Genesis: From Raw Hardware to Processes

Sarah Diesburg

Operating Systems

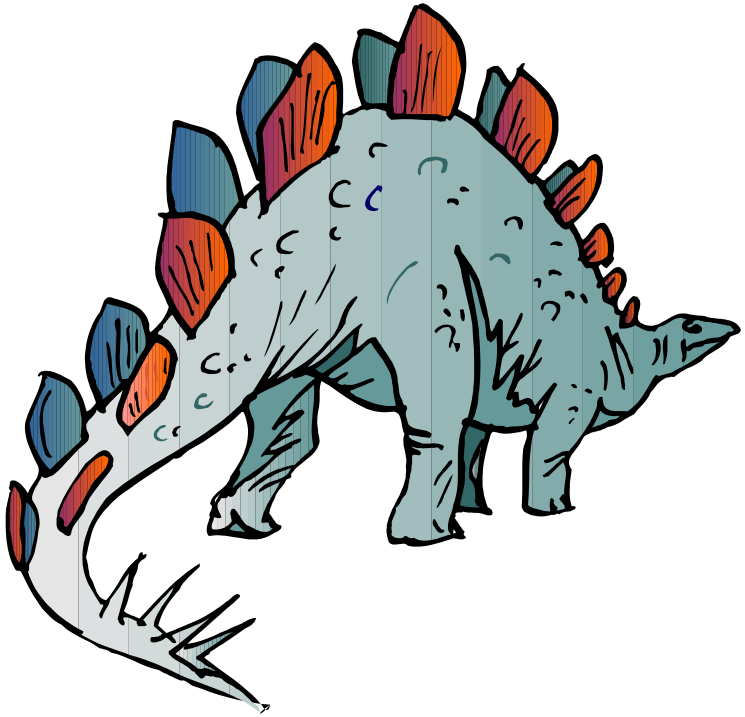
CS 3430

How is the first process created?

- What happens when you turn on a computer?
- How to get from raw hardware to the first running process, or ***process 1*** under UNIX?
- Well…it's a long story…
  - It starts with a simple computing machine

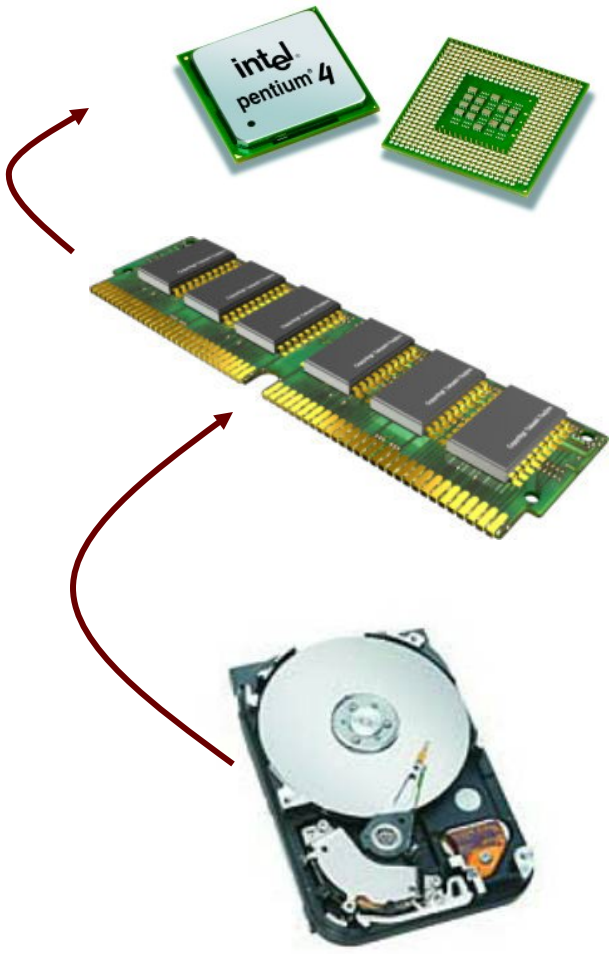
Long, Long, Long Ago…(during the 1940s)

- John von Neumann invented ***von Neumann computer architecture***
  - A CPU
  - A memory unit
  - I/O devices (e.g., disks and tapes)



