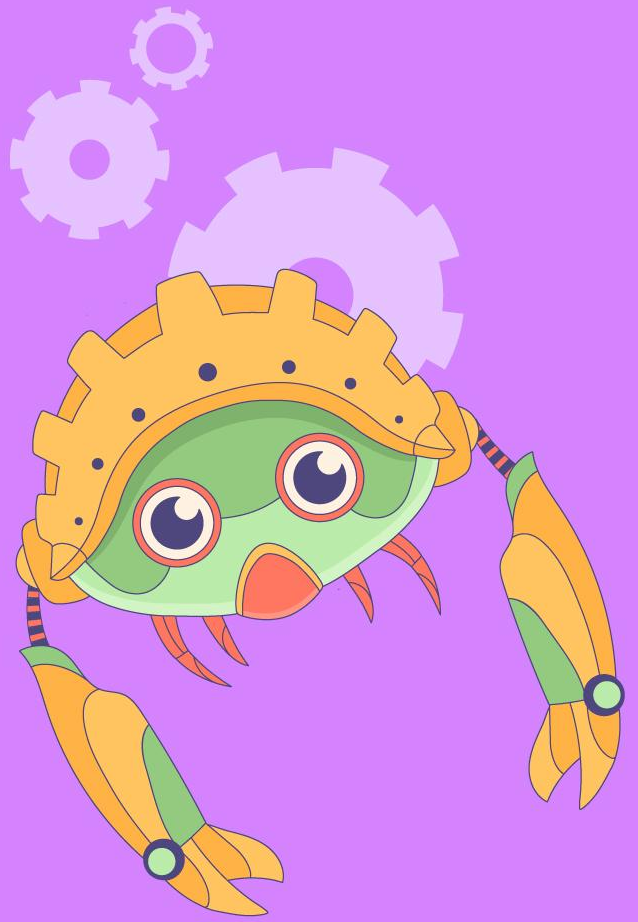
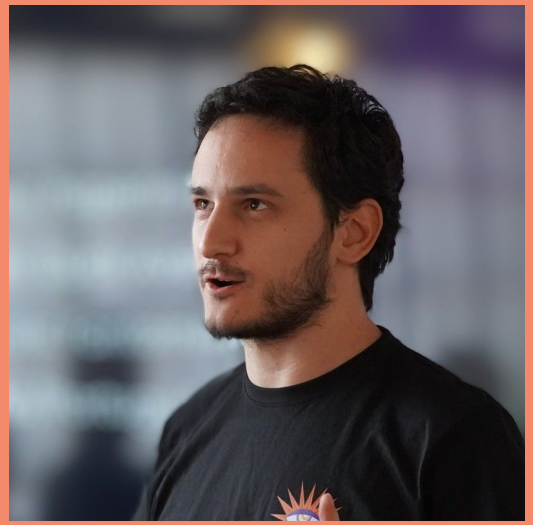
Tommaso Allevi

