Buffer-Overflows

Johannes Weiß

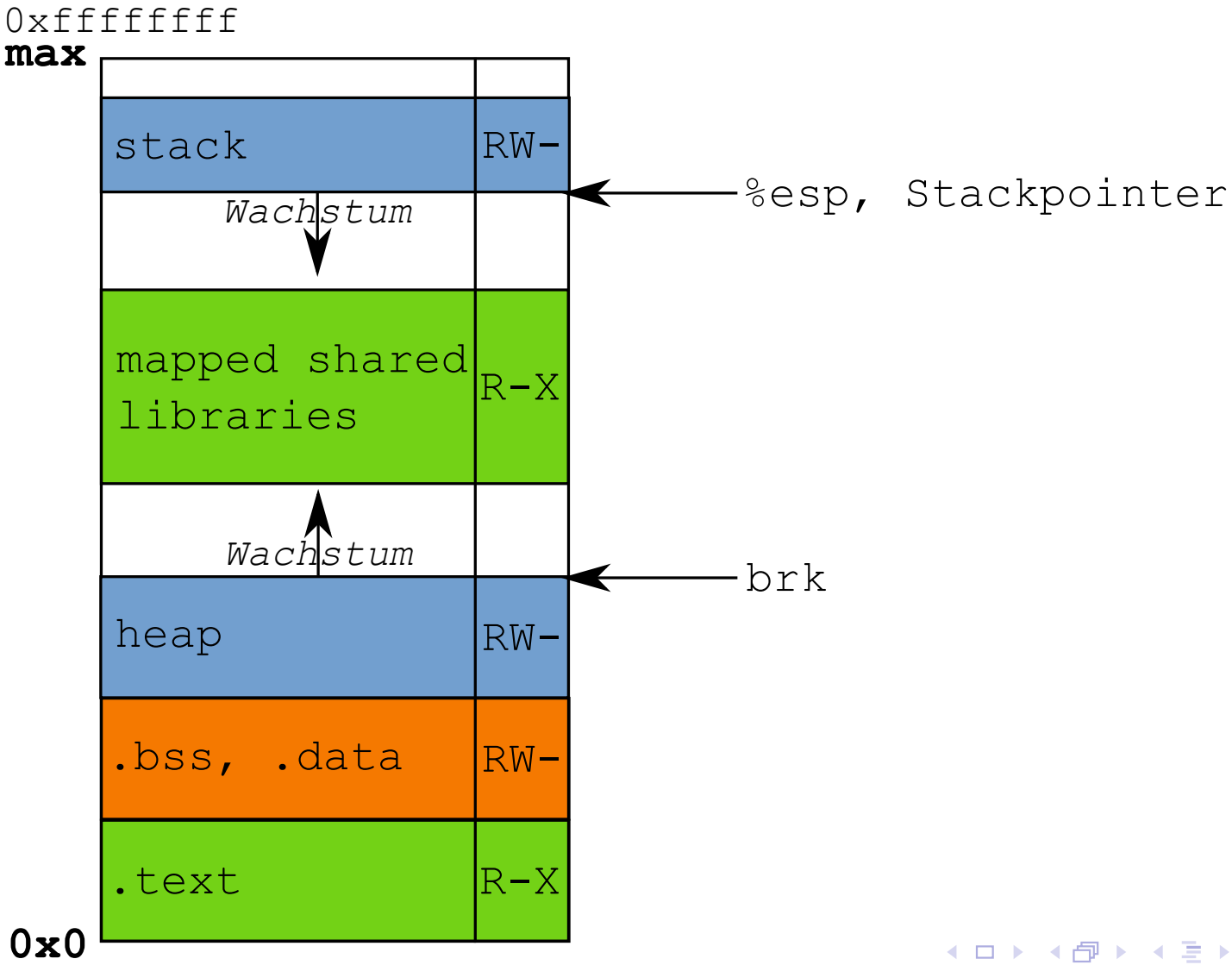
Januar 2010

Übersicht

Übersicht der Vortragsthemen

- Grundlagen
- Klassische Angriffe
- Automatische Schutzmaßnahmen des Betriebssystems
- Überwinden dieser automatischen Mechanismen
- Fazit, Fragen & Diskussion

