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

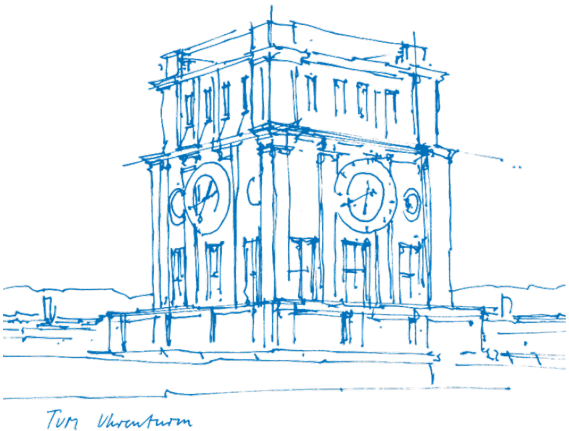
Grundlagen: Betriebssysteme und Systemsoftware

Tutorübung

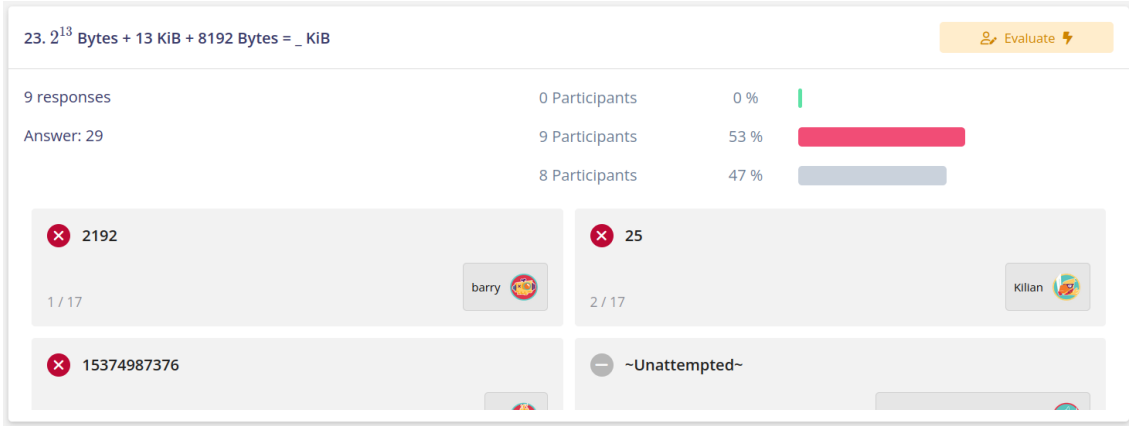
Mario Delic

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

Übungswoche 8

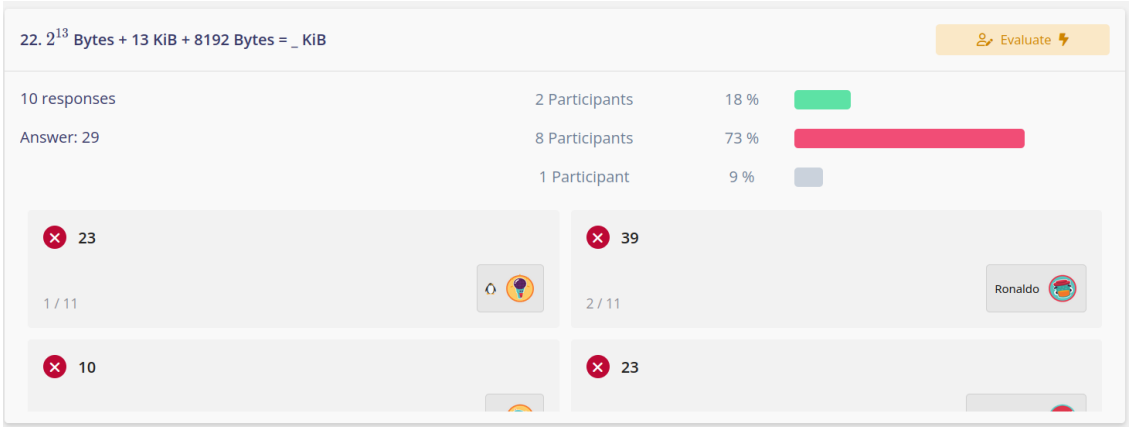


Rückblick auf die Ereignisse letzer Woche...



