CS 1550

Week 11 – Lab 4

Teaching Assistant

Maher Khan

CS 1550 – Project 3 Autograder is out

- **Due**: Friday, March 29, 2019 @11:59pm
- **Late**: Sunday, March 31, 2019 @11:59pm
  - 10% reduction per late day

For Python implementation

- Add the following to top of your script:

```
#!/usr/bin/env python
```

Project 3

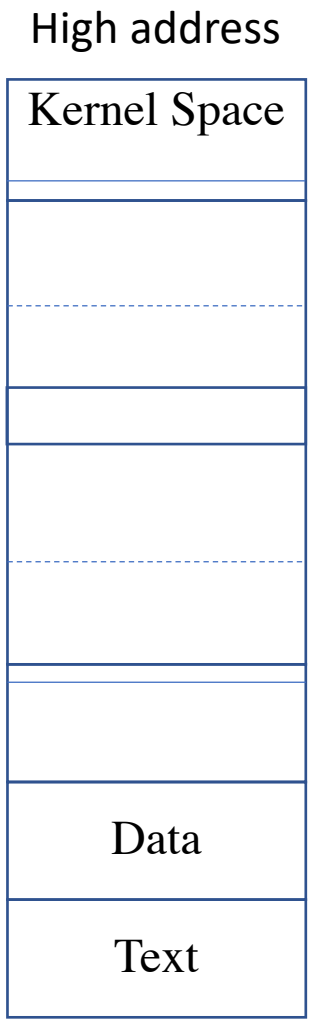
- Some implementation hints:
  - Make sure your program is efficient or it will time out
  - You only should go through the trace file once
  - Make use of data structures such as Linked List, HashMap, etc
  - Autograder may introduce very large trace files

