

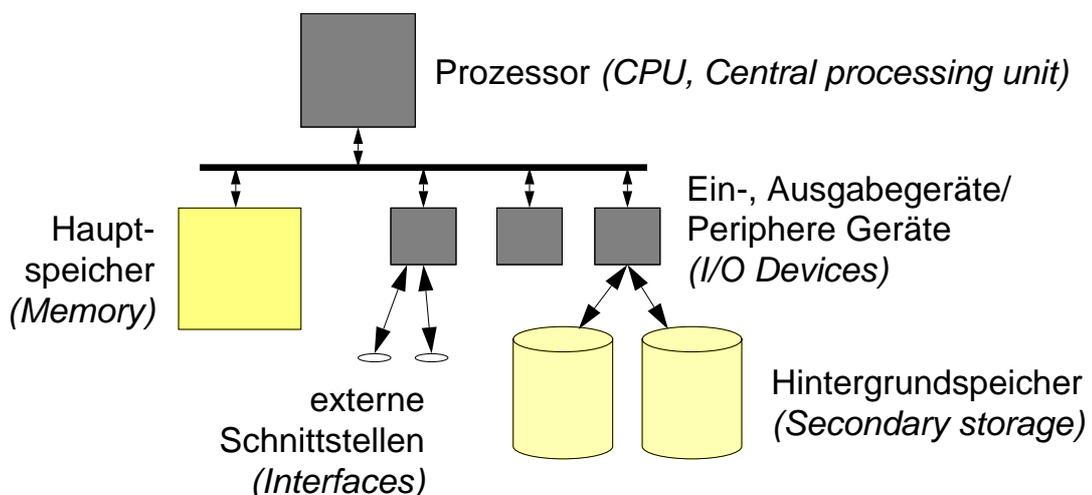
Background

6. Speicherverwaltungssystem

- 6.1 Speichervergabe
 - 6.1.1 Problemstellung
 - 6.1.2. Dynamische Speicherzuteilung
 - 6.1.2.1 Vergabestrategien
 - 6.1.3 Mehrprogrammbetrieb
 - 6.1.3.1 Problemstellung
 - 6.1.3.2 Relokation und Binden
 - 6.1.4 Segmentierung
 - 6.1.5 Seitenadress (Paging)
 - 6.1.5.1 MMU mit Seiten-Kacheltabelle
 - 6.1.6 Segmentierung und Paging
 - 6.1.7 Virtueller Speicher
 - 6.1.7.1 Demnad Paging
 - 6.1.8 Seitenersetzung
 - 6.1.8.1 Optimale Ersetzungsstrategie
 - 6.1.9 Seitenanforderung

6 Speicherverwaltungssystem

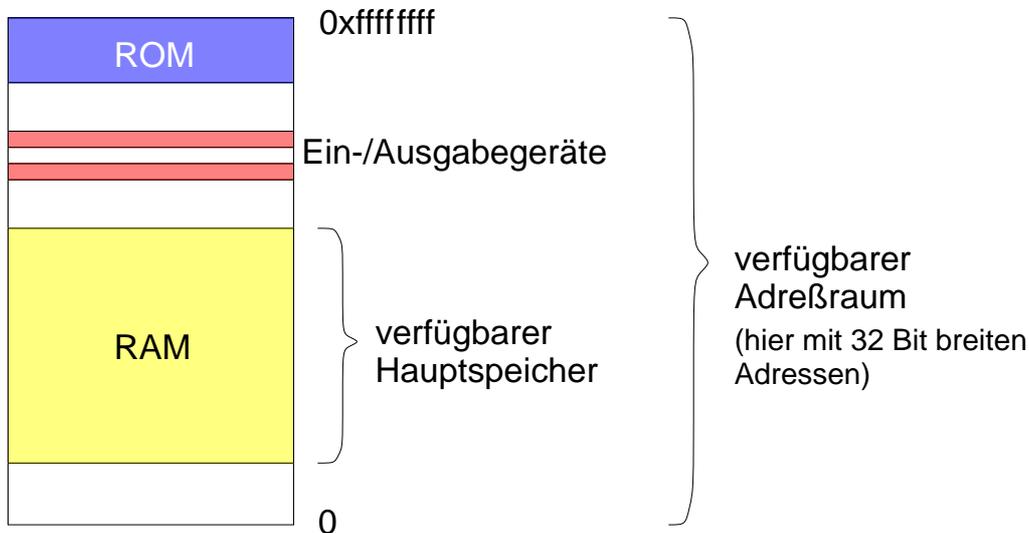
■ Betriebsmittel



6.1 Speichervergabe

1 Problemstellung

■ Verfügbarer Speicher



1 Problemstellung

■ Belegung des verfügbaren Hauptspeichers durch

- Benutzerprogramme
 - Programmbefehle (Code, Binary)
 - Programmdateien
- Betriebssystem
 - Betriebssystemcode
 - Puffer
 - Systemvariablen

