Der Slab Allocator

Vortrag im Rahmen des Proseminars
“Linux Kernel Internals”

Uni Karlsruhe

WS 2004/2005

Autor: Sven Krohlas (sven@asbest-online.de)

http://www.krohlas.de

Motivation: um was geht es?

- auch der Kernel benötigt Speicher
- Buddy-System ist jedoch ineffizient
  - Anforderung seitenweise
  - schlechte Nutzung der CPU-Caches
- Anforderungen an die Speicherverwaltung
  - effiziente Speicherausnutzung
  - schnelle Anforderung/Freigabe
  - wenig Fragmentierung

Der Slab Allocator

1) Motivation
2) Geschichte
3) Idee
4) Caches
5) Slabs
6) Beschränkungen
7) Quellen & Literatur

Der Slab Allocator

1) Motivation
2) **Geschichte**
3) Idee
4) Caches
5) Slabs
6) Beschränkungen
7) Quellen & Literatur

Geschichte

- 1994: von Jeff Bonwick für SunOS 5.4 entwickelt
  - öffentlich dokumentiert [Bon94]
  - deutlich verbesserte Effizienz
- 1999: im Linux-Kernel seit Version 2.2
- 2001: verbesserte Version von Bonwick
  - ebenso öffentlich dokumentiert [Bon01]

Übersicht

1) Motivation
2) Geschichte
3) Idee
4) Caches
5) Slabs
6) Beschränkungen
7) Quellen & Literatur

Idee: Eigenschaften von Kernobjekten

- meist klein (wenige Bytes)
- werden oft angefordert/frei gegeben
- werden mit Werten initialisiert

Idee: private Caches?

- Spannungen: private Cache ↔ restliches System
- System kann keinen Speicher zurückholen
- schlechter Überblick über Speichersituation
- kein Überblick über Bedarf anderer
- schweres Debugging
- kein Accounting
- kein Code-Sharing

