Operating Systems

Lecture 7a

Dr Ronald Grau School of Engineering and Informatics Spring term 2018

Previously

Deadlocks

- Methods for handling deadlocks
  - Deadlock prevention
  - Deadlock avoidance
  - Deadlock detection and recovery

Today

Memory management

- Addressing and address spaces
- Partitioning and segmentation

Recap: Process abstraction

Process: program in execution

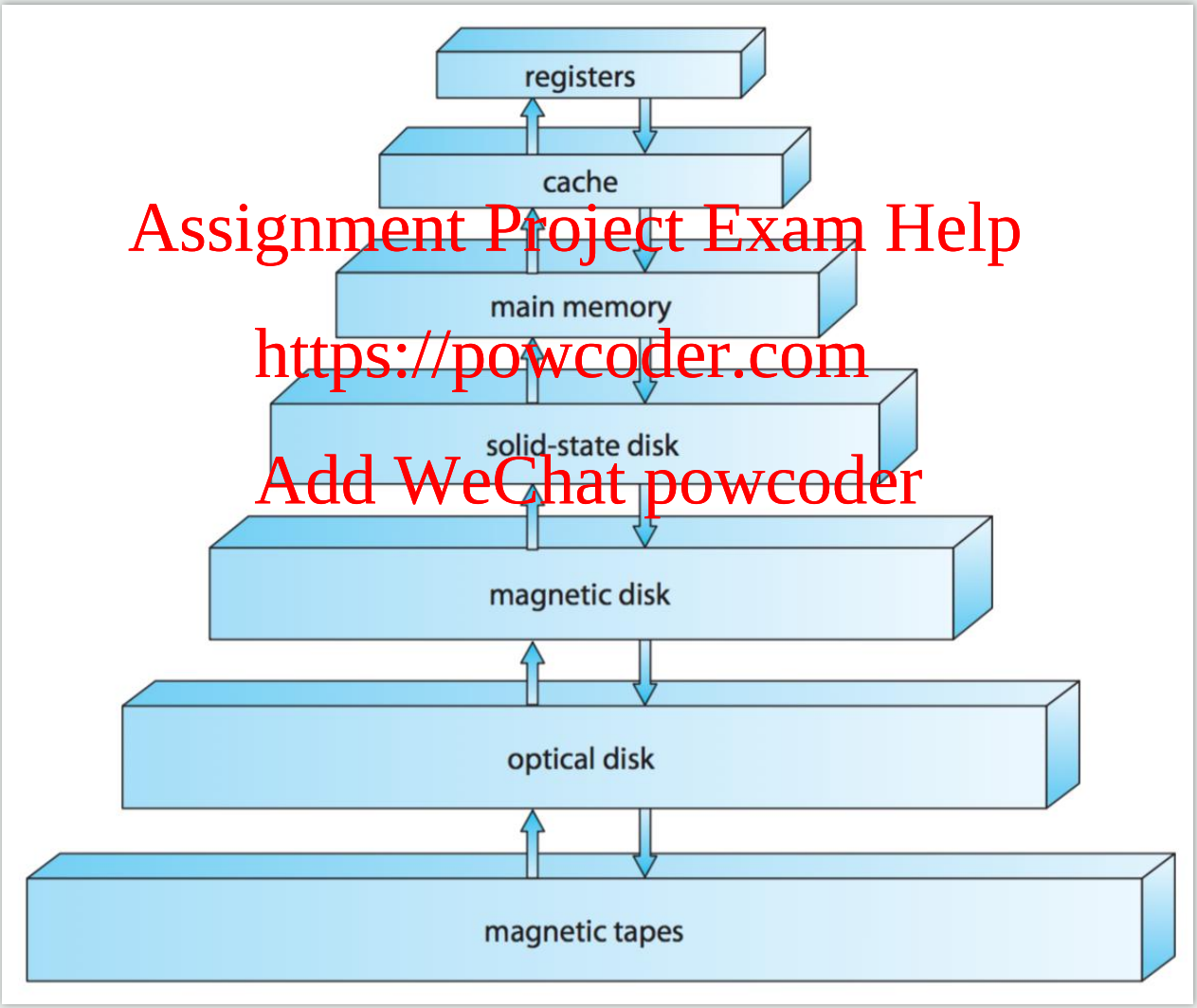
- Unit of resource management
  - Process image: code, heap, stack, les, I/O devices, . . .
  - Protection
- Threads: unit for scheduling
- Scheduling algorithms
- Communication between processes
  - Shared memory, . . .
- Synchronisation and deadlocks

Recap: Process abstraction

Multiprogramming

- Multiple processes run concurrently
- Each works under the assumption that it has sole access to the CPU and unbounded memory resources

Recap: Memory Hierarchy



Recap: Virtual Memory Abstraction

Virtual Memory

- Memory usage regardless of actual organisation and size of physical memory
- Each process has its own address space

Operating system duties:

- Address translation
- Memory management:
  - CPU can only address main memory
  - Data is only partially in memory (paging, swapping)
- Memory protection
  - Process can only access its “own" memory
- Provision of shared memory

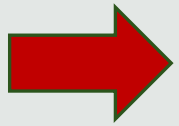
Memory addressing

CPU-Memory Interaction

- Operations and data are stored here
- Data and instructions loaded from memory into CPU
- Memory locations are accessed through addresses
- Bit-width of addresses (e.g. 32 bits) determines the size of addressable memory

