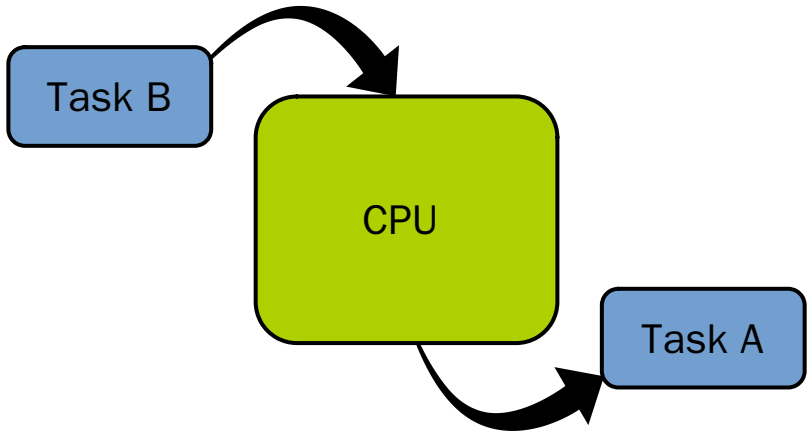
COMP26020 Programming Languages and Paradigms -- Part 1

Case Study: Operating System Kernel

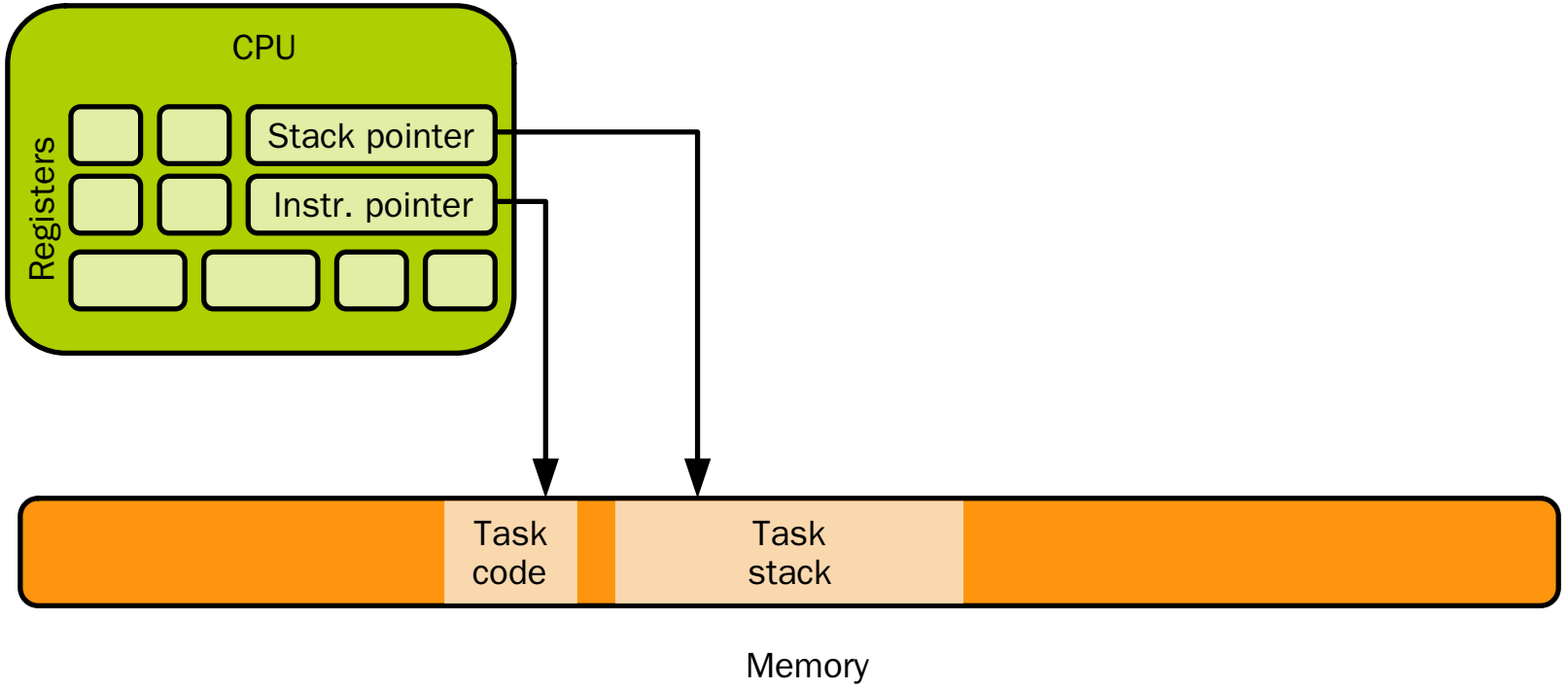
System Call and Context Switch

- C is the default language to write OS kernels
- Let's study the implementation of:
  - **System calls handling**
  - **Context switch** (replacement of a task running on the CPU by another one)