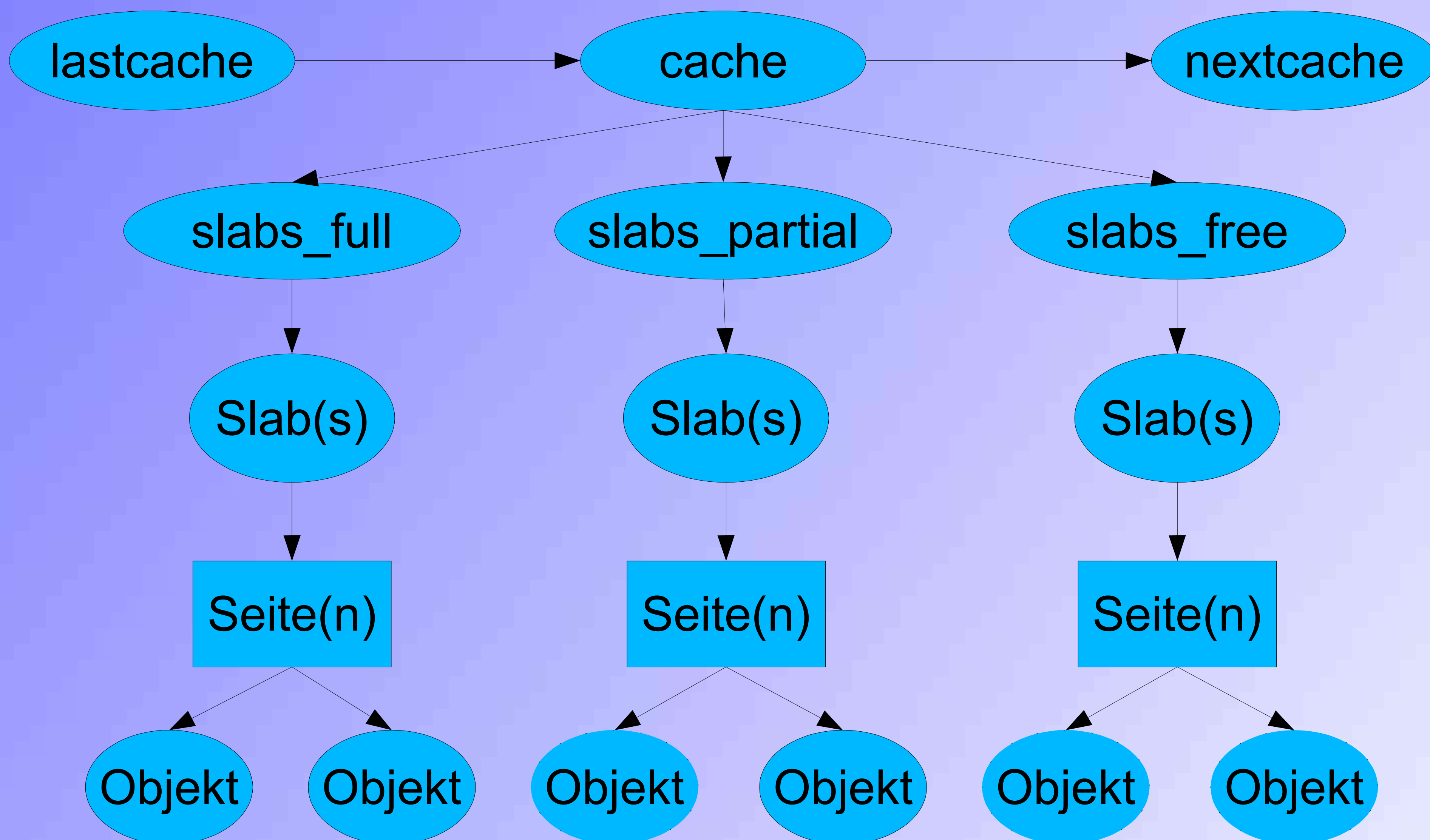
Idee: Clients & Central Allocator

- Central Allocator
  - verwaltet Speicher
  - sorgt für möglichst effiziente Nutzung
- Clients
  - beschreiben gewünschte Objekte
  - müssen sich nicht um Details kümmern

Idee: Aufbau (1)

- Objekte eines Typs werden zu “Slabs” zusammengefasst
- Slabs werden unter einem Cache organisiert
- Caching vermindert Rückgriffe auf das Buddy-System
- Speicherseiten werden vom Buddy-System geholt

Idee: Aufbau (2)



basierend auf [Gor04]

