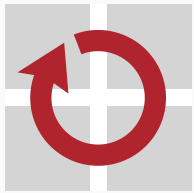
Distributed Memory Management

15. Januar 2020

Florian Fischer

Friedrich-Alexander-Universität Erlangen-Nürnberg



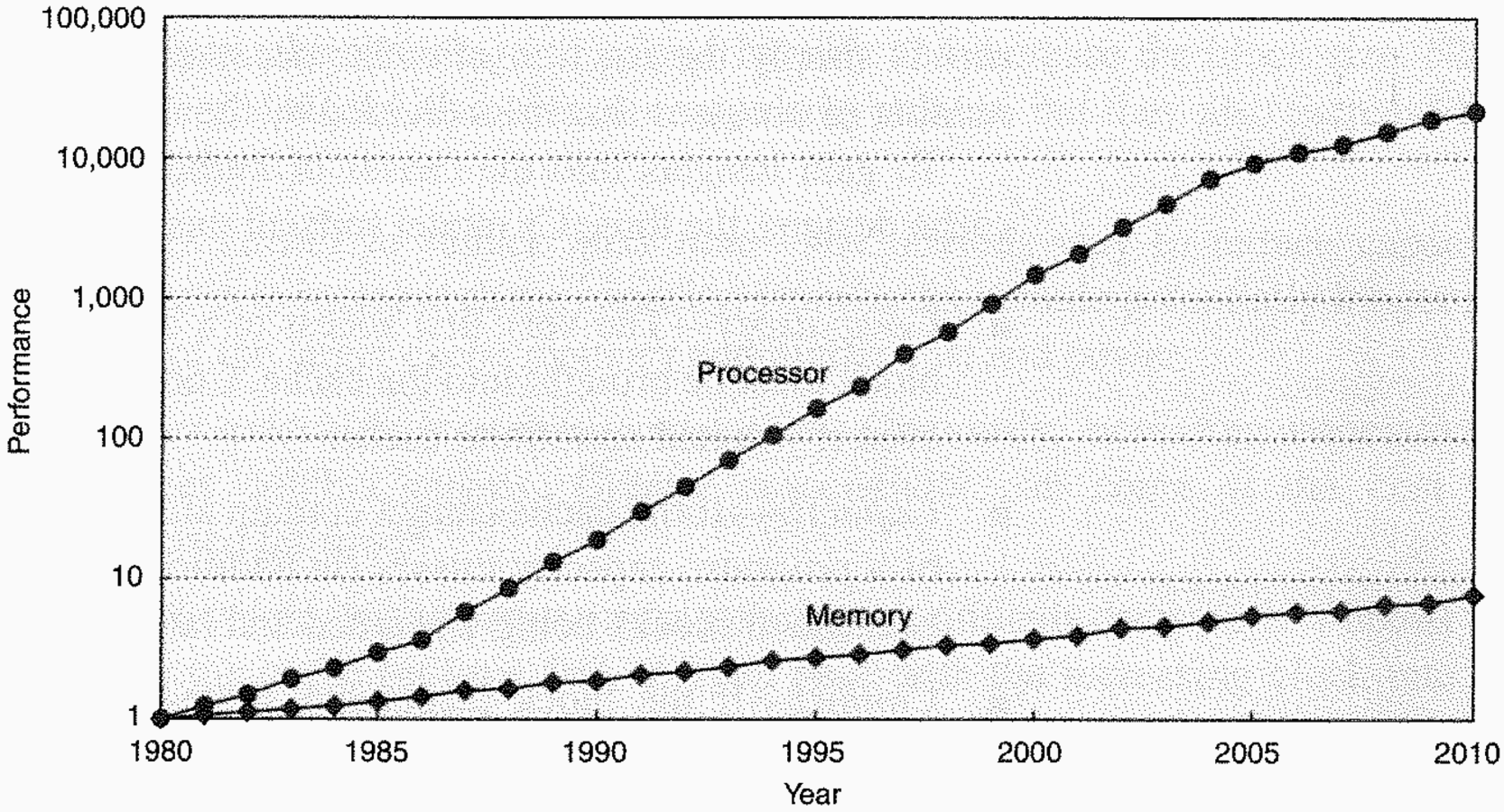
Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

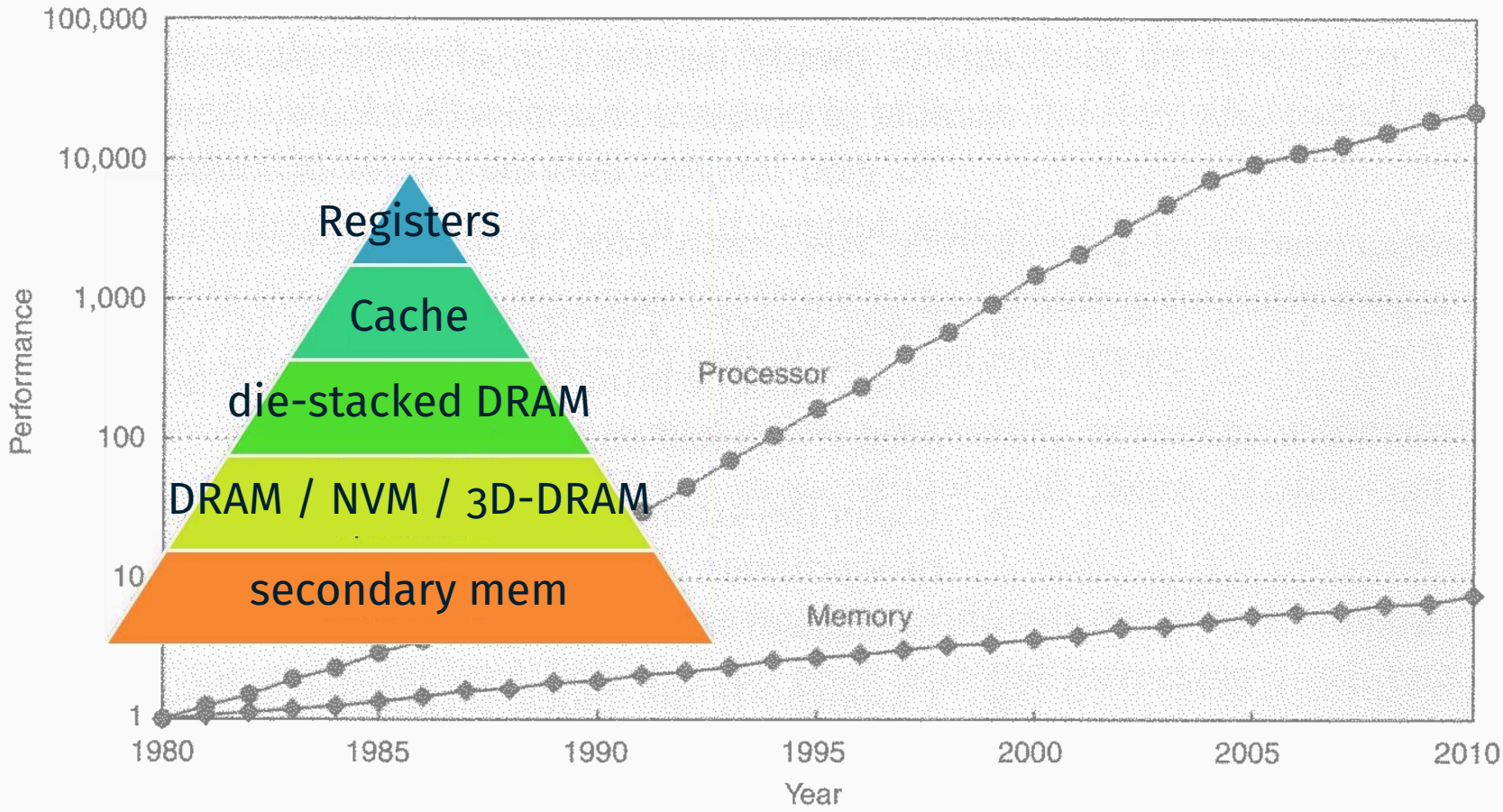
Motivation

memory wall [4]



