# Memory Management: Advanced Page Tables

*CS 571: Operating Systems (Spring 2020)*

Lecture 8a

Yue Cheng

# Paging Problems

- Page tables are too slow

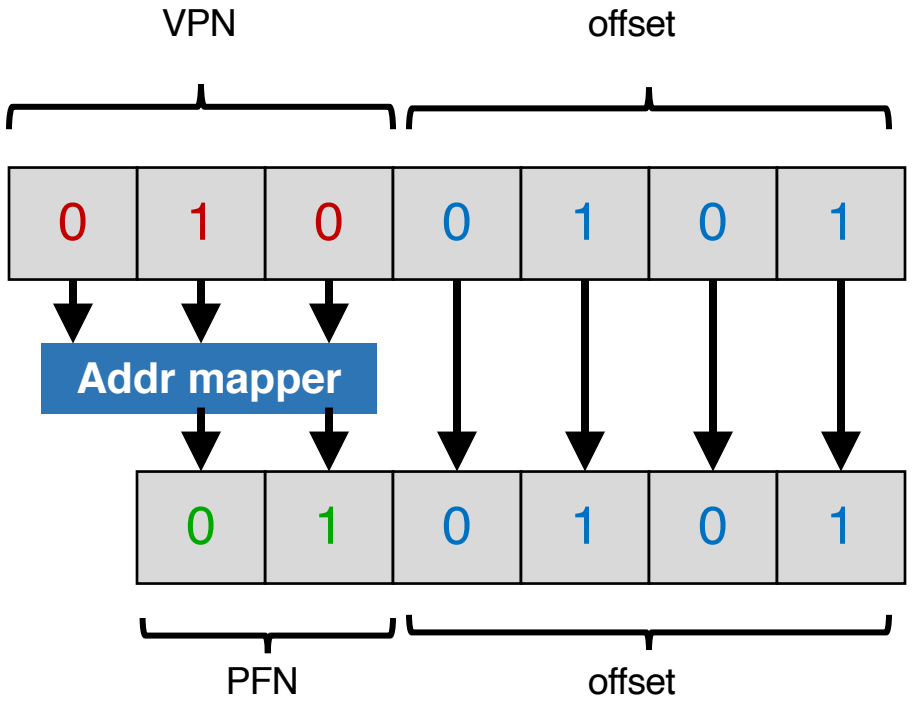
- Page tables are too big

# Review: Page Tables

# Virtual => Physical Addr Mapping

- We need a general mapping mechanism

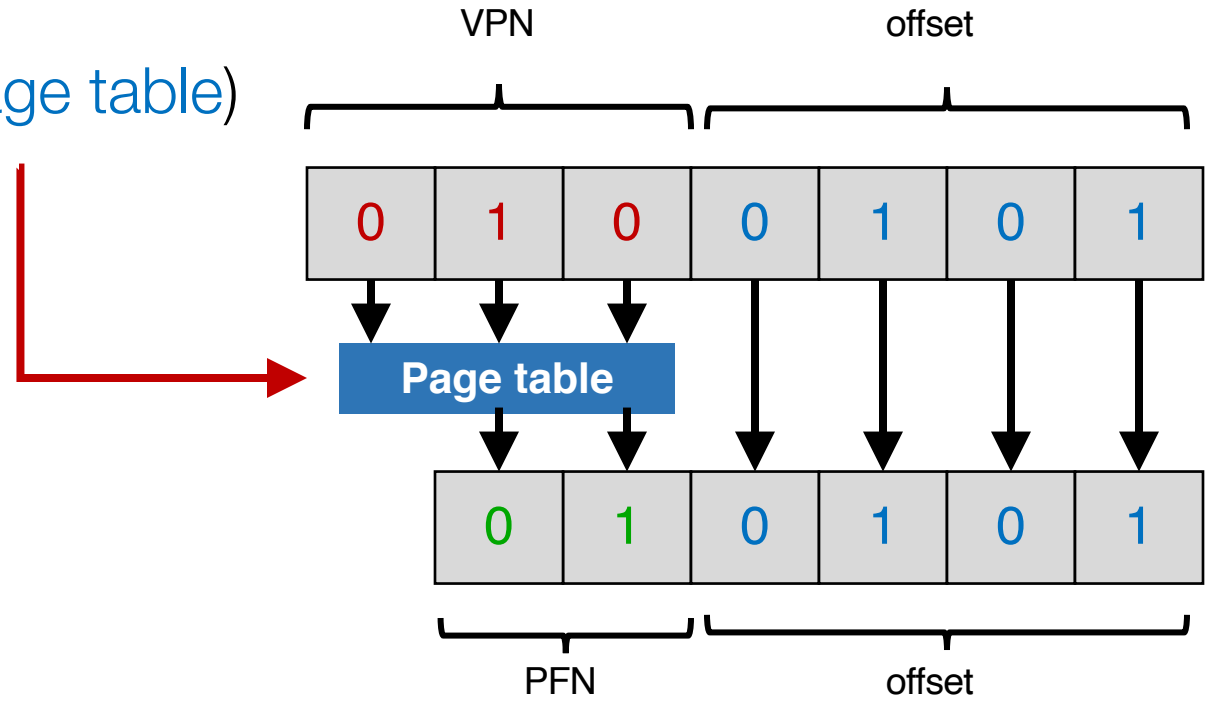
- What data structure is good?
  - Big array