■ Zuteilung des Speichers nötig

2 Dynamische Speicherzuteilung

- Segmente
 - zusammenhängender Speicherbereich (Bereich mit aufeinanderfolgenden Adressen)
- Allokation (Anforderung) und Freigabe von Segmenten
- Ein Anwendungsprogramm besitzt üblicherweise folgende Segmente:
 - Codesegment
 - Datensegment
 - Stacksegment (für Verwaltungsinformationen, z.B. bei Funktionsaufrufen)
- Suche nach geeigneten Speicherbereichen zur Zuteilung
- Speicherzuteilungsstrategien nötig

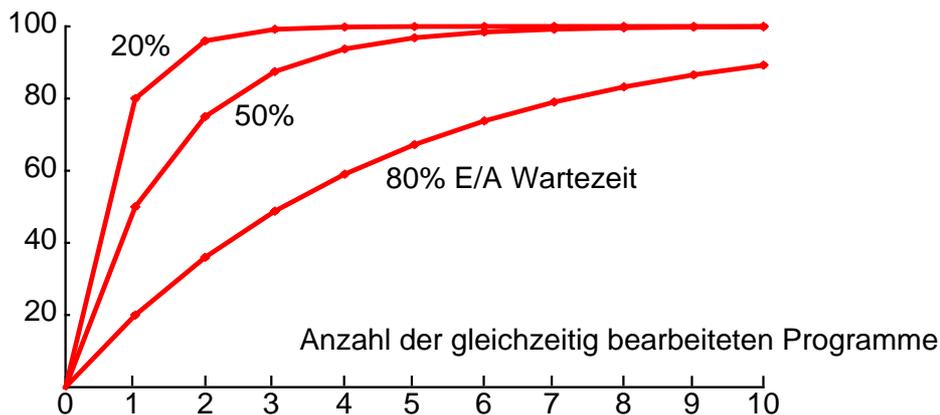
6.1.2.1 Vergabestrategien

- First Fit
 - erste passende Lücke wird verwendet
- Rotating First Fit / Next Fit
 - wie First Fit aber Start bei der zuletzt zugewiesenen Lücke
- Best Fit
 - kleinste passende Lücke wird gesucht
- Worst Fit
 - größte passende Lücke wird gesucht
- Probleme:
 - Speicherverschnitt
 - zu kleine Lücken

3 Mehrprogrammbetrieb

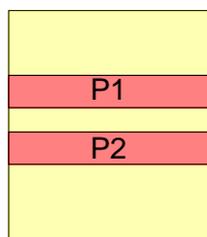
6.1.3.1 Problemstellung

- Mehrere Prozesse laufen (quasi) gleichzeitig
 - Wartezeiten von Ein-/Ausgabeoperationen ausnutzen
 - CPU Auslastung verbessern
- CPU-Nutzung in Prozent, abhängig von der Anzahl der Prozesse



6.1.3.1 Problemstellung

- Mehrere Prozesse benötigen Hauptspeicher
 - Prozesse liegen an verschiedenen Stellen im Hauptspeicher
 - Speicher reicht eventuell nicht für alle Prozesse
 - Schutzbedürfnis des Betriebssystems und der Prozesse untereinander



zwei Prozesse und deren Codesegmente im Speicher

- Ein- und Auslagern von Prozessen
- Hardwareunterstützung

6.1.3.2 Relokation und Binden (*Relocation and Linking*)

- Festlegung absoluter Adressen in den Programmbefehlen
 - z.B. ein Sprungbefehl in ein Unterprogramm oder ein Ladebefehl für eine Variable aus dem Datensegment

- Absolutes Binden (*Compile time*)
 - Adressen stehen fest
 - Programm kann nur an bestimmter Speicherstelle korrekt ablaufen

- Statisches Binden (*Load time*)
 - Beim Laden (Starten) des Programms werden die absoluten Adressen angepaßt (reloziert)
 - Relokationsinformation nötig, die vom Compiler oder Assembler geliefert wird