Memory layout

```
int  t = 0;  // Data
…
int main() {
                …

}
```

Memory layout

```
int  t = 0;  // Data
…
int main() {
                …

}
```

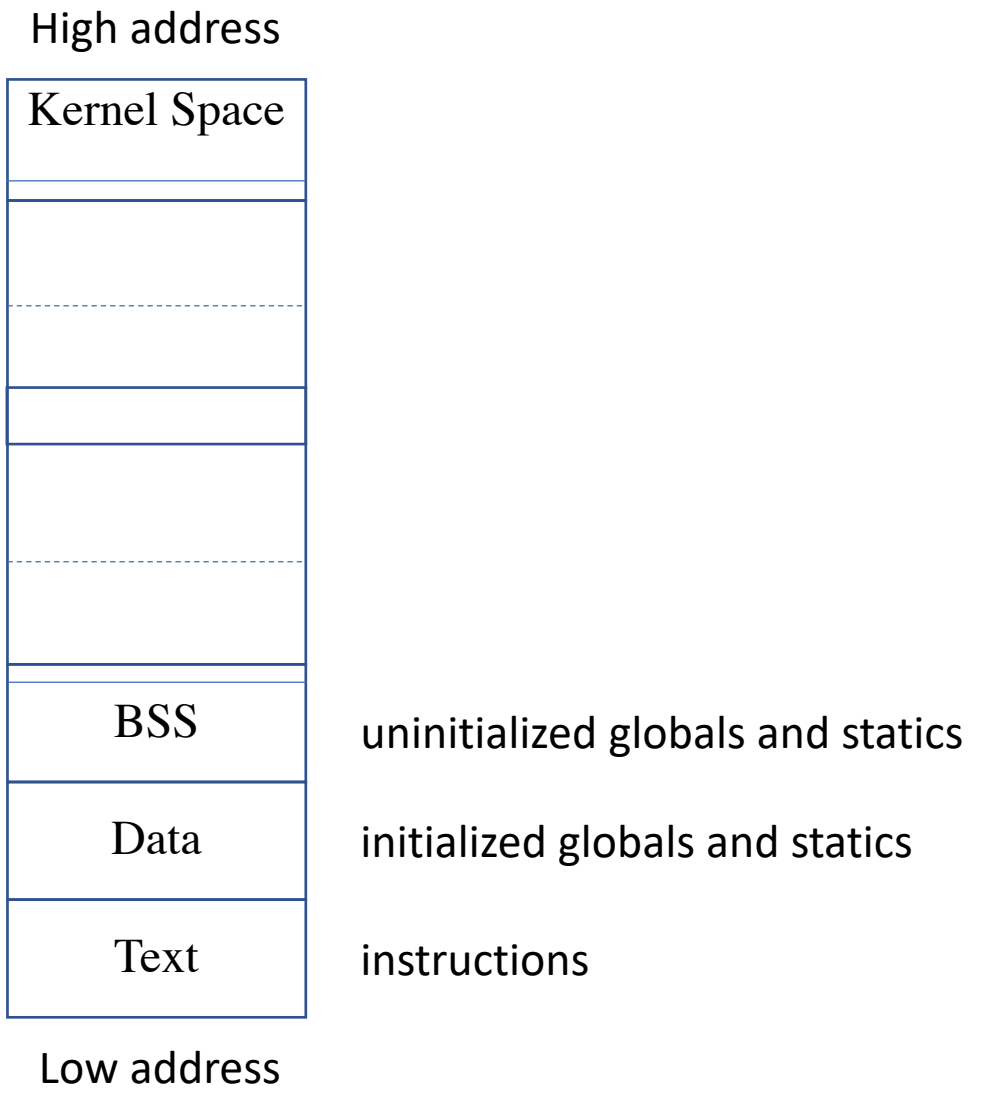


initialized globals and statics

instructions

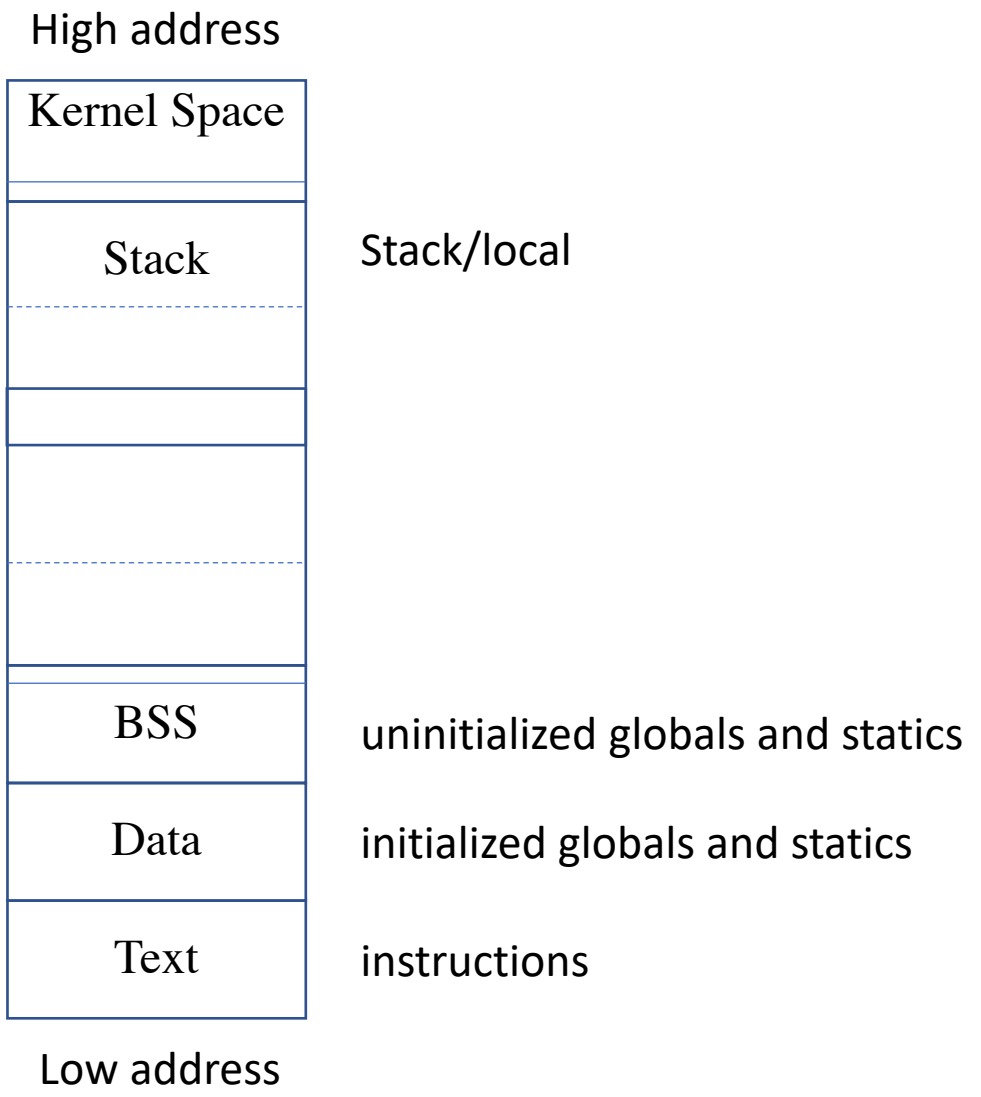
Memory layout

```
int  t = 0;  // Data
int m;       // BSS
…
int main() {
            …

            static  int  j;          // BSS

}
```



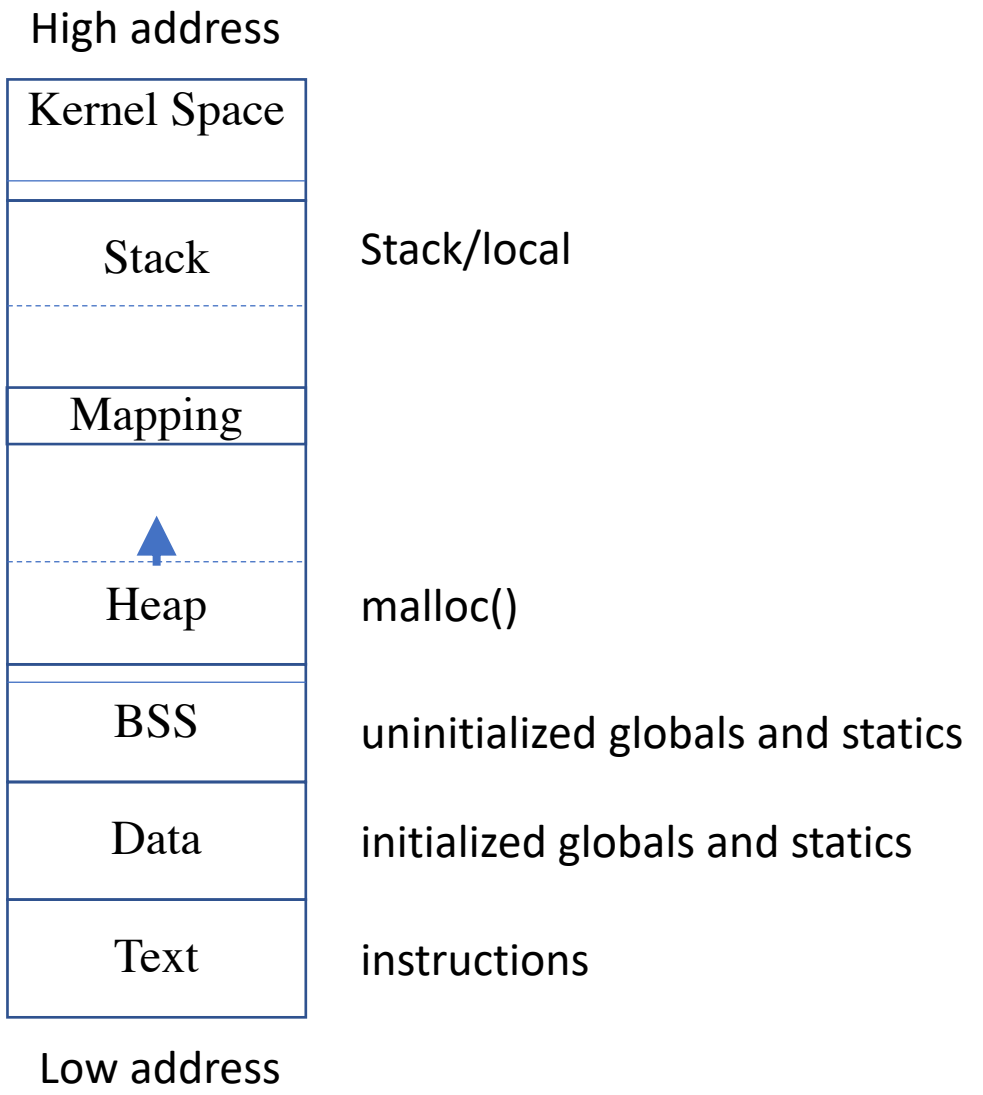
Memory layout

```
int  t = 0;  // Data
int m;       // BSS
…
int main() {
            …
            int  i;                 // Stack
            static  int  j;         // BSS


}
```