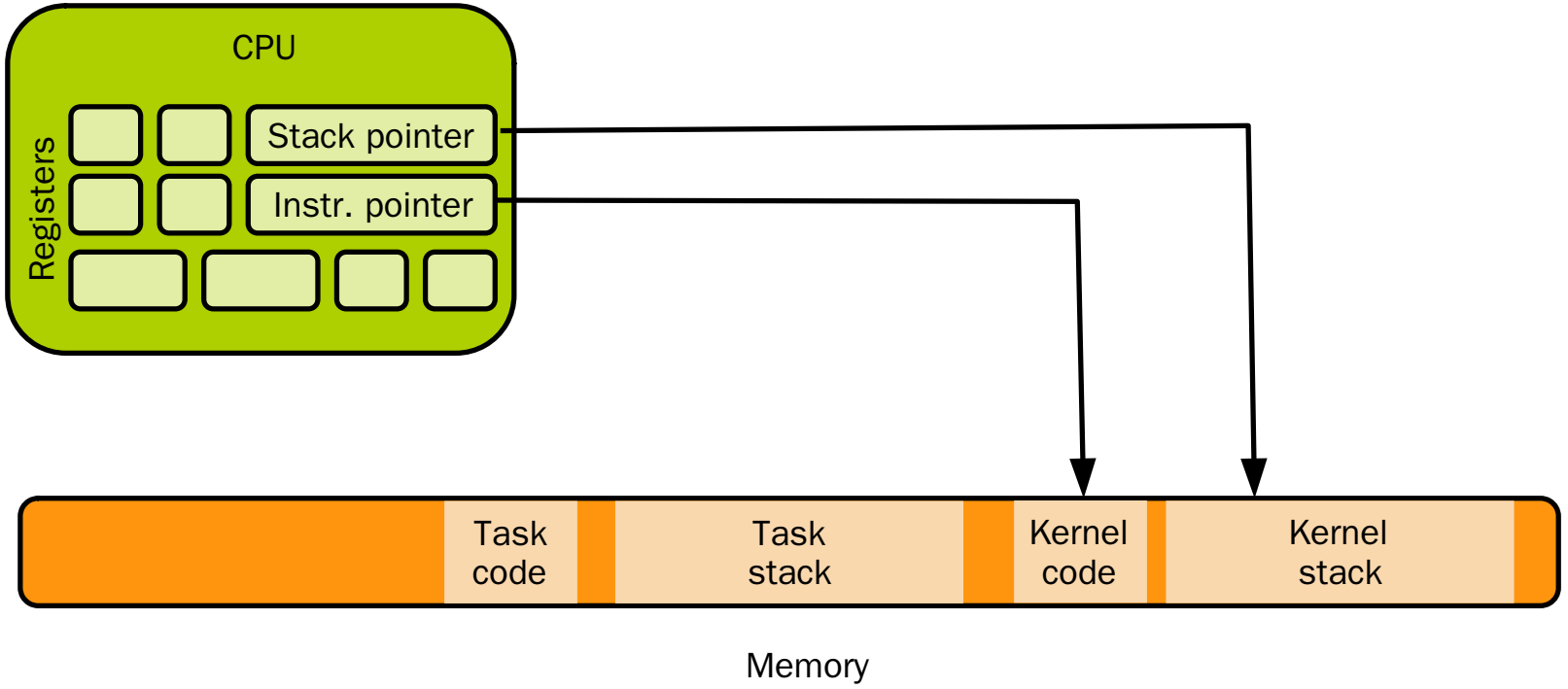


System Calls in a Nutshell

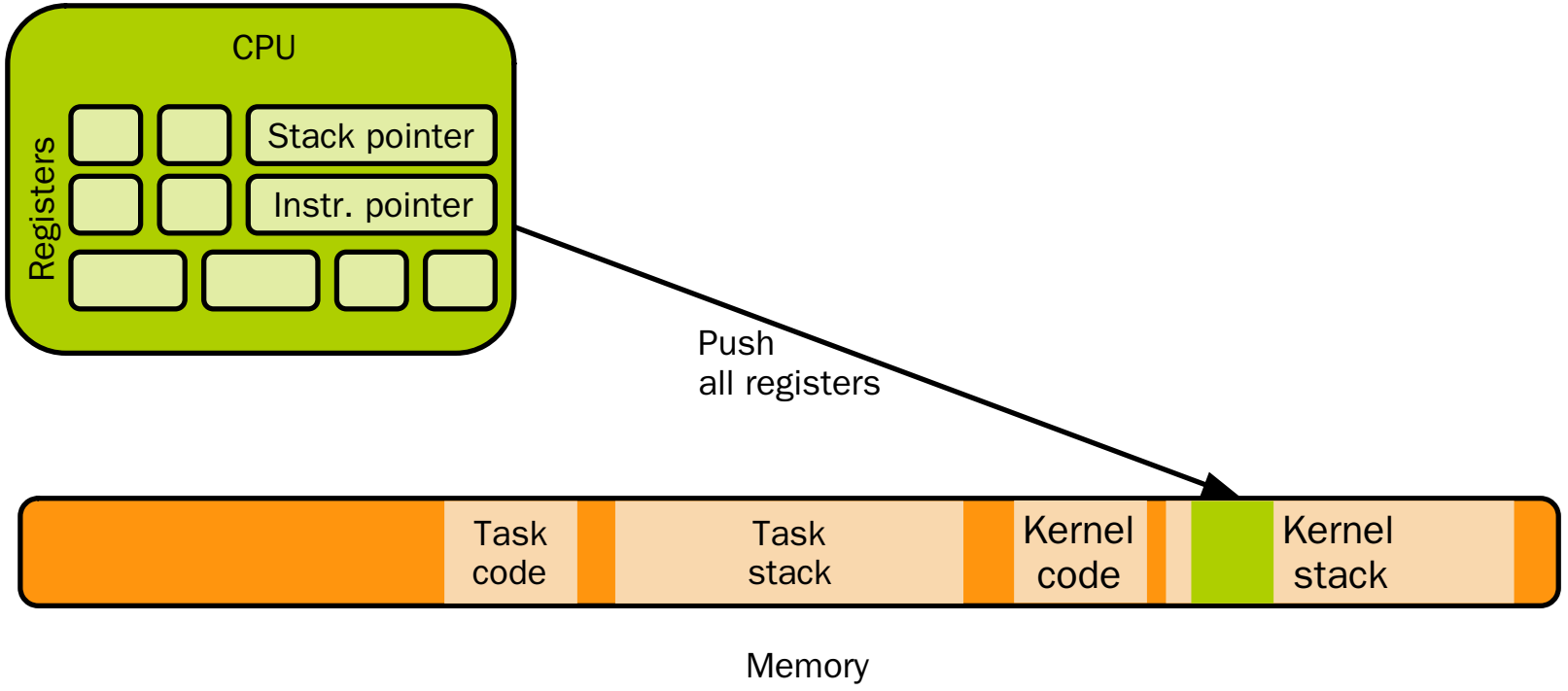
System Call in a Nutshell



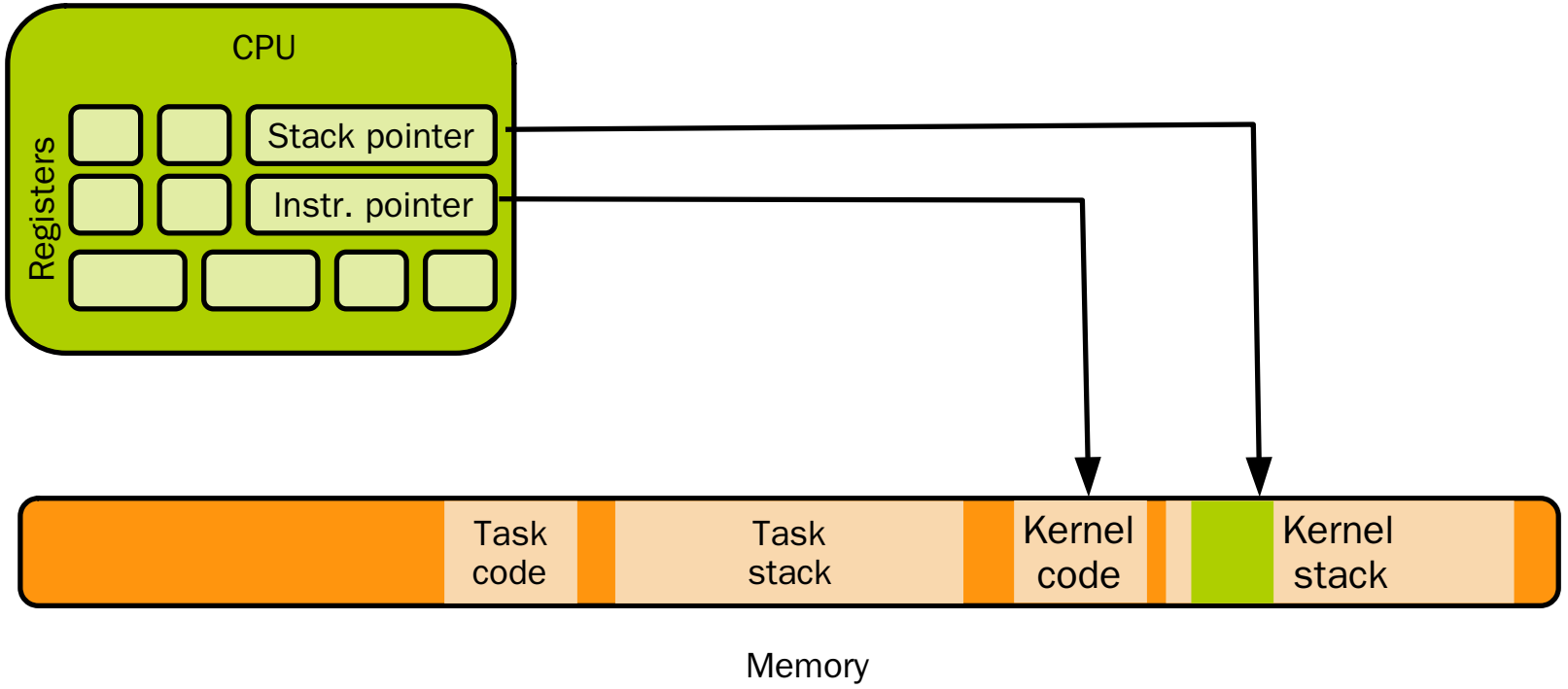
System Call in a Nutshell



