Systemprogrammierung

Grundlagen von Betriebssystemen

Teil B – V.2 Betriebssystemkonzepte: Speicher

5. Juni 2025

Rüdiger Kapitza

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Technische Fakultät

Agenda

Gliederung

Lehrstoff

- behandelt werden Aspekte der **Speicherorganisation**
  - Dauerhaftigkeit von Datenhaltung und die Speicherpyramide
  - von Prozessen generierte Referenzfolge innerhalb ihrer Adressräume
  - Unterschied zwischen realen, logischen und virtuellen Adressraum
- die Grundkonzepte der **Speicherverwaltung** kennenlernen
  - Speicherzuteilung und -virtualisierung differenzieren
  - Auflösung gekachelter/segmentierter Speicherverwaltung erklären
  - wesentliche Merkmale von virtuellem Speicher im Ansatz vorstellen
- Bedeutung der sogenannten **Platzierungsstrategie** vermitteln
  - kurz segmentierte und gekachelte Verfahren vorstellen
  - Suchaufwand und Verschnitt gegenüberstellen
  - wichtige Verfahren (*best-*, *worst-*, *first-*, *next-fit*) benennen
- Vertiefung „dynamischer Speicher“, die **Halde**
  - Freispeicherverwaltung auf Maschinenprogrammebene
  - Interaktion zwischen Maschinenprogramm und Betriebssystem zeigen
  - zwischen lokaler und globaler Speicherverwaltung unterscheiden

Gliederung

Grundlagen

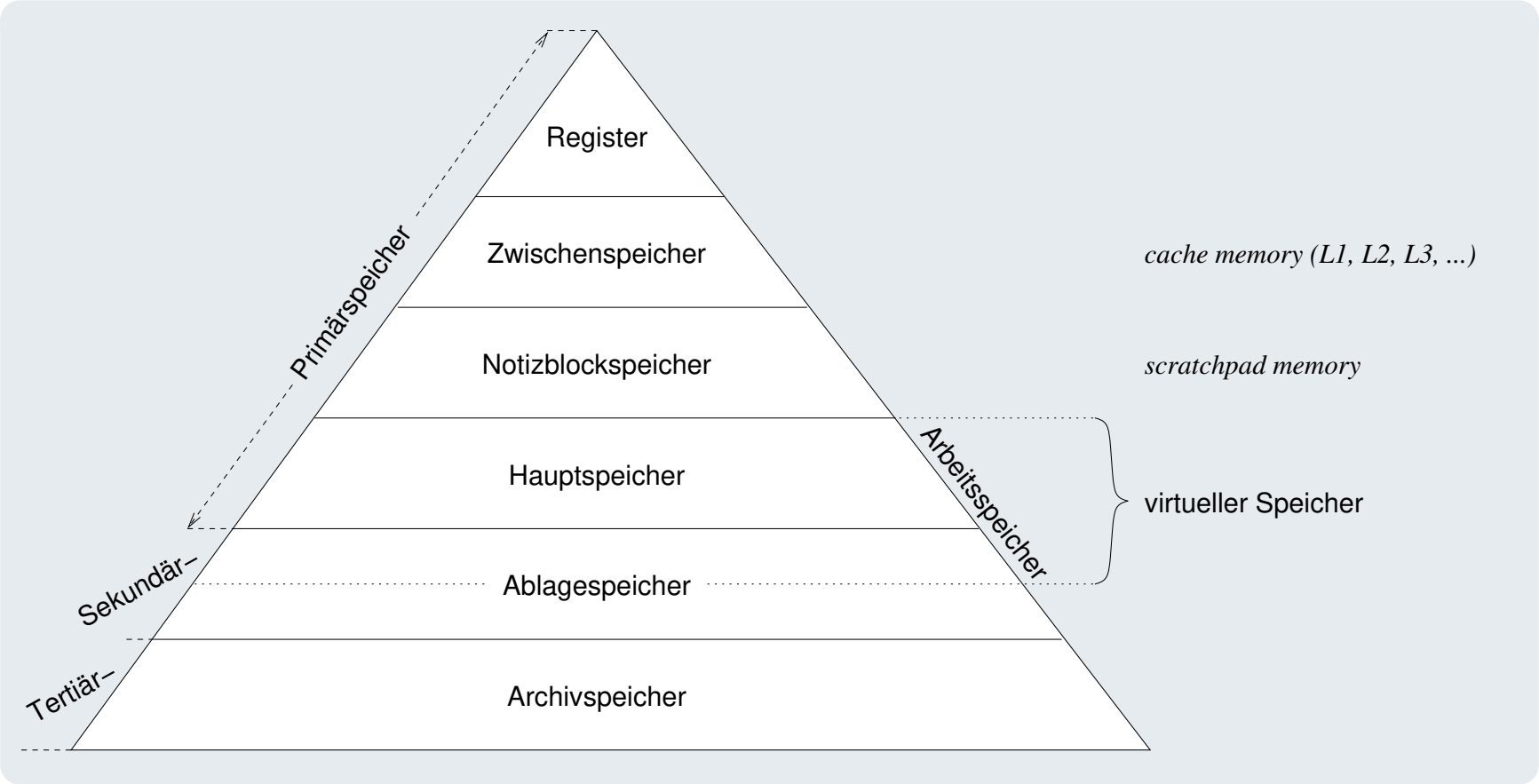
Speicherorganisation

Definition (Speicher)

von (lat.): *spicarium* Getreidespeicher, ein Ort oder eine Einrichtung zum Einlagern von materiellen oder immateriellen Objekten.

