

# ① PC: Programmcounter

Prozessor:

Anfangsadr.

Rückprungadr.

Param

Rückgabe  $\leftarrow$  Announce  
+ Invoke

offen  
geschlo.  
nest

Kommunikation:

Stack

Spezielle Register

Module

• Comps die Dienst bereitstellen

• müssen verwaltet werden

• Modul zu Speicher Zustand

## CALL

1. Rückkehr wird Register : RA; schnell aber flach
2. Rückkehr im Stack langsam aber erlaubt nested

## RET

Überschreibt PC

1 PC := RA;

2 PC := POP;

## PROZ. ERLEUGUNGS LIN

WHY?

- Startauftrag
- Useranmeldung
- Dienstleistungsproz
- Kindproz.

## PROZESSE (nicht zwingend aktiven Kehneingangs)

Proz KONTEXT: Ausführungsstand / Status CPU-Reg. Info

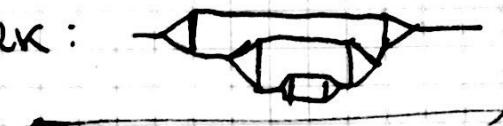
Proz IMAGE: Alle Infos  $\rightarrow$  ein Image die das ganze ding abbildet

UNI Prozess  
sequential ohne Unterbrechung.

MULTI Prozess  
wird die ganze Zeit unter Prozesse geschalten (SCHEDULE PSEUDO PARALLEL)

## MULTIPROCESSING echte Parallelität

## FORK:



DOS Baumstrukt. Keine Parallelität kann aber programme switchen.

WIN XP Echte Prozesshierarchie  $\rightarrow$  handles

HOW?

1. Proc ID wird zugewiesen

2. Speicher für IMAGE zugewiesen

3. PCB initialisiert

4. Links setzen (Zustand)

5. Daten Strukt. evtl. erweitern / ändern

MULTIPROGRAMMING : CPU führt Befehle in Reihenfolge aus. (PC: Program Counter kennt die Reihenfolge)

↳ Programme die eine Reihe aus Maschinenbefehle entsprechen (Trace)

Ausführen mehrerer Progs  
braucht Zustandbeschreibung  
in ihres  
dafür gibt's 3 bekannte Modelle:

2-ZUSTANDS-MODELL



Progs sind beschreibbar.

Spracherinfo muss zufließbar sein

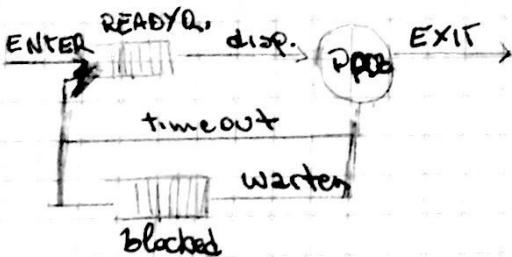
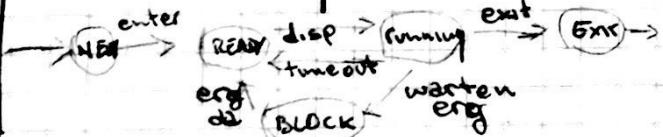
Hat mind 2 Wertespeicher.

5-ZUSTANDS-MODELL

- hat
- FIFO-Queue
- - READY
- - RUNNING
- - BLOCKED
- - NEW
- - EXIT

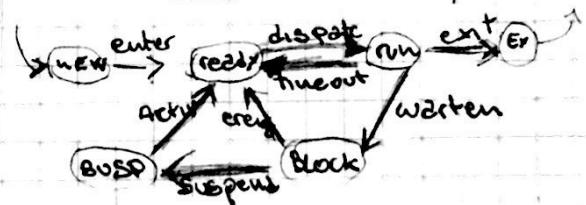
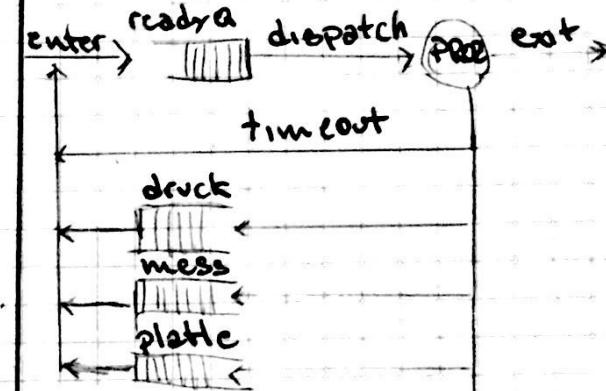
mögliche Zustandsänderung

