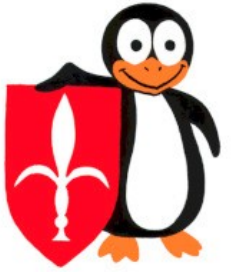
Rust in Linux

Perché sì... e perché no...

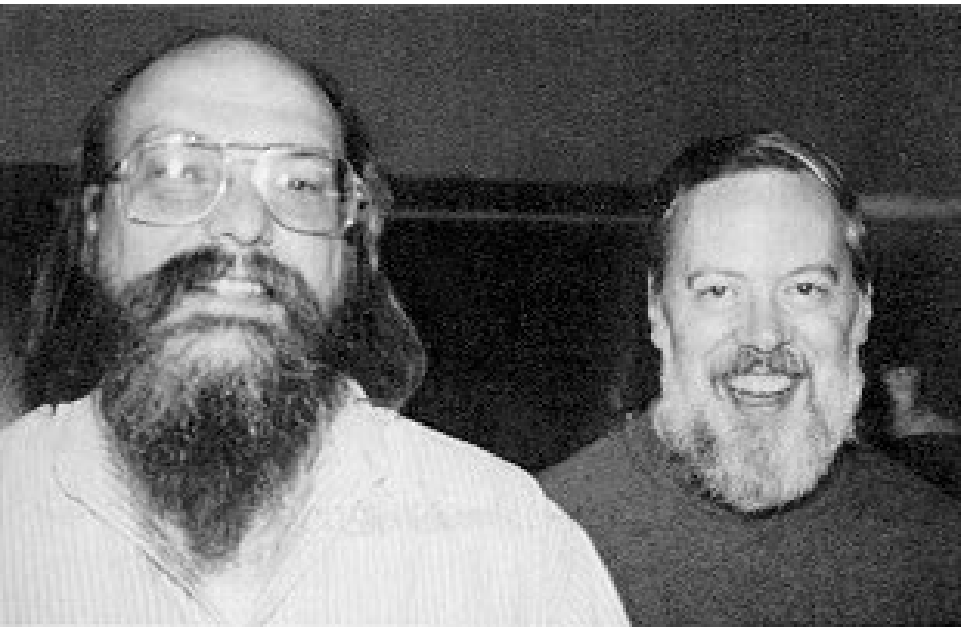
Sonia Zorba

Linux Day 2025



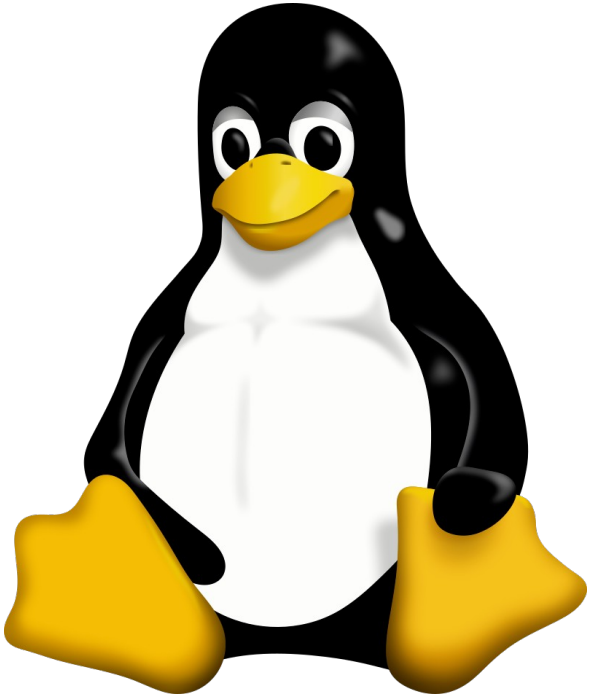
Linguaggio C

Ken Thompson Dennis Ritchie



- Inventato da Dennis Ritchie nel 1972
- Dopo il linguaggio B
- Durante lo sviluppo di Unix

