CMSC421: Principles of Operating Systems

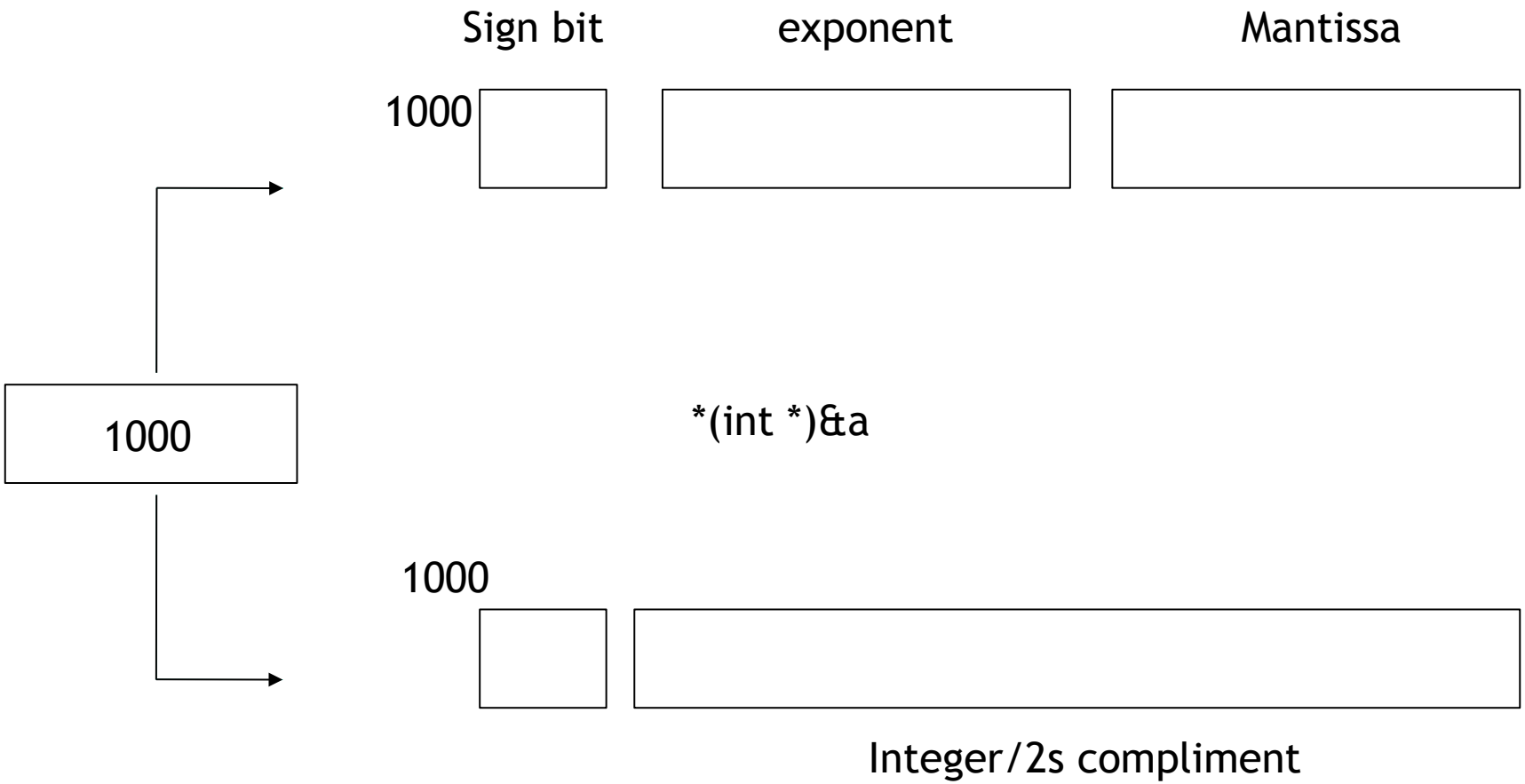
Nilanjan Banerjee

Assistant Professor, University of Maryland
Baltimore County
nilanb@umbc.edu
http://www.csee.umbc.edu/~nilanb/teaching/421/

