

# Systemprogrammierung

## Speicherverwaltung: Speichervirtualisierung

Wolfgang Schröder-Preikschat

Lehrstuhl Informatik 4

2. Februar 2011

# Gliederung

- 1 Überblick
- 2 Ladestrategie
  - Überblick
  - Seitenumlagerung
- 3 Ersetzungsstrategie
  - Überblick
  - Verfahrensweisen
  - Approximation
  - Seitenanforderung
  - Arbeitsmenge
- 4 Zusammenfassung

# Politiken bei der Speicherverwaltung

## Speicherzuteilungsverfahren

### Platzierungsstrategie (engl. *placement policy*)

- **wohin** im Arbeitsspeicher ist ein Fragment abzulegen?
  - dorthin, wo der Verschnitt am kleinsten/größten ist?
  - oder ist es egal, weil Verschnitt zweitrangig ist?

## Speichervirtualisierung [3]

### Ladestrategie (engl. *fetch policy*)

- **wann** ist ein Fragment in den Arbeitsspeicher zu laden?
  - im Moment der Anforderung durch einen Prozess?
  - oder im Voraus, auf Grund von Vorabwissen oder Schätzungen?

### Ersetzungsstrategie (engl. *replacement policy*)

- **welches** Fragment ist ggf. aus den Arbeitsspeicher zu verdrängen?
  - das älteste, am seltensten genutzte oder am längsten ungenutzte?

# Gliederung

- 1 Überblick
- 2 **Ladestrategie**
  - Überblick
  - Seitenumlagerung
- 3 Ersetzungsstrategie
  - Überblick
  - Verfahrensweisen
  - Approximation
  - Seitenanforderung
  - Arbeitsmenge
- 4 Zusammenfassung

# Einlagerung von Seiten/Segmenten

Spontanität oder vorseilender Gehorsam

Einzelanforderung „*on demand*“ → *present bit*

- Seiten-/Segmentzugriff führt zum *Trap*
  - engl. *page/segment fault*
- Ergebnis der Ausnahmebehandlung ist die Einlagerung der angeforderten Einheit

