

Eingebettete Betriebssysteme

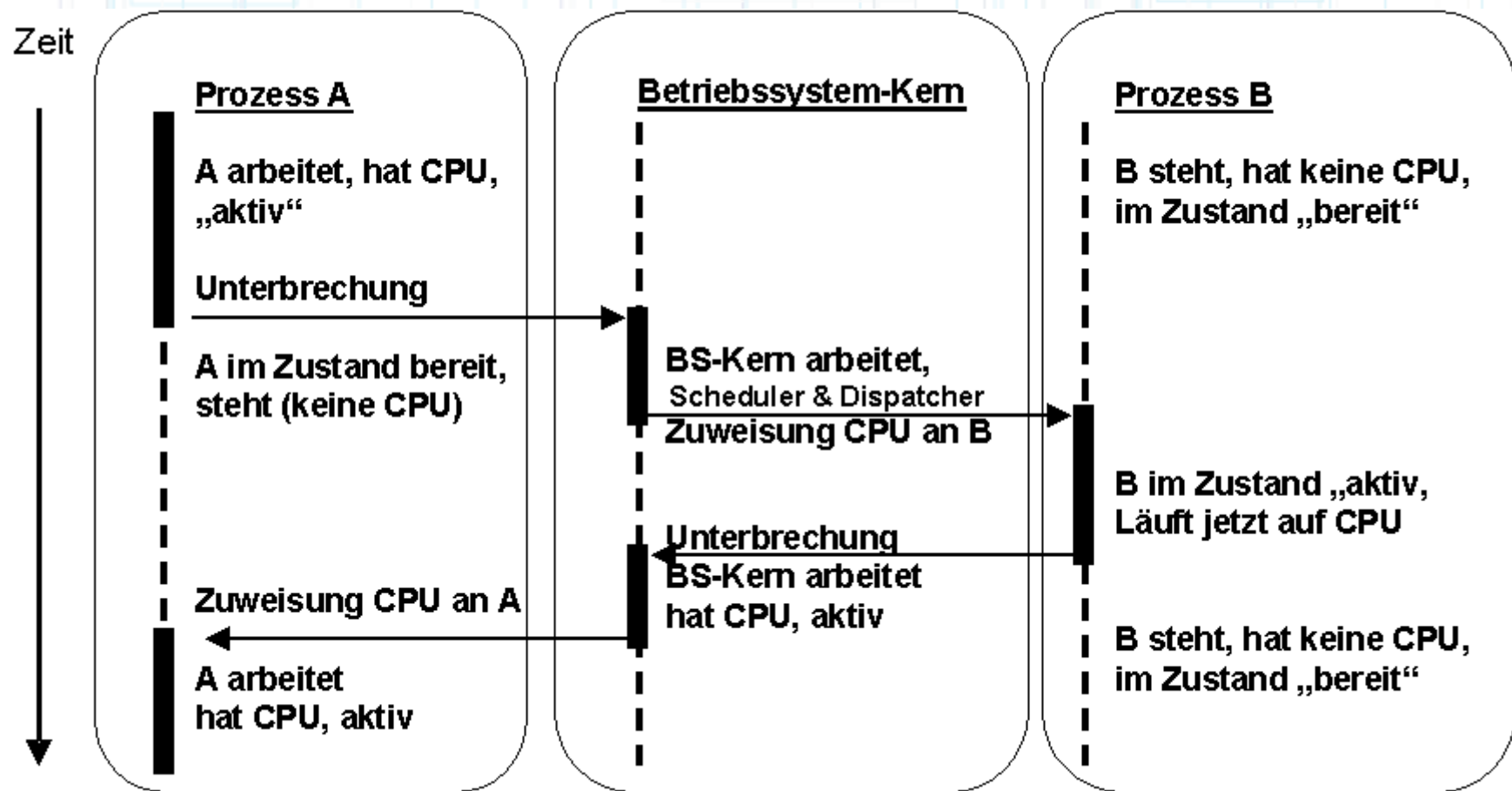
Bachelorstudiengang
Technische Informatik
Hochschule Pforzheim

Dipl.-Ing.(FH) Marc Jüttner

Scheduling

- Strategische Planung der Prozessreihenfolge
 - Problem: Konkurrieren um Ressourcen
 - Häufig nur im Bezug auf CPU-Nutzung
- Planender Teil des Betriebssystems ist der *Scheduler*
- Ausführer Teil des Betriebssystems ist der *Dispatcher*
- Quasiparallele Prozessausführung

Scheduling



Scheduling-Anforderungen

- Fairness
- Effizienz
- Antwortzeit
- Verweilzeit, Jobdauer
- Durchsatz: Zahl fertiger Jobs pro Zeiteinheit
- garantierte Reaktionszeit
- Ziele sind widersprüchlich!

Fairness

- Jeder Job erhält einen gerechten Teil der CPU-Zeit
 - was ist gerecht?
 - Prioritäten müssen berücksichtigt werden
 - Oft nicht hinreichend fair
- *non-preemptive*: Prozess kann nicht unterbrochen werden
- *preemptive*: Aktiver Prozess ist unterbrechbar

Effizienz

- Prozessor ist kostbares Betriebsmittel
- CPU-/Systemauslastung soll möglichst hoch sein
- Auslastung variiert zwischen 40% und 90%

Antwortzeit

- Bei Benutzerinteraktion
 - Zeitspanne zwischen Eingabe und Reaktion
 - bei eingebetteten Systemen besonders wichtig!
- Häufig nicht Minimierung der Antwortzeiten, sondern Minimierung der Varianz
 - gleichmäßiges Antwortverhalten

Verweilzeit

- Zeit von der Eingabe eines Auftrags bis zur Fertigstellung
- Scheduling hat wesentlichen Einfluss auf die Verweilzeit
- CPU-Scheduling kann nur Wartezeiten im „bereit“-Zustand beeinflussen
- Prozess muss möglichst effizient und vorausschauend auf Ressourcen zugreifen

Durchsatz

- Zahl fertiggestellter Jobs pro Zeiteinheit
- möglichst maximaler Durchsatz
- ein Job kann die Ausführung mehrerer Jobs bewirken
 - Kompilieren und Ausführen eines Programms
 - Beispiel: mehrere make-Instanzen parallel

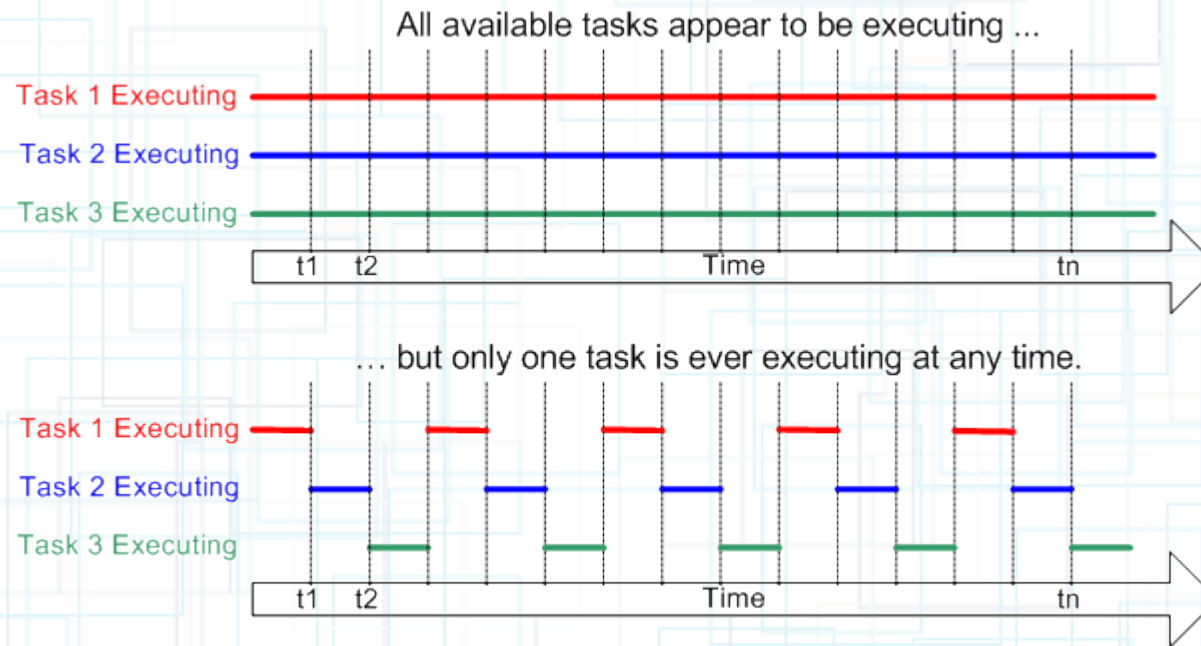
Garantierte Reaktionszeit

- bei Echtzeitsystemen
- Zeitspanne, innerhalb derer ein Rechnersystem auf Ereignisse reagieren muss
- muss garantiert sein
- abhängig von technischen Gegebenheiten und Problemstellung

