Systems 3

Linux Boot Process in Detail

Marcel Waldvogel

(Handout)

Department of Computer and Information Science
University of Konstanz

Winter 2019/2020

Photo by Scott Warman on Unsplash

Chapter Goals

- Understanding Munchhausen
- How does CPU, hardware, and kernel configure itself from an unconfigured state?
- Understanding the dynamics of hardware

Boot vs. startup process

1. Boot process
   - Power button
   - ...
   - Kernel initialized
   - Systemd started
2. Startup process
   - Services

Bootstrapping

- **Boot** is short for **bootstrap**
- From simple to complex
- From nothing to something

Klaus Herberth (CC BY 3.0)

Boot process (revisited)

Basic overview of the Linux boot process.

Actor	Actions
BIOS/UEFI	test hardware, execute Master Boot Record
MBR	load and execute Grand Unified Bootloader
GRUB	load kernel and initrd
Kernel	mount root file system and start init
Init	determine run level and start
Runlevel	start various services

Question: How does it work in detail on a modern machine with Linux OS?

Where to start?

How does the (IA) CPU obtain it's first instruction?

- Address hard coded
- Real-address mode[1]
- Interrupts disabled
- 16 bytes below the processor's uppermost address[2]: 0xFFFFFFF0
- EPROM with software-initialization code

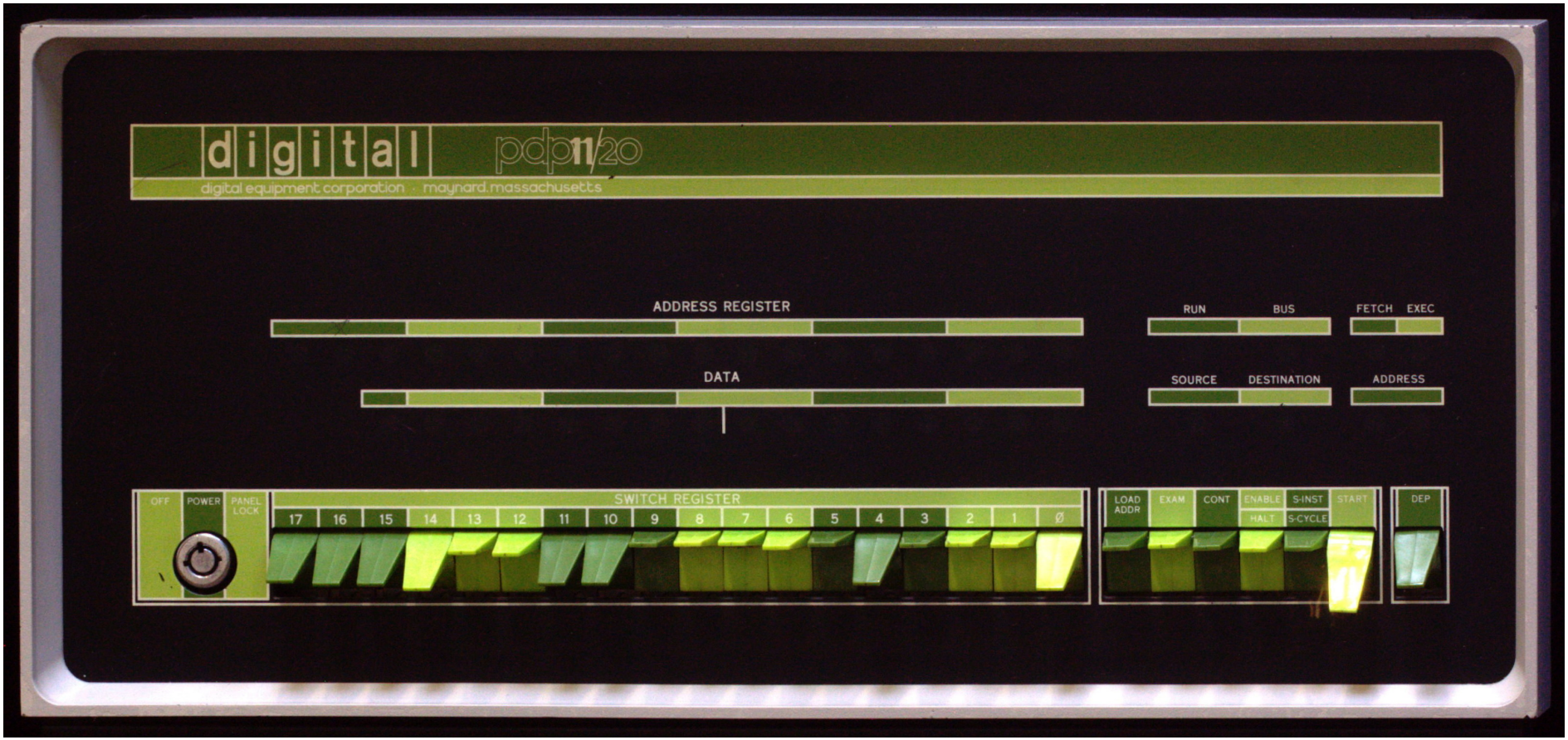
How does the CPU find memory?