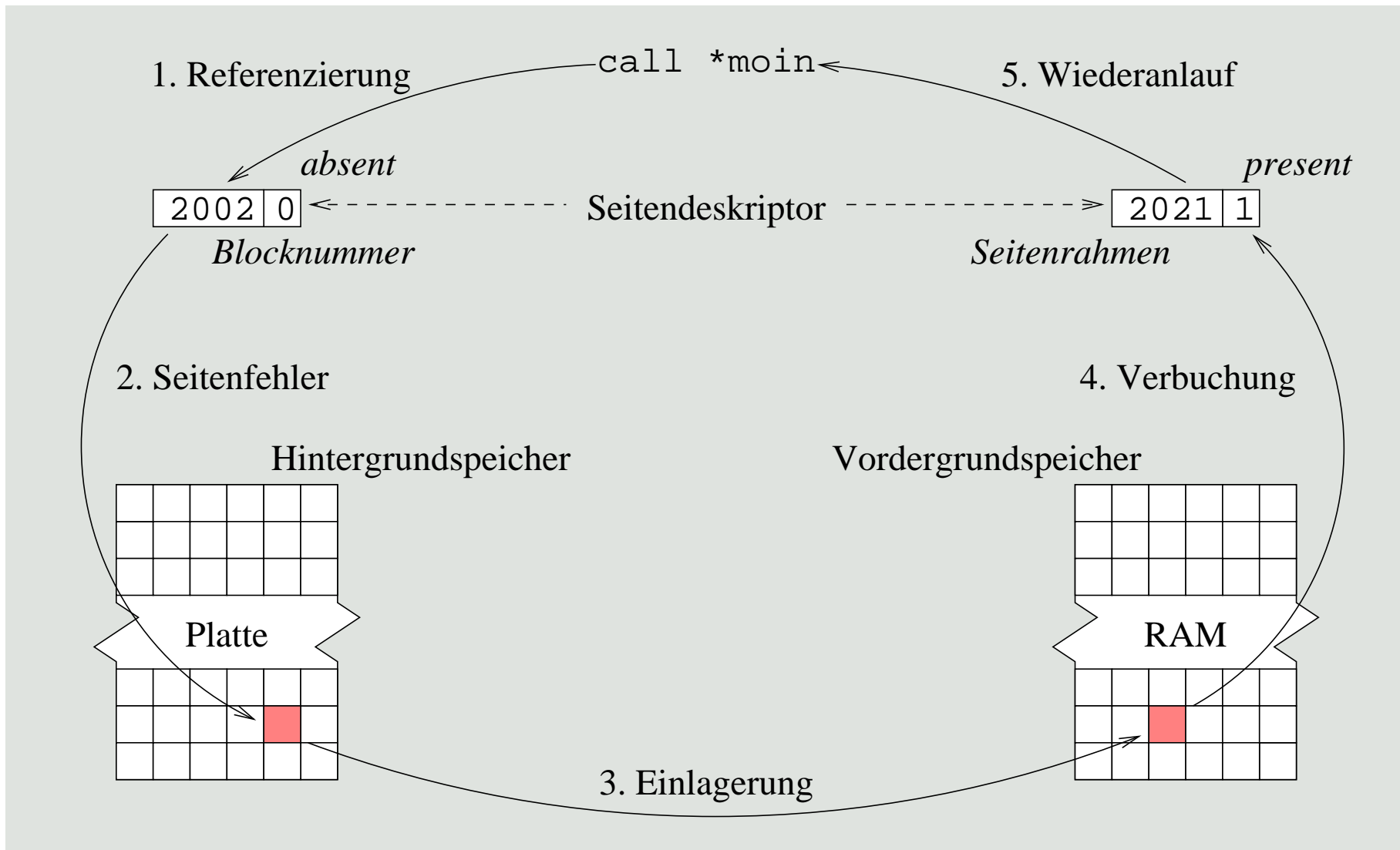
Vorausladen „*anticipatory*“

- **Heuristiken** liefern Hinweise über Zugriffsmuster
  - Programmlokalität, Arbeitsmenge (*working set*)
- alternativ auch als **Vorabruf** (engl. *prefetch*) im Zuge einer Einzelanforderung
  - z.B. zur Vermeidung von Folgefehlern

- ggf. fällt die **Verdrängung** (Ersetzung) von Seiten/Segmenten an

# Einzelanforderung

*On-demand paging* — durch ein „Rufgerät“ (engl. *pager*) des Betriebssystems



# Einzelanforderung mit anschließendem Vorausladen

Vorbeugung ggf. nachfolgender Seitenfehler

`call *moin`

- ① den gescheiterten Befehl dekodieren, Adressierungsart feststellen
- ② da der Operand die Adresse einer Zeigervariablen (`moin`) ist, den Adresswert auf Überschreitung einer Seitengrenze prüfen
- ③ da der Befehl die Rücksprungadresse stapeln wird, die gleiche Überprüfung mit dem Stapelzeiger durchführen
- ④ in der Seitentabelle die entsprechenden Deskriptoren lokalisieren und prüfen, ob die Seiten anwesend sind
  - jede abwesende Seite (*present bit* = 0) ist einzulagern
- ⑤ da jetzt die Zeigervariable (`moin`) vorliegt, sie dereferenzieren und ihren Wert auf Überschreitung einer Seitengrenze prüfen
  - hierzu wie bei 4. verfahren
- ⑥ den unterbrochenen Prozess den Befehl wiederholen lassen

- **Teilemulation** eines Maschinenbefehls durch das Betriebssystem

# Gliederung

- 1 Überblick
- 2 Ladestrategie
  - Überblick
  - Seitenumlagerung
- 3 Ersetzungsstrategie**
  - Überblick
  - Verfahrensweisen
  - Approximation
  - Seitenanforderung
  - Arbeitsmenge
- 4 Zusammenfassung



# Verdrängung eingelagerter Fragmente

Platz schaffen zur Einlagerung anderer Fragmente (d.h., Seiten oder Segmente)

Konsequenz zur **Durchsetzung der Ladestregie** bei Hauptspeichermangel

- wenn eine Überbelegung des Hauptspeichers vorliegt
  - der Speicherbedarf aller Prozesse ist größer als der verfügbare RAM
- aber auch im Falle (extensiver) externer Fragmentierung

