Exercise Session 9

Systems Programming and Computer Architecture

Autumn Semester 2024

Disclaimer

Systems@ETH zürich

- **Website**: n.ethz.ch/~falkbe/
- **(Extra) Demos on GitHub**: github.com/falkbe
- My exercise slides differ from the official ones presented on moodle, marked with a **blue heading**
- For the exam **only** the official exercise slides are relevant, if in doubt always check the ones on the official moodle page

Agenda

Systems@ETH zürich

- Remarks: Theory Assignment6
- Lecture Exploit Recap
- Demo Exploits
- Introduction Attack Lab

- Exam Tasks

Course Overview

Systems@ETH zürich

- **1. Programming Language C**
  - History and Toolchain, C Integers, Pointers, Dynamic Memory Allocation
- **2. Assembly, x86, Linking and Loading**:
  - x86 Architecture, Compiling C, Coroutines, Linking, Attacks, FP, Optimising Compilers
- **3. Computer Architecture: Processor Design, Exceptions, Virtual Memory**
  - CPU Architecture, Caches, Exceptions, Virtual Memory, Multiprocessing, Devices

In this session…

- Questions Regarding Linking
- Theory on Exploits
- Preview assignment 7: Attack-Lab

Q&A

Systems@ETH zürich

General linking questions?

Compilation Quiz: Which is the correct ordering of the compilation cycle?

Systems@ETH zürich

A) ☐ `foo.c` → `foo.o` → `foo.s` → `foo`

B) ☐ `foo.h` → `foo.o` → `foo.s` → `foo`

C) ☐ `foo.c` → `foo.so` → `foo`

D) ☐ `foo.c` → `foo.s` → `foo.o` → `foo`

E) ☐ `foo.c` → `foo.o` → `foo.so` → `foo`

F) ☐ None of the above.

Compilation Quiz: Which is the correct ordering of the compilation cycle?

Systems@ETH zürich

A) ☐ `foo.c` → `foo.o` → `foo.s` → `foo`

B) ☐ `foo.h` → `foo.o` → `foo.s` → `foo`

C) ☐ `foo.c` → `foo.so` → `foo`

D) ☒ `foo.c` → `foo.s` → `foo.o` → `foo`

E) ☐ `foo.c` → `foo.o` → `foo.so` → `foo`

F) ☐ None of the above.

Compilation Quiz: Which is ***not*** a section of an ELF File?

Systems@ETH zürich

A) ☐ `.code`

B) ☐ `.data`

C) ☐ `.bss`

D) ☐ `.text`

E) ☐ `.rodata`

Compilation Quiz: Which is ***not*** a section of an ELF File?

Systems@ETH zürich

A) ☒ `.code`

B) ☐ `.data`

C) ☐ `.bss`

D) ☐ `.text`

E) ☐ `.rodata`

Relocating Absolute References

Systems@ETH zürich

Object file with relocation entries (shown in italic) of the swap.o file:

```c
int buf[2] = {1, 2};

int main()
{
  swap();
  return 0;
}
```
main.c

```c
extern int buf[];

static int *bufp0 = &buf[0];
static int *bufp1;

void swap()
{
  int temp;
  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```
swap.c

```
1  0000000000000000 <swap>:
2     0:   48 b8 00 00 00 00 00    movabs $0x0,%rax
3  2:   R_X86_64_64 buf + 4
4     7:   00 00 00
5     a:   48 a3 00 00 00 00 00    movabs %rax,0x0
6  c:   R_X86_64_64 bufp1 + 0
7    11:   00 00 00
8    14:   48 b8 00 00 00 00 00    movabs $0x0,%rax
9  16:  R_X86_64_64 bufp0 + 0
10   1b:   00 00 00
11   1e:   48 8b 10                mov    (%rax),%rdx
12   21:   48 b8 00 00 00 00 00    movabs $0x0,%rax
13 23:  R_X86_64_64 buf + 0
14   28:   00 00 00
15   2b:   8b 70 04                mov    0x4(%rax),%esi
16   2e:   8b 0a                   mov    (%rdx),%ecx
17   30:   89 32                   mov    %esi,(%rdx)
18   32:   89 48 04                mov    %ecx,0x4(%rax)
19   35:   c3                      retq
```

Relocating Absolute References

Systems@ETH zürich

Upon linking the executable contains the following .text & .data

```c
int buf[2] = {1, 2};

int main()
{
  swap();
  return 0;
}
```
main.c

```c
extern int buf[];

static int *bufp0 = &buf[0];
static int *bufp1;

void swap()
{
  int temp;
  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```
swap.c

```
1  -------- Relocated .text section --------
2  0000000000400510 <swap>:
3    400510:       48 b8 3c 10 60 00 00    movabs $0x60103c,%rax
4    400517:       00 00 00
5    40051a:       48 a3 50 10 60 00 00    movabs %rax,0x601050
6    400521:       00 00 00
7    400524:       48 b8 40 10 60 00 00    movabs $0x601040,%rax
8    40052b:       00 00 00
9    40052e:       48 8b 10                mov    (%rax),%rdx
10   400531:       48 b8 38 10 60 00 00    movabs $0x601038,%rax
11   400538:       00 00 00
12   40053b:       8b 70 04                mov    0x4(%rax),%esi
13   40053e:       8b 0a                   mov    (%rdx),%ecx
14   400540:       89 32                   mov    %esi,(%rdx)
15   400542:       89 48 04                mov    %ecx,0x4(%rax)
16   400545:       c3                      retq
17   400546:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
18   40054d:       00 00 00
19
20 -------- Relocated .data section --------
21 0000000000601038 <buf>:
22   601038:       01 00 00 00 02 00 00 00
23
24 0000000000601040 <bufp0>:
25   601040:       38 10 60 00 00 00 00 00
```

Relocating Absolute References

Systems@ETH zürich

Line number	Address	Value
3	0x0000000000400512	0x000000000060103C
5	0x000000000040051C	0x0000000000601050
7		
10		

```
1  -------- Relocated .text section --------
2  0000000000400510 <swap>:
3    400510:       48 b8 3c 10 60 00 00    movabs $0x60103c,%rax
4    400517:       00 00 00
5    40051a:       48 a3 50 10 60 00 00    movabs %rax,0x601050
6    400521:       00 00 00
7    400524:       48 b8 40 10 60 00 00    movabs $0x601040,%rax
8    40052b:       00 00 00
9    40052e:       48 8b 10                mov    (%rax),%rdx
10   400531:       48 b8 38 10 60 00 00    movabs $0x601038,%rax
11   400538:       00 00 00
12   40053b:       8b 70 04                mov    0x4(%rax),%esi
13   40053e:       8b 0a                   mov    (%rdx),%ecx
14   400540:       89 32                   mov    %esi,(%rdx)
15   400542:       89 48 04                mov    %ecx,0x4(%rax)
16   400545:       c3                      retq
17   400546:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
18   40054d:       00 00 00
19
20 -------- Relocated .data section --------
21 0000000000601038 <buf>:
```

Relocating Absolute References

Systems@ETH zürich

Line number	Address	Value
3	0x0000000000400512	0x000000000060103C
5	0x000000000040051C	0x0000000000601050
7	0x0000000000400526	0x0000000000601040
10		

```
1  -------- Relocated .text section --------
2  0000000000400510 <swap>:
3    400510:       48 b8 3c 10 60 00 00    movabs $0x60103c,%rax
4    400517:       00 00 00
5    40051a:       48 a3 50 10 60 00 00    movabs %rax,0x601050
6    400521:       00 00 00
7    400524:       48 b8 40 10 60 00 00    movabs $0x601040,%rax
8    40052b:       00 00 00
9    40052e:       48 8b 10                mov    (%rax),%rdx
10   400531:       48 b8 38 10 60 00 00    movabs $0x601038,%rax
11   400538:       00 00 00
12   40053b:       8b 70 04                mov    0x4(%rax),%esi
13   40053e:       8b 0a                   mov    (%rdx),%ecx
14   400540:       89 32                   mov    %esi,(%rdx)
15   400542:       89 48 04                mov    %ecx,0x4(%rax)
16   400545:       c3                      retq
17   400546:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
18   40054d:       00 00 00
19
20 -------- Relocated .data section --------
21 0000000000601038 <buf>:
```

Relocating Absolute References

Systems@ETH zürich

Line number	Address	Value
3	0x0000000000400512	0x000000000060103C
5	0x000000000040051C	0x0000000000601050
7	0x0000000000400526	0x0000000000601040
10		

```
1  -------- Relocated .text section --------
2  0000000000400510 <swap>:
3    400510:       48 b8 3c 10 60 00 00    movabs $0x60103c,%rax
4    400517:       00 00 00
5    40051a:       48 a3 50 10 60 00 00    movabs %rax,0x601050
6    400521:       00 00 00
7    400524:       48 b8 40 10 60 00 00    movabs $0x601040,%rax
8    40052b:       00 00 00
9    40052e:       48 8b 10                mov    (%rax),%rdx
10   400531:       48 b8 38 10 60 00 00    movabs $0x601038,%rax
11   400538:       00 00 00
12   40053b:       8b 70 04                mov    0x4(%rax),%esi
13   40053e:       8b 0a                   mov    (%rdx),%ecx
14   400540:       89 32                   mov    %esi,(%rdx)
15   400542:       89 48 04                mov    %ecx,0x4(%rax)
16   400545:       c3                      retq
17   400546:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
18   40054d:       00 00 00
```

Relocating Absolute References

Systems@ETH zürich

Line number	Address	Value
3	0x0000000000400512	0x000000000060103C
5	0x000000000040051C	0x0000000000601050
7	0x0000000000400526	0x0000000000601040
10	0x0000000000400533	0x0000000000601038

```
1  -------- Relocated .text section --------
2  0000000000400510 <swap>:
3    400510:       48 b8 3c 10 60 00 00    movabs $0x60103c,%rax
4    400517:       00 00 00
5    40051a:       48 a3 50 10 60 00 00    movabs %rax,0x601050
6    400521:       00 00 00
7    400524:       48 b8 40 10 60 00 00    movabs $0x601040,%rax
8    40052b:       00 00 00
9    40052e:       48 8b 10                mov    (%rax),%rdx
10   400531:       48 b8 38 10 60 00 00    movabs $0x601038,%rax
11   400538:       00 00 00
12   40053b:       8b 70 04                mov    0x4(%rax),%esi
13   40053e:       8b 0a                   mov    (%rdx),%ecx
14   400540:       89 32                   mov    %esi,(%rdx)
15   400542:       89 48 04                mov    %ecx,0x4(%rax)
16   400545:       c3                      retq
17   400546:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
18   40054d:       00 00 00
```

Relocating Absolute References

Systems@ETH zürich

Now we should find for each relocated reference:

- the corresponding line number
- memory address of relocation
- value of relocated address

Line number	Address	Value
3	0x0000000000400512	0x000000000060103C
5	0x000000000040051C	0x0000000000601050
7	0x0000000000400526	0x0000000000601040
10	0x0000000000400533	0x0000000000601038

Overview of this session

- Questions Regarding Linking
- Theory on Exploits
- Preview assignment 7: Attack-Lab

Systems@ETH Zürich

Theory Assignment6

Linking

Task1: malloc

Systems@ETH zürich

b)
```
#include <stdlib.h>

void f(int n) {
  int *x = (int *) malloc(n * sizeof(int));
  // work with x
  return;
}
```

Task1: malloc

Systems@ETH zürich

b)
```
#include <stdlib.h>

void f(int n) {
  int *x = (int *) malloc(n * sizeof(int));
  // work with x
  return;
}
```

- **Always** check if malloc failed (and returned null)
- Don’t forget to **free** the stuff you allocated

Task1: malloc

Systems@ETH zürich

c)
```
#include <stdlib.h>

/* calculates y = Ax */
int *matvec(int **A, int *x, int n) {
  int i, j;
  int *y = (int *) malloc(n * sizeof(int));

  for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
      y[i] += A[i][j] * x[j];
    }
  }

  return y;
}
```

Task1: malloc

Systems@ETH zürich

c)
```
#include <stdlib.h>

/* calculates y = Ax */
int *matvec(int **A, int *x, int n) {
  int i, j;
  int *y = (int *) malloc(n * sizeof(int));

  for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
      y[i] += A[i][j] * x[j];
    }
  }

  return y;
}
```

- **Recall**: malloc does not 0 out memory (cant make any assumptions on that)

Task2: Symbol table

Systems@ETH zürich

```c
/* main.c */
void swap();

int buf[2] = {1, 2};

int main() {
  swap();
  return 0;
}
```

```c
/* swap.c */
extern int buf[];

int *bufp0 = &buf[0];
int *bufp1;

void swap() {
  int temp;
  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

Symbol	swap.o .symtab entry?	Symbol type	Module where defined	Section
buf				
bufp0				
bufp1				
swap				
temp				

Task2: Symbol table

Systems@ETH zürich

```c
/* main.c */
void swap();

int buf[2] = {1, 2};

int main() {
  swap();
  return 0;
}
```

```c
/* swap.c */
extern int buf[];

int *bufp0 = &buf[0];
int *bufp1;

void swap() {
  int temp;
  bufp1 = &buf[1];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
```

Symbol	`swap.o .symtab` entry?	Symbol type	Module where defined	Section
`buf`	yes	extern	`main.o`	`.data`
`bufp0`	yes	global	`swap.o`	`.data`
`bufp1`	yes	global	`swap.o`	`.bss`
`swap`	yes	global	`swap.o`	`.text`
`temp`	no	–	–	–

Task2: Symbol table

Linker symbols

- Global symbols
  - Symbols defined by module *m* that can be referenced by other modules.
  - E.g.: non-`static` C functions and non-`static` global variables.
- External symbols
  - Global symbols that are referenced by module *m* but defined by some other module.
- Local symbols
  - Symbols that are defined and referenced exclusively by module *m*.
  - E.g.: C functions and variables defined with the `static` attribute.
  - **Local linker symbols are *not* local program variables**

- **Recall**: local symbols are only stuff like **static global variables** or **static functions** (i.e. functions/vars only visible in its own compilation unit)

Task4: Relocating

Systems@ETH zürich

```
1  0000000000000000 <swap>:
2     0:    48 b8 00 00 00 00 00     movabs $0x0,%rax
3  2:    R_X86_64_64 buf + 4
4     7:    00 00 00
5     a:    48 a3 00 00 00 00 00     movabs %rax,0x0
6  c:    R_X86_64_64 bufp1 + 0
7    11:    00 00 00
8    14:    48 b8 00 00 00 00 00     movabs $0x0,%rax
9  16:   R_X86_64_64 bufp0 + 0
10   1b:    00 00 00
11   1e:    48 8b 10                 mov    (%rax),%rdx
12   21:    48 b8 00 00 00 00 00     movabs $0x0,%rax
13 23:   R_X86_64_64 buf + 0
14   28:    00 00 00
15   2b:    8b 70 04                 mov    0x4(%rax),%esi
16   2e:    8b 0a                    mov    (%rdx),%ecx
17   30:    89 32                    mov    %esi,(%rdx)
18   32:    89 48 04                 mov    %ecx,0x4(%rax)
19   35:    c3                       retq
```

```
0000000000000000 <swap>:
   0:   48 b8 00 00 00 00 00    movabs $0x0,%rax
2:   R_X86_64_64 buf + 4
   7:   00 00 00
   a:   48 a3 00 00 00 00 00    movabs %rax,0x0
c:   R_X86_64_64 bufp1 + 0
  11:   00 00 00
  14:   48 b8 00 00 00 00 00    movabs $0x0,%rax
16:  R_X86_64_64 bufp0 + 0
  1b:   00 00 00
  1e:   48 8b 10                mov    (%rax),%rdx
  21:   48 b8 00 00 00 00 00    movabs $0x0,%rax
23:  R_X86_64_64 buf + 0
  28:   00 00 00
  2b:   8b 70 04                mov    0x4(%rax),%esi
  2e:   8b 0a                   mov    (%rdx),%ecx
  30:   89 32                   mov    %esi,(%rdx)
  32:   89 48 04                mov    %ecx,0x4(%rax)
  35:   c3                      retq
```

```
-------- Relocated .text section --------
0000000000400510 <swap>:
  400510:       48 b8 3c 10 60 00 00    movabs $0x60103c,%rax
  400517:       00 00 00
  40051a:       48 a3 50 10 60 00 00    movabs %rax,0x601050
  400521:       00 00 00
  400524:       48 b8 40 10 60 00 00    movabs $0x601040,%rax
  40052b:       00 00 00
  40052e:       48 8b 10                mov    (%rax),%rdx
  400531:       48 b8 38 10 60 00 00    movabs $0x601038,%rax
  400538:       00 00 00
  40053b:       8b 70 04                mov    0x4(%rax),%esi
  40053e:       8b 0a                   mov    (%rdx),%ecx
  400540:       89 32                   mov    %esi,(%rdx)
  400542:       89 48 04                mov    %ecx,0x4(%rax)
  400545:       c3                      retq
  400546:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
  40054d:       00 00 00
```

Systems@ETH zürich

- **Check** line number, address, value

Task4: Relocating

```
1  0000000000000000 <swap>:
2     0:    48 b8 00 00 00 00 00     movabs $0x0,%rax
3  2:   R_X86_64_64 buf + 4
4     7:    00 00 00
5     a:    48 a3 00 00 00 00 00     movabs %rax,0x0
6  c:   R_X86_64_64 bufp1 + 0
7    11:    00 00 00
```

```
1  -------- Relocated .text section --------
2  0000000000400510 <swap>:
3    400510:       48 b8 3c 10 60 00 00    movabs $0x60103c,%rax
4    400517:       00 00 00
5    40051a:       48 a3 50 10 60 00 00    movabs %rax,0x601050
6    400521:       00 00 00
7    400524:       48 b8 40 10 60 00 00    movabs $0x601040,%rax
```

Task4: Relocating

Systems@ETH zürich

```
1  0000000000000000 <swap>:
2     0:   48 b8 00 00 00 00 00     movabs $0x0,%rax
3  2:   R_X86_64_64 buf + 4
4     7:   00 00 00
5     a:   48 a3 00 00 00 00 00     movabs %rax,0x0
6  c:   R_X86_64_64 bufp1 + 0
7    11:   00 00 00
```

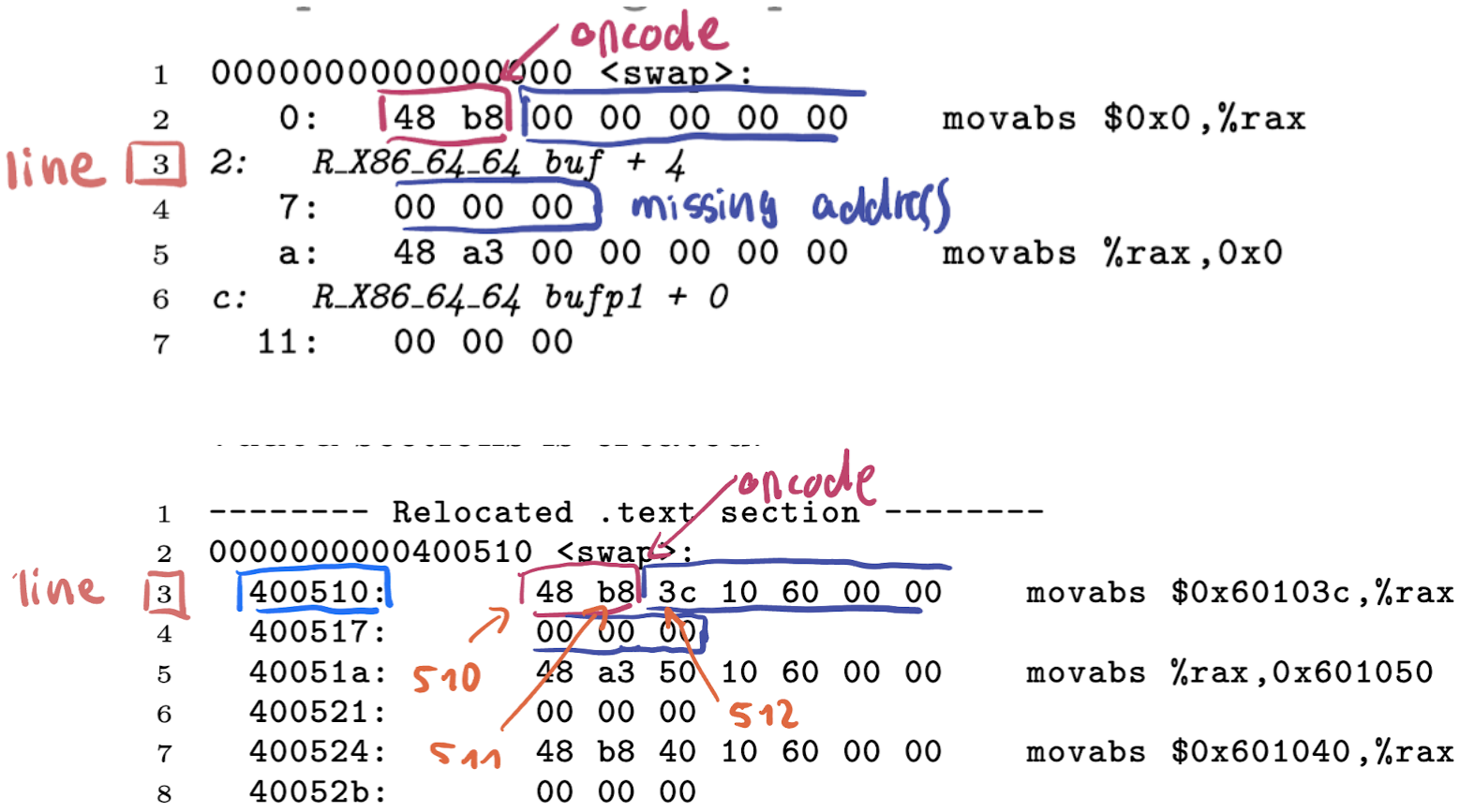
opcode

line 3

missing address

```
1  -------- Relocated .text section --------
2  0000000000400510 <swap>:
3    400510:       48 b8 3c 10 60 00 00    movabs $0x60103c,%rax
4    400517:       00 00 00
5    40051a:       48 a3 50 10 60 00 00    movabs %rax,0x601050
6    400521:       00 00 00
7    400524:       48 b8 40 10 60 00 00    movabs $0x601040,%rax
```

Task4: Relocating




```
1  0000000000000000 <swap>:
2     0:   48 b8 00 00 00 00 00     movabs $0x0,%rax
3  2:   R_X86_64_64 buf + 4
4     7:   00 00 00
5     a:   48 a3 00 00 00 00 00     movabs %rax,0x0
6  c:   R_X86_64_64 bufp1 + 0
7    11:   00 00 00
```

```
1  -------- Relocated .text section --------
2  0000000000400510 <swap>:
3    400510:       48 b8 3c 10 60 00 00     movabs $0x60103c,%rax
4    400517:       00 00 00
5    40051a:       48 a3 50 10 60 00 00     movabs %rax,0x601050
6    400521:       00 00 00
7    400524:       48 b8 40 10 60 00 00     movabs $0x601040,%rax
8    40052b:       00 00 00
```

Task4: Relocating

```
1  0000000000000000 <swap>:
2     0:   48 b8 00 00 00 00 00     movabs $0x0,%rax
3  2:   R_X86_64_64 buf + 4
4     7:   00 00 00
5     a:   48 a3 00 00 00 00 00     movabs %rax,0x0
6  c:   R_X86_64_64 bufp1 + 0
7    11:   00 00 00
```

opcode

line 3

missing address

```
1  -------- Relocated .text section --------
2  0000000000400510 <swap>:
3    400510:       48 b8 3c 10 60 00 00     movabs $0x60103c,%rax
4    400517:       00 00 00
5    40051a:       48 a3 50 10 60 00 00     movabs %rax,0x601050
6    400521:       00 00 00
7    400524:       48 b8 40 10 60 00 00     movabs $0x601040,%rax
8    40052b:       00 00 00
```

opcode

line 3

510

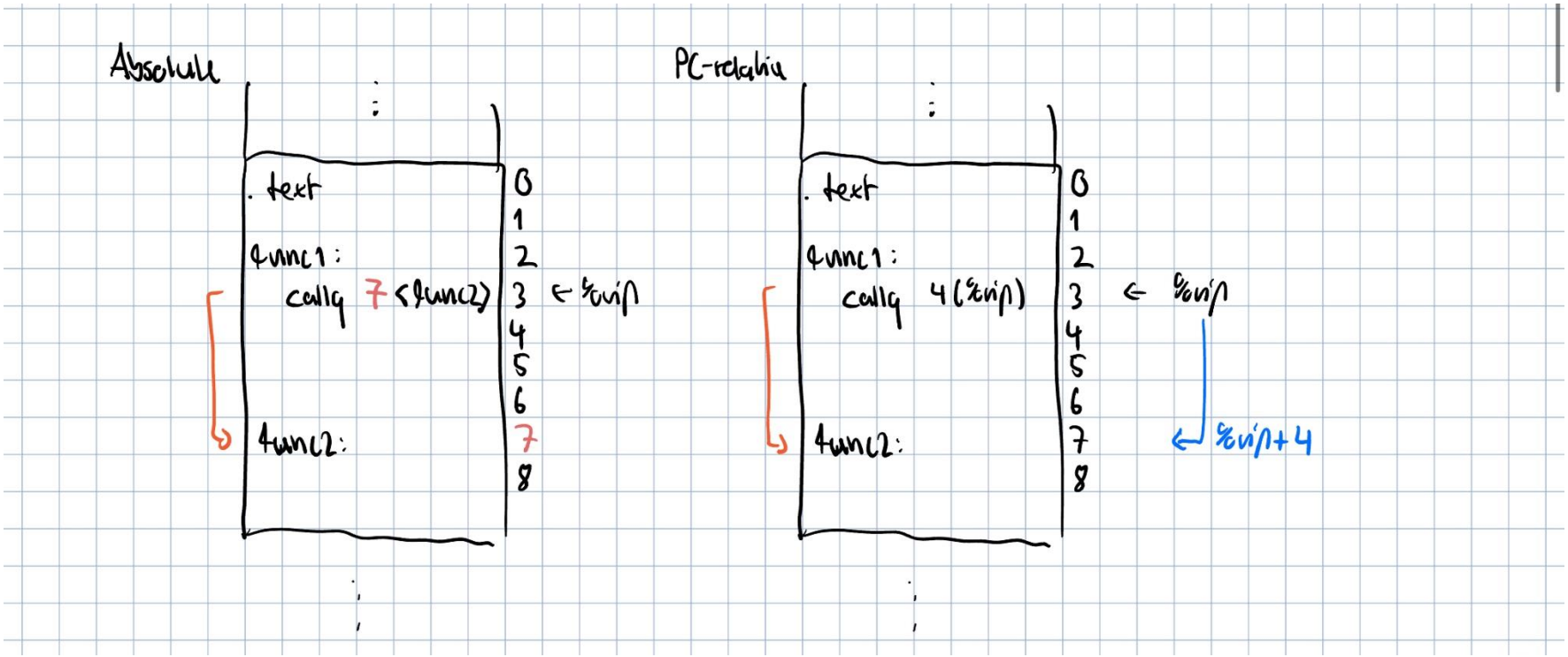
511

512

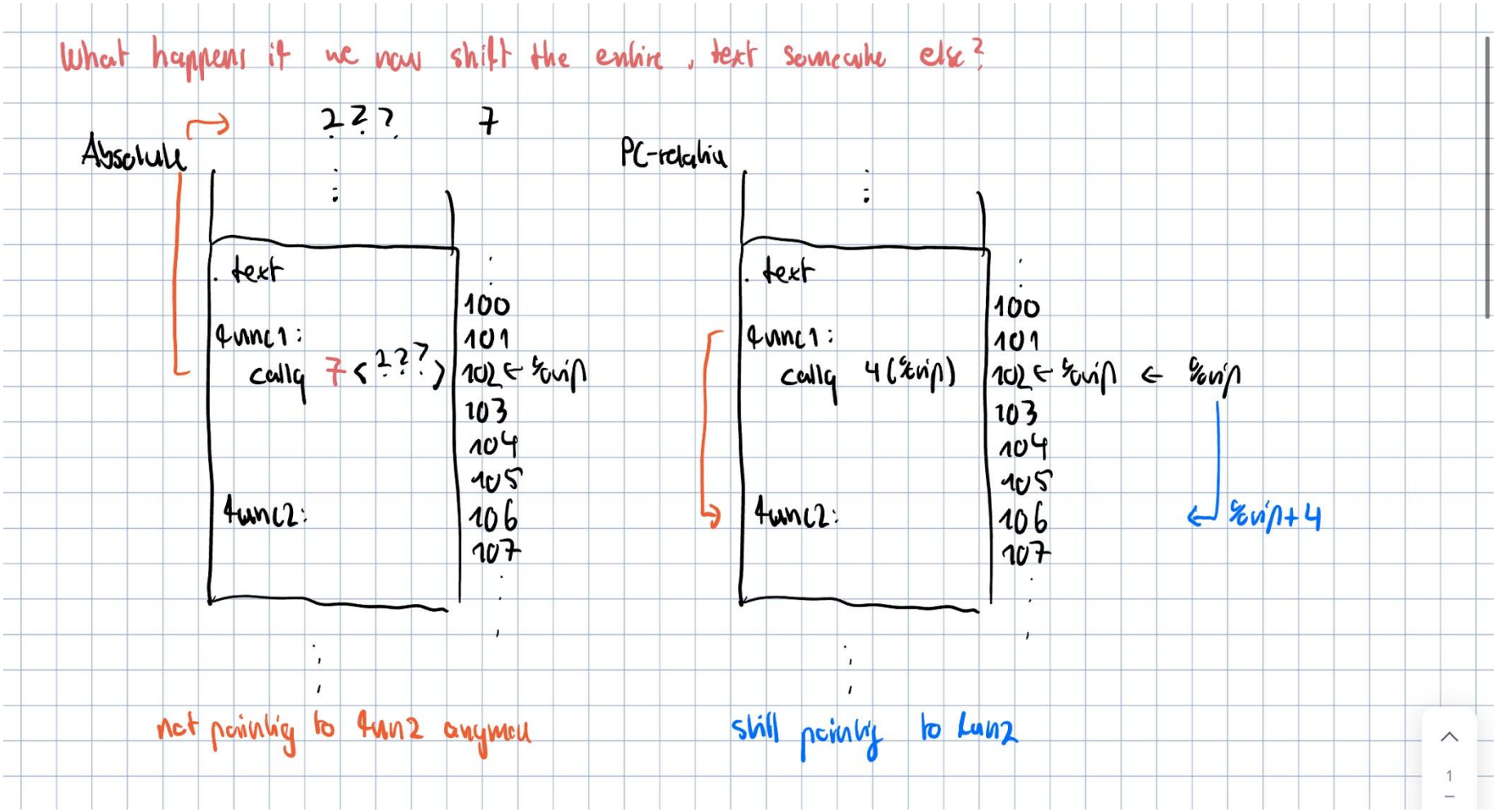
Solution:

line number	*address*	*value*
3	0x0000000000400512	0x000000000060103c

Task5: Absolute vs. PC-Relative



Task5: Absolute vs. PC-Relative



Task5: Absolute vs. PC-Relative

Systems@ETH zürich

```
1  -------- Relocated .text section --------
2  0000000000400400 <main>:
3    400400:       48 83 ec 08             sub    $0x8,%rsp
4    400404:       31 c0                   xor    %eax,%eax
5    400406:       e8 f5 00 00 00          callq  400500 <swap>
6    40040b:       31 c0                   xor    %eax,%eax
7    40040d:       48 83 c4 08             add    $0x8,%rsp
8    400411:       c3                      retq
9    400412:       66 90                   xchg   %ax,%ax
10
11 0000000000400500 <swap>:
12   400500:       48 8b 05 39 0b 20 00    mov    0x200b39(%rip),%rax        # get *bufp0 (@ 601040)
13   400507:       8b 0d 2f 0b 20 00       mov    0x200b2f(%rip),%ecx        # get buf[1] (@ 60103c)
```

c) Suppose the linker decides to locate the `.text` section at 0x800800 instead of 0x400400. What would the value inserted as the relocated reference to `swap` on line 5 be in this case?

- **This** is what they wanted to show you in 5c)
- **PC-relative** call to swap in main

Task5: Absolute vs. PC-


```
1  -------- Relocated .text section --------
2  0000000000400400 <main>:
3    400400:       48 83 ec 08             sub    $0x8,%rsp
4    400404:       31 c0                   xor    %eax,%eax
5    400406:       e8 f5 00 00 00          callq  400500 <swap>
6    40040b:       31 c0                   xor    %eax,%eax
7    40040d:       48 83 c4 08             add    $0x8,%rsp
8    400411:       c3                      retq
9    400412:       66 90                   xchg   %ax,%ax
10
11 0000000000400500 <swap>:
12   400500:       48 8b 05 39 0b 20 00    mov    0x200b39(%rip),%rax        # get *bufp0 (@ 601040)
13   400507:       8b 0d 2f 0b 20 00       mov    0x200b2f(%rip),%ecx        # get buf[1] (@ 60103c)
```

c) Suppose the linker decides to locate the `.text` section at 0x800800 instead of 0x400400. What would the value inserted as the relocated reference to `swap` on line 5 be in this case?

c) The key observation here is that no matter where the linker locates the `.text` section, the distance between the reference and the `swap` function is always the same. Thus, because the reference is a PC-relative address, its value will be `0xf5` regardless of where the linker locates the `.text` section. More generally, this form of addressing relative to the instruction pointer (e.g. `movl $0x1, 0x10(%rip)`) makes it easy to write *position independent code.*

- **This** is what they wanted to show you in 5c)
- **PC-relative** call to swap in main

Systems@ETH zürich

Lecture Recap: Code Vulnerabilities

Buffer Overflow Attacks

Why do we care?

Systems@ETH zürich

Stuxnet worm 'targeted high-value Iranian assets'

WannaCry, Petya, NotPetya: how ransomware hit the big time in 2017

Most first encountered ransomware after an outbreak shut down hospital computers and diverted ambulances this year. Is it here to stay?

Defn. Worm and Virus

Systems@ETH zürich

Worms and viruses

- ***Worm***: A program that
  - Can run by itself
  - Can propagate a fully working version of itself to other computers
- ***Virus***: Code that
  - Add itself to other programs
  - Cannot run independently
- Both are (usually) designed to spread among computers and to wreak havoc

Fundamental Issue

Systems@ETH zürich

String library code

- Implementation of Unix function `gets()`
  - No way to specify limit on number of characters to read
- Similar problems with other Unix functions
  - `strcpy`: Copies string of arbitrary length
  - `scanf`, `fscanf`, `sscanf`, when given `%s` conversion specification

```
/* Get string from stdin */
char *gets(char *dest)
{
    int c = getchar();
    char *p = dest;
    while (c != EOF && c != '\n') {
        *p++ = c;
        c = getchar();
    }
    *p = '\0';
    return dest;
}
```

Fundamental Issue

Systems@ETH zürich

Vulnerable buffer code

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

```
int main()
{
  printf("Type a string:");
  echo();
  return 0;
}
```

```
unix>./bufdemo
Type a string:123
123
```

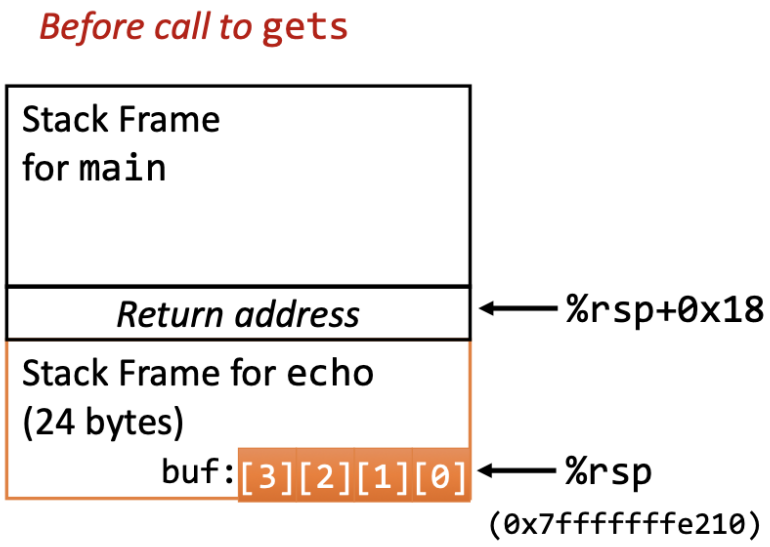
```
unix>./bufdemo
Type a string:1234567
1234567
```

```
unix>./bufdemo
Type a
string:12345678901234567890123
4
Segmentation Fault
```

Buffer Overflow Attack

Systems@ETH zürich

Buffer overflow stack




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

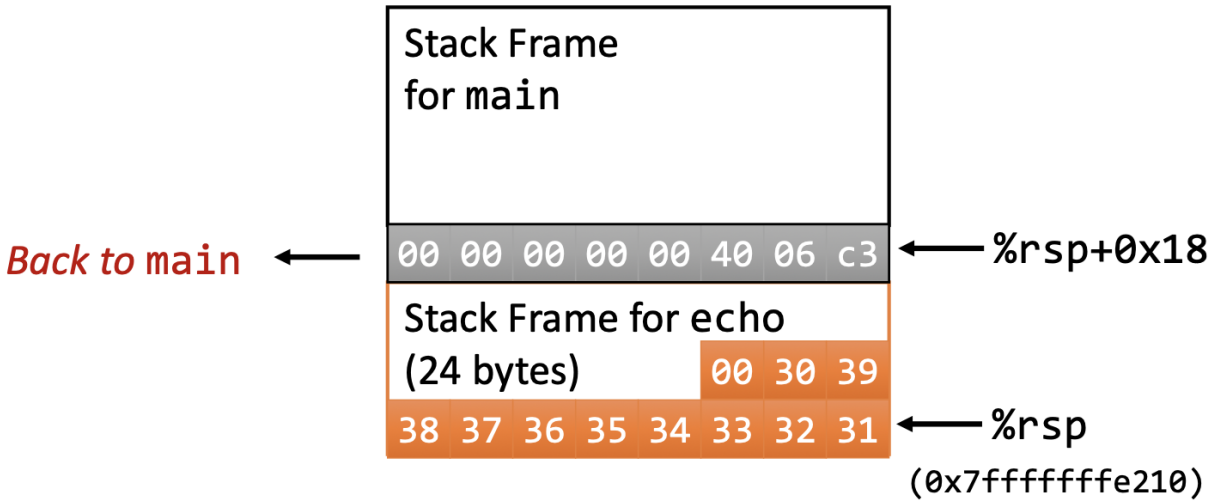
```
echo:
    subq    $24, %rsp
    movq    %rsp, %rdi
    call    gets
    movq    %rsp, %rdi
    call    puts
    addq    $24, %rsp
    ret
```

Buffer Overflow Attack

Systems@ETH zürich

Buffer overflow stack

Input: `123456790`

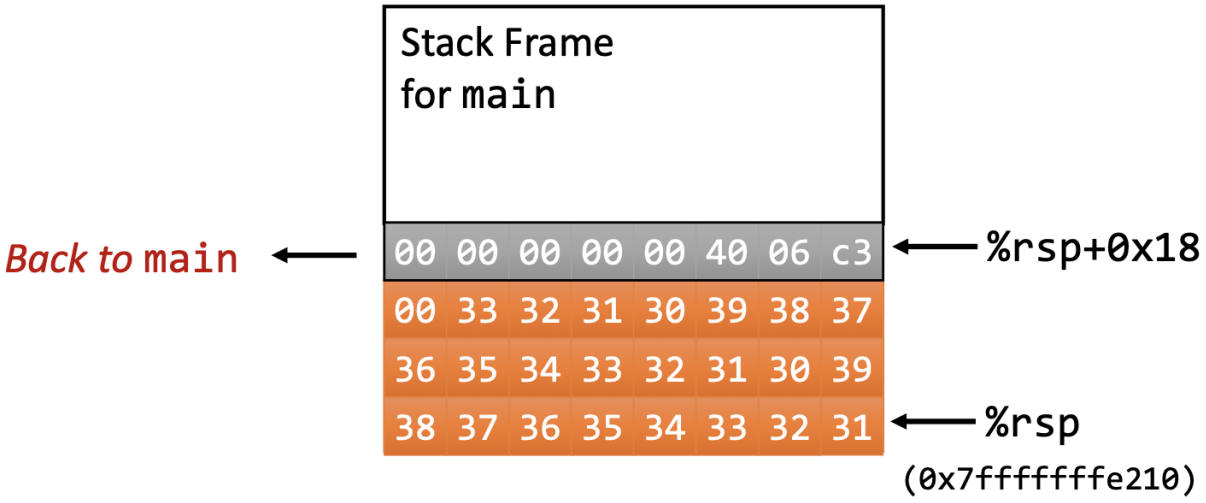


Buffer Overflow Attack

Systems@ETH zürich

Buffer overflow stack

Input: `1234567901234567901 23`

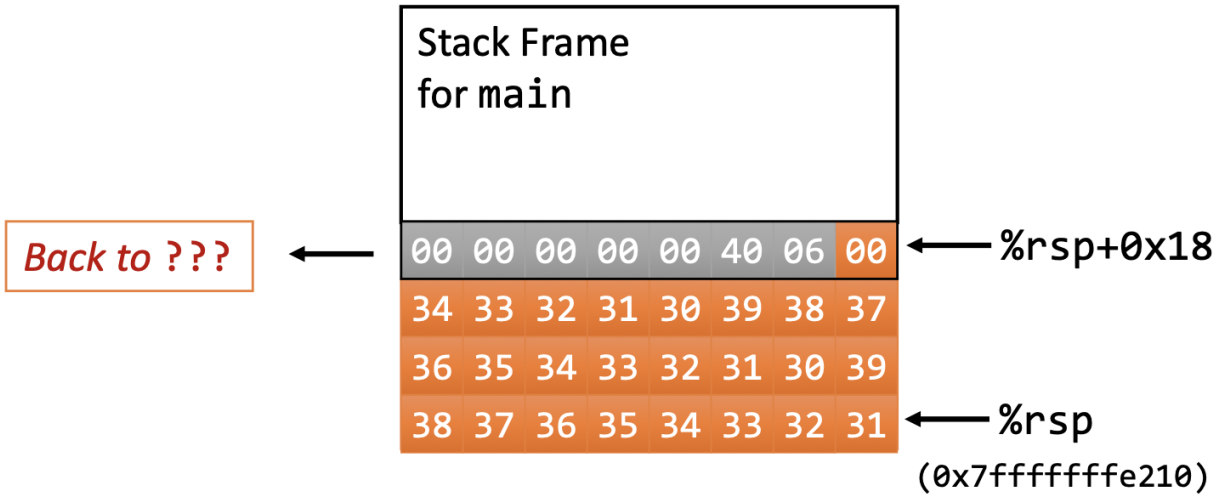


Buffer Overflow Attack

Systems@ETH zürich

Buffer overflow stack

Input: `1234567901234567901234`



Buffer Overflow Attack

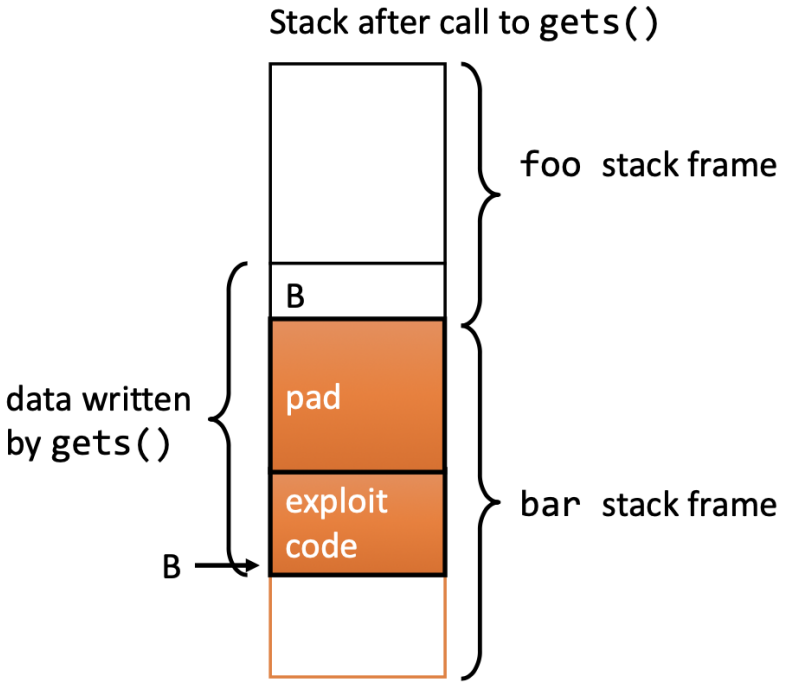
Systems@ETH zürich

Malicious use of buffer overflow

```
void foo(){
  bar();
  ...
}
```

return address A

```
int bar() {
  char buf[64];
  gets(buf);
  ...
  return ...;
}
```



- Input string contains byte representation of executable code
- Overwrite return address with address of buffer
- When `bar()` executes `ret`, will jump to exploit code

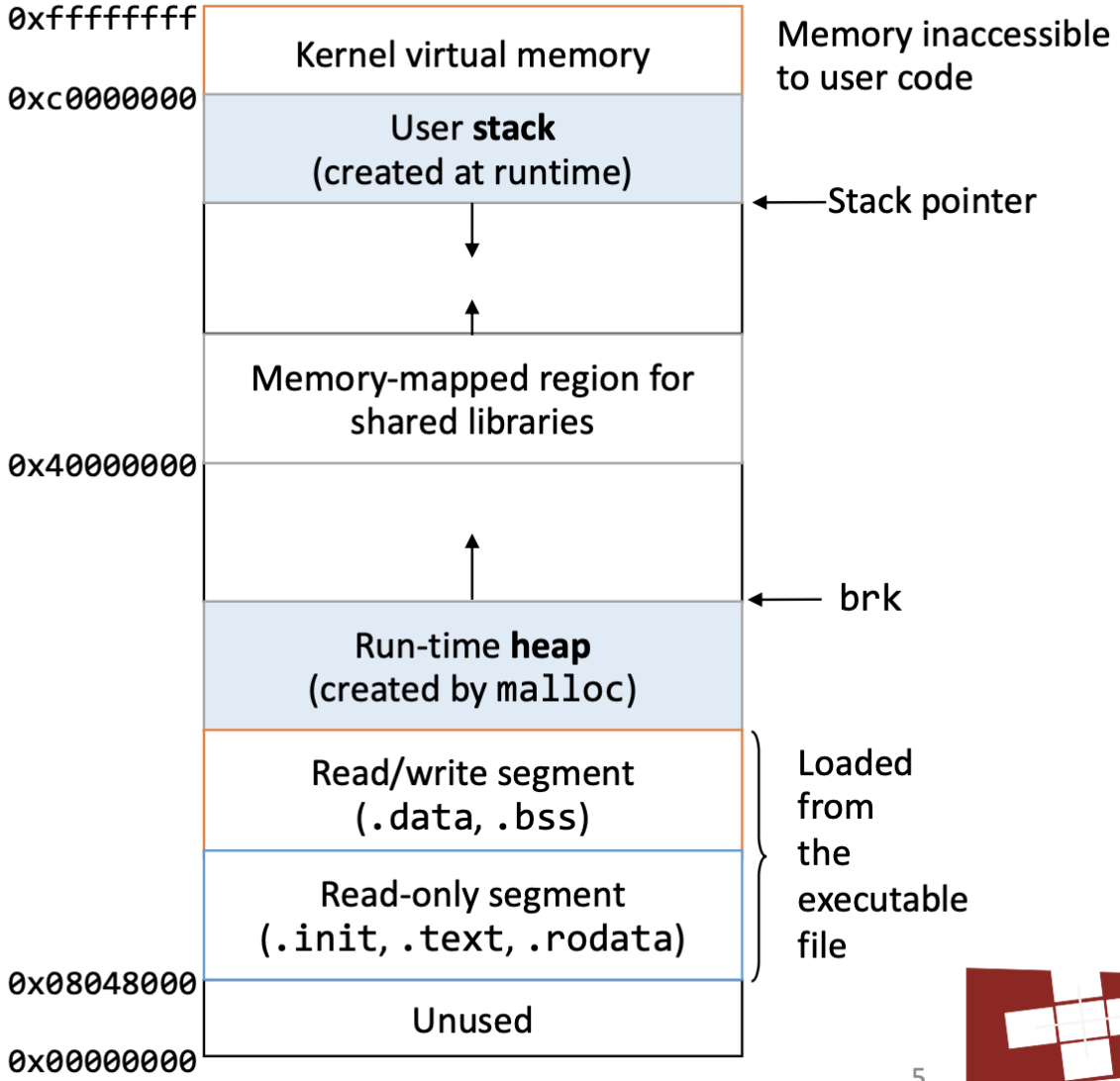
Two Types of Exploits

Systems@ETH zürich

- **Code Injection**: You write your own code in the buffer
- **Return Oriented Programming** (ROP): You take already **existing code** (we call those “gadgets”), concatenate them

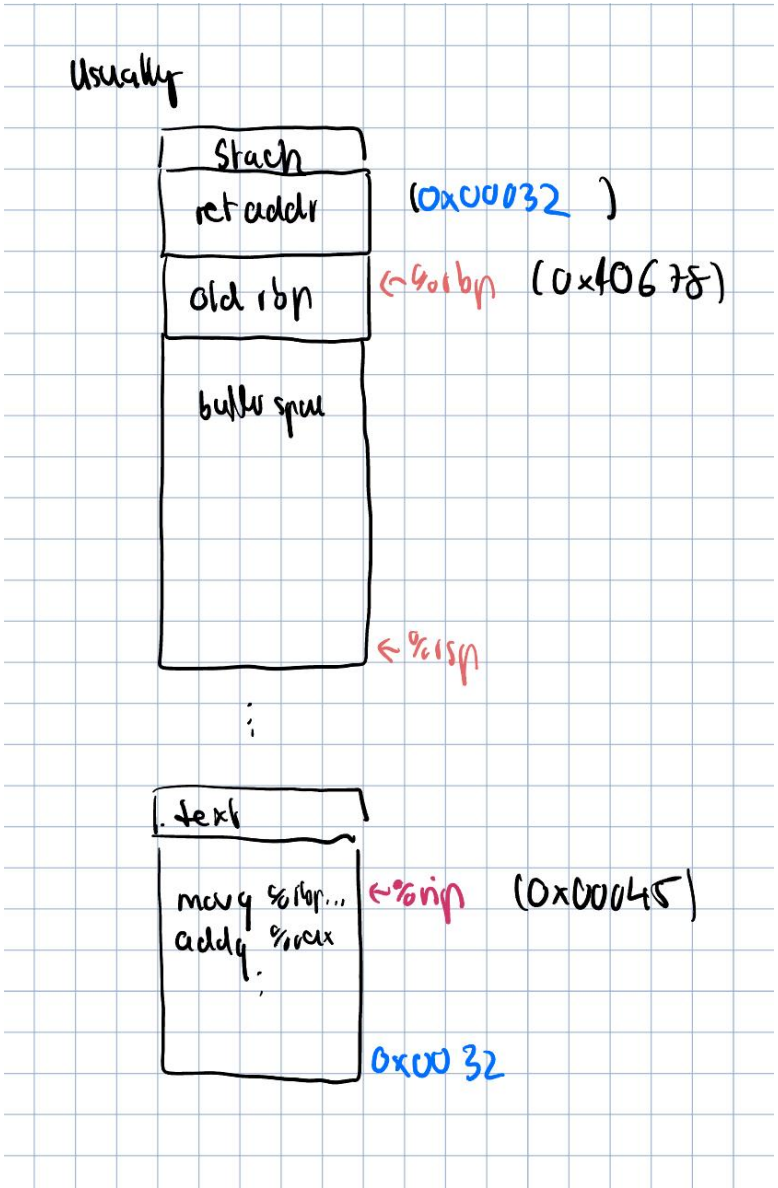
- Advantages of ROP? It doesn’t need the **stack to be executable**

Code Injections



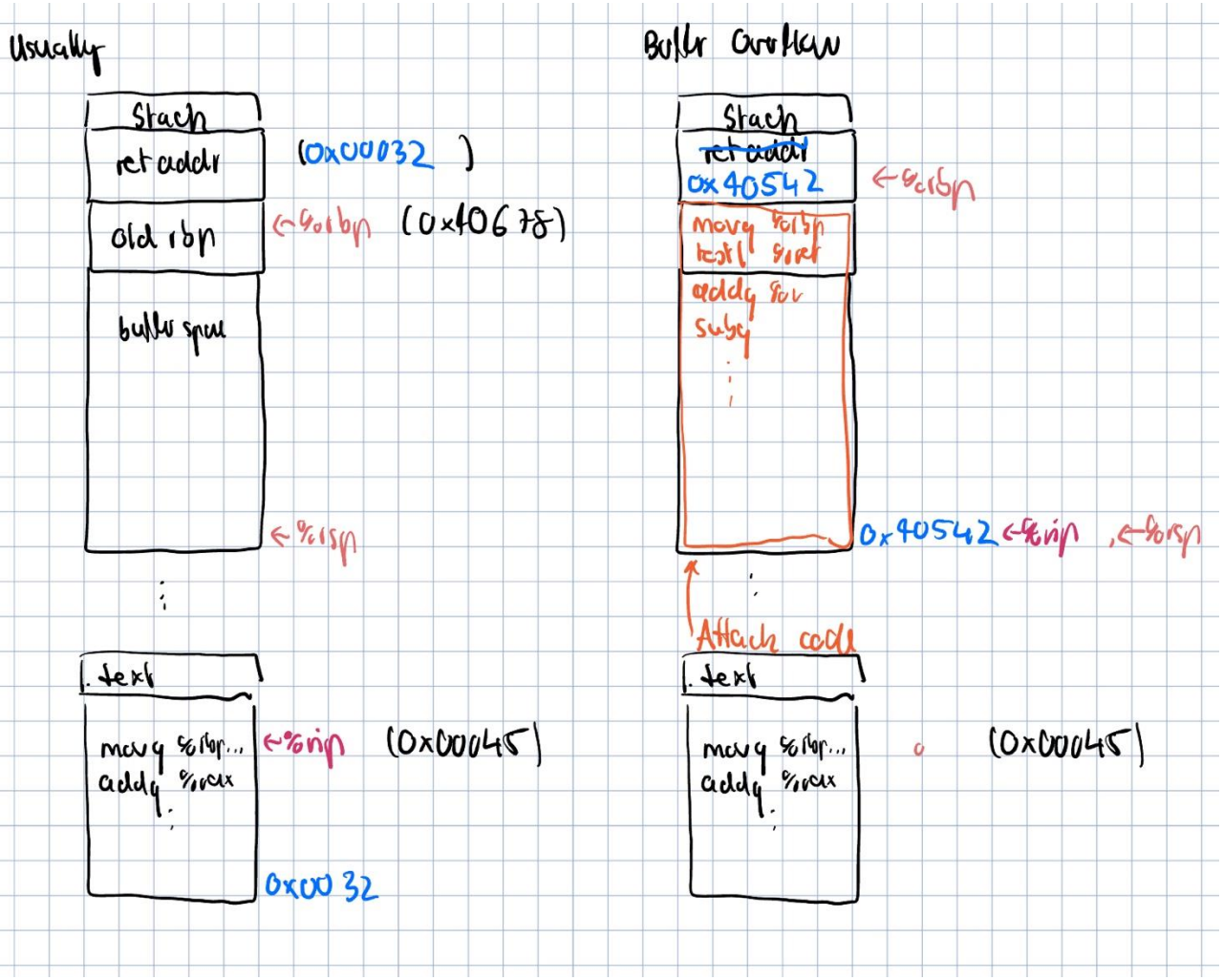
- **Usually** program in memory looks like this: we will abstract it to the following

Code Injections



- **Usually** program in memory looks like this: we will abstract it to the following

Code Injections



- **Code injections** put their own code for instance in a buffer, overwrite the ret address to point to the start
- **We now execute on the stack**

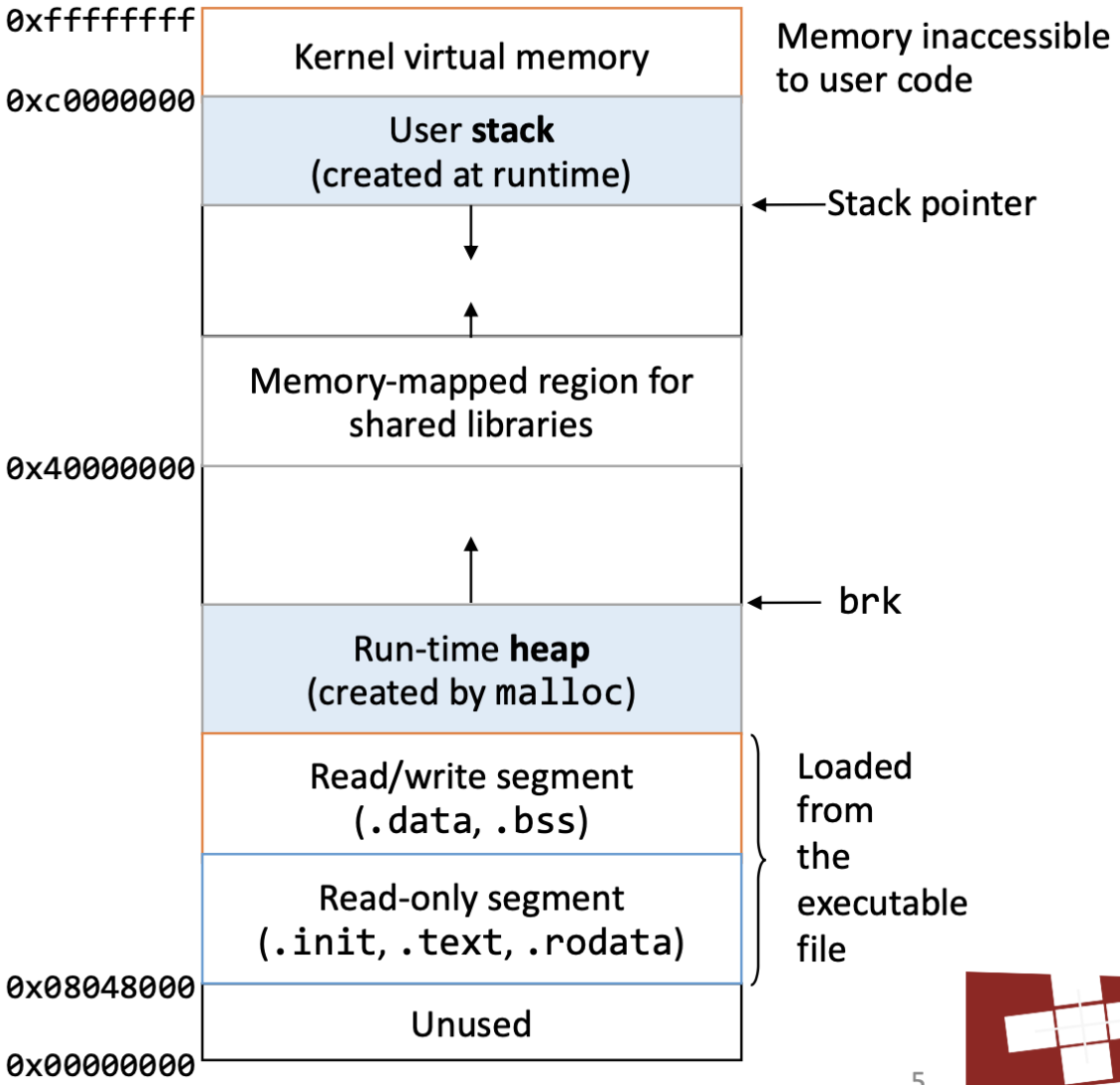
Code Injections

Systems@ETH zürich

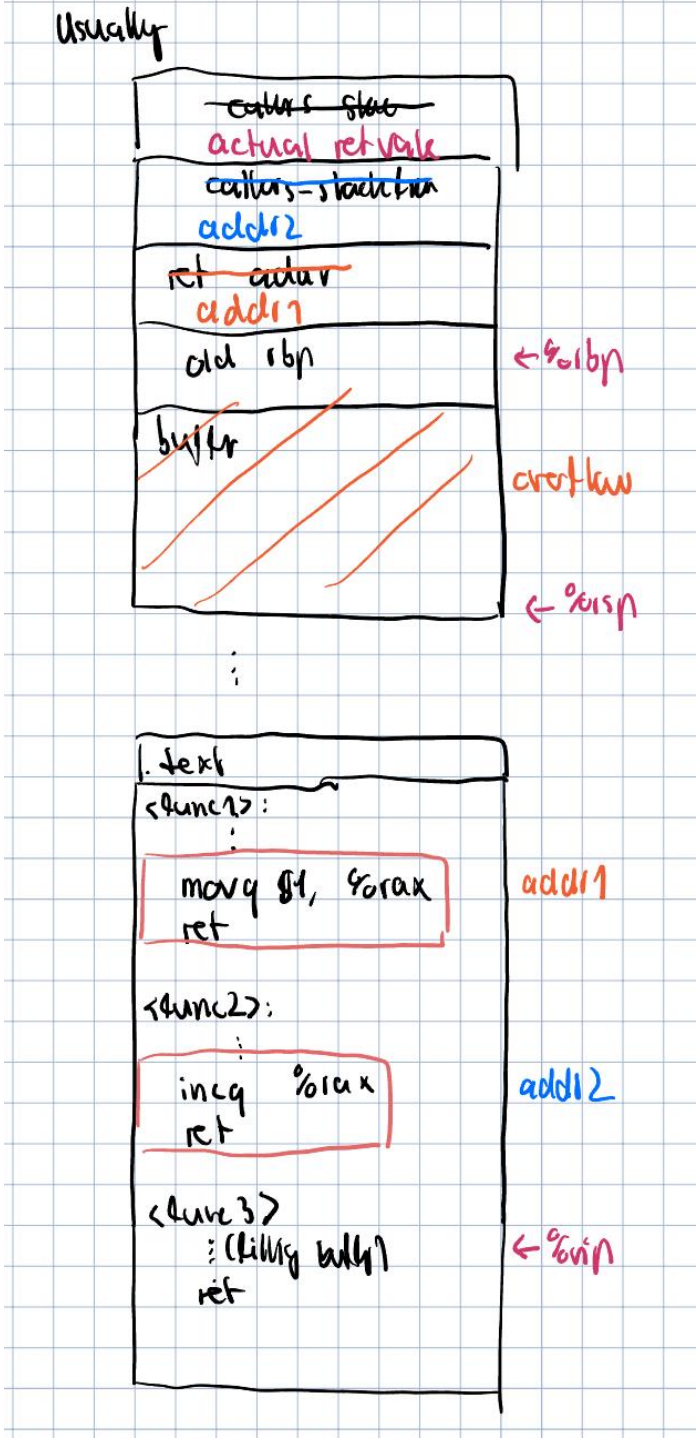
- **But** we can prevent this by **randomizing stack offset**, or **simply making the stack not executable** (i.e. if %rip points to some stack address we fail)
- So these attacks are usually easier to prevent

- This is where **ROP** comes into play: more sophisticated attack, highly non trivial if done properly on a large scale

ROP



- **Usually** program in memory looks like this: we will abstract it to the following

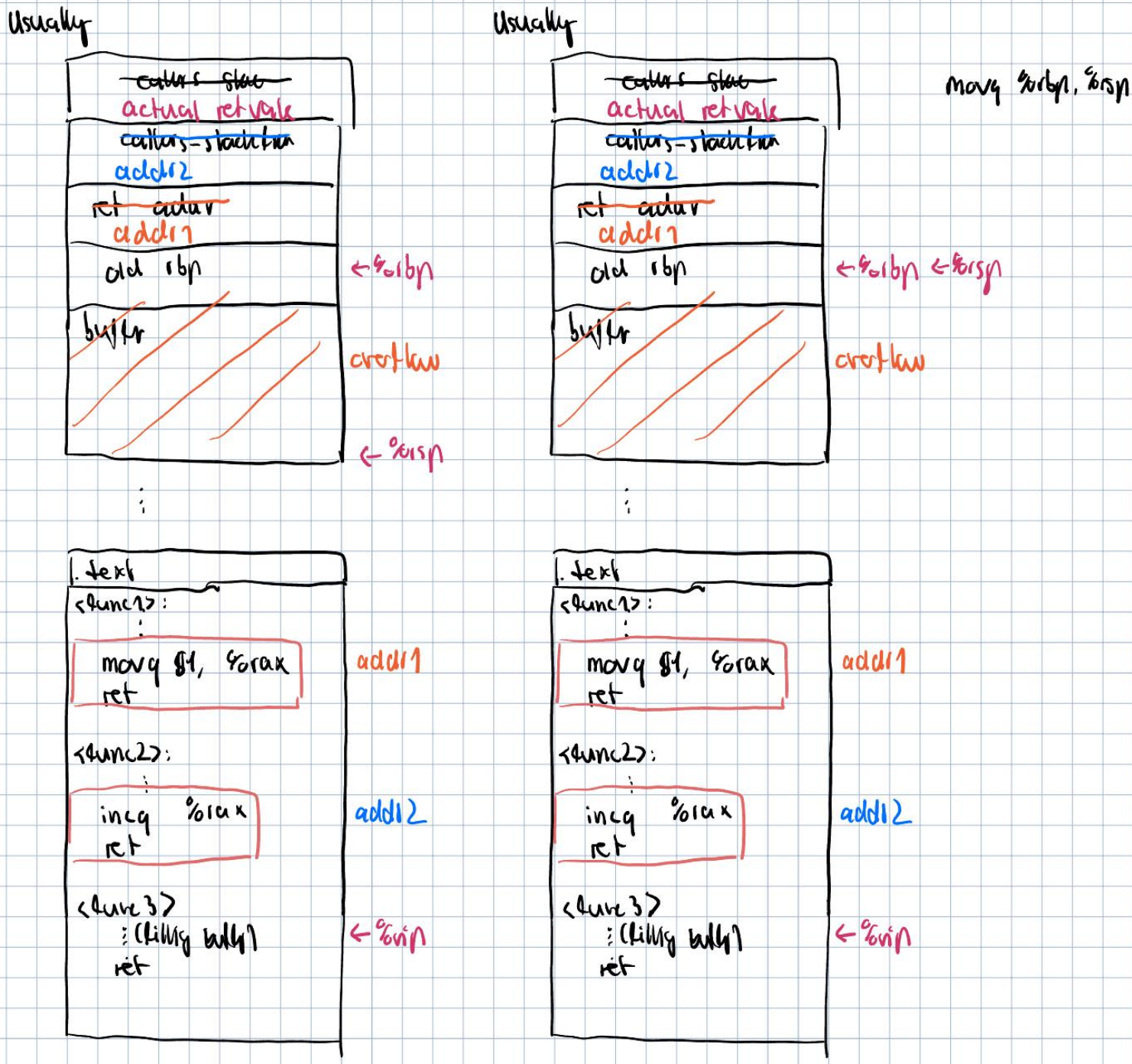


ROP

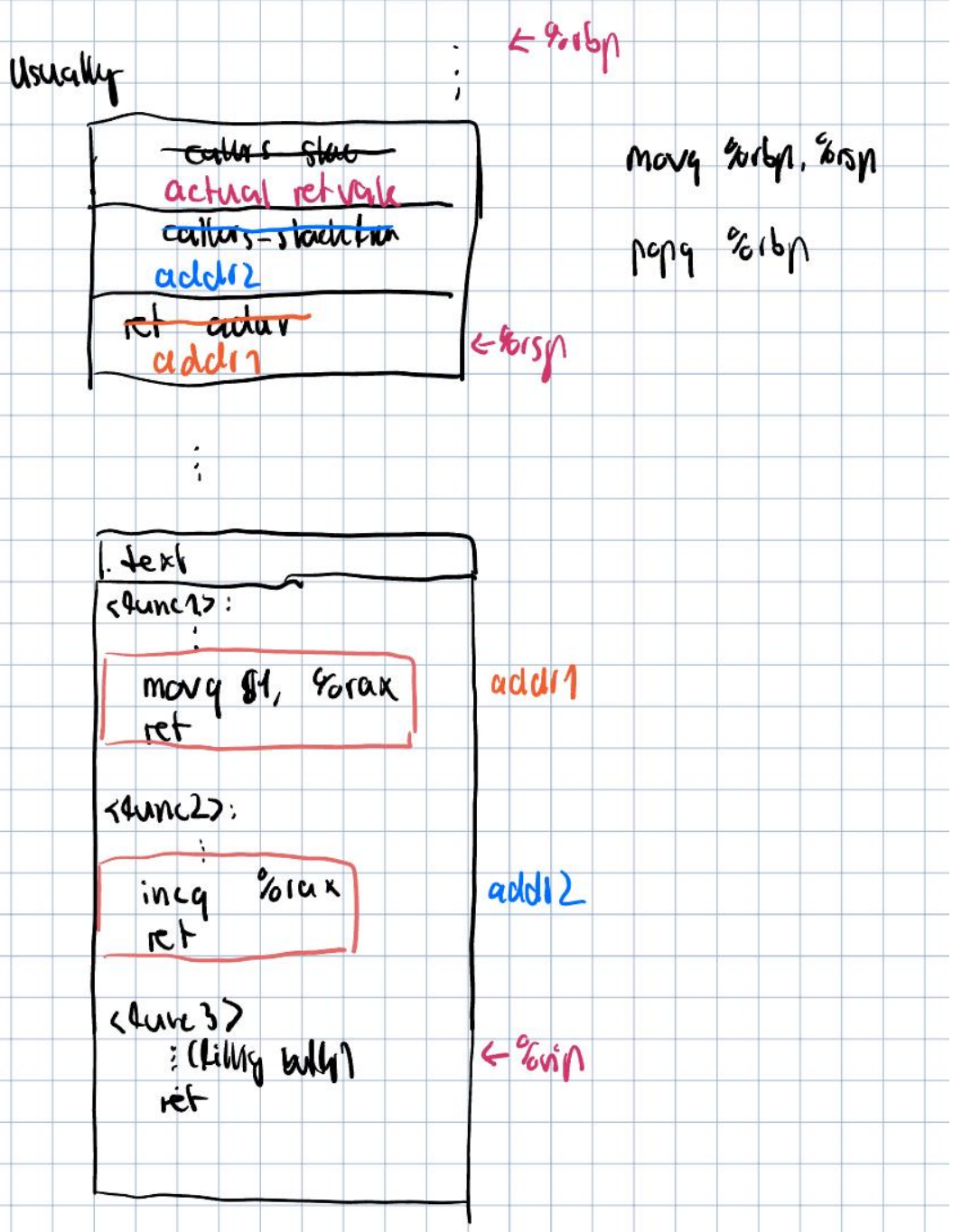
Systems@ETH zürich

- **Remember** this: we still overwrite the **return address** via buffer overflow

- What if we wanted to have **$2** in register **%rax** without writing our own code (inject)?
- **Idea**: search for assembly which does the job, and directly returns

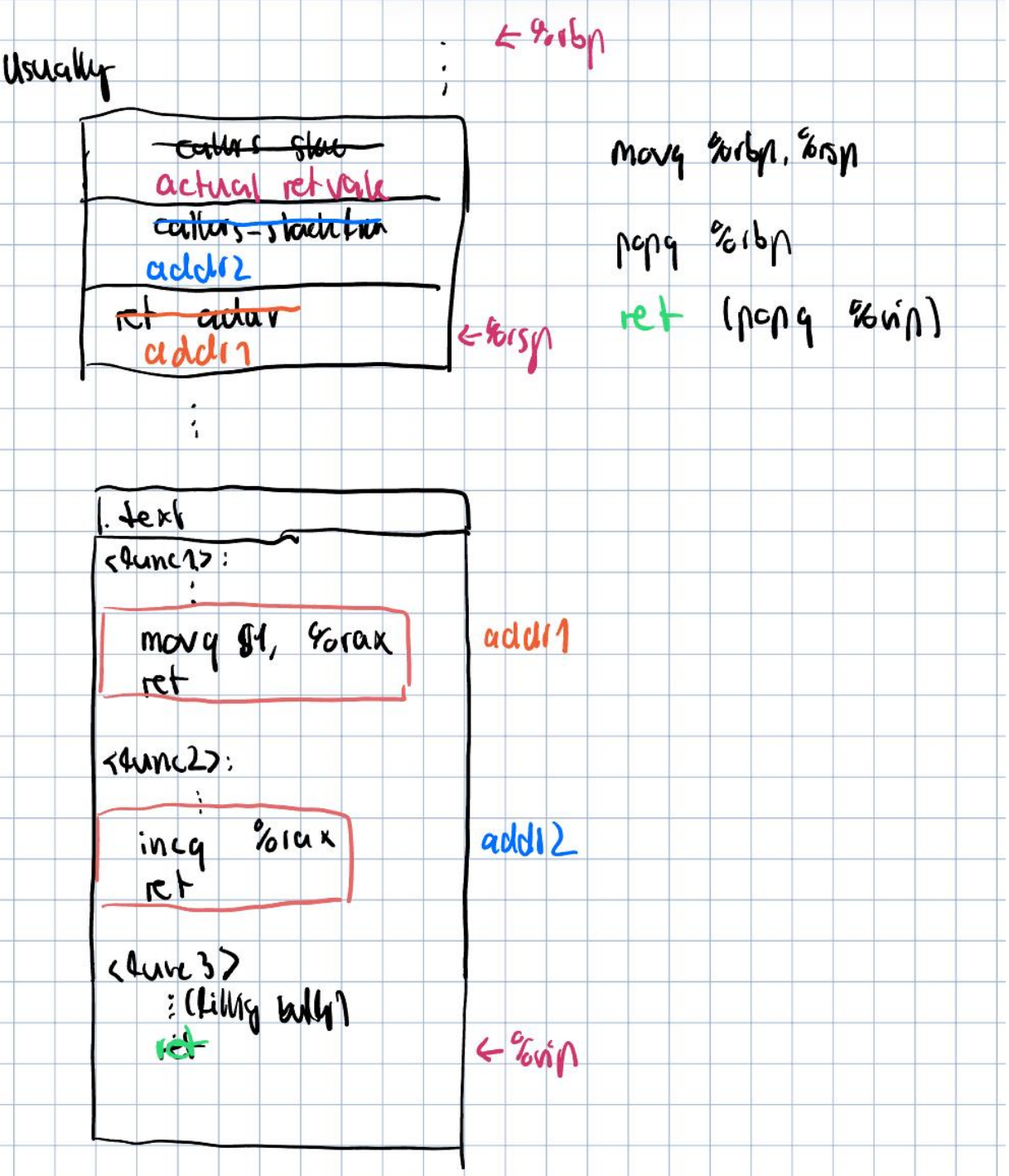

Usually
callers stack
actual ret val
callers-stack frame
addr2
ret addr
addr1
old rbp
buffer
overflow
← %rbp
← %rsp
.text
<func1>:
movq $1, %rax
ret
addr1
<func2>:
incq %rax
ret
addr2
<func3>:
; (filling bully)
ret
← %rip
Usually
callers stack
actual ret val
callers-stack frame
addr2
ret addr
addr1
old rbp
buffer
overflow
movq %rbp, %rsp
← %rbp ← %rsp
.text
<func1>:
movq $1, %rax
ret
addr1
<func2>:
incq %rax
ret
addr2
<func3>:
; (filling bully)
ret
← %rip

ROP



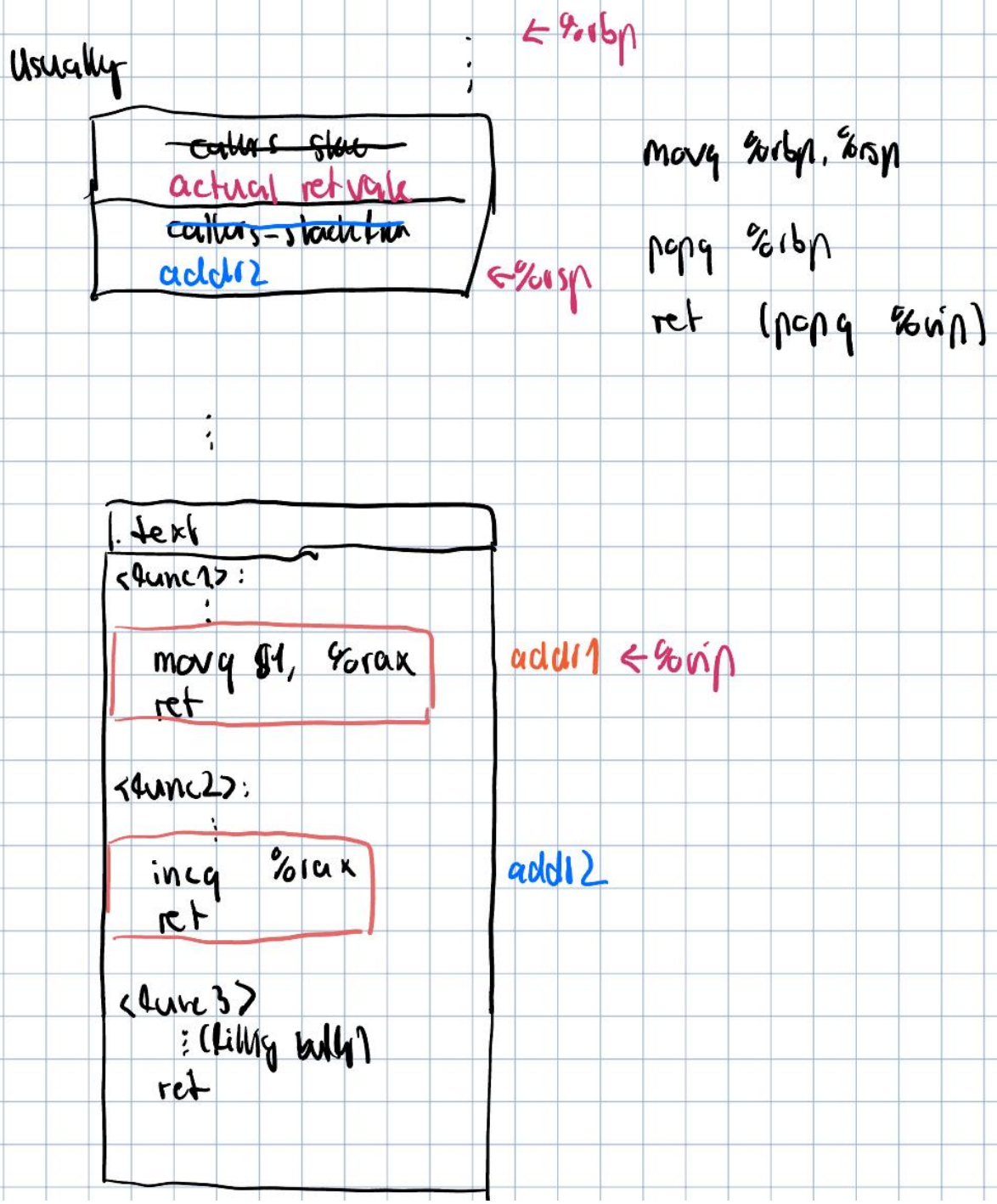
- **We** just deallocate the stackframe as usual

ROP



- **We** just deallocate the stackframe as usual
- Now we execute the return instruction: however we **changed the ret addr** via the overflow

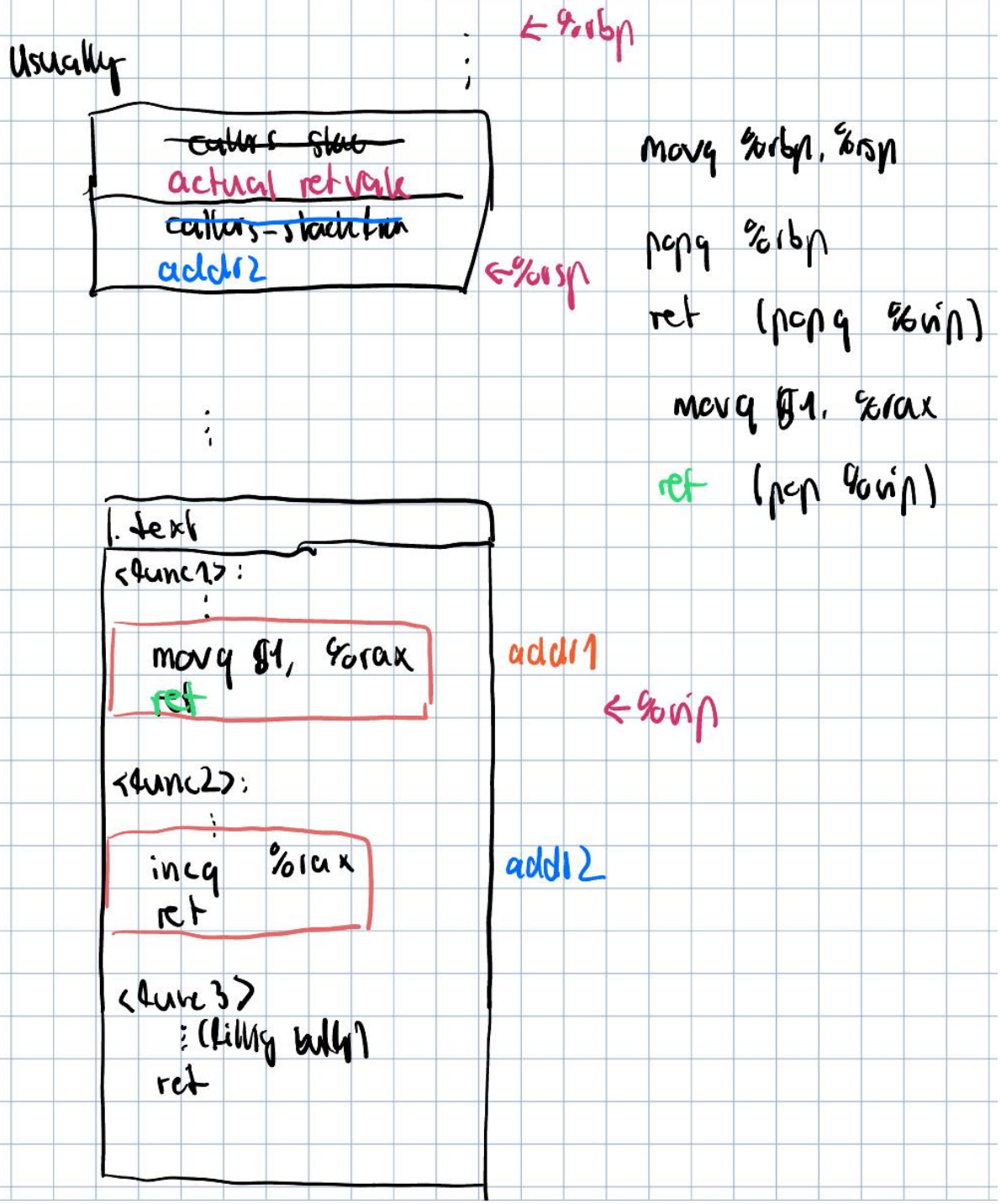
Systems@ETH zürich



ROP

- **Now %rip** points to our first **gadget**: so at addr1
- We movq $1, %rax

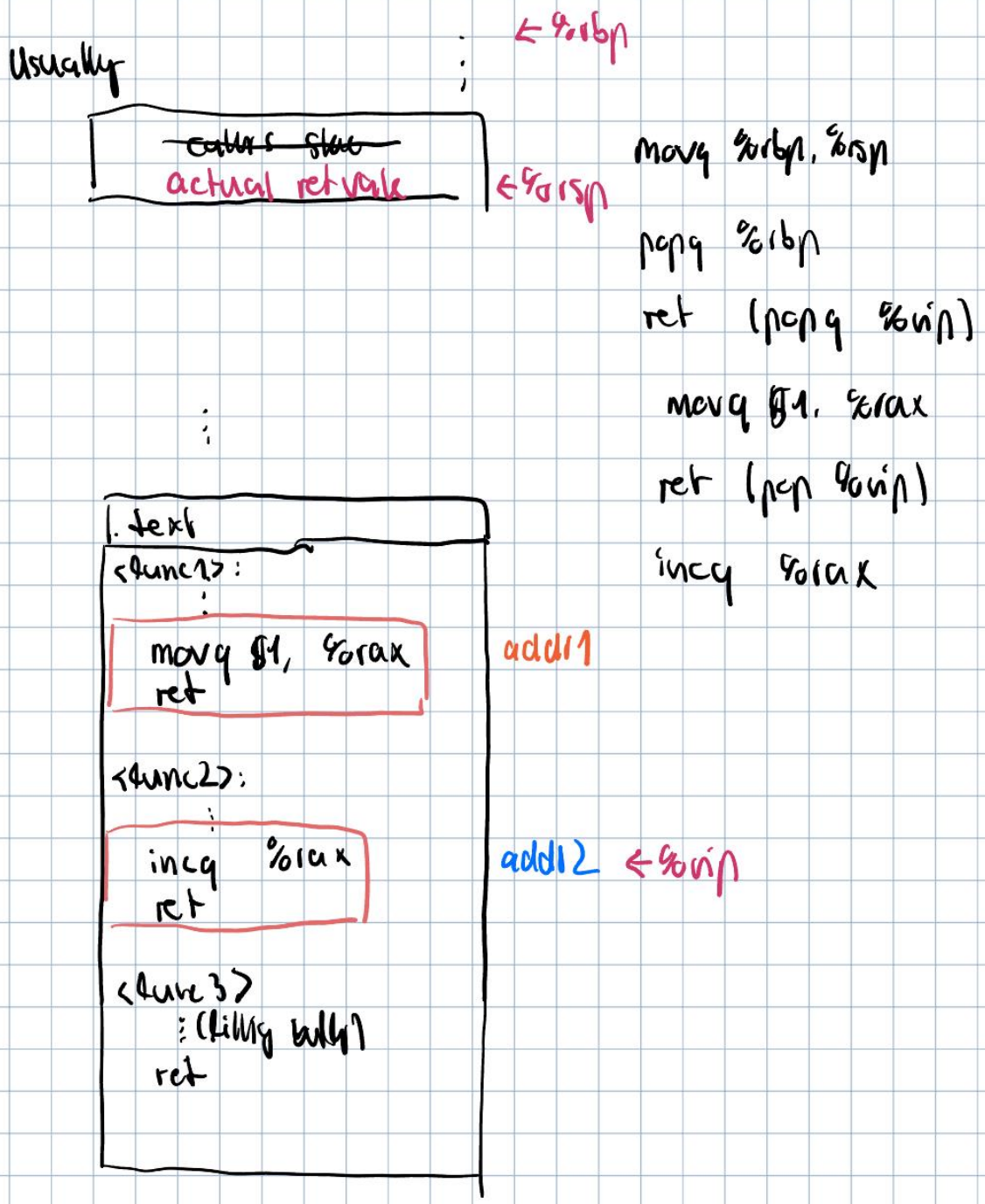
ROP



- **Now %rip** points to our first **gadget**: so at addr1
- We movq $1, %rax, and **return**

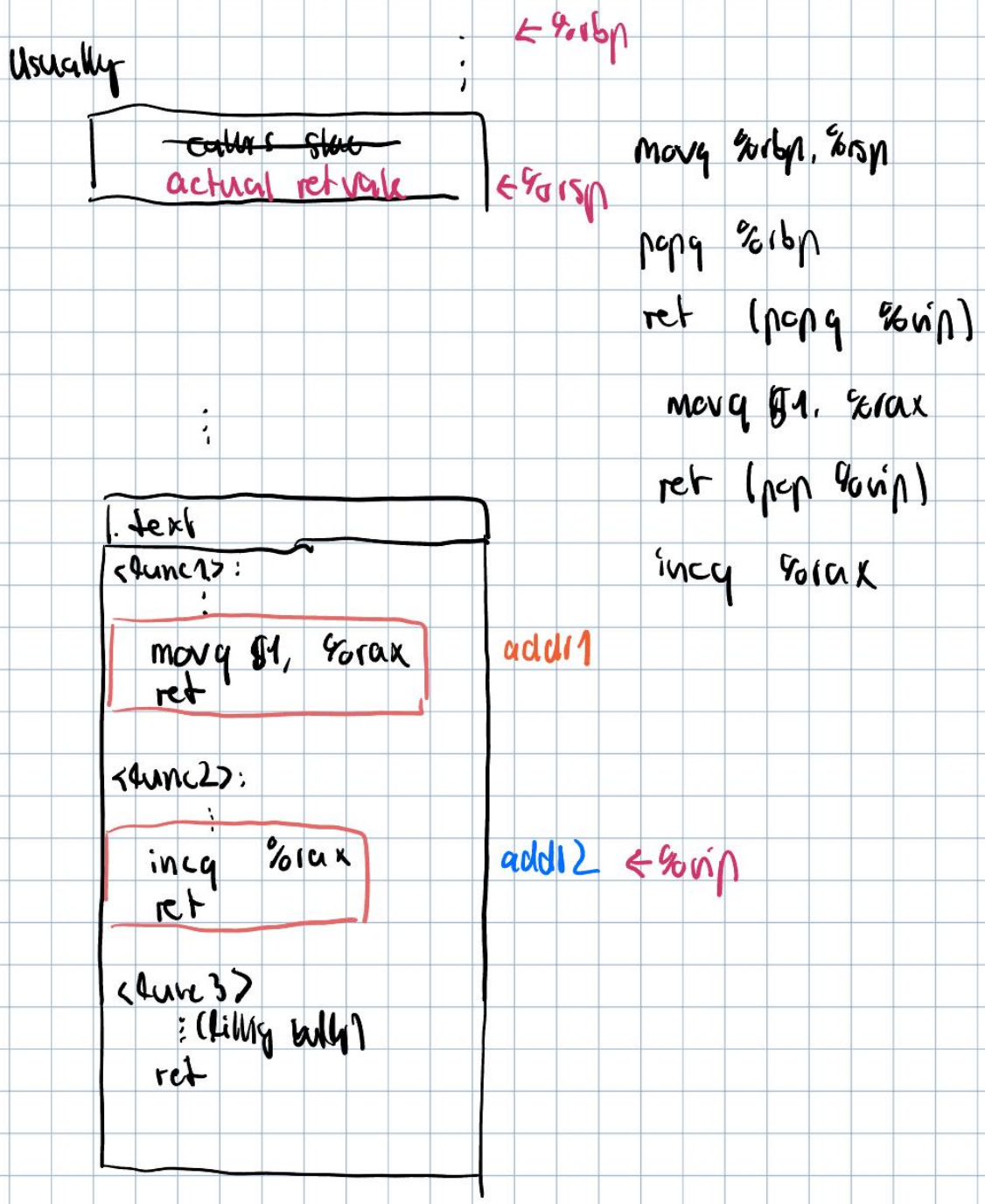
- **But what does return do**?
- Pop the current value at the %rsp in to %rip
- **We put addr2 in $rip**

ROP



- **Now %rip** points to our second **gadget**: so at addr2
- We incq %rax

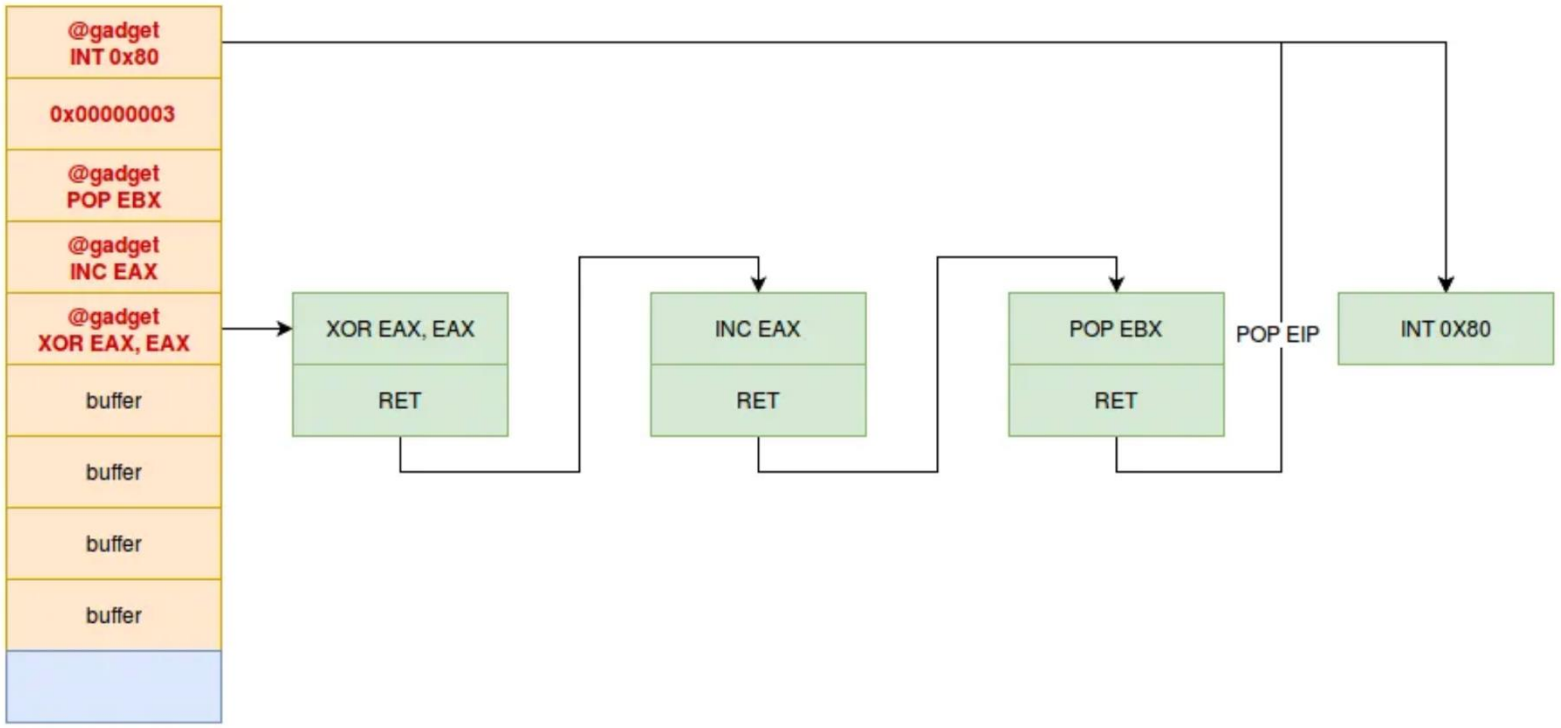
ROP



- **Now %rip** points to our second **gadget**: so at addr2
- We incq %rax, and **return**

- **We** would now return to the **actual** return address

General ROP



- **An image like above** should now make sense to you (and also the rest of official the exercise session slides)

ROP

Systems@ETH zürich

- **Notice** how we simply use already existing code snippets to execute what we want
- **Of course** this doesn't always work: if the code does not have the code we want to execute we cannot execute it

- **Notice** however how %rip **always** stays in the .text segment: the **stack does not have to be executable**
- **Note**: depending on whether or not we directly have "ret" or first a deconstruction of the stack frame like "popq %rbp" we would need to account for this

Systems@ETH zürich

Lecture Recap: Code Vulnerabilities

Preventing Buffer Overflow Attacks

System Level Protection

Systems@ETH zürich

System-level protections

- Compiler-inserted checks on functions
  - Compiler now understands library calls…
- Randomized stack offsets
  - At start of program, allocate random amount of space on stack
  - Makes it difficult to predict beginning of inserted code
- Nonexecutable code segments
  - In older x86, can mark region of memory as either “read-only” or “writeable”
    - Can execute anything readable
  - Add explicit “execute” permission to hardware

Attack Demo

Code Injection and ROP

Exam Questions

Systems@ETH zürich

Exam Questions

Linking: HS19 Question 10

Why linker symbols again

Systems@ETH zürich

- **Generally not hard**: but come often in exam (easy to ask on moodle exam too) and there are some details you need to know about

- A lot of other topics in the exam cover **contents** we have not yet visited in the lecture (Virtual Memory etc.)

Linker Symbols

For each listed identifier in the following C object file, say whether it is a **strong linker symbol**, a **weak linker symbol**, a **macro**, a symbol **local** to the compilation unit, or **none** of these.

```
#define string "hello, world"
static int count;
extern char *otherstring;
struct element {
    void           *data;
    struct element *next;
};
struct element *head = NULL;
struct element *tail;

void push( struct element *e )
{
    e->next = head;
    head    = e;
}
struct element *pull();
```

count:
element:
head:
otherstring:
pull:
push:
string:
tail:

Linker Symbol: Solution

For each listed identifier in the following C object file, say whether it is a **strong linker symbol**, a **weak linker symbol**, a **macro**, a symbol **local** to the compilation unit, or **none** of these.

```
#define string "hello, world"
static int count;
extern char *otherstring;
struct element {
    void            *data;
    struct element *next;
};
struct element *head = NULL;
struct element *tail;

void push( struct element *e )
{
    e->next = head;
    head    = e;
}
struct element *pull();
```

count: local

element: none

head: strong

otherstring: none

pull: none

push: strong

string: macro

tail: weak

Remarks

- **Otherstring none**? Because theoretically its an “external symbol”, but we **never actually use it in the code to reference something** so we do not care about it: the linker will not emit a external symbol for it
- **Pull none**: same explanation above (we do not use it)

count: local

element: none

head: strong

otherstring: none

pull: none

push: strong

string: macro

tail: weak

Systems@ETH zürich

Exam Questions

Struct Size, Preprocessor: HS19 Question 9

HS19 Question 9: Struct size

Systems@ETH zürich

Suppose the file `buffer.h` contains **only** the following declaration:

```
struct buffer {
    void    *data;
    size_t   length;
    char     flags[4];
};
```

On a 64-bit x86 Linux computer, what is `sizeof(struct buffer)`?

(2 points)

HS19 Question 9: Struct size

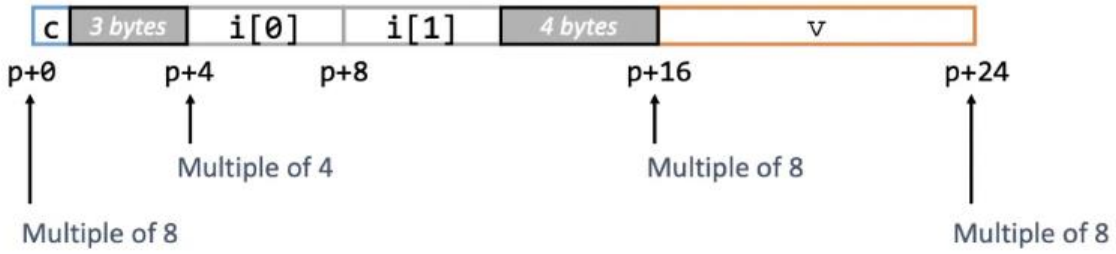
Satisfying alignment with structures

- Within structure:
  - Must satisfy element's alignment requirement
- Overall structure placement
  - Each structure has alignment requirement K
    - K = Largest alignment of any element
  - Initial address & structure length must be multiples of K

```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```

- K = 8, due to double element

```
} *p;
```



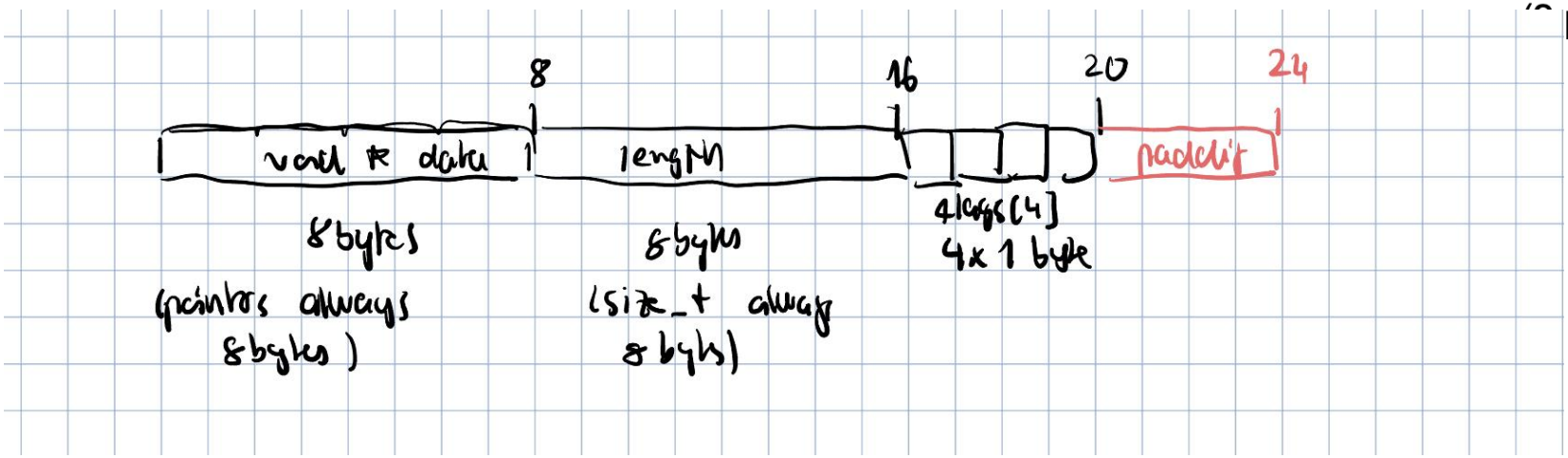
HS19 Question 9: Struct size

Suppose the file `buffer.h` contains **only** the following declaration:

```
struct buffer {
    void    *data;
    size_t   length;
    char     flags[4];
};
```

On a 64-bit x86 Linux computer, what is `sizeof(struct buffer)`?

(points)



- **Recall**: each element needs to be aligned on its own boundary, the whole struct on the size of the biggest element

HS19 Question 9: Struct size

Systems@ETH zürich

Now suppose that two other header files, `llist.h` and `hashtable.h`, both use the declaration of `struct buffer` and so include the line `#include "buffer.h"`. All is well until the programmer tries to compile the following file `.c` file:

```
#include "buffer.h"
#include "llist.h"
#include "hashtable.h"

int main(int argc, char *argv[])
{
    return 0;
}
```

How does the compilation fail, and why?

(2 points)

Show a change to **only** the file `buffer.h` which will fix this problem.

(4 points)

HS19 Question 9: Struct size

Systems@ETH zürich

- **Recall**: C Pre Processor (CPP) **copies and pasts** all the elements of the .h file into the compilation unit: so now we have 3x definition of the buffer (**redefinition not allowed**)

- **Solution**: CPP conditionals

```
#ifndef BUFFER_H

#define BUFFER_H

// Code...

#endif
```

Exam Questions

Multiple Choice: HS23 Question 1

HS23 Question 1: General Theory

Systems@ETH zürich

○	○	A pointer is a variable that stores a memory address.
○	○	Global variables are stored on the heap.
○	○	Variables stored on the heap have higher memory addresses than variables stored on the stack.
○	○	As a C programmer, you should use assert statements to notify users about any unexpected errors in the inputs they provide to your program.
○	○	The gcc compiler ensures that procedure arguments and local variables are always stored in the procedure's stack frame.
○	○	If the value of a int pointer **p** on the heap is 0x1000, then (**p**+1) is 0x1004. Assume sizeof(int)=4.

HS23 Question 1: General Theory

A pointer is a variable that stores a memory address.	✔	
Global variables are stored on the heap.	✔	Global variables are stored in the data segment, which is separate from the heap.
Variables stored on the heap have higher memory addresses than variables stored on the stack.	✔	The stack is at higher memory addresses in the virtual address space than the heap.
As a C programmer, you should use assert statements to notify users about any unexpected errors in the inputs they provide to your program.	✔	Assertions are not intended for users. They are for catching real bugs in the program.
The gcc compiler ensures that procedure arguments and local variables are always stored in the procedure's stack frame.	✔	False: if there are enough registers, arguments and local variables can be stored in registers and do not need to be stored on the stack.
If the value of a int pointer **p** on the heap is 0x1000, then (**p**+1) is 0x1004. Assume sizeof(int)=4.	✔	Pointer arithmetic depends on the pointer type. Integers are 4 bytes so adding 1 to an int pointer increments the address by 4.

HS23 Question 2: General Theory

Systems@ETH zürich

Consider the C declaration:
`int arr[10] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};`

Suppose that the compiler has placed the variable arr in the %ecx register.

How do you move the value at arr[2] into the %eax register? Assume that %ebx is 2 and sizeof(int) = 4.

You can view the x86-64 reference sheet by clicking here.

- ○ movl (%ecx,%ebx,4),%eax
- ○ movl 4(%ecx,%ebx,1),%eax
- ○ leal 8(%ecx),%eax
- ○ leal (%ecx,%ebx,4),%eax
- ○ leal 4(%ecx,%ebx,2),%eax

HS23 Question 2: General Theory

Systems@ETH zürich

Consider the C declaration:
`int arr[10] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};`

Suppose that the compiler has placed the variable arr in the %ecx register.

How do you move the value at arr[2] into the %eax register? Assume that %ebx is 2 and sizeof(int) = 4.

You can view the x86-64 reference sheet by clicking here.

✔ movl (%ecx,%ebx,4),%eax	S	✔
✖ movl 4(%ecx,%ebx,1),%eax	S	
✖ leal 8(%ecx),%eax	S	
✖ leal (%ecx,%ebx,4),%eax	S	
✖ leal 4(%ecx,%ebx,2),%eax	S	

HS23 Question 3: Linking

Systems@ETH zürich

Consider the executable *a.out*, compiled and linked as follows:

> gcc -fno-common -o a.out main.c init.c

main.c contains:

```c
#include <stdio.h>

int a = 1;
static int b = 2;
int c = 3;

int main() {
   int c = 4;
   init();
   printf("a=%d, b=%d, c=%d\n", a, b, c);
   return 0;
}
```

init.c contains:

```c
extern int a, c;
int b;

void init() {
   a = 5;
   b = 6;
   c = 7;
}
```

What values of *a*, *b*, an *c* does *a.out* print?

Consider the executable *a.out*, compiled and linked as follows:

> gcc -fno-common -o a.out main.c init.c

main.c contains:

```c
#include <stdio.h>

int a = 1;
static int b = 2;
int c = 3;

int main() {
   int c = 4;
   init();
   printf("a=%d, b=%d, c=%d\n", a, b, c);
   return 0;
}
```

init.c contains:

```c
extern int a, c;
int b;

void init() {
   a = 5;
   b = 6;
   c = 7;
}
```

What values of *a, b*, an *c* does *a.out* print?

a: 5 ✔

b: 2 ✔

c: 4 ✔

- **b**: local in main, so 2 anyway
- **C**: local variable (c=4) shadows global variable
- **A**: variable a in init refers to the same as main: changes the value 5

Remark

- **This** was already half of the theory part in the moodle exam
- **The rest was coding**: so do the labs and the Code Experts for your own good

Systems@ETH zürich

Any questions about anything in the course so far?

Otherwise I will recap what we looked at last exercise session for those who have not been here

Systems@ETH zürich

UNIX FHS – Recap From Last Lecture

Filesystem, Paths, Binaries etc.

Remark

- **This is** the same we discussed last exercise session, as many people weren't there
- In case you have been here last exercise session or are already proficient with the terminal this is not relevant for you

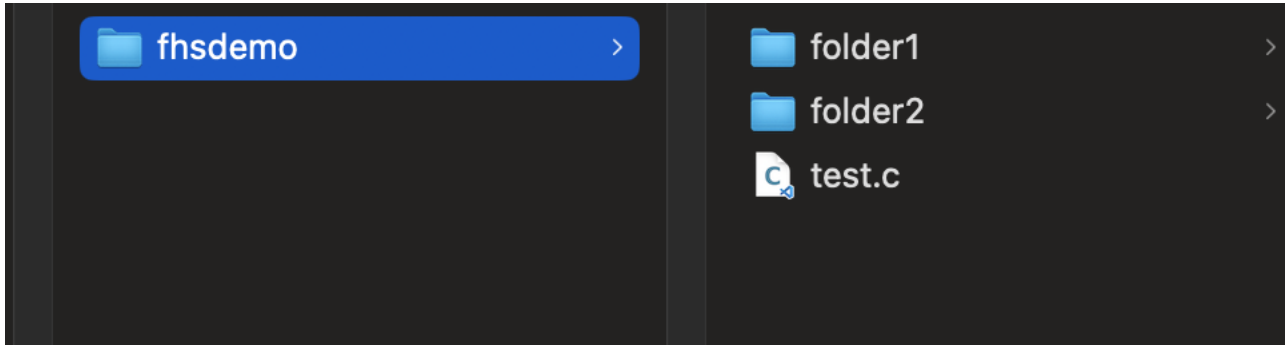
Systems@ETH zürich

Why look UNIX FHS again?

- **Fundamental** to understand, not really taught in course though
- **Will** help you a lot later on, although not super relevant for the exam its **super relevant for other courses, understanding and praxis**

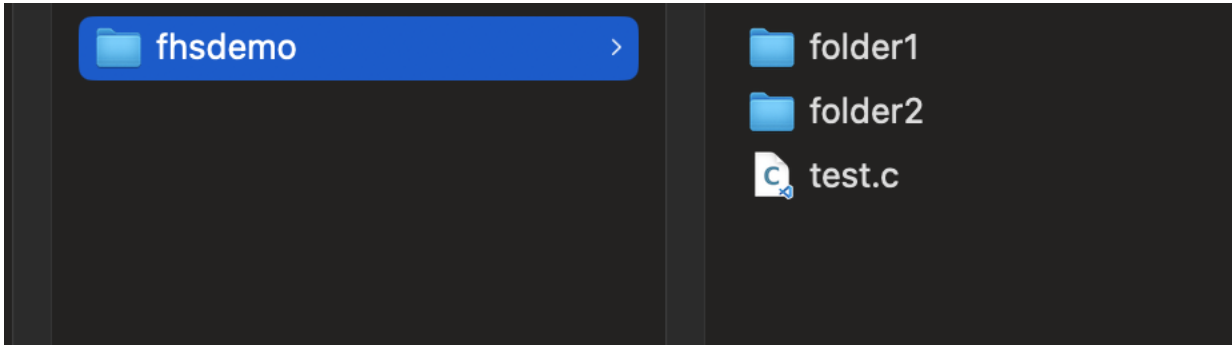
UNIX FHS

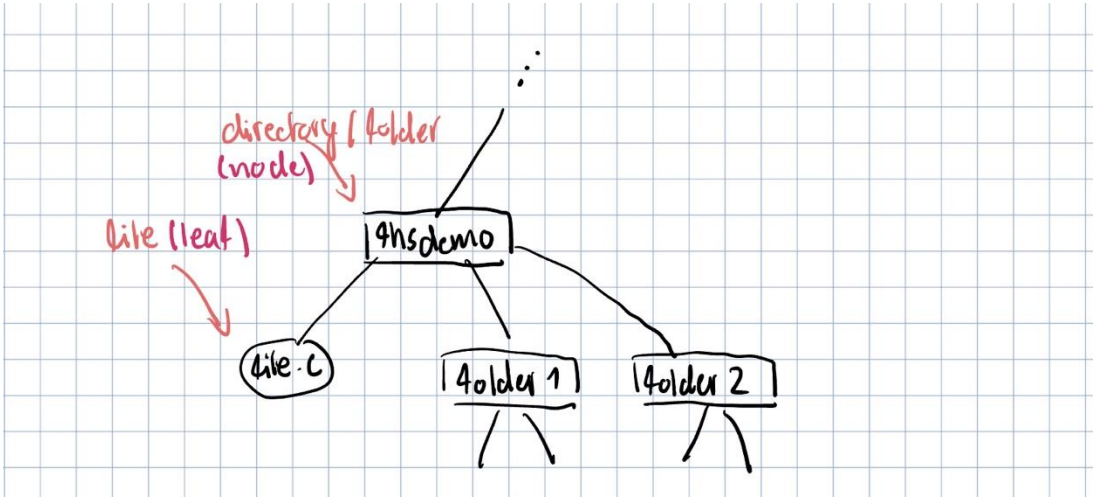
- Looked at one/multiple C files, mostly in one directory (Folder)



UNIX FHS

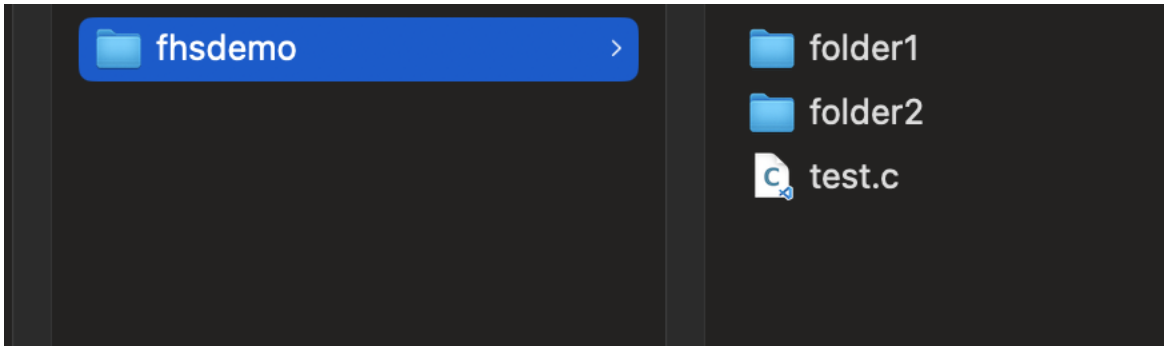
- Linux FHS is a Graph: you are just at a particular node

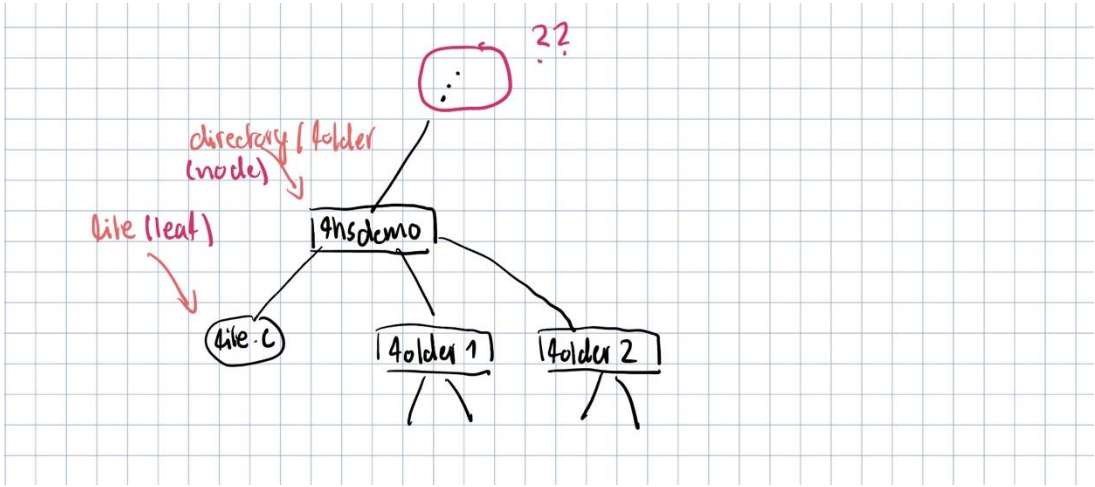




UNIX FHS

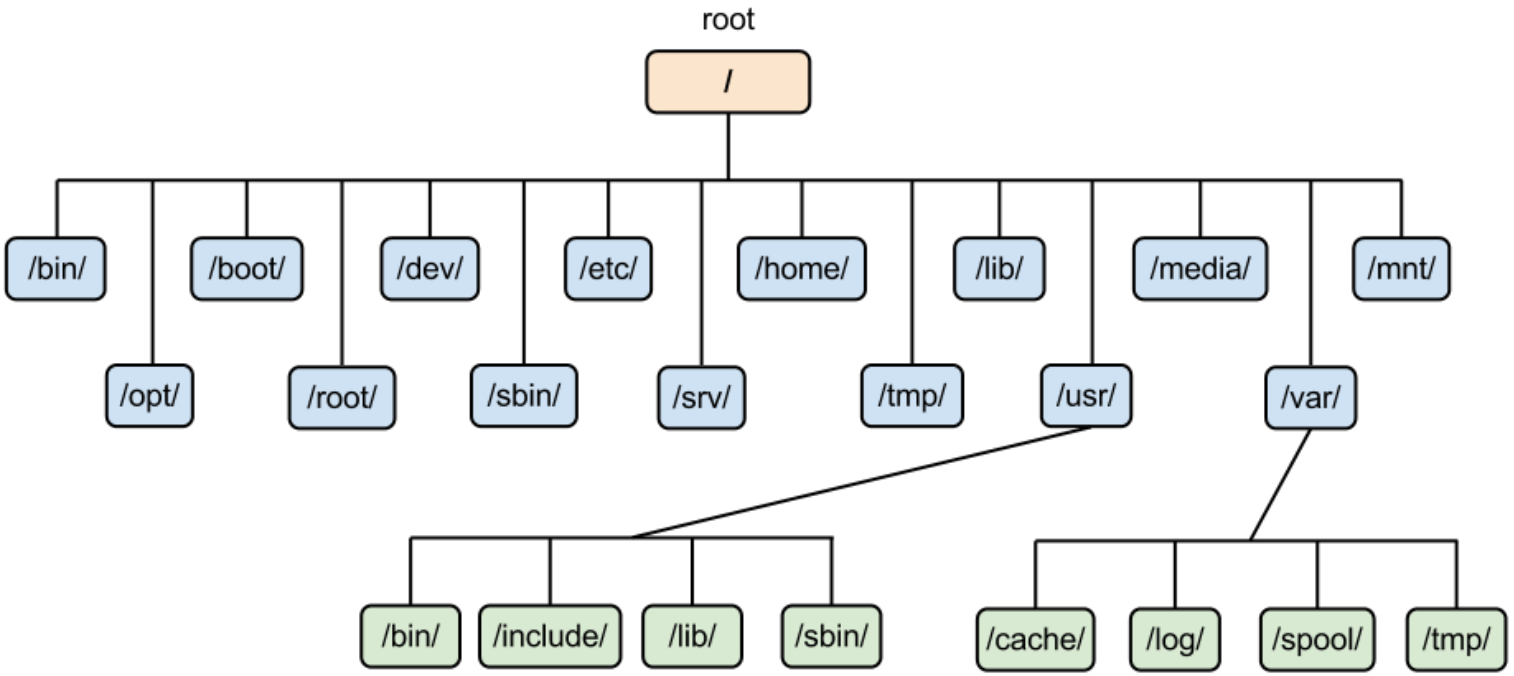
- Linux FHS is a Graph: you are just at a particular node
- What is at "..."?





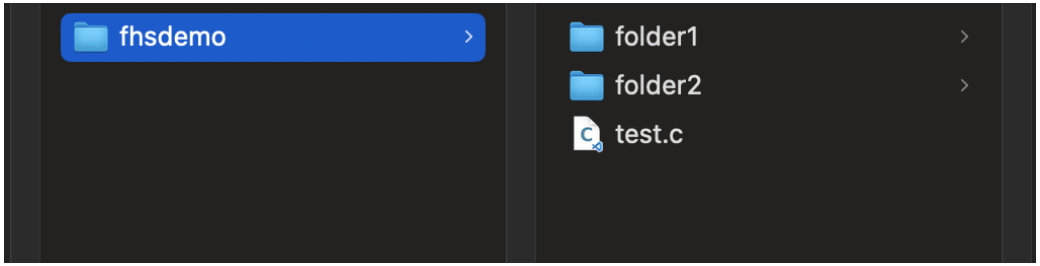
Systems@ETH Zürich

UNIX FHS

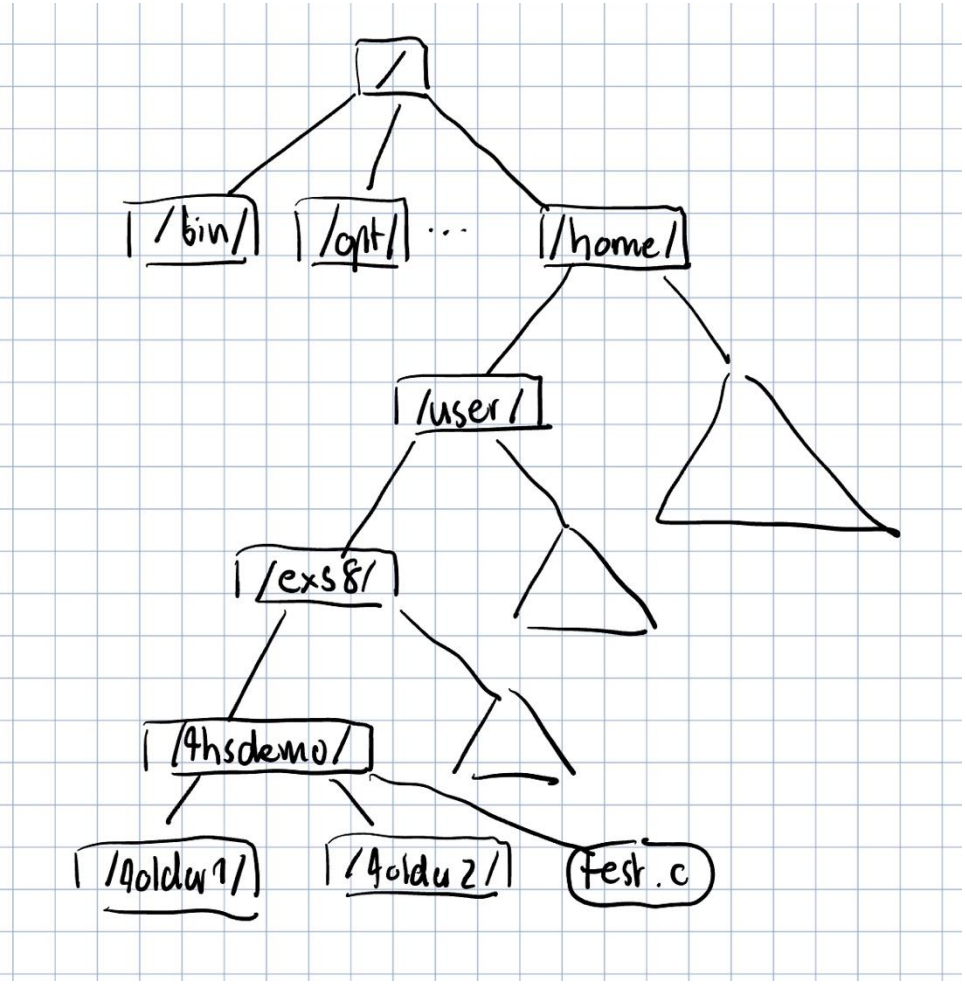


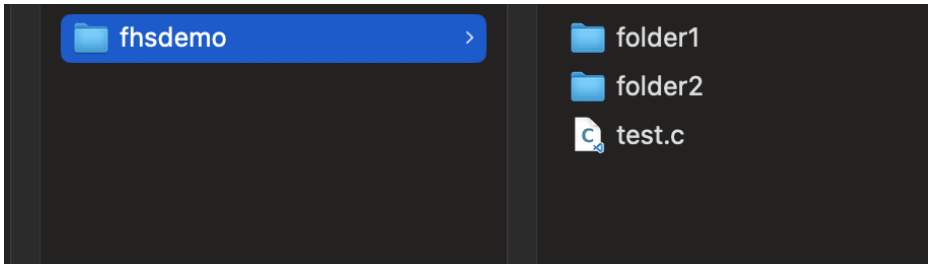
UNIX FHS

- Terminal?
- "**pwd**" shows where you are in the graph




```
user@4865b4f533e3:~/exs8/fhsdemo$ pwd
/home/user/exs8/fhsdemo
```

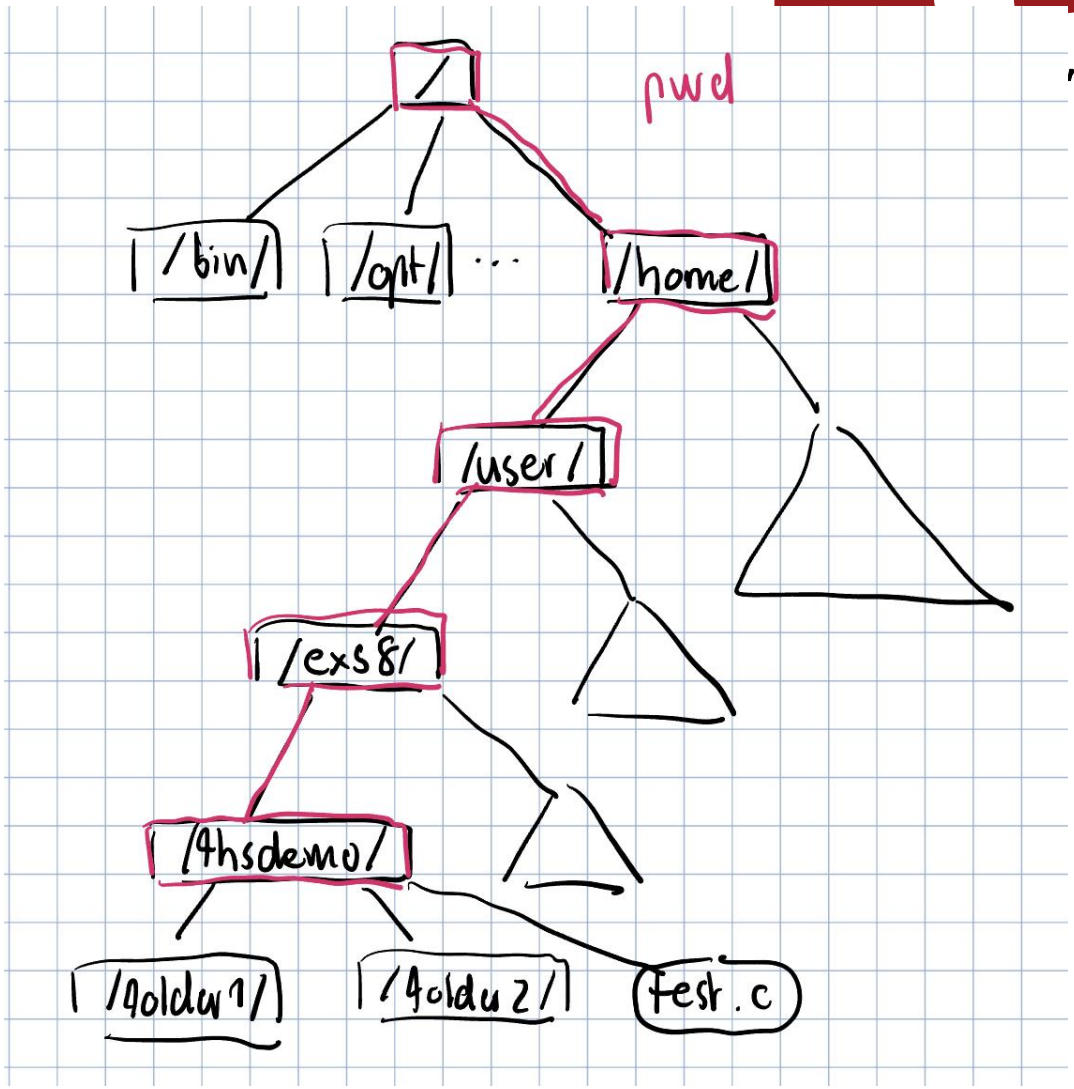





```
user@4865b4f533e3:~/exs8/fhsdemo$ pwd
/home/user/exs8/fhsdemo
```

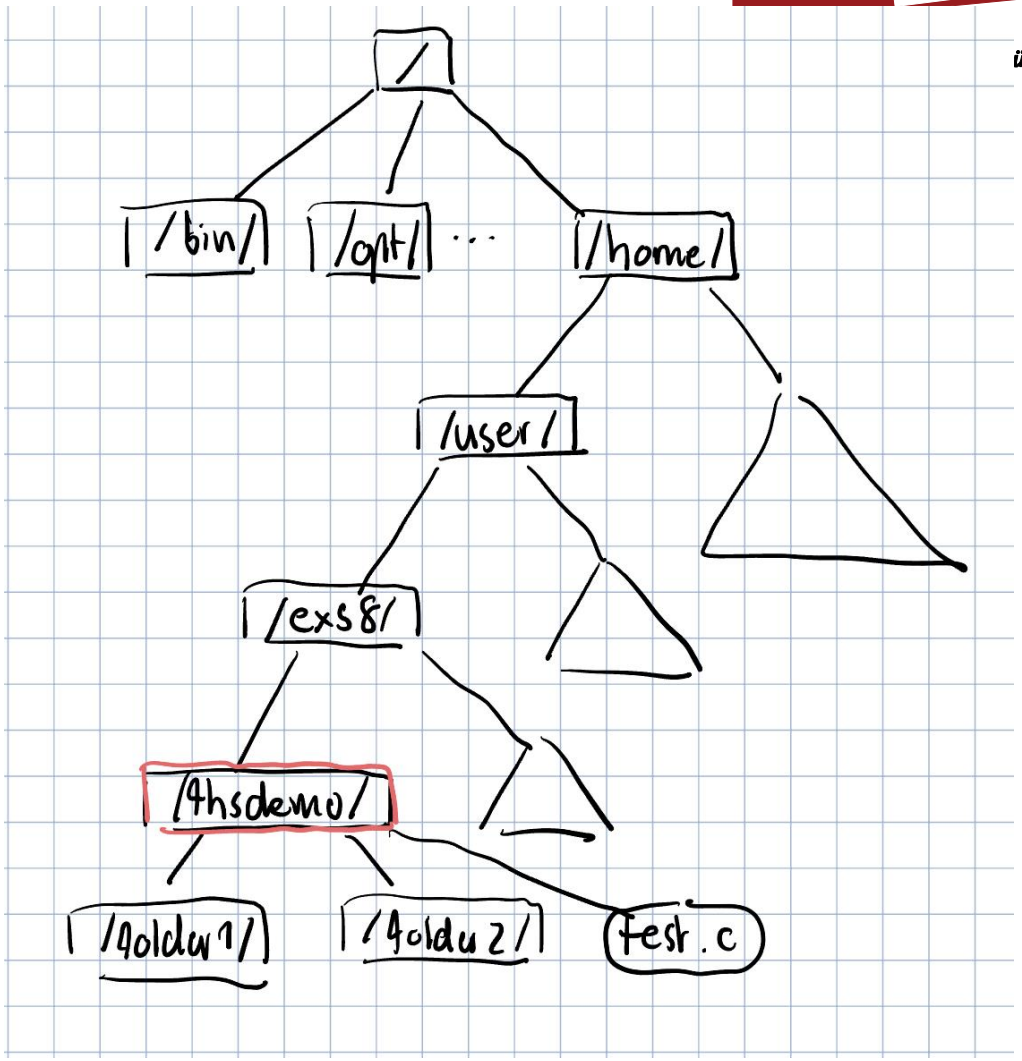
UNIX FHS

- **Pwd** shows you current path, by concatenating names of nodes to where you are



UNIX FHS

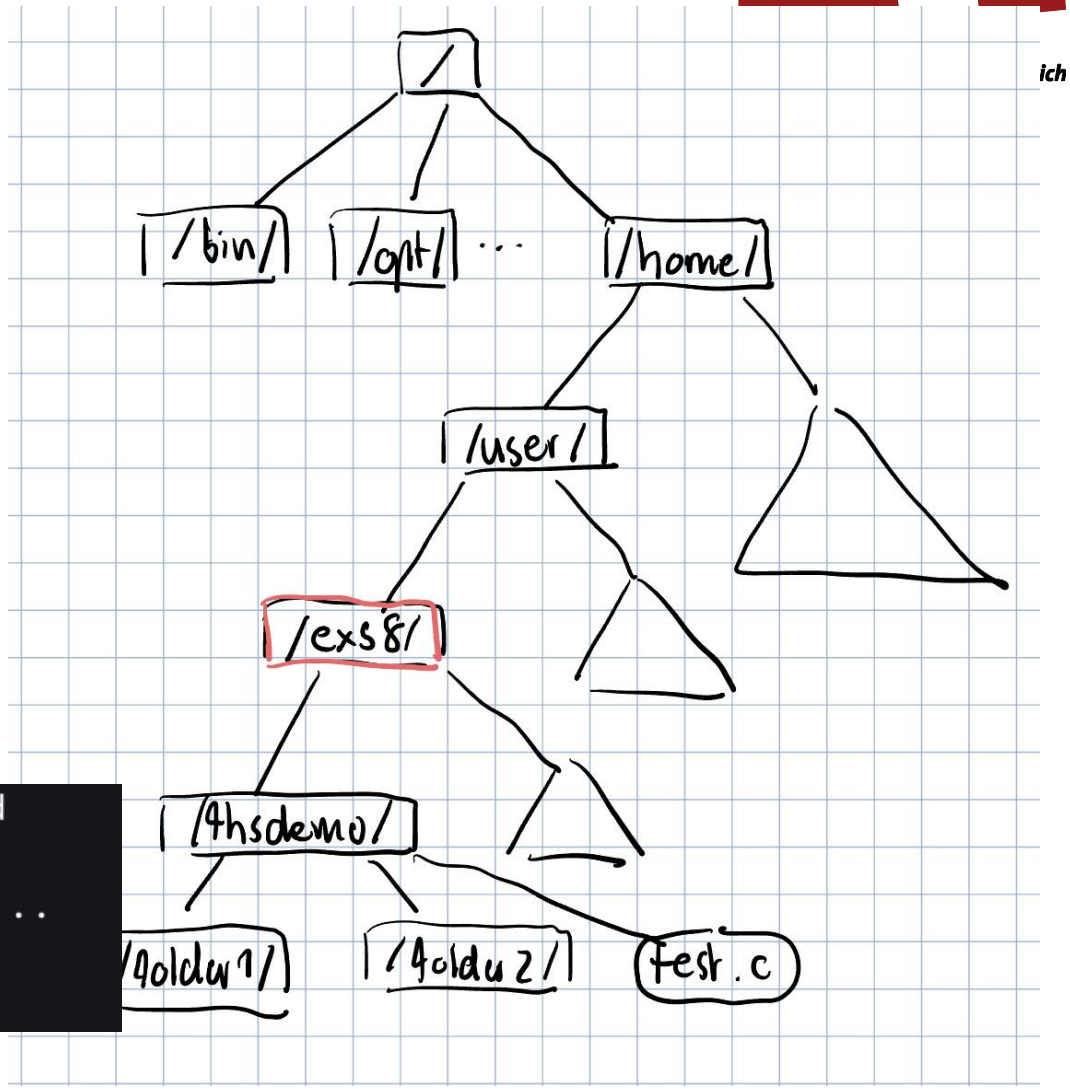
- **You** are always somewhere in this graph, the current **working directory**
- Refer to your own position as “.”



UNIX FHS

- You can move relative to your current position
- **Up**: “cd ..”
- Now check pos with “pwd”

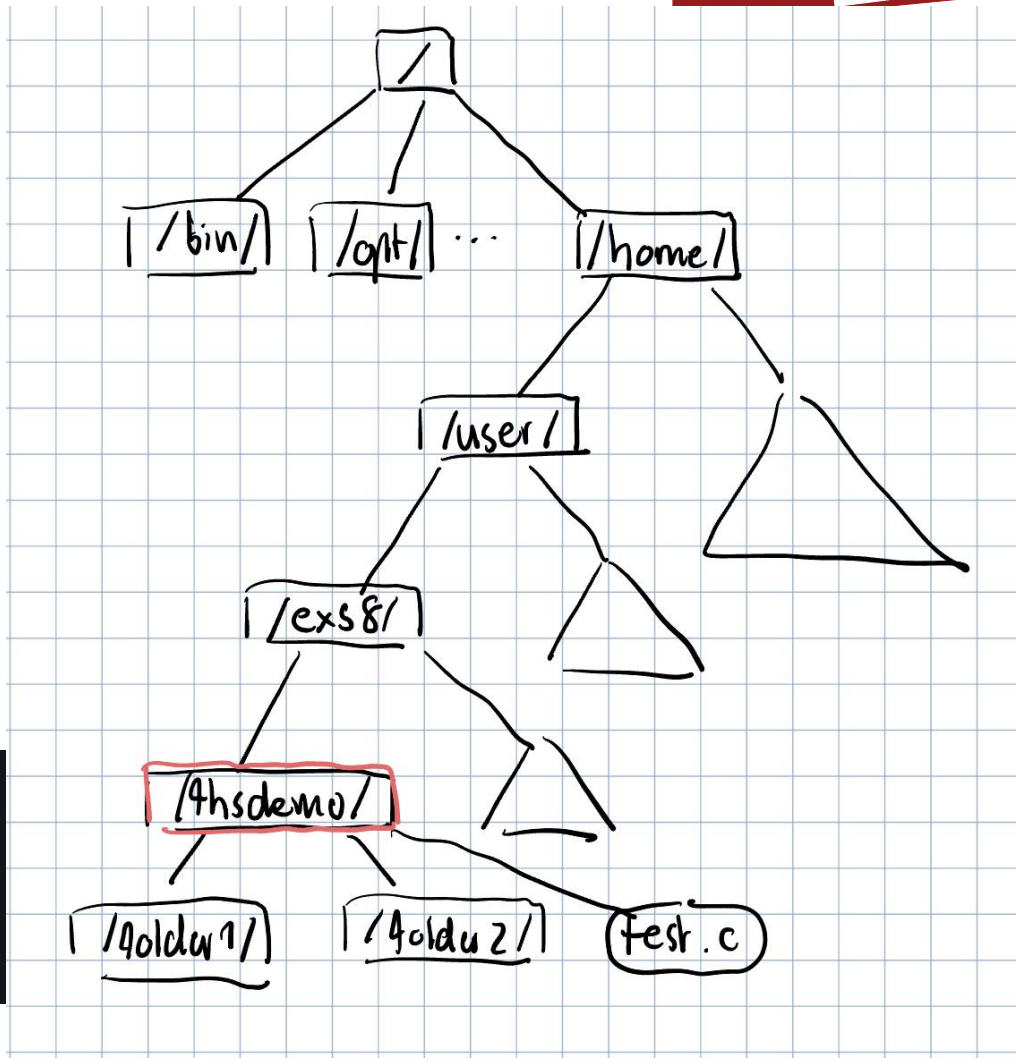
```
user@4865b4f533e3:~/exs8/fhsdemo$ pwd
/home/user/exs8/fhsdemo
user@4865b4f533e3:~/exs8/fhsdemo$ cd ..
user@4865b4f533e3:~/exs8$ pwd
/home/user/exs8
```



UNIX FHS

- You can move relative to your current position
- **Up**: “cd ..”
- **Down**: “cd fhsdemo/”
- Now check pos with “pwd”

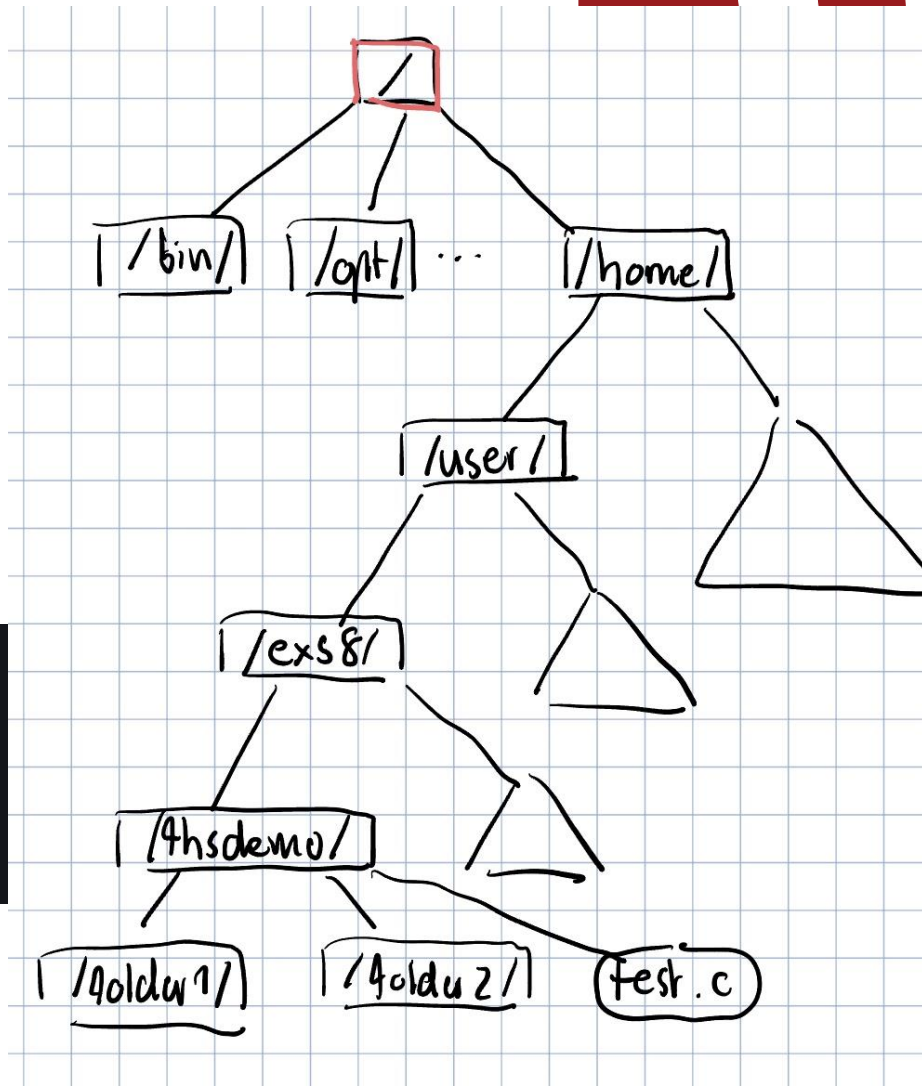
```
user@4865b4f533e3:~/exs8$ pwd
/home/user/exs8
user@4865b4f533e3:~/exs8$ cd fhsdemo/
user@4865b4f533e3:~/exs8/fhsdemo$ pwd
/home/user/exs8/fhsdemo
```



UNIX FHS

- You can move to a **absolute position**
- **“cd <absolutepath>”**
- Example: “cd /"

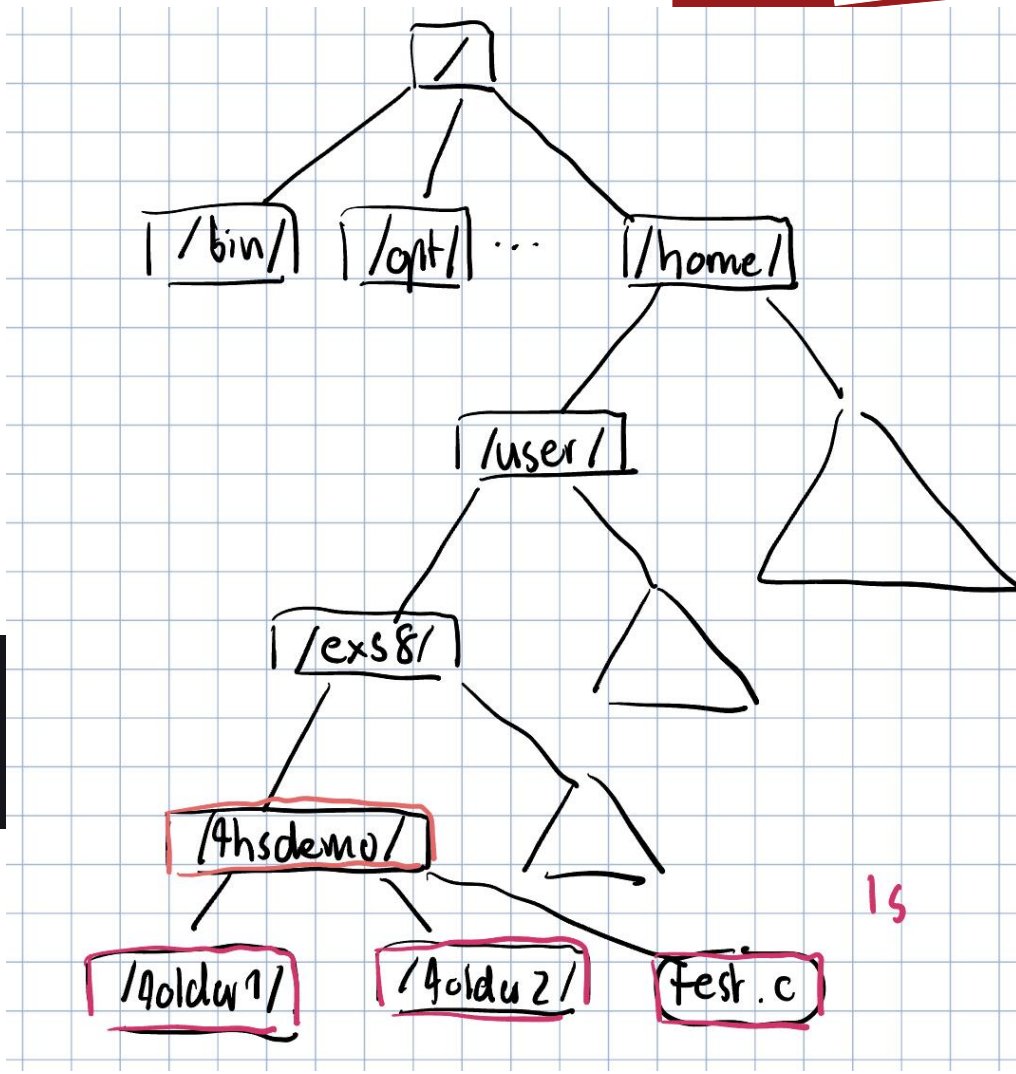
```
user@4865b4f533e3:~/exs8/fhsdemo$ pwd
/home/user/exs8/fhsdemo
user@4865b4f533e3:~/exs8/fhsdemo$ cd /
user@4865b4f533e3:/$ pwd
/
```



UNIX FHS

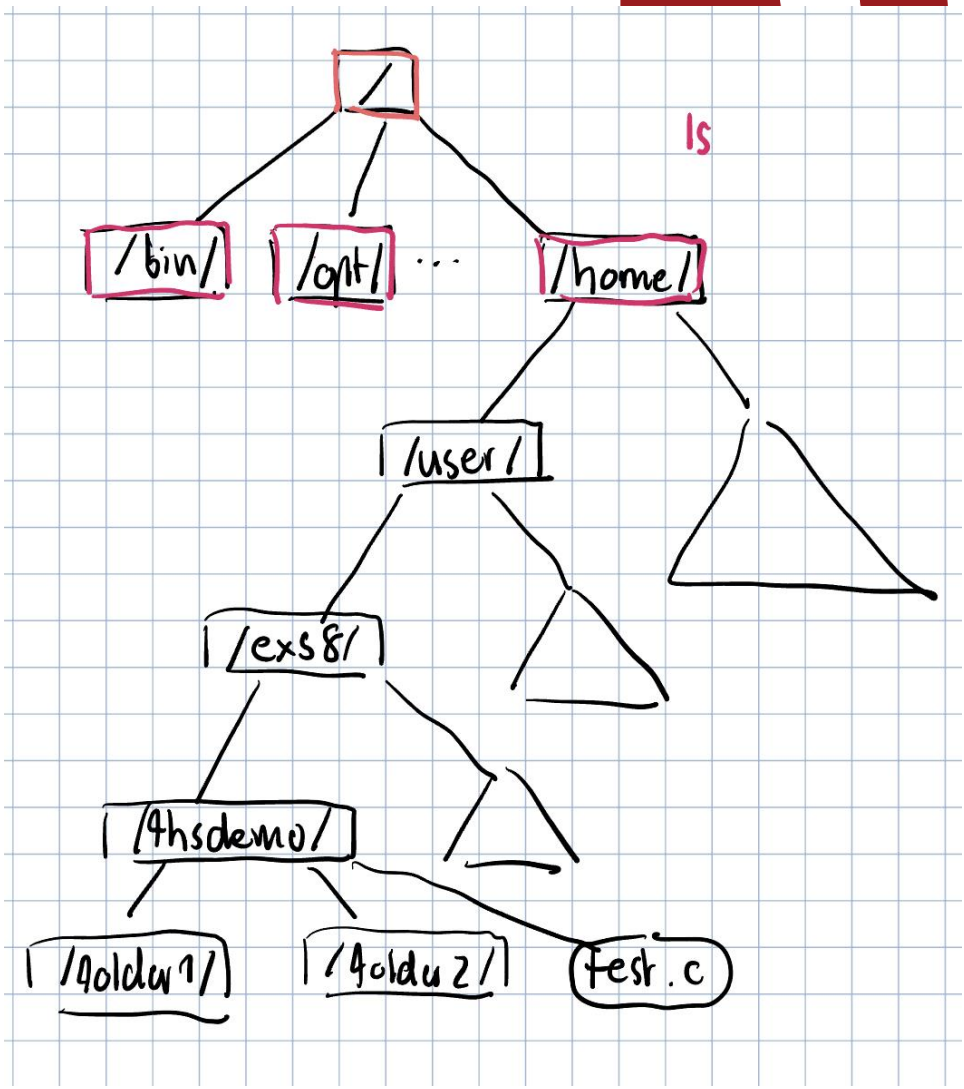
- Check whats inside your current directory (node)?
- “ls” for list

```
user@4865b4f533e3:~/exs8/fhsdemo$ pwd
/home/user/exs8/fhsdemo
user@4865b4f533e3:~/exs8/fhsdemo$ ls
folder1  folder2  test.c
```

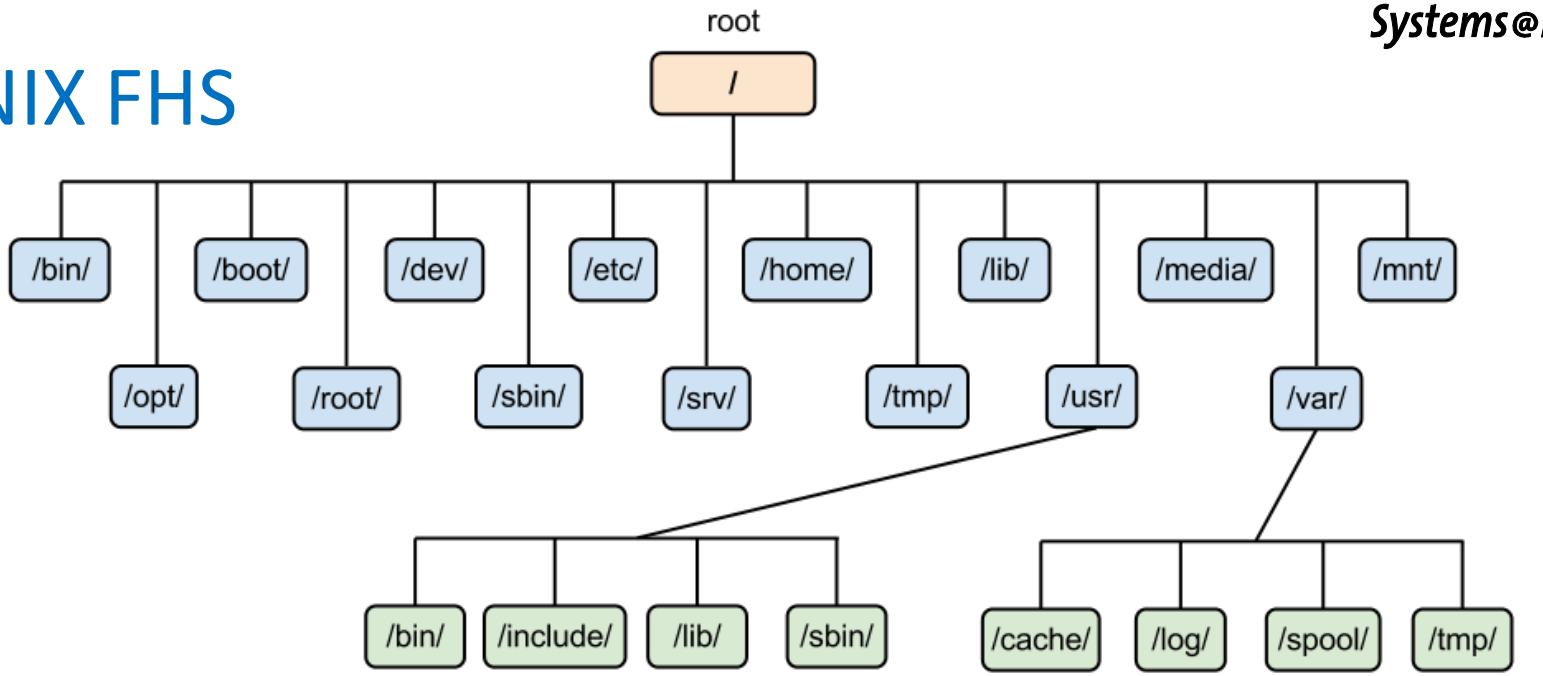


UNIX FHS

- Lets check whats inside "/"?



UNIX FHS



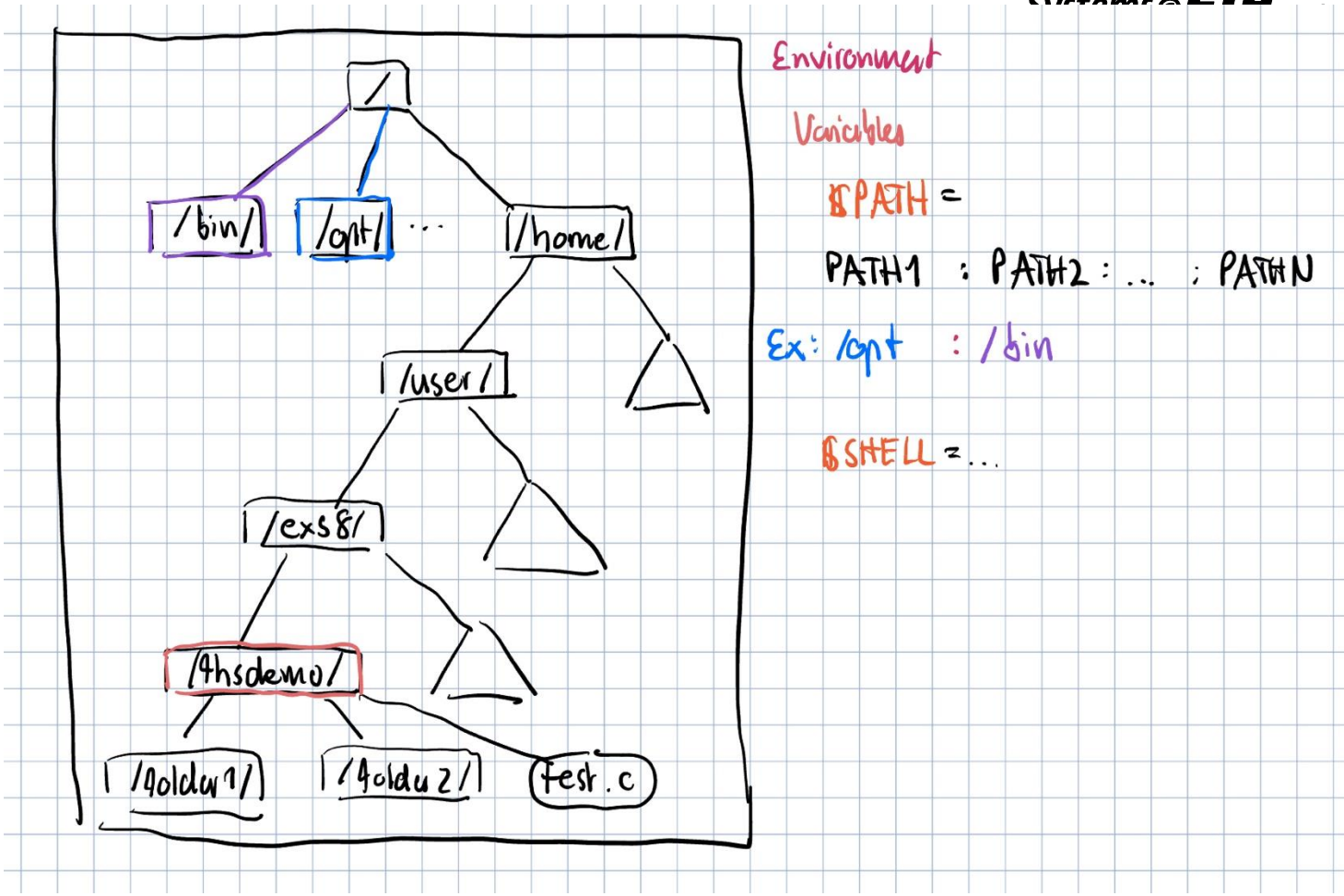

```
user@4865b4f533e3:/$ pwd
/
user@4865b4f533e3:/$ ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
```

UNIX FHS

Binaries in the FHS

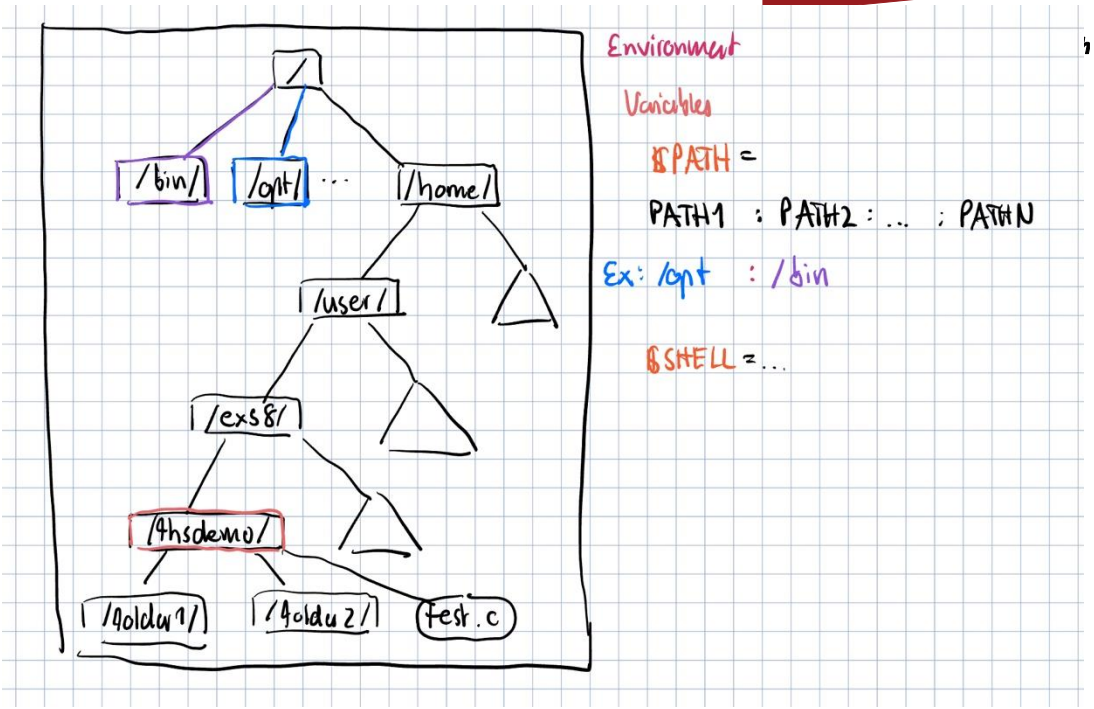
UNIX FHS

- We have an **environment** : think of having variables next to the graph
- Can store **paths** for instance



UNIX FHS

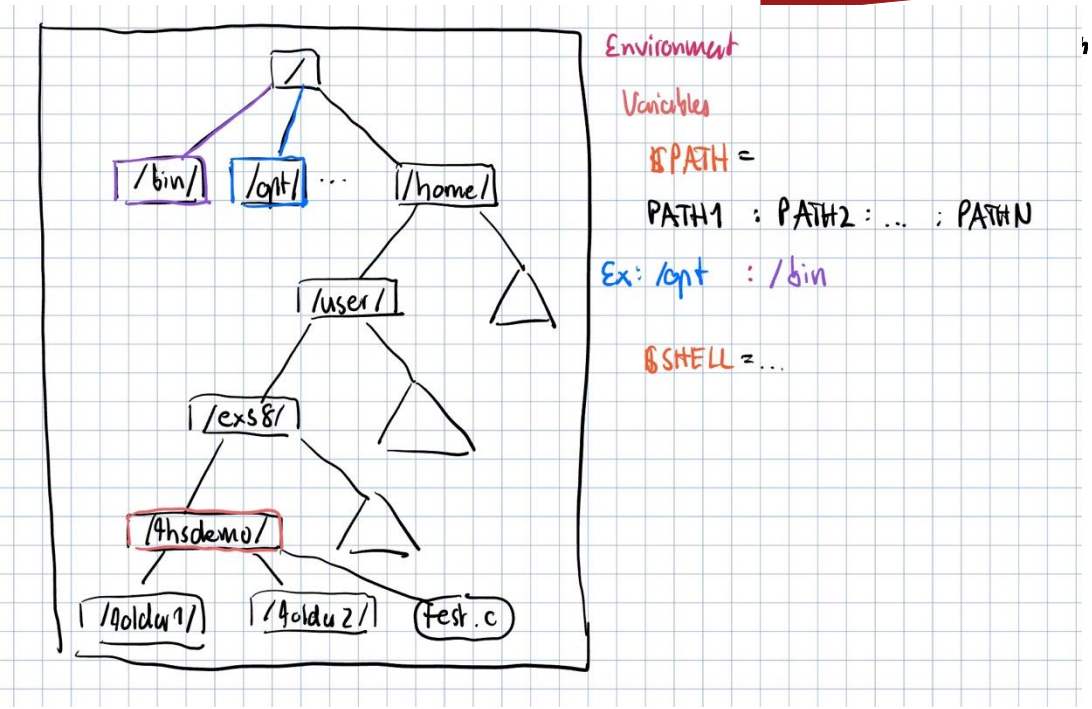
- **$PATH**: Stores paths
  PATH1 : PATH2 : … : PATHN
- Each Path separated by “:” in between
- Path1= /opt
- Path2= /bin
- $PATH=/opt:/bin




```
user@4865b4f533e3:/$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

UNIX FHS

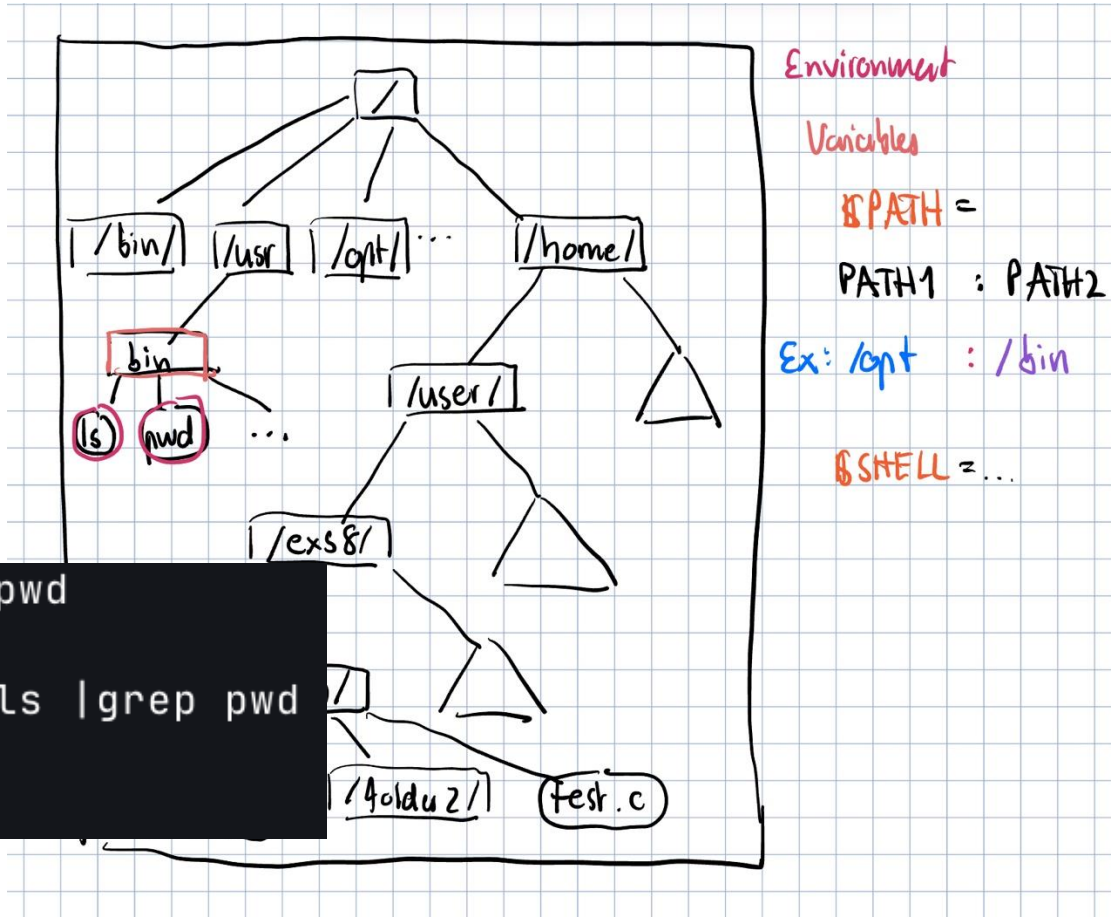
- Every “ls” or ”pwd” etc. executes a **binary** (program)
- It finds it by looking in **all directories** specified by **path variable $PATH**
- “which <binary>” tells us where it is in the graph




```
user@4865b4f533e3:/$ which ls
/usr/bin/ls
user@4865b4f533e3:/$ which pwd
/usr/bin/pwd
```

UNIX FHS

- Check out the path, and check if its actually here? **IT IS**




```
user@4865b4f533e3:/usr/bin$ pwd
/usr/bin
user@4865b4f533e3:/usr/bin$ ls |grep pwd
pwd
pwdx
```

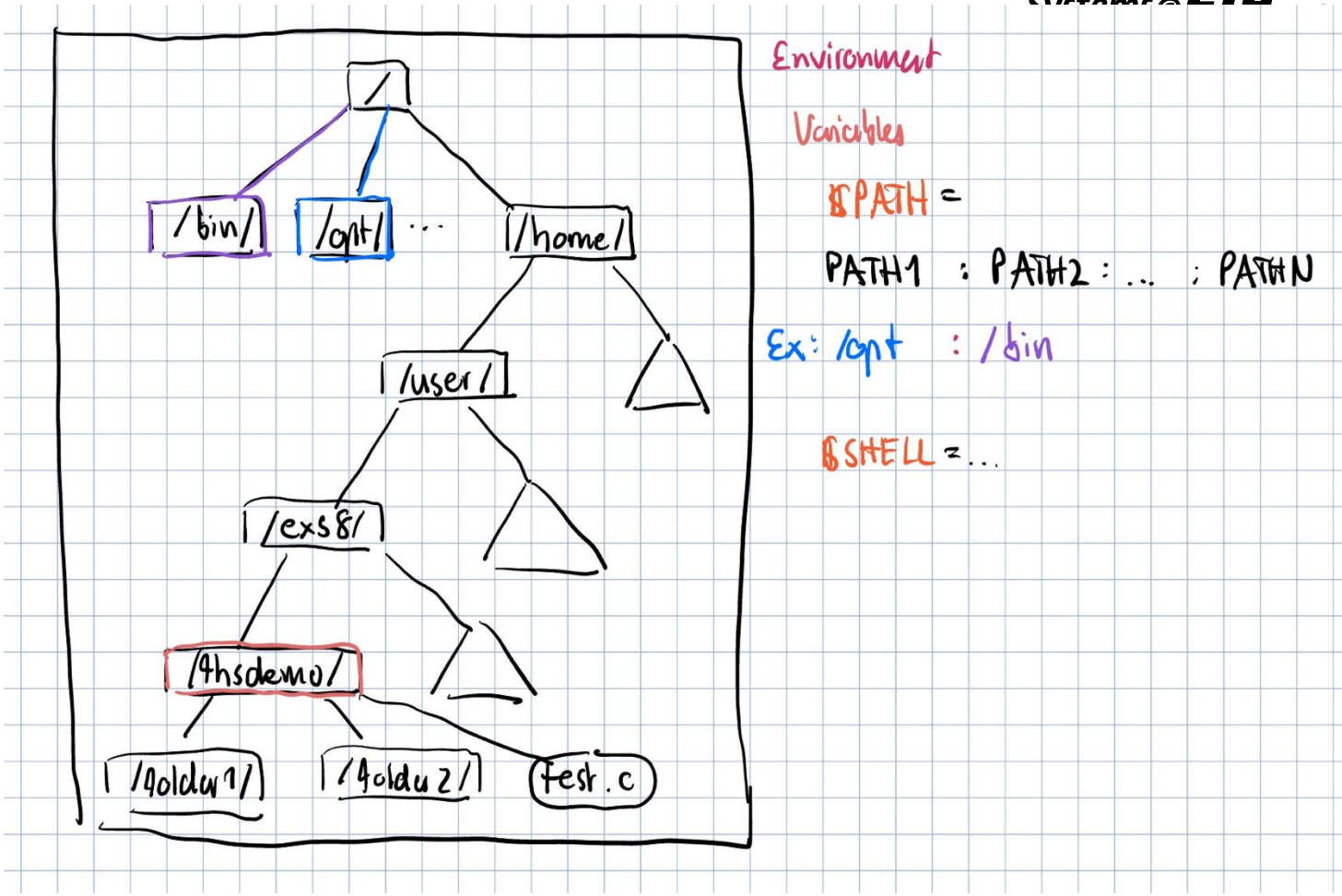
UNIX FHS

- We can do this for **any binary** specified in the **$PATH**
- Lets check out **gcc**

```
user@4865b4f533e3:/usr/bin$ which gcc
/usr/bin/gcc
user@4865b4f533e3:/usr/bin$ cd /usr/bin/
user@4865b4f533e3:/usr/bin$ pwd
/usr/bin
user@4865b4f533e3:/usr/bin$ ls |grep gcc
c89-gcc
c99-gcc
gcc
gcc-11
gcc-ar
gcc-ar-11
gcc-nm
gcc-nm-11
gcc-ranlib
gcc-ranlib-11
x86_64-linux-gnu-gcc
x86_64-linux-gnu-gcc-11
x86_64-linux-gnu-gcc-ar
x86_64-linux-gnu-gcc-ar-11
x86_64-linux-gnu-gcc-nm
x86_64-linux-gnu-gcc-nm-11
x86_64-linux-gnu-gcc-ranlib
x86_64-linux-gnu-gcc-ranlib-11
user@4865b4f533e3:/usr/bin$
```

UNIX FHS

- **Note**: we can be **anywhere** in the FHS, and execute those commands as **$PATH** specifies where to look for it
- **Do not have to be in the directory /bin to execute it**



UNIX FHS

- Write own C program printing “hello, world”
- **Compile it**

```
[bfalk@piora1 ~]$ pwd
/home/bfalk
[bfalk@piora1 ~]$ cat hello.c
#include <stdio.h>

int main(int argc, char** argv){
        printf("hello, world\n");
        return 0;
}
[bfalk@piora1 ~]$ gcc -o hello hello.c
[bfalk@piora1 ~]$ ls
hello  hello.c  simpleadd  simpleadd.c
[bfalk@piora1 ~]$
```

UNIX FHS

- Write own C program (you know how to!)
- **Compile it**
- Execute via “./hello”

- Difference between “./hello” and “/home/bfalk/hello”?

```
[bfalk@piora1 ~]$ ./hello
hello, world
[bfalk@piora1 ~]$ pwd
/home/bfalk
[bfalk@piora1 ~]$ /home/bfalk/hello
hello, world
[bfalk@piora1 ~]$
```

UNIX FHS

- Write own C program (you know how to!)
- **Compile it**
- Execute via “./hello”

- Difference between “./hello” and “/home/bfalk/hello”?

```
[bfalk@piora1 ~]$ ./hello
hello, world
[bfalk@piora1 ~]$ pwd
/home/bfalk
[bfalk@piora1 ~]$ /home/bfalk/hello
hello, world
[bfalk@piora1 ~]$
```

UNIX FHS

- Why doesn't "hello" work? "pwd" "ls" etc. worked without the "./"?

```
[bfalk@piora1 ~]$ hello
-bash: hello: command not found
[bfalk@piora1 ~]$
```

Systems@ETH zürich

UNIX FHS

- Why doesn't "hello" work? "pwd" "ls" etc. worked without the "./"?

- **Issue**: for non absolute path it checks $PATH variable, there is no "hello" executable anywhere

```
[bfalk@piora1 ~]$ hello
-bash: hello: command not found
[bfalk@piora1 ~]$ which hello
/usr/bin/which: no hello in (/cm/local/apps/gcc/13.1.0/bin:/home/bfalk/.local/bin:/home/bfalk/bin:/
cm/local/apps/environment-modules/4.5.3//bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbi
n:/usr/sbin:/cm/local/apps/environment-modules/4.5.3/bin)
[bfalk@piora1 ~]$
```

UNIX FHS

- Why doesn't "hello" work? "pwd" "ls" etc. worked without the "./"?
- **Issue**: for non absolute path it checks $PATH variable, there is no "hello" executable anywhere
- **Solution**: add directory which has "hello" executable to path

```
[bfalk@piora1 ~]$ echo $PATH
/cm/local/apps/gcc/13.1.0/bin:/home/bfalk/.local/bin:/home/bfalk/bin:/cm/local/apps/environment-mod
ules/4.5.3//bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/sbin:/cm/local/apps/en
vironment-modules/4.5.3/bin
[bfalk@piora1 ~]$ pwd
/home/bfalk
[bfalk@piora1 ~]$ export PATH=/home/bfalk:$PATH
[bfalk@piora1 ~]$ echo $PATH
/home/bfalk:/cm/local/apps/gcc/13.1.0/bin:/home/bfalk/.local/bin:/home/bfalk/bin:/cm/local/apps/env
ironment-modules/4.5.3//bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/sbin:/cm/l
ocal/apps/environment-modules/4.5.3/bin
[bfalk@piora1 ~]$ hello
hello, world
[bfalk@piora1 ~]$
```

UNIX FHS

- Now we can execute hello from **anywhere** just as “pwd” and “ls” etc.
- It appears in **which**, so we know its accessible form **everywhere**
- **“~”** is short for my home, i.e. /home/bfalk

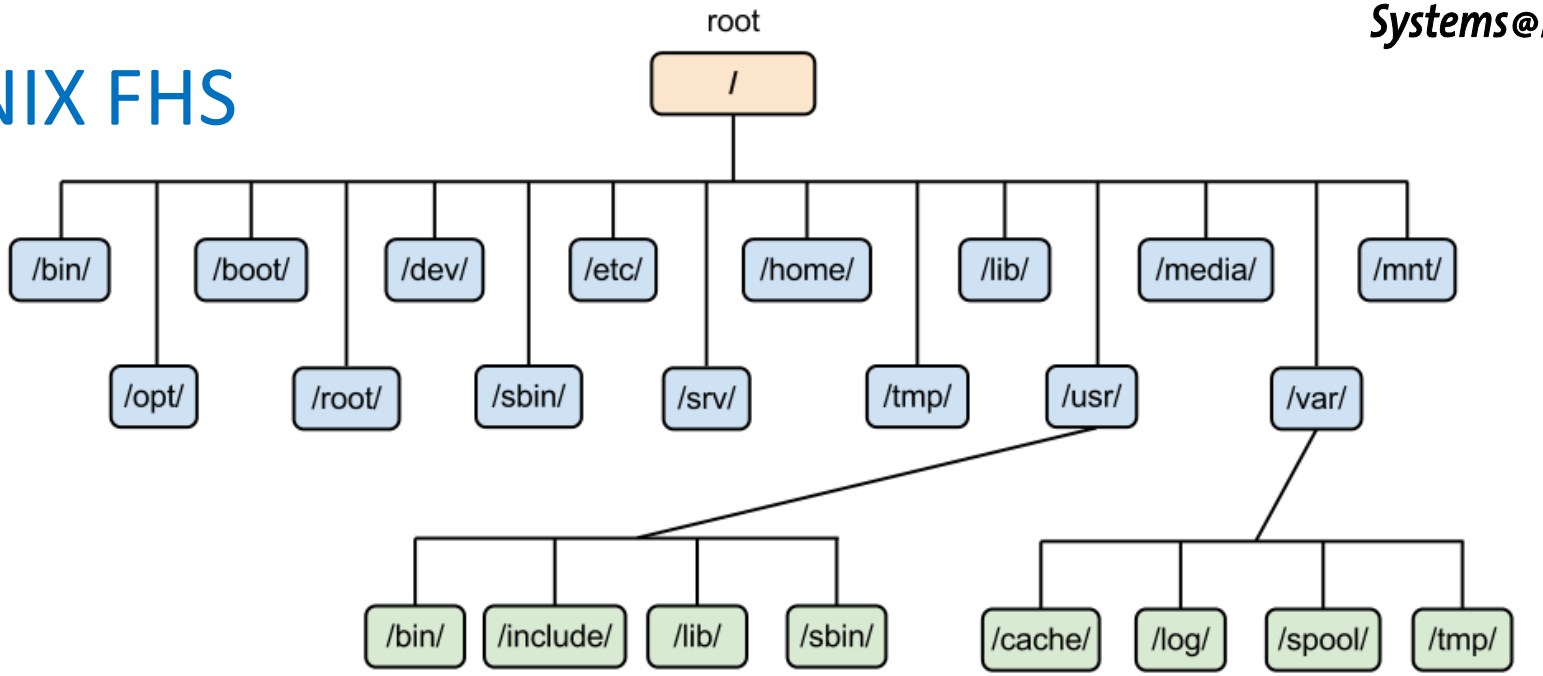
```
[bfalk@piora1 ~]$ cd /
[bfalk@piora1 /]$ pwd
/
[bfalk@piora1 /]$ hello
hello, world
[bfalk@piora1 /]$
```

```
[bfalk@piora1 /]$ which hello
~/hello
[bfalk@piora1 /]$
```

UNIX FHS

Libraries in the FHS

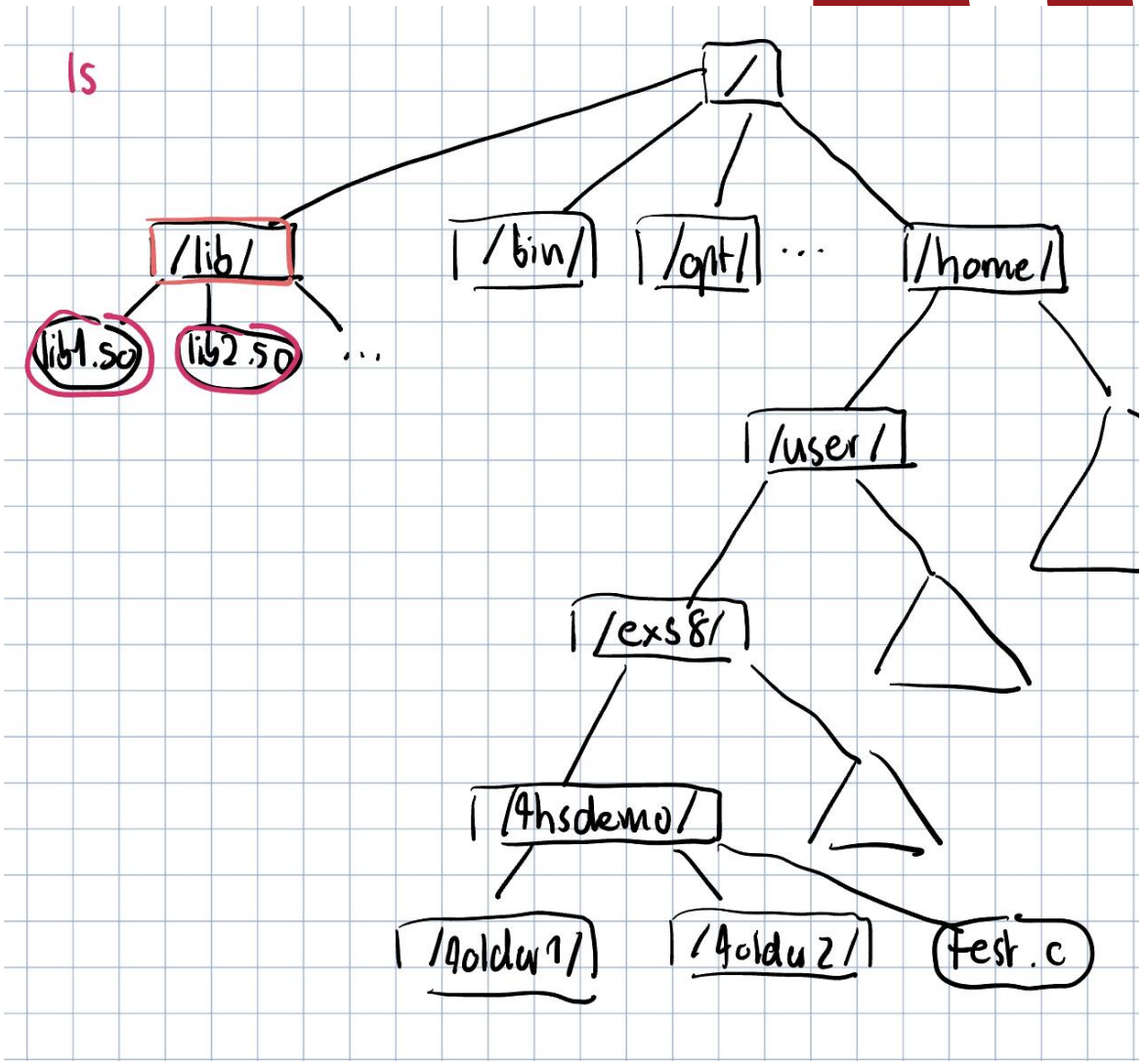
UNIX FHS

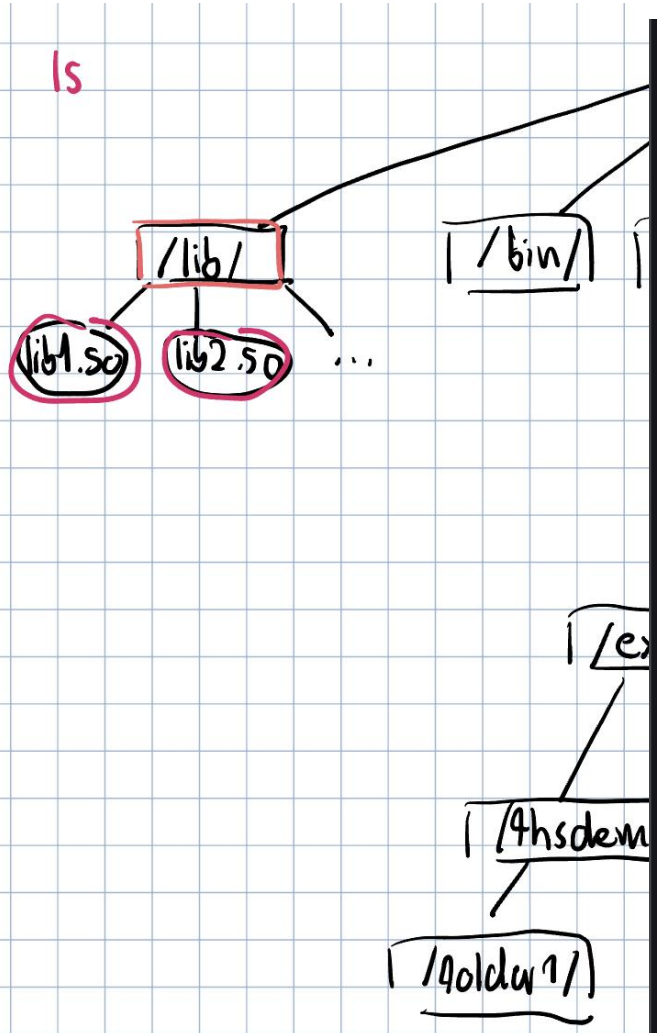



```
user@4865b4f533e3:/$ pwd
/
user@4865b4f533e3:/$ ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
```

UNIX FHS

- **Just** as binaries have their own locations in FHS (mostly /bin), **libraries** are also somewhere in **/lib** or **/usr/lib**






```
[bfalk@piora1 lib]$ pwd
/lib
[bfalk@piora1 lib]$ ls
audit                                   libnl-3.so.200
bfd-plugins                             libnl-3.so.200.26.0
binfmt.d                                libnl-genl-3.so.200
cmake                                   libnl-genl-3.so.200.26.0
cpp                                     libnl-idiag-3.so.200
crt1.o                                  libnl-idiag-3.so.200.26.0
crti.o                                  libnl-nf-3.so.200
crtn.o                                  libnl-nf-3.so.200.26.0
cups                                    libnl-route-3.so.200
debug                                   libnl-route-3.so.200.26.0
dkms                                    libnl-xfrm-3.so.200
dracut                                  libnl-xfrm-3.so.200.26.0
eclipse                                 libnss_compat.so.2
engines-3                               libnss_dns.so.2
enroot                                  libnss_files.so.2
environment.d                           libnuma.so.1
fipscheck                               libnuma.so.1.0.0
firewalld                               libnvcuvid.so
firmware                                libnvcuvid.so.1
fontconfig                              libnvcuvid.so.550.90.07
games                                   libnvidia-allocator.so
gcc                                     libnvidia-allocator.so.1
gconv                                   libnvidia-allocator.so.550.90.07
gcrt1.o                                 libnvidia-eglcore.so.550.90.07
gems                                    libnvidia-encode.so
gio                                     libnvidia-encode.so.1
girepository-1.0                        libnvidia-encode.so.550.90.07
```

Systems@ETH Zürich

Theory

Buffer Overflow Bugs and how to exploit them

Goal: Be able to use buffer overflows to change program behavior

Exploits

Systems@ETH zürich

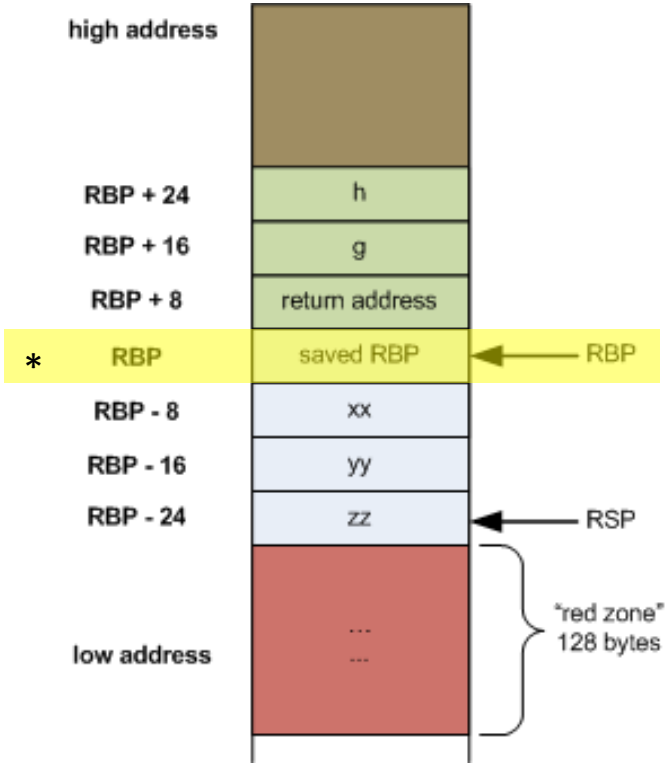
- 2 Step Process
  - Leverage memory corruption to control process execution
  - Direct process execution to injected shellcode

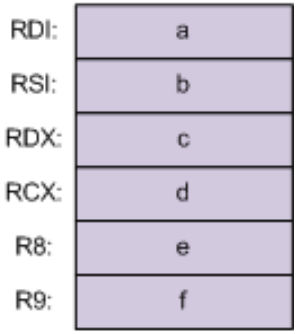
Exploits

Systems@ETH zürich

- 2 Step Process
  - Leverage memory corruption to control process execution
  - Direct process execution to injected shellcode
- 2 ways to get an exploit running:
  - Code Injection
  - Return Oriented Programming

Stack Frame Layout



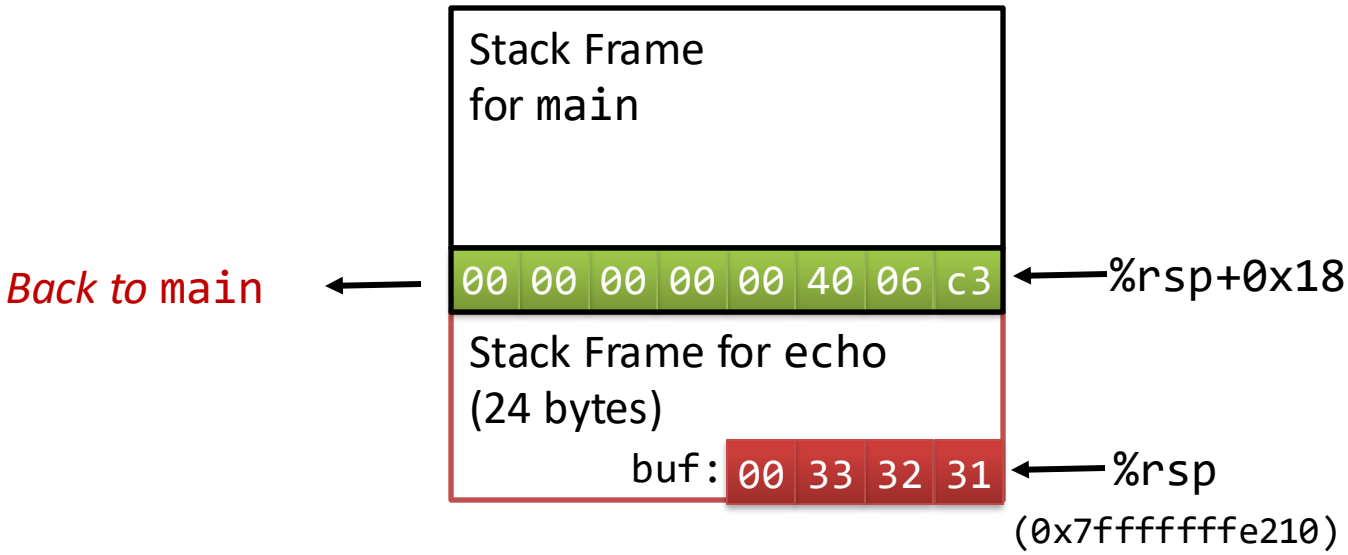



```
long myfunc(long a, long b, long c, long d,
            long e, long f, long g, long h)
{
    long xx = a * b * c * d * e * f * g * h;
    long yy = a + b + c + d + e + f + g + h;
    long zz = utilfunc(xx, yy, xx % yy);
    return zz + 20;
}
```

* Only mandatory, when stack size unknown

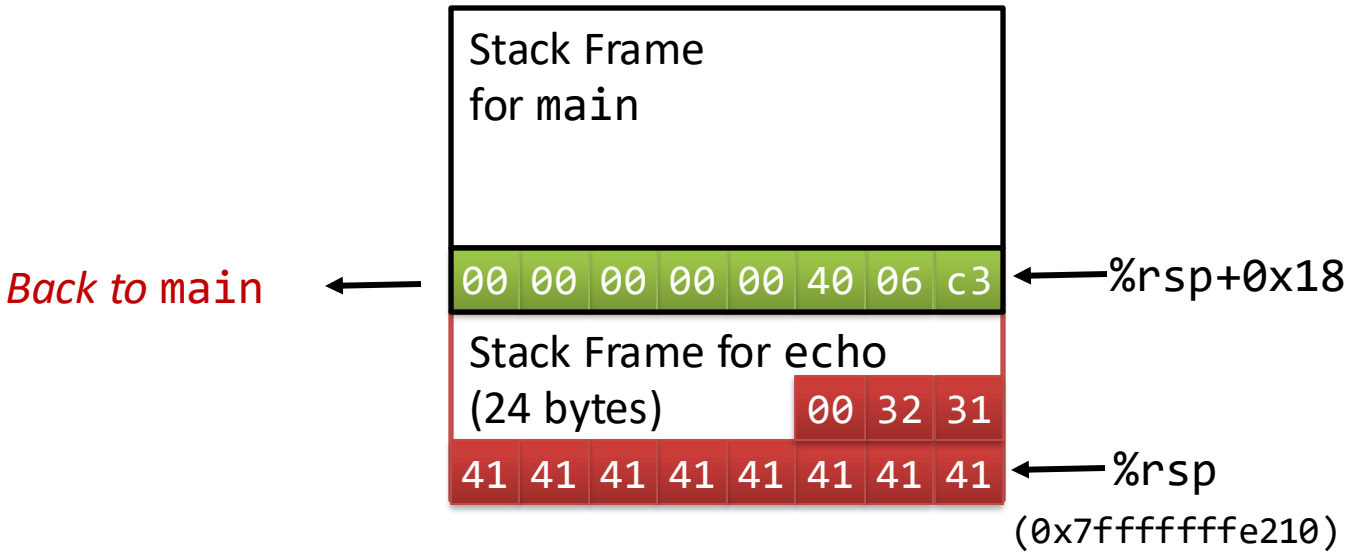
Buffer overflow stack

Input: `123`



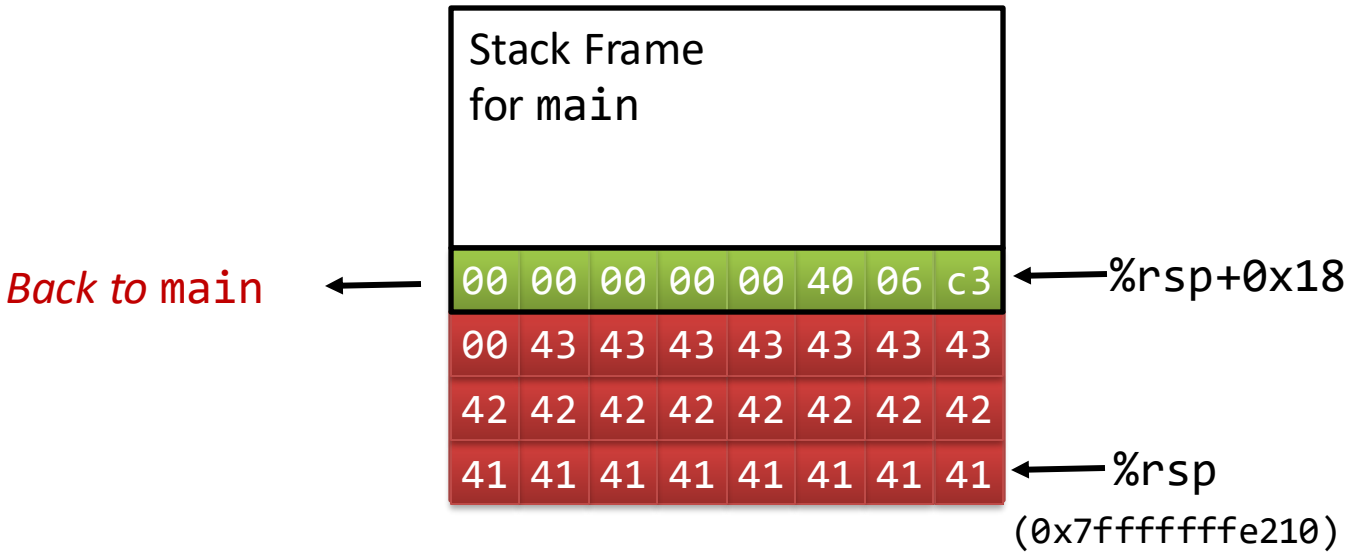
Buffer overflow stack

Input: AAAAAAAA12



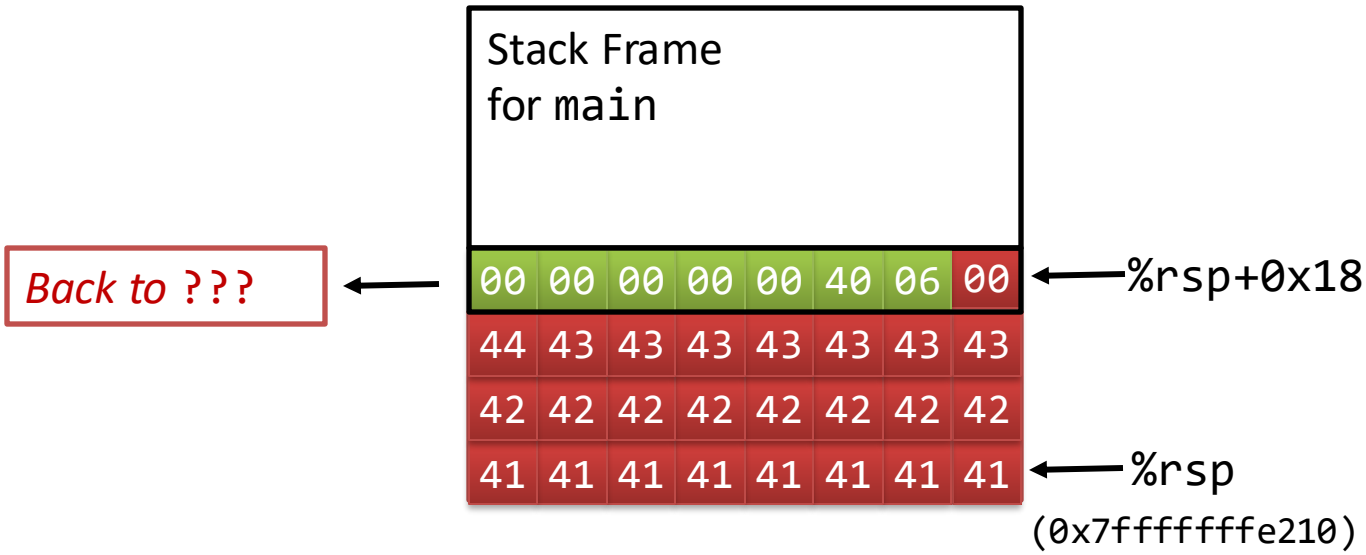
Buffer overflow stack

*Input: AA*AAAAAAABBBBBBBBCCCCCCC



Buffer overflow stack

*Input: AA*AAAAAAABBBBBBBBCCCCCCCD



What about something more interesting than crashing?

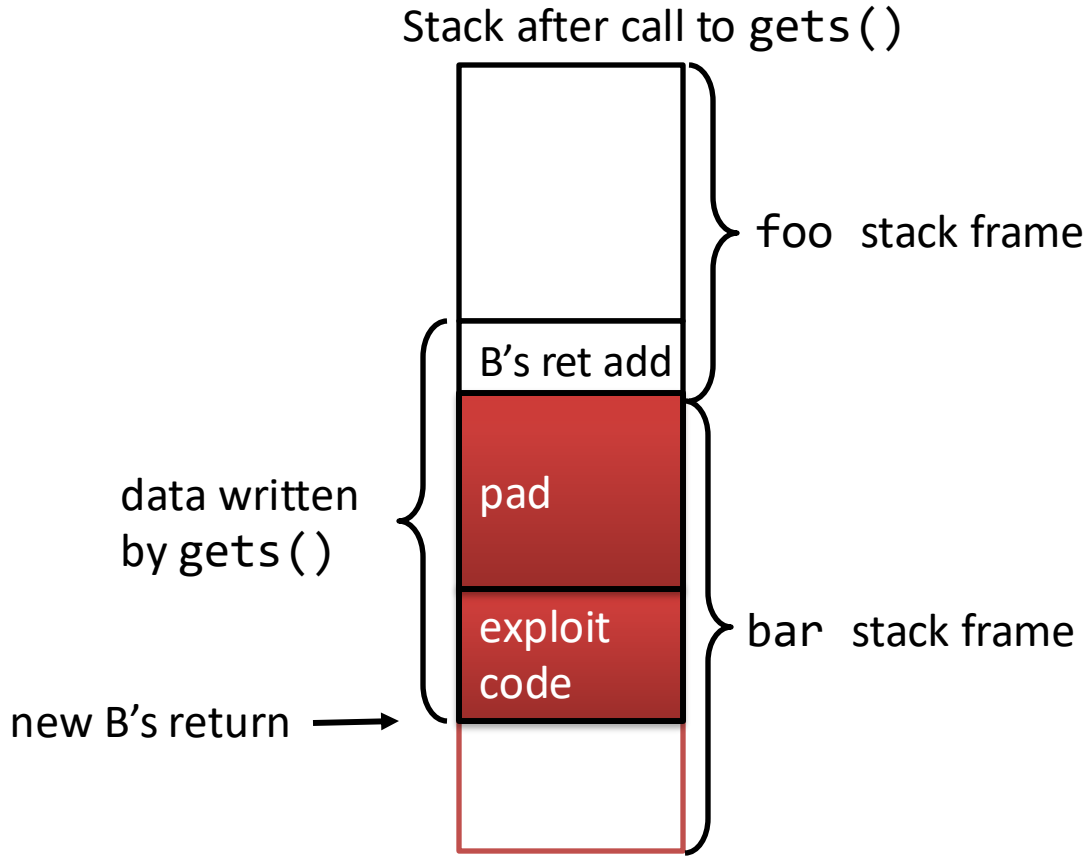
Malicious use of buffer overflow

Systems@ETH zürich

```
void foo(){
  bar();
  ...
}
```

return address A

```
int bar() {
  char buf[64];
  gets(buf);
  ...
  return ...;
}
```



- Input string contains byte representation of executable code
- Overwrite return address with address of buffer
- When `bar()` executes `ret`, will jump to exploit code

Code Injection

Systems@ETH zürich

Using a buffer overflow:

```
void echo() {
    char buf[4];
    gets(buf);
    puts(buf);
}
```

Problem with gets?

Code Injection

Using a buffer overflow:

```
void echo() {
    char buf[4];
    gets(buf);
    puts(buf);
}
```

No check for size!

Problem with gets?

What to do with this?

Systems@ETH zürich

Use buffer overflow to write code that gets executed by program

Option 1:

- Change the function to be called (or returned to)

Option 2:

- Push assembly code to the stack and use that for execution

System-level protections

- Compiler-inserted checks on functions
  - Compiler now understands library calls…

System-level protections

- Compiler-inserted checks on functions
  - Compiler now understands library calls…
- **Randomized stack offsets**
  - At start of program, allocate random amount of space on stack
  - Makes it difficult to predict beginning of inserted code

System-level protections

- Compiler-inserted checks on functions
  - Compiler now understands library calls…
- **Randomized stack offsets**
  - At start of program, allocate random amount of space on stack
  - Makes it difficult to predict beginning of inserted code
- **Nonexecutable code segments**
  - In older x86, can mark region of memory as either "read-only" or "writeable"
    - Can execute anything readable
  - Add explicit "execute" permission to hardware

Return Oriented Programming

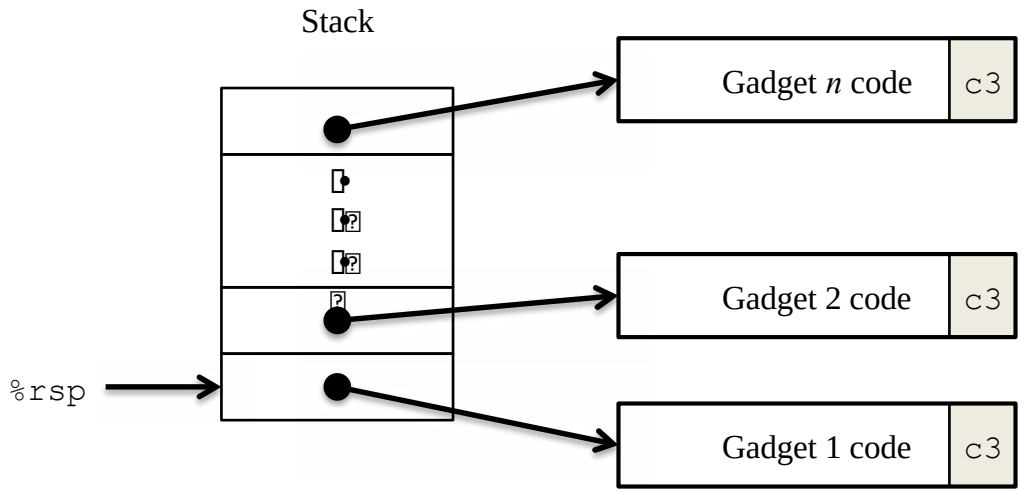
Systems@ETH zürich

- (Ab)use code snippets in the binary itself (aka “gadgets”) to change control flow
- Generally unsolved problem to defend against

Return Oriented Programming

Systems@ETH zürich

- In principle like code injection
  - Exploit buffer overflow but …
  - Use existing code gadgets
  - Build “stackframe“ from those
- Call different functions



ROP Example

Systems@ETH zürich

We want to:

- pop a value `0xBBBBBBBB` into `%rbx`
- move it into `%rax`

```
void foo(char *input) {
   char buf[32];
   …
   strcpy (buf, input);
   return;
}
```

ROP Example

Systems@ETH zürich

We want to:

- pop a value 0xBBBBBBBB into %rbx
- move it into %rax

```
void foo(char *input) {
    char buf[32];
    …
    strcpy (buf, input);
    return;
}
```

Our available gadgets:

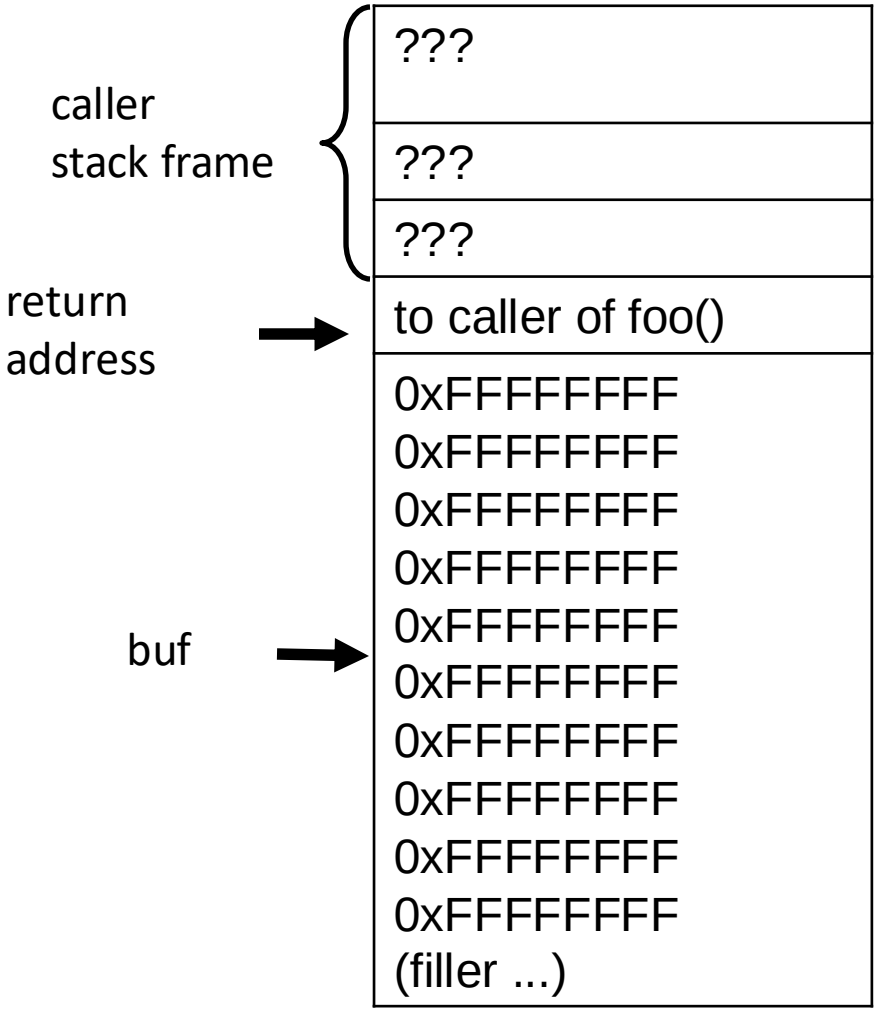
Address1:

```
mov %rbx, %rax;
```

Address2:

```
pop %rbx; ret
```

ROP Example




```
void foo(char *input) {
   char buf[32];
   …
   strcpy (buf, input);
   return;
}
```

Our available gadgets:

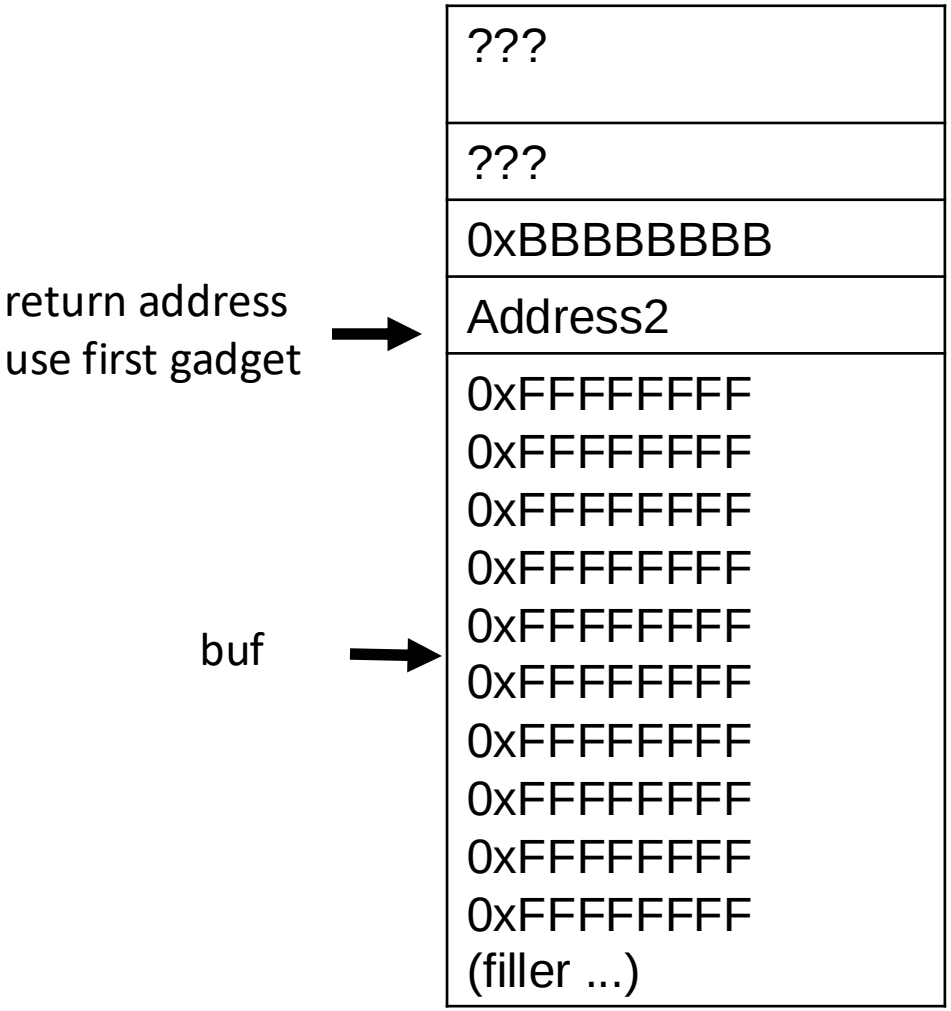
Address1:

```
mov %rbx, %rax;
```

Address2:

```
pop %rbx; ret
```

ROP Example




```
void foo(char *input) {
   char buf[32];
   …
   strcpy (buf, input);
   return;
}
```

Our available gadgets:

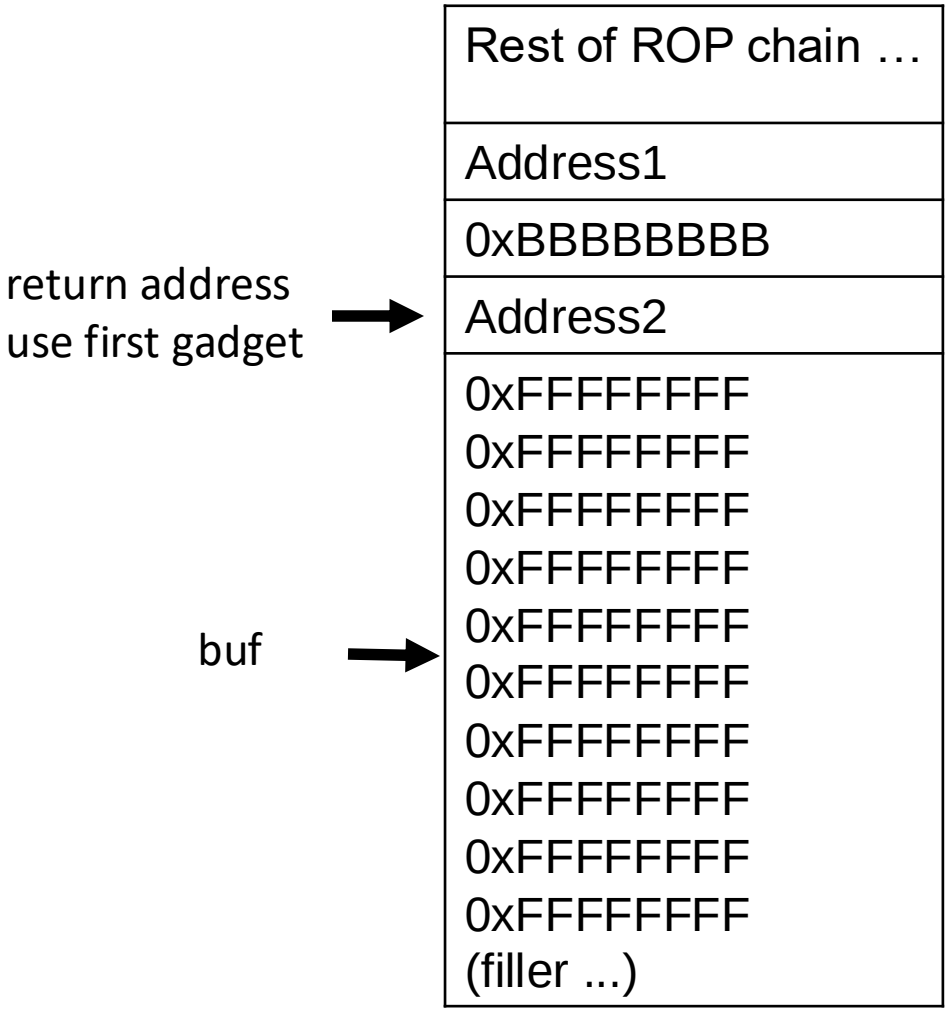
Address1:

```
mov %rbx, %rax;
```

Address2:

```
pop %rbx; ret
```

ROP Example




```
void foo(char *input) {
   char buf[32];
   …
   strcpy (buf, input);
   return;
}
```

Our available gadgets:

Address1:

```
mov %rbx, %rax;
```

Address2:

```
pop %rbx; ret
```

Hands On Demonstration

Overview of this session

- Questions Regarding Linking
- Theory on Exploits
- Preview assignment 7: Attack-Lab

Assignment 7

Systems@ETH zürich

- We have prepared 2 custom targets for you
  - assignment7/target*k*/(c|r)target
- Extra tools included:
  - Readme
  - hex2raw converter
  - “Gadget farm” source for rtarget

Other tools you may want

Systems@ETH zürich

- Use **objdump** for stack layout
- **gdb** for debugging
- **gcc** -c assembly.s for machine code generation
- May need some padding data
- You might want to use a script for automation

Overview

Systems@ETH zürich

- Goal is to teach you concepts and implementation of
  - Code Injection (ctarget)
  - Return Oriented Programming (rtarget)
- Deadline: 1 week

Target Programs

Systems@ETH zürich

- ctarget:
  - 3 phases, increasingly difficult
  - Uses pure code injection, very static

  Main focus, this we expect you to understand

- rtarget:
  - 2 phases, quite difficult

  Can still be used on modern executables

ctarget (Part 1)

Systems@ETH zürich

- Level 1
  - Similar to code exploit demonstrated
  - Call touch1()
- Level 2
  - Insert some small amount of code to fool the comparator
  - Call touch2()
- Level 3
  - Similar to previous level, but expects a string as a passed argument
  - Call touch3()

rtarget (Part 2)

Systems@ETH zürich

- Level 1
  - Repetition of ctarget's level 2 using gadgets
  - Call touch2()
- Level 2
  - Repetition of ctarget's level 3 using gadgets
  - Call touch3()

Important gadgets:

`movq/movl` (see writeup)

`popq` (see writeup)

`ret 0xc3`

`nop 0x90`

Helpful Material

Systems@ETH zürich

- Lab writeup
- Exercise sessions
- Lecture
- Google/DuckDuckGo: “Code injection” & “Return oriented programming” & “CTF writeup” of those

Systems@ETH zürich

Have fun!