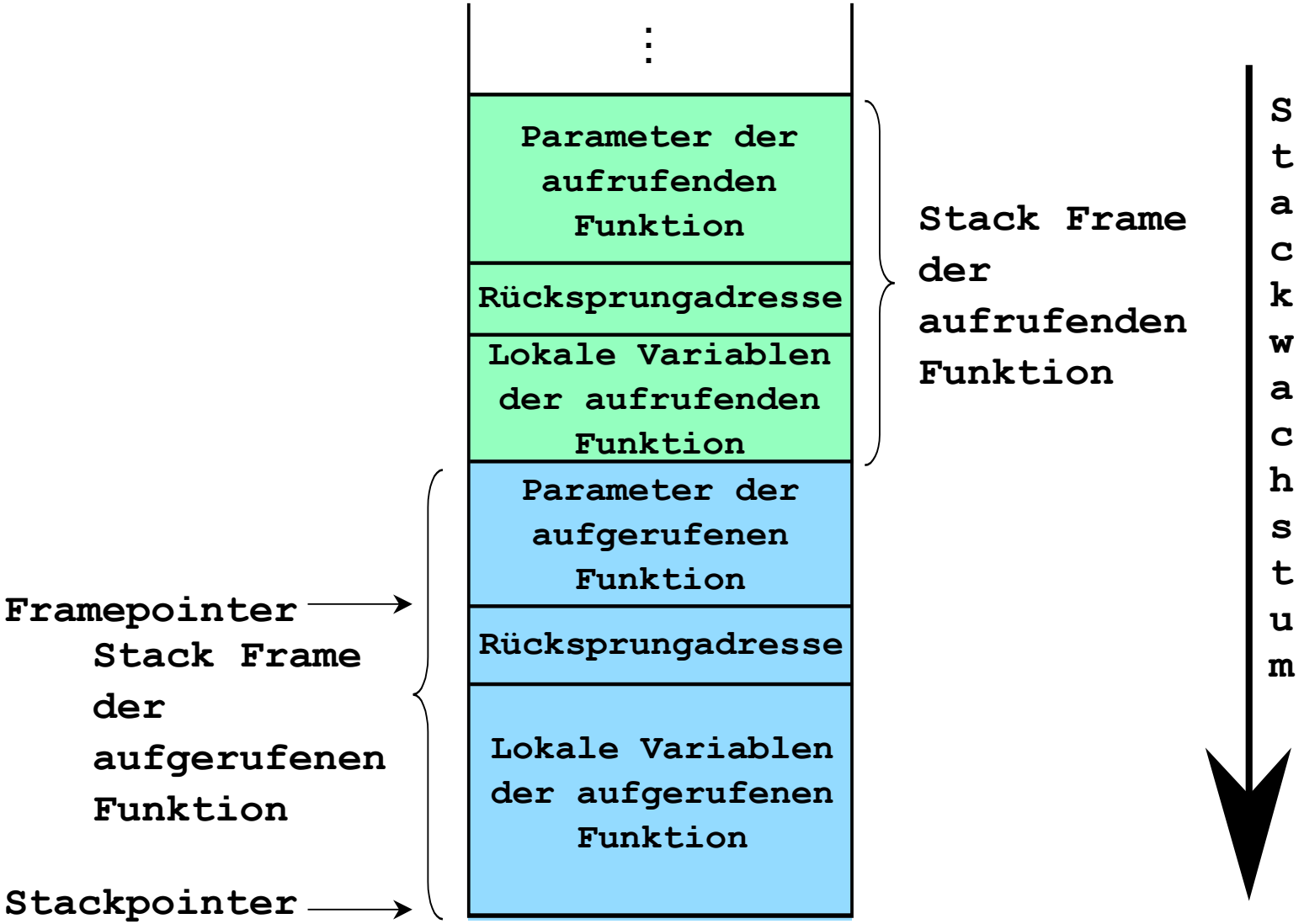
Memory-Layout unter UNIX



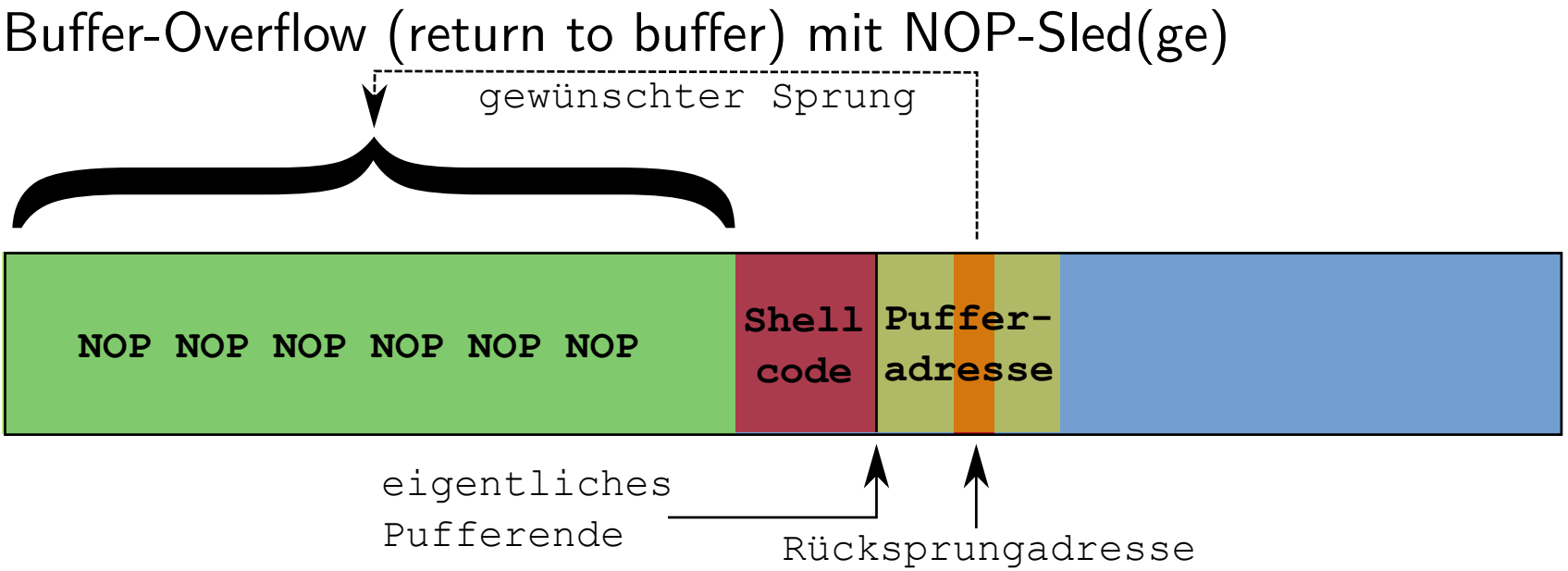
Wichtige Maschinenbefehle

- `NOP`: *No Operation*: Nichts tun
- `POP %reg`: Nimmt ein Wort vom Stack, inkrementiert den Stackpointer und speichert es im Register `%reg`
- `RET`: `POP`t eine Adresse und springt dort hin.
  Entspricht `POP %reg; JMP %reg`.