**OPT** (optimales Verfahren) Verdrängung der Seite, die am längsten nicht mehr verwendet werden wird — unrealistisch

- erfordert Wissen über die im weiteren Verlauf der Ausführung von Prozessen generierten Speicheradressen, was bedeutet:
  - das Laufzeitverhalten von Prozessen ist im Voraus bekannt ?
  - Eingabewerte sind vorherbestimmt ?
  - asynchrone Programmunterbrechungen sind vorhersagbar ?
- bestenfalls ist eine gute **Approximation** möglich/umsetzbar

- **Zugriffsfehlerwahrscheinlichkeit** und **Zugriffsfehlerrate** verringern

# Praxistaugliche Herangehensweisen

**Approximation des optimalen Verfahren** greift auf Wissen aus der Vergangenheit/Gegenwart zurück, d.h., ersetzt wird ...

**FIFO** (*first-in, first-out*) das zuerst eingelagerte Fragment

- Fragmente entsprechend des Einlagerungszeitpunkts verketteten

**LFU** (*least frequently used*) das am seltenste genutzte Fragment

- jeden Zugriff auf eingelagerte Fragmente zählen
- Alternative: **MFU** (*most frequently used*)

**LRU** (*least recently used*) das kürzlich am wenigsten genutzte Fragment

- Zeitstempel, Stapeltechniken oder Referenzlisten einsetzen
- bzw. weniger aufwändig durch Referenzbits approximieren

- zu ersetzende/verdrängende Fragmente sind vorzugsweise **Seiten**

# Zählverfahren

Auswahlkriterium ist die Häufigkeit von Seitenreferenzen

Zähler basierte Ansätze führen Buch darüber, wie häufig eine Seite innerhalb einer bestimmten Zeitspanne referenziert worden ist:

- im Seitendeskriptor ist dazu ein **Referenzzähler** enthalten
- der Zähler wird mit jeder Referenz zu der Seite inkrementiert
- aufwändige Implementierung, bei mäßiger Approximation von OPT

**LFU** ersetzt die Seite mit dem kleinsten Zählerwert

- Annahme: **aktive Seiten** haben große Zählerwerte und weniger aktive bzw. **inaktive Seiten** haben kleine Zählerwerte
- große Zählerwerte können dann aber auch jene Seiten haben, die z.B. nur in der Initialisierungsphase aktiv gewesen sind

**MFU** ersetzt die Seite mit dem größten Zählerwert

- Annahme: **kürzlich aktive Seiten** haben eher kl. Zählerwerte

# Zeitverfahren

Auswahlkriterium ist die Zeitspanne zurückliegender Seitenreferenzen

$LRU_{time}$  verwendet einen Zähler („logische Uhr“) in der CPU, der bei jedem Speicherzugriff erhöht und in den zugehörigen Seitendeskriptor geschrieben wird

- verdrängt die Seite mit dem kleinsten **Zeitstempelwert**

$LRU_{stack}$  nutzt einen Stapel eingelagerter Seiten, aus dem bei jedem Seitenzugriff die betreffende Seite herausgezogen und wieder oben drauf gelegt wird

- verdrängt die Seite am **Stapelboden**

$LRU_{ref}$  führt Buch über alle zurückliegenden Seitenreferenzen

- verdrängt die Seite mit dem größten **Rückwärtsabstand**

- entsprechen OPT, wenn allerdings die Vergangenheit betrachtet wird
  - gute Approximation von OPT, bei sehr aufwändiger Implementierung

## LRU: Alterung von Seiten (engl. *page aging*) verfolgen

**Referenzbit** (engl. *reference bit*) im Seitendeskriptor zeigt an, ob auf die zugehörige Seite zugegriffen wurde:

0  $\mapsto$  kein Zugriff

1  $\mapsto$  Zugriff bzw. Einlagerung

- das Alter eingelagerter Seiten wird periodisch (Zeitgeber) bestimmt:
  - für jede eingelagerte Seite wird ein  $N$ -Bit Zähler (*age counter*) geführt
  - der Zähler ist als **Schieberegister** (engl. *shift register*) implementiert
  - nach Aufnahme eines Referenzbits in den Zähler, wird es gelöscht
- „kürzlich am wenigsten genutzte“ Seiten haben den Zählerwert 0
  - d.h., sie wurden seit  $N$  Perioden nicht mehr referenziert (vgl. NT)

