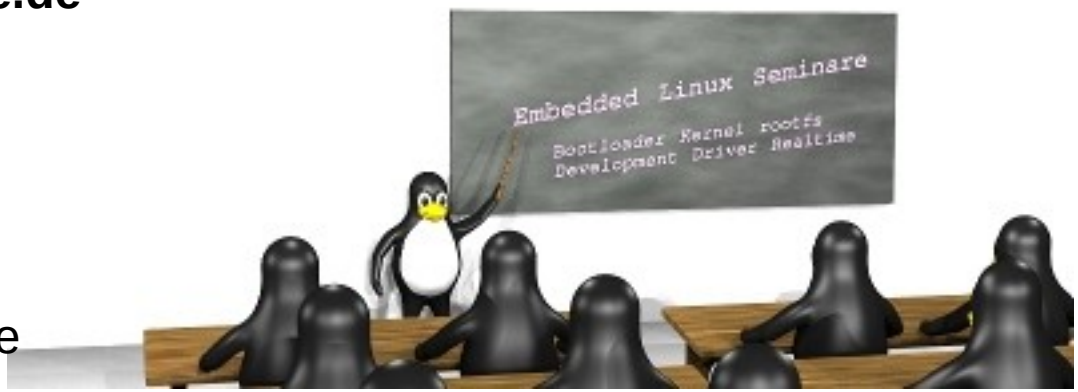
Embedded-Linux-Seminare

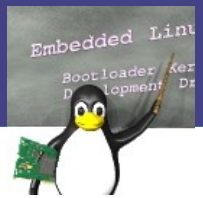
Einführung

http://www.embedded-linux-seminare.de

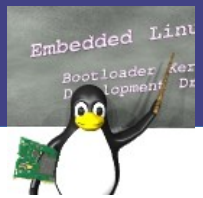
Diplom-Physiker Peter Börner
Spandauer Weg 4
37085 Göttingen

Tel.: 0551-7703465
Mail: info@embedded-linux-seminare.de



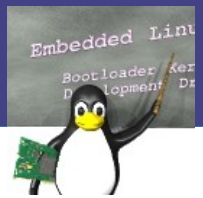


Kopier-Rechte



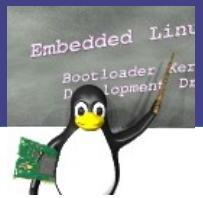
Einführung - Inhalt

- Was ist Embedded Linux?
- Komponenten eines Linux Systems

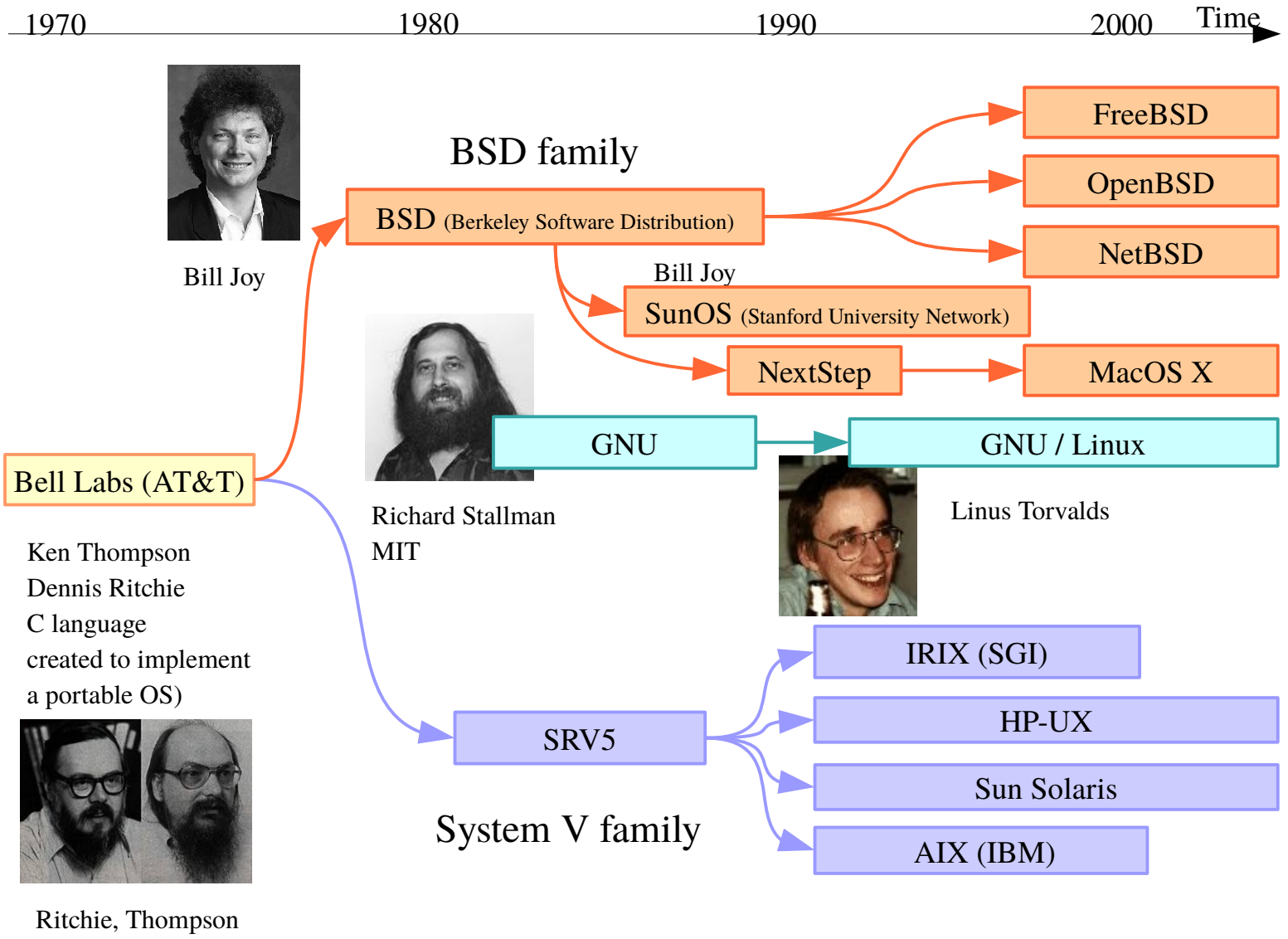


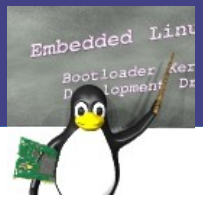
Einführung - Inhalt

- Was ist Embedded Linux?
  - Ein Überblick
  - Geschichtlicher Abriss
  - Entwicklungsprozess
  - Vergleich Desktop Linux – Embedded Linux