Stack beim Funktionsaufruf



Klassische Angriffe auf den Stack



Verhindern von klassischen Angriffen

Nichtausführbarkeitsregeln (NX-Bit)

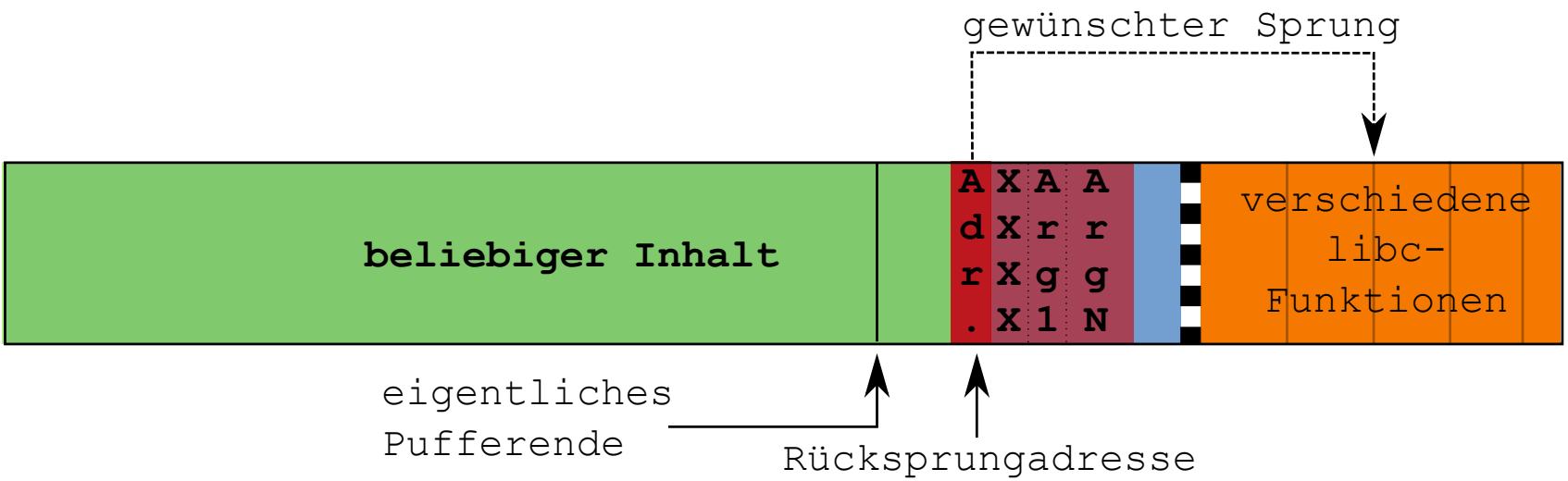
- *Schreibbare* Speicherbereiche als *nicht ausführbar* markieren
- Dadurch kann kein fremder Code mehr eingeschleust werden

Address Space Layout Randomization (ASLR)

- Einblenden mindestens des Stacks an *zufälligen Adressen*
- Dadurch kann die Pufferadresse kaum noch genau genug erraten werden
- Meistens sind nicht alle Bereiche zufällig eingeblendet. Vor allem der Code des Programms an sich ist meistens statisch eingeblendet.

