

```

Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[30103]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6216]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6941]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sshd[9077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10102]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63755
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17878]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:39 amd64 sshd[31269]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[4674]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[5489]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[5489]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 02:00:01 amd64 /usr/sbin/cron[5489]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 01:00:01 amd64 /usr/sbin/cron[12589]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted publickey for esser from ::ffff:192.168.1.5 port 59771 ssh2
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[12521]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[23197]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: end_seq_oss: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: end_seq_oss: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29299]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[6621]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11566]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778

```

Speicherverwaltung (2)

Zuteilung freien Speichers

Angeforderter Speicherbereich einer bestimmten Größe kann wie folgt zugeteilt werden:

- **First-Fit-Methode**
 - Der erste genügend große freie Speicherbereich wird zugeordnet.
- **Best-Fit-Methode**
 - Der kleinste ausreichende freie Speicherbereich wird zugeordnet.
 - Aufwendiger, da die gesamte Liste bzw. Bitmap durchsucht werden muss.
 - Starke Fragmentierung des freien Speichers in viele kleine Bereiche.

Zuteilung freien Speichers

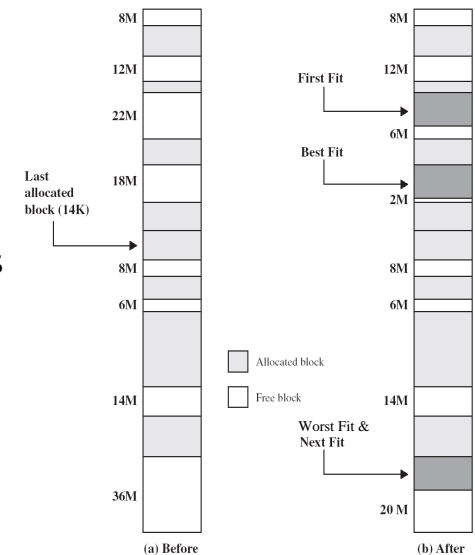
- **Worst-Fit-Methode**
 - Der größte freie Speicherbereich wird zugeteilt.
 - Es bleiben verhältnismäßig große freie Bereiche übrig.
- **Quick-Fit-Methode**
 - Unterhalten von mehreren Listen freier Speicherbereiche für bestimmte Standardgrößen (z. B. 4 KB, 8 KB, 12 KB etc.) führt zu einer schnellen Zuteilung.
 - Bei Freigabe eines Bereiches muss dieser mit evtl. benachbarten freien Bereichen zusammengelegt werden. Dies bedeutet bei mehreren Listen einen erhöhten Aufwand.
 - Spezielle Variante: **Buddy System**

Zuteilung freien Speichers: Beispiel

Beispiel für eine Speicherbelegung

- vor und
- nach

der Allokierung eines 16-MByte-Blocks



Speicherverwaltung: Buddy System

- Separate Listen freier Bereiche der Größen 1, 2, 4, 8, 16 etc. KByte bis zur Speichergröße.
- Bei Freigabe eines Speicherbereichs muss nur eine der Listen durchsucht werden, um festzustellen, ob der Bereich mit einem anderen freien Bereich zusammengefasst werden kann.
- Blockgröße immer Zweierpotenz ist nicht sehr speichereffizient.

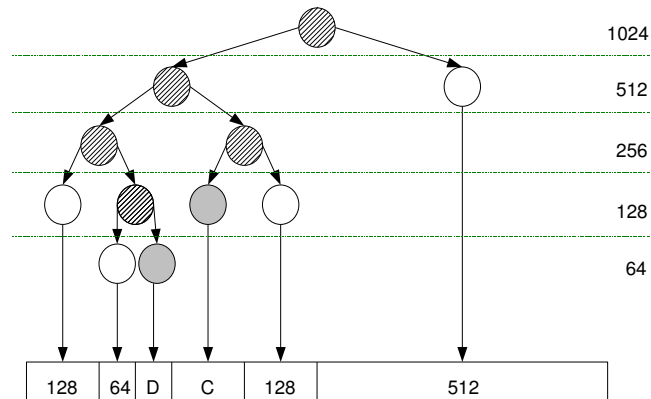
	0	128 K	256 K	384 K	512 K	640 K	768 K	896 K	1 M	
Anfang	1024									
Anfrage 70	A	128	256		512					
Anfrage 35	A	B	64	256		512				
Anfrage 80	A	B	64	C	128	512				
Freigabe A	128	B	64	C	128	512				
Anfrage 60	128	B	D	C	128	512				
Freigabe B	128	64	D	C	128	512				
Freigabe D	256		C		128	512				
Freigabe C	1024									

