BETRIEBSSYSTEME – SPEICHERVERWALTUNG

INHALT:

1. Speicherverwaltung
2. Speicherpartitionierung (Speicheraufteilung)
3. Paging und Segmentierung
4. Virtueller Speicher
5. Zusammenfassung (Skript)

SPEICHERVERWALTUNG:

- zentrale Aufgabe des BS

- Schwieriger Aspekt der BS-Entwicklung
- Uniprogramming-System zweiteiliger Hauptspeicher
  - Teil Betriebssystem
  - Anderer Teil Programme
- Multiprogramming-System Hauptspeicher mehr als zwei Teile
  - Hier kommt der Speicherverwaltung eine dynamische Aufgabe zu.

SPEICHERVERWALTUNG:

5 Anforderungen an die Speicherverwaltung:

- **Relocation** (Wiederfinden)
- **Protection** (Schutz)
- **Sharing** (Teilen/Aufteilen)
- **Logical Organization**
- **Physical Organization**

SPEICHERVERWALTUNG:

Relocation:

Ausgangssituation:

- Gemeinsame Nutzung des Hauptspeichers von verschiedenen Prozessen
- Programme werden zum Teil ausgelagert, und nur das Prozess Image der aktiven Prozesse muss im Hauptspeicher verfügbar sein.

→ Die Prozessorhardware und die Betriebssystem-Software müssen in der Lage sein, Referenzen im Programmcode in aktuelle physische Adressen umzuwandeln.

SPEICHERVERWALTUNG:

Protection:

Jeder Prozess muss gegen ungewollte Einmischungen oder Störungen anderer Prozesse geschützt werden, egal ob diese Eingriffe beabsichtigt oder unbeabsichtigt sind

→ Folglich sollten fremde Prozesse nicht in der Lage sein, die gespeicherten Informationen eines Prozesses zu lesen oder zu modifizieren, wenn sie dafür keine Erlaubnis haben.

SPEICHERVERWALTUNG:

Sharing (Teilen/Aufteilen):

Die Speicherverwaltung muss verschiedenen Prozessen den Zugriff auf einen gemeinsam genutzten Speicherbereich ermöglichen, z. B. falls mehrere Prozesse das gleiche Programm ausführen, so ist es vorteilhaft, dass jeder Prozess auf dieselbe Version des Programms zugreift statt eine eigene Kopie zu nutzen

SPEICHERVERWALTUNG:

Logical Organization:

Stellt die Umsetzung von logischen Namen in Anwenderprogrammen zur physikalischen Organisation des Speichers (linearer 1-dimensionaler Adressraum) zur Verfügung.

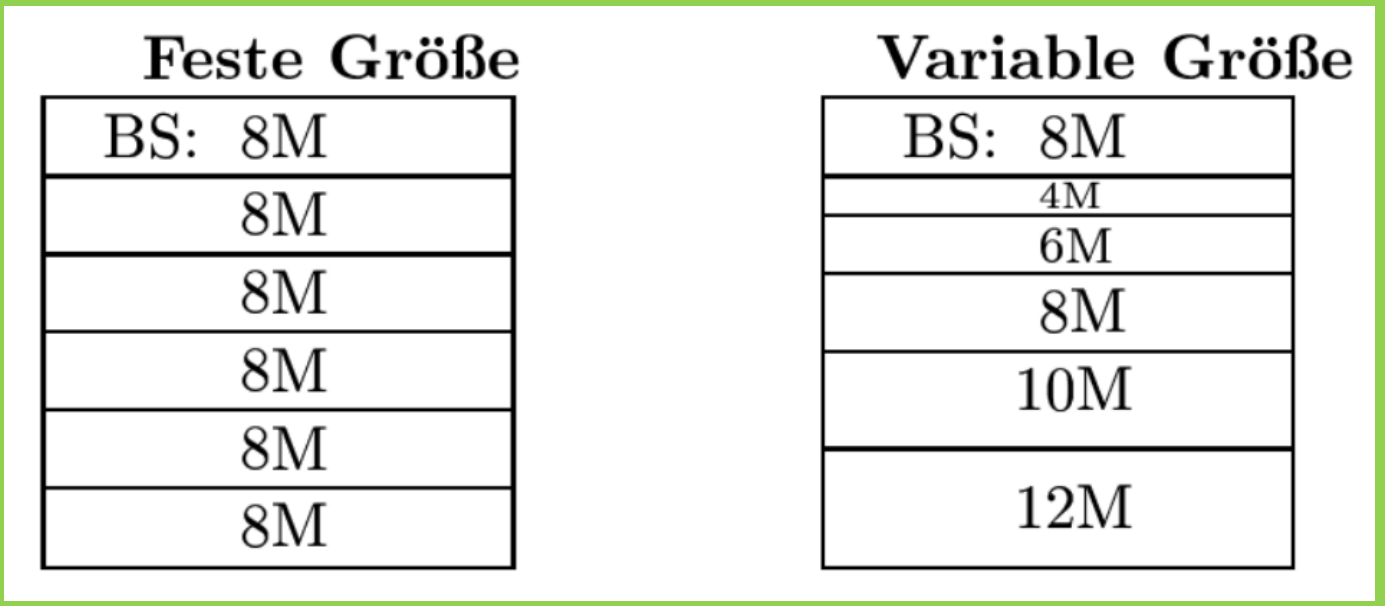
SPEICHERVERWALTUNG:

Physical Organization :

Dieses umfasst die transparente Einteilung in Hauptspeicher, d. h. schneller Zugriff mit hohen Kosten, und Hintergrundspeicher, d. h. langsamer aber billiger als der Hauptspeicher

SPEICHERPARTITIONIERUNG:

Statische Partitionierung:



SPEICHERPARTITIONIERUNG:

Statische Partitionierung:

Generell gilt:

- In die jeweilige Partition kann jeweils ein Prozess geladen werden, dessen Größe kleiner oder gleich der Partitionsgröße ist.
- Dieser Ansatz kann mit Swapping, (Folie später), kombiniert werden, wenn z. B. kein Prozess im „*running*“ oder „*ready*“ Status ist.
- Es kann eine Prozess-Warteschlange für alle Partitionen geben, sinnvoll für Partitionen fester Größe, ungünstig für Partitionierung mit variabler Größe, oder für jede Partitionsgröße eine.