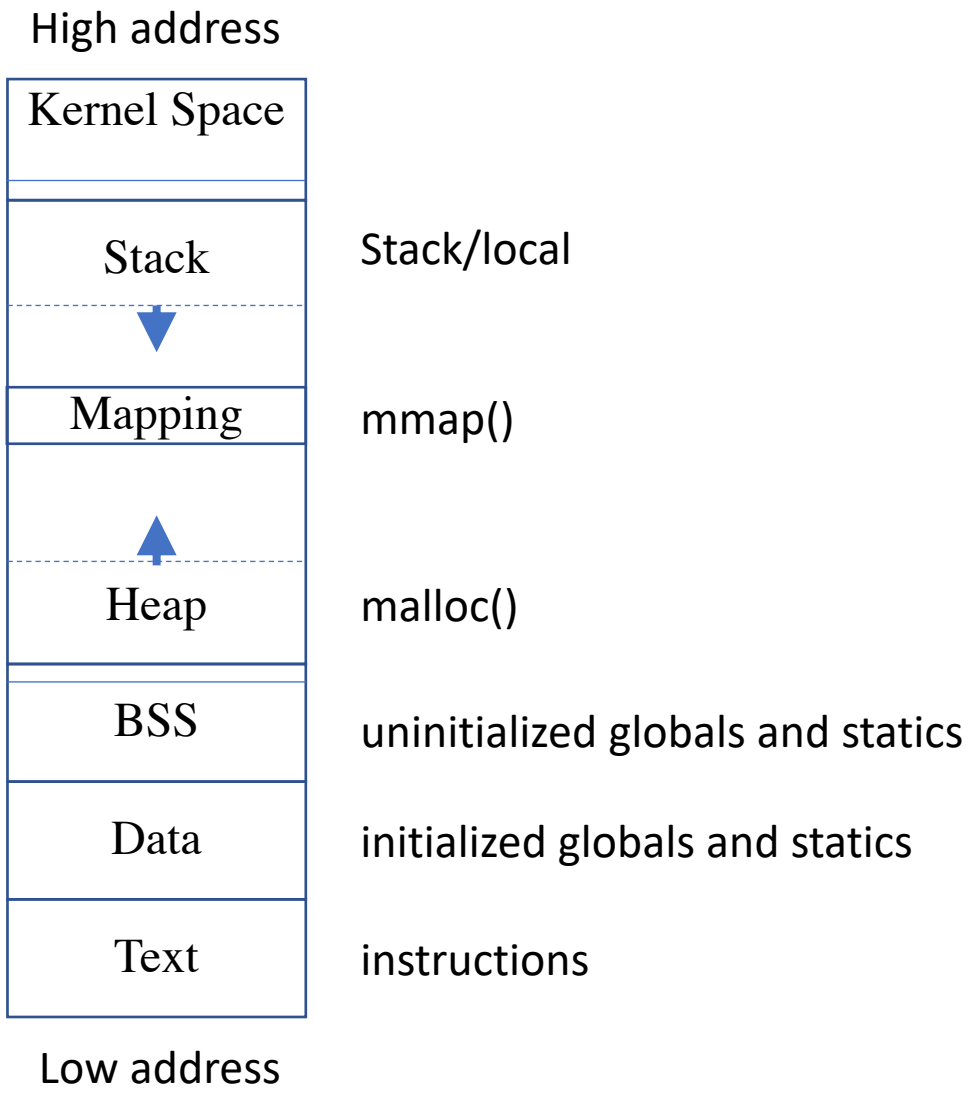
Memory layout

```
int  t = 0;  // Data
int m;       // BSS
…
int main() {
            …
            int  i;                 // Stack
            static  int  j;         // BSS

            // ptr: Stack
            // 4B pointed by ptr: Heap
            char * ptr = (char*)malloc(4);

}
```



