```
int main(_, _)
{
   Declare And Init Array();
   Print Array();
}
```

→ prints out intended ints

```
void Declare And Init Array()
{
   int array[100];
   int i;
   for (i=0; i<100; i++) {
      array[i]=i;
   }
}
```


```
void Print Array()
{ int array[100];
  int i;
  for (i=0; i<100; i++)
     printf("%d\n", array[i]);
}
```

however laying out memory before use had use cases

my comment:

e.g. cache-oblivious algos

int printf (const char * control , ...);

# of placeholders succesfull bound to → -1 if failure

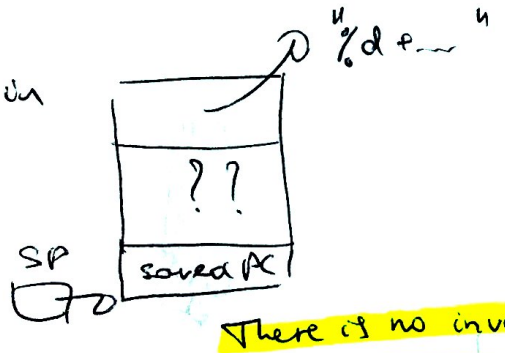
ellipsis: any # of arguments

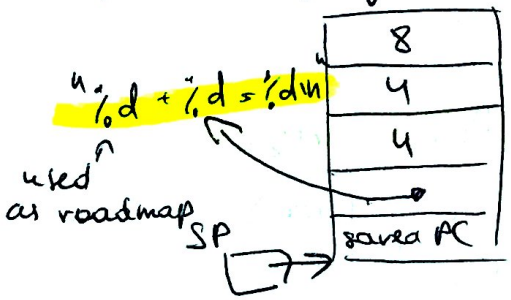
printf ("Hello \n"); → 0

printf ("%d + %d = %d \n", 4, 4, 8); → 3

gcc does type checking but not most compilers

Reason for the reverse order of parameters [redacted] on AR on stack: there maybe ... params, start with [redacted] control string and keep decrementing SP while analyzing control string.





There is no invariant to find roadmap to determine other parameters and how much space to allocate for each.

```
struct box {
    int code;
};
=
struct type_one {
    int code;   <-- 1
};
struct type_two {
    int code;   <-- 2
};
```

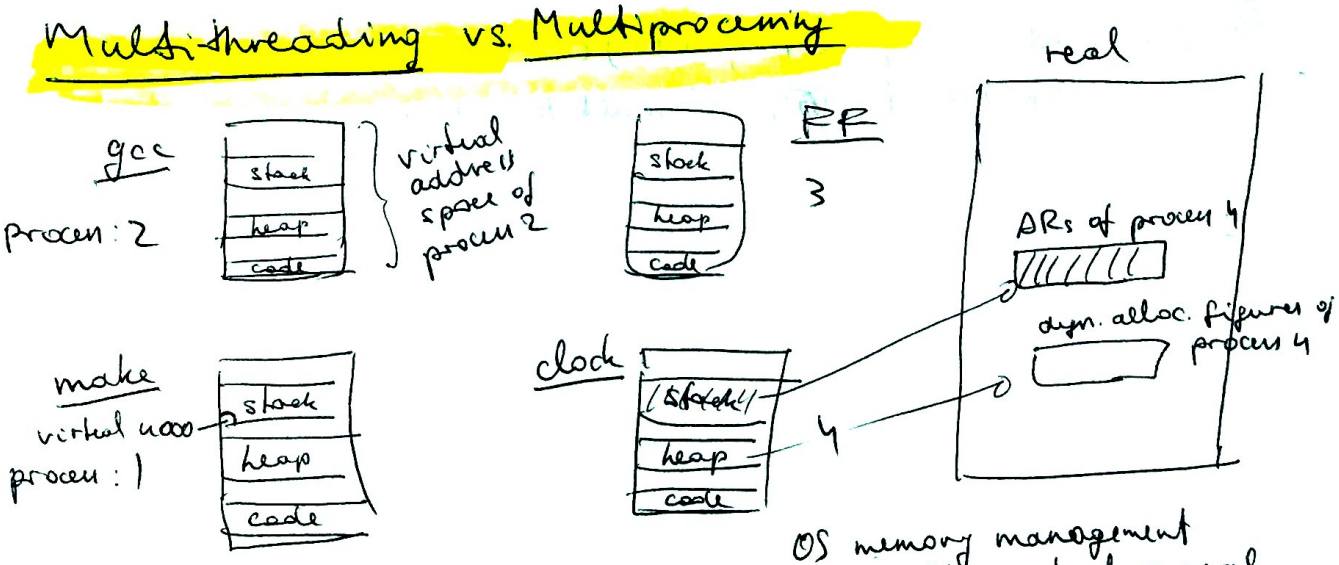
→ 1 or 2

depending of code choose type_one cast or type_two cast

e.g. Ipv4 and Ipv6 structs

if code was the last in struct it would be at a variant distance from [redacted] address to struct

Multi-threading vs. Multi-processing



multi processing

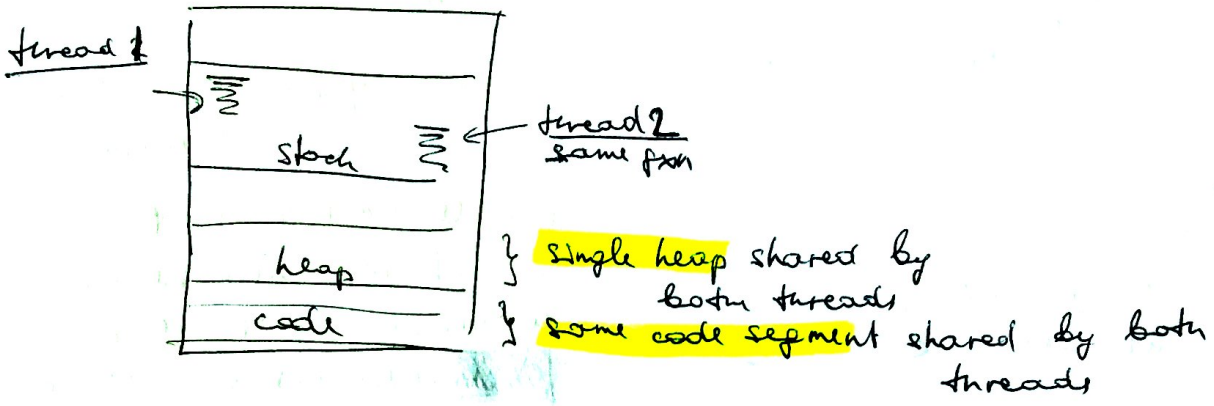
on a single processor, 2 processes : switch between processes quickly so that the run seemingly at the same time

Multithreading :

- functions in the same process to seemingly run at the same time within 1 stack, 1 heap & 1 code segment

my comment

- See parallel algorithms for a formal definition
  → algorithm parallelism




```
int main()
{
  int numAgents = 10;
  for (int i = 0 ; i < 10 ; i++){
     SellTickets (i, 15); ← multithread this function
  }
}
```