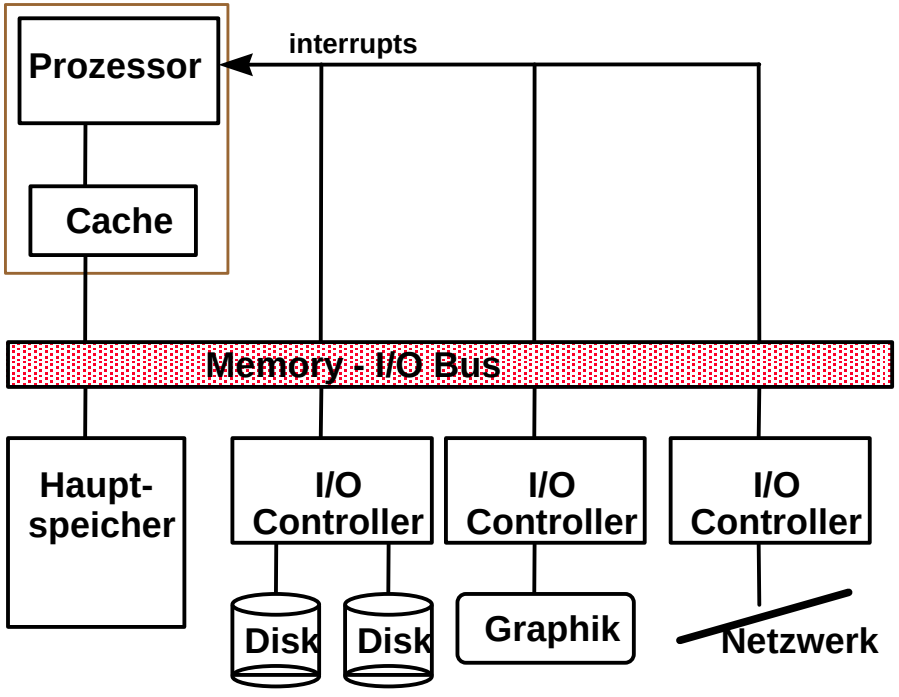
Überblick

- Einleitung
  - Lit., Motivation, Geschichte, v.Neumann-Modell
- Befehlsschnittstelle
- Mikroarchitektur
- Speicherarchitektur
- Ein-/Ausgabe
- Multiprozessorsysteme, ...

Kap.4

Speicherarchitektur

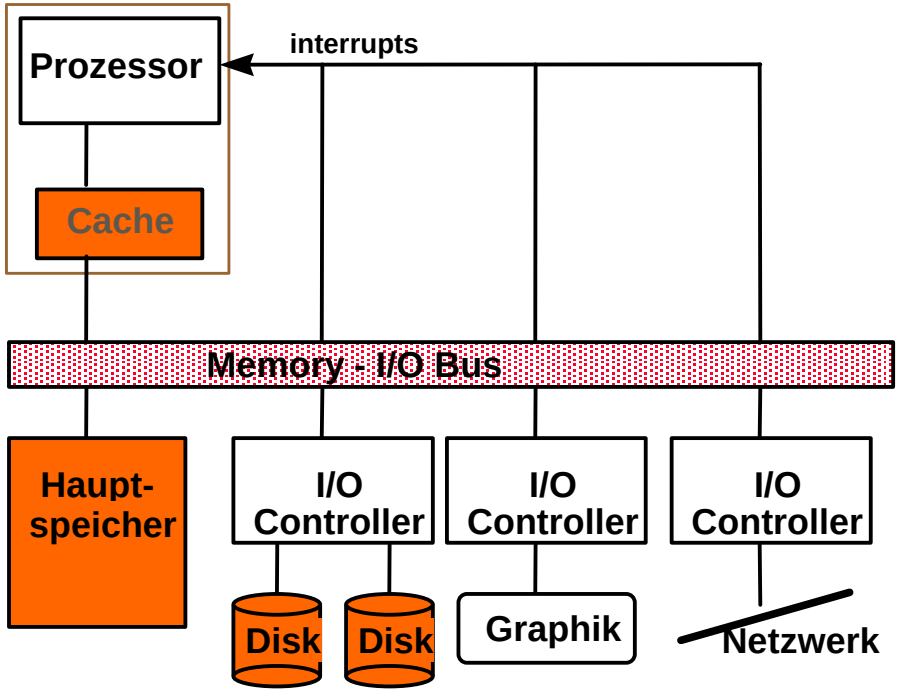
Prinzipieller Aufbau eines Rechners



Bestandteile:

- Prozessor mit Cache
- Hauptspeicher
- Externe Speicher
- Eingabegeräte (Tastatur, Maus)
- Ausgabegeräte (Bildschirm, Drucker, Plotter)
- Busse

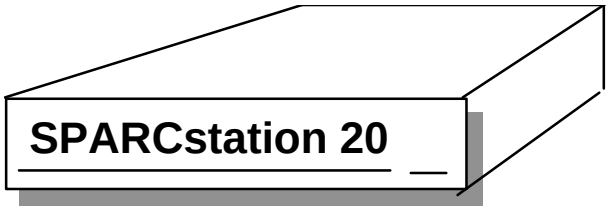
Prinzipieller Aufbau eines Rechners

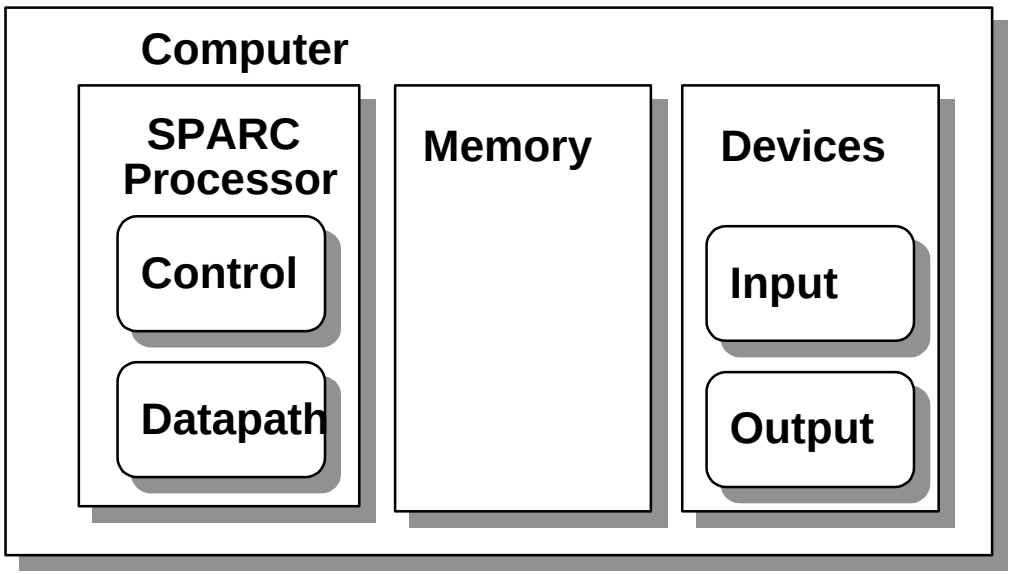


Bestandteile:

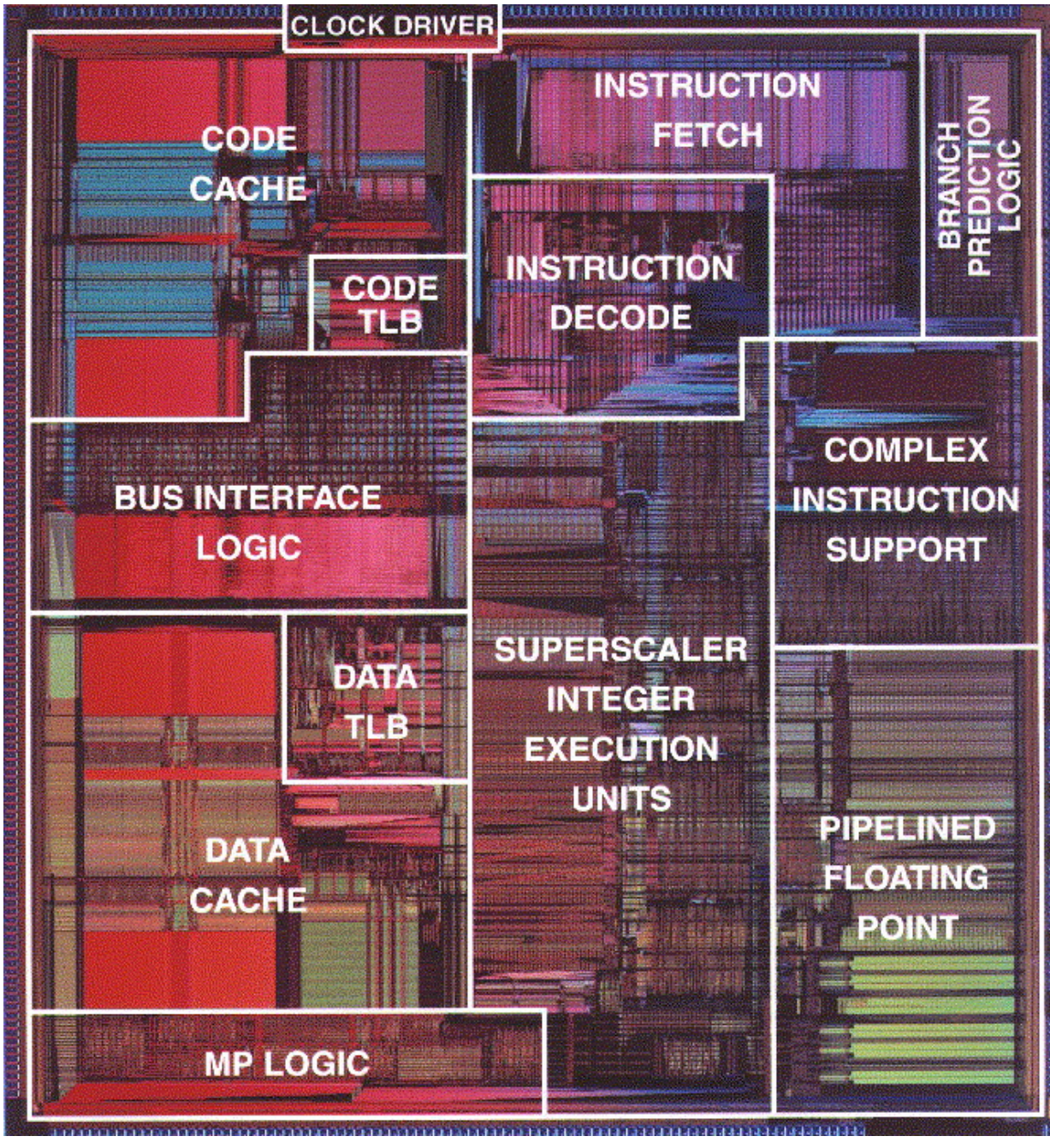
- ⌘ Prozessor mit **Cache**
- ⌘ **Hauptspeicher**
- ⌘ **Externe Speicher**
- ⌘ Eingabegeräte (Tastatur, Maus)
- ⌘ Ausgabegeräte (Bildschirm, Drucker, Plotter)
- ⌘ Busse

