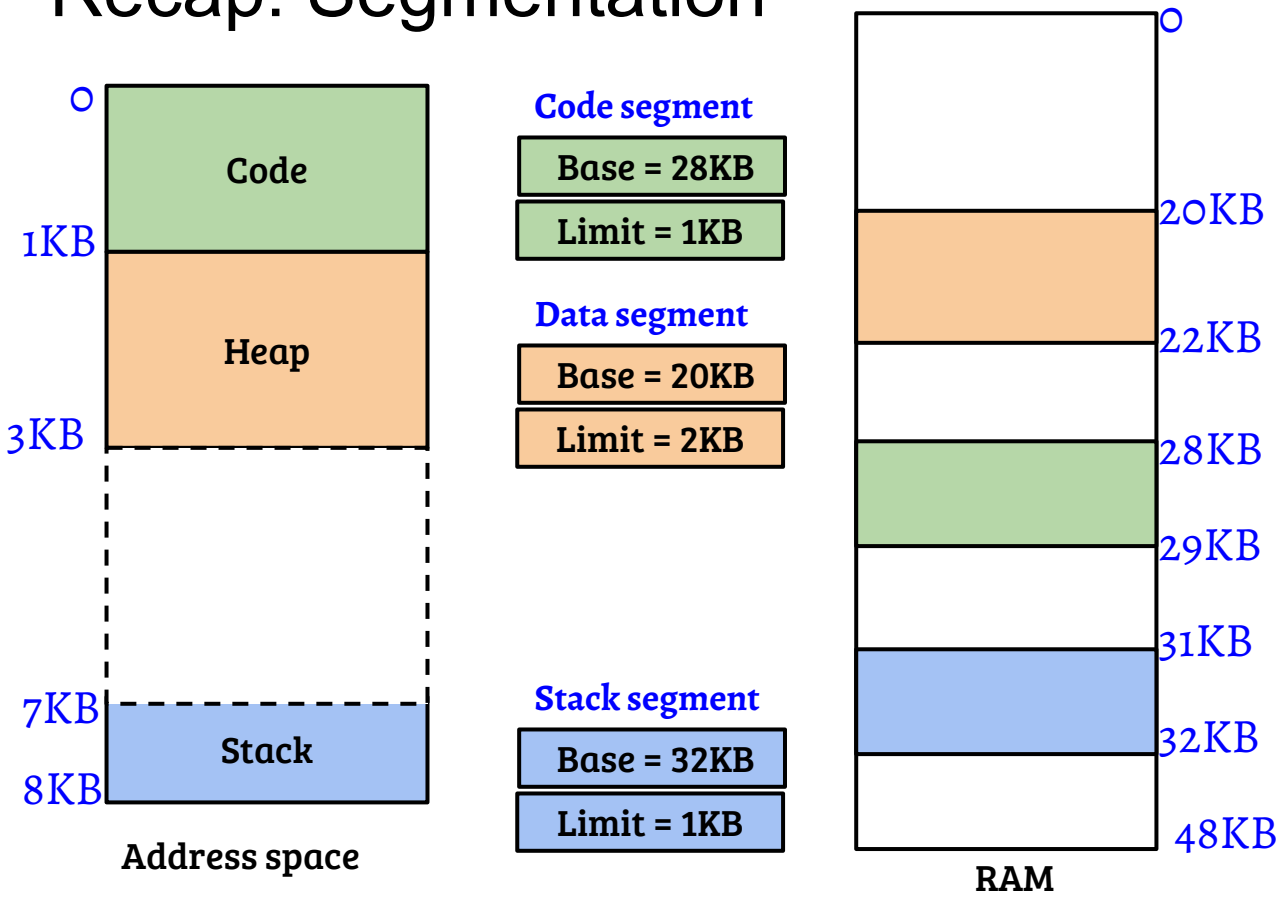
# CS330: Operating Systems

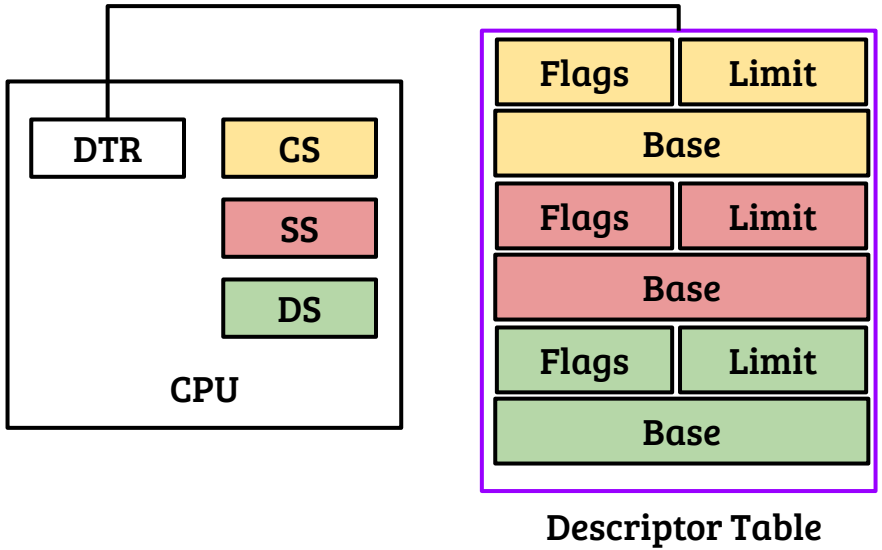
## Virtual memory: Paging

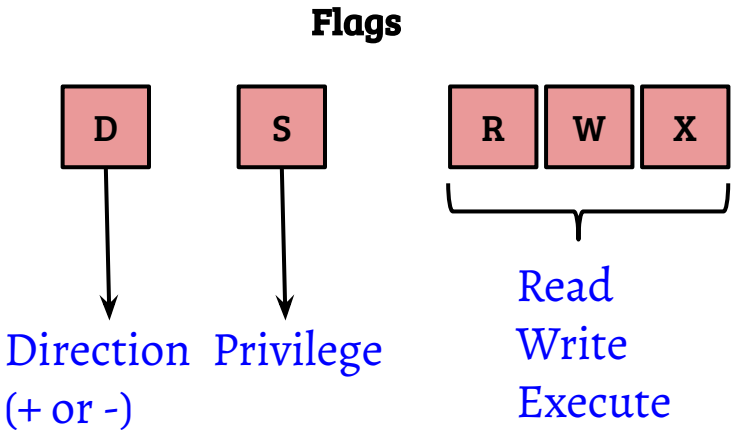
# Recap: Segmentation




- Extension of the scheme for translation ar address space granularity
- Base-limit register pairs per segment

# Recap: Segmentation in reality




- Descriptor table register (DTR) is used to access the descriptor table
- # of descriptors depends on architecture
- Separate descriptors used for user and kernel mode

# Paging

- Paging addresses the following issues with segmentation
  - External fragmentation caused due to variable sized segments
  - No support for discontinuous/sparse mapping

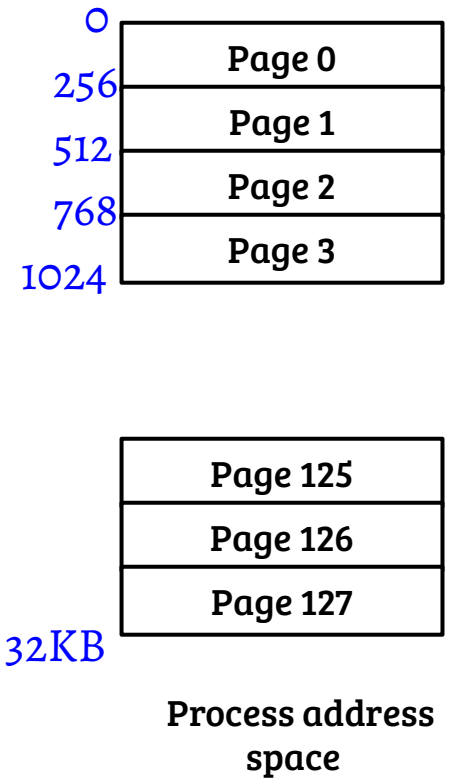
# Paging

- Paging addresses the following issues with segmentation
  - External fragmentation caused due to variable sized segments
