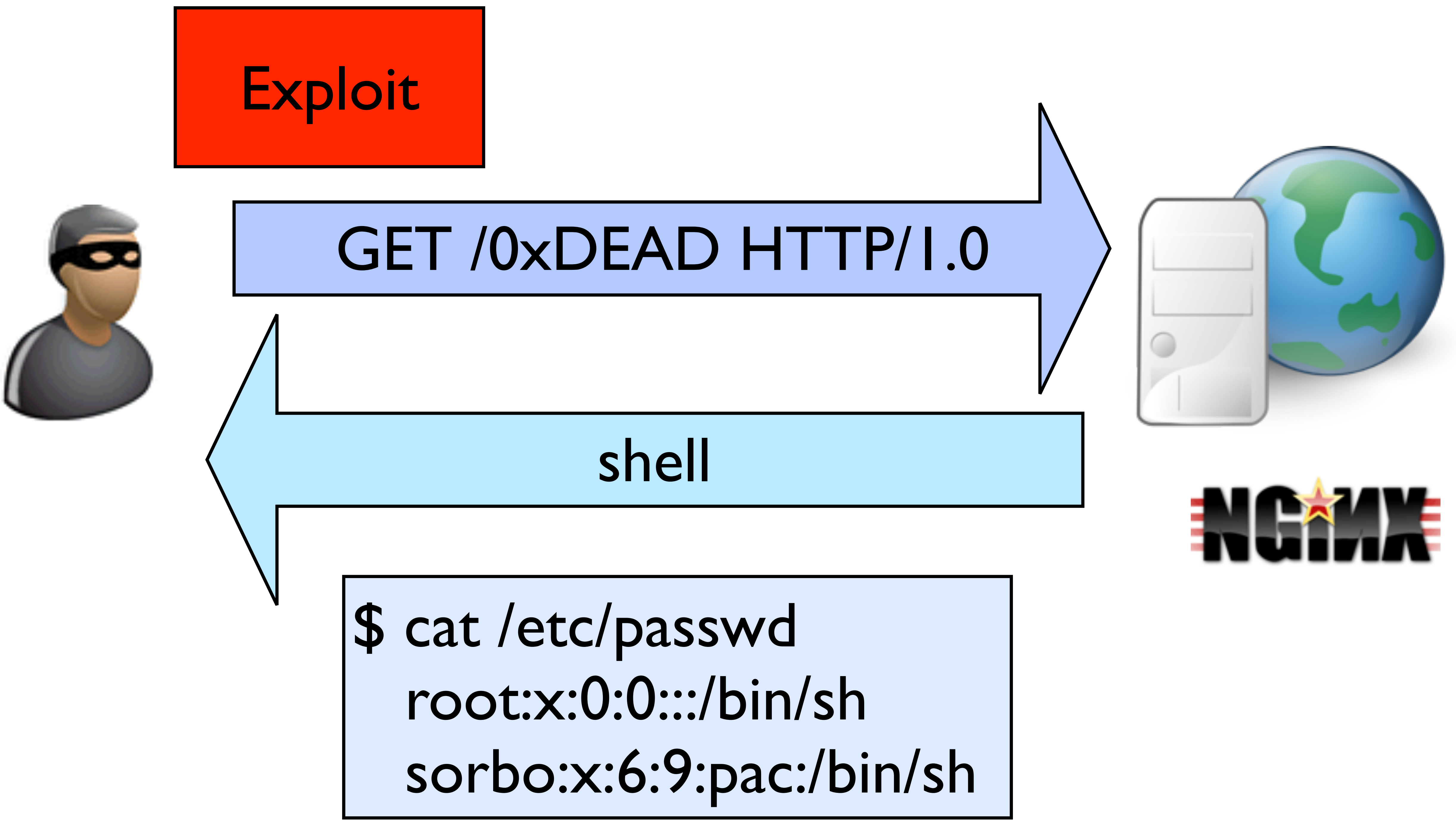
Hacking Blind

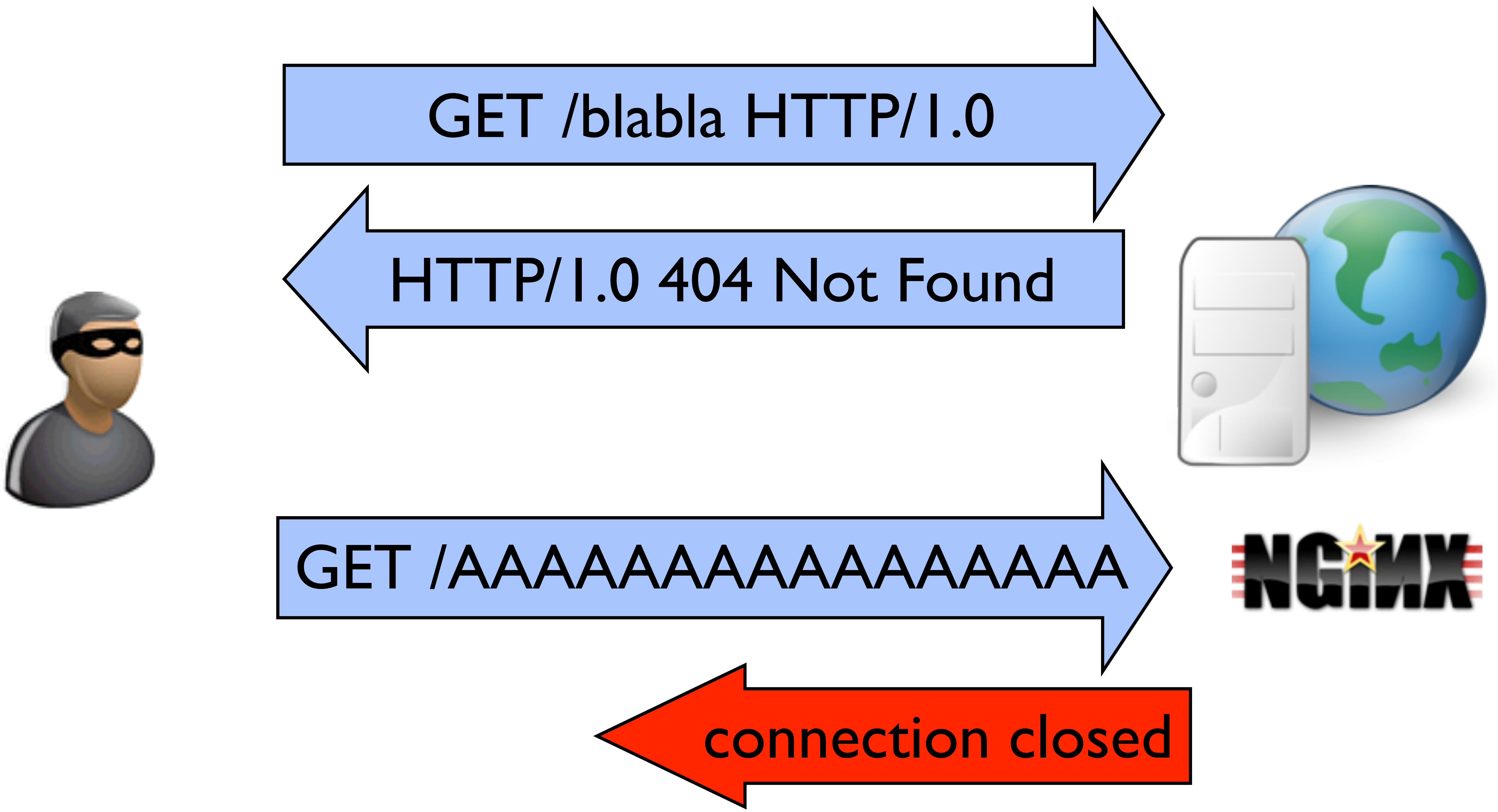
Andrea Bittau, Adam Belay, Ali Mashtizadeh,
David Mazières, Dan Boneh

Stanford University

Hacking buffer overflows



Crash or no Crash?
Enough to build exploit



Don’t even need to know what application is running!

Exploit scenarios:

1. Open source
2. Open binary
3. Closed-binary (and source) ????

Attack effectiveness

- Works on 64-bit Linux with ASLR, NX and canaries

Server	Requests	Time (mins)
nginx	2,401	1
MySQL	3,851	20
Toy proprietary service (unknown binary and source)	1,950	5

Attack requirements

1. Stack vulnerability, and knowledge of how to trigger it.
2. Server process that respawns after crash
   - E.g., nginx, MySQL, Apache, OpenSSH, Samba.

Outline

- Introduction.
- Background on exploits.
- Blind ROP (BROP).
- Optimizations.

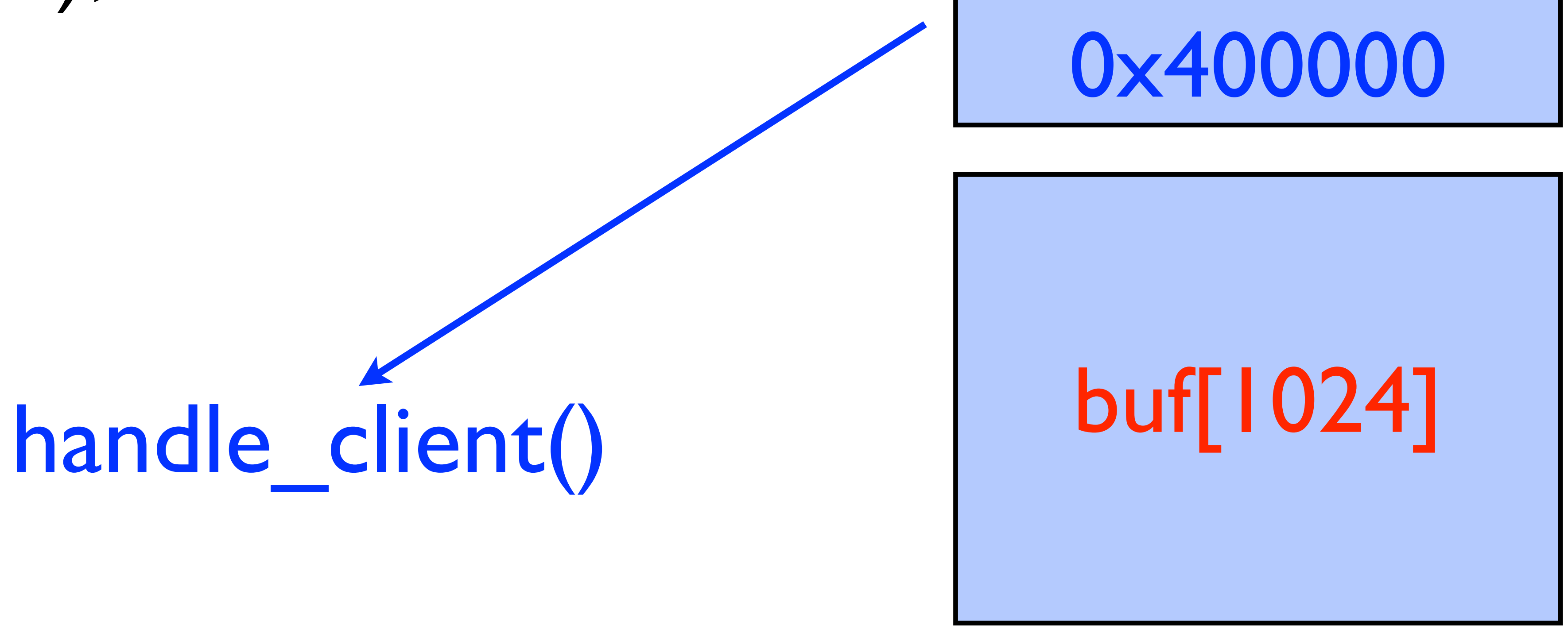
Stack vulnerabilities

```
void process_packet(int s) {
    char buf[1024];
    int len;

    read(s, &len, sizeof(len));
    read(s, buf, len);

    return;
}
```

Stack:



Stack vulnerabilities

```
void process_packet(int s) {
    char buf[1024];
    int len;

    read(s, &len, sizeof(len));
    read(s, buf, len);

    return;
}
```

Stack:

return address 0x400000
AAAAAAAAA AAAAAAAAA AAAAAAAAA AAAAAAAAA

handle_client()

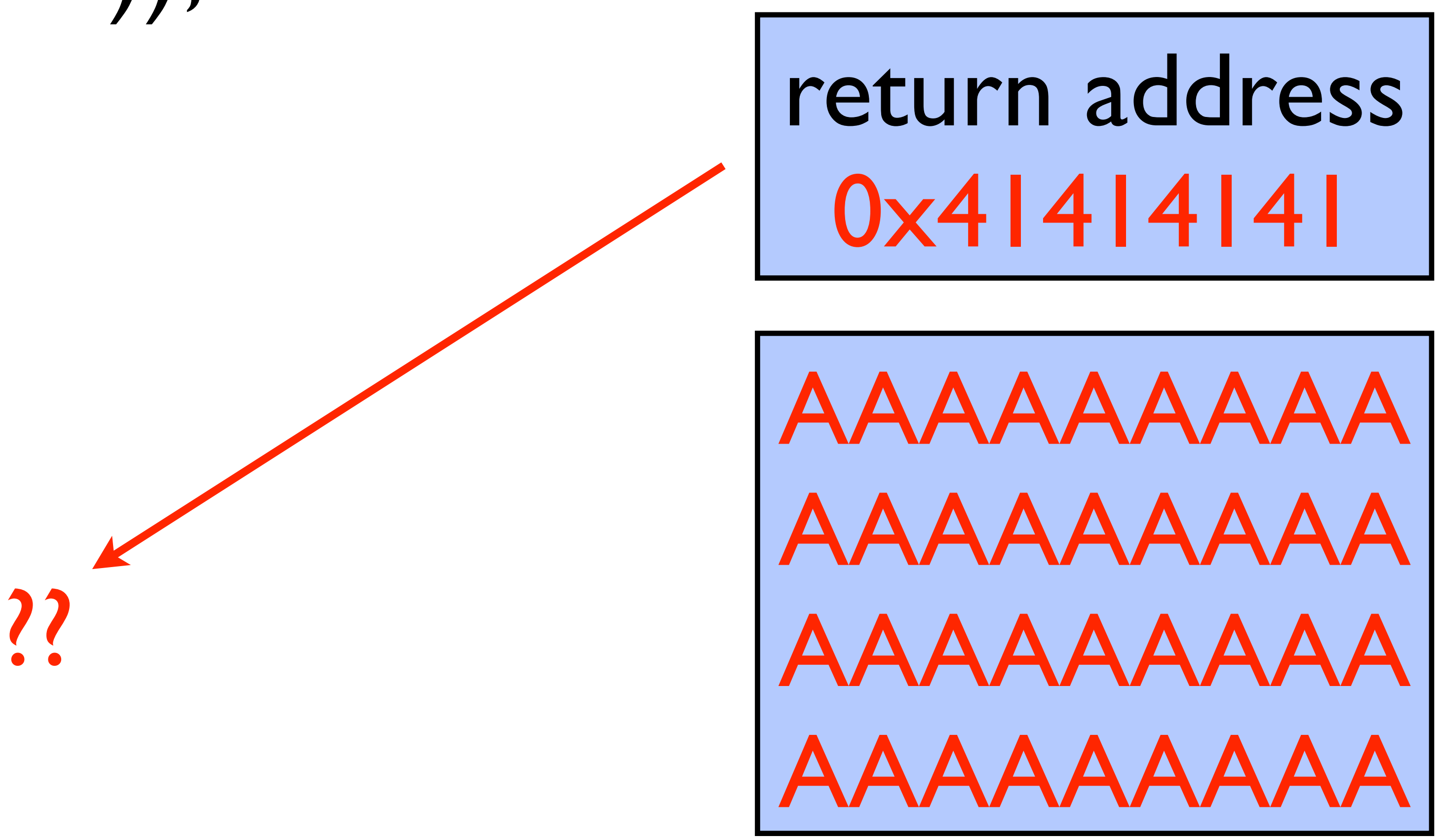
Stack vulnerabilities

```
void process_packet(int s) {
    char buf[1024];
    int len;

    read(s, &len, sizeof(len));
    read(s, buf, len);

    return;
}
```

