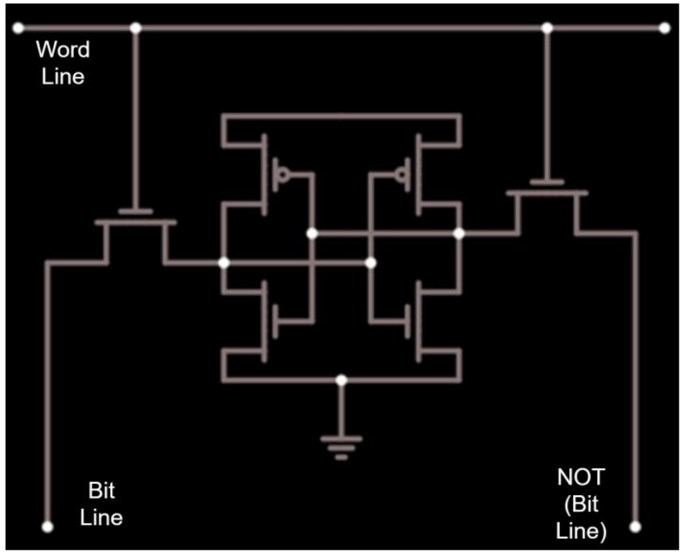
# MICROPROCESSOR SYSTEMS (IAS0430)

Department of Computer Systems
Tallinn University of Technology

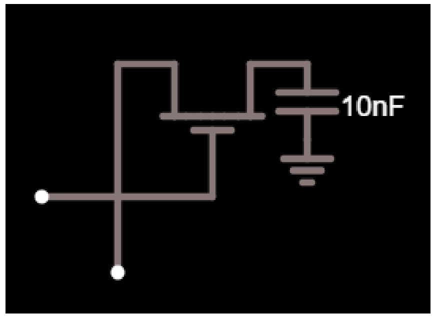
## MEMORY HIERARCHY

- **What is computer memory?**
  - Computer memory is any **physical device designed using integrated circuits** for the purpose of **storing and retrieving data and/or instructions** for short term use (**temporarily**) or long term storage (**permanently**).
  - **Memory Types:**
    - **SRAM – Static RAM**
    - **DRAM – Dynamic RAM**
    - Magnetic Memory
    - Optical Memory
  - **Memory Devices**
    - Register File

**SRAM**



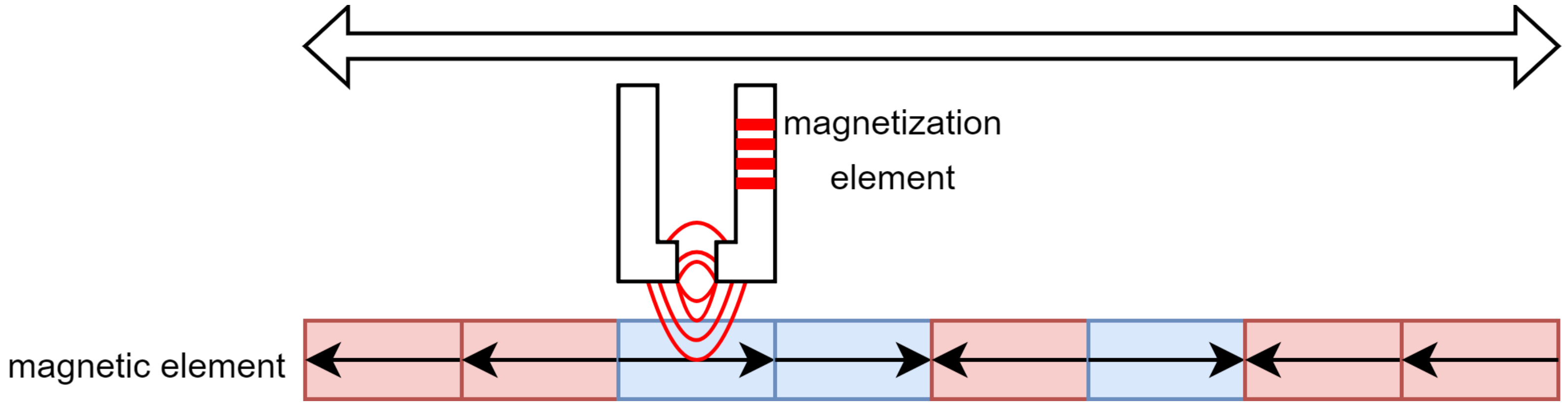
**DRAM**



## MEMORY HIERARCHY

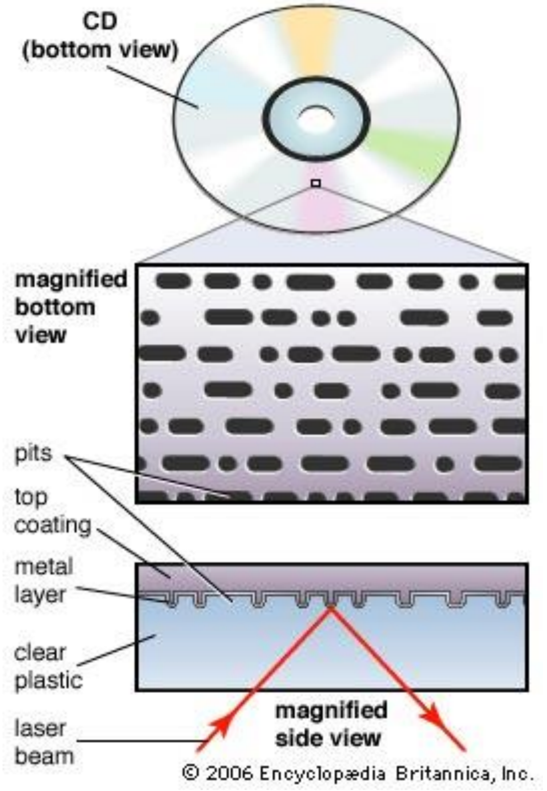
- **What is computer memory?**
  - Computer memory is any **physical device designed using integrated circuits** for the purpose of **storing and retrieving data and/or instructions** for short term use (**temporarily**) or long term storage (**permanently**).

  - **Memory Types:**
    - SRAM – Static RAM
    - DRAM – Dynamic RAM
    - **Magnetic Memory**
    - Optical Memory
  - **Memory Devices**
    - Register File



