



University of
Nottingham

UK | CHINA | MALAYSIA

Operating Systems and Concurrency

Lecture 17:Memory Management 3

Edited by: Dr Qian Zhang
University of Nottingham, Ningbo China
2023

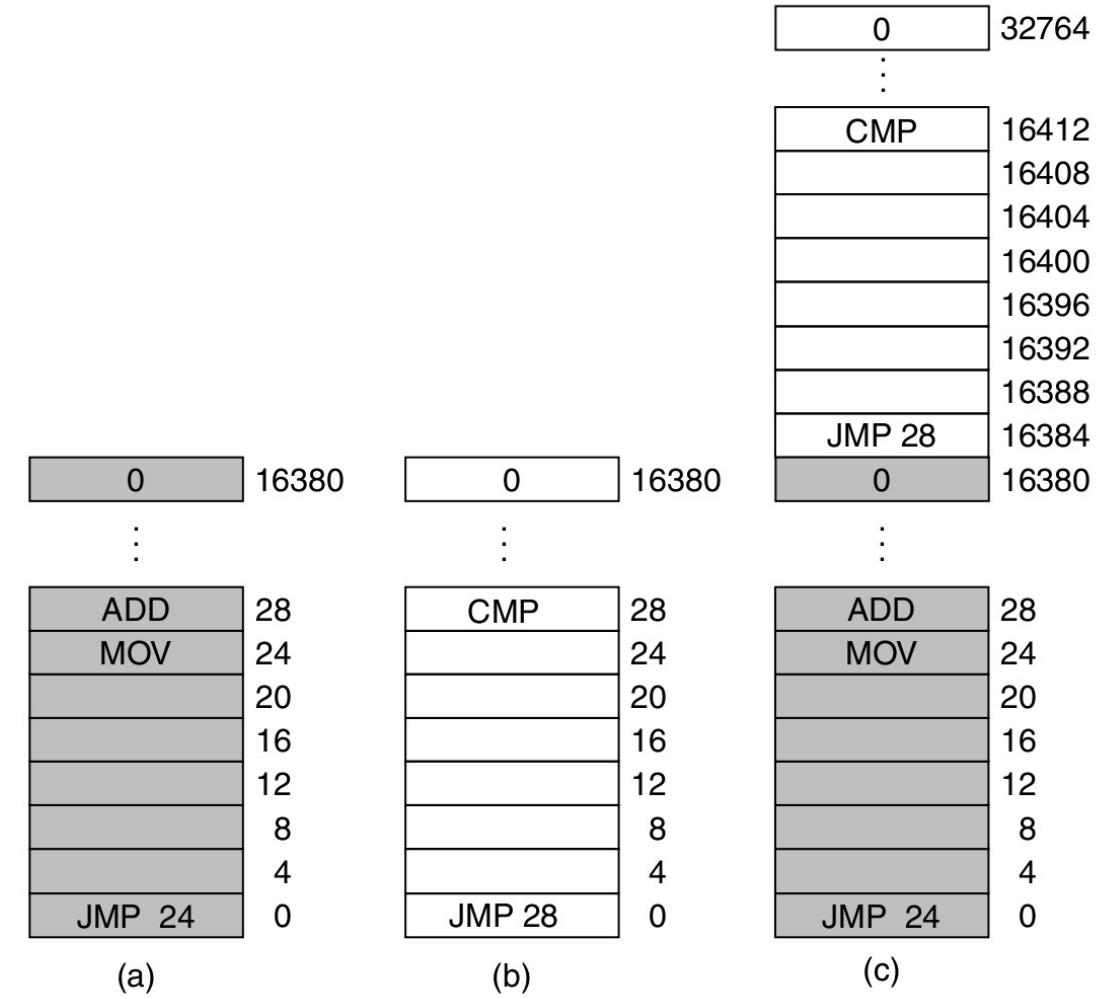
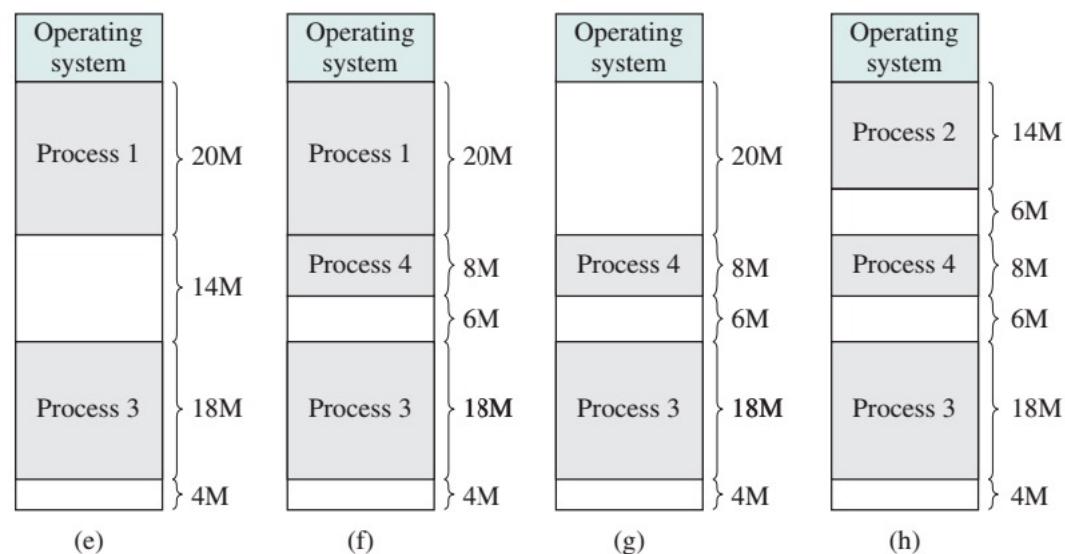
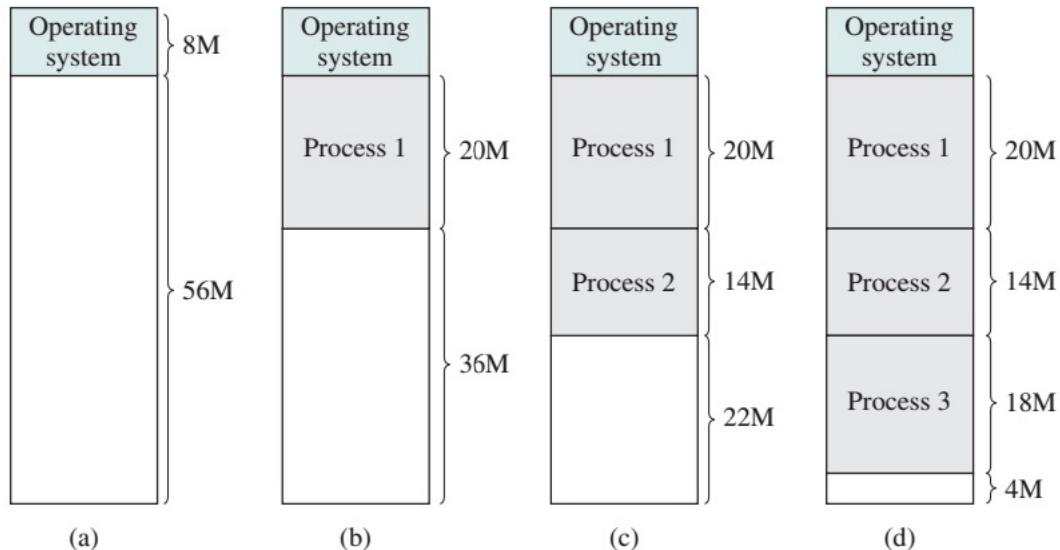


