Grundlegende Methoden der Speicherverwaltung

Partitionierung

- Speicheraufteilung zwischen verschiedenen Prozessen (Partitionierung mit festen Grenzen)

Paging

- Einfaches Paging / kombiniert mit Konzept des virtuellen Speichers

Segmentierung

- Einfache Segmentierung / kombiniert mit Konzept des virtuellen Speichers

Einfaches Paging (1)

- Zuerst wie bisher: Prozesse sind entweder ganz im Speicher oder komplett ausgelagert
- Prozessen werden Speicherbereiche zugeordnet, die nicht notwendigerweise zusammenhängend sind
- Hauptspeicher aufgeteilt in viele gleichgroße Seitenrahmen
- Prozesse aufgeteilt in Seiten derselben Größe

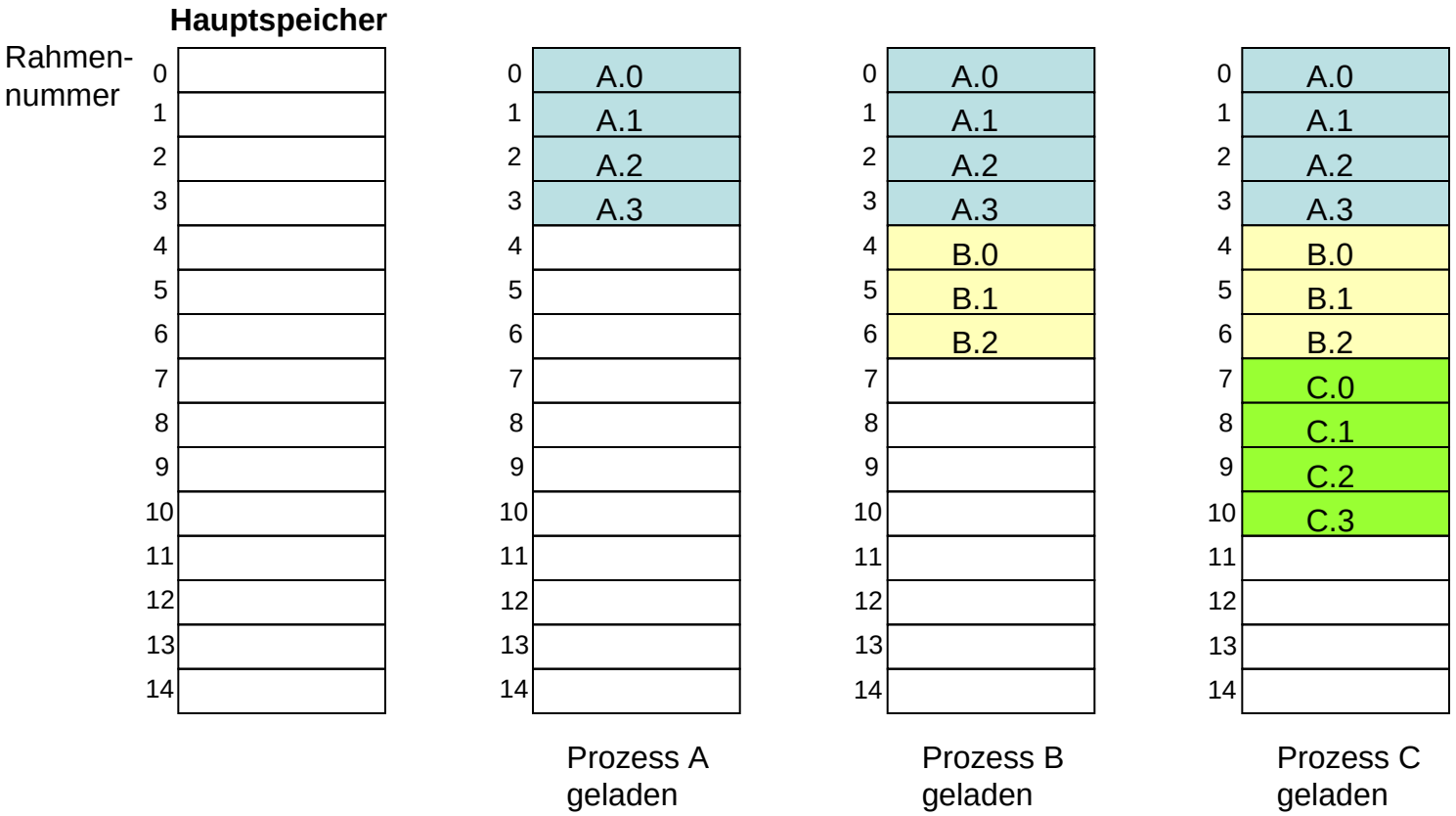
Einfaches Paging (2)

- BS verwaltet Seitentabelle für jeden Prozess
- Zuordnung von Seiten zu Seitenrahmen bei der Ausführung von Prozessen
- Innerhalb Programm: Logische Adresse der Form „Seitennummer, Offset“
- Durch Seitentabelle: Übersetzung der logischen Adressen in physikalische Adressen „Rahmennummer, Offset“

Einfaches Paging (3)

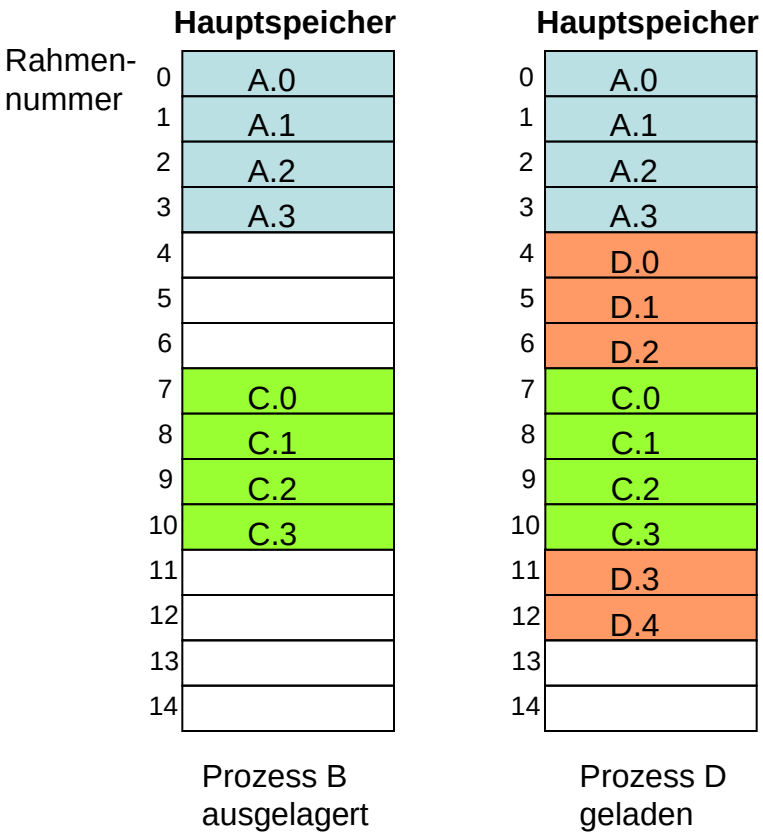
- Prozessorhardware übersetzt logische Adresse in physikalische Adresse
- Interne Fragmentierung nur bei letzter Seite eines Prozesses
- Keine externe Fragmentierung

Einfaches Paging (4)

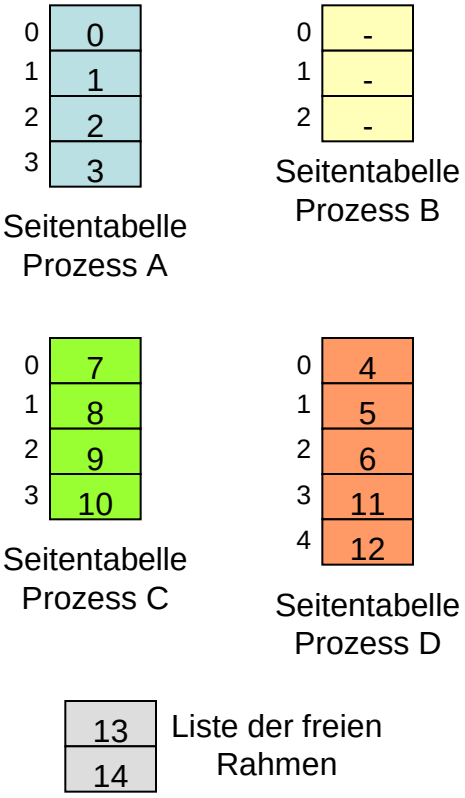


Prozess D mit 5 Seiten soll jetzt geladen werden!