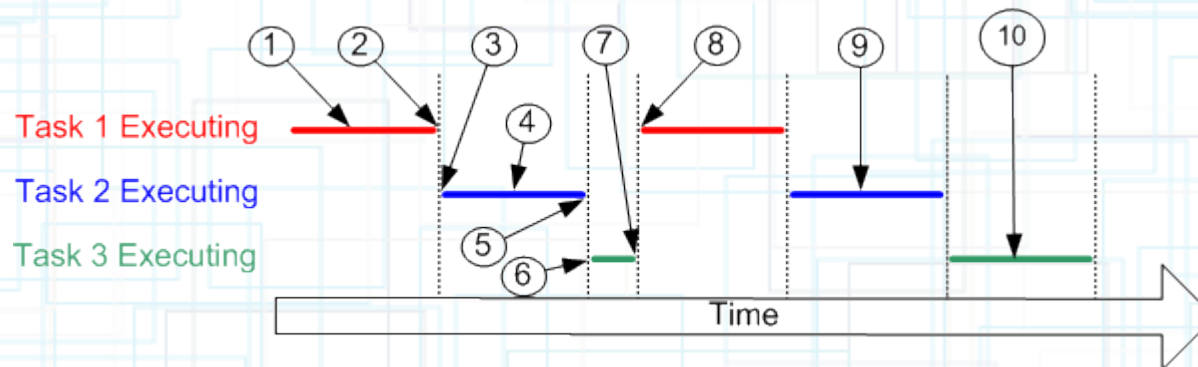
Problemstellungen

- Verhungern niedrig priorisierter Tasks möglich
 - *Process/task starvation*
- Läuft ein Task zu lang, kann das System „aus dem Takt kommen“
- Die „Kunst“ liegt im Systementwurf!

Multitasking



Multitasking mit E/A



- (1) Task 1 läuft
- (2) Task 1 wird suspendiert
- (3) Task 2 läuft
- (4) Task 2 blockiert Ressource
- (5) Task 2 wird suspendiert
- (6) Task 3 läuft
- (7) Task 3 geht schlafen
- (8) Task 1 läuft
- (9) Task 2 gibt Ressource frei
- (10) Task 3 läuft

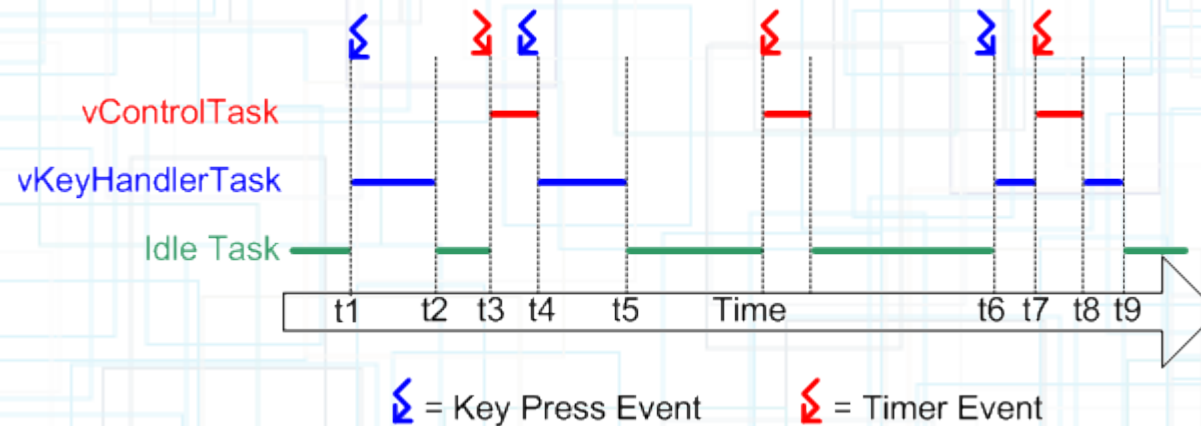
RTOS-Scheduling

- Echtzeitanforderung
- Prioritätenbasiertes Scheduling
 - *Priority levels*
 - Höhere Prioritäten laufen vorrangig
- Häufig Round Robin pro Ebene
- Tasks sind im Softwaredesign fest definiert
 - Keine dynamischen Einflüsse

RTOS-Scheduling II

- Scheduling bei „normalen“ Betriebssystemen ist zeitbasiert
 - Keine Echtzeitfähigkeit gegeben!
- RTOS: Ereignisgesteuert
 - Interrupts, Tasks, vergleichbar mit IRQ
 - Synchronisationsmechanismen steuern Ablauf
 - Partitionierung bzw. *Profiling* beim Design erforderlich

RTOS-Multitasking



- Kein Task läuft ohne Auslöser
 - *Idle-Task*
- Tasks laufen ereignisgesteuert
 - Maximal mögliche Laufzeit vorgegeben

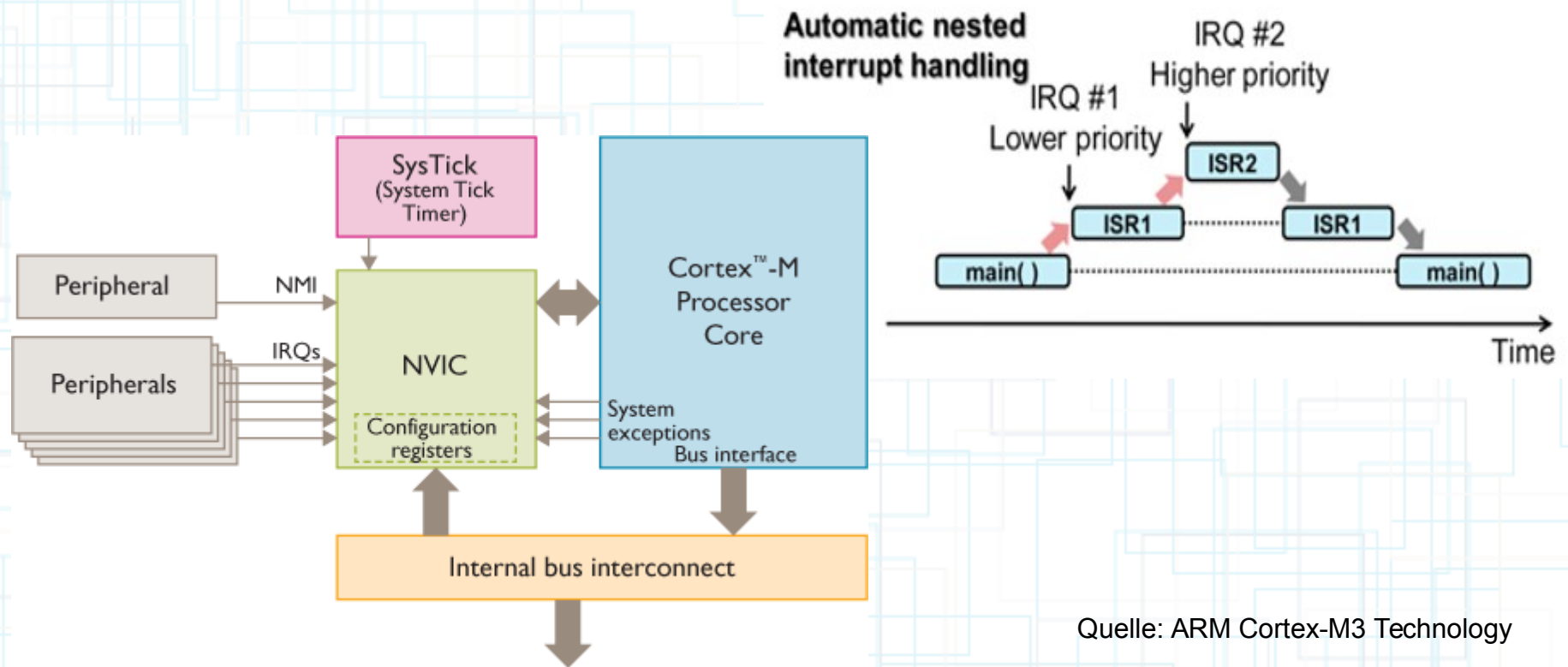
Quelle: <http://www.freertos.org/implementation/a00008.html>