## MEMORY HIERARCHY

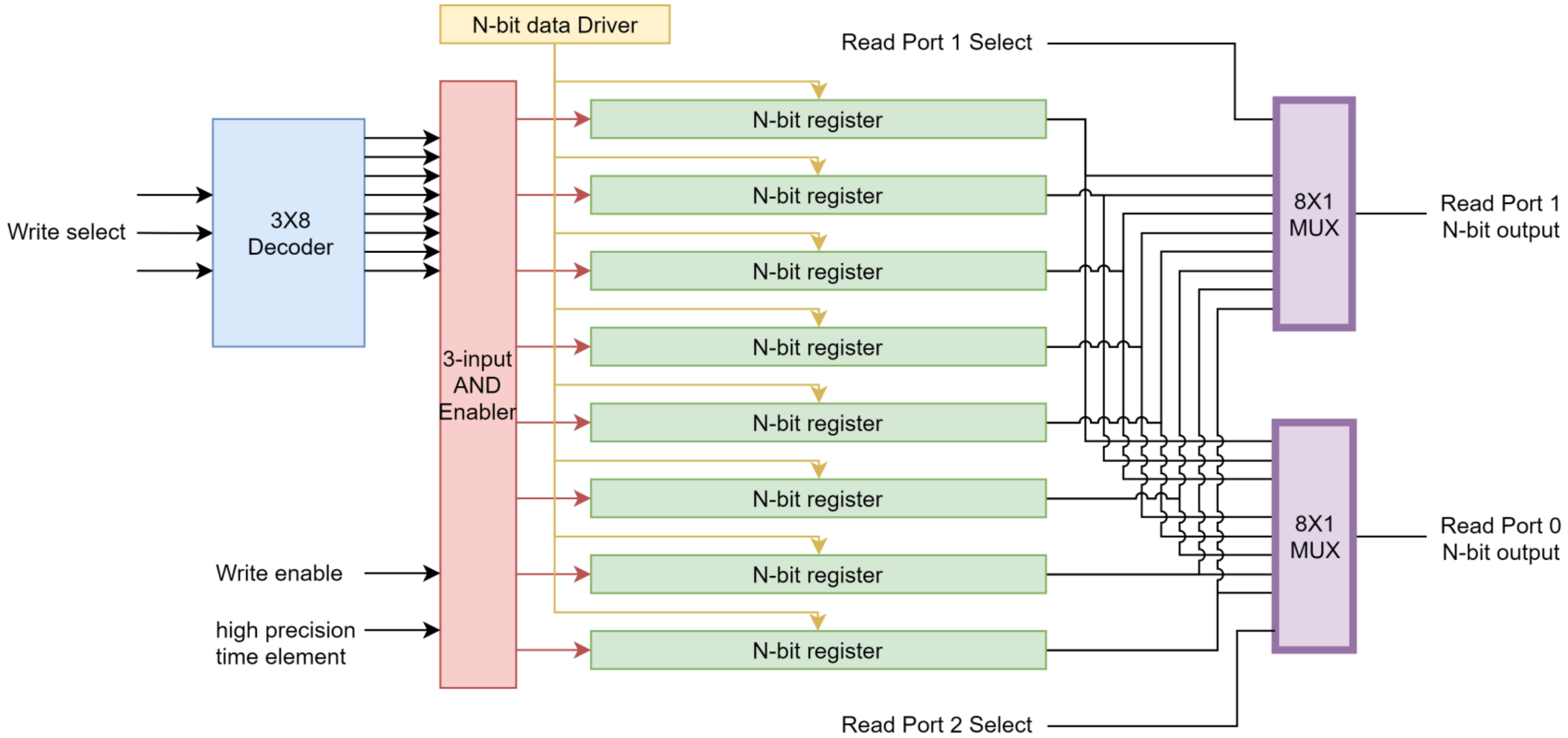
- **What is computer memory?**
  - Computer memory is any **physical device designed using integrated circuits** for the purpose of **storing and retrieving data and/or instructions** for short term use (**temporarily**) or long term storage (**permanently**).
  - **Memory Types:**
    - SRAM – Static RAM
    - DRAM – Dynamic RAM
    - Magnetic Memory
    - **Optical Memory**
  - **Memory Devices**
    - Register File



## MEMORY HIERARCHY

- **What is computer memory?**
  - Computer memory is any **physical device designed using integrated circuits** for the purpose of **storing and retrieving data and/or instructions** for short term use (**temporarily**) or long term storage (**permanently**).

  - **Memory Types:**
    - SRAM – Static RAM
    - DRAM – Dynamic RAM
    - Magnetic Memory
    - Optical Memory
  - **Memory Devices**
    - **Register File**



## MEMORY HIERARCHY

- **Caches recap**
  - The cache is a **static random access memory designed to operate and to be accesses by the CPU at a much faster rates than the Main Memory**
  - It is located right inside the CPU.
  - It bridges the speed gap between the Main memory and the CPU.
    - If data is found in the cache, this is called a cache **hit**
    - If data is not found in the cache, this is called a cache **Read miss**.
    - If data is being written into the cache in a location where data is already stored, this is called a cache **Write miss**. – we will get to see more types of misses later.
    - If a cache miss occurs, **the data is then requested from the slower main memory**

## MEMORY HIERARCHY

- **Caches recap**
- The principle of **cache locality**:
  - **Spatial locality:**
    - When an address is accessed, it **is highly likely that the CPU wants to access the sequential memory locations located after that address as well**.
  - **Temporal locality**:
    - When an address is accessed**, it is highly likely that the CPU wants to access the that address again later**.

  - **Locality** can help **determine how the cache should be accessed** and what **methods to use in cache replacement policies** for different programs– remember this for later on…

## MEMORY HIERARCHY

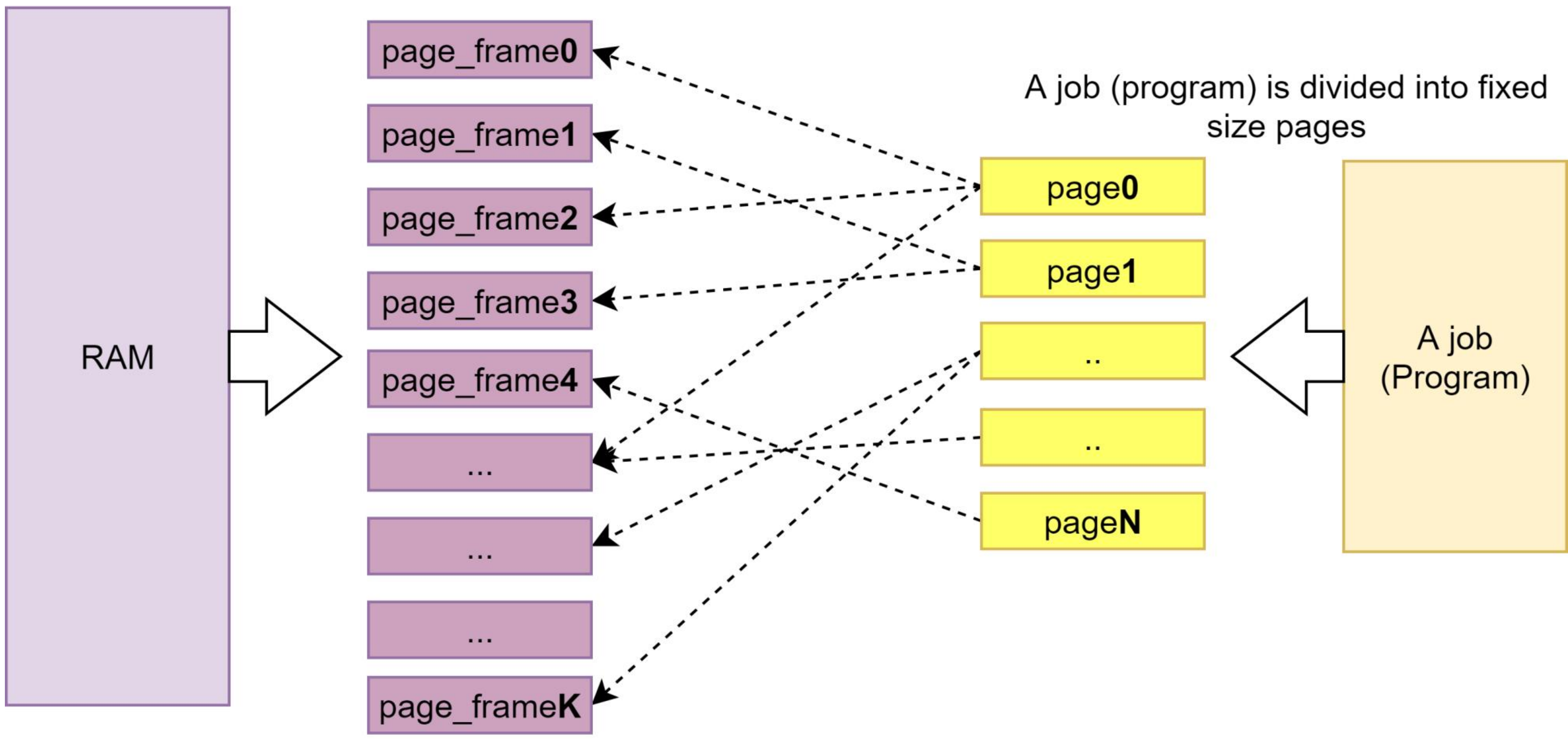
- **Cache Addressing**
    - Cache addressing **refers to the way in which a cache address is divided**.
    - Caches are divided into **sets** that are used to store a **block** of data.
    - **Blocks** are collections of data. This data is called a **word**.
    - A block can store many words, that in tern represent the different types of data that the processor can process.
    - Caches can be organized in a cache system when used with multi-core systems
        - Low level (L1) caches can be specialized to each core
            - Instruction Cache
            - Data Cache
        - Higher levels of cache are usually shared caches among two or more cores
        - Caches also require methods of keeping copies of data updated
            - WRITE BACK
            - WRITE THROUGH

