UNIVERSITÄT BERN

2405 Betriebssysteme
VIII. Virtueller Speicher

Thomas Staub, Markus Anwander
Universität Bern

Inhalt

1. Virtueller Speicher und Demand Paging
   1. Virtueller Speicher
   2. Demand Paging
      1. Ablauf
      2. Leistung
      3. Beispiel
   3. Copy-on-Write
2. Seitenersetzung
   1. FIFO
      1. FIFO-Seitenersetzung
      2. Belady`s Anomalie
   2. Optimaler Algorithmus
   3. Least Recently Used (LRU)
      1. Zähler und Stacks
      2. Referenzbit
      3. Second Chance
      4. Clock-Algorithmus
      5. Erweiterung Second Chance
   4. Zählalgorithmen
   5. Optimierungen mit Page Buffering
3. Allokation von Speicherkacheln
   1. Allokationsalgorithmen
   2. Globale und lokale Allokation
4. Thrashing
   1. Lokalität
   2. Working-Set-Modell
   3. Implementierung des Working-Set-Modells
   4. Page-Fault-Frequency-Strategie
5. Spezielle Aspekte
   1. Pre-Paging
   2. Dämon-Paging
   3. Seitengrösse
   4. Einfluss der Programmstruktur auf Seitenfehler
   5. Realzeitverarbeitung

1.1 Virtueller Speicher

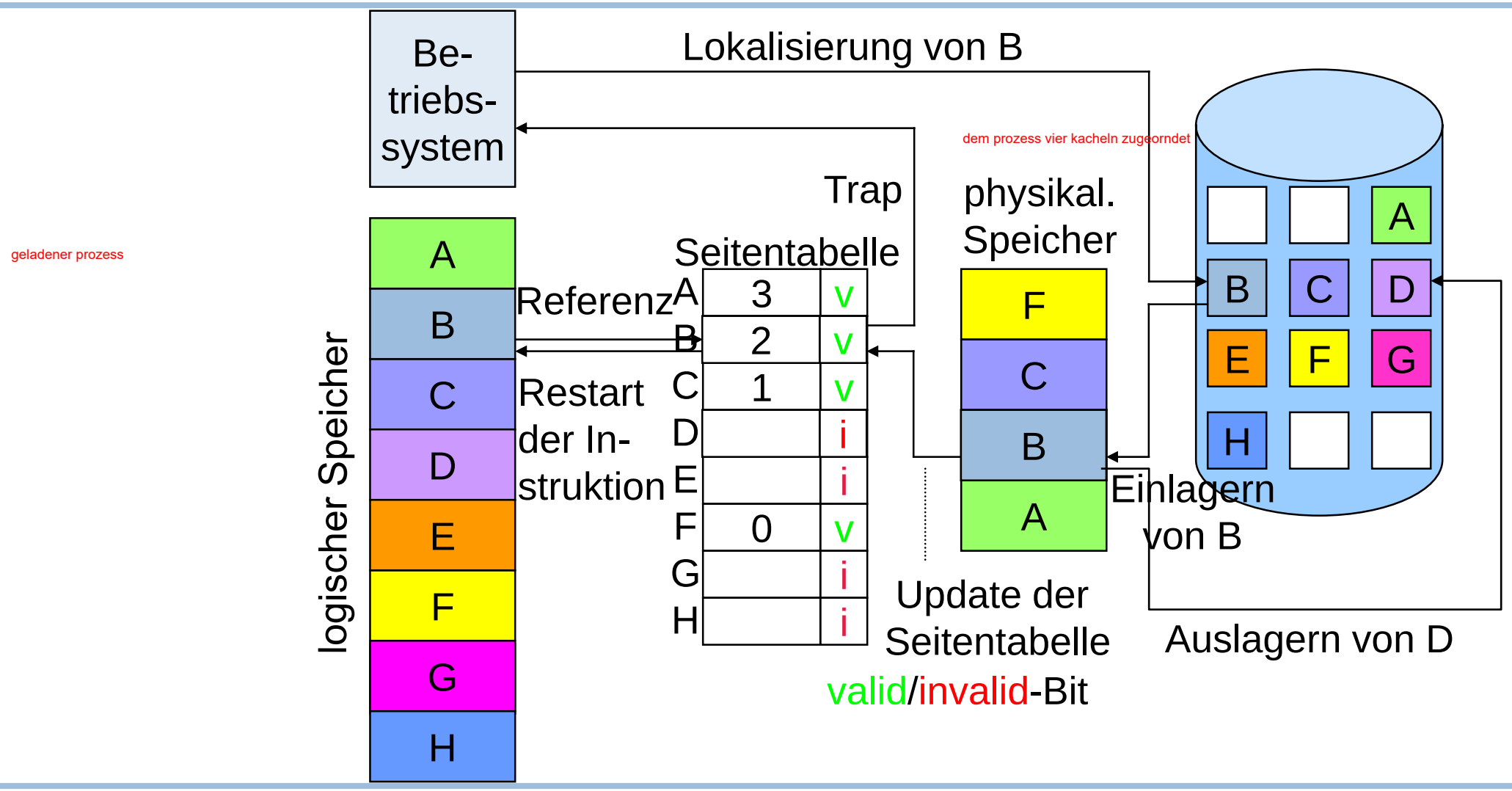
- Nur die benötigten Teile eines Programms befinden sich im Hauptspeicher.
  Alle anderen Teile befinden sich im Sekundärspeicher.
- Dadurch steht den Programmierern ein extrem grosser Speicherbereich zur Verfügung.
- Virtueller Speicher erlaubt Teilen von Speicher (Sharing).
- Realisierung von virtuellem Speicher durch ***Demand Paging***