# LRU: Schieberegistertechnik

## Bestimmung des Lebensalters einer Seite

*page aging* (Forts.) mit Ablauf eines Zeitquantums (Tick), werden die Referenzbits eingelagerter Seiten des laufenden Prozesses in die Schieberegister seiner Seitendeskriptoren übernommen

- z.B. ein 8-Bit „*age counter*“:  $age = (age \gg 1) | (ref \ll 7)$

| Referenzbit | Alter ( <i>age</i> , initial 0) |
|-------------|---------------------------------|
| 1           | 1 0 0 0 0 0 0 0                 |
| 1           | 1 1 0 0 0 0 0 0                 |
| 0           | 0 1 1 0 0 0 0 0                 |
| 1           | 1 0 1 1 0 0 0 0                 |
| ⋮           | ⋮                               |

- Schieberegisterinhalt (*age*) als Ganzzahl interpretiert liefert ein Maß für die Aktivität einer Seite
- mit abnehmendem Betrag, d.h. einer sinkenden Prozessaktivität, steigt die Ersetzungspriorität

- Aufwand steigt mit Adressraumgröße des unterbrochenen Prozesses

# LRU: Seiten eine zweite Chance einräumen

Annahme: Referenzierte Seiten sind vermeintlich aktive Seiten

*second chance* (auch: *clock policy*)

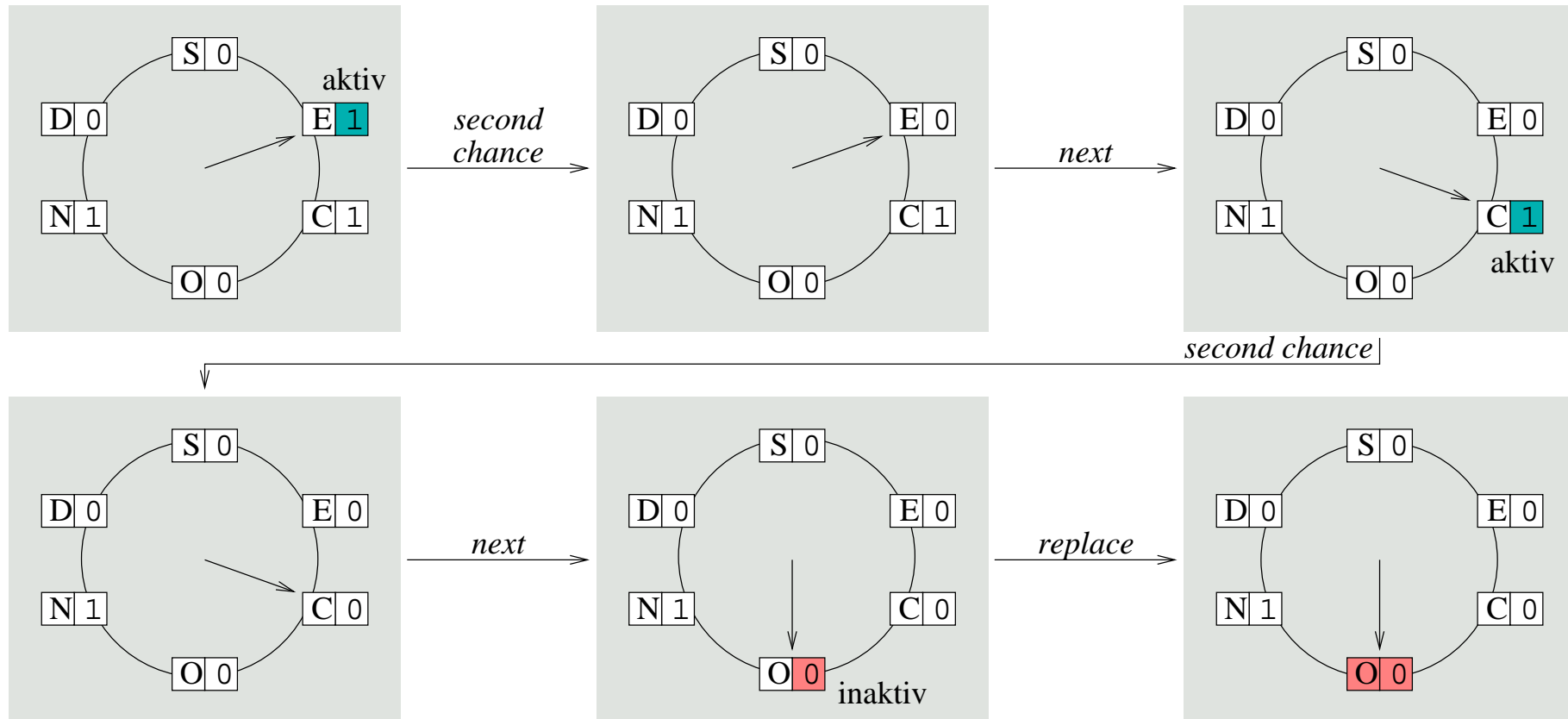
- arbeitet im Grunde nach FIFO, berücksichtigt jedoch zusätzlich noch die Referenzbits der jeweils in Betracht zu ziehenden Seiten
- periodisch (Zeitgeber, Tick) werden die Seitendeskriptoren des (unterbrochenen) laufenden Prozesses untersucht