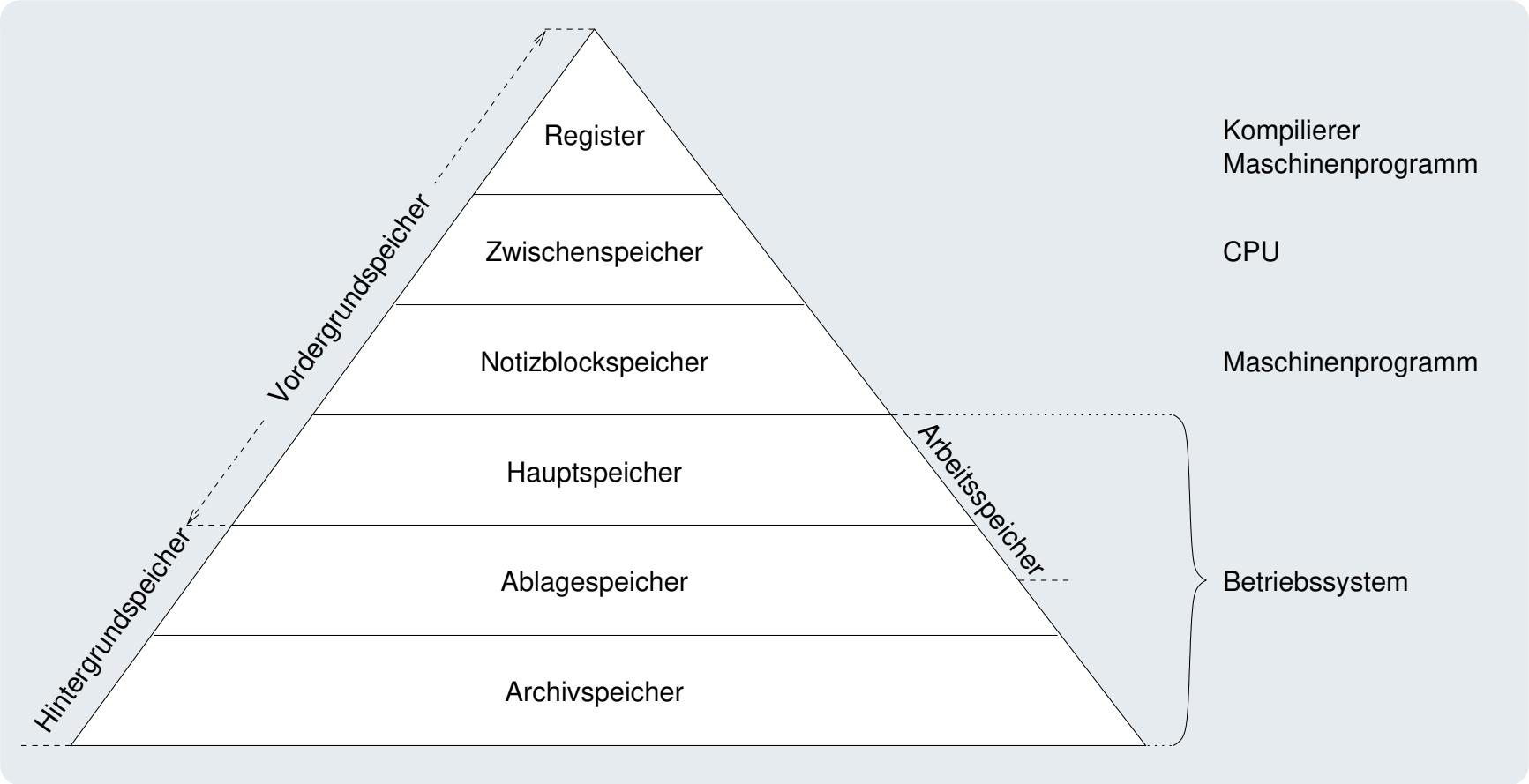
- Vorrichtung an elektro. Rechenanlagen, die eine kurz-, mittel- oder langfristige Speicherung von Informationen ermöglicht
  - **kurz**
    - hunderte von ns $\leq$ *Ladungshaltung*(*RAM*) $\leq$ dutzende von s
    - **Primärspeicher**: Register-, Zwischen-, Haupt-/Arbeitsspeicher
  - **mittel**
    - *Flash*-/Festplattenspeicher 2–10, im Mittel 5 Jahre
    - **Sekundärspeicher**: Arbeitsspeicher, Ablagesystem (Dateien)
  - **lang**
    - Festplattenarchive $\leq$ 30 Jahre $\leq$ Magnetbandarchive
    - optische Speicher (DVD), vermutlich 100 Jahre
    - in Zukunft basierend auf Glas, vermutlich 1000 Jahre [1]
    - **Tertiärspeicher**: Archiv
  - je größer die Zeitspanne, desto größer Kapazität und Zugriffszeit
- dabei sind Haupt-/Arbeitsspeicher und *bedingt* auch die Ablage in den Maschinenprogrammen direkt adressierbar
  - Multics [2] bildete Dateien auf Segmente im virtuellen Adressraum ab