SUN SPARCstation 20

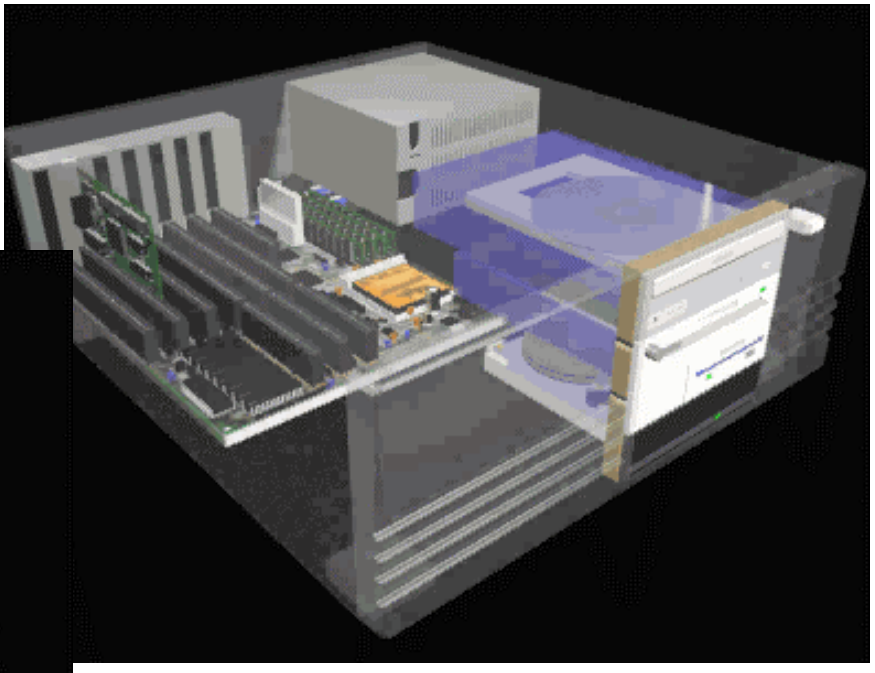




Cache

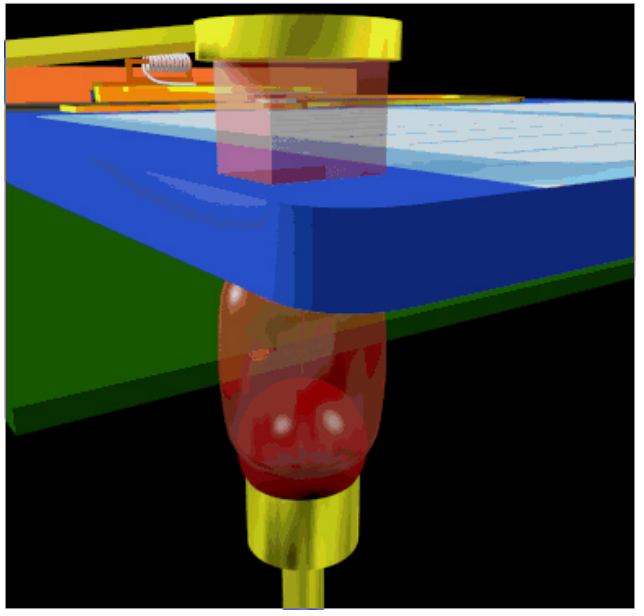

CLOCK DRIVER
CODE CACHE
INSTRUCTION FETCH
BRANCH PREDICTION LOGIC
CODE TLB
INSTRUCTION DECODE
COMPLEX INSTRUCTION SUPPORT
BUS INTERFACE LOGIC
DATA TLB
SUPERSCALER INTEGER EXECUTION UNITS
DATA CACHE
PIPELINED FLOATING POINT
MP LOGIC

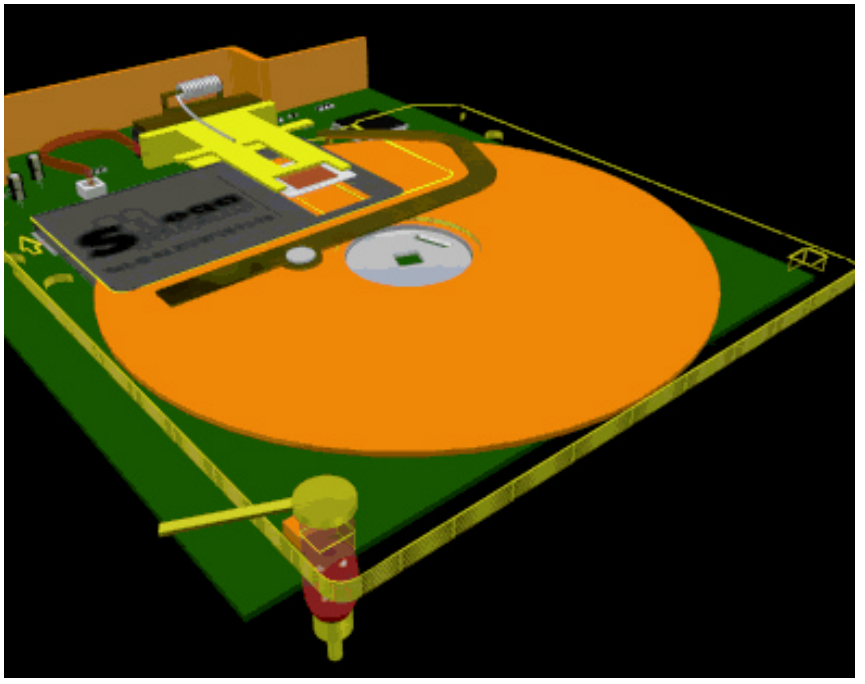
Hauptspeicher



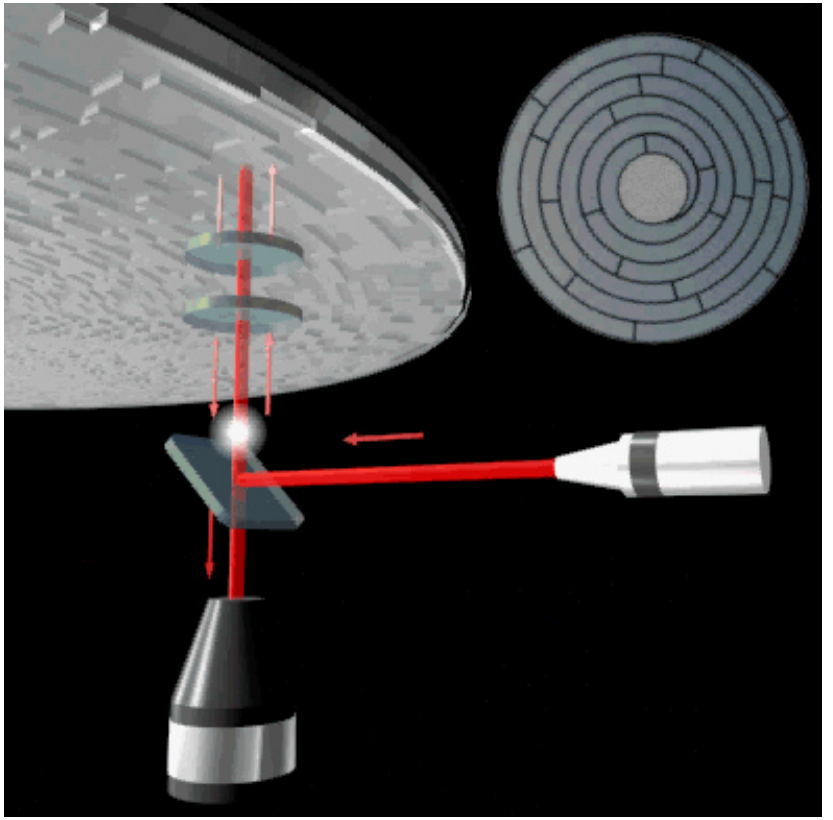
Festplatte

Floppy





CD-ROM / DVD



Speicher

- Ansammlung von Speicherzellen
- Speicherzelle:
  - speichert Information aus k Bits
  - k ist fest und heißt Wortbreite
  - kann über Adresse angesprochen werden
  - ist kleinste adressierbare Einheit

Speicher (2)

- Information:
  - wird durch elektrische Spannung dargestellt
  - Spannung hat unendlich viele Werte
  - digitale Rechner: unterscheidet nur zwischen großer und kleiner Spannung

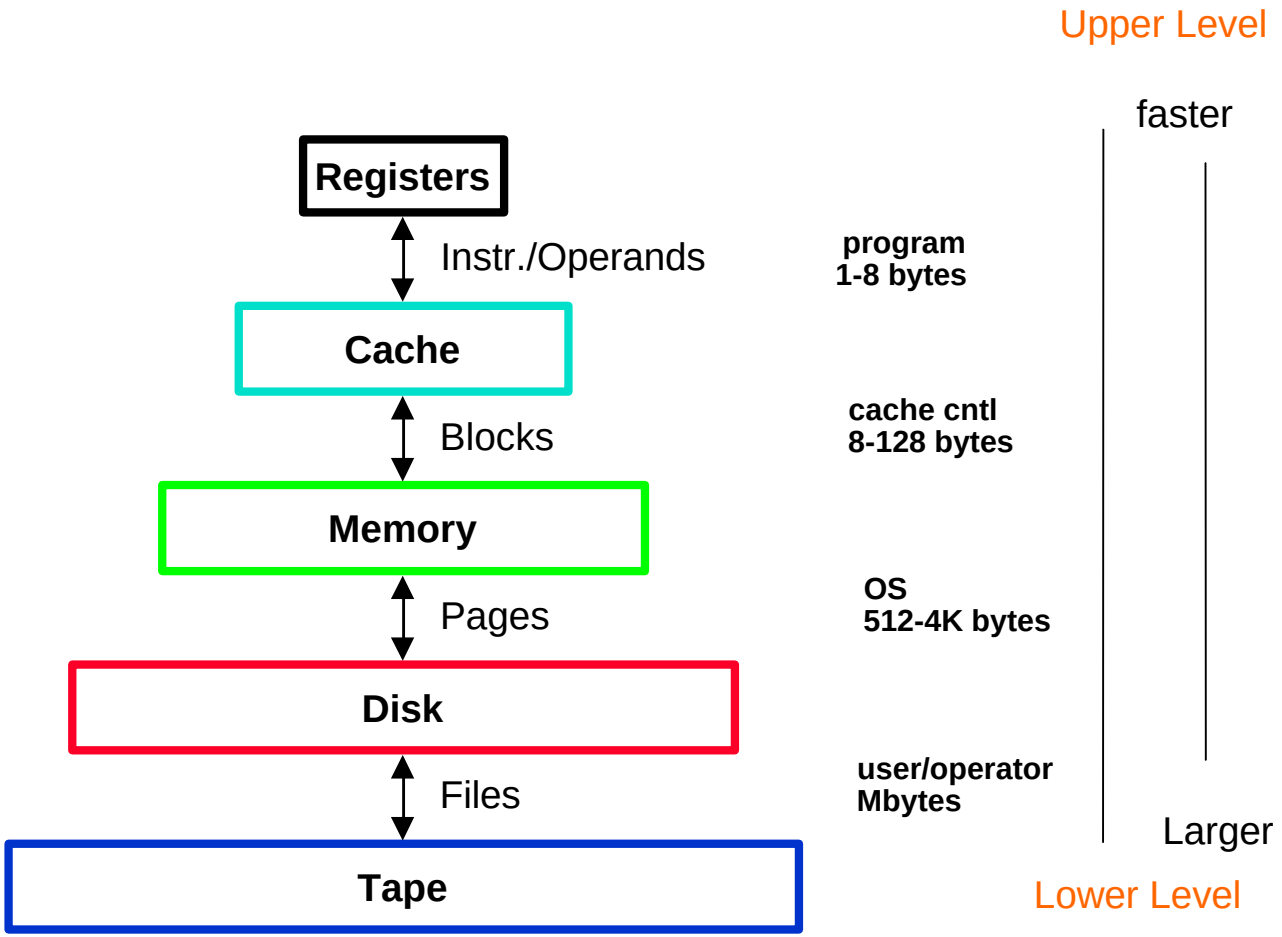
Speicher (3)

- Bit:
  - kleinste Informationsmenge
  - kann zwei verschiedene Werte speichern
  - enthält 0 oder 1
  - Wert ist durch angelegte Spannung definiert
- m Bits für Adresse
  - 2^m adressierbare Speicherzellen

