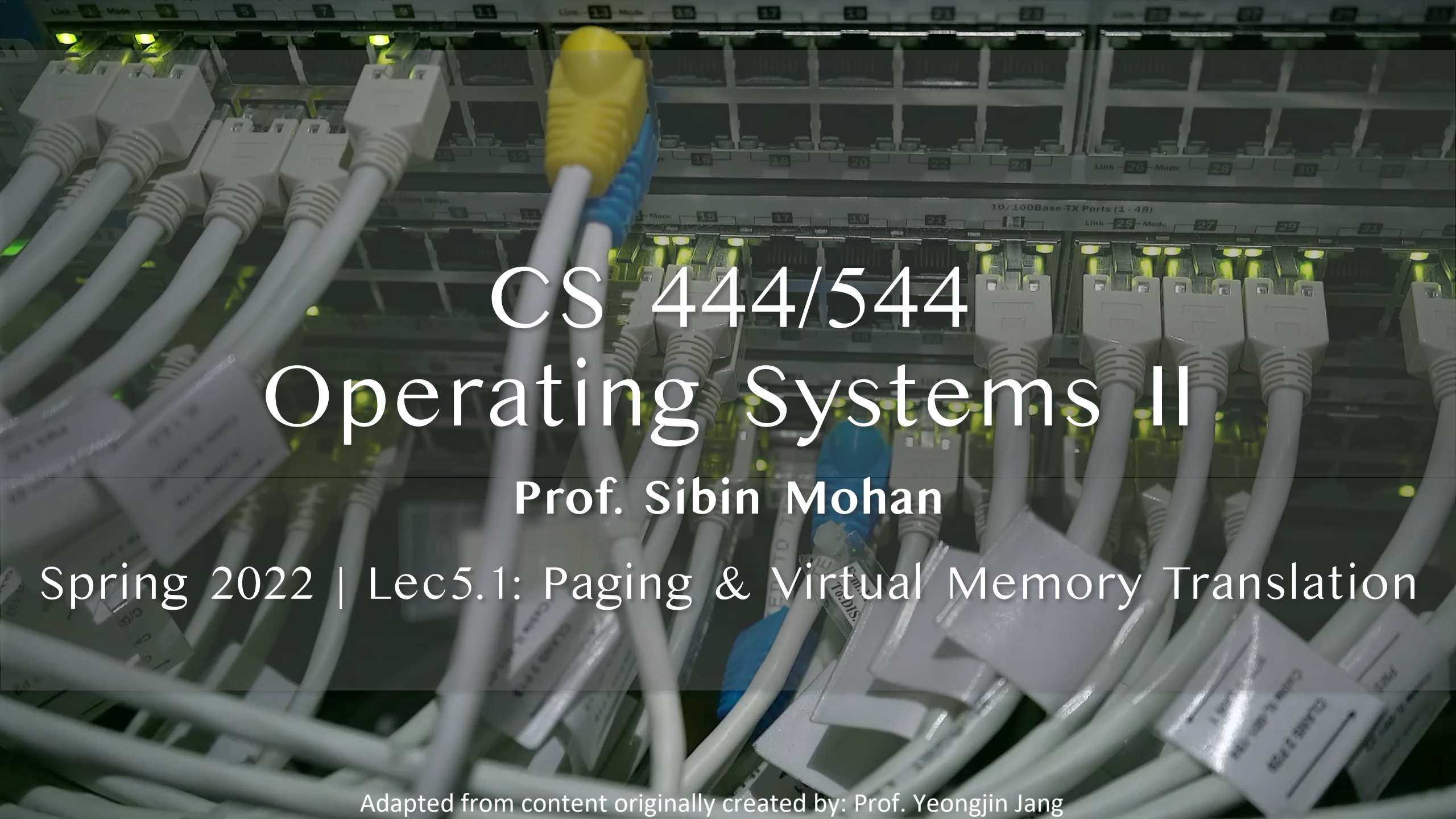
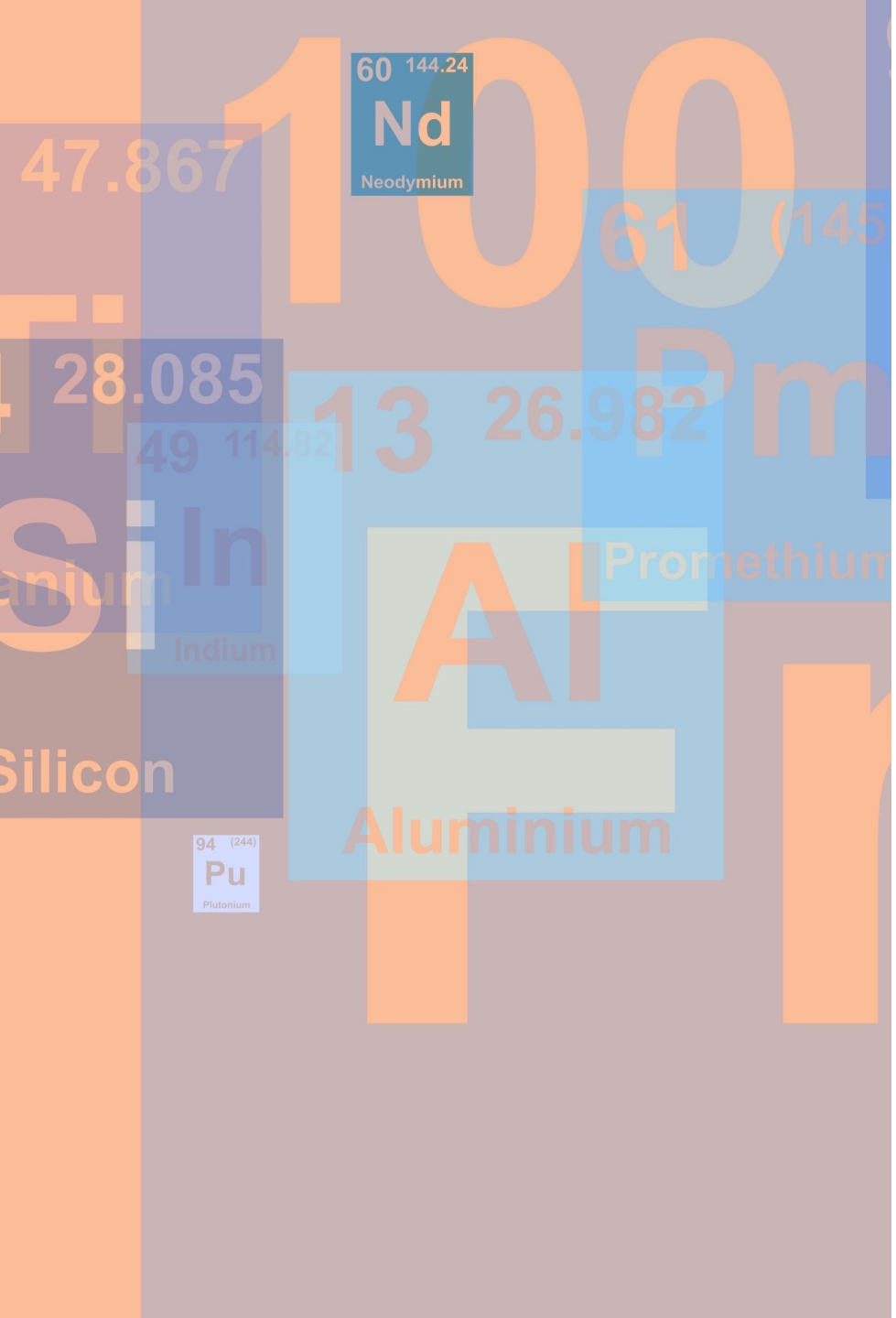


CS 444/544 Operating Systems II

Prof. Sibin Mohan

Spring 2022 | Lec5.1: Paging & Virtual Memory Translation

Adapted from content originally created by: Prof. Yeongjin Jang



Recap

- Address Translation and Memory Architectures
- TLB

Virtual Memory

Each process gets its own address space

Has access to **full memory space**

(e.g., 4 GB in 32 bit), maybe more!

