Betriebssystem-Entwicklung

2. Aufgabe: Speicherverwaltung

Michael Schöttner

Aspekte einer Speicherverwaltung

Heinrich Heine
Universität Düsseldorf

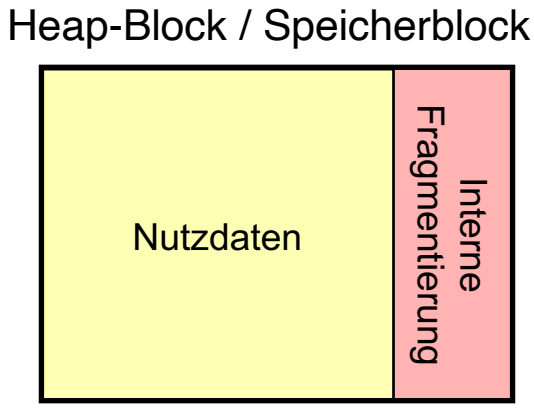
- Verschnitt (interne und externe Fragmentierung)
- Belegungsdarstellung
- Wiedereingliederung von unbenutzten Blöcken

Verschnitt: interne Fragmentierung

Heinrich Heine
Universität Düsseldorf

- **= Wird bei einer Speicherallokation mehr Speicher zugeteilt als angefordert, so geht der ungenutzte Platz verloren**
  - Es macht aber keinen Sinn bei einer Allokation sehr kleine Stücke übrigzulassen
    → diese Heap-Blöcke müssen auch verwaltet werden und sind nicht nutzbar

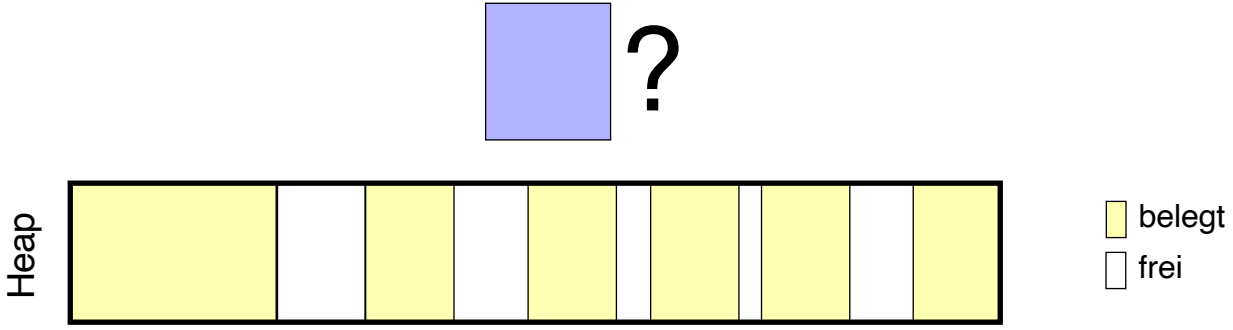
- Werden viele kleine Blöcke alloziert, so ist dies besonders kritisch
  - Beispiel: alloziert werden 28 Byte,
    man erhält 32 Byte → 4 Byte verloren
  - Wenn dies eine Milliarde Mal passiert
    verlieren wir 4 GB Hauptspeicher



Verschnitt: interne Fragmentierung

- Durch Allokationen und Freigaben von Heap-Blöcken entstehen im Laufe der Zeit verstreute Freispeicherblöcke im Heap

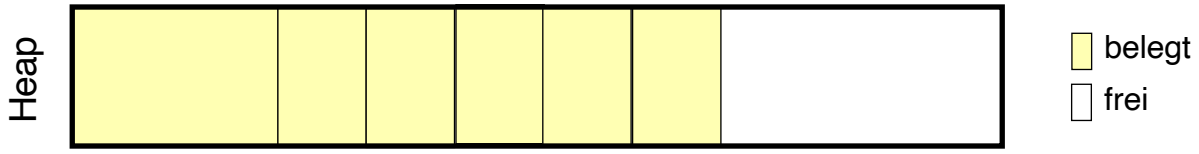
- Erfolgt nur eine Allokation für einen großen Speicherblock, so kann diese nicht bedient werden, obwohl genügend Speicherplatz vorhanden ist,
aber nicht zusammenhängend →



