

5.2 Virtuelle Speicherverwaltung mit Swapping

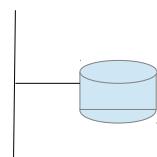
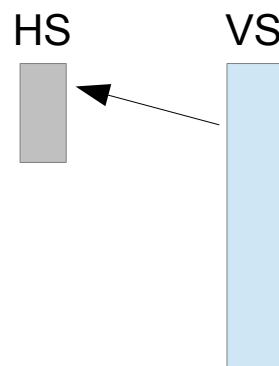
obige Verfahren vorwiegend bei

- Stapelverarbeitung oder
- in einfachen Systemen bzw. wenn
- virt. Speicherverwaltung (Adressierung) durch HW nicht unterstützt wird

heute gibt es bei PC / Workstation i. d. R. komplexere Prozessoren, welche zus. Hardware zur virtuellen Adressierung (sog. MMU) verfügen

Bei der virtuellen Speicherverwaltung unterscheidet man zwischen dem

- real zur Verfügung stehenden Hauptspeicher (HS) und
- virtuellen Speicher (VS).



5.2 Virtuelle Speicherverwaltung mit Swapping

HS ist der tatsächlich im System physikalisch vorhandene Speicherbereich (RAM)

VS befindet sich vollständig auf dem Hintergrundspeicher (i.d.R. HD-Partition)

bei Unix-Systemen i.d.R. nicht Bestandteil des Filesystems,
sondern bildet logisch ein blockorientiertes Gerät (Partition mit Gerätetreiber)

bei Win- XP,7,8 i.d.R. als File, innerhalb des Filesystems (sog. Auslagerungsdatei)

Entscheidend bei diesem Verfahren:

CPU kann nicht nur den eigenen physikalischen Adressbereich ansprechen,
sondern den sehr viel größeren virtuellen Speicher (also mehr, als Adr.-Leitungen vorh.!)

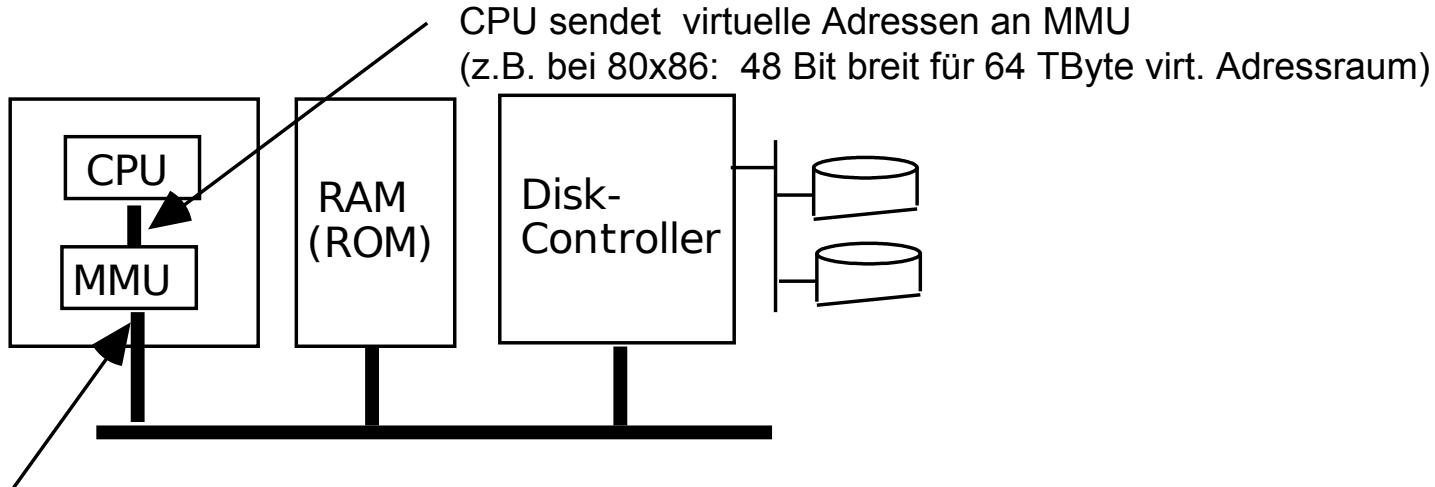
Dazu neben der eigentlichen CPU zusätzliche Speicherverwaltungseinheit
zur Umrechnung der virtuellen in die physikalischen Adressen, die sog.