NULL → NEW  
NEW → READY  
READY → RUNNING  
RUNNING → EXIT  
RUNNING → READY  
RUNNING → BLOCKED  
BLOCKED → READY

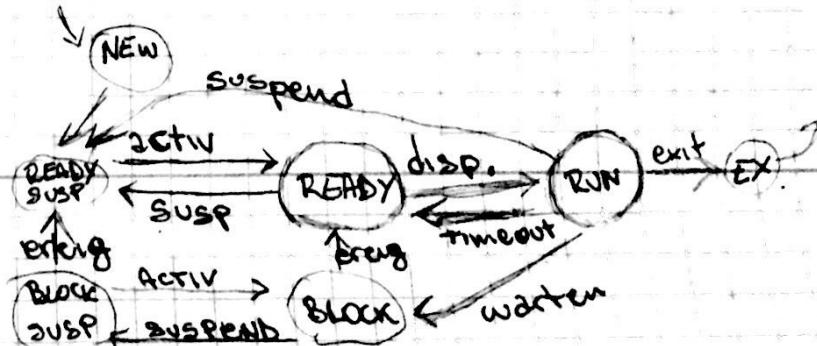


Die Ready Queue wird bei RIO  
in verschiedene Qs unterteilt  
SPLITTING DER READY QUEUE

7-ZUSTAND-MODELL



↓ Verringert mit SWAP



SWAPPING  
block pro Prog werden auf HDD gelagert  
VIRT. SPEICHER  
große SWAP

## PROZESS KONTROLLE

- Frames : Speicherzellen mit end. Größe und adr
- Pages : Proz. im mehreren Seiten aufgeteilt.

## KONTROL STRUKTUREN

- I PROZESS ADDRESSRAUM  
II PROZ KONTROLLBLOCK

1 SPEICHER TABELLEN: Info über Speicherverteilung

2 E/A TABELLEN: Verwaltung von EA Geräten

3 DATEI TABELLEN: Info über Dateien und Dateisyst.

4 PROZ TABELLEN: Info für Verwaltung aller Prozeß im System

## II PROZ. ATTRIBUTE im KONTROLL BLOCK (PCB)

### IDENTIFICATION

- ID
- ID-Parent
- ID-USER

### PROZ. ZUSTAND INFO

- Proc Register: Info running Progs
- Proc Images: Erneut laden. Ausführen Wiederholung
- Heuge der Regs: Program status word (PSW) ALLE im HDD

## PROZ. KONTROLL INFO

### SCHEDULING INFO

- Zustand (run, susp...)
- Prio
- Sched. Strategy
- Ereig auf die gewartet wird

- Datenstrukt, link auf next proc
- Sign, Mess für Proc Intercommunication
- Privileges
- User

## I. DYNAMISCHE PARTITIONIERUNG (→ SEGMENTATION)

BS muss wissen wo proz Images die genze seit abgelegt werden.  
Ganzer Proz muss für Aufführung geladen werden

## I. FESTE PARTITIONIERUNG (→ Tagesg.)

Speicher wird in feste Blöcke Partitioniert gleich groß  
Teile werden in Hauptspeicherframes geladen und ausgeführt  
Proztabelle enthält ein Eintrag pro Prozess

## I. COMPATIBLE TIME-SHARING SYSTEM (CTSS)

### UND BATCH MULTIPROGRAMMING

Batch Proc, für MAX PROZ LAST auto Kontrollengabe

TimeSharing: für SCHNELLE ANTWORT Kontrollengabe aus Terminal

PROZ. IDENT
P-STATE INFO
P CONTROLINFO
USER STACK
PRIVATE USER ADDR, SPACE
SHARED ADDR, SPACE

} PCB

Also können  
Queues als  
Verkettungen  
von PCBs  
verstanden  
werden

# PROZESS KONTROLLE

## PROZESSOR MODE

- SYSTEMMODUS (root)
- USER MODUS

1 Bit in PSW

## THREADS

light weight process  
erledigt eine Aufgabe

## MULTITHREADING

	1 Thread	x Threads
1 Proz	E	SSS
x Proz	S	SS

## 1 PROZESS

- Adressraum IMAGE
- Zugang zu Proz, Dateien, E/A
- Zugriffsrechte

DARIN SIND MEHRERE THREADS

## Bestandteile Thread

- Ausführungsstatus
- Thread-Kontext PC im Speicher
- User Stack
- Kernel Stack
- Speicher für Vars
- Zugriff auf Speicher

Dank dem Threadkonzept ist die Interkommunikation einfacher und der Erzeugung andere Prozesse threads schneller.

