OS Lab
Session 3: Process

AUT – CEIT

Instructor: @MerajNouredini

What is Process?

What is Process

- **Processes** carry out **tasks** within the operating system. A program is a set of machine code **instructions** and **data** stored in an executable image on disk and is, as such, a **passive entity**; a process can be thought of as a computer program in action

What is Process

- It is a **dynamic entity**, constantly changing as the machine code instructions are executed by the processor.
- the process also includes the **program counter** and all of the **CPU's registers** as well as the process **stacks** containing temporary data such as **routine parameters**, **return addresses** and **saved variables**.

What is Process

- During the lifetime of a process it will use many system **resources**
  - CPU
  - Physical Memory
  - Files within the filesystems
  - Other physical devices in the system

Process in Linux

- Linux is a **multiprocessing operating system**. Processes are **separate tasks** each with their own rights and responsibilities. If one process crashes it will not cause another process in the system to crash. Each individual process runs in **its own virtual address space** and is not capable of interacting with another process except through secure, **kernel managed mechanisms**.

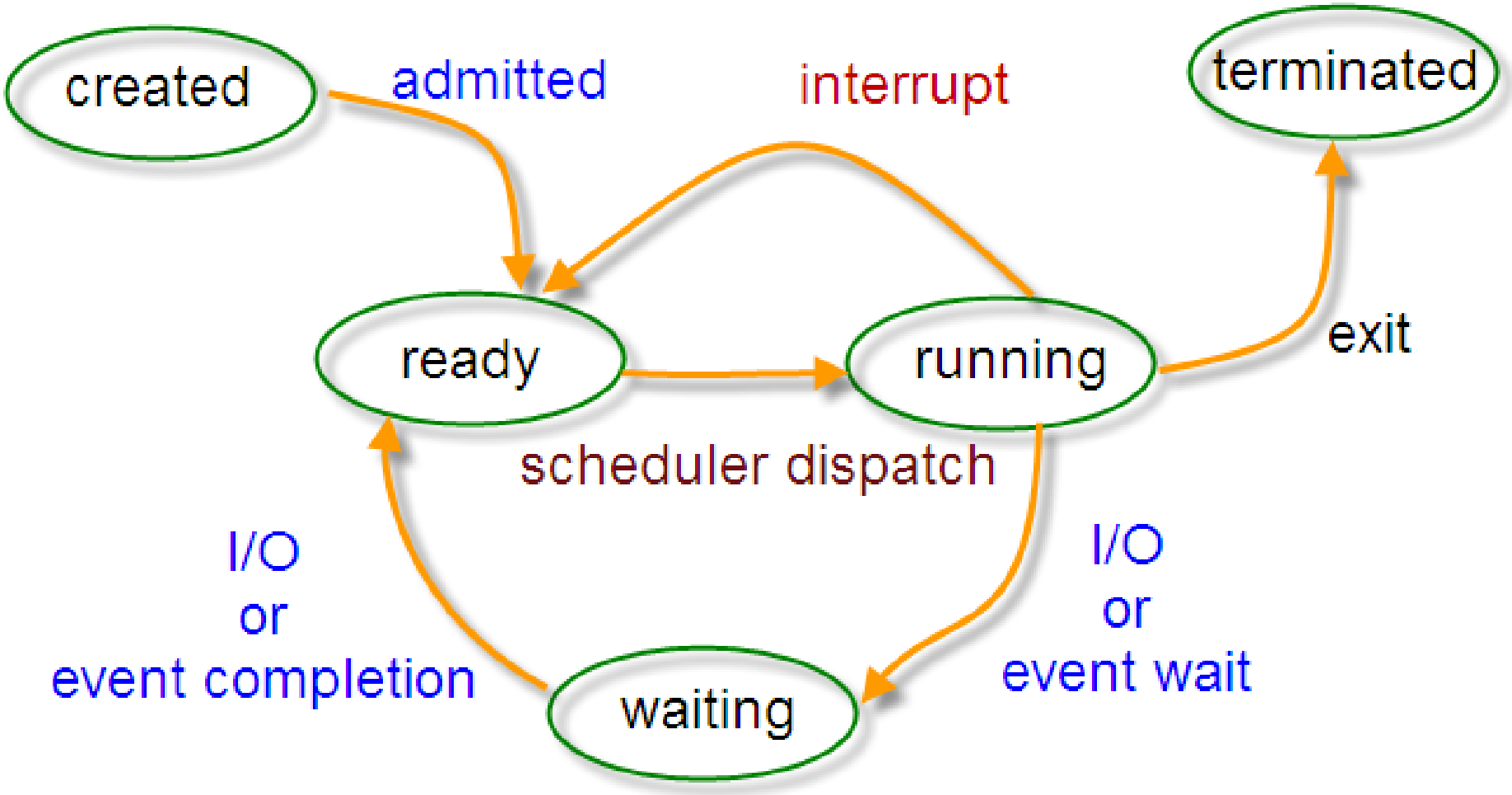
Process in Linux

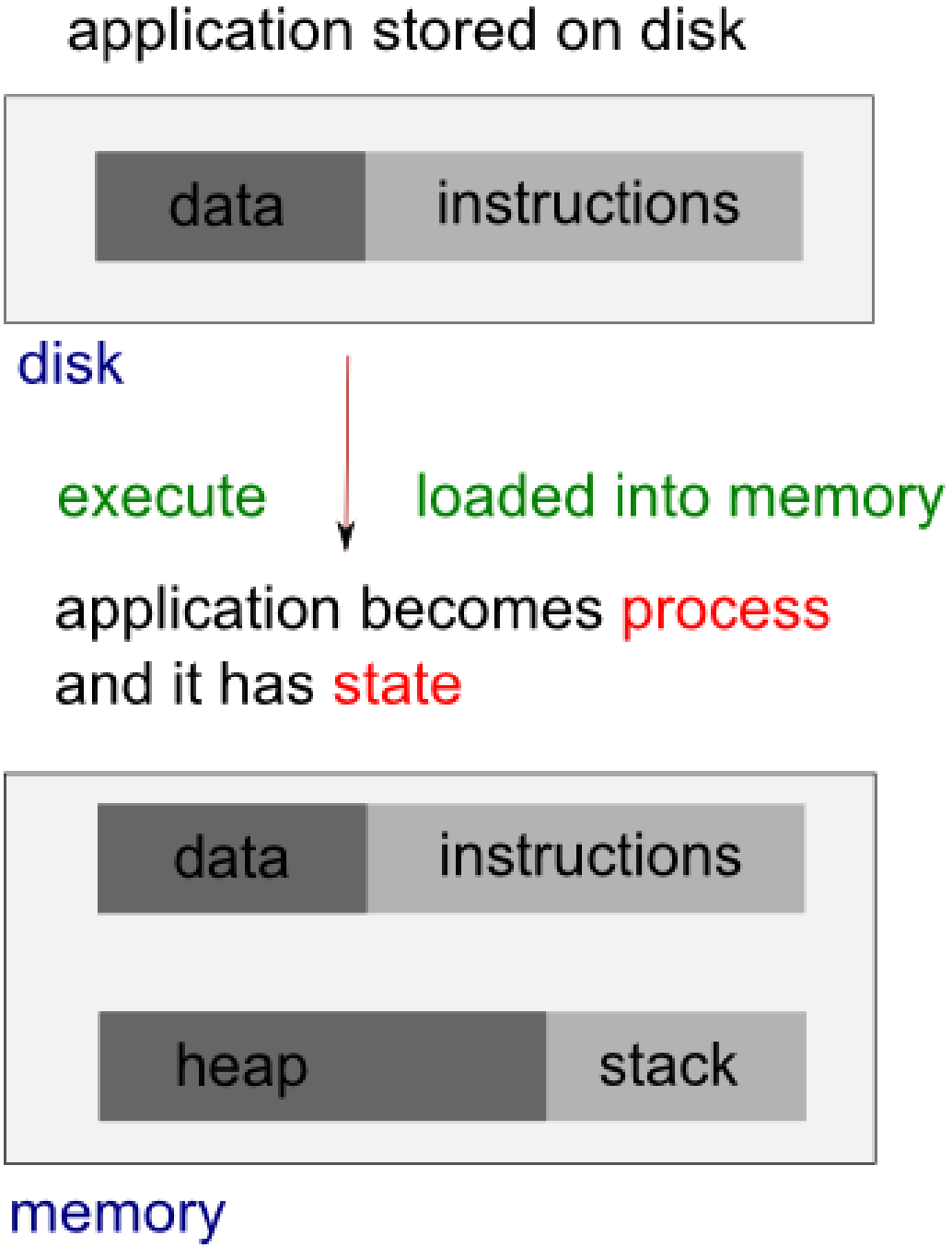
- In order to manage the processes in the system, each process is represented by a **`task_struct`** data structure
- The `task_struct` data structure is quite large and complex, but its fields can be divided into a number of functional areas:

Process in Linux

- State
  - Running
  - Waiting
  - Stopped
  - Zombie
- Scheduling Information
- Identifiers

Process State




application stored on disk
data
instructions
disk
execute
loaded into memory
application becomes process
and it has state
data
instructions
heap
stack
memory

Process in Linux

- Inter-Process Communication
- Links (pstree)
- Times and Timers (jiffies)
- File system
- Virtual memory
- Processor Specific Context

Signals

Signals

- Signals are a limited form of inter-process communication (**IPC**), typically used in Unix, Unix-like, and other POSIX-compliant operating systems.
- When a signal is sent, the operating system **interrupts** the target process' normal flow of execution to deliver the signal.
- Linux kernel implements about 30 signals
  - https://en.wikipedia.org/wiki/Signal_(IPC)

Signals

- For instance **SIGKILL** or signal number 9 tells the program that someone tries to kill it, and **SIGHUP** used to signal that a terminal hangup has occurred, and it has a value of 1 on the i386 architecture.
- With the exception of **SIGKILL** and **SIGSTOP** which always terminates the process or stops the process, respectively, processes may **control what happens** when they receive a signal.

Signals

- Signals can be:
  - Raised
  - Caught
  - Acted upon
  - Ignored

- The term **raise** is used to indicate the generation of a signal, and the term **catch** is used to indicate the receipt of a signal.

- If a process receives signals such as SIGFPE, SIGKILL, etc., the process will be terminated immediately, and a **core dump file is created.** The core file is an image of the process, and we can use it to **debug**.

Some Commands

top

- One of the most basic command to monitor processes on Linux
- It shows the top processes based on certain criterias like cpu usage or memory usage.

ps

- The ps (i.e., process status) command is used to provide information about the currently running processes.
- PID, TTY, TIME, CMD, VSZ, RSS

ps – State codes

- D uninterruptible sleep (usually IO)
- R running or runnable (on run queue)
- S interruptible sleep (waiting for an event to complete)
- T stopped by job control signal
- t stopped by debugger during the tracing
- W paging (not valid since the 2.6.xx kernel)
- X dead (should never be seen)
- Z defunct ("zombie") process, terminated but not reaped by its parent

