ELEC 377 – Operating Systems

Week 1 – Class 3

Lab

- First lab is on Monday. Partner up today!
- Lab handout will be uploaded before tomorrow Morning
- Prelab is due in paper form at the beginning of lab.
- Please send me
  1. NetIDs of you and your partner
  2. NetID of repository to be used
- Stay behind today if you don't have a partner yet, so you can pair together.

Last Class

- Computer System Structure, Controllers
- Interrupts & Traps
- I/O structure and device queues.
- Storage Structure & Caching
- Hardware Protection
- Dual Mode Operation

Command Interpreter

- User Interface
- Accept commands and execute them
- Scripting languages
- Graphical Interface
- Sometimes built into the system
  - ◊ Windows/Macintosh
  - ◊ Old Basic Systems (Pet, Apple 2)
- Sometimes separate process
  - ◊ Unix shell
  - ◊ TSO Interpreter
  - ◊ MS-DOS COMMAND.COM

System Services

- All of the systems we have discussed have to be accessible to user and user programs
- Have to have mechanisms to start programs, request I/O, communicate with other processes and with other computers
- Error detection is required (hardware, user programs)

- Other services needed are:
  - ◊ Resource Allocation
  - ◊ Accounting (time, disk space, pages, etc.)
  - ◊ Protection

System Calls

- Interface between a running process and the operating system
  - ◊ Generally implemented as an assembly language instruction (INT on INTEL, TRAP on PDP-11, ALine instruction on 68000)
  - ◊ Sometimes a function call (Amiga, NCR32000)
  - ◊ Programming languages provide libraries that wrap these instructions for easy access.

- Like most function calls, they take arguments and return a result, it's just the mechanism of the call that is different.

System Calls

- The system call is identified in one of two ways
  - ◊ One of the parameters (Dos)
  - ◊ Different instruction (Macintosh)

- Three general ways are used to pass parameters
  - ◊ Use the registers of the CPU (Amiga, MS-DOS)
  - ◊ Store them in memory and pass the address of the memory in a register (Linux)
  - ◊ Pass them on the stack

- Results are passed back in one of two ways:
  - ◊ On the stack
  - ◊ In a register

Types of System Calls

- Process Management
  - ◊ exit, abort, TSR,load, execute
  - ◊ manage child processes (attributes, wait, trace)
  - ◊ memory allocation, signals
- File System Management
  - ◊ create,open,close,read,write,etc.
- Device Management (e.g. serial speed)
  - ◊ request, release, read,write,attributes
- Information Maintenance
  - ◊ time, date, file and device attributes
- Communication
  - ◊ open,close,send,receive, status

Communication Model

- Interprocess Communication
  - ◊ Command interpreter -> child process
  - ◊ other processes need to communicate (Mac publish and subscribe)
- Message Model
  - ◊ Operating system provides message passing facility.
  - ◊ In some cases, works over network (sockets)
- Shared Memory Model
  - ◊ both processes have access to the same memory
  - ◊ small segments or all of processes memory

Systems Program

- System calls provide capability, but user must be able to invoke them.
- Users view of system defined by system programs, not system calls. No system call to copy a file, but there is a system program.
- Programs must be written that the user can invoke that make the system calls
  - ◊ Command interpreter
  - ◊ File System Management (copy, cp, mkdir, rmdir)
  - ◊ Info (date, time, disk space, list directory)
  - ◊ Editors
  - ◊ compilers, project support
  - ◊ communication (web browsers, ftp clients)

System Structure

- Simple Structure
  - ◊ Limited by hardware
  - ◊ MS-DOS - interfaces and functionality not well separated
  - ◊ Unix is two separate parts
  - – kernel system call interface (some modularity)
  - – systems programs
- Layered (OS/2)
  - ◊ Operating system divided into layers
  - ◊ Each layer only uses lower layers (no sibling calls)
- Microkernel (Mach)
  - ◊ Minimal kernel (no consensus on what minimal is)
  - ◊ Move as much as possible into system programs

Virtualization

- Multiprocessing creates the illusion that there is more than one CPU
  - ◊ What if the OS provided an interface that looked exactly like a computer system (virtual CPU, devices, etc.)
  - ◊ IBM VM Operating system (each can run it's own operating system such as CMS or Linux)
  - ◊ Requires more hardware support
  - ◊ Available on PCs (vmware/xen/kvm)
  - ◊ Limits sharing of resources
  - ◊ Perfect for OS research and development
  - ◊ Emulation (Sheepshaver/BOCHS)

Virtual Machines

- Portability
  - ◊ virtual machine provides same interface on all hardware
- Security
  - ◊ sandbox
- Like an OS
  - ◊ class loader (loads bytecode)
  - ◊ threads (sort of like virtual cpus)
- Performance
  - ◊ Just in time compilation
- Android
  - ◊ Linux Kernel
  - ◊ Dalvik virtual machine