- Gemeinsamer Adressraum  $\rightarrow$  Shared memory
- Swapping bewegt ein Prozess mit seinen Threads vollständig in den Speicher

## THREAD ZUSTÄNDE

Im geben das gleiche wie bei Prozesse ohne Auslegern

## THREAD ARTEN

### USER - LEVEL - THREADS (ULT)

BS-Kern kennt ULT nicht

Thread Bibliotheken um Userlevel Management bereit zu stellen.

#### PRO

Innerhalb vom USER SPACE werden kein Modus gewechselt

Eigenen Scheduling

#### CON

Blocking blockiert andere threads

kein Multi-processor

### KERNEL - LEVEL - THREADS

Management durch Betriebssystem

#### PRO

- KLT können schnell erzeugt und getauscht werden

#### MULTIPROCESSING

- Blockiert ein Thread folgt der nächste aus der Ready Queue

#### CON

Bei Kontrollübergabe kann zwischen Modus wechseln

## SCHEDULING entscheidet welcher Proz. als nächstes ausgeführt wird.

- NICHT PREEMPTIV alles wird bis zum Ende ausgeführt
- PREEMPTIV Proz. können jederzeit angehalten werden und neu gestartet werden.  
Das Erfordert Scheduling Algorithmen:

3 Typen von Systemen:

- BATCH System
  - INTERACTIVE System
  - ECHTZEIT System
- } benötigen
- FAIRNESS
  - POLICY ENFORCEMENT
  - BALANCE
  - DATENSICHERHEIT
  - SKALIERBARKEIT
  - EFFIZIENZ

## SCHEDULING VS DISPATCHING

Scheduling: Auswahl der Ready Proz. für Ausführung

Dispatching: Kontextwechsel dispatcher wird vom Scheduler gestartet

## BEGRIFFE SCHEDULING

ANKUNFTSZEIT create time time

STARTZEIT first CPU Zuweisung

VERWEILDAUER Start bis Ende Zeit

ANTWORTZEIT Frage bis Antwort Zeit

BEDIENZEIT Zeit mit statis Running

WARTZEIT Zeit not running

BEENDIGUNGSZEIT Zeitpunkt d. Terminierung

## NICHT PREEMPTIVE ALGS

First Come First Serve : FIFO

Shortest Process Next in. Das Prog mit der kürzeste Bedienzeit wird gestartet

## PREEMPTIVE ALGS

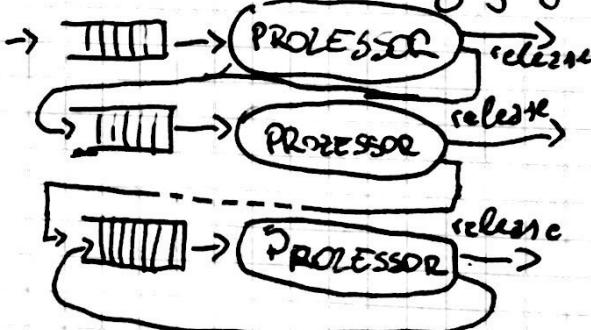
Shortest Remaining Processing Time : Preemptive variaide von SPN

Round Robin : Immer gleiche Zeitschreben abwechselnd an alle Prozesse verteilt.

PRIORITY SCHEDULING: Highest priority first → Nicht so fair

## MULTILEVEL FEEDBACK QUEUING:

n FIFO Queues gefolgt von einer Round Robin



## DEADLOCKS : 2 oder mehrere Programme

befinden sich in wechselseitigen Wartezustand und können nicht befreit werden.

- **WIEDERVERWENDBARE Ressourcen**: können von max 1 Proz. benutzt werden, unendlich oft.