Overview

Goals for Today

- **Code relocation and protection**
- **Non-contiguous approaches**
- **Paging, page tables, address translation**

Relocation and Protection Problems





Relocation and Protection Principles

- **Relocation:** when a program is run, it **does not know in advance** which **partition/addresses** it will occupy
 - The program cannot simply generate **static addresses** (e.g. jump instructions) that are **absolute**
 - Addresses should be **relative to where the program has been loaded**
 - **Relocation must be solved** in an operating system that allows processes to run at **changing memory locations (on the fly)**
- **Protection:** once you can have two programs in memory at the same time, protection must be enforced



Relocation and Protection Principles

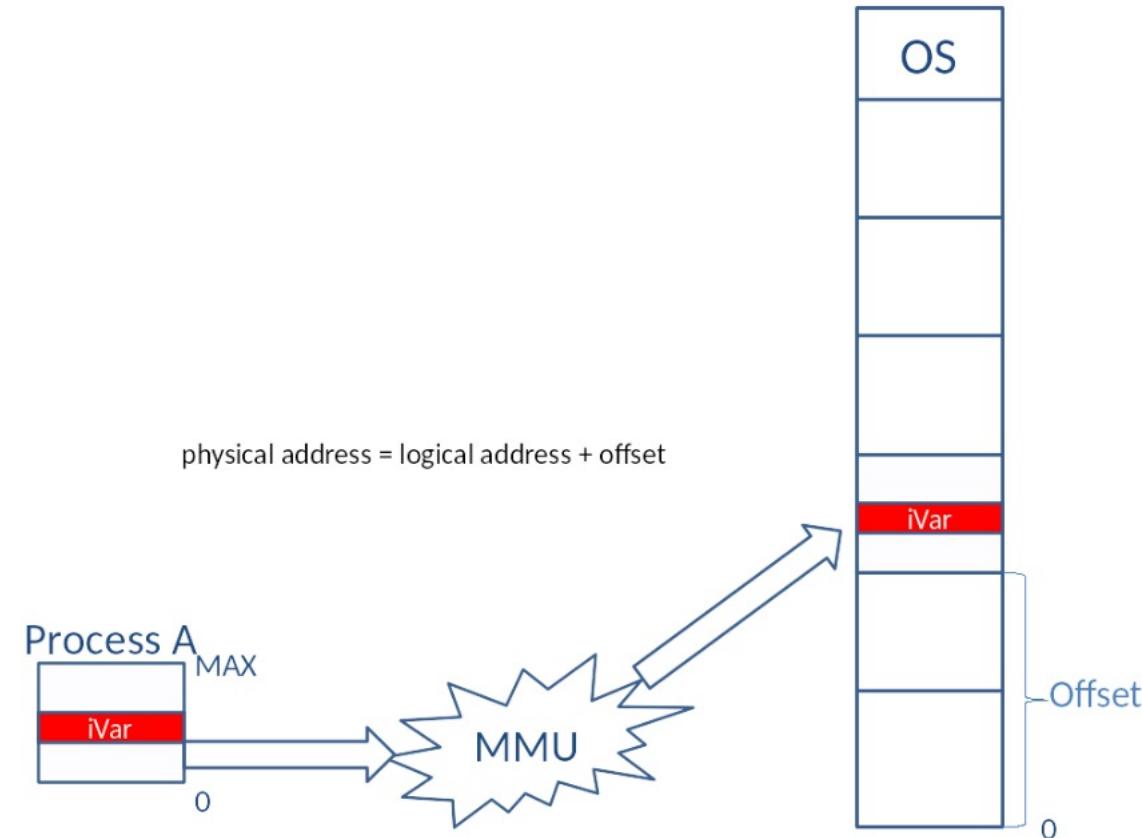


Figure: Address Relocation



Relocation and Protection

Address Types

- A **logical address** is a memory address **seen by the process**.
 - It is **independent** of the current **physical memory** assignment
 - It is, e.g., **relative to the start of the program**
- A **physical address** refers to an **actual location in main memory**
- The **logical address space** must be mapped onto the machine's **physical address space**



Relocation and Protection Approaches

- **Static “relocation” at compile time:** a process has to be located at the same location every single time (impractical)
- **Dynamic relocation at load time**
 - An **offset** is added to every logical address to **account for its physical location** in memory
 - **Slows down** the loading of a process, does not account for **swapping**