Alternativen

- Kernelcode kann nicht unterbrochen werden
 - Auch Kernelcode im Prozesskontext
- *Kernel preemption*
 - Unterbrechung im Kernelcode möglich
 - Abhängig von der Prozesspriorität
 - Erhöht das Antwortverhalten
 - Dennoch nicht immer echtzeitfähig!

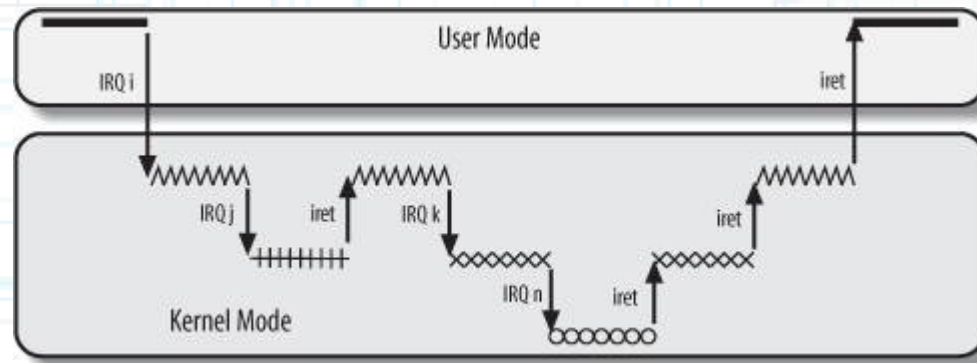
Interruptprioritäten

- Priorisierung von Interrupts



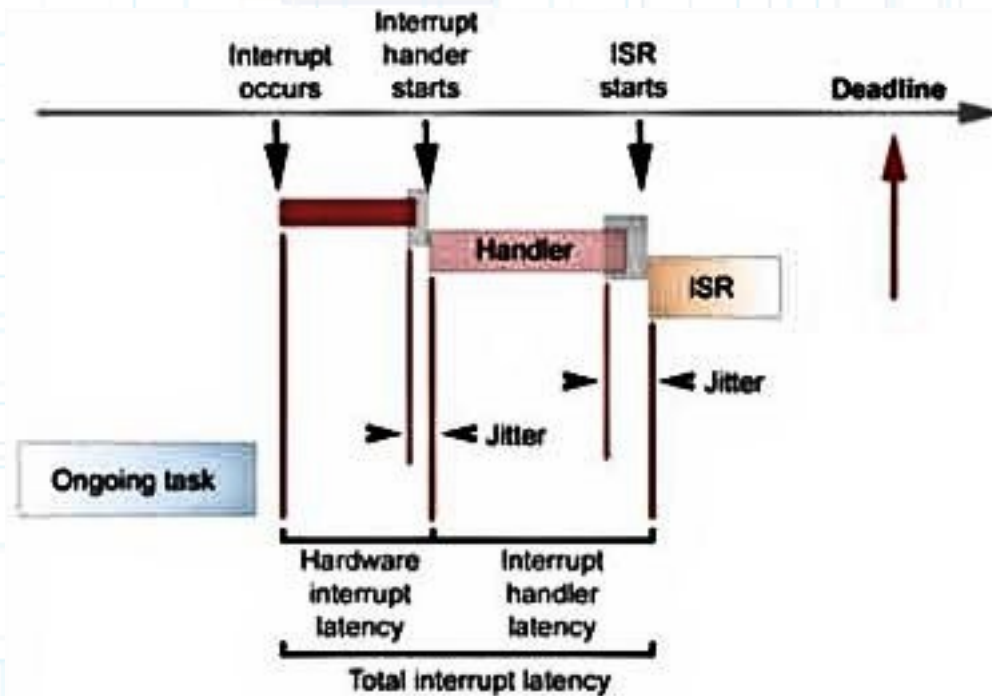
Interrupt Nesting

- Was passiert bei gleichpriorisierten Interrupts?
- Unter Linux:



Quelle: Understanding the Linux Kernel, Kapitel 4.3

Interrupt Jitter



Quelle: embedded.com – Managing intelligent I/O processing on DSP & GPP SoC

Interrupt-Latenz

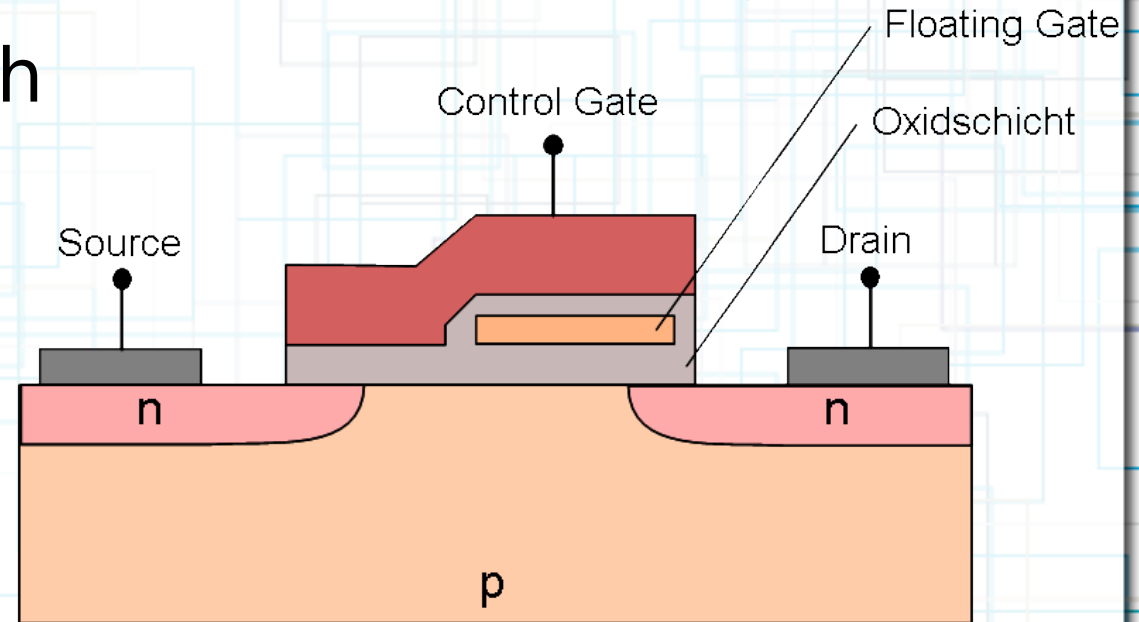
- Hardware-Latenz
 - Bei zeitweise deaktivierten Interrupts
- Handler-Latenz
 - Variable Handler-Laufzeit: Jitter
 - Cache-Einflüsse
 - Interne Bus-Arbitrierung
 - Externer Speicher...

