Betriebssystem Grundlagen

Teil 2

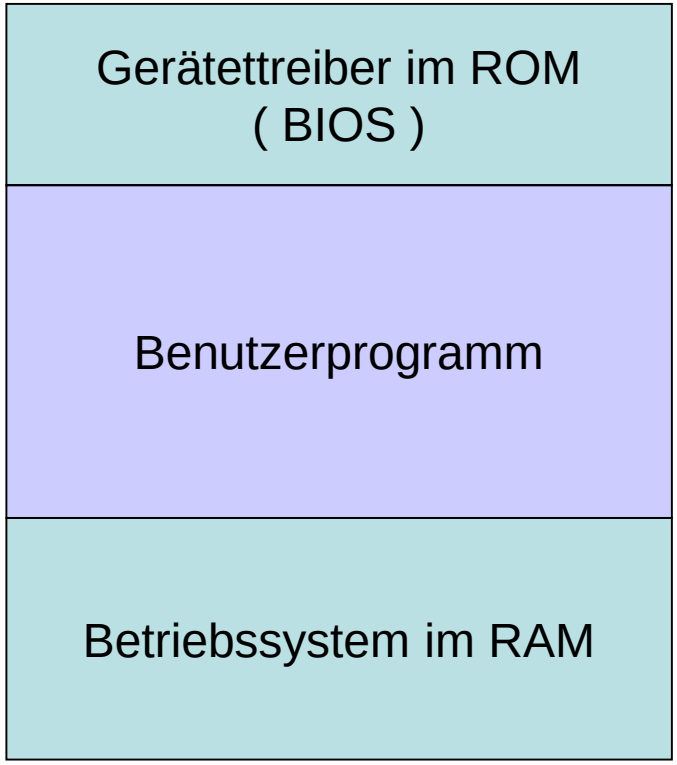
Speicherverwaltung

Paging

Seiten-Ersetzungsstrategien

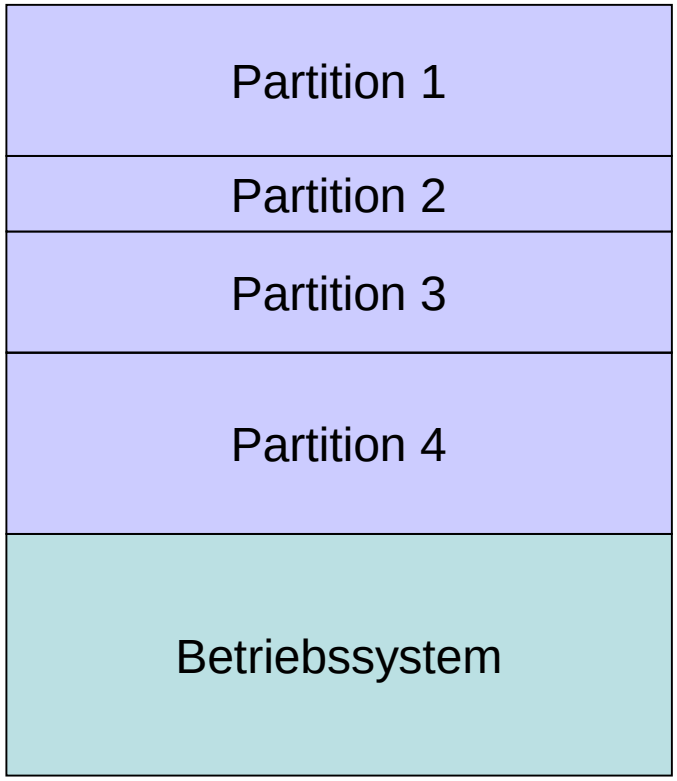
Hans Mühlehner

Speicherverwaltung für MSDOS



Fixer Speicher für OS
und das einzige! laufende
Benutzerprogramm

Speicherverwaltung für OS/MFT (IBM Main Frame OS/360)

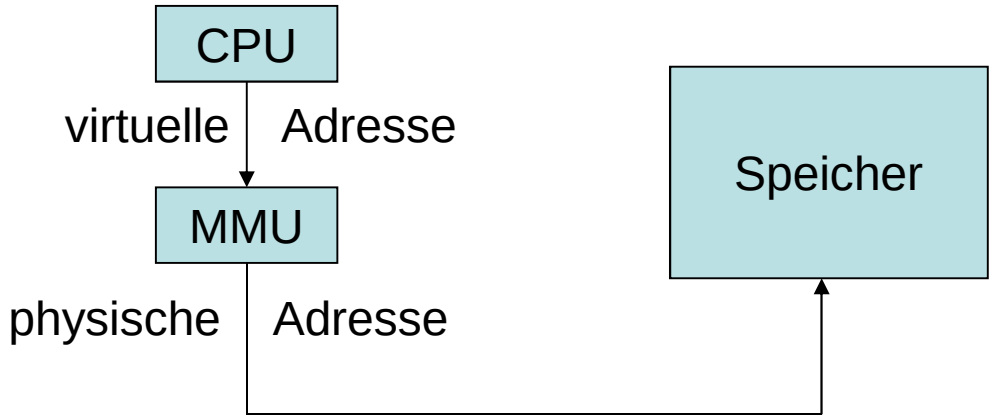


MFT: Multiprograming with Fixed Number of Tasks

In jeder bei Systemstart fix definierten Partition kann ein Programm nach dem anderen geladen und durchgeführt werden

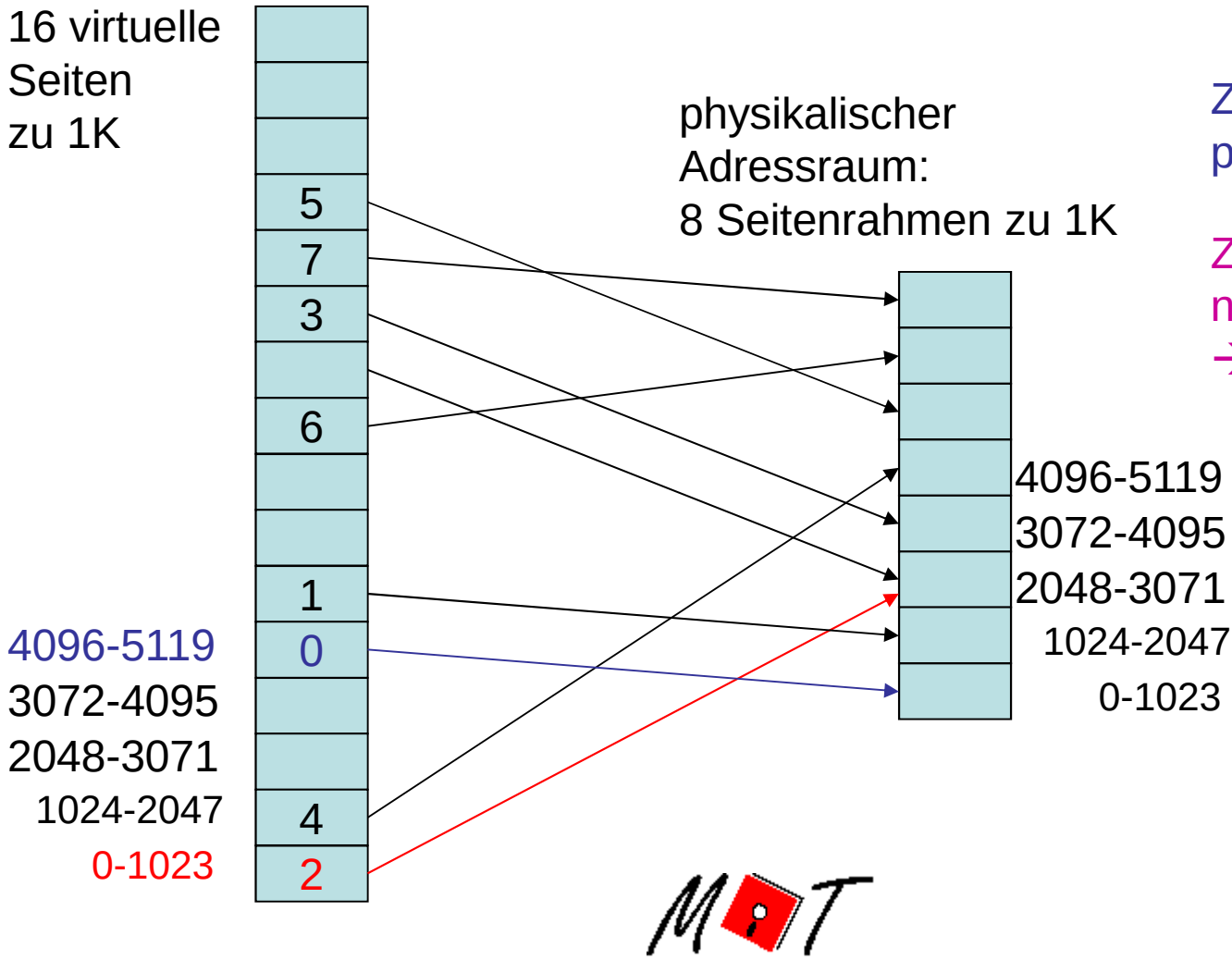
Virtueller Speicher

- Programme können größer sein, als reales RAM
- Nur gerade nötigen Seiten werden geladen
- Einlagern einer Seite (=IO) führt zu Prozesswechsel
- Paging:



Paging

Adressübersetzung mit Seitentabellen
(Seitengröße: 1K)



