

# Betriebssysteme (BS)

## 09. Virtueller Speicher

<https://sys.cs.tu-dortmund.de/DE/Teaching/SS2021/BS/>

16.06.2021

**Peter Ulbrich**

peter.ulbrich@tu-dortmund.de

Basierend auf *Betriebssysteme* von Olaf Spinczyk, Universität Osnabrück

## Wiederholung

- Bei der Speicherverwaltung arbeitet das Betriebssystem sehr eng mit der Hardware zusammen.
  - **Segmentierung** und/oder **Seitenadressierung**
  - Durch die implizite Indirektion beim Speicherzugriff können Programme und Daten unter der Kontrolle des Betriebssystems im laufenden Betrieb beliebig verschoben werden.
- Zusätzlich sind diverse strategische Entscheidungen zu treffen.
  - **Platzierungsstrategie** (*First Fit, Best Fit, Buddy, ...*)
    - Unterscheiden sich bzgl. Verschnitt sowie Belegungs- und Freigabeaufwand.
    - Strategieauswahl hängt vom erwarteten Anwendungsprofil ab.
  - Bei Ein-/Auslagerung von Segmenten oder Seiten:
    - **Logische** bzw. **virtuelle Seiten** und **physische Seitenrahmen** (*Kacheln*)
    - Ladestrategie
    - Ersetzungsstrategie



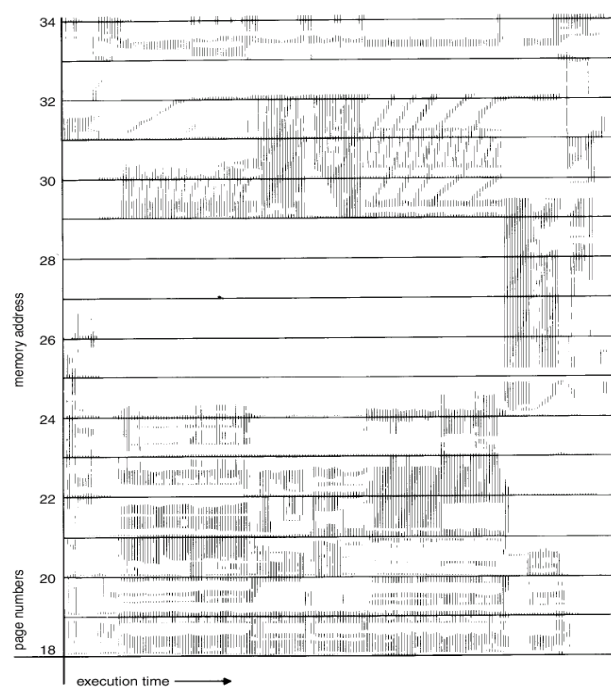
heute mehr dazu

## Inhalt

- **Motivation**
- Demand Paging
- Seitenersetzung
- Seitenzuordnung
- Ladestrategie
- Zusammenfassung

## Lokalität der Speicherzugriffe

- Einzelne Instruktionen benötigen nur wenige Speicherseiten.
- Auch über längere Zeiträume zeigt sich starke Lokalität.
  - Instruktionen werden z.B. eine nach der anderen ausgeführt.
- Die Lokalität kann ausgenutzt werden, wenn der Speicher nicht reicht.
  - z.B. „Overlay-Technik“



Quelle: Silberschatz, „Operating System Concepts“

## Die Idee des „Virtuellen Speichers“

- Entkoppelung des Speicherbedarfs vom verfügbaren Hauptspeicher
  - Prozesse benötigen nicht alle Speicherstellen gleich häufig:
    - bestimmte Befehle werden selten oder gar nicht benutzt (z.B. Fehlerbehandlungen)
    - bestimmte Datenstrukturen werden nicht voll belegt
  - Prozesse benötigen evtl. mehr Speicher als Hauptspeicher vorhanden
- Idee:
  - **Vortäuschen** eines größeren **Arbeitsspeichers**
  - **Einblenden** aktuell benötigter Speicherbereiche
  - **Auslagern** nicht benötigter Bereiche
  - Abfangen von Zugriffen auf nicht eingeblendete Bereiche, **einlagern** der benötigten Bereiche auf Anforderung

