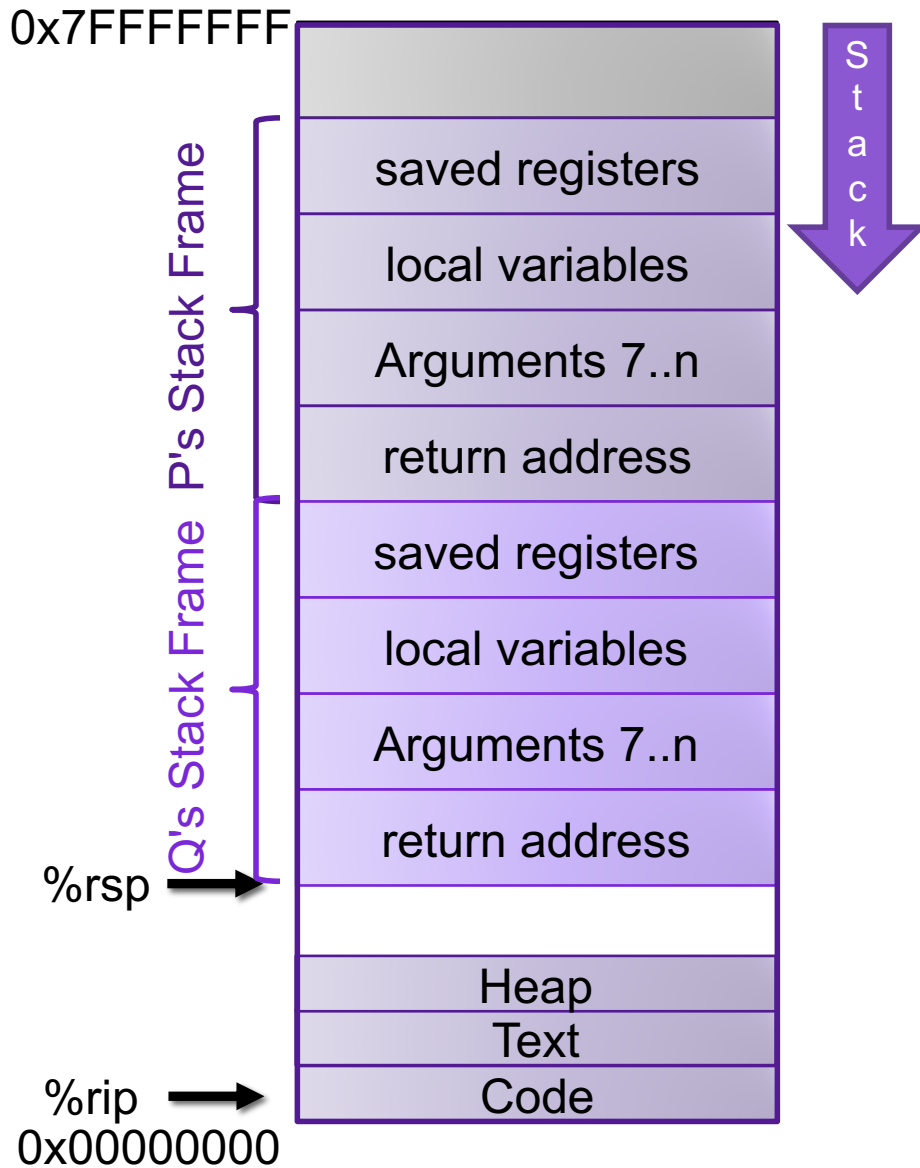
Lecture 9: Use and Abuse of the Stack (cont'd)

