Memory Allocation

reserving space in memory

- How do we decide what address to put in a pointer?
- Can point to the address of an existing variable

```
p = & a
```

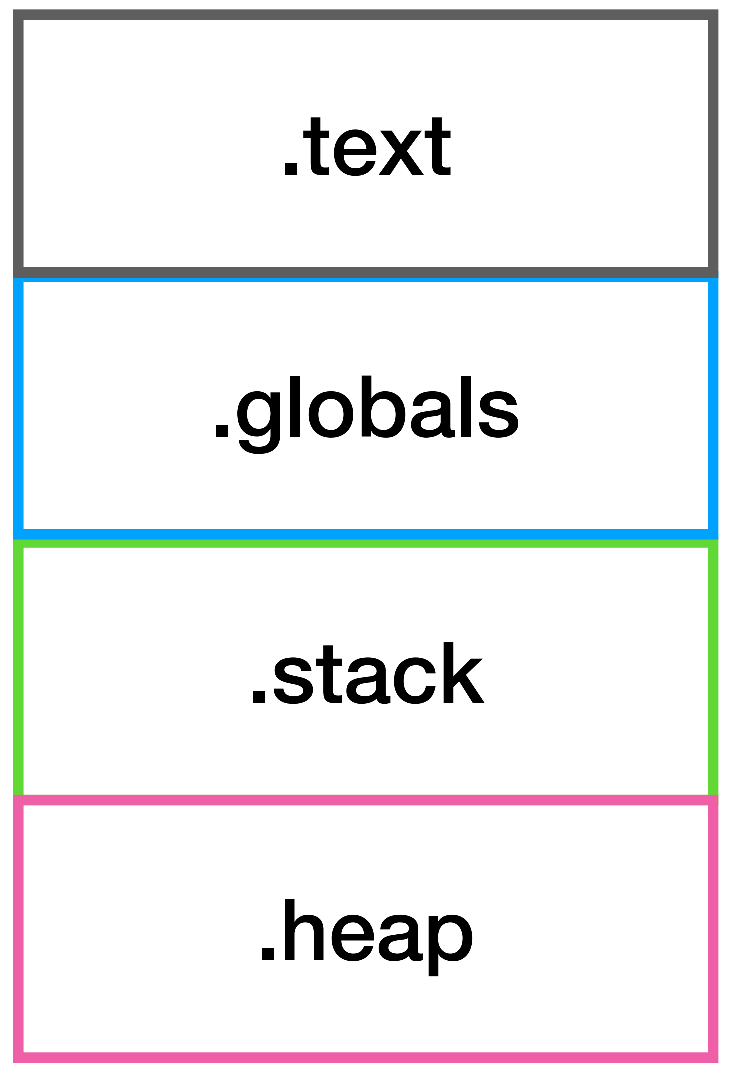
- Means addresses point either to:
  - **global** memory segment (global variables)
  - **stack** (local variables)

- Can we point elsewhere?

program heap

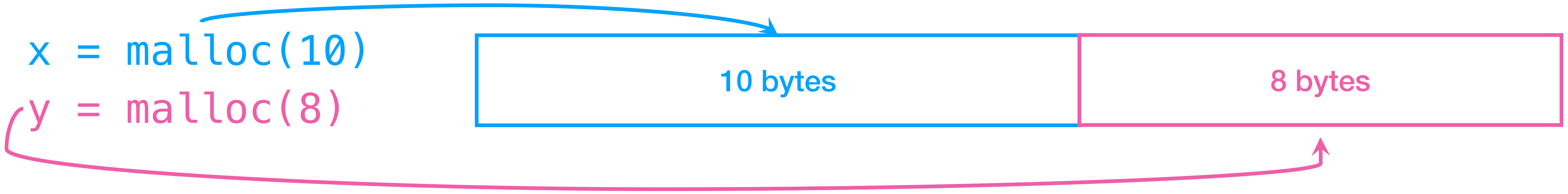
- Memory space of executing program also contains a large region called the **heap**
- Used for *dynamically allocated* data
  - Data not associated with a local variable or a global variable
  - Pointed to by pointers
  - No fixed location in memory

- How do we allocate that?