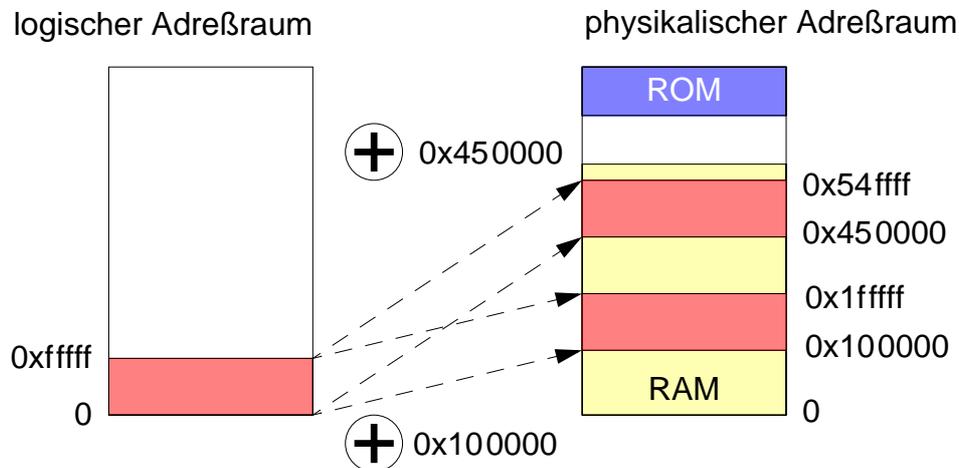
6.1.3.2 Relokation und Binden (*Relocation and Linking*)

- Dynamisches Binden (*Run time*)
 - Bibliotheken werden **einmal** getrennt von sie benutzende Programmsysteme geladen. Die Referenzen werden zur Laufzeit abgesättigt.
 - Spart erheblich Speicherplatz, da der Code gemeinsam genutzt werden kann. Code ist *sharable!*

4 Segmentierung

■ Hardwareunterstützung: Umsetzung logischer in physikalische Adressen

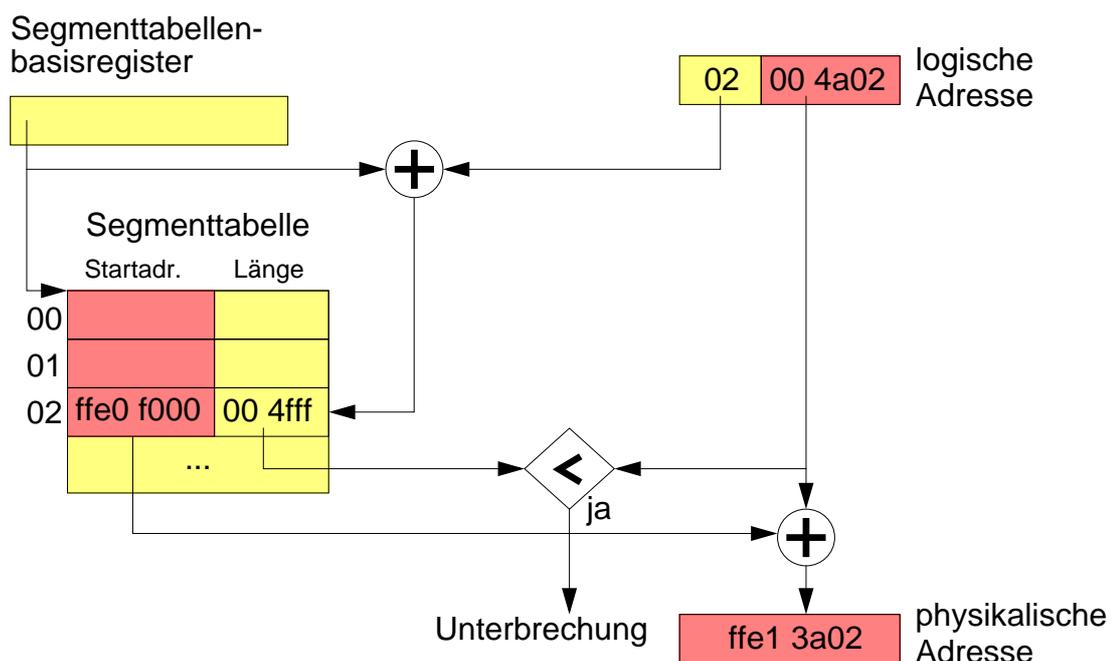
- Prozesse erhalten einen logischen Adreßraum:



Das Segment im logischen Adreßraum kann an jeder beliebige Stelle im physikalischen Adressraum liegen.

4 Segmentierung

■ Realisierung mit Übersetzungstabelle:



4 Segmentierung

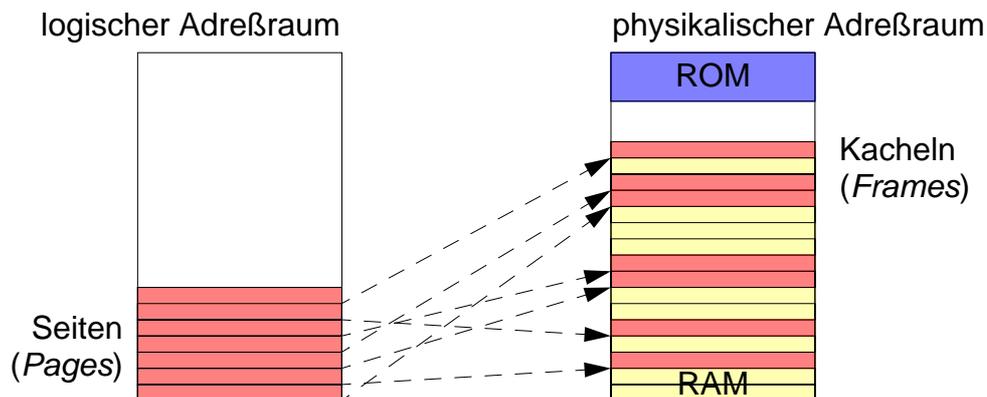
- Hardware wird MMU (*Memory management unit*) genannt
- Schutz vor Segmentübertretung
 - Unterbrechung zeigt Speicherverletzung an
 - Programme und Betriebssystem voneinander geschützt
- Prozeßumschaltung durch Austausch der Segmentbasis
 - jeder Prozeß hat eigene Übersetzungstabelle
- Ein- und Auslagerung vereinfacht
 - nach Einlagerung an beliebige Stelle muß lediglich die Übersetzungstabelle angepaßt werden
- Gemeinsame Segmente möglich
 - Befehlssegmente
 - Datensegmente (*Shared memory*)

4 Segmentierung

- Zugriffsschutz einfach integrierbar
 - z.B. Rechte zum Lesen, Schreiben und Ausführen von Befehlen, die von der MMU geprüft werden
- Fragmentierung des Speichers durch häufiges Ein- und Auslagern
 - es entstehen kleine, nicht nutzbare Lücken
- Kompaktifizieren
 - Segmente werden verschoben, um Lücken zu schließen; Segmenttabelle wird jeweils angepaßt
- lange E/A Zeiten für Ein- und Auslagerung
 - nicht alle Teile eines Segments werden gleich häufig genutzt

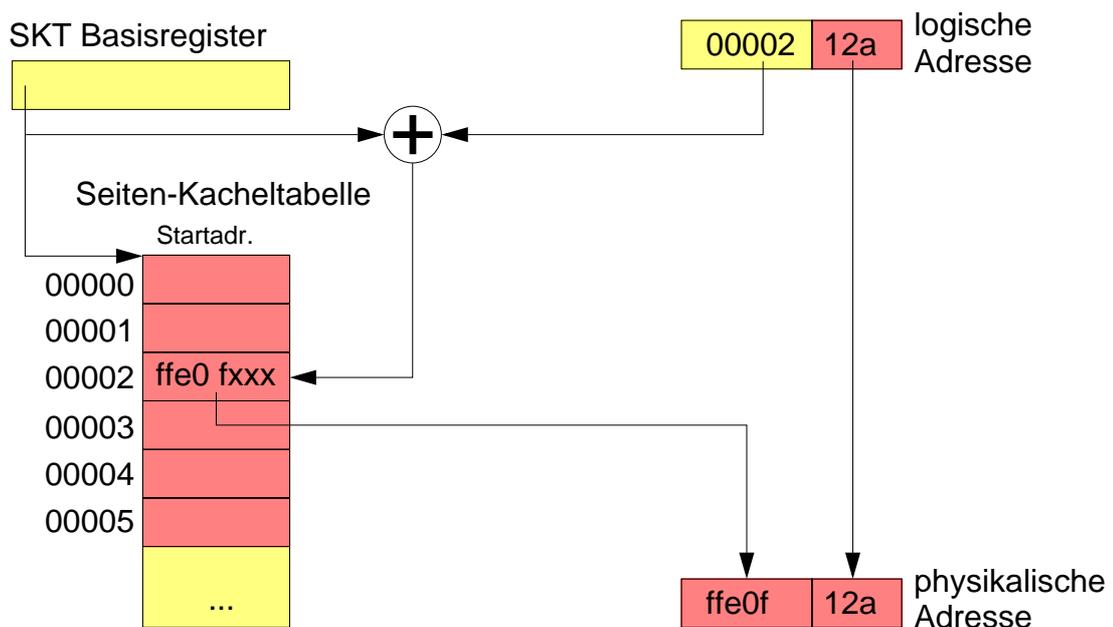
5 Seitenadressierung (*Paging*)

- Einteilung des logischen Adreßraums in gleichgroße Seiten, die an beliebigen Stellen im physikalischen Adreßraum liegen können
 - Lösung des Fragmentierungsproblem
 - keine Kompaktifizierung mehr nötig
 - Vereinfacht Speicherbelegung



6.1.5.1 MMU mit Seiten-Kacheltabelle

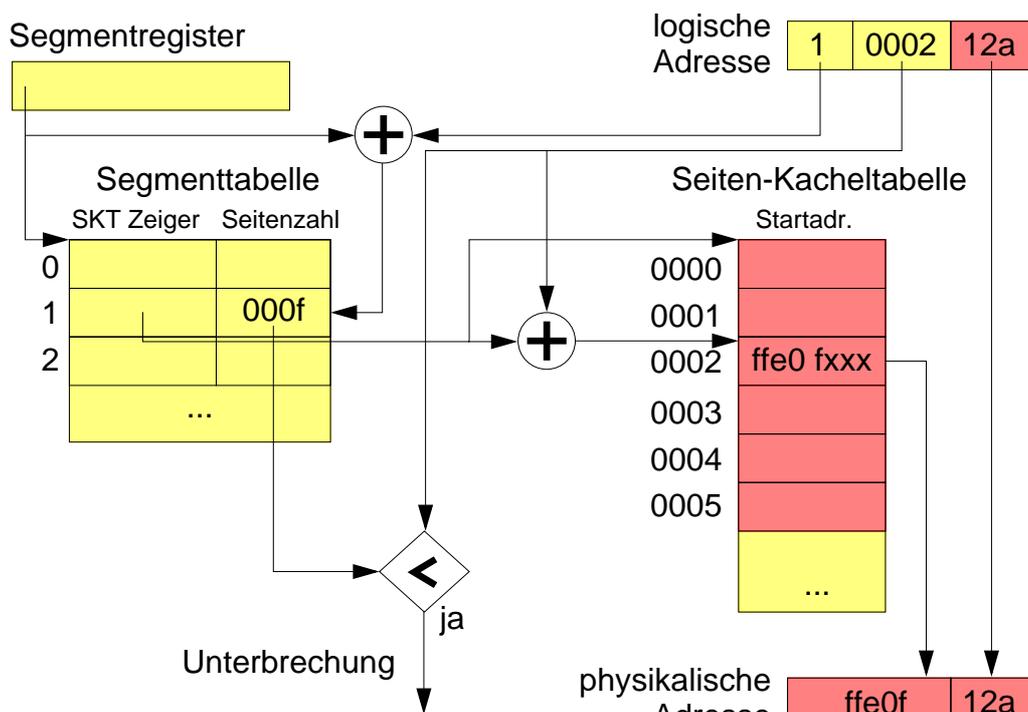
- Tabelle setzt Seiten in Kacheln um



6.1.5.1 MMU mit Seiten-Kacheltabelle

- Seitenadressierung erzeugt internen Verschnitt
 - letzte Seite eventuell nicht vollständig genutzt
- Seitengröße
 - kleine Seiten verringern internen Verschnitt, vergrößern aber die Seiten-Kacheltabelle (und umgekehrt)
 - übliche Größen: 512 Bytes — 8192 Bytes
- große Tabelle, die im Speicher gehalten werden muß
- viele implizite Speicherzugriffe nötig
- nur ein „Segment“ pro Kontext
- Kombination mit Segmentierung