Rückblick auf die Ereignisse letzer 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

Über- und Ausblick

- **Offset-bits ermitteln**:

Über Kachel-/Seitengröße:

4 KiB große Seite $\rightarrow$ 4 KiB = 2^{12} Adressierbare **Bytes** $\rightarrow$ 12-bit Offset

Über bekannte Seitennummer-bits:

Adress-bits $-$ Seitennummer-bits = Offset-bits $\rightarrow$ 24-bit Architektur, 13 Seitennummer-bits $\rightarrow$ 24 $-$ 13 = 11 Bits für Offset

- **Seitennummer-bits ermitteln**:

Über Anzahl an Seitentabelleneinträgen:

512 Seitentabelleneinträge = 2^9 Seitentabelleneinträge $\rightarrow$ 9 Bits für Seitennummer

Über bekannte Offset-bits:

Adress-bits $-$ Offset-bits = Seitennummer-bits $\rightarrow$ 24-bit Architektur, 13 Offset-bits $\rightarrow$ 24 - 13 = 11 Bits für Seitennummer

Über- und Ausblick

- **Mehrstufige Seitentabellen**:

Einstufig: Seitennummer → indizierte Seitentabelle mit Übersetzungen der v_a zu p_a
Zweistufig: Seitentabelle mit Seitennummern → Seitentabelle mit Übersetzungen der v_a zu p_a

- **Anzahl an Blöcken ermitteln**:

Speichergröße÷Blockgröße → 2 MiB Speicher, $8\,\frac{\text{KiB}}{\text{Block}}$: $2^{21}/2^{13} = 2^8 = 256$ Blöcke

- **Blockgröße ermitteln**:

Speichergröße÷Blockzahl → 8 MiB Speicher, 2048 Blöcke: $2^{23}/2^{11} = 2^{12} = 4\,\frac{\text{KiB}}{\text{Block}}$

- **Adressen pro Block (=Elemente pro Struktur)**:

Strukturgröße÷Elementlänge → Blockgröße $4\,\frac{\text{KiB}}{\text{Block}}$, 64-bit Adresslänge → $2^{12}/2^3 = 2^9$ Elemente

↳ 8 Bytes = 2^3 Bytes pro Adr.

- **Indirect (=mehrstufige) Blöcke**:

Single-Indirect: Eine Adresse → Block mit Adressen → Datenblöcke

512 8-Byte Adressen in 4KiB-Block (8 · 512 = 4096)

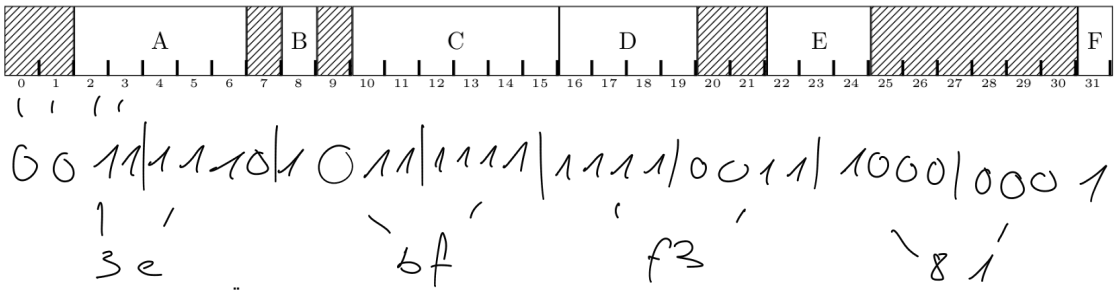
Double-Indirect: Eine Adresse → Block mit Adressen → Block mit Adressen → Datenblöcke

Aufgabe 2a
Bitmap

Jedes Bit codiert einen Block: 1 = belegt, 0 = frei.

Speicherbelegung → (Binärdarstellung →) Hexdarstellung.

Bekanntes Schema: vier Bits zu einer Hex-Ziffer zusammenfassen.

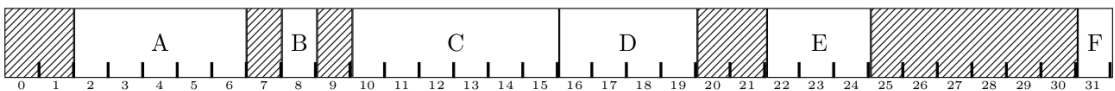


Aufgabe 2a
Bitmap

Jedes Bit codiert einen Block: 1 = belegt, 0 = frei.

