Zuteilungsverfahren (3)

first-fit verwaltet Löcher nach aufsteigenden Adressen

- den *Verwaltungsaufwand minimieren*, d. h., die Liste nicht umsortieren
- erzeugt kleine Löcher am Listenanfang, erhält große Löcher am Listenende
- der Suchaufwand nimmt zu

next-fit die *round-robin* Variante von *first-fit*

- den *Suchaufwand minimieren*, d. h., Start beim zuletzt zugeteiltem Loch
- nähert sich einer Liste/Verteilung von „gleichgroßen Löchern“
- der Suchaufwand nimmt ab

Da Adressen und nicht Größen das Sortierkriterium darstellen, muss die Liste (im Gegensatz zu *best-fit* und *worst-fit*) nur einmal durchlaufen werden.

Verschmelzung

Speicherfreigabe bedeutet nicht nur die Rückgabe eines Betriebsmittels, sondern auch die Vereinigung kleiner Löcher zu einem großen Loch

- für einen zur Freigabe bestimmten Bereich ergeben sich vier Lagen:
  1. zwischen zwei zugeteilten Bereichen
  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 Vereinigungsaufwand variiert mit dem Zuteilungsverfahren
  - klein bei *buddy*, mittel bei *first/next-fit*, groß bei *best/worst-fit*
- Löchervereinigung verringert die *externe Fragmentierung* des Speichers

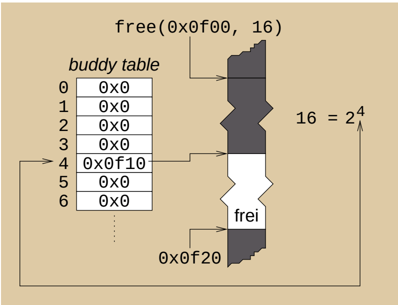
Verschmelzung vs. Zuteilungsverfahren

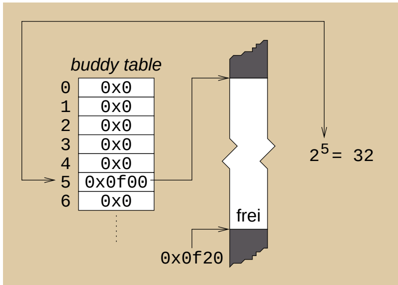
buddy anhand eines Bits der Adresse des freigegebenen Bereichs lässt sich leicht feststellen, ob sein *buddy* als Loch in der Tabelle verzeichnet ist

first/next-fit beim Durchlaufen der Freispeicherliste wird geprüft, ob Adresse plus Größe eines Lochs der Adresse des freigegebenen Bereichs bzw. ob Adresse plus Größe des freigegebenen Bereichs der Adresse eines Lochs entspricht

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- bzw. Nachfolgerelement in der Liste das ggf. andere angrenzende Loch sein muss ☞ es muss weitergesucht werden

Verschmelzung (1) ***buddy***

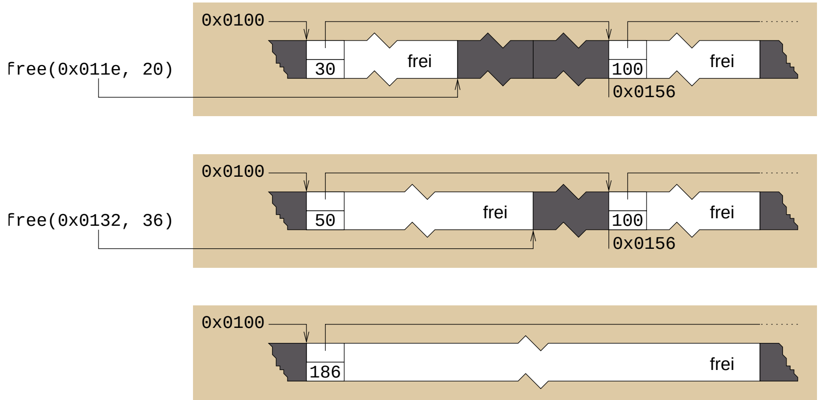




$$\begin{aligned} 0f00_{16} &= 0000\,1111\,0000\,0000_2 \\ 0f10_{16} &= 0000\,1111\,0001\,0000_2 \\ 16_{10} &= 0000\,0000\,0001\,0000_2 \end{aligned}$$

$$\begin{aligned} 0f00_{16} &= 0000\,1111\,0000\,0000_2 \\ 0f20_{16} &= 0000\,1111\,0010\,0000_2 \\ 32_{10} &= 0000\,0000\,0010\,0000_2 \end{aligned}$$