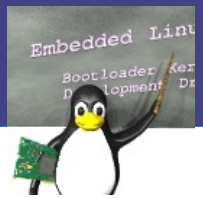
UNIX Stammbaum





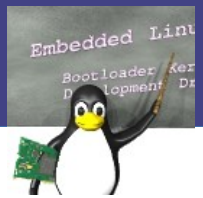
UNIX Philosophy

- Die modernen leistungsstarken Systeme basieren auf einem über 35 Jahre alten Design
  - Small is beautifull
  - Ein Programm sollte nur eine Aufgabe übernehmen
  - Portabilität geht vor Effizienz
  - Alles ist eine Datei
- Systemaufbau
  - Kernel: Hardware Layer
  - Shell: Text-Mode Layer
  - X-Server: GUI Layer



UNIX Features

- UNIX ist für große Mehr-Benutzer Mainframes entwickelt worden
  - Multi-User:
    Die Benutzer können nur auf ihre eigenen Dateien zugreifen. Sie haben keinen Zugriff auf das eigentliche Betriebssystem.
  - root:
    root ist der Administrator des Systems und hat alle Zugriffsrechte.
  - Preemptive Multitasking
  - Unterstützung von Mehrprozessor-Systemen
  - Sehr flexible
  - Netzwerk Unterstützung
  - Portierbarkeit
  - Skalierbarkeit



UNIX System Architektur

Graphische Applikationen
Firefox, Gimp, OpenOffice, ...

Kommandozeilen Tools
Shell, cp, mv, find, ls, ...

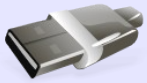
Shared Libraries
Libjpeg, libstdc++, libxml, ...

C-Library
GNU C Library, µClibc, ...

User-Space

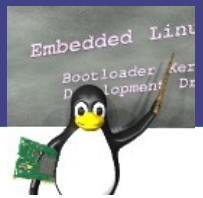
Kernel

Kernel-Space

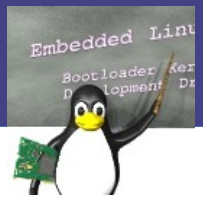

Hardware und Peripherie

Hardware



Das GNU Project

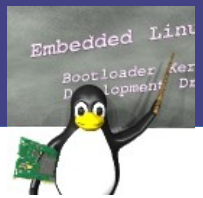
- GNU = **G**NU is **N**ot **U**nix
- Projekt um ein freies UŃIX ähnliches Betriebssystem zu schaffen
- 1984 initialisiert durch Richard Stallman, Forscher am MIT.
- Grundlegende Bestandteile:
  - C Compiler (gcc),
  - GNU make
  - Emacs
  - C-Library (glibc)
  - Coreutils (ls, cp, …)
- 1991 fehlte allerdings noch ein Kernel und die GNU Tools liefen auf proprietären UNIX Systemen.



Und dann fing alles an

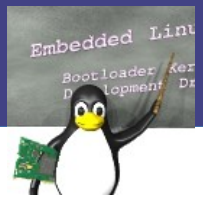
```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -
I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones. This has been brewing
since april, and is starting to get ready. I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things).
I've currently ported bash(1.08) and gcc(1.40), and things seem to work.
This implies that I'll get something practical within a few months, and
I'd like to know what features most people would want. Any suggestions
are welcome, but I won't promise I'll implement them :-)
Linus (torvalds@kruuna.helsinki.fi)
PS. Yes - it's free of any minix code, and it has a multi-threaded fs.
It is NOT protable (uses 386 task switching etc), and it probably never
will support anything other than AT-harddisks, as that's all I have :-(.
```



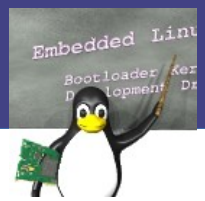
Linux und Open-Source

- Linux ist quelloffen
  - Der Quellcode des Kernels und zahlreicher Bibliotheken und Applikationen steht zur Verfügung und darf kopiert und verändert werden.
  - Das ermöglicht vielen Entwicklern weltweit, sich an der Weiterentwicklung vom Kernel und von Bibliotheken und Applikationen zu beteiligen.
  - Linux wird von einer weltweiten Entwicklergemeinde ständig weiterentwickelt.
  - Die Entwickler arbeiten umsonst oder sind bei Firmen angestellt.