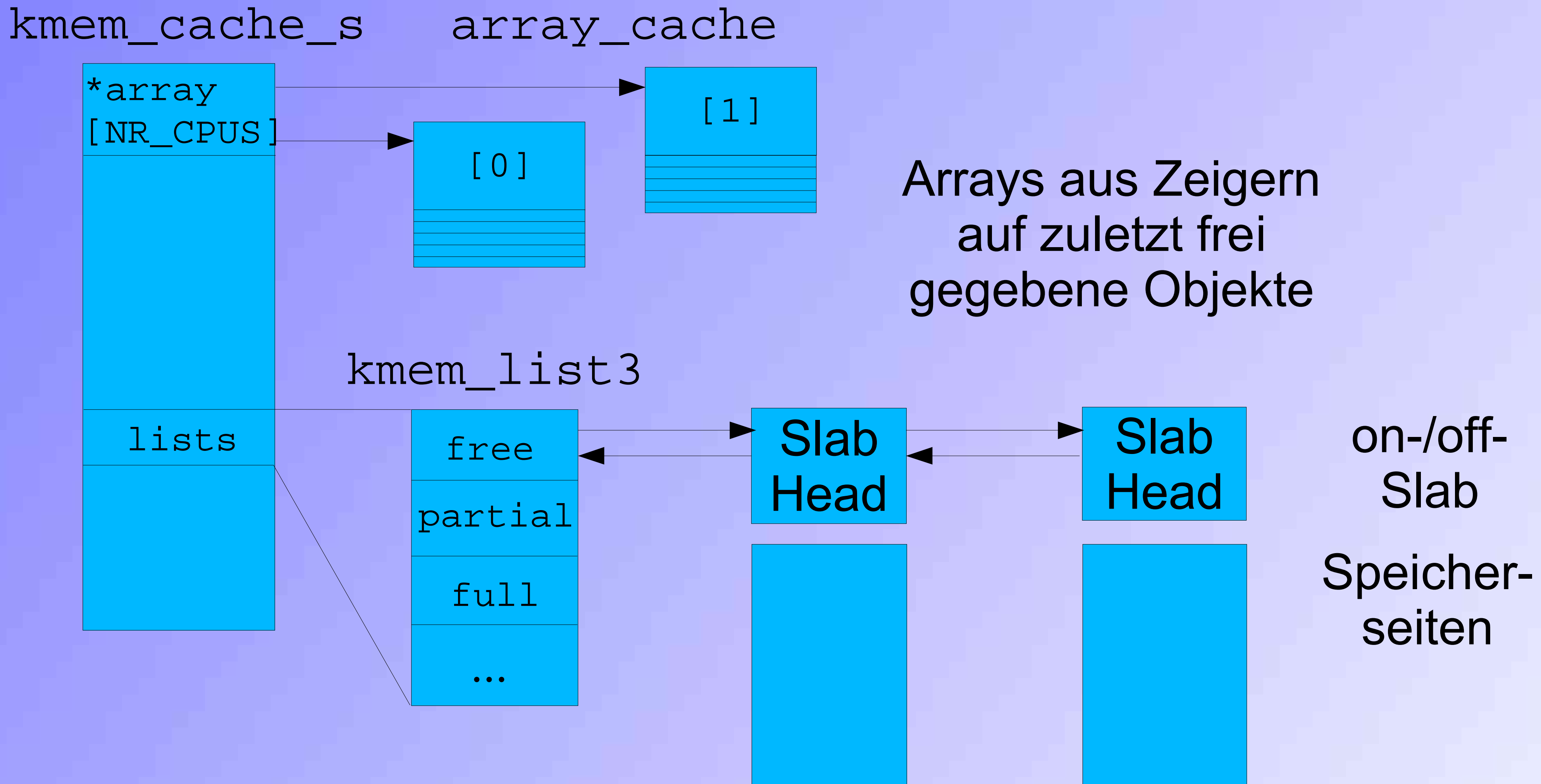
Übersicht

1) Motivation
2) Geschichte
3) Idee
4) Caches
5) Slabs
6) Beschränkungen
7) Quellen & Literatur

Cache: Aufgaben

- 3 Aufgaben
  - Caches für oft gebrauchte Objekte (fertig initialisiert)
  - Allgemeine Caches bestimmter Größen (von 2^5=32 bis 2^17=131072 Bytes)
  - Hardware-Caches möglichst gut ausnutzen

Cache: Aufbau



basierend auf [Mau04]

Cache: Informationen (1)

```
sven@linux:~> cat /proc/slabinfo
slabinfo - version: 2.0
# name <active_objs> <num_objs> <objsize> <objperslab>
<pagesperslab> : tunables <batchcount>
uhci_urb_priv          1    88    44   88    1 : tunables  120
fib6_nodes             5   119    32  119    1 : tunables  120
ip6_dst_cache          4    15   256   15    1 : tunables  120
ndisc_cache            1    20   192   20    1 : tunables  120
raw6_sock              0     0   640    6    1 : tunables   54
udp6_sock              0     0   640    6    1 : tunables   54
tcp6_sock              5     7  1152    7    2 : tunables   24

size-512(DMA)          0     0   512    8    1 : tunables   54
size-512             639   672   512    8    1 : tunables   54
size-256(DMA)          0     0   256   15    1 : tunables  120
size-256             224   495   256   15    1 : tunables  120
```

[gekürzte Ausgabe]

Cache: Informationen (2)

