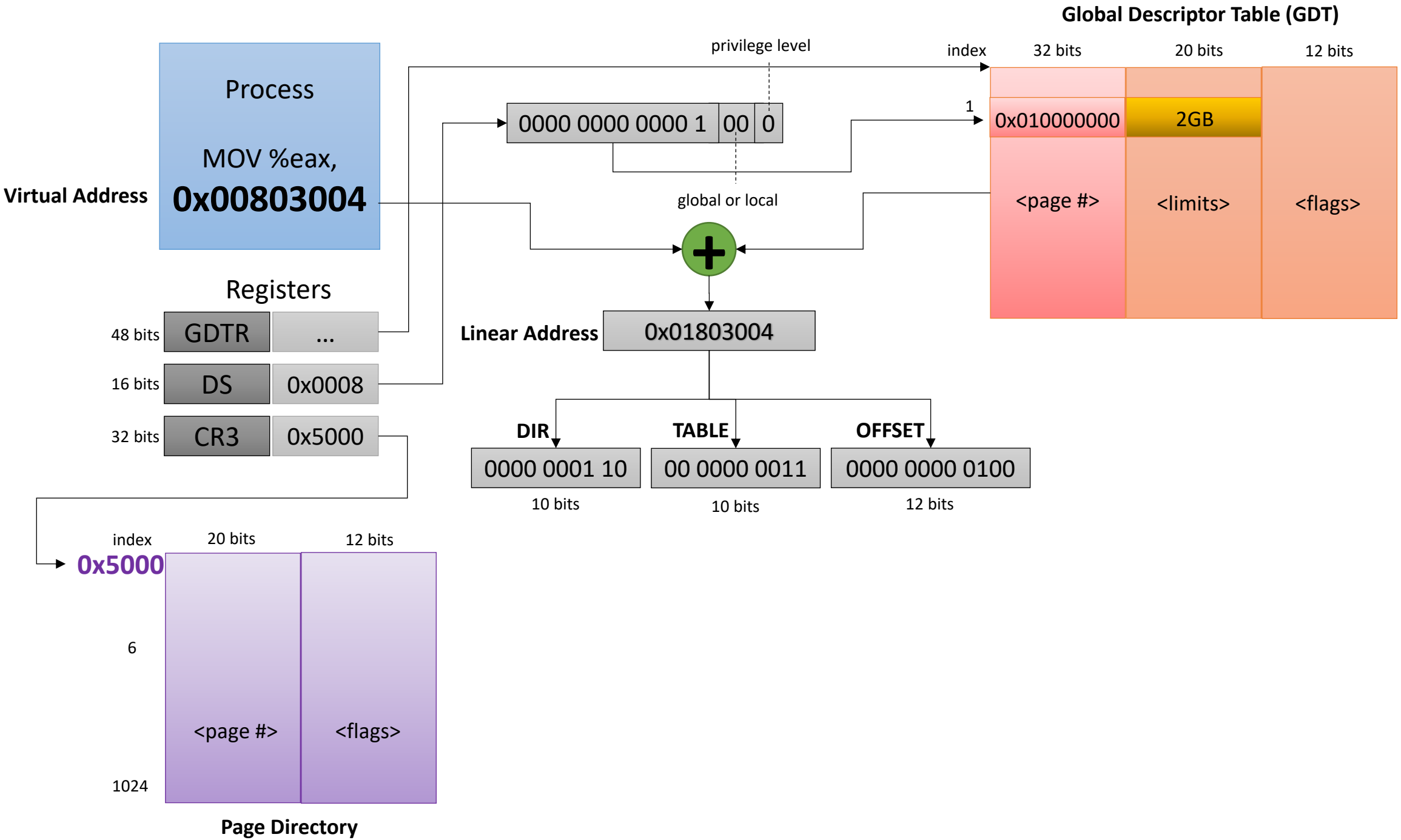
saehansy@uci.edu



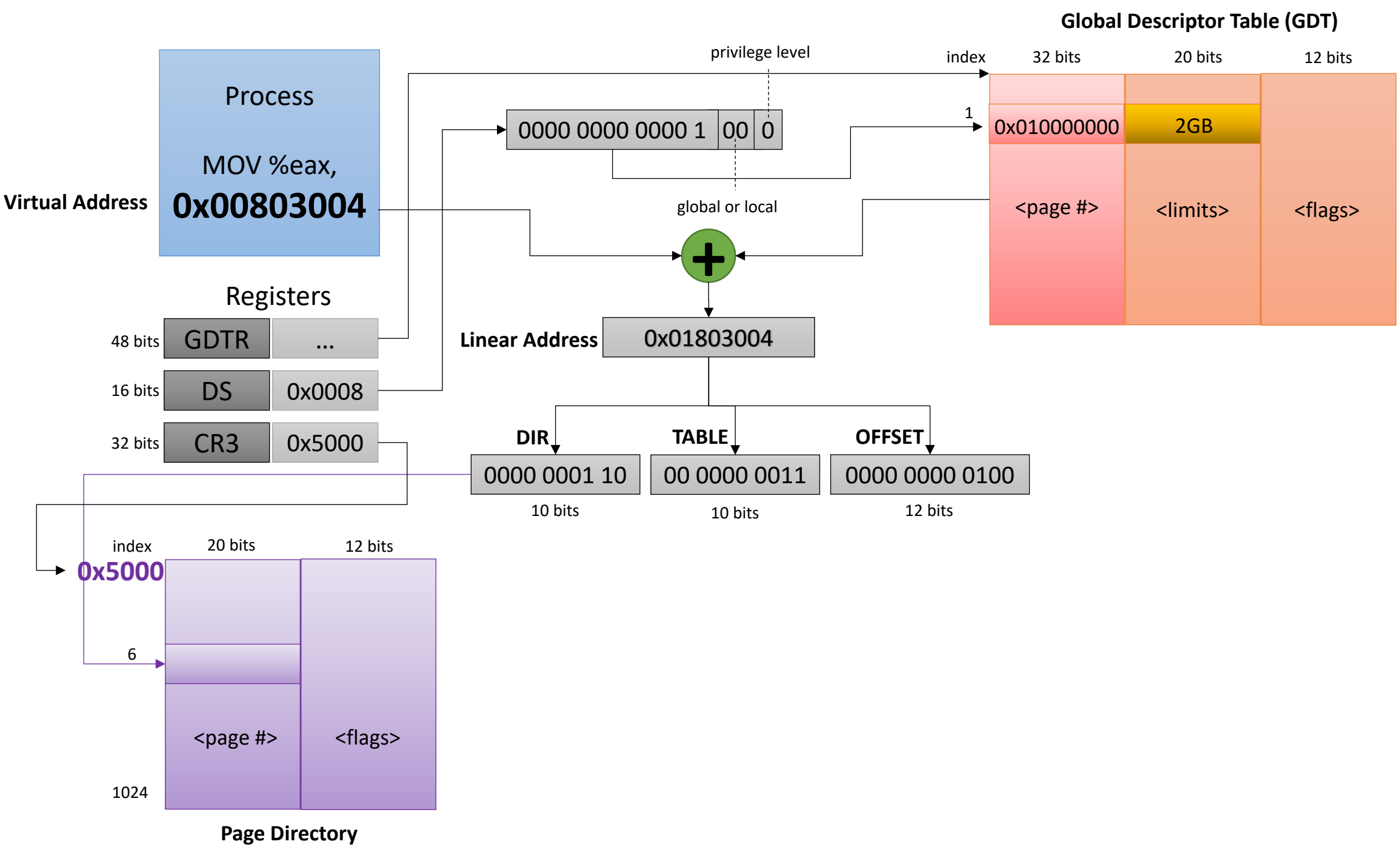
saehansy@uci.edu



saehansy@uci.edu



