6. Speicher- und Adressraumverwaltung

- Überblick
  - 6.1 Speicherverwaltung im Betriebssystem
  - 6.2 Speicherzuweisung und Auslagerung
  - 6.3 Verdrängungsstrategien

6.1 Speicherverwaltung im Betriebssystem

- Speicher ist ein „spezielles“ Betriebsmittel, das vom BS besonders verwaltet werden muss
- Speicherallokation = Zuordnung von Speicher
- Zwecke, für die Speicher benötigt wird:
  - Prozesse
    - Stack
    - Heap
    - Programm-Code und -Daten
  - Das BS selbst!
    - BS-Code
    - BS-Daten
    - Dynamisch erzeugte Verwaltungsstrukturen (PCBs etc.)

Statische und dynamische Speicherallokation

- Statische Speicherallokation: Feste Einteilung des Speichers in mehrere Bereiche
  - Nehmen einen Prozess vollständig auf
  - Prozess blockiert $\Rightarrow$ Verdrängung aus aktuellem Laufbereich und Auslagerung auf den externen Speicher
  - Wiedereinlagerung nach Wechsel blockiert $\rightarrow$ bereit
  - Nachteil: Nur ganze Prozesse werden verdrängt $\Rightarrow$ ineffizient

- Dynamische Speicherallokation: Prozessteile werden – transparent für den Benutzer – zur Laufzeit und erst bei Bedarf ein- und ausgelagert
  - Grundlage: Virtuelle Adressierung so, dass Prozesse verdrängt und ohne zusätzlichen Programmieraufwand in anderem Speicherbereich ausgeführt werden können $\Rightarrow$ Realokierbare Prozesse
  - Beim Zugriff auf fehlenden Inhalt wird ein Interrupt (Seitenfehler, Page Fault) ausgelöst, der das Nachladen des Inhalts initiiert

Datenstruktur zur Adressraumverwaltung

- Gedankenexperiment: wir erstellen einen Adressraum

- Benötigte Struktur dafür muss für jede virtuelle Adresse *V* die folgenden Dinge liefern:
  - Gültigkeit (ist *V* eine gültige Adresse in diesem Adressraum, oder löst ein Zugriff auf *V* einen Speicherfehler aus?)
  - Wenn gültig:
    - Zugehörige Physische Adresse *P*
    - Lesen, Schreiben, Ausführen von dieser Adresse erlaubt?
    - Zugriff für unprivilegierte Instruktionen erlaubt?

    $\Rightarrow$ je Eintrag brauchen wir mind. ein Maschinenwort… (wir müssen *P* speichern plus ein paar Bits extra)

Datenstruktur zur Adressraumverwaltung (2)

- Frage: wie granular wollen wir *V* zuordnen können?
  - Extrem 1: ultrafein, d.h. jedes Byte einzeln
    ⇒ dann brauchen wir mehr Speicher für die Verwaltungs-struktur als das System überhaupt Speicher hat
  - Extrem 2: ultragrob, z.B. 64MB
    ⇒ dann belegt jeder Prozess mind. 64MB Speicher, da wir kleinere Einheiten nicht verwalten können

- Trade-Off zwischen Effizienz des Lookups, Größe der Verwaltungsstruktur und anderen Überlegungen
  - Resultat für die meisten Architekturen: 4kB

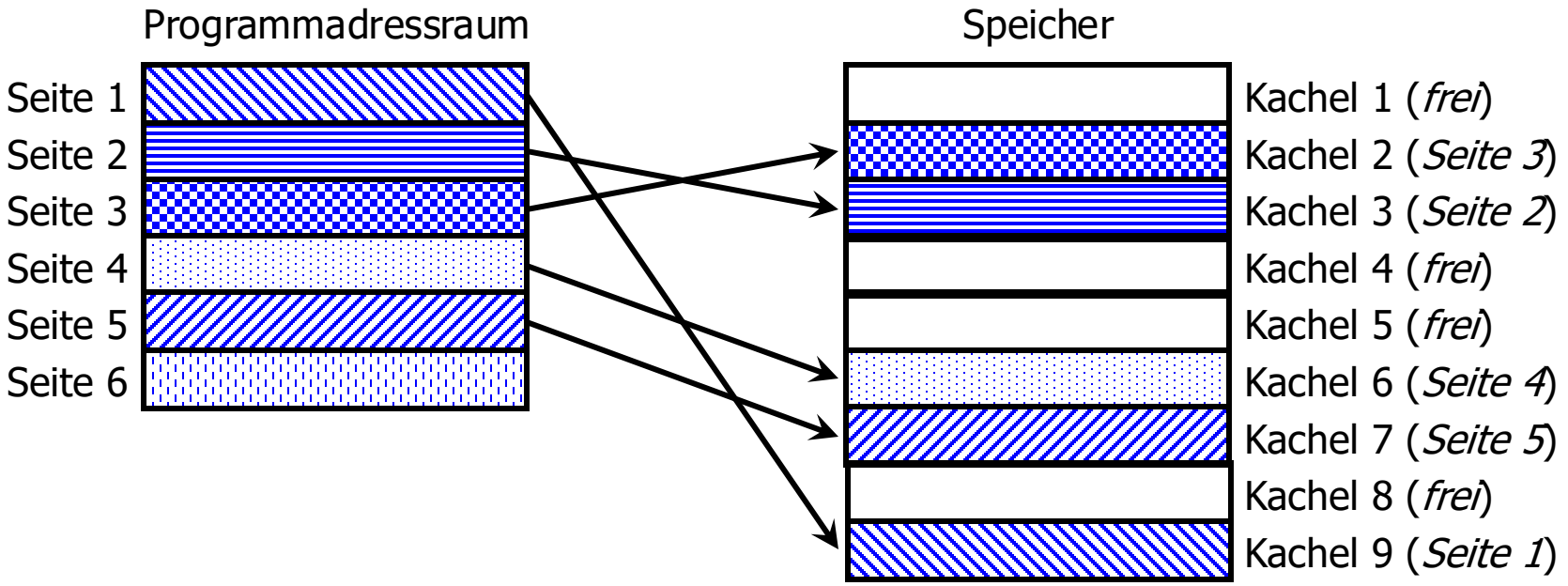
  ⇒ virtuelle Verwaltungseinheit ist eine **Seite (page)**

  ⇒ physisches Pendant heißt **Kachel (page frame)**

  ⇒ **Seitentabelle** oder Seite-Kachel-Tabelle **(page table)**

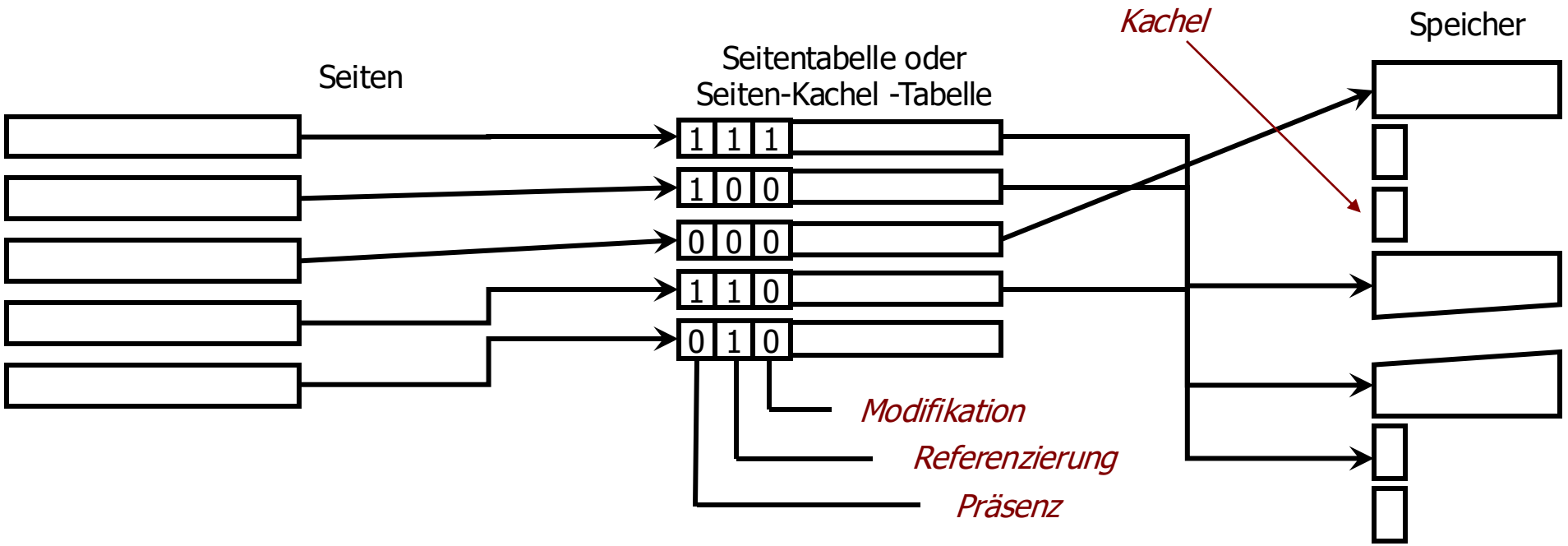
Seitenadressierung

- Bei Seitenadressierung (paging) werden der virtuelle und der physikalische Speicher in Stücke fester Länge eingeteilt
  - Stücke im logischen Adressraum heißen Seiten (pages)
  - Stücke im physikalischen Speicher heißen Kacheln (page frames)
  - Seiten und Kacheln sind gleich groß



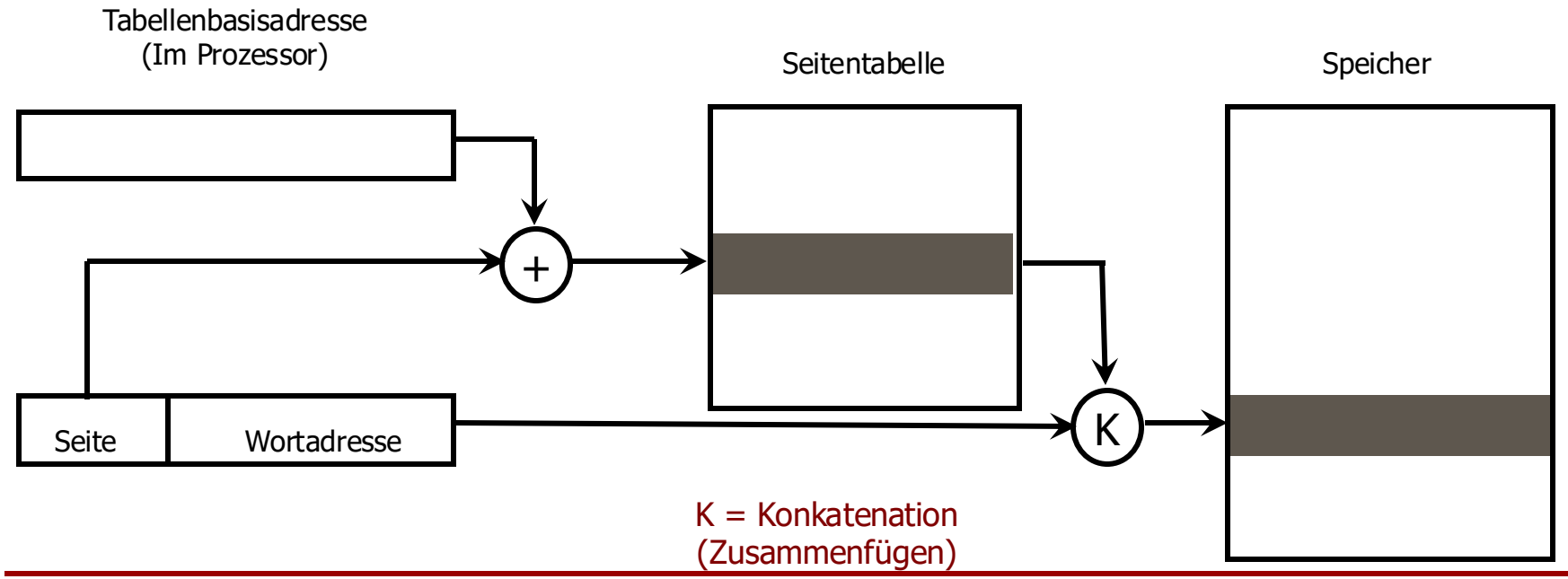
Verbindung durch Seitentabelle

- Neben der physikalischen Adresse enthält jeder Eintrag der Seitentabelle noch Informationen über
  - Seite im Hauptspeicher vorhanden? Präsenzbit (presence bit P)
  - Wurde auf die Seite bereits zugegriffen? Referenzbit (reference bit R oder access bit A)
  - Wurde die Seite modifiziert? Modifikationsbit (dirty bit D oder modification bit M)