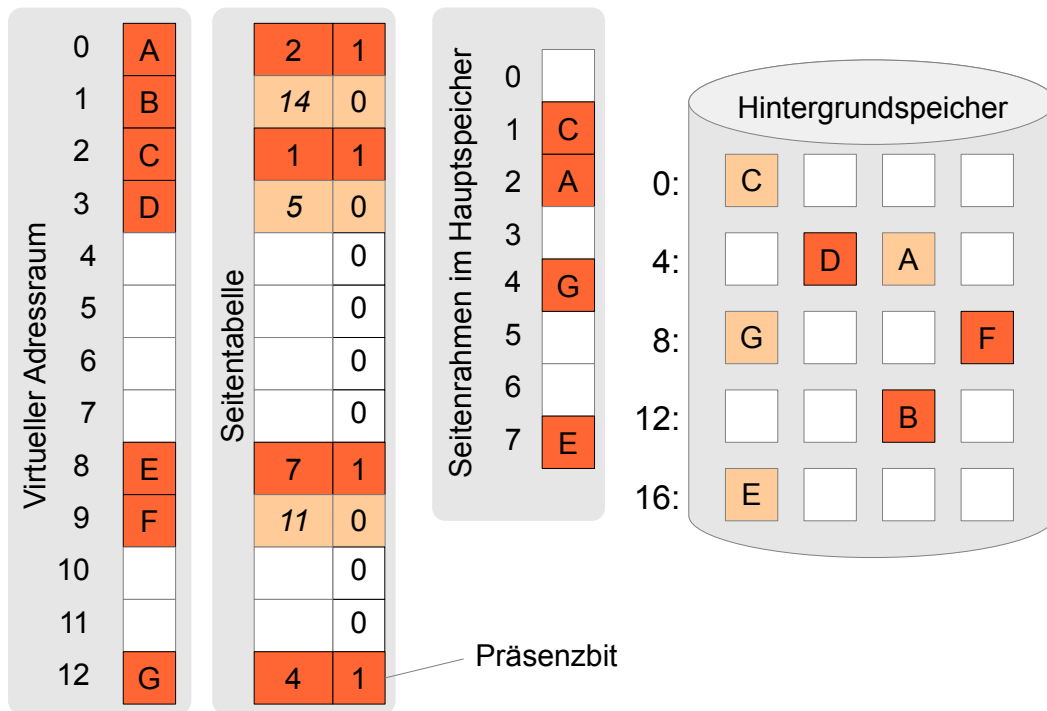
## Inhalt

- Motivation
- **Demand Paging**
- Seitenersetzung
- Seitenzuordnung
- Ladestrategie
- Zusammenfassung

*Tanenbaum*  
3: Speicherverwaltung  
*Silberschatz*  
9: Virtual Memory

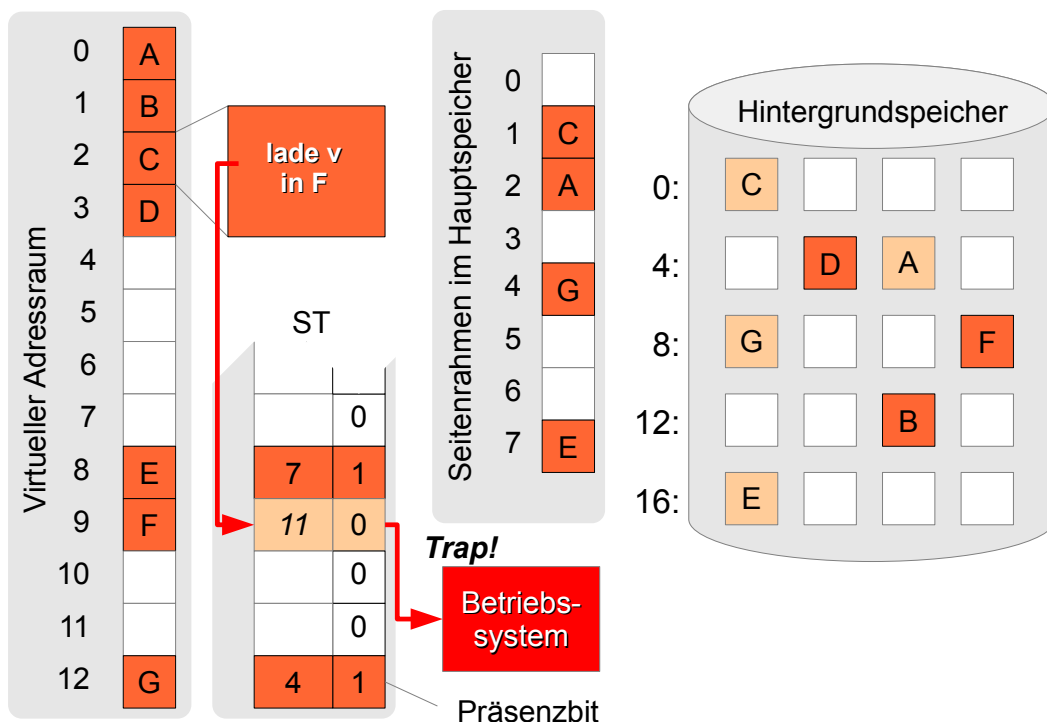
## Demand Paging (Seitenumlagerung)

- Bereitstellung von Seiten auf Anforderung



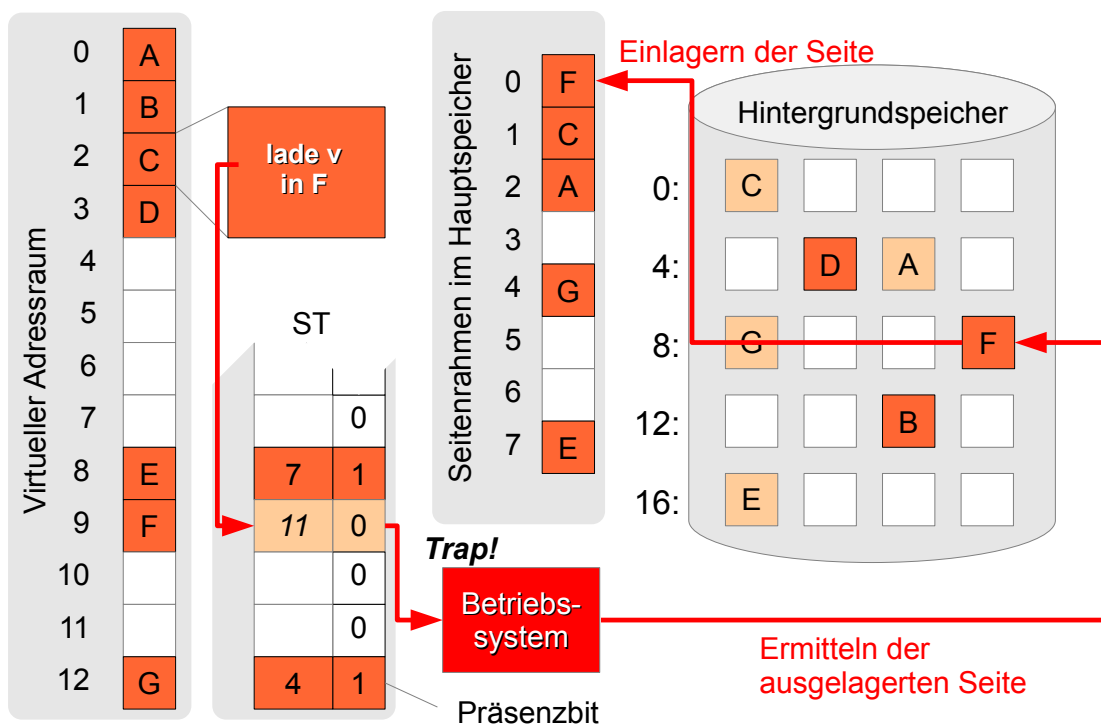
## Demand Paging (Seitenfehler)