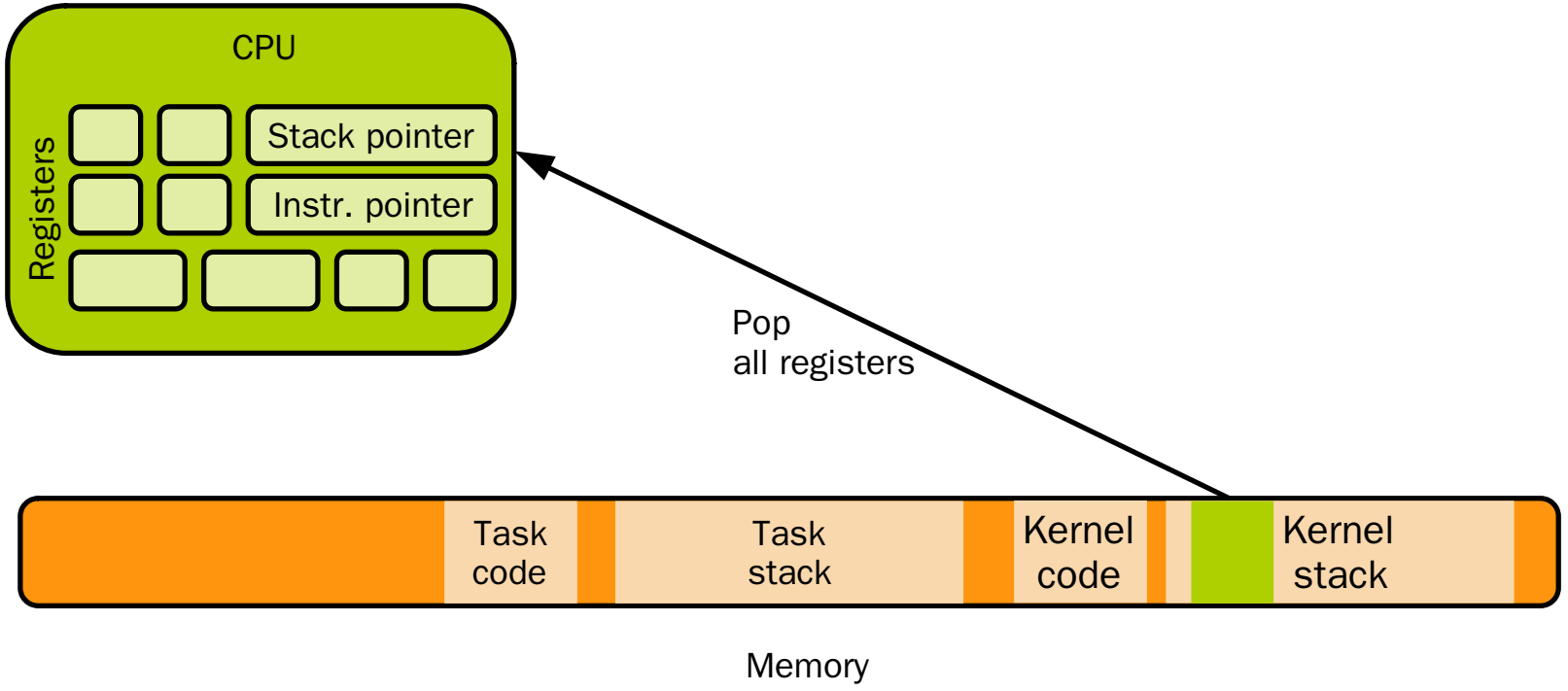
System Call in a Nutshell



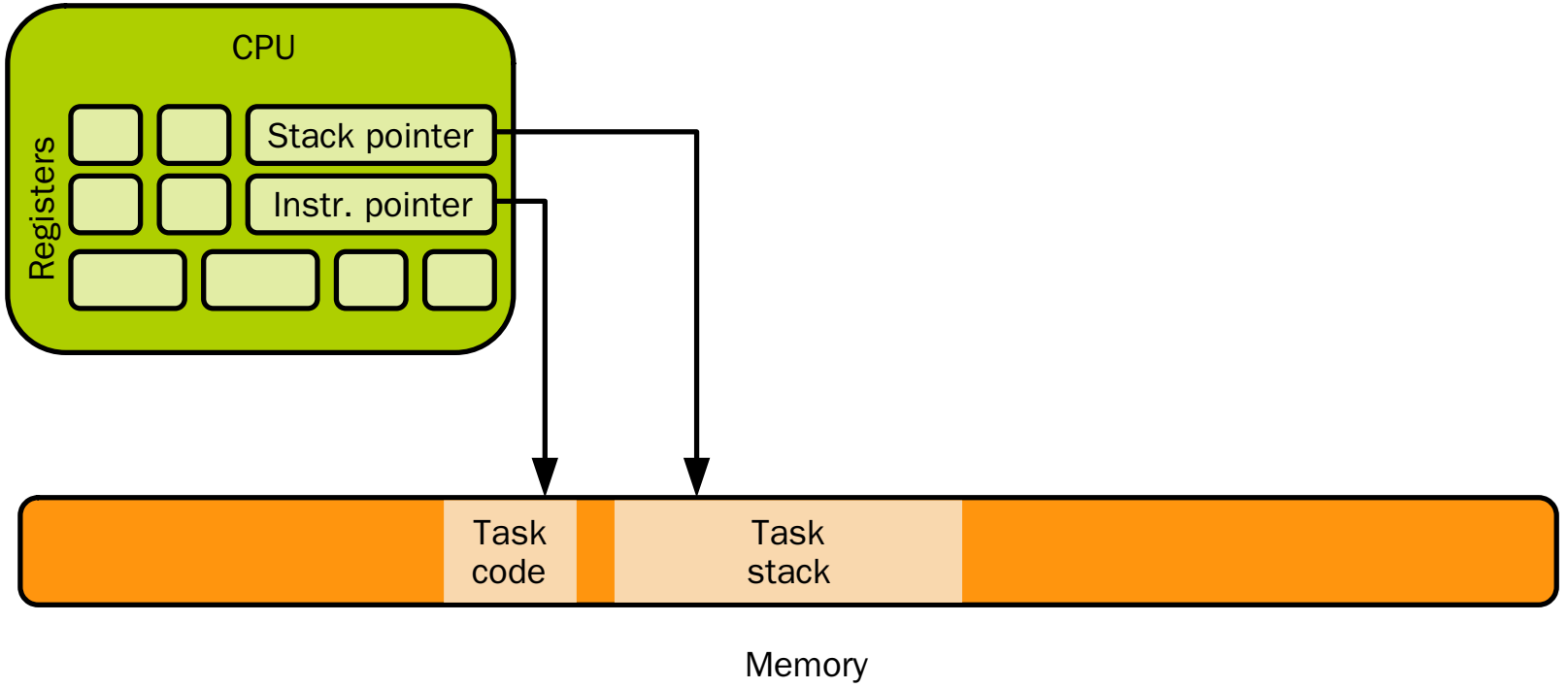
System Call in a Nutshell



System Call in a Nutshell

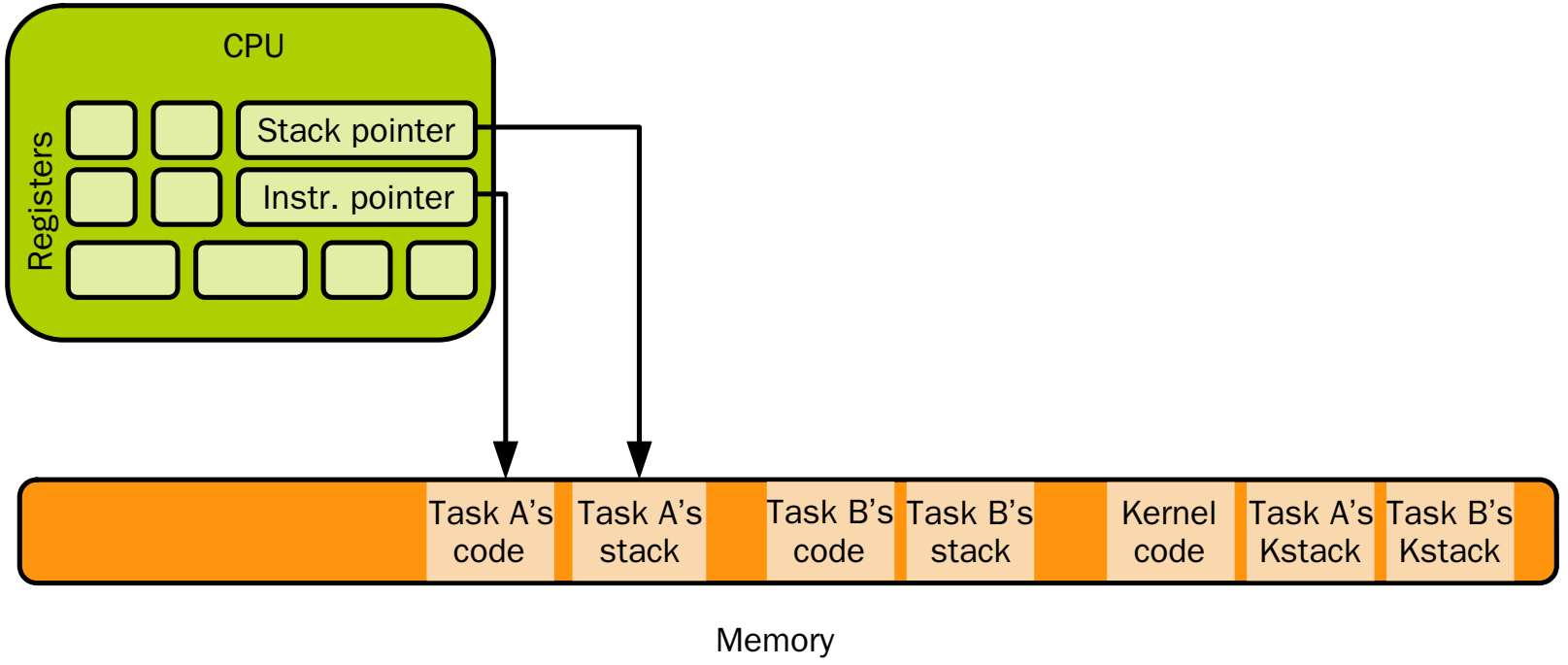