CS 105

September 26, 2019

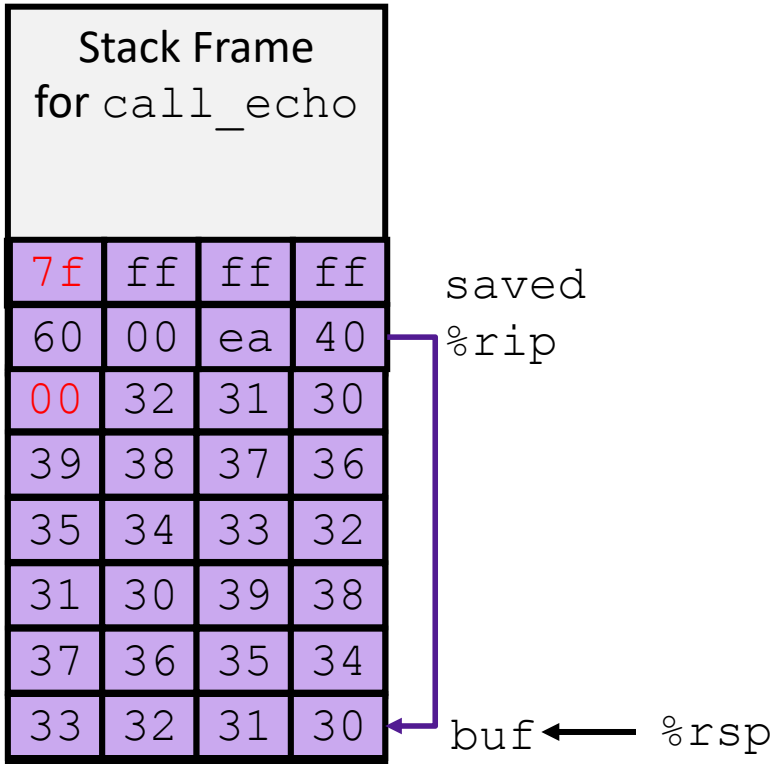
Review: Stack Frames

- Each function called gets a stack frame
- Passing data:
  - calling procedure P uses registers (and stack) to provide parameters to Q.
  - Q uses register %rax for return value
- Passing control:
  - **`call <proc>`**
    - Pushes return address (current **`%rip`**) onto stack
    - Sets **`%rip`** to first instruction of proc
  - **`ret`**
    - Pops return address from stack and places it in **`%rip`**
- Local storage:
  - allocate space on the stack by decrementing stack pointer, deallocate by incrementing



Review: Buffer Overflow

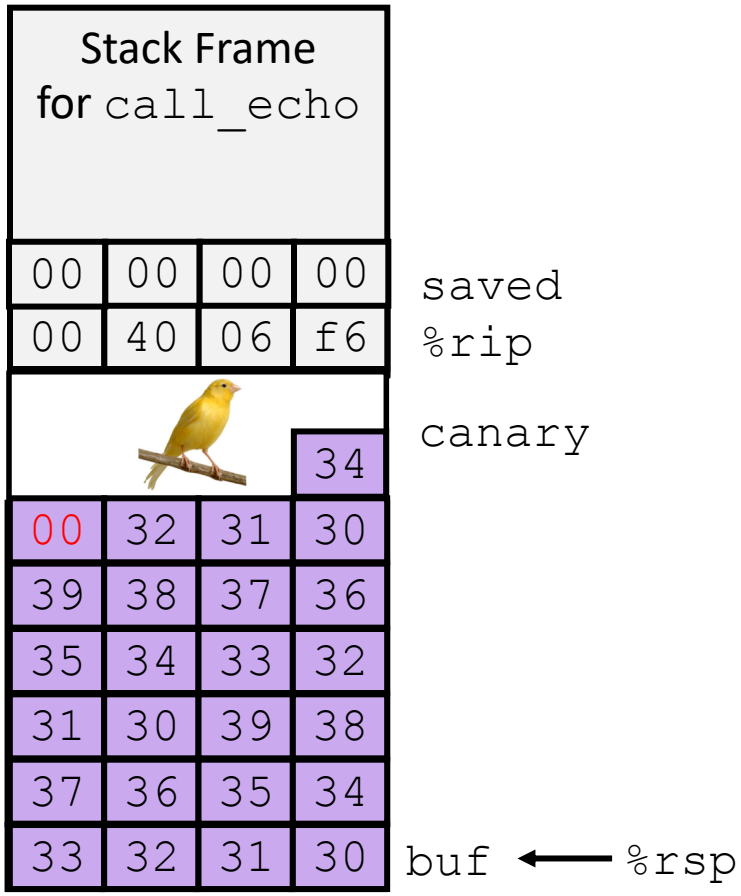
- Most common form of memory reference bug
  - Unchecked lengths on string inputs
  - Particularly for bounded character arrays on the stack
    - sometimes referred to as stack smashing




```
/* Echo Line */
void echo()
{
    char buf[4];
    gets(buf);
    puts(buf);
}
```

```
echo:
   subq   $18, %rsp
   movq   %rsp, %rdi
   call   gets
   call puts
   addq   $18, %rsp
   ret
```

Review: Stack Canaries




```
/* Echo Line */
void echo()
{
    char buf[4];
    gets(buf);
    puts(buf);
}
```

```
echo:
  subq  $24, %rsp
  movq  %rsp, %rdi
  call  gets
  call puts
  movq    24(%rsp), %rdx
  xorq    %fs:40, %rdx
  je      .L3
  call    __stack_chk_fail
.L3
  addq  $24, %rsp
  ret
```

