

Betriebssysteme

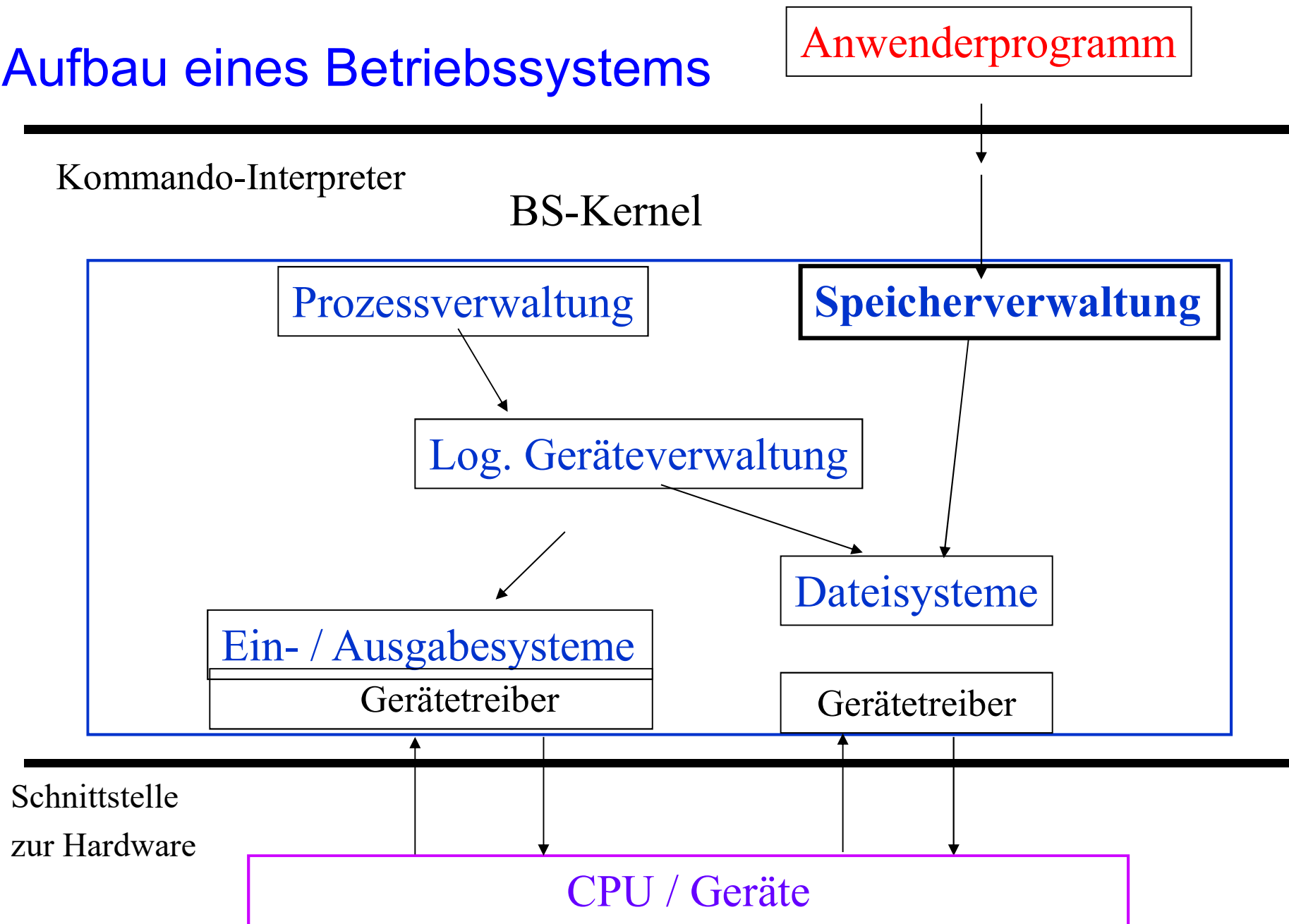
Studiengang Informatik / SAT

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- mwilhelm@hs-harz.de
- <http://www.miwilhelm.de>
- Raum 2.202
- Tel. 03943 / 659 338

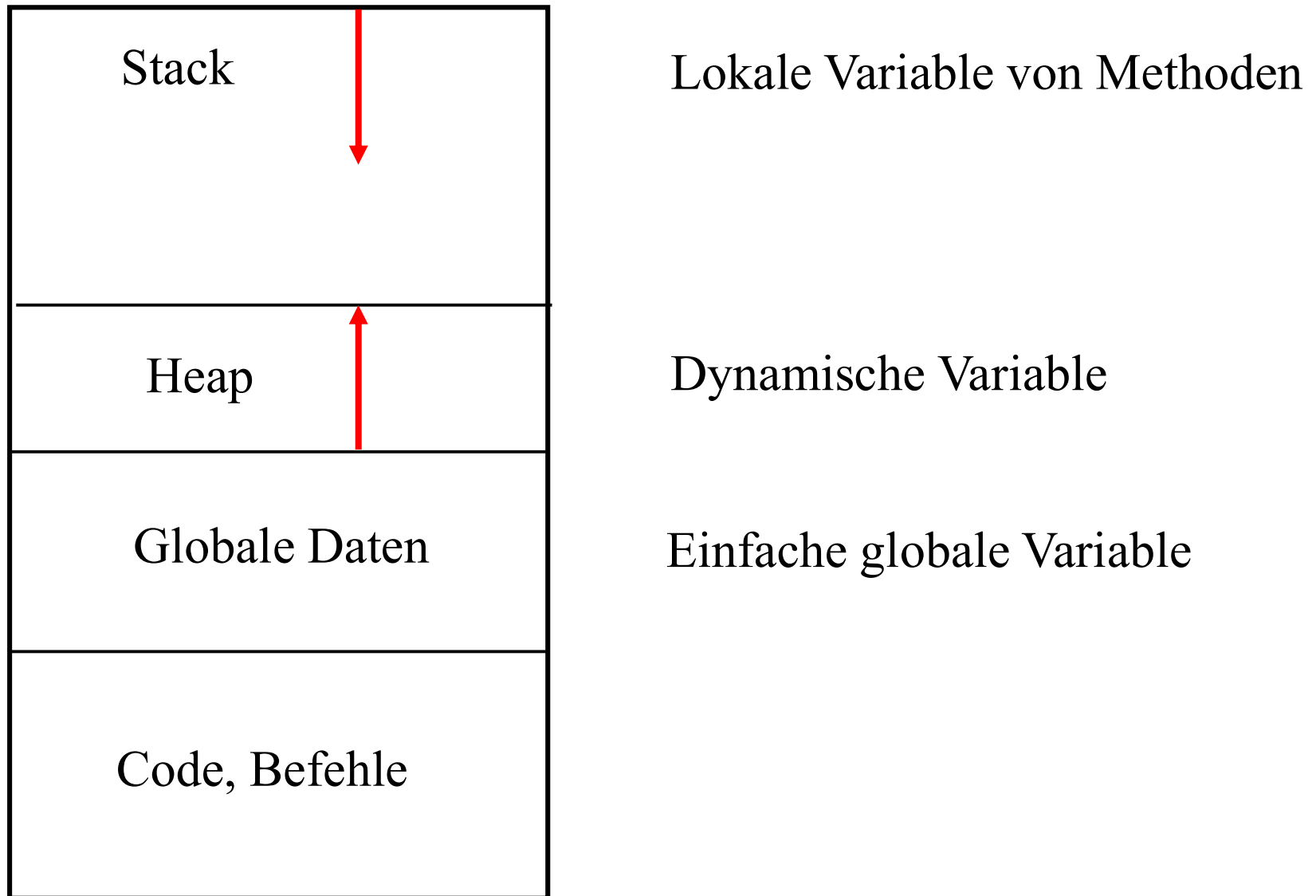
Gliederung

1. Einführung
- 2. Speicherverwaltung**
3. Dateisysteme
4. Unix, Linux
5. Prozesse, Thread
6. Deadlocks

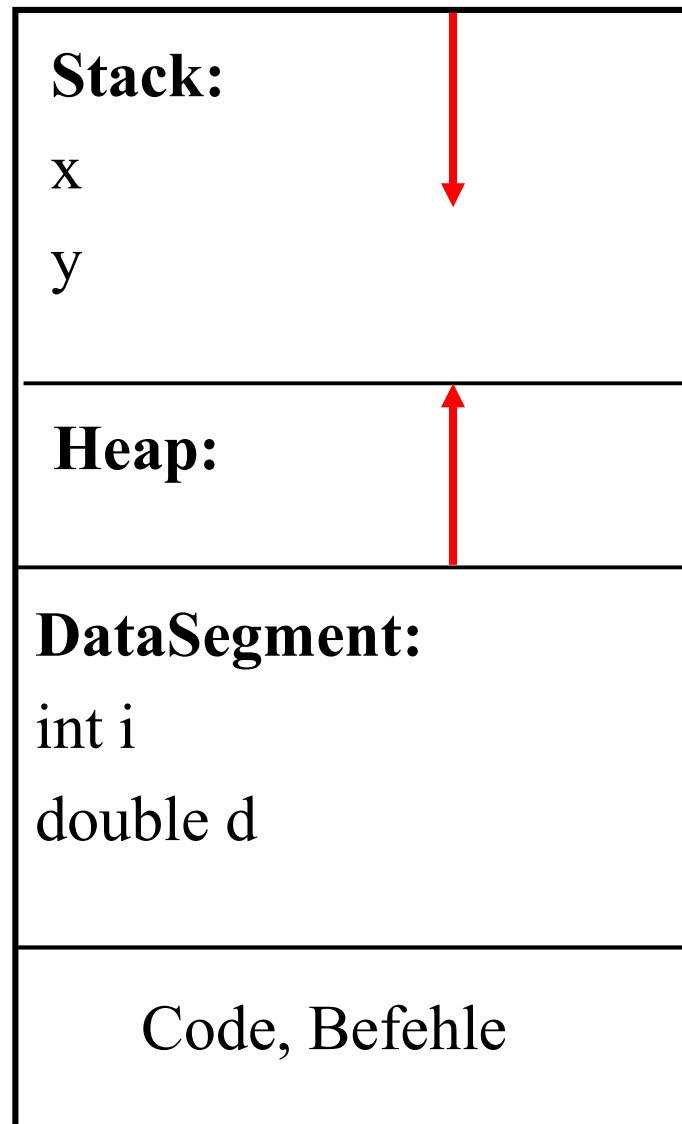
Aufbau eines Betriebssystems



Speicheraufbau eines Programms



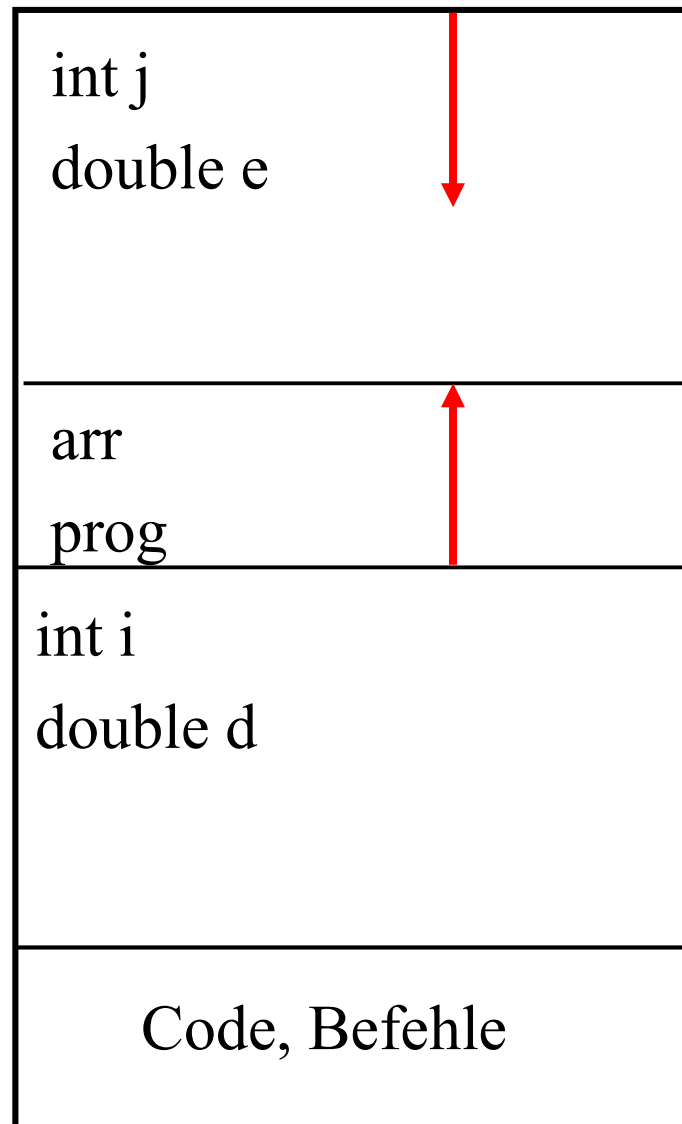
Speicheraufbau eines Programms



```
int xValue=0;  
double dValue=0.0;
```

```
class bsp3 {  
    int i;  
    double d;  
    double sin(double x) {  
        double y=....;  
        return y;  
    }  
}  
  
public static void main(String[] args) {  
    bsp3 prog = new bsp3();  
    bsp3.sin(5);  
}
```

