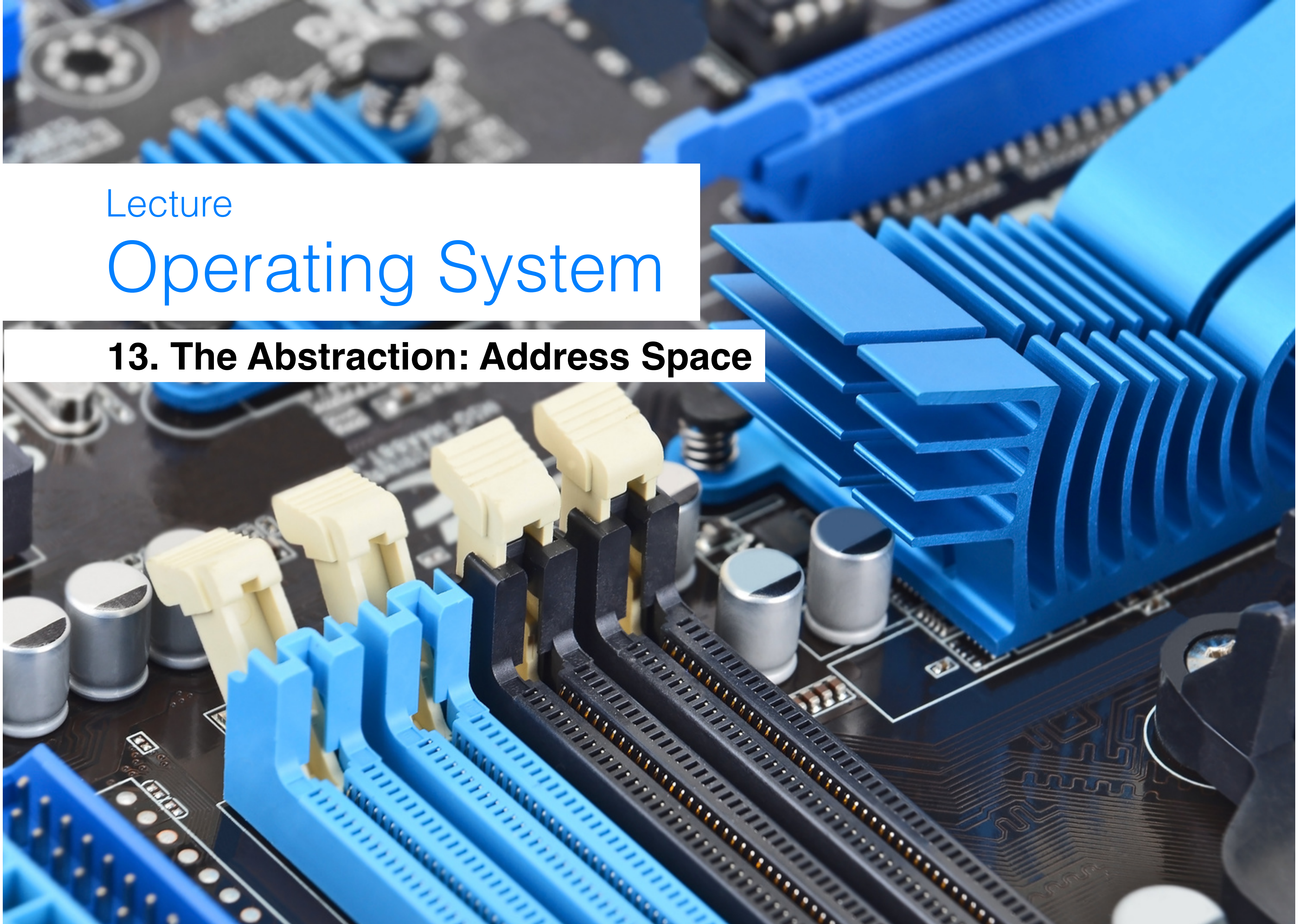


Lecture

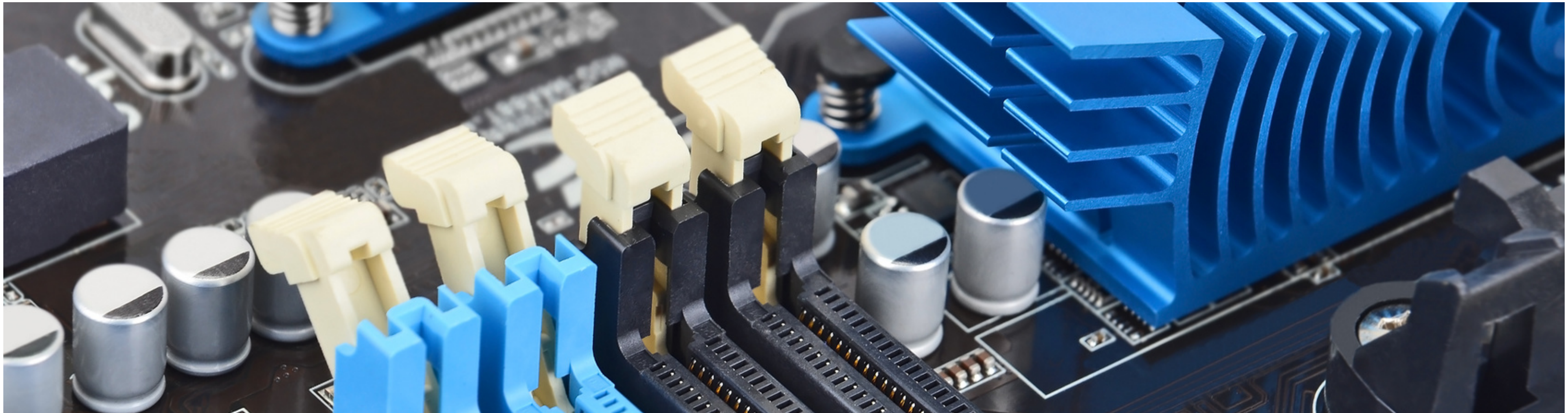
Operating System

13. The Abstraction: Address Space

Professor Dr. Michael Mächtel

13. Address Space

1. **The Abstraction**
2. **Physical Memory**
3. **Address Space**
4. **Virtual Adress**

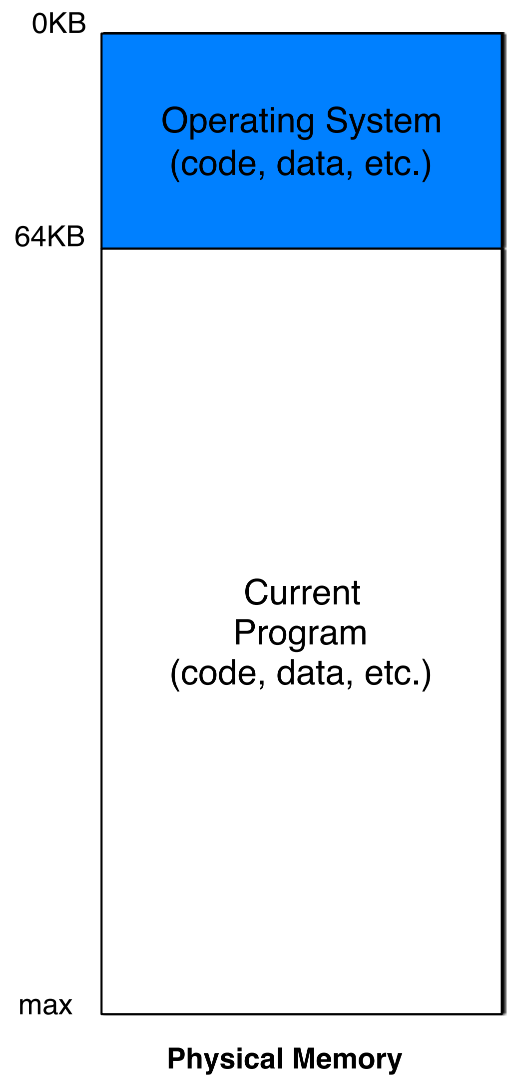


Memory Virtualization

- **What is memory virtualization?**
  - OS virtualizes its physical memory.
  - OS provides an illusion memory space per each process.
  - It seems to be seen like each process uses the whole memory.
- **Benefit of Memory Virtualization:**
  - Ease of use in programming
  - Memory efficiency in terms of times and space
  - The guarantee of isolation for processes as well as OS
  - Protection from errant accesses of other processes

