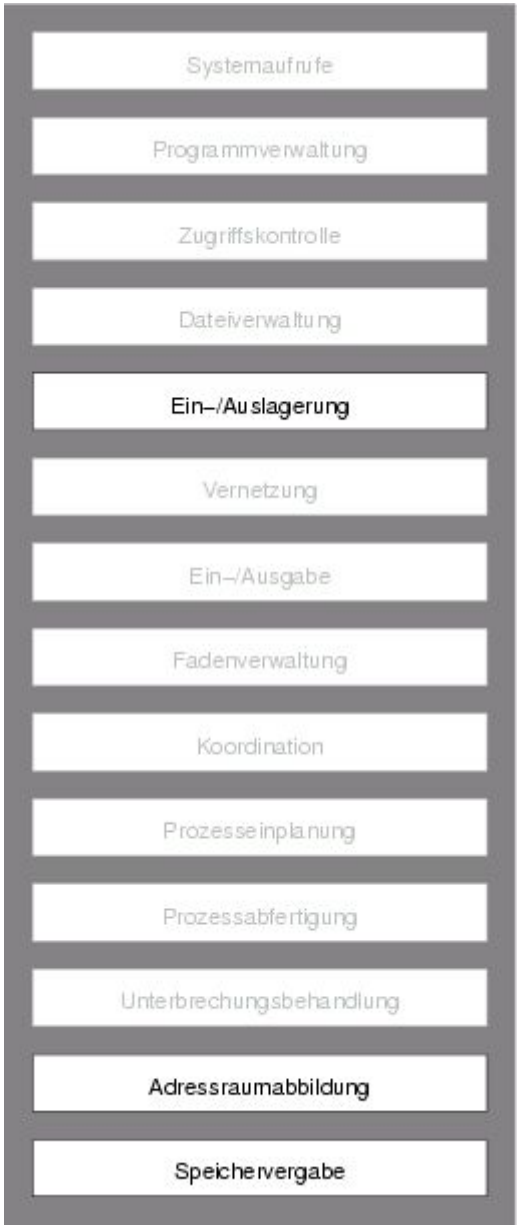
Speicherverwaltung



- Speicher als {physikalisches,logisches,virtuelles} Betriebsmittel
  - gekapselt in dazu korrespondierenden Adressräumen
- Vergabe zur *Ladezeit* und ggf. zur *Laufzeit*
  - statisch/dynamisch aus Programmsicht (”von oben“)
  - dynamisch aus Betriebssystemsicht (”von unten“)
- in den meisten Fällen ein knappes „Betriebsmittel“

Adressräume

physikalischer Adressraum definiert einen nicht-linear adressierbaren, von Lücken durchzogenen Bereich von E/A-Schnittstellen und Speicher (RAM, ROM), dessen Größe der Adressbreite der CPU entspricht

- 2^N Bytes, bei einer Adressbreite von *N* Bits

logischer Adressraum definiert einen linear adressierbaren Speicherbereich, dessen Größe selten der Adressbreite der CPU entspricht

- 2^M Bytes, $M < N$

virtueller Adressraum ein logischer Adressraum, der 2^N Bytes umfasst

Physikalischer Adressraum

Toshiba Tecra 730, 1996

Adressbereich	Belegung	Zugriffsresultat
00000000–0009ffff	RAM	r/w
000a0000–000c7fff	System	Protection Fault
000c8000–000dffff	keine	Bus Error
000e0000–000fffff	System	Protection Fault
00100000–090fffff	RAM	r/w
09100000–fffdffff	keine	Bus Error
fffe0000–ffffffff	System	Protection Fault

Logischer Adressraum

- ein *Programmadressraum* wird in (mind.) drei Abschnitte logisch aufgeteilt:

  Textsegement Maschinenanweisungen, Programmkonstanten

  Datensegment initialisierte Daten globale Variablen, Halde

  Stapelsegment lokale Variablen, Hilfsvariablen, aktuelle Parameter

- die *Memory Management Unit* setzt logische in physikalische Adressen um
  - je nach Rechnerarchitektur ist die MMU ein Teil oder Koprozessor der CPU
  - eine MMU verwendet Segmente oder/und Seiten als Verwaltungseinheiten
- das Betriebssystem bildet logische auf gültige physikalische Adressen ab

