



09.06.26
09h à 19h



4 place Jussieu
75005 Paris

Rust Paris 2026



Langage de programmation
sûr et performant

En partenariat avec



Mot d'accueil

Lætitia Jeanne

Référente communication - Hub Open Source

Systematic Paris-Region

3^{ème} édition | Sorbonne-Université, Campus de Jussieu

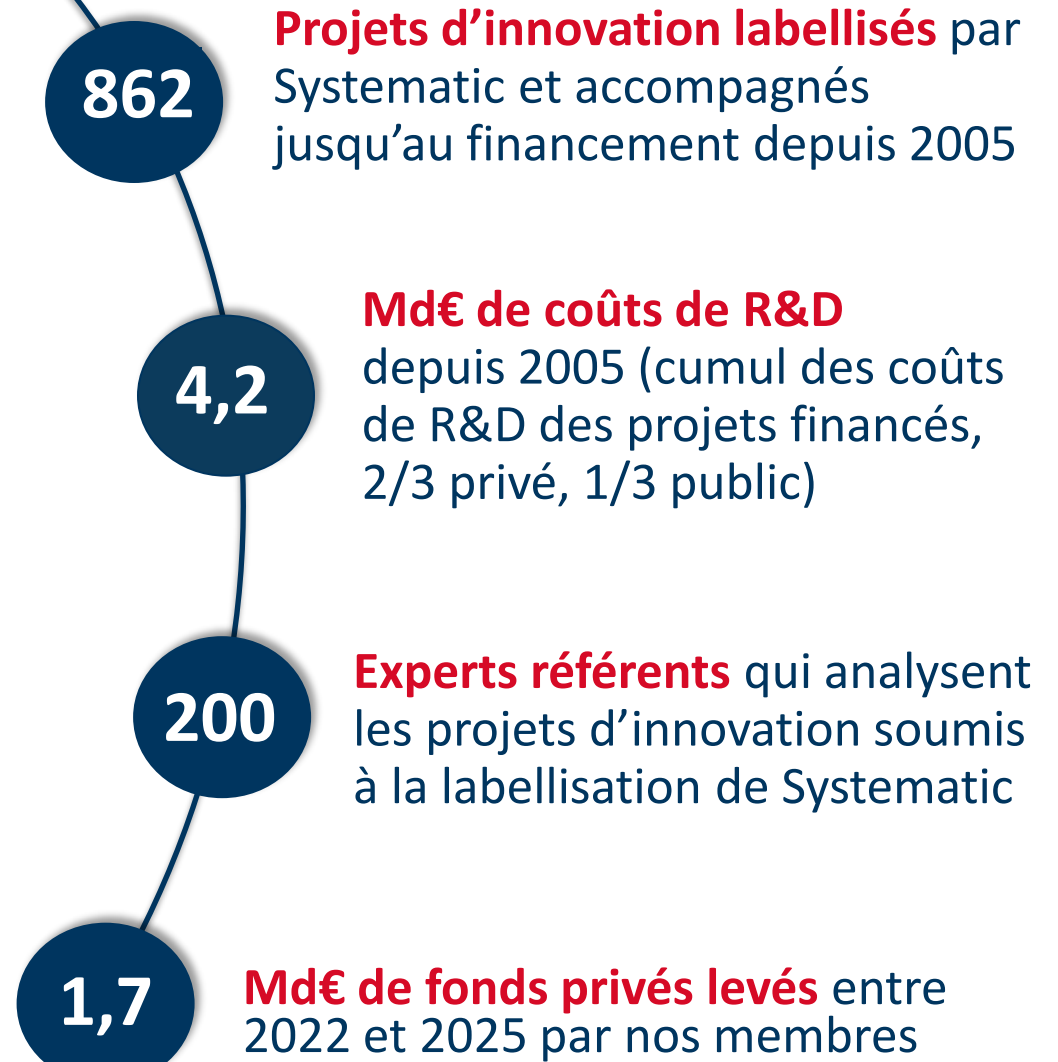
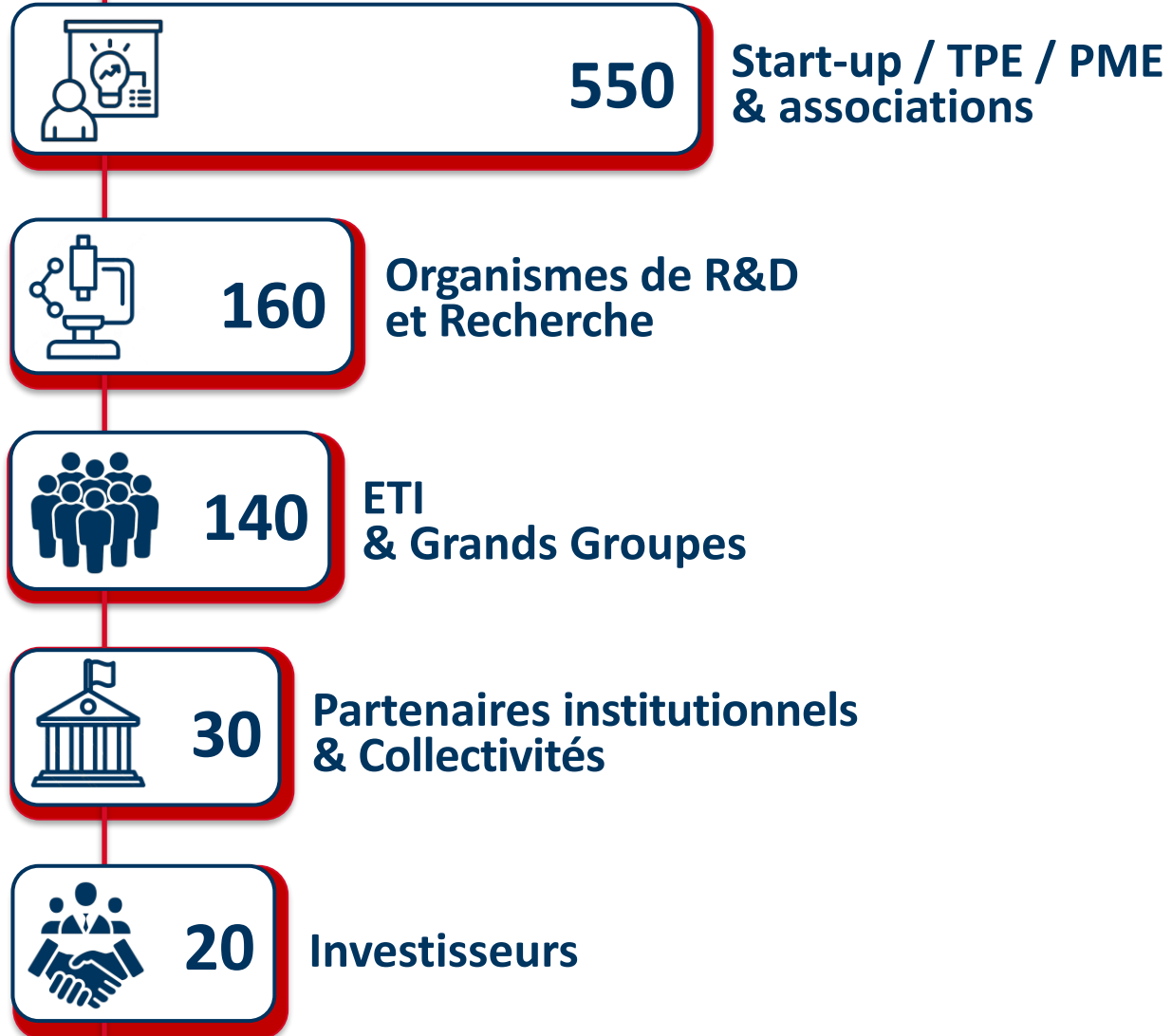
- Une journée de conférences sur l'utilisation de **RUST** dans des projets industriels et de recherche.
- **Partager** des idées et apprendre **ensemble**
- Organisé par **Systematic Paris-Region** en partenariat avec **Sorbonne Université**



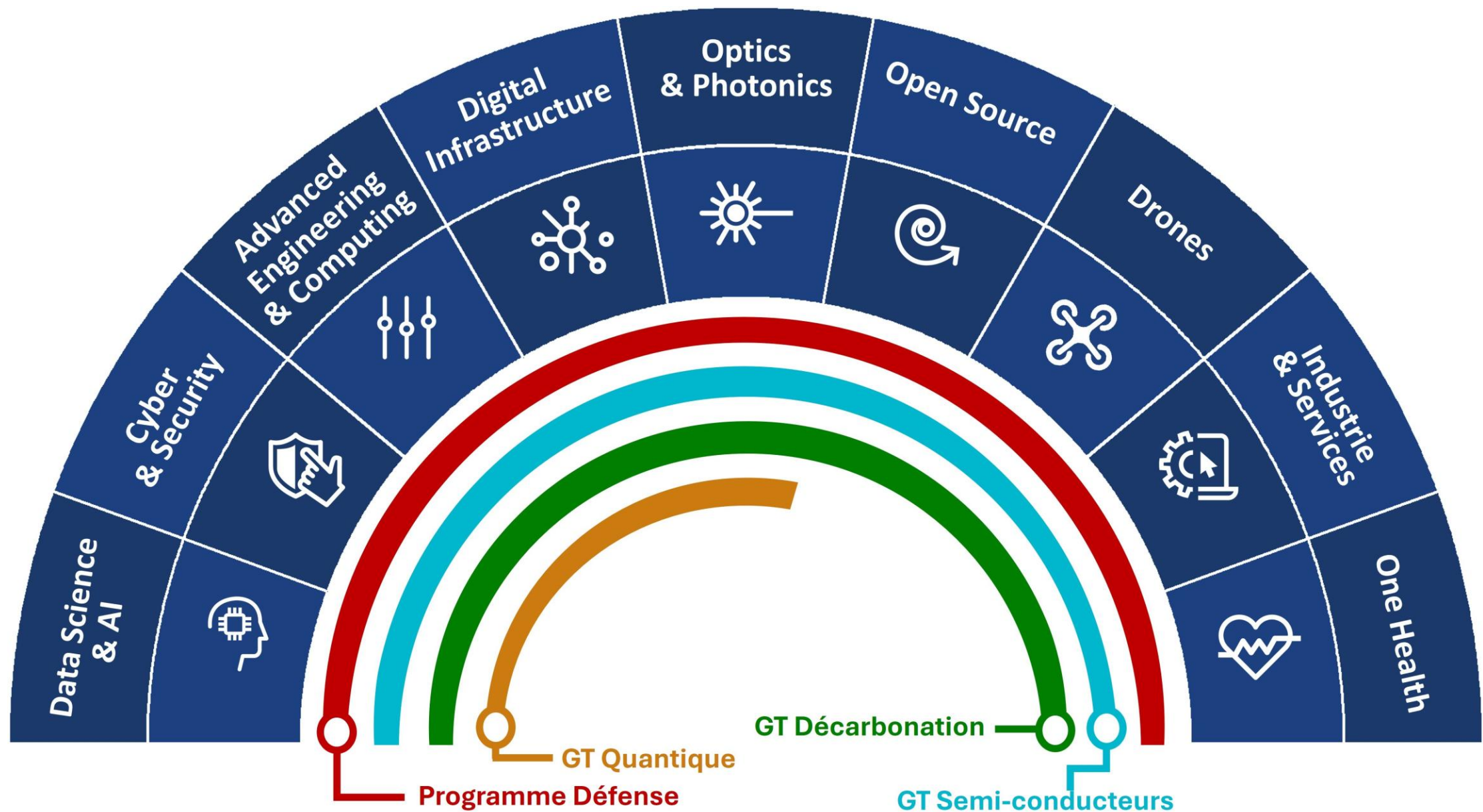
Le Pôle européen des Deep Tech

au service d'une société
responsable & souveraine

Un écosystème Deep Tech de 900 membres



La roadmap de Systematic



Le Hub Open Source



Objectifs du Hub Open Source :

- Promouvoir la filière professionnelle de l'open source,
- Accompagner les organisations sur leur approche et passage en open source,
- Soutenir le développement des membres sur les volets R&D, compétences, business, financement,
- Favoriser l'émergence de socles et briques technologiques open source et standards, et le développement de solutions souveraines



18 ans d'existence



150 membres :
PME, ETI,
Grands groupes,
académiques

Membres du Hub Open Source



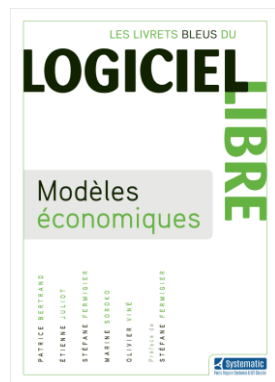
Open Source Experience 2026



Livret bleu open source pour l'IoT



Livret Bleu du Logiciel Libre : Les modèles économiques



Atelier sur les enjeux juridiques de l'open source



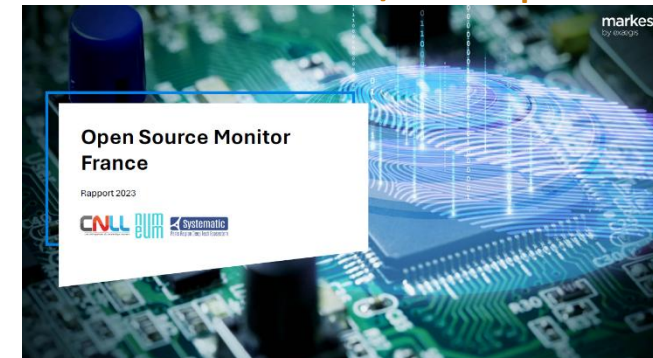
Cycle de conférences « Open Source Business Lab »



Sécurité de la supply chain des logiciels open source



Étude annuelle marché/filière Open Source



Ateliers Alternatives open source à VMware



PyParis



Merci à notre **partenaire** et à nos **sponsors** :



Sponsors Gold

CÉNOTÉLIE

TITAGONE
by OCaml PRO

THALES
Building a future we can all trust

Sponsors Silver

AdaCore

oxid[os]

Wylidrin

Rust Paris 2026

Notre partenaire



Emmanuel Chailloux

Professeur émérite

LIP6, IRILL, Sorbonne Université

Membre du comité de pilotage du Hub Open Source de Systematic



Rust Paris 2026

Sponsor Gold

CÉNOTÉLIE

Laurent Wouters

CTO

Cénotélie

Membre du comité de pilotage du Hub Open Source de Systematic



Systematic

Paris Region Deep Tech Ecosystem



CÉNÔTÉLIE

FORMATION, CERTIFICATION ET SERVICES
TECHNIQUES AUTOUR DU LANGAGE RUST

VOTRE PARTENAIRE EXPERT RUST EN FRANCE



Services

Techniques Rust

CÉNÔTÉLIE

Audit

Code et architecture logicielle

Optimisation

Performances et composants

Migration

Intégration C/C++/Python

Prototypage

Rapide et conseils stratégiques

Certification Rust en France

The background of the slide features a dimly lit office environment. In the foreground, several people are seated at long desks, working on computers. The desks are equipped with multiple monitors, some displaying code. The office has a clean, professional look with blue ambient lighting. In the background, a person is standing and gesturing, possibly presenting or explaining something to the team.

Seul centre habilité à
certifier sur Rust en
France



Formations :
Introduction, Avancé,
Embarqué



Programmes
personnalisables



Accompagnement stratégique
de la montée en compétence
de vos équipes

Contactez-nous



+33 6 73 83 21 18



contact@cenotelie.fr



cenotelie.fr



@cenotelie

"Transformez vos équipes avec l'expertise Rust de Cenotelie"

Rust Paris 2026

Sponsor Gold

THALES

Building a future we can all trust

Sébastien Lejeune

OSPO Director, Driving Open Source Collaboration

Thales

Membre du comité de pilotage du Hub Open Source de Systematic



Rust Paris 2026

THALES
Sebastien LEJEUNE
Directeur OSPO Thales

www.thalesgroup.com



Thales en quelques chiffres



85 000
collaborateurs



65 pays
Une présence
mondiale



4,5 Mds €^{*}
*+1 Md € autofinancée



22,1 Mds €
de chiffre
d'affaires





RUST @Thales

Vous voulez en savoir plus ?

Contactez nous
oss@thalesgroup.com

www.thalesgroup.com

in

f

X



Rust Paris 2026

Sponsor Gold

TITAGONE

by OCaml **PRO**

Arthur Carcano

Ingénieur R&D Rust, Docteur en mathématiques
et informatique

OcamlPro

Membre du comité de pilotage du Hub Open Source de Systematic



Merci au comité de programme :



Florian BENNETOT
Parsec



Florence BLAZY
Systematic



Arthur CARCANO
Ocamlpro



Emmanuel CHAILLOUX
Sorbonne Université



Thierry LEBLOND
Parsec



Fabrice LE FESSANT
Ocamlpro



Sébastien LEJEUNE
Thales



Orson PESSIN
Wylidrin



Laurent WOUTERS
Cénotélie



Rust Paris 2026

Sébastien Lejeune

OSPO Director,

Driving Open Source Collaboration

Thales

Membre du comité de pilotage du Hub Open Source de Systematic



Sponsors Gold

CÉNOTÉLIE

TITAGONE
by OCaml PRO

THALES
Building a future we can all trust

Sponsors Silver

AdaCore

oxid[os]

Wyliodrin

Agenda – Amphithéâtre 43

Matin

- **09h45 – 10h30 : Keynote** : Petite histoire de Rust hors des industries de la Tech
Frédéric Ameye, Head of Software Architecture | **Renesas Electronics**
- **10h35 – 11h05** : Sécuriser du code Rust en pratique : fuzzing, IA et retours terrain
Patrick Ventuzelo, CEO & Founder | **FuzzingLabs**
- **11h25 – 11h55** : Vérifier formellement des programmes Rust avec Creusot
Li-yao Xia, Ingénieur R&D | **LMF – Inria**
Jacques-Henri Jourdan, Chargé de recherche | **LMF – CNRS**
- **12h00 – 12h30** : Quand Python ne suffit plus : Transférer des Petaoctets de manière efficace et sûre
Florian Lemaitre, HPC expert and Cloud architect | **Aneo**
Dylan Brasseur, HPC expert | **Aneo**



Agenda – Amphithéâtre 43

Après-midi

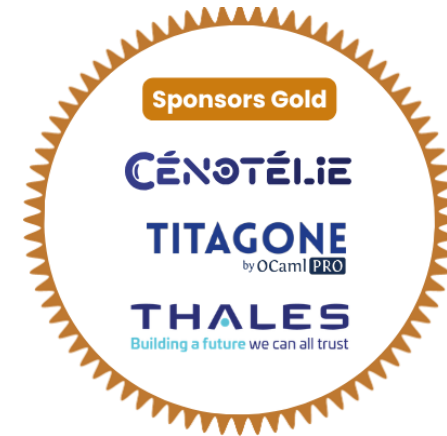
- **14h00 – 14h30** : Architecture à composants : coup de neuf avec Rust
Thomas Clemenceau, Ingénieur logiciel embarqué et critique | **Thales**
Thomas Emerdjian, Ingénieur Développeur logiciel embarqué | **Thales**
- **14h35 – 15h05** : Notre Load Balancer Rust en production : le Bon, la Brute, et l'Async
Florian Lemaitre, HPC expert and Cloud architect | **Aneo**
Jérôme Gurhem, HPC expert and Cloud architect | **Aneo**
- **15h10 – 15h40** : Optimisation des systèmes logiciels complexes via une migration vers Rust et WebAssembly
Gabin Fourcalt, Ingénieur Architecte des systèmes et des logiciels | **Thales Alenia Space**
- **16h00 – 16h30** : Projet de recherche Fos-R:
Pierre-François Gimenez, Chercheur | **Inria**
- **16h35 – 17h05** : What's left to find in Rust code?
Rolland Dudemaine, Directeur Solutions Engineering EMEA/APAC | **TrustInSoft**



Agenda – Amphithéâtre 43

Pauses

- 11h05 – 11h25 : Pause
- 12h30 – 14h00 : Cocktail déjeunatoire
- 15h40 – 16h00 : Pause
- 17h10 – 19h00 : Cocktail networking



Frédéric Ameye



Head of Software Architecture
Renesas Electronics



Petite histoire de Rust hors des industries de la Tech



Joining the Rust Foundation
as Silver member!

Who is this guy?

« This software geek » since I'm 14 years old

Career:

- Lots of volunteering in SW community
- French foreign intelligence agency
- Brain-implanted hearing aid products
- SW architecture & security @ Renault Group (Ampere)
- *Now:* Head of SW architecture @ Renesas



Where it started

„Oh, that's cool!“

„I'm actually *less afraid* of shipping this embedded product that needs to run 24/7 abroad“

The Renault case

**We had a once-in-a-lifetime opportunity for auto industry
to shift into real SW development...**

C++ or Rust ?

We chose Rust.

All new vehicles starting 2026 from Renault Group will use Rust for >60% of OEM components, including ADAS.

The resulting Google AAOS SDV stack massively uses Rust, and makes Rust the de-facto choice.

Beyond Infotainment: Extending Android Automotive OS for Software-defined Vehicles

Posted by Eser Erdem, Senior Engineering Manager, Android Automotive



> Open Source in June 2026!

Why?

**When you maintain a vehicle for 15 years after production,
you're happy to reduce by 70% the risk of vulnerabilities.**

**When you maintain a heterogeneous team of >200 developers,
you're happy to use contract-based interfaces,
strongly enforced at compile-time.**

**When you need to compete with Chinese carmakers,
you're happy to improve your time to market by 1.4x to 3x!**

Why? (cont'd)

Vehicles are complex systems-of-systems, and security and safety regulations require a careful design of interfaces.

Strong typing: helps developers collaborate even when a large organization is broken and documentation is moving.

Safe concurrency: helps developers move from single-core to multi-threaded without shooting (too much) in their own foot.

Memory safety: helps attackers not propagate through vehicle architecture.

How?

**It was not an isolated project, not a rewrite,
but a brand new platform, created from scratch.**

**Obviously it re-used existing C (Matlab-generated) and C++ code,
mostly as libraries. But the backbone was new.**

**This is the easiest path for deploying Rust.
(Or, in general, a new technology)**

Can't we do the same with C++ ?

Even with
manual

The re

AUTOSAR_RS_CPP14Guidelines.pdf
Page 1 sur 397

AUTOSAR
Guidelines for the use of the C++14 language in critical and safety-related systems
AUTOSAR AP Release 17-10

Document Title	Guidelines for the use of the C++14 language in critical and safety-related systems
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	839

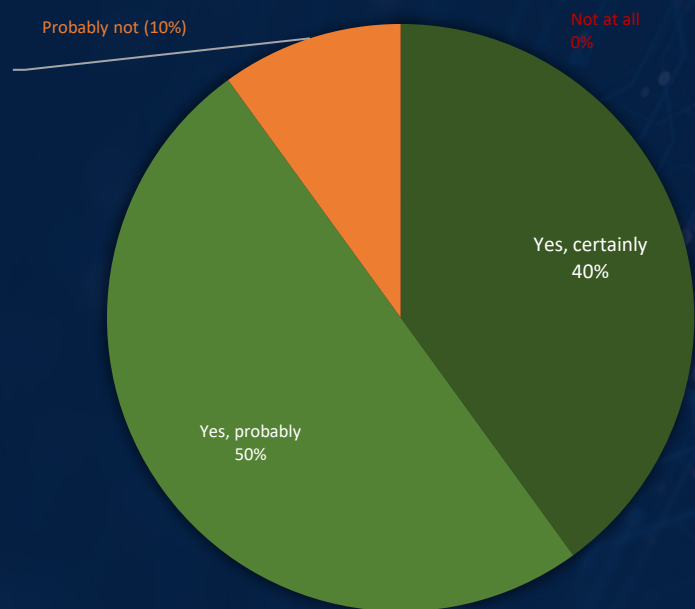
Document Status	Final
Part of AUTOSAR Standard	Adaptive Platform
Part of Standard Release	17-10

Document Change History			
Date	Release	Changed by	Description
2017-10-27	17-10	AUTOSAR Release Management	• Updated traceability for HIC, CERT, C++ Core Guideline • Partially included MISRA review of the 2017-03 release • Changes and fixes for existing rules, more details in the Changelog (C.1)

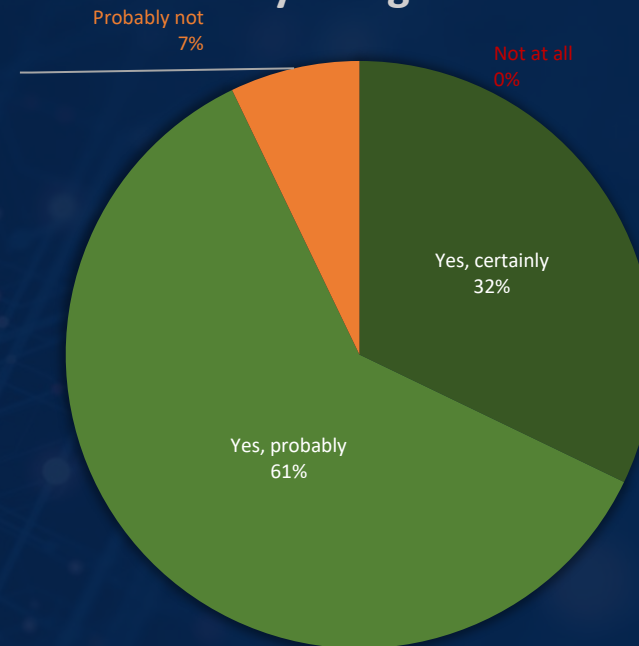


After 2 years...