Speicherpyramide I



- falls **virtueller Speicher** Merkmal der Maschinenprogrammebene ist, erstreckt sich der Arbeits- auf Hauptspeicher und Ablage
  - nur die **Arbeitsmenge** an Text/Daten ist im Hauptspeicher eingelagert
  - alle anderen Bereiche sind in der Ablage (*swap area*) ausgelagert

Speicherpyramide II



- bei **Mehrprogrammbetrieb** $\mapsto$ **Adressraumisolation** verwaltet jedes Maschinenprogramm seinen eigenen **Haldenspeicher**
  - Verwaltungsfunktionen (`malloc`/`free`) stellt das Laufzeitsystem (`libc`)
  - diese interagieren mit der Speicherverwaltung des Betriebssystems

Grundlagen

Adressraum

Referenzfolge

reference string

- ein **laufender Prozess** (vgl. [3, S. 20]) generiert Folgen von Adressen auf den Haupt-/Arbeitsspeicher, und zwar:
  - i nach Vorschrift des Programms, das diesen Prozess spezifiziert, wie auch
  - ii in Abhängigkeit von den Eingabedaten für den Programmablauf
- der zur Bildung dieser Adressen gegebene **Wertevorrat** hat anfangs eine feste Größe, dehnt sich gemeinhin dann aber weiter aus
  - dieser Vorrat ist initial statisch und gibt die zur Programmausführung mindestens erforderliche Menge an Haupt-/Arbeitsspeicher vor
  - zur Laufzeit ist diese **Menge** dynamisch, nimmt zu und kann dabei aber den „einem Prozess zugebilligten“ Wertevorrat nicht überschreiten
    - letzteres sichert entweder der Kompilierer oder das Betriebssystem zu
    - d.h., durch eine „typsichere Programmiersprache“ oder im Zusammenspiel mit der MMU der Befehlssatzebene
- der einem Prozess zugebillgte Adressvorrat gibt den **Adressraum** vor, in dem dieser Prozess (logisch/physisch) eingeschlossen ist
  - der Prozess kann aus seinem Adressraum normalerweise nicht ausbrechen und folglich nicht in fremde Adressräume eindringen
  - der **Prozessadressraum** hat eine durch (HW/BS) beschränkte Größe

Gedankenspiel I

- gegeben sei folgendes Programm in C:

```
1  #include <stdlib.h>
2
3  int main() {
4      while (1)
5          ((void(*)())random())();
6  }
```

- bzw. in ASM_{x86}:

```
12 main:
13   subl $12, %esp
14 .L2:
15   call random
16   call *%eax
17   jmp  .L2
```

- zu bestimmen sei die Referenzfolge eines diesbezüglichen Prozesses, unter folgender Annahme:

```
7  static void vain() { }
8
9  void (*(random)())() {
10     return vain;
11 }
```

```
18 vain:
19   rep
20   ret
21
22 random:
23   movl $vain, %eax
24   ret
```

  - C: $\ldots, \overline{5, 10, 7}$
  - ASM_{x86}: $\ldots, 13, \overline{15, 23, 24, 16, 19, 20, 17}$

Gedankenspiel II

- via gdb Einblick in den Prozessadressraum und damit Einsicht in den **statischen Wertevorrat** an Programmadradressen:

```
1   (gdb) info line main
2   No line number information available for address 0x80481c0 <main>
3   (gdb) disas 0x80481c0
4   Dump of assembler code for function main:
5      0x080481c0 <+0>: push   %ebp
6      0x080481c1 <+1>: mov    %esp,%ebp
7      0x080481c3 <+3>: and    $0xfffffff0,%esp
8      0x080481c6 <+6>: xchg   %ax,%ax
9      0x080481c8 <+8>: call   0x8048300 <random>
10     0x080481cd <+13>:    call   *%eax
11     0x080481cf <+15>:    jmp    0x80481c8 <main+8>
12  End of assembler dump.
13  (gdb) disas 0x8048300
14  Dump of assembler code for function random:
15     0x08048300 <+0>: mov    $0x80482f0,%eax
16     0x08048305 <+5>: ret
17  End of assembler dump.
18  (gdb) disas 0x80482f0
19  Dump of assembler code for function vain:
20     0x080482f0 <+0>: repz ret
21  End of assembler dump.
```

Hier ist nur die aus dem Befehlsabruf resultierende, statisch leicht bestimmbare Referenzfolge gezeigt. Es fehlen Referenzen, die die Stapeloperationen (`push`, `call` und `ret`) zur Folge haben. Diese sind mangels Kenntnis der Vorgeschichte des Prozesses allein durch „Programmlektüre“ nicht ableitbar.

- für den **Befehlsabruf** ergibt sich als **Referenzfolge** des Prozesses:
  - $\ldots, 0x08048[1c0, 1c1, 1c3, 1c6, \overline{1c8, 300, 305, 1cd, 2f0, 1cf}]$

Definition (realer Adressraum)

Der durch einen Prozessor definierte Wertevorrat $A_r = [0, 2^n - 1]$ von Adressen, mit $e \leq n \leq 64$ und (norm.) $e \geq 16$. Nicht jede Adresse in A_r ist jedoch gültig, d.h., A_r kann Lücken aufweisen.

