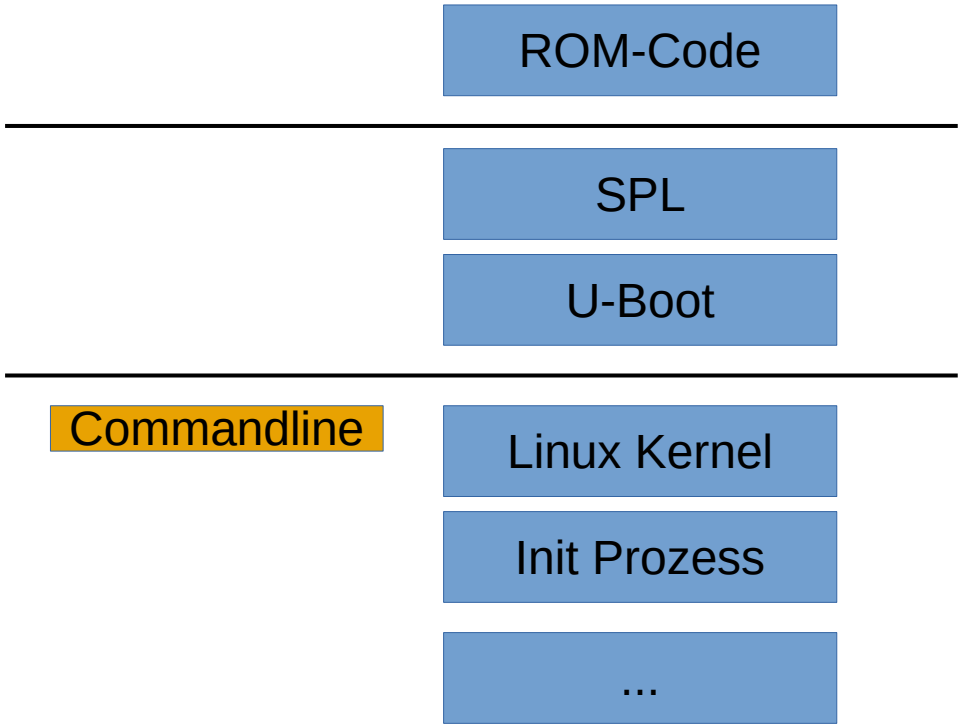
Linux RootFS

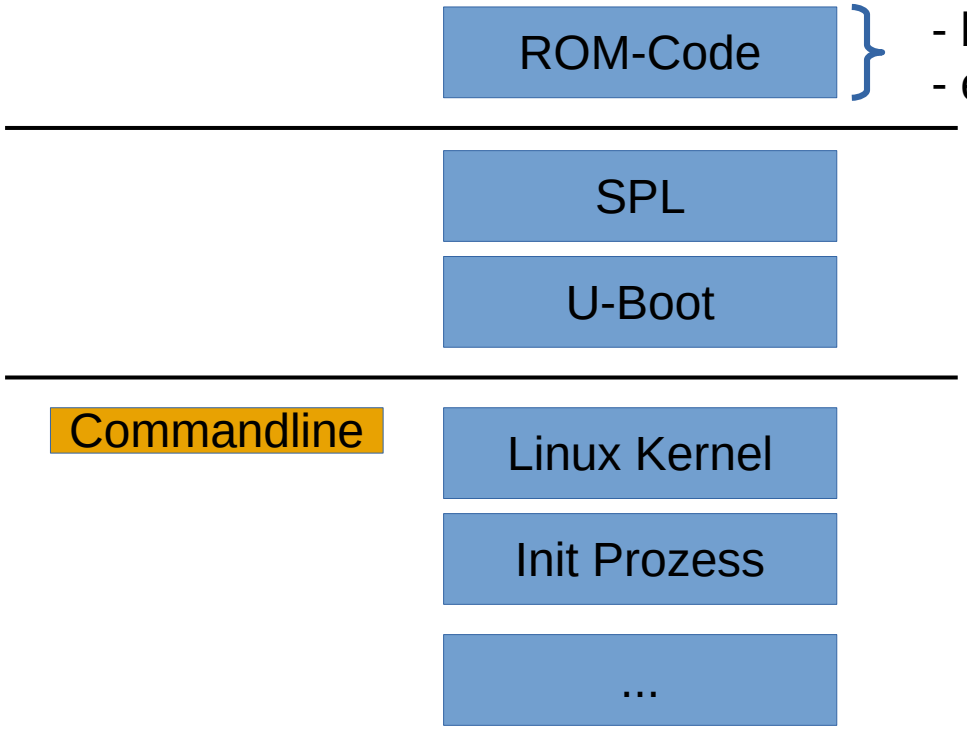
EIM/INFM

Frank Erdrich
frank.erdrich@emtrion.de

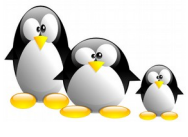
Aktueller Stand