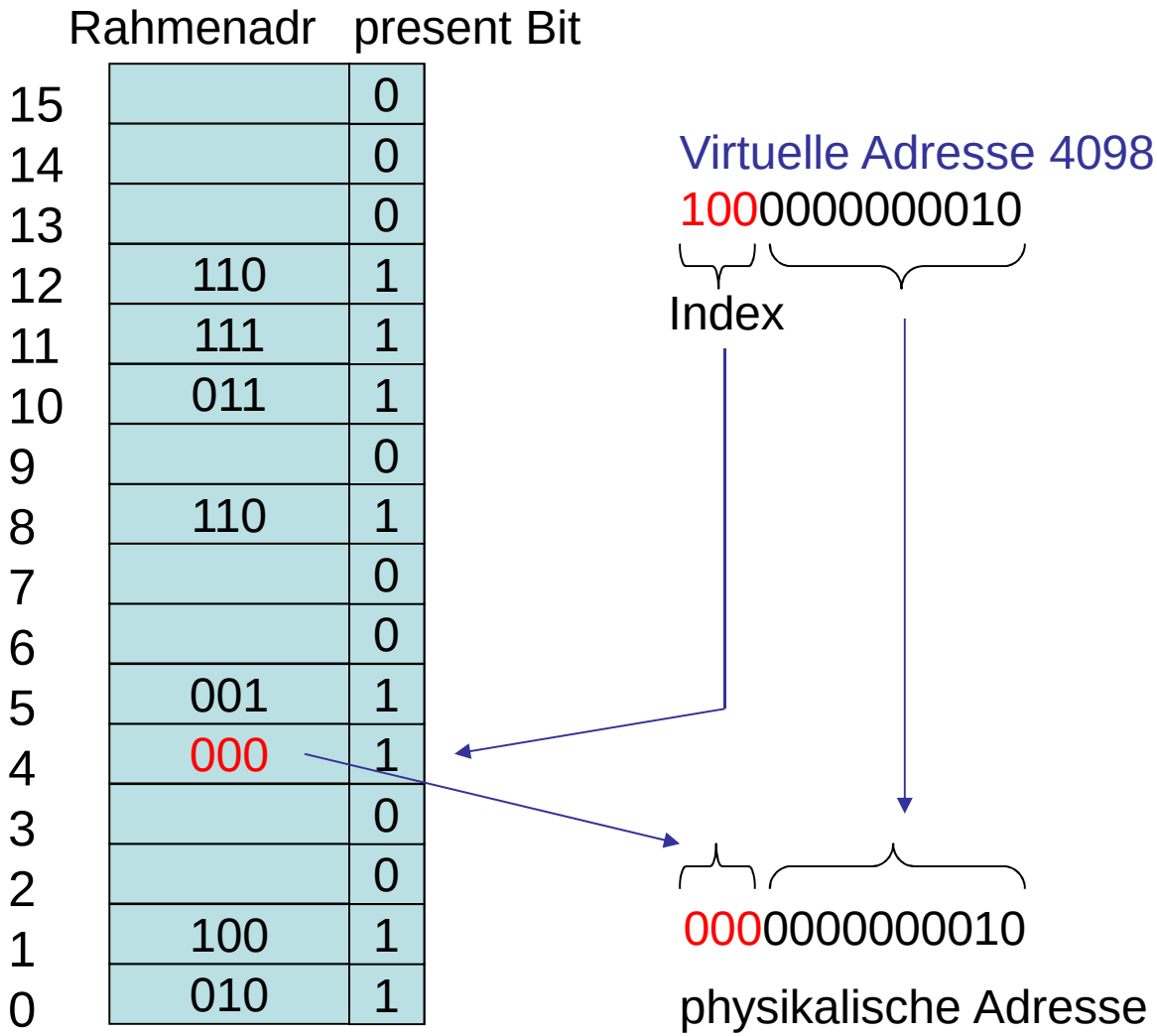
Zugriff auf virt 500:
phys: 500+2048 = 2548

Zugriff auf virt 4098:
phys: 2+0 = 2

Zugriff auf virt 3100:
nicht im Speicher
→ Seitenfehler

Adressübersetzung

Seitentabelle:



Theoretische Größe von Seitentabellen?

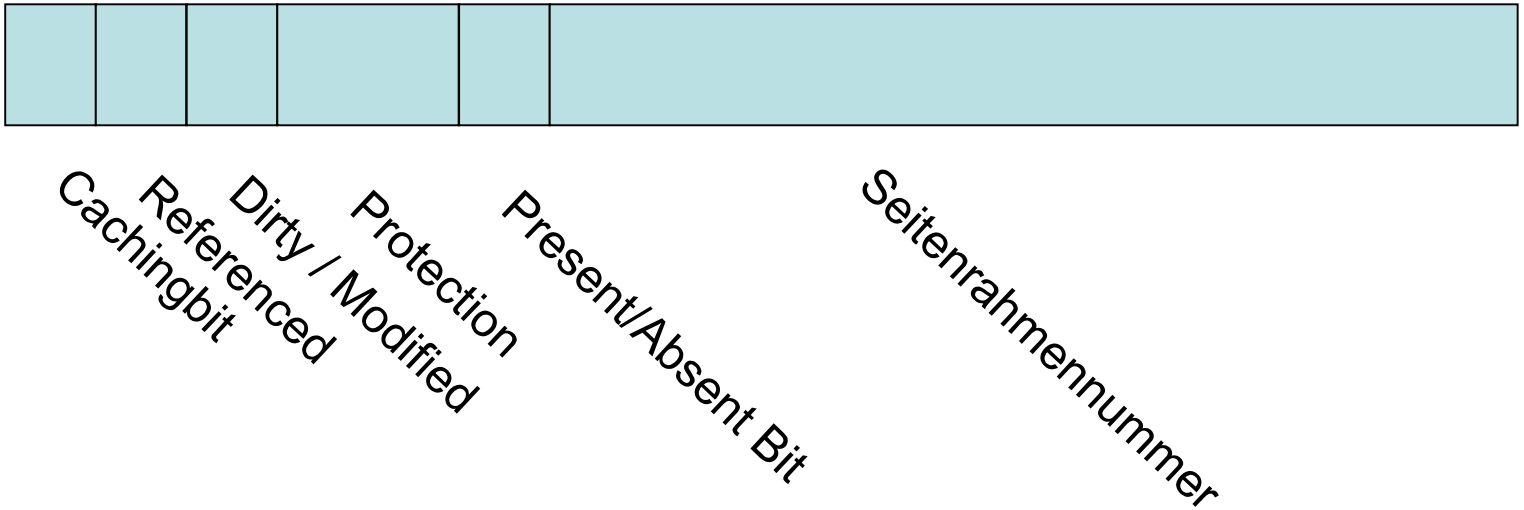
- im 32 Bit System mit 4 KB Speicherrahmen?

32 Bit Adressgröße
→2^{32} Speicherplätze (Byte)
= 4 GB können adressiert werden

2^{12} = 4 KB
→die Adresse im Rahmen ist 12 Bit lang
→bleiben also 20 Bit für die Rahmenadresse

Um 2^{20} (=1 Mega) Adressen speichern zu können, brauchen wir also
1 Mega Einträge in der Seitentabelle mit einer Größe von 20 Bit
für die Adresse + ein paar Bit für weitere Eigenschaften der Seite.
→ (1 M * (3 bis 4 Byte)) = 3-4 MB Größe der Seitentablle eines Prozesses!

Seitentabelleneinträge



Flags werden üblicherweise von der Hardware gesetzt.

Theoretische Größe von Seitentabellen?

- im 64 Bit System mit 8 KB Speicherrahmen?

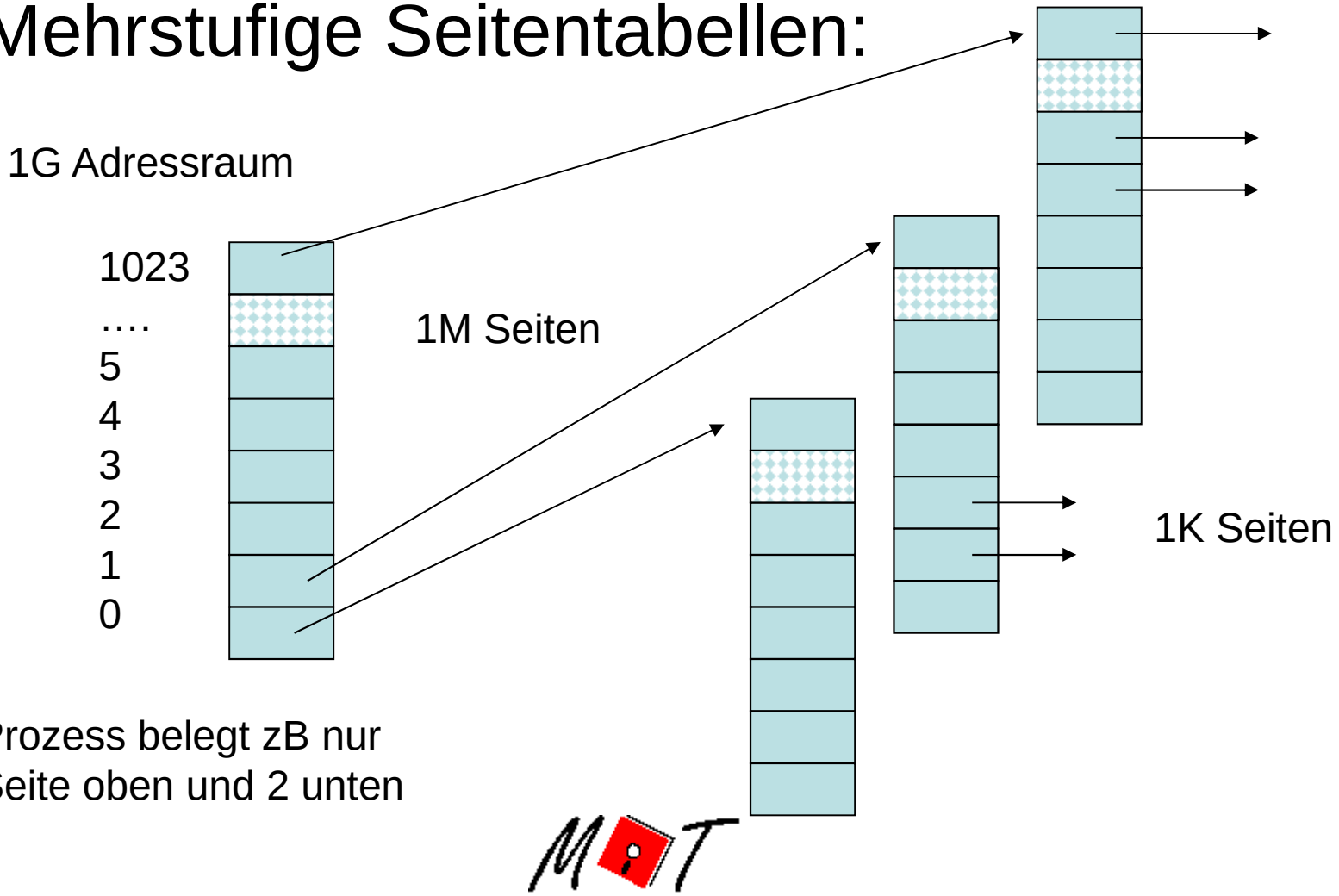
64 Bit Adressgröße
→2^{64} Speicherplätze (Byte)
~ 18.000.000.000.000.000.000

2^{13} = 8 KB
→die Adresse im Rahmen ist 13 Bit lang
→bleiben also 51 Bit für die Rahmenadresse

Um 2^{51} (~10^{15}) Adressen speichern zu können, bräuchten wir also
2^{51} Einträge in der Seitentabelle mit einer Größe von 51 Bit
für die Adresse + ein paar Bit für weitere Eigenschaften der Seite.
→ (2^{51} * (7 od. 8 Byte) ---- soviel Speicher gibts derzeit nicht!