Speicherbelegung $\rightarrow$ (Binärdarstellung $\rightarrow$) Hexdarstellung.

Bekanntes Schema: vier Bits zu einer Hex-Ziffer zusammenfassen.



0b 0011 1110 1011 11111 1111 0011 1000 0001

0x 3e bf f3 81

Aufgabe 2b
Verkettete Liste

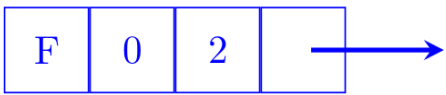
Vier Elemente pro Listeneintrag:

1. **F** (frei) oder **B** (belegt).

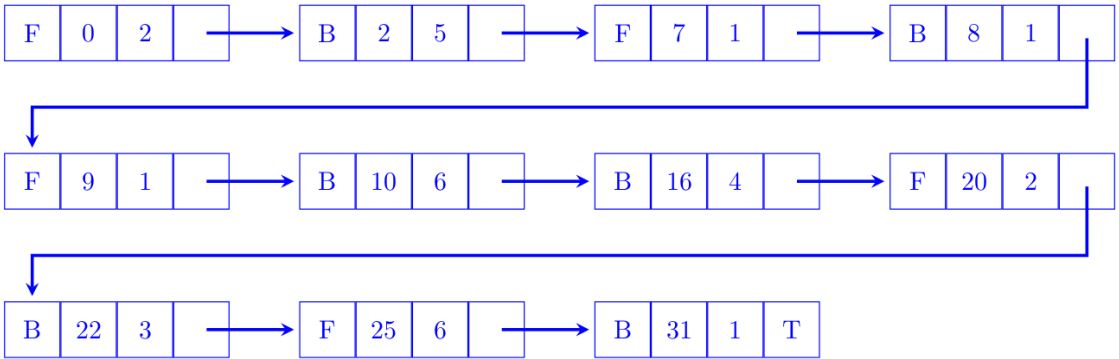
2. **Startadresse** dieses Speicherbereichs.

3. **Länge** dieses Speicherbereichs.

4. **Next-Pointer** auf Speicherbereich (modelliert durch Pfeil).



Aufgabe 2b
Verkettete List



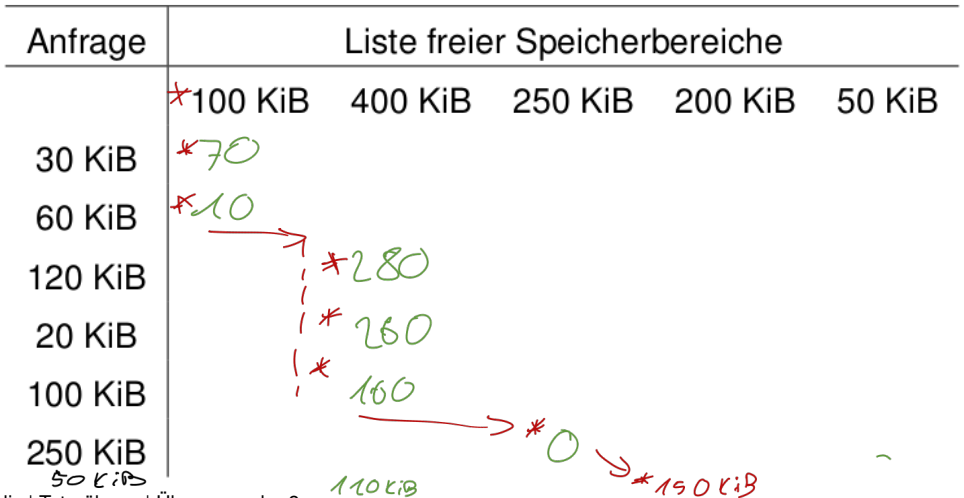
Die ’Fit’-Verfahren (informell)

• **First-Fit**:
Suche immer von Anfang an und belege den ersten, passenden Speicherplatz.
• **Next-Fit**:
Suche immer von der Stelle aus, in der zuletzt Eingefügt wurde (inklusive die Stelle selbst´!) und belege den ersten, passenden Speicherplatz.
• **Best-Fit**:
Durchsuche den gesamten Speicher und belege den Speicherplatz, bei dem der Größenunterschied am geringsten ist (Differenz $Speicherplatz - Anfrage$ so gering wie möglich).
• **Worst-Fit**:
Durchsuche den gesamten Speicher und belege den Speicherplatz, bei dem der Größenunterscheid am höchsten ist (Differenz $Speicherplatz - Anfrage$ so hoch wie möglich).

Aufgabe 3

First-Fit

Next Fit



Aufgabe 3
Next-Fit

Anfrage	Liste freier Speicherbereiche				
	100 KiB	400 KiB	250 KiB	200 KiB	50 KiB
30 KiB					
60 KiB					
120 KiB					
20 KiB					
100 KiB					
250 KiB					

siehe First F.

Aufgabe 3
Best-Fit

Anfrage	Liste freier Speicherbereiche				
	100 KiB	400 KiB	250 KiB	200 KiB	50 KiB
30 KiB					20
60 KiB	40				
120 KiB				80	
20 KiB					0
100 KiB			150		
250 KiB		150			

Aufgabe 3
Worst-Fit

Anfrage	Liste freier Speicherbereiche				
	100 KiB	400 KiB	250 KiB	200 KiB	50 KiB
30 KiB		370			
60 KiB		310			
120 KiB		190			
20 KiB			230		
100 KiB			130		
250 KiB		No space!			

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;
}

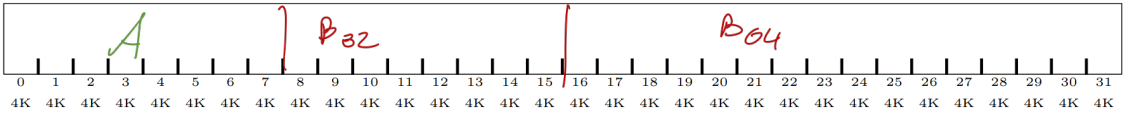
Der Buddy Algorihtmus (informell)

Freigeben einer Datenmenge F.

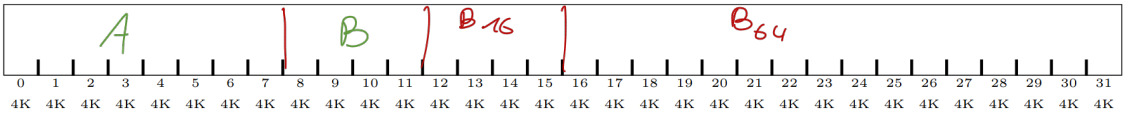
1. Lösche den Inhalt und markiere den Block von F als frei.
2. Verschmelze Blöcke solange möglich:
if (Buddy von F frei) {
Verschmelze die Buddys zum ursprünglichen Überblock;
Der neu entstandene große Block ist nun F;
continue;
}
else {
fertig;
}

Buddy-Schnellbeispiel

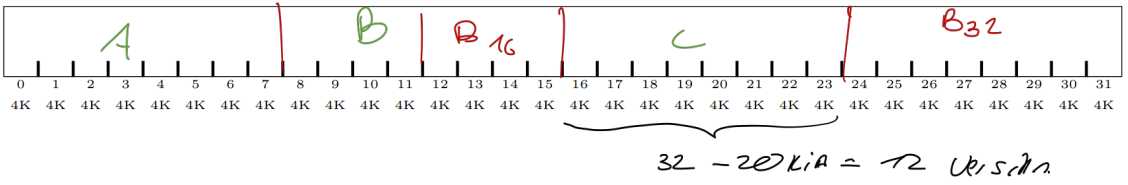
- A = allocate(32 KiB)



- B = allocate(16 KiB)

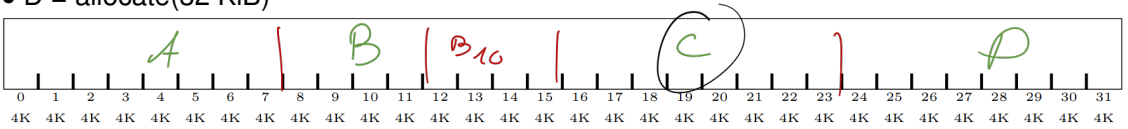


- C = allocate(20 KiB)

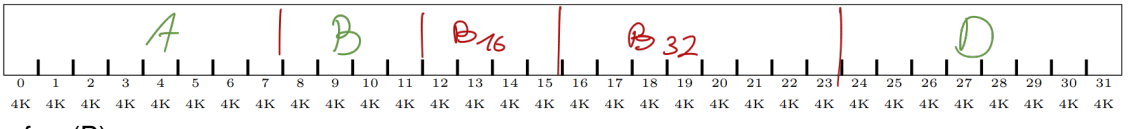


Buddy-Schnellbeispiel

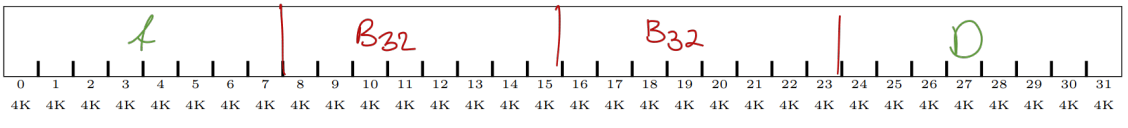
- D = allocate(32 KiB)



- free(C)



- free(B)

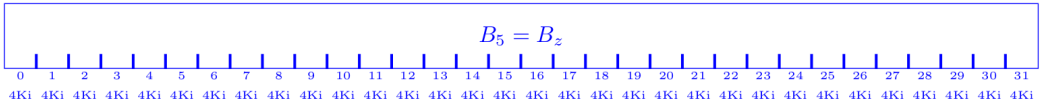


Aufgabe 4
Buddy-Algorithmus

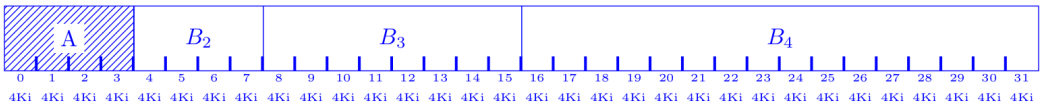
Gegeben sei der Buddy-Allocator-Algorithmus gemäß Vorlesung, sowie der Pseudocode des folgenden Programms. Gehen Sie von einer Blockgröße von 4096 Bytes (4KiB) und einer Gesamtspeichergröße von 131072 Bytes (128KiB) aus.

```
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);
```

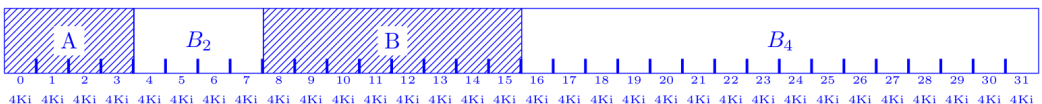
1. Initialzustand: (Verschnitt 0 Bytes)



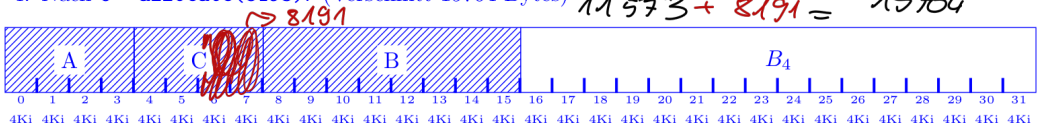
2. Nach `A = allocate(13337)`: (Verschnitt 3047 Bytes)



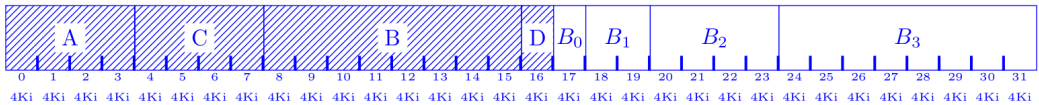
3. Nach `B = allocate(24242)`: (Verschnitt 11573 Bytes)



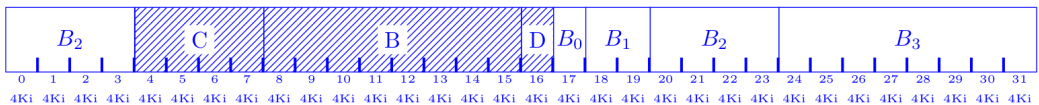
4. Nach `C = allocate(8193)`: (Verschnitt 19764 Bytes)



