O10

Linux Startup and Shutdown Part 1 of 2

Matilde L. Valdez

IBM eserver xSeries
Technical Conference

Aug. 9 - 13, 2004

Chicago, IL

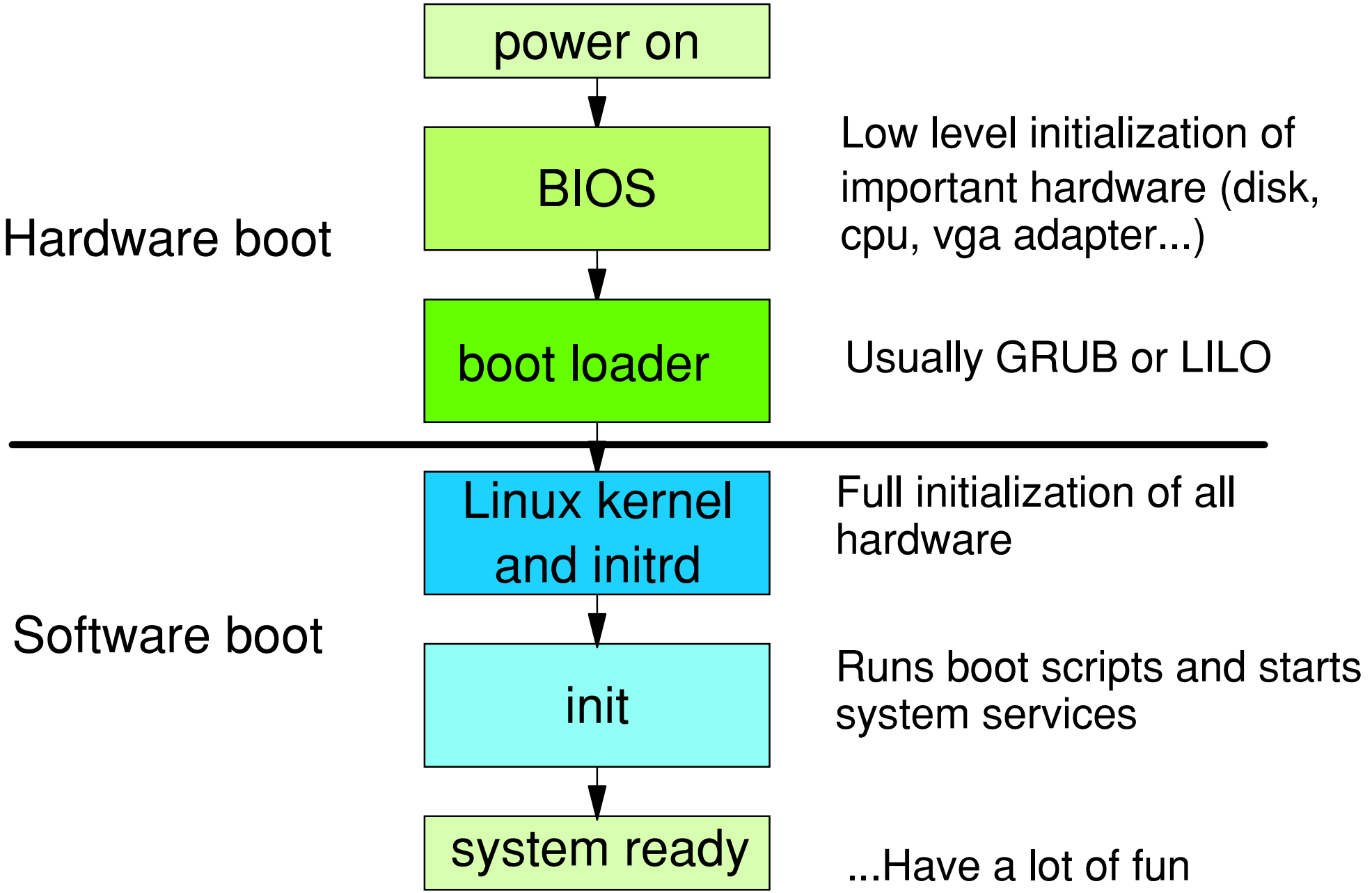
RETURN TO INDEX

Objectives

After completing this unit, you should be able to:

- Describe the Linux startup flow
- Configure autostarting services
- Boot Linux in single-user mode
- Perform a proper shutdown of a Linux system

Linux Startup Flow



Notes:

Basic Input Output System

- Checks memory and hardware (POST)
- Loads options from non-volatile memory
  - Memory timings
  - Order of boot devices
- Checks for boot devices
  - Floppy disks
  - CD-ROM
  - Hard disks
- Loads Master Boot Record of boot device and executes it

Notes:

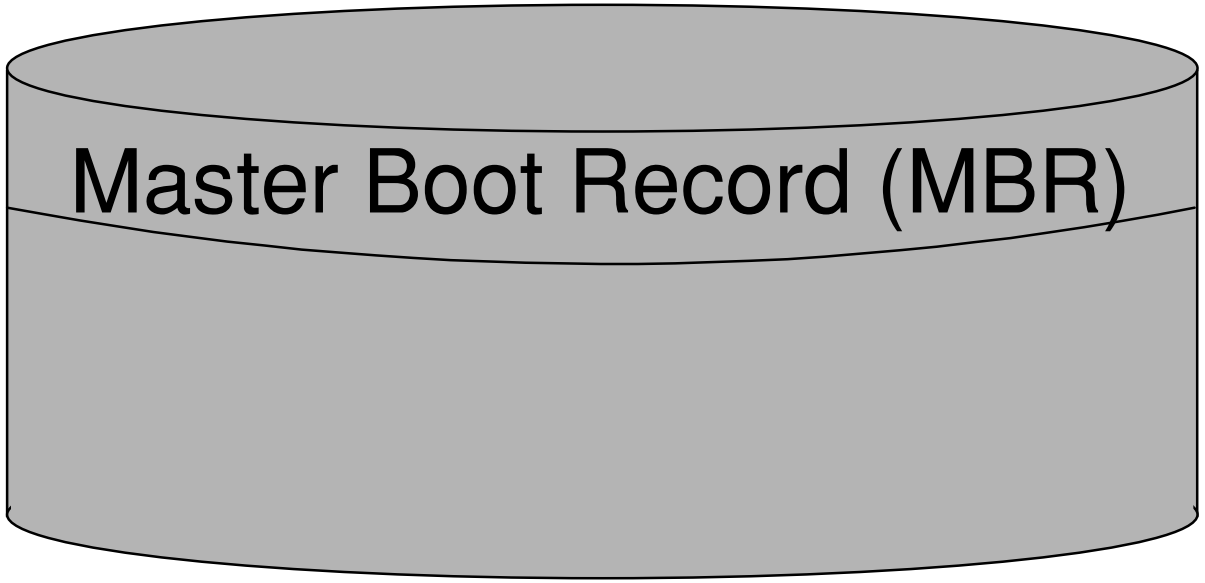
Every Intel PC has a Basic Input Output System, or BIOS for short. This is a little program which is stored in an EEPROM (Electrical Erasable Programmable Read Only Memory, sometimes also called non-volatile memory) on your motherboard. It is the first program that runs once the power is switched on. It does a number of basic tasks:

- It checks the memory
- It loads various options from non-volatile memory, for instance memory timing parameters, IRQ assignment to devices, and the order of boot devices. These options can be set by the user when pressing Del, F1, F2 or some other key while the memory is being tested.
- It checks for the availability of boot devices, and
- Loads the Master Boot Record of the first available boot device. This first sector is

stored in memory and executed.

Another thing the BIOS might do is configure (IRQ, DMA, I/O addresses) or disable peripherals (such as serial ports, parallel ports, network adapters, keyboards, mice and sound cards) that are integrated on the motherboard.

Master Boot Record



- Size: 512 Bytes (first sector of hd)
- addressed by BIOS
- Content:
  - 446 bytes program code (to boot an operating system)
  - 64 bytes **partition table with max. 4 entries**
  - 2 bytes "magic number"

Notes:

The Master Boot Record or MBR is the first sector (512 bytes) of the boot device. It contains three things:

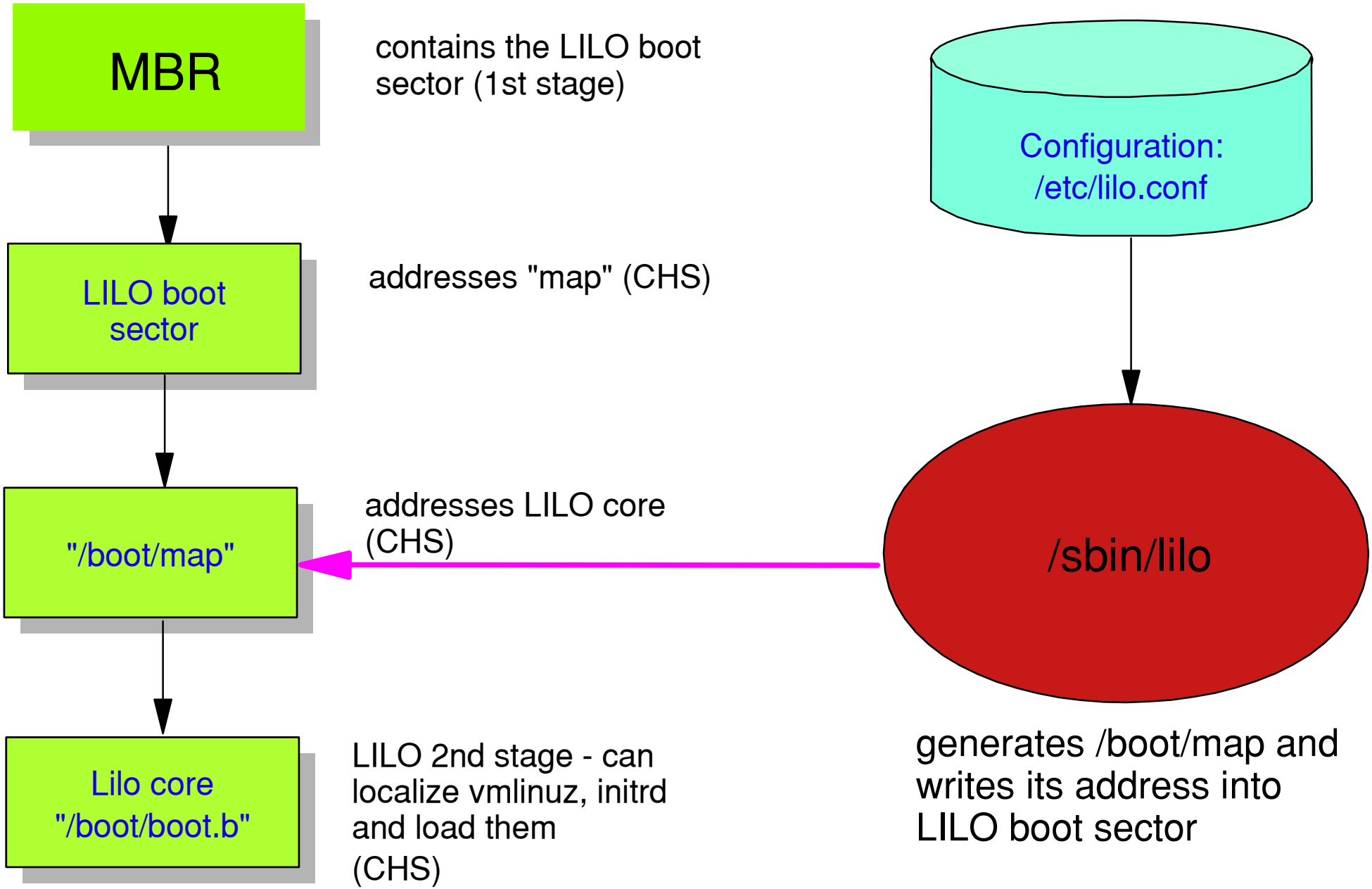
- A 446 bytes boot loader program: Software to bootstrap the operating system.
- The partition table: A 64-byte table which describes how the rest of the disk is split up into partitions.
- A 2-bytes magic number, which is used to check whether this is a valid MBR.

On systems fresh out of the shop, the bootloader is a very simple program which was configured with the MS-DOS command fdisk /mbr. This program goes through the partition table and looks for a partition that is marked "active". The program then loads the first sector of this partition and starts it. This concept is known as chain-loading.

When using Linux, the MBR is traditionally set up by the Linux Loader (LILO). It is a little more elaborate than the usual MBR, in that it can prompt the user for the operating system to load, and any options to pass to that operating system. Then, it loads the selected operating system, passing the options as it starts it.

Newer Linux distributions may use GRUB instead of LILO. GRUB is far more flexible than LILO, since it allows you to alter the configuration from the boot prompt. It is also versatile enough to boot other UNIX operating systems that can run on PC hardware, such as GNU/Hurd, *BSD and so forth. It also supports chain-loading of Windows operating systems, and supports hiding partitions, so that you can have multiple Windows operating systems on one disk simultaneously.

LILO - Linux Loader



Notes:

The Linux Loader (LILO) is the program that configures the MBR. It must be run as root with the lilo command. It parses the command line options, reads and checks the configuration file, and configures the MBR accordingly. The default configuration file is /etc/lilo.conf, but this can be overridden with the -C option. Other important options include:

-v Gives a verbose output.
-v -v Gives a very verbose output. In fact, you can have a total number of eight '-v's, giving you more and more output, until you literally drown in debug output.
-t Only tests the validity of the config file; does not actually write to the MBR.
-u, -U With this option, lilo restores an older backup copy of the MBR to the MBR on disk. This backup was made the first time lilo was run and is called /boot/boot.0300 or /boot/boot.0800.1

It can be used to recover from a mangled MBR for instance, and can be used for a complete deinstall of Linux.2
For more details, refer to the lilo manual page (man lilo)

GRand Unified Bootloader (GRUB)

- Program stored in MBR (first stage) and in /boot/grub (1.5th and second stage)
- Understands filesystem structure
  - No need to activate a configuration as with LILO
- Configuration file /boot/grub/grub.conf (Red Hat) or /boot/grub/menu.lst (SuSE)
- Installed in MBR with **grub-install**
- When system boots:
  - Select predefined OS to boot, or
  - Use command language to boot non-predefined OS
  - Command language compatible with configuration file
- GRUB additional features:
  - MD5 encrypted passwords
  - Hiding/Unhiding partitions

Notes:

GRUB (GRand Unified Boot Loader), as LILO, consists of a number of separate stages:

- The first stage, called stage1 on disk, is usually stored in your MBR.
- The 1.5th stage, called *_stage1_5 (e2fs_stage1_5, fat_stage1_5, minix_stage1_5, reiserfs_stage1_5, ...) is stored on disk, typically in /boot/grub. Several 1.5th stage files exist, each for a different filesystem.

This stage is used to add filesystem capabilities to GRUB, so that GRUB is able to use regular filename references when loading configuration files, kernels and such, instead of disk block locations.

