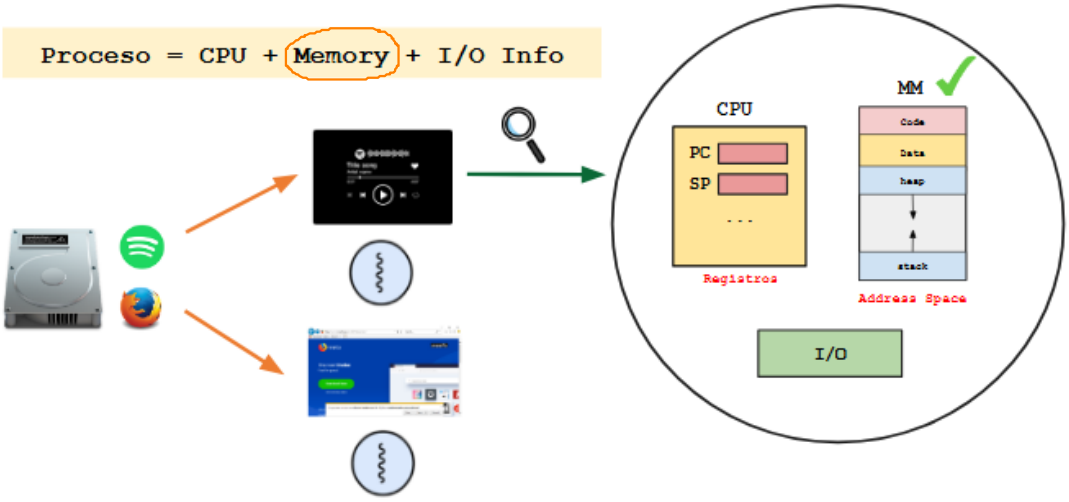
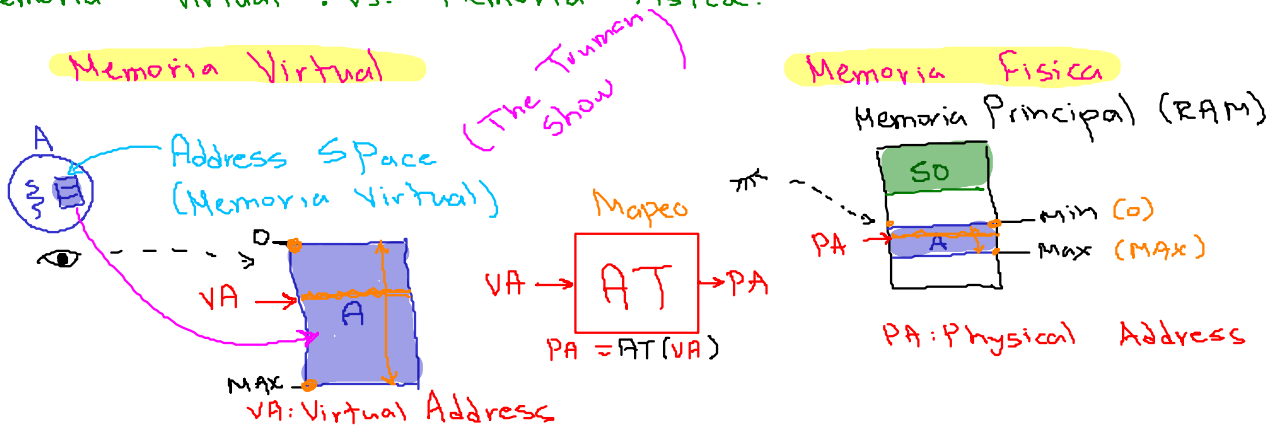
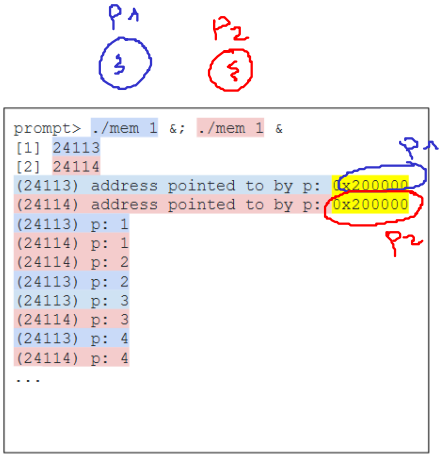
09/09/2025 - Sistemas Operativos (Ude@)

