Lehrstuhl für Connected Mobility
School of Computation, Information and Technology
Technische Universität München

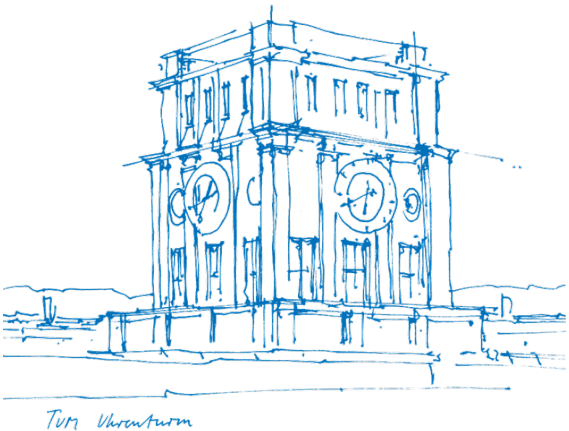
Grundlagen: Betriebssysteme und Systemsoftware

Tutorübung

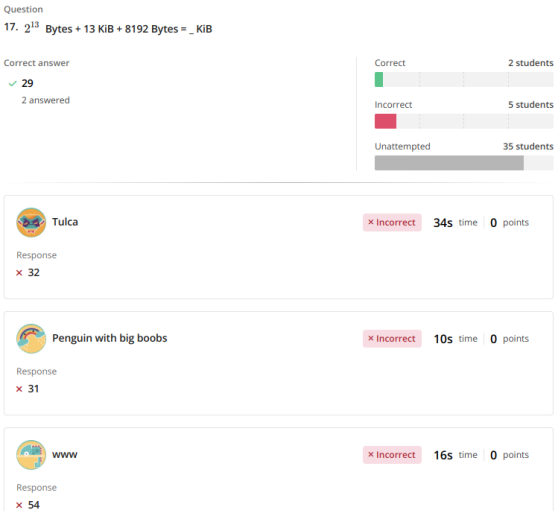
Mario Delic

Lehrstuhl für Connected Mobility
School of Computation, Information and Technology
Technische Universität München

Übungswoche 8

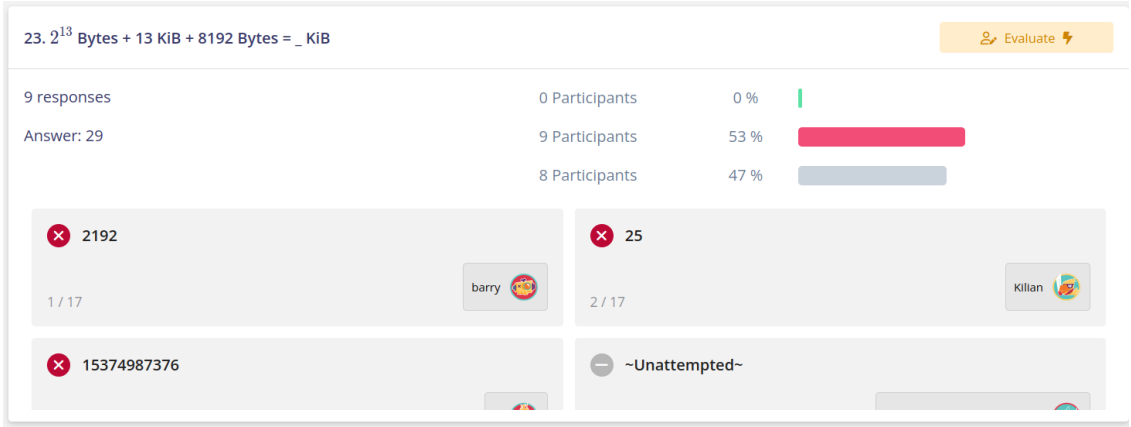


Rückblick auf die Ereignisse letzer Woche...



Rückblick auf die Ereignisse letzter Woche...

Immerhin waren die anderen genauso schlecht...



Rückblick auf die erste Übungswoche...

Dezimal:	Binär:	Hex:
1	2^0 = 1	0x1
2	2^1 = 10	0x2
4	2^2 = 100	0x4
8	2^3 = 1000	0x8
16	2^4 = 1'0000	0x10
32	2^5 = 10'0000	0x20
64	2^6 = 100'0000	0x40
128	2^7 = 1000'0000	0x80
256	2^8 = 1'0000'0000	0x100
512	2^9 = 10'0000'0000	0x200
1024	2^{10} = 100'0000'0000	0x400
2048	2^{11} = 1000'0000'0000	0x800
4096	2^{12} = 1'0000'0000'0000	0x1000

Grundwissen:

Ihr solltet im Kopf wissen:

1 KiB = 2^{10} = 1024 B
2 KiB = 2^{11} = 2048 B
4 KiB = 2^{12} = 4096 B
8 KiB = 2^{13} = 8192 B
16 KiB = 2^{14} = ... B
32 KiB = 2^{15} = ... B
...
1 MiB = 2^{20} = ... B
2 MiB = 2^{21} = ... B
4 MiB = 2^{22} = ... B
...
1 GiB = 2^{30} = ... B
2 GiB = 2^{31} = ... B

Weiteres:

8 4 2 1
0b 1 1 1 1 = 8+4+2+1 = 15 = 0xF
0b 1 0 1 0 = 8 + 2 = 10 = 0xA
0b 1 1 0 1 = 8+4 + 1 = 13 = 0xD

$2^{10} * 2^{13} = 2^{10+13} = 2^{23}$
$2^{27} \div 2^{13} = 2^{27-13} = 2^{14}$
$2^{10} + 2^{12}$ = ...ausrechnen! = 5120

Der Buddy Algorihtmus (informell)

Einfügen einer Datenmenge E.

1. Betrachte den kleinsten Block B, der größer als E ist (also wo E hineinpast).
2. Wiederhole bis Einfügen:
if (E passt nicht in hälfte von B) {
Füge D in B ein;
fertig;
}
if (E passt in hälfte von B) {
Halbiere Speicherblock B in zwei gleich große Unterblöcke;
Betrachte nun den linken der neuen Blöcke, dieser Block ist nun B;
continue;
}

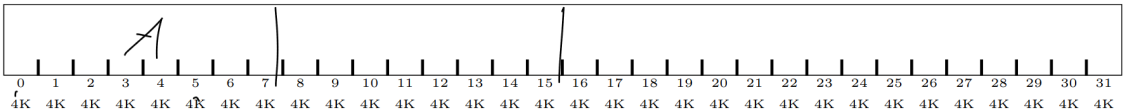
Der Buddy Algorihtmus (informell)

Freigeben einer Datenmenge F.

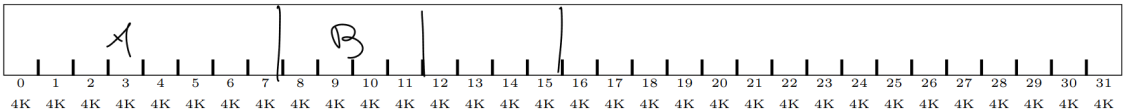
1. Lösche den Inhalt und markiere den Block von F als frei.
2. Verschmelze Blöcke solange möglich:
if (Buddy von F frei) {
Verschmelze die Buddys zum ursprünglichen Überblock;
Der neu entstandene große Block ist nun F;
continue;
}
else {
fertig;
}

Buddy-Schnellbeispiel

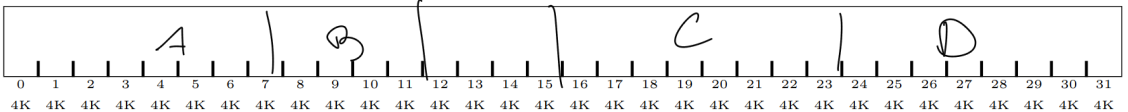
- A = allocate(32 KiB)



- B = allocate(16 KiB)

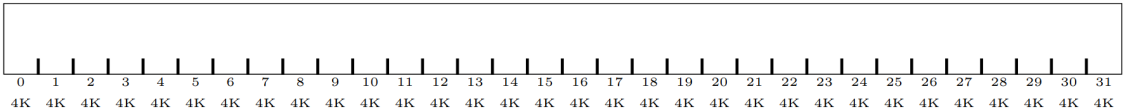


- C = allocate(20 KiB)

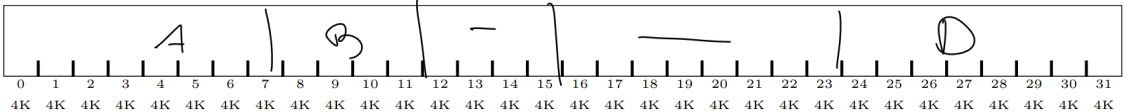


Buddy-Schnellbeispiel

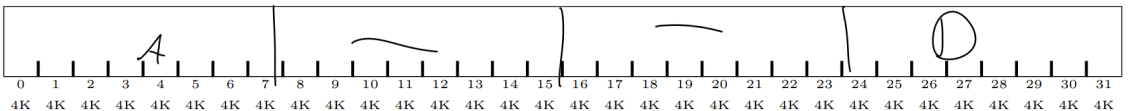
- D = allocate(32 KiB)



- free(C)



- free(B)

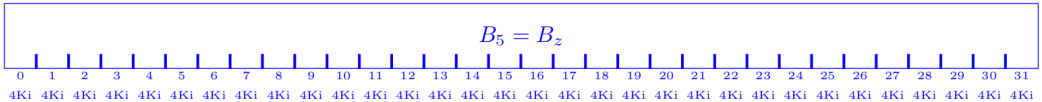


Aufgabe 2
Buddy-Algorithmus

Gegeben sei der Buddy-Allocator-Algorithmus gemäß Vorlesung, sowie der Pseudocode des folgenden Programms. Gehen Sie von einer Blockgröße von 4096 Bytes (4KiB) und einer Gesamtspeichergröße von 131072 Bytes (128KiB) aus.

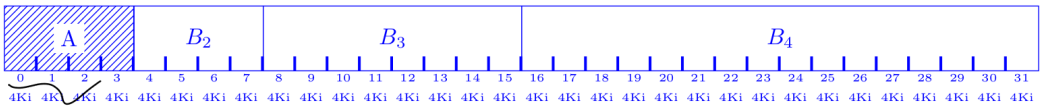
```
A = allocate(13337);
B = allocate(24242);
C = allocate(8193);
D = allocate(13);
free(A);
E = allocate(32768);
free(B);
F = allocate(10000);
G = allocate(12345);
H = allocate(11111);
free(G);
free(D);
free(H);
free(C);
free(F);
free(E);
```

1. Initialzustand: (Verschnitt 0 Bytes)

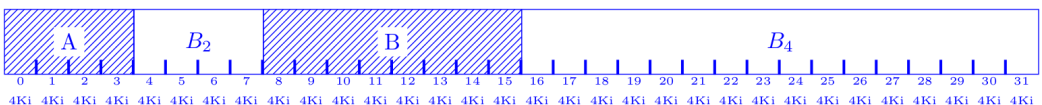


2. Nach `A = allocate(13337)`: (Verschnitt 3047 Bytes)

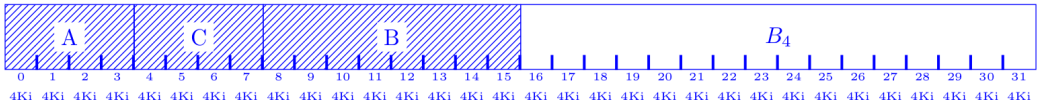
16384 − 13337 = 3047



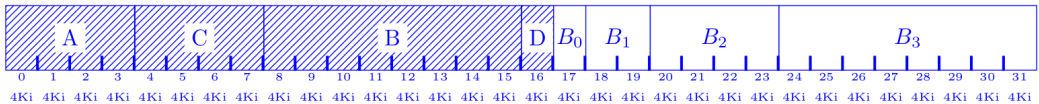
3. Nach `B = allocate(24242)`: (Verschnitt 11573 Bytes)



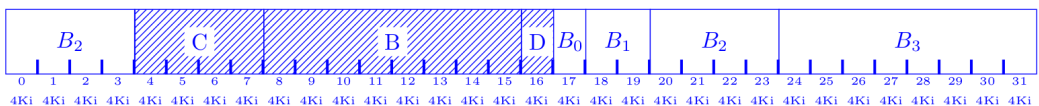
4. Nach `C = allocate(8193)`: (Verschnitt 19764 Bytes)



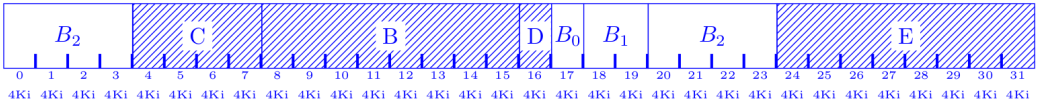
5. Nach `D = allocate(13)`: (Verschnitt 23847 Bytes)



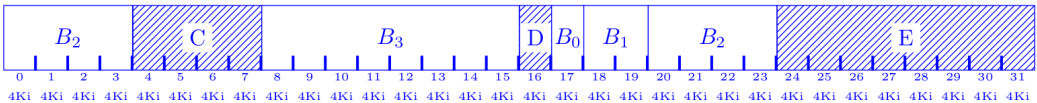
6. Nach `free(A)`: (Verschnitt 20800 Bytes)



7. Nach `E = allocate(32768)`: (Verschnitt 20800 Bytes)

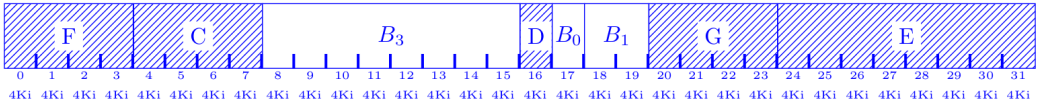


8. Nach `free(B)`: (Verschnitt 12274 Bytes)

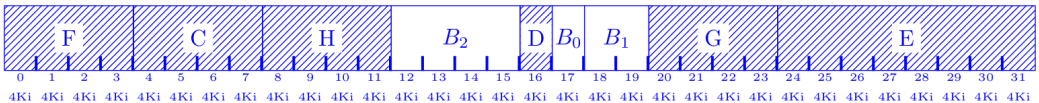


9. Nach `F = allocate(10000)`: (Verschnitt 18658 Bytes)

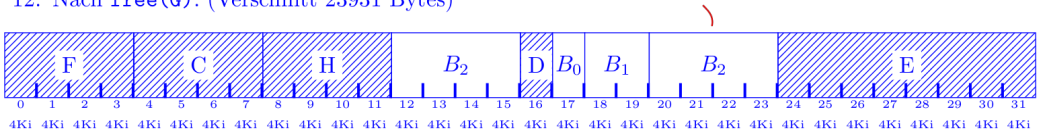
10. Nach `G = allocate(12345)`: (Verschnitt 22697 Bytes)



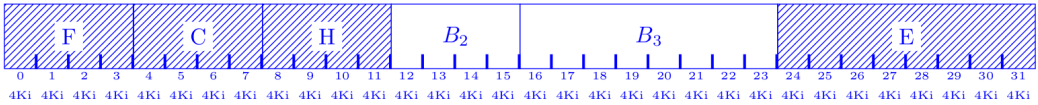
11. Nach `H = allocate(11111)`: (Verschnitt 27970 Bytes)



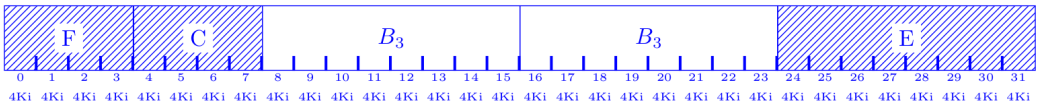
12. Nach `free(G)`: (Verschnitt 23931 Bytes)



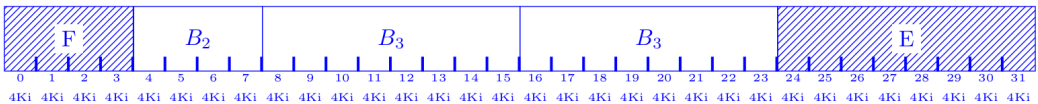
13. Nach `free(D)`: (Verschnitt 19848 Bytes)



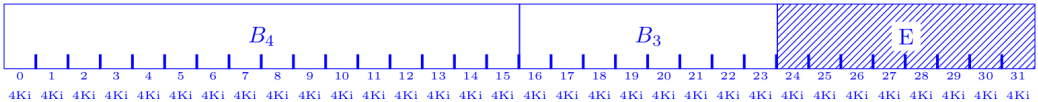
14. Nach `free(H)`: (Verschnitt 14575 Bytes)



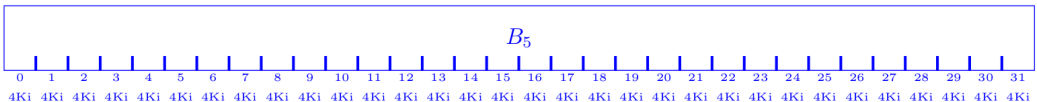
15. Nach `free(C)`: (Verschnitt 6384 Bytes)



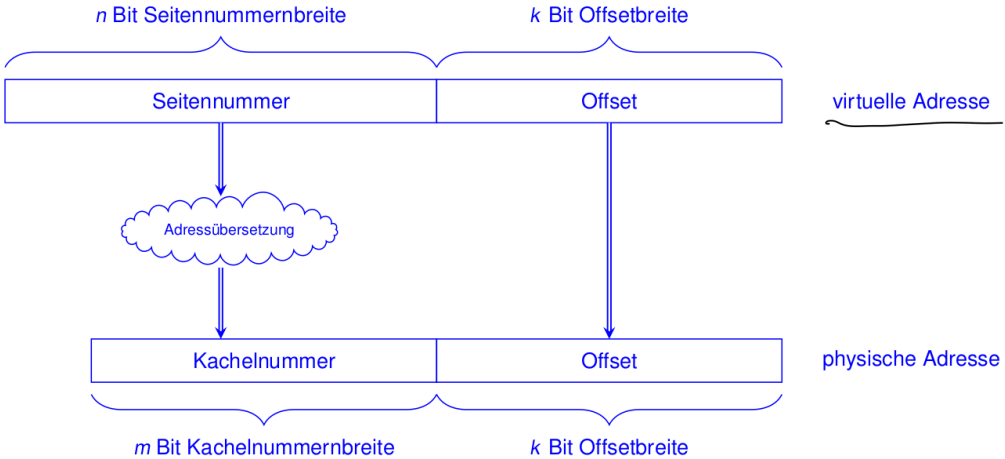
16. Nach `free(F)`: (Verschnitt 0 Bytes)