Because of this stage, GRUB is able to read its configuration file directly, and does not need to be configured beforehand, like LILO.

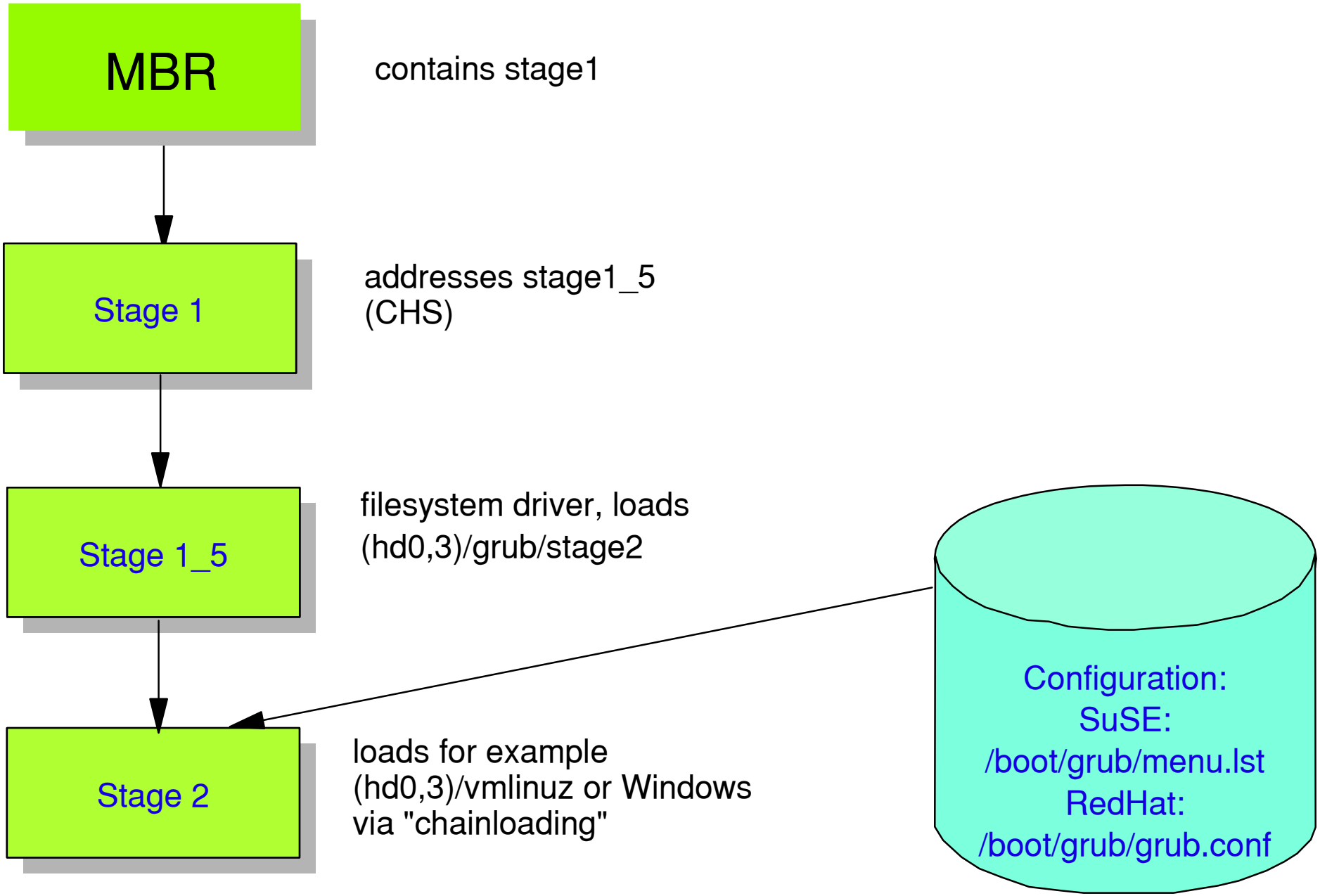
- The second stage, called stage2. This gives a menu interface which allows you to boot your predefined operating systems, or enter commands to boot a non-predefined operating system.

If a "splashimage" was included in the GRUB configuration, then the second stage will display the menu in a graphical mode, with the splash image as background. The GRUB configuration file is typically stored in your /boot filesystem, in a separate GRUB directory, and called grub.conf (Red Hat) or menu.lst (SuSE). On a regularly booted Linux system, this file is thus referenced as /boot/grub/grub.conf or /boot/grub/menu.lst. It contains all predefined operating systems and their options and peculiarities.

To install GRUB, either use the shell script grub-install or start the grub program and use GRUB commands to install GRUB manually. GRUB has some additional features that make it far more useful than LILO:

- GRUB supports MD5-encrypted passwords to protect normal users from supplying parameters and options to predefined operating system, or to define their own operating system boot procedure.
- GRUB can perform hiding and unhiding of Windows partitions. This is a requirement for running multiple Windows operating systems from the same disk.3
- If configured properly, GRUB can be used to boot from the network. This requires the netboot package, and requires you to set up a DHCP and TFTP server though. Network booting is outside the scope of this course.

