Abbildung virtuelle Adresse → reale Adresse

Virtuelle Adresse = Virtuelle Seitennummer * Seitengröße + Offset

1. Ermittlung der virtuellen Seitennummer:
   Virtuelle Seitennummer = floor (virtuelle Adresse / Seitengröße)
2. Ermittlung der Seitenrahmennummer durch Verwendung einer Seitentabelle (Index = virtuelle Seitennummer)
3. Ermittlung der realen Adresse (im Hauptspeicher):
   Reale Adresse = Seitenrahmennummer * Seitengröße + Offset
   (Offset = Virtuelle Adresse modulo Seitengröße
   = *„Positionsnummer“ innerhalb der Seite*)

Berechnungstrick:

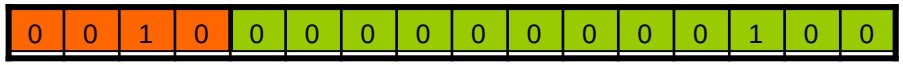
- Verwendung von Zweierpotenzen für Seitengrößen
- Bitweise Aufteilung der virtuellen Adresse für die Adressberechnung, z.B. bei 32 Bit – Virtueller Adresse und Seitengröße 4K = 2^{12} :
  20 Bit für Seitennummer + 12 Bit für Offset
- Statt Division / Addition kann bitweises Trennen / Aneinanderhängen verwendet werden!

Seitentabellen („Page Tables“)

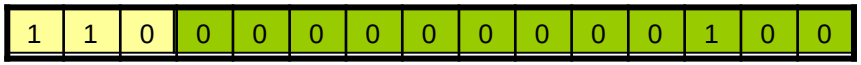
- **Eine** Seitentabelle pro Prozess
- Ein Eintrag beschreibt eine Seite (Index = virtuelle Seitennummer)
- Aufbau eines Eintrags in einer Seitentabelle:
  - Seitenrahmennummer
  - Valid-Bit: 1 = im Hauptspeicher, 0 = auf Platte (→ *Zugriff löst Seitenfehler aus → Interrupt zum Einlesen von der Platte*) *[Present-/Absent-Bit]*
  - Zugriffsrechte (Schreiben/Lesen)
  - Verwaltungs-Flags: Referenced-Bit (R-Bit), Modified-Bit (M-Bit) werden bei Lese-Schreib-Zugriffen (R-Bit) oder Schreib-Zugriffen (M-Bit) auf die Seite von der Hardware gesetzt

Beispiel: 64K (=2^{16}) virtueller Adressraum, 32K (=2^{15}) Hauptspeichergröße, Seitengröße 4K (=2^{12})

CPU: Virtuelle Adresse 8196 = 2 * 2^{12} + 4



MMU: Reale Adresse 24580 = 6 * 2^{12} + 4



4-Bit Virtuelle Seitennummer (VSN): **2**

VSN (Index)	SRN	valid
0	010	1
1	001	1
2	110	1
3	000	1
4	100	1
5	011	1
6	000	0
7	000	0
8	000	0
9	101	1
10	000	0
11	111	1
12	000	0
13	000	0
14	000	0
15	000	0

Seitenrahmen-nummer (SRN): **6** aus Seitentabelle

12-bit Offset: **4** (Seitengröße 2^{12}) wird direkt kopiert

Seitentabelle für PID X: 16 (=2^4) virtuelle Seiten

8 (=2^3) physikalische Seitenrahmen im Hauptspeicher vorhanden

Zahlen aus der Praxis

- Größen bei 32-Bit-Systemen:
  - Virtuelle Adresse: 32 Bit
    → max. 2^{32} = 4 GB Virtueller Speicher adressierbar
  - Übliche Seitengröße: 2^{12} Byte (4 KB)
  - max. Anzahl Seiten (max. Größe einer Seitentabelle)
    bei 4 KB Seitengröße: 2^{20} Einträge (~ 1 Million)
- Größen bei 64-Bit-Systemen:
  - Virtuelle Adresse: 64 Bit
    → max. 2^{64} = 18 ExaByte ($1{,}8 * 10^{19}$ Byte) Virtueller Speicher adressierbar
  - Übliche Seitengrößen: 2^{12} Byte, 2^{21} Byte (2 MB), 2^{30} Byte (1 GB)
  - max. Anzahl Seiten (max. Größe *einer* Seitentabelle)
    bei 4 KB Seitengröße: 2^{52} Einträge
    bei 2 MB Seitengröße: 2^{43} Einträge
    bei 1 GB Seitengröße: 2^{34} Einträge (~ 16 Milliarden)

Aufgaben Abschnitt 4.2.b): Paging

Quelle: Tanenbaum, Moderne Betriebssysteme, 3. Auflage, Kapitel 3, Aufgabe 5

5. Berechnen Sie die virtuelle Seitennummer und den Offset für die folgenden dezimalen virtuellen Adressen, falls die Seitengröße 4 KB bzw. 8 KB ist: 20.000, 32.768, 60.000.

Quelle: Tanenbaum, Moderne Betriebssysteme, 3. Auflage, Kapitel 3, Aufgabe 9

9. Ein Rechner hat einen 32-Bit-Adressraum und 8-KB-Seiten. Die Seitentabelle liegt komplett in der Hardware, mit einem 32-Bit-Wort pro Eintrag. Wenn ein Prozess startet, wird die Seitentabelle aus dem Arbeitsspeicher in die Hardware kopiert, was für jedes Wort 100 ns dauert. Wie groß ist der Anteil der Zeit, in der die CPU Seitentabellen lädt, wenn ein Prozess immer für 100 ms läuft (einschließlich der Zeit, in der die Seitentabelle geladen wird)?

Implementierung von Seitentabellen

- **Problem:** Größe von Seitentabellen
  (eine Seitentabelle pro Prozess!)
  Aber notwendig:
  - effizienter Zugriff (-> schnell)
  - effiziente Speicherung (-> klein)
- **Lösungsmöglichkeiten (kombinierbar):**
  1. (Seitentabellen komplett im Hauptspeicher)
  2. Caching von Tabelleneinträgen in Registern/TLB
  3. Aufteilung in mehrere Tabellen mit mehrstufigem Zugriff
  4. Invertierte Seitentabellen

