CSC1107 - OPERATING SYSTEMS

LECTURE 8: MEMORY MANAGEMENT STRATEGIES

Asst. Prof Cao Qi
Qi.Cao@Glasgow.ac.uk

WORLD
CHANGERS
WELCOME

University of Glasgow

School of Computing Science

Acknowledgement

- Contents of CSC1107 - Operating Systems derived from:
  - **COMPSCI4011 – Operating Systems (H)**, Univ. of Glasgow.

    Acknowledgement:

    Dr. Wim Vanderbauwhede, UoG, UK.

    Dr. Lito Michala, UoG, UK.

    Dr. Poh Kok Loo, SIT, Singapore.
  - **Operating System Concepts.** Authors: Silberschatz, Galvin and Gagne. Publisher: John Wiley & Sons.

    Acknowledgement: Authors and Publisher.
  - **Operating Systems Foundations with Linux on the Raspberry Pi.** Authors: Wim Vanderbauwhede and Jeremy Singer. Publisher: Arm Education Media.

    Acknowledgement: Authors and Publisher.

Lecture Contents

- ◆Logical vs. Physical Address Space.
- ◆Swapping.
- ◆Contiguous Memory Allocation.
- ◆Fragmentation.
- ◆Paging.
- ◆Structure of the Page Table.

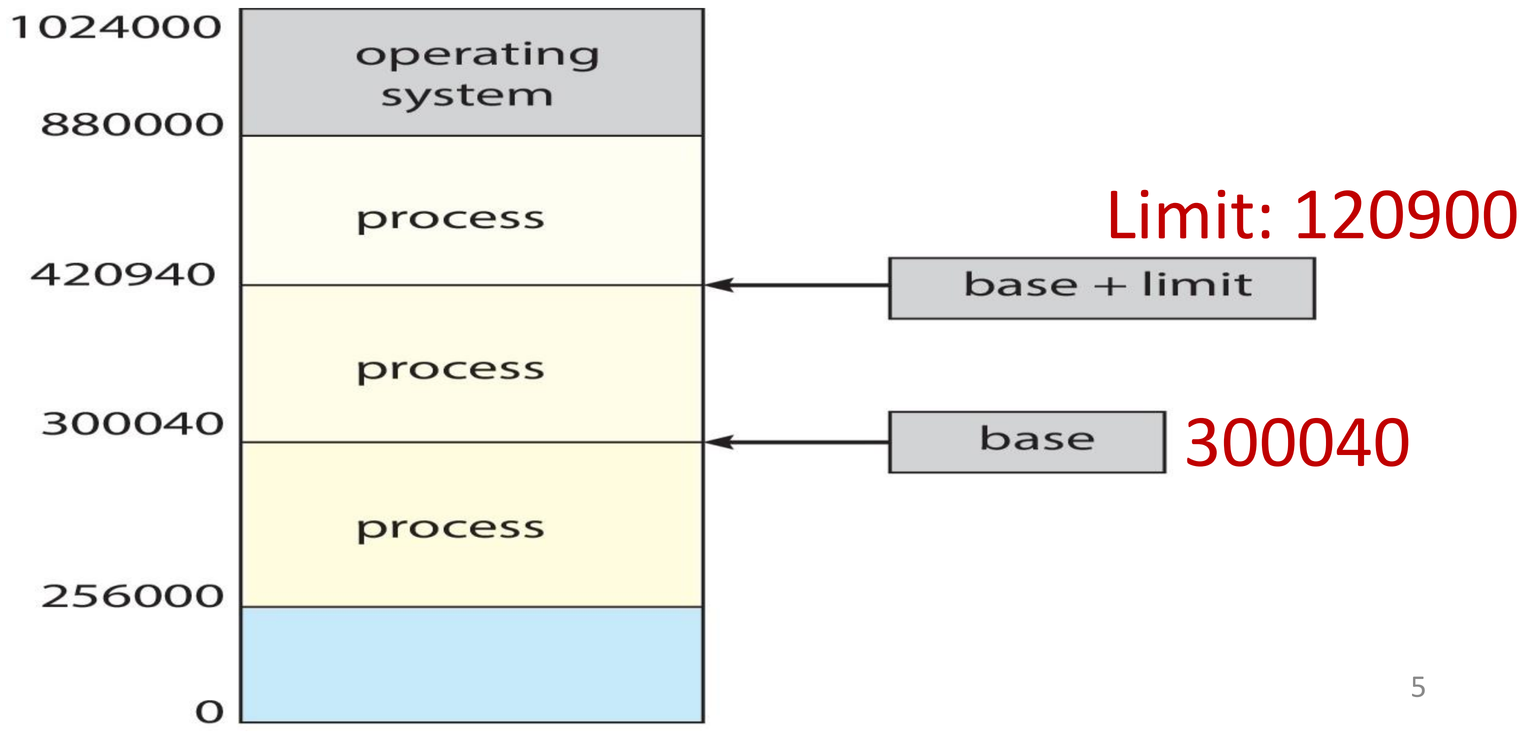
Background

- Memory only sees a stream of:
  - addresses + read requests, or
  - address + data and write requests.
- Register access is done in about 1 CPU clock.
- Main memory may take many cycles.
- **Cache** sits between main memory and CPU registers.

University of Glasgow | School of Computing Science

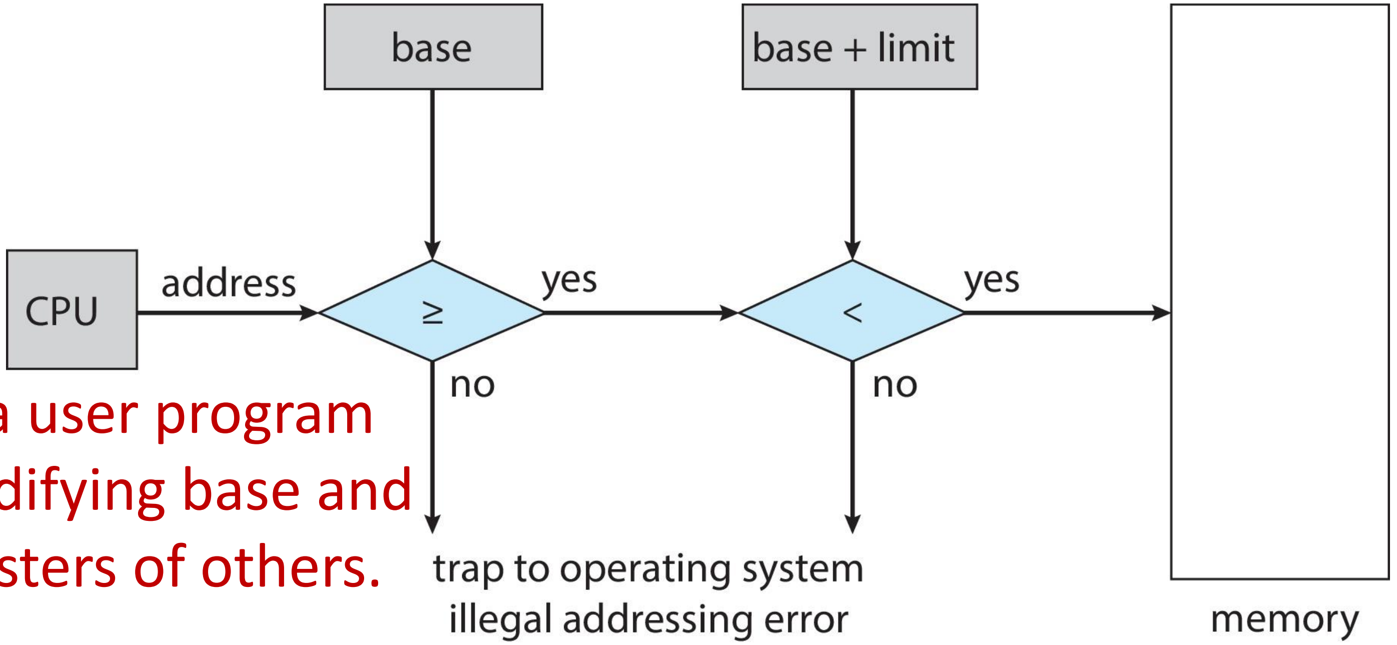
Base and Limit Registers

- Ensure a process only access its address space.
- Protection using **base** and **limit registers** in logical address space for each process.



Hardware Address Protection

- CPU must check every memory access in user mode, ensure within base and limit for users.



Prevent a user program from modifying base and limit registers of others.

- Privileged instructions to load base and limit registers by OS in kernel mode.

University of Glasgow

School of Computing Science

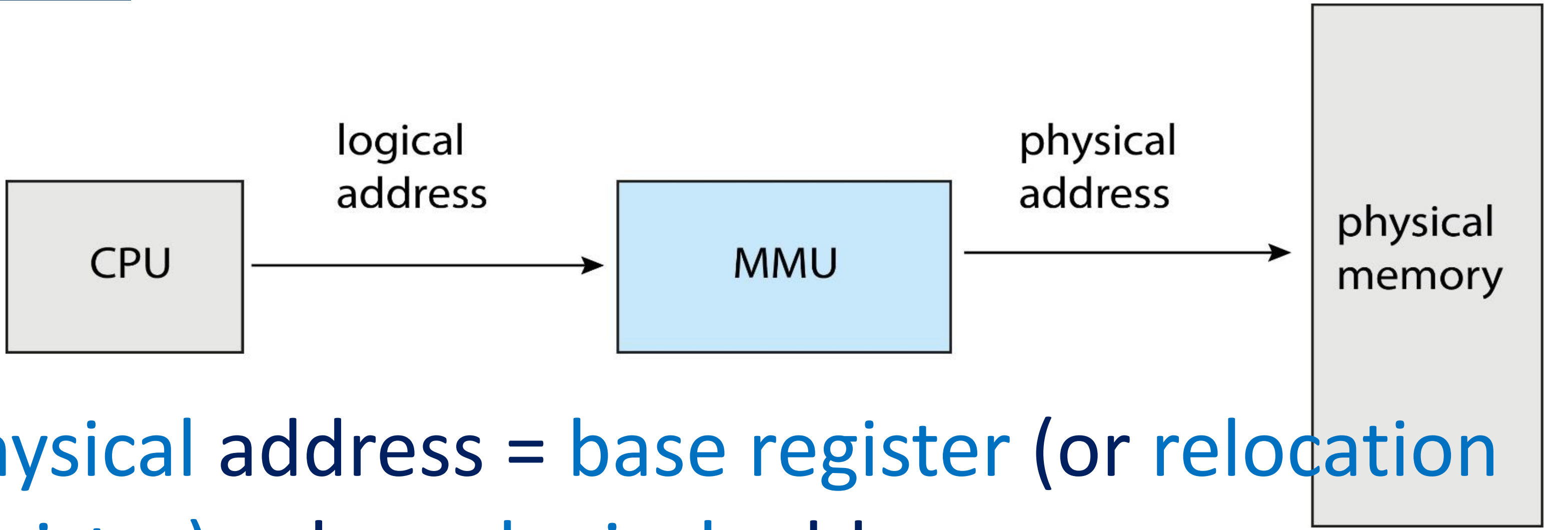
Logical vs. Physical Address Space

- **Logical address** (virtual address)**:** seen and operated by CPU.
- **Physical address:** seen and operated in physical memory.
- **Logical address space**: set of all logical addresses.
- **Physical address space**: set of all physical addresses.

