Operating Systems

Lecture 7b

Dr Ronald Grau

School of Engineering and Informatics

Spring term 2018

Previously

Memory management

- Addressing and address spaces
- Partitioning and segmentation

Today

Memory management

- Virtual memory
- Paging

Recap: Questions

1. What is a logical address?
2. What are the two main purposes of the memory management unit?
3. Why do we have to bind or translate addresses?
4. When is address-binding during run-time mandatory?
5. Which of the techniques we know performs non-contiguous memory allocation?
6. How is access to a process' memory partition protected?
7. What is segmentation?
8. What is internal fragmentation and when does it occur?
9. What is external fragmentation and when does it occur?

Virtual memory

Objectives

- Hide physical memory
- Memory protection
- Illusion of unbounded memory

- Logical address space
- Partitioning/segmentation
  - Problem: Limited size of processes/segments – overlays required

Solution: Paging – we load processes only partially into memory

Paging

Principles

- Physical memory divided in frames of equal size power of 2, e.g. 4096 bytes
- Process image divided in pages of the same size
- Pages loaded into frames
- Secondary (swap) storage for pages that are not in memory

Paging

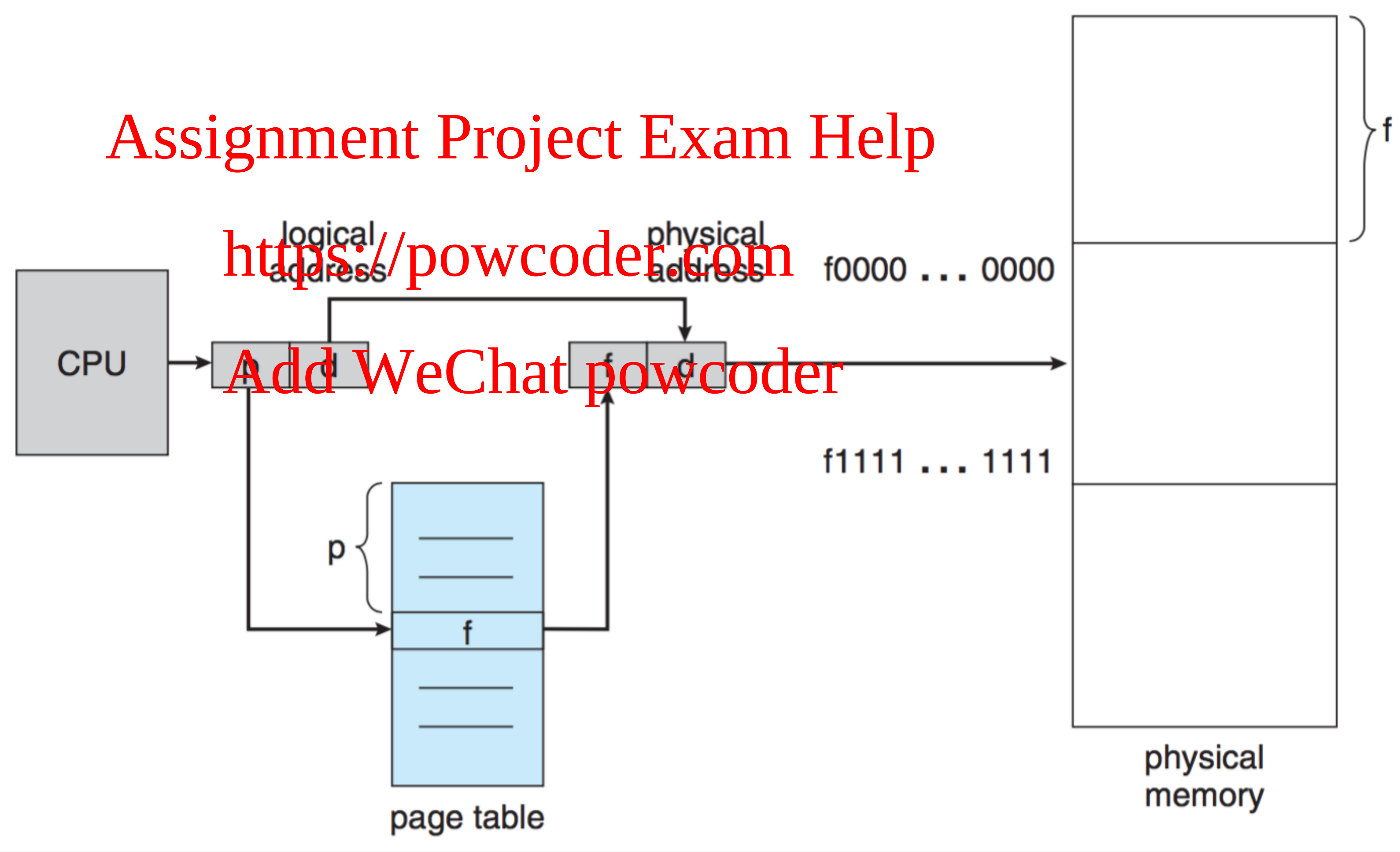
Properties

- Non-contiguous allocation
- Process image can be larger than available main memory
- Many processes can reside (partially) in memory
- Invisible to user (unlike segmentation or overlays)