2. Caching von Tabelleneinträgen in Registern/TLB

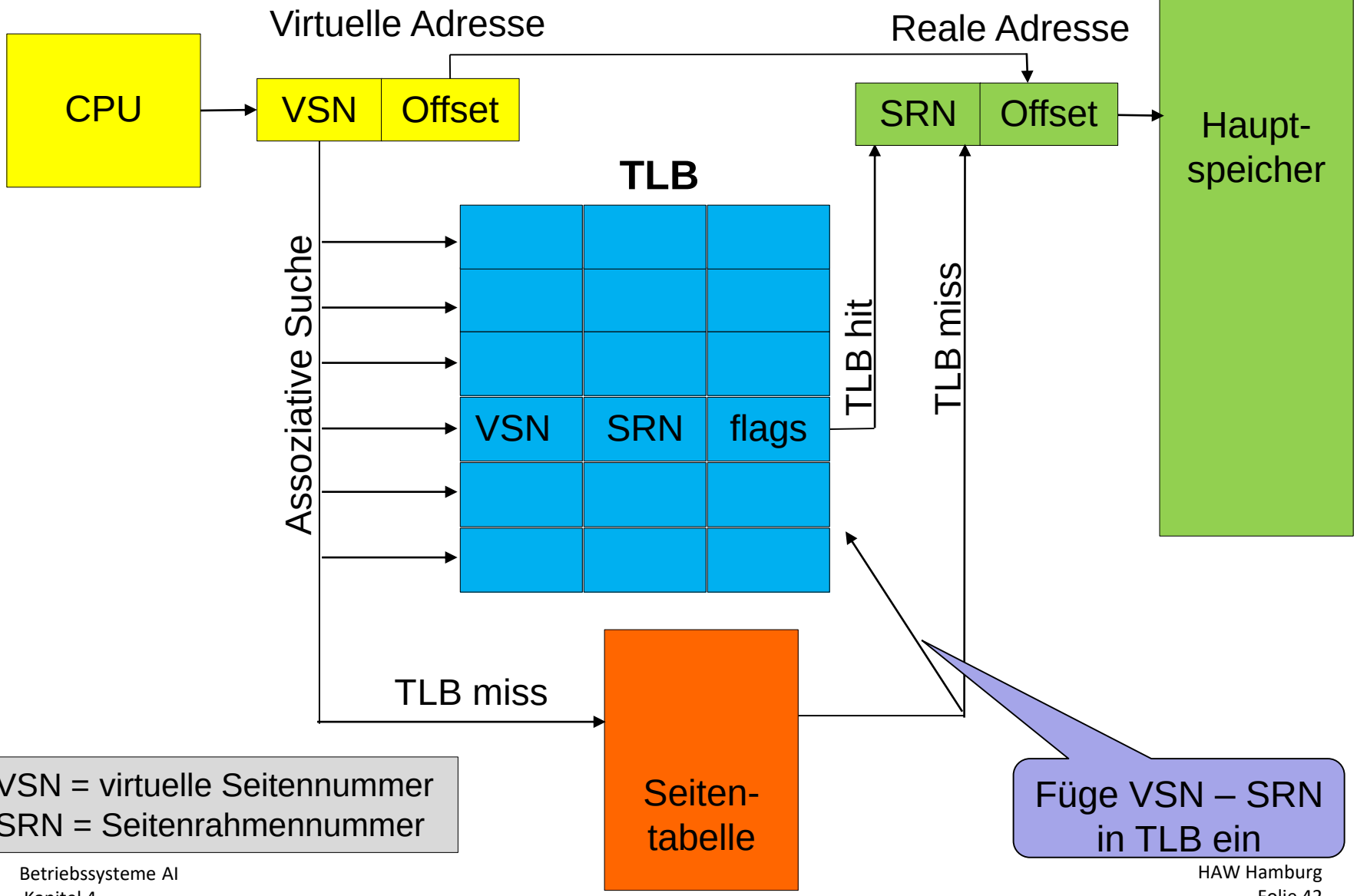
- Einführung eines schnellen **Cache** in der MMU für die Seitentabellen-Einträge
- Wird **Translation Lookaside Buffer (TLB)** genannt
- Enthält kürzlich benutzte Einträge
- Größe: 64 bis 1024 Einträge
- Ist ein Assoziativ-Speicher: **Parallele Suche nach VSN** mittels Hardware – Unterstützung

- „kürzlich benutzt“: Trefferquoten in der Praxis: 80% - 98%

Translation Lookaside Buffer (TLB): Zugriffsvorgang

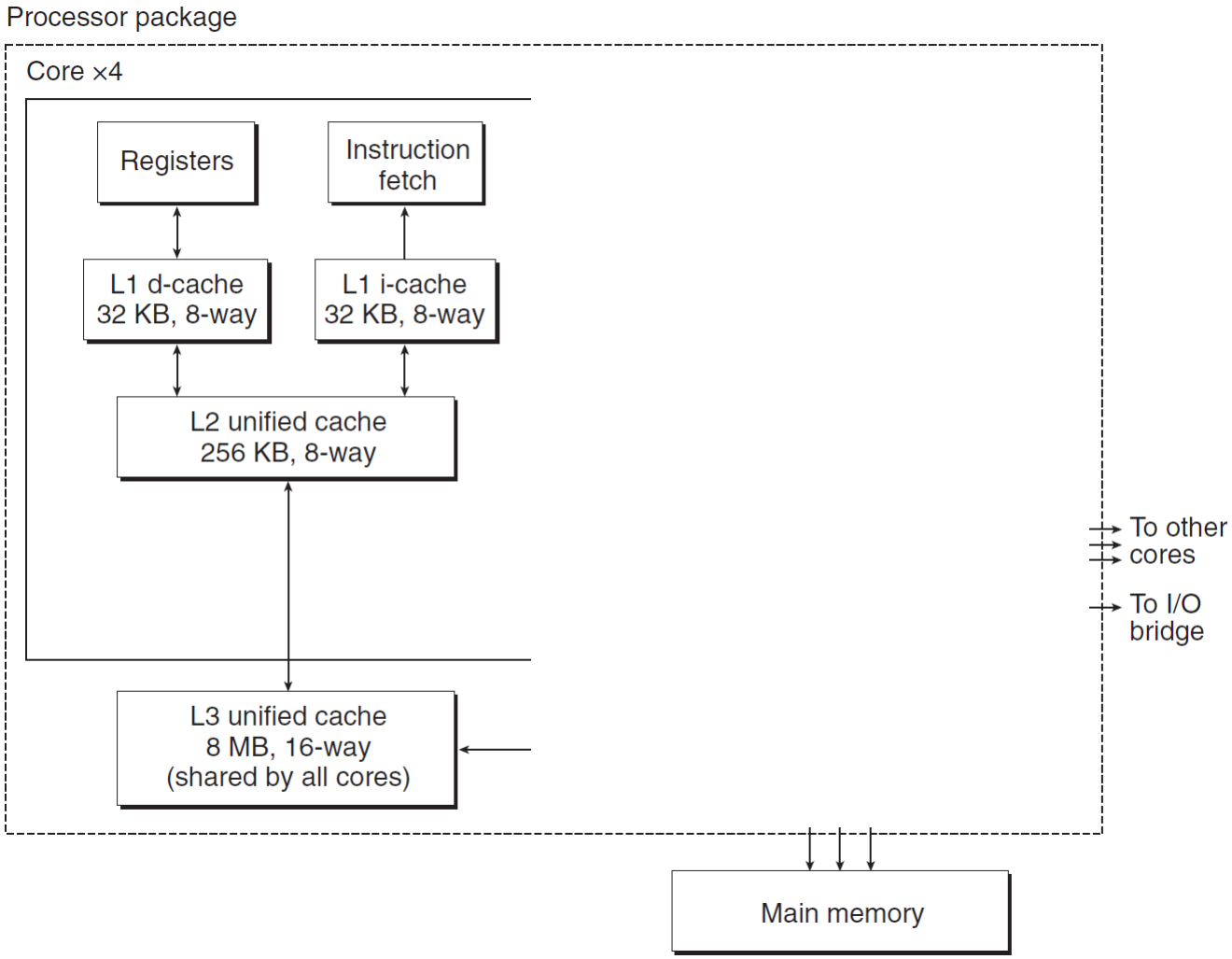
- Virtuelle Adresse → per Hardware (MMU): **Suche im TLB** nach virtueller Seitennummer (VSN)
- **Vorhanden:** → direkte Umsetzung in die Seitenrahmennr. (SRN)
- **Nicht vorhanden:** → Hardware durchsucht **Seitentabelle**
  - Eintrag existiert & Seite ist im Hauptspeicher
    - **Aktualisierung des TLB** (durch Verdrängung eines alten Eintrags)
  - Sonst: Seitenfehler („Page Fault Interrupt“)
    - Seite laden & **Seitentabelle aktualisieren**

