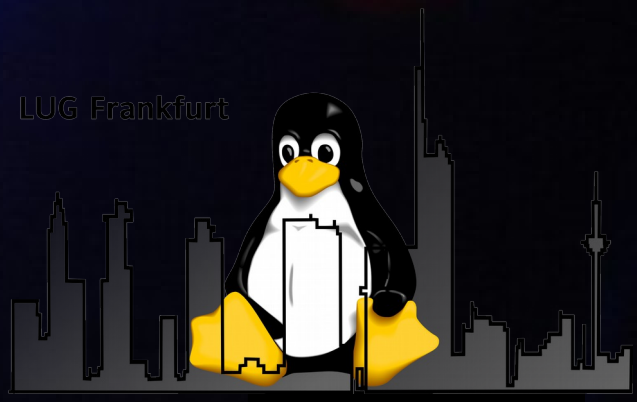
Vom "Power On" bis zum benutzbaren Desktop

Der Boot-Prozess eines GNU/Linux-Systems



Martin Feuersänger

28.9.2010

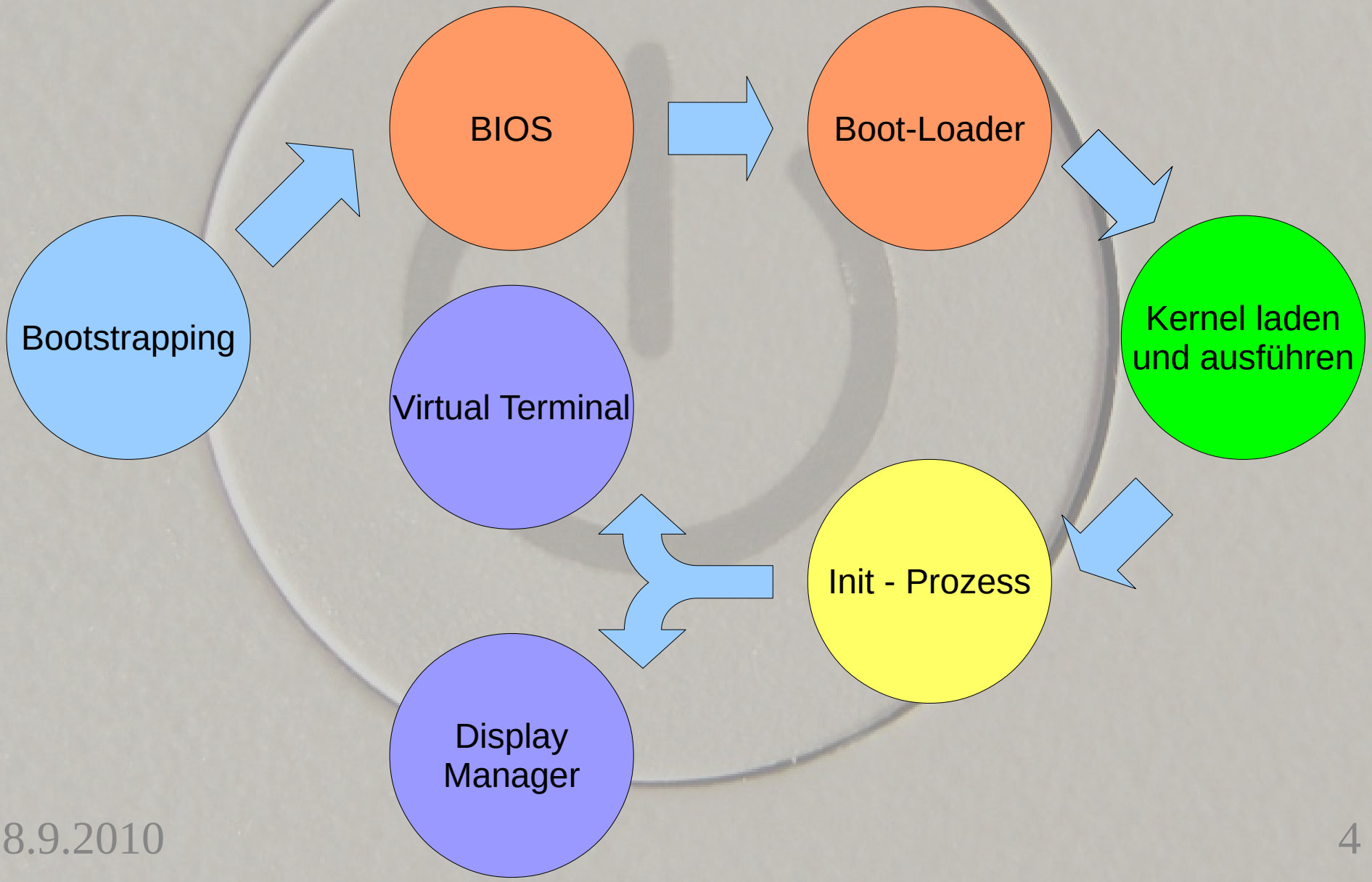
Vortrag soll die folgenden Fragen klären

- Was passiert beim “Booten”, also vom Einschalten des Rechners bis zum Erscheinen des Login Prompts oder Display Managers?
- Welche Software gibt es im Linux / Unix / OpenSource Bereich
- Was bringt die Zukunft?