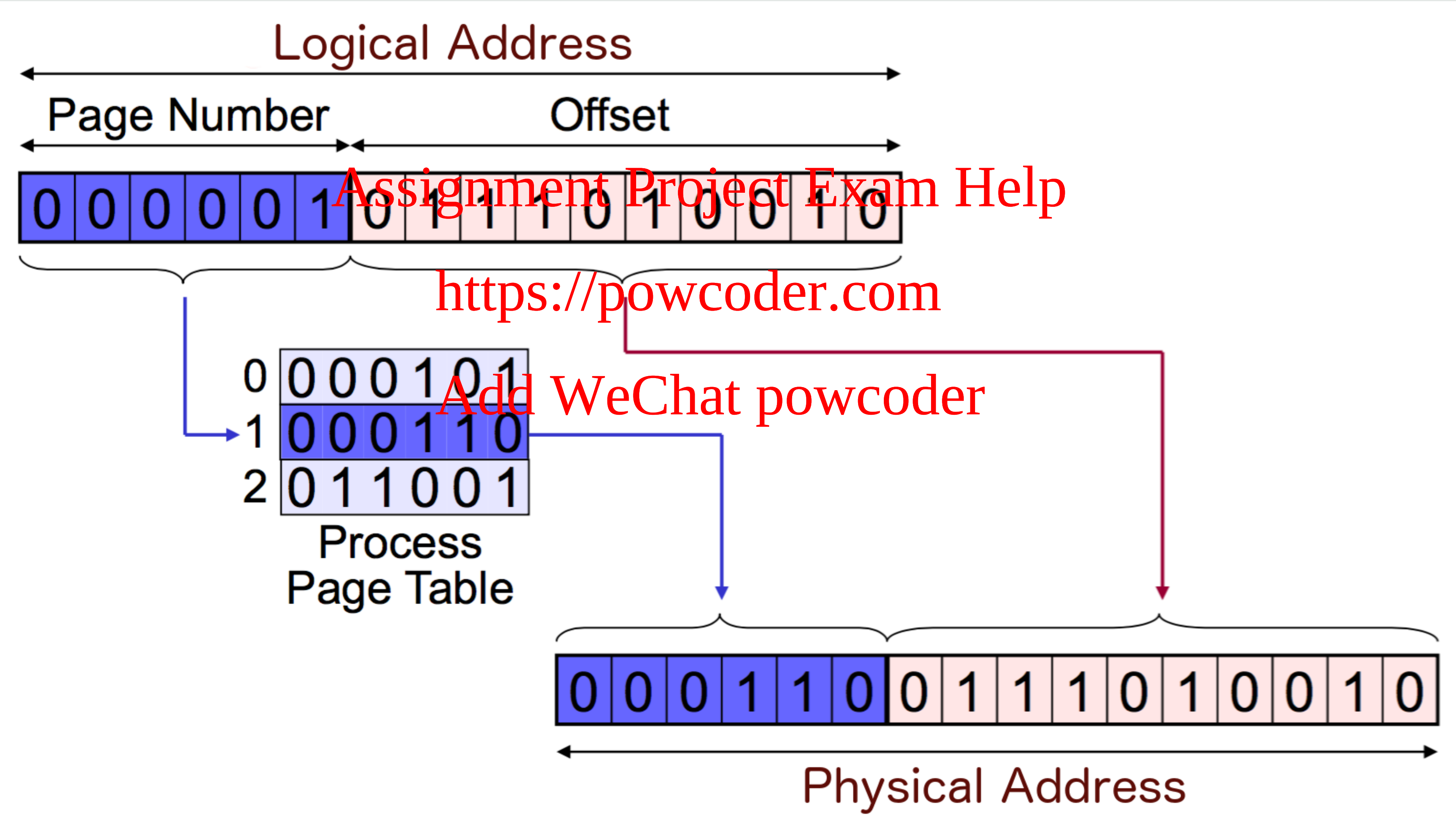
Data structures: Page table for each process

- Current frame number for each page
- Free frame list

Paging


CPU
logical address
p
d
physical address
f
d
f0000 . . . 0000
f1111 . . . 1111
f
p
f
page table
physical memory

Paging


Logical Address
Page Number
Offset
0 0 0 0 0 1 0 1 1 1 0 1 0 0 1 0
0 0 0 0 1 0 1
1 0 0 0 1 1 0
2 0 1 1 0 0 1
Process
Page Table
0 0 0 1 1 0 0 1 1 1 0 1 0 0 1 0
Physical Address

Page table

One page table per process

- Indexed by page number
- Present bit (loaded?)
- Frame number (if present)
- Modified bit (written since loaded?)
- Control and access bits (read-write-execute, kernel/user, etc)