University of Glasgow

School of Computing Science

Memory-Management Unit (MMU)

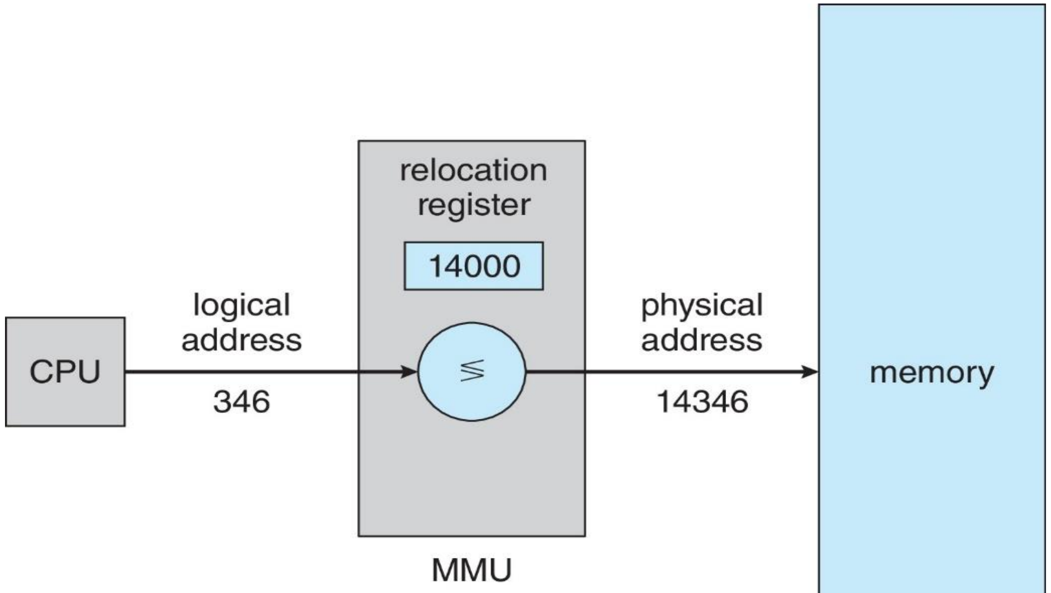


- physical address = base register (or relocation register) value + logical address.
- User programs deal with logical addresses; never sees the real physical addresses.
- Logical address (from 0 to max) bound to physical addresses (from R+0 to R+max); R=base register (or relocation register).

University of Glasgow

School of Computing Science

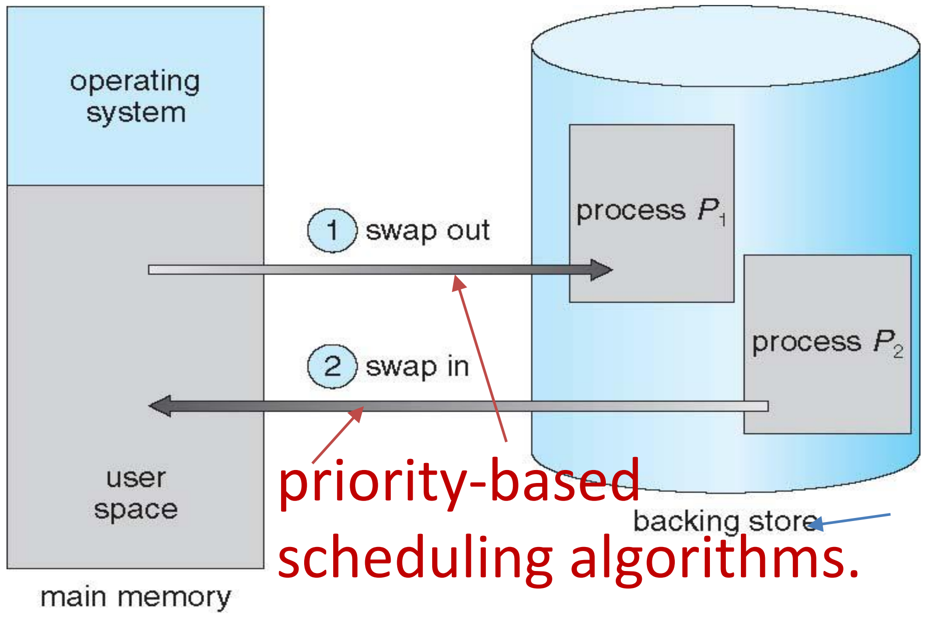
Dynamic Loading



- Size of a process is limited to size of physical memory.
- Dynamic loading: Routine is not loaded until it is called.
- Better memory-space utilization.
- When a routine needs to call another routine, it first checks if that routine is loaded. If not, load the desired routine into memory next.

Swapping

- Swapping: A process swapped out of memory; then brought back into memory for continued execution.
- Process memory space may exceed physical memory.
- Major swap time: transfer time, proportional to amount of memory space swapped.



priority-based scheduling algorithms.

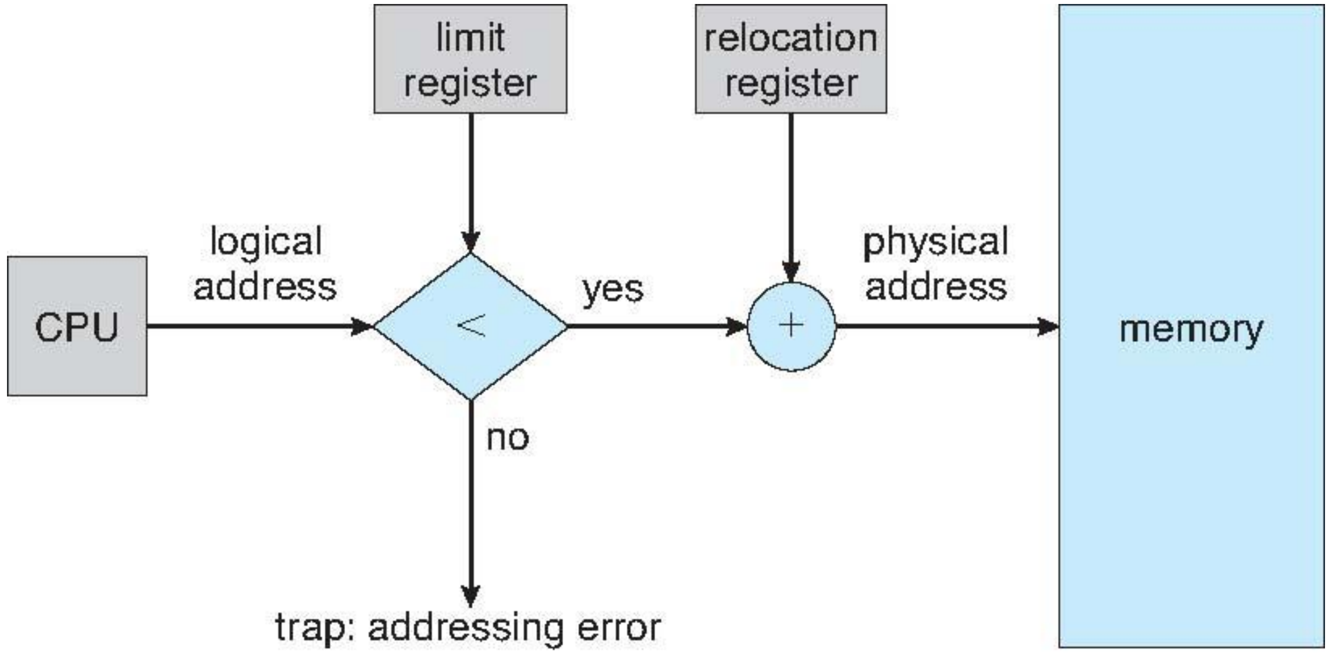
- 100 MB process, with 50 MB/second transfer time.
- Swap time = 2 seconds x 2 (swap out and in).

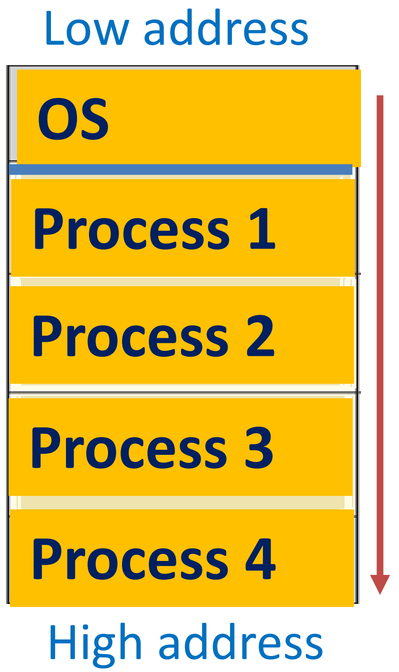
University of Glasgow

School of Computing Science

Contiguous Allocation

- Contiguous allocation is one early method.
- Main memory divided into two partitions:
  - OS in low memory
  - User processes held in high memory.