- **Dynamic relocation at runtime**

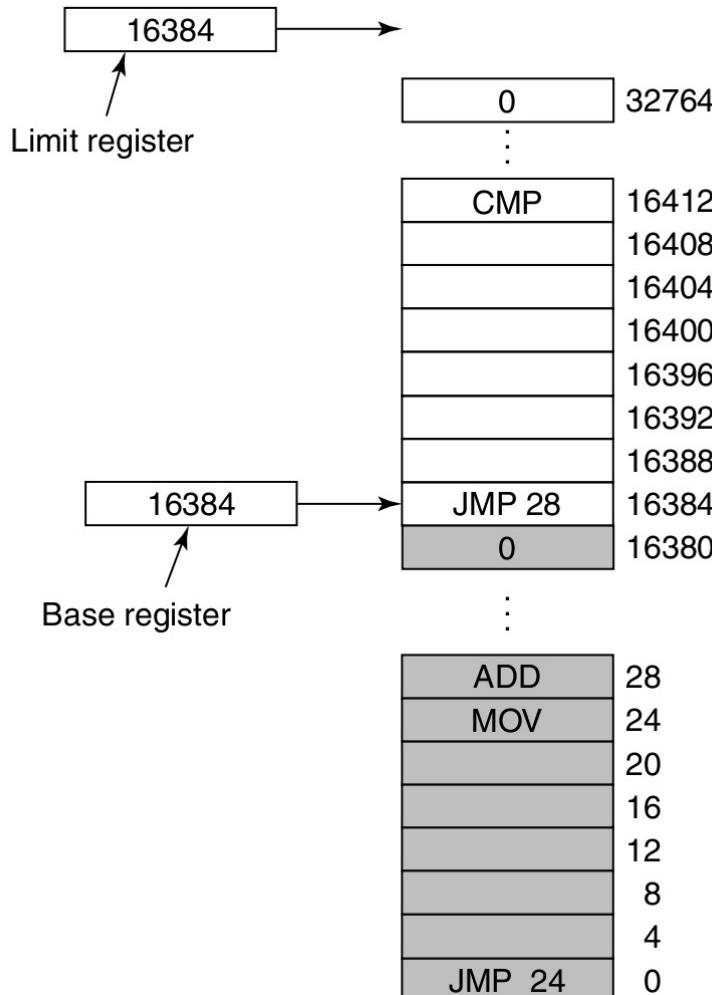


Relocation and Protection

At Runtime: Base and Limit Registers

- Two special purpose registers are maintained in the CPU (the **MMU**), containing a **base address** and **limit**
 - The **base register** stores the **start address** of the partition
 - The **limit register** holds the **size** of the partition (end of program)
- This approach requires **hardware support** (was not always present in the early days!)

Relocation and Protection Principles



- **At runtime:**

- The **base register** is added to the **logical (relative) address** to generate the **physical address**
- The resulting address is **compared** against the **limit register**
- The Relocation also provides a measure of **protection**: each process image is **isolated** by the contents of the base and bounds registers and **safe from unwanted accesses** by other processes.

Contiguous Allocation Schemes

Overview and Shortcomings

- Different contiguous memory allocation schemes have different advantages/disadvantages
 - **Mono-programming** is easy but does result in **low resource utilisation**
 - **Fixed partitioning** facilitates **multi-programming** but results in **internal fragmentation**
 - **Dynamic partitioning** facilitates **multi-programming**, reduces **internal fragmentation**, but results in **external fragmentation** (allocation methods, coalescing, and compacting help)
 - **Memory management** using **linked lists** and **bitmaps**
- Can we design a memory management scheme that **resolves the shortcomings** of contiguous memory schemes?