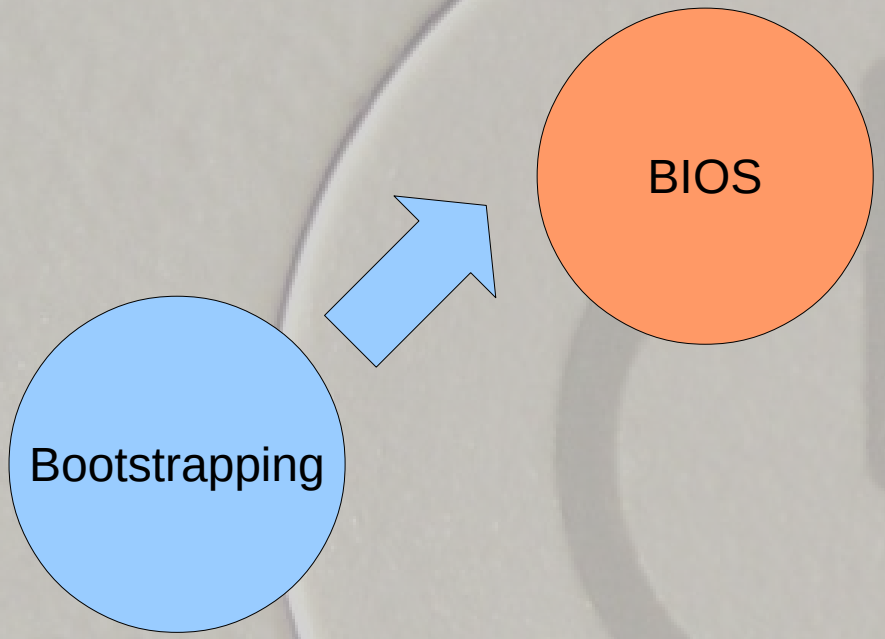
Bootstrapping

- “Am Schopfe aus dem Sumpf”
  - Dilemma: Computer brauch Software um zu laufen ↔ Um Software zu laden, muß Computer laufen
- Beim PC
  - Nach Einschalten beginnt Prozessor Ausführung von Firmware an festgelegter Adresse im ROM (heute meist Flash Memory)
  - Firmware ist meist BIOS

Die Start-Abfolge eines x86 Rechners



Vom Power On zum BIOS



BIOS

- BIOS ist im PC der Primary Boot Loader
- POST
  - Überprüfung von CPU, RAM, Grafikhardware etc.
  - Ausgabe von Fehlercodes über POSTcard, Beep Codes
- Initialisierung der Hardware
  - u.a. Konfiguration und Überprüfung von eingebauten Steckkarten

BIOS

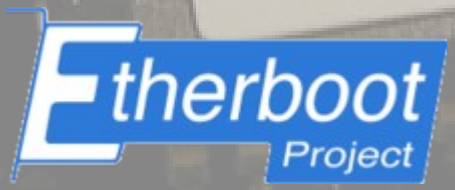
- Aufrufen von BIOS-Erweiterungen einzelner Subsysteme
  - z.B. SCSI Controler, RAID Controler, etc.
- Untersuchung von Speichergeräte auf Vorhandensein gültiger Bootsektoren (z.B. MBR bei Festplatte)
- Ausführen des Payloads im Bootsektor
  - Payload ist meist ein Boot-Loader
  - Kann aber auch anderes Programm sein

BIOS

- PXE
  - Preboot Execution Environment
  - PXE Code befindet sich im BIOS der Netzwerkkarte
  - Ermöglicht laden des Betriebssystems über das Netz
  - Vom einem DHCP-Server kommt die IP-Konfiguration
  - Ein PXE-Server stellt das Betriebssystem bereit
    - Wird mittels TFTP geladen

BIOS

- PXE
  - Gut geeignet für Thin Clients
  - Freie Implementationen
    - 

Linux Terminal Server Project
    - 

Etherboot

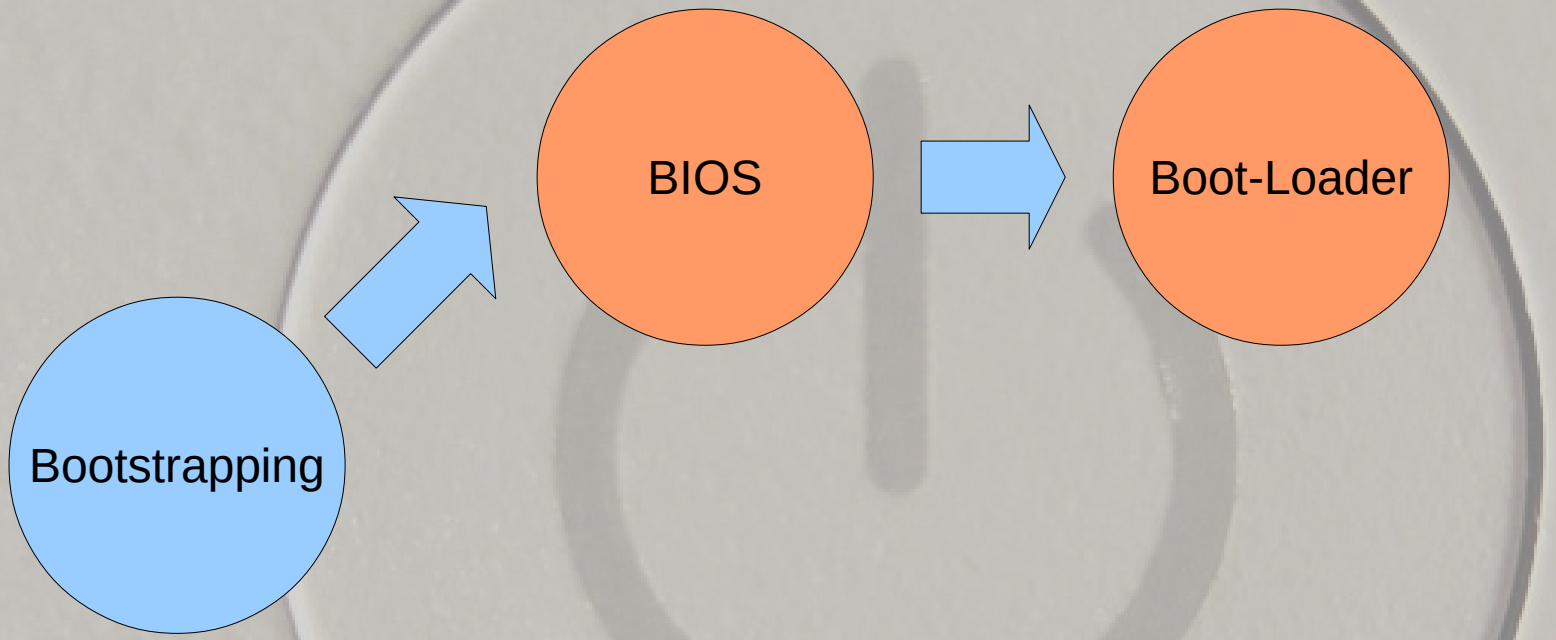
BIOS

- Freie BIOS Projekte

ist ein ein quelloffenes, freies BIOS unter der GPL, wenig Support für aktuelle Mainboards/Prozessoren, eher für Server und Embedded Systems

