MONASH University

MONASH
INFORMATION
TECHNOLOGY

VIRTUAL MEMORY

LECTURE 9 / FIT2100 / SEMESTER 2 2019

WEEK 10

VIRTUAL MEMORY

INTRODUCTION

LEARNING OUTCOMES

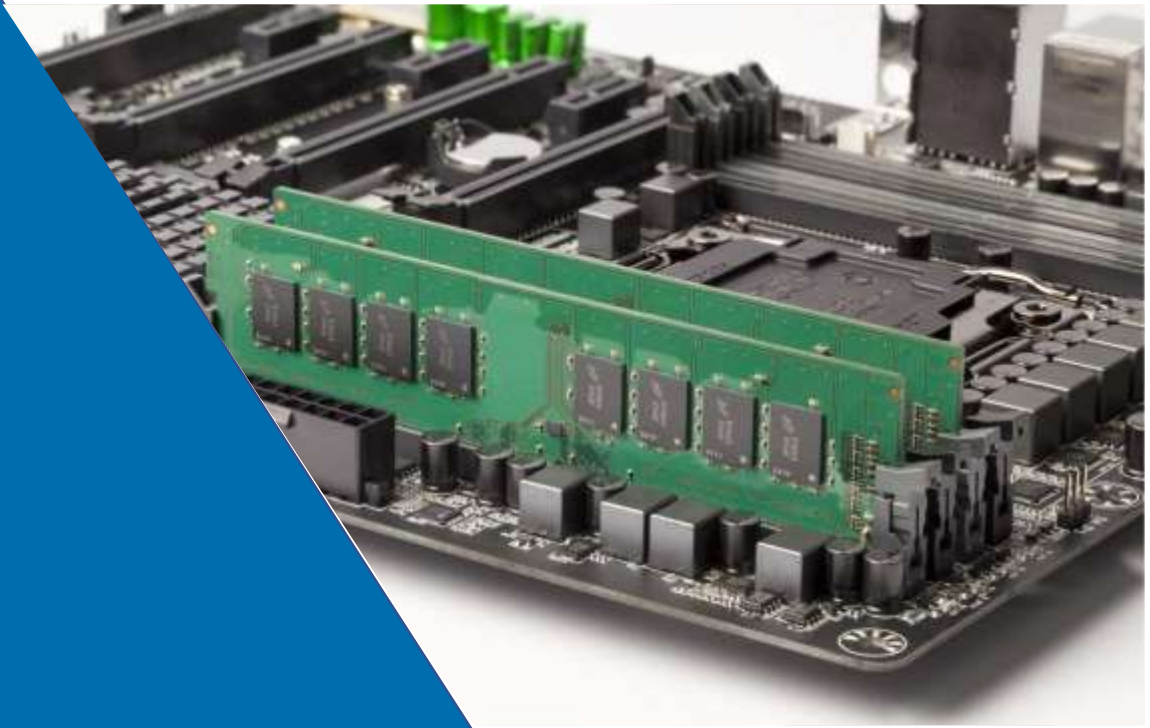
- Understand why **virtualised** memory is useful
- Discuss the differences between paging and segmentation
- Understand how to translate a logical memory address into a physical address
- Describe the most common page-replacement algorithms.

READING

- Stallings: Chapter 7, Chapter 8.

MONASH University

WHAT IS VIRTUAL MEMORY?



LET'S RECAP…

WHAT IS PHYSICAL MEMORY?

WHAT IS MAIN MEMORY?

A LONG, LINEAR LIST OF ADDRESSABLE BYTES.

- A physical memory address corresponds **directly** to a byte storage location in hardware.

MONASH University

VIRTUAL MEMORY IS…

AN ABSTRACTION OF PHYSICAL MEMORY

LOGICAL ADDRESSING

- Instead of dealing with physical memory addresses directly, the OS gives programs **logical addresses** to work with.
- Logical memory addresses are then translated into **physical addresses**.

THE 'REAL' MEMORY ADDRESS IS HIDDEN FROM THE PROGRAM.

- The logical address is also known as a **virtual address.**
- All memory address pointers you've worked with in C have been virtual addresses.

MONASH University

BUT WHY?

WHY VIRTUALISE THE ADDRESS SPACE? (1/3)

<u>EFFICIENCY</u>, SECURITY, CONCURRENCY, FLEXIBILITY, …

- Efficient allocation of resources:
  - Main memory is a limited, expensive resource.
  - Buy more memory? But most of it will be un-used most of the time…
  - Why not swap parts of memory out to secondary storage when not required?
    - Divide memory into either **pages** or **segments** that can be swapped in and out… (later in lecture)

- Make better use of available memory:
- More processes can be run at a time
- A process can be run even if larger than **all** of main memory.

BUT WHY?

WHY VIRTUALISE THE ADDRESS SPACE? (2/3)

EFFICIENCY, SECURITY, CONCURRENCY, FLEXIBILITY, …

- Enables protection against unauthorised access to another process's memory space
- Enables **shared memory** among related processes
  - A segment of memory can be 'mapped' into the memory space of multiple related processes
  - e.g. multiple processes can share certain variables
  - Multiple instances of the same program can share code.