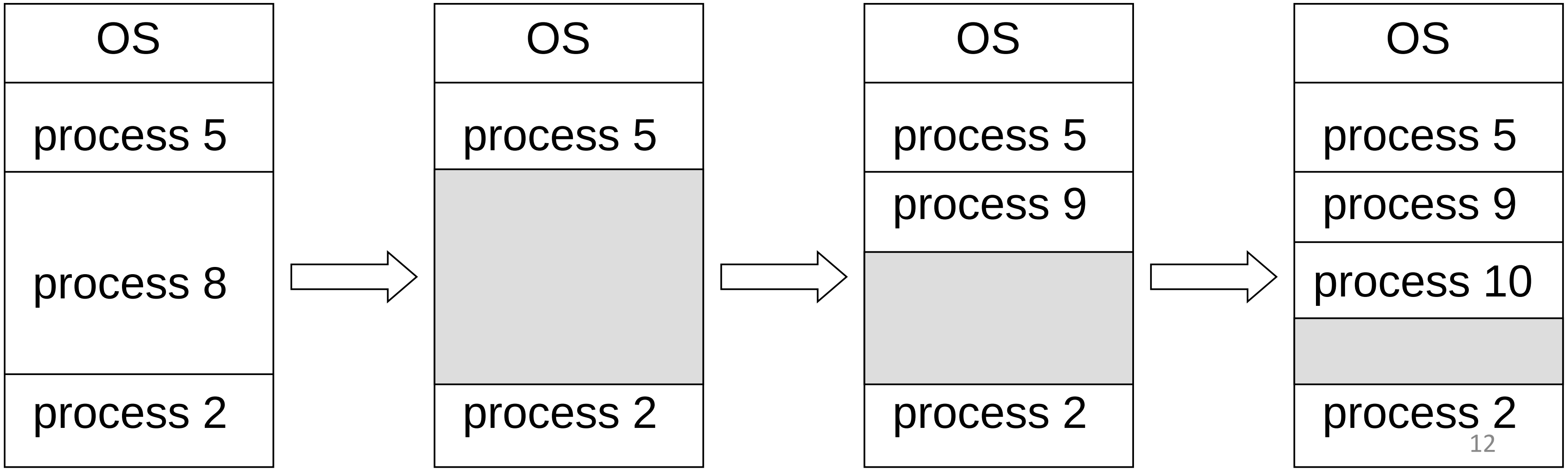




University of Glasgow | School of Computing Science

Variable Partition and Multiple-partition Allocation

- **Variable-partition** for efficiency (size = the needs of the process).
- **Hole:** available blocks; scattered in memory; but may or may not large enough to load a new process.
- 1) allocated partitions, 2) free partitions (holes), by OS



University of Glasgow School of Computing Science

Dynamic Storage-Allocation Methods

How to satisfy a process from a list of free holes?

- **First-fit**: Allocate the first hole that is big enough.
- **Best-fit**: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size. (Produces the smallest leftover hole).
- **Worst-fit**: Allocate the largest hole; must also search entire list (for the largest leftover hole).

First-fit and **best-fit** better than **worst-fit** in terms of speed and storage utilization.

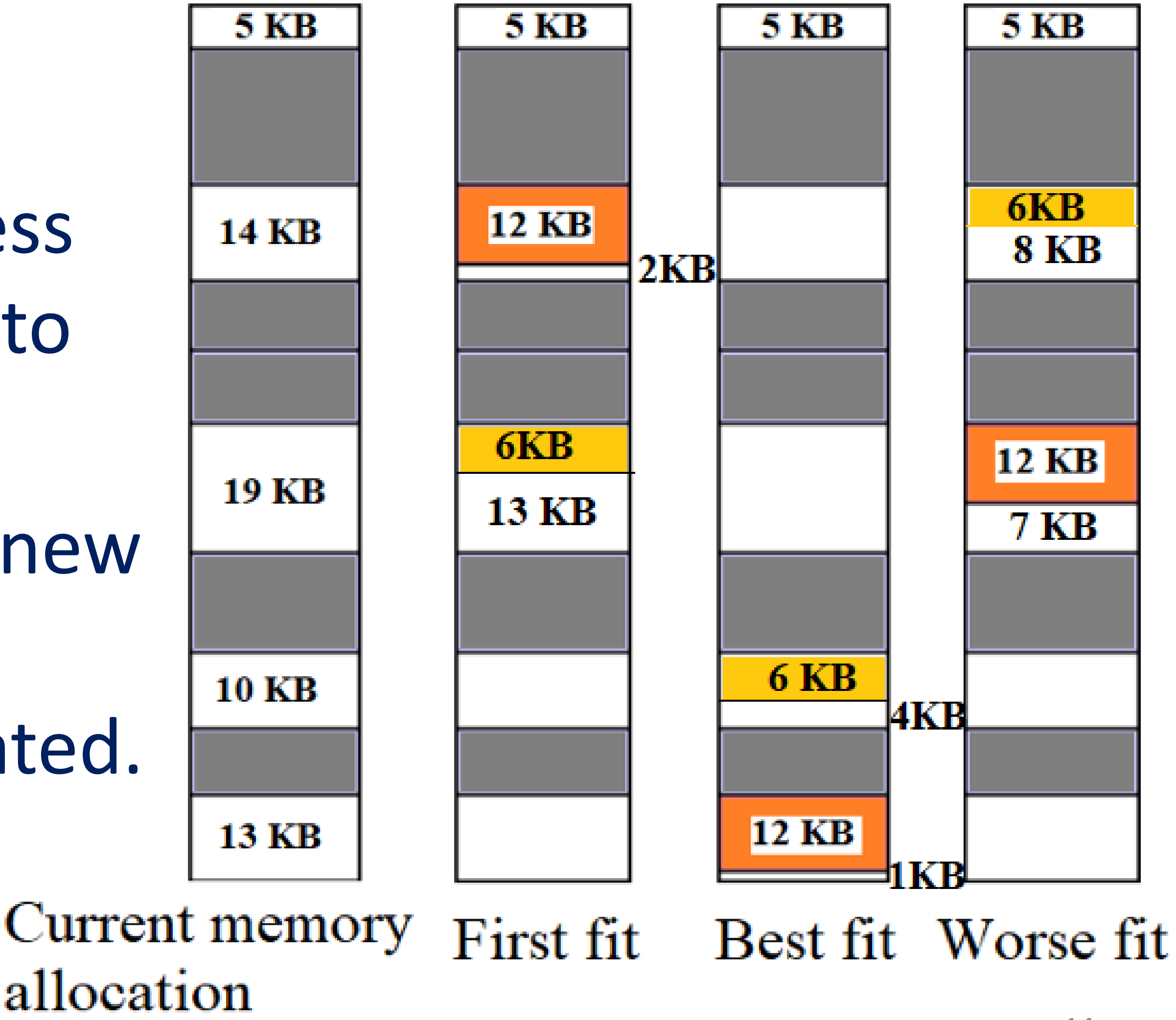
University of Glasgow | School of Computing Science

Three Storage-Allocation Methods

e.g.,

(1). if a new process with 12 KB needs to be allocated.

(2). Next another new process with 6 KB needs to be allocated.



University of Glasgow
School of Computing Science

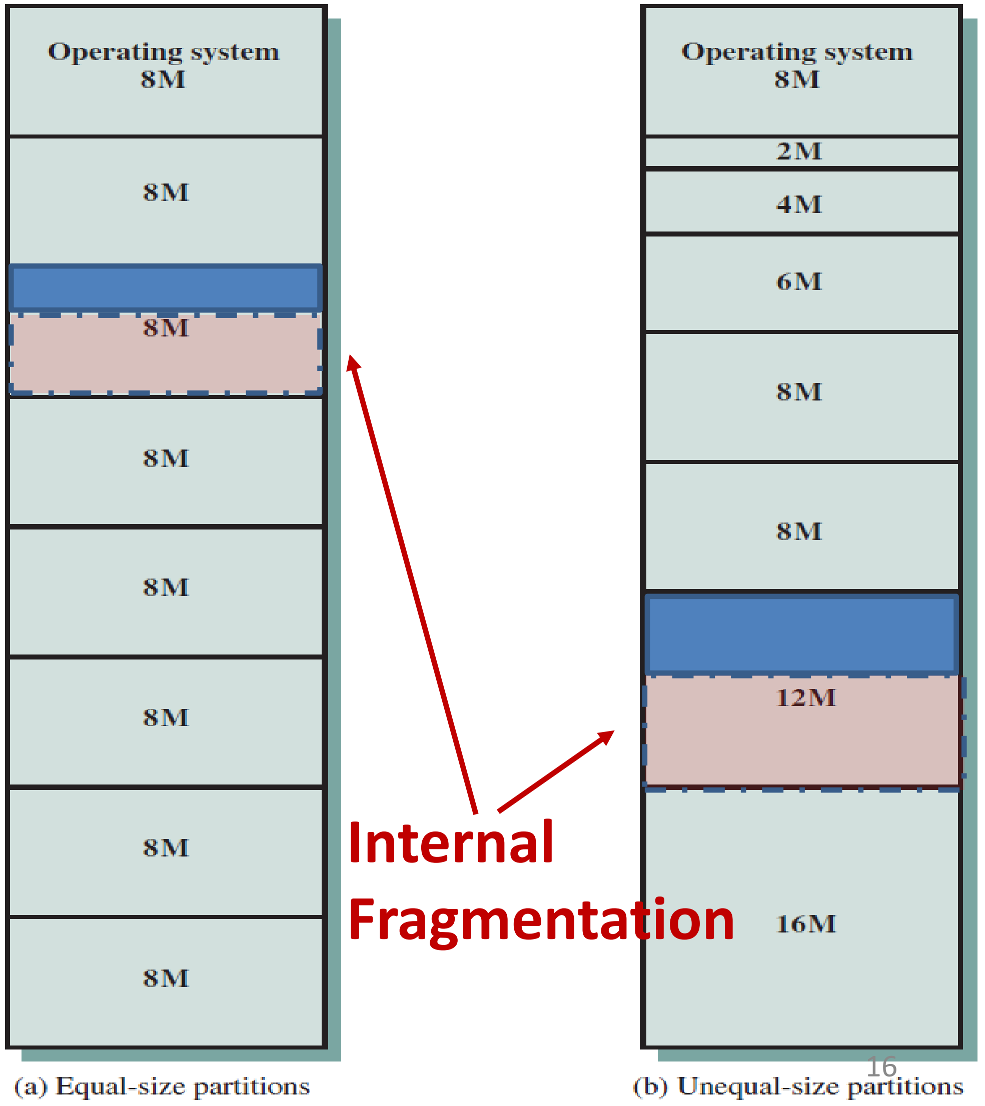
Fragmentation

- **External Fragmentation:** total memory space exists to satisfy a request, but not contiguous.
- **Internal Fragmentation:** allocated memory frames slightly larger than requested size; this size difference is memory internal to a partition, but not being used.
- First fit analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation.