- active_objs: Anzahl der belegten Objekte
- num_objs: Anzahl der Objekte (belegt und unbelegt)
- objsize: Größe der Objekte
- objperslab: Objekte pro Slab
- pagesperslab: Seiten pro Slab
- batchcount: Zahl der Objekte, die auf einmal angefordert/frei gegeben werden (½ der Objekte im per-CPU-Cache)

Caches verwalten: APIs (1)

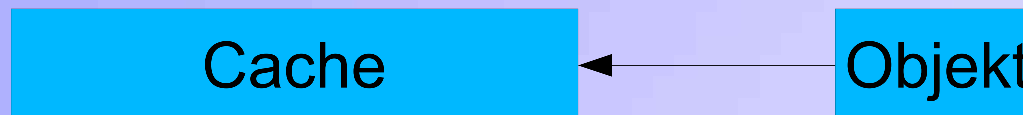
- Cache anlegen → Cache
  - **kmem_cache_create**(const char ***name**, size_t **size**, size_t **offset**, unsigned long **flags**, void (***ctor**)(void*, kmem_cache_t *, unsigned long), void (***dtor**)(void*, kmem_cache_t *, unsigned long))
- Objekt aus den Cache anfordern
  - **kmem_cache_alloc**(kmem_cache_t ***cachep**, int **flags**)

Caches verwalten: APIs (2)

- `*name`: Name des Caches
- `size`: Größe der Objekte in Bytes
- `offset`: Wunschoffset für Colouring (oft 0)
- `flags`: Einstellungen für Debugging, DMA...
- `*ctor`: Konstruktorfunktion
- `*dtor`: Destruktorfunktion
- `*cachep`: Zeiger auf `kmem_cache_t`-Instanz des Caches

Caches verwalten: APIs (3)

- Objekt an den Cache zurückgeben
  - `kmem_cache_free(kmem_cache_t *cachep, void *objp)`
  - `page`-Instanz der Seite des Objekts wird zweckentfremdet
    - `page->list.next` zeigt auf die Verwaltungsstruktur des Caches
    - `page->list.prev` auf die des Slabs
    - Versteckt hinter den Makros: `SET_PAGE_SLAB`, `GET_PAGE_SLAB` bzw. `_CACHE`



Caches verwalten: APIs (4)

- Alle Slabs des Caches frei geben und Cache zerstören
  - **kmem_cache_destroy(**kmem_cache_t ***cachep)**
- weitere siehe [Gor04]

Cache: APIs für Size-Caches

- Speicher reservieren
  - **kmalloc**(size_t **size**, int **flags**)
- Speicherbereich freigeben
  - **kfree**(const void ***ptr**)
  - "Hack" wie bei kmem_cache_free()
- vgl.: malloc()/free() aus der C-Bibliothek

Cache: interne Fragmentierung

- [Bon94]: “per buffers wasted space”
  - Lücken entstehen durch
    - Aufrunden der Objektgröße auf das nächste Vielfache von `sizeof (void *)`
    - Verschnitt
  - Verschnitt < $1/n$ der Größe von n Objekte
  - Nutzung des Verschnitts für Colouring

