Freispeicherverwaltung

Allgemeine Techniken und Anwendung unter Linux

Überblick

- Allgemeines
- Suchstrategien
- Verwaltungsstrategien
- externer / interner Verschnitt
- Buddy System Algorithmus
- Slab Allocator
- Caches
- Slab Coloring

Unterteilung des Speichers

- virtueller Speicher:
  - nur beschränkt durch Anzahl der Bits der Adressen
  - virtuelle Adressen müssen auf physikalische Adressen abgebildet werden

- physikalischer Speicher:
  - Bereich für Kernel
  - Bereich für dynamischen Speicher

Verwaltung des dynamischen Speichers entscheidet über die Geschwindigkeit des Gesamtsystems.

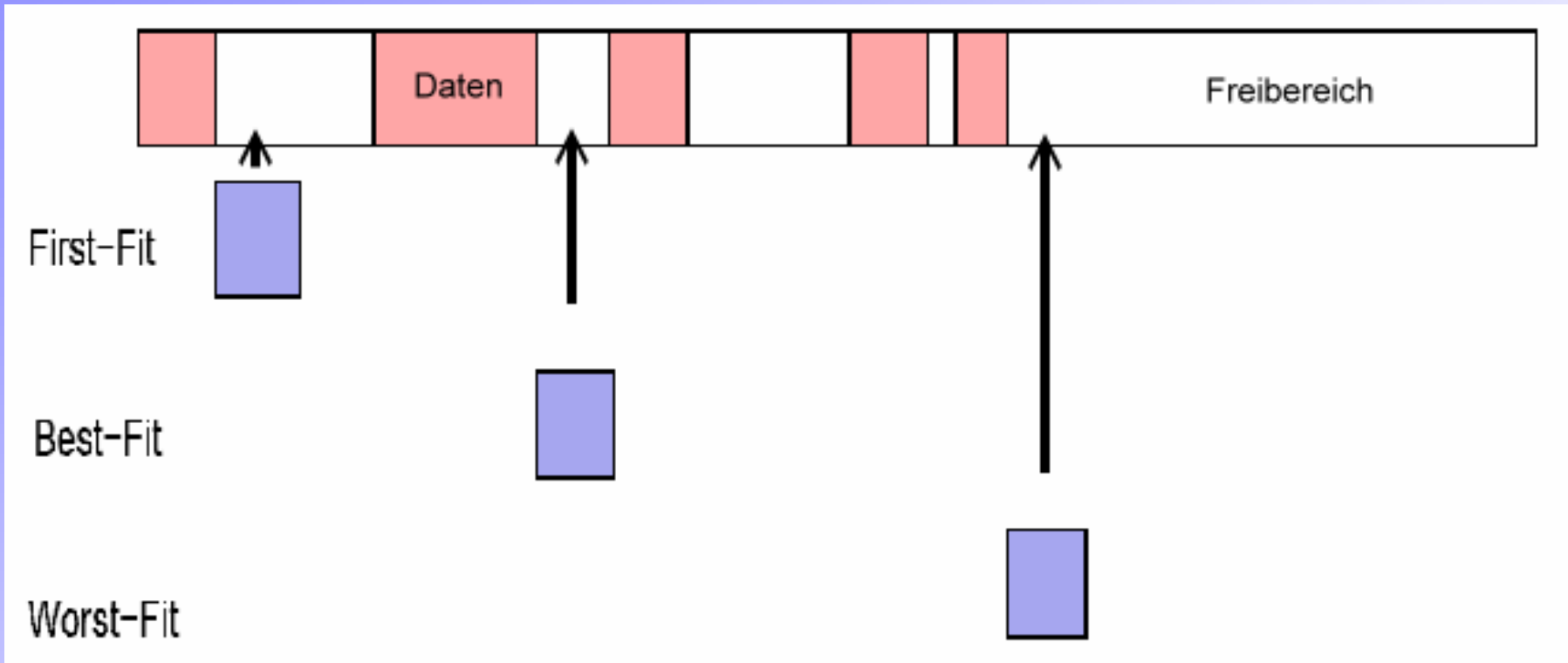
Suchstrategien für passenden Freibereich