Virtual memory mechanism and OS

Manage placement in/mapping to physical memory

Paging

Transparent to the user/programs

Efficient use of physical memory resources

Enforce **isolation**

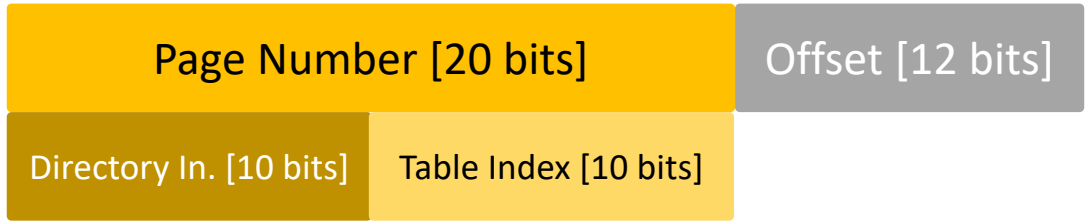
Memory Address Translation

Virtual Address

Page Directory

Memory Address Translation

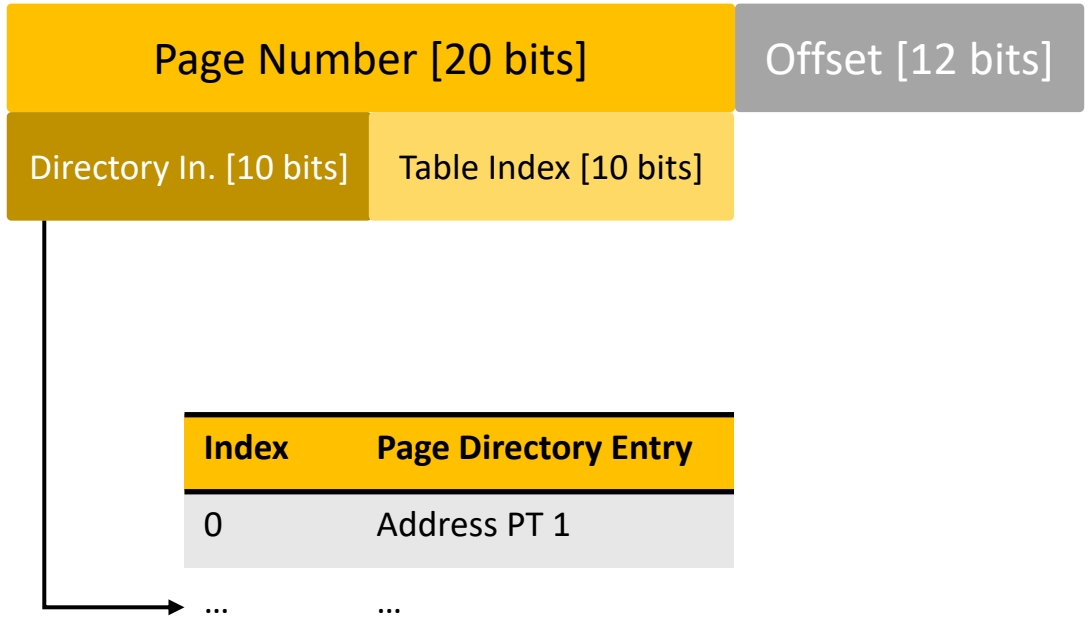
Virtual Address



Page Directory

Memory Address Translation

Virtual Address

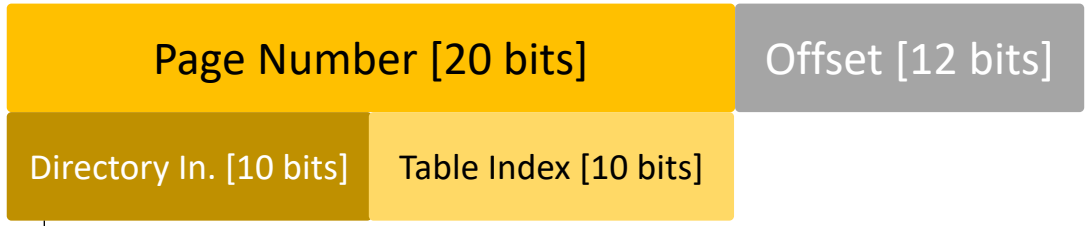


Index	Page Directory Entry
0	Address PT 1
…	…
0x20	Address PT 10
0x3ff	Address PT 50

Page Directory

Memory Address Translation

Virtual Address



Index	Page Directory Entry
0	Address PT 1
…	…
0x20	Address PT 10
0x3ff	Address PT 50

Page Directory

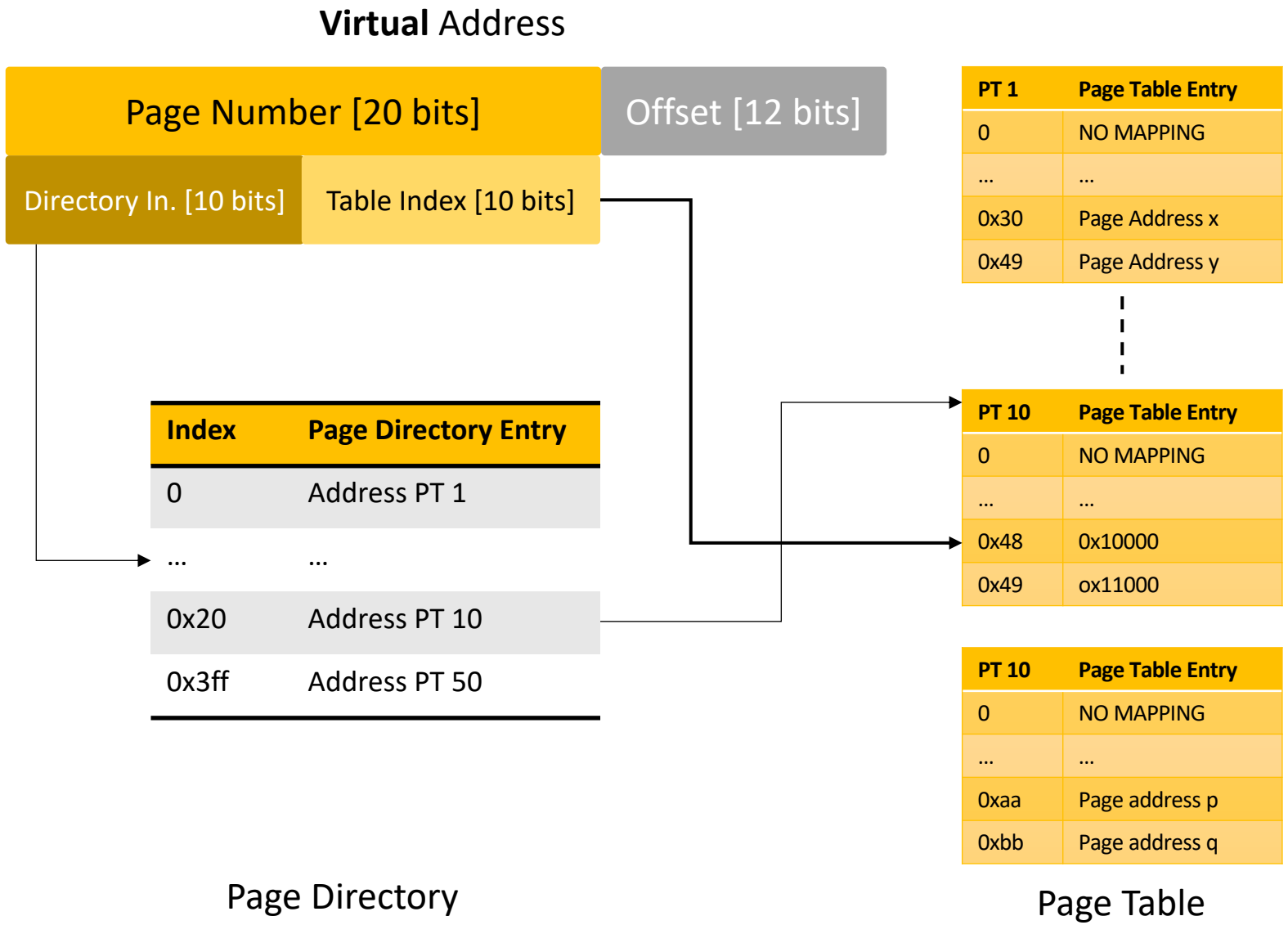
PT 1	Page Table Entry
0	NO MAPPING
…	…
0x30	Page Address x
0x49	Page Address y