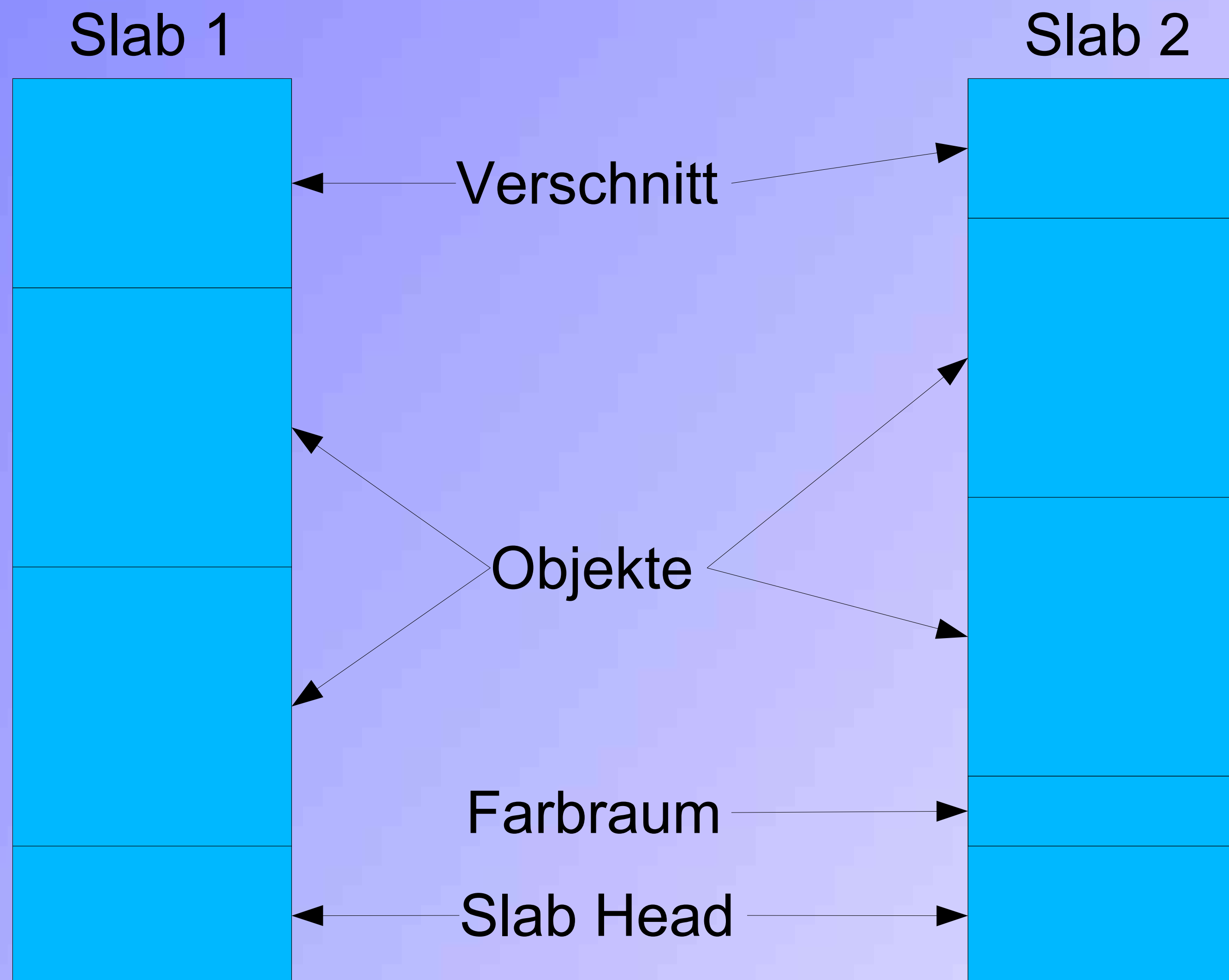
Cache: externe Fragmentierung

- [Bon94]: “unused buffers in the freelist”
  - oft werden nicht alle Objekte genutzt
- ist gering
  - gleiche Objekte → ähnliche Lebensdauer
  - Objekte werden nicht weiter aufgeteilt
- mehr Objekte → größere ext. Fragmentierung
  - Wahrscheinlichkeit einen Slab leer zu kriegen sinkt
- Ausnahme: Size-Caches
  - seltener benötigt

Cache: Colouring (1)

- Idee: Bessere Ausnutzung der CPU-Caches durch unterschiedliche Ausrichtungen der Objekte in verschiedenen Slabs.
  - Ausrichtung an Vielfachen der Cachelinegröße
  - Ermittelt wird
    - mögliche Ausrichtung (Offset: `colour_off`)
    - Anzahl der Farben (ganzzahlige Vielfache des Offsets: `colour`)
  - Falls Objektgröße < ½ Offset: 2 (4/8...) Objekte pro Cachline