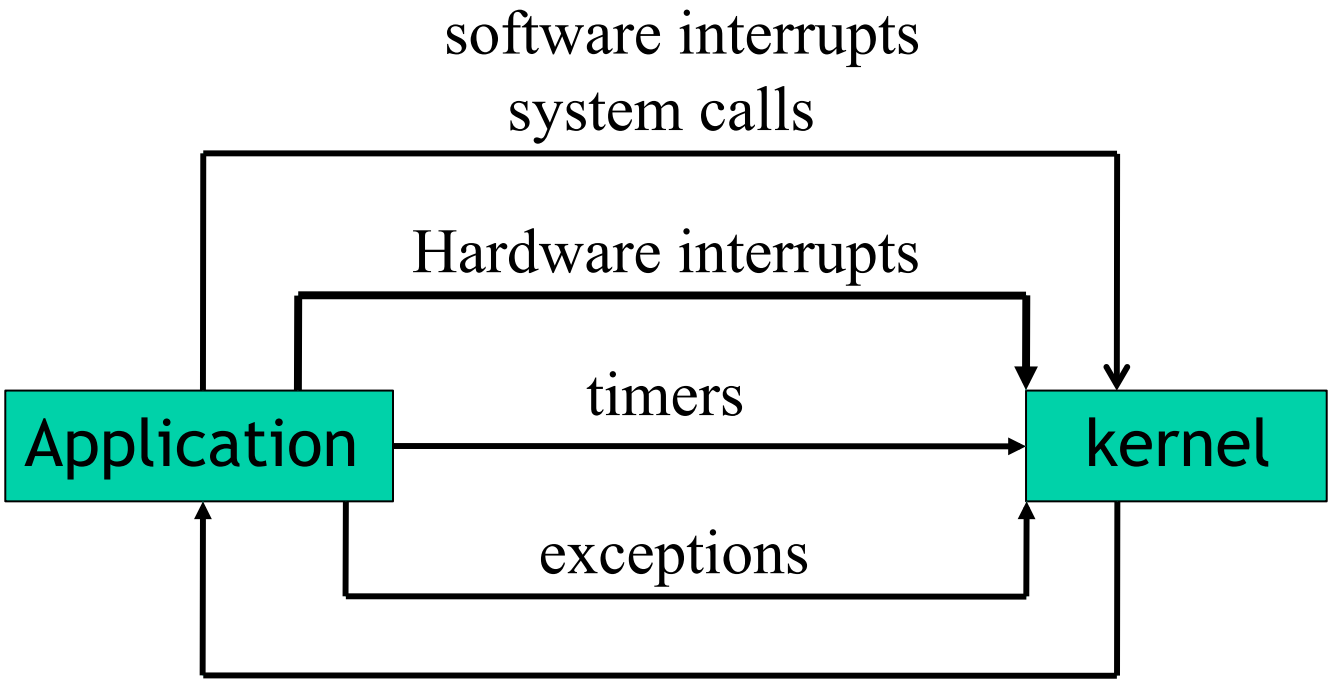
Announcements

- Project 0 and Homework 1 out
- Discussion grades will not be on Blackboard
- Readings from Silberchatz
  - Optional but important