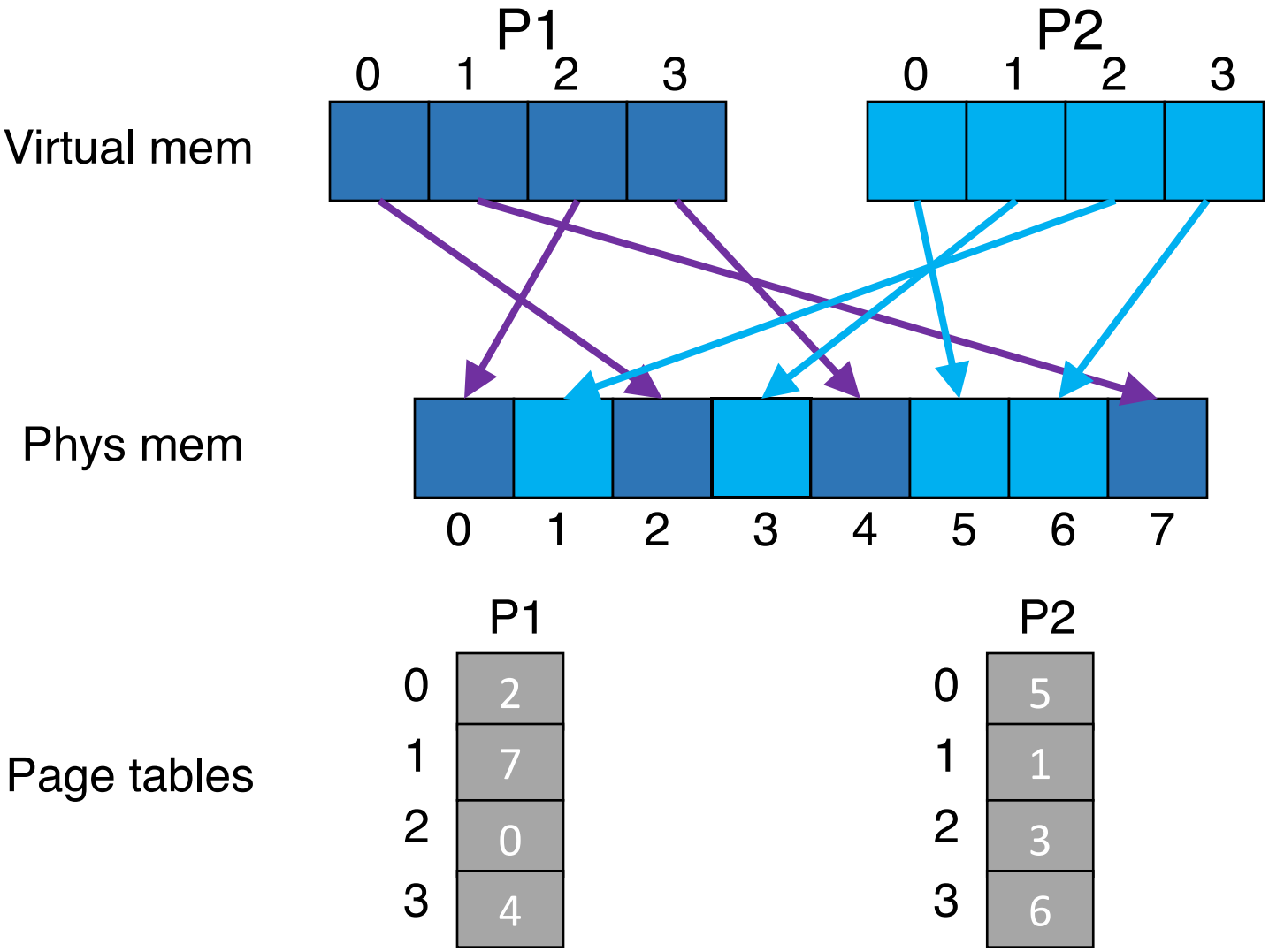
# Virtual => Physical Addr Mapping

- We need a general mapping mechanism

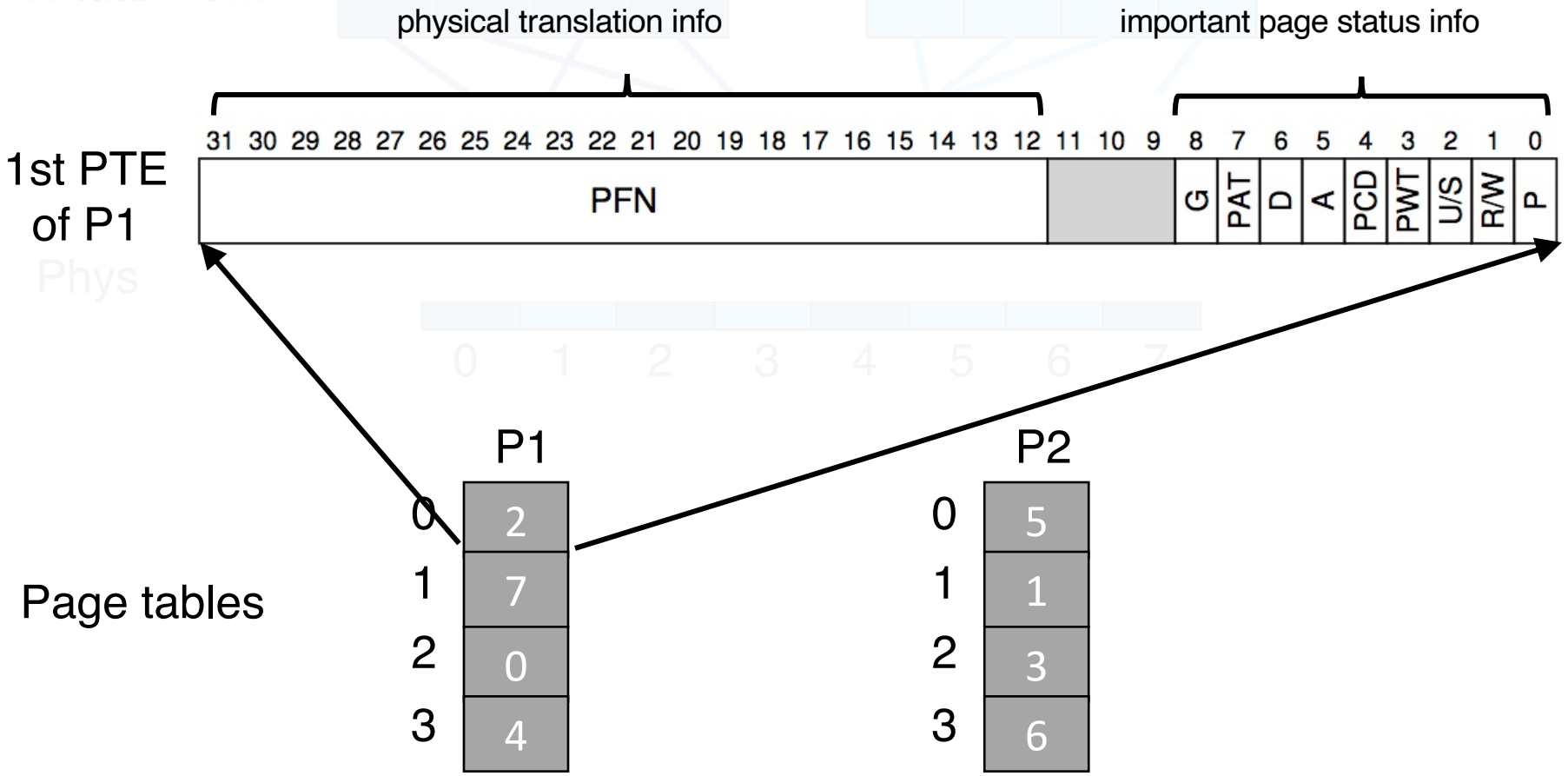
- What data structure is good?
  - Big array
  - (aka linear page table)