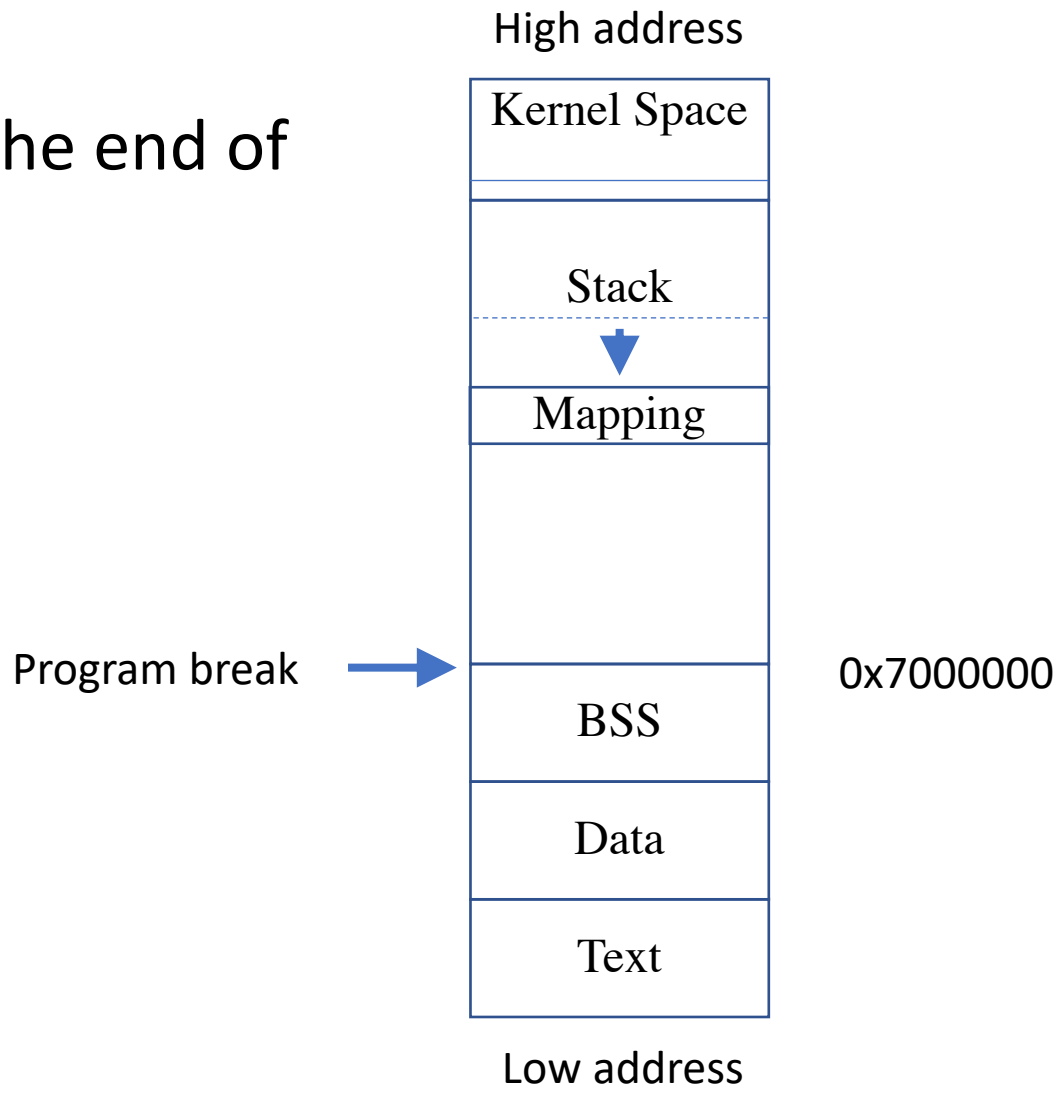
Memory layout

```
int  t = 0;  // Data
int m;       // BSS
…
int main() {
        …
        int  i;                 // Stack
        static  int  j;         // BSS

        // ptr: Stack
        // 4B pointed by ptr: Heap
        char * ptr = (char*)malloc(4);

         // mptr: Stack
        // 4K pointed by mptr: memory Mapping
        char * mptr = (char*)mmap(…,4096,…);
        …
}
```