  - OpenBIOS ist eine freie, portable Implementierung des Open-Firmware-Standards für Firmware, unter der GPL
    - Kann im Zusammenspiel mit Coreboot verwendet werden
    - wenig geeignet für IA32 Systeme

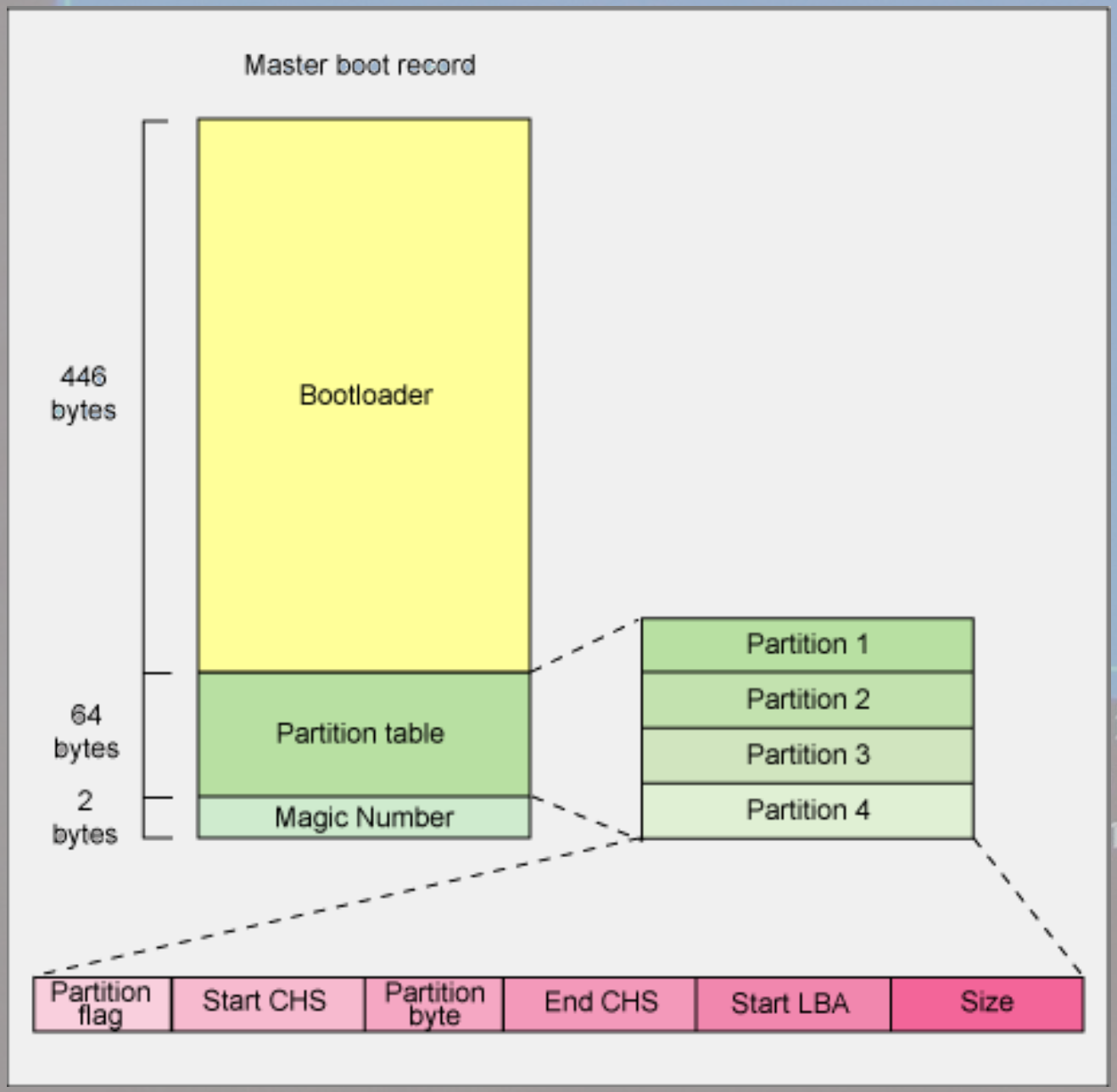
Vom BIOS zum Boot-Loader



Boot-Loader

- Second-stage Boot-Loader, im PC meist als Bootmanager bezeichet
- Grund für Booten in zwei Schritte
  - Größenbeschränkung des ROMs
  - Flexibilität
- Befindet sich im MBR der Festplatte
  - MBR meist zu klein für Bootmanager mit großem Funktionsumfang, nur 512 bytes (ein einziger Sector)
  - Wird umgangen durch Multi Stage Boot-Loader
- Boot-Loader kann weiteren Boot-Loader laden
  - Chain-Loading

Boot-Loader



- Aufbau des Master Boot Records
- 446 bytes code des Bootloaders
- 64 bytes Partitionstabelle
  - 4 Partitionen
- Endet mit einer 2 byte magic number (0xAA55)

Boot-Loader

- Aufgabe des Boot-Loaders im MBR (Stage) ist das Laden des Boot-Loaders auf der Platte (Stage 2)
- Letzte Stufe meist Laden und Starten des Kernels
- Aber auch anderer Payload möglich, z.B.
  - Memtest86(+)
  - PC Spiele
    - Viele Spiele bis Ende der 80er Jahre brachten ihr eigens Betriebssystem mit