Page table

Effect of page size

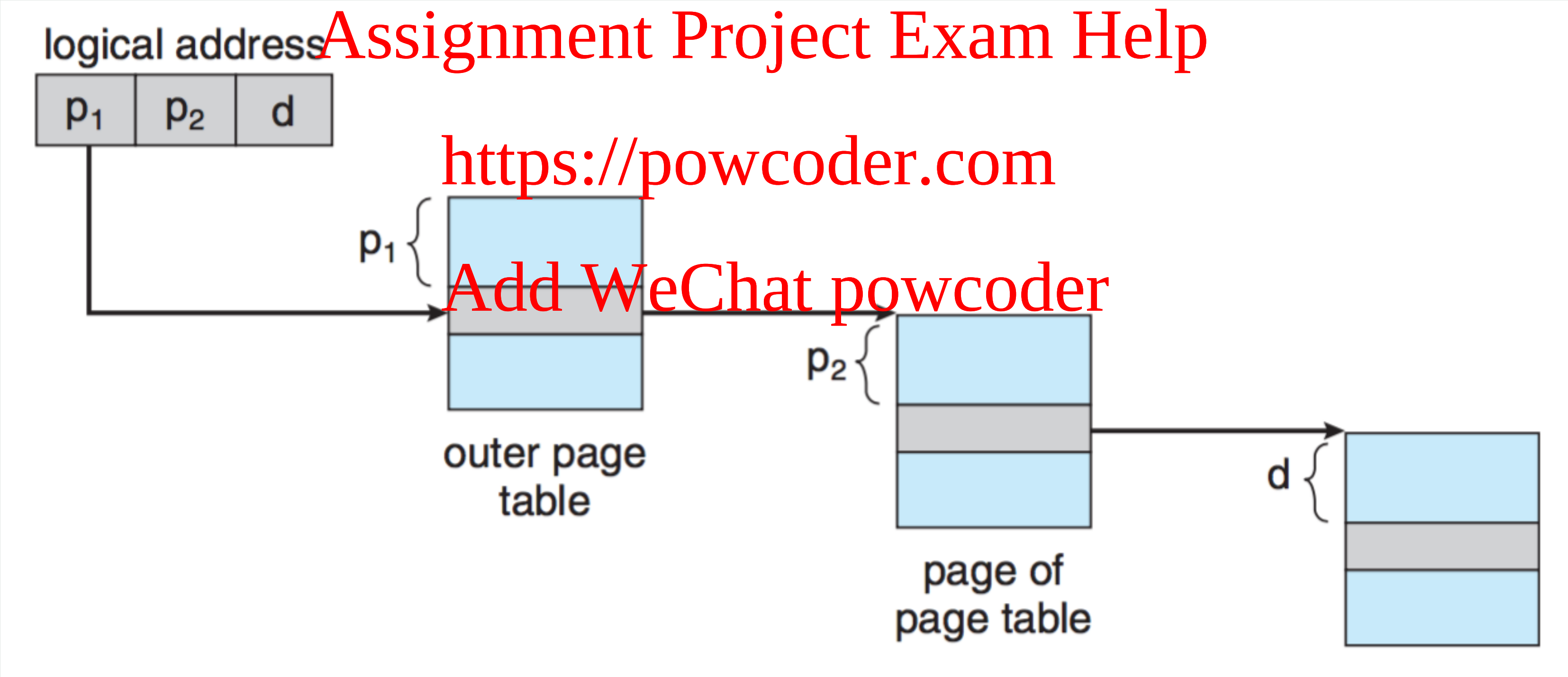
- Equal page size: Eliminates external fragmentation
- Larger page size: Larger internal fragmentation
- Smaller page size: Larger page tables

Example:

- 4KB page size, 48 bit logical address space
- 8 bytes per table entry → 512 GB just for storing the page table