- der **Hauptspeicher** ist adressierbar durch einen oder mehrere Bereiche in A_r, je nach Hardwarekonfiguration

Definition (logischer Adressraum)

Der in Programm P definierte Wertevorrat $A_l = \{[n_1, m_1], [n_2, m_2], ..., [n_i, m_i]\}$ von Adressregionen, mit $A_l \subset A_r$, der einem Prozess von P zugebilligt wird. Jede Adresse in A_l ist gültig, wobei die Regionen $[n_j, m_j]$ mit $j = 1..i$ sich nicht überlappen.

- führt **Arbeitsspeicher** ein, der partiell linear adressierbar ausgelegt ist und durch das Betriebssystem vollständig auf den Hauptspeicher abgebildet wird

Definition (virtueller Adressraum)

$A_v = A_l$: A_v übernimmt alle Eigenschaften von A_l. Jedoch nicht jede Adresse in A_v bildet ab auf ein im Hauptspeicher liegendes Datum.

- Benutzung einer solchen nicht abgebildeten Adresse in A_v verursacht in dem betreffenden Prozess einen **Zugriffsfehler**
- der Prozess erfährt eine **synchrone Programmunterbrechung** (*trap*), die vom Betriebssystem behandelt wird
- das Betriebssystem sorgt für die **Einlagerung** des adressierten Datums in den Hauptspeicher und
- der Prozess wird zur **Wiederholung** der gescheiterten Aktion gebracht

- der durch A_v für den jeweiligen Prozess benötigte Hauptspeicher ist „nicht in Wirklichkeit vorhanden, aber echt erscheinend“
- jedoch steht jederzeit genügend Arbeitsspeicher für A_v zur Verfügung
  - einesteils im Hauptspeicher, anderenteils in der Ablage (*swap area*)
- der Arbeitsspeicher ist eine virtuelle, der Hauptspeicher eine reale Größe

Gliederung

Speicherverwaltung

Einleitung

Aufgaben der Speicherverwaltung

- Aufgabe ist es, über die **Speicherzuteilung** an einen Prozess Buch zu führen und seine Adressraumgröße passend auszulegen
  - **Platzierungsstrategie** (*placement policy*)
    - wo im Hauptspeicher ist noch Platz?
- weitere Aufgabe kann die **Speichervirtualisierung** sein, um trotz knappem Hauptspeicher Mehrprogrammbetrieb zu maximieren
  - **Ladestrategie** (*fetch policy*)
    - wann muss ein Datum im Hauptspeicher liegen?
  - **Ersetzungsstrategie** (*replacement policy*)
    - welches Datum im Hauptspeicher ist ersetzbar?
- die zur Durchführung zu verfolgenden Strategien profitieren voneinander — oder bedingen einander
  - ein Datum kann ggf. erst platziert werden, wenn Platz freigemacht wurde
  - etwa indem das Datum den Inhalt eines belegten Speicherplatzes ersetzt
  - ggf. aber ist das so ersetzte Datum später erneut zu laden
  - bevor ein Datum geladen werden kann, ist Platz dafür bereitzustellen

„Reviere“ einer Speicherverwaltung

- normalerweise sind die **Verantwortlichkeiten** auf mehrere Ebenen innerhalb eines Rechensystems verteilt

  Speicherzuteilung
  - Maschinenprogramm und Betriebssystem
  - Haldenspeicher, Hauptspeicher

  Speichervirtualisierung
  - ist allein Aufgabe des Betriebssystems
  - Haupt-/Arbeitsspeicher, Ablage

- das Maschinenprogramm verwaltet den seinem Prozess (-adressraum) jeweils zugeteilten Speicher **lokal** eigenständig
  - stellt dabei **sprachenorientierte Kriterien** in den Vordergrund
    - typisch für den Haldenspeicher $\rightsquigarrow$ `malloc`/`free`
- das Betriebssystem verwaltet den gesamten Haupt-/Arbeitsspeicher **global** für alle Prozessexemplare bzw. -adressräume
  - stellt dabei **systemorientierte Kriterien** in den Vordergrund
    - hilft, einen Haldenspeicher zu verwalten $\rightsquigarrow$ z.B. `mmap`/`sbrk`
- Maschinenprogramm und Betriebssystem gehen somit eine **Symbiose** ein, sie nehmen eine **Arbeitsteilung** vor
  - genauer gesagt: das Laufzeitsystem (`libc`) im Maschinenprogramm

Speicherverwaltung

Speicherzuteilung

Granularität/Auflösung der Speicherzuteilung

- je nach Haupt-/Arbeitsspeicher (inkl. Halde) ergibt sich für die Verwaltung ein problemspezifischer Zerlegungsgrad wie folgt:

Wort
- Platzierungseinheit für Hauptspeicher
  - abhängig von CPU: Vielfaches von **Byte**

Zelle
- Platzierungseinheit für Halden- und Hauptspeicher
  - abhängig von Laufzeit-/Betriebssystem: 8 B, 16 B, 16/32 B

Kachel
- Platzierungs-, Lade- und Ersetzungseinheit für Arbeitssp.
  - abhängig von MMU: 512 B bis 1 GiB (typisch 4 KiB)

- diese Granulate werden zusammengefasst in **uniforme Segmente**, die einem Vielfachen der Granulatgröße entsprechen
  - wobei die Segmente die **Schutzeinheit** im logischen Adressraum bilden
  - bei entsprechender **Ausrichtung** (*alignment*) im Haupt-/Arbeitsspeicher

Platzierungsstrategie

Die Verfahren dazu sind je nach Speicherauflösung grob kategorisiert in **segmentierte** und **gekachelte Speicherverwaltung**.