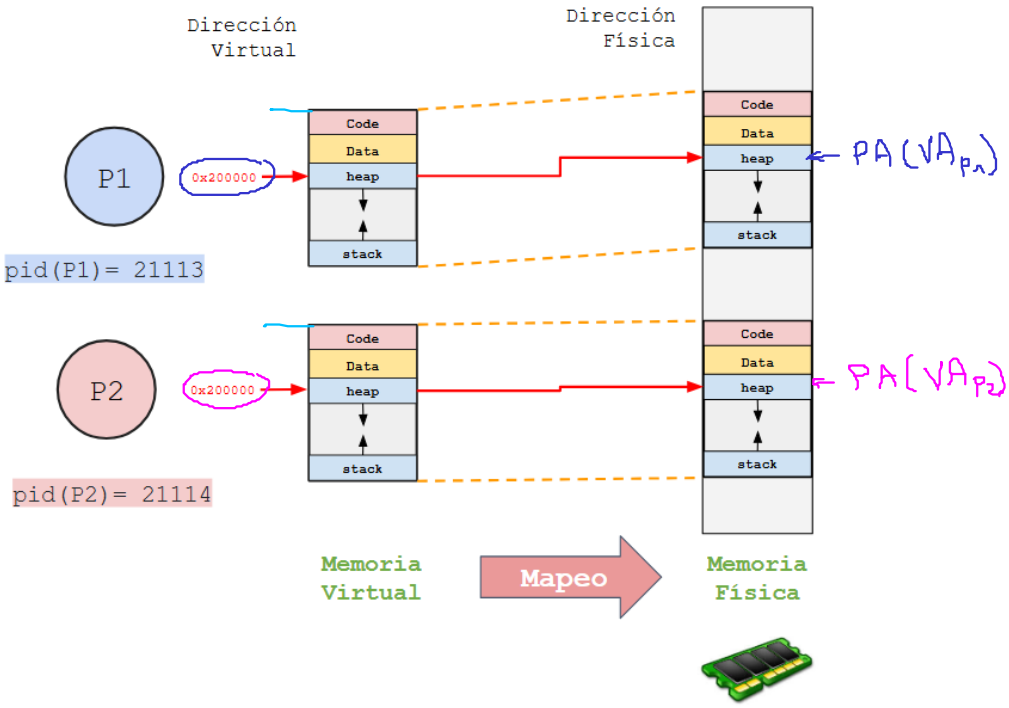
1. Proceso:



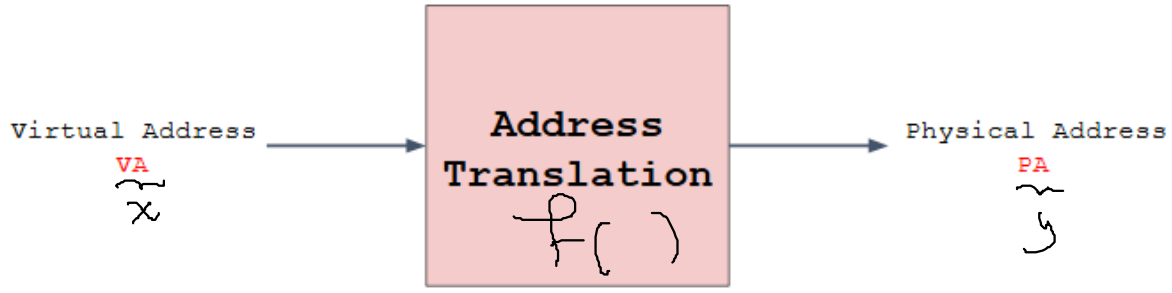
2. Memoria Virtual .vs. Memoria Física.