# A Simple Page Table Example



# A Simple Page Table Example



# Paging Problems

- Page tables are too slow (covered last week)
  - *TLB* to the rescue!

- Page tables are too big (today)

# How Large are Page Tables?

- A linear page table array for 32-bit address space ($2^{32}$ bytes) and 4KB page ($2^{12}$ bytes)
  - How many pages: $2^{20}$ pages
  - How much memory: 4MB assuming each page-table entry is of 4 bytes
    - 2 ^ (32-log(4KB)) * 4 = 4MB

# How Large are Page Tables?

- A linear page table array for 32-bit address space ($2^{32}$ bytes) and 4KB page ($2^{12}$ bytes)
  - How many pages: $2^{20}$ pages
  - How much memory: 4MB assuming each page-table entry is of 4 bytes
    - 2 ^ (32-log(4KB)) * 4 = 4MB

# How Large are Page Tables?

- A linear page table array for 32-bit address space ($2^{32}$ bytes) and 4KB page ($2^{12}$ bytes)
  - How many pages: $2^{20}$ pages
  - How much memory: 4MB assuming each page-table entry is of 4 bytes
    - 2 ^ (32-log(4KB)) * 4 = 4MB

# How Large are Page Tables?

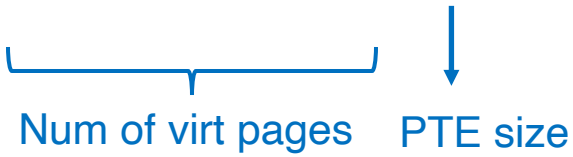
- A linear page table array for 32-bit address space ($2^{32}$ bytes) and 4KB page ($2^{12}$ bytes)
  - How many pages: $2^{20}$ pages
  - How much memory: 4MB assuming each page-table entry is of 4 bytes
    - 2 ^ (32-log(4KB)) * 4 = 4MB

# How Large are Page Tables?

- A linear page table array for 32-bit address space ($2^{32}$ bytes) and 4KB page ($2^{12}$ bytes)
  - How many pages: $2^{20}$ pages
  - How much memory: 4MB assuming each page-table entry is of 4 bytes
    - 2 ^ (32-log(4KB)) * 4 = 4MB

# How Large are Page Tables?

- A linear page table array for 32-bit address space ($2^{32}$ bytes) and 4KB page ($2^{12}$ bytes)
  - How many pages: $2^{20}$ pages
  - How much memory: 4MB assuming each page-table entry is of 4 bytes
    - 2 ^ (32-log(4KB)) * 4 = 4MB



# Page Tables are Too Big

- A linear page table array for 32-bit address space ($2^{32}$ bytes) and 4KB page ($2^{12}$ bytes)
  - How many pages: $2^{20}$ pages
  - How much memory: 4MB assuming each page-table entry is of 4 bytes
    - 2 ^ (32-log(4KB)) * 4 = 4MB
- One page table for one process!
  - A system with 100 process: 400MB only for storing page tables in memory
- Solution??

