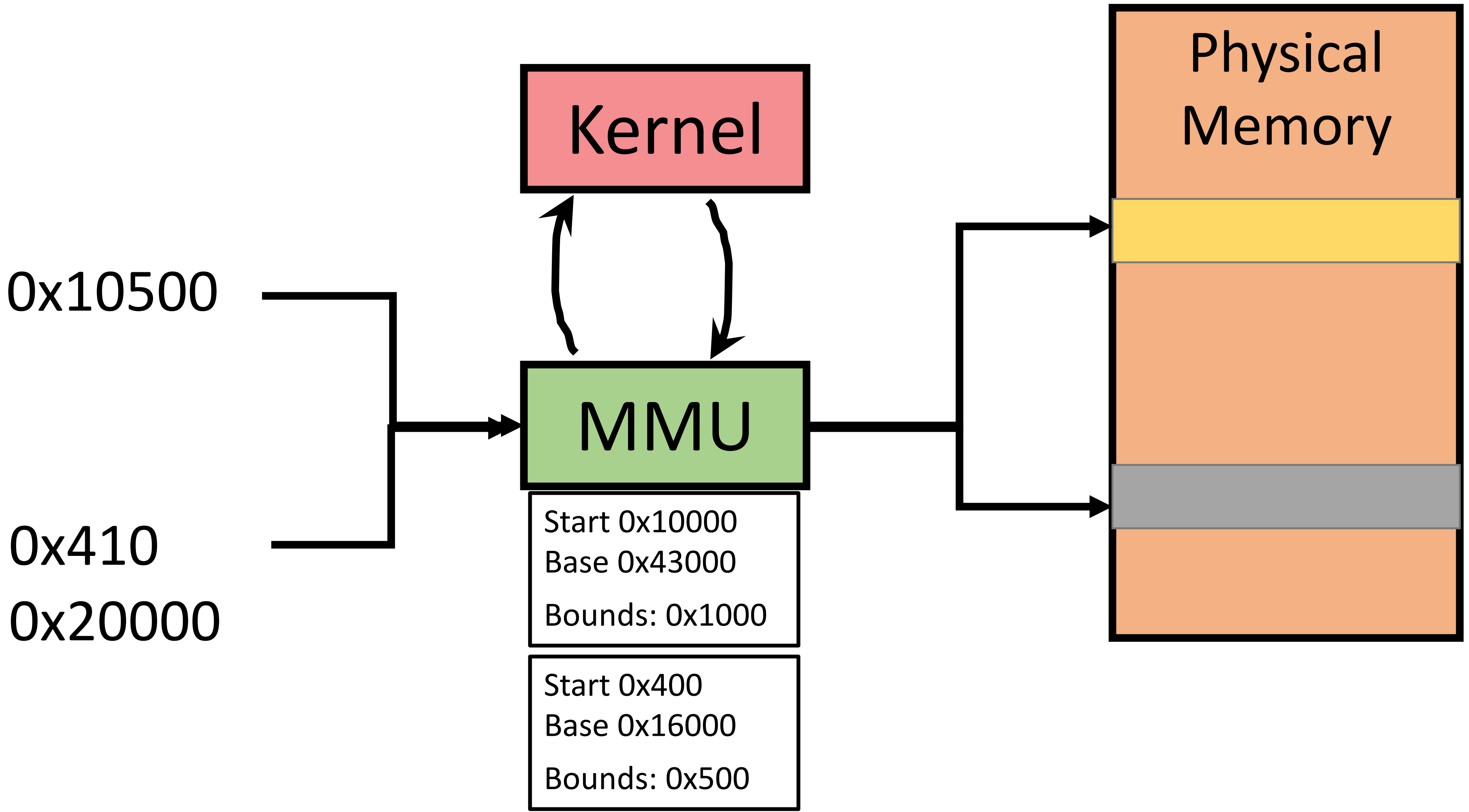
CS 1217

Lecture 15 – Segmentation Wrap, Paging

Recap: Address Translation

- Kernel is in the way for every translation causing huge overheads
    - Need a way to “ask” the kernel once for translating a chunk of addresses
- MMU: Memory Management Unit
    - Hardware mechanism for reducing overheads
    - Ask kernel for information once, cache is for large number of translations
- Base and Bounds
- Segmentation

Segment Based Translation



Start 0x10000
Base 0x43000
Bounds: 0x1000

Start 0x400
Base 0x16000
Bounds: 0x500

Segmentation Pros and Cons

- Pro: still *fairly* simple:
  - Protection (Segment Exists): N comparisons for N segments.
  - Translation: **one** addition. (Once the segment is located)

- Pro: can **organize** and **protect** regions of memory appropriately.