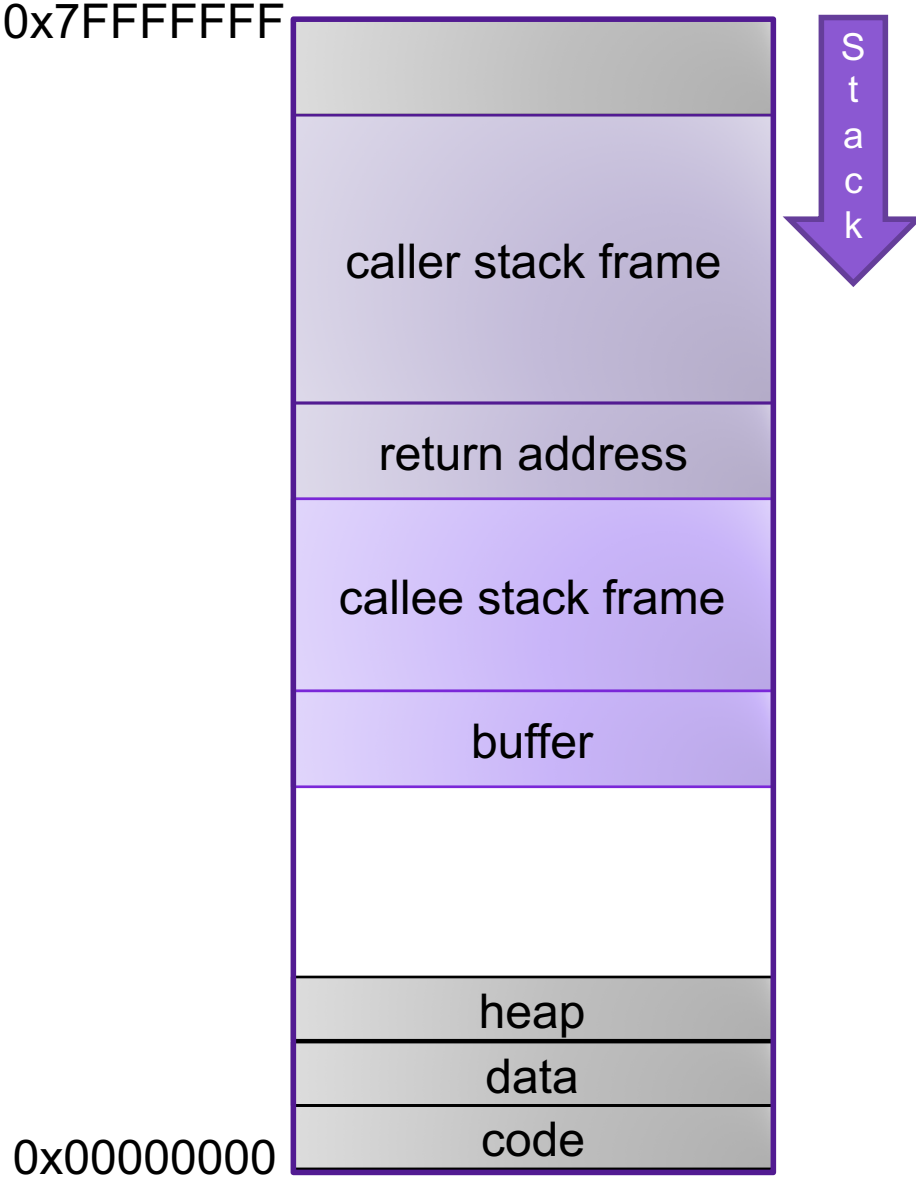
Review: Memory Tagging

Code Reuse Attacks

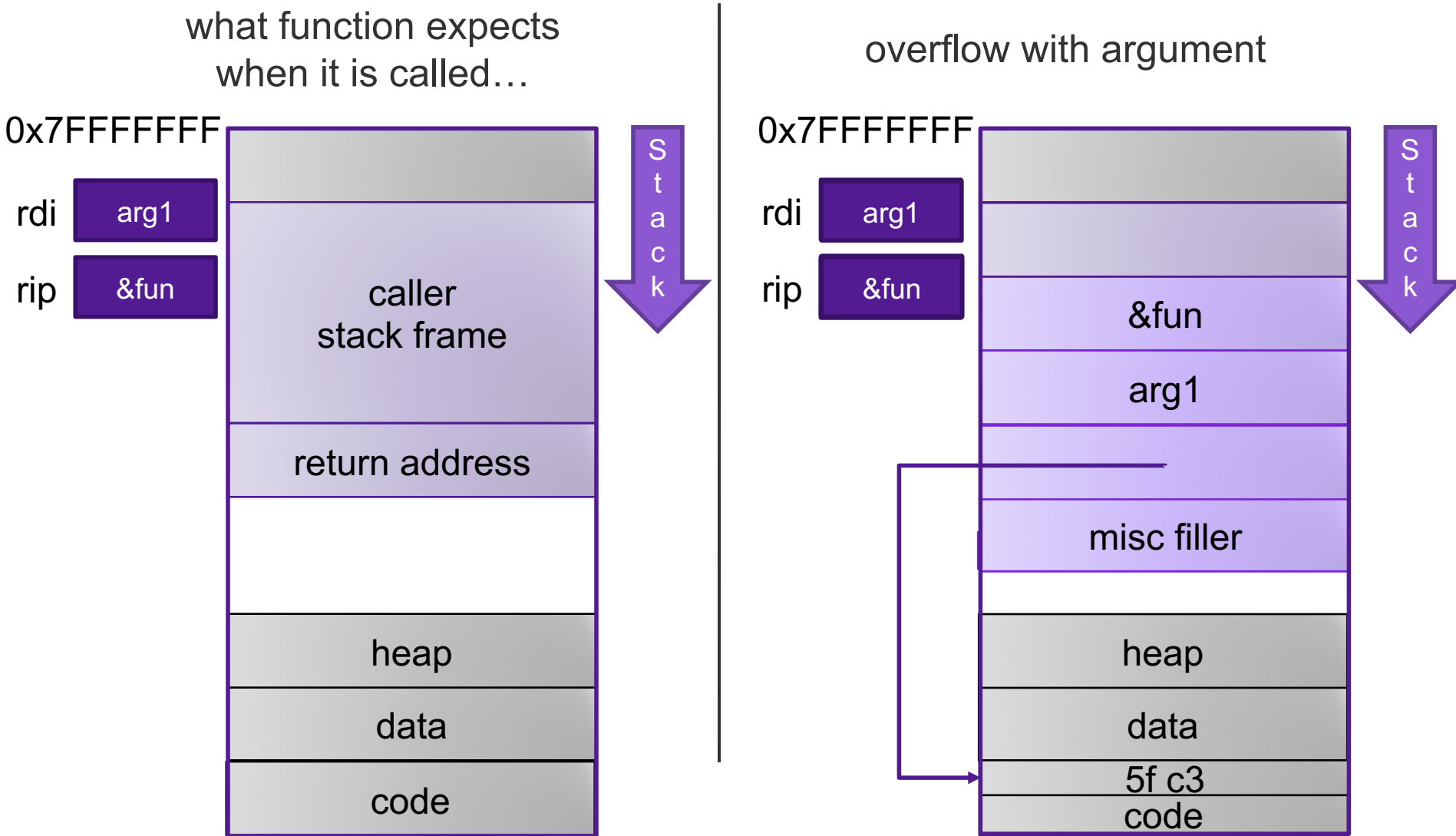
- Key idea: execute instructions that already exist
- Defeats memory tagging defenses
- Examples:
  1. return to a function in the current program
  2. return to a library function (e.g., return-into-`libc`)
  3. return to some other instruction (return-oriented programming)