Segmentierung (1)

- Aufteilung von Programmen in mehrere **Segmente**, z. B. Codesegment, Datensegment(e), Stacksegment(e) etc.
- Jedes Segment entspricht einem **linearen Adressraum** von 0 bis zu einem Maximalwert.
- Adressangaben bestehen aus:
 - einer **Segmentangabe**
 - einer **relativen Adresse im Segment**
 (Zweidimensionaler Adressraum)

Buddy-System: Baum-Repräsentierung

Zustand vor der Freigabe von Prozess D



Segmentierung (2)

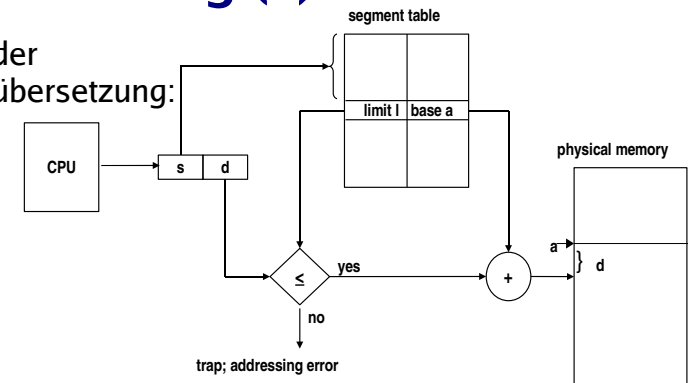
- **Vorteile** der Segmentierung:
 - Für die kleineren Segmente findet man leichter freien Hauptspeicher.
 - Segmente können unabhängig voneinander wachsen.
 - Es müssen nicht alle Segmente gleichzeitig im Hauptspeicher sein (**Swapping von Segmenten**).
 - Segmente können unterschiedlich geschützt werden.
 - Unterscheidung von prozess-privaten Segmenten und solchen Segmenten, die von mehreren Prozessen gemeinsam benutzt werden können.

Segmentierung (3)

- Segmentierung muss von der Hardware unterstützt werden.
- Für jeden Prozess gibt es eine **Segmenttabelle** im Speicher, die für jedes Segment mindestens drei Angaben enthält:
 - Ist das Segment im Hauptspeicher?
 - Anfangsadresse des Segments im Hauptspeicher
 - Länge des Segments

Segmentierung (5)

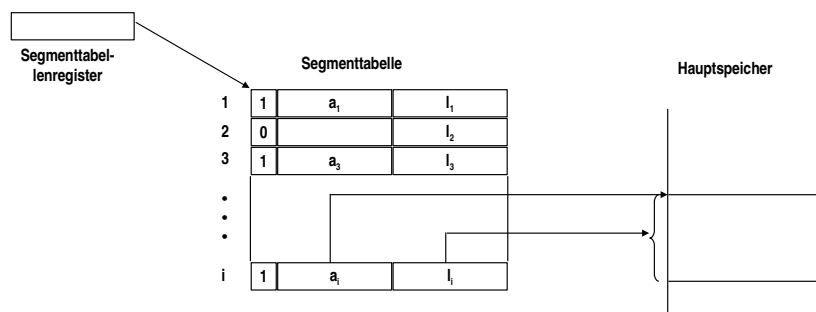
- Ablauf der Adressübersetzung:



- Für jeden Hauptspeicherzugriff ist ein zusätzlicher Speicherzugriff auf die Segmenttabelle nötig. Dies müssen Register in der Hardware beschleunigen

Segmentierung (4)

- Die Adresse der Segmenttabelle steht in einem **Segmenttabellenregister**.



Segmentierung (6)

- Zugriff auf ein Segment, das nicht im Hauptspeicher ist -> **Segment Fault**: Betriebssystem muss das Segment erst in den Hauptspeicher laden.
- Schutz eines Segments: Die Einträge in der Segmenttabelle enthalten zusätzlich einen **Schutzcode** (z. B. für die Zugriffe Lesen, Schreiben, Ausführen), der vom Programmierer (oder Compiler/Linker) festgelegt wird.