## MEMORY HIERARCHY

- **Main memory (RAM)**
  - Main memory is a Random Access Memory that holds – or should hold – all the information needed to finish an execution.
    - It contains complete programs
      - This includes all the instructions and data that a program requires to run.
  - **Programs that exist in the RAM are the programs that are currently executing.**
    - If a program is not executing or finishes execution, it is evicted from the RAM.
  - Inside the RAM, programs are divided into large blocks of data called **Pages**.
    - A **Page** is a ***fixed-size, large* block of memory**, positioned at some starting address in a program's address space. The page size is set by the hardware design. The size of a page can range between a **4KB (32-bit x86)** to **32KB (64-bit x86)** page size and so on …
    - The set of pages which a program has access to forms its **logical address space**.
    - The RAM itself is divided into **page frames**. These page frames can fit one page at a time.
      - Each page frame is given an identifier. These identifiers are unique to all the page frames inside the RAM.

## MEMORY HIERARCHY

Main memory is divided into fixed size pages frames that can only fit one page at a time



Any page can be allocated to an page frame.

## MEMORY HIERARCHY

- **RAM**
  - A **hardware Memory Decoder** called the **Memory Management Unit** (or **MMU**) maps the memory addresses to pages requested by a program to a set of page frames in physical memory (RAM).
  - **The decoder can also set protections on each of the program's page**.
    - A program's page may be marked as **read-only, read-write, invalid, or kernel-mode access only.**
    - The MMU must be aware of such protections to manage access to those pages.
  - The **MMU** takes a program and divides it into a set of pages.
    - Each page will be fixed size to a range of locations in memory.
      - **Say we have a program that is 45KB in size. How many pages do we need to store this program into a memory with a 4KB page size?**

## MEMORY HIERARCHY

- **RAM**
  - A **hardware Memory Decoder** called the **Memory Management Unit** (or **MMU**) maps the memory addresses to pages requested by a program to a set of page frames in physical memory (RAM).
  - **The decoder can also set protections on each of the program's page**.
    - A program's page may be marked as **read-only, read-write, invalid, or kernel-mode access only.**
    - The MMU must be aware of such protections to manage access to those pages.
  - The **MMU** takes a program and divides it into a set of pages.
    - Each page will be fixed size to a range of locations in memory.
      - **Say we have a program that is 45KB in size. How many pages do we need to store this program into a memory with a 4KB page size?**
      - simply divide the size of program by the size of the page
        - **45 / 4 = 11.25 ~ 12 pages**

## MEMORY HIERARCHY

- **RAM**
  - Ok, but how do we keep track of that! If we divide the program into pages, and different pages are allocated into memory separately, how do we know where each page is?
  - Depending on what type of scheme is used to manage the RAM, there are three ways to locate pages:
  - **<u>Paged Map Allocation</u>**

    **1 - When the MMU divides the program into pages, it assigns an identifier to each page – this identifier is usually a number**

    - Those identifiers are stored in what is called the **Page Map Table.**
      - The page map table is a small partition inside the main memory device that stores **information on the identifier of each page** and **what page frame it is stored in.**
      - Each page map table is unique for each job. Meaning that every job has its own separate page map table that it does not share with other jobs

    **2 - Once the MMU divides a program to pages, it creates a table in small partition of memory called the Job Table.**

      - The Job table stores information on **the size of each program** and **where to find the Page Map Table of that job inside the memory.**

## MEMORY HIERARCHY

- **RAM**
  - Now that the MMU knows where each page is stored, and which page belongs to which job, it need to keep track of which page frames are available or being used.
  - **3 – The MMU creates a Memory Map Table to keep track of the use of the page frames.**
    - The Memory Map Table is a small partition of memory where information on the status of pages frames.
    - This means that ALL PAGES ARE LOADED TO MEMEORY IF THE JOB IS BEING PROCESSED.

| Job Table | |
|---|---|
| job size | location |
| 300KB | 0xa1b2fc22 |
| 530KB | 0xf411fd54 |
| ... | ... |
| | |
| | |
| | |
| | |
| | |

| Page Map Table | |
|---|---|
| page ID | page frame ID |
| 6 | 214 |
| 4 | 854 |
| 1 | 252 |
| 2 | 345 |
| ... | ... |
| | |
| | |
| | |

| Memory Map Table | |
|---|---|
| page frame ID | status |
| 214 | BUSY |
| 366 | FREE |
| 255 | FREE |
| ... | ... |
| | |
| | |
| | |
| | |

## MEMORY HIERARCHY

- **RAM**
  - **Demand Paging**
    - It follows the same concept of Paged Map Allocation, except only pages that we need to execute are loaded into memory.
    - Unlike Paged Map Allocation, we do not load the entirety of the job into RAM, we only load the page we intend to execute part of – hence the name, on demand use.
    - To keep track of this, we need to add some more information to the **Page Map Table.**
      - We add three more columns:
        - Status: specifies if the page is in memory or not
        - Modified: specifies if any data in the page has been changed
        - Reference: tells if any information inside that page was referenced/used recenetly.

| Job Table | |
|---|---|
| job size | location |
| 300KB | 0xa1b2fc22 |
| 530KB | 0xf411fd54 |
| ... | ... |
| | |
| | |
| | |
| | |
| | |

| Memory Map Table | |
|---|---|
| page frame ID | status |
| 214 | BUSY |
| 366 | FREE |
| 255 | FREE |
| ... | ... |
| | |
| | |
| | |
| | |

| Page Map Table -- for on-demand paging | | | | |
|---|---|---|---|---|
| page ID | STATUS | MODIFIED | REF | page frame ID |
| 6 | Y | 0 | 0 | 214 |
| 4 | Y | 1 | 0 | 854 |
| 1 | N | 0 | 1 | 252 |
| 2 | N | 0 | 1 | 345 |
| ... | ... | ... | ... | ... |
| | | | | |
| | | | | |
| | | | | |

## MEMORY HIERARCHY

- **RAM**
  - **Demand Paging**
    - **What do we do if the RAM is full and we have no place to put new pages that we need?**
      - We use replacement policies for the pages