Mehrere Methoden:

- First-Fit: Suche nach ersten passenden Freibereich
- Best-Fit: Suche nach kleinsten passenden Freibereich
- Worst-Fit: Suche nach größten passenden Freibereich
- weitere Strategien

Suchstrategien für passenden Freibereich

Beispiel:



Verschnitt

Unterscheidung:

- externer Verschnitt
  Prozesse werden aufgeteilt und über den Speicher verteilt
- interner Verschnitt
  Speicher wird in Blöcke aufgeteilt, im Block bleibt meist Speicher ungenutzt

Verwaltungsstrategien

- Bit Maps:

  Seitenkacheln im Speicher werden durch Bits in einer Tabelle repräsentiert:

  Bit = 0: frei
  Bit = 1: belegt

  Vorteile:

  - einfache Handhabung
  - Bit Map-Größe statisch

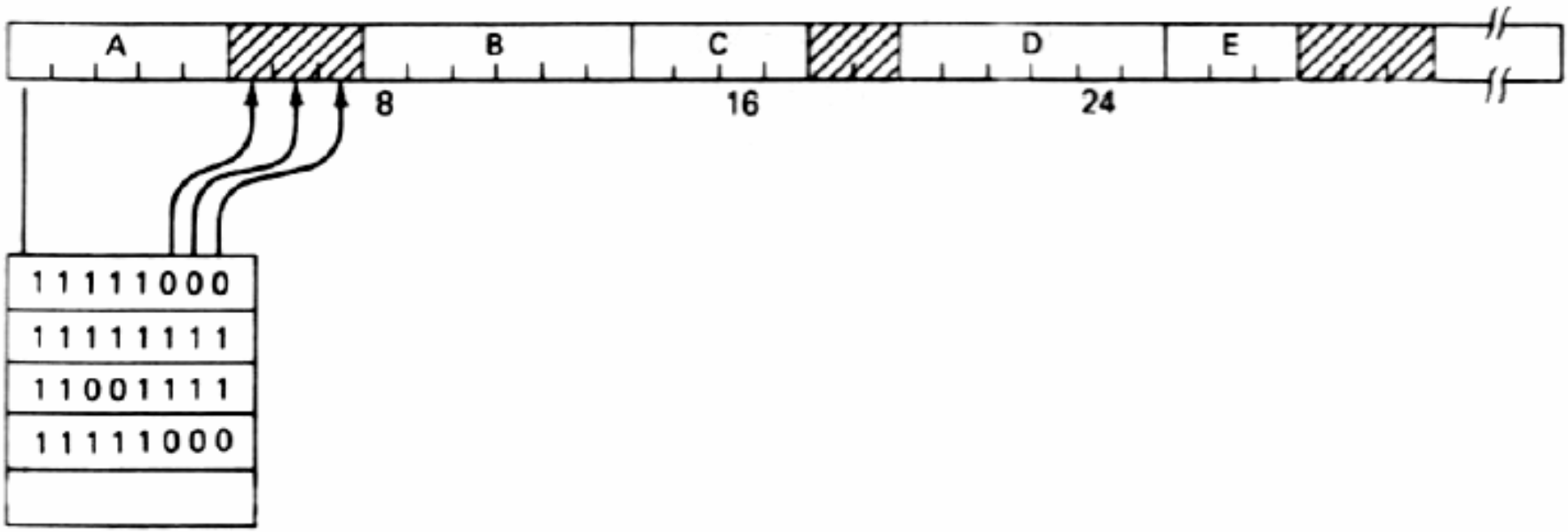
  Nachteile:

  - Bit Map muss bei Freispeichersuche nach zusammenhängenden Nullen durchsucht werden
  - schlechtester Fall: komplette Bit Map muss durchlaufen werden

  => ineffizient

Bit Maps

Beispiel:



Verwaltungsstrategien

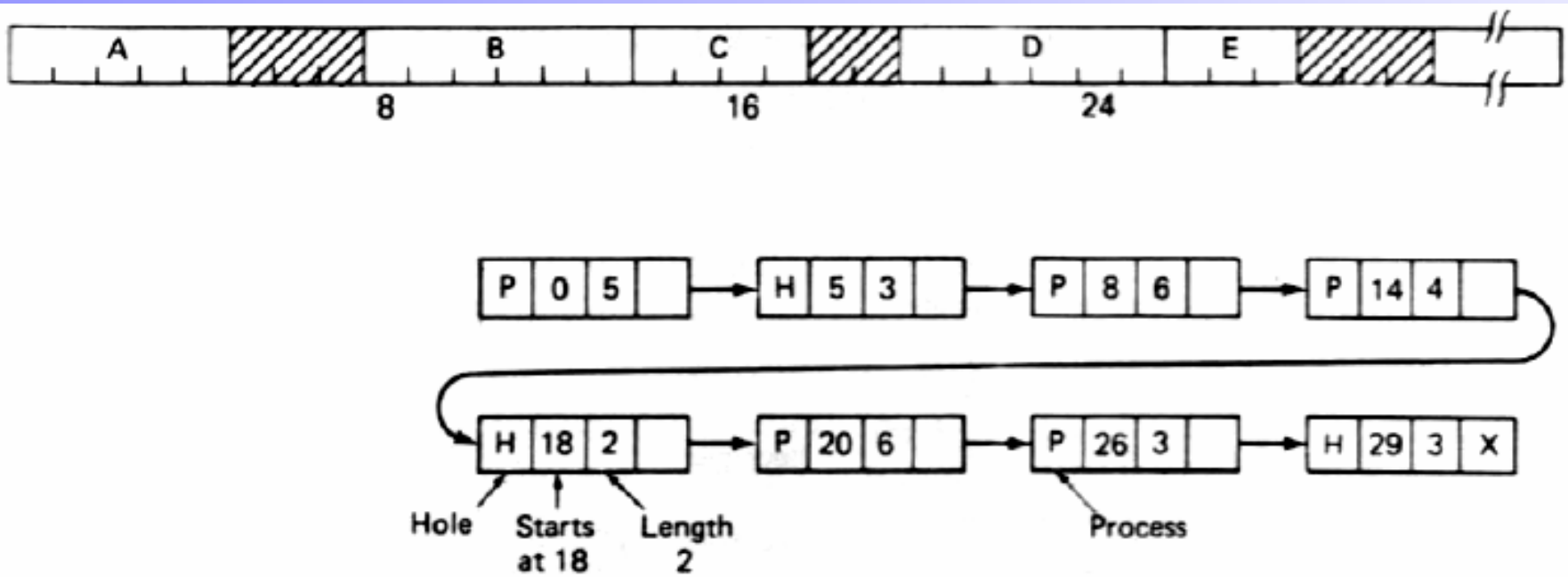
- Linked Lists:

  - Verwaltung des Speichers in verketteten Listen
  - Jedes Segment ist entweder von einem Prozess belegt oder verweist auf freien Speicher
  - Sortierung meist nach Reihenfolge der Adressen
  - Integrierung neuer Prozesse durch First/Next/Best/Worst – Fit
  - schnelle Zusammenfassung freien Speichers
  - häufige Verwendung von „double linked lists“

=> meist effizienter als Bit Maps

Linked Lists

Beispiel:



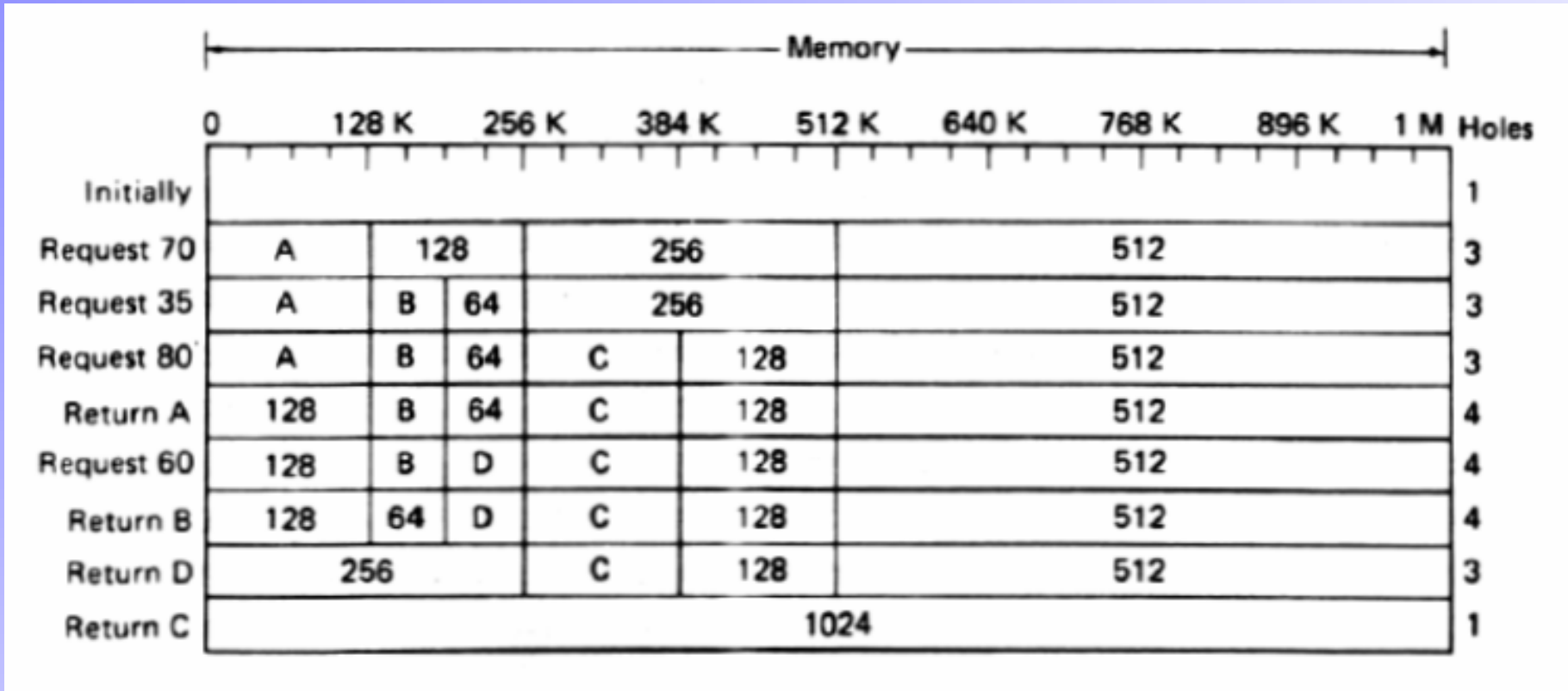
Verwaltungsstrategien

- Buddy System Algorithmus:

  - Speicher wird in unterschiedlich große Bereiche aufgeteilt, Bereichsgröße entspricht Vielfachem von 2
  - Bereiche werden in Listen verwaltet
  - jede Bereichsgröße in separater Liste
  - erste Liste enthält gesamten Speicherbereich
  - unterste Liste enthält meist 4 KByte – Bereiche (theoretisch bis zu 1 Byte)
  - am Anfang alle Listen leer, bis auf erste

Buddy System Algorithmus

Beispiel:



Buddy System Algorithmus

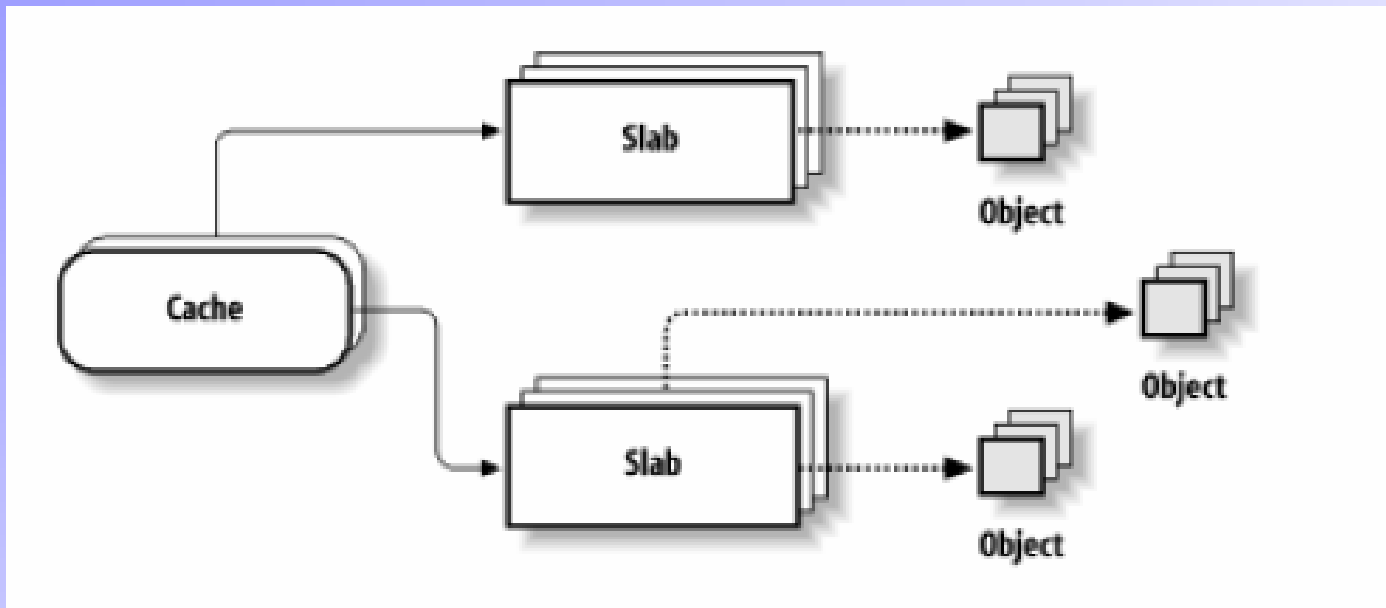
- Vorteil:
  schnelle Zusammenfassung von freien Speicherblöcken

- Nachteile:
  - ineffizient bei kleinen Datenmengen
  - interner Verschnitt

=> Weitere Techniken zur Effizienzsteigerung

Slab Allocator

Speicher als Objekt mit Methoden und Funktionen



Slab Allocator

- speichert häufig benötigte Speicherkacheln, um Kernel zu entlasten
- unterscheidet nach Häufigkeit der Anfragen für eine bestimmte Speichergröße
- erzeugt schon Objekte einer Speichergröße, bevor sie angefordert werden, wenn die Erwartung der Anfrage hoch ist
- vermeidet dadurch Aufrufe des Buddy System Algorithmus

Caches

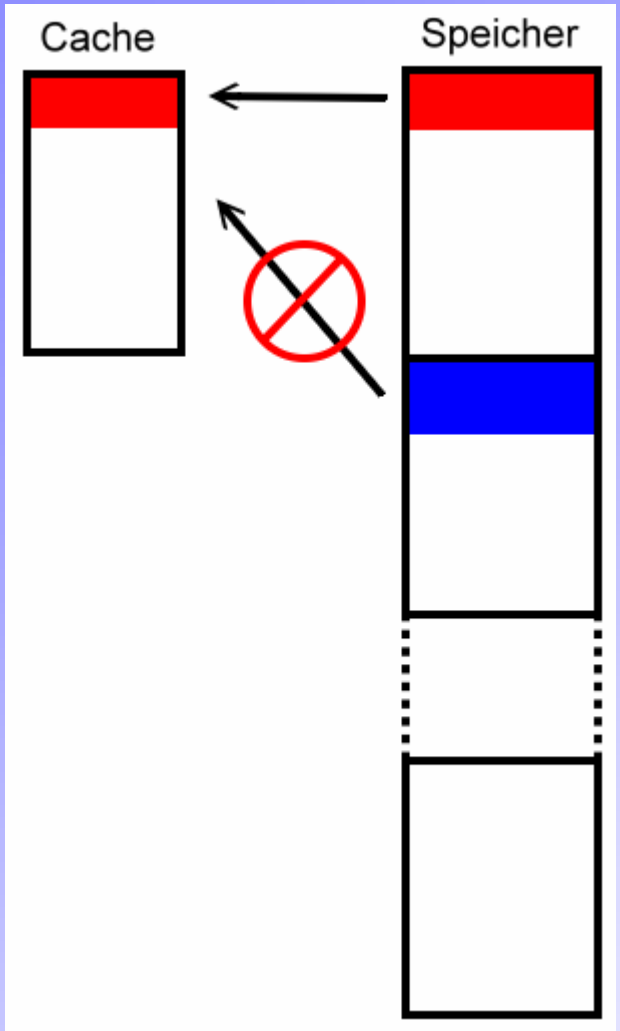
Zwischenspeicherung von Prozessen => schnellerer Zugriff

