



Universität Bielefeld  
Technische Fakultät

**R|V|S**

**Rechnernetze und  
Verteilte Systeme**

# **Speicherverwaltung**

**Technische Informatik II  
Wintersemester 2002/03**

Heiko Holtkamp  
Heiko@rvs.uni-bielefeld.de

# Speicherverwaltung

- Speicher ist eine wichtige Ressource, die sorgfältig verwaltet werden muss.
- In der Vorlesung soll untersucht werden, wie Betriebssysteme ihren Speicher verwalten.

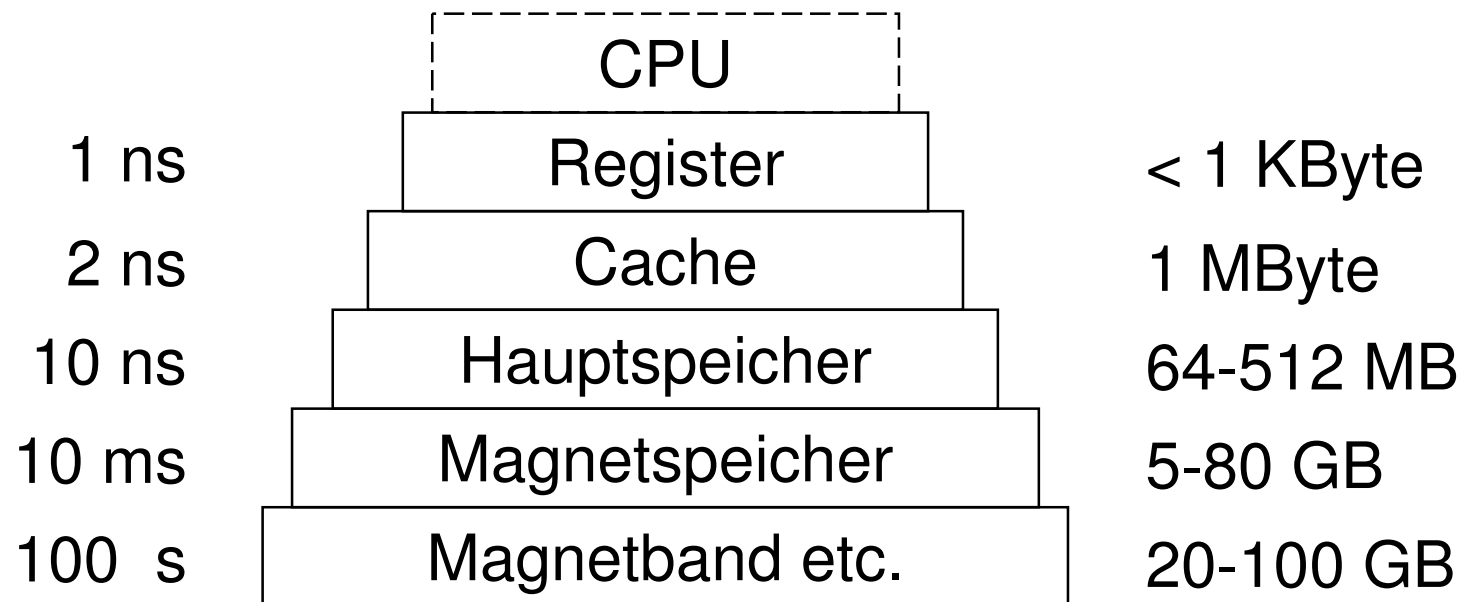
# Speicherverwaltung

- Jeder Programmierer hätte am liebsten einen unendlich großen, unendlich schnellen Speicher, der auch noch nicht flüchtig ist (und am besten ist er auch noch billig).
- Leider funktioniert realer Speicher nicht so!
- Die meisten Computer haben deshalb eine **Speicherhierarchie**.

# Speicherhierarchie

Typische Zugriffszeit\*

Typische Kapazität\*



\* Sehr grobe Schätzungen

# Speicherverwaltung

- Der Teil des Betriebssystems, der die Speicherhierarchie verwaltet, heißt **Speicherverwaltung**.
- Er verfolgt, welche Speicherbereiche gerade benutzt werden und welche nicht, teilt Prozessen Speicher zu, wenn sie ihn benötigen und gibt ihn nachher wieder frei.
- Außerdem verwaltet er die Auslagerung von Speicher auf die Festplatte (swapping), wenn der Hauptspeicher zu klein wird.

# Address Translation

- Die Adressen, die die CPU benutzt sind nicht identisch mit den Adressen, die der (physische) Speicher benutzt.
- Die Adressen müssen übersetzt werden -> **Address Translation.**
- Diese Funktion u.a. heißt **Memory Management.**

# Bit, Byte, Word

- 1 Byte = 8 Bit (genügend, um ein Zeichen in ASCII\* darzustellen).
- 1 Word =
  - 1 Byte („8-Bit-Maschine“), oder
  - 2 Byte („16-Bit-Maschine“), oder
  - 4 Byte („32-Bit-Maschine“), oder
  - 8 Byte („64-Bit-Maschine“).