Memory Management Unit (MMU)

sie übernimmt neben der Adr.-Umrechnung auch weitere Fkt. wie

- Speicherschutz
- Rechteverwaltung, ...

5.2 Virtuelle Speicherverwaltung mit Swapping



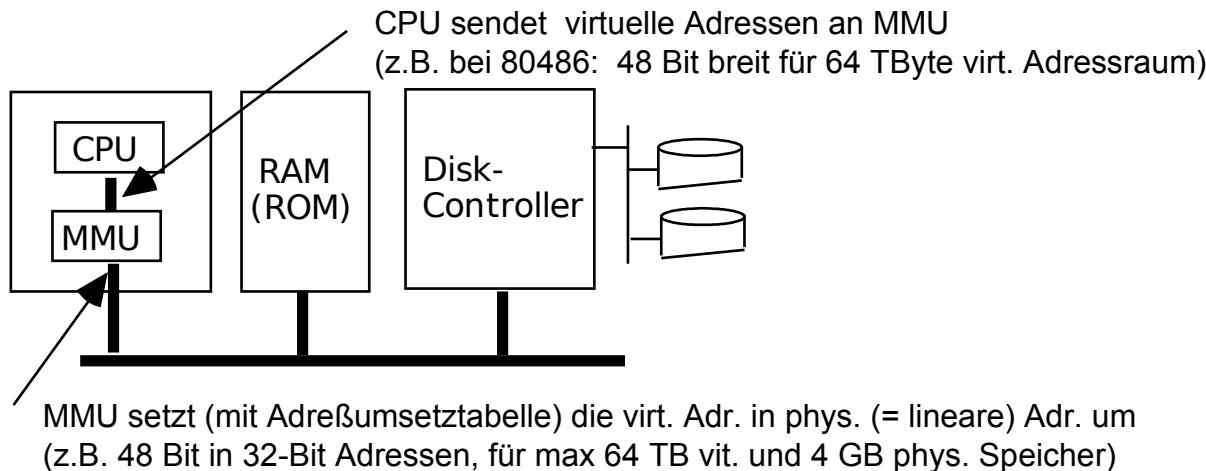
MMU setzt (mit Adreßumsetztabelle) die virt. Adr. in phys. (= lineare) Adr. um (z.B. 48 Bit in 32-Bit Adressen, für max 64 TB vit. und 4 GB phys. Speicher)

Man unterscheidet darum bei einem Prozessor (künftig stets CPU+MMU zusammen!)

- den **virtuellen** (auch logischen) und
 - den **realen** (auch physikalischen) Adressraum
-
- virt. Hintergrundspeicher auf HD kann max. so groß wie der von der CPU adressierbare virt. Adressraum sein
 - physik. Hauptsp. max. so groß wie der von der CPU real (d.h. physik.) adressierbare Speicher.

virtuelle Adressraum praktisch 2-10 mal so groß wählen, wie der tatsächliche physik. Speicher!

5.2 Virtuelle Speicherverwaltung mit Swapping



Beispiel: Intel 80386/486/...

kann virt. 2^{48} Byte (64 T-Byte) adressieren, dieser Adressbereich wird durch die MMU auf 2^{32} Byte (4 G-Byte) umgesetzt