Verschmelzung (2) *first/next-fit*



Fragmentierung

(lat.) *Bruchstückbildung*; Zerstückelung des Speichers in immer kleinere, verstreut vorliegende Bruchstücke:

interne Fragmentierung *Verschwendung*, Problem gekachelten Speichers

- Speicher wird in Einheiten gleicher, fester Größe (☞ Seiten) vergeben
  - eine angeforderte Größe muss kein Vielfaches der Seitengröße sein
  - als Folge kann am Seitenende ein Bruchstück (Verschnitt) entstehen
- der „lokale Verschnitt" kann, dürfte aber nicht genutzt werden [Weshalb?]

externe Fragmentierung *Verlust*, Problem segmentierten Speichers

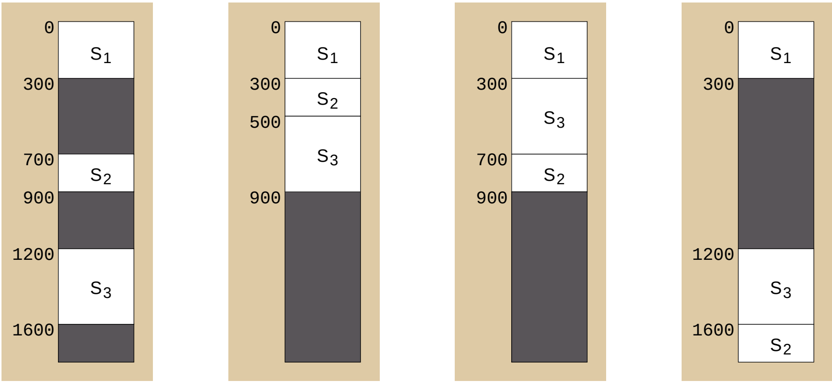
- Speicher wird in Einheiten angeforderter Größe (☞ Segmente) vergeben
- anhaltender Betrieb durchzieht den gesamten Speicher mit Bruchstücken
- der „globale Verschnitt" ist wegen ggf. zu kleiner Bruchstücke unbrauchbar

Kompaktifizierung (1)

- Auflösung externer Fragmentierung durch Vereinigung globalen Verschnitts
  - Segmente von (Bytes oder Seitenrahmen) werden so verschoben, dass am Ende ein einziges großes Loch übrigbleibt
  - alle in der Freispeicherliste erfassten Löcher werden sukzessive verschmolzen, so dass schließlich nur noch ein Listenelement existiert
  - Aus-/Einlagerung (*swapping*) von Segmenten unterstützt den Kopiervorgang
- **Relokation** der verschobenen Segmente ist erforderlich

  ☞ logische/virtuelle Adressräume, positionsunabhängiger Programmtext
- ein je nach Fragmentierungsgrad komplexes *Optimierungsproblem* . . .

Kompaktifizierung (2)



Ausgangspunkt | 600 Worte bewegt | 400 Worte bewegt | 200 Worte bewegt

Ladestrategie

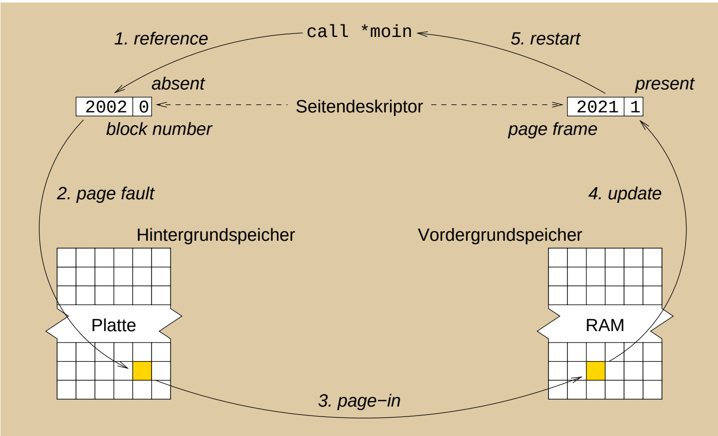
- entscheidet, wann und wie die **Einlagerung** von Seiten/Segmenten erfolgt:

  Einzelanforderung (*on demand*) ☞ *present bit* ✗ 11-15
  - 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
    ☞ Programmlokalität, Arbeitsmenge (*working set*)
  - alternativ als **Vorabruf** (*prefetch*) im Zuge einer Einzelanforderung

- ggf. fällt die *Verdrängung* von Seiten/Segmenten an ☞ Ersetzungsstrategie

Einzelanforderung — „*on demand paging*"



Vorabruf

- *Vorbeugung nachfolgender Seitenfehler* — der „`call *moin`"-GAU (✗ 11-17):
  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 prüfen
  6. wie 4.

- *partielle Interpretation* des unterbrochenen Befehls durch das Betriebssystem

Ersetzungsstrategie

- entscheidet über die **Verdrängung** von <u>Seiten</u> (aber auch Segmenten):

  FIFO (*first-in, first-out*) die zuerst eingelagerten
  - Seiten entsprechend ihres Einlagerungszeitpunkts linear verketten

  LFU (*least frequently used*) die am seltensten genutzten
  - jeden Seitenzugriff zählen, **MFU** (*most frequently used*) als Alternative

  LRU (*least recently used*) die kürzlich am wenigsten genutzten
  - auf „logische Uhrzeiten", Stapeltechniken oder Referenzlisten setzen
  - bzw. weniger aufwändig mit Hilfe von Referenzbits approximieren

- Ursache ist *Speicherüberbelegung*: der Speicherbedarf aller Prozesse ist größer als der verfügbare physikalische Programm-/Datenspeicher (RAM)

Hauptziel von Ersetzungsverfahren

☞ *Seitenfehlerwahrscheinlichkeit* senken und *Seitenfehlerrate* verringern

optimales Verfahren (OPT) Verdrängung der Seite, die am längsten nicht mehr verwendet werden wird

- dies erfordert jedoch Wissen über die im weiteren Verlauf der Ausführung von Prozessen generierten Speicheradressen
  - das Laufzeitverhalten von Prozessen müsste vorhergesagt werden
  - auch mit exaktem Wissen über Eingabewerte ist dies kaum/nicht möglich
    ☞ asynchrone Programmunterbrechungen (✗ 5-29)
- bestenfalls ist eine „gute“ *Approximation* möglich und praktisch umsetzbar

LFU Zählverfahren MFU

- für jede Seite wird mitgezählt, wie häufig sie referenziert worden ist:
  - im Seitendeskriptor ist dazu ein *Referenzzähler* enthalten
  - der Zähler wird mit jeder Referenz zu der Seite inkrementiert
  - aufwändige Implementierung, bei eher schlechter Approximation von OPT

LFU ersetzt die Seite mit dem kleinsten Zählerwert

- „aktive Seiten“ haben üblicherweise große Zählerwerte
- ebenso wie die aktiv gewesenen Seiten z. B. der Initialisierungsphase

MFU ersetzt die Seite mit dem größten Zählerwert

- „kürzlich aktive Seiten“ haben eher kleine Zählerwerte

LRU Grundsätzliche Verfahren

LRU$_{time}$ verwendet einen Zähler („logische Uhr“) in der CPU, der bei jedem Speicherzugriff erhöht und in den jeweiligen Seitendeskriptor geschrieben wird
☞ verdrängt die Seite mit dem kleinsten Zählerwert

LRU$_{stack}$ nutzt einen Stapel eingelagerter Seiten, aus dem bei jedem Seitenzugriff die betreffende Seite herausgezogen und wieder oben drauf gelegt wird
☞ verdrängt die Seite am Stapelboden

LRU$_{ref}$ führt Buch über alle zurückliegenden Seitenreferenzen (*reference string*)
☞ verdrängt die Seite mit dem größten *Rückwärtsabstand*

