OSY.SSI[2015][7]

Code injection

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

If you'd like to know more, you can search online later for this error: HAL_INITIALIZATION_FAILED

Bugs are our friends

Lesson:

Bugs are our friends

Lesson: Bugs tell us something interesting about *how things work*.

Bugs are our friends

Lesson: Bugs tell us something interesting about *how things work*.

Pedagogy:

Bugs are our friends

Lesson: Bugs tell us something interesting about *how things work*.

Pedagogy: We're gonna cause bugs.

Bugs are our friends

Lesson: Bugs tell us something interesting about *how things work*.

Pedagogy: We're gonna cause bugs. Plenty.

This will be the most technically challenging lecture.

Also probably the most interesting and useful.

We are going to talk about code injection.

Table of Contents

Preliminaries

Everything stems from this observation:

$$\text{Model} \neq \text{Reality}$$

Math *vs.* computers

Task:

- Write a program that takes two integer variables x and y and prints the value $x + y$.

Math *vs.* computers

Task:

- Write a program that takes two integer variables x and y and prints the value $x + y$.
- Choose $x > 0$ and $y > 0$ such that $x + y = 0$

Math *vs.* computers

Task:

- Write a program that takes two integer variables x and y and prints the value $x + y$.
- Choose $x > 0$ and $y > 0$ such that $x + y = 0$
- Define x and y as `floats` and find $x > 0$ and $y > 0$ such that $x + y = x$.

Math *vs.* computers

Task:

- Write a program that takes two integer variables x and y and prints the value $x + y$.
- Choose $x > 0$ and $y > 0$ such that $x + y = 0$
- Define x and y as `floats` and find $x > 0$ and $y > 0$ such that $x + y = x$.
- Find x and y two non-zero `float` values such that $x/y = \infty$.

Math *vs.* computers

Task:

- Write a program that takes two integer variables x and y and prints the value $x + y$.
- Choose $x > 0$ and $y > 0$ such that $x + y = 0$
- Define x and y as `floats` and find $x > 0$ and $y > 0$ such that $x + y = x$.
- Find x and y two non-zero `float` values such that $x/y = \infty$.
- Find x such that $x \neq x$.

Math *vs.* computers

Task:

- Write a program that takes two integer variables x and y and prints the value $x + y$.
- Choose $x > 0$ and $y > 0$ such that $x + y = 0$
- Define x and y as `floats` and find $x > 0$ and $y > 0$ such that $x + y = x$.
- Find x and y two non-zero `float` values such that $x/y = \infty$.
- Find x such that $x \neq x$.

WTF?

Math *vs.* computers

Task:

- Write a program that takes two integer variables x and y and prints the value $x + y$.
- Choose $x > 0$ and $y > 0$ such that $x + y = 0$
- Define x and y as `floats` and find $x > 0$ and $y > 0$ such that $x + y = x$.
- Find x and y two non-zero `float` values such that $x/y = \infty$.
- Find x such that $x \neq x$.

WTF? IEEE 754, ACM Turing prize.

Math *vs.* computers

Task:

- Write a program that takes two integer variables x and y and prints the value $x + y$.
- Choose $x > 0$ and $y > 0$ such that $x + y = 0$
- Define x and y as `floats` and find $x > 0$ and $y > 0$ such that $x + y = x$.
- Find x and y two non-zero `float` values such that $x/y = \infty$.
- Find x such that $x \neq x$.

WTF? IEEE 754, ACM Turing prize.

Also, $x + (y + z) \neq (x + y) + z, x(y + z) \neq xy + xz$ etc.

Math *vs.* computers

$$\texttt{int} \neq \mathbb{Z}$$
$$\texttt{float} \neq \mathbb{R}$$

Just a silly bug.. er... feature?

- The Patriot Missile incident (Dharan, 1991): 1/10.

Just a silly bug.. er... feature?

- The Patriot Missile incident (Dharan, 1991): 1/10. 28 dead, 100 injured.

Just a silly bug.. er... feature?

- The Patriot Missile incident (Dharan, 1991): 1/10. 28 dead, 100 injured.
- Ariane 5 maiden flight (1996): 64-bit to 16-bit.

Just a silly bug.. er... feature?

- The Patriot Missile incident (Dharan, 1991): 1/10. 28 dead, 100 injured.
- Ariane 5 maiden flight (1996): 64-bit to 16-bit. $370 million.

Just a silly bug.. er... feature?

- The Patriot Missile incident (Dharan, 1991): 1/10. 28 dead, 100 injured.
- Ariane 5 maiden flight (1996): 64-bit to 16-bit. $370 million.
- Therac-25 radiation machine (1985–1987)

Just a silly bug.. er... feature?

- The Patriot Missile incident (Dharan, 1991): 1/10. 28 dead, 100 injured.
- Ariane 5 maiden flight (1996): 64-bit to 16-bit. $370 million.
- Therac-25 radiation machine (1985–1987)
- WebKit (ACE, CVE-2009-2195), Ruby (ACE, CVE-2013-4164), Flash (ACE, CVE-2015-3077, CVE-2014-0502), ...

Fact:

Just a silly bug.. er... feature?

- The Patriot Missile incident (Dharan, 1991): 1/10. 28 dead, 100 injured.
- Ariane 5 maiden flight (1996): 64-bit to 16-bit. $370 million.
- Therac-25 radiation machine (1985–1987)
- WebKit (ACE, CVE-2009-2195), Ruby (ACE, CVE-2013-4164), Flash (ACE, CVE-2015-3077, CVE-2014-0502), ...

Fact: Most programmers forget that.

Just a silly bug.. er... feature?

- The Patriot Missile incident (Dharan, 1991): 1/10. 28 dead, 100 injured.
- Ariane 5 maiden flight (1996): 64-bit to 16-bit. $370 million.
- Therac-25 radiation machine (1985–1987)
- WebKit (ACE, CVE-2009-2195), Ruby (ACE, CVE-2013-4164), Flash (ACE, CVE-2015-3077, CVE-2014-0502), ...

Fact: Most programmers forget that. Please don't.

Just a silly bug.. er... feature?

- The Patriot Missile incident (Dharan, 1991): 1/10. 28 dead, 100 injured.
- Ariane 5 maiden flight (1996): 64-bit to 16-bit. $370 million.
- Therac-25 radiation machine (1985–1987)
- WebKit (ACE, CVE-2009-2195), Ruby (ACE, CVE-2013-4164), Flash (ACE, CVE-2015-3077, CVE-2014-0502), ...

Fact: Most programmers forget that. Please don't.

Table of Contents

This lecture : Notions of low-level software

This lecture will cover the basics of low-level soft.

Why? Because it's interesting.

Also it is necessary for the Lab next time.