Memory management has been evolved

- Approaches

- Contiguous Memory**

- ❖ **Mono-programming** [easy but low resource utilisation, cannot be used for modern multiprogramming machine]
No relocation, physical address
 - ❖ **Multiprogramming with fixed partition** [multi-programming but internal fragmentation]
Relocation (register), logical address
 - ❖ **Multiprogramming with dynamic partition** [low internal fragmentation but high external fragmentation]
Relocation (register), logical address

- Non-Contiguous Memory**
Relocation (table), logical address

- ❖ **Paging** (fixed-partitioning/code re-location)
[Internal fragmentation reduce to last block, no external fragmentation; Require page table to maintain page relocation]
 - ❖ **Virtual Memory** (locality → not all pages loaded)
[more processes → CPU utilisation, no external fragmentation, more memory available; Need involve replacement algorithm, page table management, thrashing, variable/fixed resident set....]



- Paging uses the principles of **fixed partitioning** and **code re-location** to devise a new **non-contiguous** management scheme:
 - Memory is split into much **smaller blocks** and **one or multiple blocks** are allocated to a process
 - e.g., a 11kb process would take up 3 blocks of 4 kb
 - These blocks **do not have to be contiguous in main memory**, but the process still **perceives them to be contiguous**
- Benefits compared to contiguous schemes include:
 - **Internal fragmentation** is reduced to the **last “block”** only
 - There is **no external fragmentation**, since physical blocks are **stacked directly onto each other** in main memory

Paging Principles (Cont.)

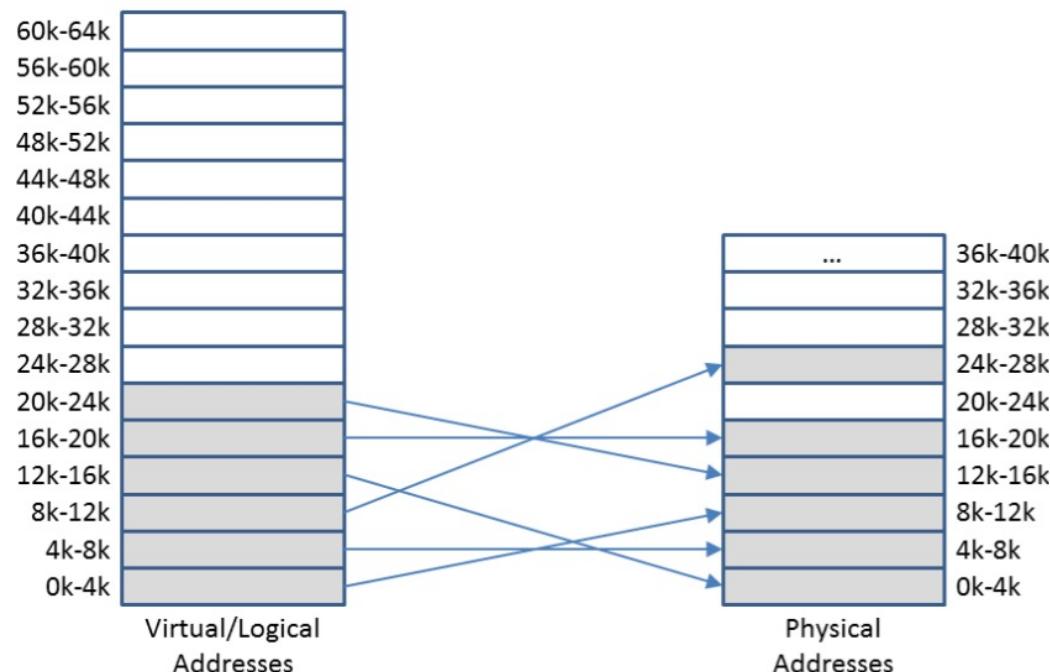


Figure: Paging in main memory with multiple processes

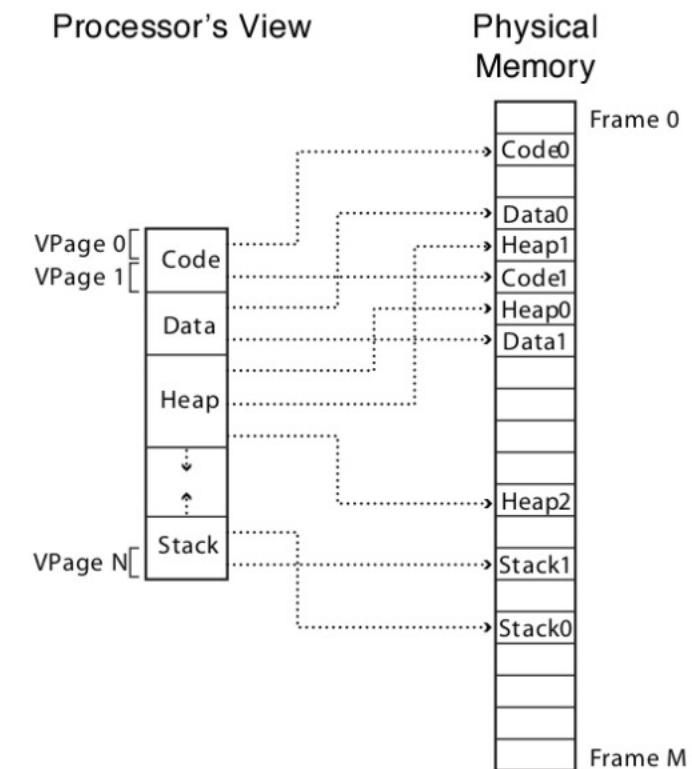


Figure: Paging in main memory with multiple processes (Anderson)

Paging Principles (Cont.)