Damit Seitentabellen im Memory nicht zu groß werden:

- Mehrstufige Seitentabellen:

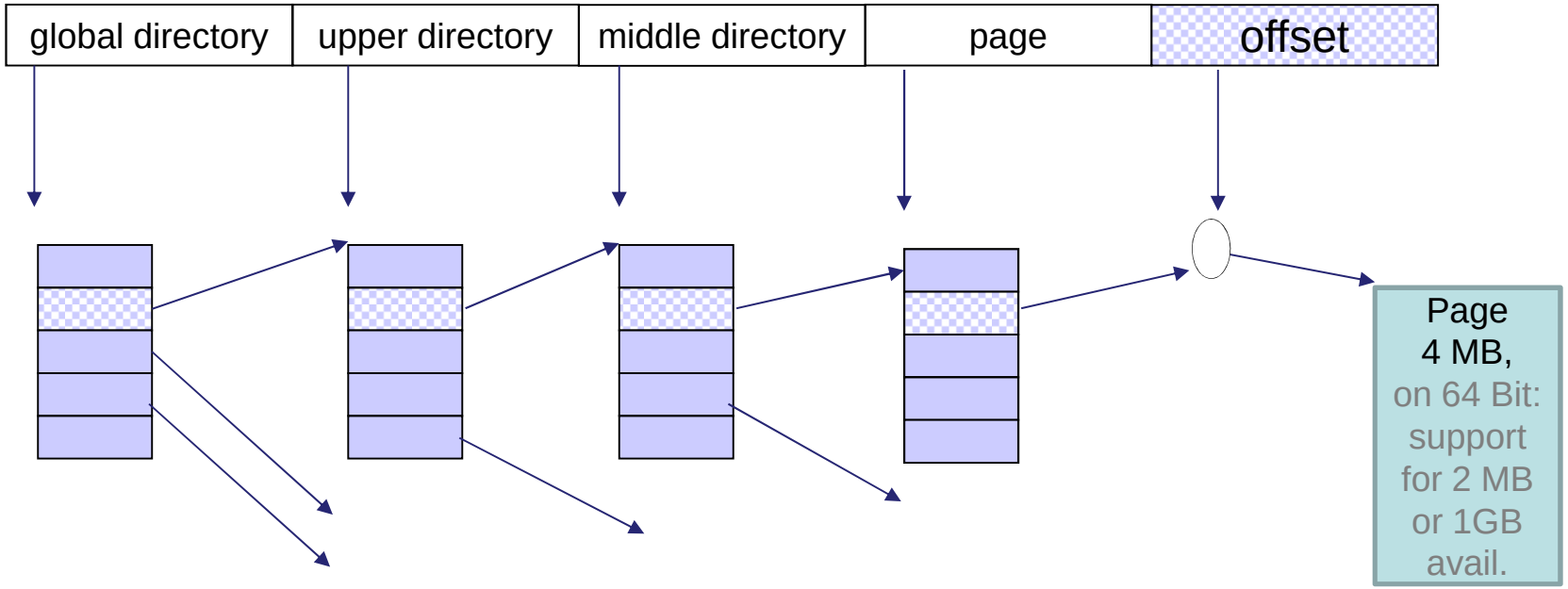


1 Prozess belegt zB nur
1 Seite oben und 2 unten

4-stufige Seitentabellen in Linux

(upper und middle dir können “leer” sein zB in 32-Bit Systemen)

virt. Adresse:



Invertierte Seitentabellen

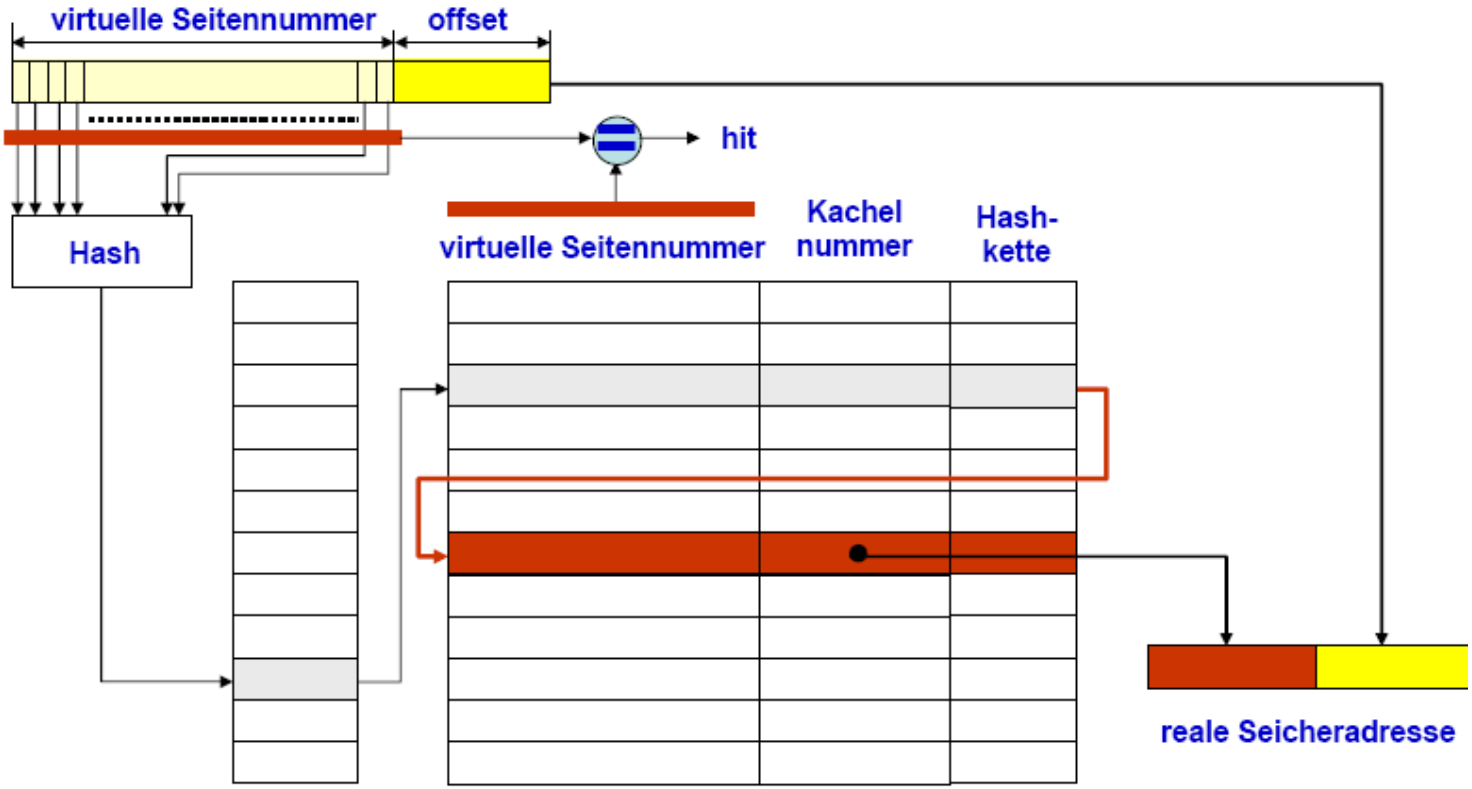
Stellen eine weitere Möglichkeit dar, mit der Größe der Seitentabellen umzugehen.

- Nicht alle virtuellen Adressen stehen in der Seitentabelle, sondern die physikalischen
- Problem Suchen: eine Virtuelle Adresse ergibt nun nicht den Index in der Seitentabelle
    - Lösungsansätze:
        - Große TLB (mit 1 Anweisung durchsuchbar)
        - Hash (virt. Adr) liefert Eintrag in Seitentabelle

- Bei 64Bit Adressraum kann eine komplette Seitentabelle nicht mehr im Speicher stehen! (Bei 4K Seiten: 2^{52} Einträge mit 8 Byte = 30 Mio GB

invertierte Seitentabelle

PPC, AS/400



aus: J. Kaiser VO Betriebssyteme Magdeburg

Translation Lookaside Buffer

- Teil der MMU
- Speichert (meist 32, 64 oder 128 Einträge der Seitentabelle,

  die zuletzt verwendet wurden
- Kann von der Hardware gleichzeitig mit virt.Adresse verglichen werden
- TLB wird vom Betriebssystem oder der Hardware verwaltet

gültig	virt. Seite	verändert	Schutz	Rahmen
1	150	0	RX	33
1	456	1	RW	23
1	33	1	RW	12

- Swapping
  - Aus- und Einlagern ganzer Prozesse (Unix Terminologie, in Linux nicht implementiert)
- Paging
  - Aus- und Einlagern einzelner Seiten

- Wie findet OS freie oder besetzte Seiten?
  - Speicherverwaltung mit Bitmaps
  - Speicherverwaltung mit verketteten Listen

Seitenersetzung

Nötig bei

- Seitenfehler
- CPU-Cachefehler
- Cachefehler bei Datenbanken, Webserver

- Clean page: einfach Überschreiben
- Dirty page: zuerst Auslagern, dann Überschr.

Wann werden Seiten geladen?

- Alles beim Programmstart
- Demand Paging

  (Alle) Seiten werden auf Anlass eines Page Faults geladen

  Typisch in moderenen OS zB Unix, Windows NT

  einfach zu implementieren
- Prepaging

  Seiten werden im Voraus geladen in der Annahme, dass sie gebraucht werden.

  zB bei MS seit XP(NT 5.1) verwendet, "Vista Superfetch"

  "dabei lernt das OS Speicherverhalten von Prozessen"

  relativ schwieri g zu implementieren

Wann werden Seiten freigegeben / ausgelagert?

- Beim Prozessende
- Demand Cleaning

  Vorteil: Memory wird voll ausgenützt bevor Paging Activity auftritt

  Nachteil: Wenn Pagefault auftritt muss nicht nur Seite eingelagert werden, sondern auch noch vorher freigemacht werden

- Precleaning

  Vorteil: Bei Pagefault ist sofort freies Memory verfügbar

  Nachteil: Es wird Ausgelagert, obwohl ev. nicht nötig

  Fast alle modernen Systeme verwenden Precleaning, wobei in der Regel die Intensität des “Freischaufelns” vom “Demand” abhängt.