- Pro: less internal fragmentation.

Segmentation Pros and Cons

- Con: still requires **entire** segment be contiguous in memory!

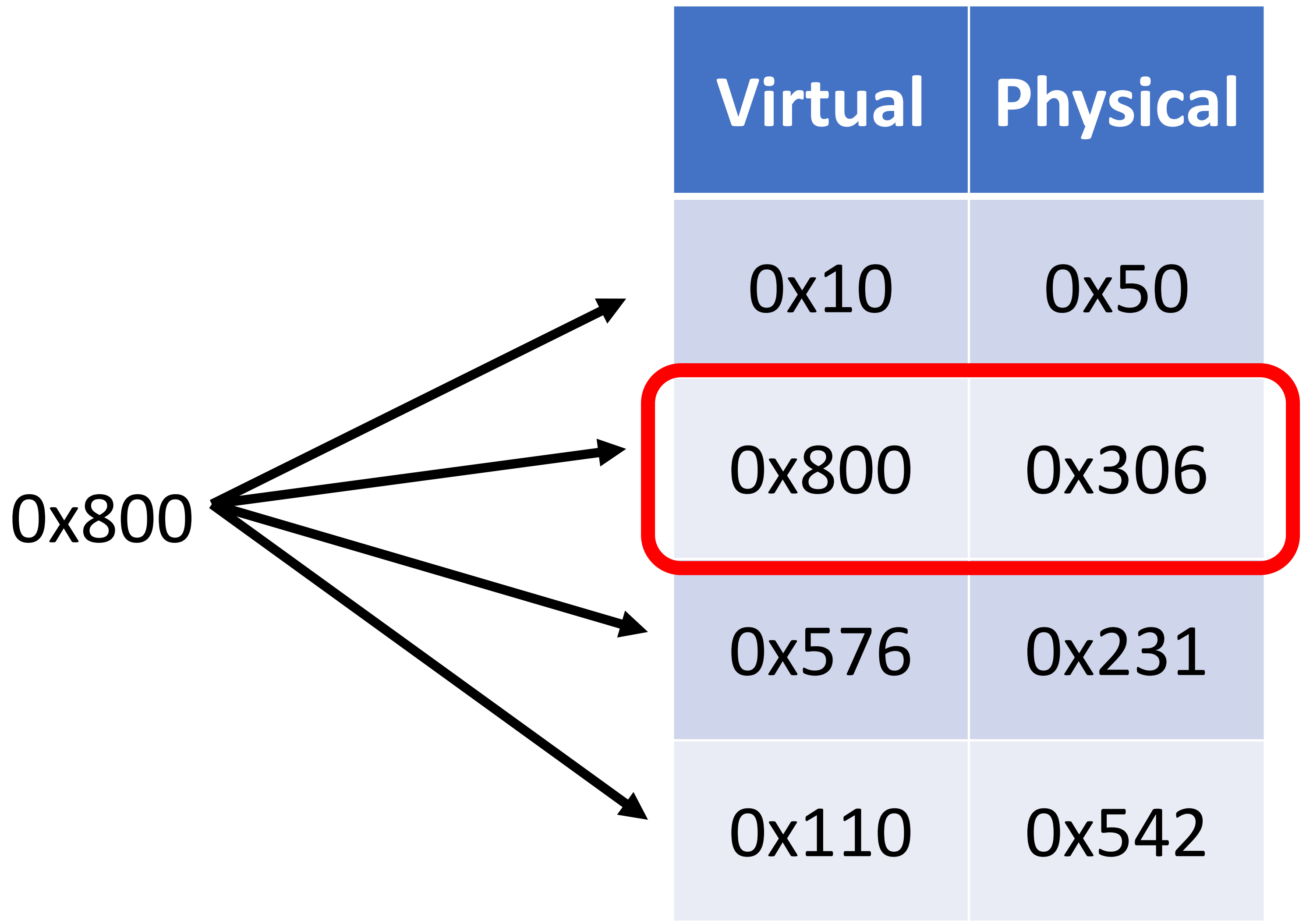
- Con: potential for external fragmentation due to segment contiguity.

Making Translations Faster

- Want to make things faster? Cache it!!