C in GNU/Linux

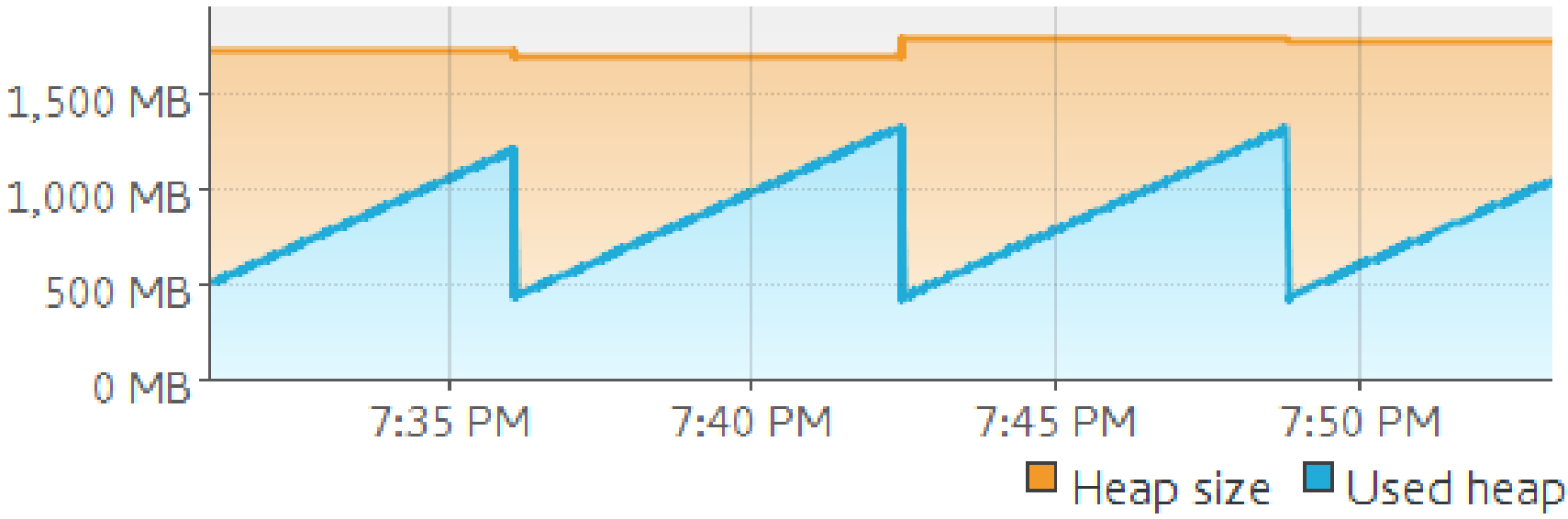


- **Kernel**
- **Coreutils** (ls, mkdir, cp, mv, rm, chmod, ...)
- Svariati altri programmi...

Alcune caratteristiche del C

- Linguaggio compilato
- Gestione manuale della memoria
  - Es: malloc(), free()
  - Assenza di garbage collector

Linguaggi con garbage collector



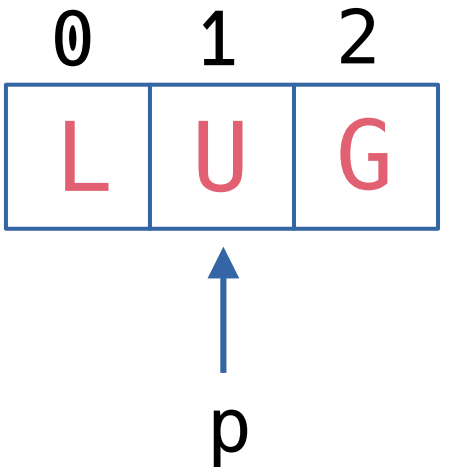
Alcune caratteristiche del C

- Supporta operazioni “a basso livello”, consentendo di gestire i puntatori

```c
char a[3] = {'L', 'U', 'G'};

char *p = a;
p = p + 1;

printf("%c\n", *p);
```



Problema: C non è “memory safe”

```
int main(int argc, char *argv[])
{
  char buffer[5];
  strcpy(buffer, argv[1]);
}
```

