¿Qué es un programa ?

A groso modo podríamos decir, que un programa consiste en **datos (memoria) e instrucciones** que son ejecutadas por un procesador.

Los lenguajes de programación nos permiten **pensar en el problema a solucionar** y olvidar las instrucciones a nível de lenguaje maquina.

```
0x1000013d0:  pushq  %rbp
0x1000013d1:  movq   %rsp, %rbp
0x1000013d4:  subq   $0x20, %rsp
0x1000013d8:  movq   0xc31(%rip), %rax
0x1000013df:  leaq   0xabe(%rip), %rcx
0x1000013e6:  movl   $0x0, -0x4(%rbp)
0x1000013ed:  movl   %edi, -0x8(%rbp)
0x1000013f0:  movq   %rsi, -0x10(%rbp)
0x1000013f4:  movq   %rax, %rdi
0x1000013f7:  movq   %rcx, %rsi
0x1000013fa:  callq  0x100001d46
0x1000013ff:  movl   $0x0, %edx
0x100001404:  movq   %rax, -0x18(%rbp)
0x100001408:  movl   %edx, %eax
0x10000140a:  addq   $0x20, %rsp
0x10000140e:  popq   %rbp
0x10000140f:  ret
```

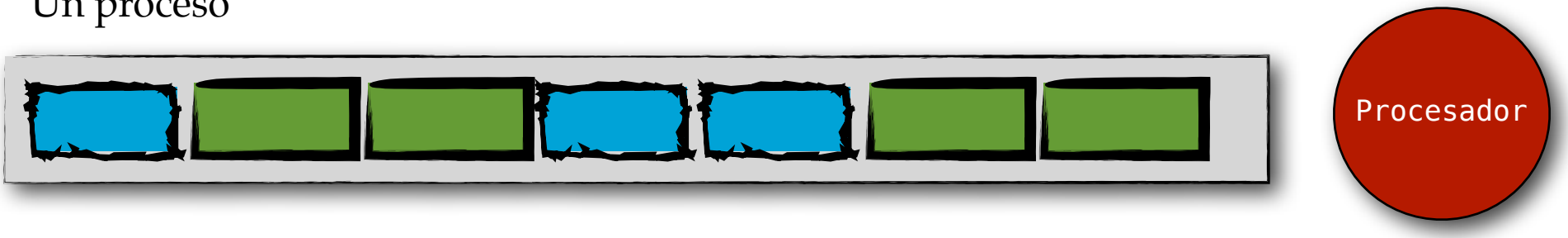
Ejemplo de hola mundo en c++

```cpp
#include <iostream>

int main(int argc, const char * argv[])
{
    std::cout << "Hello, World!\n";
    return 0;
}
```

Código ensamblador generado por el compilador, el hola mundo en c++.

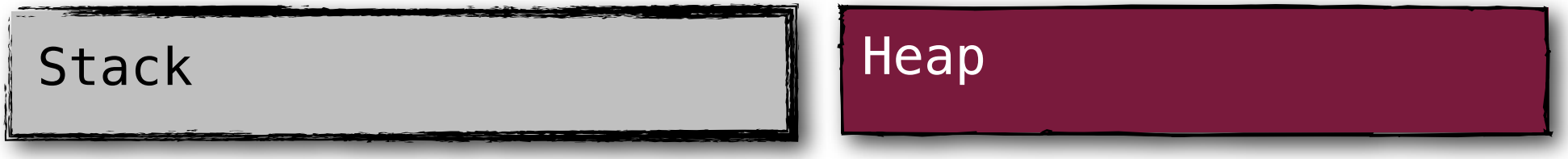
```
; (void *)0x00007fff730a4f48: std::__1::cout
; "Hello, World!\n"



; symbol stub for: std::__1::basic_ostream<char
```

Un proceso



Un proceso se define como una unidad de ejecución más básica , cuando un programa está en ejecución por el procesador se le define como proceso.