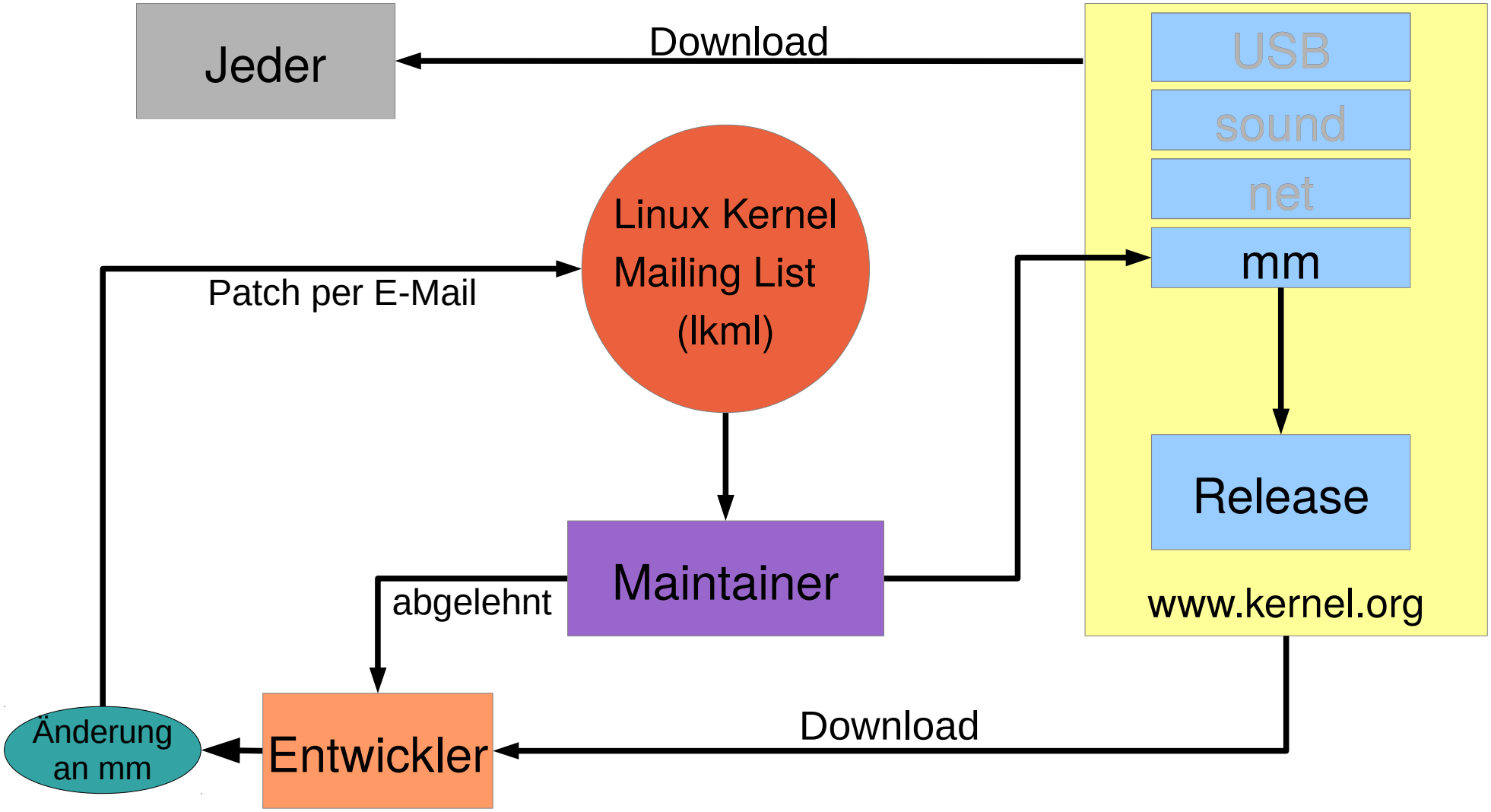


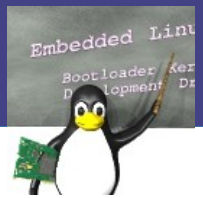
Linux und Open-Source

- „Wenn soviele daran mitarbeiten, kann doch jeder alles verändern“
  „Kann denn das überhaupt funktionieren und stabil laufen?“
- Es läuft stabil!!! Denn es ist kein Chaos sondern sehr strukturiert.



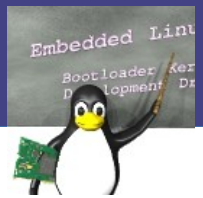
Kernel Entwicklungsprozess





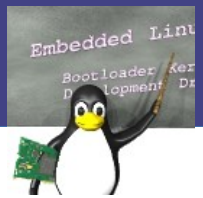
Was ist „Embedded“ Linux

- Was ist der Unterschied zwischen einem Linux für Desktop Rechner oder Server und einem Embedded-Linux?
  - Es werden die selben Quellen genutzt
  - Es werden die selben Compiler und Entwicklungswerkzeuge genutzt
  - Es werden die selben Tools und Applikationen verwendet
  - Die Konfigurationen sind auf das Embedded-System angepasst
  - Bootzeiten sind optimiert
  - Ressourcenverbrauch ist optimiert



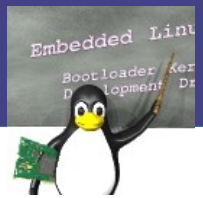
Komponenten eines Linux-Systems

- Die Komponenten eines Linux-Systems
  - Bootloader
  - Kernel
  - Root-Filesystem



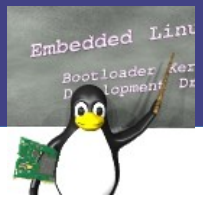
Bootloader

- Initialisiert die Hardware
- Startet eine Applikation (z.B. Kernel)
- Übernimmt in Embedded Systemen noch zusätzliche Aufgaben
  - Update
  - Verifizierung des Systems
  - Satus-Anzeigen
- Je nach Architektur wird der Bootloader unterschiedlich gestartet
  - x86: Das BIOS sucht nach einem bootfähigen Medium (Festplatte) und startet den Code im Master-Boot-Record (MBR)



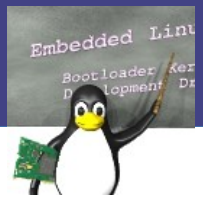
Bootloader

- Auf nicht x86 Architekturen ist der Low-Level Boot Prozess von der CPU und dem Board abhängig.
  - Einige Boards haben NOR Flash von dem aus die CPU Instruktionen nach einem Reset ausführt. Der Bootloader muss hier direkt in das NOR an eine spezifizierte Stelle geflsht sein.
  - Einige CPUs haben einen integrierten Bootcode ROM. Dieser lädt automatisch einen schmalen Bereich eines DataFlash oder NAND Flash ins statische RAM. Hierbei ist ein minimaler First Stage Bootloader notwendig, der den eigentlichen Bootloader lädt.
- Der Bootloader in nicht x86 Systemen startet direkt nach dem CPU Reset. Er muss deshalb alle benötigten Geräte und den Memory Controller initialisieren um auf das DRAM zugreifen zu können.



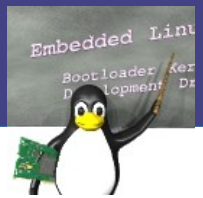
Bootloader Varianten

- Es gibt zahlreiche Bootloader für verschiedene Architekturen
- Zwei sind in Embedded Systemen sehr verbreitet
  - GRUB, Grand Unified Bootloader für x86-Architekturen
    http://www.gnu.org/software/grub/
    - Konfiguration über Dateien
    - Command Shell
    - Kommunikation über serielle Konsole möglich
    - Unterstützung für viele Dateisysteme
    - GPL
  - Das U-Boot für ARM, PPC, Coldfire, …
    - Konfiguration über Environment Variablen
    - Command Shell
    - Skript fähig
    - Download von Dateien (z.B. Kernel) über Netzwerk, Serielle Schnittstelle oder USB
    - Schreiben von Flash Speicher
    - GPL



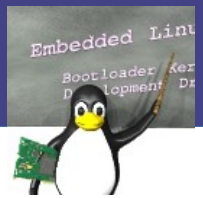
Kernel

- Der Kernel
  - stellt Umgebung zur Ausführung von Applikationen zur Verfügung
    - Scheduler, Prozess Management, System Calls
  - verwaltet die Ressourcen
    - Speicher, Prozessoren, Zugriff auf die Devices
  - verwaltet den Userzugriff
    - Zugriffskontrolle und gegenseitige Abgrenzung
  - managt Datenaustausch und Datensicherung
    - Netzwerk und I/O-Infrastruktur
    - File-System Verwaltung



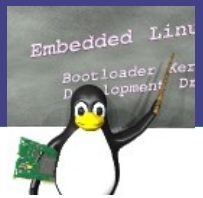
Kernel Features

- Portabilität und Hardware Unterstützung
  Läuft auf den meisten Architektueren
- Skalierbar
  Läuft auf Super-Computern und auf kleinen Embedded Geräten (4 MB Arbeitsspeicher sind ausreichend)
- Unterstützt Standards und Kompatibilität
- Vollständige Netzwerk Unterstützung
- Sicherheit
  Es kann seine Mängel nicht verbergen. Der Code ist von zahlreichen Experten überprüft worden.
- Stabilität und Beständigkeit
- Modularität
- Einfach zu programmieren
  Man kann vom existierenden Code lernen. Viele nützliche Ressourcen im Netz



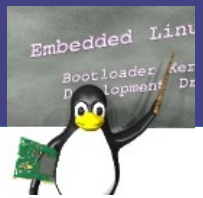
Kernel Features

- Multi-User:
  - Mehrere Benutzer können unabhängig voneinander arbeiten
  - Abgrenzung der Ressourcen und Applikationen der einzelnen Benutzer.
- Multi-Tasking / Multi-Threading:
  - Es können mehrere Prozesse/Threads unabhängig voneinander laufen.
  - Jeder Prozess hat seine eigene virtuelle Umgebung.
  - Scheduler kümmert sich um die Zuweisung der CPU Zeit
- Multi-Prozessor
- Multi-Architektur
  - Architektur kann zur Compile-Zeit festgelegt werden
  - Interface zum Userland bleibt gleich



