CS 1550

Week 10 – Lab 4

Teaching Assistant

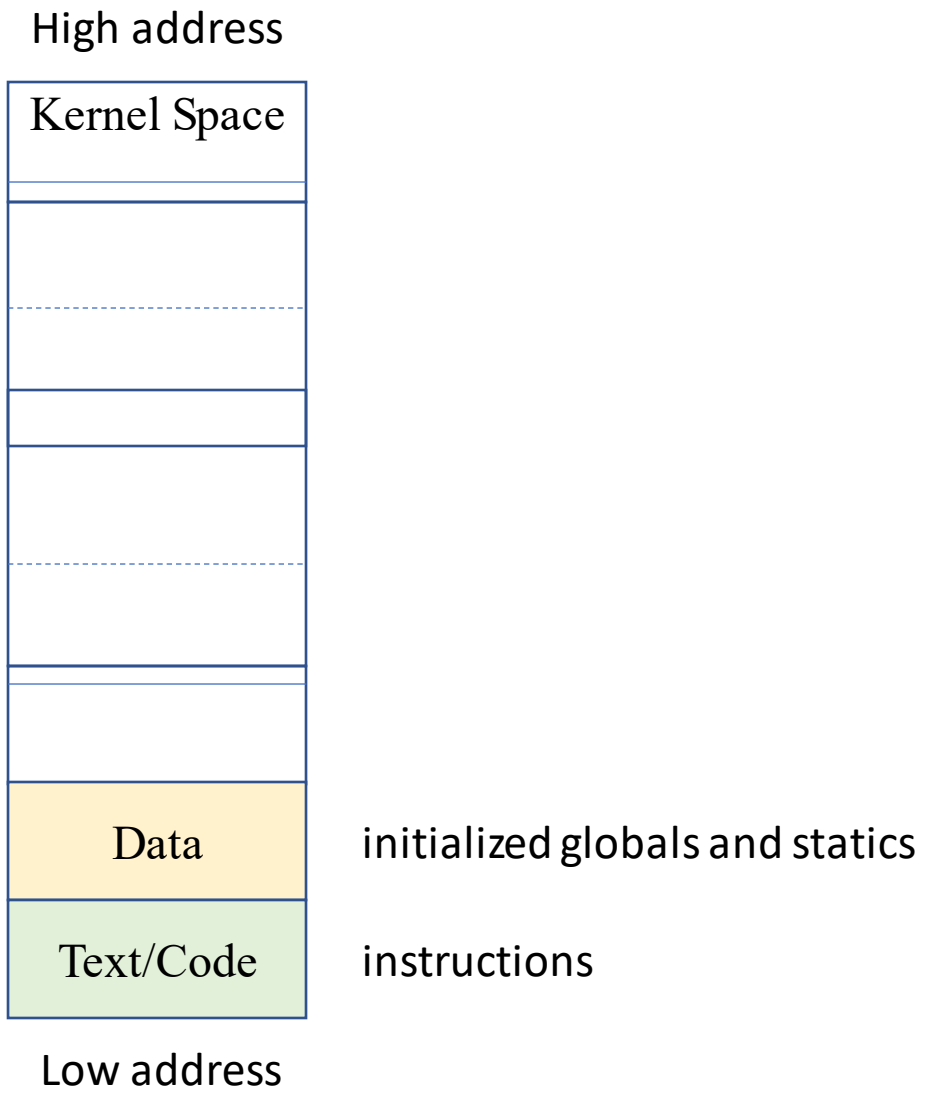
Henrique Potter

CS 1550 – Lab 4

- Implementing lazy page allocation in xv6

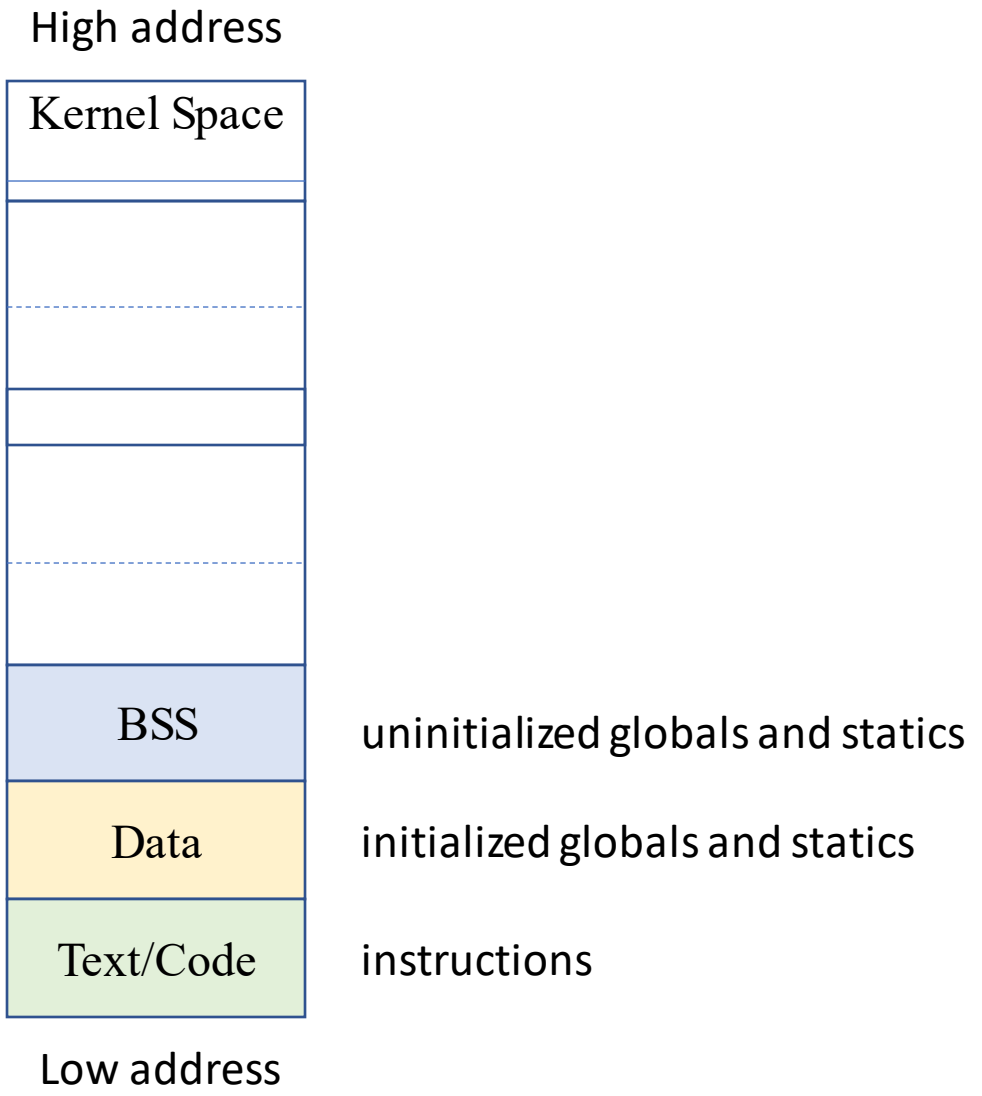
Memory layout

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

}
```



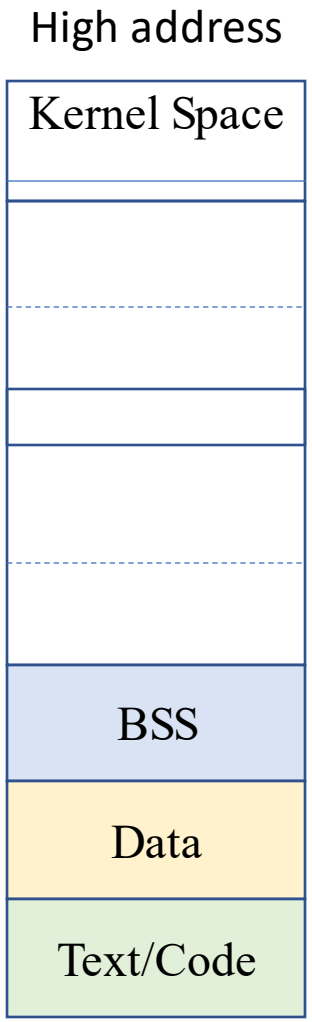
Memory layout

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

            static  int  j;          // BSS

}
```