## MEMORY HIERARCHY

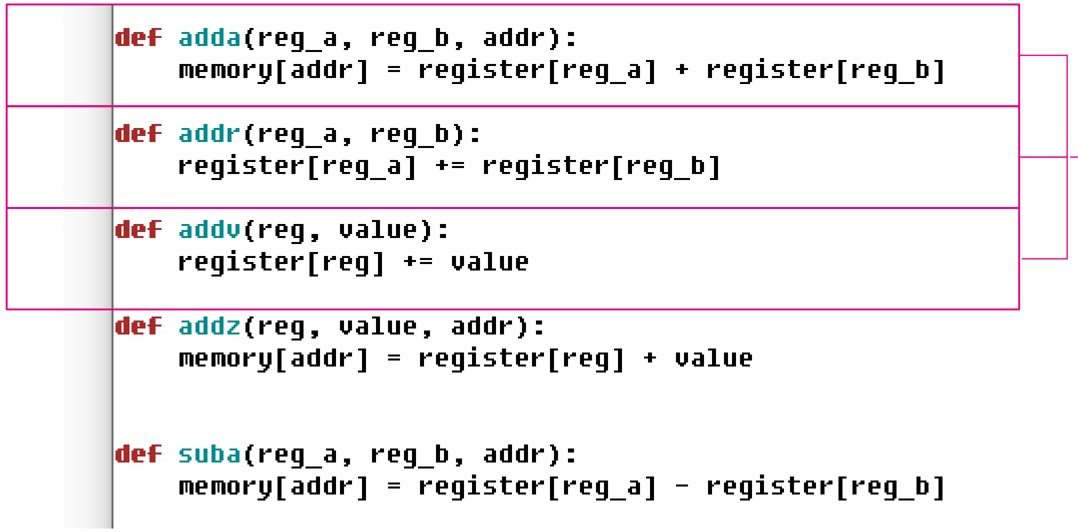
- **RAM**
  - **Demand Paging**
    - **What do we do if the RAM is full and we have no place to put new pages that we need?**
      - We use replacement policies for the pages
        - First In First Out
        - Least Recently Used
        - Least Frequently Used
        - Most Recently Used
    - These replacement policies will be discussed next week

## MEMORY HIERARCHY

- RAM
  - Segmented Memeory Allocation
    - What if we want to break the pages into even smaller chunks?

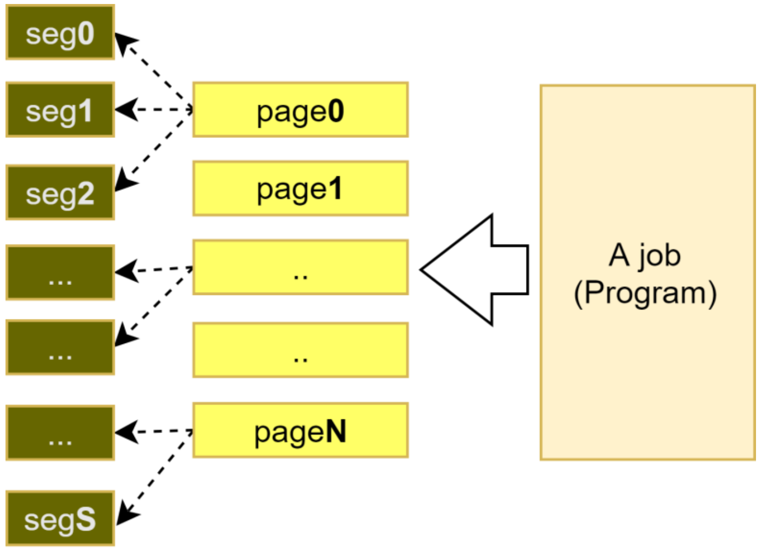
## MEMORY HIERARCHY

- RAM
  - <u>**Segmented Memeory Allocation**</u>
    - **What if we want to break the pages into even smaller chunks?**
      - **It is possible!**
        - **Those smaller chunks are called segments**
    - Unlike Page Map Allocation and Demand Paging, this scheme take into account the structure of the program.



```
def adda(reg_a, reg_b, addr):
    memory[addr] = register[reg_a] + register[reg_b]

def addr(reg_a, reg_b):
    register[reg_a] += register[reg_b]

def addv(reg, value):
    register[reg] += value

def addz(reg, value, addr):
    memory[addr] = register[reg] + value


def suba(reg_a, reg_b, addr):
    memory[addr] = register[reg_a] - register[reg_b]
```

segments



# MEMORY HIERARCHY

- **RAM**
  - **Segmented Memeory Allocation**
    - Instead of having pages that are chunks of data and loading them all at once, we can now specify which data goes to RAM when needed.
      - Pages become divided into segments
      - Segment sizes vary, which adds more for the **MMU** to manage them
    - This scheme works similar to **Page Map Allocation**
      - We have the same tables, except that we will need a Segment Map Table to specify in which page is each segment

| Job Table | |
|---|---|
| job size | location |
| 300KB | 0xa1b2fc22 |
| 530KB | 0xf411fd54 |
| ... | ... |
| | |
| | |
| | |
| | |
| | |

| Page Map Table | |
|---|---|
| page ID | page frame ID |
| 6 | 214 |
| 4 | 854 |
| 1 | 252 |
| 2 | 345 |
| ... | ... |
| | |
| | |
| | |

| Segment Map Table | |
|---|---|
| Segment ID | page pointer |
| 4 | 6 |
| 2 | 4 |
| 4 | 4 |
| 3 | 2 |
| 6 | 2 |
| 1 | 1 |
| ... | ... |
| | |

| Memory Map Table | |
|---|---|
| page frame ID | status |
| 214 | BUSY |
| 366 | FREE |
| 255 | FREE |
| ... | ... |
| | |
| | |
| | |
| | |