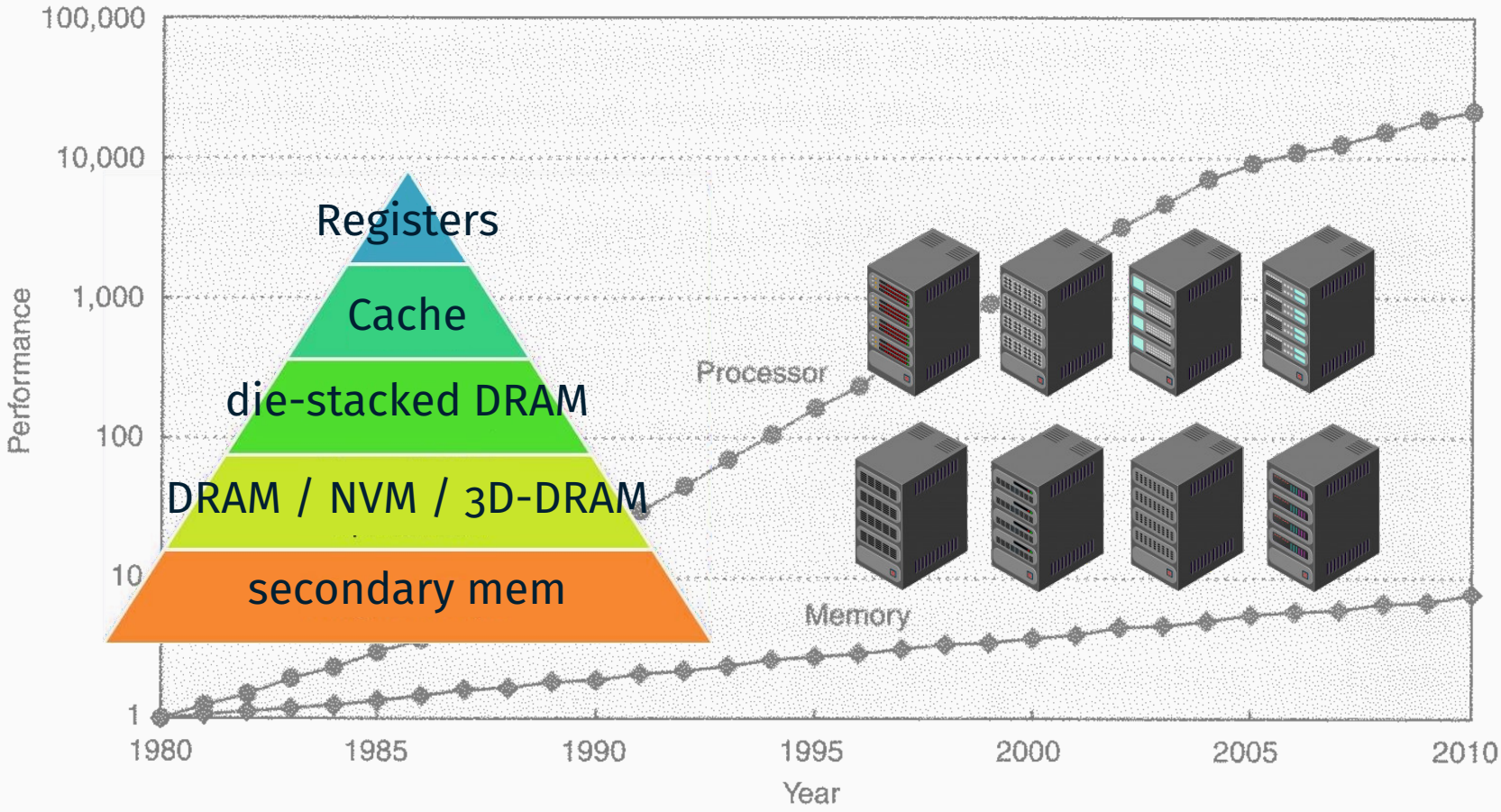
Motivation

memory wall [4]



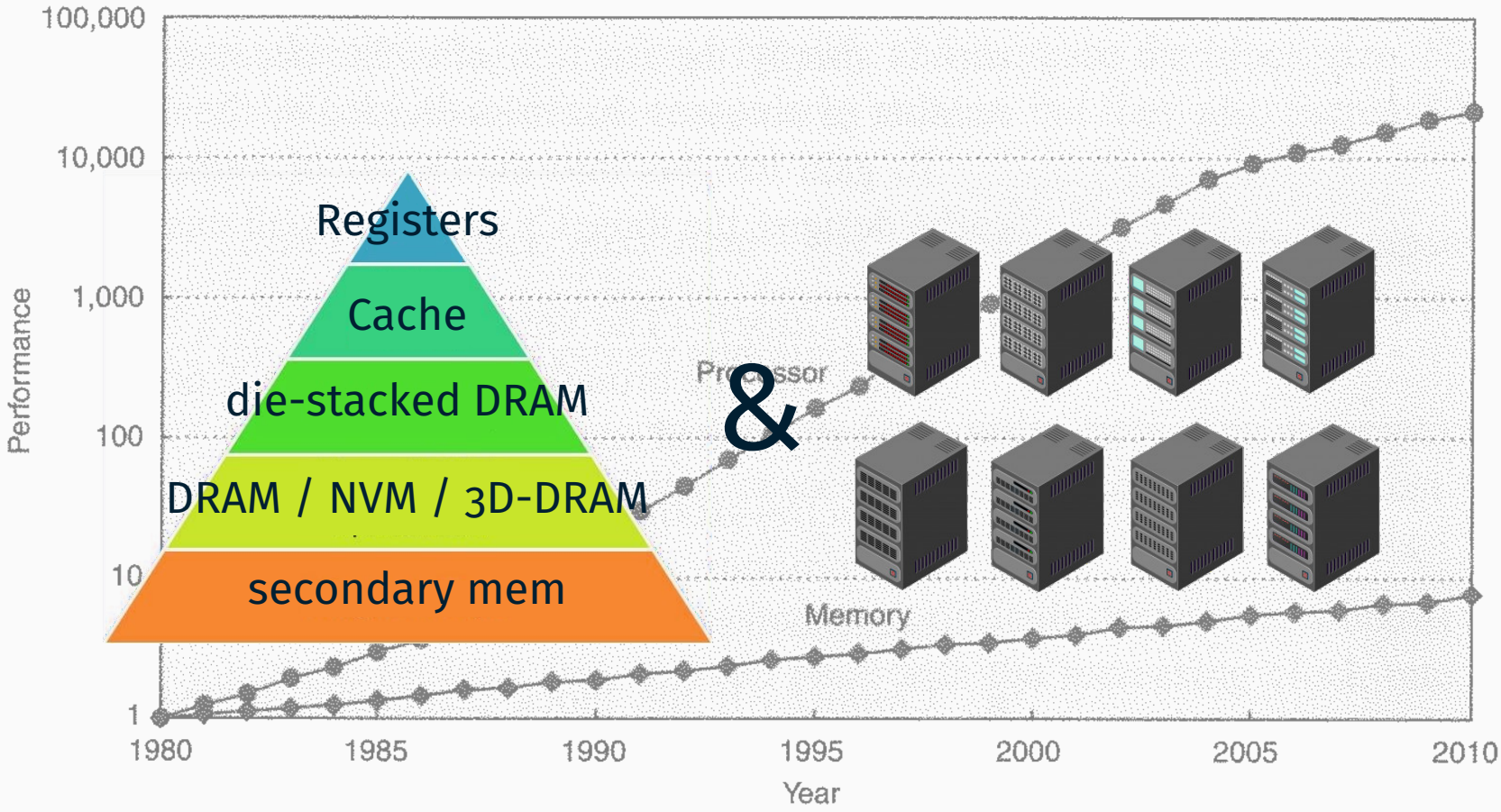
Motivation

memory wall [4]



Motivation

memory wall [4]