Memory layout

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

            static  int  j;          // BSS

}
```



"**B**lock **S**tarted by **S**ymbol"

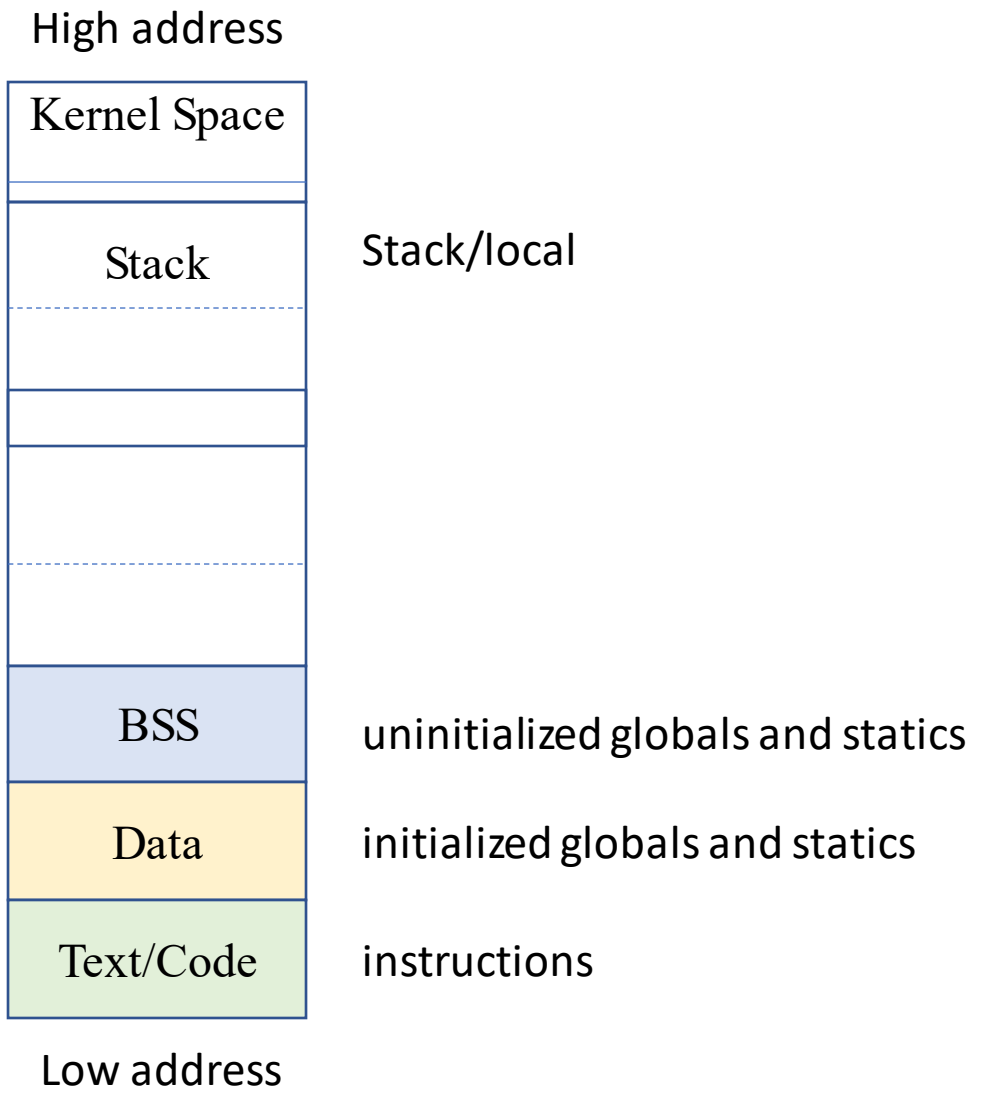
uninitialized globals and statics

initialized globals and statics

instructions

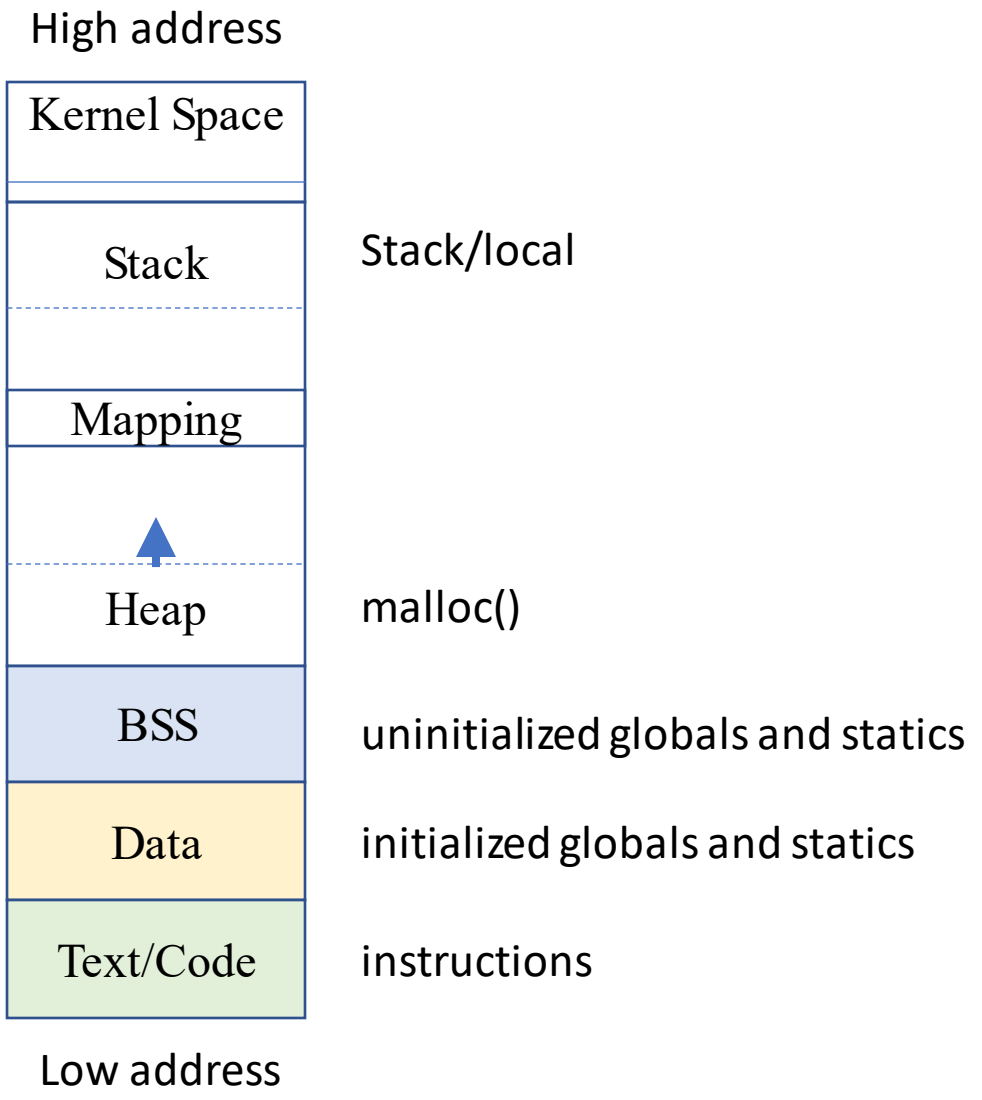
Memory layout

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



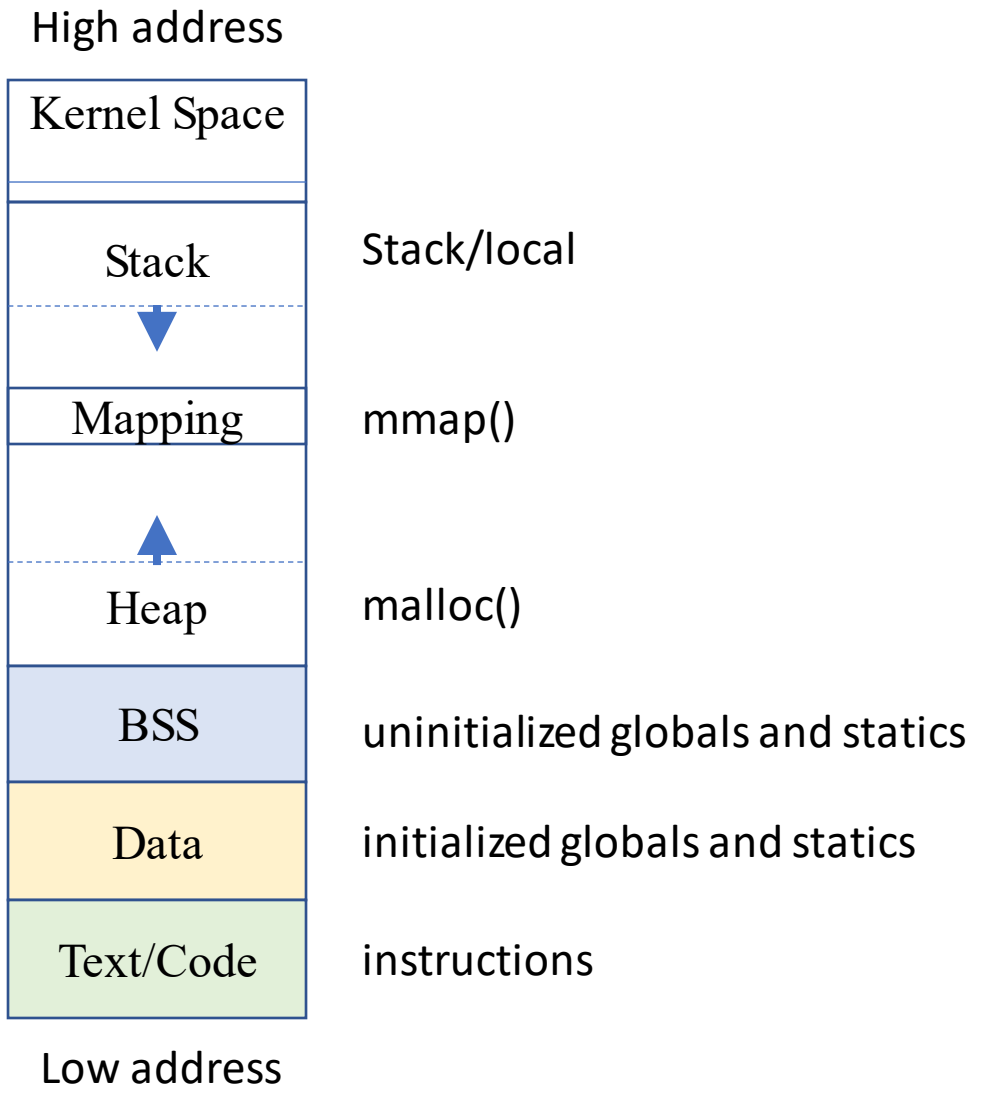
Memory layout