  - No support for discontinuous/sparse mapping
- The idea of paging
  - Partition the address space into fixed sized blocks (call it page)
  - Physical memory partitioned in a similar way (call it page frame)

# Paging

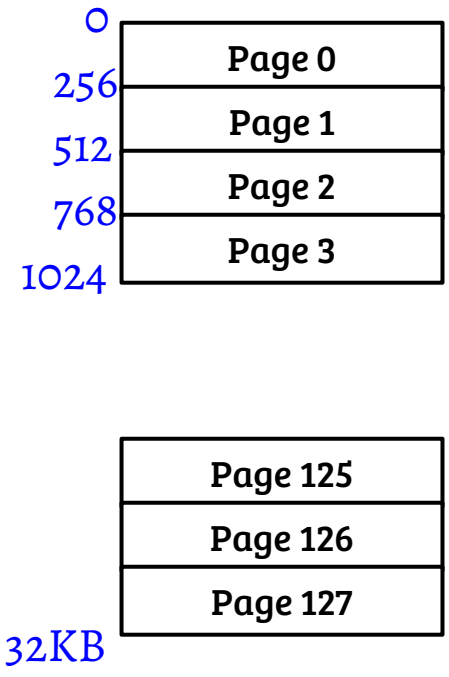
- Paging addresses the following issues with segmentation
  - External fragmentation caused due to variable sized segments
  - No support for discontinuous/sparse mapping
- The idea of paging
  - Partition the address space into fixed sized blocks (call it pages)
  - Physical memory partitioned in a similar way (call it page frames)
  - OS creates a mapping between *page* to *page frame*
  - H/W uses the mapping to translate VA to PA

# Paging example (pages)

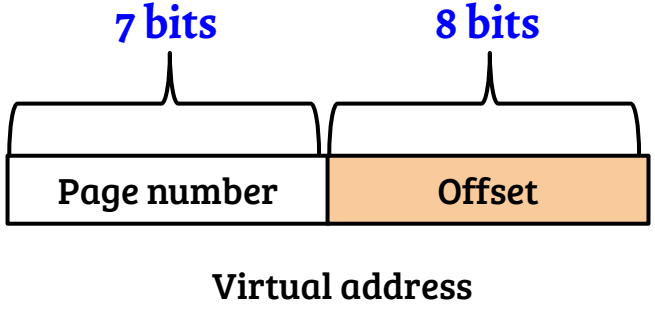



- Virtual address size = 32KB, Page size = 256 bytes
- Address length = 15 bits {0x0 - 0x7FFF}
- # of pages = 128