- e.g. When `fork`ing or running multiple instances of the **same program**, the program code only needs to be loaded once.
- Each process sees its own **logical** copy of the program code, but the logical segments of memory are translated to a common physical location.
- This is why you can't modify hard-coded string literals in C. They are part of the program code itself, which may be mapped into multiple instances' memory spaces.

```
char* str = "Hi!";
str[0] = 'h';
//segmentation fault!
```

BUT WHY?

WHY VIRTUALISE THE ADDRESS SPACE? (3/3)

EFFICIENCY, SECURITY, CONCURRENCY, <u>FLEXIBILITY</u>, …

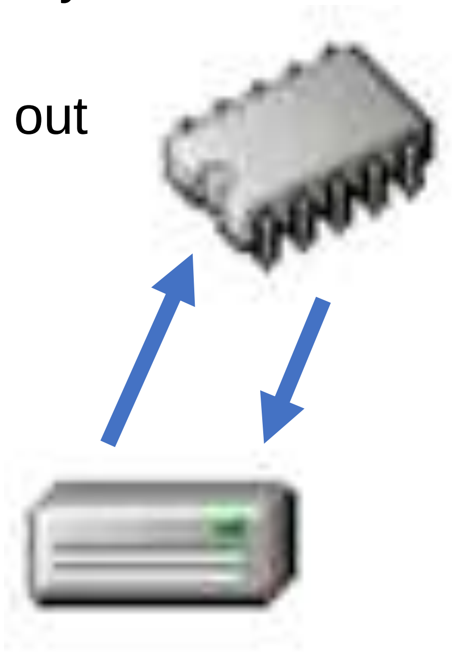
- Relocation:
  - If a program must be swapped out to secondary storage, and is swapped back into main memory later on…
  - The same location in main memory might not be available.
    - Might be relocated to a different part of physical memory.
  - Relocating a program means the physical memory addresses are all different.
  - All the program's instructions and variables now have different physical memory addresses.
  - Dealing with logical addresses rather than physical addresses means the program's pointers remain valid
  - The program can continue to run normally regardless of where it is located in physical memory.
    - Program does not depend on being loaded into a particular location.

SWAPPING

MAIN MEMORY ↔ SECONDARY STORAGE

WHAT IS SWAPPING?

- Moving chunks of data between main memory and secondary (disk) storage.
- Process data which is not currently needed can be moved out to the disk
- Moved back into physical memory when needed



IS IT USEFUL?

- Allows more complete utilisation of main memory
- But secondary storage is very **S L O W**
- Need to avoid swapping too often.
- **Thrashing:** a condition where the time spent of swapping data exceeds time spent executing instructions.

TWO APPROACHES

TO VIRTUAL MEMORY ALLOCATION

PAGING

Memory is **partitioned** into **fixed sized** chunks

SEGMENTATION

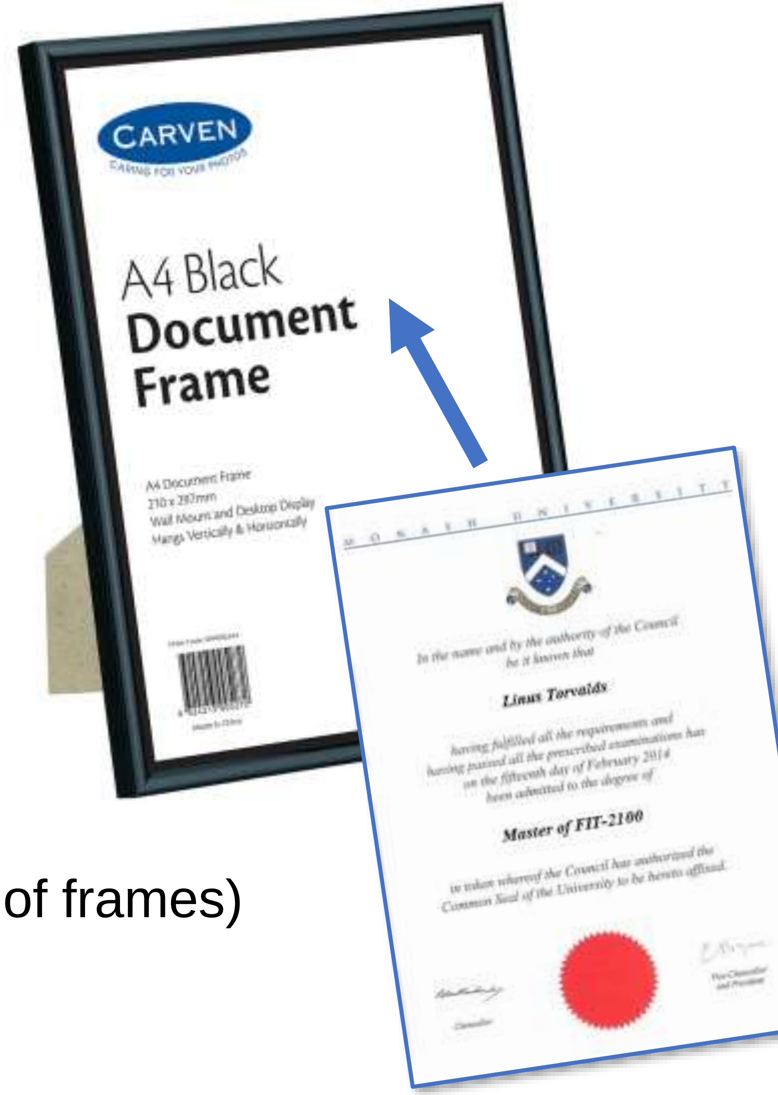
Memory is managed as multiple **segments** of **different sizes**