GRUB - Grand Unified Bootloader



Notes:

/boot/grub/grub.conf or /boot/grub/menu.lst

```
default=0
timeout=10
title Red Hat Linux 9 (2.4.20-8)
        root (hd0,2)
        kernel /vmlinuz-2.4.20-8 ro root=/dev/hda5
        initrd /initrd-2.4.20-8.img
title SuSE Linux 8.2
        kernel (hd0,2)/vmlinuz-2.4.20-4GB-i686 root=/dev/hda6
        initrd (hd0,2)/initrd-2.4.20-4GB-i686
title Windows 95
        unhide (hd0,0)
        hide (hd0,1)
        rootnoverify (hd0,0)
        makeactive
        chainloader +1
title Windows 98
        unhide (hd0,1)
        hide (hd0,0)
        rootnoverify (hd0,1)
        makeactive
        chainloader +1
```

Notes:

The GRUB configuration file, /boot/grub/grub.conf (Red Hat) or /boot/grub/menu.lst (SuSE), is nothing more than a predefined series of commands that could just as well have been entered on the GRUB command line. Storing these commands in a file though makes booting far more convenient... The file starts with a few general configuration options:

default=0 This specifies the default operating system to be started. GRUB also allows you to specify the fallback parameter, which specifies the operating system to boot in case the default fails.

timeout=10 Timeout before starting the default operating system, in seconds.
When general options are all defined, specific operating systems need to be predefined. For this, the following keywords may be needed:

title = The title of the operating system, as it shows up in the GRUB boot screen.

root = The root partition of the filesystem. All files that are referenced later on are stored on this filesystem. Specifying root is not required, but you will have to identify the root partition every time you mention a file instead, as is done with the SuSE stanza.

kernel = The kernel image that is to be loaded, and all options that need to be passed to the kernel. initrd An initial root disk that needs to be loaded.

unhide = Unhide the partition specified (i.e. change its type so that Windows systems will recognize it).

hide = Hide the partition specified (i.e. change its type so that Windows systems will not recognize it).

rootnoverify =The root of the operating system is the partition specified, but don't try to verify and access this as GRUB does not support the filesystem type.

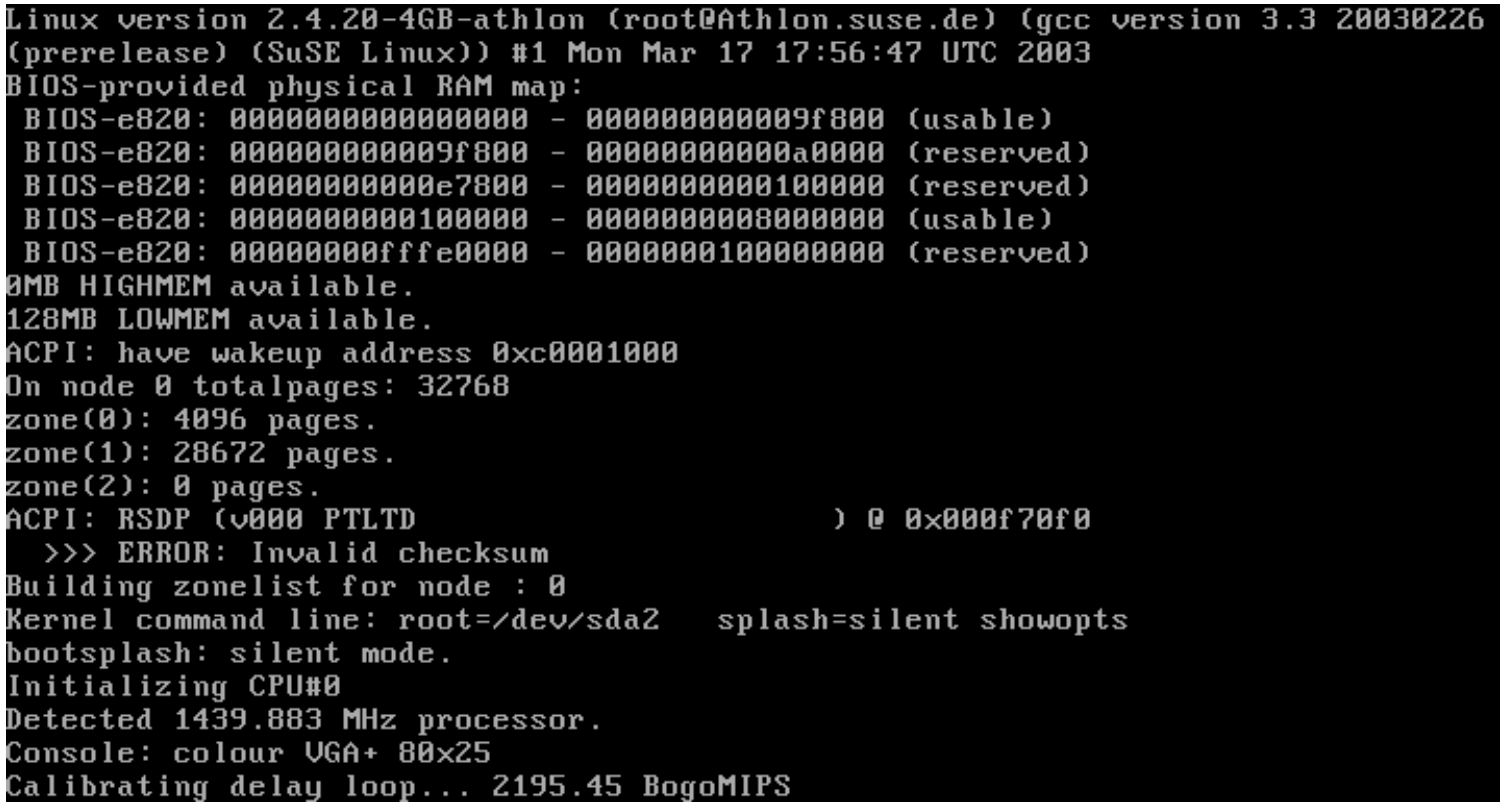
makeactive= Mark this partition active in the partition table.