Returning to a function

- Overwrite the saved return address with the location of a function in the current program



Handling Arguments



Return-into-libc

Sr.No.	Function & Description
1	**double atof(const char *str)** Converts the string pointed to, by the argument *str* to a floating-point number (type double).
2	**int atoi(const char *str)** Converts the string pointed to, by the argument *str* to an integer (type int).
3	**long int atol(const char *str)** Converts the string pointed to, by the argument *str* to a long integer (type long int).
8	**void free(void *ptr** Deallocates the memory previously allocated by a call to *calloc, malloc,* or *realloc*.
9	**void *malloc(size_t size)** Allocates the requested memory and returns a pointer to it.
10	**void *realloc(void *ptr, size_t size)** Attempts to resize the memory block pointed to by ptr that was previously allocated with a call to *malloc* or *calloc*.
15	**int system(const char *string)** The command specified by string is passed to the host environment to be executed by the command processor.
16	**void *bsearch(const void *key, const void *base, size_t nitems, size_t size, int (*compar)(const void *, const void *))** Performs a binary search.
17	**void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))** Sorts an array.
18	**int abs(int x)** Returns the absolute value of x.
22	**int rand(void)** Returns a pseudo-random number in the range of 0 to *RAND_MAX*.
23	**void srand(unsigned int seed)** This function seeds the random number generator used by the function **rand**.

ASCII Armoring

- Make sure all system library addresses contain a null byte (0x00).
- Can be done by placing this code in the first 0x01010101 bytes of memory

Properties of x86 Assembly

- variable length instructions
- not word aligned
- dense instruction set

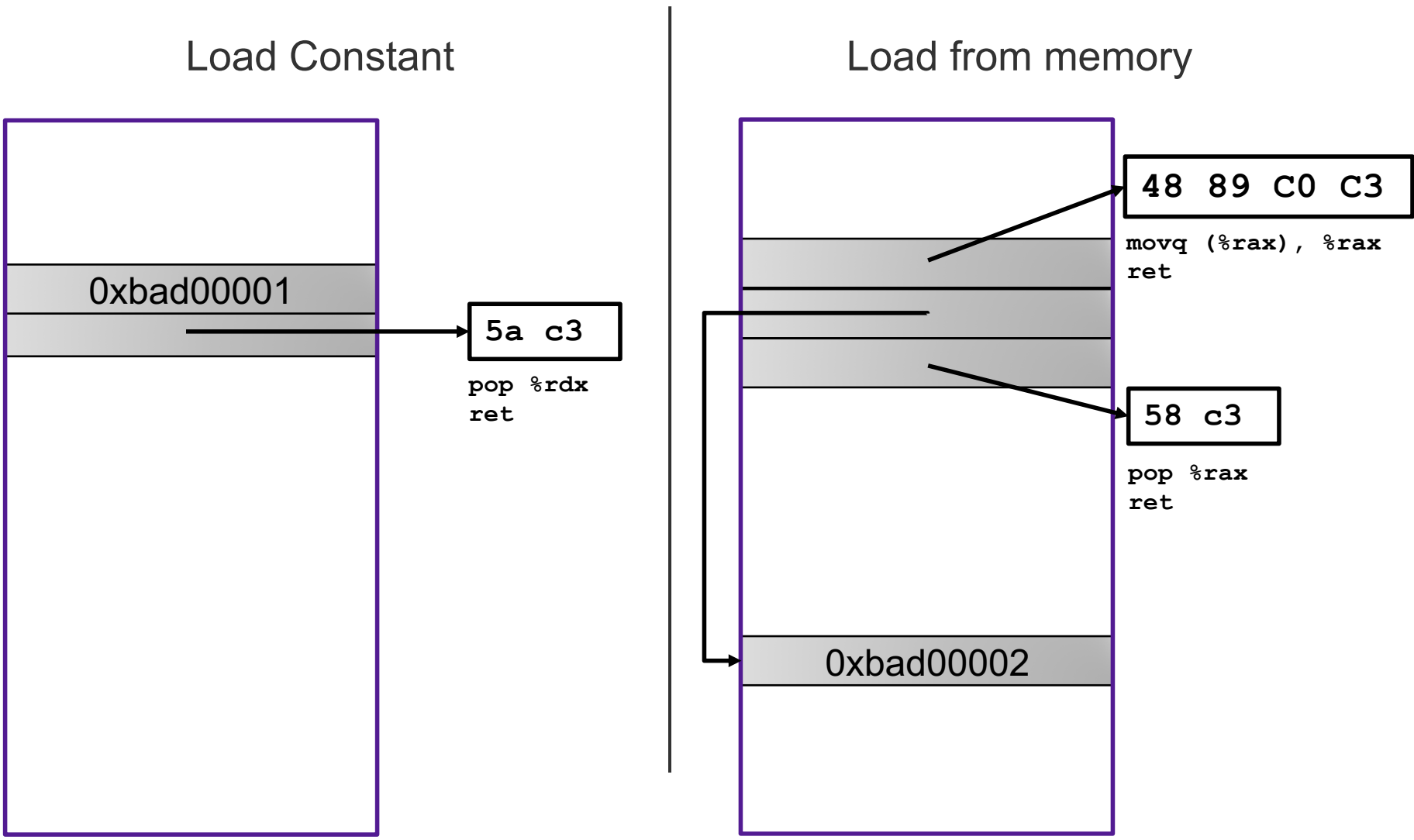
Gadgets

```
void setval(unsigned *p) {
  *p = 3347663060u;
}
```

```
<setval>:
4004d9: c7 07 d4 48 89 c7   movl $0xc78948d4,(%rdi)
4004df: c3                  ret
```