In von Neumann Architecture

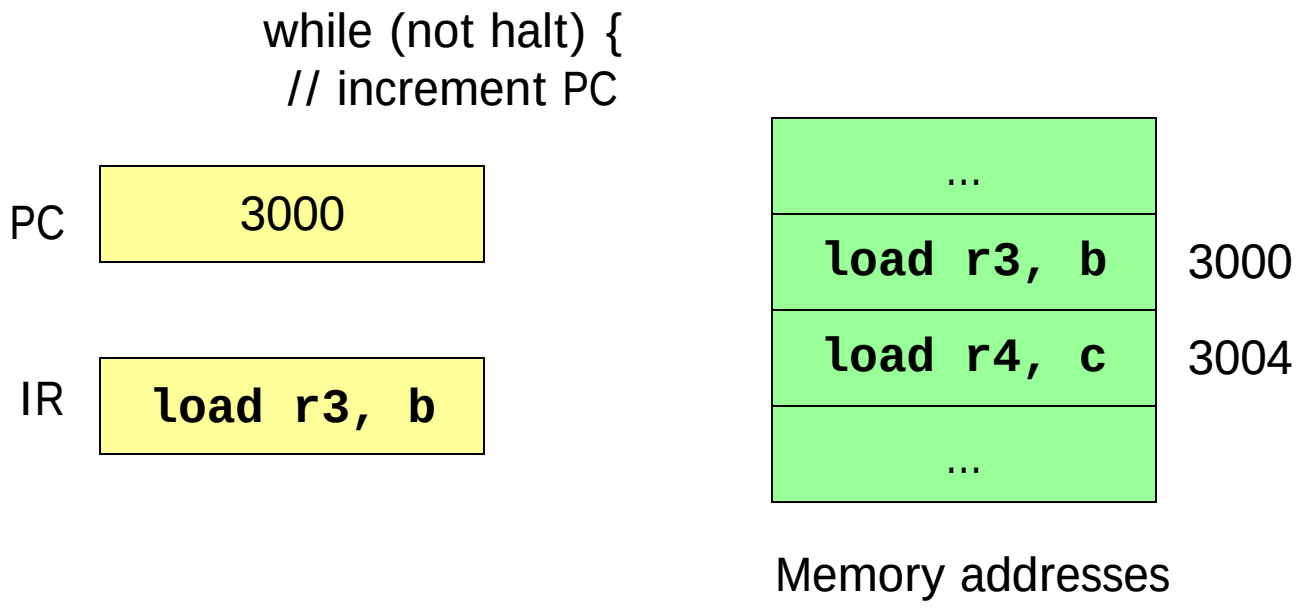
- Programs are stored on storage devices
- Programs are copied into memory for execution
- CPU reads each instruction in the program and executes accordingly



