

## 2. Foliensatz

# Betriebssysteme und Rechnernetze

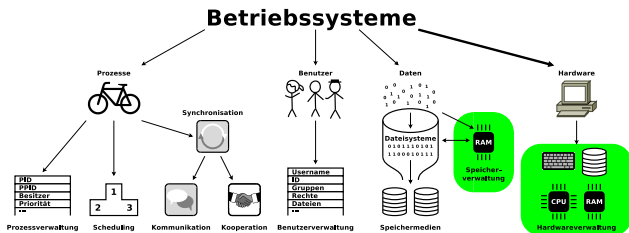
Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences  
(1971–2014: Fachhochschule Frankfurt am Main)  
Fachbereich Informatik und Ingenieurwissenschaften  
[christianbaun@fb2.fra-uas.de](mailto:christianbaun@fb2.fra-uas.de)

# Lernziele dieses Foliensatzes

- Am Ende dieses Foliensatzes kennen/verstehen Sie...
  - was für **Speicher** in Computern existiert
    - Architektur und Arbeitsweise der **Speicherhierarchie** (Speicherpyramide)
    - Arbeitsweise der **Cache-Schreibstrategien** (Write-Back/Write-Through)
  - grundlegende Konzepte der **Speicherverwaltung**
    - **Statische/Dynamische Partitionierung, Buddy-Speicherverwaltung**
  - wie Betriebssysteme auf den **Speicher zugreifen** (ihn adressieren!)
    - **Real Mode, Protected Mode**
  - **Paging**-Konzept zur Realisierung des **virtuellen Speichers**
  - die möglichen Ergebnisse bei Anfragen an einen Speicher (**Hit und Miss**)
  - die Arbeitsweise und Eckdaten wichtiger **Ersetzungsstrategien**

Übungsblatt 2 wiederholt die für die Lernziele relevanten Inhalte dieses Foliensatzes



# Speicher

Bildquelle: United States Department of Energy (Public Domain)

- Idee und Aufbau des Universalrechners, der nicht an ein festes Programm gebunden ist und über Ein-/Ausgabegeräte verfügt
  - Entwickelt 1946 von John von Neumann
  - Nach ihm benannt ist die **Von-Neumann-Architektur**, bzw. der **Von-Neumann-Rechner**
- Im Von-Neumann-Rechner werden Daten und Programme **binär kodiert** und liegen im **gleichen Speicher**
- Unterschiedliche Speicher sind durch Busse verbunden und bilden eine Hierarchie
  - ⇒ **Speicherpyramide** (siehe Folie 5)
- Grund für die Speicher-Hierarchie: Preis/Leistungsverhältnis
  - ⇒ Je schneller ein Speicher ist, desto teurer und knapper ist er



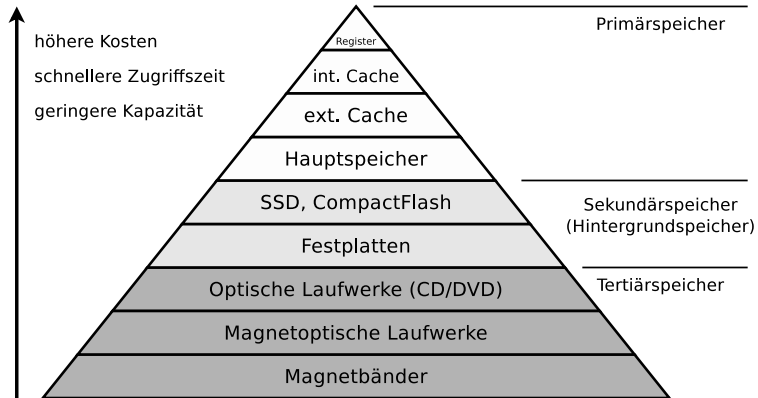
# Digitale Datenspeicher

Speicher	Speicherung	Lesevorgang	Zugriffsart	Bewegliche Teile	Persistent
Lochstreifen	mechanisch		sequentiell	ja	ja
Lochkarte	mechanisch		sequentiell	ja	ja
Magnetband	magnetisch		sequentiell	ja	ja
Magnetkarte / Magnetstreifen	magnetisch		sequentiell	ja	ja
Trommelspeicher (Drum Memory)	magnetisch		wahlfrei	ja	ja
Kernspeicher	magnetisch		wahlfrei	nein	ja
Magnetblasenspeicher (Bubble Memory)	magnetisch		wahlfrei	nein	ja
Cache und Register (SRAM)	elektronisch		wahlfrei	nein	nein
Hauptspeicher (DRAM)	elektronisch		wahlfrei	nein	nein
Flashspeicher (USB-Stick, SSD, CF/SD-Karte)	elektronisch		wahlfrei	nein	ja
Compact Cassette (Datasette)	magnetisch		sequentiell	ja	ja
Diskette (Floppy Disk)	magnetisch		wahlfrei	ja	ja
Festplatte (Hard Disk)	magnetisch		wahlfrei	ja	ja
CD-ROM/DVD-ROM	mechanisch	optisch	wahlfrei	ja	ja
CD-R/CD-RW/DVD-R/DVD-RW		optisch	wahlfrei	ja	ja
MiniDisc	magneto-optisch	optisch	wahlfrei	ja	ja
Magneto Optical Disc (MO-Disk)	magneto-optisch	optisch	wahlfrei	ja	ja

(graue Hintergrundfarbe bedeutet: veraltete bzw. überholte Technologie)

- **Wahlfreier Zugriff** heißt, dass das Medium nicht – wie z.B. bei Bandlaufwerken – von Beginn an sequentiell durchsucht werden muss, um eine bestimmte Stelle (Datei) zu finden
  - Die Köpfe von Magnetplatten oder ein Laser können in einer bekannten maximalen Zeit zu jedem Punkt des Mediums springen

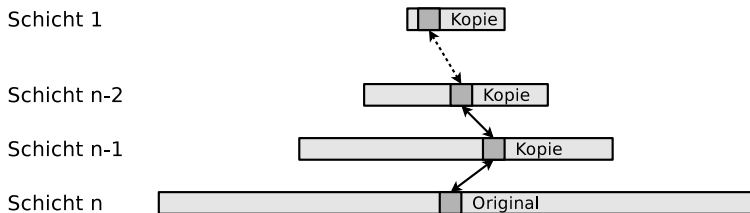
# Speicherpyramide



- **Primärspeicher:** Darauf greift die CPU direkt zu
- **Sekundärspeicher:** Wird über einen Controller angesprochen
- **Tertiärspeicher:** Nicht dauerhaft mit dem Rechner verbunden. Hauptaufgabe ist Archivierung

# Arbeitsweise der Speicherhierarchie

- Beim ersten Zugriff auf ein Datenelement, wird eine Kopie erzeugt, die entlang der Speicherhierarchie nach oben wandert



- Wird das Datenelement verändert, müssen die Änderungen irgendwann nach unten durchgereicht (zurückgeschrieben) werden
  - Beim Zurückschreiben, müssen die Kopien des Datenblocks auf allen Ebenen aktualisiert werden, um Inkonsistenzen zu vermeiden
  - Änderungen können nicht direkt auf die unterste Ebene (zum Original) durchgereicht werden!