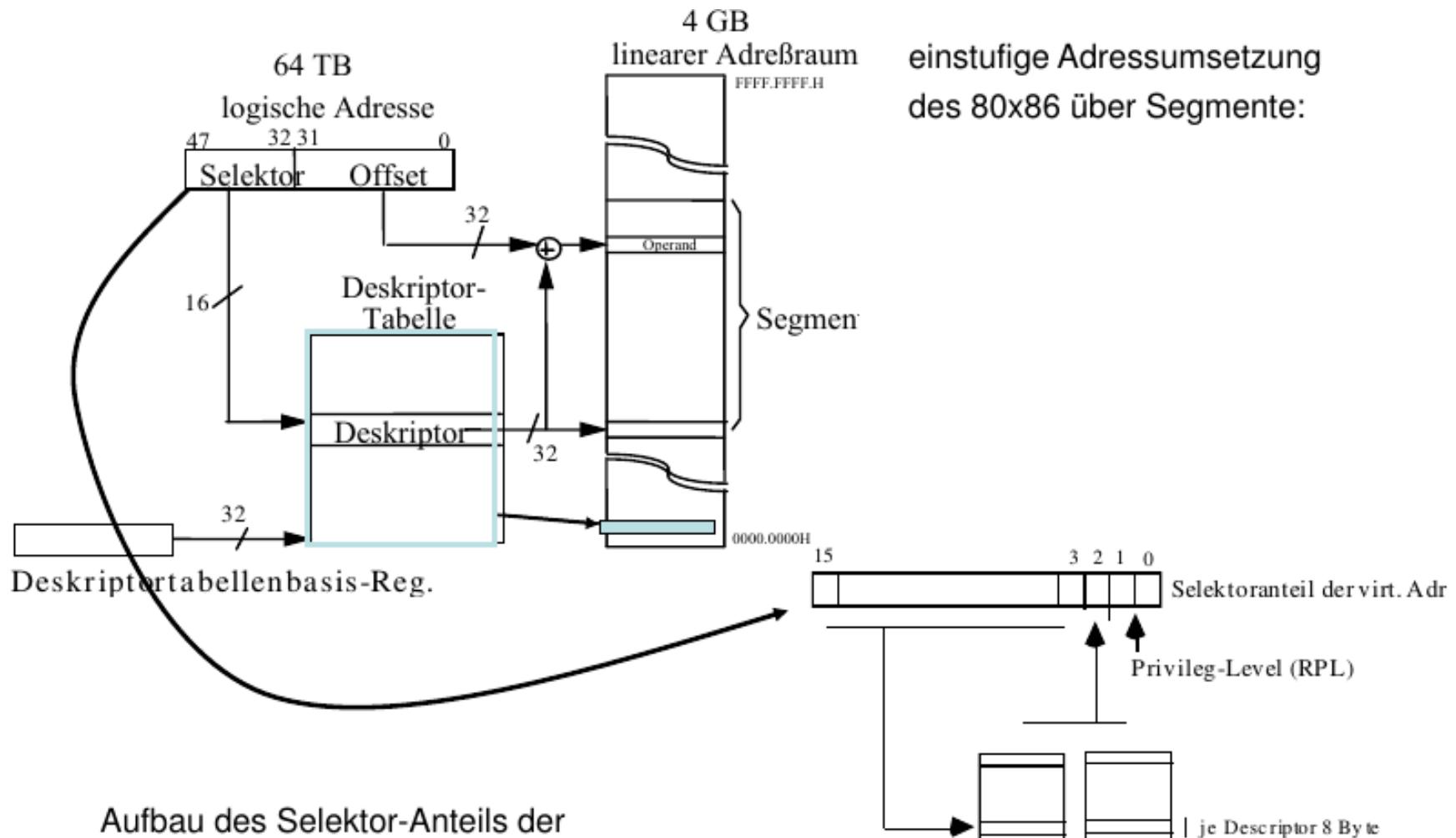
d.h.: - es können max. 4 G-Byte physikalischer Speicher angeschlossen und
- ein max. 64 T-Byte großer virt. Hintergrundspeicher (HD-Part.) verwaltet werden

z.B. real (in der Praxis):

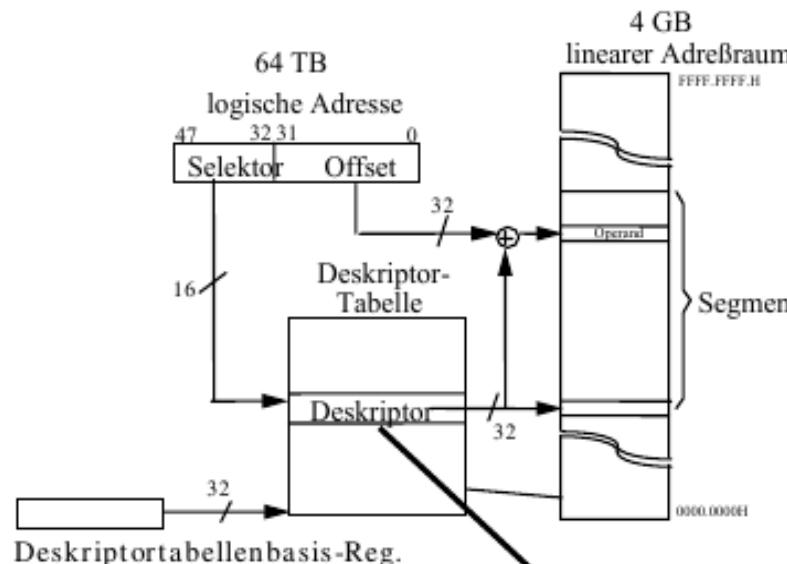
- physik. Speicher von 2 GByte (2 DDR-Speicher-Module a' 1024 MB, 60ns) und
- auf einer HD (z.B. 200GB) wird eine Partition mit 8 GB als swap-area eingerichtet
(Der Rest der HD z.B. für UNIX Boot-, System- und User-Partition)