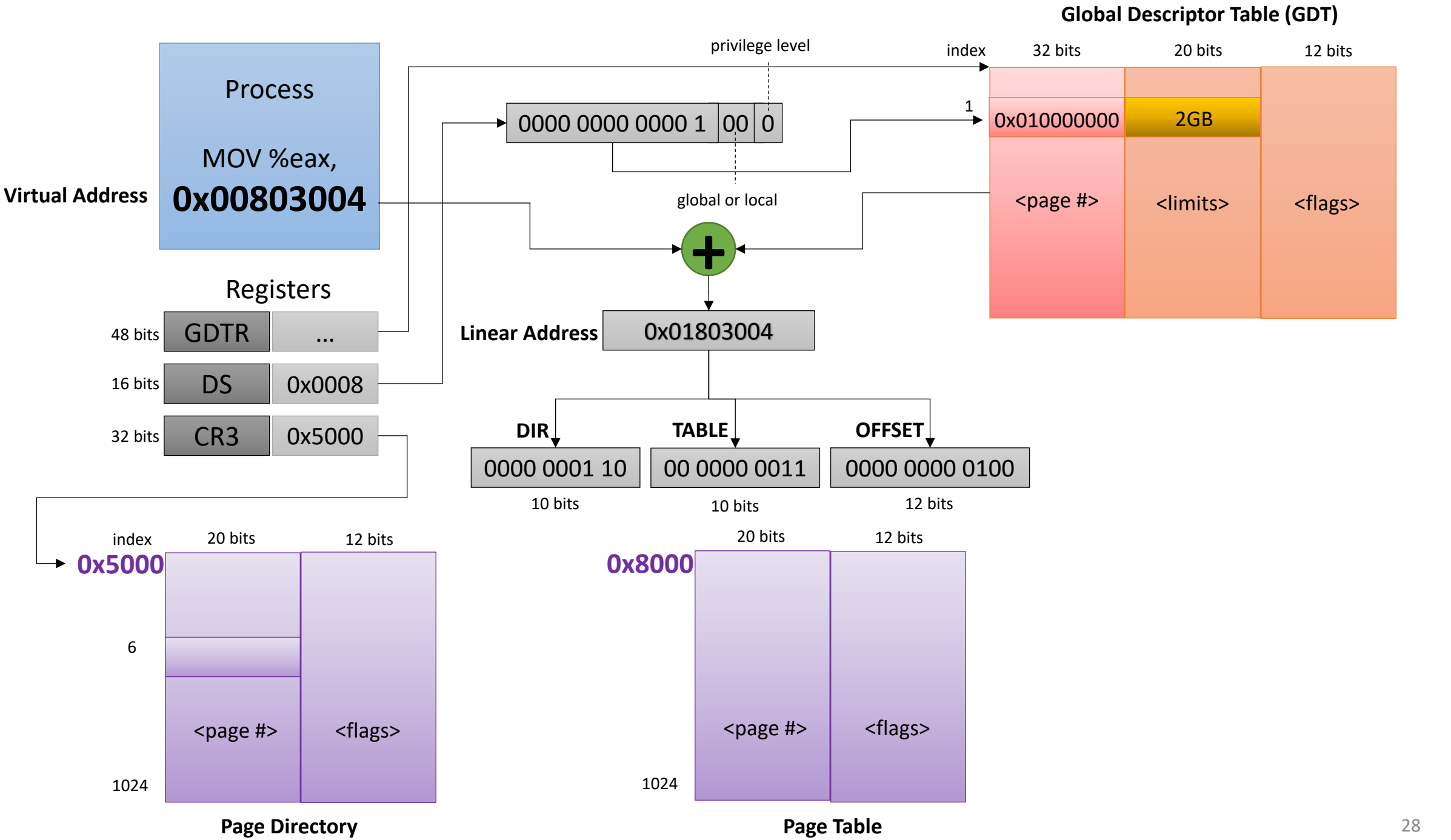
saehansy@uci.edu



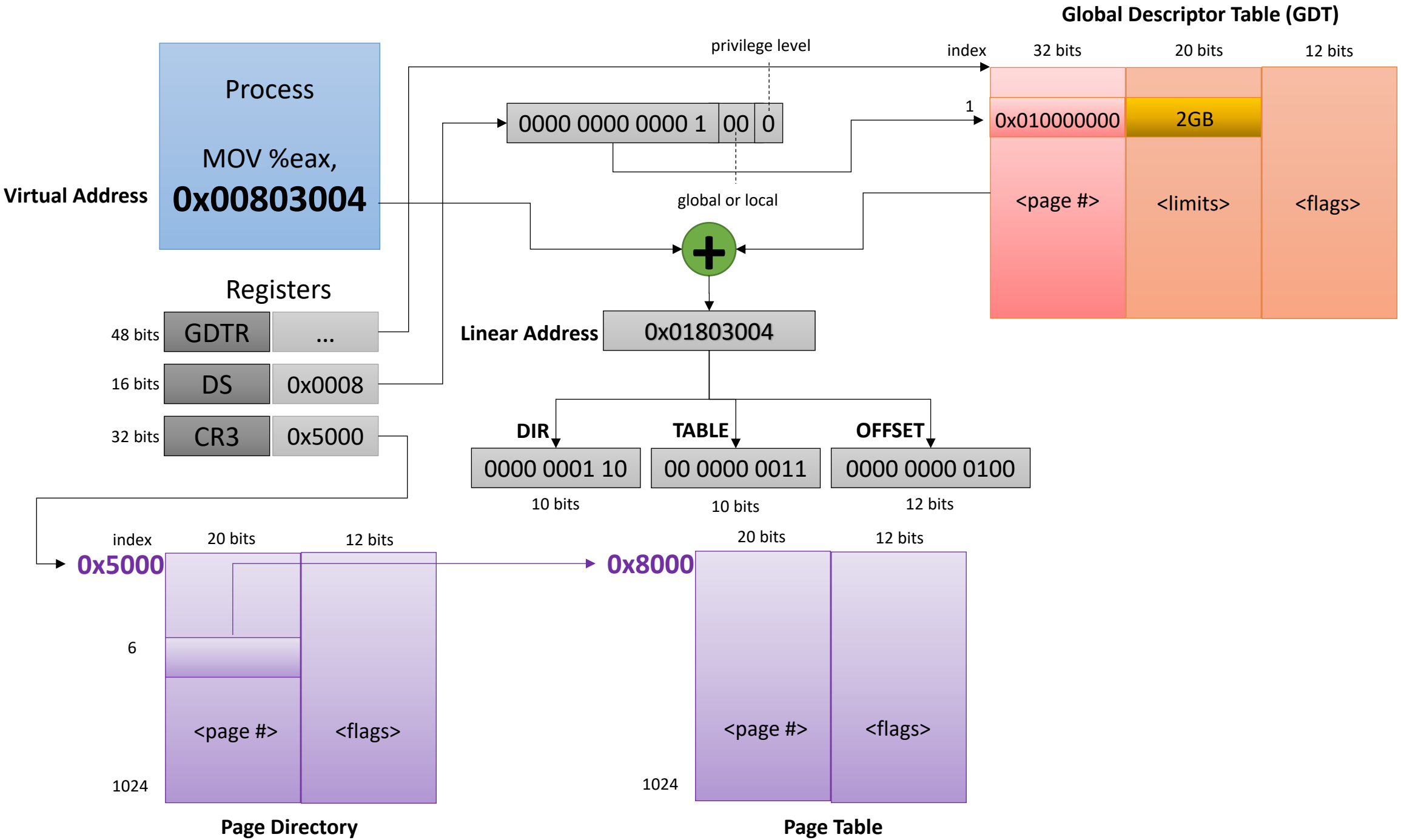
saehansy@uci.edu

A Simple Example

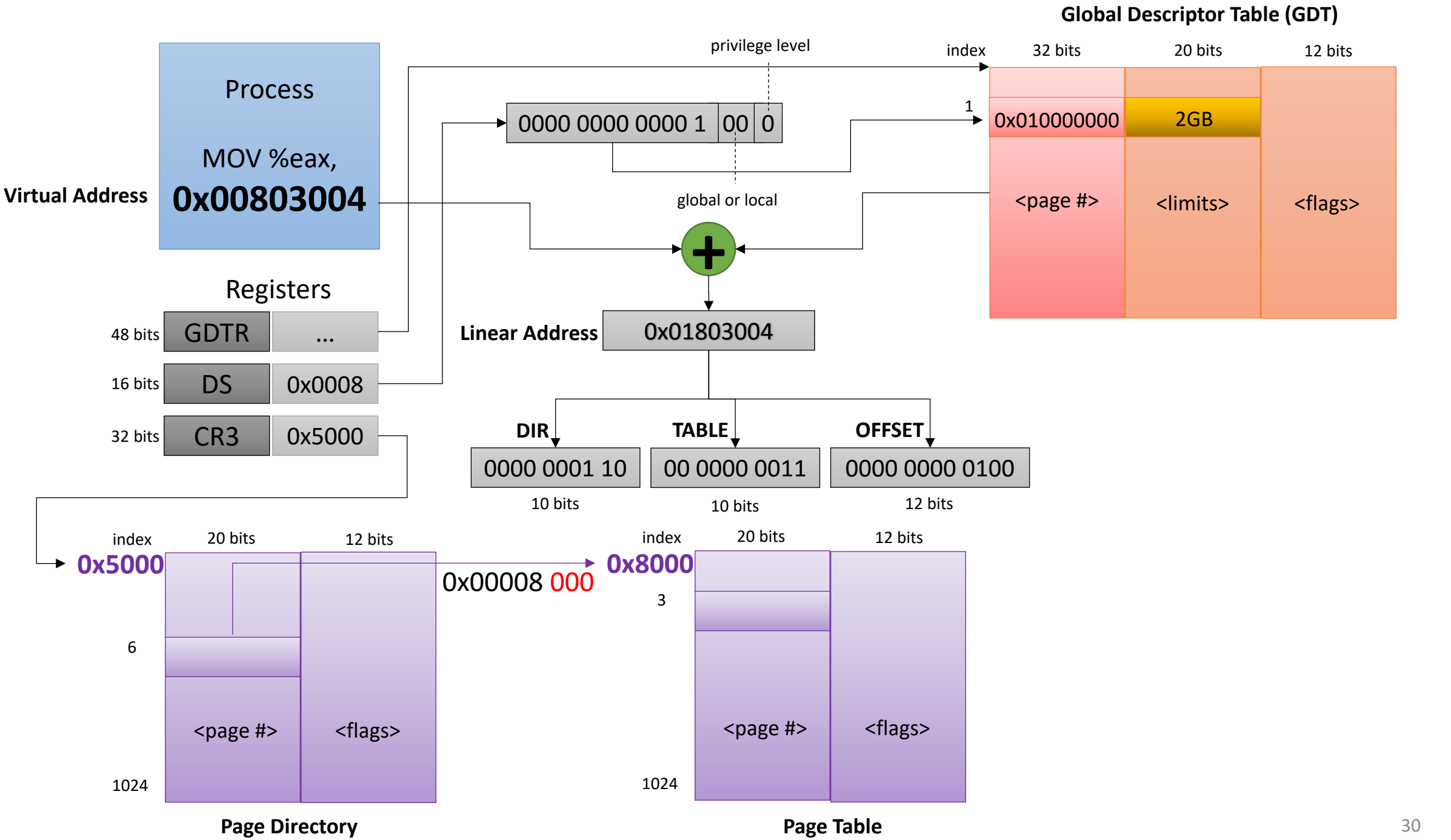
- x86, 4k page
- Logical address 0x803004 → Physical address 0x8004
- Physical address of Page Directory: 0x5000
- Physical address of the page table involved: 0x8000
- entry[1] in Global Descriptor Table: 0x1000000, 2GB
- DS register: 0x8
- **Draw the diagram of process translation**