Einfaches Paging (5)



Datenstrukturen zum aktuellen Zeitpunkt:



Einfaches Paging (6)

- Einfaches Paging ähnlich zum Konzept des statischen Partitionierens
- Aber: Beim Paging sind die Partitionen relativ klein
- Programm kann mehrere Partitionen /Rahmen belegen, die nicht aneinander angrenzen

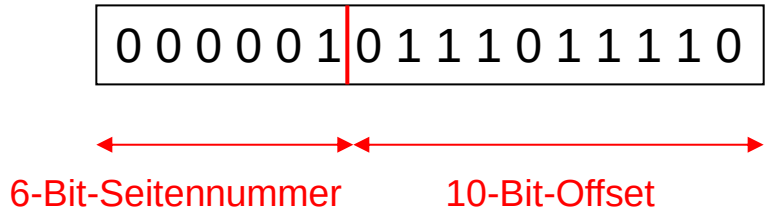
Einfaches Paging (7)

Berechnung von physikalischen Adressen aus logischen Adressen:

- Seitengröße (und Rahmengröße) ist eine Zweierpotenz
- Logische Adresse im Programm besteht aus Seitennummer und Offset
- Physikalische Adresse besteht aus Rahmennummer und Offset

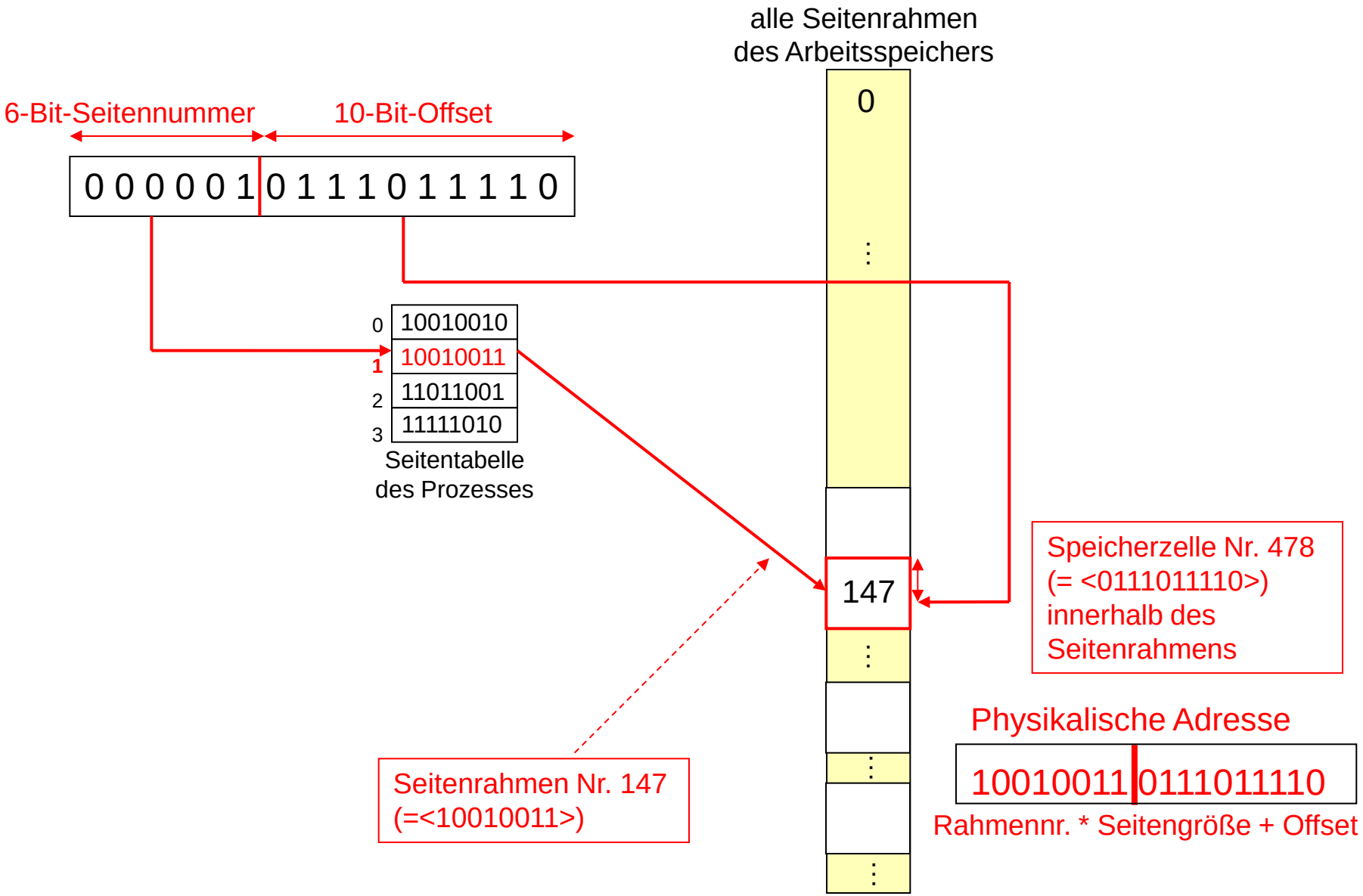
Einfaches Paging (8)

- Beispiel: Logische Adresse der Länge 16 Bit
- Seitengröße 1 KiB = 2^{10} = 1024 Bytes
- Offset-Feld von 10 Bit wird benötigt, um alle Bytes referenzieren zu können



- Der Prozess kann bis zu 2^6=64 verschiedene Seiten haben, die über die Seitentabelle des Prozesses auf Seitenrahmen im Haupt-speicher abgebildet werden

Einfaches Paging (9)



Einfaches Paging (10)

- Hauptspeicher wird in viele kleine Rahmen gleicher Größe unterteilt
- Jeder Prozess wird in Seiten geteilt, deren Größe der der Rahmen entspricht
- Seitentabelle enthält Zuordnung von Prozessseiten an Seitenrahmen des Speichers
- Interne Fragmentierung nur bei letzter Seite eines Prozesses

Einfaches Paging (11)

Entfernen eines Prozesses aus dem Speicher:

- Seitentabelle enthält Information, welche Seitenrahmen dem Prozess gehören
- Füge diese Rahmen zur Liste der freien Rahmen hinzu
- Keine zusätzlichen Datenstrukturen des Betriebssystems benötigt

Paging mit virtuellem Speicher (1)

Grundidee:

- Lagere Teile von Prozessen ein- bzw. aus anstelle kompletter Prozesse
- Programm kann auch weiter ausgeführt werden, auch wenn nur die aktuell benötigten Informationen (Code und Daten) im Speicher sind
- Bei Zugriff auf aktuell ausgelagerte Informationen: Nachladen von Seiten

Paging mit virtuellem Speicher (2)

- Hauptspeicher = realer Speicher
- Hauptspeicher + Hintergrundspeicher = virtueller Speicher
- Vorteile:
  - Platz für mehr bereite Prozesse
  - Tatsächlicher Speicherplatzbedarf eines Prozesses muss nicht im Voraus feststehen
  - Adressraum eines Prozesses kann größer sein als verfügbarer Hauptspeicher

Paging mit virtuellem Speicher (3)

Nachteile:

- Nachladen von Seiten
- Notwendiges Auslagern von anderen Seiten
- System wird langsamer

Lokalität (1)