PAGED VIRTUAL MEMORY

MONASH University

FRAMES AND PAGES

PAGED VIRTUAL MEMORY



PAGES FIT INTO FRAMES

- **Physical memory** is partitioned into equal sized **frames.**
- A **page** is a chunk of data that can fit into a frame
- All frames are of equal size, and a page is the same size as a frame.

- Some pages might be in physical memory (loaded into frames)
- Other pages might be **swapped out** to secondary storage (taken out of frames)
  - To make way for other pages to be swapped in.

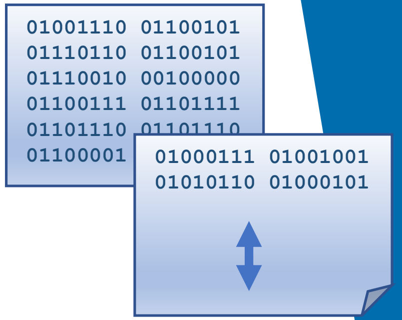
- The OS kernel maintains a **page table** to keep track of which pages are in which frames.

FRAGMENTATION

PAGED VIRTUAL MEMORY

INTERNAL FRAGMENTATION

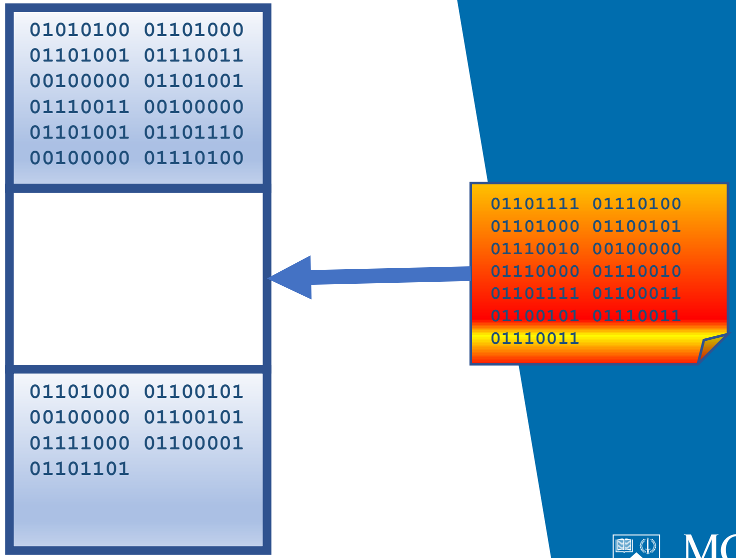
- A process may occupy one or more frames in memory.
- If an entire page is not needed, the space cannot be used by another process
- Since all pages are fixed size, any excess space is wasted.



← Wasted internal space!

NO EXTERNAL FRAGMENTATION

- Can space **between** page frames be wasted?
- No. All pages are equally sized, so any empty frame in memory can **always** be allocated a whole page.
  - ➔ no external fragmentation
- *…but this is not true for memory systems that use* ***segmentation*** *(see later)*



LOGICAL ADDRESSES

PAGED VIRTUAL MEMORY

THE MEMORY ADDRESS CONTAINS A PAGE NUMBER AND AN OFFSET

Example:

```
00000000001010000000000000000010000
Page number 10    Offset 16 bytes
```

