Paging

Chapter 18

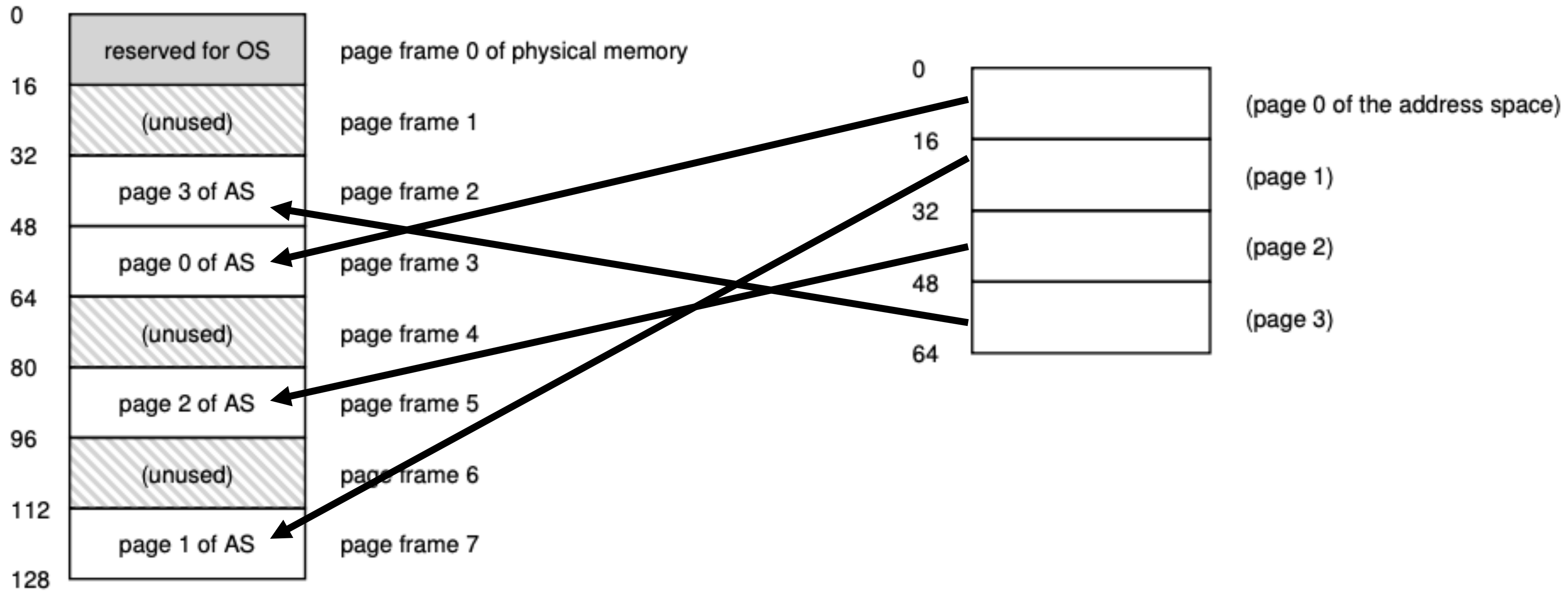
Previously in CS212…

- We solved internal fragmentation (space between stack and heap) with the concept of segments.
  - But we now caused external fragmentation and wasted space in between segments

- Can we somehow use fixed sized blocks in a more efficient way to balance the detriments from internal fragmentation and limit external fragmentation?

Paging

- Keep the idea of segmentation, but split the address space into fixed sized units called **pages** to hold the segments (no more variable sizes allocation units) and do the same for physical memory with **page frames**



Advantages of Paging

- Supports a flexible address space abstraction
  - No special treatment for heap and stack growth

- Simplicity
  - Pages and page frames are the same size
  - Easy to allocate and keep free list

Address Translation

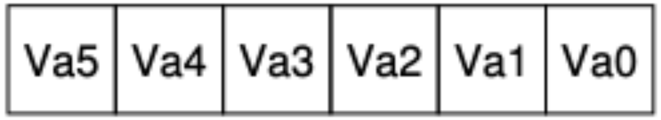
- Need a new *per-process* structure called the **page table**
  - Inverted page tables are an exception

- Stores address translations from pages in address space to page frames in physical memory

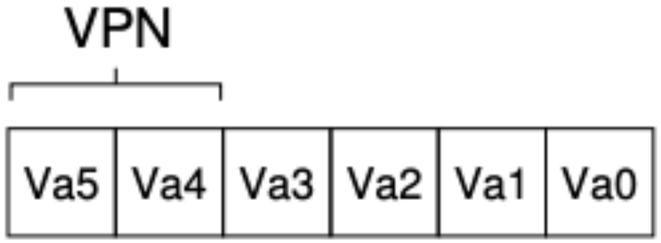
- Need two things for each virtual address:
  - VPN: Virtual Page Number
  - Offset within the page

