Speicherverwaltung (2)

Ute Bormann, TI2

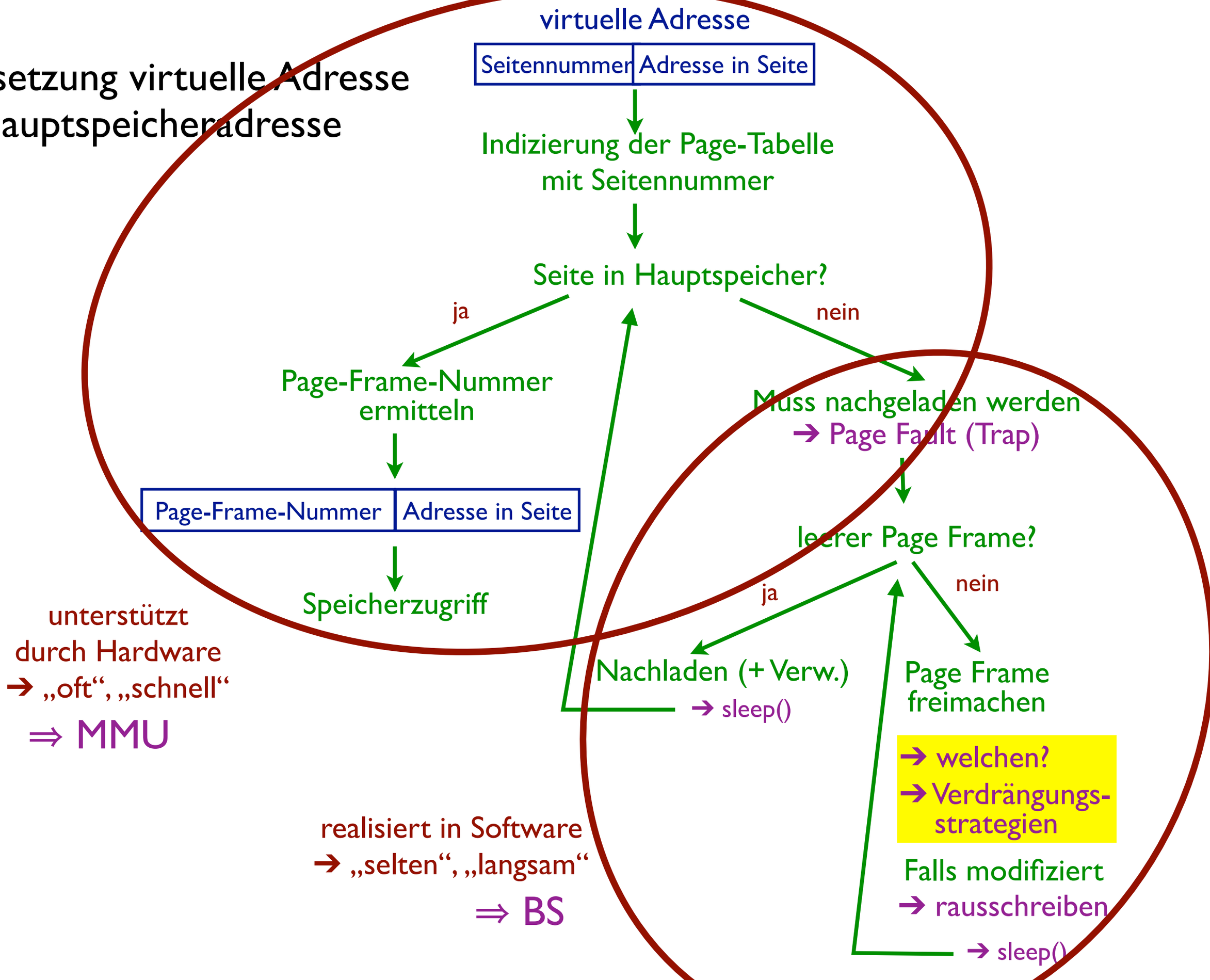
2023-10-13

Inhalt

1. Prinzipien der Seitenverdrängung
2. Annäherung an LRU
3. Weitere Aspekte des Pagings

Teil 1:
Prinzipien der Seitenverdrängung

Umsetzung virtuelle Adresse in Hauptspeicheradresse



unterstützt durch Hardware
→ „oft“, „schnell“
⇒ MMU

realisiert in Software
→ „selten“, „langsam“
⇒ BS

Verdrängung von Seiten aus dem Hauptspeicher

- Wird neue Seite im Speicher benötigt (und kein Page Frame frei)

  ⇒ Verdrängung einer anderen ⇒ Welche?

- Ziel: Jene, die am längsten nicht mehr benötigt wird (optimale Ersetzung)

Verdrängung von Seiten aus dem Hauptspeicher

- Wird neue Seite im Speicher benötigt (und kein Page Frame frei)
  ⇒ Verdrängung einer anderen ⇒ Welche?
- Ziel: Jene, die am längsten nicht mehr benötigt wird (optimale Ersetzung)
- Problem: Betriebssysteme sind keine Hellseher
- Abhilfe: Zukunftsprognose wagen
  a) Keine Prognose: Zufällige Auswahl
     ⇒ einfach, aber zu pessimistisch

Verdrängung von Seiten aus dem Hauptspeicher

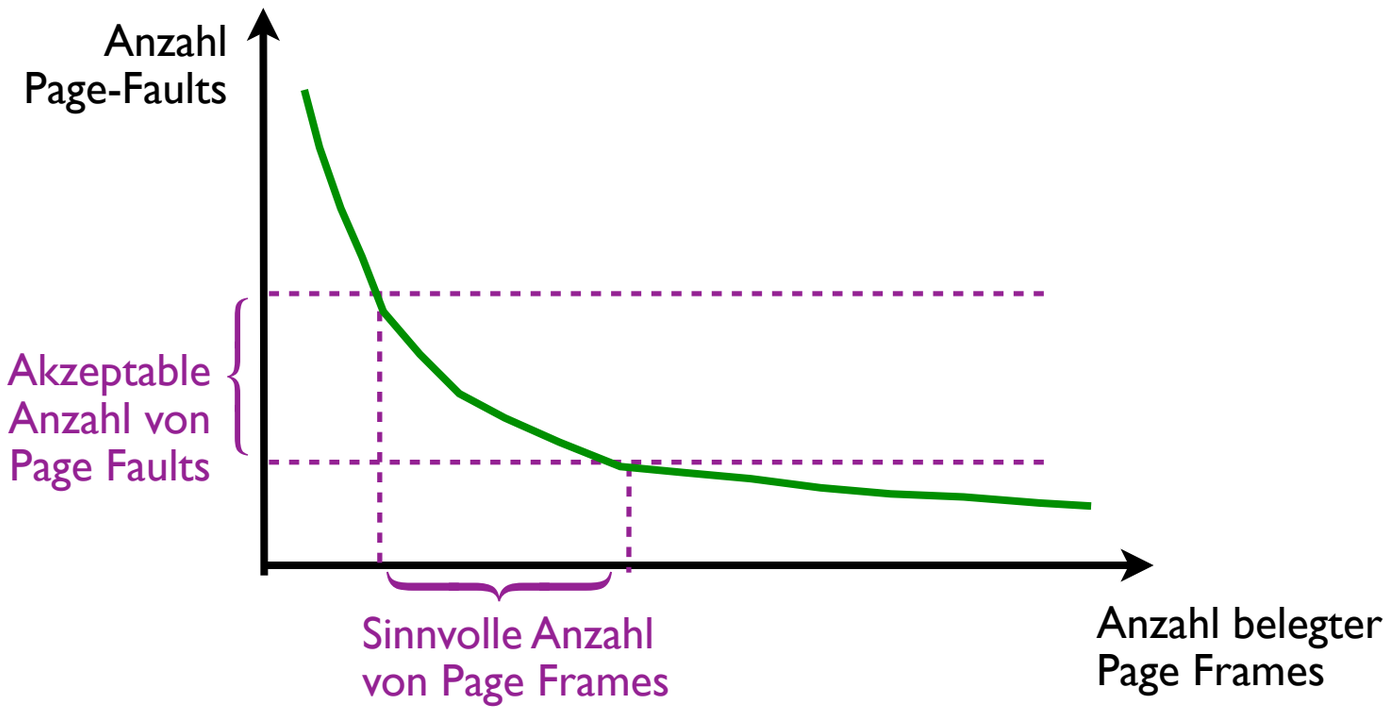
- Wird neue Seite im Speicher benötigt (und kein Page Frame frei)
  ⇒ Verdrängung einer anderen ⇒ Welche?
- Ziel: Jene, die am längsten nicht mehr benötigt wird (optimale Ersetzung)
- Problem: Betriebssysteme sind keine Hellseher
- Abhilfe: Zukunftsprognose wagen
  a) Keine Prognose: Zufällige Auswahl
     ⇒ einfach, aber zu pessimistisch
  b) Rückschlüsse aus der (jüngeren) Vergangenheit ziehen
     - „was bisher galt, wird sich demnächst nicht wesentlich ändern“
       ⇒ Lokalitätsprinzip von Prozessen
     - Ziel: Aktuelle Working Sets der lauffähigen Prozesse ermitteln

- Nicht alle Seiten des Working Sets im Hauptspeicher

  ⇒ hohe Page-Fault-Rate

- Mehr als Working Set im Hauptspeicher

  ⇒ Page-Fault-Rate sinkt nicht ⇒ Platzvergeudung



- Dazu: Informationen über (jüngere) Vergangenheit sammeln

  ⇒ Achtung: Dabei Verwaltungsaufwand bei den einzelnen Speicherzugriffen minimieren
  (darf nicht zu weiteren Speicherzugriffen führen)

Wann findet Verdrängung statt?

- Zwei Alternativen (unabhängig vom Algorithmus):

  a) Verdrängung bei Bedarf

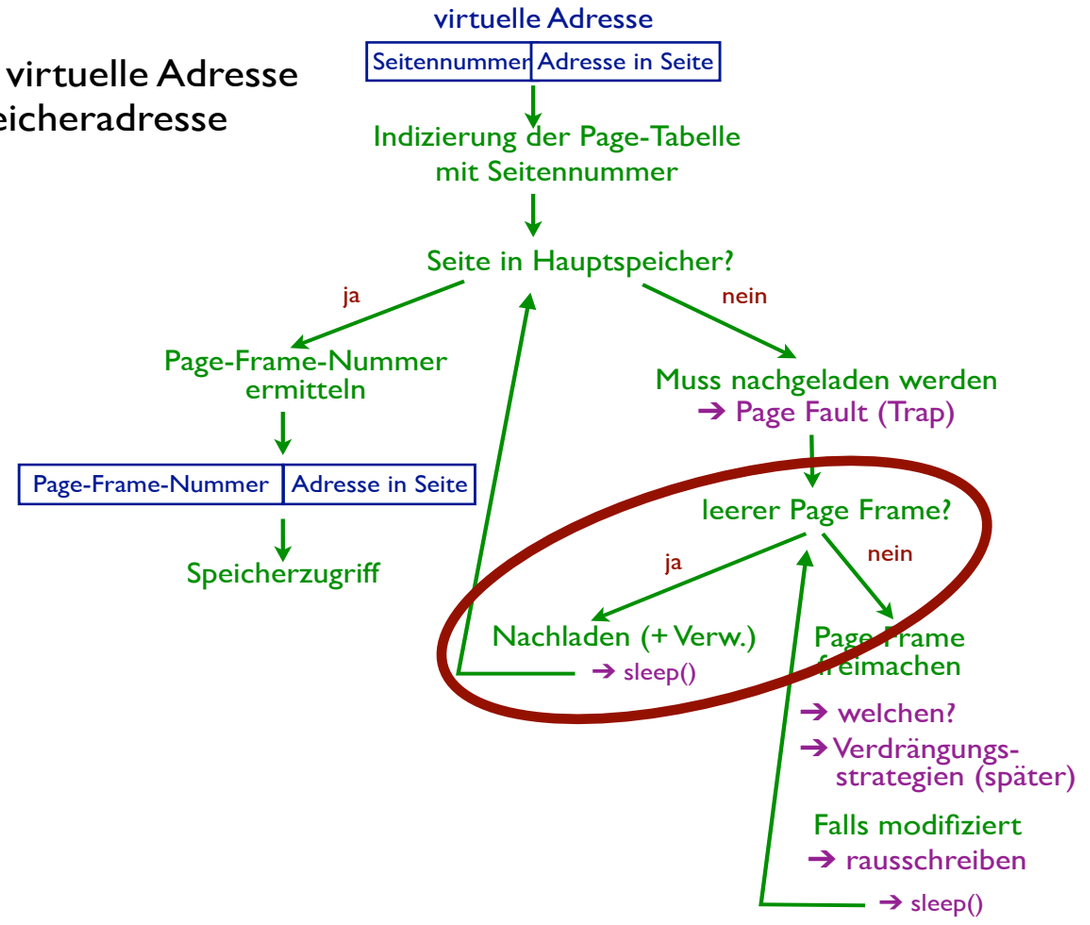
     - durch Page Fault getriggert

  b) Bereitstellung einer Freispeicherreserve

     - Verdrängung, sobald Minimalreserve unterschritten

  ⇒ Page Faults schneller behandelbar

- Im folgenden: Ermittlung des aktuellen Working Sets eines Prozesses (bzw. aller lauffähigen Prozesse)



1. Versuch: Sequentialität der Programmabarbeitung

- „Was kürzlich hinzukam, wird noch länger benötigt.“

  ⇒ Je länger im Speicher, desto wahrscheinlicher, dass inzwischen überholt

  ⇒ First-In-First-Out (FIFO)

- Entfernt „älteste“ Seite aus Hauptspeicher

- Bewertung: relativ einfach zu realisieren
  (verkettete Liste der Page Frames nach Belegungsalter)

1. Versuch: Sequentialität der Programmabarbeitung

- „Was kürzlich hinzukam, wird noch länger benötigt.“

  ⇒ Je länger im Speicher, desto wahrscheinlicher, dass inzwischen überholt

  ⇒ First-In-First-Out (FIFO)

- Entfernt „älteste“ Seite aus Hauptspeicher

- Bewertung: relativ einfach zu realisieren
  (verkettete Liste der Page Frames nach Belegungsalter)

- Aber: Bestimmt Working Set(s) i.d.R. nicht gut:
  - immer wieder benötigte Daten-/Stack-Pages
  - Code für regelmäßig benötigte Dienstprogramme

  ⇒ Belegungsalter kein brauchbares Indiz für aktuelle (Un)Wichtigkeit

⇒ Häufigkeit des Zugriffs scheint interessant...