Verschnitt: interne Fragmentierung

Heinrich Heine
Universität Düsseldorf

- Durch **Kompaktieren** des Heaps ist dieses Problem lösbar

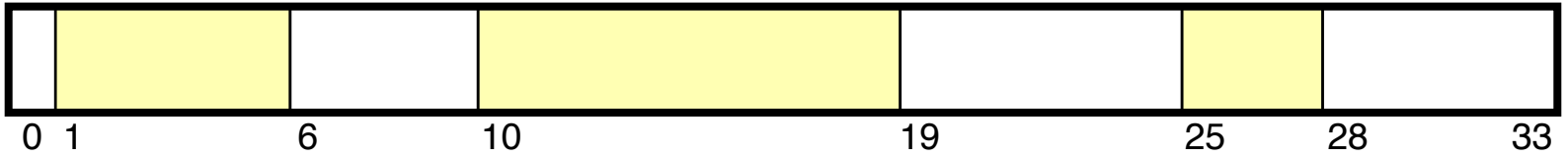


- Dies ist bei vielen Blöcken zeitaufwändig und nur mit Hardware-Unterstützung möglich.

Belegungsdarstellung: Freispeicher-Tabelle

Heinrich Heine
Universität Düsseldorf

- Freie Heap-Blöcke werden in einer separaten Tabelle verwaltet
- Beispiel (f=frei, b=belegt, jeweils KB): 1f, 5b, 4f, 9b, 6f, 3b, 5f



Größe	Adresse
1	0
4	6
5	28
6	19

Belegungsdarstellung: Freispeicher-Tabelle

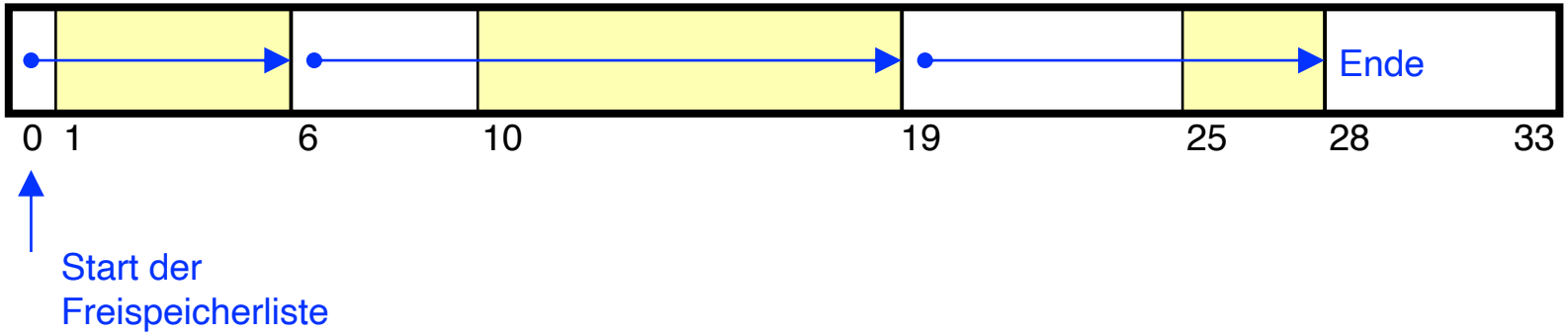
Heinrich Heine
Universität Düsseldorf

- Vorteil:
  - Einfach zu realisieren
  - Tabelle kann separat gespeichert werden und
    so vor unabsichtlichem Überschreiben durch das Programm geschützt werden

- Nachteile:
  - Der Speicherbedarf für die Tabelle ist unklar
    - Dies hängt von der Anzahl freien Blöcke ab.
    - Diese kennen wir aber vorab nicht
  - Wird ein Block belegt, so entsteht eine Lücke in der Tabelle, sodass wir den freien Speicher in der Tabelle ebenfalls wieder verwalten müssen.

