Befehlsablauf in einem Von-Neumann-Rechner:
1. Befehlszähler (PC) gibt Adresse des nächsten Befehls an
2. Befehl (1.Wort) ins Befehlsregister holen
3. Befehl dekodieren, ggf. weitere Befehlsworte holen
4. Befehl bearb. (Operand holen, Werte ber., Ergeb. Speichern)
5. Befehlszähler inkrementieren und Schleife zu 1

Was macht ein BS ? Ressourcen gut verw., viel Leistung f. wenig Geld in kurzer Zeit **BS-Typen: Stapelverarbeitung:** (batch processing): Aufgaben sind genau bekannt u. es gibt keine hohe Zeitanforderung **Dialog-/Teilnehmerbetrieb:** Mehrere Anwender erledigen verschiedene Aufgaben gleichzeitig. Kurze Antwortzeit ist erwünscht. **Echtzeitbetriebssysteme:** (real time processing): Alle äußeren Ereignisse werden so schnell wie gefordert abgearbeitet.

CISC (Complex InstructionSetComputer) mächtige, komplexe Befehle **RISC (**Reduced InstructionSetComputer) einf. schnelle Bef.

Prozess: Ein Prozess ist ein sich in Ausführung befindliches Prg einschl. des aktuellen Wertes des PC, aller Reg., Var., Betr.sys.ressourcen, usw. („Kontext der Prozesse“). Jeder Prozess besitzt konzeptionell gesehen seinen eigenen Prozessor. Eine quasi gleichzeitige Ausführung mehrere Prozesse wird möglich, in dem der reale Prozessor zwischen Prozessen umgeschaltet wird.

Einprogrammbetrieb: Nur ein Prozess befindet sich im Speicher. Der Prozess erhält den gesamten Speicher. Modifikation: Betr.syst. und E/A bleiben beim Prozesswechsel im Speicher.
Mehrprogrammbetrieb: Das Betr.sys. muss von mehreren Prozessen Speicheraufforderungen erfüllen. Mehrere Prg'e (u. Daten), d.h. Prozesse befinden sich gleichzeitig im Speicher. **Grad d. Mehrprgbetr.:** Anz. der Prozesse im Speicher zu einer best. Zeit

BITMAP (Datenstrukt. z. Speicher-verw.: Eine Bitmap zeigt, wo Speicher frei/belegt ist. Damit die Bitmap nicht zu groß ist, teilt man den Speicher in Allokationseinheiten bzw. Cluster ein.

Speicherschutz: Prüft, ob die log. Adr. im zulässigen Bereich liegt (bei M32: LIMIT-Reg.). Die Anzahl der adressierbaren Worte eines Speichers bez. man als Größe des Adressraumes. Der phys. Adressraum kann größer/gleich/kleiner sein, als der log. Adressraum.

Buddy-System: Verwalte den Speicher in Einheiten der Größe $1,2,3,\ldots 2^K,\ldots,2^N$ für jede Größe eine Liste der freien Segmente. Ges.größe des Speichers beträgt 2^N. Am Anfang gibt es nur ein freies Segm., das 2^N groß ist. Segm. der Größe 2^K müssen auf einer durch 2^K teilbaren Adr. Beginnen! Das Verschmelzen von Löchern wird einfacher, wenn nur passend liegende Segm. (Buddys) verschmelzt werden.

SWAP-Syst. bei segmentierten Speicher:
50%-Regel: Sind im Mittel n Segm. Im Speicher, so sind im Mittel $^N/_2$ Löcher im Speicher.
Unben.Speicher-Regel: s = Ø Gr. eines Segm.; n = Anz. D. Prozesse (im Mittel);
k·s = Ø Lochgröße (k = Ø rel. Lochgr.) m = Ges.sp.
Aufsummierte Lochgr. -> $^n/_2 \cdot k \cdot s = m - n \cdot s$ -> $m = n \cdot s \cdot (1+{}^k/_2)$; $f = ({}^n/_2 \cdot k \cdot s)/m = k/(2+k)$; f=Bruchteil des ungen. Speichers