Ermittlung einer realen Adresse mittels TLB und Seitentabelle durch die MMU



VSN = virtuelle Seitennummer
SRN = Seitenrahmennummer

Details gefällig?
Beispiel Intel Core i7 Memory System

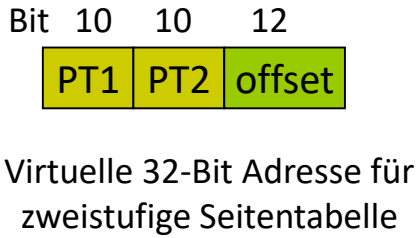


Quelle: Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, 2nd Edition 2011, Pearson.

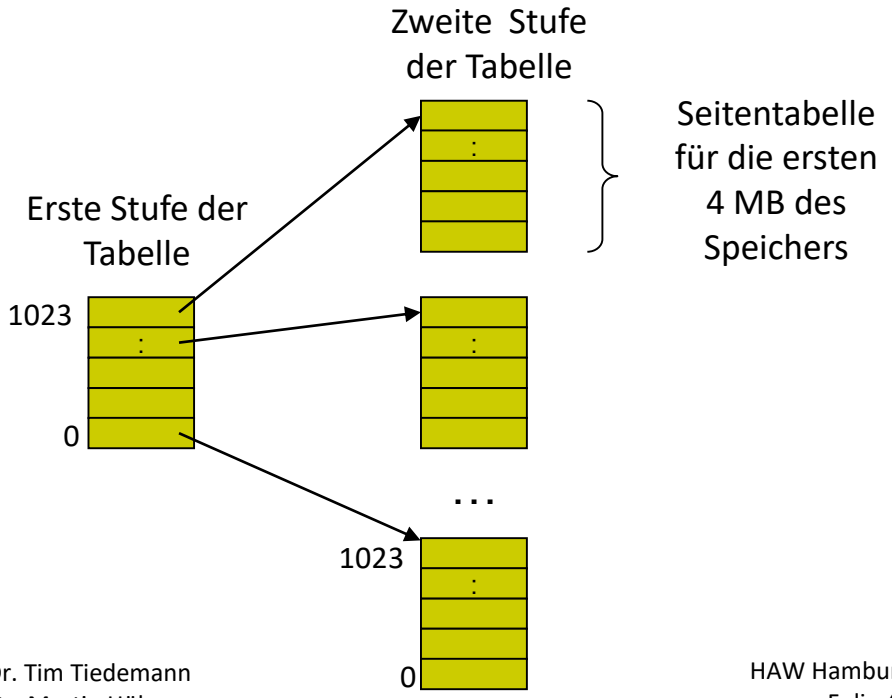
3. Aufteilung einer Seitentabelle in mehrere Tabellen mit mehrstufigem Zugriff

Nur wenige Tabellen müssen zeitgleich im Speicher geladen sein!

- Beispiel: 32-Bit Adresse zusammengesetzt aus 2x10-Bit Adressen für die Seitennummern und 12-Bit Offset: zweistufige Seitentabelle
- PT1-Feld ist Index für oberste Seitentabelle
- PT2-Feld ist Index für ausgewählte Seitentabelle der 2ten Stufe