Effizienz von virtuellem Speicher

- Typischerweise räumliche und zeitliche Lokalität von Programmen
- Zeitliche Lokalität: Nach Zugriff auf eine Speicherzelle ist die Wahrscheinlichkeit hoch, dass in naher Zukunft noch einmal darauf zugegriffen wird
- Räumliche Lokalität: Nach Zugriff auf eine bestimmte Speicherzelle gehen die Zugriffe in naher Zukunft auf Speicheradressen in der Nähe

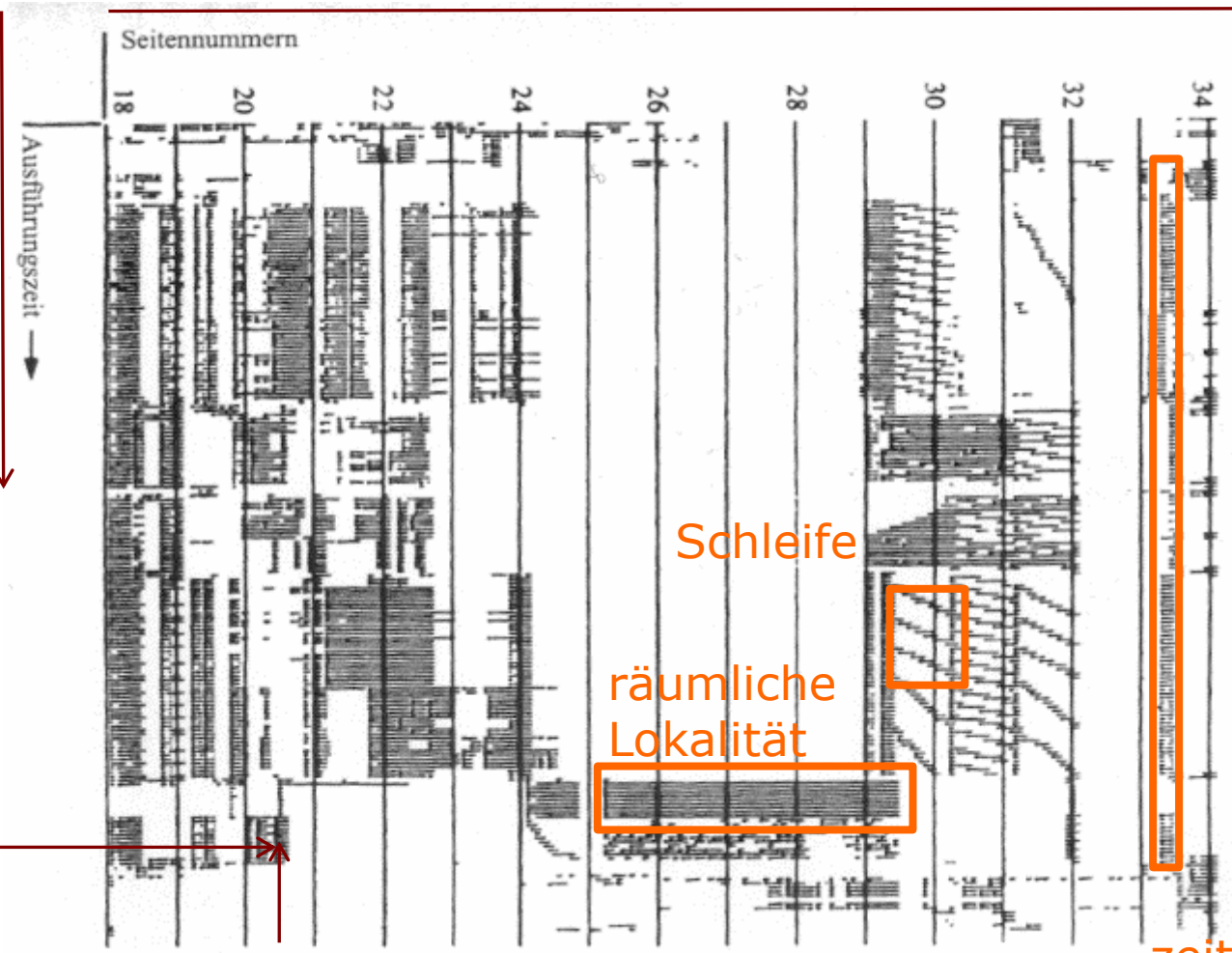
Lokalität (2)

- Die Abarbeitung während kürzerer Zeit bewegt sich häufig in engen Adress-bereichen
- Zeitliche Lokalität:
  - Abarbeitung von Schleifen
  - In zeitlich engem Abstand Zugriff auf gleiche Daten
- Räumliche Lokalität:
  - Sequentielle Abarbeitung von Programmen: Zugriffe auf benachbarte Daten
  - Lage von zusammenhängenden Daten

Lokalität (3)

zunehmende Seitennummern eines Prozesses

CPU-Zeit



zur Zeit t

wird zugegriffen auf Seite s

Lokalität (4)

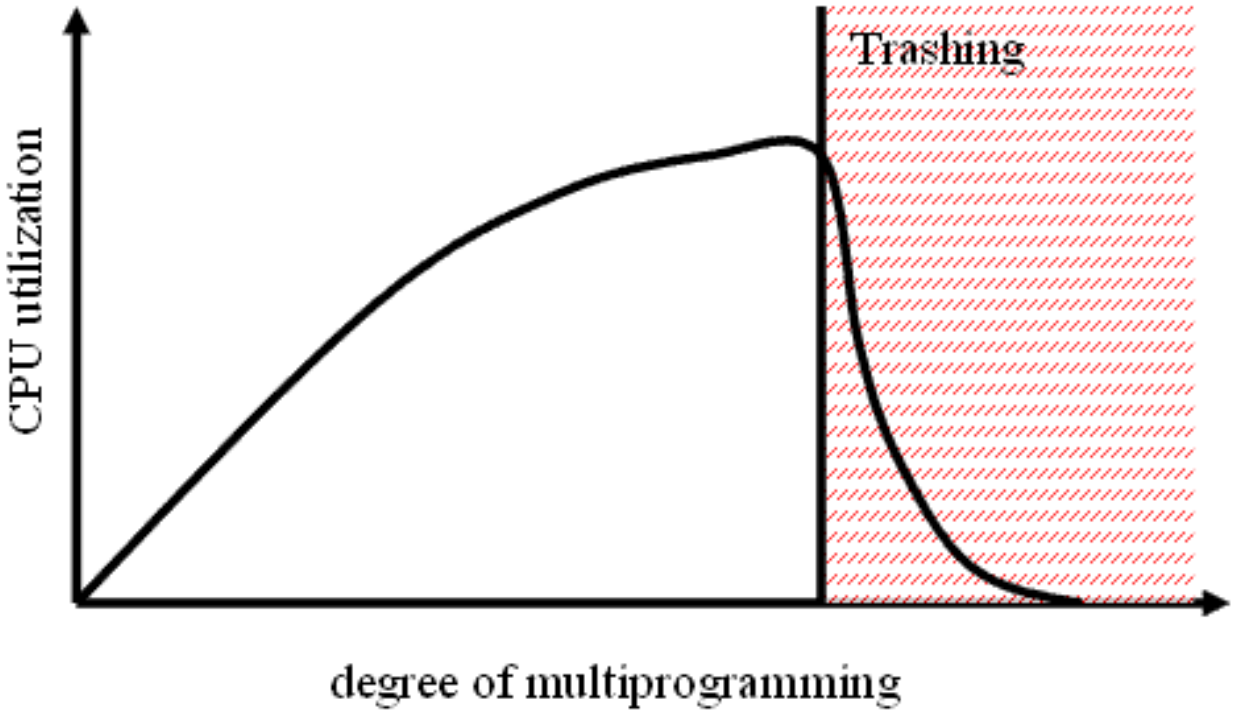
- Paging mit virtuellem Speicher ist nur dann effizient, wenn Lokalität gegeben ist
- Ansonsten Gefahr von „Thrashing“:
  - Ständiges Aus- und Einlagern von Seiten zwischen Hauptspeicher und Festplatte
  - Der Prozessor ist mehr mit Ein- und Auslagern anstatt Ausführen von Befehlen beschäftigt