Es können dann max. so viele Prozesse (quasi-) gleichzeitig (per time-sharing, gemultiplext) bearbeitet werden, wie in 8 GB Speicher Platz finden!

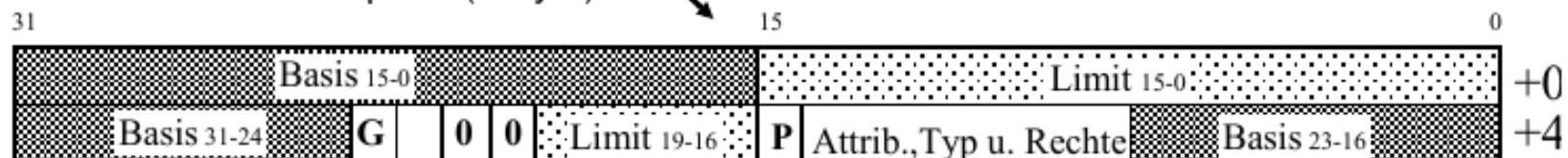
5.2 Virtuelle Speicherverwaltung mit Swapping



5.2 Virtuelle Speicherverwaltung mit Swapping



Aufbau eines Deskriptors (8 Byte):

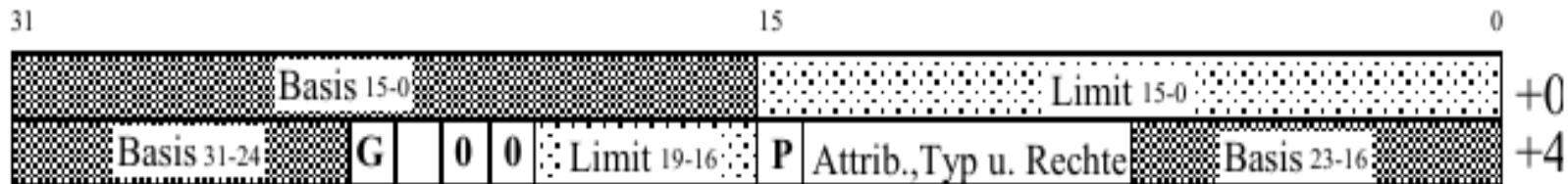


er enthält:

- Anfangsadresse des Segments, die
- Größe des Segments (limit) und
- Verwaltungsinformationen wie Typ (Code, Daten,...), Rechte, Attribute

Für den Zugriff auf eine Adresse wird zuerst ein (Segment-) Deskriptor in ein speielles Register (z.B. Datensegment-Deskriptor-Register = DS) geladen!

5.2 Virtuelle Speicherverwaltung mit Swapping



Für jedes Segment existiert in der CPU / MMU ein solches Deskriptor-Register, welches den **Selektor-Anteil** einer logischen Adresse aufnimmt:

werden beim Laden eines Selektors stets auch geladen

15	0	31	0	23	0
Selector	CS-	Basisadresse	Segment Limit	Attribute	
Selector	SS-	Basisadresse	Segment Limit	Attribute	
Selector	DS-	Basisadresse	Segment Limit	Attribute	
Selector	ES-	Basisadresse	Segment Limit	Attribute	
Selector	FS-	Basisadresse	Segment Limit	Attribute	
Selector	GS-	Basisadresse	Segment Limit	Attribute	

Beim Laden eines Deskriptor-Registers mit dem Selektor-Anteil einer Adresse werden automatisch alle im Deskriptor enthaltenen Daten in entsprechende (sonst nicht zugängige) Register der CPU (genauer der MMU) geladen.

Erst danach kann die logische in eine physikalische Adr. umgesetzt und gleichzeitig auch die Rechte und Attribute eines Segments überwacht werden.