Virtueller Speicher: Idee: Ges.größe darf Größe des phys. Speichers überschreiten. Leichte Verw. Durch d. Betr.sys. ohne Benutzereingriffe; **Lösungsans.:** Der logische Adr.raum wird mit einer MMU realisiert (mit Hilfe einer HD). Gleichzeitig: Mehrprgbetrieb: Jedes Prg/Prozess erhält einen eigenen log. Virt. Adressraum. **Realisierung:** *PAGING:* Der Speicher wird in Pages in gleich großen Teilen unterteilt (Größe z.B. 1k Worte). Größe einer Seite = Größe eines Seitenrahmens
Die Zuordnung zw. Seiten u. Seitenrahmen wird mit der Seitentabelle realisiert. Die Umrechnung der Seitennr. in Rahmennr. für alle Seiten des virt. Adr.raumes, z.B. 32bit log. Adr.; 1k Worte als Seitengröße -> Anz. Seiten: $2^{32}/1024 \approx 4M$, d.h. die Seitentab. Ist sehr groß! Jeder Prozess hat seine eigene Seitentabelle.

TranslationLookasideBuffer(TLB) = Assoziativspeicher Verwendung um häufig benutze Einträge der Seitentabelle zu speichern. Finde für wichtige Seiten schnell den Seiten-rahmen. Klappt dies nicht, benutze die Seitentabelle. Kombin. aus mehrstufigen Paging. T_s = Zugr. ü. Seitentab., Treffer im TLB = T_{TLB}; Mittl. Adressumsetzungszeit: $t = p \cdot T_{TLB} + (1-p) \cdot T_s$

Beschleunigung mit TLB: Bei Kontextwechsel muss ges. TLB auch geweschselt werden. Vorteile von Paging: Gem. Speicher f. mehrer Prozesse ist gut realisierbar UND gute Schutz-mechan. Implementierbar (auf Seiten(tabellen)ebene) durch zus. Einträge in der Seitentab.

Seitenersetzungsalgorithmen: **NRU(NotRecentlyUsed):** Feststellen, ob eine Seite häufig oder selten benutzt wird. Aufwand: Wenige Bit/Seitenrahmen. R-Bit = RecentlyUsedBit: Wird bei jedem Zugriff auf den Seitenrahmen aus 1 gesetzt. M-Bit = ModifiedBit: Wird bei jedem Schreibzugriff gesetzt. Beide Bits werden in Hardware (MMU) real. Das R-Bit wird in festen Zeitabständen vom Betr.syst. zurückges. Bei Seitenfehler durchsuche die Seitenrahmentab. U. ordne die Rahmen in 4 Klassen. Dann entferne eine Seite aus der nicht leeren Klasse mit der niedr. Kl.nr. Innerhalb der selben Kl. erfolgt Auswahl zufällig →Leicht zu implem., relativ effizient (nicht opt.) *KLASSEN:* 0(R=M=0), 1(R=0,M=1),2(R=1,M=0),3(R=M=1) **FIFO(FirstInFirstOut):** Führe verkettete Listen ein, um die Seite zu ermitteln, die am ältesten/längsten im Speicher ist. Diese Seiten müssen entfernt werden. Problem: Alte Produkte könnten Longseller sein !! **SecondChance:** Verwende R-Bit, um häufig benutzte referenzierte/benutzte Seiten zu finden. Ist in der FIFO die älteste Seite R=1, dann lagere Sie *nicht* aus, sondern stelle Sie wieder „hinten“ in die FIFO mit R=0 an. → Leichte Impl. U. effektiv ! Impl. Mit Ringpuffer wird Uhr-Algorithmus genannt. **LRU(LeastRecentlyUsed):** Entferne die Seite, die am längsten nicht benutzt wurde. Unterst. Durch MMU -> Baue Instrukt.zähler ein, der bei jeder CPU-Instr. den Wert erhöht. In d. Seitentabelle inplem. F. jeden Rahmen einen Zugriffszähler. Bei Zugriff wird der Wert des Instr.zählers in den Zugr.zähler kopiert. Bei Seitenfehler, entferne Seite mit niedrigsten Zugr.zählerwert. → Algor. oft zu zeitaufwendig ! **LFU(LeastFrequentlyUsed):** Stelle mit einem Zähler fest, welche Seiten häufig benutzt werden (1Zähler/Seite). Entferne die Seite, die am wenigsten benutzt wurde.

Workingset: Satz v. Seiten, mit denen ein Prg längere Zeit bei wenig Seitenfehlern auskommt (Lokalität der Referenz)
Demand Paging: Seiten werden nur nach Anforderung eingelagert **Pre-Paging:** Bringe am Anfang die wichtigsten Seiten eines Prozesses in den Speicher
THREATS: Nur Teile vom Kontext werden ausgetauscht, Arbeit viel schneller. Ein Threat verkörpert einen „Flow of control“ (Faden).