Buffer overflow, use-after-free, double free, dangling pointers…

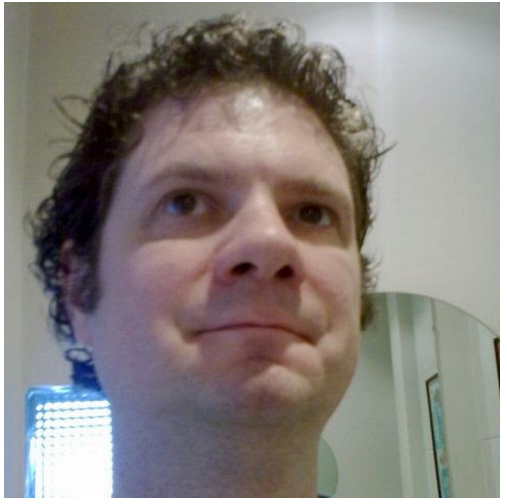
Varie mitigazioni

- Randomizzazione della memoria (ASLR)
- Stack canary
- …

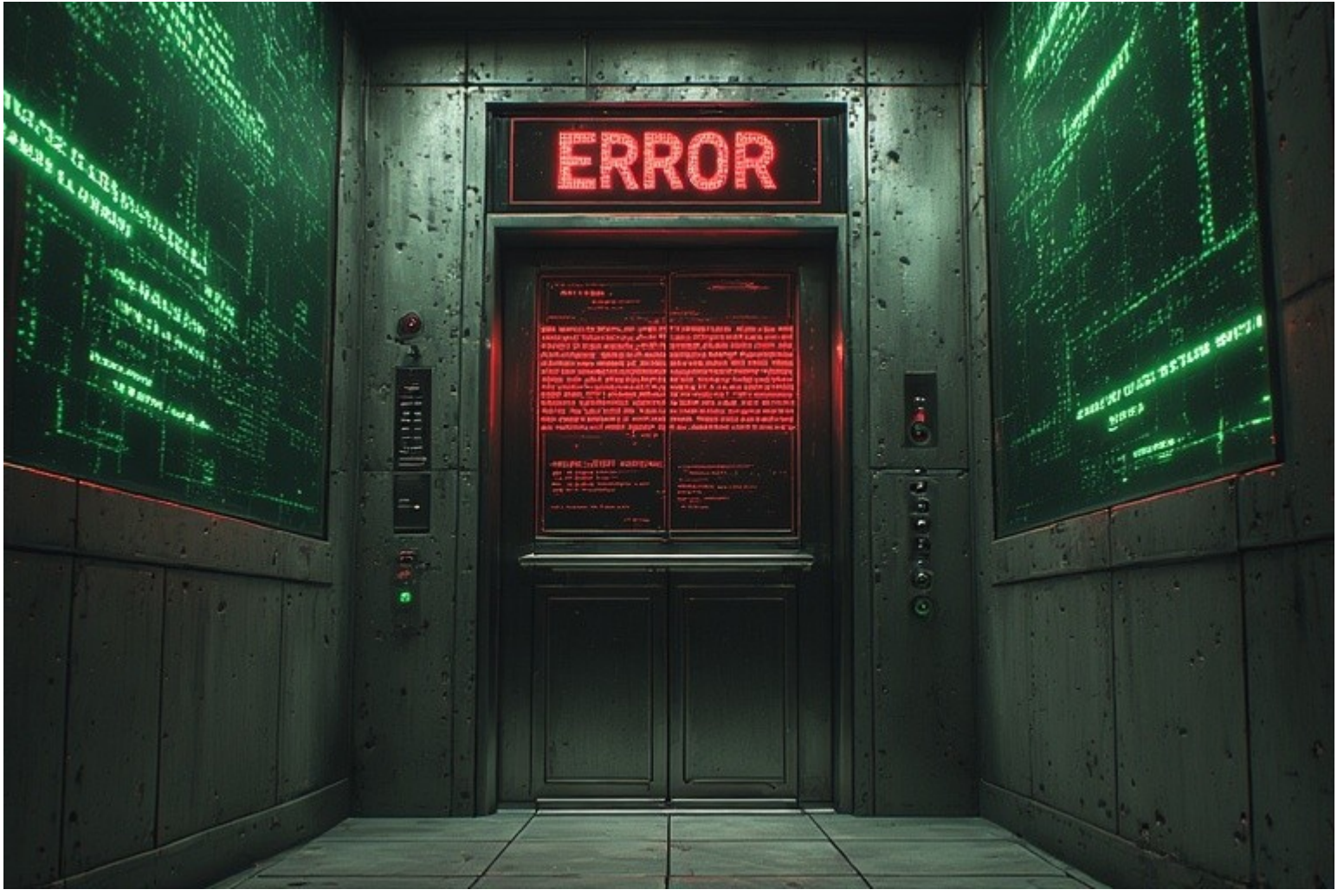
Breve storia del Rust

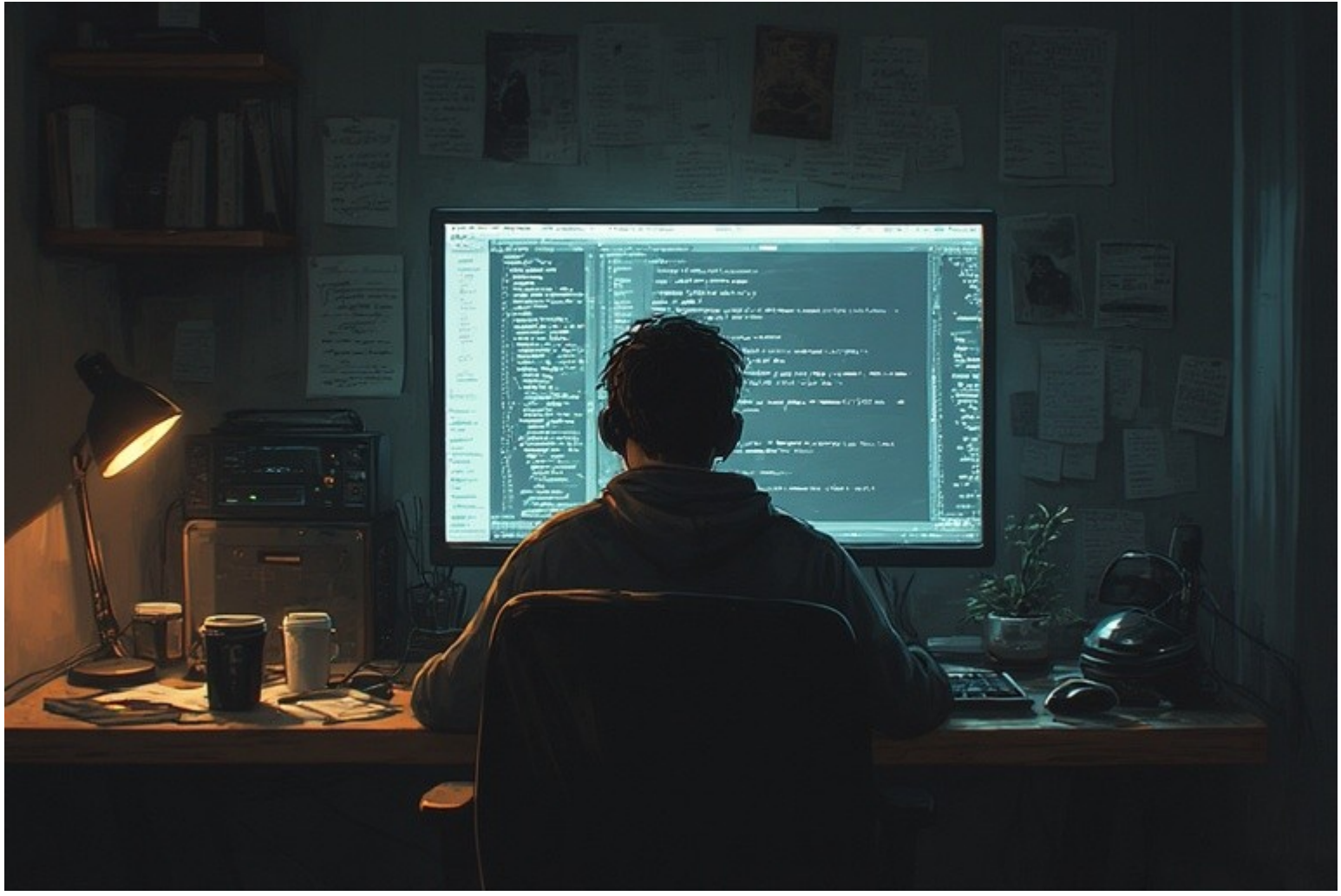
2006

Graydon Hoare



moz://a


ERROR



Rust (fungus)

From Wikipedia, the free encyclopedia

Rusts are fungal plant pathogens of the order **Pucciniales** (previously known as **Uredinales**) causing plant fungal diseases.