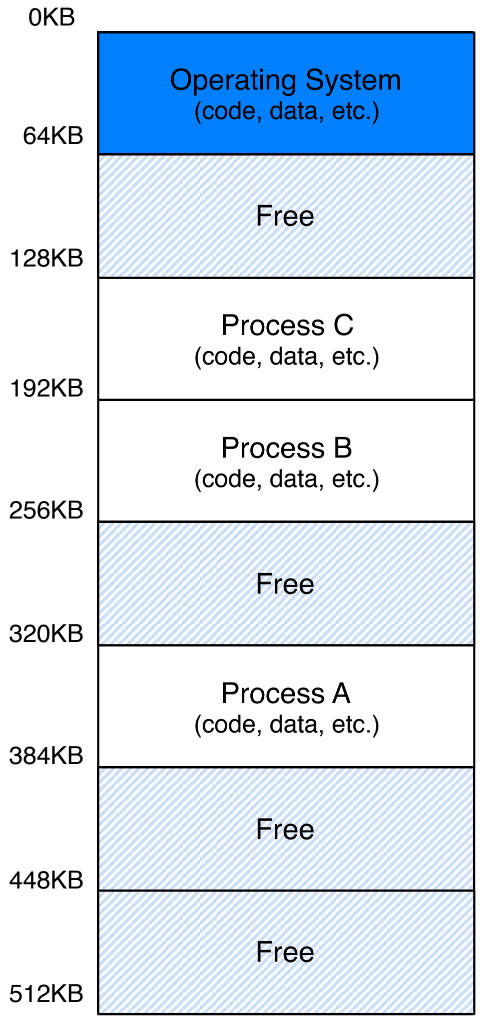
OS in The Early System

- Load only one process in memory.
    - Poor utilization and efficiency



Multiprogramming and Time Sharing

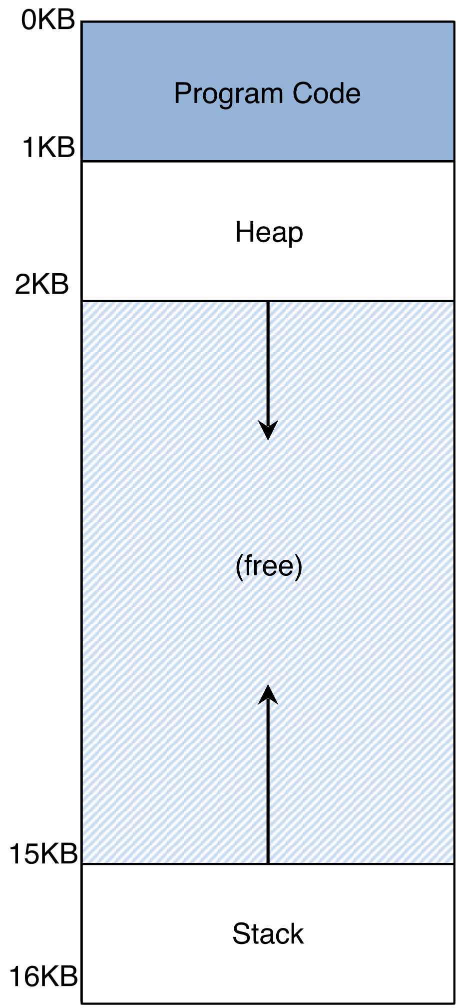
- **Load multiple processes** in memory.
  - Execute one for a short while.
  - Switch processes between them in memory.
  - Increase utilization and efficiency.
- Cause an important **protection issue**.
  - Errant memory accesses from other processes



