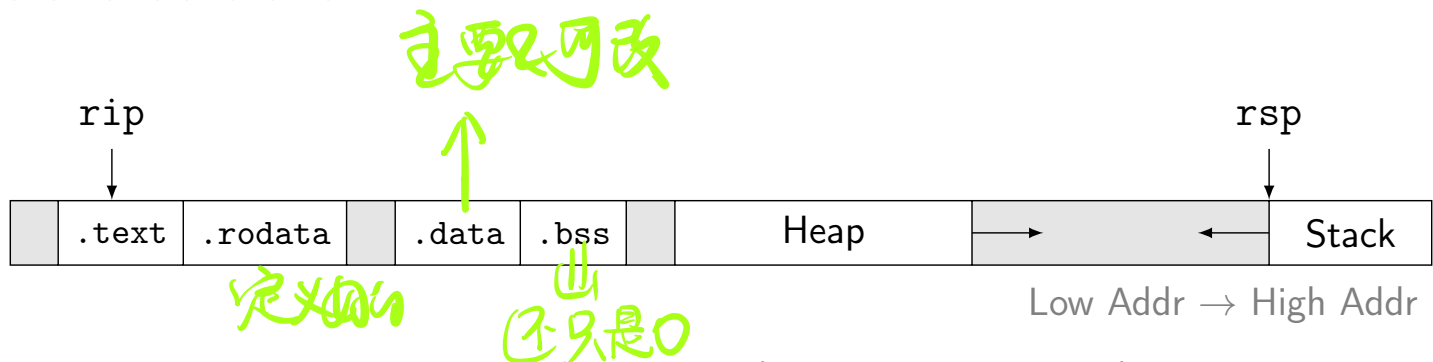


Speicherbereiche



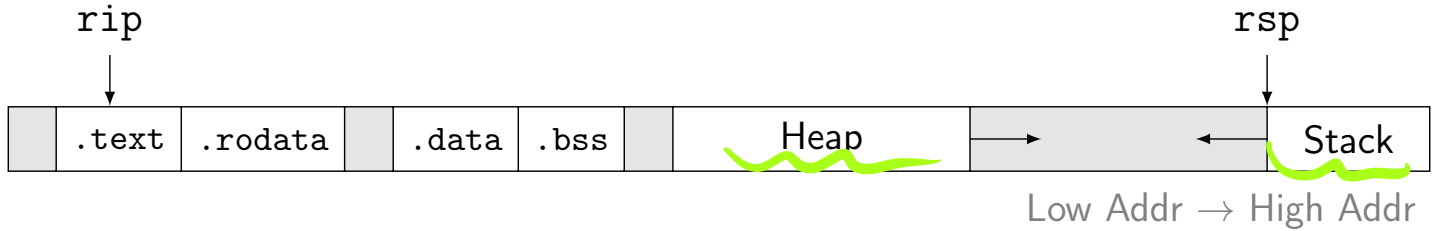
- ▶ `.text`: Beinhaltet den Programmcode (readonly, executable)
- ▶ `.rodata`: Beinhaltet globale konstante initialisierte Variablen (readonly)
 - ▶ Beispiel: `const int i = 42;` (global)
- ▶ `.data`: Beinhaltet globale initialisierte Variablen (read-write)
 - ▶ Beispiel: `int i = 42;` (global)
- ▶ `.bss`: Beinhaltet globale Variablen, die mit 0 initialisiert sind
 - ▶ Beispiel: `int i;` (global)

Speichersegmente in Assembler

In Assembler müssen die Segmente explizit gekennzeichnet werden

```
1 .data
2 my_data: .word 42
3
4 .text
5 my_fun:
6     ...
7     ret
```

Stack vs. Heap



Stack

- ▶ Für kleine Datenmengen
- ▶ Wächst von oben nach unten
- ▶ LIFO Prinzip
- ▶ Enthält lokale Variablen *本地变量*
- ▶ Automatische Speicherfreigabe

Heap

- ▶ Dynamische Allokation und Freigabe
- ▶ Für größere Datenmengen
- ▶ Allokationen global verwendbar
"Sichtbar"

Quiz: Speicherbereiche (1)

```
1 int v0 = 6;  
2 int v1;  
3 const int v2[4] = {1, 3, 3, 7};  
4  
5 int main(int argc, char** argv) {  
6     int v3 = 5;  
7     int v4[v3];  
8     const int v5 = 3;  
9     ...  
10 }
```

Wo liegt v0?