6 Segmentierung und Seitenadressierung



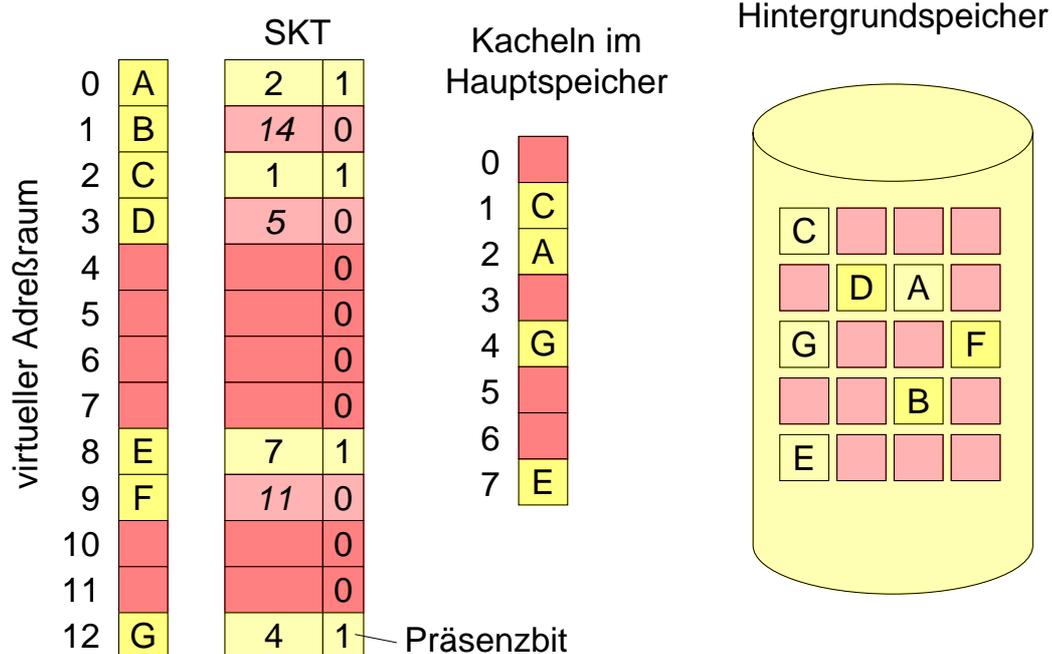
7 Virtueller Speicher

- 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
 - Einlagern benötigter Speicherbereiche
 - Abfangen von Zugriffen auf nicht eingelagerte Bereiche
 - Bereitstellen der benötigten Bereiche