An estimated 168 rust genera and approximately 7,000 species, more than half of which belong to the genus *Puccinia*, are currently accepted.[3] Rust fungi are highly specialized plant pathogens with several unique features. Taken as a group, rust fungi are diverse and affect many kinds of plants. However, each species has a range of hosts and cannot be transmitted to non-host plants. In addition, most rust fungi cannot be grown easily in pure culture.

Most species of rust fungi are able to infect two different plant hosts in different stages of their life cycle, and may produce

Rusts

Example of wheat leaf from a disease differential of *Puccinia recondita* f.sp. *tritici*

Scientific classification

Kingdom:	Fungi
Division:	Basidiomycota
Class:	Pucciniomycetes
Order:	**Pucciniales**

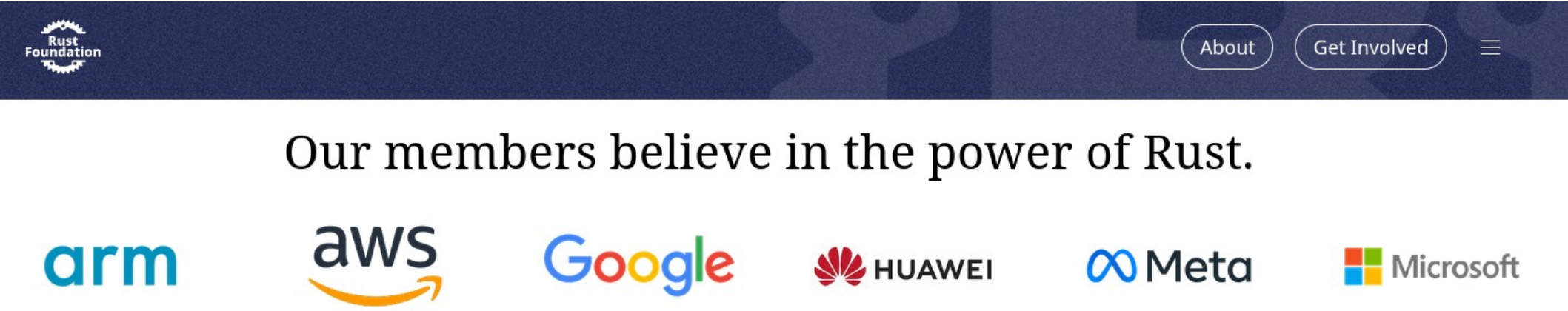
Breve storia del Rust

2009 

2015 Rust v1.0

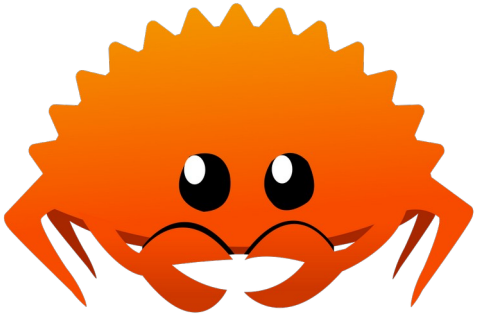
Rust Foundation

- Nata nel 2021



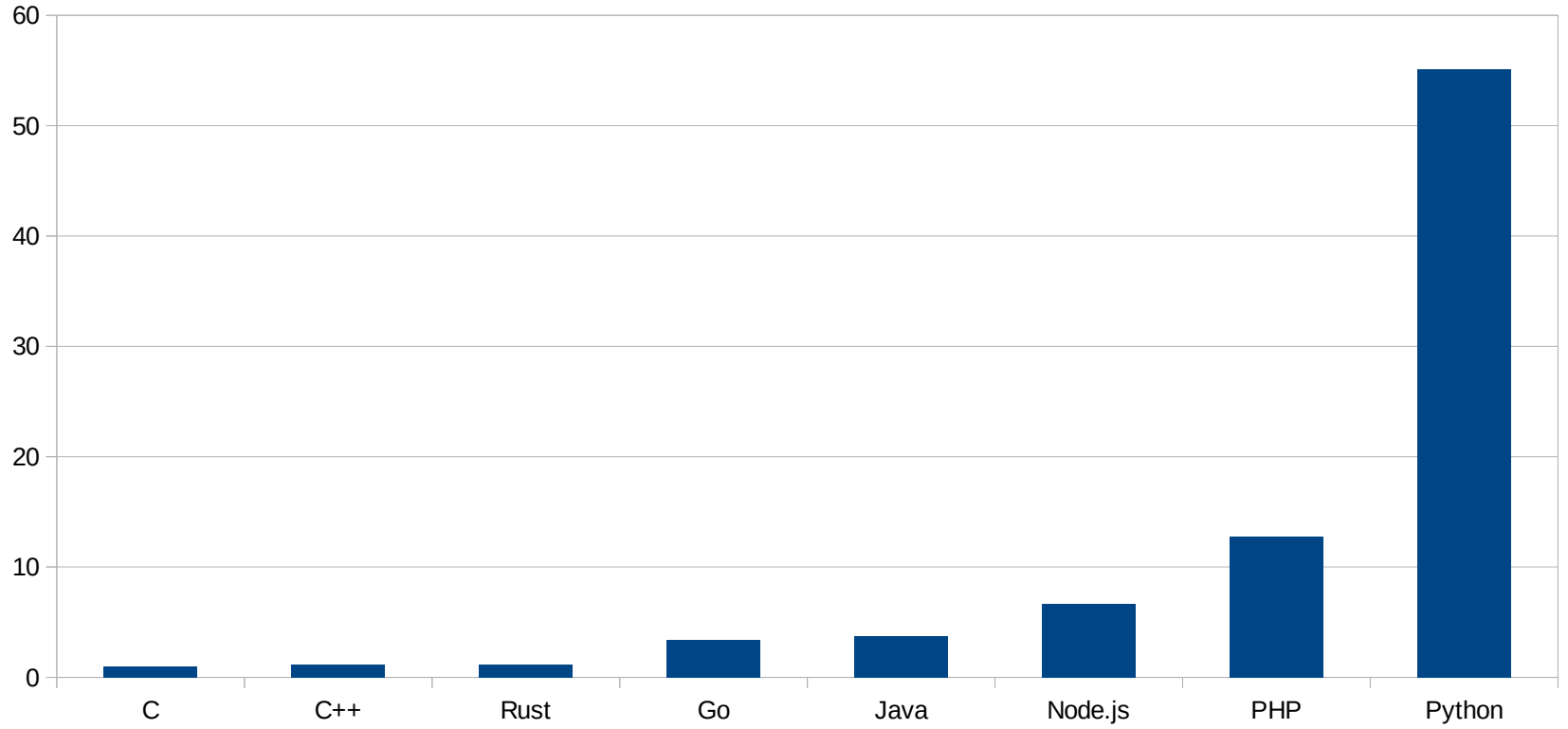
Alcune caratteristiche del Rust

- Linguaggio compilato
- Gestione automatica della memoria MA senza garbage collector!
- Memory safe
- Impedisce data race (non race condition)



Hello, crustaceans.

Benchmarks



Fonte: https://github.com/GoodManWEN/Programming-Language-Benchmarks-Visualization

Ownership

1) Ogni valore in Rust ha un *owner* (proprietario)
2) In un dato momento, un valore può avere un unico proprietario
3) Quando il proprietario va “out of scope” (es: la funzione termina), il valore viene distrutto (dropped)

Come il RAII in C++

Reference

- Simile a un puntatore, ma:
  - solo valori validi;
  - tipo di dato ben definito e non modificabile.
- L'atto di creare una reference si chiama **borrowing** (prendere in prestito)
- Si possono avere *n* reference immutabili
- Si può avere **1 sola reference mutabile**