Stack:



Stack vulnerabilities

```
void process_packet(int s) {
    char buf[1024];
    int len;

    read(s, &len, sizeof(len));
    read(s, buf, len);

    return;
}
```

Stack:

return address 0x500000
AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA

Shellcode:

```
dup2(sock, 0);
dup2(sock, 1);
execve("/bin/sh", 0, 0);
```

Stack vulnerabilities

```
void process_packet(int s) {
    char buf[1024];
    int len;

    read(s, &len, sizeof(len));
    read(s, buf, len);

    return;
}
```

Stack:

return address
0x600000

0x1029827189
123781923719
823719287319
879181823828

Shellcode:

```
dup2(sock, 0);
dup2(sock, 1);
execve("/bin/sh", 0, 0);
```

Exploit protections

```
void pro
    char
    int le

    rea
    rea

    return;
}
```

1. Make stack non-executable (NX)

2. Randomize memory addresses (ASLR)

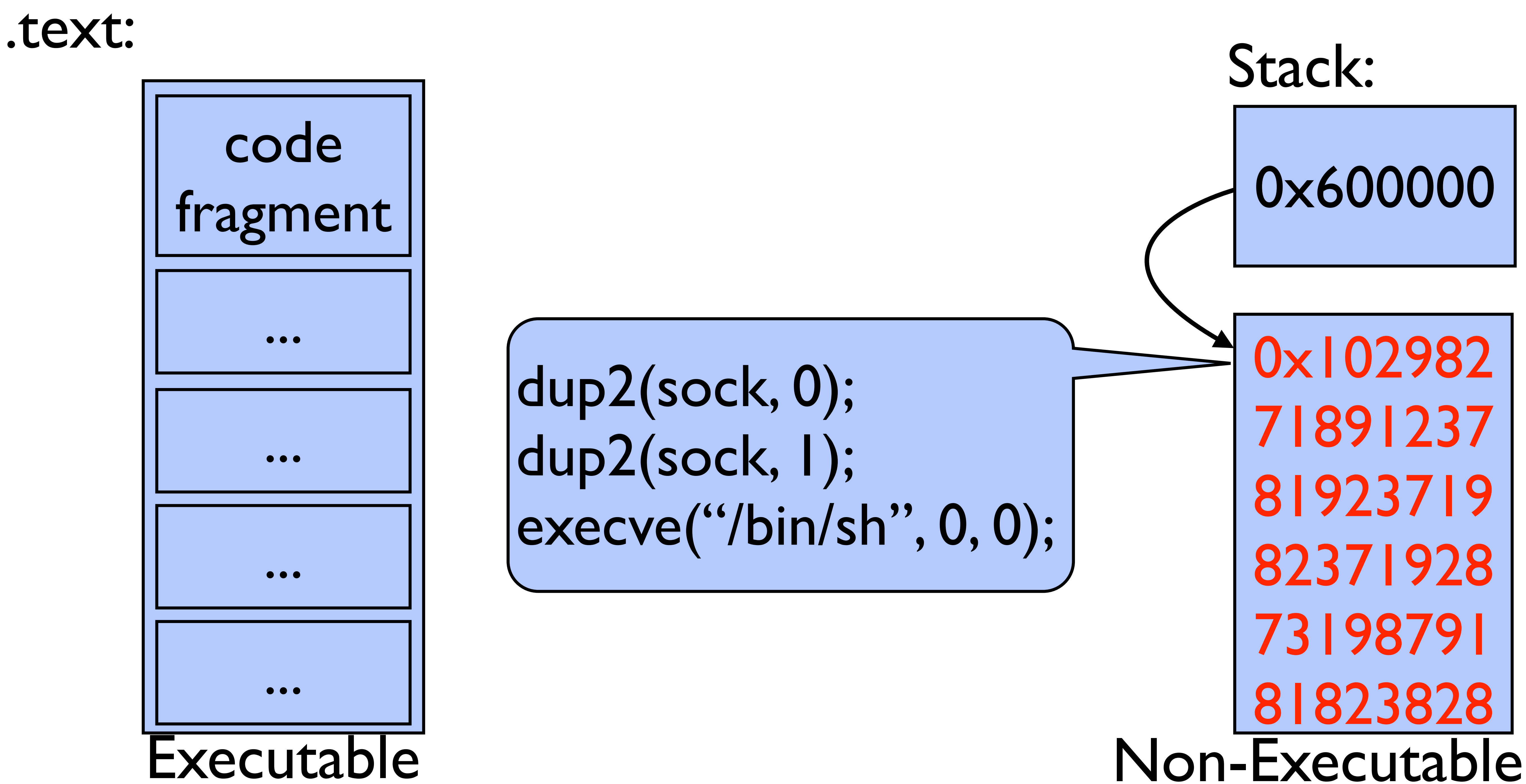
Stack:

return address 0x600000
0x1029827189 123781923719 823719287319 879181823828

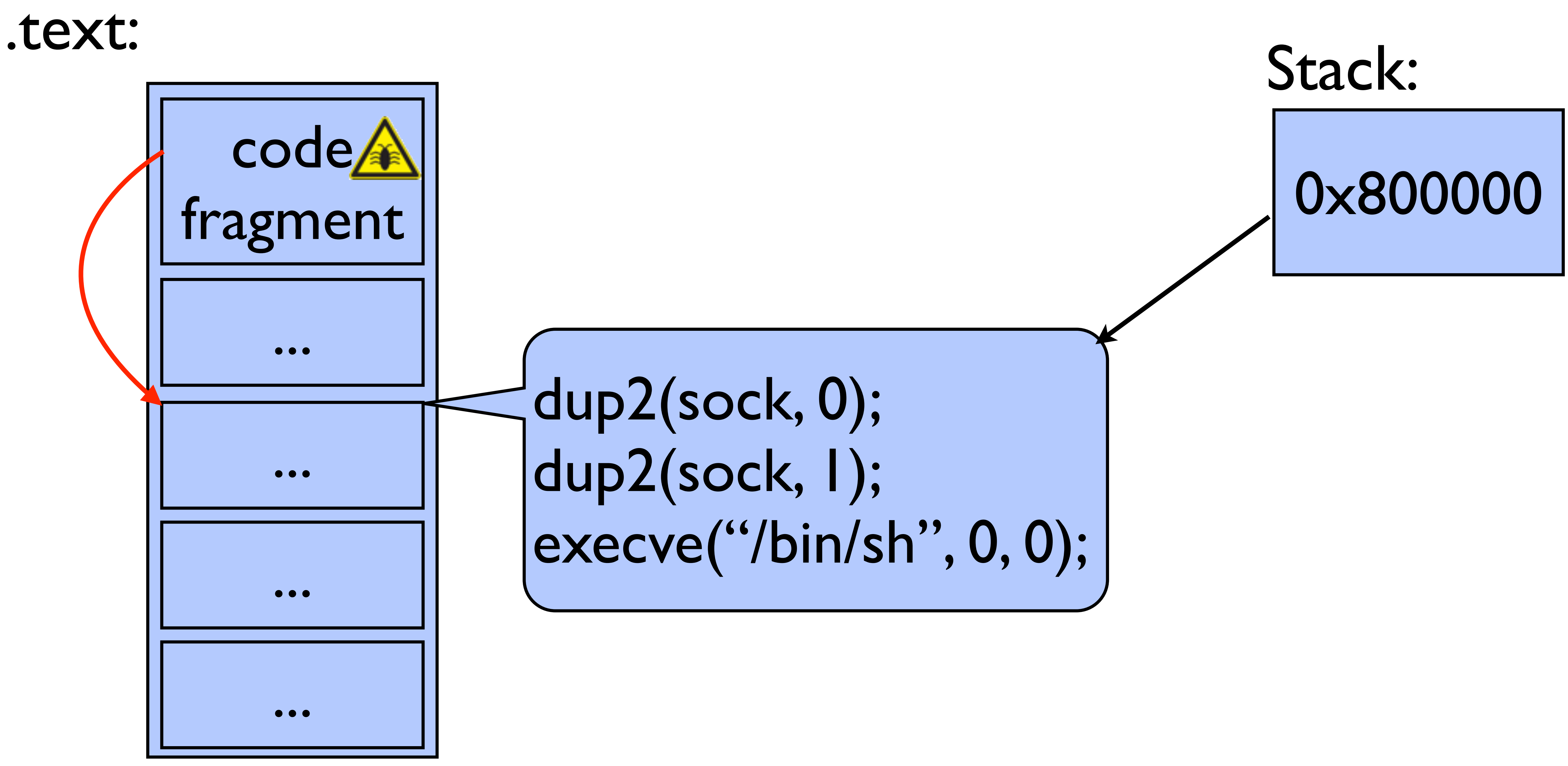
Shellcode:

```
dup2(sock, 0);
dup2(sock, 1);
execve("/bin/sh", 0, 0);
```

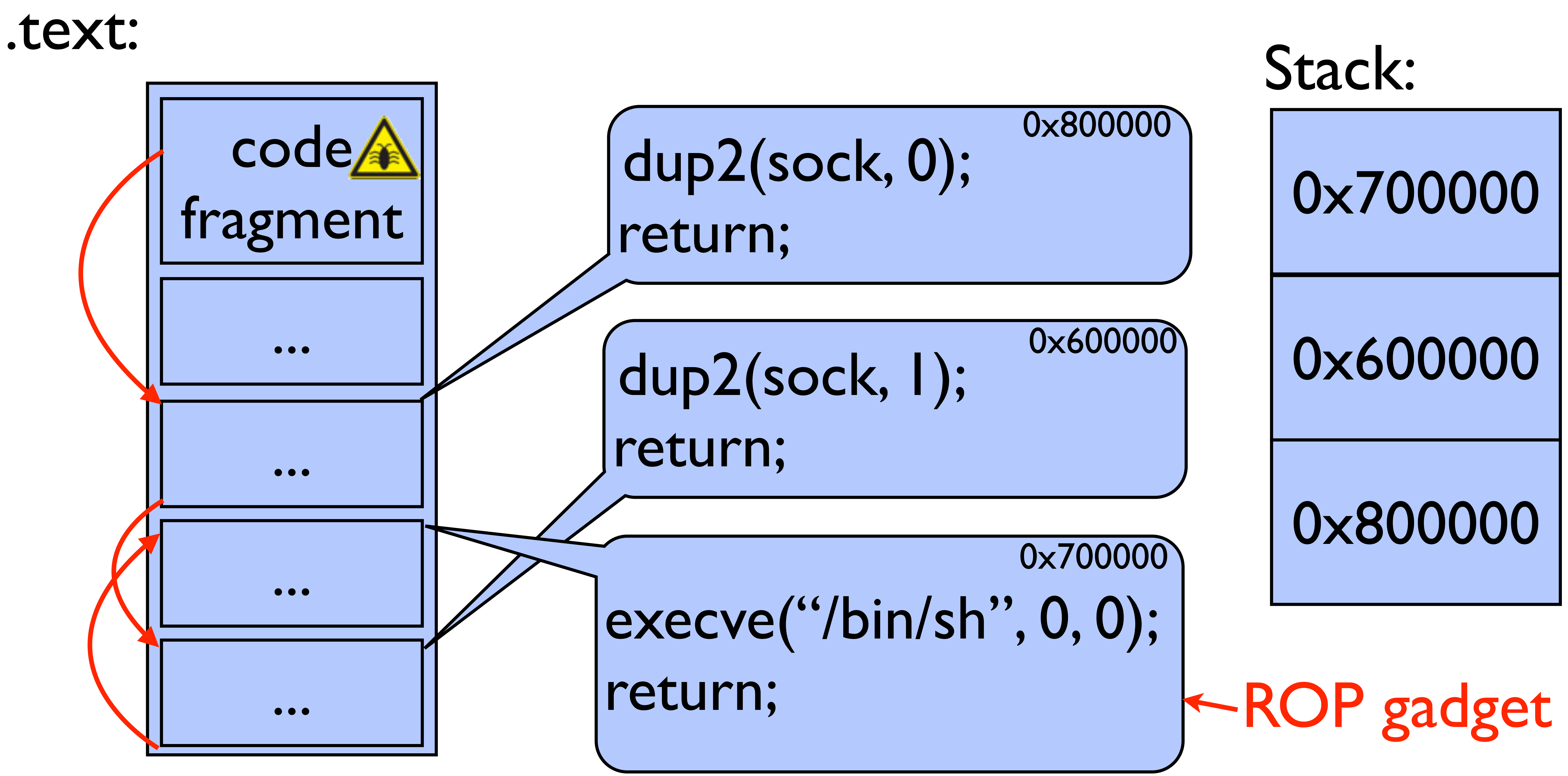
Return-Oriented Programming (ROP)