3. Address Translation (AT)

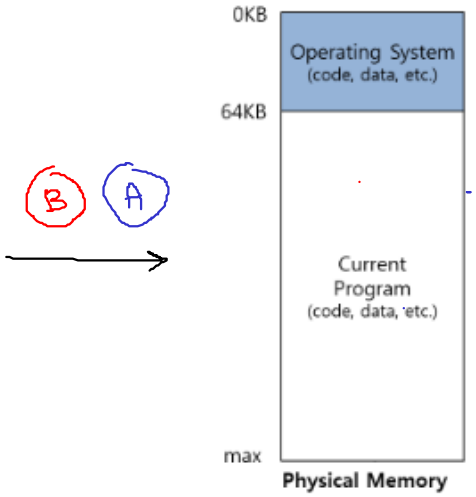


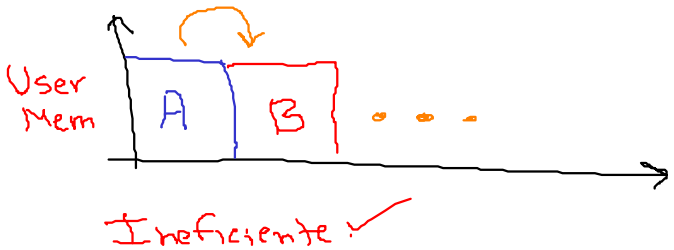
$y = f(x)$ ⇝ PA = Address Translation (VA)

4. Evolucion de los Sistemas Operativos en cuanto al manejo de memoria

* Primeros Sistemas operativos:

- Se cargaba un solo proceso





Ineficiente ✓

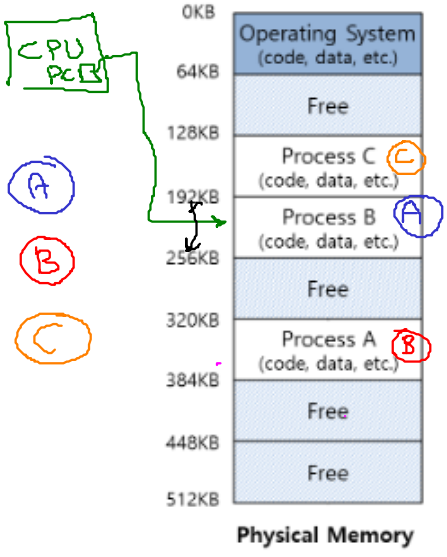
* Multiprogramacion:

- Se cargan varios procesos en memoria

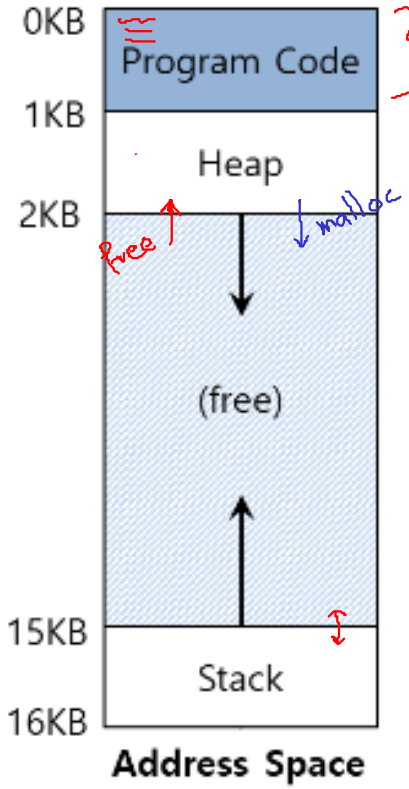
↑ Utilización ✓

↑ Eficiencia ✓

* Proteccion



5. Memory Map



Code + Data (text)