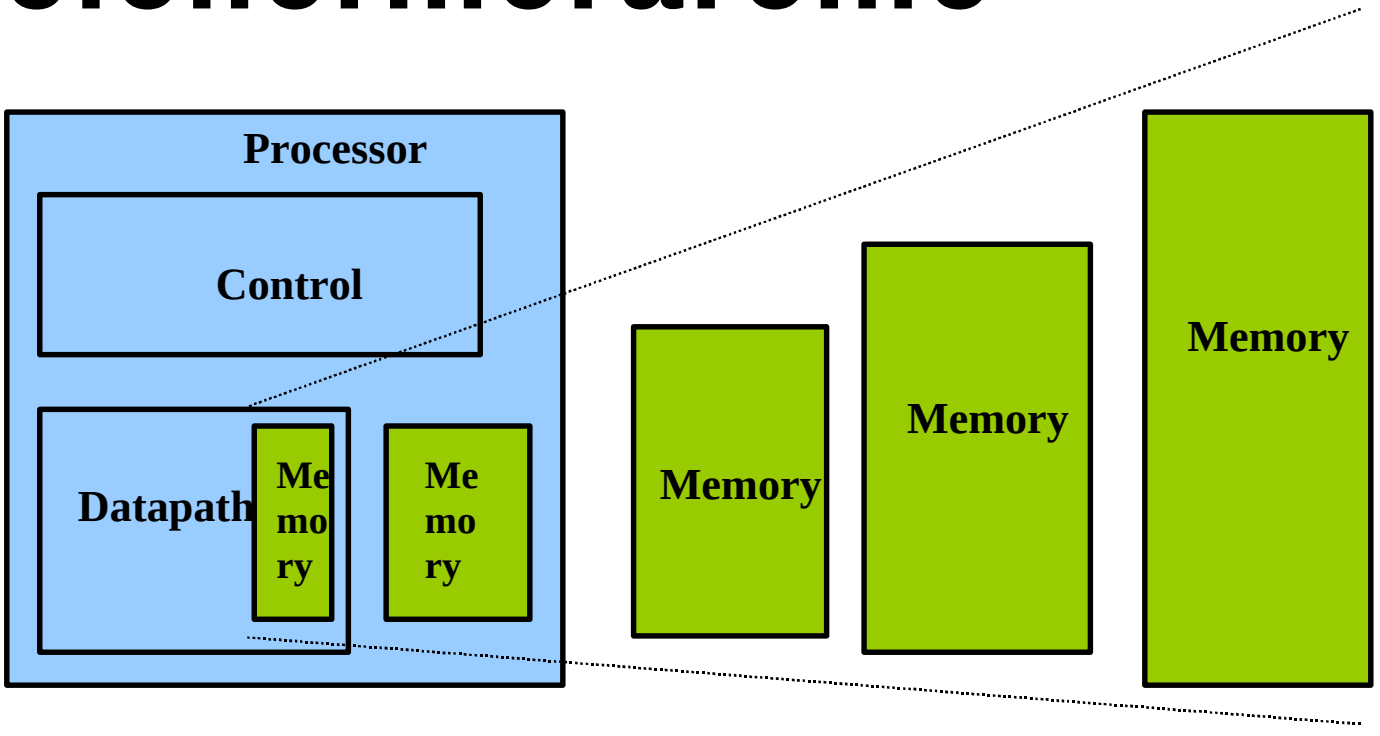
Speicher (4)

- **Kapazität**
  - Anzahl der Speicherzellen
  - Speicherwortbreite (z.B. 4 Byte=32 Bit)
- Aktivierung eines Speicherbausteins durch ‘Speichermodul-Auswahl-Signal’ (CS=chip select)
  - Bei Schreib-Lese-Speichern gibt es Schreib-Lese-Signal (R/W=read/write)
  - Zugriffszeit beim Lesen und Schreiben nennt man auch Speicherzyklus
- Handel Speicherzyklus <--> Preis

Speicherorganisation heute



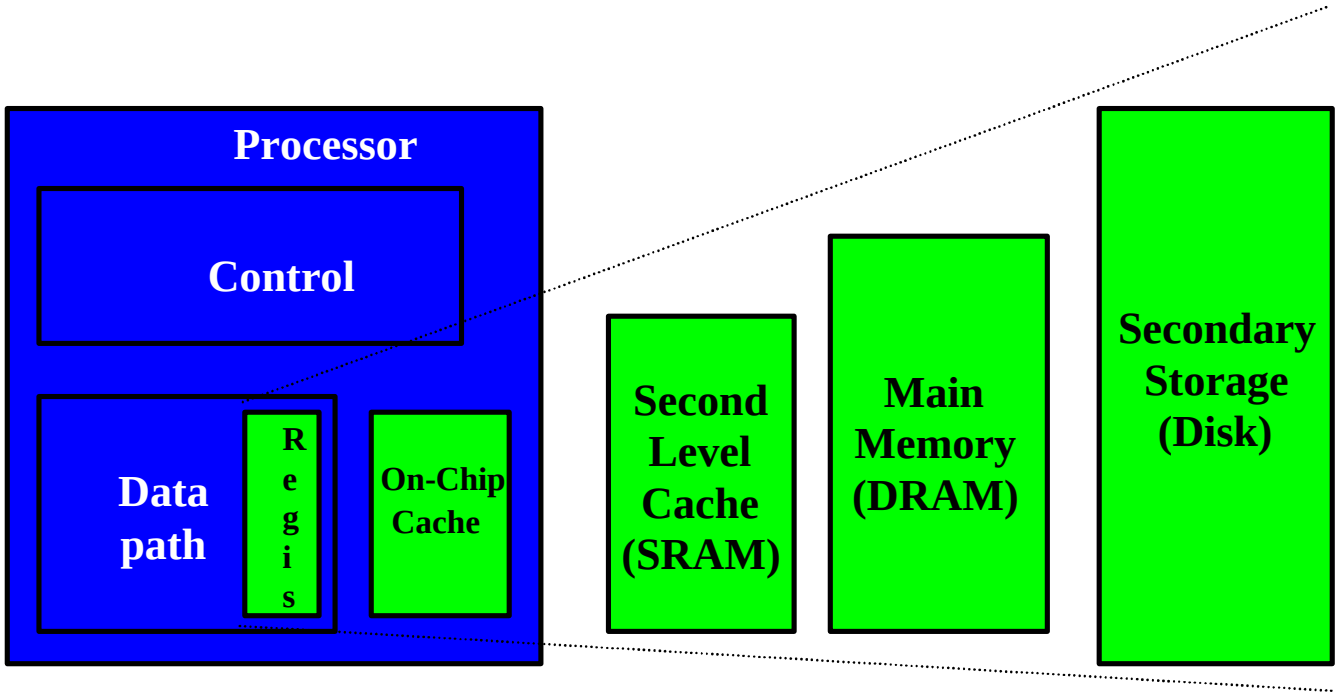
Speicherhierarchie



Speed: Fastest
Size: Smallest
Cost: Highest

Slowest
Biggest
Lowest

Speicherhierarchie



Speed:	1 ns	1-3 ns	10 ns	10 ms
Size (bytes):	100	Kilo	Mega	Giga

Fragen

- Wieso ist ein Zugriff auf eine Hauptspeicherzelle langsamer als ein Zugriff auf ein Register ?
- Wie kann verhindert werden, dass ein Rechner durch Hauptspeicherzugriffe zu sehr verlangsamt wird ?
- Wieso stellt man dem Prozessor nicht einfach einige Mbyte Register zur Verfügung ?
- Wie kommen „Massendaten“ in den Arbeitspeicher?

Gliederung

- **4.1** Speicherhierarchie
- **4.2** Arbeitsspeicher (Halbleiterspeicher)
  - Speicherhardware
  - Caches
  - Virtuelle Speicher
- **4.2** Massendaten-Speicher
  - Festplatte und Floppy
  - CDROM/DVD
  - Magnetband

4.1 Arbeitspeicher

- Speicherhardware

- Zwei verschiedene Speicherarten:

  - **Festwertspeicher** (ROM=read only memory)

  - **Schreib-Lese-Speicher** (RAM=random access memory)

Festwertspeicher (1)

- „Nur-Lese-Speicher“
- Wird vom Halbleiterhersteller nach Maßgabe des Kunden *irreversibel* programmiert

- EPROM=Erasable Programmable ROM kann durch Einstrahlung von UV-Licht gelöscht und danach *neu* programmiert werden (im Betrieb nur Lesen)

Festwertspeicher (2)

- EEPROM=Electrically Erasable Programmable ROM
  - Löschen geschieht auf *elektrischem* Wege
- weitere Varianten: Flash, NVRAMs

- Festwertspeicher beinhalten oft Initialisierungs- und Startprogramme (z.B. Selbsttest des Systems, Laden des Betriebssystems)

Schreib-Lese-Speicher

- Halten die Information, solange Versorgungsspannung anliegt
- Statische Speicher (SRAM=static RAM)
  - Speicherzelle aus regulären Flip-Flops
  - 6 Transistoren
- Dynamische Speicher (DRAM=dynamic RAM)
  - 1 Transistor
  - stellt durch Ladung Information dar ($\rightarrow$ Kondensator)
  - ‘refresh’ ist notwendig
  - Moderne DRAMs haben Kapazität von einigen Mega-Bit (SRAMs und ROMs vergleichbarer Grö gibt es nicht)

Grund für komplexe Speicherorganisation

- **Wieso ist ein Zugriff auf eine Hauptspeicherzelle langsame als ein Zugriff auf ein Register ?**
  - Hauptspeicherzellen sind DRAM-Zellen.
    während Register in der Regel SRAM-Zellen sind !
  - Bei einem Registerzugriff kommt man ohne Bus-Operation aus !
- **Wieso stellt man dem Prozessor nicht einfach einige Mbyte Register zur Verfügung ?**
  - SRAM-Zellen sind wesentlich größer als DRAM-Zellen (Faktor >=4)

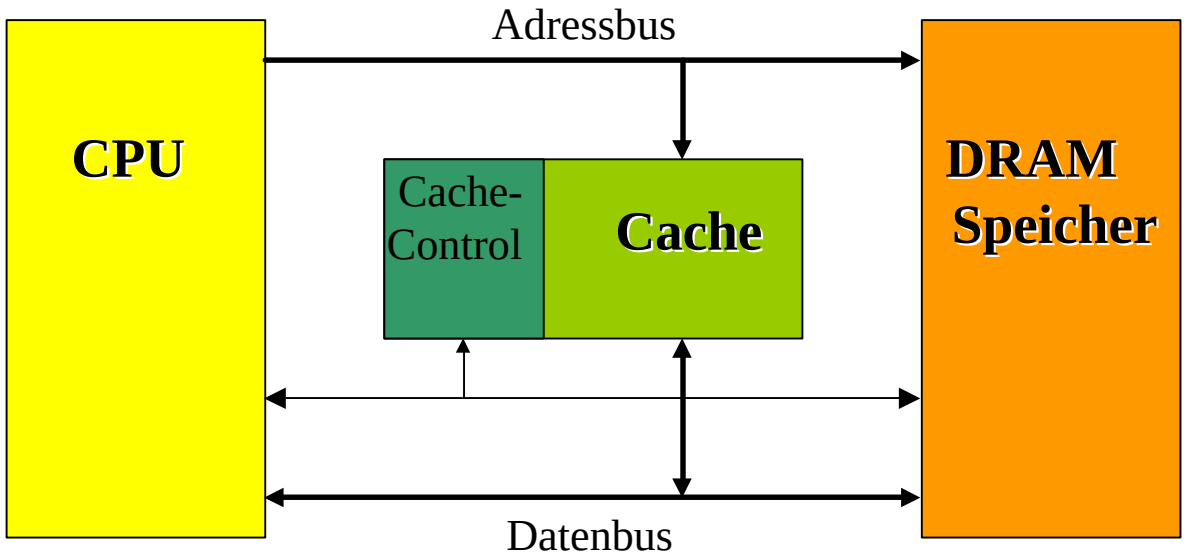
So abwegig ist die Idee nicht !