Adressraumausprägungen

eindimensional $\Rightarrow$ in Seiten aufgeteilt . . . . . (*paged*)

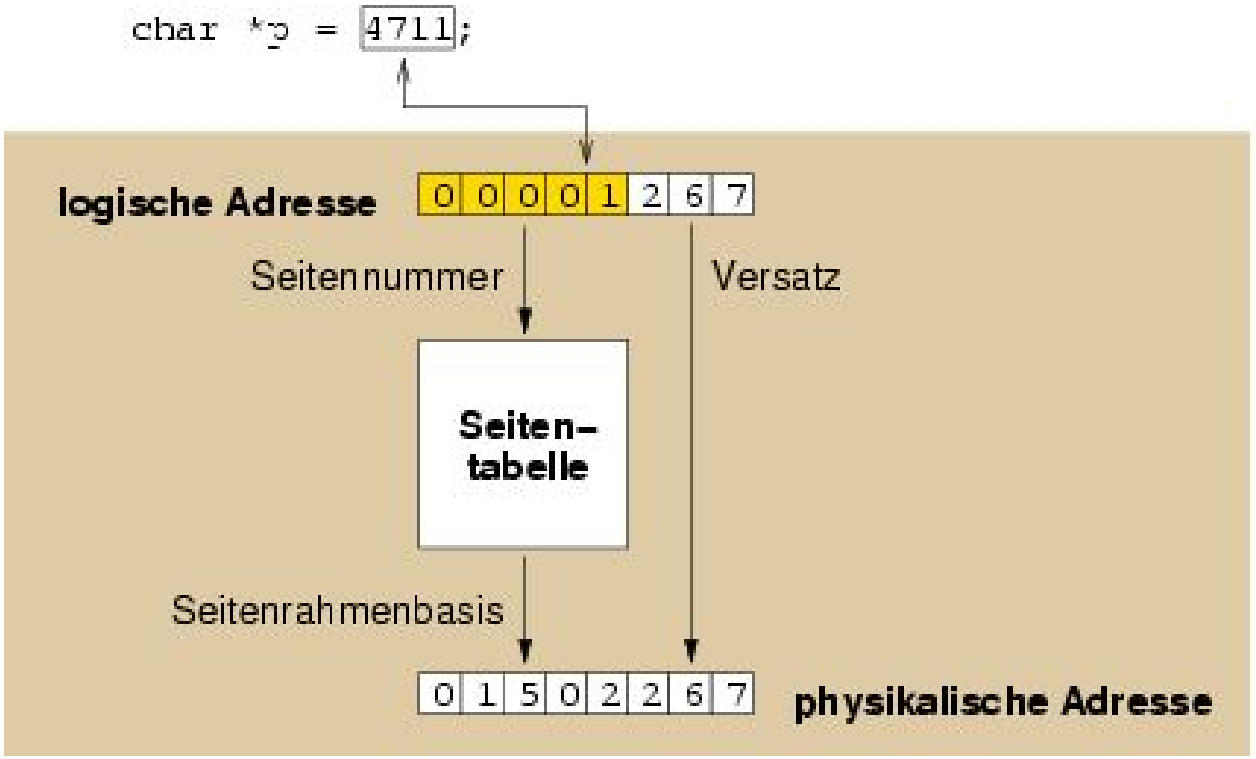
- der Prozessor interpretiert eine Programmadresse A_P als Tupel (p, o)
  - 2^N ist Seitengröße in Bytes

  Seitennummer (*page*) $p = A_P \text{ div } 2^N$

  Versatz (*offset*) $o = A_P \bmod 2^N$
- Seiten werden abgebildet auf (physikalische) Seitenrahmen, auch Kacheln

- zweidimensional $\Rightarrow$ in Segmente aufgeteilt . . . . . (*segmented*)
  - der Prozessor definiert eine Programmadresse A_S als Paar (S,A)
    - sind die Segmente gekachelt, dann wird A als A_P interpretiert
  - Segmente werden abgebildet auf einen (phys.) eindimensionalen Adressraum