Adressübersetzung mit Seitentabelle

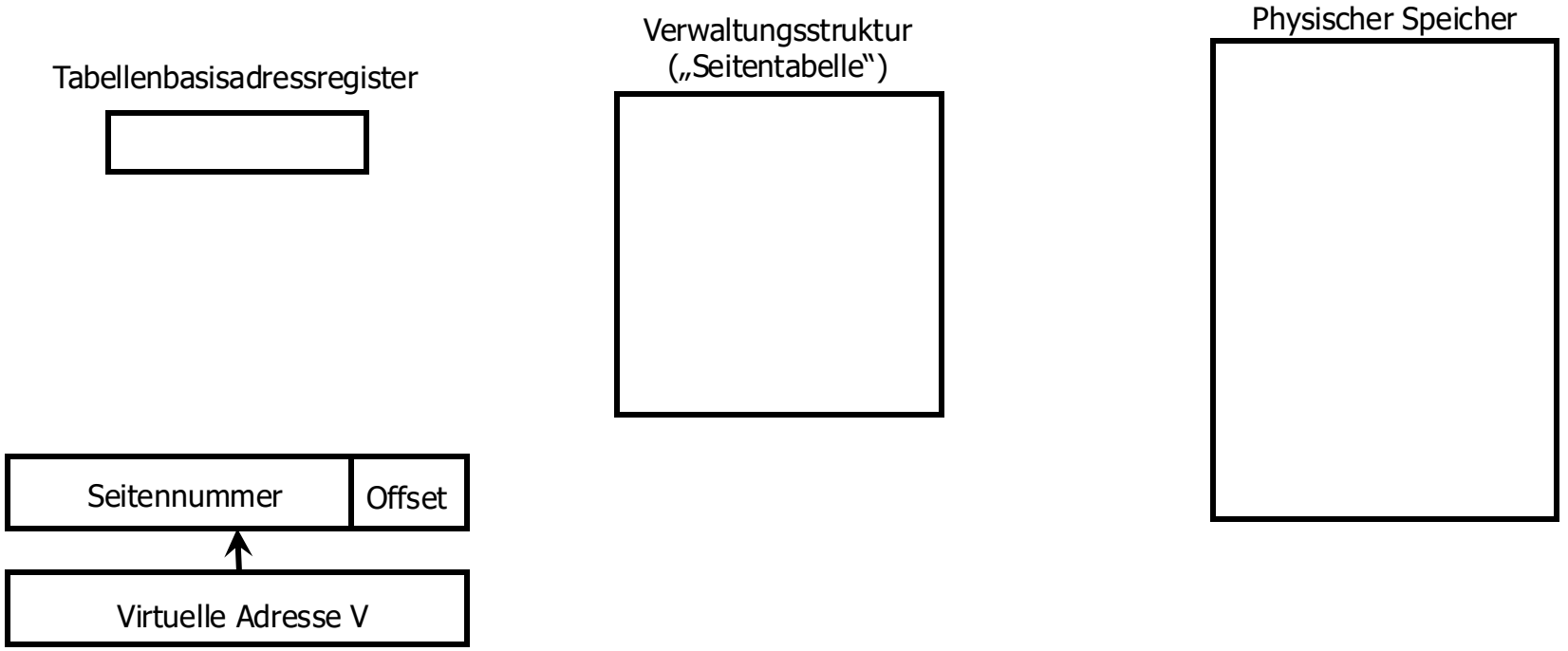
- Seitenbasierte Speicherbelegung ist Standard in Betriebssystemen
- Notwendige Angaben
  - Wo befindet sich die Seitentabelle ⇒ Tabellenbasisregister
  - Welche Seite wird angesprochen ⇒ Seitennummer
  - Adresse innerhalb der Seite ⇒ Wortadresse (offset, displacement)



Adressübersetzungsverfahren

- Prozessor enthält spezielles Kontrollregister für Basisadresse der Übersetzungstabelle (nur privilegiert, also vom BS, konfigurierbar)

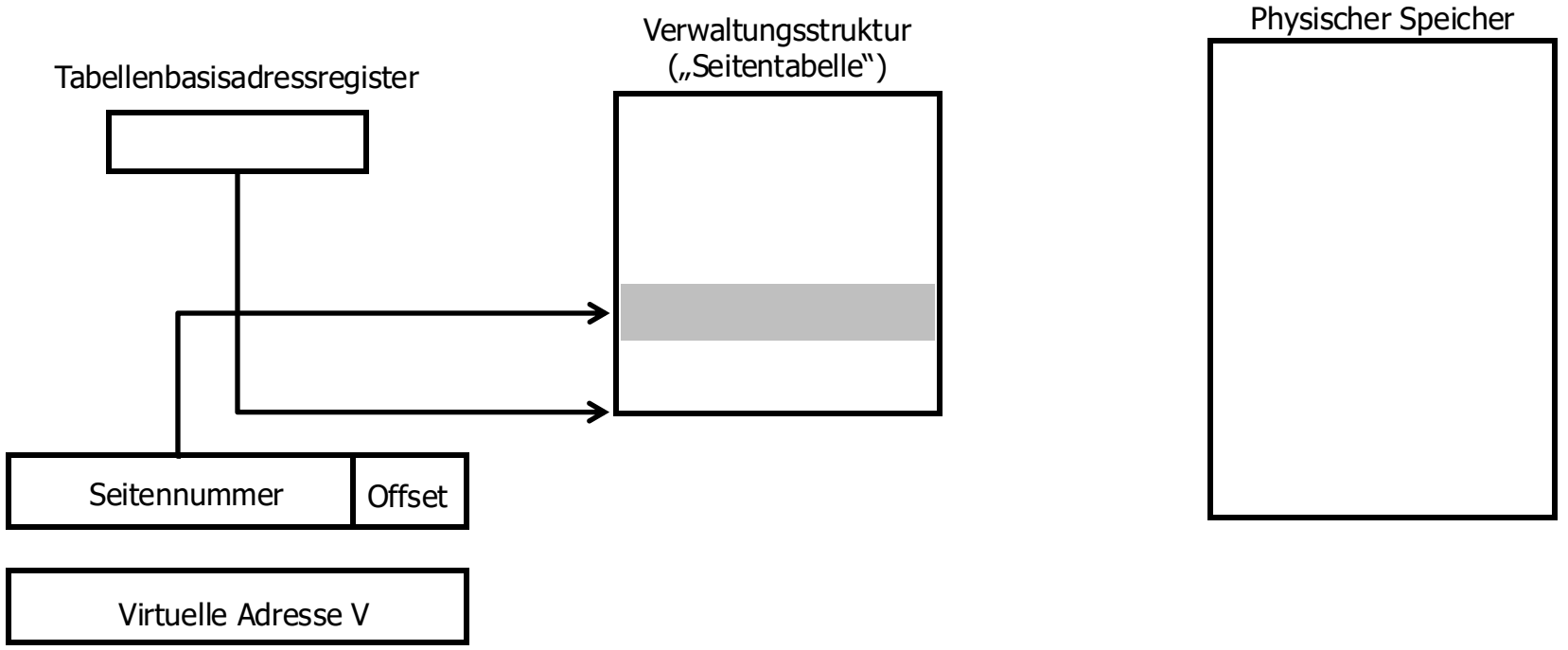
- Schritt 1: zerlege V in Seitennummer und Offset



Adressübersetzungsverfahren

- Prozessor enthält spezielles Kontrollregister für Basisadresse der Übersetzungstabelle (nur privilegiert, also vom BS, konfigurierbar)

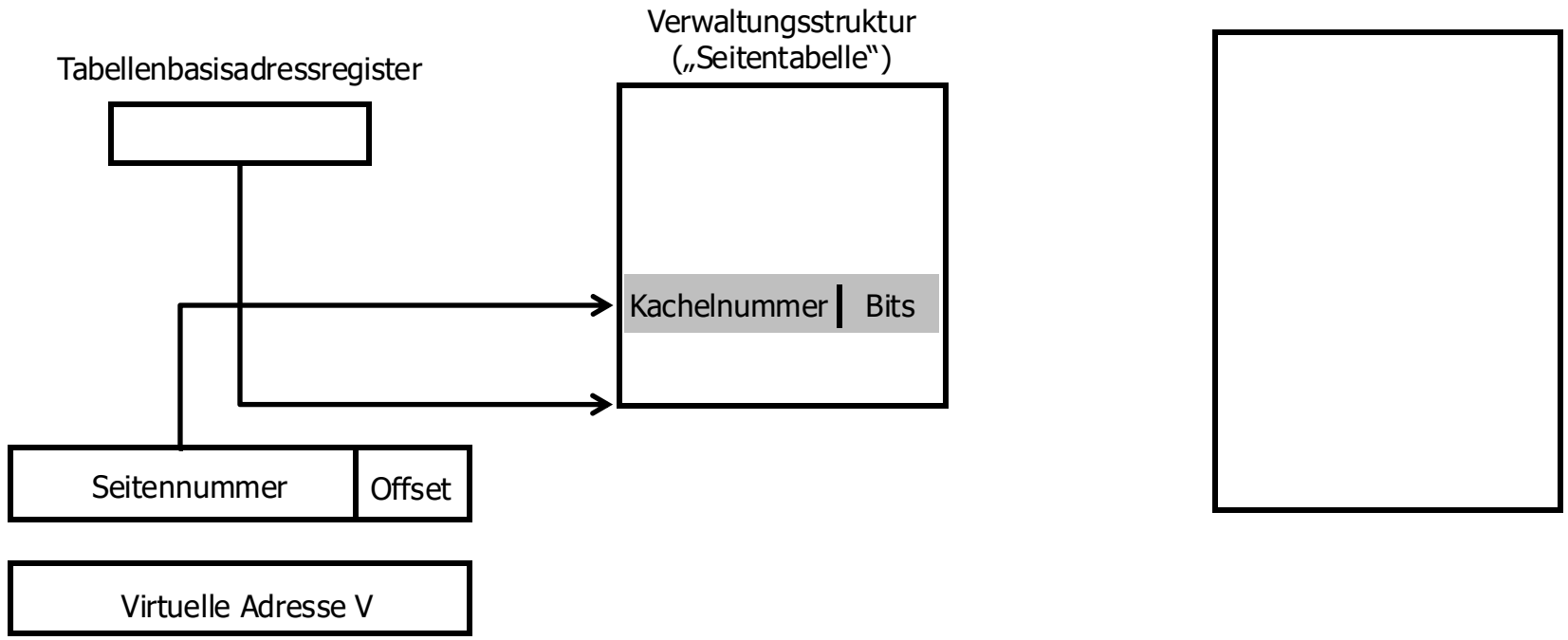
- Schritt 2: verwende Seitennummer als Index in Seitentabelle