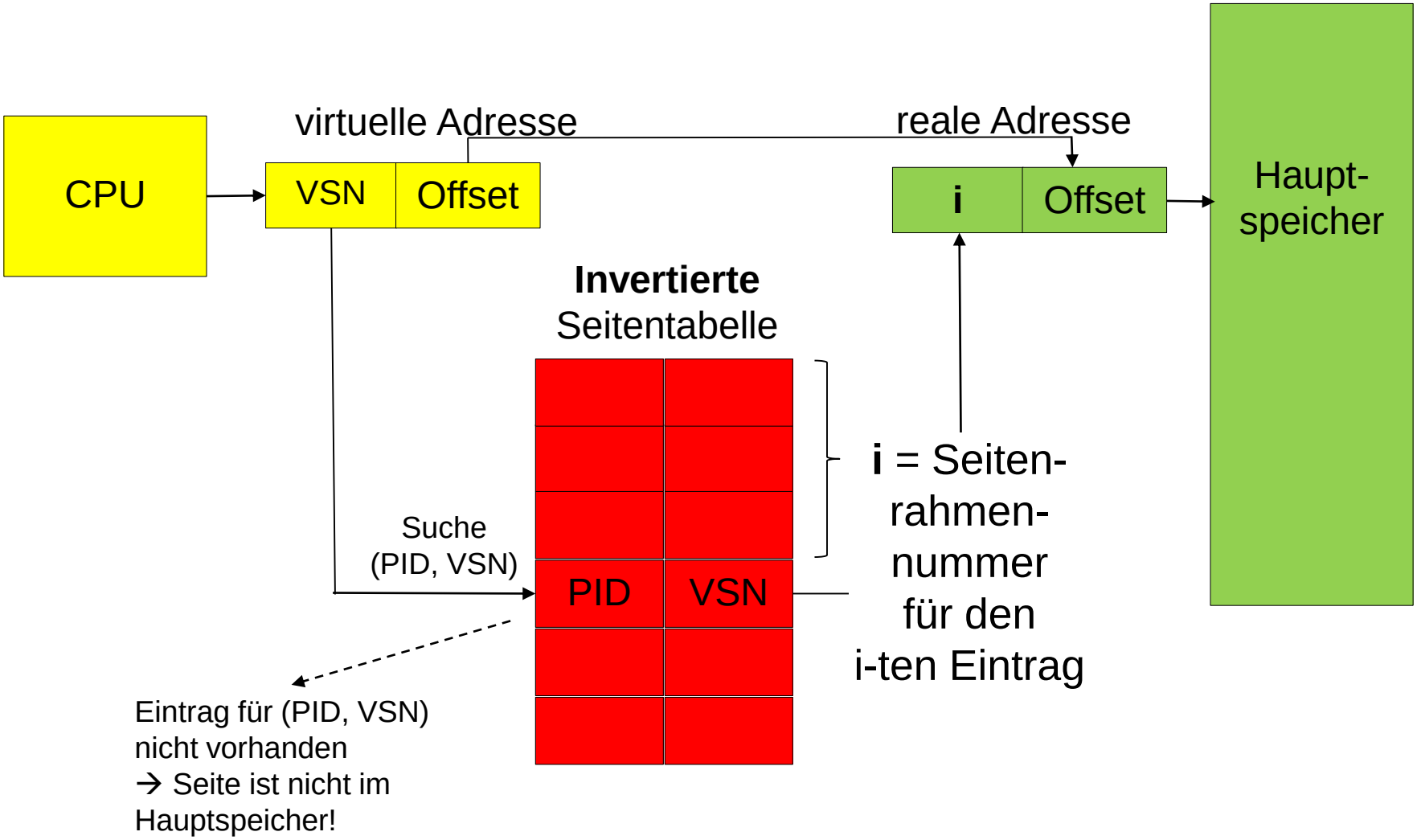
Virtuelle 32-Bit Adresse für zweistufige Seitentabelle



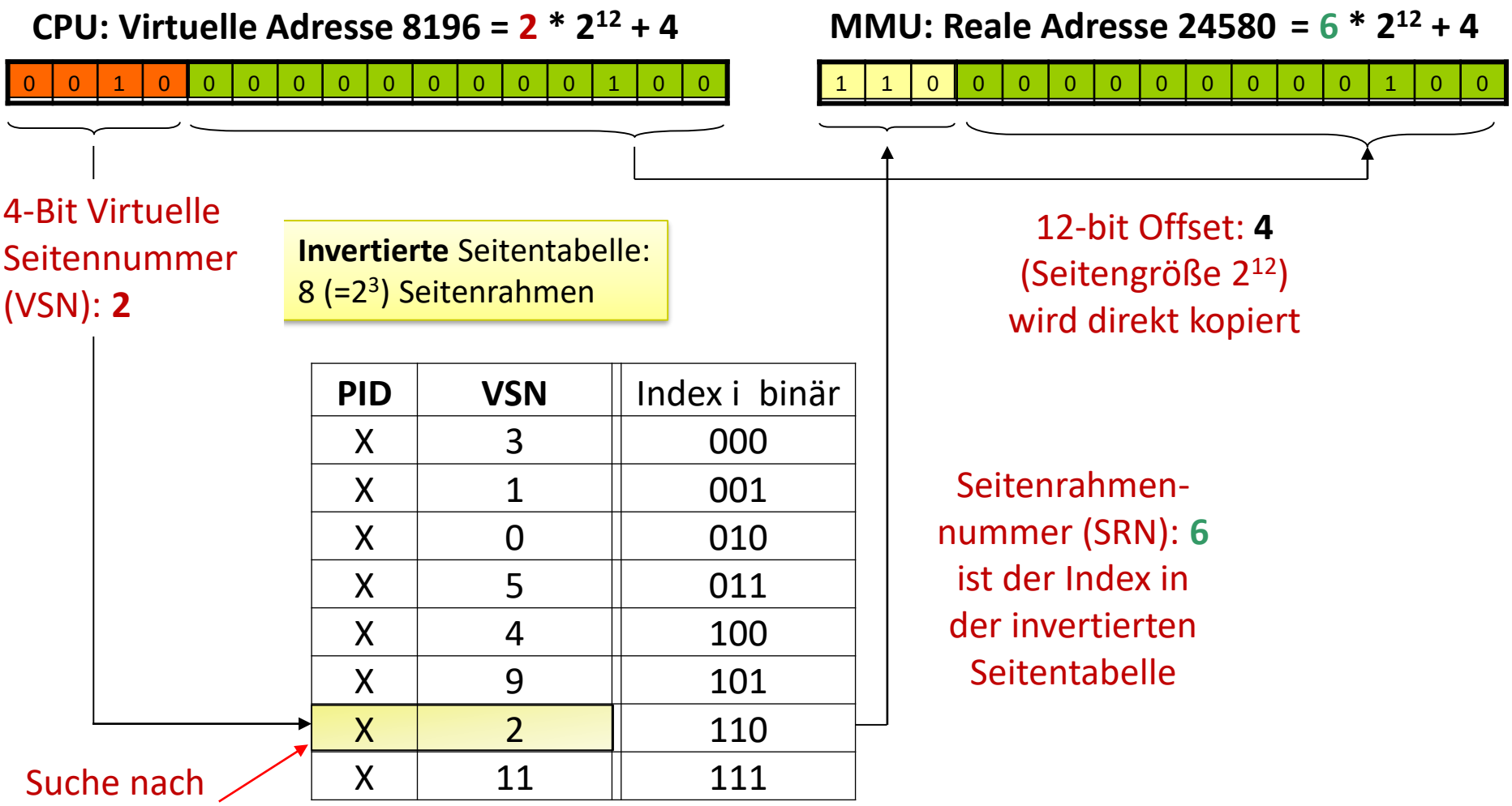
4. Invertierte Seitentabellen

- Idee: Zusätzlich zum TLB **eine** Tabelle für **alle** physischen Seitenrahmen des Hauptspeichers verwalten!
- Eigenschaften:
  - Für jeden Seitenrahmen im Hauptspeicher genau ein Eintrag
    - Prozess (**PID**) **[„Besitzer“ des Seitenrahmens]**
    - Virtuelle Seitennummer (**VSN**)
  - Unter Tabellenindex **i** steht der Eintrag für den Seitenrahmen mit der Seitenrahmennummer **i**
- Adressumsetzung: Suche (PID, VSN) in der invertierten Seitentabelle!
  - **Vorhanden:** Bilde reale Adresse (**i * Seitengröße + Offset**)
  - **Nicht vorhanden:** „normale“ Seitentabelle → Seitenfehler
- **Problem:** Suche in invertierter Seitentabelle aufwändig
  → Hash-Tabelle für virtuelle Seitennummern einsetzen!