chainloader +1 To boot this operating system, invoke the chainloader, which needs to load the first sector of the specified root partition.

Note that different distributions can and have made extensions to grub, which allow for instance graphics to be used.

Starting the Kernel

- Once the kernel is loaded, it is started by the boot loader
- On most architectures (including i386) the kernel is compressed with a decompress program included
- When the kernel starts, it detects all hardware and switches the CPU to multitasking, multiuser mode



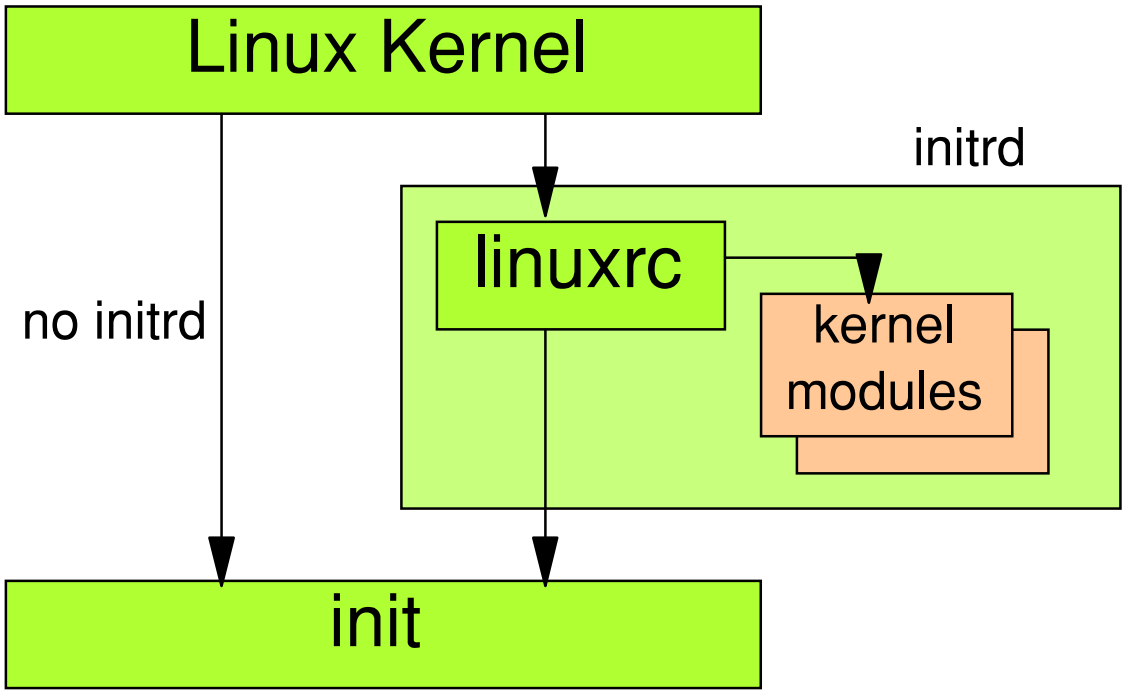
Notes:

When the user selects a Linux operating system in the boot loader, then the boot loader will load the Linux kernel. Because of space constraints, the Linux kernel is compressed, but has an uncompress program attached to it. Actually, it looks like a self-decompressing ZIP file in DOS. The uncompress program uncompresses the Linux kernel and puts it into memory. Then, it starts that kernel proper.

The first thing the kernel does is try to detect all the hardware for which it has support built in. This includes hard disks, serial devices, mice, graphical adapters, keyboards, network adapters and the like. By far most of these adapters can indeed be autodetected, but some can't. In that case, their configuration parameters (most notably, IRQ, I/O and DMA levels) need to be passed to the kernel as boot options. If this is the case, consult the Hardware-HOWTO for details.