# Paging example (pages)

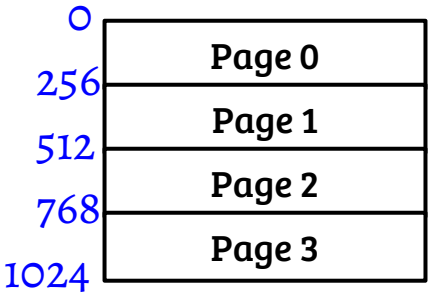



- Virtual address size = 32KB, Page size = 256 bytes
- Address length = 15 bits {0x0 - 0x7FF}
- # of pages = 128

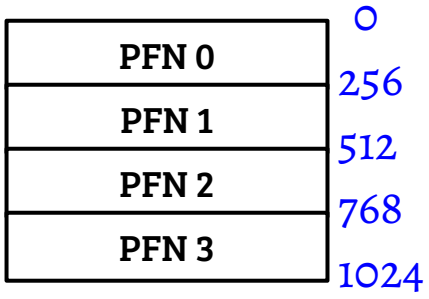


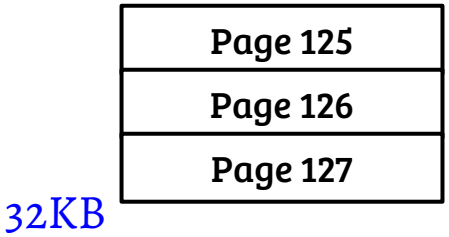

- Example: For Virtual address *0x0510*, Page number = 5, offset = 16

# Paging example (page frames)

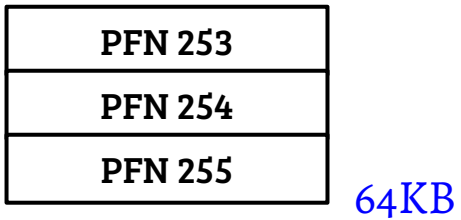



- Physical address size = 64KB
- Address length = 16 bits {0x0 - 0xFFFF}
- # of page frames = 256



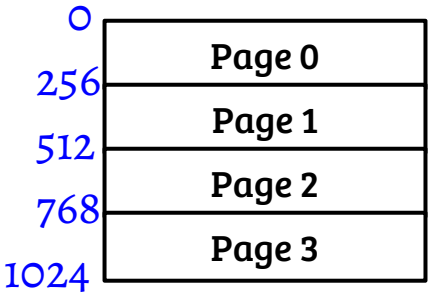




Process address space

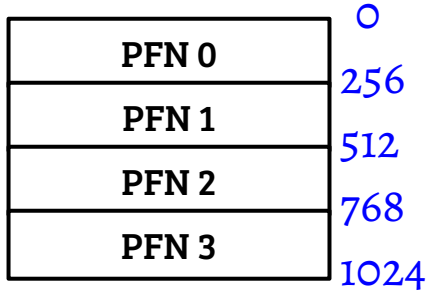


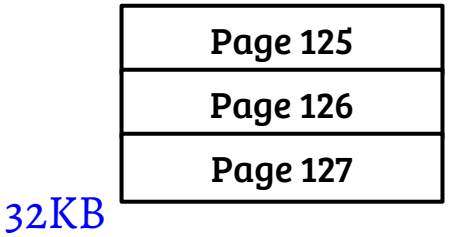

DRAM

# Paging example (page frames)

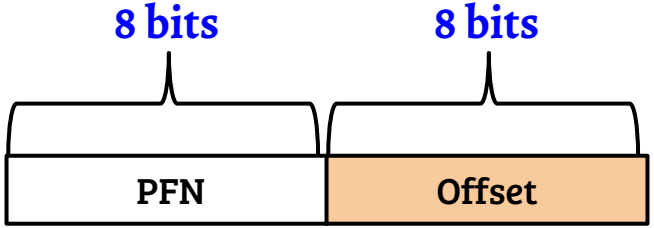



- Physical address size = 64KB
- Address length = 16 bits {0x0 - 0xFFFF}
- # of page frames = 256