E.g. addresses assigned when compiling code:

```
if(x!=0)
    x++;
```

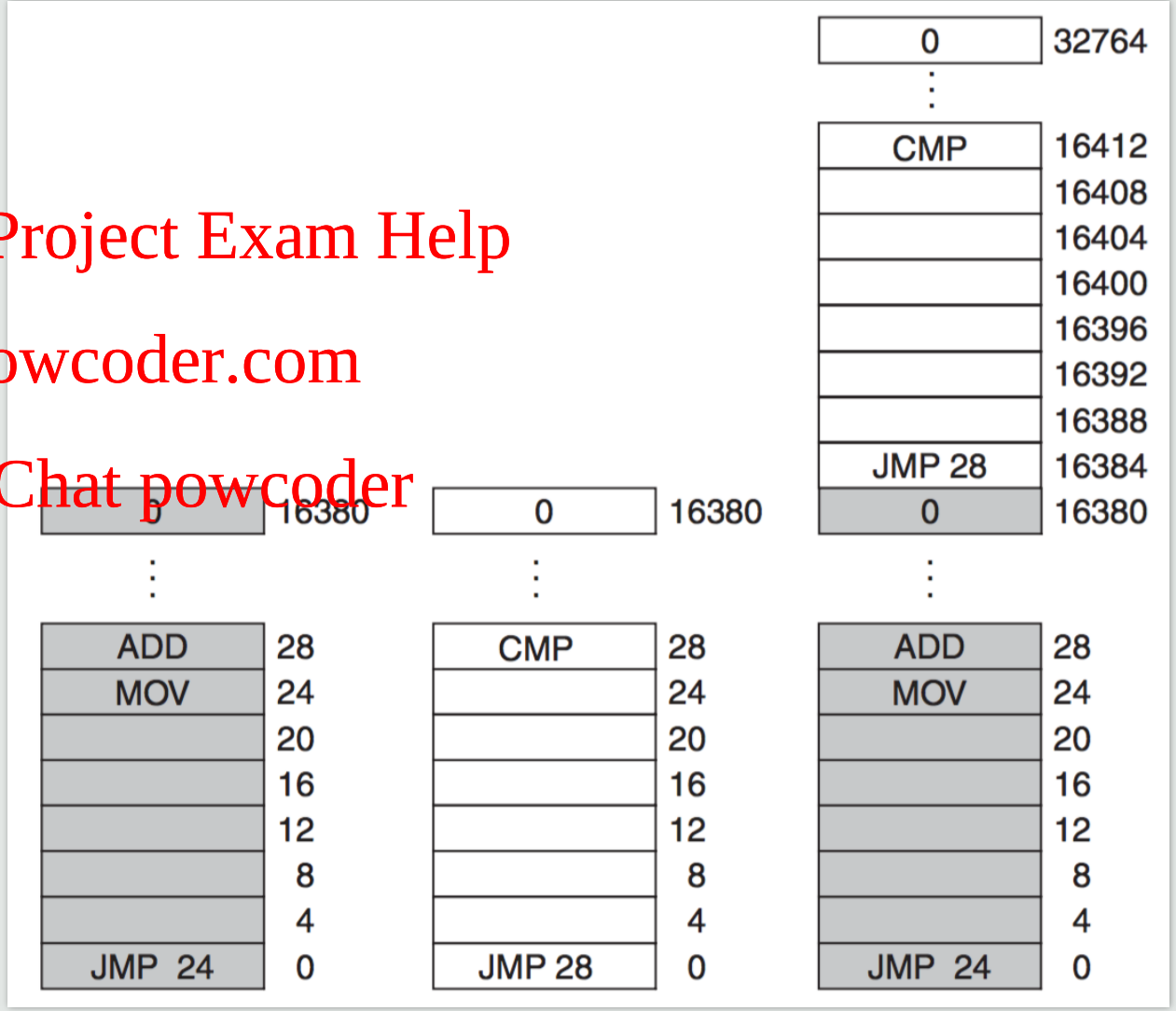


```
0x04: MOV 0x90, AX
0x06: JMZ AX, 0x0C
0x08: INC AX
0x0A: MOV AX, 0x90
0x0C: ...
```