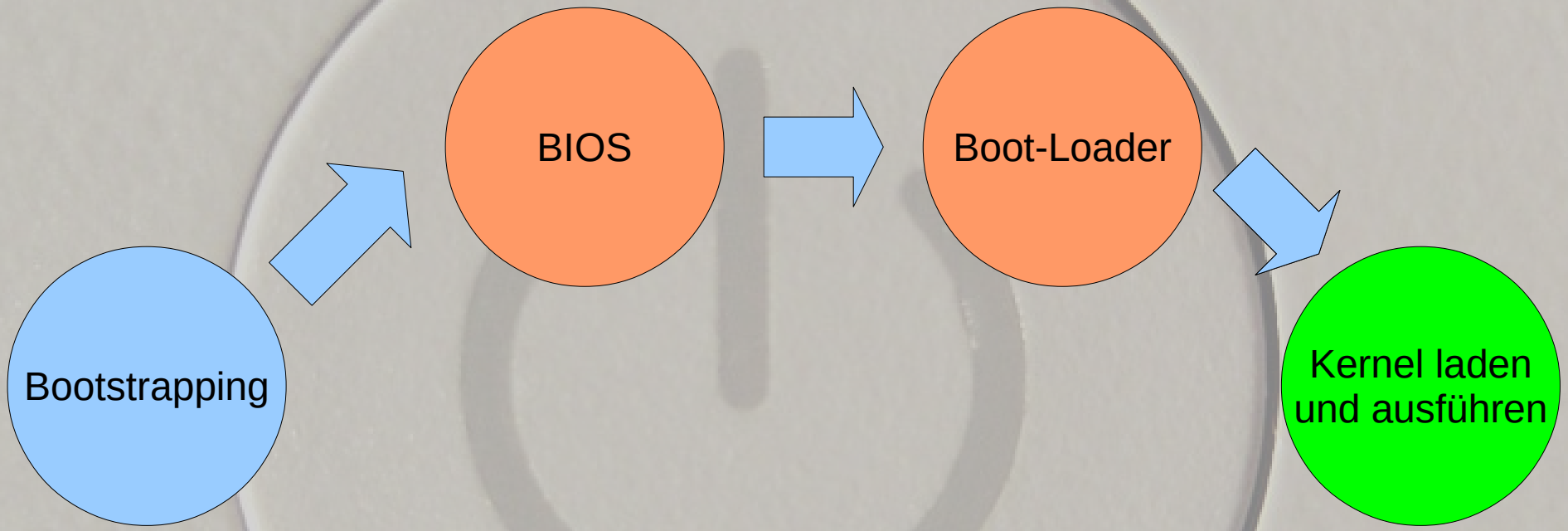
Boot-Loader

- Liste von Bootmanagern
  - Grub/Grub2
    - Bootloader des GNU Projekts, heute bei vielen großen Distributionen Standard
    - Sehr flexibel, kennt File Systeme
      - 3 Stage-Bootmananger; Stage 1.5 versteht dann File Systeme
  - Lilo
    - Früher Standard
    - Nachteil: Kennt keine File Systeme und lädt Daten mit einem rohen Offset von der Platte
    - Daher müssen Änderungen neu in den MBR geschrieben werden, damit Offset dort bekannt ist

Boot-Loader

- Liste von Bootmanagern
  - SYSLINUX-Projekt
    - ISOLINUX
      - Zum Starten von ISO 9660 Dateisystemen (CDROM)
    - SYSLINUX
      - Zum Starten von FAT Dateisystemen
      - Startet z.B. aus Windows heraus
  - U-Boot
    - Läuft auf vielen Microcontrolern, daher populär auf Embedded Systems
  - Viele, viele andere …
    - Siehe z,B. Liste bei http://de.wikipedia.org/wiki/Boot_Loader
  - Für die (Noch-)Windows User: NT-Loader und