Aspekte der Speicherzuteilung

segmentierte ↔ gekachelte Speicherverwaltung

Segmente können **verschieden groß** sein, ihre jeweiligen Attribute sind Granularität, Adresse und Länge. Kacheln dagegen sind **alle gleich groß**, sie unterscheiden sich lediglich durch ihre Adressen.

- um die „Granulate“ im Hauptspeicher platzieren zu können, ist Buch über nicht zugeteilte Speicherbereiche zu führen
  - freie Speicherbereiche werden in einer **Löcherliste** (*hole list*) geführt
    - ein freier Platz ist Loch, Lücke, Hohlraum (*hole*) zwischen belegten Plätzen
    - wobei der für ein Listenelement benötigte Speicher das Loch selbst sein kann
      ↪ Haldenspeicher: freie und belegte Plätze teilen denselben Adressraum ☺
  - die Liste ist sortiert nach Größe oder Adresse der vorhandenen Löcher
    - womit verschiedene Ziele bei der Zuteilungsstrategie verbunden sind (S. 23)
    - jedoch macht Sortierung nach Größe bei gekacheltem Speicher wenig Sinn
- wenn möglich, ist **Verschnitt** (*waste*) klein zu halten: *Zielkonflikt!*

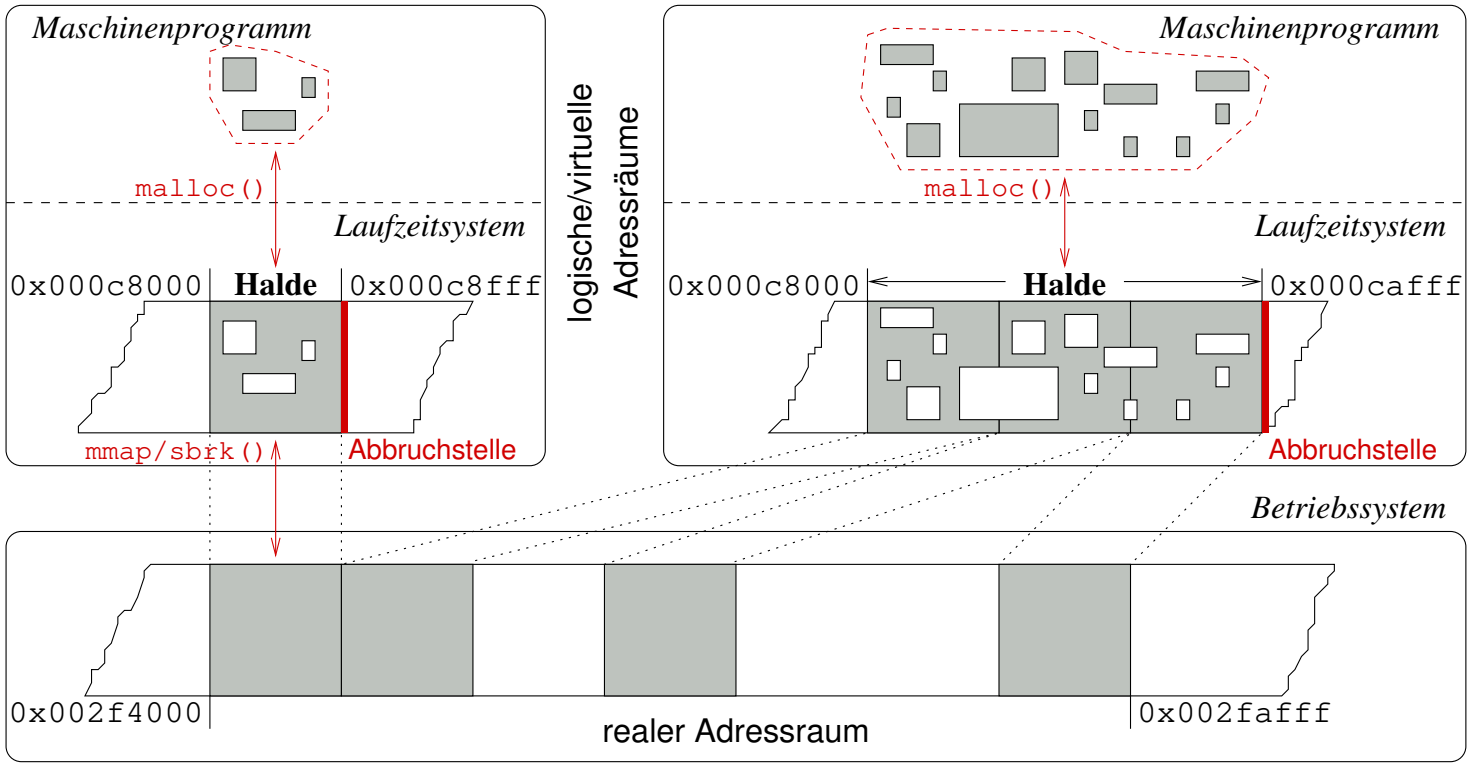
i einen Platz fester Größe zuteilen ⇝ Segmente kacheln
  - vermeidet *externen* Verschnitt auf Kosten *internen* Verschnitts