Discussion 1



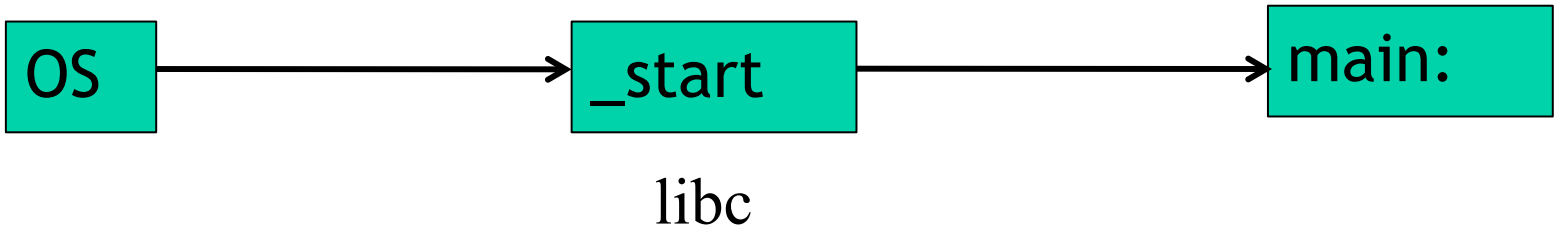
Kernel-userspace interaction

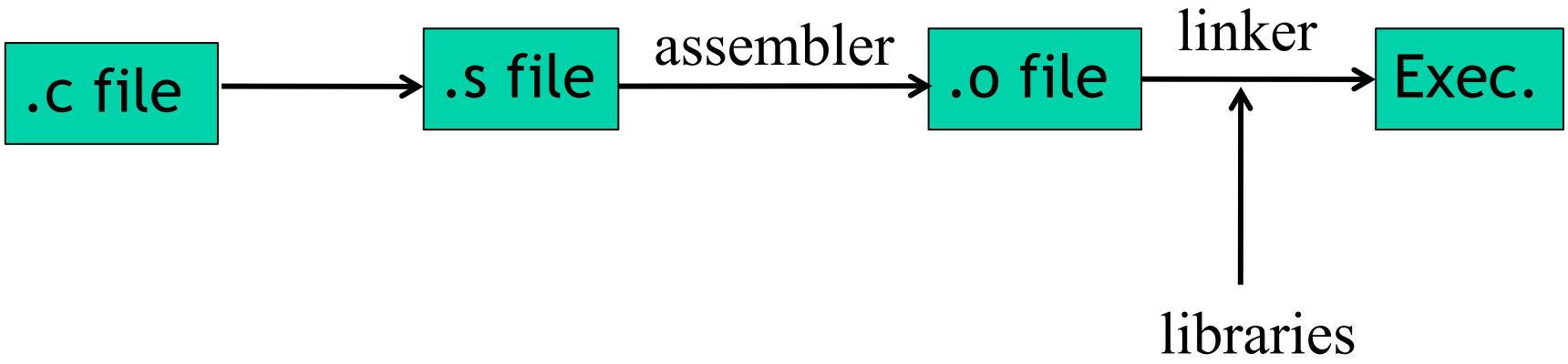


Timers/return from sys call

A closer look at system calls

Lets take an example



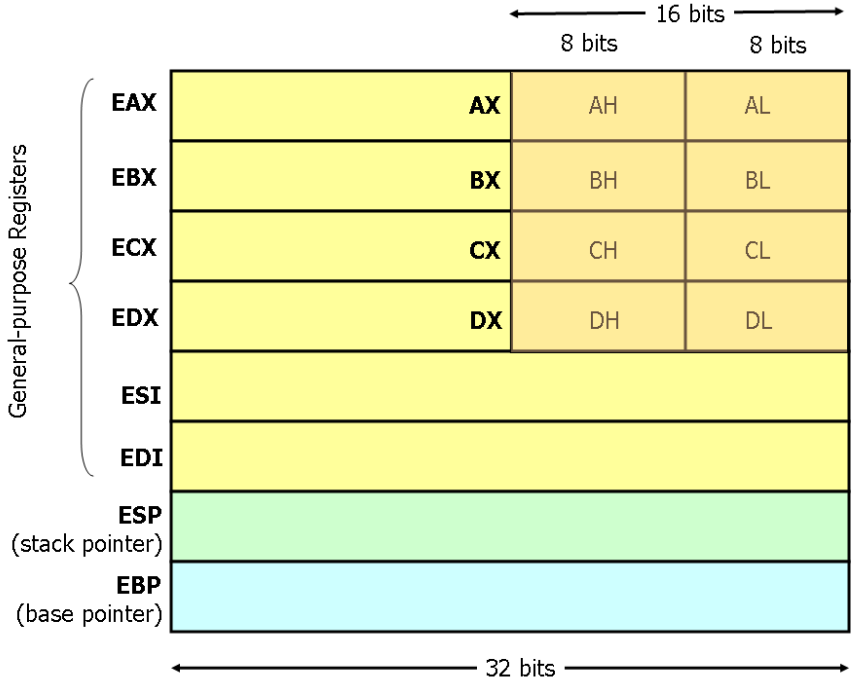


X86 assembly for system calls (older mechanism)