Physical Memory

Address Space per Process

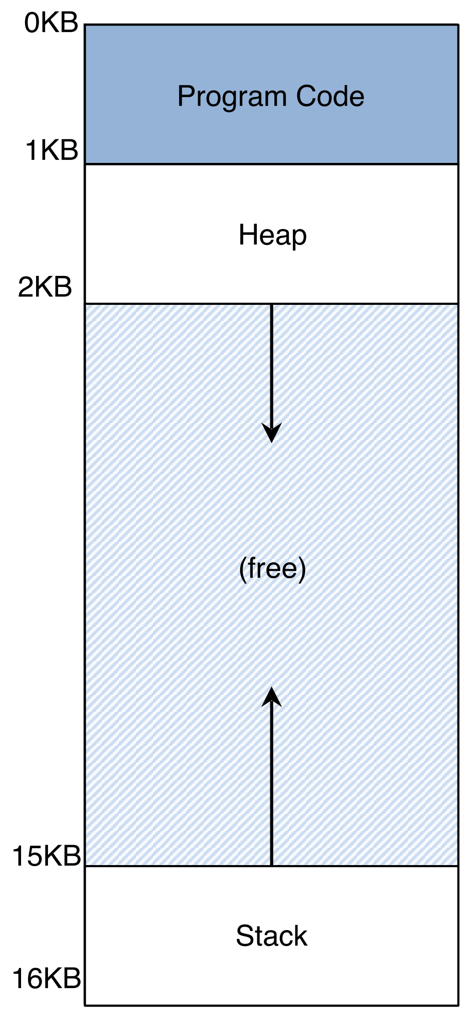
- OS creates an **abstraction** of physical memory.
  - The address space contains all about a running process.
  - That is consist of program code, heap, stack and etc.
- Address space has static and dynamic components
  - *Static*: Code and some global variables
  - *Dynamic*: Stack and Heap



Address Space

Address Space per Process

- **Code**
  - Where instructions live
- **Heap**
  - Dynamically allocate memory.
    - `malloc` in C language
    - `new` in object-oriented language
- **Stack**
  - Store return addresses or values.
  - Contain local variables arguments to routines.



Address Space

Motivation for Dynamic Memory

- Do **not know** amount of memory needed **at compile time**
- Must be **pessimistic** when allocate **memory statically**
  - Allocate enough for worst possible case; Storage is used inefficiently
- **Recursive** procedures:
  - Do not know how many times procedure will be nested
- **Complex data** structures: lists and trees
  - `struct my_t *p = (struct my_t *)malloc(sizeof(struct my_t));`
- → Two types of dynamic allocation:
  - Stack and Heap

Stack Organization

- Definition: Memory is freed in opposite order from allocation
- Simple and efficient implementation:
  - Pointer separates allocated and freed space
    - Allocate: Increment pointer
    - Free: Decrement pointer
- No fragmentation
- OS uses stack for procedure call frames (local variables and parameters)

```
alloc(A);
alloc(B);
alloc(C);
free(C);
alloc(D);
free(D);
free(B);
free(A);
```

```
main() {
 int A = 0;
 foo(A);
 printf("A: %d\n", A);
}

void foo(int Z) {
 int A = 2;
 Z = 5;
 printf("A : % d Z : % d\n", A, Z);
}
```

Heap Organization

- Definition: Allocate from any random location:
  - `malloc()`, `new()`
- Heap memory consists of allocated areas and free areas (holes)
  - Order of allocation and free is unpredictable

Heap Organization (Cont.)