# Naïve Solution

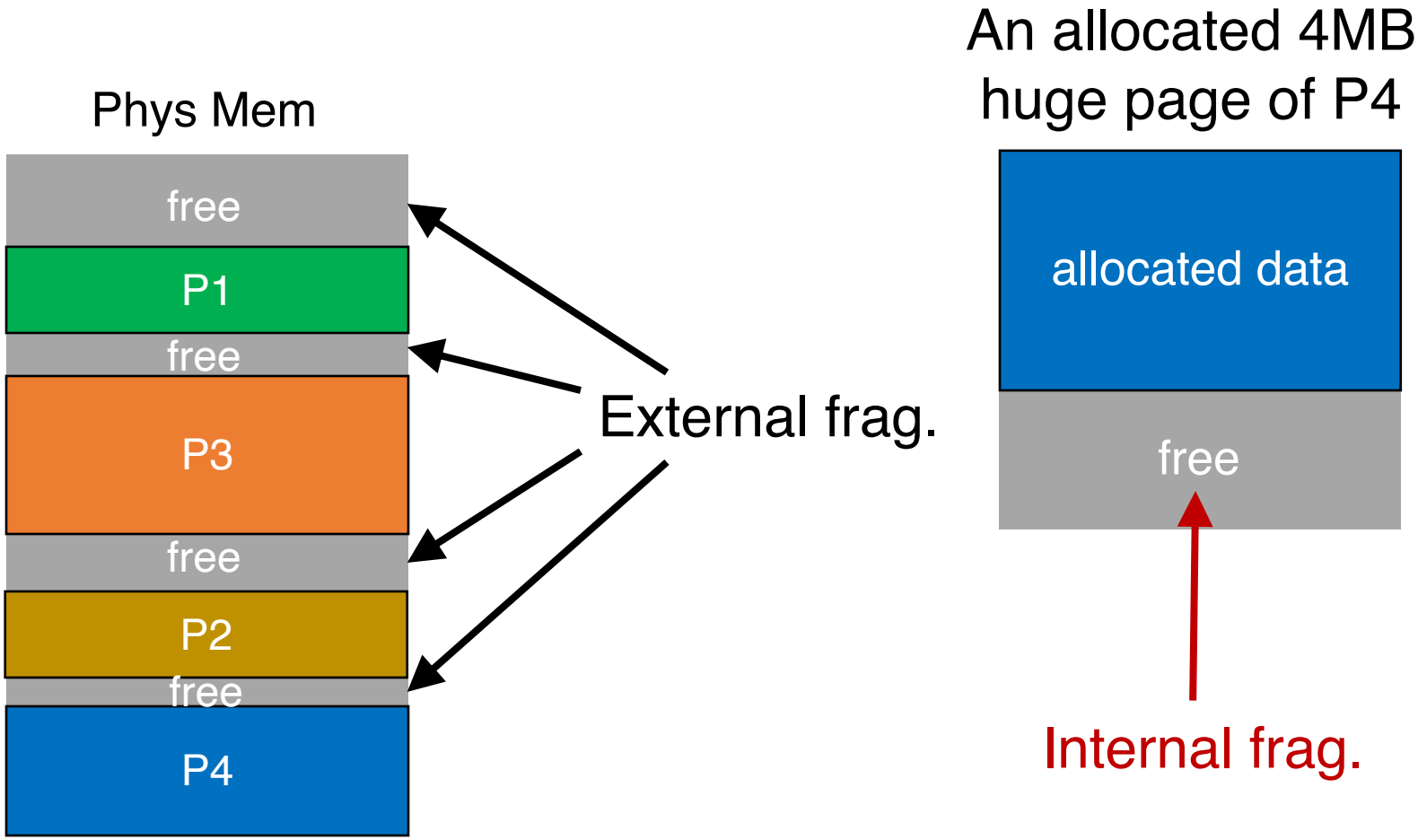
- Reduce the granularity
  - by increasing the page size

# Naïve Solution

- Reduce the granularity
  - by increasing the page size

- Why are 4MB pages bad?
  - Internal fragmentation!

# Fragmentation




Assume each process consists of multiple 4MB pages

# Fragmentation

# Approaches

- Approach 1: Linear Inverted Page Table
- Approach 2: Hashed Inverted Page Table
- Approach 3: Multi-level Page Table

# Approaches

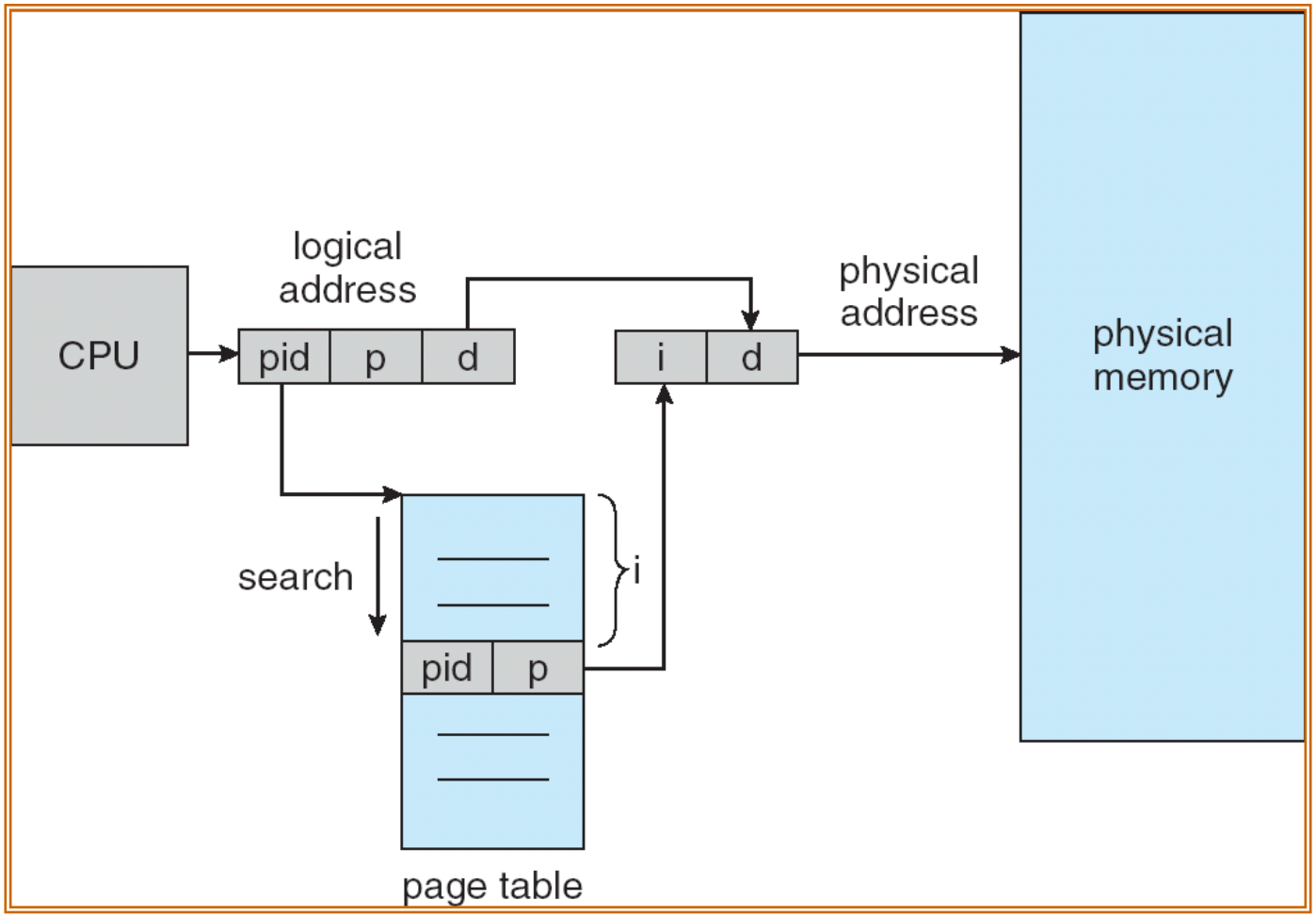
- Approach 1: Linear Inverted Page Table
- Approach 2: Hashed Inverted Page Table
- Approach 3: Multi-level Page Table