# Cache-Schreibstrategien

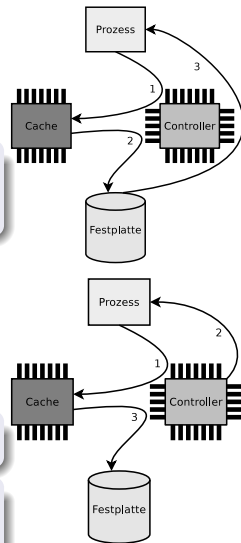
- **Write-Through:** Änderungen werden sofort an tiefere Speicherebenen weitergegeben
  - **Vorteil:** Konsistenz ist gesichert
  - **Nachteil:** Geringere Geschwindigkeit

Bild: Ein Prozess möchte eine Schreibweisung abarbeiten. Er schreibt (1) die Daten in den Cache und übergibt dem Controller die Schreibweisung. Der Controller weist (2) das Schreiben der Daten auf dem Datenspeicher an. Wurden die Daten erfolgreich geschrieben, meldet (3) der Controller das erfolgreiche Schreiben der Daten an den Prozess

- **Write-Back:** Änderungen werden beim Verdrängen der Seite aus dem Cache weitergegeben
  - **Vorteil:** Höhere Geschwindigkeit
  - **Nachteil:** Änderungen sind beim Systemausfall verloren

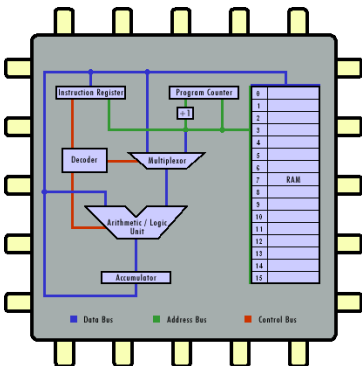
Für jede Seite im Cache wird ein **Dirty Bit** im Cache gespeichert, das angibt, ob die Seite geändert wurde

Bild: Ein Prozess möchte eine Schreibweisung abarbeiten. Er schreibt (1) die Daten in den Cache und übergibt dem Controller die Schreibweisung. Der Controller meldet (2) **sofort** das erfolgreiche Schreiben der Daten an den Prozess. Das Schreiben (3) der Daten auf dem Datenspeicher erfolgt asynchron zur Schreibweisung im Prozess



# Register, Cache und Hauptspeicher (1/4)

- **Register** enthalten die Daten, auf die die CPU sofort zugreifen kann
- Die Register sind genauso schnell getaktet wie die CPU selbst



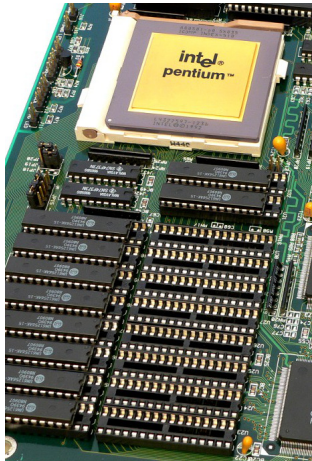
- **Datenregister** (= *Accumulatoren*) speichern Operanden für die ALU und deren Resultate
  - z.B. EAX, ECX, EDX, EBX (32-Bit)  
RAX, RBX, RCX, RDX (64-Bit)  
⇒ Foliensatz 4
- **Adressregister** für Speicheradressen von Operanden und Befehlen
  - z.B. **Basisadressregister** (= *Segmentregister*) und **Indexregister** (für den Offset) ⇒ Folie ??
- **Befehlszähler** (= *Program Counter*) (= *Instruction Pointer*) enthält die Speicheradresse des nächsten Befehls
- **Befehlsregister** (= *Instruction Register*) speichert den aktuellen Befehl
- **Stapelregister** (= *Stack Pointer*) enthält die Speicheradresse am Ende des Stack ⇒ Foliensatz 4

Bildquelle: [http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/CPU/cpu\\_circuit.gif](http://courses.cs.vt.edu/~csonline/MachineArchitecture/Lessons/CPU/cpu_circuit.gif)

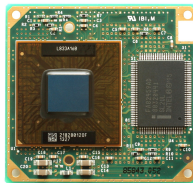


# Register, Cache und Hauptspeicher (2/4)

- **Cache** (Pufferspeicher) enthält Kopien von Teilen des Arbeitsspeichers um den Zugriff auf diese Daten zu beschleunigen



- **First Level Cache (L1-Cache)**
  - In die CPU integriert
- **Second Level Cache (L2-Cache)**
  - Langsamer und größer
  - Ursprünglich außerhalb der CPU

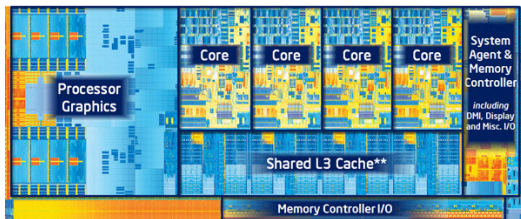


Bildquelle:  
Wikipedia (Konstantin Lanzet CC-BY-SA-3.0)  
Das Bild zeigt eine Intel Mobile Pentium II „Tongae“  
233 MHz CPU mit externem 512 kB L2-Cache. Der  
L2-Cache läuft mit halber Taktfrequenz

Bildquelle: <http://www.amoretro.de/2012/03/si5pi-aio-rev-1-1-socket-4-motherboard.html>  
Das Bild zeigt ein Elitegroup SI5PI AIO mit einem Pentium 60.  
Das Mainboard verfügt über 16 Sockel für Speichermodule für L2-Cache

# Register, Cache und Hauptspeicher (3/4)

- Seit 1999/2000 integrieren die CPU-Hersteller zunehmend den L2-Cache in die CPUs
  - Das führte zur Etablierung des **Third Level Cache** (L3-Cache) als CPU-externen Cache
- Bei modernen CPUs (z.B. Intel Core-i-Serie und AMD Phenom II) ist auch der L3-Cache in die CPU integriert
  - Bei Multicore-CPU mit integriertem L3-Cache teilen sich die Kerne den L3-Cache, während jeder Kern einen eigenen L1-Cache und L2-Cache hat



## Einige CPU-Architekturen haben einen L4-Cache

- Intel Itanium 2 (2003): 64 MB
- Wenige Intel Haswell CPUs - Core i7-4770R, Core i5-4670R, Core i5-4570R (2013): 128 MB
- Wenige Intel Broadwell CPUs - Core i7-5775C, Core i5-5675C (2015): 128 MB
- Intel Meteor Lake CPUs (ab 2024/25)

Bildquelle: Intel

Das Bild zeigt eine Intel Core i7-3770K „Ivy Bridge“ CPU mit 4 Kernen und integriertem L3-Cache

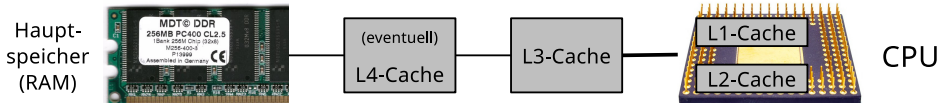
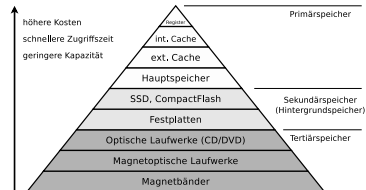
# Register, Cache und Hauptspeicher (4/4)

- Typische Kapazitäten der Cache-Ebenen:

- L1-Cache: 4 kB bis 256 kB
- L2-Cache: 256 kB bis 4 MB
- L3-Cache: 1 MB bis 16 MB

- Hauptspeicher**, auch **Arbeitsspeicher** oder **RAM** (*Random Access Memory* = Speicher mit wahlfreiem Zugriff) genannt

- Kapazität: Wenige hundert MB bis mehrere GB
- Alle Anfragen der CPU, die nicht vom Cache beantwortet werden können, werden an den Hauptspeicher weitergeleitet



## RAM und ROM

Im Gegensatz zum flüchtigen Hauptspeicher **RAM** ist **ROM** (*Read Only Memory*) ein nicht-flüchtiger Lesespeicher

# Grundlagen zum Speicher und Speicherverwaltung

## Wiederholung

- Bislang haben wir bzgl. Speicher geklärt:
  - Er nimmt Daten und die auszuführenden Programme auf
  - Er bildet eine Hierarchie ( $\Rightarrow$  Speicherpyramide)
    - Grund: Preis/Leistungsverhältnis
    - Je schneller ein Speicher ist, desto teurer und knapper ist er
  - Beim ersten Zugriff auf ein Datenelement, wird eine Kopie erzeugt, die entlang der Speicherhierarchie nach oben wandert
  - Da die obersten Speicherebenen praktisch immer voll belegt sind, müssen Daten verdrängt/ersetzt werden
  - Änderungen müssen durchgereicht (zurückgeschrieben) werden
- Es ist zu klären:
  - **Speicherverwaltung** (siehe Folie 13)
  - **Speicheradressierung** (siehe Folie 22)

# Speicherverwaltung

- Eine wesentliche Funktion von Betriebssystemen
- Weist Programmen auf deren Anforderung hin Teile des Speichers zu
- Gibt auch Teile des Speichers frei, die Programmen zugewiesen sind, wenn diese nicht benötigt werden

Gedankenspiel...

Wie würden Sie eine Speicherverwaltung realisieren ?!

- 3 Konzepte zur Speicherverwaltung:
  - 1 Statische Partitionierung
  - 2 Dynamische Partitionierung
  - 3 Buddy-Speicherverwaltung

Diese Konzepte sind schon etwas älter...



Bildquelle: unbekannt (eventuell IBM)

Eine gute Beschreibung der Konzepte zur Speicherverwaltung enthält...

- Operating Systems – Internals and Design Principles, William Stallings, 4.Auflage, Prentice Hall (2001), S.305-315
- Moderne Betriebssysteme, Andrew S. Tanenbaum, 3.Auflage, Pearson (2009), S.232-240

# Konzept 1: Statische Partitionierung

- Der Hauptspeicher wird in **Partitionen gleicher oder unterschiedlicher Größe** unterteilt
- Nachteile:
  - Es kommt zwangsläufig zu **interner Fragmentierung**  $\Rightarrow$  ineffizient
    - Das Problem wird durch Partitionen unterschiedlicher Größe abgemildert, aber nicht gelöst
  - Anzahl der Partitionen limitiert die Anzahl möglicher Prozesse
- Herausforderung: Ein Prozess benötigt mehr Speicher, als eine Partition groß ist
  - Dann muss der Prozess so implementiert sein, dass nur ein Teil des Programmcodes im Hauptspeicher liegt
    - Beim Nachladen von Programmcode (Modulen) kommt es zum *Overlay*  $\Rightarrow$  andere Module und Daten werden eventuell überschrieben

IBM OS/360 MFT in den 1960er Jahren nutzte statische Partitionierung

# Statische Partitionierung (1/2)

Partitionen  
gleicher Größe

- Werden **Partitionen gleicher Größe** verwendet, ist es egal, welche freie Partition ein Prozess zugewiesen wird
  - Sind alle Partitionen belegt, muss ein Prozess aus dem Hauptspeicher verdrängt werden
    - Die Entscheidung, welcher Prozess verdrängt wird, hängt vom verwendeten Scheduling-Verfahren ( $\implies$  Foliensatz 5) ab

Betriebssystem
8 MB
8 MB
8 MB
8 MB
8 MB
8 MB
8 MB
8 MB

## Statische Partitionierung (2/2)

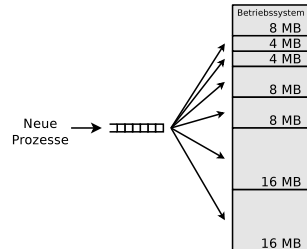
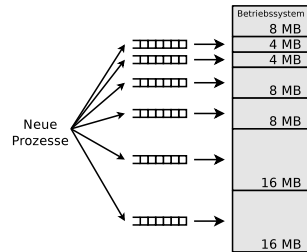
- Prozesse sollen eine möglichst passgenaue Partition erhalten
  - Ziel: Wenig interne Fragmentierung
- Werden **Partitionen unterschiedlicher Größe** verwendet, gibt es 2 Möglichkeiten, um Prozessen Partitionen zuzuweisen:

### 1 Eine eigene Prozess-Warteschlange für jede Partition

- Nachteil: Bestimmte Partitionen werden eventuell nie genutzt

### 2 Eine einzelne Warteschlange für alle Partitionen

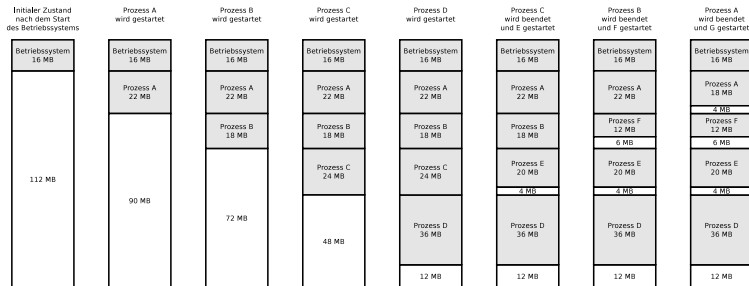
- Die Zuweisung der Partitionen an Prozesse ist flexibler möglich
- Auf veränderte Anforderungen der Prozesse kann rasch reagiert werden





# Konzept 2: Dynamische Partitionierung

- Jeder Prozess erhält im Hauptspeicher eine zusammenhängende Partition mit exakt der notwendigen Größe



- Es kommt zwangsläufig zu **externer Fragmentierung**  $\Rightarrow$  ineffizient
  - Mögliche Lösung: Defragmentierung
    - Voraussetzung: Verschiebbarkeit von Speicherblöcken
    - Verweise in Prozessen dürfen durch ein Verschieben von Partitionen nicht ungültig werden

# Realisierungskonzepte für dynamische Partitionierung

## • First Fit

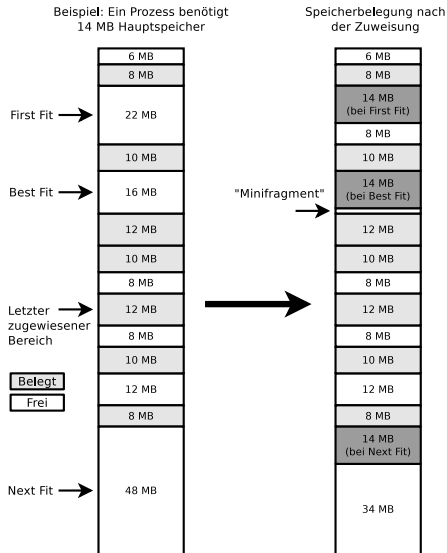
- Sucht ab dem Anfang des Adressraums einen passenden freien Block
- Schnelles Verfahren

## • Next Fit

- Sucht ab der letzten Zuweisung einen passenden freien Block
- Zerstückt schnell den großen Bereich freien Speichers am Ende des Adressraums

## • Best Fit

- Sucht den freien Block, der am besten passt
- Produziert viele Minifragmente und ist langsam



## Konzept 3: Buddy-Speicherverwaltung von Donald Knuth

- Zu Beginn gibt es nur einen Block, der den gesamten Speicher abdeckt
- Fordert ein Prozess einen Speicher an, wird zur nächsthöheren Zweierpotenz aufgerundet und ein entsprechender, freier Block gesucht
  - Existiert kein Block dieser Größe, wird nach einem Block doppelter Größe gesucht und dieser in 2 Hälften (sogenannte *Buddies*) unterteilt
    - Der erste Block wird dann dem Prozess zugewiesen
  - Existiert auch kein Block doppelter Größe, wird ein Block vierfacher Größe gesucht, usw. . .
- Wird Speicher freigegeben, wird geprüft, ob 2 Hälften gleicher Größe sich wieder zu einem größeren Block zusammenfassen lassen
  - Es werden nur zuvor vorgenommene Unterteilungen rückgängig gemacht!