2. Versuch: Zugriffshäufigkeit (z.B. Abarbeitung von Schleifen)

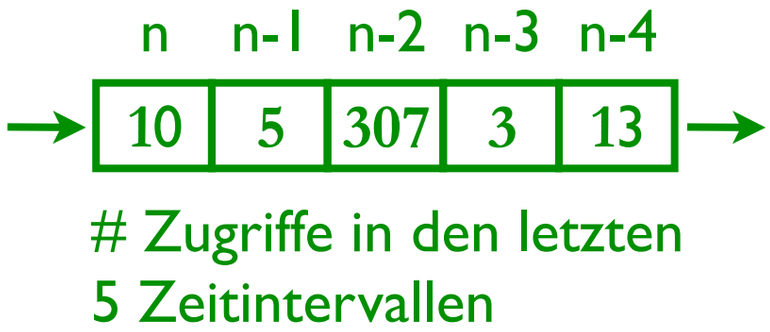
- „Was oft benutzt wird, muss direkt zugreifbar sein.“
  ⇒ Je seltener benutzt, desto wahrscheinlicher, dass unwichtig
  ⇒ Least-Frequently-Used (LFU)
- Entfernt die am seltensten benutzte Seite aus dem Hauptspeicher
- Bewertung: Zu aufwendig, da Zugriffszähler mitgeführt werden muss
  ⇒ bei jedem Speicherzugriff weitere Zugriffe nötig

2. Versuch: Zugriffshäufigkeit (z.B. Abarbeitung von Schleifen)

- „Was oft benutzt wird, muss direkt zugreifbar sein.“
  ⇒ Je seltener benutzt, desto wahrscheinlicher, dass unwichtig
  ⇒ Least-Frequently-Used (LFU)
- Entfernt die am seltensten benutzte Seite aus dem Hauptspeicher
- Bewertung: Zu aufwendig, da Zugriffszähler mitgeführt werden muss
  ⇒ bei jedem Speicherzugriff weitere Zugriffe nötig
- Außerdem: Annahme stimmt oft nicht
  - Seiten, die vor langer Zeit häufig benutzt wurden, bleiben im Hauptspeicher

⇒
  - Frisch geladene Seite wird entfernt

  ⇒ nur mit Hardwareunterstützung und einem Veralterungskonzept tragfähig
  ⇒ Aging



3. Versuch: Zugriffszeitpunkt

- „Was kürzlich gebraucht wurde, ist im Working Set“

  ⇒ Je länger nicht mehr benutzt, desto wahrscheinlicher, dass es unwichtig ist

  ⇒ Least-Recently-Used (LRU)

- Entfernt Seite im Hauptspeicher, auf die am längsten nicht zugegriffen wurde

- Bewertung: Lokalitätsprinzip i.d.R. gut erfasst
  (Gibt natürlich auch Gegenbeispiele, z.B. extrem lange Schleifen)

⇒ Gute Approximation an optimalen Algorithmus

3. Versuch: Zugriffszeitpunkt

- „Was kürzlich gebraucht wurde, ist im Working Set“

  ⇒ Je länger nicht mehr benutzt, desto wahrscheinlicher, dass es unwichtig ist

  ⇒ Least-Recently-Used (LRU)

- Entfernt Seite im Hauptspeicher, auf die am längsten nicht zugegriffen wurde

- Bewertung: Lokalitätsprinzip i.d.R. gut erfasst
  (Gibt natürlich auch Gegenbeispiele, z.B. extrem lange Schleifen)

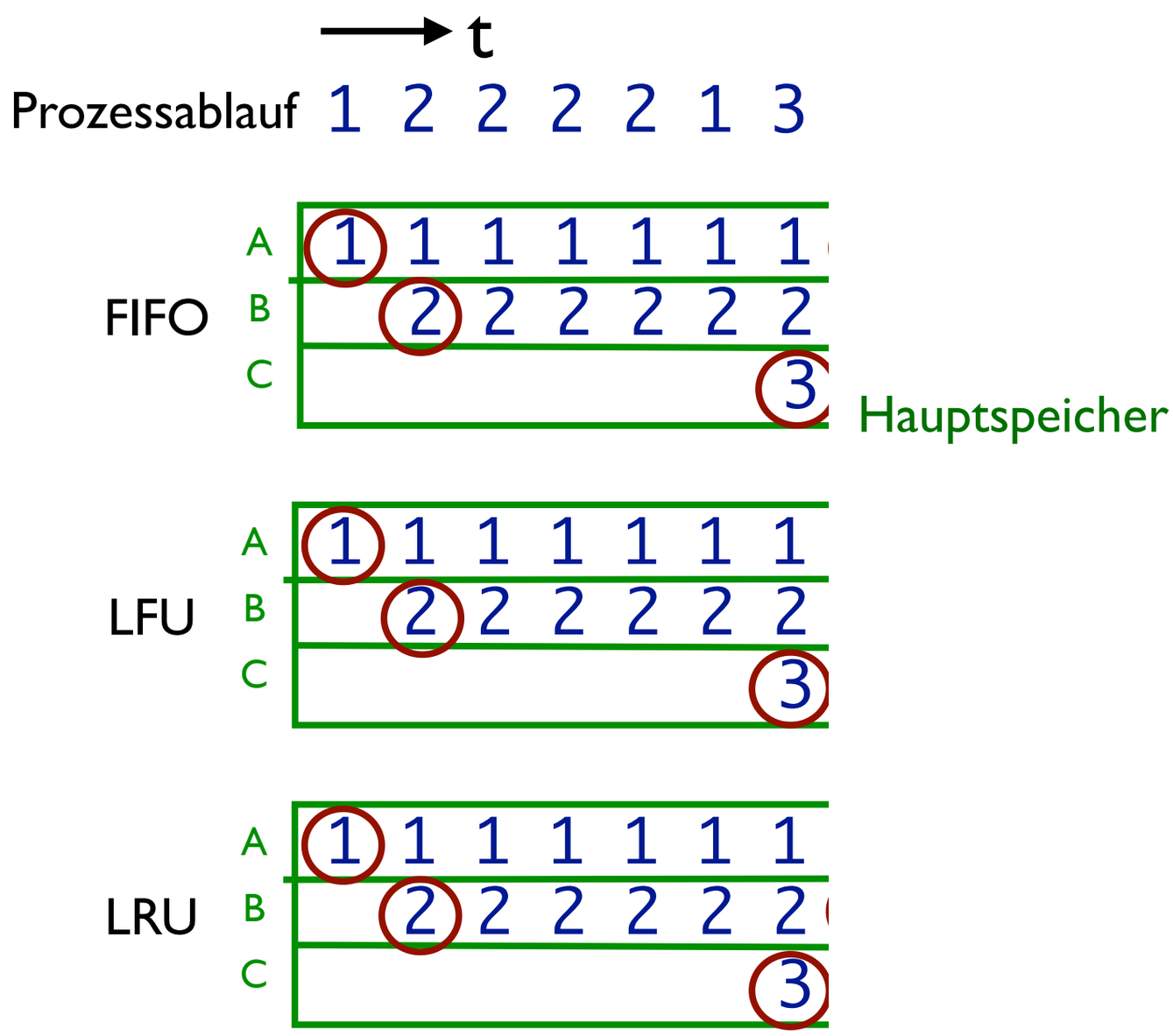
  ⇒ Gute Approximation an optimalen Algorithmus

- Aber:
  - Erfordert Erfassung aller Zugriffszeiten auf Seiten

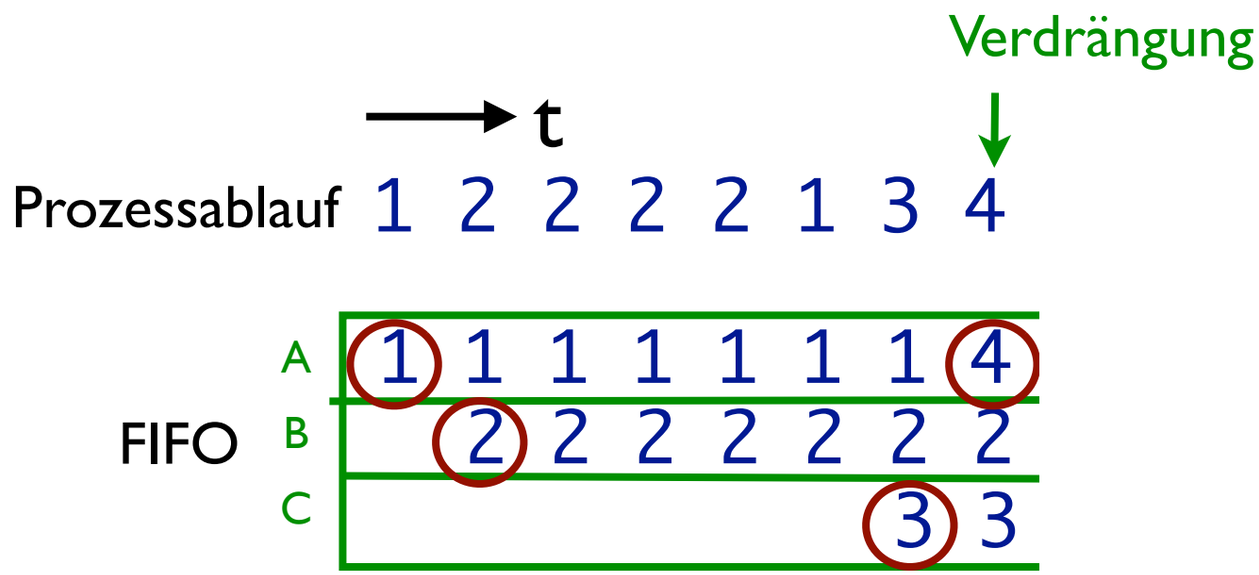
    ⇒ Bei jedem Speicherzugriff weitere Zugriffe nötig
  - Ohne Spezialhardware viel zu aufwendig
  - Solche Hardware selten vorhanden

⇒ Neues Ziel: Approximation von LRU

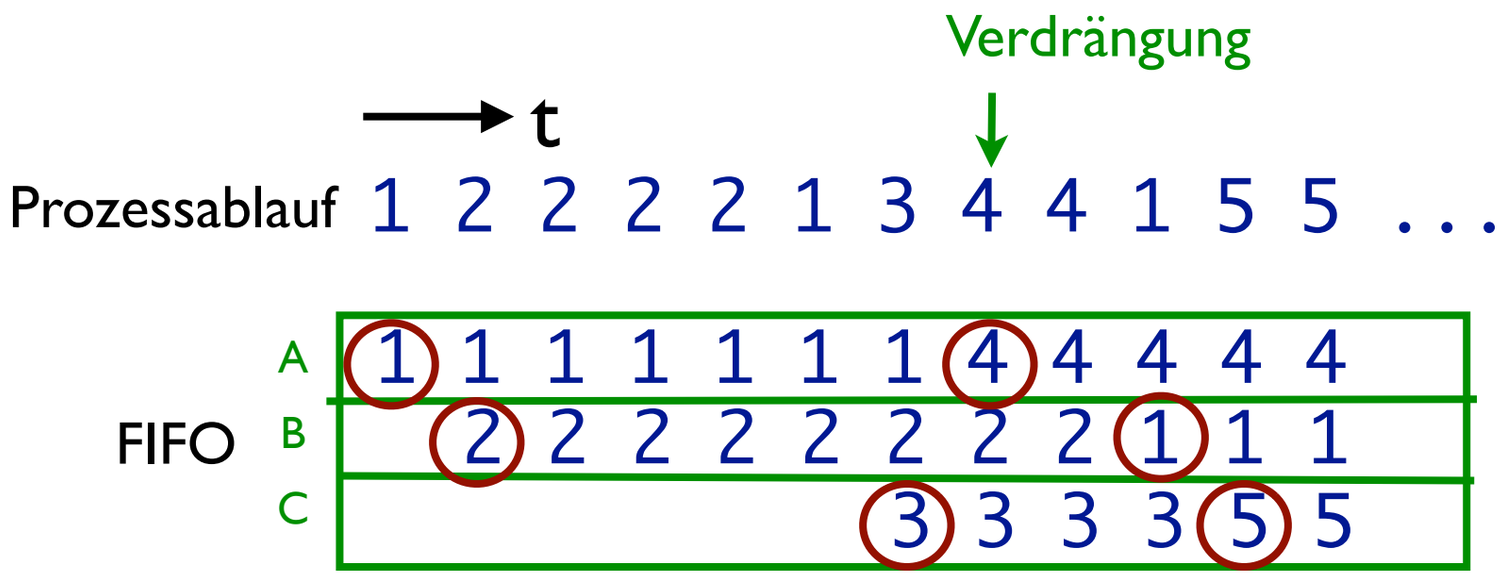
Beispiel:


t
Prozessablauf 1 2 2 2 2 1 3
FIFO
A 1 1 1 1 1 1 1
B 2 2 2 2 2 2
C 3
Hauptspeicher
LFU
A 1 1 1 1 1 1 1
B 2 2 2 2 2 2
C 3
LRU
A 1 1 1 1 1 1 1
B 2 2 2 2 2 2
C 3

Beispiel:

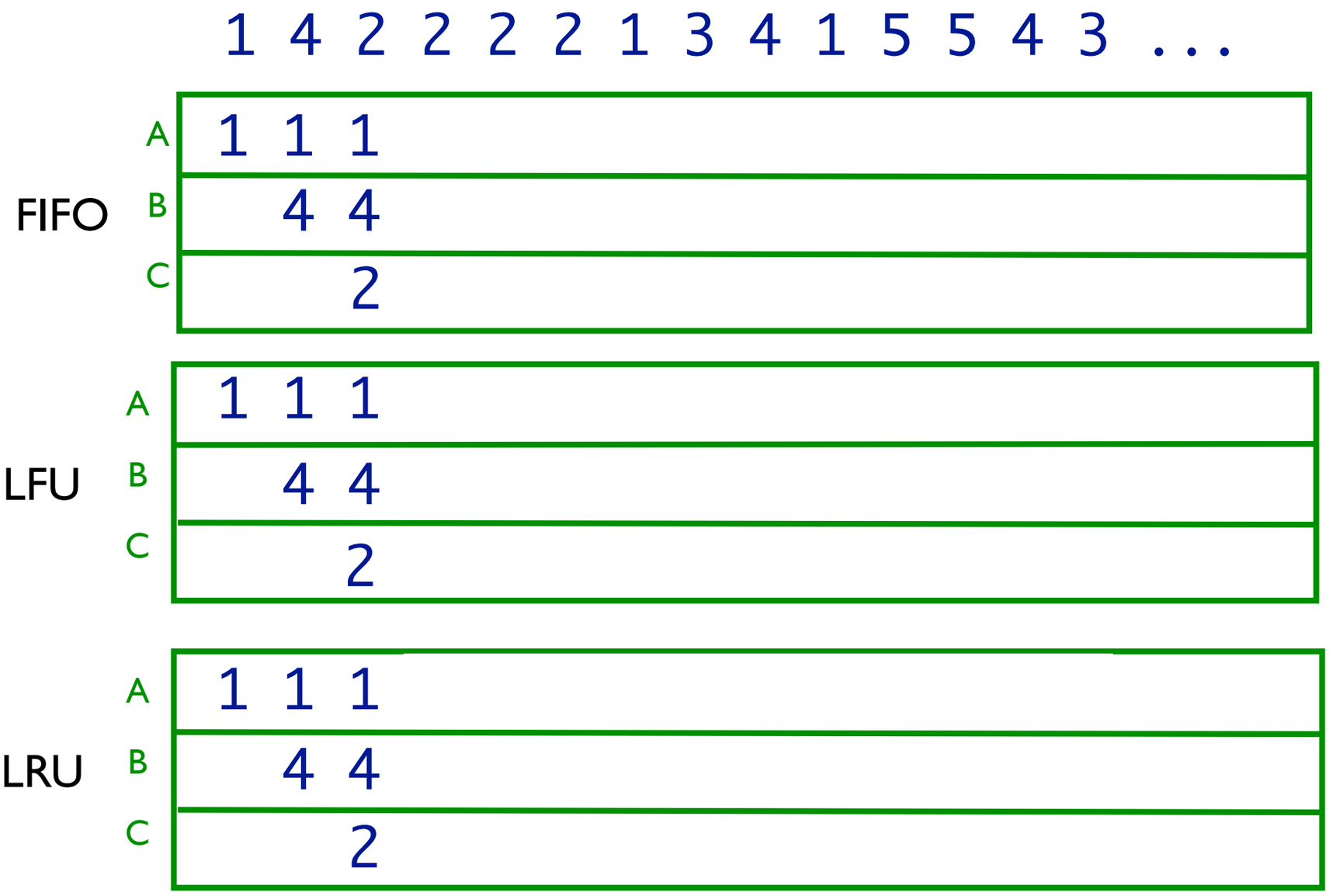

Verdrängung
t
Prozessablauf 1 2 2 2 2 1 3 4
FIFO
A 1 1 1 1 1 1 1 4
B 2 2 2 2 2 2 2
C 3 3
LFU
A 1 1 1 1 1 1 1 1
B 2 2 2 2 2 2 2
C 3 4
LRU
A 1 1 1 1 1 1 1 1
B 2 2 2 2 2 2 4
C 3 3

Beispiel:


Verdrängung
t
Prozessablauf 1 2 2 2 2 1 3 4 4 1 5 5 . . .
FIFO
A 1 1 1 1 1 1 1 4 4 4 4 4
B 2 2 2 2 2 2 2 2 1 1 1
C 3 3 3 3 5 5
LFU
A 1 1 1 1 1 1 1 1 1 1 1 1
B 2 2 2 2 2 2 2 2 2 2 2 2
C 3 4 4 4 5 5
LRU
A 1 1 1 1 1 1 1 1 1 1 1 1
B 2 2 2 2 2 2 4 4 4 4 4
C 3 3 3 3 5 5

Kleine Aufgabe

- Wie arbeiten die drei angegebenen Verdrängungs-algorithmen bei den angegebenen Zugriffen?



Fragen – Teil 1

- Warum ist ein perfekter Algorithmus zur Verdrängung von Pages aus dem Hauptspeicher nicht realisierbar?
- Wie arbeiten die folgenden Algorithmen in etwa:
  - FIFO (First-In-First-Out),
  - LFU (Least-Frequently-Used),
  - LRU (Least-Recently-Used)?

Teil 2:
Annäherung an LRU

Annäherungen an LRU

- „Kürzlich“ referenzierte Seiten von lange nicht referenzierten Seiten unterscheiden
- „Güte“ des Algorithmus hängt von Auswahlverfahren und Alterungsprozess ab.
- Beispiele:
  - (Aging ⇒ Literatur)
  - (Second-Chance-FIFO ⇒ Literatur)

⇒
  - Not-Recently-Used (NRU)

Not-Recently-Used (NRU)

Sehr einfache LRU-Approximation: Vorgehen:

1. Bei Zugriff auf Seite Referenzbit (R) setzen

   ⇒ Achtung: Zusätzlicher Speicherzugriff auf PTE?

   - Entweder Hardwareunterstützung nötig
   - Oder weitere Optimierung nötig (⇒ später)

Not-Recently-Used (NRU)

Sehr einfache LRU-Approximation: Vorgehen:

1. Bei Zugriff auf Seite Referenzbit (R) setzen

   ⇒ Achtung: Zusätzlicher Speicherzugriff auf PTE?

   - Entweder Hardwareunterstützung nötig
   - Oder weitere Optimierung nötig (⇒ später)

2. Alterungsprozess: Verfahren zum Rücksetzen des R-Bits (z.B. periodisch)

Not-Recently-Used (NRU)

Sehr einfache LRU-Approximation: Vorgehen:

1. Bei Zugriff auf Seite Referenzbit (R) setzen

   ⇒ Achtung: Zusätzlicher Speicherzugriff auf PTE?

   - Entweder Hardwareunterstützung nötig
   - Oder weitere Optimierung nötig (⇒ später)

2. Alterungsprozess: Verfahren zum Rücksetzen des R-Bits (z.B. periodisch)

3. Ggf. weitere Zugriffe... (s. 1.)

Not-Recently-Used (NRU)

Sehr einfache LRU-Approximation: Vorgehen:

1. Bei Zugriff auf Seite Referenzbit (R) setzen

   ⇒ Achtung: Zusätzlicher Speicherzugriff auf PTE?

   - Entweder Hardwareunterstützung nötig
   - Oder weitere Optimierung nötig (⇒ später)

2. Alterungsprozess: Verfahren zum Rücksetzen des R-Bits (z.B. periodisch)

3. Ggf. weitere Zugriffe... (s. 1.)

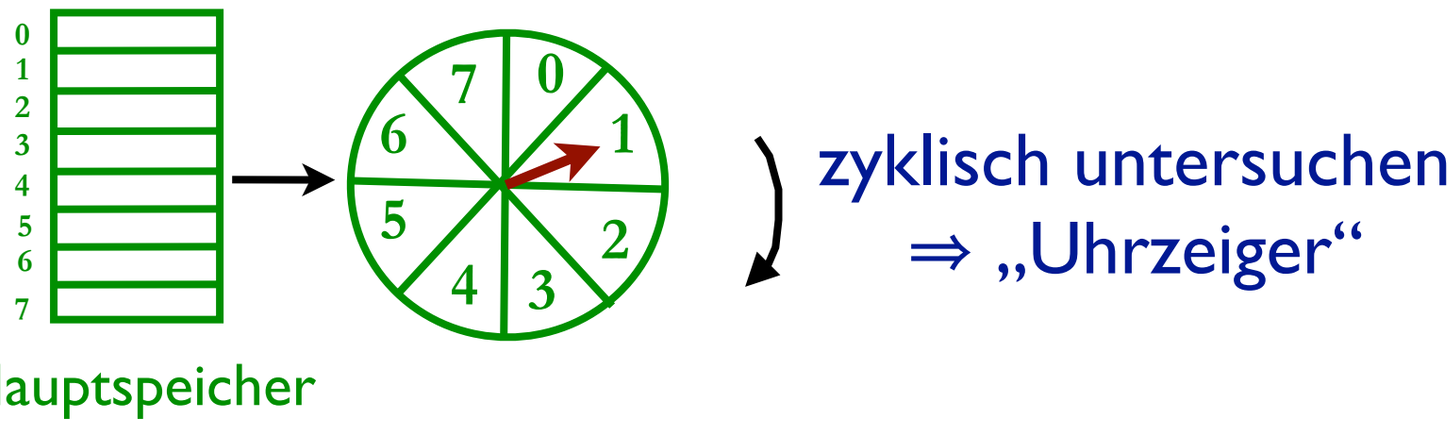
4. Verdrängung: Seite auswählen, deren R-Bit nicht (wieder) gesetzt ist

   ⇒ Seite wurde länger nicht benötigt

   Bei NRU: zufällige Auswahl einer solchen Seite (wesentlicher Unterschied zu Alternativ-Verfahren)

Seitenersetzung in Unix: Varianten des Clock-Hand-Algorithmus

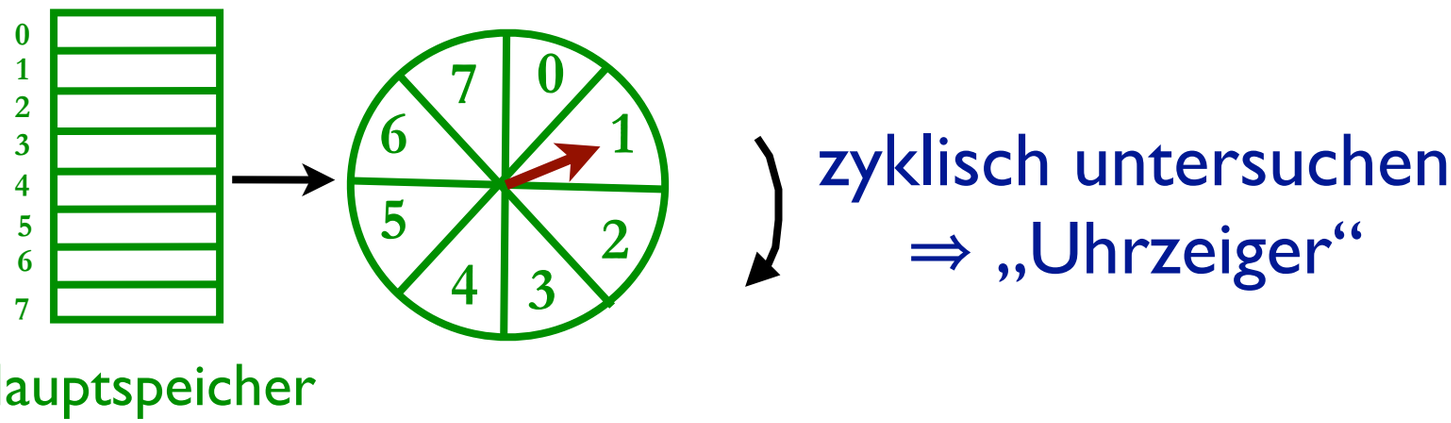
- Grundlage: Einfache NRU-Variante
- Page Frames wie Ziffernblatt einer Uhr angeordnet vorstellen



R-Bit = 1 ⇒ zurücksetzen (und ggf. weitersuchen)
R-Bit = 0 ⇒ Page Frame freimachen

Seitenersetzung in Unix: Varianten des Clock-Hand-Algorithmus

- Grundlage: Einfache NRU-Variante
- Page Frames wie Ziffernblatt einer Uhr angeordnet vorstellen



R-Bit = 1 ⇒ zurücksetzen (und ggf. weitersuchen)
R-Bit = 0 ⇒ Page Frame freimachen

- Darüber hinaus: Verdrängung nicht an Page Fault gekoppelt
  ⇒ Uhr läuft „automatisch“ weiter

Wann findet Verdrängung statt?