System Call in a Nutshell

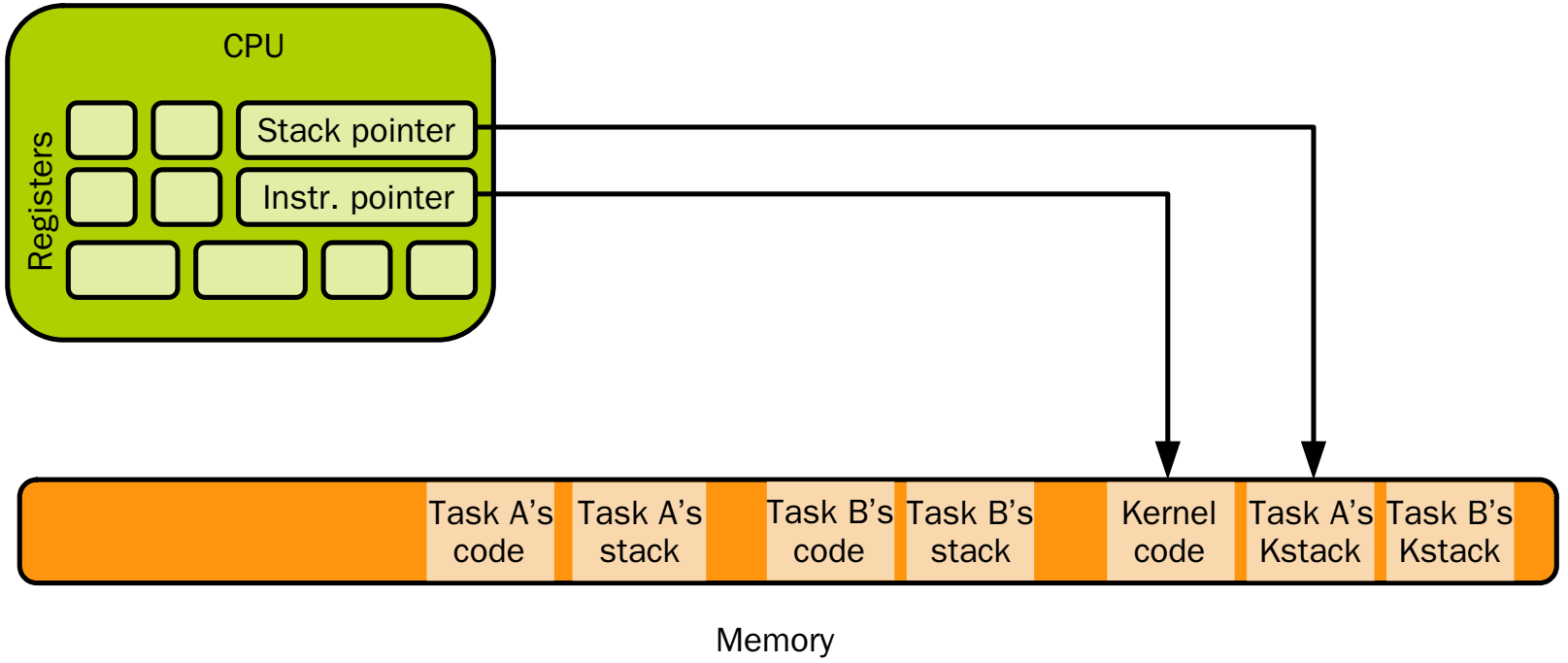


Context Switches in a Nutshell

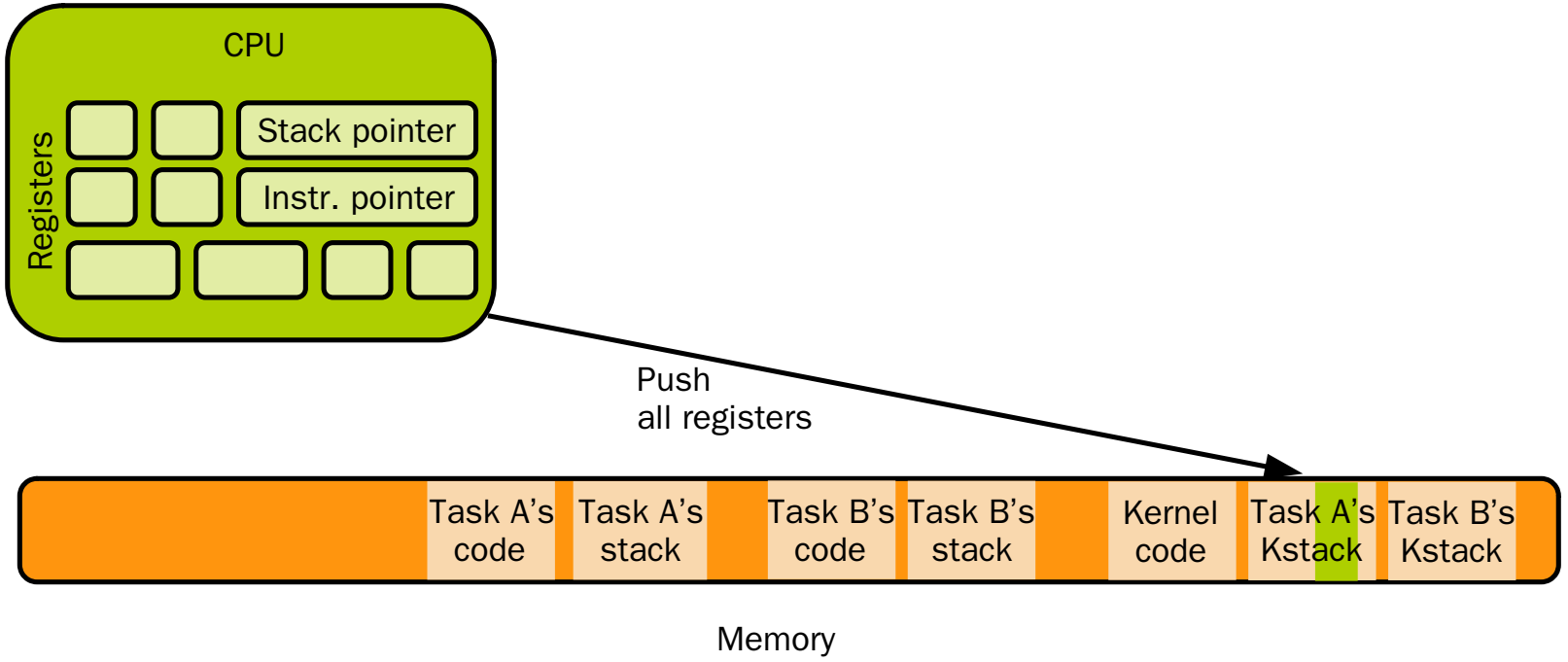
Context Switch in a Nutshell