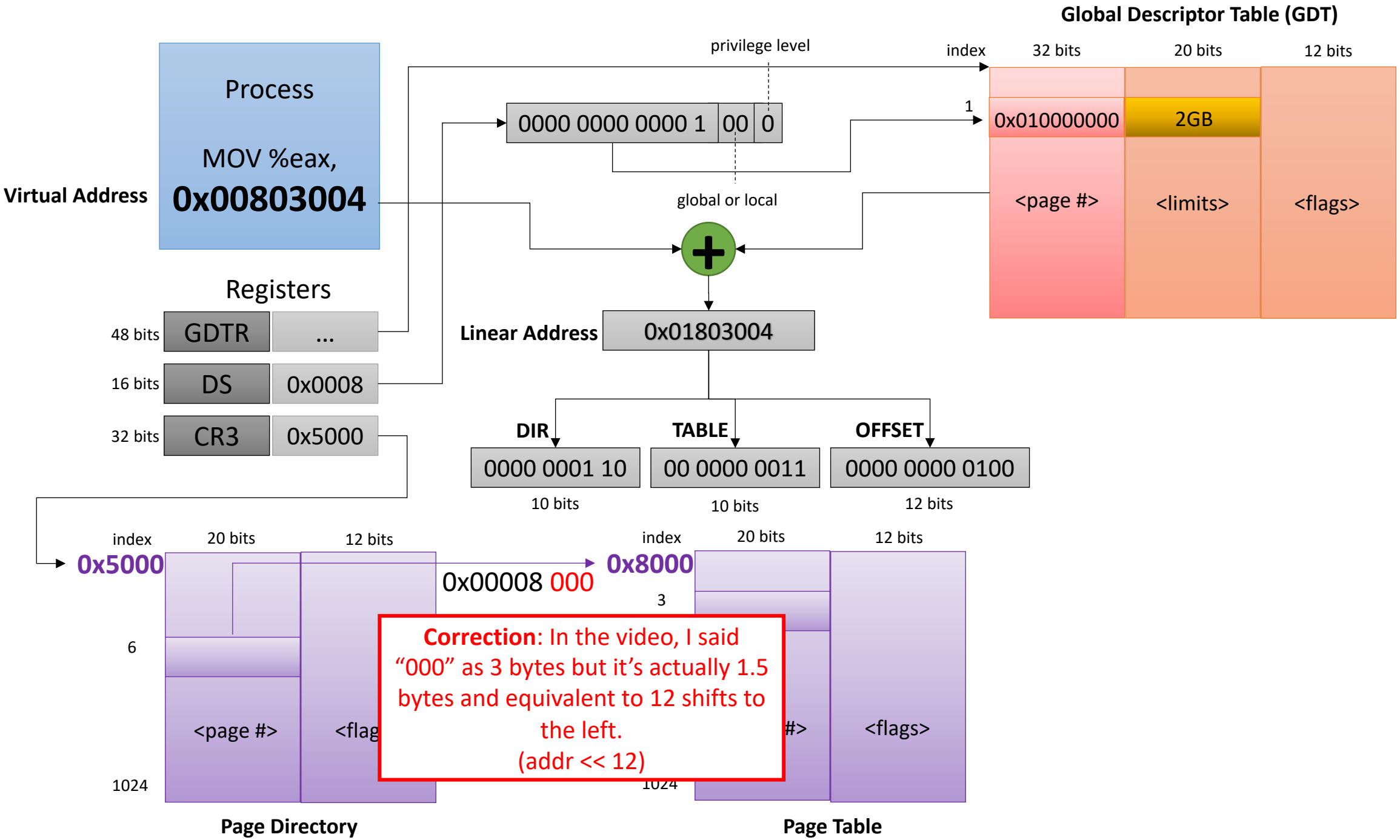
saehansy@uci.edu



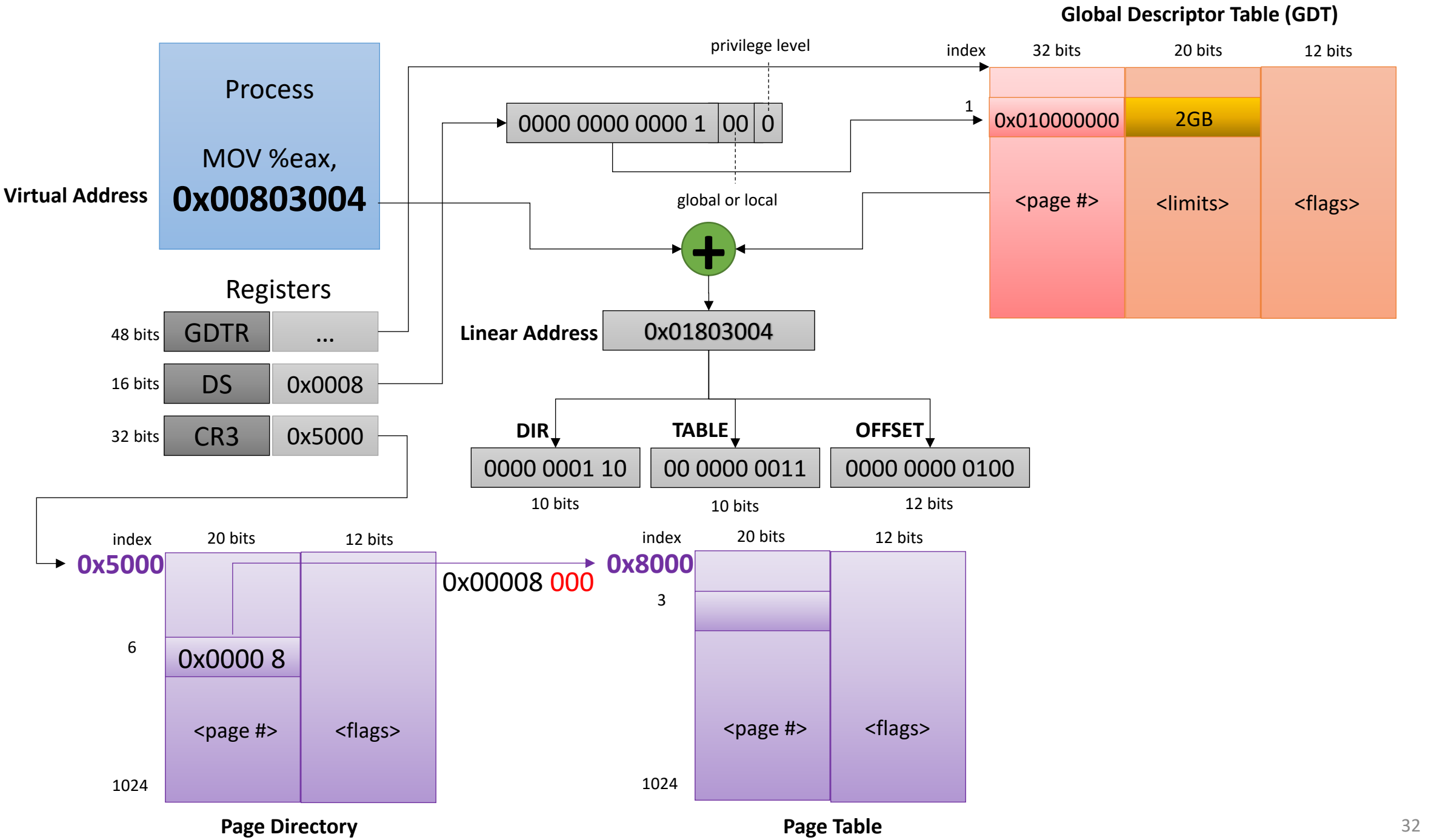
saehansy@uci.edu



saehansy@uci.edu



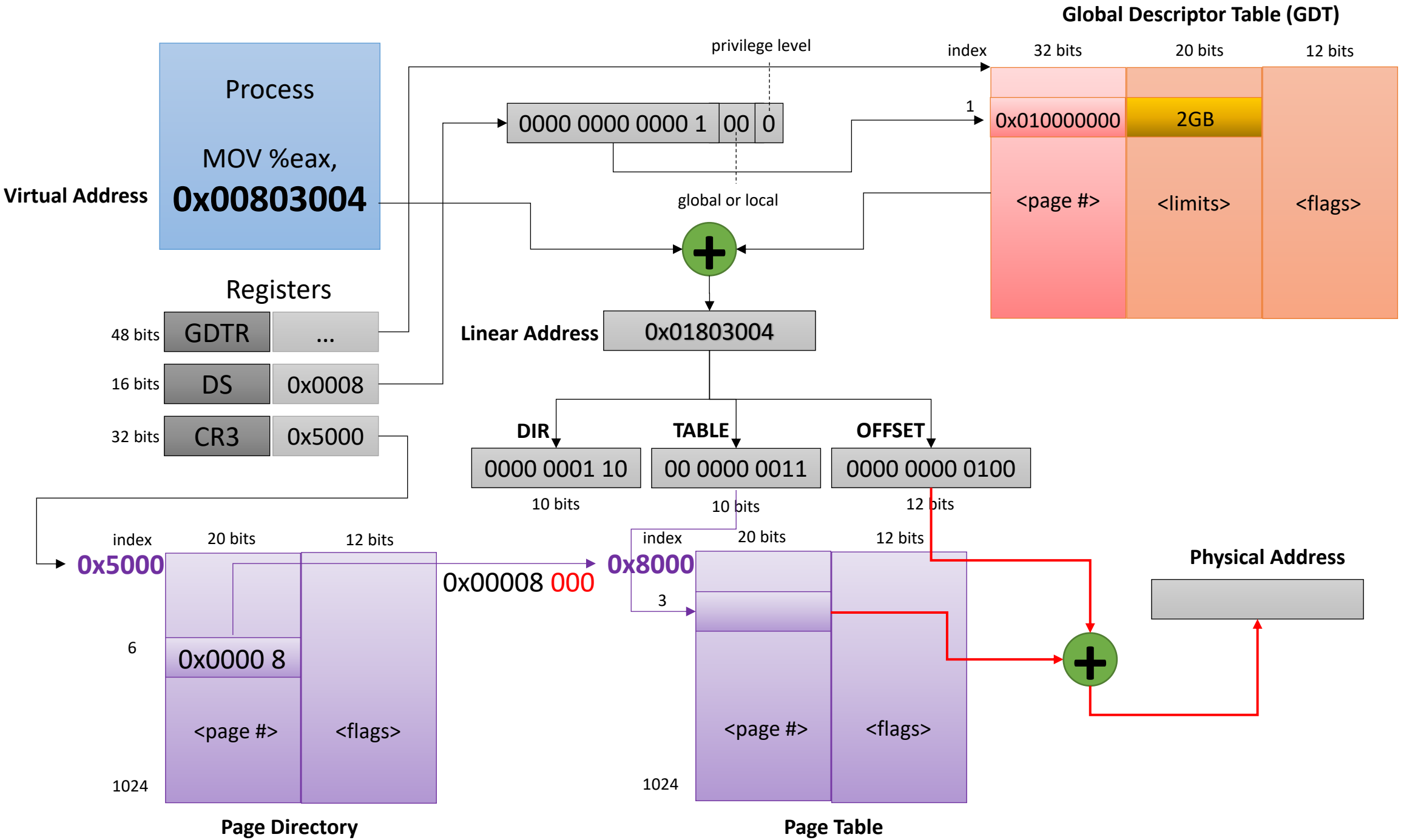
saehansy@uci.edu