- Zwei Alternativen (unabhängig vom Algorithmus):

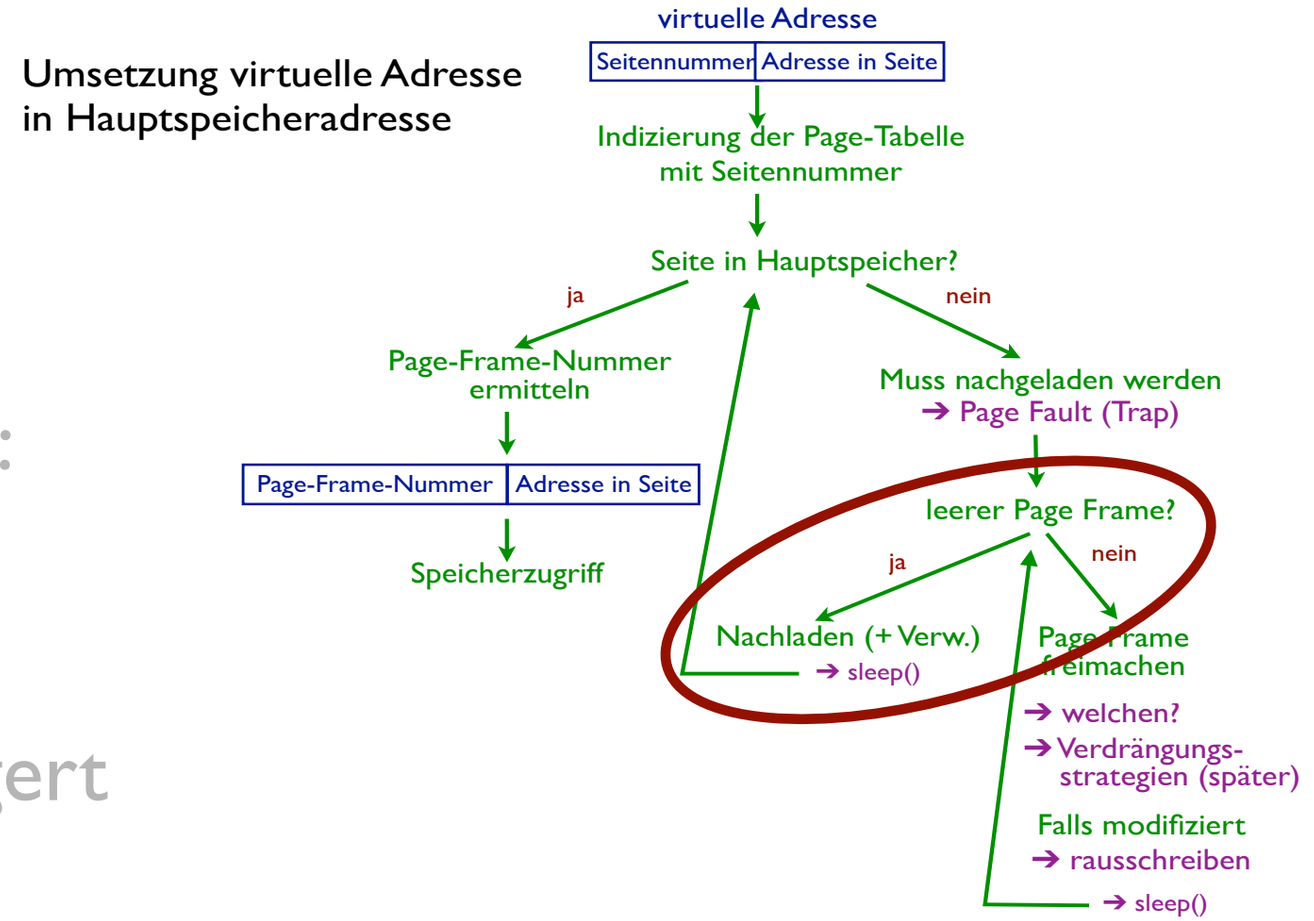
  a) Verdrängung bei Bedarf

  - durch Page Fault getriggert

→ b) Bereitstellung einer Freispeicherreserve

  - Verdrängung, sobald Minimalreserve unterschritten

⇒ Page Faults schneller behandelbar



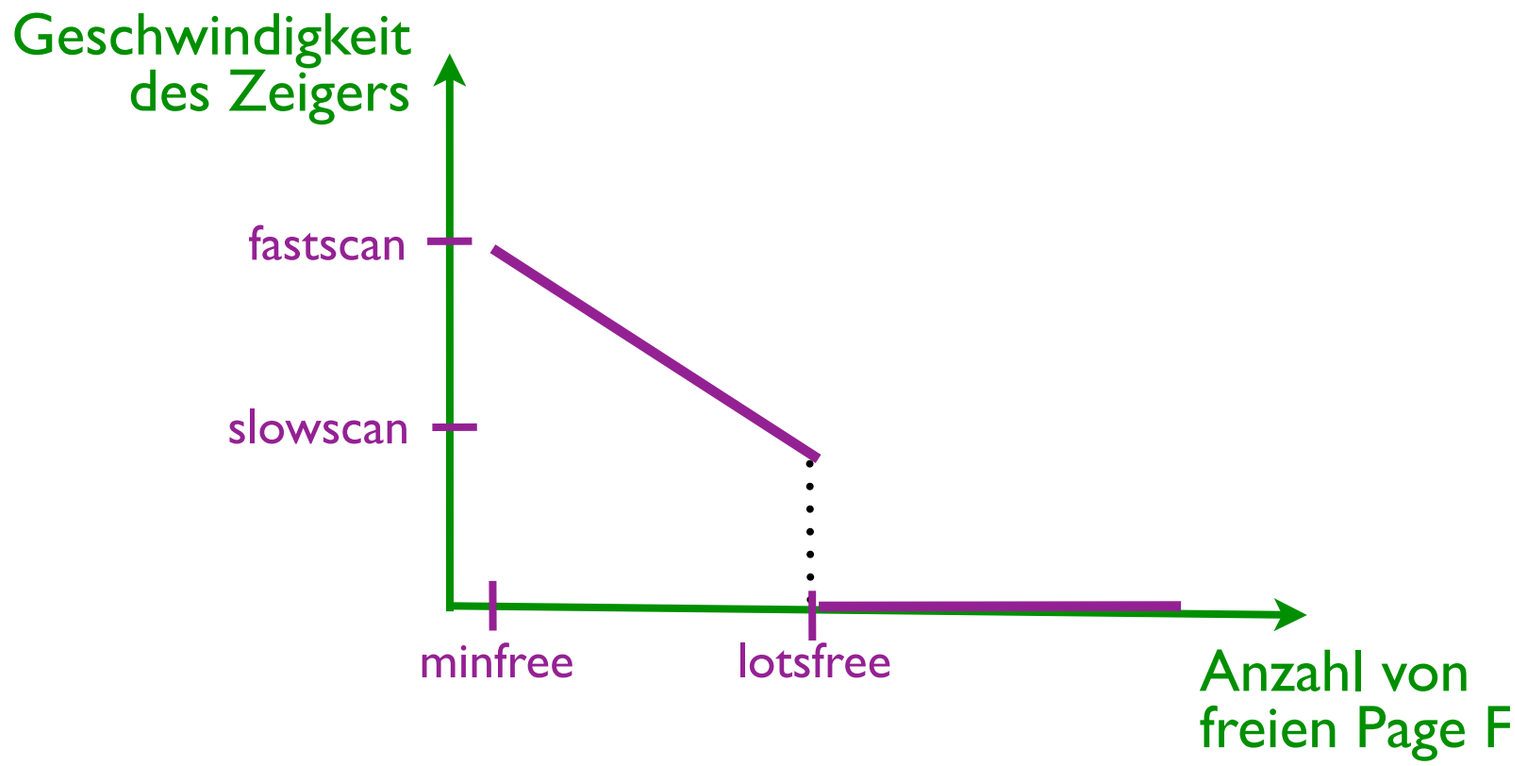
Geschwindigkeit des Zeigers:

- zu langsam:
  - Freispeicherreserve ⇒ leer
  - ggf. alle Page Frames (wieder) referenziert
    ⇒ keine Auswahl möglich

Geschwindigkeit des Zeigers:

- zu langsam:
  - Freispeicherreserve ⇒ leer
  - ggf. alle Page Frames (wieder) referenziert
    ⇒ keine Auswahl möglich
- zu schnell:
  - Viele Page Frames unreferenziert
    ⇒ große Freispeicherreserve
  - Viele Inhalte doch noch benötigt
    ⇒ unnötige Verdrängung
  - Allerdings: Falls noch nicht wieder vergeben
    ⇒ einfach wiederzurückfordern (Reclaim).
    ⇒ Neuvergabe nach FIFO-Strategie

- Geschwindigkeit leicht anpassbar (abhängig von Freispeicherreserve):



- Anzahl freier Page Frames:

  > lotsfree: Zeiger stoppt

  < minfree: weitere Beschleunigung würde System zu sehr belasten

  ⇒ Swapping (später)

- Mögliche Werte z.B.

  slowscan ca. 100 PageFrames/s

  fastscan ca. 200 PageFrames/s

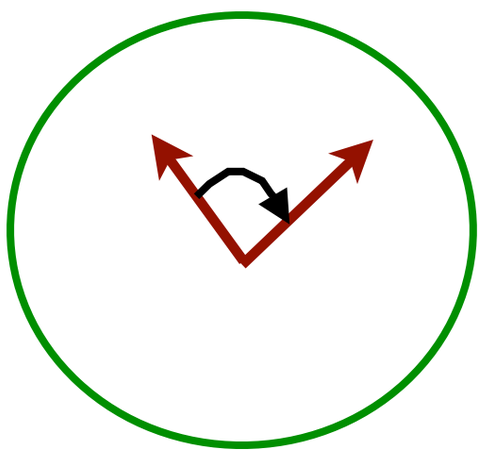
Der Two-Handed-Clock-Algorithmus

- Großer Speicher

  ⇒ Umlauf dauert zu lange

  ⇒ Träge Reaktion

- Einführung eines zweiten Zeigers in festem Abstand



- Erster Zeiger setzt ggf. R-Bits zurück
- Zweiter Zeiger markiert Page Frame als frei, falls noch nicht wieder referenziert

Wichtige Unix-Datenstrukturen

- Pro Seite: Page-Table-Entry (PTE)
  - Valid (im Speicher)
  - Page Frame Number
  - Dirty
  - Schutzbits...

- Pro Page Frame: Eintrag in Core-Map
  - Zustand (belegt, frei, gesperrt,...)
  - Wohin auslagern... (d.h. was steht drin)

Wichtige Unix-Datenstrukturen

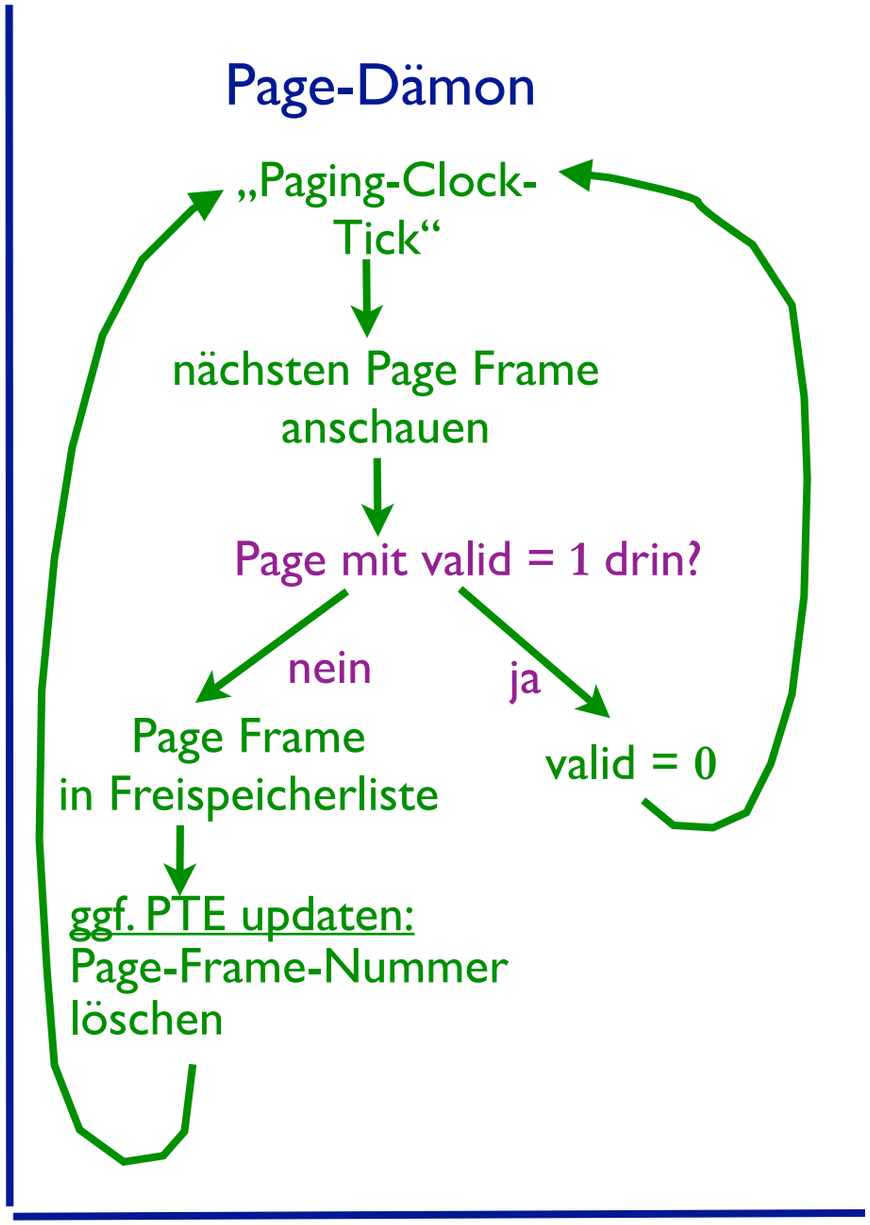
- Pro Seite: Page-Table-Entry (PTE)
  - Valid (im Speicher)
  - Page Frame Number
  - Dirty
  - Schutzbits...