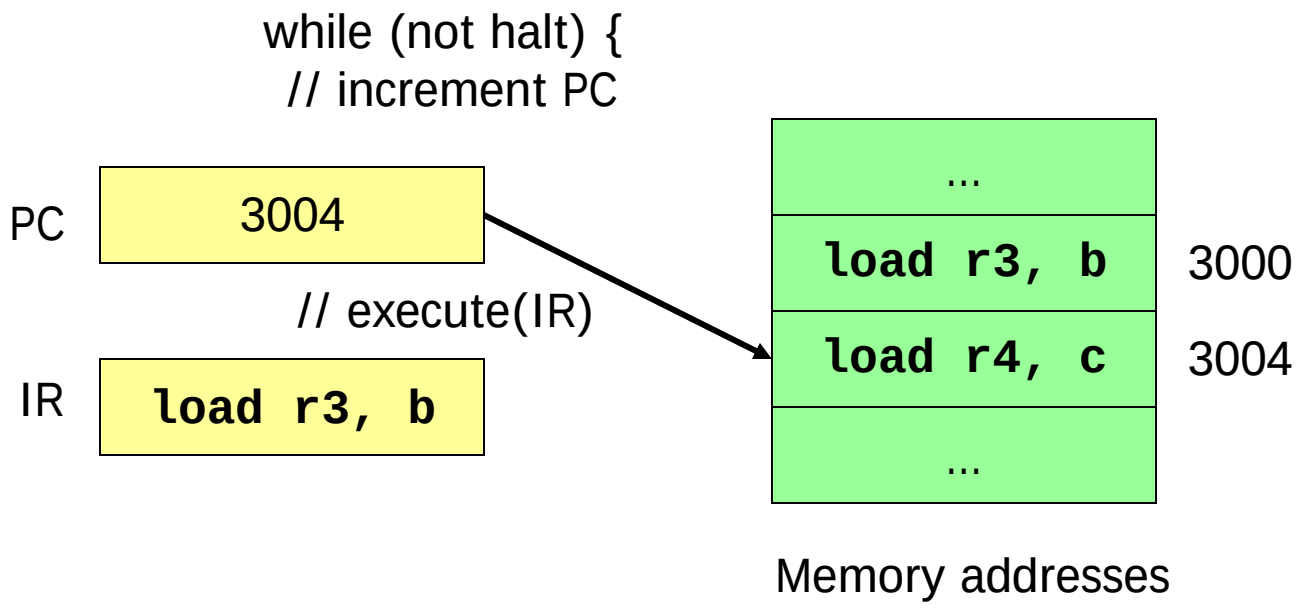
A Simple CPU Model

- ***Fetch-execute algorithm***
- During a boot sequence, the program counter (PC) is loaded with the address of the first instruction
- The instruction register (IR) is loaded with the instruction from the address

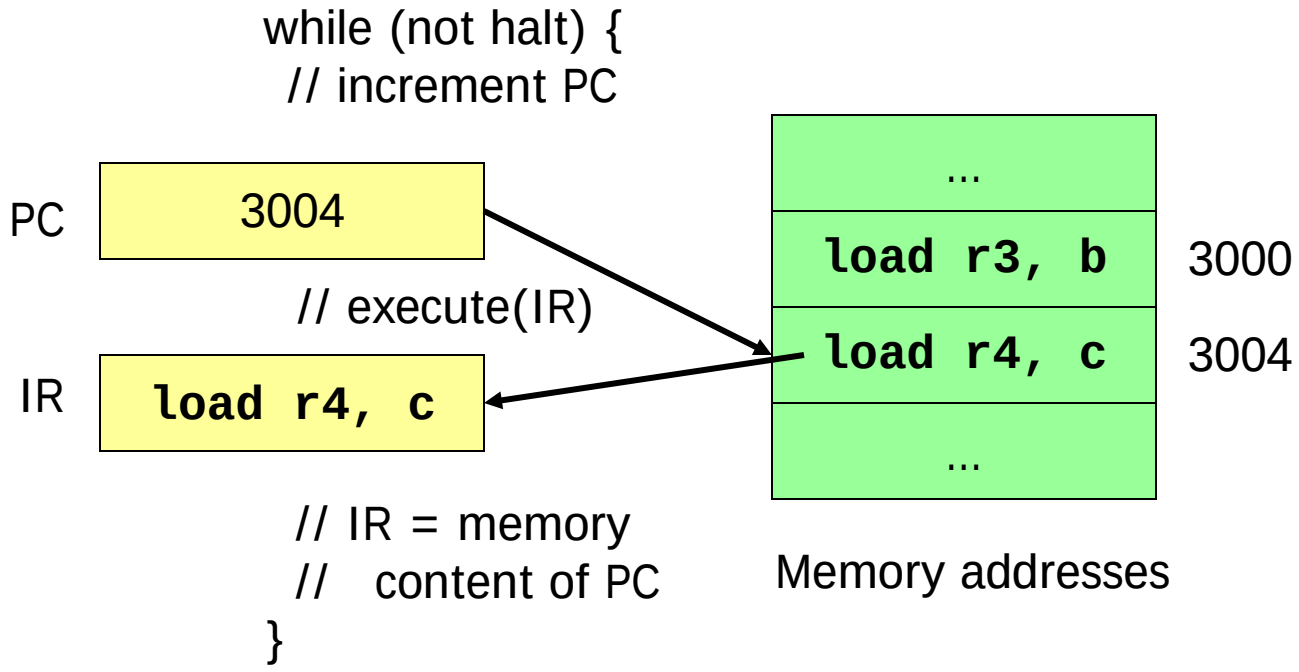
Fetch-Execute Algorithm



Fetch-Execute Algorithm



Fetch-Execute Algorithm



Booting Sequence

- The address of the first instruction is fixed
- It is stored in read-only-memory (ROM)
  - Why ROM instead of RAM?