TALLINN UNIVERSITY OF TECHNOLOGY

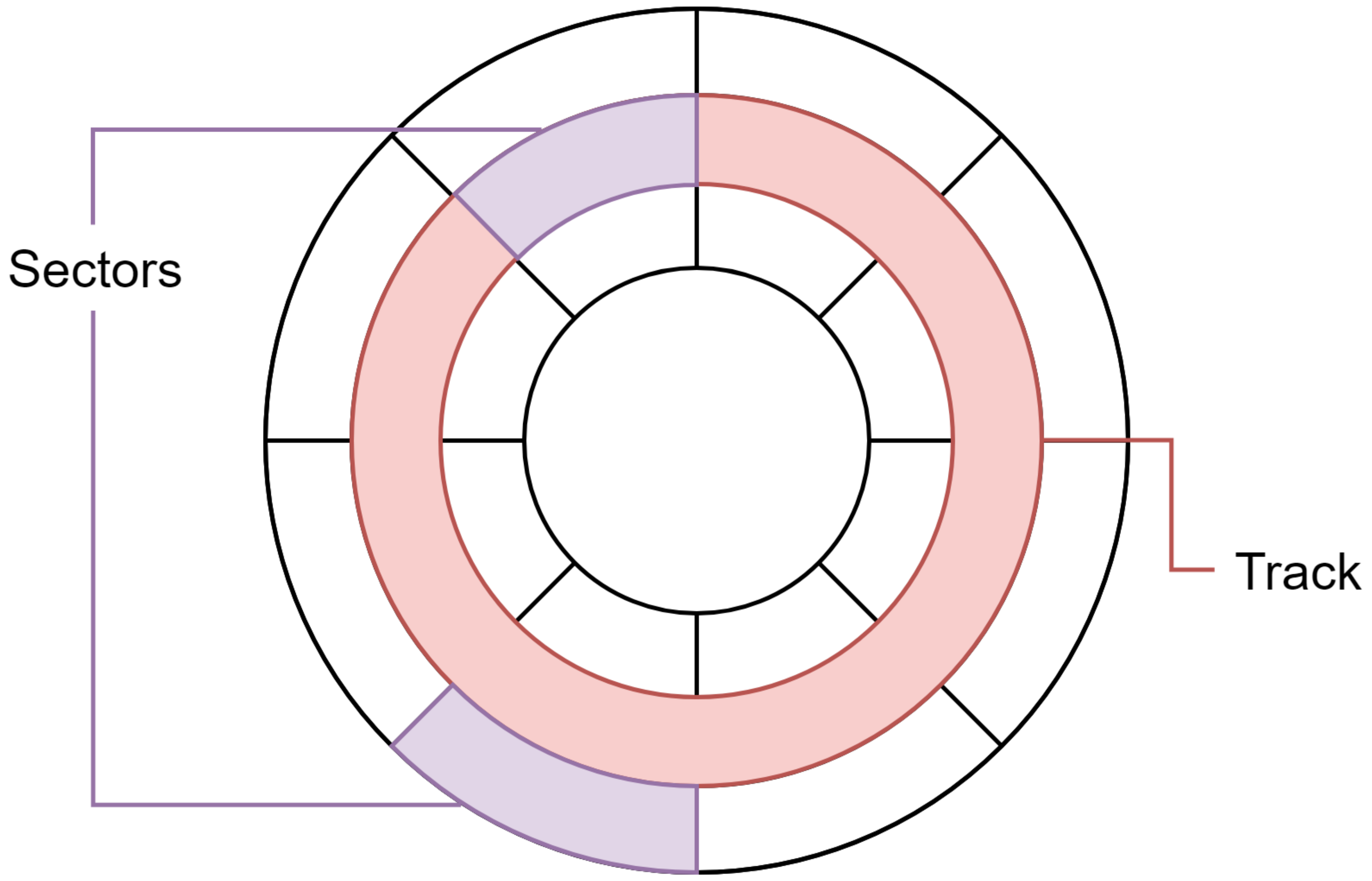
## MEMORY HIERARCHY

- **The Hard Disk Drive**
  - The Hard Disk Drive is among the longest used forms of permanent storage.
    - It was introduced in 1956 and is still used today
  - The HDD is a magnetic type of storage, it uses magnetization of a high speed rotating disk to store information
  - Data on the HDD is stored as **magnetic patterns** formed as a **cluster** of small surface area locations on the physical disk called **magnetic grains.**
    - **On the HDD, a bit is represented by one cluster of magnetic grains that have the same magnetic rotation in a fixed location.**

  - The physical disk itself is divided into areas called **tracks**. Tracks are then divided into smaller parts called **sectors.**
    - Since the HDD relies on mechanical part called the READER to write and read data, dividing the disk into tracks and sectors, allows the reader to locate areas on the disk faster. This will result in faster retrieve of information as well as faster writing into the disk.