Overview

Grundlagen

HeteroOS

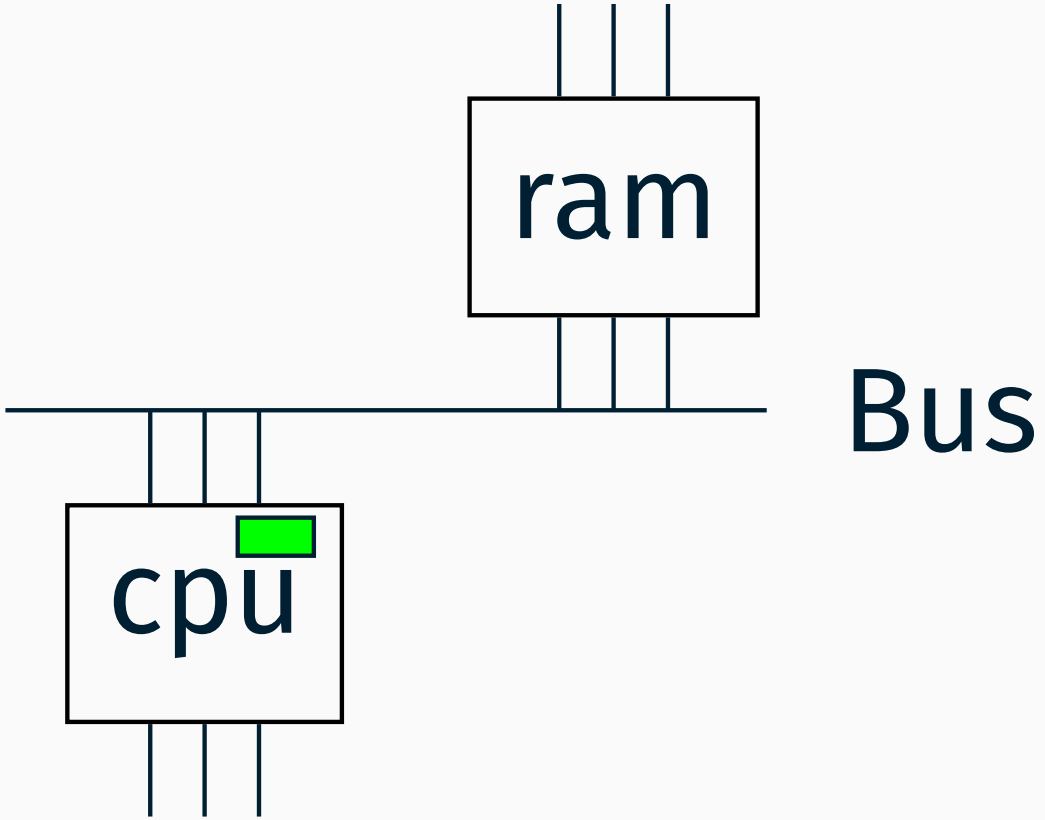
RAMCloud

Myrmics Speicherverwaltung

Fazit

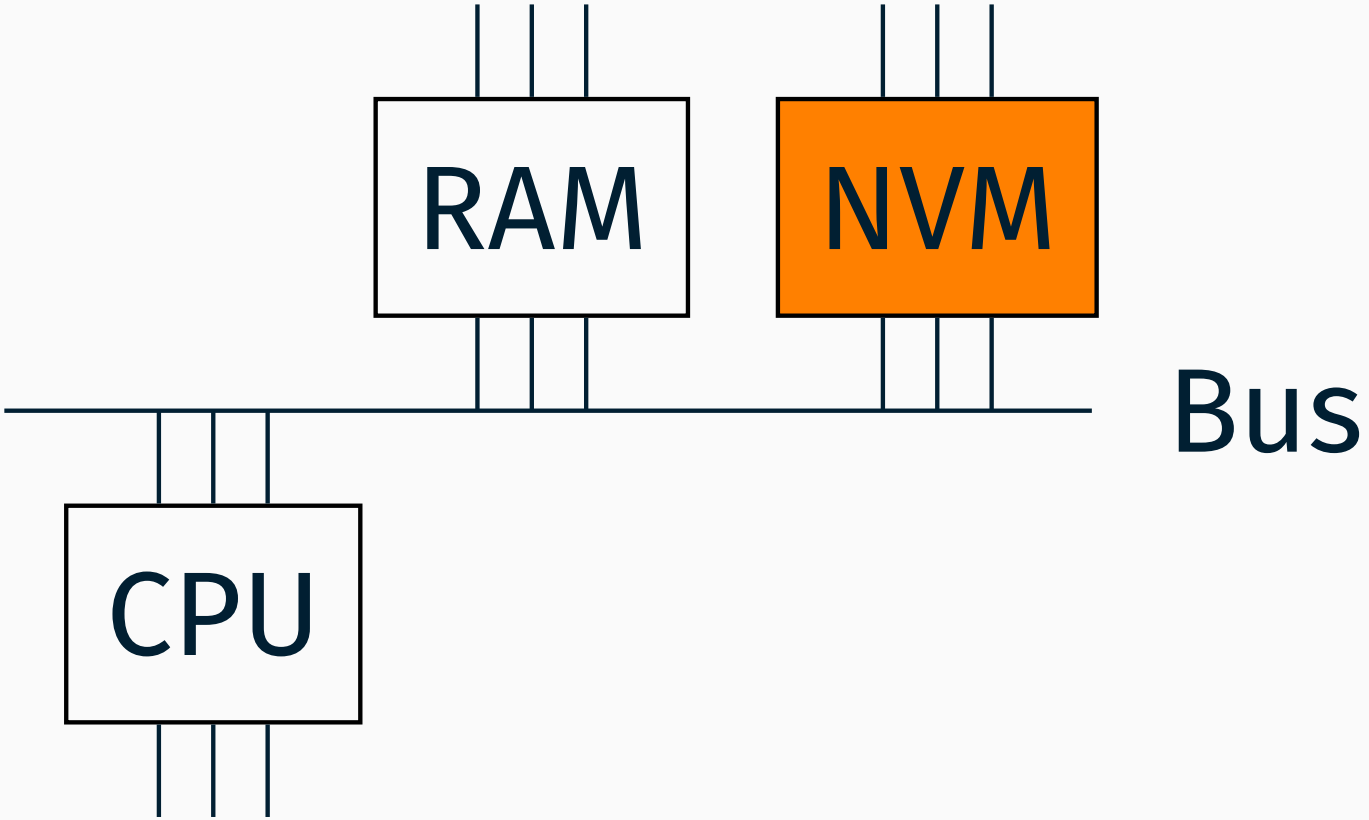
Grundlagen

Die-stacked DRAM



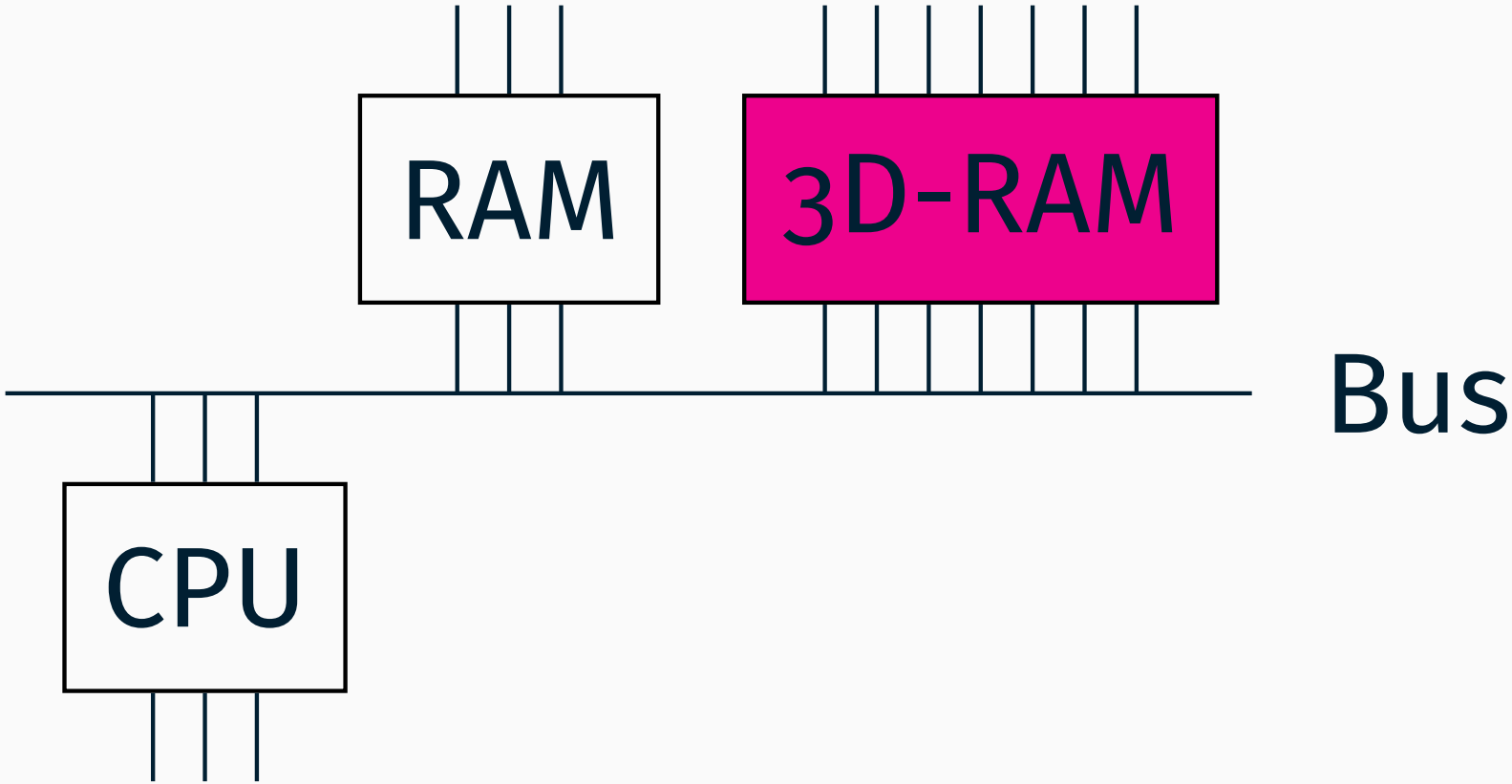
+ Kürzere Zugriffzeiten
- Geringere Kapazität

NVM [5]