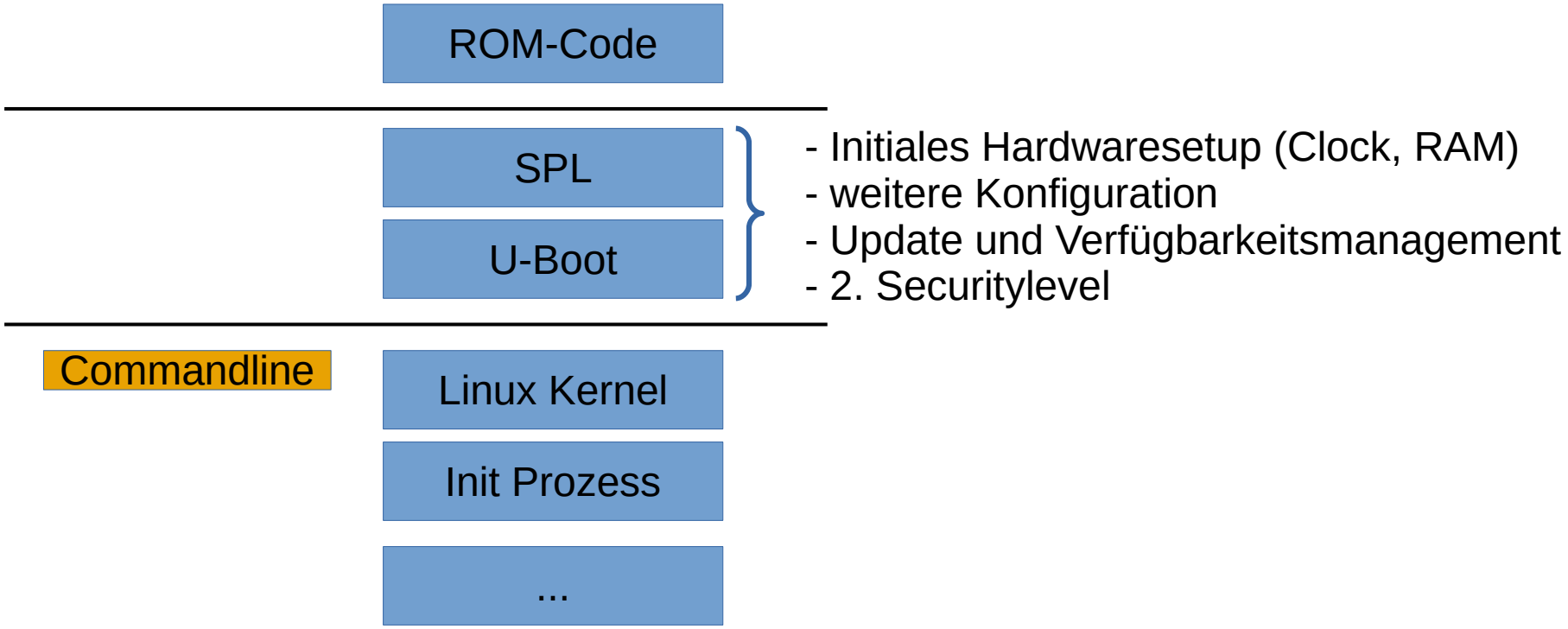
Aktueller Stand



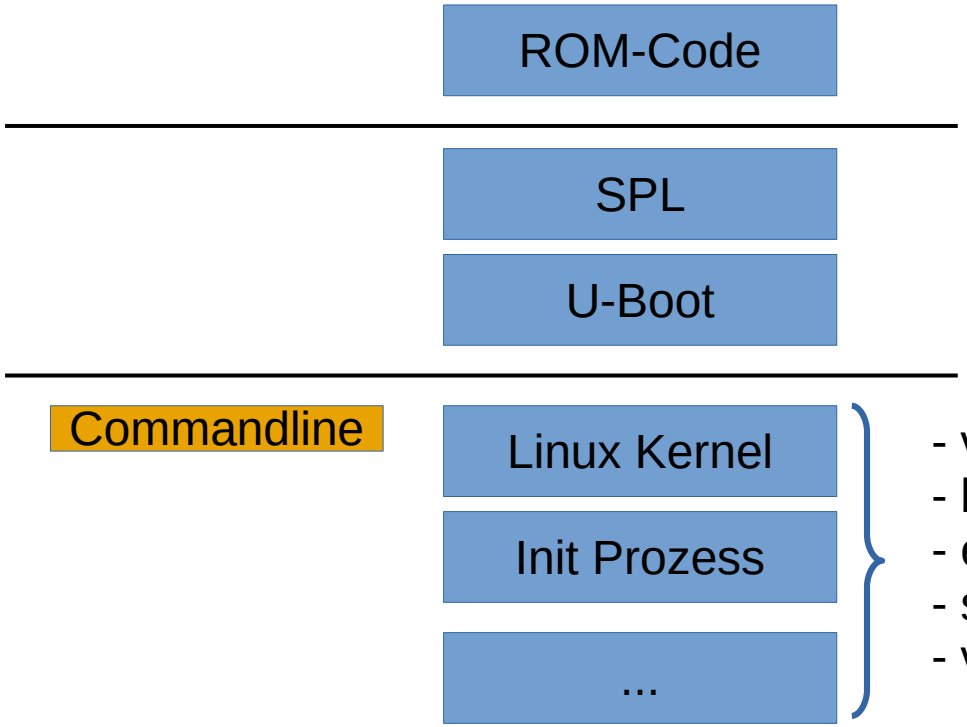
- Herstellerspezifisch
- nicht änderbar im SoC hinterlegt
- bootet nur von definierten Medien
- erster Level für Security