ii bei Freigabe zwei in A_r angrenzende Löcher zu einem Loch verschmelzen

Strategien der Speicherzuteilung

- **Sortierkriterien** der Freispeicherliste und damit verbundene **Ziele**:
  - ***best-fit***
    - zunehmende Größen, von vorne: **Verschnitt minimieren**
  - ***worst-fit***
    - abnehmende Größen, von vorne: **Suchaufwand minimieren**
  - ↪ beide Strategien machen die Einsortierung eines anfallenden Rests und somit einen zweiten Listendurchlauf notwendig
  - ↪ aus gleichem Grund ist Verschmelzung angrenzender Löcher zu einem großen Loch in beiden Fällen aufwendig
  - ***first-fit***
    - aufsteigende Adressen: **Laufaufwand minimieren**
  - ***next-fit***
    - wie zuvor, aber ab letzter Fundstelle: **Verschnitt nivellieren**
  - ↪ Verschmelzung angrenzender Löcher zu einem großen Loch ist in beiden Fällen unaufwendig
  - ↪ beide Strategien tendieren dazu, große Löcher zu zerschlagen und dadurch mehr Verschnitt zu generieren (FF mehr als NF)
- die „Einfachheit“ macht *first-* und *next-fit* zu guten Kompromissen…

Synergie bei der Speicherzuteilung



- das **Laufzeitsystem** verwaltet Speicher, der dem Adressraum eines einzelnen Maschinenprogramms vom Betriebssystem zugeteilt wurde
- das **Betriebssystem** verwaltet Speicher, der den Adressräumen aller Maschinenprogramme zugeteilt worden ist oder werden kann

Speicherrückgabe

…macht sehr wohl Sinn!

Speicherrückgabe durch `free` *an das Betriebssystem ist nicht nötig, da bei virtuellem Speicher ungenutzter Speicher schon rückgewonnen wird — kolportiert die Informatikfolklore.*

- Implementierung virtuellen Speichers basiert auf **Schätzungen**
  - Rückgewinnung ungenutzten Speichers leistet die Ersetzungsstrategie
  - alle dazu bekannten Verfahren greifen auf **Heuristiken** zurück
  - ob Speicher endgültig ungenutzt ist, weil er frei ist, bleibt daher ungewiss
  - nur das Maschinenprogramm selbst kann darüber Gewissheit haben
- eine Folge daraus ist, dass **Programmierfehler** unentdeckt bleiben
  - Adressen zu ungenutztem Haldenspeicher sind im Adressraum noch gültig
  - nur Rückgabe ans Betriebssystem kann diese Adressen ungültig machen
- zudem verursacht diese Heuristik **nichtdeterministische Prozesse**
  - eine kontraproduktive Eigenschaft für (festen/harten) Echtzeitbetrieb
  - deshalb implementiert nicht jedes Betriebssystem virtuellen Speicher…

Verschmelzung und Kompaktifizierung

Ist notwendig bzw. wünschenswert für große, rückgabefähige Löcher.