University of Glasgow | School of Computing Science

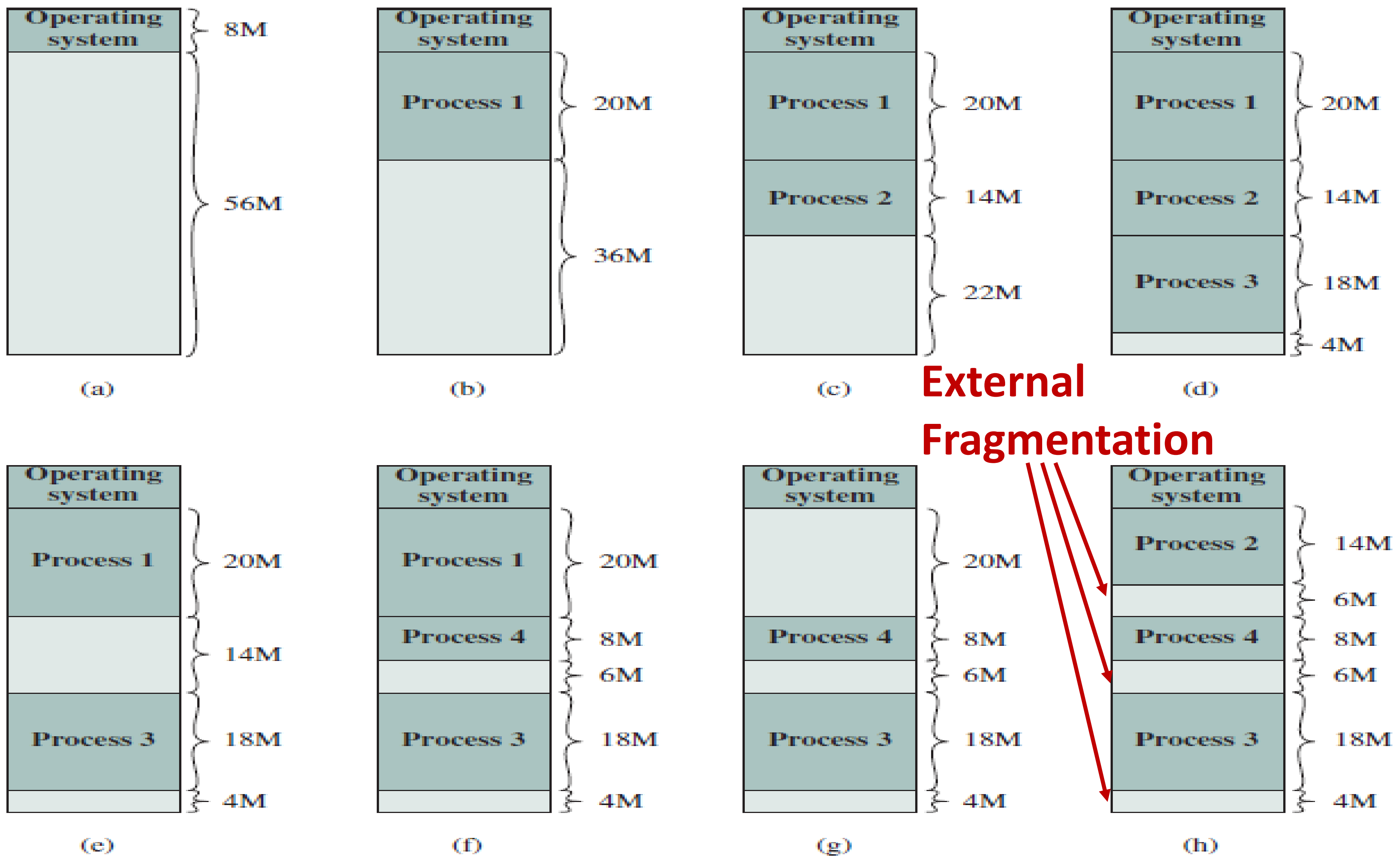
Fixed Partitioning

- Any program, no matter how small, occupies an entire partition.
- Other processes cannot use empty space in a partition.
- This is called internal fragmentation.



School of Computing Science

Dynamic Partitioning



Reduce External Fragmentation

- **Method 1:** compaction - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible *only* if relocation is dynamic.
  - If relocation is static, address is fixed; can't move.
- **Method 2:** Paging - permit logical address space of processes non-contiguous, allowing a process to be allocated to non-contiguous physical memory, wherever such memory is available.

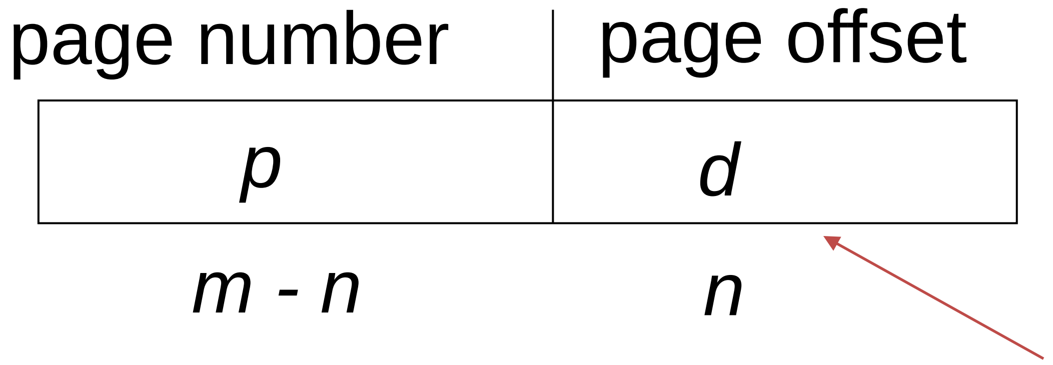
Paging

- **Frames**: divide physical memory into fixed-sized blocks (Size is power of 2).
- **Pages**: Divide process into blocks of same size.
- To execute a program of size ***N*** pages, need to find ***N*** free frames and load program.
- Keep track of all free frames. Set up a **page table** to translate logical to physical addresses.
- Cause internal fragmentation.

University of Glasgow School of Computing Science

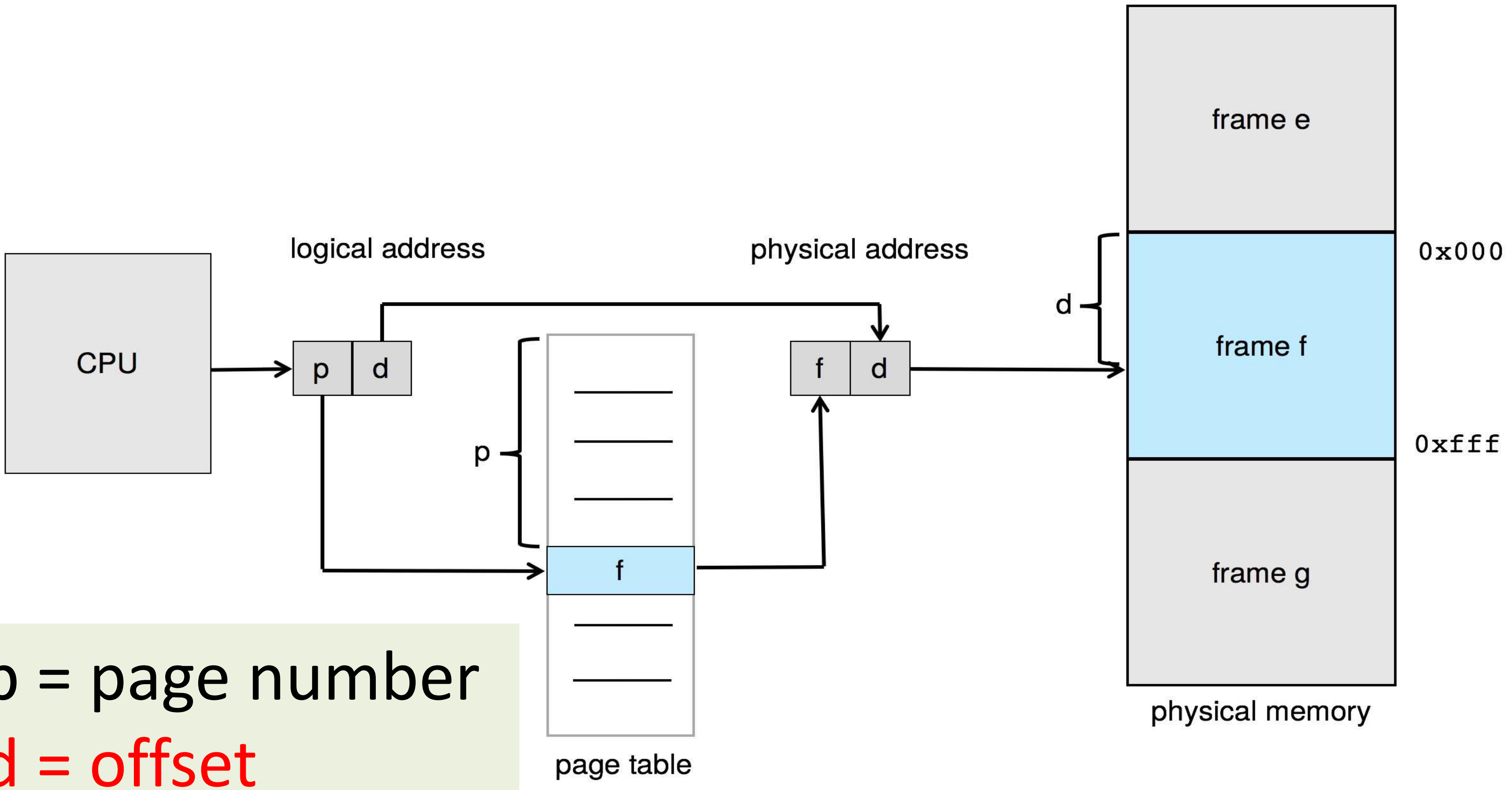
Address Translation Scheme

- Address is divided into:
  - **Page number (*p*)** – used as an index into a **page table**; number of pages in physical memory.
  - **Page offset (*d*)** – Size of each page.



  - For given logical address space 2^m and page size 2^n.