Seitenersetzungsstrategien

- Not Recently Used
- First In First Out
- Second Chance Algorithm
- Clock Algorithm
- Least Recently Used
- Aging
- Working Set Algorithm
- WSClock Seitenersetzung

Seitenersetzung
Not Recently Used Algorithm

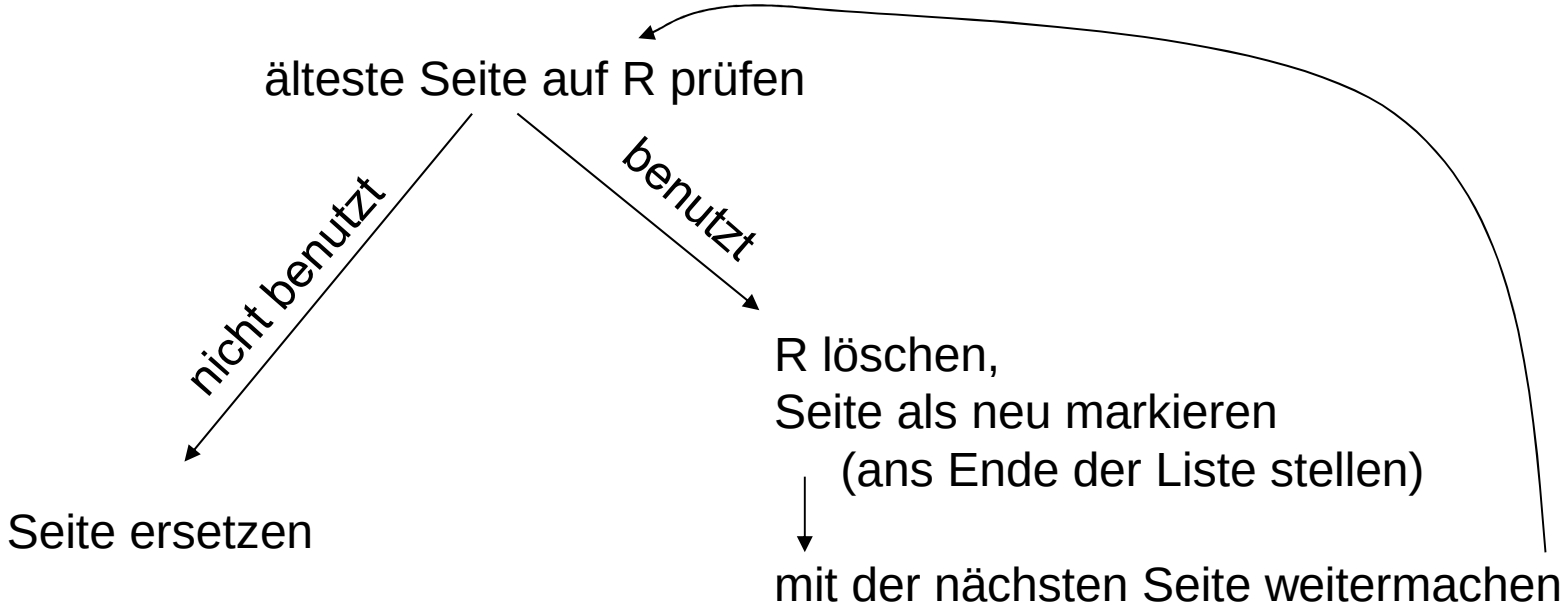
- Arbeitet mit dem Referenced und Modified Bit (beide von der Hardware gesetzt und Teil der Seitentabelleneinträge)
- Periodisches Löschen der Referenced Bits (~20ms)
- ergibt folgende Klassifizierung der Seiten:
  - nicht benutzt, nicht geändert
  - nicht benutzt, geändert
  - benutzt, nicht geändert
  - benutzt, geändert

ziemlich wenig treffsicher!!!

- ersetzt wird eine beliebige Seite der niedrigsten Klasse
- einfach zu implementieren!

Seitenersetzung
Second Chance Algorithm

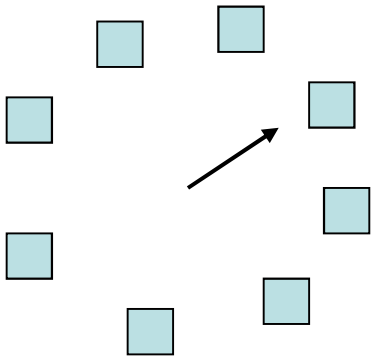
- FIFO + Berücksichtigung des (R)eferenced Bits



- guter, ineffizienter Algorithmus (Verschieben in der Liste!)

Seitenersetzung
Clock Algorithm

- Wie Second Chance, nur werden die Seiten in einer ringförmigen Liste gehalten, der „Uhrzeiger“ zeigt immer auf die älteste Seite.
- Bei Seitenfehler:
  - Überprüfen des „R“ Bit der Seite, auf die Zeiger Zeigt
    - bei ungenutzt austauschen,
    - bei benutzt: auf unbenutzt setzen, Zeiger vorrücken und



Seitenersetzung
Least Recently Used

Zuletzt häufiger benutzte Seiten werden vermutlich auch in den nächsten Befehlen gebraucht.

Möglichkeiten:

- Alle Seiten in einer Liste ordnen nach Benutzungszeitpunkt
- Großer Zähler für alle Seiten nötig, der bei jedem Zugriff aktualisiert werden muss

Aufwendig, kaum realisierbar!

Seitenersetzung
Aging

- Für jeden Seitenrahmen wird ein zB 8-Bit „Aging-Zähler“ definiert
- In Intervallen (zB alle 20 ms) werden die R-Bits vorne in den Aging Zähler geschoben und gelöscht
- Die Seite, mit dem niedrigsten Zählerstand kann ausgewechselt werden

vorher:

R	Aging Zähler
1	10010010
0	00110010
0	10000010
1	00000000

nachher:

R	Aging Zähler
0	11001001
0	00011001
0	01000001
0	10000000

Seitenersetzung
Working Set Algorithm

Workingset: Ist die Menge an Seiten, die ein Prozess in einem definierten Zeitfenster benutzt hat

- Versucht Seiten auszulagern, die nicht zum „Working Set“ eines Prozesses gehören

Möglicher Algorithmus:

Durchlaufe alle Seiten-Deskriptoren und

- if R==1: set vt to current_time and set R=0;
- if R==0 und (current_time - vt) > wss: verdränge die Seite;
- if R==0 und (cvt - vt) < wss : keine Änderung;
- wenn keine Seite gefunden wird mit:

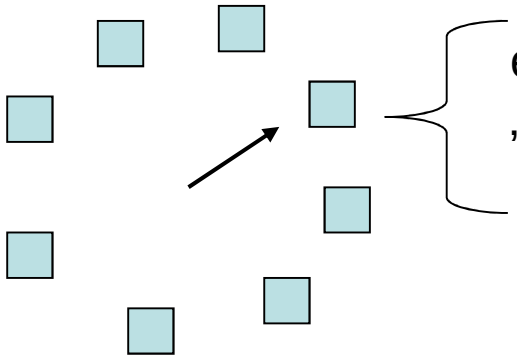
  R==0 und (cvt - vt) > wss : verdränge älteste Seite;
- wenn alle Seiten referenziert wurden: verdränge beliebige Seite.

Jeder Seitendescriptor benötigt:
vt … die (virt.) Zeit des letzten Zugriffs
R … das Referenced Bit

wss …. ist die “working set size”, also das maximale Alter der Seite um zum WS zu gehören

Seitenersetzung
WSClock Algorithm

- Kompbination aus Clock und Working Set Algorithm



eine Seitenbeschreibung enthält das
„R“ (referenzed) und einen Eintrag für die Zeit

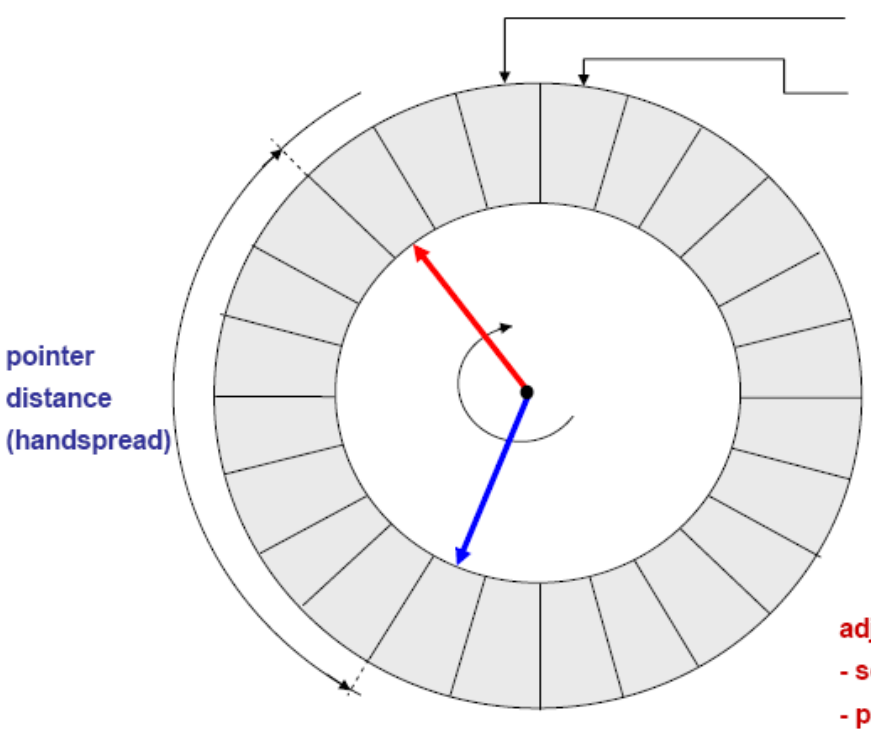
1	2342

Auslagerungskandidaten sind:
- Unbenutzt (R nicht gesetzt) und
- Zeit seit letztem Zugrifffs > wss (lt.Workingset Definition)

Zusätzliches Kriterium: Not Modified

2-händiger Uhrenalgorithm (vgl: Clock) (**Unix SVR4**)

releasing pages: clock with two pointers



scans list of pages and clears the R-Bit

scans list of pages and moves pages with R=0 to the free list (pages that can be replaced) or replaces pages if necessary.

adjustable parameters for the algorithm:
- scan rate
- pointer distance

Merkmale der in Unix Systemen verwendeten Speichertechniken

- Einfache Ersetzungsstrategien, die effizient implementierbar sind aber nicht unbedingt optimale Treffsicherheit aufweisen (2-handed Clock Algorithm)
- Paging mit global Scope (normalerweise werden Seiten eines beliebigen Prozesses ausgelagert) und ev. Swapping (ganze Prozesse wenns wirklich eng wird im Speicher)
- Kerneltunables ermöglichen Einstellungen über die Aggressivität des Precleanings
- Pures Demandpaging

Linux PFRA (Page Frame Reclaiming Algorithm)

4 Types of Pages:

- Unreclaimable (Locked pages, kernel.. - not pageable)
- Swappable (Anonymous user-mode pages - Write to swap area)
- Syncable (Parts of files -Write to file if changed)
- Discardable (Unused pages can be reclaimed without action)

preferred use

Special Clock Algorithm with 2 LRU Lists:

- Active List
- Inactive List

works with Referenced Bit:

inactive and 2 times referenced → move to active list

active and 2 times not referended → move to inactie list

if memory is very low, any pages can be demoted

Is running as a

kswapd Process for each memory node

(and a pdflush daemon [writes out dirty pages] if memory is low)

Linux physical Memory Management

Entire kernel and memory maps are pinned in memory!

