CS 1217

Lecture 16 – Paging, Page Table Entries

Pages : Recap

- 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

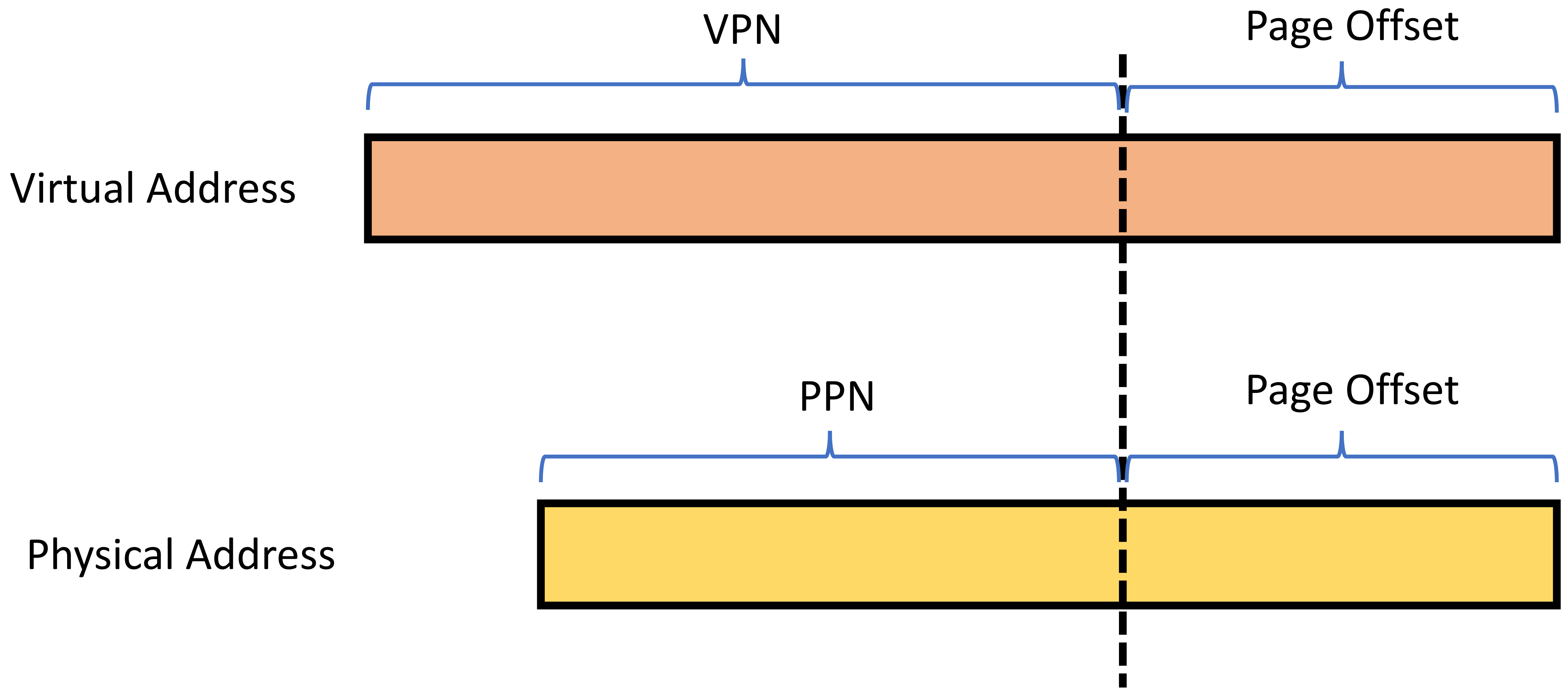
Address Translation using Pages

- 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

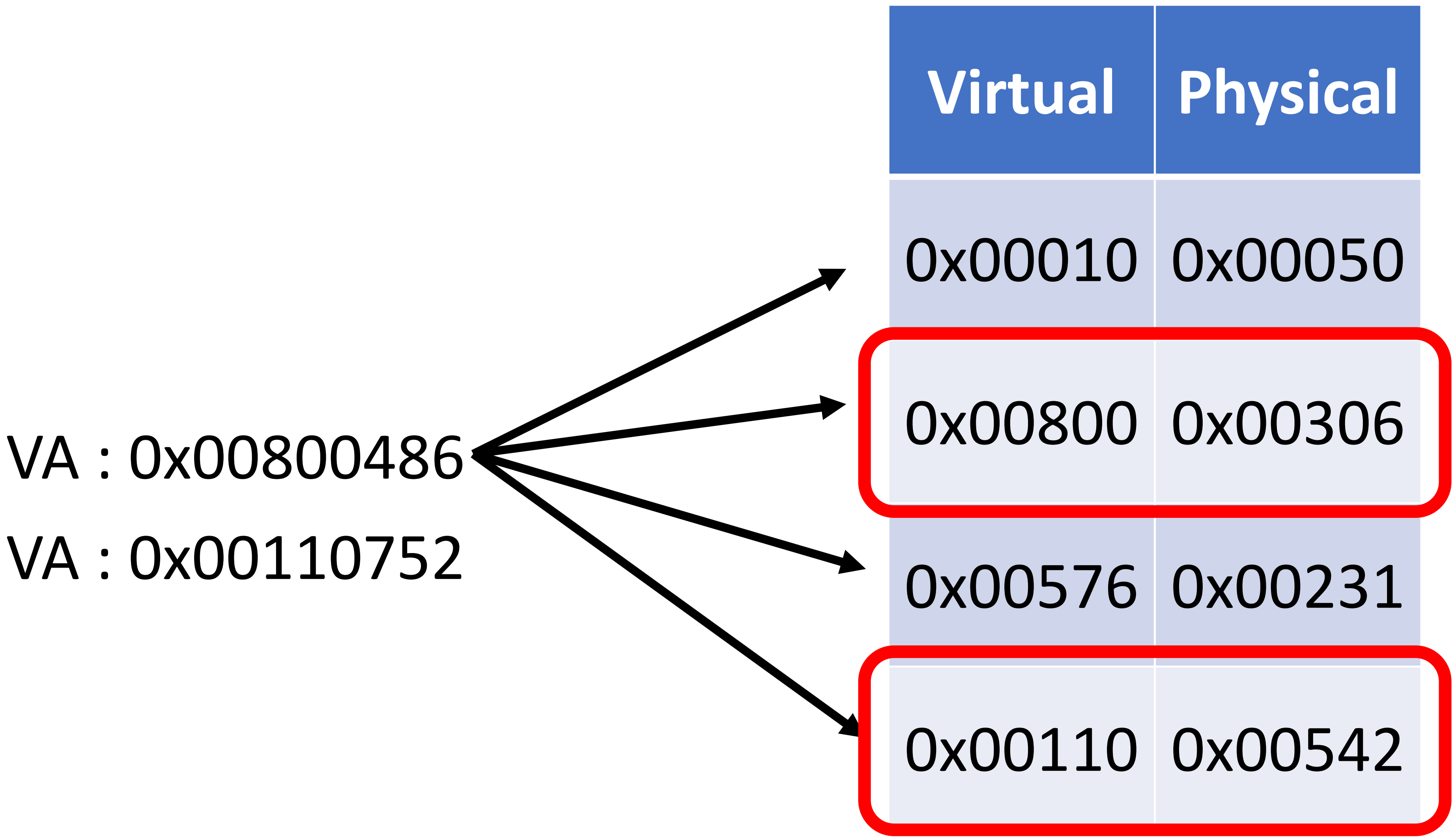
Virtual Address to Physical Address



Address Translation using Pages

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

Role of TLBs now



VA : 0x00800486

VA : 0x00110752

Virtual	Physical
0x00010	0x00050
0x00800	0x00306
0x00576	0x00231
0x00110	0x00542

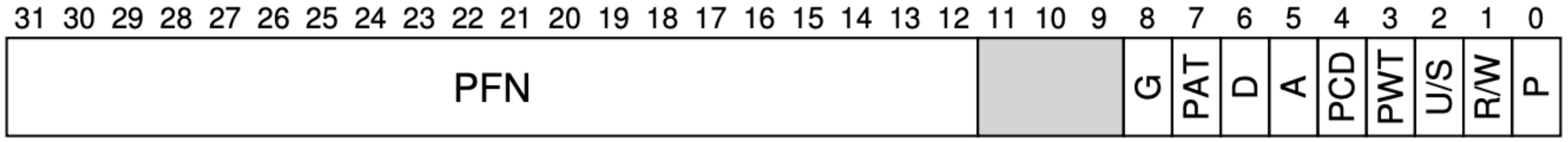
Pros and Cons

- Access permissions?
  - Can now be associated on a per-page basis
- External fragmentation?
  - None: All allocations are multiples of Page Size (e.g. 4KB)
- Internal fragmentation?
  - Maybe a little, but much reduced as compared to segmentation
- Con: requires per-page hardware translation, need h/w help
- Con: requires per-page operating system state (page tables)

Page based Address Translations

- TLBs **cache** virtual to physical translations
- Where are the translations really located?
    - i.e., the ones that are **not cached** in the TLB

- **Page Table Entries (PTEs)**
    - a single-entry storing information about a single virtual page used by a single process
    - Information stored?