La memoria asociada a un proceso es ordenada en dos estructuras de datos esenciales, el heap (montonera) y el stack (pila). Aunque en el lenguaje de alto nível no es necesario preocuparse por la memoria directamente, si es necesario en evitar crear multiples copias, de los datos que estamos utilizando, ya que no queremos resolver un problema devorando los recursos de nuestro hardware de forma ineficiente.




```
{
    int x = 3;
    int myArray[10];

 }
```

```
{
      int *x = new int;
      int *myArray = new int[10];

      delete x;
      delete[] myArray;
 }
```

Producto Punto

Se desea multiplicar dos vectores u*v .

```
#include <iostream>

int dot(int *x, int *y, int size){
    int product=0;
    for (int i=0; i < size; i++){
        product += (*(x+i) * *(y+i));

    }
    return product;
}

int main()
{
    /* Ambito Principal */
    int u[6] = {1, 9, 3, 1, 5,  6};
    int v[6] = {4, 2, 8, 4, 7, -1};
    /* Inicia el ambito de la funcion */
    auto result = dot(u, v, 6);

    std::cout<<"result:"<<result<<"\n";
    return 0;
}
```

Preguntas

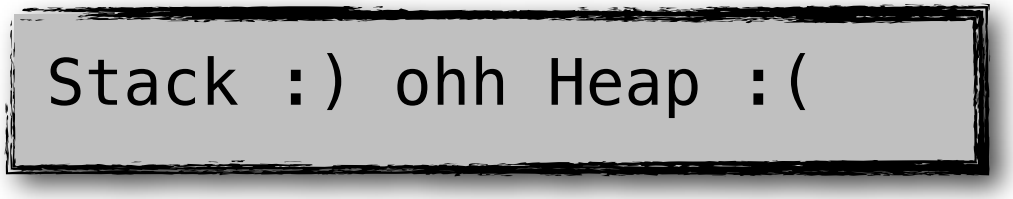
¿Cuántas copias se hacen de los vectores u, v en todo el programa?

¿Cuáles variables están siendo asignadas al Stack?

¿Cuáles variables son asignadas en la montonera?

```
int main()
{
    /* Ambito Principal */
    int *u = new int[6];
    int *v = new int[6];

    for (int i =0;i<6; i++){
        u[i] = i+7;
        v[i] = i+4;
    }

    /* Inicia el ambito de la funcion */
    auto result = dot(u, v, 6);
    delete [] u;
    delete [] v;

    std::cout<<"result:"<<result<<"\n";
    return 0;
}
```

La pregunta se reduce a para que necesitamos saber donde se están asignando nuestros datos.