- Ogni reference ha una **lifetime** (periodo di validità)

Borrow checker

Componente che verifica le regole sulle reference

Un “semplice” esempio

```
fn longest(x: &str, y: &str) -> &str {
    if x.len() > y.len() {
        return x;
    } else {
        return y;
    }
}
```

???

```
error[E0106]: missing lifetime specifier
 --> src/main.rs:5:33
  |
5 | fn longest(x: &str, y: &str) -> &str {
  |               ----     ----     ^ expected named lifetime parameter
  |
  = help: this function's return type contains a borrowed value,
but the signature does not say whether it is borrowed from `x` or `y`

help: consider introducing a named lifetime parameter
  |
5 | fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
  |           ++++     ++          ++          ++
```

Soluzione

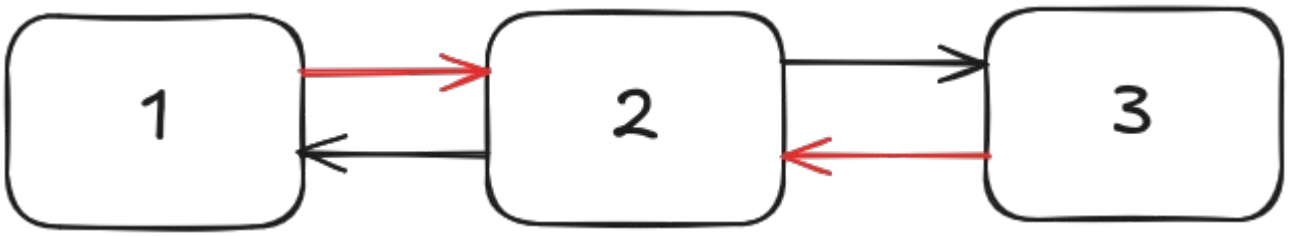
```
fn longest<'a>(x: &'a str, y: &'a str) -> &'a str {
    if x.len() > y.len() {
        return x;
    } else {
        return y;
    }
}
```

Esempio con lifetime non compatibili

```
fn main() {
    let a = String::from("longest");
    let result;
    {
        let b = String::from("xyz");
        result = longest(a.as_str(), b.as_str());
    } // ← b muore qui
    println!("The longest string is {result}");
} // ← a, result muoiono qui
```

Linked List

Doubly Linked List



1. Introduction
2. A Bad Stack
  2.1. Layout
  2.2. New
  2.3. Ownership 101
  2.4. Push
  2.5. Pop
  2.6. Testing
  2.7. Drop
  2.8. Final Code
3. An Ok Stack
  3.1. Option
  3.2. Generic
  3.3. Peek
  3.4. IntoIter
  3.5. Iter
  3.6. IterMut
  3.7. Final Code
4. A Persistent Stack
  4.1. Layout
  4.2. Basics
  4.3. Drop
  4.4. Arc
  4.5. Final Code
5. A Bad Safe Deque

Learn Rust With Entirely Too Many Linked Lists

Got any issues or want to check out all the final code at once? Everything's on Github!