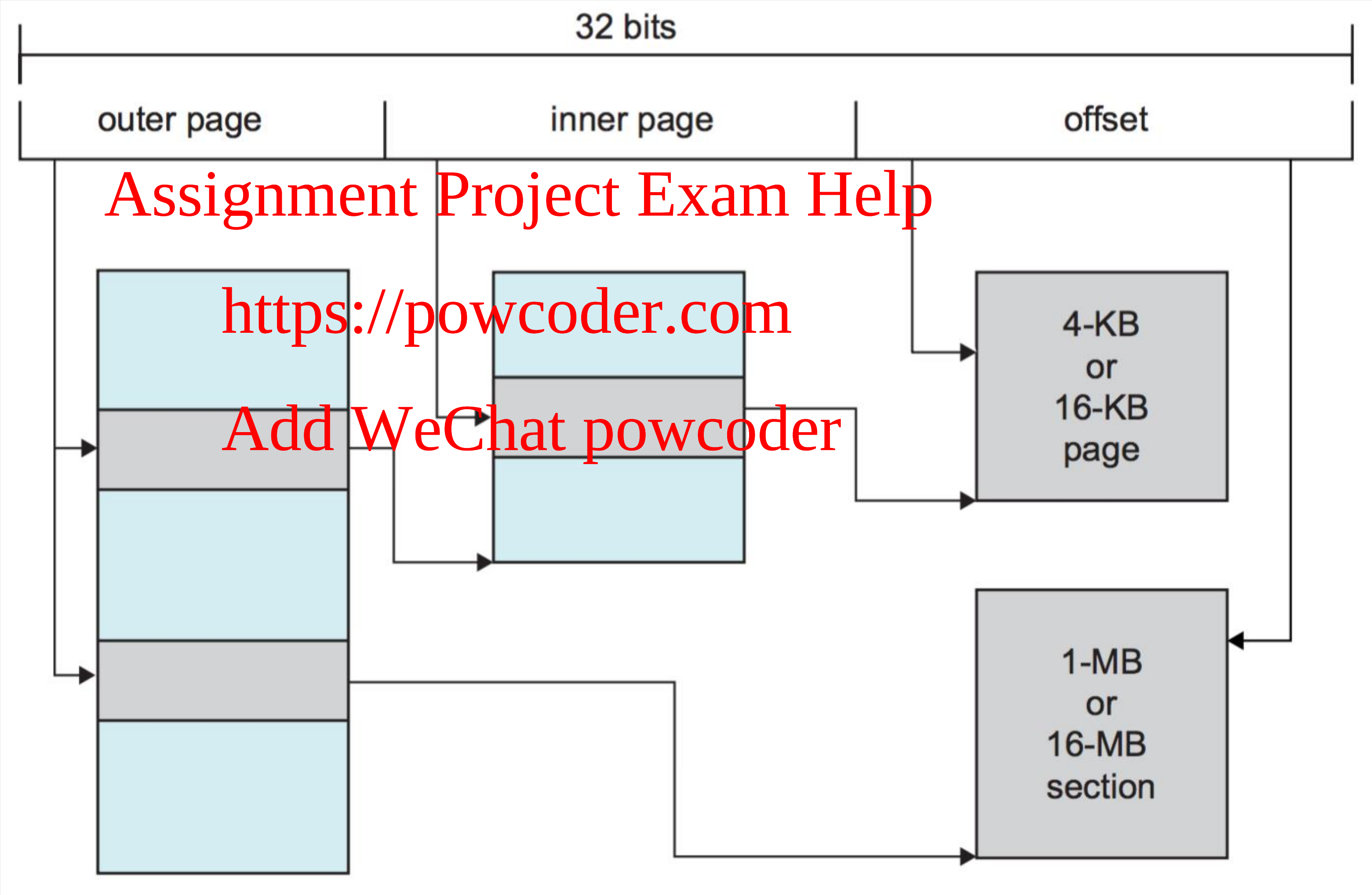
Multi-Level Paging

Page table itself is paged



E.g. four levels on many Intel architectures

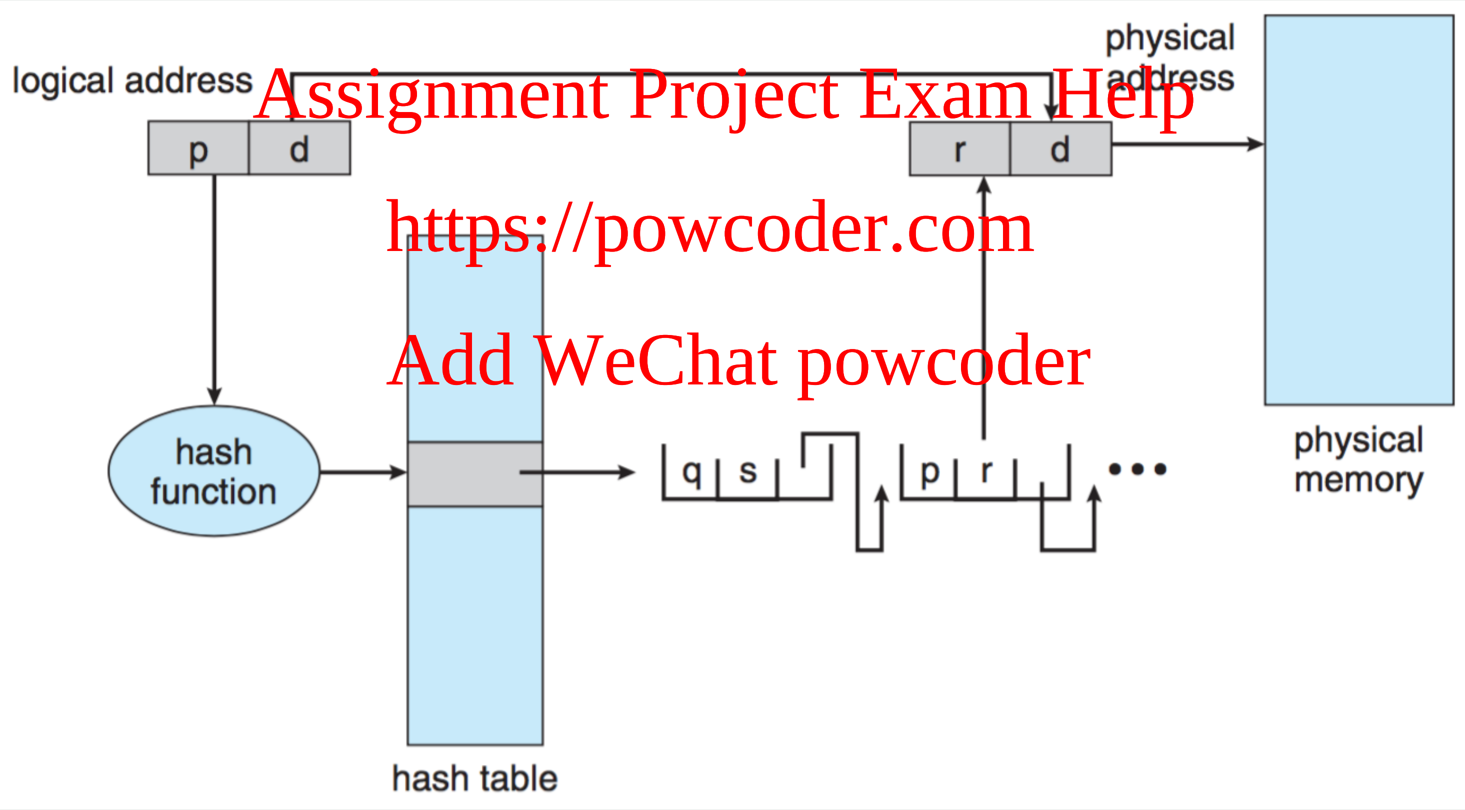
Hashed Page Table



E.g. ARM