Speicheraufbau eines Programms



```
public class bsp3 {  
    int i;  
    double d;  
    private void sin(double x) {  
        // calc  
    }  
    private void test1() {  
        int j;  
        double e;  
        int[] arr = new int[10];  
    }  
    public static void main(String[] args) {  
        bsp3 prog = new bsp3();  
        bsp3.test1();  
    }  
}
```

Aufgaben der Speicherverwaltung

- Verwaltung des freien Speichers
- Verwaltung des belegten Speichers
- Speicherzuordnung an einem Prozess
- Speicherentzug von einem Prozess
- Organisation des Speicherschutzes

Typen der Speicherverwaltung

- Speicherverwaltung (Zusammenhängende Bereiche)
 - ohne Swapping, ohne Auslagerung
 - Mit einem Programm
 - Mit mehreren Programmen
- Speicherverwaltung (Zusammenhängende Bereiche)
 - Mit Swapping, mit Auslagerung
 - Mit mehreren Programmen
- Speicherverwaltung mit Swapping, mit Paging
 - Mit mehreren Programmen

Benötigter Hauptspeicher / Festplattenspeicher

Beispiel: König von Narnia

Verwendete Speicher:

52 TByte = 52 TeraByte = 52,000 GigaByte

Speicherverwaltung (Zusammenhängende Bereiche)

Die Speicherbereiche werden frei und dynamisch eingeteilt

Keine Auslagerung

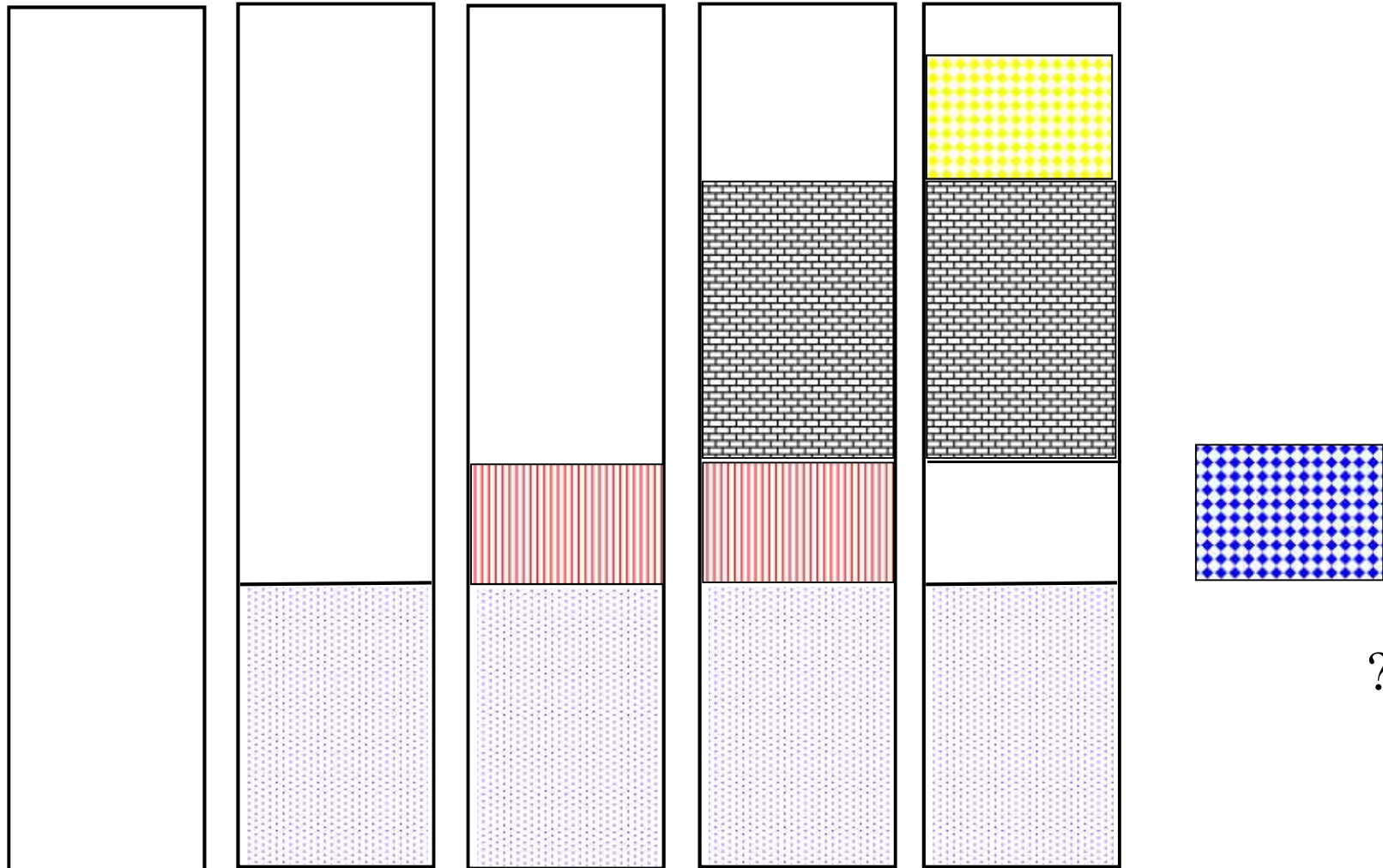
Vorteile:

- bessere Ausnutzung des Speichers
- Mehrere Strategien zum Ausfüllen der Lücken

Nachteile:

- Adressierung Problem mit Anfangsadresse $\neq 0$ (Segmentreg)
- Adresstransformation
 - ⊗ Statisch per Compiler
 - ⊗ Dynamisch pro Segment
 - ⊗ Dynamisch mit Basisregister
- Fragmentierung
- Speicher-Kompaktierung

Speicherbereich mehrerer Programme



C) Mehrere Programme mit Swapping (Auslagern von Speicher)

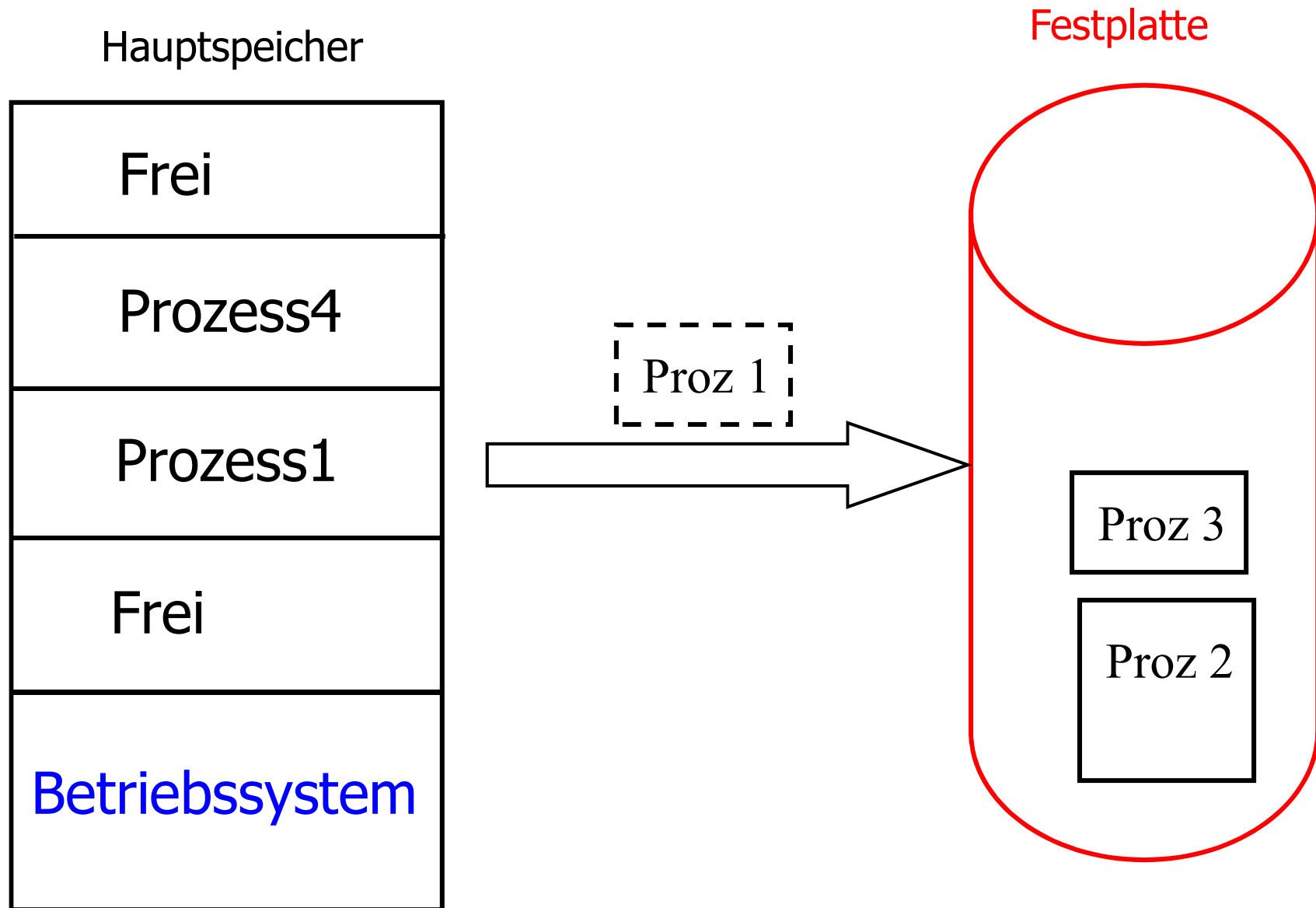
Definition:

Das Verlagern von Prozessen vom Hauptspeicher auf Platte und umgekehrt, heißt **Swapping**.

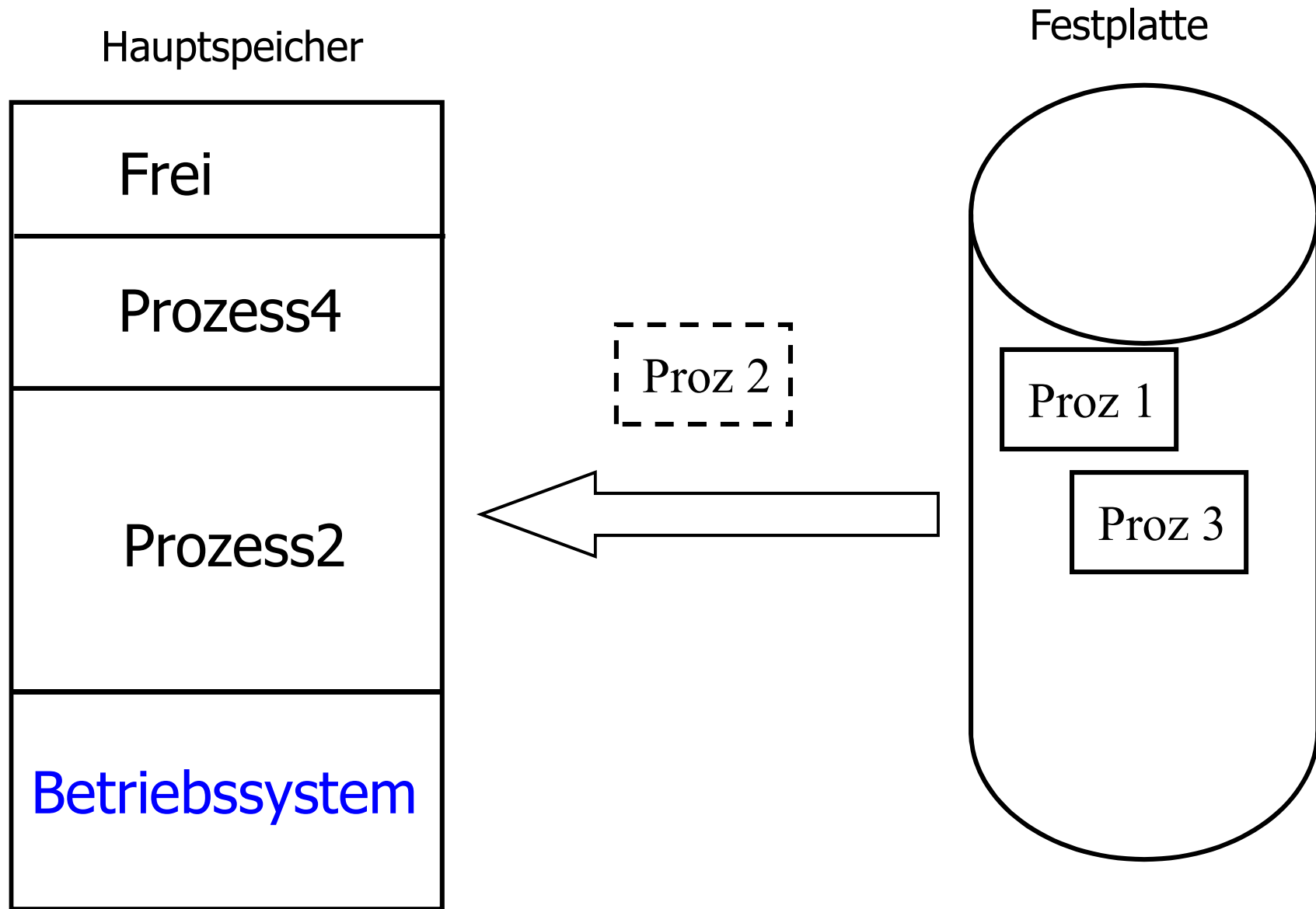
Problem:

bei variablen Partitionsgrößen entstehen Lücken im Speicher, wenn Prozesse beendet werden (Externe Fragmentierung) !