```
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

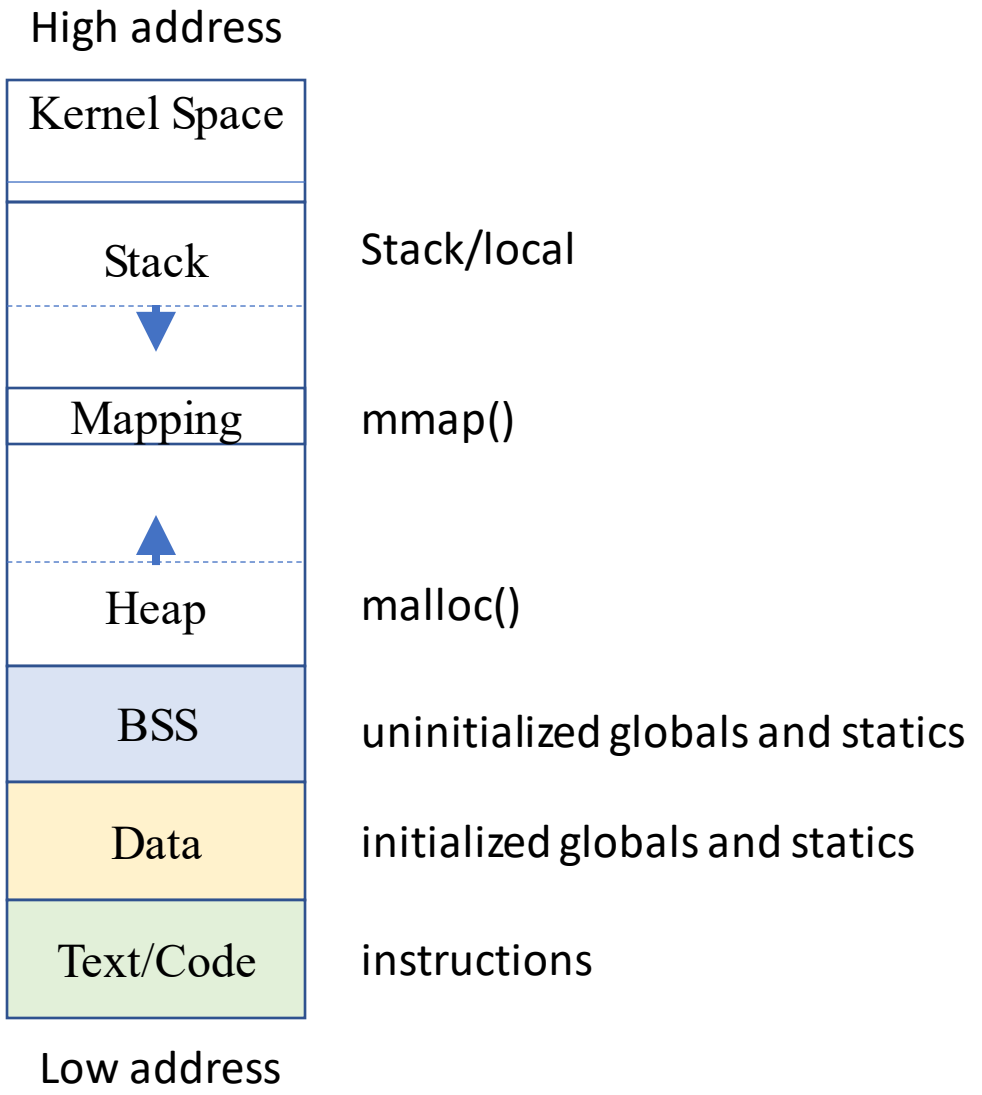
```
int  t = 0;  // Data
int m;       // BSS
…
int main() {
            …
            int  i;                  // Stack
            static  int  j;          // BSS

            // 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,…);
            …
}
```