SPEICHERPARTITIONIERUNG:

Statische Partitionierung:

Die Einreihung eines Prozesses in eine Warteschlange geschieht dabei über einen Belegungs-Algorithmus (**placement algorithm**)

- Eine Prozess-Warteschlange pro Partition: Best-Fit
- Eine Prozess-Warteschlange für alle Partitionen: First-Fit
- **Vorteile:**
  - einfache Implementierung
  - geringer Overhead zur Laufzeit

SPEICHERPARTITIONIERUNG:

Statische Partitionierung:

Nachteile:

- Wenn ein Programm zu groß ist, um in eine Partition hineinzupassen, dann ist es die Aufgabe des Programmierers, das Programm in Teilprogramme zu zerlegen (*Overlay*) oder sich einen anderen Hauptspeicher zuzulegen, d. h. größer oder anders partitioniert.
- Die Partitionierung ist extrem ineffizient, d. h. viele kleine Prozesse können die Partitionen belegen, so dass viel Platz verschwendet wird. ⟶ Dieser Sachverhalt wird als **interne Fragmentierung** bezeichnet.
- Es kann nur eine fixe Anzahl von Prozessen geben.

(heutige BS nutzen SP üblicherweise nicht mehr)

SPEICHERPARTITIONIERUNG:

Dynamische Partitionierung:

Die einzelnen Partitionen haben eine variable Länge und eine variable Anzahl. Soll ein Prozess gespeichert werden, so wird für ihn genau so viel Platz bereitgestellt, wie dieser braucht - und nicht mehr.

Wird von freien Speicher ausgegangen, so funktioniert dieser Algorithmus recht gut - der Speicher jedoch fragmentiert zunehmend mit der Zeit.

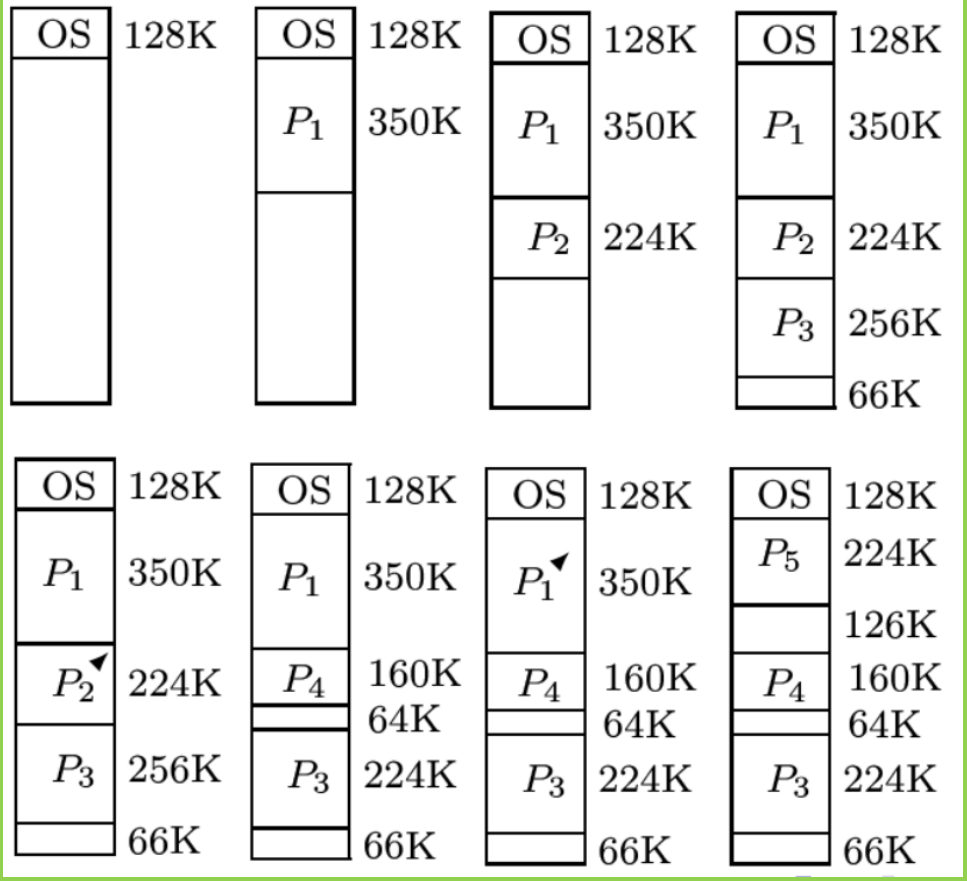
Dieses Phänomen heißt **externe Fragmentierung.**

Dagegen hilft die Komprimierung (Kompaktierung, **Compaction**): Von Zeit zu Zeit werden die Prozesse zu einem Block zusammengeschoben.

Dies setzt eine dynamische Relokation der Prozesse voraus.

SPEICHERPARTITIONIERUNG:

Dynamische Partitionierung:



SPEICHERPARTITIONIERUNG:

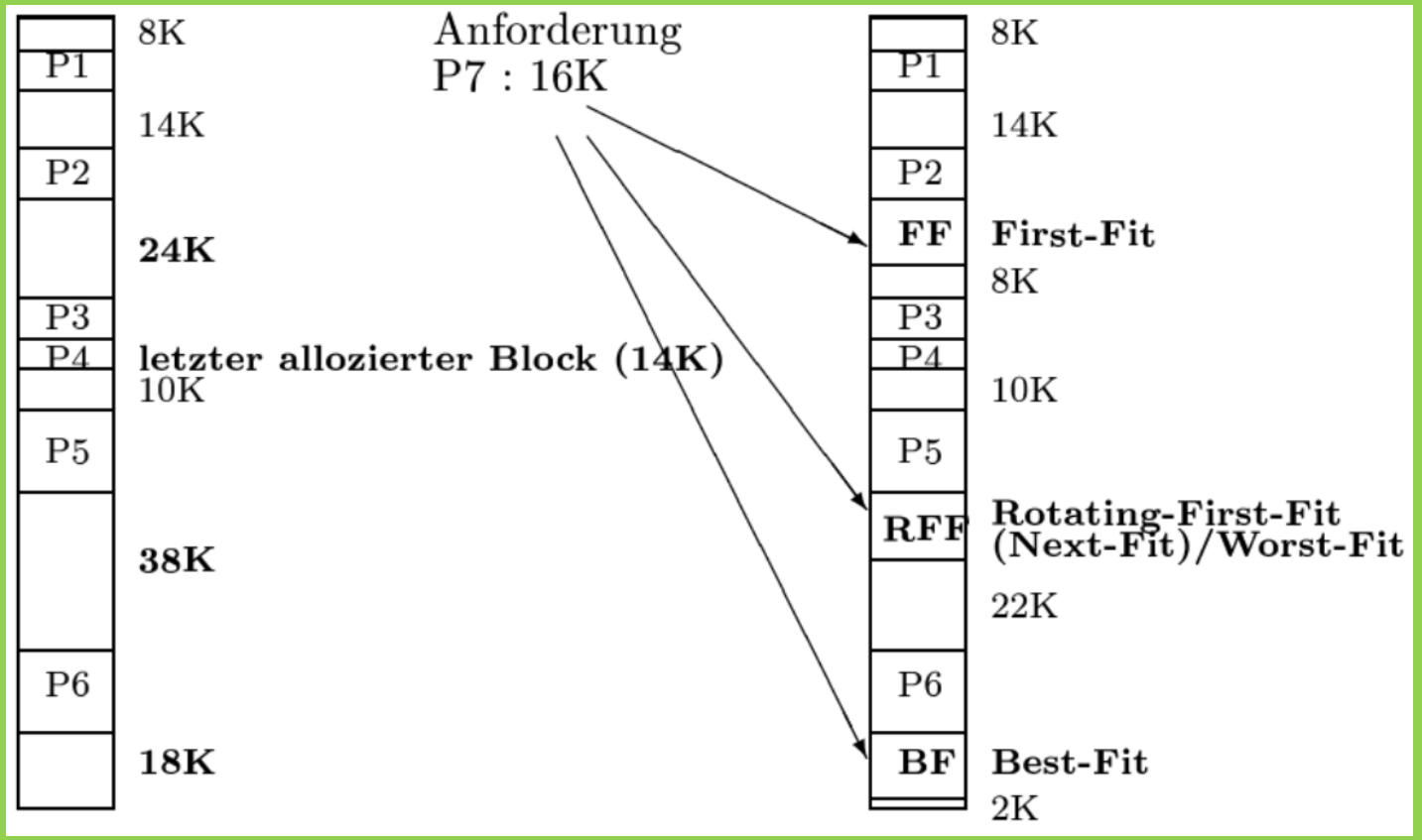
Dynamische Partitionierung (placement algorithm):

Um innerhalb einer Speicherstruktur, in der einzelne Blöcke abgespeichert sind, ein optimales Füllen der freien Speicherbereiche zu erhalten, gibt es verschiedene Belegungs-Algorithmen (**placement algorithm**):

- First-Fit: nimm von vorne gesehen die erste passende Lücke.
- Next-Fit: (**Rotating-First-Fit**) nimm ab der letzten Belegung die nächste passende Lücke.
- Best-Fit: suche die kleinste Lücke, in die der Prozess passt.
- Worst-Fit: suche die größte Lücke, in die der Prozess passt.

SPEICHERPARTITIONIERUNG:

Dynamische Partitionierung:



SPEICHERPARTITIONIERUNG:

Buddy-Systeme:

- Bislang: Feste Partitionierung hatte den Nachteil, dass die Anzahl aktiver Prozesse begrenzt war und der Speicherplatz ineffizient genutzt wurde.
- Eine dynamische Partitionierung ist dagegen schwer zu verwalten und lässt unnutzbare Freiräume, wenn nicht eine aufwändige Kompaktierung vorgenommen wird.
- Ein interessanter Kompromiss sind die sogenannten Buddy-Systeme.
- Dabei wird von Speicherblöcken der Größe 2^K ausgegangen, wobei $L \leq K \leq U$ mit 2^L = kleinster verfügbarer Block und 2^U = größter verfügbarer Block (i.d.R. gesamter verfügbarer Speicher)
- **Anfangszustand:** Der verfügbare Speicherplatz wird als einzelner Block der Größe 2^U betrachtet.

SPEICHERPARTITIONIERUNG:

Buddy-Systeme - Algorithmus:

- Ist $S > 2^U$, so kann die Anforderung nicht erfüllt werden.
- Kommt nun eine Speicheranforderung S, so wird die Bedingung $2^{(U-1)} < S \leq 2^U$ geprüft und in diesem Fall ein Block der Größe 2^U alloziert.
- Andernfalls wird der Speicherblock in zwei Buddies der Größe $2^{(U-1)}$ aufgesplittet.
- Wenn $2^{(U-2)} < S \leq 2^{(U-1)}$, dann Allokation der Anforderung in einem der beiden Buddies.
- Andernfalls Wiederholung, bis der kleinste Block einer 2er-Potenz erreicht ist mit $2^l < S$

SPEICHERPARTITIONIERUNG:

Buddy-Systeme - Beispiel:

1M					1MB Block
A=128K	128K	256K	512K		Anf. A 110K
A=128K	128K	B=256K	512K		Anf. B 230K
A=128K	C=64K	64K	B=256K	512K	Anf. C 60K
A=128K	C=64K	64K	B=256K	D=256K	256K	Anf. D 250K
A=128K	C=64K	64K	256K	D=256K	256K	Freigabe B
128K	C=64K	64K	256K	D=256K	256K	Freigabe A
E=128K	C=64K	64K	256K	D=256K	256K	Anf. E 70K
E=128K	128K	256K	D=256K	256K	Freigabe C
512K			D=256K	256K	Freigabe E
1M					Freigabe D

SPEICHERPARTITIONIERUNG:

Buddy-Systeme - Beispiel:

- Das Buddy-Verfahren ist also eine Mischform aus starrer und dynamischer Segmentierung.
- Die effiziente Implementierung ist trickreich und z. B. in [Knu73] angegeben.
- Bei Buddy-Systemen tritt sowohl externe als auch interne Fragmentierung auf