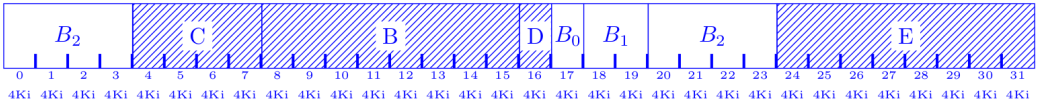
5. Nach `D = allocate(13)`: (Verschnitt 23847 Bytes)



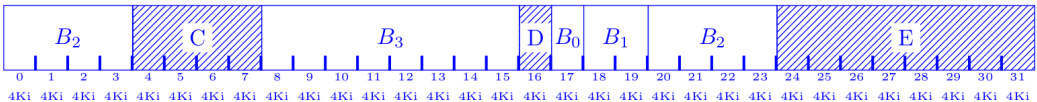
6. Nach `free(A)`: (Verschnitt 20800 Bytes)



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

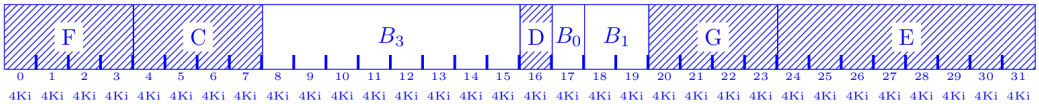


8. Nach `free(B)`: (Verschnitt 12274 Bytes)

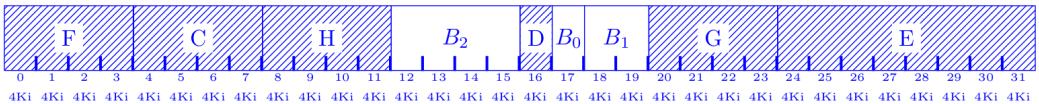


9. Nach `F = allocate(10000)`: (Verschnitt 18658 Bytes)

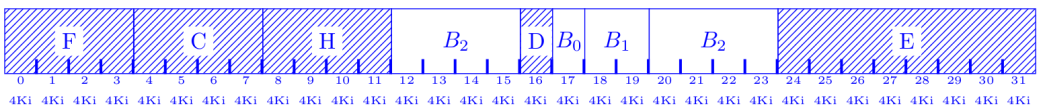
10. Nach `G = allocate(12345)`: (Verschnitt 22697 Bytes)



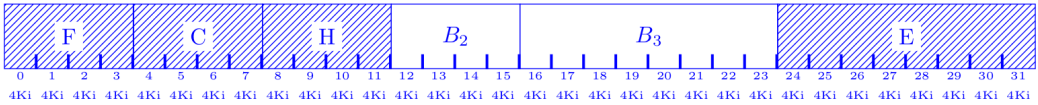
11. Nach `H = allocate(11111)`: (Verschnitt 27970 Bytes)



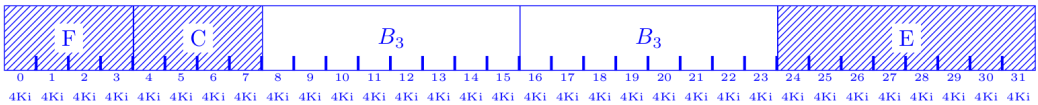
12. Nach `free(G)`: (Verschnitt 23931 Bytes)



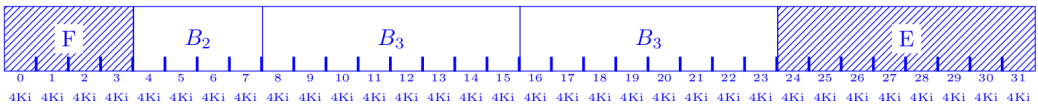
13. Nach `free(D)`: (Verschnitt 19848 Bytes)



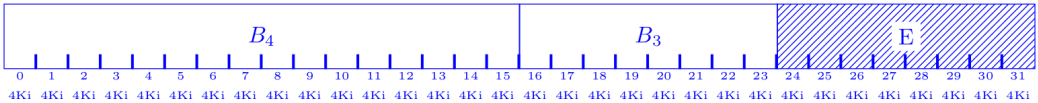
14. Nach `free(H)`: (Verschnitt 14575 Bytes)



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



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



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