- entspricht OPT, wenn die Vergangenheit (nicht die Zukunft) betrachtet wird
  - gute Approximation von OPT, bei sehr aufwändiger Implementierung

LRU Approximation (1)

page aging verwendet pro Seitendeskriptor ein **Referenzbit** (*reference bit*), das bei Einlagerung bzw. jedem Seitenzugriff auf 1 gesetzt wird

- das Alter eingelagerter Seiten wird periodisch (☞ Zeitgeber) bestimmt:
  - für jede eingelagerte Seite wird ein N-Bit Zähler (*age counter*) geführt
  - der Zähler funktioniert als *Shiftregister* zur Aufnahme von Referenzbits
  - nach Aufnahme eines Referenzbits in den Zähler, wird es gelöscht
- „kürzlich am wenigsten genutzte“ Seiten haben einen Zählerwert von 0
  - d. h., sie wurden seit N Perioden nicht mehr referenziert
- ein TLB[53] vermeidet die zusätzlichen Speicherreferenzen (*caching*)

[53] *Translation Lookaside Buffer*, eine Tabelle von Kreuzverweisen zwischen virtuellen und physikalischen Adressen kürzlich referenzierter Seiten bzw. Seitenrahmen, inkl. Referenzbit. Der TLB ist Bestandteil des Prozessors.

LRU — Approximation (2)

page aging (Forts.) mit Ablauf eines Zeitquantums (Tick), werden Referenzbits eingelagerter Seiten in die Shiftregister übernommen

- z. B. ein 8-Bit „*age counter*“: $age = (age >> 1) \,|\, (ref << 7)$

Referenzbit	Alter (*age*, initial 0)
1	10000000
1	11000000
0	01100000
1	10110000
⋮	⋮

Den Inhalt des 8-Bit Schiftregisters (*age*) als ganze Zahl intepretiert liefert ein Maß für die Aktivität bzw. Ersetzungspriorität einer Seite: mit abnehmender Wertigkeit der Aktivität steigt die Ersetzungspriorität.

- Aufwand steigt mit der Adressraumgröße des unterbr. Prozesses ☞ UNIX

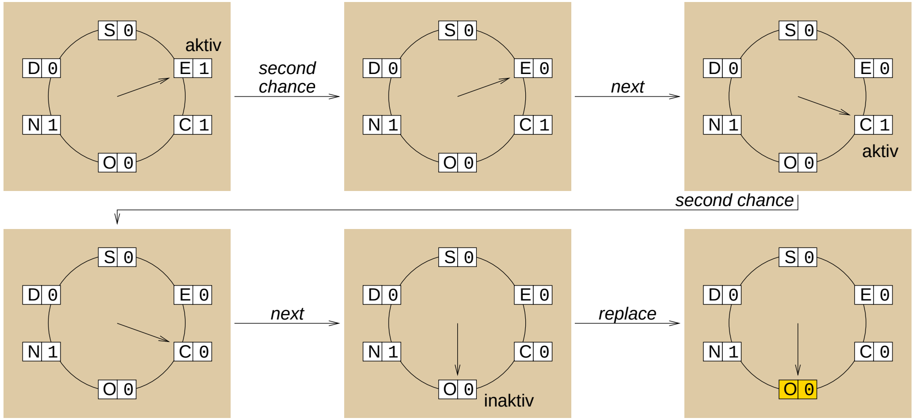
LRU — Approximation (3)

clock policy (*second chance*) ein Ersetzungsverfahren, das im Grunde nach FIFO arbeitet, jedoch zusätzlich noch die Referenzbits der jeweils in Betracht zu ziehenden Seiten berücksichtigt

Referenzbit	Aktion	Bedeutung
1	Referenzbit auf 0 setzen	Seite erhält zweite Chance
0	—	Seite ist Ersetzungskandidat

- im schlimmsten Fall erfolgt ein „Rundumschlag“ über alle Seiten ☞ FIFO
  - falls die Referenzbits aller betrachteten Seiten (auf 1) gesetzt waren
- allgemein gilt: zu ersetzende **beschriebene Seiten** („*dirty*“) auslagern