Adressumsetzung – Gekachelter Adressraum

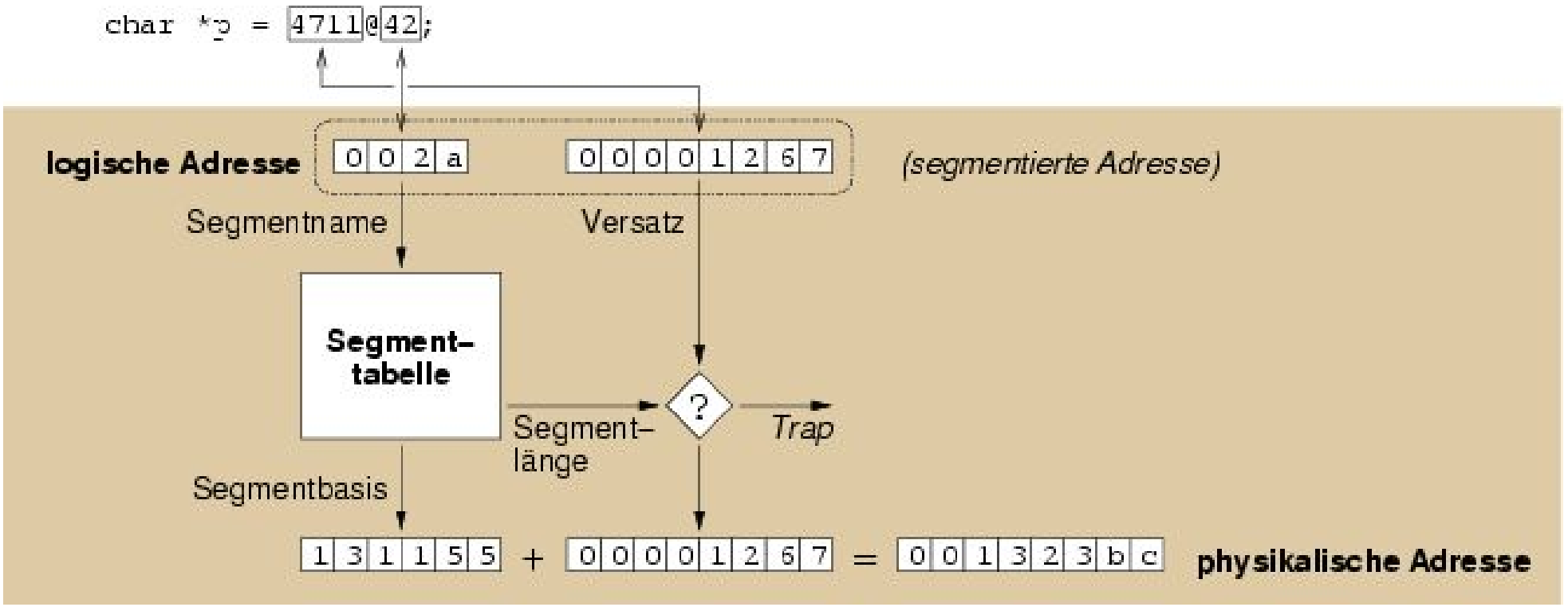


4711_{10} = 0x1267 (in C)