Flash-Speicher

- 1980 erfunden
- Basiert auf EPROM-Technologie
 - Erasable Programmable Read Only Memory
 - Damals: Löschen durch UV-Licht
 - Später: EEPROM
- Speicherung durch persistente Ladungen in einem *Floating Gate*

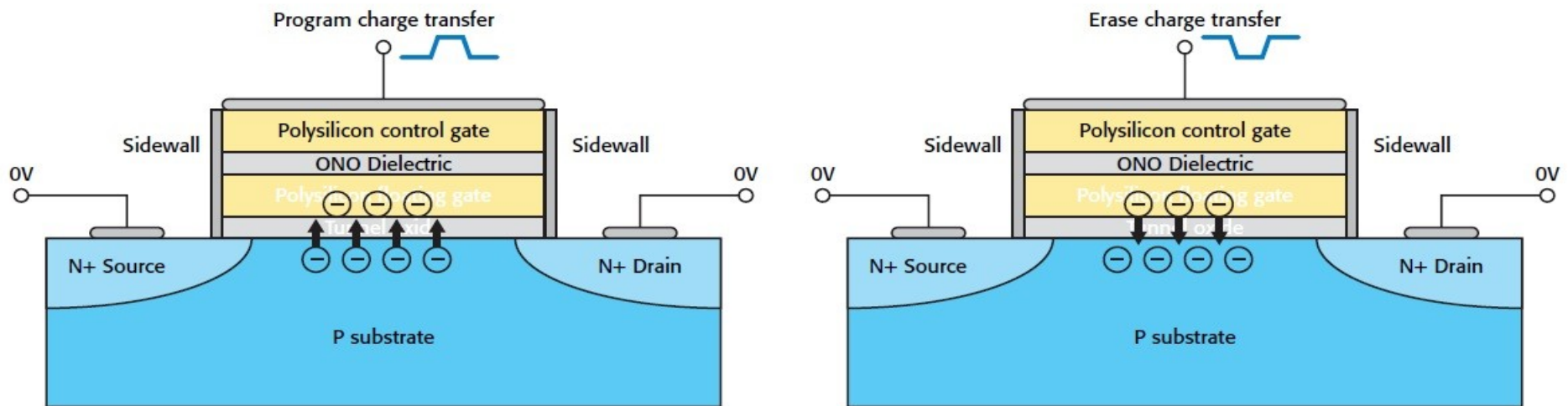
Flash-Technologien

- Zwei Ausprägungen
 - Abhängig von der Verschaltung
 - NAND-Flash
 - NOR-Flash



Quelle: <http://blog.bgaechter.ch>

Programmieren/Löschen



Quelle: <http://www.eetimes.com/design/memory-design/4403035/Memory-Test-Tips--3--Improve-flash-memory-testing-with-pulse-generators>

Bit-Kapazität

- SLC: Single Level Cell
 - Ein Bit pro Zelle
- MLC: Multi Level Cell
 - Mehrere Bit pro Zelle
 - 2x, 4x, 8x verfügbar
 - Fehleranfällig!

Flash-Besonderheiten

- Blockweises Löschen
 - Nur ganze Blöcke auf einmal löschar
 - Trotz byteweisem Lesen/Schreiben!
- Softwarebehandlung erforderlich
 - Auslesen des ganzen Blocks
 - Löschen
 - Erneutes Schreiben

Flash-Nachteile

- Alterungserscheinungen
 - Nach 100.000 Löschzyklen entstehen Zellendefekte
 - Mechanische Abnutzung im Oxid unter dem *Floating Gate*
- Maximale Zahl nur garantiert für Block 0
- Seiteneffekte durch Lesezugriffe
 - Bitfehler beim Lesen

Flash-Vorteile

- Preis pro MiB sehr niedrig bei NAND
- Lange Speicherung möglich
- Blockzugriffe bei NAND perfekt für Festplattensimulation
 - Blockzugriffe auf Massenspeicher
- Einfacher elektrischer Anschluss
 - Nur mäßig anspruchsvoll

Vergleich RAM/Flash

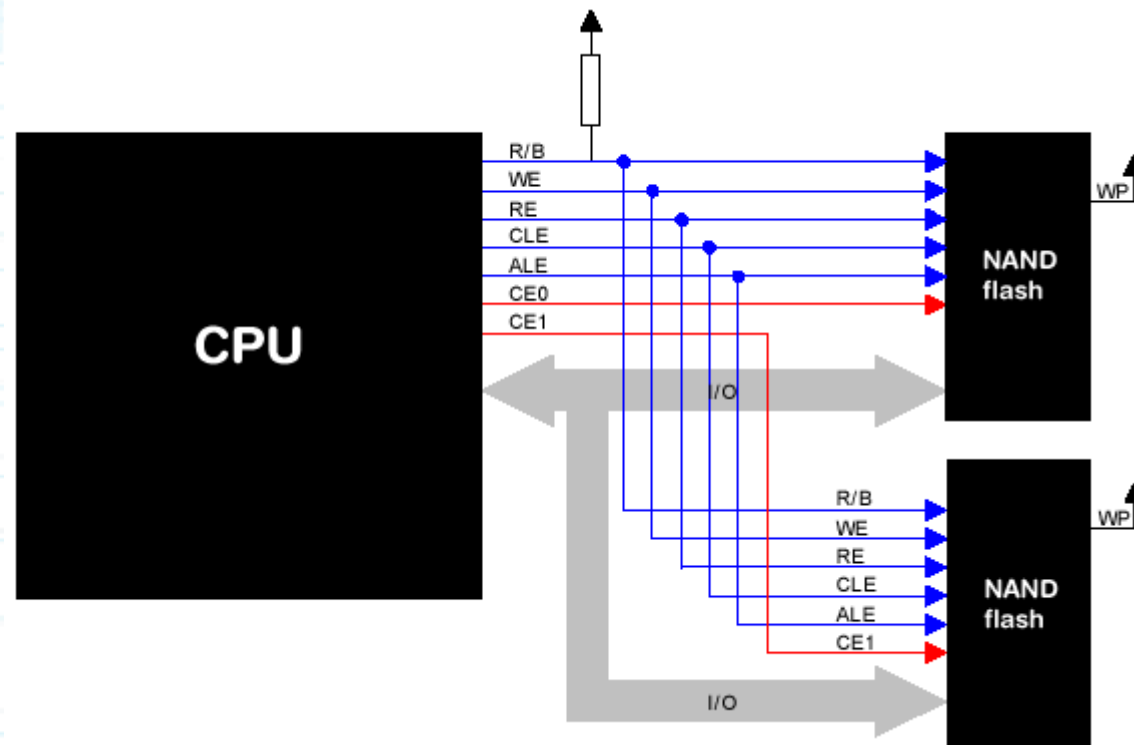
		RAM	NAND-Flash	NOR-Flash
Zugriff	lesen	+++	-	---
	schreiben	+++	+ (1)	---
Daten	lesen	Byte	Seite	Byte
	schreiben	Byte	Seite	Byte/Wort
	löschen	Byte	Block	Block
Persistent		nein	ja	ja
Linear adressierbar		ja	nein	ja
Alterungserscheinungen		nein	ja	ja

(1): bei großen Datenmengen

NAND-Zugriff

- Kombiniertes Adress- und Datenbus
- Kommandobasiert
 - read page
 - ...
- Spezieller *NAND-Controller* erforderlich
 - Meist in SoC integriert

