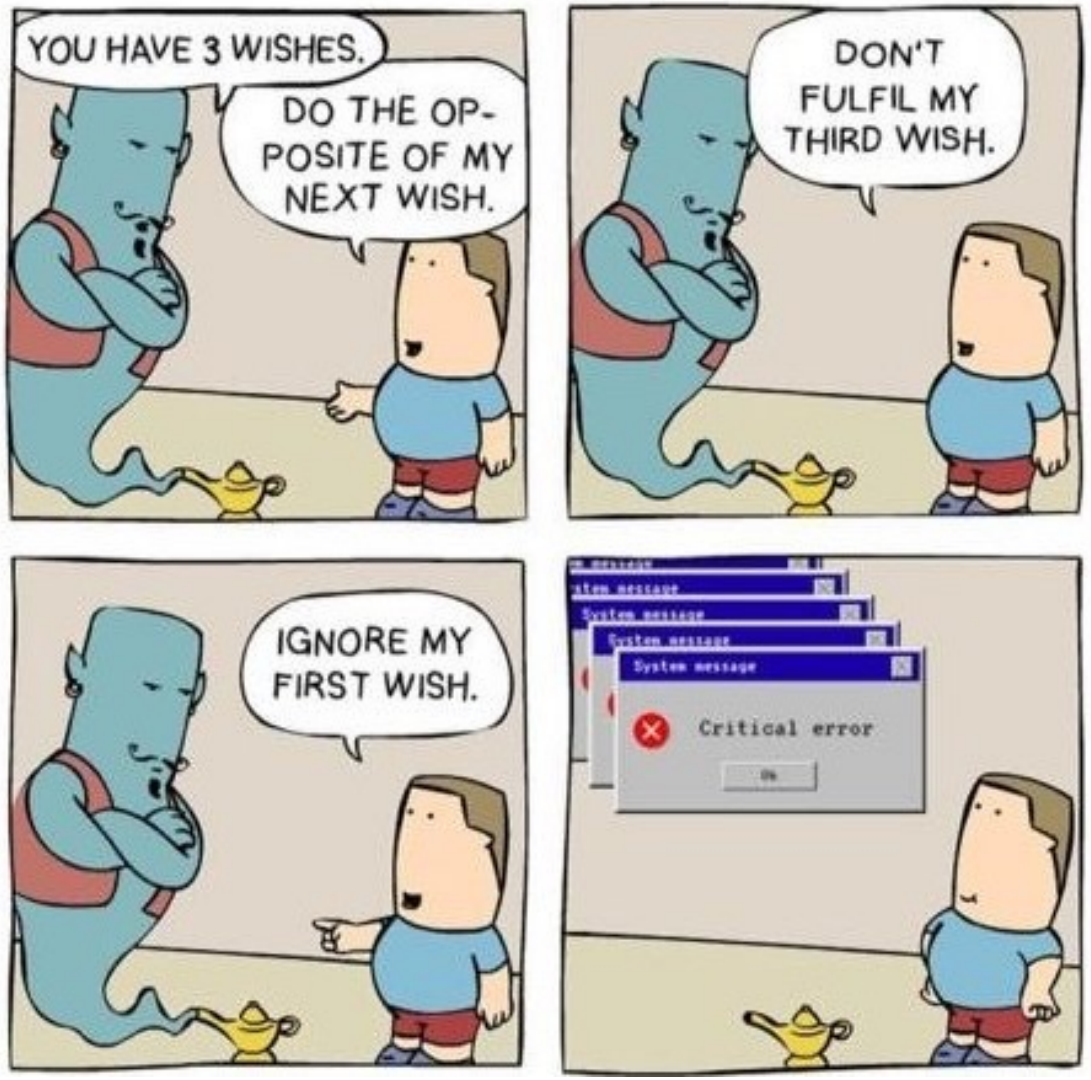
Introduction to Operating Systems and SQL for Data Science

Lecture 4 – Memory

Previous lecture

- Threads
- Races
- Mutex & Semaphore
- Producer-Consumer, Reader-Writer
- Dining philosophers
- Deadlock
- Banking algorithm