- **VERBRAUCHENDE Ressourcen**: können erzeugt und zerstört werden, und können kein Deadlock

## PREVENTION

### - DIREKTE METHODE

Umstand (Circular Wait)

Geschlossene Kette von Prozessen die immer mind. eine Resource kriegen.

### - INDIREKTE METHODEN

1 Bedingung: wechselseitiger Zuschluss

2 Bedingung: Hold and wait  
Resource wird gehalten und auf eine andere gewartet wird.

3 Bedingung: No Preemption  
Keine Unterbrechung

## AVOIDANCE :

1 Proz. darf nicht gestartet werden PROCESS INITIATION falls seine Anforderungen zu Deadlock führen können. DENIAL

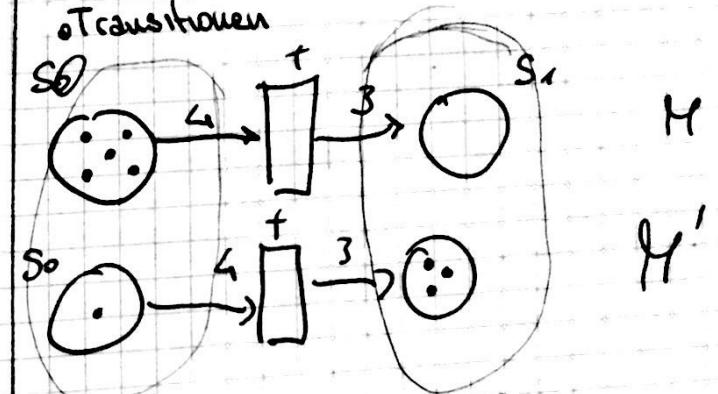
2 Eine Ressourcenanfrage geht nicht RESOURCE ALLOCATION falls sie zu einem Deadlock führen könnte. DENIAL

## PETRI NETZE $\mathcal{P}N = (S, T, F, K, W, M_0)$

besteht aus

• Stellen  
• Transitionen

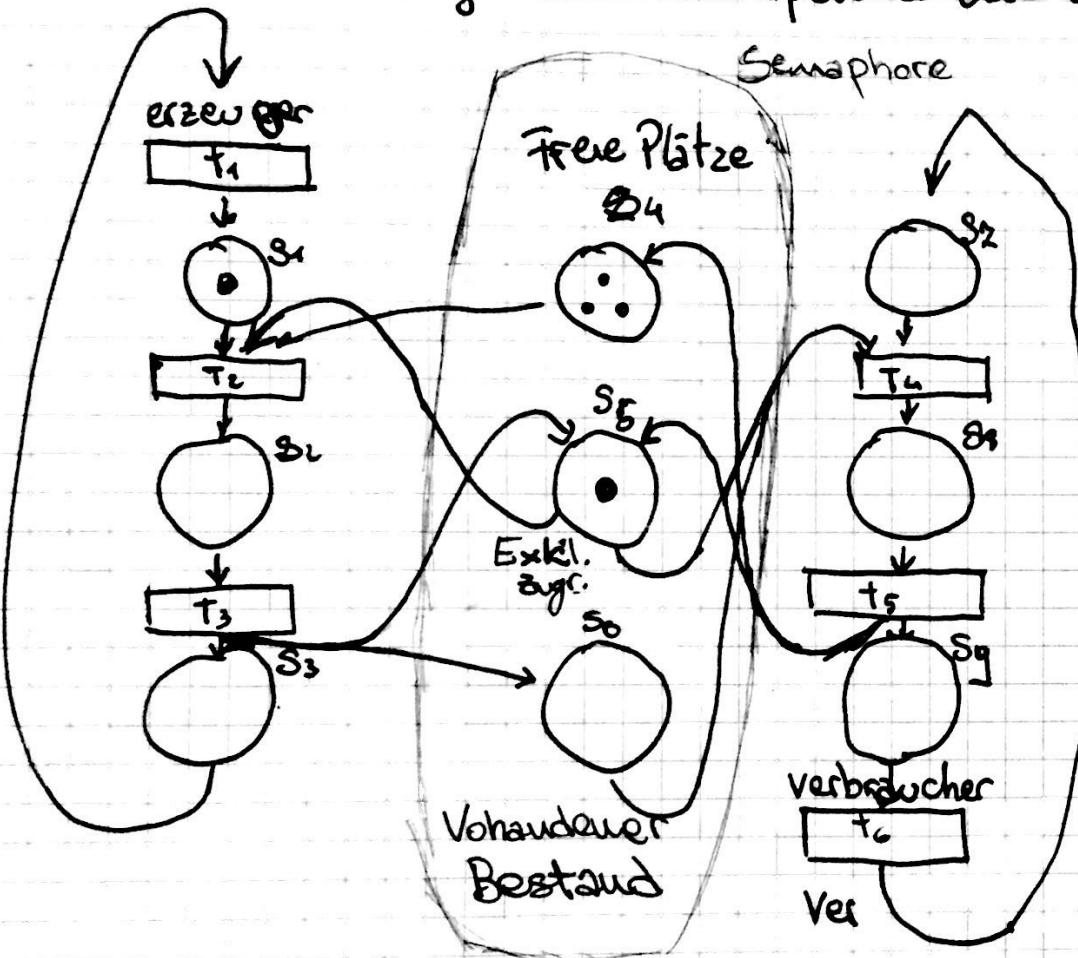
~~stellen~~ Kanten / Kettengraph  
~~transi~~ Kapazität  
Haken



