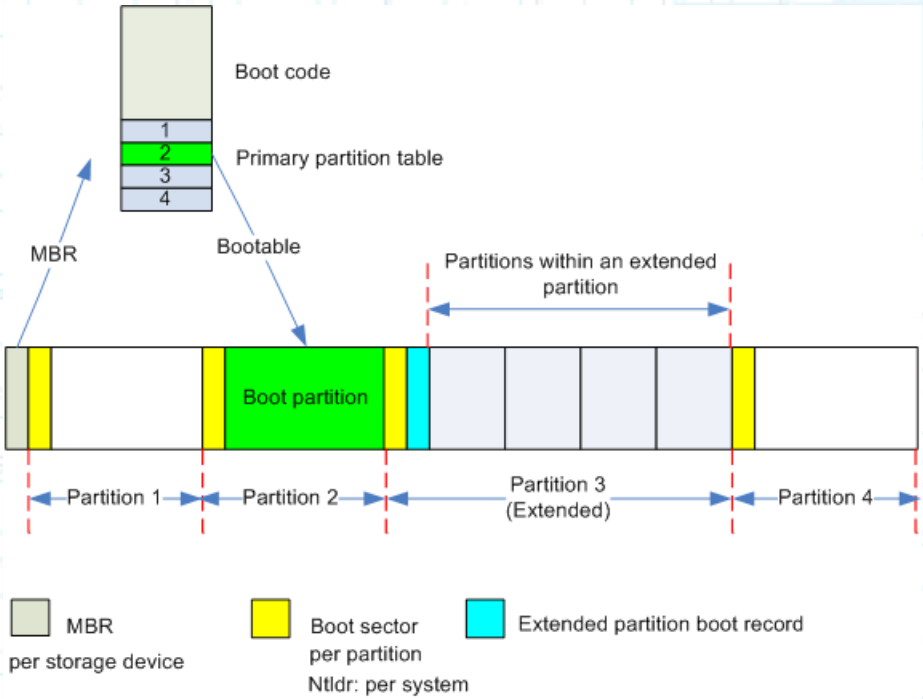
Eingebettete Betriebssysteme

Bachelorstudiengang
Technische Informatik
Hochschule Pforzheim

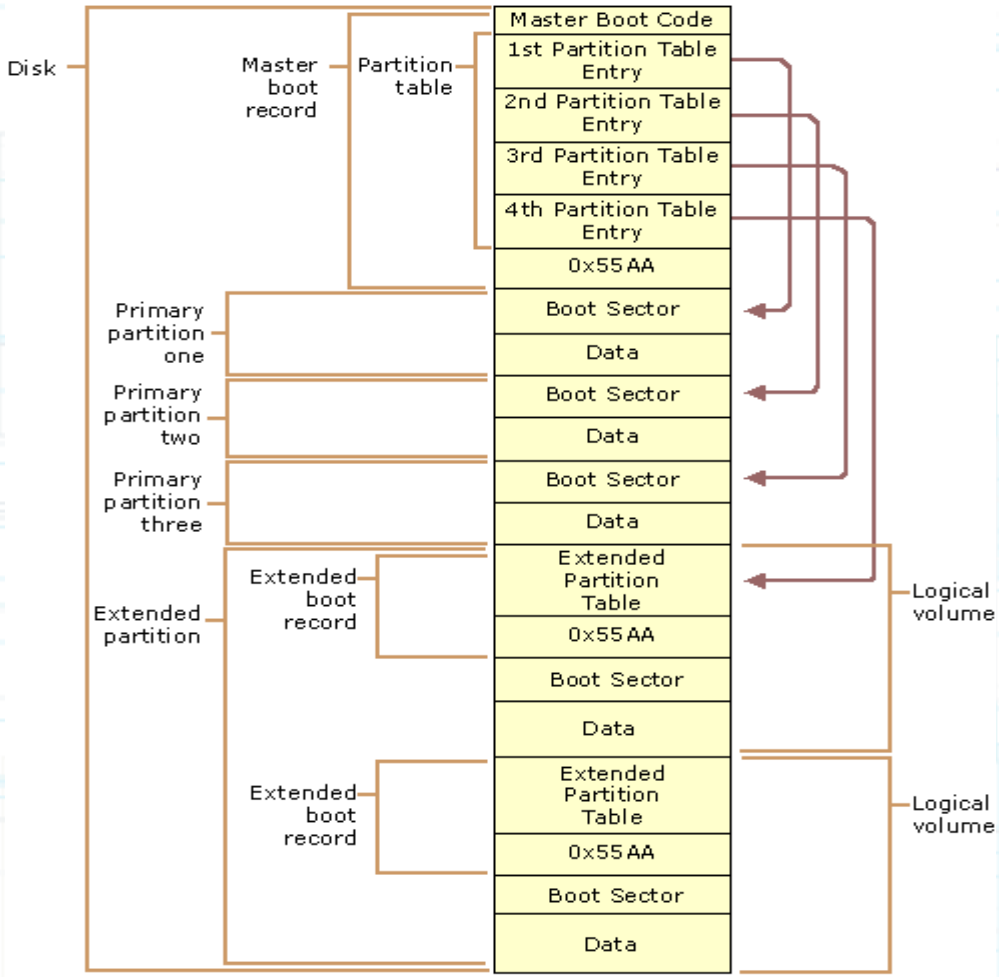
Dipl.-Ing.(FH) Marc Jüttner

Partitionierung



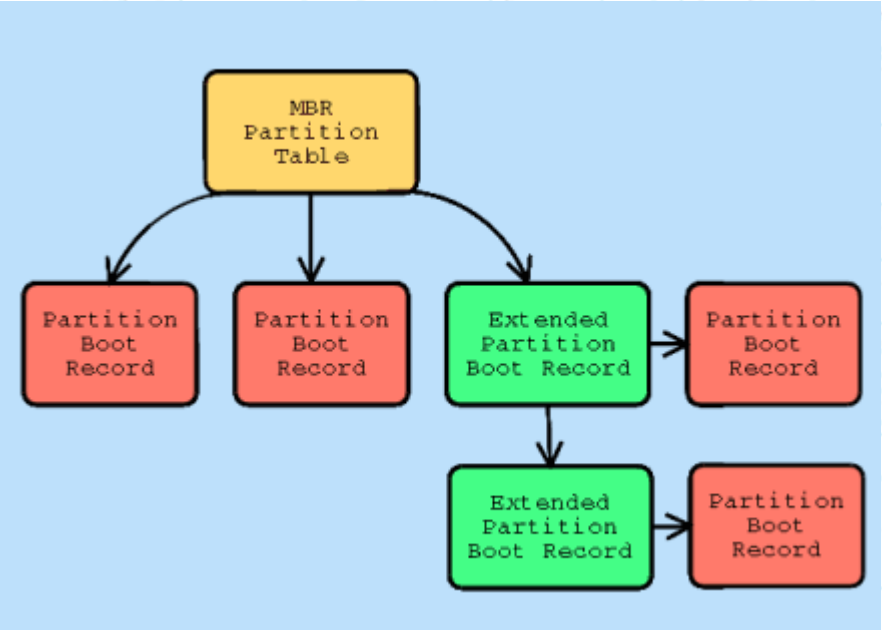
Quelle: dralu.com

Partitionstabelle



Quelle: MS TechNet

Partitionierung



Quelle: techrepublic.com

Master Boot Record

- In den ersten 512 Bytes eines Datenträgers
- Scan der Partitionstabelle
  - Aktive Partition ermitteln
- Startsektor der aktiven Partition ermitteln
- Bootsektor in den Speicher laden
- Ausführen des Codes
  - Startet das Betriebssystem

Flash-Partitionierung

- Für Flash existiert kein Partitionierungsschema
  - Speichern der Partitionstabelle könnte fehlschlagen
  - Partitionstabelle könnte defekt sein
- Partitionierung erfolgt meist hartkodiert
- Woher kennt ein HLOS die Partitionierung?