Example

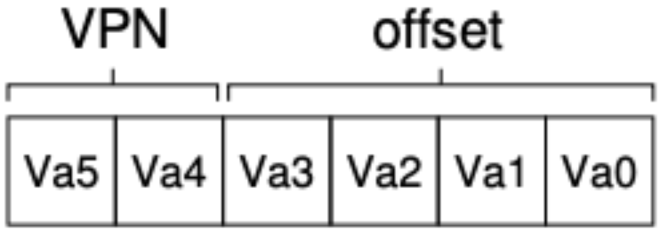
- Suppose we have a 64-byte address space. How many bits do we need to represent that?
  - $2^6 = 64$ which means we have 6 total bits



- If each page is 16 bytes how many pages will we have
  - 64 / 16 = 4 pages
- How many bits will we need to represent 4 pages?
  - $2^2 = 4$ which means we have 2 bits for the VPN
  - 00, 01, 10, 11

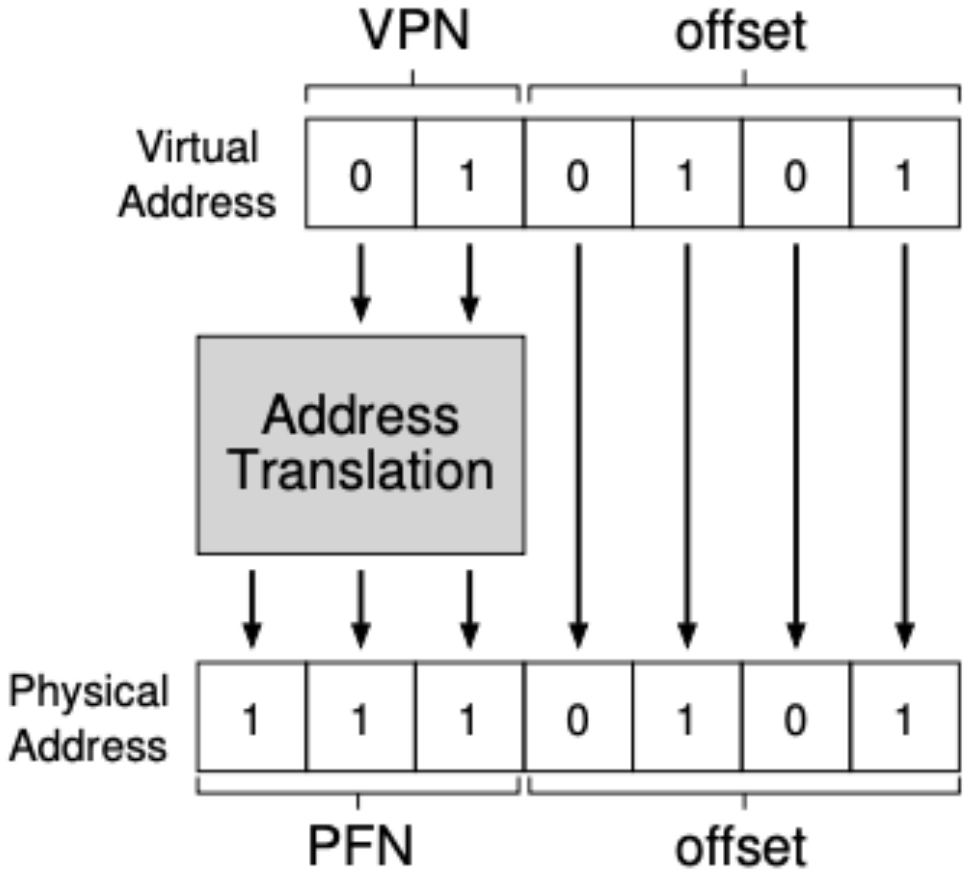


- How many bytes does that leave for the offset?
  - 6 bits – 2-bit VPN = 4-bit offset



Virtual to Physical Address

- Page table provides the address translation
- Offset remains the same
- Example virtual address “21”
  - First page, 5th byte offset
- Page table finds physical location of the first page for the physical address



Storing Page Tables

- Can get very big

- 32-bit address space with 4KB (4096 bytes) pages
  - Addressing 4K requires 12-bits ($2^{12} = 4096$)
  - Leaves 20-bits for VPN (~ 1 Million pages)
    - Fun fact: At 4KB per page that is 4GB, the approximate limit of addressable RAM on a 32-bit OS
  - If each page table entry is 4 bytes that uses 4 MB of ram
    - With 100 processes, that’s 400 MB!

- Can’t store them on the CPU (MMU) so we need the page tables in memory somewhere

Page Table Anatomy

- In the simplest representation, a linear page table is an array. The OS indexes page table by the virtual page number to find the **page table entry (PTE)** to find the **physical frame number (PFN)**.