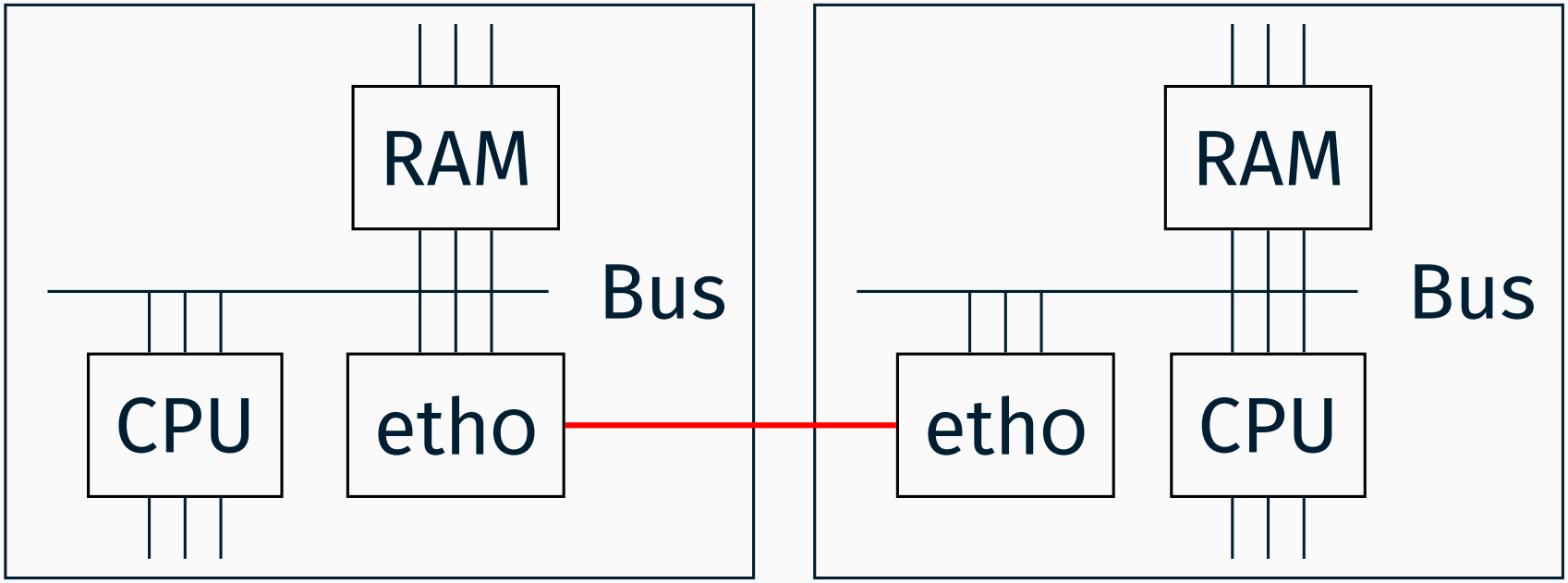
+ Persistenz
- Bedarf konsistenter Verwaltung
- Read & Write Asymmetrie
- Geringe Lebenszeit

3D-RAM

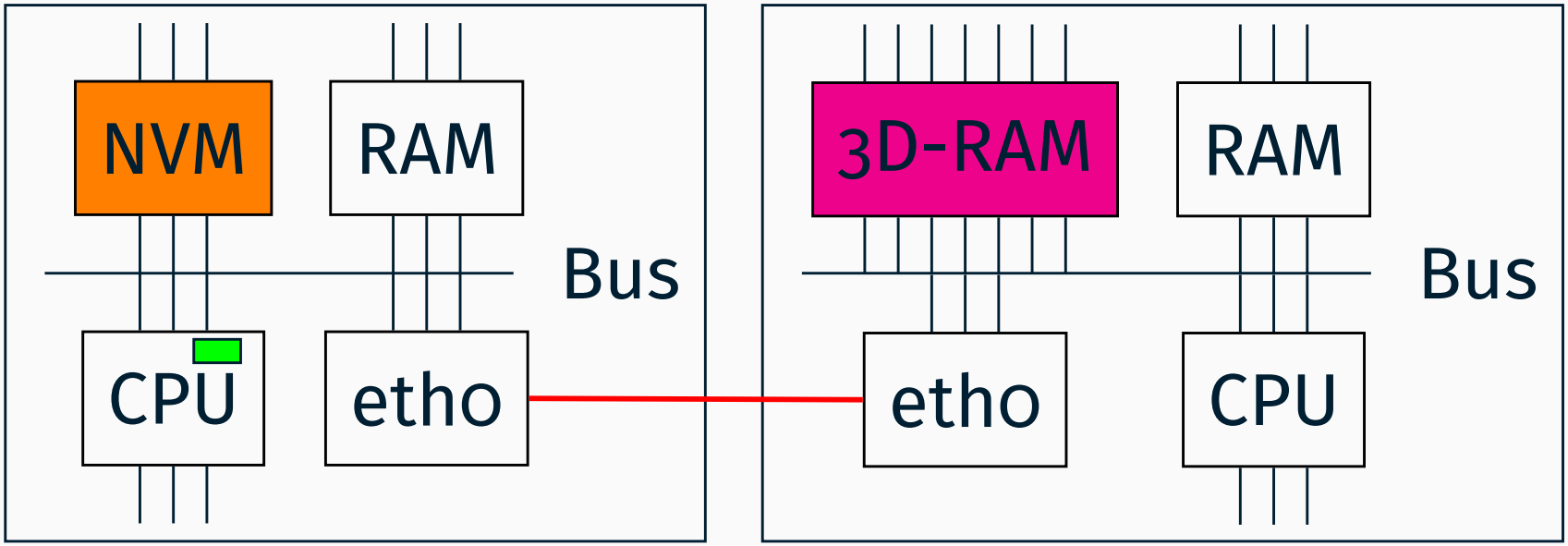


+ Hohe Bandbreite
- Geringe Kapazität
- Teuer

Netzwerk



+ Deutlich höhere Kapazitäten möglich
- Bedarf Kommunikation
- Hohe Latenzen
- Gemeinsame Verwaltung schwierig



+ Alle Vorteile der verbauten Speicher
+ Höhere Effizienz möglich durch geschickte Verwaltung
- Alle Nachteile der verbauten Speicher
- Große Verwaltungskomplexität

HeteroOS

Ziele & Ansatz[1]

Was? Heterogenität-gewahrer Hypervisor & Gast-BS

Wo? Elastisches Rechenzentrum mit heterogener Hardware

Warum? Faire & effiziente Verwaltung der Speicherressourcen

Wie?

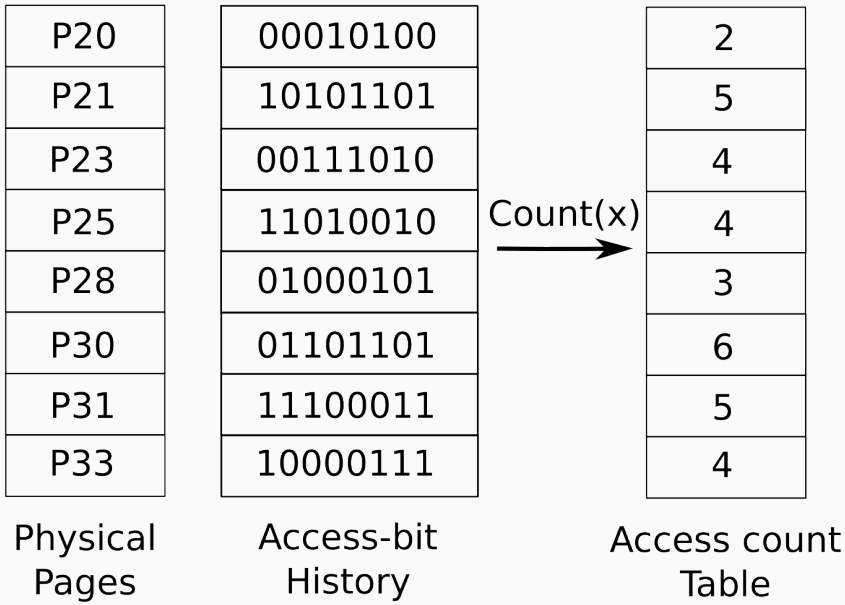
- Optimierung unterschiedlicher Anwendungen
- Beachtung mehrerer BS-Subsysteme
- Kooperation der Gäste mit dem Hypervisor

[1] "HeteroOS: OS Design for Heterogeneous Memory Management in Datacenter" - Kannan u. a. 2017

Gast-BS [8]

- Pro Speicherart ein unoptimierter NUMA node
- Zusätzliche Page Allokator Dimension pro Speicherart
- Speichertracking für I/O-Puffer, Kernel Objekte
- Erweitertes LRU und aktive Ersetzung
- Kooperation mit Hypervisor
  - Anforderung von Ressourcen
  - Beauftragung von *hotness tracking*

- Basierend auf HeteroVisor [3]
- DRF [2] Verteilung der Speicherressourcen
- On-demand Aktualisierung der Hotness-Bitmap [3]



→ Delegation der Seiten Migration an das Gast-BS

RAMCloud

Ziele & Ansatz[2]