Adressübersetzungsverfahren

- Prozessor enthält spezielles Kontrollregister für Basisadresse der Übersetzungstabelle (nur privilegiert, also vom BS, konfigurierbar)

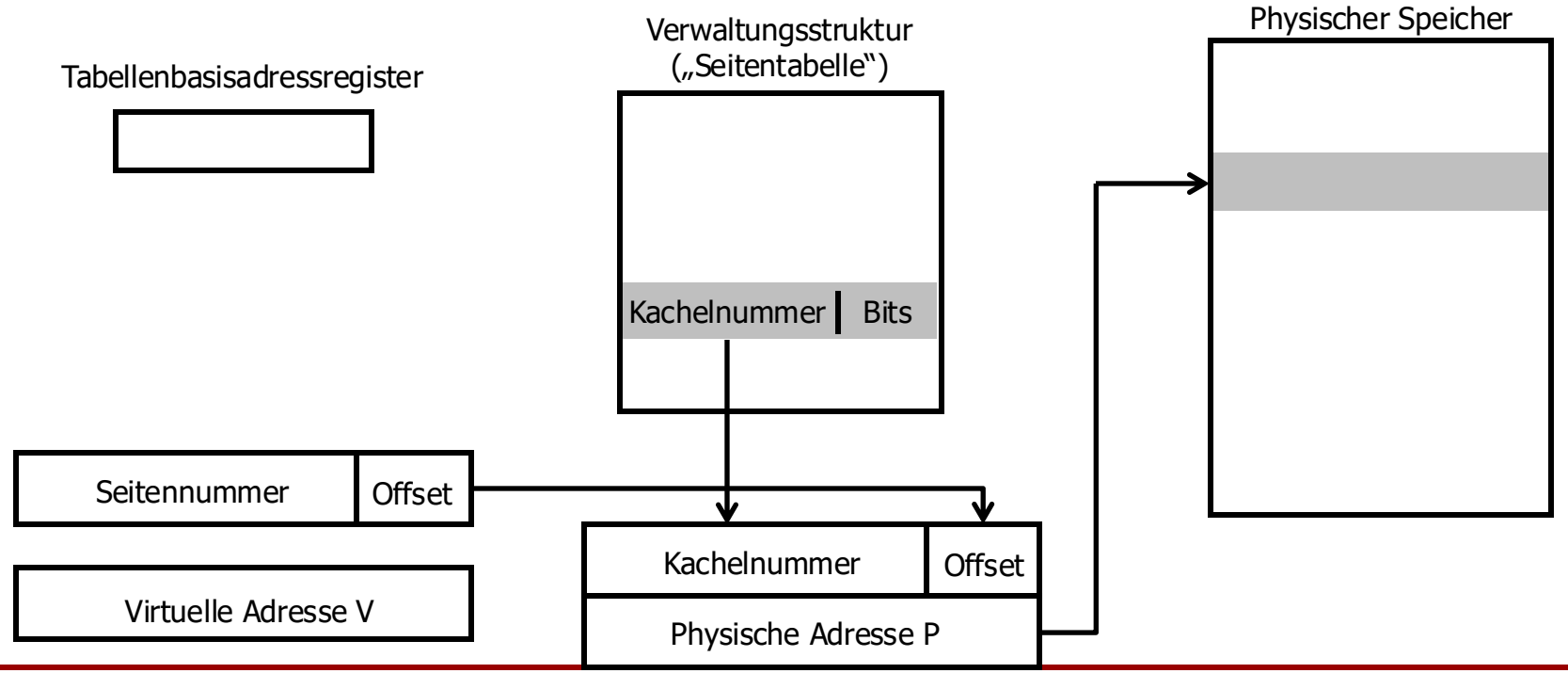
- Schritt 3: prüfe Gültigkeit und Zugriffsrechte in den Metadaten des Eintrags („Bits“), lese Kachelnummer



Adressübersetzungsverfahren

- Prozessor enthält spezielles Kontrollregister für Basisadresse der Übersetzungstabelle (nur privilegiert, also vom BS, konfigurierbar)

- Schritt 4: bilde aus Kachelnummer und Offset das Ergebnis, die physische Adresse P



Datenstruktur zur Adressraumverwaltung (3)

- Annahme: 32-Bit-Architektur, d.h. 4GB Adressraum, Verwaltungseinheit 4kB

$\Rightarrow$ es gibt 4GB/4kB=2^{20} Seiten pro virtuellem Adressraum

- Wenn die Verwaltungsstruktur eine einfache Tabelle ist, wäre sie 2^{20} Einträge lang (wovon häufig die allermeisten ungültig sein werden) [Wie ein 2000-seitiges Telefonbuch, wo hinter 99,9% der Namen keine Rufnummer steht…]

- Für 64-Bit-Architekturen erst recht nicht handhabbar (2^{52}!)

Datenstruktur zur Adressraumverwaltung (4)

- Abhilfe-Idee 1: invertierte Tabelle
  - Ein Eintrag pro Kachel, nicht pro Seite
  - Größe der Tabelle skaliert mit dem tatsächlich vorhandenen Speicher, nicht mit der virtuellen Adressraum-Größe

- Problem: wenn das eine Kachel-Tabelle ist, wie löst man die Aufgabe „gegeben V, suche P"?
  - [Wer in diesem Telefonbuch hat die Rufnummer 33344499?]
  - Naiv: lineare Suche durch die gesamte Tabelle
  - Etwas trickreicher: durch Einsatz von Hash-Verfahren muss nur ein Teil der Einträge durchsucht werden…

  $\Rightarrow$ letztlich dennoch ineffizient

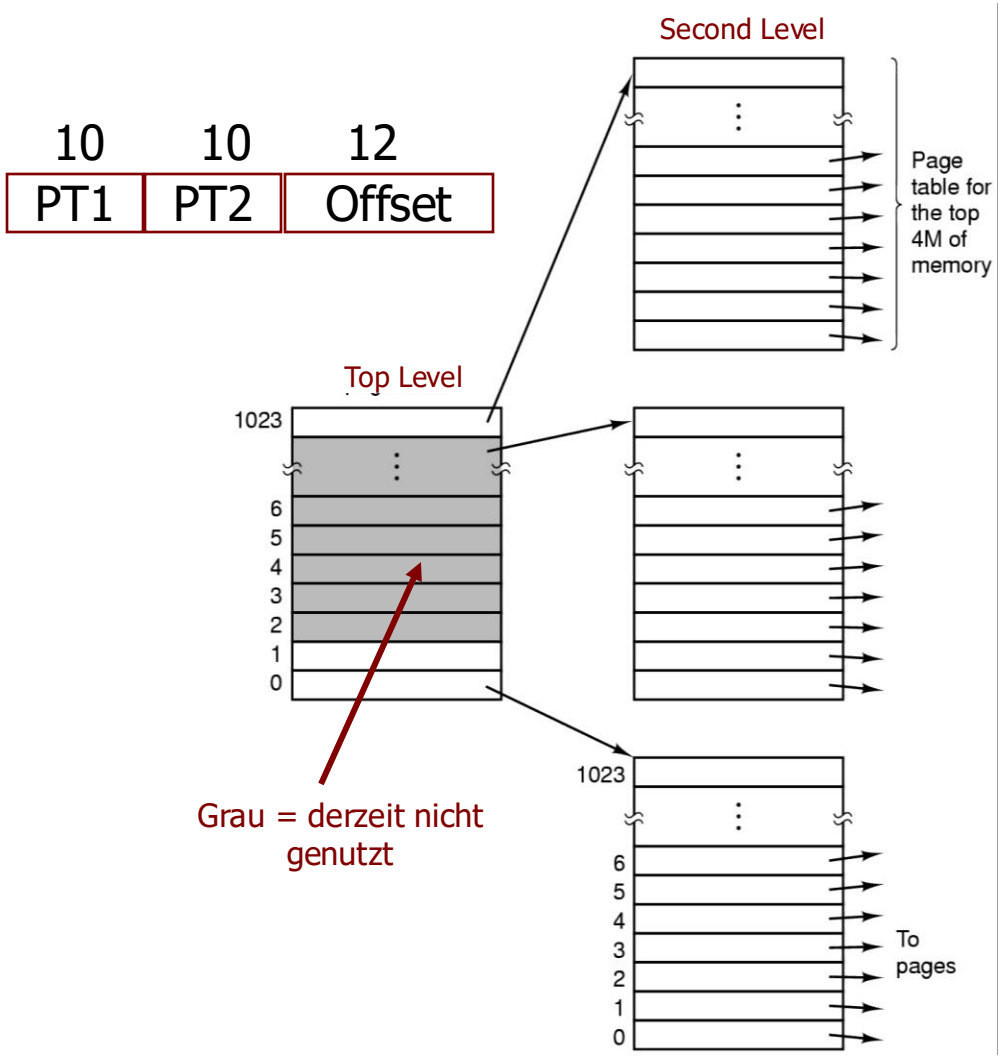
Datenstruktur zur Adressraumverwaltung (5)

- Abhilfe-Idee 2: mehrstufige Tabelle
  - Anstatt alle 20 Bits mit einer einzigen 2^{20}-Tabelle zu übersetzen, mache zwei Schritte à 10 Bits
  - Erste Tabelle enthält 2^{10} Einträge
  - Nur an Stellen, wo virtuelle Adressen tatsächlich übersetzt werden sollen, verweist die erste Tabelle auf eine zweite Tabelle zur Übersetzung der übrigen Bits dieses Adressbereichs

$\Rightarrow$ für üblichen Fall des spärlich besetzten Adressraums: sehr effiziente Darstellung