NAND: Anbindung



Quelle: http://www.segger.com/emfile_driver_nand_flash.html

NOR-Zugriff

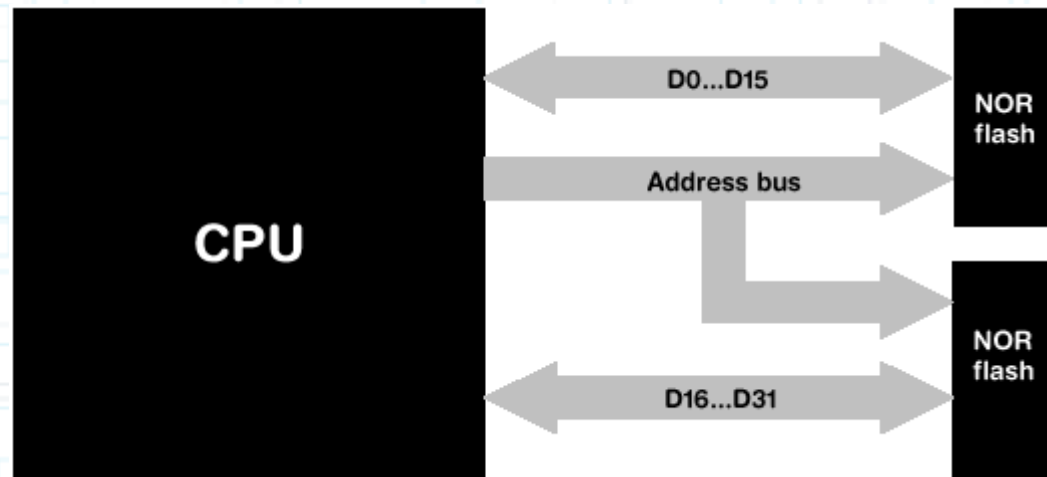
- Separater Adress- und Datenbus
- Zugriff wie auf SDRAM
- Kein Controller erforderlich

NOR: Anbindung 16bit



Quelle: http://www.segger.com/emfile_driver_nor_flash.html

NOR: Anbindung 32bit

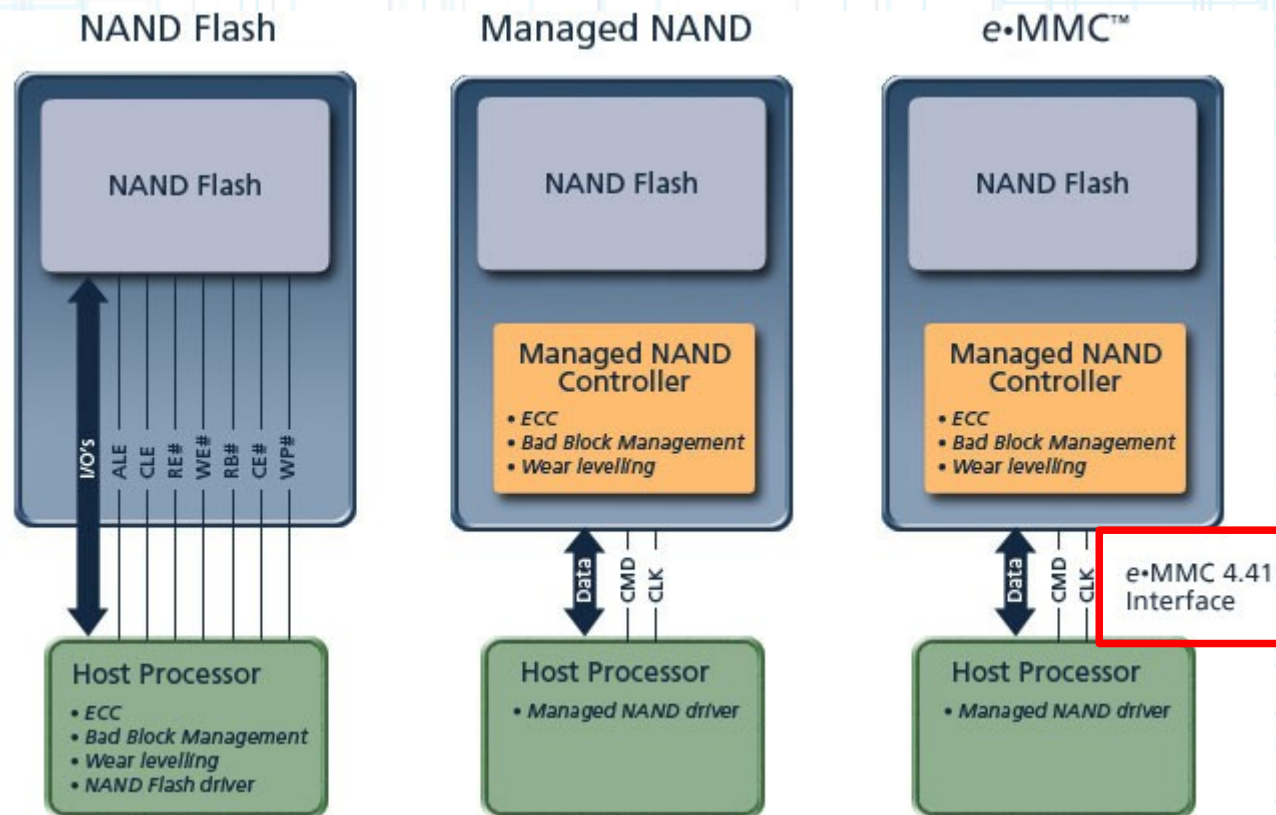


Quelle: http://www.segger.com/emfile_driver_nor_flash.html

NOR: Mögliche Probleme

- Löschen blockiert u.U. Laden aus anderen Seiten
 - Problematisch, wenn Programmcode geladen werden soll
- Empfohlen ist, Programmcode für den Lese-/Schreibzugriff im RAM zu halten

Neuere Technologien

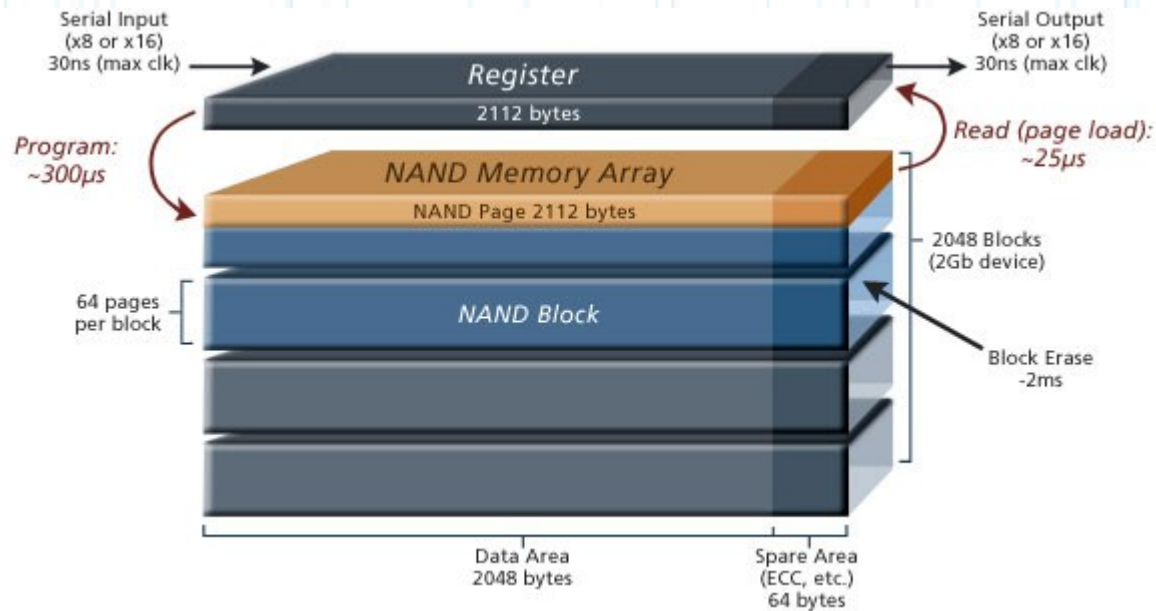


Quelle: <http://www.micron.com/products/nand-flash/choosing-the-right-nand>
<http://www.micron.com/products/managed-nand/e-mmc>