PT 10	Page Table Entry
0	NO MAPPING
…	…
0x48	0x10000
0x49	ox11000

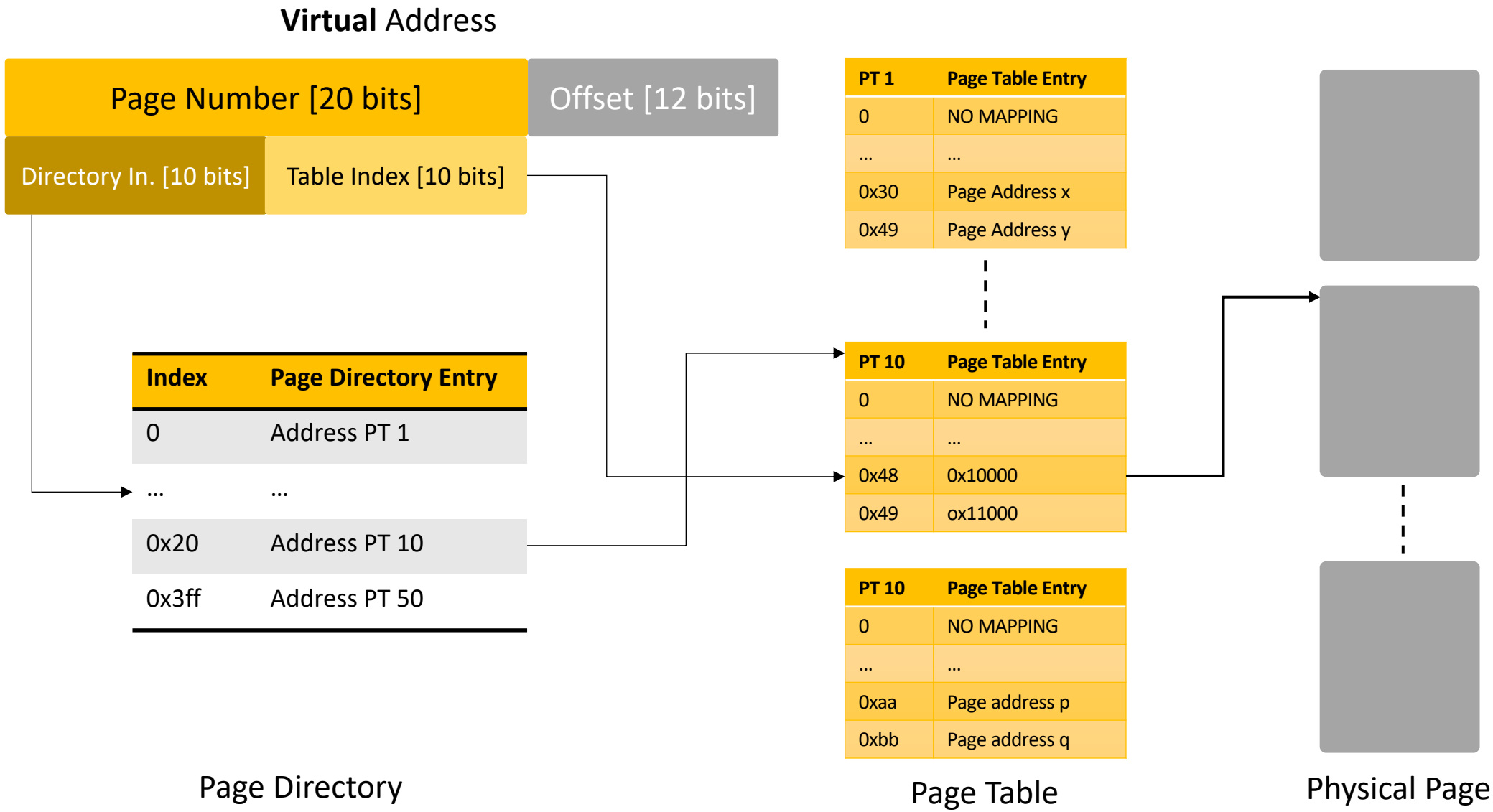
PT 10	Page Table Entry
0	NO MAPPING
…	…
0xaa	Page address p
0xbb	Page address q