Flash-Partitionierung Linux

- Mehrere Möglichkeiten
  - Übergabe über Kommandozeile durch Bootloader
  - Hartkodierung in der Board-Implementierung
  - Devicetree

Übergabe durch Bootloader

- Formatierung einer Kommandozeilenoption für Linux
  - 2MiB für U-boot
  - 30MiB für den Linux-Kernel
  - Rest für Userspace
- Flash-Treiber muss einkompiliert sein!

```
mtdparts=atmel_nand:2M@0x40000(u-boot),
30M@0x200000(kernel),-@0x2000000(user),
```

Flashlayout TI DM365-EVM

```
#define NAND_BLOCK_SIZE		SZ_128K

static struct mtd_partition davinci_nand_partitions[] = {
	{
		/* UBL (a few copies) plus U-Boot */
		.name		= "bootloader",
		.offset		= 0,
		.size		= 30 * NAND_BLOCK_SIZE,
		.mask_flags	= MTD_WRITEABLE, /* force read-only */
	}, {
		/* U-Boot environment */
		.name		= "params",
		.offset		= MTDPART_OFS_APPEND,
		.size		= 2 * NAND_BLOCK_SIZE,
		.mask_flags	= 0,
	}, {
		.name		= "kernel",
		.offset		= MTDPART_OFS_APPEND,
		.size		= SZ_4M,
		.mask_flags	= 0,
	}, {
		.name		= "filesystem1",
		.offset		= MTDPART_OFS_APPEND,
		.size		= SZ_512M,
		.mask_flags	= 0,
	}, {
		.name		= "filesystem2",
		.offset		= MTDPART_OFS_APPEND,
		.size		= MTDPART_SIZ_FULL,
		.mask_flags	= 0,
	}
	/* two blocks with bad block table (and mirror) at the end */
};
```

Quelle: Linux 3.9 arch/arm/mach-davinci/board-dm365evm.c

USB-Sticks...

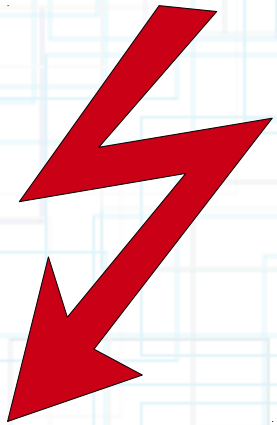
- ...sind doch auch Flash?
- Richtig! Aber
  - Flash-Controller verbirgt tatsächliche Architektur
    - Wear Leveling, ECC, ...
  - Daher auch FAT32/NTFS möglich, trotz fehlender Flash-Unterstützung!

Firmware-Updates

- Austausch der Firmware eingebetteter Systeme
  - In der Entwicklung
  - Im Feld (Kundenbetrieb)
- Hohe Anforderungen im Vorfeld
  - Tests auf Fehlerfreiheit
  - Durch Updates eingeschleppte Fehler können sich fatal auswirken!

Trivialer Ansatz

- Start eines Update-Programms
- Entgegennehmen eines Abbilds
- Beim Entgegennehmen überschreiben des alten Abbilds
- Fertig…



Entwicklung

- Austausch erfolgt unter Kontrolle des Entwicklers
- Fehler können korrigiert werden
  - Direkter Flashzugriff
  - Laden von Reparaturcode via JTAG
  - Erzwungenes Löschen von Flash-Daten

Firmwarespeicher

- Häufige Firmwarespeicher sind
  - Persistente Medien
    - Flash, SD-Karte, Festplatte
- Flash kann altern
- Fehlerkorrekturmechanismen benötigen mindestens einen Teil fehlerfreien Speichers

Im Feld

- Gerät kann im Einsatzgebiet verbaut sein
- Kein Zugang zu Debug-Schnittstellen
  - Serielle Konsole
  - JTAG
- Fehler beim Update können Systemstart unmöglich machen
  - Wie also kann ein Update sicher durchgeführt werden?

Fehlermöglichkeiten

- Unterbrechung der Stromversorgung
- Fehlerhafte Firmware
- *Flash corruption*
  - Flash kann altern!
- Kommunikationsfehler
  - Bei Remote-Update

Störung der Stromversorgung

- Firmwareabbild nur teilweise geschrieben
- Inkonsistente Daten
  - Mit unbekanntem Fehlerort
  - Möglicherweise nicht unmittelbar bemerkbar
- Kann man einen Ausfall der Stromversorgung überhaupt vermeiden?

Ansätze...

- Verwendung einer *fail-safe firmware*
  - Firmware, die nie überschrieben wird
  - Notfalllösung: Starten, wenn ein Fehler bemerkt wurde
- Erkennen des Ausfalls der Stromversorgung
- Atomare Mechanismen für das Update
- *Journalling filesystem*

Normaler Start

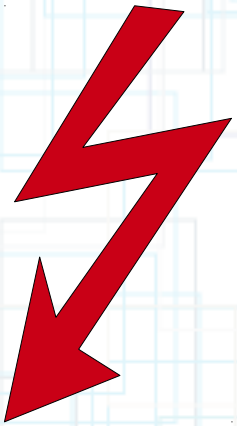
- Normaler Start
- Umschalten in Update-Modus
- Vorbereitende Maßnahmen
  - Sicherung der Konfiguration

Start der fail-safe firmware

- Start des Updateprogramms
  - Neue Firmware empfangen
  - Vorher empfangene Firmware
  - Schreiben der neuen Firmware

Firmware-Update

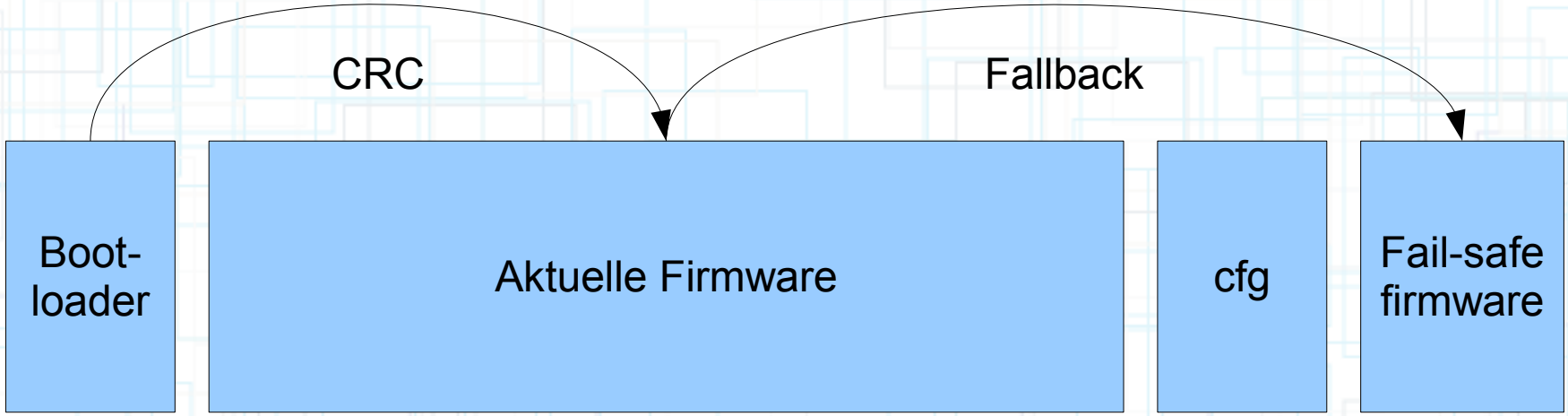
- Schreiben der neuen Firmware
- Fehlererkennung erforderlich
  - Stromversorgung
  - Datenfehler
  - *Dirty-Marker*