mov $1, %eax

mov $25, %ebx

int $0x80 ← **Execute interrupt # 128**
In the interrupt vector table



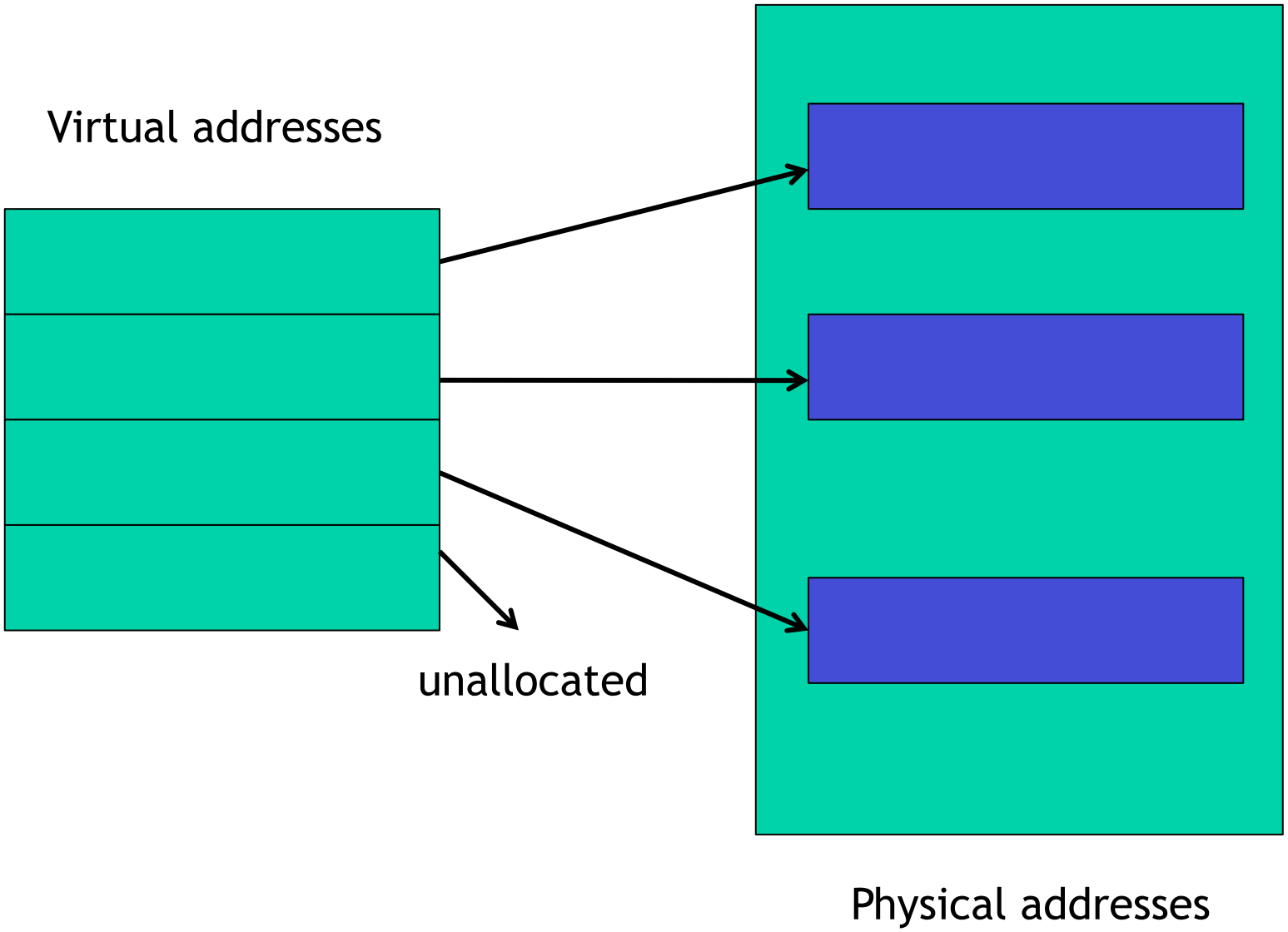
↓

Software interrupt