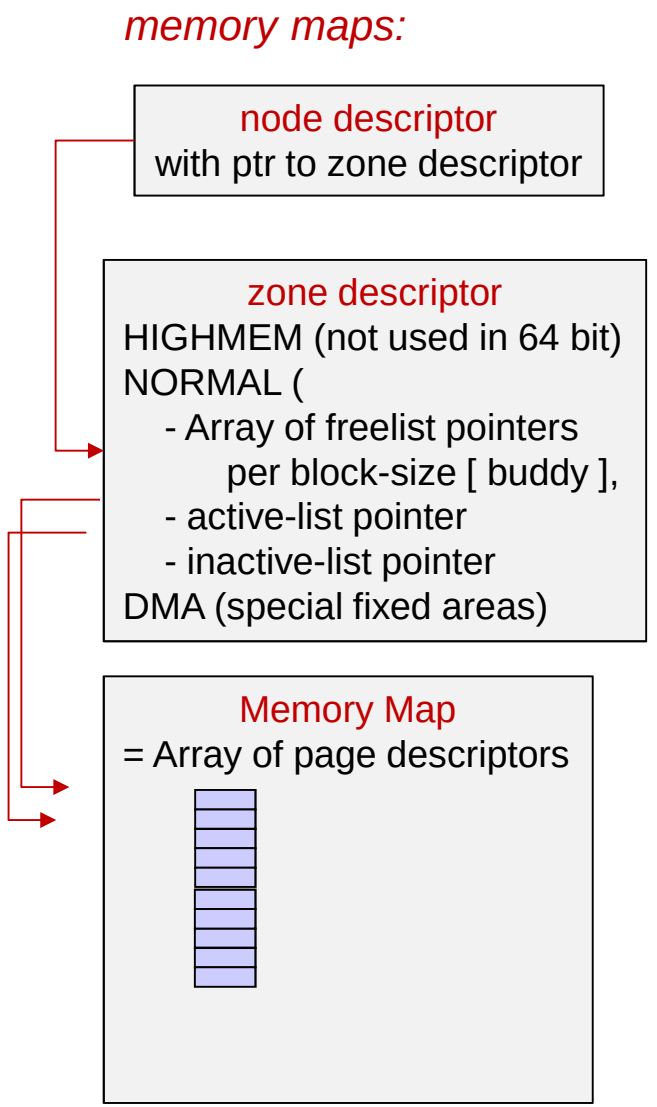
NUMA Support: 1 node-descriptor per NUMA-Node

page descriptor (32 bytes per phys. page):

- has pointer to used/mapped address space
- or pointer to next and prev free block of same size

Linux page allocater uses buddy algotrithm

→ free areas are managed in 2^i sized blocks



Merkmale der in MS NT Systemen verwendeten Speichertechniken

- Seitengröße ist 2er-Potenz: Default 4KB. Large Pages 2MB
- Speicher eines Prozesses besteht aus Regions (zB für Stack, Data, DLL, Programm…)
- Region wird im Virtual Address Descriptor (VAD) beschrieben:
  - Adress-Range
  - Backingstore Info
  - Rechte, ….
- Prozess hat balanzierten Baum mit seinen VADs
- Beim 1. Zugriff auf eine Seite einer Region wird die Pagetable dazu geladen und im Prozeßobjekt eingetragen
- Pagetable Entries [64 Bit] haben viele Eigenschaften, zB:
  - Dirty und Accessed Bit [von Hardware gesetzt → aging for WS Implementation]
  - physical Pagenumber [40 Bit]
  - verschiedene Permissionbits, Cacheinfos, Copy-on-write Info, …

Merkmale der in MS NT Systemen verwendeten Speichertechniken

- Workingset Algorithmus (komplex, aufwendig, dafür relativ treffsicher)
  - Workingsetverfahren mit “Variable Allocation, Local Scope”:
    - Bei einem neuen Prozess wird je nach Programmtyp bzw. –anforderung die Anzahl der Rahmenseiten bestimmt
    - **Working set manager:** Wenn ein Seitenfehler auftritt, wird eine Seite aus einem dazuhörigen Pool des Prozess gewählt.
    - **Balance set manager:** Periodische Neubewertung der Allokationen
- Nicht nur Demandpaging, sondern seit XP /2003 auch Prepaging – ab Vista mit einem Feature namens “superfetch” erweitert.

Windows NT - Store Manager (Unified Caching)

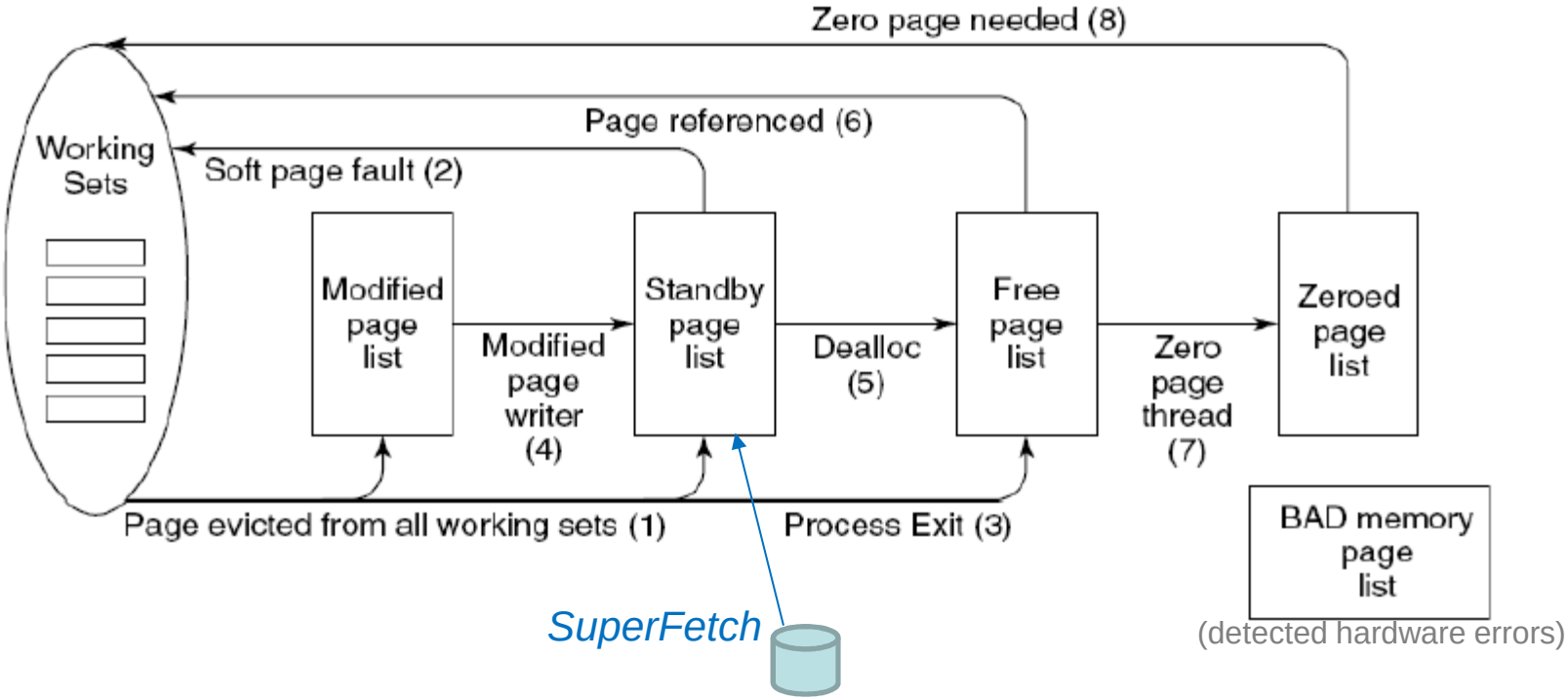
- seit NT 6.3 (Windows 8 / Server 2012) implementiert
- Berücksichtigt Eigenschaften von Backing Storages (Speichergeräte, z.B: HD, Floppy, CD, DVD, Memory Stick, Magnetbänder)
- Superfetch kann zB anstatt alles in den RAM Teile auf auf schnelle SSD laden
- Optimiert die Zugriffe durch bevorzugen von schnelleren HDs (SSD oder Flash Memory) innerhalb der Pagefile-Konfiguration bzw. innerhalb einer Volume-Konfiguration für FS etc.
- Schiebt im Hintergrund länger nicht verwendeten Content auf den langsameren Speicher,

Was passiert mit Seiten, die Freigegeben werden?

- Replaced pages are placed in two lists
  - Modified and unmodified
- Pages in the modified list are periodically written out [in Unix zB pageout Daemon)
- Pages in the unmodified list are either reclaimed if referenced again or lost when its frame is assigned to another page
- Pages should be “nulled” for security reason before they can be used by another process