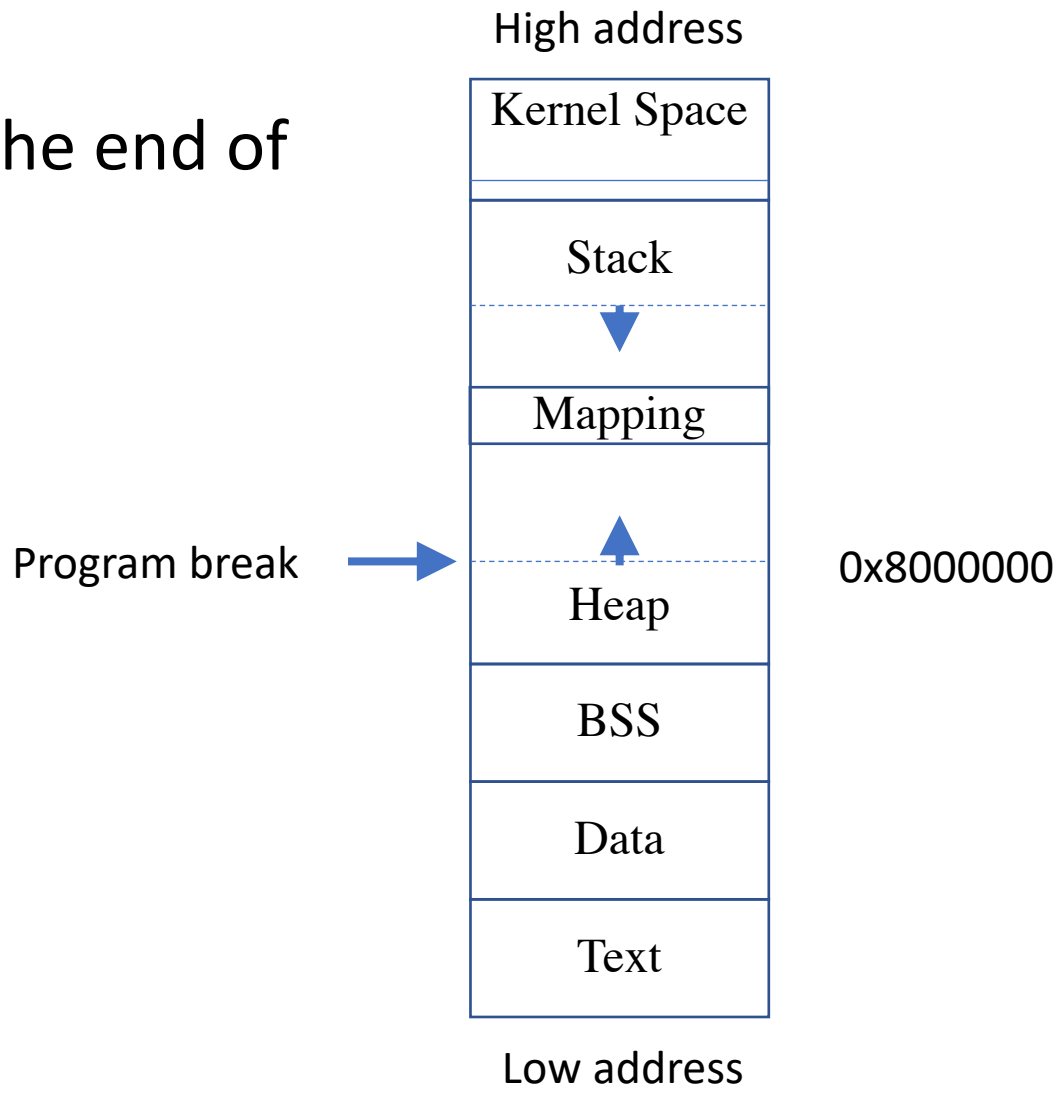
Program break

- Program break marks the end of the uninitialized data



Program break

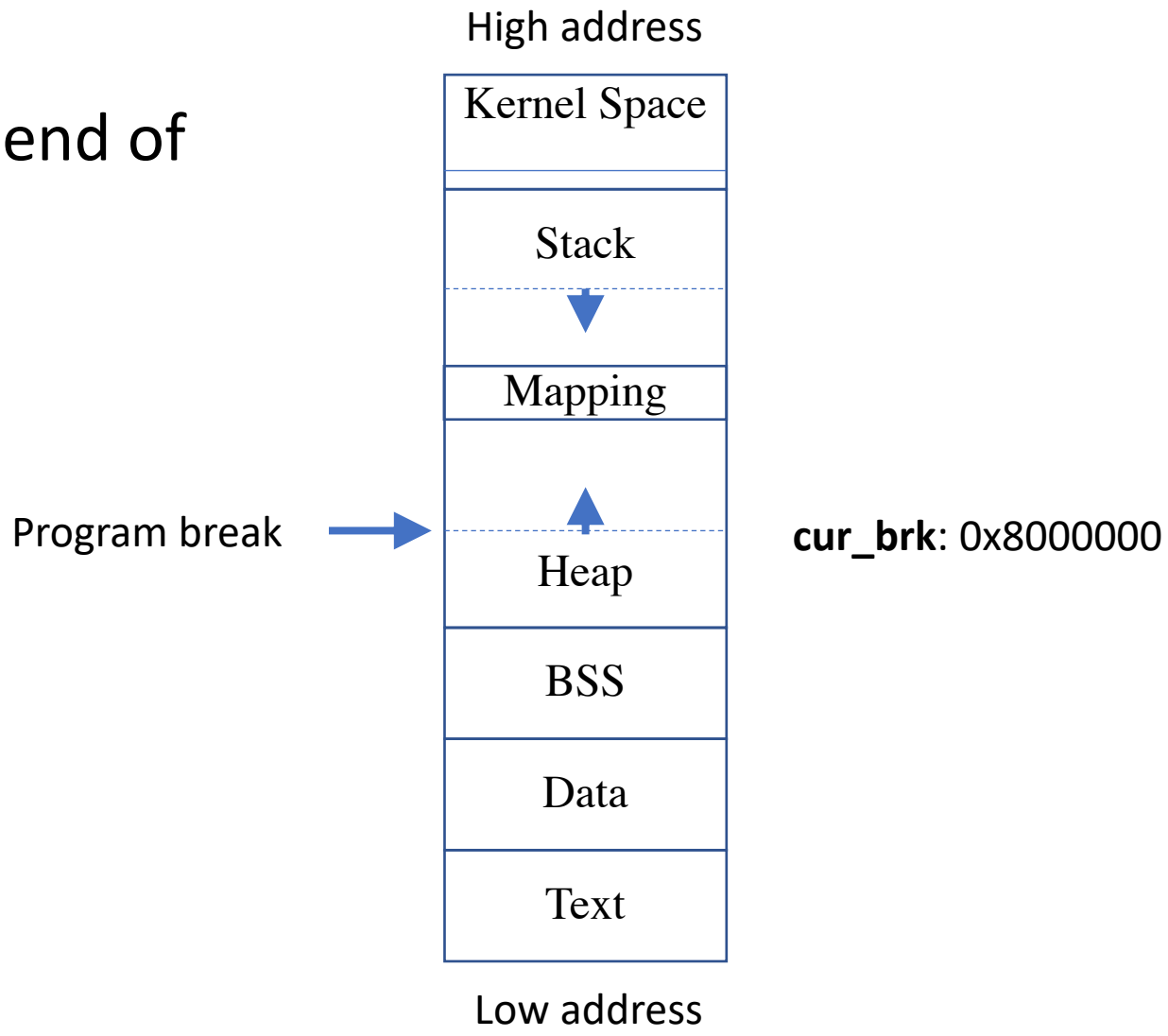
- Program break marks the end of the uninitialized data