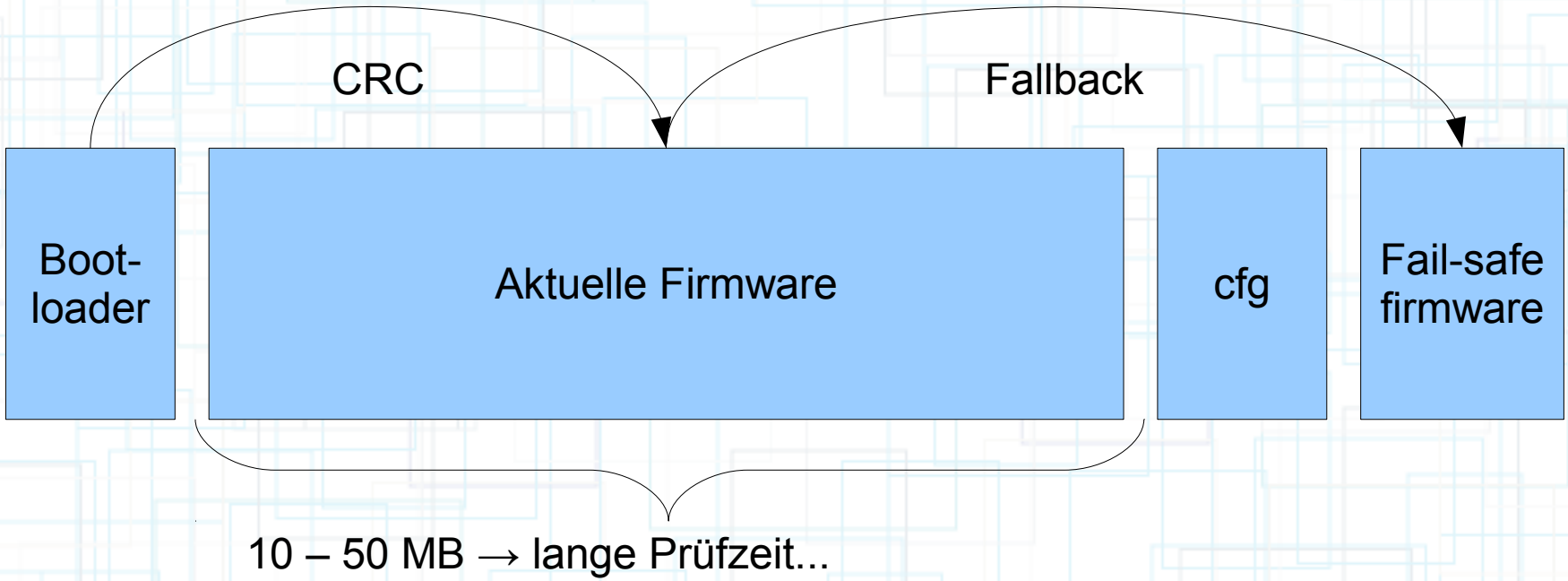
Fehlerbehandlung

- Vor dem Schreiben: Dirty-Marker setzen
  - Zeigt an, dass ein Schreibvorgang gestartet wurde
- Schreiben der Firmware wird unterbrochen
  - Automatischer Neustart der fail-safe firmware durch den Bootloader
  - Erneutes Schreiben
- Danach: Dirty-Marker löschen

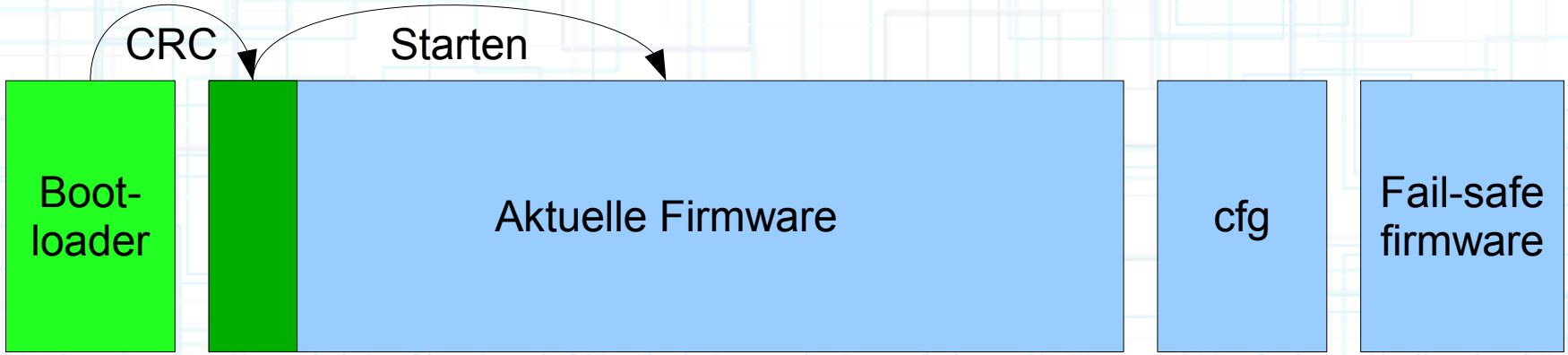
Prüfsummen


CRC
Fallback
Boot-
loader
Aktuelle Firmware
cfg
Fail-safe
firmware

Fallstrick...

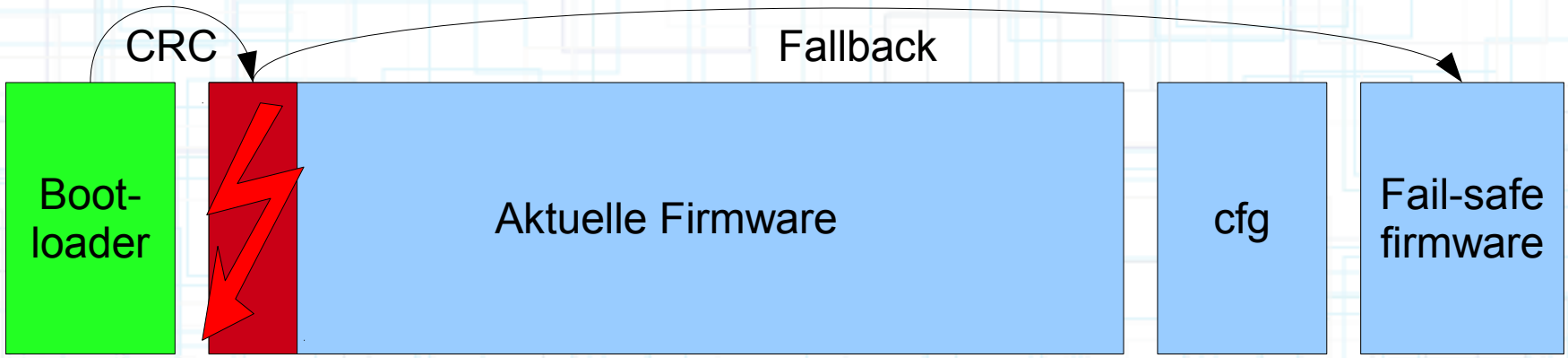

CRC
Fallback
Boot-loader
Aktuelle Firmware
cfg
Fail-safe firmware
10 – 50 MB → lange Prüfzeit...

