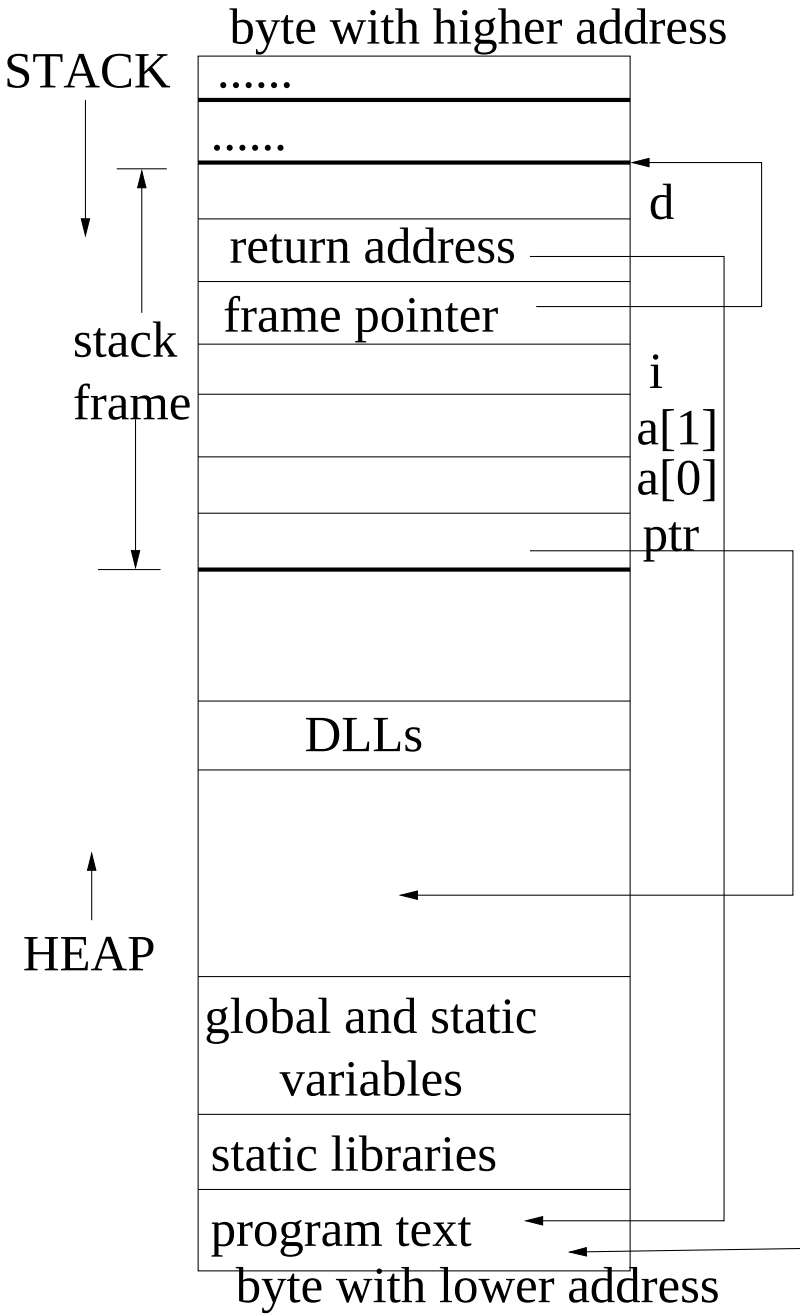
Organization of process memory

R. Inkulu
http://www.iitg.ac.in/rinkulu/



program counter register

```
int func(double d)  {
    int i;
    char a[2];
    float *ptr = (float*) malloc(
          ceil(d/10)*sizeof(float));
    //I AM HERE
    ...
}
```

* every byte is addressable

Process memory segments/regions

- *text/code segment* and the segment storing static libraries are not mutable once the program is loaded into memory; all the other regions are mutable

- *data segment* comprises of
  * *bss segment* comprises of global and static variables that are intialized to zero or do not have explicit initialization in source code
  * explicitly initialized global variables, static variables, mutable constant-lengthed strings, and
  * the heap[1]

- *stack region* comprises of a sequence of *stack frames*, each correspond to a function on the call stack

[1] dynamic linked libraries (DLLs) are loaded as and when needed i.e., while the program is in execution

More on the stack frame

- suppose main calls func_1, func_1 calls func_2, ..., func_{i-1} calls func_i; and let the func_i is being currently executed: then the stack frames are organized from the top of the stack region, one corresponding to each of main, func_1, func_2, ..., func_i, respectively

- *return address* of a stack frame points to the address of the instruction that needs to be executed when this function returns

- *frame pointer* of a stack frame points to the beginning address of the current stack frame — useful to remove the stack frame when the function scope ceases

Program counter

- *program counter register* points to the address of the instruction that is being executed

Virtual memory vs physical memory

- RAM is the *physical memory*; hard disk etc., are said to be *secondary storage* devices
- memory assigned to each process (typically 4GB) is from *virtual memory*, which comprises of pages; each page is a contiguous 4KB block (typically): any page may reside either in physical memory or on the secondary storage

Benefits of having virtual memory

- conceptually use more memory than the physically available
- to give each process a private memory
- hiding the programmer from the fragmentation of physical memory
- managing shared memory

Page tables

- each process is associated with a *page table*
- each entry in the page table holds a flag indicating whether the corresponding page is in physical memory or not, and the corresponding address
- when a page that is not currently in physical memory is referred, the hardware raises a *page fault* exception, which causes the OS to
  - find a page of memory in physical memory and bring the page located in secondary storage to physical memory, and
  - accordingly update the appropriate page table entry