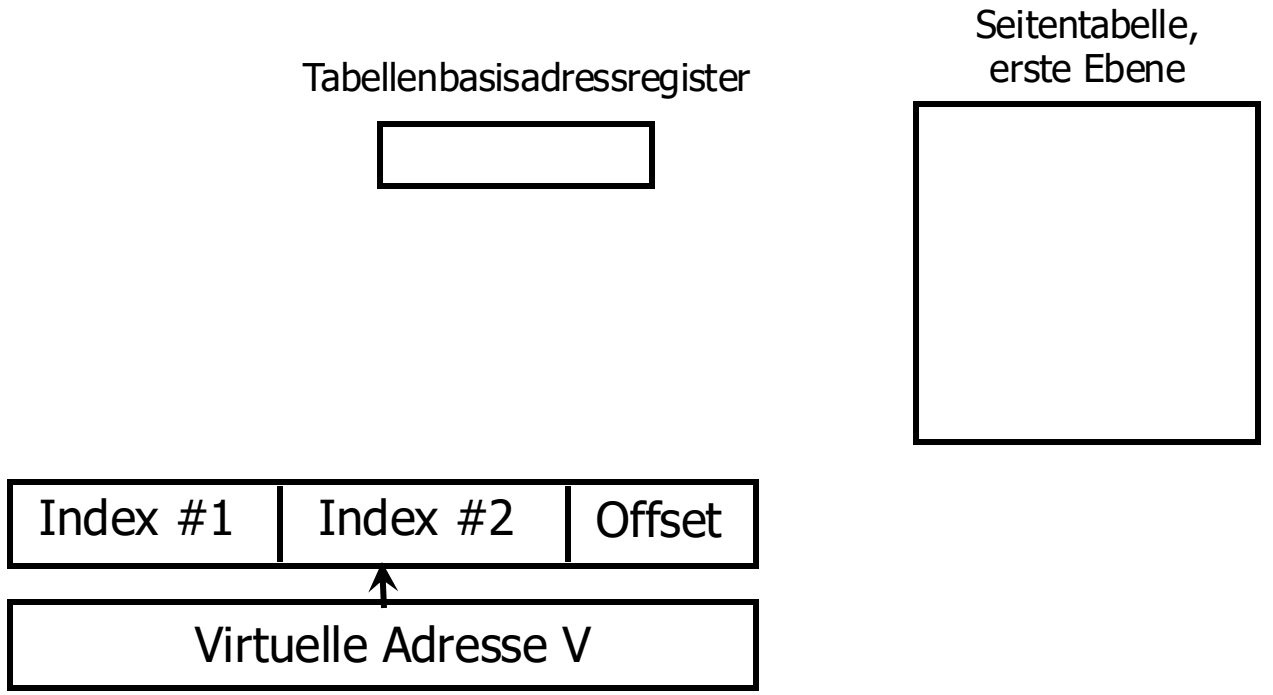
  - Leerer Adressraum: eine Tabelle à 2^{10}
  - Adressraum mit fünf Seiten: max. sechs Tabellen à 2^{10}
  - Voll besetzter Adressraum: $2^{10}+1$ Tabellen à 2^{10} (auch nicht schlimmer als eine einzige große Tabelle)

Mehrstufige Seitentabellen



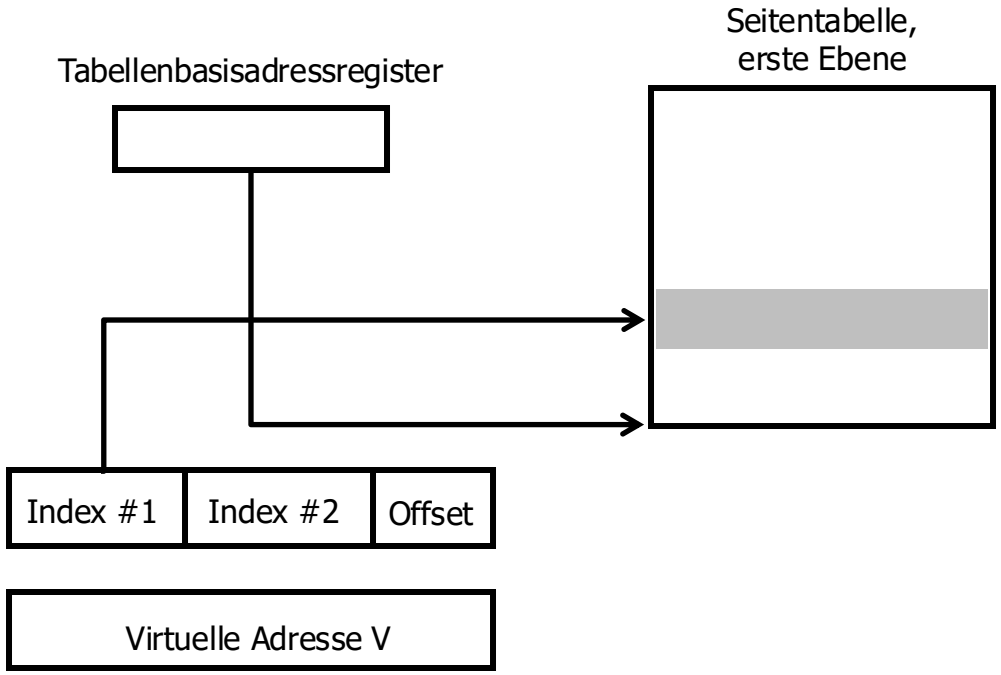
Mehrstufiges Adressübersetzungsverfahren

- Prozessor enthält spezielles Kontrollregister für Basisadresse der ersten Ebene der hierarchischen Übersetzungstabelle (nur privilegiert, also vom BS, konfigurierbar)
- Schritt 1: zerlege V in Tabellen-Indices und Offset



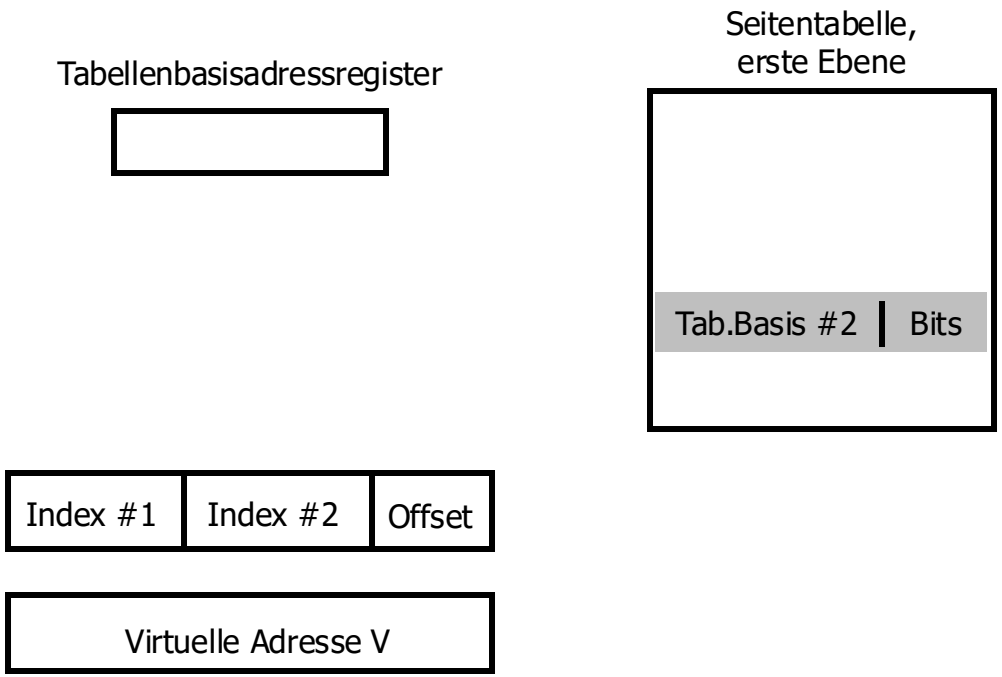
Mehrstufiges Adressübersetzungsverfahren

- Prozessor enthält spezielles Kontrollregister für Basisadresse der ersten Ebene der hierarchischen Übersetzungstabelle (nur privilegiert, also vom BS, konfigurierbar)
- Schritt 2: benutze Tabellenbasisadressregister und Index #1 für Lookup in der ersten Tabellenstufe



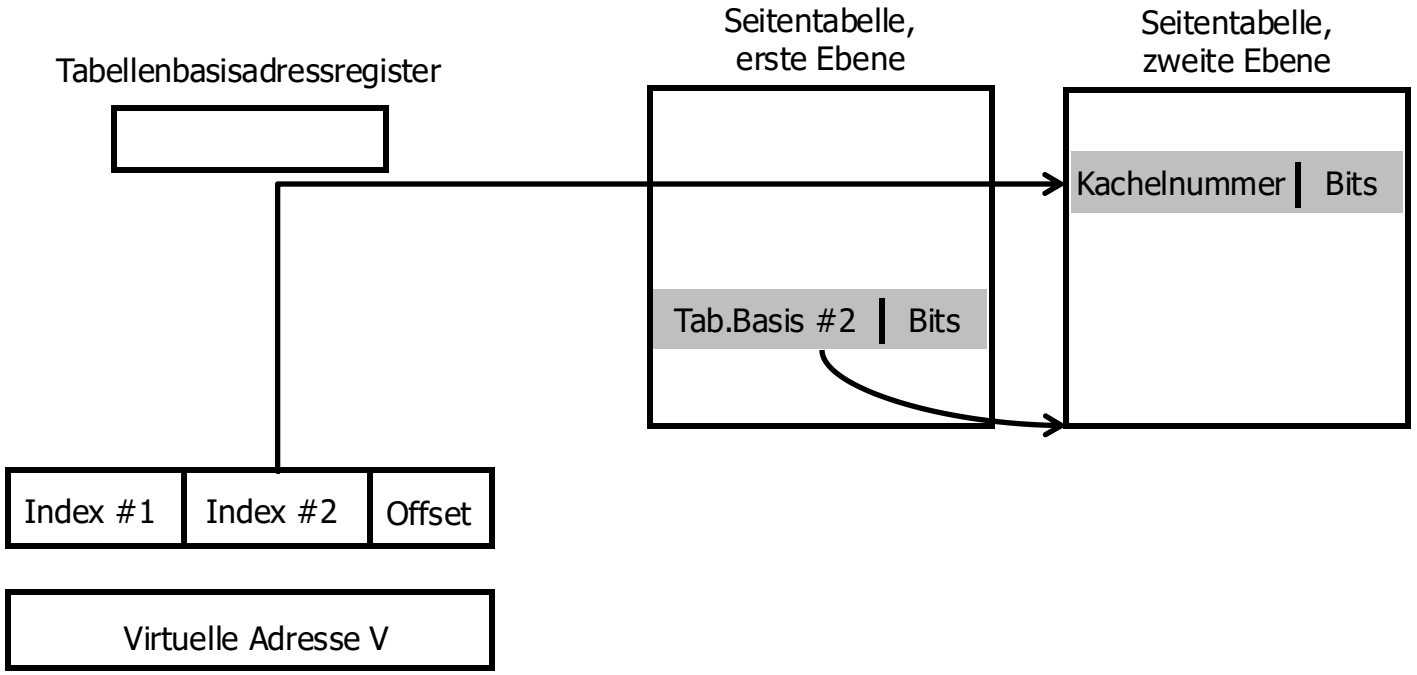
Mehrstufiges Adressübersetzungsverfahren

- Prozessor enthält spezielles Kontrollregister für Basisadresse der ersten Ebene der hierarchischen Übersetzungstabelle (nur privilegiert, vom BS konfigurierbar)
- Schritt 3a: wenn gültig, enthält Eintrag Basisadresse einer Tabelle der zweiten Ebene für die Übersetzung der restlichen Bits