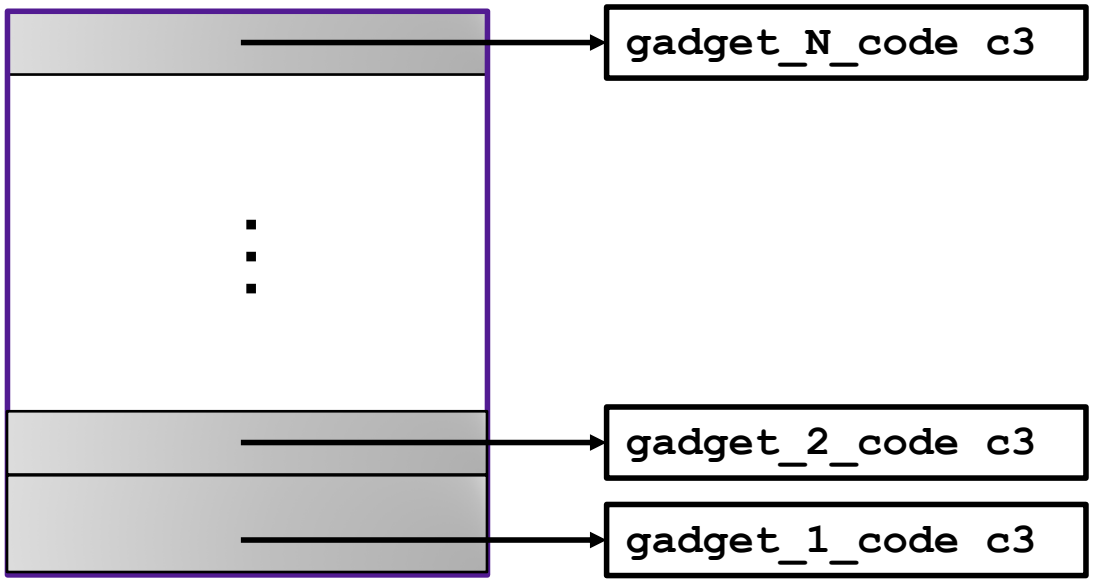
gadget address: `0x4004dc`
encodes: `movq %rax, %rdi`
`ret`
executes: `%rdi <- %rax`