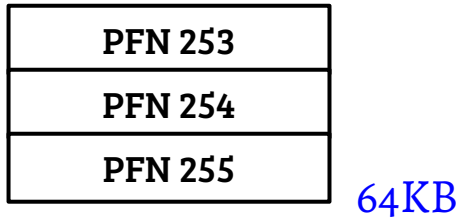



Process address space



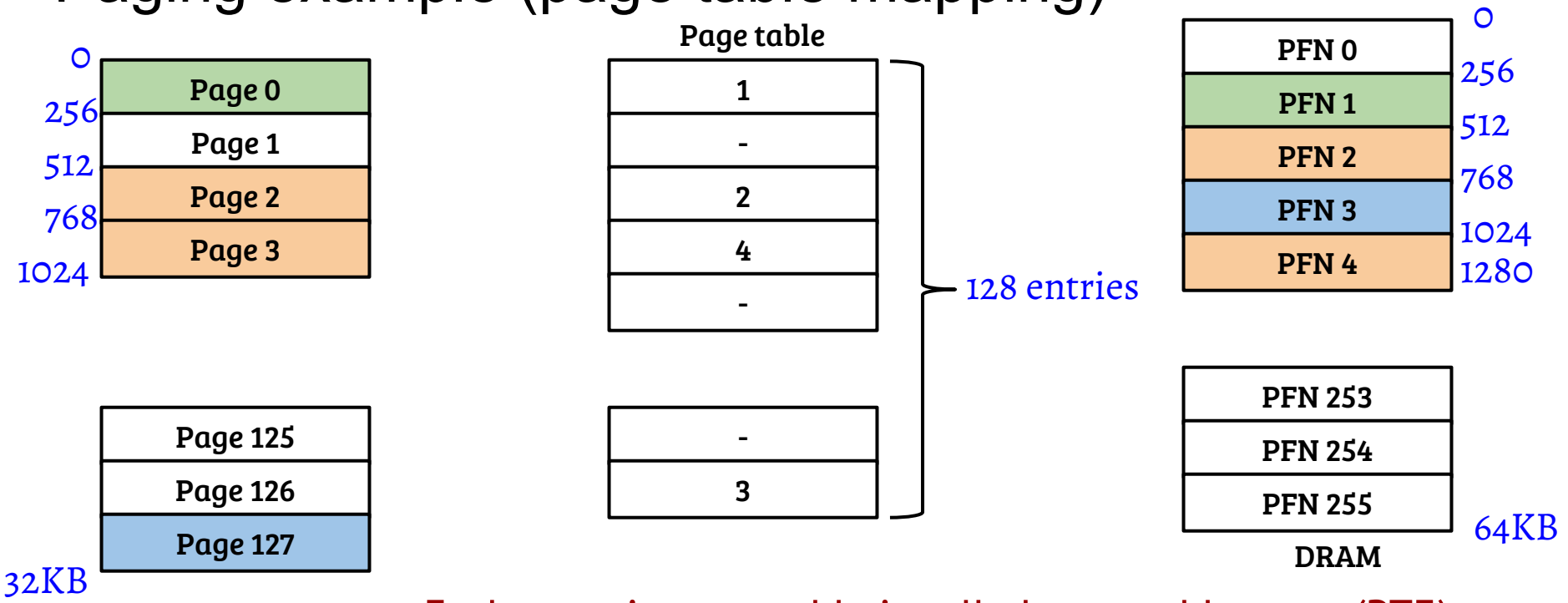

Physical address




DRAM

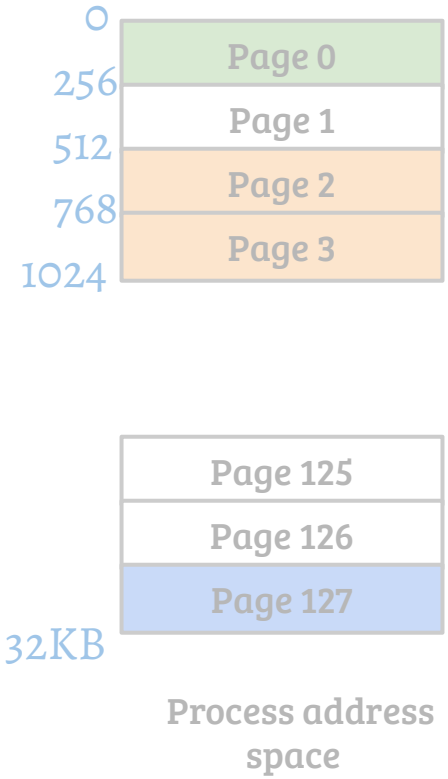
- Example: For physical address *0x1F51*, PFN = 31, offset = 81

# Paging example (page table mapping)

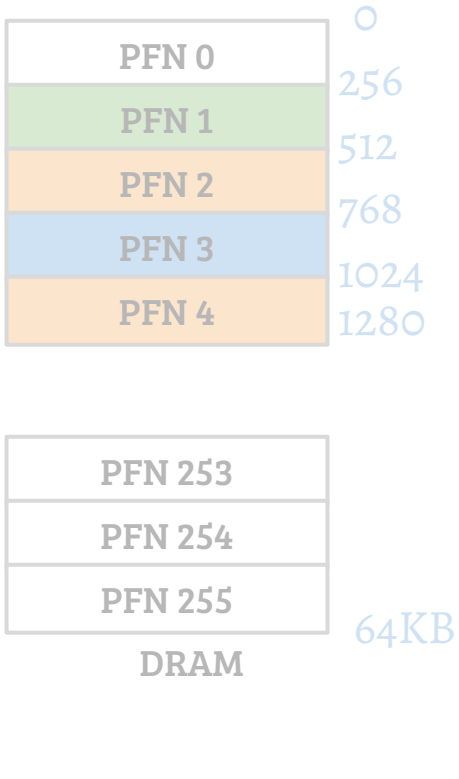



- Each entry in page table is called page table entry (PTE)
- Example mapping: page 0 ⇒ PFN 1, page 2 ⇒ PFN 2 and so on