Aktueller Stand



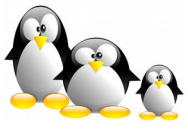
Aktueller Stand



- vollständiges Hardwaresetup
- benötigt DeviceTree für Hardwareerkennung
- einbinden von Dateisystemen
- starten von Autostartprogrammen
- verschiedene Runlevel

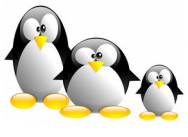
Kernel Commandline

- Schnittstelle zwischen Bootloader und Linux, Parameterübergabe und Bootargumente
  - Console: console=ttyS0,115200
  - RootFS: root=/dev/mmcblk0 rw rootwait
  - Frühe Bootmsg: earlyprintk
  - Anderes init: init=/home/init
  - Und vieles mehr, siehe:
    - https://www.kernel.org/doc/html/v4.10/admin-guide/kernel-parameters.html



Bootmeldungen

- Gibt Auskunft über gefundene Hardware
  - Zeigt teilweise Konfiguration von Treibern
  - Wichtig für Debugging
  - Mit Befehl „dmsg“ später abrufbar
    - Kann auf Loglevel filtern
  - Kann mit geeignetem Debugger auch ohne serielle Schnittstelle gelesen werden

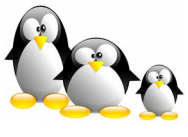


Loglevel Bootlog

Level	Name	Bedeutung
0	KERN_EMERG	system is unusable
1	KERN_ALERT	action must be taken immediately
2	KERN_KRIT	critical conditions
3	KERN_ERR	error conditions
4	KERN_WARNING	warning conditions
5	KERN_NOTICE	normal but significant condition
6	KERN_INFO	informational
7	KERN_DEBUG	debug-level messages

Über Kernel Commandline mit „loglevel=level“ einstellbar.

level < loglevel ist immer mit aktiv



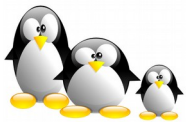
PROCFS

- Process Filesystem
- Virtuelles Dateisystem
- Enthält:
  - Generelle Systeminfos
  - Informationen zu jedem Prozess

Details siehe: https://www.kernel.org/doc/Documentation/filesystems/proc.txt

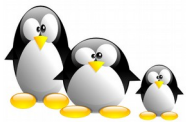
PROCFS

- Enthält eine Menge an Systeminformationen
  - cpuinfo
  - Speicher
  - Kernel commandline
  - Eingehängte Datenträger
  - Memory Technology Devices Übersicht (MTD)
  - Info zu jedem laufenden Prozess



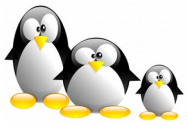
SYSFS

- System Filesystem
- Virtuelles Dateisystem mit Systeminfos
- Sollte das PROCFS entschlacken, welches nur Prozessinformationen darstellen soll
- Exportiert Informationen aus dem Kernelspace (EL1) in den Userspace (EL0)
- Dient auch zur (rudimentären) Konfiguration von Treibern (siehe etwa LED oder GPIO)



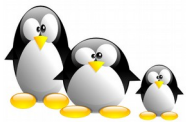
SYSFS

- Einige wichtige Verzeichnisse:
  - /sys/bus: diverse Bussysteme, etwa I²C
  - /sys/class: Devices nach Klassen, etwa LED, GPIO oder Backlight
  - /sys/firmware: Devicetree aufgeschlüsselt und binär
- Hilfreich, um sich auf unbekannten Systemen auch ohne Quellcode eine Übersicht verschaffen zu können



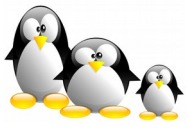
SYSFS

- Weitere Verzeichnisse:
  - /sys/kernel/debug
  - /sys/module
  - /sys/power
- Details zur Erstellung eigener Einträge im Kapitel Treiberentwicklung



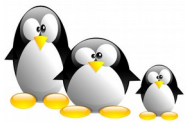
CONFIGFS

- Weiteres virtuelles Dateisystem
- Kann Kernelobjekte anlegen und konfigurieren
- Erstellen eines Verzeichnisses generiert ein Kernelobjekt, welches anschließend konfiguriert werden kann



CONFIGFS

- Als Ergänzung zum SYSFS zu sehen
- SYSFS repräsentiert den aktuellen Status von Treibern im Kernel
- CONFIGFS kann Kernelobjekte über den Userspace anlegen
- Steht in Konkurrenz zu den IOCTLs
- Beispiel:
  - USB-Gadget Treiber



RootFS

- Root Filesystem – grundlegendes Dateisystem
- Mindestens ein Init-Programm muss vorhanden sein
- Enthält diverse notwendige Tools
  - Shell, z.B. bash oder sh
  - ls, cd, mkdir, rm, ls, ...
  - mount

RootFS

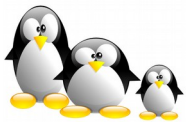
- Wie erstellt man ein RootFS?
- Und was muss mindestens enthalten sein?
- Genügt nicht eine fertige Distribution?
- Zertifizierung?