### Buddy-Speicherverwaltung in der Praxis

- Der Linux-Kernel verwendet eine Variante der Buddy-Speicherverwaltung für die Zuweisung der Seiten
- Das Betriebssystem verwaltet für jede möglich Blockgröße eine „Frei-Liste“
- <https://www.kernel.org/doc/gorman/html/understand/understand009.html>

# Beispiel zum Buddy-Verfahren

	0	128	256	384	512	640	768	896	1024
Anfangszustand	1024 KB								
100 KB Anforderung (=> A)	512 KB				512 KB				
	256 KB			256 KB	512 KB				
	128 KB	128 KB	256 KB			512 KB			
	A	128 KB	256 KB			512 KB			
240 KB Anforderung (=> B)	A	128 KB	B			512 KB			
60 KB Anforderung (=> C)	A	64 KB	64 KB	B			512 KB		
	A	C	64 KB	B			512 KB		
251 KB Anforderung (=> D)	A	C	64 KB	B			256 KB	256 KB	
	A	C	64 KB	B			D	256 KB	
Freigabe B	A	C	64 KB	256 KB			D	256 KB	
Freigabe A	128 KB	C	64 KB	256 KB			D	256 KB	
75 KB Anforderung (=> E)	E	C	64 KB	256 KB			D	256 KB	
	E	64 KB	64 KB	256 KB			D	256 KB	
Freigabe C	E	128 KB	256 KB			D	256 KB		
Freigabe E	128 KB	128 KB	256 KB			D	256 KB		
	256 KB			256 KB			D	256 KB	
	512 KB				D				
Freigabe D	512 KB				256 KB			256 KB	
	512 KB					512 KB			
	1024 KB								

- Nachteil: Interner und externer Verschnitt (Fragmentierung)

# Informationen zur Fragmentierung des Speichers

- Die DMA-Zeile zeigt die ersten 16 MB im System
  - Die Adressbusbreite des Intel 80286 ist  $2^{24} \Rightarrow 16$  MB max. adressierbarer Speicher
- Die DMA32-Zeile zeigt den Speicher  $> 16$  MB und  $< 4$  GB im System
  - Die Adressbusbreite beim Intel 80386, 80486, Pentium I/II/III/IV, ... ist  $2^{32} \Rightarrow 4$  GB max. adressierbarer Speicher
- Die Normal-Zeile zeigt den Speicher  $> 4$  GB im System
  - Moderne Systeme haben meist eine Adressbusbreite von 36, 44 oder 48 Bits

Weitere Information zu den Zeilen: <https://utcc.utoronto.ca/~cks/space/blog/linux/KernelMemoryZones>

```
# cat /proc/buddyinfo
Node 0, zone  DMA      1      1      1      0      2      1      1      0      1      1      3
Node 0, zone  DMA32    208    124    1646    566    347    116    139    115    17      4    212
Node 0, zone  Normal    43      62     747    433    273    300    254    190    20      8    287
```

- Spalte 1  $\Rightarrow$  Anzahl freier Blöcke („Buddies“) der Größe  $2^0 * \text{PAGE\_SIZE} \Rightarrow 4$  kB
- Spalte 2  $\Rightarrow$  Anzahl freier Blöcke („Buddies“) der Größe  $2^1 * \text{PAGE\_SIZE} \Rightarrow 8$  kB
- Spalte 3  $\Rightarrow$  Anzahl freier Blöcke („Buddies“) der Größe  $2^2 * \text{PAGE\_SIZE} \Rightarrow 16$  kB
- ...
- Spalte 11  $\Rightarrow$  Anz. frei. Blöcke („Buddies“) der Größe  $2^{10} * \text{PAGE\_SIZE} \Rightarrow 4096$  kB = 4 MB

PAGESIZE = 4096 Bytes = 4 kB

Die Seitengröße eines Linux-Systems gibt folgendes Kommando aus: `$ getconf PAGESIZE`

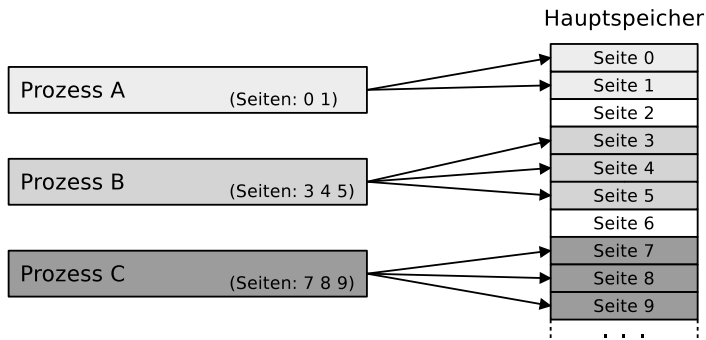
# Speicheradressierung

!!! Frage !!!

Wie greifen Prozesse auf den Speicher zu?

- Auf 16 Bit-Architekturen sind  $2^{16}$  Speicheradressen und damit bis zu 65.536 Byte, also **64 kB** adressierbar
- Auf 32 Bit-Architekturen sind  $2^{32}$  Speicheradressen und damit bis zu 4.294.967.296 Byte, also **4 GB** adressierbar
- Auf 64 Bit-Architekturen sind  $2^{64}$  Speicheradressen und damit bis zu 18.446.744.073.709.551.616 Byte, also **16 Exabyte** adressierbar

# Idee: Direkter Zugriff auf den Speicher



- Naheliegende Idee: Direkter Speicherzugriff durch die Prozesse  
 ⇒ **Real Mode (Real Address Mode)**
- Betriebsart x86-kompatibler Prozessoren
- **Kein Zugriffsschutz**
  - Jeder Prozess kann auf den gesamten adressierbaren Speicher zugreifen
    - Inakzeptabel für Multitasking-Betriebssysteme

# Real Mode (Real Address Mode)

- **Maximal 1 MB Hauptspeicher adressierbar**
  - Maximaler Speicherausbau eines Intel 8086
  - Grund: Der Adressbus des 8086 verfügt nur über 20 Adressleitungen
    - 20 Busleitungen  $\implies$  20 Bits lange Speicheradressen  $\implies$  Die CPU kann  $2^{20} = \text{ca. } 1 \text{ MB}$  Speicher adressieren
- Die Bezeichnung „Real Mode“ wurde mit dem Intel 80286 eingeführt
  - Im Real Mode greift die CPU wie ein 8086 auf den Hauptspeicher zu
  - Jede x86-kompatible CPU startet im Real Mode

<https://wiki.osdev.org/UEFI>

- Auf einem veralteten System mit einer **BIOS**-Firmware wird nach der Systeminitialisierung durch das BIOS (Konfiguration des Speichercontrollers, Konfiguration des PCI-Bus, Initialisierung der Grafikkarte usw.) der Real-Modus gestartet. Der Bootloader oder das Betriebssystem müssen anschließend in den Schutzmodus bzw. Protected Mode (Paging) wechseln
- Bei einem System mit einer **UEFI**-Firmware (Unified Extensible Firmware Interface) führt die Firmware alle diese Initialisierungsschritte durch und schaltet in den Schutzmodus bzw. Protected Mode (Paging)



# Anforderungen an die Speicherverwaltung

## ● Relokation

- Werden Prozesse aus dem Hauptspeicher verdrängt, ist nicht bekannt, an welcher Stelle sie später wieder in den Hauptspeicher geladen werden
- Erkenntnis: Prozesse dürfen keine Referenzen auf physische Speicheradressen enthalten

## ● Schutz

- Speicherbereiche müssen geschützt werden vor unbeabsichtigtem oder unzulässigem Zugriff durch anderen Prozesse
- Erkenntnis: Zugriffe müssen (durch die CPU) überprüft werden

## ● Gemeinsame Nutzung

- Trotz Speicherschutz muss eine Kooperation der Prozesse mit gemeinsamem Speicher (Shared Memory) möglich sein  $\Rightarrow$  Foliensatz 6