Program break: The syscall sbrk

- Sbrk adds a size to the end of cur_brk

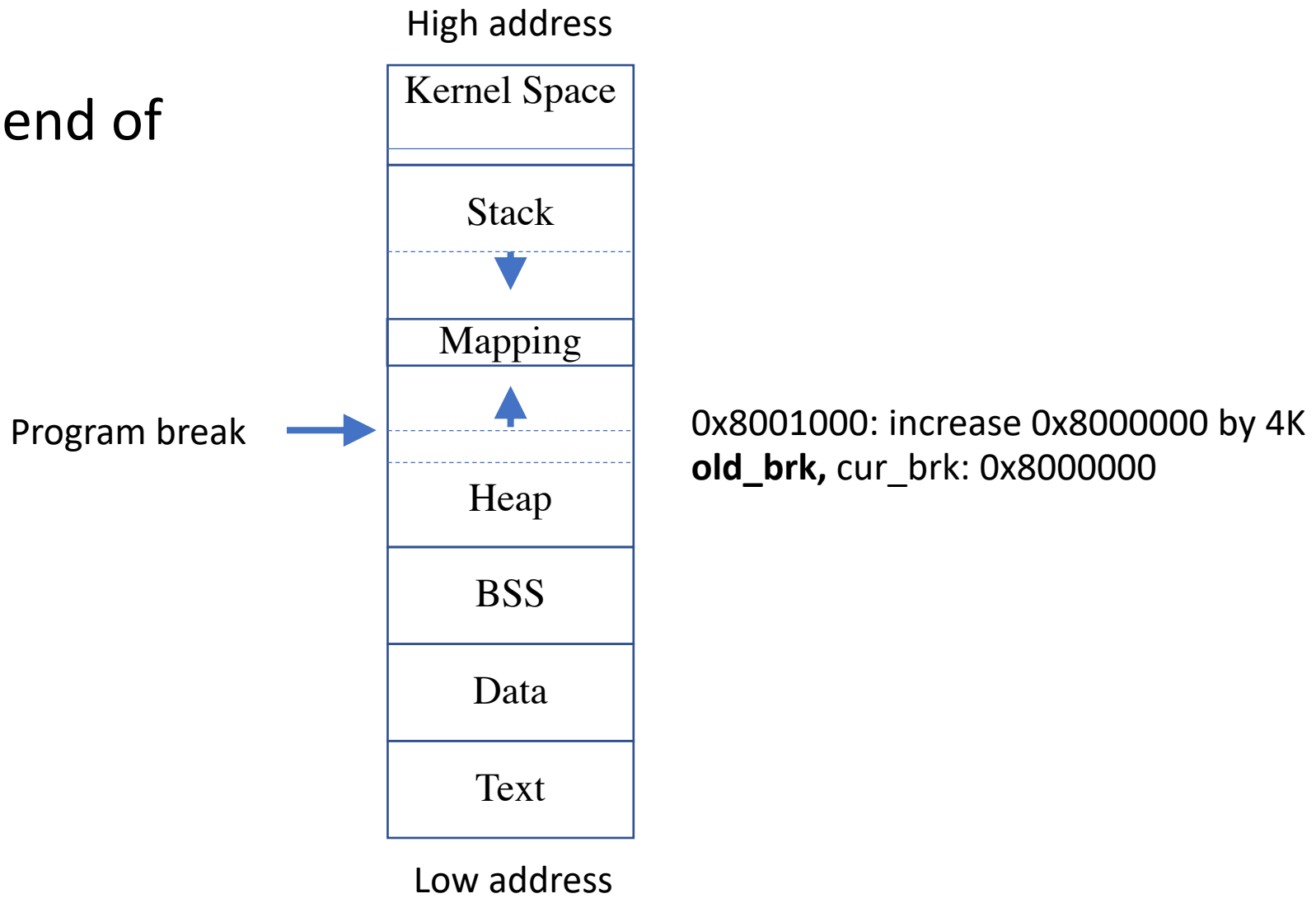
```
void *cur_brk = sbrk(0);

void *old_brk = sbrk(1024);

void *new_brk = sbrk(0);
```



Program break: The syscall sbrk

- Sbrk adds a size to the end of cur_brk

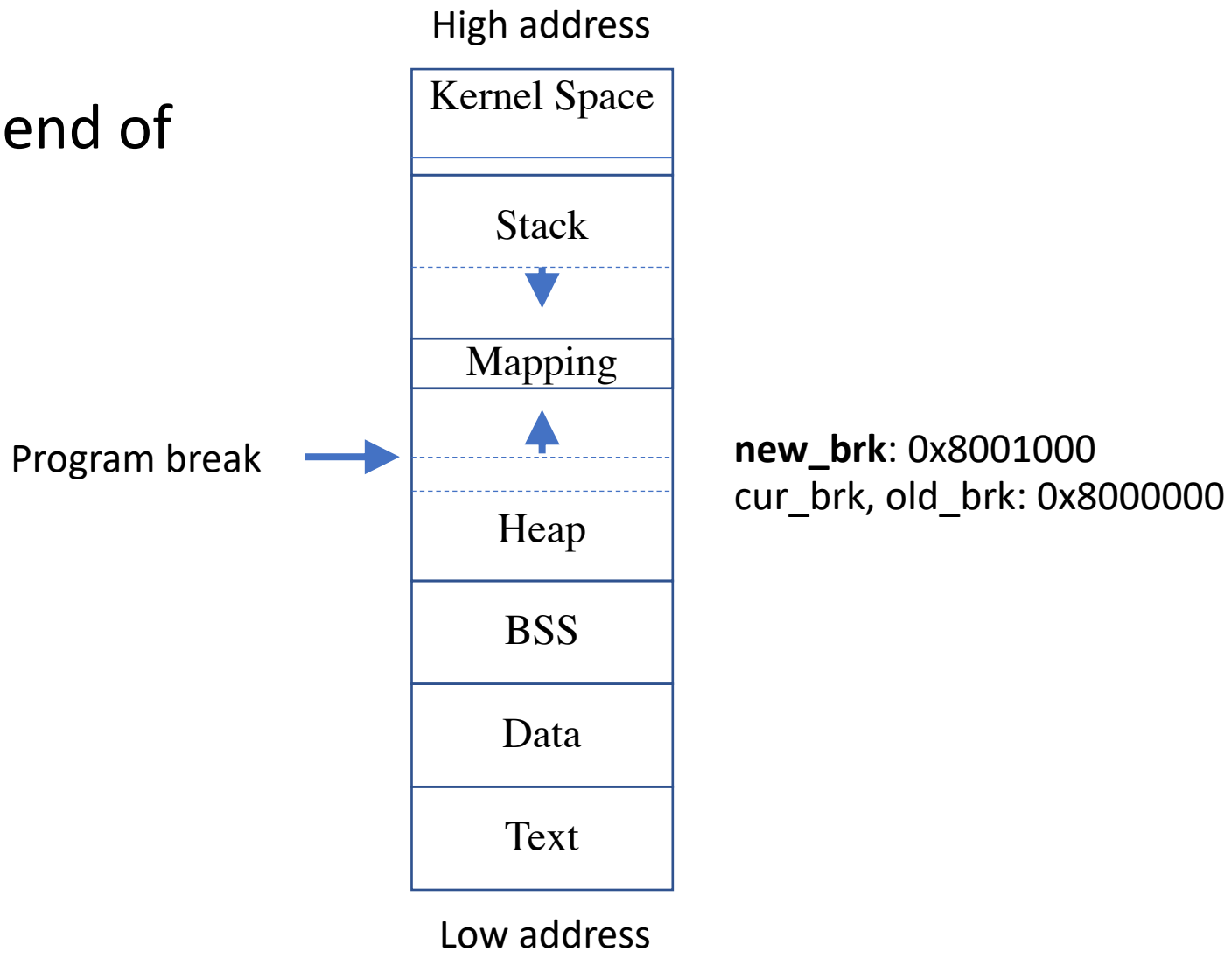
```
void *cur_brk = sbrk(0);

void *old_brk = sbrk(4096);

void *new_brk = sbrk(0);
```