Main memory	
0	A.0
1	A.1
2	A.2
3	A.3
4	B.0
5	B.1
6	B.2
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

(d) Load process C

Main memory	
0	A.0
1	A.1
2	A.2
3	A.3
4	
5	
6	
7	C.0
8	C.1
9	C.2
10	C.3
11	
12	
13	
14	

(e) Swap out B

Main memory	
0	A.0
1	A.1
2	A.2
3	A.3
4	D.0
5	D.1
6	D.2
7	C.0
8	C.1
9	C.2
10	C.3
11	D.3
12	D.4
13	
14	

(f) Load process D

Figure: Concept of Paging (Stallings)



Paging

Principles: Definitions

- A **page** is a small block of **contiguous memory** in the **logical address space**, i.e. as seen by the process
- A **page frame** is a small contiguous block in physical memory
- Pages and frames (usually) have the **same size**:
 - The size is usually a power of 2
 - Sizes range between 512 bytes and 1Gb

Paging Relocation

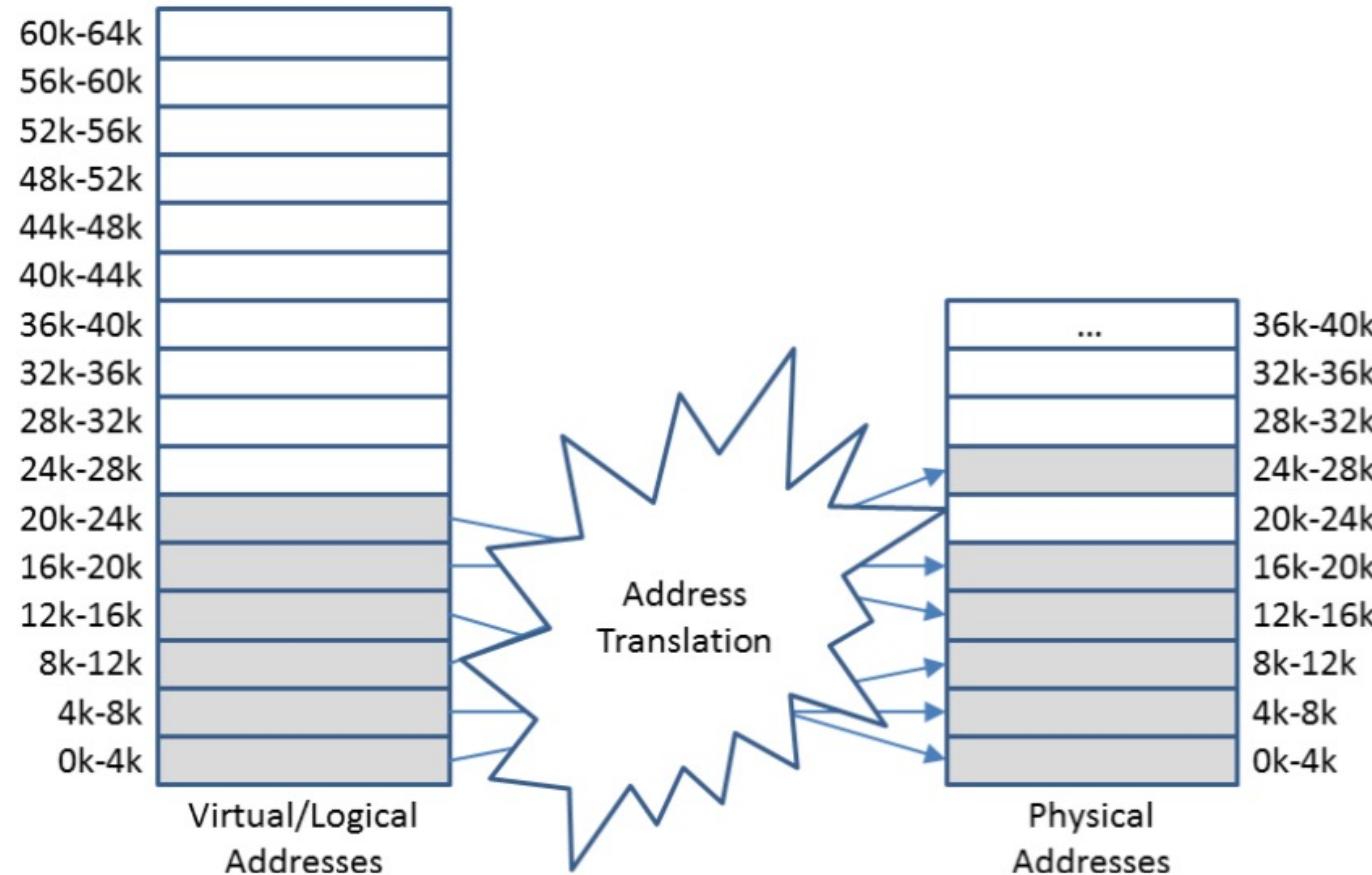


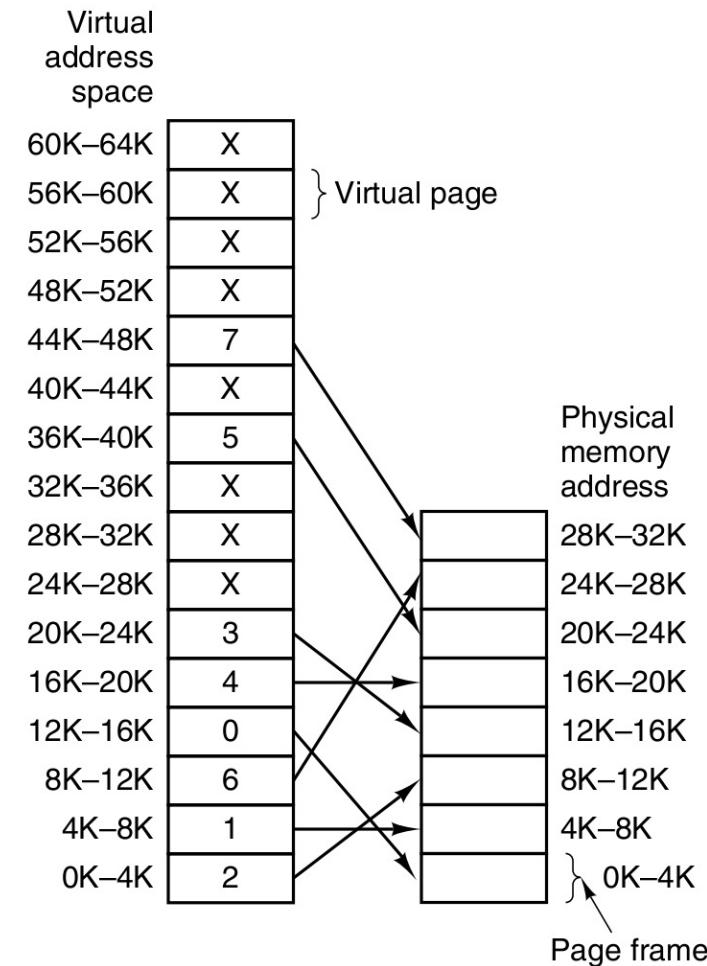
Figure: Address Translation



- **Logical address** (page number, offset within page) needs to be translated into a **physical address** (frame number, offset within frame)
- **Multiple “base registers”** will be required:
 - Each logical page needs a **separate “base register”** that specifies the start of the associated frame
 - I.e, a **set of base registers** has to be maintained for each process
- The base registers are stored in the **page table**

Paging Address

- The relations between virtual addresses and physical memory addresses is given by the page table.
- Every page begins on a multiple of $4096 (2^{12})$ and ends 4095 addresses higher,
 - so 4K-8K really means 4096-8191
 - and 8K to 12K means 8192-12287