Dalvik virtual machine

- Google's Android Operating System
  - Runs apps
  - Written in Java and compiled in bytecode
    -> Converted from JVM to .class -> .dex
- Open source
- Register-Based
- Performs optimizations to make it suited to embedded domains.
- Just-in-time compiler (stores the programs in memory as byte code, then compiles it to machine code JIT)

Virtual Machines (Java)

- Portability
  - ◊ virtual machine provides same interface on all hardware
- Security
  - ◊ sandbox
- Like an OS
  - ◊ class loader
  - ◊ threads
- Performance
  - ◊ Just in time compilation

Design and Implementation

- Mechanisms and Policies
  - ◊ Mechanisms are how
  - – Use timer to protect CPU
  - ◊ Policy is what will be done
  - – timer value (max time process can have CPU)
  - ◊ more flexibility
- Implementation
  - ◊ language
  - ◊ efficiency
  - ◊ portability (Linux runs on many architectures)
  - ◊ monitoring capability
  - ◊ sysgen (customization of operating system)

What is a process?

- An operating system handles a variety of programs in a variety of ways
  - ◊ Batch systems handle jobs
  - ◊ Timesharing systems calls them user programs or tasks.
- Job and process are used almost interchangeably.

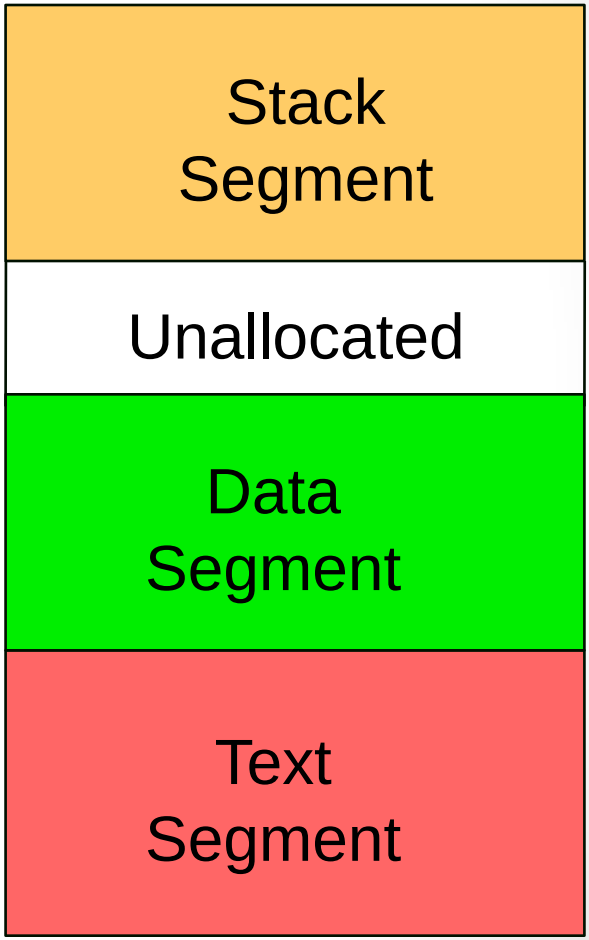
- A process is a program in execution, and all of the resources associated with that executing instance of the program
  - ◊ Memory
  - ◊ Program Counter
  - ◊ Open files, other devices, etc.

Program Layout

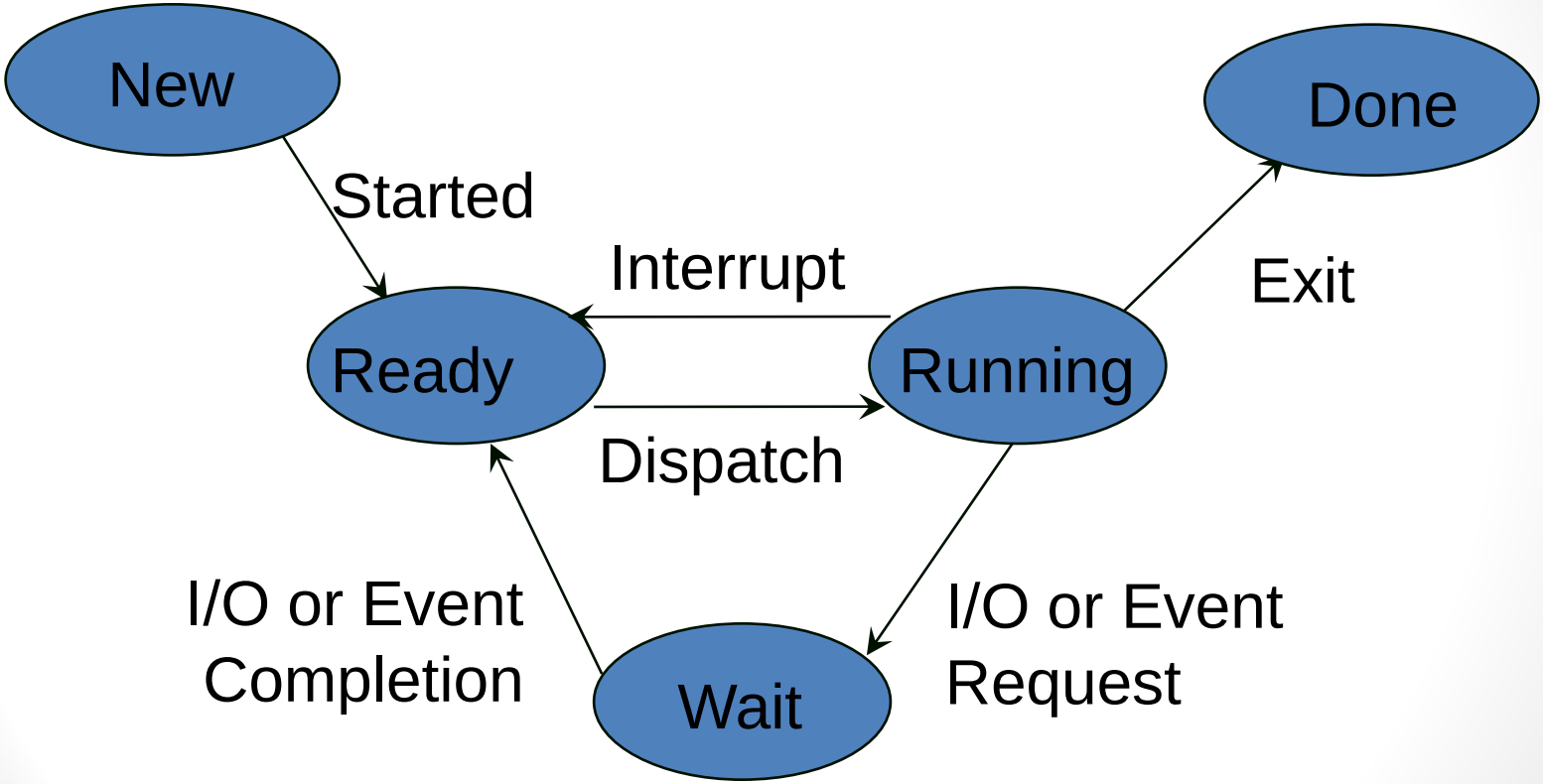
Procedure Call/Return
Local Variables
– grows downwards