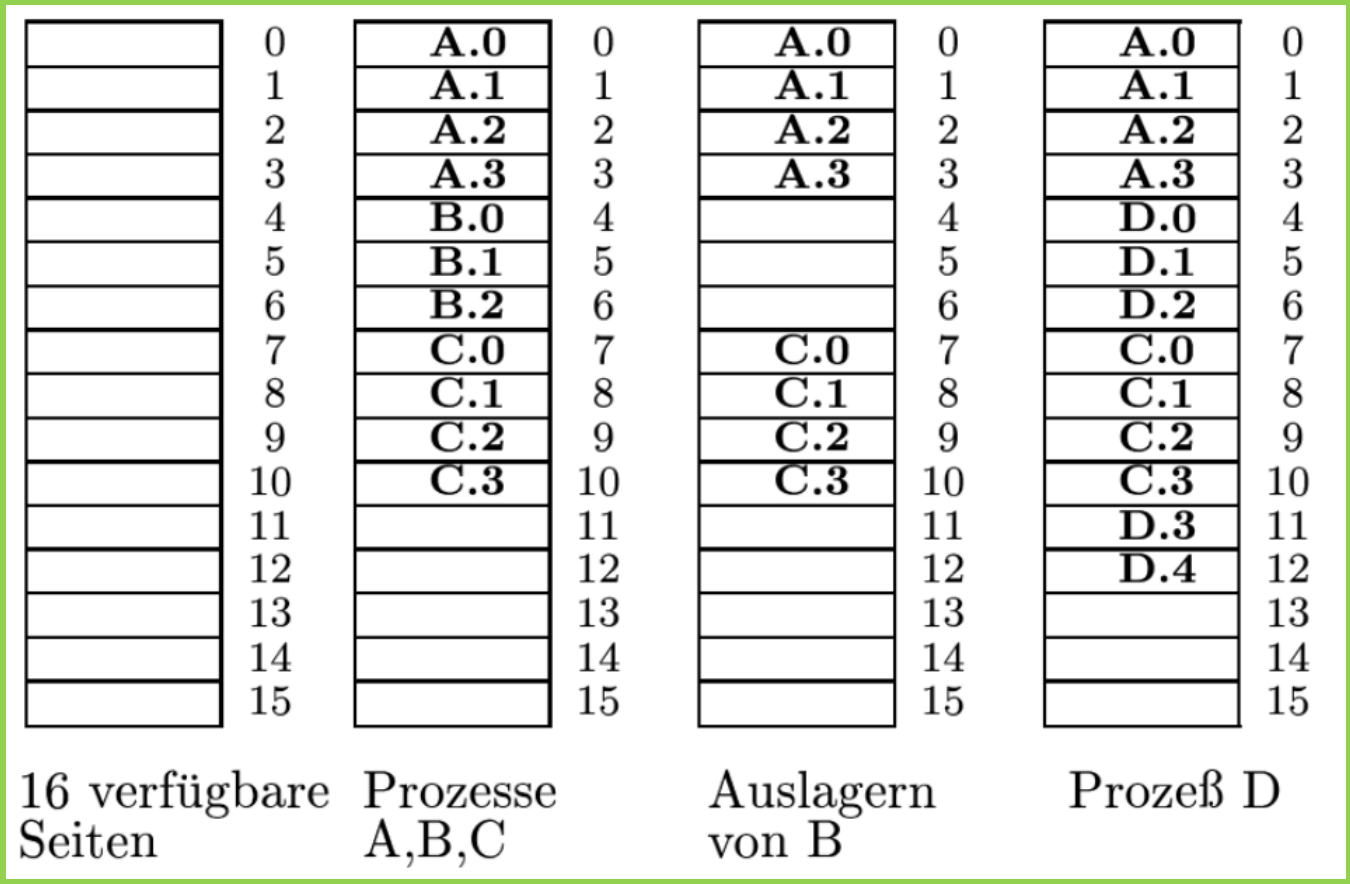
SPEICHERVERWALTUNG:

Paging:

- In diesem Abschnitt sollen zwei Verfahren zur Speicherverwaltung besprochen werden, die besonders im Zusammenhang mit virtuellem Speicher zum Einsatz kommen.
- Vorausgesetzt wird, dass der Hauptspeicher und der Prozess jeweils in gleichgroße Blöcke eingeteilt wird.
- Die Blöcke des Prozesses werden als **Pages** (Seiten) bezeichnet.
- Diese können den Blöcken des Hauptspeichers, den sogenannten **Frames** (Seitenrahmen oder Kacheln; page frames) zugeordnet werden.
- Dabei wird der Prozess nicht notwendigerweise in zusammenhängende Seiten geladen

SPEICHERVERWALTUNG:

Paging:



SPEICHERVERWALTUNG:

Paging:

- Betriebssystem verwaltet für jeden Prozess eine Seitentabelle (**PageTable**), die die benutzten Frames für jeden Prozess speichert.
- Innerhalb des Programms enthält jede logische Adresse eine Seitennummer und einen Offset (Abstand vom Seitenbeginn) innerhalb der Seite.
- Die Hardware des Prozessors (oder Software bei RISC-Prozessoren) übernimmt Umrechnung einer logischen in eine physische Adresse.

Beispiel: Für den Prozess D aus der letzten Abbildung sieht die Seitentabelle wie Folgt aus:

4	0
5	1
6	2
11	3
12	4

Abbildung: Beispiel für eine Seitentabelle

SPEICHERVERWALTUNG:

Paging:

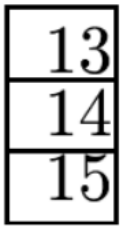
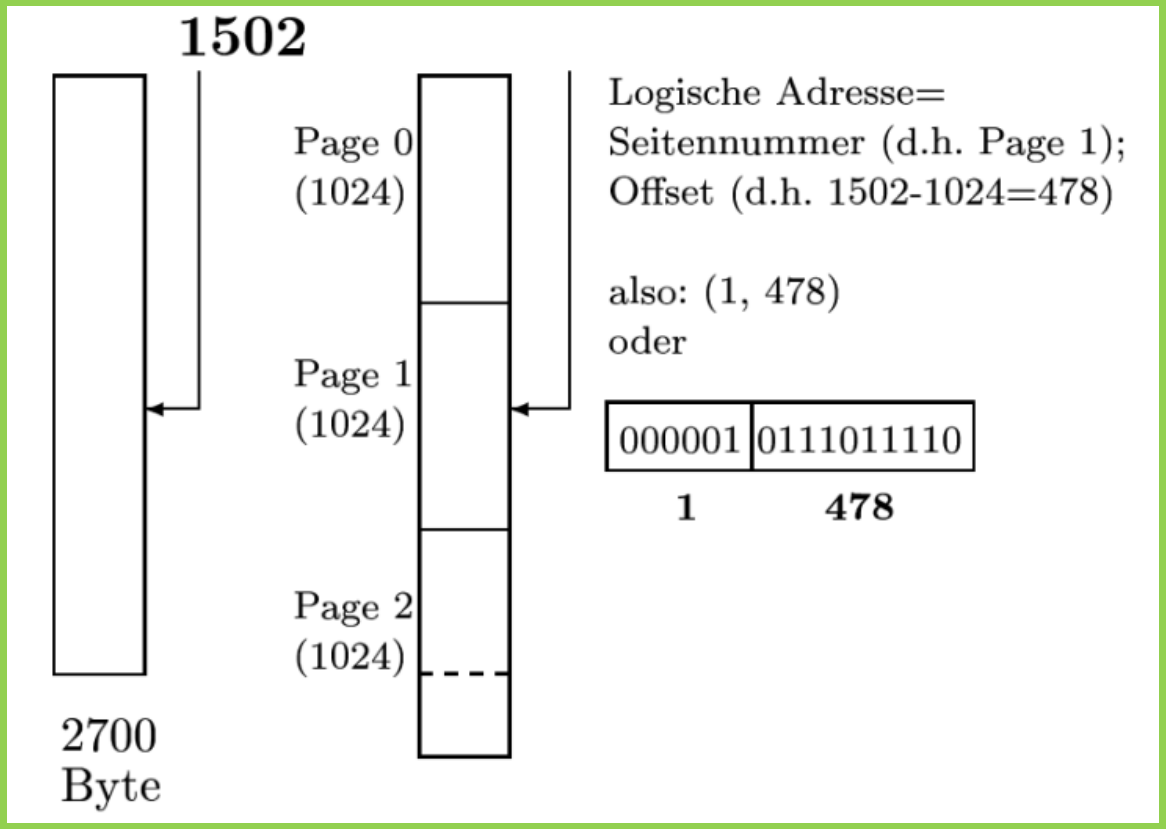


Abbildung: Beispiel für eine Tabelle freier Speicherseiten

- Nun soll eine Adresse im Prozess B mit folgender Annahme betrachtet werden: Der Prozess B bestehe aus 2700 Byte. Ein Frame soll aus Gründen der Adressierung in der Größe einer Zweierpotenz entsprechen, z. B. 1 KByte = 1024 Byte.
- Die relative, 16-bit breite, Adresse 0000010111011110 (dezimal: 1502) soll als Beispiel der Berechnung der logischen Adresse dienen. Mit dieser Adresse wird das 1502-te Byte innerhalb in der Beschreibung von Prozess B angesprochen

SPEICHERVERWALTUNG:

Paging – Berechnung der logischen Adresse:



SPEICHERVERWALTUNG:

Paging (Berechnung der logischen Adresse):

- Bei einer Größe der Frames von 1 KByte = 2^{10} Byte heißt dies, dass 10 Bit zur Adressierung des Offset und 6 Bit zur Adressierung von 2^6 = 64 Seiten (Frames) der Größe 1 KByte benötigt werden. In diesem Fall, ausgehend von 16 Seiten, sind nur 4 Bit notwendig, um die 2^4 = 16 Seiten zu adressieren. Die restlichen 2 Bit eines Halbworts der Länge 16 Bit bleiben ungenutzt.

- **Vorteile des Pagings:** Eine Verschwendung von Speicherplatz tritt nur im Sinne der internen Fragmentierung innerhalb der letzten Seite auf.

SPEICHERVERWALTUNG:

Paging (Berechnung der physischen Adresse):

- Zuerst logische Adresse berechnen (Seitennummer, Offset)
- Danach Seitenrahmennummer anhand Seitennummer ablesen
- Rahmennummer mit Rahmengröße multiplizieren, Offset addieren

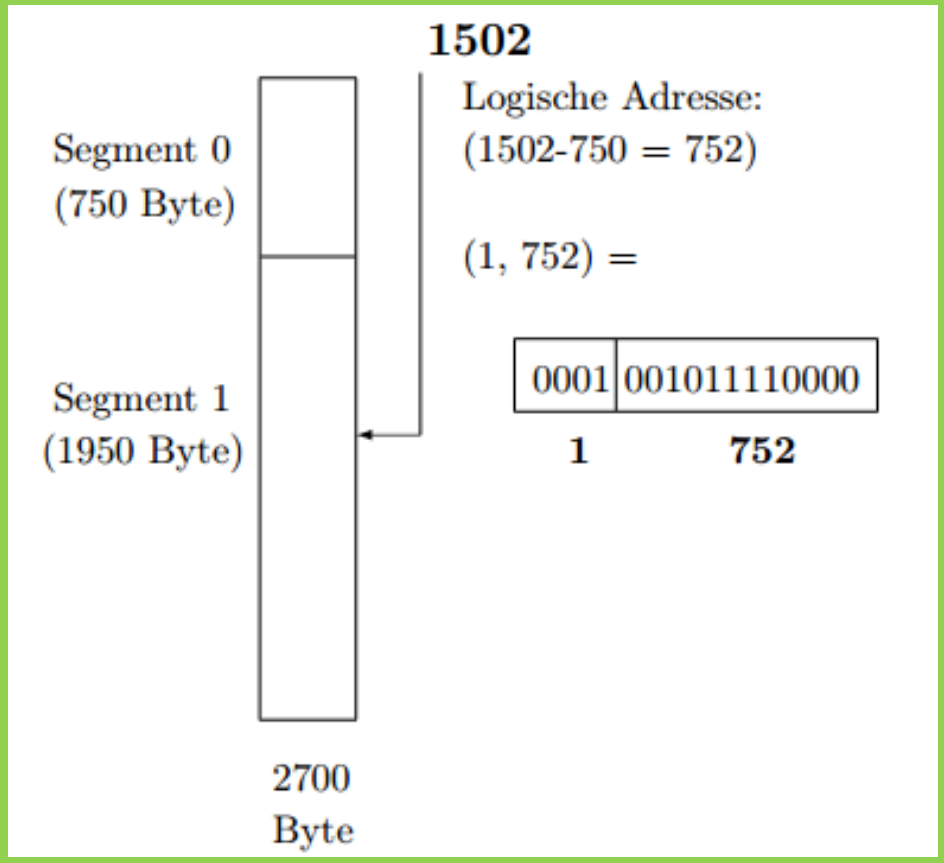
SPEICHERVERWALTUNG:

Segmentierung:

- Bei der Segmentierung wird jeder Prozess in mehrere Teile (Segmente)verschiedener Größe aufgesplittet und diese in nicht notwendigerweise zusammenhängende Partitionen geladen. Für die Segmentlänge existiert i. A. eine obere Schranke (z. B. die Größe des Hauptspeichers).
- Analog zum Paging werden auch bei der Segmentierung logische Adressen verwendet, die aus zwei Teilen bestehen: einer Segmentnummer und einem Offset. Segmentierung vermeidet interne Fragmentierung, dafür ist jedoch externe Fragmentierung möglich. Diese ist jedoch, aufgrund der Aufsplittung von Prozessen in einzelne Teile, geringer als bei der dynamischen Partitionierung.
- **Beispiel:** Als Beispiel dient wieder der Prozess B und die Speicherstelle (dezimal) 1502. Der Prozess ist jetzt in zwei Segmente eingeteilt. Das erste Segment hat eine Größe von 750 Byte und das zweite eine Größe von 950 Byte, entsprechend der folgenden Abbildung.

SPEICHERVERWALTUNG:

Segmentierung (Berechnung der logischen Adresse):



SPEICHERVERWALTUNG:

Segmentierung:

Die Adressberechnung ist bei der Segmentierung nicht so effektiv und elegant wie beim Paging. Insbesondere gibt es durch die verschieden großen Segmente keine einfache Abbildungsvorschrift da keine 2er Potenzen von Segmenten adressiert werden.

SPEICHERVERWALTUNG:

Virtueller Speicher:

Zwei Grundgedanken führten zum Konzept des Virtuellen Speichers:

1. Alle Speicheradressen innerhalb eines Prozesses sind logische Adressen, die zur Laufzeit auf dynamische Adressen abgebildet werden, deshalb kann ein Prozess beliebig ein- und ausgelagert werden und innerhalb der Abarbeitung auch unterschiedliche Adressen einnehmen.
2. Ein Prozess kann in eine gewissen Anzahl von Teilen zerlegt werden (Pages oder Segmente), die sich während der Ausführung nicht alle permanent im Speicher befinden müssen. Es brauchen nicht alle Teile eines Prozesses während dessen Ausführung stets im Hauptspeicher vorliegen.

Folglich muss sich nur eine Teilmenge des Prozesses ständig im Hauptspeicher befinden. Diese heißt residente Menge (**resident set**) desProzesses. Insbesondere ist darin der Teil des Prozesses enthalten, der zumStart notwendig ist.

SPEICHERVERWALTUNG:

Virtueller Speicher - Ablauf:

- Das Programm kann nun solange abgearbeitet werden, bis eine logische Adresse aufgerufen wird, die sich nicht im Speicher befindet.
- Dann generiert der Prozessor Interrupt, um einen Speicherzugriffsfehler (**memory access fault**) anzuzeigen.
- Das BS ändert den Prozesszustand in „blocked“ und übernimmt die Kontrolle. Für eine weitere Abarbeitung dieses Prozesses ist es seitens des Betriebssystems nun nötig, ein weiteres Stück des Prozesses in den Hauptspeicher zu laden.
- Das Betriebssystem macht einen Leseaufruf auf einen Hintergrundspeicher, meist eine Platte. Nach der Initiierung dieses Aufrufes kann das Betriebssystem einen anderen Prozess dem Prozessor zuweisen, der während der Leseoperation ausgeführt werden kann.
- Ist das Stück des Prozesses gelesen, so wird eine E/A-Unterbrechung ausgeführt, die Kontrolle dem Betriebssystem zurückgegeben und gegebenenfalls der ursprüngliche Prozess wieder in den Zustand „ready“ überführt.

SPEICHERVERWALTUNG:

Virtueller Speicher:

Das Verfahren hat, trotz allem Aufwand, zwei Vorteile:

1. Im Hauptspeicher können mehr Prozesse gespeichert werden. Damit kann bei vielen blockierten Prozessen sich immer noch eher einer im Zustand „ready“ befinden, als ohne Benutzung dieses Verfahrens.
2. Es können Prozesse abgearbeitet werden, die mehr Speicherplatz beanspruchen als der Hauptspeicher zur Verfügung stellt.

Da ein Prozess nur im Hauptspeicher ausgeführt werden kann, wird dieser auch als **realer Speicher** bezeichnet. Der Programmierer hat dagegen einen viel größeren Speicher - gegebenenfalls auf Platte - zur Verfügung. Dieser wird als **virtueller Speicher** bezeichnet.
Virtueller Speicher ist in der Regel mit Paging verbunden. Im folgenden erfolgt eine Beschränkung auf diesen Fall. Neben dem Paging kann virtueller Speicher auch mittels Segmentierung realisiert werden oder mittels einer Kombination von beidem

SPEICHERVERWALTUNG:

Virtueller Speicher:

Ausgangspunkt sind virtuelle Adressen im Adressraum A und reale Adressen im Speicherraum S, wobei $S << A$ gilt. Da ein Programm nur virtuelle Adressen enthält (Seitennummer und Offset), der Zugriff während der Ausführung aber auf reale Adressen erfolgt, müssen wir eine Adresstransformation $f: A \longrightarrow S$ durchführen.