Prüfsummen



- Speichern einer Prüfsumme
- Prüfsumme nur über einen Teil der Firmware (*firmware header*)
  - Deutlich weniger Zeitaufwand
  - Fehler trotzdem erkennbar?

Prüfsummen

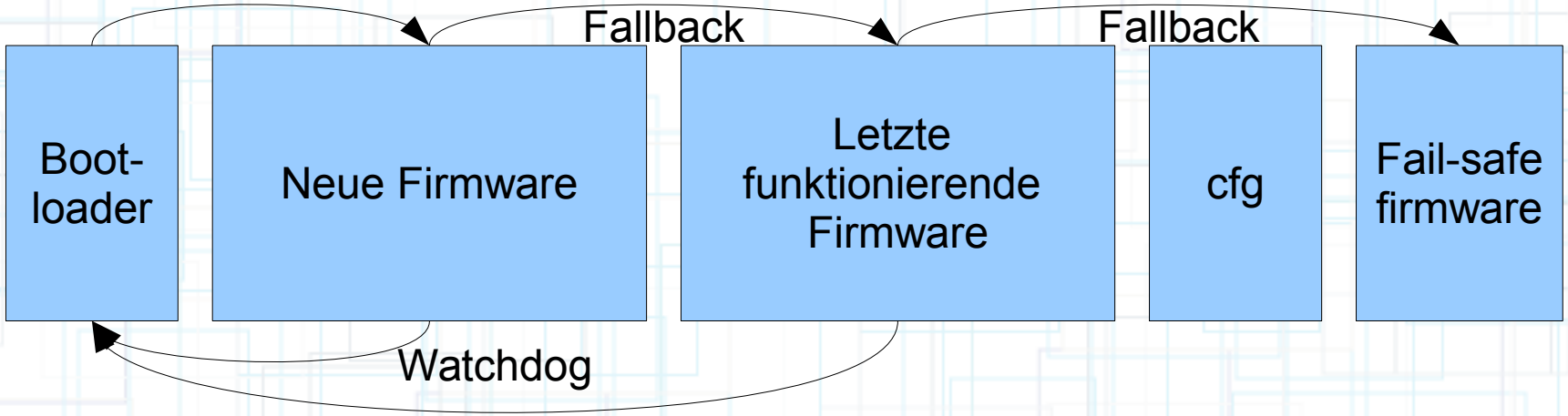


- Bei fehlerhafter Prüfsumme
  - Automatischer Start der *failsafe firmware*

Defekte neue Firmware

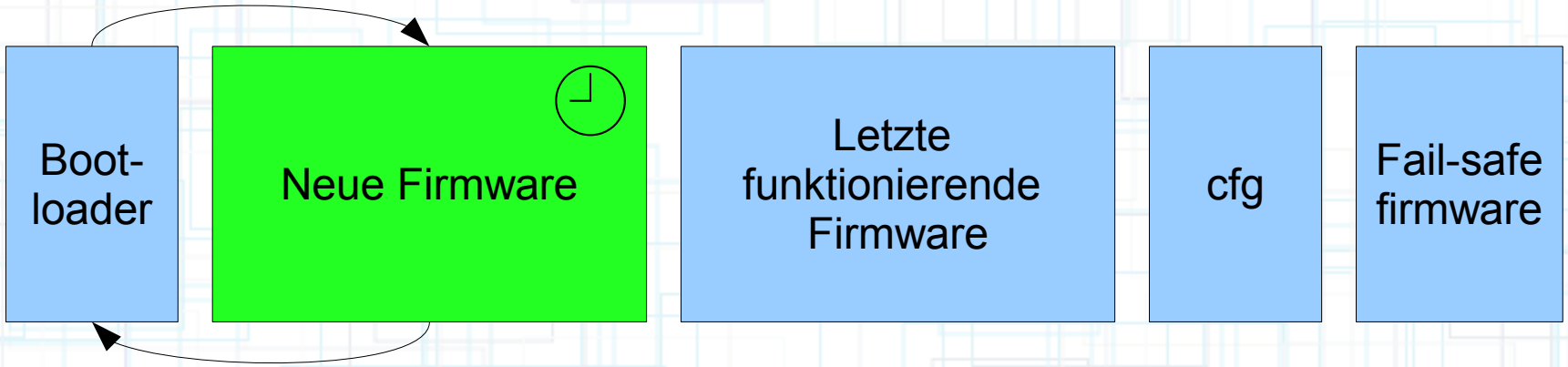
- Ausfall der Firmware auf einigen Geräten
  - Nicht zwangsläufig auf allen Ausstattungsvarianten!
- Abhilfemöglichkeit wäre, die „alte“ Firmware wieder zu starten
- Erneute Updates mit *failsafe firmware*
- Aber was, wenn Firmware „nur“ stehenbleibt?

Mehrere Firmwares



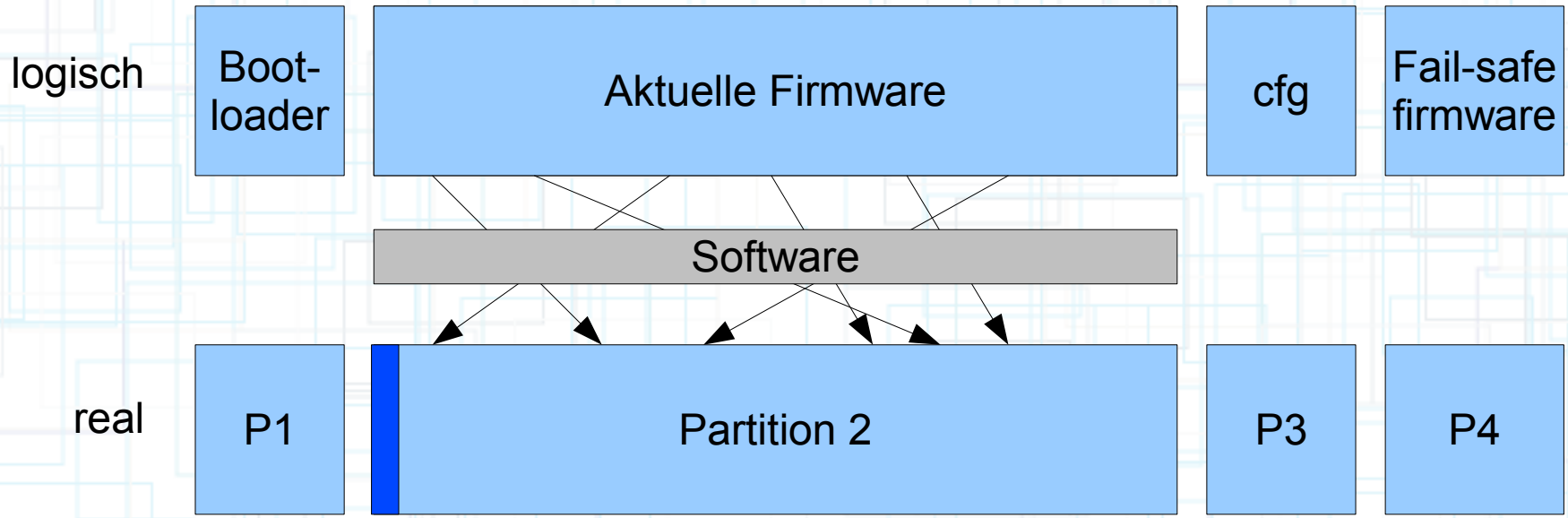
- Mehrere Firmwares zur Auswahl
  - Aber: Wie entscheiden?