Memory layout

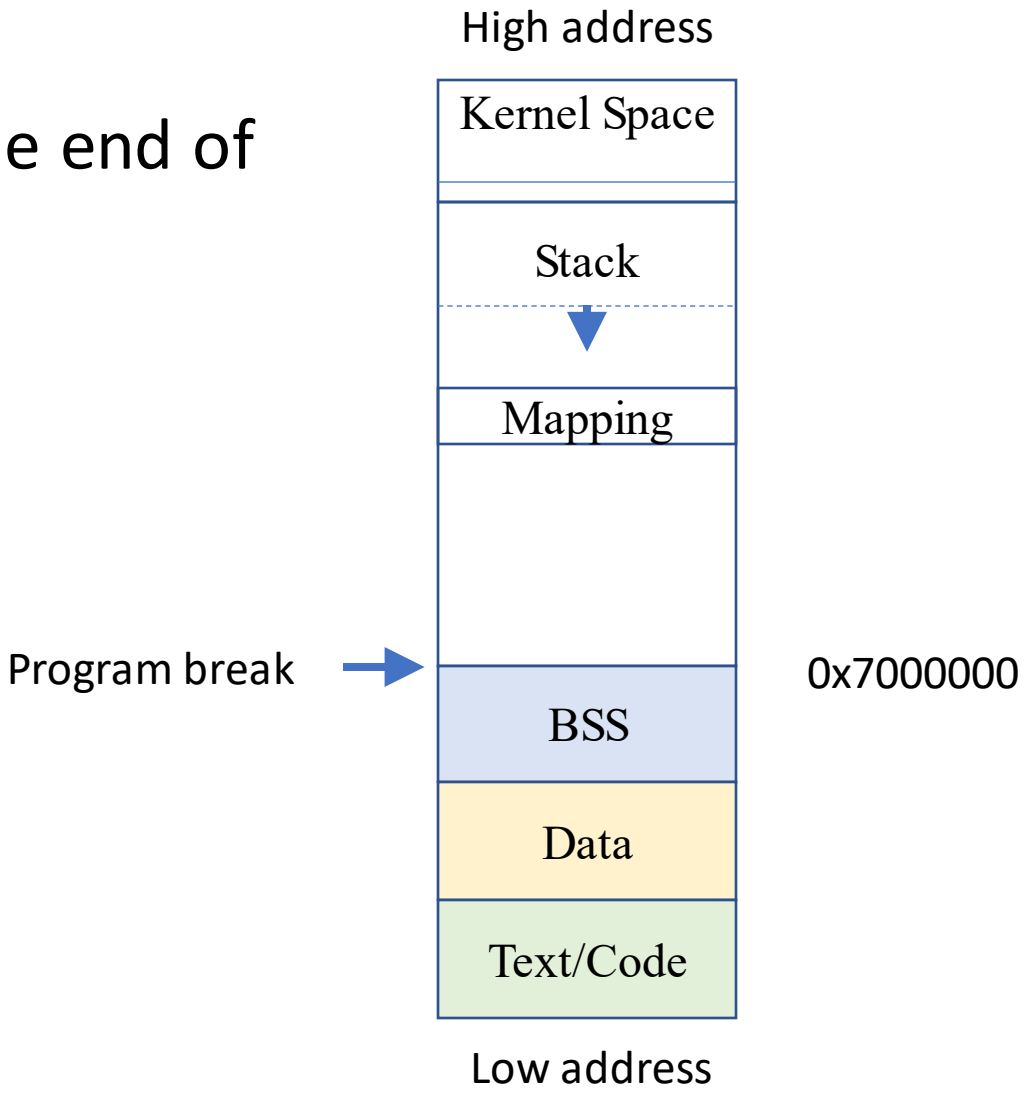
```
int  t = 0;  // Data
int m;       // BSS
…
int main() {
            …
            int  i;                 // Stack
            static  int  j;         // BSS

            // 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,…);
            …
}
```

xv6 do not have malloc!