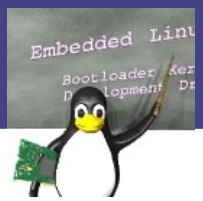
Kernel und unterstützte Architekturen

- Der Kernel unterstützt eine wachsende Anzahl Architekturen.
  Im Linux-Quellcode Pfad unter */arch* zu finden.
- Minimale Vorraussetzung:
  - 32 Bit Prozessor
  - Gcc muss Prozessor unterstützen
- MMU lose CPUs werden auch unterstützt.
- 32 Bit Architekturen (*/arch* Unterverzeichnisse)
  arm, avr32, blackfin, cris, frv, h8300, m32r, m68k, m68knommu, microblaze, mips, mn10300, parisc, s390, sparc, um, xtensa
- 64 Bit Architekturen
  alpha, ia64, sparc64
- 32/64 Bit Architekturen
  powerpc, x86, sh

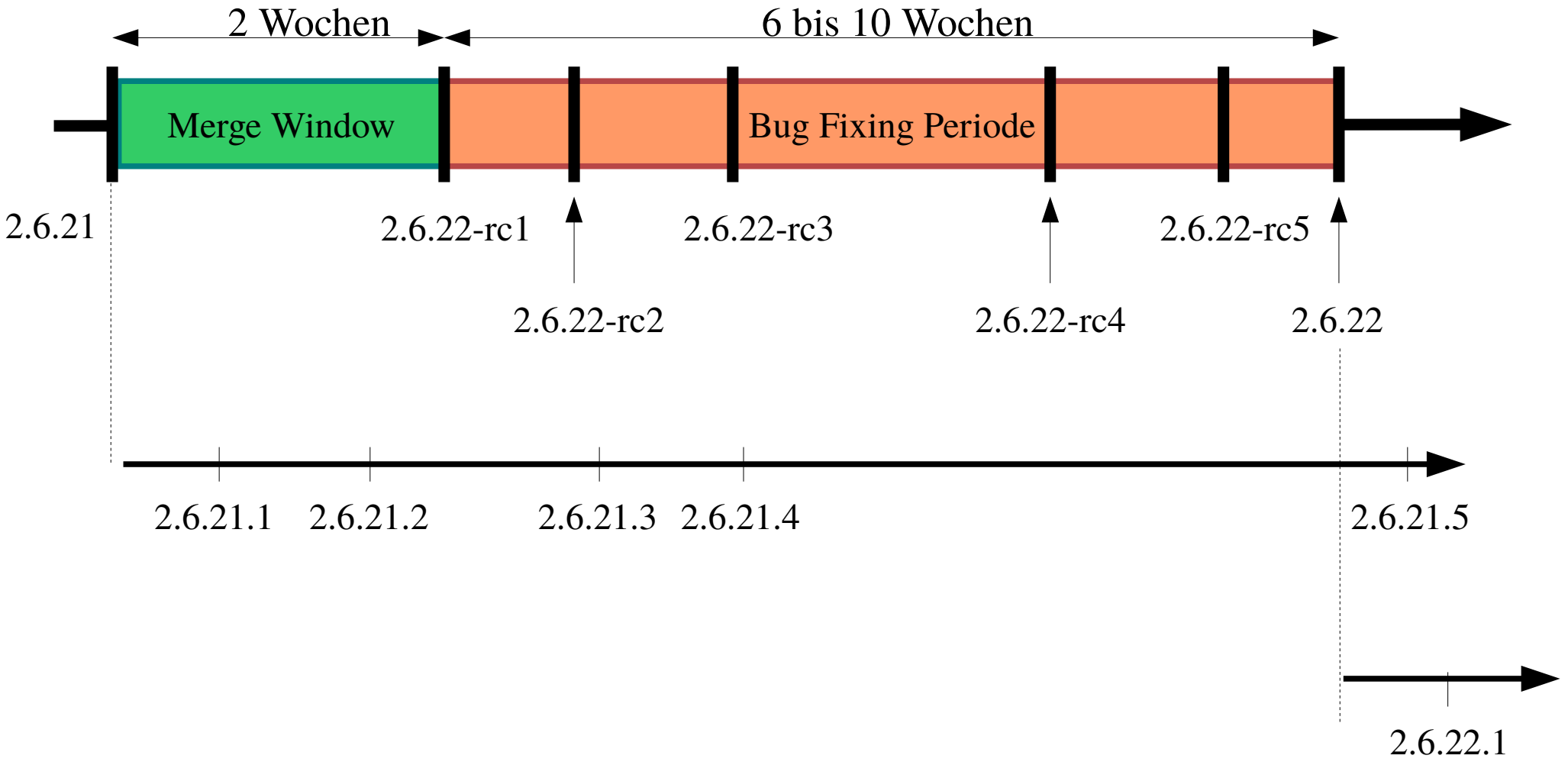


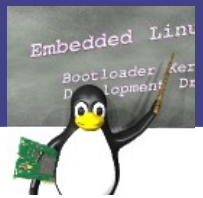
Kernel Versionierung bis 2.6

- Alle 2 bis 3 Jahre ein stabiler Major Branch
  - Identifiziert durch eine gerade mittlere Zahl
  - Beispiele: 1.0, 2.0, 2.2, 2.4
- Ein Entwicklungs-Branch um neue Funktionalitäten und wichtige Änderungen zu integrieren
  - Identifiziert durch eine ungerade mittlere Zahl
  - Beispiele: 2.1, 2.3, 2.5
  - Nach einiger Zeit wird die Entwicklungsversion zur Basis Version für den Stable-Branch
- Ab und zu Minor Release: `2.2.23, 2.5.12`, etc.