- No RAM configured at first (different memory banks, different sizes, different speeds, ...)
- (E)EPROM

[1] 8086/8088 compatibility, page tables or virtual addresses etc. disabled

[2] Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 3A Section 9.1.4

Where to start? (cont'd)

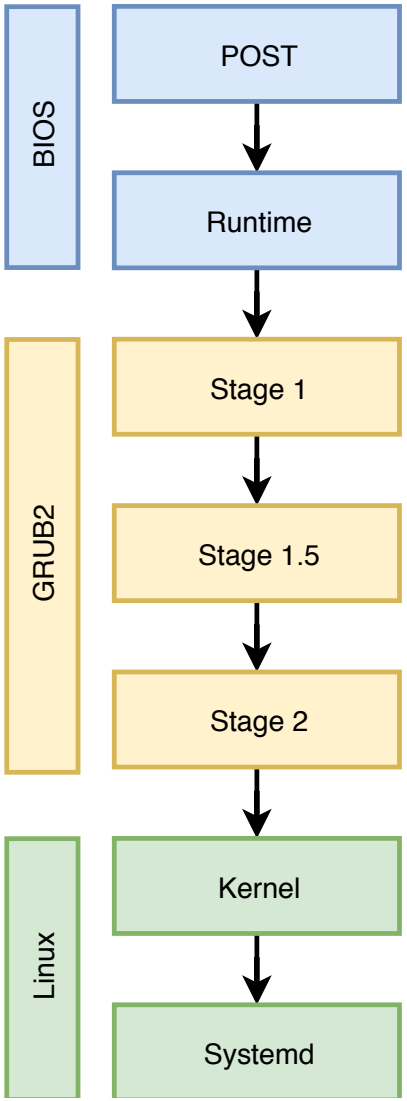


[2] Photo credit: Wikpedia user Rama and Musée Bolo, CC BY-SA 2.0 France

BIOS with MBR

- GRUB (successor to LILO)
- Able to read config from filesystem
- GRUB2 is rewrite of GRUB. Improvements include:
  - Dynamic module loading
  - Rescue mode
  - Themes
  - Boot ISO images directly from hard drive