Do you think Ampere would benefit to have more SW developed with Rust?



Do you think vehicle safety/security is improved by using Rust?



„I’m not a developer, just doing system testing, but with Rust I can tweak / investigate *without fear of crashing everything!*“

What about chipmakers?

Side-note: btw, who is Renesas?

#3 in microcontroller market (ex-Hitachi/Mitsubishi/NEC)

22 000 employees

8 Bn\$ revenue

Billions of chips everywhere, 1100M+ units in cars

The industry has invested massively to fight memory-safety vulnerabilities

- **Hardening C/C++ code**
(stack canaries, ASLR, CFI, code peer review)
- **Hardening microarchitecture** (+ performance overheads...)
(NX bit, PAC, PAN, BTI, MTE, MPX, SGX)
- Infinitely long **C/C++ rules** (eg. MISRA, CERT-C)
- A whole industry of expensive **QA toolchains** (SAST, DAST)
- **Isolation** (proxies, microkernels, VMs, sandboxing, micro-VMs,)

**But despite those efforts,
the never-ending flow of vulnerabilities continues.**

And, again

vulnerabilities

regular bugs

**memory-safety vulnerabilities
are only the
visible tip of the iceberg**

Two big areas where Rust is *killer*

Machine-enforce
a contract
between HAL
developers and
end-users

- *APIs documented as PDFs to APIs enforced by compiler*
- While usually possible with C HALs, much more can be enforced with Rust, in a more idiomatic way.
- Reduce misunderstanding between customers (=users) and developers.



Improve internal
efficiency *at
scale*

- Large organization = high overhead of inter-team interfaces
- Reduce likelihood of bugs/vulnerabilities in low-level critical HW (which might be safety qualified as well)



cognitive load



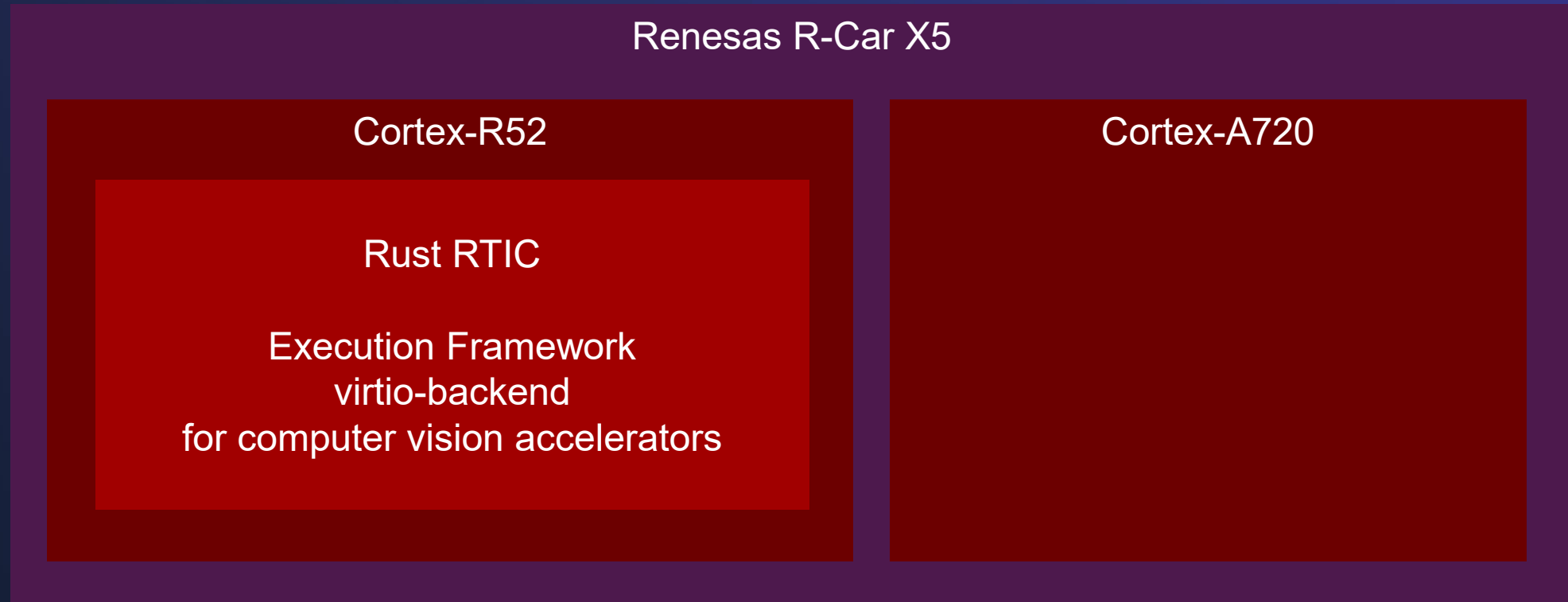
what if you could have less of it?

WHY DOES IT MATTERS TO RENESAS?

- For customer-facing code (HALs, Drivers)
 - Provide users with **state-of-the-art ecosystem** (Embedded-HAL)
 - Provide users with **state-of-the-art debug & profiling tooling** (Probe-rs and de fmt)
- For all embedded development
 - **Where security is key**: bootloaders, code exposed to untrusted inputs, mixed-criticality systems
 - Where features are complex (or concurrent) but environment is constrained or safety-regulated

PILOT PROJECT

- A production-scale pilot project is already ongoing on Renesas R-Car X5H Cortex-R52 cores, in a functional safety context (ISO 26262 ASIL D)



PILOT PROJECT

- This is actually a pilot project.
- While Rust is not a magic (sometimes help), the concurrency is not solved.

Software written in Rust starterpack

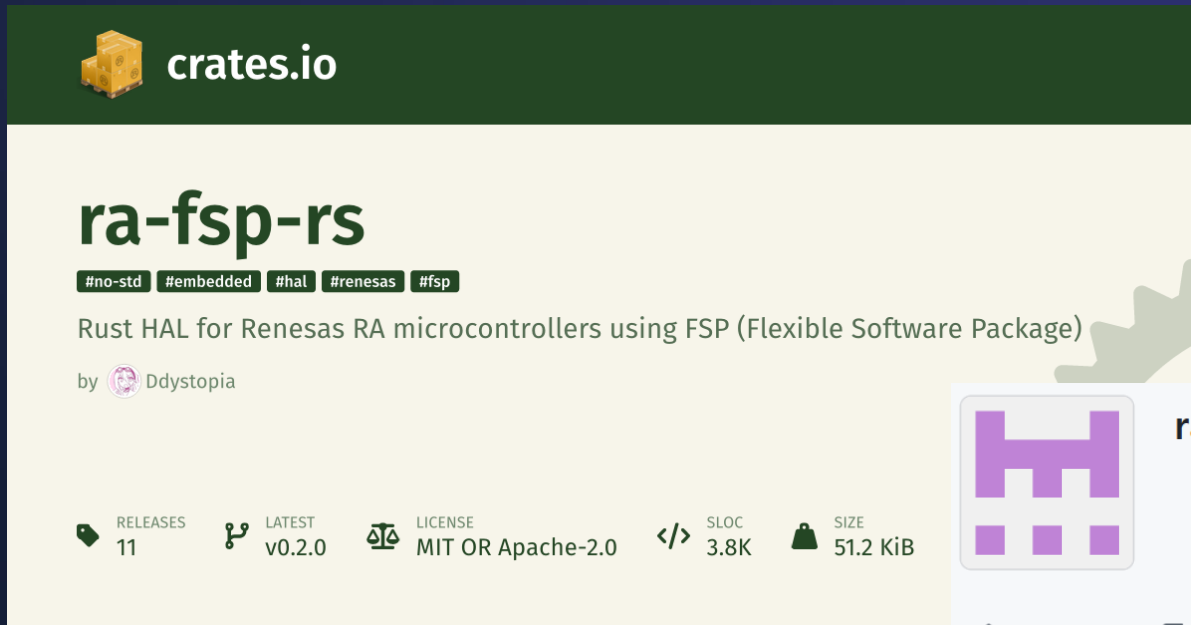
README.md

"blazingly fast..."



WHY DOES IT MATTERS TO RENESAS?

Some volunteers actually already make it happen...



The screenshot shows the crates.io website with a dark green header. The main content area is light yellow and features the crate name 'ra-fsp-rs' in large green letters. Below the name are tags: #no-std, #embedded, #hal, #renesas, and #fsp. A description reads 'Rust HAL for Renesas RA microcontrollers using FSP (Flexible Software Package)'. The author is listed as 'by Ddystopia'. At the bottom, statistics are shown: 11 releases, latest version v0.2.0, MIT OR Apache-2.0 license, 3.8K SLOC, and 51.2 KiB size.

crates.io

ra-fsp-rs

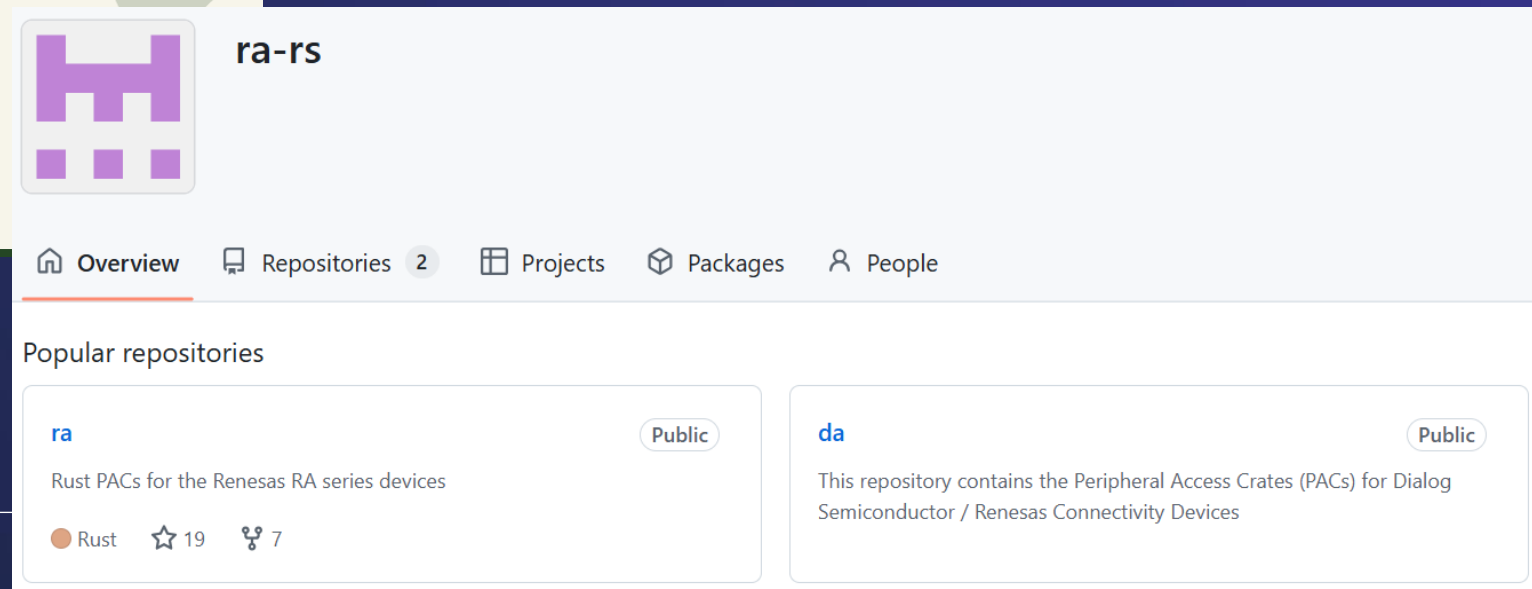
#no-std #embedded #hal #renesas #fsp

Rust HAL for Renesas RA microcontrollers using FSP (Flexible Software Package)

by Ddystopia

RELEASES 11 LATEST v0.2.0 LICENSE MIT OR Apache-2.0 SLOC 3.8K SIZE 51.2 KiB

Thank you!



The screenshot shows the 'ra-rs' repository page on crates.io. It has a light blue header with the repository name and a purple icon. Below the header is a navigation bar with links for Overview, Repositories (2), Projects, Packages, and People. The 'Overview' tab is selected. The main content area is titled 'Popular repositories' and shows two repository cards. The first card is for 'ra' (Rust PACs for the Renesas RA series devices) with 19 stars and 7 forks. The second card is for 'da' (Peripheral Access Crates (PACs) for Dialog Semiconductor / Renesas Connectivity Devices).

ra-rs

Overview Repositories 2 Projects Packages People

Popular repositories

ra Public

Rust PACs for the Renesas RA series devices

Rust ☆ 19 🍴 7

da Public

This repository contains the Peripheral Access Crates (PACs) for Dialog Semiconductor / Renesas Connectivity Devices

Why does it matters to support?

A general
commitment
towards OSS
since years...

- Renesas has been heavily contributing to Linux Kernel, Zephyr, Xen, and Rust is a logical continuation
- Funding OSS efforts that the industry relies on. Long-term maintenance, supply chain security, crates.io infrastructure: we shall contribute!

A willingness
to support
better SoC
ecosystem

- Profiling / debugging / flashing
- Embedded HALs
- Functional Safety Working Group



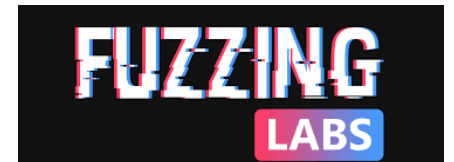
Have a good Rust Paris!
– from Tokyo, with love.

Patrick Ventuzelo

Sécuriser du code Rust en pratique : fuzzing, IA et retours terrain



CEO & Founder
FuzzingLabs



Sécuriser du code Rust en pratique

Fuzzing · IA · Retours terrain

Patrick Ventuzelo · Founder & CEO · fuzzinglabs.com · Rust Paris 2026



8 ans à fuzzer du code Rust en production

Depuis 2019, FuzzingLabs entraîne des équipes sécurité à trouver des bugs Rust, bien avant que ce soit cool.

PATRICK VENTUZELO

Founder & CEO, FuzzingLabs

- Ex-Ministère des Armées
- 27+ talks (Black Hat, OffensiveCon, REcon, SSTIC)
- Rust security audits & trainings since 2019



FUZZINGLABS

Offensive security · EU

- 20+ engineers, bootstrapped, profitable
- Audits Rust : blockchain, crypto, parsing
- Trainings : CrowdStrike, Meta, Google, Apple
- 3x Pwn2Own (AI · IoT · Automotive)



CE TALK

Au-delà de la safety

- Pourquoi Rust safe ≠ Rust sûr
- Bugs qu'on trouve vraiment en prod
- Comment fuzzer du Rust en 2026
- Ce que l'IA change (et ce qu'elle ne change pas)

Rust élimine toute une classe de vulnérabilités mémoire. Pas les bugs.

Rust safe ≠ Rust sûr

Ce que le compilateur garantit. Ce qu'il ne garantit pas. Ce que vous trouvez en production.

✓ Éliminé par le compilateur

- Use-after-free
- Double-free
- Data races
- Null pointer dereference
- La plupart des buffer overflows
- Type confusion via casts implicites

✗ Toujours possible

- Panics : unwrap, expect, panic!, assert!
- Arithmetic overflow (silencieux en release)
- Index out of bounds → panic runtime
- OOM / resource exhaustion → DoS
- Logic errors, state machine bugs
- Unsafe (FFI, transmute, ptr::copy)

La memory safety n'est pas la robustesse. Et la robustesse, en prod, c'est ce qui compte.

Internal server error

Error code 500

Visit cloudflare.com for more information.

2025-11-18 11:35:32 UTC



You

Browser
Working



London

Cloudflare
Error



London

Host
Working

What happened?

There is an internal server error on Cloudflare's network.

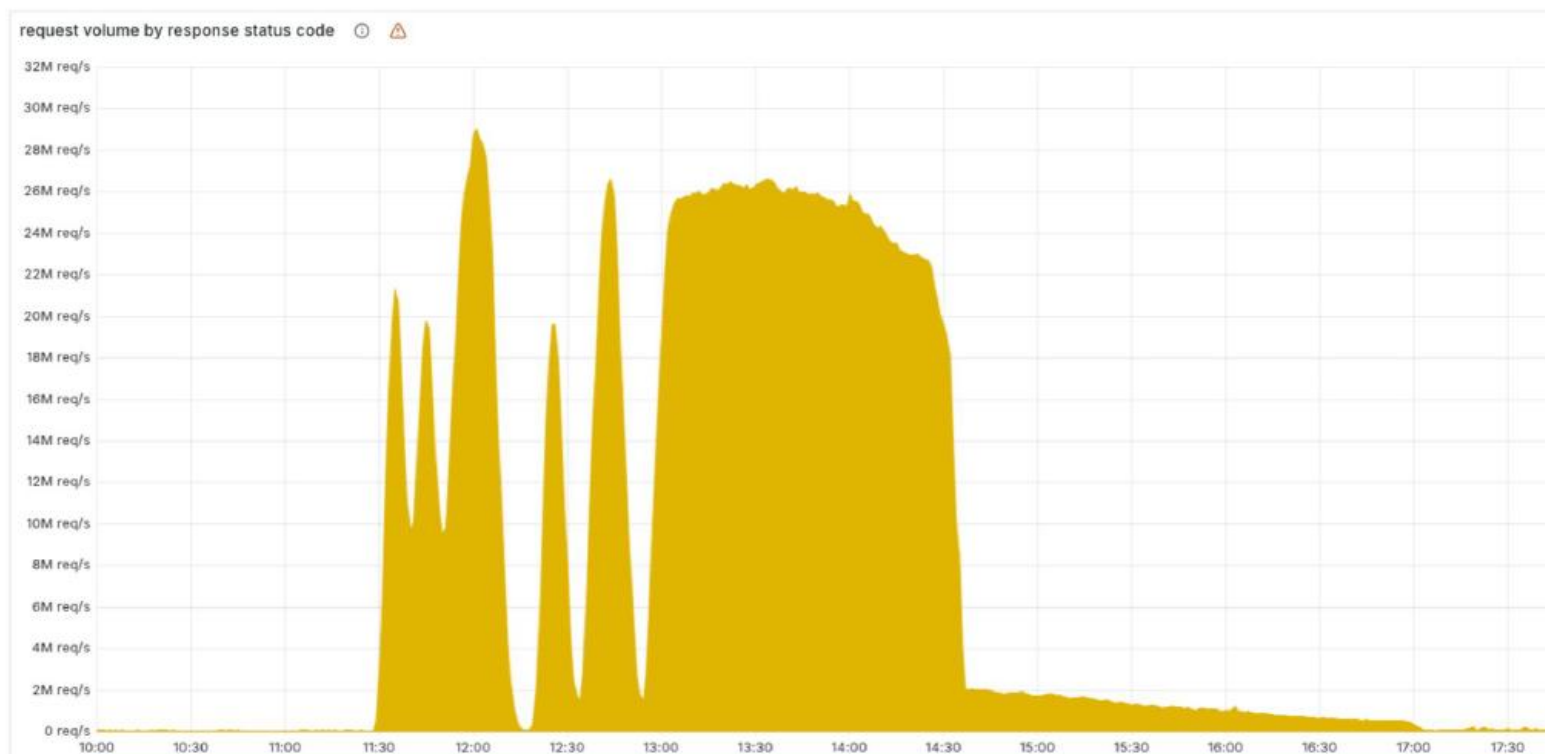
What can I do?

Please try again in a few minutes.

Cloudflare – 18 novembre 2025

[The outage](#)

The chart below shows the volume of 5xx error HTTP status codes served by the Cloudflare network. Normally this should be very low, and it was right up until the start of the outage.

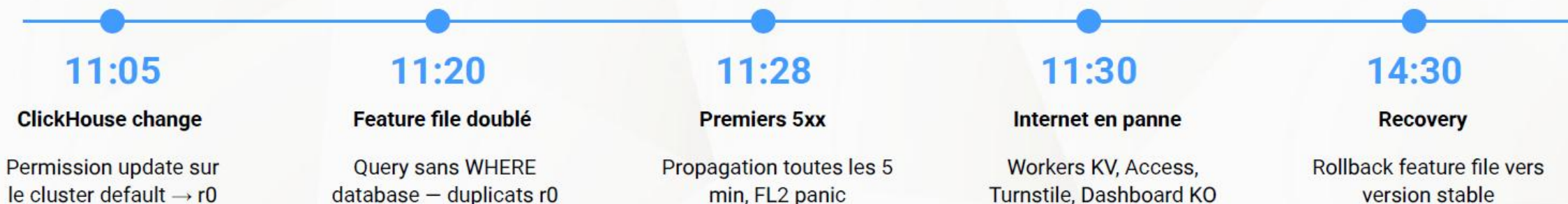


<https://blog.cloudflare.com/18-november-2025-outage/>

Cloudflare – 18 novembre 2025

Worst outage depuis 2019. Aucune attaque. Aucun unsafe. Un seul unwrap().

```
thread fl2_worker_thread panicked: called Result::unwrap() on an Err value
```



Impact : 3h d'outage · CDN · Workers KV · Access · Turnstile · Dashboard · des millions de sites

Pas de RCE. Pas de heap overflow. **Juste un .unwrap() sur un Result.**

Une ligne. Une assumption. La moitié d'Internet à terre. C'est ça, un panic en production.

Cloudflare – 18 novembre 2025

```
71    /// Fetch edge features based on `input` struct into [`Features`] buffer.
72    pub fn fetch_features(
73        &mut self,
74        input: &dyn BotsInput,
75        features: &mut Features,
76    ) -> Result<(), (ErrorFlags, i32)> {
77        // update features checksum (lower 32 bits) and copy edge feature names
78        features.checksum &= 0xFFFF_FFFF_0000_0000;
79        features.checksum |= u64::from(self.config.checksum);
80        let (feature_values, _) = features
81            .append_with_names(&self.config.feature_names)
82            .unwrap();
```

This resulted in the following panic which in turn resulted in a 5xx error:

```
thread fl2_worker_thread panicked: called Result::unwrap() on an Err value
```

Anatomie d'un panic à l'échelle Internet

5 étages. Aucun de mémoire-unsafe. Tous évitables avec du fuzzing.