Page Table

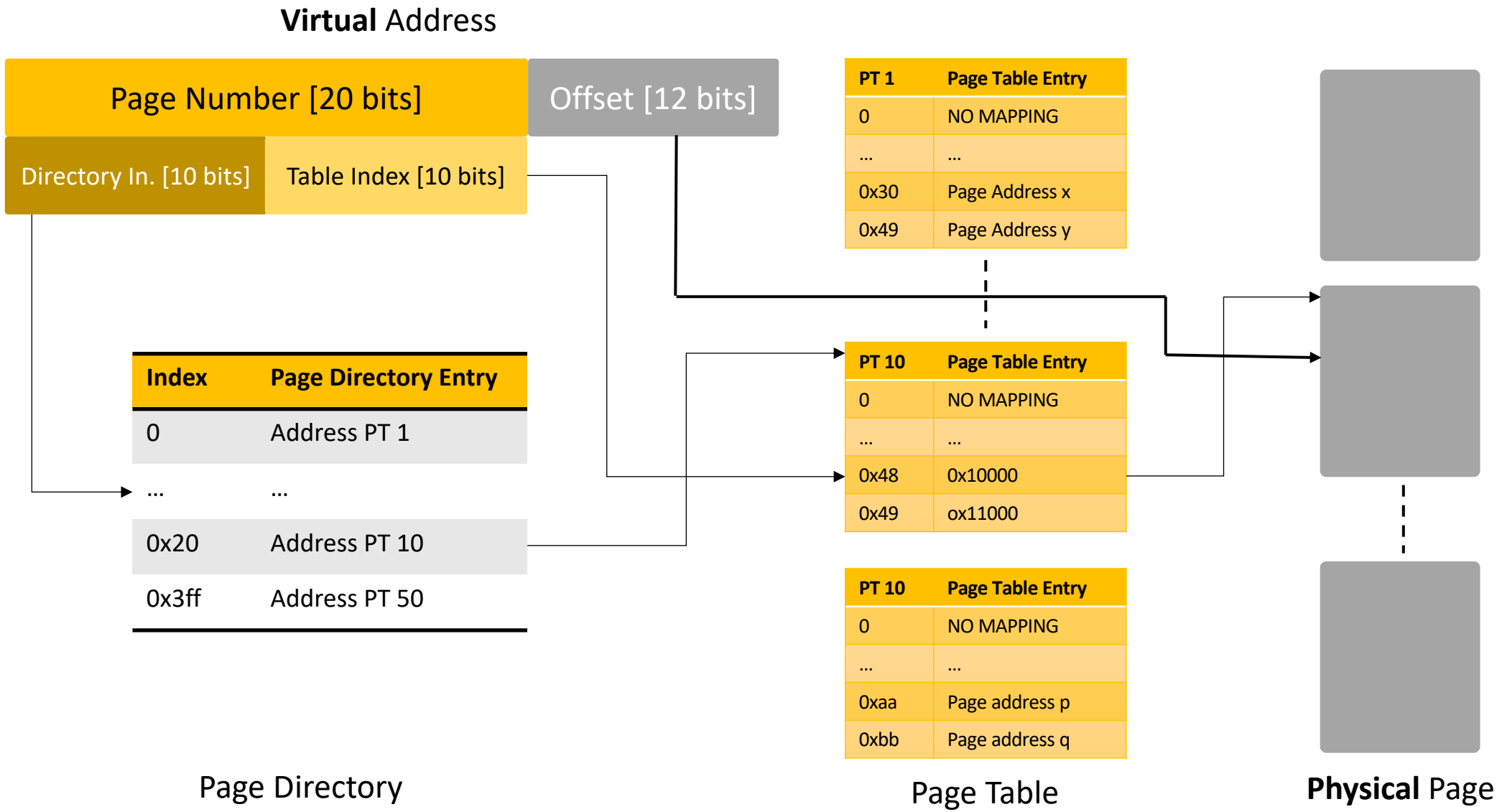
Memory Address Translation



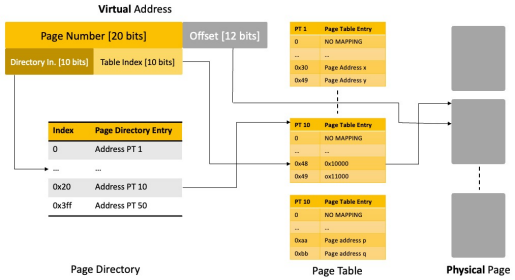
Memory Address Translation



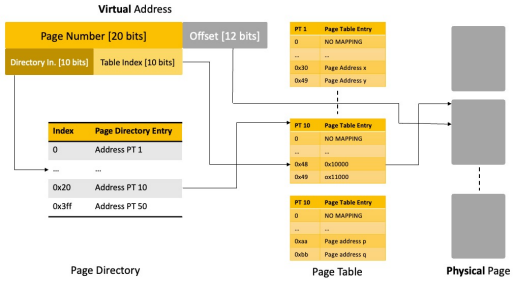
Memory Address Translation



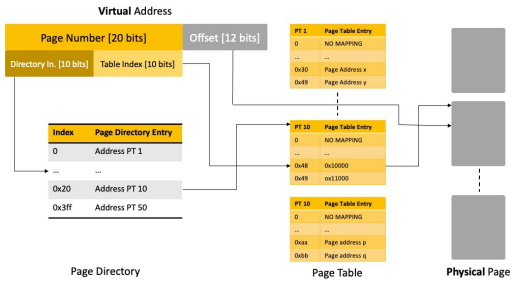
Memory Address Translation



Process 1 Page Table



Process 2 Page Table

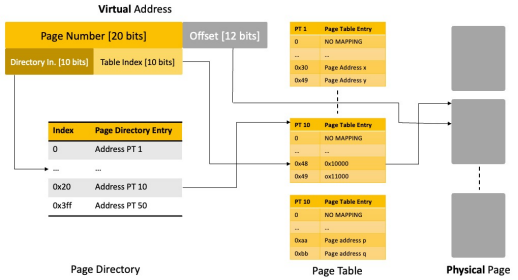


Process N Page Table

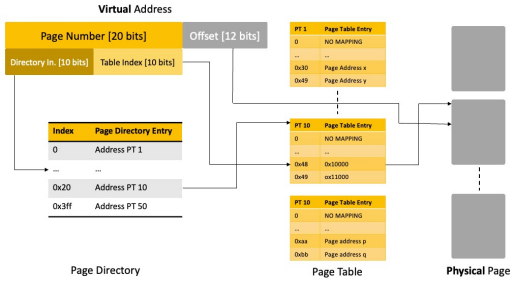
Memory Address Translation

CR3 REGISTER

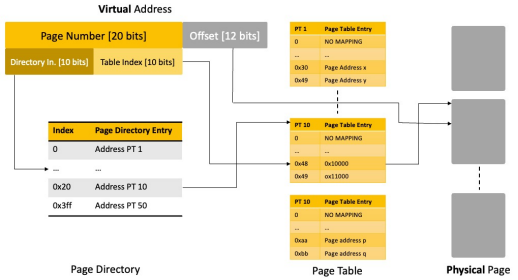
Point to the root of the PDE



Process 1 Page Table



Process 2 Page Table

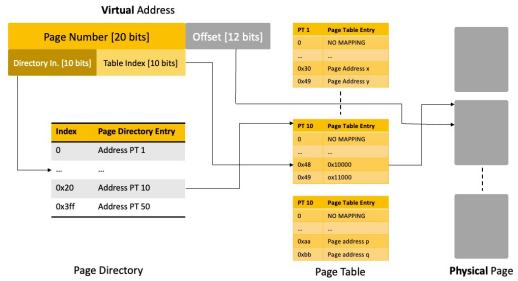


Process N Page Table

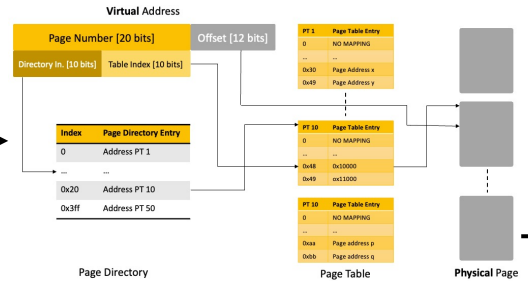
Memory Address Translation

CR3 REGISTER

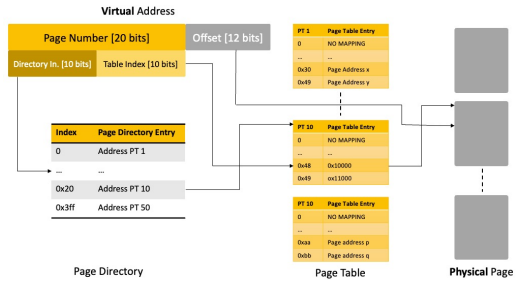
Point to the root of the PDE



Process 1 Page Table



Process 2 Page Table



Process N Page Table

VPN	PPN	Valid
0x12345	0x0	1
0x12346	0x5	1
0x12347	0xff	1
0x12348	0xfff	1
0	0	0

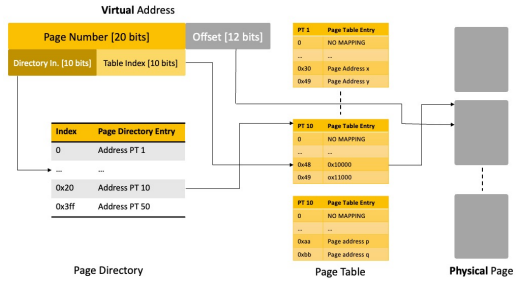
TLB

Caches most recent translation

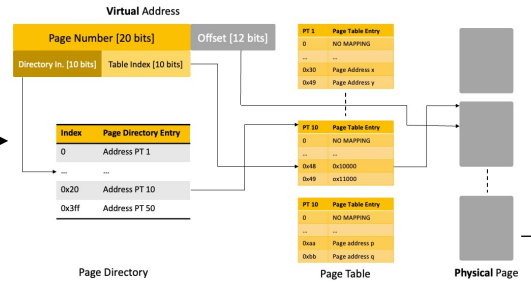
Memory Address Translation

CR3 REGISTER

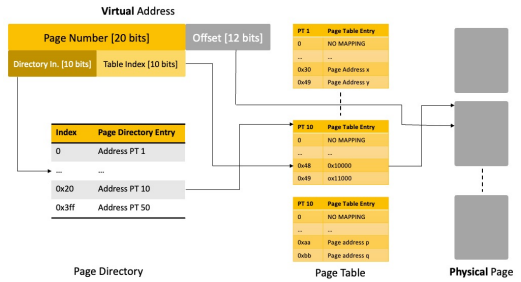
Point to the root of the PDE



Process 1 Page Table



Process 2 Page Table



Process N Page Table

VPN	PPN	Valid
0x12345	0x0	1
0x12346	0x5	1
0x12347	0xff	1
0x12348	0xfff	1
0	0	0

TLB

Caches most recent translation

Can avoid expensive address translation process for cached addresses!