Hashed Page Table

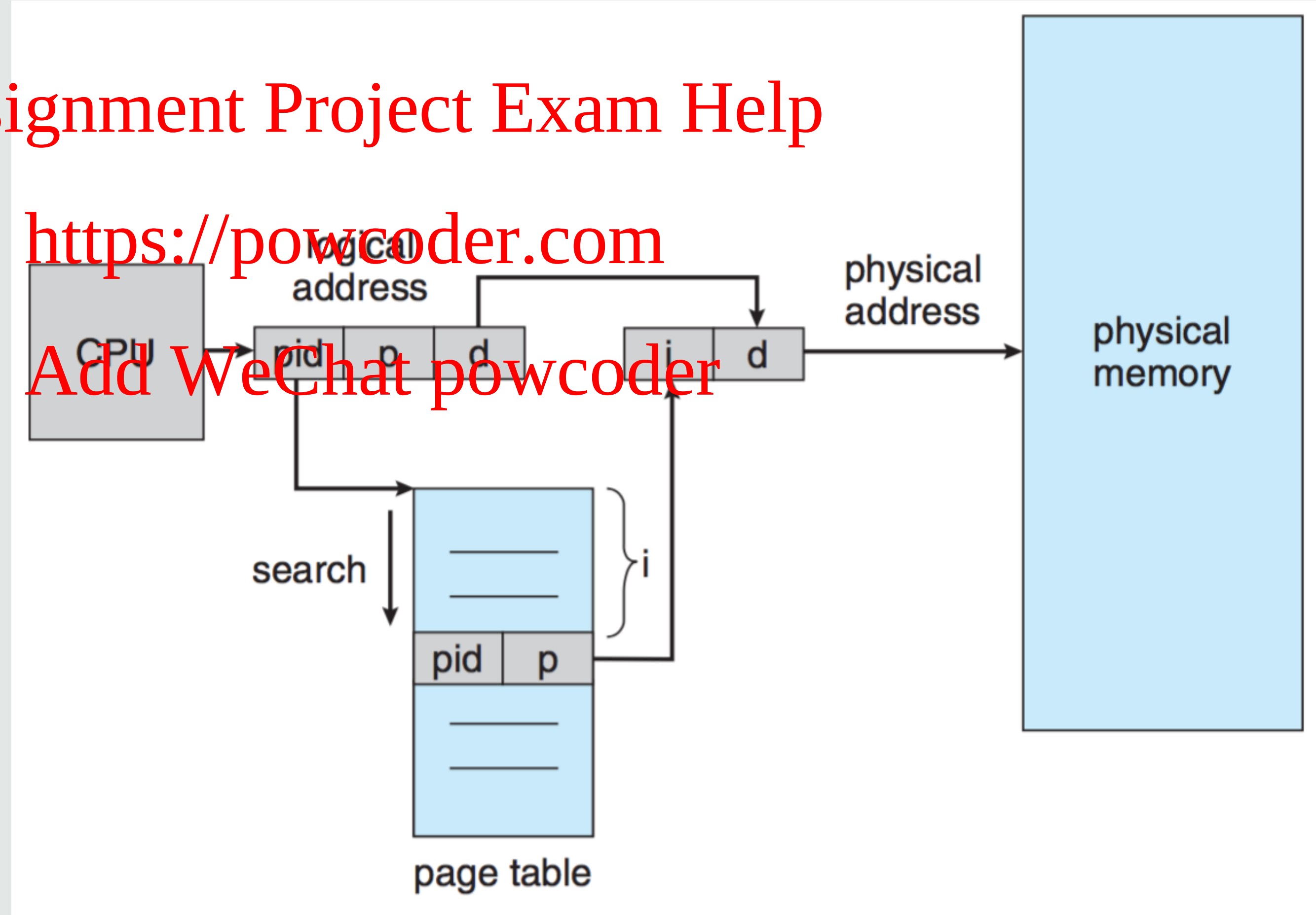
Table size ~number of used frames



E.g. Solaris on SPARC

Inverted Page Table

One table for all processes



E.g. Power PC

Translation Look-Aside Buffer (TLB)