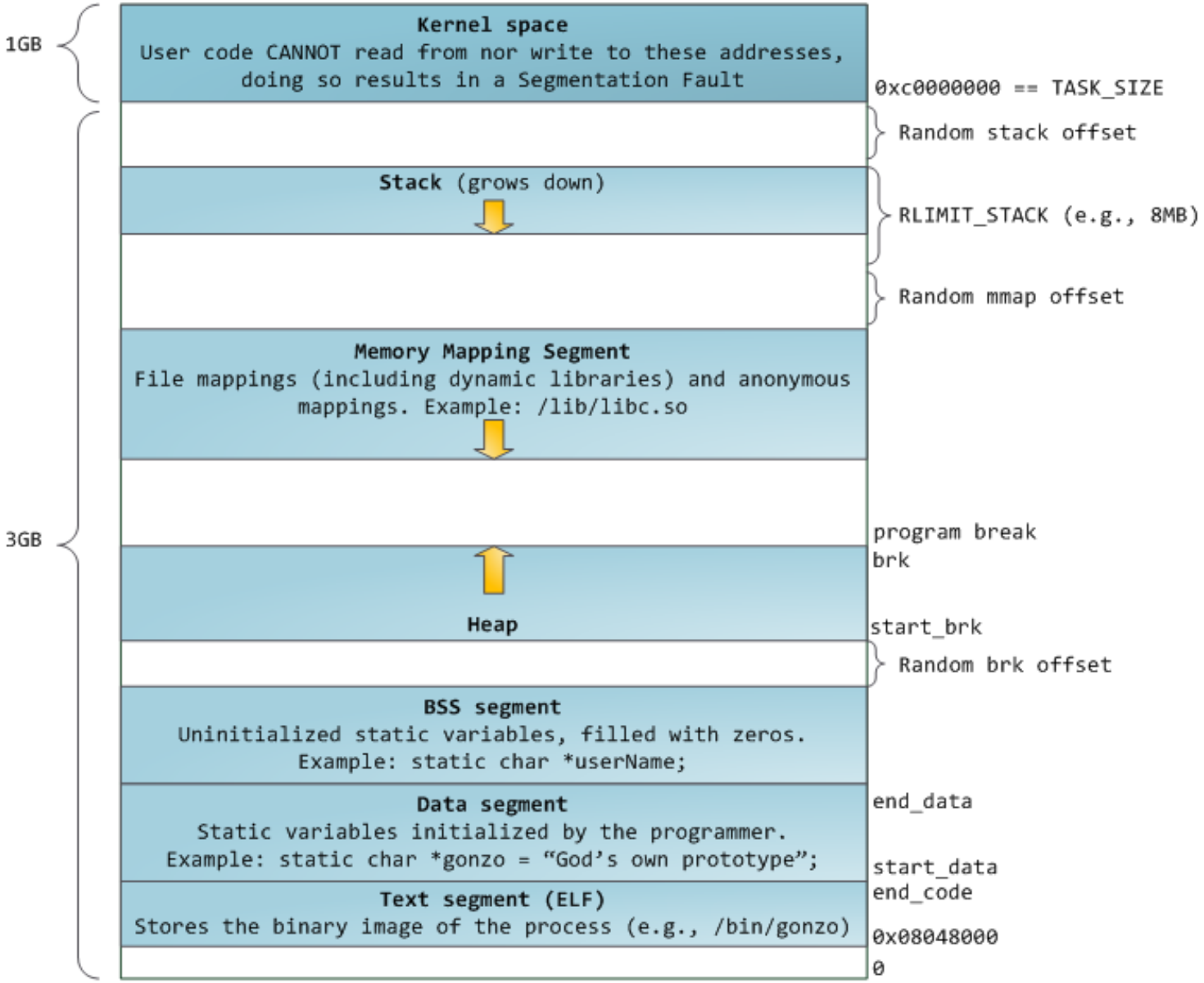
↓

Jump to an address in the kernel
where the syscall table is stored
And execute syscall # stored in %eax
args for syscall in registers
[ebx, ecx, edx, esi, edi]