# Linear Inverted Page Table

- Idea: Instead of keeping one page table per process, the system keeps a single page table that has an entry for each physical frame of the system

- Each entry tells which process owns the page, and VPN to PFN translation

# Linear Inverted Page Table Example



# Linear Inverted Page Table

- Idea: Instead of keeping one page table per process, the system keeps a single page table that has an entry for each physical frame of the system
- Each entry tells which process owns the page, and VPN to PFN translation
- Goal: use linear search to find the index `i`
  - The reason why it's "**inverted**"

- Pros: Extreme memory savings
- Cons: A linear search is expensive
  - Solution??

# Approaches

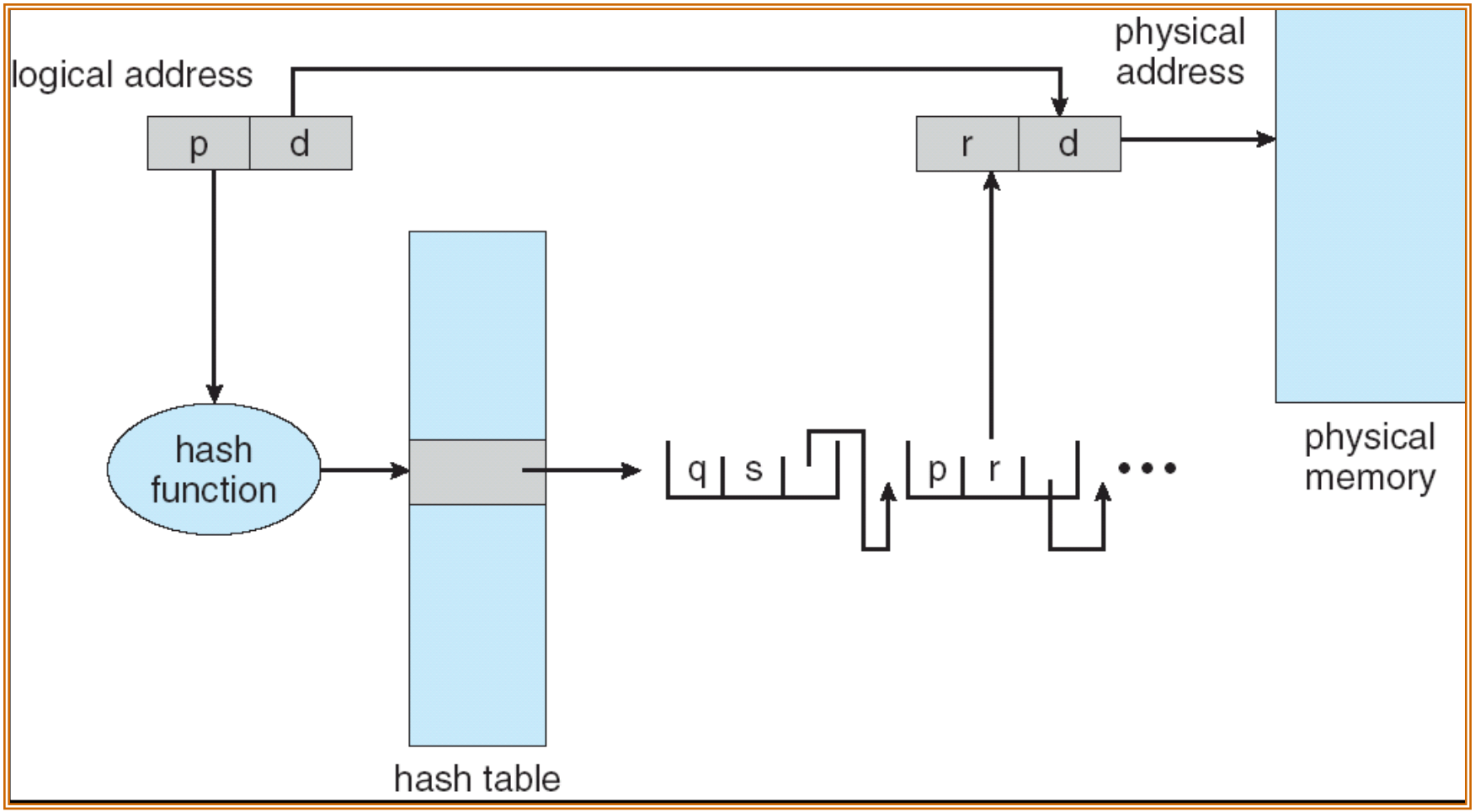
- Approach 1: Linear Inverted Page Table
- Approach 2: Hashed Inverted Page Table
- Approach 3: Multi-level Page Table