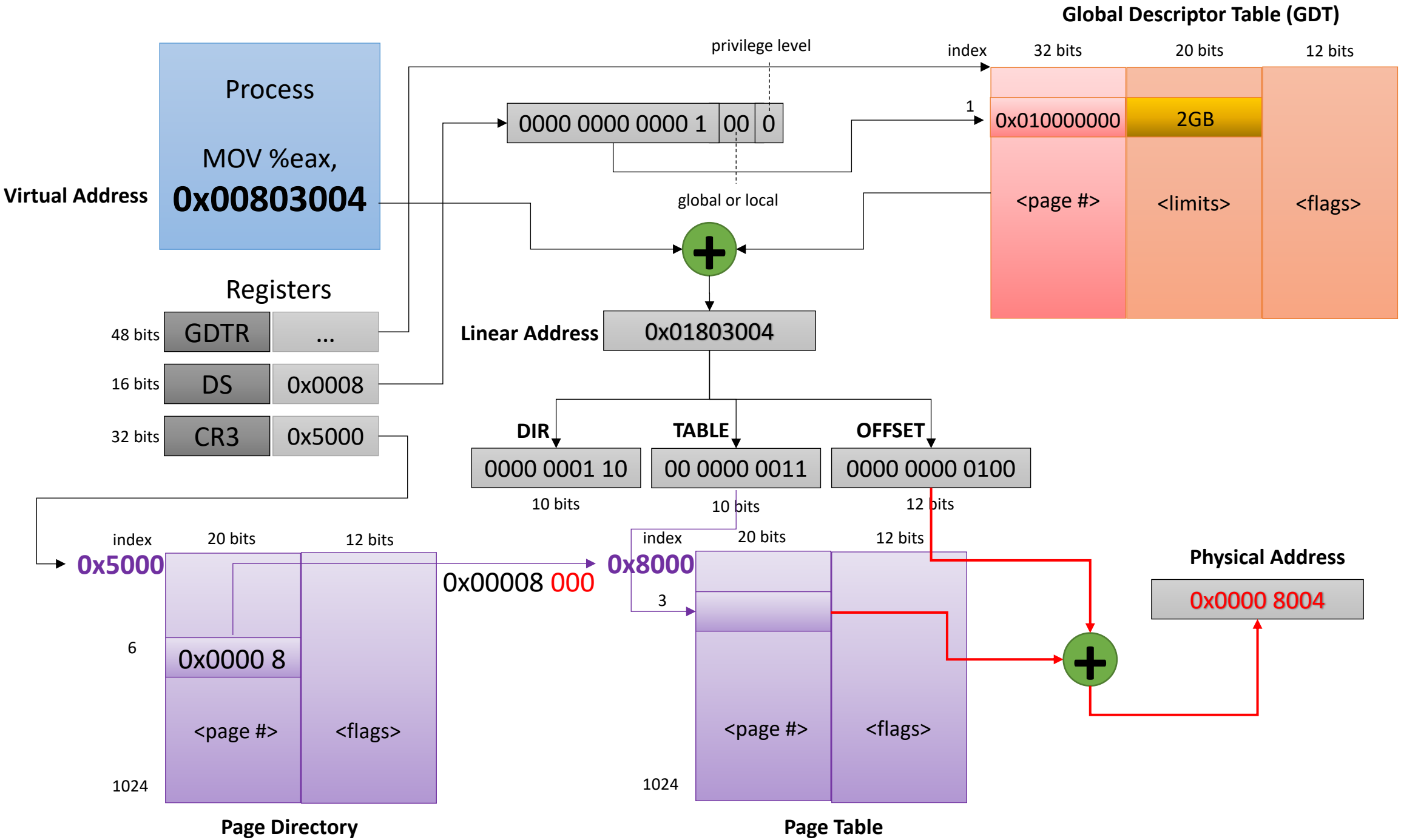
saehansy@uci.edu

A Simple Example

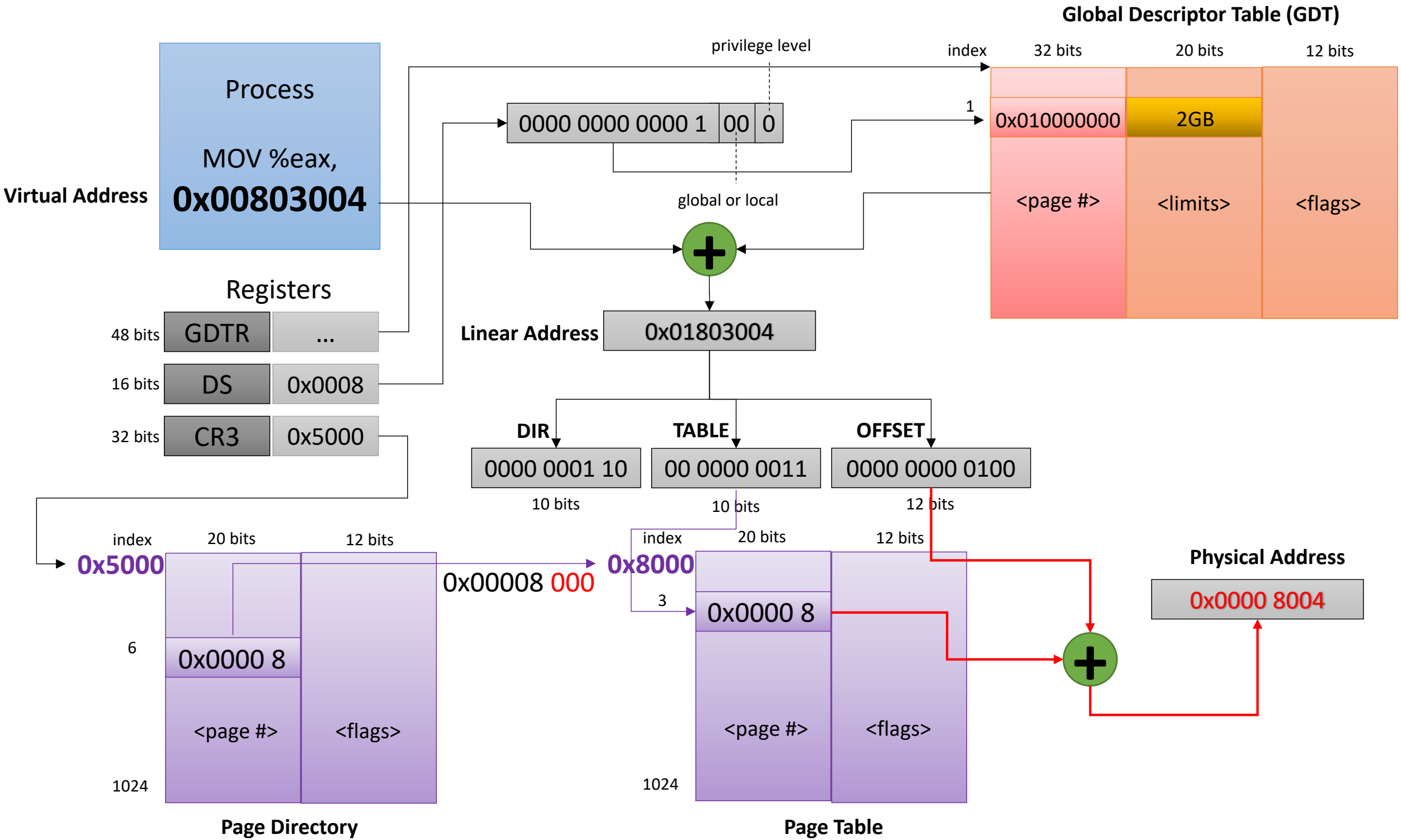
- x86, 4k page
- Logical address 0x803004 → Physical address 0x8004
- Physical address of Page Directory: 0x5000
- Physical address of the page table involved: 0x8000
- entry[1] in Global Descriptor Table: 0x1000000, 2GB
- DS register: 0x8
- **Draw the diagram of process translation**