Booting Procedure

- ROM stores a ***Basic Input/Output System (BIOS)***
  - BIOS contains information on how to access storage devices

BIOS Code

- Performs Power-On Self Test (POST)
  - Checks memory and devices for their presence and correct operations
  - During this time, you will hear memory counting, which consists of noises from the floppy and hard drive, followed by a final beep

After the POST

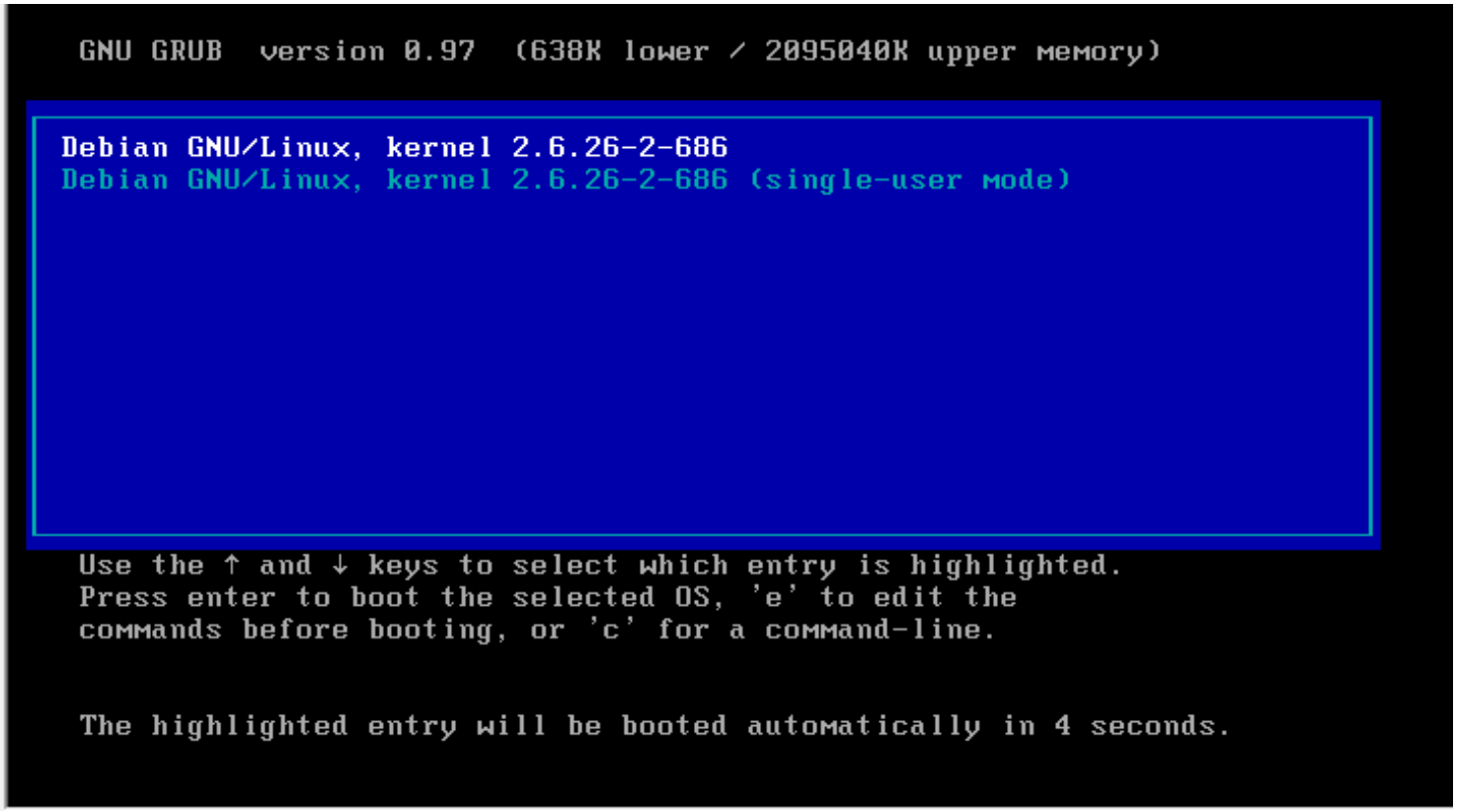
- The ***master boot record (MBR)*** is loaded from the ***boot device*** (configured in BIOS)
- The MBR is stored at the first logical sector of the boot device (e.g., a hard drive) that
  - Fits into a single 512-byte disk sector (***boot sector***)
  - Describes the physical layout of the disk (e.g., number of tracks)