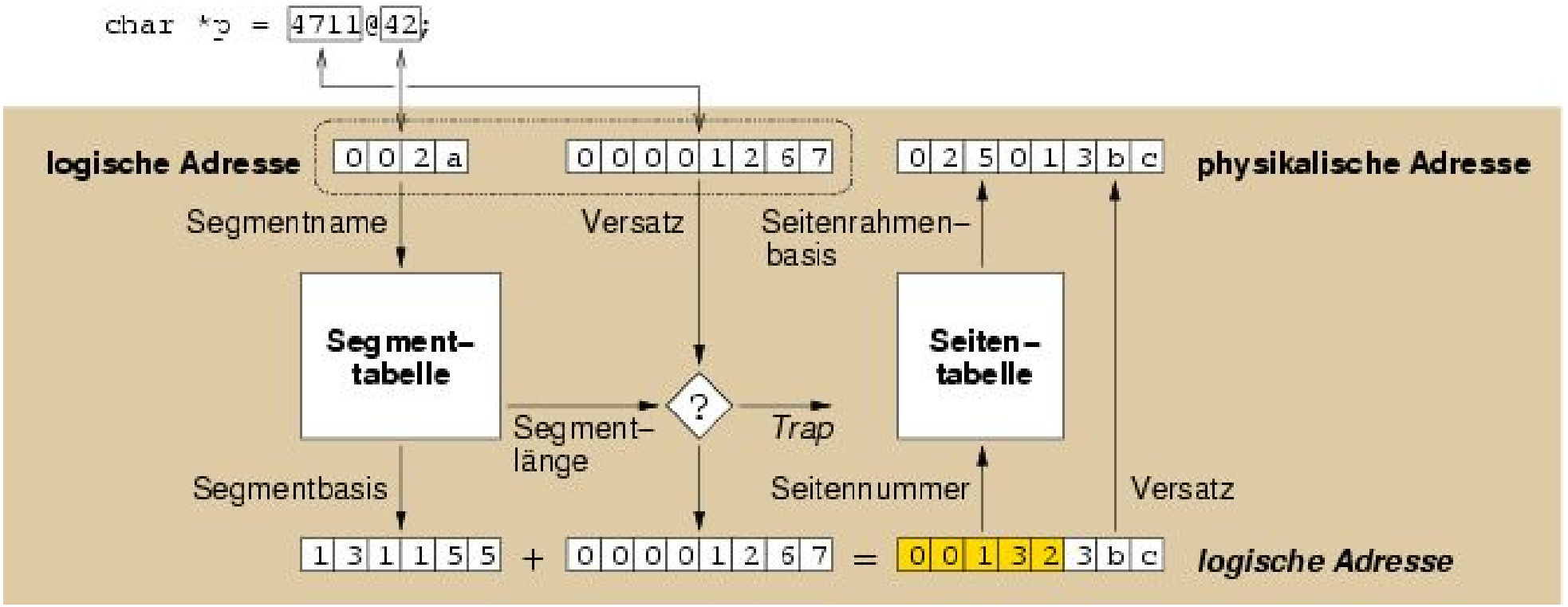
Eine Seitennummer ist ein Index in eine (pro Prozess vorhandende) Seitentabelle. Der dem Index entsprechende Seitendeskriptor enthält die Basisadresse des Seitenrahmens im physikalischen Adressraum.

$\Rightarrow$ Relokation

Adressumsetzung – Segmentierter Adressraum



Adressumsetzung – Segmentierte-gekachelter Adressraum



Attribute von Segmenten/Seiten (1)

- Für jede Seite bzw. jedes Segment existiert ein (Seiten-/Segment) **Deskriptor**, der Relokations- und Zugriffsdaten prozessbezogen verwaltet:
  - die physikalische Basisadresse des Seitenrahmens/Segments im Hauptspeicher
  - die Zugriffsrechte des Prozesses: lesen (*read*), schreiben (*write*)
- **Segmentdeskriptor** Segmente sind (im Gegensatz zu Seiten) von variabler, dynamischer Größe; als zusätzliche Verwaltungsdaten fallen an:
  - die Segmentlänge, um Segmentverletzungen abfangen zu können
  - die Expansionsrichtung: Halde (*"bottom-up"*), Stapel (*"top-down"*)

Lage und Ausdehnung von Seiten-/ Segementtabellen

base/limit-Registerpaar definiert die Anfangsadresse (*base*) einer Tabelle und die Anzahl (*limit*) der Tabelleneinträge

- bei der Adressumsetzung wird eine Indexprüfung wie folgt durchgeführt:

```
descriptor = (index < limit) ? &base[index] : trap this process
```

- wobei `index` eine Seitennummer oder einen Segmentnamen repräsentiert

- Die Inhalte dieser Prozessorregister gehören zum Prozesszustand:
  - initial bestimmt zur Ladezeit von Programmen $\Rightarrow$ exec(2)
  - aktualisiert zur Laufzeit der Programme $\Rightarrow$ brk(2)

Virtueller Adressraum

- Abstraktion von der Größe und Örtlichkeit des verfügbaren Arbeitsspeichers
  - vom Prozess nicht benötigte Programmteile können ausgelagert sein