Memory addressing

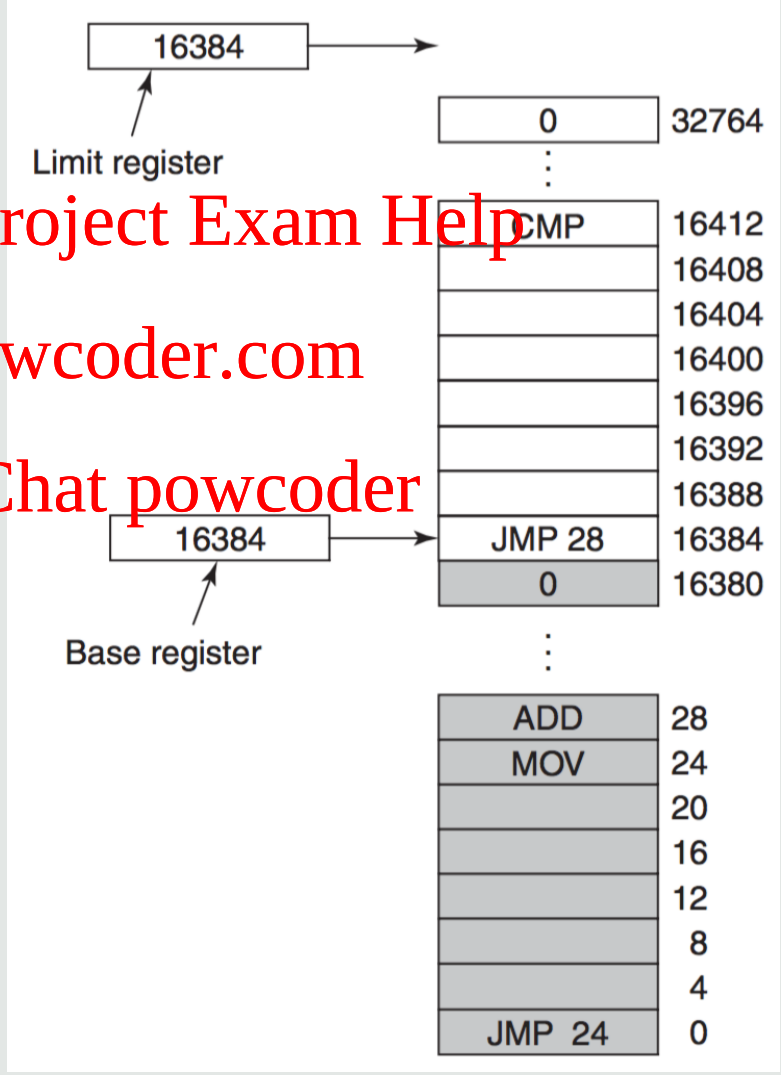
Absolute addressing

- Relocation problem



Memory addressing

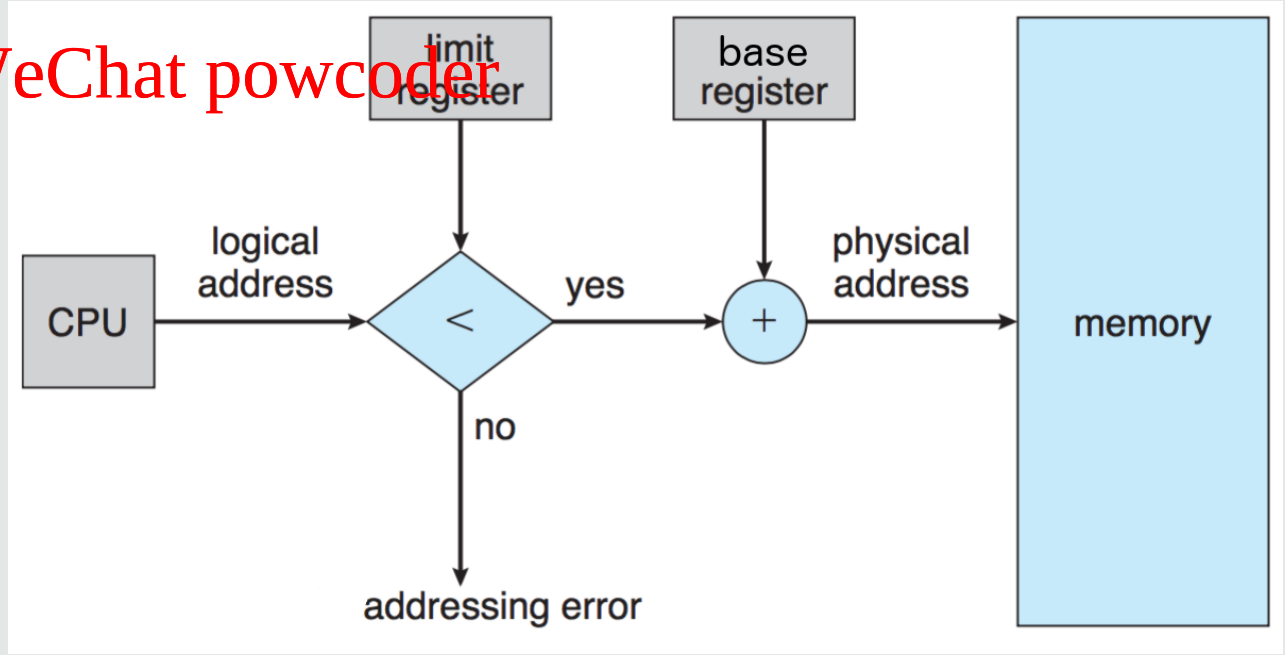
Dynamic relocation



Memory addressing

Logical addressing

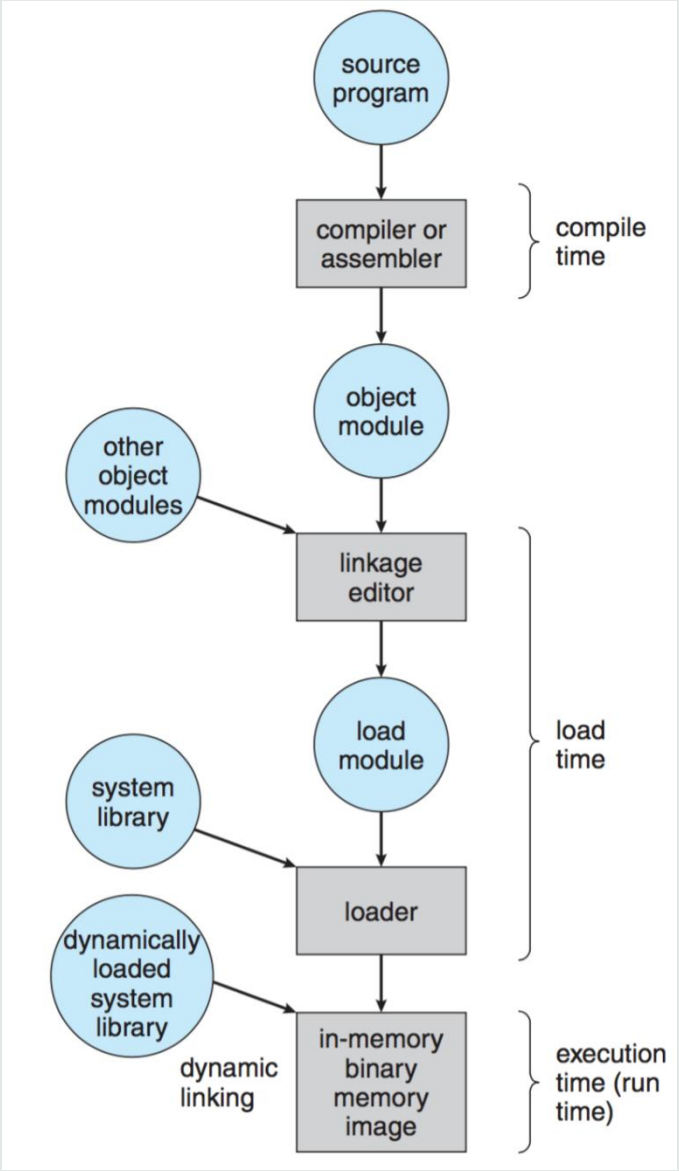
- Logical address
  - "Offset" relative w.r.t. a base address
  - Limit: for protection
- Address Translation
  - E.g. by MMU:



Address binding

Bind instructions and data to memory addresses

- Compile time
  - If memory location known a priori
  - Must recompile code if starting address changes
- Load time
  - Must generate relocatable code
  - I.e. information (relocation pointers) allowing the loader to adapt addresses
- Execution time
  - Binding delayed until run time
  - Process can be moved during its execution within memory
  - Requires hardware support (MMU)

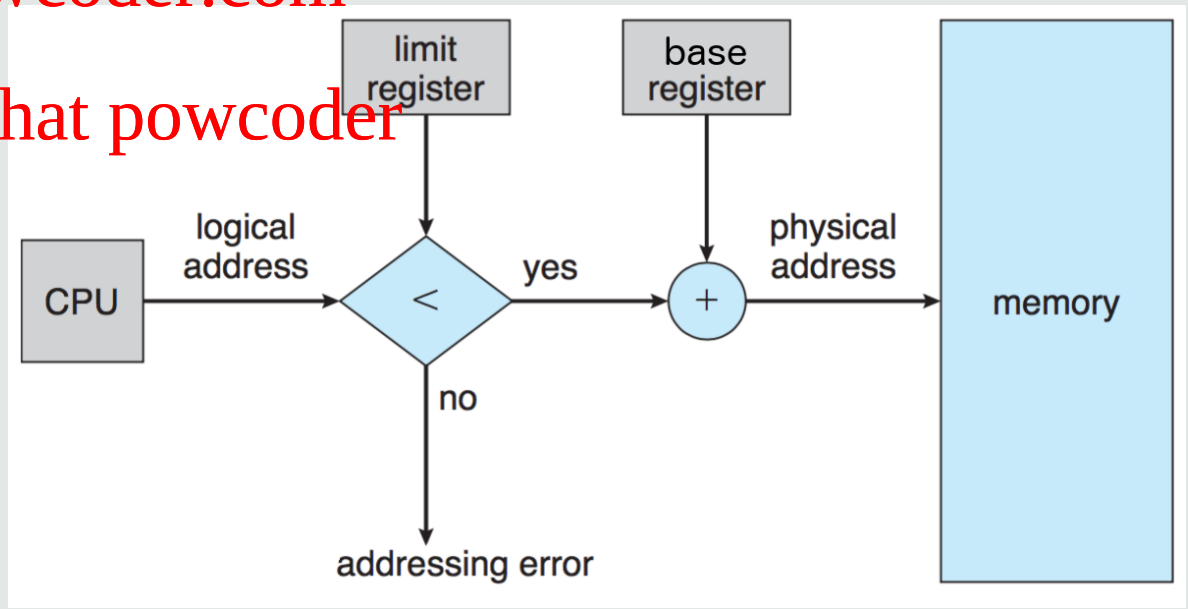


Static and dynamic linking

- Static Linking
  - Libraries and program code combined into the executable
  - No OS support required
- Dynamic Linking
  - Linking postponed until execution time
  - Useful for shared libraries (.so, .dll)
  - OS support by dynamic linker
  - Dynamic linker is invoked when a library function is called
    - Loads the library (if not loaded yet)
    - Maps library into the process' address space
    - Binds addresses of calls to the library functions