1.2 Demand Paging

UNIVERSITÄT BERN

- Pager lagert nur ***benötigte*** Seiten eines Prozesses ein.
- Vorteile
  - weniger Ein-/Ausgabe-Tätigkeit
  - geringerer Speicherbedarf eines Prozesses
  - Mehr Prozesse können sich gleichzeitig im Hauptspeicher befinden.
  - kürzere Antwortzeit
- Probleme
  - Seitenersetzungsstrategie
  - Allokation von Speicherkacheln, um minimale Anforderungen der Prozesse zu erfüllen

1.2.1 Ablauf von Demand Paging



1.2.2 Leistung von Demand Paging

$t_{VS} = (1-p) \cdot t_{HS} + p \cdot t_{SF}$

t_{VS} : Zugriffszeit auf virtuellen Speicher
t_{HS} : mittlere Zugriffszeit auf Hauptspeicher
t_{SF} : mittlere Verzögerung zur Behandlung eines Seitenfehlers

$t_{SF} = t_{Ubr} + t_{Aus} + t_{Ein} + t_{Akt} + t_{Wdh} \approx 2\, t_{Aus/Ein}$

t_{Ubr} : Ausführungszeit der Unterbrechungsroutine
t_{Aus} : Zeit für Auswahl und Auslagern einer Seite
t_{Ein} : Zeit für Einlagern der referenzierten Seite
t_{Akt} : Zeit zur Aktualisierung der Seitentabelle
t_{Wdh} : Zeit zur Wiederholung der unterbrochenen Instruktion

1.2.3 Beispiel

t_{HS} = 70 ns

$t_{Aus/Ein}$ = 10 ms

$t_{VS} = (1-p) \cdot t_{HS} + p \cdot t_{SF} < 1.1 \cdot t_{HS}$

(Zugriff auf virtuellen Speicher soll maximal um 10 % höher sein als Hauptspeicherzugriff)

⇔ $p \cdot t_{SF} - p \cdot t_{HS} < 0.1 \cdot t_{HS}$

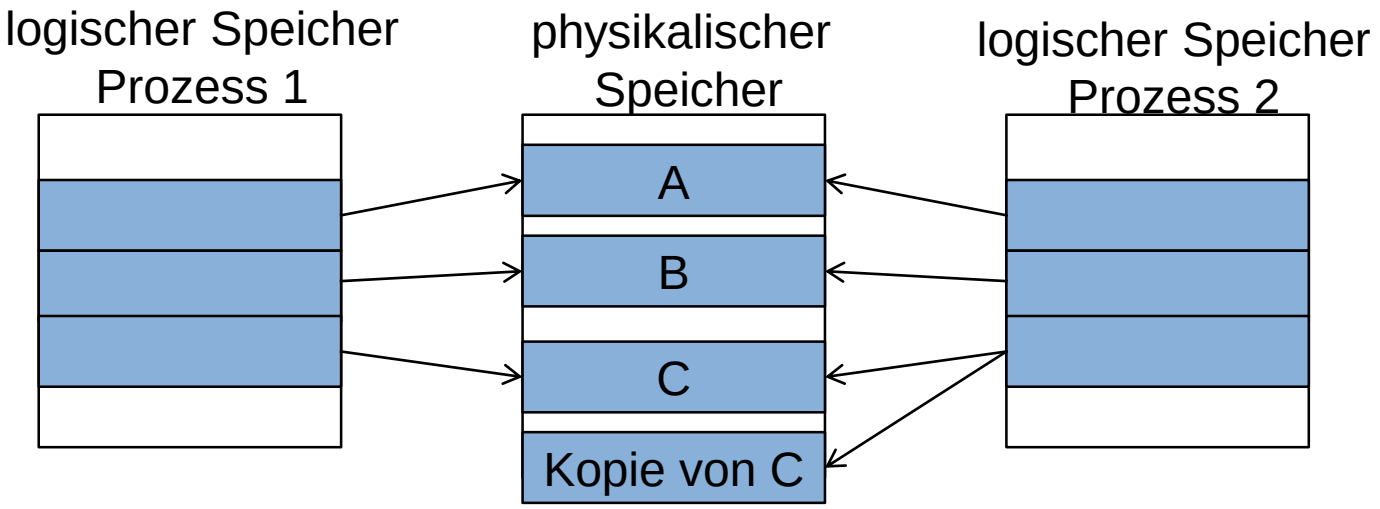
⇔ $p \cdot (t_{SF} - t_{HS}) < 0.1 \cdot t_{HS}$

⇔ $p \cdot < 0.1 \cdot t_{HS} / (t_{SF} - t_{HS}) \approx 7 \text{ ns} / 20 \text{ ms} = 3.5 \cdot 10^{-7}$