Page Table Entries

- Page Table Entry (PTE) = A single entry, storing information about a virtual page

- Can contain a lot of information
  - Location – which physical page does this virtual page map to
  - Permissions – Read/Write/Execute
  - Present – whether this page is present in memory
  - Dirty – has this page been written to recently?

The Brave New (World) Translation Path

- *Process:* Store to address 0x10000
- *MMU:* I don't know which physical address 0x10000 maps to
- **Exception!**
- *Kernel*: Need to look up PTE to which 0x10000 belongs

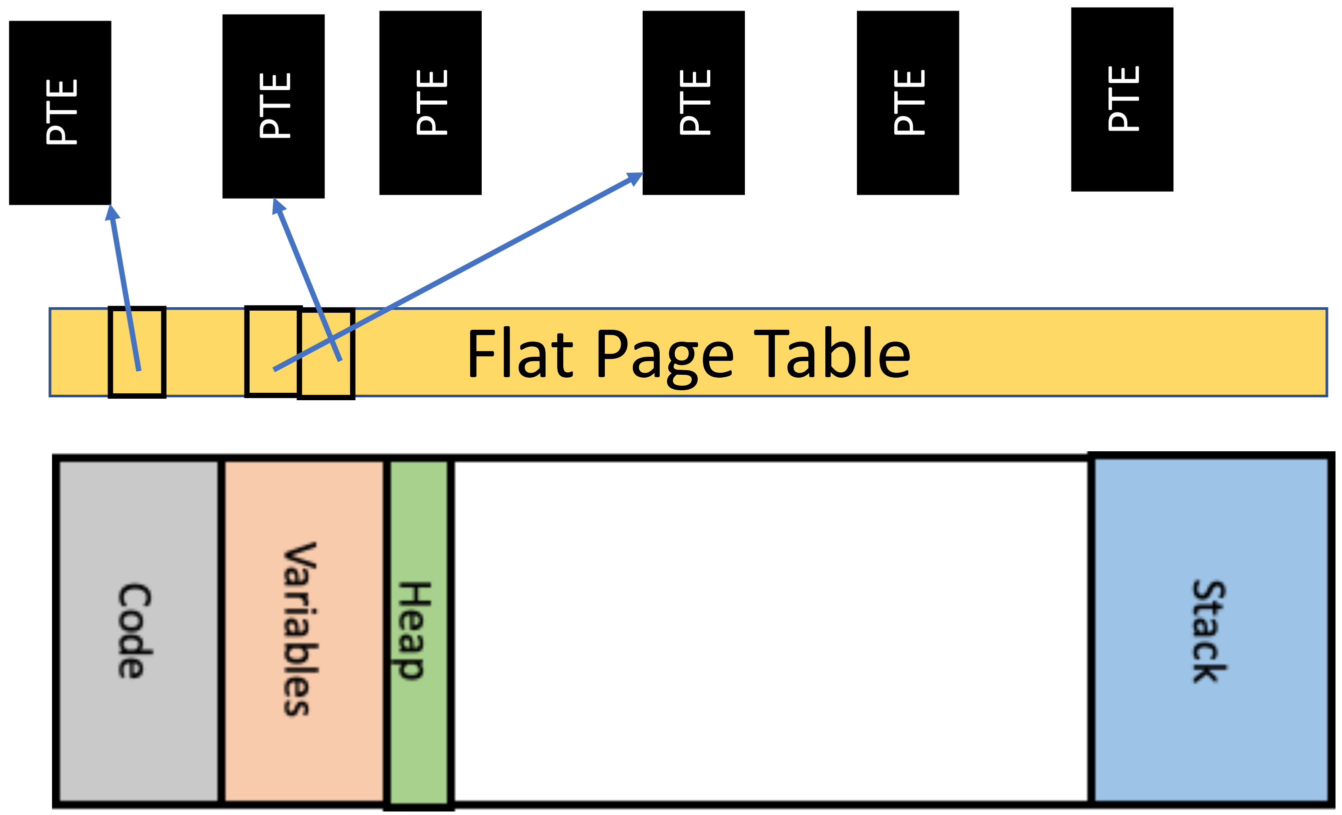
- What should be the properties of this lookup?
    - Fast (Time efficiency)
    - Compact (Space efficiency)

Page Tables

- In essence, a data structure, which
- Maps : a **virtual page number** to a **page table entry**

- Each process has its own page table
    - Why?