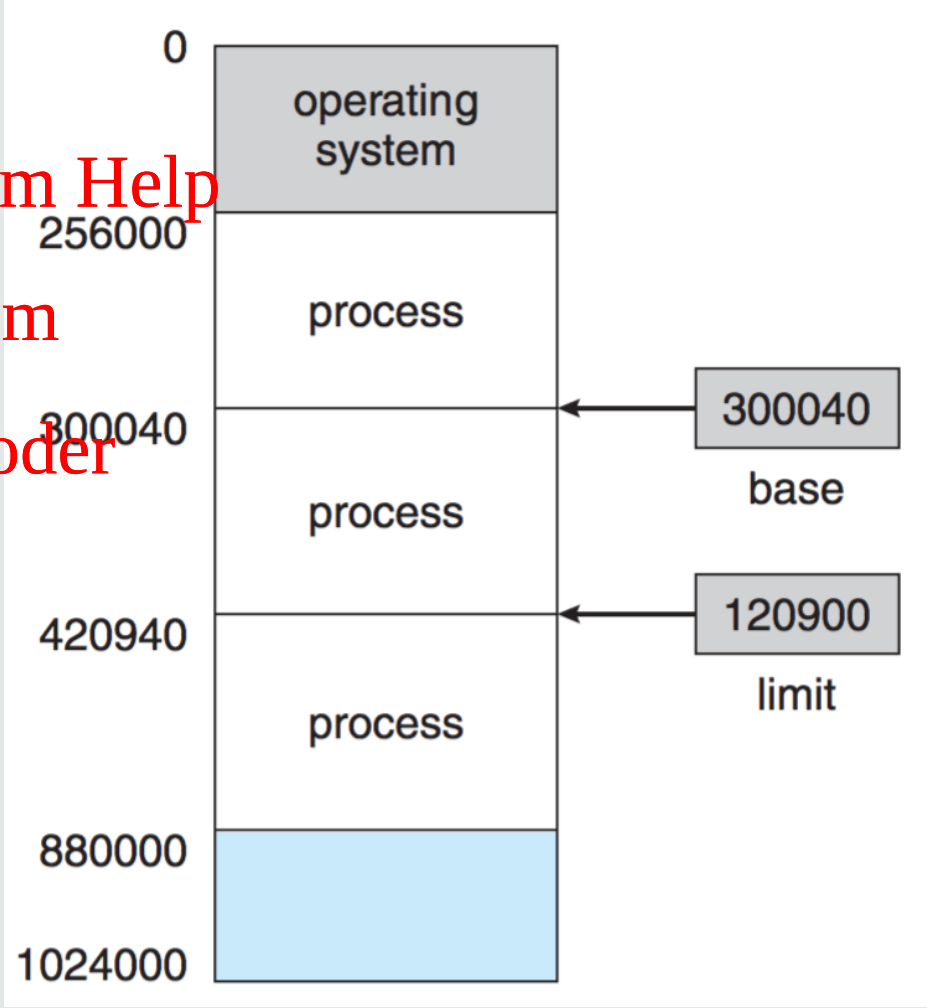
Memory Management Unit

- Hardware device that at run time maps logical to physical addresses
- Performs execution-time binding of addresses
- Many methods possible. Simplest scheme: base address + offset
- Implements memory protection



Logical and Physical Address Spaces

- Logical address space
  - Addresses in the code being executed
  - Addresses sent from CPU to MMU
- Physical address space
  - Addresses output by MMU
  - Actual addresses in main memory



Memory Allocation

- Addressing
  - Relocation
  - Logical addresses
  - Protection
- **Memory allocation**
  - Partitioning: contiguous memory partition for each process
  - Segmentation: process memory subdivided into segments
  - Paging: process memory subdivided into pages
- Aspects to consider:
  - Fragmentation
  - Swapping

Memory Partitioning

Fixed-size partitions

- Memory divided into fixed-size partitions
- A partition is a contiguous section of memory
- Process image resides in memory as one single chunk
- Process is assigned a free partition
- Easy to implement with base address and offset

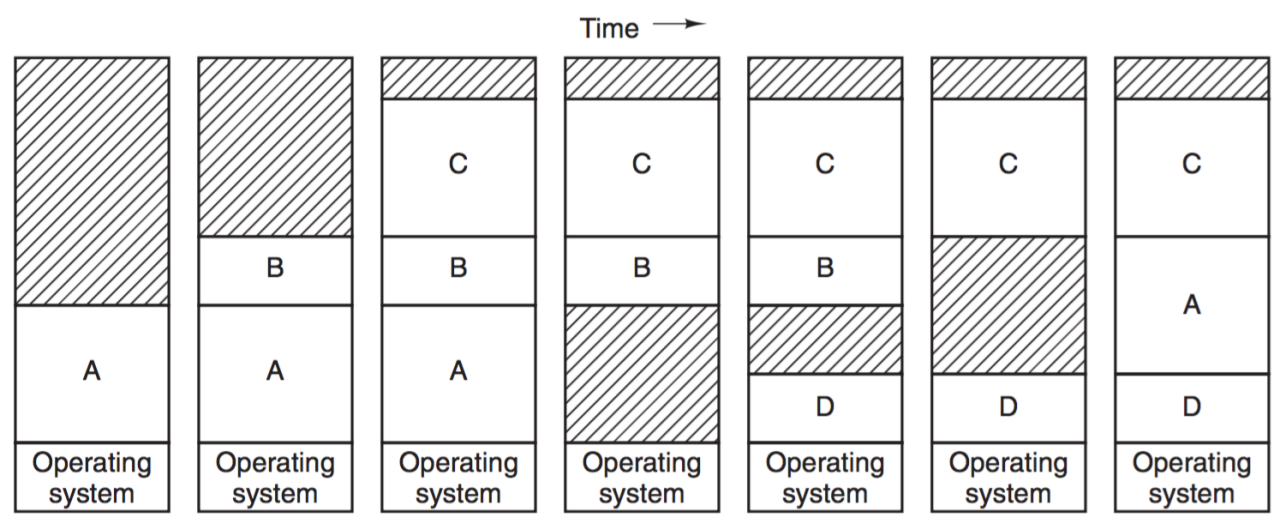
Disadvantages

Maximum number of processes in memory is fixed.
Maximum size of process images is limited.
Internal fragmentation: There may be unused memory in the assigned partitions.

Memory Partitioning

Variable-size partitions

- Process image resides in memory as one single chunk
- Process is allocated a free partition of required size
- On termination the partition is freed and combined with adjacent free partitions
- OS maintains information on allocated and free partitions



Memory Partitioning

Dynamic Storage-Allocation Problem

- Which free partition to use for a process of size n?
- First-fit
  - Allocate the first free section that is big enough
- Best-fit
  - Allocate the smallest free section that is big enough
  - Must search entire list, unless ordered by size
  - Produces the smallest leftover free section
- Worst-fit
  - Allocate the largest free section
  - Must search entire list, unless ordered by size
  - Produces in the largest leftover free section (Is this useful?)

