

Systemprogrammierung

Speicherverwaltung: Zuteilungsverfahren

Wolfgang Schröder-Preikschat, Jürgen Kleinöder

Lehrstuhl Informatik 4

10. Januar 2013

Politiken bei der Speicherverwaltung

Speicherzuteilungsverfahren



Platzierungsstrategie (engl. *placement policy*)

- **wohin** im Arbeitsspeicher ist ein Fragment abzulegen?
 - dorthin, wo der Verschnitt am kleinsten/größten ist?
 - oder ist es egal, weil Verschnitt zweitrangig ist?

Speichervirtualisierung

Ladestrategie (engl. *fetch policy*)

- **wann** ist ein Fragment in den Arbeitsspeicher zu laden?
 - im Moment der Anforderung durch einen Prozess?
 - oder im Voraus, auf Grund von Vorabwissen oder Schätzungen?

Ersetzungsstrategie (engl. *replacement policy*)

- **welches** Fragment ist ggf. aus den Arbeitsspeicher zu verdrängen?
 - das älteste, am seltensten genutzte oder am längsten ungenutzte?

Gliederung

- 1 Überblick
- 2 Platzierungsstrategie
 - Freispeicherorganisation
 - Verfahrensweisen
- 3 Speicherverschnitt
 - Fragmentierung
 - Verschmelzung
 - Kompaktifizierung
- 4 Zusammenfassung

Gliederung

- 1 Überblick
- 2 Platzierungsstrategie
 - Freispeicherorganisation
 - Verfahrensweisen
- 3 Speicherverschnitt
 - Fragmentierung
 - Verschmelzung
 - Kompaktifizierung
- 4 Zusammenfassung

Verwaltung der „Hohlräume“ im Arbeitsspeicher

Bitkarte (engl. *bit map*) von Fragmenten fester Größe

- eignet sich für seitennummerierte Adressräume
- grobkörnige Vergabe auf Seitenrahmenbasis
- alle freien Fragmente sind gleich gut ☺

verkettete Liste (engl. *free list*) von Fragmenten variabler Größe

- ist typisch für segmentierte Adressräume
- feinkörnige Vergabe auf Segmentbasis
- nicht alle freien Fragmente sind gleich gut ☹

Freispeicher erscheint als „Hohlräume“ (auch „Löcher“ genannt) im RAM, die mit Programmtext/-daten auffüllbar sind

- als Bitkarte/verk. Liste implementierte **Löcherliste** (engl. *hole list*)

Freispeicher(bit)karte: Gemeinkosten

Erfassung freier Fragmente beansprucht mehr oder weniger viel Speicher:

- z.B. ein System mit 1 GB Hauptspeicher und 4 KB Seitengröße
- die dazu passende Bitkarte hat eine Größe von 32 KB:

$$\begin{aligned} \text{sizeof}(\text{bit map}) &= 1 \text{ GB} \div 4 \text{ KB} \div 8 \text{ Bits} \\ &= 2^{30} \div 2^{12} \div 2^3 = 2^{15} \text{ Bytes} \end{aligned}$$

- der Bitkartenumfang variiert mit der Seiten(rahmen)größe
 - gleich gr. Speicher \leadsto ungleich gr. indirekte Speicherkosten
- die MMU erlaubt ggf. eine Feinabstimmung (engl. *tuning*)
 - durch einstell- bzw. programmierbare Seiten(rahmen)größen

- je feinkörniger die Speicherzuteilung, desto größer die Bitkarte

Freispeicher(bit)karte

Erfassung freien Speichers fester Größe

Fragmenten des Arbeitsspeichers ist (mind.) ein **Zustand** zugeordnet, der durch einen Bitwert repräsentiert wird: 0 \mapsto **belegt**, 1 \mapsto **frei**

- Suche, Belegung und Freigabe \leadsto Operationen zur Bitverarbeitung

Anforderungen von K Bytes zu erfüllen bedeutet, M Zuteilungseinheiten in der Bitkarte zu suchen, deren Zustand „frei“ anzeigt:

$$M = \frac{K + \text{sizeof}(\text{unit}) - 1}{\text{sizeof}(\text{unit})}$$

- mit *unit* definiert als *char*[N], d.h. allgemein ein Bytefeld darstellend
 - $N = 1$ im Falle segmentierter Adressräume
 - $N = \text{sizeof}(\text{page})$ im Falle seitennummerierter Adressräume