Was? Persistenter Key-Value-Store

Wo? Elastisches Rechenzentrum

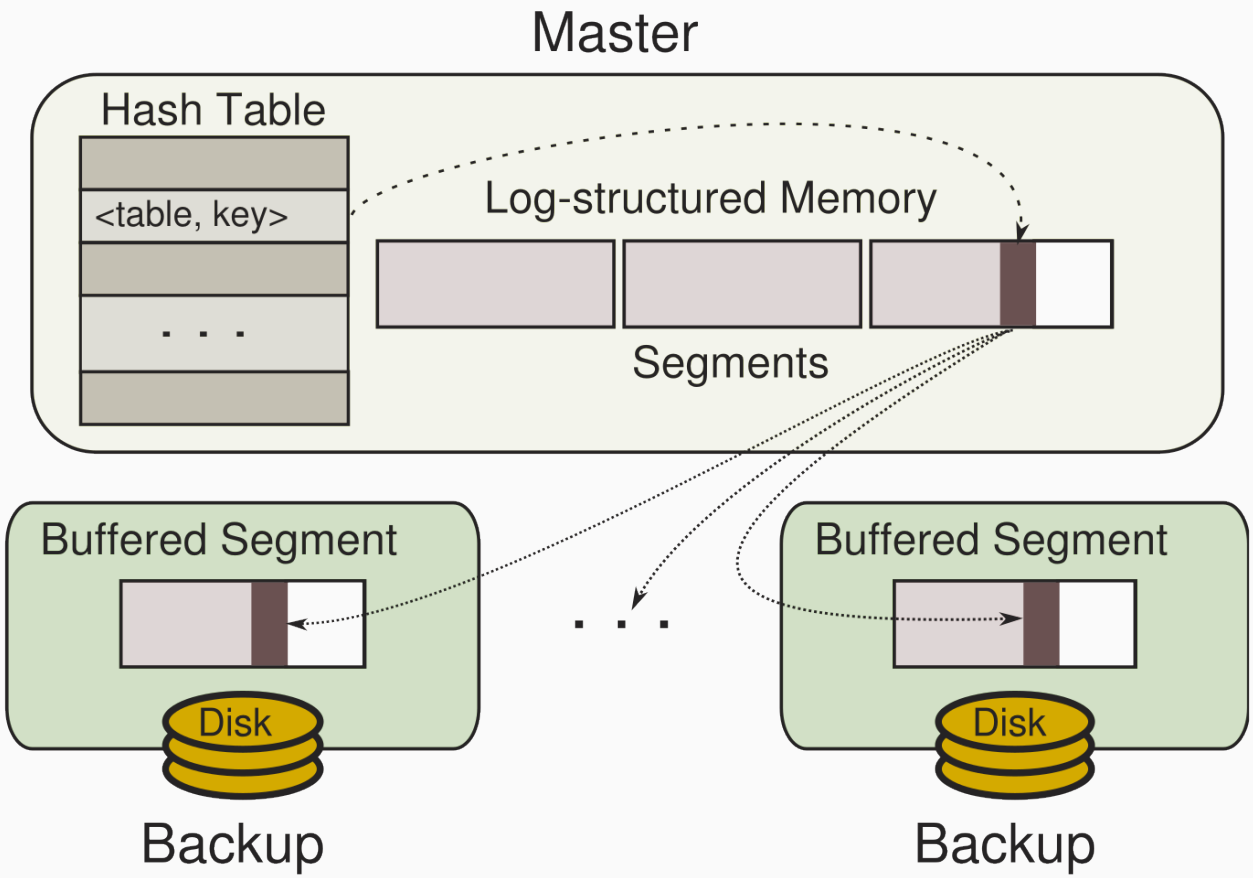
Warum? Hohe Speicherausnutzung des DRAM

Wie?

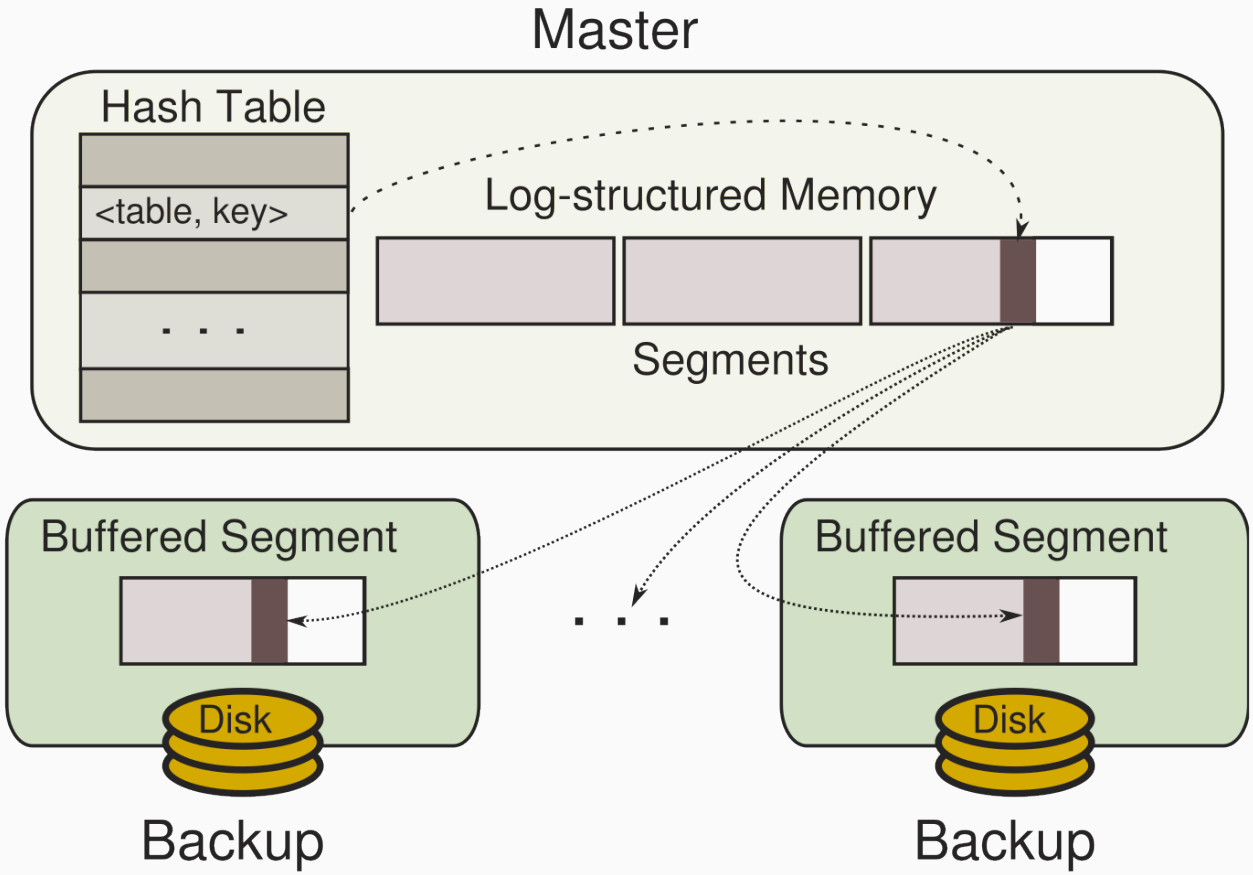
- Optimierung unterschiedlicher Anwendungen
- Garantierte Speicherausnutzung bei hohem Durchsatz
- Parallele und partielle Garbage Collection
- Replikation für ausfallsichere Persistenz

[2] "Log-structured memory for DRAM-based storage" - Rumble, Kejriwal und Ousterhout 2014

Architektur [8]