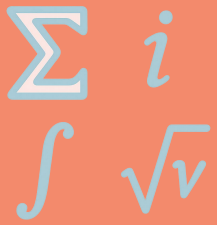
Software Dev/Eng @ Orama

# Track your memory allocations



# About me

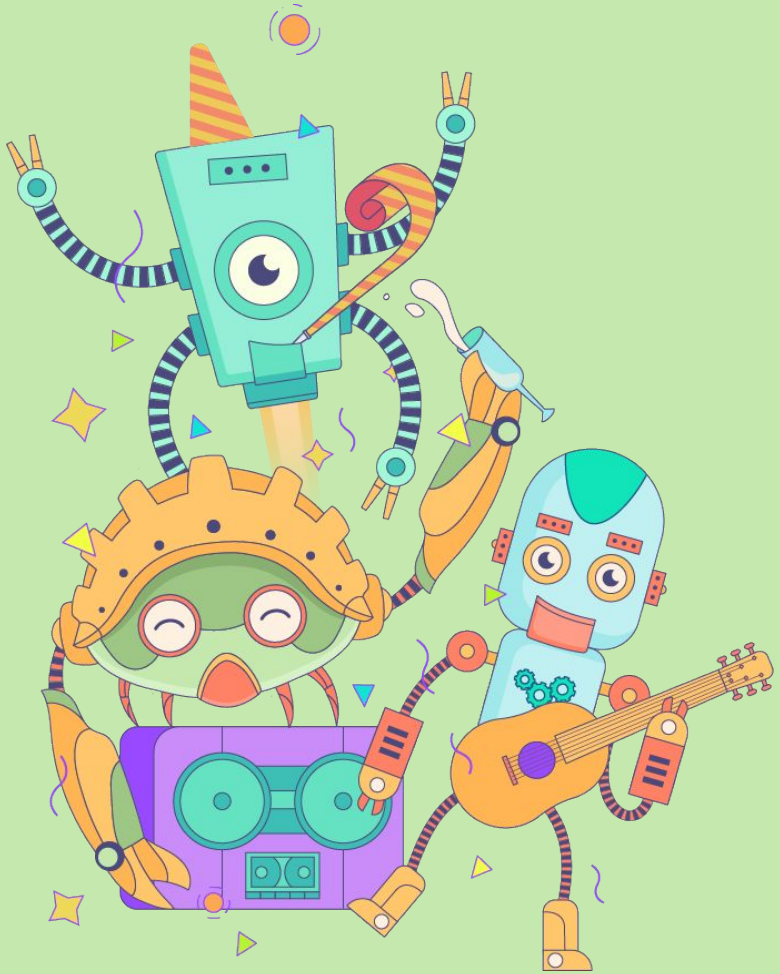




# Disclaimers

- I’m not a UI designer
- I love to feel inexperienced
- I’m bold-opinionated person
- *Any* feedback are appreciated

# Some questions before starting...

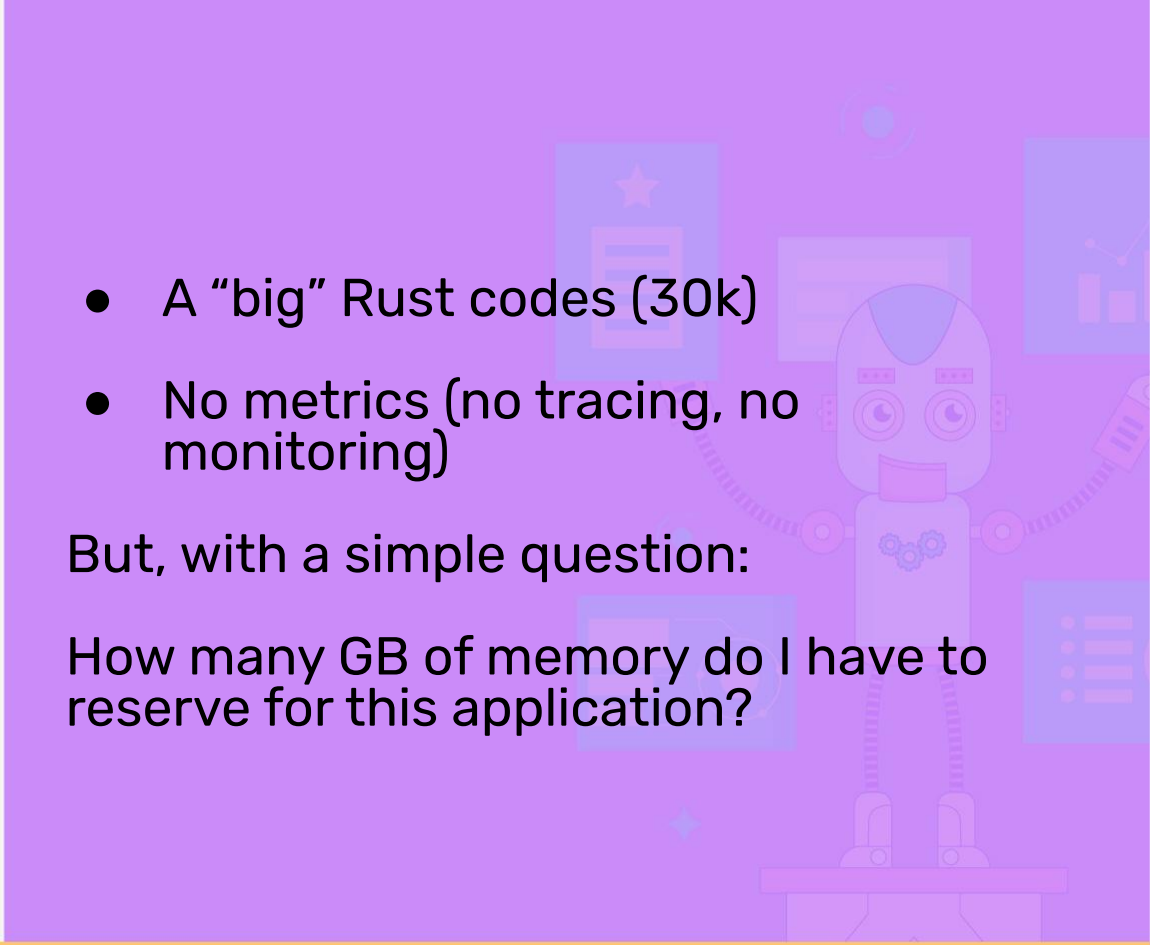


# From where I started

- A “big” Rust codes (30k)
- No metrics (no tracing, no monitoring)

But, with a simple question:

How many GB of memory do I have to reserve for this application?



# The tools I tried to use

...

# Memory management

I understood I have to study.