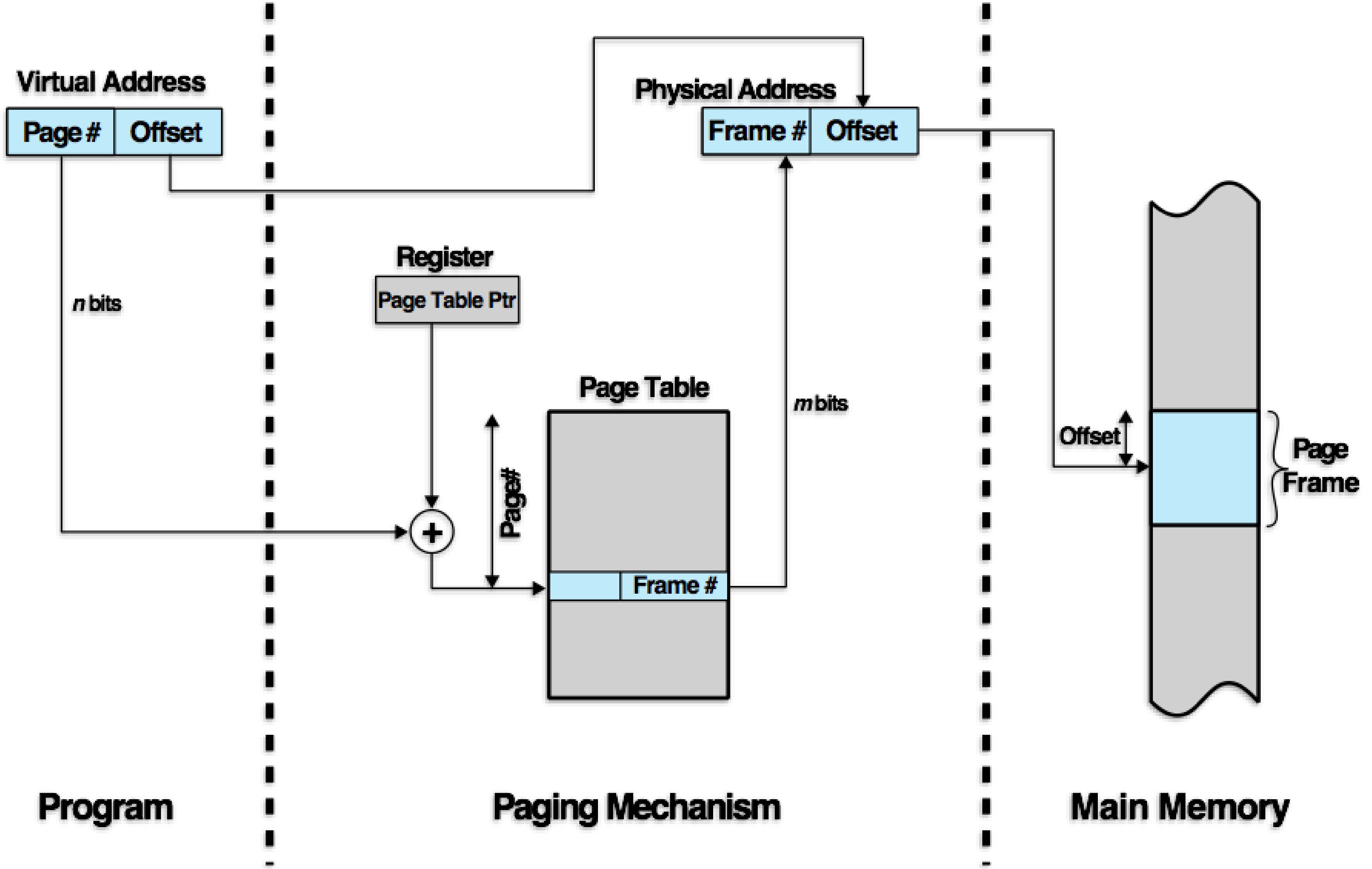
The most significant bits in the **logical address** are used for the page number. The remaining bits are an **offset** from the start of the page (i.e. position inside the page)

The address is structured so the **number of offset bits** matches the addressable range of the **page size**. Different systems may use a different page size.

- It is not possible to specify an offset that goes 'out of bounds' of the specified page.

ADDRESS TRANSLATION (PAGING)

Change (logical) page number to (physical) frame number → **physical address.**



WHAT IS A PAGE FAULT?

PAGED VIRTUAL MEMORY

WHAT HAPPENS IF A PAGE DOES NOT HAVE A FRAME IN THE PAGE TABLE?

- It means the page does not have a physical address in memory

‘PAGE FAULT’!

- The requested **logical** memory address is **valid**, but the page is in **secondary storage**. Must be loaded in to a frame.

IS A PAGE FAULT ‘BAD’?

- If all the frames are already full, another page needs to be swapped out of main memory to make room for the needed one.
- We need a ‘page replacement algorithm’ to determine which page should be swapped out to make way for the one that’s needed.
- **NOTE**: even if there **is** an empty frame to load the needed page into, it is still a **page fault** if the page was not found in physical memory.

MONASH University

PAGE REPLACEMENT ALGORITHMS

PAGED VIRTUAL MEMORY

PAGE FAULT == PAGE NOT FOUND IN PHYSICAL MEMORY

- If we get a page fault, we need to load a page from disk into a frame in RAM.
- What if all the frames are already full?
- Need a strategy for knowing which page to swap out.
- AIM:
    - Swap out a page that won't be needed again for a long time
        - PREVENT **THRASHING**
    - Try to make a 'fast guess'. Don't try to be 'too clever'.
        - MORE ACCURATE PREDICTION → MORE COMPUTATION TIME NEEDED
        - LESS ACCURATE PREDICITON → MORE PAGE SWAPS NEEDED
        - TRY TO FIND A BALANCE.