After Getting the Info on the Boot Device

- BIOS loads a more sophisticated loader from other sectors on disk
- The more sophisticated loader loads the operating system

Operating System Loaders

- GRUB (*GRand Unified Bootloader*)



More on OS Loaders

- Is partly stored in MBR with the disk partition table
  - A user can specify which disk partition and OS image to boot
  - Windows loader assumes only one bootable disk partition
- After loading the kernel image, OS loader sets the kernel mode and jumps to the entry point of an operating system

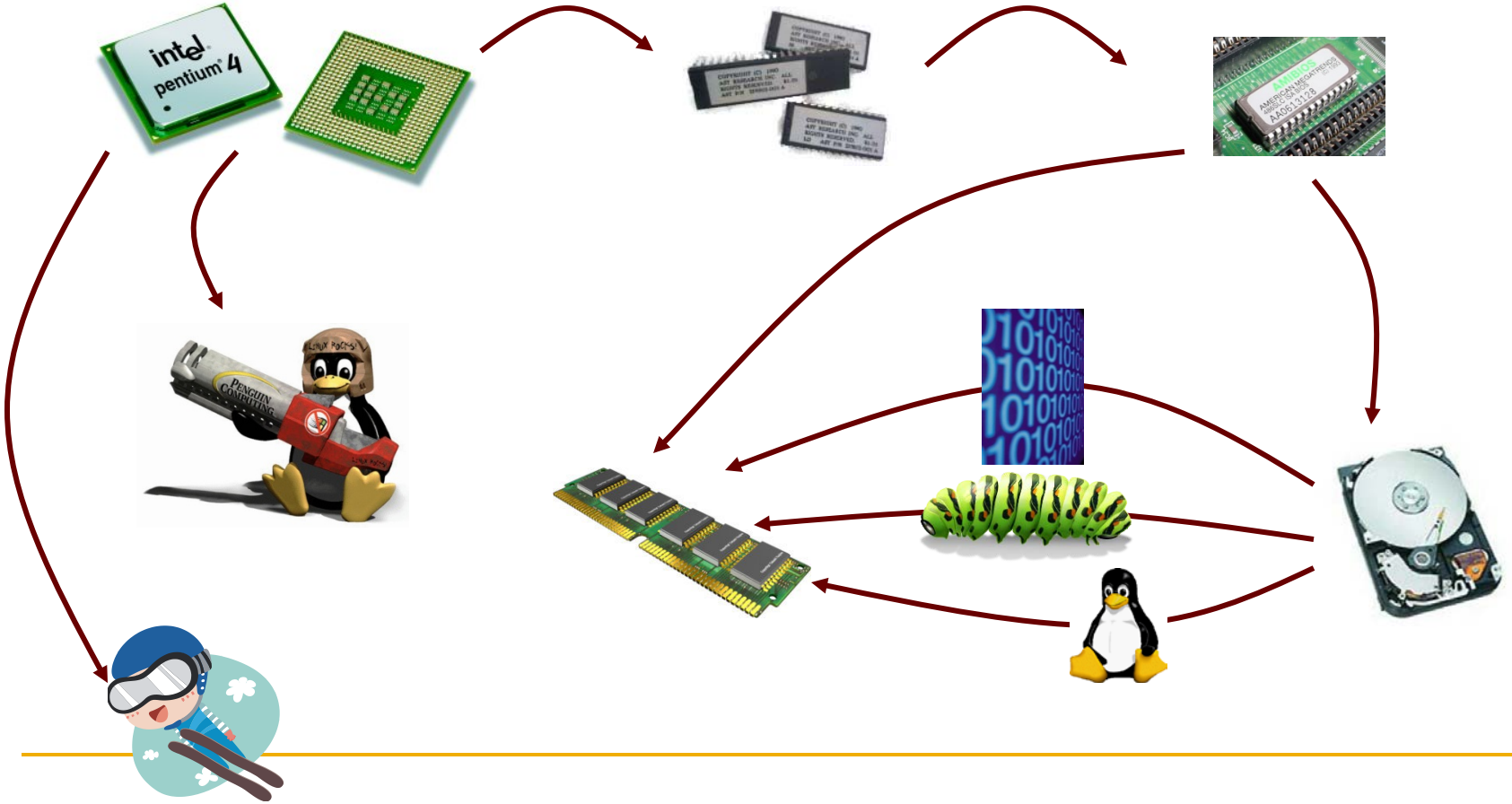
Kernel Mode?