TLB | Issues to Consider

- **Limited size**
  - Lots of programs and page tables
- **Stale entries**
  - What if process changes (context switch) or
  - TLB entry is no longer valid?

TLB Replacement Policy

- **Smart: LRU** (Least Recently Used)
  - Mark when the block was accessed
    - update timestamp access
  - When an eviction is necessary, evict the **oldest timestamp**
- **Random**
  - Do not do anything when accessed
  - When an eviction is required, evict a **random entry**
  - Sometimes, this works better than LRU!
- We will learn more later
  - process scheduling

Synchronizing TLB with Page Table

- CPU uses the TLB for caching Page Table Entries

- If **mismatch** between content in TLB and PTE?
  - Accesses wrong physical memory
  - Does not honor the correct privilege in PTE (if we updated PTE after caching)
  - Running a new process with new CR3
    - Use old process's mapping, wrong access

TLB | Page Table Update

Address
0x12349678

TLB

VPN	PPN	Valid
0x12345	0x0	1
0x12346	0x5	1
0x12347	0xff	1
0x12348	0xfff	1
0x12349	0x101	1

	Page Directory Entry
0	Addr PT
..	Addr PT
0x48	Addr PT
0x3ff	Addr PT

	Page Table Entry
0	Addr PT
0x347	0xff | FLAG
0x348	0xfff | FLAG
0x349	0x101 | FLAG

TLB | Page Table Update

Address
0x12349678

TLB

VPN	PPN	Valid
0x12345	0x0	1
0x12346	0x5	1
0x12347	0xff	1
0x12348	0xfff	1
0x12349	0x101	1

	Page Directory Entry
0	Addr PT
..	Addr PT
0x48	Addr PT
0x3ff	Addr PT

	Page Table Entry
0	Addr PT
0x347	0xff | FLAG
0x348	0xfff | FLAG
0x349	**0x102** | FLAG

Update

TLB | Page Table Update

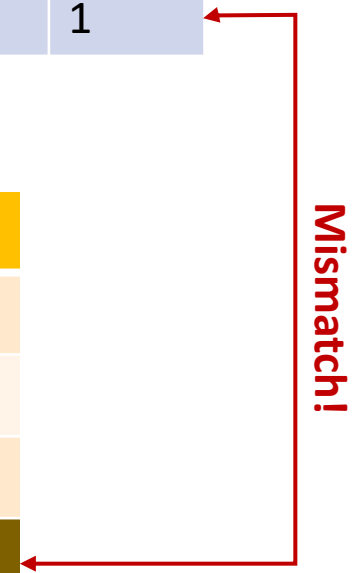
Address
0x12349678

TLB

VPN	PPN	Valid
0x12345	0x0	1
0x12346	0x5	1
0x12347	0xff	1
0x12348	0xfff	1
0x12349	0x101	1

	Page Directory Entry
0	Addr PT
..	Addr PT
0x48	Addr PT
0x3ff	Addr PT

	Page Table Entry
0	Addr PT
0x347	0xff | FLAG
0x348	0xfff | FLAG
0x349	**0x102** | FLAG



TLB | Page Table Update

Address
0x12349678

TLB

VPN	PPN	Valid
0x12345	0x0	1
0x12346	0x5	1
0x12347	0xff	1
0x12348	0xfff	1
0x12349	0x101	1

	Page Directory Entry
0	Addr PT
..	Addr PT
0x48	Addr PT
0x3ff	Addr PT

	Page Table Entry
0	Addr PT
0x347	0xff | FLAG
0x348	0xfff | FLAG
0x349	**0x102** | FLAG

Mismatch!

Process could end up accessing a page from a different process!

Or one without permissions!

TLB | Page Table Update

Address
0x12349678

TLB

VPN	PPN	Valid
0x12345	0x0	1
0x12346	0x5	1
0x12347	0xff	1
0x12348	0xfff	1
0x12349	0x101	**0**

Invalidate this entry

	Page Directory Entry
0	Addr PT
..	Addr PT
0x48	Addr PT
0x3ff	Addr PT

	Page Table Entry
0	Addr PT
0x347	0xff | FLAG
0x348	0xfff | FLAG
0x349	**0x102** | FLAG

Update

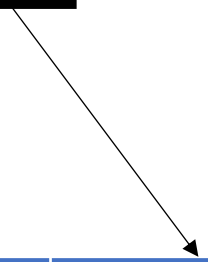
TLB | Process Context Switch

TLB

VPN	PPN	Valid
0x12345	0x0	1
0x12346	0x5	1
0x12347	0xff	1
0x12348	0xfff	1
0x12349	0x101	1

