Eingebettete Betriebssysteme

Bachelorstudiengang
Technische Informatik
Hochschule Pforzheim

Dipl.-Ing.(FH) Marc Jüttner

Systemstart

- Systemstart auf dem PC: BIOS
  - Initialisierung der Hardware
  - Laden relevanter Konfiguration
  - Starten des Bootloaders
  - Laden und starten des Betriebssystems
- Wie läuft das bei eingebetteten Systemen?

Systemstart: Varianten

- Häufig zwei Varianten implementiert
- Start zur Entwicklung
  - Auswahl verschiedener Betriebssystem-Abbilder
  - Modifikation von Startparametern
  - U-boot, BiosLT
- Start des Produktivsystems
  - Keine unnötige Zeitverschwendung

Systemstart: CPU-Sicht

- CPU startet, nachdem das Reset-Signal inaktiv wird
  - CPU startet an Adresse 0x0
  - Reset-Wert des Befehlsregisters
- ausführbarer Code erwartet
  - SoCs verfügen über verschiedene Speicherarten
  - Welcher liegt an Adresse 0x0?

Systemstart: SoC-Sicht

- Mehrere Startmedien
  - Flash: NOR, NAND
  - SD-Karte
  - Serielle Protokolle SPI, I2C, UART
  - USB
  - Ethernet
  - ...
- Aber: Oft ist ein Protokoll erforderlich!

Systemstart: XIP

- Start von linear adressierbaren Medien
  - ROM, RAM, Flash (teilweise)
  - Lineare Adressierung ist nicht bei jeder Flash-Variante möglich
- XIP: eXecute In Place
  - Flashmedium ist linear adressierbar
  - NOR-Flash
- Geht für Code und Read-Only-Daten

Systemstart: XIP

- Was tun mit variablen Daten?
- Arbeitsspeicher wird benötigt für BSS- und DATA-Segment
  - Was tun bei nicht-initialisierten externen Speicherbausteinen?
- Spezielle Übersetzung erforderlich

Mehrstufiger Start

- Verwendung von ROM-Bootcode
  - *Primary Bootloader*
- Laden von Initialisierungsfunktionen
  - *Secondary Bootloader*
- Laden des Betriebssystems
  - Übergabe von Parametern!
- Start der Anwendung
  - Userspace bei HLOS

Primary Bootloader

- Fest implementiert, nicht änderbar
- Stark angepasst auf CPU und SoC
- Nicht angepasst an das Systemdesign
  - Peripherieparameter
  - Externe Hardware
- Implementiert Zugriff auf SD-Karte, Flash und andere Peripherie
  - SPI, I2C, UART, USB, ...

Primary Bootloader: Start

- Nach einem Reset ist zunächst nur die SoC-Peripherie verfügbar
- Nur interner Arbeitsspeicher verwendbar
  - Begrenzung der Codegröße für Secondary Bootloader
- Controller für externen Speicher ist unkonfiguriert
  - Taktrate, Latenzen, sonstige Parameter

Wahl der Bootmedien

- Bootmedium kann bei den meisten SoCs vorgegeben werden
  - Power-on-latched pins (BOOTMODE o.ä.)
  - „Schick“ für Entwicklung
  - DIP-Switches
- Oft auch Suchreihenfolge möglich
  - Flash → SD-Karte → UART → ...

Bootmodes

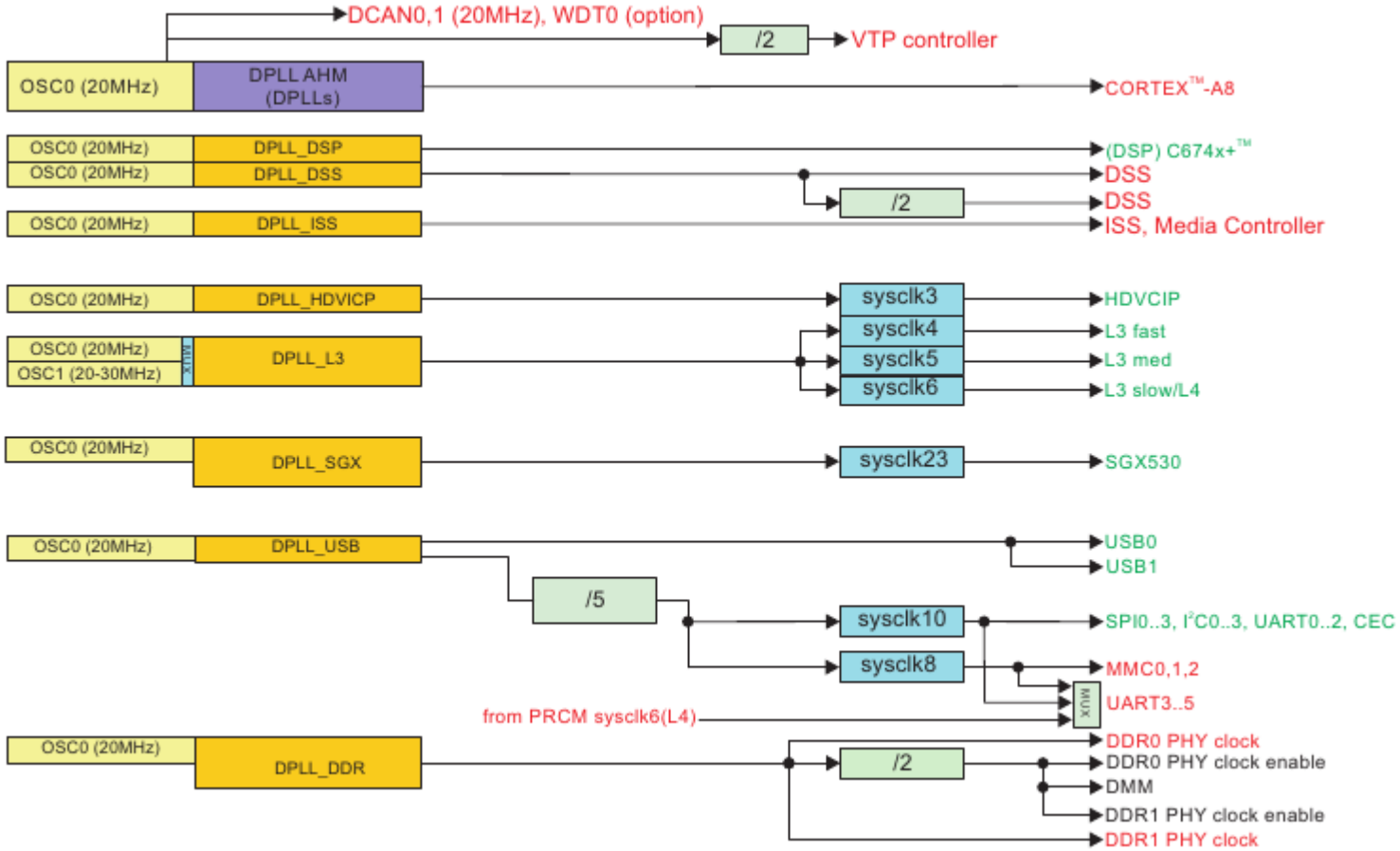
Boot Modes				BTMODE[4:0]
1st	2nd	3rd	4th	
Reserved	Reserved	Reserved	Reserved	00000
UART	XIP w/ WAIT (MUX0)	MMC	SPI	00001
UART	SPI	NAND	NANDI2C	00010
UART	SPI	XIP (MUX0)	MMC	00011
EMAC	SPI	NAND	NANDI2C	00100
Reserved	Reserved	Reserved	Reserved	00101
Reserved	Reserved	Reserved	Reserved	00110
EMAC	MMC	SPI	XIP (MUX 1)	00111
NAND	NANDI2C	SPI	UART	10010
NAND	NANDI2C	MMC	UART	10011
NAND	NAND12C	SPI	EMAC	10100
NANDI2C	MMC	EMAC	UART	10101
SPI	MMC	UART	EMAC	10110
MMC	SPI	UART	EMAC	10111
SPI	MMC	PCIE_32	Reserved	11000
SPI	MMC	PCIe_64	Reserved	11001
XIP (MUX0)	UART	SPI	MMC	11010
XIP w/ WAIT (MUX0)	UART	SPI	MMC	11011