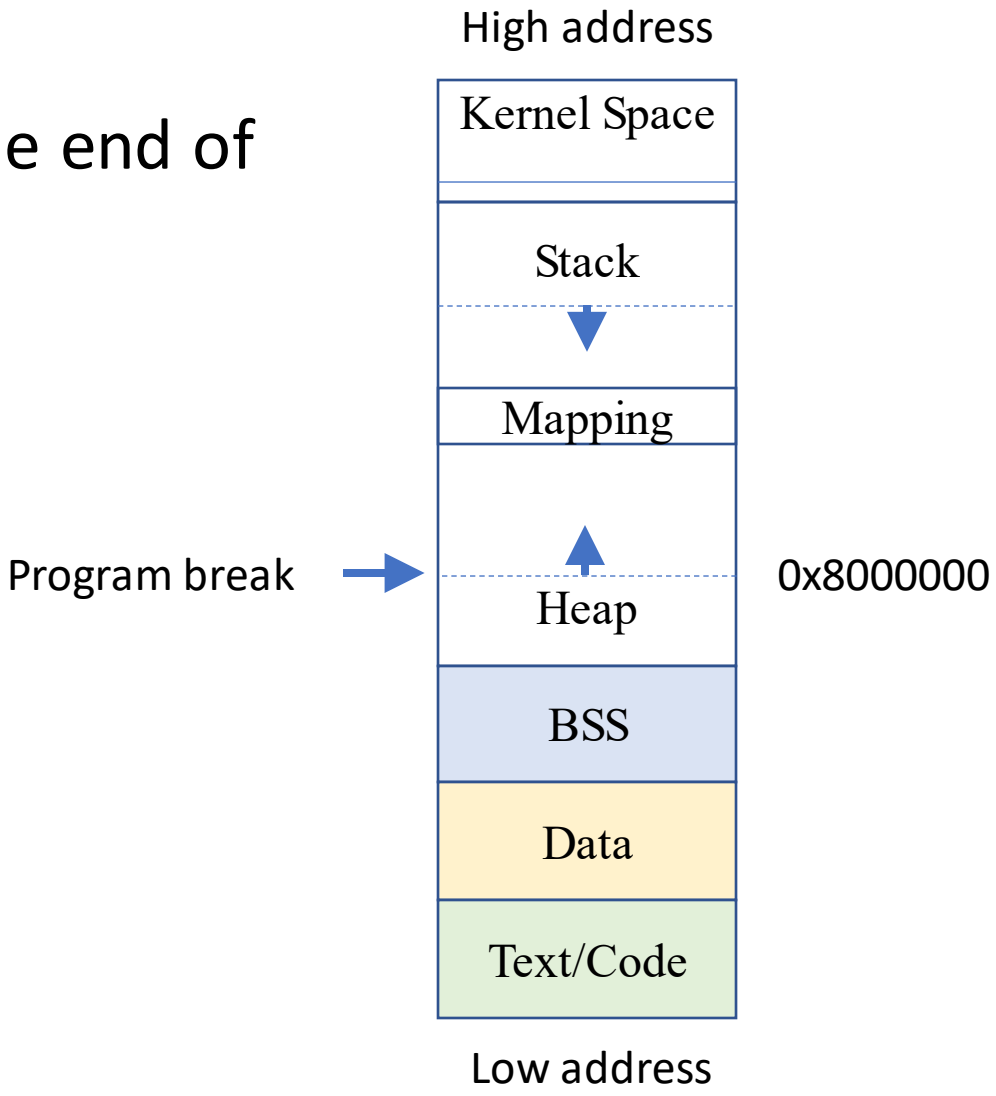
Program Break

- Program break marks the end of the uninitialized data



Program Break

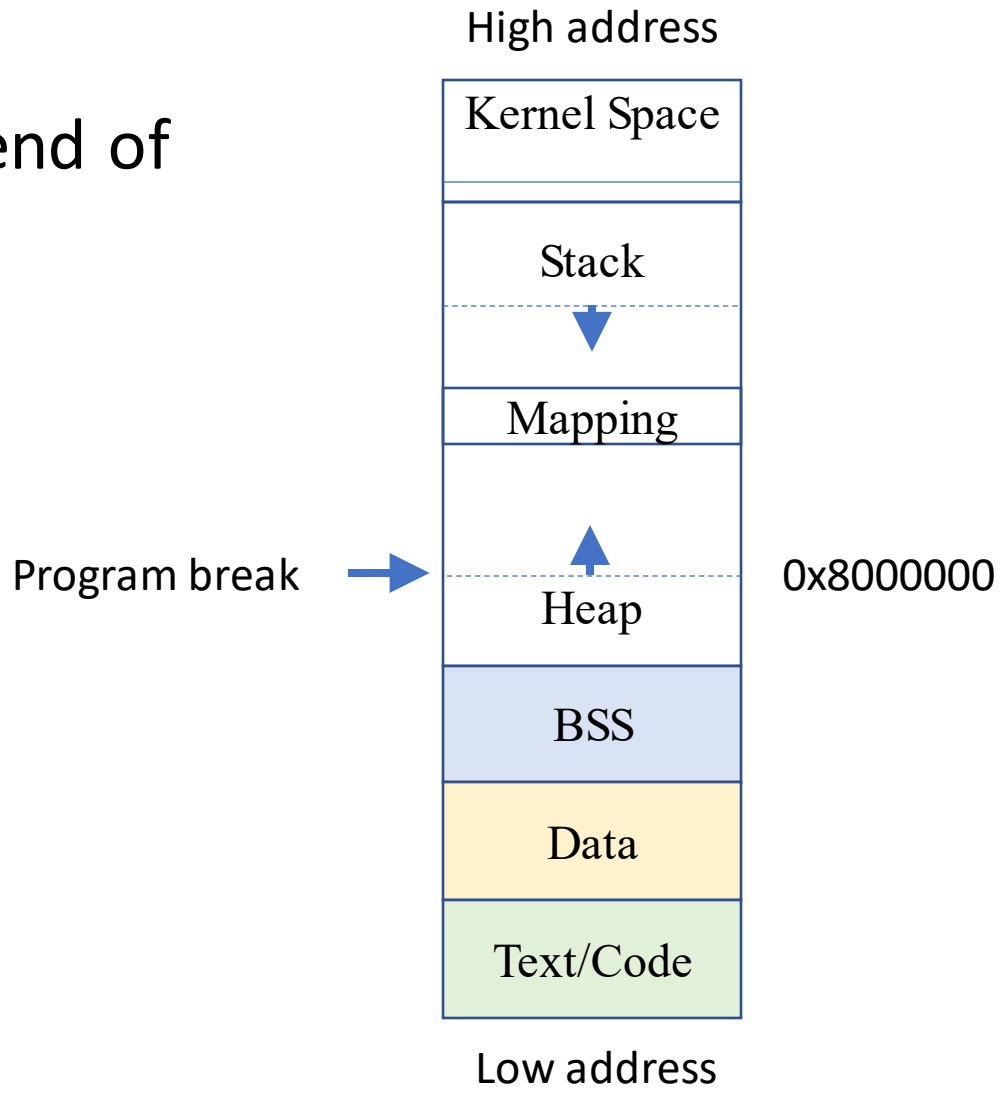
- Program break marks the end of the uninitialized data