Page Table Organization



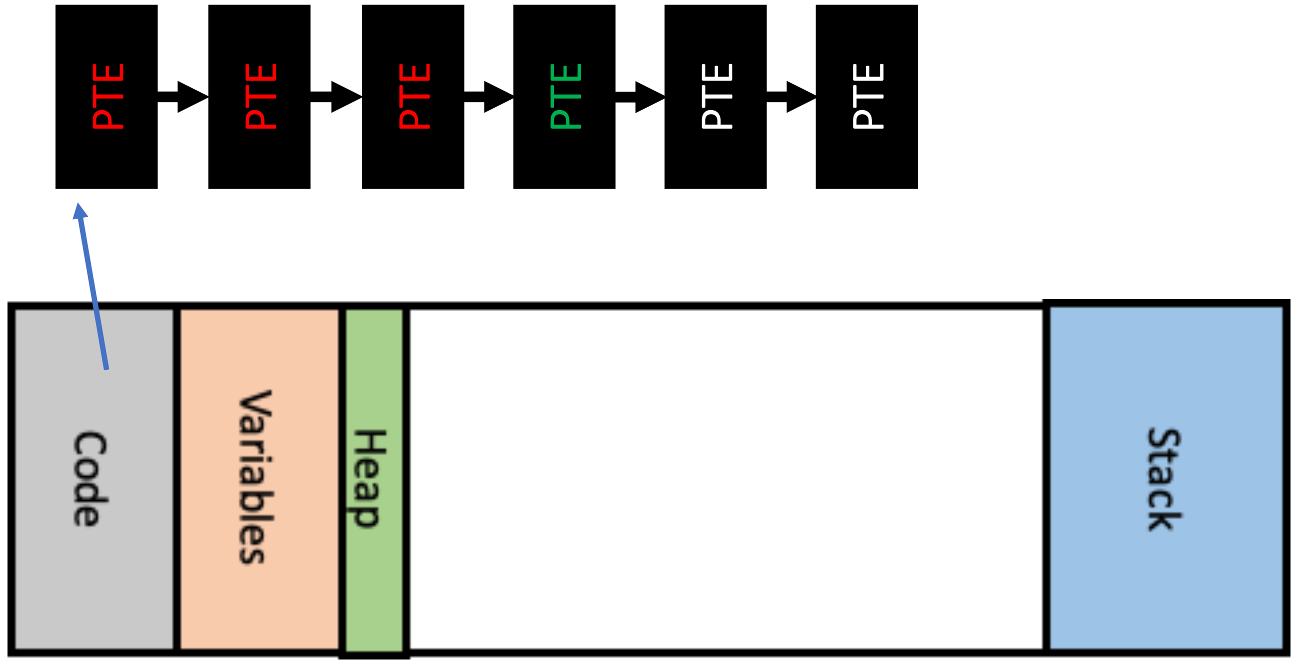
Flat Page Tables

- Time Complexity
  - O(1)
  - Fast!

- Space Complexity
  - 4 MB per process
  - Gets expensive really fast with number of processes

Linked List Page Table

- Maintain LL of **valid** PTEs, to be searched on each lookup



Linked List Page Table

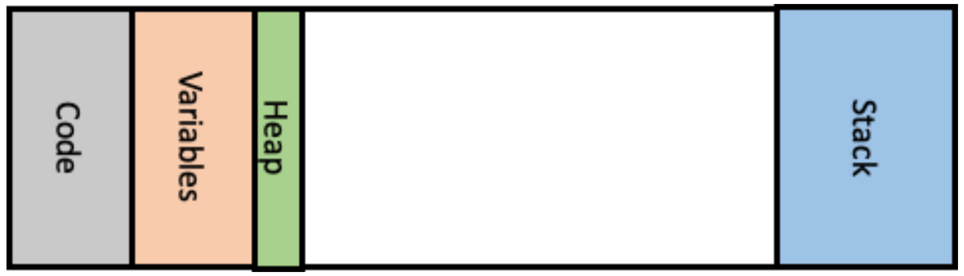
- Time Complexity
    - O(N) for N allocated virtual pages