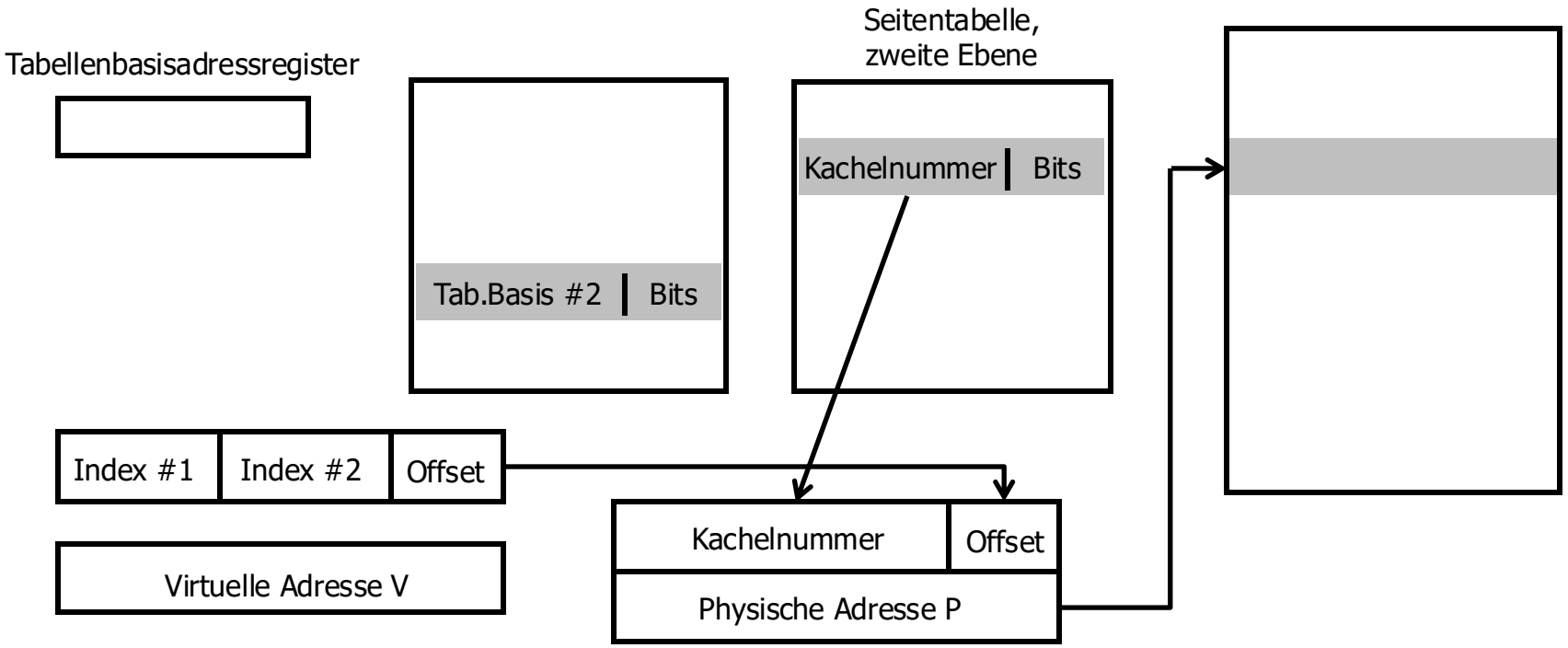
Mehrstufiges Adressübersetzungsverfahren

- Prozessor enthält spezielles Kontrollregister für Basisadresse der ersten Ebene der hierarchischen Übersetzungstabelle (nur privilegiert, vom BS konfigurierbar)
- Schritt 3b: benutze diese zweite Basisadresse und Index #2 für einen weiteren Tabellen-Lookup



Mehrstufiges Adressübersetzungsverfahren

- Prozessor enthält spezielles Kontrollregister für Basisadresse der ersten Ebene der hierarchischen Übersetzungstabelle (nur privilegiert, vom BS konfigurierbar)
- Schritt 4: wenn auch dieser Eintrag gültig, füge wieder Kachelnummer und Offset zu P zusammen



Metadaten in der Seitentabelle

- Je ein Bit für (nur eine Auswahl, es gibt noch mehr):
  - Seite-Kachel-Eintrag ist gültig (**P**resent)
  - Seite darf gelesen werden (**R**ead)
  - Seite darf beschrieben werden (**W**rite)
  - Seite darf ausgeführt werden (e**X**ecute)
  - Seite darf von unprivilegiertem Code benutzt werden (**U**ser); manchmal auch mit umgekehrter Logik als (**S**upervisor)
  - Auf die Seite wurde bereits lesend zugegriffen (**A**ccessed)
  - Auf die Seite wurde bereits schreibend zugegriffen (**D**irty)
- **A** und **D** werden von der CPU aktualisiert(!)

6.2 Speicherzuweisung und Auslagerung

- BS verwaltet sowohl die Kacheln selbst (frei/belegt, und womit?) als auch die Adressräume (welche Kachel ist an welcher virtuellen Adresse in welchen Adressräumen verfügbar?)

- Wir unterscheiden zwei Fälle, in denen etwas zu tun ist (Speicher zuweisen bzw. wieder entziehen) und wir eine Strategie benötigen:
  - Nachschub: ein neuer Prozess startet oder ein bestehender fordert zusätzlichen Speicher an
  - Verdrängung: es besteht Speicherknappheit, so dass durch Auslagerung Platz geschaffen werden muss

Nachschubstrategien (Fetch Policies)

- Auf Verlangen (Demand Paging): Seiten werden erst bei Bedarf nachgeladen
  - Bedarf äußert sich durch Zugriff auf ungültige virtuelle Adresse ($\Rightarrow$ Seitenfehler tritt auf, BS „löst das Problem“)
- Vorgeplant (Pre-Paging) Transport einer Seite in den Arbeitsspeicher so, dass die Seite zum Referenzierungszeitpunkt zur Verfügung steht
  - Voraussetzung: exaktes Prozessverhalten im Voraus bekannt, meist nicht erfüllt
- Kombinierte Ansätze
  - Beim Prozess-Start Pre-Paging: Programm laden, mind. eine Seite für je Heap und Stack bereitstellen, usw.
  - Während Laufzeit: Demand Paging

Verdrängung

- Speicherknappheit erfordert Verdrängung, d.h. Auslagerung von Speicherinhalten auf andere Medien (z.B. Festplatte)
- Möglichkeit 1: Prozesse vollständig ein- und auslagern
  - Prozess blockiert ⇒ Auslagerung
  - Prozess wieder bereit ⇒ Wiedereinlagerung
  - Aber: zu viel Arbeit, insb. bei kurzen Blockierungen ⇒ ineffizient

- Möglichkeit 2: Prozessteile (Seiten!) werden – transparent für den Benutzer – zur Laufzeit und erst bei Bedarf ein- und ausgelagert
  - Ausgelagerte Seiten werden in der Seitentabelle als „ungültig" markiert, beim Zugriff darauf wird ein Seitenfehler ausgelöst

  ⇒ BS kann Seite wieder einlagern, ohne dass der Prozess merkt, dass sie weg war

Verdrängung

- Auslagerung von Seiten nutzt die Seitentabelle „kreativ“:
  - Gültiger Eintrag in der Seitentabelle enthält:
    - Gültig-Bit P = 1
    - Virtuelle Adresse V
    - Zugriffsrechtebits R,W,X,U
    - Zugriffsprotokollbits A,D
  - Für einen ungültigen Eintrag muss nur gelten: Gültig-Bit P = 0
  - Die übrigen Felder werden von der MMU dann ignoriert
  - Diese können also z.B. dazu genutzt werden, die Blocknummer auf der Festplatte zu speichern, wohin der Seiteninhalt ausgelagert wurde!

- Wenn das D-Bit in der Seitentabelle nicht gesetzt ist, kann das Schreiben auf die Festplatte u.U. unterbleiben!

Detaillierter Ablauf

Belegen des VS

Ersatzspeicher belegen

Initialisierung

Zugriff

Seite vorhanden?