Quelle: Texas Instruments DM8148 TRM Tabelle 4.8

Initialisierung: System

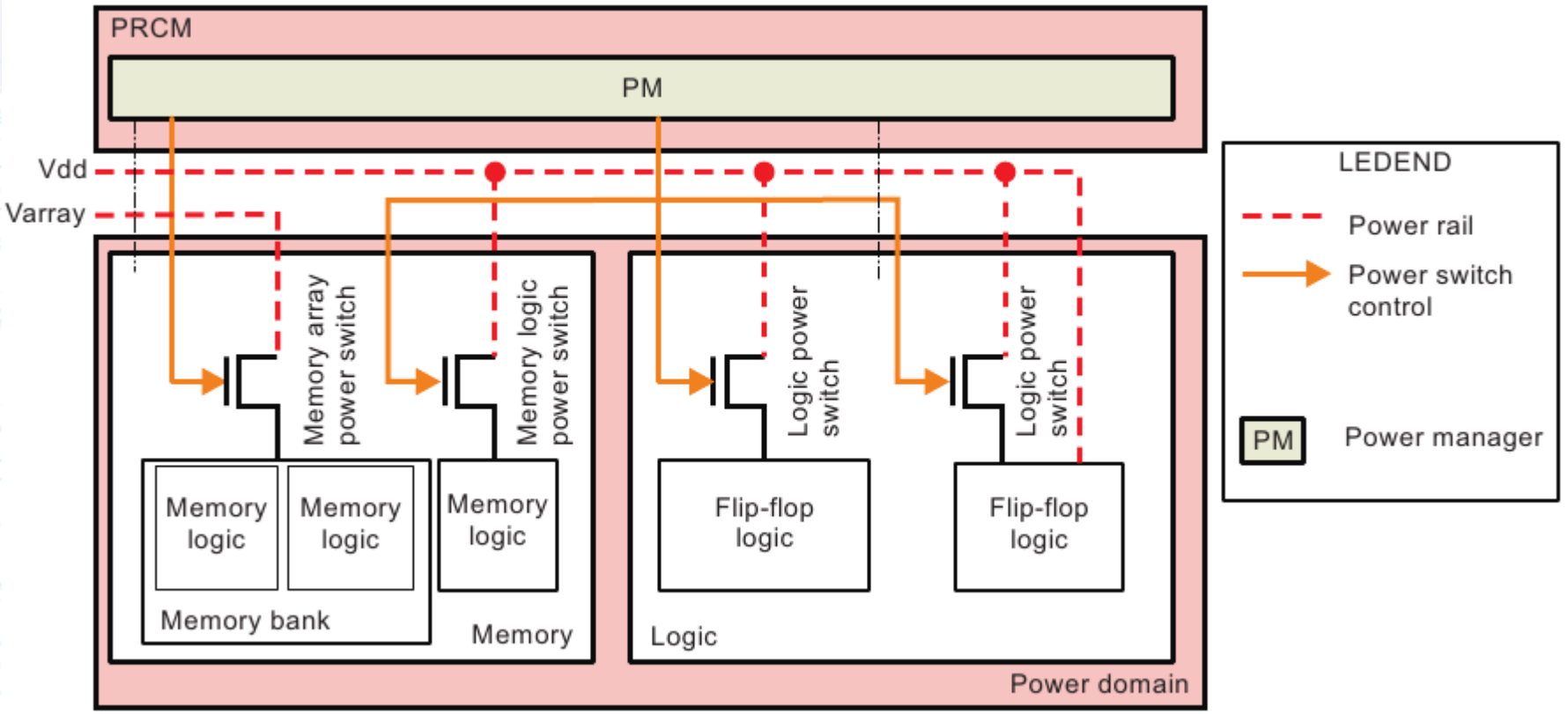
- Aktivierung erforderlich
  - *Power/clock domains*
  - Beide oft *default off*
  - PLL-Instanzen
- Initialisierung nur der Peripherie, die tatsächlich benötigt wird
  - Viele Betriebssysteme erwarten nichtinitialisierte Hardware!

Clock domains



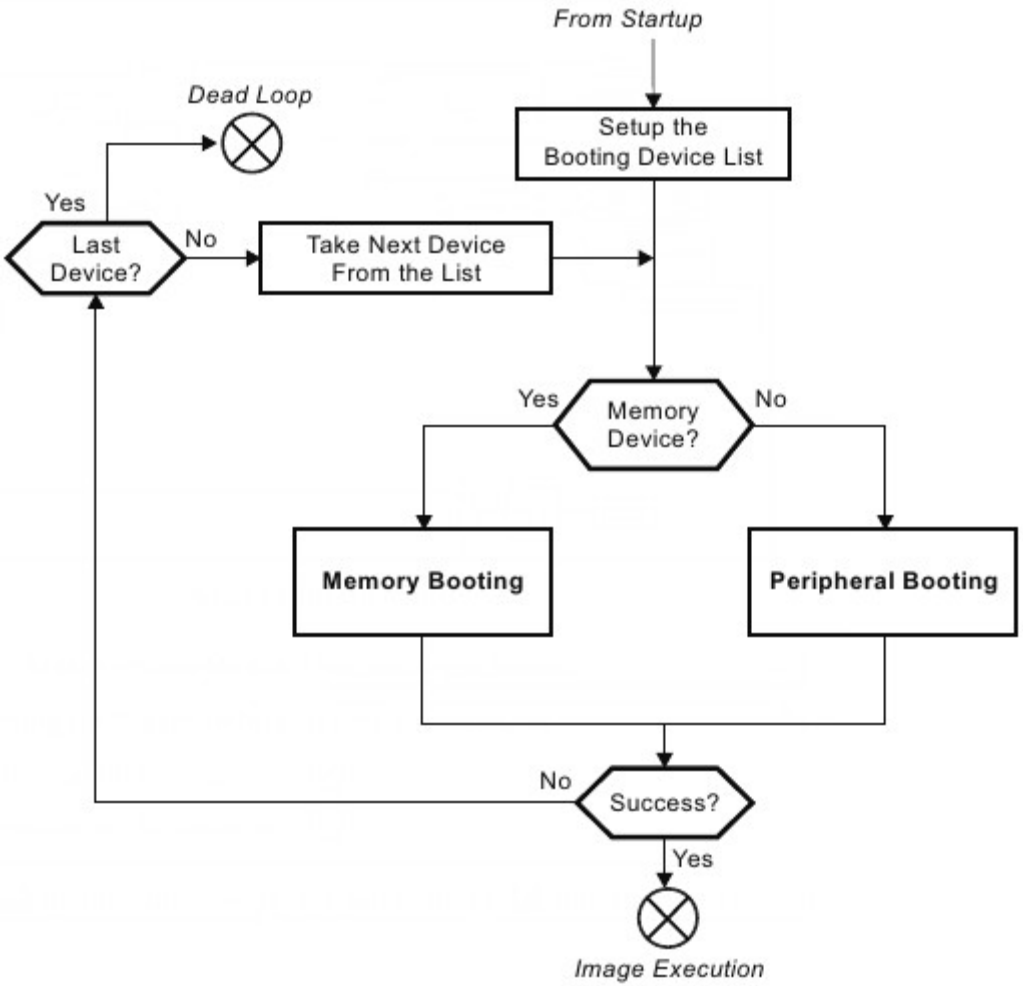
Quelle: Texas Instruments DM8148 TRM Fig. 2.6

Power domains



Quelle: Texas Instruments DM8148 TRM Fig. 2.4

ROM-Boot bei TI DM8148



Quelle: Texas Instruments DM8148 TRM Fig. 4.2

Imageformat für 2nd stage

Field	non-XIP device (offset)	XIP device (offset)	Size (bytes)	Description
Size	0000h	-	4	Size of the image
Destination	0004h	-	4	Address where to store the image / code entry point
Image	0008h	0000h	x	Executable code

Quelle: Texas Instruments DM8148 TRM Tabelle 4.37

- Destination gibt Ziel im Hauptspeicher an
- Muss beim Erstellen der Software bekannt sein!