Paging

Relocation: Address Translation

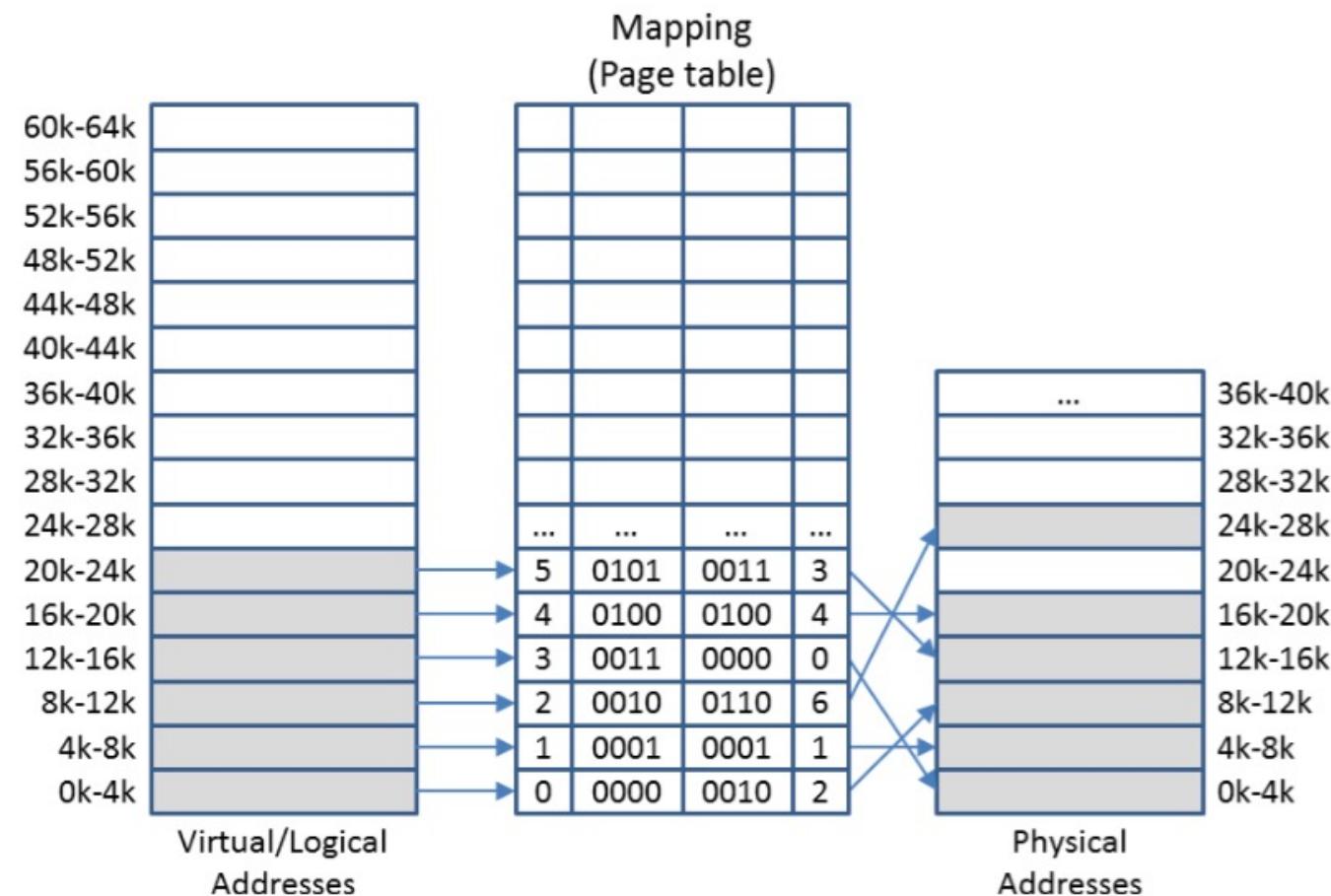


Figure: Address Translation



Paging

Relocation: Page Tables

- The page table can be seen as **a function**, that **maps the page number** of the logical address **onto the frame number** of the physical address
 - $\text{frameNumber} = f(\text{pageNumber})$
- The **page number** is used as **index to the page table** that lists the **number of the associated frame**, i.e. it contains the location of the frame in memory
- Every process has its **own page table** containing its own “base registers”
- The **operating system** maintains a **list of free frames**

Paging

Address Translation: Implementation

- A **logical (physical) address** is relative to the start of the **program (memory)** and consists of two parts:
 - The **right most m bits** that represent the **offset within the page (frame)**
 - e.g. 12 bits for the offset, allowing up to 4096 (2^{12}) bytes per page (frame)