Kernel Versionierung 2.6

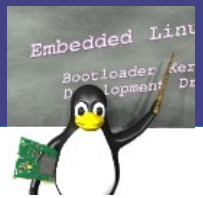




Kernel Zeitleiste

Zweig	Version	Veröffentlichung	Dateien	Quellcode-Zeilen	Größe in kB	Bemerkungen
	0.01	17. September 1991	88	8413	230	erste Veröffentlichung
1.0	1.0.0	13. März 1994	563	170581	1259	
1.1	1.1.0	6. April 1994	561	170320	1256	
	1.1.95	2. März 1995			2301	Entwicklungsversion
1.2	1.2.0	7. März 1995	909	294623	2301	
	1.2.13	2. August 1995			2355	
1.3	1.3.0	12. Juni 1995	992	323581	2558	
	1.3.100	10. Mai 1996			5615	Entwicklungsversion
2.0	2.0.0	9. Juni 1996	2015	716119	5844	
	2.0.40	8. Februar 2004			7551	
2.1	2.1.0	30. September 1996	1727	735736	6030	
	2.2.0-pre9	21. Januar 1999			13077	Entwicklungsversion
2.2	2.2.0	26. Januar 1999	4599	1676182	13080	
	2.2.26	24. Februar 2004			19530	
2.3	2.3.0	11. Mai 1999	4721	1763358	13804	
	2.3.99-pre9	23. Mai 2000			20882	Entwicklungsversion
2.4	2.4.0	4. Januar 2001	8187	3158560	24379	letzter noch gepflegter Kernel-Zweig vor 2.6; die Unterstützung soll beendet werden wenn ein Jahr lang keine Aktualisierung des Zweiges mehr erforderlich war, momentan wäre dies im Dezember 2011 der Fall
	2.4.37	2. Dezember 2008			38735	**Aktuelle Version: 2.4.37.11 vom 18. Dezember 2010**
2.5	2.5.0	23. November 2001	9893	3833603	29405	
	2.5.75	10. Juli 2003			40969	Entwicklungsversion
2.6	2.6.0	18. Dezember 2003			41614	
	2.6.38	15. März 2011	35877	14294439	94144	

Quelle: http://de.wikipedia.org/wiki/Linux_Kernel



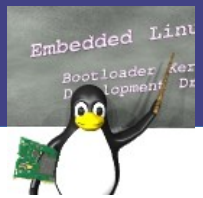
Kernel Website

Protocol	Location
HTTP	http://www.kernel.org/pub/
FTP	ftp://ftp.kernel.org/pub/
RSYNC	rsync://rsync.kernel.org/pub/

Latest Stable Kernel:

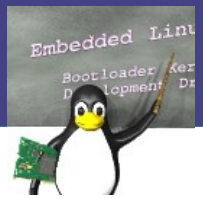
2.6.38.6

linux-next:	**next-20110512**	2011-05-12		[Patch]	[View Patch]		[Gitweb]	
snapshot:	**2.6.39-rc7-git3**	2011-05-12		[Patch]	[View Patch]			
mainline:	**2.6.39-rc7**	2011-05-10	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
stable:	**2.6.38.6**	2011-05-09	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
stable:	**2.6.37.6**	2011-03-27	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
stable:	**2.6.36.4**	2011-02-17	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
longterm:	**2.6.35.13**	2011-04-28	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
stable:	**2.6.35.9**	2010-11-22	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
longterm:	**2.6.34.9**	2011-04-17	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
longterm:	**2.6.33.13**	2011-05-09	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
longterm:	**2.6.32.40**	2011-05-09	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
stable:	**2.6.32.28**	2011-01-07					[Gitweb]	
longterm:	**2.6.27.59**	2011-04-30	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
stable:	**2.6.27.57**	2010-12-09	[Full Source]	[Patch]	[View Patch]	[View Inc.]	[Gitweb]	[Changelog]
stable:	**2.4.37.11**	2010-12-18	[Full Source]	[Patch]	[View Patch]		[Gitweb]	[Changelog]



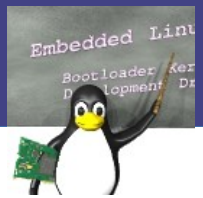
Root Filesystem

- Überblick
- File Hierarchie Standard
- Zugriffsrechte
- Gerätерepräsentation
- Spezielle Filesysteme
- Journaling Filesystem
- Flash Filesysteme



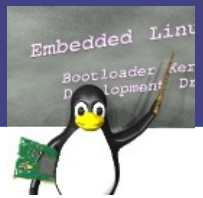
Filesystem - wozu?

- Ein Filesystem
  - sichert Daten an definierten Stellen
  - verhindert ungewollten Zugriff
  - erlaubt Modifikationen
  - ermöglicht das Einbinden weiterer Geräte (USB-Stick, DVD)
  - ermöglicht das Einbinden von Verzeichnissen auf anderen Rechnern
  - Stellt Daten-Konsistenz nach Absturz sicher



Features Embedded Filesysteme

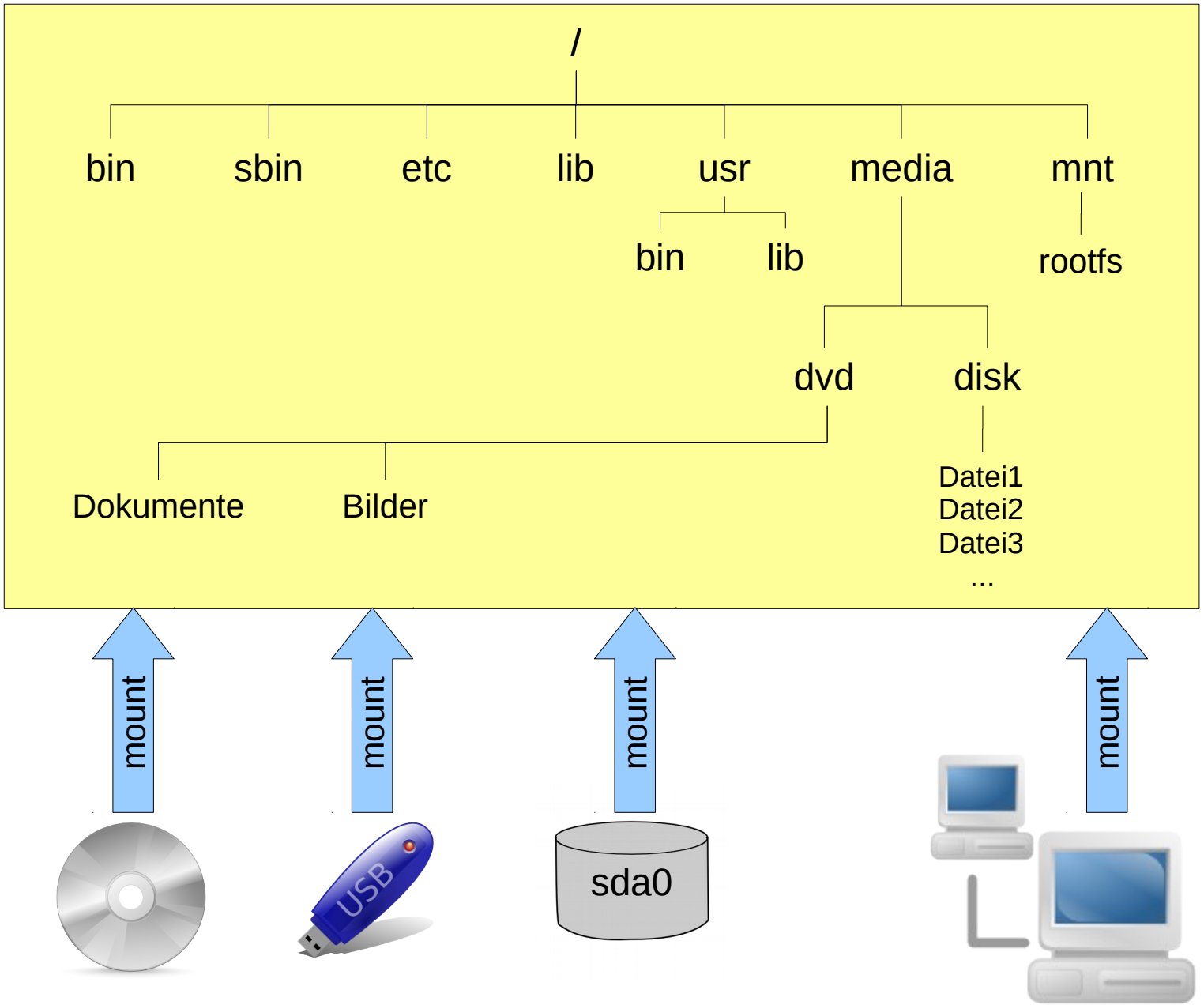
- XIP, Execute In Place. Bei sehr wenig Arbeitsspeicher
- Flash Unterstützung
  - Flash Filesystem
  - Journaling
  - Wear Leveling
  - jffs2, yast, ubifs
- Nur lesbar gemountetes Root-Filesystem
  - Sicherung der Datenkonsistenz bei Power-Failure