Thrashing (1)

Mögliche Gründe für Thrashing:

1. Zu wenig Speicher
2. Zu viele Prozesse
3. Zu viele speicherintintensive Prozesse
4. Schlechte Ausnutzung von Lokalität
5. …

Thrashing (2)



- Um Thrashing zu vermeiden, versucht das Betriebssystem zu vorherzusagen, welche Seiten in naher Zukunft nicht benötigt werden

[Bild: http://andy.mgt.ncu.edu.tw/wordpress-zh_TW/?cat=2]

Thrashing (3)

Es hängt auch vom Programmierer ab:

```
int m[256][256];
for (i=0; i<256; i++)
      for (j=0; j< 256; j++)
            m[i][j] = 0;
```

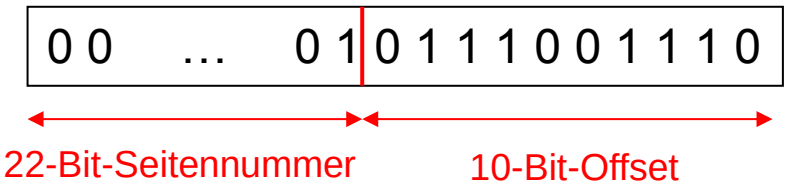
im Vergleich zu:

```
int m[256][256];
for (j=0; j<256; j++)
      for (i=0; i< 256; i++)
            m[i][j] = 0;
```

Während die erste Version die Lokalität ausnutzt, hat die zweite eine deutlich schlechtere Effizienz, weil sie ständig zwischen weiter entfernten Speicherbereichen springt.

Technische Realisierung (1)

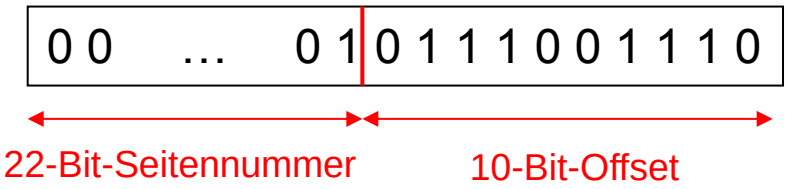
- Einfachstes Modell: Prozess (Daten+Code) befindet sich im Hintergrundspeicher
- Bei teilweise eingelagerten Prozessen: Zusätzlich Teile im Hauptspeicher
- Logische Adressen überdecken kompletten virtuellen Adressraum
- Wie bei einfachem Paging: Trennung der logischen Adressen in Seitennummer und Offset



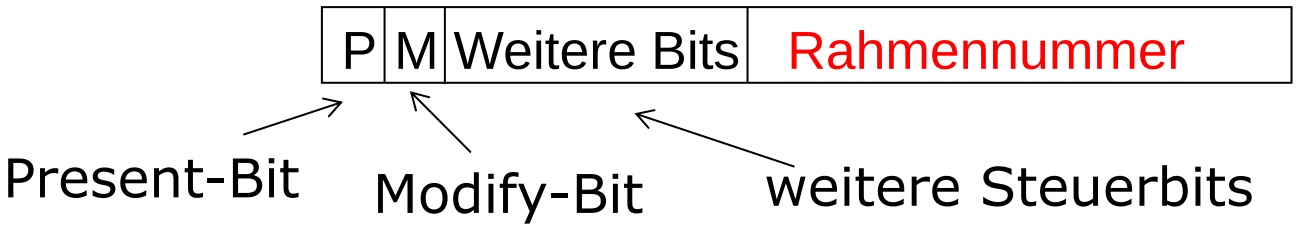
Bsp.: 32-Bit-Adresse
1 KiB Seitengröße

Technische Realisierung (2)

- Zusätzliche Informationen in Seitentabelle:
  - Ist die Seite im Hauptspeicher präsent?
  - Wurde der Inhalt der Seite seit letztem Laden in den Hauptspeicher verändert?
- Logische Adresse:



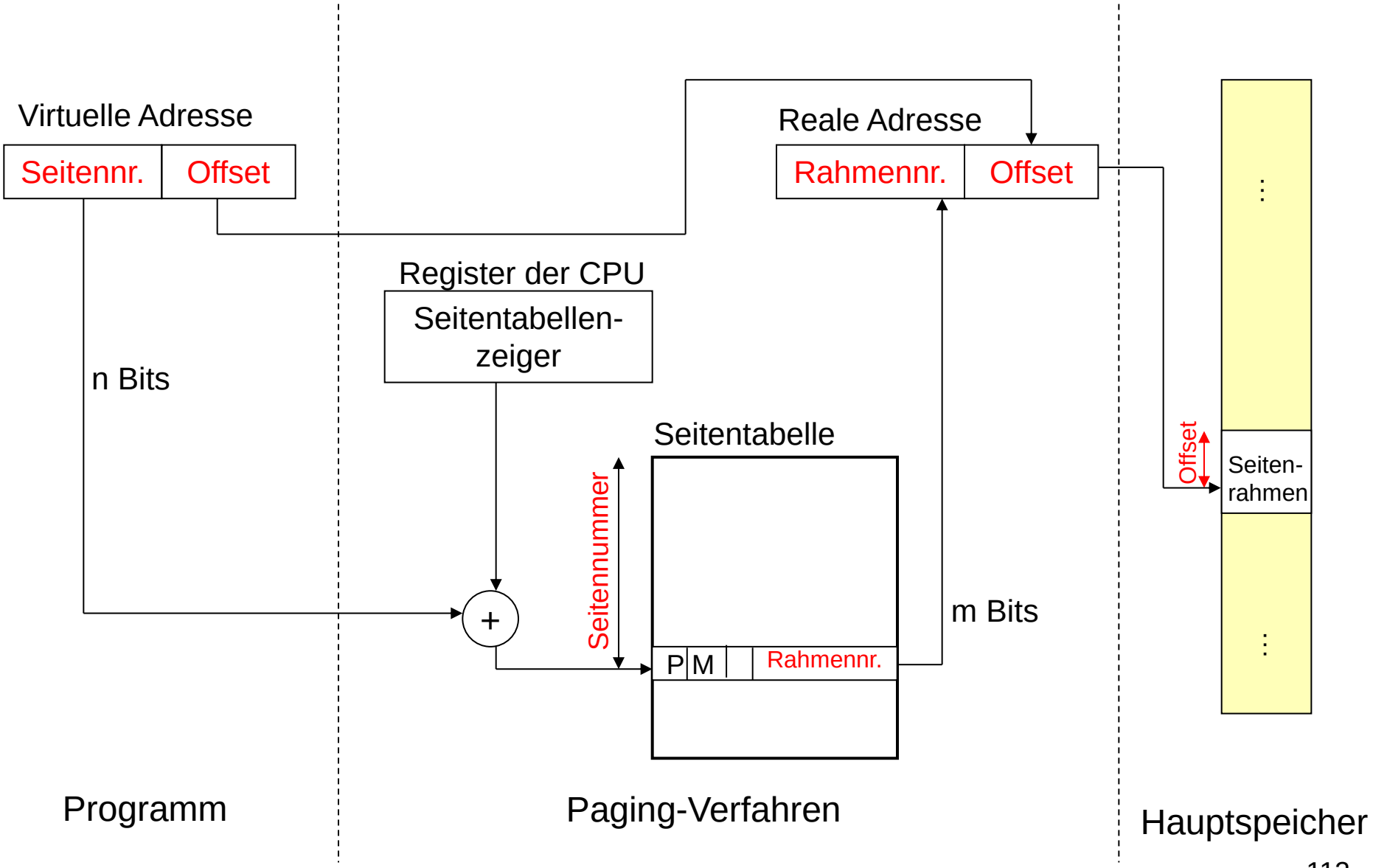
- Seitentabelleneintrag:



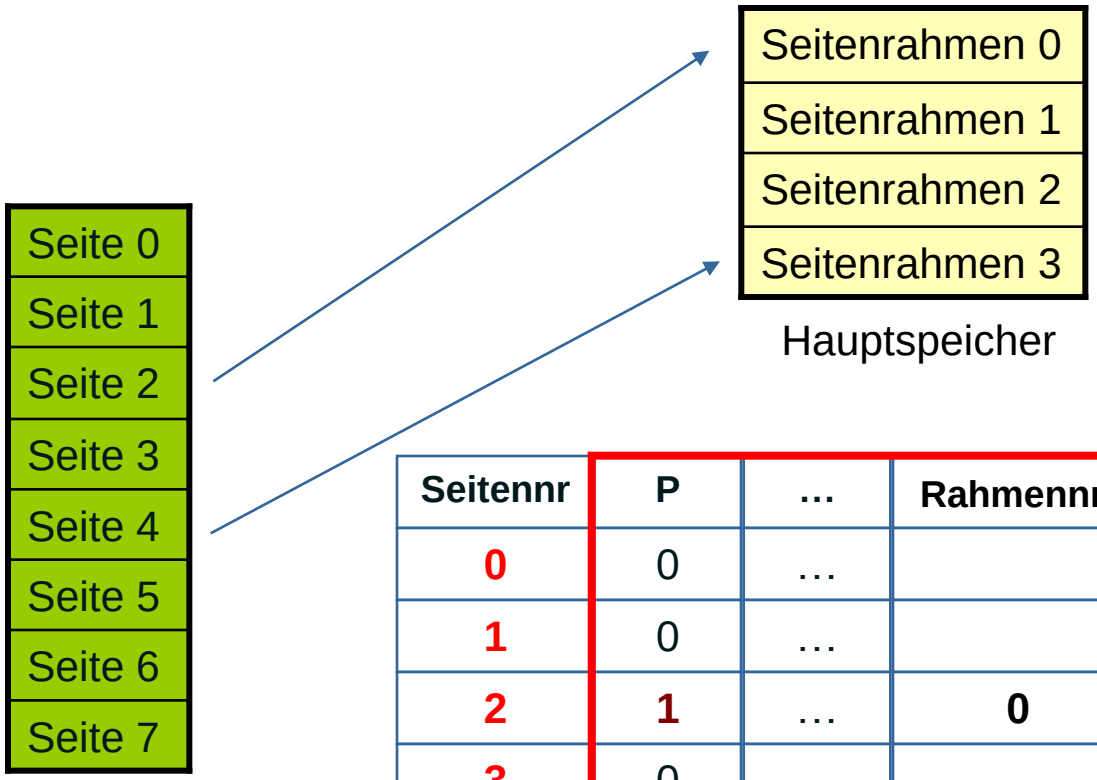
Technische Realisierung (3)

- Seitentabelle liegt im Hauptspeicher
- Umsetzung der virtuellen Adressen in reale Adressen mit Hardwareunterstützung
- Memory Management Unit (MMU) des Prozessors führt Berechnung durch

Adressumsetzung



Seitentabelle



Seitennr	P	…	Rahmennr.
0	0	…	
1	0	…	
2	1	…	0
3	0	…	
4	1	…	3
5	0	…	
6	0	…	
7	0	…	

Seitentabelle des Prozesses im Hauptspeicher

Was passiert z.B. bei Zugriff auf Seite 0?

Seitenfehler (1)

- Bei Zugriff auf Seite, die nicht im Haupt-speicher vorhanden
- Hardware (MMU) hat durch Present-Bit die Information, dass angefragte Seite nicht im Hauptspeicher ist
- Laufendes Programm wird unterbrochen, aktueller Programmzustand gesichert

Seitenfehler (2)

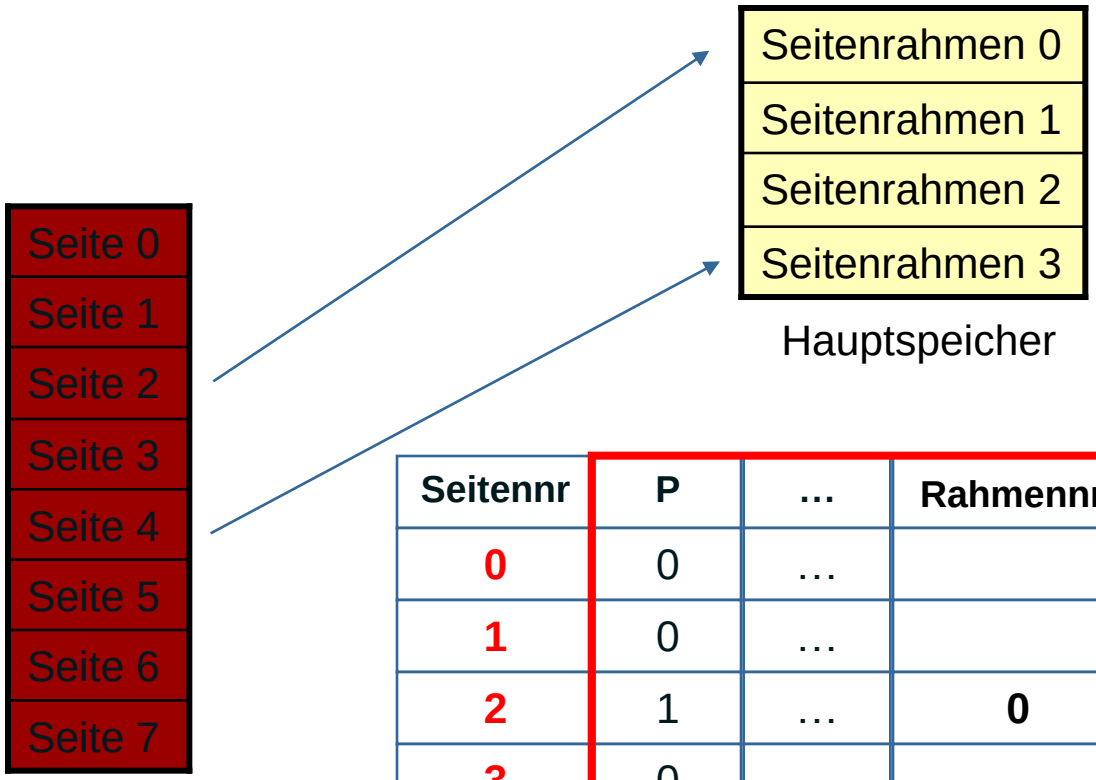
- Betriebssystem lädt die entsprechende Seite von der Festplatte in einen freien Rahmen
- Falls kein Rahmen frei: Vorheriges Verdrängen der Daten eines belegten Rahmens (beachte dabei Modify-Bit)
- Aktualisierung der Seitentabelleneinträge (Present-Bit und Rahmennummer)
- Danach kann unterbrochenes Programm fortgesetzt werden, Prozess ist rechenbereit

Seitenfehler (3)

Welche Informationen benötigt das Betriebssystem zum Einlagern von Seiten?

- Abbildung Seitennummer auf Festplatten-adresse
- Liste freier Seitenrahmen

Seitenfehler (4)