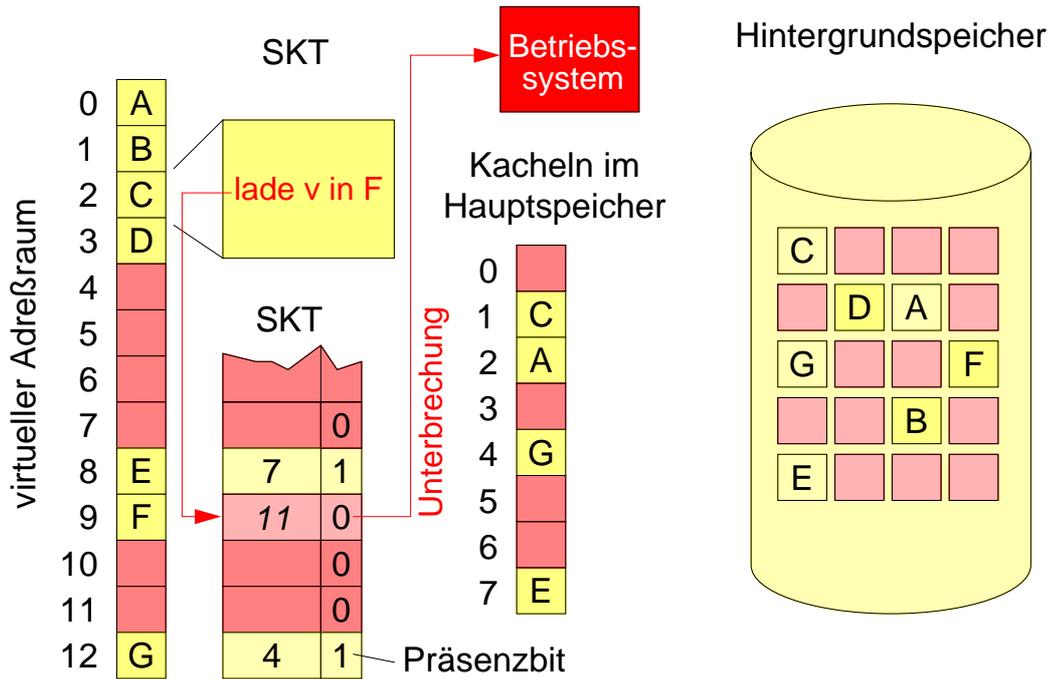
6.1.7.1 Demand Paging

- Bereitstellen von Seiten auf Anforderung



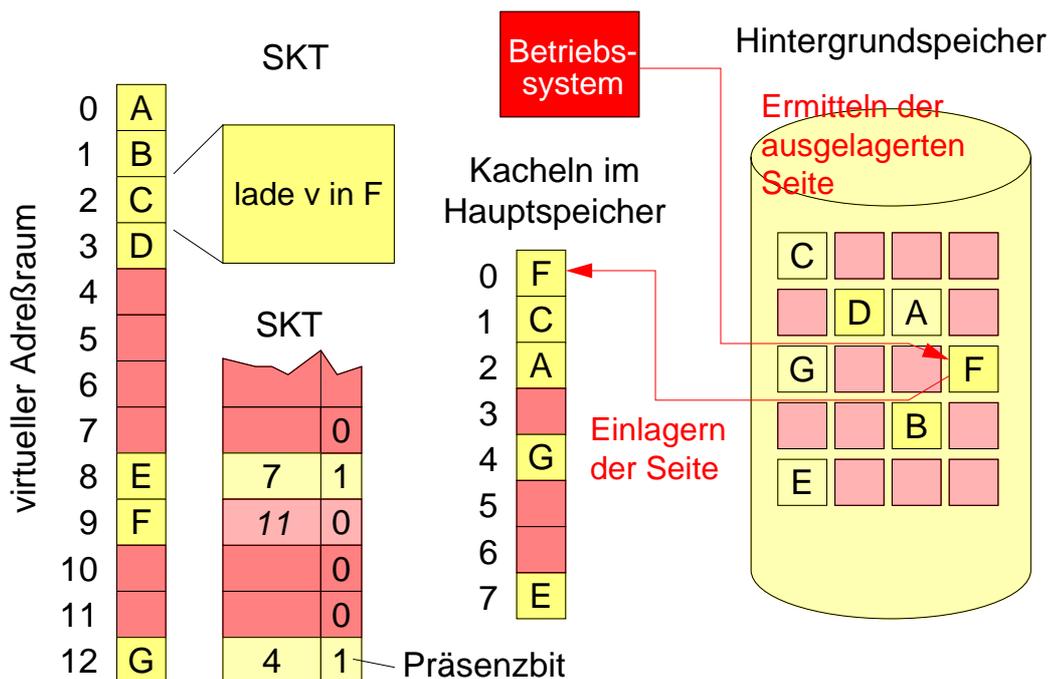
6.1.7.1 Demand Paging

■ Reaktion auf Seitenfehler



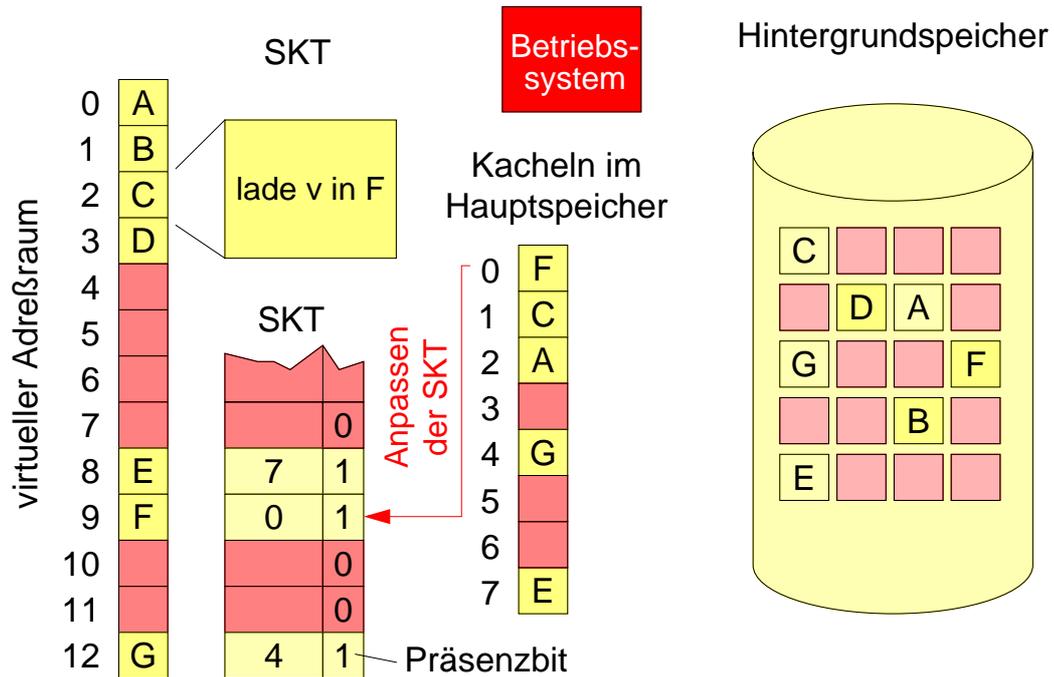
6.1.7.1 Demand Paging

■ Reaktion auf Seitenfehler



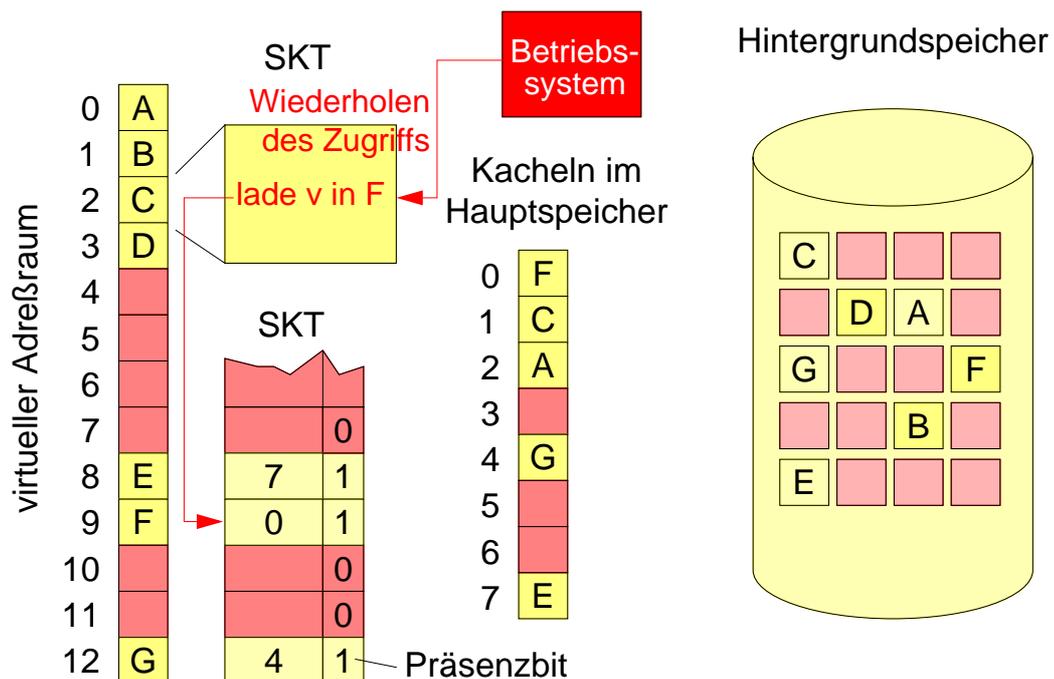
6.1.7.1 Demand Paging

■ Reaktion auf Seitenfehler



6.1.7.1 Demand Paging

■ Reaktion auf Seitenfehler



6.1.7.1 Demand Paging

- Performance 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
= 8 ms Latenzzeit + 15 ms Positionierzeit + 1 ms Übertragungszeit
= 25 ms
 - 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

8 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?

6.1.8.1 Optimale Ersetzungsstrategie

- Es gibt eine optimale Ersetzungsstrategie B_0 :
 - Die tatsächliche Referenzfolge
 - Seite mit dem größten "Vorwärtsabstand" wird ersetzt.
 - d.h. Seite die am längsten in der Zukunft nicht mehr benötigt wird
 - Problem: Referenzfolge ist vorher nicht bekannt.

- Suche nach Strategien, die möglichst nahe an B_0 kommen:
 - z.B. *Least recently used* (LRU):
 - Wähle die Seite zum Auslagern aus, die am längsten nicht mehr referenziert wurde
 - also die Seite mit dem größten "Rückwärtsabstand".

6.1.8.1 Optimale Ersetzungsstrategie