Speicherverwaltung

Speichervirtualisierung

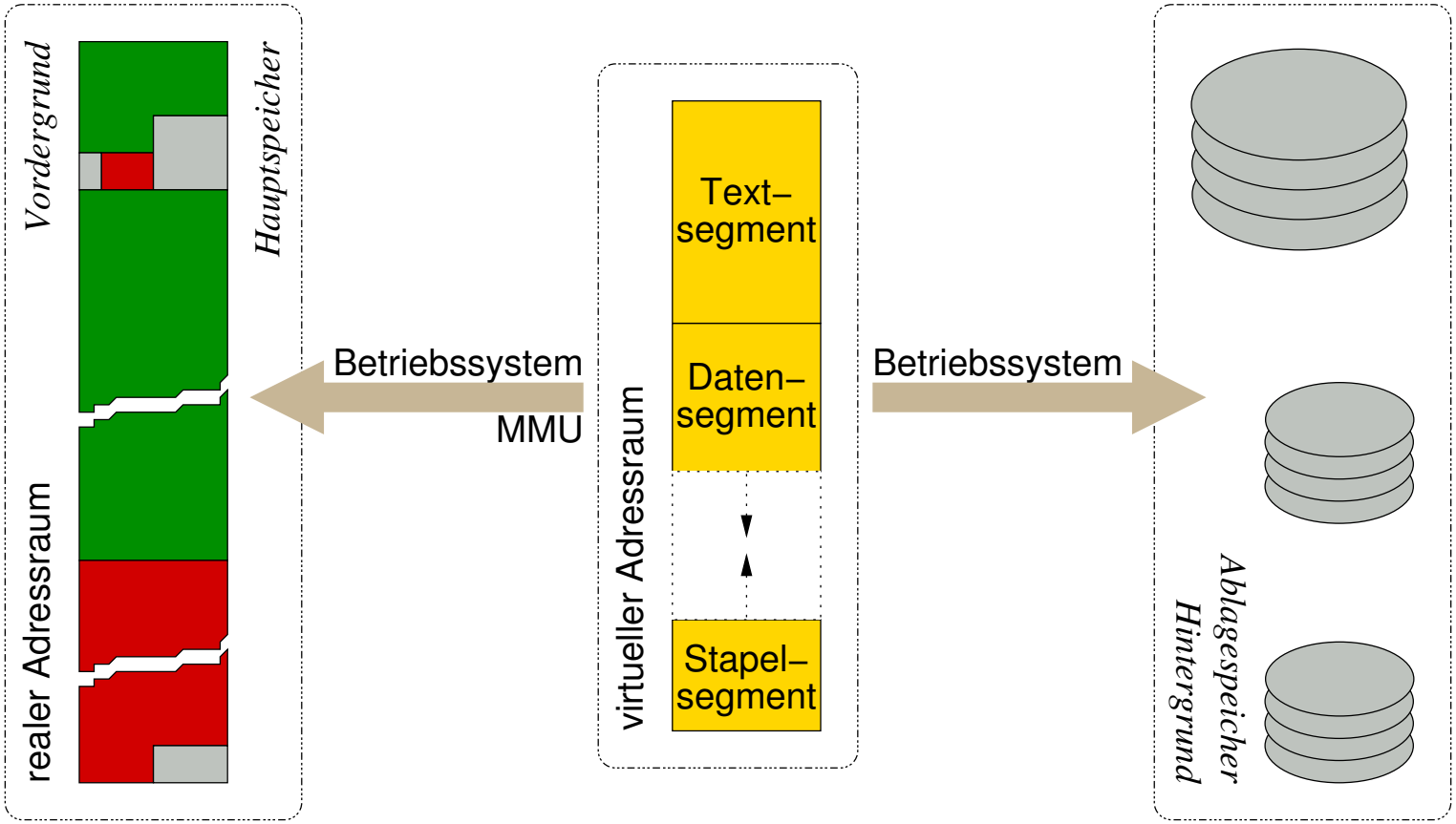
Virtueller Speicher

*Eine **Betriebssystemtechnik**, die laufende (ggf. nichtsequentielle) Prozesse ermöglicht, obwohl ihre Maschinenprogramme samt Daten nicht komplett im Hauptspeicher liegen.*

- sobald/solange Spannung anliegt, erfordert die Befehlssatzebene (d.h., der reale Prozessor) einen kontinuierlichen Befehlsstrom
  - anderenfalls gelingt der Befehlsabruf- und -ausführungszyklus nicht
- aus welchen Programmabläufen sich dieser Befehlsstrom ergibt, ist für die Befehlssatzebene jedoch nicht von Bedeutung
  - Abläufe innerhalb eines realen/logischen Adressraums erfordern, dass die betreffenden Programme vollständig im Hauptspeicher vorliegen
    - jede Adresse muss auf ein im Hauptspeicher vorliegendes Datum abbilden
  - im Gegensatz zu Abläufen innerhalb eines virtuellen Adressraums, der die **partielle Abbildung** einer Adresse auf ein Datum ermöglicht (vgl. S. 15)
- die durch die **Lokalität** eines Prozesses definierte **Referenzfolge** gibt die Adressen vor, die auf den Hauptspeicher abzubilden sind
  - alle anderen Adressen bilden ab auf die Ablage (*swap area*)

Partielle Abbildung virtueller Adressräume

(vgl. S. 15)



- Abbildungseinheit ist eine **Seite** (*page*),
  - Strukturierungselement sowohl des logischen als auch des virtuellen Adressraums ist
  - sie passt exakt auf eine Kachel, auch **Seitenrahmen** (*page frame*)