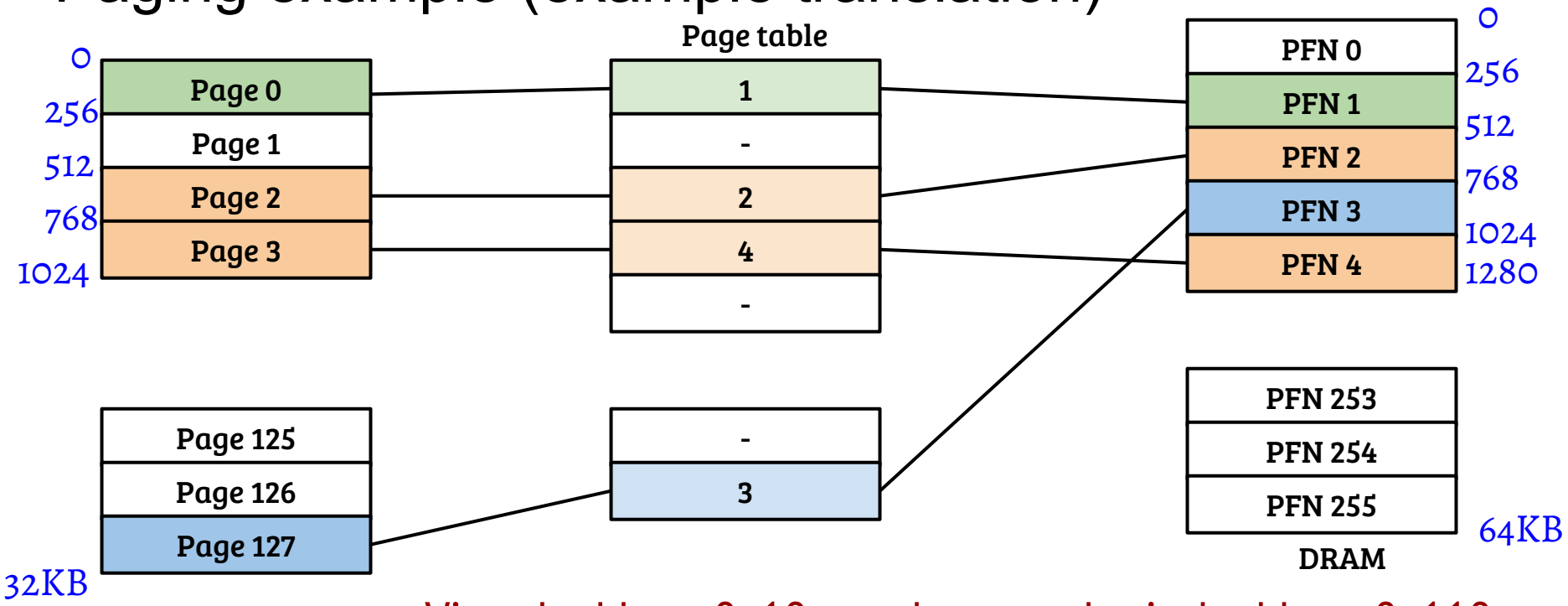
# Paging example (page table walk)




```
PTW (vaddr V, PTable P)
// Input: Virtual address, Page table
// Returns physical address
{
  Entry = P[V >> 8];
  if (Entry.present)
      return (Entry.PFN << 8) + (V & 0xFF);
   Raise PageFault;
}
```