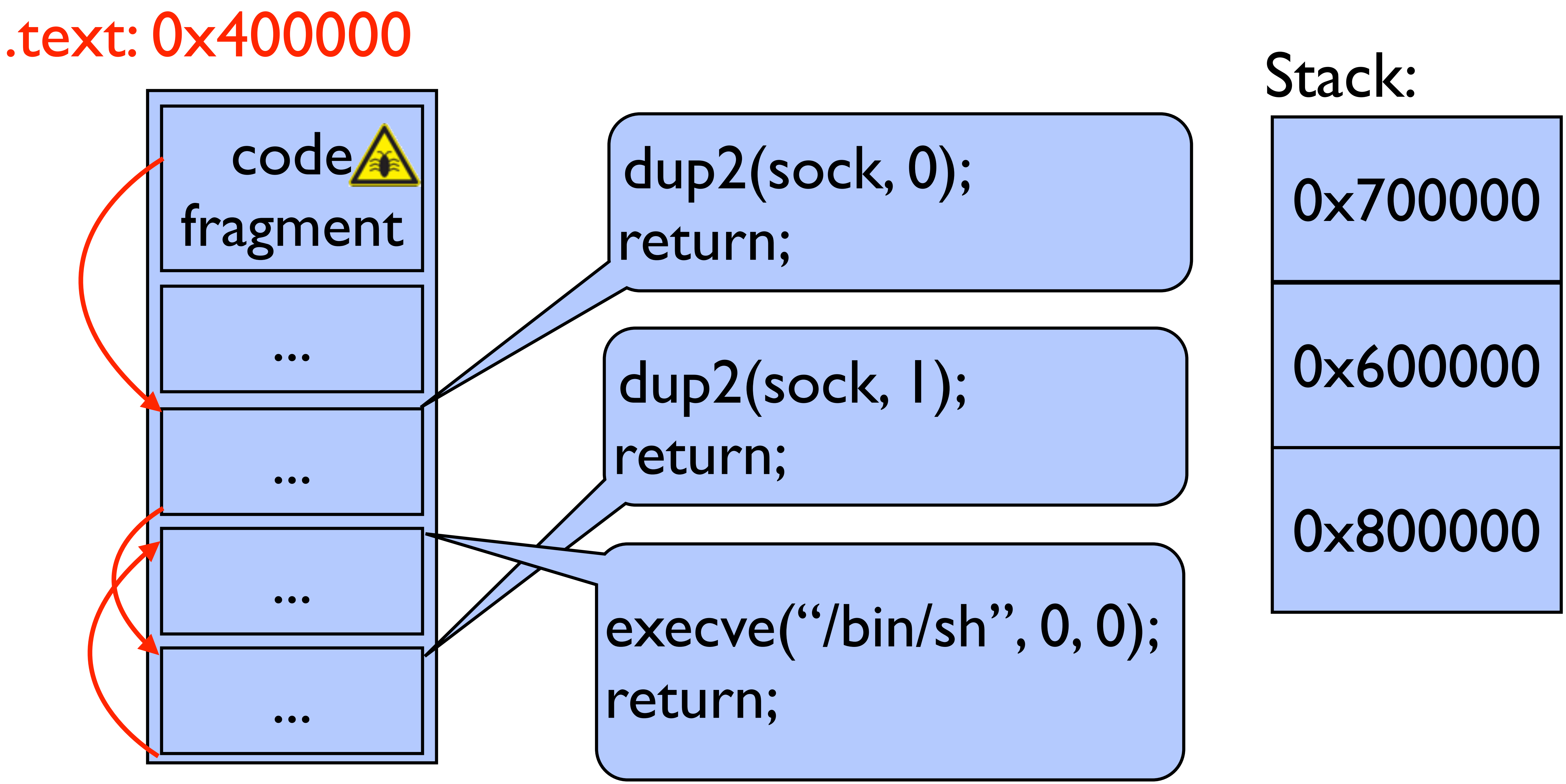
Return-Oriented Programming (ROP)



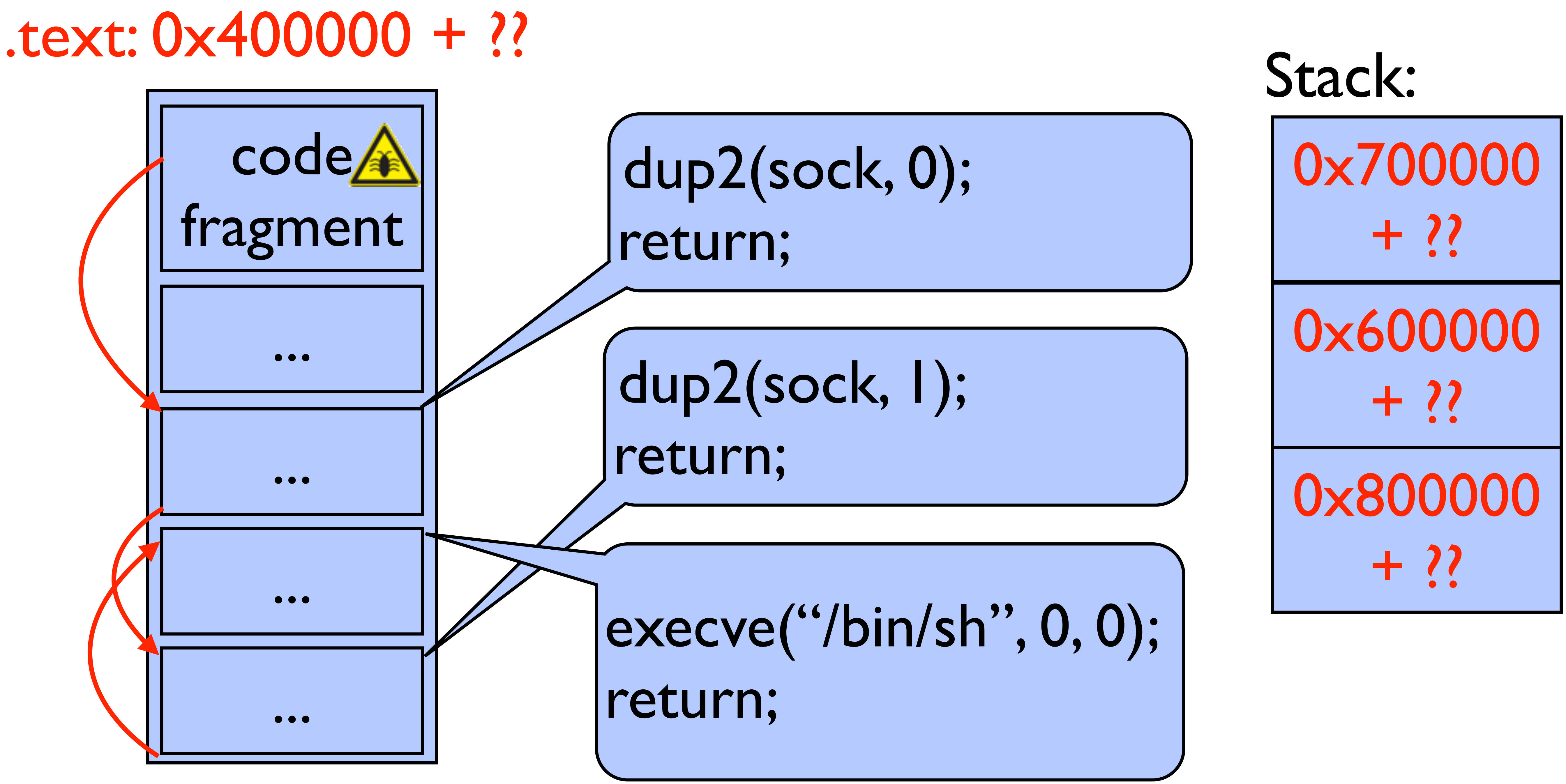
Return-Oriented Programming (ROP)



Address Space Layout Randomization (ASLR)



Address Space Layout Randomization (ASLR)



Exploit requirements today

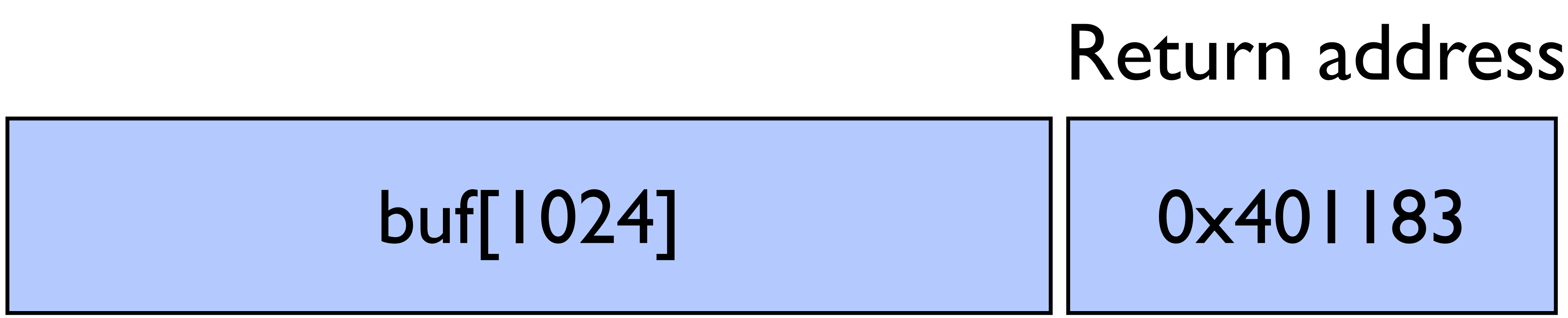
1. Break ASLR.
2. Copy of binary (find ROP gadgets / break NX).
   - Is it even possible to hack unknown applications?

Blind Return-Oriented Programming (BROP)

1. Break ASLR.
2. Leak binary:
    - Remotely find enough gadgets to call write().
    - write() binary from memory to network to disassemble and find more gadgets to finish off exploit.

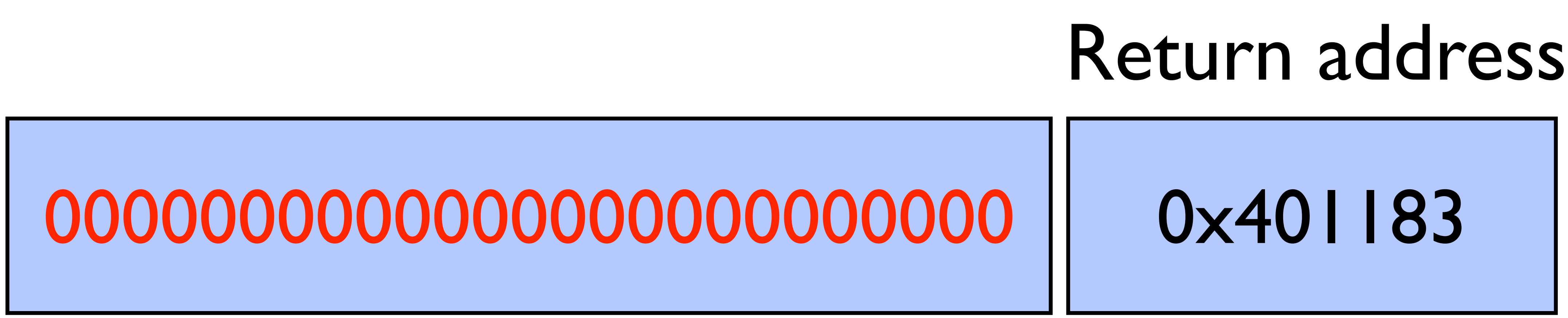
Defeating ASLR: stack reading

- Overwrite a single byte with value X:
  - No crash: stack had value X.
  - Crash: guess X was incorrect.
- Known technique for leaking canaries.



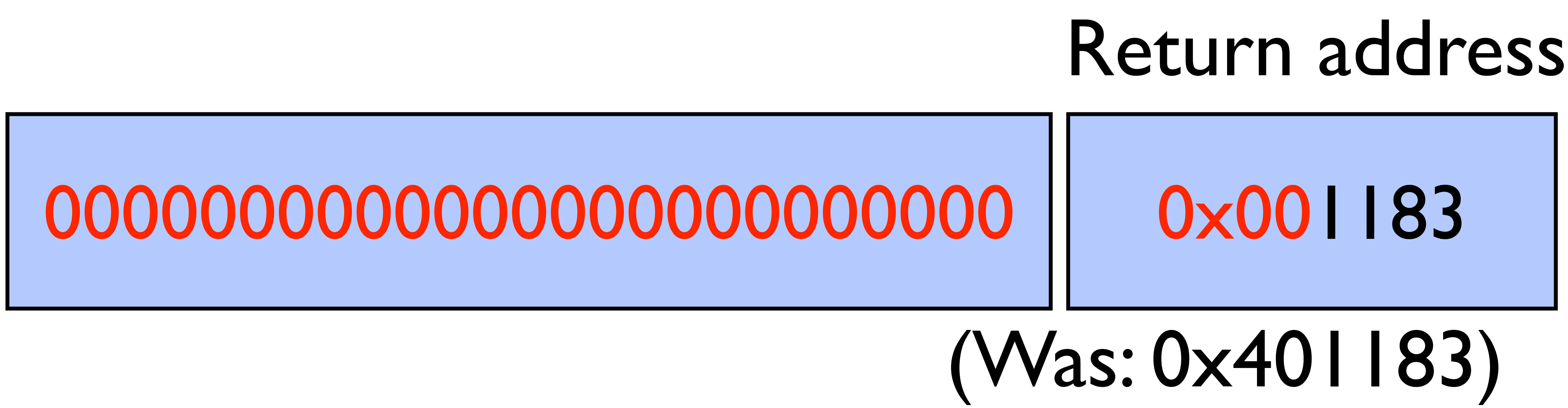
Defeating ASLR: stack reading

- Overwrite a single byte with value X:
  - No crash: stack had value X.
  - Crash: guess X was incorrect.
- Known technique for leaking canaries.



Defeating ASLR: stack reading

- Overwrite a single byte with value X:
  - No crash: stack had value X.
  - Crash: guess X was incorrect.
- Known technique for leaking canaries.