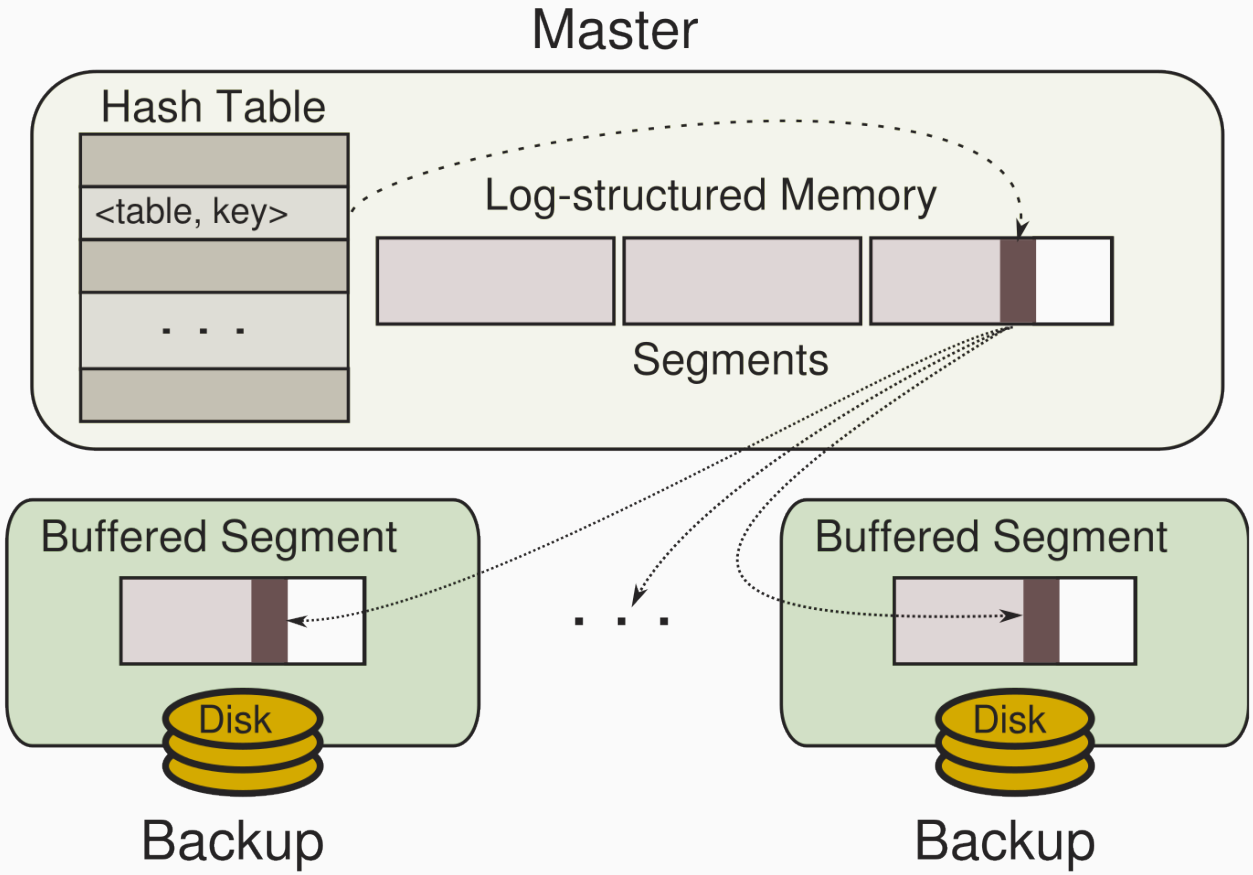


Architektur [8]



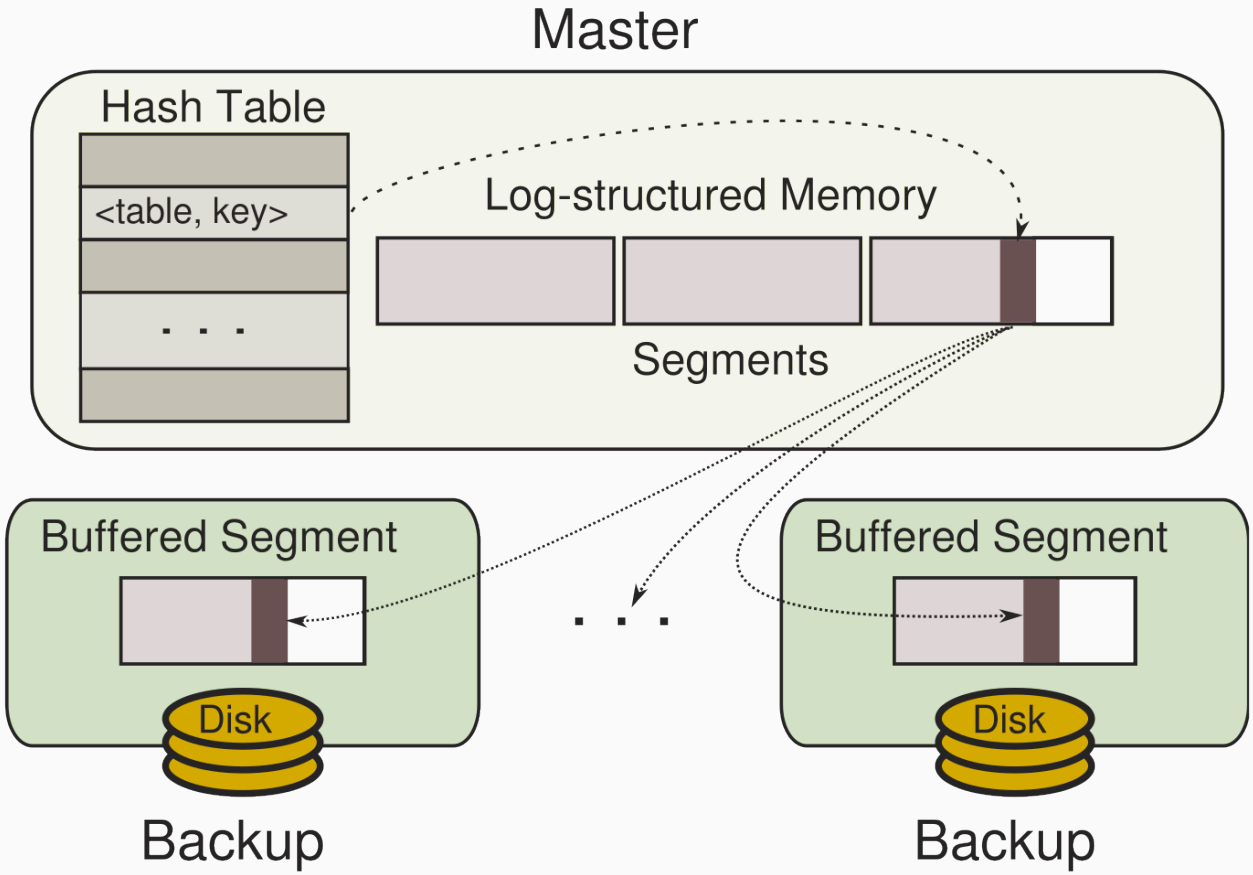
- Repliziertes Log in RAM und Backup Festplatten

Architektur [8]



- Repliziertes Log in RAM und Backup Festplatten
- Hashtabelle um Objekte im Log zu finden

Architektur [8]



- Repliziertes Log in RAM und Backup Festplatten
- Hashtabelle um Objekte im Log zu finden
- Persistente Puffer um Festplattenlatenz zu verstecken

Cleaning [8]

- Arbeitet auf einzelnen Segmenten
- Umkopieren lebendiger Objekte
- Benötigt viel Bandbreite
- Kosten proportional zur Speicherausnutzung

Cleaning [8]

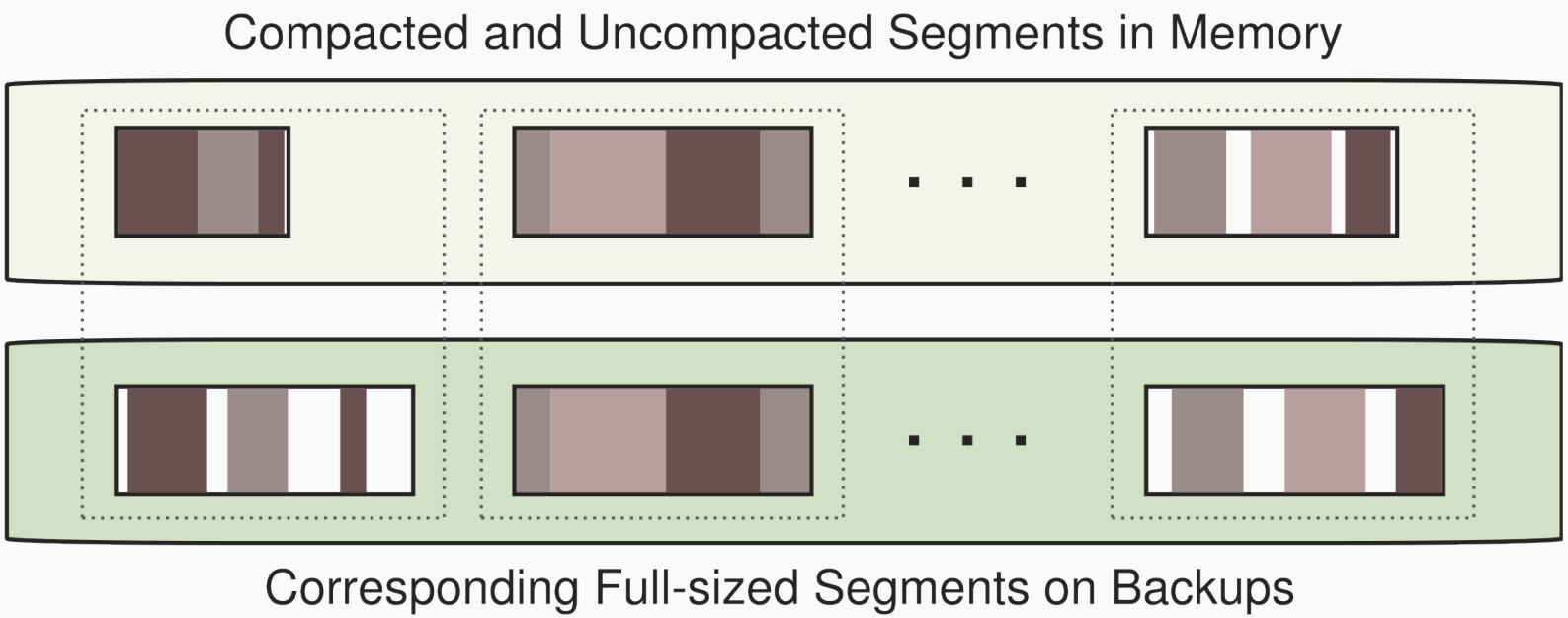
- Arbeitet auf einzelnen Segmenten
- Umkopieren lebendiger Objekte
- Benötigt viel Bandbreite
- Kosten proportional zur Speicherausnutzung

→ Zweistufige Speicherbereinigungen

Cleaning [8]

- Arbeitet auf einzelnen Segmenten
- Umkopieren lebendiger Objekte
- Benötigt viel Bandbreite
- Kosten proportional zur Speicherausnutzung

→ Zweistufige Speicherbereinigungen



Myrmics Speicherverwaltung

Ziele & Ansätze [3]

Was? Speicherverwaltung in task-based Laufzeitsystem

Wo? HPC / massive parallel computing

Warum? Kommunikation zeigerbasierter Datenstrukturen

Wie?