Start mit Watchdog



- Normaler Start mit *force reboot*
  - Mit aktiviertem Watchdog
  - Reset, wenn Watchdog abläuft
  - Abschalten, wenn Firmware korrekt läuft

Mit Flashdateisystem

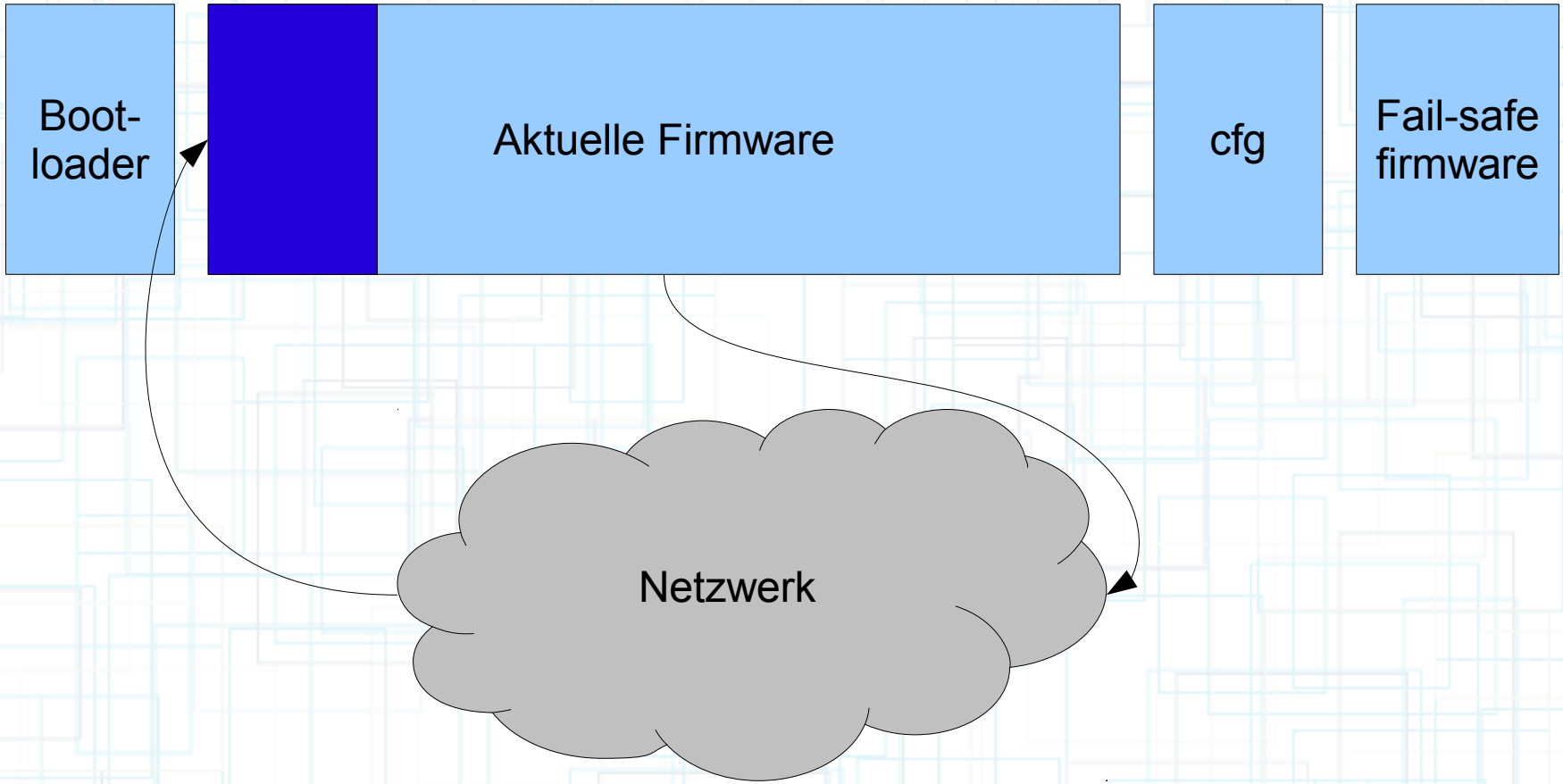


- Softwareaufwand erforderlich
- Verwaltungsinformationen benötigt

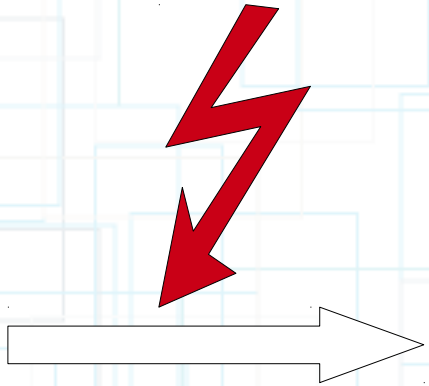
Fallstricke

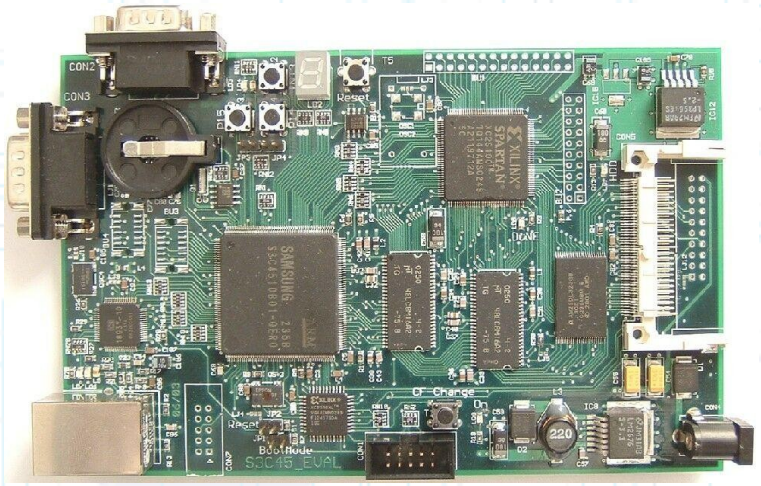
- Wichtige Fragestellungen
  - Wo liegt die Partitionstabelle?
  - Wie wird die Partitionstabelle gesichert?
  - Was tun bei Ausfall?
- Statisches Layout im Bootloader
  - Kann nie wieder geändert werden
  - Wenn doch: Datenkonvertierung
    → Konfigurationspartition!