Next lecture: Lab

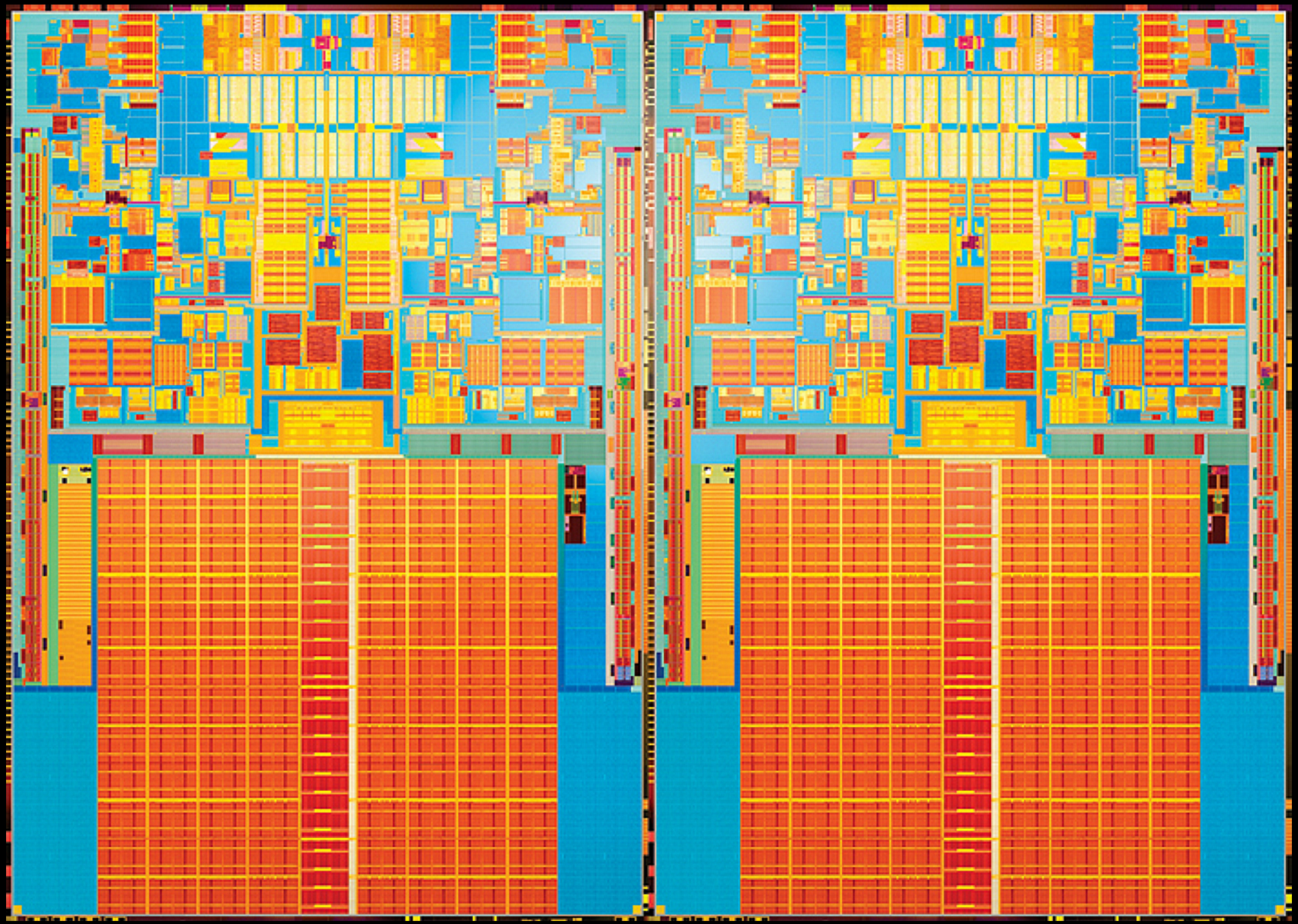
Next time you learn and practice code injection.

Especially stack smashing and return-oriented programming.

So pay attention (especially to the stack!)