Address
0x12349678

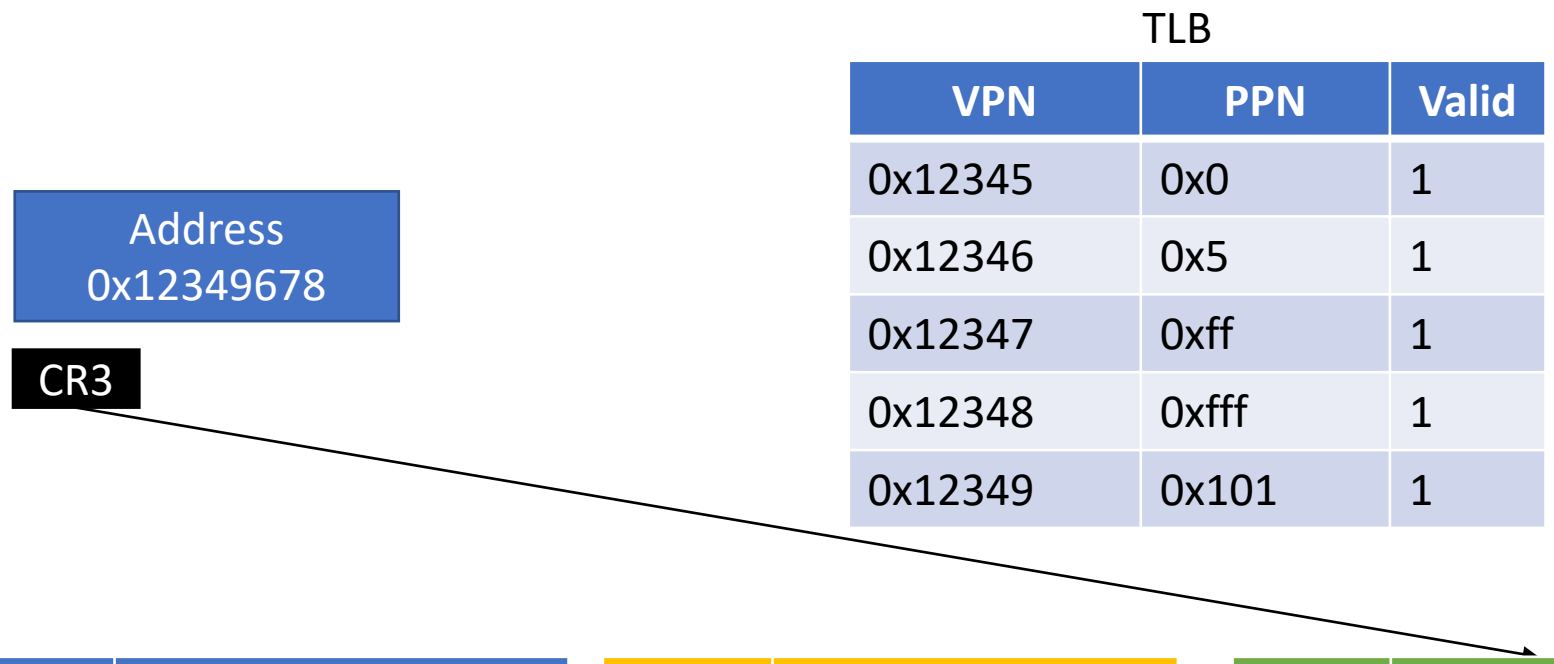
CR3



	Page Directory Entry
0	Addr PT
..	Addr PT
0x48	Addr PT
0x3ff	Addr PT

	Page Table Entry
0	Addr PT
0x347	0xff | FLAG
0x348	0xfff | FLAG
0x349	0x101 | FLAG

TLB and Process Context Switch



TLB

VPN	PPN	Valid
0x12345	0x0	1
0x12346	0x5	1
0x12347	0xff	1
0x12348	0xfff	1
0x12349	0x101	1

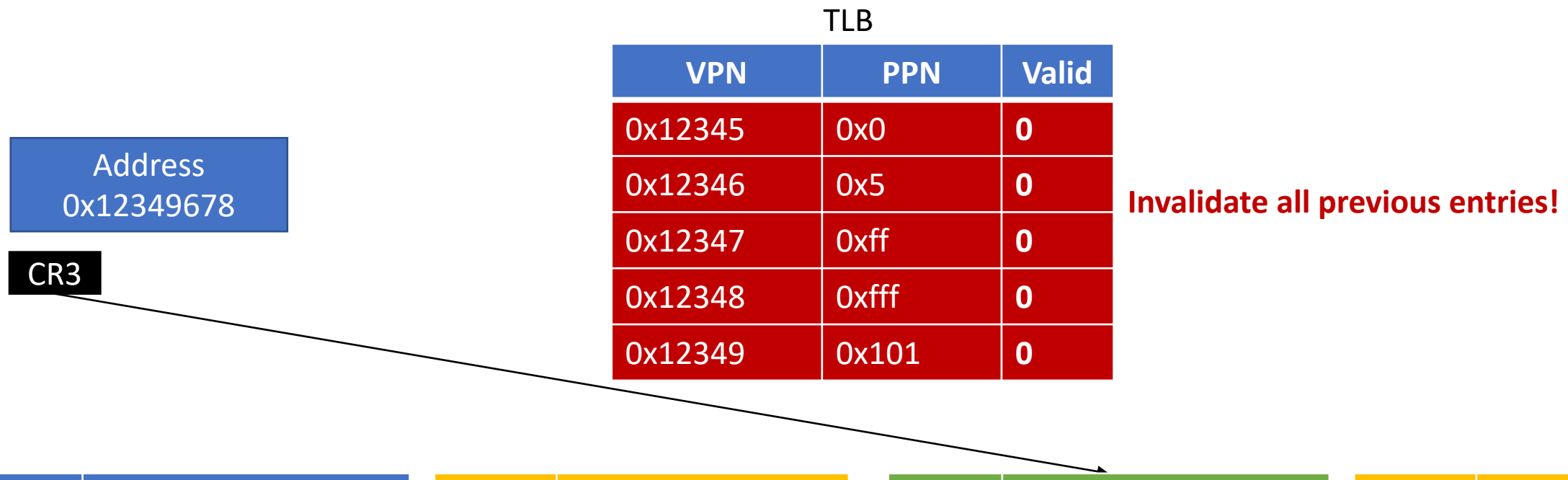
	Page Directory Entry
0	Addr PT
..	Addr PT
0x48	Addr PT
0x3ff	Addr PT

	Page Table Entry
0	Addr PT
0x347	0xff | FLAG
0x348	0xfff | FLAG
0x349	0x101 | FLAG

	Page Directory Entry
0	Addr PT
..	Addr PT
0x48	Addr PT
0x3ff	Addr PT

	Page Table Entry
0	Addr PT
0x347	**0x20** | FLAG
0x348	**0x30** | FLAG
0x349	**0x50** | FLAG

TLB and Process Context Switch



	Page Directory Entry
0	Addr PT
..	Addr PT
0x48	Addr PT
0x3ff	Addr PT

	Page Table Entry
0	Addr PT
0x347	0xff | FLAG
0x348	0xfff | FLAG
0x349	0x101 | FLAG

	Page Directory Entry
0	Addr PT
..	Addr PT
0x48	Addr PT
0x3ff	Addr PT

	Page Table Entry
0	Addr PT
0x347	**0x20** | FLAG
0x348	**0x30** | FLAG
0x349	**0x50** | FLAG

Updating Page Table

- When updating a Page Table Entry
  - invalidate TLB for that entry
  - **`invlpg`**

INVLPG — Invalidate TLB Entries

		Op/En	64-Bit Mode	Compat/Leg Mode	Description
		M	Valid	Valid	Invalidate TLB entries for page containing *m*.

Updating Page Table

- In JOS (kern/pmap.c & inc/x86.h)

```
//
// Invalidate a TLB entry, but only if the page tables being
// edited are the ones currently in use by the processor.
//
void
tlb_invalidate(pde_t *pgdir, void *va)
{
    // Flush the entry only if we're modifying the current address space.
    // For now, there is only one address space, so always invalidate.
    invlpg(va);
}
```

```
static inline void
invlpg(void *addr)
{
    asm volatile("invlpg (%0)" : : "r" (addr) : "memory");
}
```

Manipulating Page Tables in JOS

- **`PGNUM(x)`**
  - Get the page number (x >> 12) of the address x
- **`PDX(x)`**
  - Get the page directory index (top 10 bits) of the address x
- **`PTX(x)`**
  - Get the page table index (mid 10 bits) of the address x
- **`PGOFF(x)`**
  - Get the page offset (lower 12 bits) of the address x

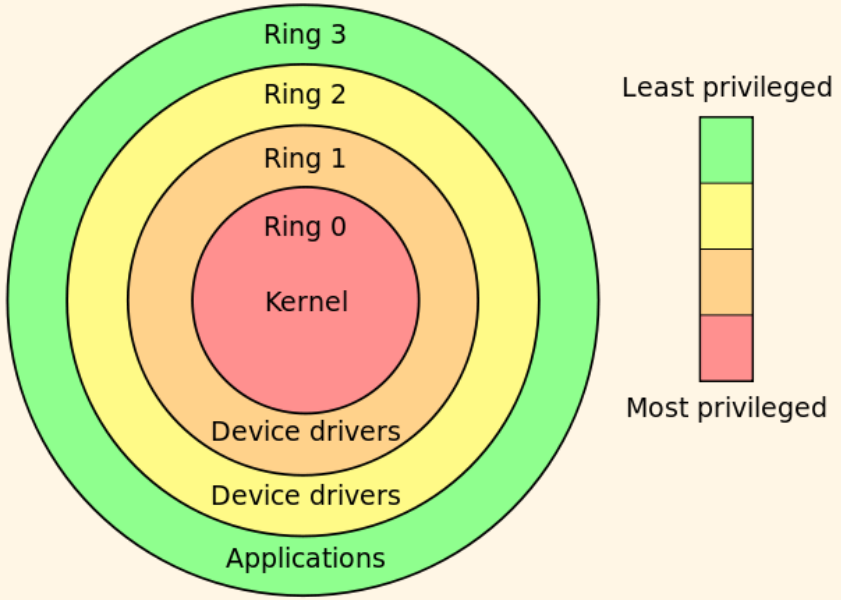
Manipulating Page Tables in JOS

- **`PTE_ADDR(pte)`**
  - Get the physical address that a pte points
  - **`PTE_ADDR(pte) + PGOFF(x)`**
    - The physical address pointed by the PTE, translated by a virtual address x