RootFS Build

- Linux from Scratch
- Eigene Buildscripte
- Buildroot
- Yocto Project - Open Embedded
- Debian based: Elbe
- ...

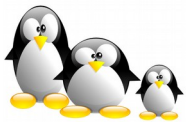
Linux from Scratch (LFS)

- Anleitung zum Bau eines Linux-Systems
- Zusätzlich
  - Diverse Packages
  - Patches
- Ideal zum lernen, wie ein Linux-System aufgebaut ist
- Kann Grundlage eines Custom-RootFS sein



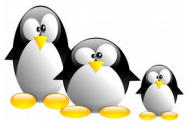
Linux from Scratch (LFS)

- Vorteile:
  - Kann sehr genau angepasst werden
  - Starker Lerneffekt, führt zu gutem Verständnis bei anderen Buildsystemen
- Nachteile:
  - Schwerer Einstieg
  - Zeitaufwändig
  - Fehleranfällig

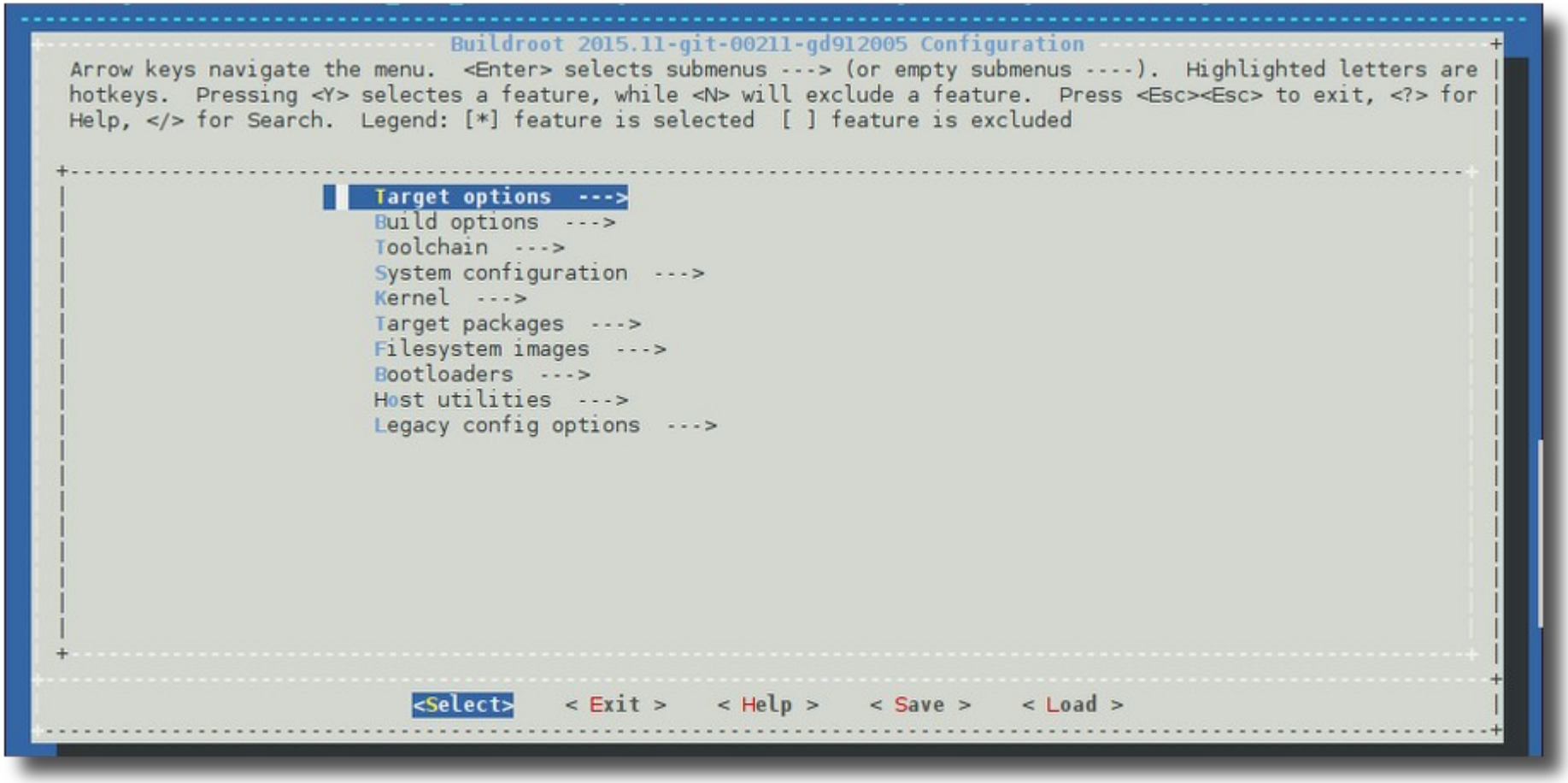


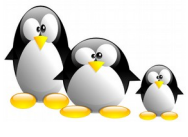
Buildroot

- www.buildroot.org
- https://github.com/buildroot/buildroot
- Open Source Projekt
- Makefile-basiert (siehe auch Linux Kernel)
- Viele vorgefertigte Pakete