Mit der weiteren Technologieentwicklung (noch kleinere Strukturen wird die verfügbare Chip-Fläche vorwiegend dazu benutzt werden, um schnellen Speicher zu integrieren.

Cache

- Cache kommt aus dem Französischen: **cacher** (verstecken)
- Er kann durch ein Anwendungsprogramm nicht explizit adressiert werden
- Er ist **software-transparent**, d.h. der Benutzer braucht nichts von seiner Existenz zu wissen.

Lage des Caches



In der Regel besitzt ein Rechner einen getrennten Cache für Instruktionen (**Instruktionscache**) und für Daten (**Datencache**)

Ziel des Cache - Einsatzes

- Versuche stets die Daten im Cache zu halten, die als nächstes gebraucht werden

  - ➔ der Prozessor kann die Mehrzahl der Zugriffe auf dem Cache und nicht auf dem langsamen DRAM Speicher ausführen.

- Voraussetzung, um dieses Ziel erreichen zu können: **Lokalitätsprinzip**

  - d.h. zu jedem Zeitpunkt während eines Programmablaufs werden bestimmte Speicherzellen bevorzugt und wiederholt angesprochen (siehe z.B. Schleifen)

Prinzipielle Funktionsweise: Lesezugriff

Lese Datum aus dem Arbeitsspeicher unter Adresse *a*

CPU überprüft, ob eine Kopie der Hauptspeicherzelle *a* im Cache abgelegt ist

- Falls ja (**cache hit**)
  - so entnimmt die CPU das Datum aus dem Cache. Die Überprüfung und das eigentliche Lesen aus dem Cache erfolgt in einem Zyklus, ohne einen Wartezyklus einfügen zu müssen.
- Falls nein (**cache miss**),
  - so greift die CPU auf den Arbeitsspeicher zu
  - lädt das Datum in den Cache und
  - lädt das Datum gleichzeitig in die CPU.
  - **Anmerkung**: Insbesondere bei Großrechnern wird mit jedem Datum auch dessen umgebender Block von Daten geladen in der Erwartung, daß folgende Zugriffe auf diese Daten erfolgen (hier

Illustration des Lesezugriffs: cache hit

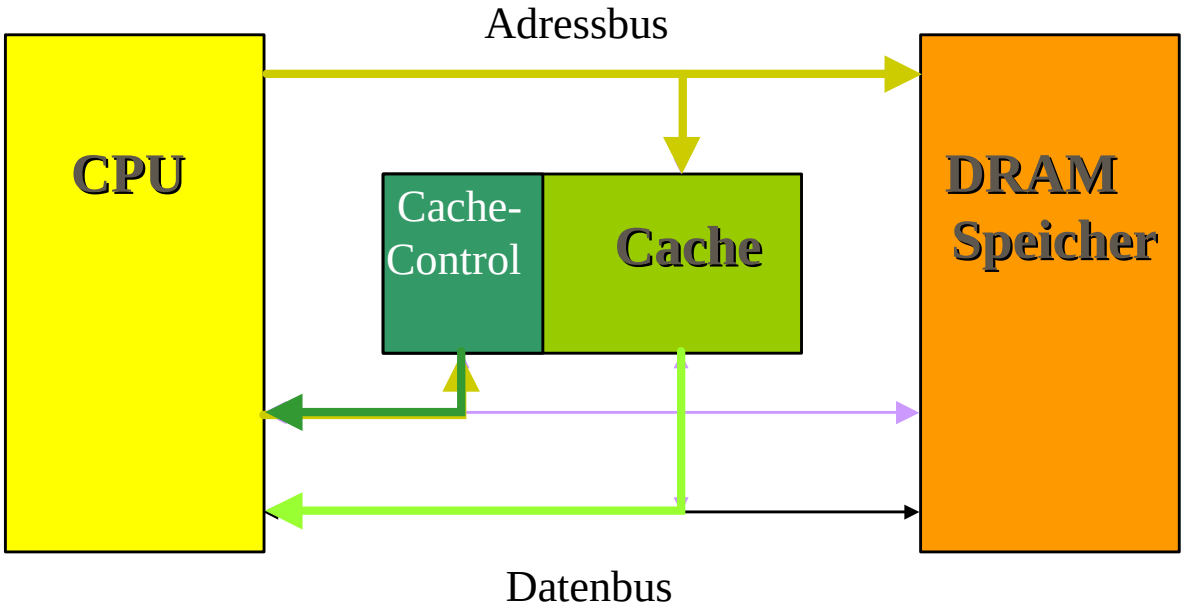
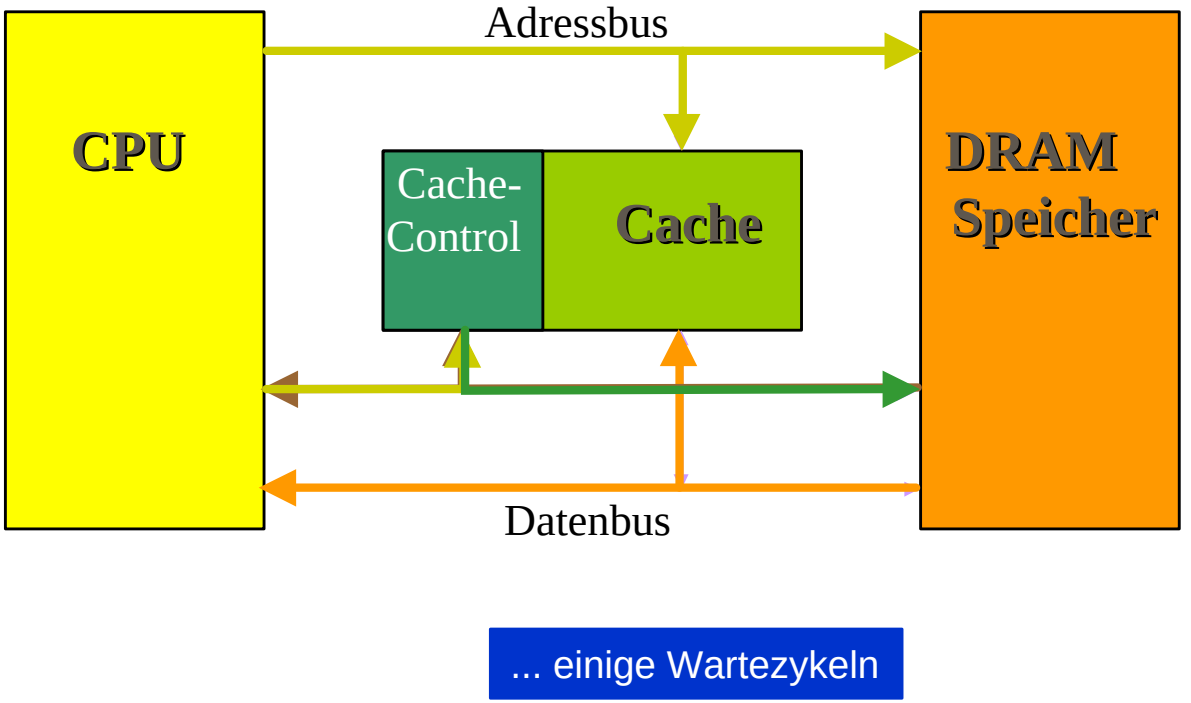


Illustration des Lesezugriffs: cache miss



Mittlere Zugriffszeit beim Lesen

- c : Zugriffszeit des Caches
- m : Zugriffszeit beim Hauptspeicher
- h : Trefferrate

➔ durchschnittliche Zugriffszeit: $c+(1-h)\times m$

Rechenbeispiel für c=50 ns und m=200 ns

Trefferrate h	∅–Zugriffszeit
50%	150 ns
60%	130 ns
70%	110 ns
80%	90 ns
90%	70 ns
95%	60 ns

Verdrängen alter Daten aus dem Cache

Szenario

- cache miss
- alle Speicherbereiche des Caches belegt

Ausweg

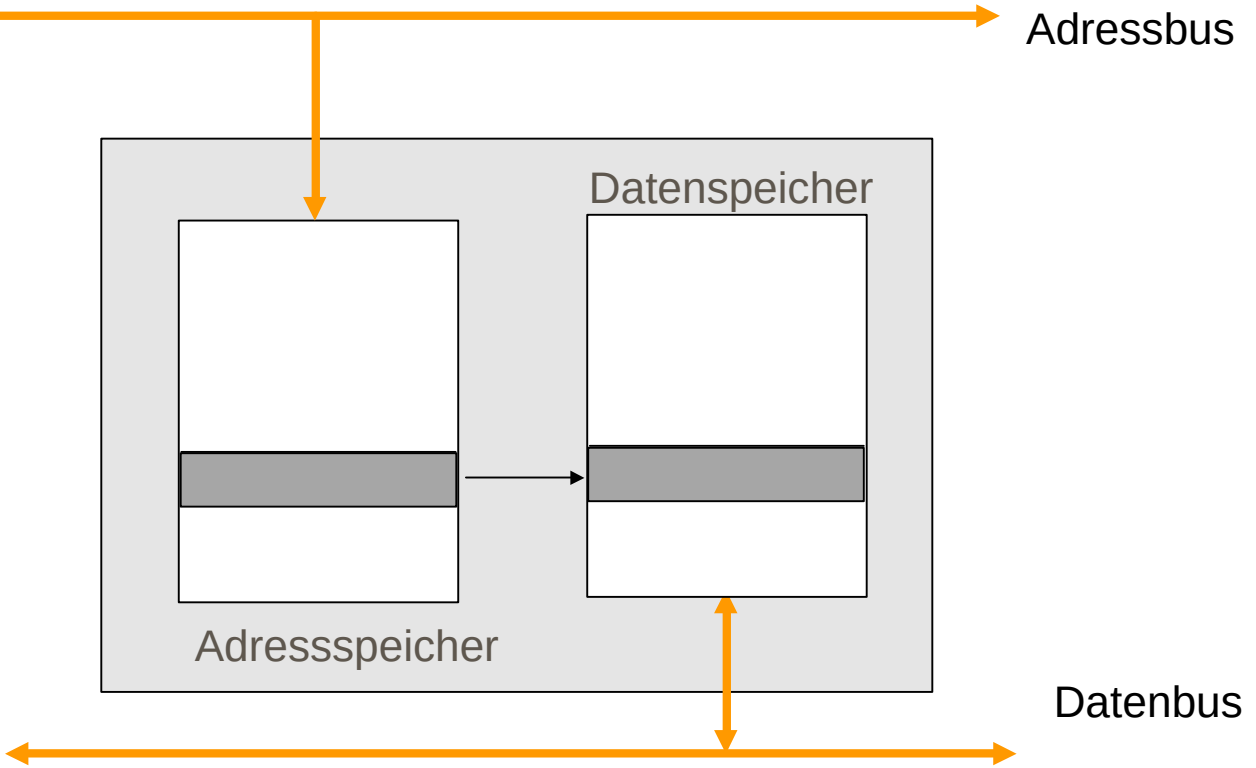
- verdränge Datum (Block) aus dem Cache
- lade an seine Stelle das gerade benötigte Datum (Block)

Denkbare Verdrängungsstrategien

- least recently used
  verdränge das Datum (Block), das am längsten nicht benutzt wurde
- least frequently used
  verdränge das Datum (Block), auf das am wenigsten zugegriffen wurde
- first in, first out
  verdränge das Datum (Block), das am längsten im Cache ist

Aufbau eines Caches

- Ein Cache besteht aus zwei Speicher-Einheiten, die wortweise einander fest zugeordnet sind.