Program Code	• Instrucciones • Variables globales
Heap	• Memoria **asignada dinámicamente** ◦ `malloc` en C ◦ `new` en Java o C++
Stack	• Memoria **automática** • Direcciones y valores de retorno • Variables locales y argumentos pasados a rutinas

(usuario)

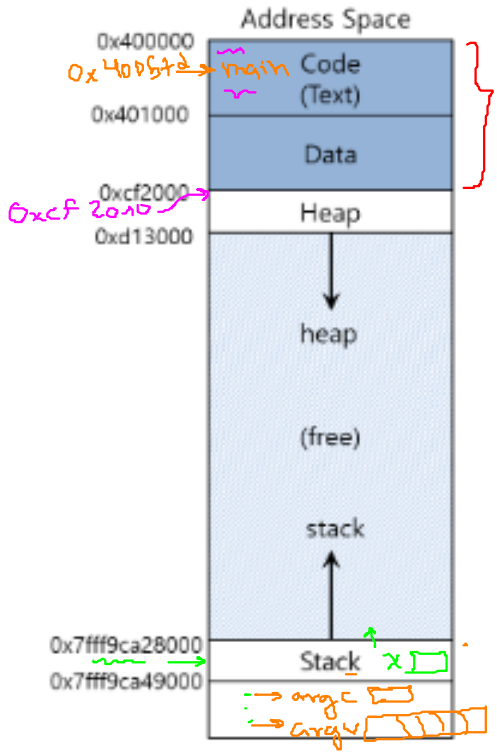
(Automatica)

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  printf("location of code  : %p\n", (void *) main);
  printf("location of heap  : %p\n", (void *) malloc(1));
  int x = 3; // local
  printf("location of stack : %p\n", (void *) &x);
  return x;
}
```

```
location of code  : 0x40057d
location of heap  : 0xcf2010
location of stack : 0x7fff9ca45fcc
```

The output in 64-bit Linux machine

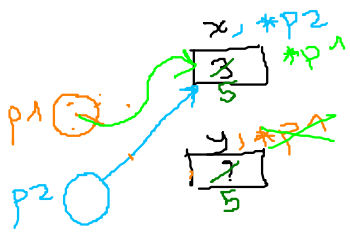


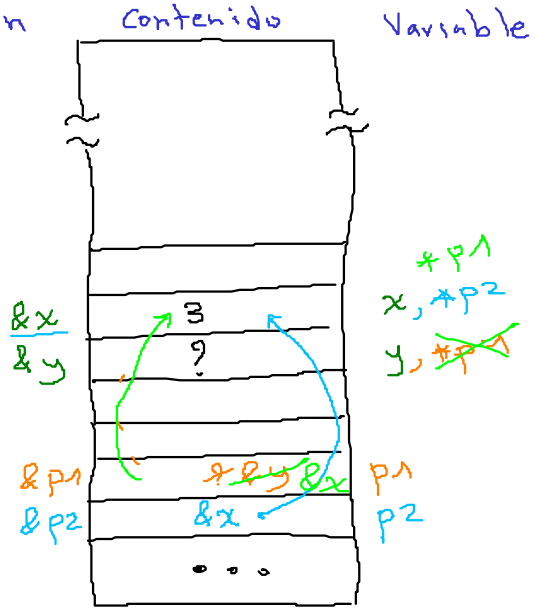
6. API de memoria

- ✓ `int x = 3, y;`
- ✓ `int *p1;`
- ✓ `int *p2 = &x;`
- ✓ `p1 = &y;`
- ✓ `*p1 = *p2 + 2; // y = x + 2`

`p1 = p2; // p1 = &x;`

`*p1 = *p2 + 2; // x = x + 2`





Problemas:

stack . vs . heap ← Usuario

sizeof ( )

- malloc ( )
- free ( )
- realloc ( )
- mmap ( ).

Problemas: