Functional Views of OS

Operating Systems

School of Data & Computer Science
Sun Yat-sen University

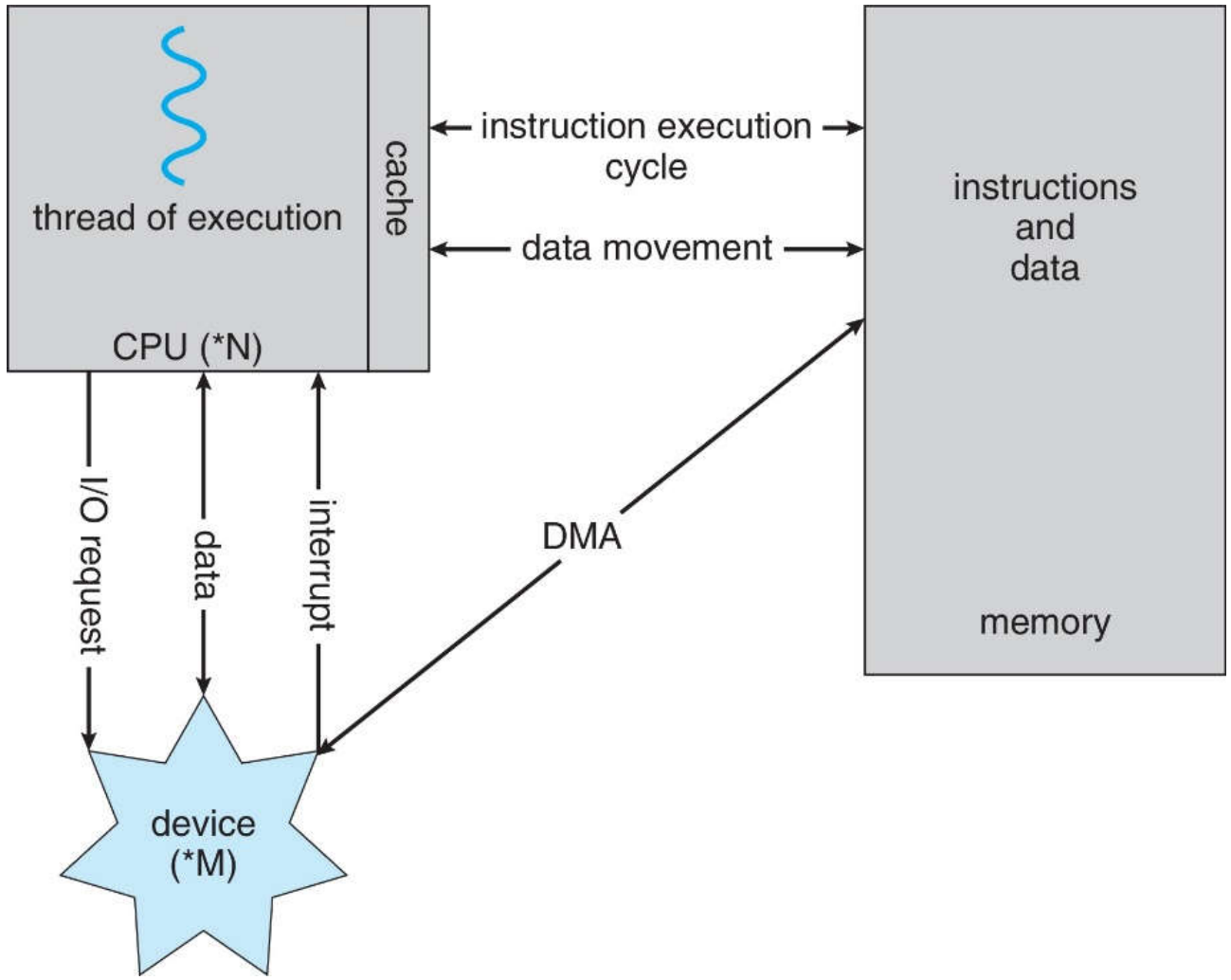
Lecture Notes: os_sysu@163.com
Instructor: Guoyang Cai
email: isscgy@mail.sysu.edu.cn

Contents

- Computer Dynamics
- Hardware Protection
  - Dual-mode Protection
  - I/O Protection
  - Memory Protection
  - CPU Protection
- Fundamental OS Concepts
  - Thread
  - Address space
  - Process
  - Dual-mode Operation
- **Resource Management (Lecture 04: Common OS Components)
  - Process Management
  - Memory Management
  - File Management
  - Mass-storage Management
  - Cache Management
  - I/O Management
- Free and Open Source Operating Systems

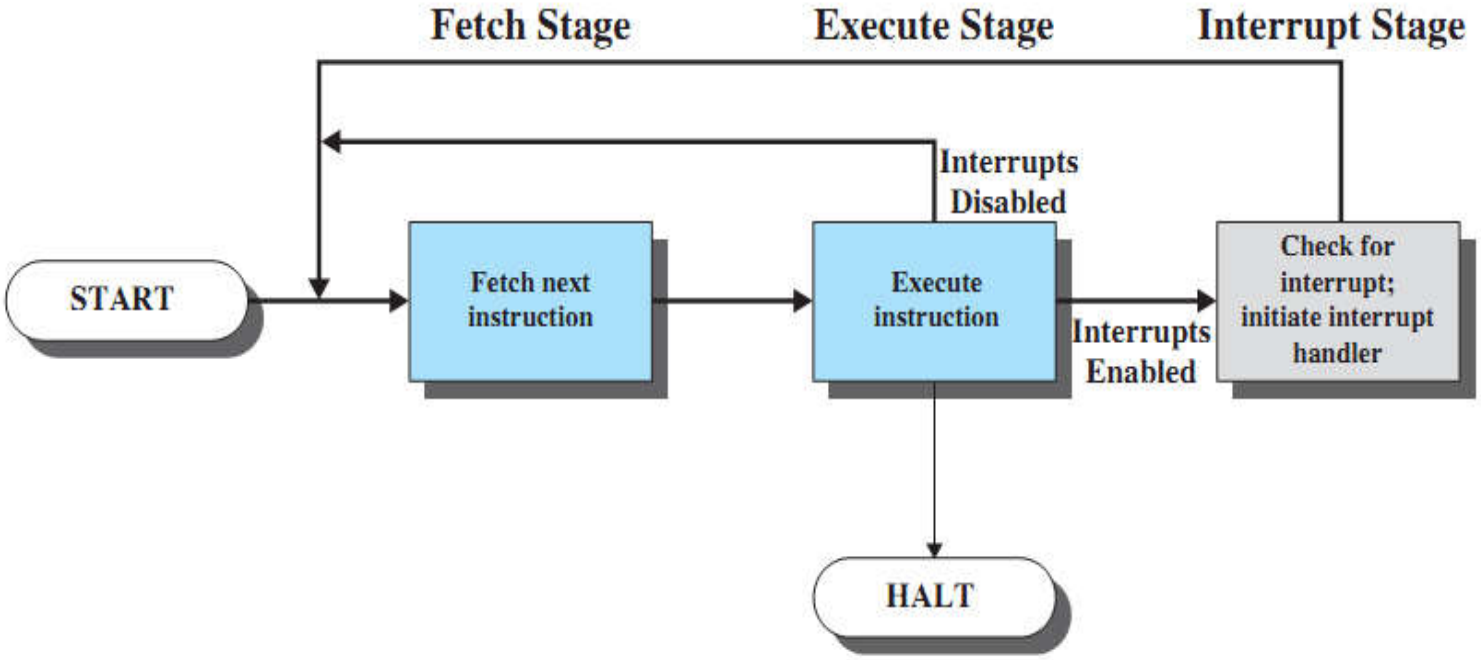
Computer Dynamics

- How a modern computer system works.



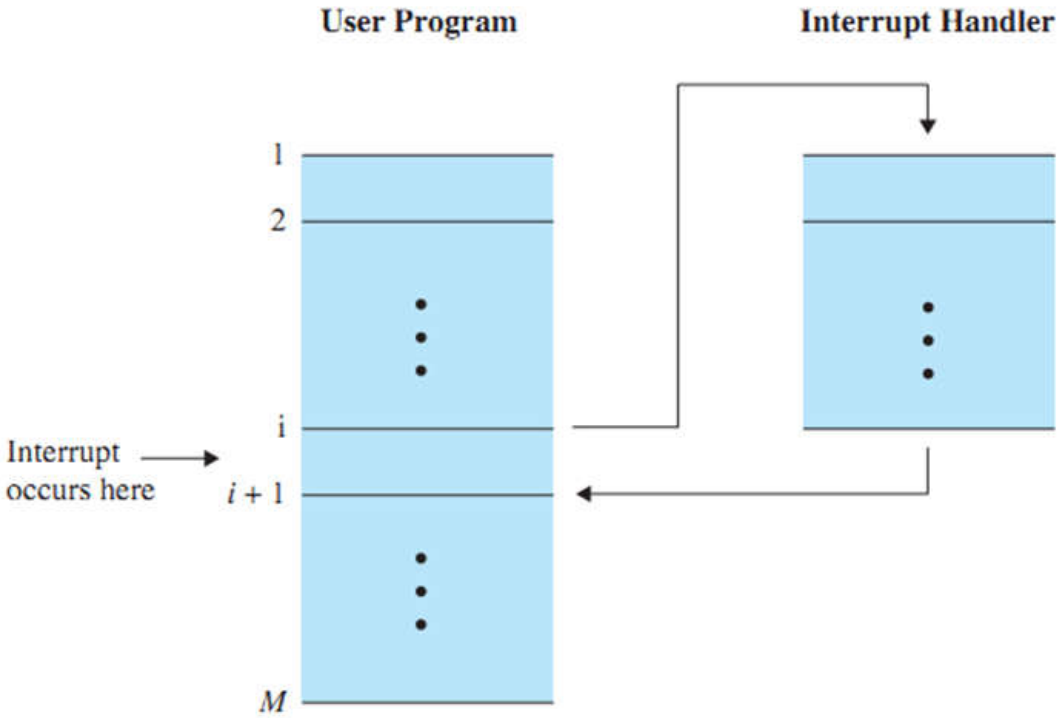
Computer Dynamics

- Instruction Cycle with Interrupts
  - CPU checks for interrupts after each instruction.
  - If no interrupts, then fetch next instruction of current program.
  - If an interrupt is pending, then suspend execution of the current program, and execute the interrupt handler.



Computer Dynamics

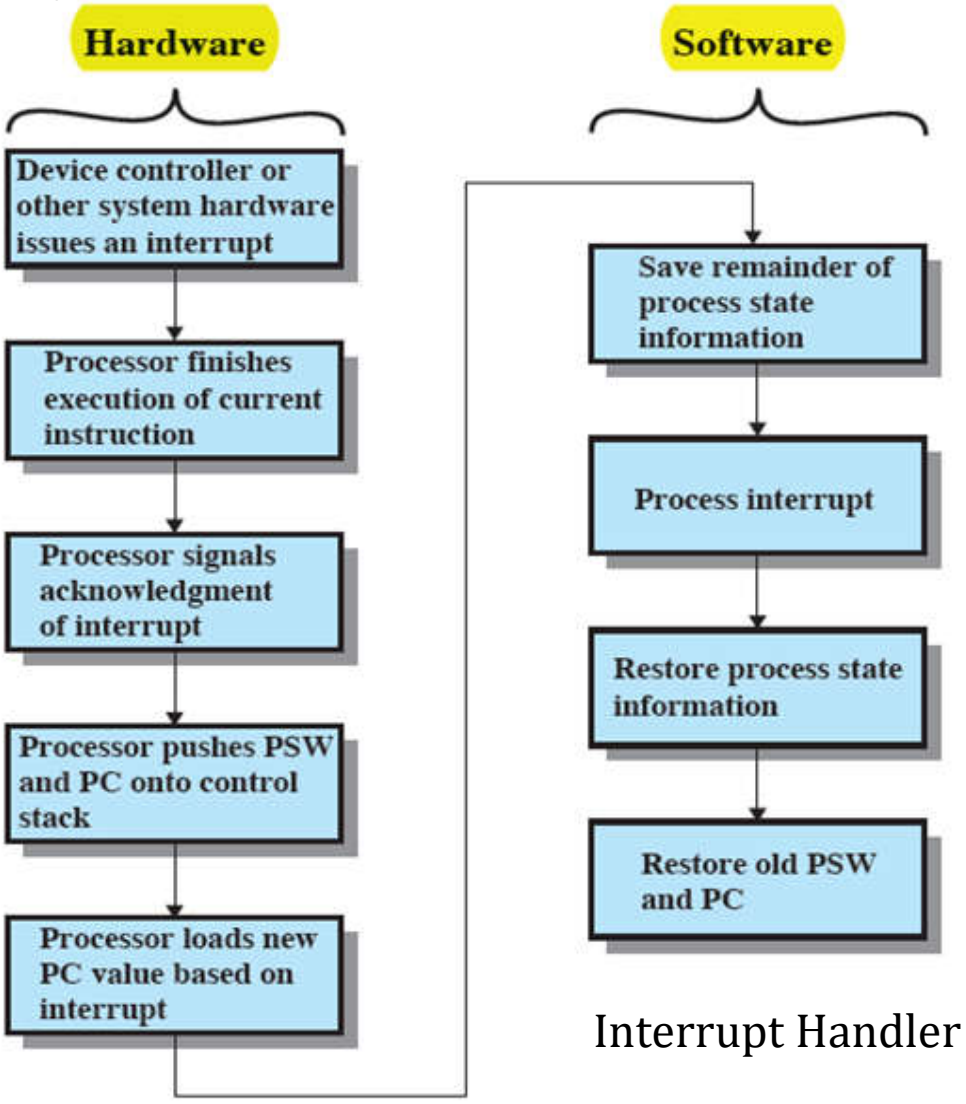
- Instruction Cycle with Interrupts
  - CPU checks for interrupts after each instruction.
  - If no interrupts, then fetch next instruction of current program.
  - If an interrupt is pending, then suspend execution of the current program, and execute the interrupt handler.



Transfer of Control via Interrupt

Computer Dynamics

- Instruction Cycle with Interrupts
  - Interrupt Mechanism.



Interrupt Handler

Computer Dynamics

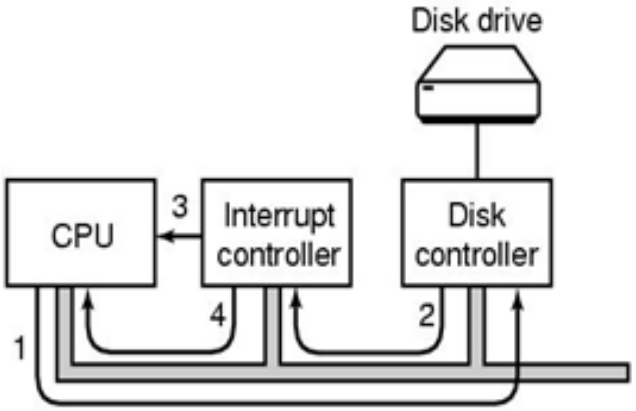
- External Interrupt
  - An external interrupt is a temporal suspension of a process caused by an event external to that process and performed in such a way that the process can be resumed.
  - Events external to the process that cause external interrupts:
    - I/O
    - Timer
    - Hardware failure.

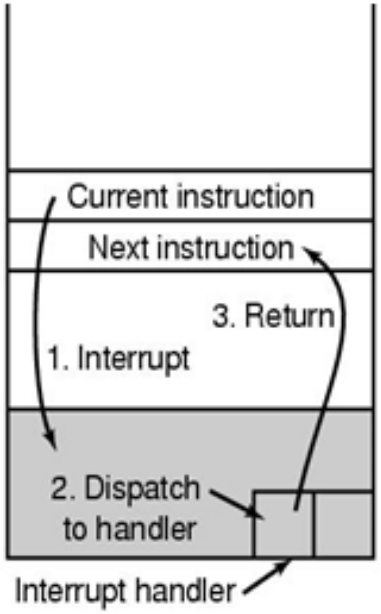
Computer Dynamics

- Interrupt Handler
  - *Interrupt handler* is a program that determines nature of the interrupt and performs whatever actions are needed.
  - When an external interrupt event occurs, the external interrupt hardware transfers control to the interrupt handler, generally through the *interrupt vector*, which contains the addresses of all interrupt service routines (ISR).
  - Interrupt architecture must save the state of the program (content of PSW, PC, registers, ...).
  - Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.
  - Later, control must be transferred back to the interrupted program so that it can be resumed from point of interruption.

Computer Dynamics

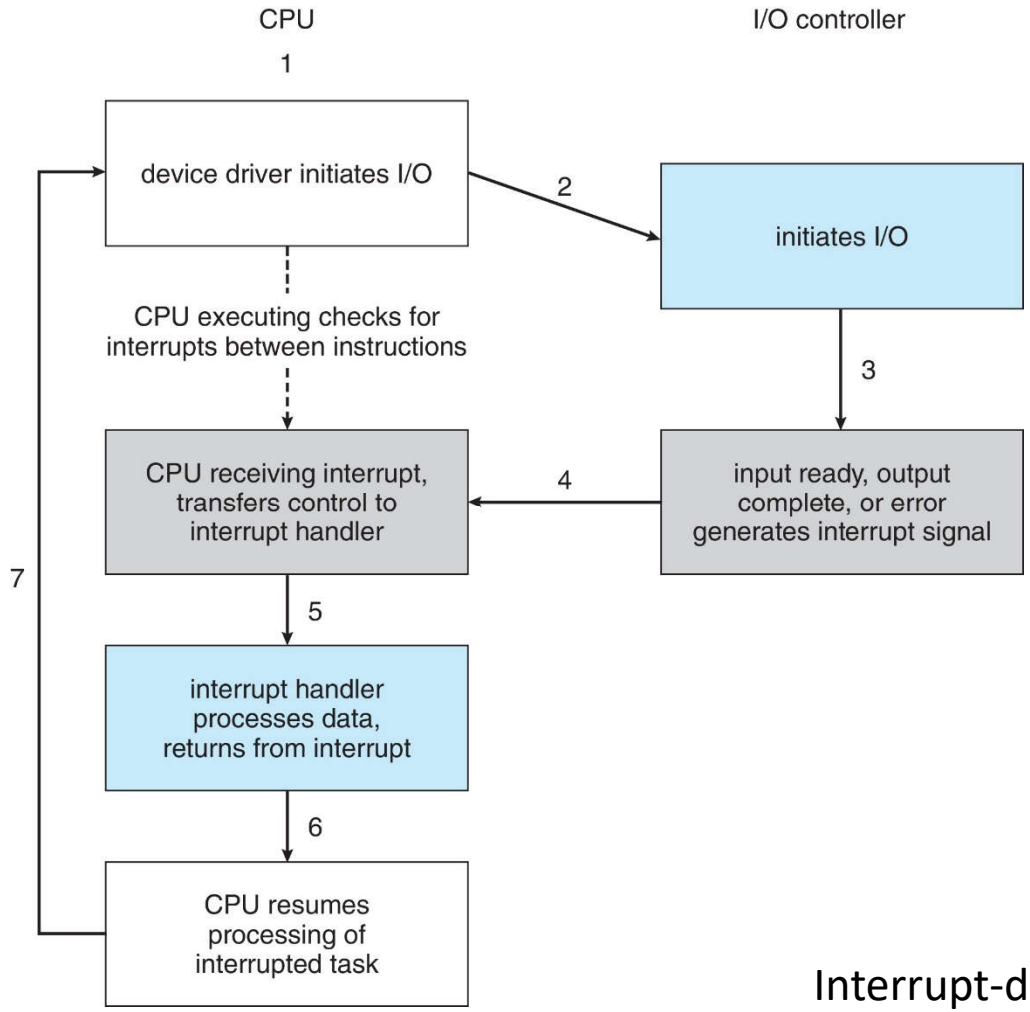
- Interrupt-driven I/O
  - I/O devices and the CPU can execute concurrently.
  - Each device controller is in charge of a particular device type.
  - Each device controller has a local buffer.
  - CPU moves data from/to main memory to/from local buffers.
  - I/O is from/to the device to/from local buffer of controller.
  - Device controller informs CPU that it has finished its operation by causing an External Interrupt.





Computer Dynamics

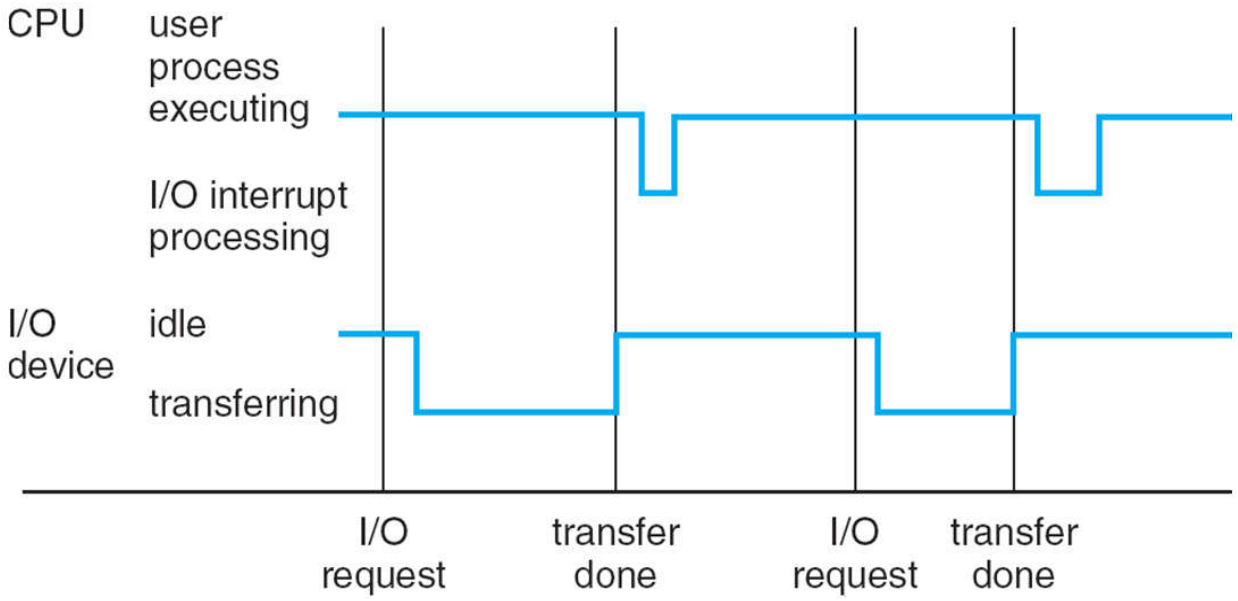
- Interrupt-driven I/O.



Interrupt-driven I/O Cycle

Computer Dynamics

- Interrupt Timeline of CPU and I/O Device.

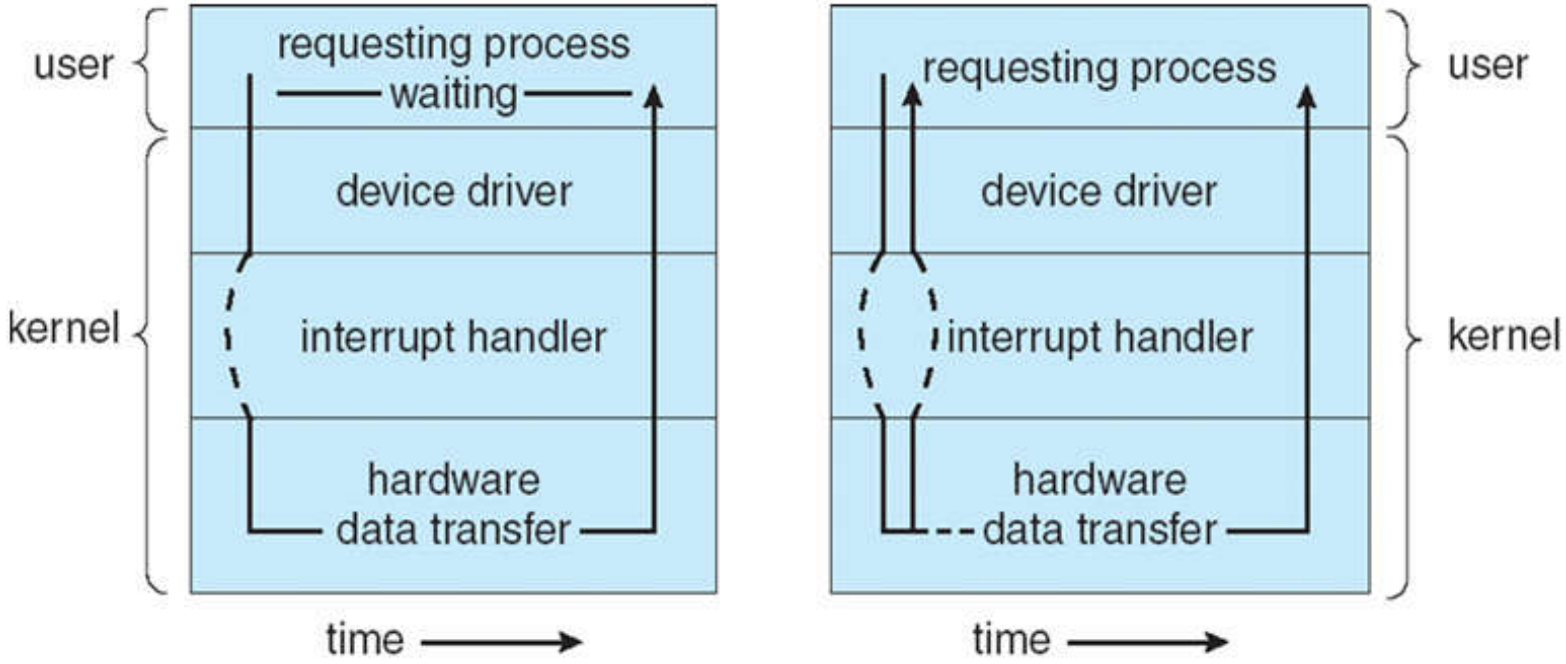


Computer Dynamics

- Synchronous and Asynchronous I/O Methods
  - Synchronous I/O
    - After I/O starts, control returns to user program only upon I/O completion.
    - Wait instruction idles the CPU until the next interrupt.
    - Wait loop (contention for memory access).
    - At most one I/O request is outstanding at a time, no simultaneous I/O processing.
  - Asynchronous I/O
    - After I/O starts, control returns to user program without waiting for I/O completion.
    - *System call* requests to OS to allow user to wait for I/O completion.
    - *Device-status table* contains entry for each I/O device indicating its type, address, and state.
      - Operating system indexes into I/O *Device-status Table* to determine device status and to modify table entry to include interrupt.

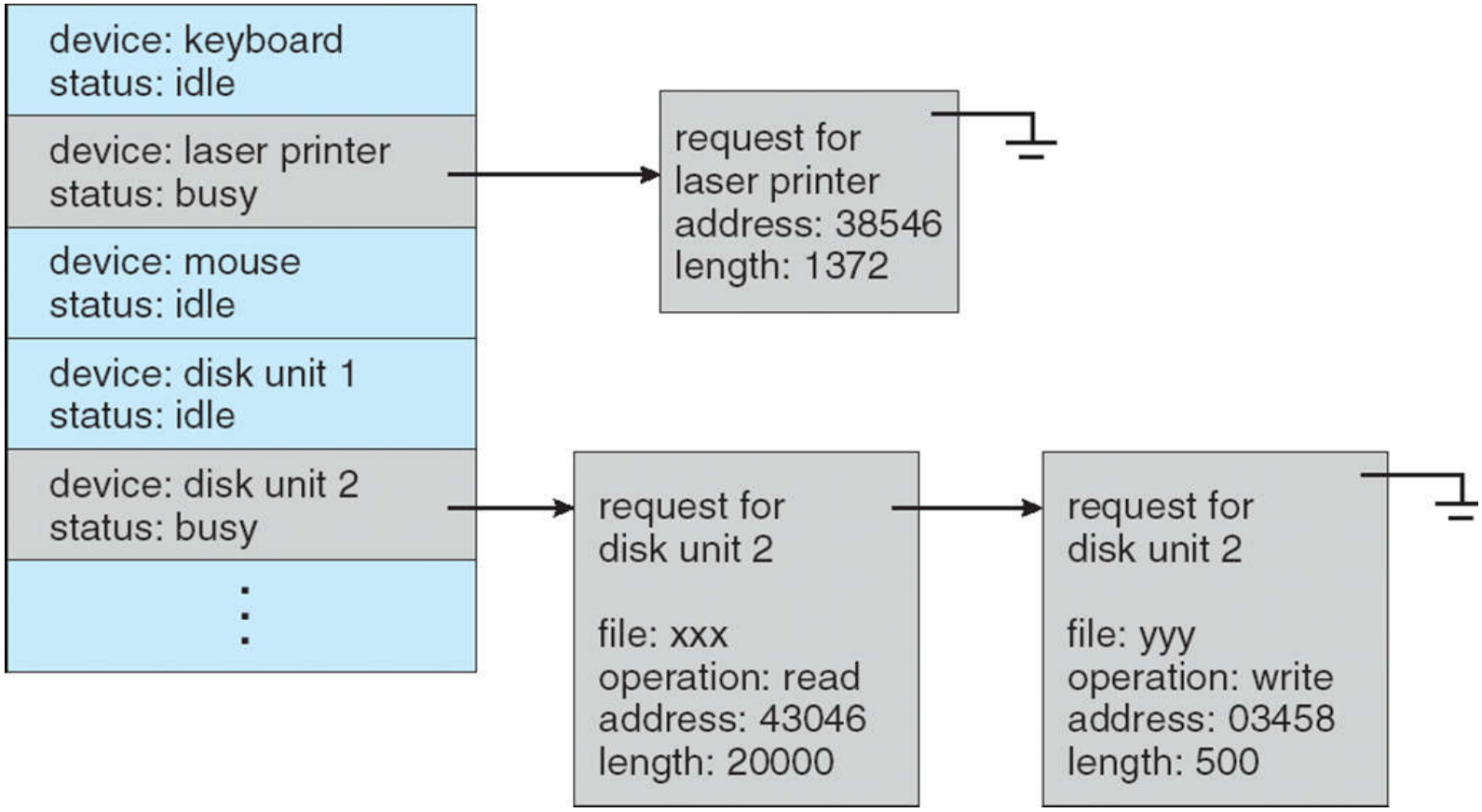
Computer Dynamics

- Synchronous and Asynchronous I/O Methods.



Computer Dynamics

- Synchronous and Asynchronous I/O Methods.



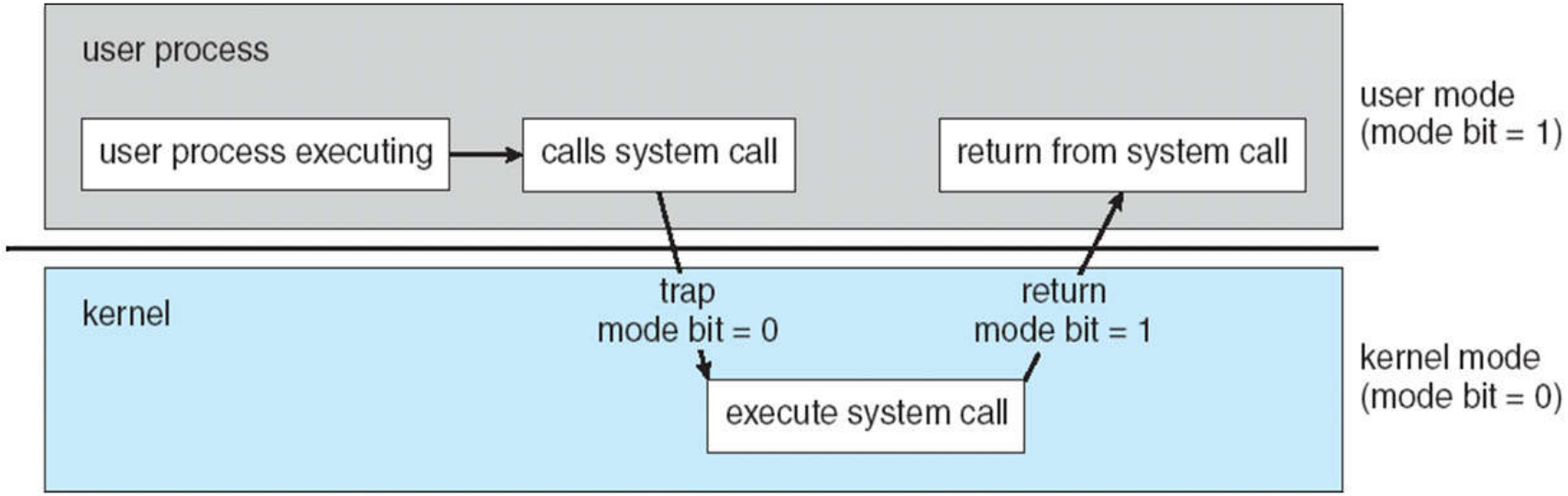
Device-status Table

Computer Dynamics

- Direct Memory Access (DMA)
  - DMA is used by smart high-speed I/O devices able to transmit information at close to memory speeds.
  - DMA Device controller transfers blocks of data *from buffer* storage directly *to main memory* without CPU intervention.
  - Only *one interrupt* is generated *per block*, rather than one interrupt per byte.
  - The CPU goes back to work after given the DMA controller:
    - disk address,
    - memory address, and
    - a byte count.

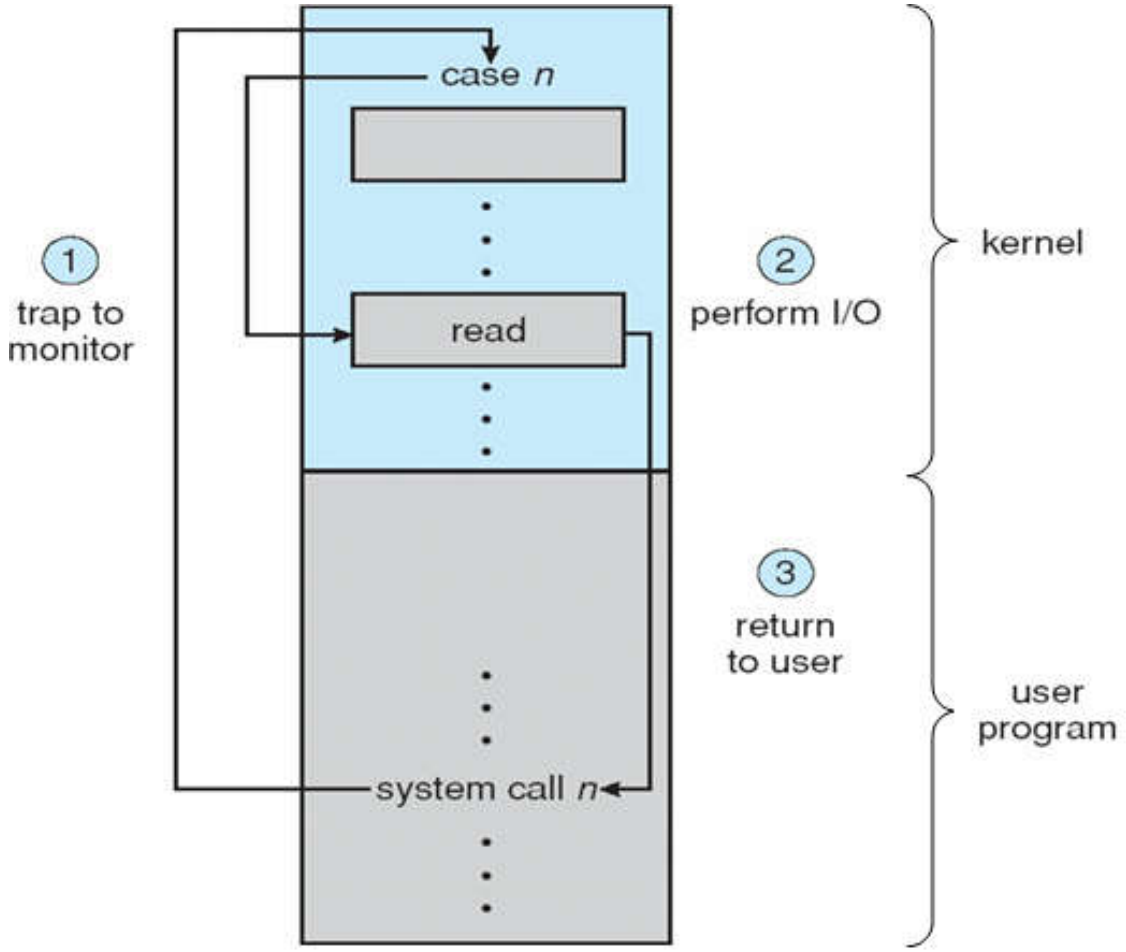
Computer Dynamics

- System Call
  - System Call is the method used by a process to request action by the operating system:
    - After system call parameter preparations, it uses the trap instruction to transfer control to the requested service routine in the OS kernel.
    - The system verifies that the parameters are correct and legal, and executes the request.
    - Returns control to the instruction following the system call.



Computer Dynamics

- System Call
  - System Call used to perform I/O.



Hardware Protection

- Dual-Mode Protection
  - Dual-Mode Operation ensures that an incorrect program cannot cause other programs (including kernel process) to execute incorrectly.
  - Realization: Provide hardware support to differentiate between at least two modes of operations. *Mode bit* is added to computer hardware to indicate the current mode: Kernel Mode (0) or User Mode (1).
    - User Mode (用户态或目态) – execution done on behalf of a user.
    - Kernel Mode (aka monitor mode, system mode 内核态, 管态或系统态) – execution done on behalf of operating system.
  - Privileged instructions can be issued only in kernel mode.
  - Three types of Unprogrammed Control Transfer from User Mode to Kernel Mode:
    - System call
    - Interrupt
    - Trap

Hardware Protection

- Dual-Mode Protection
  - System Call
    - Process requests a system service, e.g., exit.
    - Like a function call, but “outside” the process.
    - Does not have the address of the system function to call.
    - Like a Remote Procedure Call (RPC).
    - Marshall the syscall id in `EAX`, args in `EBX`, `ECX`, `EDX` and exec syscall (`int 0x80`).
  - Interrupt
    - External asynchronous event triggers context switch.
      - e.g., Timer, I/O device
    - It is independent of user process.
  - Trap
    - Internal synchronous event in process triggers context switch.
      - e.g., Protection violation (segmentation fault), Divide by zero, …

Hardware Protection

- I/O Protection
  - User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions.
    - All I/O devices need to be protected from wrongdoing by the users.
      - e.g., prevent current program from reading control cards of next job.
    - All I/O instructions need to be privileged instructions.
  - Given that the I/O instructions are privileged, how does the user program perform I/O?
    - Solution: System Calls from user programs.
- Memory Protection
  - In order to have memory protection, *Base Register* and *Limit Register* are used to determine the range of legal addresses a program may access.
  - Memory outside the defined range is protected.
  - The load instructions for the base and limit registers are *privileged* instructions.

Hardware Protection

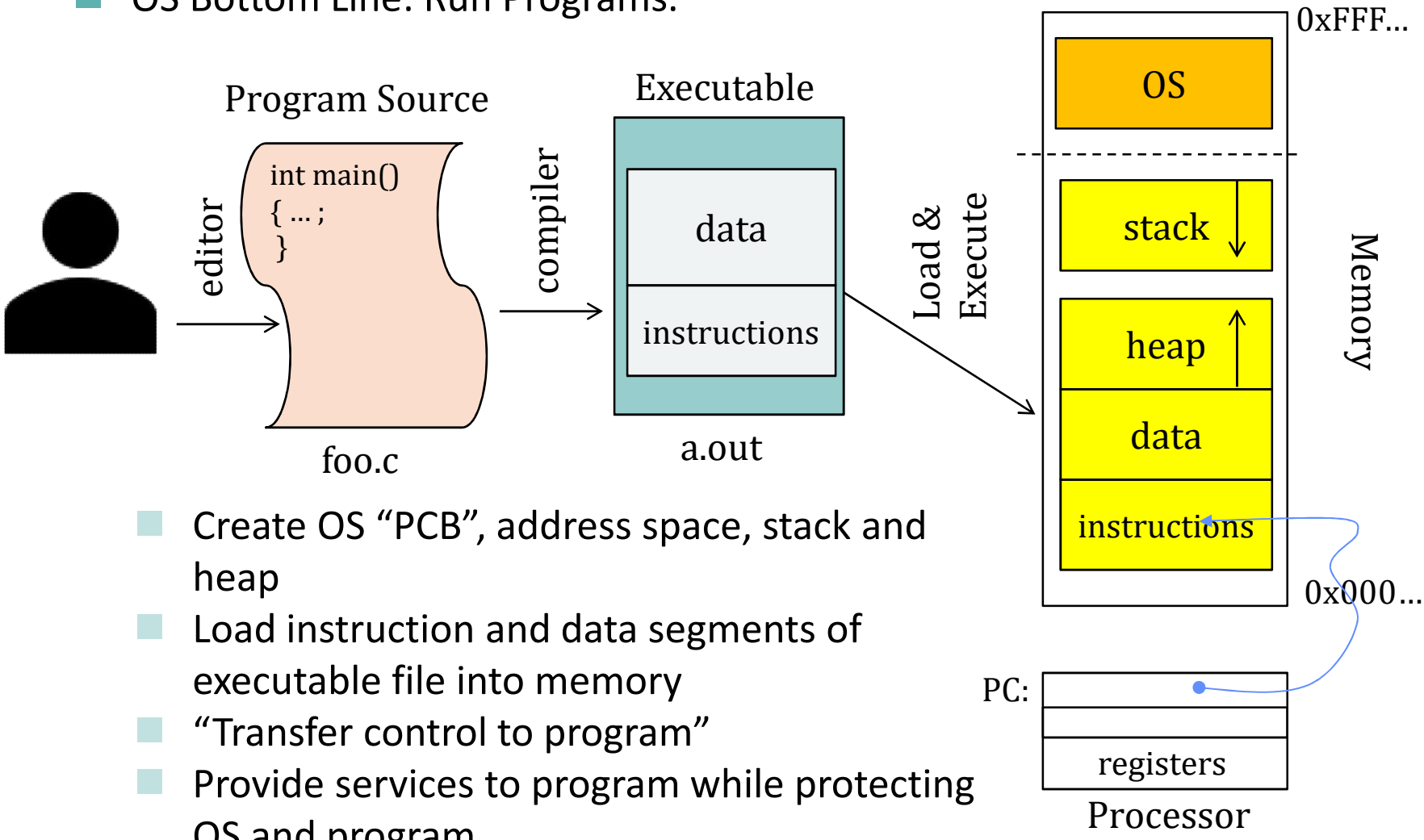
- CPU Protection
  - Timer
    - Timer interrupts computer after specified period to ensure operating system maintains control.
    - Programmable interval timer is used for timings, periodic interrupts.
    - Set timer is a privileged instruction.
  - Timer is commonly used to implement Time Sharing Systems.
  - Timer is set to interrupt the computer after some time period.
    - Timer is decremented every physical clock tick.
    - When timer reaches the value 0, an interrupt occurs.
    - Load-timer (OS sets the counter) is a *privileged* instruction.
    - Set up before scheduling process to regain control or terminate program that exceeds allotted (分配的) time.

Fundamental OS Concepts

- Thread: Execution Context
  - Fully describes program state
  - Program Counter, Registers, Execution Flags, Stack
- Address Space (with or w/o translation)
  - Set of memory addresses accessible to program (for read or write)
  - May be distinct from memory space of the physical machine (in which case programs operate in a virtual address space)
- Process: an instance of a running program
  - Protected Address Space + One or more Threads
- Dual Mode Operation / Protection
  - Only the “system” has the ability to access certain resources
  - Combined with translation, isolates programs from each other and the OS from programs

Fundamental OS Concepts

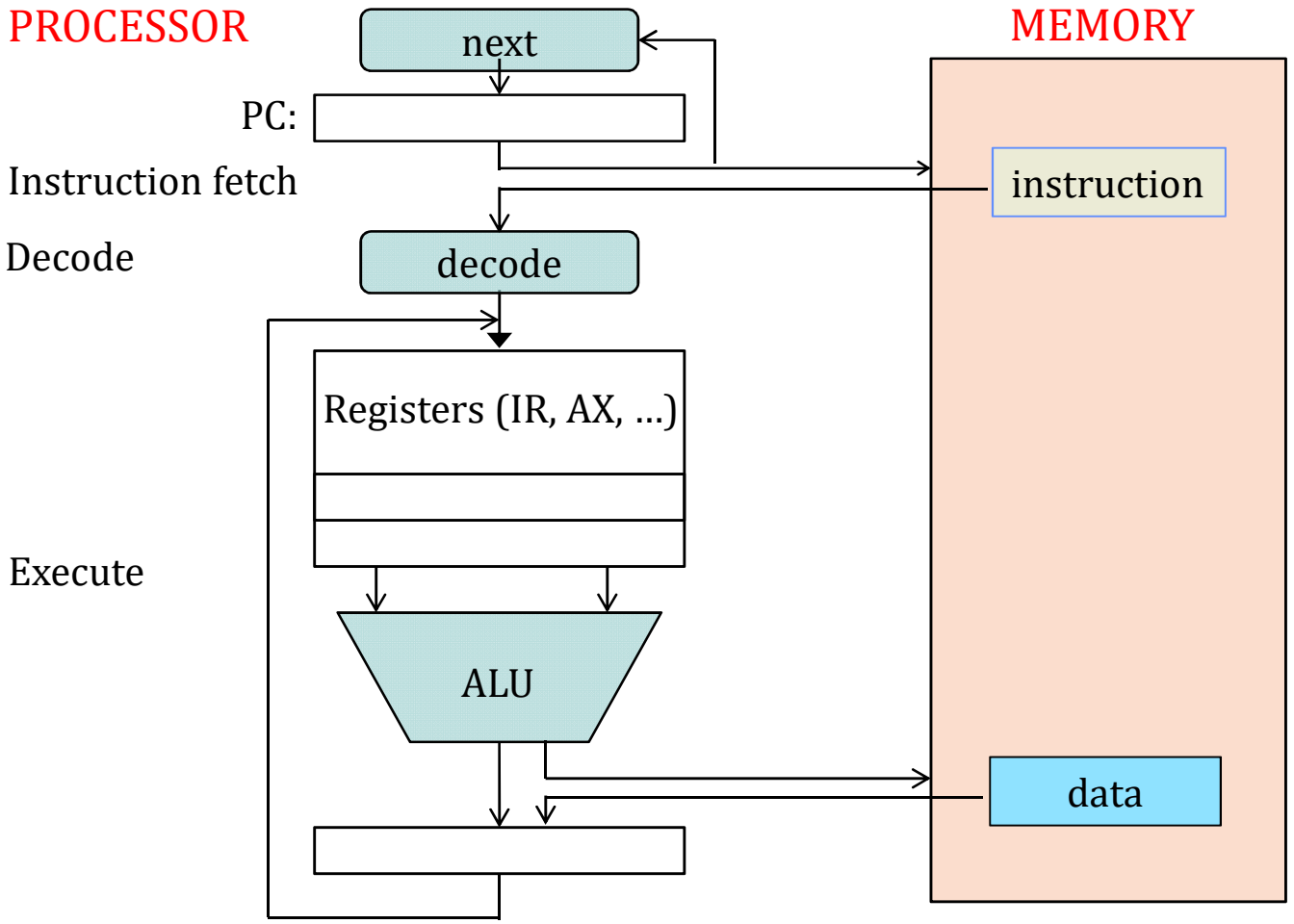
- OS Bottom Line: Run Programs.



- Create OS "PCB", address space, stack and heap
- Load instruction and data segments of executable file into memory
- "Transfer control to program"
- Provide services to program while protecting OS and program

Fundamental OS Concepts

- The instruction cycle.



Threads

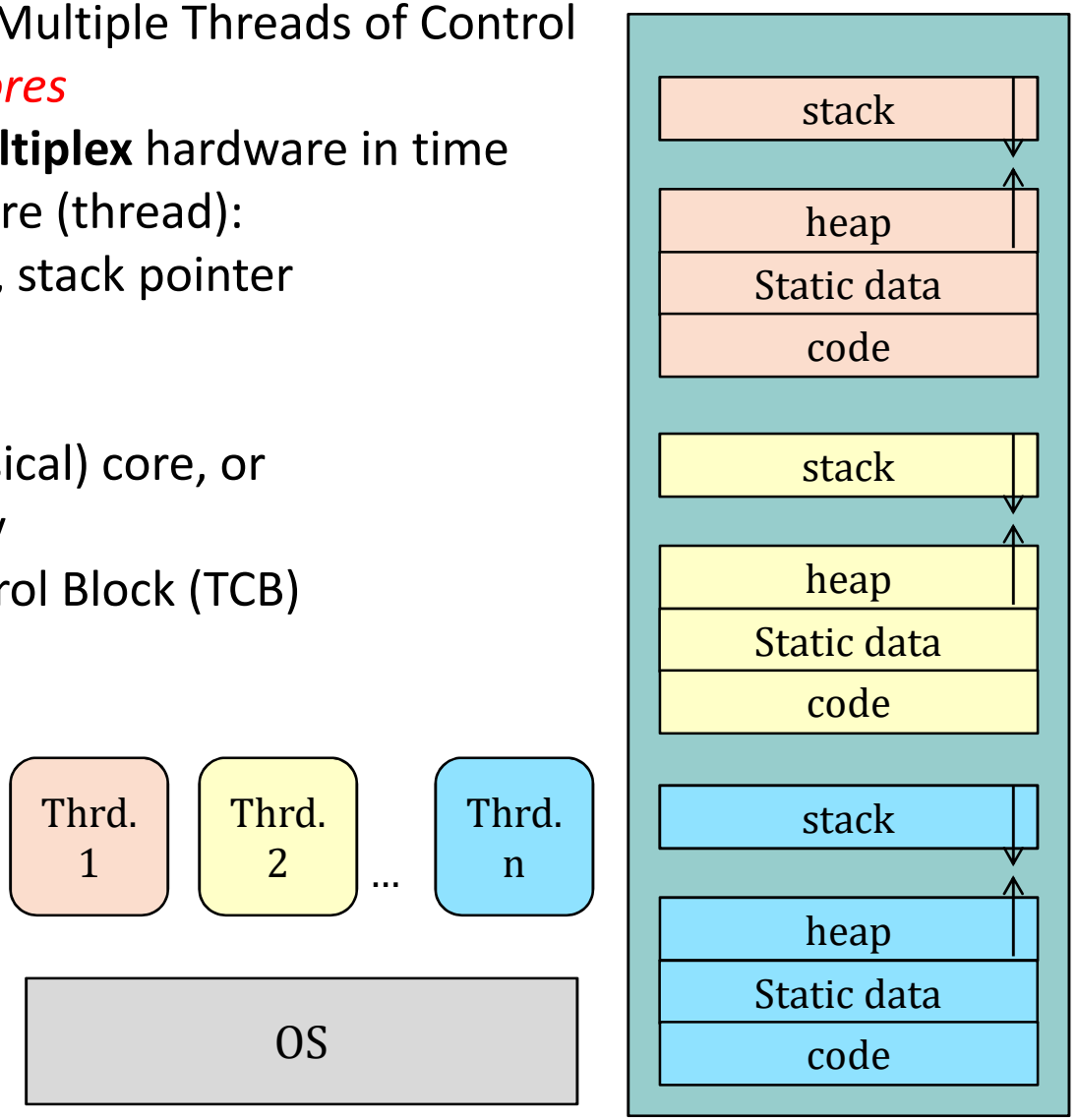
- A single, unique execution context
  - Program counter, Registers, Execution flags, Stack, Memory state
- A thread is *executing* on a processor (core) when it is *resident* in that processor registers
- *Resident* means: Registers hold the root state (context) of the thread:
  - Including program counter (PC) register & currently executing instruction
    - PC points at next instruction stored in memory.
  - Including intermediate values for ongoing computations
    - Can include actual values (like integers) or pointers to values in memory
  - Stack pointer holds the address of the top of the thread's own stack (which is in memory)
  - The rest is "in memory"

Threads

- A thread is *suspended* (not *executing)* when its state *is not* loaded (resident) into the processor
  - Processor state pointing at some other thread
  - Program counter register *is not* pointing at next instruction from this thread
  - Often: a copy of the last value for each register stored in memory

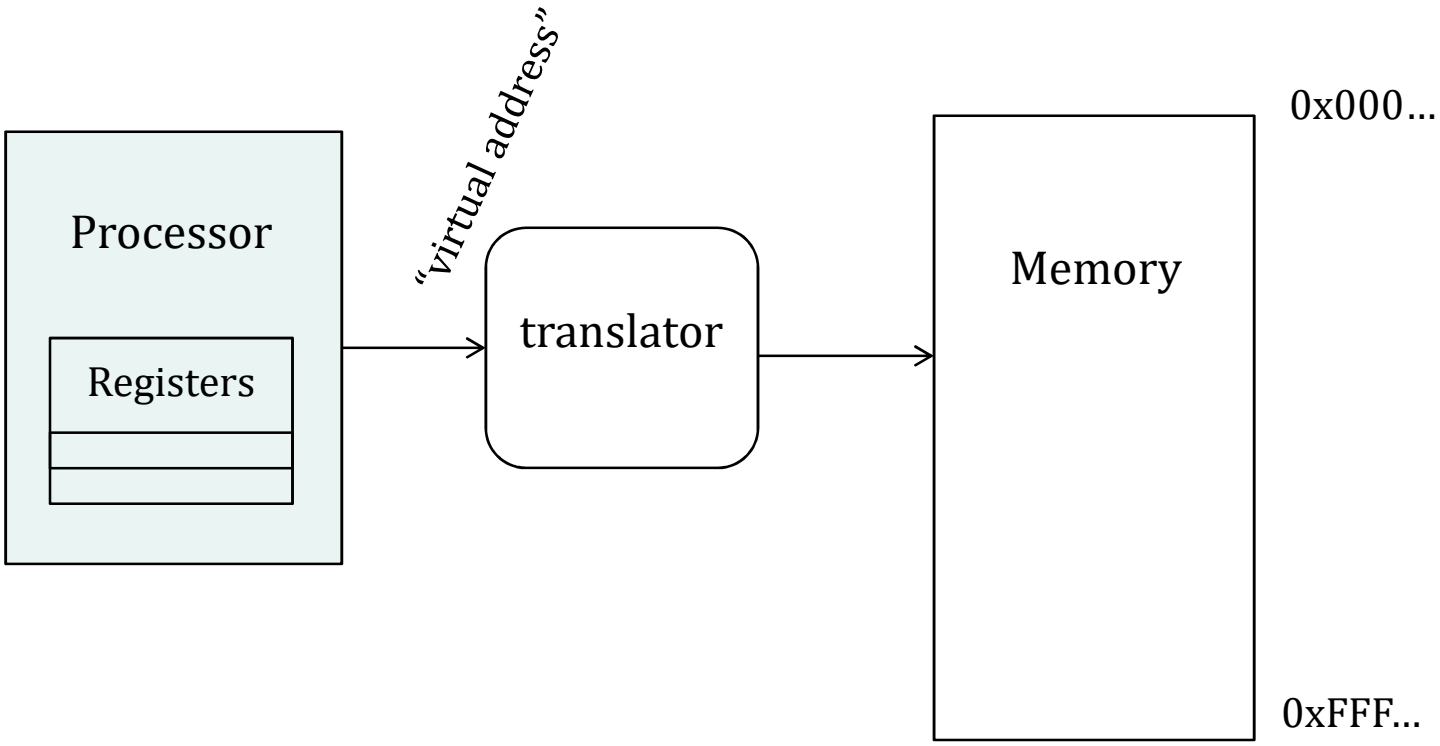
Threads

- *Multiprogramming* - Multiple Threads of Control
- Threads are *virtual cores*
- Multiple threads: **Multiplex** hardware in time
- Contents of virtual core (thread):
  - Program counter, stack pointer
  - Registers
- Where is it?
  - On the real (physical) core, or
  - Saved in memory
    - Thread Control Block (TCB)



Address Space

- Program operates in an address space that is distinct from the physical memory space of the machine.

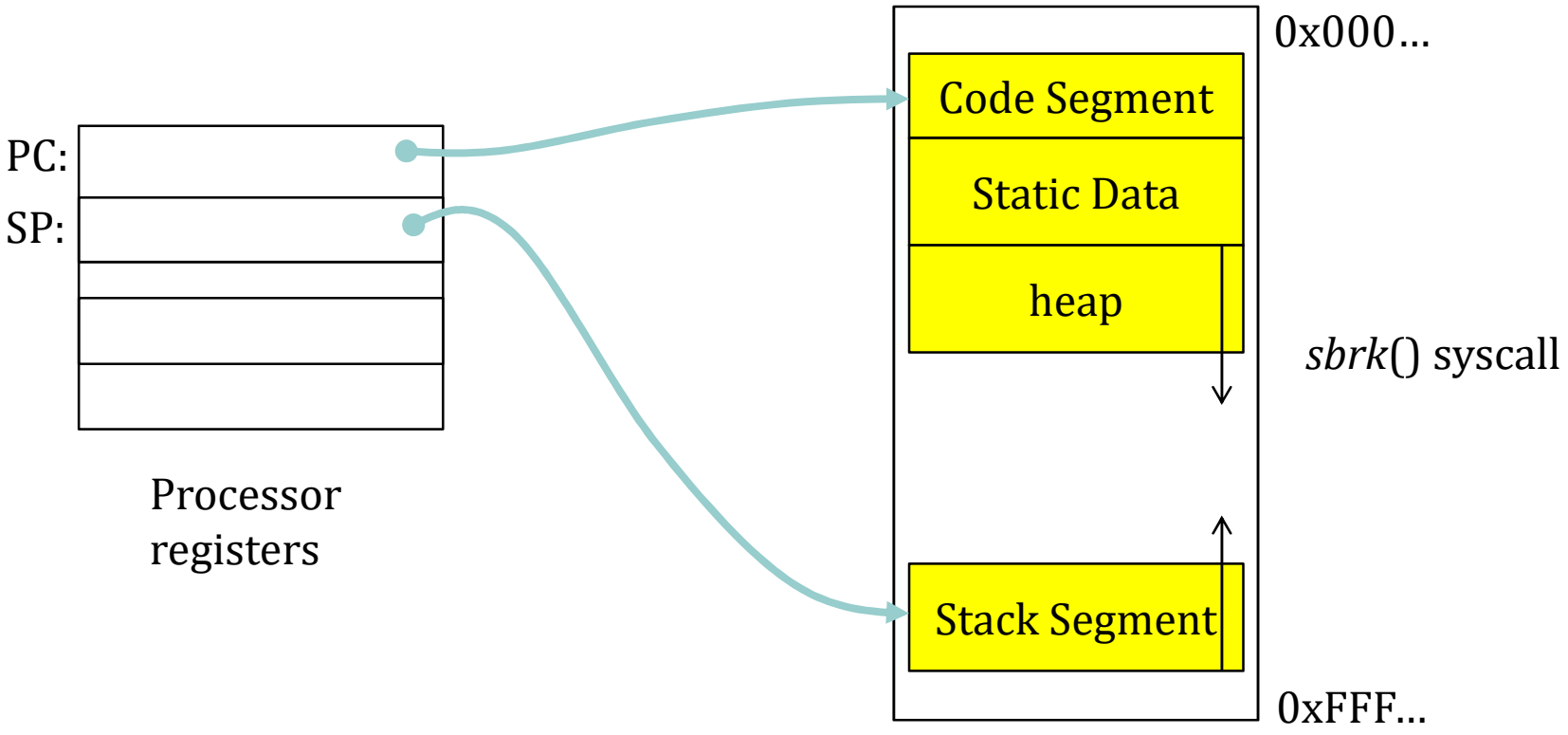


Address Space

- Definition
  - Set of accessible addresses and the state associated with them
  - $2^{32} \approx 4$ billion on a 32-bit machine
- What happens when you read or write to an address?
  - Perhaps acts like regular memory
  - Perhaps causes I/O operation
    - (Memory-mapped I/O)
  - Causes program to abort (segfault)?
  - Communicate with another program
  - …

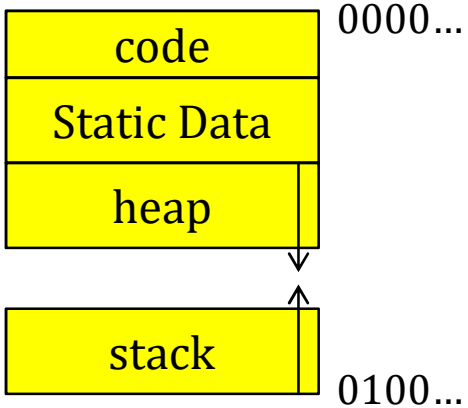
Address Space

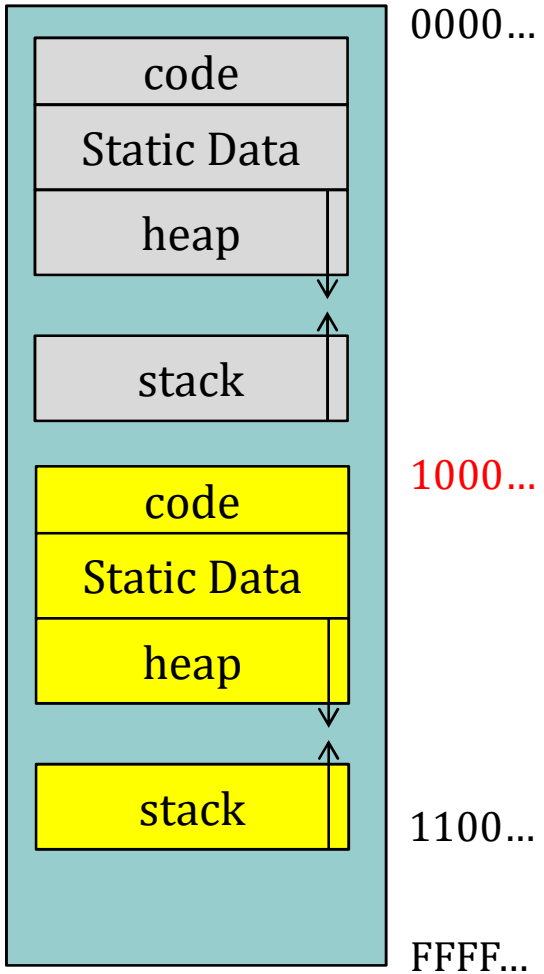
- Typical Structure.



Address Space

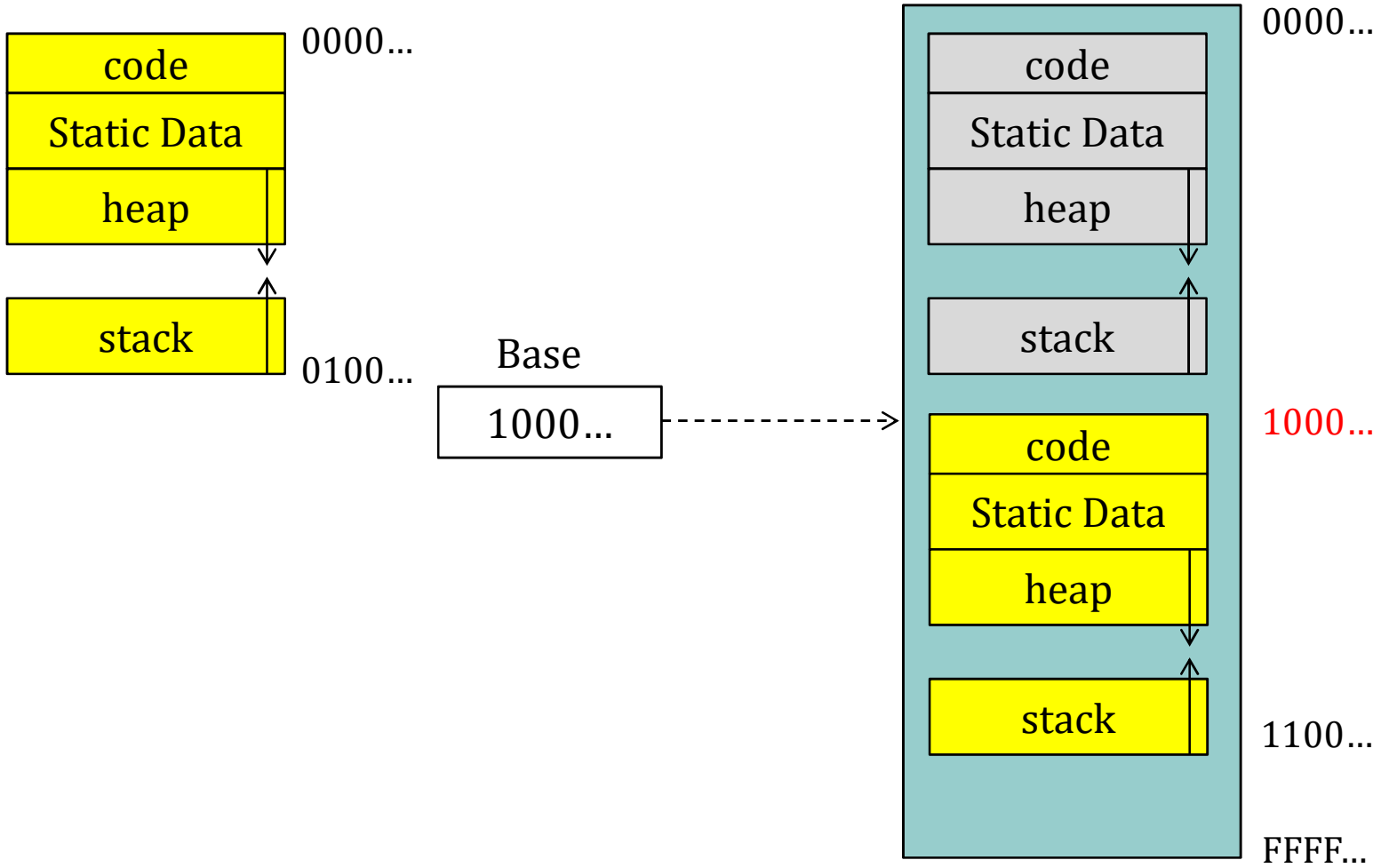
- Base and Bound - Protects OS and isolates program.





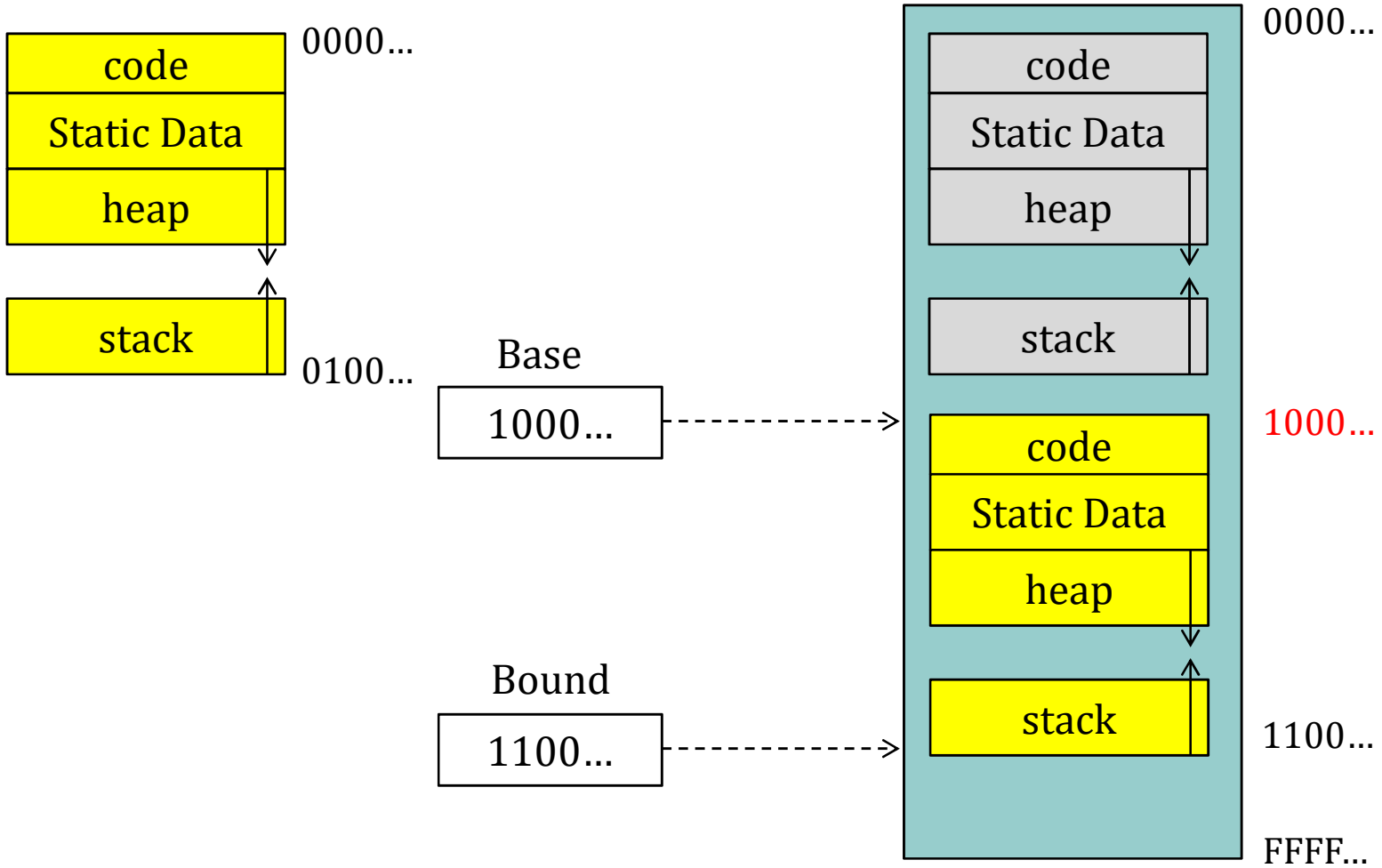
Address Space

- Base and Bound - Protects OS and isolates program.



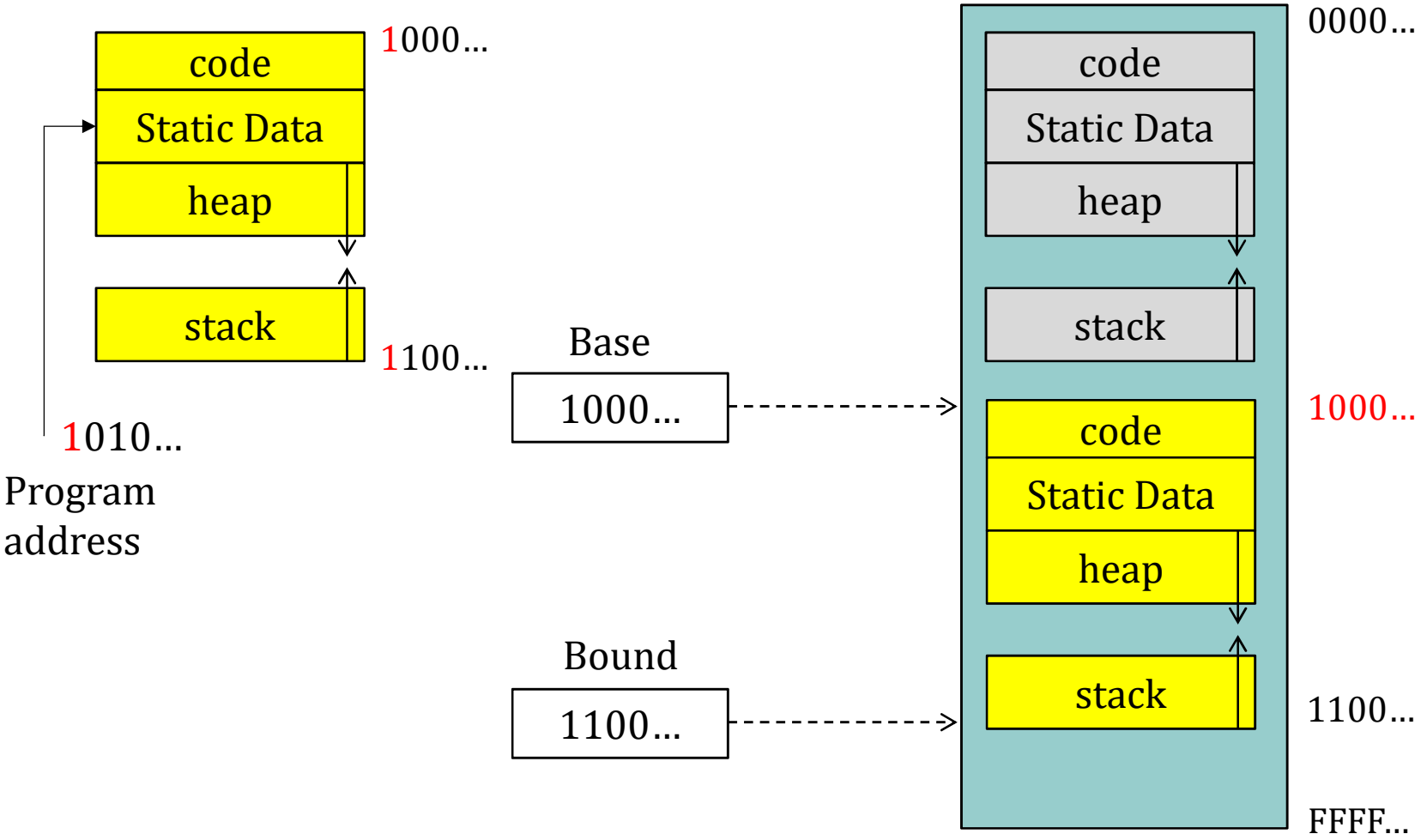
Address Space

- Base and Bound - Protects OS and isolates program.



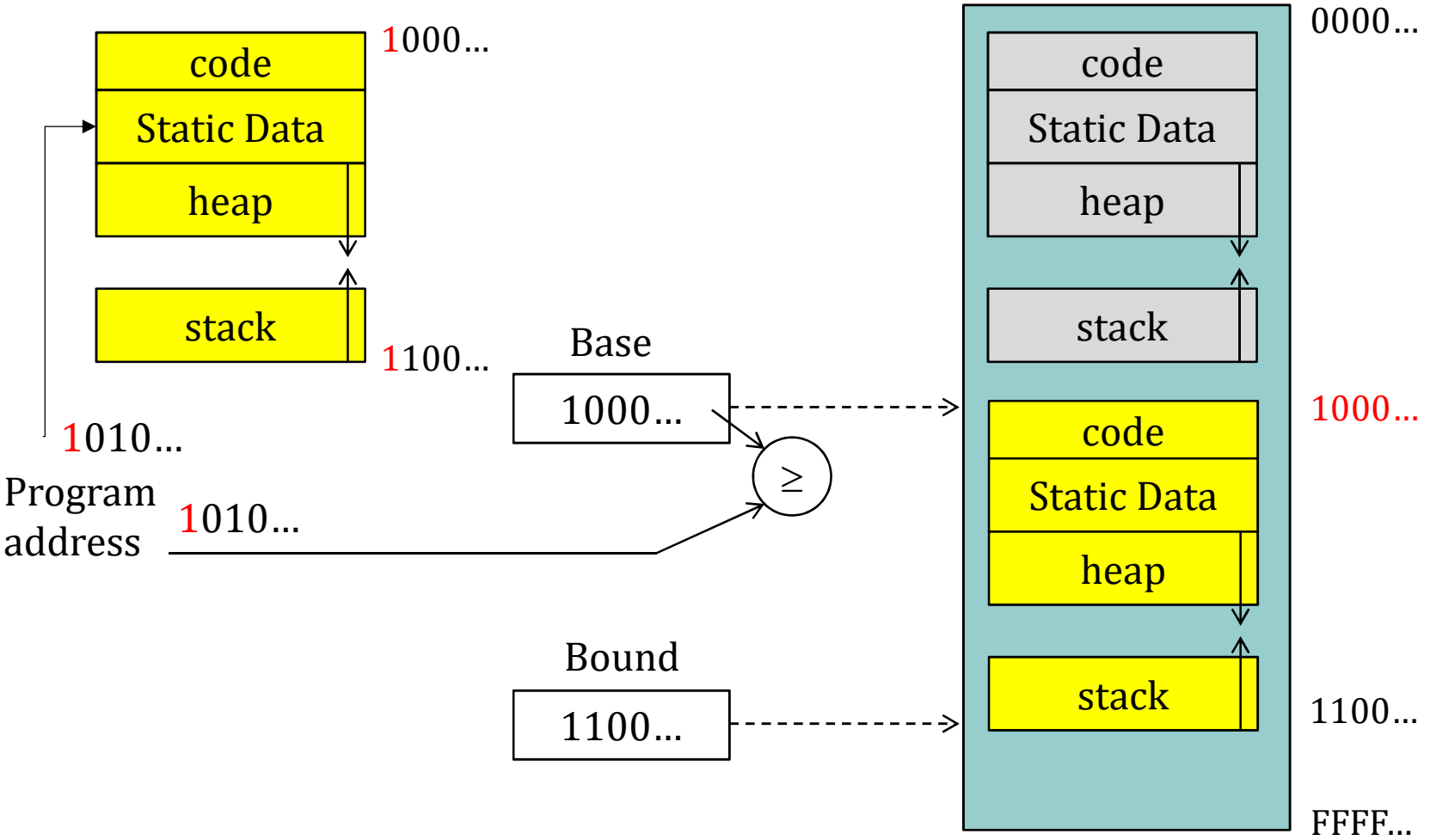
Address Space

- Base and Bound - Protects OS and isolates program.



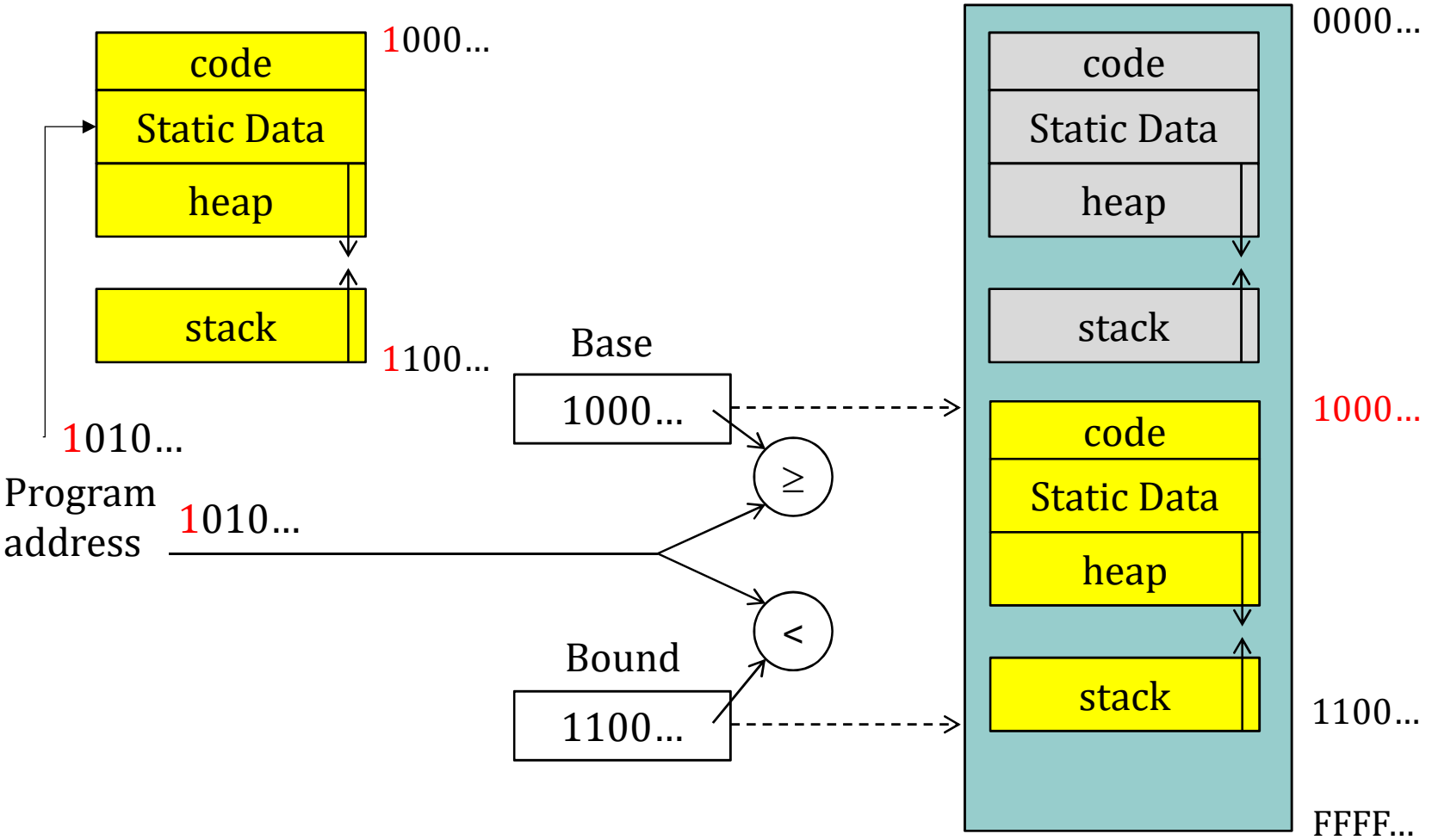
Address Space

- Base and Bound - Protects OS and isolates program.



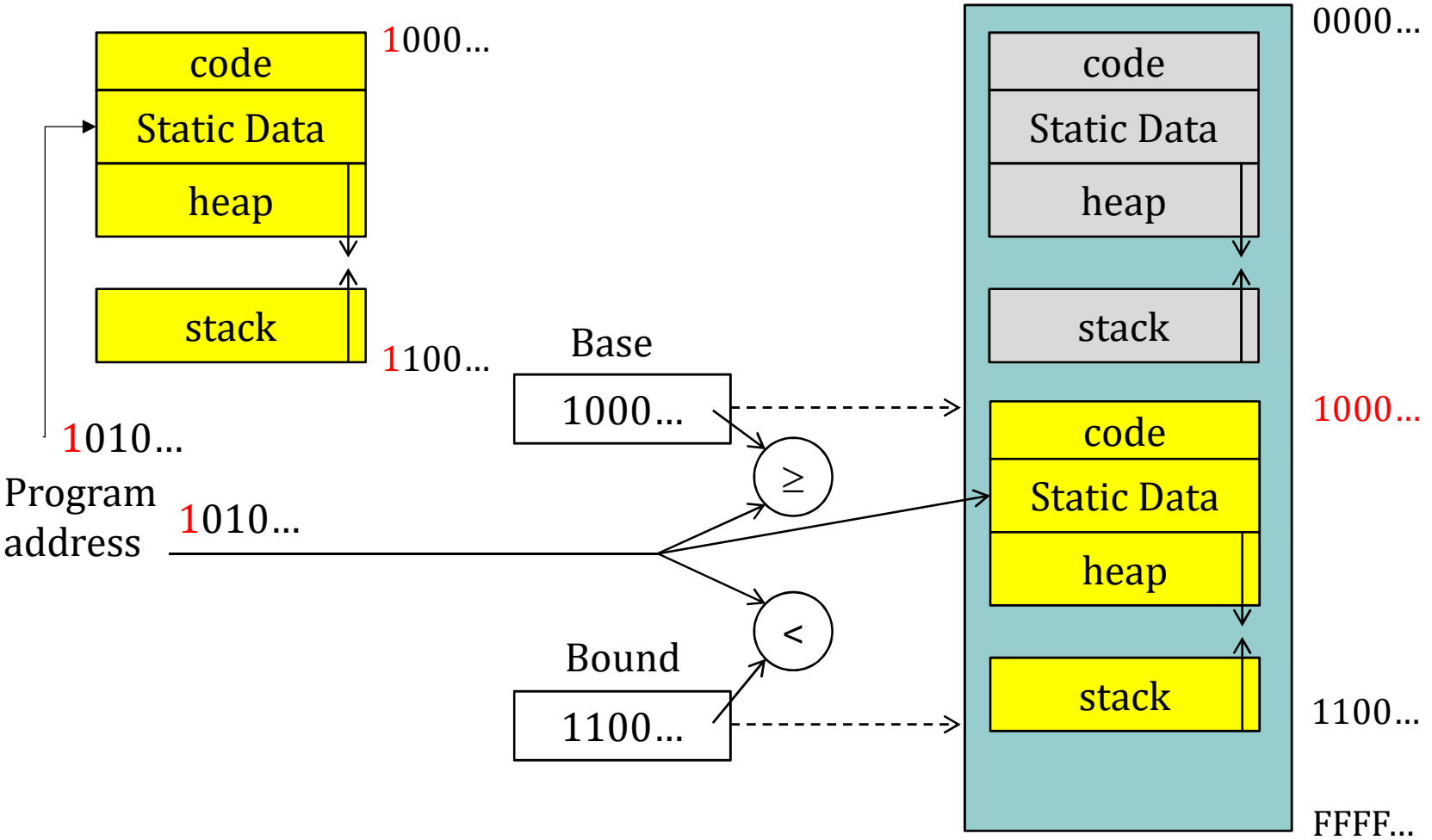
Address Space

- Base and Bound - Protects OS and isolates program.



Address Space

- Base and Bound - Protects OS and isolates program.



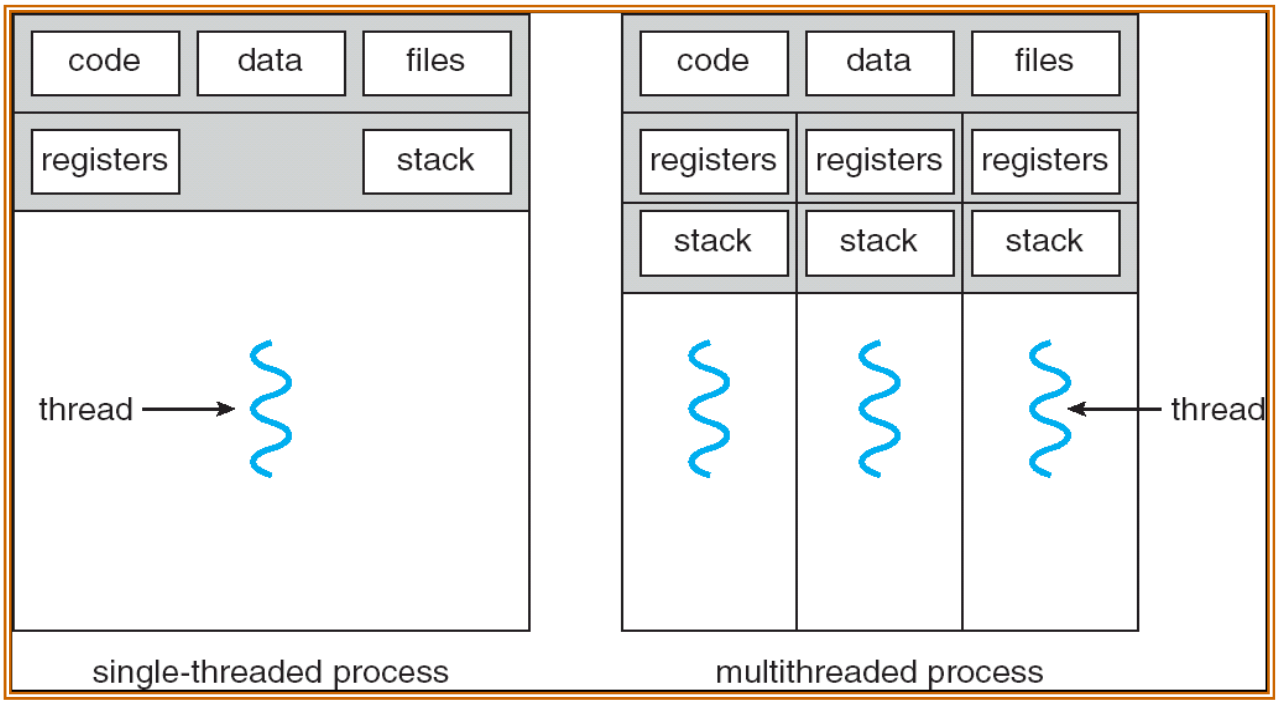
- Requires *relocation* in loading

Process

- Definition
  - execution environment with restricted rights
    - address Space with One or More Threads
    - owns memory (mapped pages)
    - owns file descriptors, file system context, …
    - encapsulates one or more threads sharing process resources
- Application program executes as a process
  - Complex applications can fork/exec child processes
- Why processes?
  - protected from each other. OS protected from them.
  - execute concurrently
  - basic unit OS deals with

Process

- Single and Multithreaded Processes.



- Threads encapsulate concurrency: “Active” component
- Address spaces encapsulate protection: “Passive” part
  - keeps buggy program from trashing the system
- Why have multiple threads per address space?

Process

- Typical memory layout of a process on Linux/IA-32.

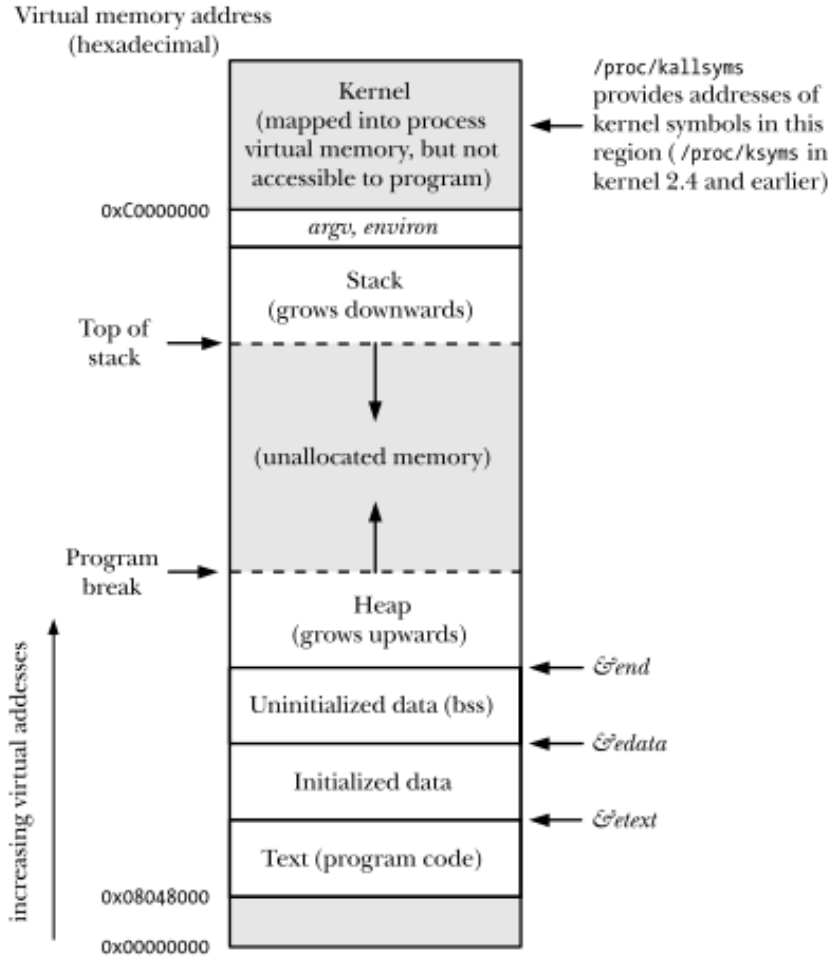
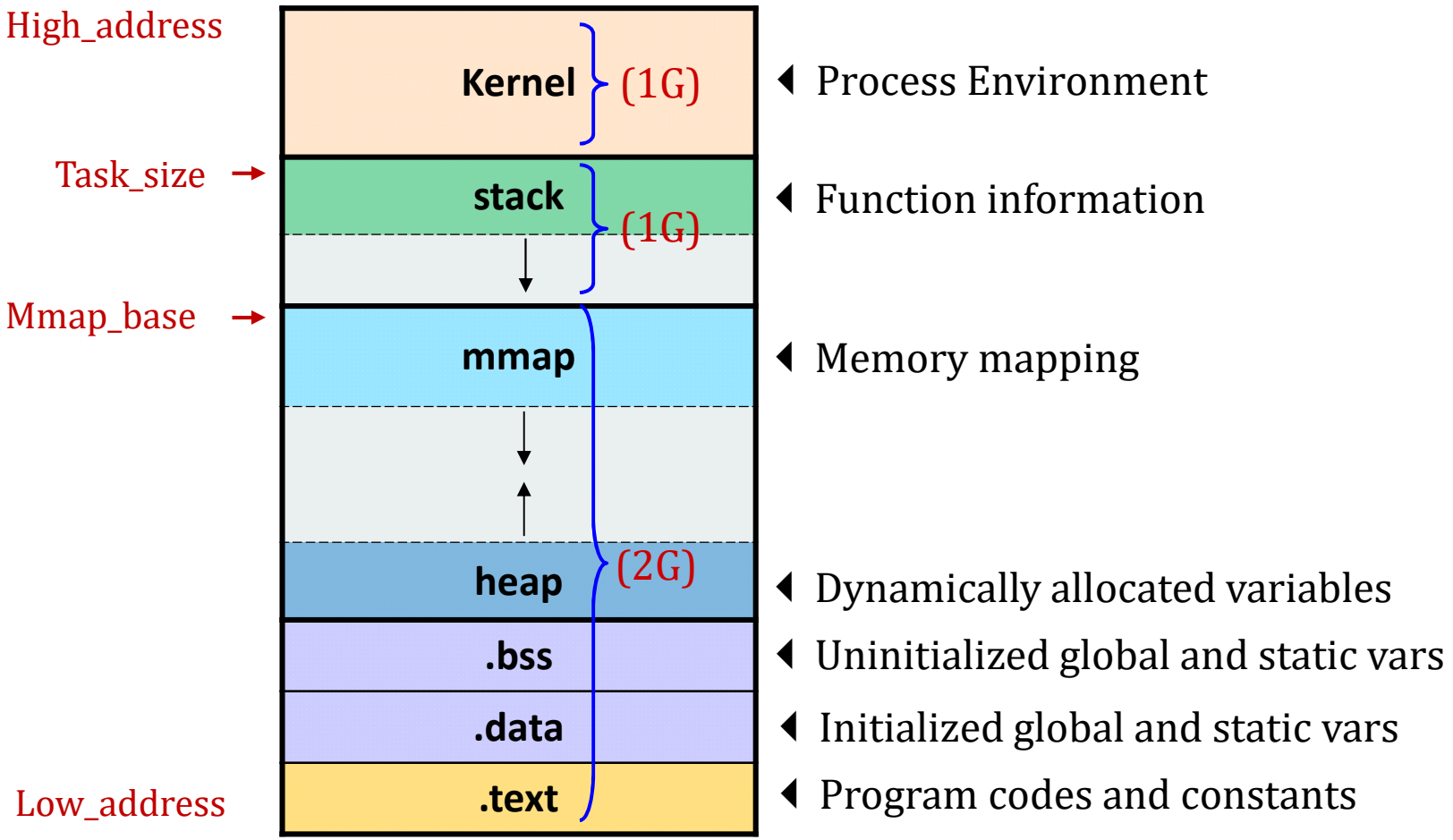


Figure 6-1: Typical memory layout of a process on Linux/x86-32

Process

- Typical memory layout of a process on Linux/IA-32.



Process Virtual Memory (PVM)

Process Management

- A process is a program *in execution.*
  - A program is a *passive* entity, in static state like the contents of a file stored on disk; a process is an *active* entity, in dynamic state with a system context.
- A process needs resources to accomplish its task.
  - CPU (registers), memory, I/O devices, files.
  - Data Initialization.
  - Process termination requires reclaim of any reusable resources.
- Process is a unit of work within the system. The operating system is responsible for the following activities in connection with process management:
  - *Creating* and *Deleting* both user and system processes.
  - *Suspending* and *Resuming* processes.
  - providing mechanisms for process *Synchronization*.
  - providing mechanisms for process *Communication*.
  - providing mechanisms for *Deadlock* handling.

Memory Management

- The main memory (*memory* in short) is central to the operation of a modern computer system. It is a repository of quickly accessible data shared by the CPU and I/O devices.
  - It is generally the only large storage device that the CPU is able to address and access directly.
  - To execute a program, all (or part) of the instructions and data needed must be in main memory.
- The operating system is responsible for the following activities in connection with memory management:
  - *keeping track* of which parts of memory are currently being used and by whom.
  - *deciding* when and which processes (or parts of processes) and data to move into and out of memory.
    - optimizing CPU utilization and computer response to users.
  - *Allocating* and *Deallocating* memory space as needed

File System Management

- Computers store information on several different types of physical media, each of these media has its own characteristics and physical organization.
  - The properties include *access speed*, *capacity*, *data-transfer rate*, and *access method* (sequential or random).
- A file is a collection of related information. Commonly, files represent programs (both source and object forms) and data.
- The operating system implements the abstract concept of a file by managing mass storage media and the devices that control them.
- Files are normally organized into *directories*. *Access control* is applied on most file systems to determine *who can access what*.
- File management activities for operating systems:
  - *Creating* and *Deleting* files
  - *Creating* and *Deleting* directories
  - supporting *Primitives* for manipulating files and directories
  - *Mapping* files onto mass storage
  - *Backing up* files on stable (nonvolatile) storage media.

Mass-Storage Management

- Usually mass-storages like HDDs and other NVM devices are used to store data that does not fit in main memory or data that must be kept for a “long” period of time.
  - Entire speed of computer operation may hinge on disk subsystem and its algorithms.
  - Proper management of these *secondary storage* is of central importance.
- Some storage may need not be fast:
  - Tertiary storage includes optical storage, magnetic tape.
  - Varies between WORM (write-once, read-many-times) and RW (read-write)
- Mass-storage management activities for operating systems:
  - *Mounting* and *Unmounting*
  - *Free-space* management
  - storage *Allocation*
  - disk *Scheduling*
  - *Partitioning*
  - *Protection*.

Cache Management

- Caching is an important principle of computer systems. Because caches have limited size, cache management is an important design problem.
- In a multiprocessor environment, cache management becomes more complicated. In addition to maintaining internal registers, each of the CPUs also contains a local cache.
  - Cache Coherency (缓存一致性)
    - E.g., a copy of an integer A may exist simultaneously in several caches. Since the various CPUs can all execute in parallel, It must be sure that an update to the value of A in one cache is immediately reflected in all other caches where A resides. This *cache coherency* is usually a hardware issue (handled below the operating-system level).
- In a distributed environment, this situation becomes even more complex. Several copies (or replicas) of the same file can be kept on different computers. Since the various replicas may be accessed and updated concurrently, some distributed systems ensure that, when a replica is updated in one place, all other replicas are brought up to date as soon as possible.

I/O System Management

- One purpose of an OS is to hide the peculiarities of specific hardware devices from the user.
- I/O subsystem consists of several components:
  - Memory management of I/O including
    - *Buffering*
      - storing data temporarily while it is being transferred.
    - *Caching*
      - storing parts of data in faster storage for performance.
    - *SPOOLing*
      - the overlapping of output of one job with input of other jobs.
  - General device-driver interface.
    - *Device drivers* for specific hardware devices
      - Only the device driver knows the peculiarities of the specific device to which it is assigned.

Free and Open-Source Operating Systems

- Both *free operating systems* and *open-source operating systems* are available in source-code format.
- Free software (aka *libre software* 自由软件) not only makes source code available but also is licensed to allow no-cost use, redistribution, and modification. But some open-source software is not “free.”
  - GNU/Linux is the most famous open-source operating system, with some distributions free and others open source only.
  - Microsoft Windows is a well-known example of the opposite *closed-source* approach. Windows is *proprietary* software (专利软件) —Microsoft owns it, restricts its use, and carefully protects its source code.
  - Apple’s MacOS operating system comprises a hybrid approach. It contains an open-source kernel named Darwin but includes proprietary, closed-source components as well.

Free and Open-Source Operating Systems

- Learning operating systems by examining the source code
  - Student can modify the source code of an operating system, then compile and run the object code to try out those changes, which is an excellent learning tool.
- Open-source Community
  - A community of interested, unpaid programmers contribute to the open-source code by helping to write it, debug it, analyze it, provide support, and suggest changes.
  - Arguably, open-source code is more secure than closed-source code because many more eyes are viewing the code. Bugs of open-source code tend to be found and fixed faster owing to the number of people using and viewing the code.

GNU Project

- *Richard M. Stallman* started developing a free, UNIX-compatible operating system called GNU (which is a recursive acronym for “GNU’s Not Unix!”) in 1984 and published the GNU Manifesto in 1985. He also formed the *Free Software Foundation* (FSF) with the goal of encouraging the use and development of free software.
- The *GNU General Public License* (GPL) is a common license under which free software is released. Fundamentally, the GPL requires that the source code be distributed with any binaries and that all copies (including modified versions) be released under the same GPL license.

GNU/Linux

- The GNU kernel never became ready for prime time. In 1991, a student in Finland, *Linus Torvalds*, released a rudimentary UNIX-like kernel using the GNU compilers and tools and invited contributions worldwide.
  - The advent of the Internet meant that anyone interested could download the source code, modify it, and submit changes to *Torvalds*.
- In 1992, *Torvalds* rereleased Linux under the GPL, making it free software.
- The resulting GNU/Linux operating system (with the kernel properly called Linux but the full operating system including GNU tools called GNU/Linux) has spawned hundreds of unique distributions, or custom builds, of the system. Major distributions include:
  - Red Hat
  - SUSE
  - Fedora
  - Debian
  - Slackware
  - Ubuntu.

BSD UNIX

- BSD UNIX Releases came from the University of California at Berkeley (UCB) in 1978 as a derivative of AT&T's UNIX with a license from AT&T required.
  - 4.4BSD-lite, released in 1994, was its fully functional, open-source version.
- There are many distributions of BSD UNIX, including FreeBSD, NetBSD, OpenBSD, and DragonflyBSD.
  - FreeBSD source code online at *https://svnweb.freebsd.org*.
- Darwin, the core kernel component of macOS, is based on BSD UNIX and is open-sourced as well.
  - Darwin source code online at *http://www.opensource.apple.com/*.