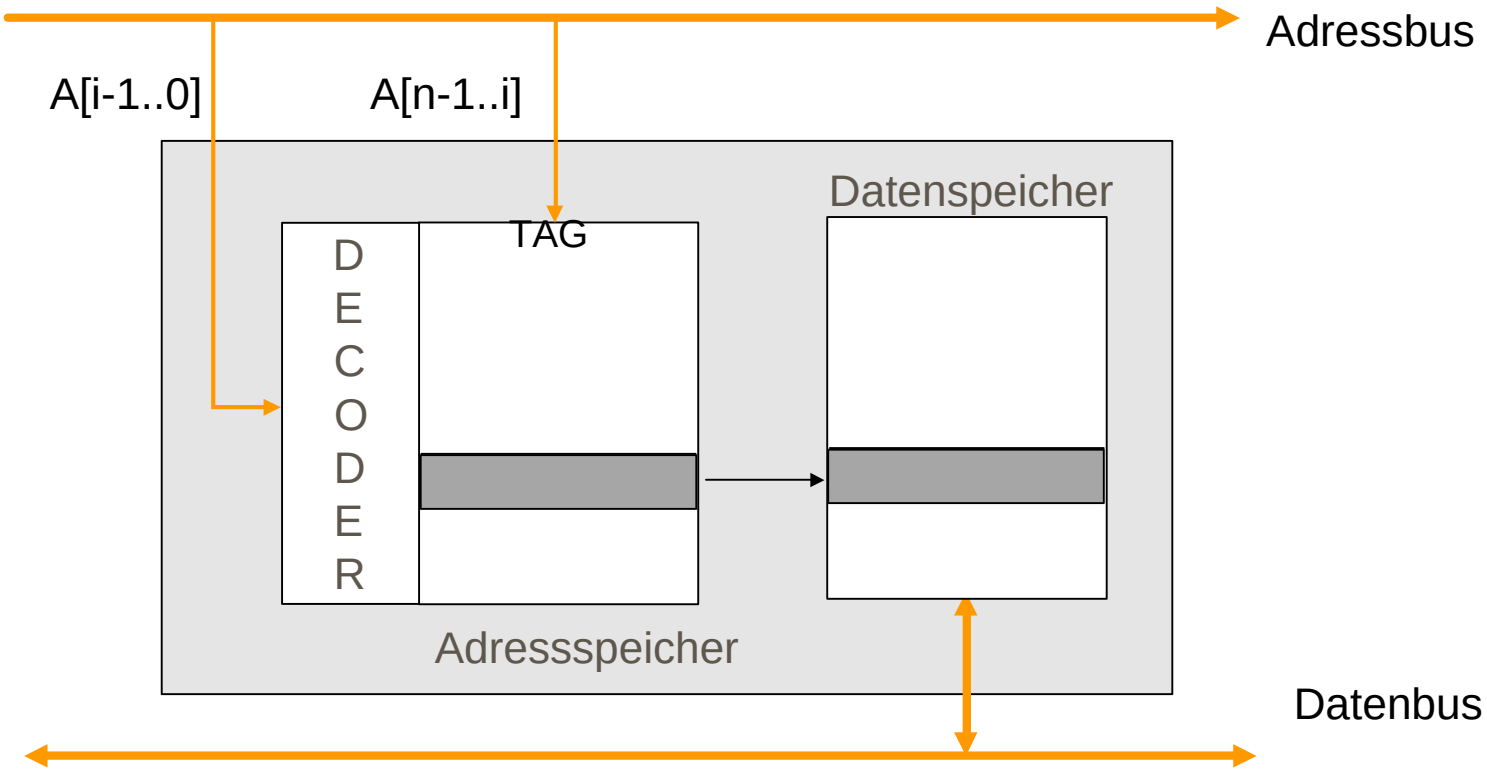
Cache als assoziativer Speicher

- **Idealfall**: assoziativer (inhaltsorientierter) Speicher
- Die von der CPU angelegte Adresse wird **parallel** mit allen im Adressspeicher des Caches vorhandenen Adressen verglichen.
- Außerdem kann ein neues Datum an jeder beliebigen freien Stelle im Cache abgelegt werden.

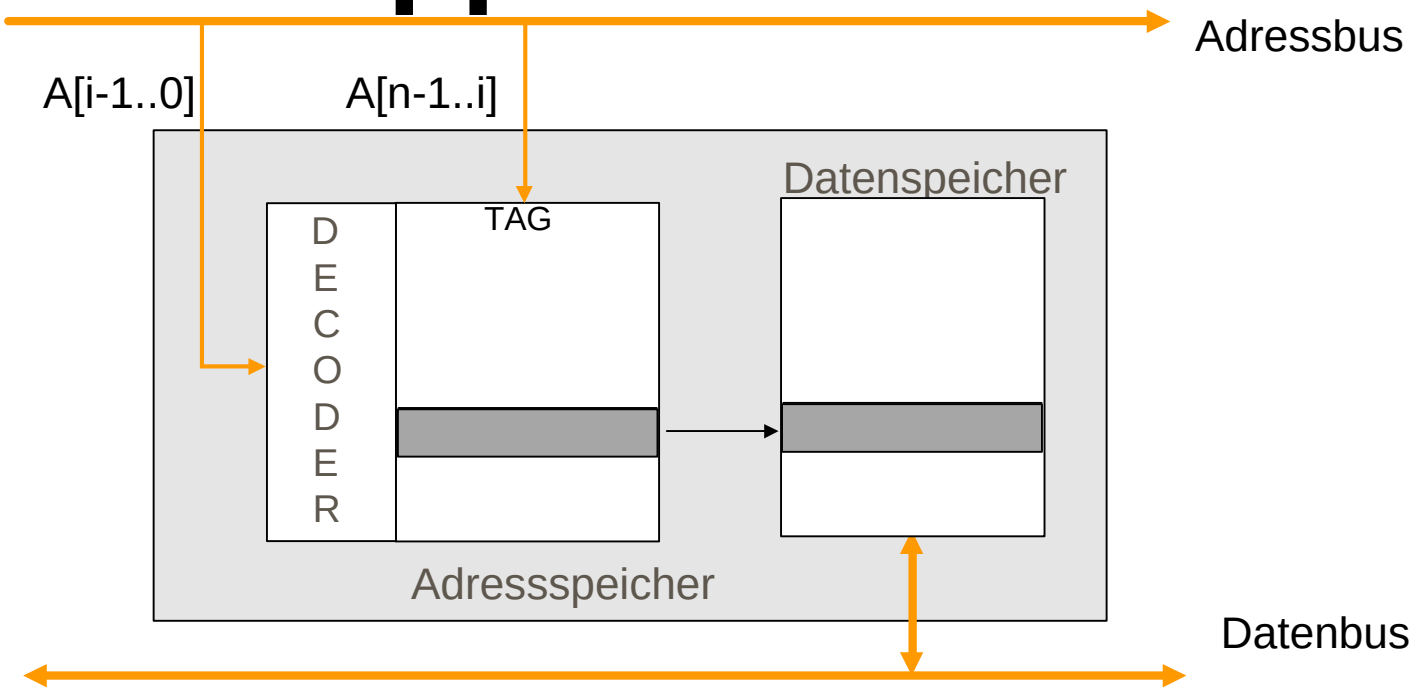
➔ Aufwendige Logik für den parallelen Vergleich!
➔ Assoziative Speicher nur für kleine Cache-Größen.

Direct Mapped Cache

- **Feste Abbildung** der Hauptspeicher-Adressen auf die Cache-Adressen
  - ➔ Kein assoziativer Speicher nötig
  - ➔ Keine Verdrängungsstrategien nötig

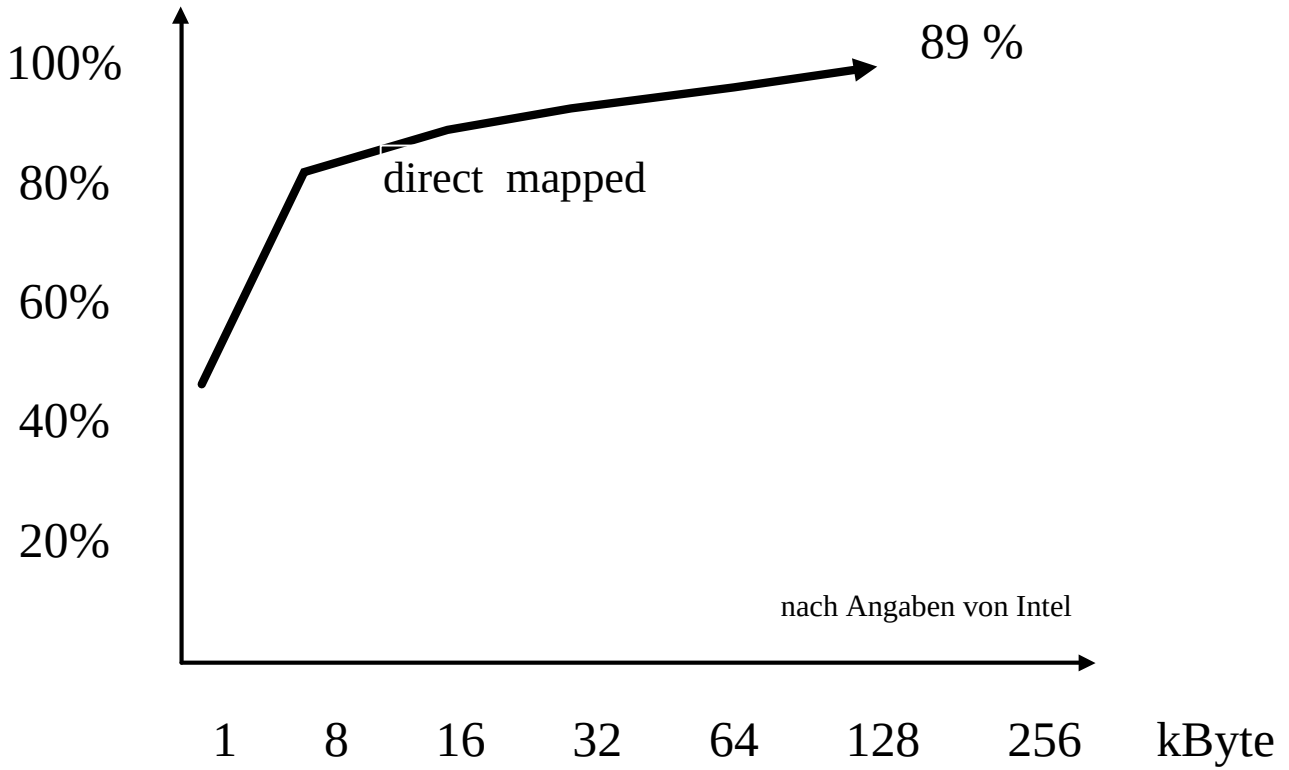


Direct Mapped Cache



- Die von der CPU angelegte Adresse wird in 2 Teile gespalten.
  - Die *i* niederwertigsten Bits adressieren einen Eintrag im Adressspeicher.
  - Dieser Eintrag wird mit den *n-i* höherwertigen Bits der Adresse verglichen.
- Speicherzellen des Hauptspeichers, deren *i* niederwertigste Stellen gleich sind, werden auf die gleiche Position im Cache abgebildet.