# Hashed Inverted Page Table

- For large address spaces, a hashed page table can be used, with the hash value being the VPN

- Idea:
  - A PTE contains a chain of elements hashing to the same location (to handle collisions) within PT
  - Each element has three fields: (a) VPN, (b) PFN, (c) a pointer to the next element in the linked list
  - VPNs are compared in this chain searching for a match. If a match is found, the corresponding PFN is extracted

# Hashed Inverted Page Table Example



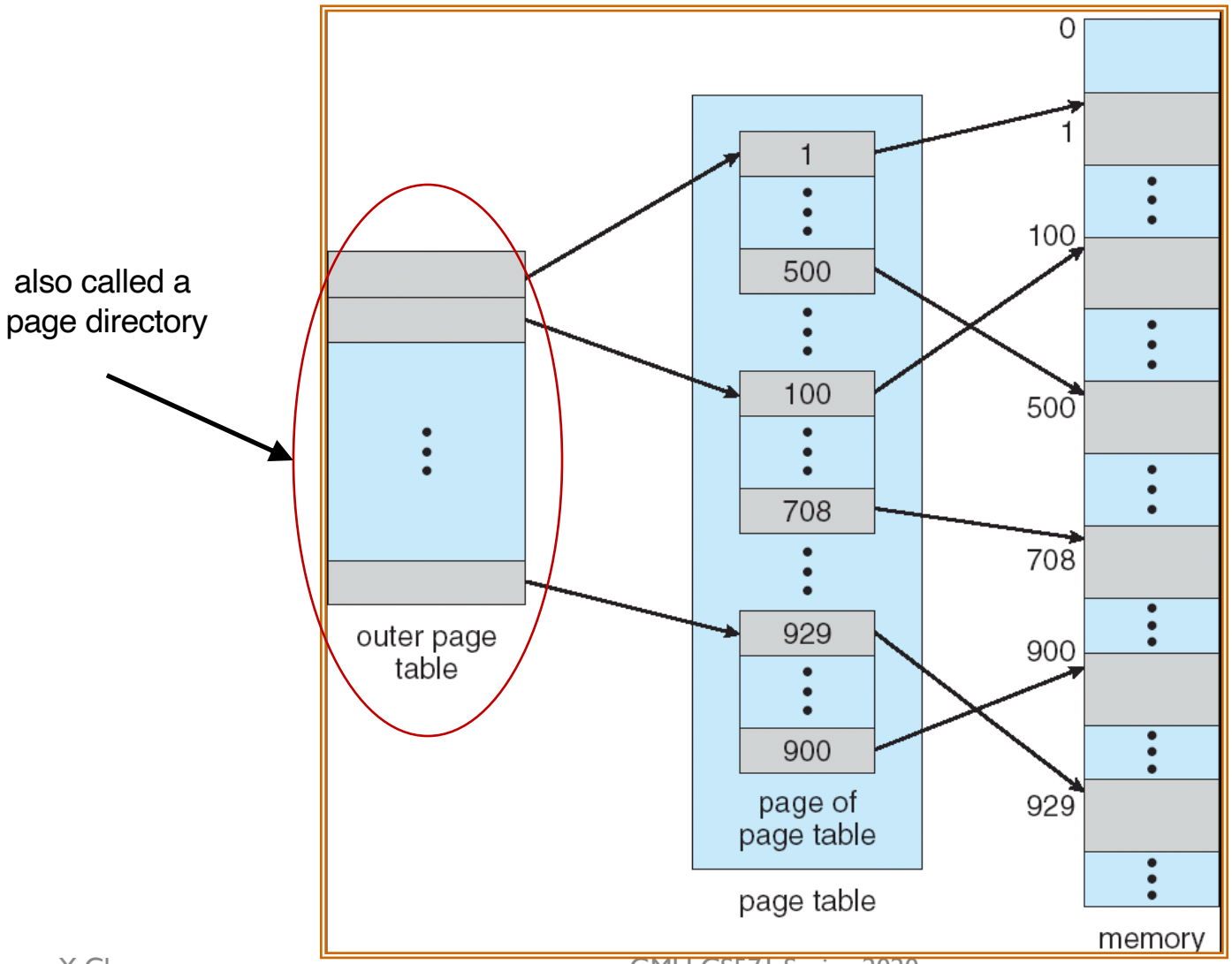
# Approaches

- Approach 1: Linear Inverted Page Table
- Approach 2: Hashed Inverted Page Table
- Approach 3: Multi-level Page Table

# Multi-level Page Table

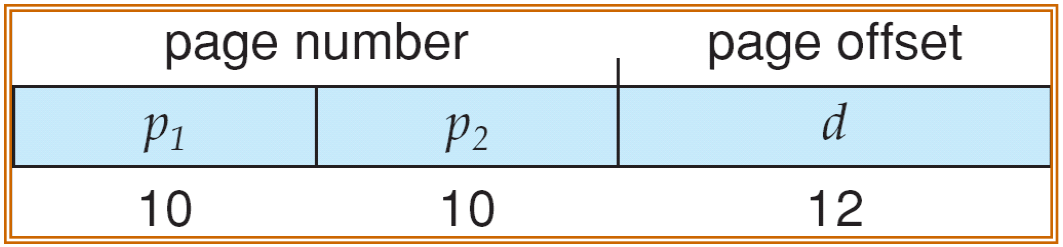
- Idea:
  - Break the page table into pages (the entire page table is paged!)
  - Only have pieces with >0 valid entries
    - Don't allocate the page of page table if the entire page of page-table entries is invalid
- Used by x86
- A simple technique is a two-level page table