Vom Boot-Loader zum Kernel



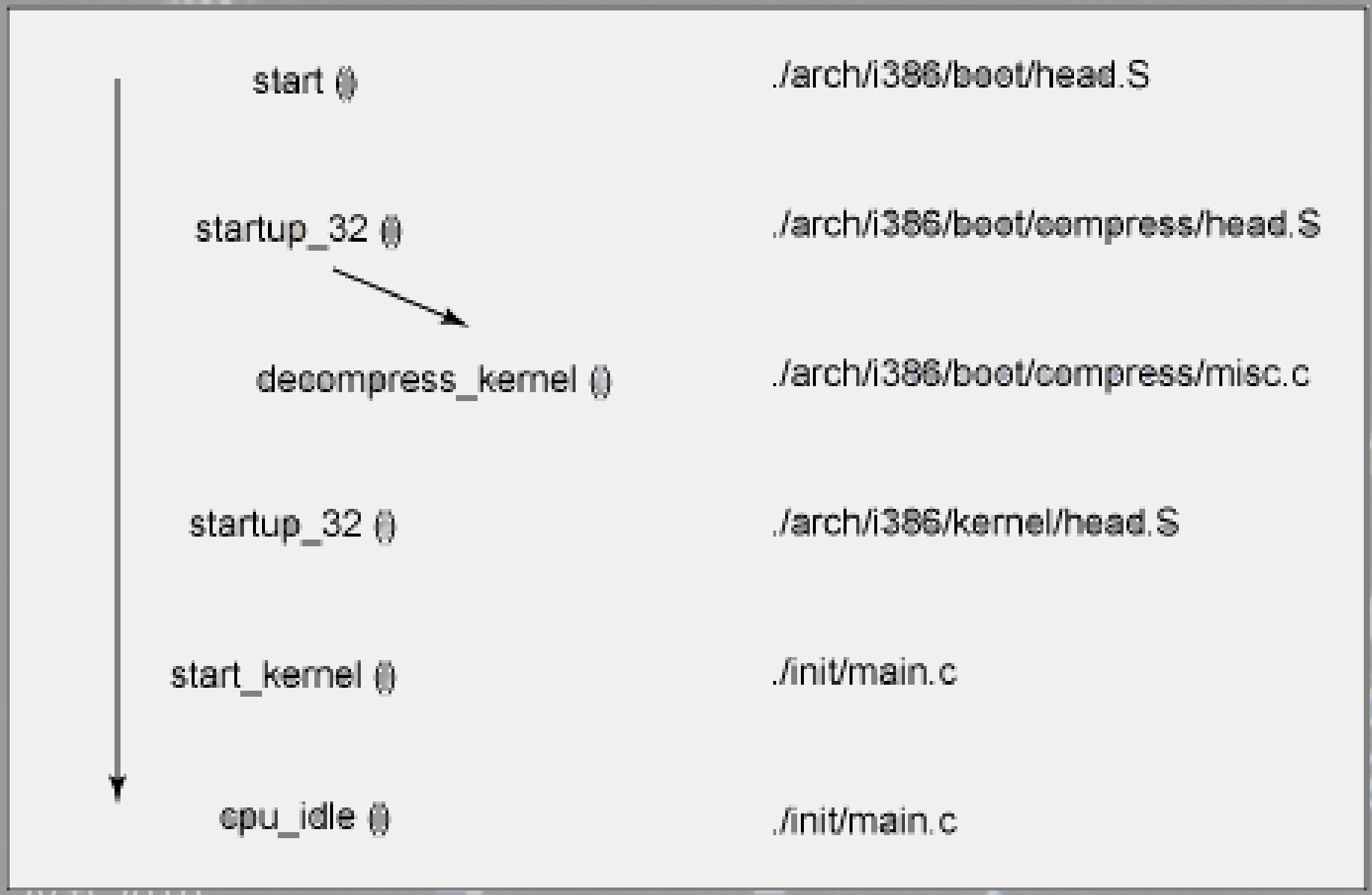
Kernel laden und ausführen

- Boot-loader läd in seinem letzten Schritt den Kernel
  - Kenrl-Image und Initial RAM-Disk werden in den Speicher geladen
  - Kernel-Parameter werden mit übergeben
- Kernel Image
  - Zlib-komprimiertes Image (zImage < 512 kB, bzImage > 512 kB)
  - Im Kopf des Images befindet sich eine Routine zum minimalen Hardware-Setup *start ()* und zur Dekomprimierung *decompress_kernel()*

Kernel laden und ausführen

- Nach Dekomprimierung wird start_kernel() aufgerufen
  - Enthält viele Initialisierungsfunktionen
  - Konfiguriert den Speicher
  - Lädt die Initial RAM-Disk
  - Ruft *kernel_thread() auf* (in arch/i386/kernel/process.c) um init (den ersten Prozess) zu starten
  - Started den Scheduler

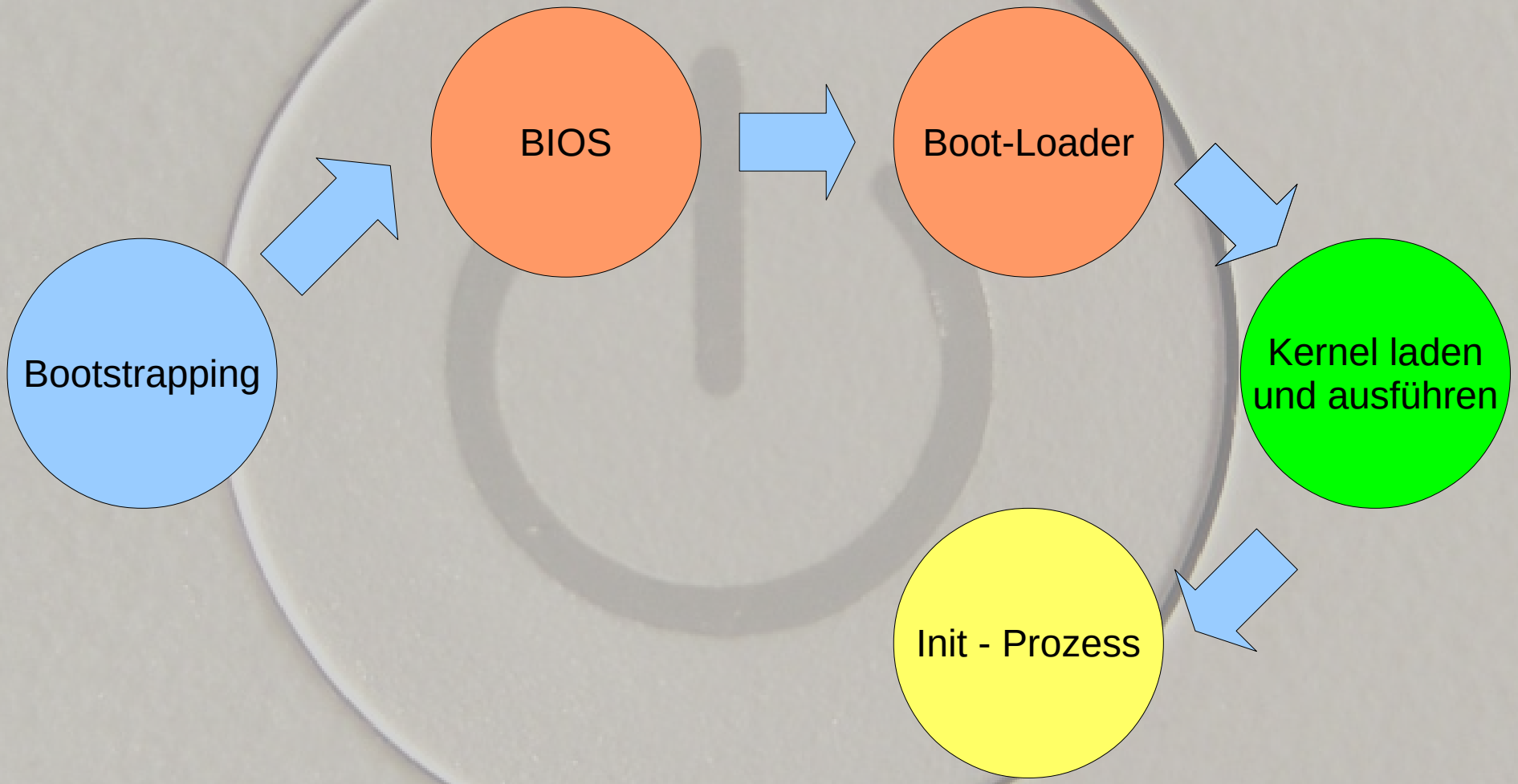
Kernel laden und ausführen



Kernel laden und ausführen

- Initial RAM-Disk
  - Ist ein komprimiertes cpio Archiv (im Fall von initramfs)
  - Enthält ein temporäres Root-Filesystem
  - Wird beim PC genutzt um Kernel-Module zu laden, bevor das eigentliche Root-Filesystem gemountet ist, d.h. auch Module, die für den späteren Disk-Zugriff auf das eigentliche RootFS nötig sind
  - Kernel ruft /init auf, Befehle werden ausgeführt, letzter Befehl ist meis das mounten des eigentlichen Root-Filesystems und das Starten von /sbin/init