## MEMORY HIERARCHY

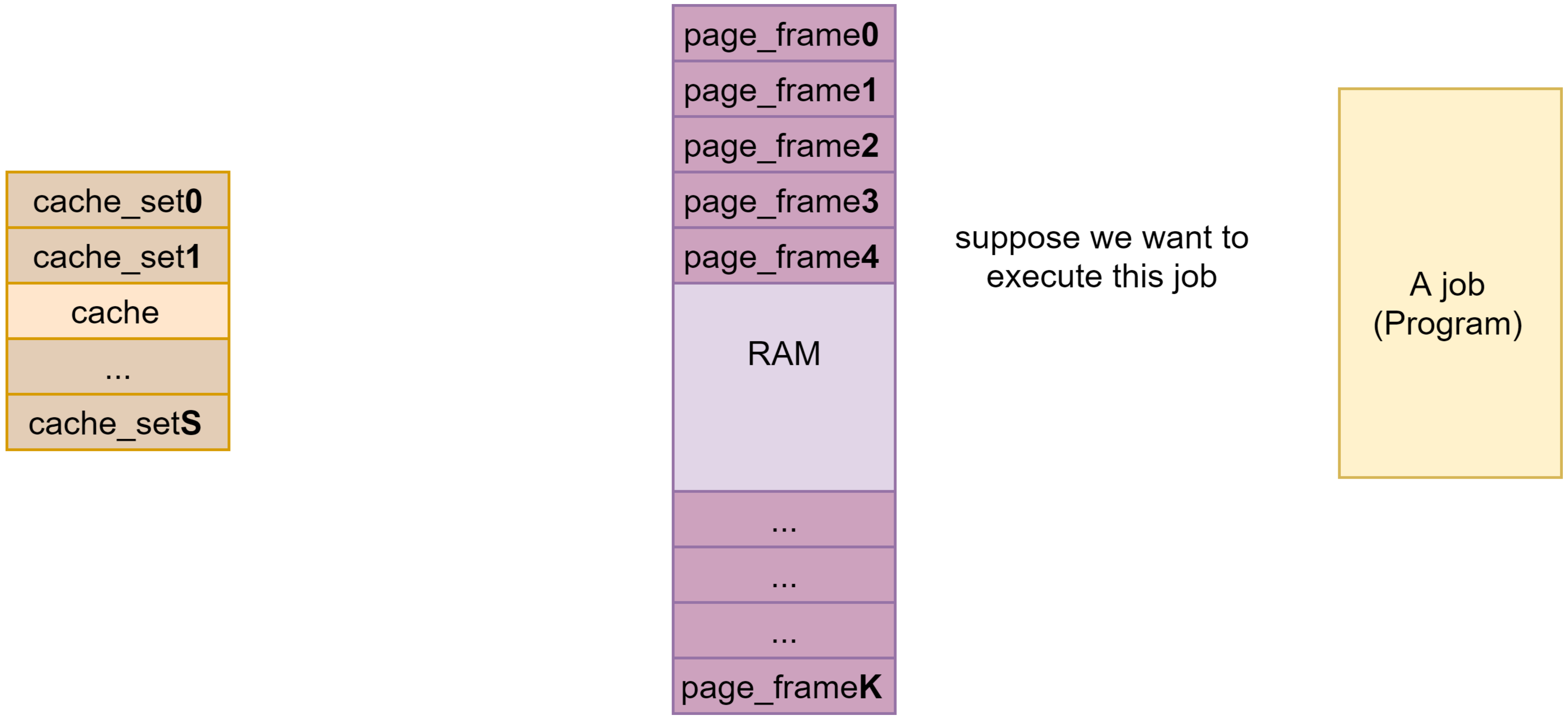
- **The Hard Disk Drive**



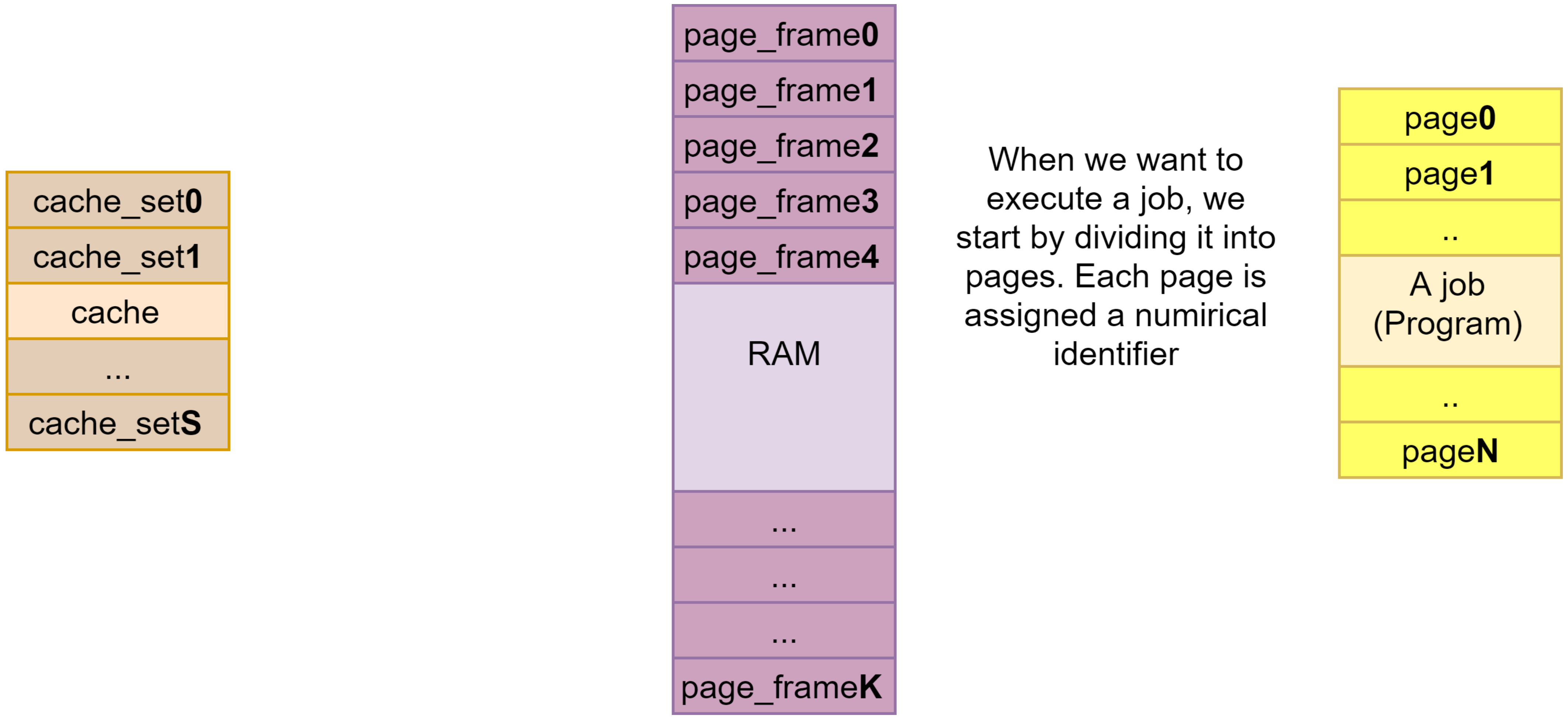
## MEMORY HIERARCHY

- **Where are we so far**
  - We know that a job is divided into pages
  - Pages are loaded to the RAM once needed
    - RAM is divided into page frames
  - Those pages are then divided into blocks
  - Those blocks are loaded into cache once needed
    - Caches are divided into sets
  - Blocks contain words