Aufteilung der Seitenrahmen auf Prozesse:
equal allocation → Jeder Prozess erhält gleich viele Rahmen, m Rahmen insgesamt, n Prozesse -> Jeder bekommt $^m/_n$ Rahmen
proportionale Aufteilung → Jeder Prozess erhält eine Rahmenzahl, die proport. zu seiner Anforderung ist. Alle Rahmen verteilt.
Beide Verfahren ändern sich nach Grad des Mehrprogrammbetriebs
Global Replacement: Der Allocator oder Payer darf, wenn Prozess B einen Rahmen benötigt, einen Rahmen verwenden, den A belegt hat. Prozess A ist nicht alleine für seine Page-fault-Rate verantwortlich. Prozesse mit hoher Prior. u. hoher Page-fault-Rate bereichern sich !! ABER, GlobRepl kann wenig benutzte Seiten anderen Prozessen zur Verfügung stellen. Daraus resultiert eine höhere Systemleistung; **Local Replacement:** Wenn Prozess A einen neuen Rahmen benötigt, muss A auch einen Rahmen abgeben.

Thrashing-Problem: Wenn ein Prozess zu wenig Seitenrahmen hat muss er gestoppt und ausgelagert werden. Wenn ein Prozess wenig Seitenrahmen allokiert hat, ist seine Page-fault-Rate hoch ! ***Thrashing:*** Ein Prozess verbringt mehr Zeit mit Paging, als mit der eigentl. Ausführung. Ursache: z.B. Betr.sys. merkt, dass die CPU-Auslastung sinkt; Erhöhe den Grad d. Mehrprogbetr; mehr Prozesse; System erzeugt mehr Page-faults; Ein Prozess, der gerade auf das Einlagern einer Seite wartet, ist nicht rechenbereit; CPU-Auslastung sinkt; …
Programme haben oft einen Satz von Seiten (Arbeitsbereich, Working-Set) mit dem sie längere Zeit bei wenig Seitenfehlern auskommen (Lokalität der Referenzen). Um ständig genügend freie Seiten zu haben wird ein **Page Deamon** eingesetzt. Dieser „stiehlt“ ständig einige Seiten von Prozessen.

ProzessControlBlock (PCB): Verwaltung von Prozessen durch das Betrsys! ***Inhalt:*** Prozess-Zust.; PC; Reg u. sonst. Kontext; Schedulerinfos; Speicher-verw. u. Abrechn.infos; E/A-Info; Proz.kommunik.;Zeiger auf nächsten PCB

Scheduling: Auswahl des nächsten Prozesses der aktiviert/rechnend wird, aus Menge der rechenbereiten Prozessen. ***Preemptiv:*** Prozesse, die „running“ sind, dürfen vom Betr.syst. in den „ready“ Zustand versetzt werden. Generell: Ein neuer Prozess wird „running“ ohne dass ein alter Prozess beendet ist. (Wird für Mehrbenutzerbetr. bevorzugt. ***Run to completion:*** Gegensatz zum Preemtiven Scheduling. Der Prozess läuft bis zu seinem Ende. (Ein lang rechn. Prozess kann alles blockieren) ***Kooperativ:*** Prozesse sagen, dass sie unterbrochen werden dürfen (Kompromiss).

Dispatcher: Prozesswechsel: Kontext retten, neuen Kontext herstellen, beim neuen Befehlszählerstand weiter machen. ***Dispatch Latency:*** Verlorene Zeit beim Prozesswechsel (ms).
Uhralgor.: Zeiger zeigt auf Seite, die als nächstes zur Auslagerung in Frage kommt. Seite mit gesetzten R-Bit werden nicht eingelagert.

NTFS: Volume = Diskpartition; Verw. von Clustern: Wenn Ges.größe <512MB, dann 1Cluster = 1Sektor; wenn < 1GB → 1C=1kB; wenn < 2GB → 1C=2kb; wenn ≥2GB → 4kB; Clustergröße ist < als bei FAT16 → weniger int. Fragm. !! FILE ist Obj. U. kein Bytestream

NTFS-Sicherheit: Alle Dateizugriffe werden als Transaktionen ausgeführt. Es existieren LOG-Records mit redo u. undo.Info und commit-records. Alle fünf Sekunden werden so genannte checkpoints gesichert.