Cache: Colouring (2)



Cache: Beispiel

- Cache für Objekte von 256 Bytes bei 4KB-Pages, Ausrichtung auf 32 Bytes L1-Cachline
  - Slab-Kopf: on-slab (da Objekte < ⅛ Pagesize)
  - 15 Objekte
  - 1 Speicherseite
  - 5 Farben, Offset: 32 Bytes

Caches: Speicheranforderung

- Reihenfolge, in der versucht wird Objekte anzufordern
  - Aus CPU-spezifischer Liste (`array_cache`)
  - Aus bestehendem Slab
  - Aus frisch angelegtem Slab (über das Buddy-System allokiert)
- Aufwand und negativer Einfluss auf Caches/TLBs steigt von Punkt zu Punkt

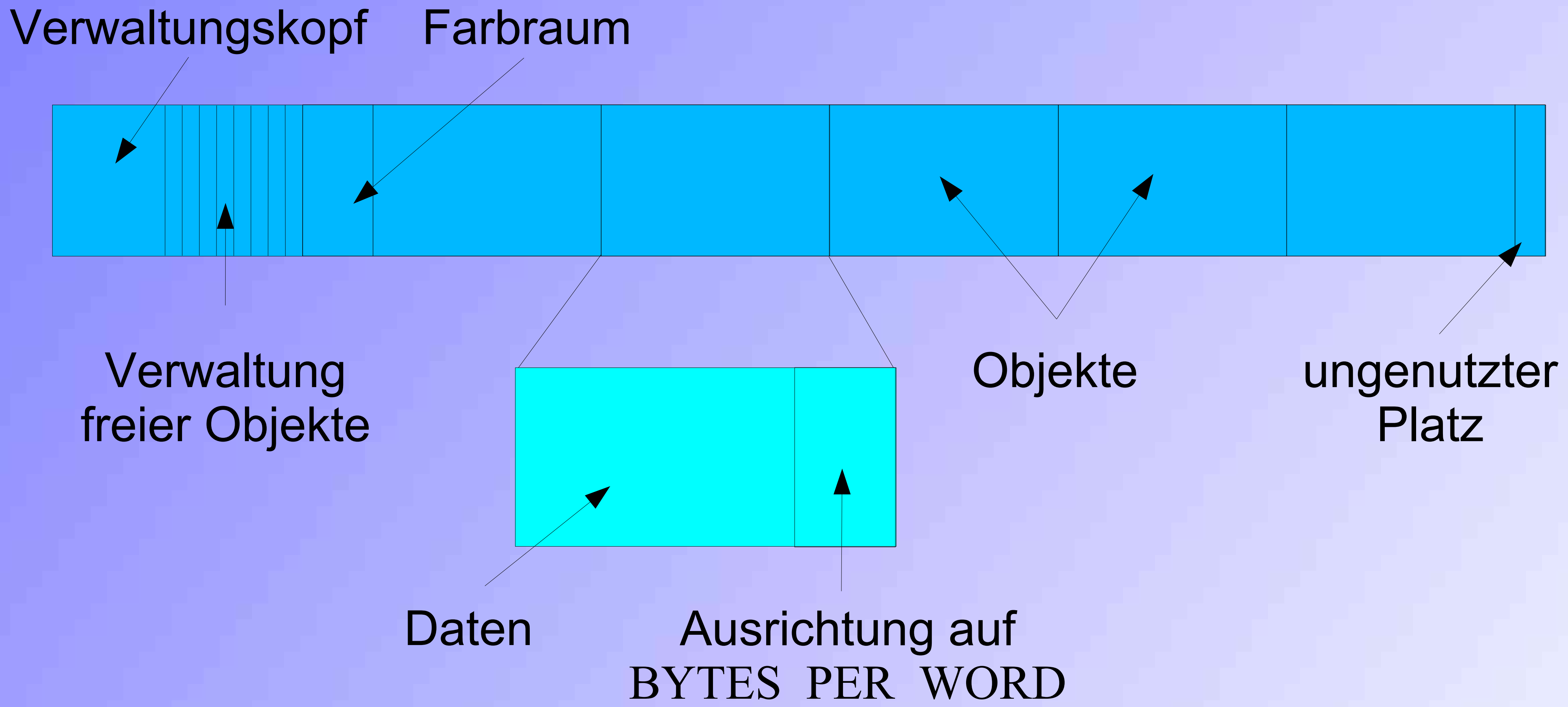
Übersicht

1) Motivation
2) Geschichte
3) Idee
4) Caches
5) Slabs
6) Beschränkungen
7) Quellen & Literatur

Slabs: Aufbau (1)

- Slab Head
  - Management-Struktur
  - Informationen welche Objekte frei/belegt sind
- Farbraum (Farbe*Offset)
- Objekte
  - Mit Füllbytes aufgerundet auf Vielfache von `sizeof (void *)`
  - schnellere Zugriffe dank ausgerichteter Adressen
- restlicher Verschnitt

Slabs: Aufbau (2)



basierend auf [Mau04]

Slabs: Verwaltungskopf

- enthält
  - `struct list_head list`: Verknüpfung mit nächstem & vorherigen Slab der full/partial/free-Liste
  - `unsigned long colouroff`: Offset des Slabs
  - `void *s_mem`: Zeiger auf das erste Objekt
  - `unsigned int inuse`: Zahl der belegten Objekte
  - `kmem_bufctl_t free`: Index des ersten freien Objekts

Slabs: Verwaltung freier Objekte

- Eine Zahl pro freiem Objekt
  - gibt Index des nächsten freien Objekts an
  - letzter Eintrag: `BUFCTL_END`

basierend auf [Mau04]

Übersicht

1) Motivation
2) Geschichte