## MEMORY HIERARCHY

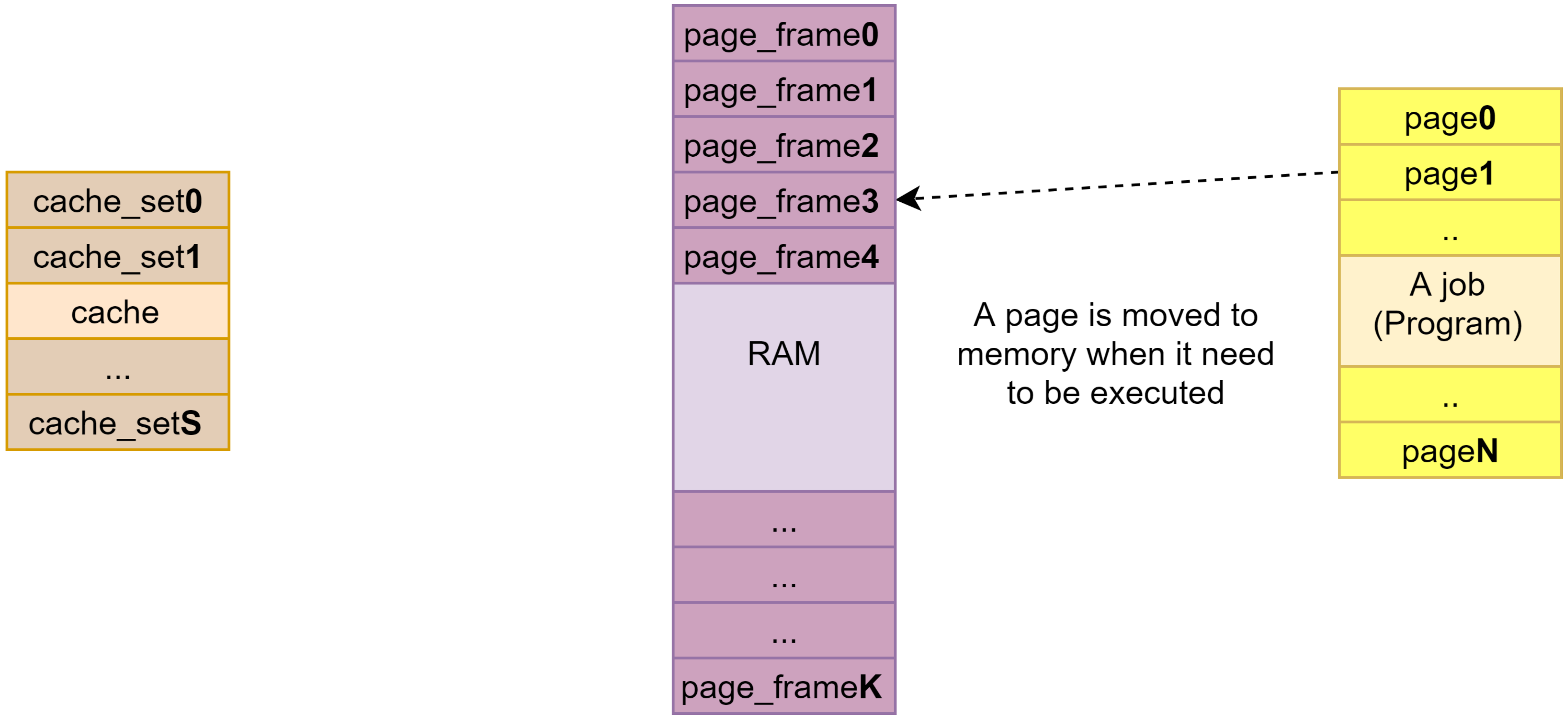


## MEMORY HIERARCHY

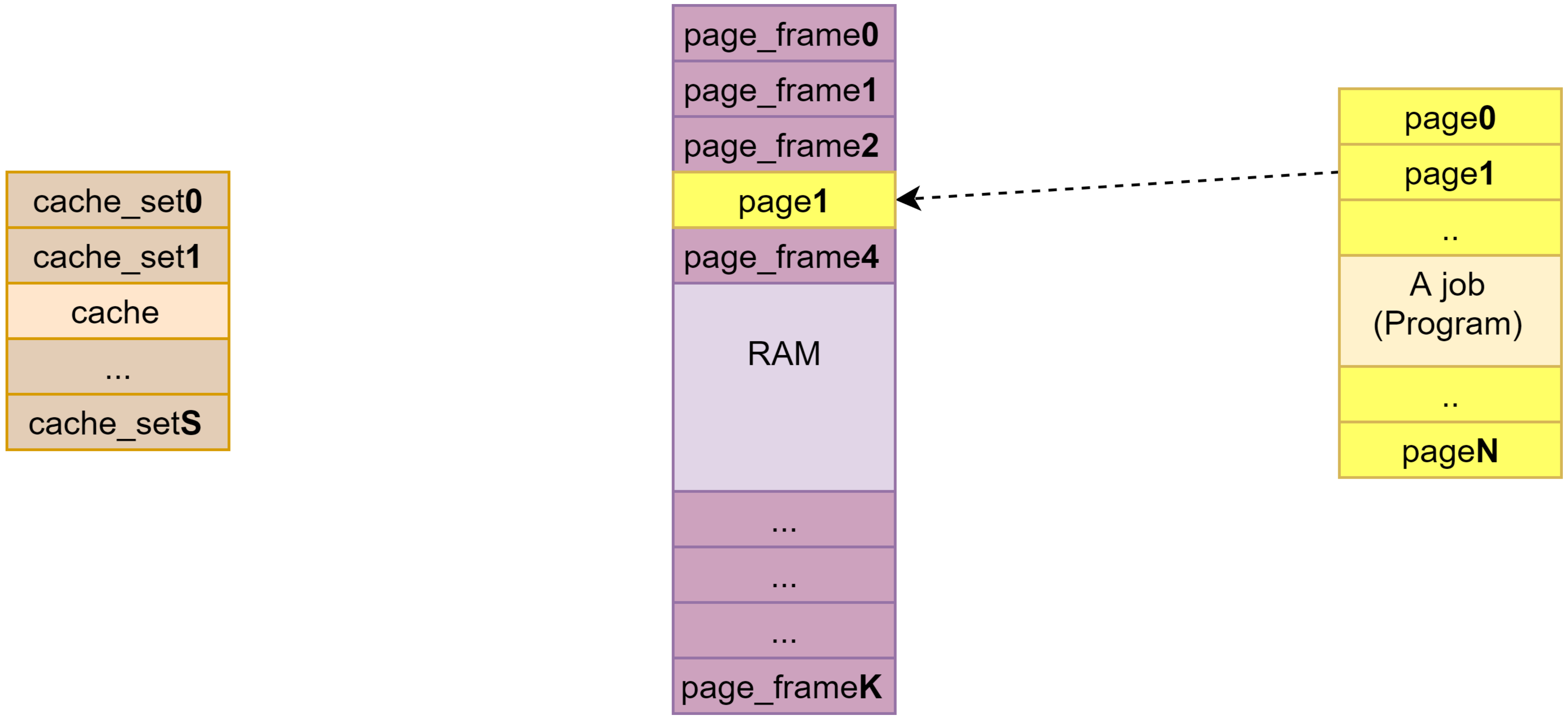


When we want to execute a job, we start by dividing it into pages. Each page is assigned a numirical identifier