Program break: The syscall brk

- Sbrk adds a size to the end of cur_brk

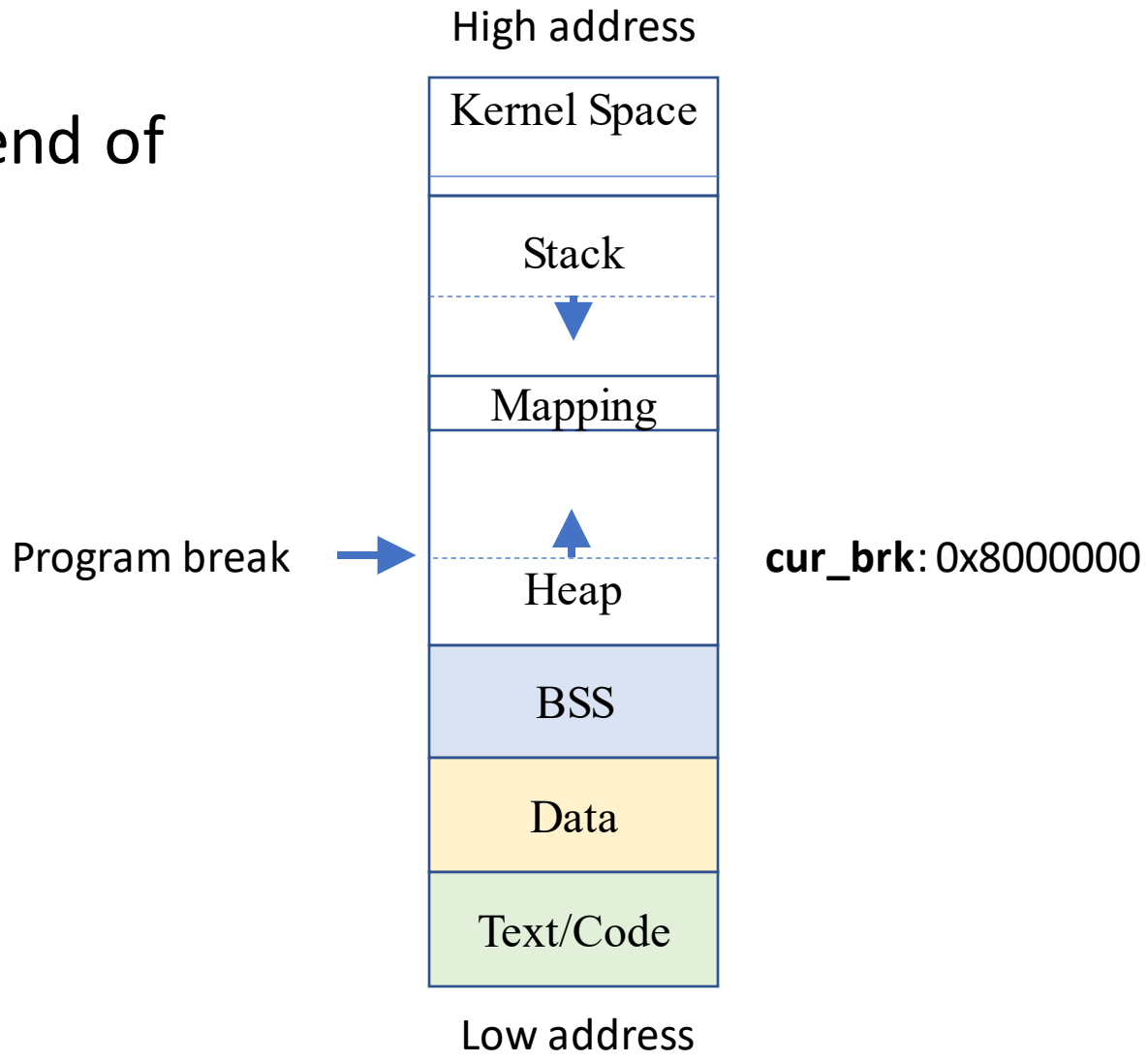
```
void *cur_brk = sbrk(0);
```



Program break: The syscall brk

- Sbrk adds a size to the end of cur_brk

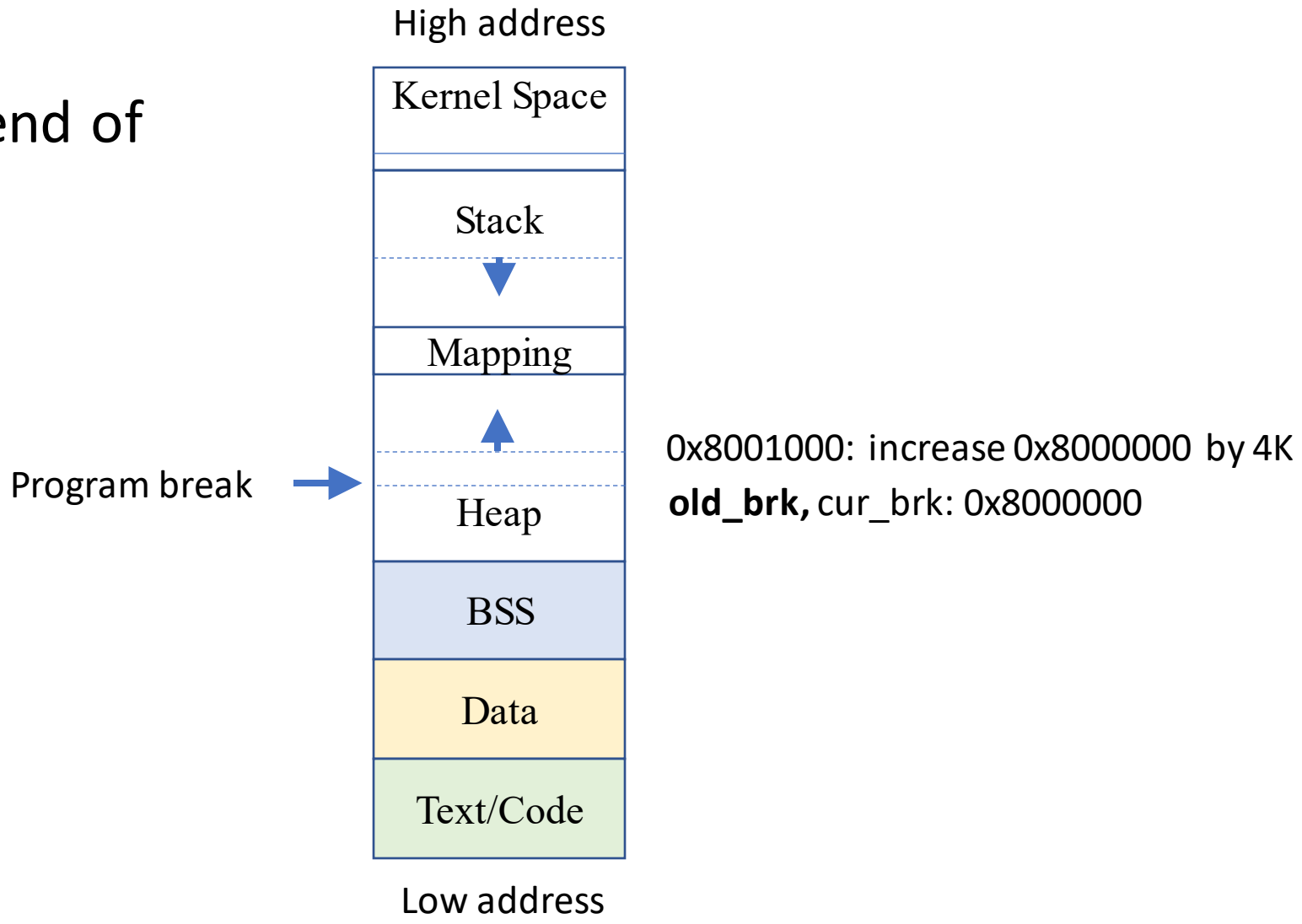
```
void *cur_brk = sbrk(0);
```



Program break: The syscall brk

- Sbrk adds a size to the end of cur_brk

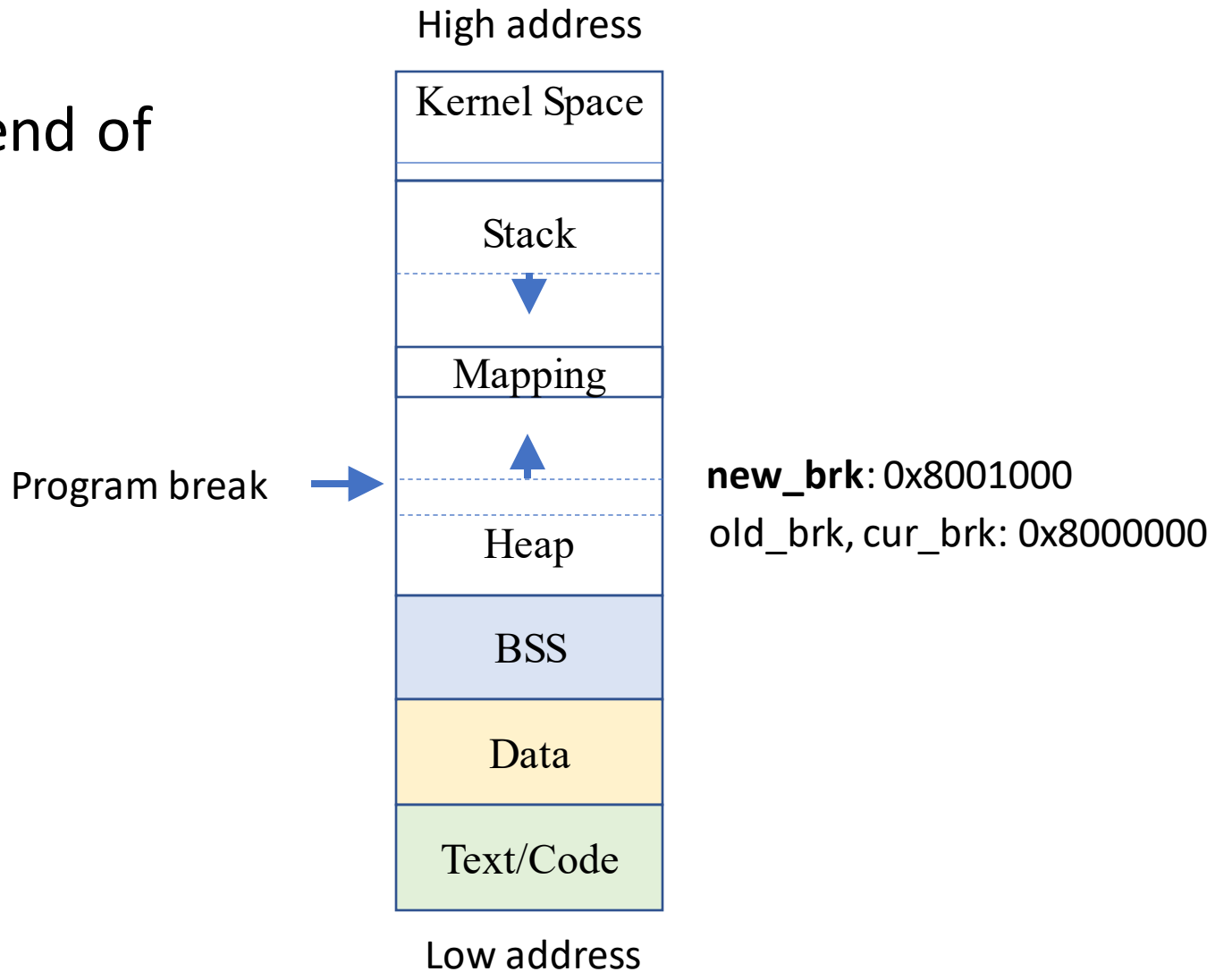
```
void *cur_brk = sbrk(0);

void *old_brk = sbrk(4096);
```