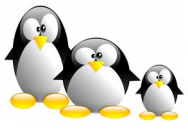
Buildroot


Buildroot 2015.11-git-00211-gd912005 Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected [ ] feature is excluded
Target options --->
Build options --->
Toolchain --->
System configuration --->
Kernel --->
Target packages --->
Filesystem images --->
Bootloaders --->
Host utilities --->
Legacy config options --->
<Select>
< Exit >
< Help >
< Save >
< Load >



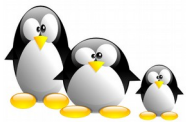
Buildroot

- Vorteile
  - Schneller Einstieg
  - Kleine RootFS (siehe OpenWRT)
  - Viele vorgefertigte Pakete
- Nachteile
  - Grundsystem nur schwer anpassbar
  - Handarbeit notwendig
  - Lange Buildzeiten



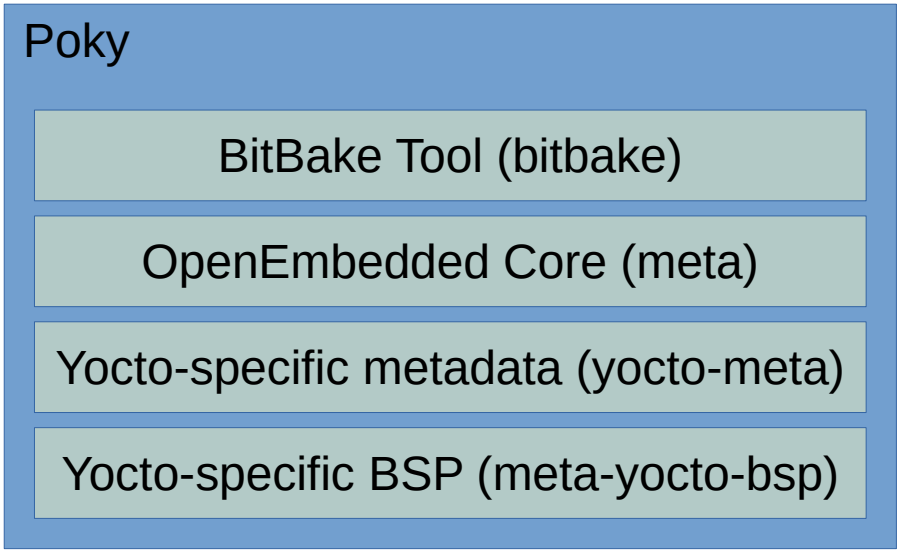
Yocto Project

- https://www.yoctoproject.org
- Open Source
- Besteht aus Layern und Rezepten
- Open Embedded als Basis (zusammengefasst als Poky)
  - OE-Core: Basislayer
  - BitBake: Buildsystem