 - The **left most n bits** that represent the **page (frame) number**
 - e.g. 4 bits for the page number allowing 16 (2^4) pages (frames)
- The **offset** within the page and frame **remains the same** (they are the same size)
- The page number to frame number mapping is held in the **page table**

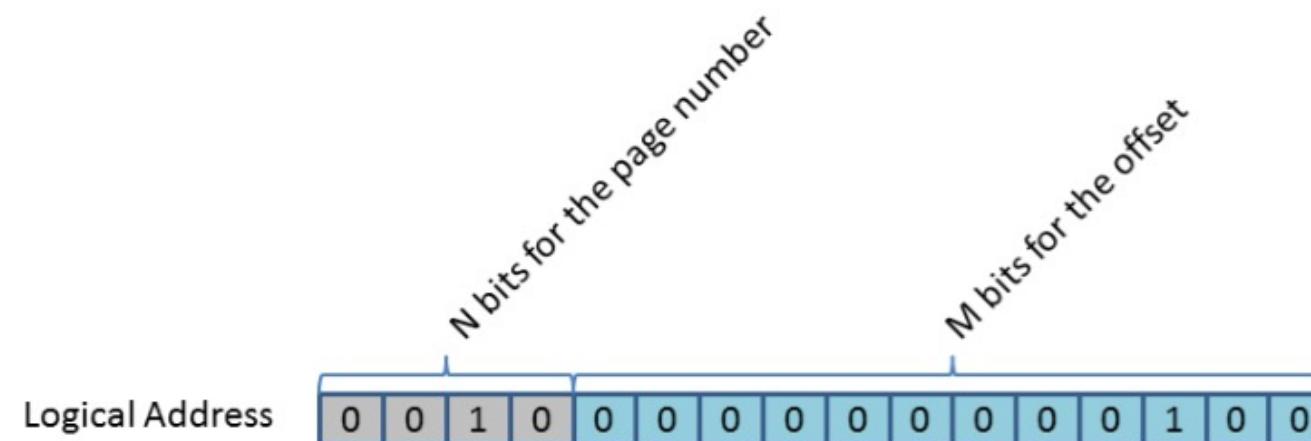
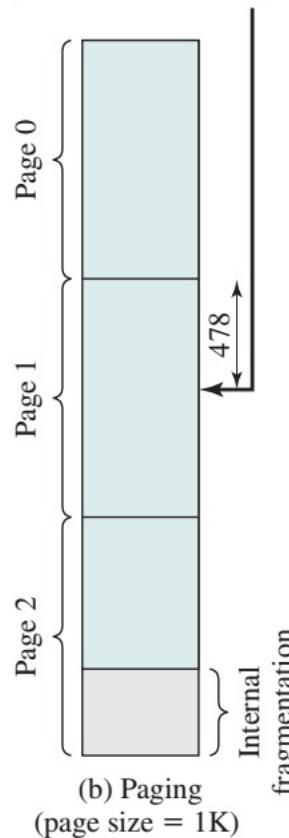


Figure: Logical Address

Paging

Address Translation: Implementation

Logical address =
Page# = 1, Offset = 478
`000001|0111011110`



Logical Address

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Logical Address

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Logical Address

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Logical Address

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Logical Address

0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Logical Address

0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Logical Address

0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Page # = 0, offset = 0, address = 0