Primer into virtual memory management

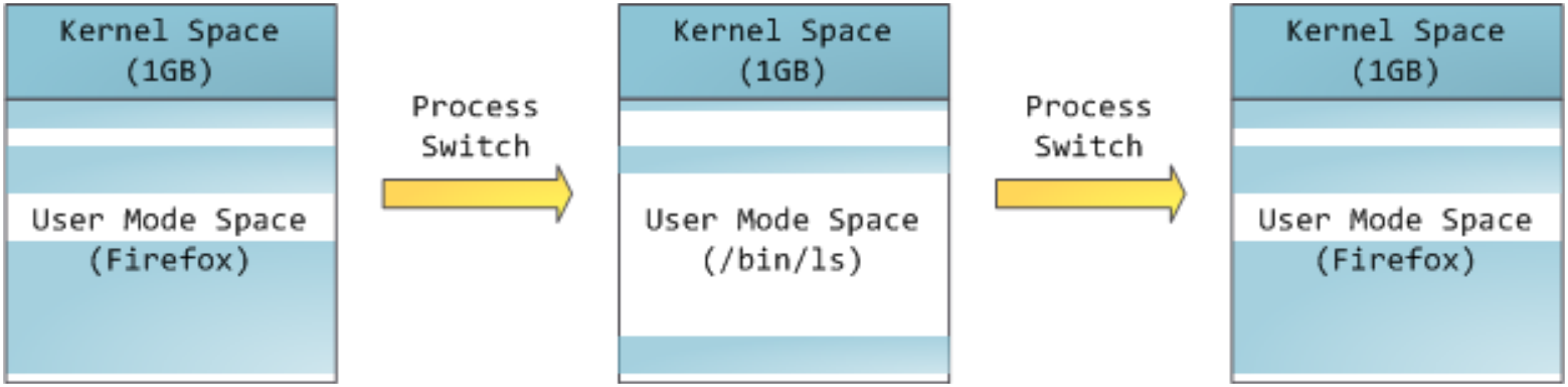


Primer into kernel and user space memory



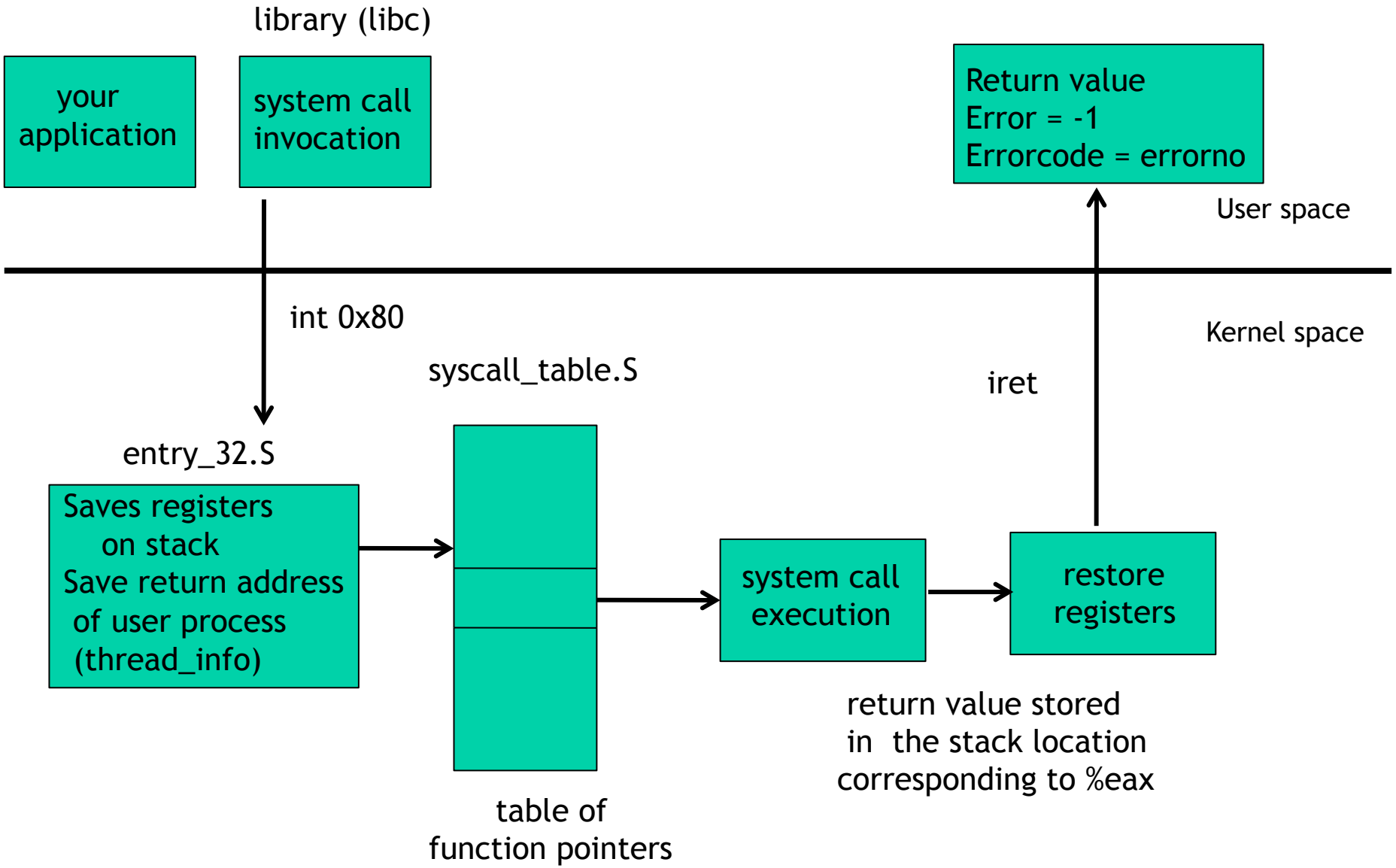
Acknowledgement: http://duarts.org/gustavo/blog/category/internals

Primer into how context switching happens



Acknowledgement: http://duarts.org/gustavo/blog/category/internals

Flow of control during a system call invocation



Kernel dive.