  $\Rightarrow$ sie liegen im Hintergrundspeicher, z.B. auf der Festplatte
  - der Prozessadressraum könnte über ein Rechnernetz verteilt sein

  $\Rightarrow$ Programmteile sind über die Arbeitsspeicher anderer Rechner verstreut
- Zugriffe auf nicht eingelagerte Programmteile fängt der Prozessor ab: Trap
  - sie werden stattdessen partiell interpretiert vom Betriebssystem
  - das Betriebssystem zwingt den unterbrochenen Prozess in einen E/A-Stoß
  - die Wiederaufnahme des CPU-Stoßes führt zur Wiederholung des Zugriffs

Attribute von Segmenten/Seiten (2)

- je nach Konzept sind Segmente und/oder Seiten von der Einlagerung betroffen
  - {segmentierte, segmentiert-gekachelte, gekachelte} virtuelle Adressräume
- das ”present bit“ im (Segment/Seiten) Deskriptor regelt die Zugriffsart:

  1 $\Rightarrow$ eingelagert; Instruktion lesen, Operanden lesen/schreiben

  0 $\Rightarrow$ ausgelagert, Trap; partielle Interpretation des Zugriffs, Einlagerung

Segment-/Seitenfehler

present bit = 0 je nach Befehlssatz und Adressierungsarten der CPU kann der Behandlungsaufwand und Leistungsverlust beträchtlich sein

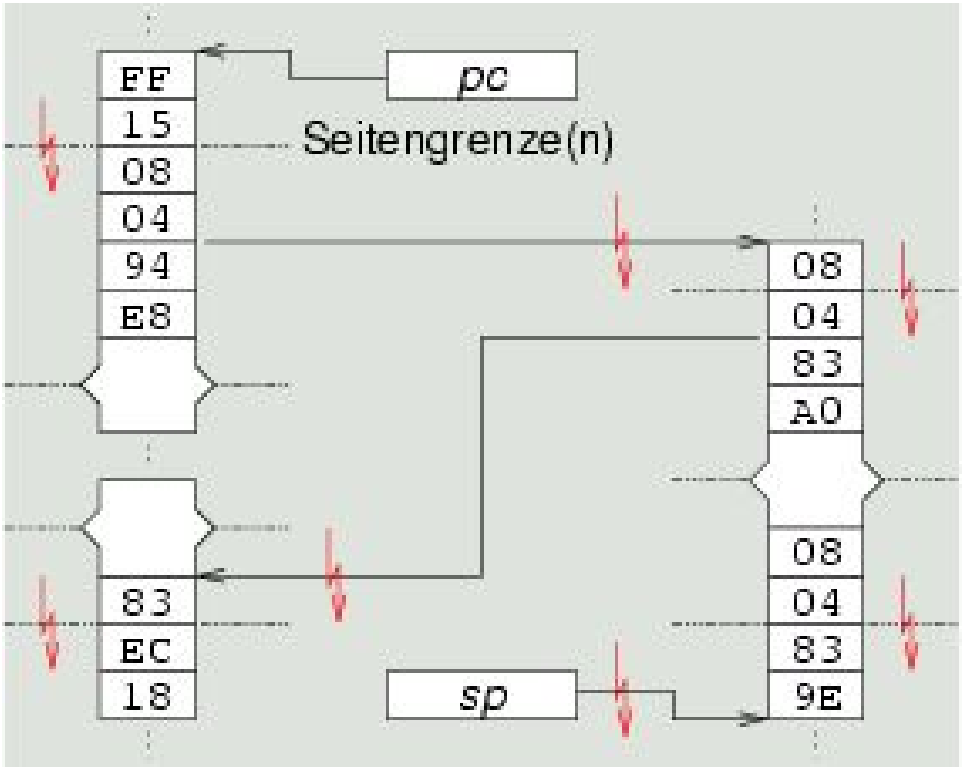
```
void hello () {
  printf("Hi!\n");
}

void (*moin)() = &hello;
main () {
   (*moin)();
}
```

```
main:
pushl %ebp
movl %esp,%ebp
pushl %eax
pushl %eax
andl $-16,%esp
call *moin
leave
ret
```

```
...
FF15080494E8
...
```