- Translate a virtual address **`va`** to physical address **`pa`**
- **`pte = CR3[PDX(va)][PTX(va)]`**
  - `pte_t **cr3 = lcr3();` # or, kern_pgdir
  - `pte_t *pt = cr3[PDX(va)]`
  - `pte_t pte = pt[PTX(va)]`
- **`physaddr_t pa = PTE_ADDR(pte) + PGOFF(va);`**

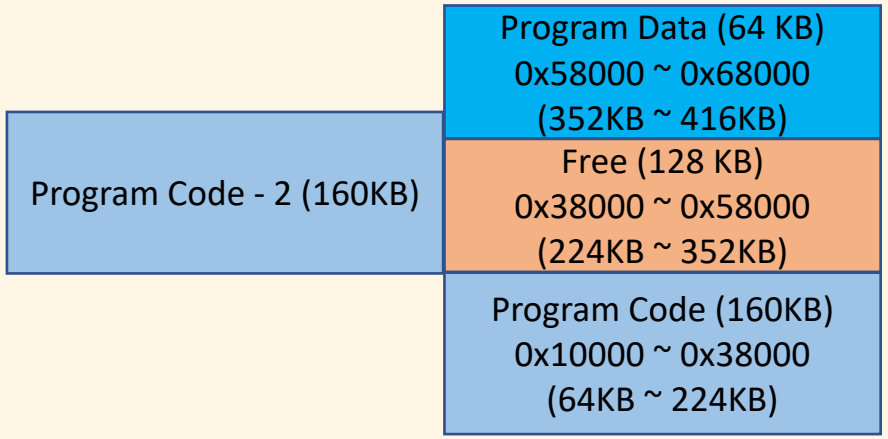
Pros/Cons of Segmentation [PROS]

- **Easy implementation** (in hardware)
  - Select [Base address] and add offset
- **Small memory** requirement
  - 8 byte for address translation in GDT (per region)
- Can allocate by the exact size (within 20bits limit)
  - Base = 0x10000, limit = 0x10
- Memory protection (DPL)



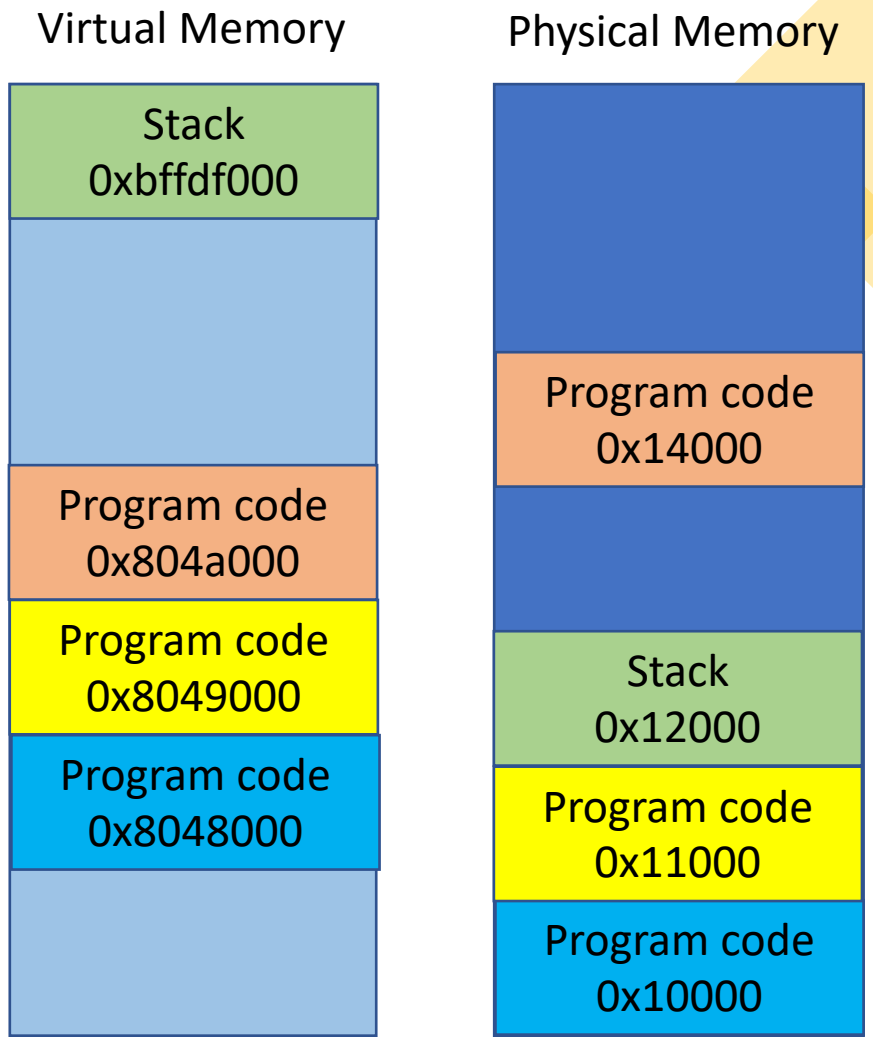
Cons of Segmentation

- **Fragmentation**
- Must allocate **contiguous space** for the region
  - Need to search for such region (regarding size)
  - Free: need to consolidate free memory chunks, etc..
- Memory protection placed for the **entire region**