$$M = \begin{cases} M(s) - W(s,t) & t \in s \\ M(s) + W(t,s) & t \in e(t) \setminus (s \cup t) \\ M(s) - W(s,t) + W(t,s) & t \in e(t) \cap (s \cup t) \\ M(s) & t \in s \cup e(t) \end{cases}$$

PETRINETZ SEMAPHORE : Werden gebraucht bei Erzeuger / Verbraucher Problemen.

$\tilde{E}$  und  $V$  haben einen gemeinsamen Speicher und dürfen nur exklusiv lesen oder schreiben.



- Leser muss auf Schreiber warten und umgekehrt
  - Schreiber wartet  $\Rightarrow$  kein neuer Leser kann gestartet werden
  - Jede Karte entspricht ein Prozess

## Nebenräufigkeit von Prozesse

- Unabhängige Abläufe : Parallel
  - Abhängige Abläufe : Kooperation
  - Quasi Parallelität : Stückweise Abarbeitung
  - Parallelität : Echt parallel.

NEBENLAUFIGKEIT = Parallelität + Abhängigkeit

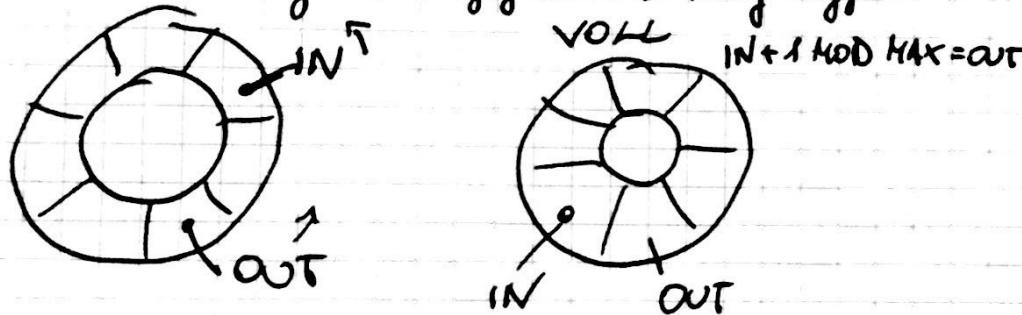
Ablaufverfahren < Sync  
                                    Async

## NEBENLÄUFIGKEIT

KRITISCHE BEREICHE: Die Phase in der Globale benötigt

### Leser/Schreiber Problem

- No Parallel  $\Rightarrow$  Nicht Preemptive Sched
- Vermeidung Abhängigkeit  $\Rightarrow$  Ring-Puffer



### WECHSELSETZER AUSCHLUSS

Wenn ein Proz sich im kritischen Bereich befindet darf kein anderer rein.

- 1 Mutual exklusion: Max 1 Proz im critical
- 2 Progress: Kein Critical es geht weiter mit/Proz
- 3 Bounded Waiting: Proz bleibt nur endl. Zeit im Crit

## AUSSCHLUSS AKS

Decker: Strict Alternation

- 2 Prozesse arbeiten wenn ein Flag . sagt sie sind dran .
- Flag am Ende von Critical section

Schwer zu beweisen also einfacher,

Peterson Algorithmus:

Jeder Prozess besagt wer als nächstes kommt  
Und befret somit den kritischen Bereich  
für das nächste Prozess

SEMAPHORE : Eine Integer Var die durch 3 Atomare Operationen geändert werden kann

- Init (s, AnfangWert) (Init)
- Wert (s) oder P(s) (Inc)
- Signal (s) oder V(s) (Dec)

Wird einer Warteschlange zugeordnet.