| Referenzbit | Aktion                   | Bedeutung                    |
|-------------|--------------------------|------------------------------|
| 1           | Referenzbit auf 0 setzen | Seite erhält zweite Chance   |
| 0           | —                        | Seite ist Ersetzungskandidat |

- schlimmstenfalls erfolgt ein Rundumschlag über alle Seiten, wenn die Referenzbits aller betrachteten Seiten (auf 1) gesetzt waren
  - die Strategie „entartet“ dann zu FIFO

- unterscheidet nicht zwischen lesende und schreibende Seitenzugriffe

# LRU: Funktionsweise der *Clock Policy*



- E** ist aktiv, Referenzbit auf Null setzen, Seite im Hauptspeicher behalten
- C** ist aktiv, Referenzbit auf Null setzen, Seite im Hauptspeicher behalten
- O** ist inaktiv, Seite ersetzen: den Seitenrahmen neu „bespannen“



# LRU: Schreibzugriffe stärker gewichten als Lesezugriffe

*enhanced second chance* prüft zusätzlich, ob eine Seite schreibend oder nur lesend referenziert wurde

- Grundlage dafür ist ein **Modifikationsbit** (*modify/dirty bit*)
  - ist als weiteres Attribut in jedem Seitendeskriptor enthalten
  - wird bei Schreibzugriffen auf 1 gesetzt, bleibt sonst unverändert
- zusammen mit dem Referenzbit zeigen sich vier Paarungen ( $R, D$ ):

|        | Bedeutung            | Entscheidung         |
|--------|----------------------|----------------------|
| (0, 0) | ungenutzt            | beste Wahl           |
| (0, 1) | beschrieben          | keine schlechte Wahl |
| (1, 0) | kürzlich gelesen     | keine gute Wahl      |
| (1, 1) | kürzlich beschrieben | schlechteste Wahl    |

- kann für jeden Prozess(adressraum) zwei Umläufe erwirken
  - gibt referenzierten Seiten damit eine **dritte Chance** (vgl. MacOS)

# Freiseitenpuffer

Reserve freier (d.h. ungebundener) Seitenrahmen garantieren

Alternative zur Seitenersetzung, die den **Vorabruf** von Seiten nutzt

- Steuerung der Seitenüberlagerung über **Schwellwerte** (*water mark*):
  - low* Seitenrahmen als frei markieren, Seiten ggf. auslagern
  - high* Seiten ggf. einlagern, **Vorausladen**
- die Auswahl greift z.B. auf Referenz-/Modifikationsbits zurück

**Freiseiten** gelangen in einen **Zwischenspeicher** (engl. *cache*)

- die Zuordnung von Seiten zu Seitenrahmen bleibt jedoch erhalten
- vor ihrer Ersetzung doch noch benutzte Seiten werden zurückgeholt
- sog. *Reclaiming* von Seiten durch Prozesse (vgl. Solaris, Linux)