Adressumsetzung mit invertierter Seitentabelle



Beispiel mit Invertierter Seitentabelle



PID	VSN	Index i binär
X	3	000
X	1	001
X	0	010
X	5	011
X	4	100
X	9	101
X	2	110
X	11	111

Aufgaben Abschnitt 4.2.b): Paging

Quelle: Tanenbaum, Moderne Betriebssysteme, 3. Auflage, Kapitel 3, Aufgabe 10

10. Angenommen, eine Maschine hat virtuelle 48-Bit-Adressen und physische 32-Bit-Adressen.

a. Wenn eine Seite 4 KB groß ist, wie viele Einträge sind in der Seitentabelle, wenn sie nur eine Ebene hat? Warum?

b. Nehmen Sie nun an, dass dasselbe System ein TLB mit 32 Einträgen hat. Außerdem soll es ein Programm geben, dessen Befehle in eine Seite passen und das lange Festkommazahlen sequenziell aus einem Feld einliest, welches sich über Tausende von Seiten erstreckt. Wie effektiv ist der TLB in diesem Fall?

Anm.: Gehen Sie davon aus, dass die langen Festkommazahlen 32 Bit lang sind.

Quelle: Tanenbaum, Moderne Betriebssysteme, 3. Auflage, Kapitel 3, Aufgabe 18

18. Eine Maschine hat virtuelle 48-Bit-Adressen und physische 32-Bit-Adressen. Wie viele 8-KB-Seiten sind in der Seitentabelle?

Aufgaben Abschnitt 4.2.b): Paging

Quelle: Tanenbaum, Moderne Betriebssysteme, 3. Auflage, Kapitel 3, Aufgabe 13

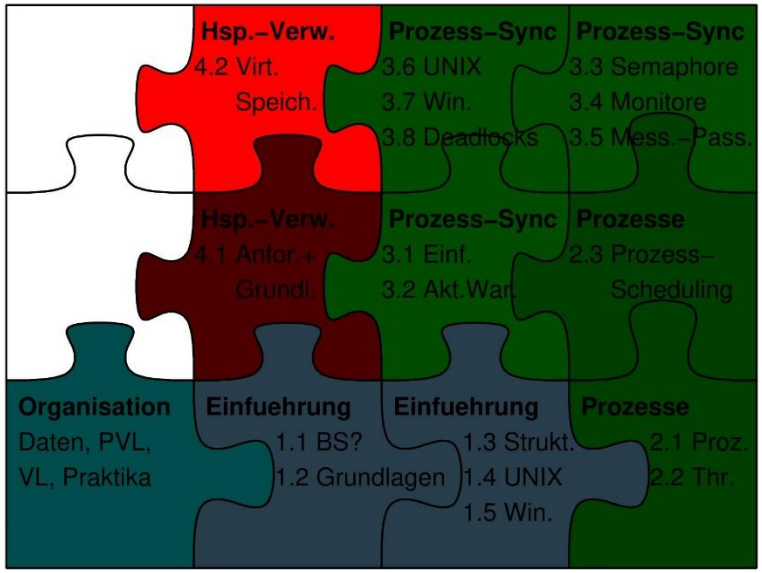
13. Eine virtuelle 32-Bit-Adresse wird in vier Teile a, b, c und d aufgeteilt. Die ersten drei Felder sind die Indizes für eine dreistufige Seitentabelle. Das vierte Feld, d, ist der Offset. Hängt die Anzahl der Seiten von der Länge aller vier Felder ab? Wenn nein, welche Felder sind unwichtig?

Quelle: Tanenbaum, Moderne Betriebssysteme, 3. Auflage, Kapitel 3, Aufgabe 15

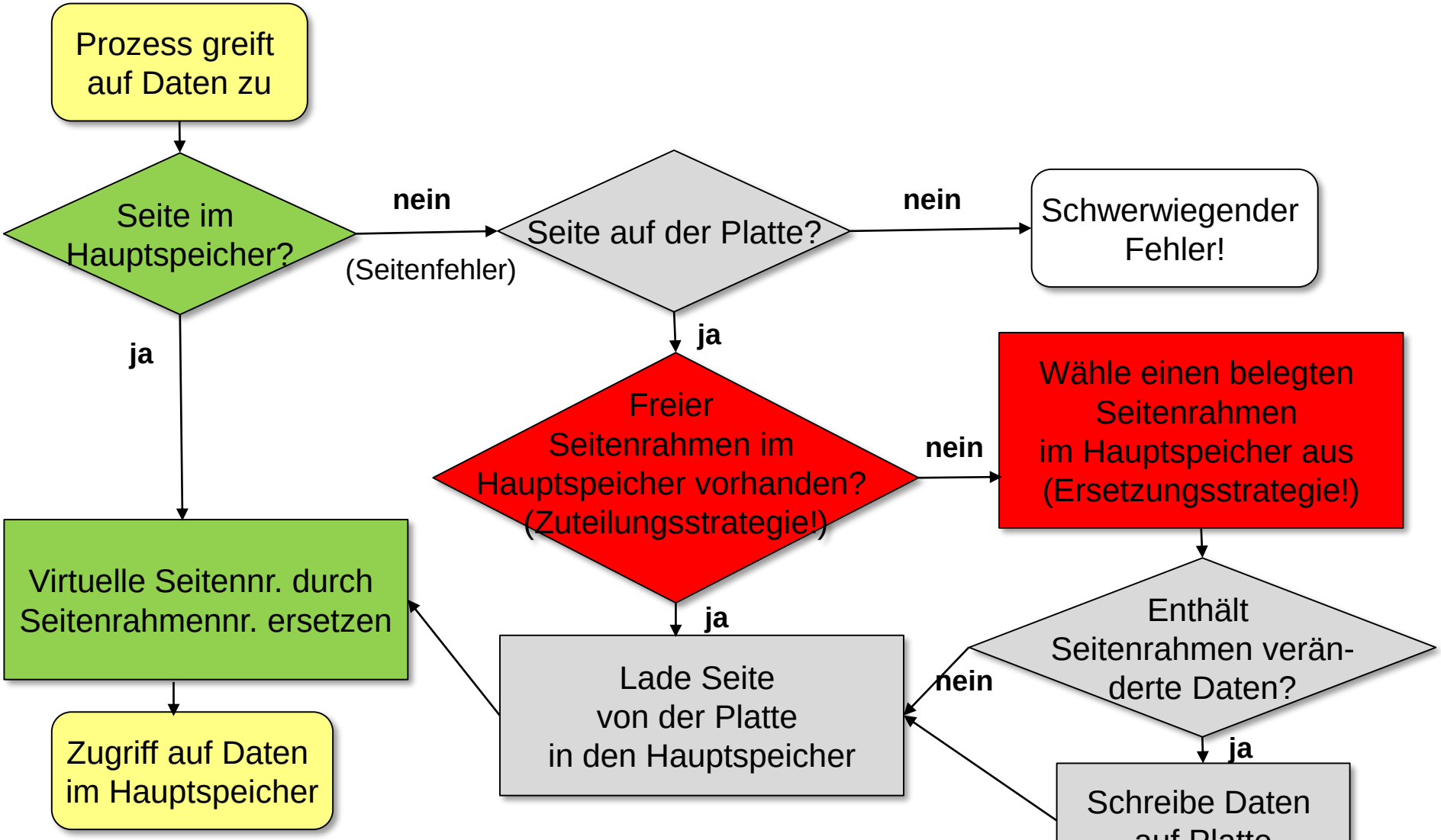
15. Ein Computer, dessen Prozesse 1.024 Seiten in ihrem Adressraum haben, hält seine Seitentabellen im Speicher. Der Aufwand, ein Wort aus einer Seitentabelle zu lesen, ist 5 ns. Um ihn zu verringern, hat der Computer einen TLB, der 32 Paare (virtuelle Seite, physischer Seitenrahmen) enthält und in 1 ns durchsucht werden kann. Welche Trefferrate ist nötig, um den durchschnittlichen Aufwand auf 2 ns zu reduzieren?