MONASH University

FIRST IN FIRST OUT (FIFO)

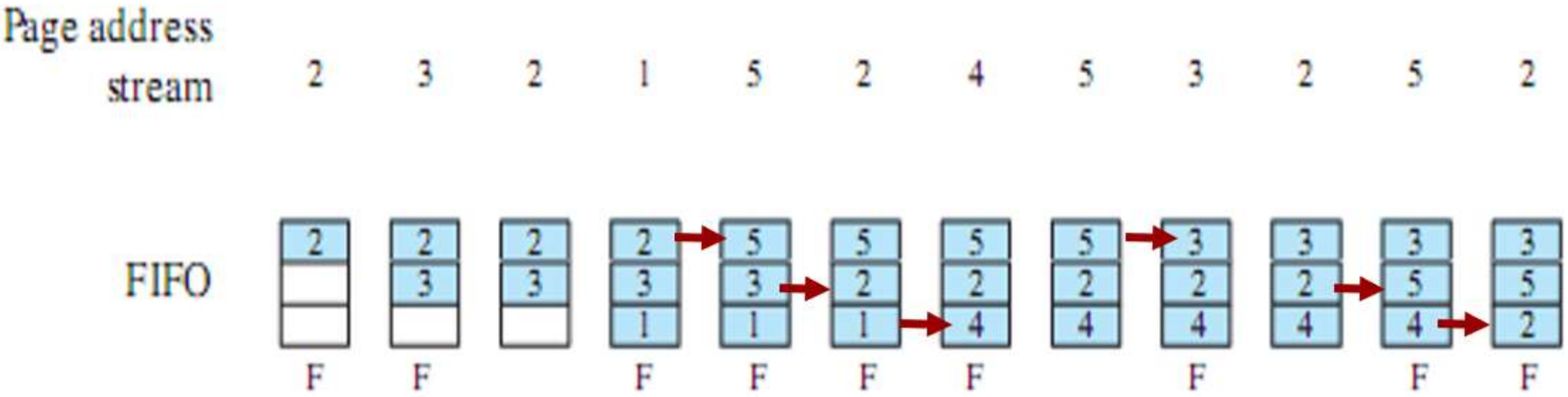
SIMPLEST REPLACEMENT POLICY TO IMPLEMENT

PAGE THAT HAS BEEN IN MEMORY THE LONGEST IS REPLACED

Treat frames allocated to processes as a ‘queue’ for replacement.

Pages are removed in round-robin style.

EXAMPLE



F= page fault

Total **9 page faults**.
(or 6 page faults after all frames filled)

MONASH University

LEAST RECENTLY USED (LRU)

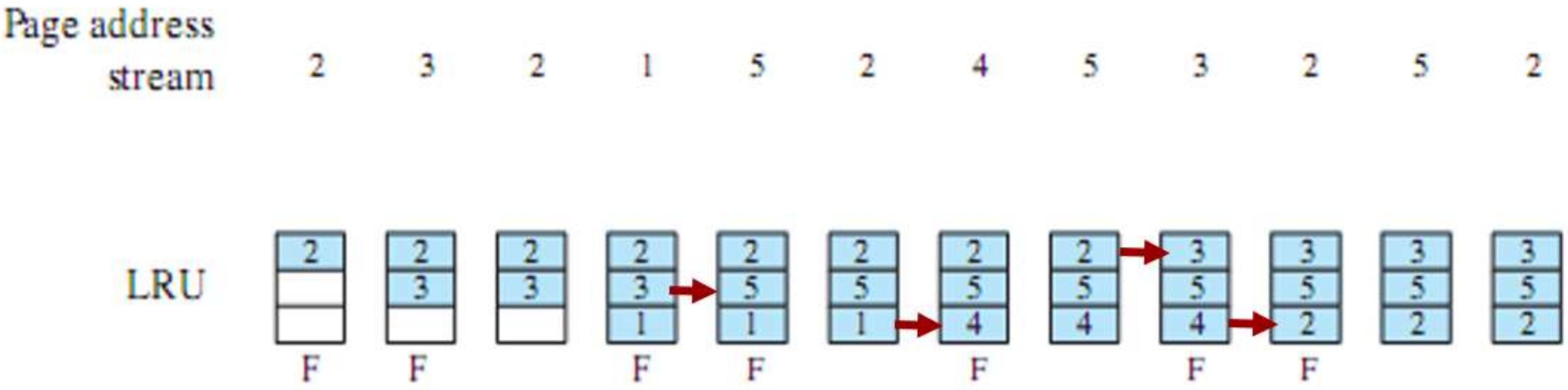
DIFFICULT TO IMPLEMENT

REPLACE THE PAGE THAT HAS NOT BEEN REFERENCED FOR THE LONGEST TIME

By the **principle of locality**, pages that have been referenced recently are likely to be referenced again in near future.

A **lot of overhead** to maintain/check the time since last reference for every page.