d.h. 1 Seitenfehler auf ca. 2‘857‘000 Speicherzugriffe

1.3 Copy-on-Write

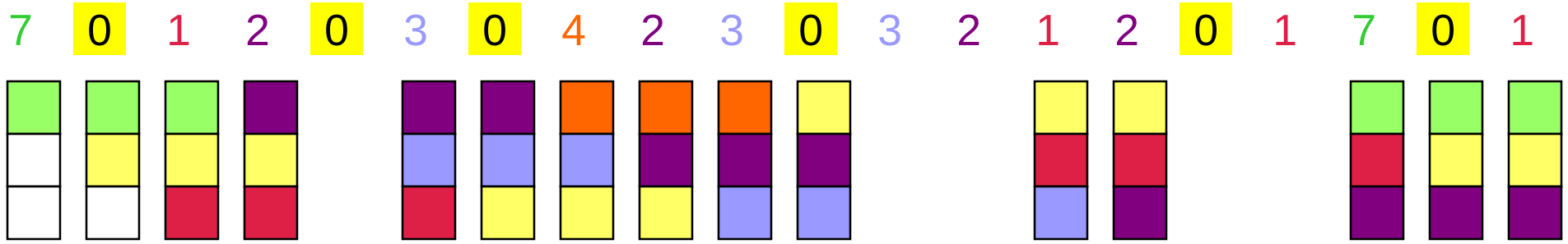
- Systemaufruf **`fork()`** kann Seitenanforderung zu Prozessstart umgehen, indem für ein Kindprozess ein Duplikat des Elternprozess erstellt wird, vgl. Teilen von Seiten
- Copy-on-write
  - Anfangs Teilen von Seiten
  - Markieren von Seiten als copy-on-write
  - Falls ein Prozess eine solche Seite beschreibt, wird Kopie angelegt.



2. Seitenersetzung

- Problem
  - Zugriff auf Seite, aber keine freie Hauptspeicherkachel
- Lösung
  - Auslagern einer (nicht benötigten) Seite vom Hauptspeicher in den Sekundärspeicher abhängig vom Modify-Bit der Seitentabelle
  - Seitenersetzungsalgorithmus sollte Seitenfehler minimieren.
  - Seitenersetzung erlaubt Ausführen von grossen Programmen auf Rechnern mit kleinem Speicher.

2.1.1 FIFO-Seitenersetzung



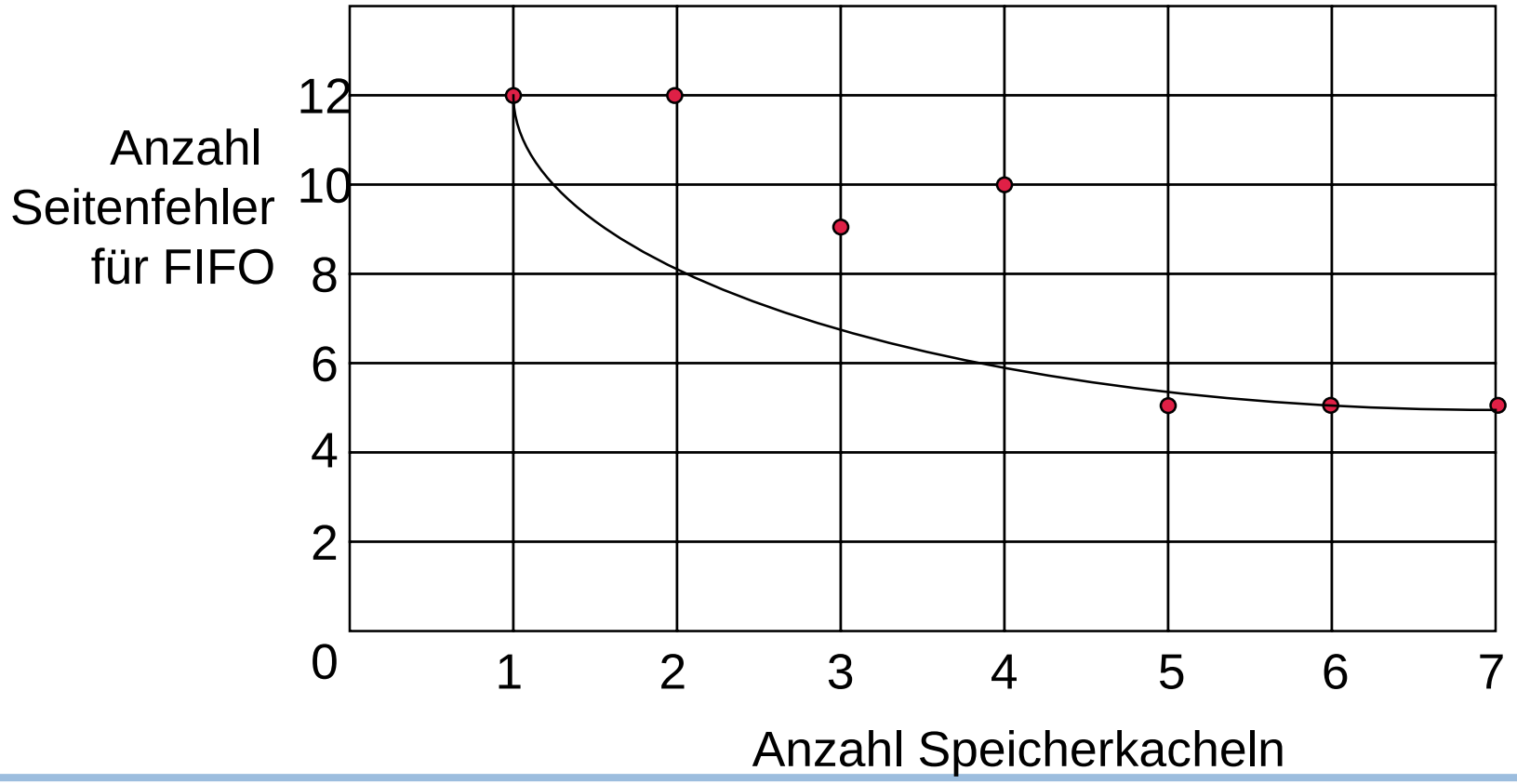
- First In First Out, d.h. älteste Seite wird ausgelagert
- einfach zu programmieren
- Häufiges Ein- und Auslagern aktiver Seiten → schlechte Leistung
- im Beispiel: 15 Seitenfehler bei 20 Referenzen