NAND-Datenorganisation

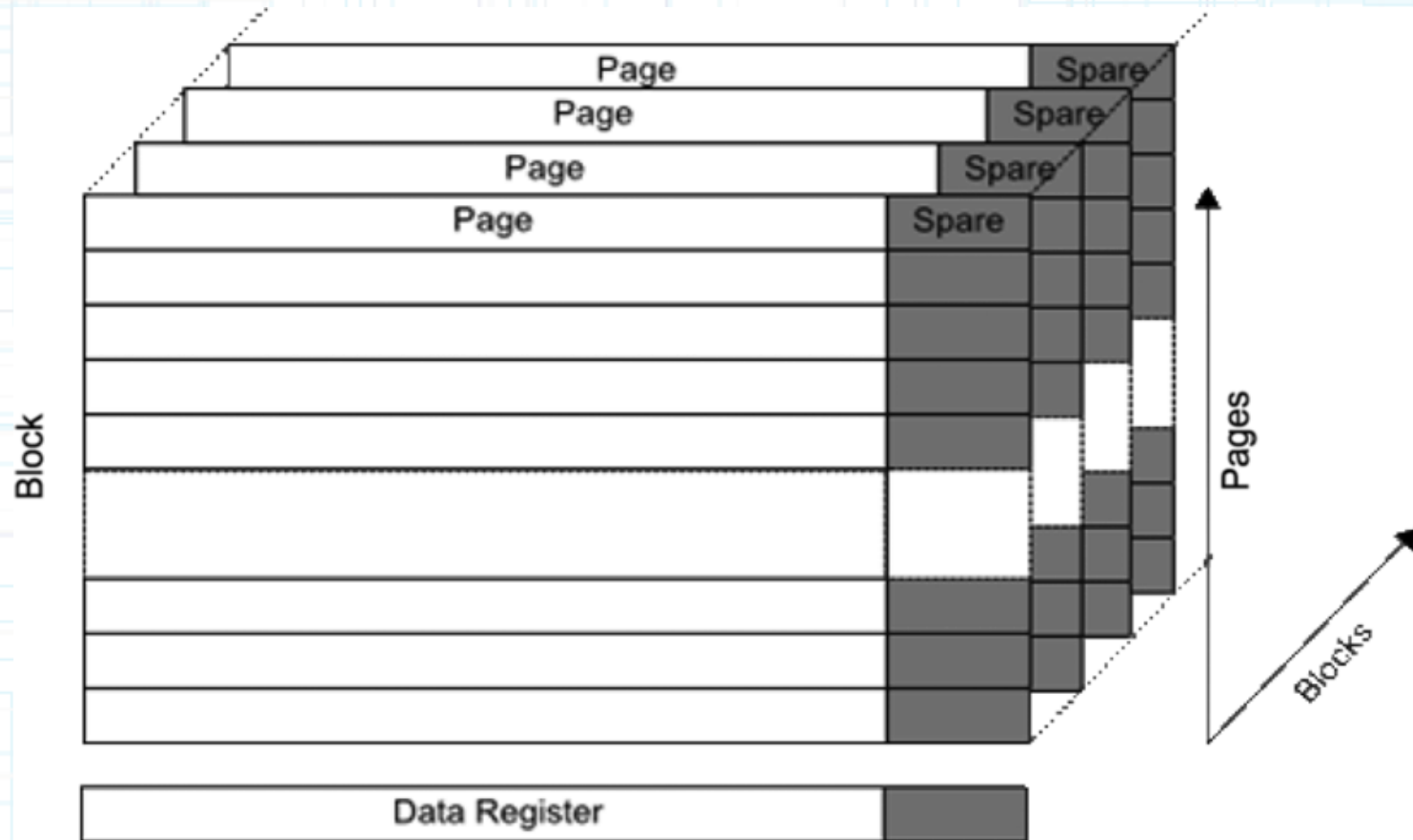
- Seitenstruktur
 - Mehrere Bits pro Seite
- Blockstruktur
 - Mehrere Seiten pro Block
- Gesamtstruktur
 - Mehrere Blöcke pro Chip

NAND: Datenorganisation



Quelle: <http://www.micron.com/products/nand-flash/choosing-the-right-nand>

NAND: Datenorganisation



Quelle: http://ieeexplore.ieee.org/ieee_pilot/articles/96jproc11/jproc-MSanvido-2004319/article.html





NAND-Größen, Beispiel

- 2048 Bytes pro Seite
 - Zzgl. 64 Bytes OOB-Daten
- 64 Seiten pro Block
 - 128kiB pro Block
- 1024 Blöcke pro Chip
 - 128MiB Gesamtkapazität
 - 4MiB OOB-Daten

OOB-Daten

U-boot/Linux standard OOB layout





0							7
8							15
16							23
24							31
32							39
40							47
48							55
56							63


 ECC data
  other OOB data
  Factory BBM

ECC data byte: 10 bytes per 512 bytes of page data

TI RBL/UBL standard OOB layout

0							7
8							15
16							23
24							31
32							39
40							47
48							55
56							63


 ECC data
  other OOB data
  Factory BBM

ECC data byte: 10 bytes per 512 bytes of page data

nach: [U-Boot] Layout of OOB data in NAND flash, DENX-Mailingliste

Betriebssystemeffekte

- Viele Nachteile können durch OS-Maßnahmen umgangen werden
- Softwareaufwand entsteht dennoch
 - Implementierung von Treibern und Protokollen
 - Test und Validierung
 - Zertifizierung...

Wear Levelling

- Gleichmäßige Abnutzung von Flashzellen
- Bei jedem Schreibvorgang werden die Daten in einen anderen Block geschrieben
- Erfordert Softwarebehandlung
 - Nicht jedes Dateisystem sinnvoll
 - Ausnahme: USB-Massenspeicher haben WLC integriert

Löschen und Caches

- Blockweise Löschoperationen
 - erfordern vorheriges Lesen des ganzen Blocks
- Vorhalten von Daten im Speicher reduziert Löscho- und Schreibzyklen
- Gefahr bei Spannungsverlust
 - USB-Sticks
 - *Hard resets* ohne Herunterfahren