- Pro Page Frame: Eintrag in Core-Map
  - Zustand (belegt, frei, gesperrt,...)
  - Wohin auslagern... (d.h. was steht drin)

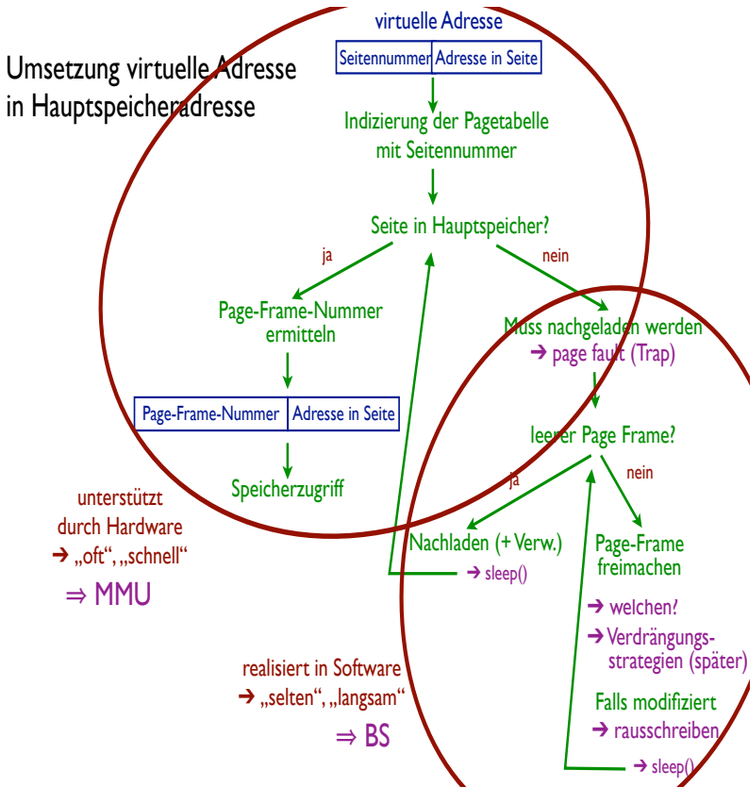
⇒ kein Referenz-Bit

⇒ wird über Valid-Bit simuliert

Seitenverdrängung in Unix



Seitenverdrängung in Unix



Arbeit

valid-Bit = 0?
→ Page Fault → Trap

Kern

Page-Fault-Handler

Page-Frame-Nummer in PTE?

ja: valid = 1

nein: Page Frame mit richtigem Inhalt in Freispeicherliste?

ja: Reclaim

PTE updaten:
valid = 1
Page-Frame-Nummer eintragen

nein: freien Page Frame suchen

(+ ggf. alten Inhalt vorher rausschreiben)

Einlesen anstoßen (→ Platte)

Warten auf Ergebnis → sleep()

ok? (Page kann schon wieder raus sein!)

ja

Page-Dämon

„Paging-Clock-Tick"

nächsten Page Frame anschauen

Page mit valid = 1 drin?

nein: Page Frame in Freispeicherliste

ggf. PTE updaten:
Page-Frame-Nummer löschen

ja: valid = 0

Seitenverdrängung in Unix

Arbeit

valid-Bit = 0?
→ Page Fault → Trap

Kern

Page-Fault-Handler

Page-Frame-Nummer in PTE?

ja: valid = 1

nein: Page Frame mit richtigem Inhalt in Freispeicherliste?

ja: Reclaim

PTE updaten:
valid = 1
Page-Frame-Nummer eintragen

nein: freien Page Frame suchen (+ ggf. alten Inhalt vorher rausschreiben)

Einlesen anstoßen (→ Platte)

Warten auf Ergebnis → sleep()

ok? (Page kann schon wieder raus sein!)

ja

Page-Dämon

„Paging-Clock-Tick“

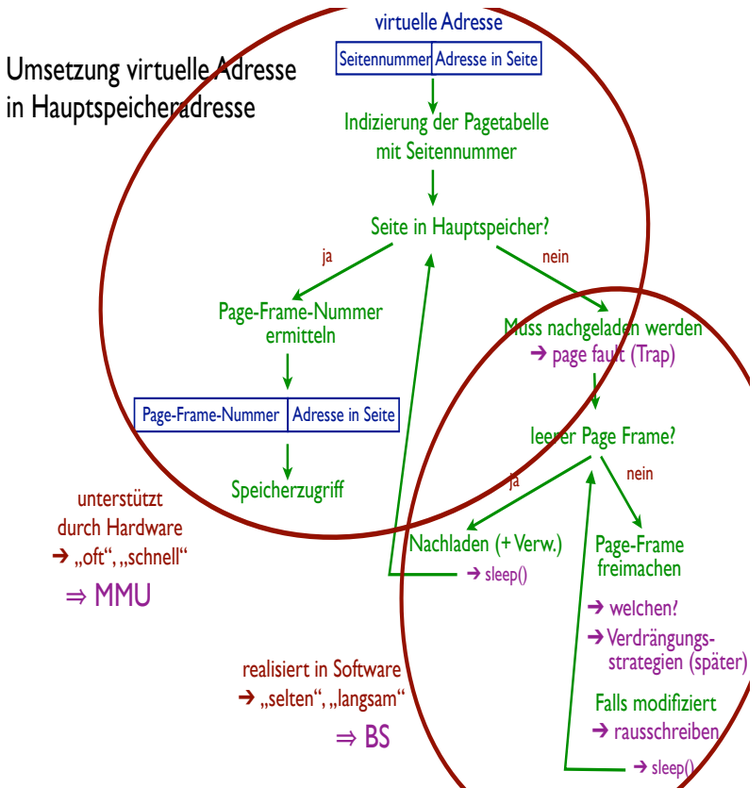
nächsten Page Frame anschauen

Page mit valid = 1 drin?

nein: Page Frame in Freispeicherliste

ggf. PTE updaten:
Page-Frame-Nummer löschen

ja: valid = 0



Seitenverdrängung in Unix

Arbeit

valid-Bit = 0?
→ Page Fault → Trap

Kern

Page-Fault-Handler

Page-Frame-Nummer in PTE?

ja → valid = 1

nein → Page Frame mit richtigem Inhalt in Freispeicherliste?

ja → Reclaim

PTE updaten:
valid = 1
Page-Frame-Nummer eintragen

nein → freien Page Frame suchen (+ ggf. alten Inhalt vorher rausschreiben)

Einlesen anstoßen (→ Platte)

Warten auf Ergebnis → sleep()

ok? (Page kann schon wieder raus sein!)

ja

Page-Dämon

„Paging-Clock-Tick“

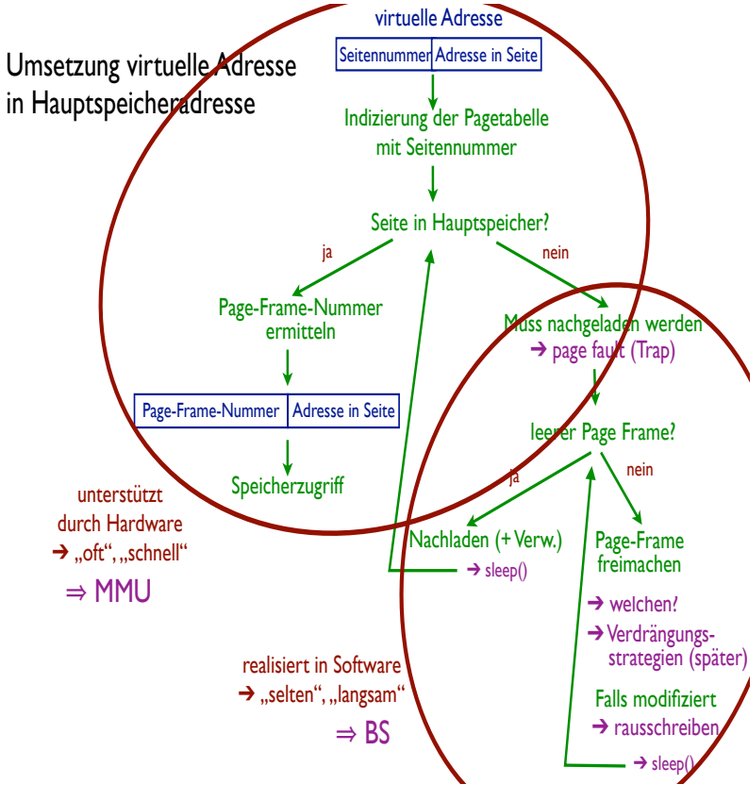
nächsten Page Frame anschauen

Page mit valid = 1 drin?

nein → Page Frame in Freispeicherliste

ggf. PTE updaten:
Page-Frame-Nummer löschen

ja → valid = 0



Seitenverdrängung in Unix

Arbeit

valid-Bit = 0?
→ Page Fault → Trap

Kern

Page-Fault-Handler

Page-Frame-Nummer in PTE?

ja: valid = 1

nein: Page Frame mit richtigem Inhalt in Freispeicherliste?

ja: Reclaim

PTE updaten:
valid = 1
Page-Frame-Nummer eintragen

nein: freien Page Frame suchen (+ ggf. alten Inhalt vorher rausschreiben)

Einlesen anstoßen (→ Platte)

Warten auf Ergebnis → sleep()

ok? (Page kann schon wieder raus sein!)

ja

Page-Dämon

„Paging-Clock-Tick“

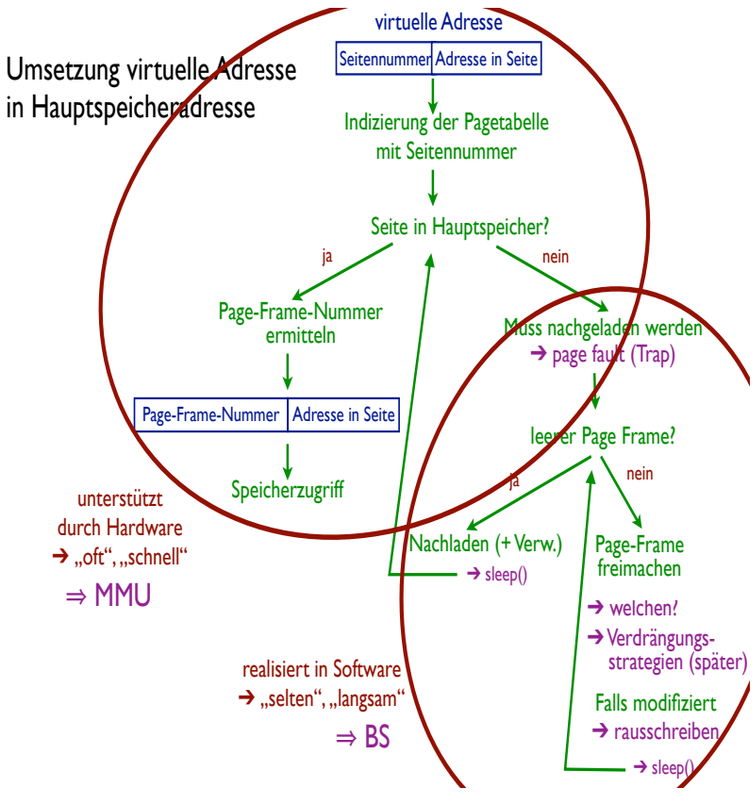
nächsten Page Frame anschauen

Page mit valid = 1 drin?

nein: Page Frame in Freispeicherliste

ggf. PTE updaten:
Page-Frame-Nummer löschen

ja: valid = 0



Fragen – Teil 2

- An welchen der drei zuvor vorgestellten Verdrängungsalgorithmen ist NRU (Not-Recently-Used) angelehnt?
- Wie arbeitet der *Clock-Hand-Algorithmus*?
- Was passiert, wenn die Umlaufzeit des Zeigers beim Clock-Hand-Algorithmus zu groß bzw. zu klein gewählt wird? Wie kann ein zweiter Zeiger den Algorithmus verbessern?

Teil 3:
Weitere Aspekte des Pagings

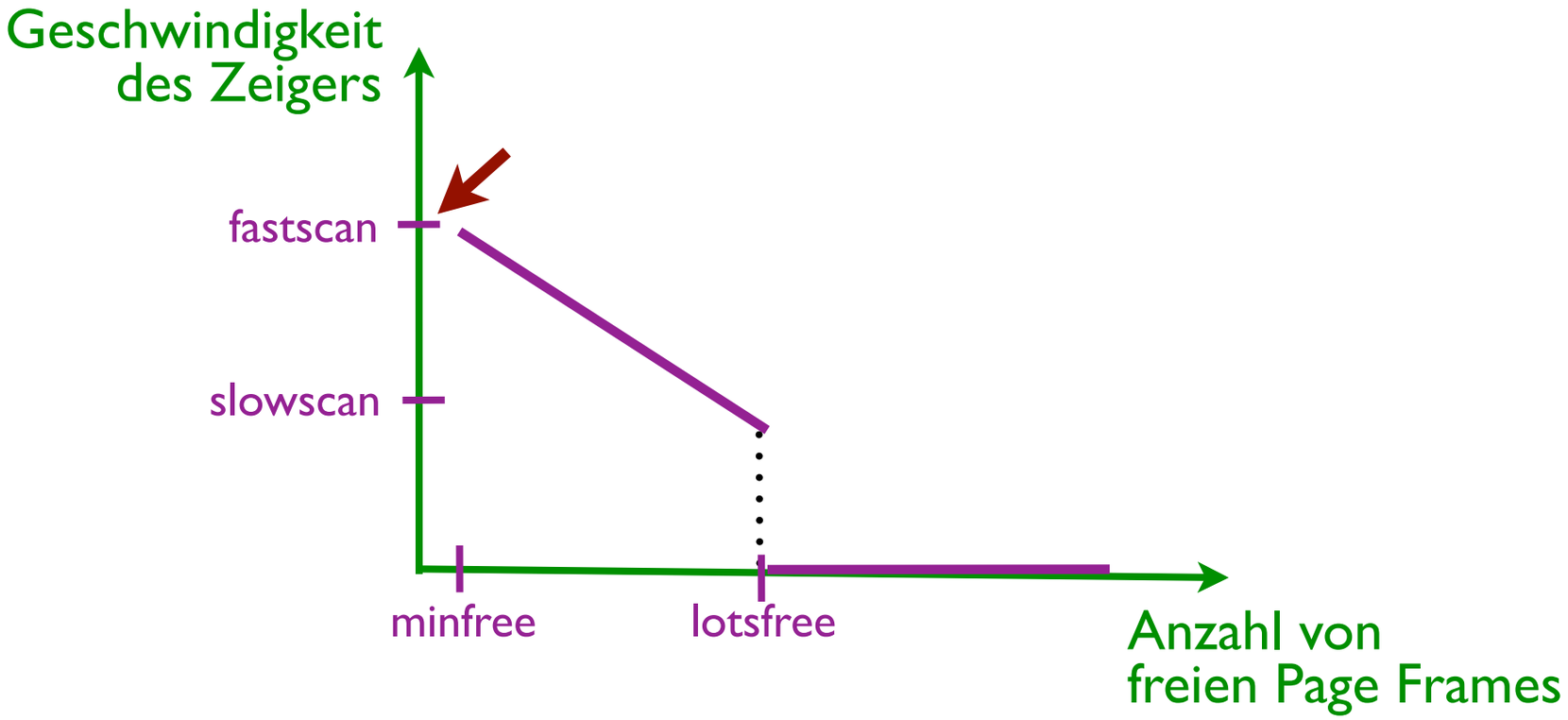
Swapping

- Passen Working Sets der „aktiven“ Prozesse nicht vollständig in Speicher

  ⇒ hohe Page-Fault-Rate (ständiges Nachladen erforderlich)

  ⇒ „Seiten-Flattern“ (Thrashing)

  ⇒ Durchsatz sinkt rapide (Platte kommt nicht mit)



Swapping

- Passen Working Sets der „aktiven“ Prozesse nicht vollständig in Speicher

  ⇒ hohe Page-Fault-Rate (ständiges Nachladen erforderlich)

  ⇒ „Seiten-Flattern“ (Thrashing)