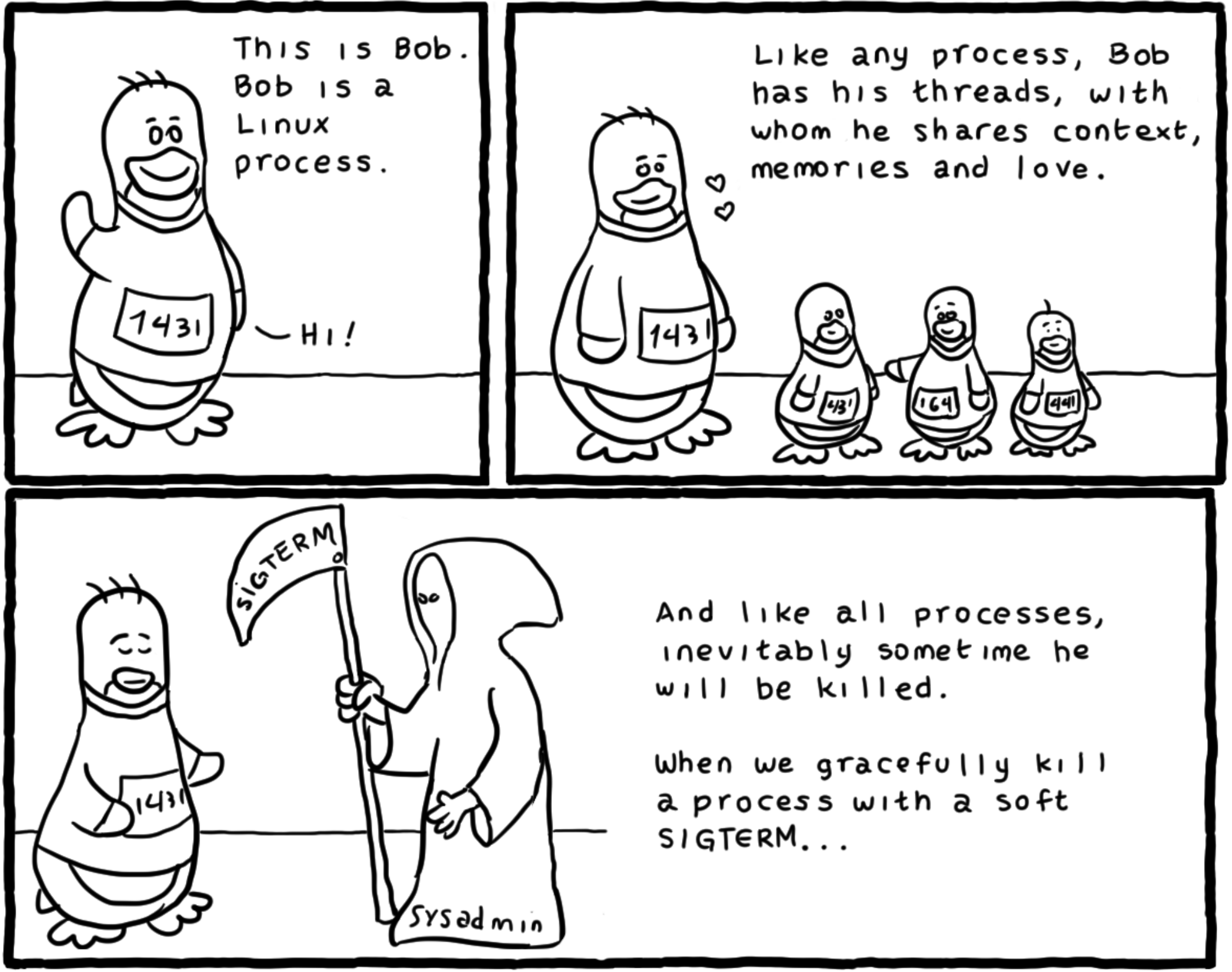
- < high-priority (not nice to other users)
- N low-priority (nice to other users)
- L has pages locked into memory (for real-time and custom IO)
- s is a session leader
- l is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
- + is in the foreground process group