Ablauf


Boot-
loader
Aktuelle Firmware
cfg
Fail-safe
firmware
Netzwerk

Kommunikationsfehler

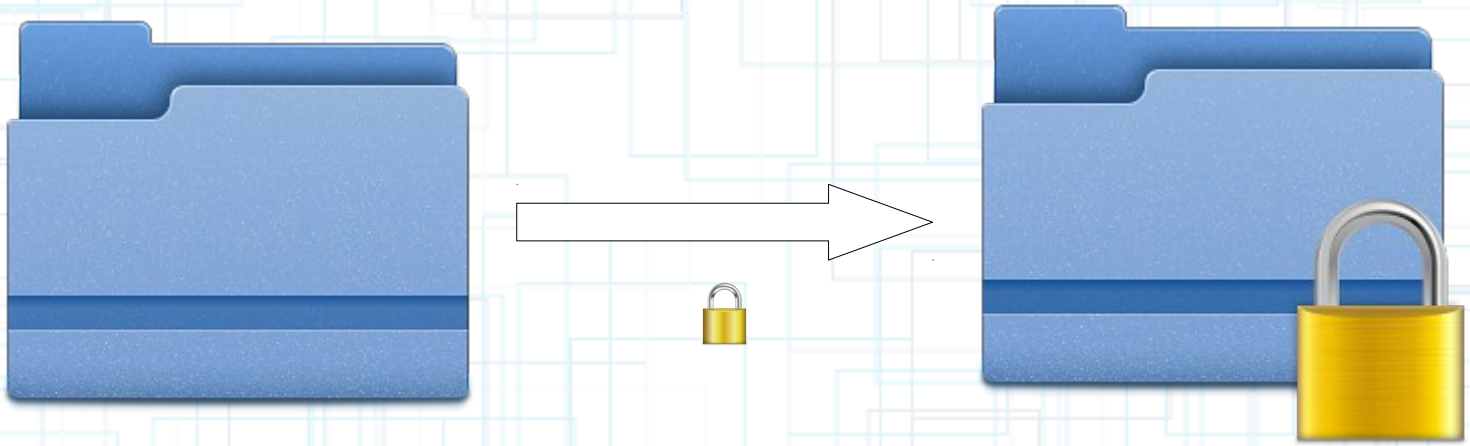




Fehlermöglichkeiten

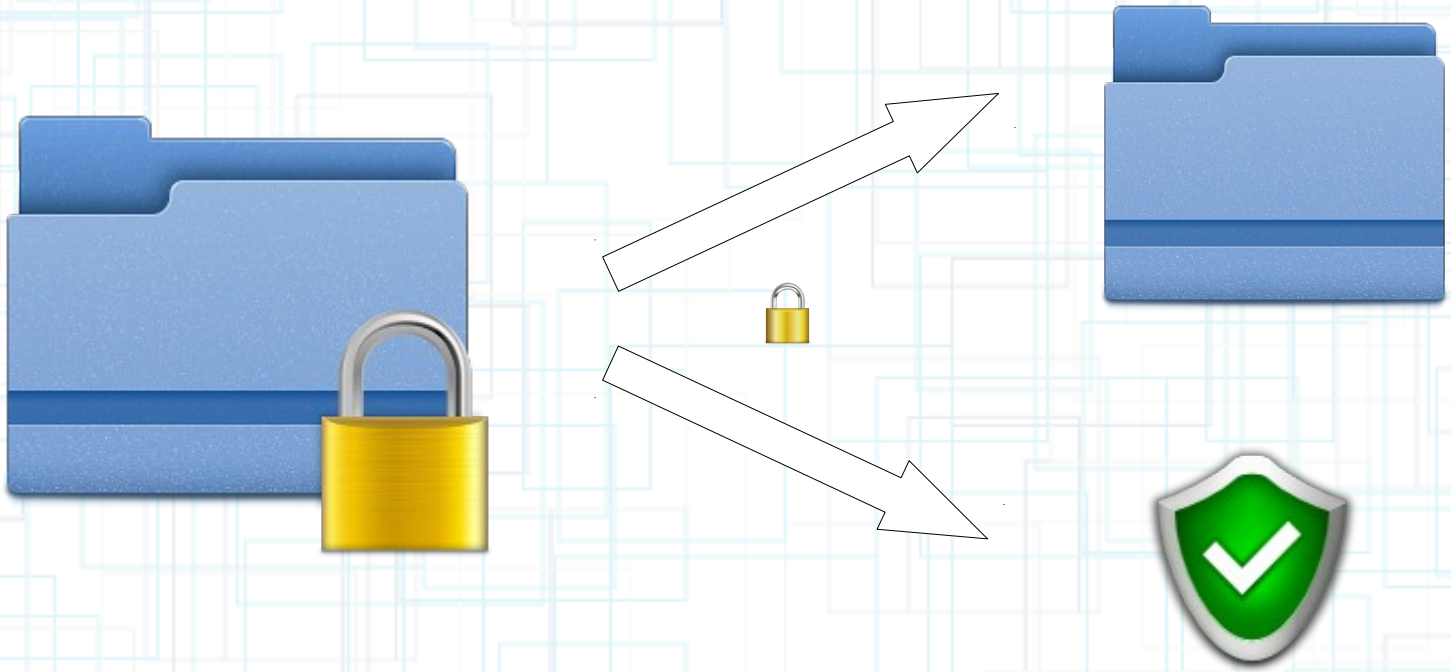
- Fehler bei der Datenübertragung
  - Bitfehler in komprimiertem Datenstrom
  - Bitfehler mit folgenden falschen Prozessorbefehlen
- Mutwillige Einbringung von Fehlern
  - Man-in-the-middle

Lösung: Signaturen



- Signieren des Abbilds mit privatem Schlüssel
  - `gpg --sign <abbild>`

Beim Update



- Prüfung mit öffentlichem Schlüssel
  - `gpg < <abbild> > <firmware>`

Zu beachten

- Vorsicht mit signierten Dateien!
- Möglichkeit zur Installation neuer Schlüssel ist erforderlich
  - Signierschlüssel können ablaufen
  - Third party upgrades
- Schlüsselrevokation
  - Signierschlüssel könnte gestohlen werden