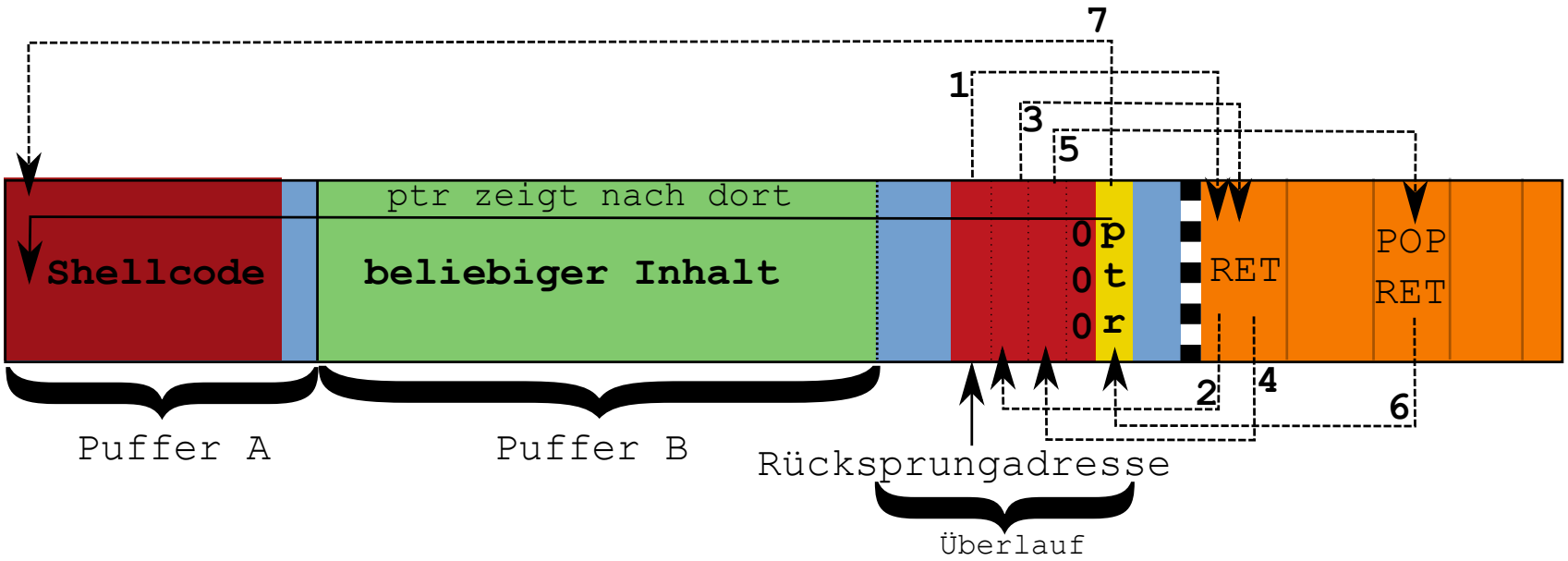
return-into-libc-Exploits

- *Statische Adressen*
- keine *ausführbaren und schreibbaren* Speicherbereiche



ret2pop-Exploits

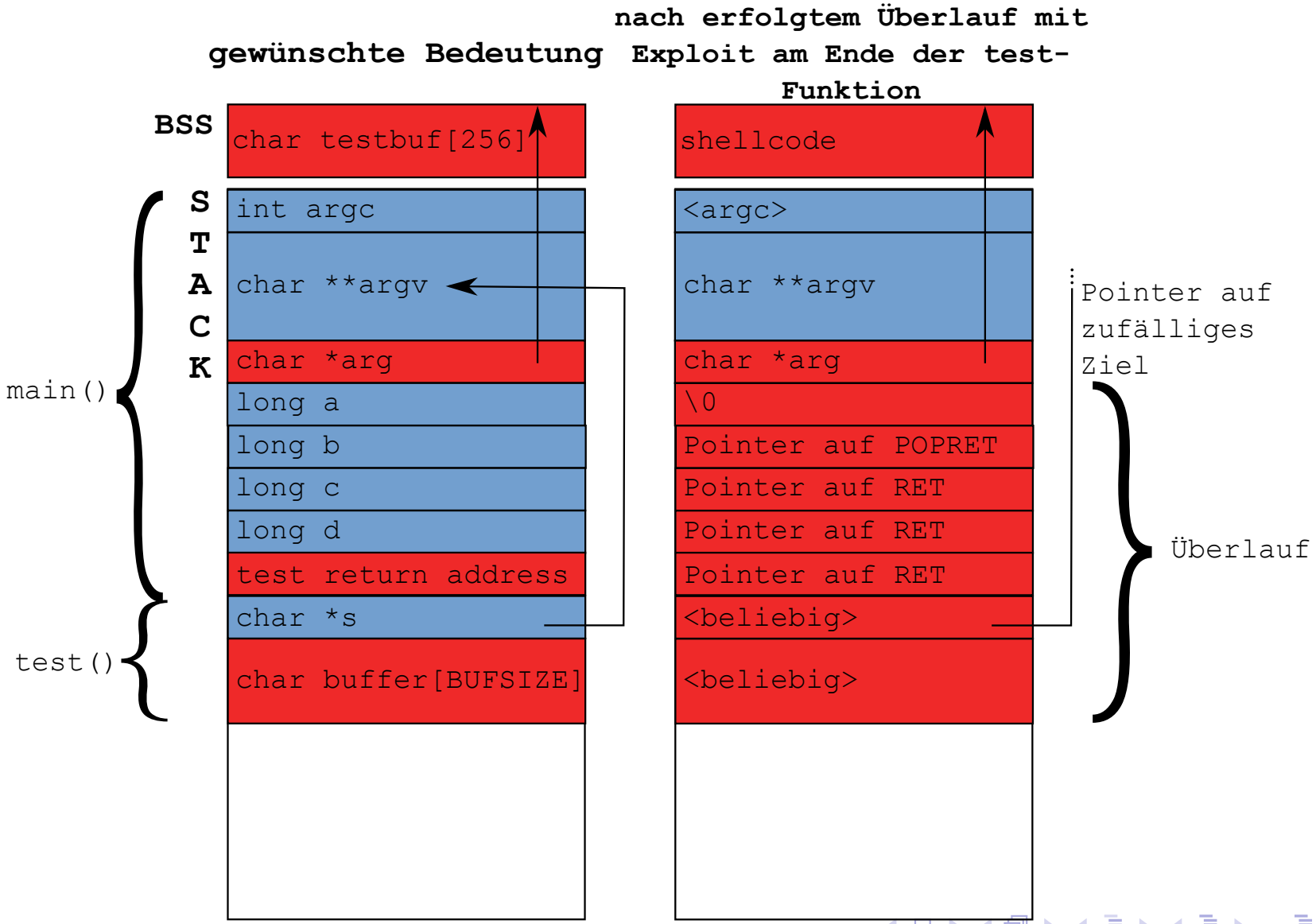
- Stack, Heap und dynamischen Bibliotheken an zufälligen Adressen eingeblendet
- Code des Programms ist statisch eingeblendet
- Jeder *lesbare* Speicherbereich ist auch *ausführbar* (Standard-Situation bei x86 mit 32bit)