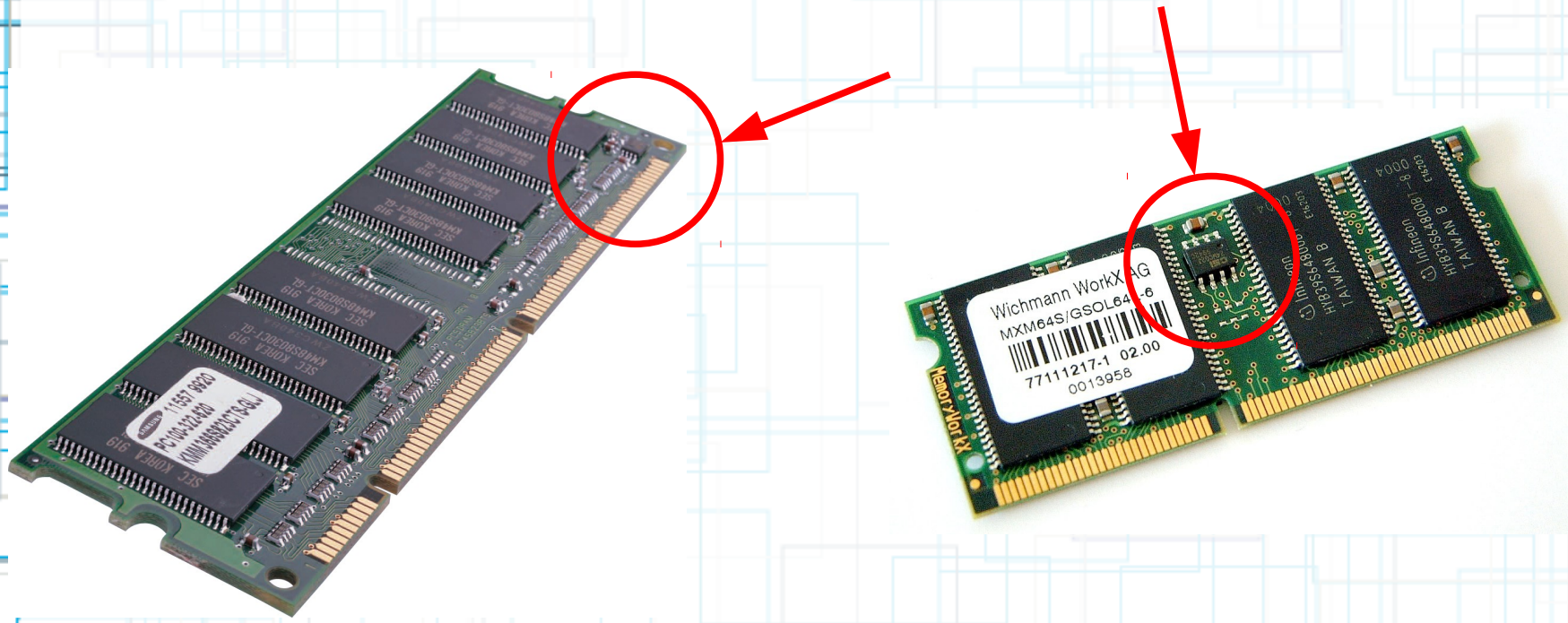
Secondary Bootloader

- Angepasst an das Systemdesign
  - Peripherieparameter
  - Externe Hardware
- Initialisiert das System anhand ihm bekannter Parameter und Peripherie
- Lädt das endgültige Betriebssystem

Initialisierung: Speicher

- Initialisierung des Controllers für externen Speicher
  - Taktrate
  - Latenzzeiten
  - Zahl der chips
  - Interne Organisation der Chips
- Hartkodierte Parameter!

DDR-Parameter beim PC?



Quelle: Blogspot/Wikimedia Commons

Laden des Betriebssystems

- Betriebssystemabbild ist ein binäres Programm
  - Auf bestimmte Speicherstelle gelinkt
  - Ausführung durch Laden des Befehlszeigers
- Meist aus Flashspeicher zu laden
- Erfordert u.U. Startparameter

Security-Anforderungen

- Signed code
  - Mit Prozessorunterstützung wird nur signierter Code geladen
  - Erfordert Schlüsselprogrammierung
- Sicherheitskonzept: Firewalls
  - Peripheriezugriff wird von sicherem Code konfiguriert
  - Keine Änderung zur Laufzeit

Linux-Startparameter

- Linux erwartet ein Feld von Parametern
  - struct tag
- Startmarker
  - ATAG_CORE
- Liste
- Endemarker
  - ATAG_NONE

```
struct tag {
	struct tag_header hdr;
	union {
		struct tag_core		core;
		struct tag_mem32	mem;
		struct tag_videotext	videotext;
		struct tag_ramdisk	ramdisk;
		struct tag_initrd	initrd;
		struct tag_serialnr	serialnr;
		struct tag_revision	revision;
		struct tag_videolfb	videolfb;
		struct tag_cmdline	cmdline;

		/*
		 * Acorn specific
		 */
		struct tag_acorn	acorn;

		/*
		 * DC21285 specific
		 */
		struct tag_memclk	memclk;
	} u;
};
```

Quelle: linux/arch/arm/include/uapi/asm/setup.h (Linux 3.8.6)

ATAG-Parameter

- Verschiedene Parameter möglich
  - Speicherkonfiguration
  - Grafikausgabe (Video)
  - Ramdisk
  - …

```
struct tag_mem32 {
        __u32   size;
        __u32   start;  /* physical start address */
};
```

```
struct tag_cmdline {
        char    cmdline[1];     /* this is the minimum size */
};
```

```
/* The list ends with an ATAG_NONE node. */
#define ATAG_NONE       0x00000000
/* The list must start with an ATAG_CORE node */
#define ATAG_CORE       0x54410001
/* it is allowed to have multiple ATAG_MEM nodes */
#define ATAG_MEM        0x54410002
/* VGA text type displays */
#define ATAG_VIDEOTEXT  0x54410003
/* describes how the ramdisk will be used in kernel */
#define ATAG_RAMDISK    0x54410004
/* command line: \0 terminated string */
#define ATAG_CMDLINE    0x54410009
```

Quelle: linux/arch/arm/include/uapi/asm/setup.h (Linux 3.8.6)

Übergabe an Linux

- Beispiel: ARM
- Erzeugen der ATAG-Liste
- Setzen von r0 = 0
- Speichern der *machine id* in r1
- Adresse der ATAG-Liste in r2
- Laden der Kerneladresse in den Befehlszeiger

Device Tree

- Neues Bootkonzept für ARM
- Statt board.c: Devicetree
- Übergabe der Systemkonfiguration an den Linuxkernel über Devicetree Blob (DTB)
- Abbildung von SoC und Board
  - Include-Möglichkeit
- Treiber müssen DT-Informationen verarbeiten!