Nein: Seitenfehler aktivieren

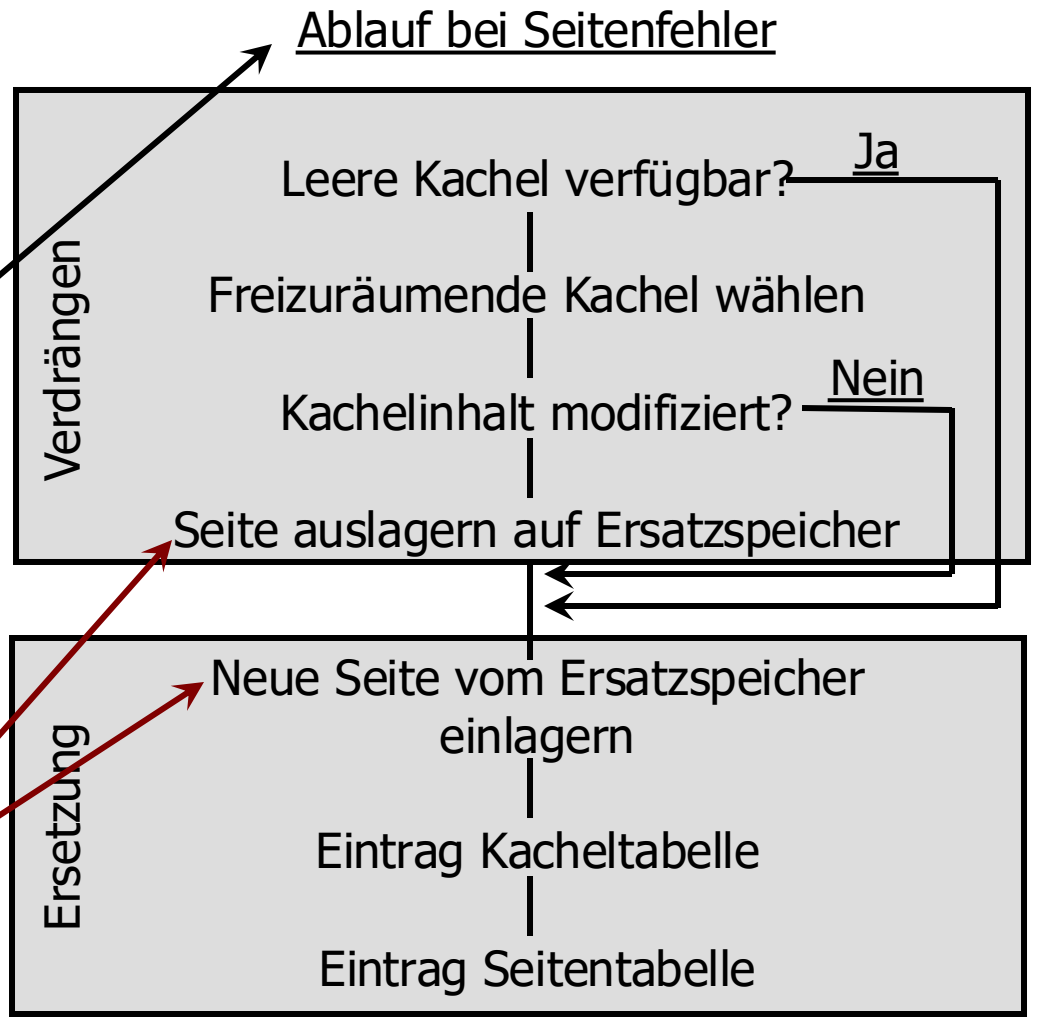
Ja: Setze Verarbeitung fort

Freigabe des VS

Belegte Kacheln freigeben

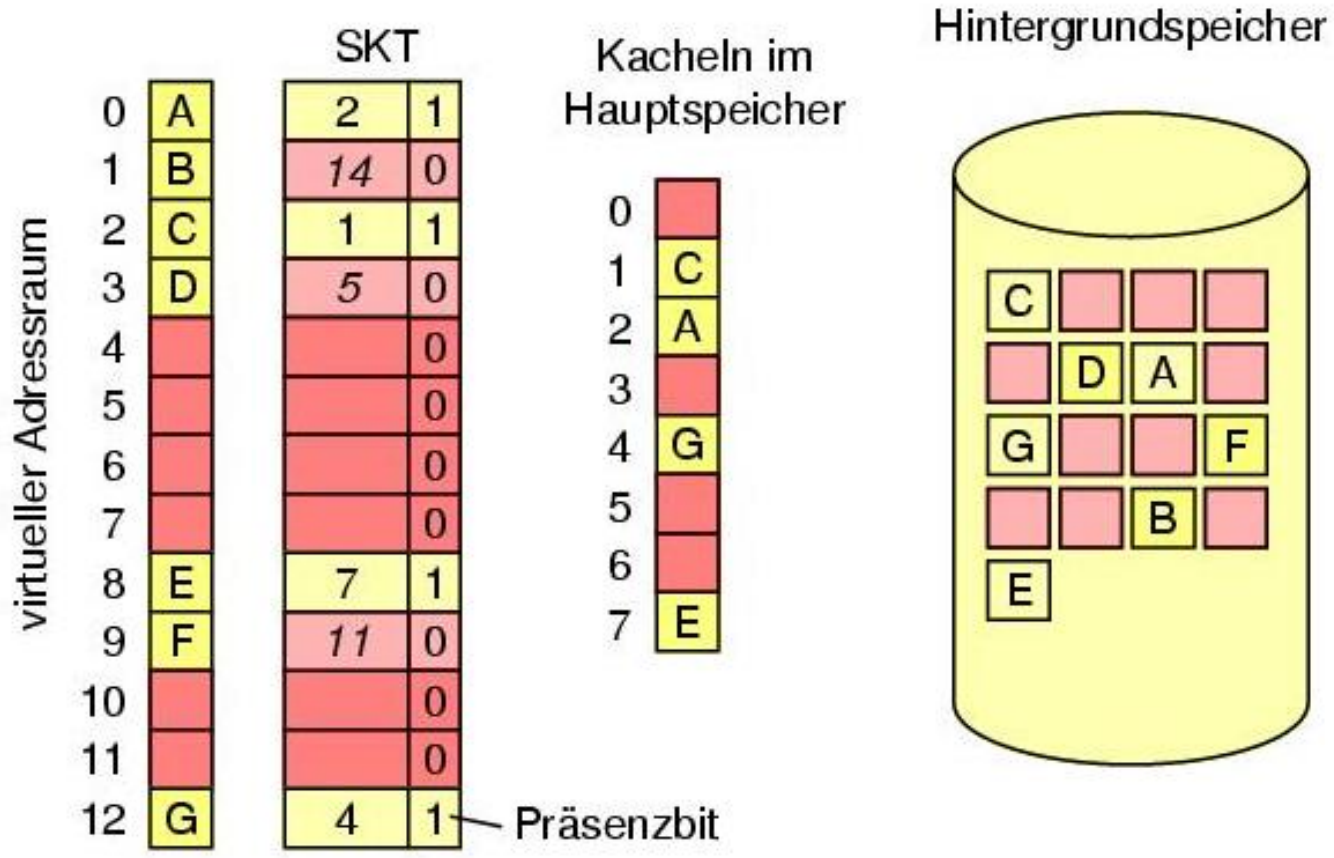
Ersatzspeicher freigeben

Zeitintensive Verarbeitungsschritte

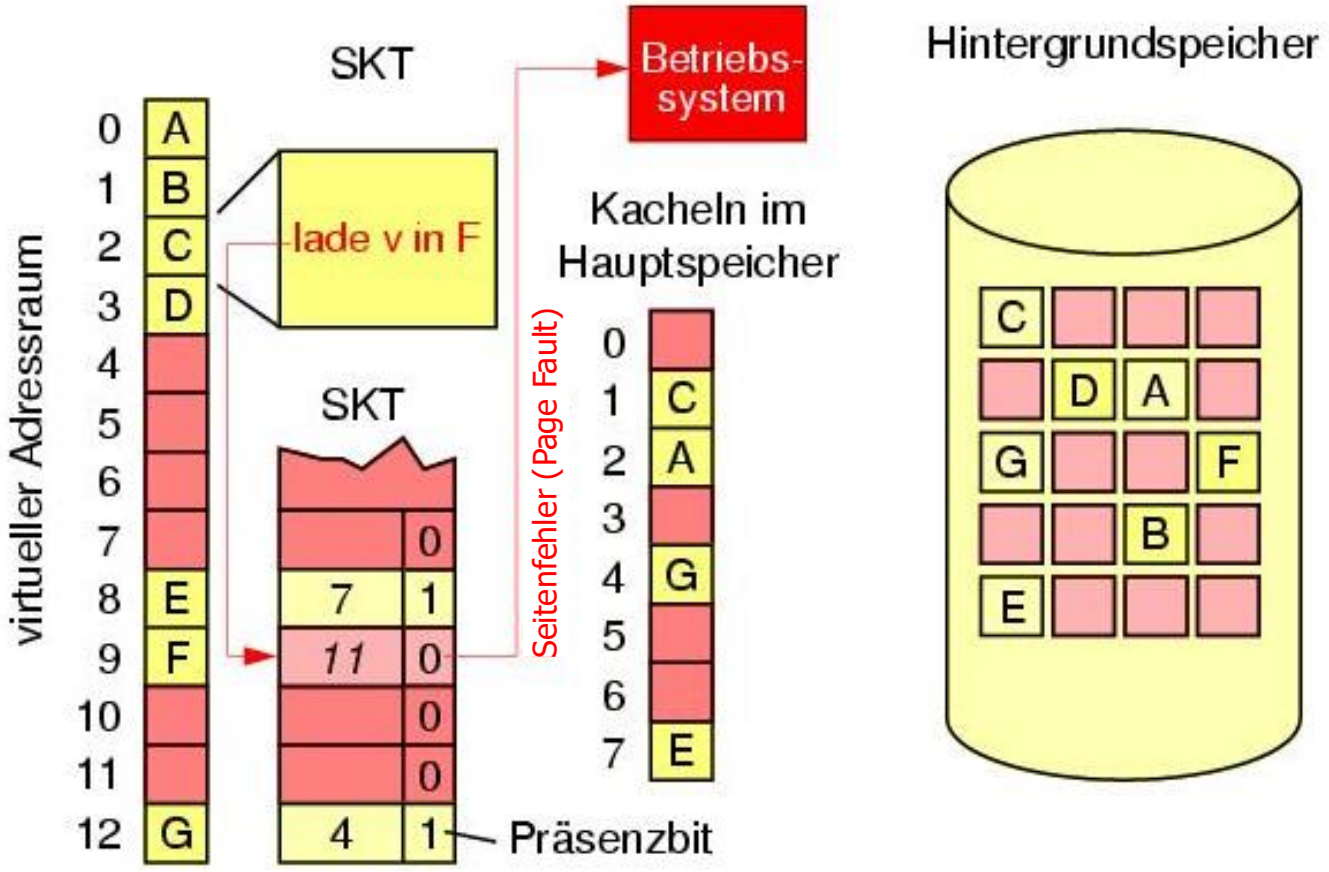


Demand Paging

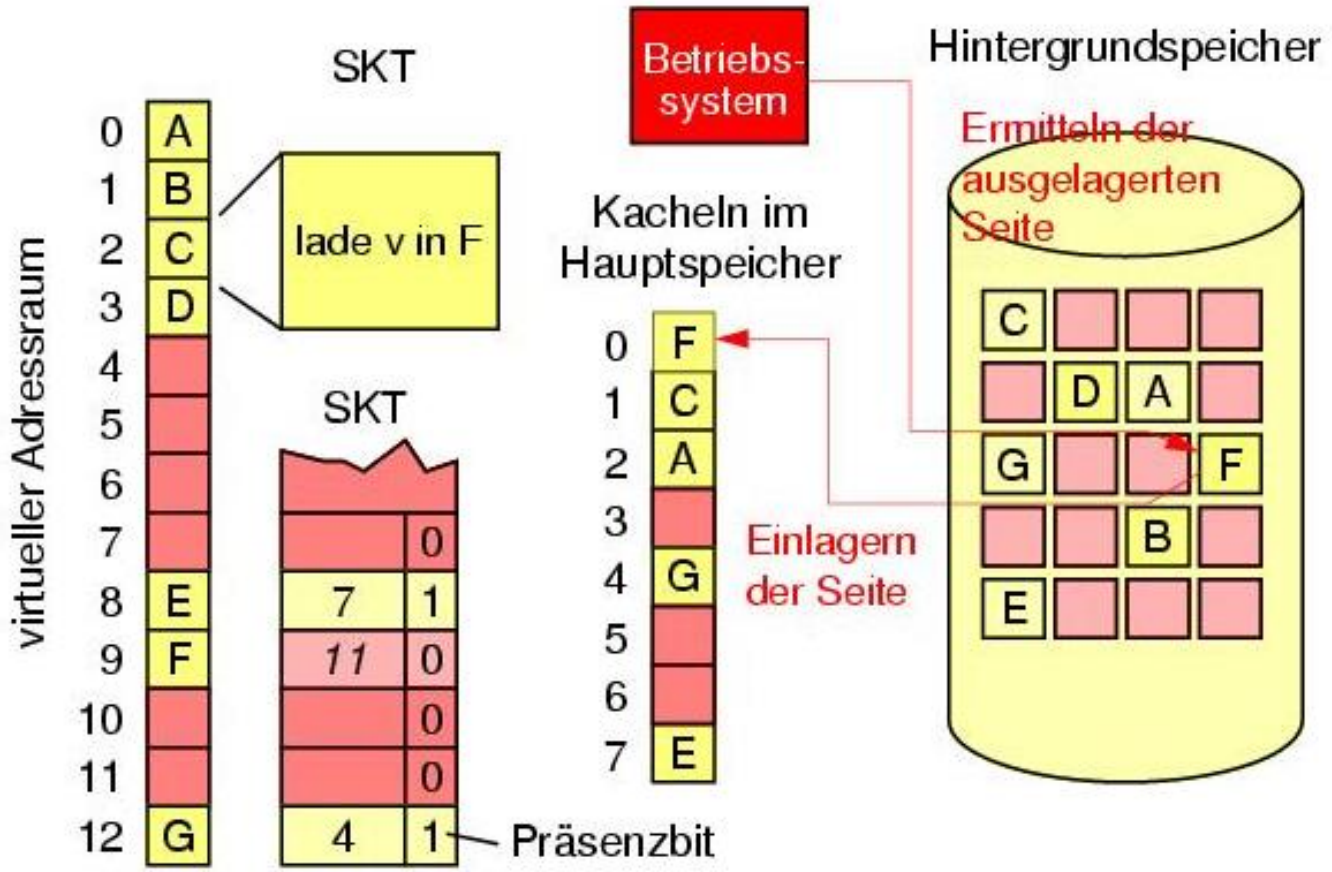
SKT = Seite-Kachel-Tabelle (anderer Name für Seitentabelle)