# Two-level Page Table Example



# Two-level Paging

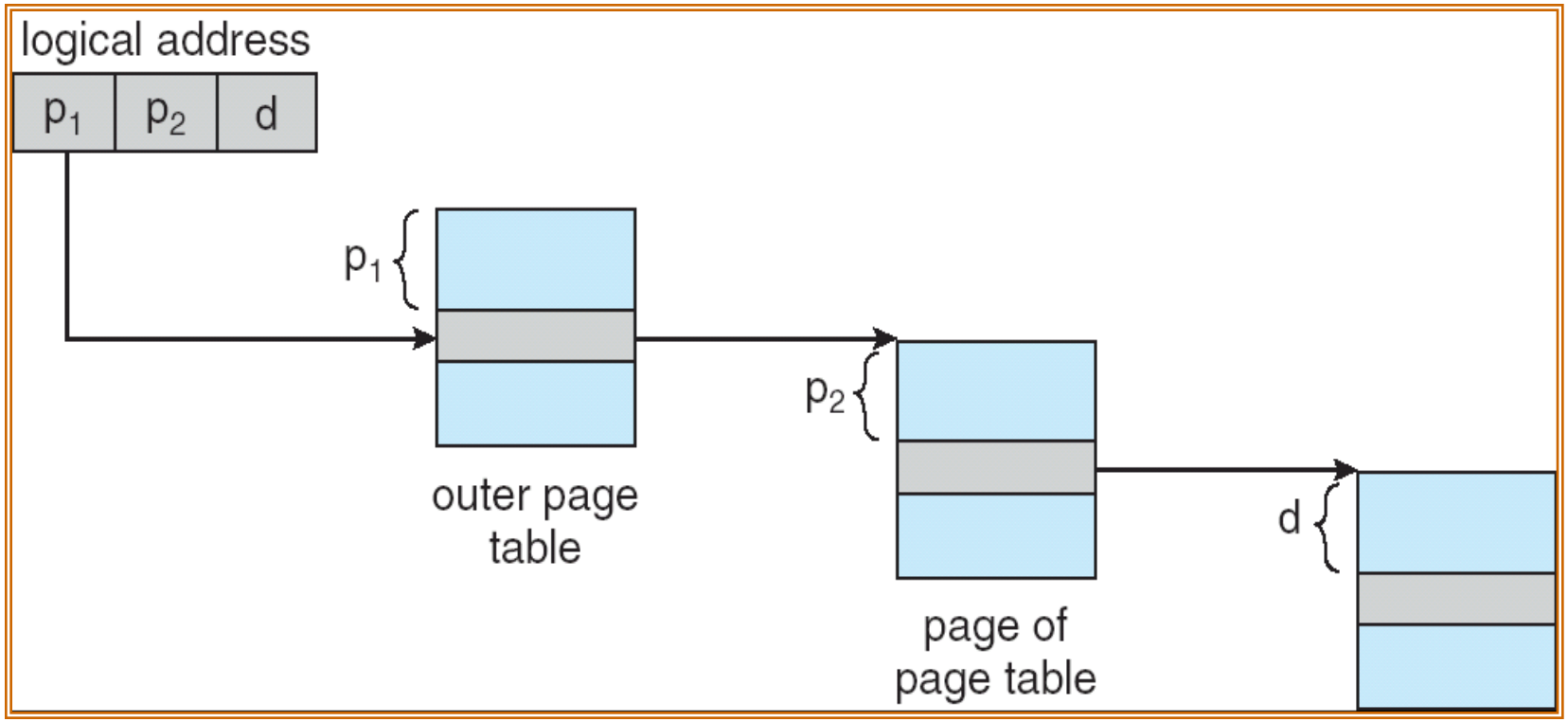
- A logical address (on 32-bit machine with 4KB page size) is divided into
  - a page number consisting of 20 bits
  - a page offset consisting of 12 bits
- A page table entry is 4 bytes
- Since the page table is paged, the page number is further divided into
  - $\mathtt{p}_1$: a 10-bit page directory index
  - $\mathtt{p}_2$: a 10-bit page table index
- The logical address is as follows:




where $p_1$ is an index into the outer page table, and $p_2$ is the displacement within the page of the inner page table

# Address Translation Scheme

- Address translation scheme for a two-level 32-bit paging architecture



# > 2 Levels

- Problem: page directory may not fit in a page

- Solution:
  - Split page directories into pieces
  - Use another page dir to refer to the page dir pieces