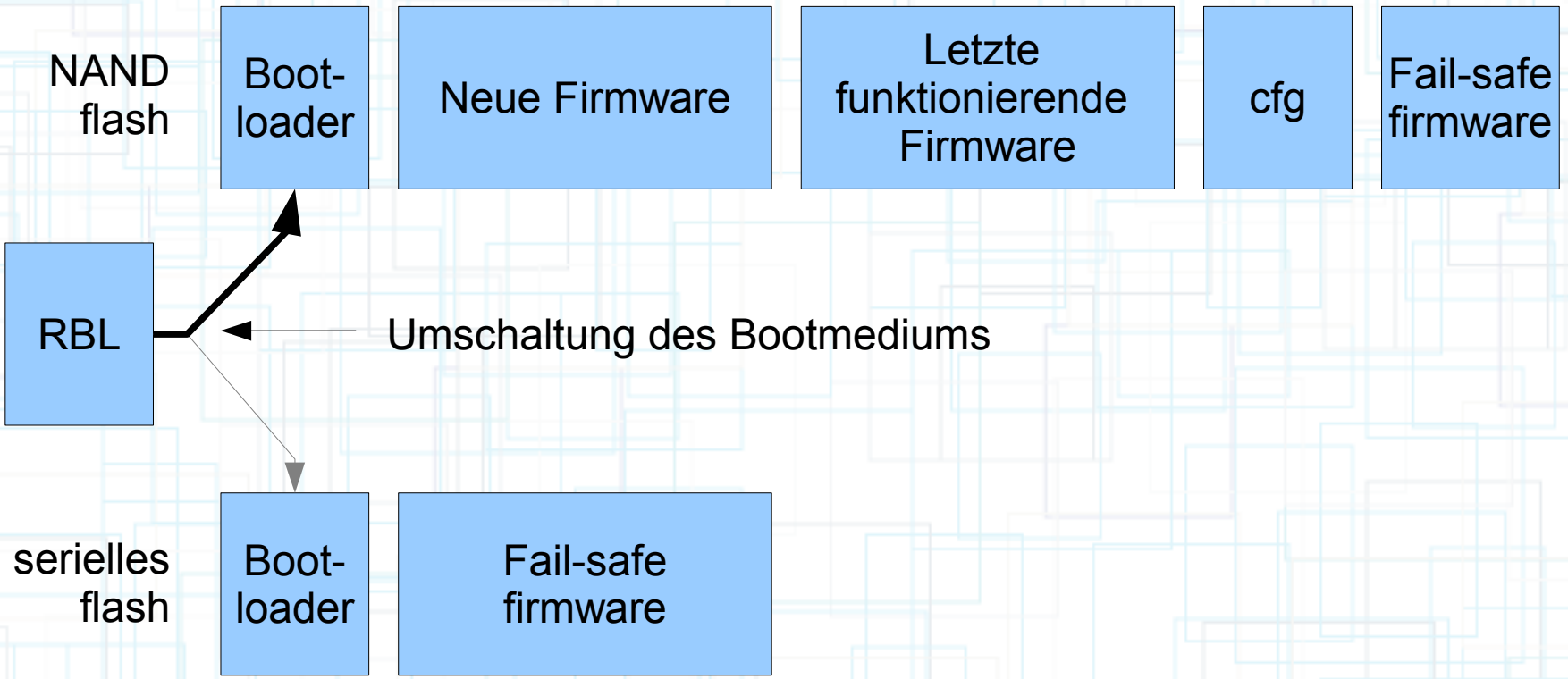
Zu beachten

- Neue Schlüssel und Revokationsschlüssel müssen der *failsafe firmware* zugänglich gemacht werden!
- Signaturberechnung kann speicherintensiv sein
  - Aufsplitten in Teilpakete

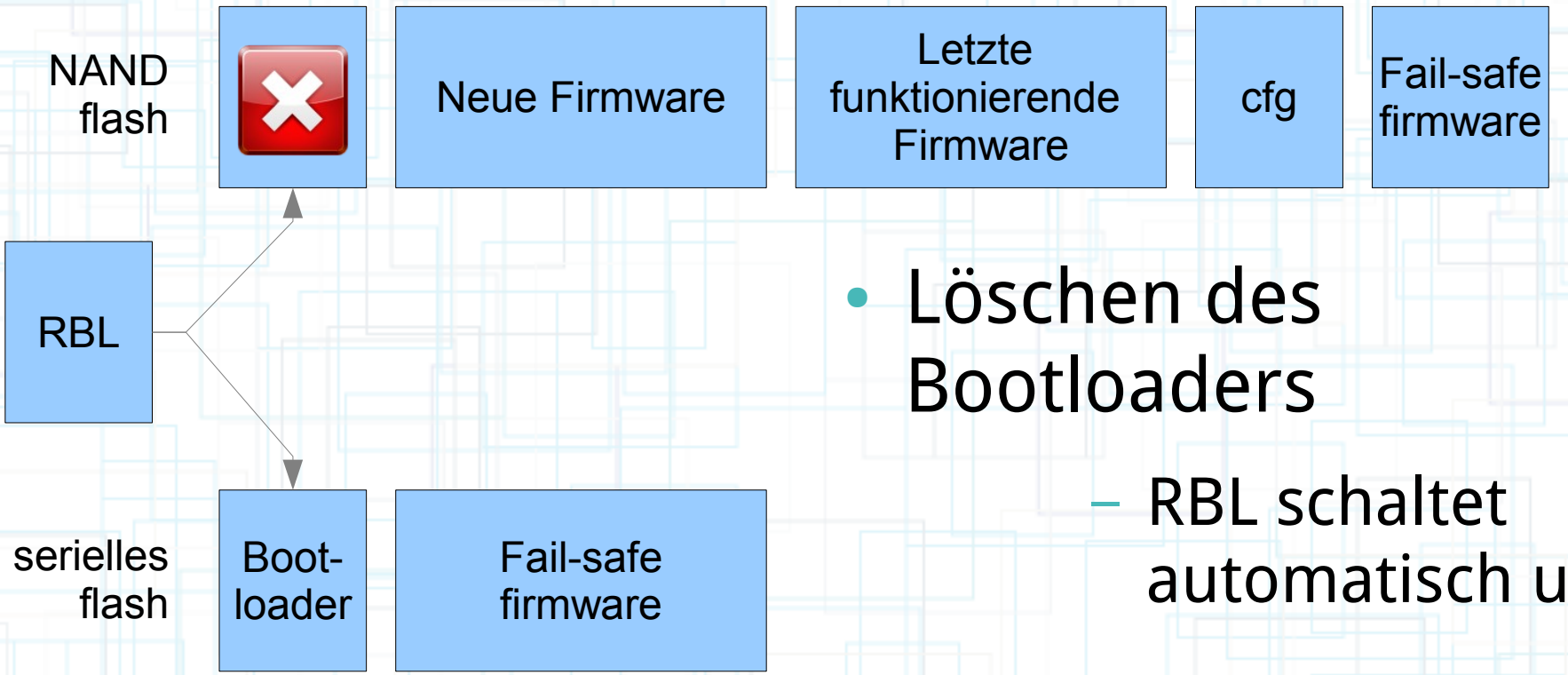
Bootloader-Update

- Sehr riskantes Unterfangen
- Was bei defektem Bootloader?
  - Kein Systemstart mehr möglich
  - **Reparatur im Feld ist unmöglich!**
- Also
  - Keine Fehler im Bootloader
  - Keine Features im Bootloader!