Various Lists in Windows page-replacement

stored in the PageFrameNumber Database



Windows page-replacement

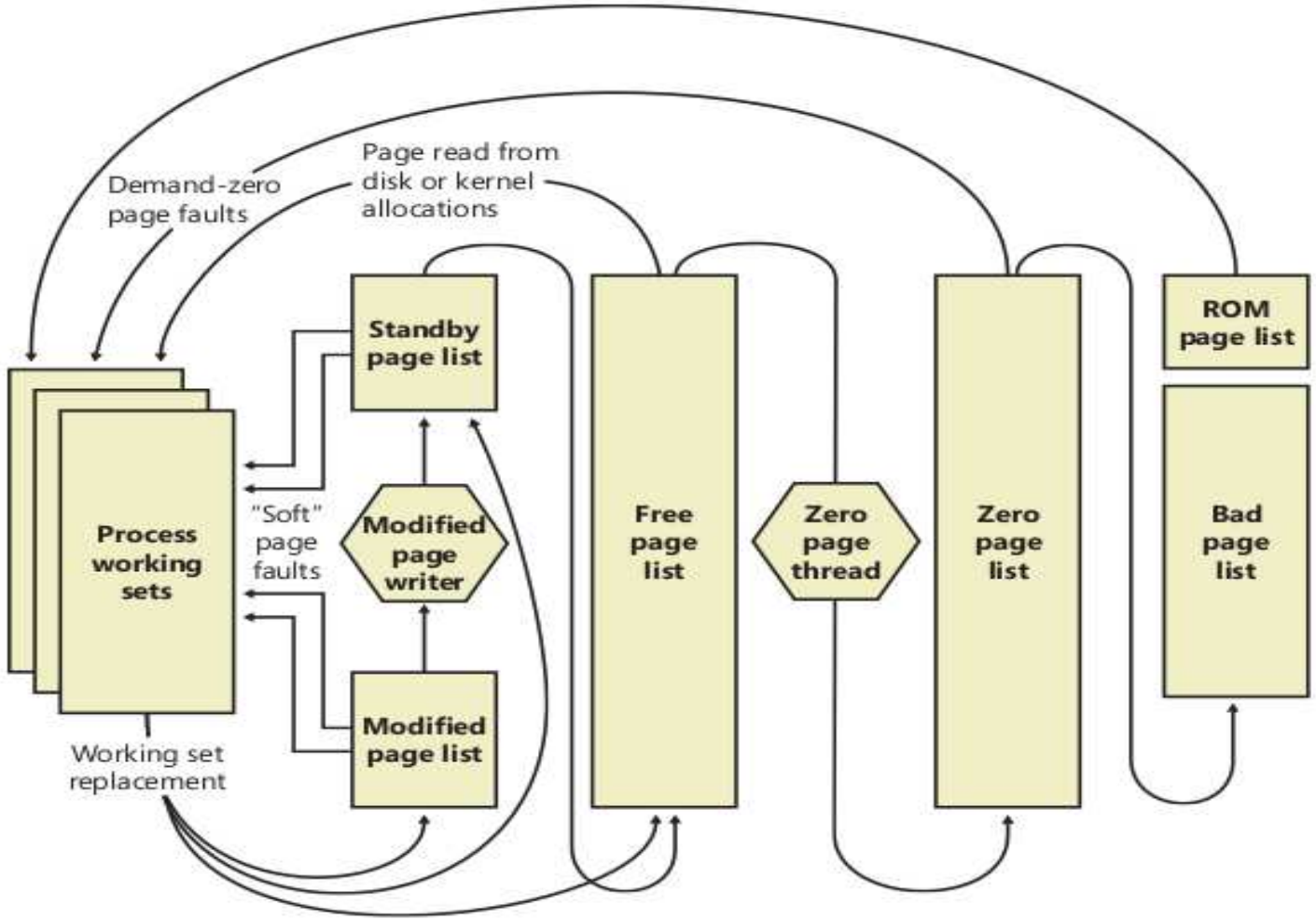
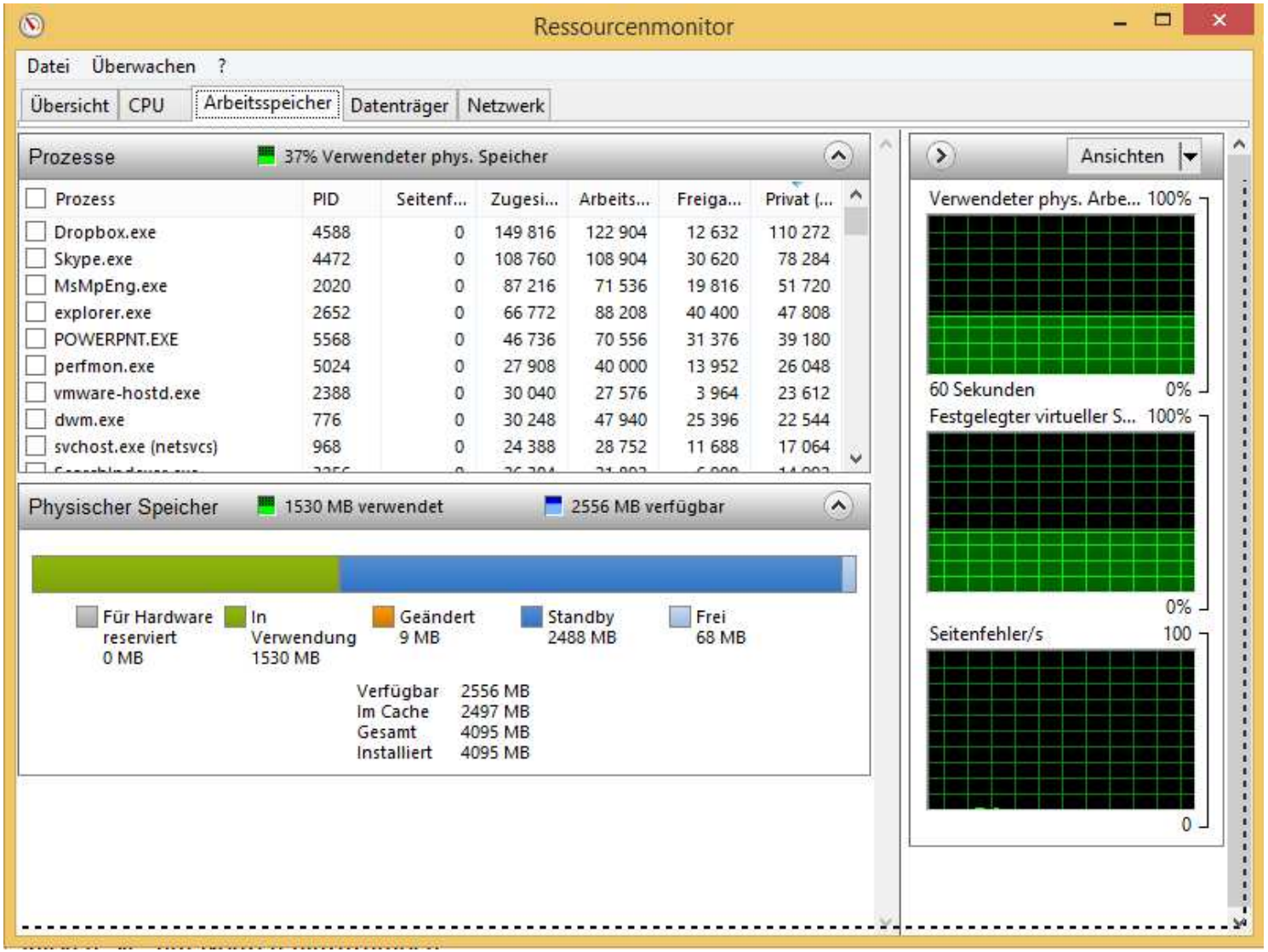


FIGURE 10-39 State diagram for page frames

(Quelle: Windows Internals Part 2 6th Edition, Seite 301)

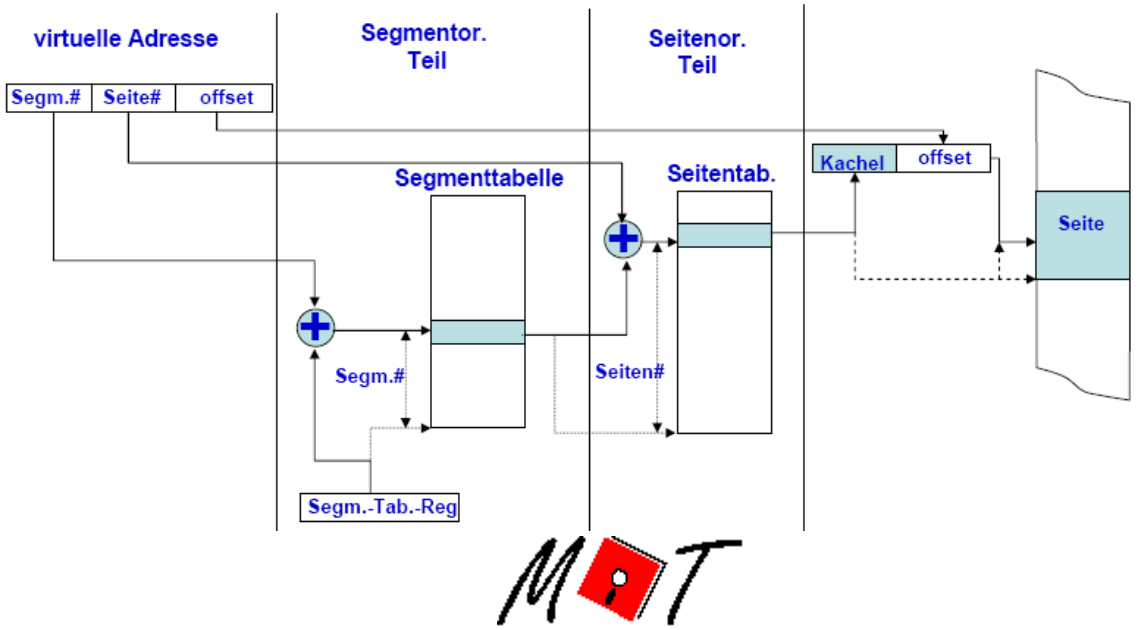
Windows page-replacement



Segmentierung

Wenn die Speicherbereiche, die adressiert werden nicht gleich große Seiten (pages) sind, spricht man von Segmentierung.

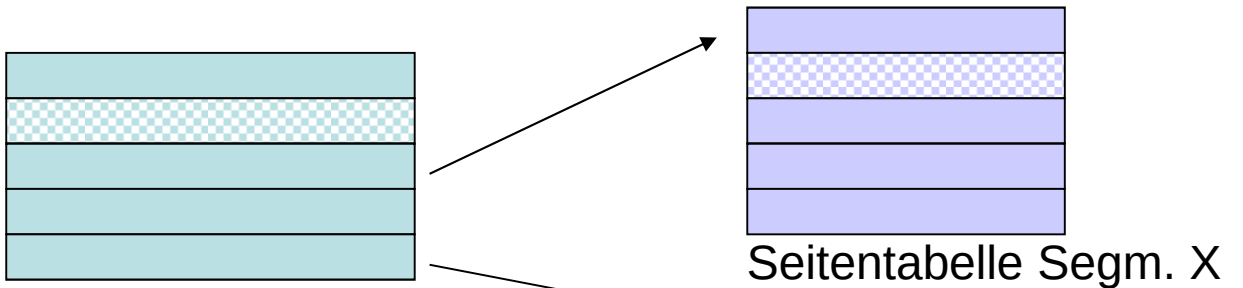
- getrennte Segmente für: Stack, Daten, Code, shared Library, Konstanten, …
- jedes Segment bildet „eigenen Adressraum“
- Segment hat variable Größe, die unabhängig geändert werden kann
- Segmentadressierung kann mit Seitenadressierung kombiniert werden:



Segmentierung und Paging
Beispiel MULTICS

Jeder Prozess kann 2^{18} Segmente haben

Jedes Segment ist ein virtueller Speicher mit max. 65536 Worten

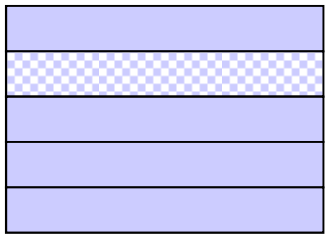


Seitentabelle Segm. X

Segmenttabelle

enhält Segmentdeskriptoren mit:

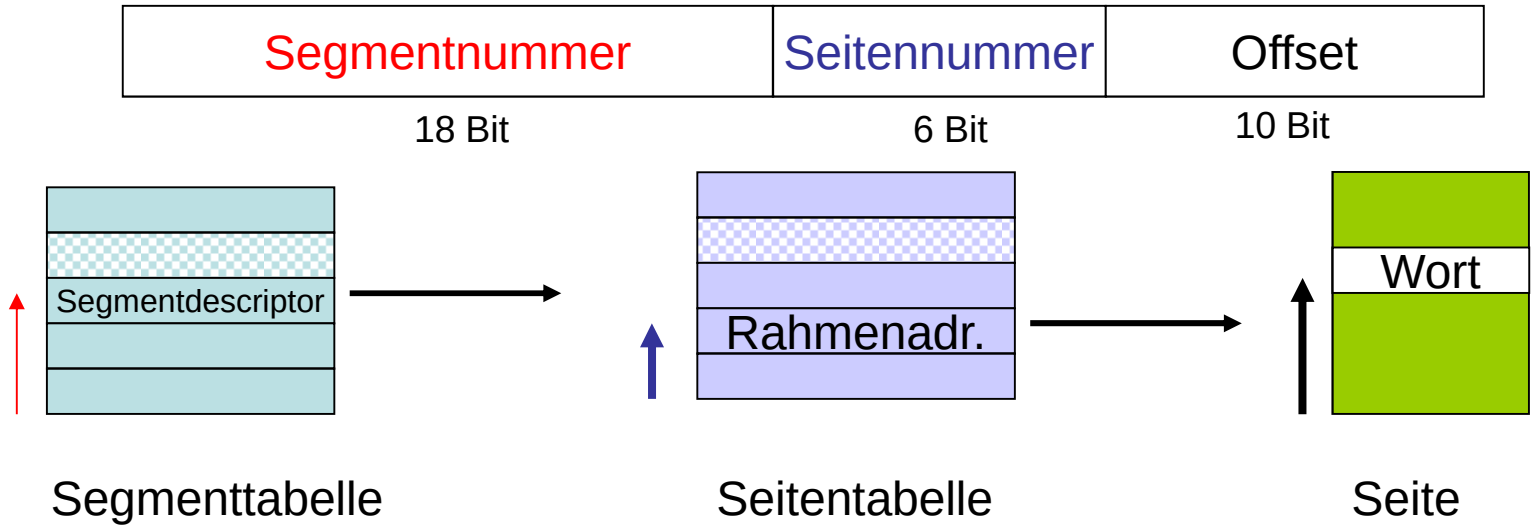
- Seitentabellenadresse,
- Seitengröße
- vorhanden?
- Schutzbits



Seitentabelle Segm. Y

Segmentierung und Paging
Beispiel MULTICS

Aufbau der virtuellen Adresse:



+ TLB mit den letzten 16 Seitenadressen!!

Segmentierung und Paging Beispiel Intel

verwendet auch TLB für die letzten Seitenadressen!!

- Unterstützt 16K Segmente (8K lokal im Prozess und 8K global im System) mit max. je 1G 32Bit-Worten (4GB)
- Virtuelle Adresse besteht aus: (Selektor, Offset)
- Ein Selektor adressiert einen Eintrag (den PentiumDeskriptor bzw. Segment Deskriptor) in der (lokalen oder globalen) SegmentDeskriptorTabele (LDT,GDT).
- Der Pentiumdeskriptor (8 Byte) enthält eine Basisadresse, ein Limit (Länge), Privilege Level (0-3), Seg.Type / Protection, …
- Basisadresse + Offset ergeben die „lineare Adresse“

Segmentierung und Paging
Beispiel Intel x86

Betrieb mit abgeschaltetem Paging:

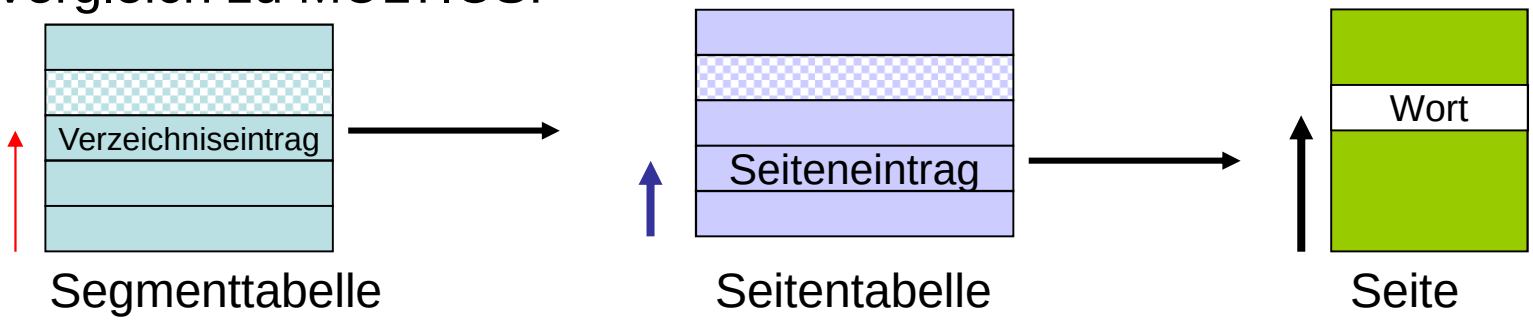
lineare Adresse = physikalische Adresse

Betrieb mit Paging:

2-stufiges Paging:

Lineare Adresse:

Vergleich zu MULTICS:



Segmentierung und Paging
Beispiel Intel x86 (32-bit)

Verwendung der Segmente bei Intel:

Alle heutigen Betriebssysteme verwenden Sie nicht (sondern nur für Zugriffsschutz (siehe nächste Folie))

Sie arbeiten in einem einzigen 32-Bit Adressraum (in alle Segmentregister wird gleicher Wert geladen)

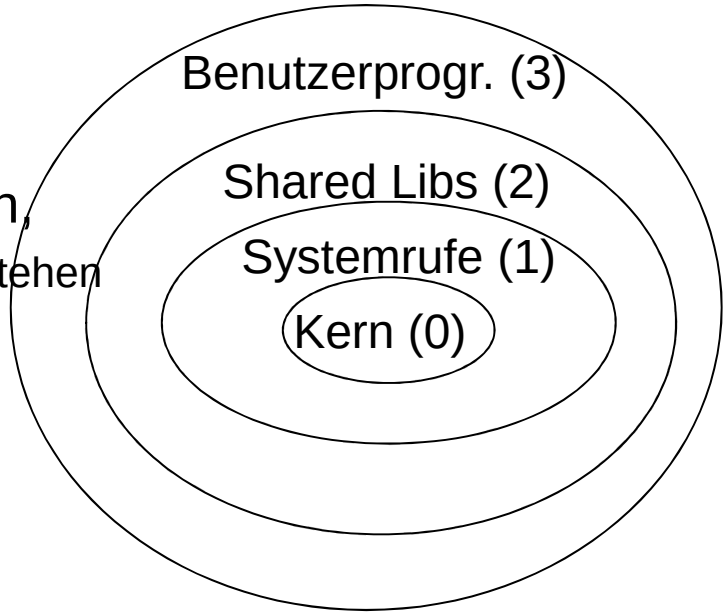
Einzige Ausnahme war OS/2

Schutz: 4 Zugriffsschutzebenen

Descriptor Privilege Level / DPL

Codiert in 2 Bit in den Segmentregistern,

in denen auch die Selektoren für die Segmente stehen



Segmentierung und Paging
Beispiel Intel x86 (32-bit)

Verwendung der Segmente unter Linux:

Linux definiert vier Haupt-Segmente für den Zugriffsschutz

Segment	Base	G	Limit	S	Type	DPL	D/B	P
User Code	0x00000000	1	0xfffff	1	10	3	1	1
User Data	0x00000000	1	0xfffff	1	2	3	1	1
Kernel Code	0x00000000	1	0xfffff	1	10	0	1	1
Kernel Data	0x00000000	1	0xfffff	1	2	0	1	1

G … Granularity flag: 0 → Limit in Bytes, 1 → Limit 4KB
S … System flag: 0 → System segment, 1 → Normal Code od. Data Seg.
Type … Zugriffsrechte
DPL … Descriptor Privilege Level: 0 → Kernel Mode, 3 → User Mode
D/B … 1 → Segment-Offsets sind 32 Bit lang
P … Segment-Present flag: 1 → Im Speicher, 0 → ausgelagert

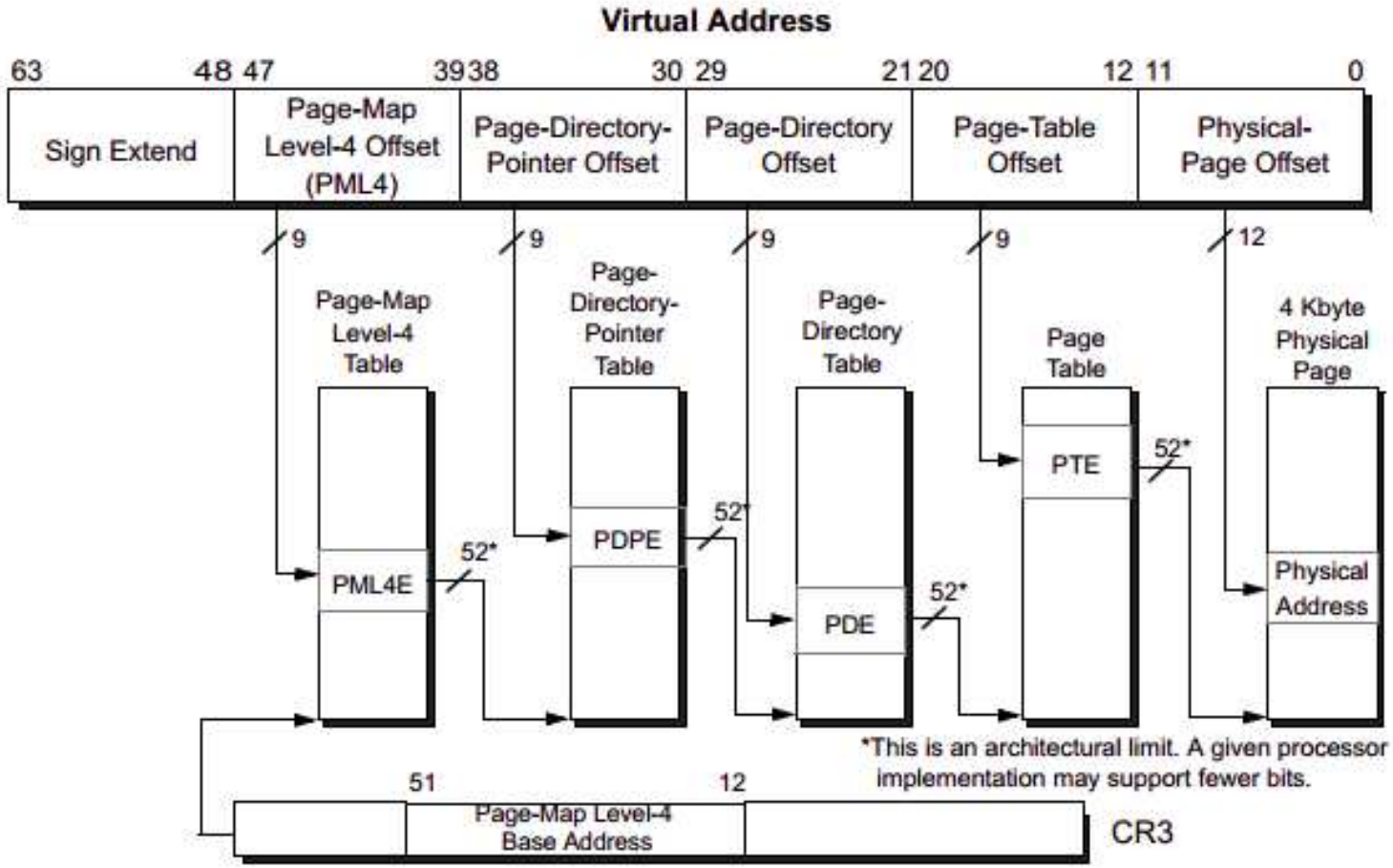
Seitenverwaltung bei ia64 (Itanium) und amd64 (intel 64)