Belegungsdarstellung: Freispeicher-Liste

- Freie Heap-Blöcke mit Zeiger verketten
- Beispiel (f=frei, b=belegt, jeweils KB): 1f, 5b, 4f, 9b, 6f, 3b, 5f



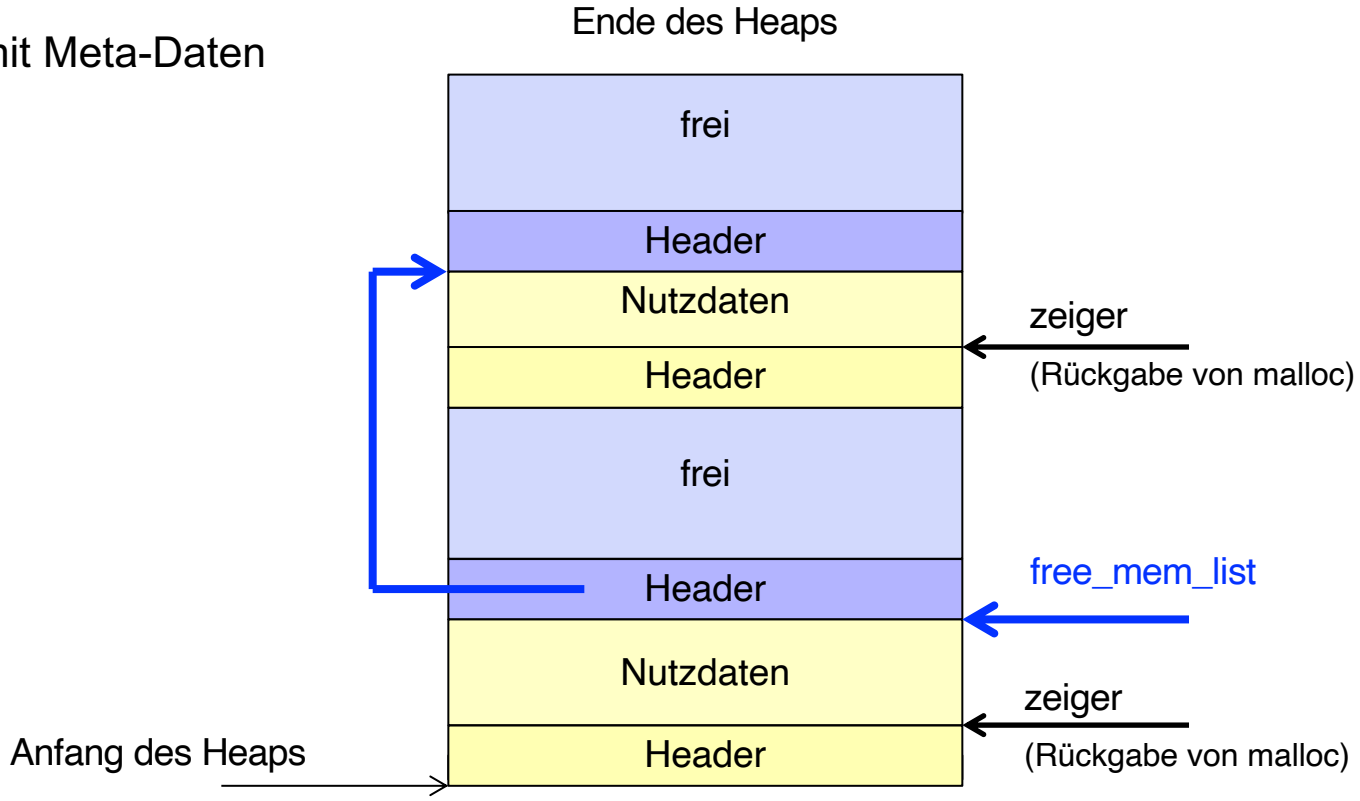
Belegungsdarstellung: Freispeicher-Liste