01

Permission ClickHouse

Query SQL sans filtre
database — retourne r0 en
plus de default

02

Doublement features

Fichier ML feature passe de
~60 à 200+ lignes

03

Préalloc dépassée

MAX_FEATURES = 200
hardcodé dans Bot Mgmt

04

Result::unwrap()

Code FL2 fait .unwrap() sur
le check → panic

05

5xx en cascade

Worker threads panic,
propagé à KV, Access,
Turnstile

LA LEÇON : les inputs « internes » ne sont pas safe. Cloudflare le dit dans son post-mortem : « Hardening ingestion of Cloudflare-generated configuration files in the same way we would for user-generated input. »

Le fuzzing aurait trouvé ça. Tous nos trainings Rust depuis 2019 enseignent exactement ce pattern.

Cloudflare a passé des mois à survivre aux panics

Avril 2026 : « Making Rust Workers reliable » — panic=unwind, Wasm Exception Handling, upstream contributions.
Rust

« For users of Rust Workers, panics were historically fatal, poisoning the instance and possibly even bricking the Worker for a period of time. [...] An unhandled Rust abort might escalate into a broader failure... »

Cloudflare Blog — 22 avril 2026

panic=unwind sur Wasm

Reach extension and retargeting for warm audiences

Abort recovery hooks

- set_on_abort
- exception tags pour distinguer foreign exceptions des aborts
- reentrancy guards

Contributions upstream

- wasm-bindgen
- Walrus parser
- backport modern EH sur Node.js 22/24

<https://blog.cloudflare.com/making-rust-workers-reliable/>

Si une boîte comme Cloudflare doit investir autant pour rendre les panics survivables, c'est un problème systémique.

Le panel des bugs qu'on trouve

Cas réels, crates publiques, 8 ans de fuzzing Rust. Tous trouvables. Tous évitables.

CATÉGORIE	OUTIL DE DÉTECTION	EXEMPLES RÉELS
unwrap / panic	regex + coverage fuzzing	Cloudflare · regex · ntp · image-png
Arithmetic overflow	clippy + fuzzing	zip-rs · base64 · hyperion/http · jpeg-decoder
UTF-8 boundaries	fuzzing avec inputs unicode	xml-rs · juniper · pulldown-cmark
OOM / resource exhaust	fuzzing avec timeout/limit	lewton · image · jpeg-decoder · noir-lang/acvm
Logic errors	differential fuzzing	sodiumoxide · regex · toml-rs · serde-yaml
Unsafe / FFI	sanitizers + MIRI	pancurses · ncurses-rs · lewton (libc)

Aucun de ces bugs n'est de la memory corruption. Tous sont des DoS, des panics, ou des logic flaws.

Le minimum vital : cargo-fuzz

libFuzzer pour Rust. AddressSanitizer activé par défaut. 3 commandes. Aucune excuse.

04

COMMANDES POUR DÉMARRER

- cargo install cargo-fuzz
- cargo fuzz init
- cargo fuzz add parse_input
- **cargo fuzz run parse_input**

Coverage-guided · ASan par défaut

```
fuzz_targets/parse_input.rs

#![no_main]
use libfuzzer_sys::fuzz_target;
use my_crate::Parser;

fuzz_target!(|data: &[u8]| {
    if let Ok(s) = std::str::from_utf8(data) {
        let _ = Parser::new(s).parse();
    }
});
```

C'est tout. Pas d'excuse pour ne pas fuzzer un parser, un protocole, ou une fonction publique exposée.

Structure-aware fuzzing

Fuzzer un parser binaire avec `&[u8]` random ? 99% des inputs rejetés au premier byte. Solution : trait `Arbitrary`.

LE PROBLÈME

`&[u8]` random

- Magic bytes rejetés
- Checksums invalides
- Pas de coverage profonde
- Phase de parsing coûteuse

```
solution - derive(Arbitrary)

#[derive(Arbitrary, Debug)]
struct Packet {
    version: u8,
    flags: PacketFlags,
    payload: Vec<u8>,
}

fuzz_target!(|p: Packet| {
    my_protocol::handle(p);
});
```

Indispensable pour les protocoles binaires, les configs, les AST. Vous sautez le parsing, vous fuddez la logique.

Differential fuzzing – le secret pour les bugs logiques

Le seul moyen efficace de trouver les bugs qui ne crashent pas. La technique la plus sous-utilisée du Rust security.



Input



flate2
(Rust binding)



miniz_oxide
(pure Rust)



assert_eq!(a, b)
Outputs différents → bug
logique

CAS D'USAGE TYPIQUES

01

DEFLATE

flate2 · miniz_oxide ·
libflate · inflate

02

YAML

yaml-rust · serde_yaml ·
libyaml-rs

03

WEBASSEMBLY

wasmer · wasmtime ·
wasmi · wasm3

04

CRYPTO / ASN.1

cryptofuzz · rust-asn1
vs der-parser

C'est comme ça qu'on trouve les bugs qui NE panic pas. La majorité des vrais bugs logiques.

Le crash, et après ?

80% du temps réel est ici. Bucketing, minimization, root cause. C'est là que l'IA va devenir essentielle.

01

Bucketing

Regrouper par stack trace.
Sinon vous noyez dans 10
000 crashes « du même
bug ».

02

Minimization

halfempty, cargo-fuzz tmin.
Le PoC de 1 Mo devient 12
bytes. Reproducible.

03

Debug & RCA

RUST_BACKTRACE=1,
gdb/lldb, rr pour le
time-travel. Comprendre la
racine.

04

Patch & regression

Reproducer ajouté aux
corpora. CI rejouée à
chaque PR. Plus jamais ce
bug.

Le bottleneck n'est plus le fuzzer. C'est le triage. Et c'est exactement là que l'IA arrive.

Le fuzzing manuel ne scale pas

Un projet Rust moderne : 200+ crates transitives, des dizaines d'entrypoints publics, des centaines de PR/an.

200+

Crates
dans un projet Rust moyen

10x

Plus de PRs / an
qu'il y a 3 ans

5 000+

Crashes par Campaign
à trier manuellement

1 jour

Pour écrire un harness
× N targets

OÙ VOUS PERDEZ DU TEMPS

- **Écrire un harness** = boilerplate × expertise du target
- **Trier 5 000 crashes** = stack trace × heuristique manuelle
- **Auditer 200 dépendances** = impossible à la main
- **Maintenir les fuzzers** à travers les refactorings = jamais fait

L'IA dans le fuzzing : multiplicateur, pas

remplaçant

Ce qu'un agent LLM fait mieux que vous. Ce qu'il ne fait toujours pas.



L'IA EXCELLE

Génération de harnesses

Lit pub fn, génère fuzz_target! correct (~80% des cas).

Bucketing sémantique

Clustering de stack traces. Mieux qu'un hash bête.

Reduction de PoC

Itère sur la minimization plus vite que halfempty.

Audit assisté

Repère les patterns dangereux : unwrap, cast, alloc.



L'IA NE FAIT PAS

Inventer un nouveau type de bug

Pas créatif. Reproduit des patterns connus.

Comprendre une vuln logique profonde

Sans spec/RFC, l'agent rate les invariants subtils.

Remplacer le fuzzer

Le LLM ne génère pas les inputs — il génère le harness.

Garantir l'exhaustivité

Pas de preuve. Toujours besoin de coverage measurement.

L'IA accélère ce qui est mécanique. Pas ce qui est créatif. Les deux ensemble = nouvelle frontière.

Architecture d'un agent fuzzing Rust

Orchestration LLM autour de cargo-fuzz. Sortie : issues + PoC + suggestion de patch, prêtes à reviewer.



Code Rust
Cargo project



Audit Agent
AST + LLM
attack surface



Harness Agent
génère
fuzz_target!



Cargo-fuzz
libFuzzer
+ ASan



Triage Agent
bucket
+ minimize



Output
Issue + PoC
+ patch sug.

Open-weight models

Qwen2.5-Coder · DeepSeek-Coder.
Pas besoin de Mythos pour générer
un harness.

On-prem / EU-managed

Le code Rust ne quitte pas le
périmètre client. Sovereign by design.

Continuous integration

Re-fuzz à chaque PR. Detect
regressions. Block bad merges.

Ce n'est pas un scanner. C'est un pipeline d'agents qui auditent et fuzzent du Rust en continu.

Ce qu'il faut retenir

01

Rust élimine les vulns mémoire, pas les bugs.

Cloudflare le prouve à l'échelle Internet. Un `.unwrap()` suffit à faire tomber la moitié du web.

02

Le fuzzing reste indispensable.

Même (surtout) sur du code Rust bien écrit. Panics, OOM, logic bugs : tout est trouvable.

03

Differential fuzzing > coverage-guided.

Pour les bugs logiques qui ne crashent pas. Sous-utilisé en pratique.

04

L'IA accélère ce qui est mécanique.

Harnesses, triage, minimization. Pas la créativité. Multiplicateur de productivité.

05

Mettez du fuzzing en CI dès le jour 1.

Pas le jour où vous avez votre Cloudflare. cargo install cargo-fuzz, ce soir.

Pour aller plus loin



CLOUDFLARE 18 NOV 2025

Post-mortem officiel

blog.cloudflare.com/18-november-2025-outage



MAKING RUST WORKERS RELIABLE

panic=unwind + Wasm EH (avril 2026)

blog.cloudflare.com/making-rust-workers-reliable



FUZZINGLABS ACADEMY

Training Rust Security & Fuzzing complet

academy.fuzzinglabs.com



RUST FUZZ BOOK

Référence officielle cargo-fuzz

rust-fuzz.github.io/book

COMMENCEZ CE SOIR : `$ cargo install cargo-fuzz`

Merci pour votre attention !

Avec-vous des questions ?



Patrick Ventuzelo

Founder & CEO · FuzzingLabs

✉ | patrick@fuzzinglabs.com

🌐 | fuzzinglabs.com · academy.fuzzinglabs.com

Rust Paris 2026

Pause

Reprise à 11h25



Sponsors Gold

CÉNOTÉLIE

TITAGONE
by OCaml PRO

THALES
Building a future we can all trust

Sponsors Silver

AdaCore

oxid[os]

Wylliodyn

Jacques-Henri Jourdan



Chargé de recherche
**Laboratoire Méthodes
Formelles**



Vérifier formellement des programmes Rust avec Creusot

Li-yao Xia



Ingénieur R&D
**Laboratoire Méthodes
Formelles**



Vérifier formellement des programmes Rust avec Creusot



Laboratoire
Méthodes
Formelles



Inria

creusot.rs



Rust Paris - 9 juin 2026

Li-yao Xia
Jacques-Henri Jourdan

Qui sommes-nous ?



Laboratoire
Méthodes
Formelles



Inria

Laboratoire Méthodes Formelles — équipe Toccata
(Inria, CNRS, ENS Paris-Saclay, Université P.-S.)

Creusot en développement depuis 2021
3 chercheurs, 2 ingénieurs, 2 étudiants

Projet *DéCySif*

AdaCore



TRUST  SOFT

*Améliorer la sécurité et la sûreté des
systèmes critiques en utilisant des outils
de **vérification formelle**.*

Comment savoir si mon
programme Rust est **correct**
?

Comment savoir si mon programme Rust est correct ?

Mécanisme 1 : **Typage statique**

- **Système de possession** (*ownership*) pour gérer l'*aliasing* et les durées de vie.
- Aussi : pattern-matching, modules, fonctions génériques, traits...
- « *Si ça compile, ça marche* », dans des cas simples.

Comment savoir si mon programme Rust est correct ?

Mécanisme 2 : **Testing**

- Le langage même encourage et facilite les tests unitaires (`#[test]`, `cargo test`).
- Des crates permettent aussi des tests plus sophistiqués.
- Mais : on ne peut toujours que tester un **nombre fini de cas**.

Comment savoir si mon programme Rust est correct ?

Mécanisme 3 : **Vérification formelle** (preuves de programme)

- **Exhaustivité** : aucune entrée ne peut faire crasher le programme.
- **Précision** : spécifications formelles des entrées et sorties des fonctions.
- « *Si ça compile, ça marche* », mais **pour de vrai**.
- Exemples de programmes formellement vérifiés (dans d'autres langages) :
 - *CompCert* (prouvé avec Rocq) — Compilateur C
 - *seL4* (Isabelle/HOL), *CertiKOS* (Rocq) — Microkernels
 - *fiat-crypto* (Rocq), *HACL** (F*) — Cryptographie utilisée dans Google Chrome et Mozilla Firefox
- **Creusot** permet vérifier formellement des programmes Rust.

Vérifier un programme Rust avec Creusot

```
#[requires(n < u64::MAX)]
```

Précondition : spécifie quelles entrées sont attendues.

```
#[ensures(result == n + 1)]
```

Postcondition : spécifie le résultat de la fonction.

```
pub fn add_one(n: u64) -> u64 {
```

```
    n + 1
}    Cette addition déborde si  $n = 2^{64}-1 = u64::MAX$ 
```

Une fonction respecte sa **spécification** si :

pour toute entrée qui satisfait la **précondition**,

- la fonction ne **panique** pas,
- ne cause pas de **comportement indéfini**,
- et renvoie un résultat qui satisfait la **postcondition**.

Résumé de la méthode :

1. Ajouter des **spécifications** (pre/postconditions...) aux fonctions.
2. Creusot essaie de les prouver automatiquement.

Comment ça marche ?

Rust + Pearlite (spécifications)

```
#[requires(n < u64::MAX)]  
#[ensures(result == n + 1)]  
pub fn add_one(n: u64) -> u64 {  
    n + 1  
}
```

Compilateurs creusot-rustc + Why3

Creusot

formules logiques ($\forall \Rightarrow \exists$)

Prouveurs: Z3, CVC4/5, Alt-Ergo

prouvé/non-prouvé

Une technologie éprouvée

- Technologies similaires pour d'autres langages : Frama-C, SPARK Ada, Why3.
- **Creusot** : vérification formelle appliquée à Rust.
- En plus : grâce à son système de possession, **Rust simplifie vos specs**.

Les intervalles `[dst..dst+n[` et `[src..src+n[` doivent être **disjoints**.

```
/* requires ... */  
void *memcpy(void *dst, void *src, size_t n)
```

C

Rust

```
#[requires(dst.len() == src.len())]
fn memcpy(dst: &mut [u8], src: &[u8])
```

Référence mutable et référence partagée sont nécessairement **disjointes**. Pas besoin de spécifier cela en précondition.

En Rust : **beaucoup moins d'obligations de preuve** pour gérer l'*aliasing*.

Creusot par l'exemple

Exemples d'algorithmes

Recherche par dichotomie

Gnome sort

Cas d'étude

CreuSAT

Bibliothèque standard (`std`)

Recherche par dichotomie

```
#[requires(sorted(arr))]  
#[ensures(match result {  
    Some(i) => arr[i] == elem,  
    None => !arr.contains(elem),  
})]  
pub fn binary_search(arr: &[u32], elem: u32) -> Option<usize> {  
    // ...  
}
```

Le tableau en entrée doit être trié.

Le résultat contient l'indice d'**elem**...
... ou bien **elem** ne se trouve pas dans le tableau.

Facile ?

cf. [Nearly all binary searches and merge sorts are broken](#), Joshua Bloch, Software Engineer, Google (2006)

"I was shocked to learn that the binary search program that Bentley proved correct and subsequently tested in Chapter 5 of *Programming Pearls* [1986] contains a bug."

La preuve "sur papier" avait oublié le débordement arithmétique dans la formule $(1+r)/2$!

Recherche par dichotomie: code et preuve

```
pub fn binary_search(arr: &[u32], elem: u32) -> Option<usize> {  
    let mut size = arr.len();  
    let mut base = 0;
```

```
    #[invariant( ... )]
```

```
    while size > 0 {
```

```
        let half = size / 2;
```

```
        let mid = base + half;
```

```
        match arr[mid].cmp(&elem) {
```

```
            Greater => size = half,
```

```
            Equal => return Some(mid),
```

```
            Less => {
```

```
                base = mid + 1;
```

```
                size = size - half - 1
```

```
            }
```

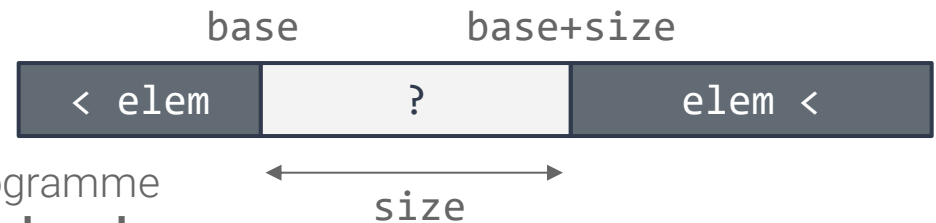
```
        };
```

```
    }
```

```
    None
```

```
}
```

Invariant de boucle : arr =



↑ Découpe la preuve du programme
en morceaux **sans boucle**.

On a trouvé **elem** et on renvoie son indice dans **Some**.

size == 0, donc :

arr =



elem n'est pas dans le tableau et on renvoie **None**.

Recherche par dichotomie: terminaison

`#[check(terminates)]` ← Vérifie que cette fonction **termine**.

`#[requires( ... )]`

`#[ensures( ... )]`

```
pub fn binary_search(arr: &Vec<u32>, elem: u32) -> Option<usize> {  
    let mut size = arr.len();
```

`#[variant( log2(size) )]` ←

```
while size > 0 { ... }  
}
```

Dans chaque boucle,
un **variant** doit décroître à chaque itération, et il ne peut pas
décroître infiniment (ici, c'est un entier positif).

Le variant peut donner une idée de la complexité en **temps**.
Ici, la recherche par dichotomie se fait en temps logarithmique.

Gnome sort

```
pub fn gnome_sort(arr: &mut [u64]) {
    let mut gnome = 0;

    while gnome < arr.len() {
        if i == 0 || arr[gnome-1] <= arr[gnome] {
            i += 1;
        } else {
            arr.swap(gnome-1, gnome);
            i -= 1;
        }
    }
}
```

Cas 1 : `arr[gnome-1] <= arr[gnome]`

Avant :

...	1	2	...
-----	---	---	-----

↑
gnome

Après :

...	1	2	...
-----	---	---	-----

↑
gnome

Cas 2 : arr[gnome-1] > arr[gnome]

Avant :

...	4	3	...
-----	---	---	-----

↑
gnome

Après :

...	3	4	...
-----	---	---	-----

↑
gnome

Gnome sort : spécification

```
#[ensures(sorted(^arr))]
#[ensures((^arr).is_permutation(*arr))]
pub fn gnome_sort(arr: &mut [u64]) {
    // ...
}
```

La valeur finale de **arr** est **triée**.

La valeur finale de **arr** (c'est-à-dire **^arr**) est une **permutation** de sa valeur initiale (***arr**).

Définition de “**trié**” (dans l’ordre croissant) :

```
#[logic]
fn sorted(a: Seq<u64>) -> bool {
    forall<i, j> ... && i < j ==> a[i] <= a[j]
}
```

On peut définir des fonctions et prédicats **réutilisables** dans les specs.

Gnome sort : preuve

```
#[ensures(sorted(^arr))]  
#[ensures((^arr).is_permutation(*arr))]  
pub fn gnome_sort(arr: &mut [u64]) {  
    let mut gnome = 0;  
    let old_arr = snapshot! { *arr };  
  
    #[invariant(sorted_range(*arr, 0, gnome))]  
    #[invariant((*arr).is_permutation(old_arr))]  
    while gnome < arr.len() {  
        if gnome == 0 || arr[gnome-1] <= arr[gnome] {  
            gnome += 1;  
        } else {  
            arr.swap(gnome-1, gnome);  
            gnome -= 1;  
        }  
    }  
}
```

`old_arr` : un “**instantané**” de la valeur initiale de `arr` pour s’y référer dans les `specs`.

Invariant 1 :



Invariant 2 : le tableau courant `*arr` est une permutation de sa valeur initiale `old_arr`.

Creusot par l'exemple

Exemples d'algorithmes

Recherche par dichotomie

Gnome sort

Cas d'étude

CreuSAT

Bibliothèque standard (`std`)

Cas d'étude : CreuSAT (2022)

CreuSAT : un solveur SAT formellement vérifié avec Creusot.

Projet de master de Sarek Skotåm.

Entrée : **formule booléenne** (NOT, OR, AND)

$(NOT(a) OR b) AND (a OR c) AND (b OR c)$

Sortie : si possible, une valuation qui **satisfait la formule**

$a = vrai, b = vrai, c = faux$

Implémente **plusieurs optimisations et heuristiques** non-triviales.

Spécification :

```
#[ensures(match result {  
  Sat(v)  => formula_sat(formula, v),  
  Unsat   => formula.not_satisfiable(),  
  Unknown => true,  
}])  
pub fn solver(formula: &Formula) -> SatResult
```

la valuation v satisfait la formule

la formule n'est pas satisfiable

le solveur peut abandonner

Cas d'étude : la bibliothèque standard

Challenge proposé par Amazon : vérifier des fonctions de la bibliothèque standard de Rust, prouver leur **sûreté** (absence de comportement indéfini).

Challenge 17 : Slices (un des types de tableaux de Rust)

- Exemples : `index`, `split_at`, `copy_within`, `binary_search_with`.
- Manipulations de **pointeurs bruts** (*raw pointers*, équivaut aux pointeurs de C).
- **Actuellement : 30 fonctions vérifiées, 7 restantes.**
- Ce projet a motivé le développement de fonctionnalités pour **vérifier du code pré-existant**.

Aussi en progrès : `Rc` et `Arc` (*reference counting*, [Challenge 26](#))

VerifyThis 2026

VerifyThis : concours de vérification formelle, depuis 2011.

- Implémenter des algorithmes et surtout **prouver formellement leur correction**.
- **4 problèmes** en temps limité (~ 2h / problème).
- Jugés sur la rigueur, la complétude, et le style.
- ~ 20 équipes. Pas restreint à Rust : **tous les outils, tous les langages sont permis**.
- Cette année, **l'équipe Creusot a remporté le premier prix !** 🎉

Conclusion

Creusot permet de **prouver** la correction de programmes Rust.

Ingrédient crucial : **spécifications formelles**.

Preuves **semi-automatiques** avec solveurs SMT.

Soutien pour l'essentiel des features de Rust : génériques, traits, clôtures...

Et plus encore :

- preuves de **terminaison**,
- code unsafe avec **pointeurs**,
- concurrence avec **atomiques** : seq. consistent, release-acquire, relaxed.



creusot.rs

Dylan Brasseur



HPC expert
Aneo



Florian Lemaitre



HPC expert &
Cloud architect
Aneo



Quand Python ne suffit plus :
transférer des **Petaoctets** de
manière efficace et sûre



Quand Python ne suffit plus

Transférer des Pétaoctets de manière efficace et sûre

Dylan Brasseur & Florian Lemaitre

Rust Paris 2026

Qui sommes nous ?



Florian Lemaitre
HPC Expert



Dylan Brasseur
HPC Senior Consultant



aneo, en 30 secondes



Cabinet de conseil

Tech & transformation, basé à Boulogne-Billancourt.



Advanced Computing

HPC, calcul distribué, performance, cloud hybride.



Open source

ArmoniK et ses composants publiés sous licence libre.