- 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 Freiseitepuffer ausgetragen und wieder dem entsprechenden Prozeß zugeordnet.

8 Seitenanforderung

- Begrenzungen
 - Maximale Seitenmenge: begrenzt durch Anzahl der Kacheln
 - Minimale Seitenmenge: 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)

- Mögliche Zuordnungen (Anzahl der Kacheln pro Prozess):
 - Anzahl der Prozesse bestimmt die Kachelmenge, die ein Prozeß bekommt
 - "Größe" des Programms fließt in die zugeteilte Kachelmenge ein

8 Seitenanforderung

- Sind zuviele Prozesse aktiv, werden welche deaktiviert
 - ◆ Kacheln teilen sich auf weniger Prozesse auf
 - Verbindung mit dem Scheduling nötig
 - Verhindern von Aushungerung
 - Erzielen kurzer Reaktionszeiten
 - Vermeidung von "Seitenflattern" (Thrashing = ausgelagerte Seite wird wieder angefordert)

 - ◆ guter Kandidat: Prozeß mit wenigen Seiten im Hauptspeicher
 - geringe Latenz bei Wiedereinlagerung bzw. wenige Seitenfehler bei Aktivierung und Demand paging

8 Seitenanforderung

- Working Set Model (Arbeitsmengenmodell):
 - Aus der Beobachtung der Vergangenheit eines Prozessverhaltens wird mit (wahrscheinlichkeitstheoretischen) Annahmen auf das Verhalten der Zukunft geschlossen.
 - Ziel ist es, dem Prozess die Seiten zur Verfügung zu stellen, die die Anzahl der Seitenanforderungen (*Page Faults*) minimiert.