## ● Vergrößerte Kapazität

- Es soll mehr Speicher verwendet werden können, als physisch existiert
- Erkenntnis: Ist der Hauptspeicher voll, können Daten ausgelagert werden

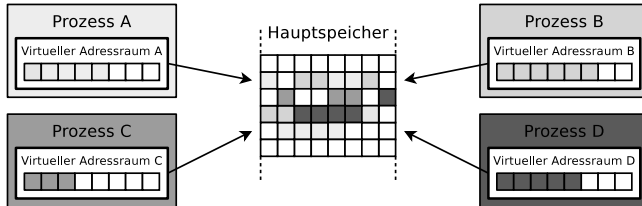
## ● Lösung: **Protected mode** und **virtueller Speicher**

# Protected Mode (Schutzmodus)

- Betriebsart x86-kompatibler Prozessoren
  - Eingeführt mit dem Intel 80286
- Erhöht die Menge des adressierbaren Speichers
  - 16-Bit Protected Mode beim 80286  $\implies$  16 MB Hauptspeicher
  - 32-Bit Protected Mode beim 80386  $\implies$  4 GB Hauptspeicher
  - Bei späteren Prozessoren hängt die Menge des adressierbaren Speichers von der Anzahl der Busleitungen im Adressbus ab
- Realisiert **virtuellen Speicher**
  - Prozesse verwenden keine physischen Hauptspeicheradressen
    - Das würde bei Multitasking-Systemen zu Problemen führen
  - Stattdessen besitzt jeder Prozess einen eigenen **Adressraum**
    - Der **virtuelle Speicher** ist unabhängig von der verwendeten Speichertechnologie und den gegebenen Ausbaumöglichkeiten
    - Er besteht aus logischen Speicheradressen, die von der Adresse 0 aufwärts durchnummeriert sind

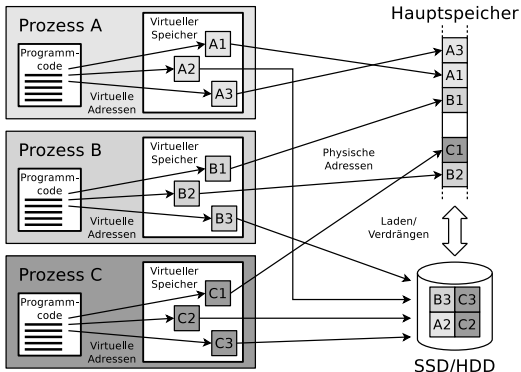
# Virtueller Speicher (1/2)

- Adressräume können nach Bedarf erzeugt oder gelöscht werden und sie sind geschützt
  - Kein Prozess kann ohne vorherige Vereinbarung auf den Adressraum eines anderen Prozesses zugreifen
- Mapping** = Virtuellen Speicher auf physischen Speicher abbilden



- Dank virtuellem Speicher wird der Hauptspeicher besser ausgenutzt
  - Prozesse müssen nicht am Stück im Hauptspeicher liegen
  - Darum ist die Fragmentierung des Hauptspeichers kein Problem

# Virtueller Speicher (2/2)



- Durch virtuellen Speicher kann mehr Speicher angesprochen und verwendet werden, als physisch im System vorhanden ist
- **Auslagern (Swapping)** geschieht für Benutzer und Prozesse transparent

Virtueller Speicher ist anschaulich erklärt bei...

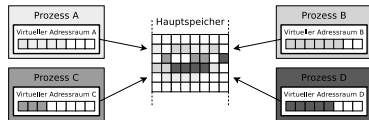
**Betriebssysteme**, Carsten Vogt, 1. Auflage, Spektrum Akademischer Verlag (2001), S. 152

Im Protected Mode unterstützt die CPU 2 Methoden zur Speicherverwaltung

- **Segmentierung** existiert ab dem 80286
- **Paging** (siehe Folie 29) existiert ab dem 80386
- Beide Verfahren sind Implementierungsvarianten des **virtuellen Speichers**

# Paging: Seitenorientierter Speicher

- **Virtuelle Seiten** der Prozesse werden auf **physische Seiten** im Hauptspeicher abgebildet
  - Alle Seiten haben die gleiche Länge
    - Die Seitenlänge ist üblicherweise 4 kB (bei der Alpha-Architektur und der UltraSPARC-Architektur: 8 kB, bei Apple Silicon: 16 kB)
- Vorteile:
  - Keine externe Fragmentierung
  - Interne Fragmentierung kann nur in der letzten Seite jedes Prozesses auftreten
- Das Betriebssystem verwaltet **für jeden Prozess eine Seitentabelle**
  - In dieser steht, wo sich die einzelnen Seiten des Prozesses befinden
- Prozesse arbeiten nur mit **virtuellen Speicheradressen**
  - Virtuelle Speicheradressen bestehen aus 2 Teilen
    - Der werthöhere Teil enthält die Seitennummer
    - Der wertniedrigere Teil enthält den Offset (Adresse innerhalb einer Seite)
  - Die Länge der virtuellen Adressen ist architekturabhängig (hängt von der Anzahl der Busleitungen im Adressbus ab) und ist 16, 32 oder 64 Bits



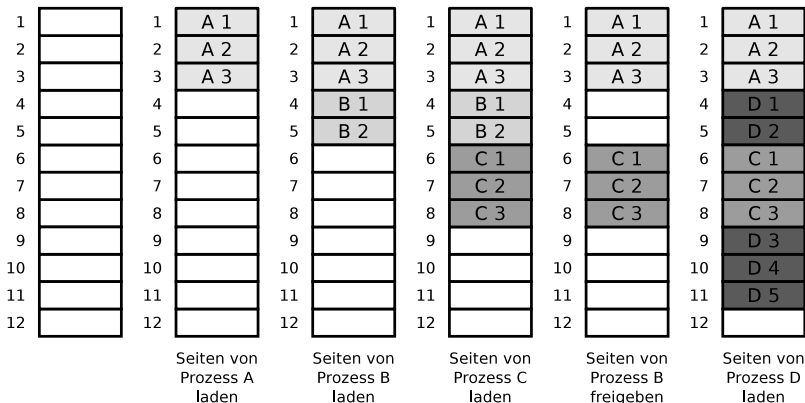
# Zuweisung von Prozessseiten zu freien physischen Seiten

- Prozesse müssen nicht am Stück im Hauptspeicher liegen

⇒ Keine externe Fragmentierung

Hauptspeicher

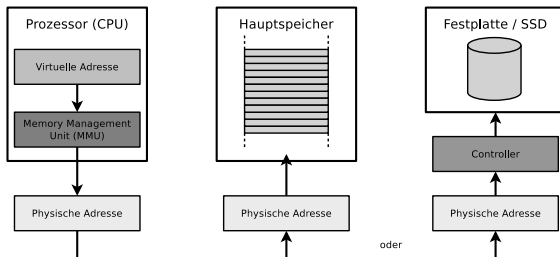
Physische  
Seiten-  
nummer



Das Thema ist anschaulich erklärt bei: **Operating Systems**, William Stallings, 4.Auflage, Prentice Hall (2001)

# Adressumwandlung durch die Memory Management Unit

- Virtuelle Speicheradressen übersetzt die CPU mit der **MMU** und der Seitentabelle in physische Adressen
  - Das Betriebssystem prüft, ob sich die physische Adresse im Hauptspeicher, oder auf der SSD/HDD befindet



- Befinden sich die Daten auf der SSD/HDD, muss das Betriebssystem die Daten in den Hauptspeicher einlesen
- Ist der Hauptspeicher voll, muss das Betriebssystem andere Daten aus dem Hauptspeicher auf die SSD/HDD verdrängen (*swappen*)

Das Thema MMU ist anschaulich erklärt bei...