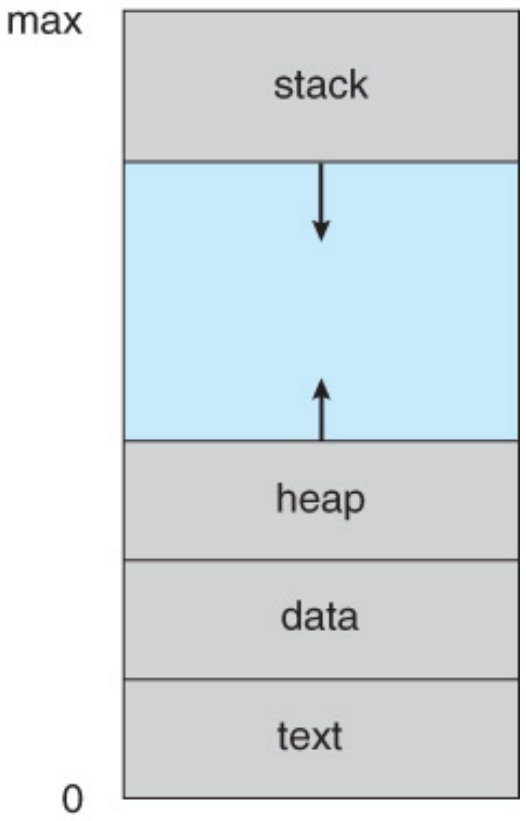
Deadlock – cyclic waiting



Reichman University

Memory - recap

Every process has address space.



Desirable properties of memory:

1. Big (or better infinity)
2. Fast (or better immediate)
3. Private – every process has its own memory
4. Non-volatile – saves information even when there is no power.
5. Cheap

Reichman University

Memory – hierarchy

1. Registers – fast(2ns), very expensive, volatile.

(We will not talk about cache in this part of the course)

2. RAM (Random Access Memory) – Fast (10ns), medium price, medium size, volatile.

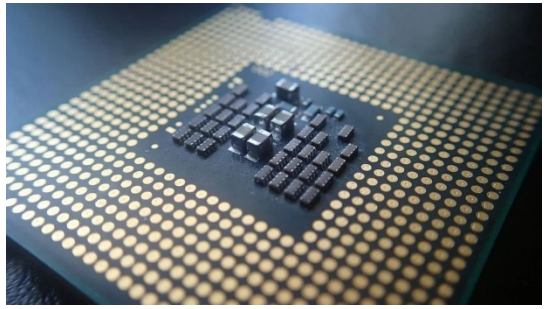
(We will not talk about SSD or DOK)

3. Disk – slow(5ms), big, non-volatile, cheap.

4. Tape drive – very slow, (close to) infinity. Usually made for backups.

Reichman University

Memory – hierarchy



Registers

RAM

Disk

(IBM) Tape drive

Reichman University

Memory management

- Creates an abstraction of hierarchy memory (transparent to the user/app developer)

- Managing memory resources, for example:
  - Which parts are used and by whom.
  - Memory allocation.
  - Collecting memory after use.

Memory without abstraction

- In the past, processes just easily access every cell directly in the memory.

(Meaning the memory was implemented as an array)

- Problem 1 – A process could sabotage memory of other processes or even the OS memory.

- Problem 2 – it was very hard to implement parallelism of processes since every process could access an address.

Memory abstraction

- The address space is the subset of addresses that a process is allowed to access.
- Every process has its own private address space. Hence, in practice address x of a process is in a different physical space.
- We shall grant every process a block of memory which will store the address space.

Reichman University

Address space implementation

Simple implementation

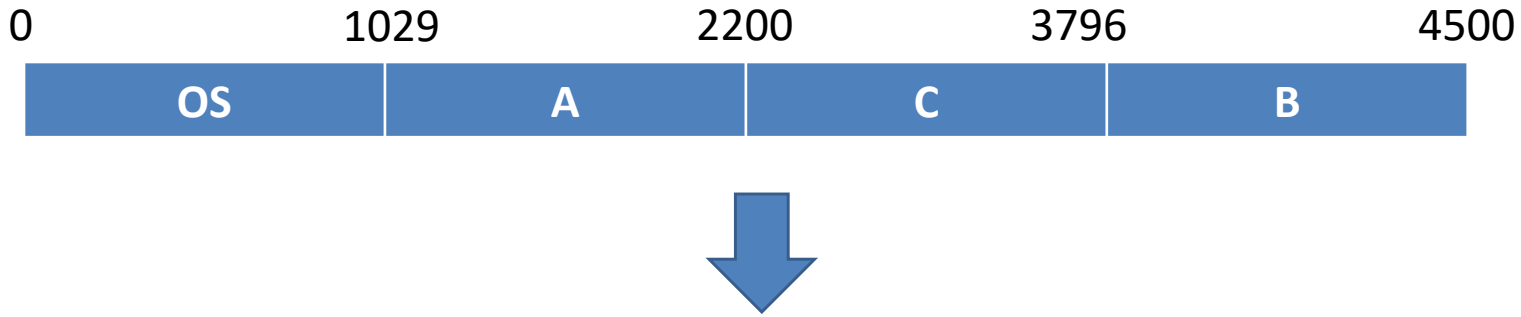
The implementation contain 2 registers – base and limit. So, We divide the memory into continues regions.