SPEICHERVERWALTUNG:

Virtueller Speicher:

Adressraum A enthalte n Seiten

Speicherraum S enthalte $m \leq n$ Seitenrahmen

$$f(i) = \begin{cases} k & \text{Seite } i \text{ in Seitenrahmen } k \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Zu einer gültigen virtuelle Adresse a mit $0 \leq a \leq n \cdot z$ (z Bytes pro Seite)

$a = (i-1) \cdot z + d$ mit $0 \leq d < z$ ist die reale Adresse s dann

$s = \{f(i)\} - 1 \cdot z + d$ mit $0 \leq d < z$

Wenn eine referierte virtuelle Adresse nicht im Speicherraum S ist, so wird von einem Seitenfehler (memory access fault; hier: **page fault**)gesprochen, d. h. $f(i)$ ist undefiniert. Die Abbildungsfunktion f der Adresstransformation wird i. A. durch Seitentabellen, siehe folgende Abschnitte, realisiert

SPEICHERVERWALTUNG:

Virtueller Speicher:

Eine wichtige Entscheidung, die beim Betriebssystementwurf getroffen werden muss, ist die Größe der einzelnen Pages. Je kleiner - desto geringer ist die interne Fragmentierung, aber desto mehr Seiten müssen pro Prozess geladen werden und desto mehr Einträge hat die Seitentabelle. Der Verwaltungsaufwand und die anteilige Prozessorzeit steigen.

Problem:

Im eingeschwungenen Zustand (steady state - statistisches Gleichgewicht) wird praktisch der gesamte Hauptspeicher mit Teilen von Prozessen belegt sein. Um ein neues Stück eines Speichers laden zu können, muss irgendein Stück, ggf. auch eines anderen Prozesses, ausgelagert werden.

Dabei tritt das Problem des **Trashings** auf. Unter Umständen verschwendet der Prozessor mehr Zeit mit dem Ein- und Auslagern von Prozessstücken als mit der eigentlichen Abarbeitung

SPEICHERVERWALTUNG:

Virtueller Speicher:

Das Vermeiden dieses Problems basiert darauf, möglichst solche Prozessstücke auszulagern, die nicht mehr oder möglichst lange nicht mehr gebraucht werden. D. h. auf keinen Fall sollten solche Prozessstücke ausgelagert werden, die recht schnell wieder eingelagert werden müssen.

Diese Problematik war in den 70er Jahren ein grosses Forschungsthema.

SPEICHERVERWALTUNG:

Virtueller Speicher:

Wie wird Virtueller Speicher durch das BS unterstützt?

An dieser Stelle soll ausschließlich Paging betrachtet werden. Wird Virtueller Speicher durch Segmentierung oder beides unterstützt, so ist die Wirkungsweise ähnlich.

Zielsetzung: Das Speichermanagement soll eine hohe Performance aufweisen. Insbesondere soll die Anzahl der Seitenfehler minimal sein. Dabei gibt es verschiedene Vorschriften (Policies), die das Betriebssystem für den Virtuellen Speicher macht. Im Folgenden sollen die Organisation von Seitentabellen und einige Beispiele solcher Policies betrachtet werden.

SPEICHERVERWALTUNG:

Virtueller Speicher – Organisation von Seitentabellen:

Hierbei unterscheidet man ein- und mehrstufige Seitentabellen:

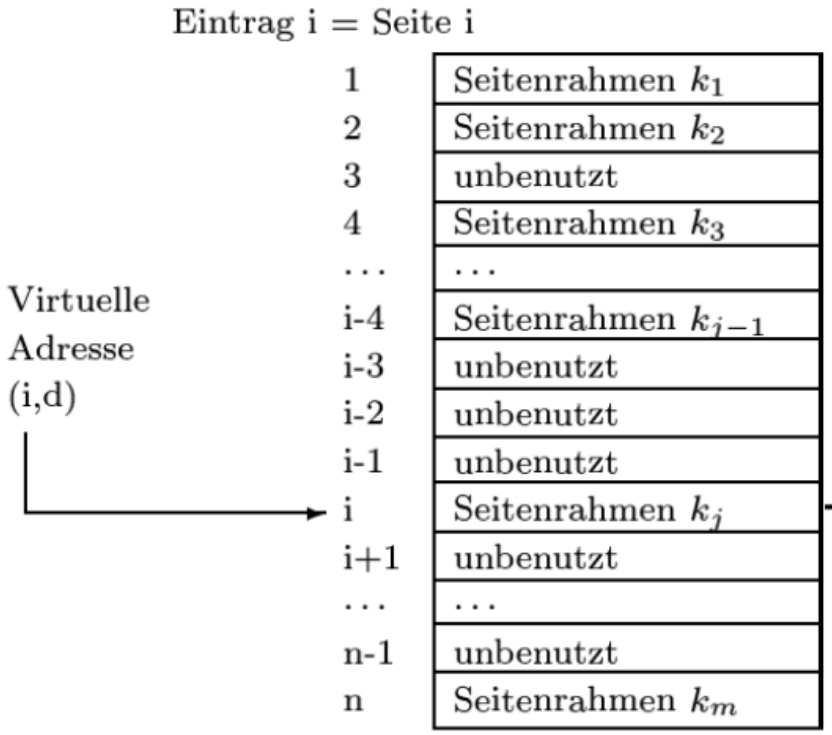
- Einstufige Seitentabellen:
  - nach Seiten-Nummern geordnet
  - nach Seitenrahmen-Nummern geordnet
- Zwei- oder mehrstufige Seitentabellen

Notation:

A: *n* Seiten
S: *m* Seitenrahmen mit $m << n$.

SPEICHERVERWALTUNG:

Virtueller Speicher – Organisation von Seitentabellen:



Vorteil:
Seiten-Nummer weist unmittelbar auf den Eintrag (einfach, schnell)

Nachteil:
Seitentabelle sehr groß und großer Teil leer (Platzvergeudung)

Abbildung: Einstufige Seitentabelle nach Seiten-Nummern geordnet

SPEICHERVERWALTUNG:

Virtueller Speicher – Organisation von Seitentabellen:

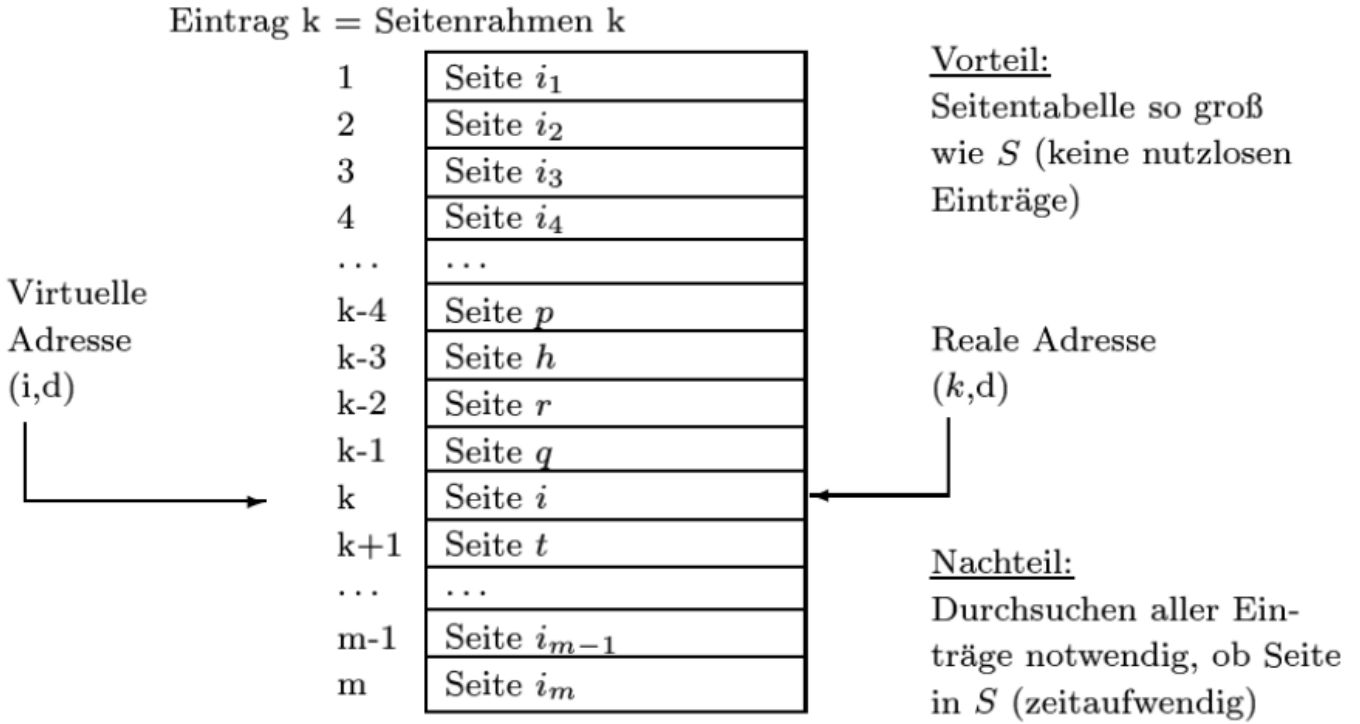


Abbildung: Einstufige Seitentabelle nach Seitenrahmen-Nummern geordnet

SPEICHERVERWALTUNG:

Replacement Policy: Longest Forward Distance (Optimalstrategie):

Als ein Vergleichsmaß der verscheidnen Strategien, bietet sich der nicht implementierbare Offline-Algorithmus LFD (*longest forward distance*) an.

Strategie

- Lagere die Seite aus, die erst am weitesten in der Zukunft nachgefragt wird.

Es kann gezeigt werden, siehe [Bel66], dass dies ein optimaler Algorithmus ist. Praktisch kann er aber nur eingesetzt werden, wenn man alle Seitenzugriffe kennt, also i.d.R. nur im Rückblick auf die Abarbeitung eines Prozesses. Deshalb bezeichnet man diesen Algorithmus auch als Offline-Algorithmus. Er dient hauptsächlich dazu, um die Güte der anderen Strategien beurteilen zu können.

SPEICHERVERWALTUNG:

Replacement Policy: Longest Forward Distance (Optimalstrategie):

Ausgehend von 3-Seitenrahmen und bei einer sequenziellen Anforderung der Seiten
$\{2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2\}$ ist folgende Belegung denkbar:

Anforderung	2	3	2	1	5	2	4	5	3	2	5	2	
Kachel 1		2	2	2	2	2	2*	4	4	4*	2	2	2
Kachel 2			3	3	3	3	3	3	3	3	3	3	3
Kachel 3					1*	5	5	5	5	5	5	5	5

Ein „*“ deutet einen Seitenfehler an. $\Rightarrow$ 3 Seitenfehler.

SPEICHERVERWALTUNG:

Replacement Policy: FIFO

Strategie

Die FIFO Strategie ordnet die Seiten nach dem Alter. Im Bedarfsfall wird die älteste Seite, d. h. die Seite, die sich bereits am längsten im Hauptspeicher befindet, ausgelagert

SPEICHERVERWALTUNG:

Replacement Policy: Last Recently used

Strategie:

Die LRU Strategie verdrängt Seiten nach dem Alter, d. h. die am längsten nicht mehr benutzte Seite wird aus dem Hauptspeicher ausgelagert.
Die zugrundeliegende Idee besteht in der Lokalität, d. h. die Seite, die gerade gebraucht wurde, wird wahrscheinlich wieder gebraucht.
Diese Strategie kommt der Optimalstrategie relativ nahe.

SPEICHERVERWALTUNG:

Replacement Policy: Clock Strategie

Strategie:
Im Folgenden soll die Verwaltung von Seiten in einer zyklischen Liste betrachtet werden, die die Form einer Uhr darstellt. Der Zeiger der Uhr zeigt dabei auf die älteste Seite.
Ein zusätzliches Bit pro Seite wird auch als *Use Bit* bezeichnet. Wird eine Seite das erste mal in ein Frame des Hauptspeichers geladen, so wird dieses Use Bit auf 1 gesetzt. Bei einer nachfolgenden Referenzierung erfolgt ebenfalls ein Setzen auf 1.
Ist es nötig, eine Seite zu ersetzen, so wird der Speicher durchsucht, um ein Frame mit einem Use Bit = 0 zu finden. Trifft er dabei auf ein Frame mit einem Use Bit = 1, so wird dieses auf den Wert 0 zurückgesetzt.
Haben alle Frames Use Bits = 0, so wird das erste Frame, auf das der Zeiger zeigt, für die Ersetzung gewählt. Haben andererseits alle Frames ein Use Bit = 1, so durchläuft der Zeiger einen kompletten Zyklus durch den Speicher, setzt alle Use Bits auf 0 und stoppt dann an der ursprünglichen Stelle. Das Frame wird genutzt, um eine Seite zu ersetzen.

Dieser Algorithmus ähnelt FIFO, allerdings mit der Ausnahme, dass Frames mit einem Use Bit = 1 durch den Algorithmus übersprungen werden.

VIELEN DANK