- There are the stack allocations
- There are the heap allocations
- There are other kinds of allocations

# Stack Allocation

- Created at “runtime”, one per thread.
- Fixed.
- It contains:
  - “local” variables
  - function input
  - function output
- other stuff like call pointer
- 8MB but may vary

# Heap Allocation

- Created at “runtime”, when needed.
- Dynamic
- `malloc`, `realloc`, `valloc`, `calloc`, `reallocf`, `aligned_alloc`
  - man malloc
  - man brk
  - man mmap
- Virtually infinite

# What Rust gives to us

- `Arc` **vs** `Box`
- `Vec` **vs** `arrayvec`
- `String` **vs** `&str` **vs** `ArrayString`
- `clone` **vs** `copy`

## Behind the scene

All allocations are rules by a global allocator. `System` is used by default.

Developers can replace it using a different allocation. Just implement `GlobalAlloc` trait and use `#[global_allocator]` attribute.

# GlobalAlloc trait

```
pub unsafe trait GlobalAlloc {
    // Required methods
    unsafe fn alloc(
           &self,
           layout: Layout
     ) -> *mut u8;

    unsafe fn dealloc(
           &self,
           ptr: *mut u8,
           layout: Layout
     );

     // Other methods
}
```

```
pub struct Layout {
     size: usize,
     align: Alignment,
}

let layout: Layout::for_value(&T);

let size: usize = layout.size();
let align: align = layout.align();
```

## Some custom allocators

- tikv-jemallocator (jemallocator)
- mimalloc
- wee_alloc (WebAssembly)

And many others.

# Live coding

# Rallo

What happen if we obtain the trace for every allocation?

# Simple to use - init

```
use rallo::RalloAllocator;

// This is the maximum length of a frame
const MAX_FRAME_LENGTH: usize = 128;
// Maximum number of allocations to track
const MAX_LOG_COUNT: usize = 1_024 * 10;

#[global_allocator]
static ALLOCATOR: RalloAllocator<
     MAX_FRAME_LENGTH,
     MAX_LOG_COUNT
> = RalloAllocator::new();
```

# Simple to use - track

```
fn foo() {
     let v: Vec<u8> = Vec::with_capacity(20);
}

// Start tracking
unsafe { ALLOCATOR.start_track() };

foo();

// Stop tracking
ALLOCATOR.stop_track();
```

# Simple to use - print

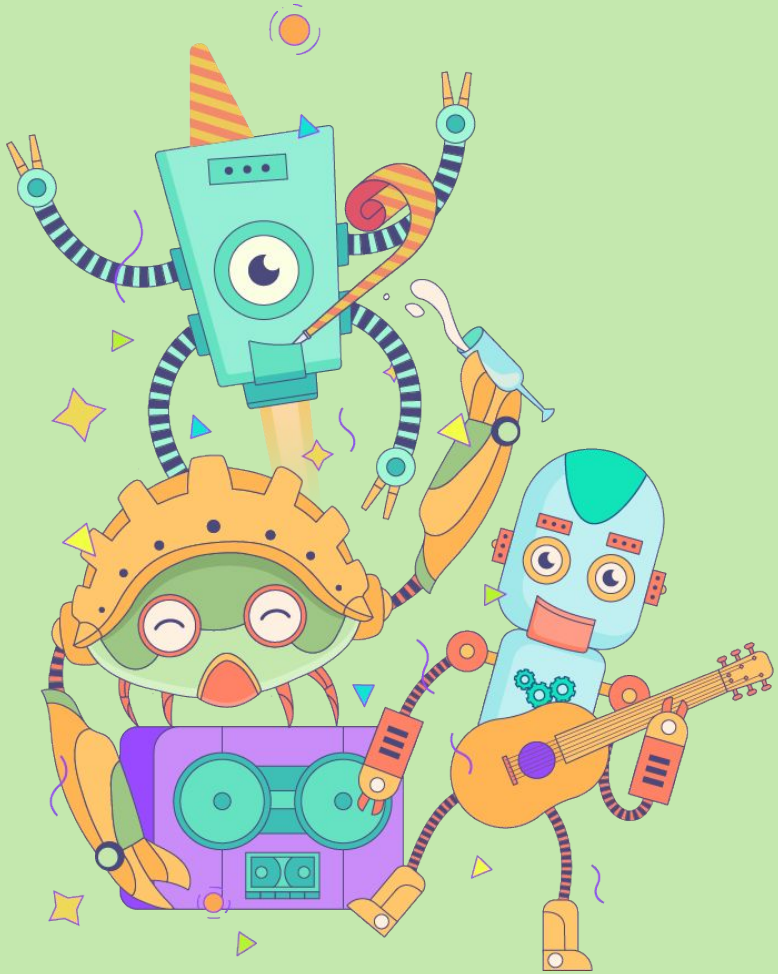
```
let stats = unsafe { ALLOCATOR.calculate_stats() };
let tree = stats.into_tree().unwrap();

let path = std::env::current_dir()
     .unwrap()
     .join("flamegraph.html");
tree.print_flamegraph(&path);
```

# Demo

# Rallo

https://crates.io/crates/rallo

# Greetings and conclusion



# Tommaso Allevi

tomallevi@gmail.com

https://www.linkedin.com/in/tommaso-allevi-a9979045/

@allevo