Defeating ASLR: stack reading

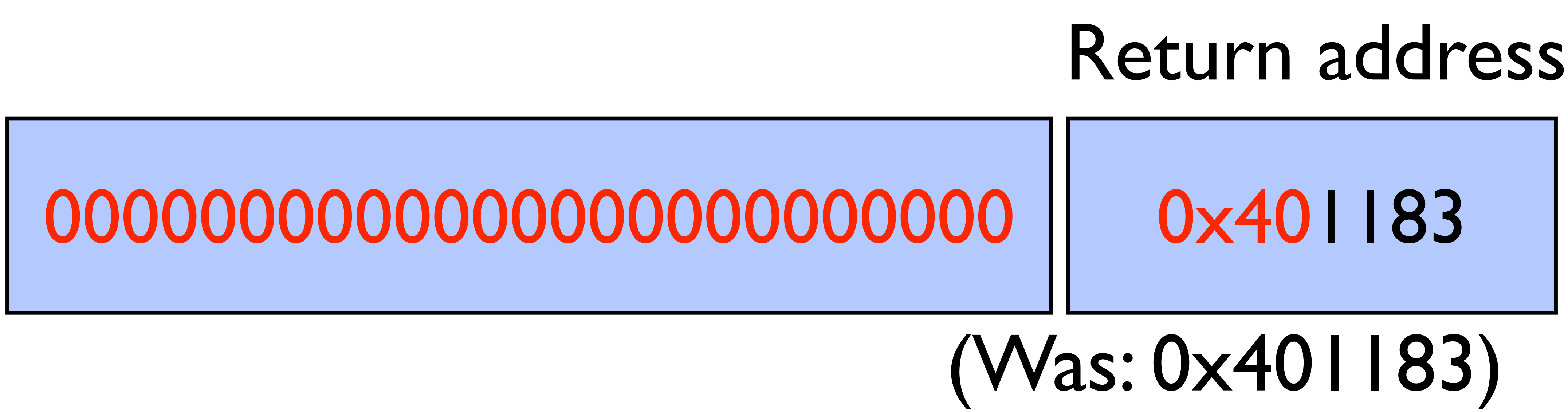
- Overwrite a single byte with value X:
  - No crash: stack had value X.
  - Crash: guess X was incorrect.
- Known technique for leaking canaries.

	Return address
0000000000000000000000000	0x011183

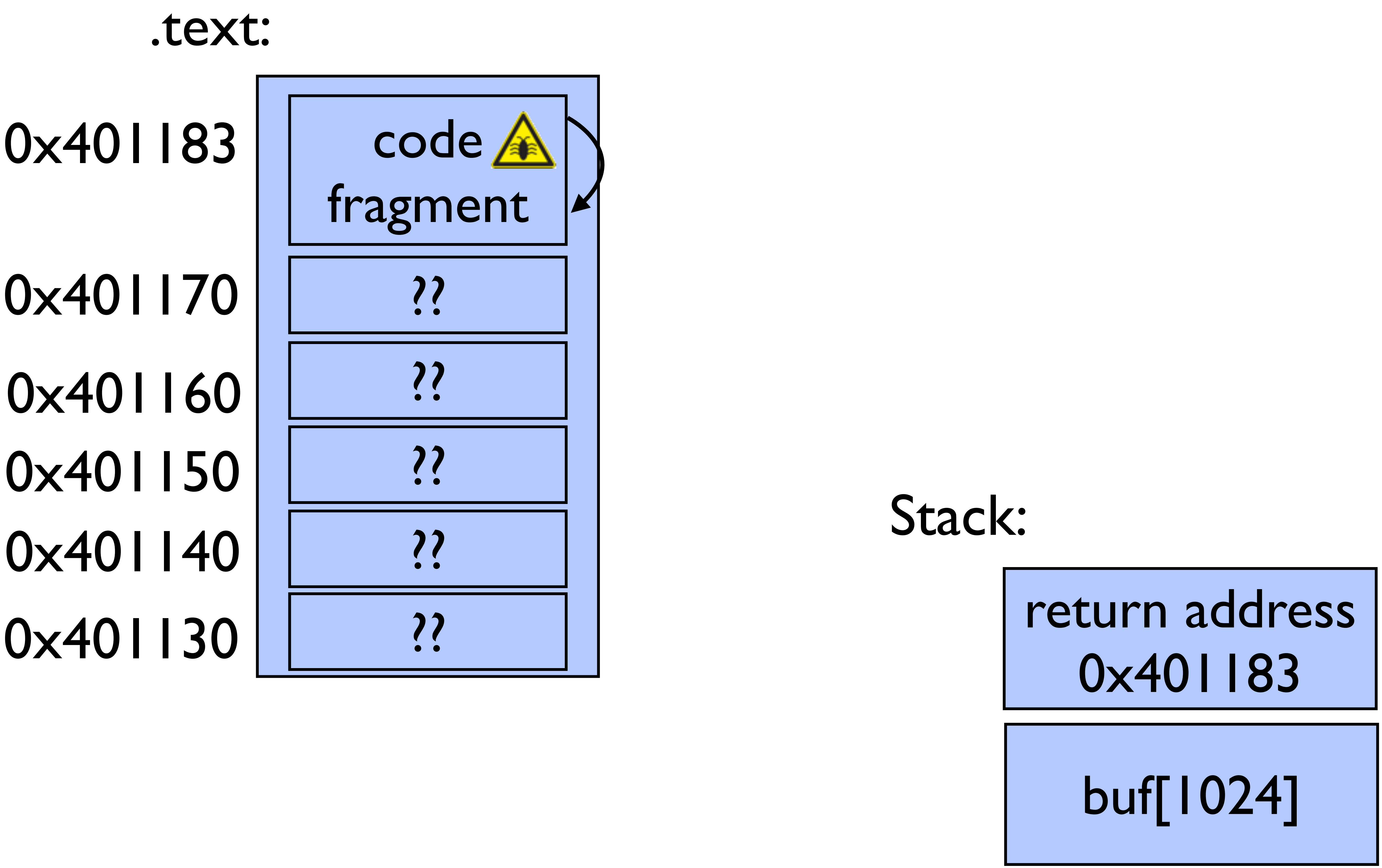
(Was: 0x401183)

Defeating ASLR: stack reading

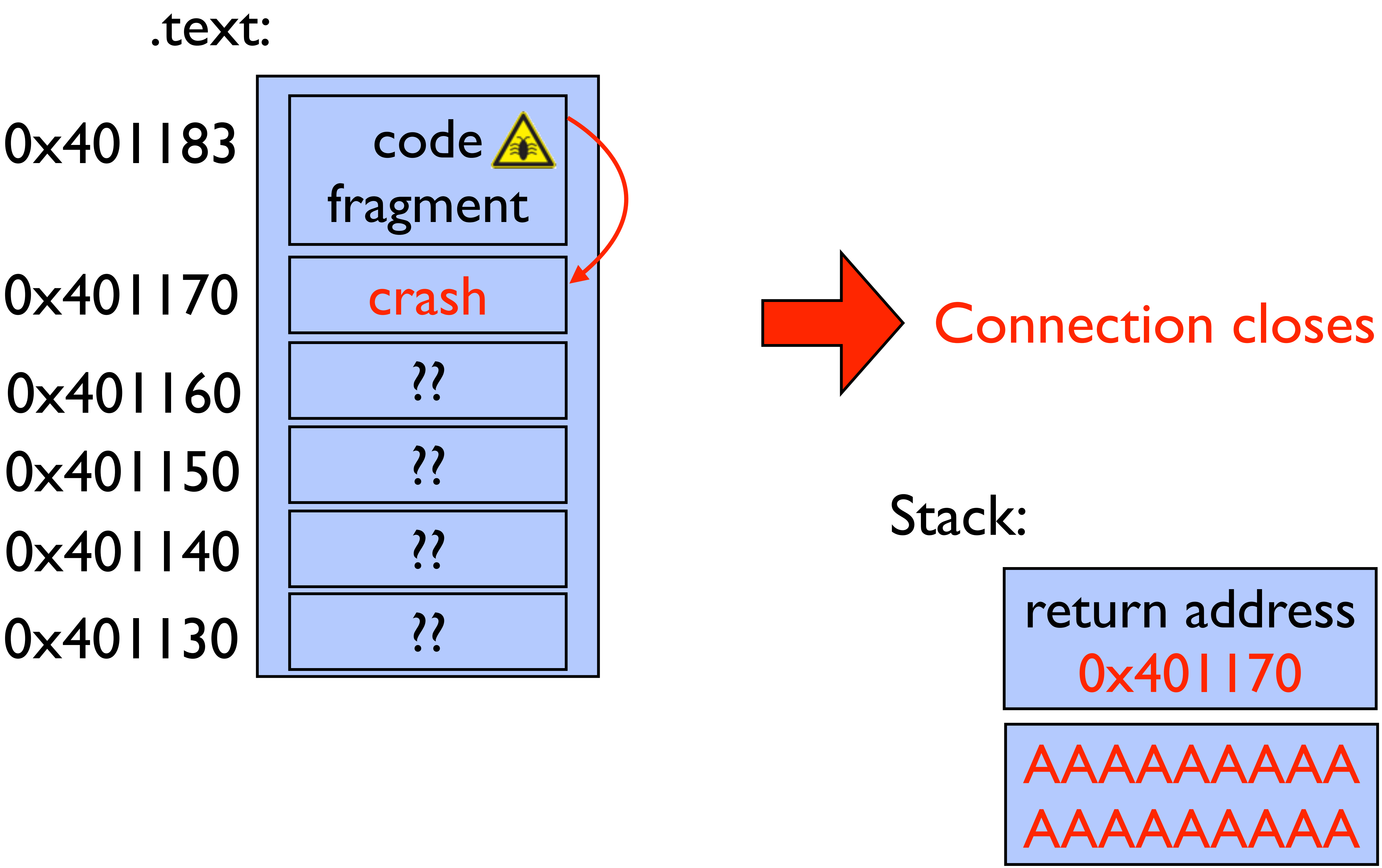
- Overwrite a single byte with value X:
  - No crash: stack had value X.
  - Crash: guess X was incorrect.
- Known technique for leaking canaries.



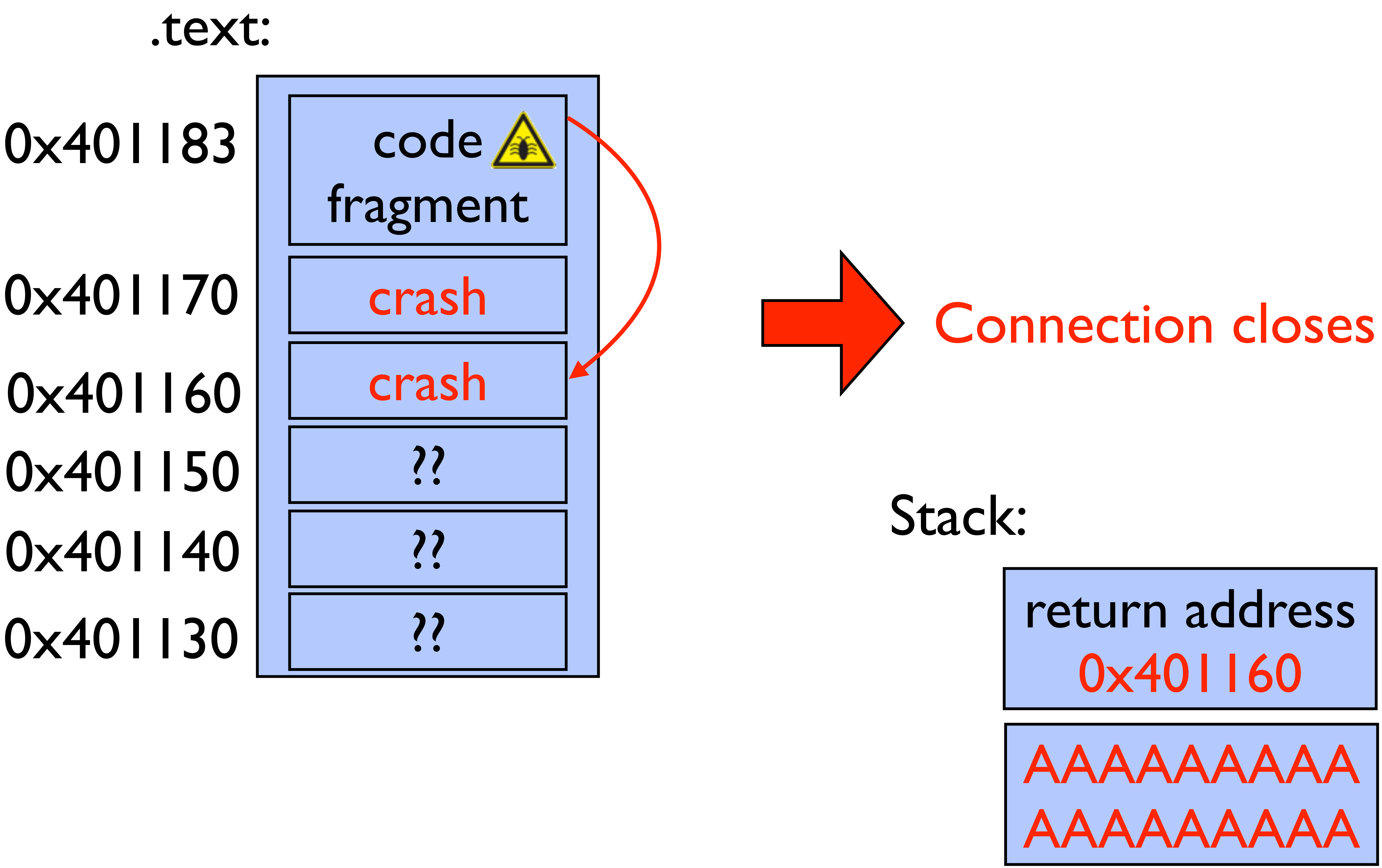
How to find gadgets?