LRU — *Second Chance*



LRU — Approximation (4)

enhanced second chance prüft, ob eine Seite schreibend referenziert wurde

- Grundlage dafür ist ein **Modifikationsbit** (*modify/dirty bit*) pro Seite
  - wird bei Schreibzugriffen auf 1 gesetzt, behält sonst seinen Zustand bei
- zusammen mit dem Referenzbit ergeben sich vier Paarungen (R, D):

	Bedeutung	Entscheidung
(0, 0)	ungenutzt	beste Wahl
(0, 1)	beschrieben	keine schlechte Wahl
(1, 0)	kürzlich gelesen	keine gute Wahl
(1, 1)	kürzlich beschrieben	schlechteste Wahl

- kann zwei Durchläufe erwirken, Seiten eine weitere Chance geben ☞ MacOS

Freiseitenpuffer

- Alternative zu Ersetzungsstrategien, basiert auf **Vorabruf** (☞ *Ladestrategie*)
  - das System sorgt anhaltend für eine bestimmte Menge freier Seitenrahmen
  - über **Schwellwerte** („*water mark*“) wird die Aus-/Einlagerung gesteuert:
    low Seitenrahmen als frei markieren, Seiten ggf. auslagern
    high Seiten ggf. einlagern, *Vorausladen*
- frei markierte Seiten wandern in einen **Zwischenpuffer** (*cache*) von Freiseiten
  - die Zuordnung von Seiten zu Seitenrahmen bleibt jedoch erhalten
  - vor ihrer Ersetzung doch noch benutzte Seiten werden „zurückgeholt“
  - sogenanntes „*reclaiming*“ von Seiten durch Prozesse ☞ Solaris
- effizient: Ersetzungszeit entspricht weitestgehend der Ladezeit

Seitenanforderung

- wiederverwendbare Betriebsmittel „Seitenrahmen“ (begrenzter Anzahl) sind mehreren Prozessen zuzuordnen
  - Rechnerausstattung und -architektur geben „harte“ Begrenzungen vor:
    Obergrenze festgelegt durch die Größe des Vordergrundspeichers
    Untergrenze definiert durch den „komplexesten“ Maschinenbefehl
  - „weiche“ Begrenzungen: Systemlast, Grad an Mehrprogrammbetrieb
- innerhalb der durch die Hardware gegebenen Grenzen, ist die Zuordnung . . .

  gleichverteilt in Abhängigkeit von der Prozessanzahl und/oder
  größenabhängig bedingt durch den (statischen) Programmumfang

Einzugsbereich bei der 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

☞ Kombination beider Ansätze ist möglich (Prozess-/Adressraumklassen)

Seitenflattern

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

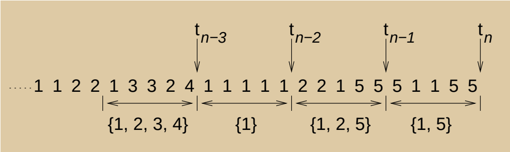
- Ein-/Auslagerung von Seiten bestimmt (phasenweise) die Systemaktivität
  - eben erst ausgelagerte Seiten werden sofort wieder eingelagert
  - Prozesse verbringen mehr Zeit beim „*paging*“ als beim Rechnen
- ein die Systemleistung verringerndes Phänomen globaler Seitenersetzung
  - Prozesse bewegen sich zu nahe am Seiten(rahmen)minimum
  - zu hoher Grad an Mehrprogrammbetrieb
  - ungünstige Ersetzungsstrategie
- Lösungsansätze: „*swapping*“ (✗ 10-4), (lokale/globale) Arbeitsmengen

☞ 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

Approximation einer Arbeitsmenge

- Referenzbit, Seitenalter und periodische Unterbrechungen als Grundlage: bei Ablauf eines Zeitquantums die Referenzbits eingelagerter Seiten prüfen

	Aktion
1	Referenzbit und Alter auf 0 setzen
0	Alter der Seite um 1 erhöhen

Seiten von Arbeitsmengen haben Alterswerte kleiner als die Fensterbreite

- beim lokalen Ansatz altern nur Seiten des jeweils unterbrochenen Prozesses

  ☞ Problem: gemeinsam genutzte Seiten (z. B. *shared libraries*)

- Alternative ist der globale Ansatz, der jedoch mehr Seiten zu untersuchen hat

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 benötigte 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