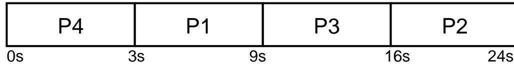
Scheduling-Algorithmen: Anforderungen: Fairness (Jeder Prozess erhält einen gerechten Anteil an Prozessorzeit), Effizienz (Der Prozess soll gut ausge-lastet sein), Kurze Antwortzeit für interaktive Benutzer, Kurze Verweilzeit bei Stapelverarbeitung, Durchsatz soll hoch sein, d.h. viele Aufträge/Zeit.
FirstComeFirstServed-Algorithmus (FCFS): Verwaltung aller Prozesse die „ready“ sind, in einer FIFO. Der zuerst angekommene Prozess wird zuerst verarbeitet/ausgeführt, z.B.: P1 (24ms) kommt zuerst, dann P2 (3ms) u. zuletzt Pe (3ms). Belegung d. Prozessors über die Zeit wird GANTT-Chart genannt.

P1	P2	P3
0	24	27 – 30

Wartezeit eines Prozesses: Zeit vom Eintreffen in der Warteschlange bis zum Scheduling
Mittl. Wartezeit: Mittelwert der Wartezeit aller Prozesse (hier: 17)
TurnAround-Zeit: Zeit von Eintreffen des Prozesses bis zur vollständigen Abarbeitung

ShortestJobFirst-Algorithmus (SJF): z.B.

	Rechenz	Wartezeit
P1	6s	3s
P2	8s	16s
P3	7s	9s
P4	3s	0s



Mittl. Wartezeit: (3+16+9+0)/4 = 7s → Bei FCFS wäre die mittl. Wartezeit 10,25s !! ALSO: SJF minimiert die mittl. Wartezeit bei geg. Prozessen. PROBLEM: Man kennt die Rechenzeit (bis zum nächsten Blocked Zustand) nicht im Voraus ! Man muss versuchen, die Länge des nächsten Rechenintervalls vorauszusagen. Vorhersage: τ_n
Wahre Länge: t_n $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\, \tau_n$ Wähle z.B. α=0,5

ShortestRemainingTimeFirst-Algorithmus (SRTF): (=Preemptives SJF) Wenn ein Job mit kürzerer Restrechenzeit eintrifft wird er vorgezogen und der aktuelle Prozess (mit längerer Restrechenzeit) unterbrochen. VORSICHT: Wenn viele kurze Prozesse schnell genug ankommen, dann „verhungern" die großen Prozesse. FAZIT: Reines SRTF ist verboten !!

RoundRobin Scheduling/Algorithmus: Zeitscheiben fester Größe (Quanten) werden nacheinander an die laufbereiten Prozesse vergeben (immer preemptiv). Verwaltung mit FIFO.

Prioritäts-Scheduling: Rangordnung von Prozessen. Hohe Priorität = wichtige Aufg. (Interruptbehandlung). Jedem Prozess wird eine Prio. zugewiesen. Es wird bei jeder Schedulingentscheidung der Prozess mit der höchsten Prio. ausgewählt. Statische Zuweisung der Priorität → Gefahr: „Verhungern" !! Besser: Dynam. Zuweisung der Prio., d.h. Prio. ändert sich über die Laufzeit. ***Verwaltung:*** Einteilung in (wenige) Prio.klassen. Eine Warteschlange (als FIFO) pro Klasse. Innerhalb einer Klasse: RoundRobinScheduling → Es besteht immer noch die Gefahr des „Verhungerns"! Abhilfe: Erhöhe für Prozesse, die lange nicht Geschedult wurden die Prio.klasse. Eventuell auch Prioritäts-Boost bei externen Ereignissen → MULTI-LEVEL-SCHEDULING.