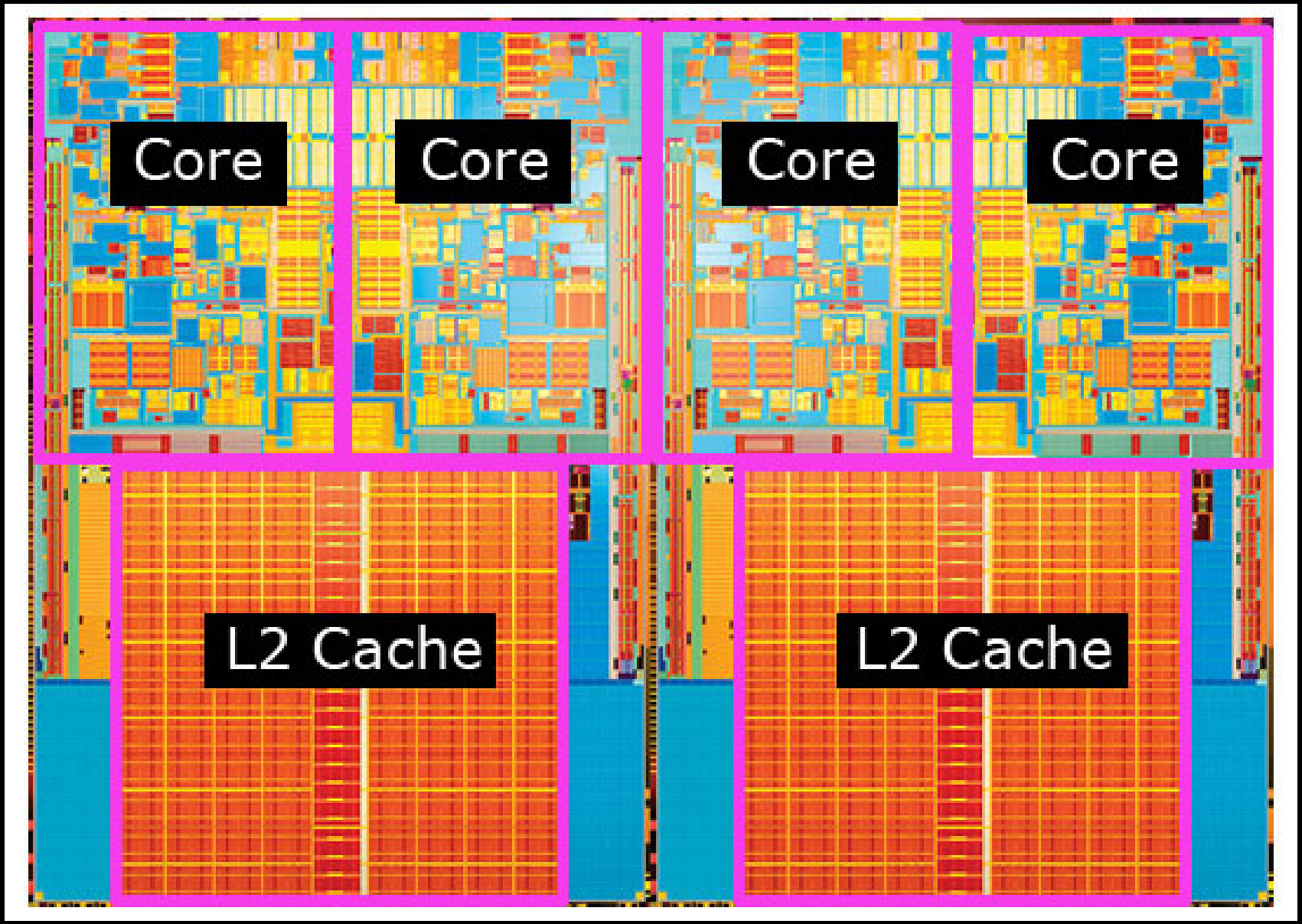
CPU

Central Pwning Unit – Pictured: Intel® Core™ 2 Quad



CPU

Central Pwning Unit – Pictured: Intel® Core™ 2 Quad



CPU

Central Pwning Unit

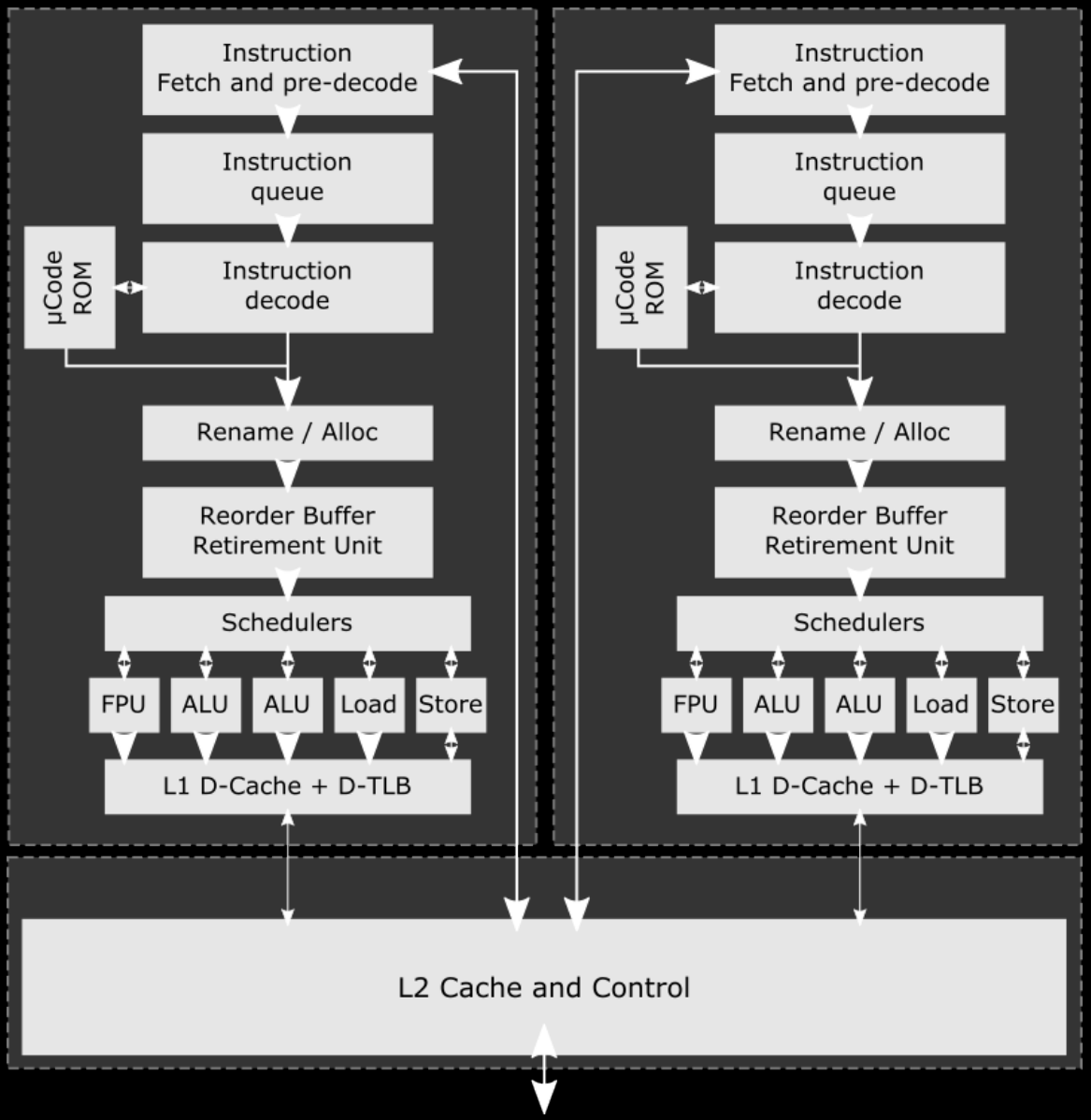
Essentially 3 parts:

- Memory (RAM, L1, L2, L3 caches, registers)
- Computation (Core, ALU, FPU)
- Communication (MMU, bus, etc.)

We will (mostly) focus on memory. Here's why.

CPU

Central Pwning Unit – Picture: Intel® Core™ Architecture (2 cores)



CPU

Why we target memory

“L’information, c’est exactement le système du contrôle.”

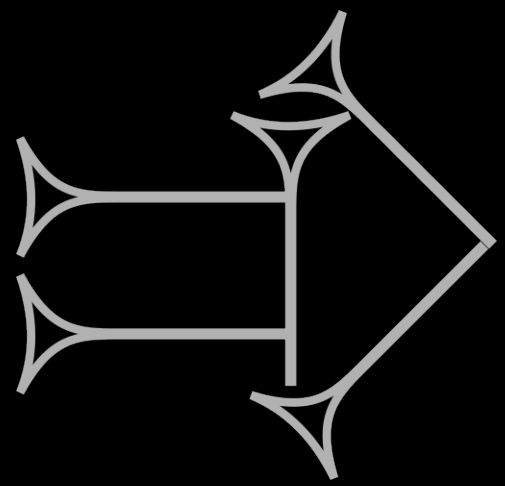
— Gilles Deleuze, Conférence du 17 mai 1987

Table of Contents

Instructions and data

Intructions or data: John von Neumann

What distinguishes data from commands is *how you use them*.



- Sumerian pictograph: gu_4
- Akkadian pictograph: *alp*
- Akkadian/Hittite cuneiform: *'a*

Instructions and data

Programming

Programming is simple:

> We send data. Data is understood as commands. Easy peasy.

Ok so now what do commands look like?

A standard description is called *assembly*.

Instructions and data

Registers

Registers $\approx$ variables.

Example (Intel)

- 4 GP 64-bit registers: `rax, rbx, rcx, rdx`
- 2 stack registers: `rbp, rsp`
- 2 string registers: `rsi, rdi`
- instruction register: `rip`
- ...

"Flags" $\approx$ implicit result holders (e.g. `ZF, CF`,...)

Instructions and data

Arithmetics

A generic instruction :

```
OPCODE [DST / ARG1], [ARG2]
```

Examples:

`mov eax, ebx`	$(eax \leftarrow ebx)$
`add eax, ebx`	$(eax \leftarrow eax + ecx)$
`xor eax, eax`	$(eax \leftarrow eax \oplus eax)$

(may modify flags too)

Instructions and data

Comparisons and jumps

Examples:

`jmp 0xbadc0fee`	(goto address `0xbadc0fee`)
`cmp eax, ebx`	($eax \stackrel{?}{=} ebx$, (implicit))
`jne 0xdeadbeef`	(if $eax \neq ebx$ goto `0xdeadbeef`)

(may modify flags too)

Instructions and data

A very simple example (GCC)

```
000f: movl $0, ebx
0016: jmp 0021
0018: call beep
001d: addl $1, ebx
0021: cmpl $9, ebx
0025: jle 0018
0027: movl $0, ebx
```