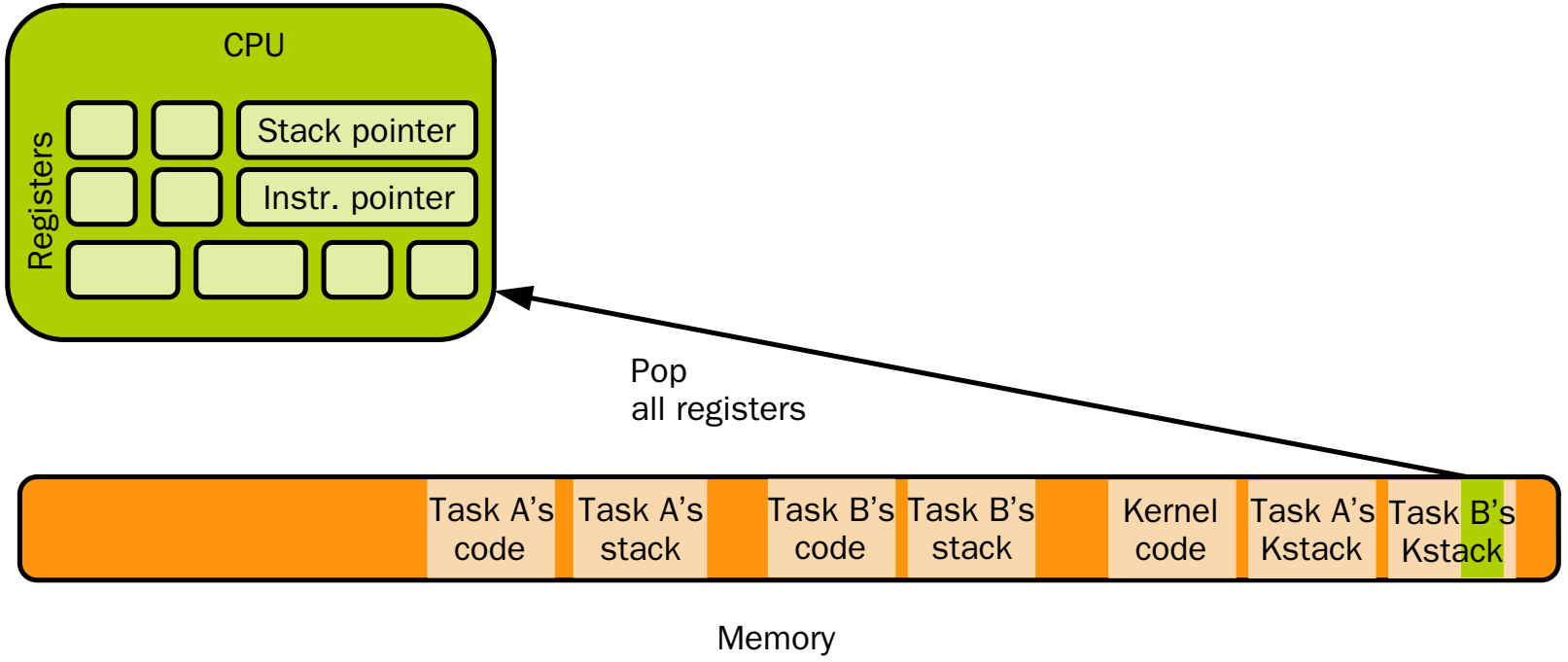
Context Switch in a Nutshell



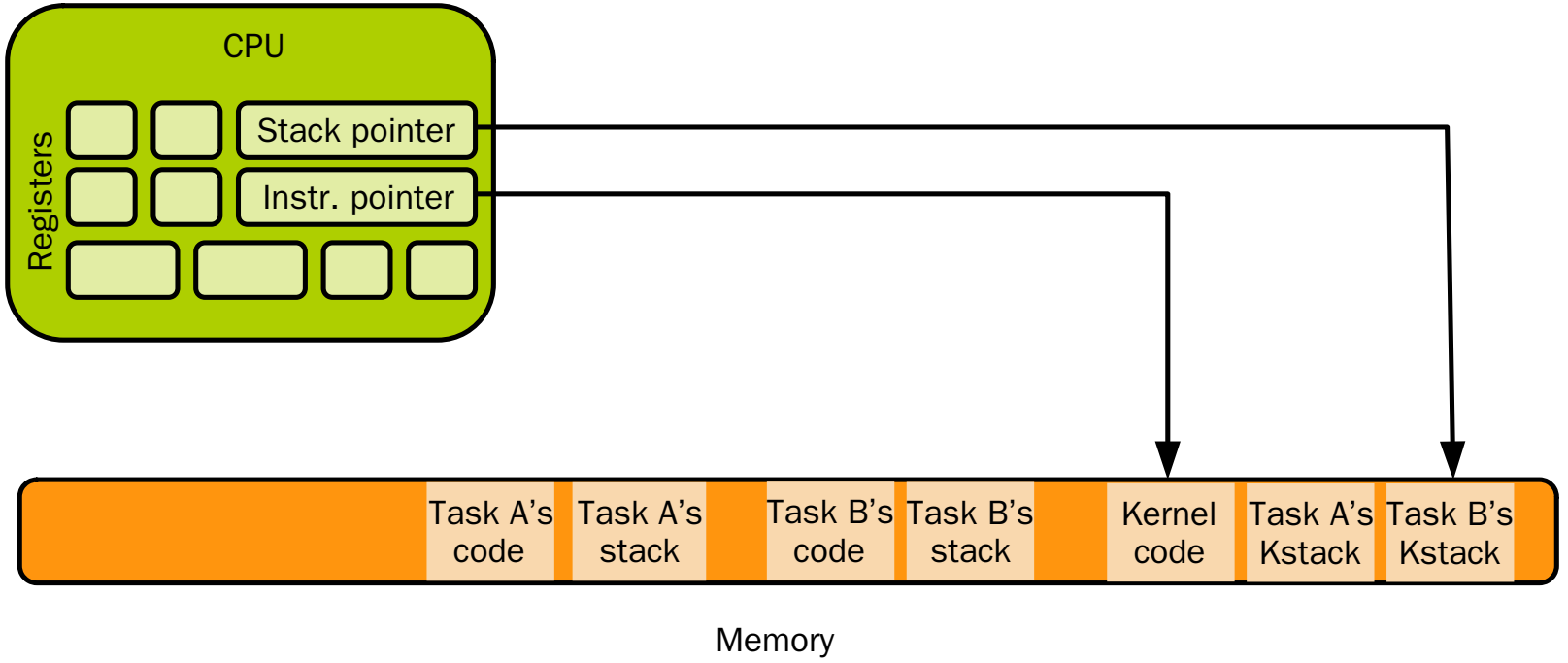
Context Switch in a Nutshell



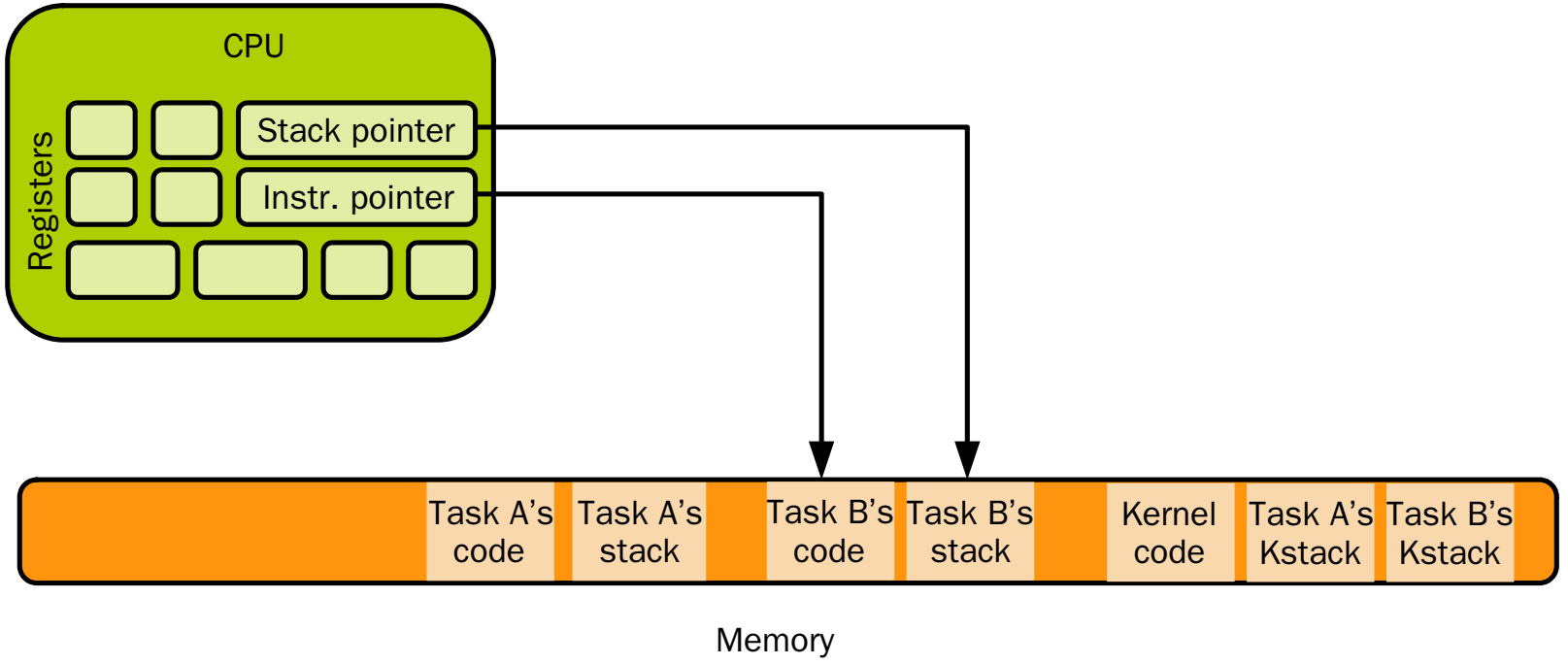
Context Switch in a Nutshell