Swapping, Auslagern von Speicher



Swapping, Auslagern von Speicher



Hauptspeicher

- Beispiele der Speichergröße (RAM)
 - 128 KB
 - 4 GB
 - 1500 TB, nur 64-bit Betriebssystem, Apple
- Cache
 - Level1: Cache (Register): 1 MB
 - Level2: Cache 16x1MB
 - Level3: Cache 22 MB
- Speicheradressierung abhängig von der BUS-Breite

Auslagerungszeit: Seiten von 10 MB auslagern

PC / Apple:

- Übertragungszeit: $40,000 \text{ Byte /s} = 40 \text{ MB/s}$
- Positionierzeit: 8 ms
- Speicherungszeit: $10 / 40 + 8 \text{ ms} = 0,258 \text{ s}$
- Ladezeit: $10 / 40 + 8 \text{ ms} = 0,258 \text{ s}$
- Gesamtzeit: $2 * 0,258 \text{ s} = 0,516 \text{ s}$

Sun Fire 15K Starcat:

- Disk-Durchsatz (mittel): $12,0 \text{ GB/s}$
- Gesamtzeit: $2 * (10 / 12000) + 0,008 \text{ s} = 1,6 \text{ ms}$

Speicherverdichtung:

Kopiervorgang, um belegte Partitionen zusammenzuschieben (CPU ist beteiligt oder es gibt eine spezielle Hardware)

Adressen müssen gegebenenfalls geändert werden

Problem:

Halden und Keller, Heap und Stack, wachsen dynamisch, wieviel Speicher ein Programm benötigt, ist anfangs oft nicht bekannt

Beim Einlagern eines Prozesses von Platte wird noch zusätzlicher Raum zum Wachsen mitgegeben, (erneutes Auslagern, falls der Raum irgendwann nicht mehr ausreicht)

Freier Speicherraum muss ebenfalls verwaltet werden:

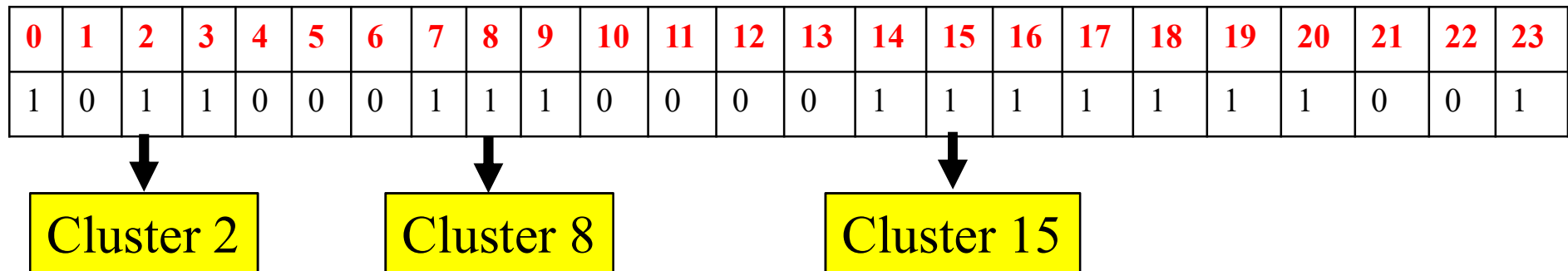
- mit Bitmaps,
- mit verknüpften Listen oder // MSDOS FAT
- mit Buddy System Liste mit 2^n

Speicherverwaltung mit Bitmaps (lange Bitfelder)

Clustergröße des Speichers: 512 Byte bis 64 kB

innerhalb einer Bitmap wird linear auf alle Cluster verwiesen

(0 = frei, sonst belegt)



Implementierung in Java

a[0]

0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	1

a[1]

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
1	1	0	0	0	0	1	1

Programmcode: ?

Speicherverwaltung mit Bitmaps (lange Bitfelder)

Beispiel:

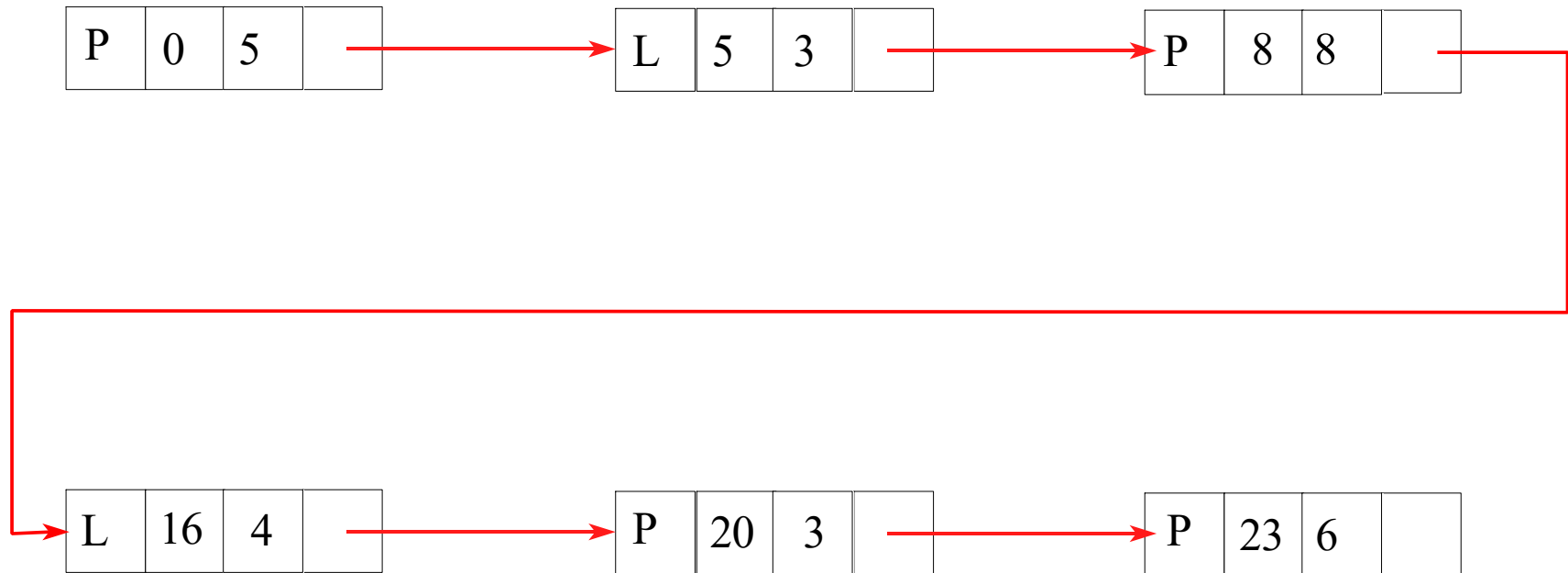
- Cluster hat 32 Bit, Variable ist 4 Byte lang (int).
- 1 Bit wird für das Markieren von 32 Bit verwendet.
- 1/33 des Speichers werden für die Bitmap verwendet.

4 Bytes = 32 Bits + 1Bit für die Bitmap

Eigenschaften:

- Aufwändige Suche
- Externe Fragmentierung
- BitArrays in Java und STL (C++) eingebaut

Beispiel: Einfach verkettete Liste (3. Semester)



P: Prozess

L: Leer

Anfangsadresse

Anzahl des
belegten Speichers

Bevor Prozess X terminiert



Nachdem Prozess X terminiert



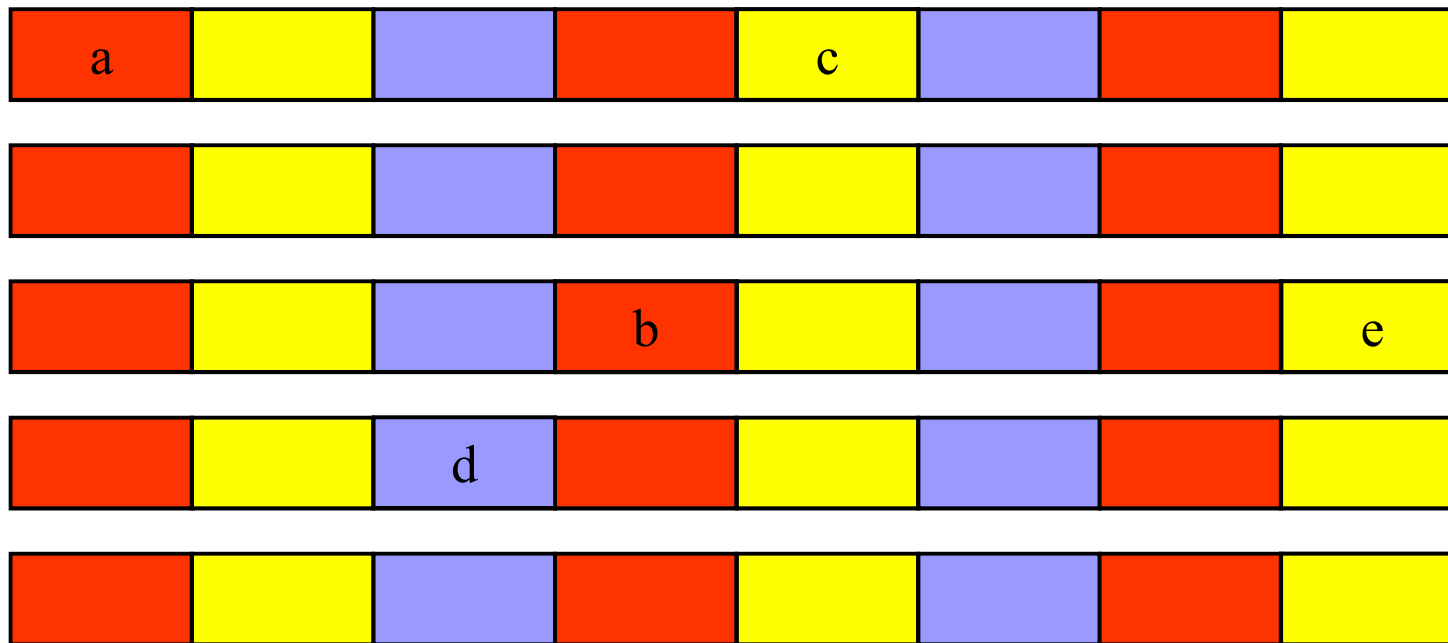
Seitenaufteilung mit festen Seiten (Pages, Paging)

Eigenschaften:

- Der Hauptspeicher wird in feste Blöcke aufgeteilt
- Dadurch ist eine Auslagerung einfach zu realisieren
- Wo liegt nun der Speicher eines Programms?
- In einem Block liegen **mehrere** Variablen
- Eine Variable kann aus **mehreren Blöcken** bestehen (Array)
- Wie ändert sich die Zuordnung, wenn ein Block aus- und eingelagert wird?

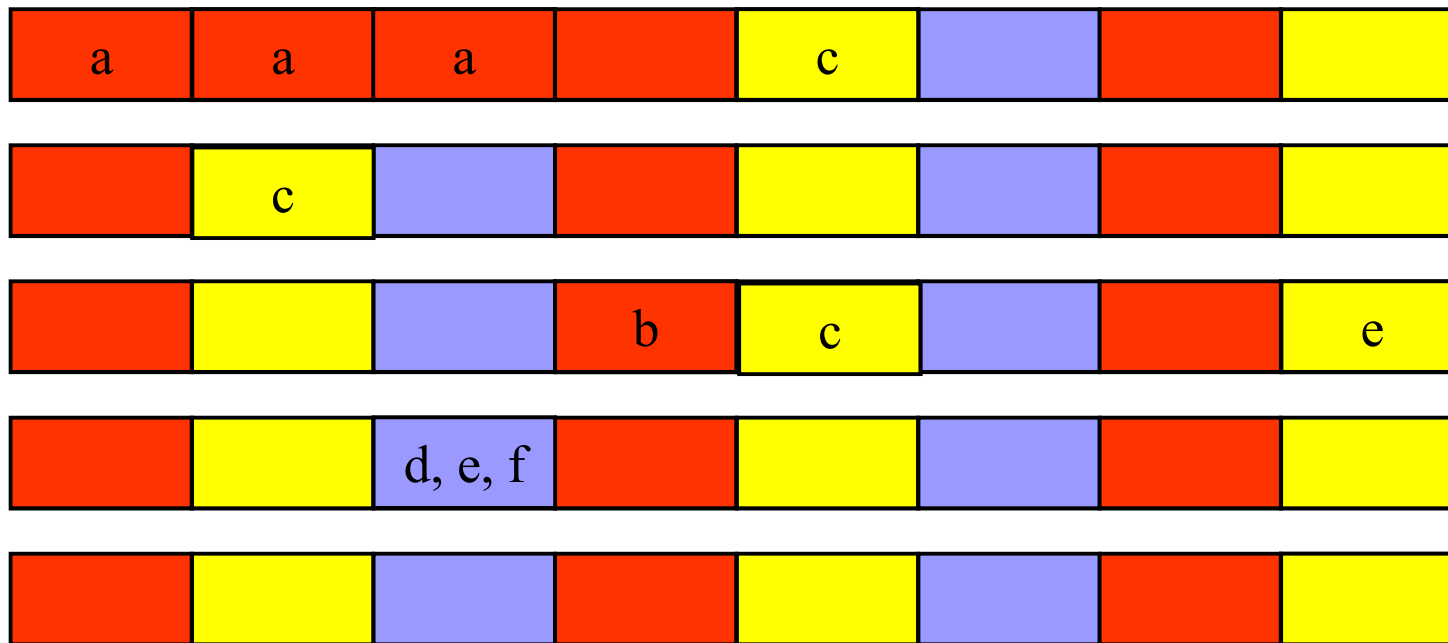
Seitenaufteilung mit festen Seiten (Paging)

Variablen eines Programms: a,b,c,d,e



Seitenaufteilung mit festen Seiten (Paging)

Variablen eines Programms: a,b,c,d,e



Virtueller Speicher / Paging

Lösung:

- Einteilen des Hauptspeichers in gleichgroße Bereiche (Seiten/Pages, Seitenrahmen)
- Pro Prozess ein „unendlicher“ Adressraum. Jeder Prozess kann jede Seite haben! Ein RAM-Array für jeden Prozess.
- Die meisten BS haben eine RAM-Pagesize von 4096 Bytes=12 Bits. Das heißt, dass die ersten 12-Bit innerhalb der Page adressieren.
- Aufteilung der virtuellen Adresse in Segment und Offset
 - 16 Bit Adresse (4/12) 16 Seiten à 4KB
 - 32 Bit Adresse (20/12) 1 Million Seiten à 4KB !
 - 64 Bit Adresse (52/12) $4 \cdot 10^{15}$ Million Seiten à 4KB !