\* American Standard Code for Information Interchange

# Adressraum (Address Space)

- Prinzip: Die möglichen Adressen sind 1 Word groß.
- 1 Word = 8 Bit, wir haben eine „8-Bit-Maschine“
- 1 Word = 2 Bytes = 16 Bit, wir haben eine „16-Bit-Maschine“
- ... 4 Bytes... 32 Bit... „32-Bit“
- ... 8 Bytes... 64 Bit... „64-Bit“



# Arithmetik

- Beispiel:  $\text{exp}(2,5) = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 32$
- Beispiel:  $\text{exp}(5,2) = 5 \cdot 5$
- Potenzgesetze:
  - $a^x a^y = a^{x+y}$
  - $(a^x)^y = a^{xy}$
  - $(ab)^x = a^x b^x$
  - $(a/b)^x = a^x / b^x$
  - $a^{-x} = 1/a^x$

# Aufpassen!

- 1 KB = 1 Kilobyte = 1024 Bytes
- Aber: 1 Kb = 1 Kilobit
- $\text{Exp}(1\text{K},2) = \text{exp}(1024,2) = 1\text{M}$
- $1\text{M} \cdot 1\text{K} = 1\text{G}$

# Adressraum

- 8-Bit-Maschine:  $\exp(2,8) = 256$  Bit
- 16-Bit:  $\exp(2,16) = \exp(2,6) \cdot \exp(2,10) = 64K$
- 32-Bit:  $\exp(2,32) = \exp(2,2) \cdot \exp(2,30) = 4 \cdot \exp(\exp(2,10),3) = 4 \cdot \exp(1K,3) = 4G$
- 64-Bit:  $\exp(2,64) = \exp(4G,2) = \text{enorm groß!}$

# Adressraum

- 8-Bit-Maschine: klein
- 16-Bit-Maschine: kleiner als ein Chip (DRAM)
- 32-Bit-Maschine: noch relativ teuer und nimmt Platz
- 64-Bit-Maschine: derzeit noch (fast) unbezahlbar

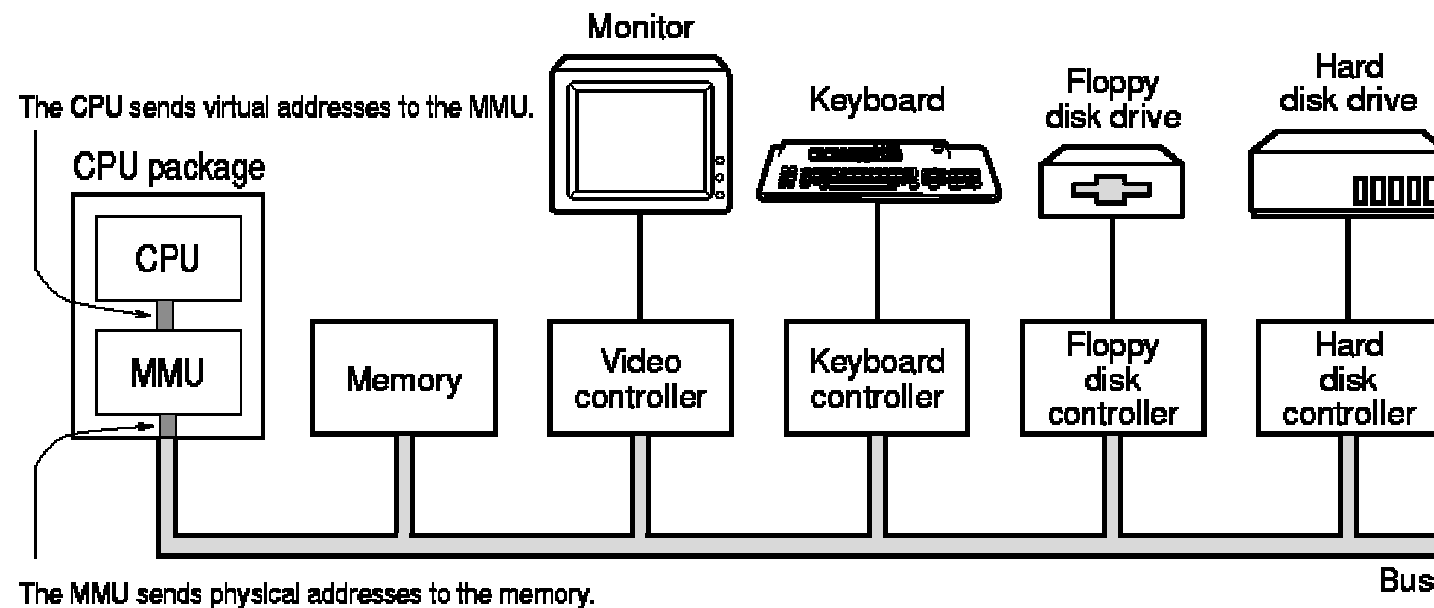
# Adressraum

- 32-Bit und 64-Bit Maschinen haben (oft) mehr Adressraum als **Physischen Speicher (Physical Memory)**
- Dieser Adressraum heißt **Virtual Memory (Virtueller Speicher)**
- Es gibt (oft) viel mehr VM als PM

# Aufteilung des VM

- Ein Teil des VM liegt im PM
- Der andere Teil liegt auf der Festplatte
- Der VM wird in **Pages (Seiten)** aufgeteilt
- Eine Seite ist 512B oder 1KB oder 4KB oder ... groß
- Nehmen wir im weiteren an, eine Seite ist 4KB groß