Direct Mapped Cache: Trefferquote



Prinzipielle Funktionsweise: Schreibzugriff (write-through-Verfahren)

Schreibe Datum in den Arbeitsspeicher unter Adresse *a:*

CPU überprüft, ob eine Kopie der Hauptspeicherzelle *a* im Cache abgelegt ist

- **write-through Verfahren**:
  - **cache miss:** CPU schreibt das Datum in die Hauptspeicherzelle mit Adresse *a*. Der Inhalt des Cache wird nicht verändert.
  - **cache hit:** die Kopie der Hauptspeicherzelle im Cache wird aktualisiert, die Hauptspeicherzelle selbst wird sofort aktualisiert

Prinzipielle Funktionsweise: Schreibzugriff (write-back-Verfahren)

Schreibe Datum in den Arbeitsspeicher unter Adresse *a:*

CPU überprüft, ob eine Kopie der Hauptspeicherzelle *a* im Cache abgelegt ist

- **write-back Verfahren**:
  - **cache miss:** CPU schreibt das Datum in die Hauptspeicherzelle mit Adresse *a*. Der Inhalt des Cache wird nicht verändert.
  - **cache hit:** die Kopie der Hauptspeicherzelle im Cache wird aktualisiert und durch „dirty bit“ als verändert markiert, die Hauptspeicherzelle selbst wird erst später aktualisiert, nämlich wenn die Kopie aus dem Cache verdrängt wird

Prinzipielle Funktions-weise: Schreibzugriff (write-allocation-Verfahren)

Schreibe Datum in den Arbeitsspeicher unter Adresse *a:*

CPU überprüft, ob eine Kopie der Hauptspeicherzelle *a* im Cache abgelegt ist

- **write-allocation Verfahren**:
  - **cache miss:** CPU schreibt Datum in den Cache (markiert mit „dirty bit“), aber nicht in den Hauptspeicher (dort erst aktualisiert, wenn Kopie aus dem Cache verdrängt).
  - **cache hit:** die Kopie der Hauptspeicherzelle im Cache wird aktualisiert (markiert durch „dirty bit“), die Hauptspeicherzelle selbst wird erst später aktualisiert, nämlich wenn die Kopie aus dem Cache verdrängt wird

Write-through versus Write-back/Write-alloaction

Vorteile von Write-back / Write-alloaction

- Schreibzugriffe auf Cache bei cache hit ohne Wartezyklus möglich (bei Write-allocation sogar bei cache miss)
- Belastung des Systembusses kleiner, wenn das Rückschreiben in den Hauptspeicher erst nach mehreren Schreibvorgängen erfolgen muss

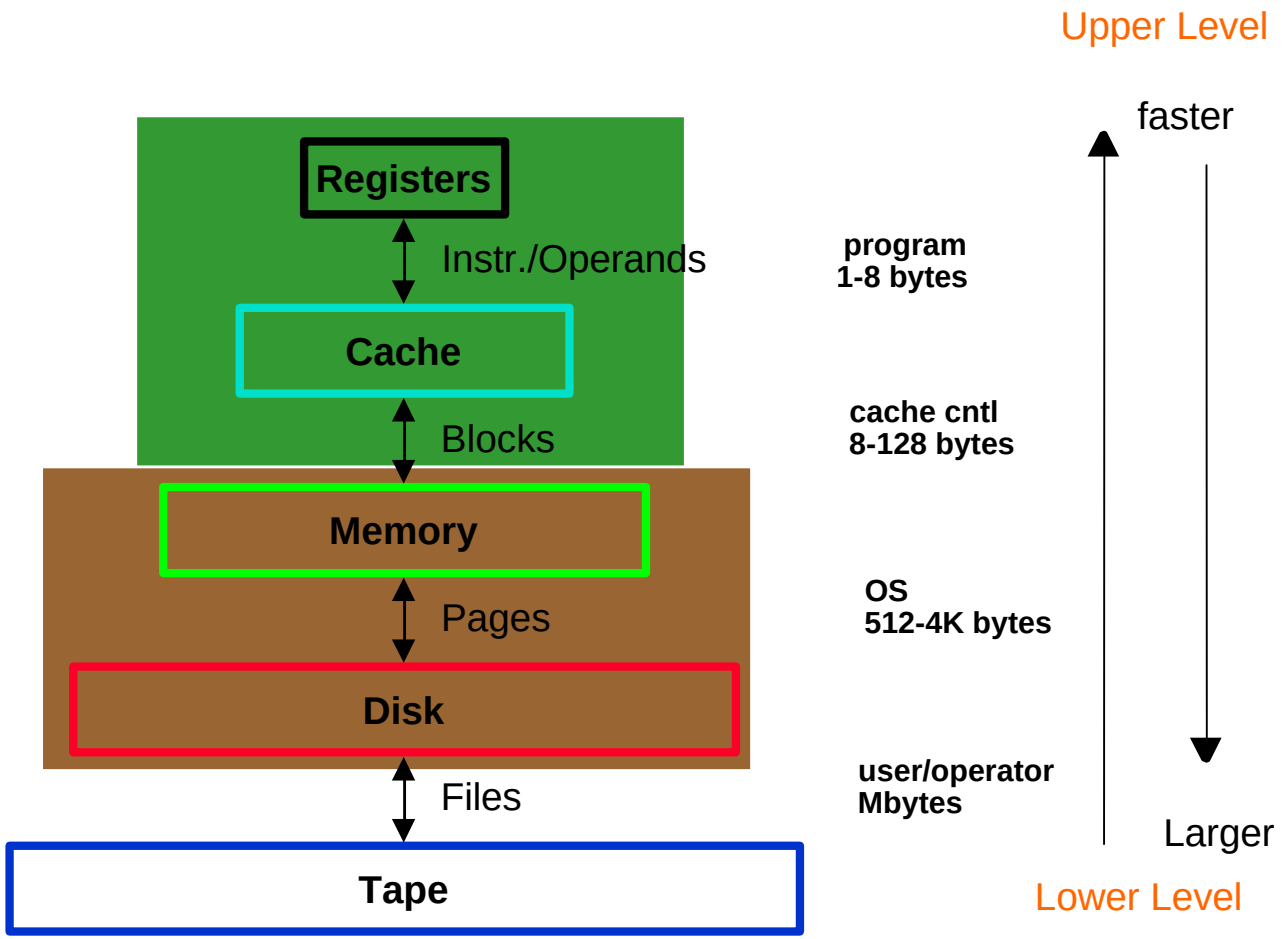
Nachteile von Write-back / Write-alloaction

Schwierigkeit bei der Datenkonsistenz, wenn andere Komponenten auf den Hauptspeicher zugreifen können, z.B. ein DMA-Controller oder ein zweiter Prozessor in einer Shared Memory Machine. Zu vermeiden ist, dass die anderen Module veraltete Werte vorfinden oder der Cache mit veralteten Werten rechnet

Beispiele

- **MIPS R3000**
  - Befehls- und Datencache getrennt
  - write through
  - Blockgröße einstellbar (4 bis 32 Worte)
- **SUN Sparc**
  - write back und write through (wahlweise)
- **Intel Pentium**
  - zwei ‚on-chip‘ Caches

Virtuelle Speicher



Das Problem mit dem Hauptspeicher

- Adressraum von heutigen Rechnern ist sehr groß !
- Bei Busbreite *n* sind 2^n Speicherzellen adressierbar !

- ... soviel Hauptspeicher kann man nicht bereitstellen !

16	65536	--
32	$4{,}3 \cdot 10^9$	13.000 DM
64	$1{,}8 \cdot 10^{19}$	$5 \cdot 10^{13}$ DM

Benutze Festplatte als virtuellen Speicher

Benutzersicht

- Programm und alle Daten befinden sich im Hauptspeicher, sofern der Adressraum nicht ausgeschöpft ist

Wirklichkeit

- multi-user System, d.h. nicht jedem Benutzer kann der ganze Hauptspeicher zur Verfügung stehen.
- Programme werden nicht für einen spezifischen 32-Bit Rechn mit maximaler Hauptspeichergröße geschrieben. Sie sollen au auf 32-Bit Rechner mit kleinerem Hauptspeicher lauffähig sei

Ausweg

- Benutze die Festplatte, um die "Hauptspeicherdaten" eines Prozesses zu speichern, die aufgrund von Kapazitätsgründen nicht im vorhandenen Hauptspeicher liegen können,

Verwaltung des virtuellen Adressraumes

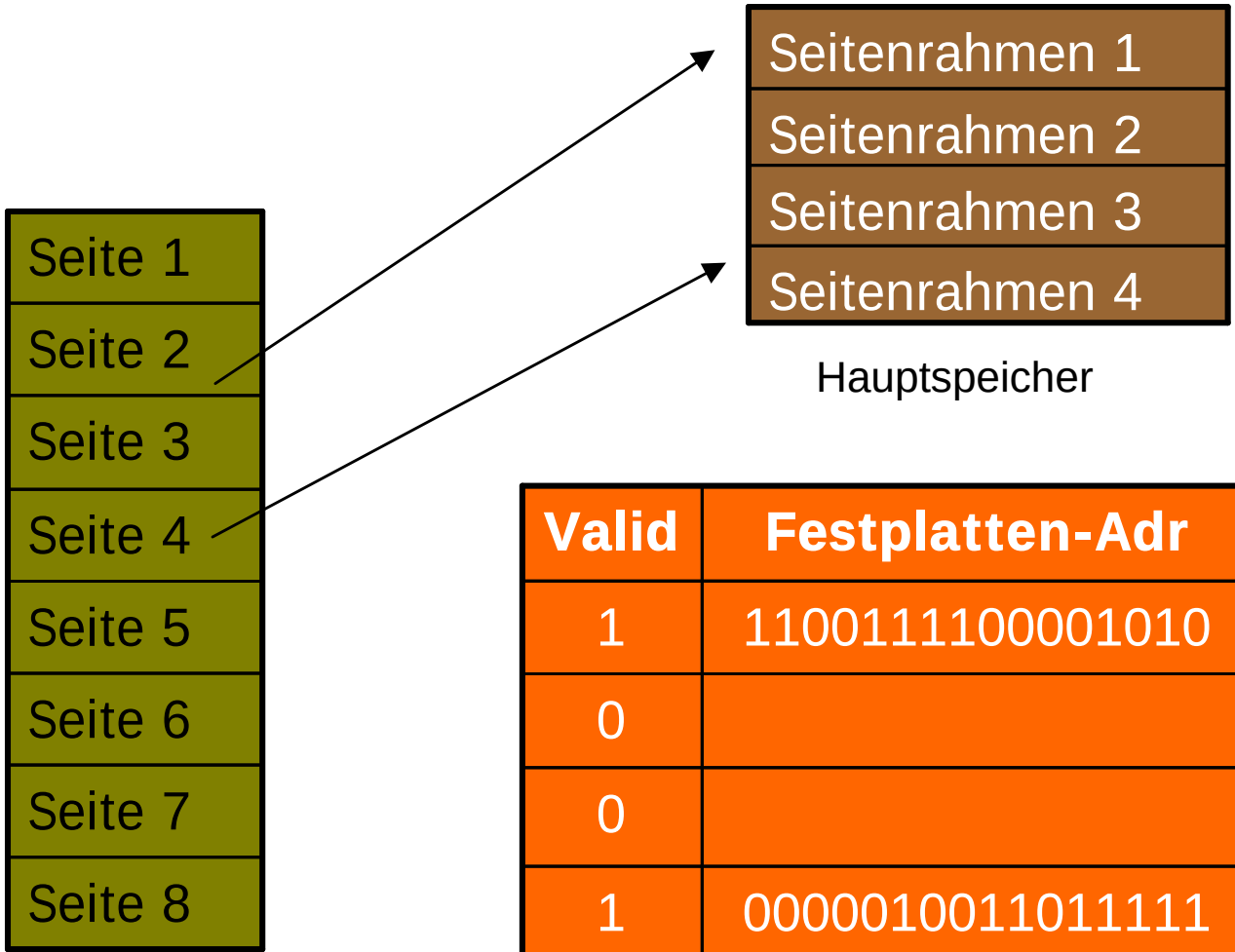
- Jedem Prozess steht ein virtueller Adressraum zur Verfügung
- Diese virtuellen Adressräume werden von dem Betriebssystem verwaltet:
  - Welche Daten bzw. Programmteile werden im Hauptspeicher gehalten ?
  - Welche Daten werden ausgelagert, wenn neue Daten benötigt werden ?
- In diesem Zusammenhang werden die Konzepte
  - **Paging**
  - **Segmentierung**

Paging

Grundidee

- Der virtuelle Speicher wird auf dem Sekundärspeicher abgelegt
- ... und in **Seiten** (**pages**) fester Größe unterteilt.
- Der Hauptspeicher besteht aus **Seitenrahmen** (**page frames**), die jeweils eine Seite aufnehmen kann
- Eine **Seitentabelle** (**page table**) gibt an, welche Seitenrahmen durch welche Seiten belegt sind.