Program break: The syscall sbrk

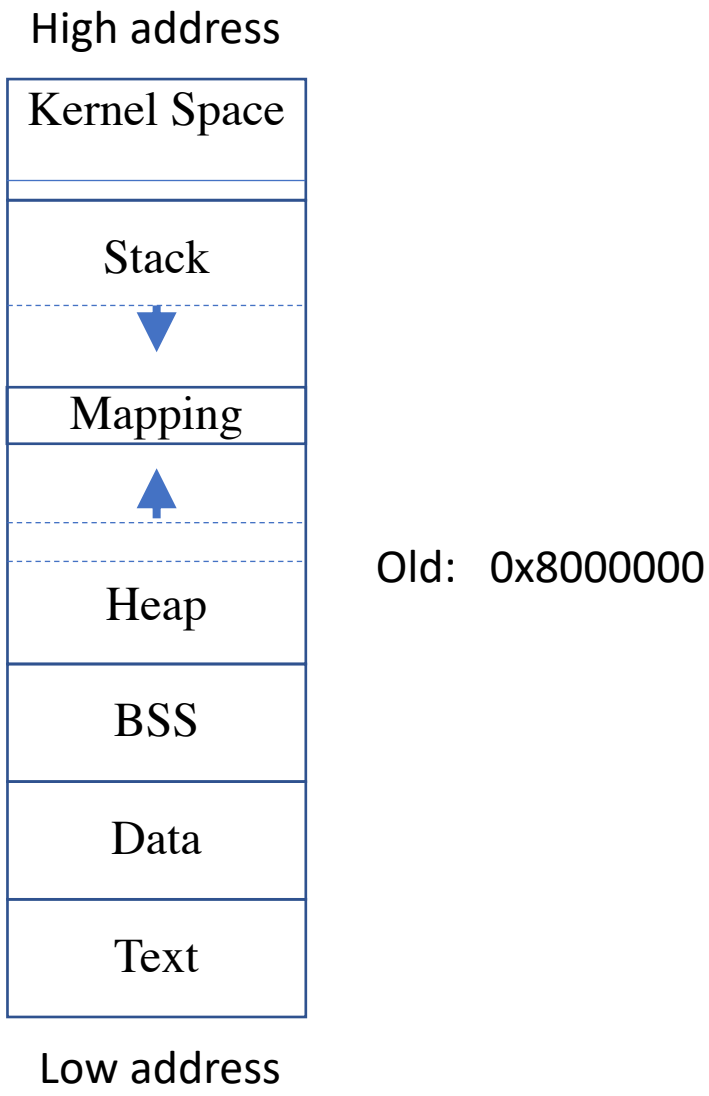
- Sbrk adds a size to the end of cur_brk

```
void *cur_brk = sbrk(0);

void *old_brk = sbrk(4096);

void *new_brk = sbrk(0);
```



Program break: The syscall brk

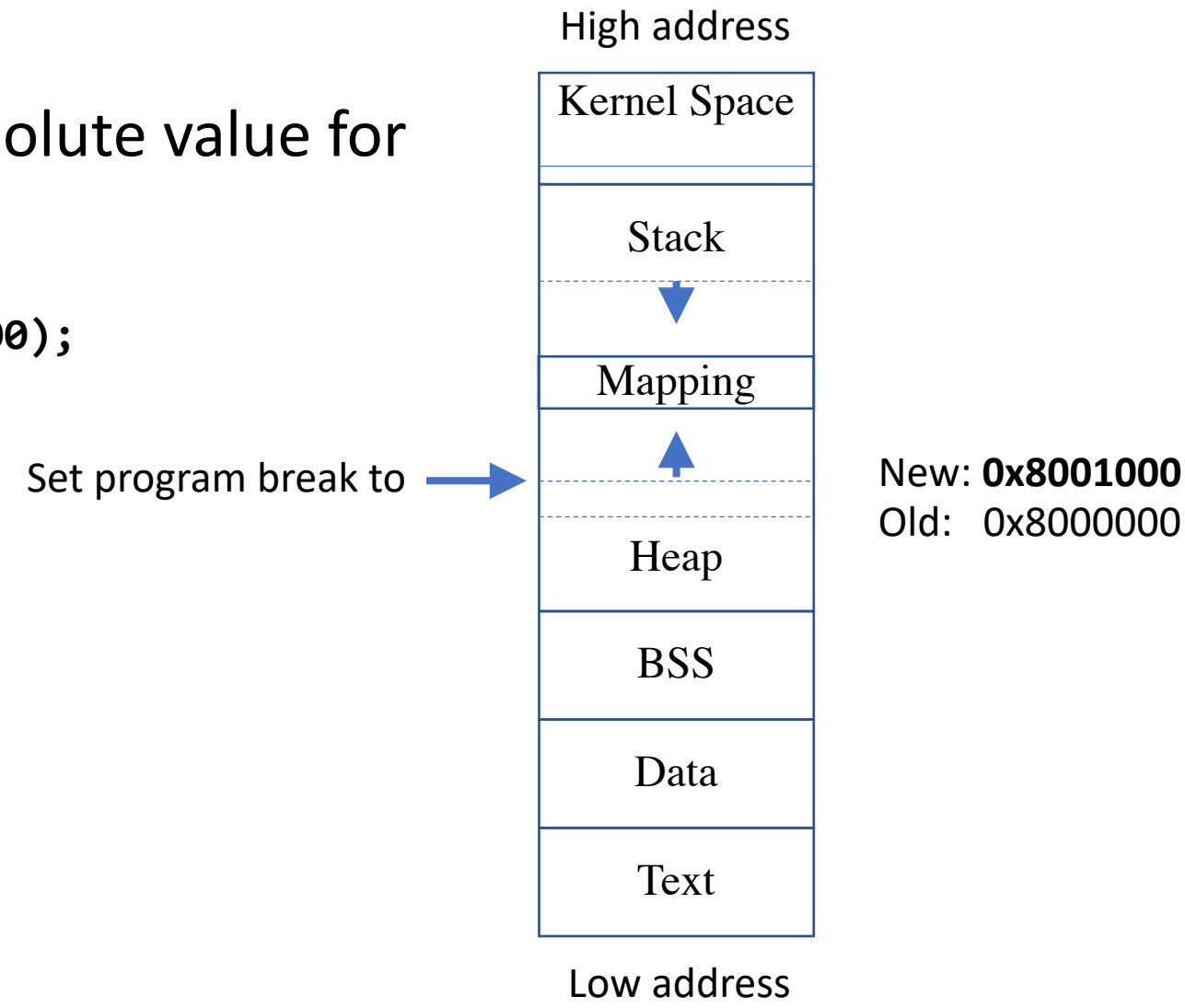
- brk defines the absolute value for heap's end