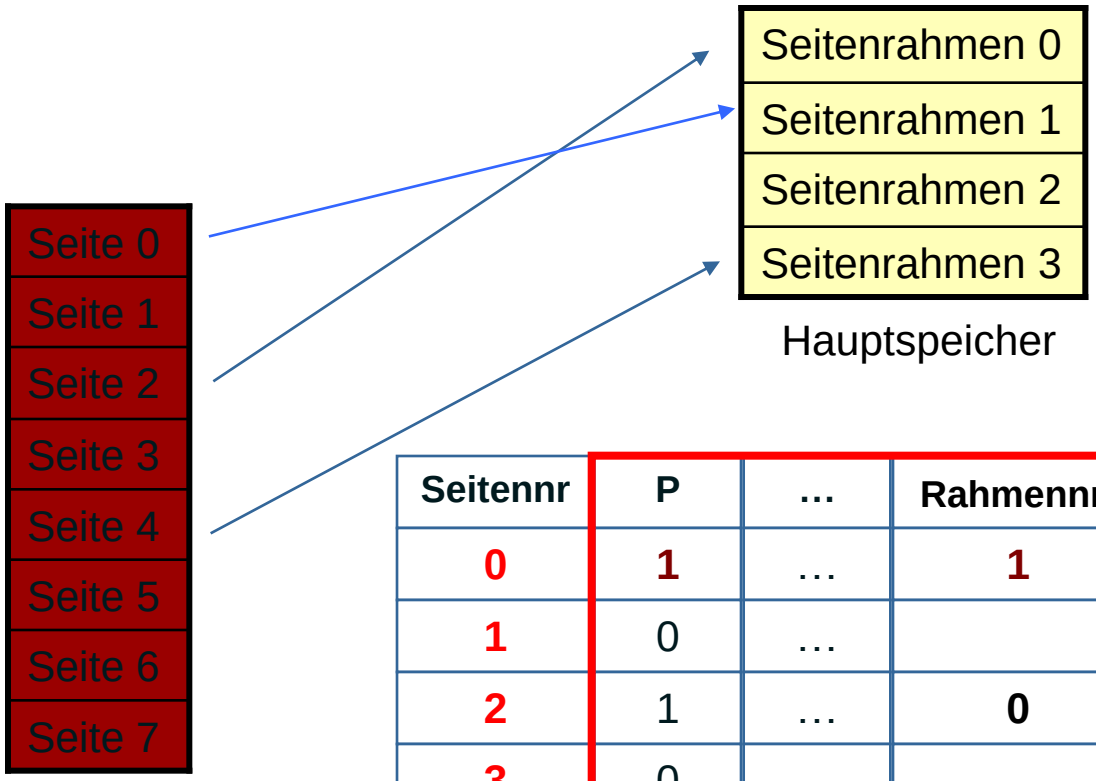
Seitennr	P	…	Rahmennr.
0	0	…	
1	0	…	
2	1	…	0
3	0	…	
4	1	…	3
5	0	…	
6	0	…	
7	0	…	

Seitentabelle des Prozesses im Hauptspeicher

Diskblock-Deskriptor

Festplatten-Adresse
A
D
B
X
Y
C
E
F

Seitenfehler (4)



Seitennr	P	…	Rahmennr.
0	1	…	1
1	0	…	
2	1	…	0
3	0	…	
4	1	…	3
5	0	…	
6	0	…	
7	0	…	

Seitentabelle des Prozesses im Hauptspeicher

Diskblock-Deskriptor

Festplatten-Adresse
A
D
B
X
Y
C
E
F

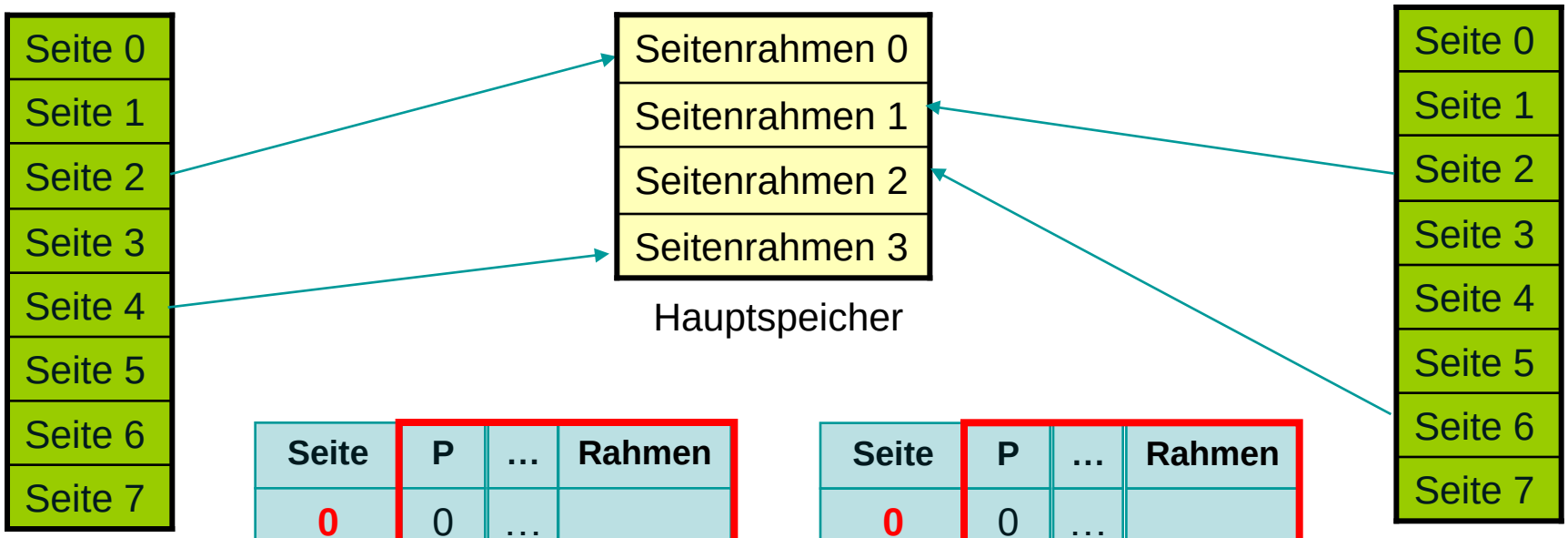
Verdrängung (1)

- Wenn kein freier Rahmen vorhanden: Verdrängen von Seitenrahmen auf die Festplatte
- Je nach Betriebssystem:
  - Alle Seitenrahmen sind Kandidaten für Verdrängung oder
  - Nur Seitenrahmen des eigenen Prozesses
- Entscheidung unter diesen Kandidaten gemäß Verdrängungsstrategie
- Ziel: Gute Ausnutzung von Lokalität

Verdrängung (2)

- Modify-Bit gesetzt: Schreibe Seite im entsprechenden Rahmen auf Festplatte zurück
- Aktualisiere Seitentabelle (P-Bit, Rahmennummer)
- Generell: Suche von bestimmten Seiten-rahmen in Seitentabelle von Prozessen ineffizient
- Weitere Tabelle: Abbildung von Seitenrahmennummer auf <Prozessnummer, Seitennummer>

Verdrängung (3)