More information:
https://help.ubuntu.com/community/Grub2



Power-on self-test

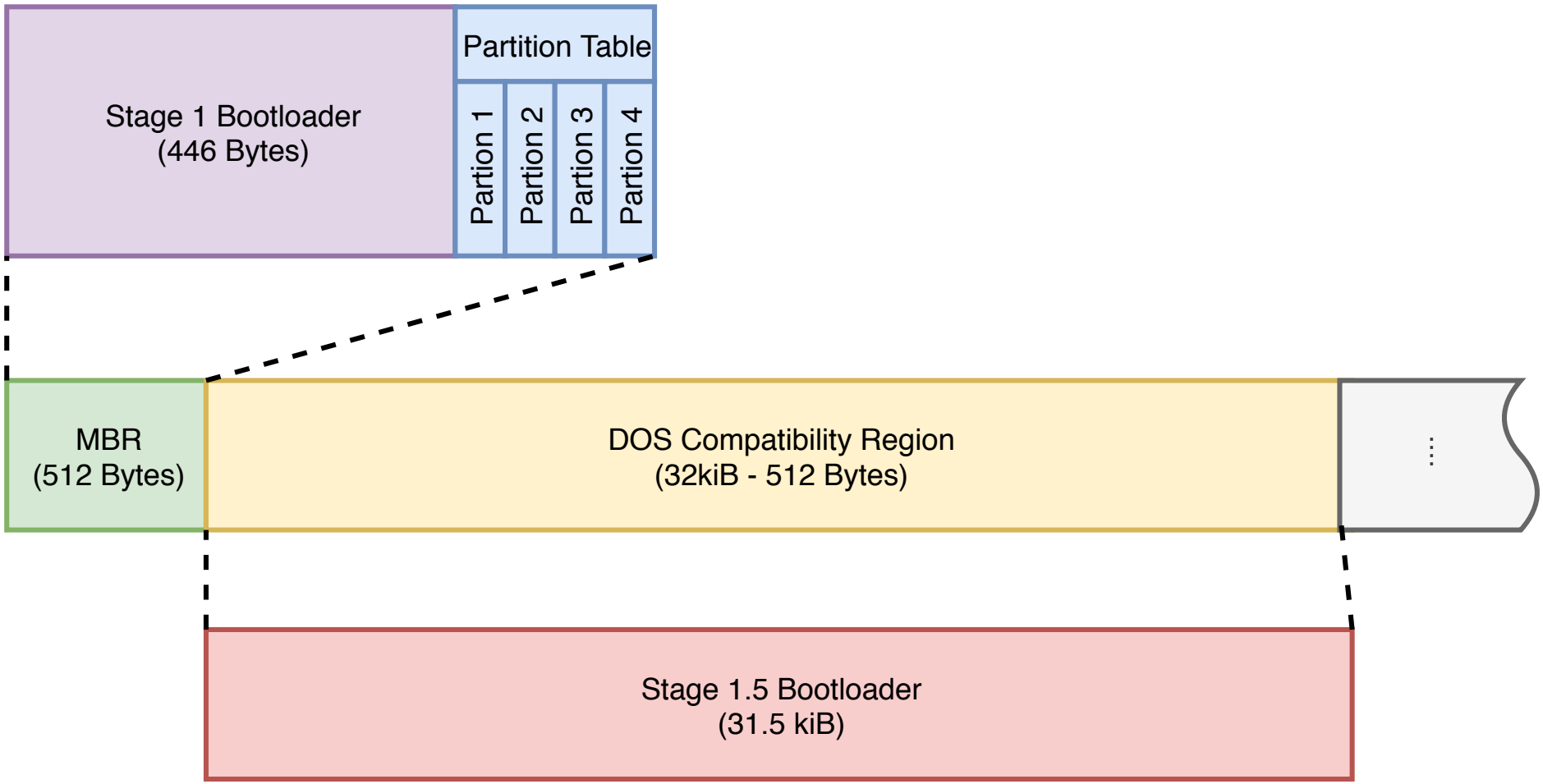
Performed immediately after a computer is powered on.

- verify CPU registers
- verify BIOS code
- verify basic components
- verify RAM
- initialize BIOS

BIOS Runtime

- Search bootable device
- Load next step from bootable device

Bootloader


Stage 1 Bootloader
(446 Bytes)
Partition Table
Partion 1
Partion 2
Partion 3
Partion 4
MBR
(512 Bytes)
DOS Compatibility Region
(32kiB - 512 Bytes)
Stage 1.5 Bootloader
(31.5 kiB)

Stage 1

- Stage 1 bootloader lives in MBR
- Size limitation of 446 bytes
- Needs to be simple
- Loads only stage 1.5

You can easily extract and save your MBR:

```
1 $ dd if=/dev/sda of=/tmp/mbr.dat bs=512 count=1
```

Stage 1.5

- DOS required that partition starts at cylinder boundaries
- Mount filesystem
- Load stage 2 bootloader from disk

Stage 2

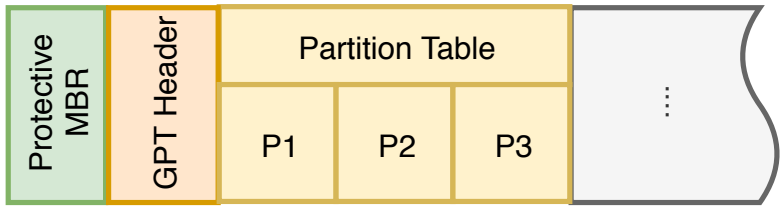
- Read `grub.cfg`
- Show boot menu (optional)
- Load kernel into RAM
- Load `initrd` into RAM

MBR weaknesses

- pretty old (1983)
- disk size limitation; originally physical (CHS) addressing
- only support for 4 primary partitions

GUID Partition Tables (GPT)

- globally unique identifier
- no size limitation
- unlimited number of partitions
- multiple copies
- CRC