Memory Partitioning

External fragmentation

- Total memory space exists to fit a process, but it is not contiguous

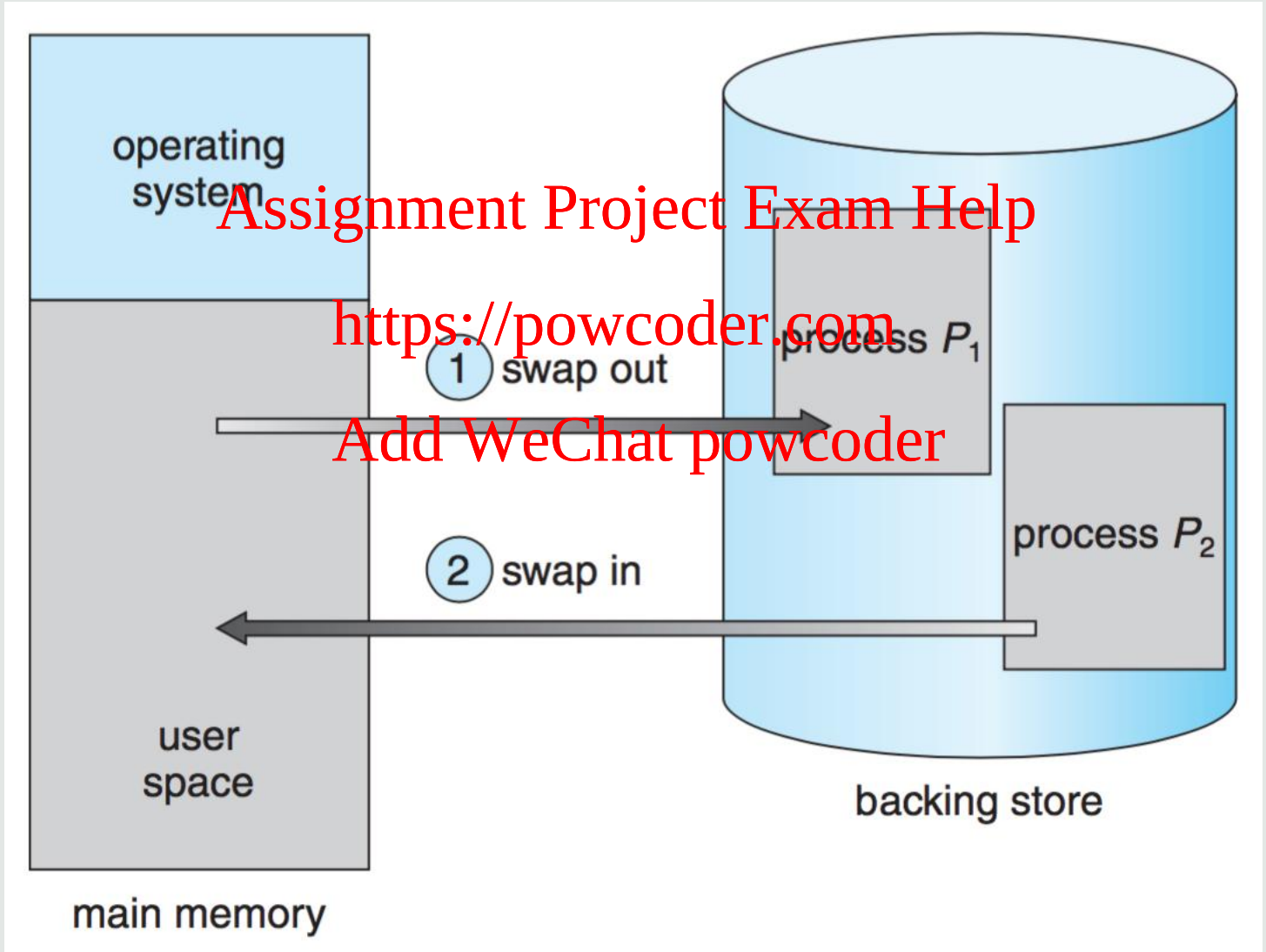
- Compaction:

  - Move processes to eliminate small free partitions

  - Requires execution-time address binding
  - I/O problem:
    - Keep job in memory while waiting for I/O
    - Use double-buffering to decouple I/O from process image

What happens when a process is to be run but there is insufficient memory available?

Swapping



Memory Partitioning

Summary

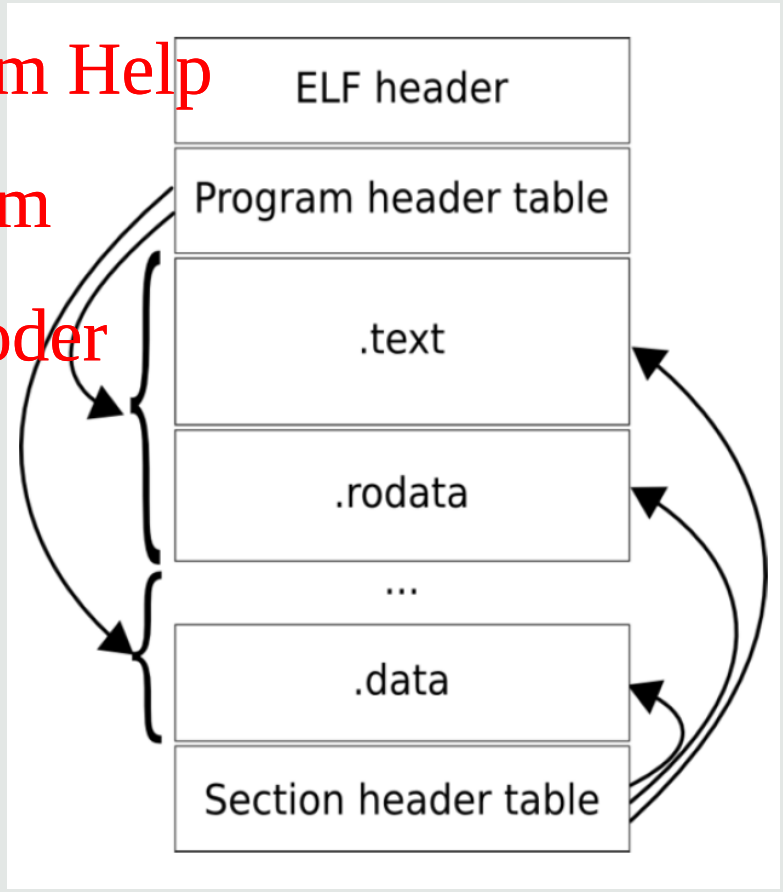
- Advantages
  - Enables multiprogramming
  - Memory protection between processes
  - Easy to implement
- Disadvantages
  - Limit maximum size of process image
  - No memory protection within processes
  - Suffers from internal/external fragmentation