Paging



Valid	Festplatten-Adr	Seitennr
1	1100111100001010	011
0		
0		
1	0000010011011111	101

Seitentabelle (liegt im Hauptspeicher)

Paging: Zugriff auf Datenseite i

Überprüfe, ob die Datenseite *i* im Hauptspeicher liegt;

if Seitenfehler
then Überprüfe, ob ein Seitenrahmen im Hauptspeicher leer ist;
    if kein Seitenrahmen leer
    then Verdränge eine Seite aus dem Hauptspeicher und
        aktualisiere die Seitentabelle;
     fi
    Schreibe die Datenseite *i* in einen freien Seitenrahmen;
    Aktualisiere
fi

Greife auf Seite *i* im Hauptspeicher zu;

Verdrängungsstrategien

- **LFU-Strategie** **L**east **F**requently **U**sed

  Verdränge die Seite aus dem Hauptspeicher, auf die seit ihrer Einlagerungszeit am seltensten zugegriffen wurde

- **LRU-Strategie** **L**east **R**ecently **U**sed

  Verdränge die Seite, auf die am längsten nicht zugegriffen wurde.

- **FIFO-Strategie** (**F**irst **I**n, **F**irst **O**ut)

  Verdränge die Seite, die am längsten im Hauptspeicher liegt

Kleiner Verwaltungsaufwand, da nur bei einem Seitenfehler etwas getan werden muss.

Segmentierung

Unterteilung des virtuellen Speichers in Segmente unterschiedlicher Größen

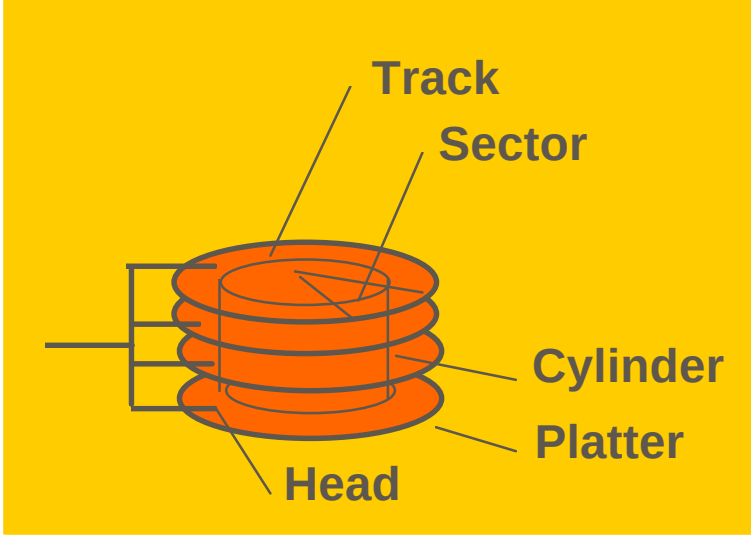
- Zum Beispiel ein Segment für
  - den Programmcode
  - den Stack
  - die statischen Variablen
  - ....
- Falls ein angefordertes Segment nicht im Hauptspeicher ist: **segment fault**
- Das Betriebssystem kann dafür Sorge tragen, dass bestimmte Segmente dauernd im Hauptspeicher liegen und nicht verdrängt werden können.

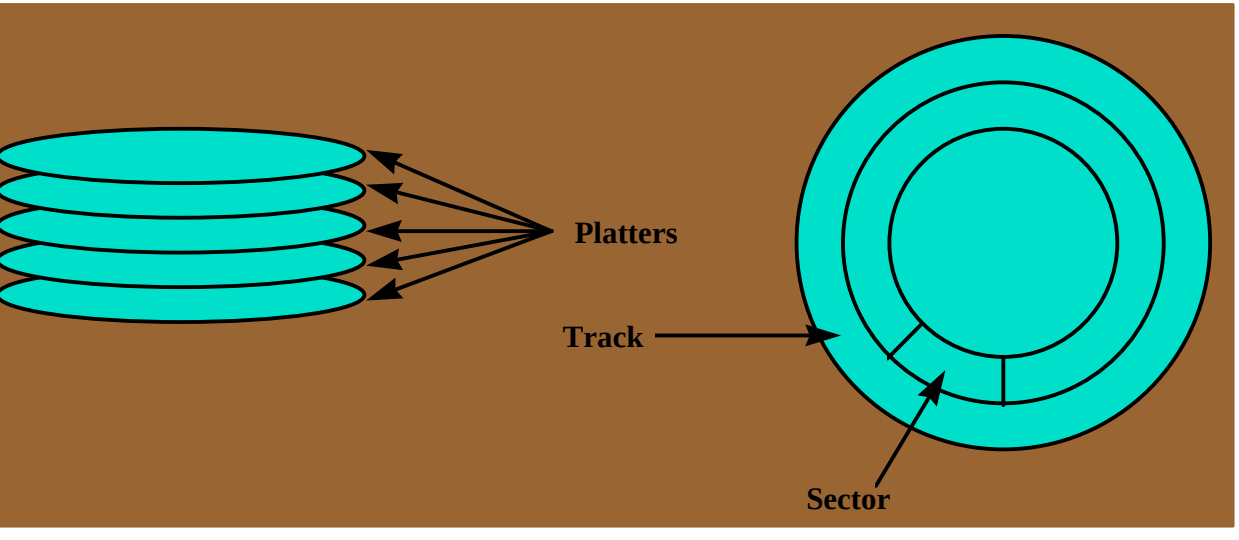
Kombination von Paging und Segmentierung

- Benutze Segmentierung
- Jedes Segment wird für sich mit Paging realisiert

Festplatte

Festplatte (Harddisc)





- Eine **Festplatte** bestel aus
  - mehreren Platten (4
  - mehreren Lese / Schreibköpfen (6-16
- Eine **Platte** besteht au konzentrischen Spuren
- Eine **Spur** besteht aus Sektoren
- Ein **Sektor** ist die klein beschreibbare Einheit
- Ein **Zylinder** besteht a übereinanderliegenden

tationsgeschwindigkeit: .600 rpm bis 10.800 rpm (**r**ounds **p**er **m**inute)
ktorgröße: 128 byte bis 1 kbyte
hreibdichte 50.000 bis 100.000 bits/cm
urendichte 800 bis 2.000 Spuren/cm

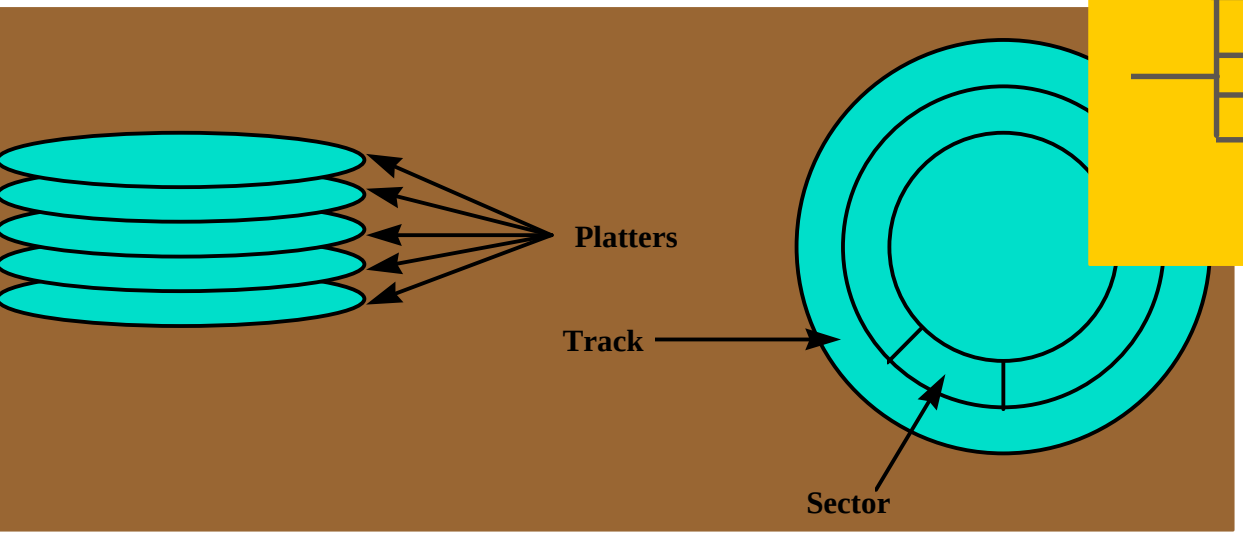
Ausführung eines Festplattenzugriffs

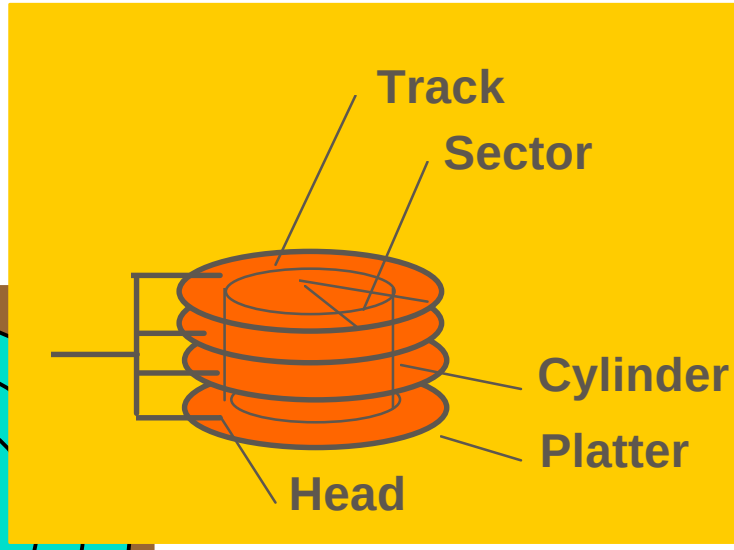
- Bewege die Köpfe zu dem richtigen Zylinder (Æ Dauer: ca. 10 msec)
- Warte bis der gesuchte Sektor zum Kopf rotiert (Æ Dauer: $0.5 \times rpm^{-1}$)
- Übertrage den Inhalt des Sektors (Transferrate: 5-20 Mbyte/sec)

Beispielrechnungen bei Sektorgrösse 512 Byte

vernachlässigbar

	10 msec	8 msec	102 µsec	18
	10 msec	6 msec	102 µsec	16
	10 msec	4 msec	102 µsec	14
	10 msec	2 msec	102 µsec	12
	10 msec	8 msec	25 µsec	18
	10 msec	6 msec	25 µsec	16
	10 msec	4 msec	25 µsec	14





Wie ist eine Festplatte logisch aufgebaut ?

D.h. wie werden die Files auf einer Festplatte abgelegt ?

Logischer Plattenaufbau bei MS-DOS

- den **Systembereich** bestehend aus
  - dem **Urladerbereich** (**Boot block**): Spur 0, Seite 0, Sektor 1
  - der **Dateizuordnungstabelle** (**File allocation block**): Adresse (0,0,2-4)
  - dem **Dateiverzeichnis** (**directory**): Adresse (0,0,5-9) und (0,1,1-2)
- den **Datenbereich**: ab Adresse (0,1,3)

(Spur 0 ist die äußere Spur einer Festplatte)

Urladerbereich bei MS-DOS

- besteht aus einem kurzen Programm (Länge kleiner als 512 byte), das nach dem Einschalten des Rechners den Prozess zum Laden des Betriebssystems in den Arbeitsspeicher aktiviert

Dateizuordnungstabelle bei MS-DOS

- Dateien werden in untereinander verketteten Sektoren gespeichert.

➔ zu jedem Sektor muss demnach
  - Zylindernummer
  - Seitennummer (d.h. welche Stapeloberfläche)
  - Sektornummer

  des nächsten zu dieser Datei gehörigen Sektors aufbewahrt werden.