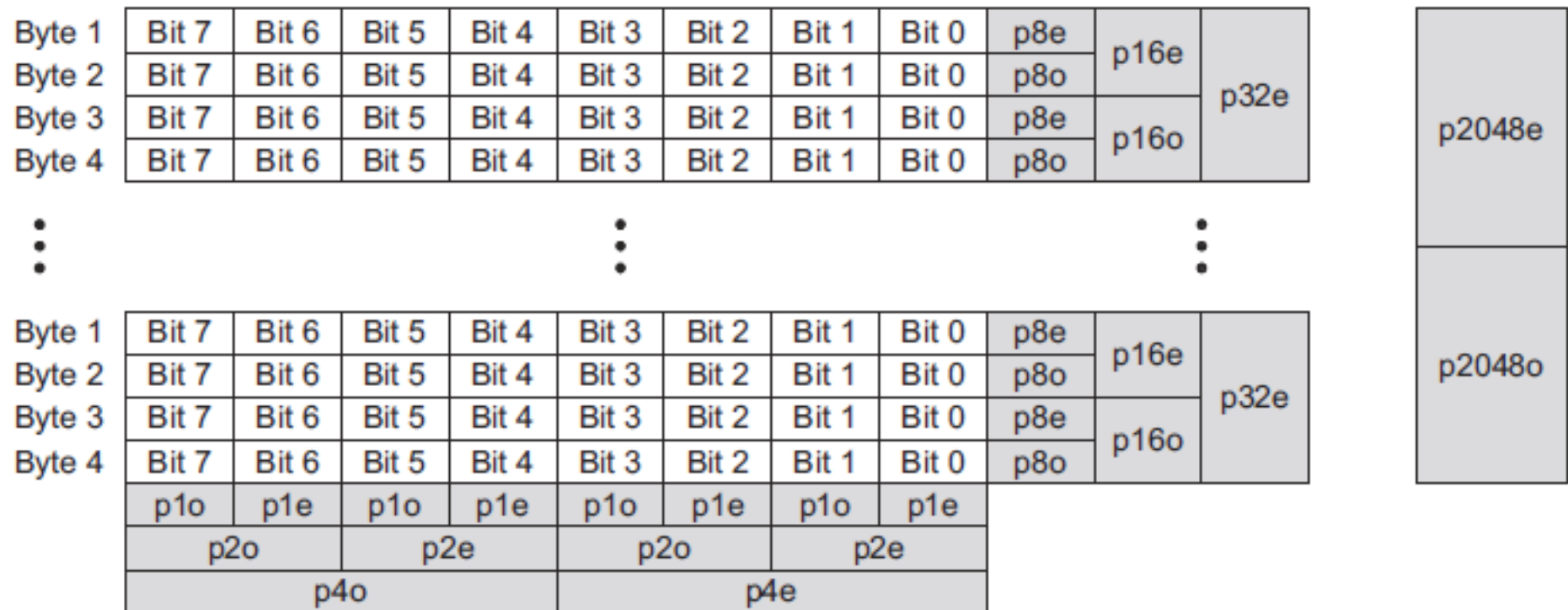
Bitfehlerkorrektur

- Bitfehler möglich
 - durch defekte Blöcke
 - Durch Seiteneffekte beim Lesen
- Korrektur erforderlich
 - Möglich bis zu bestimmter Zahl von Fehlern
 - ECC: *Error Checking and Correction*
- Oft hardware-unterstützt

Bitfehlerkorrektur

- ECC: Fehlererkennung und/oder Korrektur
 - Mindestabstand zwischen fehlerhaften Bytes erforderlich
- Über Zeilen und Spalten der Datenbits werden gerade und ungerade Paritäten berechnet
- Prüfsumme und Daten werden auf Konsistenz geprüft

1 bit ECC

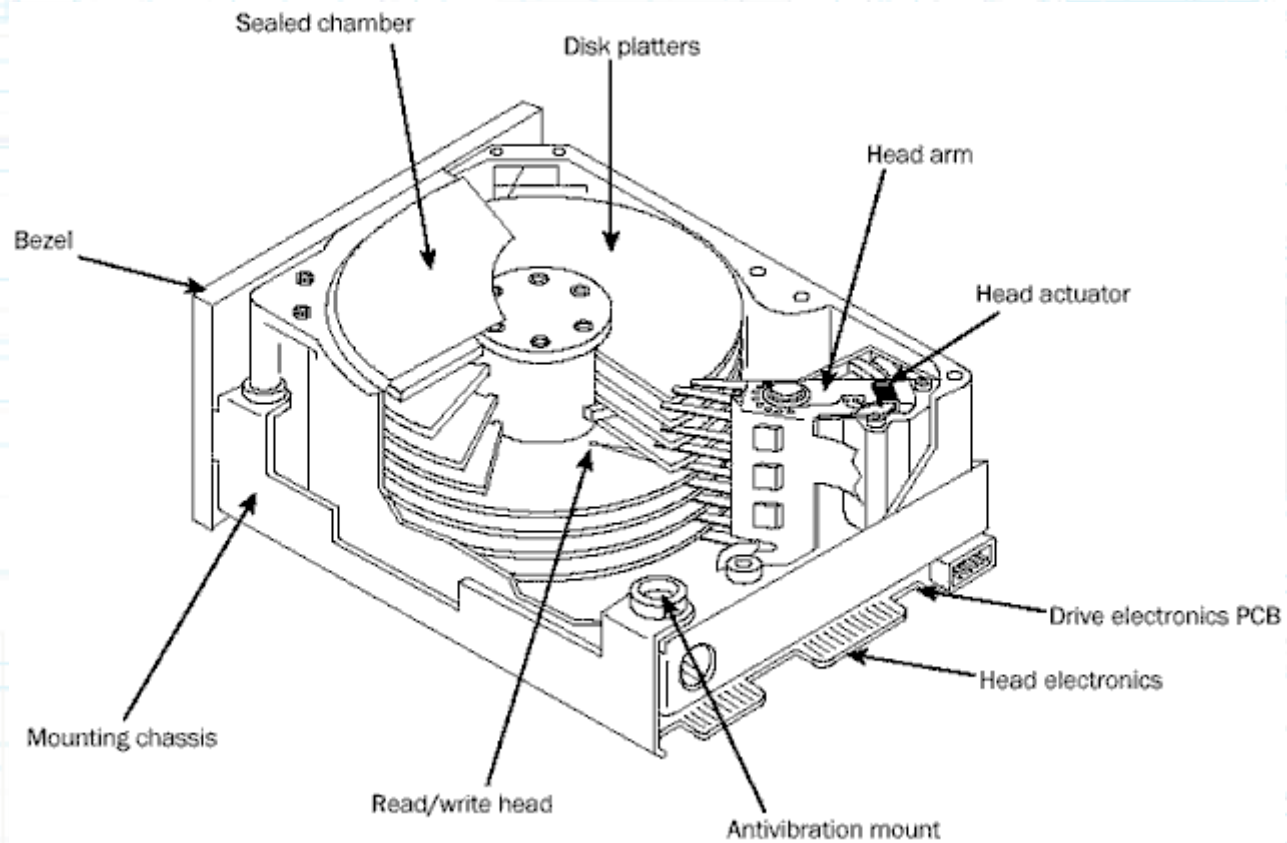


Quelle: TI DM36x AEMIF Data Manual, Figure 9

Dateisysteme

- Organisation von Dateien auf einem Datenträger
- Datenträgerverwaltung
 - Sektoren, Zylinder, Köpfe
- Abstraktion in Blöcke
- Nicht auf Festplatten beschränkt

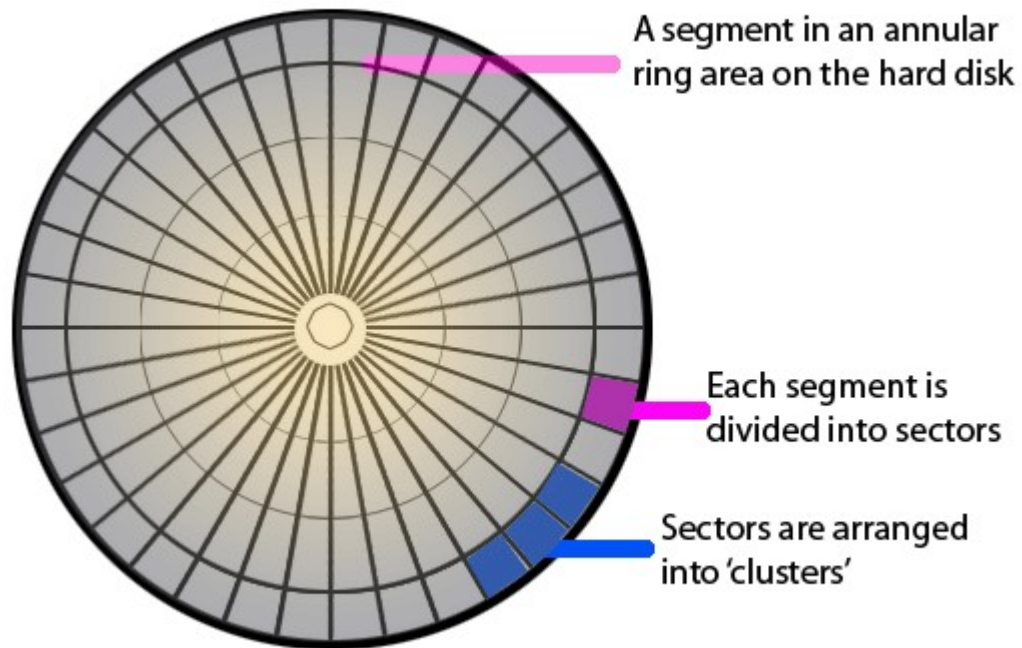
Zylinder, Köpfe, ...



Quelle: alasir.com

Spuren, Sektoren...