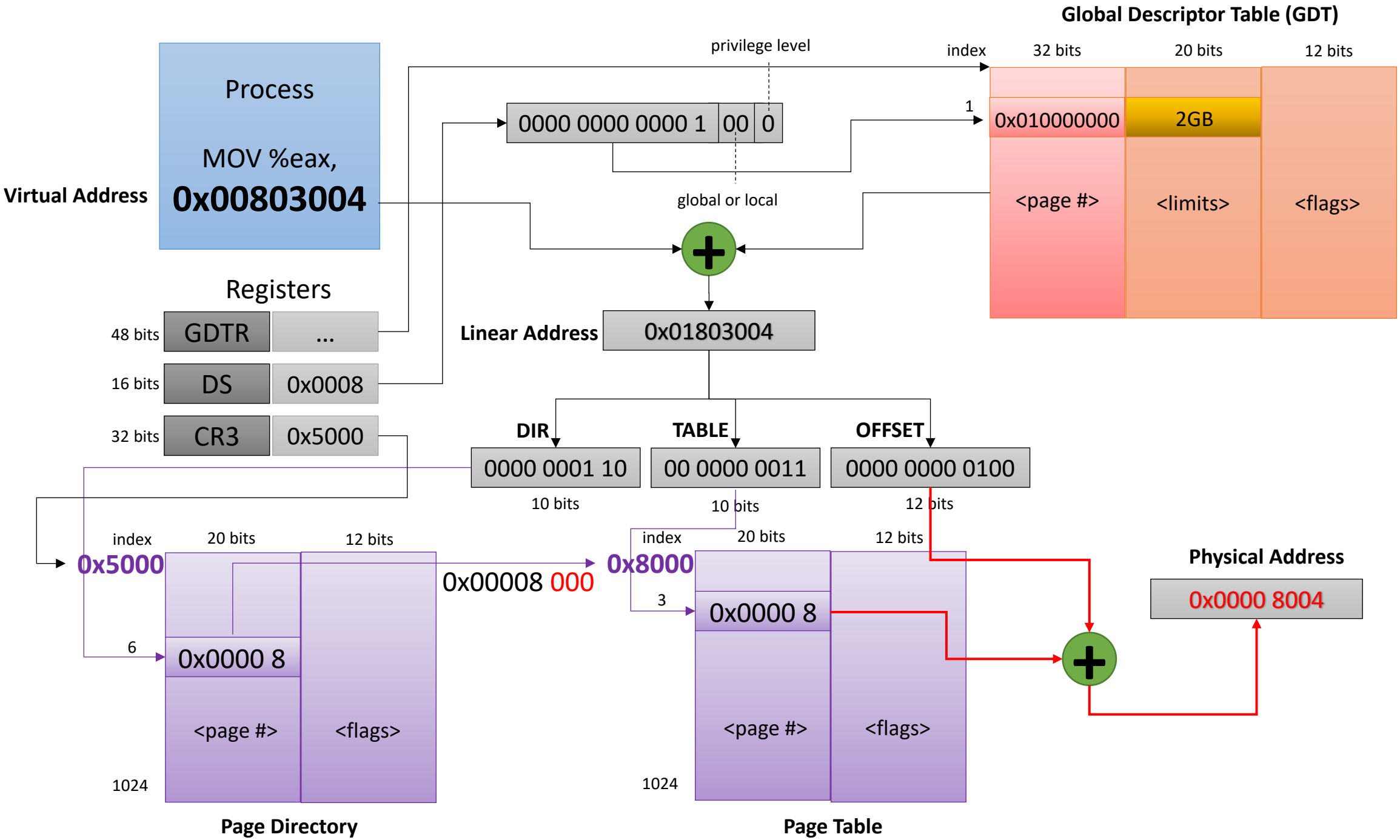
saehansy@uci.edu



saehansy@uci.edu



saehansy@uci.edu



saehansy@uci.edu