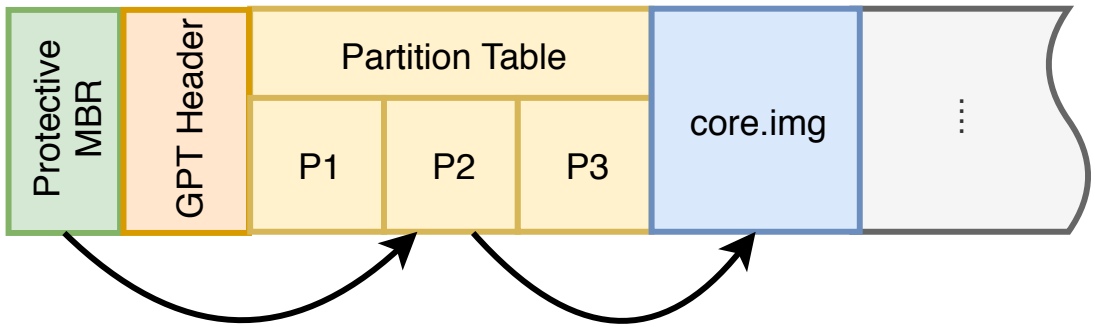


Question: What is Protective MBR?

Problem: No space for bootloader between MBR and GPT header.

BIOS with GPT

- BIOS boot partition
- Type EF02
- GUID 21686148-6449-6E6F-744E-656564454649
- GUID in ASCII is `Hah!IdontNeedEFI`



BIOS weaknesses

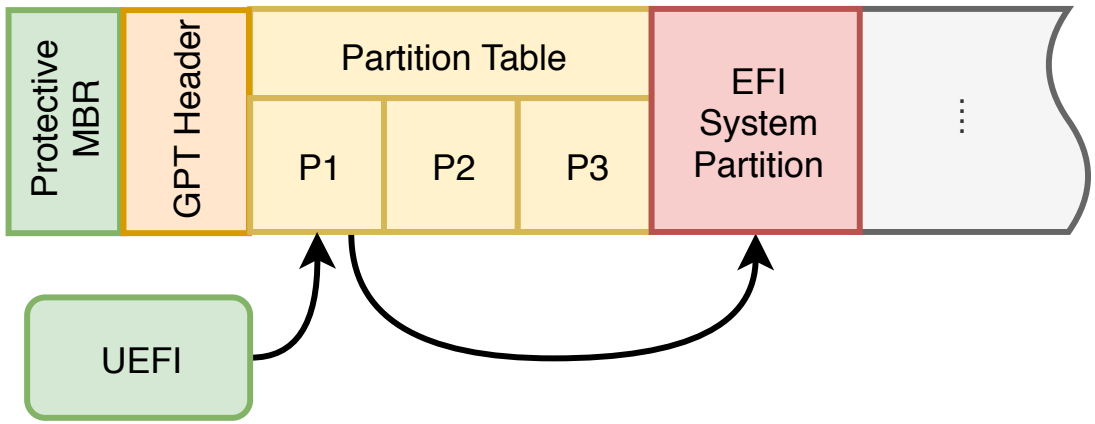
- Used since MS-DOS (1980)
- Must run in 16-bit processor mode (real-address mode)
- 1MB of space to execute in

UEFI

- 32-bit and 64-bit mode
- Nicer boot screen
- Secure Boot
- Network support
- Configuration via OS (no wait time required)

UEFI with GPT

- UEFI processor reads partition table
- EFI System Partition (ESP) is of type EF00
- ESP is VFAT formatted
- EFI application is named `grub.efi`
- Next `/boot` will be mounted
- Secure boot requires to invoke `shim.efi` at the beginning



Booting kernel

- kernel is packed as self-extracting archive
- archive gets loaded into memory
- program counter is set to begin of archive
- kernel gets extracted
- systemd is initialized (startup process)

```
$ ls -l /boot/
total 230483
-rw-r--r-- 1 root root   235885 Nov 13 23:41 config-5.3.0-24-generic
-rw-r--r-- 1 root root   235804 Dez 18 06:21 config-5.3.0-26-generic
drwx------ 3 root root     4096 Jan  1  1970 efi
drwxr-xr-x 5 root root     1024 Jan 22 09:39 grub
lrwxrwxrwx 1 root root       27 Jan  7 12:36 initrd.img -> initrd.img-5.3.0-26-gene
-rw-r--r-- 1 root root 80842621 Dez  9 11:32 initrd.img-5.3.0-24-generic
-rw-r--r-- 1 root root 80844710 Jan  7 12:36 initrd.img-5.3.0-26-generic
lrwxrwxrwx 1 root root       27 Jan  7 12:36 initrd.img.old -> initrd.img-5.3.0-24-
...
lrwxrwxrwx 1 root root       24 Jan  7 12:36 vmlinuz -> vmlinuz-5.3.0-26-generic
-rw------- 1 root root 11399928 Nov 14 01:14 vmlinuz-5.3.0-24-generic
-rw------- 1 root root 11399928 Dez 18 06:27 vmlinuz-5.3.0-26-generic
lrwxrwxrwx 1 root root       24 Jan  7 12:36 vmlinuz.old -> vmlinuz-5.3.0-24-generi
```