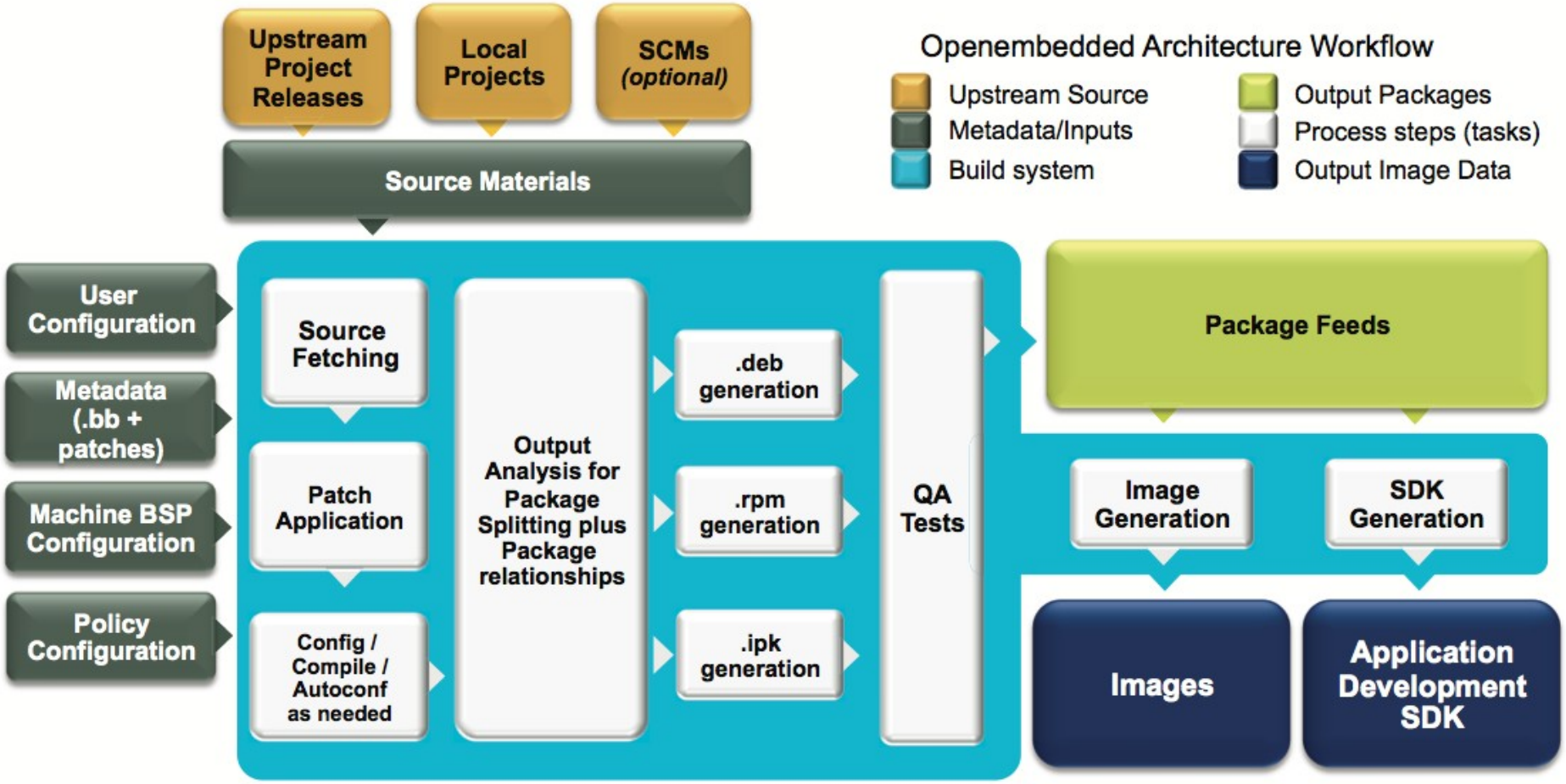


Yocto Project

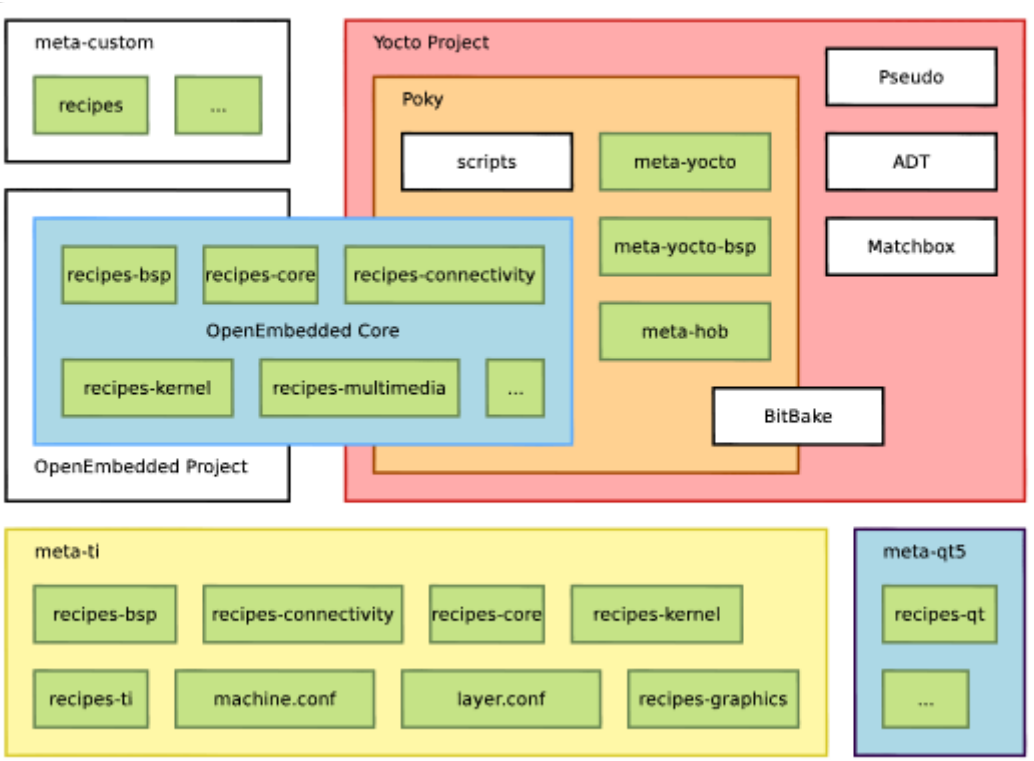
- Rezepte beschreiben, wie etwas zu bauen ist
  - Dabei kann jeder Schritt überschrieben oder ergänzt werden (fetch, configure, build, install, ...)
- Layer fassen Rezepte zu sinnvollen Einheiten zusammen