Context Switch in a Nutshell



Context Switch in a Nutshell



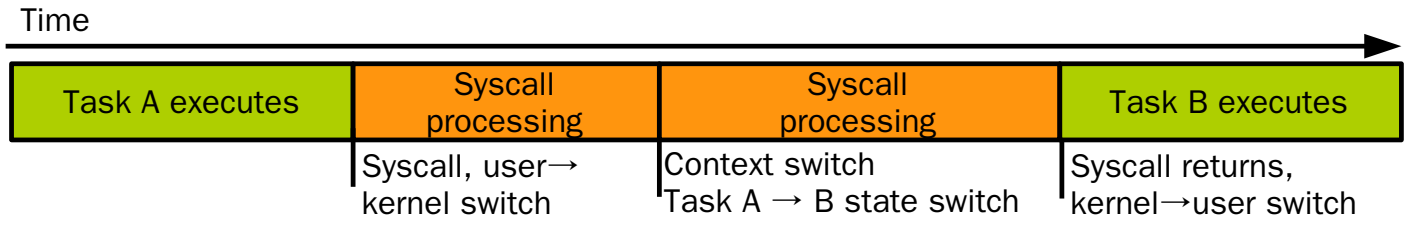
System Call and Context Switch Implementations in a Real OS

HermiTux

- Very small OS for cloud applications
  - Originally based on HermitCore (https://github.com/hermitcore/libhermit)
  - ~10K lines of C code and a bit of assembly, great to study and learn about OSes
  - https://github.com/ssrg-vt/hermitux