Vom Kernel zum Init-Prozess



Init-Prozess

- Der init Prozess ist die erste User-Space Applikation (/sbin/init) und hat die ProzessID 1
- System V Init
  - /etc/inittab kontrolliert den Ablauf
  - Ruft je nach Runlevel spezifische Scripte auf, um Services (daemons) zu starten oder zu stoppen

Init-Prozess

- Runlevel
  - Es sind 8 Runlevel definiert
    - 0: Halt
    - 1: Single User Mode
      - Nur Superuser ist erlaubt, Maintainence Mode
    - 6: Reboot
    - 2-5: distributionsspezifisch, /etc/inittab definiert, was jeder Runlevel macht, initdefault bestimmt den Defaultrunlevel
    - S: wird nicht direkt benutzt, enthält die Skriptaufrufe um in Runlevel 1 zu gelangen
  - Der Default Runlevel ist distributionsspezifisch

Init-Prozess

- Beispiel (Debian) für Runlevel 5

/etc/rc5.d/S10sysklogd
/etc/rc5.d/S11klogd
/etc/rc5.d/S12acpid
/etc/rc5.d/S12dbus
/etc/rc5.d/S16ssh
/etc/rc5.d/S17mysql-ndb-mgm
/etc/rc5.d/S18mysql-ndb
/etc/rc5.d/S19lirc
/etc/rc5.d/S19mysql
/etc/rc5.d/S20acct
/etc/rc5.d/S20dirmngr
/etc/rc5.d/S20exim4
/etc/rc5.d/S20gpm
/etc/rc5.d/S20ifplugd
/etc/rc5.d/S20irda-utils
/etc/rc5.d/S20irqbalance
/etc/rc5.d/S20nfs-common
/etc/rc5.d/S20rsync
/etc/rc5.d/S20saned
/etc/rc5.d/S24dhcdbd
/etc/rc5.d/S24hal
/etc/rc5.d/S25bluetooth
/etc/rc5.d/S26network-manager
/etc/rc5.d/S50cups
/etc/rc5.d/S89cron
/etc/rc5.d/S91apache2
/etc/rc5.d/S99bootchart
/etc/rc5.d/S99kdm
/etc/rc5.d/S99rc.local
/etc/rc5.d/S99rmnologin
/etc/rc5.d/S99stop-bootlogd
/etc/rc5.d/K00isdnutils
/etc/rc5.d/K00vdr
/etc/rc5.d/K00vdradmin-am
/etc/rc5.d/K00virtualbox-ose

Init-Prozess

- In den Runlevelverzeichnissen sind symbolische Links enthalten
  - Zeigen auf ein Skriptverzeichnis
    - Bei Debian /etc/init.d/
  - Jenachdem, ob die Links mit S oder K beginnen werden die Skripte mit der start oder stop Funktion aufgerufen
  - Die Nummer bestimmt die Reihenfolge des Aufrufs
- /etc/inittab started eine konfigurierbare Anzahl an virtuellen Terminals (getty)

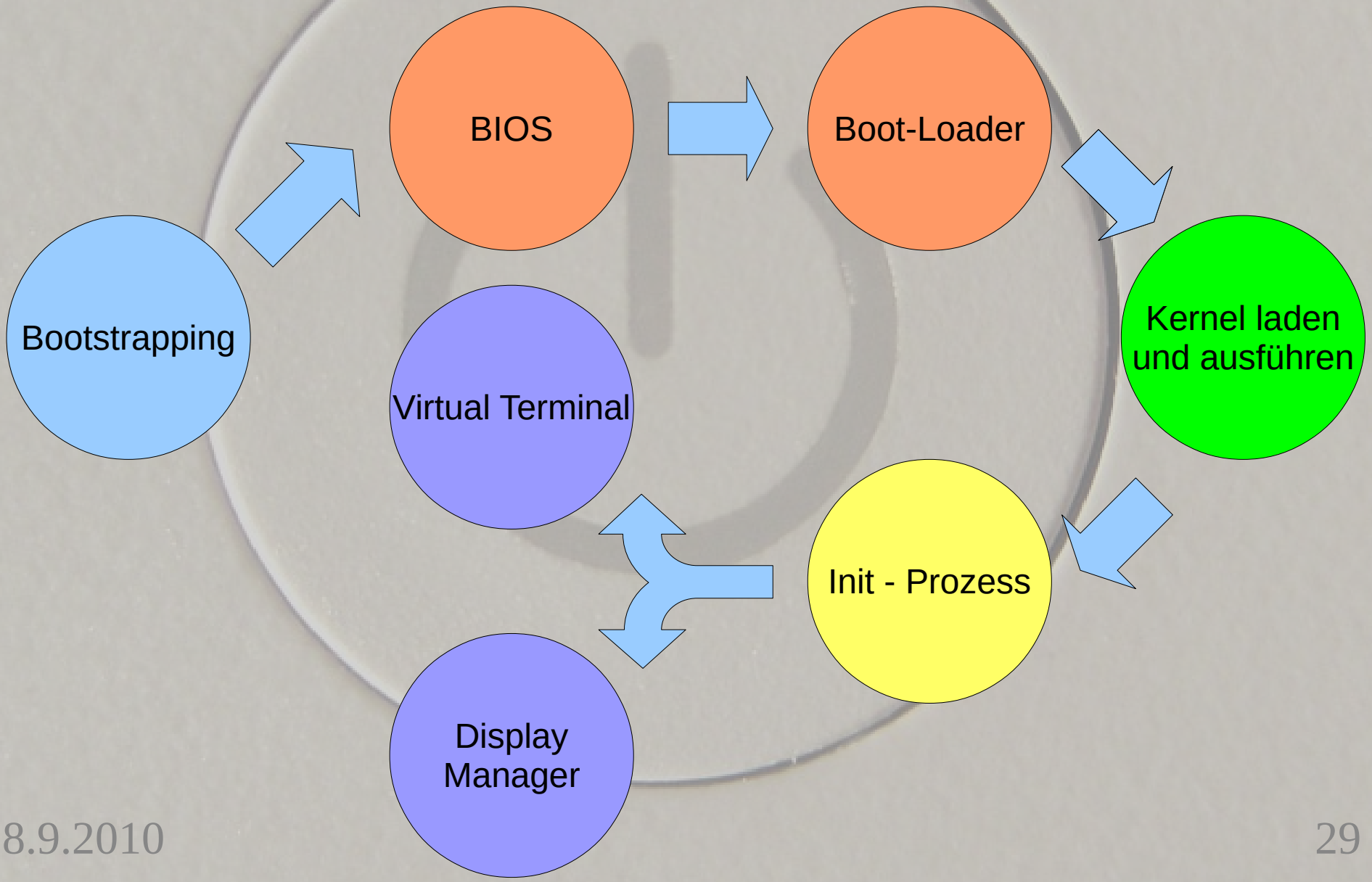
Init-Prozess

```
#!/bin/sh
# Start/stop the cron daemon.

test -f /usr/sbin/cron || exit 0
PIDFILE=/var/run/crond.pid

[...]

case "$1" in
start)  log_daemon_msg "Starting periodic command scheduler" "crond"
        start-stop-daemon --start --quiet --pidfile $PIDFILE --name cron --startas /usr/sbin/cron -- $LSBNAMES $EXTRA_
        log_end_msg $?
        ;;
stop)   log_daemon_msg "Stopping periodic command scheduler" "crond"
[…]
        ;;
restart) log_daemon_msg "Restarting periodic command scheduler" "crond"
[..]
        ;;
reload|force-reload) log_daemon_msg "Reloading configuration files for periodic command scheduler" "crond"
        # cron reloads automatically
        log_end_msg 0
        ;;
*)      log_action_msg "Usage: /etc/init.d/cron {start|stop|restart|reload|force-reload}"
        exit 2
        ;;
esac
exit 0
```