Backup-Lösung

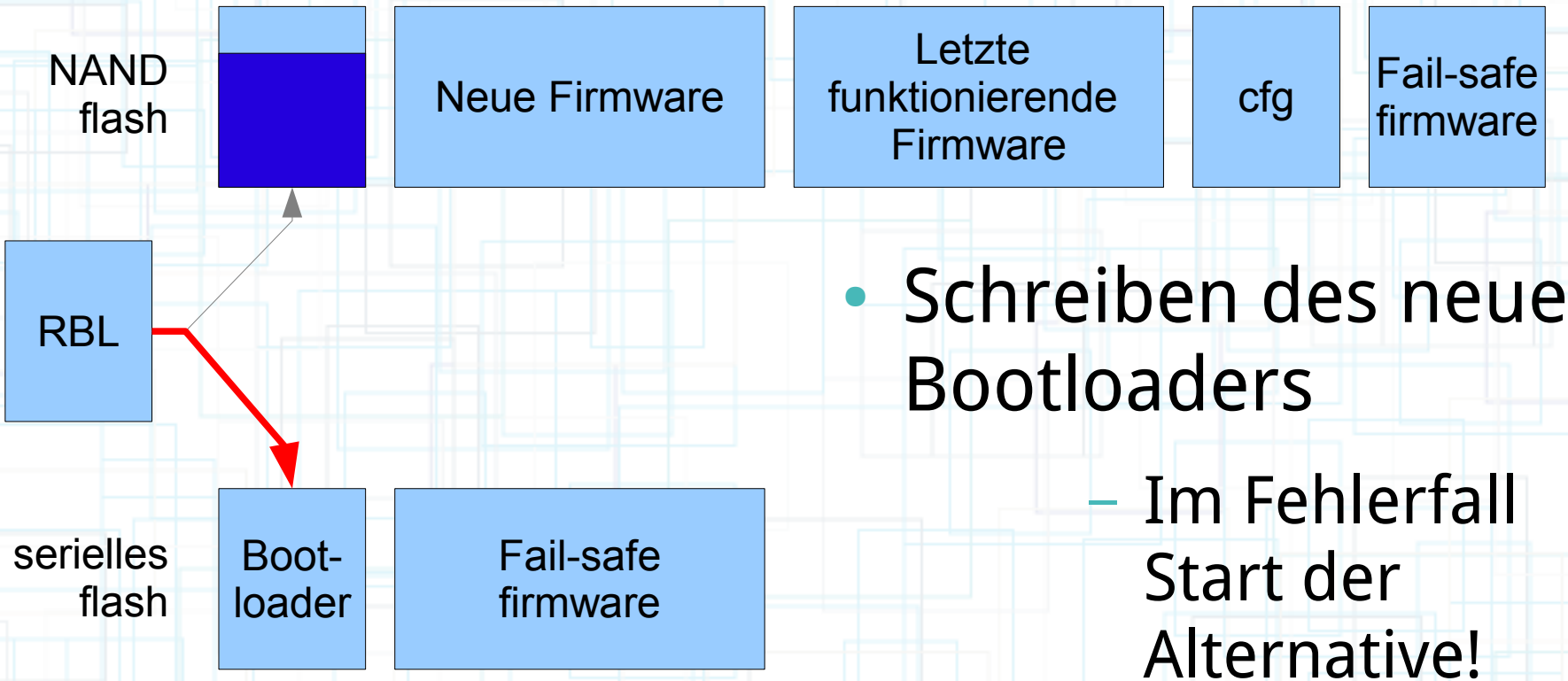


Ablauf 1



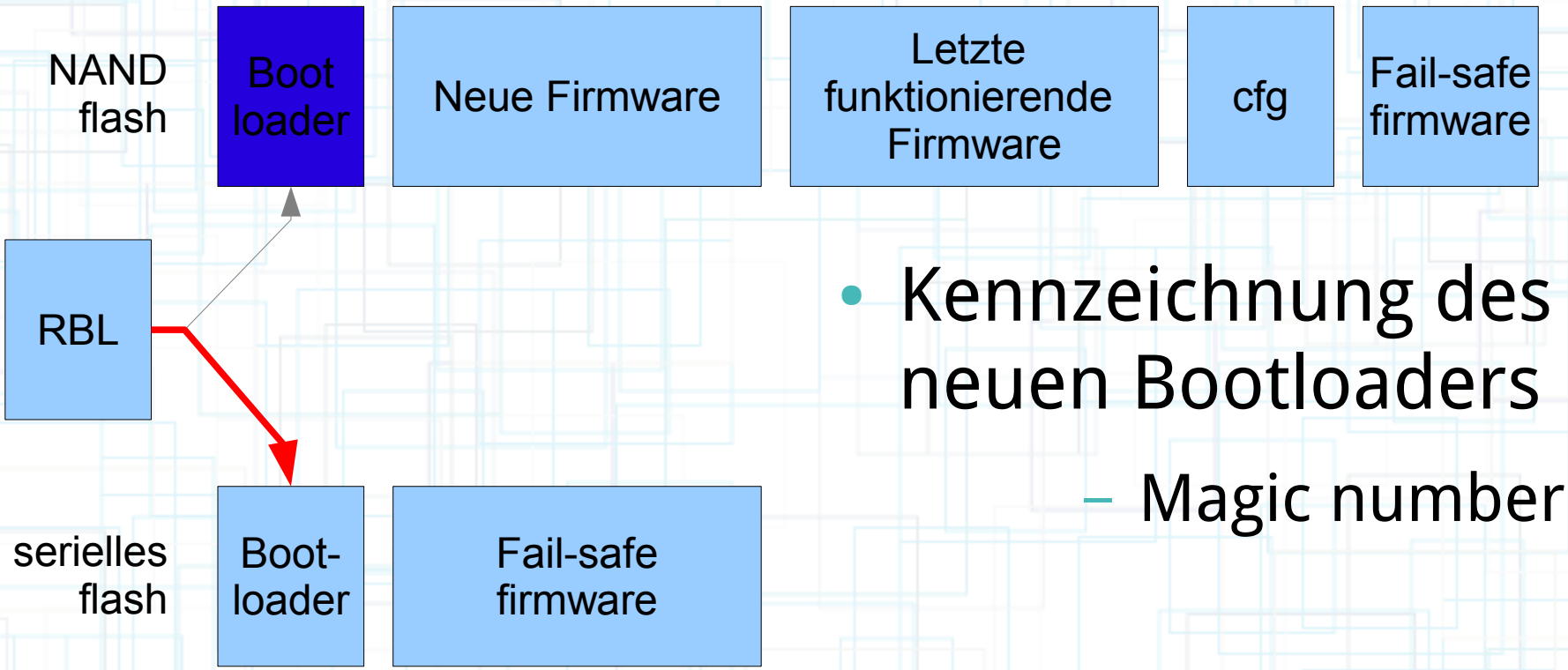
- Löschen des Bootloaders
  - RBL schaltet automatisch um

Ablauf 2



- Schreiben des neuen Bootloaders
  - Im Fehlerfall Start der Alternative!

Ablauf 3



- Kennzeichnung des neuen Bootloaders
  - Magic number

Alternativen

- Update muss nicht immer Austausch ganzer Partitionen bedeuten
  - Jedenfalls bei HLOS
- Funktionsupdates sind häufig auf einzelne Pakete begrenzt
  - Userspace-Software bei HLOS
- Paketmanagement kann u.U. Ausreichen
  - Update einzelner Pakete

Update-Durchführung

- Ein Update kann unterschiedlich durchgeführt werden
  - Low-level-Software ähnlich Bootloader
  - Mit Unterstützung durch ein HLOS
- Unterschied liegt im Implementierungsaufwand

Update in Low-level-Software

- Geringe Größe der erforderlichen Software
- Hoher Eigenanteil an der Implementierung
  - Gerätetreiber und Gerätezugriff
  - Sicherungsmechanismen
    - ECC, *Wear Leveling*
- Geschwindigkeitsvorteil

Update in HLOS

- Nutzung vorhandener Gerätetreiber für den Gerätezugriff
- Verwendung abstrakter Konstrukte
  - Partitionen
  - Dateisysteme
- Sicherungsmechanismen oft schon vorhanden

Vergleich Updatemechanismen

	Software-Unterstützung	Software-Größe
low-level	u.U. selbst zu erstellen	klein
HLOS	vorhanden	umfangreich

Klassischer Fall: Nichts ist optimal...