- In der FAT (**F**ile **A**llocation **T**able) gibt es für jeden Sektor einen Eintrag, der
  - die Adresse des folgenden Sektors der Datei enthält, bzw.
  - die Information, dass der folgende Sektor nicht belegt ist.

Dateiverzeichnis bei MS-DOS

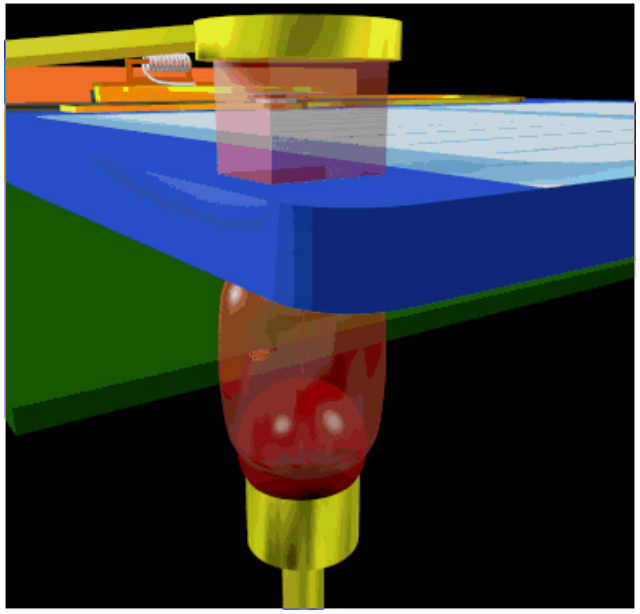
- Hier wird für jede Datei neben verschiedenen Attributen die Adresse des ersten Sektors der Datei hinterlegt.

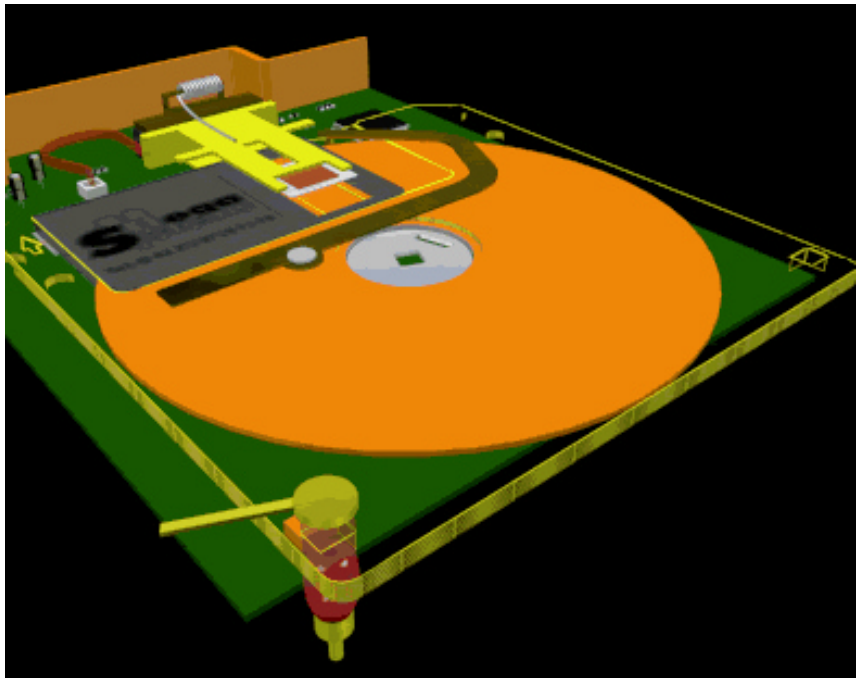
x00-x07	Dateiname oder not valid
x08-x0A	Kennung (z.B. .txt, .exe, .com)
x0B	Schreibschutz, versteckt, ...
x0C-x15	Reserviert für MSDOS-Erweiterungen
x16-x17	Uhrzeit des letzten Schreibzugriffs
x18-x19	Datum des letzten Schreibzugriffs
x1A-x1B	Adresse des ersten Sektors
x1C-x1F	Länge der Datei

Weitere Sekundärspeicher

- Floppy
- CD-ROM
- DVD
- Magnetband

Floppy



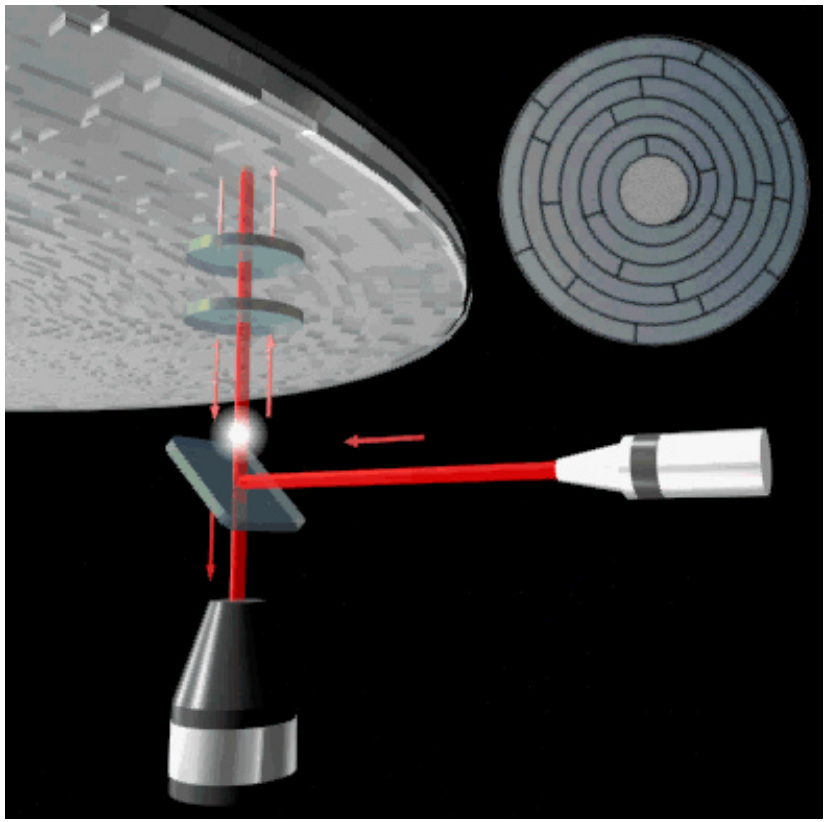


Unterschiede zu einer Festplatte

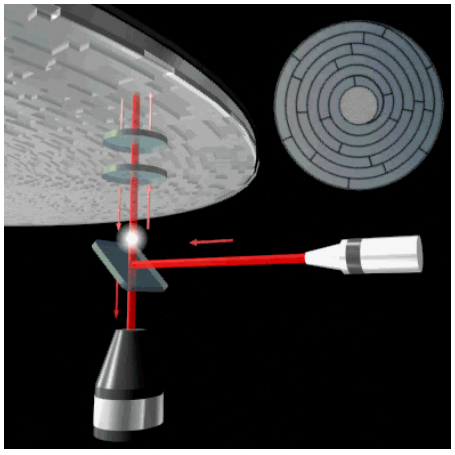
- Nur 1 Platte, die zudem flexibel
- Lese/Schreibköpfe sind im Laufwerk getrennt von der Diskette untergebracht

- Beim Zugriff fester Kontakt zwischen Lesekopf und Diskettenoberfläche
- Umdrehungsgeschwindigkeit 360 rp
- Langsamere Zugriffszeit: 150 ms
- Speicherkapazität: bis zu 2 Mbyte
- Sehr viel billiger

CD-ROM / DVD



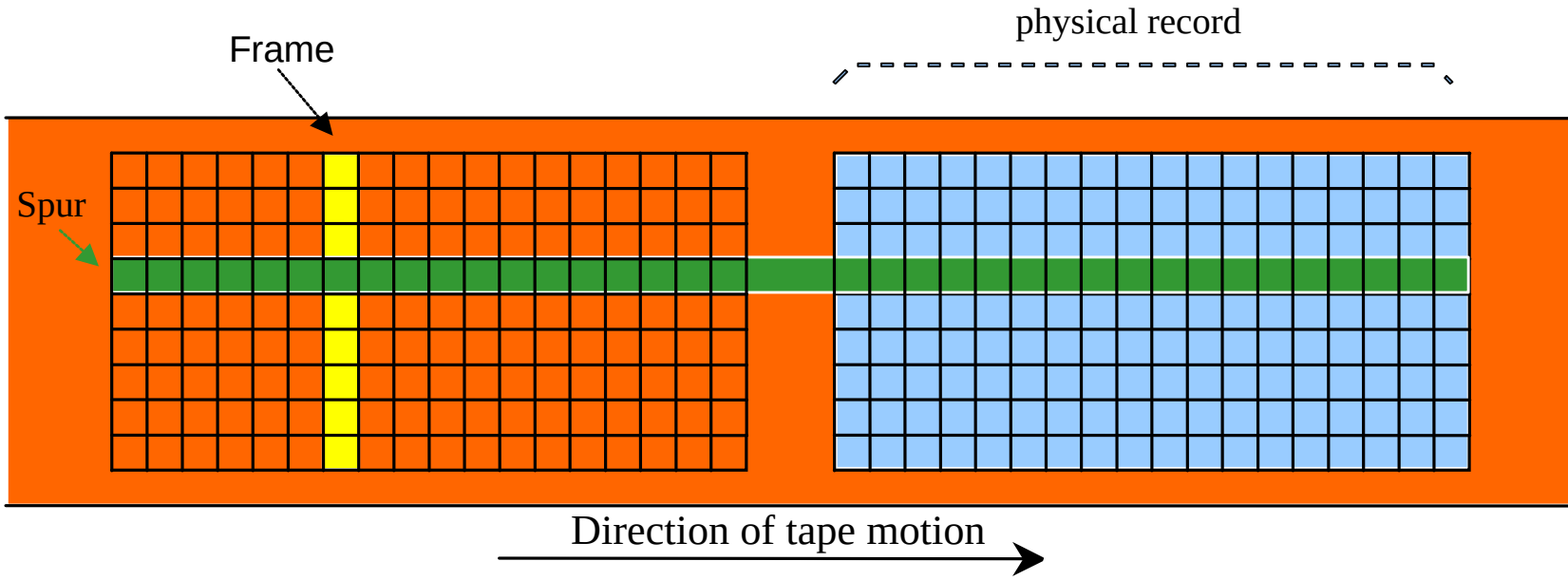
- Daten werden auf einer spiralförmigen Spur abgelegt.
- Beschriebene Oberfläche besteh aus **Gruben** (**pit**) und **Böden** (**land**), über die die Informatio kodiert
  - Übergang Grube-Grube: 0
  - Übergang Boden-Boden: 0
  - Übergang Grube-Boden: 1
  - Übergang Boden-Grube: 1
- Das Laufwerk besteht aus einer rotem Laser und einem Sensor, der das empfangene Licht auswertet.
  - Gruben reflektieren schwach
  - Böden reflektieren stark



CD-ROM / DVD

Scheiben-Durchmesser	120 mm	120 mm	
Grösse der Gruben	0,80 microns	0,40 microns	
Spur-Abstand	1,60 microns	0,74 microns	
Wellenlänge des Lasers	0,78 microns	0,65 microns	
Kapazität	650 MByte	4,7 GByte	
Einfache Übertragungs-geschwindigkeit	150 KByte/sec	1,4 Mbyte/sec	10 Mbyte/sec

Magnetband



- Bandlänge: 700-1000 m
- Schreib / Lesedichte: 1600-6250 bpi (**b**yte **p**er **i**nch)
- Schreibgeschwindigkeit: 10^6 Byte/sec

- Speicherworte werden in Frames abgespeichert bestehend aus
  - 8 Datenbits
  - 1 Paritybit (Überprüfungsbit)
- Information wird byteseriell gelesen / geschrieben
- Die Aufzeichnung erfolgt