Yocto Project



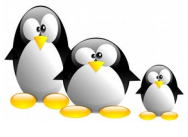
Yocto Project



Quelle: Bootlin (www.bootlin.org)

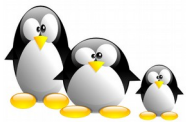
Yocto Project

- Vorteile
  - Sehr gute Auflösung der Abhängigkeiten
  - Im Prinzip zu 100 % anpassbar
  - Große Community, zu fast jedem SoC/Board existiert ein Layer
  - Große Vielzahl an Rezepten verfügbar
  - Alles lässt sich überschreiben und ändern



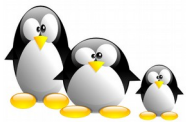
Yocto Project

- Nachteile
  - Sehr komplex
  - Build dauert lange
  - Anfangs schwer zu verstehen
    - Viele vordefinierte Variablen und Pfade
    - Zusammenhang zwischen den Rezepten, Layern und Konfigurationen
  - Alles lässt sich überschreiben und ändern



Elbe

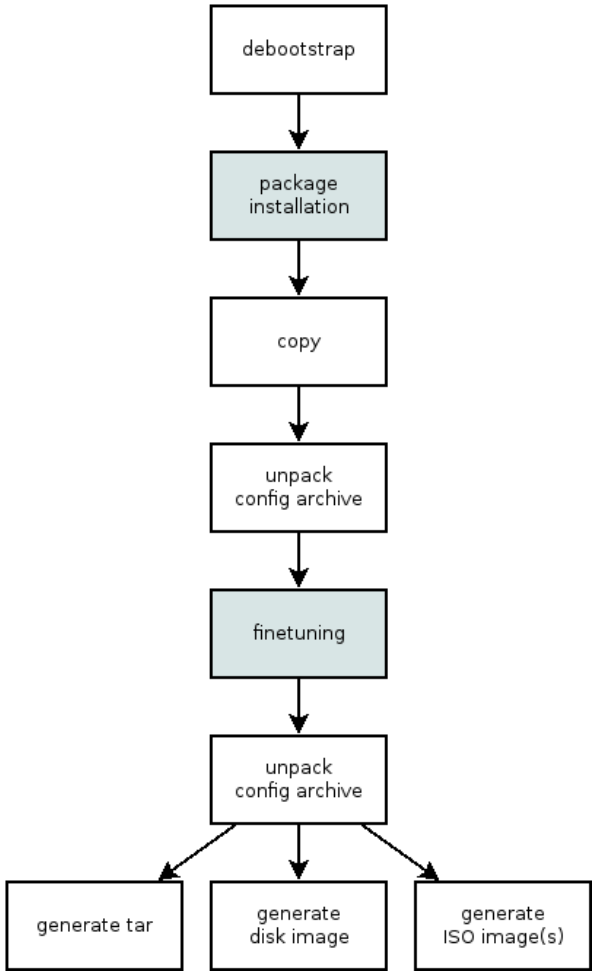
- https://elbe-rfs.org
- Open Source
- Entwickelt von Linutronix
- Baut Debian-basierte RootFS
  - Verwendet offizielle Debian Paketserver
  - Nutzt Debootstrap in einer VM
  - XML als Beschreibungssprache