Program break: The syscall brk

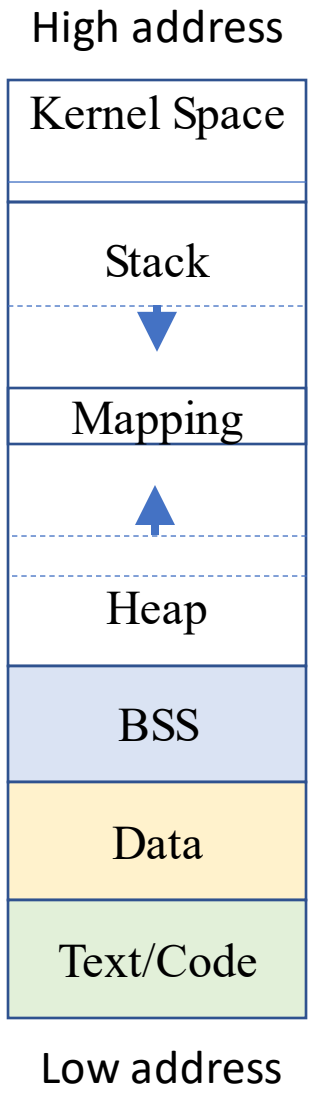
- Sbrk adds a size to the end of cur_brk

```
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



sbrk on XV6

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

```
int
sys_sbrk(void)
{
  int addr;
  int n;

  if(argint(0, &n) < 0)
    return -1;
  addr = myproc()->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.

```
int
sys_sbrk(void)
{
  int addr;
  int n;

  if(argint(0, &n) < 0)
    return -1;
  addr = myproc()->sz;
  if(growproc(n) < 0)
    return -1;
  return addr;
}
```

Get the current heapsize

sbrk on XV6

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

```
int
sys_sbrk(void)
{
  int addr;
  int n;

  if(argint(0, &n) < 0)
    return -1;
  addr = myproc()->sz;
  if(growproc(n) < 0)
    return -1;
  return addr;
}
```

Get the current heapsize

Increase heapsize by n

growproc on XV6

The ***growproc()*** is in ***proc.c***:

```
int
growproc(int n)
{
  uint sz;
  struct proc *curproc = myproc();

  sz = curproc->sz;
  if(n > 0){
    if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0)
      return -1;
  } else if(n < 0){
    if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0)
      return -1;
  }
  curproc->sz = sz;
  switchuvm(curproc);
  return 0;
}
```

growproc on XV6

The ***growproc()*** is in ***proc.c***:

```
int
growproc(int n)
{
  uint sz;
  struct proc *curproc = myproc();

  sz = curproc->sz;
  if(n > 0){
    if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0)
      return -1;
  } else if(n < 0){
    if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0)
      return -1;
  }
  curproc->sz = sz;
  switchuvm(curproc);
  return 0;
}
```

Allocates physical page, updates page table

growproc on XV6

The ***growproc()*** is in ***proc.c***:

```
int
growproc(int n)
{
  uint sz;
  struct proc *curproc = myproc();

  sz = curproc->sz;
  if(n > 0){
    if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0)
      return -1;
  } else if(n < 0){
    if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0)
      return -1;
  }
  curproc->sz = sz;
  switchuvm(curproc);
  return 0;
}
```

Allocates physical page, updates page table

Deallocation, updates page table, free physical page

growproc on XV6

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
- How about physical memory?

XV6: Immediately allocate all 100 physical page frames

allocuvm on xv6

The ***allocuvm()*** is in ***vm.c***

Allocate page tables and physical memory to grow process from ***oldsz*** to ***newsz***. Return ***newsz*** if succeed, 0 otherwise

allocuvm on xv6

The ***allocuvm()*** is in ***vm.c***

Allocate page tables and physical memory to grow process from ***oldsz*** to ***newsz***. Return ***newsz*** if succeed, 0 otherwise

```
int
allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
{
  char *mem;
  uint a;

  if(newsz >= KERNBASE)
    return 0;
  if(newsz < oldsz)
    return oldsz;

  a = PGROUNDUP(oldsz);
  for(; a < newsz; a += PGSIZE){
    mem = kalloc();
    if(mem == 0){
      cprintf("allocuvm out of memory\n");
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
      cprintf("allocuvm out of memory (2)\n");
      deallocuvm(pgdir, newsz, oldsz);
      kfree(mem);
      return 0;
    }
  }
  return newsz;
}
```

allocuvm on xv6

The ***allocuvm()*** is in ***vm***

Allocate page tables and physical memory to grow process from ***oldsz*** to ***newsz***. Return ***newsz*** if succeed, otherwise

```
int
growproc(int n)
{
  uint sz;
  struct proc *curproc = myproc();

  sz = curproc->sz;
  if(n > 0){
    if((sz = allocuvm(curproc->pgdir, sz, sz + n)) == 0)
      return -1;
  } else if(n < 0){
    if((sz = deallocuvm(curproc->pgdir, sz, sz + n)) == 0)
      return -1;
  }
  curproc->sz = sz;
  switchuvm(curproc);
  return 0;
}
```

```
int
allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
{
    memset(mem, 0, PGSIZE);
    if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
      cprintf("allocuvm out of memory (2)\n");
      deallocuvm(pgdir, newsz, oldsz);
      kfree(mem);
      return 0;
    }
  }
  return newsz;
}
```

allocuvm on xv6

The ***allocuvm()*** is in ***vm.c***

Allocate page tables and physical memory to grow process from ***oldsz*** to ***newsz***. Return ***newsz*** if succeed, 0 otherwise

mappages(pde_t *pgdir, void *va, unit size, unit pa, int perm)