Seite 0 von Prozess 1 soll eingelagert werden

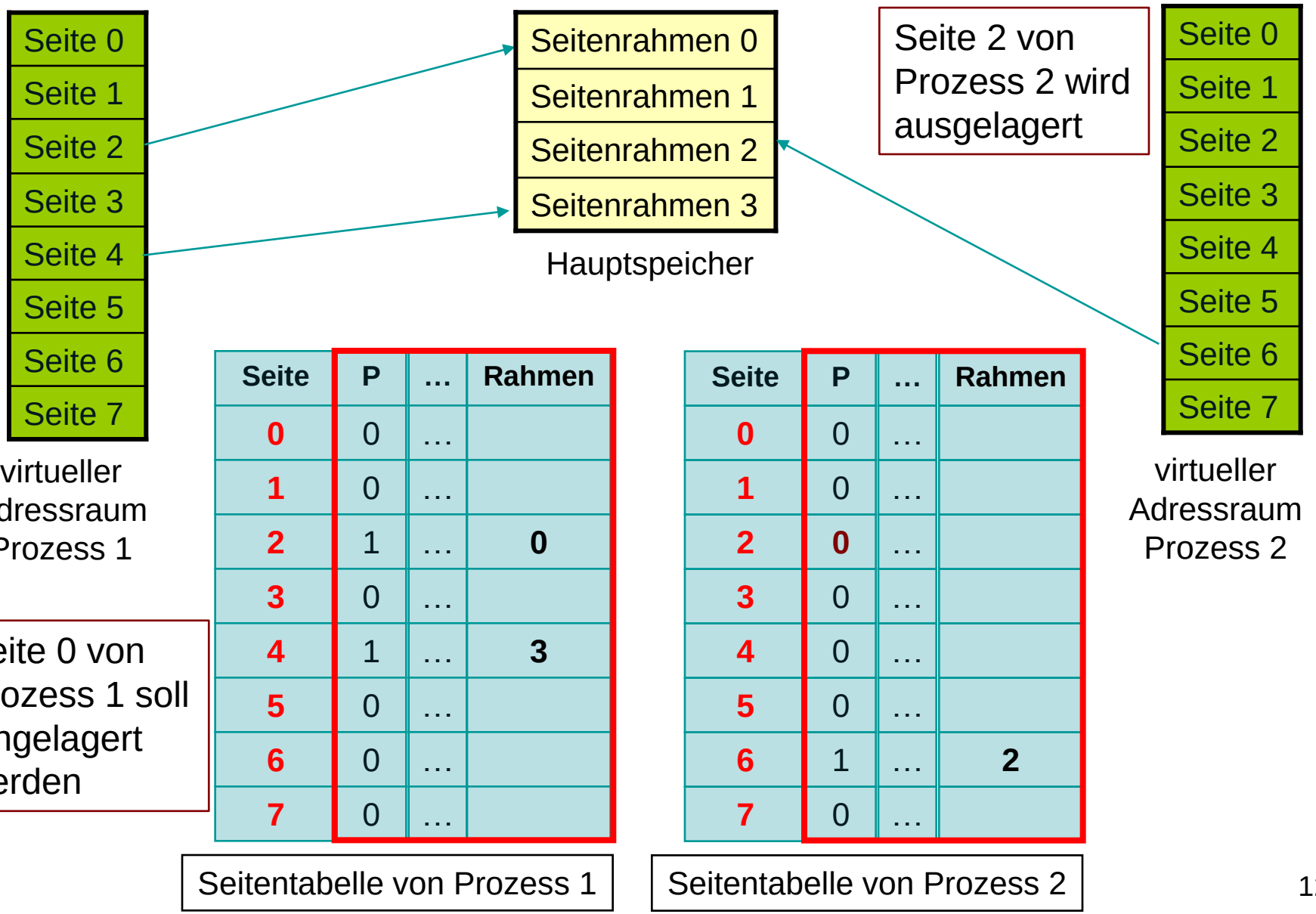
Seite	P	…	Rahmen
0	0	…	
1	0	…	
2	1	…	**0**
3	0	…	
4	1	…	**3**
5	0	…	
6	0	…	
7	0	…	

Seitentabelle von Prozess 1

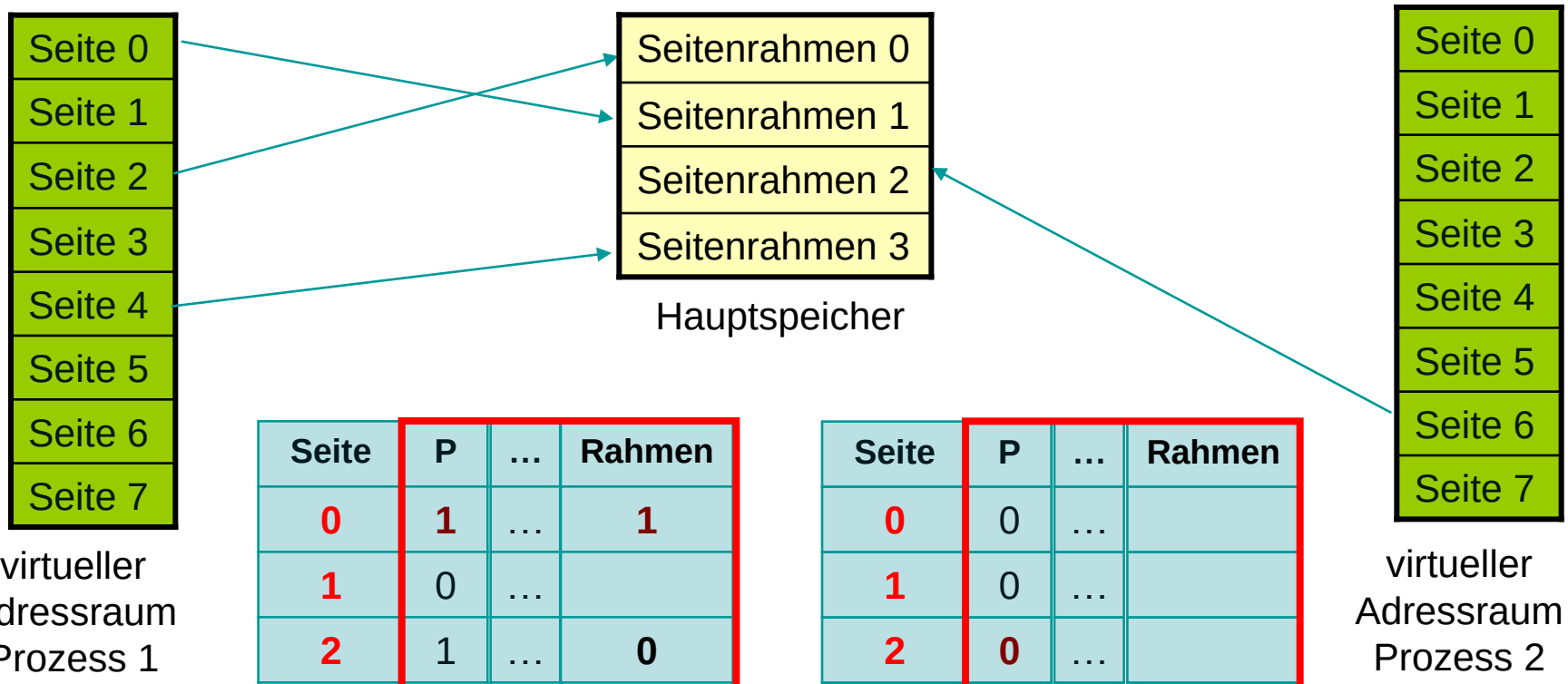
Seite	P	…	Rahmen
0	0	…	
1	0	…	
2	1	…	**1**
3	0	…	
4	0	…	
5	0	…	
6	1	…	**2**
7	0	…	

Seitentabelle von Prozess 2

Verdrängung (3)



Verdrängung (3)



Seite	P	…	Rahmen
0	**1**	…	**1**
1	0	…	
2	1	…	**0**
3	0	…	
4	1	…	**3**
5	0	…	
6	0	…	
7	0	…	

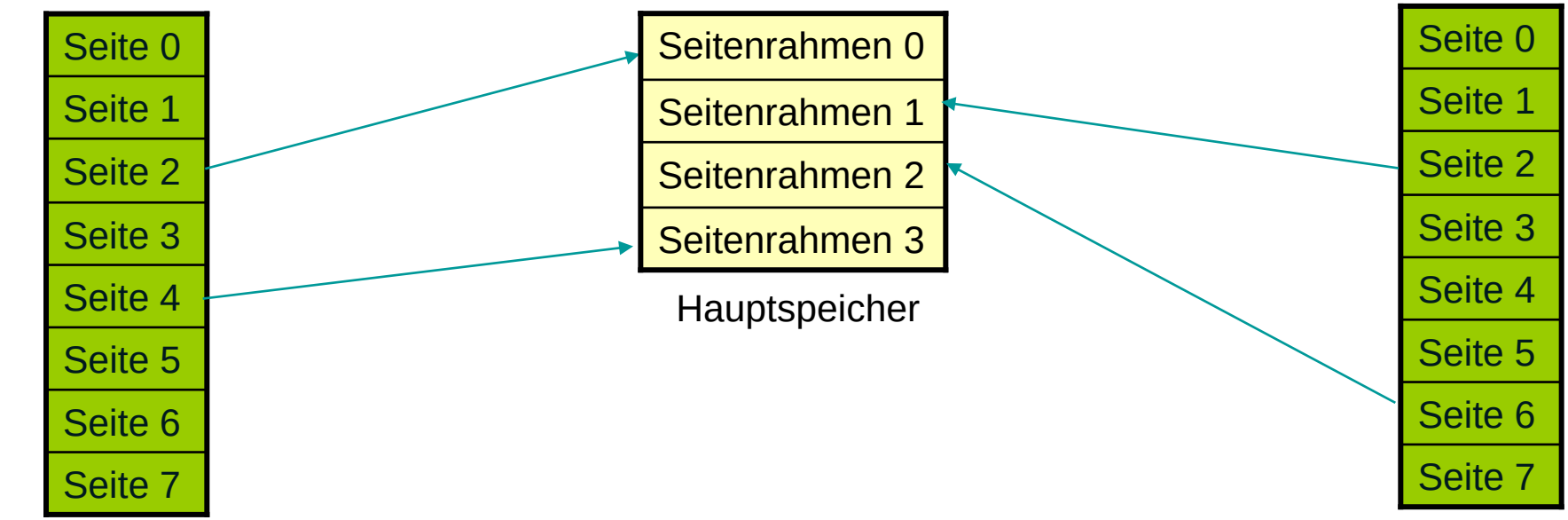
Seitentabelle von Prozess 1

Seite	P	…	Rahmen
0	0	…	
1	0	…	
2	**0**	…	
3	0	…	
4	0	…	
5	0	…	
6	1	…	**2**
7	0	…	

Seitentabelle von Prozess 2

Seite 0 von Prozess 1 wird eingelagert

Verdrängung (4)