Beispiel: ret2pop-Exploit: Code

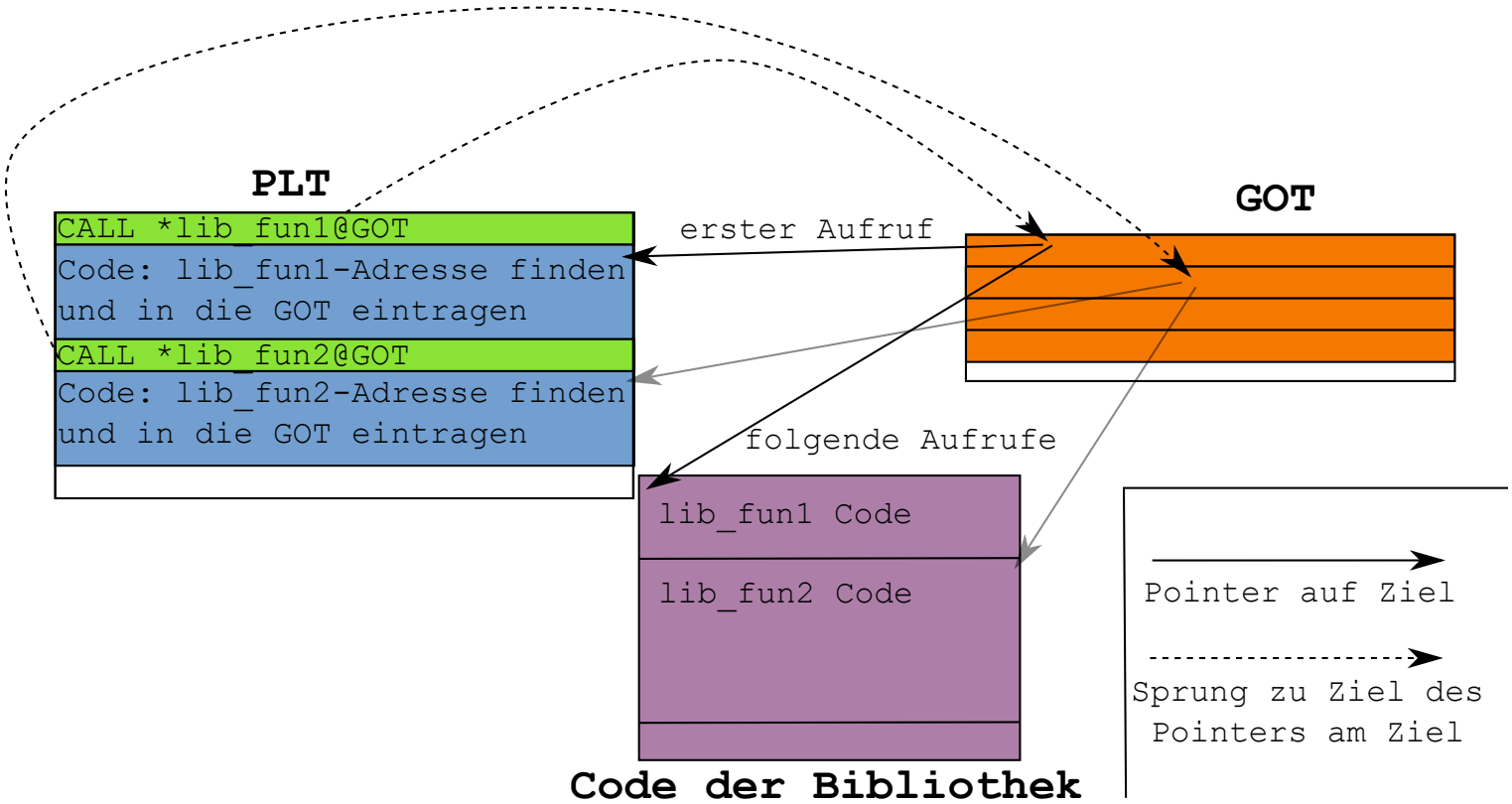
```
 1 char testbuf[256];
 2
 3 void test(char *s) {
 4     char buffer[BUFSIZE];
 5     memset(buffer, 0, BUFSIZE);
 6     strcpy(buffer, s);
 7 }
 8
 9 int main(int argc, char **argv) {
10     char *arg = testbuf;
11     long a = 0; long b = 1; long c = 2; long d = 3;
12     strncpy(testbuf, argv[1], 255);
13     test(argv[2]);
14     return 0;
15 }
```