2.1.2 Belady`s Anomalie

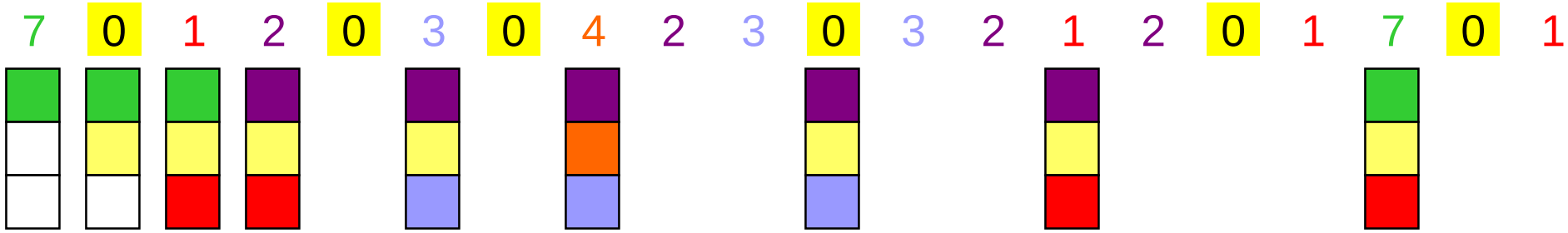
Referenzkette 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

3 Kacheln: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

4 Kacheln: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

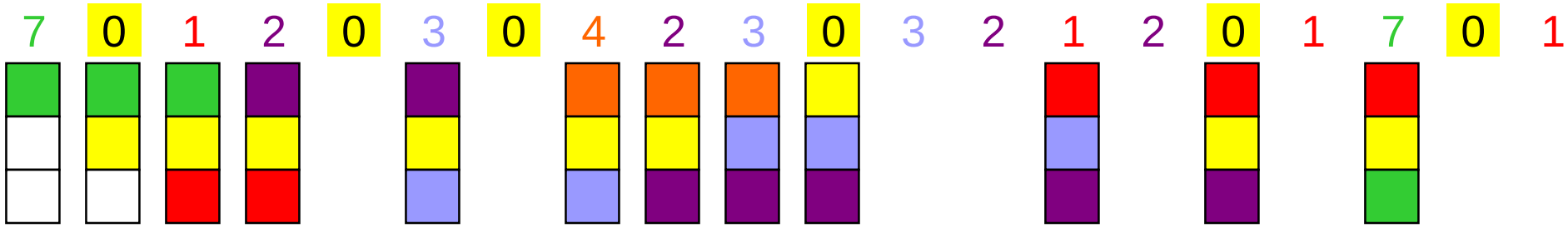


2.2 Optimaler Algorithmus



- Ersetze die Seite, die zukünftig am längsten nicht mehr benötigt wird.
- im Beispiel: 9 Seitenfehler (FIFO: 15)
- Problem: Wissen über die Zukunft erforderlich

2.3 Least Recently Used (LRU)



- Auslagern der am längsten nicht mehr benutzten Seite
- im Beispiel: 12 Seitenfehler (optimal: 9, FIFO: 15)
- LRU ist aufwändig zu implementieren.

2.3.1 Zähler und Stacks

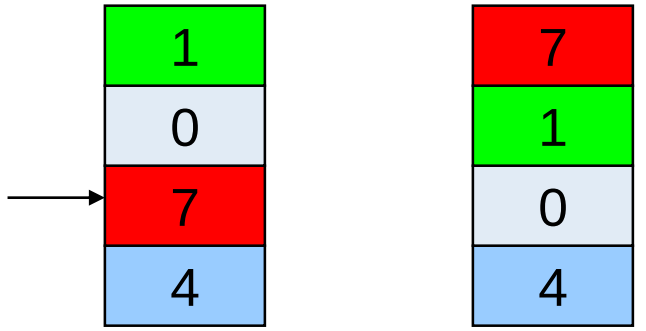
UNIVERSITÄT BERN

Zähler

- Verfahren
  - Fortlaufende Erhöhung eines Zählers (logische Uhr)
  - Zählerwert wird bei Referenz in Seitentabelle kopiert.
  - Auslagern der Seite mit kleinstem Zählerstand
- Probleme
  - Zählerstände müssen bei jedem Zugriff aktualisiert werden.
  - erfordert Durchsuchen der ganzen Tabelle
  - Zähler-Overflow

Stacks

- Verfahren
  - Referenzierte Seite wird im Stack nach oben verschoben.
  - Ggf. wird eine Seite in der Mitte des Stacks verschoben.
  - Auslagern der untersten Seite