Segmentierung (7)

- **Sharing** eines Segments im Hauptspeicher ist **einfach**:
Man lässt die Einträge in den Segmenttabellen mehrerer Prozesse auf die gleiche Hauptspeicheradresse zeigen.
- **Fragmentierung**:
Da jedes Segment im Hauptspeicher zusammenhängend abgelegt wird, tritt **externe Fragmentierung** auf.

Ausgangslage

- Speicher zu knapp für große Programme
-> Overlay-Programmierung
- Programmteile dynamisch nachladen, wenn sie benötigt werden
- Programmierer muss sich um Aufteilung in Overlays kümmern

```

Sep 19 14:27:41 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -c 'severity=DEBUG')
Sep 20 01:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[30103]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd[6516]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd[6609]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd[6694]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:35 amd64 sshd[6077]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:35 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd[10102]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd[10140]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c 'severity=DEBUG')
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 02:00:01 amd64 /usr/sbin/cron[17878]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd[31088]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:39 amd64 sshd[31241]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[29278]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[5499]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:23:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[24720]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[25555]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd[6554]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:14 amd64 sshd[6606]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c 'severity=DEBUG')
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13255]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd[20998]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd[31371]: Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: end_seq_mid_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: end_seq_oss: unsupported module, tainting kernel.
Sep 24 20:25:31 amd64 sshd[29399]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:31 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:02 amd64 /usr/sbin/cron[662]: (root) CMD (/sbin/evlogmgr -c 'severity=DEBUG')
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c 'age > *30d*')
Sep 25 02:00:02 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd[8889]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd[8921]: Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:02 amd64 sshd[9372]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd[11594]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd[11586]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd[11608]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd[11630]: Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd[12930]: Accepted rsa for esser from ::ffff:87.234.201.207 port 62778
    
```

**Nichtzusammenhängende
(virtuelle)
Speicheraufteilung**

Overlay-Programmierung

Turbo Pascal, um 1985-90:

```
program grossesprojekt;
```

```
overlay procedure kundendaten;
```

```
...
```

```
overlay procedure lagerbestand;
```

```
...
```

```
{ Hauptprogramm }
```

```
begin
```

```
  while input <> "exit" do begin
```

```
    case input of
```

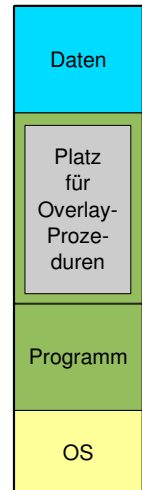
```
      "kunden": kundendaten;
```

```
      "lager": lagerbestand;
```

```
    end;
```

```
  end;
```

```
end.
```



Lösung des Problems

- Virtueller Speicher, der das gesamte Programm aufnehmen kann
- Programm sieht Speicherbereich, der ihm zur Verfügung gestellt wurde – wie viel wirklich vorhanden ist, spielt (für das Programm) keine Rolle

Virtuelle Speicherverwaltung (Paging)

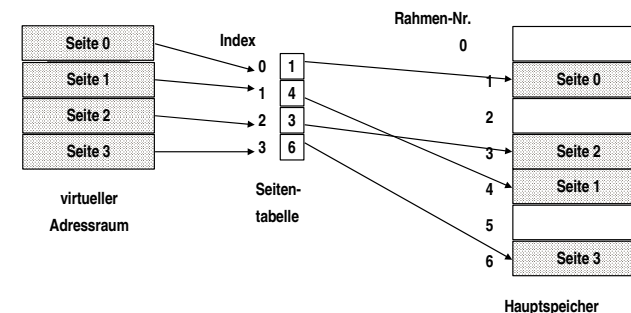
- Die Berechnung der **physikalischen** Speicheradresse aus der vom Programm angegebenen **virtuellen** Adresse
 - geschieht zur Laufzeit des Programms,
 - ist transparent für das Programm,
 - muss von der Hardware unterstützt werden.
- **Vorteile** der virtuellen Speicherverwaltung:
 - Einfache Zuteilung von Hauptspeicher.
 - Keine externe Fragmentierung, geringe interne Fragmentierung.
 - Kein Aufwand für den Programmierer.