Beispiel: ret2pop-Exploit: Speicher



Einführung in PLT und GOT

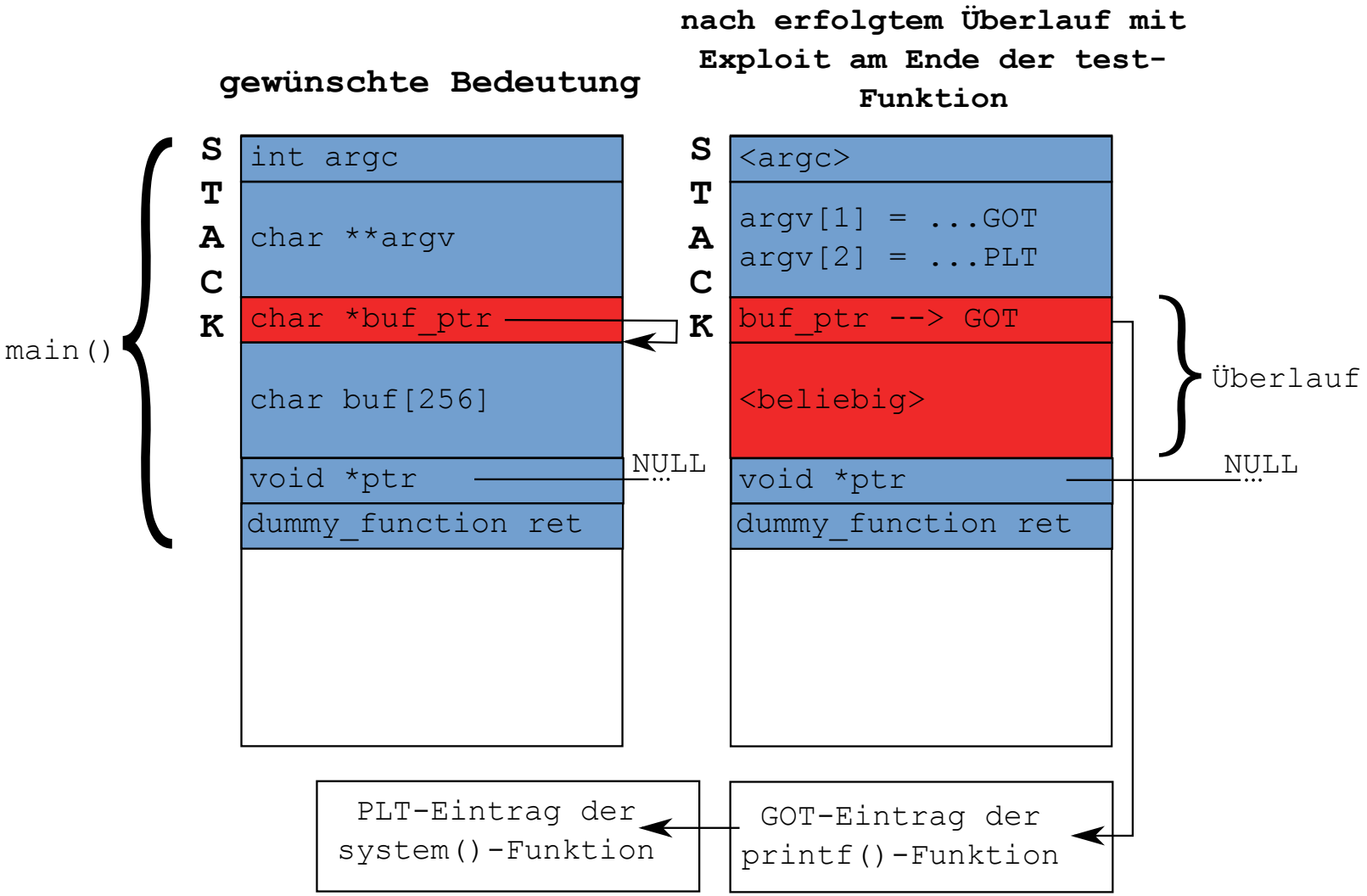
- Programmcode ist an *statischer Adresse* eingeblendet
- Alles andere an *zufälligen Adressen*
- Jeder *schreibbare* Speicherbereich ist *nicht ausführbar*.



Beispiel: ret2got-Exploit: Code

```
 1 int main(int argc, char **argv) {
 2     char *buf_ptr = 0;
 3     char buf[256];
 4     void *ptr;
 5     buf_ptr = buf;
 6     printf("Hello World! %d\n", 007);
 7     //buf_ptr in die GOT umbiegen
 8     strcpy(buf_ptr, argv[1]);
 9     //Ziel von buf_ptr beschreiben
10     strcpy(buf_ptr, argv[2]);
11     printf("You told me %s and %s!\n",
12             argv[1], argv[2]);
13     printf("Goodbye, World!\n");
14     return 0;
15 }
```

Beispiel: ret2got-Exploit: Speicher



Fazit

- In Sprachen wie Java, Python, Haskell, C#, ... bestehen die hier vorgestellten Probleme nicht, da dort keine Buffer-Overflows möglich sind
- ASLR, NX-Bit, StackProtection machen das Ausnutzen von Sicherheitslücken deutlich schwieriger, aber nicht unmöglich
- Fragen?