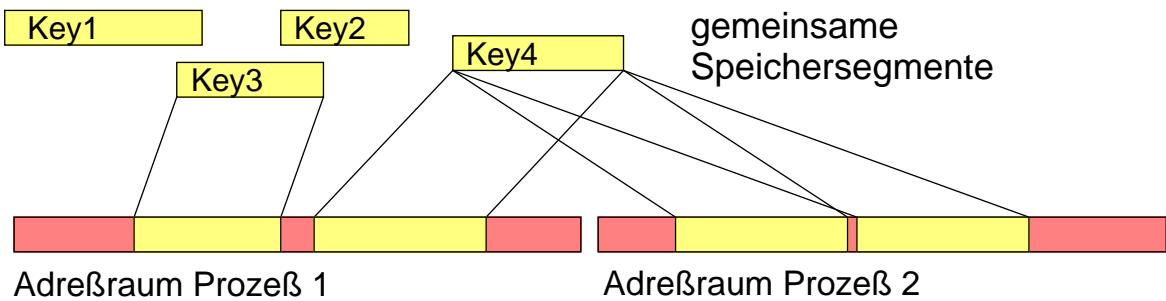


E.5 Gemeinsamer Speicher (3)

■ Prinzip der `shm*` Operationen



```
id1= shmget( Key3, ...);  
shmat( id1, NULL, ...);  
id2= shmget( Key4, ...);  
shmat( id2, NULL, ...);
```

```
id1= shmget( Key4, ...);  
shmat( id1, NULL, ...);  
shmat( id1, NULL, ...);
```

E.5 Gemeinsamer Speicher (4)

■ Verwendung des Keys

- ◆ Alle Prozesse, die auf ein Speichersegment zugreifen wollen, müssen den Key kennen
- ◆ Keys sind eindeutig innerhalb eines (Betriebs-)Systems
- ◆ Ist ein Key bereits vergeben, kann kein Segment mit gleichem Key erzeugt werden
- ◆ Ist ein Key bekannt, kann auf das Segment zugegriffen werden
 - gesetzte Zugriffsberechtigungen werden allerdings beachtet
- ◆ Es können Segmente ohne Key erzeugt werden (private Segmente)

■ Keys werden benutzt für:

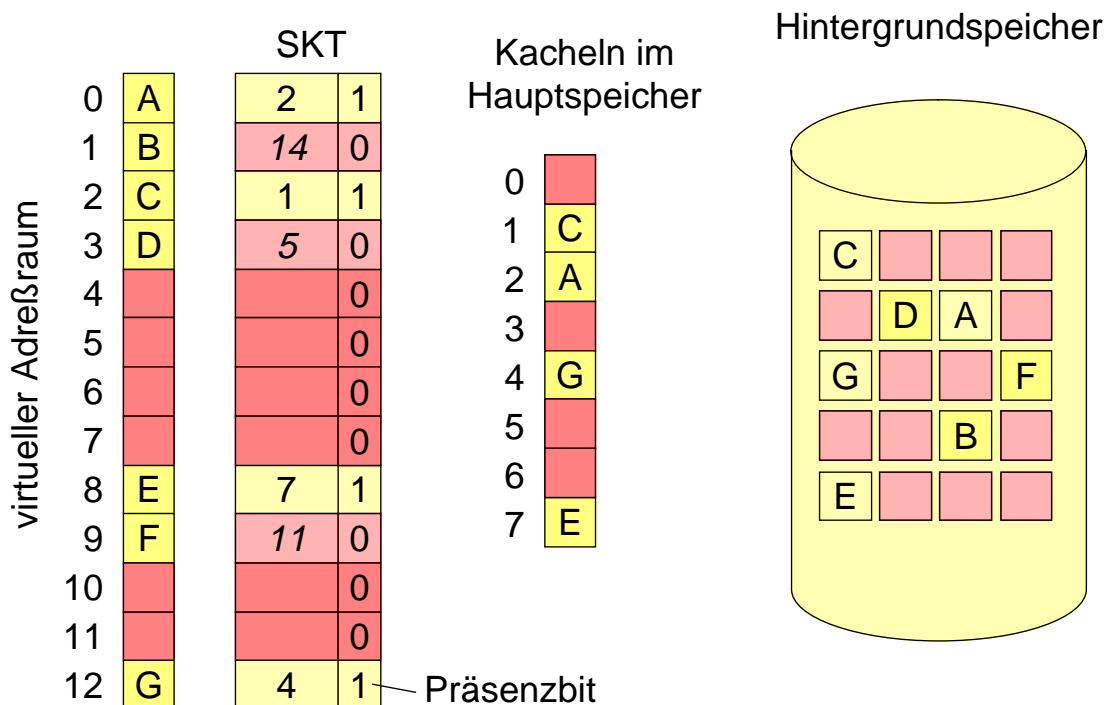
- ◆ Queues
- ◆ Semaphore
- ◆ Shared memory segments

E.6 Virtueller Speicher

- 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 großen Hauptspeichers
 - ◆ Einblenden benötigter Speicherbereiche
 - ◆ Abfangen von Zugriffen auf nicht eingeblendete Bereiche
 - ◆ Bereitstellen der benötigten Bereiche

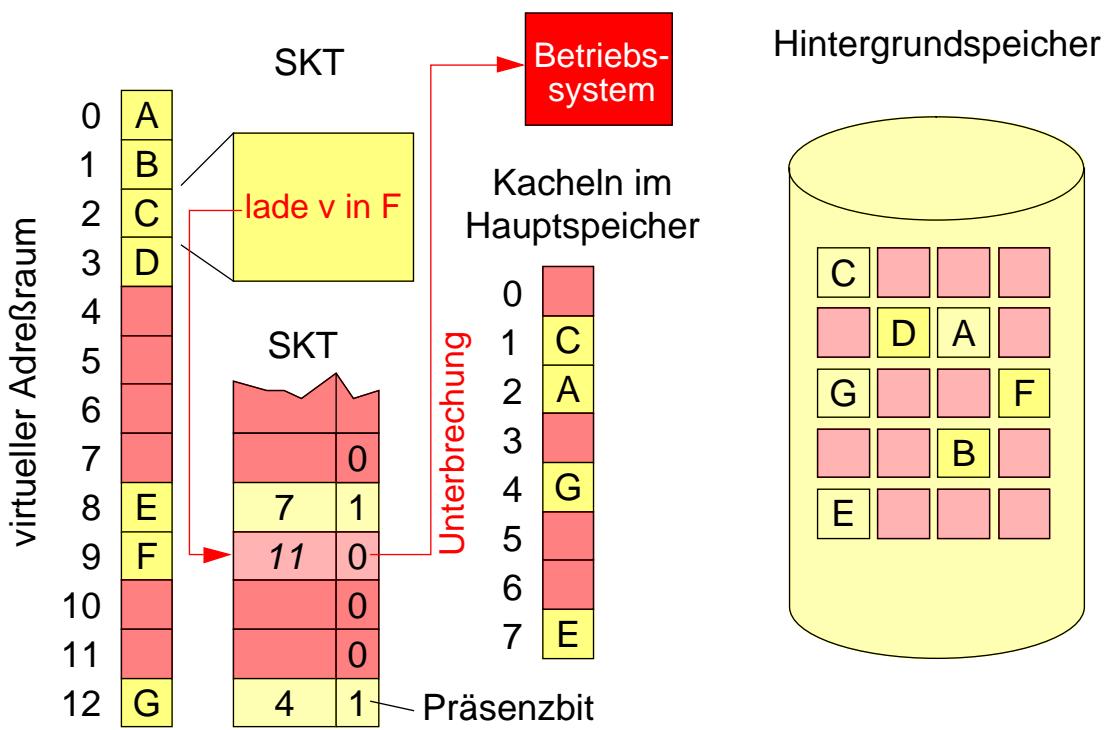
1 Demand Paging

- Bereitstellen von Seiten auf Anforderung



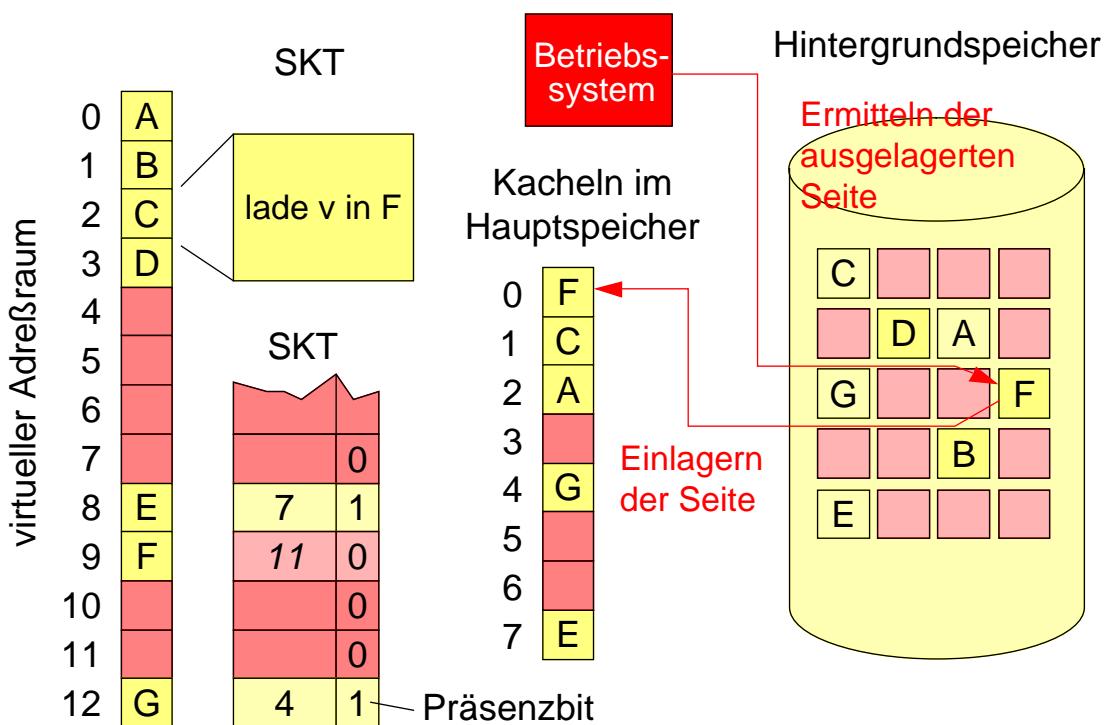
1 Demand Paging (2)

■ Reaktion auf Seitenfehler



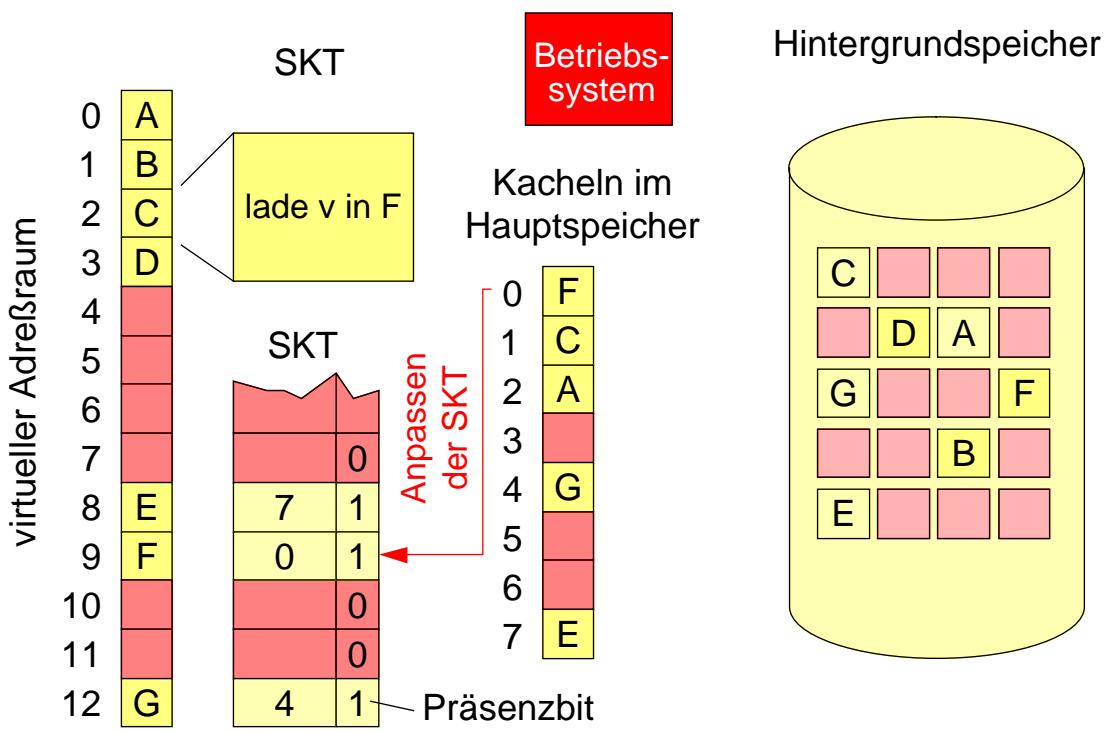
1 Demand Paging (3)

■ Reaktion auf Seitenfehler



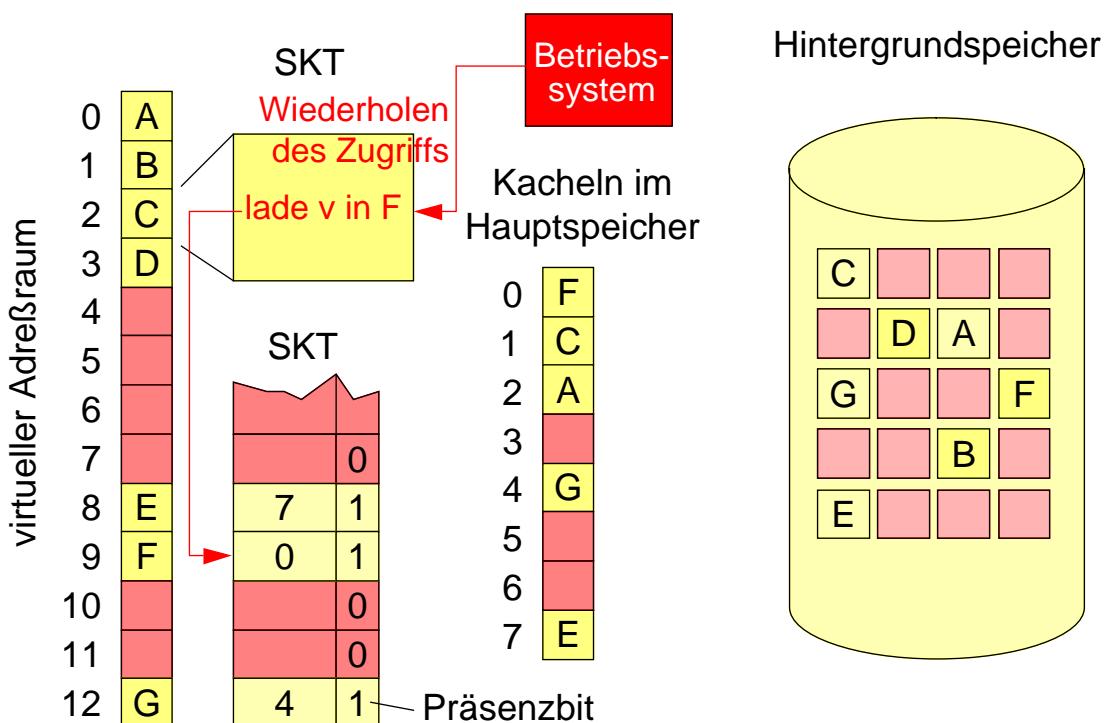
1 Demand Paging (4)

■ Reaktion auf Seitenfehler



1 Demand Paging (5)

■ Reaktion auf Seitenfehler



1 Demand Paging (6)

▲ Performanz von Demand paging

◆ Keine Seitenfehler

- effektive Zugriffszeit zw. 10 und 200 Nanosekunden

◆ Mit Seitenfehler

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

▲ Seitenfehler müssen so niedrig wie möglich gehalten werden

■ Abwandlung: *Demand zero* für nicht initialisierte Daten

2 Seitenersetzung

■ Was tun, wenn keine freie Kachel vorhanden?

- ◆ Eine Seite muß verdrängt werden, um Platz für neue Seite zu schaffen!
- ◆ Auswahl von Seiten, die nicht geändert wurden (*Dirty bit* in der SKT)
- ◆ Verdrängung erfordert Auslagerung, falls Seite geändert wurde

■ Vorgang:

- ◆ Seitenfehler (*Page fault*): Unterbrechung
- ◆ Auslagern einer Seite, falls keine freie Kachel verfügbar
- ◆ Einlagern der benötigten Seite
- ◆ Wiederholung des Zugriffs

▲ Problem

- ◆ Welche Seite soll ausgewählt werden?

E.7 Ersetzungsstrategien

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

1 First-In, First-Out

- Älteste Seite wird ersetzt
- Notwendige Zustände:
 - ◆ Alter bzw. Einlagerungszeitpunkt für jede Kachel
- Ablauf der Ersetzungen (9 Einlagerungen)

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

1 First-In, First-Out

- Größerer Hauptspeicher mit 4 Kacheln (10 Einlagerungen)

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

- FIFO Anomalie (Belady's Anomalie, 1969)

2 Optimale Ersetzungsstrategie

- Vorwärtsabstand

◆ Zeitdauer bis zum nächsten Zugriff auf die entsprechende Seite

- Strategie B_0 (OPT oder MIN) ist optimal (bei fester Kachelmenge): minimale Anzahl von Einlagerungen/Ersetzungen (hier 7)

◆ „Ersetze immer die Seite mit dem größten Vorwärtsabstand!“

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1	1	1	1	3	4	4
	Kachel 2		2	2	2	2	2	2	2	2	2	2	2
	Kachel 3			3	4	4	4	5	5	5	5	5	5
	Kachel 4				1	3	2	1	>	>	>	>	>
Kontrollzustände (Vorwärts- abstand)	Kachel 1	4	3	2	1	3	2	1	>	>	>	>	>
	Kachel 2	>	4	3	2	1	3	2	1	>	>	>	>
	Kachel 3	>	>	7	7	6	5	5	4	3	2	1	>

2 Optimale Ersetzungsstrategie (2)

- Vergrößerung des Hauptspeichers (4 Kacheln): 6 Einlagerungen

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1	1	1	1	1	4	4
	Kachel 2		2	2	2	2	2	2	2	2	2	2	2
	Kachel 3			3	3	3	3	3	3	3	3	3	3
	Kachel 4				4	4	4	5	5	5	5	5	5
Kontrollzustände (Vorwärts- abstand)	Kachel 1	4	3	2	1	3	2	1	>	>	>	>	>
	Kachel 2	>	4	3	2	1	3	2	1	>	>	>	>
	Kachel 3	>	>	7	6	5	4	3	2	1	>	>	>
	Kachel 4	>	>	>	7	6	5	5	4	3	2	1	>

- ◆ keine Anomalie

2 Optimale Ersetzungsstrategie (3)

- Implementierung von B_0 nahezu unmöglich
 - ◆ Referenzfolge müßte vorher bekannt sein
 - ◆ B_0 meist nur zum Vergleich von Strategien brauchbar
- Suche nach Strategien, die möglichst nahe an B_0 kommen
 - ◆ z.B. *Least recently used* (LRU)

3 Least Recently Used (LRU)

- Rückwärtsabstand
 - ◆ Zeitdauer, seit dem letzten Zugriff auf die Seite

- LRU Strategie (10 Einlagerungen)
 - ◆ „Ersetze die Seite mit dem größten Rückwärtsabstand !“

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	4	4	4	5	5	5	3	3	3
	Kachel 2		2	2	2	1	1	1	1	1	1	4	4
	Kachel 3			3	3	3	2	2	2	2	2	2	5
Kontrollzustände (Rückwärts- abstand)	Kachel 1	0	1	2	0	1	2	0	1	2	0	1	2
	Kachel 2	>	0	1	2	0	1	2	0	1	2	0	1
	Kachel 3	>	>	0	1	2	0	1	2	0	1	2	0

3 Least Recently Used (2)

- Vergrößerung des Hauptspeichers (4 Kacheln): 8 Einlagerungen

Referenzfolge		1	2	3	4	1	2	5	1	2	3	4	5
Hauptspeicher	Kachel 1	1	1	1	1	1	1	1	1	1	1	1	5
	Kachel 2		2	2	2	2	2	2	2	2	2	2	2
	Kachel 3			3	3	3	3	5	5	5	5	4	4
	Kachel 4				4	4	4	4	4	4	3	3	3
Kontrollzustände (Rückwärts- abstand)	Kachel 1	0	1	2	3	0	1	2	0	1	2	3	0
	Kachel 2	>	0	1	2	3	0	1	2	0	1	2	3
	Kachel 3	>	>	0	1	2	3	0	1	2	3	0	1
	Kachel 4	>	>	>	0	1	2	3	4	5	0	1	2

3 Least Recently Used (3)

■ Keine Anomalie

- ◆ Allgemein gilt: Es gibt eine Klasse von Algorithmen (Stack-Algorithmen), bei denen keine Anomalie auftritt:
 - Bei Stack-Algorithmen ist bei n Kacheln zu jedem Zeitpunkt eine Untermenge der Seiten eingelagert, die bei $n+1$ Kacheln zum gleichen Zeitpunkt eingelagert wären!
 - LRU: Es sind immer die letzten n benutzten Seiten eingelagert
 - B_0 : Es sind die n bereits benutzten Seiten eingelagert, die als nächstes zugegriffen werden

▲ Problem

- ◆ Implementierung von LRU nicht ohne Hardwareunterstützung möglich
- ◆ Es muß jeder Speicherzugriff berücksichtigt werden

3 Least Recently Used (4)

■ 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

▲ Aufwendige Implementierung

- ◆ viele zusätzliche Speicherzugriffe

4 Second Chance (Clock)

■ Einsatz von Referenzbits

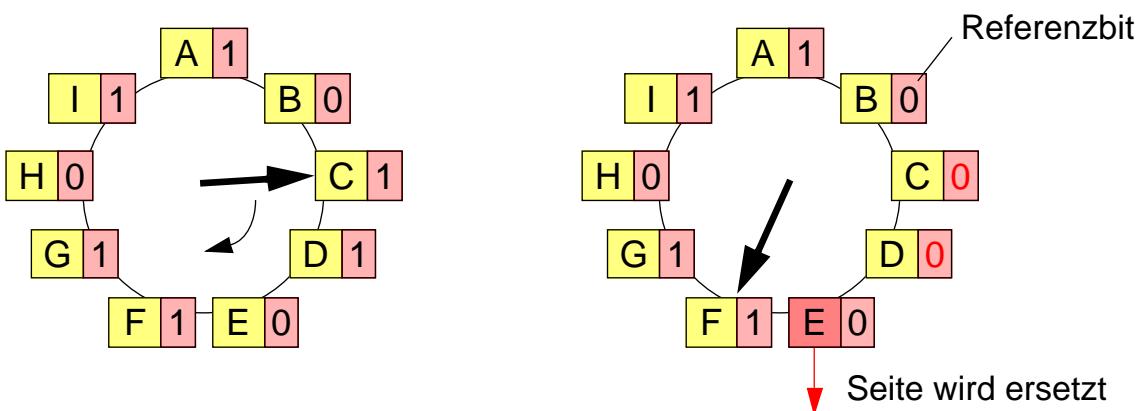
- ◆ Referenzbit im Seitendeskriptor wird automatisch durch Hardware gesetzt, wenn die Seite zugegriffen wird
 - einfacher zu implementieren
 - weniger zusätzliche Speicherzugriffe
 - moderne Prozessoren bzw. MMUs unterstützen Referenzbits (z.B. Pentium: *Access bit*)

■ Ziel: Annäherung von LRU

- ◆ das Referenzbit wird zunächst auf 0 gesetzt
- ◆ wird eine Opferseite gesucht, so werden die Kacheln reihum inspiziert
- ◆ ist das Referenzbit 1, so wird es auf 0 gesetzt (zweite Chance)
- ◆ ist das Referenzbit 0, so wird die Seite ersetzt

4 Second Chance (2)

■ Implementierung mit umlaufendem Zeiger (*Clock*)



- ◆ an der Zeigerposition wird Referenzbit getestet
 - falls Referenzbit eins, wird Bit gelöscht und Zeiger weitergestellt
 - falls Referenzbit gleich Null, wurde ersetzbare Seite gefunden
- ◆ falls alle Referenzbits auf 1 stehen, wird Second chance zu FIFO

4 Second Chance (3)

■ Ablauf bei drei Kacheln (9 Einlagerungen)

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

4 Second Chance (4)

■ Vergrößerung des Hauptspeichers (4 Kacheln): 10 Einlagerungen

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