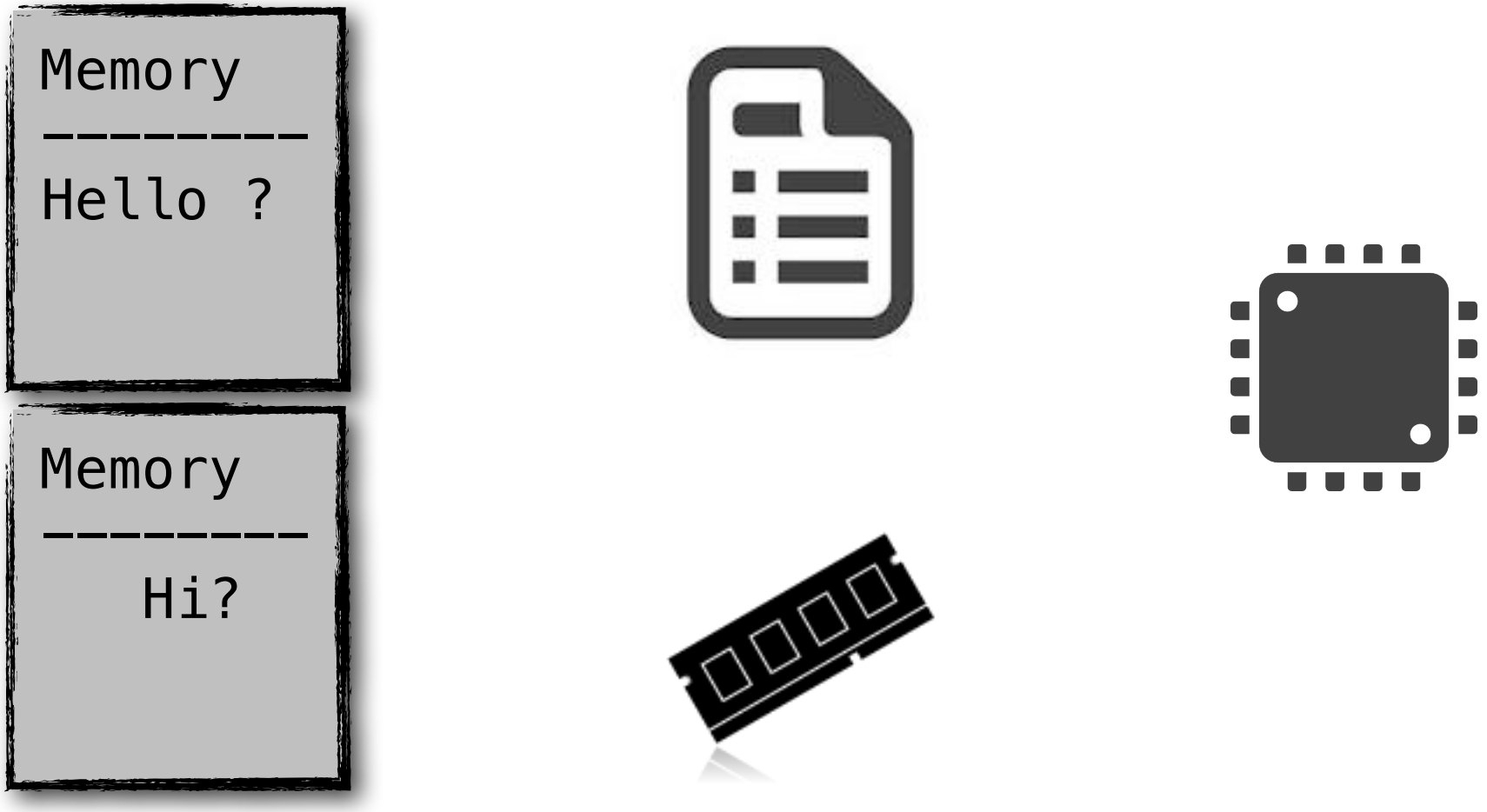
I. Un stack es asignado por proceso.
II. El tiempo de vida de una variable termina cuando el ambito en el que fue asignada termina

```
int y = 30; // Asignacion de memoria para y
int *address = 0;

{
    int z = 10; // Asignación de memoria z
    address = &z;
    z+=y;
    std::cout<<z<<"\n";
}// Memoria liberada z

int z = 50; // Nueva asignacion para z
std::cout<<&z<<"->"<<z<<"\n";
std::cout<<address<<"->"<<*address<<"\n";
```

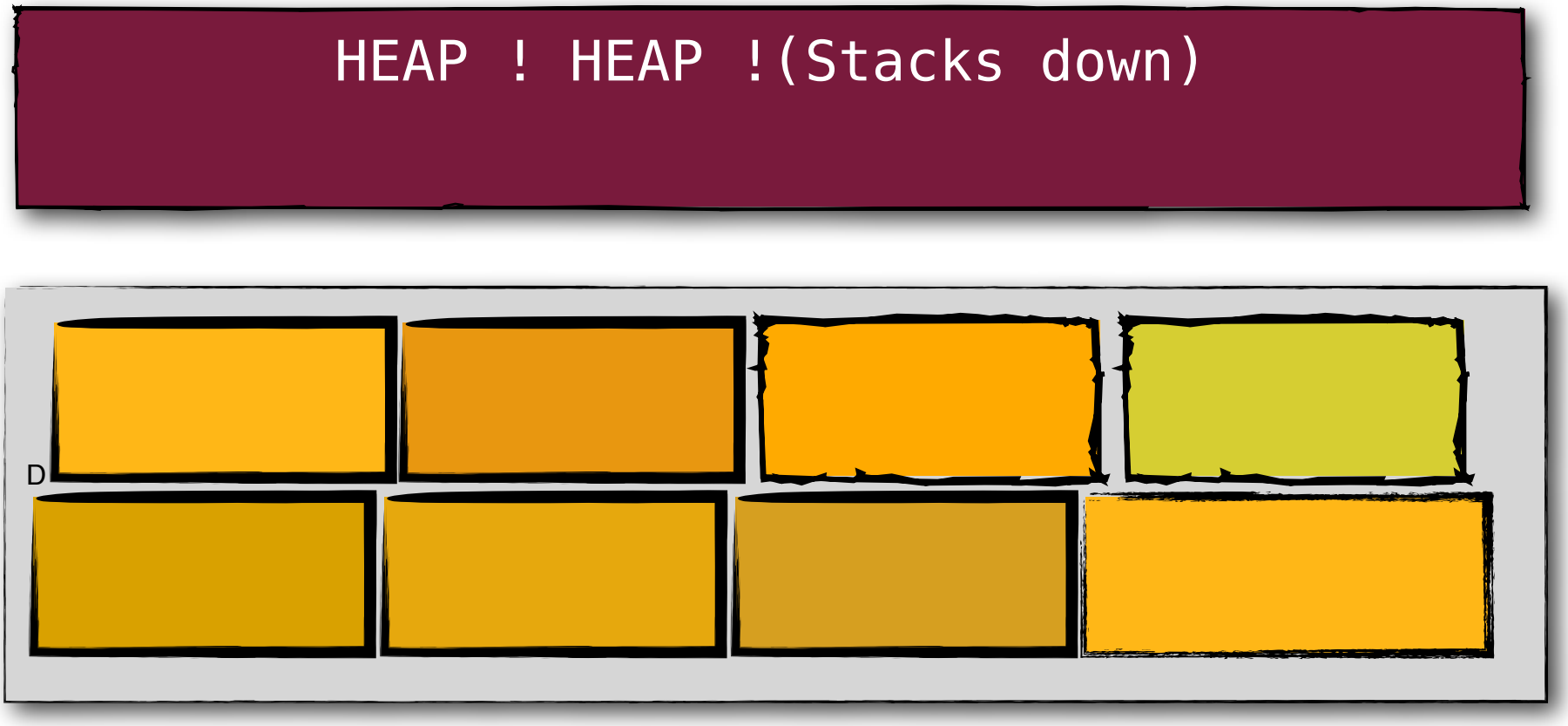
```
40
0x7fff5fbff898->50
0x7fff5fbff89c->40
```