17. Nach `free(E)`: (Verschnitt 0 Bytes)



Adressen

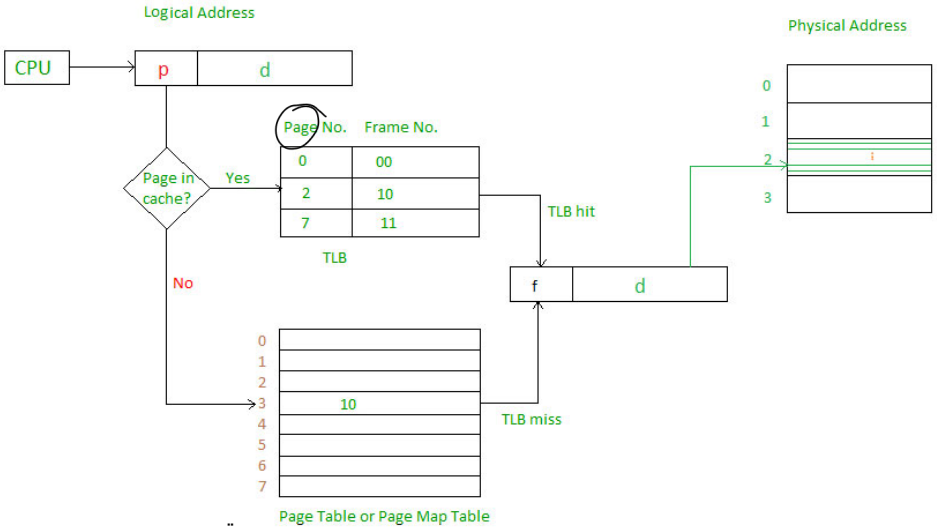

n Bit Seitennummernbreite
k Bit Offsetbreite
Seitennummer
Offset
virtuelle Adresse
Adressübersetzung
Kachelnummer
Offset
physische Adresse
m Bit Kachelnummernbreite
k Bit Offsetbreite

Paging Basics

• **Swapping**	↔	• **Paging**
Auslagern der **gesamten** Prozessdaten auf den Hintergrundspeicher.		**Seitenweises** Auslagern auf den Hintergrundspeicher (einzelne Prozessdaten können seperat ausgelagert werden).

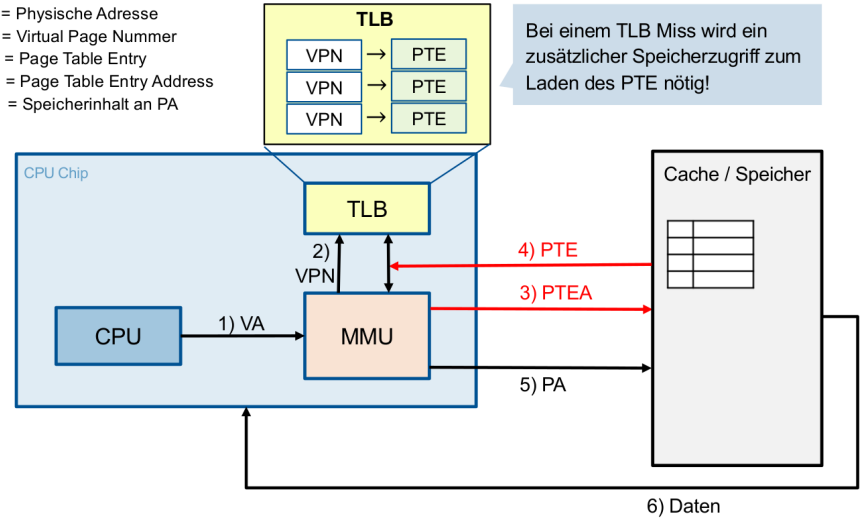
- Seiten sind virtuell, Kacheln sind physisch
- **Page Fault**: Wenn **P**resent-Bit der Seite nicht gesetzt ist raised die MMU einen PF ↪ Interrupt → Page Fault Handler → ggf. einlagern, PTE updaten → Interrupt-Kontext verlassen
- **Memory Management Unit (MMU)** ist für die Adressübersetzung zuständig
- MMU hat eigenen Cache: **Translation-Lookaside-Buffer (TLB)**

TLB


Logical Address
CPU
p
d
Page No.
Frame No.
Page in cache?
Yes
0
00
2
10
7
11
TLB
TLB hit
No
f
d
Physical Address
0
1
2
3
0
1
2
3
4
5
6
7
10
TLB miss
Page Table or Page Map Table

TLB

VA = Virtuelle Adresse
PA = Physische Adresse
VPN = Virtual Page Nummer
PTE = Page Table Entry
PTEA = Page Table Entry Address
Daten = Speicherinhalt an PA



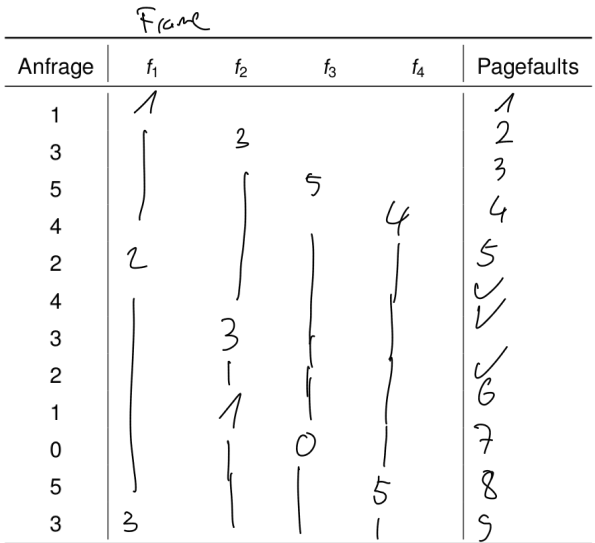
Aufgabe 2
Seitenersetzungsstrategien

- **FIFO**: Als erstes eingelagerte Seite wird als erstes ersetzt.

- **LRU (Least-Recently-Used)**:
Die Seite, deren Zugriff am längsten in der Vergangeneheit liegt wird ausgelagert.
↪ Speichern des Zeitpunkts t an dem letzter Zugriff erfolgte zus. zur Seitennnumer.

- **NFU (Not-Frequently-Used)**:
Die aktuell am wenigsten genutzt Seite wird ausgelagert.
↪ Speichern der Häufigkeit #, welche die Anzahl der Zugriffe seit Einlagerung darstellt.

FIFO



LRU

Anfrage	f_1, t	f_2, t	f_3, t	f_4, t	Nr Pagefaults
1	**1,1**				**1**
3	1,1	**3,2**			**2**
5	1,1	3,2	**5,3**		**3**
4	1,1	3,2	5,3	**4,4**	**4**
2	**2,5**	3,2	5,3	4,4	**5**
4	2,5	3,2	5,3	**4,6**	5
3	2,5	**3,7**	5,3	4,6	5
2	**2,8**	3,7	5,3	4,6	5
1	2,8	3,7	**1,9**	4,6	**6**
0	2,8	3,7	1,9	**0,10**	**7**
5	2,8	**5,11**	1,9	0,10	**8**
3	**3,12**	5,11	1,9	0,10	**9**

NFU

Anfrage	$f_1, \#$	$f_2, \#$	$f_3, \#$	$f_4, \#$	Nr Pagefaults
1	**1,1**				**1**
3	1,1	**3,1**			**2**
5	1,1	3,1	**5,1**		**3**
4	1,1	3,1	5,1	**4,1**	**4**
2	**2,1**	3,1	5,1	4,1	**5**
4	2,1	3,1	5,1	**4,2**	5
3	2,1	**3,2**	5,1	4,2	5
2	**2,2**	3,2	5,1	4,2	5
1	2,2	3,2	**1,1**	4,2	**6**
0	2,2	3,2	**0,1**	4,2	**7**
5	2,2	3,2	**5,1**	4,2	**8**
3	2,2	**3,3**	5,1	4,2	8

FIFO (5 Kacheln)

Anfrage	f_1	f_2	f_3	f_4	f_5	Nr Pagefaults
1	**1**					**1**
3	1	**3**				**2**
5	1	3	**5**			**3**
4	1	3	5	**4**		**4**
2	1	3	5	4	**2**	**5**
4	1	3	5	**4**	2	5
3	1	**3**	5	4	2	5
2	1	3	5	4	**2**	5
1	**1**	3	5	4	2	5
0	**0**	3	5	4	2	**6**
5	0	3	**5**	4	2	6
3	0	**3**	5	4	2	6

LRU (5 Kacheln)

Anfrage	f_1, t	f_2, t	f_3, t	f_4, t	f_5, t	Nr Pagefaults
1	**1,1**					**1**
3	1,1	**3,2**				**2**
5	1,1	3,2	**5,3**			**3**
4	1,1	3,2	5,3	**4,4**		**4**
2	1,1	3,2	5,3	4,4	**2,5**	**5**
4	1,1	3,2	5,3	**4,6**	2,5	5
3	1,1	**3,7**	5,3	4,6	2,5	5
2	1,1	3,7	5,3	4,6	**2,8**	5
1	**1,9**	3,7	5,3	4,6	2,8	5
0	1,9	3,7	**0,10**	4,6	2,8	**6**
5	1,9	3,7	0,10	**5,11**	2,8	**7**
3	1,9	**3,12**	0,10	5,11	2,8	7

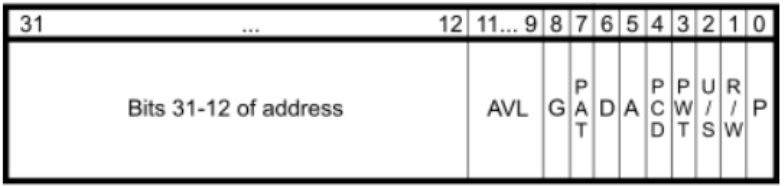
NFU (5 Kacheln)