- TLB: **Translation Lookaside Buffer**: Caches virtual to physical address translations in a hardware cache
  - use *content-addressable memory* or CAMs

TLBs



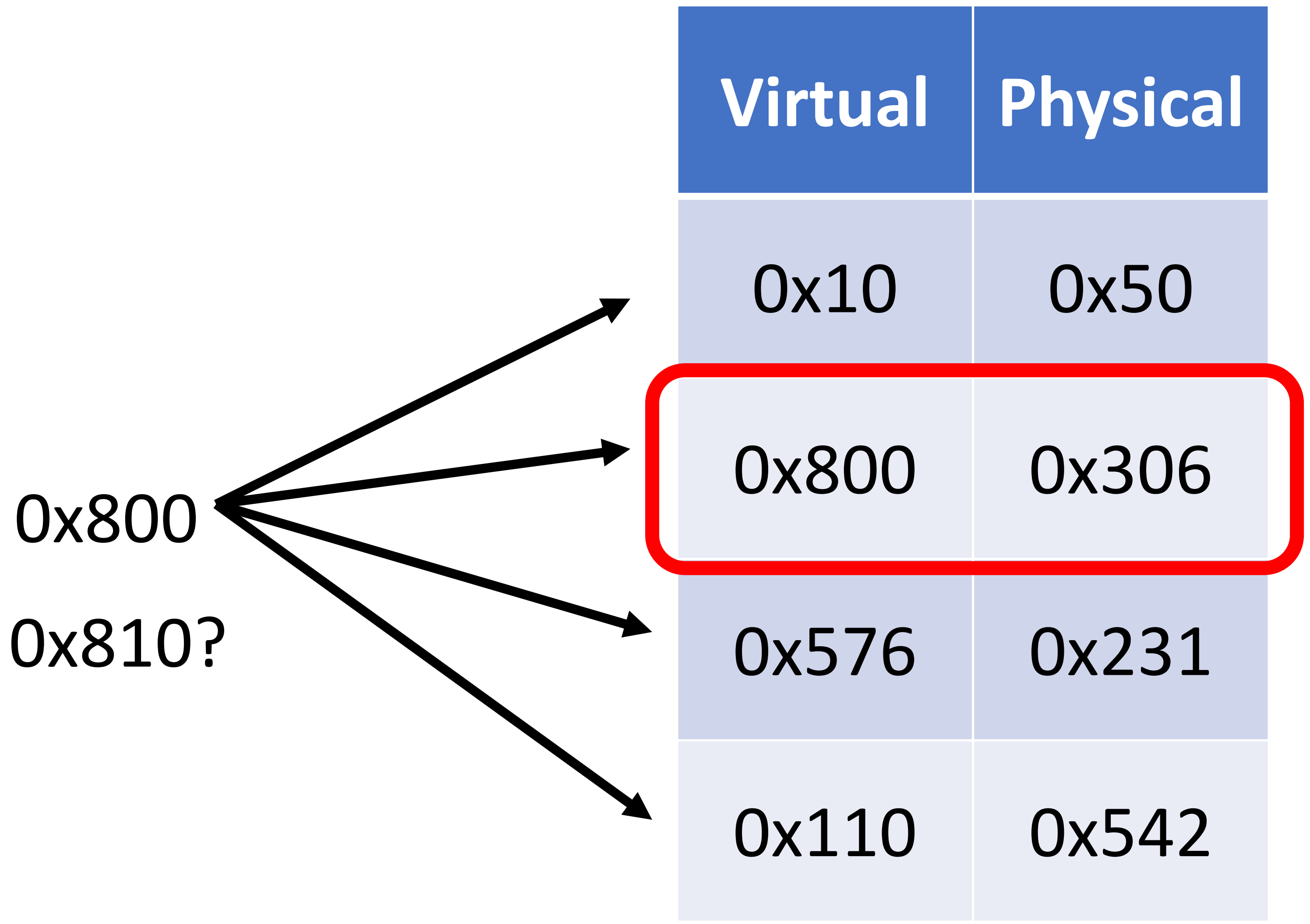
Virtual	Physical
0x10	0x50
0x800	0x306
0x576	0x231
0x110	0x542

Role of the TLB

- Caching *virtual* to *physical address* ***translations*** on the CPU

- use *Content-Addressable Memory* -- **CAM**s
    - CAMs are expensive in area and latency, can not have too many of them

TLBs



Virtual	Physical
0x10	0x50
0x800	0x306
0x576	0x231
0x110	0x542

So far

- Base and Bounds : Too big an allocation, leads to internal fragmentation
- Segmentation: Better, but leads to external fragmentation
- How about mapping individual bytes?
- Ideally: choose a translation granularity that is **small enough** to limit internal fragmentation but **large** enough to allow the TLB to cache entries covering a significant amount of memory