Top Half is the Heap (malloc)
– grows upwards
Bottom Half is Global Vars

Executable Code
Binary Machine Instructions
Usually Shared

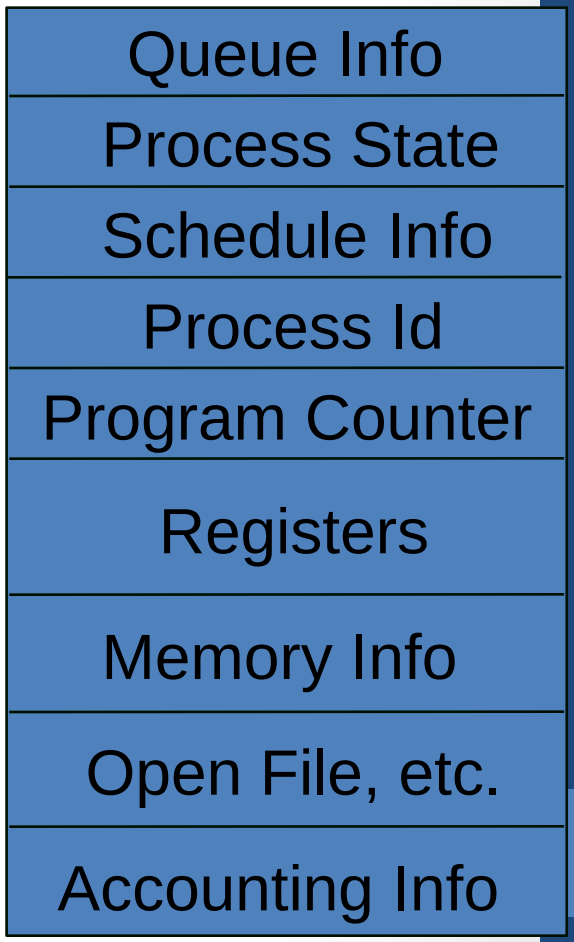


Process State

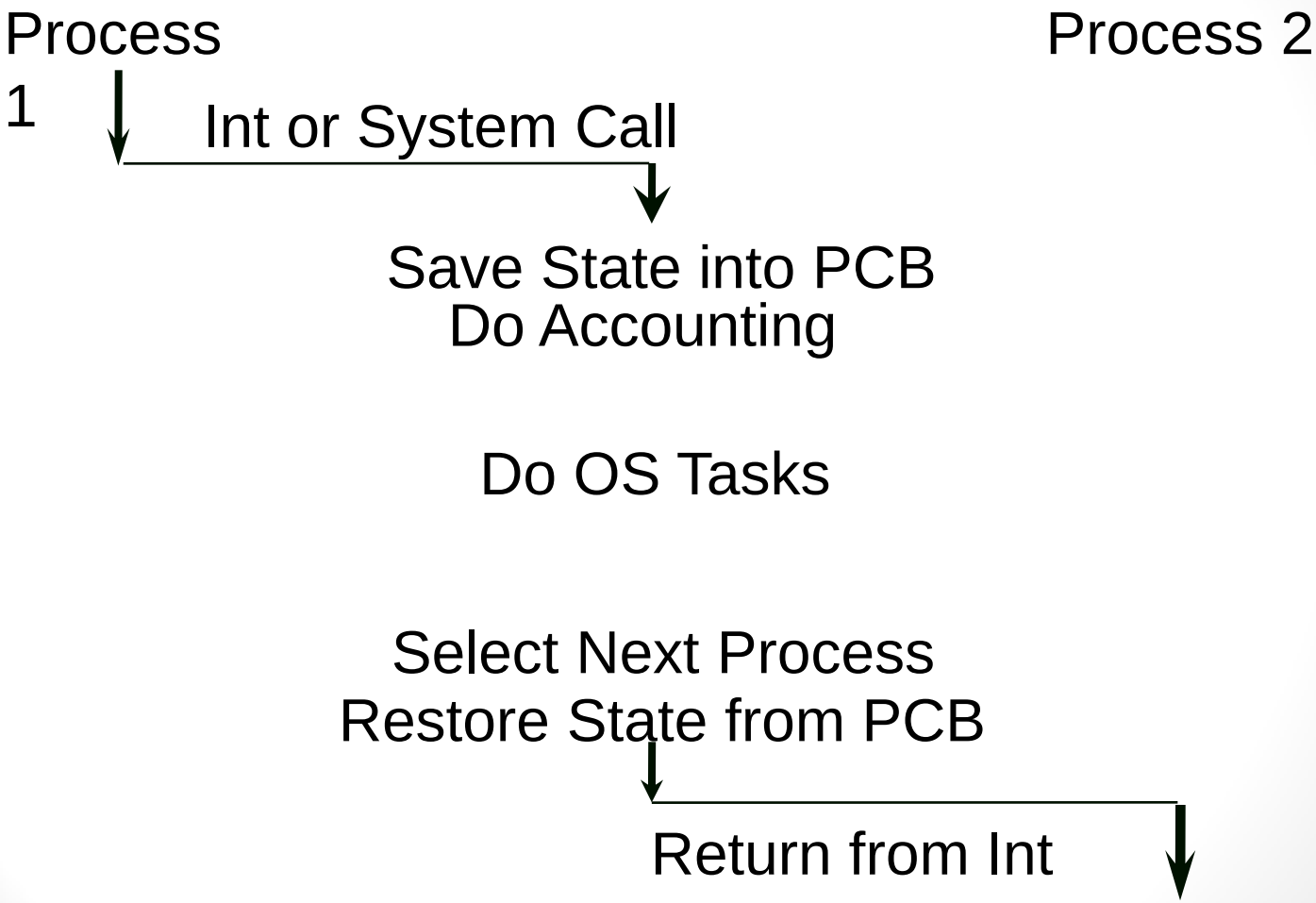


Process Control Block (PCB)