- **Betriebssysteme**, Carsten Vogt, 1. Auflage, Spektrum Akademischer Verlag (2001), S. 152-153
- **Moderne Betriebssysteme**, Andrew S. Tanenbaum, 2. Auflage, Pearson (2009), S. 223-226

# Implementierung der Seitentabelle

- Die Länge der Seiten hat Auswirkungen:
  - Kurze Seiten:** Weniger interner Verschnitt, aber längere Seitentabelle
  - Lange Seiten:** Kürzere Seitentabelle, aber mehr interner Verschnitt
- Seitentabellen liegen im Hauptspeicher

$$\text{Maximale Größe der Seitentabelle} = \frac{\text{Virtueller Adressraum}}{\text{Seitengröße}} * \text{Größe der Seitentableneinträge}$$

- Maximale Größe der Seitentabellen bei 32 Bit-Betriebssystemen:

$$\frac{4 \text{ GB}}{4 \text{ kB}} * 4 \text{ Bytes} = \frac{2^{32} \text{ Bytes}}{2^{12} \text{ Bytes}} * 2^2 \text{ Bytes} = 2^{22} \text{ Bytes} = 4 \text{ MB}$$

- Jeder Prozess in einem Multitasking-Betriebssystem hat eine Seitentabelle

Bei 64 Bit-Betriebssystemen können die Seitentabellen der einzelnen Prozesse deutlich größer sein

Da aber die meisten im Alltag laufenden Prozesse nicht mehrere Gigabyte Speicher benötigen, fällt der Overhead durch die Verwaltung der Seitentabellen auf modernen Computern gering aus



# Struktur der Seitentabellen (Page Table)

- Jeder Eintrag in der Seitentabelle enthält u.a.:
  - **Present-Bit**: Gibt an, ob die Seite im Hauptspeicher liegt
  - **Dirty-Bit** (*Modified-Bit*): Gibt an, ob die Seite verändert wurde
  - **Reference-Bit**: Gibt an, ob es einen (auch lesenden!) Zugriff auf die Seite gab  $\implies$  das ist evtl. wichtig für die verwendete Seitenersetzungsstrategie
  - **Weitere Steuerbits**: Hier ist u.a. festgelegt, ob...
    - Prozesse im Benutzermodus nur lesend oder auch schreibend auf die Seite zugreifen dürfen (**Read/Write-Bit**)
    - Prozesse im Benutzermodus auf die Seite zugreifen dürfen (**User/Supervisor-Bit**)
    - Änderungen sofort (*Write-Through*) oder erst beim verdrängen (*Write-Back*) durchgeschrieben werden (**Write-Through-Bit**)
    - Die Seite in den Cache geladen darf oder nicht (**Cache-Disable-Bit**)
  - **Physische Seitenadresse**: Wird mit dem Offset der virtuellen Adresse verknüpft

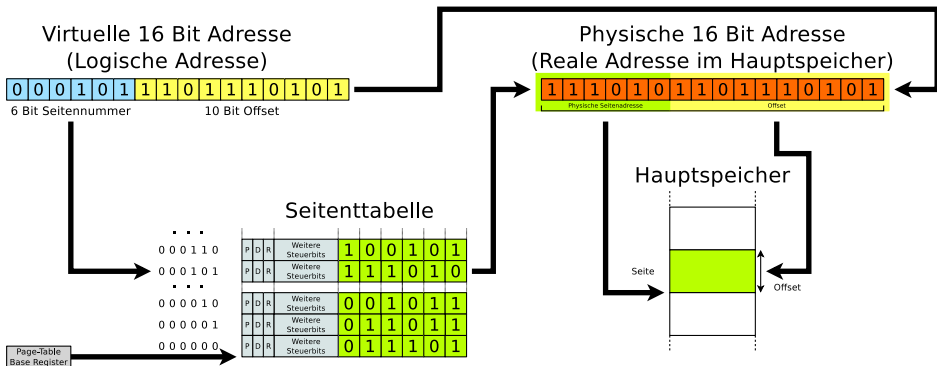
Virtuelle (logische) Adresse

Seitennummer	Offset
--------------	--------

Seitentabelleneintrag

P	D	R	Weitere Steuerbits	Physische Seitenadresse
---	---	---	--------------------	-------------------------

# Adressumwandlung beim Paging (einstufig)



- Einstufiges Paging ist auf 16 Bit-Architekturen ausreichend
- Auf Architekturen  $\geq 32$  Bit realisieren die Betriebssysteme mehrstufiges Paging

2 Register ermöglichen der MMU den Zugriff auf die Seitenabelle

- **Page-Table Base Register (PTBR):** Adresse wo die Seitentabelle des laufenden Prozesses anfängt
- **Page-Table Length Register (PTLR):** Länge der Seitentabelle des laufenden Prozesses

# Adressumwandlung beim Paging (zweistufig)

Virtuelle 32 Bit Adresse  
(Logische Adresse)



Hauptseitentabelle  
(erste Stufe)

1023	P	Weitere Steuerbits	Seitentabelle
1022	P	Weitere Steuerbits	Seitentabelle
1021	P	Weitere Steuerbits	Seitentabelle
1020	P	Weitere Steuerbits	Seitentabelle
⋮	⋮	⋮	⋮
1	P	Weitere Steuerbits	Seitentabelle
0	P	Weitere Steuerbits	Seitentabelle

32 Bit

Page-Table Base Register

Zweite Stufe der  
Seitentabelle

1023	P	D	R	Weitere Steuerbits	Physische Seitenadresse
1022	P	D	R	Weitere Steuerbits	Physische Seitenadresse
⋮	⋮	⋮	⋮	⋮	⋮
137	P	D	R	Weitere Steuerbits	Physische Seitenadresse
136	P	D	R	Weitere Steuerbits	Physische Seitenadresse
135	P	D	R	Weitere Steuerbits	Physische Seitenadresse
⋮	⋮	⋮	⋮	⋮	⋮
1	P	D	R	Weitere Steuerbits	Physische Seitenadresse
0	P	D	R	Weitere Steuerbits	Physische Seitenadresse

32 Bit

Physische 32 Bit Adresse  
(Reale Adresse im Hauptspeicher)

Physische Seitenadresse    Offset

Hauptspeicher

Seite

Offset

Das Thema Paging ist anschaulich erklärt bei...

- Betriebssysteme, Eduard Glatz, 2.Auflage, dpunkt (2010), S.450-457
- Betriebssysteme, William Stallings, 4.Auflage, Pearson (2003), S.394-399
- <http://wiki.osdev.org/Paging>

# Warum mehrstufiges Paging?

Wir wissen bereits...

- Bei 32 Bit-Betriebssystemen mit 4 kB Seitenlänge kann die Seitentabelle jedes Prozesses 4 MB groß sein (siehe Folie 32)
- Bei 64 Bit-Betriebssystemen können die Seitentabellen wesentlich größer sein

- Mehrstufiges Paging entlastet den Hauptspeicher
  - Bei der Berechnung einer physischen Adresse durchläuft das Betriebssystem die Teilseiten Stufe für Stufe
  - Einzelne Teilseiten können bei Bedarf auf den Auslagerungsspeicher verdrängt werden, um Platz im Hauptspeicher zu schaffen

Architektur	Seitentabelle	Virtuelle Adresslänge	Aufteilung <sup>a</sup>
IA32 (x86-32)	zweistufig	32 Bits	10+10+12
IA32 mit PAE <sup>b</sup>	dreistufig	32 Bits	2+9+9+12
PPC64	dreistufig	41 Bits	10+10+9+12
AMD64 (x86-64)	vierstufig	48 Bits	9+9+9+9+12
Intel Ice Lake Xeon Scalable <sup>c</sup>	fünfstufig	57 Bits	9+9+9+9+9+12

<sup>a</sup> Die letzte Zahl gibt die Länge des Offset in Bits an. Die übrigen Zahlen geben die Längen der Seitentabellen an.