Pages

- OS granularity of memory management
- 4KB typical size, other sizes also used
- Pages: 4KB sized allocations of memory (segments), where the **bound** is fixed

- Basic idea:
  - Divide Virtual **and** Physical Address Spaces into **Page** sized chunks
  - Establish one – to – one mapping between virtual and physical pages

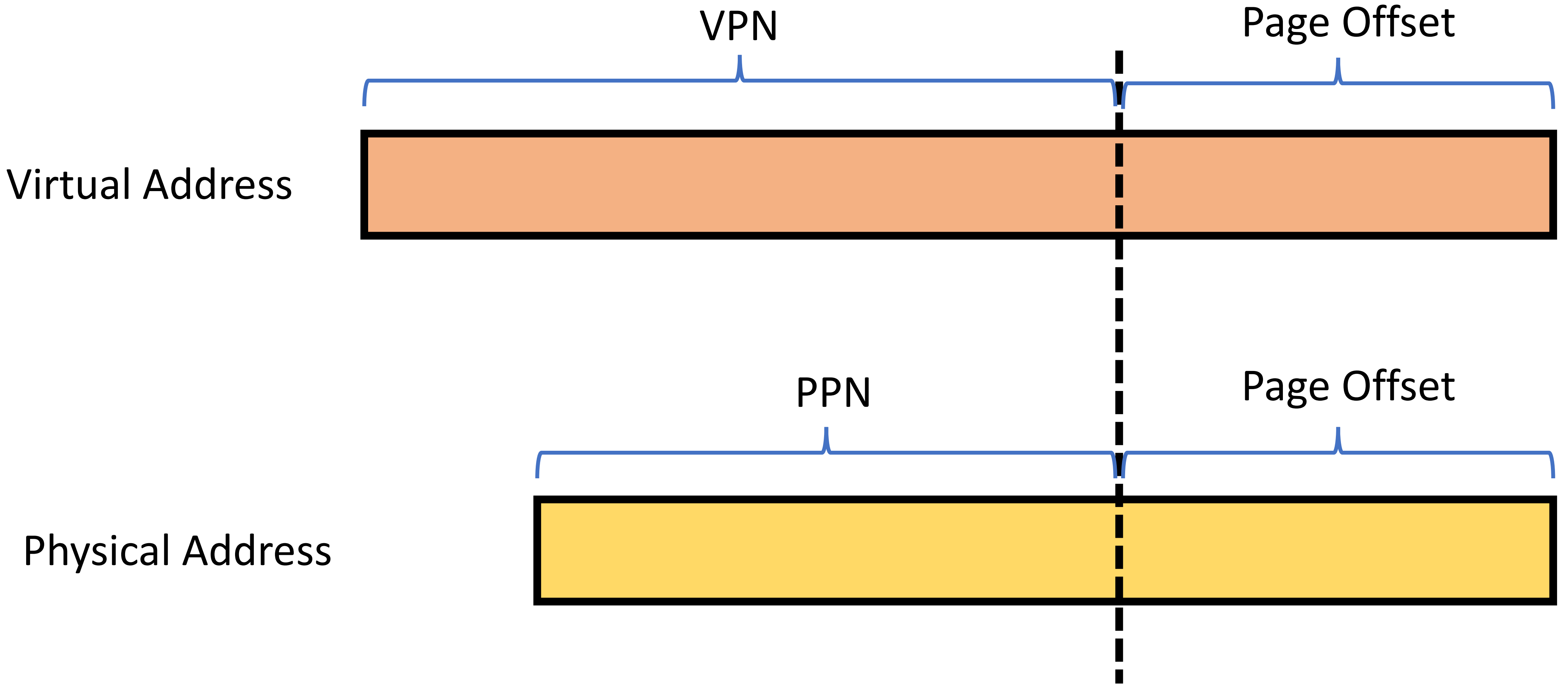
Address Translation using Pages

- Assuming 4K pages, 32-bit virtual memory address space
  - What is the size of the virtual address space?
  - How many bytes can be addressed per page?
  - How many bits are needed for the offset?

- **Check**: Get VPN, Check if a virtual to physical page translation exists
- **Translate:** Physical Address = Physical Page Number + Offset.

Virtual Address

Virtual Address to Physical Address


VPN
Page Offset
Virtual Address
PPN
Page Offset
Physical Address