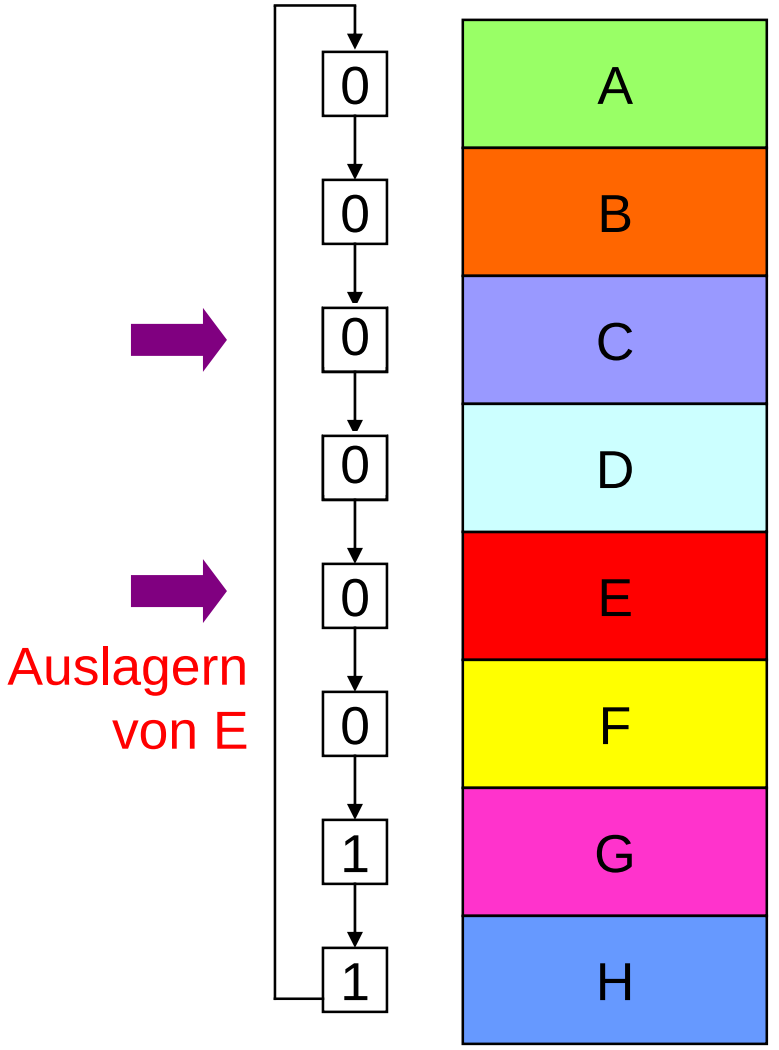


2.3.2 Referenzbits

- Referenzbits (für jede Seite) werden durch Hardware bei Seitenzugriff gesetzt.
- Erweiterung
  - 8-Bit-Schieberegister für jede Seite
  - zyklische Verschiebeoperationen
  - Zugriff auf Seite mit Schieberegisterinhalt 00110111 liegt länger zurück als Zugriff auf Seite mit Schieberegisterinhalt 01000100.
  - Seite mit kleinstem Schieberegisterinhalt wird ausgelagert.
- nur 1 Referenzbit $\rightarrow$ Second Chance

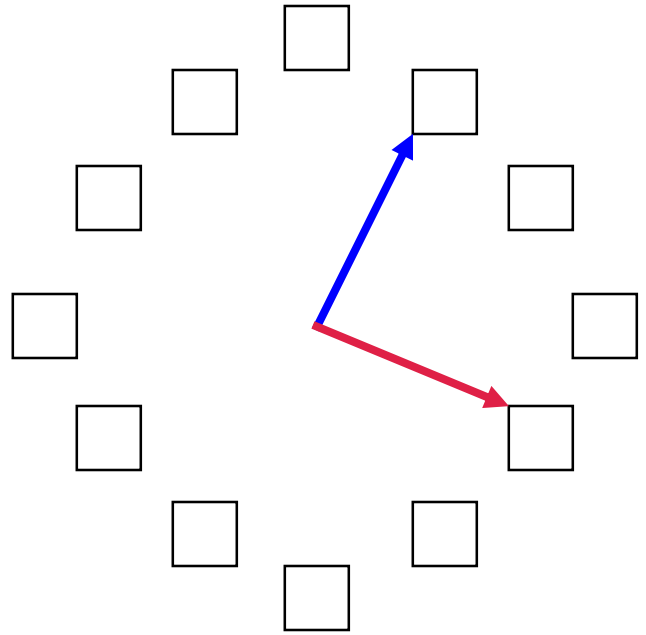
2.3.3 Second Chance

- 1 Referenzbit
- zirkulierende Warteschlange
- Seite mit Referenzbit 0 wird ausgelagert.
- Seite mit Referenzbit 1 wird zunächst nicht ausgelagert, Referenzbit aber auf 0 zurückgesetzt.
- alle Bits = 1: FIFO



2.3.4 Clock-Algorithmus

- Problem bei Second Chance: möglicherweise lange Suche nach Verdrängungskandidat
- Implementierung mit 2 Zeigern
- Vorderer Zeiger setzt Referenzbit auf 0 zurück.
- Hinterer Zeiger prüft Referenzbit.
  - Bei zurückgesetztem Bit wird die Seite verdrängt.
  - Bei gesetztem Referenzbit werden beide Zeiger weitergeschaltet.
- Abstand zwischen Zeigern bestimmt die maximale Anzahl von Schritten.



2.3.5 Erweiterung Second Chance

- Berücksichtigung von Referenz- und Modify-Bit
- Klassen
  - (0,0): weder kürzlich referenziert noch verändert (beste Kandidaten für Auslagerung)
  - (0,1): kürzlich nicht referenziert, aber verändert (Auslagerung benötigt Zeit)
  - (1,0): kürzlich referenziert, aber unverändert (wird ggf. bald wieder benötigt)
  - (1,1): kürzlich referenziert und verändert (schlechtester Kandidat)
- Auswahl einer Seite in der niedrigsten nicht leeren Klasse
- Implementierung in MacOS
- Benutzung einer Variante von Second-Chance bei Speicherverwaltung (Demand Paging) in Unix/4BSD