Example Gadgets



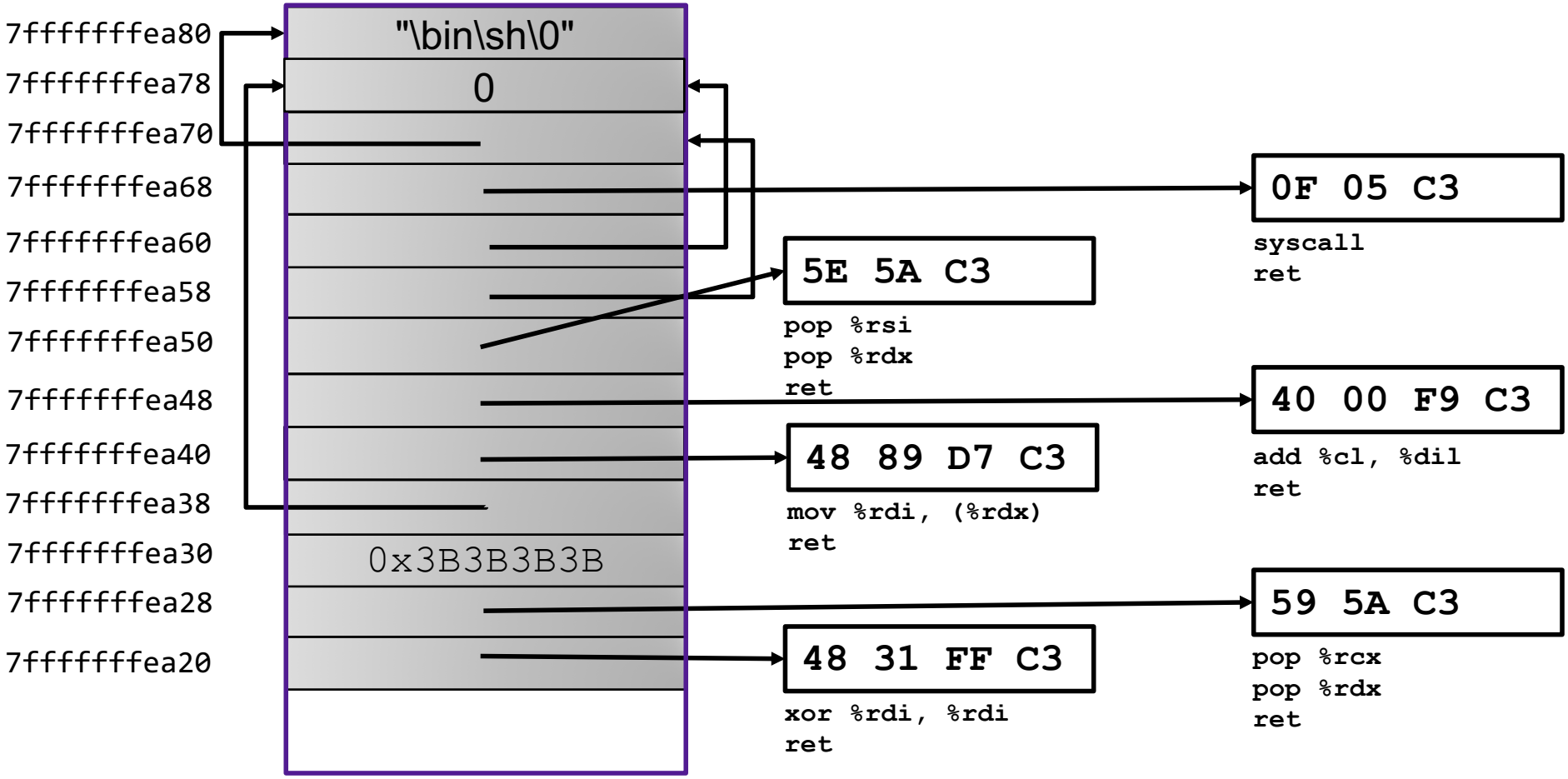
Return-oriented Programming

Return-oriented Programming



Final ret in each gadget sets pc (%rip) to beginning of next gadget code

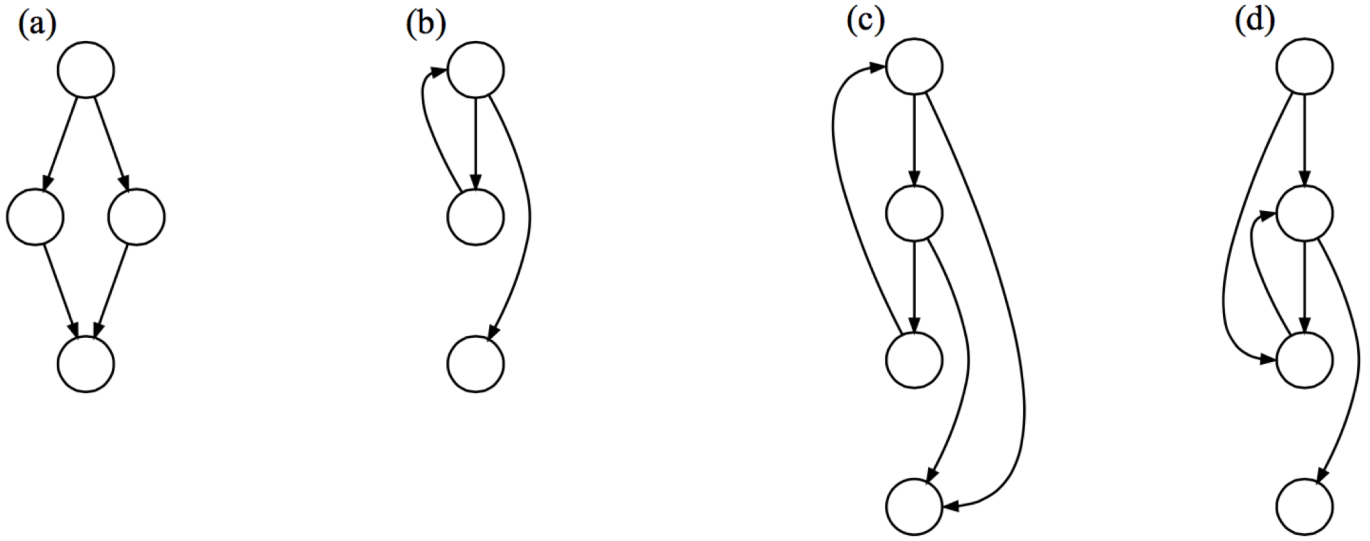
Return-Oriented Shellcode



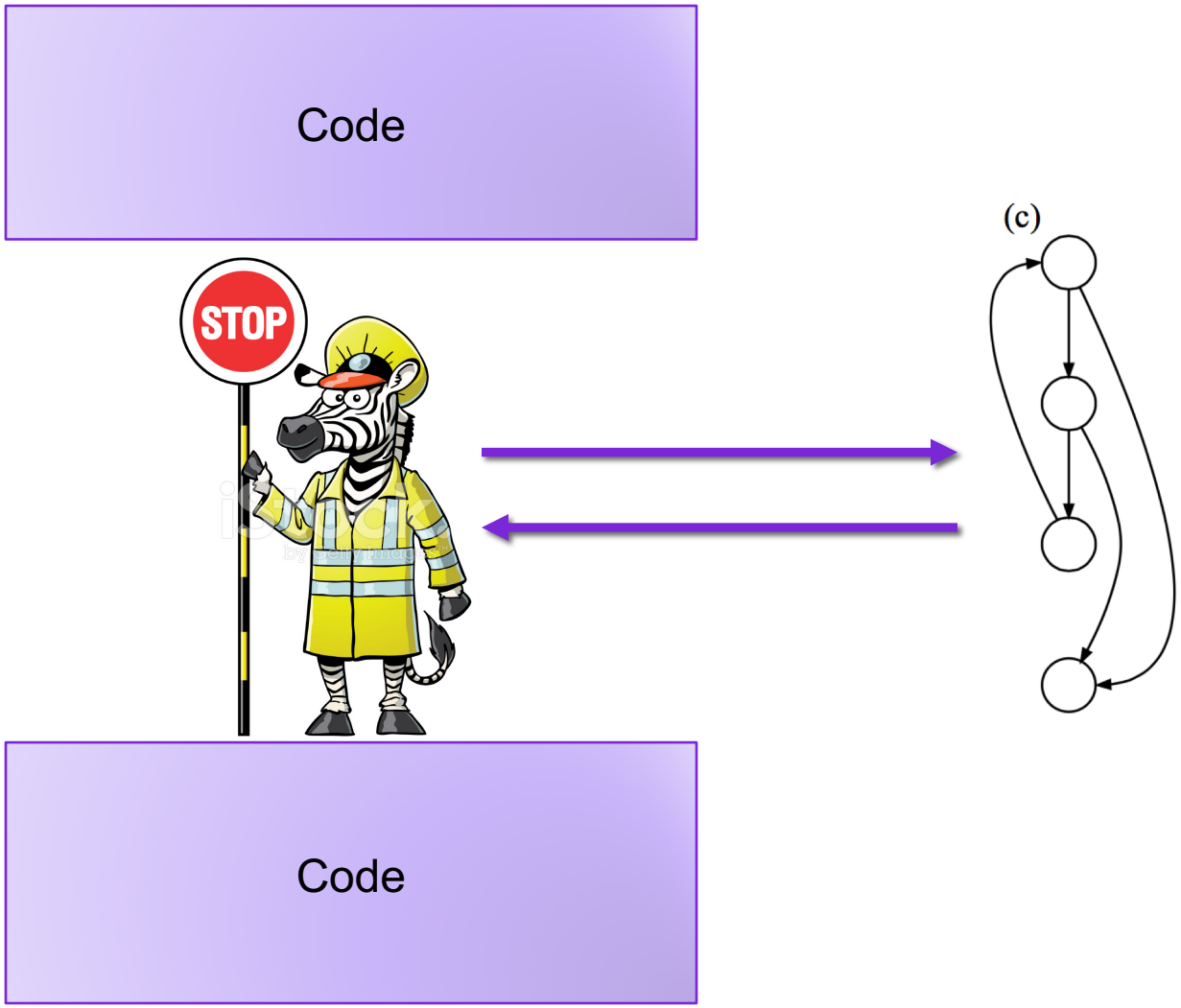