Page # = 0, offset = 1, address = 1

Page # = 0, offset = 2, address = 2

Page # = 0, offset = 3, address = 3

Page # = 0, offset = $2^{12} - 1 = 4095$,
address = 4095

Page # = 1, offset = 0,
address = $1 \times 2^{12} = 4096$

Page # = 1, offset = 3,
address = $1 \times 2^{12} + 3 = 4099$

Paging

Address Translation: Implementation

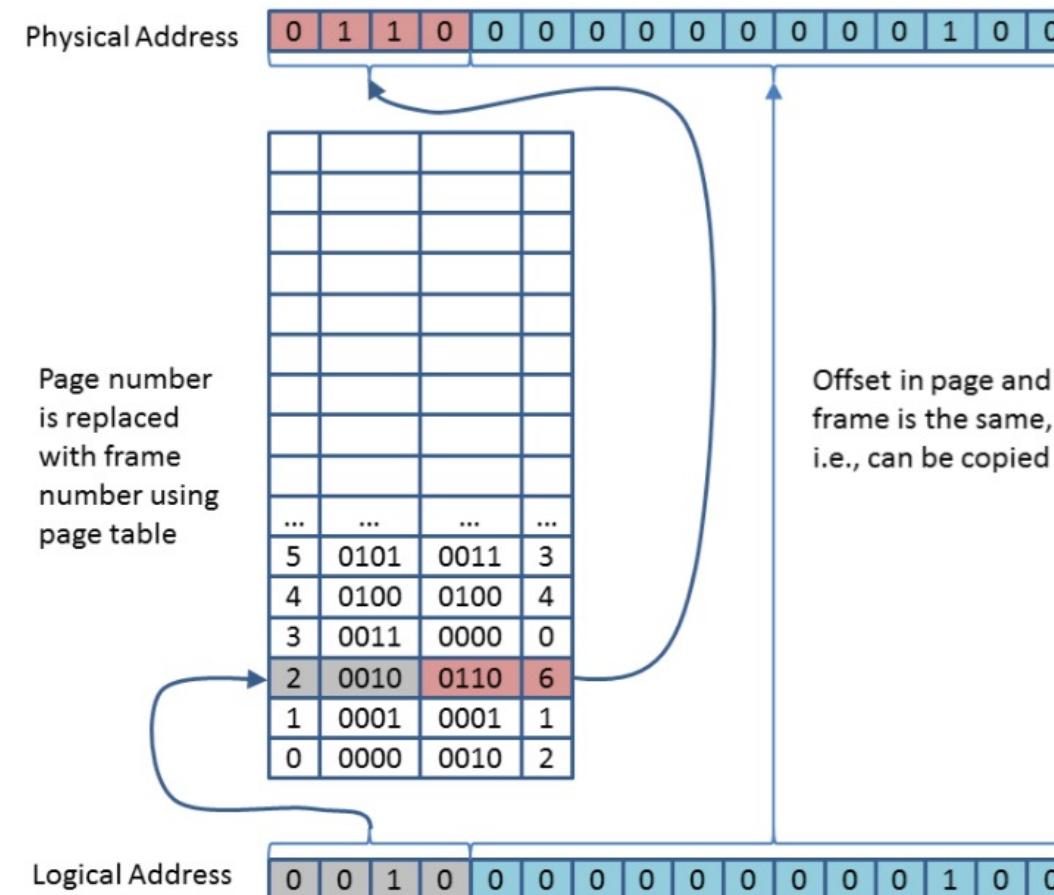


Figure: Address Translation



- Steps in **address translation**:
 - 1 **Extract the page number** from logical address
 - 2 Use page number as an index to **retrieve the frame number** in the page table
 - 3 **Add the “logical offset within the page”** to the start of the physical frame
- **Hardware implementation** of address translation
 - 1 The CPU’s **memory management unit** (MMU) intercepts logical addresses
 - 2 MMU uses a page table as above
 - 3 The resulting **physical address** is put on the **memory bus**



Recap

Take-Home Message

- **Relocation and protection : principles underpin paging and virtual memory!**
- **Paging, page tables, and address translation**



University of
Nottingham

UK | CHINA | MALAYSIA

Quiz!

Paging

Address Translation: Examples

- Let us assume 4KB pages (2^{12}), leaving $2^4 = 16$ pages
- What is the physical address of 0, 8192, 20500 using the page table below

Pages	Frames	
0 0000	0010	2
1 0001	0001	1
2 0010	0110	6
3 0011	0000	0
4 0100	0100	4
5 0101	0011	3
6 0110	X	X
7 0111	X	X
8 1000	X	X
9 1001	0101	5
10 1010	X	X
11 1011	0111	7
12 1111	X	X

Table: Page Table

- Virtual address 0 falls in page with index 0 which is mapped onto frame with index 2, starting at 2×4096 , i.e., the physical address is 8192
- Virtual address 8192 falls in page with index 2 which is mapped onto frame with index 6, starting at 6×4096 , i.e., the physical address is 24576
- Virtual address 20500 falls in page with index 5 ($20500 > 5 \times 4096$) which is mapped onto page with index 3, starting at 3×4096 , i.e., the physical address is $12288 + 20 = 12308$