Changing the kernel

- Originally the kernel was monolithic
- Changes (new drivers, fixes, ...) required to
  - recompile
  - reinstall
  - reboot
- Solution was to use kernel modules

Practical demo

Use `lsmod` to show a list of the currently loaded kernel modules.

Loading kernel modules from FS

With a modular kernel the boot process is more complicated. For example, in order to mount the root filesystem a kernel module is required. Unfortunately, this module is stored on the filesystem.

Question: How are modules loaded during initialization?

Init RAM Disk (initrd)

- Grub allows to specify an initrd file
- Compressed archive containing a few kernel modules and scripts
- Work only with a specific kernel version

```
$ ls -l /boot/
total 230483
...
lrwxrwxrwx 1 root root       27 Jan  7 12:36 initrd.img -> initrd.img-5.3.0-26-gene
-rw-r--r-- 1 root root 80844710 Jan  7 12:36 initrd.img-5.3.0-26-generic
...
lrwxrwxrwx 1 root root       24 Jan  7 12:36 vmlinuz -> vmlinuz-5.3.0-26-generic
-rw------- 1 root root 11399928 Dez 18 06:27 vmlinuz-5.3.0-26-generic
...
```

Practical hint

Use `mkinitramfs(8)` to generate a new initrd file.

Startup process

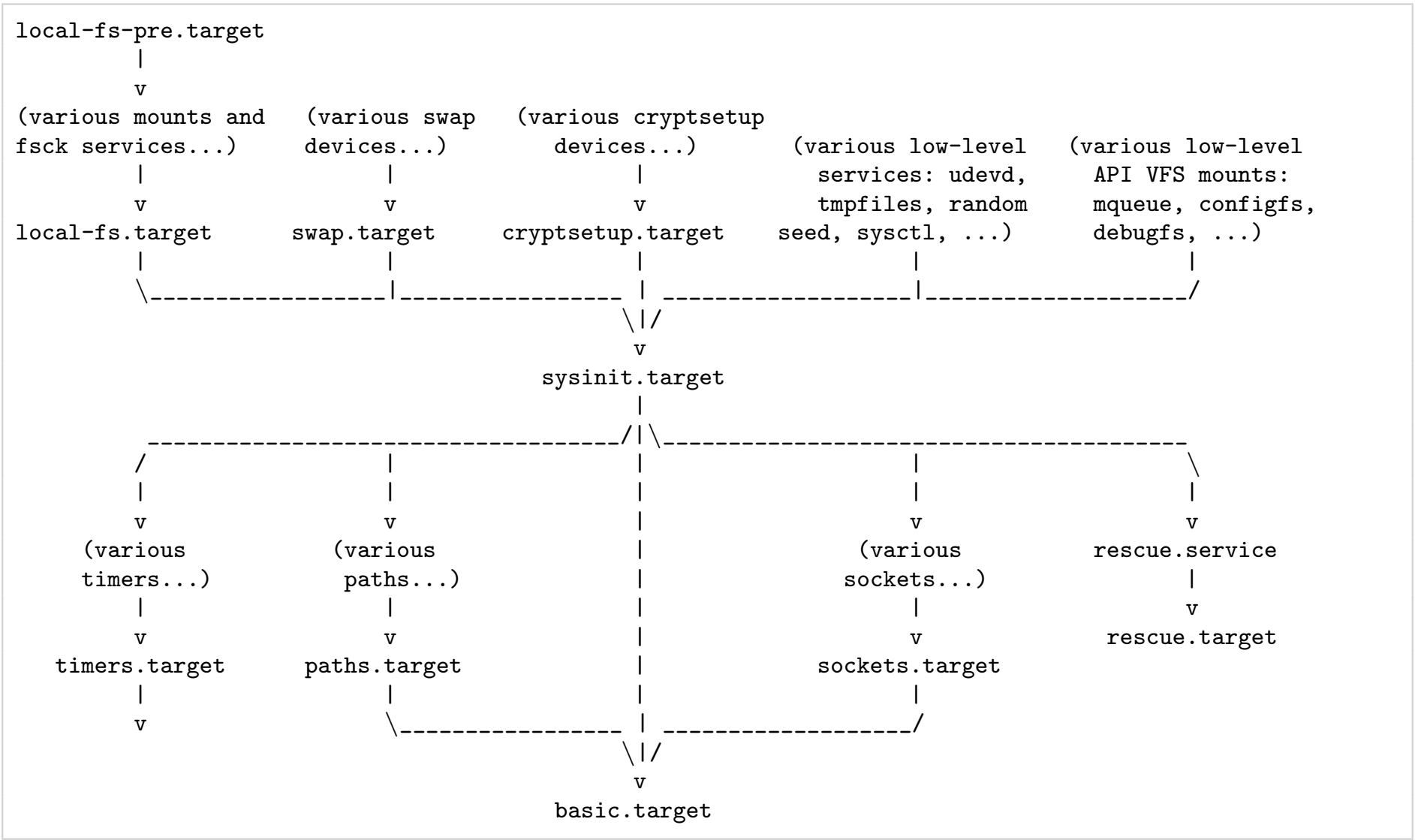
"The startup process follows the boot process and brings the Linux computer up to an operational state in which it is usable for productive work."[3]