  ⇒ Durchsatz sinkt rapide (Platte kommt nicht mit)

- Abhilfe:
  - Anzahl der aktiven Prozesse reduzieren
  - Davon belegte Page Frames freigeben

⇒ Swapping

Swapping

- Passen Working Sets der „aktiven“ Prozesse nicht vollständig in Speicher

  ⇒ hohe Page-Fault-Rate (ständiges Nachladen erforderlich)
  ⇒ „Seiten-Flattern“ (Thrashing)
  ⇒ Durchsatz sinkt rapide (Platte kommt nicht mit)

- Abhilfe:
  - Anzahl der aktiven Prozesse reduzieren
  - Davon belegte Page Frames freigeben

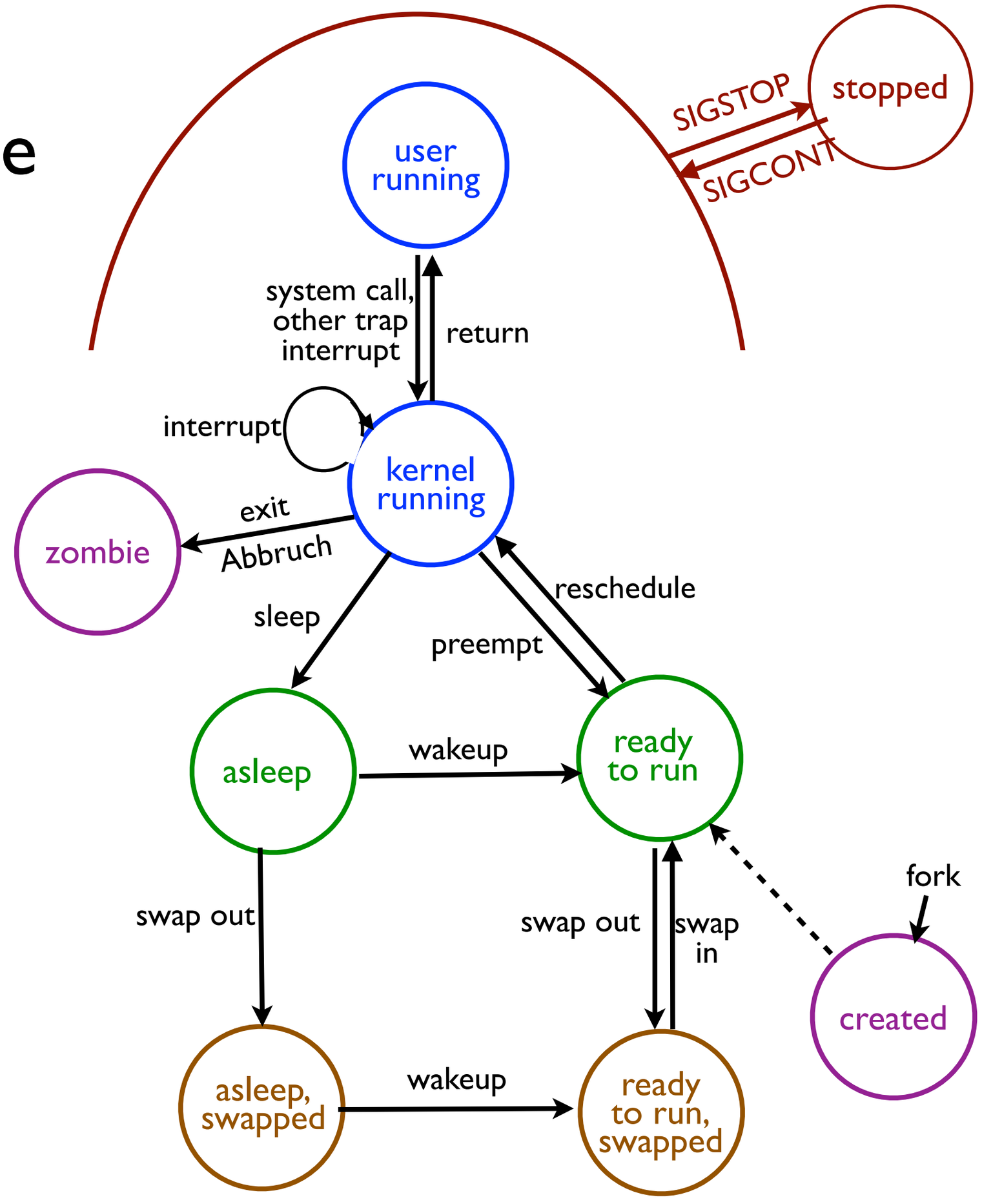
  ⇒ Swapping

- Swapping-Strategie:
  - ggf. Prozesse auswählen, die länger inaktiv sind
  - (wenige) große Prozesse auswählen, schafft viel Platz
  - Aber: Interaktive Prozesse erfordern erträgliches Antwortzeitverhalten

  ⇒ nach gewisser Zeit Tausch erforderlich

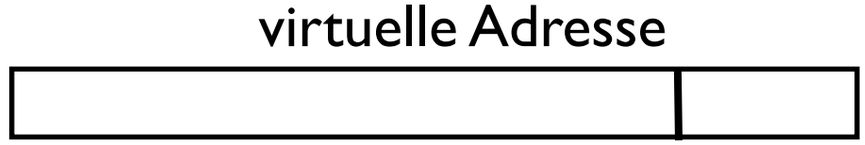
Prozesszustände

(vereinfacht)



Weitere Aspekte des Pagings

- Seiten-Größe:
  - Zu klein:
    - Zuviele Plattenteraktionen (langsam)
    - Große Page-Tabellen
  - Zu groß:
    - Hohe interne Fragmentierung
    - Umfasst u.U. mehr Infos als im Working Set



⇒ Typische Größe: 4–16 KiB

- Zeitpunkt des Nachladens von Seiten
  - On demand: Sobald Page Fault auftritt
  - Prefetching: Auf Verdacht Folgeseite(n) mitladen
    ⇒ u.U. wenig zusätzlicher Aufwand
  - Prepaging: (nicht Unix)
    Nach Unterbrechung durch Swapping alle Seiten des aktuellen Working Sets sofort laden
    ⇒ weniger Page Faults beim Arbeiten

- Residente Seiten
  - Bestimmte Seiten sollten nicht verdrängt werden können (z.B. wichtige Betriebssysteminfos, gerade zu füllende Ein-/Ausgabepuffer, ...)

- Page-Tabelle im Hauptspeicher?

  ⇒ zusätzlicher Speicherzugriff

  ⇒ zu langsam

  - Daher: Adressumsetzung über speziellen Cache
  - Für gesamte Page-Tabelle?

  ⇒ - teuer
    - neu laden bei Prozesswechsel

- Page-Tabelle im Hauptspeicher?

  ⇒ zusätzlicher Speicherzugriff

  ⇒ zu langsam

  - Daher: Adressumsetzung über speziellen Cache
  - Für gesamte Page-Tabelle?

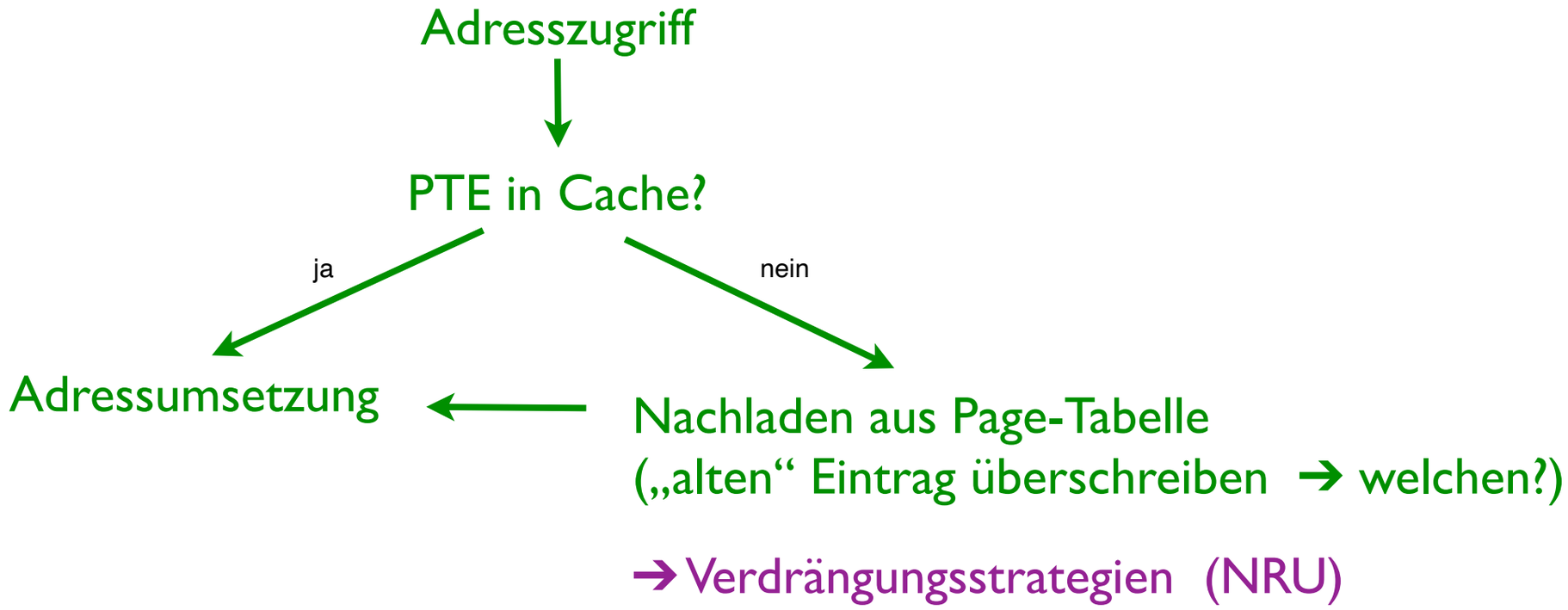
    ⇒ • teuer

      • neu laden bei Prozesswechsel

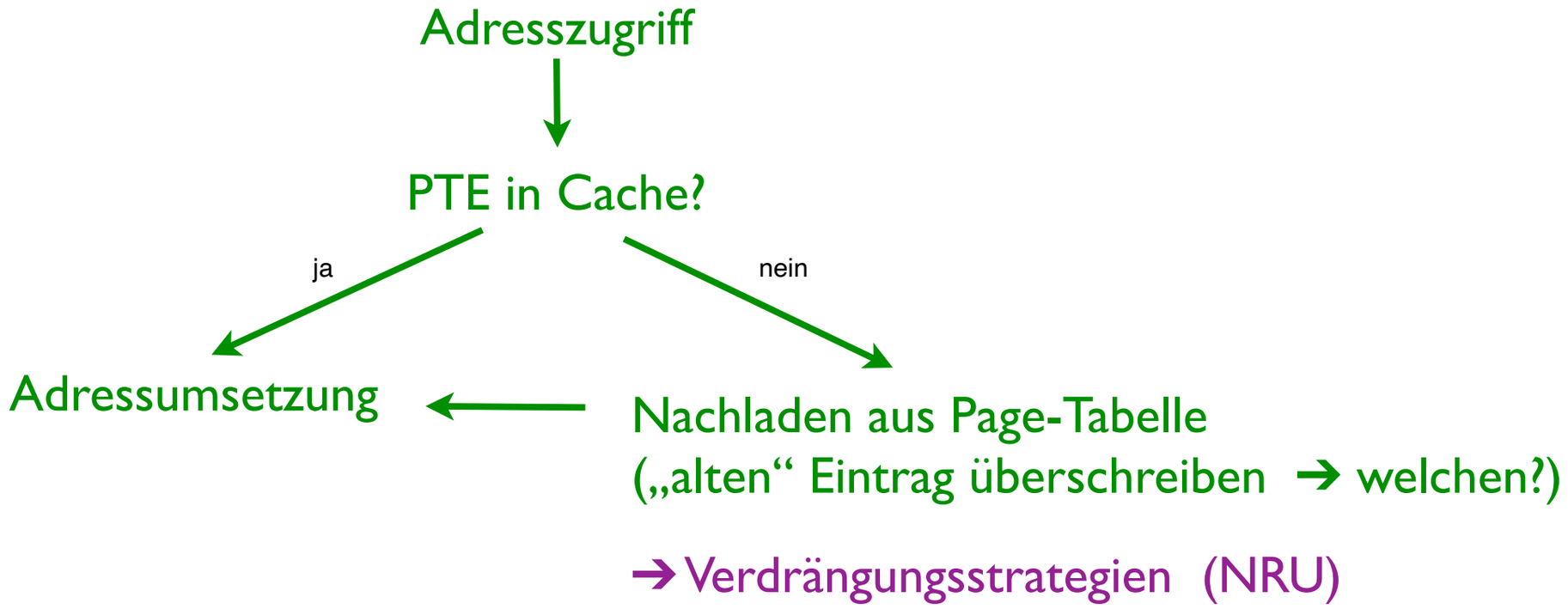
  - Andererseits: Lokalitätsprinzip von Prozessen