Paging Hardware

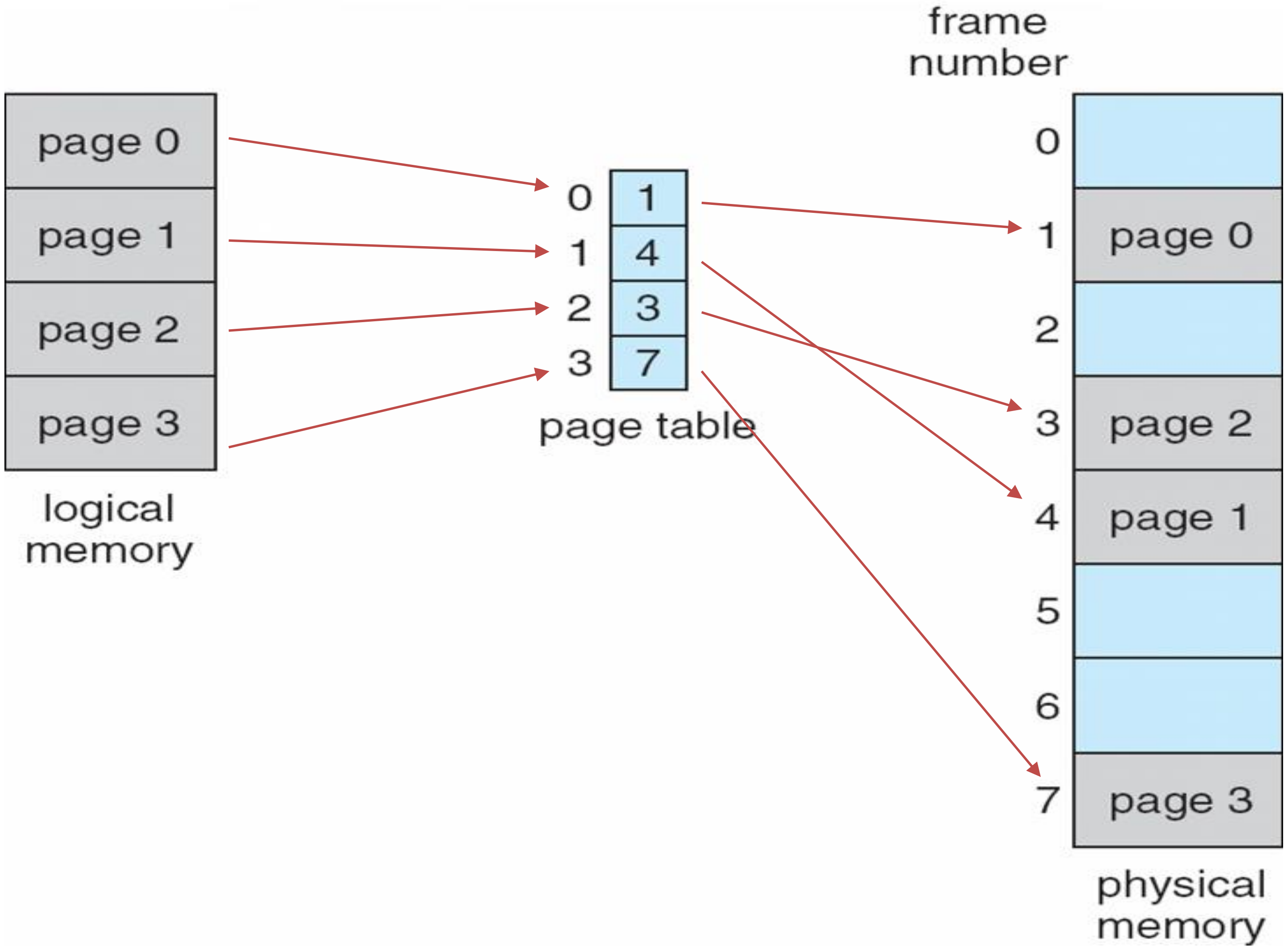


p = page number

d = offset

f = frame number

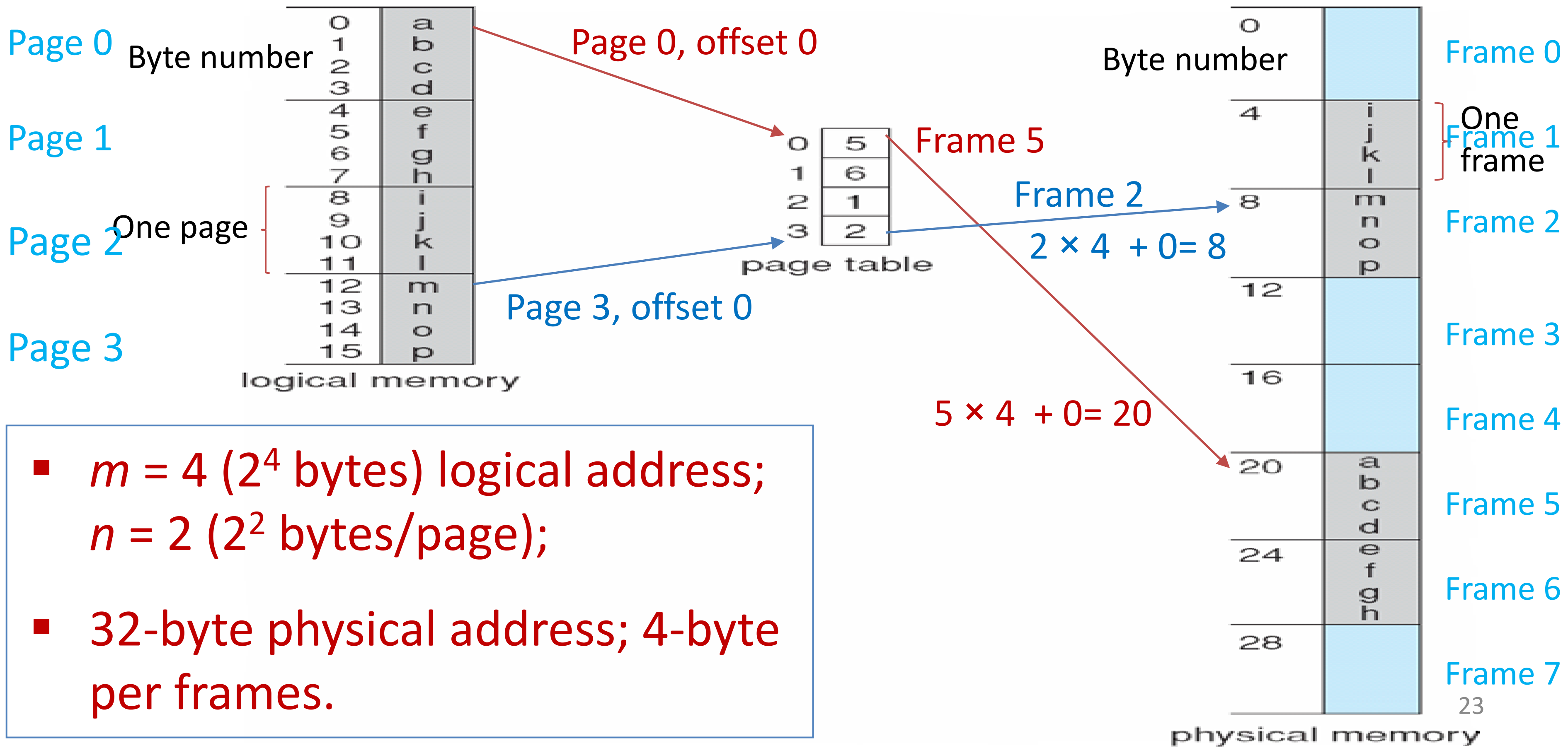
Paging Model of Logical and Physical Memory



University of Glasgow School of Computing Science

Example - Paging

- Page size n = 2 bits; logical address m = 4 bits. Map to page size of 4 bytes; physical memory of 32 bytes (8 frames).



- m = 4 (2^4 bytes) logical address; n = 2 (2^2 bytes/page);
- 32-byte physical address; 4-byte per frames.

Calculating Internal Fragmentation of Paging

Allocate a 72,766 bytes process into a page size 2,048.

- Page size = 2,048 bytes.
- Process size = 72,766 bytes.
- 35 pages + 1,086 bytes.
- **Internal fragmentation**: 2,048 - 1,086 = 962 bytes.
- Worst case fragmentation: 1 byte occupies 1 frame.
- On average fragmentation = 1 / 2 frame size.
- Page sizes ↓, entries in page table ↑, memory space ↑, access overhead ↑. Data transfer is more efficient with large page size (typically 4 KB – 2 MB).

How many pages needed?

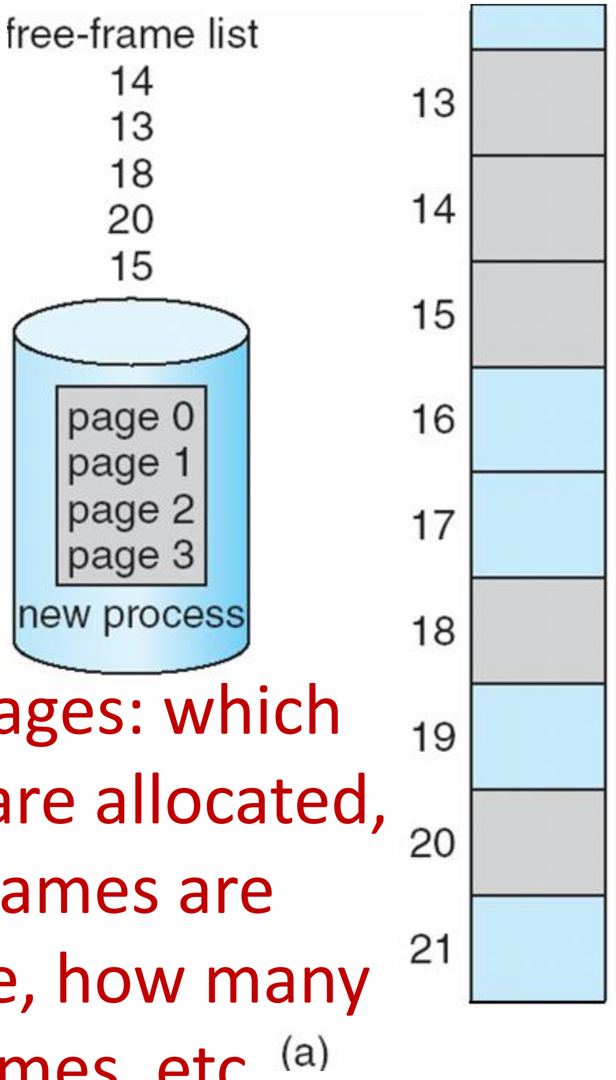
72,766 / 2,048 = 35.5303 pages

How many bytes left over?