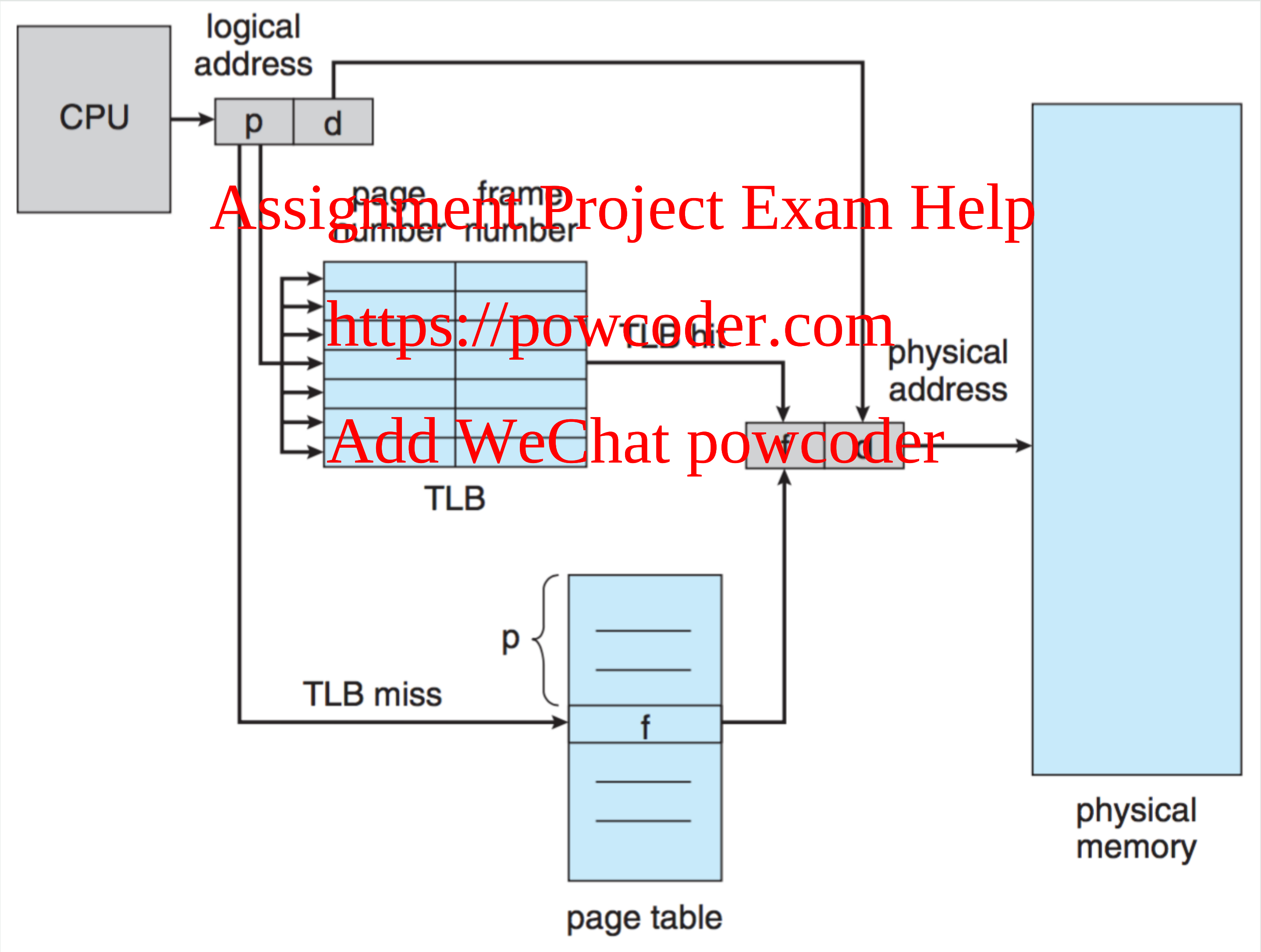
Page table in main memory

- Each address translation requires at least two memory accesses

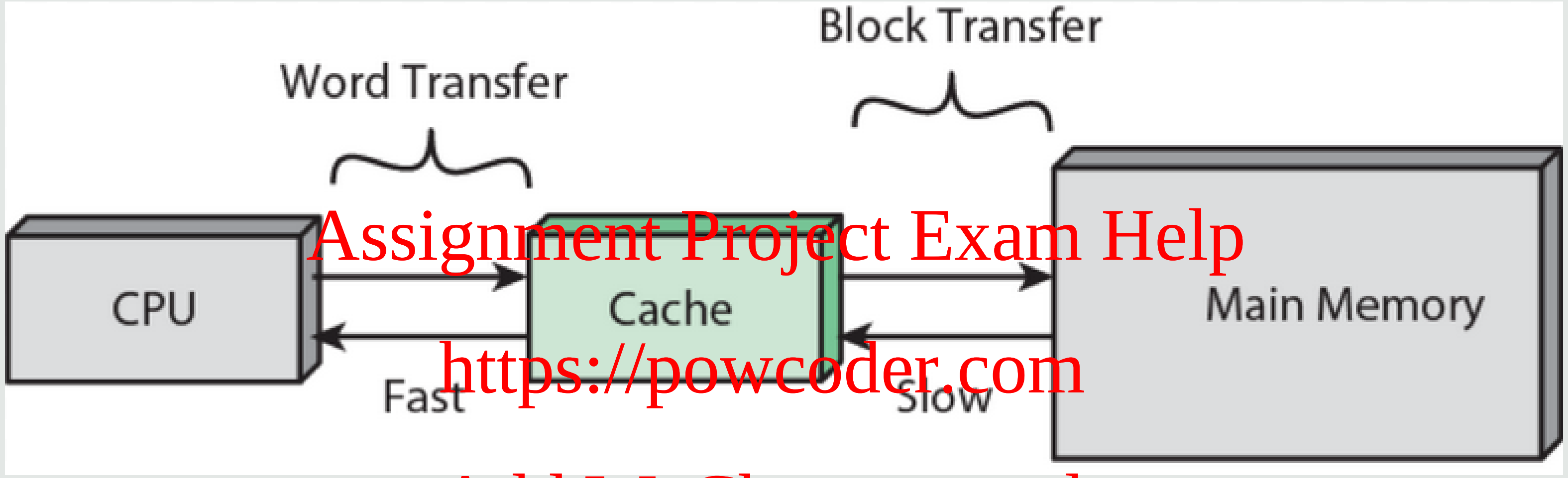
Translation Look-Aside Buffer

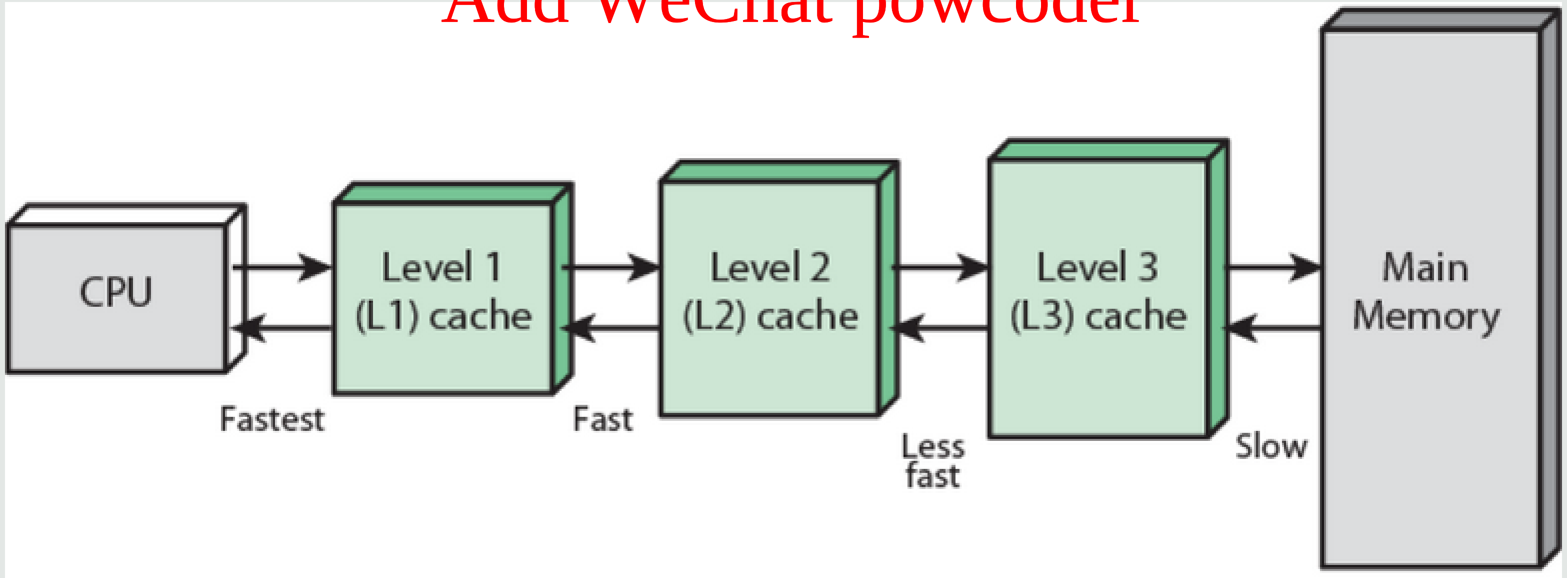
- Cache in MMU for page table entries
- Cache policy, e.g. most recently used
- Associative access
- Cleared on each context switch