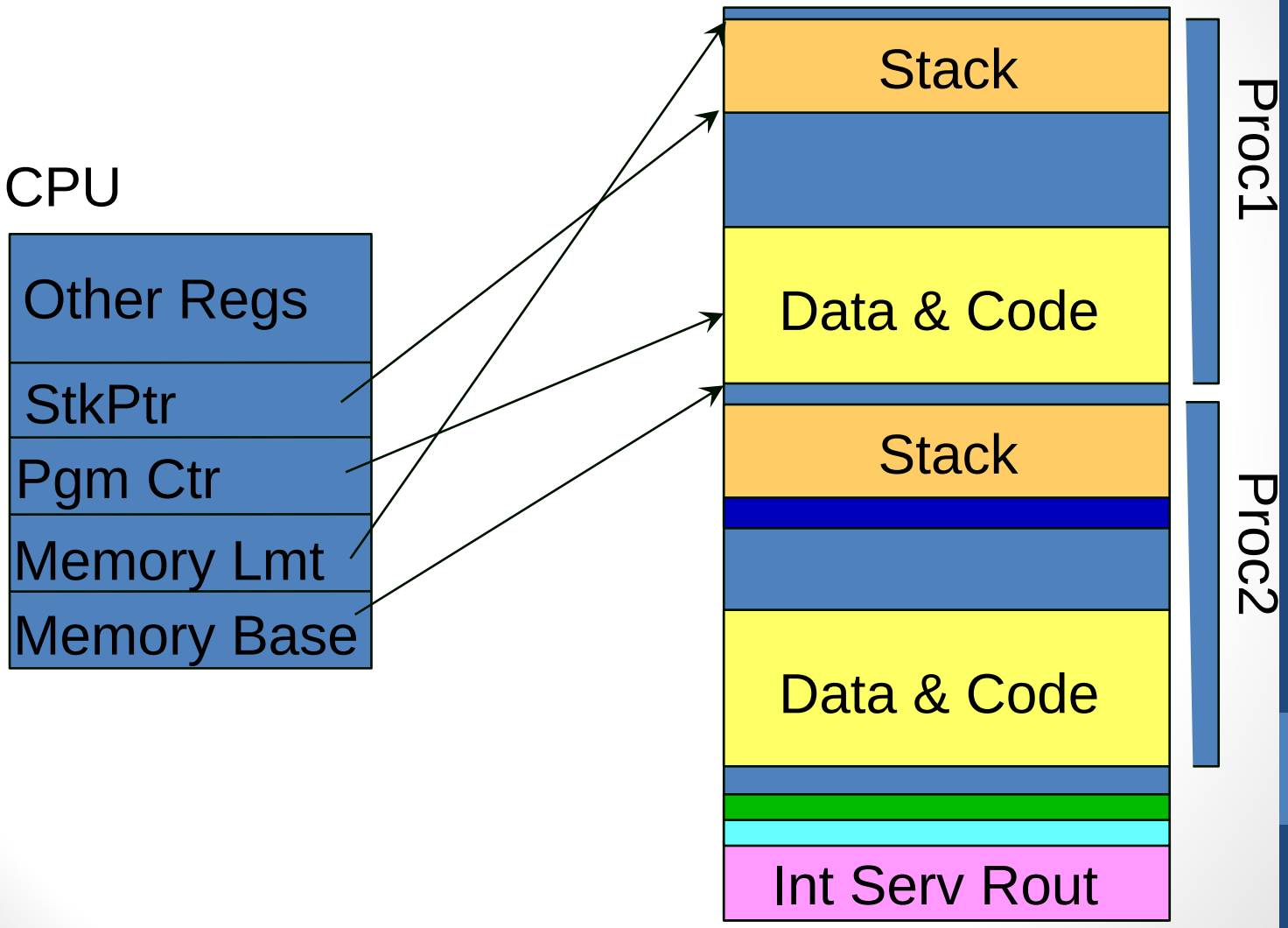
- One allocated for each process and sometimes for each thread
- Repository for information that varies from process to process
- Some operating systems have a pre allocated number of them, that is an array (early UNIX)
- Some permit dynamic allocation (Amiga OS)