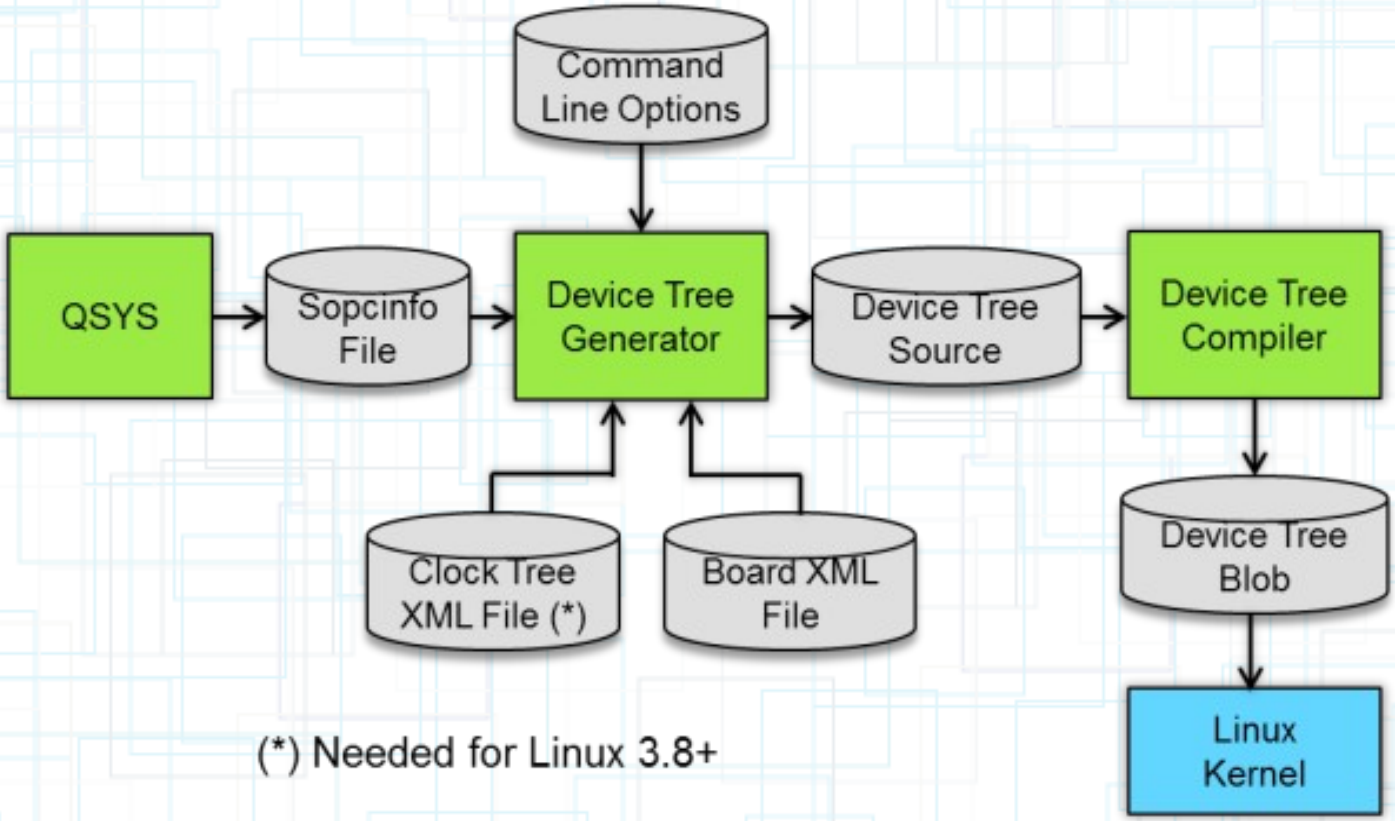
Systembeschreibung

- Zahl der CPUs
- Größe und Adresse von Speicherbereichen
- Busse und Bridge Devices
- Peripherie
- Interruptcontroller und IRQ-Konfiguration
- Spezifische Treiberparameter
  - MAC-Adresse für Ethernet

Aufbau

- Root Node
  - CPUs
  - Memory
  - System
    - UART
    - i2c
      - EEPROM
    - Clocks

Devicetree-Erzeugung



Quelle: rocketboards.org

Beispiel

```
/*
 * Device Tree for DA850 EVM board
 *
 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
 *
 * This program is free software; you can redistribute  it and/or modify it
 * under  the terms of  the GNU General  Public License as published by the
 * Free Software Foundation, version 2.
 */
/dts-v1/;
/include/ "da850.dtsi"

/ {
	compatible = "ti,da850-evm", "ti,da850";
	model = "DA850/AM1808/OMAP-L138 EVM";

	soc {
		serial0: serial@1c42000 {
			status = "okay";
		};
		serial1: serial@1d0c000 {
			status = "okay";
		};
		serial2: serial@1d0d000 {
			status = "okay";
		};
	};
};
```

Quelle: arch/arm/boot/dts/da850-evm.dts

Beispiel

```
soc {
        compatible = "simple-bus";
        model = "da850";
        #address-cells = <1>;
        #size-cells = <1>;
        ranges = <0x0 0x01c00000 0x400000>;

        serial0: serial@1c42000 {
                compatible = "ns16550a";
                reg = <0x42000 0x100>;
                clock-frequency = <150000000>;
                reg-shift = <2>;
                interrupts = <25>;
                interrupt-parent = <&intc>;
                status = "disabled";
        };
        serial1: serial@1d0c000 {
                compatible = "ns16550a";
                reg = <0x10c000 0x100>;
                clock-frequency = <150000000>;
                reg-shift = <2>;
                interrupts = <53>;
                interrupt-parent = <&intc>;
                status = "disabled";
        };
        serial2: serial@1d0d000 {
                compatible = "ns16550a";
                reg = <0x10d000 0x100>;
```

Quelle: arch/arm/boot/dts/da850.dtsi

Linux-Start

- Linux erwartet, als einziger Kontrolle über das System zu haben
- Voraussetzungen müssen erfüllt sein
  - MMU inaktiv
  - Caches deaktiviert
  - Protected mode nicht aktiviert
  - Erste Konsole initialisiert

ARM-Linux-Startphasen

- Start in drei Phasen
- zImage-Dekompression
  - Bei ARM ist der Kernel meist komprimiert
- ARM-spezifischer Code
  - Initialisierung der CPU selbst
  - Unabhängig von der Peripherie
- Prozessorunabhängiger Code

ARM-spezifischer Code

- Ermitteln des tatsächlichen Prozessortyps
  - Ein Kernelabbild könnte mehrere ARM-Typen unterstützen
  - ARMv5, ARMv6, ARMv7, ...
- Ermitteln der tatsächlichen *machine id*
  - Ein Kernelabbild könnte mehrere ARM-Ausprägungen unterstützen
  - Ein Image für mehrere unterschiedliche Systemkonfigurationen möglich

ARM-spezifischer Code 2

- Erzeugen der Seitentabellen
  - Erste MMU-Tabellen
  - Kritische Phase: Umschaltung von physikalischer zu virtueller Adressierung
- Aktivieren der MMU
  - Restrukturierung abschließen
- Start des „eigentlichen“ Linux
  - Bisher nur ARM-Initialisierung

Prozessorunabhängiger Code

- start_kernel()
- Initialisierung des Systemtimers
- Setup der Architektur
  - Nochmal architekturspezifisch
  - Low-level Interruptbehandlung
- Parsen der Kommandozeile
- Initialisierung des Speichermanagements

Prozessorunabhängiger Code 2

- Initialisierung
  - Kernel-Caches
  - Scheduler
  - Interrupts
  - Timer
  - *hrtimers*
  - Soft-IRQs, Traps, Exceptions
  - Konsole

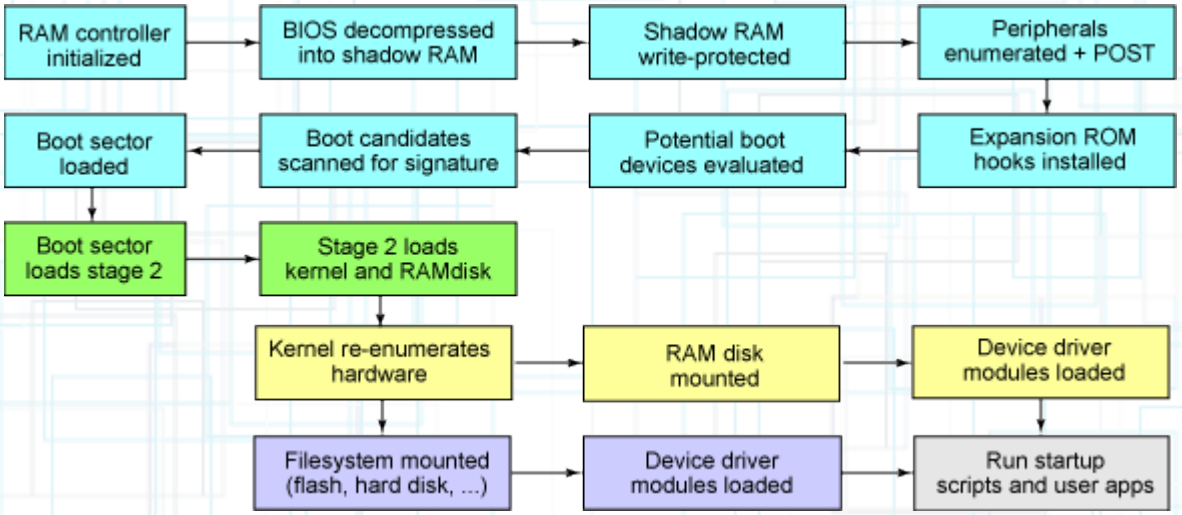
Prozessorunabhängiger Code 3

- Initialisierung
  - VFS: Virtual File System
  - Mount-Subsystem
  - Rootdateisystem
  - Zeichenbasierte Gerätetreiber
  - Signalbehandlung
  - Kernelthreads
- Einkompilierte Treiber vorbereiten

Prozessorunabhängiger Code 4

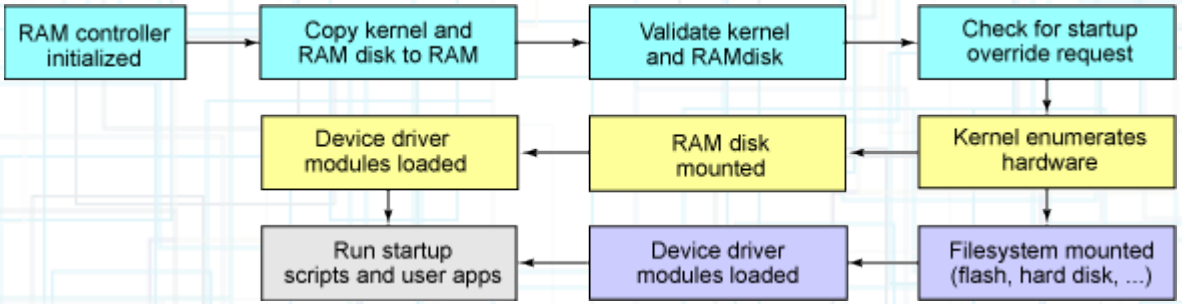
- Initialisierung aller Subsysteme
- Laden der initrd
  - Wenn verwendet
- Mounten des Rootdateisystems (rootfs)
- Starten des Userspace
  - init
  - „Vater“ aller Prozesse

x86 Schematisch...