Heinrich Heine
Universität Düsseldorf

- Vorteile:
  - Benötigt keine zusätzlichen Speicher, da freier Speicher dafür genutzt wird
- Nachteile:
  - Problematisch ist, wenn die Kette kaputtgeschrieben wird.
    Dann sind viele freien Blöcke verloren.
  - Zudem ist die sequentielle Suche nach einem passenden Block langsam
- Optimierung
  - Eventuell mehrere Listen verwenden, um verschiedene Größenordnungen separat zu verketten
    - Man benötigt dafür nur mehrere Listeneinstiegspunkte
      (jeweils einen pro Größenordnung)

Belegungsdarstellung: Freispeicher-Liste

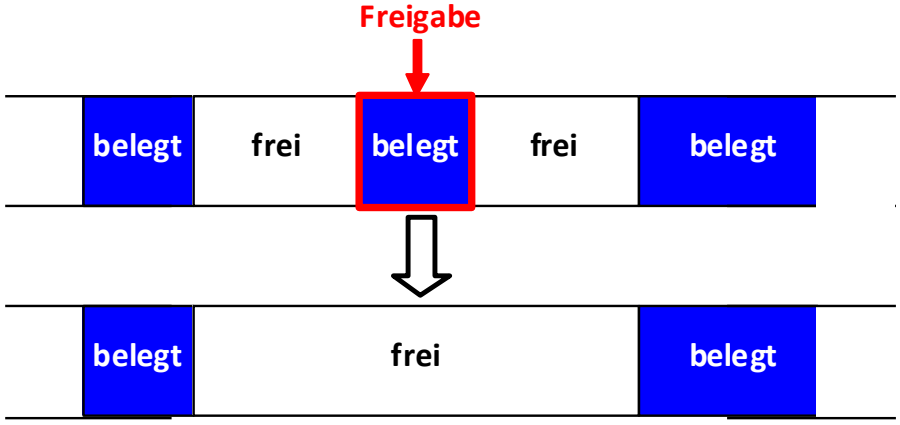
- Detail-Ansicht mit Meta-Daten



Wiedereingliederung

- Bei Freigabe eines Speicherblocks prüfen, ob unmittelbar davor- oder danach angrenzende Blöcke frei sind und gegebenenfalls zusammenfassen.

- Hiermit entstehen wieder größere Blöcke.



Speicherverwaltung für C++

Heinrich Heine
Universität Düsseldorf

- Wir legen in hhuTOSc den Heap an die Adresse 4 MB (mit fester Größe)
- Dies reicht für unsere Zwecke, da wir keine Prozesse keine Privilegstufen verwenden
- Für die Implementierung reichen zwei Funktionen:
  - `mm_alloc(unsigned int size);`
  - `mm_free(void *ptr);`

- Damit die C++ Funktion `new` und `delete` unser `mm_alloc` und `mm_free` nutzen, müssen die Operatoren für `new` und `delete` überschrieben werden, u.a.
  - `void* operator new ( size_t size );`
  - `void operator delete ( void* ptr );`
  - Alle notwendigen Operatoren sind in der Vorgabe vollständig implementiert

Speicherverwaltung für Rust

Heinrich Heine
Universität Düsseldorf

- Wir legen in hhuTOSr den Heap an die Adresse 3 MB (mit fester Größe)
- Dies reicht für unsere Zwecke, da wir keine Prozesse keine Privilegstufen verwenden
- Für die Implementierung benötigen wir im Wesentlichen zwei Funktionen:
  - `alloc(&mut self, layout: Layout) -> *mut u8)`
  - `dealloc(&mut self, ptr: *mut u8, layout: Layout);`

- Zusätzlich sind noch einige andere Dinge zu beachten, welche in der Vorgabe sind und im Blog von Philipp Oppermann sehr schön beschrieben sind: https://os.phil-opp.com/heap-allocation/