Base – the start of the running process, each process has its own base.

Limit – the memory size, the value of limit allow us to provide the private attribute.

Reichman University

Simple implementation - example



	Base	Limit
OS	0	1029
A	1029	1171
B	3796	704
C	2200	1596

Reichman University

Simple implementation

When a process is accessing address x, we compute on the CPU, the physical address in the following way:

address = base +x

In the context switch process, we need to change the base register value to the base of the incoming process.

Simple implementation

If x>= limit or x<0, we have a segmentation fault – meaning a process is trying to access another process memory.

Unfortunately, in this implementation the RAM is too small for running all the processes together.

Solution

1. Part of the processes will be held in the disk – some of the processes are very slow, so it won't affect the parallelism.

2. Swapping – We swap between processes that are on the disk and on the RAM. (When a process runs on the CPU its address space has to be in the RAM)

Swapping

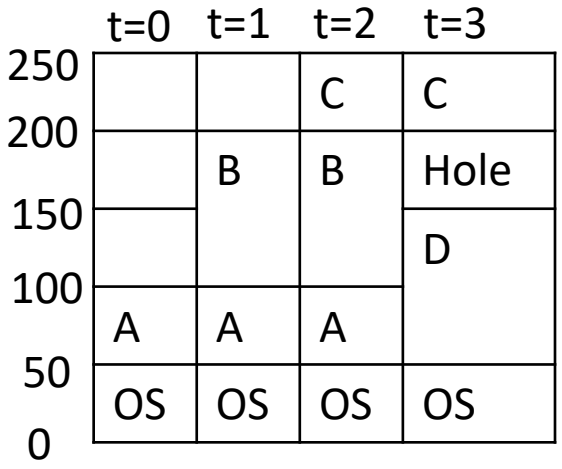
Swapping - example

Swap in – entering to RAM

Swap out – exit from RAM

Problem – when we need to allocate space (if a new process creates or a process that is on disk is schedule to run). We need to decide which process will go to the disk.

Swapping - example



"Hole" – an empty space between processes.

Fill holes with copying – very expansive.

Problems with holes – there might be enough space but not continuous one, which resolved in redundant swapping.

Reichman University

Which process we move to/out the disk?

For that we need algorithm/policy that helps us.

Move processes to disk:

a) Optimal – the process which was blocked the most, but we only know that in the end.
b) LRU (Least Recently Use) – the process on RAM that wasn't running the most.

Where do we fit a new process

There are many holes that we can fit a new process, which one should we choose?

a) First fit – Because we are looking at all the holes that will suffice the process size.

It could be that the first hole is big one and we won't best fit processes if a large process will come.

b) Best fit – the smallest hole that will do.

Reichman University

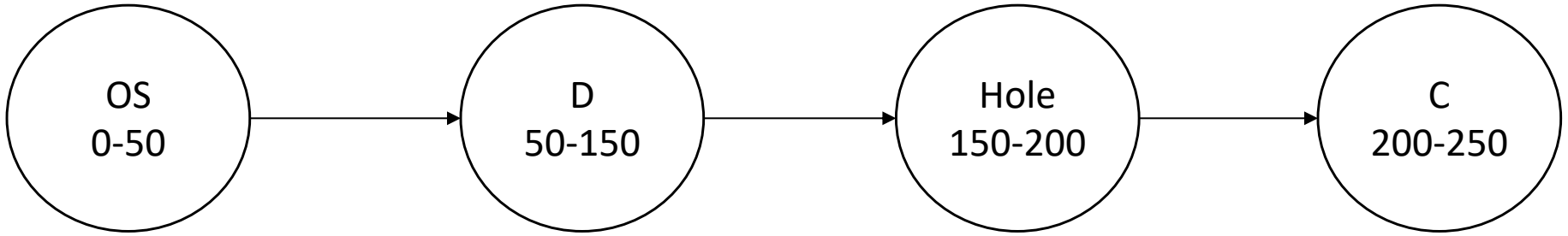
Data Structure for first fit

An array of memory map which every cell contains Boolean value (catch or not), disadvantage – very wasteful, when we want to find a hole, we need to move iteratively.