Kapitel 4
Hauptspeicher-Verwaltung

1. Anforderungen und Grundlagen
2. Virtueller Speicher
   a) Einführung und Prinzipien
   b) Paging
   c) Pagingstrategien
   d) Unix / Windows
   e) Meltdown / Spectre



Übersicht Seitentransferprozess (vereinfacht)



Pagingstrategien

- **Ersetzungsstrategie**
  - Welche Seiten werden ersetzt (verdrängt), wenn Hauptspeicherplatz benötigt wird?
- **Speicherzuteilungsstrategie**
  - Wieviele Hauptspeicherseiten bekommt ein Prozess zugeteilt?

Ersetzungsstrategien

- Auftreten eines Seitenfehlers („Page fault“):
  - Laden einer neuen Seite von der Platte
    → Verdrängung einer „alten“ Seite im Hauptspeicher
  - Was passiert mit einer alten Seite?
    - Verändert (M-Bit): Zurückschreiben auf die Platte
    - Nicht verändert: Sofort Überschreiben mit neuer Seite
- **Problem**: Auswahl der „alten“ Seite im Hauptspeicher, die durch die neue Seite ersetzt wird!
- **Optimum**: Die Seite ersetzen, die am längsten in der nächsten Zeit nicht mehr benutzt werden wird

➔ Die optimale Strategie kennt die Zukunft!

- Vorhersage über die genaue Benutzung der Seiten in der Zukunft ist fast immer unmöglich
- Die guten Strategien treffen daher aus dem Verhalten in der jüngsten Vergangenheit Entscheidungen für die nahe Zukunft

Ersetzungsstrategie 1/3: First-In, First-out (FIFO)

- Es wird immer die Seite ersetzt, die am längsten im Hauptspeicher ist
- Einfachstes Verfahren
- Verwaltet die Hauptspeicher-Seiten eines Prozesses in einer Liste (Kopf: älteste Seite, Ende: jüngste Seite)
- Berücksichtigt nicht, ob die Seite gerade jetzt häufig benutzt wird
  - Könnte sofort nach Ersetzung wieder gebraucht werden
- Ineffizientes Verhalten

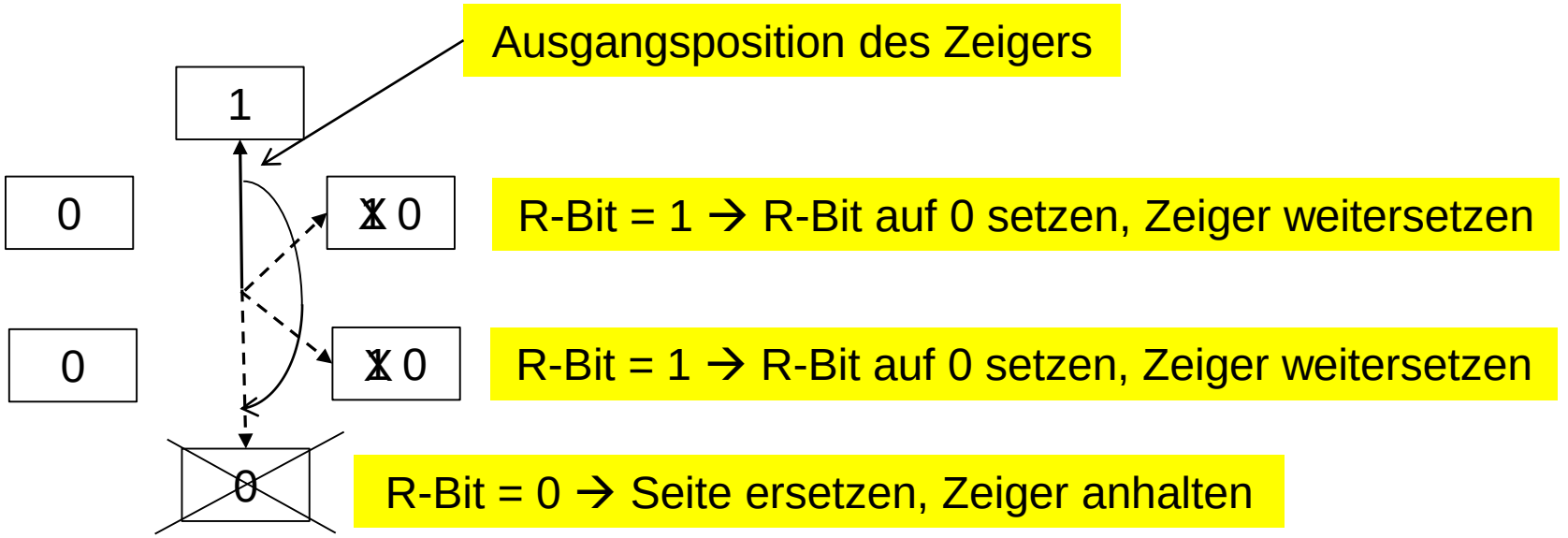
Ersetzungsstrategie 2/3: Least Recently Used (LRU)

- Annäherung an die optimale Strategie
- Ersetzt die Seite, die am längsten nicht benutzt wurde
- Nach dem **Lokalitätsprinzip** ist die Wahrscheinlichkeit hoch, dass die Seite auch in Zukunft lange nicht benutzt wird
- Implementierung: Jede Seite müsste mit einem Zeitstempel für den letzten Zeitpunkt der Benutzung versehen werden
  - Das ist in der Realität aber extrem aufwendig!