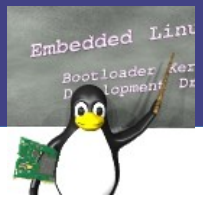


Virtuelles Filesystem

- Linux Rechner besitzen ein virtuelles Filesystem.
- Alle Geräte, Objekte und Verzeichnisse sind dort eingegliedert
- Neue Geräte können zur Laufzeit hinzugelinkt werden
- Kernel mountet das Root-Filesystem
- Das Root-Filesystem enthält Applikationen, Bibliotheken und Konfigurationsdateien für den weiteren Betrieb des Systems.
- initrd Initial Ram-Disk
  - Falls der Kernel z.B. nachladbare Module braucht um die Festplatte mit dem Root Filesystem zu mounten.
  - Muß vom Bootloader unterstützt werden
  - Komprimiertes Root-Filesystem
  - Wird in den Arbeitsspeicher geladen
  - Wird nach dem Booten durch echtes Root-Filesystem ersetzt

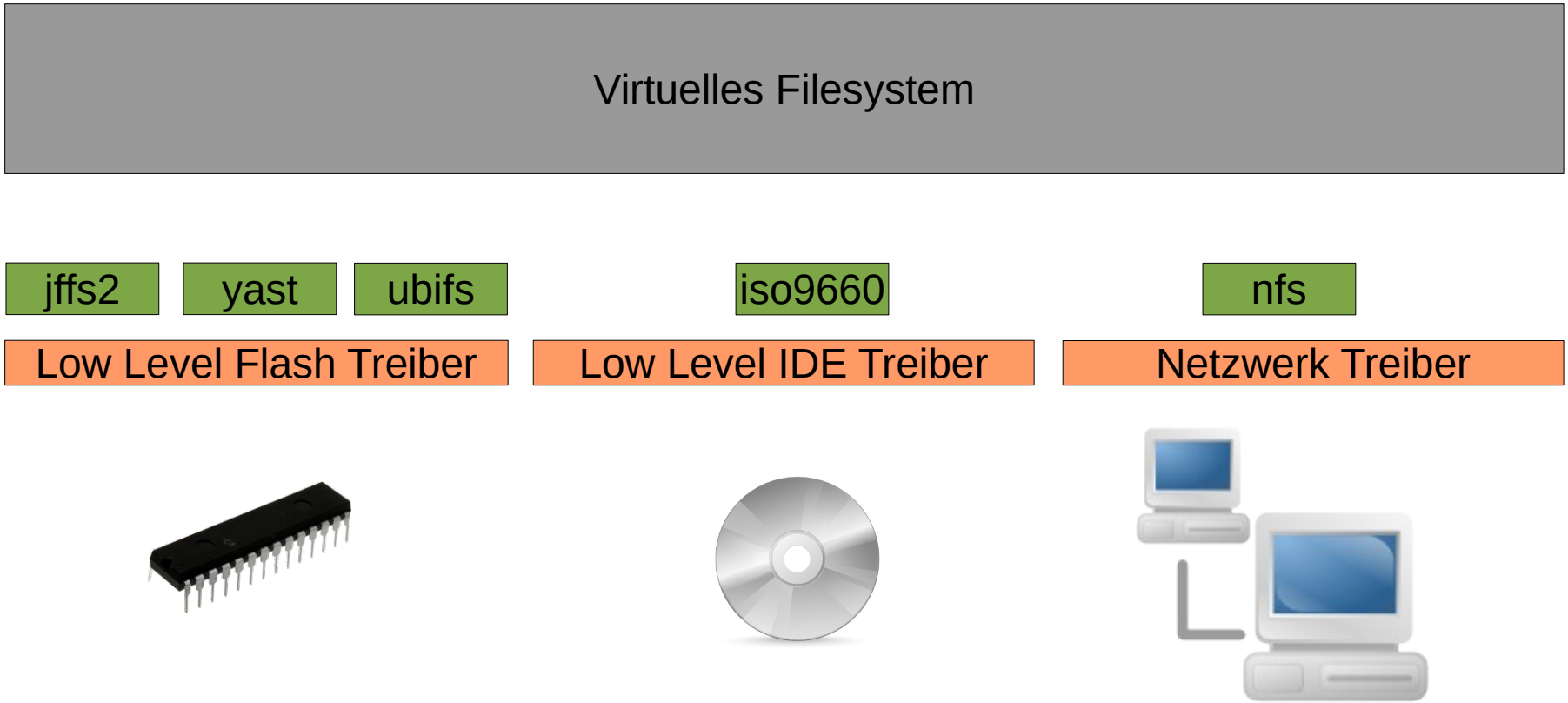
Virtuelles Filesystem

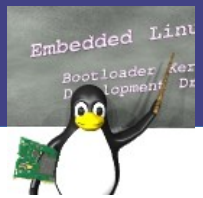




Virtuelles ↔ Reales Filesystem

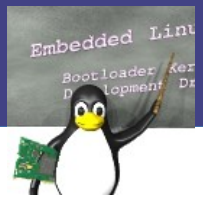
- Virtuelles Filesystem braucht reale Filesysteme zur Kommunikation mit dem Medium.
- Treiber müssen vom Kernel zur Verfügung gestellt werden
- Filesystem Treiber sind groß!!!