Aufteilung der Adresse:

Beispiel:

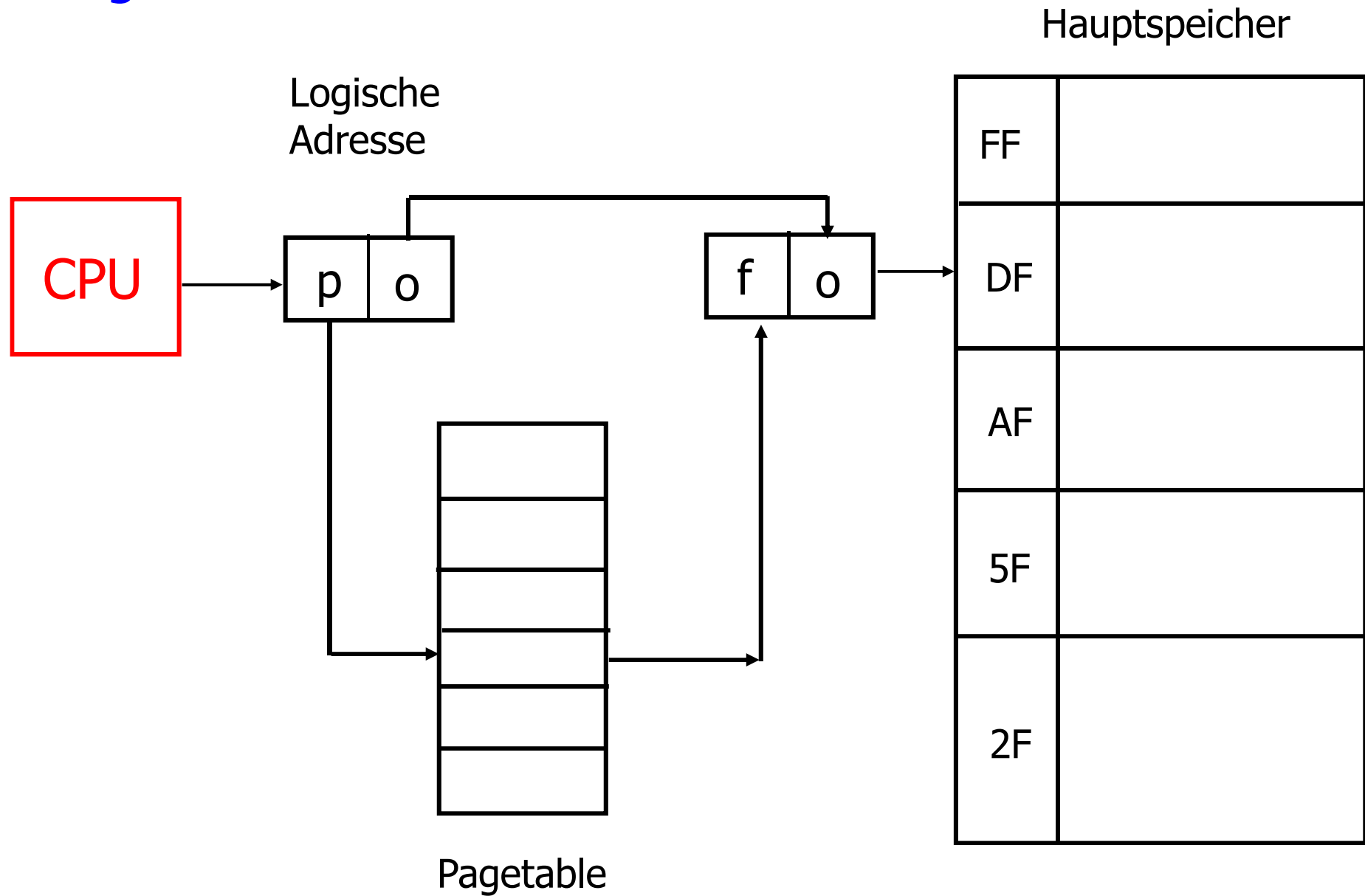
16 Bit Adresse wird aufgeteilt in

- 4 Bit Seitennummer (16 Seiten)
- 12 Bit Offset-Adresse (4096 Bytes)

echte Adr = $f(\text{seitennr}) + 12 \text{ Bit Adresse}$

es existiert eine Seitentabelle mit 16 Seiten

Pagetable



Beispiel: Pagetable

Logische
Adresse

Page0
Page1
Page2
Page3
Page4
Page5

Pagetable

P#	Nr	i,v
Page0		
Page1		
Page2		
Page3		
Page4		
Page5		
Page6		
Page7		

Hauptspeicher: Nr

8	Page 1
7	
6	Page 4
5	
4	Page 2
3	
2	Page 6
1	Page 5
0	

- Invalid, valid

Beispiel: Pagetable

Logische
Adresse

Page0
Page1
Page2
Page3
Page4
Page5

Pagetable

P#	Nr	i,v
Page0	580	i
Page1	8	v
Page2	4	v
Page3	?	i
Page4	6	v
Page5	1	v
Page6	2	v
Page7	?	i

Hauptspeicher: Nr

8	Page 1
7	
6	Page 4
5	
4	Page 2
3	
2	Page 6
1	Page 5
0	

- Invalid, Valid

Beispiel: Pagetable

Hauptspeicher: 4294971392 Bytes = 4 GB

Pages = 4096 Bytes

Anzahl der Pages: 1048577 = 1 MB

Log. Adresse Phys. Adresse

0	1566
1	155467
2	
...	
...	
...	
1048575	
1048576	

Frage:

- Speicherung der Belegung in einem Array
- Pro Feld 4 Byte als Adresse
- Wie groß ist der Speicherbedarf ?
- Auslastung des Feldes

Beispiel: Pagetable

Der virtuelle Adressraum ist in Einheiten, in Seiten, eingeteilt.

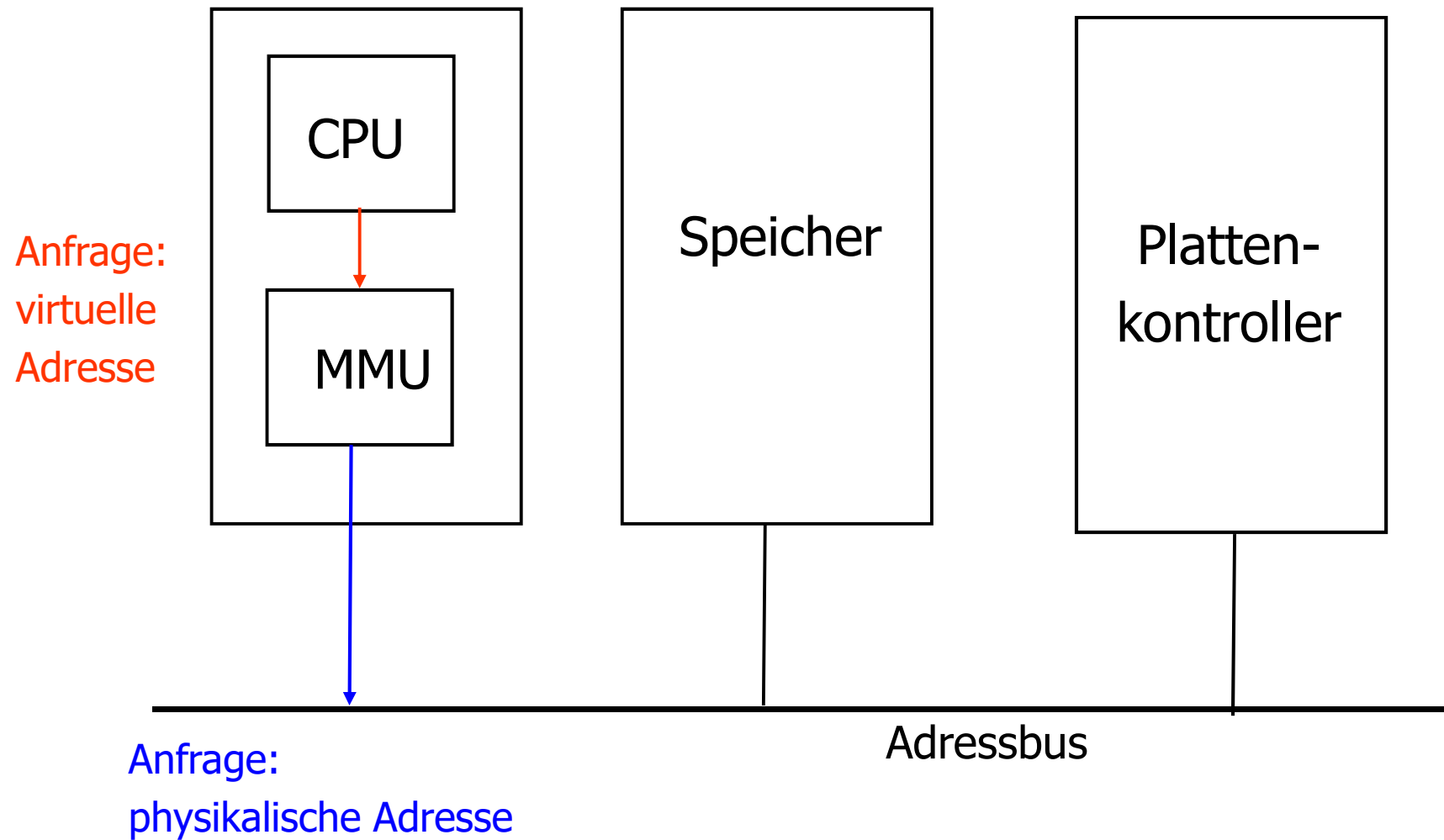
Diese physikalischen Seiten heißen Seitenrahmen.

Übliche Größe 512 Bytes bis 8 kB (meistens 4 kB).

Programmseiten haben einen Present/Absent-Bit, welches für die „memory management unit“ (MMU) als Kennung dient, ob auf diese Seite im Hauptspeicher zugegriffen werden kann.

Ist die angeforderte Seite nicht im Hauptspeicher, so wird eine Unterbrechung - „Page Fault“ - ausgelöst. Es muss eine Seite ausgelagert werden und die geforderte eingelagert werden.