Anfrage	$f_1, \#$	$f_2, \#$	$f_3, \#$	$f_4, \#$	$f_5, \#$	Nr Pagefaults
1	**1,1**					**1**
3	1,1	**3,1**				**2**
5	1,1	3,1	**5,1**			**3**
4	1,1	3,1	5,1	**4,1**		**4**
2	1,1	3,1	5,1	4,1	**2,1**	**5**
4	1,1	3,1	5,1	**4,2**	2,1	5
3	1,1	**3,2**	5,1	4,2	2,1	5
2	1,1	3,2	5,1	4,2	**2,2**	5
1	**1,2**	3,2	5,1	4,2	2,2	5
0	1,2	3,2	**0,1**	4,2	2,2	**6**
5	1,2	3,2	**5,1**	4,2	2,2	**7**
3	1,2	**3,3**	5,1	4,2	2,2	7

Die Bits

- P-bit (present): Seite ist im Hauptspeicher (1) oder nicht (0)
- U/S-bit (user/supervisor): Gibt an ob user(prozesse) Zugriff auf die Seite haben (1) oder nur der BS-Kern (0)
- XD-bit (eXecute Disable): Bytes in Page können nur gelesen/geschrieben werden (1) oder auch ausgeführt werden (0)
- **R-bit (referenced)**: Auf die Seite wurde lesend/schreibend zugegriffen (1)
- **M-bit (modified)**: Der Inhalt wurde verändert und muss auf den Hintergrundspeicher zurückgeschrieben werden (1), oder ist nicht verändert und kann verworfen werden (0)

So würde ein PageTableEntry aussehen



P: Present	**D:** Dirty
R/W: Read/Write	**G:** Global
U/S: User/Supervisor	**AVL:** Available
PWT: Write-Through	**PAT:** Page Attribute Table
PCD: Cache Disable	
A: Accessed	

Aging
Rechtsshift + Addition

- Aging ist eine Erweiterung der (NFU) Zählerstandberechnung
- Die Zählerstände werden nach einer Zeit t immer geupdated:

1) Bitshift nach rechts
2) Addieren des R-bits auf das vorderste (reingeshiftete 0) bit

Beispiel:
Seite A: Zugriffszähler $Z_A = 13_{10} = 1101_2$, R-bit $R_A = 0$
Seite B: Zugriffszähler $Z_B = 7_{10} = 0111_2$, R-bit $R_B = 1$

1) Rechtsshift $\rightarrow Z_A = 0110$, $Z_B = 0011$.
2) Addition R $\rightarrow Z_A = 0110$, $Z_B = 1011$.
$\hookrightarrow Z_A = 6$, $Z_B = 11$.

Second-Chance Clock

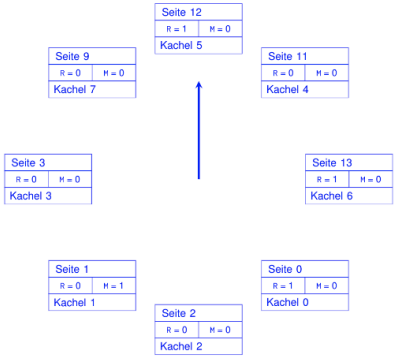
Clock-'Element' enthält: R-bit, M-bit, Seitennummer, Kachelnummer.

Besonderheiten bei der Clock:
- 'Zirkuläre' Anordnung
- Zeiger, der hinter das zuletzt ersetzte Element zeigt.
- Bei Seitenfehler bewegt sich der Zeiger (=Suche nach einer ersetzbaren Seite)

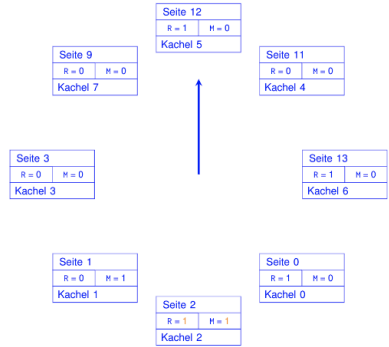
Lese- bzw. Schreibzugriffe ohne Seitenfehler modifizieren lediglich das R-bit bzw. R- und M-bit (der Zeiger bleibt unberührt)-