PME française fondée en 2002
Modèle aplati depuis 2015

22,5M de CA en 2025

130 collaborateurs

Au programme

1. Le contexte
2. Premier jet : Python et C++
3. Réécriture en Rust
4. Bonus : la suppression massive





1. Le contexte

Contexte

- Un industriel automobile européen nous contacte pour migrer ses données sur GCP
- Des PB à transférer, rapidement, et en garantissant l'intégrité
- Une « petite » partie d'un grand plan de migration des ressources de calcul
- Données stockées sur des buckets S3 on-premises

Besoin d'un système :

- Rapide
- Facile à exécuter
- Fiable



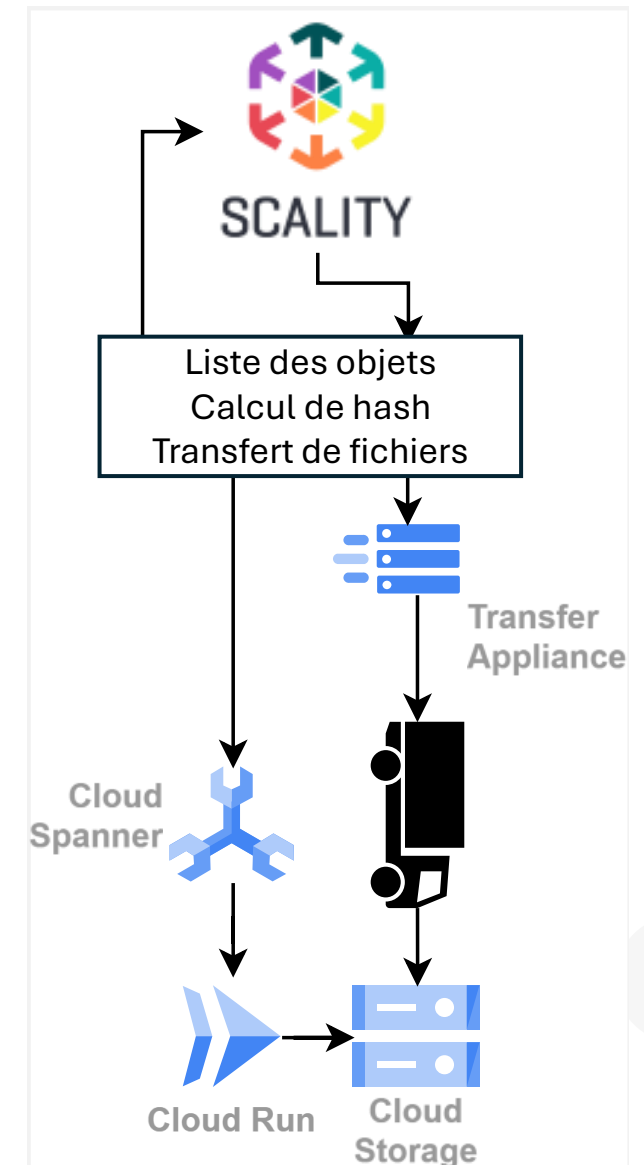
Campagne 1 : Transfer Appliances

Le plan :

- Utiliser des **Transfer Appliances** de Google : des baies de disques chiffrées, livrées physiquement chez Google une fois remplies.
- On y écrit les fichiers, on les expédie, GCP les ingère.

Garantir l'intégrité:

- Calculer les hash à la volée
- Comparer la source et ce qu'on a calculé
- Une fois chez GCP: On vérifie l'arrivée et ce qu'on a calculé, depuis une base de données
- Triage des données selon les besoins



Campagne 1 : Solution et résultats

Les données et le réseau:

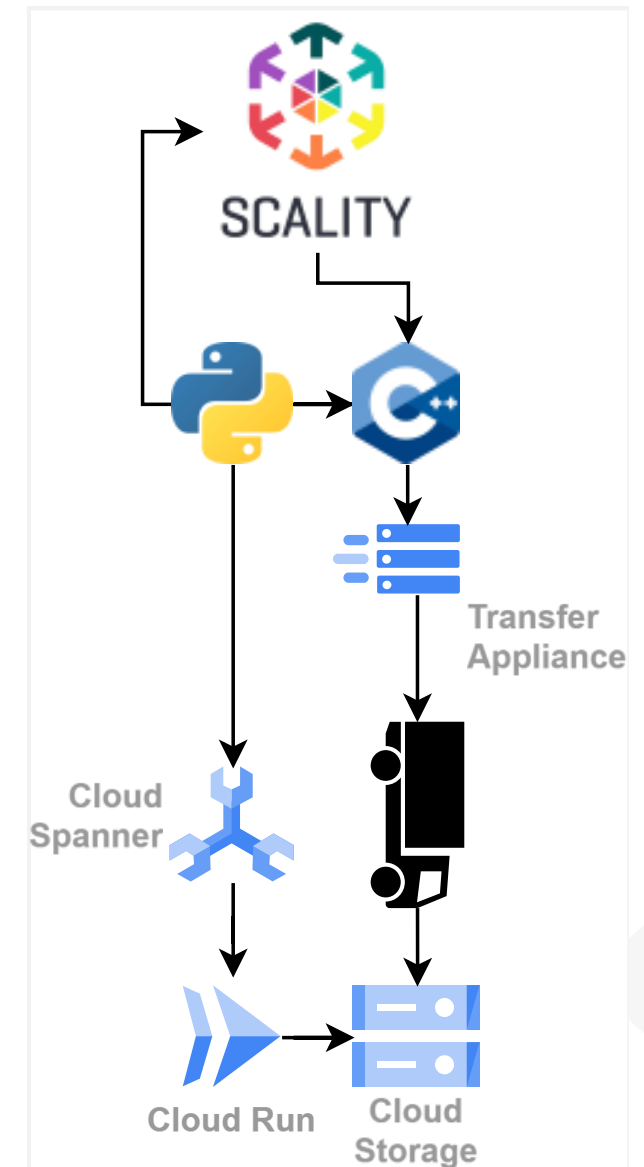
- Fichiers volumineux
- Peu de fichiers
- Latence faible (réseau interne seulement)

⇒ Latence par objet anecdotique, seul le débit compte

La solution :

- Coordonner les transferts via un script Python
- Calculs de hash à la volée et transfert via un programme en C++
- Comparaison des hashes de source et d'arrivée.

⇒ 6 PB transférés, client content.



Campagne 2 : Interconnect

Pas de Transfer Appliance cette fois:

Passage par un Interconnect de 10Gbps

Envoi plutôt qu'écriture sur disque

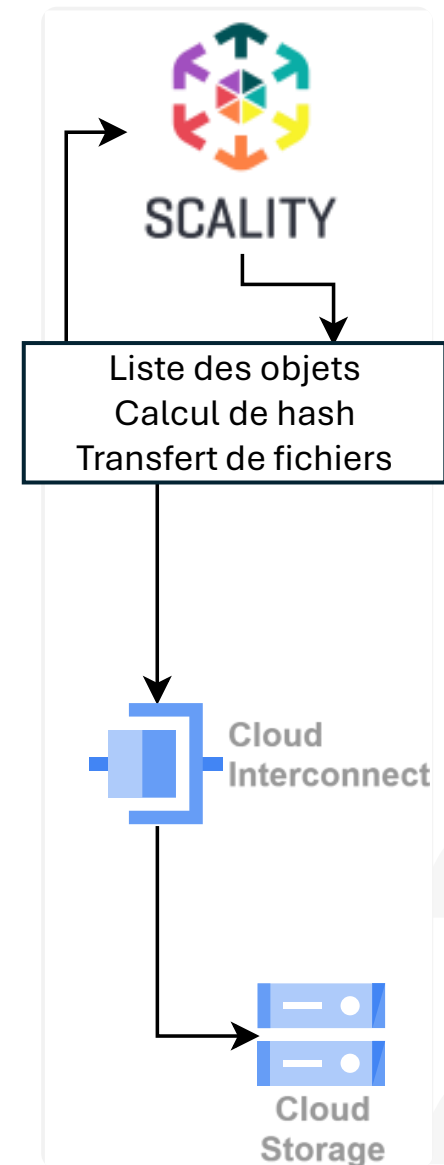
Typologie de fichiers:

Beaucoup de petits fichiers

Quelques PB à transférer

Facile!

On reprend ce qu'on a fait pour la première campagne, et on change juste le nécessaire!



2. Premier jet : Python et C++

Tentative 1 : Python & C++ & Gcloud CLI

Ne jamais matérialiser les fichiers

Manipulation de pipe Unix et file descriptors

Au lieu d'écrire sur disque, le programme C++ écrit dans un pipe

Utilisation de gcloud storage cp pour envoyer le contenu du pipe sur GCS

Avantages

Très peu de changements

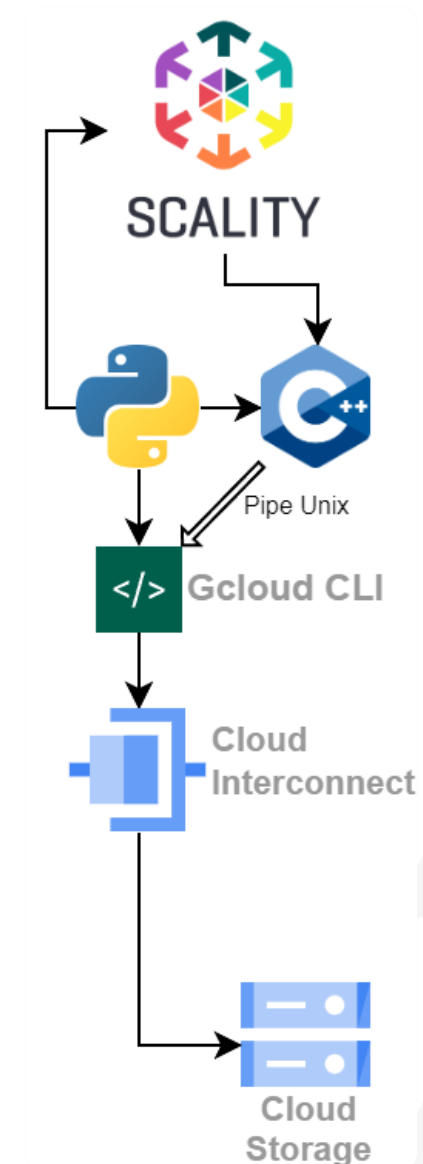
Pas de gestion de credentials

Problèmes

Fork de 2 process pour chaque fichier

Startup Gcloud ajoute de la latence (~300ms)

Avec des millions de fichiers: intenable



Tentative 2 : Python & C++ & GCS SDK

Utiliser le SDK GCS

Utilisation avec multiprocessing pour éviter le GIL

Avantages

Eviter un fork couteux

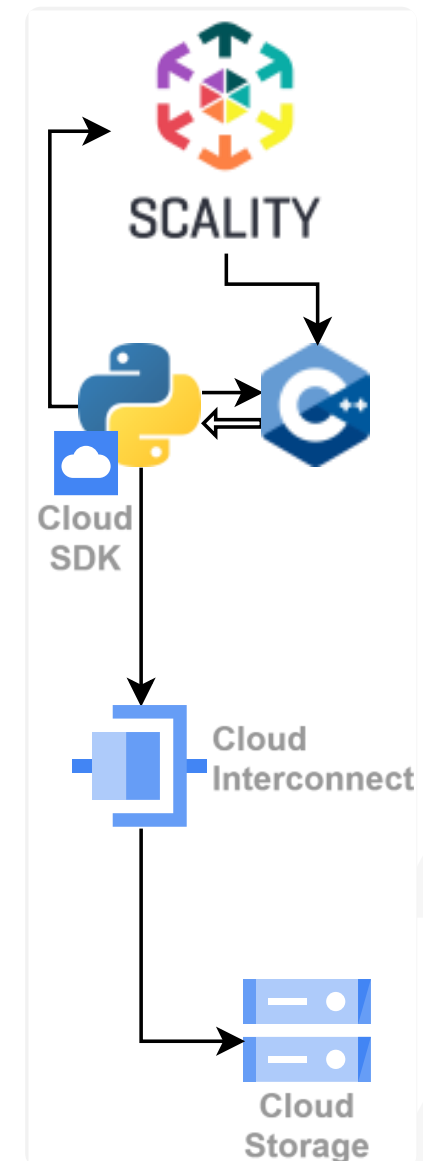
Réutiliser l'authentification à GCP

Problèmes

Latence d'établissement de connexion HTTPS à chaque transfert

Explosion mémoire/CPU des process Python en parallèle

C'est mieux, mais on est trop lourd et toujours trop lent pour les petits fichiers.



3. Réécriture en Rust



Architecture du pipeline Rust

On reprend la chaîne entière dans un même programme Rust :

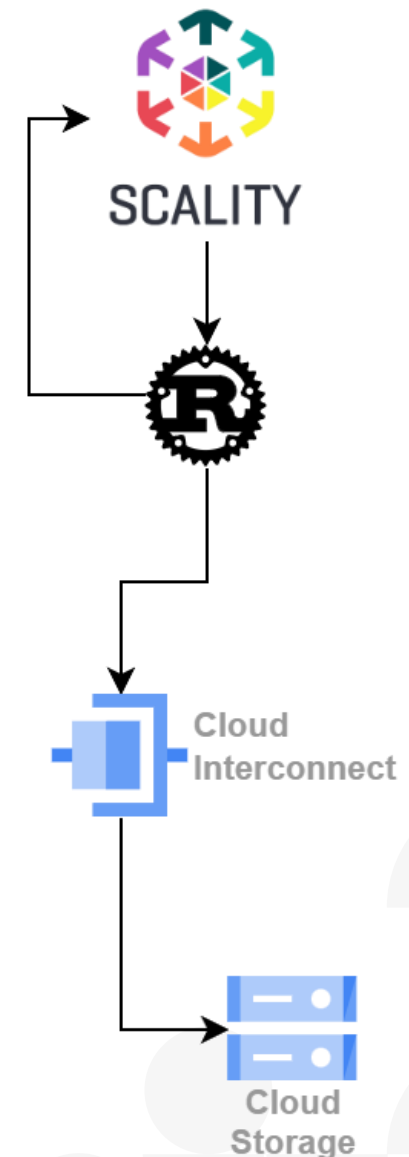
- Téléchargement depuis Scality
- Calcul des hashes
- Upload vers GCS

Pas d'écriture disque, pas de sous-process.

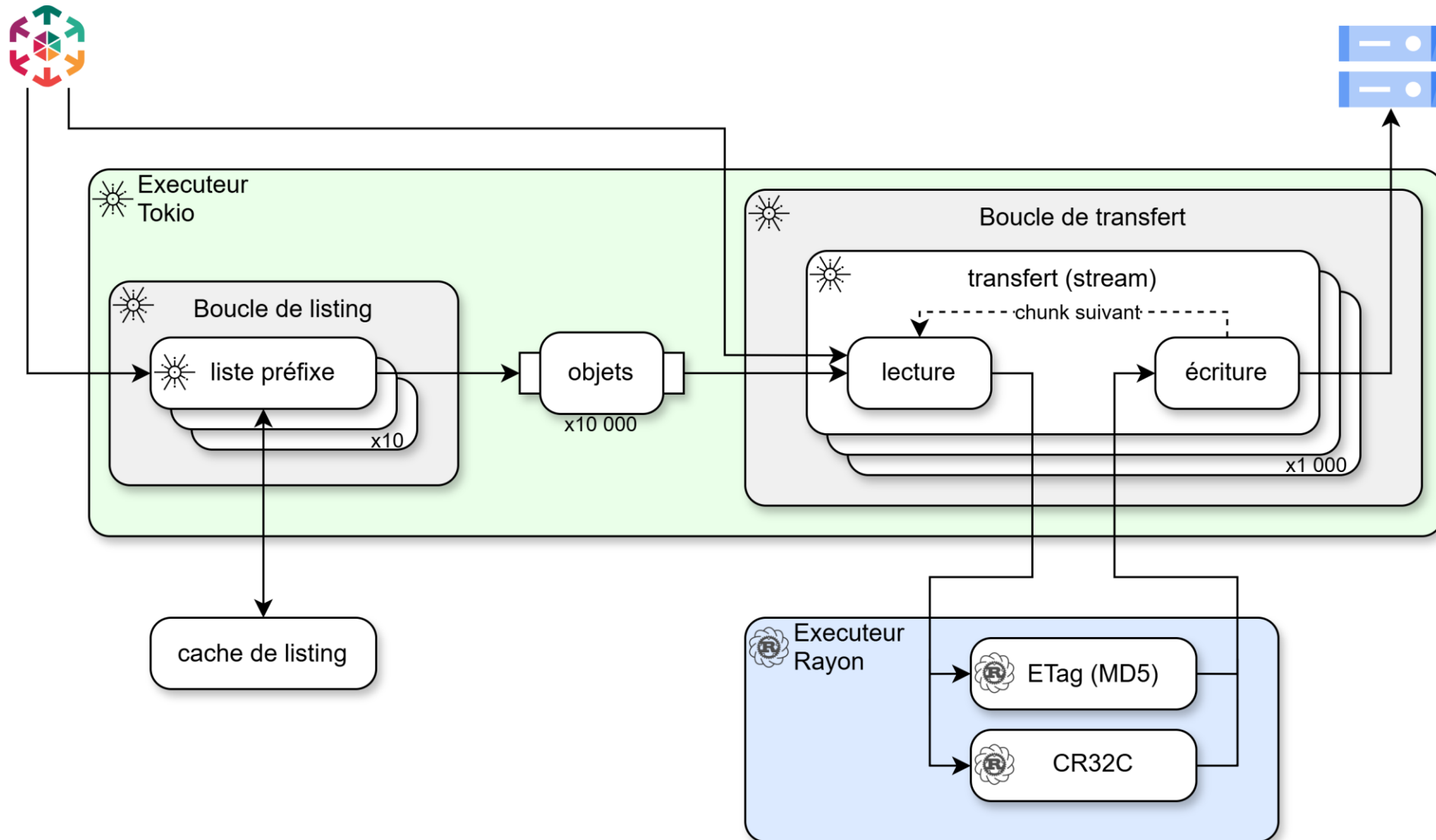
Tokio pour les I/O asynchrones (un seul runtime, des milliers de transferts concurrents)

Rayon pour le travail CPU du hashing.

Client GCS Rust partagé réutilise les connexions HTTPS au lieu d'en ouvrir une par objet.



Architecture du pipeline Rust



Montre-moi du Rust !

Stream de transfert

```
while let Some(chunk) = downloader.try_next().await? {
    let mut hashes = Vec::with_capacity(hashers.len());
    for mut hasher in hashers.drain(..) {
        let chunk = chunk.clone();
        let (tx, rx) = tokio::sync::oneshot::channel();
        rayon::spawn_fifo(move || {
            hasher.1.write(&chunk);

            if tx.send(hasher).is_err() {
                log::error!("Could not send back hasher to transfer task");
            }
        });

        hashes.push(rx);
    }

    hashers.append(&mut futures::future::try_join_all(hashes).await?);

    ctx.instant_bytes.success.add(chunk.len());
    yield chunk;
}
```



```
Listing 987/987 [0.2255/s] 24m/24m ETA:0s Skipped 0 Errors 0
Objects 442831/443,132 [490.3305/s] 24m/24m ETA:1s Skipped 662747 Errors 0
Data 309.99 GiB/312.72 GiB [403.97 MiB/s] 24m/24m ETA:7s
```


Résultats de la migration

10 Gbps

Débit soutenu côté client : lien Interconnect 10 Gbps saturé, hashes calculés à la volée.

5 jours

Temps de développement total du programme Rust, comparable au temps qui aurait été nécessaire pour tuner la version Python.

200 M

Objets transférés avec intégrité vérifiée, dans les délais de la mise en production.

Avant



1300 lignes de Python



700 lignes de C++

Débit: ~100 objets/s
CPU: 1600%

Après



1700 lignes de Rust

Débit: ~1500 objets/s
CPU: 600%



4. Bonus : la suppression massive

Bonus: Supprimer 26M d'objet GCS

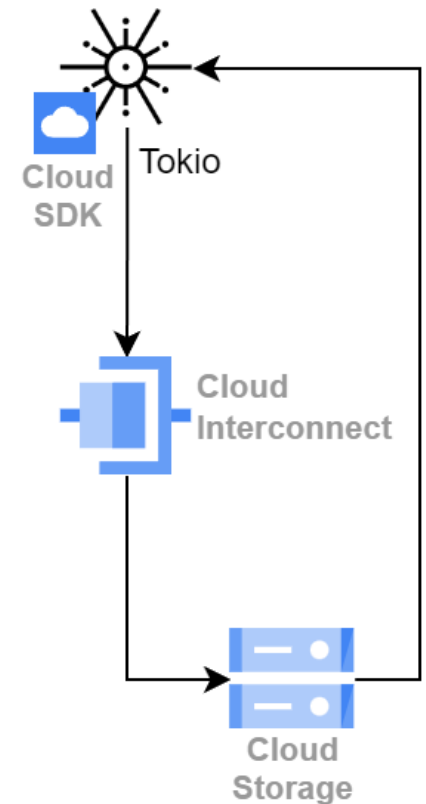
Devoir supprimer des fichiers selon des critères précis sur GCS

Notre client fait un script en **bash** (gsutil + gcloud):

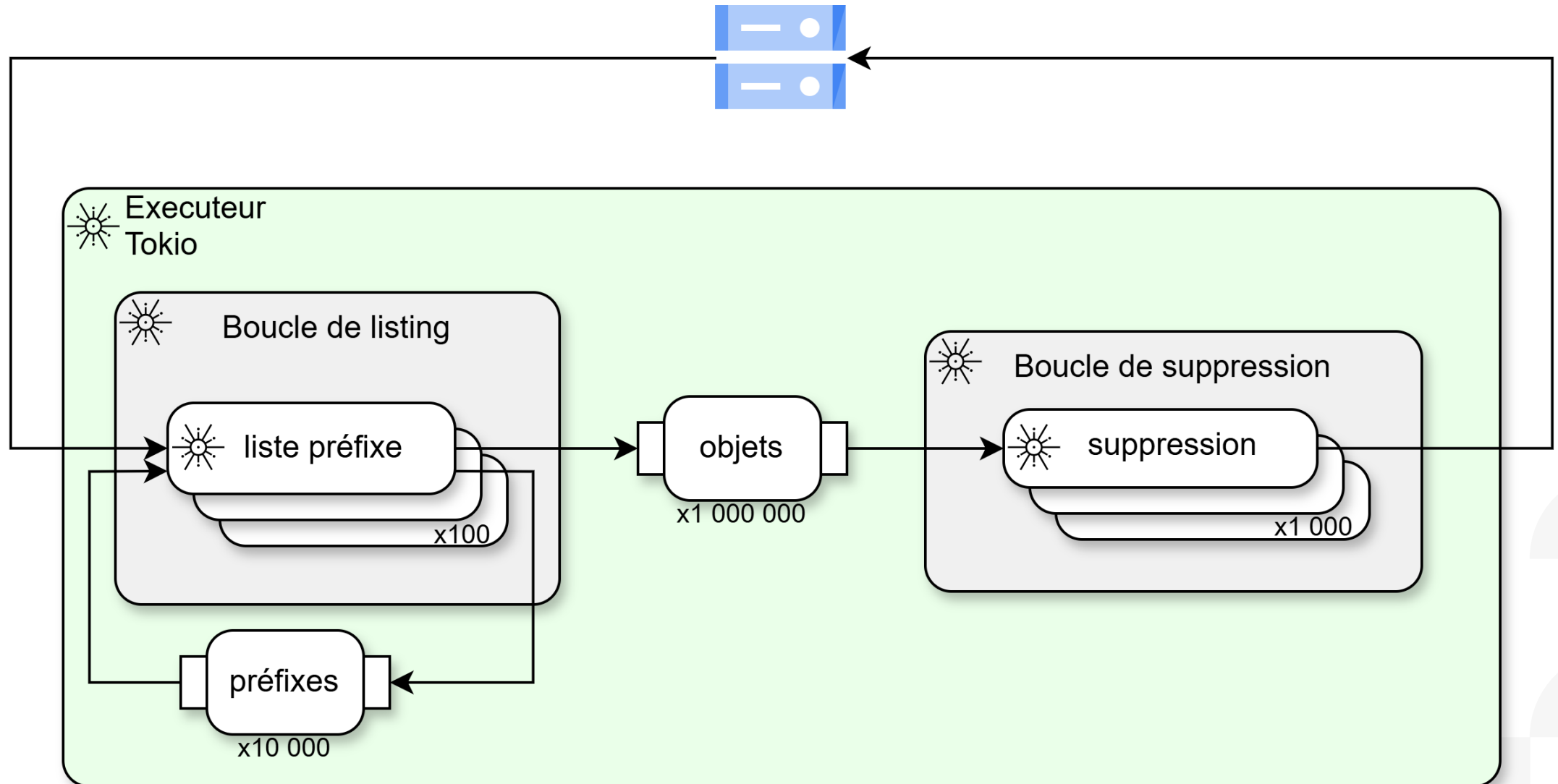
Après plus de 3 jours d'exécution, le script est encore bloqué dans le listing des objets. Pas un seul objet n'a été supprimé.