Translation Look-Aside Buffer (TLB)

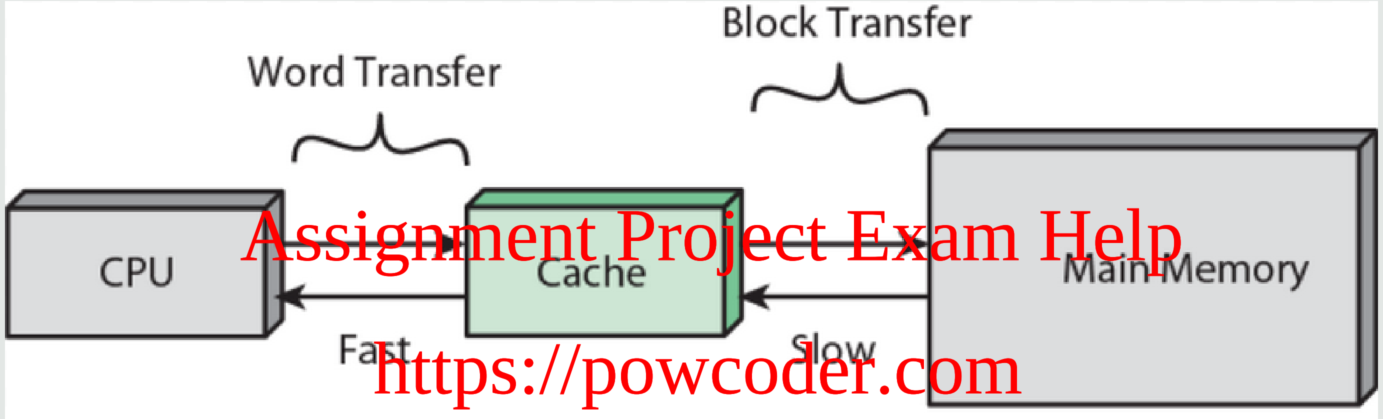


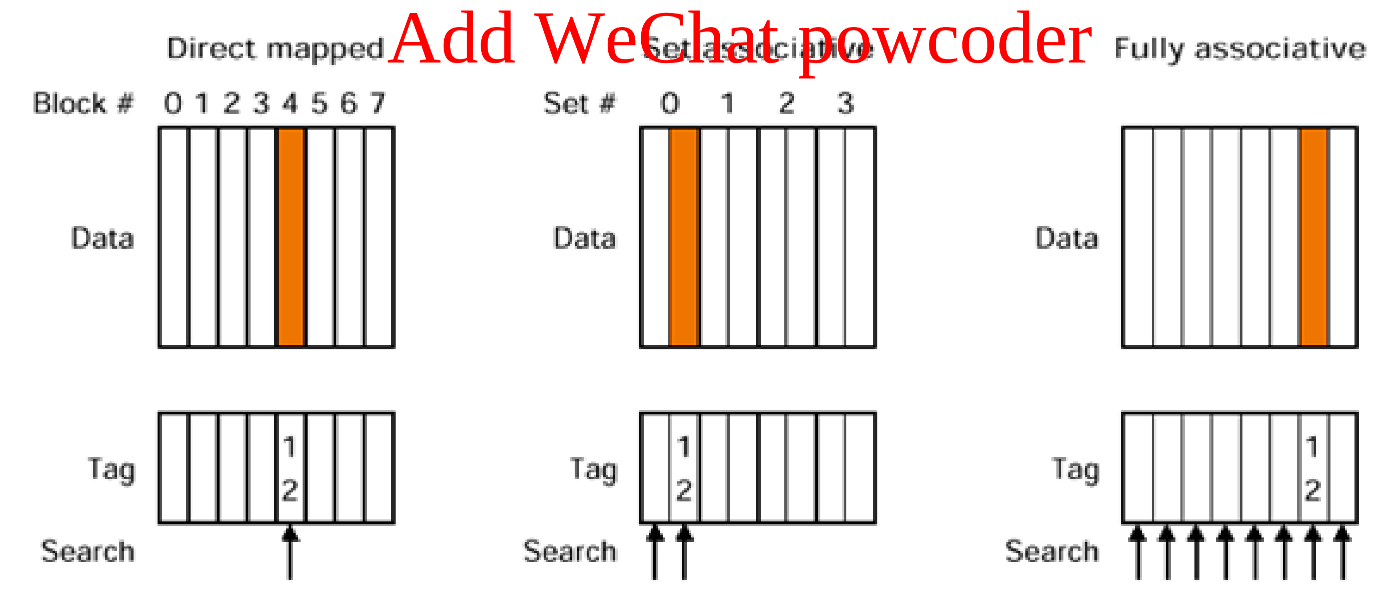
Cache Memory and Associativity





Cache Memory and Associativity





Page Faults and Thrashing

Resident set

- Pages of a process that are currently assigned to frames

Page fault

- Access to a page that is not resident
  → need to swap in page
- Which page should be replaced?
  → page replacement
- What to do on process start?
  → demand vs prepaging