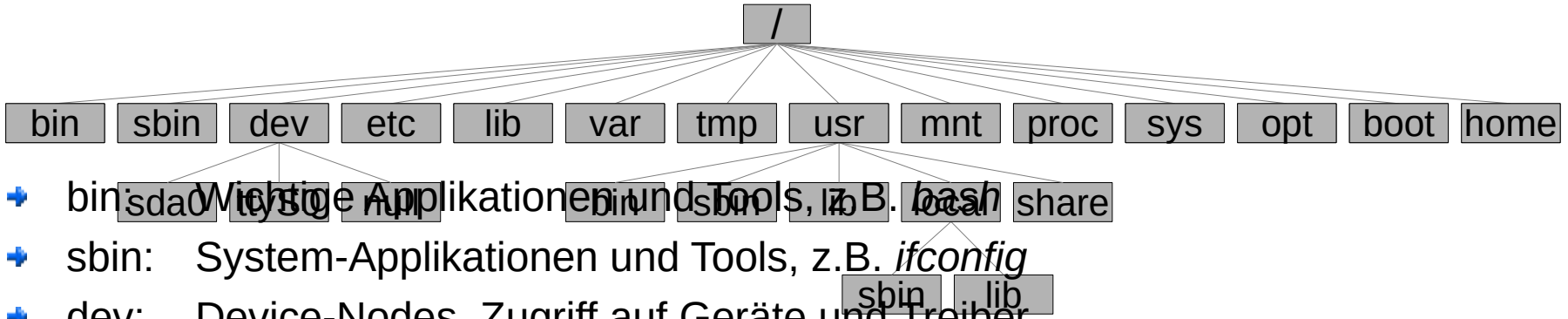


Reale Filesysteme

- Standard Filesysteme
  - ext2, ext3, ext4 — Festplatten, CF-Karten, etc. Linux-Standard
  - VFAT — Kompatibilität und Austausch mit Windows Systemen (USB-Stick)
  - ISO9660 / UDF — CD-Rom, DVD
  - tmpfs — RAM Disks
- Filesysteme für Embedded Geräte
  - jffs2, ubifs — Flash Speicher mit Journaling
  - cramfs, squashfs — Komprimierte Filesysteme
- Filesysteme für Netzwerk
  - nfs — UNIX Standard
  - cifs, smb — Windows Freigaben

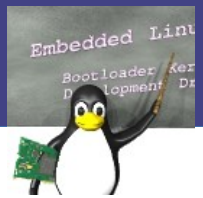


File Hierarchie Standard



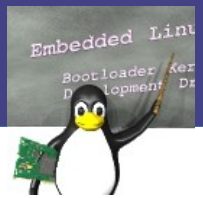
- bin: Wichtige Applikationen und Tools, z.B. *bash*
- sbin: System-Applikationen und Tools, z.B. *ifconfig*
- dev: Device-Nodes, Zugriff auf Geräte und Treiber
- etc: Konfigurations-Dateien
- lib: Systembibliotheken und Kernelmodule
- var: Variable Daten, z.B. Log-Files
- tmp: Temporäre Daten
- usr: User Applikationen, Tools und Bibliotheken
- mnt: Mount-Point

- proc: Informationen über laufende Prozesse
- sys: Informationen über das laufende System
- opt: Optionale Komponenten
- boot: Kernel und Bootloader
- home: Verzeichnisse für die Nutzer



Objekt-Attribute

- Es gibt verschiedene Objekte im virtuellen Filesystem
  - Dateien
  - Verzeichnisse
  - Symbolische Links
  - UNIX Domain Sockets
  - Named Pipes
  - Device-Nodes
- Jedes Objekt besitzt Attribute
  - Name
  - Größe
  - Erstellungs- und Modifikations-Datum
  - Besitzer und Gruppe
  - Zugriffsrechte

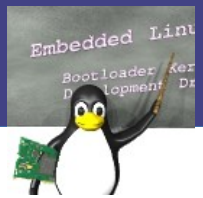


Objekt-Attribute

```
ls -l /tmp/testdatei

crw-r--r-- 1 boerner users 536870912 2011-05-18 22:26 /tmp/testdatei
```

- Objekt Typ
  - \- Datei
  - d Directory
  - l Symbolischer Link
  - s Socket
  - p Named Pipe
  - b Block Device
  - c Character Device
- Zugriffsrechte (Besitzer, Gruppe, Andere)
  - r Lesezugriff
  - w Schreibzugriff
  - x Ausführbar bei Datei
Betretbar bei Verzeichnis
- Anzahl der Hard-Links
- Besitzer
- Gruppe
- Größe in Blöcken
ls -k gibt Kilobytes aus
- Veränderungsdatum
- Objektname



Zugriff auf Treiber und Geräte

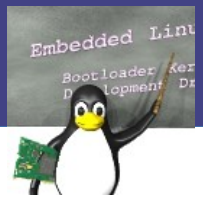
- UNIX Philosophie

  „Alles ist ein File“

  - Auch die Geräte sind eine Datei im virtuellen FileSystem
  - Device Nodes unter /dev
  - Block Geräte: Wahlfreier Zugriff auf Datenblöcke (Festplatten)
  - Character Geräte: Sequentieller Zugriff (Serielle Schnittstelle)
  - Verbindung zum Kernel über Major/Minor Nummern

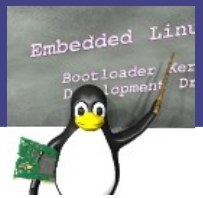
```
crw-rw---- 1 root dialout 4, 64 2011-05-23 18:09 /dev/ttyS0
crw-rw---- 1 root dialout 4, 65 2011-05-23 18:09 /dev/ttyS1
crw-rw---- 1 root dialout 4, 66 2011-05-23 18:09 /dev/ttyS2
crw-rw---- 1 root dialout 4, 67 2011-05-23 18:09 /dev/ttyS3
```

  - Ausnahme: Netzwerkgeräte tauchen nicht auf



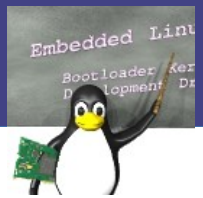
Zugriff auf Geräte und Treiber

- Zugriff auf ein Gerät erfolgt mit normalen File Funktionen
  - open(): öffnet den Zugriff
  - read(): liest Daten vom Gerät
  - write(): schreibt Daten zum Gerät
  - ioctl(): dient zur Steuerung des Geräts
    (z.B. Baudrate setzen)
- Die Zugriffsrechte sind wie bei Dateien geregelt
  - !! Durch Anlegen eines eigenen Device-Nodes können die gesetzten Zugriffsrechte umgangen werden.