    ⇒ nur einige Seiten z.Zt. in Gebrauch

- Cache enthält nur die z.Zt. „benötigten“ PTEs des Prozesses

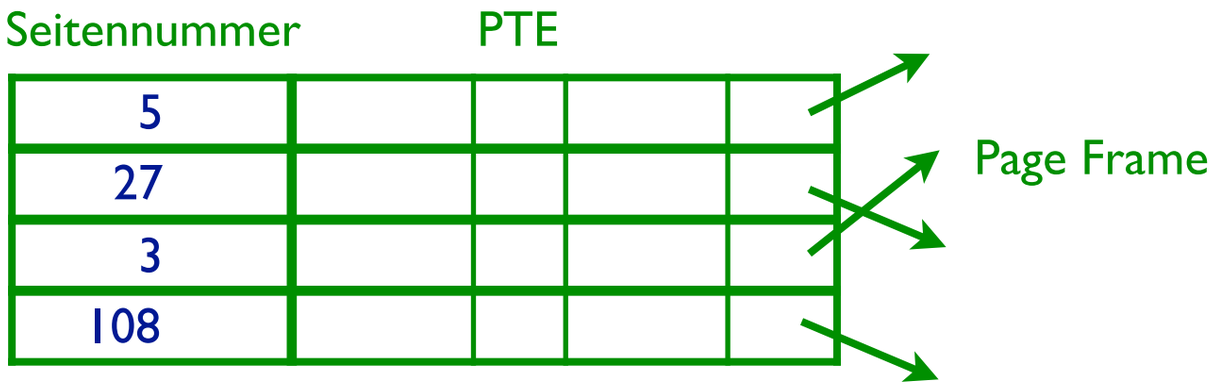


- Cache enthält nur die z.Zt. „benötigten“ PTEs des Prozesses

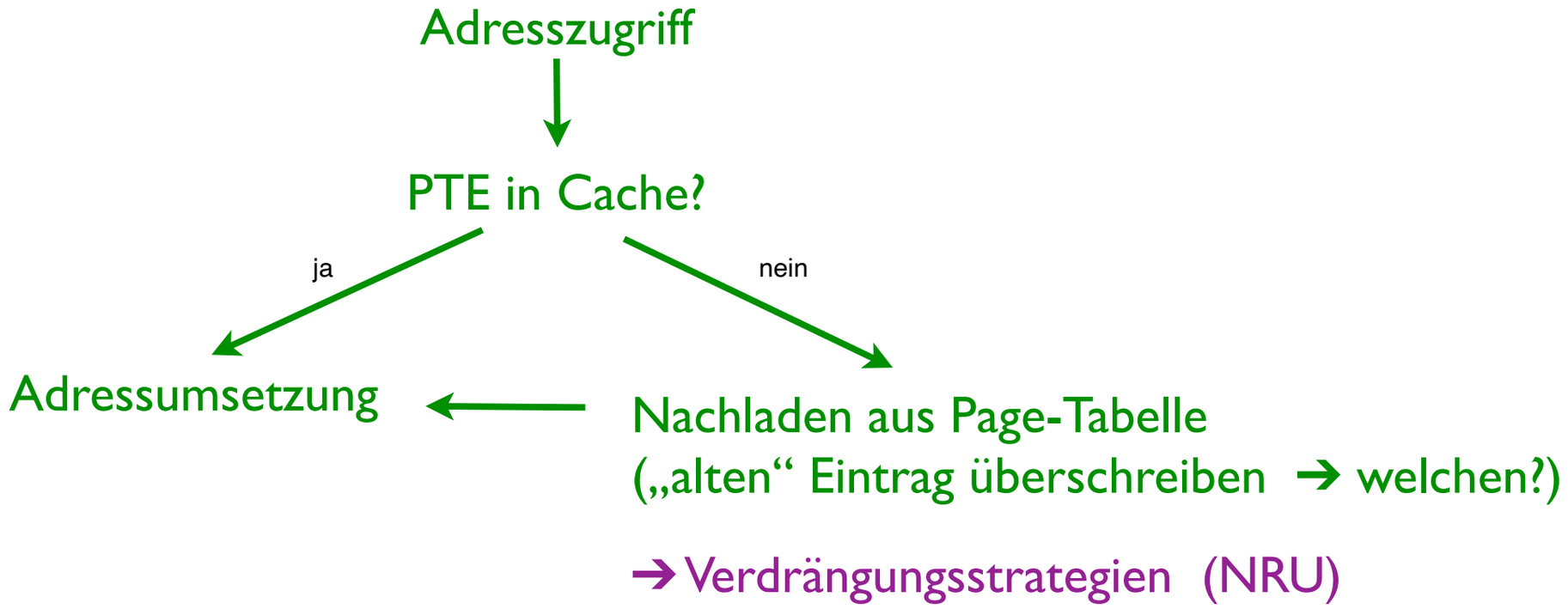


- Keine Indizierung über Seitennummern möglich

  ⇒ Cache ist Assoziativspeicher (paralleler Zugriff auf alle Einträge)

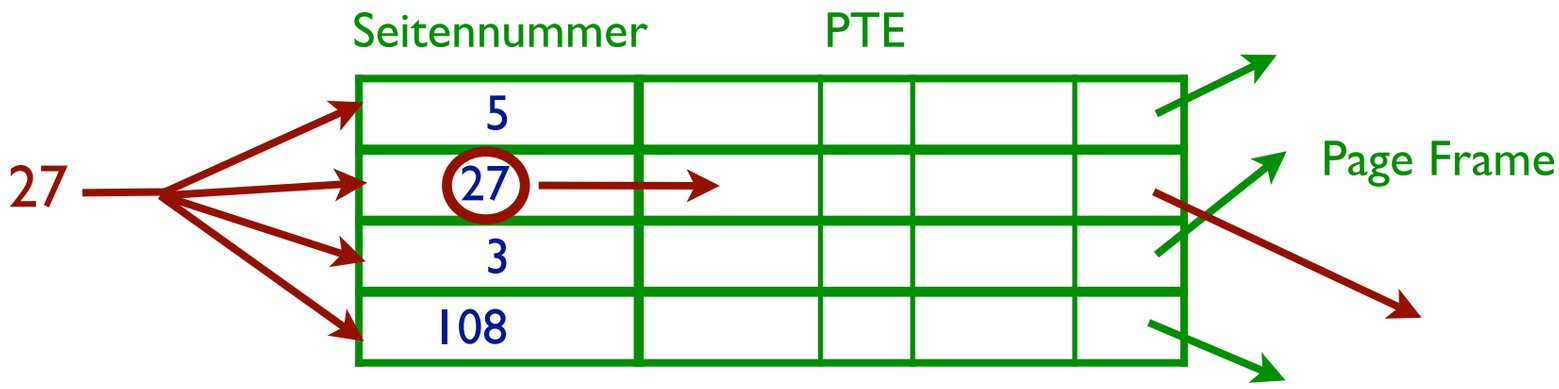


- Cache enthält nur die z.Zt. „benötigten“ PTEs des Prozesses



- Keine Indizierung über Seitennummern möglich

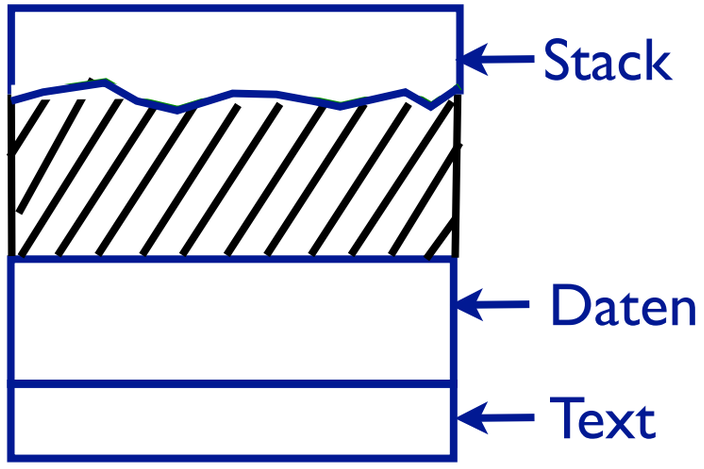
  ⇒ Cache ist Assoziativspeicher (paralleler Zugriff auf alle Einträge)



  ⇒ TLB (Translation Lookaside Buffer)

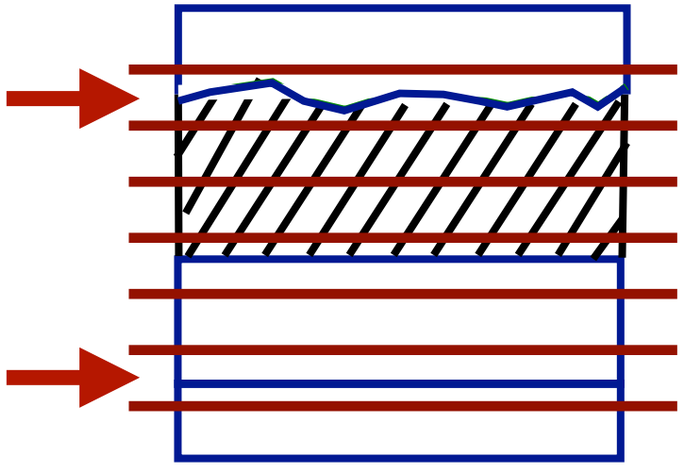
Paging vs. Segmentierung

- Feste Seiten-Größen ⇒ keine logische Aufteilung des Adressraums



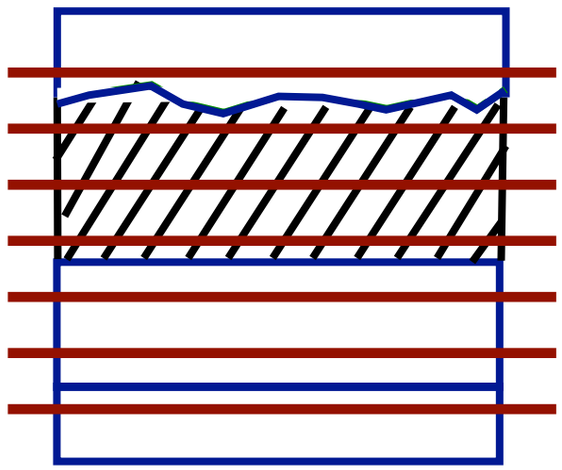
Paging vs. Segmentierung

- Feste Seiten-Größen ⇒ keine logische Aufteilung des Adressraums

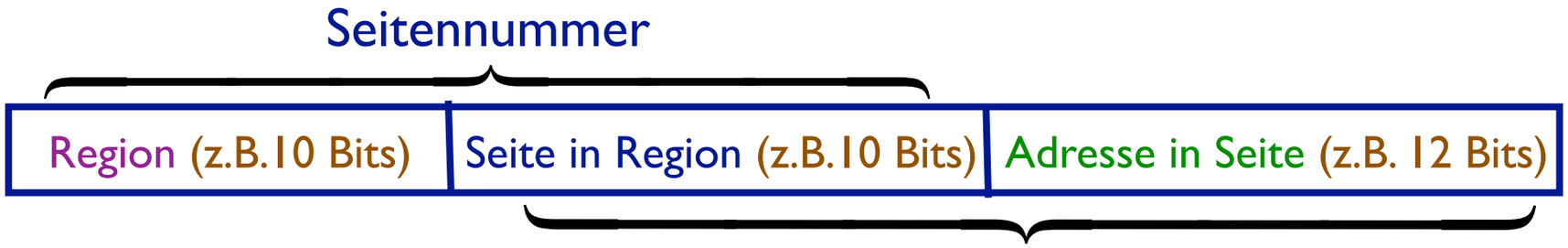


Paging vs. Segmentierung

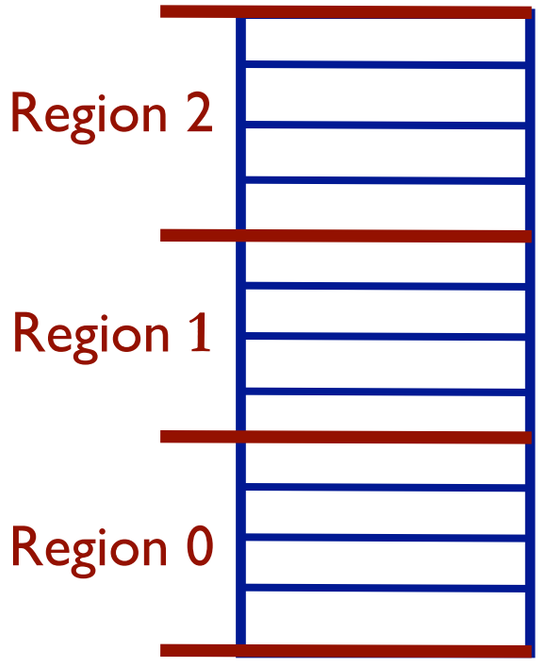
- Feste Seiten-Größen ⇒ keine logische Aufteilung des Adressraums

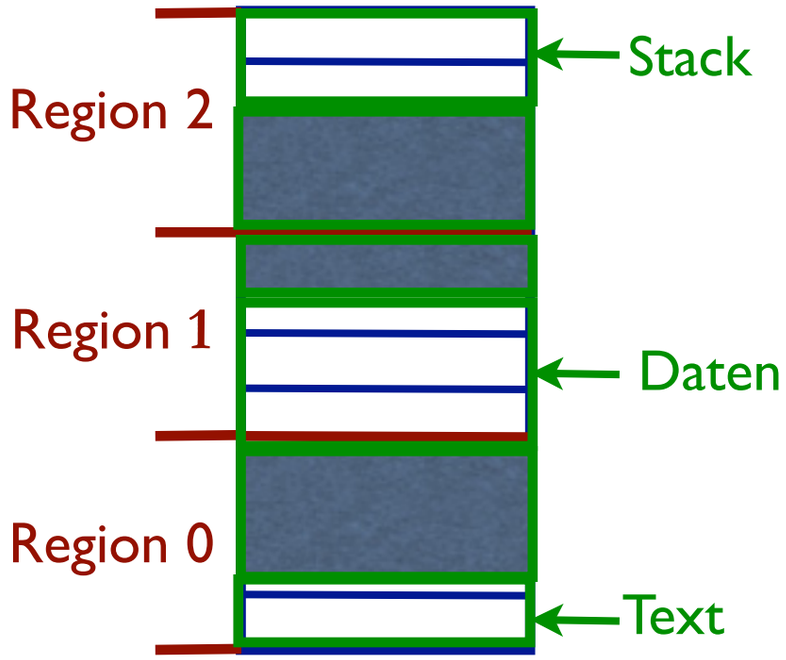


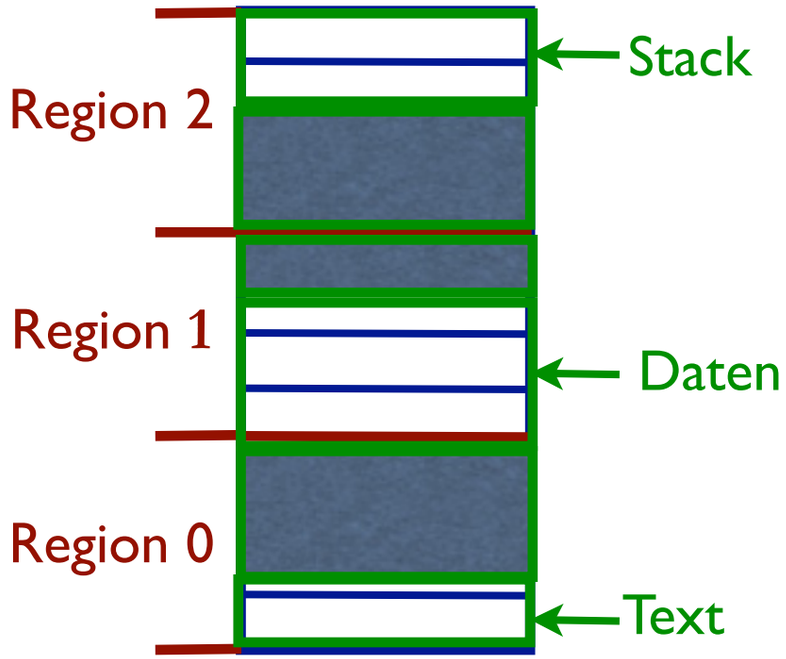
- Heute häufig mehrstufiges Paging (3-teilige Adressen) ⇒ Regionen