Init-Prozess

- In den Runlevelverzeichnissen sind symbolische Links enthalten
  - Zeigen auf ein Skriptverzeichnis
    - Bei Debian /etc/init.d/
  - Jenachdem, ob die Links mit S oder K beginnen werden die Skripte mit der start oder stop Funktion aufgerufen
  - Die Nummer bestimmt die Reihenfolge des Aufrufs

Die Start-Abfolge eines x86 Rechners



Terminals und DisplayManager

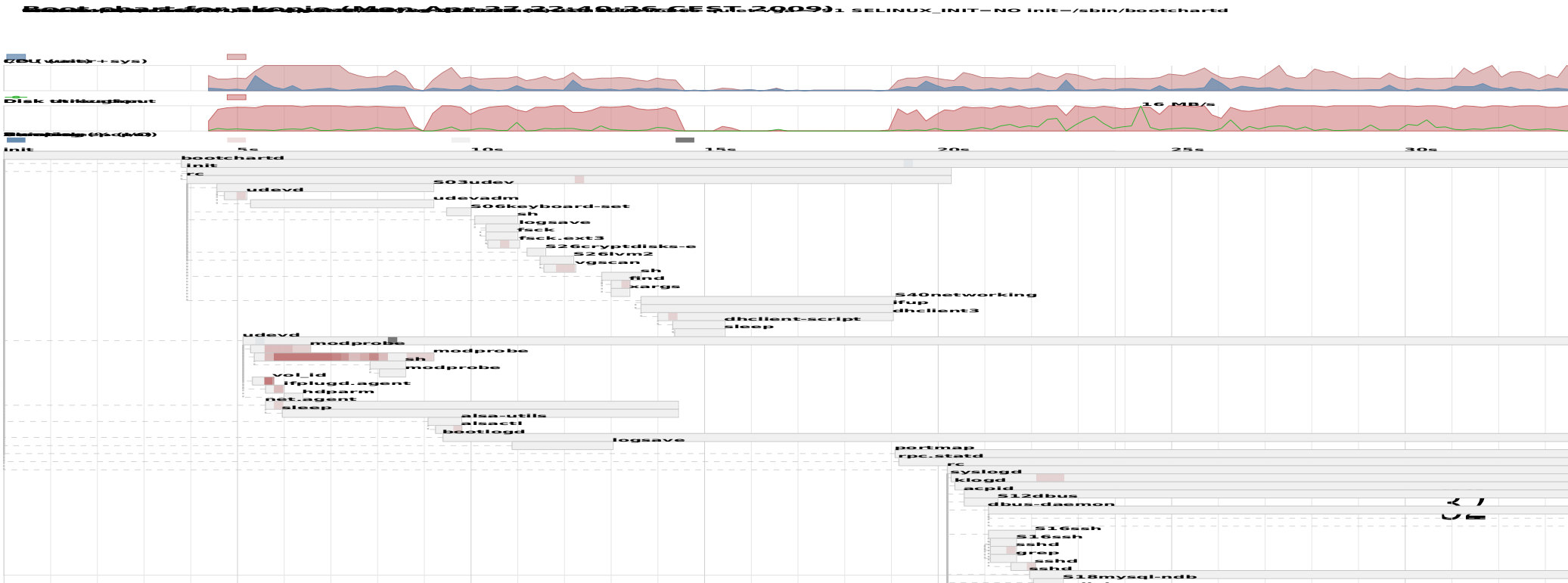
- Wie schon erwähnt, started init virtuelle Terminals
- Ob ein DisplayManager gestarted wird, bestimmen startupskripte
  - /etc/init.d/kdm startet und stoppt KDM
  - /etc/init.d/gdm startet und stoppt GDM

Boot-Optimierung

- Bootchart
  - http://www.bootchart.org/
  - Programm zur Performance Analyse und Visualisierung des Init-Prozesses
  - Sammelt Informationen über Prozesse, CPU Auslastung und Disk Aktivität über das /proc Filesystem
  - Wird dem Kernel als init – Prozess übergeben
    - Init=/sbin/bootchartd
    - Alternativer Init-Prozess mit bootchart_init=INIT

Boot-Optimierung

- Bootchart
  - Daten, die von bootchartd aufgezeichnet wurden, können mit bootchart in ein Performancechart übersetzt werden



Was bringt die Zukunft?