Tel un script, on reprend notre architecture et on change quelques appels:

Listing + filtrage + suppression en flux, en parallèle sur un même client GCS.



Bonus: Supprimer 26M d'objet GCS



Résultats de la suppression

3 jours

De développement, autant que le temps d'exécution du script bash (sans succès)

14 Heures

D'exécution pour tout supprimer

26 M

Objets supprimés

Avant



100 lignes de Bash

0 Objets supprimés après 3 jours

Après



700 lignes de Rust

Débit de listing: 60k/s
Débit de suppression: 1000/s
CPU: 80%



À l'échelle du Pétaoctet, c'est la donnée qui choisit l'outil.

Merci !

Rust Paris 2026

Cocktail Déjeunatoire

Reprise à 14h



Sponsors Gold

CÉNOTÉLIE

TITAGONE
by OCaml PRO

THALES
Building a future we can all trust

Sponsors Silver

AdaCore

oxid[os]

Wyliodrin

Rust Paris 2026

Florian Bennetot
DevOps Engineer
Parsec



Sponsors Gold

CÉNOTÉLIE

TITAGONE
by OCaml PRO

THALES
Building a future we can all trust

Sponsors Silver

AdaCore

oxid[os]

Wyliodrin

Thomas Clemenceau



Ingénieur logiciel
embarqué et critique
Thales

THALES
Building a future we can all trust

Thomas Emerdjian



Ingénieur Développeur
logiciel embarqué
Thales

THALES
Building a future we can all trust

Architecture à composants : coup de neuf avec Rust

Architecture à composants Coup de neuf avec Rust !

Présentation par Thomas²



www.thalesgroup.com



Contexte



> Contexte industriel défense

Fort historique

- Logiciel critique
- Temps-réel
- Cibles embarquées

> Architecture à composants

Séparation stricte

- Métiers d'ingénierie spécialisés
- Secrets
- Technique



Le contrat critique



Spécification système

└ Contrats fonctionnels

└ Architecture logicielle

└ Composants / Code

└ Tests unitaires

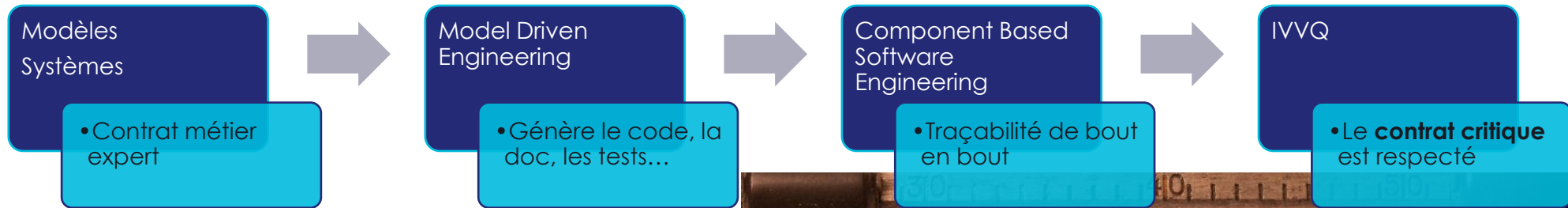
└ Tests d'intégration

└ Validation système

└ Certification – SDF

La chaîne qui tient

> Le langage change, la chaîne outillée reste



ETUDES MENÉES



- ▶ **A : Etude sur la faisabilité** d'utiliser le Rust dans une architecture à composants et sur un générateur Rust à partir de modèles.
- ▶ **B : Etude sur la maturité des middlewares** de communication disponibles en Rust et sur l'interopérabilité Rust/CPP. (+ retex Zenoh)
- ▶ **C : Mise en conditions réelles** avec le cas d'usage Reaction Wheel (Ada to Rust).

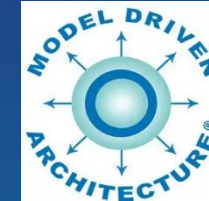
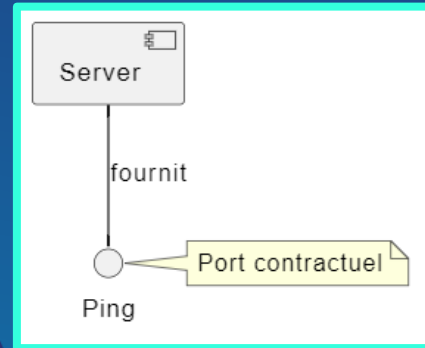


Étude A : Rust pour CBSE

Générateur Rust

www.thalesgroup.com

BUSINESS MODEL



TEMPLATES



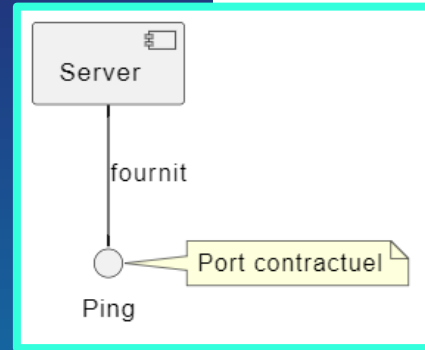
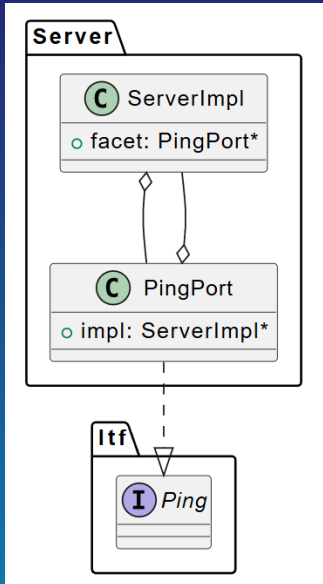
METAMODELS



GENERATED
CODE BASE



C++ Architecture historique

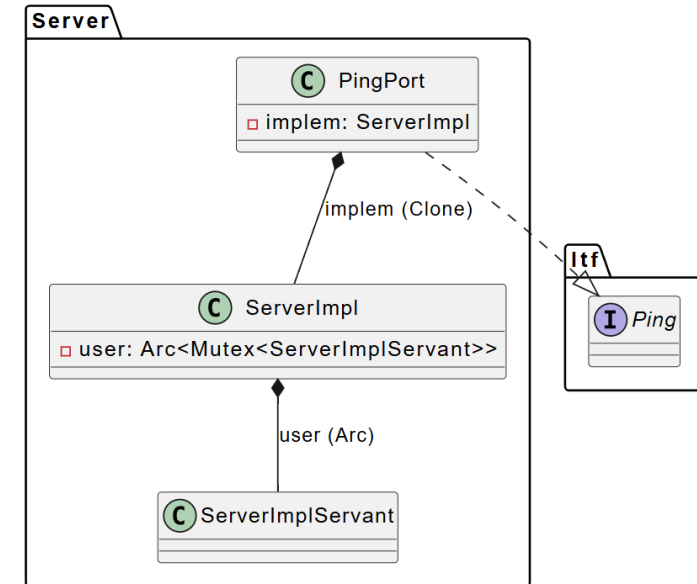


```
class PingPort {
    ServerImpl* implem; // Pointeur nu - C++ : OK
};

class ServerImpl {
    PingPort port; // Cycle - C++ : OK
};

int main() {
    PingPort ping = PingPort();
    ServerImpl server = ServerImpl(ping);
    ping->implem = server;
}
```

Rust Nouvelles contraintes



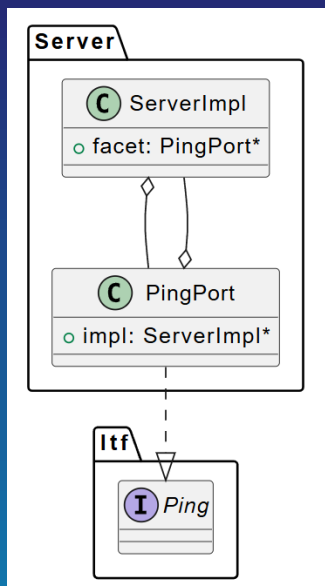
```
pub struct PingPort {
    implem: ServerImpl; // Composition - Rust : OK
}

pub struct ServerImpl {
    port: PingPort; // Cycle - Rust : KO
}

fn main() {
    PingPort ping = PingPort
    implem: ServerImpl { facet: ping }
}
```



C++ Architecture historique



```

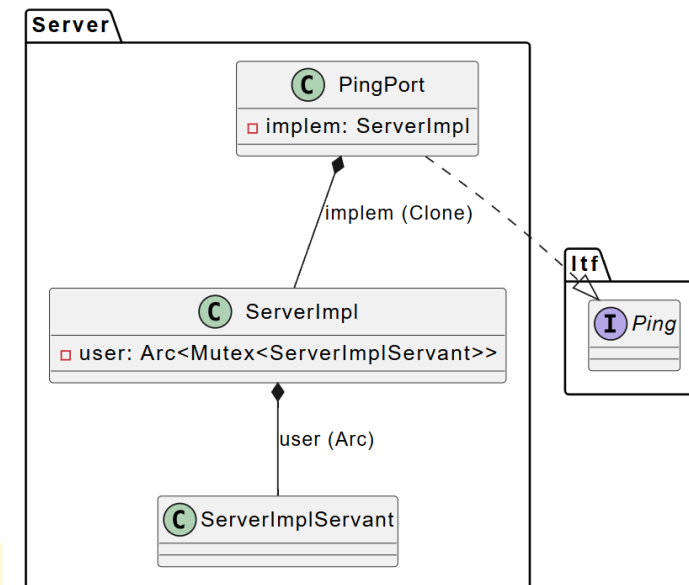
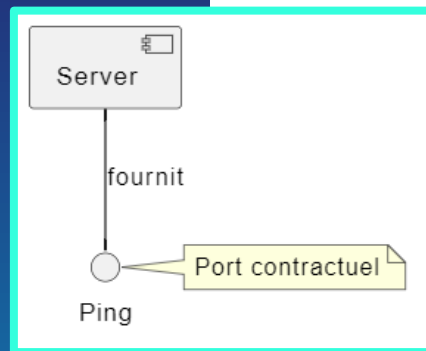
class PingPort {
    ServerImpl* implem; // Pointeur nu - C++ : OK
};

class ServerImpl {
    PingPort port; // Cycle - C++ : OK
};

int main() {
    PingPort ping = PingPort();
    ServerImpl server = ServerImpl(ping);
    ping->implem = server;
}

```

Rust Nouvelles contraintes



+ OWNERSHIP

```

#[derive(Clone)]
pub struct ServerImpl {
    context: Arc<RwLock<ServerImplContext>>,
}

pub struct PingPort {
    implem: ServerImpl,
}

fn main() {
    let server = ServerImpl::new();
    // Meme contexte pour tous les clones de 'server'
    server.config_context();

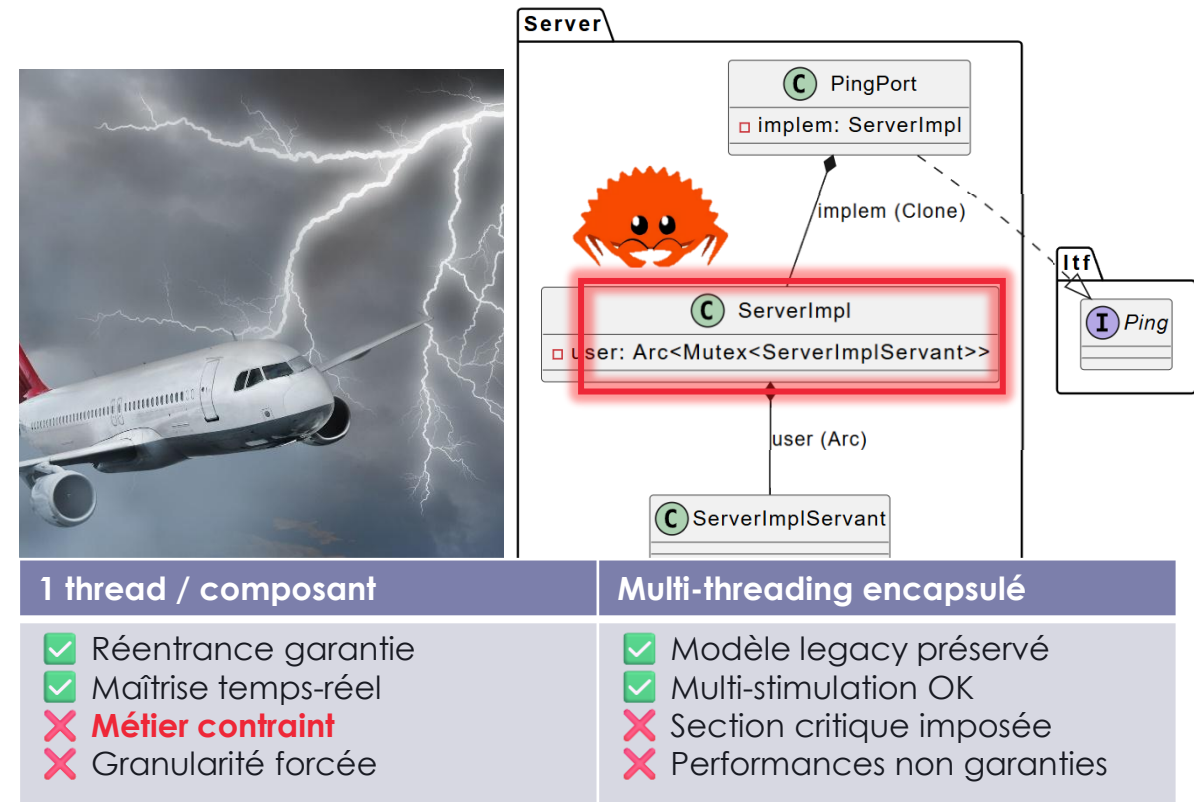
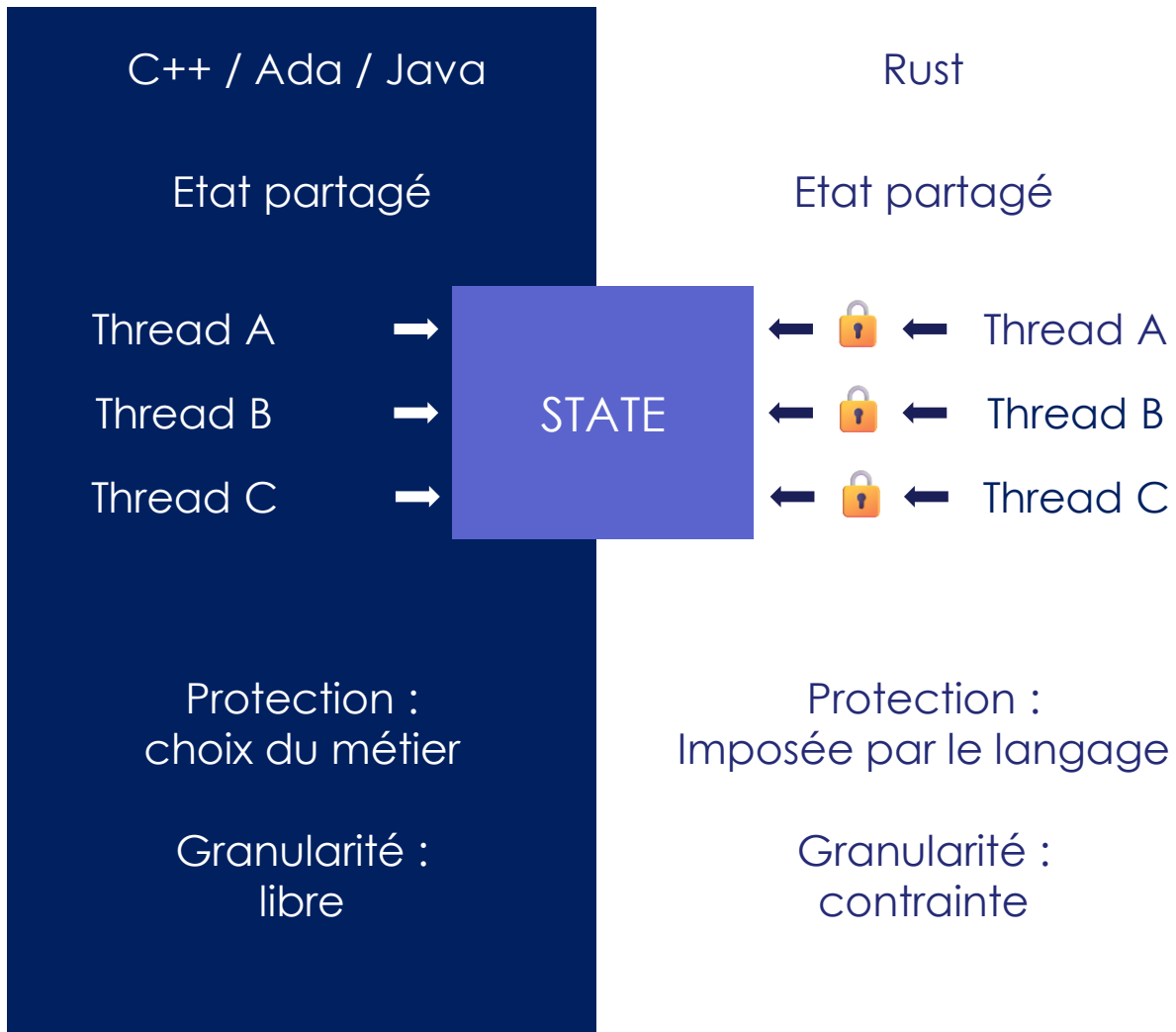
    let port_une = PingPort::new(server.clone());
    let port_deux = PingPort::new(server.clone());
}

```



Un choix imposé

 Rust et l'état partagé : quand le langage devient architecte



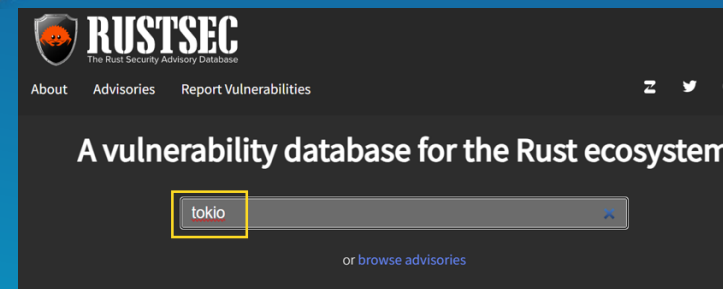
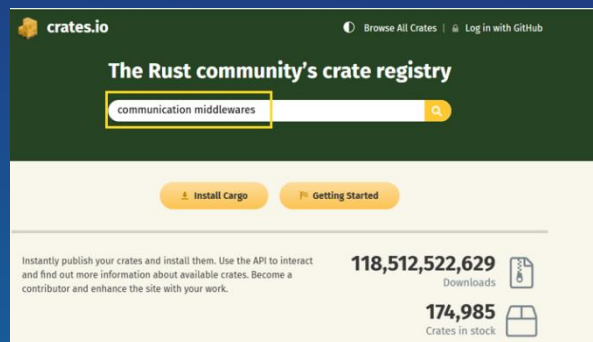
Deux options — deux coûts

NOTRE CHOIX

Étude B : Middlewares de communication

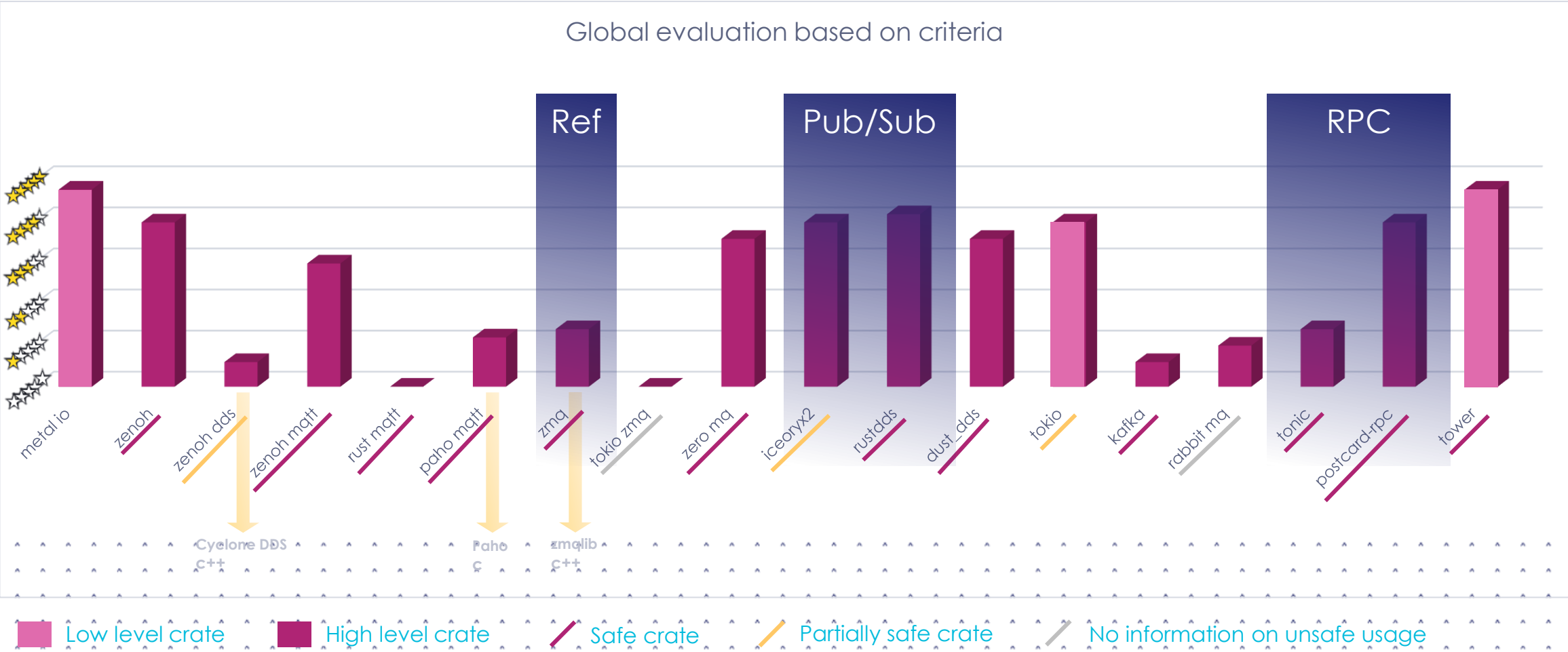
Maturités des crates

www.thalesgroup.com



Middleware et maturité

> Peut-on s'appuyer sur l'écosystème Rust dans un contexte embarqué Offline?