Quelle: ibm.com

PPC/ARM Schematisch...



Quelle: ibm.com

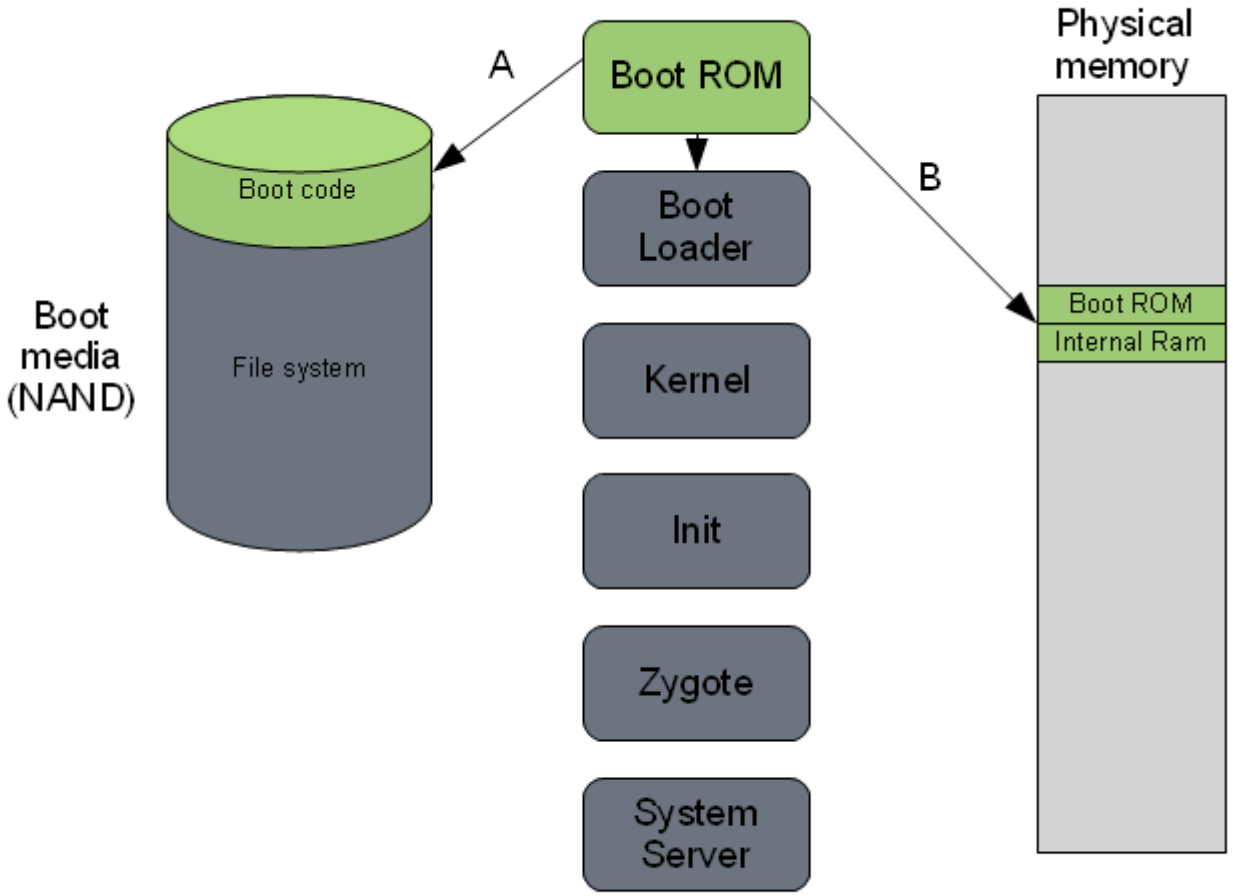
Android-Start

- Sechs Hauptschritte
  - ROM-Bootloader
  - Bootloader
  - Linux-Kernel
  - Init-Prozess
  - Zygote und Dalvik-VM
  - Systemserver

ROM-Bootloader

- Nach Power-On
  - System uninitialisiert
  - Interne Takte deaktiviert
  - Nur interner Speicher verfügbar
- CPU startet mit ROM-Bootloader
  - Prozessorspezifisch
  - Nicht anpassbar

ROM-Bootloader

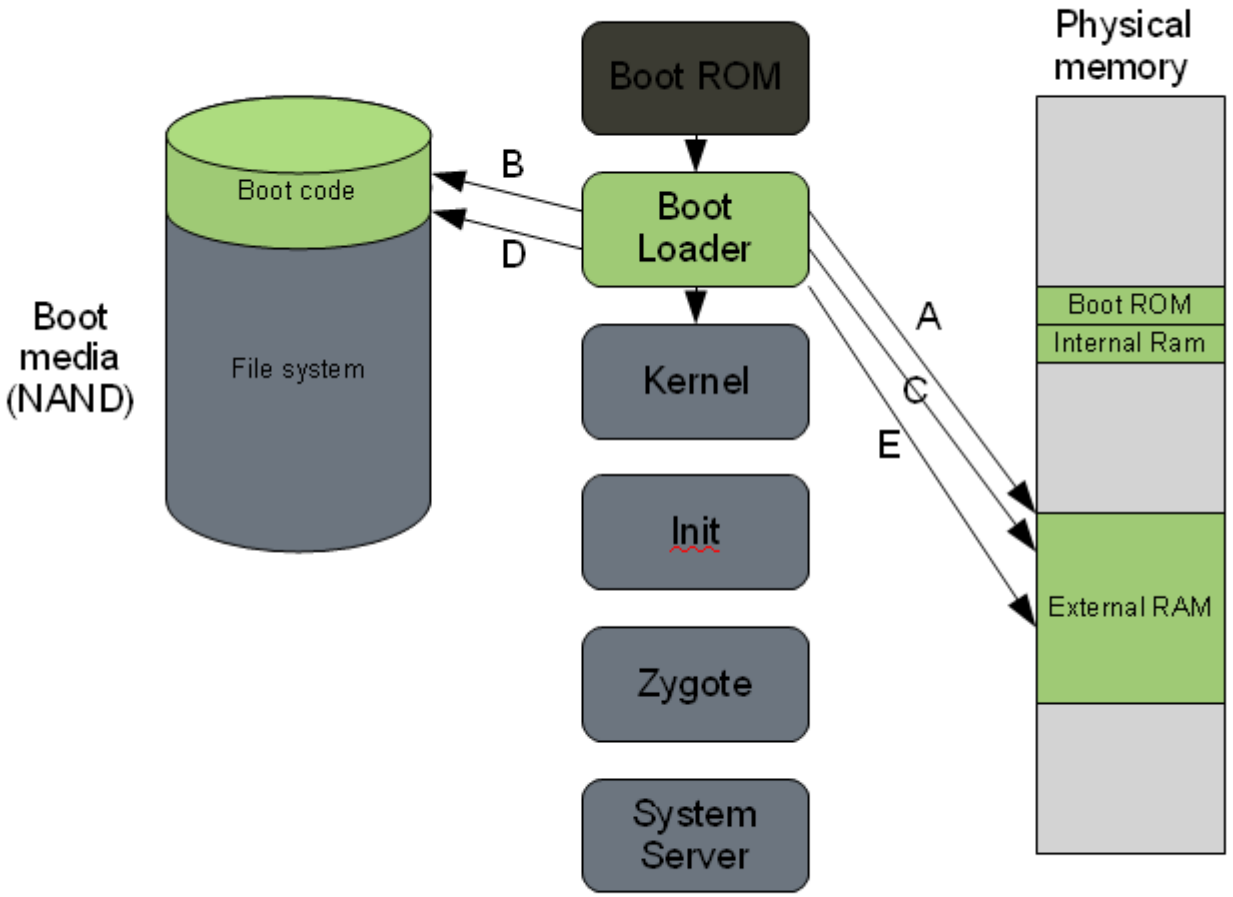


Quelle: http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html

Bootloader

- Setup des externen Speichers
- Laden des zweiten Teils des Bootloaders
- Laden des Codes für das Modem
  - Oft eigene CPU (Basisbandprozessor)
- Laden des Linux-Kernels
- Erzeugen der Initialisierungsinformation
- Starten des Linux-Kernels