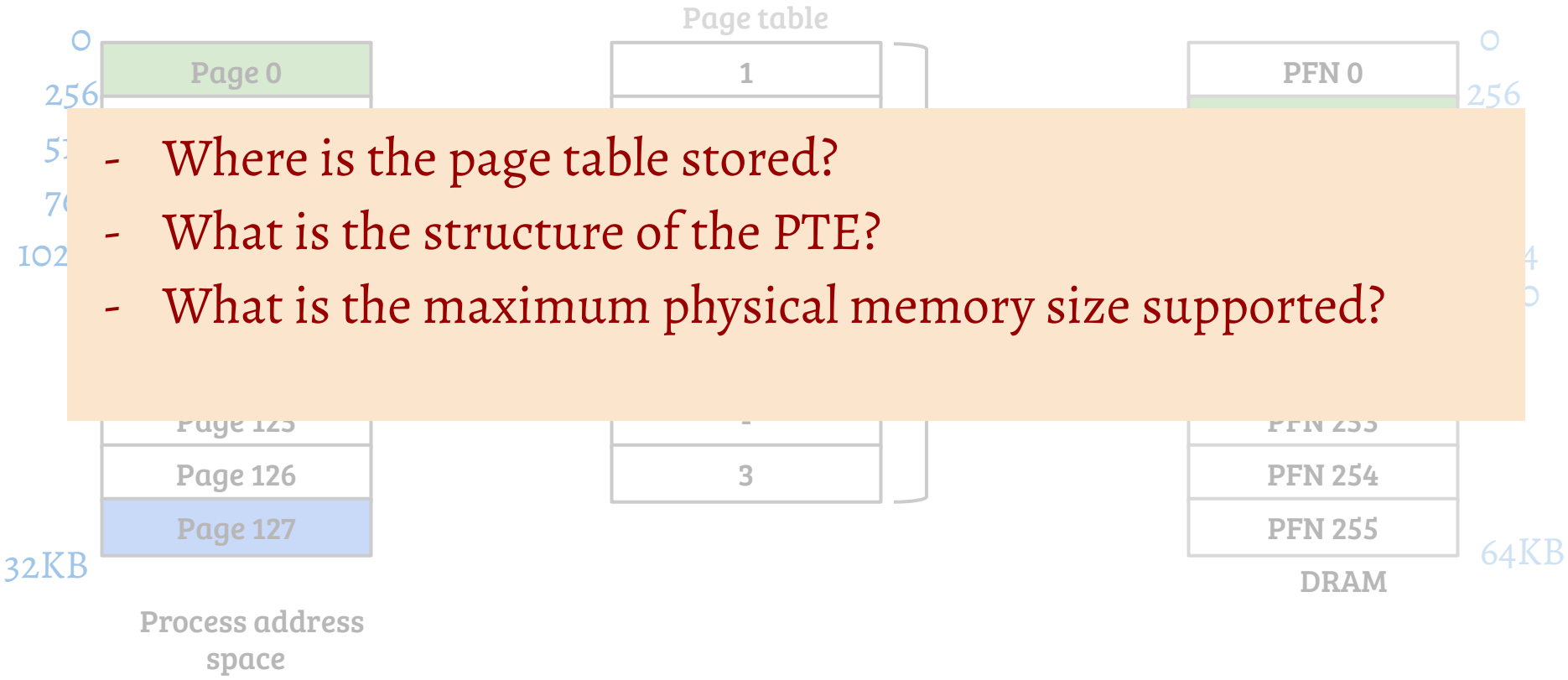


# Paging example (example translation)



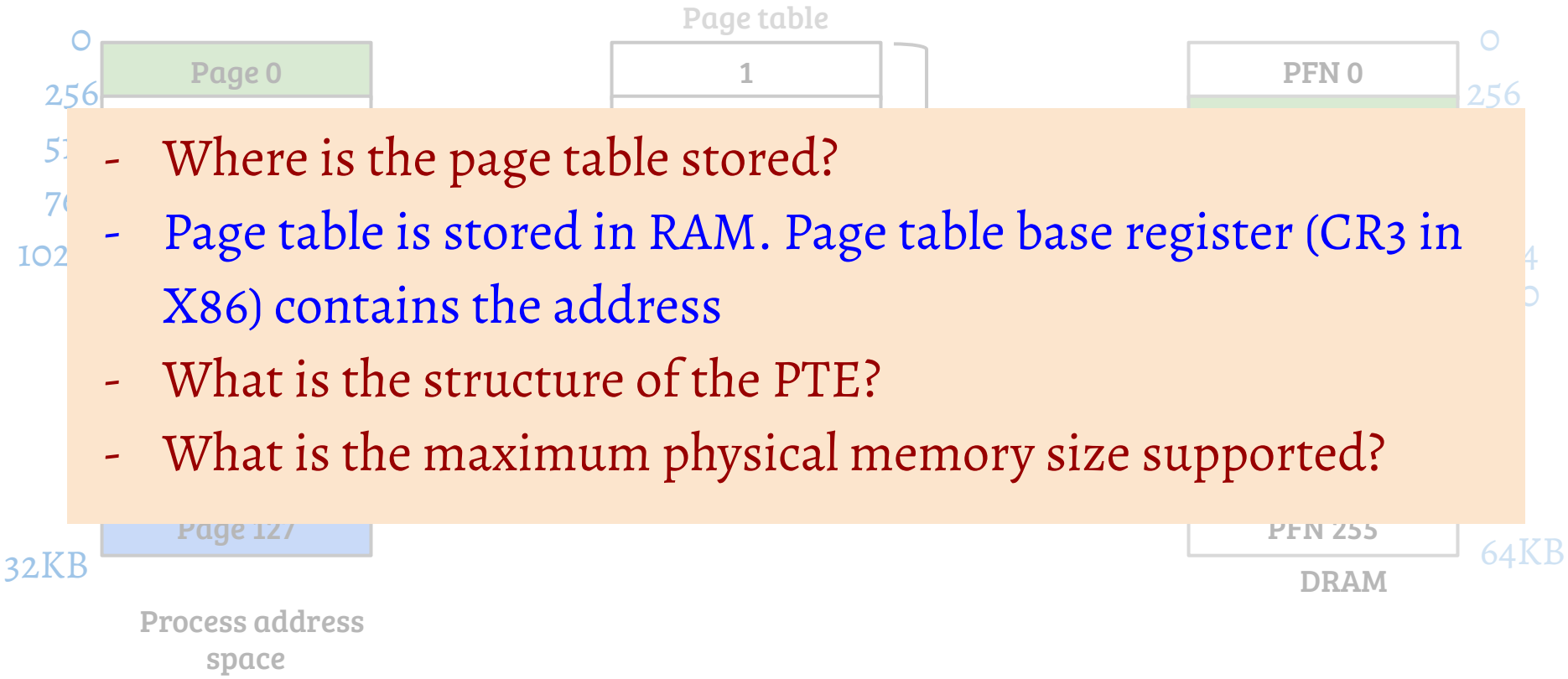

- Virtual address 0x10 translates to physical address 0x110
- Virtual address 0x7FF0 translates to physical address 0x3F0

# Paging example (page table walk)




- Where is the page table stored?
- What is the structure of the PTE?
- What is the maximum physical memory size supported?