Hilos y procesos.

Como vimos anteriormente los procesos tienen su propia memoria asignada, agrupada en el heap y el stack, sin embargo son independientes, si queremos comunicar dos procesos o más, se necesita un recurso compartido al que ambos puedan acceder, pueden ser un archivo en el disco duro, o incluso un bloque de memoria independiente de ambos administrado por un sistema operativo.



Sin embargo , si necesitamos hacer una actividad de forma concurrente, un proceso puede tener multiples hilos de ejecución, y podemos compartir información entre ellos, mediante el uso de variables que se encuentren en el heap. Cada hilo tiene su propio stack.



Lanza un grupo de n trabajadores que calculen el factorial de acuerdo a su numero asignado en 0....n , al terminar el calculo deben de escribir en un diccionario el calculo que realizaron. Workers

```cpp
#include <thread>
#include <iostream>
#include <map>
#include <vector>

// Generic function
long long factorial(int n){
    if (n){
        return n*factorial(n-1);
    }
    return 1;
}

void worker(int id, std::map<int, long long> *results){

    auto fact = factorial(id);
    (*results)[id]=fact;

}

int main(){

    std::vector<std::thread> threads;
    auto results = new std::map<int, long long>;

    for(int i = 0; i < 5; ++i){
        threads.push_back(std::thread(worker, i , results));
    }
```

```
    for(auto& thread : threads){
            thread.join();
    }

    for (auto& it: *results){
        std::cout<<" id "<<it.first<<" val :"<<it.second<<"\n";

    }

    delete results;

    return 0;
}
```

En el código anterior, aún cuando tengamos un heap comun, existe un problema inherente al uso de la memoria compartida, llamada condición de carrera donde crees que ocurra.

```
id 0 val :1
id 1 val :1
id 2 val :2
id 3 val :6
id 4 val :24
```

Realiza los siguientes ejercicios

1- Realiza un programa que sea capaz de realizar multiplicación de 2 matrices cuadradas de **forma concurrente**, las matrices pueden ser reales o complejas.

2- Te encuentras en el salon de clases y quieres descubrir quién es tu vecino más cercano, para poder pedirle un lápiz, la lógica nos dice que podríamos calcular la distancia con cada uno de ellos, sin embargo este metodo de fuerza bruta es tedioso, y no queremos desperdiciar nuestro valioso tiempo, como ordenarias la posición de tus compañeros en el salon para hacer una busqueda más eficiente y determinar a tu vecino más cercano?