NOTE: The current edition of this book is written against Rust 2018, which was first released with rustc 1.31 (Dec 8, 2018). If your rust toolchain is new enough, the Cargo.toml file that `cargo new` creates should contain the line `edition = "2018"` (or if you're reading this in the far future, perhaps some even larger number!). Using an older toolchain is possible, but unlocks a secret **hardmode**, where you get extra compiler errors that go completely unmentioned in the text of this book. Wow, sounds like fun!

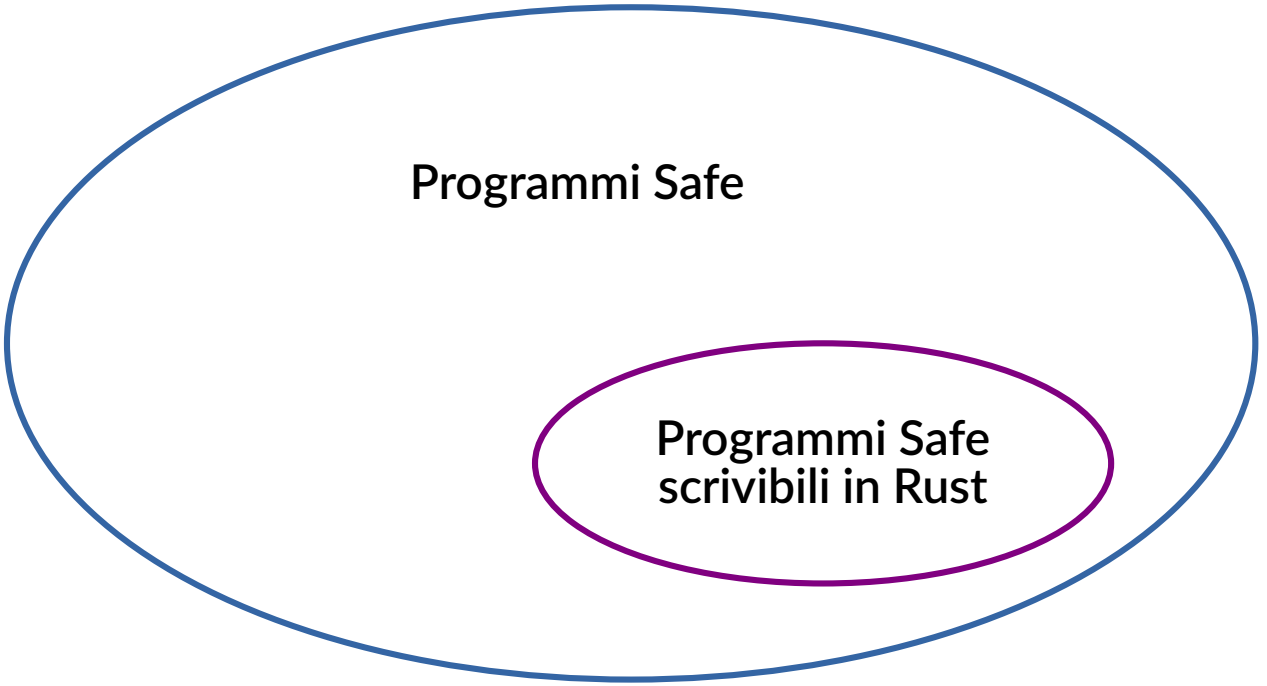
I fairly frequently get asked how to implement a linked list in Rust. The answer honestly depends on what your requirements are, and it's obviously not super easy to answer the question on the spot. As such I've decided to write this book to comprehensively answer the question once and for all.

In this series I will teach you basic and advanced Rust programming entirely by having you implement 6 linked lists. In doing so, you should learn:

- The following pointer types: `&`, `&mut`, `Box`, `Rc`, `Arc`, `*const`, `*mut`, `NonNull`(?)
- Ownership, borrowing, inherited mutability, interior mutability, Copy
- All The Keywords: struct, enum, fn, pub, impl, use, ...
- Pattern matching, generics, destructors
- Testing, installing new toolchains, using `miri`
- Unsafe Rust: raw pointers, aliasing, stacked borrows, UnsafeCell, variance

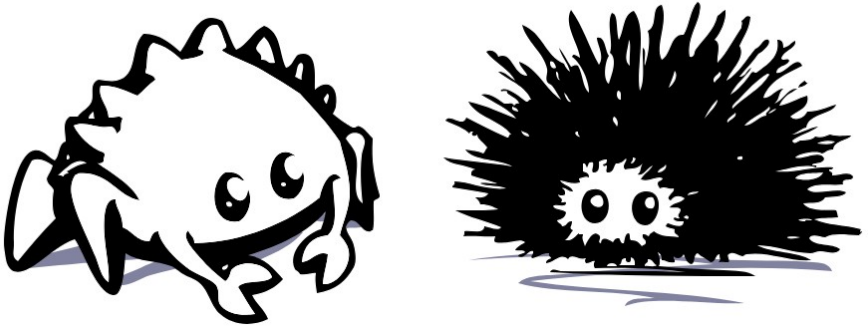
Yes, linked lists are so truly awful that you deal with all of these concepts in making them real.