Address Space Layout Randomization

Gadget Elimination

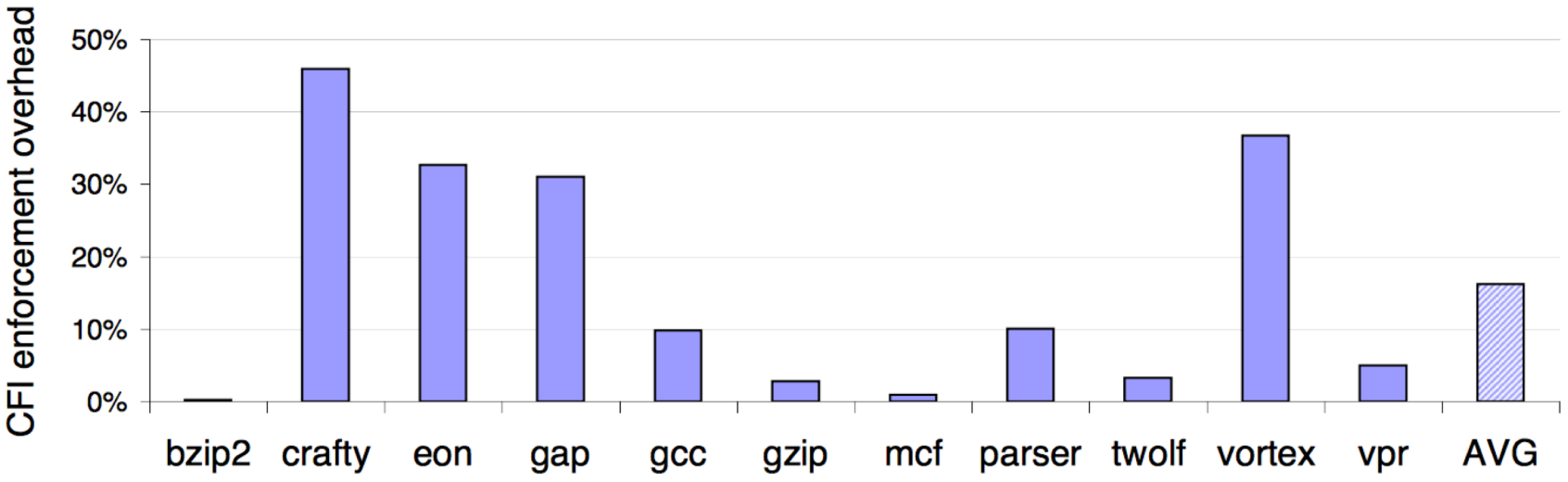
Control Flow Integrity



CFI = Insert Monitors

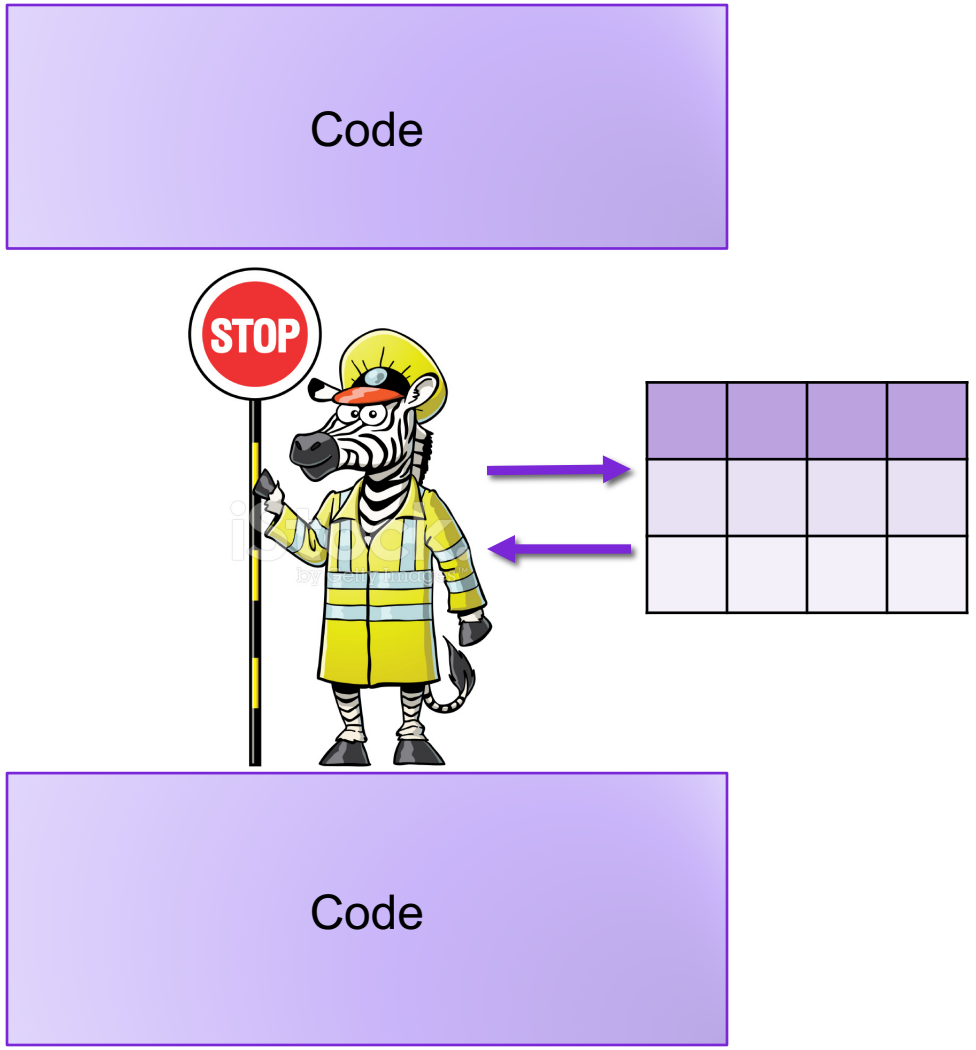


CFI Overhead



Control Flow Guard

- Approximate CFI implementation in Windows 8.1, 10
- Jump is valid if it beginning of function
  - Granularity: 8 bytes
- Check implemented as bitmap



The state of the world

Defenses:

- high-level languages
- Stack Canaries
- Memory tagging
- ASLR
- continuing research and development…

But all they aren't perfect!

The state of the world