Virtuelle Speicherverwaltung (Paging)

- Aufteilung des Adressraums in **Seiten (pages)** **fester Größe** und des Hauptspeichers in **Seitenrahmen (page frames)** **gleicher** Größe.
 - Typische Seitengrößen: 512 Byte bis 8192 Byte (immer Zweierpotenz).
- Der lineare, zusammenhängende Adressraum eines Prozesses („**virtueller Adressraum**“) wird auf beliebige, nicht zusammenhängende Seitenrahmen abgebildet.
- **Eine** einzige Liste freier Seitenrahmen wird vom Betriebssystem verwaltet.

Virtueller Adressraum (1)

- Beim Paging wird der Zusammenhang zwischen Programmadresse und physikalischer Hauptspeicheradresse erst zur Laufzeit mit Hilfe der Seitentabellen hergestellt.



Virtueller Adressraum (2)

- Die vom Programm verwendeten Adressen werden deshalb auch **virtuelle Adressen** genannt.
- Der **virtuelle Adressraum** eines Programms ist der **lineare, zusammenhängende Adressraum**, der dem Programm zur Verfügung steht.

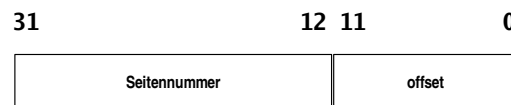
Adressübersetzung beim Paging (2)

- Für jeden Prozess gibt es eine **Seitentabelle (page table)**. Diese enthält für jede Prozessseite
 - eine Angabe, ob die Seite im Speicher ist,
 - die Nummer des Seitenrahmens im Hauptspeicher, der die Seite enthält.
- Ein spezielles Register enthält die Anfangsadresse der Seitentabelle für den aktuellen Prozess.
- Die Seitennummer wird als Index in die Seitentabelle verwendet.

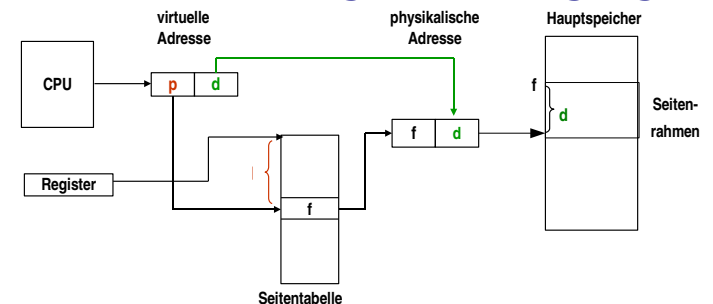
Adressübersetzung beim Paging (1)

- Die Programmadresse wird in zwei Teile aufgeteilt:
 - eine Seitennummer
 - eine relative Adresse (offset) in der Seite

Beispiel: 32-bit-Adresse bei einer Seitengröße von 4096 (=2¹²) Byte:



Adressübersetzung beim Paging (3)



- Für jeden Hauptspeicherzugriff wird ein zusätzlicher Hauptspeicherzugriff auf die Seitentabelle benötigt. Dies muss durch Caches in der Hardware beschleunigt werden!
- Seite nicht im Speicher -> spezielle Exception, einen sog. **page fault (Seitenfehler)** auslösen.

Virtueller Speicher allgemein (1)

- Mehr Prozesse können effektiv im Speicher gehalten werden
--> bessere Systemauslastung
- Ein Prozess kann viel mehr Speicher anfordern als physikalisch verfügbar

Virtueller Speicher allgemein (3)

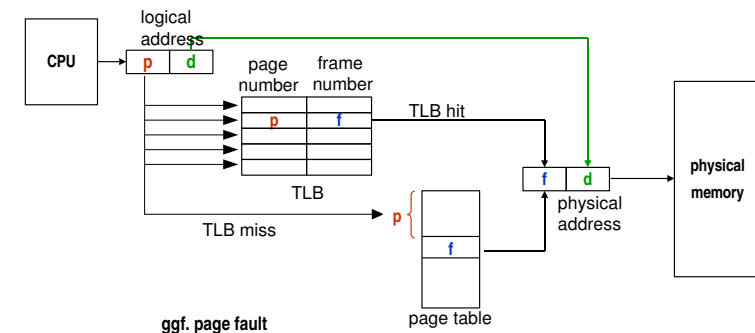
- „*thrashing*“ (siehe später): Prozessor verbringt die meiste Zeit mit Ein- und Auslagern von Prozessteilen statt mit der Ausführung von Prozessanweisungen
- **Lokalitätsprinzip:**
 - Zugriffe auf Daten und Programmcode häufig lokal gruppiert;
--> Annahme gerechtfertigt, dass nur wenige Prozessstücke während einer kurzen zeitlichen Periode gleichzeitig vorgehalten werden müssen