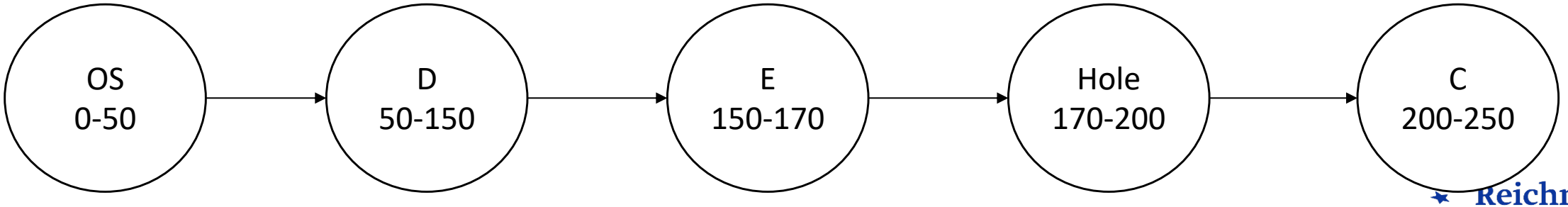
A list of process and holes – we need to find a first hole in the list.

Data Structure for first fit

A list of process and holes (example):

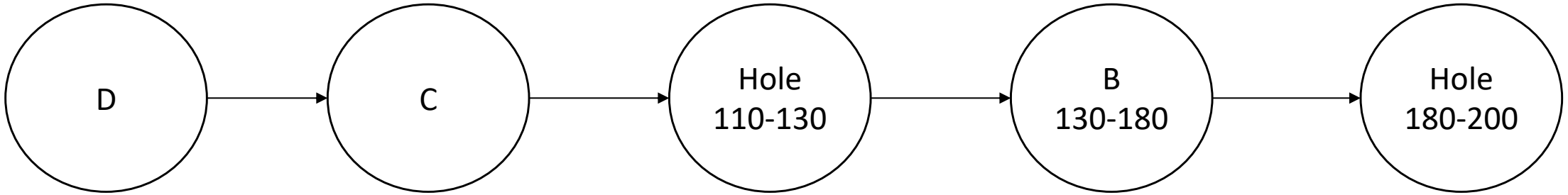


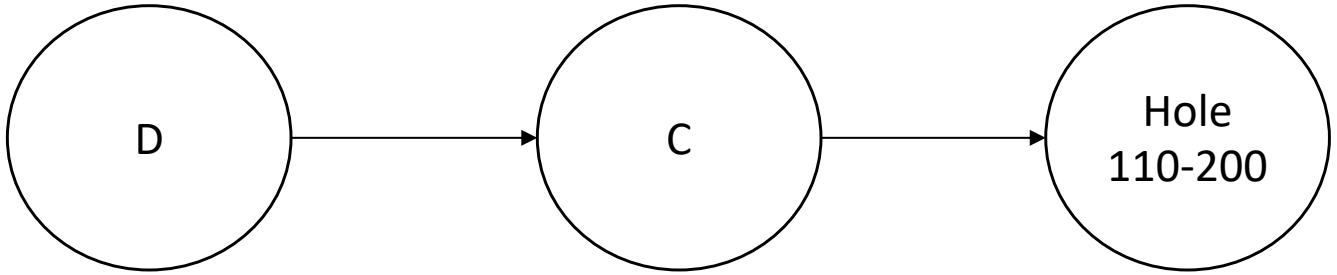
If we would like to add process E=20:



Data Structure for first fit

If we would like to remove process B from RAM:





Reichman University

When we remove a process (first fit)

After removing a process, we create a hole instead of the process.

If there is a hole “neighbor” we merge them together to a bigger hole. Hence, we will have a continuous holes.

Reichman University

Data structure to best-fit

We can hold a linked list of holes by size.

We can use binary-search for finding best fit.

This data structure is hard to update – it is hard to find neighborhoods(before and after if we need to merge)

We can hold another linked-list that we can find neighbors (like in first-fit)

Reichman University

Best-fit problem

Because best-fit is choosing the best hole, every time we choose a hole that is not the same, there will be small holes which we can't insert more processes. The accumulated holes will resolve in a wasteful space.

Reichman University

Additional policies

- Worst fit – we choose the biggest hole. Problem when there is a large process, we need to make many swaps.

- Quick fit – we divide process into group: small (less than 10KB), medium (10KB-10MB), big (more than 10MB) and in each group we can use best-fit.