2.4 Zählalgorithmen

- Zähler zählt Anzahl der Referenzen auf einzelne Seiten.
- Least Frequently Used (LFU)
  - Seite mit kleinstem Zählerstand (inaktive Seiten)
- Most Frequently Used (MFU)
  - Seite mit höchstem Zählerstand (möglicherweise am längsten im Speicher)
- Zählalgorithmen werden selten benutzt.
  - schlechte Approximation des optimalen Algorithmus
  - aufwändige Implementierung

2.5 Optimierungen mit Page Buffering

- Halten eines Pools freier Speicherkacheln
  - Neue Seite kann sofort eingelagert werden, ohne auf Ende einer Auslagerung zu warten.
- Schreiben modifizierter Seiten auf Sekundärspeicher bei untätigem Paging-System
  - Es sind ggf. keine Schreiboperationen beim späteren Auslagern notwendig.
- Pool freier Speicherkacheln und Merken des Inhalts / der Seite
  - Falls die entsprechende Seite eingelagert werden soll, ist der Speicherinhalt noch aktuell.

3. Allokation von Speicherkacheln

Jeder Prozess benötigt eine minimale Anzahl von Speicherkacheln.

- Instruktionen können sich über zwei verschiedene Seiten erstrecken.
- Kopieroperationen über mehrere Seiten
- mehrere Operanden
- indirekte Adressierung (mehrere Stufen)

3.1 Allokationsalgorithmen

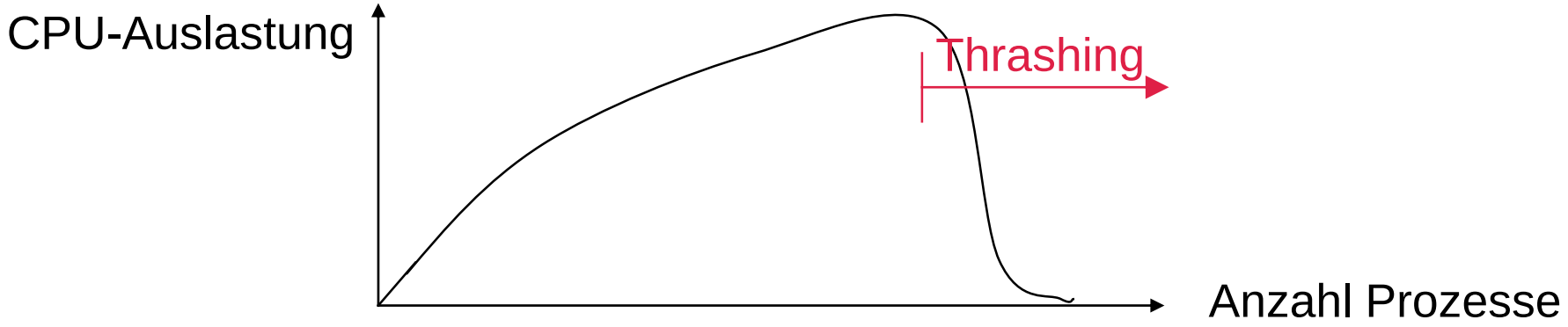
- Gleichverteilte Allokation
  - N Prozesse, k Speicherkacheln $\Rightarrow$ k/N Speicherkacheln pro Prozess
- Proportionale Allokation
  - Prozesse p_i mit der Grösse s_i, $S = \Sigma\, s_i$
  - Allokation für Prozess p_i: $a_i = \frac{s_i}{S} \bullet k$
- Prioritäts-abhängige Allokation
  - Proportionale Allokation unter Berücksichtigung der Priorität anstatt der Prozessgrösse
- Kombination von Priorität und Prozessgrösse

3.2 Globale und lokale Allokation

- Globale Allokation
  - Ein Prozess selektiert zu ersetzende Kachel aus der Menge aller Kacheln.
  - Ein Prozess kann einem anderen Prozess Kacheln wegnehmen.
  - Wachstum von Prozessen mit hoher Priorität möglich
  - Optimierung des Gesamtsystems
  - Einzelner Prozess kann seine Seitenfehlerrate nicht alleine beeinflussen, daher gegenseitige Beeinflussung des Paging-Verhaltens
- Lokale Allokation
  - Jeder Prozess selektiert nur eigene Speicherkacheln.
  - Speicherkacheln können unbenutzt bleiben.