- Space Complexity
    - 4B * N per process

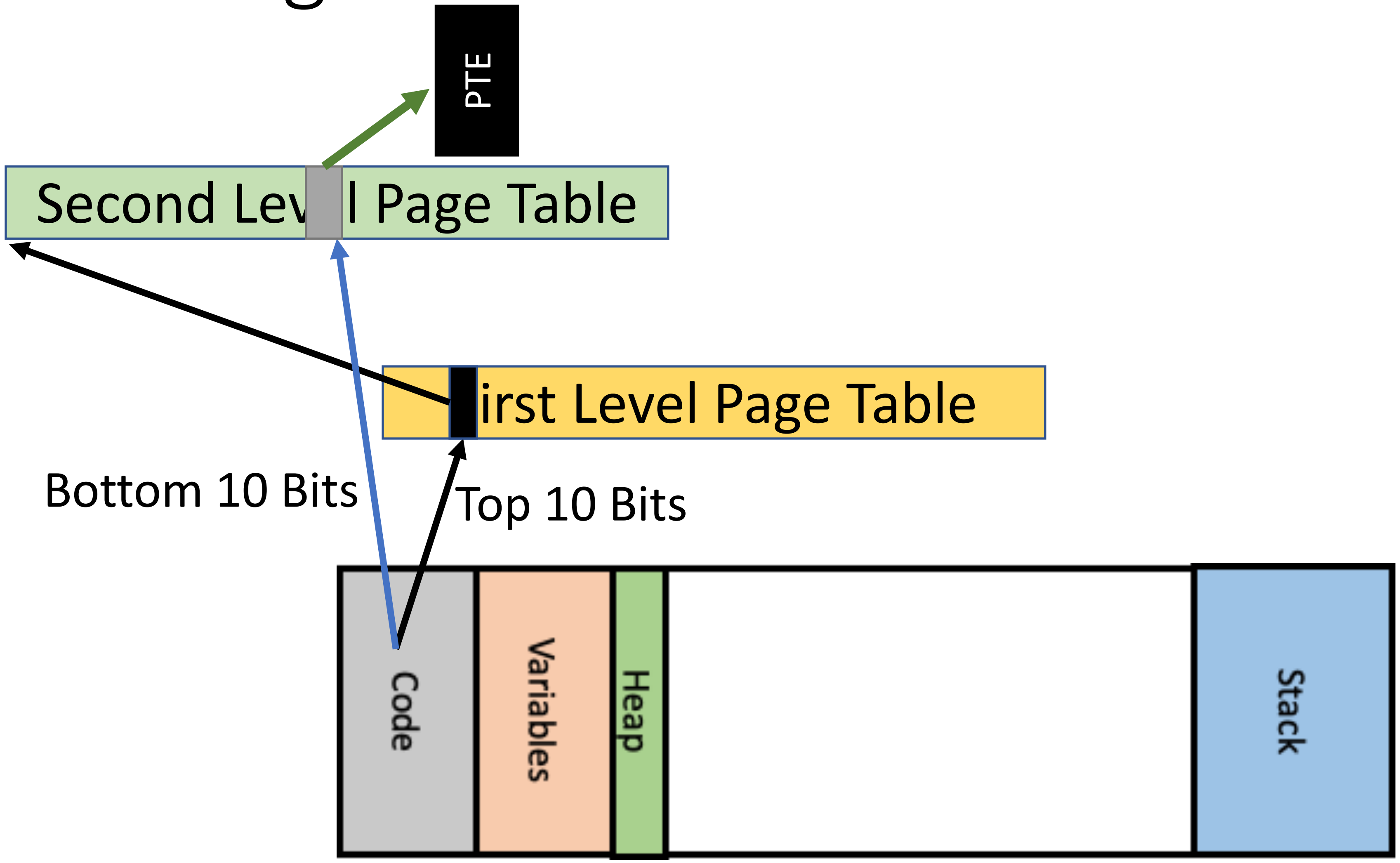
Multi Level Page Tables

- Insight: Utilize the sparsity of virtual address space
- Key Idea: Break VPN into multiple parts, each used as an index at a separate level of the tree.

- Example: with 4K pages VPN is **20** bits. Use **top 10** bits as index into **top-level** page table, **bottom 10 bits** as index into second-level page table



Multi Level Page Tables



Multi Level Page Tables

- Time Complexity:
- O(c). Constant number of look ups per translation depending on tree depth

- Space Complexity:
- Depends on sparsity of address space, but better than flat.