API COMMUNE

API Commune peu-importe le middleware !

```
pub trait EventProducer {  
    fn reserve(&self) -> Writer;  
    fn push(&self, buffer: Writer);  
}  
  
pub trait EventConsumer {  
    fn read(&self) -> Reader;  
}
```

Spécificités pour chacun

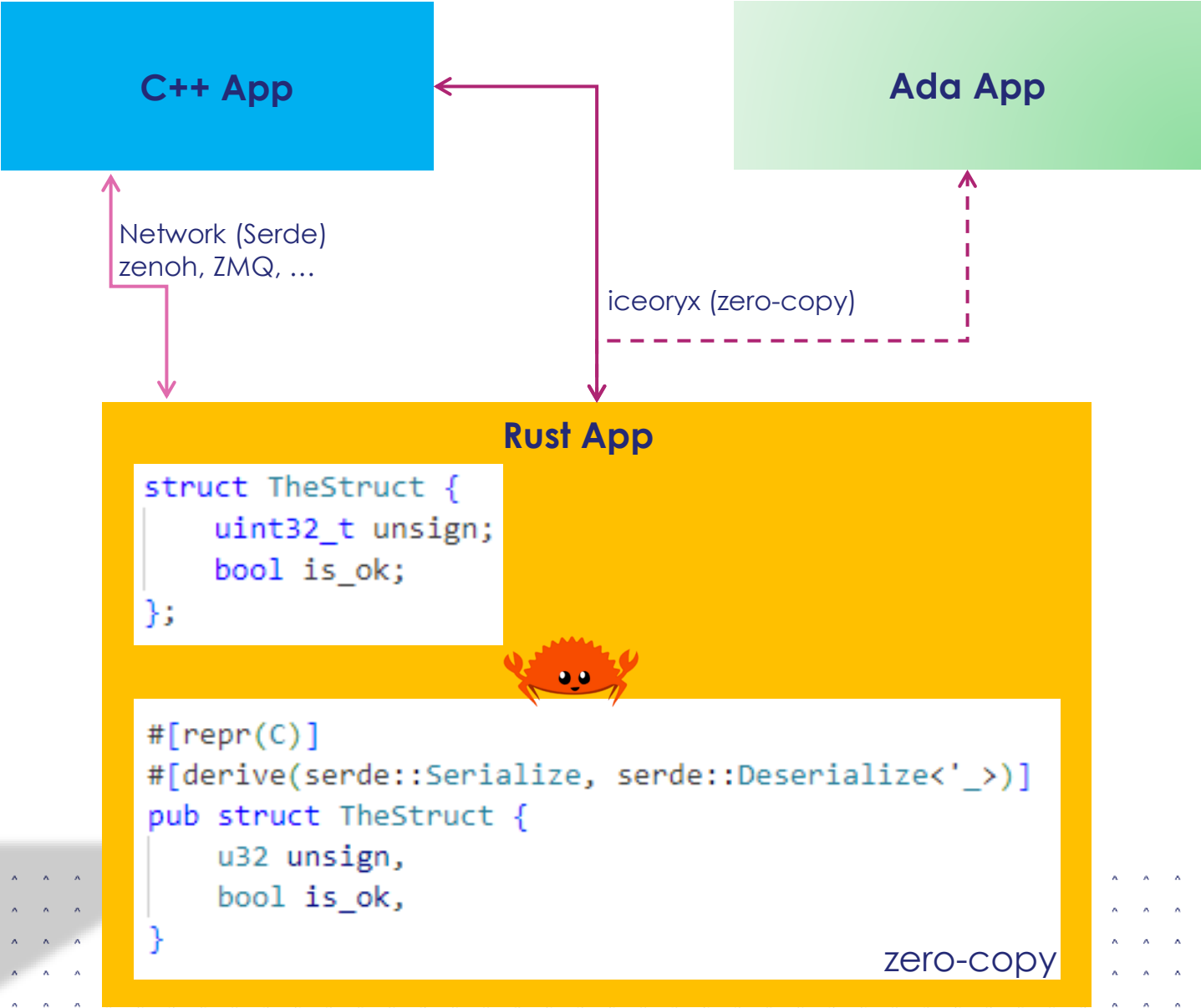
```
pub struct ZenohPub {  
    stuff: ZenohStuff,  
}  
  
pub struct ZenohSub {  
    stuff: ZenohStuff,  
}  
  
impl EventProducer for ZenohPub {...}  
impl EventConsumer for ZenohSub {...}
```



Mapping and Interop



IDL	Rust	C++	Ada use Standard;
boolean	bool	bool	Boolean
int32	i32	int32_t	Long_Integer
char	char	char	Character
string	String / &str	char*	String_Type
module	module (mod)	namespace	package
alias	type Alias = Type;	typedef Type Alias;	Type dérivé
array	[Type; size]	std::array<Type, size>	Array(<>) of Type



#[CFG] ET DÉPENDANCES OFFLINE

⚠ Des dépendances qui compilent 200 crates

- `zmq` → `libzmq` → `openssl` → ... → 47 crates
- `iceoryx2` → ... → 23 crates

En offline : toutes vendorées, auditées, figées
(`cargo audit`, `cargo vendor`)

Coût : temps de build × surface d'audit

Imports

- code lié aux crates -> module dédié
- Proxy sur les libs + cycle de vie encapsulés

-> *surface unsafe minimale et localisée*
(`auditable cargo geiger`)

Étude C : REACTION WHEEL

ADA -> RUST

www.thalesgroup.com



Cas d'usage réel : réaction WHEEL

Code existant en Ada et C

Peut-on porter vers Rust sans changer les modèles ni les plans de test ?

INVARIANTS

- Contrat métier critique
- Modèles
- Plan de tests et validation

Contraintes :

- Temps-réel OS-dépendant
- CI/CD : UT + couverture + intégration + environnement offline



FFI, TEMPS RÉEL, NO-DEP

Le RETEX honnête

- Ada/GNAT Pro : l'édifice certifié
- Rust + crate libc : les briques – à assembler (ex: paramétrage ajouté à std::sync)

Ce que nous faisons en Ada depuis 20 ans

On le reconstruit en Rust

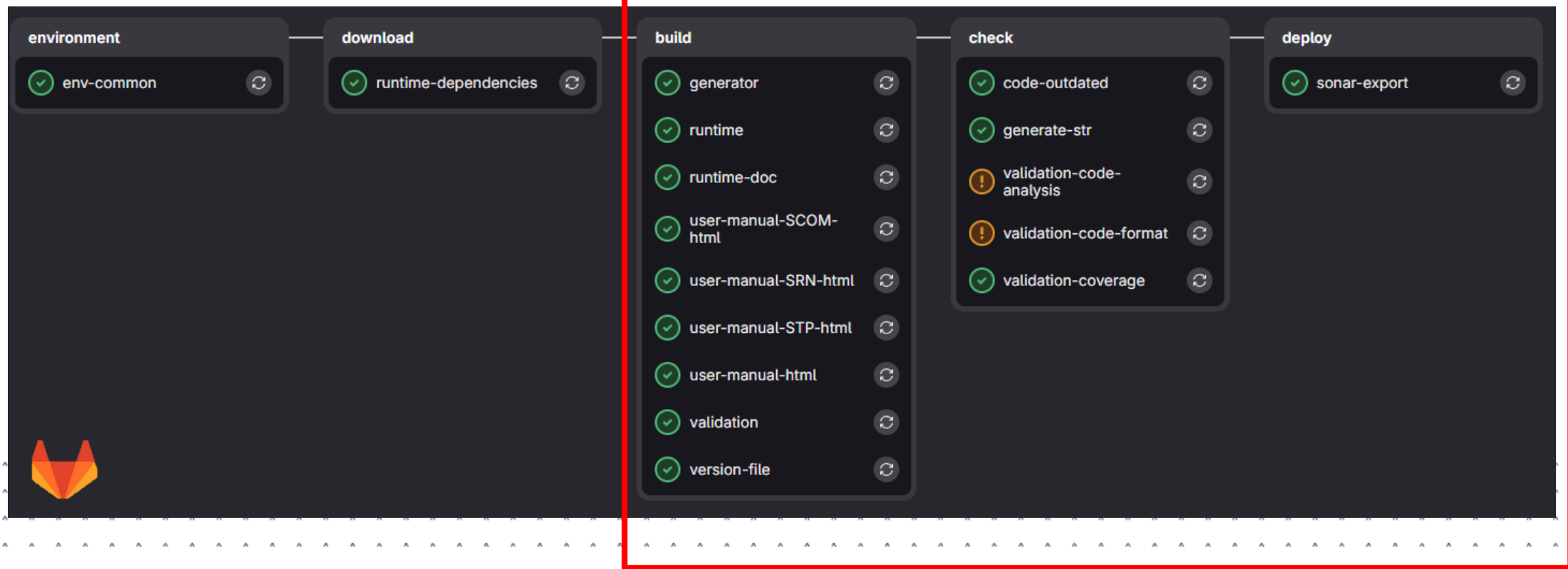
Rust std – ce qui manque pour le critique temps-réel (std::sync::Condvar + std::thread::Thread)

- Priorité du thread
- Politique d'ordonnancement
- Affinité CPU
- Horloge explicite **CLOCK_MONOTONIC**

CI/CD offline : le pipeline

> Tests d'intégration sur modèles legacy

- Plans de test Ada/C++/Java/Rust — identiques
- Le contrat métier prime sur le code ✓



CONCLUSIONS





Les modèles ne changent PAS

> Les contrats ne changent PAS.

> Les plans de test ne changent PAS.

> Ce qui change par langage :

- Architecture code généré : relation cyclique → Arc
- Granularité : état partagé → Mutex<T> explicite
- Temps-réel : libc → reconstruit
- Certification : en cours → maturité à construire

Rust : un langage de plus

Un seul maître :

le contrat critique



Ce que Rust apporte au contrat critique :

- ✓ **Ownership** → propriétés démontrées à la compilation
- ✓ **Result<T,E>** → gestion d'erreur non ignorable
- ✓ **Arc<Mutex>** → concurrence explicite et auditable
- ✓ **repr(C)** → interop certifiable
- ✓ **CI/CD** → reproductibilité bit-à-bit

Ce que Rust ne remplace pas :

- ⚙ Le contrat métier défini par vos experts
- ⚙ La SDF, elle précède tout langage
- ⚙ SPARK/Ada, la preuve formelle native
- ⚙ IVVQ plans de test, validation
- ⚙ La maturité de certification à construire

Questions ?

Clemenceau Thomas

Ada and RT expert

✉ Thomas.clemenceau@thalesgroup.com

Emerdjian Thomas

Rust insider

✉ Thomas.emerdjian@thalesgroup.com

Jérôme Gurhem



HPC expert &
Cloud architect
Aneo



Florian Lemaitre



HPC expert &
Cloud architect
Aneo



Notre Load Balancer Rust en production : le Bon, la Brute, et l'Async

Notre Load Balancer Rust en production :

*le Bon, **la Brute**, et l'Async*

Jérôme Gurhem

Lead ArmoniK · Aneo

Florian Lemaitre

Lead Advanced Computing · Aneo



Aneo, en 30 secondes



Cabinet de conseil

Fondé en 2002, focus HPC dès le départ. 25 experts, dont 10 PhD.



Advanced Computing

HPC depuis 2004, HPC-cloud depuis 2013.
AI, quantique, calcul distribué.



ArmoniK, open source

8 personnes sur le produit, 3 équipes d'intégration. Apache 2.0.

22 ans de HPC, sans détour par le buzz.

ON EST QUI ?

Et qui va parler de quoi



Jérôme Gurhem

Lead ArmoniK

HPC expert & Cloud architect

PARTIE 1

Contexte, specs, implémentation.
Comment on en est arrivés là.



Florian Lemaitre

Lead Advanced Computing

HPC expert & Cloud architect

PARTIE 2

Benchs, contre-intuitions,
et comment on a tué NGINX.

ArmoniK : orchestrateur de tâches

Un moteur d'exécution distribué pour charges de calcul massivement parallèles.



Soumission, orchestration et suivi de millions de tâches.



Workers polyglottes : Python, C++, .NET, Java...



Déploiement Kubernetes, on-prem ou cloud public.

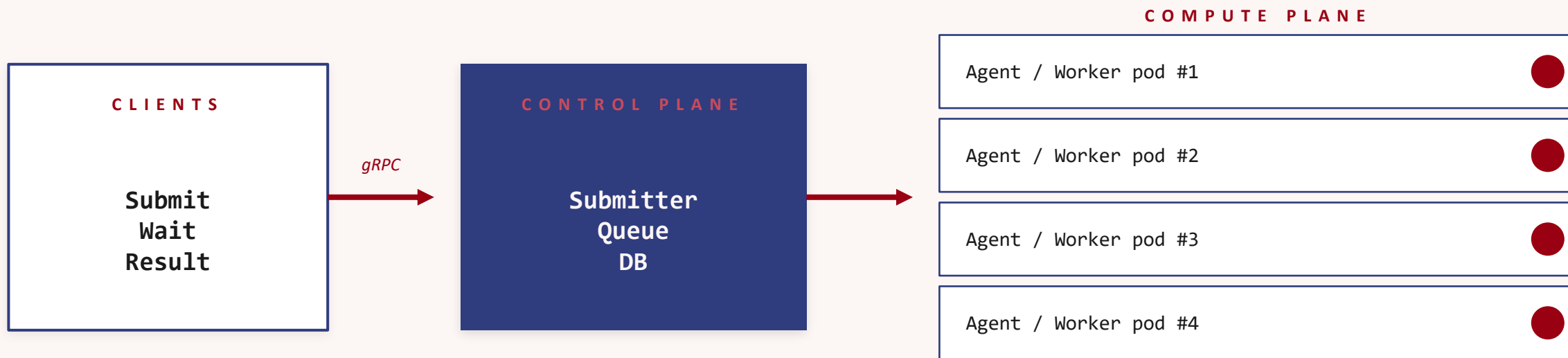


Stack open source, sous licence Apache 2.0.

CAS D'USAGE TYPIQUES

- **Pricing et risk en finance**
Simulations Monte-Carlo, grecques, VaR.
- **Simulation scientifique**
Énergie, climat, biologie, ingénierie.
- **ML & data pipelines à grande échelle**
Inférence batch, fine-tuning distribué.

ArmoniK, vue d'avion



Multi-cluster : plusieurs Compute Planes derrière un seul point d'entrée.

→ c'est là que le Load Balancer entre en scène.

Pourquoi un Load Balancer maison ?



Multi-cluster transparent

Un seul endpoint pour N clusters ArmoniK, sans logique côté client.



gRPC, pas HTTP/1.1

Streaming bidirectionnel, multiplexage HTTP/2, longues connexions.



Routage applicatif

Décisions basées sur la charge, les sessions, les versions d'API.

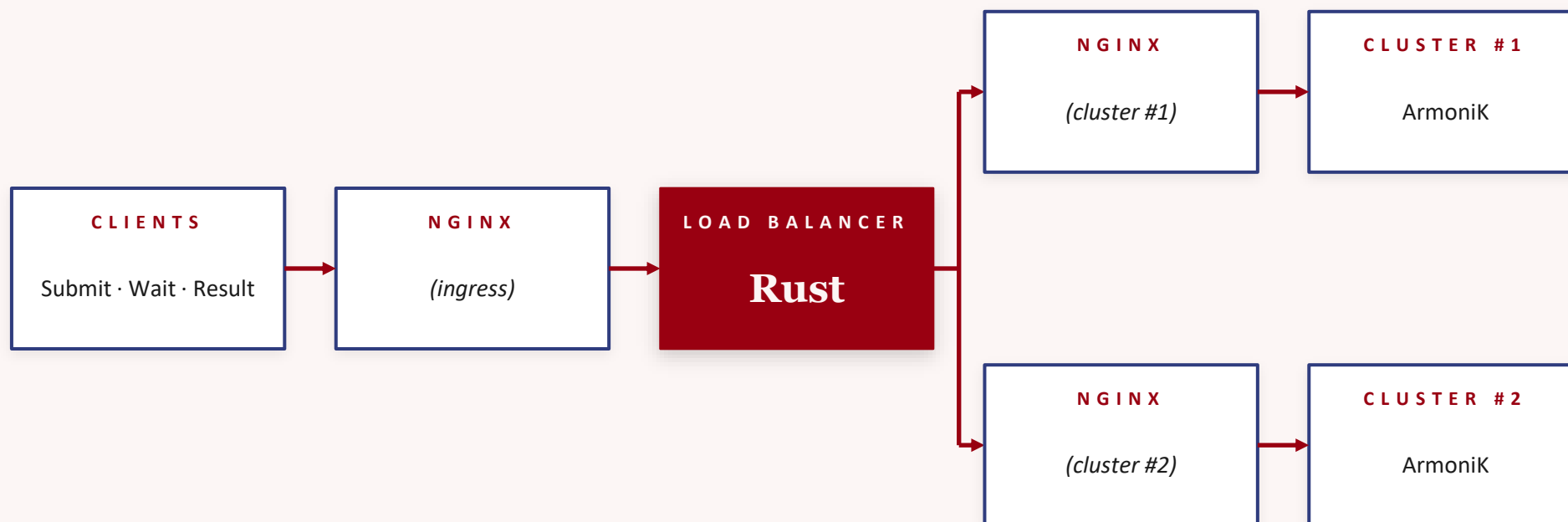


Observabilité de bout en bout

Traces distribuées, métriques par cluster, audit des routages.

Aucune brique du marché ne cochant toutes les cases.

Le LB, pièce centrale



Un endpoint unique, N clusters derrière : totalement transparent pour le client.

Voilà la cible. Reste à la construire.

Ce qu'on devait livrer

100k+

requêtes/s soutenues

stateless

pas d'état partagé

resilient

perte de LB ou de clusters

gRPC

natif, streaming inclus

0

perte de tâche au reload

k8s

déploiement natif

Pourquoi Rust ?

STABILITÉ

Sûreté mémoire, typage strict : on déploie sans transpirer.

ÉCOSYSTÈME ASYNC MATURE

tokio, tower, tonic, hyper : toute la stack gRPC est là, bien testée.

CODE RAPIDE, EMPREINTE FAIBLE

Pas de GC, footprint mémoire minimal sous charge.

PARCE QU'ON AVAIT ENVIE

Soyons honnêtes. Le choix n'avait rien d'héroïque.



La stack qu'on a empilée

tokio

Runtime async

Multi-threadé, work-stealing. La base.

hyper

HTTP/2 plomberie

Serveur et client HTTP/2 bas niveau.

tower

Middleware HTTP

Couches composables : timeouts, retries, observabilité.

tonic

gRPC idiomatique

Basé sur hyper/tower. Génération de code, codec.

prost

Codec Protobuf

Sérialisation/désérialisation des messages.

tracing

Observabilité

Spans, events, intégration distributed tracing.

Pas de framework opaque : on assemble nous-mêmes les couches qu'on veut.

Bataille de bibliothèques

TLS avec rustls + ring



Attendu

Ajouter du TLS, juste une feature flag.



Réalité

TLS en Rust = deux camps : openssl (bindings sur la lib C système) ou rustls. Mais rustls s'appuie par défaut sur aws-lc-rs... écrit en C. Pour compiler statiquement sur toutes les architectures, on aligne toutes les crates utilisant TLS sur rustls+ring.

prost & protoc



Attendu

Pure Rust pour le Protobuf.



Réalité

Mais il faut installer protoc séparément. CI, Dockerfile, machines de dev : autant d'endroits où ça coince.

SQLite n'est pas async



Attendu

Un petit stockage local pour l'état du LB.



Réalité

SQLite est synchrone par nature. Bloquer un thread tokio = bloquer tout le runtime. Il faut le pousser dans un pool blocking dédié (spawn_blocking) et accepter le coût.

LE TWIST

Trop fonctionnel.

Ça marchait.

Ça marchait même un peu trop bien.

→ *Et tout ce qui était autour s'est mis à craquer.*

Ce qu'on a mesuré

9k req/s/core

Scaling linéaire avec le nombre de cœurs.

500 μ s latence ajoutée

Hors réseau. Bruit négligeable face au temps applicatif.

20k+ connexions

Entrantes ou sortantes, sans dégradation.

100k+ streams simultanés

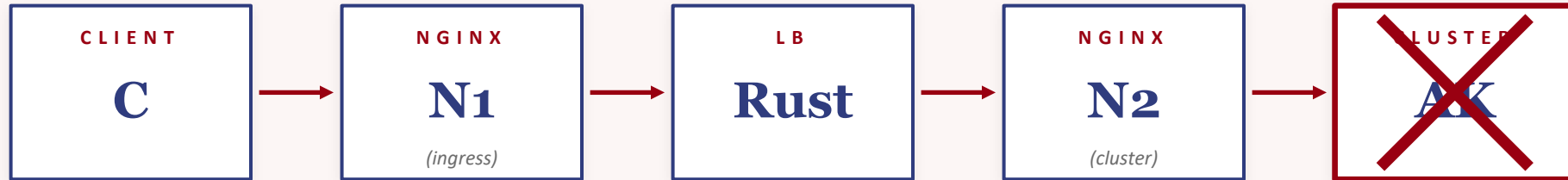
Multiplexés HTTP/2, sur une ou plusieurs connexions.

AK n'a pas le temps

Erreurs 502

CAS #1 — AK SATURE

AK n'a pas le temps



ENCHAÎNEMENT

- Le client envoie une rafale de requêtes ; N1, LB et N2 propagent.
- AK sature et ne répond plus en temps voulu.
- N2 marque AK comme unhealthy et propage l'erreur en amont.
- LB propage à son tour ; N1 finit par marquer le LB unhealthy.

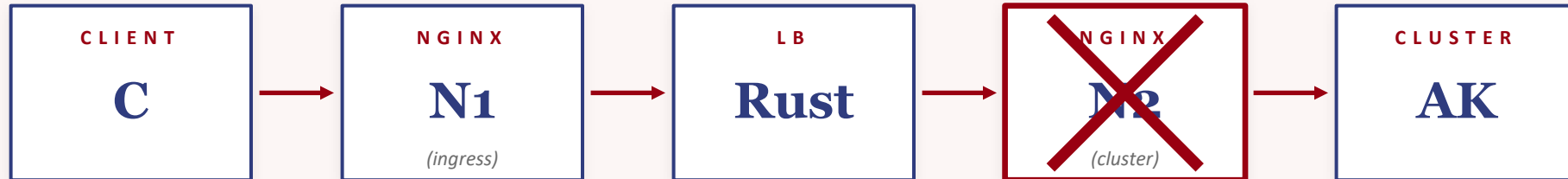
MITIGATION

- Rate limit pour laisser à AK le temps de traiter.
- Services Kubernetes headless + résolution DNS côté NGINX (≥ 1.27) pour éviter les faux positifs unhealthy.

La 1000^e requête

Erreurs 502

La 1000^e requête



ENCHAÎNEMENT

- Le client envoie des requêtes ; N1 et le LB propagent à N2.
- À la 1000^e requête de la même connexion, N2 atteint `keepalive_requests`.
- N2 ferme brutalement la connexion : la dernière requête est retentée, mais tous les streams ouverts sont coupés.

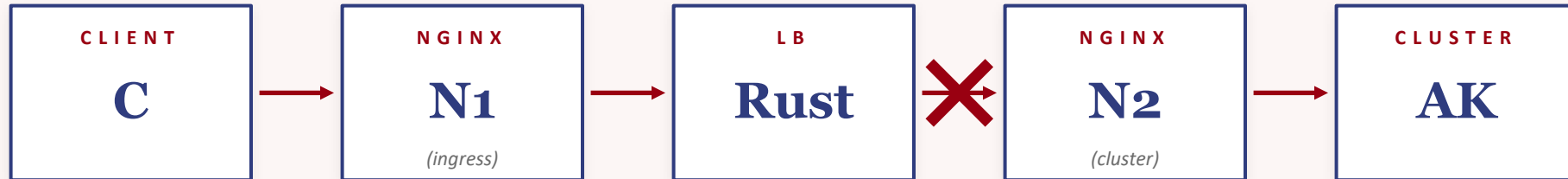
MITIGATION

- Désactiver le multiplexage des requêtes en sortie du LB.
- Compteur côté LB : fermer la connexion proprement avant d'atteindre la limite de NGINX.

Trop de connexions sortantes

Erreurs 502

Trop de connexions sortantes



ENCHAÎNEMENT

- Le client envoie une rafale ; N1 propage.
- Le LB ouvre beaucoup de connexions vers N2 pour paralléliser le traitement.
- N2 refuse les nouvelles connexions (limit_conn dépassé).
- Le LB propage l'erreur ; N1 marque le LB unhealthy.