Bootloader

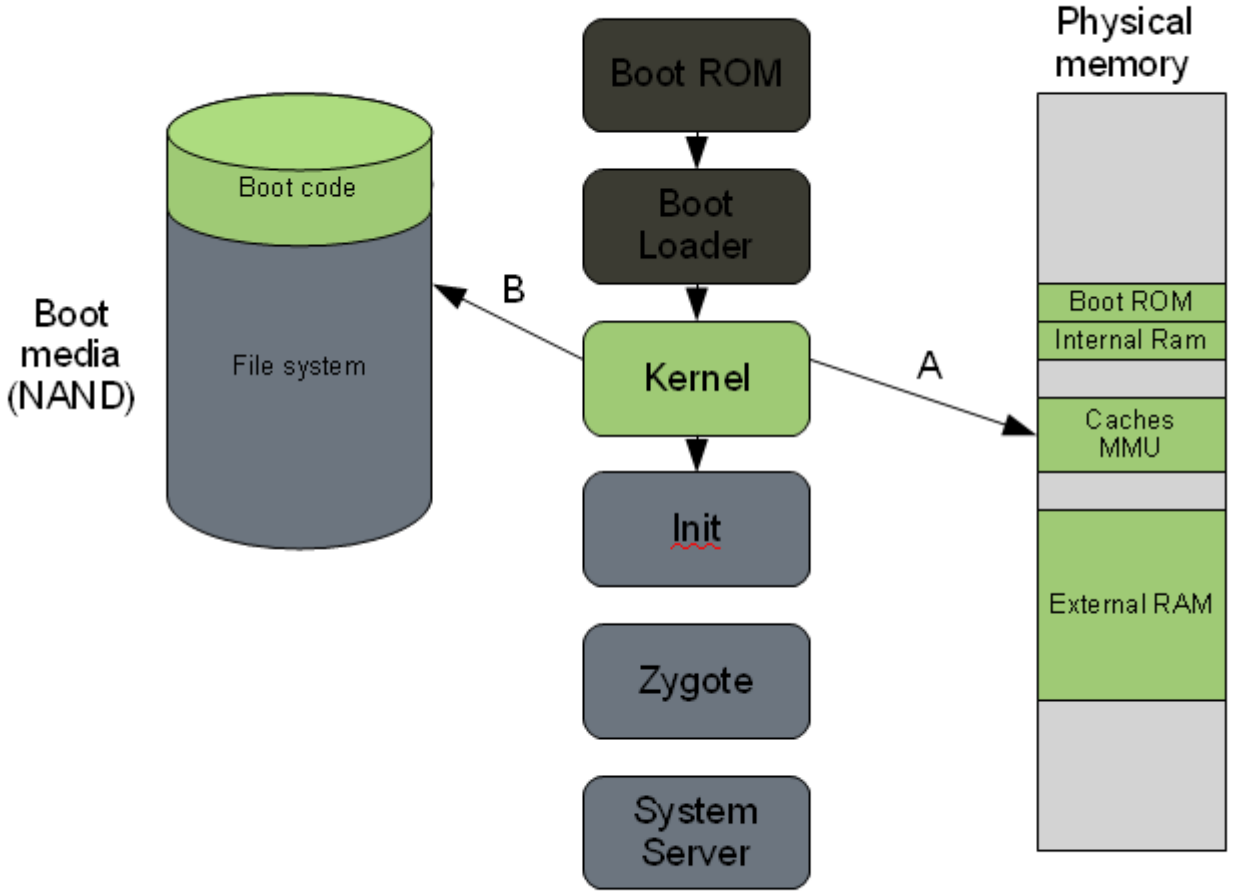


Quelle: http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html

Linux-Kernel

- Initialisieren des Systems
  - Speicherverwaltung
  - Caches
- Mounten des Rootdateisystems
- Laden und starten des Init-Prozesses

Linux-Kernel

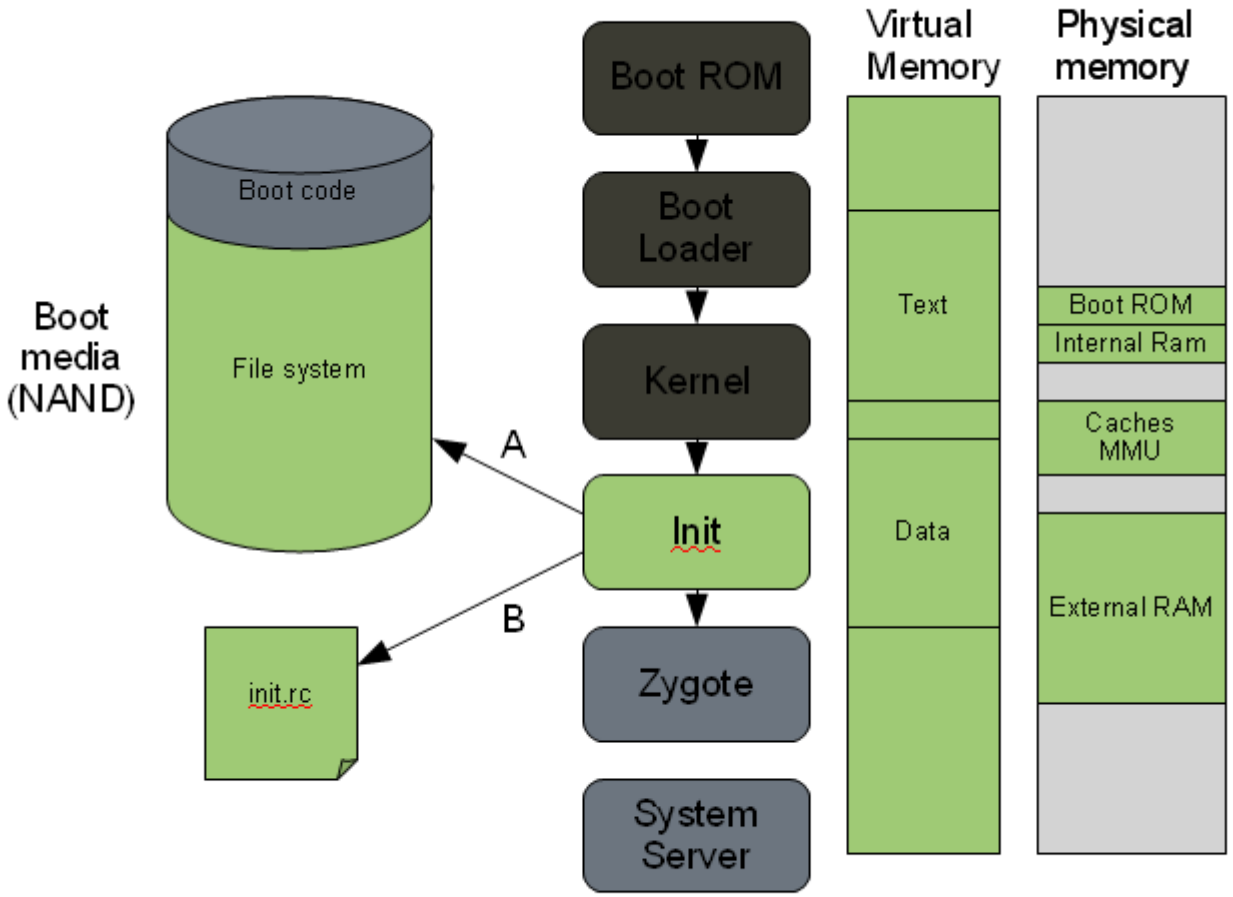


Quelle: http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html

Init-Prozess

- „Vater“ aller Prozesse im System
- Sucht im Rootdateisystem nach Konfiguration
  - init.rc: Konfigurationsskript
- Start von Systemdiensten
- Mounten zusätzlicher Dateisysteme