Important kernel files/ data structures for system calls

- implementation file for the sys call
  - kernel/sys.c (most of the system calls are implemented)
  - You can implement a system call anywhere

- include/asm-i386/unistd.h
  - Defines the *number* of a system call
  - Defined the total number of system calls.

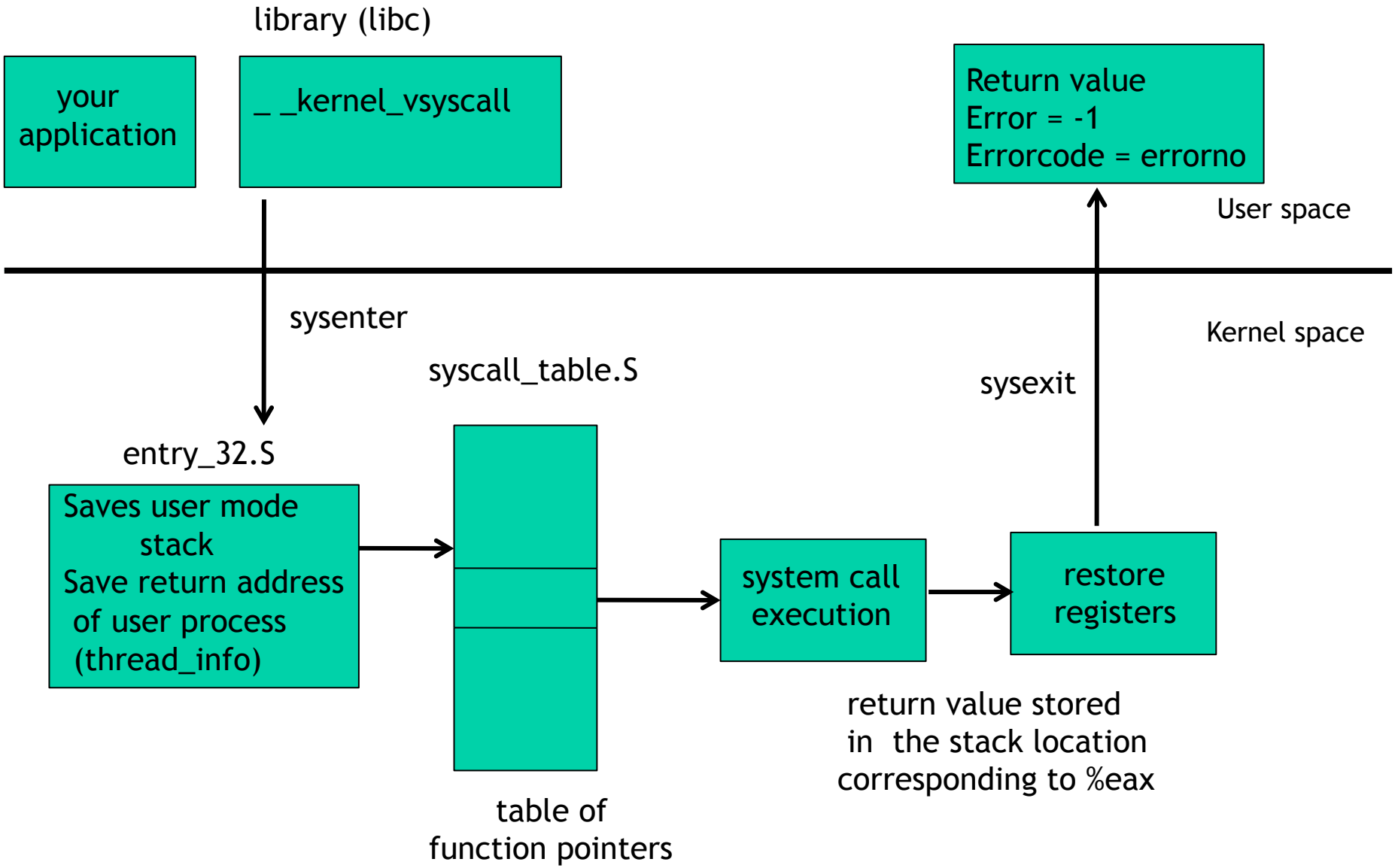
- arch/i386/kernel/syscall_table.S
  - Stores the system call table
  - Stores the function pointers to system call definition