Fallstudie Seitenfehler

call *moin (x86) Prozeduraufruf, indirekte Adressierung des Unterprogramms über einen Zeiger (”pointer to function returning void“)



Bis zu sieben Seitenfehler können bei der Ausführung dieses einen Befehls auftreten:

1. Operandenadresse holen (08 04 94 E8)
2. Funktionszeiger lesen (08)
3. Funktionszeiger weiterlesen (04 83 A0)
4. Rücksprungadresse stapeln (08 04)
5. Rücksprungadresse weiterstapeln (83 9E)
6. Operationskode holen (83)
7. Operanden holen (EC 18)

Speicherzuteilung

statisch Benutzerprogrammen und Betriebssystem sind Gebiete maximaler, fester Größe zugewiesen

- innerhalb eines Gebiets kann Speicher jedoch dynamisch vergeben werden
- Probleme: Brachliegen von Betriebsmitteln, Leistungsbegrenzung/-verluste

  $\Rightarrow$ ungenutzter Speicher eines Gebiets ist in anderen Gebieten nicht nutzbar

  $\Rightarrow$ begrenzte E/A-Bandbreite mangels Puffer im Betriebssystemgebiet

  $\Rightarrow$ erhöhte Wartezeit von Prozessen wegen zu kleinen Puffern

dynamisch das Betriebssystem ermittelt ”Segmente“ angeforderter Größe im Arbeitsspeicher und teilt diese den Benutzerprogrammen bzw. sich selbst zu

Politiken bei der Speicherzuteilung

Platzierungsstrategie (*placement policy*) wohin die Information ablegen?

- wo der Verschnitt am kleinsten, am größten bzw. zweitrangig ist?

Ladestrategie (*fetch policy*) wann ist Information zu laden?

- auf Anforderung oder im Voraus?

Ersetzungsstrategie (*replacement policy*) welche Information ist zu verdrängen?

- die älteste, am seltensten genutzte oder am längsten ungenutzte?

Platzierungsstrategie

- verwaltet nicht-zugeteilten Speicher, definiert die Freispeicherorganisation:

  Bitkarte (*bit map*) freier Bereiche fester Größe

  - eignet sich für die Verwaltung gekachelter Adressräume
  - grobkörnige Vergabe freien Speichers auf Seitenrahmenbasis

  verkettete Liste (*free list*) freier Bereiche variabler Größe

  - ist typisch für die Verwaltung segmentierter Adressräume
  - feinkörnige Vergabe freien Speichers auf Segmentbasis

  $\Rightarrow$ freie Bereiche bilden ”Löcher“, die mit Programmen/Daten auffüllbar sind

- segmentierter Speicher ist aufwändiger zu verwalten als gekachelter
  - Löcher auf Rahmenbasis sind alle gleich gut, auf Segmentbasis nicht

Ladestrategie

- entscheidet, wann und wie die Einlagerung von Seiten/Segmenten erfolgt:

  Einzelanforderung (*on demand*) $\Rightarrow$ present bit

  - Seiten-/Segmentfehler (*page/segment fault*) führt zur Trap-Behandlung
  - Behandlungsergebnis ist die Einlagerung der angeforderten Speichereinheit

  Vorausladen (anticipatory)