- A PTE can contain bits for:
  - Valid (is the translation valid) – unused space is also marked invalid
    - Important for sparse address space (we don't need to allocate frames for those pages
  - Protection – permissions to the page (Read, write, execute)
  - Present – In or out of physical memory (can be moved to disk)
  - Dirty – has page been modified
  - Reference – has the page been accessed

Paging Disadvantage

- Slow!

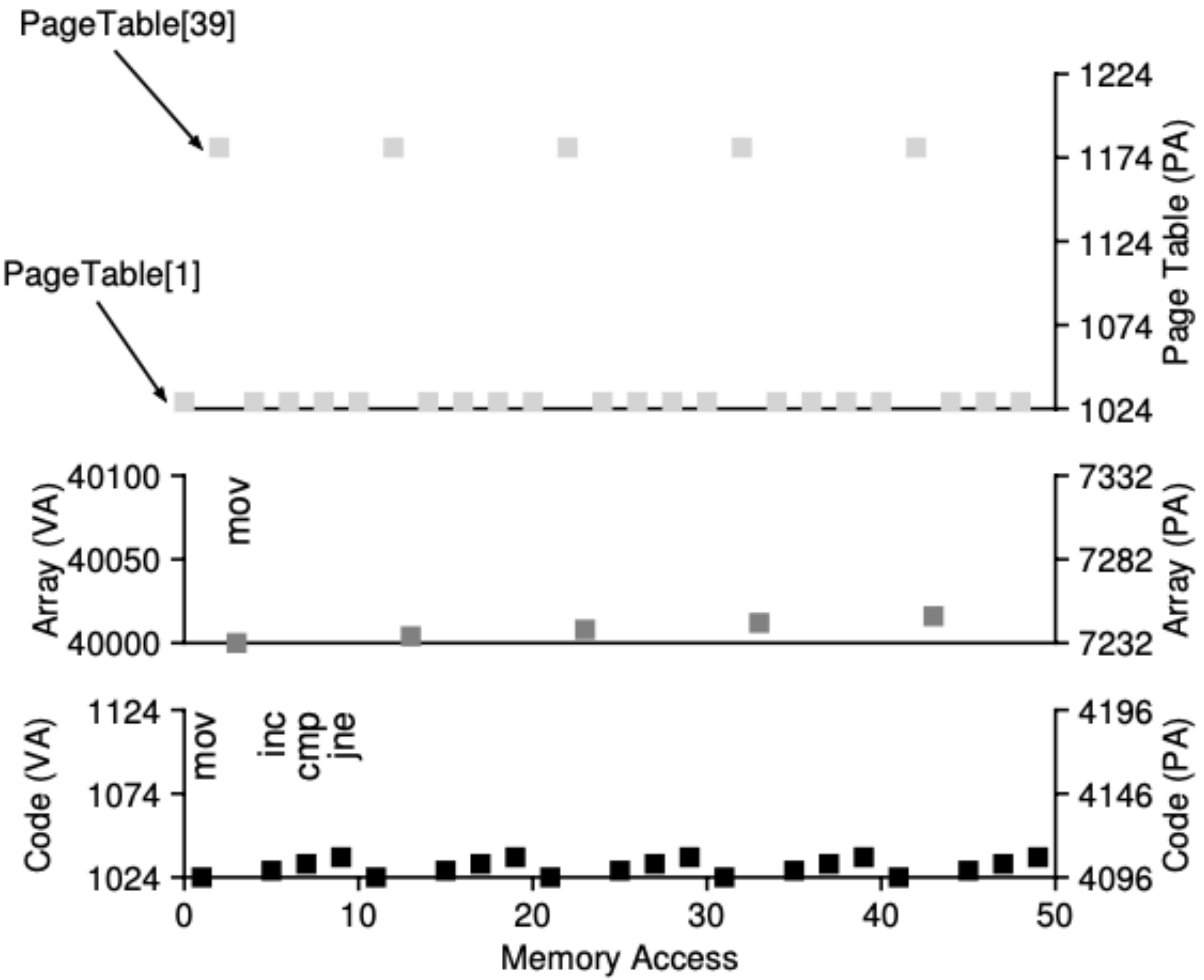
- No longer simply applying base and bound approach to for calculation

- Need to reference the page table for each code line and data in memory
  - Faster than disc, but still way slower than register computation

Disadvantage in practice

```
int array[1000];
...
for (i = 0; i < 1000; i++)
    array[i] = 0;
```

```
1024 movl $0x0,(%edi,%eax,4)
1028 incl %eax
1032 cmpl $0x03e8,%eax
1036 jne  0x1024
```



10 memory access requests per loop!

Next Time…

- This is the end of the content for the exam
- Weds we will work on the Unix Shell
- Attendance on Wednesday is **NOT** optional
  - Leaving early is **not** permitted either (unless there is a valid reason)
- Work on the assignments, if you finish early, look over the study guide and prepare questions for Friday