- **Abfall** $M * \text{sizeof}(\text{unit}) - K$ ist nur möglich für $\text{sizeof}(\text{unit}) > 1$

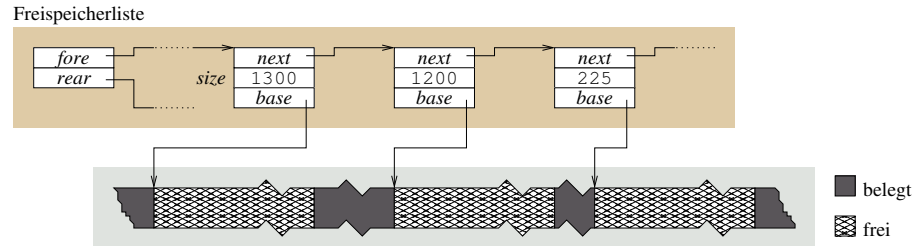
Freispeicherliste

Erfassung freien Speichers variabler Größe

Löcherliste (engl. *hole list*) führt Buch über freie Speicherbereiche

- ein Listenelement hält **Anfangsadresse** und **Länge** eines Bereichs
 - d.h., es erfasst genau ein freies Fragment
- für die Liste ergeben sich **zwei grundlegende Repräsentationen**:
 - 1 Liste und Löcher sind („adressräumlich“) **voneinander getrennt**
 - verteilt über logischen/virtuellen (BS) und physikalischen Adressraum
 - die Listenelemente sind Löcherdeskriptoren, sie belegen Betriebsmittel
 - Löcher haben eine beliebige Größe N , $N > 0$
 - 2 Liste und Löcher sind („adressräumlich“) **miteinander vereint**
 - sie liegen gemeinsam im physikalischen Adressraum
 - die Listenelemente sind die Löcher, sie belegen keine Betriebsmittel
 - Löcher haben eine Mindestgröße N , $N \geq \text{sizeof}(\text{list element})$
- **strategische Überlegungen** bestimmen die Art der Listenverwaltung

Freispeicherliste: Listenelemente als L cherdeskriptoren



- die Liste und der Listenkopf liegen im Betriebssystemadressraum
 - *fore*, *rear* und *next* sind logische/virtuelle Adressen
 - *size* ist die Gr  e des Lochs, Vielfaches von *sizeof(unit)*
 - *base* ist die physikalische Adresse des freien Fragments
- Listenmanipulationen erfolgen innerhalb eines log./virt. Adressraums

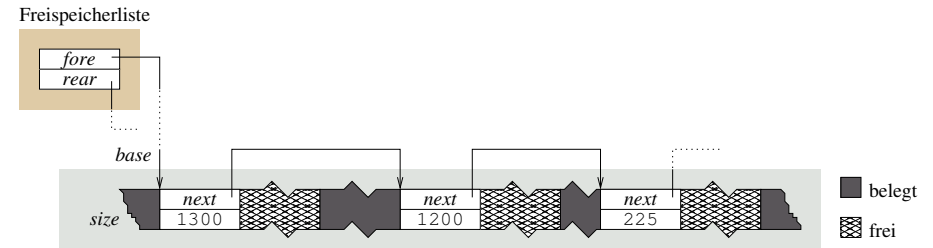
L cher der Gr  e nach sortieren

- best-fit* verwaltet L cher nach aufsteigenden Gr  en
- Ziel ist es, den **Verschnitt** zu **minimieren**
 - d.h., das kleinste passende Loch zu suchen
 - erzeugt kl. L cher am Anfang, erh lt gr. L cher am Ende
 - hinterl sst eher kleine L cher, bei steigendem Suchaufwand
- worst-fit* verwaltet L cher nach absteigenden Gr  en
- Ziel ist es, den **Suchaufwand** zu **minimieren**
 - ist das erste Loch zu klein, sind es alle anderen auch
 - zerst rt gr. L cher am Anfang, macht kl. L cher am Ende
 - hinterl sst eher gro e L cher, bei konstantem Suchaufwand

Ist die angeforderte Gr  e kleiner als das gefundene Loch, f llt Verschnitt an, der als verbleibendes Loch in die Liste neu einsortiert werden muss:

- d.h., **die Freispeicherliste ist ggf. zweimal zu durchlaufen**

Freispeicherliste: Listenelemente als L cher



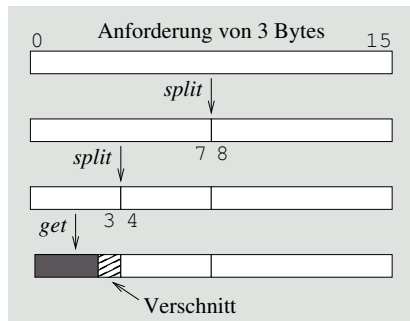
- nur der Listenkopf liegt im Betriebssystemadressraum
 - *fore*, *rear*, *next* und *base* sind physikalische Adressen
 - *size* ist die Gr  e des Lochs, Vielfaches von *sizeof(unit)*
- Listenmanipulationen m ssen ggf. Adressraumgrenzen  berschreiten
 - die Listenoperationen laufen im Betriebssystemadressraum ab
 - der Betriebssystemadressraum ist ein logischer/virtueller Adressraum
 - die Liste ist im physikalischen Adressraum: **Adressraumumschaltung**

L cher der Gr  e nach sortieren: Gr  e = 2er-Potenz

- buddy* (Kamerad, Kumpel) verwaltet L cher nach aufsteigenden Gr  en
- das kleinste passende Loch *buddy_i* der Gr  e 2^i suchen
 - *i*, Index in eine Tabelle von Adressen auf L cher der Gr  e 2^i
 - *buddy_i* entsteht durch sukzessive Splittung von *buddy_j*, $j > i$:
 - $2^n = 2 \times 2^{n-1}$
 - zwei gleichgro e Bl cke, die *Buddy* des jeweils anderen sind
 - der ggf. anfallende Verschnitt kann betr chtlich sein
 - schlimmstenfalls $2^i - 1$ bei $2^i + 1$ angeforderten Einheiten
 - ein Kompromiss zwischen *best-fit* und *worst-fit*
 - vergleichsweise geringer Such- und Aufsplittungsaufwand
 - passt gut zum Laufzeitsystem, das in 2er-Potenzen anfordert

- jeder Rest ist als Summe freier *Buddies* darstellbar, wie auch jede Dezimalzahl als Summe von 2er-Potenzen

Löcher der Größe nach sortieren: *Buddy*-Verfahren



- 1 Block 2^4 teilen: $3 < 2^4/2$
- 2 Block 2^3 teilen: $3 < 2^3/2$
- 3 Block 2^2 vergeben: $3 \geq 2^2/2$
 - Verschneidung von $2^2 - 3 = 1$ Byte

Ob der Verschneidung als interne Fragmentierung zu verbuchen ist, hängt von der MMU ab.

Verschmelzung bei Speicherfreigabe wird zum „Kinderspiel“ (vgl. S. 21)

- zwei freie Blöcke lassen sich verschmelzen, wenn sie *Buddies* sind
 - die Adressen von *buddies* unterscheiden sich nur in einer Bitposition
- zwei Blöcke der Größe 2^i sind genau dann *Buddies*, wenn sich ihre Adressen in Bitposition i unterscheiden

Löcher der Adresse nach sortieren

first-fit verwaltet Löcher nach aufsteigenden Adressen

- Ziel ist es, den **Verwaltungsaufwand** zu **minimieren**
 - invariante Adressen sind das Sortierkriterium
 - die Liste ist bei anfallendem Rest nicht umzusortieren
- erzeugt kl. Löcher vorne, erhält gr. Löcher am Ende
 - hinterlässt eher kl. Löcher bei steigendem Suchaufwand

next-fit reihum (engl. *round-robin*) Variante von *first-fit*

- Ziel ist es, den **Suchaufwand** zu **minimieren**
 - Suche beginnt immer beim zuletzt zugeteiltem Loch
- nähert sich einer Verteilung „gleichgroßer Löcher“
 - als Folge nimmt der Suchaufwand ab

Ist die angeforderte Größe kleiner als das gefundene Loch, fällt Verschneidung an, der jedoch nicht als Restloch in die Liste einsortiert werden muss:

- d.h., die **Freispeicherliste ist nur einmal zu durchlaufen**

Gliederung

- 1 Überblick
- 2 Platzierungsstrategie
 - Freispeicherorganisation
 - Verfahrensweisen
- 3 Speicherverschnitt
 - Fragmentierung
 - Verschmelzung
 - Kompaktifizierung
- 4 Zusammenfassung

Verschneidung durch zuviel zugeteilte/nicht nutzbare Bereiche

Abfall eines zugeteilten Bereichs oder Hohlräume im Arbeitsspeicher

(lat.) Bruchstückbildung

- Zerstückelung des Speichers in immer kleinere, verstreut vorliegende Bruchstücke