Wechselseitiger Ausschluss (Mutual Exclusion = MUTEX): Kritische Bereiche dürfen nur von einem Prozess ausgeführt werden. Der Teil eines Programms, der auf gemeinsam genutzten Speicher zugreift, ist ein Beispiel für einen kritischen Bereich. Es dürfen sich NIE zwei Prozesse gleichzeitig im kritischen Bereich befinden ! ***Lösung***: Wechselseitiger Ausschluss; Kein Prozess, der sich nicht selbst im kritischen Bereich befindet darf andere Prozesse blockieren; Kein Prozess muss ∞ warten, um in den kritischen Bereich zu kommen; das Verfahren funkt. für beliebig viele Prozesse !! ***Andere Lösung:*** Variable TURN regelt, wer den kritischen Bereich betreten darf. Bei Verlassen des kritischen Bereiches wird jeweils dem anderen Prozess die Eintrittserlaubnis erteilt. Problem: Verletzung der 2.Bedingung → Evtl. muss ein Prozess unendlich lange warten, ohne dass ein anderer Prozess sich im kritischen Bereich befindet. Bemerkung: Die Ausführung einer Schleife bis zum Eintreten eines Ereignisses wird AKTIVES WARTEN (**Spinlock-Waiting, Busy-Waiting**) genannt. Spinlock's sind auf Mehrprozess-Systemen sinnvoll u. wenn Kontextwechsel länger dauern, als die mittlere Wartezeit. *Einfache Lösung mit Hardware-Unterstützung:* Man muss gleichzeitig (unteilbar) den Wert einer Var. testen u. setzen können → **TEST and SET Lock Instruktion (TSL)**

HD=32768 Bytes/Spur, Rotzeit = 16,67ms (:= 3600rpm), ØPos.zeit = 30ns, Zusammenhängender Block=200kB → Datei ben. 200/32=6,25 Spuren Im günstigsten Fall verteilen sie sich auf 7 Spuren (ungünstigsten Fall 8).. Ges.Lesezeit (bei einer Oberfläche): **Günst.Fall (7Spuren)**: Am Anfang kein Spurwechsel! Dann 6 Spurwechsel = 6·30ms=180ms; Kopf steht nach Spurwechsel immer am Anfang des richtigen Sektors -> Rot.latenz = 0ms, reine Lesezeit: 6,25·16,67ms=104,1875ms -> Σ≈285ms. **Ungünstig.Fall(8Spuren):** Am Anfang ein Spurwechsel, dann 7 Spurwechsel -> 8·30ms=210ms. Kopf steht nie nach SpWechsel am Anfang des richtigen Sektors -> 8 · Rotat.latenz = 8·16,67ms; Σ=210ms+133,36ms+104,1875=447,5475ms; **Im Mittel:** 7SpWechsel,8·½Rot.latenz u. Lesezeit → Σ=381,555ms

TSL-Instruktion: Verwendung: Variable FLAG (=1 → Sperre gesetzt), Initialisiert: FLAG=0; Spinlock wait:

```
while TSL (&FLAG)==1 do {}
….. kritischer Bereich……
FLAG=0;
```

Wichtiges Prinzip: Unteilbare **atomare Operation**. Realisierung: a) Hardware b) Sperren aller Interrupts für die Dauer der atom. Operation. Ohne TSL/Int.Sperr.: Peterson-Verfahren

Abhilfe: **Blockierendes Warten:** Ein Prozess kann sich „schlafen legen", d.h. er ruft sleep() auf. Ein Prozess kann an einen anderen Prozess einen Aufweckruf wakeup(process_id/Empfänger) schicken. Wakeup führt beim Empfänger zum Übergang Blocked in ready. Benutzung z.B. eines Erzeuger-Verbraucher-Systems → Ist nicht einfach zu kontrollieren !! Lösung: **SEMAPHOREN !**

Semaphoren: Sind Int-Var., auf die nur in festgelegter Weise zugegriffen werden darf. Idee: Der Wert S ((S ist eine Semaphorvar.) gibt an, wie viele Objekte noch verfügbar sind. Tanenbaum: down: if s≤0 sleep; s=s-1; Dies wird atomar implementiert. up: s=s+1; Wenn jemand schläft: Ein Prozess aus d. Schlafenden wecken.

Oberfläche: unterteilt in konzentr. Spuren; **Spur:** unterteilt in Sektoren; **Sektor:** phys. Einheit, die am Stk gelesen/geschrieben wird; **Zylinder:** Übereinander liegende Spuren aller Oberflächen m. gl. Radius; **Cluster:** Allokat. v. Einheiten. Ein Cluster sind mehrer Sektoren, übliche Größe 2er Potenz von Sektoren (z.B. 512 B)

Lesen von Daten: *a) Positionieren des Kopfes:* Angabe der Pos.zeit: HD: für eine beliebige Spur im Mittel; *b) Warten auf den richtigen Sektor:* 1) Im günstigsten Fall = 0 Umdrehungen; 2) im Mittel: ½ Umdrehung; 3) Im ungünstigsten Fall = 1 Umdrehung; *c) Lesen oder Schreiben der Daten:* Eigentliche Übertragung der Daten, ggf. können mehrere Köpfe parallel lesen. Transferrate: Bit/sek für einen Kopf oder alle gemeinsam.