- Reaktion auf Seitenfehler (*page fault*)



## Demand Paging (Seitenfehler)

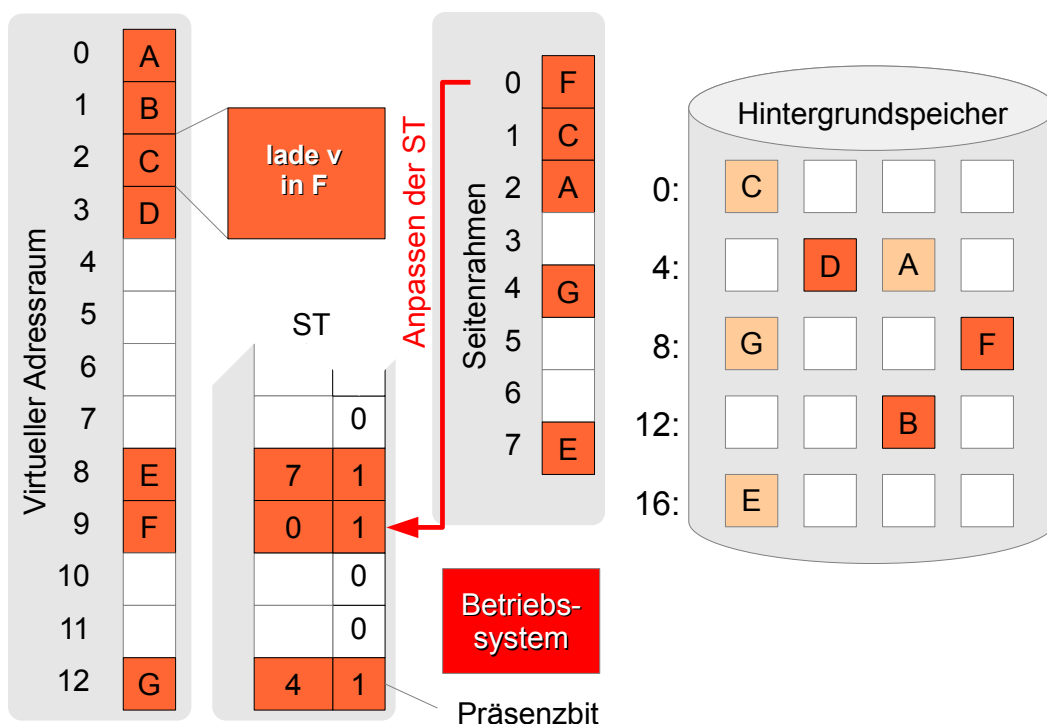
- Reaktion auf Seitenfehler (*page fault*)



9

## Demand Paging (Seitenfehler)

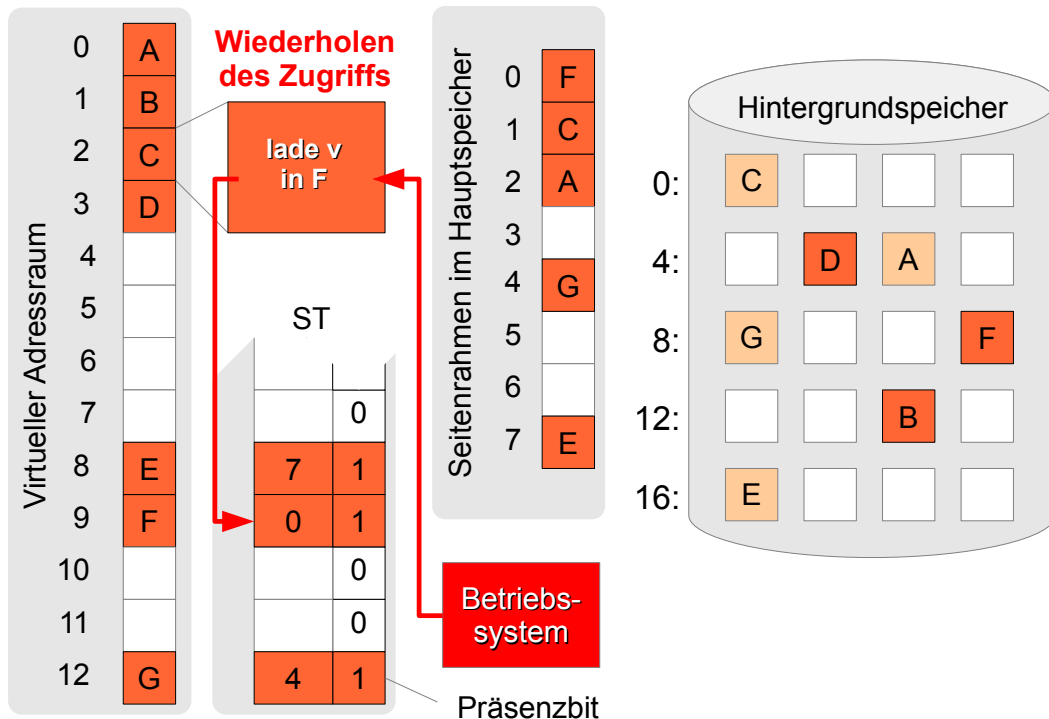
- Reaktion auf Seitenfehler (*page fault*)