5.2 Virtuelle Speicherverwaltung mit Swapping

Dazu werden z.B. die:

- Privilegstufe vom aktuellen Prog. Segment, welches den Intersegment call, jmp,... ausführt (= current privilege level, **CPL**) er steht im **Masch. Status Word** (= Erweiterung von Flag-Register ab 286)
- der privileg-level in der Ziel-Adresse (**RPL**= requested priv. lev.) und
- der privileg-level des Zielsegments (steht im Deskriptor des Zielsegments (**DPL**= descriptor priv. level))

vereinfacht wird wie folgt ausgewertet:

call / jmp mit: CPL < DPLCodeSeg. **nie möglich** → Exception (meist Abbruch)

CPL > DPLCodeSeg. nur via spez. Gate-Deskriptor
(d.h. call- oder trapp-Deskriptor)

d.h. kleinerer Privileglevel (0-3 bei 80x86) bedeutet höhere Rechte, und **sichererer Code !**

aus Sicherheitsgründen nur Sprünge von **unsicherem** in **sichereren** Code erlaubt !!!

5.2 Virtuelle Speicherverwaltung mit Swapping

Aufgabe der MMU:

- Berechnung der Adresse (einfache Summenbildung)
- Überprüfung ob Adr. innerhalb des Segments (Limit und Granularität: Byte/4KB)
- Überprüfung der Schreib, Lese, Ausführungsrechte bei Speicherzugriff
- Überprüfung der Privileg-Level (CPL, RPL, DPL)

In Deskriptor-Tabelle(n) wird für jedes Segment ein Deskriptor gespeichert

es gibt bei 80x86 CPUs die:

LDT= alle, für einen Prozess benötigten Segmente

GDT= alle global benötigten Segmente

und noch die IDT = Interrupt Descriptor Table

weitere Sicherheitsaspekte:

- Das Laden (Erzeugen) eines nicht CS-Deskriptors ist ein privilegierter Befehl z.B. MOV SS,... MOV DS,... (bzw. ES FS GS) ist nur im Kernel-Mode möglich!
- CS wird via call / jmp geladen mit entspr. Überprüfung der Priv.-Level (s.o.) !
- Bei Adressierungsfehler erfolgt Exception (kein Interrupt!)

alles klar ? :-)

5.2 Virtuelle Speicherverwaltung mit Swapping

Zur Bearbeitung eines Prozesses müssen:

- vorher **alle seine Segmente vollständig** in den Hauptspeicher transferiert werden
- die **Seg. + Segm.Register** (außer CS) entsprechend ihrer akt. Pos. in HS geladen werden