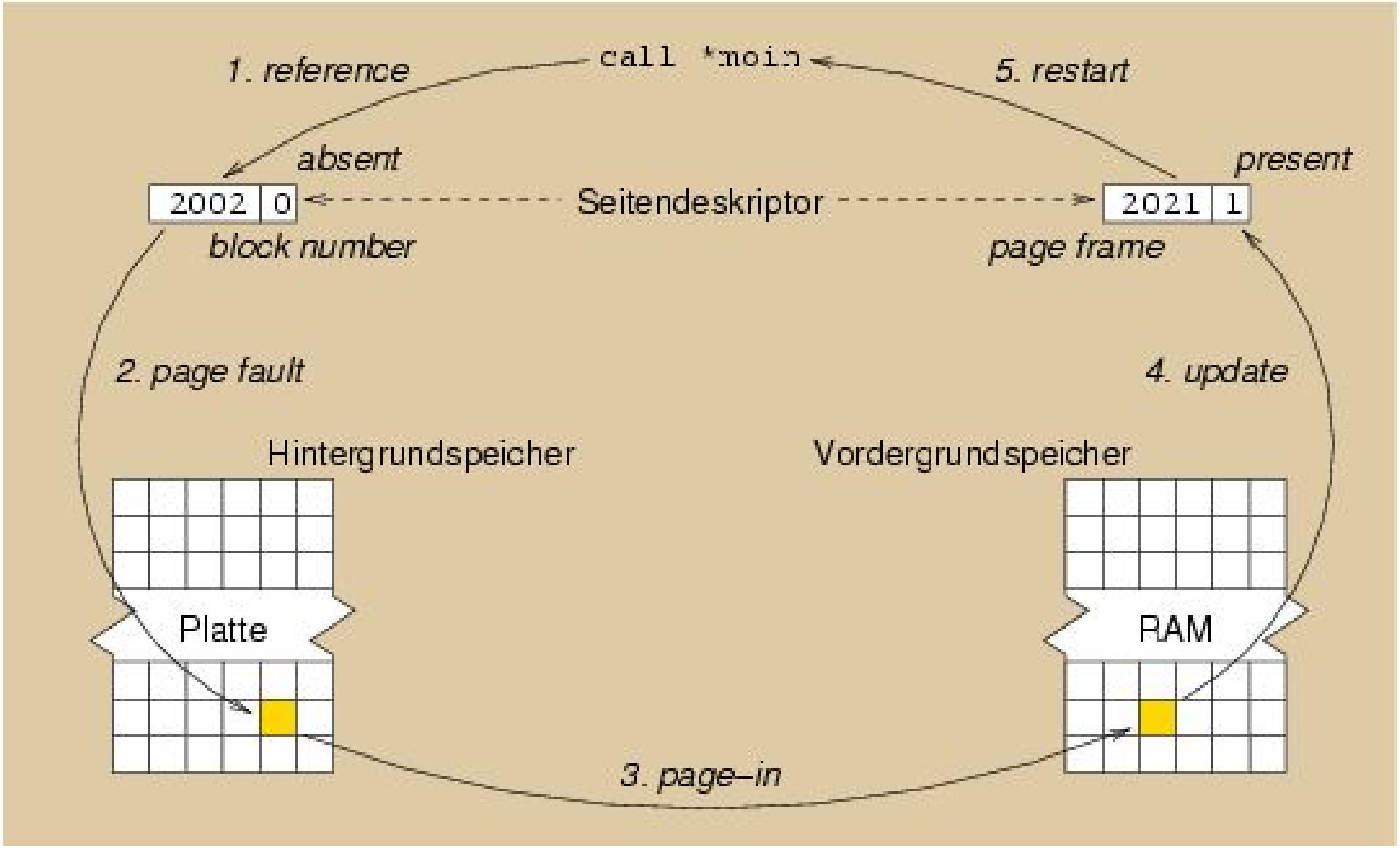
  - Heuristiken können Hinweise über zukünftige Zugriffsmuster liefern

  $\Rightarrow$ Programmlokalität, Arbeitsmenge (working set)

  - alternativ als Vorabruf (prefetch) im Zuge einer Einzelanforderung

- ggf. fällt die *Verdrängung* von Seiten/Segmenten an $\Rightarrow$ Ersetzungsstrategie

Einzelanforderung - *„on demand paging“*



Vorabruf

- Vorbeugung nachfolgender Seitenfehler — der ”call *moin“-GAU:
  1. den unterbrochenen Befehl dekodieren, die Adressierungsart feststellen
  2. da der Operand die Adresse einer Zeigervariablen (moin) ist, den Adresswert auf Überschreitung einer Seitengrenze prüfen
  3. da der Befehl die Rücksprungadresse stapeln wird, die gleiche Überprüfung mit dem Stapelzeiger durchführen
  4. in der Seitentabelle die entsprechenden Deskriptoren lokalisieren und prüfen, ob die Seiten anwesend sind: jede abwesende Seite ist einzulagern
  5. da jetzt die Zeigervariable (moin) vorliegt, sie dereferenzieren und ihren Wert auf überschreitung einer Seitengrenze pr¨ufen
  6. wie 4.
- partielle Interpretation des unterbrochenen Befehls durch das Betriebssvstem