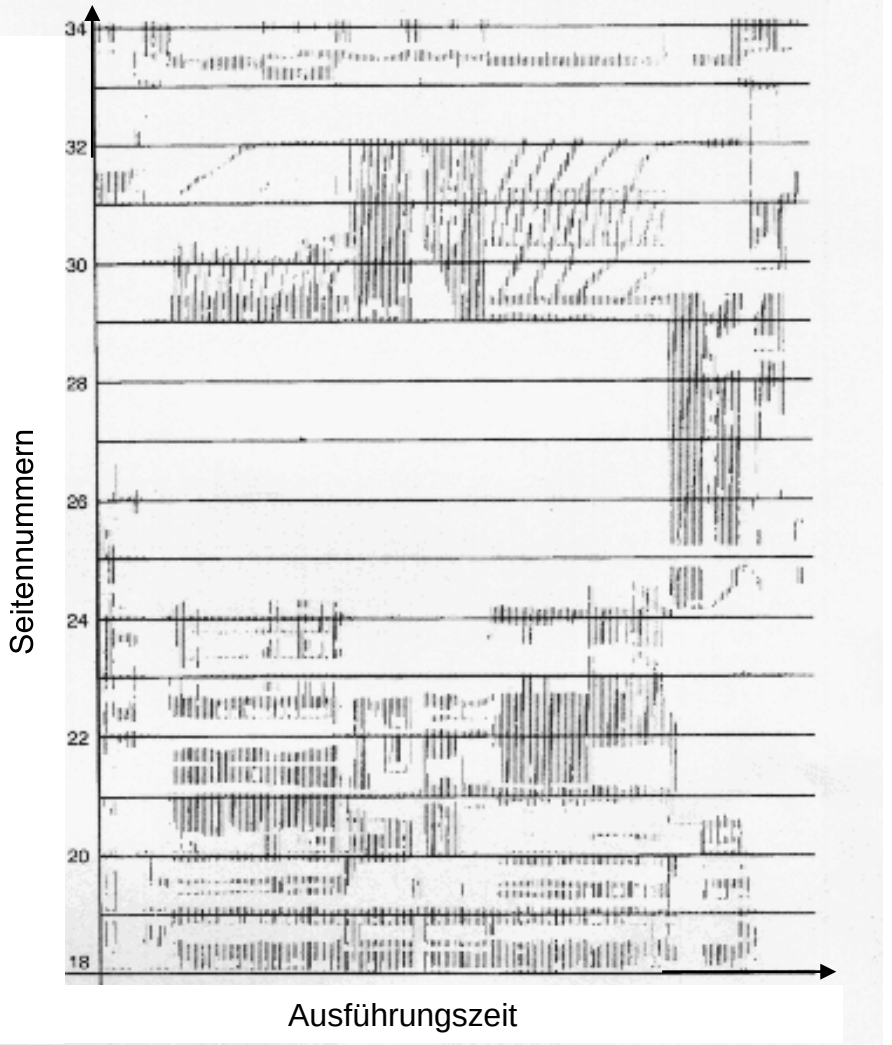
4. Thrashing

- Thrashing: Prozess ist mehr mit Ein-/Auslagern als mit der eigentlichen Verarbeitung beschäftigt.
  - Seitenfehlerrate steigt, wenn Prozess zu wenig Speicherkacheln besitzt.
  - Thrashing erhöht Bedienzeit anderer Prozesse bei einem Seitenfehler, da sich Warteschlange für Paging (Ein- und Auslagern) füllt.
- Lösungsansatz
  - Vermeiden von Thrashing durch Zuordnen einer ausreichenden Anzahl von Kacheln
    - Problem: Bestimmen dieser Anzahl
    - Lösungsansatz: Lokalität bzw. Working-Set-Modell
  - Beenden von Prozessen mit niedriger Priorität, falls nicht genügend Kacheln verfügbar



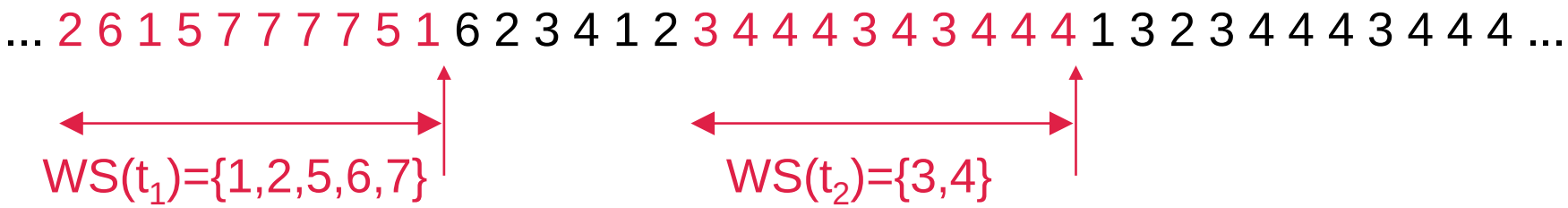
4.1 Lokalität

- Lokalität: Prozesse benutzen während einer Phase meist nur relativ kleine Menge von Seiten (Working Set, WS)
- Prozess bewegt sich von einer Lokalität, z.B. Subroutine (Code, lokale Variablen), zur nächsten.
- Grundlage für Caching und Seitenersetzung (Kandidaten sind nicht in der Working Set der Prozesse.)
- Approximation durch Working-Set-Modell



4.2 Working-Set-Modell

- Working Set Window Δ = feste Anzahl von Seitenreferenzen, z.B. $\Delta = 10$
- Working Set = Menge der Δ kürzlich referenzierten Seiten