EXAMPLE



F= page fault

Total **7 page faults**.
(or 4 page faults after all frames filled)

MONASH University

OPTIMAL POLICY (OPT)

THEORETICAL BEST CASE

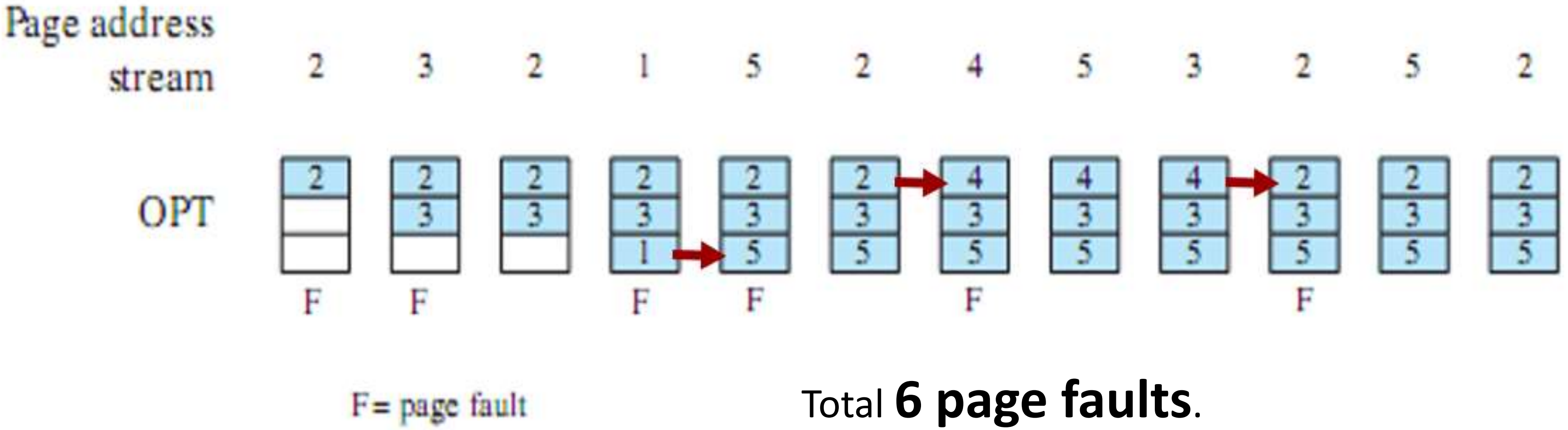
REPLACE THE PAGE THAT <u>WILL NOT BE REFERENCED</u> FOR THE <u>LONGEST TIME</u>

Produces the smallest possible number of page faults.

Impossible to implement in practice: the operating system cannot see the future.

- If we knew everything the process would do in the future, we would not need to run the program!

EXAMPLE



Total **6 page faults**.
(or 3 page faults after all frames filled)

PAGE REPLACEMENT ALGORITHMS

SUMMARY

PERFORMANCE

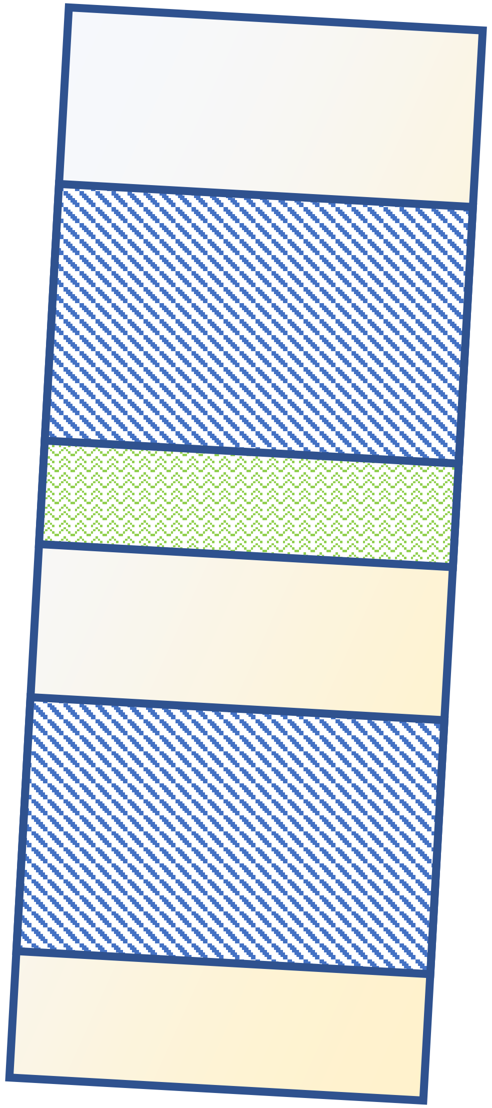
	PAGE FAULTS	COMPLEXITY
FIFO	9	Simplest
LRU	7	Complex
OPT	6	‘Impossible’

PAGE REPLACEMENT ALGORITHMS ARE NOT PERFECT

- Page replacement is a trade-off between **minimising** page faults and **minimising** computational complexity.
- Page replacement can happen **very often** (whenever data is swapped in). Identifying a page to replace must to be done in **as few instructions as possible** while still giving a ‘good enough’ result.
    - So that the program’s instructions can be executed instead!