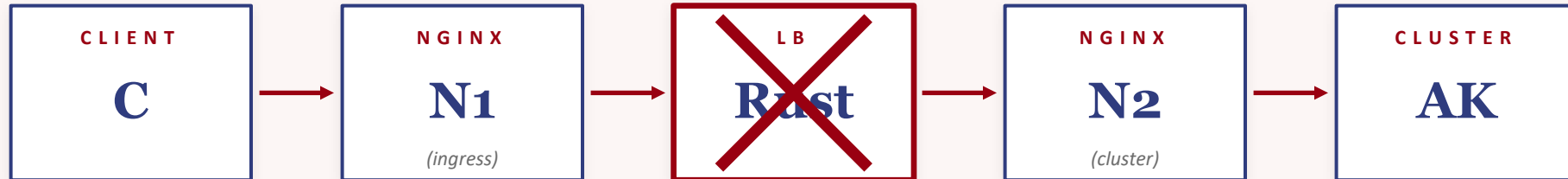
MITIGATION

- Limiter côté LB le nombre de connexions sortantes concurrentes.

Connexions SQLite par milliers

Erreurs 502

Connexions SQLite par milliers



ENCHAÎNEMENT

- Le client envoie une rafale ; N1 propage.
- Le LB traite toutes les requêtes en parallèle.
- Pool de connexions SQLite sans limite : 1 connexion = 1 thread.
- Le nombre de threads explose, le process LB sature ; N1 marque le LB unhealthy.

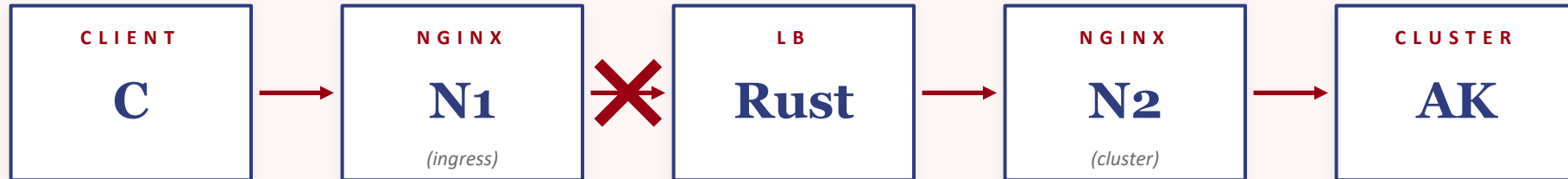
MITIGATION

- Déplacer les connexions SQLite dans une thread pool rayon avec du thread-local.
- Une connexion SQLite par thread, point. Le pool n'est plus illimité.

Une connexion par requête

Erreurs 502

Une connexion par requête



ENCHAÎNEMENT

- Le client envoie une rafale ; N1 ouvre une nouvelle connexion par requête.
- Le LB doit gérer toutes ces nouvelles connexions en parallèle.
- Le LB sature au handshake (surtout avec peu de CPU); N1 marque le LB unhealthy.

MITIGATION

- Configurer NGINX upstream avec keepalive pour réutiliser les connexions sortantes.

Contre-intuitions de perf

ON PENSAIT

Plus on est async, mieux c'est.

EN VRAI

Sans back-pressure, l'async amplifie la surcharge en aval : il faut savoir freiner.

ON PENSAIT

Un proxy léger doit être transparent.

EN VRAI

Un proxy **trop** transparent transmet aussi les rafales : il devient un amplificateur.

ON PENSAIT

Nginx sait encaisser des charges énormes, et on peut y aller franco.

EN VRAI

Nginx a plein de limites qui lui permettent en temps normal de ne pas saturer, mais nécessite de jouer selon ses règles pour éviter de tout casser.

T A K E A W A Y S

Leçons retenues

01**Async sans back-pressure = bombe à retardement***Le runtime ne dira jamais non pour vous.***02****La perf est un système, pas un composant***Optimiser une brique ne fait que déplacer le goulot si le reste du système ne suit pas.***03****Maîtrisez chaque couche et leurs limites***Sinon vous accuserez le mauvais coupable.***04****Rust ne vous protège pas du design***Le compilateur attrape les data races. Pas les architectures pressées.*

EN RÉSUMÉ

Rust nous a donné un LB rapide. Le vrai travail a commencé après.



ArmoniK est open source

github.com/aneconsulting/ArmoniK

M E R C I

Questions ?

C O N T A C T

Jérôme Gurhem

jgurhem@aneos.fr

Florian Lemaitre

flemaitre@aneos.fr

aneos.eu · github.com/aneosconsulting

Gabin Fourcalt

Optimisation des systèmes logiciels complexes via une migration vers Rust et WebAssembly



Ingénieur Architecte des
systèmes et des logiciels
Thales Alenia Space



LITEINTHEBOX

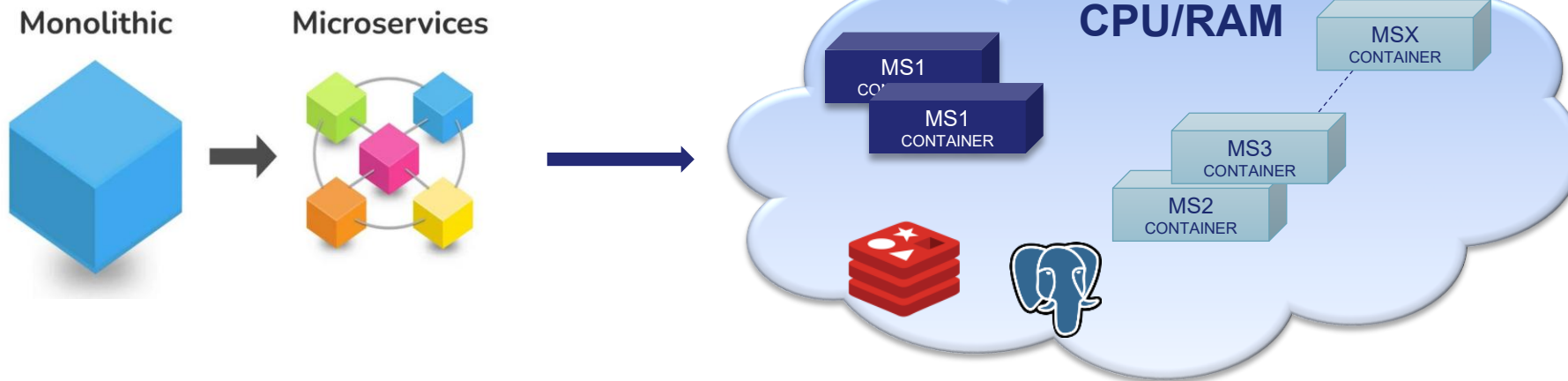
RUST PARIS 2026



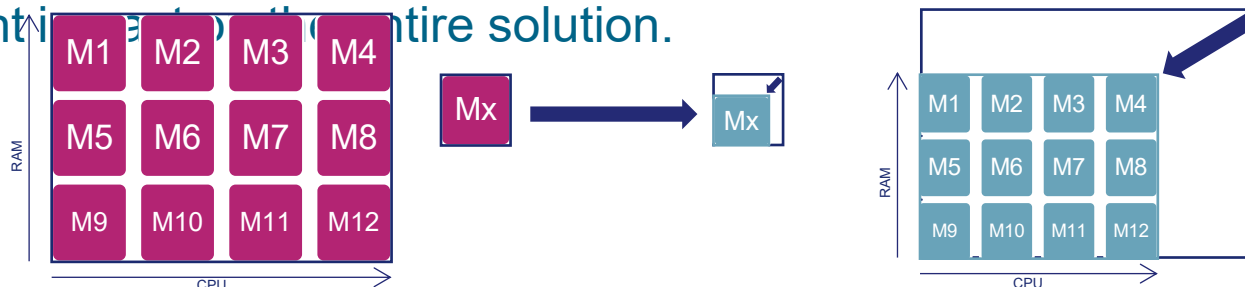
Challenge to address within new microservice architecture

/// Software solutions are becoming larger with each passing year. With the adoption of cloud technologies, these solutions are split into multiple small microservices, sometimes deployed in duplicate or more to ensure redundancy.

/// Space software solutions often use complex algorithms that usually consume lots of CPU and RAM



/// Even a small reduction in resource consumption (CPU and RAM) of each microservice could have a significant impact on the entire solution.



IDEA :
Reduce code footprint
Reduce container footprint

ESTIMATED ENVIRONMENTAL BENEFITS

/// Reduce platform costs everywhere

- On production resources, cloud or on premises
- On validation and integration resources
- On developer and continuous integrations resources

/// Act on code, act on target

- Some programming languages are more resource-intensive and slower, we propose to move algorithms implementations from Python (slow and resource-intensive) to Rust
- The container technology could influence the size and the speed execution of micro-services, we propose to package some micro services with modern solutions that accelerate speed and reduce the size of the executable
- The 2 previous options will be examined individually, and then their findings will be merged together for the purpose of benefiting from both



What we did

/// Find the appropriate algorithm(s) for the study

- ! Not too small to be representative & Not too big to avoid spending all our time in code migration

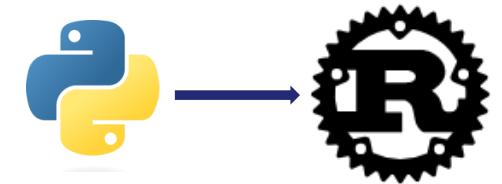
/// Change code packaging

- ! Move Python code from Docker to Web Assembly packaging
- ! Run Web Assembly container in a cluster
- ! Measure performance gain between each packaging method



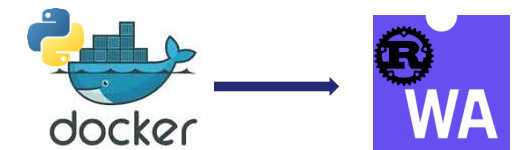
/// Code migration

- ! Rewrite from Python to Rust
- ! Measure performance gain between each programming language



/// Merge previous steps and build a report to generalize the expected gains

- ! Package Rust code in Web Assembly
- ! Run Web Assembly container in a cluster
- ! Measure performance gain



Explorations & outcomes

PROPRIETARY INFORMATION

Ce document ne peut être reproduit, modifié, adapté, publié, traduit d'une quelconque façon en tout ou partie, ni divulgué à un tiers sans l'accord préalable et écrit de Thales Alenia Space.
© 2026 Thales Alenia Space Tous droits réservés

THALES ALENIA SPACE LIMITED DISTRIBUTION

Explorations – General

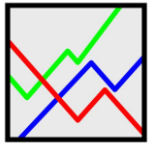
/// Methodology

/ Identify two use cases

- Use Case 1: **Orbit Calculation Interpolation**
- Use Case 2: **Signal Processing**

/ Given an isolated algorithm written in python :

- we transform it as a web server to make request on it and dockerized it.
- Convert it in Rust and make as a web server and dockerized it
- Build the Rust application as a WebAssembly using the Spin SDK framework



Explorations – General

/// Methodology to transform from python to rust:

- / Analyze Python code and rewrite "line by line" to Rust.
- / Ensure tests produce the same result.
- / Benchmarking.

/// Methodology to transform from rust to spin app (wasm):

- / Use the Spin SDK Fermyon Framework to build, deploy and run webassembly applications
- / Implement the Spin HTTP component to expose the service
- / Build the spin application
- / Prepare the environment with SpinKube to deploy the spin app and run it

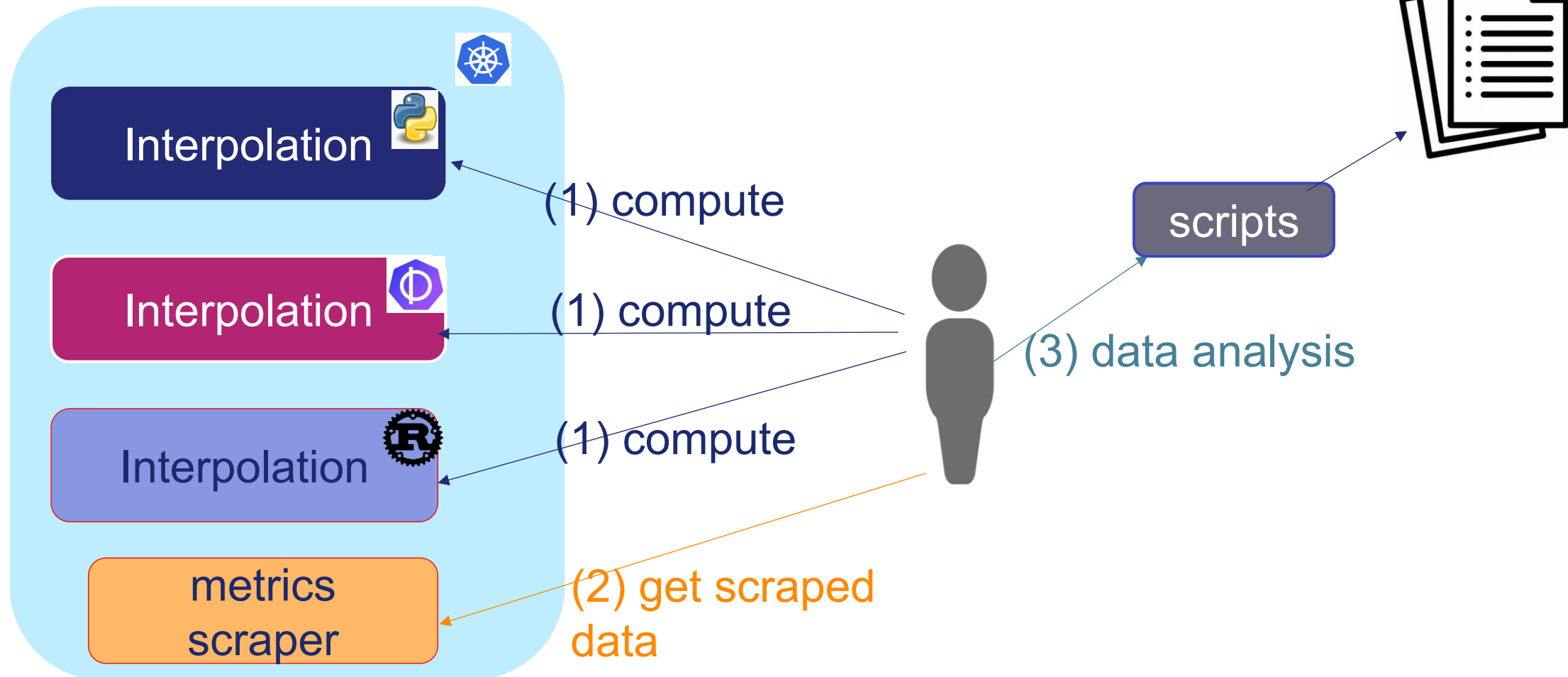
Exploration - General

/// A unique Execution environment, where we are able to run docker images and webassembly

- / K3d: a lightweight kubernetes distribution that runs on Docker
- / Install SpinKube, an open-source Kubernetes-native platform that simplifies the deployment and execution of WebAssembly (Wasm) workloads by integrating components like the Runtime Class Manager, Containerd Shim Spin, and Spin Operator to efficiently schedule and run Spin application
- / Add metrics server in kubernetes cluster and configure it to scrap cpu/memory metrics every 10s.
- / Prepare the deployment files for each applications python, Rust, wasm and deploy them on the cluster.
- / Collect metrics data :
 - Launch all components on kubernetes and request them using curl command
 - During this stage, memory/cpu usage are scrapped automatically inside Kubernetes
 - Then we request metrics API server and retrieve data and format them as a csv file to be interpret by gnuplot tools.

Explorations – General

/// How to run and capture metrics

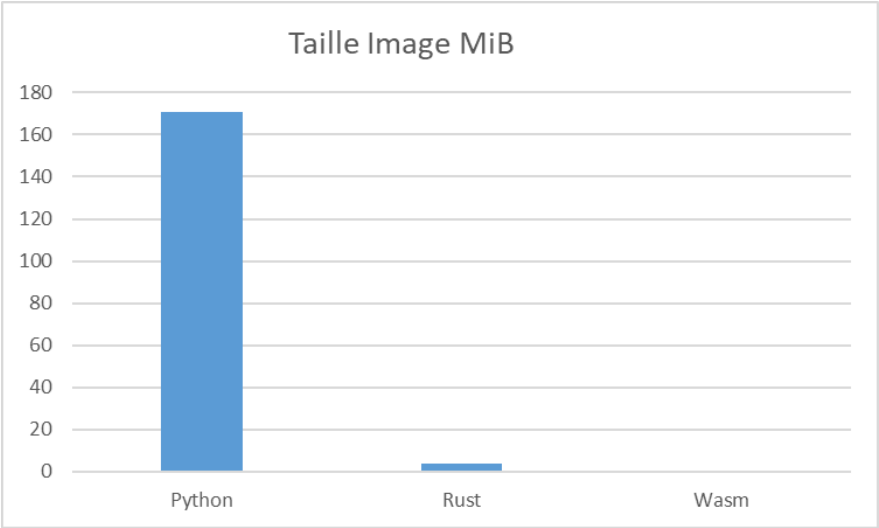


Use Case 1 – Metrics after rewrite

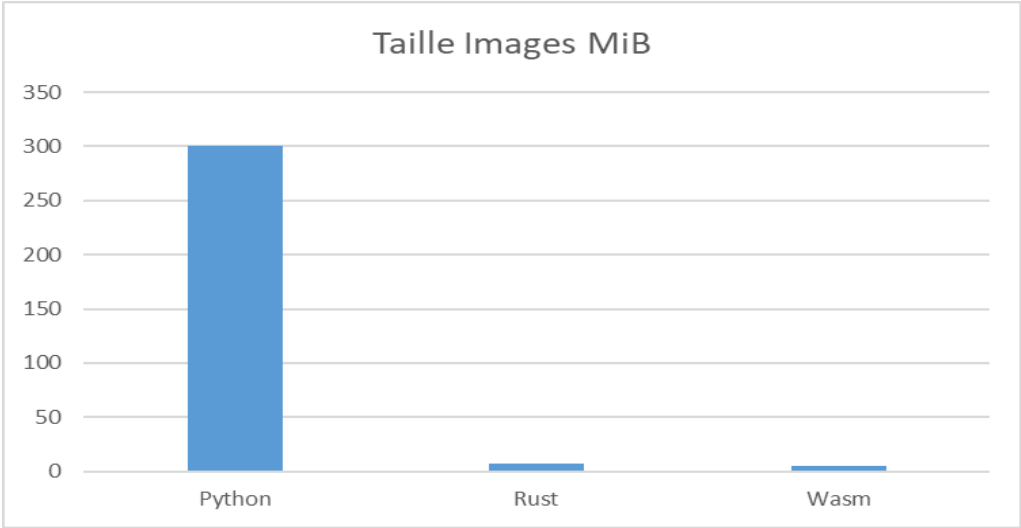
/// First metrics

Contexte de mesure	Python (ms)	Rust (lib) (ms)	Rust (exe)(ms)	facteur
-10; 10	15.2	0.052	0.048	~292
-20;20	15.4	0.050	0.048	~292
-100;100	15.7	0.052	0.051	~292
-1000;1000	16.7	0.084	0.084	~198
-10_000; 10_000	17.3	0.390	0.327	~52
-50_000; 50_000	24.4	1.985	1.878	~13
-100 000; 100 000	36.69	3.006	3.030	~12
-500 000; 500 000	113.52	15.66	16.167	~7
-1 000 000; 1 000 000	209.73	31.05	32.029	6.7
-5 000 000; 5 000 000	1483.1	184.34	170.88	8.6
-10 000 000; 10 000 000	2960	493.75	382.18	7.7

Use case 1



Use case 2



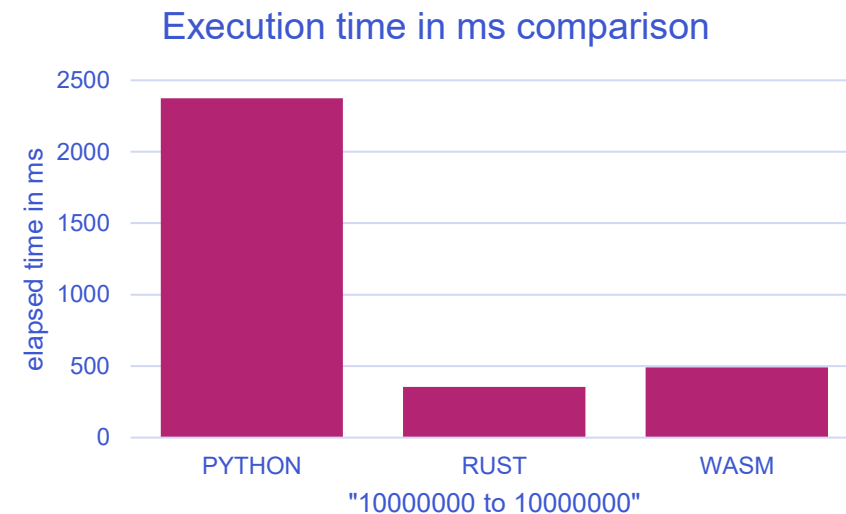
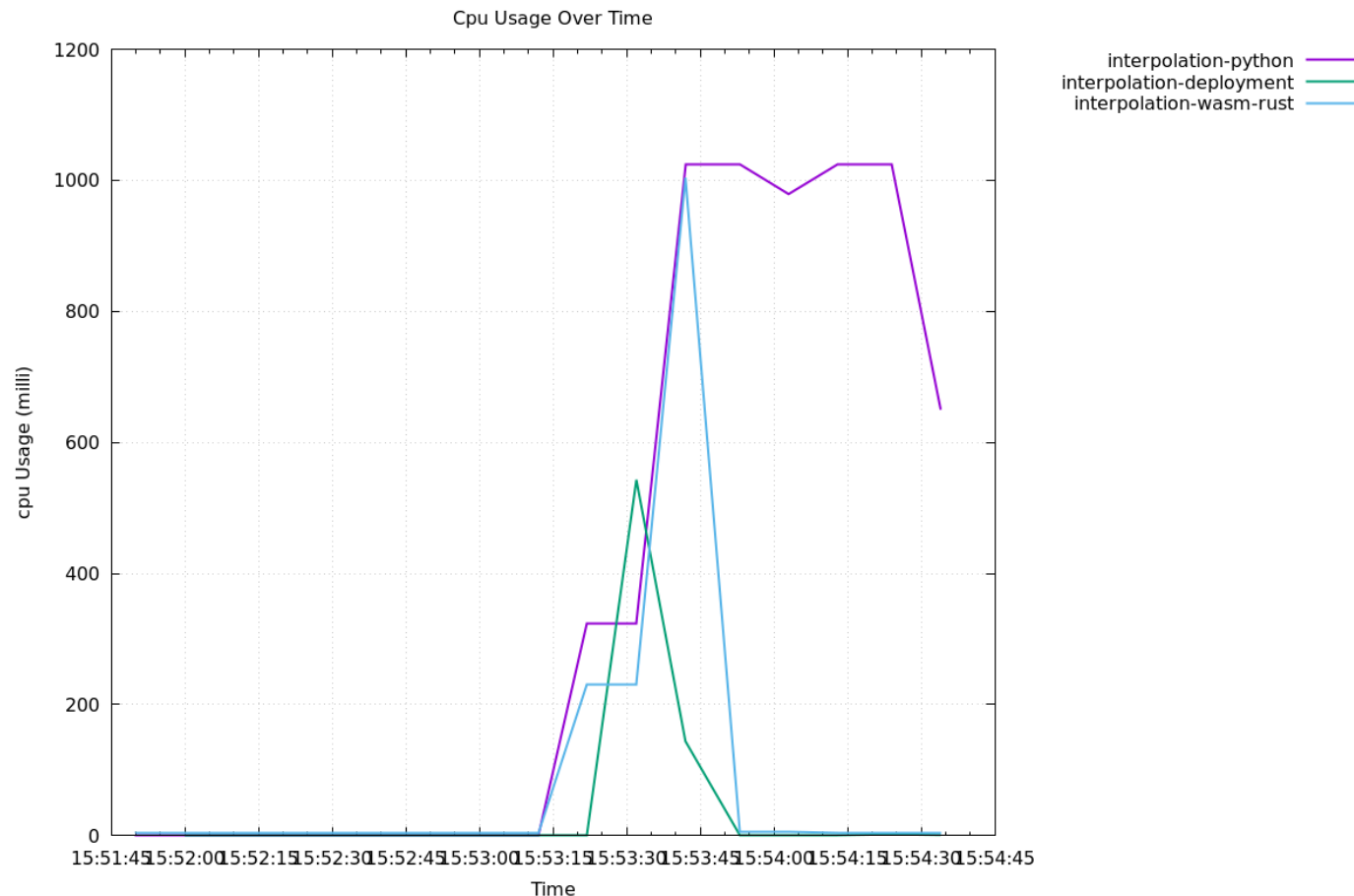
Python images are significantly larger, likely due to dependencies, runtime, and interpreter overhead.