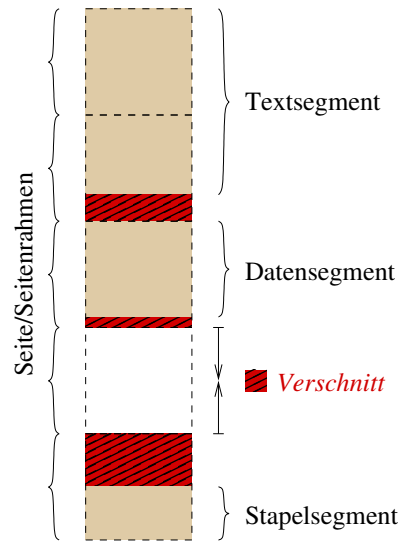
intern bei seitennummerierten Adressräumen \leadsto **Verschwendung**

- Speicher wird in Einheiten gleicher, fester Größe vergeben
 - eine angeforderte Größe muss kein Seitenvielaches sein
 - am Seiten(rahmen)ende kann ein Bruchstück entstehen
- der „lokale Verschneidung“ ist nutzbar, dürfte aber nicht sein

extern bei segmentierten Adressräumen \leadsto **Verlust**

- Speicher wird in Einheiten variabler Größe vergeben
 - eine linear zusammenhängende Bytefolge passender Länge
 - anhaltender Betrieb produziert viele kleine Bruchstücke
- der „globale Verschneidung“ ist ggf. nicht mehr zuteilbar
- **Verschmelzung** und **Kompaktifizierung** schaffen Abhilfe

Interne Fragmentierung



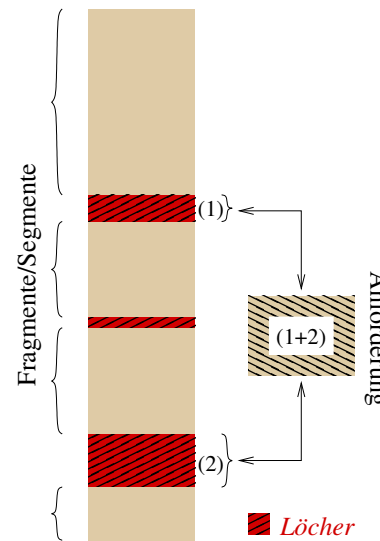
Seitennumerierter Adressraum

- abzubildende Programmsegmente sind Vielfaches von Bytes
- der (log./virt.) Prozessadressraum ist aber ein Vielfaches von Seiten
- die jew. letzte Seite der Segmente ist ggf. nicht komplett belegt

Seitenlokaler Verschnitt

- wird vom Programm *logisch* nicht beansprucht
- ist vom Prozess *physisch* jedoch adressierbar
- da eine seitennummerierte MMU Seiten schützt, keine Segmente

Externe Fragmentierung



Segmentierter Adressraum

- die zu platzierenden Fragmente sind Vielfaches von Bytes
- sie werden 1:1 auf Segmente einer MMU abgebildet
- die jew. eine lineare Bytefolge im phys. Adressraum bedingen

Globaler Verschnitt

- die Summe von Löchern ist groß genug für die Speicheranforderung
- die Löcher liegen aber verstreut im phys. Adressraum vor und
- jedes einzelne Loch ist zu klein für die Speicheranforderung

Vereinigung eines Lochs mit angrenzenden Löchern

Verschmelzung von Löchern erzeugt ein großes Loch, die Maßnahme...

- beschleunigt Speicherzuteilung, **verringert externe Fragmentierung**
- erfolgt bei Speicherfreigabe oder scheiternder Speichervergabe

Löchervereinigung sieht sich mit vier Situationen konfrontiert, je nach dem, welche relative Lage ein Loch im Arbeitsspeicher hat:

- | | |
|----------------------------------|------------------------------|
| ① zw. zwei zugeteilten Bereichen | • keine Vereinigung möglich |
| ② direkt nach einem Loch | • Vereinigung mit Vorgänger |
| ③ direkt vor einem Loch | • Vereinigung mit Nachfolger |
| ④ zwischen zwei Löchern | • Kombination von 2. und 3. |

- der Aufwand variiert z.T. sehr stark mit dem Zuteilungsverfahren

Bezug zum Zuteilungsverfahren

Aufwand ist klein bei *buddy*, mittel bei *first/next-fit*, groß bei *best/worst-fit*

buddy anhand eines Bits der Adresse des zu verschmelzenden Lochs lässt sich leicht feststellen, ob sein *Buddy* bereits als Loch in der Tabelle verzeichnet ist

first/next-fit beim Durchlaufen der Freispeicherliste (bei Freigabe) wird jeder Eintrag daraufhin überprüft, ob...