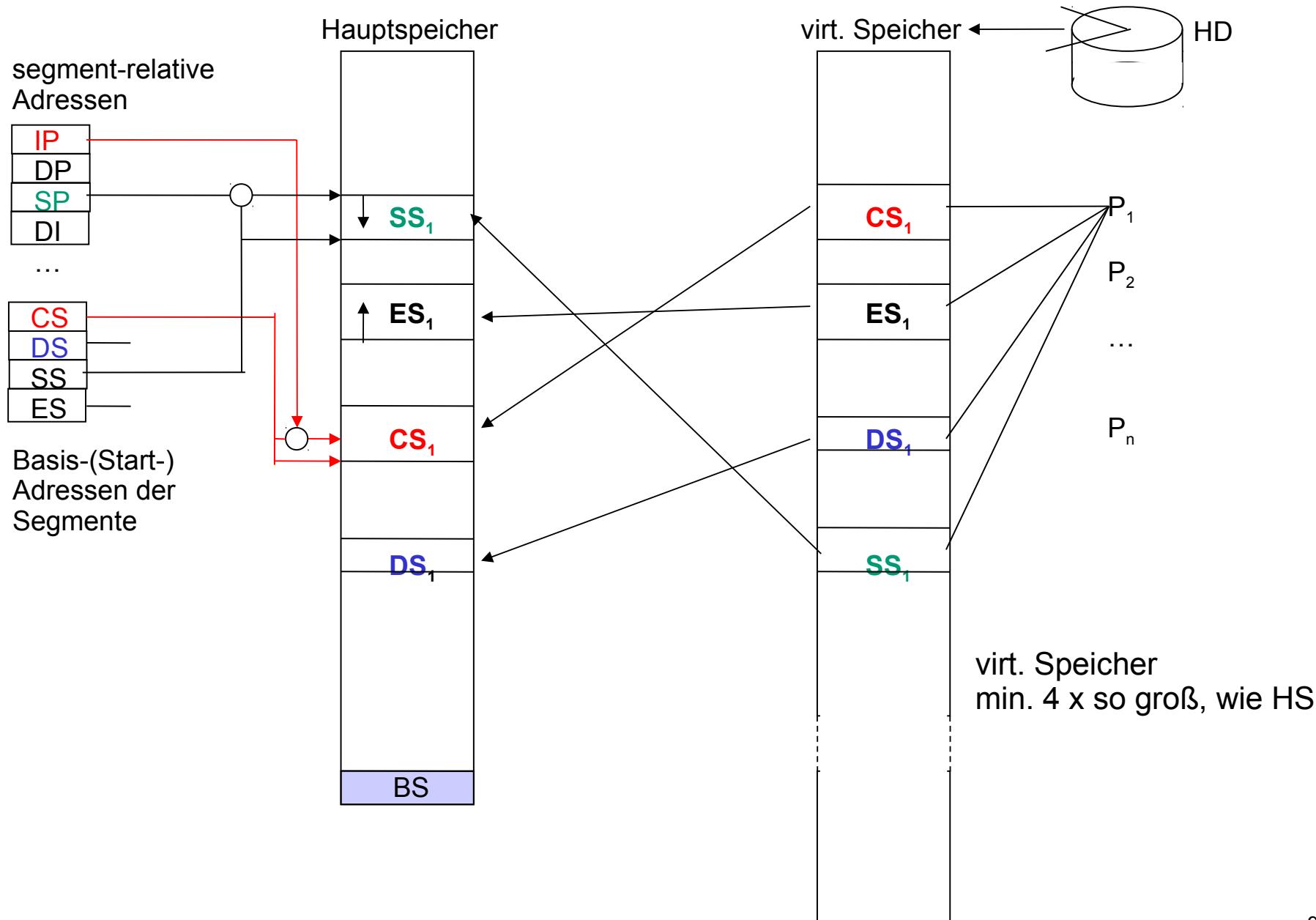
Programme (und Daten) sind dabei relativ zu den Segmentanfängen adressiert, d.h.
relativ beginnen alle Segmente bei Adr.= 0 und müssen nicht an absolute Adressen gebunden werden

Hat ein Prozess die ihm zustehende Rechenzeit verbraucht und besteht Speicherplatzmangel, so wird er auf den Hintergrundspeicher ausgelagert

Was passiert, wenn er dynamisch mehr Speicher benötigt ???

Es werden (wenn notwendig) **nur Daten- und Stack-Bereich dynamisch ausgelagert**, warum i.a. keine Code-Segment ?

5.2 Virtuelle Speicherverwaltung mit Swapping



5.2 Virtuelle Speicherverwaltung mit Swapping

Daraus resultiert:

- Es können aktuell nur so viele Prozesse quasiparallel bearbeitet werden, wie auf dem virtuellen Hintergrundspeicher (HD-Part.) Platz finden.
- Sollen mehr Prozesse aktiv sein (quasiparallel), als im HS Platz zur Verfügung steht, so müssen sie zyklisch zwischen VS und HS ausgetauscht werden.
d.h. zweistufiges Scheduling vorhanden
- Ein Prozess muss zu seiner aktuellen (Weiter-) Bearbeitung vollständig in den HS eingelagert werden
d.h. kein Prozess darf größer sein, als der phys. HS-Ausbau insgesamt
- Besteht keine Möglichkeit ein Programm vor der Auslagerung auf den Hintergrundspeicher zu schützen (d.h. permanent im HS zu halten), so kann sich die Reaktion eines Programms auf ein externes Ereignis (z.B. in der PDV durch einen Sensor) unvorhergesehen verzögern
(Das BS kann damit **kein Realzeitsystem** sein!)

5.2 Virtuelle Speicherverwaltung mit Swapping

Problem welches dadurch entsteht, dass Prozesse zur Laufzeit ihre Größe verändern können, ist mit Swapping (bis zur Größe des tatsächlichen HS-Ausbaus) gelöst!

- alloc ist System call und kann evtl. zur Vergrößerung des Datensegments führen
- evt. dazu auf größeres Segment im virt. Speicher auslagern und später erneut einlagern

Problem, wenn ein Programm nicht vollständig in den HS passt !
wird durch Swapping nicht gelöst

Ist hier die Overlay-Technik (evtl. in modifizierter Form) einsetzbar ?