Betr.m.klassen: R_j **Prozesse:** P_j
$R_j \rightarrow P_j$ = Zuteilungskante
$P_k \rightarrow R_L$ = Anforderungskante

$$R = \begin{matrix} P1 \\ P2 \\ P3 \end{matrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$
(Spalten: R1 R2 R3)

Anford.matrix: R **Betr.m.vektor:** E;
Beleg.matrix: C; **Restvektor:** A (unbenutzten Betriebsmittel)

Sind Anforderungen erfüllbar ? Eine Zeile der R muss ≤ A sein. Wenn ja Zeile streichen u. zugehörige Zeile aus C zu A addieren → Neuer Restvektor !! Wenn zum Schluss A = E → Kein Deadlock vorhanden !!

FAT12: Verw. max. 2^{12} Einträge/Cluster (4096), jeder Eintrag 12 Bit
rpm ↔ ms: 1/3600rpm=277,7µ * 60 = 16,6ms z.B.: 10000rpm=166,6rps=6ms

Phys. Kap einer HD: 16Köpfe, 63Sektoren, 512Bytes/Sektor u. 4092 Zylinder: 512·63=32256 Bytes/Spur→ 32256·4092 Bytes/Oberfl.=131991552 Byte≈132 MByte→132MB · 16 Oberfl.≈ 2,1GB

DEADLOCK: Jeder Prozess wartet auf ein Betr. mittel, die von einem anderen Prozess belegt werden. ***Notwendige Bedingung für das Auftreten von Deadlocks***: 1.) *Wechselseitiger Ausschluss*: Jedes Betr.m. wird entweder von genau einem Prozess belegt oder es ist verfügbar; 2.) *Belegungs- u. Wartebedingung (Wait and Hold):* Ein Prozess, der bereits Betr.m. belegt, darf weiter Betr.m. anfordern. 3) *Ununterbrechbarkeitsbedingung (No preemption):* Belegte Betr.m. können nicht entzogen werden. Diese müssen zuerst frei gegeben werden. 4) *Zyklische Wartebedingung (Circular Wait):* Eine zyklische Kette $P_0, P_1, \ldots, P_n$ muss derartig existieren, dass P_0 auf Betr.m. wartet, welches P_1 besitzt,….

Int. Fragm.: Verlust innerhalb von Clustern durch freien ungen. Speicherplatz, z.B. Cluster > Datei;

```
Line  RAM                CODE              TEXT
001   ___00H :                             ORG  0
002   ___00H :                         L1  EQU  7
003   ___00H : 21002001H __038H            MOV  R1,56
004   ___02H : 21002002H __02H         X   MOV  R2,X
005   ___04H : 21004203H __000H            MOV  R3,0(R2)
006   ___06H : 01000000H                   NOP
007   ___07H : F2000000H                   HALT
008   ___08H :                         Y   DS   8
009   __010H : _____003H ___07H        Z   DW   3, L1
010   __012H :                             END
```

Symbol	Wert
L1	7H
X	2H
Y	8H
Z	10H

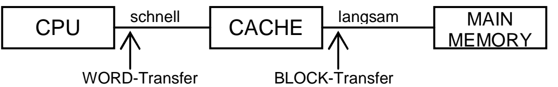
R1 = 38H
R2 = 2
R3 = 21002002H
R4 = 0

Ext. Fragm.: Zerstückelung von Dateien in kl. Teile; keine großen Blöcke sind frei ! Im Mittel geht ½ Cluster/Datei verloren.