# Memory Management Unit (MMU)

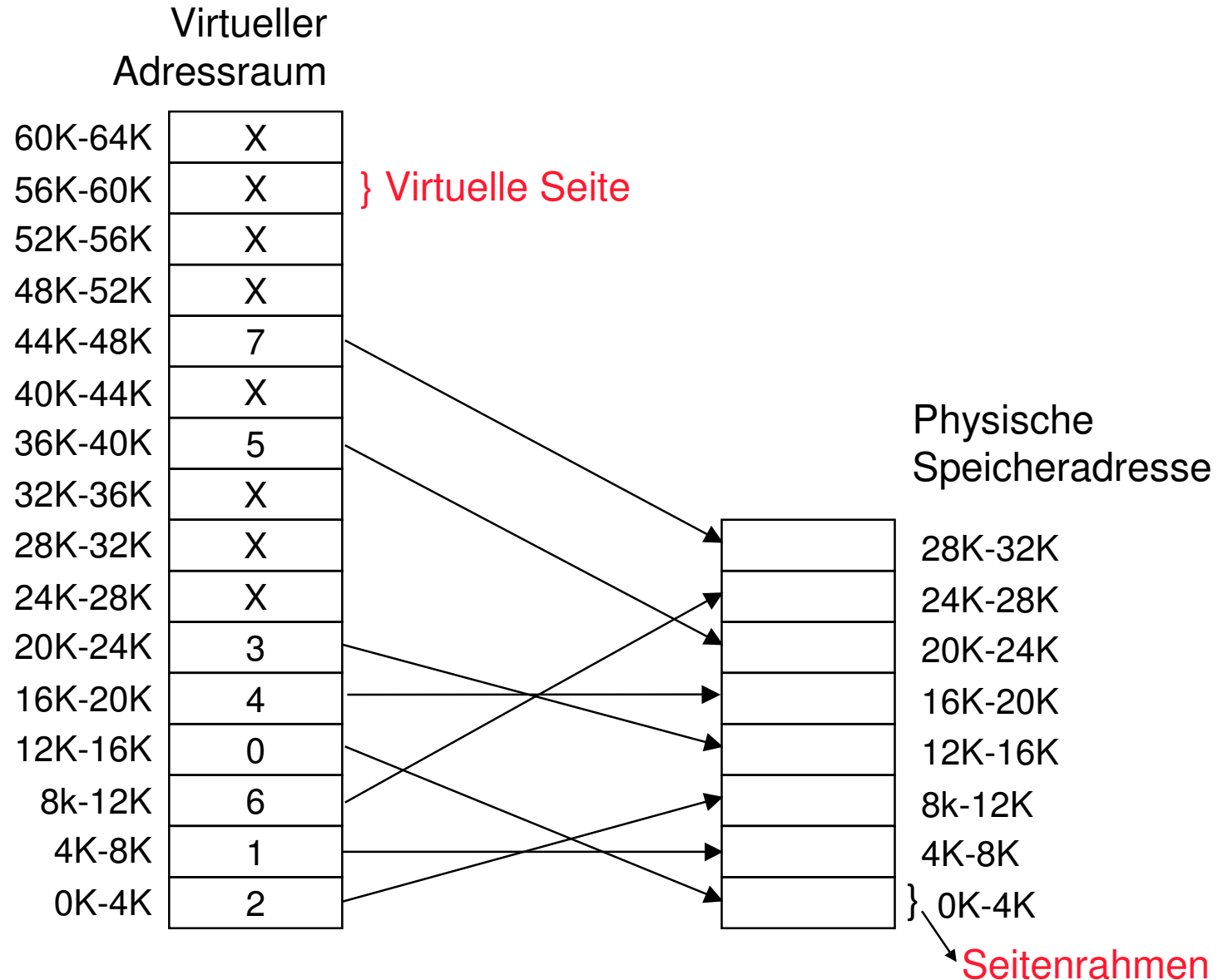


# Memory Managemet

- Die MMU enthält eine **Page Table** (**Seitentabelle**)
- Eine Page Table ist eine Datenstruktur
- Eine Page Table ist ein Array



# Virtueller Speicher: Paging



# Paging: Beispiel

Virtueller Adressraum

60K-64K	X
56K-60K	X
52K-56K	X
48K-52K	X
44K-48K	7
40K-44K	X
36K-40K	5
32K-36K	X
28K-32K	X
24K-28K	X
20K-24K	3
16K-20K	4
12K-16K	0
8K-12K	6
4K-8K	1
0K-4K	<b>2</b>

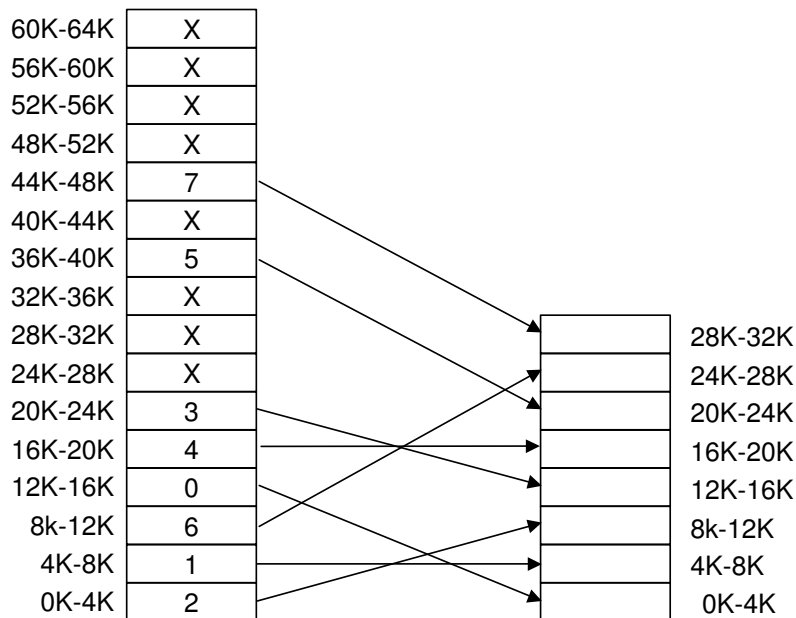
Ein Programm will auf die Adresse 0 zugreifen,  
`MOV REG, 0`