- Advantage
  - Works for all data structures
- Disadvantages
  - Allocation can be slow
  - End up with small chunks of free space - fragmentation
  - Where to allocate 12 bytes? 16 bytes? 24 bytes?
- What is OS's role in managing heap?
  - OS gives big chunk of free memory to process; library manages individual allocations

Quiz: Match that Address Location

```
int x;

main() {
 int y;
 int *z = malloc(sizeof(int));
}
```

Address	Location
x	Static data -> Code
main	Code
y	Stack
z	Stack
*z	Heap

Virtual Address

- Every address in a running program is virtual.
- OS translates the virtual address to physical address

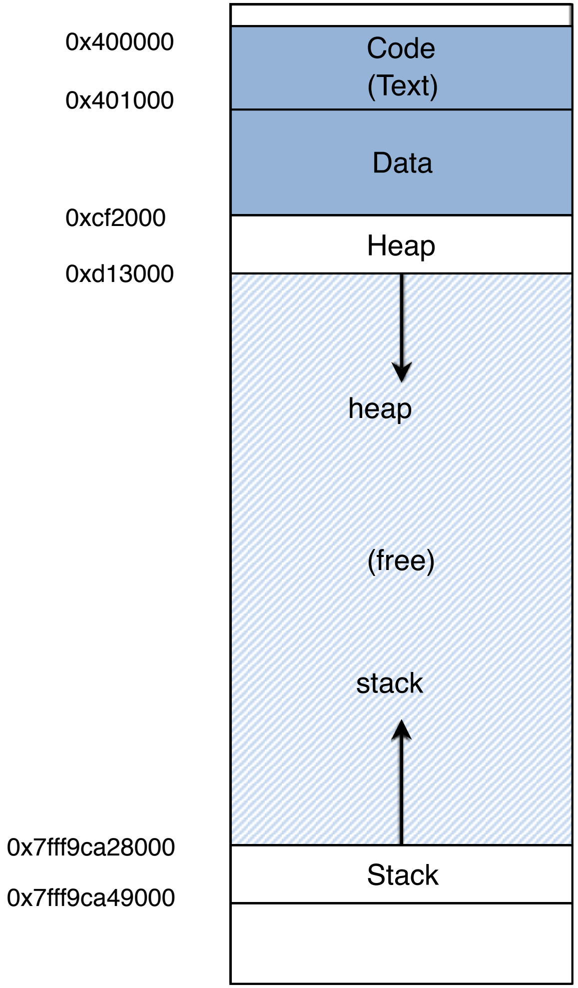
adresses.c

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){

    printf("location of code  : %p\n", (void *) main);
    printf("location of heap  : %p\n", (void *) malloc(1));
    int x = 3;
    printf("location of stack : %p\n", (void *) &x);