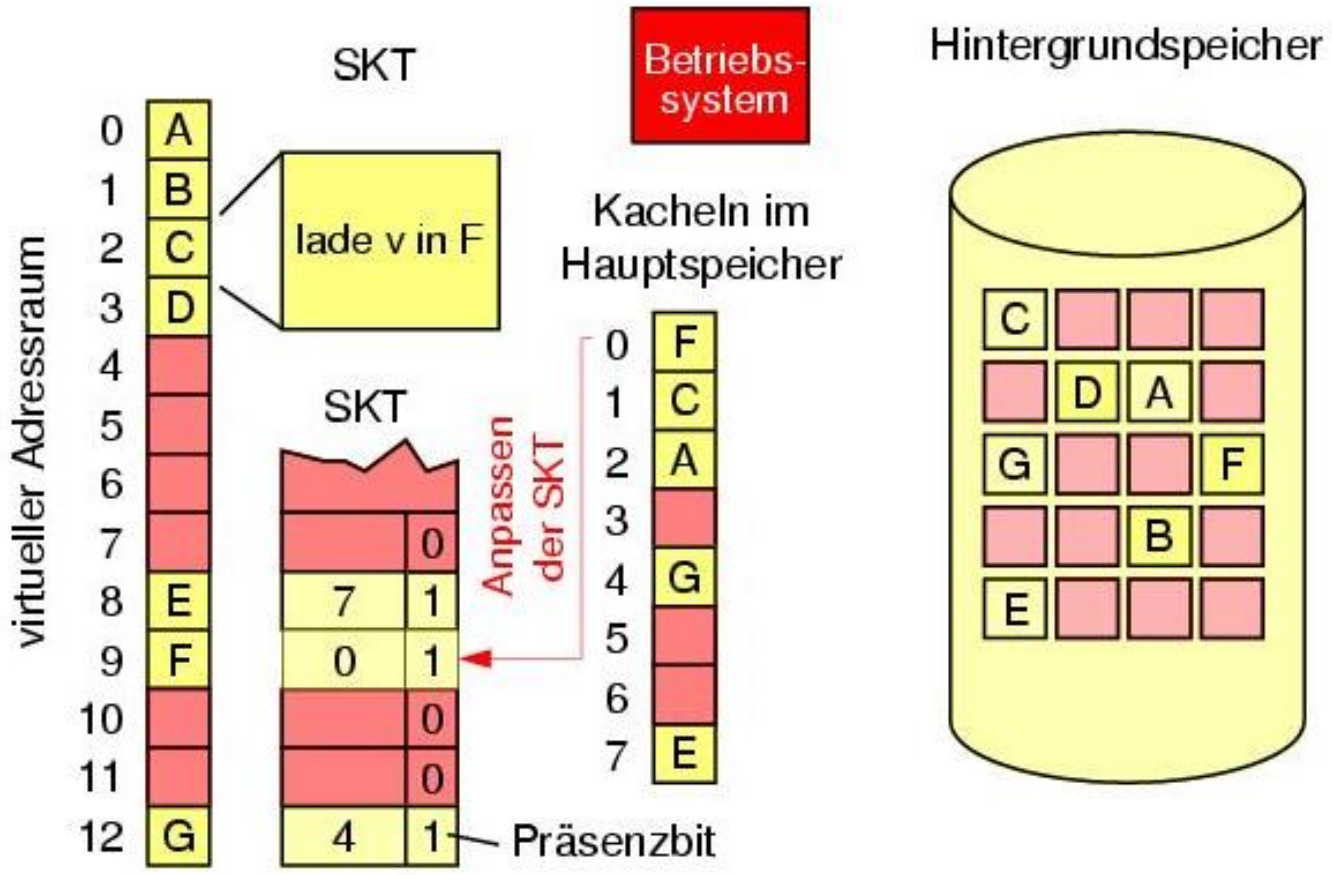
Demand Paging (2)



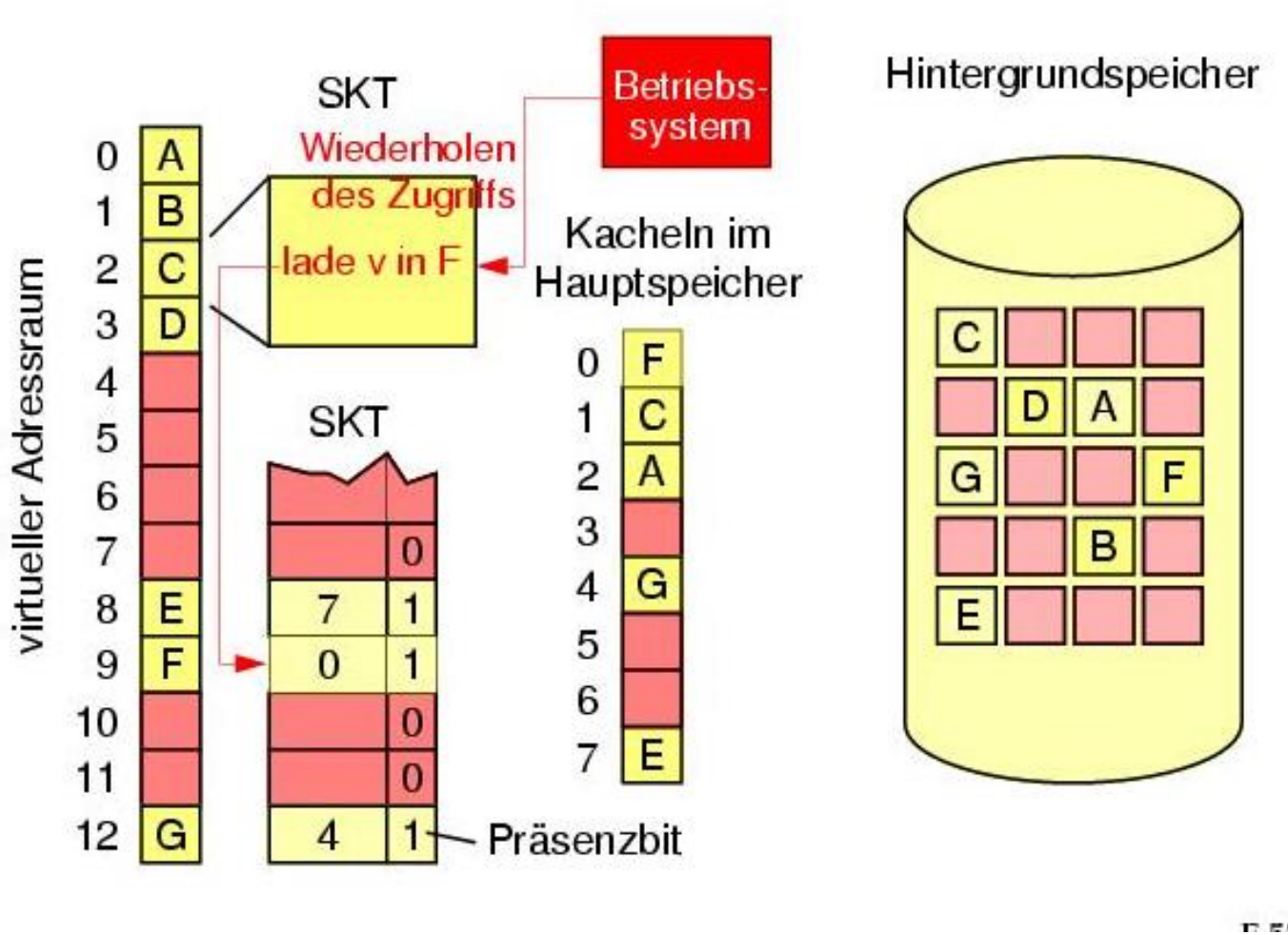
Demand Paging (3)



Demand Paging (4)



Demand Paging (5)



6.3 Verdrängungsstrategien (Replacement Policies)

- Leistungsfähigkeit von Demand Paging stark von der Anzahl der Seitenfehler abhängig

⇒ Seitenfehler durch intelligente Verdrängung minimieren

- Welche Kachel soll geleert und der Inhalt ausgelagert werden?
  - Lokale Auswahlstrategie: Es wird eine Kachel geleert, welche dem den Seitenfehler verursachenden Prozess zugeordnet ist
  - Globale Auswahlstrategie: Eine beliebige Kachel – auch von fremden Prozessen – darf geleert werden

- Modellierung der Seitenersetzung
  - Ist ri die Nummer der Seite, auf der zum Zeitpunkt t zugegriffen wird, so heißt R = r1, r2, r3,…, rn Seitenreferenzfolge

Optimale Auswahlstrategie

- Verdränge diejenige Seite, die am längsten nicht mehr benötigt werden wird ☺
  - Minimierung der Seitenfehleranzahl bei gegebener Speichergröße
  - Nicht realisierbar ohne hellseherische Fähigkeiten ⇒ wird als Messlatte für realisierbare Strategien eingesetzt, d.h. wie weit ist eine Strategie vom optimalen Ergebnis noch entfernt

Zeit	1	2	3	4	5	6	7	8	9	10	11	12
Zugriff Seite	A	C	E	D	C	D	A	B	A	D	A	C
Kachel 1	A	A	A	A	A	A	A	A	A	A	A	**C**
Kachel 2		C	C	C	C	C	C	**B**	B	B	B	B
Kachel 3			E	**D**	D	D	D	D	D	D	D	D

Ergebnis: 3 Fehler (*ohne Initialseitenfehler*)

Realisierbare Strategien

- Lokalitätsprinzip: Zugriffsverhalten in unmittelbarer Vergangenheit = gute Schätzung für das Verhalten in nächster Zukunft
  - Räumliche Lokalität: Nach Zugriff auf Adresse a ist ein Zugriff auf eine Adresse in der Nähe von a sehr wahrscheinlich
  - Zeitliche Lokalität: Nach einem Zugriff auf Adresse a ist ein erneuter Zugriff (in Kürze) auf a sehr wahrscheinlich

- Warum?
  - Sequentielle Ausführung von Anweisungen, Schleifen
  - Ausführung bestimmter Programmteile nur in Ausnahmefällen
  - 90/10-Regel: Prozesse verbringen 90% der Zeit in 10% des Adressraums

Realisierbare Strategien

- Verdrängung der Seite, die
  - Am längsten im Speicher war (First In - First Out, FIFO)
  - Am längsten nicht benutzt wurde (Least Recently Used, LRU)
  - Am wenigsten häufig benutzt wurde (Least Frequently Used, LFU)
  - Innerhalb eines Zeitraums nicht referenziert (Recently Not Used, RNU)

FIFO-Strategie

- FIFO: 4 Seitenfehler für die betrachtete Referenzfolge
- Anomalie der FIFO-Strategie
  - Bei steigender Kachelzahl kann die Anzahl von Seitenfehlern steigen(!)
  - Beispiel: 4 Kacheln ⇒ 7 Fehler, 5 Kacheln ⇒ 8 F., 6 Kacheln ⇒ 6 F.
- Erwünscht
  - Weniger Seitenfehler, wenn mehr Speicher zur Verfügung steht
  - Monoton fallende Seitenfehlerrate bei steigender Kachelrate

Zeit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Zugriff Seite	A	B	C	D	E	A	B	F	G	A	B	C	D	F	G
Kachel 1	A	A	A	A	**E**	E	E	E	**G**	G	G	G	G	G	G
Kachel 2		B	B	B	B	**A**	A	A	A	A	A	**C**	C	C	C
Kachel 3			C	C	C	C	**B**	B	B	B	B	B	**D**	D	D
Kachel 4				D	D	D	D	**F**	F	F	F	F	F	F	F

LFU-Strategie

- Verdränge Seite, die am wenigsten häufig benutzt wurde
  - Eine kaum verwendete Seite wird auch zukünftig selten benötigt
  - Ein Zähler pro Seite ⇒ Inkrementierung bei jedem Zugriff
  - Bei gleicher Frequenz: 2. Strategie, z.B. FIFO