Second-Chance Clock

1. Initialbelegung

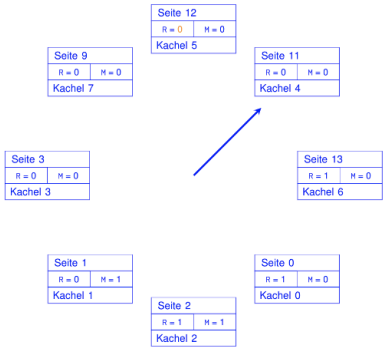


2. Schreibzugriff auf Seite 2; Befindet sich im Speicher ⇒ Setzen von R und M

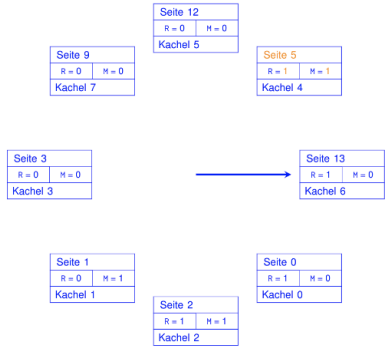


Second-Chance Clock

3. Schreibzugriff auf Seite 5; Seitenfehler ⇒ Suchen der ersten Seite mit R gleich 0, Löschen aller R-Bits auf dem Weg

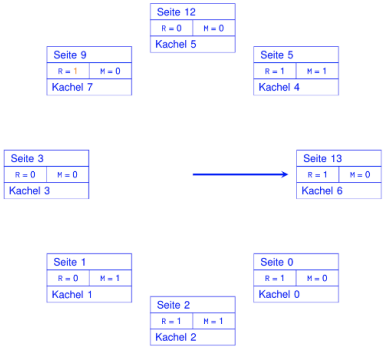


4. M und R gleich 0 ⇒ Überschreiben von Seite 11, Setzen des Zeigers hinter ersetzte Seite

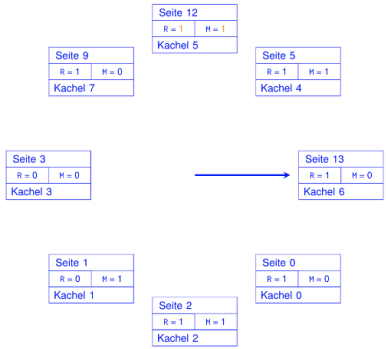


Second-Chance Clock

5. Lesezugriff auf Seite 9; Befindet sich im Speicher ⇒ Setzen von `R`

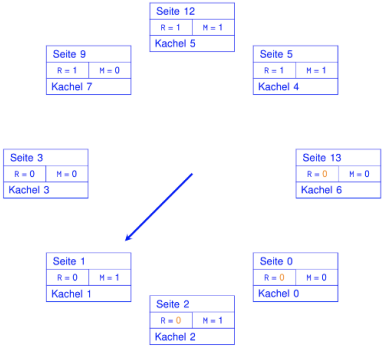


6. Schreibzugriff auf Seite 12; Befindet sich im Speicher ⇒ Setzen von `R` und `M`

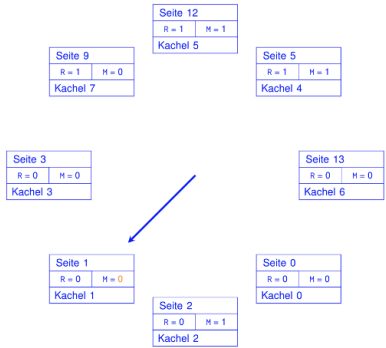


Second-Chance Clock

7. Schreibzugriff auf Seite 7; Seitenfehler ⇒ Suchen der ersten Seite mit R gleich 0, Löschen aller R-Bits auf dem Weg

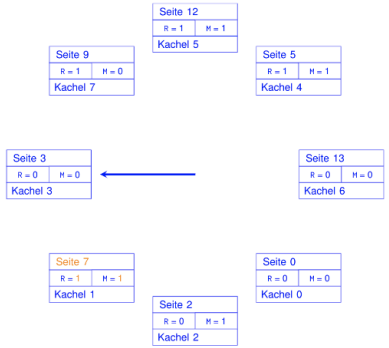


8. R gleich 0, M gleich 1 ⇒ Zurückschreiben von Seite 1 auf den Hintergrundspeicher, Anschließend Löschen des M-Bits.

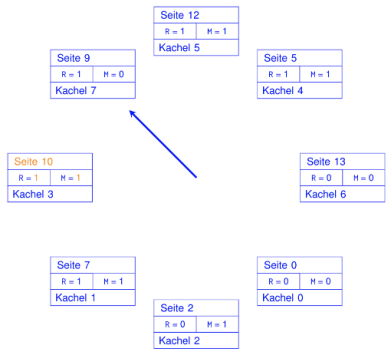


Second-Chance Clock

9. M und R gleich 0 ⇒ Überschreiben von Seite 1, Setzen des Zeigers hinter ersetzte Seite

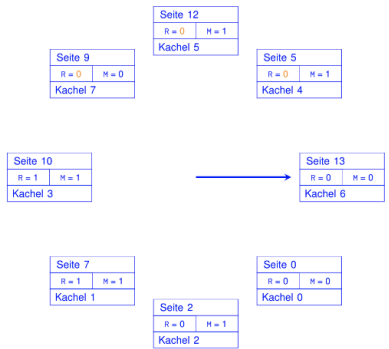


10. Schreibzugriff auf Seite 10; Seitenfehler ⇒ R und M gleich 0, Überschreiben von Seite 3, Setzen des Zeigers hinter ersetzte Seite



Second-Chance Clock

11. Lesezugriff auf Seite 1; Seitenfehler ⇒ Suchen der ersten Seite mit `R` gleich 0, Löschen aller `R`-Bits auf dem Weg



12. `M` und `R` gleich 0 ⇒ Überschreiben von Seite 13, Setzen des Zeigers hinter ersetzte Seite