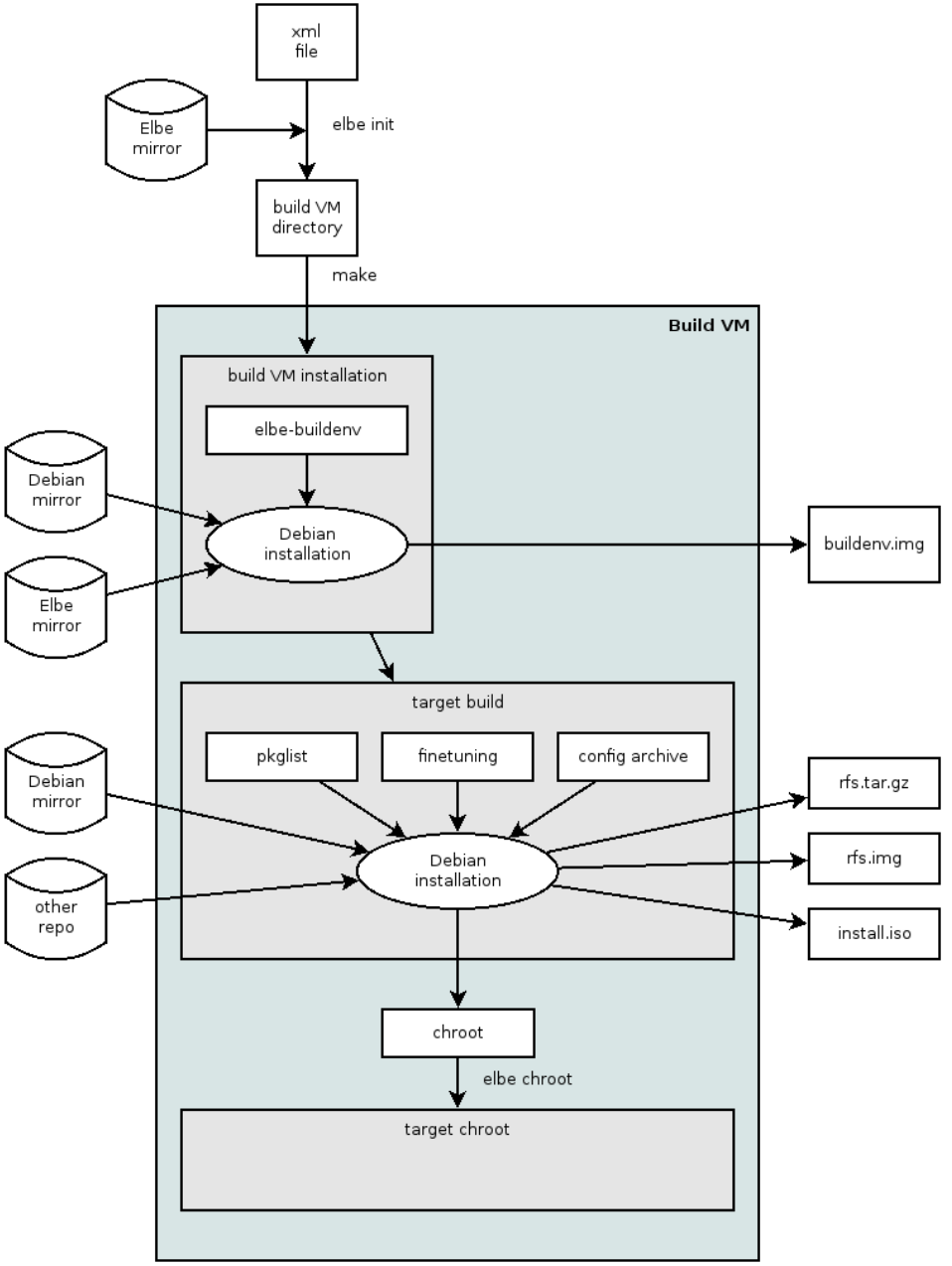


Elbe - Debian

- Warum Debian?
  - Long Term Support (um die 4 Jahre)
  - Regelmäßige Securityupdates
  - Auch für ARM viele Pakete verfügbar
  - Große Community

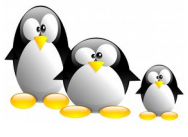
Elbe


debootstrap
package installation
copy
unpack config archive
finetuning
unpack config archive
generate tar
generate disk image
generate ISO image(s)


xml file
Elbe mirror
elbe init
build VM directory
make
Build VM
build VM installation
elbe-buildenv
Debian mirror
Elbe mirror
Debian installation
buildenv.img
target build
pkglist
finetuning
config archive
Debian mirror
other repo
Debian installation
rfs.tar.gz
rfs.img
install.iso
chroot
elbe chroot
target chroot

Elbe

- Vorteile:
  - Viele fertige Pakete vorhanden
  - Offizielle Pakete sind getestet (Debian QS)
  - Regelmäßige Securitypatches
  - Eigene Pakete können einfach erstellt werden
  - Eingebautes Update-System
  - Schneller Build



Elbe

- Nachteile:
  - Starres Buildsystem
  - Grundsystem nicht einfach anpassbar
  - Ohne Virtualisierung kein Elbe

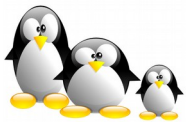
Elbe

- Beispiel zeigen

???

- Recap: Kernel und Userspace sind getrennt
  - EL0: Userspace
  - EL1: Kernel

- Frage: Wie kann ein Userspaceprogramm mit dem Kernel kommunizieren, beispielsweise eine Datei öffnen? Oder wie wird mit Treibern kommuniziert, die nicht im SYSFS eingetragen sind?



System Call

- Warum System Calls?

System Call

- Kernel verwaltet die Ressourcen
  - Speicher (RAM, Festspeicher)
  - Peripherie
  - Threads und Prozesse
- Zugriffsverwaltung

System Call

- Syscalls Tabelle in Kernel 4.1:
  - https://github.com/torvalds/linux/blob/v4.1/arch/x86/syscalls/syscall_64.tbl

IOCTL

- Steht für Input Output Control
-
- Syscalls Tabelle in Kernel 4.1:
  - https://github.com/torvalds/linux/blob/v4.1/arch/x86/syscalls/syscall_64.tbl