Using sysenter/sysexit in Linux > 2.5

- Sysenter/sysexit is also called “Fast system Call”
  - Available in Pentium II +

- Sysenter is made of three registers
  - SYSENTER_CS_MSR -- selecting segment of the kernel code (figuring out which kernel code to run)
  - SYSENTER_EIP_MSR --- address of the kernel entry
  - SYSENTER_ESP_MSR --- kernel stack pointer

Simplified view of sysenter/sysexit in Linux > 2.5



Lets write a system call in the kernel (sys_strcpy)

int strcpy(char *src, char *dest, int len)

can return values of
size of at most long? Why?

asmlinkage long sys_strcpy(char *src, char *dest, int len)

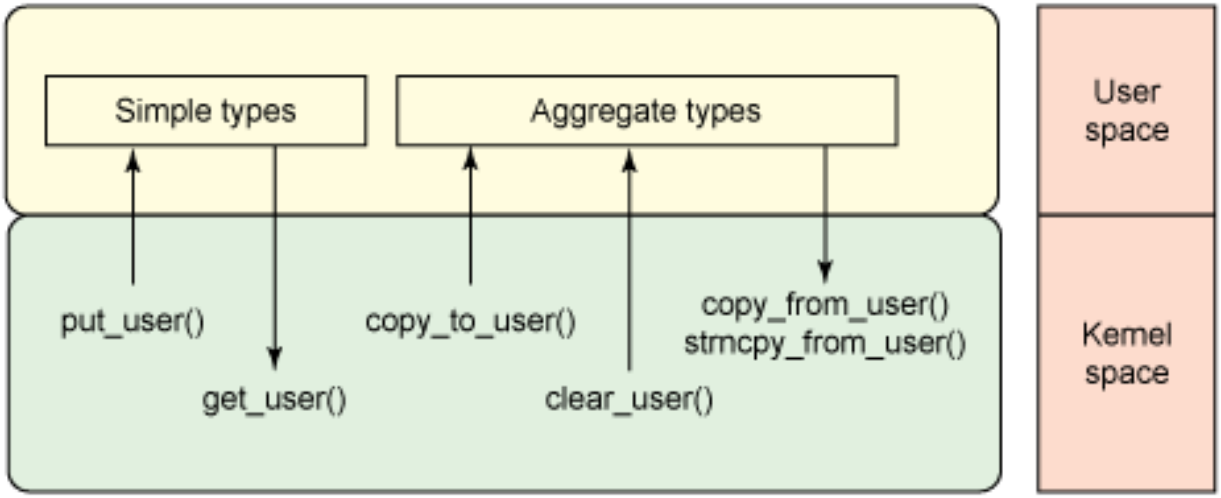
compiler directive
params will be read from stack

Issues to think about when writing system calls

- Moving data between the kernel and user process
  - Concerns: security and protection

- Synchronization and concurrency **(will revisit)**
  - Several (so called) kernel threads might be accessing the same data structure that you want to read/write
  - Simple solution (disable interrupts “cli”)
    - Usually not a good idea
  - Big problem in preemptive CPU (which is almost every CPU) and multi-processor systems
    - CONFIG_SMP or CONFIG_PREEMPT

Useful kernel API functions for bidirectional data movement



- *access_ok (type, addr, size)*: type (VERIFY_READ, VERIFY_WRITE)
- *get_user(x, ptr)* --- read a char or int from user-space
- *put_user(x, ptr)* --- write variable from kernel to user space
- *copy_to_user(to, from, n)* --- copy data from kernel to userspace
- *copy_from_user(to, from, n)* – copy data to kernel from userspace
- *strnlen_user(src, n)* – checks that the length of a buffer is n
- *strcpy_from_user(dest, src, n)* ---copies from kernel to user space

Acknowledgement: http://www.ibm.com/developerworks/linux/library/l-kernel-memory-access/index.html

Next class

- Linux Boot process
  - How the first process gets started

- Process management
  - Process creation and basic IPC: fork(), pipe(), dup2(), wait()
  - Theory on processes

An in-class discussion
(a Microsoft Interview Question)