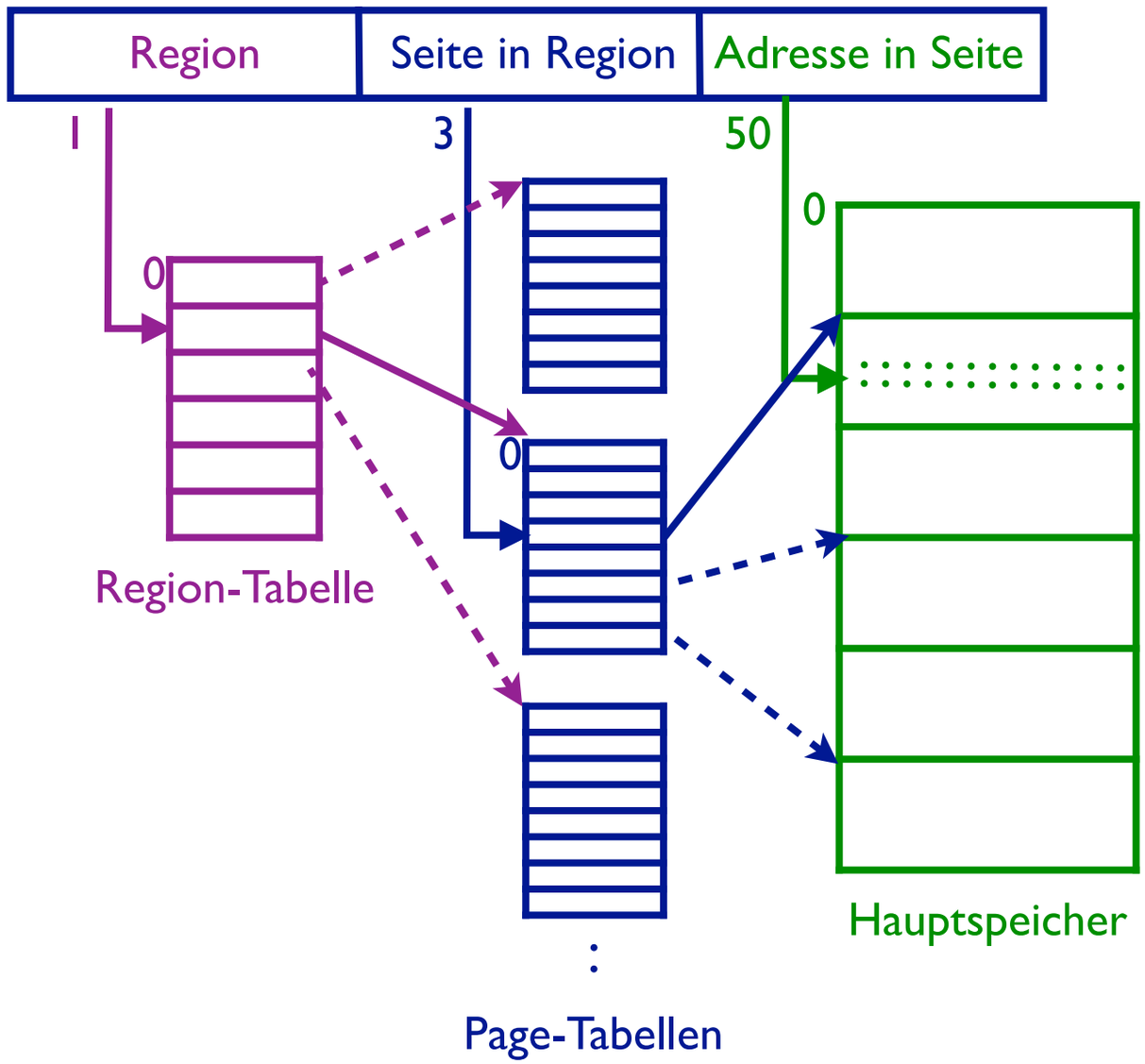
- Virtueller Adressraum wird in Regionen unterteilt (≙ Segment)
  ⇒ logische Gliederung des Adressraums
- Region wird in Seiten unterteilt
  ⇒ feste Größe der Speicherzuteilung
  ⇒ eigene Page-Tabelle pro Region


Region 2
Region 1
Region 0


Region 2
Region 1
Region 0
Stack
Daten
Text



- Adressumsetzung:



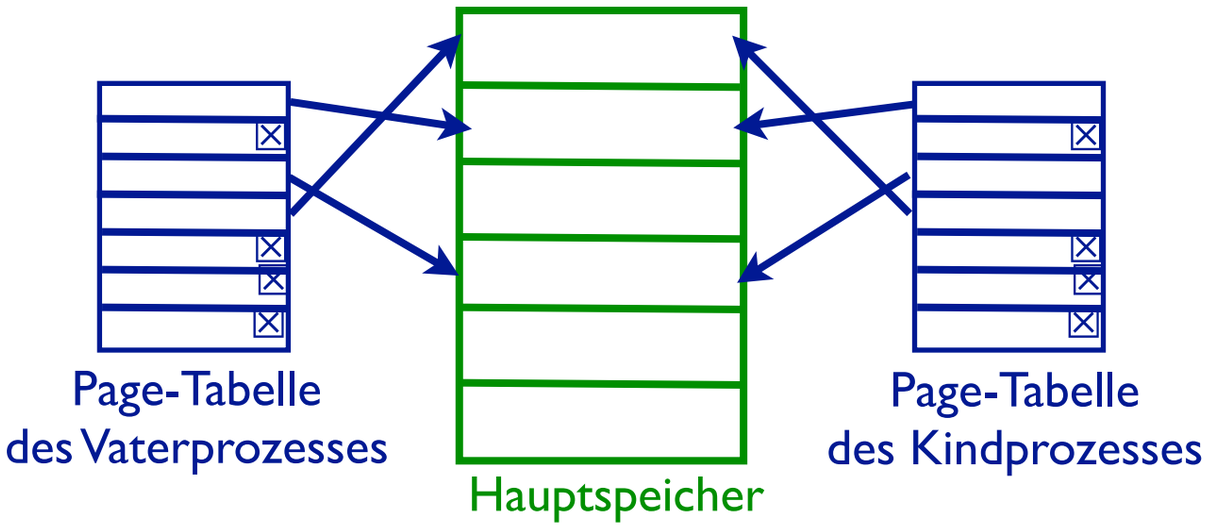
Virtuelle Adressräume und Prozesserzeugung

- fork() erzeugt (fast) Kopie des Vaterprozesses (inkl. Adressraum)

Virtuelle Adressräume und Prozesserzeugung

- fork() erzeugt (fast) Kopie des Vaterprozesses (inkl. Adressraum)
- (Zunächst) keine Kopie der Seiten erforderlich

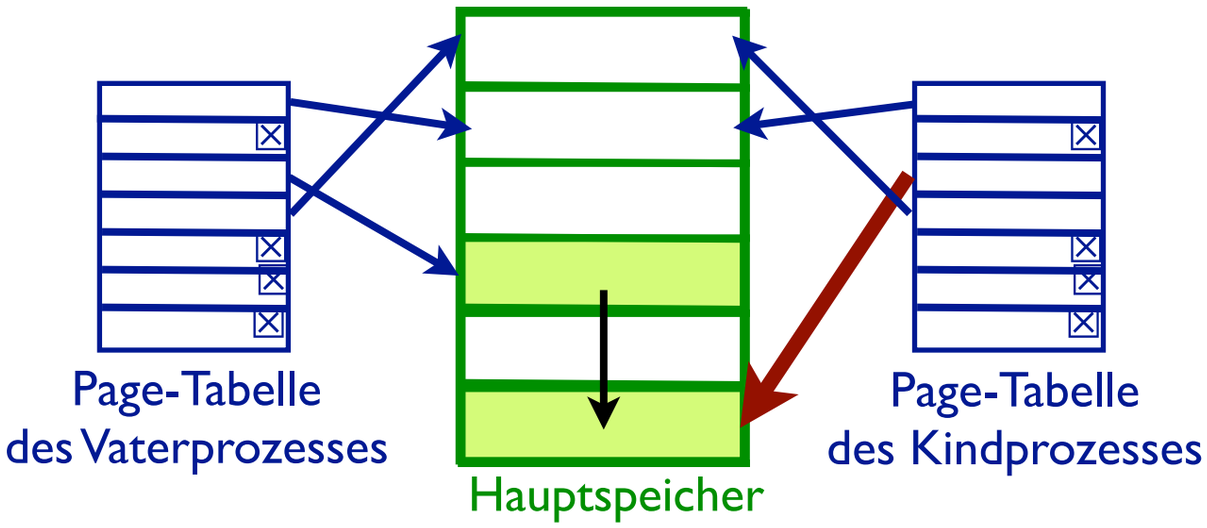
  ⇒ Kopie der Page-Tabelle ausreichend



Virtuelle Adressräume und Prozesserzeugung

- fork() erzeugt (fast) Kopie des Vaterprozesses (inkl. Adressraum)
- (Zunächst) keine Kopie der Seiten erforderlich

  ⇒ Kopie der Page-Tabelle ausreichend

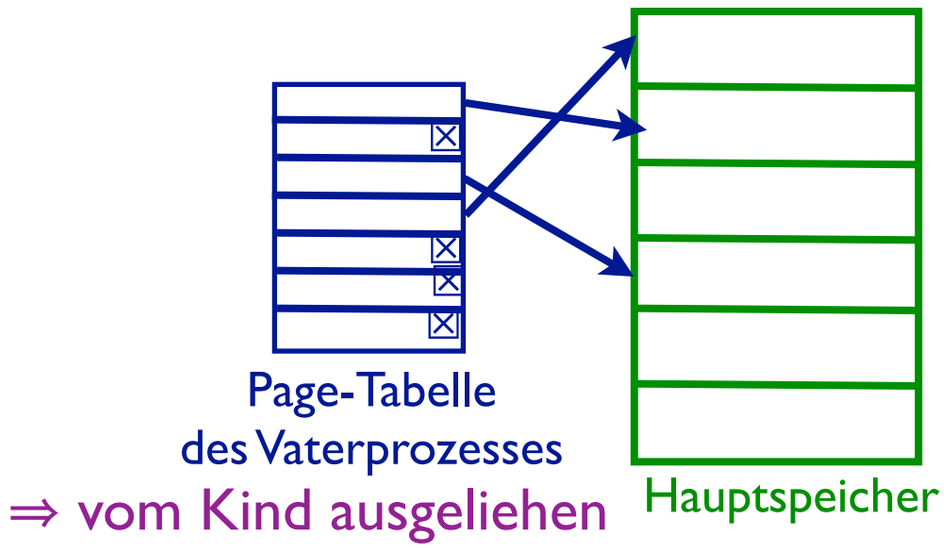


- Sobald auf eine Seite geschrieben wird ⇒ kopieren

  ⇒ Copy-on-write (Page Fault ⇒ Betriebssystem kopiert)

Virtuelle Adressräume und Prozesserzeugung

- fork() erzeugt (fast) Kopie des Vaterprozesses (inkl. Adressraum)



- Variante: vfork()
  - Auch Page-Tabelle wird nicht kopiert
  - Vater legt sich schlafen, bis Kind exec() oder exit() aufruft
    ⇒ Kind leiht Adressraum aus, kann ihn auch verändern

Fragen – Teil 3

- Was ist *Swapping*? Warum wenden auch Paging-Systeme dieses Verfahren an bzw. unter welcher Bedingung?
- Wie kann man die Vorteile von *Paging* und *Segmentierung* kombinieren?
- Wozu bzw. wo wird bei der Speicherverwaltung häufig ein *Assoziativspeicher* eingesetzt?

Zusammenfassung

- Seitenverdrängung aus dem Hauptspeicher:
  - FIFO, LFU, LRU
  - NRU, Clock-Hand-Algorithmus
  - Seitenverdrängung in Unix
- Swapping
- Kombination von Paging und Segmentierung
- Virtuelle Adressräume und Prozesserzeugung

Speicherverwaltung 2 – Fragen

1. Warum ist ein perfekter Algorithmus zur Verdrängung von Pages aus dem Hauptspeicher nicht realisierbar?

2. Wie arbeiten die folgenden Algorithmen in etwa:

   a) FIFO (First-In-First-Out),
   b) LFU (Least-Frequently-Used),
   c) LRU (Least-Recently-Used)?

3. An welchen der drei zuvor vorgestellten Verdrängungsalgorithmen ist NRU (Not-Recently-Used) angelehnt?

4. Wie arbeitet der *Clock-Hand-Algorithmus*?

5. Was passiert, wenn die Umlaufzeit des Zeigers beim Clock-Hand-Algorithmus zu groß bzw. zu klein gewählt wird? Wie kann ein zweiter Zeiger den Algorithmus verbessern?

6. Was ist *Swapping*? Warum wenden auch Paging-Systeme dieses Verfahren an bzw. unter welcher Bedingung?

7. Wie kann man die Vorteile von *Paging* und *Segmentierung* kombinieren?

8. Wozu bzw. wo wird bei der Speicherverwaltung häufig ein *Assoziativspeicher* eingesetzt?