10

## Demand Paging (Seitenfehler)

- Reaktion auf Seitenfehler (*page fault*)



11

## Diskussion: Kosten der Seitenumlagerung

- Performanz von Demand Paging

- **ohne Seitenfehler:**

- Effektive Zugriffszeit zwischen 10 und 200 Nanosekunden

- **mit Seitenfehler:**

- p sei Wahrscheinlichkeit für Seitenfehler
- Annahme:  
Zeit zum Einlagern einer Seite vom Hintergrundspeicher entspricht 25 Millisekunden (8 ms Latenz, 15 ms Positionierzeit, 1 ms Übertragungszeit)
- Annahme: normale Zugriffszeit 100 ns
- Effektive Zugriffszeit:  
 $(1 - p) \cdot 100 + p \cdot 25000000 = 100 + 24999900 \cdot p$

- **Seitenfehlerrate muss extrem niedrig sein**

- p nahe Null

## Diskussion: Weitere Eigenschaften

- **Prozesserzeugung**
  - **Copy-on-Write**
    - auch bei *Paging* MMU leicht zu realisieren
    - feinere Granularität als bei Segmentierung
  - Programmausführung und Laden erfolgen verschränkt:
    - Benötigte Seiten werden erst nach und nach geladen.
- **Sperren von Seiten**
  - notwendig bei Ein-/Ausgabeoperationen

## Diskussion: Segmentumlagerung

Prinzipiell möglich, hat aber **Nachteile** ...

- **Grobe Granularität**
    - z.B. Code-, Daten-, Stack-Segment
  - **Schwierigere Hauptspeicherverwaltung**
    - Alle freien Seitenrahmen sind gleich gut für ausgelagerte Seiten.  
Bei der Einlagerung von Segmenten ist die Speichersuche schwieriger.
  - **Schwierigere Hintergrundspeicherverwaltung**
    - Hintergrundspeicher ist wie Seitenrahmen in Blöcke strukturiert  
(2er-Potenzen)
- **In der Praxis hat sich Demand Paging durchgesetzt**

## Inhalt

- Motivation
- Demand Paging
- **Seitenersetzung**
- Seitenzuordnung
- Ladestrategie
- Zusammenfassung

## Seitenersetzung

- **Was tun, wenn kein freier Seitenrahmen vorhanden?**
  - Eine **Seite muss verdrängt** werden, um Platz für neue Seite zu schaffen!
  - Auswahl von Seiten, die nicht geändert wurden (*dirty bit* in der ST)
  - Verdrängung erfordert **Auslagerung, falls Seite geändert** wurde
- **Vorgang:**
  - Seitenfehler (*page fault*): Trap in das Betriebssystem
  - Auslagern einer Seite, falls kein freier Seitenrahmen verfügbar
  - Einlagern der benötigten Seite
  - Wiederholung des Zugriffs
- **Problem:**
  - Welche Seite soll ausgewählt werden (das „Opfer“)?



## Ersetzungsstrategien

- Betrachtung von Ersetzungsstrategien und deren **Wirkung auf Referenzfolgen**
- **Referenzfolge:**
  - **Speicherzugriffsverhalten** eines Prozesses → Folge von Seitennummern
  - Ermittlung von Referenzfolgen z.B. durch Aufzeichnung der zugegriffenen Adressen
    - Reduktion der aufgezeichneten Sequenz auf Seitennummern
    - Zusammenfassung von unmittelbar folgenden Zugriffen auf die gleiche Seite
  - Beispiel für eine **Referenzfolge: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**

## First-In, First-Out

- **Älteste Seite wird ersetzt**
- **Notwendige Zustände:**
  - Alter bzw. Einlagerungszeitpunkt für jeden Seitenrahmen
- **Ablauf der Ersetzungen (9 Einlagerungen)**

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Rahmen 1	1	1	1	4	4	4	5	5	5	5	5	5
	Rahmen 2		2	2	2	1	1	1	1	1	3	3	3
	Rahmen 3			3	3	3	2	2	2	2	2	4	4
Kontrollzustände (Alter pro Rahmen)	Rahmen 1	0	1	2	0	1	2	0	1	2	3	4	5
	Rahmen 2	>	0	1	2	0	1	2	3	4	0	1	2
	Rahmen 3	>	>	0	1	2	0	1	2	3	4	0	1

## First-In, First-Out

- Größerer Hauptspeicher mit 4 Seitenrahmen (10 Einlagerungen!)
- FIFO-Anomalie (Bélády's Anomalie, 1969)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Rahmen 1	1	1	1	1	1	1	5	5	5	5	4	4
	Rahmen 2		2	2	2	2	2	2	1	1	1	1	5
	Rahmen 3			3	3	3	3	3	3	2	2	2	2
	Rahmen 4				4	4	4	4	4	4	3	3	3
Kontrollzustände (Alter pro Rahmen)	Rahmen 1	0	1	2	3	4	5	0	1	2	3	0	1
	Rahmen 2	>	0	1	2	3	4	5	0	1	2	3	0
	Rahmen 3	>	>	0	1	2	3	4	5	0	1	2	3
	Rahmen 4	>	>	>	0	1	2	3	4	5	0	1	2

## Optimale Seitenersetzungsstrategie

- **Vorwärtsabstand**
  - Ersetze die Seite die am längsten nicht referenziert **wird**
- Strategie OPT (oder MIN) ist optimal (bei fester Seitenrahmenzahl):
- ➔ **Minimale Anzahl von Umlagerungen (hier 7)**
  - Wähle die Seite mit dem größten Vorwärtsabstand

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Rahmen 1	1	1	1	1	1	1	1	1	1	3	4	4
	Rahmen 2		2	2	2	2	2	2	2	2	2	2	2
	Rahmen 3			3	4	4	4	5	5	5	5	5	5
Kontrollzustände (Vorwärtsabstand)	Rahmen 1	4	3	2	1	3	2	1	>	>	>	>	>
	Rahmen 2	>	4	3	2	1	3	2	1	>	>	>	>
	Rahmen 3	>	>	7	7	6	5	5	4	3	2	1	>

## Optimale Seitenersetzungsstrategie

- Vergrößerung des Hauptspeichers (4 Seitenrahmen)

→ **6 Einlagerungen**

- keine Anomalie

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Rahmen 1	1	1	1	1	1	1	1	1	1	1	4	4
	Rahmen 2		2	2	2	2	2	2	2	2	2	2	2
	Rahmen 3			3	3	3	3	3	3	3	3	3	3
	Rahmen 4				4	4	4	5	5	5	5	5	5
Kontrollzustände (Vorwärtsabstand)	Rahmen 1	4	3	2	1	3	2	1	>	>	>	>	>
	Rahmen 2	>	4	3	2	1	3	2	1	>	>	>	>
	Rahmen 3	>	>	7	6	5	4	3	2	1	>	>	>
	Rahmen 4	>	>	>	7	6	5	5	4	3	2	1	>

## Diskussion: Optimale Seitenersetzungsstrategie

- Implementierung von OPT leider **praktisch unmöglich**

- Referenzfolge müsste vorher bekannt sein
- OPT ist nur zum Vergleich von Strategien brauchbar

→ Suche nach Strategien, die möglichst nahe an OPT kommen

- z.B. **Least Recently Used (LRU)**

## Least Recently Used (LRU)

- **Rückwärtsabstand**

- Zeitdauer, seit dem letzten Zugriff auf die Seite

- **LRU-Strategie (10 Einlagerungen)**

- Wähle den Seitenrahmen mit dem größten Rückwärtsabstand

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Rahmen 1	1	1	1	4	4	4	5	5	5	3	3	3
	Rahmen 2		2	2	2	1	1	1	1	1	1	4	4
	Rahmen 3			3	3	3	2	2	2	2	2	2	5
Kontrollzustände (Rückwärtsabstand)	Rahmen 1	0	1	2	0	1	2	0	1	2	0	1	2
	Rahmen 2	>	0	1	2	0	1	2	0	1	2	0	1
	Rahmen 3	>	>	0	1	2	0	1	2	0	1	2	0

## Least Recently Used (LRU)

- **Vergrößerung des Hauptspeichers (4 Seitenrahmen):  
8 Einlagerungen**

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Rahmen 1	1	1	1	1	1	1	1	1	1	1	1	5
	Rahmen 2		2	2	2	2	2	2	2	2	2	2	2
	Rahmen 3			3	3	3	3	5	5	5	5	4	4
	Rahmen 4				4	4	4	4	4	4	3	3	3
Kontrollzustände (Rückwärtsabstand)	Rahmen 1	0	1	2	3	0	1	2	0	1	2	3	0
	Rahmen 2	>	0	1	2	3	0	1	2	0	1	2	3
	Rahmen 3	>	>	0	1	2	3	0	1	2	3	0	1
	Rahmen 4	>	>	>	0	1	2	3	4	5	0	1	2

## Diskussion: Least Recently Used (LRU)

### ■ Keine Anomalie

- Allgemein gilt: Es gibt eine Klasse von Algorithmen (*Stack-Algorithmen*), bei denen keine Anomalie auftritt:
  - Bei Stack-Algorithmen ist bei  $k$  Rahmen zu jedem Zeitpunkt eine Teilmenge der Seiten eingelagert, die bei  $k+1$  Rahmen zum gleichen Zeitpunkt eingelagert wären!
  - **LRU**: Es sind immer die letzten  $k$  benutzten Seiten eingelagert.
  - **OPT**: Es sind die  $k$  bereits benutzten Seiten eingelagert, die als nächstes zugegriffen werden.

### ■ Problem:

- Implementierung von LRU nicht ohne **Hardwareunterstützung** möglich.
- Es muss jeder Speicherzugriff berücksichtigt werden.

## Least Recently Used – Hardwareunterstützung

### ■ Naive Idee: Hardwareunterstützung durch Zähler

- CPU besitzt einen Zähler, der bei jedem Speicherzugriff erhöht wird (inkrementiert wird)
- bei jedem Zugriff wird der aktuelle Zählerwert in den jeweiligen Seitendeskriptor geschrieben
- Auswahl der Seite mit dem kleinsten Zählerstand (Suche!)

### ■ Aufwändige Implementierung:

- viele zusätzliche Speicherzugriffe
- hoher Speicherplatzbedarf
- Minimum-Suche in der Seitenfehler-Behandlung

## Second Chance (Clock)

- **So wird's gemacht: Einsatz von Referenzbits**

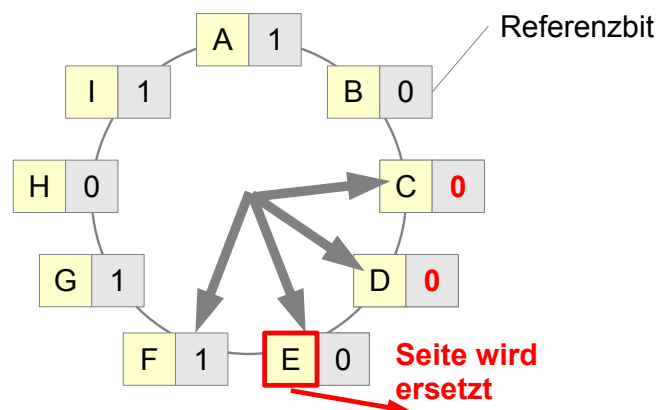
- Referenzbit im Seitendeskriptor wird automatisch durch Hardware gesetzt, wenn die Seite zugegriffen wird
  - einfacher zu implementieren
  - weniger zusätzliche Speicherzugriffe
  - moderne Prozessoren bzw. MMUs unterstützen Referenzbits (z.B. x86: access bit)

- **Ziel: Annäherung von LRU**

- bei einer frisch eingelagerten Seite wird das Referenzbit zunächst auf 1 gesetzt
- wird eine Opferseite gesucht, so werden die Seitenrahmen reihum inspiziert
  - ist das Referenzbit 1, so wird es auf 0 gesetzt (zweite Chance)
  - ist das Referenzbit 0, so wird die Seite ersetzt

## Second Chance (Clock)

- Implementierung mit **umlaufendem Zeiger (Clock)**



- an der Zeigerposition wird Referenzbit getestet
  - falls Referenzbit 1, wird Bit gelöscht
  - falls Referenzbit gleich 0, wurde ersetzbare Seite gefunden
  - Zeiger wird weitergestellt; falls keine Seite gefunden: Wiederholung
- falls alle Referenzbits auf 1 stehen, wird *Second Chance* zu FIFO

## Second Chance (Clock)

- Ablauf bei drei Seitenrahmen (9 Einlagerungen)

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Rahmen 1	1	1	1	4	4	4	5	5	5	5	5	5
	Rahmen 2		2	2	2	1	1	1	1	1	3	3	3
	Rahmen 3			3	3	3	2	2	2	2	2	4	4
Kontrollzustände (Referenzbits)	Rahmen 1	1	1	1	1	1	1	1	1	1	0	0	1
	Rahmen 2	0	1	1	0	1	1	0	1	1	1	1	1
	Rahmen 3	0	0	1	0	0	1	0	0	1	0	1	1
	Umlaufzeiger	2	3	1	2	3	1	2	2	2	3	1	1

## Second Chance (Clock)

- Vergrößerung des Hauptspeichers (4 Seitenrahmen):  
10 Einlagerungen

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Rahmen 1	1	1	1	1	1	1	5	5	5	5	4	4
	Rahmen 2		2	2	2	2	2	2	1	1	1	1	5
	Rahmen 3			3	3	3	3	3	3	2	2	2	2
	Rahmen 4				4	4	4	4	4	4	3	3	3
Kontrollzustände (Referenzbits)	Rahmen 1	1	1	1	1	1	1	1	1	1	1	1	1
	Rahmen 2	0	1	1	1	1	1	0	1	1	1	0	1
	Rahmen 3	0	0	1	1	1	1	0	0	1	1	0	0
	Rahmen 4	0	0	0	1	1	1	0	0	0	1	0	0
	Umlaufzeiger	2	3	4	1	1	1	2	3	4	1	2	3

## Second Chance (Clock)

- Bei **Second Chance** kann es auch zur **FIFO-Anomalie** kommen:
  - Wenn alle Referenzbits gleich 1, wird nach FIFO entschieden.
- Im Normalfall kommt man aber LRU nahe.
- Erweiterung:
  - **Modifikationsbit** kann zusätzlich berücksichtigt werden (Dirty Bit)
  - Drei Klassen: (0,0), (1,0) und (1,1) mit (Referenzbit, Modifikationsbit)
  - Suche nach der niedrigsten Klasse (Einsatz im MacOS)

## Diskussion: Freiseitenpuffer

### Freiseitenpuffer beschleunigt die Seitenfehlerbehandlung

- Statt eine Seite zu ersetzen, wird permanent eine Menge freier Seiten gehalten
  - **Auslagerung geschieht im Voraus**
  - Effizienter: Ersetzungszeit besteht im Wesentlichen nur aus Einlagerungszeit
- **Behalten der Seitenzuordnung auch nach der Auslagerung**
  - Wird die Seite doch noch benutzt, bevor sie durch eine andere ersetzt wird, kann sie mit hoher Effizienz wiederverwendet werden.
  - Seite wird aus Freiseitenpuffer ausgetragen und wieder dem entsprechenden Prozess zugeordnet.



## Inhalt

- Motivation
- Demand Paging
- Seitenersetzung
- **Seitenzuordnung**
- Ladestrategie
- Zusammenfassung

## Zuordnung von Seitenrahmen zu Prozessen

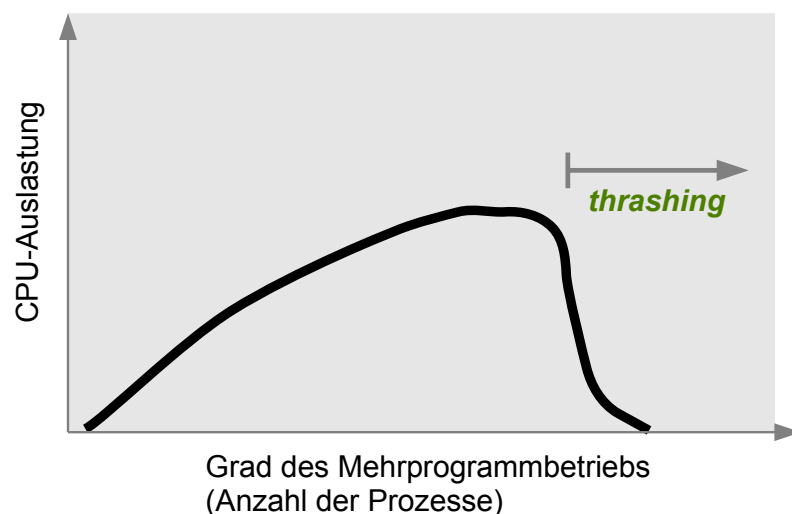
- **Problem: Aufteilung der Seitenrahmen auf die Prozesse**
  - Wie viele eingelagerte Seiten soll man einem Prozess zugestehen?
    - **Maximum:** begrenzt durch Anzahl der (physischen) Seitenrahmen
    - **Minimum:** abhängig von der Prozessorarchitektur
      - Mindestens die Anzahl von Seiten nötig, die theoretisch bei einem Maschinenbefehl benötigt werden  
(z.B. zwei Seiten für den Befehl, vier Seiten für die adressierten Daten)
- **Gleiche Zuordnung**
  - Anzahl der Prozesse bestimmt die Menge, die ein Prozess bekommt
- **Größenabhängige Zuordnung**
  - Größe des Programms fließt in die zugeteilte Menge der Seitenrahmen ein

## Zuordnung von Seitenrahmen zu Prozessen

- **Globale und lokale Anforderung von Seiten**
  
- **Lokal:** Prozess ersetzt nur immer seine eigenen Seiten
  - Seitenfehler-Verhalten liegt nur in der Verantwortung des Prozesses
  - Keine Interferenz zwischen Prozessen
  - Auslagerung unter Umständen unnötig (global ungenutzte Seiten)
  
- **Global:** Prozess ersetzt auch Seiten anderer Prozesse
  - Ungenutzt Seitenrahmen anderen Prozesse können verwendet werden
  - Interferenz zwischen Prozessen (Seitenfehler-Verhalten)

## Seitenflattern (*Thrashing*)

- **Ausgelagerte Seite wird gleich wieder angesprochen:**
  - Prozess verbringt mehr Zeit mit dem Warten auf das Beheben von Seitenfehlern als mit der eigentlichen Ausführung.



## Seitenflattern (*Thrashing*)

### ■ Ursachen:

- Prozess ist nahe am Seitenminimum
- Zu viele Prozesse gleichzeitig im System
- Schlechte Ersetzungsstrategie

→ **Lokale Seitenanforderung** behebt *Thrashing* zwischen Prozessen

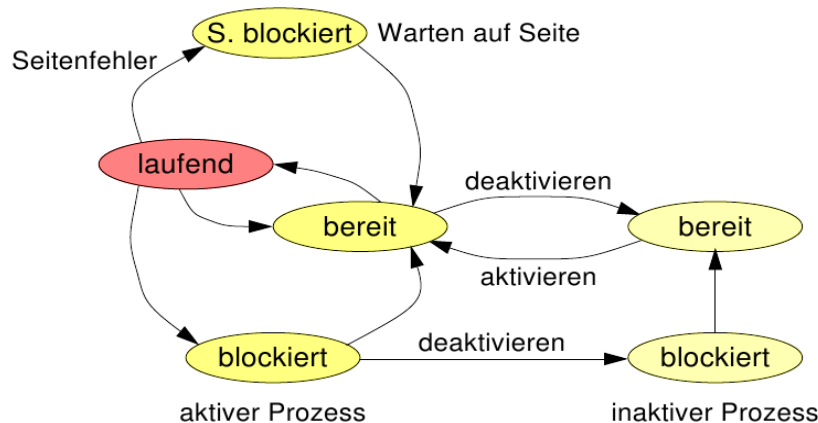
→ Zuteilung einer genügend großen Zahl von Rahmen behebt Prozess-lokales *Thrashing*

- **Begrenzung der Prozessanzahl**

## Lösung 1: Auslagerung von Prozessen

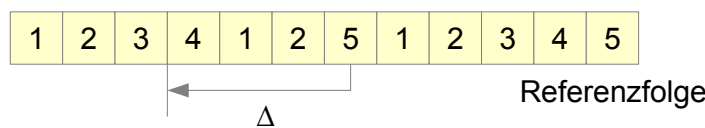
### ■ inaktiver Prozess benötigt keine Seitenrahmen

- Seitenrahmen teilen sich auf weniger Prozesse auf
- Verbindung mit der Ablaufplanung (*Scheduling*) nötig
  - Verhindern von Aushungerung
  - Erzielen kurzer Reaktionszeiten



## Lösung 2: Arbeitsmengenmodell

- Seitenmenge, die ein Prozess wirklich braucht (**Working Set**)
  - Kann nur angenähert werden, da üblicherweise nicht vorhersehbar
- Annäherung durch Betrachten der letzten  $\Delta$  Seiten, die angesprochen wurden
  - geeignete Wahl von  $\Delta$ 
    - **zu groß**: Überlappung von lokalen Zugriffsmustern
    - **zu klein**: Arbeitsmenge enthält nicht alle nötigen Seiten



- **Hinweis**:  $\Delta >$  Arbeitsmenge, da Seiten in der Regel mehrfach hintereinander angesprochen werden

## Arbeitsmengenmodell

- **Beispiel**: Arbeitsmengen bei verschiedenen  $\Delta$

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
$\Delta=3$	Seite 1	X	X	X		X	X	X	X	X	X		
	Seite 2		X	X	X		X	X	X	X	X	X	
	Seite 3			X	X	X					X	X	X
	Seite 4				X	X	X					X	X
	Seite 5							X	X	X			X
$\Delta=4$	Seite 1	X	X	X	X	X	X	X	X	X	X	X	
	Seite 2		X	X	X	X	X	X	X	X	X	X	X
	Seite 3			X	X	X	X				X	X	X
	Seite 4				X	X	X	X				X	X
	Seite 5							X	X	X	X		X

## Diskussion: Arbeitsmengenmodell

- **Annäherung der Zugriffe durch die Zeit**
  - Bestimmtes Zeitintervall ist ungefähr proportional zu Anzahl von Speicherzugriffen
- **Virtuelle Zeit des Prozesses muss gemessen werden**
  - Nur die Zeit relevant, in der der Prozess im Zustand RUNNING ist
  - Verwalten virtueller Uhren pro Prozess