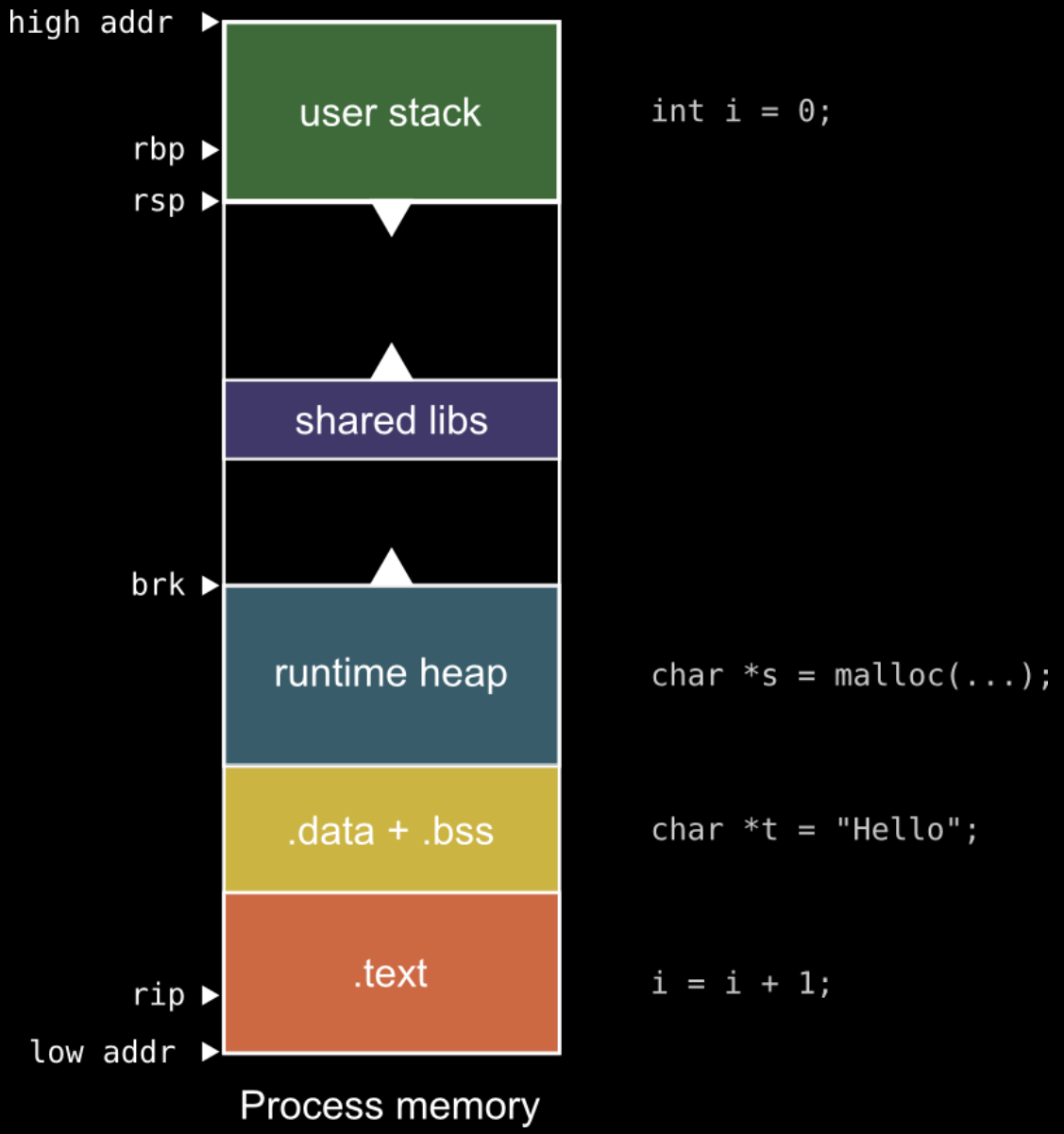
```
000f: BB00000000
0016: EB0E
0018: E800000000
001d: 83C301
0021: 83FB09
0025: 7EEC
0027: BB00000000
```

More on that next time.

Table of Contents

The stack

The program's *Weltanschauung*



How functions work

What is a “function”?

How functions work

What is a “function”?

- A subroutine

How functions work

What is a “function”?

- A subroutine
- (possibly) with arguments

How functions work

What is a “function”?

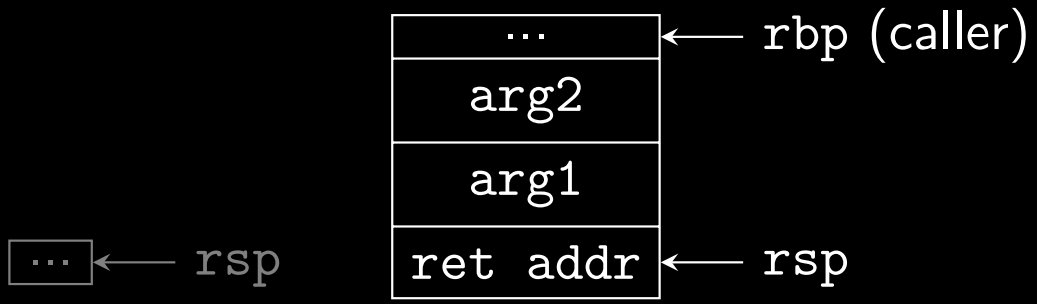
- A subroutine
- (possibly) with arguments
- that (possibly) returns a result.

How functions work

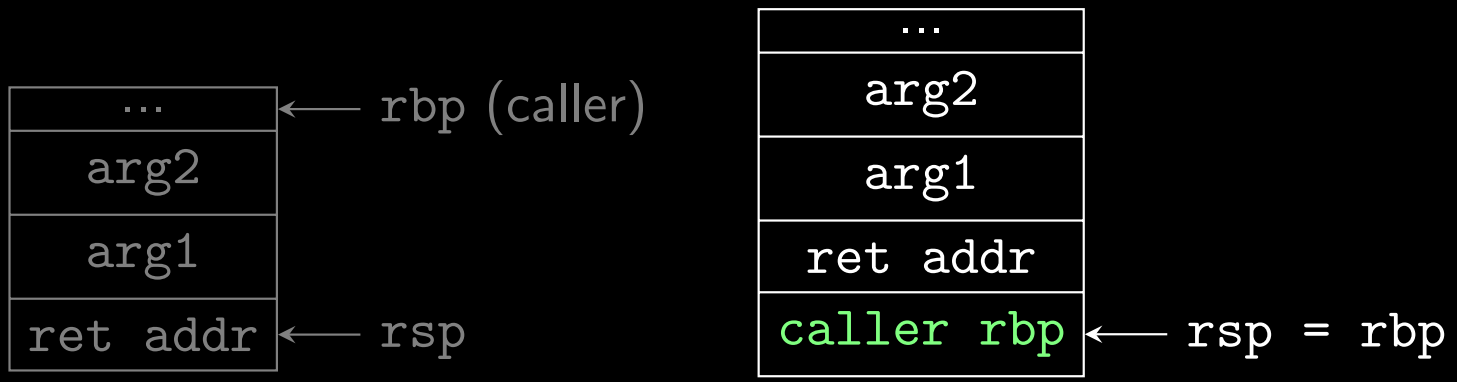
```
void main() {
    function1(1, 2);
}

void function1(int arg1, int arg2) {
  int a;
  int buffer[10];
  function2(12, 42);
}

void function2(int arg3, int arg4) {
  // do stuff
}
```