# Paging example (page table walk)



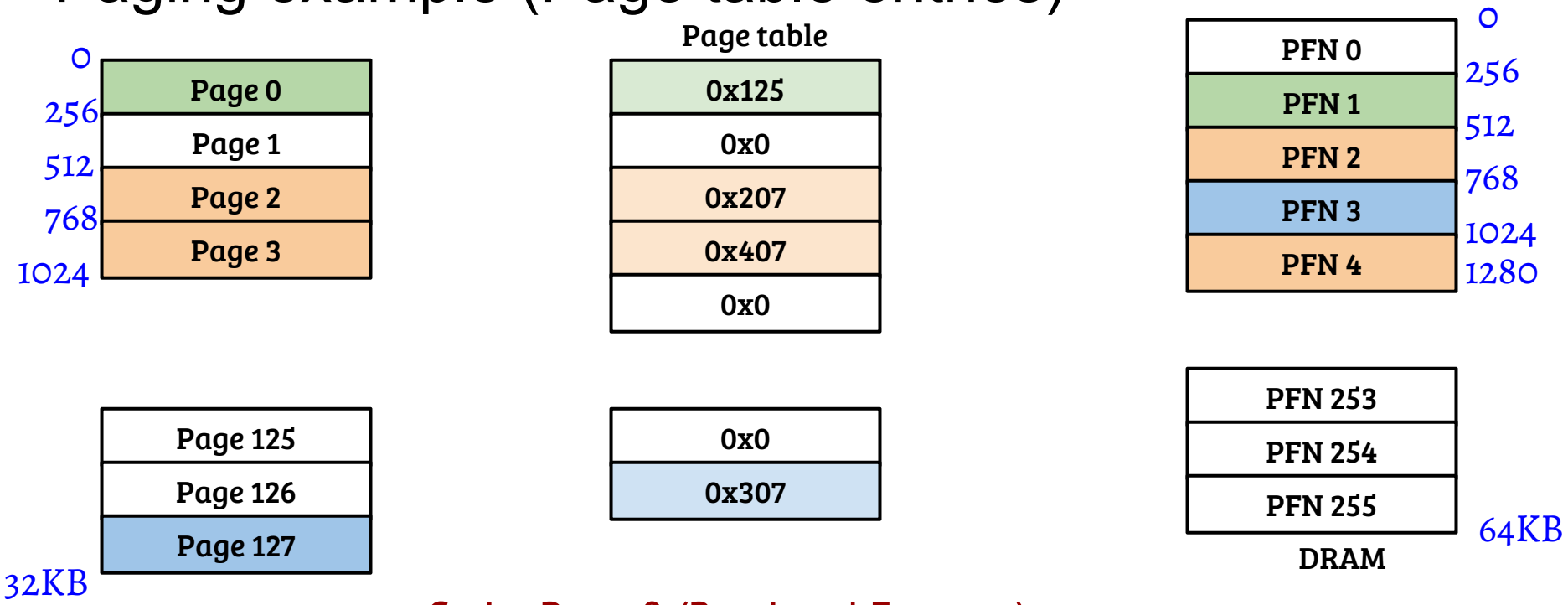

- Where is the page table stored?
- Page table is stored in RAM. Page table base register (CR3 in X86) contains the address
- What is the structure of the PTE?
- What is the maximum physical memory size supported?

# Paging example (structure of an example PTE)


- PFN occupies a significant portion of PTE entry (8 bits in this example)

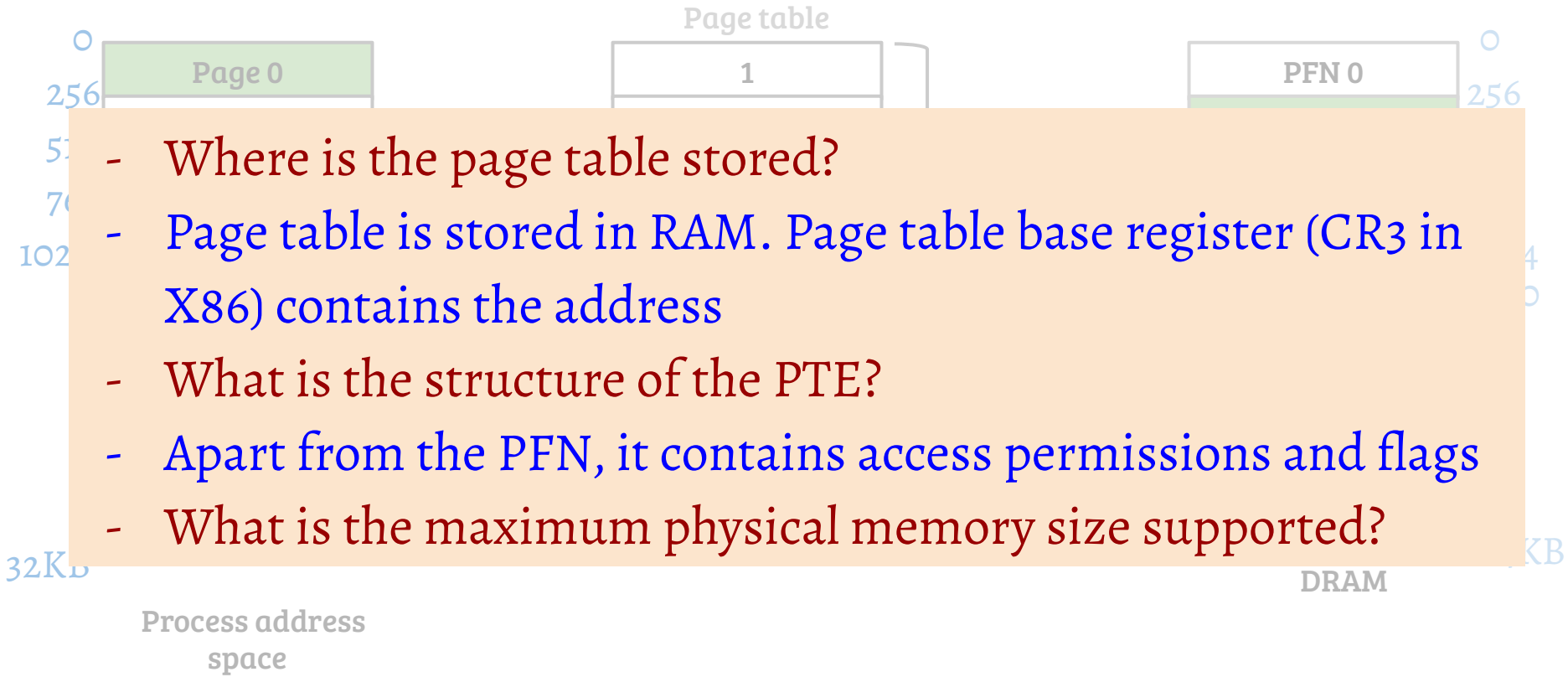
| | |
|---|---|
| P | Present bit, 1 ⇒ entry is valid |
| W | Write bit, 1 ⇒ Write allowed |
| S | Privilege bit, 0 ⇒ only kernel mode access is allowed |
| A | Accessed bit, 1 ⇒ Address accessed (set by H/W during walk) |
| D | Dirty bit, 1 ⇒ Address written (set by H/W during walk) |
| X | Execute bit, 1 ⇒ Instruction fetch allowed for this page |
| | Reserved/unused bits |