Paging: 256Bytes/Seite → 8bit Offset in der Seite, 32bit virt. Adr.raum, 20bit phys. Adr. : 32-8 = 24bit Seitennr. → 2^{24} Einträge = 16777216 Einträge in Seitentabelle; $2^{20-8} = 2^{12}$ = 4096 Seitenrahmen

Unified-Cache: Cache für Befehle und Daten
Split-Cache: Einen Cache für Befehle u. einen Cache für Daten

Cache-Speicher: Ziel: Höhere Datentransferrate z. Prozessor bei langsamen Speicherbus. Prinzipiell:



Wegen des inneren Aufbaus des Main-Memory ist ein Blocktransfer sehr effektiv. MainMemory: 2^N adressierbare Worte
Block: k Worte $M=2^N/k$ Blöcke

```
int TSL(int *TSLvar) {
    DISABLE(); //Interr. Aus
    if(*TSLvar==0) { *TSLvar=1;
        ENABLE(); //Interr. Ein
        return0;
    }else{
        ENABLE(); return1; }}
```

```
// Prozess A:
while(TSL(&MUTEX)==1)
 {} //Warte
Atemp=U; U=V;
V=Atemp; MUTEX=0;
//kritischen Bereich verl.
```
→ **Prozess B** genauso !

Teilprobl. beim Cache: ***Write-Policy-Problem:*** MainMemory wird nicht nur von der CPU, sondern auch von der I/O benutzt (DMA); ***Write-Through-Policy:*** Schreibe immer in Cache u. in den Memory; ***Update-Policy:*** Schreibe immer im Cache. Result.: Man muss dann auch immer im Cache lesen! → Besser: Multilevel-Cache

```
Int Konto[100];
Semaphor Mutex[100];
    Down(&Mutex[I]);
    Down(&Mutex[J]);
        Kontobewegungen
    Up(&Mutex[I]);
    Up (&Mutex[J]);
```

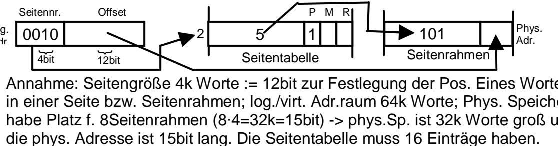
Deadlocks können auftreten! z.B.: A bewegt 1→2, Scheduling, B bewegt 2→1: Zeitl Reihenf. d. Ereign:
A Down (1) → erfolgreich
B Down (2) → erfolgreich
B Down (1) → schläft
A Down (2) → schläft } Deadlock

Virt.Adr.raum = 64kB → log./virt.Adr. hat 16Bit; Seite = 1024Byte → 10 Bit → Seitennr. hat 6 Bit (16-10=6)

HD m. 16 Köpfen, 63Sektoren, 512 Byte/Sektor, 4092 Zylinder → 512·63= 32256 Byte/Spur; 32256·4092=131991552 Bytes ≈ 131 MB; 131·16=2,1GB Hat eine HD pro Oberfläche 800Spuren hat sie auch 800 Zylinder !!

Cluster=4096Bytes; HD=20GByte=20·2^{30} Bytes; 3 FAT's, FAT 32-bit-Einträge → (20·2^{30})/4096=5242880 Cluster; (ln5242880/ln2)≈22,3Bit → 23Bit zur Verkettung → 9Bit frei für Sonderaufgaben

HD m. 2^{21} Sektoren u. ca.34GB, FAT32, 1Cluster=1Sektor → (34 ·10^9) / 2^{21}=16212 Anz. Sektoren → 2^{14} Bytes/Sektor → 16384 · 2^{21}=34,3597…·10^9 Bytes



Annahme: Seitengröße 4k Worte := 12bit zur Festlegung der Pos. Eines Wortes in einer Seite bzw. Seitenrahmen; log./virt. Adr.raum 64k Worte; Phys. Speicher habe Platz f. 8Seitenrahmen (8·4=32k=15bit) -> phys.Sp. ist 32k Worte groß u. die phys. Adresse ist 15bit lang. Die Seitentabelle muss 16 Einträge haben.

Virt.Adr.: 1024 Seiten·4096 Byte/Seite=4MB; Phys.Speicher: 16 Seitenrahmen·4096Byte/Seite=64kB; **4096 Byte/Seite = 12 Bit Offset !!**

	virt.Adresse	phys.Adr.	
W	23 : 647 H	0 : 647 H	→ Seitenrahmen 0 -> Seite 23H
R	10 : 263 H	1 : 263 H	→ Seitenrahmen 1 -> Seite 10H

Wie viele Sektoren belegt eine FAT ? → FAT32: (4Bytes/Eintrag · 2^{21} Einträge/FAT) / 16384 Bytes/Sektor = 512 Sektoren

Wie viele Cluster bleiben für Daten frei, wenn 16 Bootsekt, 3FATs u. ein Hauptverzeichnis angelegt werden ? → 2^{21} – 3· 512 – 16 – 98 = 2095502 Sektoren