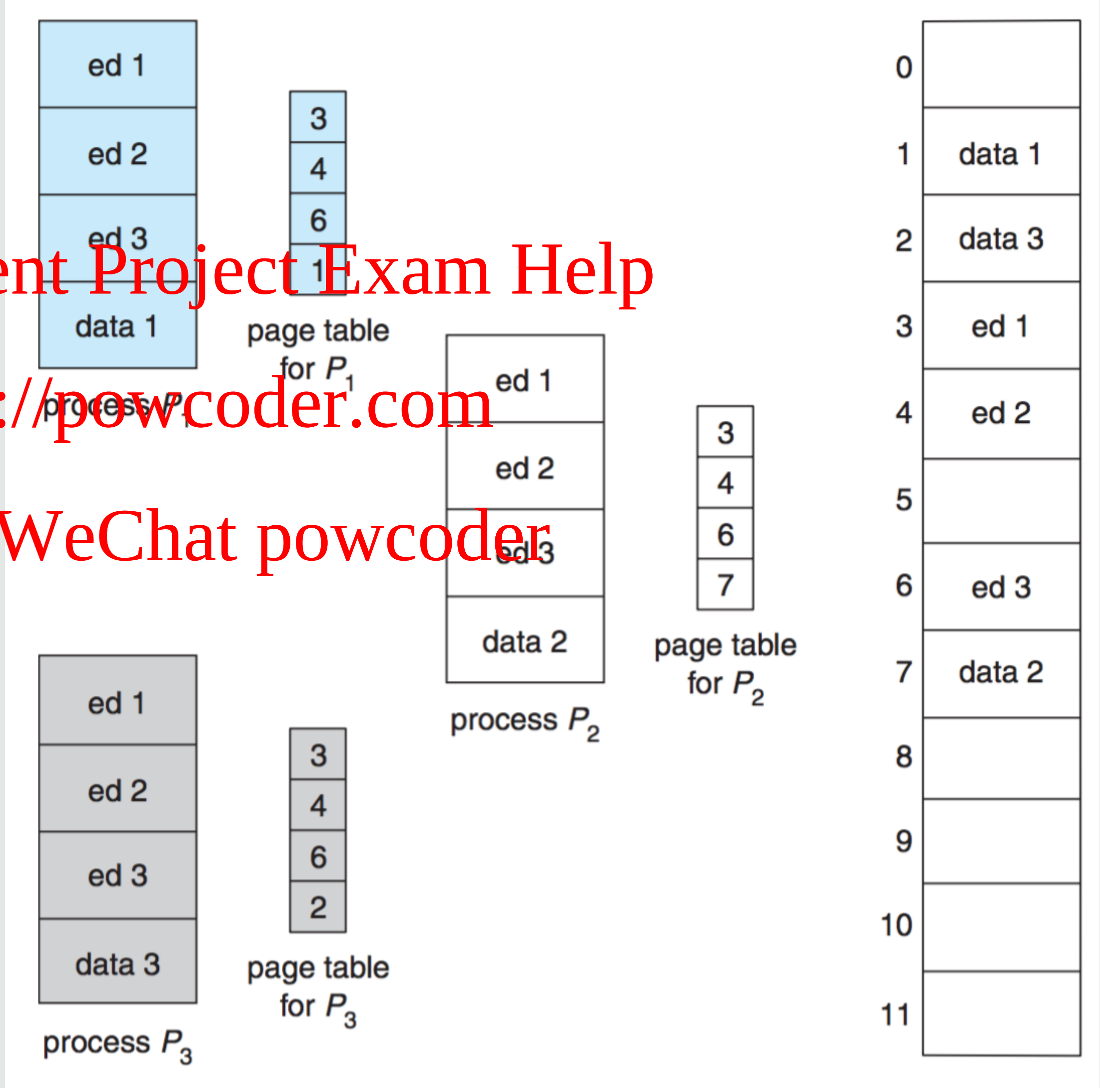
Page Faults and Thrashing

Thrashing

- Performance degradation by high number of page faults, i.e. resident set is too small
- How many pages should be loaded per process?
  → working set

Shared pages

- Shared memory
- Shared libraries



Summary

Memory management

- Paging
  - Basis for modern virtual memory systems
- Page table implementations
  - Page tables can be large
  - Clever implementations are essential
- Hardware support in MMU
  (e.g. Translation Look-Aside Buffer)

Read

- Tanenbaum & Bos., Modern Operating Systems
  - Chapter 3
- Silberschatz et al., Operating System Concepts
  - Chapter 8

Homework

- Start reviewing the module content we have covered thus far.
- Finish remaining question sheets and exercises, if you haven't done so already.

Happy Easter!

Image source: spick.ch

Next Lecture

- Introduction
- Operating System Architectures
- Processes
- Threads - Programming
- Process Scheduling - Evaluation
- Process Synchronisation
- Deadlocks
- **Memory Management (continued)**
- **File Systems**
- Input / Output
- Security and Virtualisation