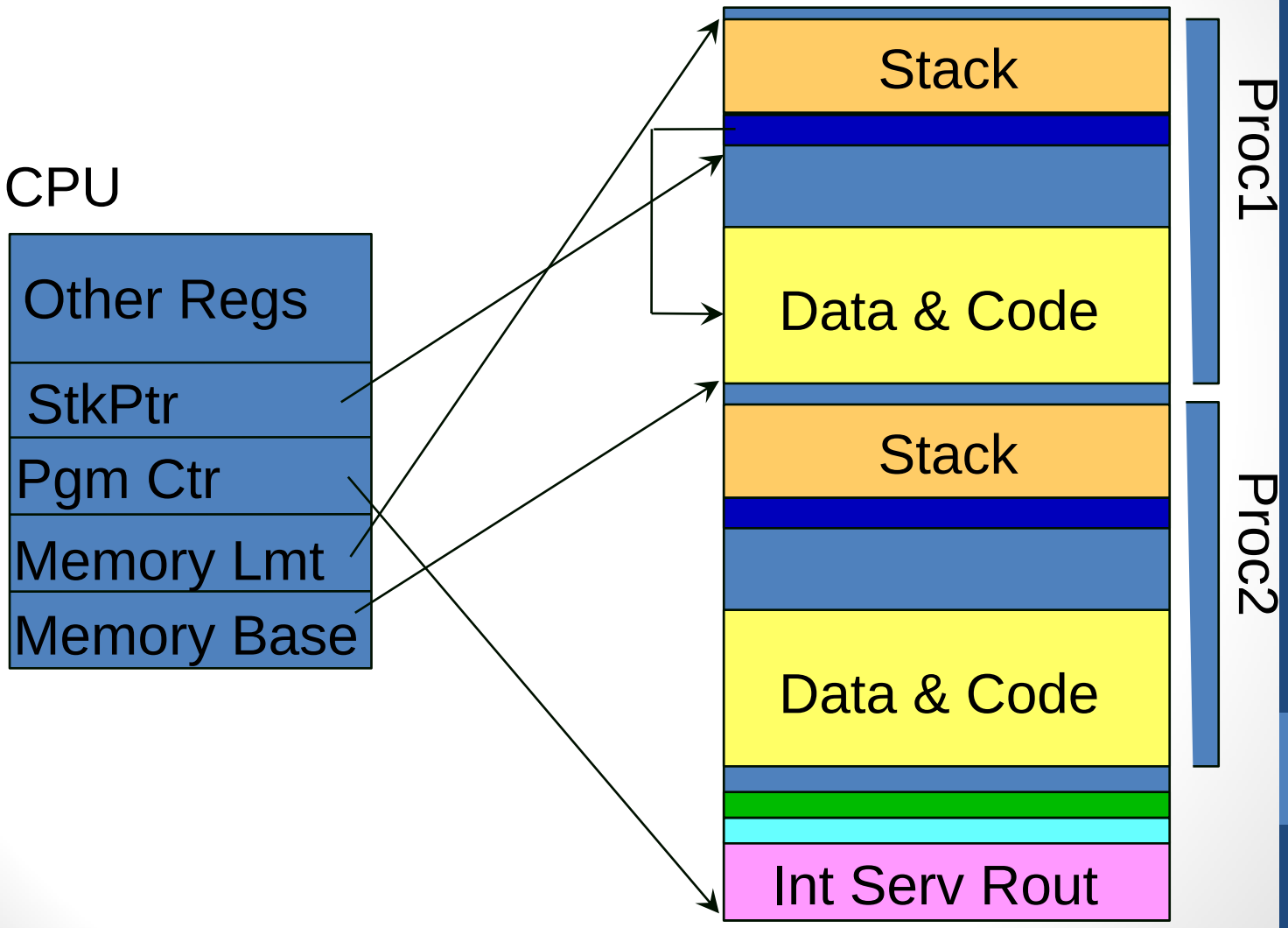
Context Switch



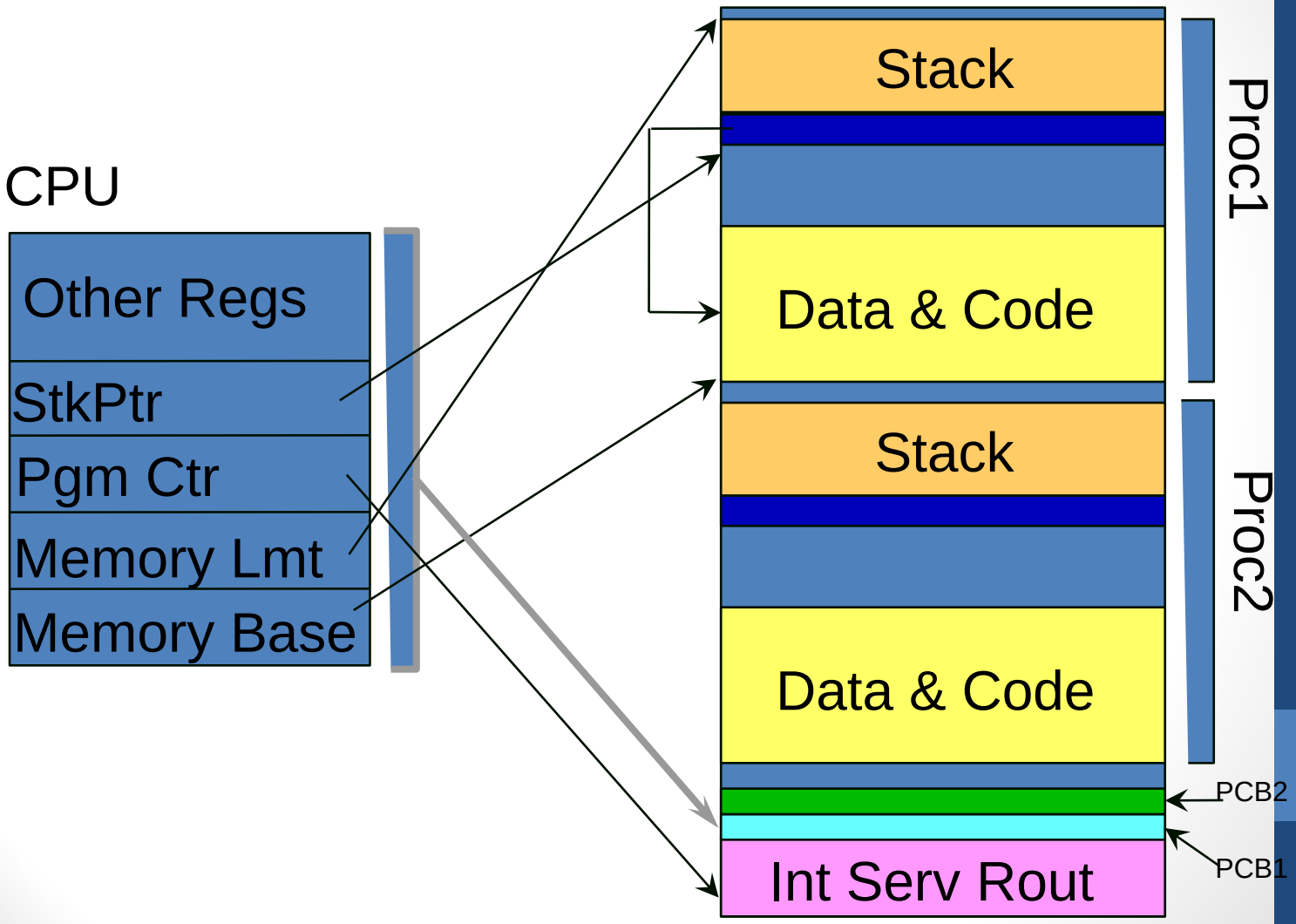
Context Switch - Start



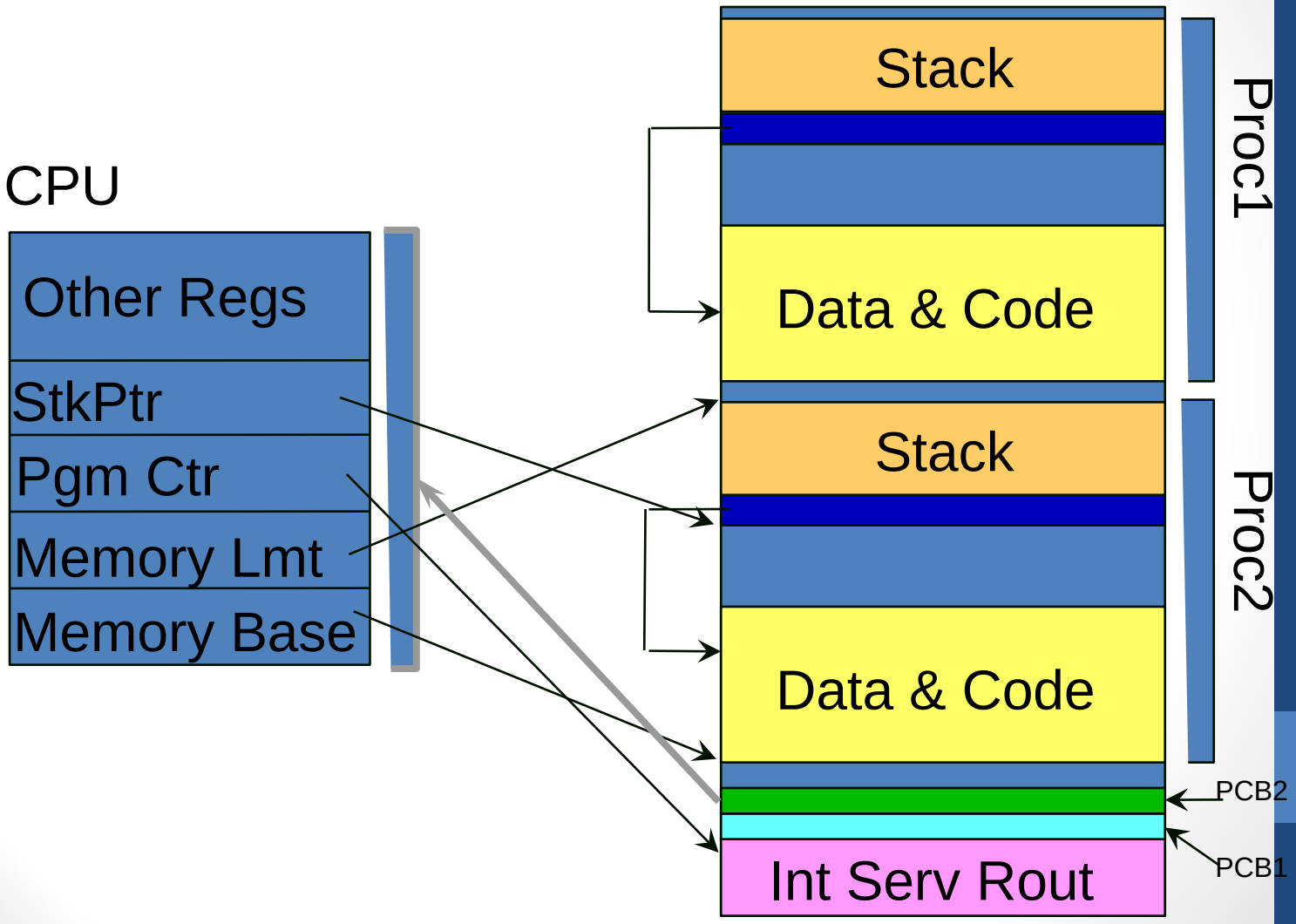
Context Switch - Int. or Sys Call