Rahmen	Prozess	Seite
0	1	2
1	2	2
2	2	6
3	1	4

Abbildung Seitenrahmennummer auf <Prozessnummer, Seitennummer>

Größe von Seitentabellen

- Problem: Größe der Seitentabelle bei großem virtuellen Adressraum
- Beispiel: 32-Bit-Adressraum, 4 KiB Seiten
  - 20-Bit-Seitennummer, 12-Bit-Offset
  - Also: 2^{20} Seiten der Größe 2^{12} Byte, Seitentabelle mit 2^{20} Zeilen!
  - Annahme: 4 Byte pro Zeile
  - Also: 2^{22} Byte für Seitentabelle, d.h. 2^{10} Rahmen für Seitentabelle eines Prozesses im Hauptspeicher benötigt

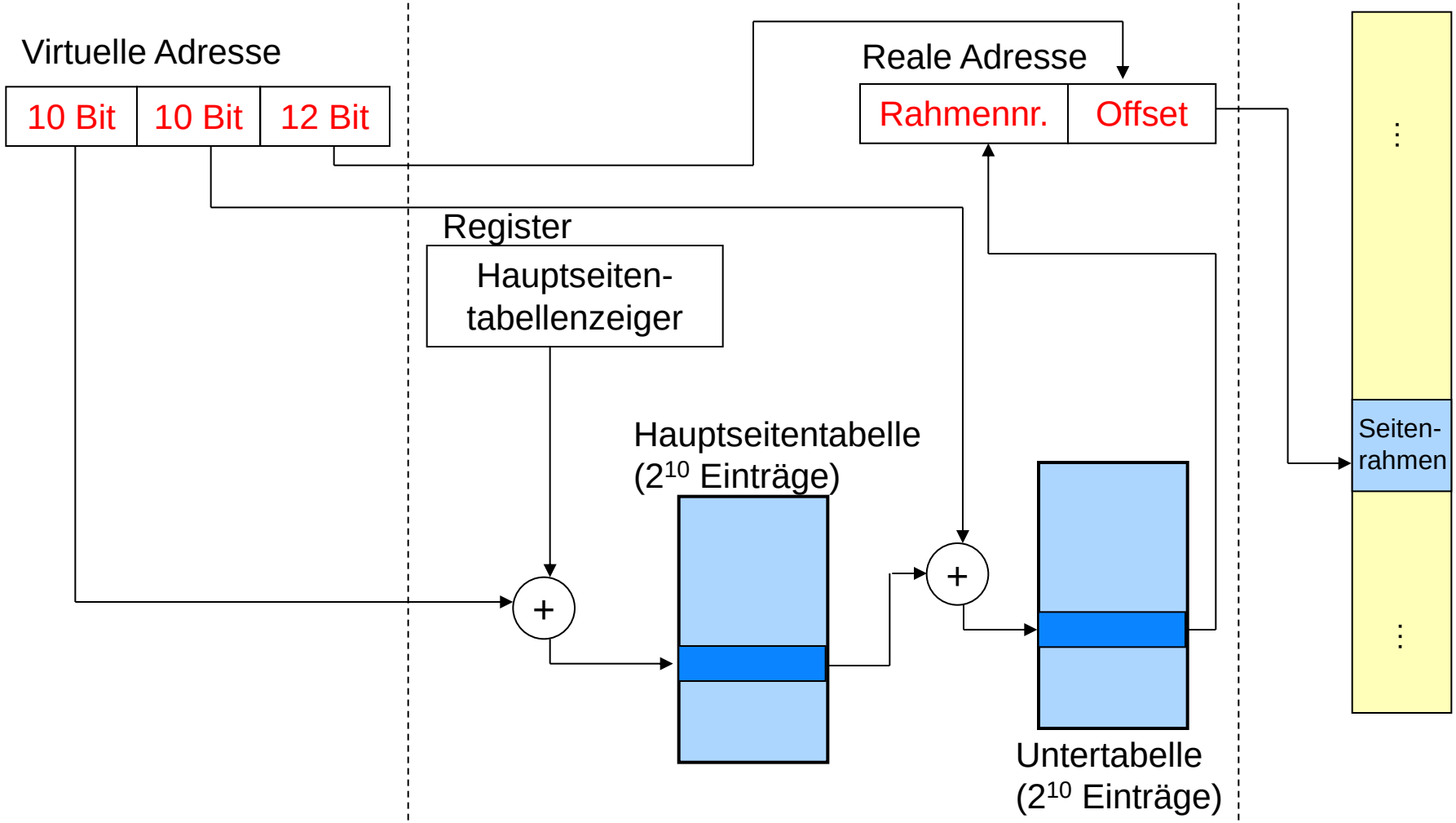
Zweistufige Seitentabellen (1)

- Hierarchische Seitentabelle
- Idee: Speichere auch Seitentabelle im virtuellen Speicher
- Im Beispiel: 2^{20} Seiten müssen angesprochen werden, also werden 2^{10} Rahmen für Seitentabelle benötigt
- Idee: Benutze eine Hauptseitentabelle, die immer im Speicher liegt
- Diese enthält 2^{10} Verweise auf Untertabellen

Zweistufige Seitentabellen (2)

- Erste 10 Bits einer virtuellen 32-Bit-Adresse: Index für die Hauptseite, um die benötigte Untertabelle zu finden
- Wenn entsprechende Seite nicht im Speicher: Lade in freien Seitenrahmen
- Mittlere 10 Bits der Adresse: Index von Seitenrahmen in Untertabelle
- Also Referenzen auf 2^{20} Seiten möglich
- Restliche 12 Bit der virtuellen Adresse: wie vorher Offset innerhalb des Seitenrahmens

Adressumsetzung



Invertierte Seitentabellen (1)

- Alternative für noch größere Adressräume
- Viel größere Anzahl von Seiten des virtuellen Adressraumes als zugeordnete Rahmen von Prozessen
- Seitentabellen meist nur sehr dünn besetzt
- Seitentabellen zur Abbildung Seitennummer auf Rahmennummer verschwenden Speicherplatz

Invertierte Seitentabellen (2)

- Nicht für jede virtuelle Seite einen Eintrag, sondern für jeden physischen Seitenrahmen
- Speichere zu Seitenrahmen die zugehörige Seitennummer
- Unabhängig von Gesamtanzahl der Seiten der Prozesse: Ein fester Teil des realen Speichers für die Tabellen benötigt

Invertierte Seitentabellen (3)

- Nachteil: Aufwändiger, eine virtuelle Adresse auf eine physische abzubilden
- Benutze eine Hashtabelle um Seitennummern mit Rahmen zu speichern
- n = #Seiten, m = #Rahmen, Hashfunktion:
  $h : \{0, \ldots, n-1\} \rightarrow \{0, \ldots, m-1\}$
- Sei k_i Seitennummer, einfaches Beispiel:
  $h(k_i) = k_i \bmod m$
- Bei Vergabe eines neuen Seitenrahmens v_i: Speichere an Platz $h(k_i)$ das Paar (k_i, v_i) ab