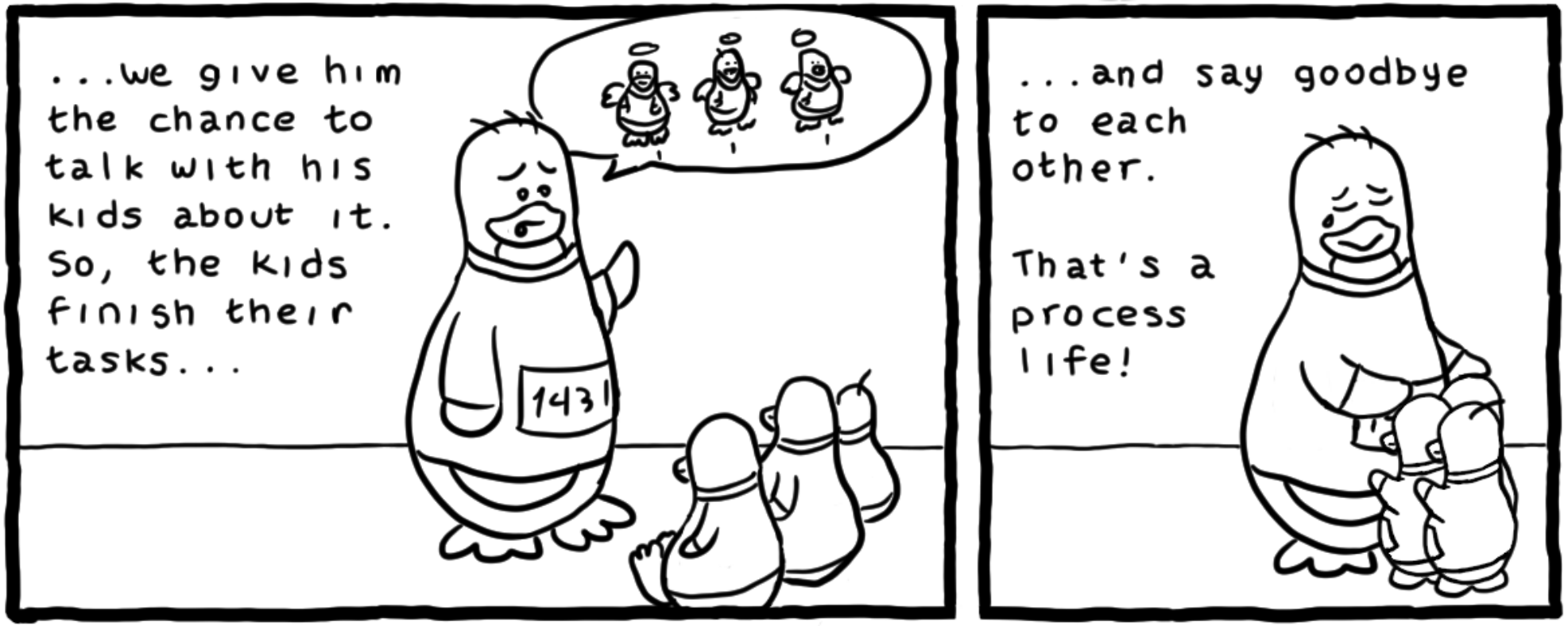
nice

- Linux kernel uses a **process scheduler** to decide which process will get the next **time slice** based on the process **priority**
- Well-behaved programs are termed **nice** programs, and in a sense this niceness can be measured.
- A niceness of -20 is the highest priority and 19 or 20 is the lowest priority.

kill

- The kill command has a misleading name because it does not actually kill processes. Rather, it sends signals to them.
- When no signal is explicitly included in the command, signal 15, named SIGTERM, is sent by default.

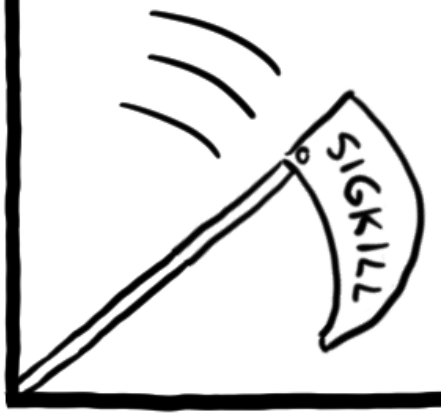

This is Bob. Bob is a Linux process.
1431
HI!
Like any process, Bob has his threads, with whom he shares context, memories and love.
1431
1431
164
4411
SIGTERM
1431
SYSadmin
And like all processes, inevitably sometime he will be killed.
When we gracefully kill a process with a soft SIGTERM...


...we give him the chance to talk with his kids about it. So, the kids finish their tasks...
1431
...and say goodbye to each other.
That's a process life!

On the other hand, when we brutally kill a process with a SIGKILL, we prevent them from finishing their job and say goodbye...

...and this is so SAD!

DAD!!