Context Switch & System Call in HermiTux

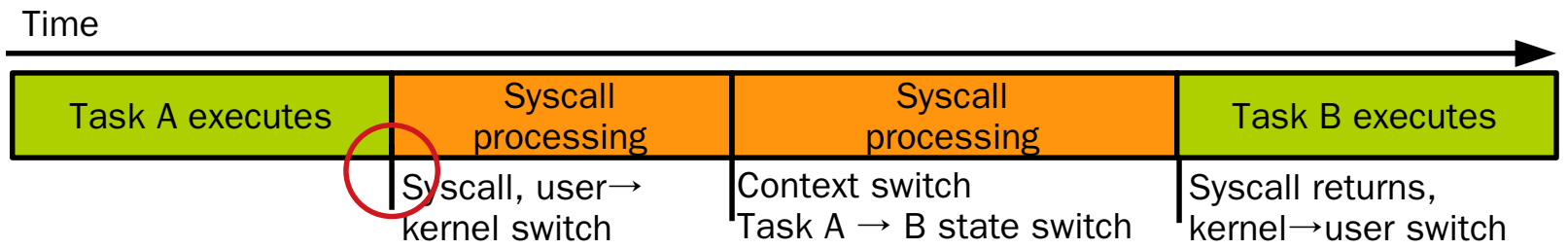
- **Context switch & syscall handling can only be done by the kernel**
- The only way to enter the kernel is through an **interrupt**:
- Interrupt traps to the kernel which saves the state of the interrupted task for resuming properly later
- Example with `sched_yield` syscall (syscall + context switch):



System Call in HermiTux

- Syscall entry point is in [arch/x86/kernel/entry.asm]:

```
isyscall:
    cli             ; disable interrupts
    push rax        ; start pushing the task state on the stack
    push rcx
    ; ...           push many other registers
    mov rdi, rsp ; prepare a datastructure containing register values for use by the kernel
    sti             ; enable interrupts

    call syscall_handler  ; jump to kernel (C) code managing the syscall
```



System Call in HermiTux

`syscall_handler` is in [arch/x86/kernel/isrs.c]:

```
void syscall_handler(struct state *s) {
    switch(s->rax) {
        // ... one "case" for each syscall number
        case 24:                                // sched_yield's number is 24
            s->rax = sys_sched_yield();
            break;
        /* ... */
    }
}
```

System Call in HermiTux

- `sys_sched_yield` (in kernel/syscalls/sched_yield) simply calls `check_scheduling`, in `kernel/tasks.c`:

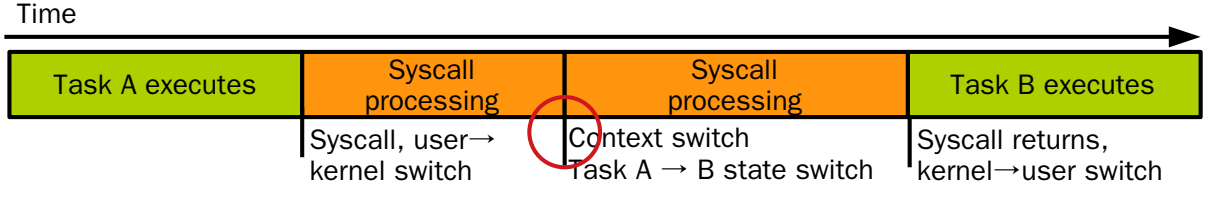
```
void check_scheduling(void)
{
    /* ... */
    uint32_t prio = get_highest_priority();
    task_t* curr_task = per_core(current_task);

    if (prio > curr_task->prio) {
        reschedule();

    /* ... */
}
```

- Finally, `reschedule` calls `switch_context` (in `kernel/tasks.c`)

Context Switch in HermiTux

- `switch_context` is in [arch/x86/kernel/entry.asm]():

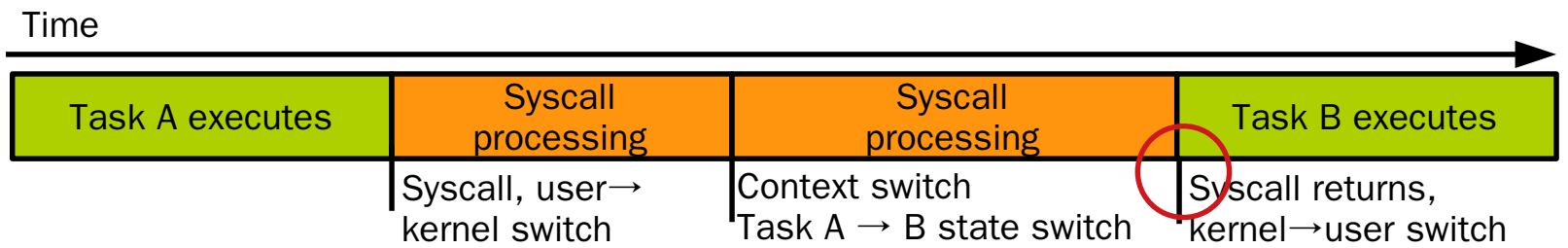
```
switch_context:
    push rax ; ... start pushing the state of the scheduled out task on the stack
    push rcx
    push rdx
    ; ...
    jmp common_switch
    ; ...
common_switch:
    ; ...
    call get_current_stack      ; get new rsp
    mov rsp, rax                ; switch stack to the new task's one
    ; ...
    pop r15                     ; start restoring the scheduled in task state (reverse order)
    pop r14
    ; ...
    add rsp, 16 ; at that point the stack pointer points to the saved instruction pointer
    iretq ; restore instruction pointer from the stack, i.e. jumps there
```



Context Switch in HermiTux

- Executes return path in the kernel until we reach the kernel/user boundary

```
isyscall:
    ; ...
    call syscall_handler
    ; ...
    pop r15   ; start restoring the userland state of the task
    pop r14
    pop r13
    ; ...
    sysret    ; return from syscall handler to userspace
```



Summary

- To summarise, we have seen how syscall handling and context switch are implemented
- It's made with a combination of C and assembly
- Assembly is necessary for low level operations such as saving CPU state or switching tasks
- It integrates very well with C
- In the next video, we'll talk about a different topic: memory safety

---

Feedback form: https://bit.ly/2VD4kfx