- Die virtuelle Adresse 0 wird an die MMU geschickt
- Die MMU stellt fest, dass die Adresse zur Seite 0 gehört (0 – 4095)
- Dies entspricht Seitenrahmen 2 (8192-12287)
- Also bildet die virt. Adr. 0 auf die physik. Adr. 8192 ab

Physische Speicheradresse

	28K-32K
	24K-28K
	20K-24K
	16K-20K
	12K-16K
	8K-12K
	4K-8K
	0K-4K

# Paging: Beispiel

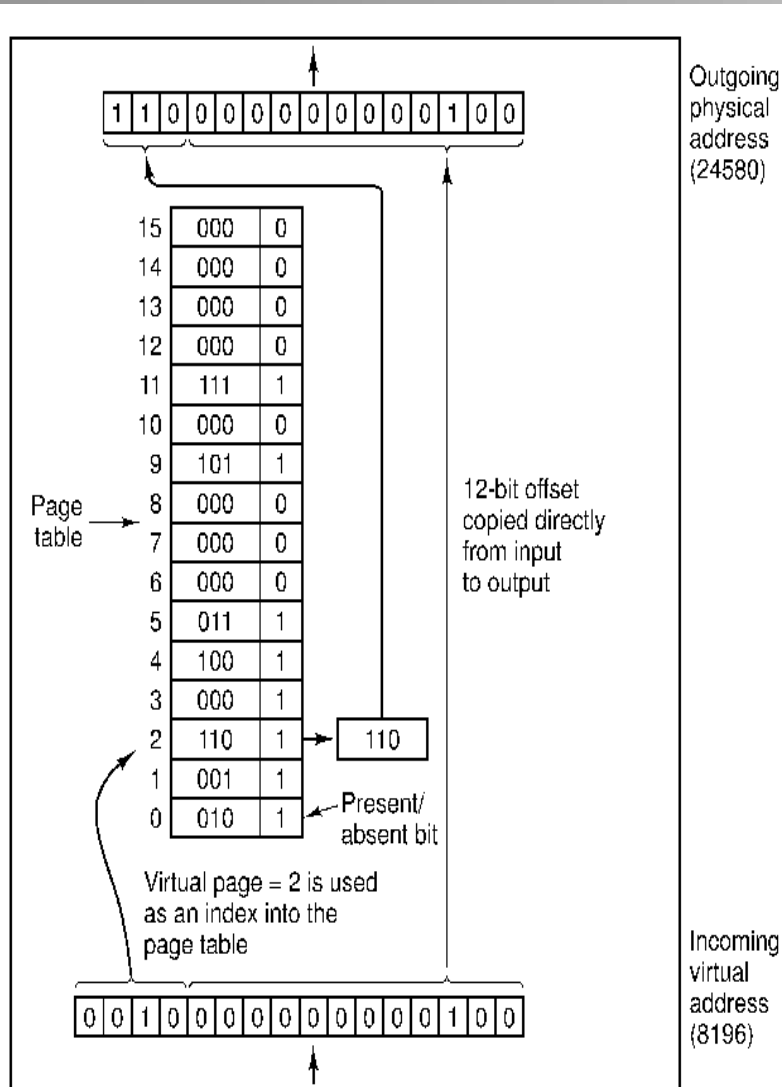


- Analog wird der Befehl  
MOV REG, 8192  
auf den Befehl  
MOV REG, 24576  
abgebildet.
- Die Adresse 20500 ist 20 Byte  
vom Beginn der virt. Seite 5  
(20480 – 24575) entfernt, und  
wird auf die physische Adresse  
 $12288 + 20 = 12308$  abgebildet.

# Paging

- Pfeile sind „Pointer“
- Wie wird gepointet?
- VM-Adress-Space = 16 Pages =  $\exp(2,4)$  Pg
- PM-Adress-Space = 8 Pages =  $\exp(2,3)$  Pg
- VM braucht 4 Bit, um die Page zu identifizieren
- PM braucht 3 Bit, um die Page zu identifizieren
- Innerhalb einer Page hat man 4K =  $\exp(2,12)$  Speicherelemente
- 12 Bit werden gebraucht, um ein Speicherelement innerhalb der Seite zu identifizieren
- Die 12-Bit Adresse heißt **Offset**