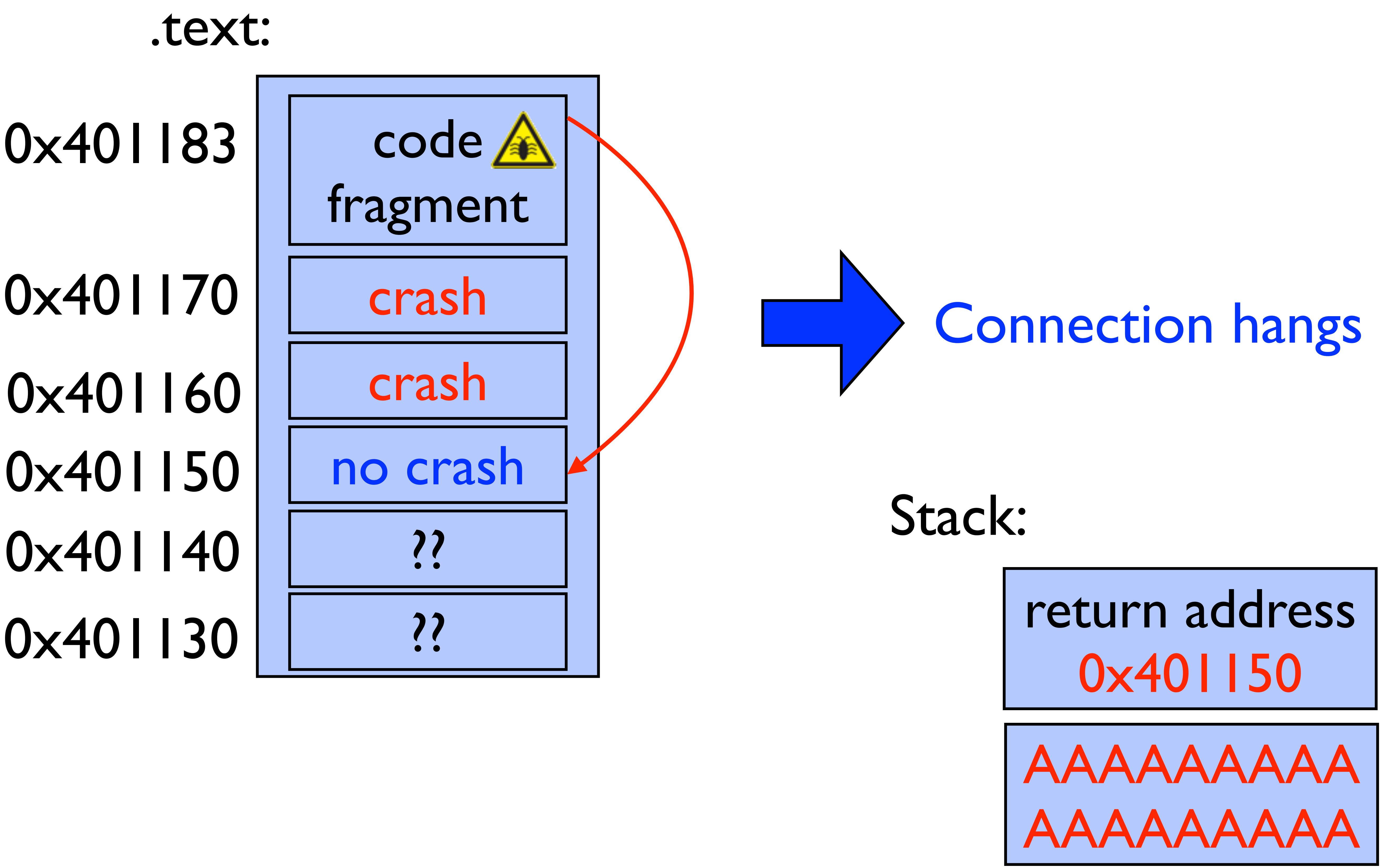
How to find gadgets?



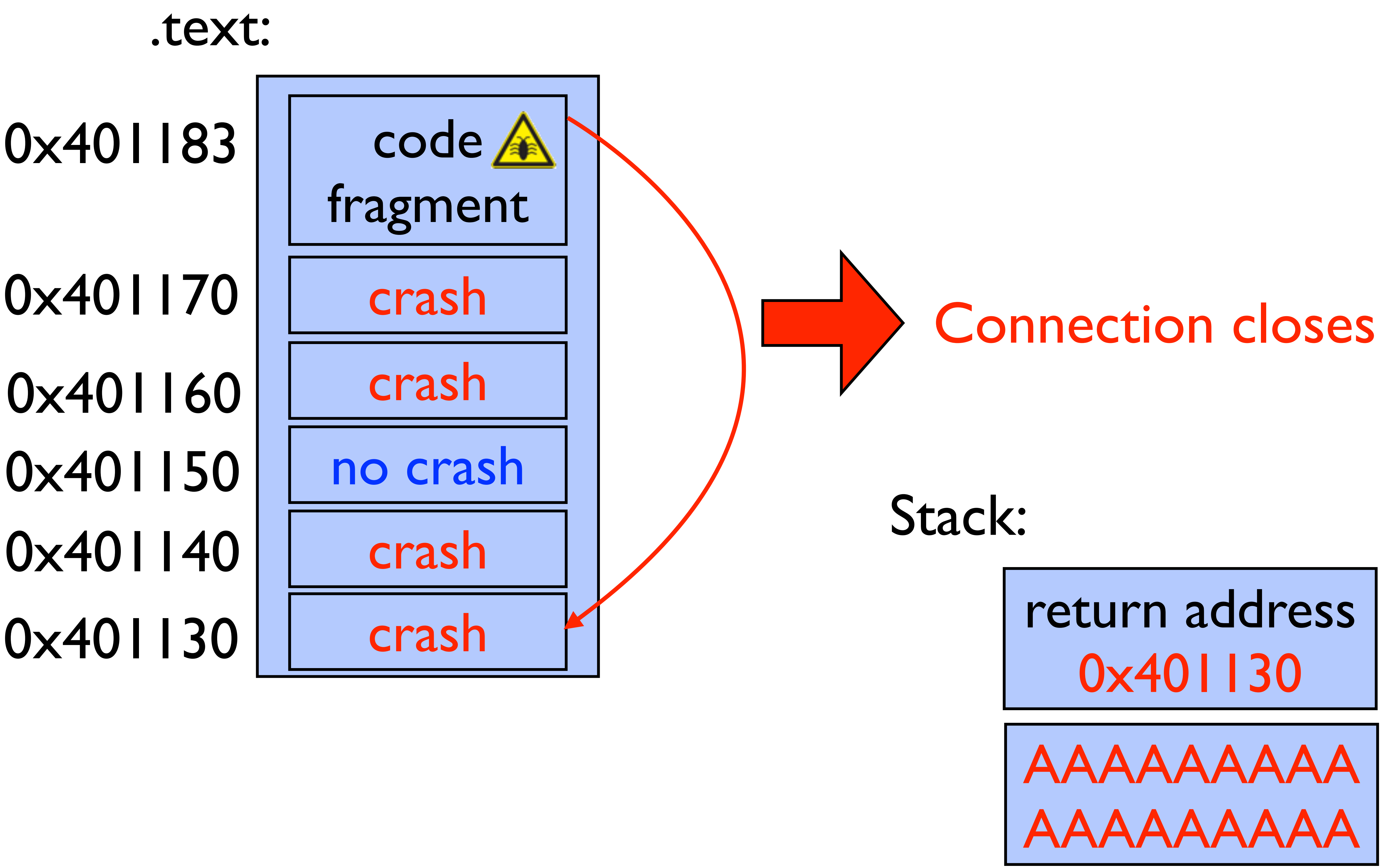
How to find gadgets?



How to find gadgets?



How to find gadgets?



Three types of gadgets

Stop gadget

```
sleep(10);
return;
```

- Never crashes

Crash gadget

```
abort();
return;
```

- Always crashes

Useful gadget

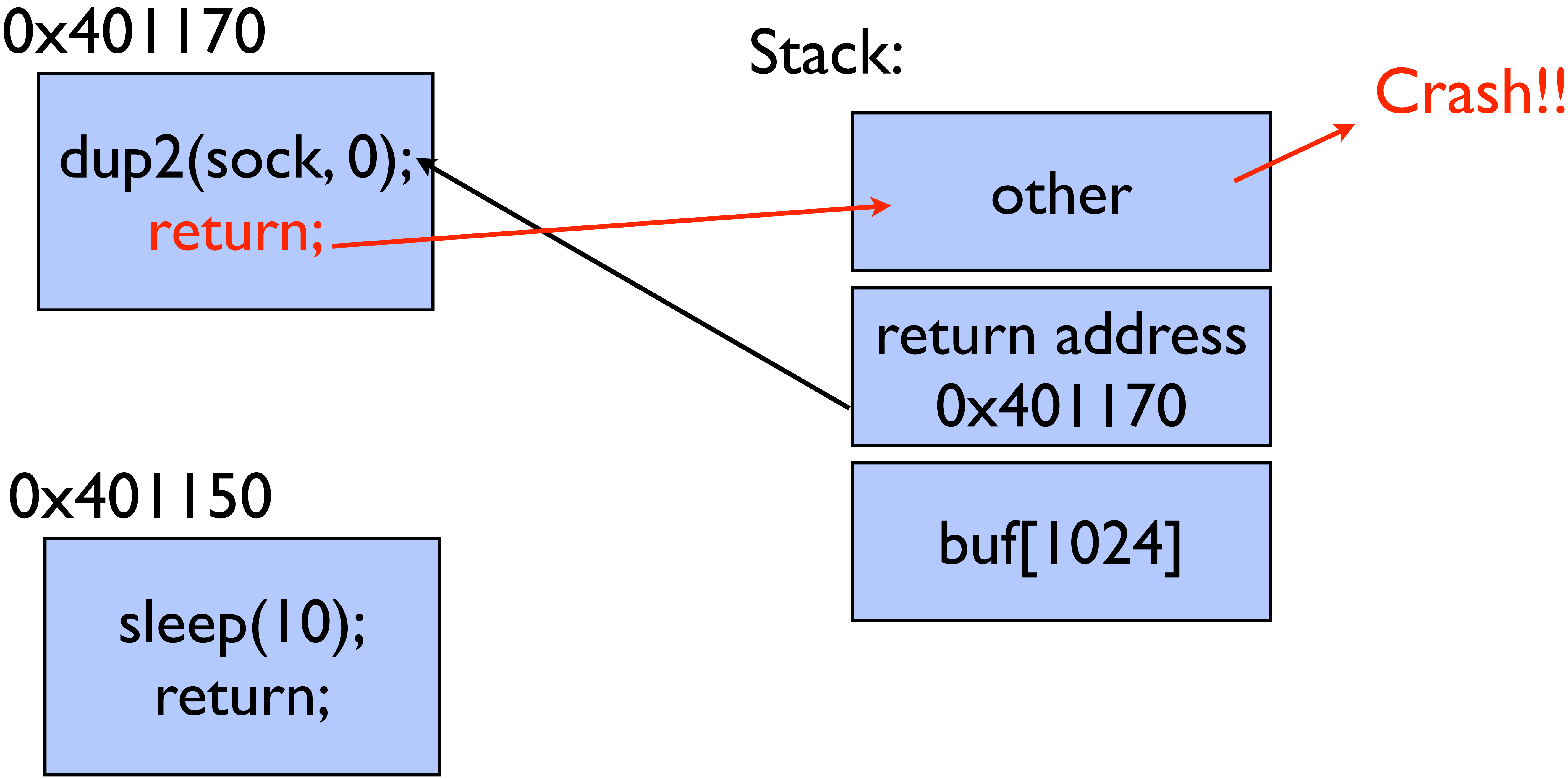
```
dup2(sock, 0);
return;
```

- Crash depends on return

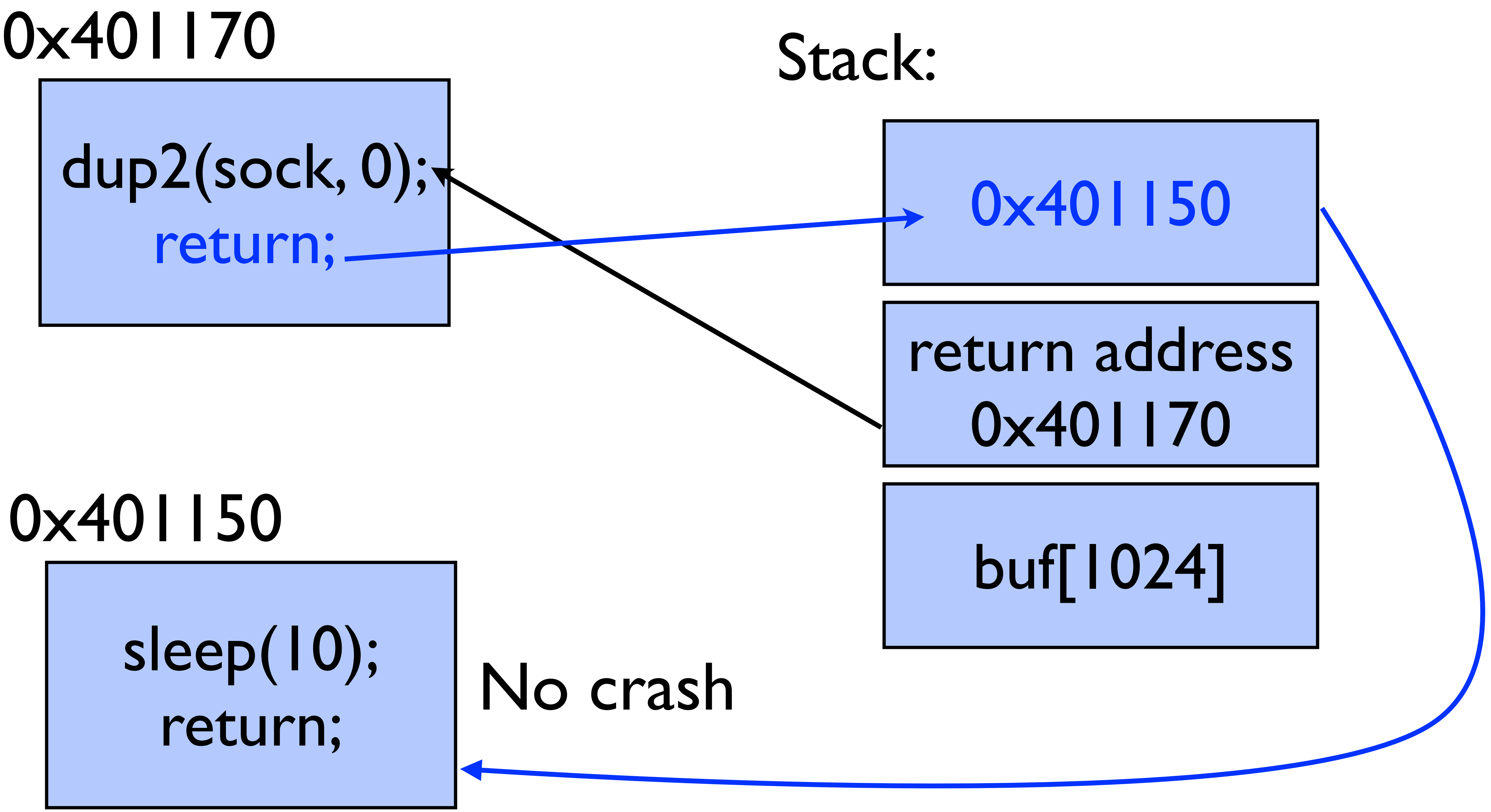
Three types of gadgets

Stop gadget	Crash gadget	Useful gadget
`sleep(10);` `return;`	`abort();` `return;`	`dup2(sock, 0);` `return;`
• Never crashes	• Always crashes	• Crash depends on return