Unterscheidung:

- generelle Caches:
  vom Slab Allocator benutzt, erster Cache enthält Beschreibung der restlichen Caches
- spezifische Caches:
  vom Rest des Kernels benutzt

Ein neu initialisierter Cache enthält noch keine Slabs / Objekte, sie werden erst erzeugt, wenn sie benötigt werden

Slab Coloring



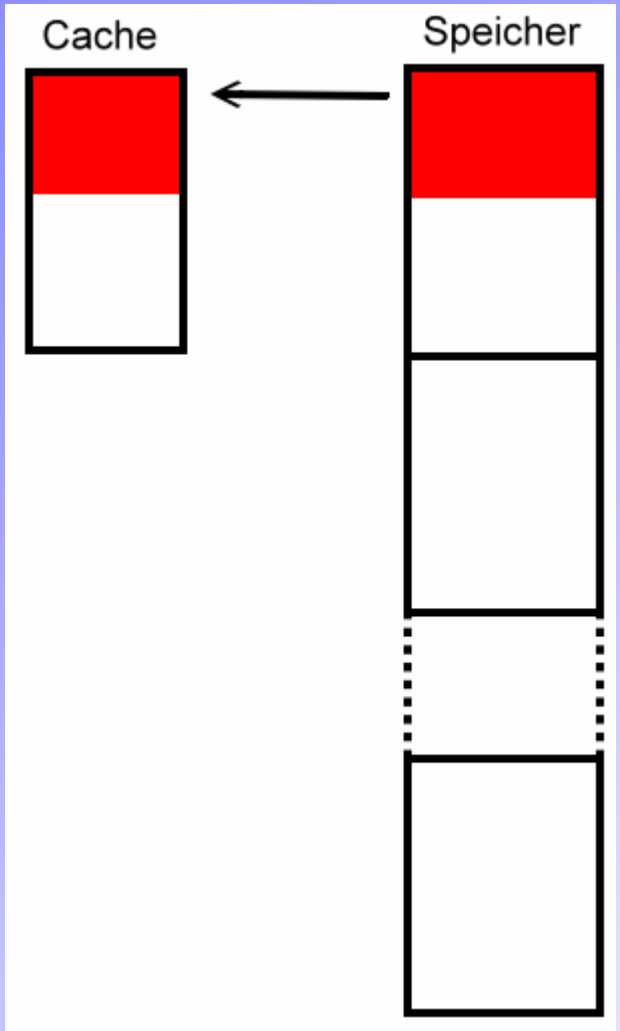
Cache-großer Bereich wird in den Cache geladen.

Problem:

Wenn Prozess verteilt im Speicher liegt und an gleicher Position innerhalb der Slabs, muss abwechselnd geladen werden.

=> ineffizient

Slab Coloring



Lösung: Slab Coloring

Verwendung von „Farben“, um Slabs zu kennzeichnen, dass schon beim Laden der Prozesse in den Speicher Prozesse zusammenhängend gespeichert werden.

„Einfärben“ durch unterscheidliche Verschiebung freien Speichers vom Ende zum Anfang des Slabs; nur bei genügend freiem Speicher innerhalb der Slabs möglich.