How functions work

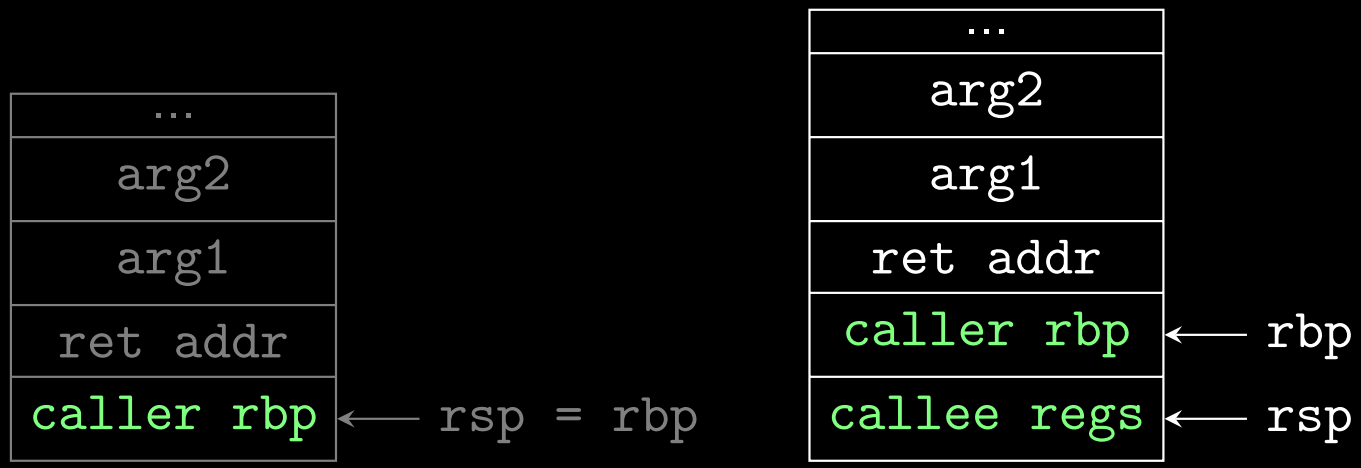
main called `function1`



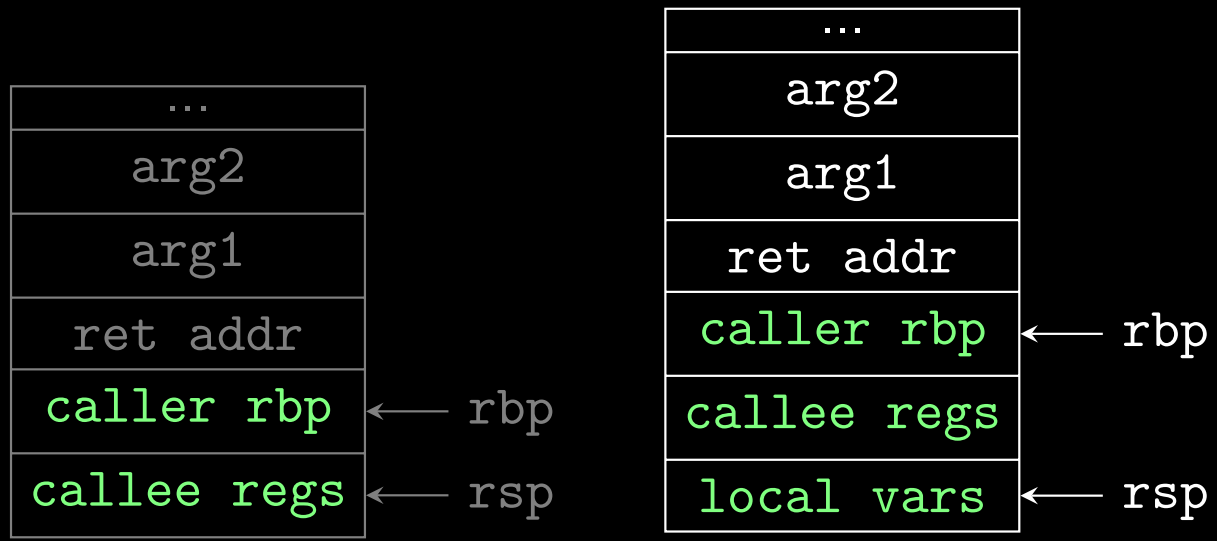
`function1` takes control : push caller rbp, rbp ← rsp, arg1 at rbp+8 or rbp+16



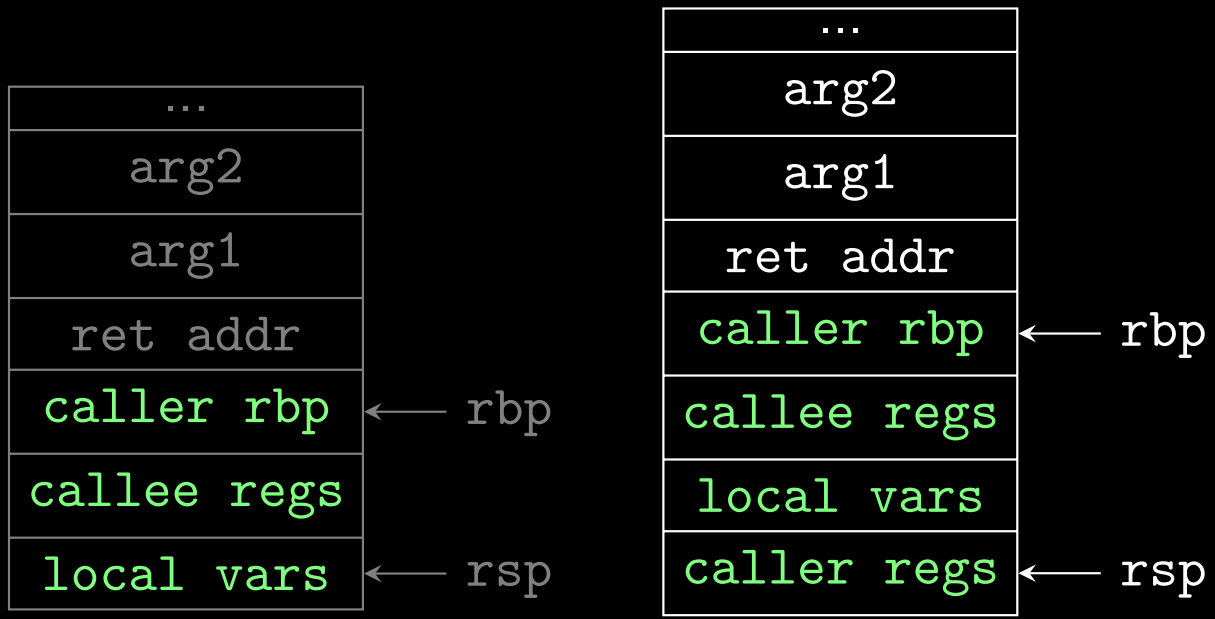
Save registers if used (rdi, rsi, rbx...)



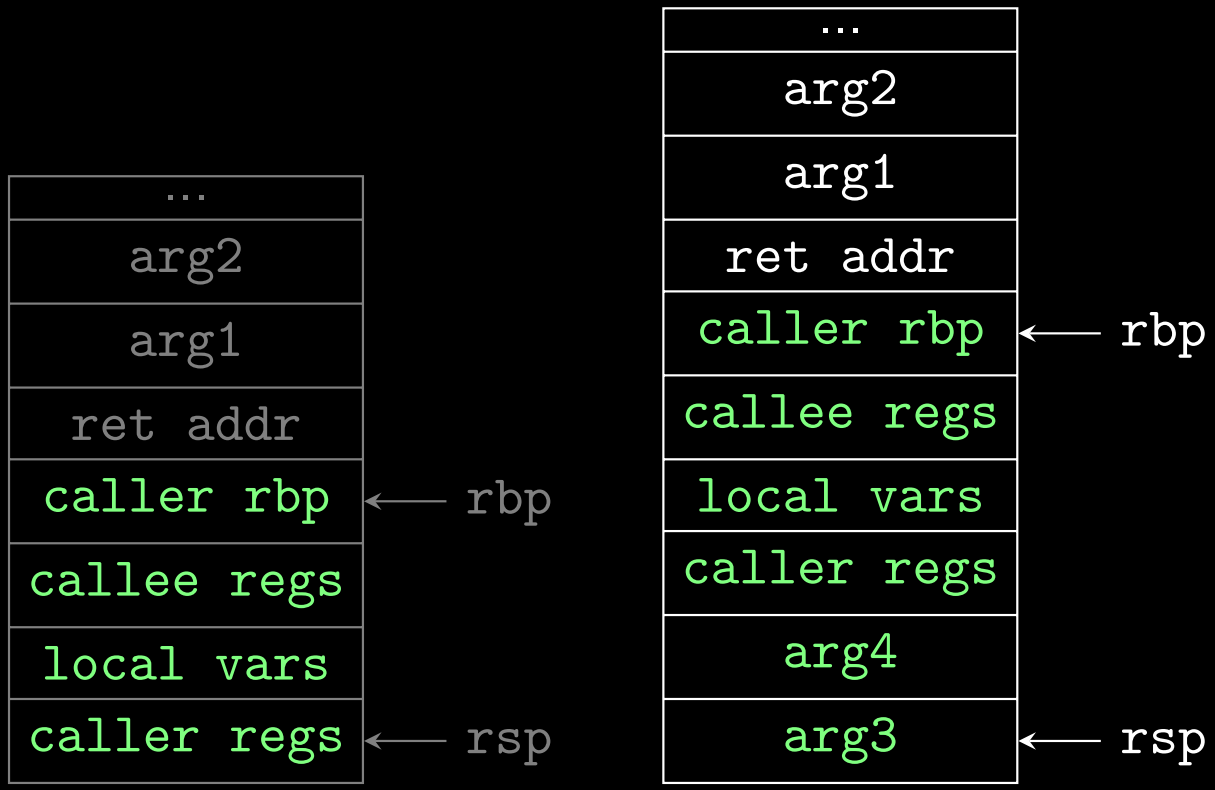
Allocation for local variables (esp ← esp - size)



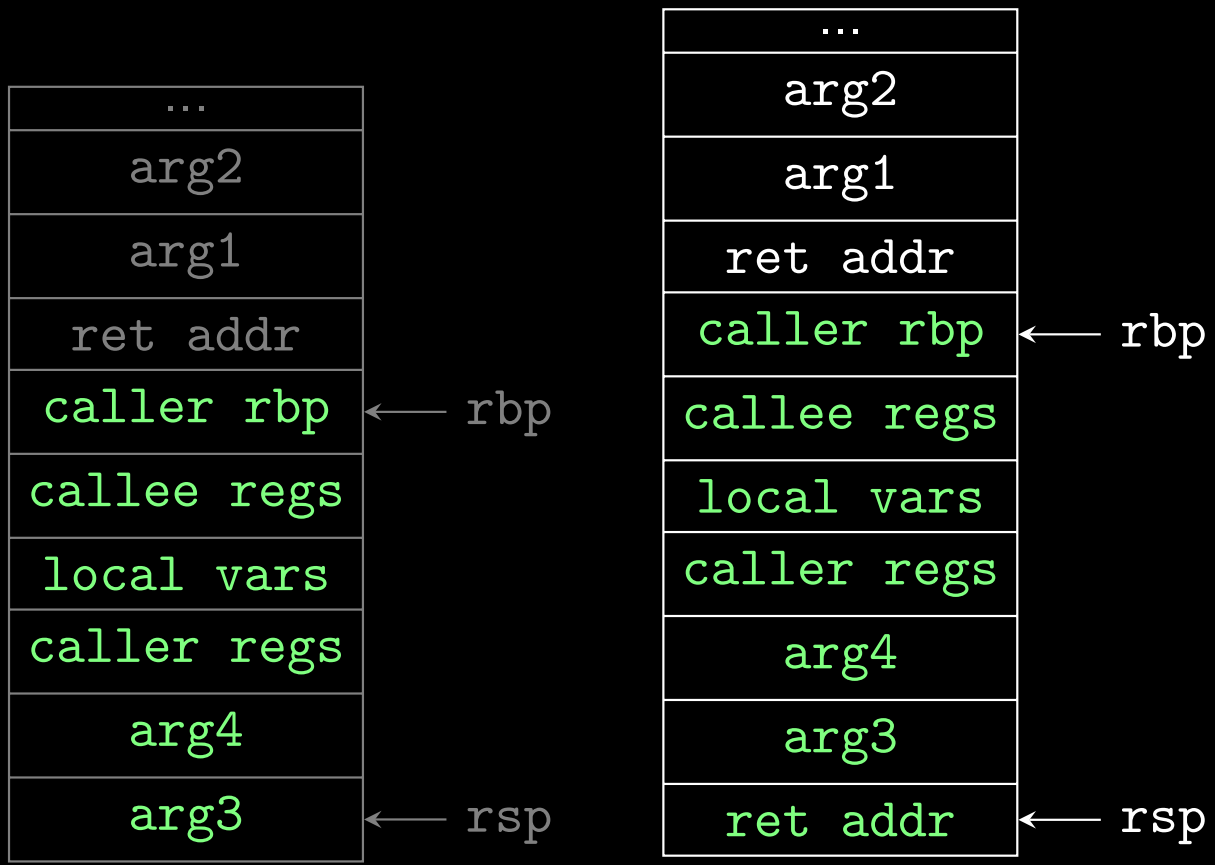
Start calling `function2` : save registers if needed



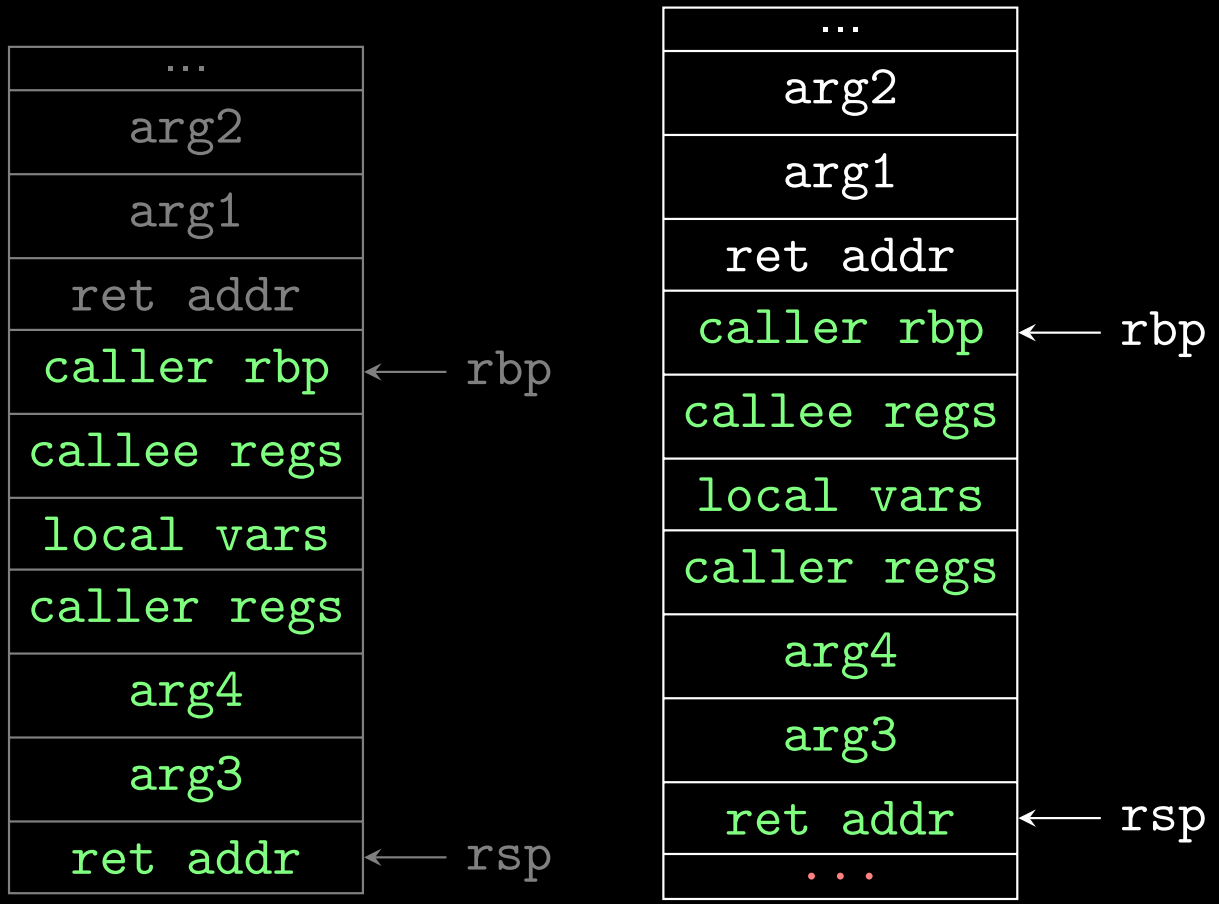
Push arguments for `function2`:



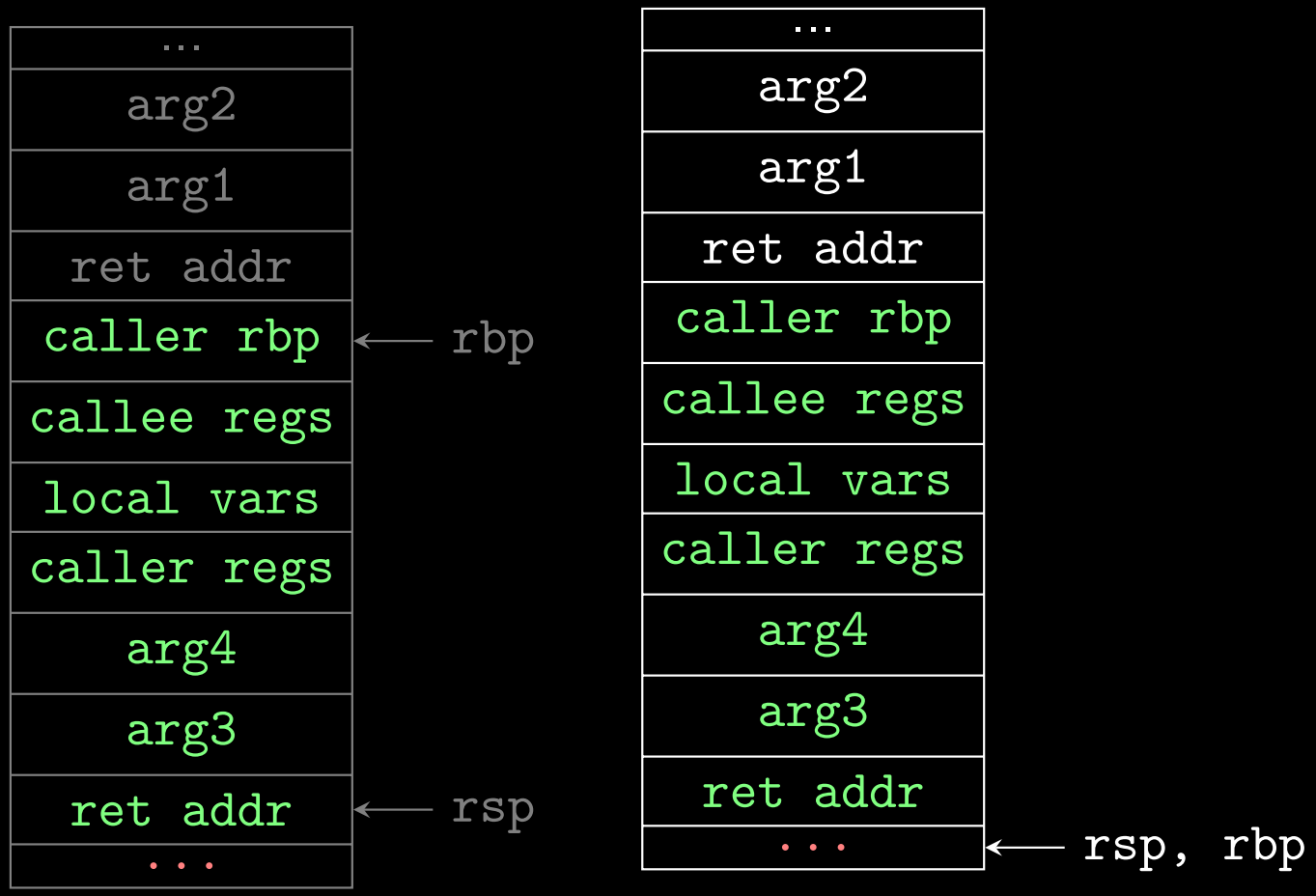
Push return address



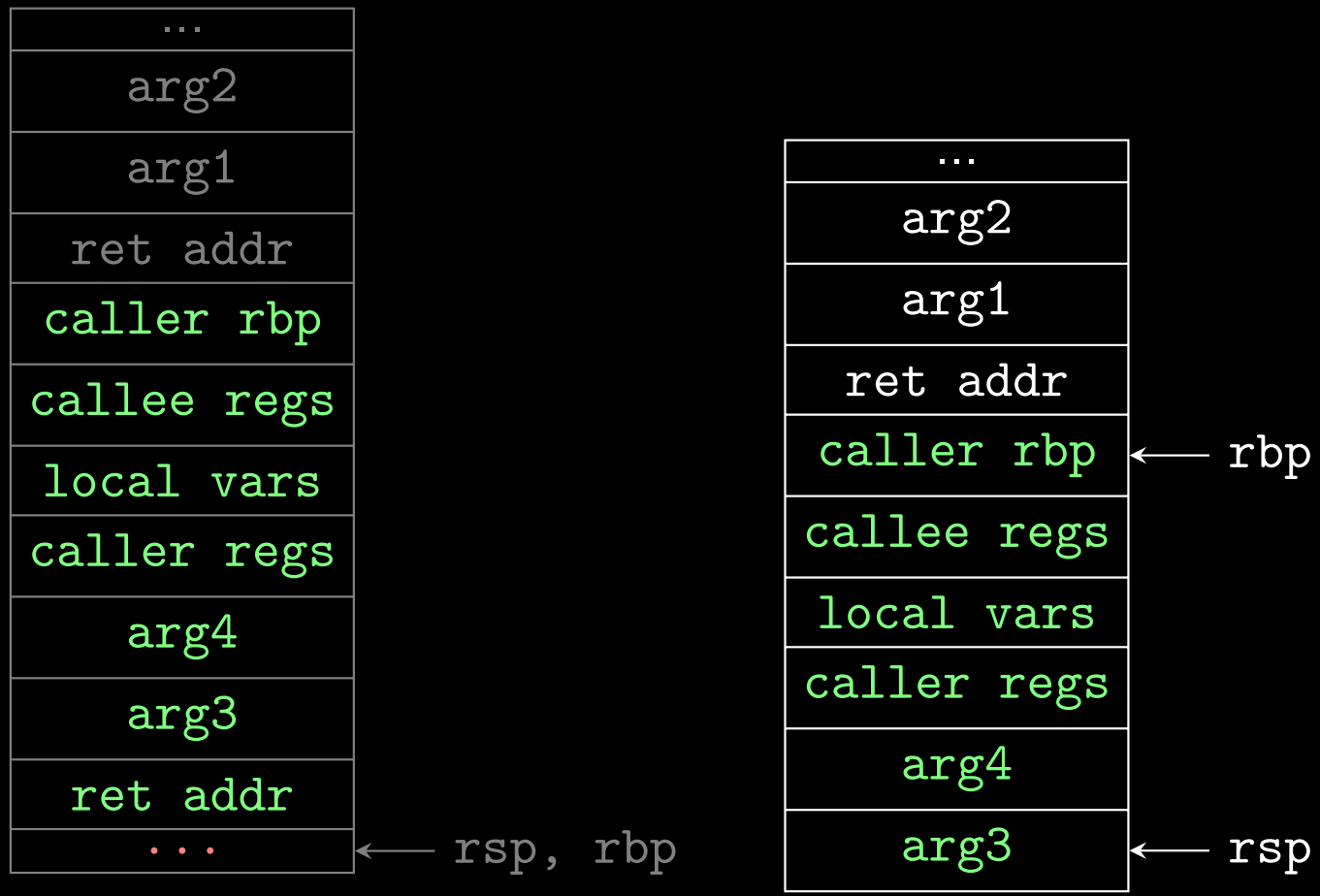
Pass control to `function2`.



Then `function2` does its stuff.



When `function2` has finished, it cleans up: stores its return value in rax, rsp += size, pop registers, pop rbp, ret (=pop ret + jmp there)



Then `function1` finishes and cleans up: rsp += size, pop registers, pop rbp, ret.

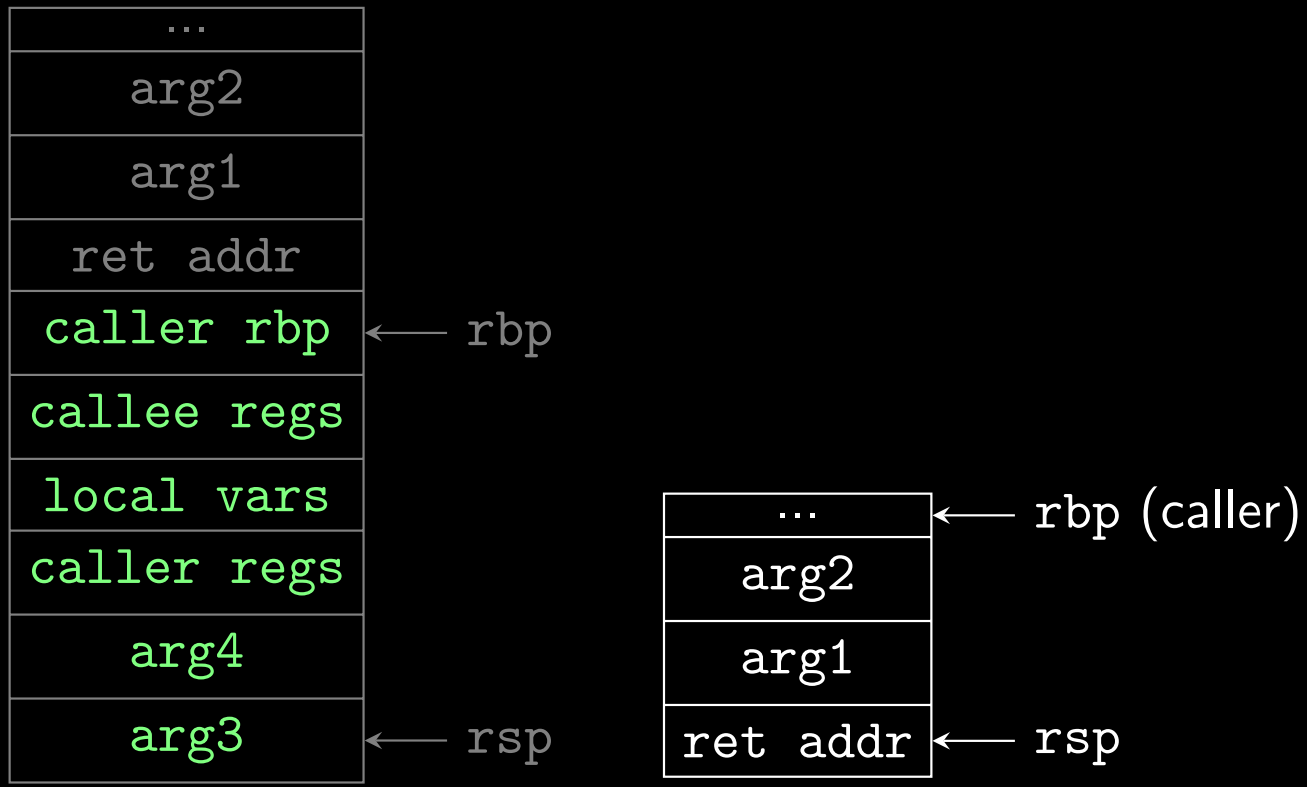


Table of Contents

Syscalls

You can do any data processing you want.

Syscalls

You can do any data processing you want.
But that's not interesting.

Syscalls

You can do any data processing you want.
But that's not interesting.

e.g. You can't write on the screen!

Syscalls

Asking for permission

You ask the kernel to do this kind of things for you.

```
syscall(4, 1, s, L)
```

Syscalls

Asking for permission

You ask the kernel to do this kind of things for you.

`syscall(4, 1, s, L)`

- Syscall 4 is “write” (there is a complete list)
- 1 means `stdout`
- s is the string you want to print
- L is its length

Syscalls

Next time

Next time we will analyse the following file:

```
\x31\xc9\xf7\xe1\x51\x68\x2f
\x2f\x73\x68\x68\x2f\x62\x69
\x6e\x89\xe3\xb0\x0b\xcd\x80
```

It is a (short) program that performs one syscall!

Table of Contents

Our friend the segmentation fault

Demo ?

Our friend the segmentation fault

Next time

Next time we will analyse this program:

- Reveal the password
- Bypass password protection
- Make the program do what we want

... just by sending the appropriate passwords.