Initial RAM Disk (initrd)

- An Initial RAM Disk (initrd) is needed if the kernel can't access the root filesystem without modules (SCSI, LVM, RAID, ext3, reiser)
- The initrd is loaded into memory by the boot loader
- The initrd contains a **linuxrc** script that loads the modules from the RAM disk, then starts **init**



Notes:

Not all hardware is supported in the core kernel image. In fact, almost all hardware support in Linux today comes in the form of modules. These modules are pieces of code that are loaded into kernel memory only if required.

This works well, but leads to a minor problem if kernel modules are needed to mount the root filesystem. This can happen, for instance because:

- The root filesystem sits on a hard disk type for which support was not compiled into the kernel image. This applies mostly to SCSI
- LVM or RAID was used, and LVM or RAID support was not compiled into the kernel image.
- The root filesystem uses ext3, JFS or ReiserFS as filesystem type, and support for these filesystems was not compiled into the kernel image.

In these cases, you are going to need an Initial RAM Disk (sometimes also called an Initial Root Disk). This is a file containing a compressed image of an ext2 filesystem, which in turn contains two things:

- A linuxrc script
- The kernel modules that are needed

The initrd image is loaded into memory by the boot loader, just like the Linux kernel. When the Linux kernel starts, it detects the presence of the initrd. It then proceeds to uncompress and mount this filesystem as temporary root. The kernels last direct action is then to start the linuxrc script. The linuxrc script loads all the required modules, mounts the true root filesystem and thenexecutes a system call pivot_root. This switches the position of the initrd and the true root filesystem. From that point on, the actual root filesystem is mounted at its correct location, and linuxrc is able to continue the boot process by starting the /sbin/init program.

init

- **init** is started by the kernel or **linuxrc**
- **init** reads configuration file /etc/inittab
- Decides on default runlevel if no runlevel is given
- Runlevels have different meaning:
  - 0: halt
  - 1: single user mode
  - 2: multiuser without NFS
  - 3: full multiuser mode
  - 4: unused
  - 5: multiuser with graphical login
  - 6: reboot
- **init** will start all programs for that runlevel
- Note: Once the system has started, you can switch runlevels with **init <runlevel>** or **telinit <runlevel>**

Notes:

/etc/inittab (Red Hat/SuSE)

Red Hat

```
# Default runlevel
id:3:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

SuSE

```
# Default runlevel
id:3:initdefault:

# System initialization.
si::bootwait:/etc/init.d/boot

l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6

# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -r -t4 now

# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty --noclear tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

The default runlevel is 3

Always run /etc/rc.d/rc.sysinit (RH) or /etc/init.d/boot (SuSE)

Run /etc/rc.d/rc (RH) or /etc/init.d/rc (SuSE) with the runlevel as parameter

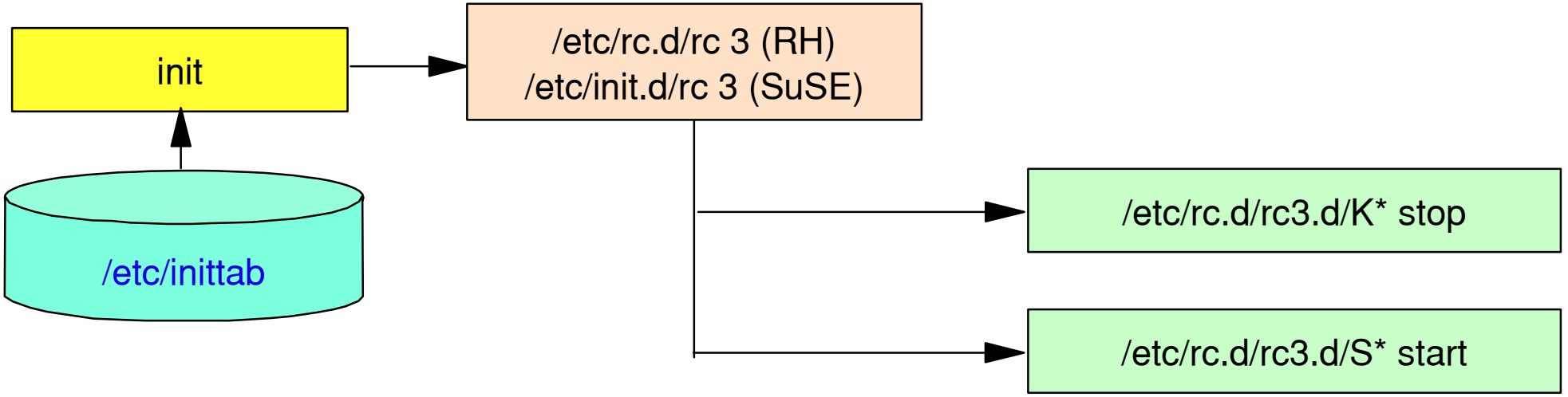
Trap the three-finger salute

Allow users to log in on six virtual consoles (Virtual consoles can be activated with Alt-F1 through Alt-F6)

Start a graphical login prompt (xdm, kdm or gdm) in runlevel 5

Notes:

Starting Services (System V init style)



(Symlinks to the actual start/stop script)

```
# ls -l /etc/rc.d/rc3.d
lrwxrwxrwx   1 root root 17 Mar 16 21:17 K10pulse -> ../init.d/pulse
lrwxrwxrwx   1 root root 17 Mar 16 21:17 K10xntpd -> ../init.d/xntpd
lrwxrwxrwx   1 root root 17 Mar 16 21:17 S10network -> ../init.d/network
lrwxrwxrwx   1 root root 17 Mar 16 21:17 S11portmap -> ../init.d/portmap
.
lrwxrwxrwx   1 root root 17 Mar 16 21:11 S99local -> ../rc.local
```

Notes:

The rc script is a very important script. Although small, it is responsible for starting almost all services that are active in the runlevel that was specified as parameter.