## Arbeitsmengenbestimmung mit Zeitgeber

- **Annäherung der Arbeitsmenge mit**
  - Referenzbit
  - Altersangabe pro Seite (Zeitintervall ohne Benutzung)
  - Timer-Interrupt (durch Zeitgeber)
- **Algorithmus:**
  - durch regelmäßigen Interrupt wird mittels Referenzbit die Altersangabe fortgeschrieben:
    - ist Referenzbit gesetzt (Seite wurde benutzt), wird das Alter auf Null gesetzt;
    - ansonsten wird Altersangabe erhöht.
    - Es werden nur die Seiten des gerade laufenden Prozesses „gealtert“.
  - Seiten mit Alter  $> \Delta$  sind nicht mehr in der Arbeitsmenge des jeweiligen Prozesses.

## Arbeitsmengenbestimmung mit Zeitgeber

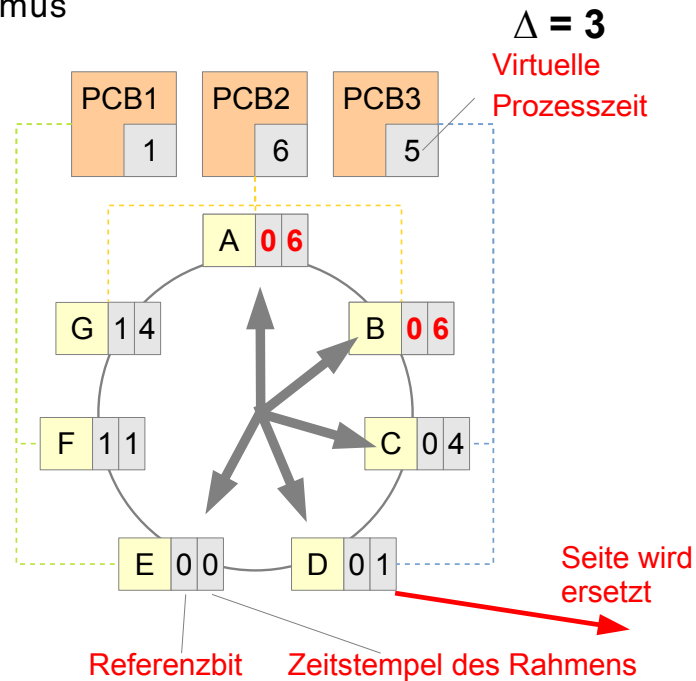
- **ungenau**
  - System ist aber nicht empfindlich auf diese Ungenauigkeit
  - Verringerung der Zeitintervalle: höherer Aufwand, genauere Messung
- **ineffizient**
  - große Menge von Seiten zu betrachten

## Arbeitsmengenbestimmung mit WSClock

- Algorithmus WSClock (working set clock)
  - Arbeitet wie Clock
  - Seite wird nur dann ersetzt, wenn sie **nicht zur Arbeitsmenge ihres Prozesses gehört** oder der Prozess deaktiviert ist.
- Bei Zurücksetzen des Referenzbits wird die virtuelle Zeit des jeweiligen Prozesses eingetragen, die z.B. im PCB gehalten und fortgeschrieben wird.
- Bestimmung der Arbeitsmenge erfolgt durch **Differenzbildung** von virtueller Zeit des Prozesses und Zeitstempel in dem Seitenrahmen.

## Arbeitsmengenbestimmung mit WSClock

### ■ WSClock-Algorithmus



## Diskussion: Arbeitsmengenprobleme

- **Speicherplatzbedarf** für Zeitstempel
- Zuordnung zu einem Prozess nicht immer möglich
  - gemeinsam genutzte Seiten in modernen Betriebssystemen eher die Regel als die Ausnahme
    - Shared Libraries
    - Gemeinsame Seiten im Datensegment (Shared Memory)
- **Lösung 3: Thrashing kann durch direkte Steuerung der Seitenfehlerrate leichter vermieden werden**
  - Messung pro Prozess
    - **Rate < Schwellwert:** Menge der Seitenrahmen verkleinern
    - **Rate > Schwellwert:** Menge der Seitenrahmen vergrößern

## Inhalt

- Motivation
- Demand Paging
- Seitenersetzung
- Seitenzuordnung
- **Ladestrategie**
- Zusammenfassung

## Ladestrategie

- **Auf Anforderung laden**
  - Damit ist man auf der sicheren Seite
- **Im Voraus laden**
  - **Schwierig:** Ausgelagerte Seiten werden eigentlich nicht gebraucht.
  - Oftmals löst eine Maschineninstruktion mehrere Seitenfehler aus.
    - Durch Interpretation des Befehls beim ersten Seitenfehler können die benötigten anderen Seiten im Voraus eingelagert werden.
    - Weitere Seitenfehler werden verhindert.
  - Komplettes Working Set bei Prozesseinlagerung im Voraus laden
  - Sequentielle Zugriffsmuster erkennen und Folgeseiten vorab laden



## Inhalt

- Motivation
- Demand Paging
- Seitenersetzung
- Seitenzuordnung
- Ladestrategie
- **Zusammenfassung**

## Zusammenfassung

- Virtueller Speicher ermöglicht die Nutzung großer logischer Adressräume trotz Speicherbeschränkung.
- Komfort hat aber seinen Preis:
  - Aufwand in der Hardware
  - Komplexe Algorithmen im Betriebssystem
  - „Erstaunliche“ Effekte (wie „Thrashing“)
  - Zeitverhalten nicht vorhersagbar
- Einfache (Spezialzweck-)Systeme, die diesen „Luxus“ nicht unbedingt benötigen, sollten besser darauf verzichten.