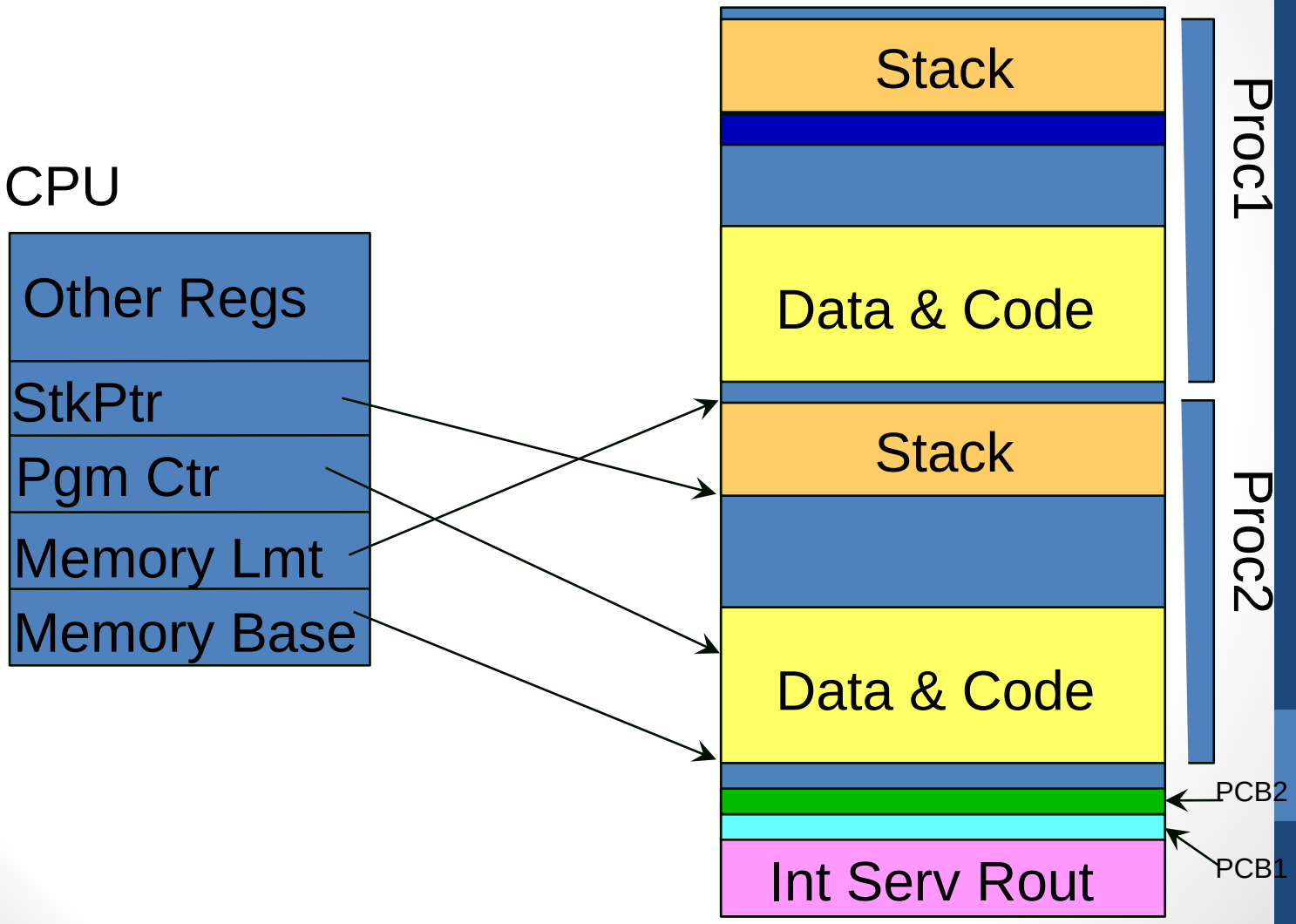
Context Switch - Save State



Context Switch - Restore State



Context Switch - Ret from Int or Trap



Context Switch

- Depends on hardware
  - ◊ number of registers to save/restore
  - ◊ supporting hardware (memory management, etc).
  - ◊ hardware support
  - – banks of registers
  - – supporting hardware (cache, TLB, etc.)

- System is not doing useful work
  - ◊ overhead of multitasking
  - ◊ 1 to 1000 micro seconds