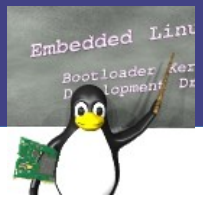
Device Nodes erzeugen

- Device Nodes werden mit mknod erstellt
  - `mknod <name> <type> <major> <minor>`
    - name: Frei wählbarer Name
    - type: b = Block, c = Character
    - major: Major Nummer
      - Treiber Dokumentation
      - linux/Documentation/devices.txt
      - /proc/devices
    - minor: Minor Nummer
      - Treiber Dokumentation
      - linux/Documentation/devices.txt
  - `z.B. mknod /dev/ttyS5 c 4 4`
- `Udev erzeugt automatisch Device Nodes für erkannte angeschlossene Geräte`



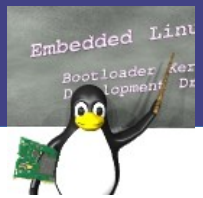
Spezielle Geräte

- Es gibt Device-Nodes für spezielle Einsätze
  - /dev/null
    - Ist der System-Schredder
    - Wird z.B. verwendet, um Ausgaben bei Bauvorgang nicht auf dem Bildschirm erscheinen zu lassen und z.B. nur Fehlermeldungen zu sehen:
      `make > /dev/null`
  - /dev/zero
    - Liefert binäre Nullen solange gelesen wird
    - Wird z.B. als erster Schritt beim Erzeugen eines virtuellen Images genutzt:
      `dd if=/dev/zero of=/tmp/virt.img count=256M`



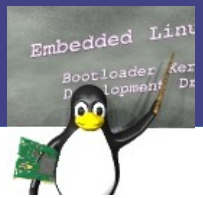
Spezielle Geräte

- Es gibt Device-Nodes für spezielle Einsätze
  - /dev/random
    - Liefert nicht deterministische binäre Zufallszahlen zurück
    - Benutzt zur Generierung der Zufallszahlen zufällige Events, wie z.B. Mausbewegung und Tastatur
    - Generiert bei Events einen Vorrat an Zufallszahlen
    - Lesen von diesem Device Node wird gestoppt wenn der Vorrat erschöpft ist
  - /dev/urandom
    - Liefert deterministische binäre Zufallszahlen zurück
    - Es kann immer gelesen werden



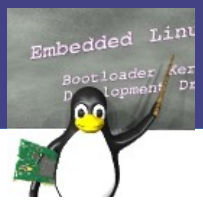
procfs

- Prozess Filesystem procfs
  - Enthält Informationen laufender Prozesse
  - Wird vom Kernel und den Treibern zur Laufzeit erzeugt
  - Keine statischen Einträge
  - Beim Lesen eines Eintrags wird Funktion im Kernel/Treiber aufgerufen
  - Wird standardmäßig unter /proc gemountet
    `mount -t procfs null /proc`
  - Bietet Informationen zu
    - Allen laufenden Prozessen anhand der Prozess ID
    - Netzwerk (/proc/net)
    - Speicherverwaltung
    - …

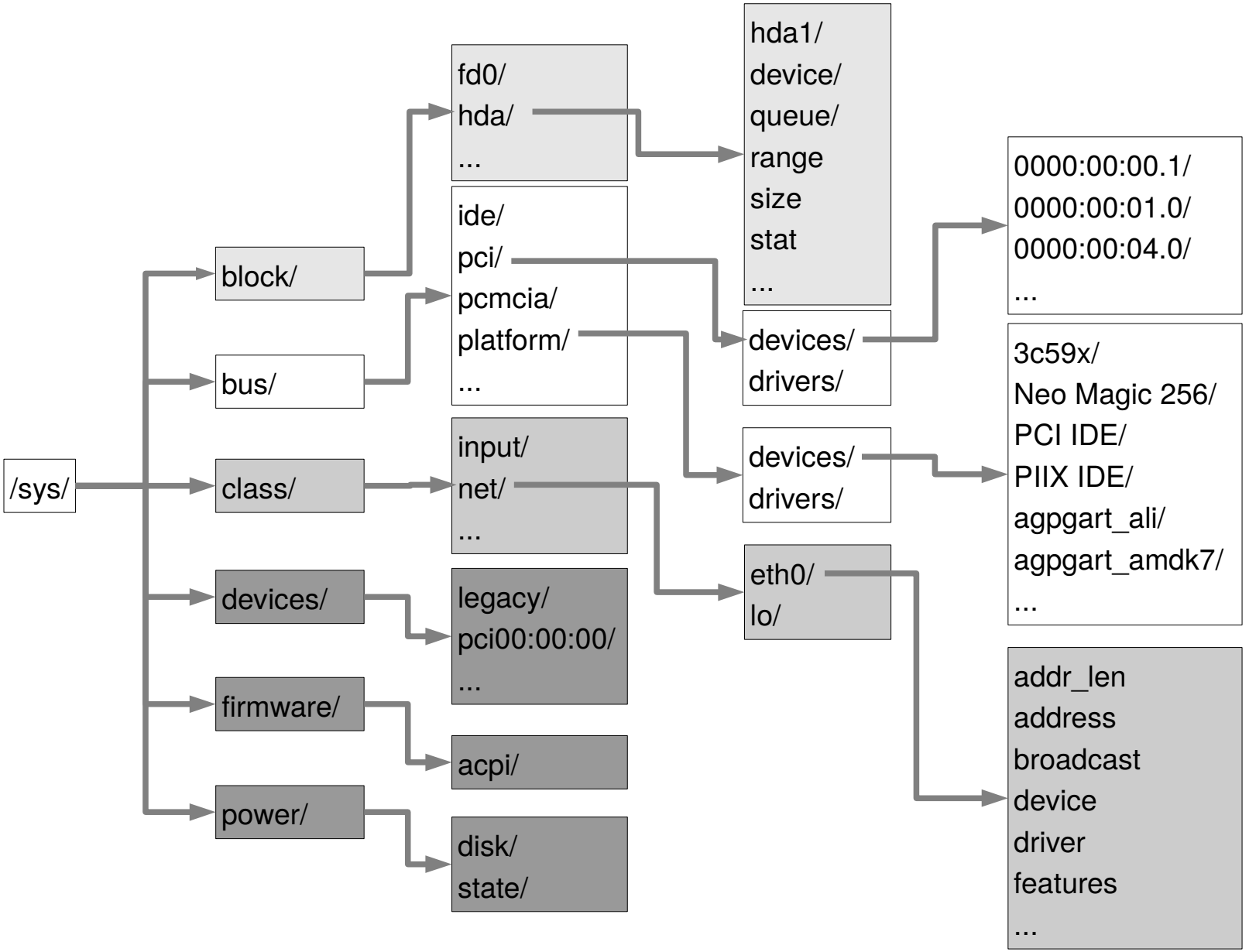


sysfs

- System Filesystem
  - Bietet Informationen und Zugriff auf das komplette laufende System
  - Kann zur Konfiguration des Systems zur Laufzeit genutzt werden
  - Wird standardmäßig unter /sys gemountet
    `mount -t sysfs none /sys`
  - Bietet Informationen zu
    - Angeschlossene Geräte
    - Laufende Treiber
    - CPUs
    - Speicherbereiche
    - GPIOs
    - Debugging
    - ...


Embedded Linu

sysfs


/sys/
block/
bus/
class/
devices/
firmware/
power/
fd0/
hda/
...
ide/
pci/
pcmcia/
platform/
...
input/
net/
...
legacy/
pci00:00:00/
...
acpi/
disk/
state/
hda1/
device/
queue/
range
size
stat
...
devices/
drivers/
devices/
drivers/
eth0/
lo/
0000:00:00.1/
0000:00:01.0/
0000:00:04.0/
...
3c59x/
Neo Magic 256/
PCI IDE/
PIIX IDE/
agpgart_ali/
agpgart_amdk7/
...
addr_len
address
broadcast
device
driver
features
...