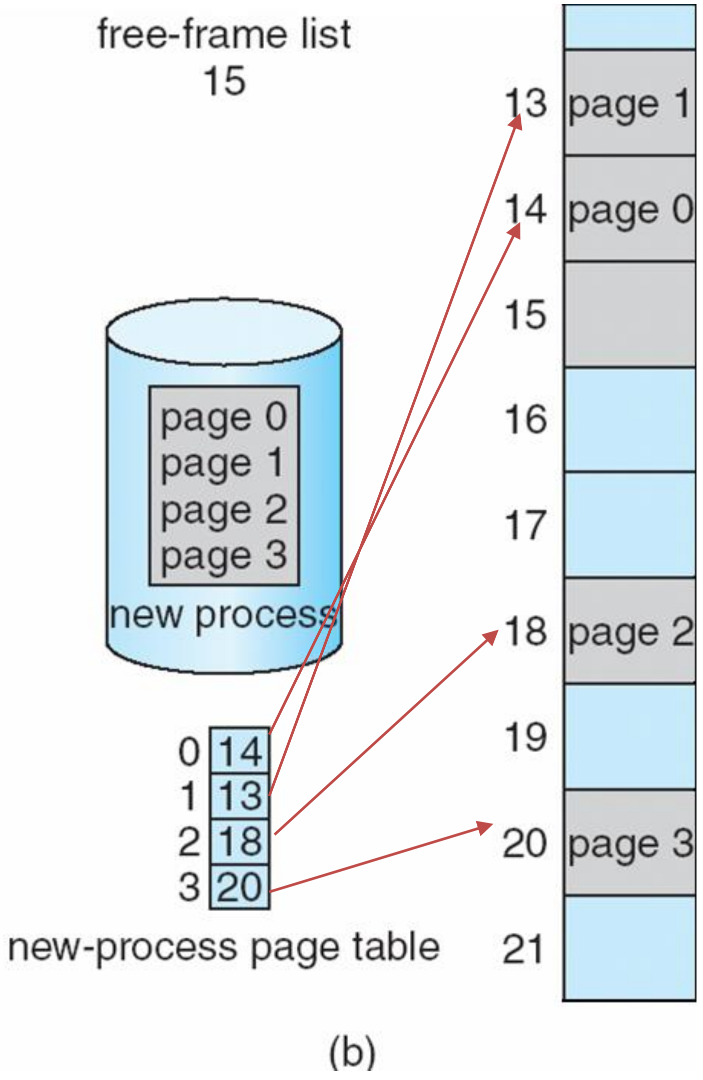
72,766 - 2,048 × 35 = 1,086 bytes

University of Glasgow School of Computing Science

Free Frames



Before allocation



After allocation

OS manages: which frames are allocated, which frames are available, how many total frames, etc.

University of Glasgow | School of Computing Science

Implementation of Page Table

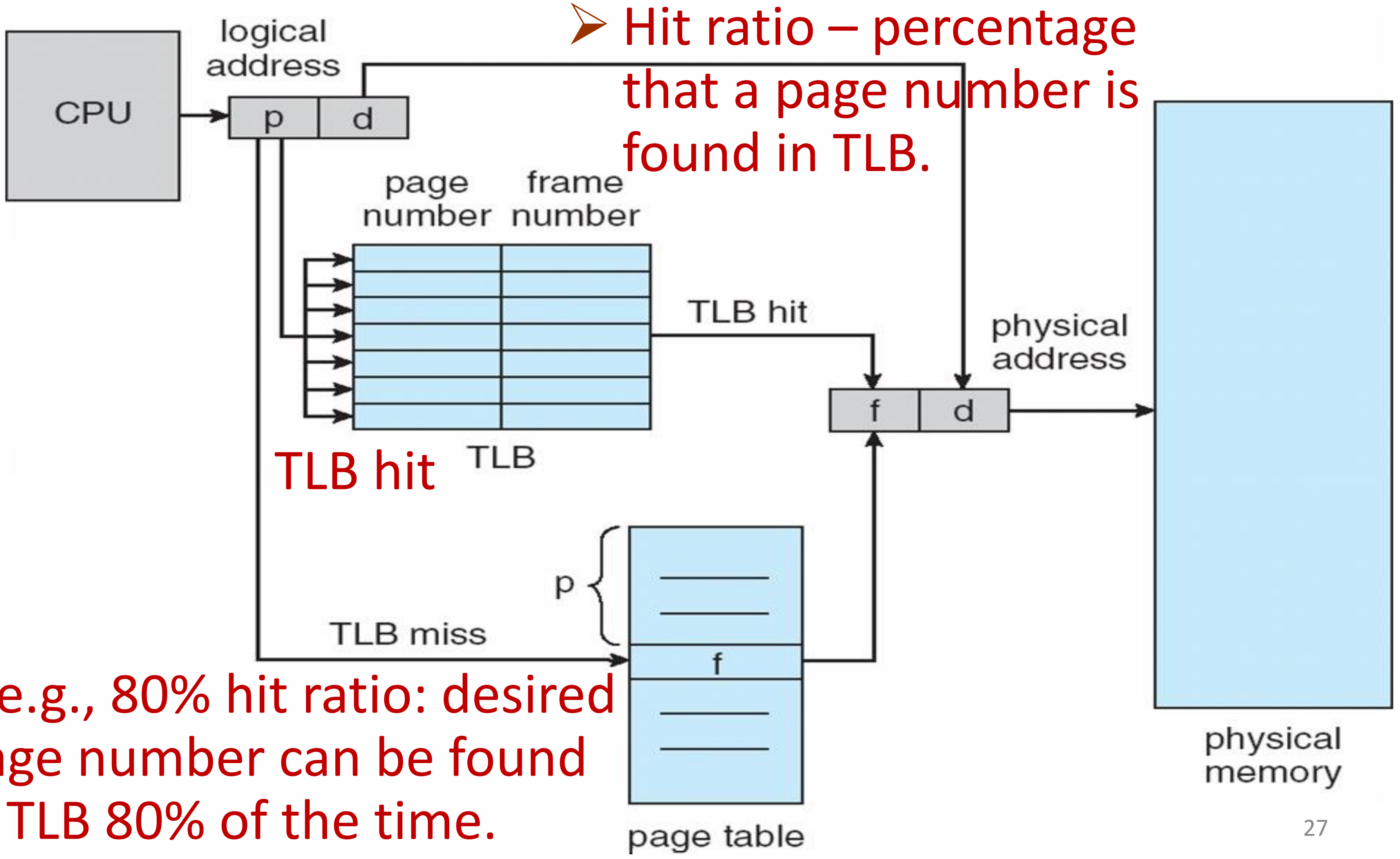
- Every data requires two memory accesses:
  - One for the page table; one for data/instruction.
- This problem can be solved by a special fast-lookup hardware cache: **translation look-aside buffers (TLBs)** (or **associative memory**).
- TLBs typically small (64 to 1,024 entries).
- On a TLB miss, value is loaded into TLB for faster access next time.

University of Glasgow

School of Computing Science

Paging Hardware With TLB

- Hit ratio – percentage that a page number is found in TLB.



TLB hit

- e.g., 80% hit ratio: desired page number can be found in TLB 80% of the time.

University of Glasgow | School of Computing Science

Paging Effective Access Time

- TLB Lookup = ε time unit.
- Hit ratio = α
- Consider α = 80%, ε = 20 ns for TLB search, 100 ns for RAM memory access.
- **Effective Access Time (EAT)**:

  EAT = (ε + 1 RAM access) α + (ε + 2 RAM access)(1 – α)

  - EAT = 0.80 x (20+100) + 0.20 x (20+200) = 140 ns
- Consider more realistic hit ratio → α = 99%, ε = 20 ns for TLB search, 100 ns for memory access.
  - EAT = 0.99 x 120 + 0.01 x 220 = 121 ns.

Memory Protection

- Implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed.
- **Valid-invalid** bit attached to each entry in page table:
  - valid: associated page is in the logical address space of the process, thus a legal page.
  - Invalid: page is not in the logical address space of the process.
- Any violations result in a trap to OS (error).

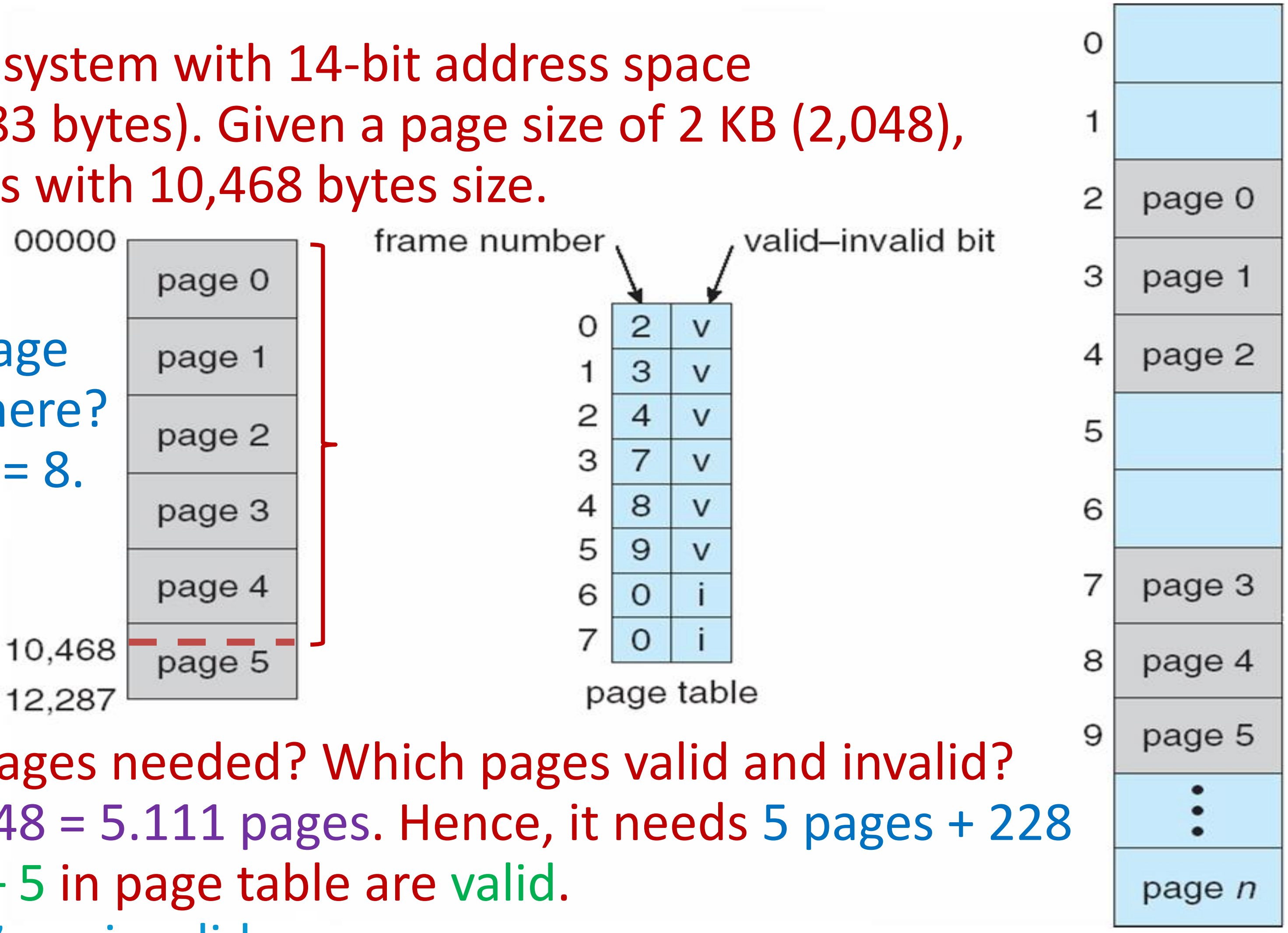
University of Glasgow

School of Computing Science

Valid (v) or Invalid (i) Bit in A Page Table

- e.g., a system with 14-bit address space (0..16,383 bytes). Given a page size of 2 KB (2,048), a process with 10,468 bytes size.