Sintassi molto vincolante!



Unsafe Rust

```
unsafe {
    // have fun
}
```



- Usi:
  - Chiamare funzioni esterne (es: codice C)
  - Performance
  - Doubly linked list


Rust
C
Go
Java

Why Rust is the most admired language among developers

Rust continues to top the charts as the most admired and desired language by developers, and in this post, we dive a little deeper into how (and why) Rust is stealing the hearts of developers around the world.

Programming, scripting, and markup languages

Rust is the most admired language, more than 80% of developers that use it want to use it again next year. Compare this to the least admired language: MATLAB. Less than 20% of developers who used this language want to use it again next year.

https://survey.stackoverflow.co/2023/#section-admired-and-desired-programming-scripting-and-markup-languages

crates.io

Browse All Crates | Log in with GitHub

The Rust community's crate registry

Type 'S' or '/' to search

Install Cargo

Getting Started

Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.

183,665,342,891
Downloads

200,393
Crates in stock

Crates (pacchetti)

- Si scarica il sorgente da crates.io, che viene compilato e “inglobato” nel binario finale
- Standard library minimale → mi servono crate di terze parti anche per “cose di base”

Rust: Does the published crate match the upstream source?

2021-10-03

This project began the way many of my long Rust articles do-- I got curious about something. I started to wonder a few weeks ago about the relationship between crates that I download from crates.io, and the crate's upstream repository. Here are some of the questions I wanted to answer:

- How do I tell which git commit matches the published crate?
- Is there any guarantee that the published crate source matches the git source?
- What kinds of best practices exist? Is there room for improvement?

https://codeandbitters.com/published-crate-analysis/

Shared libraries?

- Si possono fare, ma non sono così comuni
- Rust non ha un'ABI stabile
  - La libreria non funzionerà se si usa una versione diversa del compilatore
  - A meno di non usare #[repr(C)] (o stabby)
  - Non è un bug, è una feature

Rust non ha uno standard

- Il C è definito da documenti formali
  - C98 (ANSI C), C90 (ISO C), C95…
  - Esistono svariati compilatori (es: gcc, clang, …)
- Il Rust non ha uno standard
  - Pochi compilatori (2?): rustc e mrustc


RISCRIVERE
TUTTO
CODICE
STABILE
imgflip.com

The project

- Contact
- Contributing
- Rust kernel policy
- Branches
- Rust reference drivers
- Rust version policy
- Unstable features
- Backporting and stable/LTS releases
- Third-party crates
- Out-of-tree modules
- Industry and academia support
- Sponsors

Subprojects

- `klint`
- `pin-init`

Tools

- Coccinelle for Rust
- `rustc_codegen_gcc`
- `gccrs`

Rust for Linux

Rust for Linux is the project adding support for the Rust language to the Linux kernel.

This website is intended as a hub of links, documentation and resources related to the project.

The project

- Contact
- Contributing
- Rust kernel policy
- Branches
- Rust reference drivers
- Rust version policy
- Unstable features
- Backporting and stable/LTS releases

The Case for Memory Safe Roadmaps

Why Both C-Suite Executives and Technical Experts Need to Take Memory Safe Coding Seriously

Publish Date: December 06, 2023

RELATED TOPICS: CYBERSECURITY BEST PRACTICES

The Case for Memory Safe Roadmaps: Why Both C-Suite Executives and Technical Experts Need to Take Memory Safe Coding Seriously details how software manufacturers can transition to memory safe programming languages (MSLs) to eliminate memory safety vulnerabilities. The guidance provides manufacturers steps for creating and publishing memory safe roadmaps that will show their customers how they are owning security outcomes, embracing radical transparency, and taking a top-down approach to developing secure products—key Secure by Design tenets.

Rust nel kernel

- Si tratta di **gestire 2 diversi linguaggi** → impatto sulla mantenibilità
- Nel 1997 tentarono di aggiungere il C++… per 2 settimane
- Al momento fornisce solo delle “safe astraction” (wrapper) per il codice C sottostante
- In futuro…