Zeit		1	2	3	4	5	6	7	8	9	10	11	12
Zugriff Seite		A	C	E	D	C	D	A	B	A	D	A	C
Kachel 1		A	A	A	**D**	D	D	D	D	D	D	D	D
Kachel 2			C	C	C	C	C	C	**B**	B	B	B	**C**
Kachel 3				E	E	E	E	**A**	A	A	A	A	A
Zähler	A	1	1	1	(1)	(1)	(1)	2	2	3	3	4	4
	B	-	-	-	-	-	-	-	1	1	1	1	(1)
	C	-	1	1	1	2	2	2	(2)	(2)	(2)	(2)	3
	D	-	-	-	1	1	2	2	2	2	3	3	3
	E	-	-	1	1	1	1	(1)	(1)	(1)	(1)	(1)	(1)

LRU-Strategie

- Verdränge Seite, die am längsten nicht mehr referenziert wurde
  - Seitenzugriffs-Stapel: Seite, auf die zuletzt zugegriffen wurde, wird auf oberste Stapelposition gelegt ⇒ oberste k Seiten im Speicher
  - Falls Seite schon im Speicher: im Stapel wieder nach oben befördern
  - Bei Auslagerung: k-te Seite austauschen; neu referenzierte Seite kommt auf oberste Position ⇒ vorhandene Seiten rutschen nach unten, k-te Seite rutscht aus dem Stapel

Zeit		1	2	3	4	5	6	7	8	9	10	11	12
Zugriff Seite		A	C	E	D	C	D	A	B	A	D	A	C
Kachel 1		A	A	A	**D**	D	D	D	D	D	D	D	D
Kachel 2			C	C	C	C	C	C	**B**	B	B	B	**C**
Kachel 3				E	E	E	E	**A**	A	A	A	A	A
Stapel	1	A	C	E	D	C	D	A	B	A	D	A	C
	2	-	A	C	E	D	C	D	A	B	A	D	A
	3	-	-	A	C	E	E	C	D	D	B	B	D

RNU-Strategie

- Verdränge Seite, die innerhalb eines Zeitraums nicht mehr referenziert wurde
  - Definition des Zeitraums über ein Fenster, das k zuletzt referenzierte Elemente umfasst
  - Für eine Verdrängung kommen alle Seiten in Frage, die nicht innerhalb des Fensters referenziert wurden
  - Kritische Größe: Fensterbreite k>0, aber k klein
  - Beispielreferenzfolge: 4 Seitenfehler, bei k=2

Zeit	1	2	3	4	5	6	7	8	9	10	11	12
Zugriff Seite	A	C	E	D	C	D	A	B	A	D	A	C
Kachel 1	A	A	A	**D**	D	D	D	D	D	D	D	D
Kachel 2		C	C	C	C	C	C	**B**	B	B	B	**C**
Kachel 3			E	E	E	E	**A**	A	A	A	A	A

Vergleich der Strategien

- Von den Strategien zeigt LRU im Durchschnitt die beste Leistung, d.h. geringste Seitenfehlerrate
  - ⇒Häufige Anwendung in realer Systemsoftware

- Probleme bei der Realisierung
  - Alle realisierbaren Strategien erfordern bei jedem Zugriff gewisse Datenoperationen (Stapeloperationen, Zählerinkrementierung, …)
  - Durchführung dieser Operationen (vollständig in Software, mit Unterstützung von Hardware) ist zu aufwendig
  - Daher werden hauptsächlich Annäherungsverfahren realisiert

Angenäherte LRU/NRU-Strategie

- Hardwareunterstützung beim Seitenzugriff
  - Jede Seite besitzt Zugriffsbit (A) in der Seitentabelle
  - Das Zugriffsbit wird beim Lesen der Seite von der CPU gesetzt
  - Keine Information über den Zeitpunkt eines Zugriffs
- Fehlende Zeitinformation wird durch eine periodische Rücksetzung der Zugriffsbits simuliert
- Beispiel: Second-Chance-Algorithmus (Clock-Algorithmus)

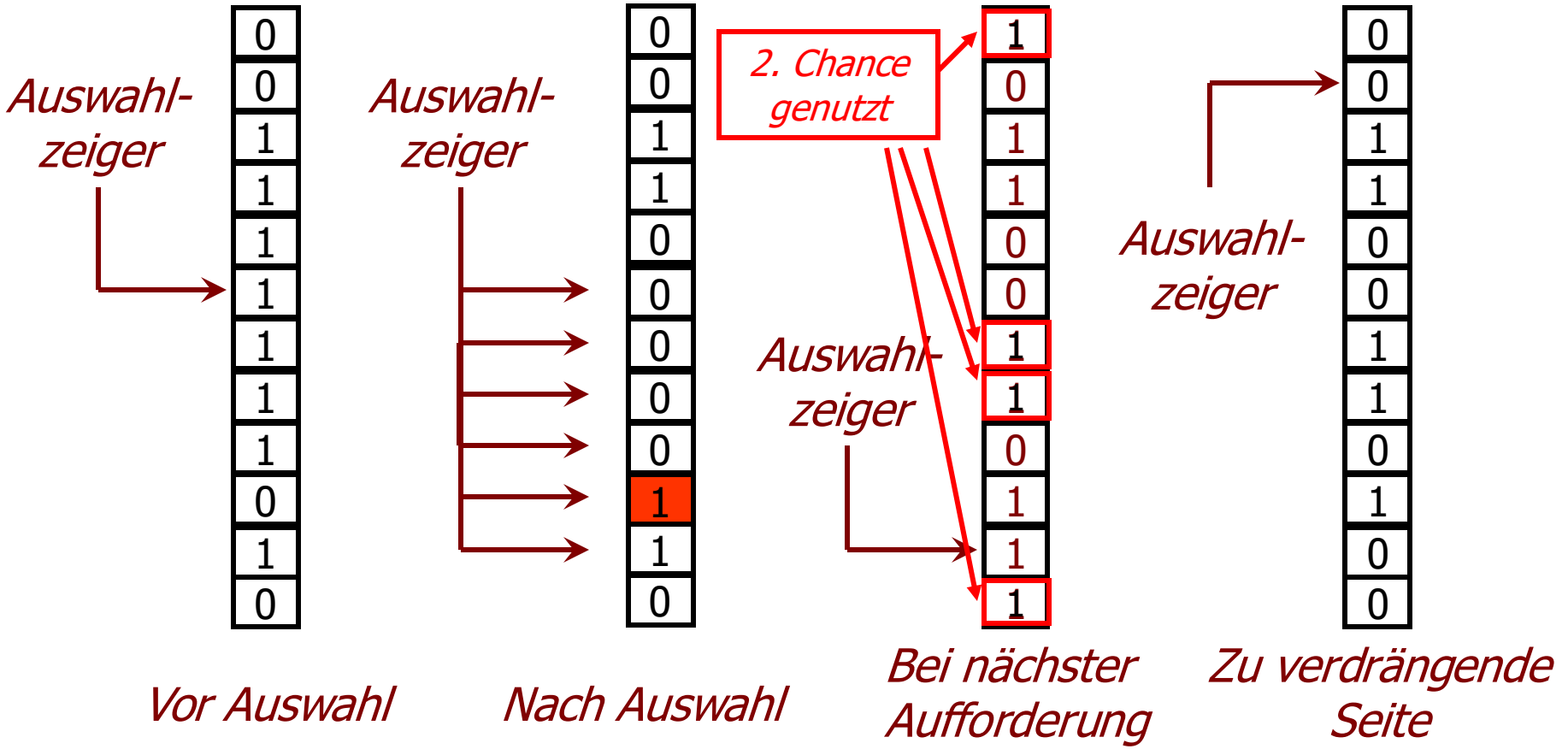
Second-Chance-Algorithmus (Clock-Algorithmus)

- FIFO-Modifikation durch Berücksichtigung von Referenzen
  - Sortiere die Seiten/Referenzbits gemäß Einlagerungsdauer
  - Durchlaufe zyklisch den Vektor mit Referenzbits
    - Falls das Referenzbit der aktuellen Seite = 1
      - Setze das Referenzbit auf 0
      - Betrachte die nächste Seite
    - Falls das Referenzbit der aktuellen Seite = 0
      - Verdränge die Seite
      - Setze die Suche mit der nächsten Seite fort

- Rücksetzung von Teilmengen von Referenzbits (Alte bis Neue Position). Alle anderen Seiten mit R-Bit 0 haben eine zweite Chance, referenziert zu werden
- Clock-Algorithmus ist eine Implementierungsvariante

Second-Chance-Algorithmus

- Beispielsituation (Bitfolgen = Referenzindikatoren)



Solaris 2: Algorithmus

- Speicherverwaltung mit dem Prozess pageout
  - Variante des Clockalgorithmus (Zweizeigeralgorithmus, two-handed-clock)
  - Erster Zeiger scannt die Seiten und setzt die Referenzbits auf 0
  - Später läuft der zweite Zeiger nach und gibt alle Seiten mit Referenzbit immer noch 0 wieder frei
- Wichtige Parameter
  - Scanrate: Scangeschwindigkeit (Seiten/Sekunde)
    - Anpassung an aktuellen Systemzustand (slowscan = 100 Seiten/s bis fastscan = Gesamtanzahl von Seiten/2 aber max = 8192)
  - Handspread: Statischer Abstand in Seiten zwischen den Zeigern
    - Tatsächlicher Abstand ergibt sich durch Kombination von scanrate und handspread, z.B. scanrate = 100 und handspread = 1024 $\Rightarrow$ 10s zwischen den beiden Zeigern
    - Bei höher Belastung Abstände von einigen Tausendstel nicht ungewöhnlich
- Erweiterung: Überspringe Seiten von Shared Libraries

Paging-Daemon

- Technik zur Sicherung eines ausreichenden Vorrats an freien Kacheln für schnelle Reaktion auf weitere Speicheranforderungen
- Grundidee
  - Trennung von Seitenverdrängung und Seiteneinlagerung
  - Speicherverwaltung hält eine vorab festgelegte Anzahl an Kacheln frei
  - $\Rightarrow$ Seitenfehler können ohne Verdrängung beseitigt und neue Adressräume direkt angelegt werden
- Realisierung
  - Paging-Daemon wird periodisch aktiviert (Üblicher Wert: alle 250 ms)
  - Überprüfung, ob die a-priori vorgegebene Anzahl freier Kachel unterschritten wurde (Üblicher Wert: ca. 25% der Kachel frei)
    - Falls ja $\Rightarrow$ Auslagerung von Seiten auf die Festplatte gemäß der eingesetzten Verdrängungsstrategie
    - Falls nein $\Rightarrow$ Blockierung des Paging-Daemons bis zum nächsten Sollzeitpunkt

Realisierung eines Paging Daemons in Solaris 2

- Wichtige Parameter
  - Lotsfree: mindestens 1/4 des Speichers frei, Überprüfung alle 0.25s
  - Desfree: Fällt die Anzahl freier Seite unter diesen Schwellwert (Durchschnitt über 30s) ⇒ Start Swapping, Überprüfung alle 0.1s
  - Minfree: Nur noch minimale Menge freier Seiten vorhanden ⇒ Aufruf der Seitenersetzung bei jeder Speicheranforderung

Page-Fault-Frequency-Modell

- Seitenfehlerrate = Anzahl Seitenfehler / Zeiteinheit
- Für jeden Prozess wird die Seitenfehlerrate r gemessen
  - Einführung von zwei Schwellwerten r1 (obere Intervallgrenze) und r2 (untere Intervallgrenze)
  - Einstellung der Kachelanzahl S in Abhängigkeit von Seitenfehlerrate
    - Verringere Anzahl Kacheln, falls r < r1, d.h. S =S -1
    - Erhöhe Anzahl Kacheln, falls r > r2, d.h. S = S + 1