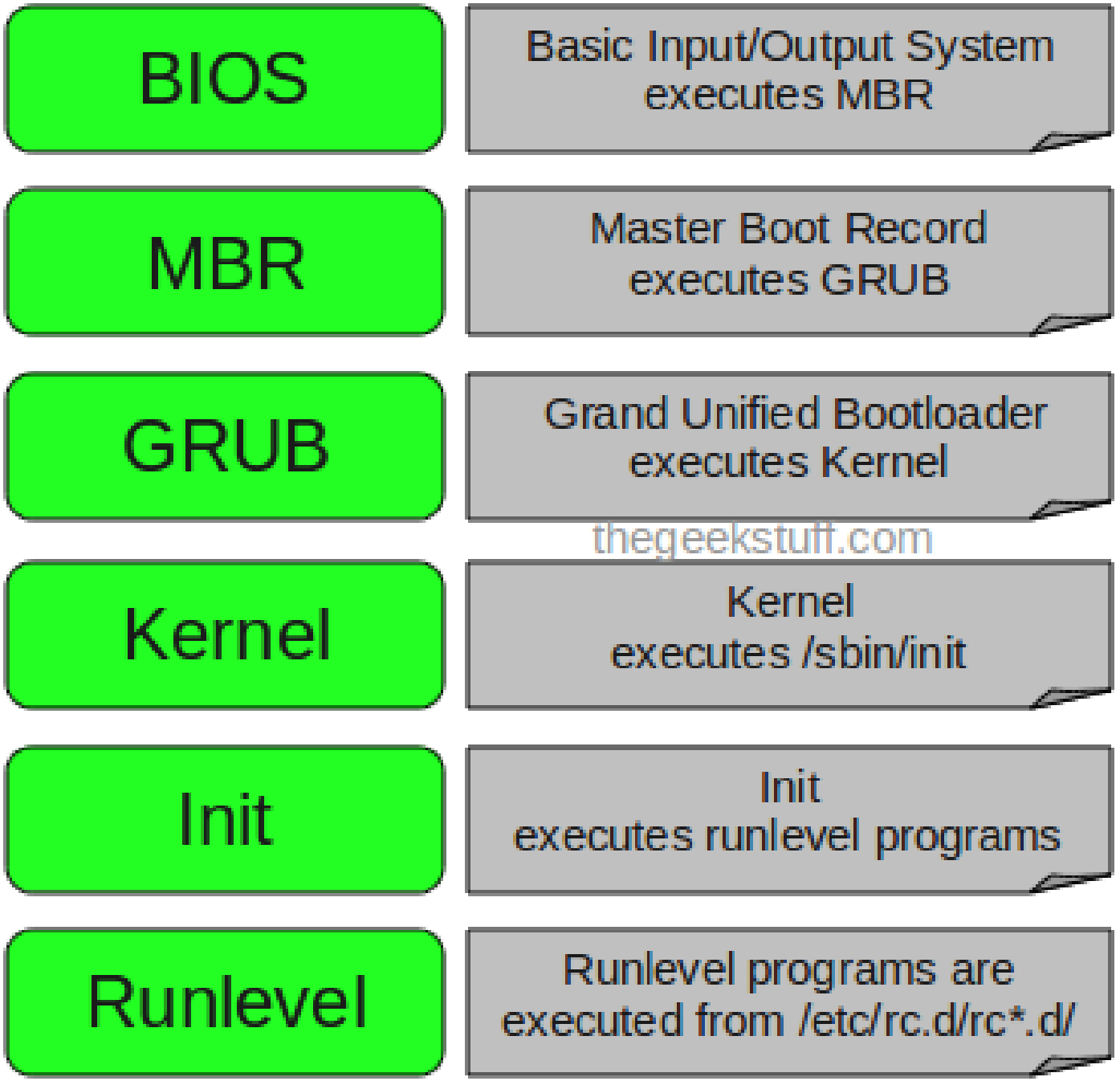
Dad, where are we going?

So please, DON'T use SIGKILL. Give the kids the chance to leave the kernel in peace.

Be nice.

Dad, where are you?

6 Stages of Linux Boot Process


BIOS
Basic Input/Output System
executes MBR
MBR
Master Boot Record
executes GRUB
GRUB
Grand Unified Bootloader
executes Kernel
thegeekstuff.com
Kernel
Kernel
executes /sbin/init
Init
Init
executes runlevel programs
Runlevel
Runlevel programs are
executed from /etc/rc.d/rc*.d/