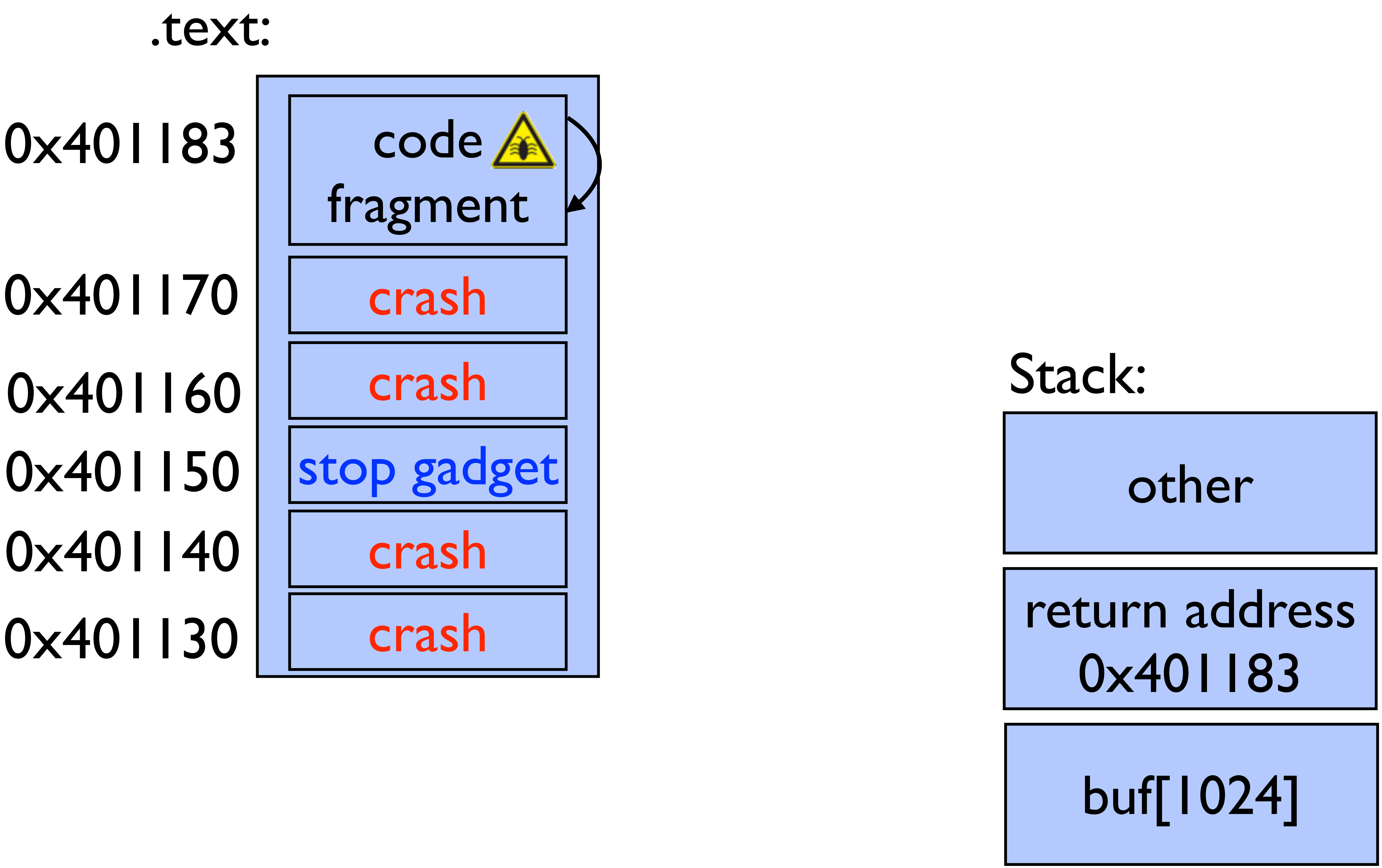
Finding useful gadgets



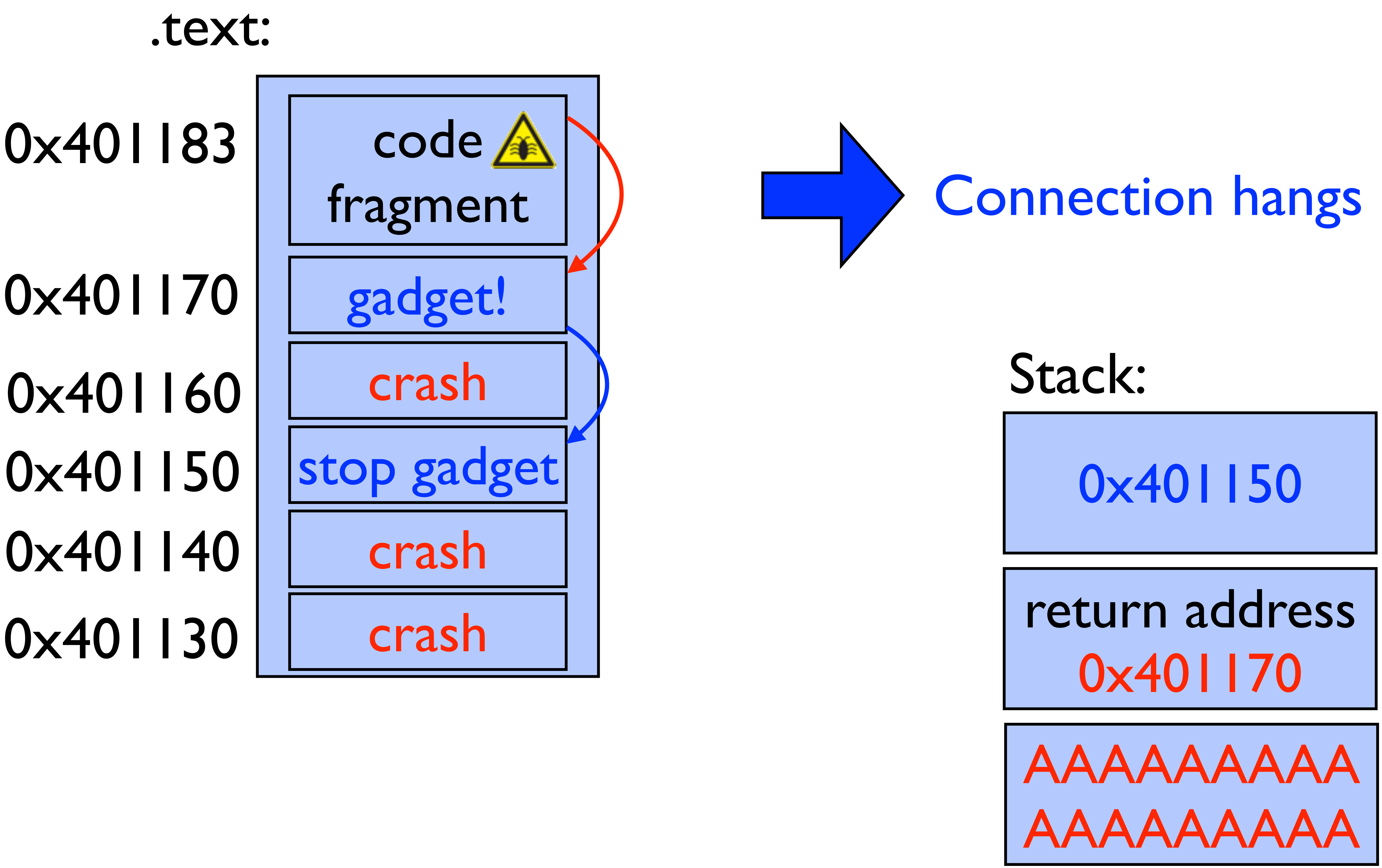
Finding useful gadgets



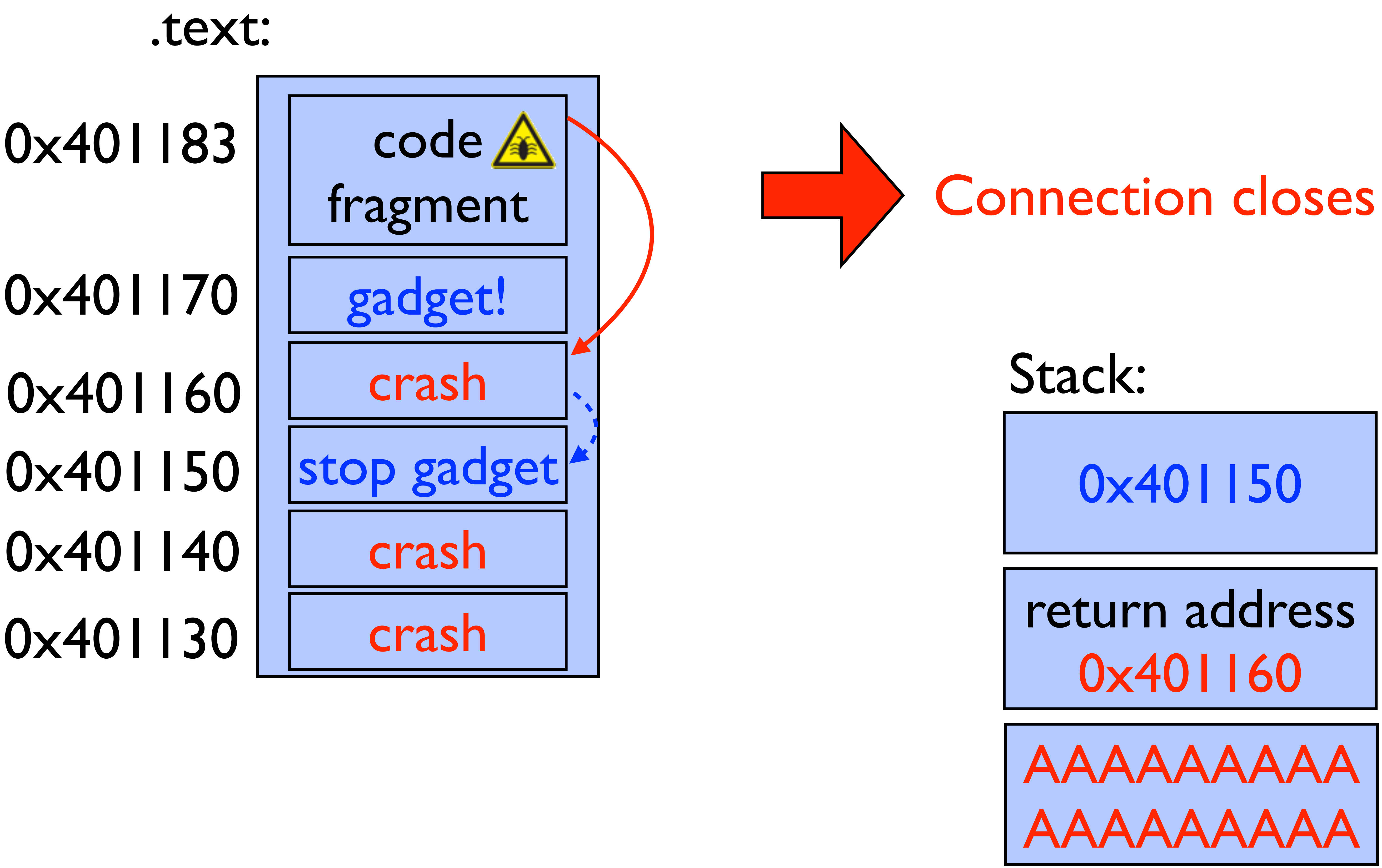
How to find gadgets?



How to find gadgets?



How to find gadgets?



What are we looking for?

write(int sock, void *buf, int len)

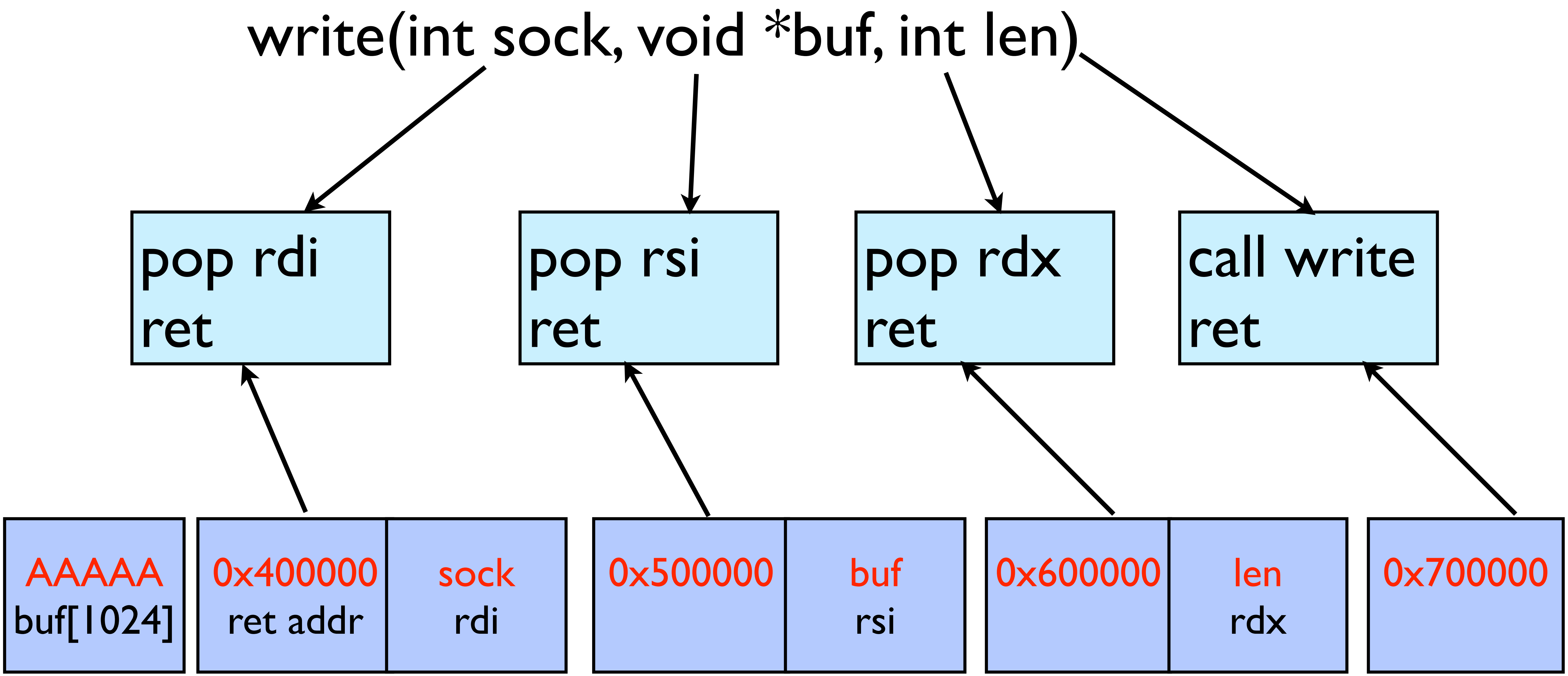
```
pop rdi
ret
```

```
pop rsi
ret
```

```
pop rdx
ret
```

```
call write
ret
```

What are we looking for?