- `malloc` API mit Regionen
- Verwaltungskerne organisieren globalen Adressraum
- Effizientes Versenden zusammenhängender Objekte
- Mittelweg zwischen PGAS-Sprache und MPI

[3] "The myrmics memory allocator: hierarchical, message-passing allocation for global address spaces" - Lyberis u. a. 2012

API [7]

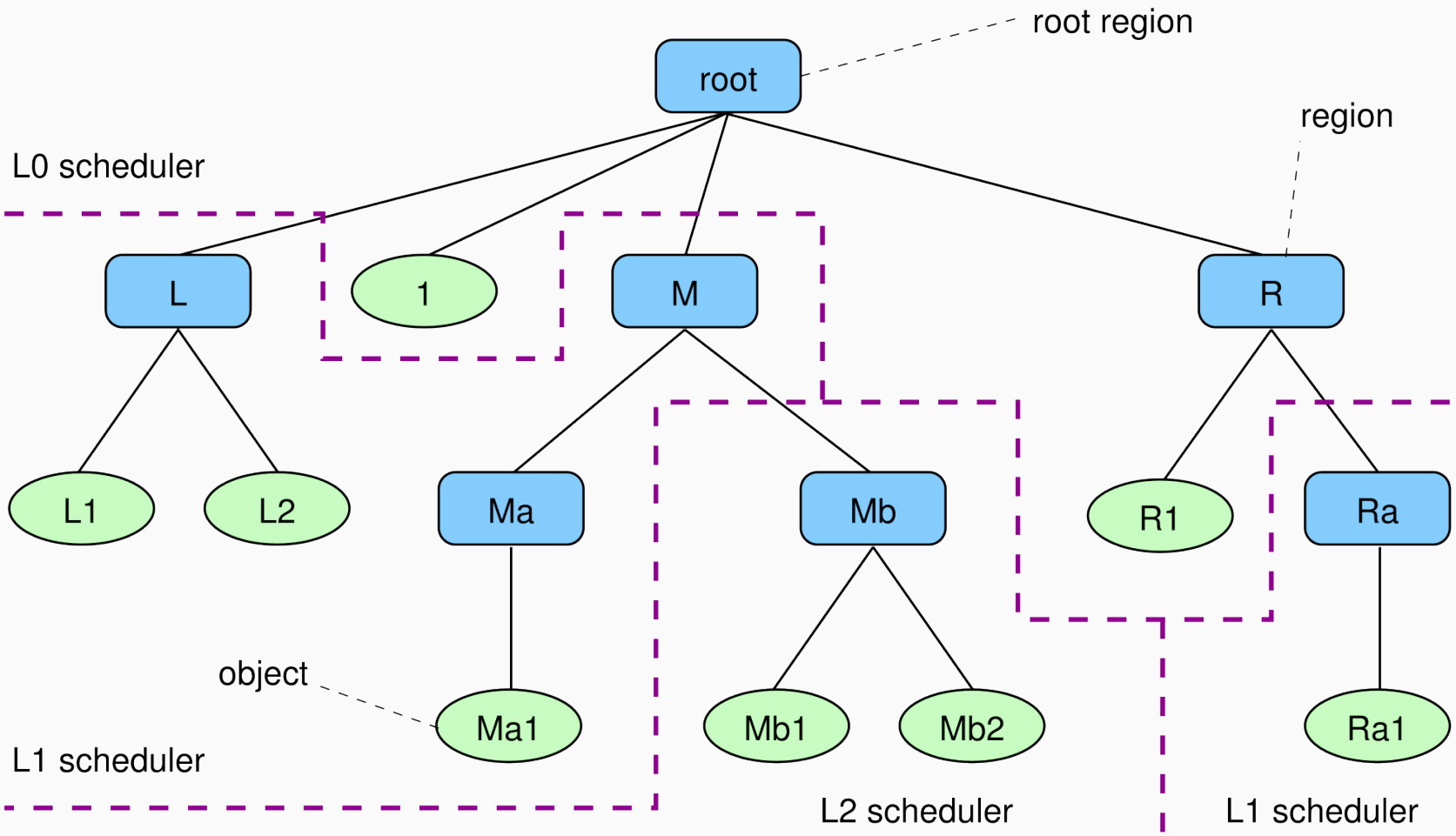
```
// Region management
rid_t sys_ralloc(rid_t parent, int level_hint);
void sys_rfree(rid_t region);
// Object management
void *sys_alloc(size_t size, rid_t region);
void sys_free(void *ptr);
...
// Communication
void sys_send(int peer_worker_id,
              rid_t *regions, int num_regions,
              void **objects, int num_objects);
void sys_recv(int peer_worker_id,
              rid_t **regions, int num_regions,
              void ***objects, int num_objects);
void sys_barrier();
```

Regionen [7]

- Bilden Hierarchie
- Exklusiv einem Verwaltungskern zugewiesen
- Strikte Seperation von Metadaten und Objekten
- Slab Verwaltung mit dynamischen Größenklassen

$\rightarrow$ leicht pack- und verschickbar

Verteilte Verwaltung [7]



Fazit

Fazit

	+		-
Kontrolle	Myrmics	RAMCloud	HeteroOS
Kommunikation	Myrmics	RAMCloud	HeteroOS
Transparenz	HeteroOS	RAMCloud	Myrmics
Effizienz	Alle		
Komplexität		RAMCloud HeteroOS	Myrmics

Danke für eure Aufmerksamkeit!

Fragen ?

Literatur

- Chiachen Chou, Aamer Jaleel und Moinuddin Qureshi. “BATMAN: Techniques for maximizing system bandwidth of memory systems with stacked-DRAM”. In: *Proceedings of the International Symposium on Memory Systems*. ACM. 2017, S. 268–280.
- Ali Ghodsi u. a. “Dominant Resource Fairness: Fair Allocation of Multiple Resource Types.”. In: *Nsdi*. Bd. 11. 2011. 2011, S. 24–24.

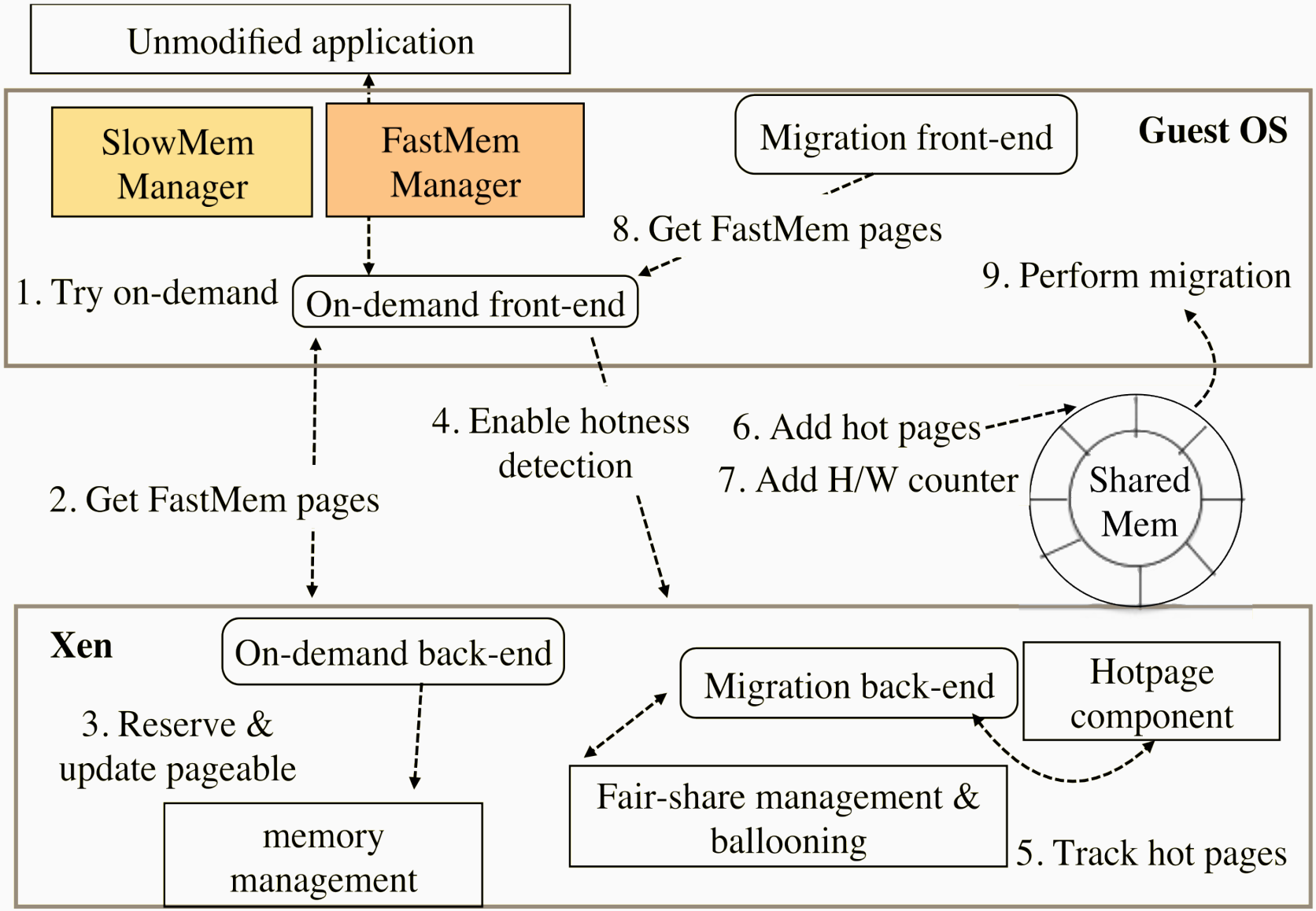
- Vishal Gupta, Min Lee und Karsten Schwan. “Heterovisor: Exploiting resource heterogeneity to enhance the elasticity of cloud platforms”. In: *ACM SIGPLAN Notices*. Bd. 50. 7. ACM. 2015, S. 79–92.
- John L Hennessy und David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.
- Sudarsun Kannan, Ada Gavrilovska und Karsten Schwan. “pVM: persistent virtual memory for efficient capacity scaling and object storage”. In: *Proceedings of the Eleventh European Conference on Computer Systems*. ACM. 2016, S. 13.