Memory Segmentation

Process image subdivided in segments

- Compiler can create separate segments for:

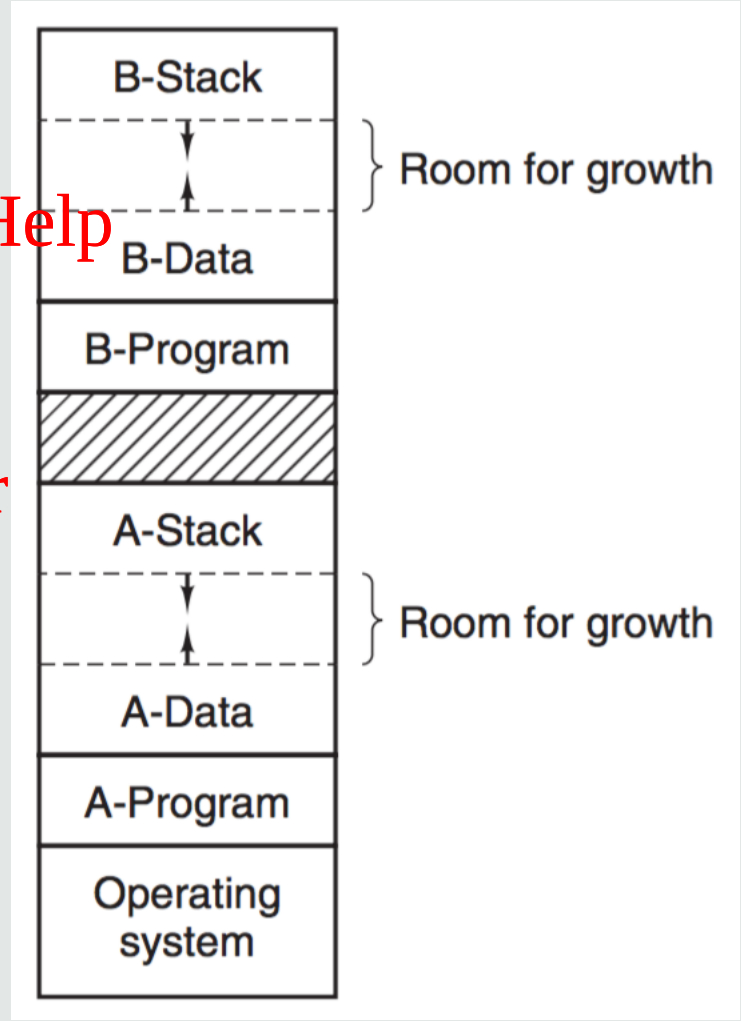
  E.g., for
  - The code
  - Global variables
  - The heap, from which memory is allocated
  - The stacks used by each thread