# Paging example (Page table entries)




- Code: Page 0 (Read and Execute)
- Data: Page 2 and Page 3 (Read and Write)
- Stack: Page 127 (Read and Write)

# Paging example (page table walk)




- Where is the page table stored?
- Page table is stored in RAM. Page table base register (CR3 in X86) contains the address
- What is the structure of the PTE?
- Apart from the PFN, it contains access permissions and flags
- What is the maximum physical memory size supported?

# Paging example (page table walk)

- Where is the page table stored?
- Page table is stored in RAM. Page table base register (CR3 in X86) contains the address
- What is the structure of the PTE?
- Apart from the PFN, it contains access permissions and flags
- What is the maximum physical memory size supported?
- For this example, 8-bits can be used to specify 256 page frames. Maximum RAM size = 256 * 256 = 64KB

# Paging: one level of page table may not be feasible!

- Consider a 32-bit address space (=4GB)
- What should be the page size for this system?

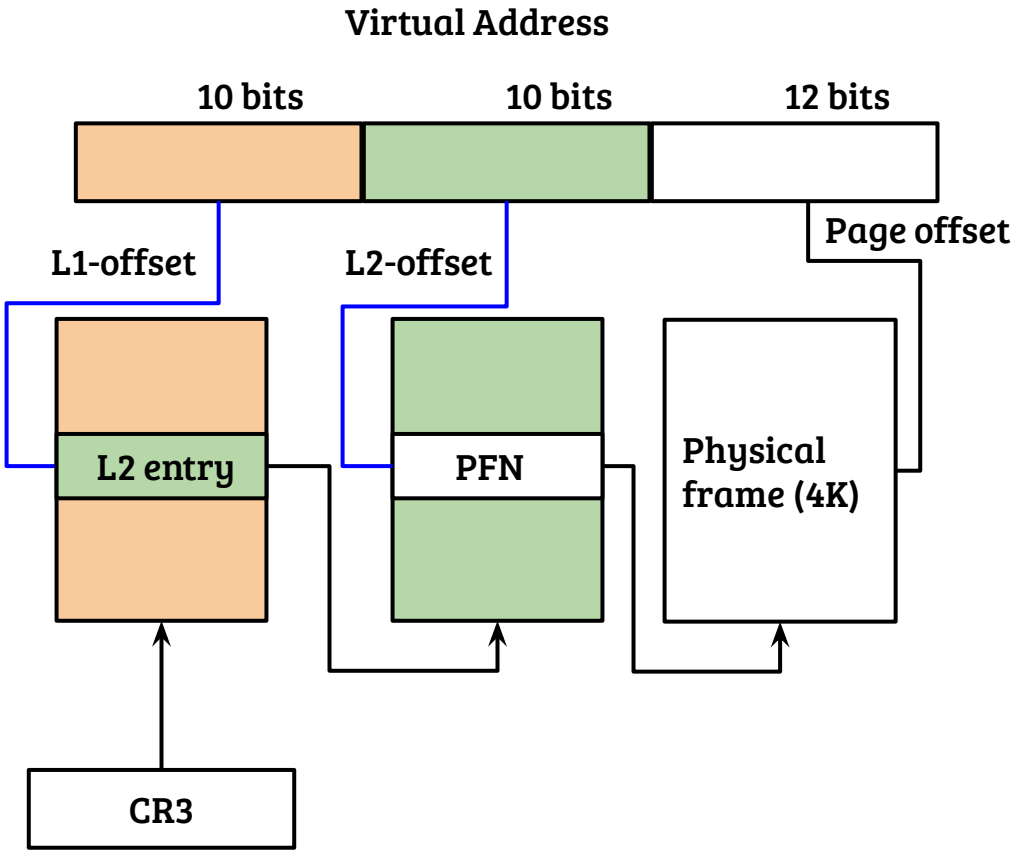
# Paging: one level of page table may not be feasible!

- Consider a 32-bit address space (=4GB)
- What should be the page size for this system?
- Large page size results in *internal fragmentation*
- Assuming page size = 4KB, How many entries are required in a one-level paging system?

# Paging: one level of page table may not be feasible!

- Consider a 32-bit address space (=4GB)
- What should be the page size for this system?
- Large page size results in *internal fragmentation*
- Assuming page size = 4KB, How many entries are required in a one-level paging system? ($2^{20}$ entries)
- Not possible to hold $2^{20}$ entries in a single page
- Therefore, multi-level page tables are used in modern systems

# Two-level page tables (32-bit virtual address)




- Two-level page table
- Level-1 page table contains entries pointing to Level-2 page table structures
- Level-2 entry contains PFN along with flags