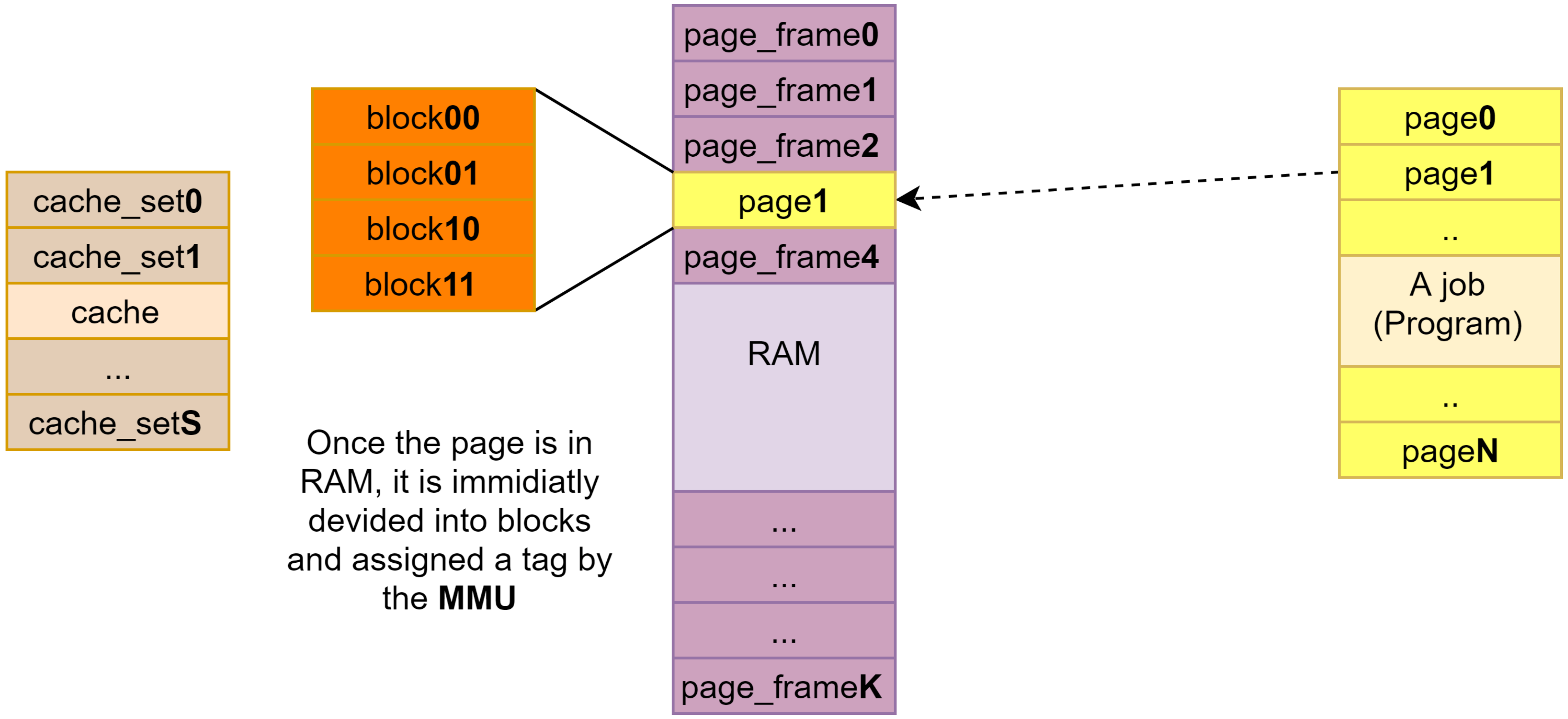
## MEMORY HIERARCHY



## MEMORY HIERARCHY

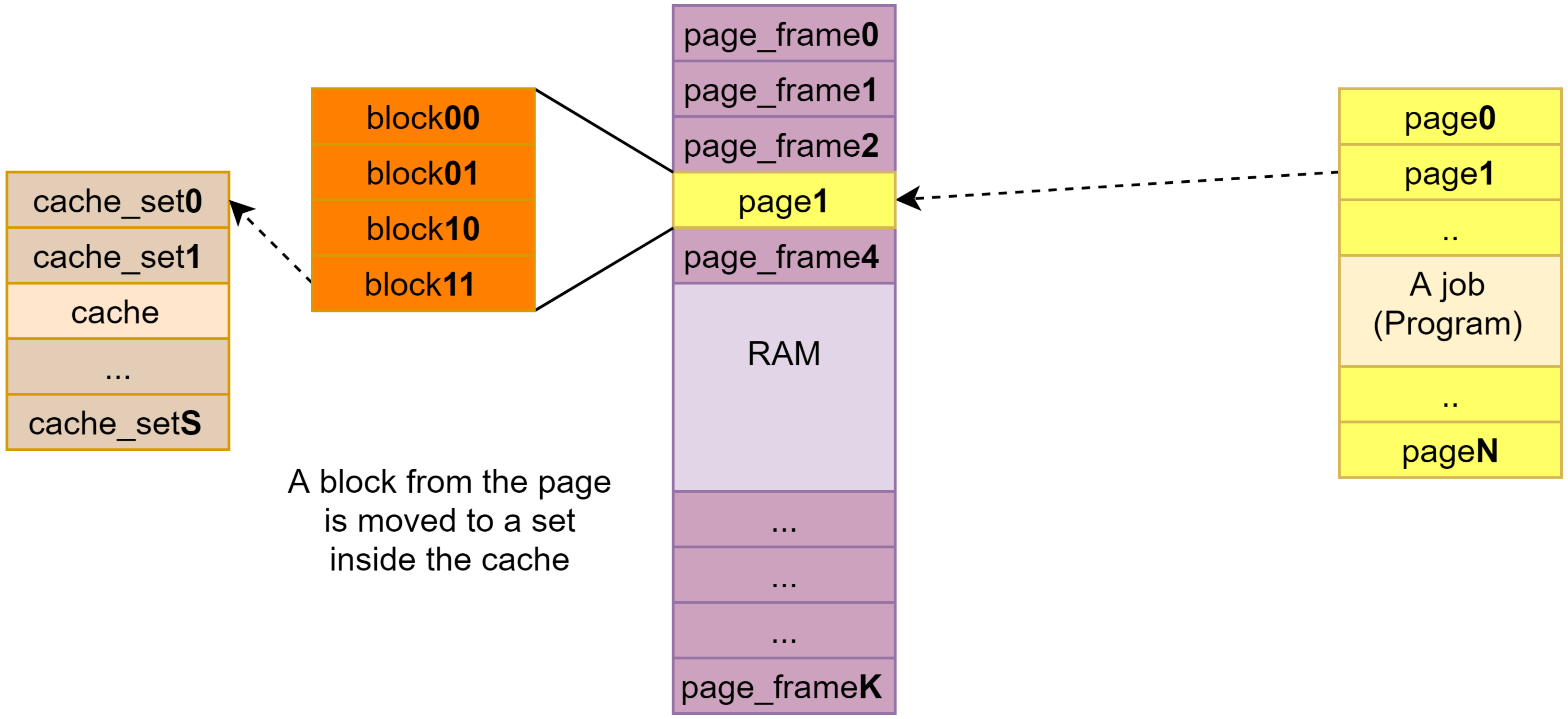


## MEMORY HIERARCHY



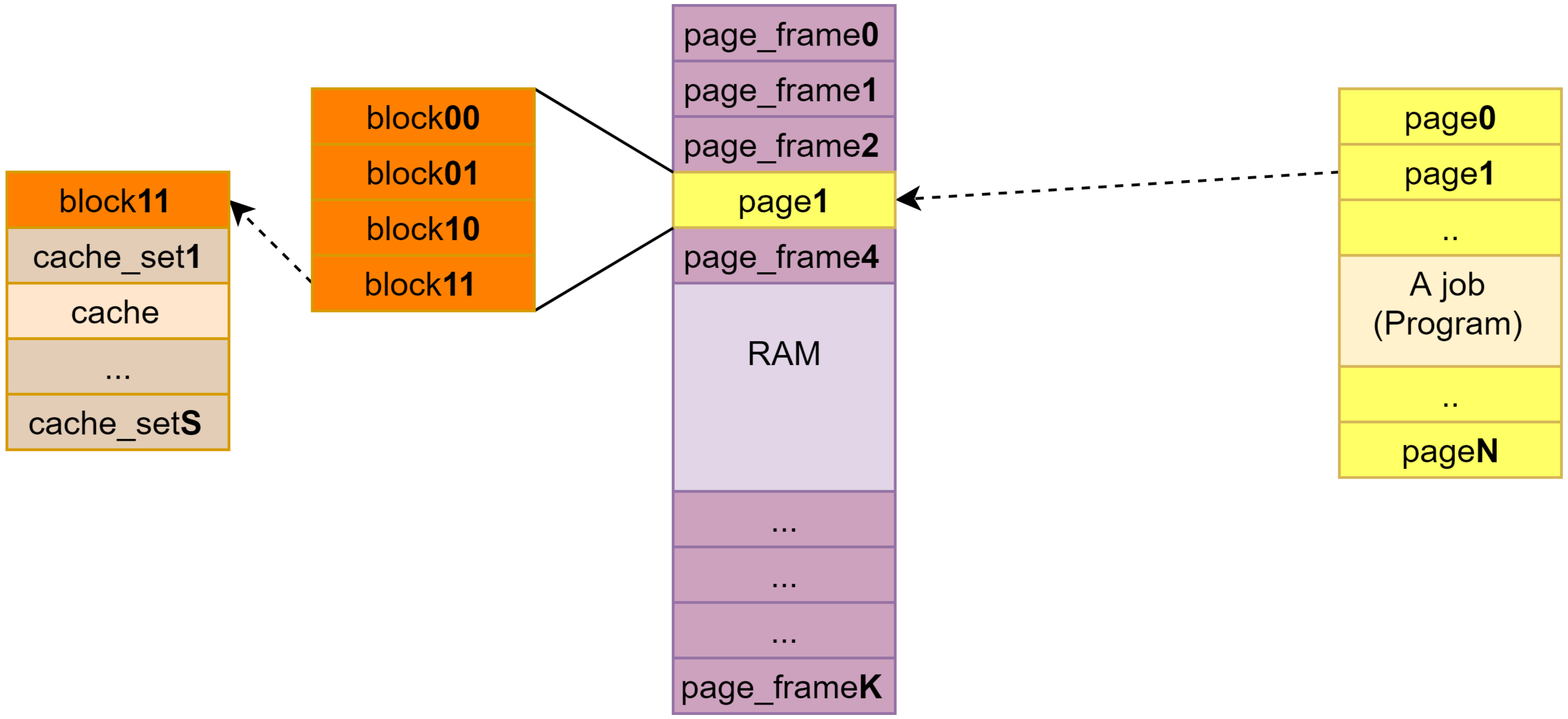
Once the page is in RAM, it is immidiatly devided into blocks and assigned a tag by the **MMU**

## MEMORY HIERARCHY



A block from the page is moved to a set inside the cache

## MEMORY HIERARCHY



## MEMORY HIERARCHY

- **Replacement policies?**
  - Applies both to memory pages and cache blocks/lines
  - **Time**
    - First In First Out (FIFO)
    - Least Recently Used
    - Least Often Used
    - …
  - **Space**
    - Direct Mapping
      - Every block (of memory) has only one possible location (in the cache)
    - Fully Associative Mapping
      - Any block can be mapped onto any location
    - K-way Associative Mapping
      - Combination of Direct and Fully Associative Mapping
    - …

## MEMORY HIERARCHY

- **But, how do we know which page to load to RAM and which block to load into cache?**
  - This is all done by the MMU.
    - The MMU is designed to understand the whole memory system.
    - It knows the number of blocks that are addressable (the tag bits), and organizes the blocks in a way that blocks in memory do not exceed the number addressable blocks
  - Alright, let us have an activity. **Answer the following questions**:
    - Assuming the following:
      - A page is divided into 4 blocks. A word is 4 bytes. We have 4-bit offset. The cache can fit two pages at a time.
        - What is the size of the block? What is the size of the cache?
        - Which block, in which set, and which word this address is referencing: **0x01b6**?
        - What is the maximum number of blocks that this system can address?
        - What happens when this block is required by the system?