- vergleichsweise effizient: die Ersetzungszeit entspricht der Ladezeit

# Verteilung von Seitenrahmen auf Prozessadressräume

Prozessen ist mindestens die **kritische Masse von Seitenrahmen** zur Verfügung zu stellen, um in ihrer Ausführung voranschreiten zu können

- Rechnerausstattung/-architektur geben „harte“ Begrenzungen vor:
  - Obergrenze** die Größe des Arbeitsspeichers
  - Untergrenze** definiert durch den komplexesten Maschinenbefehl
    - schlimmster Fall möglicher Seitenfehler
- innerhalb dieser Grenzen, ist die Zuordnung ...
  - gleichverteilt** in Abhängigkeit von der Prozessanzahl und/oder
  - größenabhängig** bedingt durch den (statischen) Programmumfang
- „weiche“ Begrenzungen ergeben sich z.B. durch die gegenwärtige Systemlast und den gewünschten Grad an Mehrprogrammbetrieb

# Einzugsbereiche: Wirkungskreis der Seitenüberlagerung

**lokal** nur Seitenrahmen des von der Seitenersetzung betroffenen Prozessadressraums nutzen (vgl. NT)

- Seitenfehlerrate ist von einem Prozess selbst kontrollierbar
  - Prozesse verdrängen niemals Seiten anderer Adressräume
  - fördert ein deterministisches Laufzeitverhalten von Prozessen
  - Seitenfehler haben **Trap-Eigenschaften**
- **statische Zuordnung** von Seitenrahmen zum Adressraum

**global** Seitenrahmen aller Adressräume nutzen; **dynamische Zuordnung**

- Verdrängung/Ersetzung von Seitenrahmen ist unvorhersehbar und auch nicht bzw. nur schwer reproduzierbar
  - Seitenfehler haben **Interrupt-Eigenschaften**
- der zeitliche Ablauf einer Programmausführung ist abhängig von den in anderen Adressräumen vorgehenden Aktivitäten

- Kombination beider Ansätze möglich: Prozess-/Adressraumklassen
  - z.B. nur Echtzeitprozesse der lokalen Seitenersetzung unterziehen

## Seitenflattern (engl. *thrashing*)

„Dresche beziehen“ — wenn durch Seitenüberlagerung verursachte E/A die gesamten Systemaktivitäten bestimmt [1]

- eben erst ausgelagerte Seiten werden sofort wieder eingelagert
  - Prozesse verbringen mehr Zeit beim „*paging*“ als beim Rechnen
  - das Problem ist immer in Relation zu der Zeit zu setzen, die Prozesse mit sinnvoller Arbeit verbringen
- ein mögliches Phänomen der globalen Seitenersetzung
  - Prozesse bewegen sich zu nahe am Seiten(rahmen)minimum
  - zu hoher Grad an Mehrprogrammbetrieb
  - ungünstige Ersetzungsstrategie
- verschwindet ggf. so plötzlich von allein, wie es aufgetreten ist ...

- **Lokalität** der Speicherzugriffe **von Prozessen** beachten/bestimmen
- **Umlagerung** (*Swapping*) **von Adressräumen bzw. Arbeitsmengen**