Creates translations from ***va*** (virtual address) to ***pa*** (physical address) in existing page table ***pgdir.*** Returns 0 if successful, -1 if not.

```
int
allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
{
  char *mem;
  uint a;

  if(newsz >= KERNBASE)
    return 0;
  if(newsz < oldsz)
    return oldsz;

  a = PGROUNDUP(oldsz);
  for(; a < newsz; a += PGSIZE){
    mem = kalloc();
    if(mem == 0){
      cprintf("allocuvm out of memory\n");
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
      cprintf("allocuvm out of memory (2)\n");
      deallocuvm(pgdir, newsz, oldsz);
      kfree(mem);
      return 0;
    }
  }
  return newsz;
}
```

Round the address to the higher multiple of PGSIZE

Fill a block of memory with a particular value

Virtual address

allocuvm on xv6

The ***allocuvm()*** is in ***vm.c***

Allocate page tables and physical memory to grow process from ***oldsz*** to ***newsz***. Return ***newsz*** if succeed, 0 otherwise

mappages(pde_t *pgdir, void *va, unit size, unit pa, int perm)

Creates translations from ***va*** (virtual address) to ***pa*** (physical address) in existing page table ***pgdir.*** Returns 0 if successful, -1 if not.

```
int
allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
{
  char *mem;
  uint a;

  if(newsz >= KERNBASE)
    return 0;
  if(newsz < oldsz)
    return oldsz;

  a = PGROUNDUP(oldsz);
  for(; a < newsz; a += PGSIZE){
    mem = kalloc();
    if(mem == 0){
      cprintf("allocuvm out of memory\n");
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
      cprintf("allocuvm out of memory (2)\n");
      deallocuvm(pgdir, newsz, oldsz);
      kfree(mem);
      return 0;
    }
  }
  return newsz;
}
```

Round the address to the higher multiple of PGSIZE

Fill a block of memory with a particular value

Default page size

allocuvm on xv6

The ***allocuvm()*** is in ***vm.c***

Allocate page tables and physical memory to grow process from ***oldsz*** to ***newsz***. Return ***newsz*** if succeed, 0 otherwise

mappages(pde_t *pgdir, void *va, unit size, unit pa, int perm)

Creates translations from ***va*** (virtual address) to ***pa*** (physical address) in existing page table ***pgdir.*** Returns 0 if successful, -1 if not.

```
int
allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
{
  char *mem;
  uint a;

  if(newsz >= KERNBASE)
    return 0;
  if(newsz < oldsz)
    return oldsz;

  a = PGROUNDUP(oldsz);
  for(; a < newsz; a += PGSIZE){
    mem = kalloc();
    if(mem == 0){
      cprintf("allocuvm out of memory\n");
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
      cprintf("allocuvm out of memory (2)\n");
      deallocuvm(pgdir, newsz, oldsz);
      kfree(mem);
      return 0;
    }
  }
  return newsz;
}
```

Round the address to the higher multiple of PGSIZE

Fill a block of memory with a particular value

Translating virtual address to physical address

allocuvm on xv6

The ***allocuvm()*** is in ***vm.c***

Allocate page tables and physical memory to grow process from ***oldsz*** to ***newsz***. Return ***newsz*** if succeed, 0 otherwise

mappages(pde_t *pgdir, void *va, unit size, unit pa, int perm)

Creates translations from ***va*** (virtual address) to ***pa*** (physical address) in existing page table ***pgdir.*** Returns 0 if successful, -1 if not.

```
int
allocuvm(pde_t *pgdir, uint oldsz, uint newsz)
{
  char *mem;
  uint a;

  if(newsz >= KERNBASE)
    return 0;
  if(newsz < oldsz)
    return oldsz;

  a = PGROUNDUP(oldsz);
  for(; a < newsz; a += PGSIZE){
    mem = kalloc();
    if(mem == 0){
      cprintf("allocuvm out of memory\n");
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0){
      cprintf("allocuvm out of memory (2)\n");
      deallocuvm(pgdir, newsz, oldsz);
      kfree(mem);
      return 0;
    }
  }
  return newsz;
}
```

Round the address to the higher multiple of PGSIZE

Fill a block of memory with a particular value

Flags the page as writeable and to be used by programs (otherwise only the kernel can access it).

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?

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);
```

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

XV6: Immediately allocate all 100 physical page frames

Lab 4: **On Demand Allocation**

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?

XV6: Immediately allocate all 100 physical page frames

Lab 4: **On Demand Allocation**

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?

XV6: Immediately allocate all 100 physical page frames

Lab 4: **On Demand Allocation**

allocate one physical page frame upon the 1st access on that page.

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?

XV6: Immediately allocate all 100 physical page frames

Lab 4: **On Demand Allocation**

allocate one physical page frame upon the 1st access on that page.

allocate one physical page frame when page fault happens.

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

translation lookaside buffer

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**

Lab 4 – Part 2 Lazy Allocation

- In file **trap.c** and method **trap()**

```
void
trap(struct trapframe *tf)
…
  //cases
…
  //PAGEBREAK: 13
  default:
….
```

Lab 4 – Part 2 Lazy Allocation

- In file **trap.c** and method **trap()**

```
void
trap(struct trapframe *tf)
…
case T_PGFLT:
   growproc(4092);
   break;

  //PAGEBREAK: 13
  default:
….
```

Lab 4 – Part 1 Eliminate Allocation from sbrk()

- Just increment the process's size (proc->sz) by n and return the old size.
- Delete the call to growproc()

Comment out

```
int
sys_sbrk(void)
{
  int addr;
  int n;

  if(argint(0, &n) < 0)
    return -1;
  addr = myproc()->sz;
  if(growproc(n) < 0)
    return -1;
  return addr;
}
```

Lab 4 – Part 2 Lazy Allocation

- In file **trap.c** and method **trap()**

```
void
trap(struct trapframe *tf)
…
case T_PGFLT:
  // add just one frame. Exactly where the user requests adjusted by page start address.
  break;

 //PAGEBREAK: 13
 default:
….
```

Lab 4 – Part 2 Lazy Allocation

- In file **trap.c** and method **trap()**

```
void
trap(struct trapframe *tf)
…
case T_PGFLT:
  // add just one frame. Exactly where the user requests adjusted by page start address.
  PGROUNDDOWN(rcr2());
  // alloc memory, clean and map it in the process page table.
  break;

 //PAGEBREAK: 13
 default:
….
```

Lab 4 – Part 2 Lazy Allocation

- Hint: find the virtual address that caused the page fault
  - In trap.c, find the cprintf arguments for “pid XX XX: trap XX err X on cpu X eip …”
- Hint: you can check whether a fault is a page fault by
  - By checking if ***tf->trapno*** is equal to **T_PGFLT**
- Hint: reference the logic of allocuvm() in vm.c
- Hint: use PGROUNDDOWN(va) to round the faulting virtual address down to a page
- Hint: break or return in order to avoid the cprintf and the proc->killed = 1
- Hint: call int mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm)
  - Delete the “static” in the declaration of mappages() in vm.c
  - Declare mappages() in trap.c

Lab 4

If all goes well, your lazy allocation code should result in “*echo hi*” working.

CS 1550

Week 10 – Lab 4

Teaching Assistant

Henrique Potter