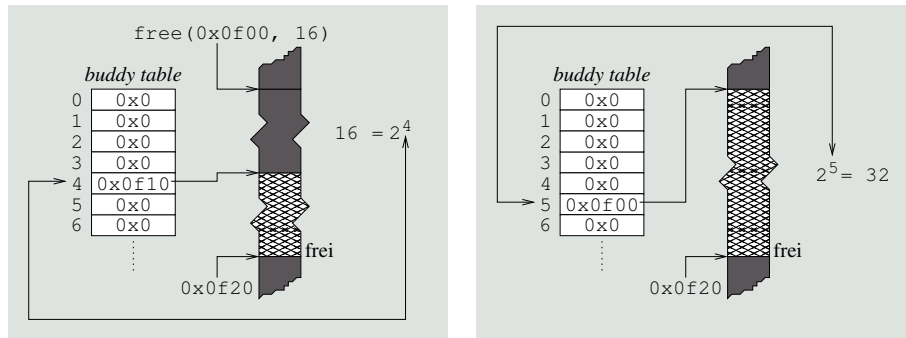
- Adresse plus Größe eines Eintrags gleich der Adresse des zu verschmelzenden Lochs ist (S. 19, 2.)
- Adresse plus Größe eines zu verschmelzenden Lochs der Adresse eines Eintrags entspricht (S. 19, 3.)

best/worst-fit ähnlich wie bei *first/next-fit*, jedoch kann im Gegensatz dazu nicht davon ausgegangen werden, dass bei einem angrenzenden Loch das Vorgänger-/Nachfolgerelement in der Liste das ggf. andere angrenzende Loch sein muss

- es muss weitergesucht werden

Vereinigung von Löchern: *Buddy*-Verfahren

Zwei Blöcke 2^i sind *Buddies*, wenn sich ihre Adressen in Bitposition i unterscheiden



$$0f00_{16} = 0000\ 1111\ 0000\ 0000_2$$

$$0f10_{16} = 0000\ 1111\ 0001\ 0000_2$$

$$16_{10} = 0000\ 0000\ 0001\ 0000_2$$

$$0f00_{16} = 0000\ 1111\ 0000\ 0000_2$$

$$0f20_{16} = 0000\ 1111\ 0010\ 0000_2$$

$$32_{10} = 0000\ 0000\ 0010\ 0000_2$$

Auflösung externer Fragmentierung

Vereinigung des globalen Verschnitts

Segmente von (Bytes oder Seitenrahmen) werden so verschoben, dass am Ende ein einziges großes Loch vorhanden ist

- alle in der Freispeicherliste erfassten Löcher werden sukzessive verschmolzen, so dass schließlich nur noch ein Loch übrigbleibt
- durch **Umlagerung** (engl. *swapping*) kompletter Segmente bzw. Adressräume wird der Kopiervorgang „erleichtert“

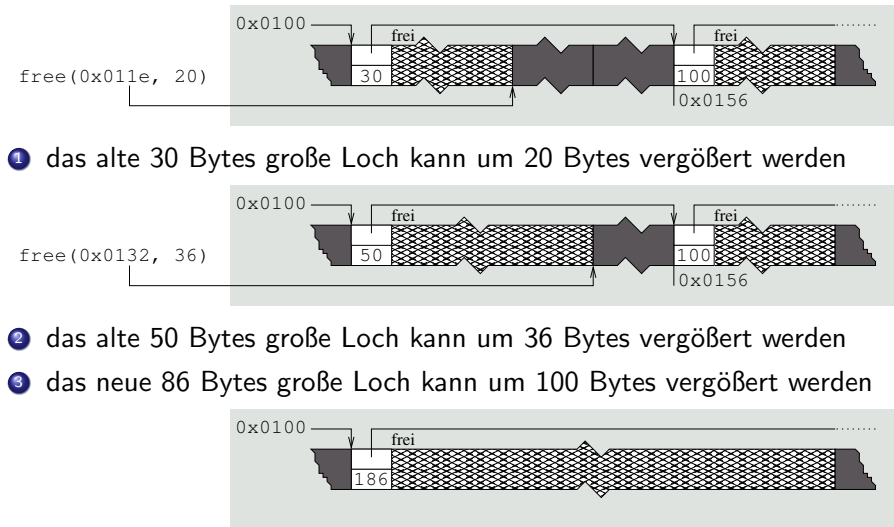
Relokation der verschobenen Segmente/Seiten(rahmen) ist erforderlich