Ersetzungsstrategie 3/3: Clock-Algorithmus

- Ziel: Leicht implementierbare Annäherung an LRU
- Führt eine zirkulare Liste („Clock“) mit Zeigern auf Seitentabelleneinträge für alle Seiten des Prozesses, die sich im Hauptspeicher befinden (mit Speicherung der letzten Zeigerposition)
- Verwendet das Reference-Bit (R-Bit)
  - Wenn die Seite geladen wird: R-Bit = 0
  - Wenn auf die Seite zugegriffen wird: R-Bit = 1 (durch die MMU)
- Seitenfehler → Zeiger durchläuft die Clock-Liste ab der letzten Position (im „Uhrzeigersinn“)
  - Seite mit R-Bit = 1 → R-Bit zurücksetzen auf 0
  - Erste gefundene Seite mit R-Bit = 0 → Seite ersetzen (auch in der Clock-Liste)!

Clock-Algorithmus: Beispiel



Die ersetzte Seite wurde während des gesamten letzten Durchlaufs nicht gebraucht

= Annäherung an „lange“ nicht gebraucht!

Aufgaben Abschnitt 4.2.c): Pagingstrategien

Quelle: Tanenbaum, Moderne Betriebssysteme, 3. Auflage, Kapitel 3, Aufgabe 28

28. Ein Computer hat vier Seitenrahmen. Die Tabelle zeigt für jede Seite die Ladezeit, die Zeit des letzten Zugriffes sowie die *R*- und *M*-Bits. Die Zeiten sind jeweils in Timerintervallen angegeben.

Seite	geladen	letzter Zugriff	R	M
0	126	280	1	0
1	230	265	0	1
2	140	270	0	0
3	110	285	1	1

a. Welche Seite ersetzt NRU?

b. Welche Seite ersetzt FIFO?

c. Welche Seite ersetzt LRO?

d. Welche Seite ersetzt Second Chance?

Anmerkung:

LRO = LRU

Speicherzuteilungsstrategien: Grundlegende Überlegungen (1/2)

- Je **weniger Platz für den einzelnen Prozess** zur Verfügung steht, desto **mehr Prozesse** können im Hauptspeicher resident sein
  (➔ Seitenzahl pro Prozess minimieren!)
- Stehen **einem Prozess zu wenig Seiten** zur Verfügung, dann wird die **Seitenfehlerrate** trotz Lokalitätsprinzip sehr hoch sein
  (➔ Seitenzahl pro Prozess maximieren!)
- **Über eine bestimmte Größe** hinaus wird sich zusätzlicher Speicher nur **geringfügig** auf die Seitenfehlerrate auswirken
  (➔ Seitenzahl pro Prozess optimieren!)

Speicherzuteilungsstrategien: Grundlegende Überlegungen (2/2)

- Verteilung der freien Hauptspeicherseiten auf die existierenden Prozesse:
  - Einem Prozess wird eine **feste Anzahl** von Hauptspeicherseiten zugebilligt.
  - Einem Prozess werden während seiner Ausführung abhängig von seinem Verhalten (Seitenfehlerrate) eine **variable Anzahl** von Hauptspeicherseiten zur Verfügung gestellt.
- Strategien nach Auftreten von Seitenfehlern :
  - Bei **lokalen Strategien** muss eine Seite des aktiven Prozesses ersetzt werden.
  - Bei **globalen Strategien** sind alle Seiten im Hauptspeicher Kandidaten für die Ersetzung.

fest variabel lokal global

Speicherzuteilung:
V1: Feste Seitenanzahl und lokale Strategie

lokal fest

- Einem Prozess steht für seine Ausführung eine feste Anzahl von Hauptspeicherseiten zur Verfügung.
- Tritt ein Seitenfehler auf, dann muss das Betriebssystem entscheiden, welche andere Seite dieses Prozesses ersetzt werden muss
  - Das wesentliche Problem liegt in der Festlegung der Seitenzahl (x % der gesamten Prozessgröße?)

Speicherzuteilung:
V2: Variable Seitenzahl und globale Strategie

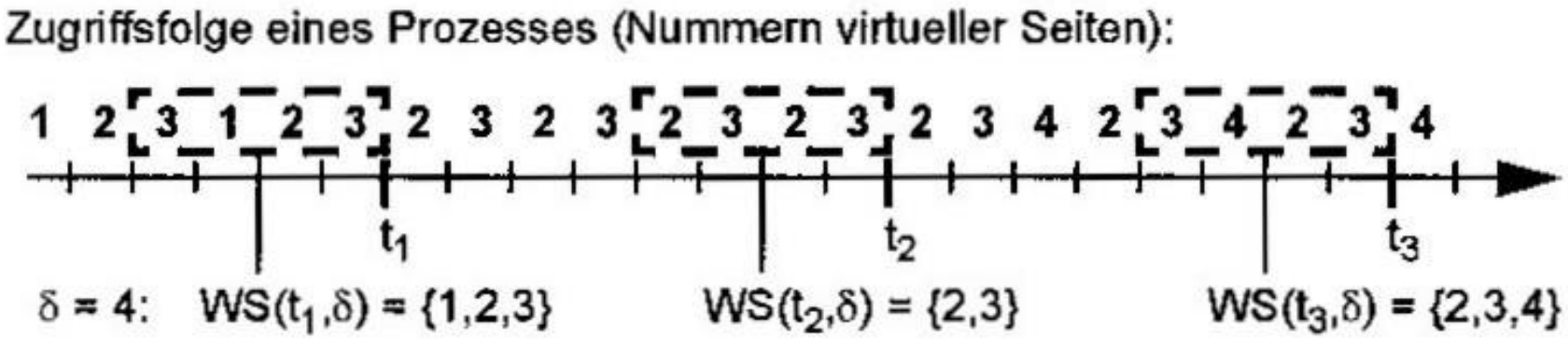
global variabel

- Den im Hauptspeicher befindlichen Prozessen stehen eine variable Anzahl von Seiten zur Verfügung.
- Seitenfehler → Zuweisung einer freien Seite, falls verfügbar
- Keine freien Seiten mehr verfügbar → Seite zur Ersetzung auswählen (beliebiger Prozess!)
- Zu viele Prozesse im Hauptspeicher → u.U. sind die zur Verfügung stehenden Speicherbereiche nicht mehr ausreichend und die Seitenfehlerrate wird sehr groß
- Jedes Programm, das zusätzliche Seiten benötigt, erhält diese auf Kosten von Seiten anderer Programme. Da diese aber ebenfalls noch benötigt werden, werden in immer schnellerer Folge weitere Seitenfehler erzeugt. (Seitenflattern, engl. „**Thrashing**“). Die Prozesse verbrauchen dann für das Seitenwechseln mehr Zeit als für ihre Ausführung.