# Paging: Seitentabelle (page table)



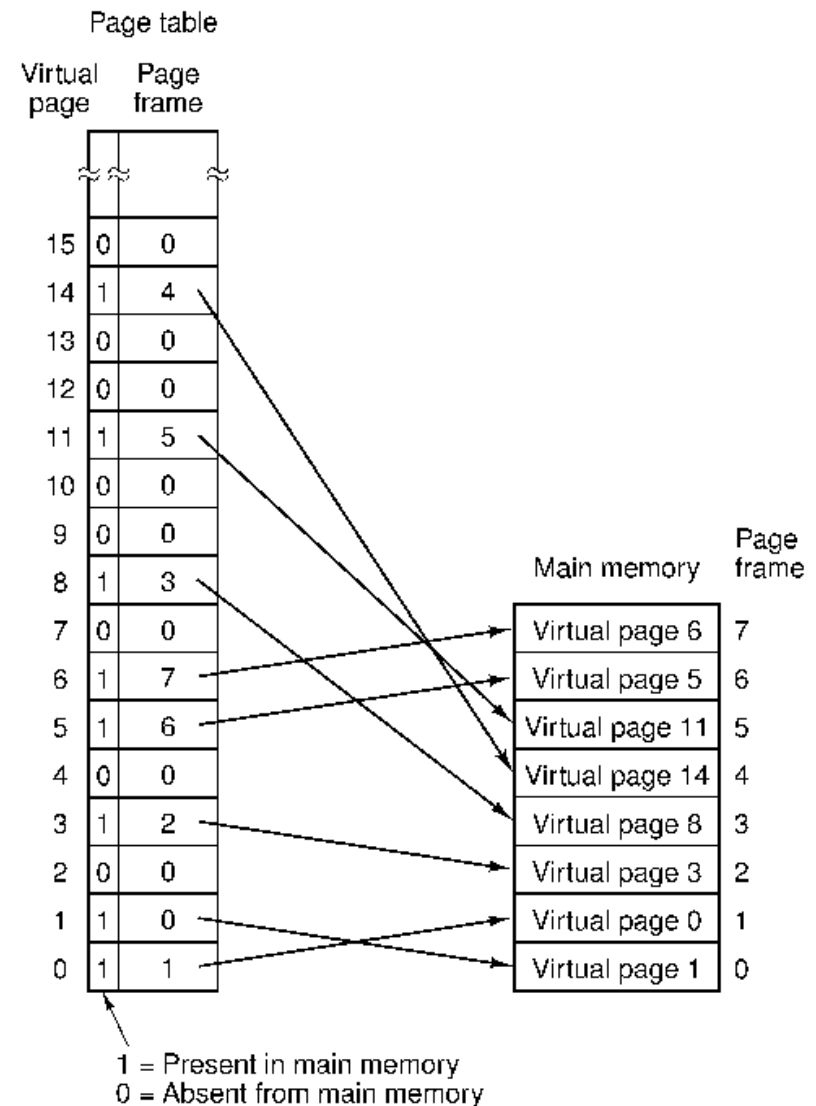
- Beispiel: Virtuelle Adresse 8196 (Binär 0010000000000100)
- Die virtuelle 16-Bit-Adresse wird in eine *4-Bit-Seitennummer* und einen *12-Bit-Offset* zerlegt.
- Mit 4 Bits für die Seitennummer lassen sich 16 Seiten ( $2^4$ ) repräsentieren.
- Mit einem 12-Bit-Offset können alle 4096 Byte ( $2^{12}$ ) einer Seite adressiert werden.
- Die Seitennummer wird als Index für die **Seitentabelle (page table)**, die die Nummer des Seitenrahmens enthält, der der virtuellen Adresse entspricht, genutzt.
- Ist die Seite ‚vorhanden‘ wird die Nummer des Seitenrahmens in die oberen drei Bits des Ausgaberegisters kopiert. Der 12-Bit Offset wird unverändert in die unteren 12 Bit des Ausgabereg. kopiert.

# Seitentabellen

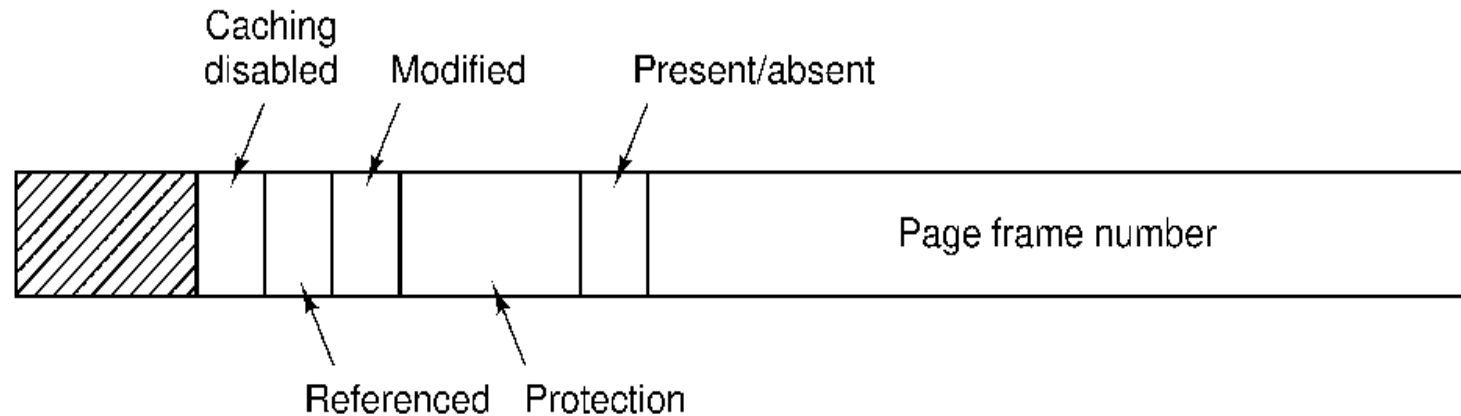
- Im einfachsten Fall funktioniert die Abbildung von virtuellen auf physische Adressen genau wie beschrieben.
- Der Zweck der Seitentabelle ist es, virtuelle Seiten auf Seitenrahmen abzubilden.
- Die Tabelle ist eine Funktion, mit der virtuellen Seitennummer als Argument und der Seitenrahmennummer als Ergebnis.
- Mit dem Ergebnis dieser Funktion kann der Teil einer virtuellen Adresse, der die Seitennummer enthält, durch einen Adressteil für den Seitenrahmen ersetzt werden, wodurch dann eine physische Adresse gebildet wird.