Reichman University

Increasing memory for a running process

Sometimes, a process needs more memory in running time. This is a hard problem (few solutions):

- Swap-out for the running process and swap-in with the new size.
- Swap-out for the next process.
- We shall grant more memory with swap-in.

Not good
Those are processes that want to run

Reichman University

Virtual Memory

Virtual memory

We saw that a swap of the entire process could be expensive, and sometimes we want to add more memory to the process.

Locality of reference – Process won’t access randomly to memory, if an access to address x was made, the access to address that is close to x is more likely to happen.

Reichman University

Virtual memory

Instead of loading all the process memory we divide processes memory to pages with uniform size.

Each process has its own virtual memory.

Not all the pages must to be in the physical memory when the process is running.

More about Physical address and Virtual address: link

Reichman University

Virtual memory

Page (Disk) is equivalent to frame(RAM).

We divide the RAM to chunks that equal to the page size. Thus, the size of holes will always be the same of the pages (mitigate the holes problem).

Usually a thread that runs will contain the code, data and stack (Page 0 will contain the code).

Virtual memory

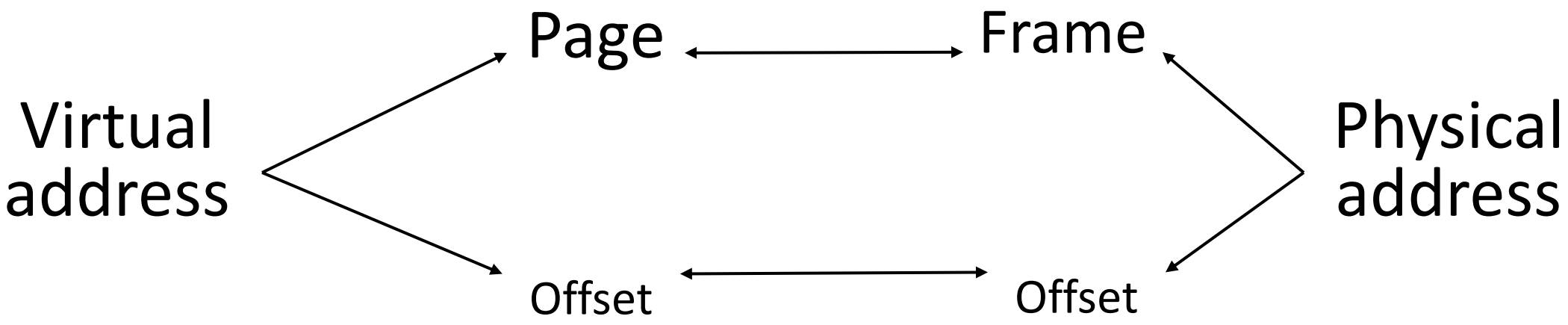
Virtual address – the address in the virtual memory.

Physical address – the address in the RAM

The processes are accessing only the virtual memory which the OS transforms it to the physical addresses.

Reichman University

Virtual vs physical addresses



Reichman University

Virtual vs physical addresses - example

Assume bus size (data and address) of 32 bit and RAM memory of 16MB size.

Page size of 1024 words. How many frames we have in the system?

Virtual vs physical addresses - example

First let's calculate word size – 32/8 = 4Byte (8 bits inside Byte)

Page size:

1024(words) * 4Byte = 4096Byte = 4kB

Page size is equal to the frame size.

Reichman University

Virtual vs physical addresses - example

In order to calculate the frame size we divide the RAM size with frame size:

$$\frac{RAM\ size}{page\ size} = \frac{16MB}{4KB} = \frac{16 * 2^{20}}{4 * 2^{10}} = \frac{2^{24}}{2^{12}} = 2^{12} = 4096\ frames$$

So, we have 4096 frames in our RAM.

Reichman University

Page table

A mechanism to map virtual addresses to physical address.

There could be a table for each process, or one table for all the processes.

For each page we save in the table:

- If the page is currently in the RAM.
- Number of page frames.
- Modified bit – if there any change in the page (to made the change in the disk)
- Referenced bit – if there were an access to the page

Reichman University

When we swap pages?

When we need to insert a page into the RAM.

The process is refraining to a page which is not on the RAM.

We will upload a page to an available frame

Reichman University

Page swapping algorithms

Which frame we move from the RAM.

1. Optimal – we want to avoid PF(page fault) as long as we can. We'll take out the page that has not accessed for the longest time. (Only available in the end of the ruining).
2. FIFO – Not that good we want to remain frames that are accessing a lot.

There are more page swapping algorithms that we not cover in the scope of the course.

Paging vs Swapping: link