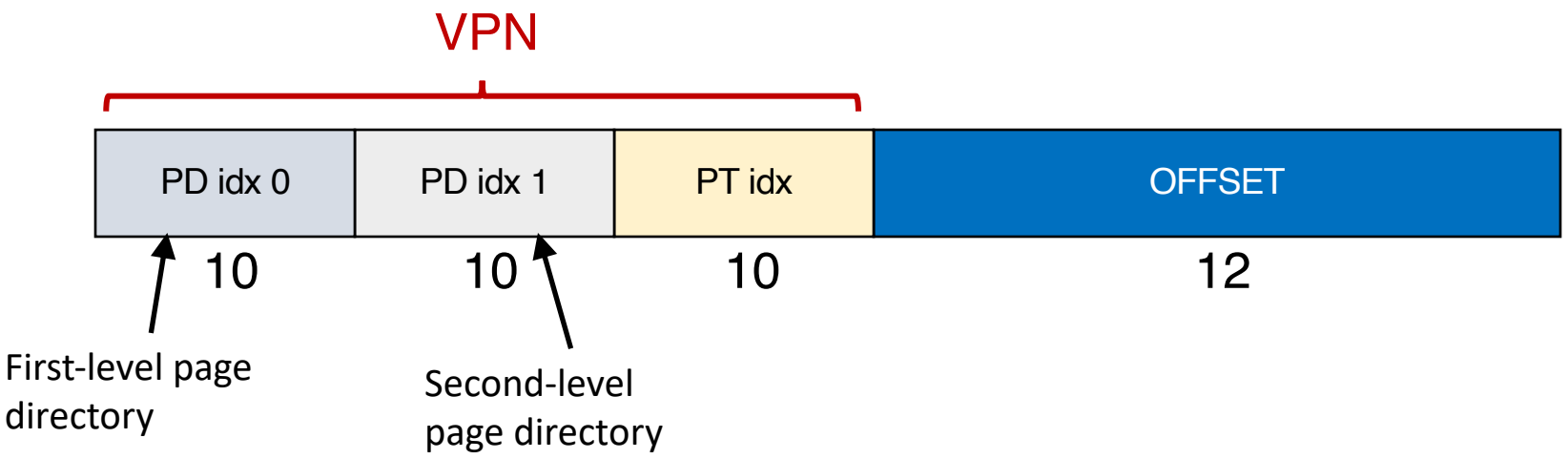
# > 2 Levels

- Problem: page directory may not fit in a page
- Solution:
  - Split page directories into pieces
  - Use another page dir to refer to the page dir pieces
- Possible to extend to 3- or 4-level
- E.g., 64-bit Ultra-SPARC would need 7 levels of paging

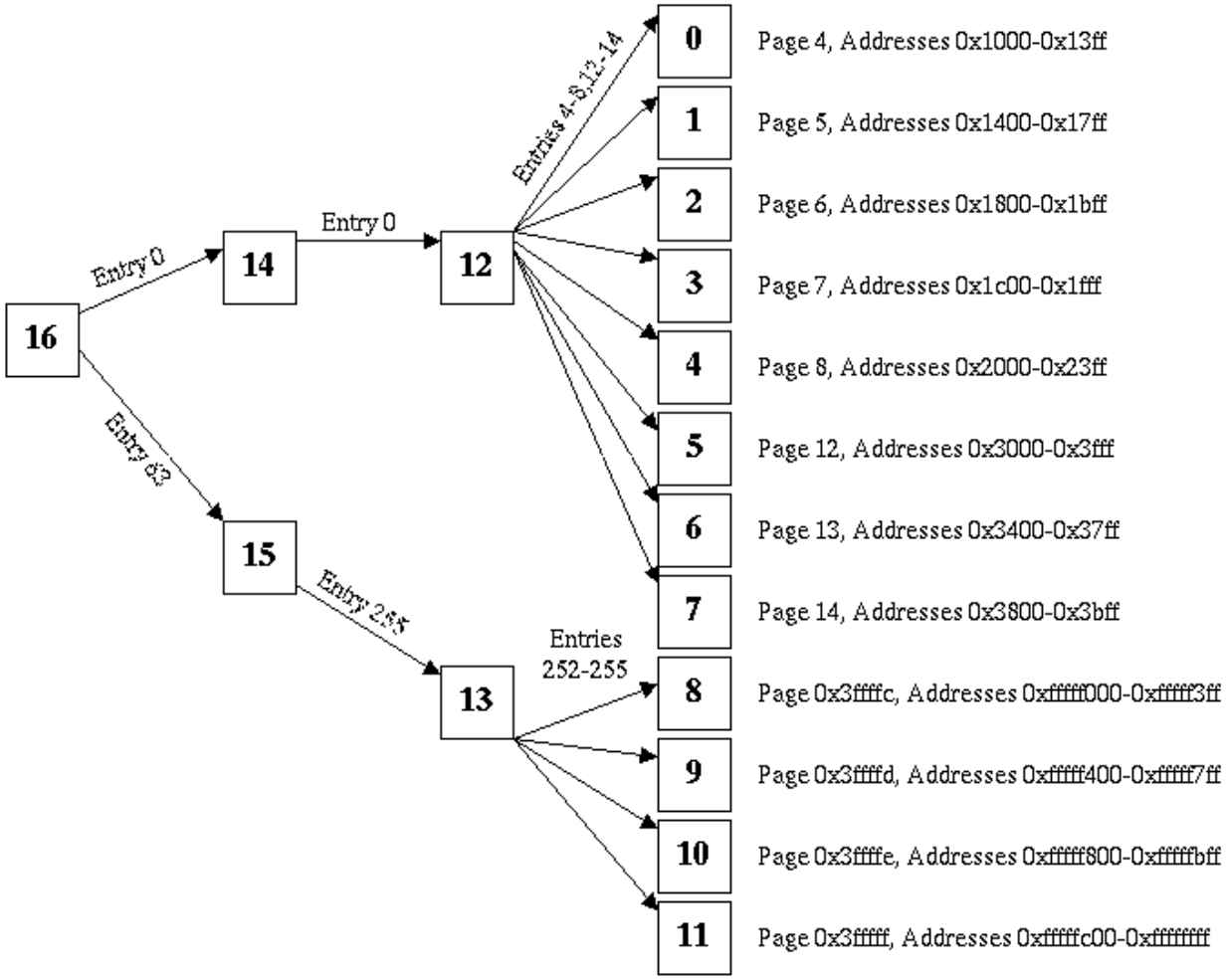
# > 2 Levels

- Problem: page directory may not fit in a page

- Solution:
  - Split page directories into pieces
  - Use another page dir to refer to the page dir pieces



# Multi-level Page Table Example




http://web.eecs.utk.edu/~mbeck/classes/cs560/560/oldtests/t2/2003/Answers.html