### DAS PHILOSOPHEN PROBLEM

5 Philosophen und 5 Stäbchen

2 Stäbchen werden zum essen gebraucht.

Nur ein Philosoph darf essen, um deadlocks zu verhindern.

### SPEICHER

SPEICERVERWALTUNG HAT 5 ANFORDERUNGEN :

- 1 RELOCATION (Wiederaufbau)
- 2 PROTECTION (Schutz)
- 3 SHARING (Teilen/Aufteilen)
- 4 LOGICAL ORGANISATION
- 5 PHYSICAL ORGANISATION

### SPEICHER PARTITIONIERUNG

#### FESTE PARTITIONIERUNG

→ kann zu Fragmentation führen.

#### DYNAMISCHE PARTITIONIERUNG

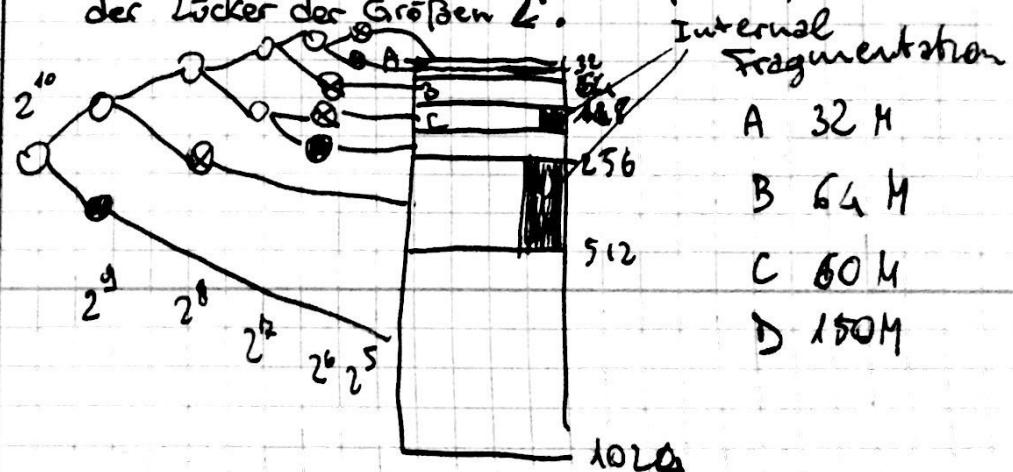
kann zu externe Fragmentierung führen.  
die durch compaction beseitigt werden kann

- Best-Fit kleinste Lücke für 2 Proz reingetragen
- Next-Fit ab letzte Belegung die next passende lücke
- First-Fit ab letzte Belegung die erste passende lücke

### Buddy System

Alternative zu Fest- und Dyn- Partitionierung.

Das Buddy-System verwaltet pro zweierpotenzen eine Liste der freien Speicherplätze, d.h. der Lücken der Größen  $2^i$ .



PAGING : Es werden ausschließlich Speicherblöcke einheitlicher Größe verwendet  
diese werden in Frames geladen.

### 3 Strategien

- Demand Paging : Seite fehlt, wird nachgeladen
- Demand Prepaging : Seite fehlt, werden mehrere geladen
- Look-ahead-paging : Es können Nachladeoperationen stattfinden

### POLICIES

- Resident Set Management Policy : Sagt ob Fixe oder Variable Anz. Pages pro Prog.
- Fetch Policy : Bestimmt ob eine weitere Seite dann nachgeladen wird.
- Placement Policy : Wo ein Teil eines Prog. geschrieben wird (z.B. Best-Fit, First-Fit-)
- Replacement Policy : Alle Frames sind belegt, welche wird befreit um Platz zu machen?
  - Optimal Strategy : Die Seite die als letzte gebraucht wurde ausgelagert
  - FIFO
  - LRU : Die am längsten unbemerkte Ablagerung

### Policies

- Least Frequently Used : niedrigste Nutzungsfrequenz
- Climb :
- Clock Strategie : Verwaltung in einer zyklischen Liste.