V3: Variable Seitenzahl und lokale Strategie: Working Set-Strategie -> Working Set - Modell

lokal variabel

- Der **Working Set WS (t,δ)** eines Prozesses zur Zeit t ist die Menge aller Seiten, die er bei den letzten δ Speicherzugriffen angesprochen hat.
- Der Umfang des Working Set ist abhängig von δ. Im Extremfall umfasst der Working Set das gesamte Programm
- Je größer der Working Set, umso geringer die Seitenfehlerrate
- Beispiel:



V3: Variable Seitenzahl und lokale Strategie: Working Set-Strategie

lokal variabel

- Der Working Set jedes Prozesses wird beobachtet.
- Seitenfehler → Zuweisung einer freien Seite, falls verfügbar
- Periodisch wird die zugewiesene Seitenanzahl an den Working Set angepasst: diejenigen Seiten werden aus dem Speicher entfernt, die nicht mehr zum Working Set gehören
- Ein Prozess darf nur dann ausgeführt werden, wenn sein Working Set im Hauptspeicher resident ist

Mögliche Implementierung (grobe Annäherung):

- Periodisch werden diejenigen Seiten aus dem Speicher entfernt, deren Reference-Bit = 0 ist
  (kein Zugriff erfolgt seit letzter Überprüfung).
- Seiten mit Reference-Bit =1 (~ Working Set):
  Reference-Bit zurücksetzen!

Alternative / Ergänzung: Page-Fault Frequency (PFF) - Strategie

- Für die Seitenfehlerrate werden untere und obere Grenzen definiert.
- Seitenfehler → Ersetzung einer Prozess-eigenen Seite
- Wird die untere Grenze erreicht, dann werden dem Prozess Speicherseiten weggenommen
- Erreicht ein Prozess in Ausführung die obere Grenze, dann werden ihm – wenn möglich – neue Speicherseiten hinzugefügt. Sind keine weiteren Speicherseiten verfügbar, so muss der Prozess suspendiert und ausgelagert werden.

Aufgaben Abschnitt 4.2.c): Pagingstrategien

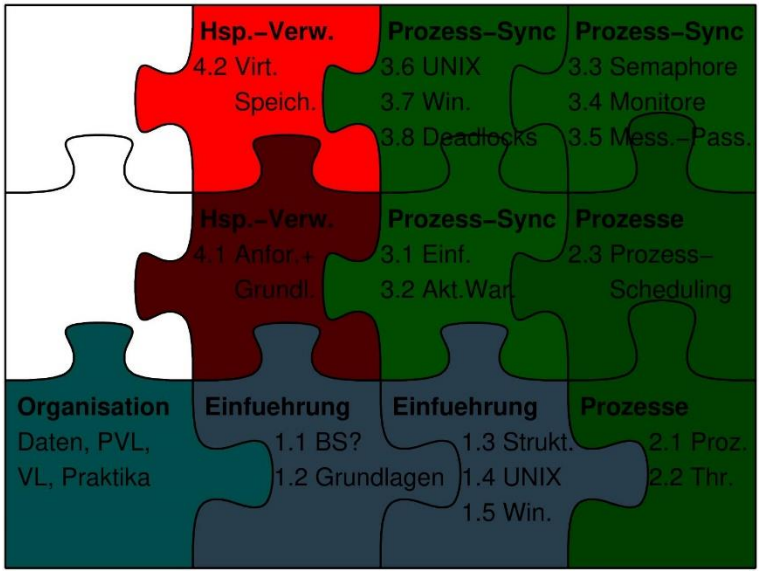
Quelle: Tanenbaum, Moderne Betriebssysteme, 3. Auflage, Kapitel 3, Aufgabe 31

31. Ein Computer stellt jedem Prozess einen Adressraum von 64 KB zur Verfügung, aufgeteilt in 4-KB-Seiten. Ein Programm hat 32.768 Byte Programmcode, 16.386 Byte Daten und 15.870 Byte Stack. Passt dieses Programm in den Adressraum? Würde es passen, wenn die Seiten 512 Byte groß wären? Denken Sie daran, dass eine Seite nicht zwei Teile von verschiedenen Segmenten enthalten kann.

Anmerkung: Aufgrund von Rechten liegen Programm-, Daten- und Stacksegment nicht auf gleichen Seiten.

Kapitel 4
Hauptspeicher-Verwaltung

1. Anforderungen und Grundlagen
2. Virtueller Speicher
   a) Einführung und Prinzipien
   b) Paging
   c) Pagingstrategien
   d) Unix / Windows
   e) Meltdown / Spectre



Virtueller Speicher in UNIX: Paging

- Speicherzuteilung: **Variable** Seitenzahl und **globale** Strategie
- Seitenfehler → Zuweisung einer freien Seite
- **Page Daemon** (Prozess-ID 2)
  - Wacht alle 250 ms auf
  - Falls Anzahl freier Seiten zu klein: Verwendet modifizierten Clock-Algorithmus, um **beliebigen** Prozessen Seiten zu entziehen
- **Swapper**
  - lagert Prozesse aus, falls zu viele Seitenfehler auftreten
  - lagert erst wieder ein, wenn genügend freie Seiten zur Verfügung stehen (Scheduling!)

Virtueller Speicher in Windows: Paging (1/2)

- Speicherzuteilung: **Variable** Seitenzahl und **lokale** Strategie
- **Eigene Working Set-Definition:** Alle Seiten des Prozesses, die sich momentan im Hauptspeicher befinden
- Ein Working-Set hat eine **minimale** und **maximale** Grenze (bei Start für alle Prozesse gleich)
- Seitenfehler:
  - **Working Set < Maximum** → **neue** Seite hinzufügen
  - sonst: Seite aus Working Set **ersetzen**! (lokal)
  - bei vielen Seitenfehlern: Maximum für den Prozess erhöhen (Kombination mit PFF-Strategie)

Virtueller Speicher in Windows: Paging (2/2)

- **Balance-Set-Manager**
  - Wacht einmal pro Sekunde auf
  - Falls Anzahl freier Seiten zu klein: Startet **Working-Set-Manager**, um Prozessen Seiten zu entziehen
- **Working-Set-Manager**
  - Untersucht alle Prozesse in definierter Reihenfolge (große Prozesse vor kleinen, Hintergrundprozesse vor Vordergrundprozessen)
  - Falls **Working Set < Minimum** oder bereits **viele Seitenfehler** → **ignorieren**
  - Sonst: Prozess-Seiten anhand von modifiziertem Clock-Algorithmus aus Hauptspeicher (Working Set) **entfernen**