MONASH University

SEGMENTED VIRTUAL MEMORY

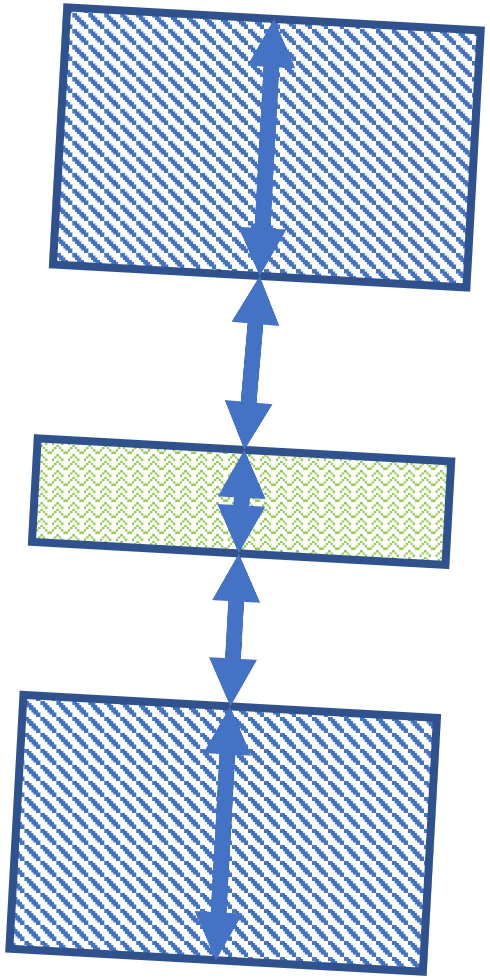


SEGMENTS

SEGMENTED VIRTUAL MEMORY

WHAT IS A SEGMENT?

- A process is allocated one or more logical **segments** of memory.
- A segment can be swapped in or out, just like a page.
- Segments can vary in size, up to a defined maximum size.
- There are no fixed ‘frames’ in physical memory: segments are placed where space is available.
- Swapping out a segment leaves a ‘hole’ where another segment may fit.



FRAGMENTATION

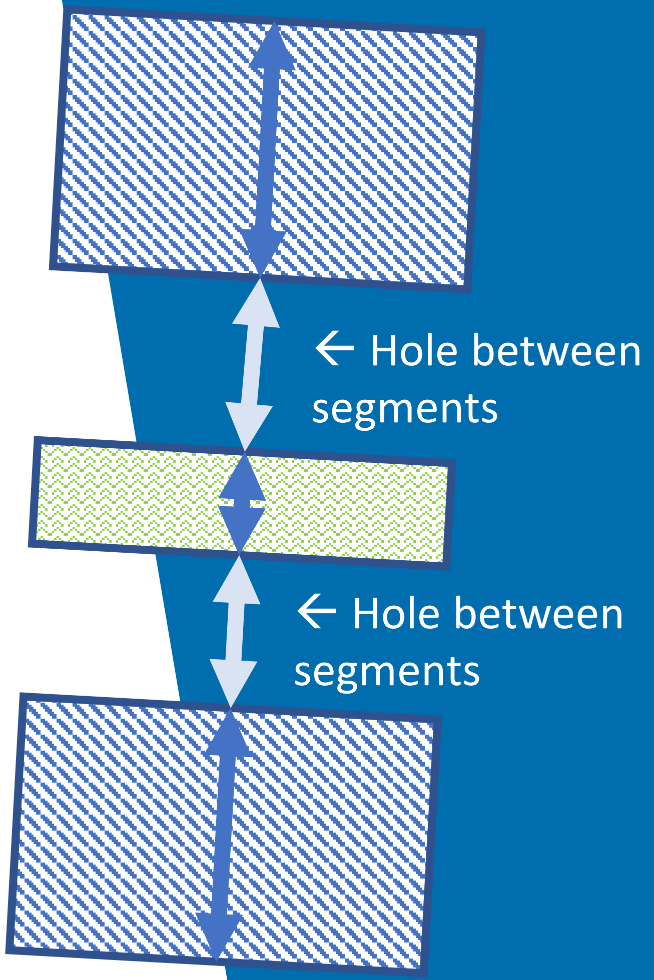
SEGMENTED VIRTUAL MEMORY

NO INTERNAL FRAGMENTATION

- A segment may grow or shrink dynamically depending on required size. There is no ‘wasted’ space within a segment that cannot be used.

EXTERNAL FRAGMENTATION

- Can space **between** segments be wasted? **YES.**
  - A small segment may be swapped out (or a small process may exit the system).
  - The **hole** left behind may be too small to reasonably fit another segment
  - Even though the space is available, it is not used.
- **Compaction** is the technique of periodically re-arranging segments to remove gaps. Computationally expensive.



PLACEMENT ALGORITHMS

SEGMENTED VIRTUAL MEMORY

Best-fit	First-fit	Next-fit
• chooses the block that is **closest in size** to the request	• begins to **scan memory from the beginning** and chooses the first available block that is large enough	• begins to **scan memory from the location of the last placement** and chooses the next available block that is large enough

MONASH University

LOGICAL ADDRESSES

SEGMENTED VIRTUAL MEMORY

THE MEMORY ADDRESS CONTAINS A SEGMENT NUMBER AND AN OFFSET (Just like paging)

Example:

```
00000000011000000000000000010001
Seg number 12    Offset 17 bytes
```