Sudarsun Kannan u. a. “HeteroOS: OS Design for Heterogeneous Memory Management in Datacenter”. In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ISCA ’17. Toronto, ON, Canada: ACM, 2017, S. 521–534. ISBN: 978-1-4503-4892-8. DOI: 10.1145/3079856.3080245. URL: http://doi.acm.org/10.1145/3079856.3080245.

Spyros Lyberis u. a. “The myrmics memory allocator: hierarchical, message-passing allocation for global address spaces”. In: *ACM SIGPLAN Notices*. Bd. 47. 11. ACM. 2012, S. 15–24.


Stephen M Rumble, Ankita Kejriwal und John Ousterhout. “Log-structured memory for DRAM-based storage”. In: *Proceedings of the 12th {USENIX} Conference on File and Storage Technologies ({FAST} 14)*. 2014, S. 1–16.

Speicherallokation [6]



Dominant Ressource Fairness[4]

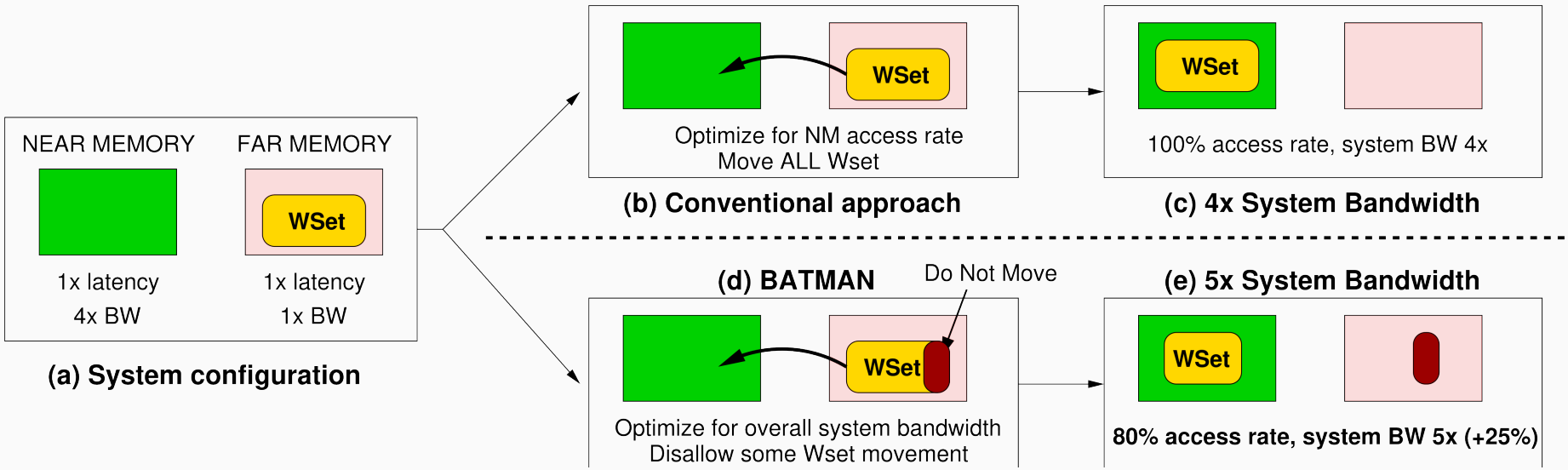
Algorithm 1 DRF pseudo-code

$R = \langle r_1, \cdots, r_m \rangle$ ▷ total resource capacities
$C = \langle c_1, \cdots, c_m \rangle$ ▷ consumed resources, initially 0
$s_i \ \ (i = 1..n)$ ▷ user i's dominant shares, initially 0
$U_i = \langle u_{i,1}, \cdots, u_{i,m} \rangle \ \ (i = 1..n)$ ▷ resources given to user i, initially 0

pick user i with lowest dominant share s_i
$D_i \leftarrow$ demand of user i's next task
if $C + D_i \leq R$ **then**
    $C = C + D_i$ ▷ update consumed vector
    $U_i = U_i + D_i$ ▷ update i's allocation vector
    $s_i = \max_{j=1}^{m} \{u_{i,j}/r_j\}$
else
    return ▷ the cluster is full
end if

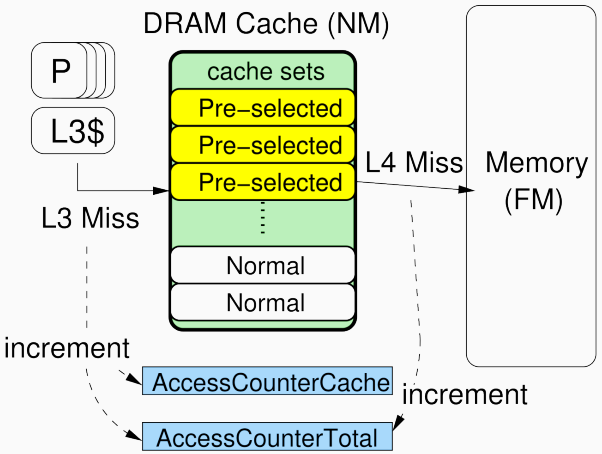
[4] "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types." - Ghodsi u. a. 2011

BATMAN - Approach[5]

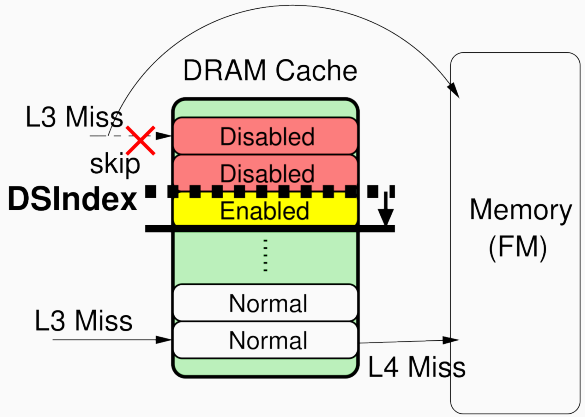


[5] "BATMAN: Techniques for maximizing system bandwidth of memory systems with stacked-DRAM" - Chou, Jaleel und Qureshi 2017

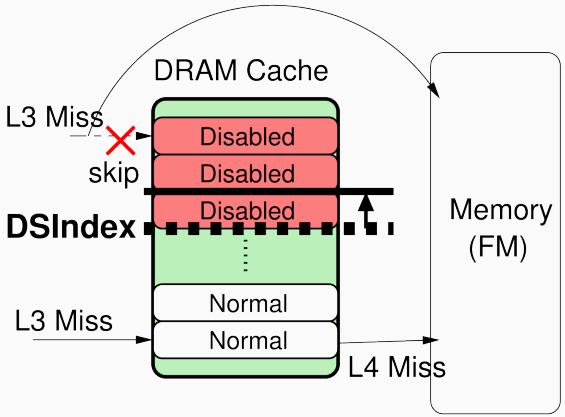
BATMAN - Implementation [1]



(a) BATMAN in the Cache–mode



(b) NM Access Rate > TAR:
Disable more sets



(c) NM Access Rate < TAR:
Enable more sets