Program break: The syscall brk

- brk defines the absolute value for heap's end

```
int ret =  brk(0x8001000);

if (ret != 0) {
        // error
}
```



Sbrk on XV6

The **sys_sbrk()** in **sysproc.c** is the XV-6 implementation for sbrk.

```
…
addr = proc->sz;
if(growproc(n) < 0)
     return -1;
…
return addr;
```

Sbrk on XV6

The **sys_sbrk()** in **sysproc.c** is the XV-6 implementation for sbrk.

```
…
addr = proc->sz;   // get current brk
if(growproc(n) < 0)
     return -1;
…
return addr;
```

Sbrk on XV6

The **sys_sbrk()** in **sysproc.c** is the XV-6 implementation for sbrk.

```
…
addr = proc->sz;   // get current brk
if(growproc(n) < 0)          // increase brk by n
     return -1;
…
return addr;
```

growproc

The growproc() in proc.c:

```
…
if(n > 0) {
        allocuvm();
} else if (n < 0) {
        deallocuvm();
}
```

growproc

The growproc() in proc.c:

```
…
if(n > 0) {                    // allocation
        allocuvm();            // allocate physical pages, update page table
} else if (n < 0) {
        deallocuvm();
}
```

growproc

The growproc() in proc.c:

```
…
if(n > 0) {                    // allocation
        allocuvm();            // allocate physical pages, update page table
} else if (n < 0) {            // deallocation
        deallocuvm();          // update page table, free physical page
}
```

Physical memory allocation

Given 4KB per page and allocating an array with size of 100 pages:

```
char * ptr = (char*) malloc (4096 * 100);
```

Physical memory allocation

Given 4KB per page and allocating an array with size of 100 pages:

```
char * ptr = (char*) malloc (4096 * 100);
```

- This only allocates virtual memory: ptr to ptr+4096*100
- How about physical memory?

Physical memory allocation

Given 4KB per page and allocating an array with size of 100 pages:

```
char * ptr = (char*) malloc (4096 * 100);
```

- This only allocates virtual memory: ptr to ptr+4096*100

XV6: Immediately allocate all 100 physical page frames

Problems?

Physical memory allocation

Given 4KB per page and allocating an array with size of 100 pages:

```
char * ptr = (char*) malloc (4096 * 100);
// assume ptr is 0x8000000, i.e., a page-aligned virtual address
```

- This only allocates virtual memory: ptr to ptr+4096*100

Lab 4: allocate physical page frame upon the 1st access on that page.

```
ptr[4096*99 + 50] = 'a';  //  the 1st access on the 100th page
```

Page Fault

- Page Table: Stores mapping from virtual page to physical page frame

  E.g., Virtual Page 0x8000000 -> Physical 0x400000

- Translating a virtual address to physical address:

  Virtual address → (TLB →) Page Table → Physical address

Page Fault

- Translating virtual address 0x8000005:
  1. Get its page-start-address 0x8000000, and **offset**-in-page **5**.
  2. Search (TLB &) Page Table to find the mapping of 0x8000000
  3. If found, e.g., 0x8000000->0x4000000:

     then physical address is 0x4000005.

  If not

     found, Page Fault

Page Fault

```
char *ptr = (char*) malloc(4096*100);
ptr[4096*99 + 50] = ‘a’;    // 1st access, no physical page frame: Page Fault
```

Page Fault

```
char *ptr = (char*) malloc(4096*100);
ptr[4096*99 + 50] = 'a';    // 1st access, no physical page frame: Page Fault
```

In XV6, Page Fault on ptr[4096*99 + 50] (inside 100th page):

Page Fault

char *ptr = (char*) malloc(4096*100);

ptr[4096*99 + 50] = ‘a’; // 1st access, no physical page frame: **Page Fault**

In XV6, Page Fault on ptr[4096*99 + 50] (inside 100th page):

1. Issue Page Fault trap. All traps are handled by trap() in **trap.c**.

Page Fault

char *ptr = (char*) malloc(4096*100);

ptr[4096*99 + 50] = ‘a’; // 1st access, no physical page frame: **Page Fault**

In XV6, Page Fault on ptr[4096*99 + 50] (inside 100th page):

1. Issue Page Fault trap. All traps are handled by trap() in trap.c.
2. **Handle Page Fault (*Hint: T_PGFLT, how to find the faulting addr*) in trap():**
   1) **Allocate a physical page frame for this 100th page**
   2) **Update page table**

Allocate physical pages, update page table

Recall the growproc() in proc.c.

```
if(n > 0) {                    // allocation
        allocuvm();            // allocate physical pages, update page table
} else if (n < 0) {            // deallocation
        deallocuvm();          // update page table, free physical page
}
```

Check how allocuvm & deallocuvm work.

(*Hint: PGROUNDDOWN, mappages)*

CS 1550 – Lab 4

- **Due**: Friday, 5th March, 2019 @11:59pm

CS 1550

Week 8 – Lab 4

Teaching Assistant

Maher Khan