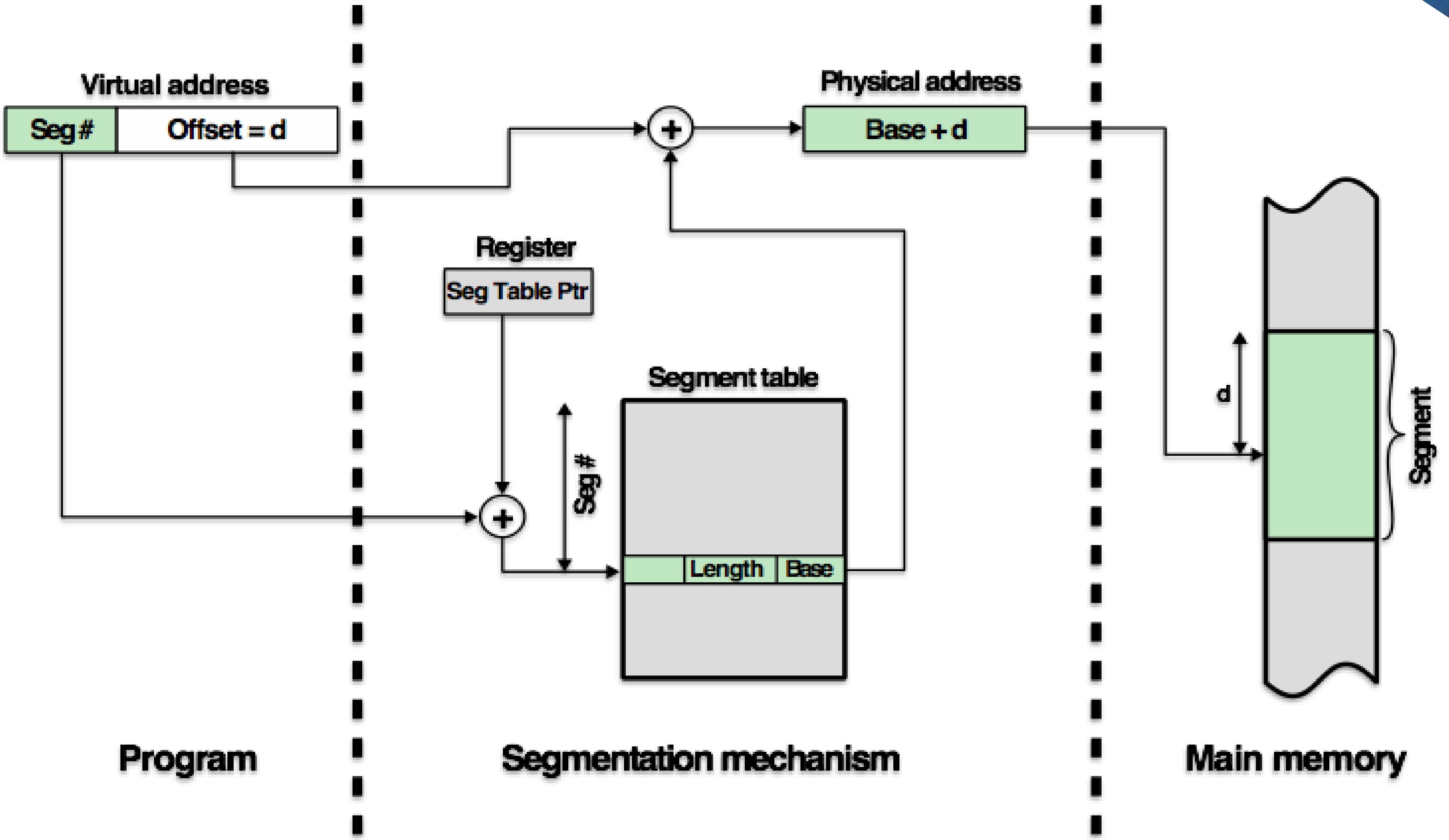
The most significant bits in the **logical address** are used for the **segment number**. The remaining bits are an **offset** from the **start** of the segment in physical memory.

Segments can be different sizes. **If the offset goes past the end of a segment**, the logical address is invalid.

'SEGMENTATION FAULT'!!! ← Process is usually terminated.

MONASH University

ADDRESS TRANSLATION (SEGMENTATION)



SHARED MEMORY

A SEGMENT IN MEMORY THAT IS ALLOCATED TO MULTIPLE PROCESSES

- A memory segment may be shared among multiple processes
- Like with multi-threading, multiple processes can access each other's memory
- Unlike with multi-threading, it does not happen automatically.
    - Must manually request a new block of shared memory to be allocated.
    - Required number of bytes must be specified.
    - Must manually assign a data structure to the shared block of bytes.
- There is no concurrency protection offered. Mutual exclusion mechanisms like semaphores must be used.

- Shared memory is an important tool for inter-process communication. Even unrelated processes can be written to share memory.

SUMMARY (LECTURE 9)

VIRTUAL MEMORY

- We now understand why **virtualised** memory is useful
- Paging and segmentation are different approaches to virtual memory and create different challenges.
- We understand how to translate a logical memory address into a physical address by adding the offset to the starting address of the page/segment in physical memory.
- We have discussed three different page-replacement algorithms.

- **Next week: Inter-process communication**