[3] https://opensource.com/article/17/2/linux-boot-and-startup (visited 2020-01-23)

Systemd targets

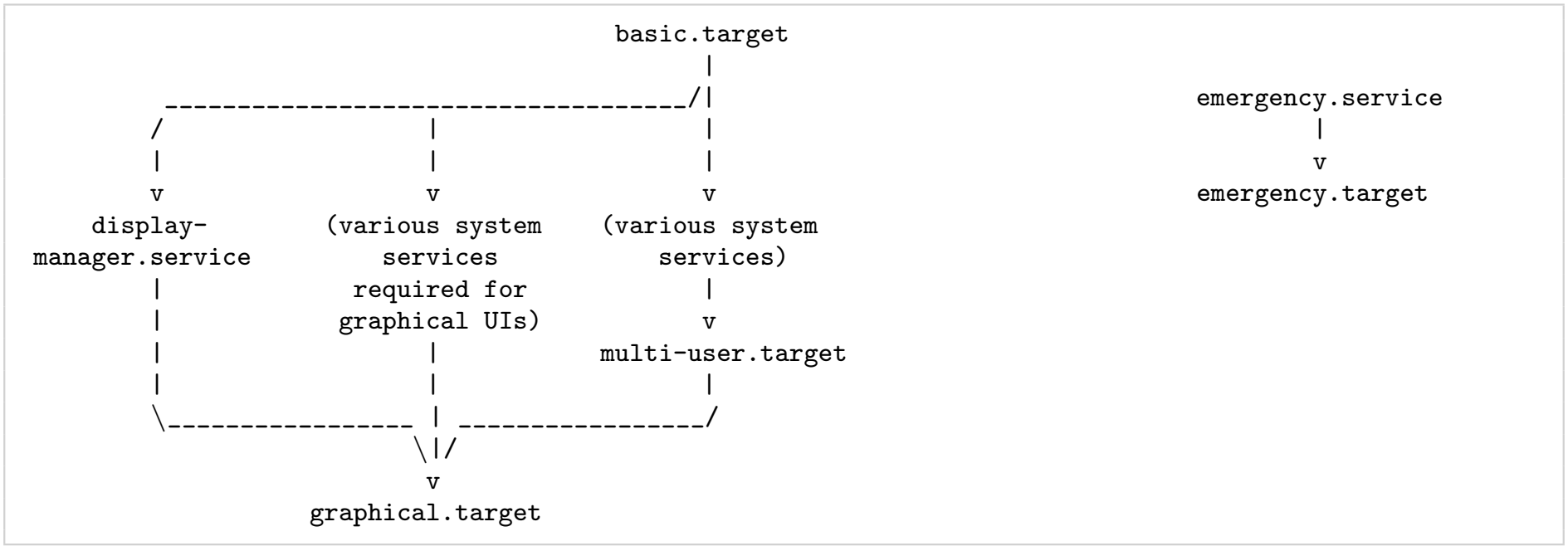
Target	Description
poweroff.target	Halts the system and turns the power off
emergency.target	Single user mode. No services are running; ...
rescue.target	Base system
multi-user.target	All services with CLI only
graphical.target	Multi-user with GUI

Target dependencies (1)



See bootup(7) for more details.

Target dependencies (2)



See bootup(7) for more details.