- das Betriebssystem implementiert logische/virtuelle Adressräume oder
- der Übersetzer generiert positionsunabhängigen Programmtext

• je nach Fragmentierungsgrad ein komplexes **Optimierungsproblem**

Vereinigung von Löchern: *first/next-fit*-Verfahren

Freizugebender Block ist Nachfolger und/oder Vorgänger



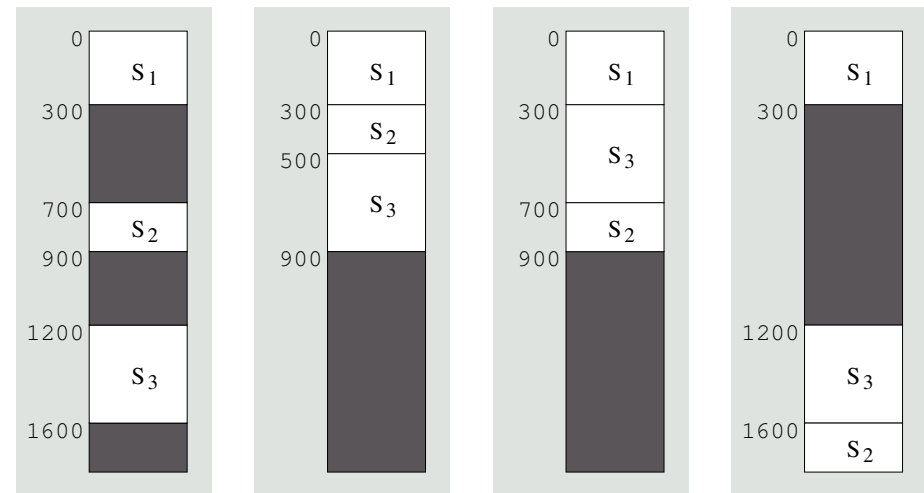
1 das alte 30 Bytes große Loch kann um 20 Bytes vergrößert werden

2 das alte 50 Bytes große Loch kann um 36 Bytes vergrößert werden

3 das neue 86 Bytes große Loch kann um 100 Bytes vergrößert werden

Auflösung externer Fragmentierung: Optionen

Loslegen und Aufwand riskieren oder vorher nachdenken und Aufwand einsparen...



Ausgangspunkt 600 Worte bewegt 400 Worte bewegt 200 Worte bewegt

Gliederung

- 1 Überblick
- 2 Platzierungsstrategie
 - Freispeicherorganisation
 - Verfahrensweisen
- 3 Speicherverschnitt
 - Fragmentierung
 - Verschmelzung
 - Kompaktifizierung
- 4 Zusammenfassung

Resümee

- Zuteilung von Arbeitsspeicher ist Aufgabe der **Platzierungsstrategie**
 - die Erfassung freier Bereiche hängt u.a. ab vom Adressraummodell
 - (a) Seiten bzw. Seitenrahmen \leadsto Bitkarte oder Löcherliste
 - (b) Segmente \leadsto Löcherliste
 - Folge davon ist **interne (a)** oder **externe (b) Fragmentierung**
 - Speicherverschnitt durch zuviel zugeteilte bzw. nicht nutzbare Bereiche
- die **Zuteilungsverfahren** verwalten Löcher nach Größe oder Adresse
 - nach abnehmender Größe *worst-fit*
 - nach ansteigender $\left\{ \begin{array}{ll} \text{Größe} & \text{best-fit, buddy} \\ \text{Adresse} & \text{first-fit, next-fit} \end{array} \right.$
- angefallener **Speicherverschnitt** ist zu reduzieren oder aufzulösen
 - (a) Verschmelzung von Löchern verringert externe Fragmentierung
 - beschleunigt die Speicherzuteilungsverfahren und
 - lässt die Speicherzuteilung im Mittel häufiger gelingen
 - (b) Kompaktifizierung der Löcher löst externe Fragmentierung auf
 - hinterlässt (im Idealfall) ein großes Loch
 - erfordert aber positionsunabhängige Programme oder log. Adressräume