Pieces of the puzzle

pop rsi
ret

pop rdi
ret

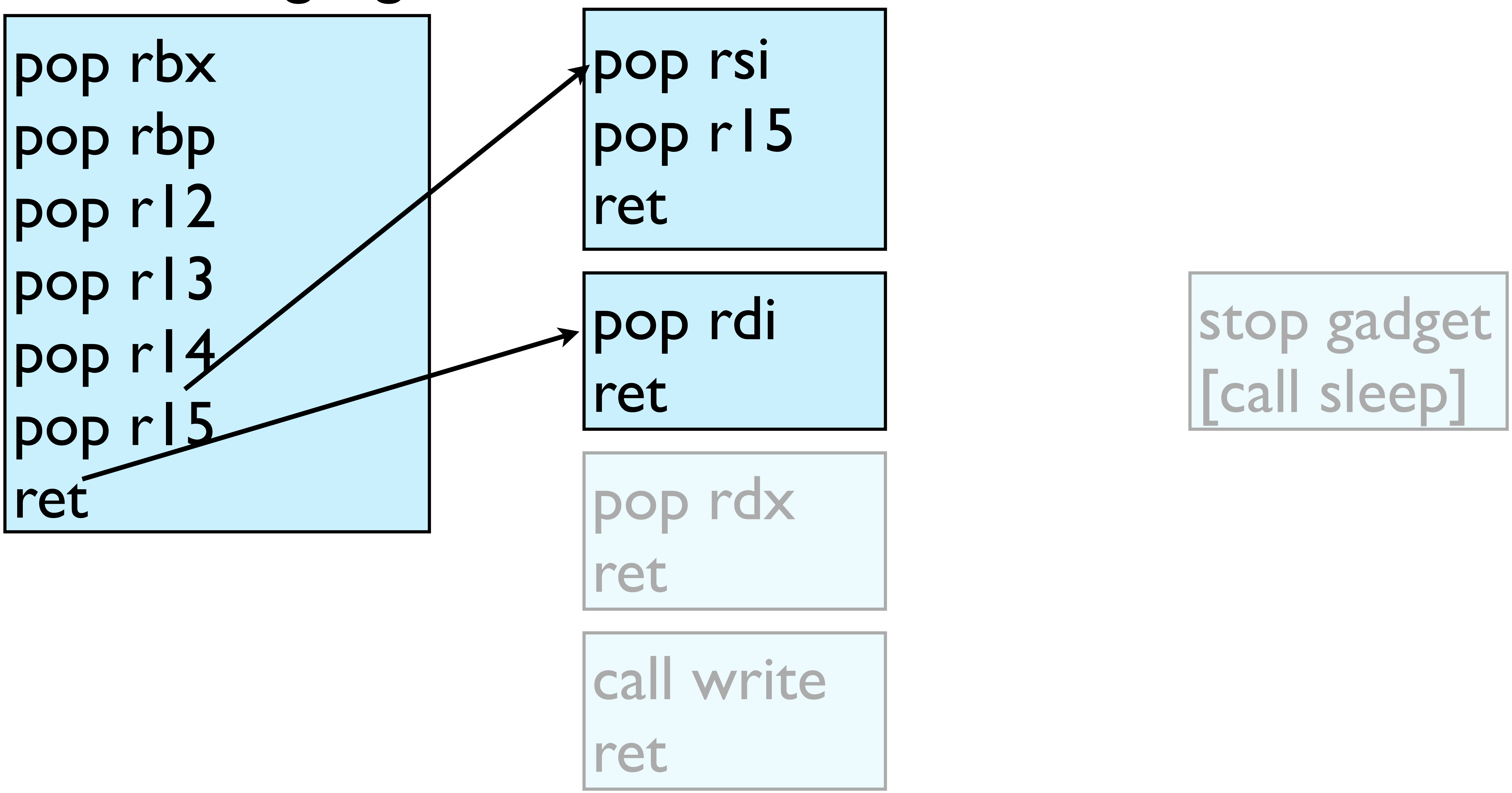
pop rdx
ret

call write
ret

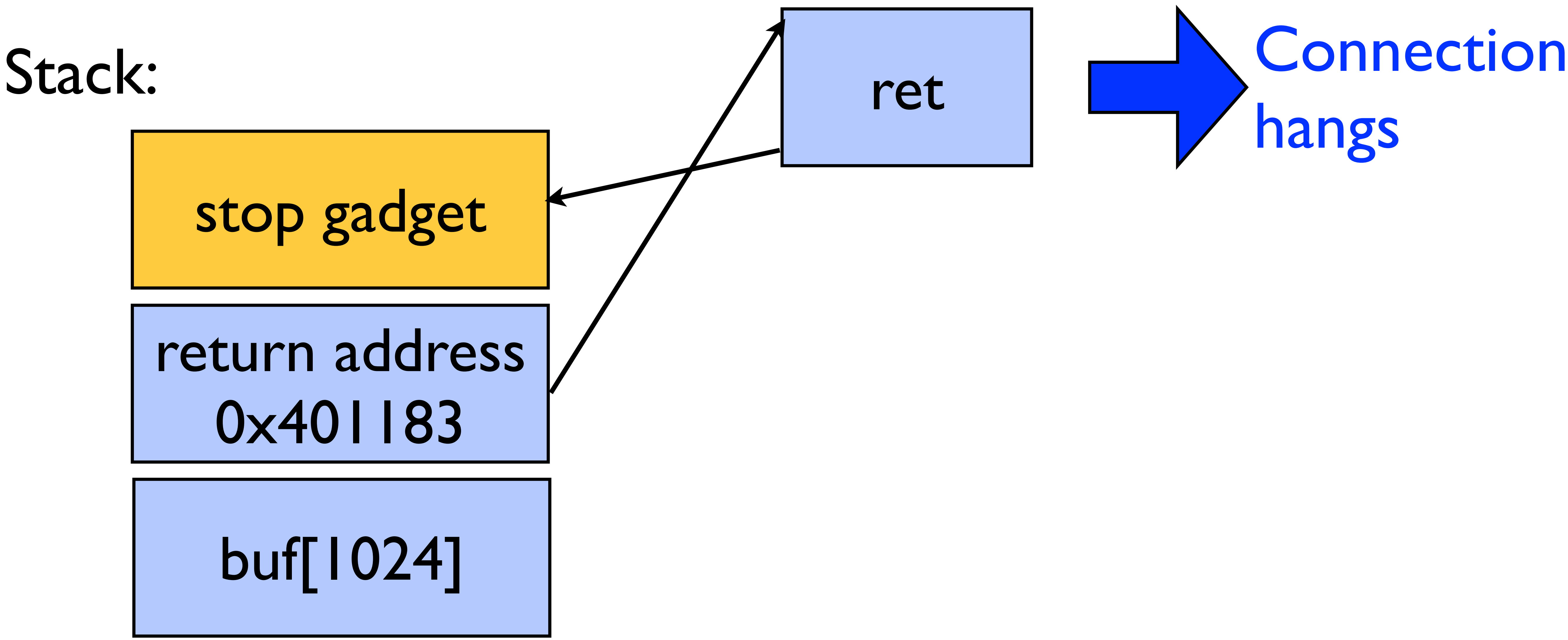
stop gadget
[call sleep]

Pieces of the puzzle

The BROP gadget

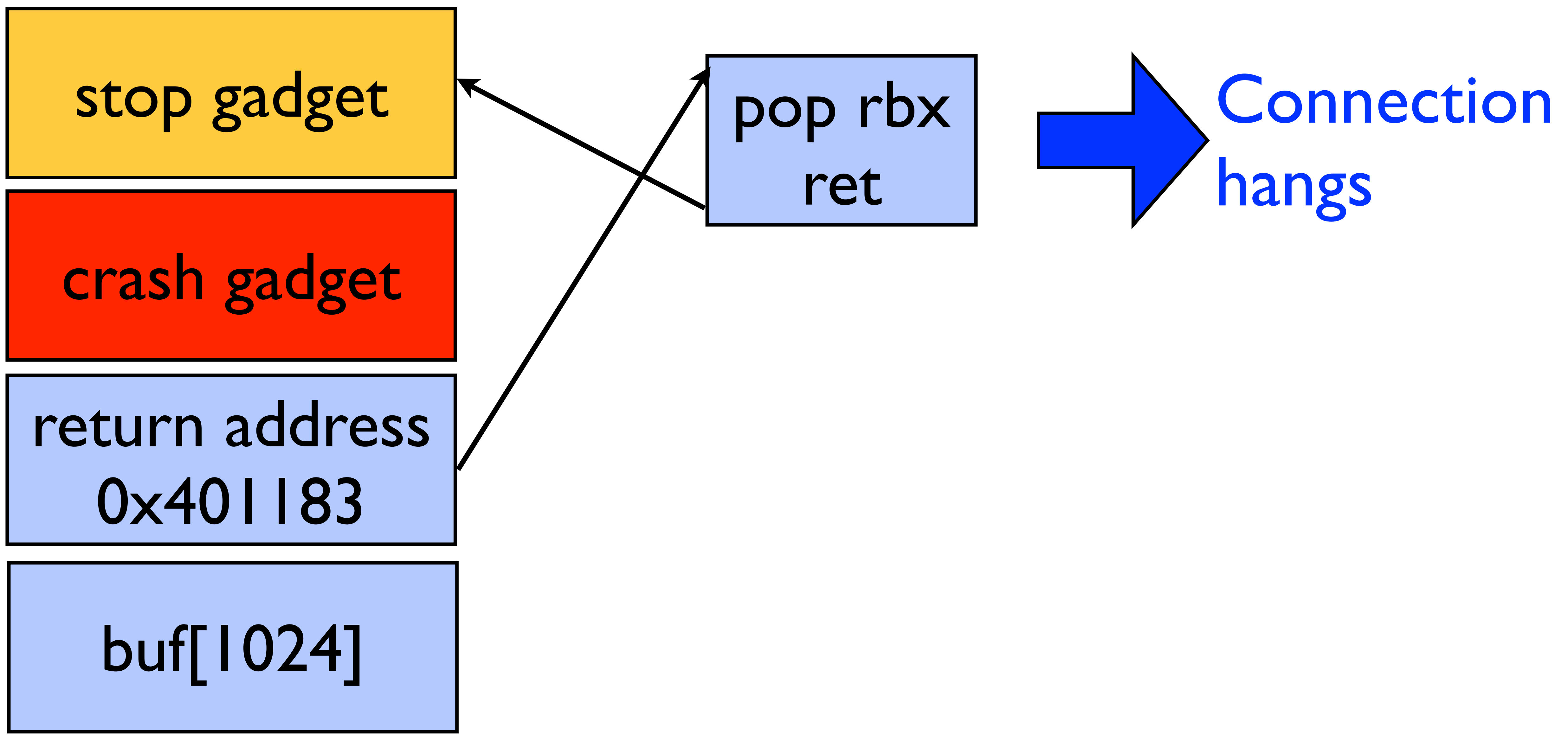


Finding the BROP gadget

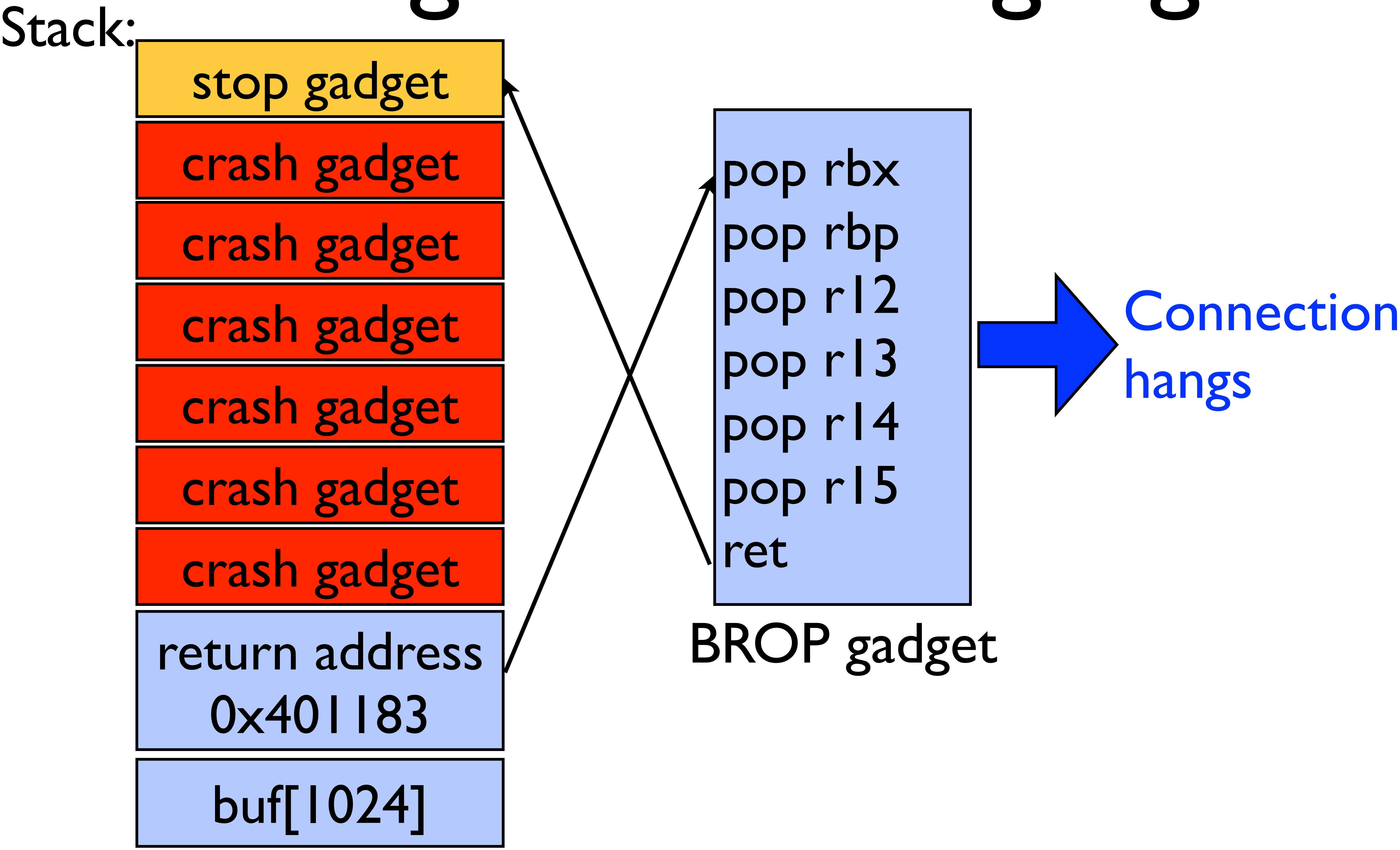


Finding the BROP gadget

Stack:



Finding the BROP gadget



Pieces of the puzzle

The BROP gadget

pop rbx
pop rbp
pop r12
pop r13
pop r14
pop r15
ret

pop rsi
pop r15
ret

pop rdi
ret

pop rdx
ret

call write
ret

stop gadget
[call sleep]

Pieces of the puzzle

The BROP gadget

```
pop rbx
pop rbp
pop r12
pop r13
pop r14
pop r15
ret
```

```
pop rsi
pop r15
ret
```

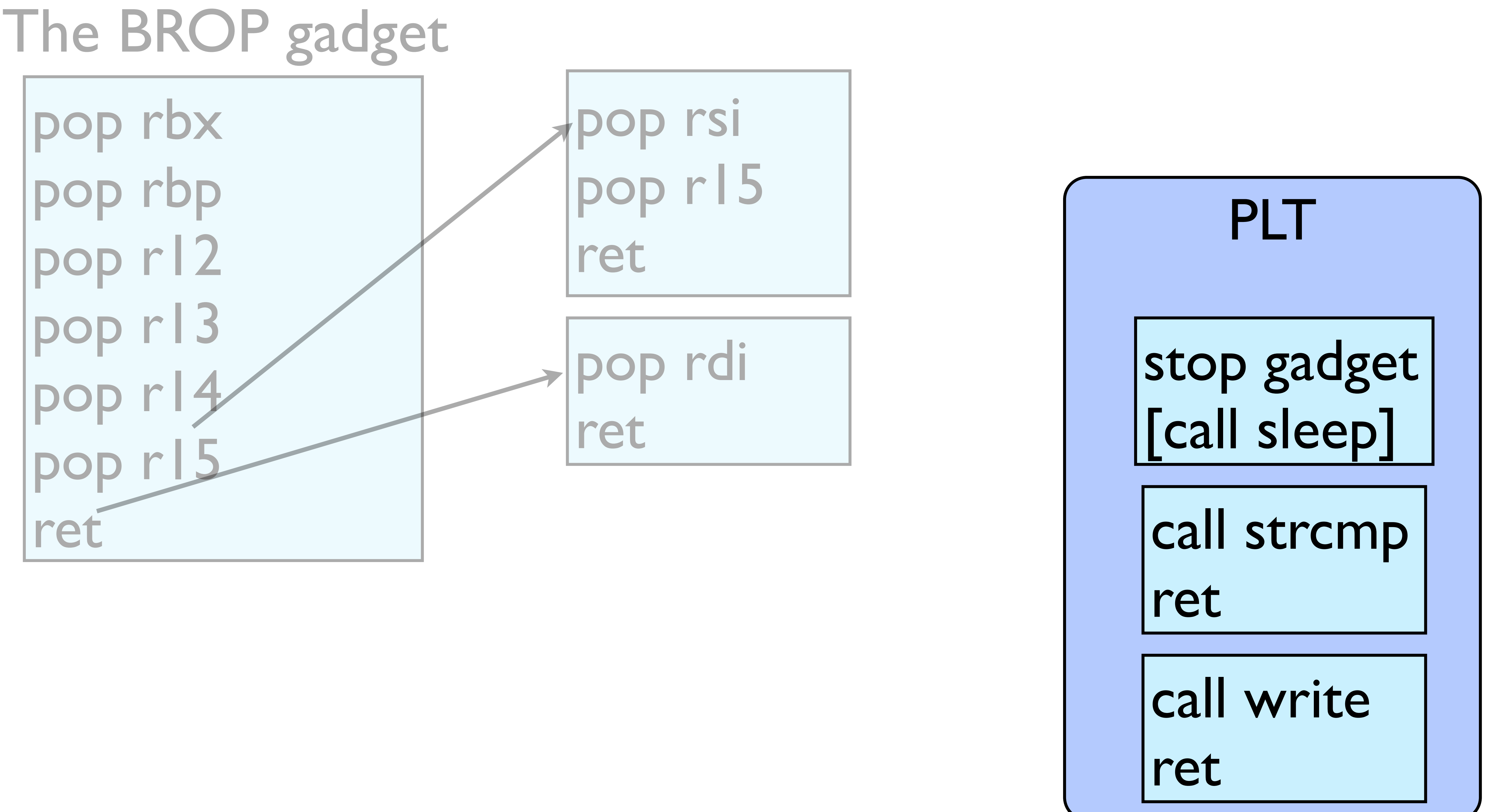
```
pop rdi
ret
```

```
call strcmp
ret
```

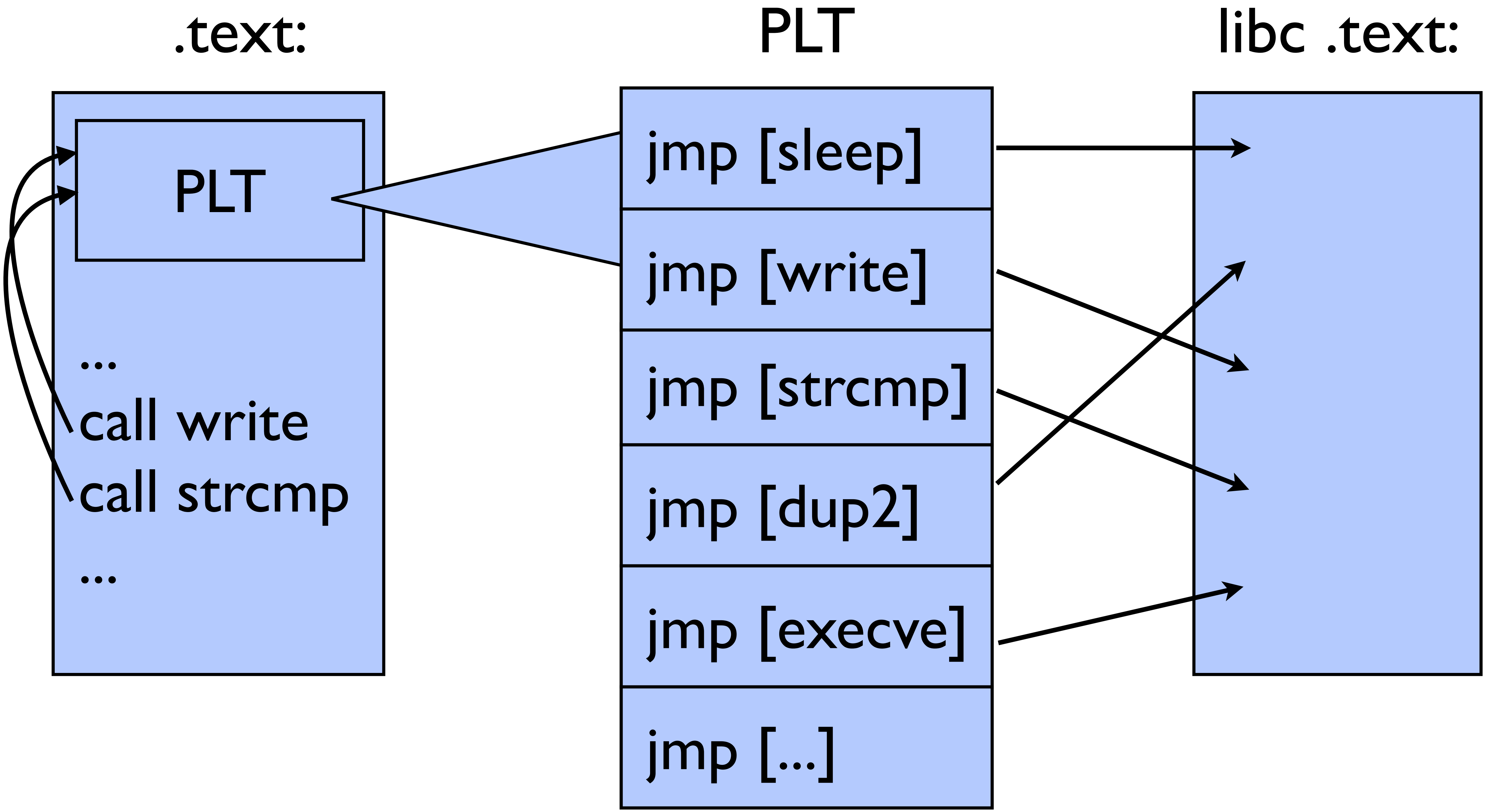
```
call write
ret
```

stop gadget
[call sleep]

Pieces of the puzzle



Procedure Linking Table (PLT)



Fingerprinting strcmp

arg1	arg2	result
readable	0x0	crash
0x0	readable	crash
readable	readable	nocrash

Can now control three arguments:
strcmp sets RDX to length of string

Finding write

- Try sending data to socket by calling candidate PLT function.
- check if data received on socket.
- chain writes with different FD numbers to find socket. Use multiple connections.

Launching a shell

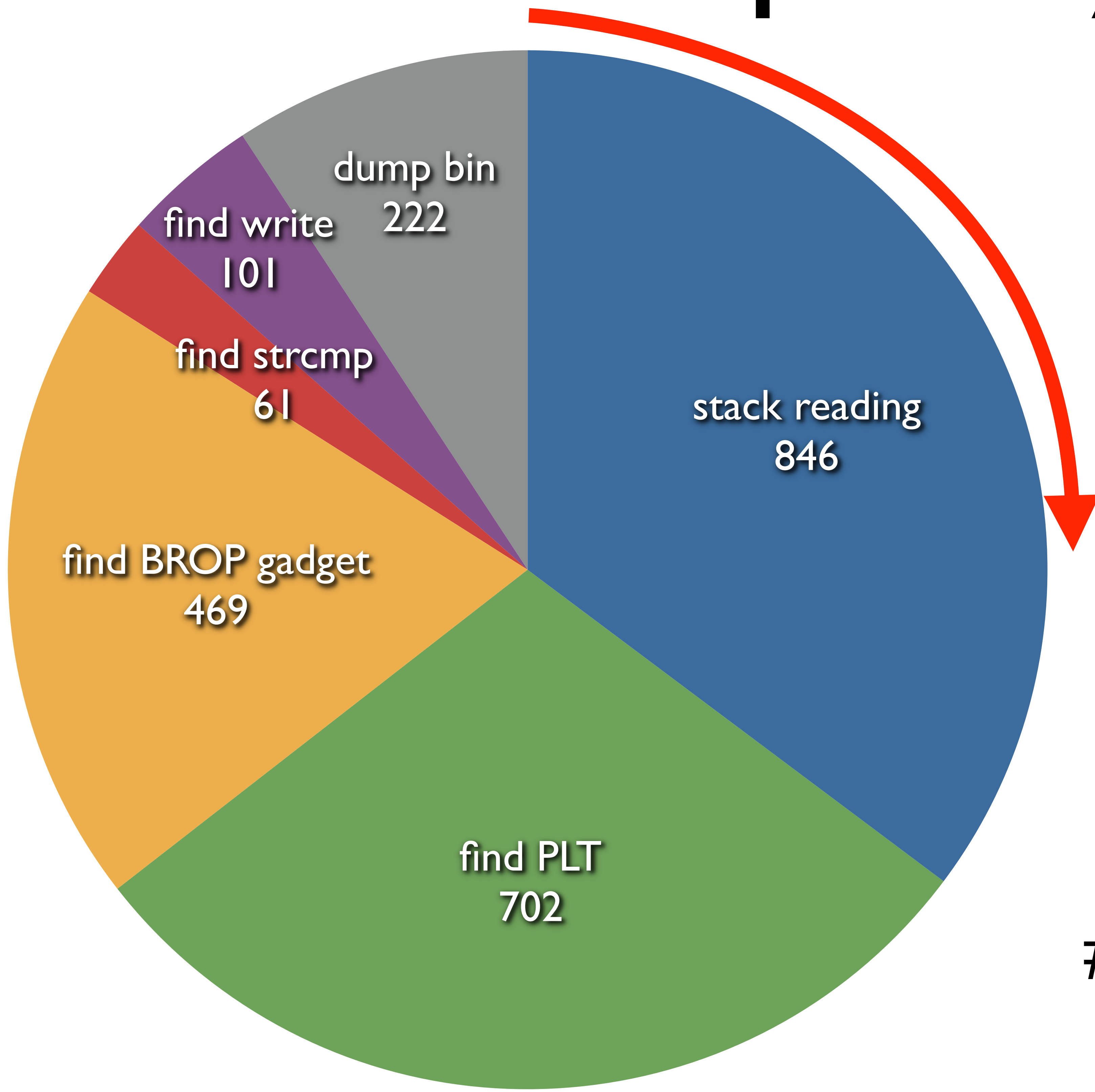
1. dump binary from memory to network. Not blind anymore!
2. dump symbol table to find PLT calls.
3. redirect stdin/out to socket:
   - dup2(sock, 0); dup2(sock, 1);
4. read() “/bin/sh” from socket to memory
5. execve(“/bin/sh”, 0, 0)

Braille

- Fully automated: from first crash to shell.
- 2,000 lines of Ruby.
- Needs function that will trigger overflow:
  - nginx: 68 lines.
  - MySQL: 121 lines.
  - toy proprietary service: 35 lines.

```
try_exp(data) → true  crash
                false no crash
```

Attack complexity



# of requests
for nginx