<sup>b</sup> PAE = Physical Address Extension. Mit dieser Paging-Erweiterung des Pentium Pro Prozessors können mehr als 4 GB RAM vom Betriebssystem adressiert werden. Der pro Prozess nutzbare Arbeitsspeicher ist jedoch weiterhin auf 4 GB begrenzt.

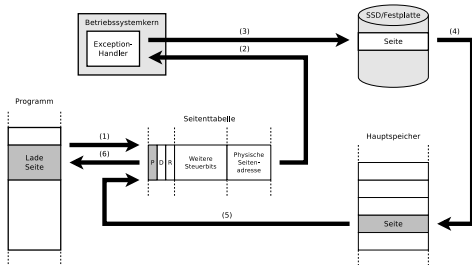
<sup>c</sup> <https://software.intel.com/content/dam/develop/public/us/en/documents/5-level-paging-white-paper.pdf>

Gute Quelle zum Thema: **Architektur von Betriebssystemen**, Horst Wettstein, Hanser (1984), S.249

# Page Fault Ausnahme (Exception) – Seitenfehler

- Ein Prozess versucht (1) auf eine Seite zuzugreifen, die nicht im physischen Hauptspeicher liegt

- Das **Present-Bit** in jedem Eintrag der Seitentabelle gibt an, ob die Seite im Hauptspeicher ist oder nicht



- Ein Software-Interrupt (Exception) wird ausgelöst (2), um vom Benutzermodus in den Kernelmodus zu wechseln
- Das Betriebssystem...
  - lokalisiert (3) die Seite mit Hilfe des Controllers und Gerätetreibers auf dem Auslagerungsspeicher (SSD/HDD)
  - kopiert (4) die Seite in eine freie Hauptspeicherseite
  - aktualisiert (5) die Seitentabelle
  - gibt die Kontrolle an das Programm zurück (6)
    - Dieses führt die Anweisung, die zum Page Fault führte, erneut aus

## Access Violation Ausnahme (Exception) oder General Protection Fault Ausnahme (Exception)

- Heißt auch **Segmentation fault** oder **Segmentation violation**
  - Ein Paging-Problem, das nichts mit Segmentierung zu tun hat!
- Ein Prozess versucht auf eine virtuelle Speicheradresse zuzugreifen, auf die er nicht zugreifen darf

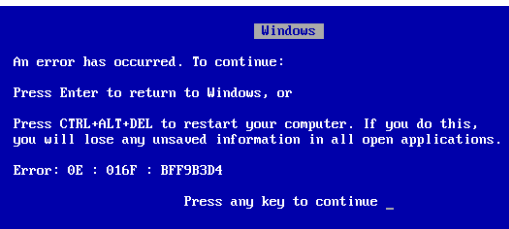
```
A problem has been detected and Windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPMD0CON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.
```



- Ergebnis: Systemabstürze bei alten Windows-Betriebssystemen (Blue Screen), Linux gibt das Signal SIGSEGV zurück
- Beispiel: Ein Prozess versucht in eine Seite zu schreiben, auf die er nur lesend zugreifen darf

Quelle: Herold H. (1996) *UNIX-Systemprogrammierung*. 2. Auflage. Addison-Wesley

Bildquelle (oben): Reader781. Wikimedia (CC0)

Bildquelle (unten): Akhristov. Wikimedia (CC0)

# Hitrate und Missrate

## Eine effiziente Speicherverwaltung für Hauptspeicher und Cache...

- hält diejenigen Seiten im Speicher, auf die häufig zugegriffen wird
  - identifiziert diejenigen Seiten, auf die in naher Zukunft vermutlich nicht zugegriffen wird und verdrängt diese bei Bedarf
- 
- Bei einer Anfrage an einen Speicher sind 2 Ergebnisse möglich:
    - **Hit**: Angefragte Daten sind vorhanden (Treffer)
    - **Miss**: Angefragte Daten sind nicht vorhanden (verfehlt)
  - 2 Kennzahlen bewerten die Effizienz eines Speichers:
    - **Hitrate**: Anzahl der Anfragen an den Speicher mit Ergebnis Hit, geteilt durch die Gesamtanzahl der Anfragen
      - Ergebnis liegt zwischen 0 und 1
      - Je höher der Wert, desto höher ist die Effizienz des Speichers
    - **Missrate**: Anzahl der Anfragen an den Speicher mit Ergebnis Miss, geteilt durch die Gesamtanzahl der Anfragen
      - $\text{Missrate} = 1 - \text{Hitrate}$

# Seitenersetzungsstrategien

- Es ist sinnvoll, die Daten ( $\implies$  **Seiten**) im Speicher zu halten, auf die häufig zugegriffen wird
- Einige **Ersetzungsstrategien**:
  - **OPT** (Optimale Strategie)
  - **LRU** (Least Recently Used)
  - **LFU** (Least Frequently Used)
  - **FIFO** (First In First Out)
  - **Clock / Second Chance**
  - **TTL** (Time To Live)
  - **Random**

Eine gute Beschreibung der Seitenersetzungsstrategien...

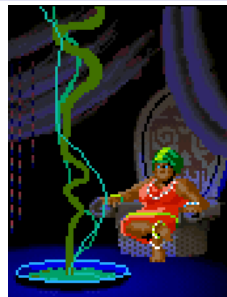
- OPT, FIFO, LRU und Clock enthält **Operating Systems**, William Stallings, 4.Auflage, Prentice Hall (2001), S.355-363
- FIFO, LRU, LFU und Clock enthält **Betriebssysteme**, Carsten Vogt, 1.Auflage, Spektrum Verlag (2001), S.162-163
- FIFO, LRU und Clock enthält **Moderne Betriebssysteme**, Andrew S. Tanenbaum, 2.Auflage, Pearson (2009), S.237-242
- FIFO, LRU, LFU und Clock enthält **Betriebssysteme**, Eduard Glatz, 2.Auflage, dpunkt (2010), S.471-476



# Optimale Strategie (OPT)

Bildquelle: Lukasfilm Games

- Verdrängt die Seite, auf die **am längsten in der Zukunft nicht zugegriffen** wird
- Unmöglich zu implementieren
  - Grund: Niemand kann in die Zukunft sehen
    - Darum muss das Betriebssystem die Vergangenheit berücksichtigen
- Mit OPT bewertet man die Effizienz anderer Ersetzungsstrategien


Anfragen: **1 2 3 4 1 2 5 1 2 3 4 5**

1. Seite:	<b>1</b>	1	1	1	1	1	1	1	1	<b>3</b>	3	3
2. Seite:		<b>2</b>	2	2	2	2	2	2	2	2	<b>4</b>	4
3. Seite:			<b>3</b>	<b>4</b>	4	4	<b>5</b>	5	5	5	5	5

→ 7 Miss

Wenn nicht mehr genug Anfragen in der Zukunft da sind, um eine Entscheidung zu treffen, gilt FIFO

Die **Anfragen** sind Anforderungen an Seiten im virtuellen Adressraum eines Prozesses. Wenn eine angefragte Seite nicht schon im Cache ist, wird sie aus dem Hauptspeicher oder dem Auslagerungsspeicher (Swap) nachgeladen

# Least Recently Used (LRU)

- Verdrängt die Seite, auf die **am längsten nicht zugegriffen** wurde
- Das Betriebssystem verwaltet eine Warteschlange, in der die Seitennummern eingereiht sind
  - Wird eine Seite in den Speicher geladen oder auf diese zugegriffen, wird sie am Anfang der Warteschlange eingereiht
  - Ist der Speicher voll und es kommt zum Miss, lagert das Betriebssystem die Seite am Ende der Warteschlange aus
- Nachteil: Berücksichtigt nicht die Zugriffshäufigkeit