    return x;
}
```

Virtual Address

- The output in 64-bit Linux machine

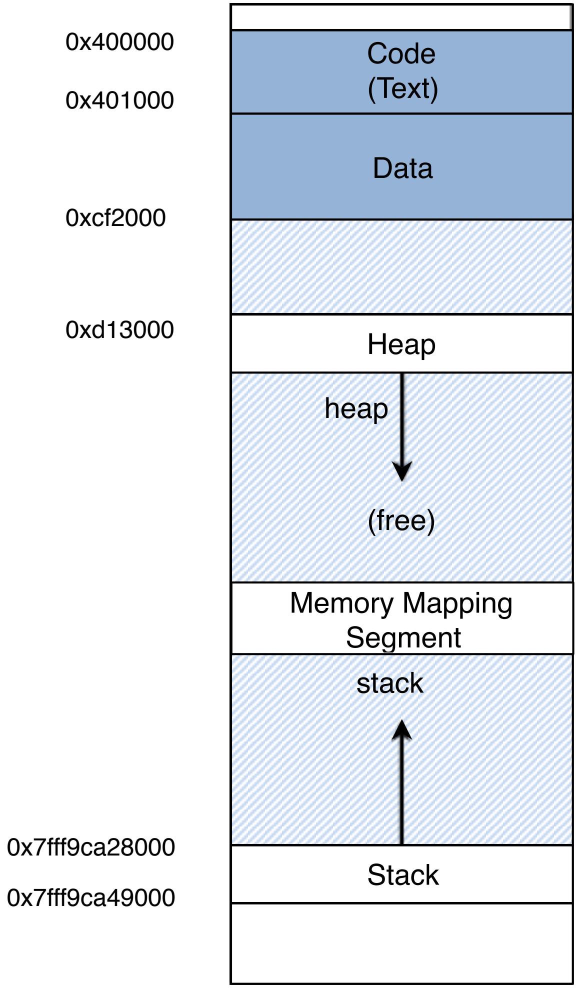
```
prompt> ./addresses
location of code  : 0x40057d
location of heap  : 0xcf2010
location of stack : 0x7fff9ca45fcc
```



Address Space

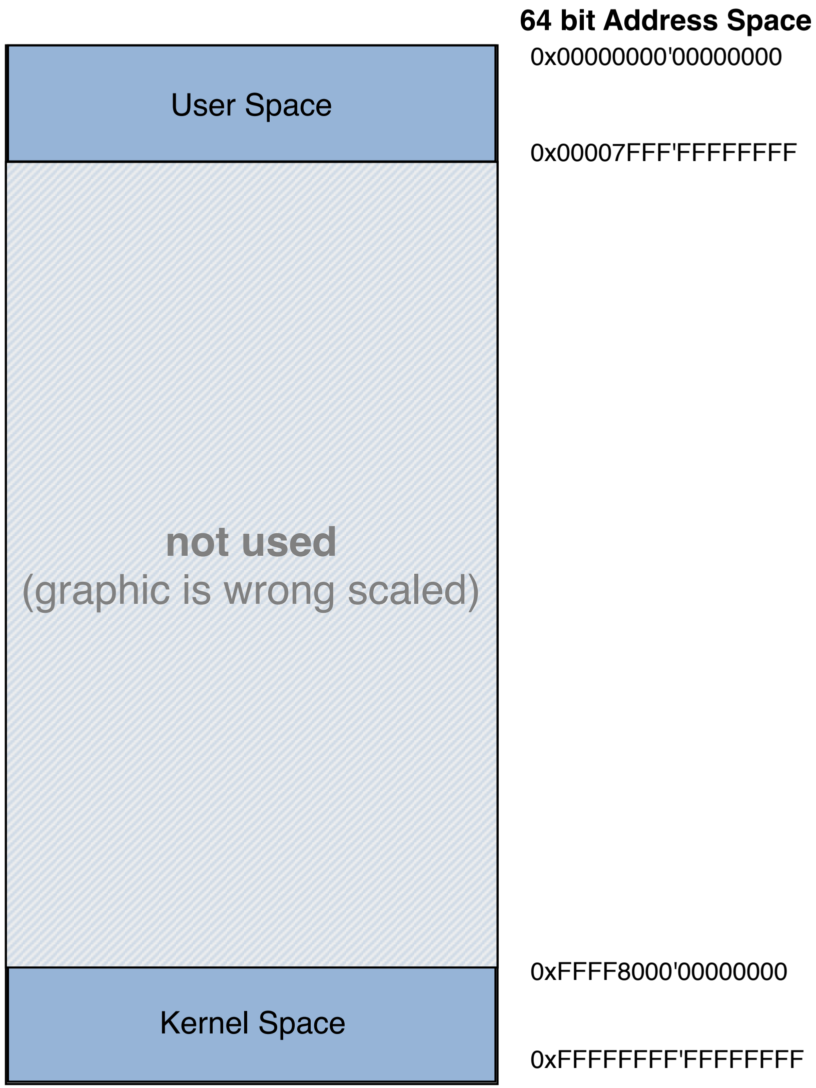
Virtual Address (Cont.)

- Code + Data
  - Usually starts at 0x400000
    - Rust uses –fPIE (position independent executable) linker flag: executable is mapped to random address
- Heap & Stack: continuous block
- Memory Mapping Segment
  - Several blocks
  - Not continuous!
- ASLR: Random offsets to increase security



Address Space

User Space and Kernel Space



- In theory 64 bit
  - 16 ExaBytes
- Today: 48 bit
  - Canonical form
  - 256 TB
- Separated between
  - User Space
  - Kernel Space

Thanks

Questions?