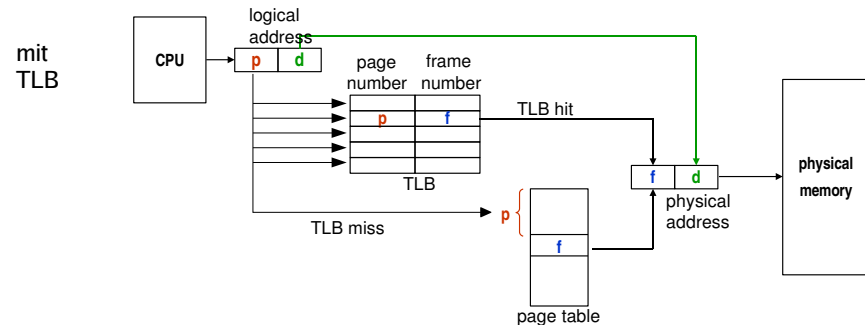
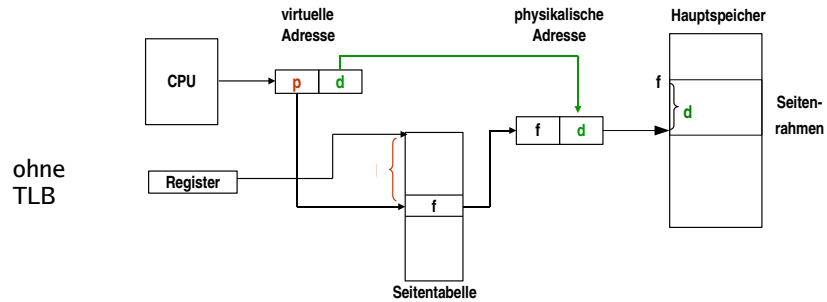
Virtueller Speicher allgemein (2)

- allgemeiner Vorgang:
 - Nur Teile des Prozesses befinden sich im physikalischen Speicher
 - falls Zugriff auf eine Adresse, die ausgelagert ist:
 - BS setzt den Prozess auf blockiert
 - BS setzt eine Disk-I/O-Leseanfrage ab
 - Nach Laden des fehlenden Stücks (Seite oder Segment) wird ein I/O-Interrupt abgesetzt
 - das BS setzt Prozess zuletzt wieder in den Bereit- (Ready-) Zustand

Translation Look-Aside Buffer (1)

- **Translation Lookaside Buffer (TLB):** schneller Hardware-Cache, mit den zuletzt benutzten Seitentableneinträgen
- **Assoziativ-Speicher:** bei Übersetzung einer Adresse wird deren Seitennummer gleichzeitig mit allen Einträgen des TLB verglichen.





Translation Look-Aside Buffer (3)

- Inhalt des TLB ist **prozessspezifisch!**
Zwei Möglichkeiten:
 - Jeder Eintrag im TLB enthält ein „valid bit“.
Bei **Prozesswechsel** (Context Switch) wird der gesamte Inhalt des TLB **invalidiert**.
 - Jeder Eintrag im TLB enthält Prozessidentifikation (**PID**), die mit der PID des zugreifenden Prozesses verglichen wird.
- **Beispiele** für TLB-Größen:
 - Intel 80486: 32 Einträge.
 - Pentium-4, PowerPC-604: 128 Einträge für jeweils Code und Daten.

Translation Look-Aside Buffer (2)

- **Treffer im TLB**
-> Speicherzugriff auf Seitentabelle unnötig
- **Fehlertreffer**
-> Zugriff auf die Seitentabelle
Alten Eintrag im TLB durch neuen ersetzen
- **Trefferquote (hit ratio)** beeinflusst die durchschnittliche Zeit einer Adressübersetzung.
- **Lokalitätsprinzip:** Programme greifen meist auf benachbarte Adressen zu
-> auch bei kleinen TLBs **hohe Trefferquoten** (typisch: 80-98%).

Translation Look-Aside Buffer und Speicher-Cache