Anfragen: **1 2 3 4 1 2 5 1 2 3 4 5**

1. Seite:	1	1	1	4	4	4	5	5	5	3	3	3
2. Seite:		2	2	2	1	1	1	1	1	1	4	4
3. Seite:			3	3	3	2	2	2	2	2	2	5

→ 10 Miss

Queue:	1	2	3	4	1	2	5	1	2	3	4	5
		1	2	3	4	1	2	5	1	2	3	4
			1	2	3	4	1	2	5	1	2	3

# Least Frequently Used (LFU)

- Verdrängt die Seite, auf die **am wenigsten** **zugegriffen** wurde
- Das Betriebssystem verwaltet für jede Seite im Speicher in der Seitentabelle einen Referenzzähler, der die Anzahl der Zugriffe speichert
  - Sind alle Speicherplätze belegt und kommt es zum Miss, wird die Seite verdrängt, deren Referenzzähler den niedrigsten Wert hat
- Vorteil: Berücksichtigt die Zugriffshäufigkeit
- Nachteil: Seiten, auf die in der Vergangenheit häufig zugegriffen wurde, können den Speicher blockieren

Anfragen: **1 2 3 4 1 2 5 1 2 3 4 5**

1. Seite:	<sub>1</sub> 1	<sub>1</sub> 1	<sub>1</sub> 1	<sub>1</sub> 4	<sub>1</sub> 4	<sub>1</sub> 4	<sub>1</sub> 5	<sub>1</sub> 5	<sub>1</sub> 5	<sub>1</sub> 3	<sub>1</sub> 4	<sub>1</sub> 5
2. Seite:		<sub>1</sub> 2	<sub>1</sub> 2	<sub>1</sub> 2	<sub>1</sub> 1	<sub>1</sub> 1	<sub>1</sub> 1	<sub>2</sub> 1	<sub>2</sub> 1	<sub>2</sub> 1	<sub>2</sub> 1	<sub>2</sub> 1
3. Seite:			<sub>1</sub> 3	<sub>1</sub> 3	<sub>1</sub> 3	<sub>1</sub> 2	<sub>1</sub> 2	<sub>1</sub> 2	<sub>2</sub> 2	<sub>2</sub> 2	<sub>2</sub> 2	<sub>2</sub> 2

→ 10 Miss

Wenn der Referenzzähler keinen eindeutigen Wert liefert, um eine Entscheidung zu treffen, gilt FIFO

# First In First Out (FIFO)

- Verdrängt die Seite, die sich **am längsten im Speicher** befindet
- Annahme: Eine Vergrößerung des Speichers führt zu weniger oder schlechtestenfalls gleich vielen Miss
- Problem: Laszlo Belady zeigte 1969, dass bei bestimmten Zugriffsmustern FIFO bei einem vergrößerten Speicher zu mehr Miss führt ( $\implies$  **Belady's Anomalie**)
  - Bis zur Entdeckung von Belady's Anomalie galt FIFO als gute Ersetzungsstrategie

# Belady's Anomalie (1969)

Anfragen: **1 2 3 4 1 2 5 1 2 3 4 5**

1. Seite:	1	1	1	4	4	4	5	5	5	5	5
2. Seite:		2	2	2	1	1	1	1	1	3	3
3. Seite:			3	3	3	2	2	2	2	2	4

→ 9 Miss

1. Seite:	1	1	1	1	1	1	5	5	5	5	4
2. Seite:		2	2	2	2	2	2	1	1	1	5
3. Seite:			3	3	3	3	3	3	2	2	2
4. Seite:				4	4	4	4	4	4	3	3

→ 10 Miss

## Weitere Informationen zu Belady's Anomalie

Belady, Nelson and Shedler. *An Anomaly in Space-time Characteristics of Certain Programs Running in a Paging Machine*. Communications of the ACM. Volume 12 Issue 6. June 1969

# Clock / Second Chance

- Dieses Verfahren verwendet das *Reference-Bit* (siehe Folie 33), das das Betriebssystem für jede Seite in der Seitentabelle führt
  - Wird eine Seite in den Speicher geladen  $\Rightarrow$  Reference-Bit = 0
  - Wird auf eine Seite zugegriffen  $\Rightarrow$  Reference-Bit = 1
- Ein Zeiger zeigt auf die zuletzt zugegriffene Seite
- Beim Miss wird der Speicher ab dem Zeiger nach der ersten Seite durchsucht, deren Reference-Bit den Wert 0 hat
  - Diese Seite wird ersetzt
  - Bei allen bei der Suche durchgesehenen Seiten, bei denen das Reference-Bit den Wert 1 hat, wird es auf 0 gesetzt

Anfragen: **1 2 3 4 1 2 5 1 2 3 4 5**

1. Seite:	<sub>0</sub> 1 <sup>x</sup>	<sub>0</sub> 1	<sub>0</sub> 1	<sub>0</sub> 4 <sup>x</sup>	<sub>0</sub> 4	<sub>0</sub> 4	<sub>0</sub> 5 <sup>x</sup>	<sub>0</sub> 5	<sub>0</sub> 5	<sub>0</sub> 3 <sup>x</sup>	<sub>0</sub> 4 <sup>x</sup>	<sub>0</sub> 4
2. Seite:		<sub>0</sub> 2 <sup>x</sup>	<sub>0</sub> 2	<sub>0</sub> 2	<sub>0</sub> 1 <sup>x</sup>	<sub>0</sub> 1	<sub>0</sub> 1	<sub>1</sub> 1 <sup>x</sup>	<sub>1</sub> 1	<sub>1</sub> 1	<sub>1</sub> 1	<sub>0</sub> 5 <sup>x</sup>
3. Seite:			<sub>0</sub> 3 <sup>x</sup>	<sub>0</sub> 3	<sub>0</sub> 3	<sub>0</sub> 2 <sup>x</sup>	<sub>0</sub> 2	<sub>0</sub> 2	<sub>1</sub> 2 <sup>x</sup>	<sub>1</sub> 2	<sub>0</sub> 2	<sub>0</sub> 2

→ 10 Miss

Linux, BSD-UNIX, VAX/VMS (ursprünglich von Digital Equipment Corporation) und Microsoft Windows NT 4.0 auf Uniprocessor-Systemen verwenden Clock oder Varianten davon

## Weitere Ersetzungsstrategien

- **TTL (Time To Live):** Jede Seite bekommt beim Laden in den Speicher eine Lebenszeit zugeordnet
  - Ist die TTL überschritten, kann die Seite verdrängt werden

Das Konzept wird nicht von Betriebssystemen verwendet. Es ist aber sinnvoll zum Caching von Webseiten (Inhalten aus dem WWW)

Interessante Quelle: **Caching with expiration times**. Gopalan P, Harloff H, Mehta A, Mihail M, Vishnoi N (2002)  
<https://www.cc.gatech.edu/~mihail/www-papers/soda02.pdf>

- **Random:** Zufällige Seiten werden verdrängt
  - Vorteile: Simple und ressourcenschonende Ersetzungsstrategie
    - Grund: Es müssen keine Informationen über das Zugriffsverhalten gespeichert werden

Die Ersetzungsstrategie Random wird (wurde) in der Praxis eingesetzt

- Die Betriebssysteme IBM OS/390 und Windows NT 4.0 auf SMP-Systemen verwenden die Ersetzungsstrategie Random (Quelle OS/390: Pancham P, Chaudhary D, Gupta R. (2014) *Comparison of Cache Page Replacement Techniques to Enhance Cache Memory Performance*. International Journal of Computer Applications. Band 98, Nummer 19) (Quelle NT4: <http://www.itprotoday.com/management-mobility/inside-memory-management-part-2>)
- Die Intel i860 RISC-CPU verwendet die Ersetzungsstrategie Random für den Cache (Quelle: Rhodehamel M. (1989) *The Bus Interface and Paging Units of the i860 Microprocessor*. Proceedings of the IEEE International Conference on Computer Design. S. 380-384)