Einzugsbereich bei Seitenersetzung

lokal nur Seitenrahmen des von der Seitenersetzung betroffenen Prozesses nutzen

- die Seitenfehlerrate ist von einem Prozess selbst kontrollierbar
  - Prozesse verdrängen niemals Seiten anderer Adressräume
  - fördert ein deterministisches Laufzeitverhalten von Prozessen
- statische Zuordnung von Seitenrahmen für den Prozessadressraum

global alle verfügbaren Seitenrahmen heranziehen; dynamische Zuordnung

- Verdrängung von Seitenrahmen/Ersetzung von Seiten ist unvorhersehbar
- adressraumübergreifende Beeinflussung des Laufzeitverhaltens von Prozessen

$\Rightarrow$ Kombination beider Ansätze ist möglich (Prozess-/ Adressraumklassen)

Seitenflattern

thrashing Flattern; Überlastung; die Dresche, Tracht Prügel; Niederlage

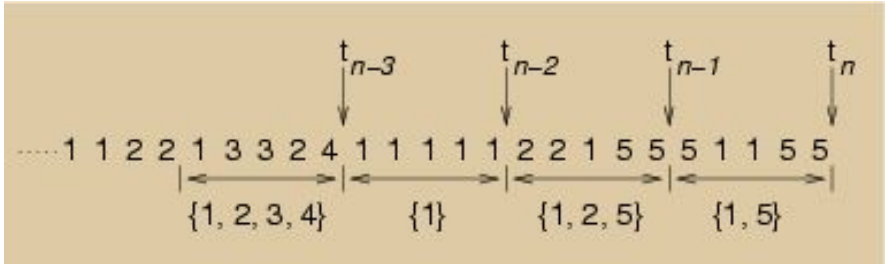
- Ein-/Auslagerung von Seiten bestimmt (phasenweise) die Systemaktivität
  - eben erst ausgelagerte Seiten werden sofort wieder eingelagert
  - Prozesse verbringen mehr Zeit beim ”*paging*“ als beim Rechnen
- ein die Systemleistung verringerndes Phänomen globaler Seitenersetzung
- Prozesse bewegen sich zu nahe am Seiten(rahmen)minimum
  - zu hoher Grad an Mehrprogrammbetrieb
  - ungünstige Ersetzungsstrategie
- Lösungsansätze:
  - ”*swapping*“ , (lokale/globale) Arbeitsmengen

$\Rightarrow$ steht in Relation zu der Zeit, die Prozesse mit sinnvoller Arbeit verbringen

Arbeitsmengen

working set die Menge der Seiten, die ein Prozess (lokal) bzw. das System(global) in naher Zukunft aktiv in Benutzung haben wird

- die Berechnung der Arbeitsmenge ist nur annäherungsweise möglich:
  - Ausgangspunkt ist die Seitenreferenzfolge der jüngeren Vergangenheit
  - Die Fensterbreite deckt eine ”feste Anzahl von Maschinenbefehlen“ ab, approximiert in Form von periodischen Unterbrechungen: sie ist bestimmt durch die Periodenlänge.
  - regelmäßig wird ein Fenster (*working set window*) darauf geöffnet
- kleine Fenster liefern wenig aktive Seiten, große zeigen Überlappungen



Zusammenfassung

- der logische Adressraum abstrahiert von den Widrigkeiten der Hardware
  - unbestückbare/reservierte Adressbereiche, Lücken in der Adressbelegung
  - die Illusion eines linear adressierbaren Speicherbereichs wird geschaffen
- der virtuelle Adressraum ermöglicht ”ortstransparente Speicherzugriffe“
  - nicht beötigte Programmteile liegen im Hintergrundspeicher
  - bei Bedarf/im Voraus werden sie in den Vordergrundspeicher transferiert
- die Verwaltung des (virtuellen) Speichers verfolgt dazu mehrere Politiken
  - Platzierungs-, Lade- und Ersetzungsstrategie für Seiten bzw. Segmente
  - der Fragmentierung wird durch Verschmelzung/Kompaktifizierung begegnet