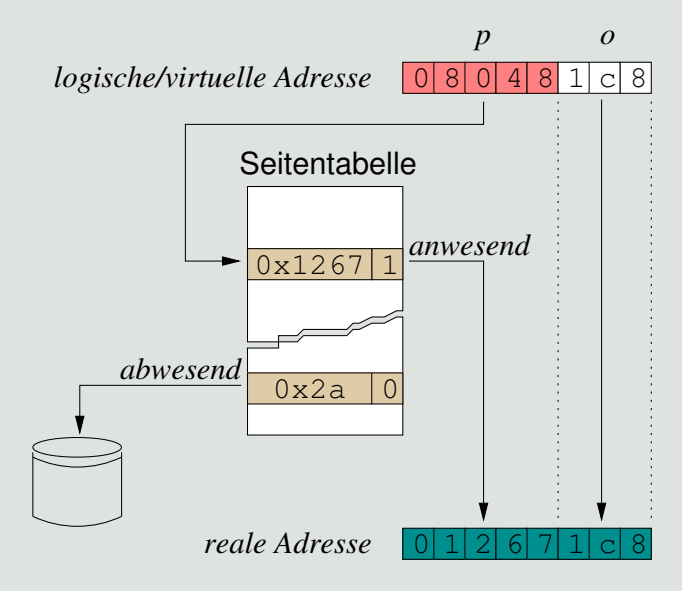
Abbildungstabelle

f : **Seiten** $\mapsto$ **Seitenrahmen d.h. Kacheln**

- eine ein- oder mehrstufig organisierte **Seitentabelle** (*page table*)
  - ein Feld (*array*), Datentypelement „**Seitendeskriptor**“ (*page descriptor*)
    - definiert durch das Betriebssystem, verarbeitet von der MMU
  - im Deutschen auch bezeichnet als „Seiten-Kachel-Tabelle“
- indiziert durch die **Seitennummer** (*page number*) einer Adresse
  - jede logische/virtuelle Adresse bildet ein Tupel $A = (p, o)$
  - mit Seitennummer p und Versatz (*offset*) o innerhalb dieser Seite
    - Wertevorrat $o = [0, 2^i - 1]$, mit $9 \leq i \leq 30$ (vgl. S. 21)
    - Wertevorrat $p = [0, 2^{n-i} - 1]$, mit $32 \leq n \leq 64$
    - mit p als Indexwert liest die MMU den A abbildenden Seitendeskriptor
- der Seitendeskriptor enthält die für die Abb. nötigen **Attribute**
  - Informationen über den gegenwärtigen Zustand der Seite
    - anwesend, referenziert, modifiziert, schreibbar, ausführbar, zugänglich, …
  - Kachelnummer/-adresse im Hauptspeicher (falls anwesend, eingelagert) oder Blocknummer in der Ablage (falls abwesend, ausgelagert)
- jeder Zugriff auf eine abwesende, ausgelagerte Seite verursacht einen **Seitenfehler** (*page fault*) $\rightsquigarrow$ Teilinterpretation des Zugriffs

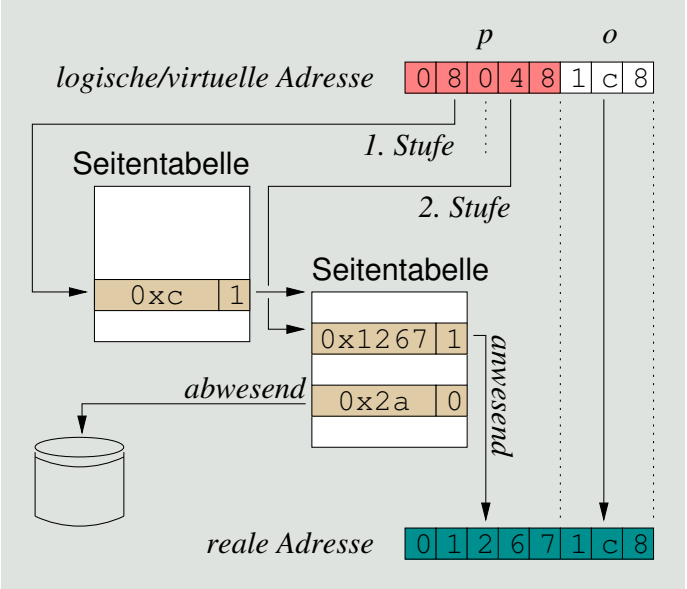
Abbildung (Prinzip)

- angenommen, die CPU ruft folgenden Befehl zur Ausführung ab:
  `0x080481c8 <+8>: call 0x8048300 <random>` (vgl. S. 13)
  - einstufige Abbildung



  - *base*/*limit* Registerpaar (MMU) grenzt die Seitentabelle ein
  - verschieden große Seitentabellen

  - zweistufige Abbildung (x86)



  - *base* Register (MMU) lokalisiert die Seitentabelle der 1. Stufe
  - gleich große Seitentabellen

↪ ***Trap***, falls $p \geq$ *limit* (li.) oder ungültiger/leerer Seitendeskriptor (beide)

Gliederung

Resümee

…Speicherrückgabe ist nicht obsolet

- behandelt wurde die **Speicherorganisation** von Rechensystemen
  - Primär-, Sekundär- und Tertiärspeicher als die drei Hauptkategorien
  - Verfeinerungen sind Haupt-, Arbeitsspeicher und Ablage
  - Bausteine von Vorder- und Hintergrundspeicher zur Programmausführung
  - bilden Ebenen einer **Speicherpyramide** (auch: Speicherhierarchie)
- die für Speicherverwaltung relevante **Adressraumlehre** präsentiert
  - **Referenzfolgen** sind wichtig, um virtuellen Speicher zu begreifen
  - Bezugspunkt dabei ist die **Adressraumart**: real, logisch, virtuell
  - logischer und virtueller Adressraum sind zwei verschiedene Konzepte:

    totale Abbildung $f : A_l \to A_r$ von logischen auf realen Adressen

    partielle Abbildung $f : A_v \rightsquigarrow A_r$ von virtuellen auf realen Adressen
- Aufgaben der Speicherverwaltung sind in **Politiken** untergliedert
  - Platzierungs-, Lade- und Ersetzungsstrategie
  - erstere meint Speicherzuteilung, letztere beiden Speichervirtualisierung
  - virtueller Speicher ermöglicht Prozesse unvollständiger Programme
- das **Ausmaß** eines virtuellen Adressraums kann riesig sein
  - der Hauptspeicher im realen Adressraum ist indes verschwindend klein…

Zusammenfassung

Bibliographie

Literaturverzeichnis (1)

[1] Anderson, P. ; Aranas, E. B. ; Assaf, Y. ; Behrendt, e. a.:
Project Silica: Towards Sustainable Cloud Archival Storage in Glass.
In: *Proceedings of the 29th Symposium on Operating Systems Principles*, Association for Computing Machinery, 2023 (SOSP '23). –
ISBN 9798400702297, S. 166–181

[2] Daley, R. C. ; Dennis, J. B.:
Virtual Memory, Processes, and Sharing in MULTICS.
In: *Communications of the ACM* 11 (1968), Mai, Nr. 5, S. 306–312

[3] Kleinöder, J. ; Schröder-Preikschat, W. :
Prozesse.
In: Lehrstuhl Informatik 4 (Hrsg.): *Systemprogrammierung*.
FAU Erlangen-Nürnberg, 2015 (Vorlesungsfolien), Kapitel 6.1