



Betriebssystem- Entwicklung

2. Aufgabe: Speicherverwaltung

Michael Schöttner

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

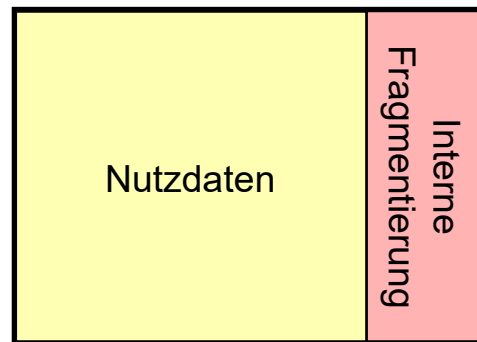
- = Wird bei einer Speicherallokation mehr Speicher zugeteilt als angefordert, so geht der ungenutzte Platz verloren

- Es macht aber keinen Sinn bei einer Allokation sehr kleine Stücke übrigzulassen
→ diese Heap-Blöcke müssen auch verwaltet werden und sind nicht nutzbar

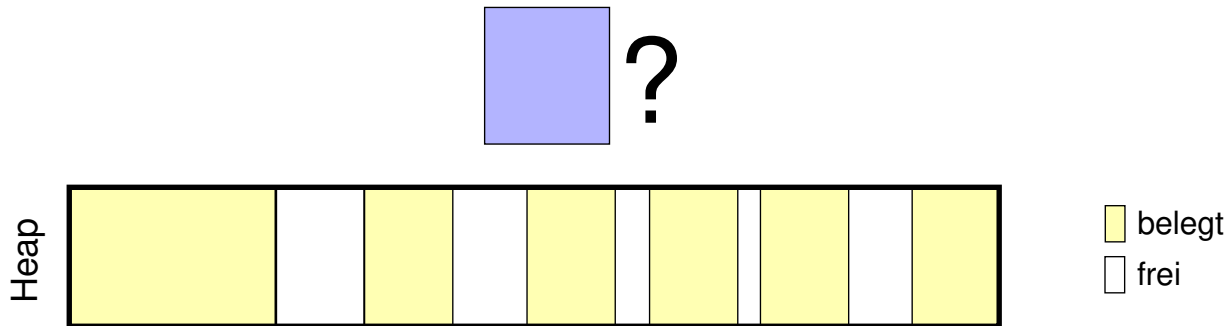
- Werden viele kleine Blöcke alloziert, so ist dies besonders kritisch

- Beispiel: alloziert werden 28 Byte, man erhält 32 Byte → 4 Byte verloren
- Wenn dies eine Milliarde Mal passiert verlieren wir 4 GB Hauptspeicher

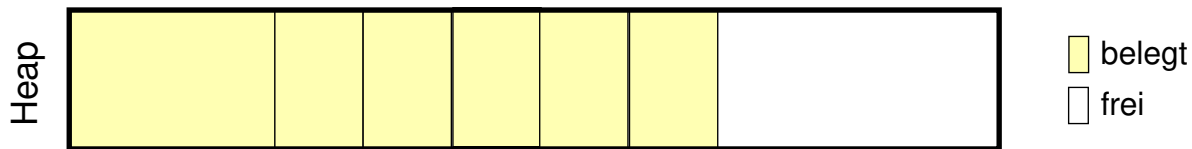
Heap-Block / Speicherblock



- 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 →

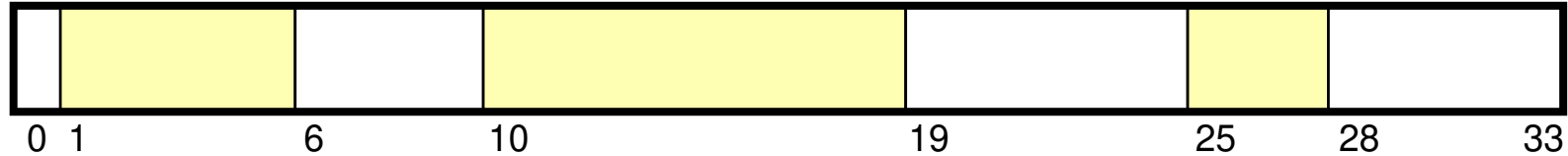


- Durch **Kompaktieren** des Heaps ist dieses Problem lösbar



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

- 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

■ Vorteil:

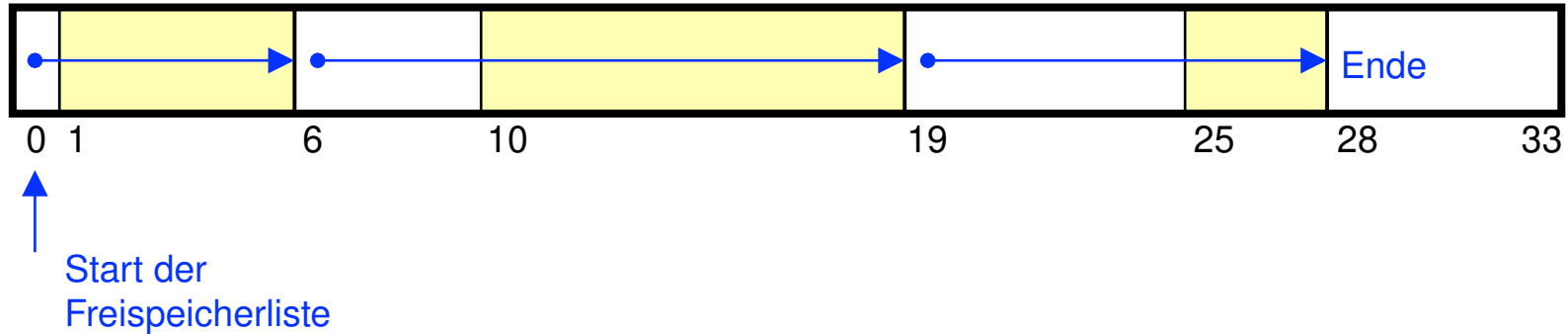
- Einfach zu realisieren
- Tabelle kann separat gespeichert werden und so vor unabsichtlichem Überschreiben durch das Programm geschützt werden

■ 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



■ Vorteile:

- Benötigt keine zusätzlichen Speicher, da freier Speicher dafür genutzt wird

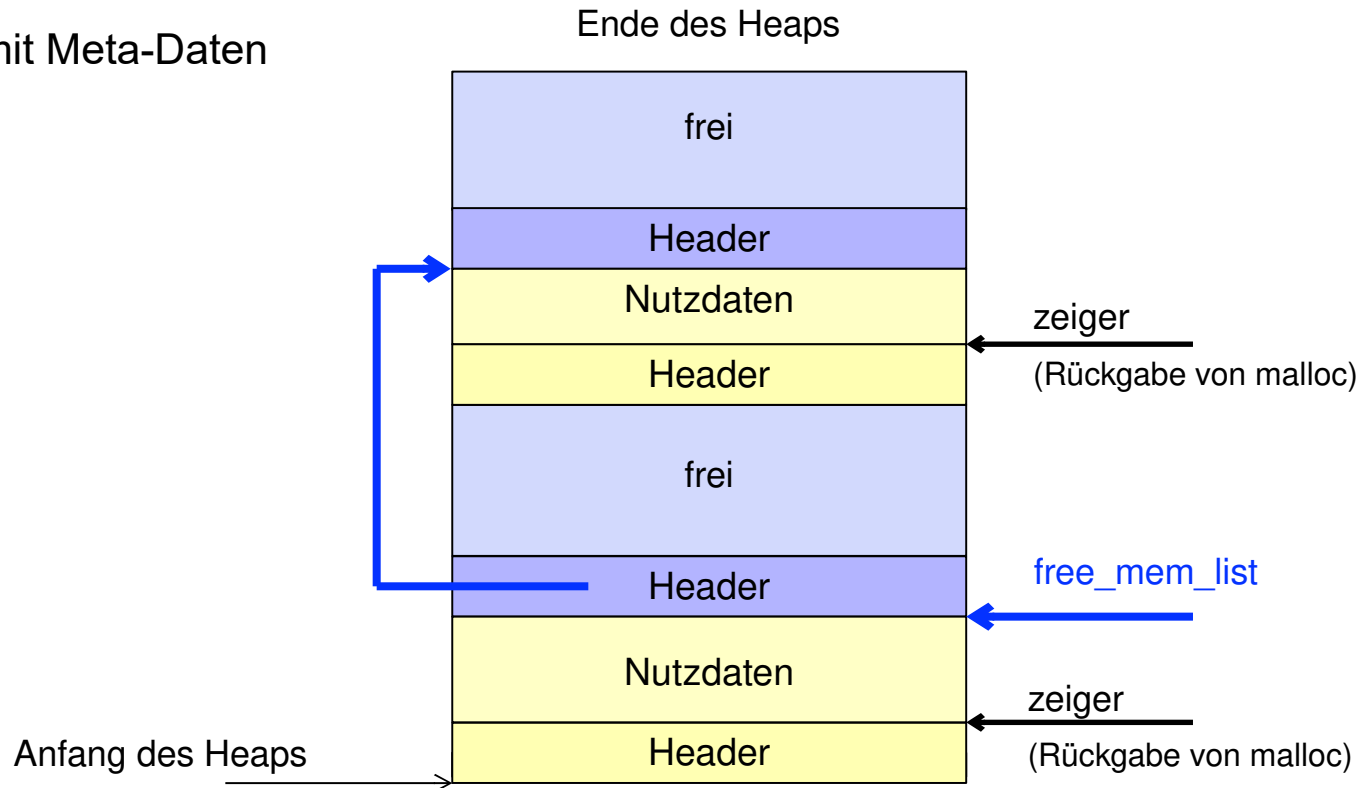
■ Nachteile:

- Problematisch ist, wenn die Kette kaputtgeschrieben wird.
Dann sind viele freien Blöcke verloren.
- Zudem ist die sequentielle Suche nach einem passenden Block langsam

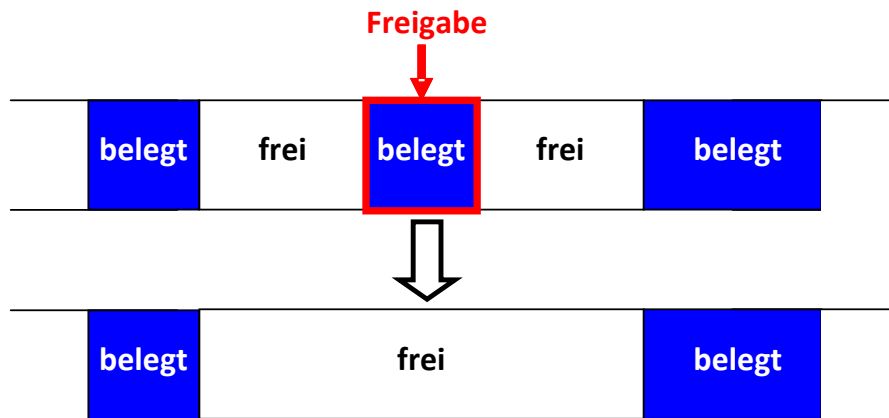
■ Optimierung

- Eventuell mehrere Listen verwenden, um verschiedene Größenordnungen separat zu verketteten
 - Man benötigt dafür nur mehrere Listeneinstiegspunkte
(jeweils einen pro Größenordnung)

■ Detail-Ansicht mit Meta-Daten



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



- Wir legen in hhuTOSr den Heap an die Adresse 3 MB (mit fester Größe)
- 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/>