# Seitenflattern vermeiden

Heuristik über zukünftig erwartete Seitenzugriffe erstellen

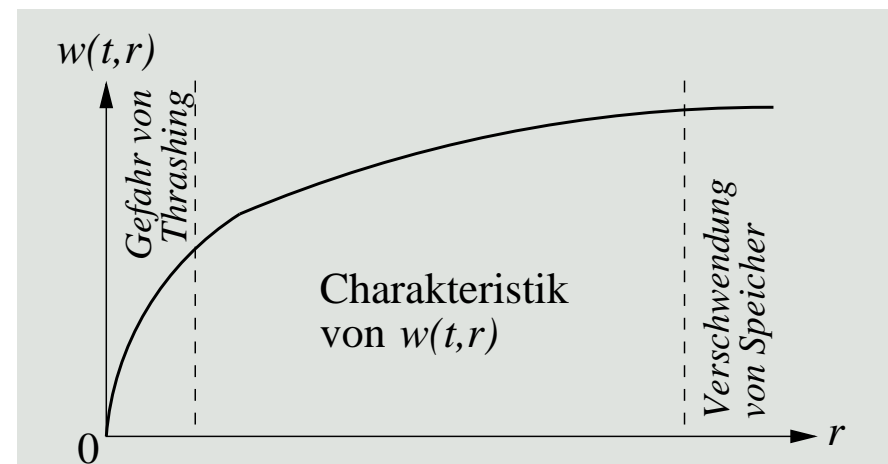
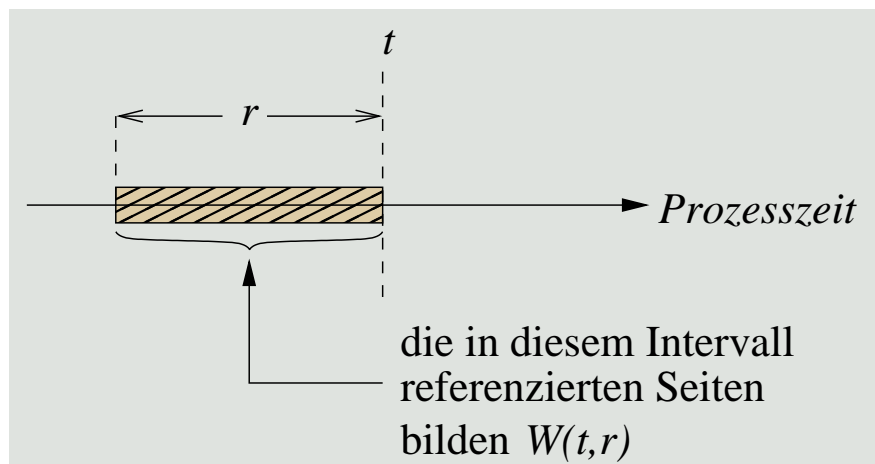
$W(t, r)$  **Arbeitsmenge** (engl. *working set*, [2, 4]): Satz von Seiten, den ein Prozess lokal/das System global in Benutzung haben wird

$t$  Beobachtungszeitpunkt

$r$  Arbeitsmengenparameter (engl. *working set parameter*)

$w(t, r)$  **Arbeitsmengengröße**: Anzahl **aktiver Seiten** in  $W(t, r)$

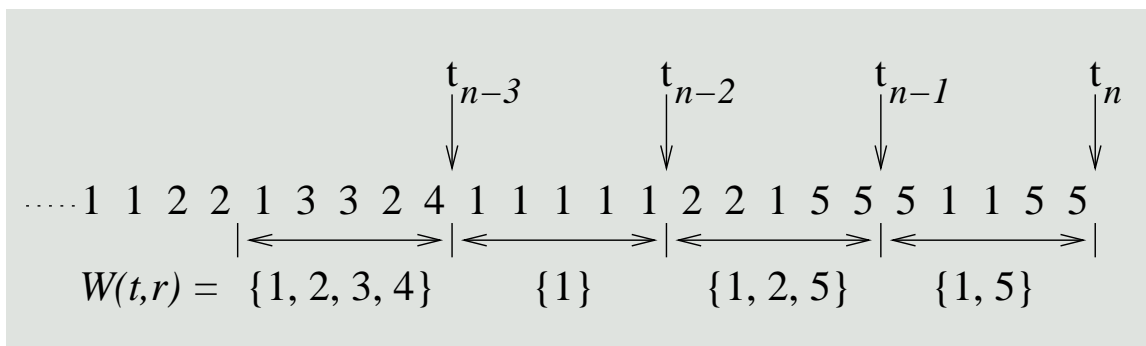
Blick in die Zukunft, abgeschätzt auf Basis der im **Zeitintervall**  $(t - r, t)$  jüngst von einem Prozess jeweils referenzierten Seiten



# Seitenflattern vermeiden: Arbeitsmengendynamik

Bewegliches Arbeitsmengenfenster (engl. *moving working set window*)