☐

.rodata

☐

Heap

☒

.data

Quiz: Speicherbereiche (2)

```
1 int v0 = 6;
2 int v1;
3 const int v2[4] = {1, 3, 3, 7};
4
5 int main(int argc, char** argv) {
6     int v3 = 5;
7     int v4[v3];
8     const int v5 = 3;
9     ...
10 }
```

Wo liegt v1?

☐

.rodata

☒

.bss

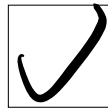
☐

.data

Quiz: Speicherbereiche (3)

```
1 int v0 = 6;  
2 int v1;  
3 const int v2[4] = {1, 3, 3, 7};  
4  
5 int main(int argc, char** argv) {  
6     int v3 = 5;  
7     int v4[v3];  
8     const int v5 = 3;  
9     ...  
10 }
```

Wo liegt v2?



.rodata



.bss



.data

Quiz: Speicherbereiche (4)

```
1 int v0 = 6;
2 int v1;
3 const int v2[4] = {1, 3, 3, 7};
4
5 int main(int argc, char** argv) {
6     int v3 = 5;
7     int v4[v3];
8     const int v5 = 3;
9     ...
10 }
```

Wo liegt v3?

☐

Heap

☒

Stack

☐

.data

Quiz: Speicherbereiche (5)

```
1 int v0 = 6;
2 int v1;
3 const int v2[4] = {1, 3, 3, 7};
4
5 int main(int argc, char** argv) {
6     int v3 = 5;
7     int v4[v3];
8     const int v5 = 3;
9     ...
10 }
```

Wo liegt v4?

☐

Heap

☒

Stack

☐

.bss

(lokal)

Quiz: Speicherbereiche (6)

```
1 int v0 = 6;
2 int v1;
3 const int v2[4] = {1, 3, 3, 7};
4
5 int main(int argc, char** argv) {
6     int v3 = 5;
7     int v4[v3];
8     const int v5 = 3;
9     ...
10 }
```

Wo liegt v5?



.rodata



Stack

Speicherverwaltung auf dem Heap

Speicherallokation:

- ▶ Funktionen aus stdlib.h:
 - ▶ `void* malloc(size_t size);`
 - ▶ `void* calloc(size_t nmemb, size_t size);`
- ▶ Im Falle eines Fehlers: NULL-Pointer Rückgabewert – *Immer überprüfen!*

Speicherfreigabe

- ▶ Kein Garbage Collector
- ▶ Funktion `void free(void* ptr)` aus `stdlib.h`
 - ▶ Nur *original* ptr von `malloc`, `calloc` etc. freen!
 - ▶ Nach `free`, ptr nicht mehr für Speicherzugriffe und -verwaltung verwenden!

Speicherverwaltung auf dem Heap: Beispiel

```
1 char* p = malloc(256 * sizeof(char));
2 if(p == NULL) {
3     // Behandlung von Fehler bei Speicherallokation
4     abort();
5 }
6 // ... arbeite mit p
7 free(p);
```

► Weitere Informationen in den Man-Pages:

- man 3 malloc
- man 3 calloc
- man 3 free

Quiz: Speicherbereiche (7)

```
1 #include <alloca.h>
2 #include <stdlib.h>
3
4 int main(int argc, char** argv) {
5     int* v6 = malloc (v3 * sizeof(int));
6     if (v6 == NULL) abort();
7     int* v7 = alloca(v3 * sizeof(int));
8     ...
9 }
```

Wo liegt v6?

☐

.data

☐

Stack

☒

Heap

Quiz: Speicherbereiche (8)

Wo liegt v7?

```
1 #include <alloca.h>
```

```
2 #include <stdlib.h>
```

```
3  
4 int main(int argc, char** argv) {
```

```
5     int* v6 = malloc (v3 * sizeof(int));
```

```
6     if (v6 == NULL) abort();
```

```
7     int* v7 = alloca(v3 * sizeof(int));
```

```
8     ...
```

```
9 }
```

int* v6 [3];

☐

.data

☒

Stack

☐

Heap

malloc: 在Heap, 要free
alloca: 在Stack, 不用free

Weitere Funktionen zur Speicherverwaltung auf dem Heap

`void* realloc(void* ptr, size_t size);`

- ▶ Alter Speicher bei Erfolg automatisch freigegeben
- ▶ Vergrößerung: Neue Daten uninitialisiert
- ▶ `realloc(NULL, size) $\hat{=}$ malloc(size)`
- ▶ NULL-Pointer Rückgabewert – Fehler!
 - ▶ Alter Speicherbereich wird nicht freigegeben

`void* aligned_alloc(size_t alignment, size_t size);`

- ▶ `alignment` muss Zweierpotenz sein
- ▶ `size` muss Vielfaches des `alignments` sein