malloc/free

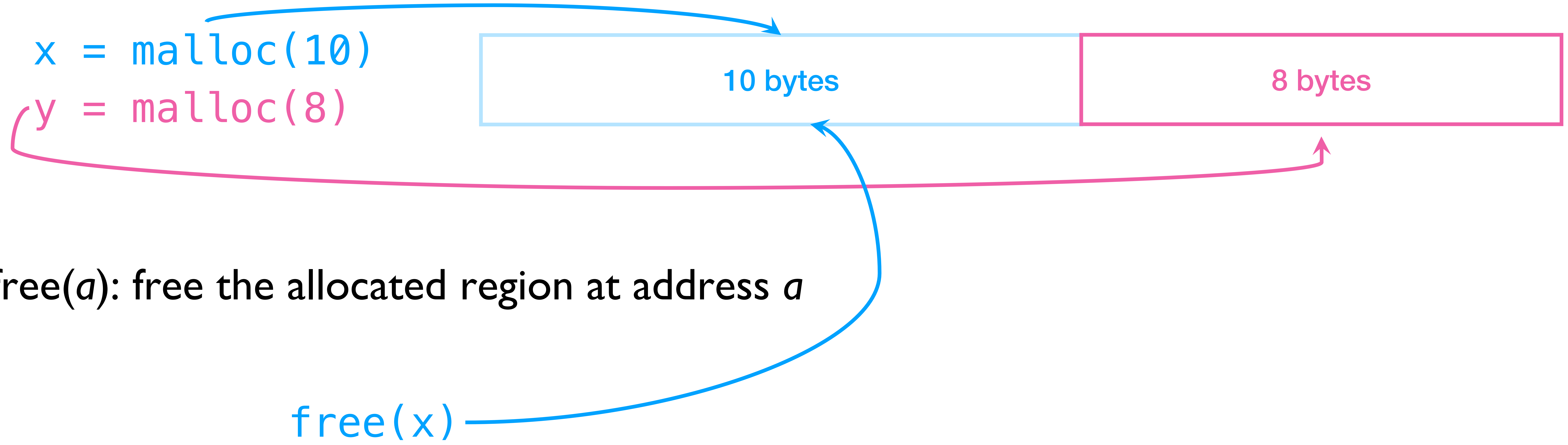
- malloc(*n*): allocate (reserve) *n* bytes of data in the heap, return the *address* of the first byte of the allocated region



```
x = malloc(10)
y = malloc(8)
```

malloc/free

- malloc(*n*): allocate (reserve) *n* bytes of data in the heap, return the *address* of the first byte of the allocated region



```
x = malloc(10)
y = malloc(8)
```

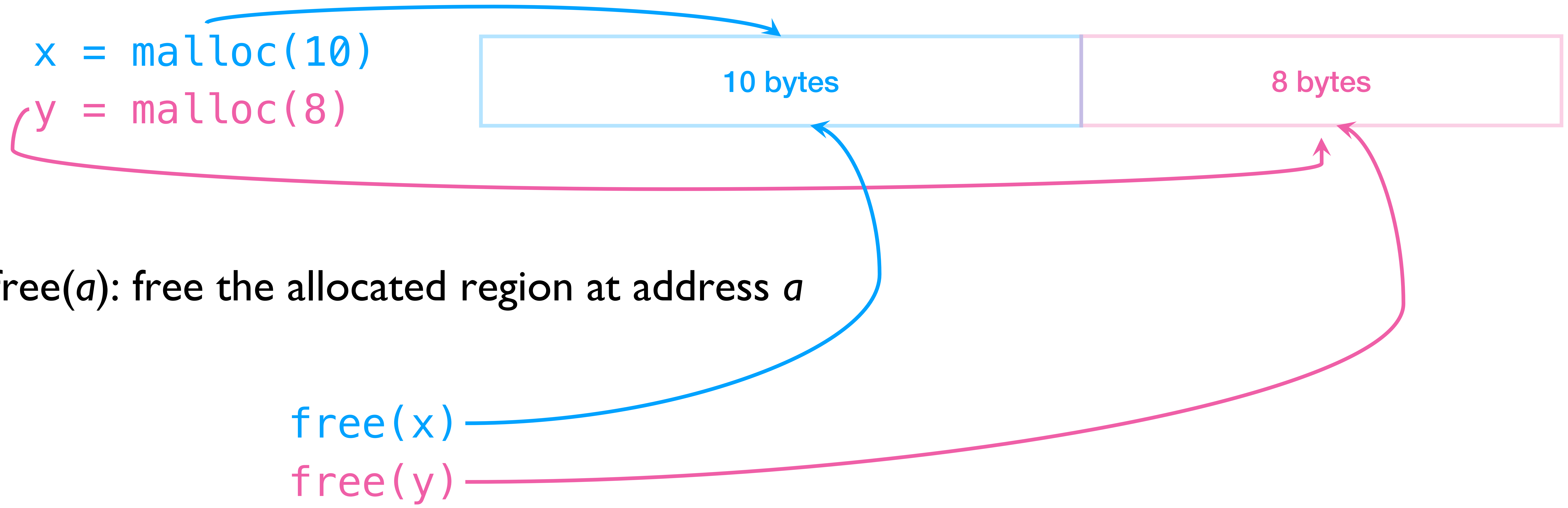
- free(*a*): free the allocated region at address *a*

```
free(x)
```

malloc/free

- malloc(*n*): allocate (reserve) *n* bytes of data in the heap, return the *address* of the first byte of the allocated region

```
x = malloc(10)
y = malloc(8)
```



- free(*a*): free the allocated region at address *a*

```
free(x)
free(y)
```

Guarantee: malloc will not return a region that overlaps with a current location

implementing malloc and free

- Implementation of memory allocator (malloc/free) is the responsibility of the operating system or the virtual machine
- Language usually provides a standard library that interfaces with the operating system to perform allocation
- In our course project, we don't have a standard library or an operating system
  - But the RISC simulator is essentially a virtual machine
- malloc/free implemented as "magic" instructions in the simulator
  - Compiler should detect invocations of malloc/free and generate magic instructions

next: arrays