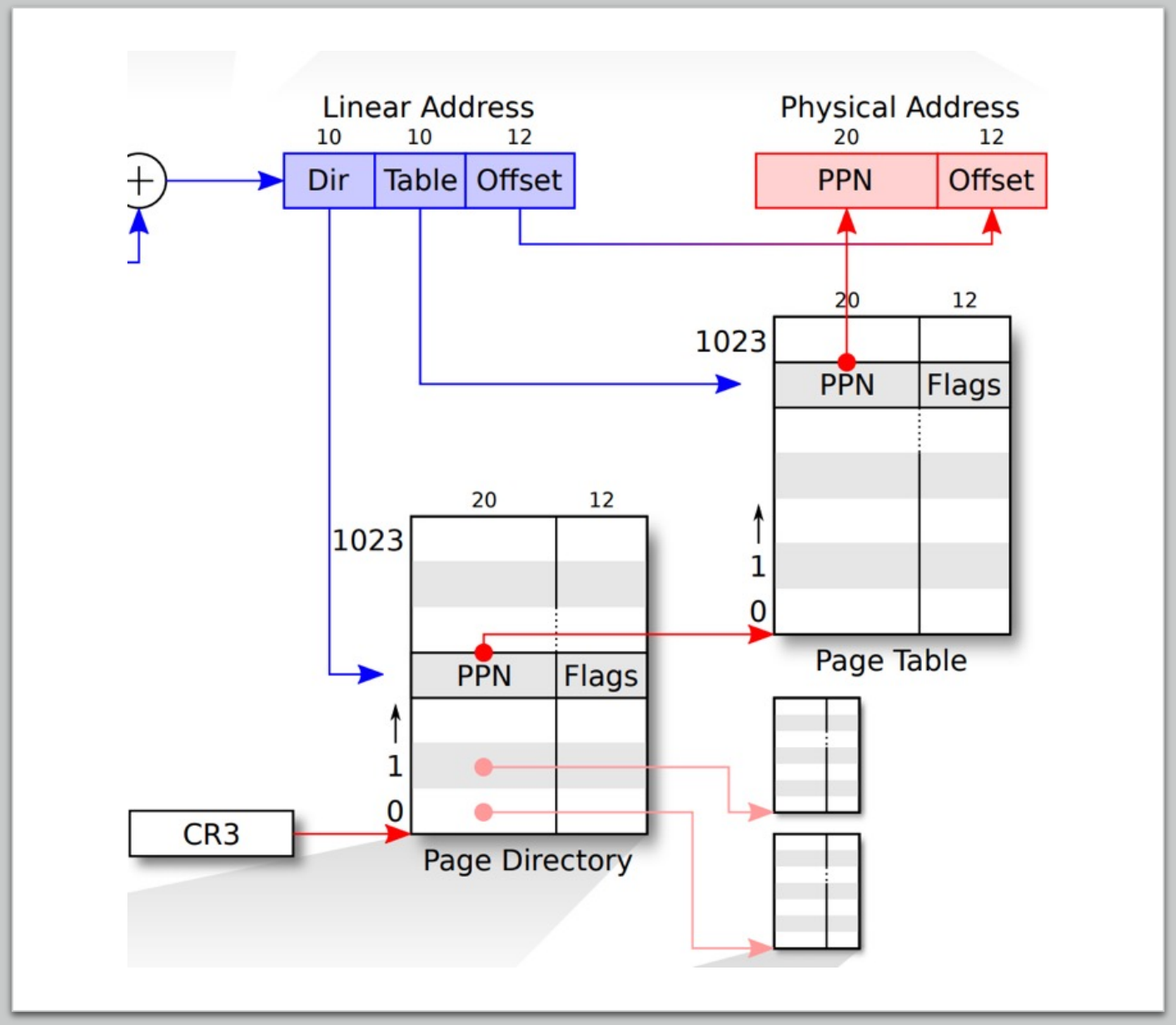
Pros/Cons of Paging [**Pros**]

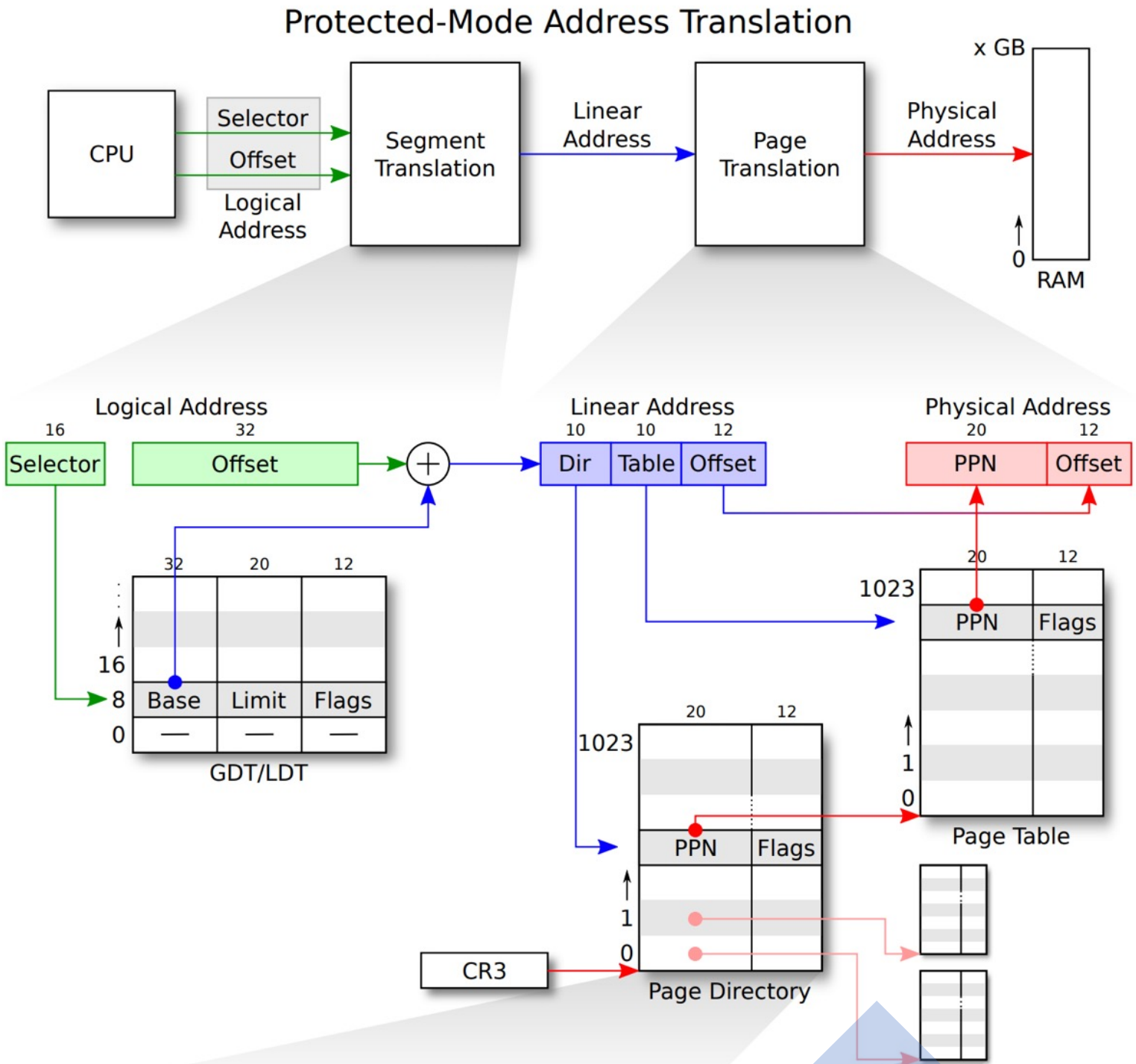
- **No fragmentation** (split into 4KB each blocks)
- **No requirements for contiguous space**
  - 0x8048000 -> 0x10000
  - 0x8049000 -> 0xff000
  - 0x804a000 -> 0xe0000
- **Fast allocation**
  - No need to search available blocks
  - just check how many pages are available
  - Free: removing PTE!
- **Memory protection can be enforced in page granularity**
  - Part of region could have a different protection bits



Cons of Paging

- **Internal fragmentation wastes memory**
  - 1 byte – allocate 4KB
  - 10 byte – allocate 4KB
- **Storage** for Page Table
  - 4MB for the full page table
  - At least 8KB (2 pages, one for PD and the other one for PT) is required
- **Page table lookup** for translation
  - Slow, requires complex hardware




Protected-Mode Address Translation
CPU
Selector
Offset
Logical Address
Segment Translation
Linear Address
Page Translation
Physical Address
x GB
0
RAM
Logical Address
16
32
Selector
Offset
Linear Address
10
10
12
Dir
Table
Offset
Physical Address
20
12
PPN
Offset
32
20
12
16
8
0
Base
Limit
Flags
GDT/LDT
1023
PPN
Flags
1
0
Page Table
1023
PPN
Flags
CR3
Page Directory

Additional Reading

- Address translation and protection (hardware-centric view)

  https://web.csl.cornell.edu/courses/ece4750/handouts/ece4750-T16-mem-translation-protection.pdf