Memory Management Unit (MMU):



1. Lösung des Problems des großen Arrays:

Begrenzung des Speichers pro Prozess auf 1 GigaByte

30 Bit Adressraum

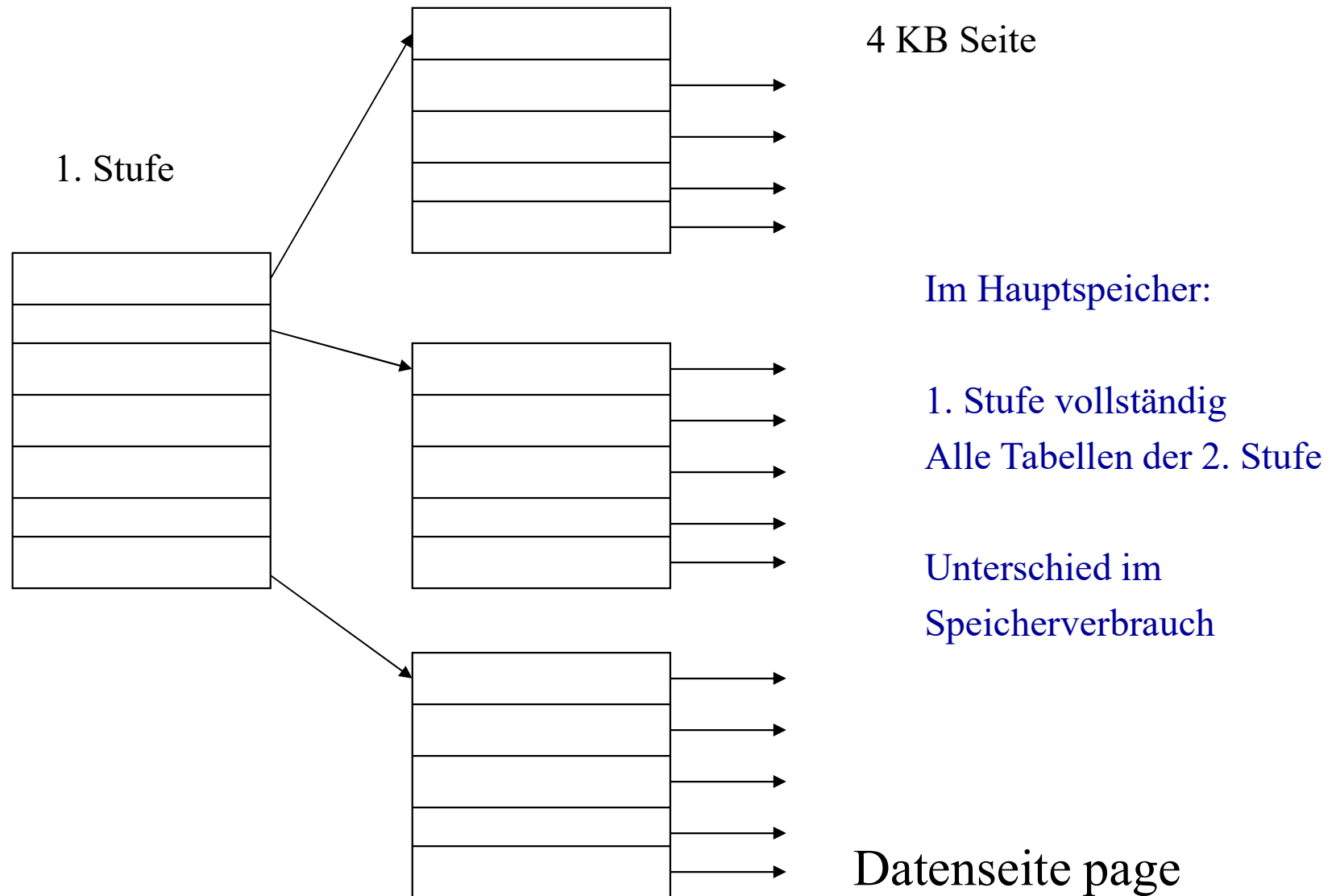
→ 18 Bit für Seitentabelle (262144 Einträge)

→ 1 MB Speichertabelle (4Byte pro Eintrag)

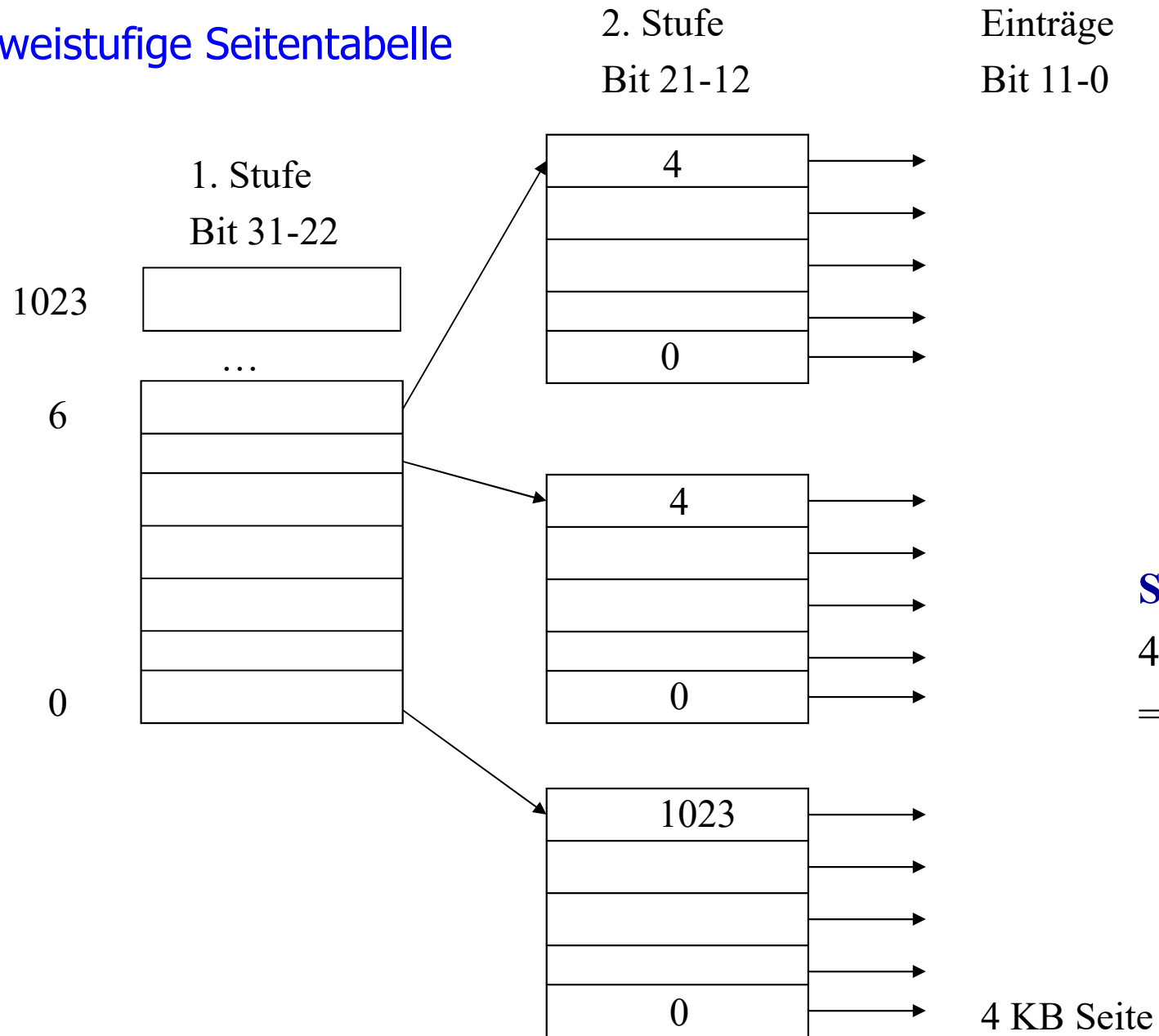
Für einen Prozess mit 1 GB Hauptspeicher ist diese Lösung vertretbar.

Für einen kleinen Prozess von 50 KB nicht!

2. Lösung des Problems: Multi-Level-Tabellen



Zweistufige Seitentabelle

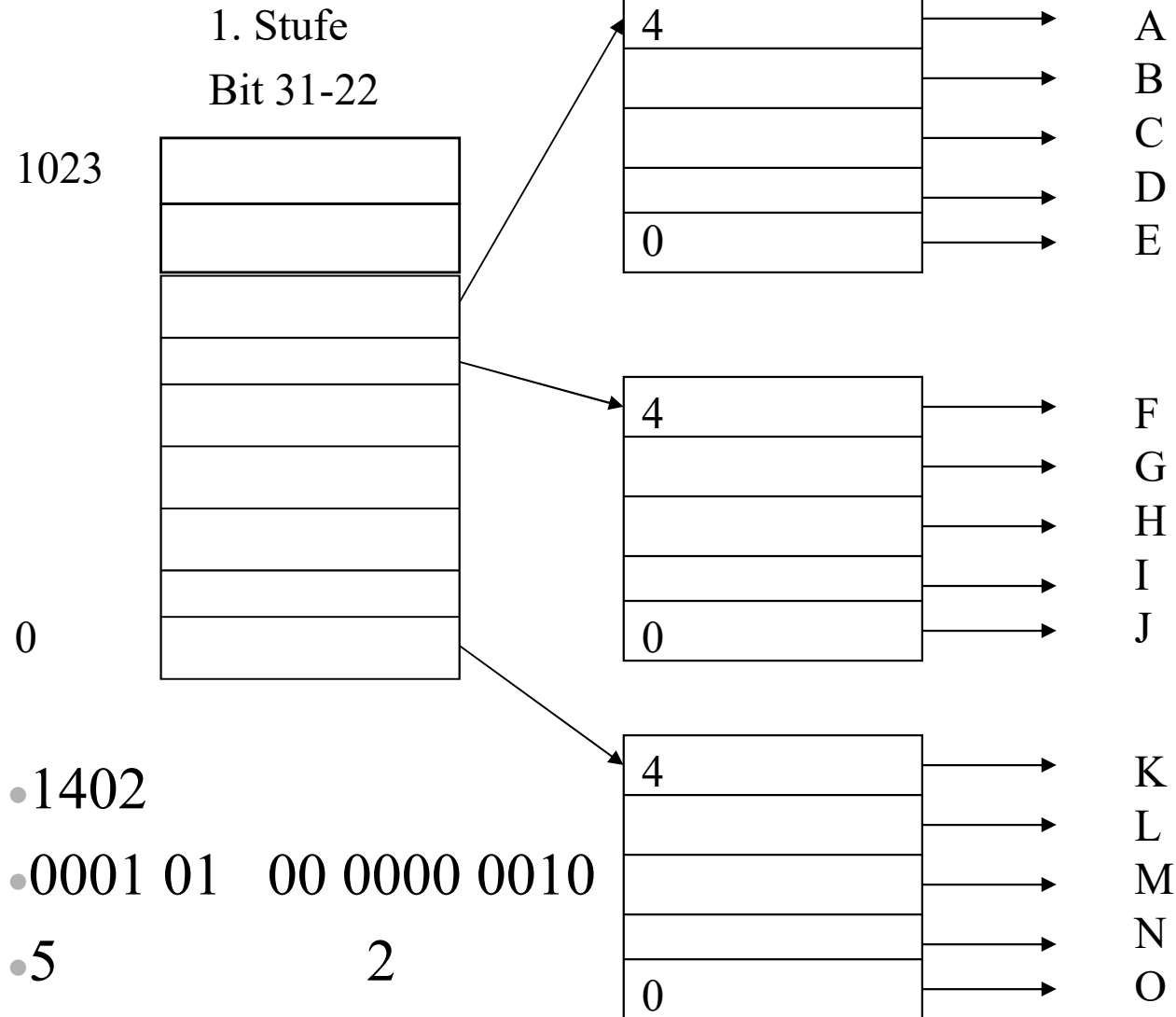


Speicherbedarf:

$$4 \cdot 1024 \cdot 4 \text{ Byte} \\ = 16384 \text{ Byte}$$

Beispiel:

20982227 = 14029D3₁₆



Beispiel:

20982227 = 14029D3₁₆

1. Stufe

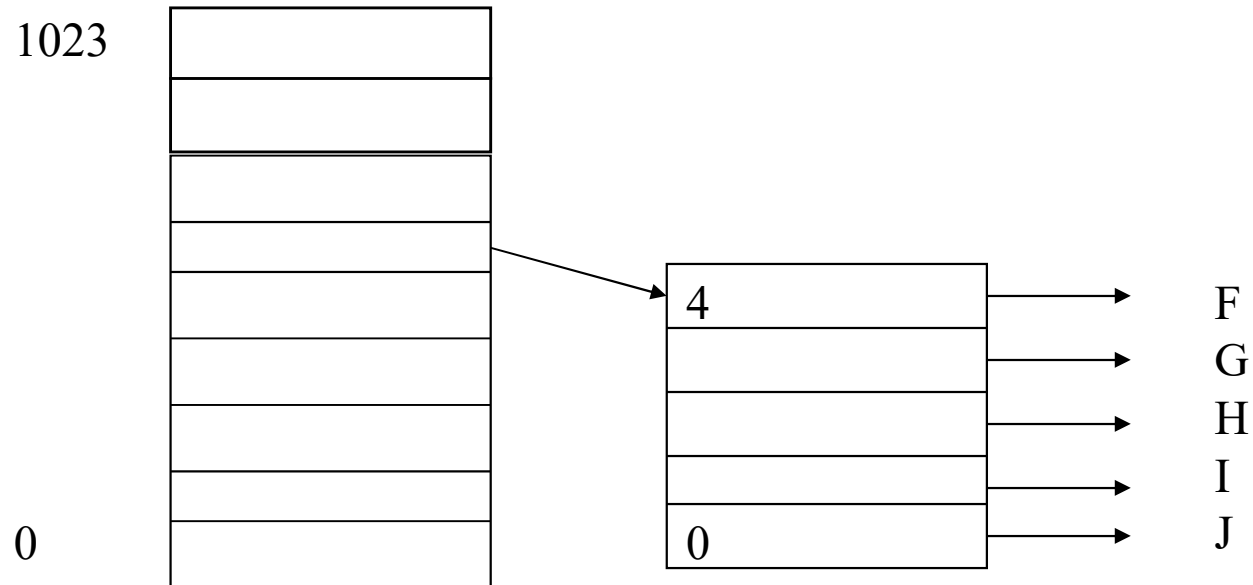
Bit 31-22

2. Stufe

Bit 21-12

Einträge

Bit 11-0



• 1402

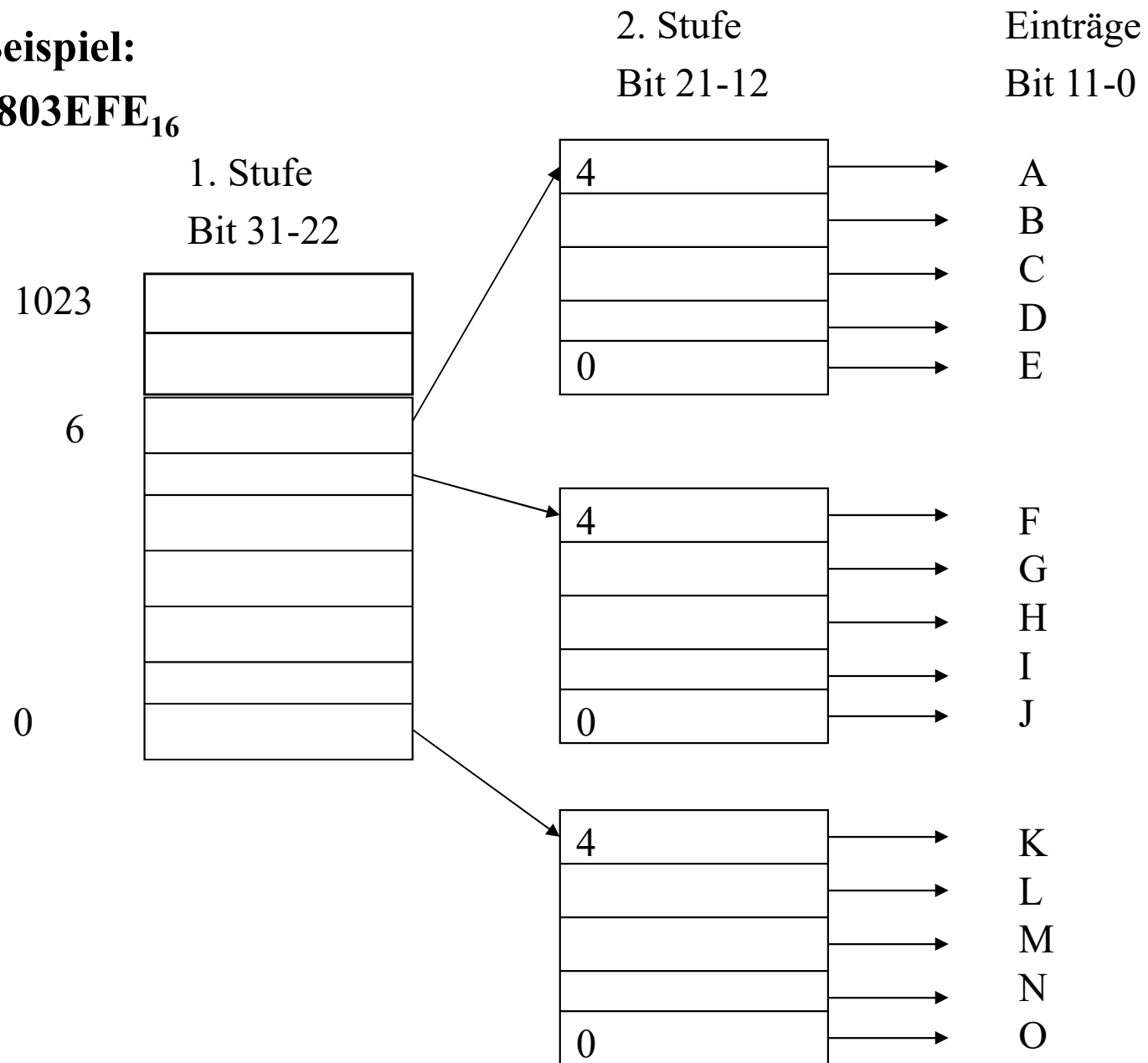
• 0001 01 00 0000 0010

• 5

2

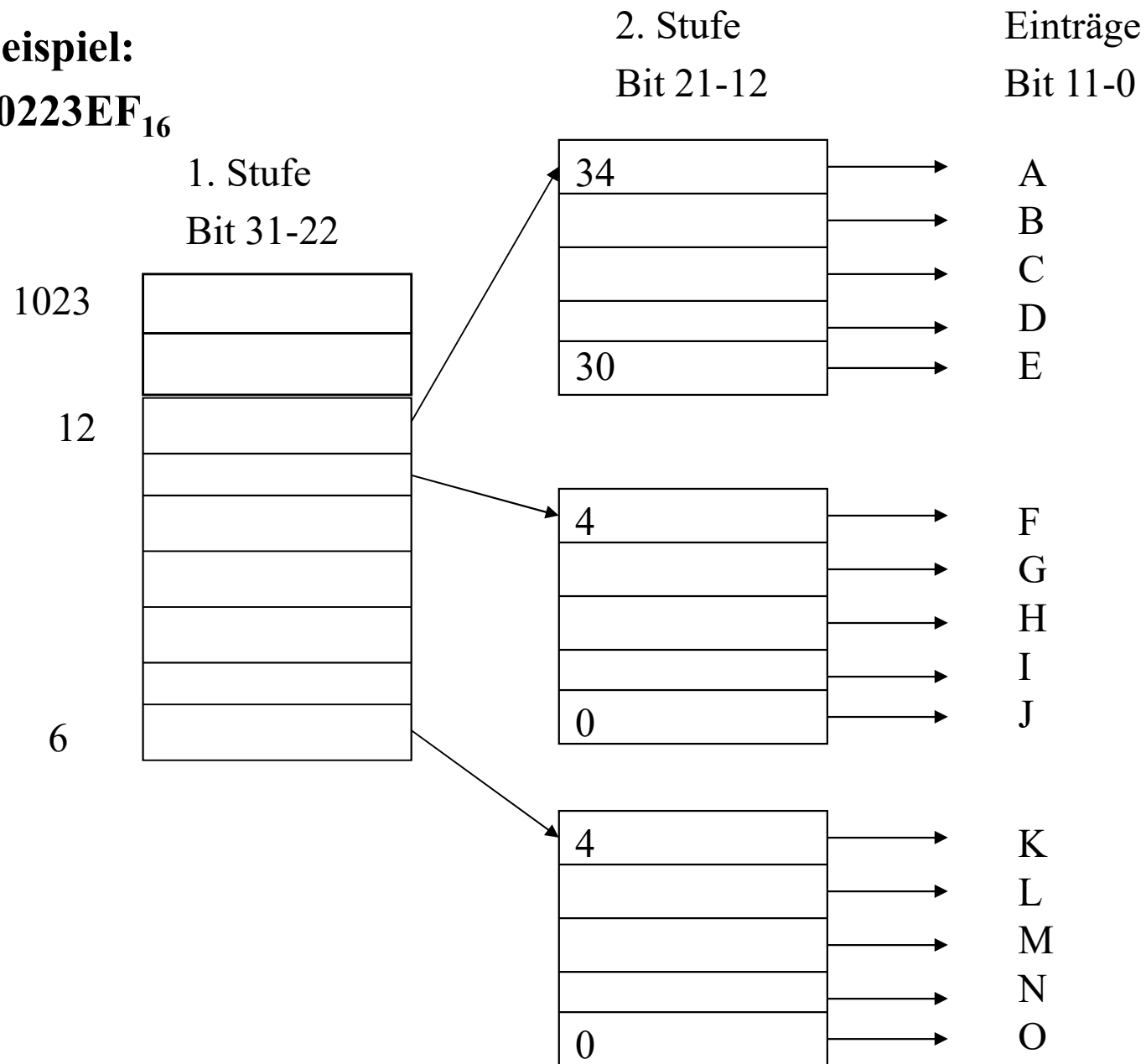
Beispiel:

1803EFE₁₆



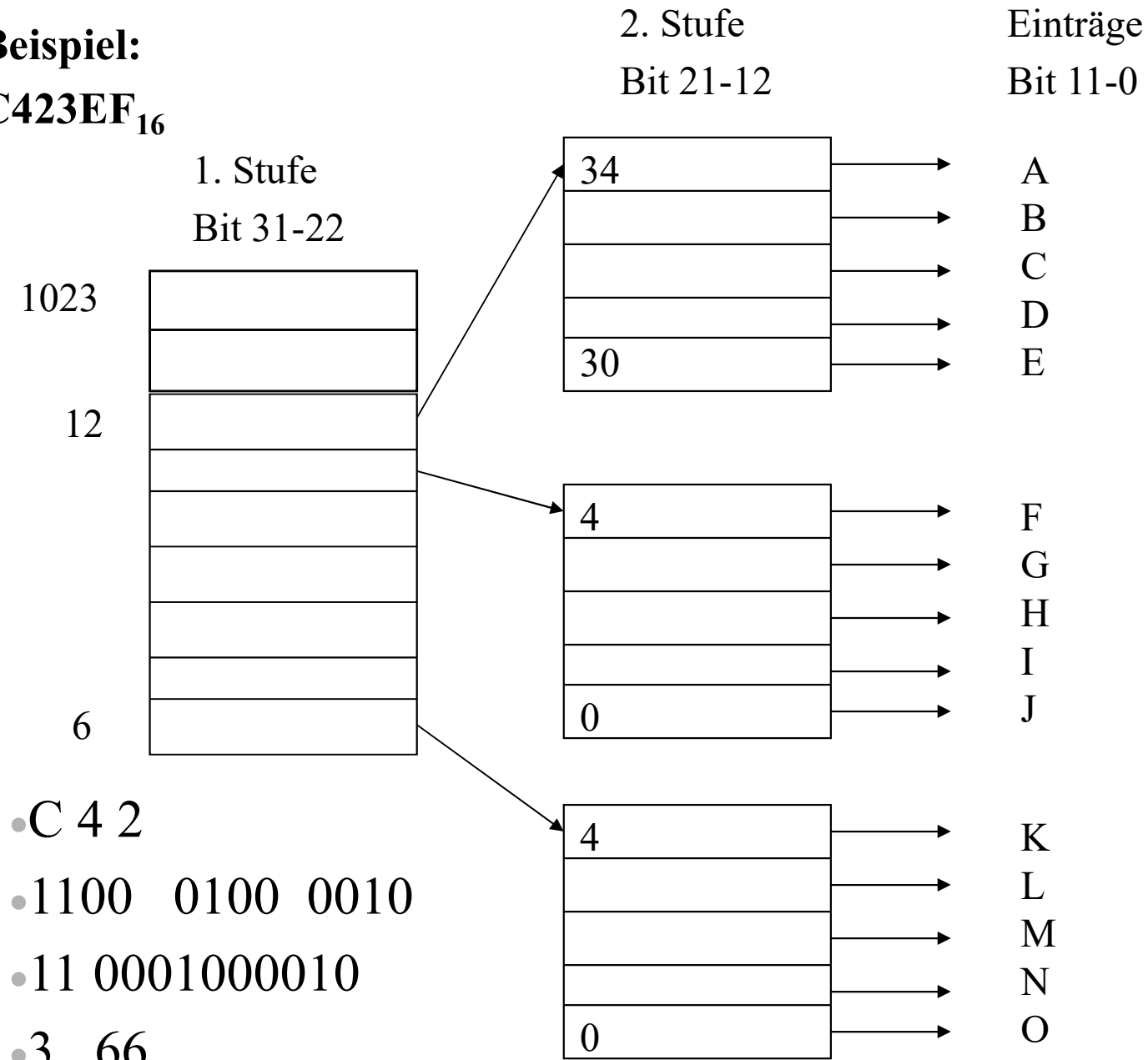
Beispiel:

30223EF₁₆



Beispiel:

C423EF₁₆



64 Bit Betriebssystem: Unix, Windows, Linux

Aufteilung der Adresse:

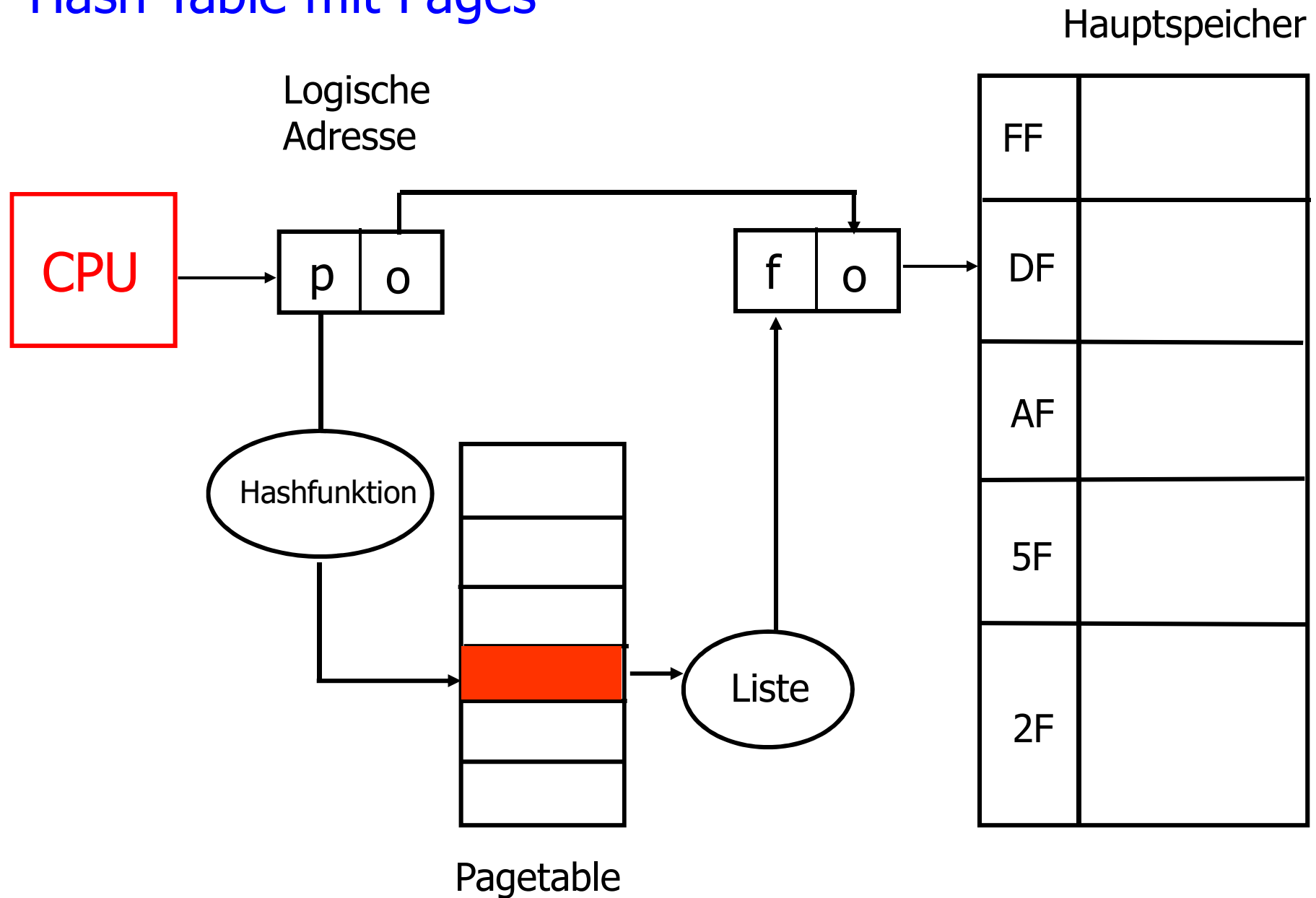
64 Bit Adresse wird aufgeteilt in

- 42 Bit 1. Seitennummer ($4,398 \cdot 10^{12}$ Einträge, 32768 TByte)
- 10 Bit 2. Seitennummer (1024 Einträge)
- 12 Bit Offset-Adresse (4096 Bytes)

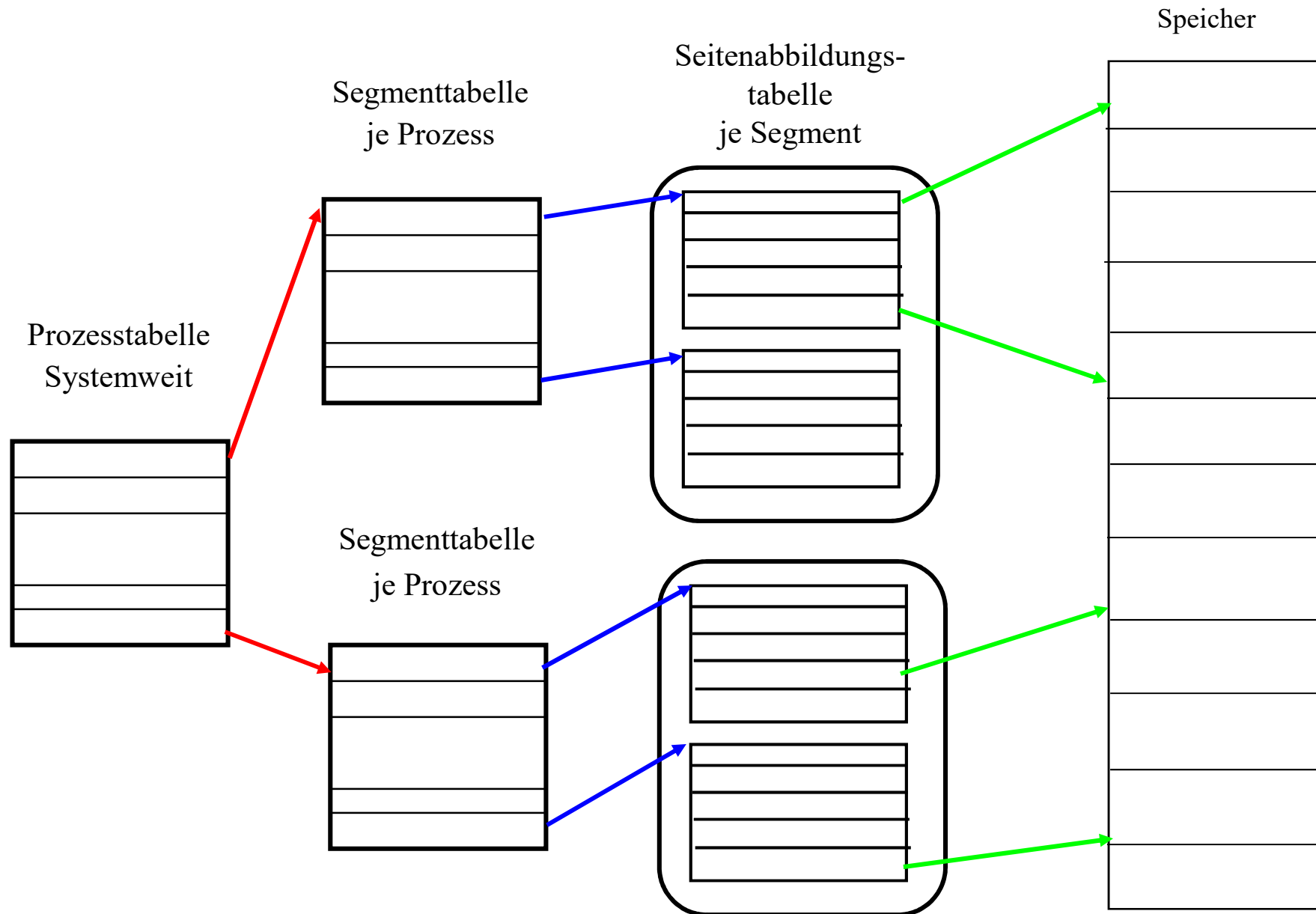
3 stufiges Adress-Schema: 32 GByte in der 3. Stufe

32 Bit	10 Bit	10 Bit	12 Bit
--------	--------	--------	--------

Hash-Table mit Pages



Vollständige Tabellenstruktur bei Paging mit Segmenten



Seitengröße

Die Seitengröße beeinflusst die interne Fragmentierung und die Größe der Seitentabelle.

Beispiel: Programmgröße: 8 KB

Seitengröße	Anzahl Seiten	Fragmentierung
1 KB	8	0 KB
4 KB	2	0 KB
8 KB	1	0 KB
16 KB	1	8 KB
32 KB	1	24 KB
64 KB	1	56 KB

Seitengröße

Bei kleinen Seitengrößen brauchen Programme viele Seiten.

Die Übertragungszeit ist weitgehend bei kleinen und großen Seiten identisch (Such- und Positionierungszeit).

Des Weiteren müssen die Seitentabellen bei Prozesswechsel mit geändert werden.

Auslagerungszeit

Seiten von 10 MB auslagern

- Übertragungszeit: $40,000 \text{ Byte /s} = 40 \text{ MB/s}$
- Positionierzeit: 8 ms
- Speicherungszeit: $10 / 40 + 8 \text{ ms} = 0,258 \text{ s}$
- Ladezeit: $10 / 40 + 8 \text{ ms} = 0,258 \text{ s}$
- Gesamtzeit: $2 * 0,258 \text{ s} = 0,516 \text{ s}$

Sun Fire 15K Starcat:

- Disk-Durchsatz (mittel): $12,0 \text{ GB/s}$
- Gesamtzeit: $2 * (10 / 12000) + 0,008 \text{ s} = 1,6 \text{ ms}$

Page Fault Rate

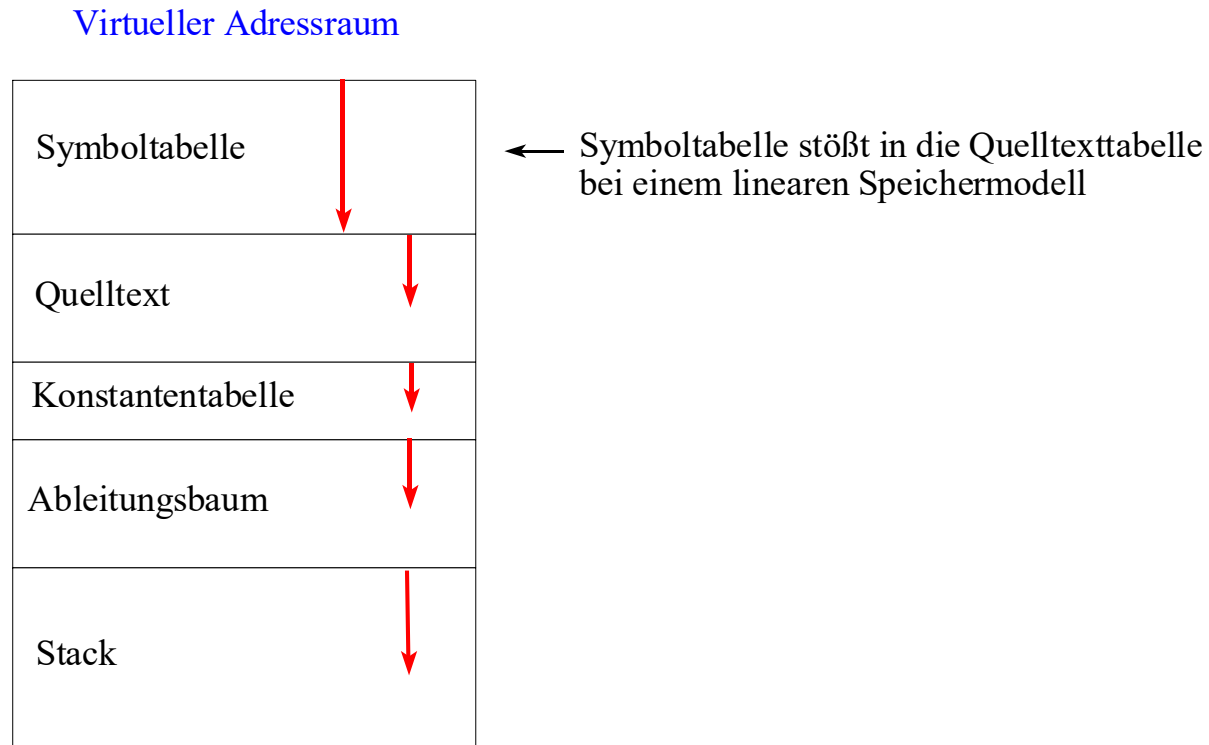
- Positionierzeit einer Festplatte: 3 ms
- Zugriffszeit einer Festplatte: 5 ms
- Hauptspeicherzugriff: 5 ns
- p ist die Wahrscheinlichkeit eines Speicherfehlers
- **mittlere Zugriffszeit: t für p = 1/1000 ?**
- $t = (1-p) \cdot 5\text{ns} + p \cdot 8\text{ms}$
- $t = (1-p) \cdot 5 + p \cdot 8000000 \quad \text{ns}$
- $t = 5 + 7999995 \cdot p \quad \text{ns}$
- $t = 5 + 7999,995 \text{ ns} = 8004,995 \text{ ns}$ (**Faktor 1601 zu 5ns**)

Trashing

- Das Betriebssystem hat zu wenig Hauptspeicherseiten
- Es muss Seiten vom Prozess P2 auslagern ($2 \cdot 1601$)
- Aktueller Prozess (P1) läuft weiter
- Der nächste Prozess braucht die ausgelagerten Seiten
- Folge: Auslagern von Seiten des Prozesses P1
- Einlesen der Seiten von P2 ($2 \cdot 1601$)
- Danach: Prozess P1 ist aktiv
- Hauptzeit erfolgt mit der Übertragung an/von der Festplatte

Segmentierung

Beispiel: Compiler mit verschiedenen Adressbereichen



Ziel: Unabhängigkeit des Programmierers von der Adressierung!

Lösung: Segmentierung

Vorher: Jeder Prozess hatte einen Speicherbereich

Nachher: Jedes Objekt hatte einen eigenen Speicherbereich

Jedes Segment besteht aus einer linearen Sequenz aus Adressen.

0 bis Maximum(i)

$$\text{adr}_{\text{MMU}} = \text{Segmentberechnung}(\text{HIPART}(\text{adr}_{\text{CPU}})) + \text{LOWPART}(\text{adr}_{\text{CPU}})$$

externe Fragmentierung kann korrigiert werden

Jedes Segment enthält ein Objekt. Schutzmaßnahmen sind damit möglich.

Welche ?