- BIOS
  - BIOS ist alt, älteste Codeteile schon 35 Jahre “dabei”, Flickschusterrei um neue Hardware zu unterstützen,
  - Extensible Firmware Interface (EFI) ist Nachfolger des BIOS
    - MACs mit Intel Prozessor benutzen breits EFI
- System V Init wird ersetzt durch
  - Upstart
    - derzeit in Ubuntu)
  - Systemd
    - Derzeit in Fedora, bald in Ubuntu (?)

Was bringt die Zukunft?

- Upstart
  - Basiert auf System V Init und ist abwärtskompatibel
  - Ist im Gengesatz zu Sys V Init asynchron
    - Ereignisorientierter Ansatz
  - Ist bei Ubuntu und Fedora schon länger Standard
  - Auch in Maemo, webOS und Chrome OS
  - http://upstart.ubuntu.com/

Was bringt die Zukunft?

- Systemd
  - Architekturanleihen bei MAC's launchd
  - Startupprozeduren werden parallel ausgeführt
  - Wird in einer der nächsten Fedora Versionen Upstart ersetzen
  - http://www.freedesktop.org/wiki/Software/systemd

Abschließendes

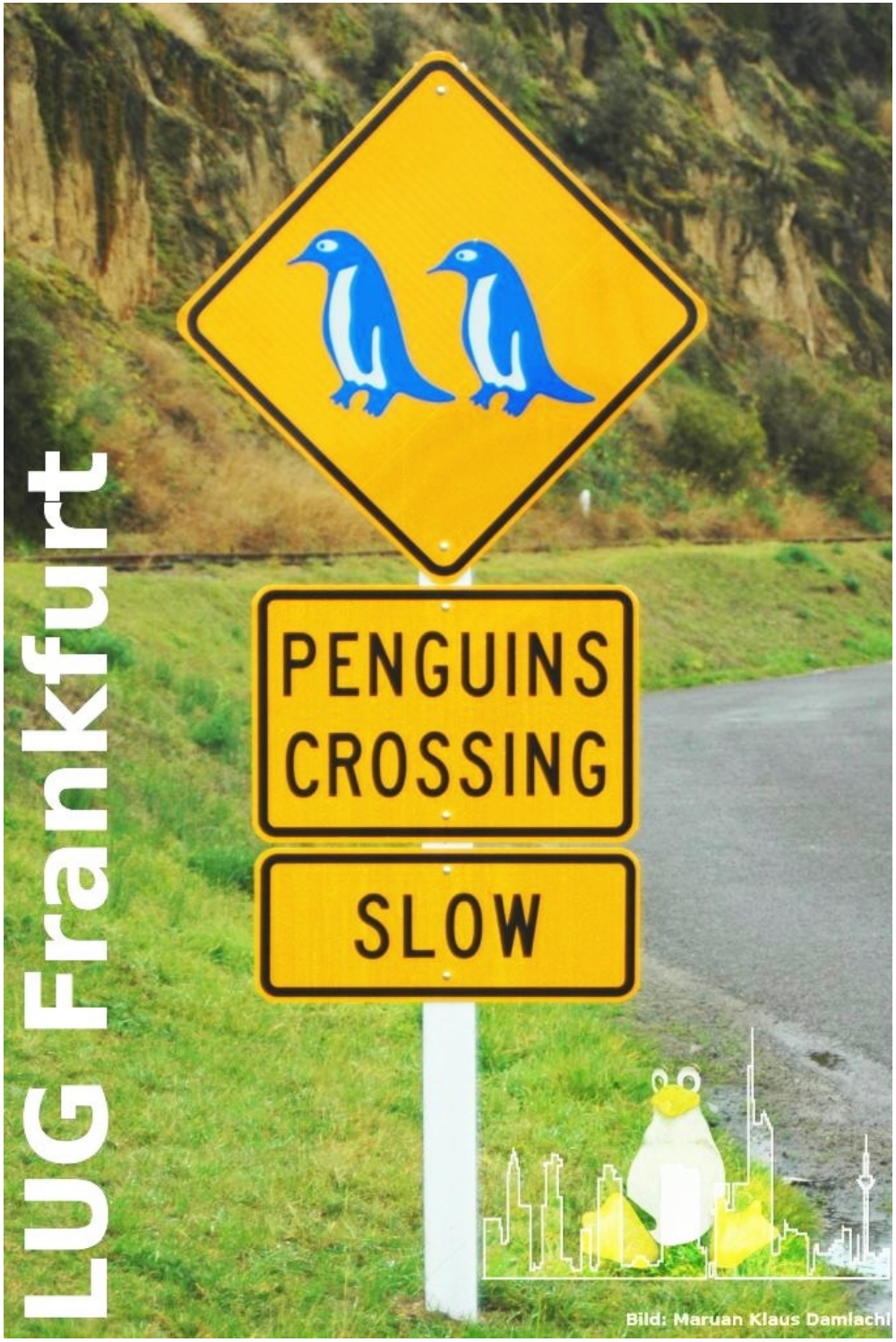
- Simple Praxistips zum schnelleren Booten
  - Timouts minimieren (BIOS, Bootloader)
  - Nicht benötigte Daemons nicht im Runlevel starten lassen
    - Skripte können deaktiviert werdens
  - Init-Profiling mit bootchart
    - anspruchsvoller

Abschließendes

- Graphischer Login
  - Ladezeit des Display Managers kann stark variieren
    - Auch vom ausgewählten Theme abhängig
  - Laden des Desktops kann sehr unterschiedlich sein
    - Auf automatisch gestartete Programme achten
- Bootvorgang “austricksen”
  - Suspend to RAM und Suspend to Disk benutzen

Danke für die Aufmerksamkeit!

Noch Fragen?



28.9.2010