3) Idee
4) Caches
5) Slabs
6) Beschränkungen
7) Quellen & Literatur

Beschränkungen

- Skaliert schlecht auf mehrere CPUs
  - globaler Lock
- nur für Kernspeicher (z.B. Perl verwendet eigene Implementierung)
- Lösung: [Bon01]
  - bietet per-CPU Speicheranforderung
  - verfügbar als user-level Library
  - "...first resource allocator that can satisfy arbitrary-size allocations [...] in guaranteed constant time."

Übersicht

1) Motivation
2) Geschichte
3) Idee
4) Caches
5) Slabs
6) Beschränkungen
7) Quellen & Literatur

Quellen & Literatur (1)

- [Bau03]: Baumgartl, Robert: *Speicherverwaltung kleiner Bereiche: Der Slab Allocator*. TU Chemnitz, 2003.
  - http://osg.informatik.tu-chemnitz.de/mitarb/robge/talks/slaballocator.pdf
- [Bon94]: Bonwick, Jeff: *The Slab-Allocator: An Object-Caching Kernel Memory Allocator*. Usenix proceedings, 1994.
  - http://www.usenix.org/publications/library/proceedings/bos94/full_papers/bonwick.ps

Quellen & Literatur (2)

- [Bon01]: Bonwick, Jeff: *Extending the Slab Allocator to Many CPUs and Arbitrary Resources.* Usenix, 2001.
  - http://www.usenix.org/event/usenix01/full_papers/bonwick/bonwick.pdf
- [Gor04]: Gorman, Mel: *Understanding The Linux Virtual Memory Manager*. 2004.
  - http://www.skynet.ie/~mel/projects/vm/guide/html/understand/understand-html.html
- [Mau04]: Mauerer, Wolfgang: *Linux Kernelarchitektur*. Hanser, 2004.

Quellen & Literatur (3)

- [Tan01]: Tanenbaum, Andrew S.: *Modern Operating Systems.* Prentice Hall, 2001.
- Linux 2.6.10-rc2; mm/slab.c
  - http://www.kernel.org/

Fragen?