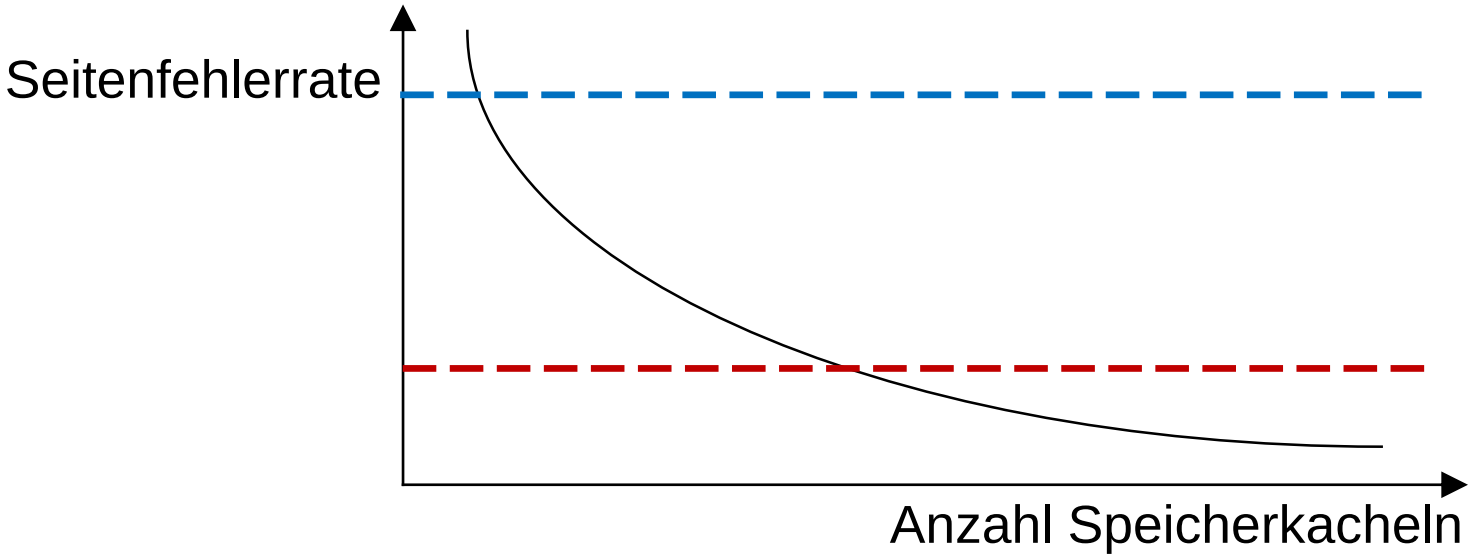
- Δ muss passend gewählt werden.
  - Δ zu gross: WS umfasst mehrere Lokalitäten.
  - Δ zu klein: WS umfasst nicht die ganze Lokalität.
- $B = \Sigma\ WSS_i$
  - WSS_i: Working Set Size von Prozess i
  - B: Gesamtbedarf von Speicherkacheln
- B > k (k: Anzahl verfügbarer Kacheln) $\Rightarrow$ Thrashing $\Rightarrow$ Beenden von Prozessen

4.3 Implementierung des Working-Set-Modells

- Problem: Working Set (Size) ändert sich laufend.
- Lösungsansatz:
  Kopieren/Speichern und Löschen eines Referenzbits nach Ablauf eines Timers
  - z.B. Timer-Interrupt alle 5000 Speicherreferenzen für $\Delta = 10000$, Speichern von 2 Bits
  - Ist eines der beiden gespeicherten Bits oder das aktuelle Referenzbit gesetzt, wurde auf die Seite während der letzten 10000-15000 Referenzen zugegriffen.
  - Reduzieren der Ungenauigkeit durch Ändern der Parameter, z.B. Speichern von 10 Bits und Timer-Intervall von 1000 Referenzen

4.4 Page-Fault-Frequency-Strategie

- Erhöhen der Anzahl von Speicherkacheln, falls Seitenfehlerrate über oberen Schwellwert ansteigt bzw. Verdrängen eines Prozesses, falls keine Speicherkacheln verfügbar
- Freigabe von Speicherkacheln bei Unterschreiten des unteren Schwellwerts



5.1 Pre-Paging

- Demand Paging
  - Seiten werden nur bei vorangegangenem Seitenfehler geladen.
- Pre-Paging
  - Seiten werden in Hauptspeicher ohne vorangegangenen Seitenfehler geladen.
  - Versuch, hohe Seitenfehlerrate bei (Neu-)Start eines Prozesses zu vermeiden
  - Allokation von N Kacheln auf Verdacht
  - Kosten/Nutzen-Verhältnis von Pre-Paging hängt davon ab, wie genau N gewählt werden kann.
  - Merken der Working Set und Einlagern vor Neustart
- Kombination von Pre- und Demand Paging:
  - Pre-Paging zu Beginn der Programmausführung vermeidet anfänglich hohe Seitenfehlerrate.
  - Pre-Paging für Anfang des Programmcodes, statische Daten, Teile des Heaps und Stacks
  - weitere Einlagerungen durch Demand Paging

5.2 Dämon-Paging

- Trennung von Seitenauslagerung und -einlagerung, um ausreichenden Vorrat freier Speicherkacheln für schnelle Reaktion auf weitere Anforderungen zu haben
- BSD Unix:
  Spezieller Prozess prüft periodisch (z.B. alle 250 ms) die Anzahl freier Kacheln und lagert ggf. Seiten aus (mit Clock-Algorithmus) bis Mindestanzahl freier Kacheln erreicht wird.

5.3 Seitengrösse

	grosse Seiten	kleine Seiten
interner Verschnitt	-	+
Lokalität	-	+
I/O-Leistung	+	-
Grösse der Seitentabelle	+	-
Seitenfehlerrate	+	-

5.4 Einfluss der Programmstruktur

```
int i, j, A[128][128];
for (j=0;j<128;j++)
{
    for (i=0;i<128;i++)
   {
       A[i][j]=0;
   }
}
```

- 1 Reihe benötigt 1 Seite
- Allokation von weniger als 128 Seiten: 128·128 = 16´384 Seitenfehler

```
int i, j, A[128][128];
for (i=0;i<128;i++)
{
    for (j=0;j<128;j++)
   {
       A[i][j]=0;
   }
}
```

- 1 Seitenfehler für jede neue Reihe
- insgesamt 128 Seitenfehler

Speichern der Elemente: `A[0][0],A[0][1],...,A[0][127],A[1][0],...,A[127][127]`

5.5 Realzeitverarbeitung

- Konflikt zwischen virtuellem Speicher und Realzeitprozessen
- Realzeitprozesse sollten keinen virtuellen Speicher nutzen.
- Sperren der Seiten von Realzeitprozessen
- POSIX: **`mlock/munlock`** zum (Ent)Sperren von Seiten