- How many page numbers are there?
  $2^{14} / 2^{11} = 2^3 = 8$.



	frame number	valid–invalid bit
0	2	v
1	3	v
2	4	v
3	7	v
4	8	v
5	9	v
6	0	i
7	0	i

0	
1	
2	page 0
3	page 1
4	page 2
5	
6	
7	page 3
8	page 4
9	page 5
	⋮
	page n

- How many pages needed? Which pages valid and invalid?
  10,468 / 2,048 = 5.111 pages. Hence, it needs 5 pages + 228 bytes. Page 0 – 5 in page table are valid.
  Page 6 and 7 are invalid.

Structure of Page Table

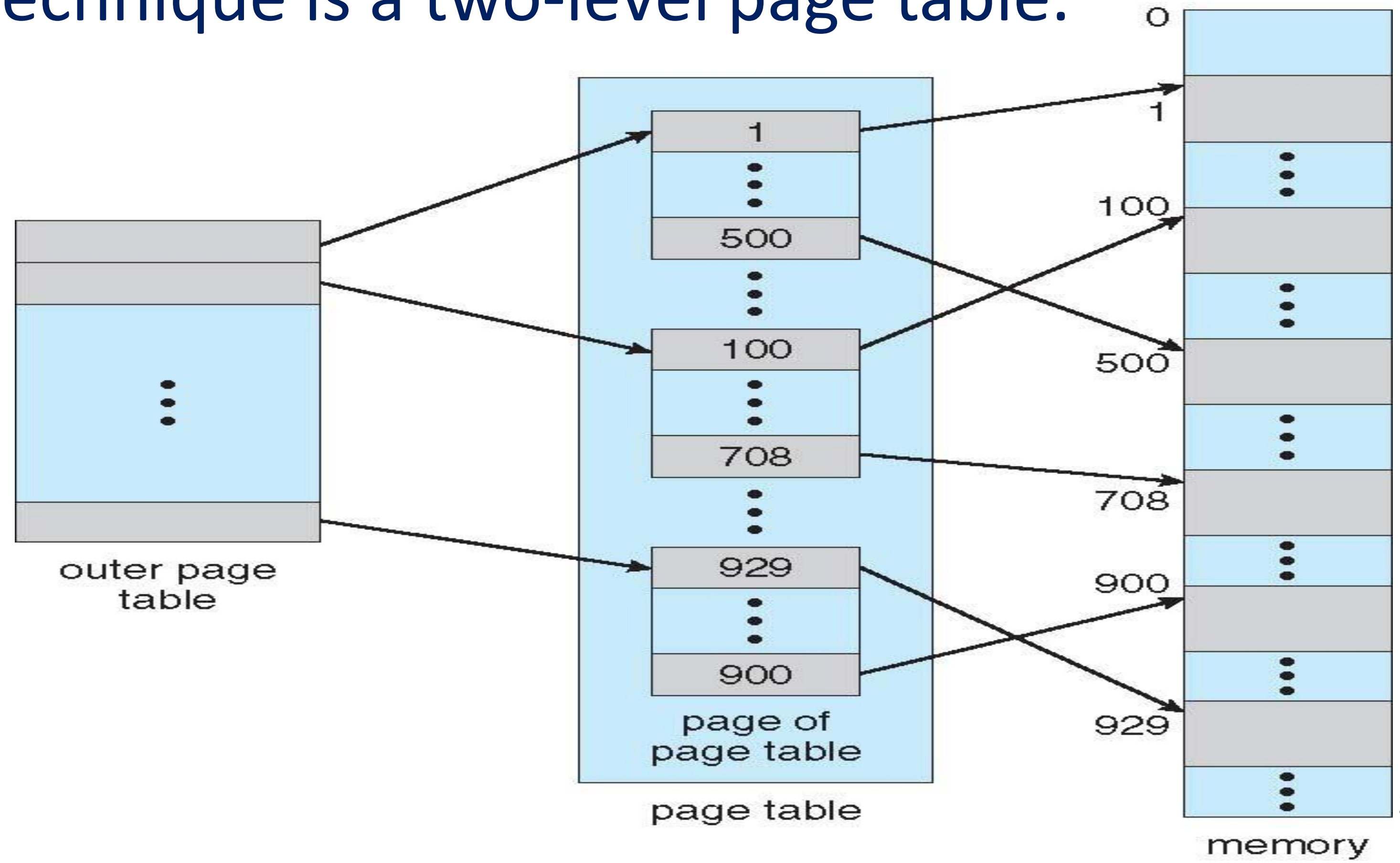
- Memory structures for paging can get huge using straight-forward methods:
  - e.g., 32-bit logical address space on a PC (2^{32}).
  - Page size = 4 KB (2^{12}).
  - Page table entries = 2^{32} / 2^{12} = 1 M.
  - If 4 bytes per entry ➔ 4 MB of physical address for page table alone (per process!).
  - Solution to divide page table into smaller units.
    1) Hierarchical Paging.
    2) Hashed Page Tables.
    3) Inverted Page Tables.

University of Glasgow

School of Computing Science

1st Method: Hierarchical Page Tables

- Break up logical address into multiple page tables.
- A technique is a two-level page table.



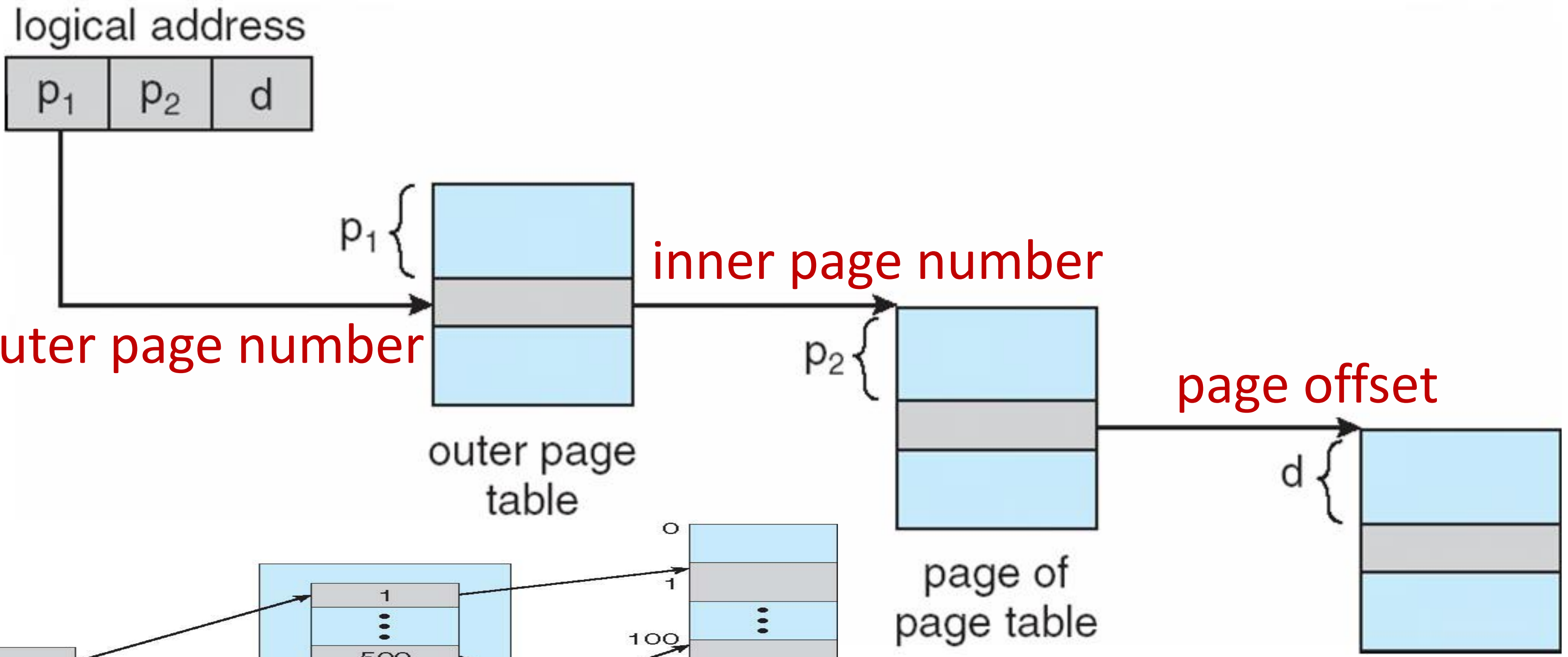
Two-Level Paging Example

- A 32-bit logical address with 4 KB page size divided:
  - Page number = 20 bits (2^{20}).
  - Page offset = 12 bits (4 KB = 2^{12}). (d: page offset)
- Page number is further divided into:
  - 10-bit page number. (p_1: index of outer page table)
  - 10-bit page offset. (p_2: displacement within the page of inner page table)
- Thus, logical address is:

outer page	inner page	page offset
p_1	p_2	d
10	10	12

University of Glasgow | School of Computing Science

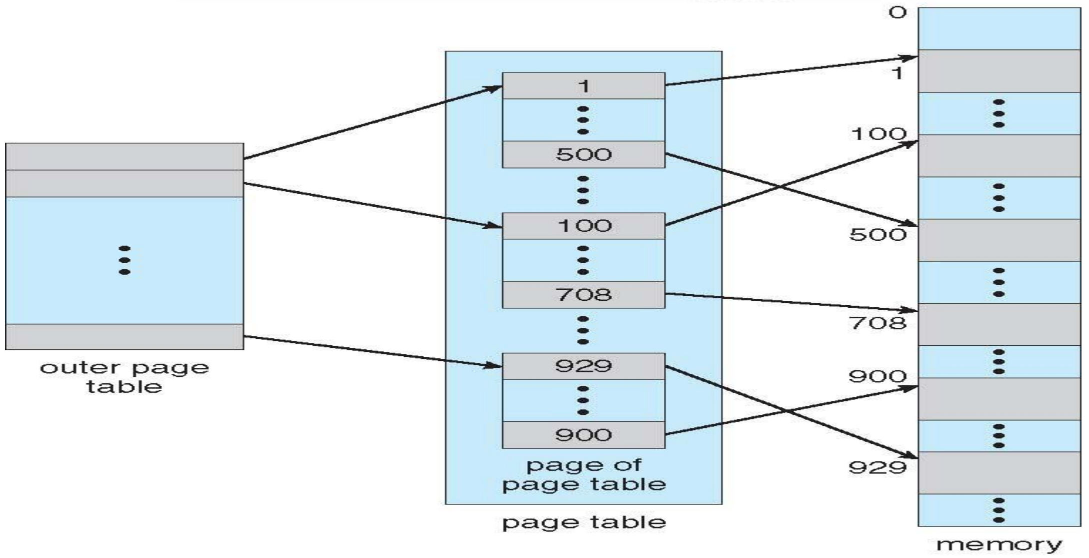
Address-Translation Scheme



inner page number

outer page number

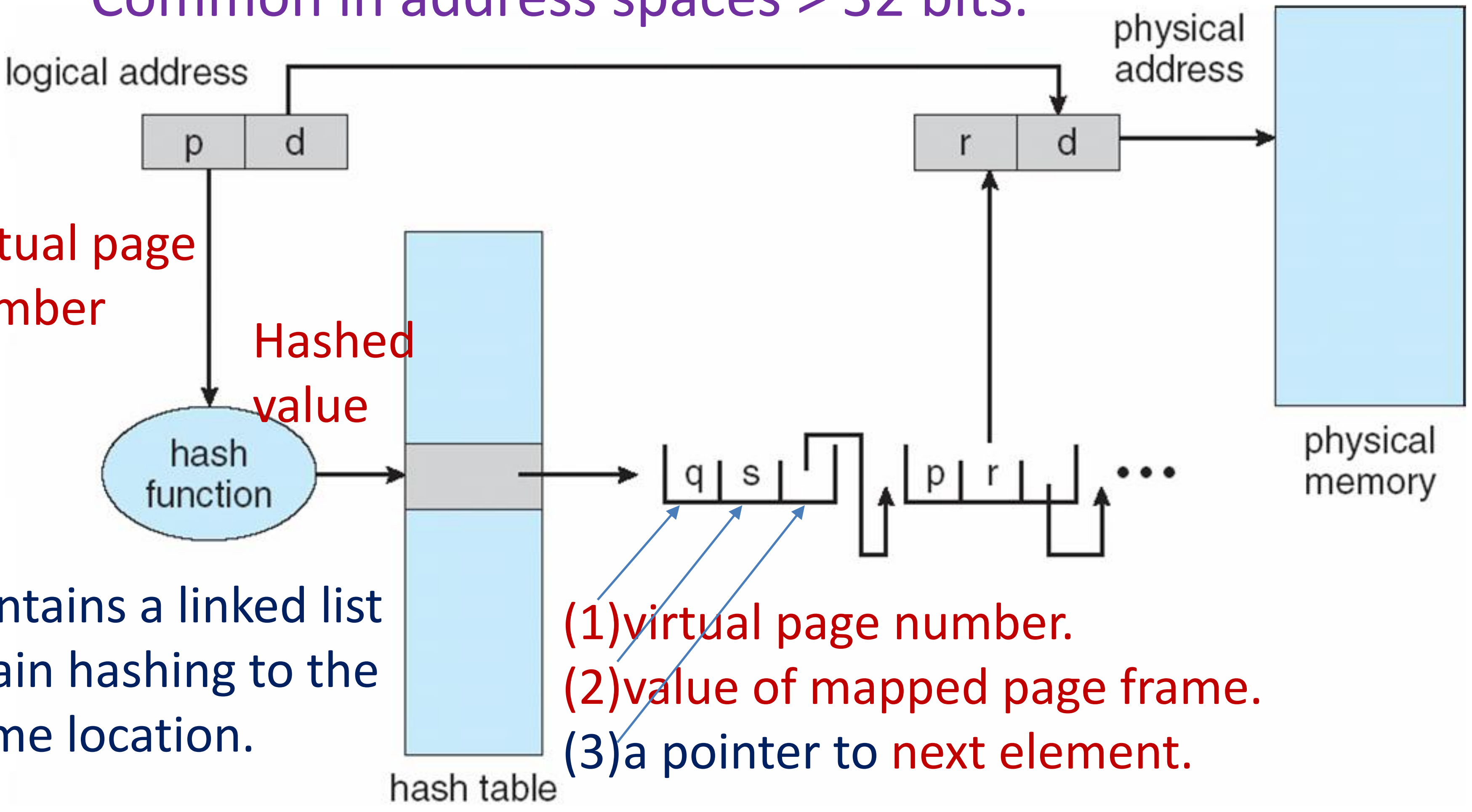
page offset



University of Glasgow | School of Computing Science

2nd Method: Hashed Page Tables

Common in address spaces > 32 bits.



Virtual page number

Hashed value

Contains a linked list chain hashing to the same location.

(1)virtual page number.
(2)value of mapped page frame.
(3)a pointer to next element.

University of Glasgow

School of Computing Science

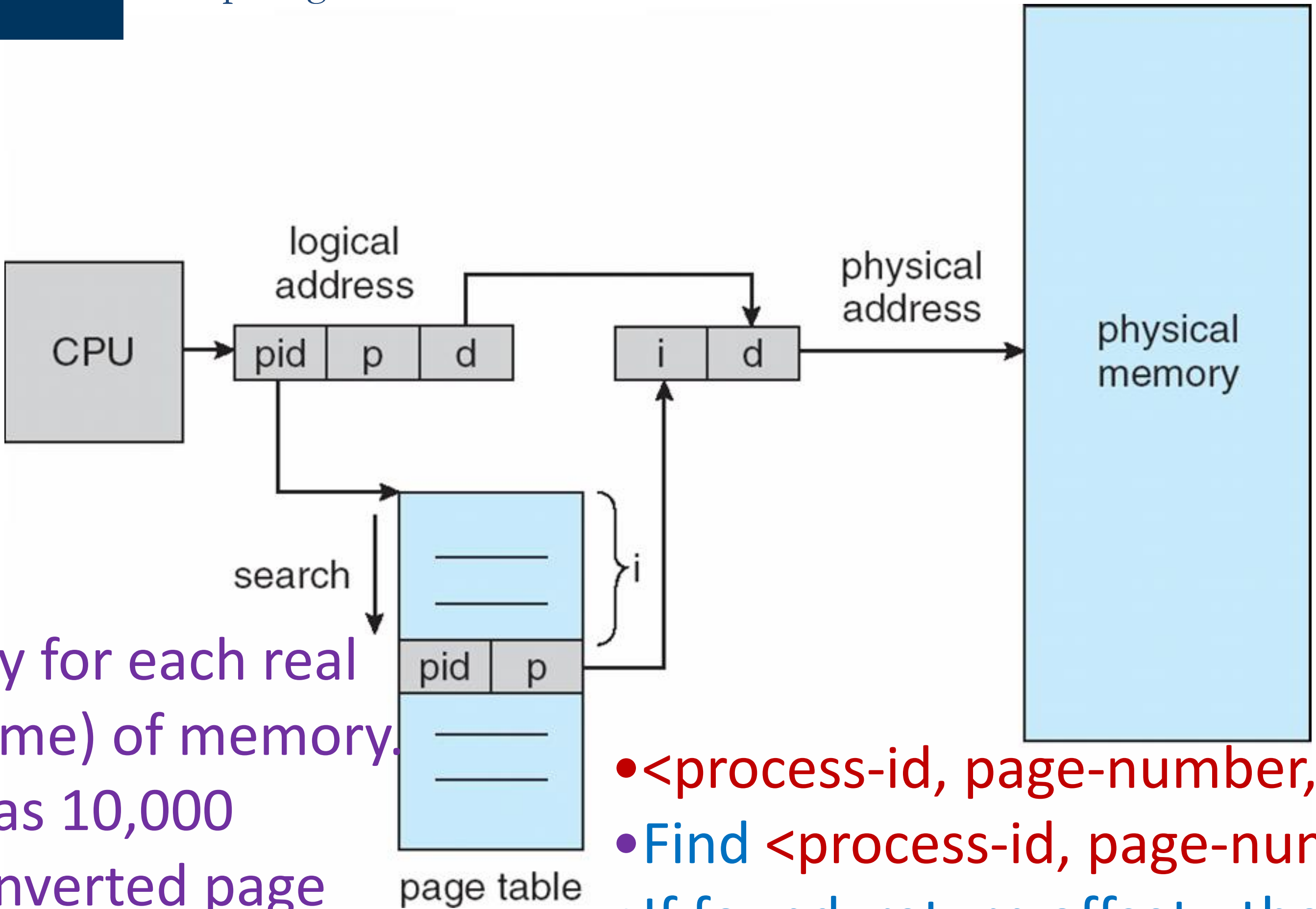
3rd Method: Inverted Page Table

- Rather than each process with a page table, there is only 1 page table shared by all processes. Each entry is for each real page (frame) of memory.
- Entry consists of page virtual address stored in real memory locations, with process info (pid).
- Decreases memory to store the page table, but increases time to search the table.

University of Glasgow

School of Computing Science

Inverted Page Table



- One entry for each real page (frame) of memory.
- If RAM has 10,000 entries, inverted page table has 10,000 entries.

- <process-id, page-number, offset>
- Find <process-id, page-number>
- If found, return offset - the *i*th entry in invented page table.

- Since inverted page table has the same number of entry as physical memory, *i* is the exact location in memory.

Next Lecture

Lecture 9: Virtual-Memory Management