Init-Prozess

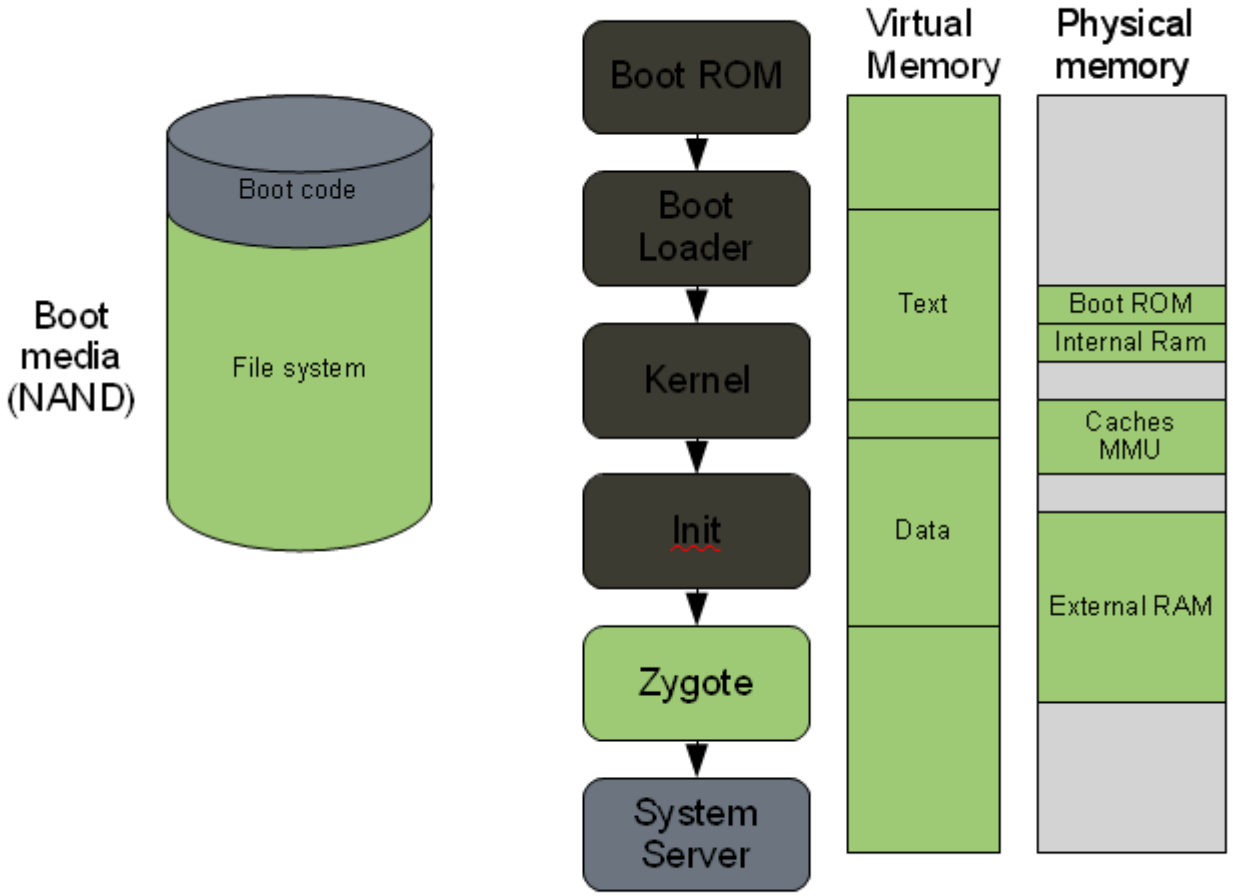


Quelle: http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html

Zygote und Dalvik-VM

- Zygote hat nur eine Aufgabe:
  - Start der Dalvik-VM
- Dalvik-VM
  - Virtuelle Registermaschine
  - Eigenes Bytecodeformat
  - Ressourcenschonend und effizient
  - Eine Dalvik-VM für jedes Android-Programm