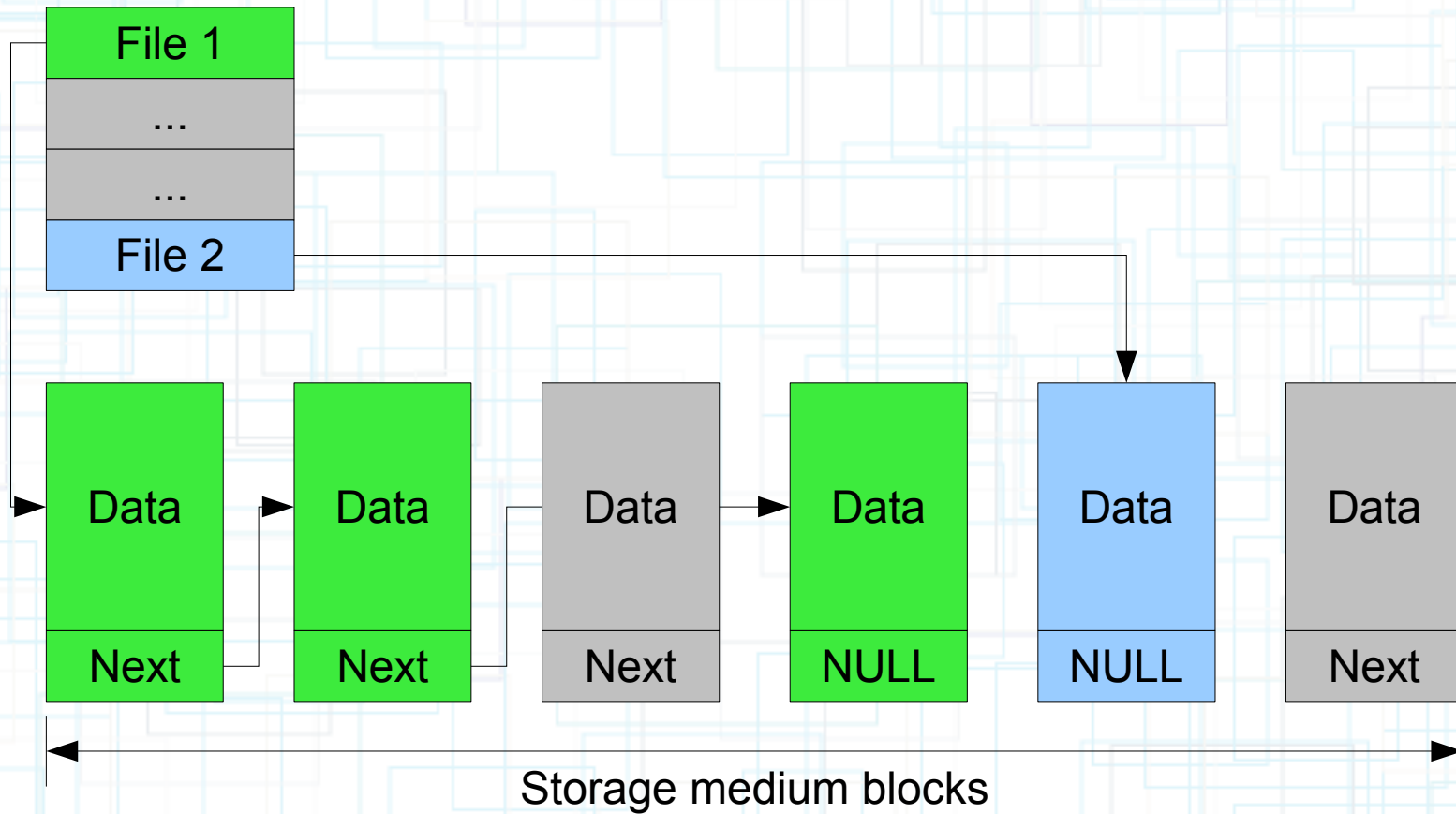
Hard disk format



(c) www.teach-ict.com

Quelle: cheadledatarecovery.com

Dateisysteme



Übersicht

	romfs	cramfs	ext2	ext3	yaffs2	jffs2	squashfs
rw/ro	ro	ro	rw	rw	rw	rw	ro
Kompression	nein	ja	nein	nein			ja
Wear leveling	-	-	nein	nein	ja	ja	-
Power-safe	-	-	-	ja	ja	ja	-
Benutzer-rechte	Nur root	Nur root	ja	ja	ja	ja	Nur root
Dateianzahl	-	2^{16}	10^{18}	10^{18}	2^{32}	2^{32}	2^{32}
Größe	-	256 MiB	2 TiB	2 TiB	2^{32} GiB	2^{32} GiB	2^{32} GiB

Problemstellung

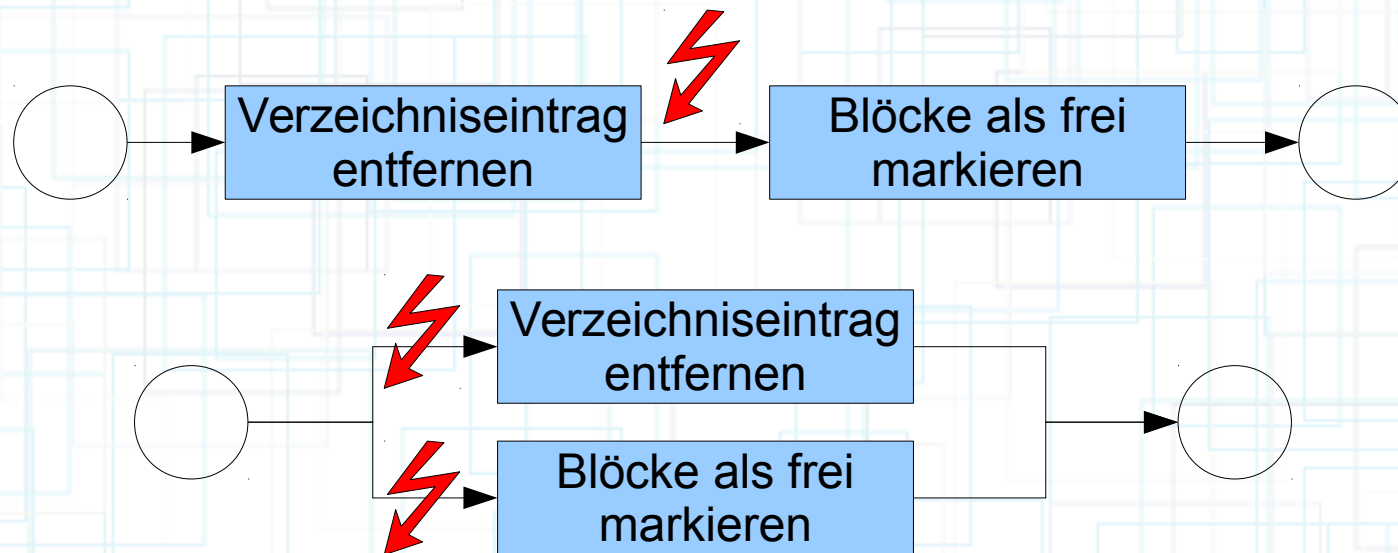
- Unterbrechung von Schreiboperationen können zu Inkonsistenz führen
 - Verwaiste Blöcke
 - Inkonsistente Zuordnungstabellen
- Verschiedene Folgen
 - Überschreiben gültiger Daten
 - Verlust freier Blöcke (storage leak)

Journalle

- Aufzeichnen geplanter und erledigter Vorgänge
- Bei Unterbrechnung kann die Operation fertiggestellt werden
- Abwägung von Vor- und Nachteilen
 - Doppelte Zahl von Operationen pro Block
 - Sicherheit vs. Geschwindigkeit

Löschoperation

- Löschen einer Datei in zwei Schritten
 - Verzeichniseintrag entfernen
 - Blöcke als frei markieren



Identifikation

- Dateisystem-ID einer Partition
- Hartkodiertes Wissen über Dateisystem
- Datenträger und Partitionen beim Systemstart bekannt
 - Wechseldatenträger?