# Paging

- Für sich genommen, löst die Möglichkeit, die 16 virtuellen Seiten über die MMU auf jeden beliebigen der 8 Seitenrahmen abzubilden, noch nicht das Problem, dass der virtuelle Adressraum größer als der physische Speicher ist.
- Es gibt nur 8 physische Seitenrahmen, deshalb werden nur 8 der Seiten auf physischen Speicher abgebildet.
- In realer Hardware wird ein **present/absent-Bit** (anwesend/abwesend) genutzt, um Überblick zu haben, welche Seiten auf physischen Speicher abgebildet sind.



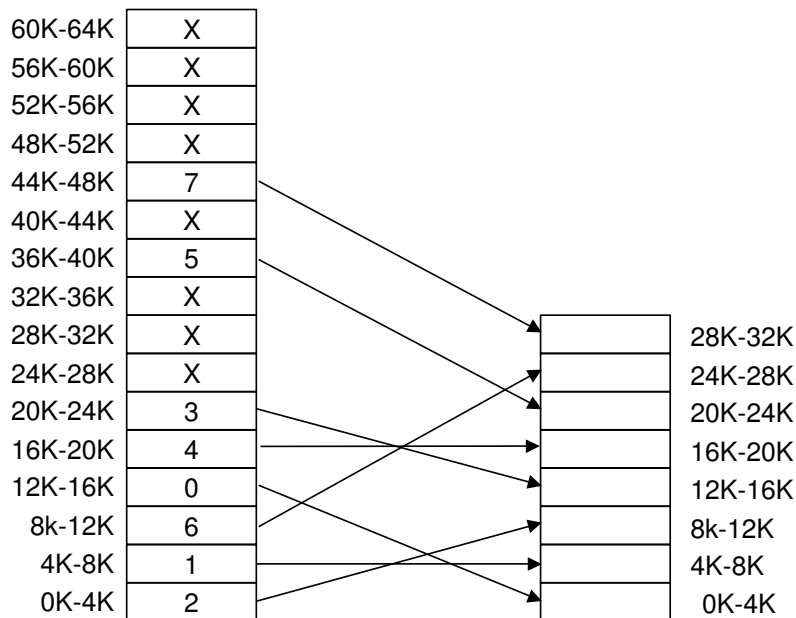
# Seitentabellen



- Seitenrahmennummer: Zeiger auf die phys. Speicheradresse
- P/A: Seite liegt im Speicher ja/nein
- Protection-Bit(s): 1 Bit -> 0 = Lesen/Schreiben; 1 = Lesen  
3 Bit -> Lesen; Überschreiben; Ausführen
- Modified-Bit: Seite verändert ja/nein (auch **Dirty Bit**)
- Referenced-Bit: Protokolliert die Zugriffe auf die Seite
- Caching: erlaubt / nicht erlaubt



# Paging



- Was passiert, wenn versucht wird auf eine Seite zuzugreifen, die nicht im physischen Speicher liegt?

```
MOV REG, 32780
```

- Ein Systemaufruf mit einem **Seitenfehler (page fault)** wird ausgelöst.
- Ein wenig genutzter Seitenrahmen wird ausgelagert und lädt die Seite, die den Seitenfehler ausgelöst hat in den frei gewordenen Seitenrahmen.
- Die Seitentabelle wird angepasst und der Befehl nochmals ausgeführt.

# Seitentabellen

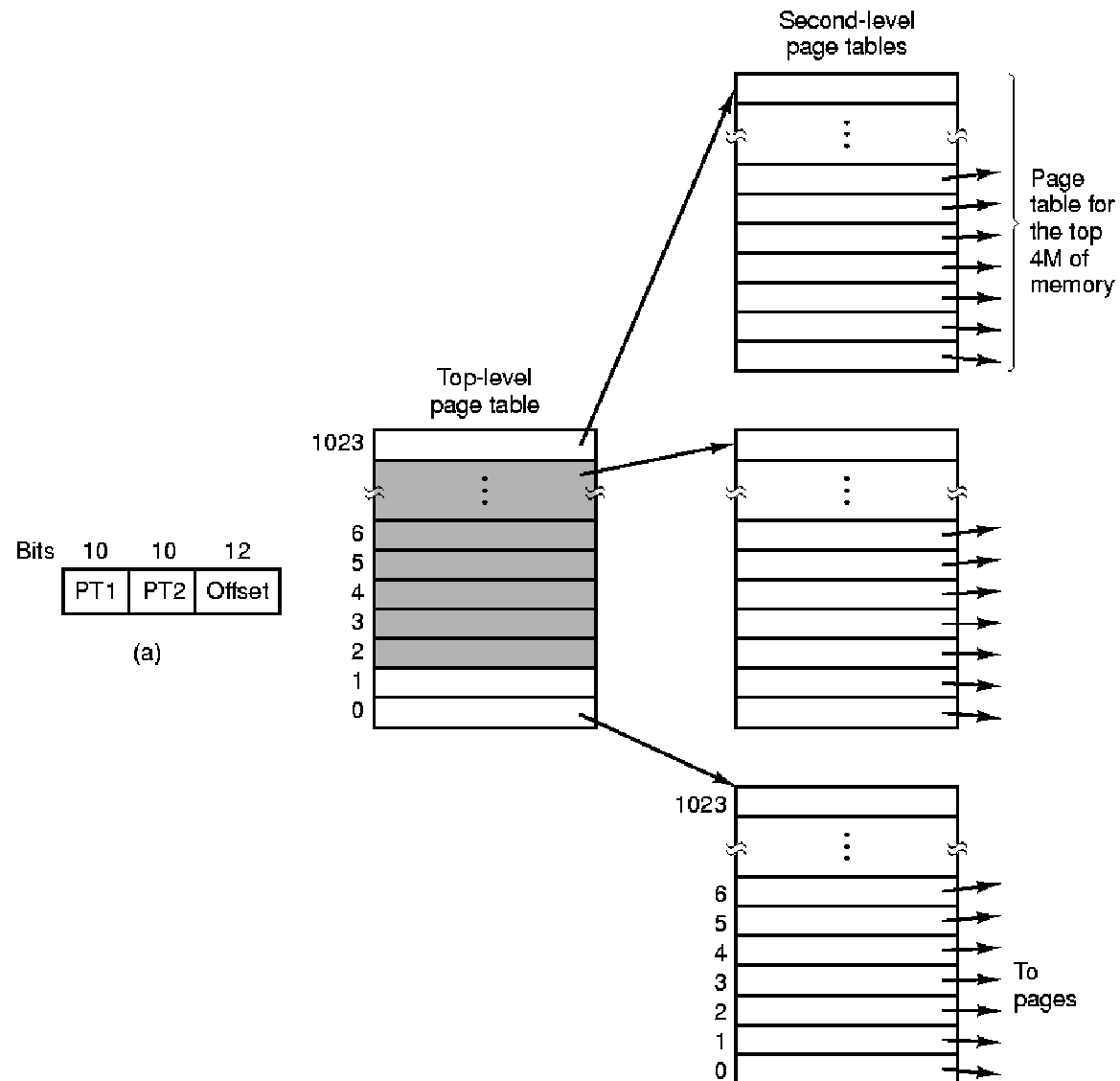
- Es ergeben sich zwei wichtige Problemstellungen:

1. Die Seitentabelle kann extrem groß werden

Virtuelle Adressen moderner Computer sind mind. 32 Bit lang. Bei einer Seitengröße von 4 KB hat ein 32-Bit-Adressraum eine Million Seiten.

2. Die Umrechnung muss sehr schnell sein

# Mehrstufige Seitentabellen



# TLB - Translation Lookaside Buffer

- Programme neigen dazu, sehr viele Zugriffe auf sehr wenige Seiten auszuführen.
- Lösung: Hardware, die virtuelle Adressen ohne Umweg über die Seitentabelle auf physische Adressen abbildet: **TLB** (oder auch **Assoziativspeicher**)
- Der TLB besteht aus einer kleinen Zahl von Einträgen, typischerweise selten mehr als 64

Gültig	Virtuelle Seite	Verändert	Schutz	Seitenrahmen
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

# Invertierte Seitentabellen

- Traditionell benötigen Seitentabellen von der bisher vorgestellten Art einen Eintrag pro Seite
- Wenn der Adressraum  $2^{32}$  Byte enthält, mit 4096 Byte pro Seite, werden über eine Million Seiten gebraucht.
- Das absolute Minimum für die Größe der Seitentabelle ist also 4 MB
- Das ergibt Probleme bei 64-Bit-Systemen: Adressraum  $2^{64}$  Byte, 4KB große Seiten -> Seitentabelle hat  $2^{52}$  Einträge; bei 8 Byte pro Eintrag wäre die Tabelle dann 30 Millionen GB groß
- Lösung: **Invertierte Seitentabelle**
- In der Seitentabelle wird ein Eintrag für jeden physischen Seitenrahmen gespeichert, anstatt für jede Seite im virtuellen Adressraum
- Eine Seitentabelle mit 64-Bit großem Adressraum, 4KB großen Seiten und 256 MB Speicher hätte dann nur 65 536 Einträge

# Invertierte Seitentabellen

- Es wird enormer Speicherplatz gespart
- Nachteil: es ist wesentlich aufwändiger eine virtuelle auf eine physische Adresse abzubilden!
- Wenn Prozess  $n$  auf die virtuelle Seite  $p$  zugreifen will, kann der physische Seitenrahmen nicht einfach gefunden werden, in dem sie die virtuelle Seitennummer als Index für die Seitentabelle benutzt.
- Die gesamte Seitentabelle muss nach dem Eintrag  $(p,n)$  durchsucht werden.
- Lösung: TLB nutzen

# Seitenersetzungsalgorithmen

- Bei jedem Seitenfehler muss das Betriebssystem eine Seite auswählen, die aus dem Speicher entfernt wird, um für die neue Seite Platz zu machen.
- Probleme sind ähnlich wie bei Caching-Algorithmen

# Seitenersetzungsalgorithmen

- Basis: Festplatten-Speicher-Austausch ist zeitlich ineffizient (Zeit-Ressourcen-Verschwendung)
- Versuch: Ineffizienz so weit wie möglich vermeiden



# Seitenersetzungsalgorithmen

- Es gibt ein Present/Absent-Bit
- Dies bezeichnet, ob die Seite im PM liegt (set) oder nicht (not set)
- Falls die Seite abwesend ist, wird die 4-Bit VM-Seitennummer weitergegeben
- Ein Algorithmus wählt einen Platz (1 Page) im PM aus
- Die Seite wird von der Festplatte in den PM gebracht
- Die Seitentabelle wird modifiziert
- Die Adresse wird wie vorher übersetzt

# Seitenersetzungsalgorithmen

- Seitenersetzungsauswahl
  - Hat die Seite Sonderprivilegien?
  - Ist die Seite benutzt worden?
  - Falls ja, ist sie modifiziert worden?