- die Berechnung der Arbeitsmenge ist nur näherungsweise möglich:
  - Ausgangspunkt ist die **Seitenreferenzfolge** der jüngeren Vergangenheit
  - darauf wird ein **Fenster** (engl. *working set window*) geöffnet
  - das Fenster bewegt sich vorwärts mit jeder weiteren Speicherreferenz



- zu kleine Fenster halten benutzte Seiten draußen
  - Seitenfehlerrate steigt
- zu große Fenster halten unbenutzte Seiten drinnen

- die **Fensterbreite**  $r$  gibt eine „feste Anzahl von Maschinenbefehlen“
  - sie wird approximiert durch periodische Unterbrechungen
  - die Befehlsanzahl ergibt sich in etwa aus der **Periodenlänge**

## Seitenflattern vermeiden: Approximation der Arbeitsmenge

Grundlage sind Referenzbit, Seitenalter und periodische Unterbrechungen

- bei jedem Tick werden die Referenzbits eingelagerter Seiten geprüft:
  - 1  $\rightsquigarrow$  Referenzbit und Alter auf 0 setzen
  - 0  $\rightsquigarrow$  Alter der Seite um 1 erhöhen (ähnlich zu S. 14)
- Alterswerte von Arbeitsmengenseiten sind kleiner als die Fensterbreite

Lösungsansätze unterscheiden **lokale (a)** und **globale (b) Arbeitsmenge**:

(a) nur Seiten des laufenden Prozesses altern

- Problem: gemeinsam genutzte Seiten (z.B. *shared libraries*)

(b) Seiten aller „aktiven Adressräume“ altern

- Problem: vergleichsweise (sehr) hoher Systemaufwand

**Umlagerung** bzw. **Vorausladen** ganzer Arbeitsmengen praktizieren

- **Mindestmenge von Seitenrahmen** laufbereiter Prozesse vorhalten
- Prozesse mit unvollständigen Arbeitsmengen stoppen/suspendieren



# Gliederung

- 1 Überblick
- 2 Ladestrategie
  - Überblick
  - Seitenumlagerung
- 3 Ersetzungsstrategie
  - Überblick
  - Verfahrensweisen
  - Approximation
  - Seitenanforderung
  - Arbeitsmenge
- 4 Zusammenfassung

# Resümee

- die **Ladestrategie** sorgt für die Einlagerung von Seiten (Segmenten)
  - auf Anforderung oder im Voraus
- eingelagerte Seiten unterliegen der **Ersetzungsstrategie**
  - Ersetzungsverfahren: OPT, FIFO, LFU, MFU, LRU (*clock*)
    - alternativer Ansatz ist der Freiseitenpuffer
  - Verdrängung arbeitet adressraumlokal oder systemglobal
- **Arbeitsmengen** auseinanderreißen kann zum **Seitenflattern** führen
  - $W(t, r)$  umfasst die aktiven Seiten im Fenster der Breite  $r$  zur Zeit  $t$ 
    - beschreibt damit die zur Zeit  $t$  gegebene **Lokalität** eines Prozesses
  - Approximation der Arbeitsmenge mittels Referenzbits und Seitenalter
  - die Berechnung erfolgt auf Basis periodischer Unterbrechungen
- **Umlagerung** bzw. **Vorausladen** immer kompletter Arbeitsmengen
  - Prozesse mit unvollständigen Arbeitsmengen werden ausgesetzt
  - sie unterliegen der langfristigen Einplanung (engl. *long-term scheduling*)

# Literaturverzeichnis

- [1] DENNING, P. J.:  
Thrashing: Its Causes and Prevention.  
In: *AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference (AFIPS '68), December 9–11, 1968, San Francisco, CA, USA* Bd. 33, ACM, 1968 (Part I), S. 915–922
- [2] DENNING, P. J.:  
The Working Set Model for Program Behavior.  
In: *Communications of the ACM* 11 (1968), Mai, Nr. 5, S. 323–333
- [3] DENNING, P. J.:  
Virtual Memory.  
In: *Computing Surveys* 2 (1970), Sept., Nr. 3, S. 153–189
- [4] DENNING, P. J.:  
Working Sets Past and Present.  
In: *IEEE Transactions on Software Engineering* SE-6 (1980), Jan., Nr. 1, S. 64–84