What this script basically does is the following:
• It changes to the directory /etc/rc.d/rc<runlevel>.d4
• In this directory, it makes a list of all scripts that start with a K, sorts this list on the two digits after the K, and executes these scripts with the stop parameter.5
• Then, it makes a list of all scripts that start with an S, sorts it, and executes them with the start parameter.

These scripts are in fact not scripts at all, but are symbolic links to generic scripts in /etc/rc.d/init.d or /etc/init.d.6 Every server program that is installed on a Linux system is supposed to have a corresponding control script in this directory, with the same name as that service. By making a symbolic link from /etc/rc.d/rc3.d to that particular script, the administrator ensures that a particular service is started (or stopped) in a certain runlevel. And by specifying a two-digit number after the S or K, the administrator can even influence the order in which services are started and stopped. This scheme was first used in AT&T's system V (five) Unix. That's why it is called the System V init style. It is used, among others, by Red Hat and SuSE. Other Linux distributions may use other init styles. But for all distributions the principle holds: init reads the /etc/inittab files and starts all the programs that are listed there. There is never a magic or secret program or script being started. That means that it doesn't really matter which distribution you use. Take a look at the /etc/inittab file and read the scripts that are listed here. This will tell you how the system is started.

Configuring Services per Runlevel

- Use **chkconfig** to create the appropriate K- and S- links for each service.

```
# chkconfig --list
acpid      0:off  1:off  2:off  3:off  4:off  5:off  6:off
atd        0:off  1:off  2:on   3:on   4:off  5:on   6:off
...
# chkconfig acpid on
# chkconfig --list
acpid      0:off  1:off  2:on   3:on   4:off  5:on   6:off
atd        0:off  1:off  2:on   3:on   4:off  5:on   6:off
...
```

Notes:

Each of the service scripts includes information for chkconfig listing the default runlevels for which the service should be on or off, and the default priority. This leads to the following:

- chkconfig <service> off switches the service off for all runlevels
- chkconfig <service> on switches the service on for the runlevels listed in the service script. In most cases, this is runlevels 2 through 5.

If you want chkconfig to work on other runlevels than the default 2 through 5, you can specify this with the --levels <levels> option. In addition to chkconfig, you can also use your distributions system management tool (redhat-config-services or yast) to manage these services.

Starting and Stopping Services Manually

- Scripts in init.d directory can be used to start/stop services manually
  - On Red Hat, the **service** command calls this script
  - On SuSE, **rc<service>** is a symlink to the init.d script
- Default options: start, stop, status, restart
- Other options may also be available

```
redhat# service atd restart
Stopping atd:                                          [ OK ]
Starting atd:                                          [ OK ]
```

```
suse# rcatd restart
Shutting down service at daemon                          done
Starting service at daemon                               done
```

Notes:

Sometimes it is necessary to have full control over your system, with no users or other programs doing all kinds of unexpected things. This is possible in Linux, and is called Single-User Mode. For single-user mode, you will need to specify the single option to the kernel when your system boots. The Linux kernel will then boot as normal, but init will only run /etc/rc.d/rc.sysinit or /etc/init.d/boot and then start a bash shell. It will not start all the normal services, so users can't log in over the network.

On a Red Hat system, the single user mode will not even ask for a root password. This is done so that it can be used if you forgot your root password, and need to set a new one. Obviously, in single user mode the system is not very useful, except for you. So after your system maintenance, you need to switch back to normal mode (runlevel 3 or 5). The safest course of action here is to do a full reboot (shutdown -r now).

Booting Linux in Single-User Mode

- Single-User Mode
  - No networking (so no incoming hackers)
  - No services being started
  - No root password being asked (Red Hat)
- Very useful for system maintenance
- To start from LILO: add "single" parameter to boot-prompt

```
LILO
For Linux, type linux, for Windows 95, type win
Boot: linux single
```

- To start from GRUB: add "single" to the corresponding menu entry
- When finished:
  - **exit** the shell to start the default runlevel, or
  - **shutdown -r now** to reboot

Notes:

Shutting Down a Linux System

- DO NOT switch power off to shut down
- Use **shutdown** command or Ctrl-Alt-Delete
  - Warns users
  - Stops all running processes
  - Unmounts filesystems
  - Does an orderly shutdown
  - Reboots if necessary
- Example:
  - To reboot: **`shutdown -r now`** or **`reboot`**
  - To halt: **`shutdown -h now`** or **`halt`**
- Some Display Managers allow a user to perform a shutdown as well

Notes:

RETURN TO INDEX