Zygote und Dalvik-VM

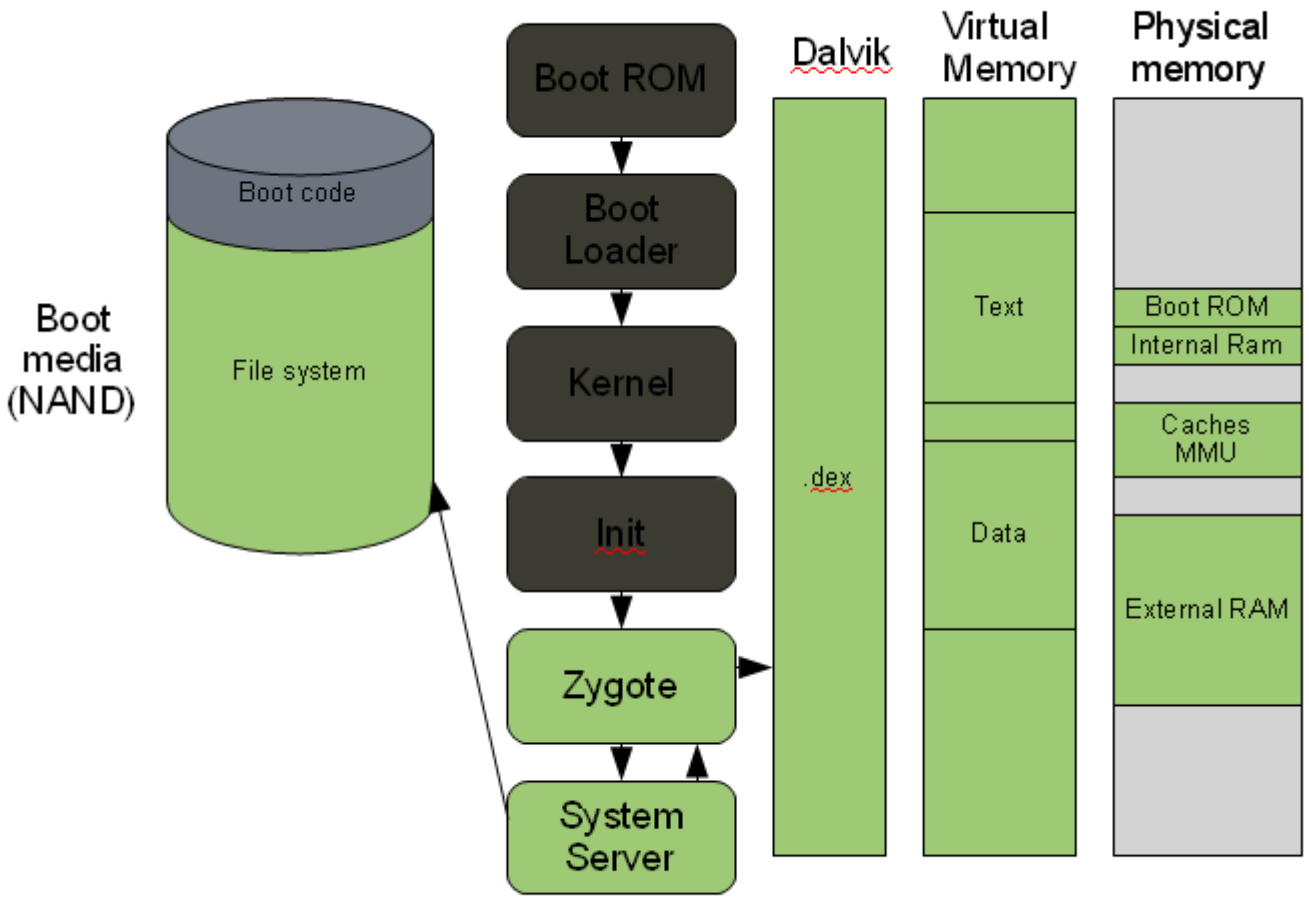


Quelle: http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html

Systemserver

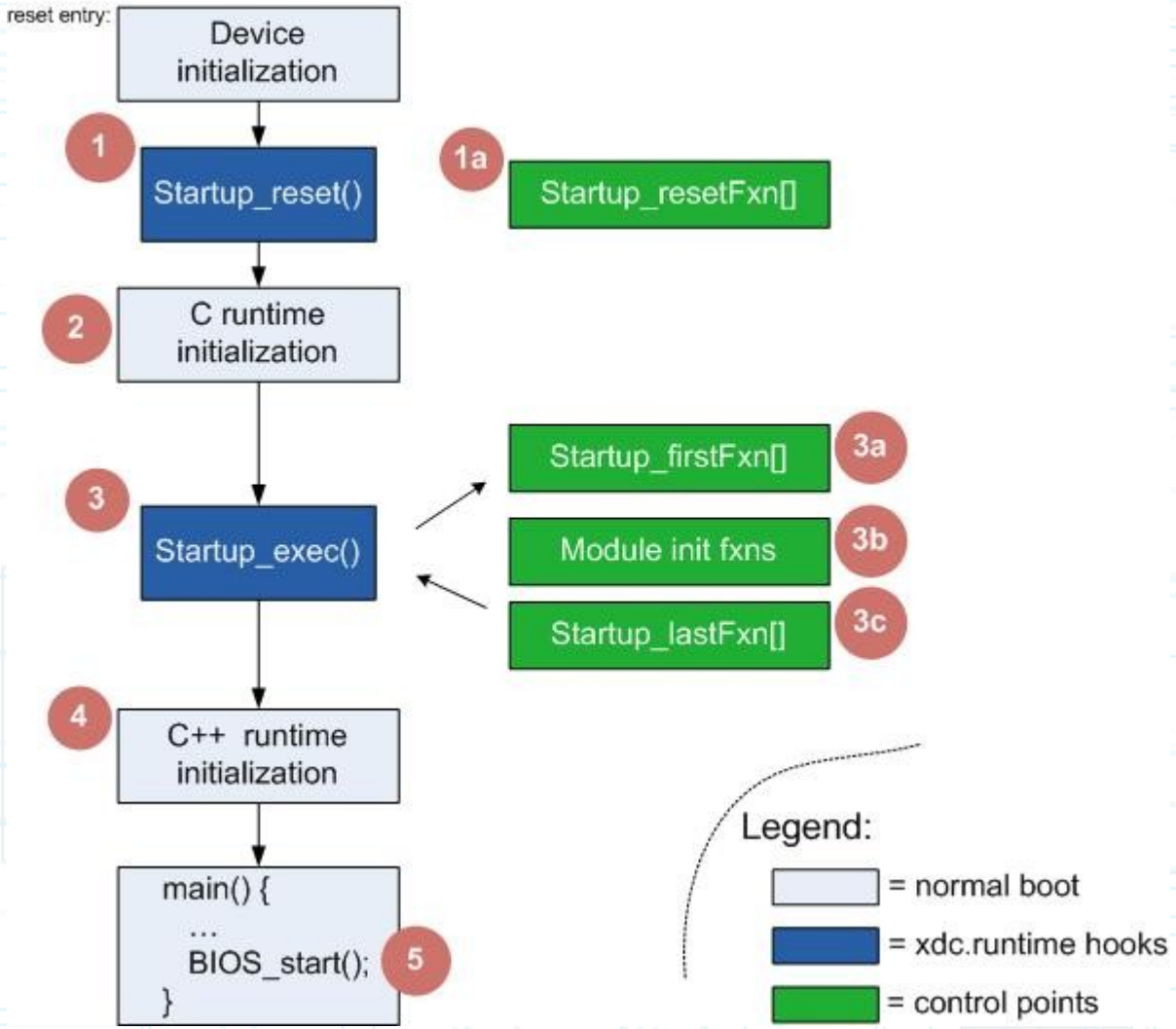
- Erste Java-Komponente im System
- Start aller Android-Dienste
  - *Telephony Manager*
  - Bluetooth
  - Medienserver
  - ...

Systemserver



Quelle: http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html

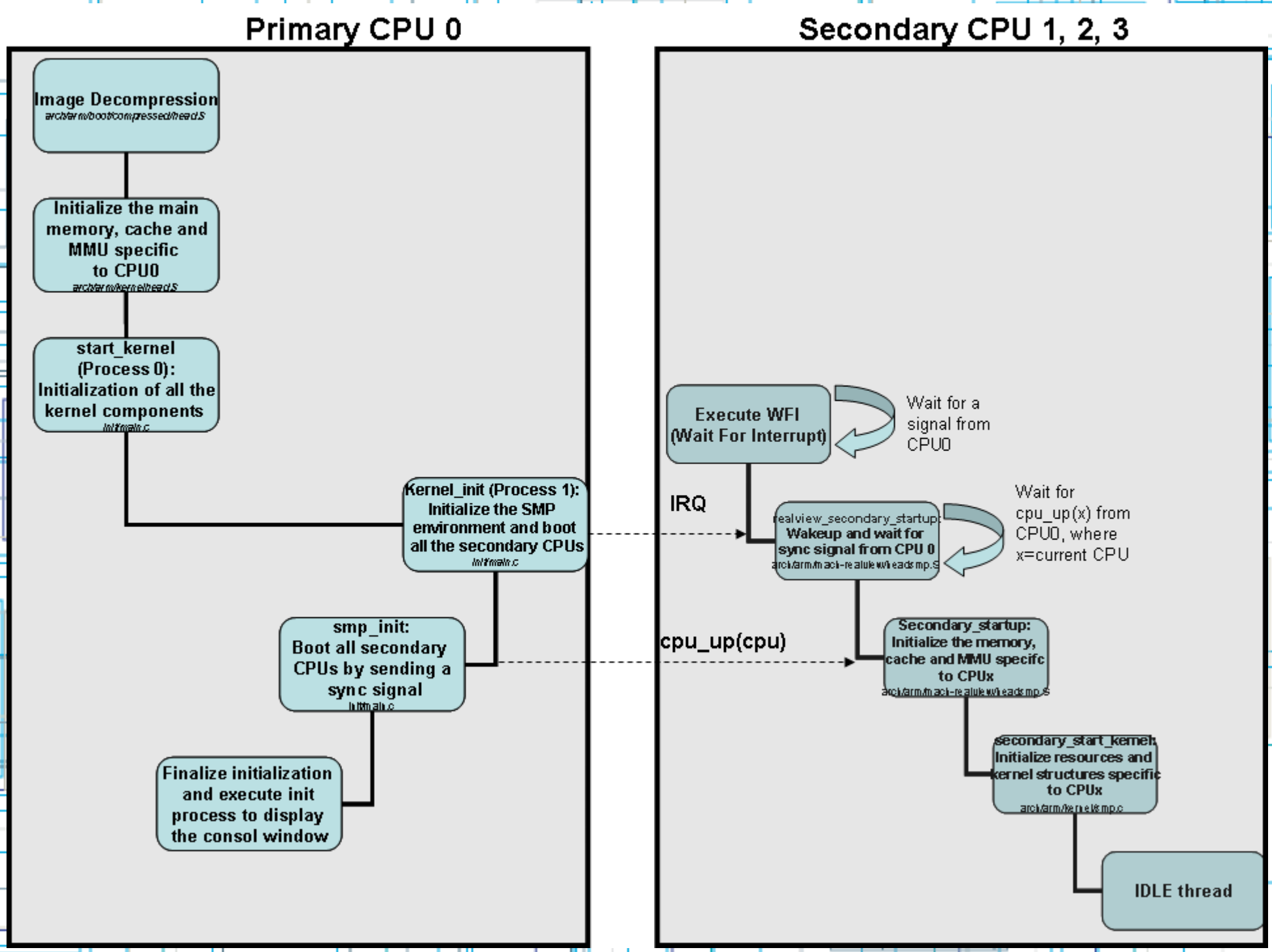
SYS/BIOS



Quelle: ti.com

Start bei SMP

- SMP: Symmetrisches MultiProcessing
- Mehrere CPUs laufen unter Kontrolle eines Betriebssystems
- Sowohl Anwendungen als auch Betriebssystem auf allen Kernen
- Gegenteil: AMP, Asymmetrisches MultiProcessing



Quelle: http://www.linux-arm.org/LinuxBootLoader/SMPBoot