- Two hardware modes: kernel mode and user mode
  - Implemented as a single bit
  - Some privileged instructions can only be run in kernel mode to protect OS from errant users
  - Operating system must run in kernel mode

Booting Sequence in Brief

- A CPU jumps to a fixed address in ROM,
- Loads the BIOS,
- Performs POST,
- Loads MBR from the boot device,
- Loads an OS loader,
- Loads the kernel image,
- Sets the kernel mode, and
- Jumps to the OS entry point.

Booting Sequence Visualized



Linux Initialization

- Set up a number of things:
  - Trap table
  - Interrupt handlers
  - Scheduler
  - Clock
  - Kernel modules (hardware and software drivers)
  - …
  - Process manager

Process 1

- Is instantiated from the ***init*** program
- Is the ancestor of all processes
- Controls transitions between ***runlevels***
- Executes startup and shutdown scripts for each runlevel

Runlevels

- Level 0: shutdown
- Level 1: single-user
- Level 2: multi-user (without network file system)
- Level 3: full multi-user
- Level 5: X11 (the GUI)
- Level 6: reboot

Process Creation

- Via the ***`fork`*** system call family

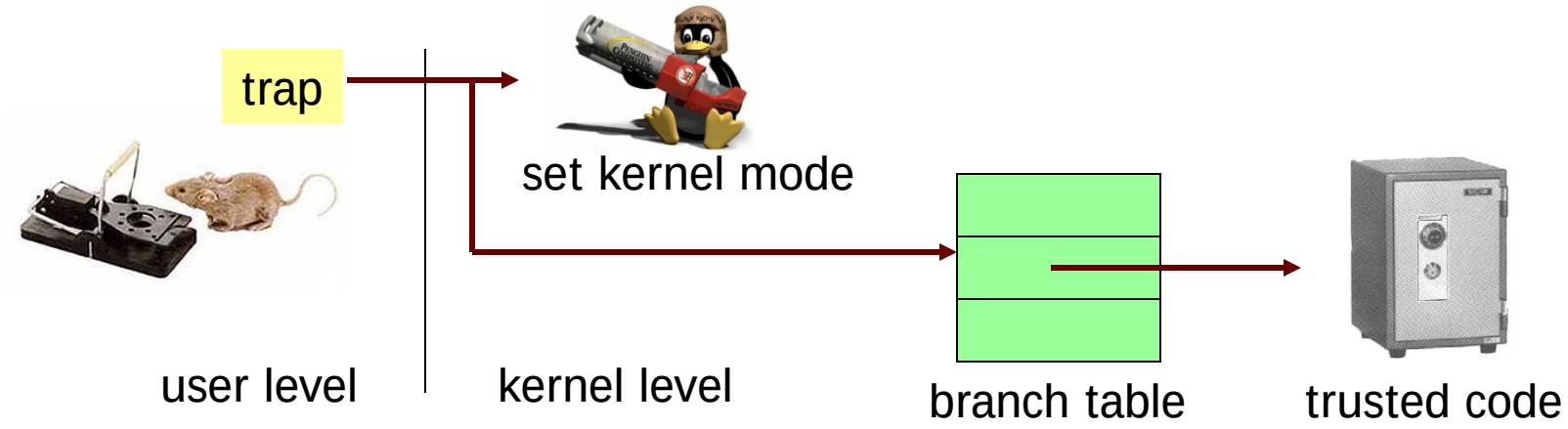
Before we discuss process creation, a few words on system calls…

System Calls

- ***System calls*** allow processes running at the ***user mode*** to access kernel functions that run under the ***kernel mode***
- Prevent processes from doing bad things, such as
  - Halting the entire operating system
  - Modifying the MBR

UNIX System Calls

- Implemented through the ***trap*** instruction



More on Fork

- ***Fork*** is a system call to create a new process
  - What does each process have (two things)??
- Two processes may be bulky
  - Can create multiple ***threads*** instead

Thread Creation

- Use **`pthread_create()`** instead of **`fork()`**
- A newly created thread will share the address space of the current process and all resources (e.g., open files)

+ Efficient sharing of states

- Potential corruptions by a misbehaving thread