# Seitenersetzungsalgorithmen

- Sonderprivilegien
  - Schlüsseltabelle für eine Mehrseiten-DB
  - Speicherverwaltungsseite für den Benutzerraum
  - „Benutzerausgewählt“ als immer anwesend:
    - Sonderseite für Programmnotfälle
    - Einer zeit heftiger Benutzung folgt eine lange Pause

# Seitenersetzungsalgorithmen

- Page referenziert?
  - Daten schon benutzt worden
  - Zeichen, dass sie weiter benutzt wird (könnte zum „Working Set“ des Prozesses gehören)
  - Wenn die Seite ausgelagert wird und zum Working Set gehört, muss sie bald wieder in den Speicher kommen
  - Ineffizient, wenn das passiert!

# Seitenersetzungsalgorithmen

- Page modifiziert?
  - Falls nein, kann die neue Seite sie einfach überschreiben ohne dass sie vorher wieder auf der Festplatte gespeichert werden muss
  - Falls ja, muss sie auf der Festplatte gespeichert werden, bevor die neue Seite in den Speicherbereich geladen werden kann

# Seitenersetzungsalgorithmen

- „Working Set“ (Peter J. Denning, 1968)
  - Prozesse neigen dazu, ihre Zugriffe in jeder Phase ihrer Ausführung auf einen relativ kleinen Teil ihrer Seiten zu beschränken
  - Dieses Verhalten wird als **Lokalität der Referenzen (locality of reference)** bezeichnet
  - Die Menge von Seiten, die ein Prozess zu einem bestimmten Zeitpunkt nutzt wird als **Working Set (Arbeitsbereich)** bezeichnet

# Working Set

- Wenn das gesamte Working Set im Speicher ist, läuft der Prozess ohne viele Seitenfehler bis zur nächsten Phase seiner Ausführung (Beispiel Compiler-Läufe)
- Reicht der verfügbare Speicher nicht für alle Seiten im Arbeitsbereich aus, erzeugt der Prozess sehr viele Seitenfehler und läuft sehr langsam ab
- In einem System mit Multiprogrammierung werden Prozesse häufig ausgelagert, um andere Prozesse zum Zug kommen zu lassen
- Dabei stellt sich die Frage, was man tun soll, wenn ein Prozess wieder eingelagert wird

# Working Set

- Nichts tun: Der Prozess erzeugt so lange Seitenfehler, bis sein Working Set geladen ist
- **Demand Paging (Einlagern bei Bedarf)**
- Diese Strategie ist sehr langsam, weil sehr viele Seitenfehler erzeugt werden



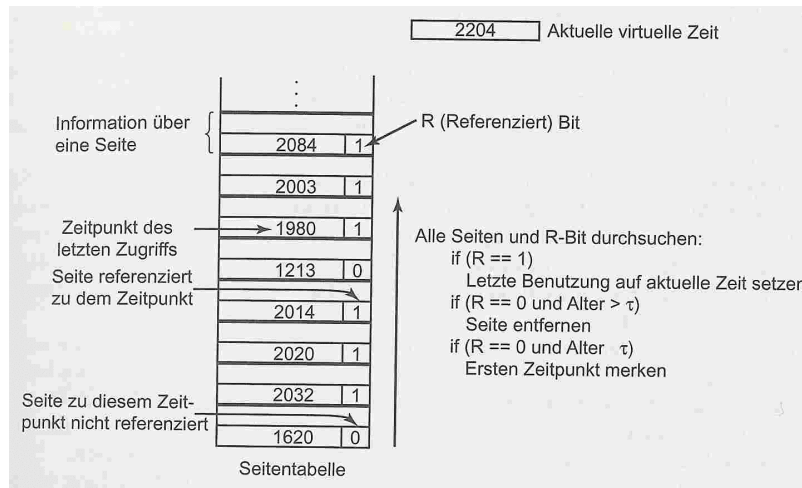
# Working Set

- Viele Betriebssysteme merken sich deshalb den Arbeitsbereich eines Prozesses, wenn sie ihn auslagern und sorgen dafür, dass er wieder geladen wird, bevor sie den Prozess weiter ausführen.
- Dieser Ansatz wird **Working Set Modell** genannt
- Strategien, die Seiten laden, noch bevor sie gebraucht werden, werden auch **Prepaging** genannt

# Working Set

- Für die Implementation des Working-Set-Modells muss das Betriebssystem zu jedem Zeitpunkt wissen, welche Seiten im Arbeitsbereich eines Prozesses sind.
- Daraus ergibt sich direkt ein Seitenersetzungsalgorithmus:
  - Wenn ein Seitenfehler auftritt, finde eine Seite, die nicht zum Arbeitsbereich gehört und lagere sie aus.
  - Benötigt wird ein Kriterium, welche Seiten zu einem bestimmten Zeitpunkt zum Arbeitsbereich gehören und welche nicht.

# Working Set



- Grundidee: Bei einem Seitenfehler eine Seite auslagern, die nicht zum WS gehört.
- Jeder Eintrag enthält zwei Informationen:
  - Ungefähre Zeit des letzten Zugriffs
  - R-Bit (referenziert ja/nein)
- Voraussetzung:
  - R-Bit wird gesetzt
  - R-Bits werden periodisch gelöscht (Timerunterbrechung)
- Bei jedem Seitenfehler wird die Tabelle nach einer Seite durchsucht, die ausgelagert werden kann.