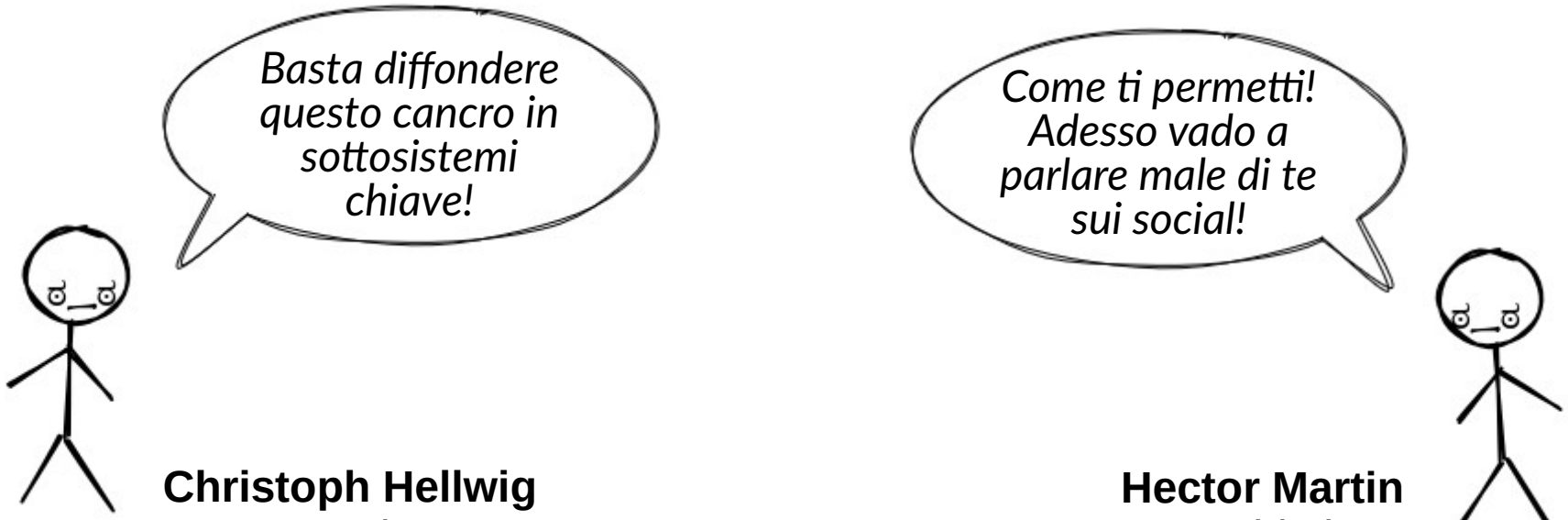
Rust nel kernel

- Creare queste “safe abstraction” interdipendenza tra codice C e codice Rust
- Problema: alcuni programmatori C non conoscono il Rust e viceversa
- Modifiche al C possono rompere il codice Rust
- La PR non viene mergiata finché tutto non compila

Flame

Christoph Hellwig
DMA Mapping

Hector Martin
Asahi Linux



Christoph Hellwig
DMA Mapping

Hector Martin
Asahi Linux

https://lkml.org/lkml/2025/1/28/300

Christoph Hellwig
DMA Mapping

Hector Martin
Asahi Linux

Christoph Hellwig Si Dimette da Maintainer dei DMA Mapping Helpers nel Kernel Linux!

DI ALEX · 26 FEBBRAIO 2025

Sipario, dopo Rust nel Kernel Linux, Hector Martin abbandona anche la posizione di project lead per Asahi Linux

Patrick Di Fazio 18 Febbraio 2025

Ubuntu 25.10, release transitoria, comincia a presentarsi con sudo e coreutils scritti in Rust

Raoul Scarazzini 9 Settembre 2025

Coreutils in Rust

- https://github.com/uutils/coreutils
- Talk del Fosdem di Sylvestre Ledru: “**Reimplementing the Coreutils in a modern language (Rust)**” https://archive.fosdem.org/2023/schedule/event/rust_coreutils/

Licenze...

- GNU Coreutils: **GPLv3**
  - Se modifichi il codice devi distribuire le modifiche con la stessa licenza
- Rust Coreutils: **MIT**
  - Puoi modificare il codice e ridistribuire le modifiche con una licenza proprietaria

The most notable thing about this project is it allows proprietarization while GNU coreutils doesnt #1781

Closed

ian-kelling opened on Mar 9, 2021

You are recreating copyleft programs with a non-copyleft license. It is not unethical but it likely will be harmful to user's freedoms. It allows someone to incorporate it into a device or distribution of software, and not give them the source code and a means to install new versions so that they have the freedom to run, copy, distribute, study, change and improve the software. If you plan to carry on, please show some respect to GNU coreutils and the free software movement by adding a note in your README which says that you're using a different license than GNU Coreutils, and GNU Coreutils does not agree with your license, for reasons such as these: https://www.gnu.org/philosophy/why-copyleft.en.html

👍 7 👎 64 😄 3 😕 6 ❤️ 1

Ubuntu abbandona GNU/Linux: rivoluzione agghiacciante e affascinante allo stesso tempo

Ubuntu ha annunciato l'intenzione di integrare le implementazioni in Rust delle core utilities di sistema a partire dalla release 25.10, sostituendo progressivamente le GNU Core Utilities con U-utils. Vediamo nel dettaglio cosa significa questa mossa per gli utenti, le opportunità e i rischi.

Michele Nasi
Pubblicato il 21 mar 2025

Rust & MIT

- La toolchain di Rust è in MIT
- Nelle linee guida suggeriscono di rilasciare i pacchetti in MIT

» **Crate and its dependencies have a permissive license (C-PERMISSIVE)**

The software produced by the Rust project is dual-licensed, under either the MIT or Apache 2.0 licenses. Crates that simply need the maximum compatibility with the Rust ecosystem are recommended to do the same, in the manner described herein. Other options are described below.

Alternative (in C)

- Static analyzer
- Librerie più furbe per le stringhe https://github.com/antirez/sds
- FilC, compilatore che rende il C safe (1.5x più lento) https://fil-c.org/
- Macro per creare costrutti safe. Talk Fosdem 2025 “Imposing memory security in C”

Alternative (altri linguaggi)

- Zig (2016)
- Odin (2016)
- Carbon (2022)
- Ada (1980)

Unix-like microkernel-based operating system written in Rust

Domande?