Memory Segmentation

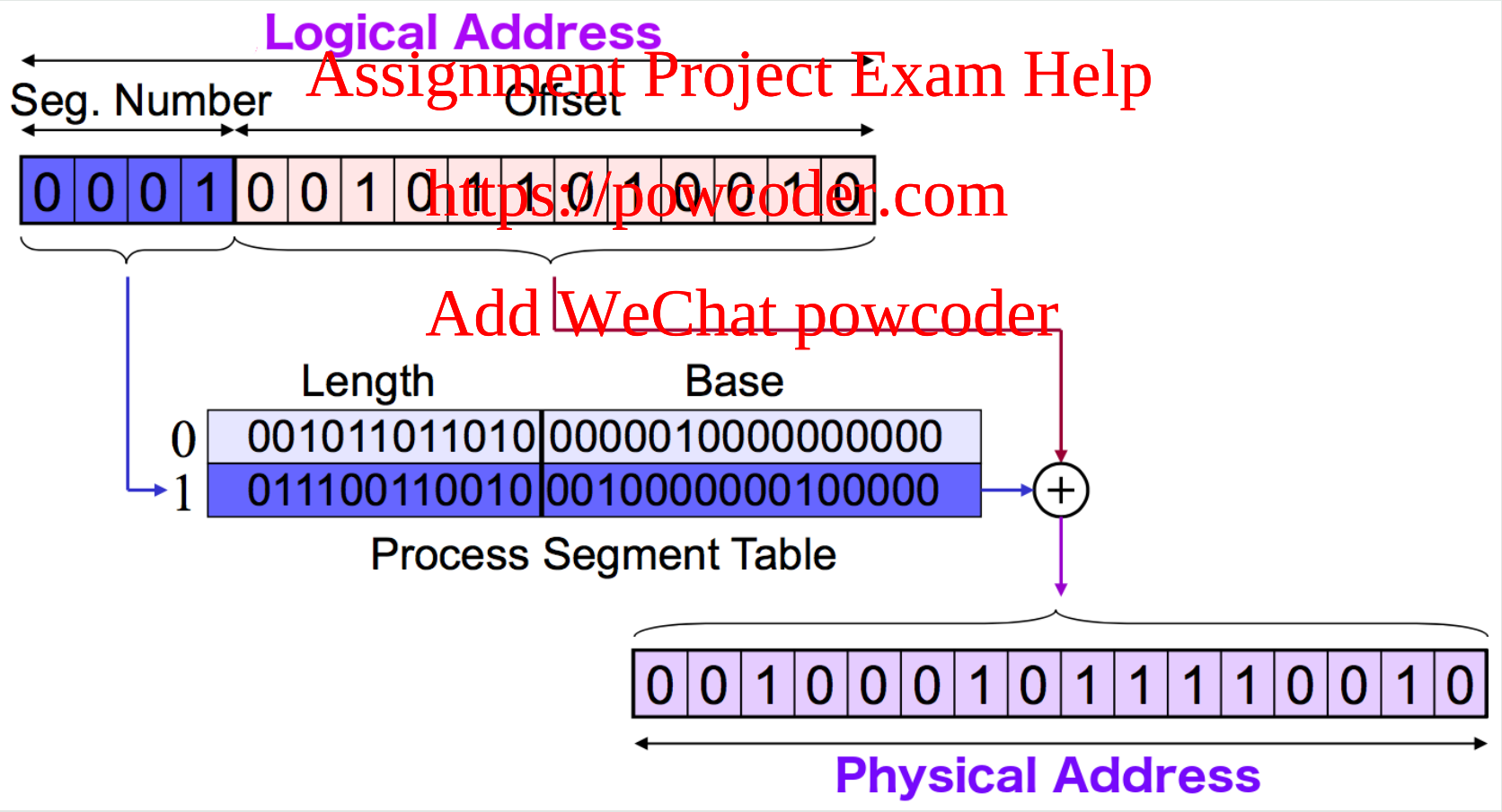
Segment table

- Indexed by segment numbers
  - Segment base
  - Segment limit
  - Access bits (read, write, execute)



Memory Segmentation

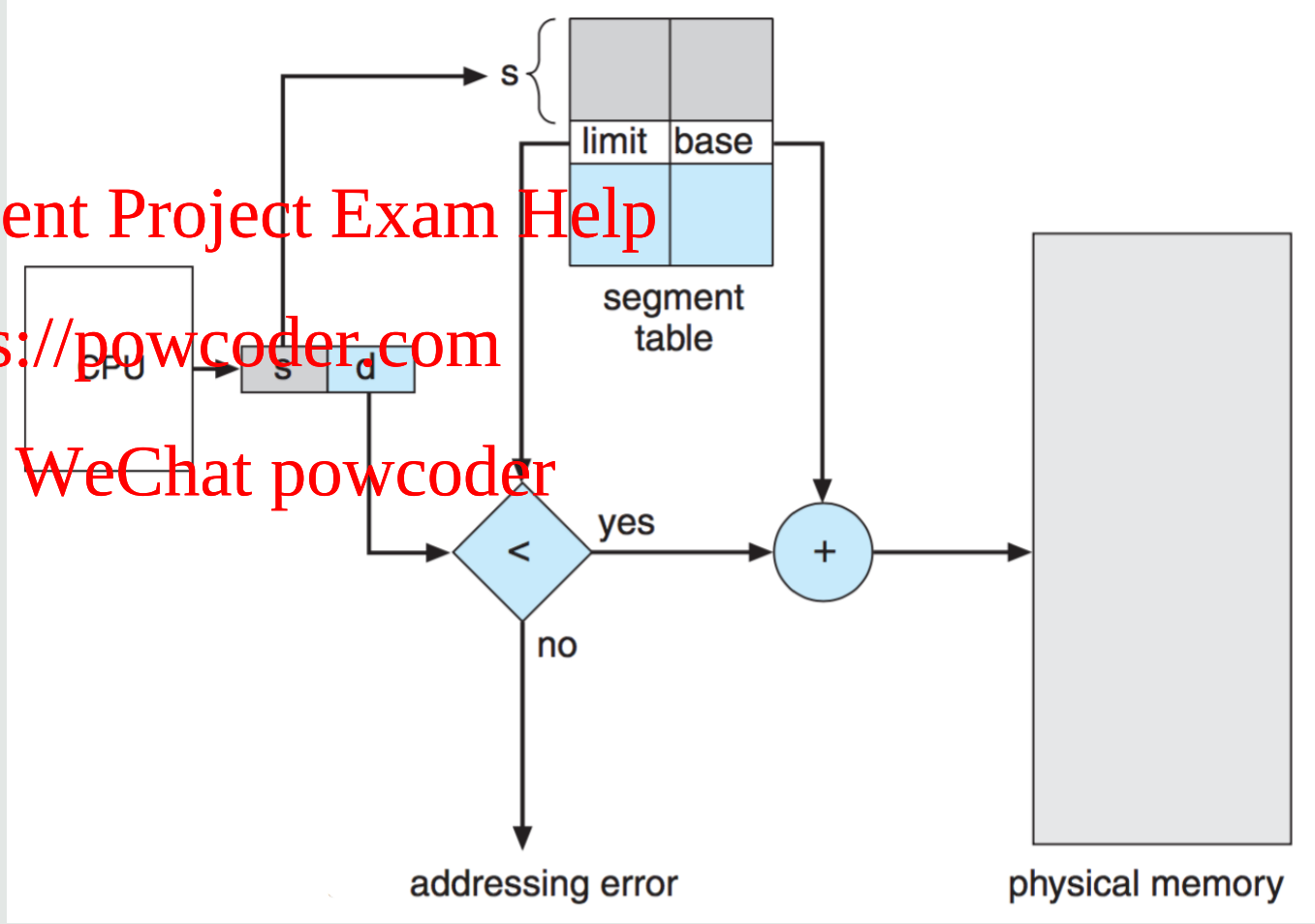
Segment address translation



Memory Segmentation

Segment protection

- Offset (d) encoded in the logical address together with the segment number (s)
- Look up allowed limit (length) for this segment in the segment table
- Proceed addressing if offset (d) within limit bounds, else addressing error



Memory Segmentation

Summary

- Advantages
  - Non-contiguous allocation
  - Segments can be swapped in/out independently
  - Memory protection within segments of process memory
  - Less fragmentation
- Disadvantage
  - Limit maximum size of segments

Summary

Memory management

- Memory addressing
  - Absolute vs relative addresses
  - Physical vs logical addresses
  - Address translation
- Aspects to consider
  - Relocation
  - Dynamic linking
  - Fragmentation
  - Swapping
- Memory allocation concepts
  - Partitioning
  - Segmentation
  - Paging (next lecture)

Read

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

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