- flaches Speichermodell (keine Segmentierung)

- Adressleitungen in einem amd64 (**Opteron**, zB Athlon 64 X2):
  - 48 Leitungen für logische Adressen
    - die 16 höherwertigen Bits der 64 Bit virt. Adr. sind alle 0
  - 40-48 Leitungen für physikalische Adressen
    - → kleinerer Adressraum, kleinere Tabellen

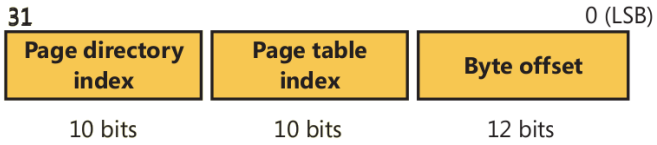
- 64 ausgeführte logische Adressleitungen im **Itanium**-Prozessor
  - Verwendet invertierte Seitentabelle

Seitenverwaltung bei amd64 (intel 64)

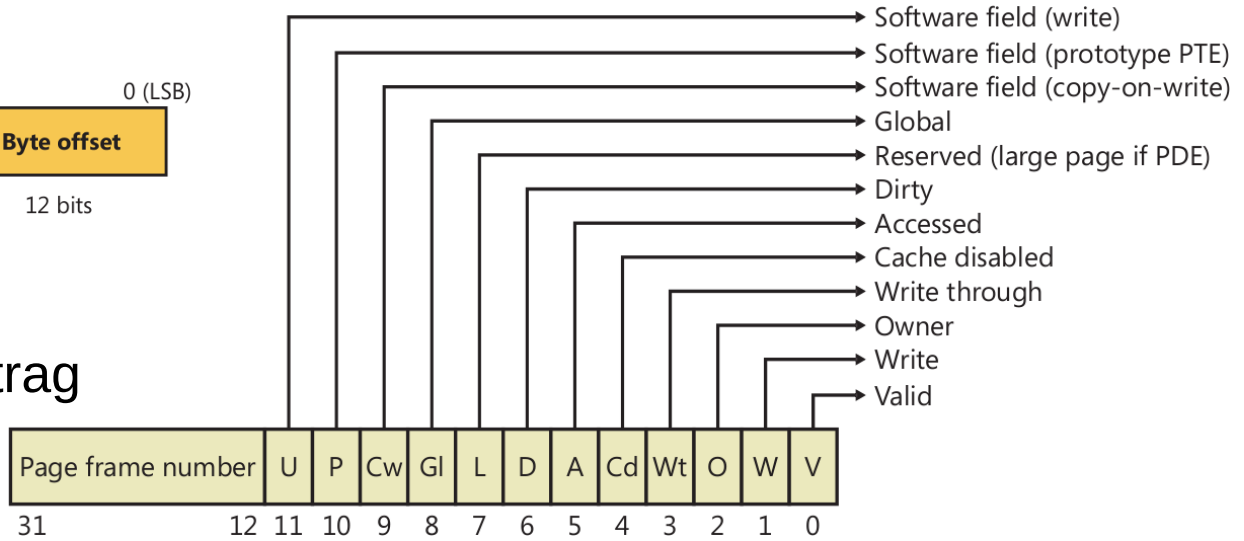


Seitenverwaltung bei x86 vs amd64

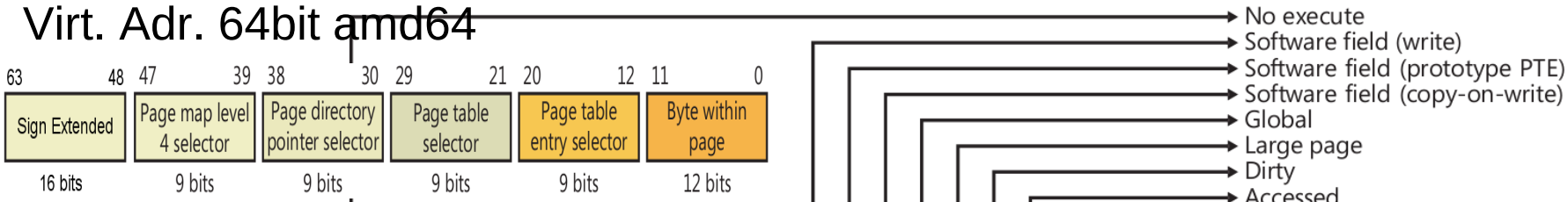
Virt. Adr. 32bit x86



Seitentabelleneintrag 32bit x86

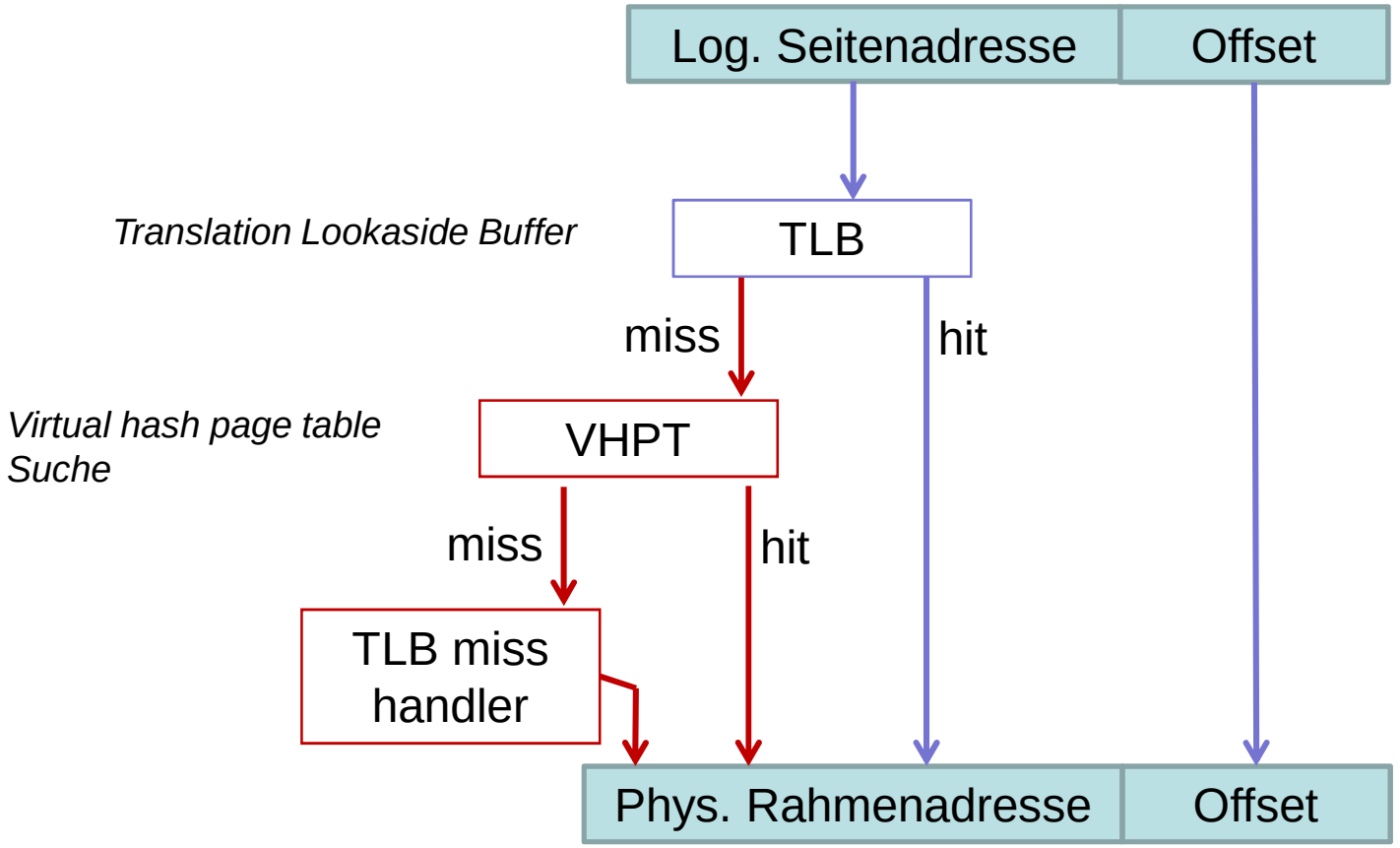


Virt. Adr. 64bit amd64



Seitentabelleneintrag 64bit amd64

Hard- und Software Teamwork bei Adressübersetzung im Itanium (IA-64)



Pageing in Linux

Demand Pageing

- Minor Page Fault: Initialisierung div. Strukturen muss gemacht werden
- Major Page Fault: Seiten Ein-/Auslagerung

```
cat /proc/*/stat | cut –d" " -f2,10,12    # ProcName, MinorFaults, Majorfaults

cat /proc/vmstat | grep "pg.*fault"

vmstat

cat /proc/meminfo
```