Überblick über die Segmentierung

Segment 1
Segment 2
Segment 3
Segment 4
Segment 5
Segment 6
Segment 7

Eigenschaften:

- Wachsen und schrumpfen pro Segment
- Jedes Segment hat eine individuelle Länge
- Fragmentierung
- Einsatz von Verdichtung des Speichers
7 Änderungen
Speicherverschiebung
- Inhalt eines Segments nur von einer Sorte

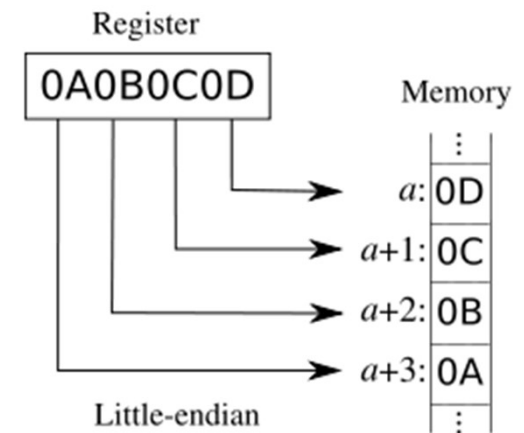
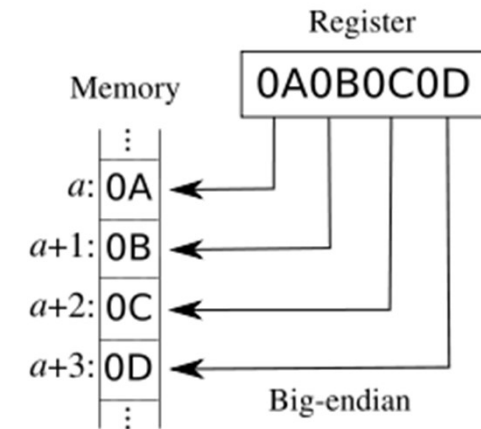
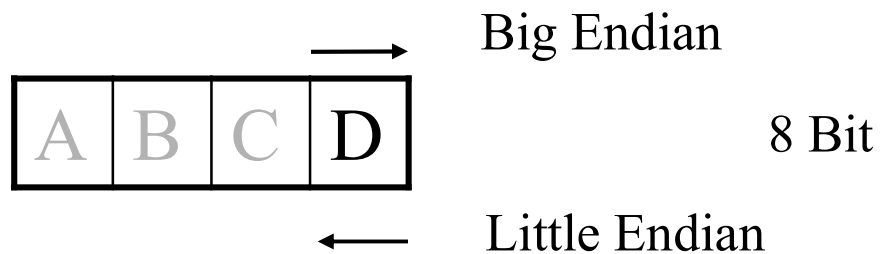
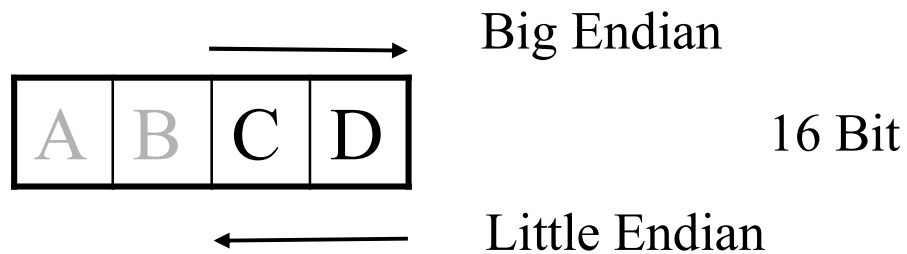
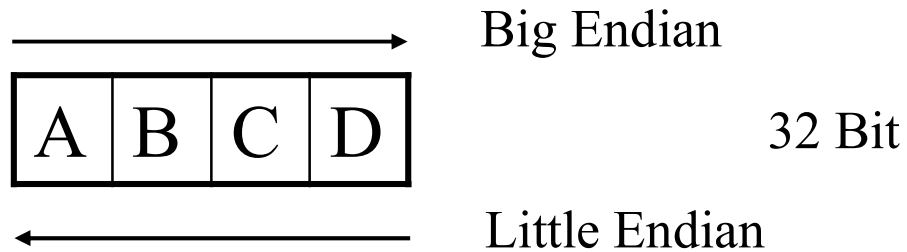
Vergleich von Paging und Segmentierung

Merkmal	Paging	Segmentierung
Muss der Programmierer sich der Adressierung bewußt sein?	Nein	ja
Anzahl der linearen Adreßräume	1	Viele
Kann der Adressraum den physikalischen Speicher übersteigen?	Ja	ja
Prozeduren und Daten getrennt?	Nein	Ja
Können dynamische Speicheranforderungen leicht erfüllt werden?	Nein	Ja
Gemeinsame Benutzung von Prozeduren unterhalb von Benutzern	Nein	Ja
Hauptgrund	Größerer Adressraum	Programm und Daten trennen

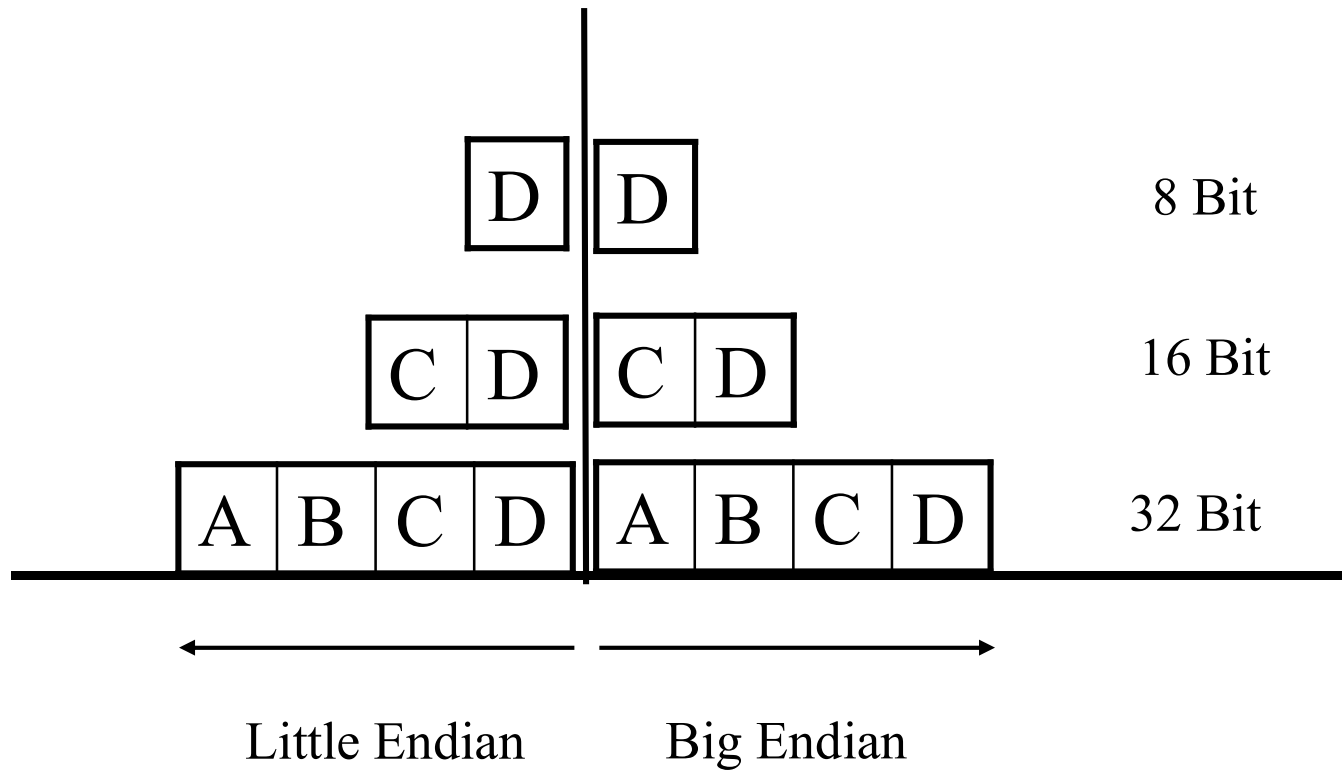
Speicheradressierung von Datentypen

- Die Datentypen `int` belegen für verschiedene Betriebssysteme eine unterschiedliche Anzahl von Bytes
 - Ältere BS $\Rightarrow$ 16 Bit pro int (-32768 bis 32767)
 - Neuere BS $\Rightarrow$ 32 Bit pro int ($\pm 2,3 \cdot 10^9$)
 - Unix/Windows $\Rightarrow$ 64 Bit pro int ($\pm 9,22 \cdot 10^{18}$)
- Wie werden die Daten im Hauptspeicher abgelegt?
 - Little Endian
 - Big-Endian

Register



Speicher



Beispiel des GIS-Datenformat Shape, Firma Esri

Table 1
Description of the Main File Header

Position	Field	Value	Type	Byte Order
Byte 0	File Code	9994	Integer	Big
Byte 4	Unused	0	Integer	Big
Byte 8	Unused	0	Integer	Big
Byte 12	Unused	0	Integer	Big
Byte 16	Unused	0	Integer	Big
Byte 20	Unused	0	Integer	Big
Byte 24	File Length	File Length	Integer	Big
Byte 28	Version	1000	Integer	Little
Byte 32	Shape Type	Shape Type	Integer	Little
Byte 36	Bounding Box	Xmin	Double	Little
Byte 44	Bounding Box	Ymin	Double	Little
Byte 52	Bounding Box	Xmax	Double	Little
Byte 60	Bounding Box	Ymax	Double	Little
Byte 68*	Bounding Box	Zmin	Double	Little
Byte 76*	Bounding Box	Zmax	Double	Little
Byte 84*	Bounding Box	Mmin	Double	Little
Byte 92*	Bounding Box	Mmax	Double	Little

* Unused, with value 0.0, if not Measured or Z type

Speicherausrichtung

- Die Ausrichtung im Speicher obliegt dem Betriebssystem
- Unterschieden wird zwischen Programmen und Daten
- Intel: Paragraph-Grenze für Segmente
- Benutzt in Klassen, struct, Records

```
struct mystruct {  
    int a;  
}
```

Länge 4 Byte

```
struct mystruct {  
    int a;  
    int b;  
}
```

Länge 8 Byte

```
struct mystruct {  
    int a;  
    char c;  
    int b;  
}
```

Länge ? Byte

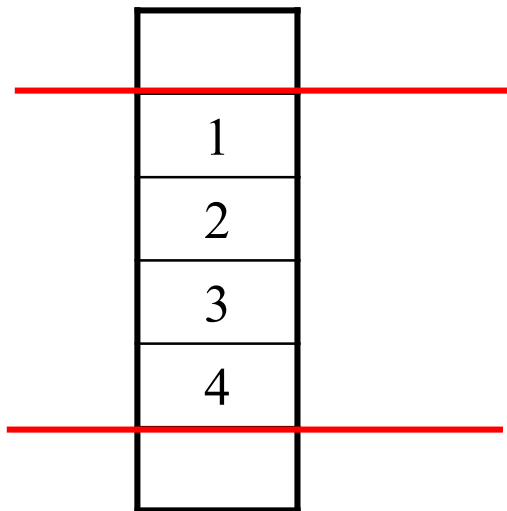
Speicherausrichtung

- Die Ausrichtung im Speicher obliegt dem Betriebssystem
- Unterschieden wird zwischen Programmen und Daten
- Intel: Paragraph-Grenze für Segmente
- Benutzt in Klassen, struct, Records
- Kann durch Compilereinstellungen verändert werden
- Vorteil
 - schnellerer Zugriff auf Daten
- Nachteile
 - langsam bei „misalignment“
 - mehr Speicherverbrauch

Speicherausrichtung

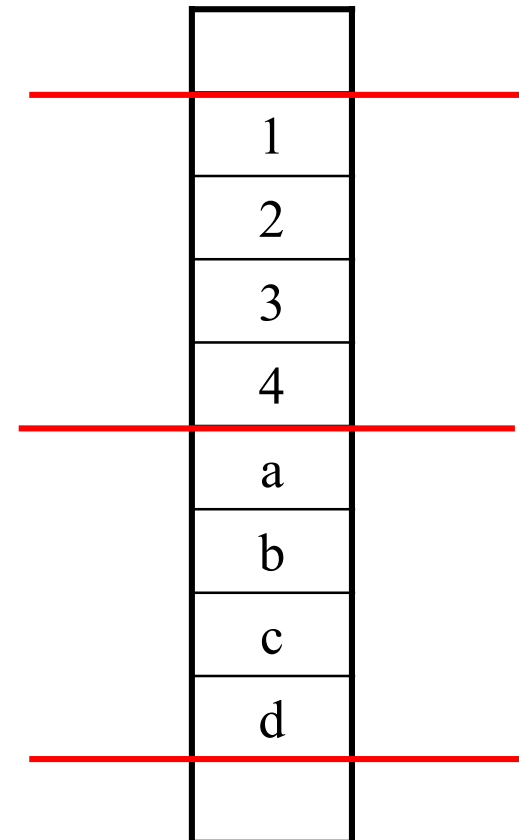
```
struct mystruct {  
    int a;  
}
```

Länge 4 Byte



```
struct mystruct {  
    int a;  
    int b;  
}
```

Länge 8 Byte



Speicherausrichtung

```
struct mystruct {  
    int a;  
    char c;  
    int b;  
}
```

Länge ? Byte

Zyklen ?