Rust produces a much smaller image, as it compiles to native machine code and does not require an interpreter.

Wasm is the most lightweight option, optimizing further over Rust, possibly due to minimal runtime and efficient binary representation.

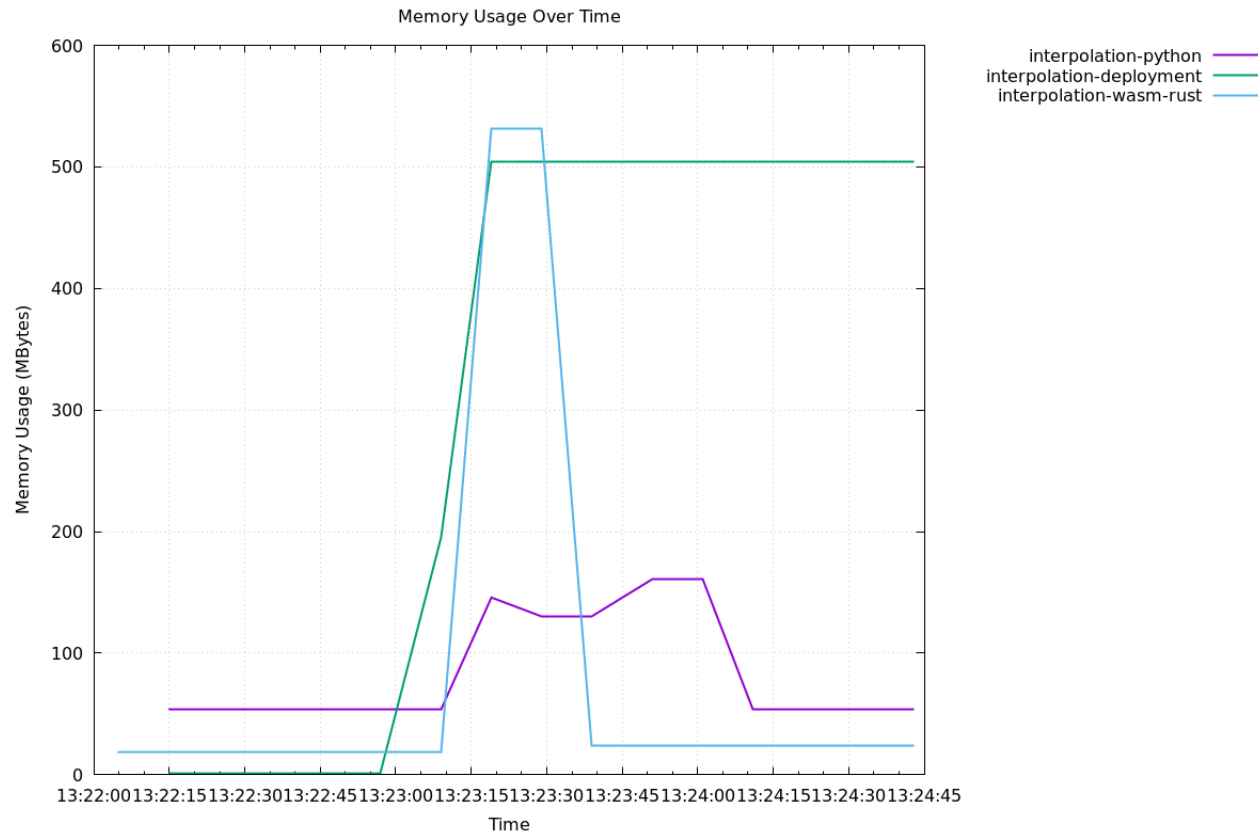
Wasm reduces the size by ~34.4% compared to Docker image Rust.

Explorations – Use Case 1 - CPU



- **Python has the worst performance**, likely due to interpreter overhead and inefficient execution.
- **Wasm-Rust is slightly more CPU-intensive than Docker-Rust**, possibly due to WebAssembly runtime overhead, but still much better than Python.

Use case 1 – Memory



/// Python Pod: Stable at ~100MB.

/// Rust Pod: Spikes to 500MB (400% more than Python).

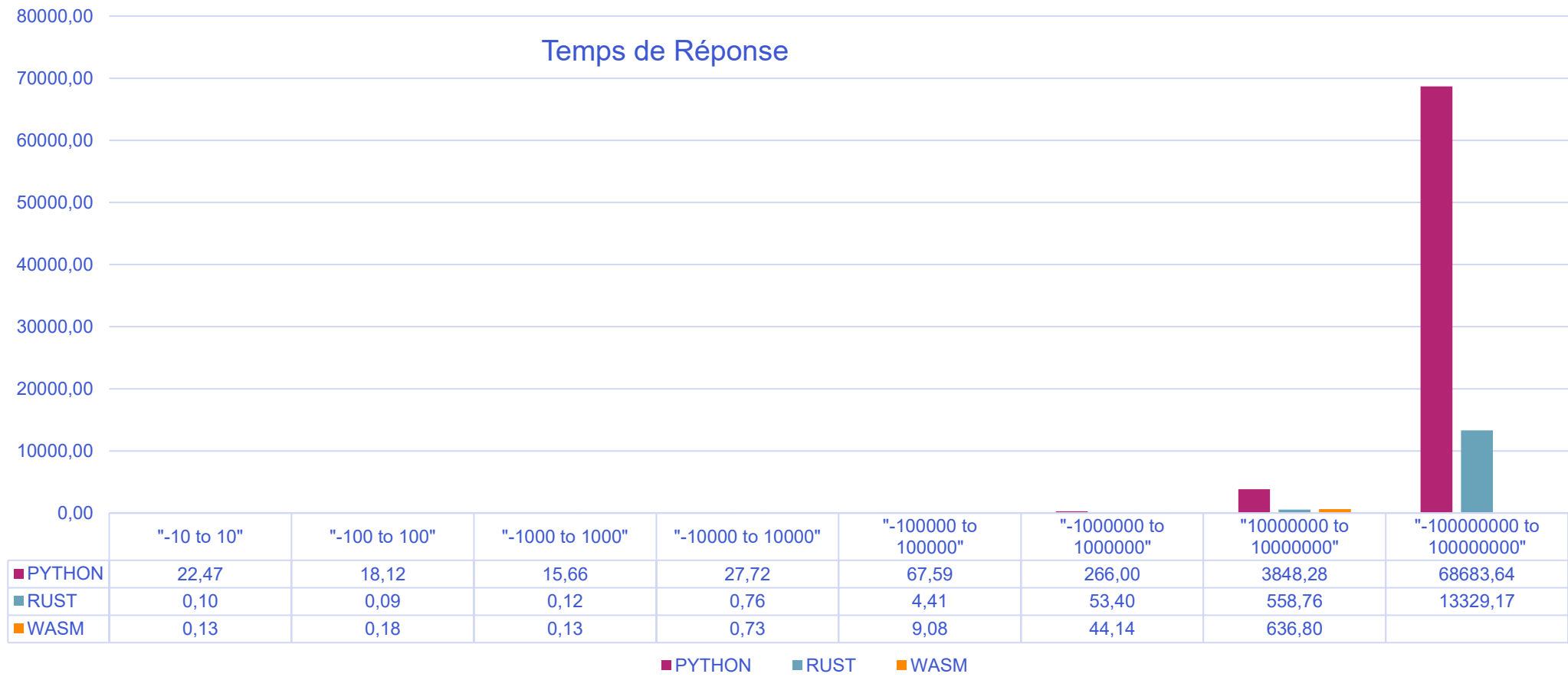
/// WASM Pod: Spikes to 500MB, then drops (same as Rust).

=>

/// Rust & WASM consume 5× more memory than Python at peak.

/// WASM quickly releases memory, while Rust holds it steady.

/// USE CASE 1 – Execution time

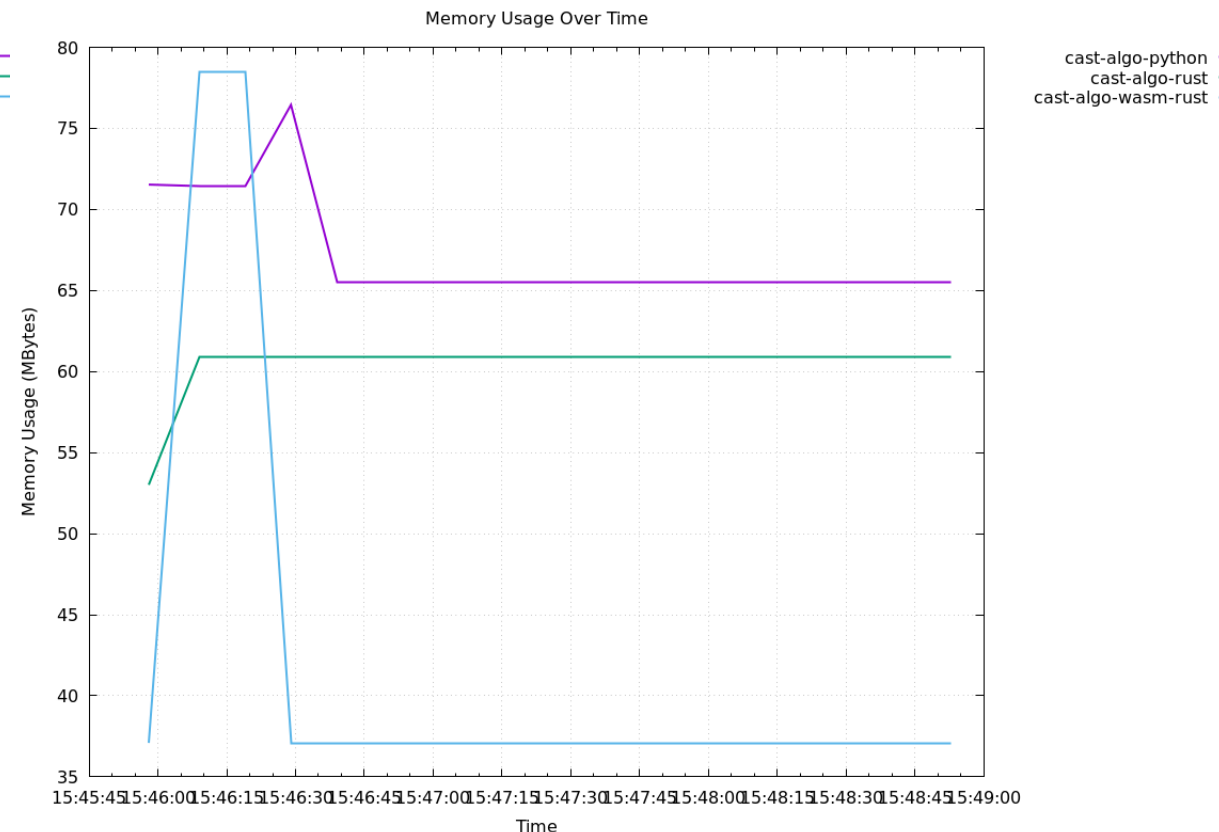
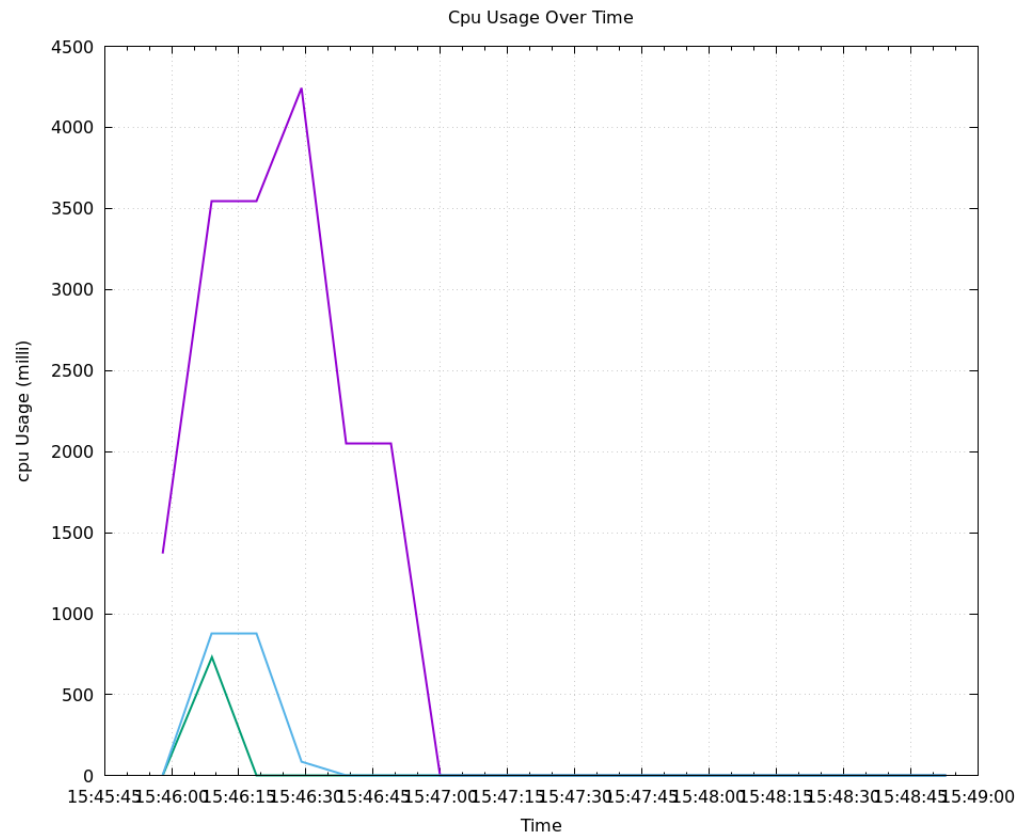


Use Case 2 – Metrics after rewrite

//// First metrics

Test réalisé	Python calcul (Ms)	Rust calcul (Ms)	Facteur
Pll_dll_loop (re-sync)	1285.76	418.99	~3
Pll_dll_loop(not_sync)	714.52	14.45	~49

Explorations – Use Case 2

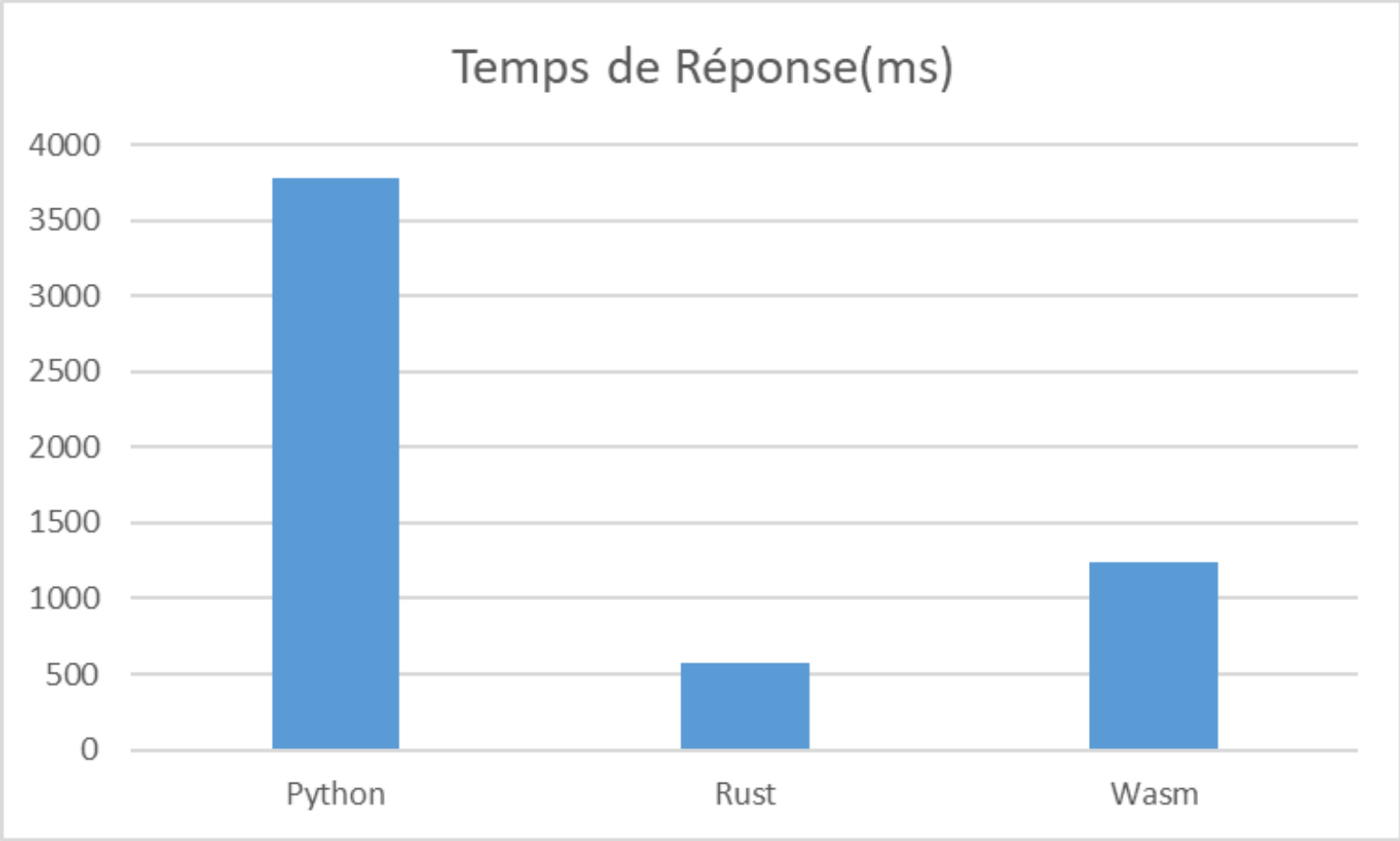


/// Docker Rust: Offers the best performance with the lowest CPU and memory usage, resulting in the shortest execution time.

/// Wasm-Rust: Provides a balance between performance and portability, with moderate CPU and memory usage, and slightly longer execution times compared to native Rust.

/// Python: Exhibits the highest resource consumption and the longest execution time among the three implementations.

/// Use Case 2 – Execution time



CONCLUSION

- The different tests and collected metrics show that:
 - There is a significant gain between Python and Rust for time-sensitive projects. => There is a real benefit to writing our algorithms in Rust.
 - Instead of lightweight applications and the isolated environment provided by WASM, we didn't notice a significant difference in resource gains. Therefore, we should carefully assess when to choose Docker images for Rust applications and when to choose WebAssembly.
- **Adopt WASM now** if you are focusing on edge computing applications, need a secure and isolated environment, or if you can work within its current limitations (limited system access, threading).
- **Wait for WASM to mature** if your application requires complex system-level access, heavy multithreading, or relies on libraries and frameworks not yet fully supported in WASM.
- We have some limitations with WebAssembly :
 - Lack of support for many libraries (python libraries)
 - Memory allocation limitations: WebAssembly operates in a sandboxed environment, which imposes restrictions on direct memory access. The size of the memory allocated for WASM modules is generally limited
 - Lack of multithreading support

Questions

Date : 08/06/2026

Ref : xxxx

Ref Modele : 83230347-COM-TAS-FR-012

PROPRIETARY INFORMATION

Ce document ne peut être reproduit, modifié, adapté, publié, traduit d'une quelconque façon en tout ou partie, ni divulgué à un tiers sans l'accord préalable et écrit de Thales Alenia Space.

© 2026 Thales Alenia Space Tous droits réservés

THALES ALENIA SPACE LIMITED DISTRIBUTION

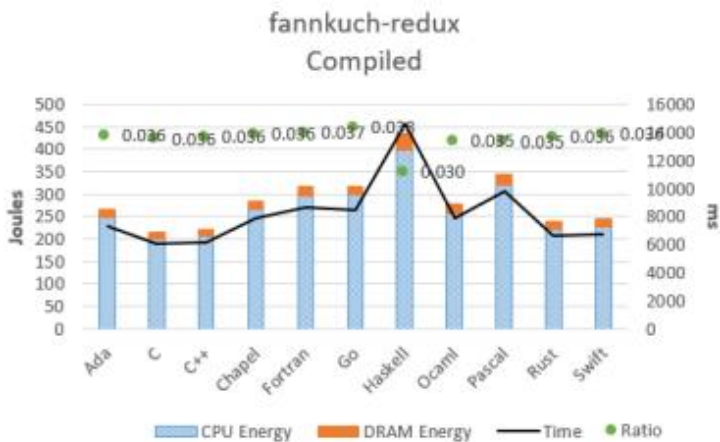
FROM PYTHON TO RUST - DERISKING

Energy Efficiency across Programming Languages

How Do Energy, Time, and Memory Relate?

Table 1. CLBG corpus of programs.

Benchmark	Description	Input
n-body	Double precision N-body simulation	50M
fannkuch-redux	Indexed access to tiny integer sequence	12
spectral-norm	Eigenvalue using the power method	5,500
mandelbrot	Generate Mandelbrot set portable bitmap file	16,000
pidigits	Streaming arbitrary precision arithmetic	10,000
regex-redux	Match DNA 8mers and substitute magic patterns	fasta output
fasta	Generate and write random DNA sequences	25M
k-nucleotide	Hashtable update and k-nucleotide strings	fasta output
reverse-complement	Read DNA sequences, write their reverse-complement	fasta output
binary-trees	Allocate, traverse and deallocate many binary trees	21
chameneos-redux	Symmetrical thread rendezvous requests	6M
meteor-contest	Search for solutions to shape packing puzzle	2,098
thread-ring	Switch from thread to thread passing one token	50M



	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

Rust Paris 2026

Pause

Reprise à 16h



Sponsors Gold

CÉNOTÉLIE

TITAGONE
by OCaml PRO

THALES
Building a future we can all trust

Sponsors Silver

AdaCore

oxid[os]

Wyliodrin

Rust Paris 2026

Orson Pessin

CEO

Wylidrin



Sponsors Gold

CÉNOTÉLIE

TITAGONE
by OCaml PRO

THALES
Building a future we can all trust

Sponsors Silver

AdaCore

oxid[os]

Wylidrin

Pierre-François Gimenez



Chercheur
Inria

Inria

Projet de recherche Fos-R:

Rust for Research: Feedback from the Fos-R Project



Rust Paris 2026

Pierre-François Gimenez, Inria

A quick presentation

Who am I?

- ▶ Researcher at Inria, a national research institute, in the PIRAT team
- ▶ PhD in machine learning in 2018
- ▶ Research in AI for cybersecurity since then

Research topics of my team

- ▶ Malware analysis (Windows and Android)
- ▶ Intrusion detection (EDR, NDR)
- ▶ Automatic vulnerable infrastructure deployment (digital twin, honeynets and CTF)



Software in research

Relation between software development and research

- ▶ Fundamental research with low TRL (1 to 4)
- ▶ We develop prototypes to validate ideas
- ▶ Software are almost always open source to allow reproducibility
- ▶ The output are the ideas and the experimental evaluation, the software implementation is a secondary deliverable
- ▶ The vast majority of our software has no update after initial release

The current practices

- ▶ Python is by far the most used language
- ▶ We also used a few more specific languages: R, OCaml, Rocq (Coq), etc.



Research project: Fos-R

Scientific context: data is key

- ▶ We need data to train ML/DL models
- ▶ We need data to evaluate our tools
- ▶ Background traffic for more realistic honeypot and CTF

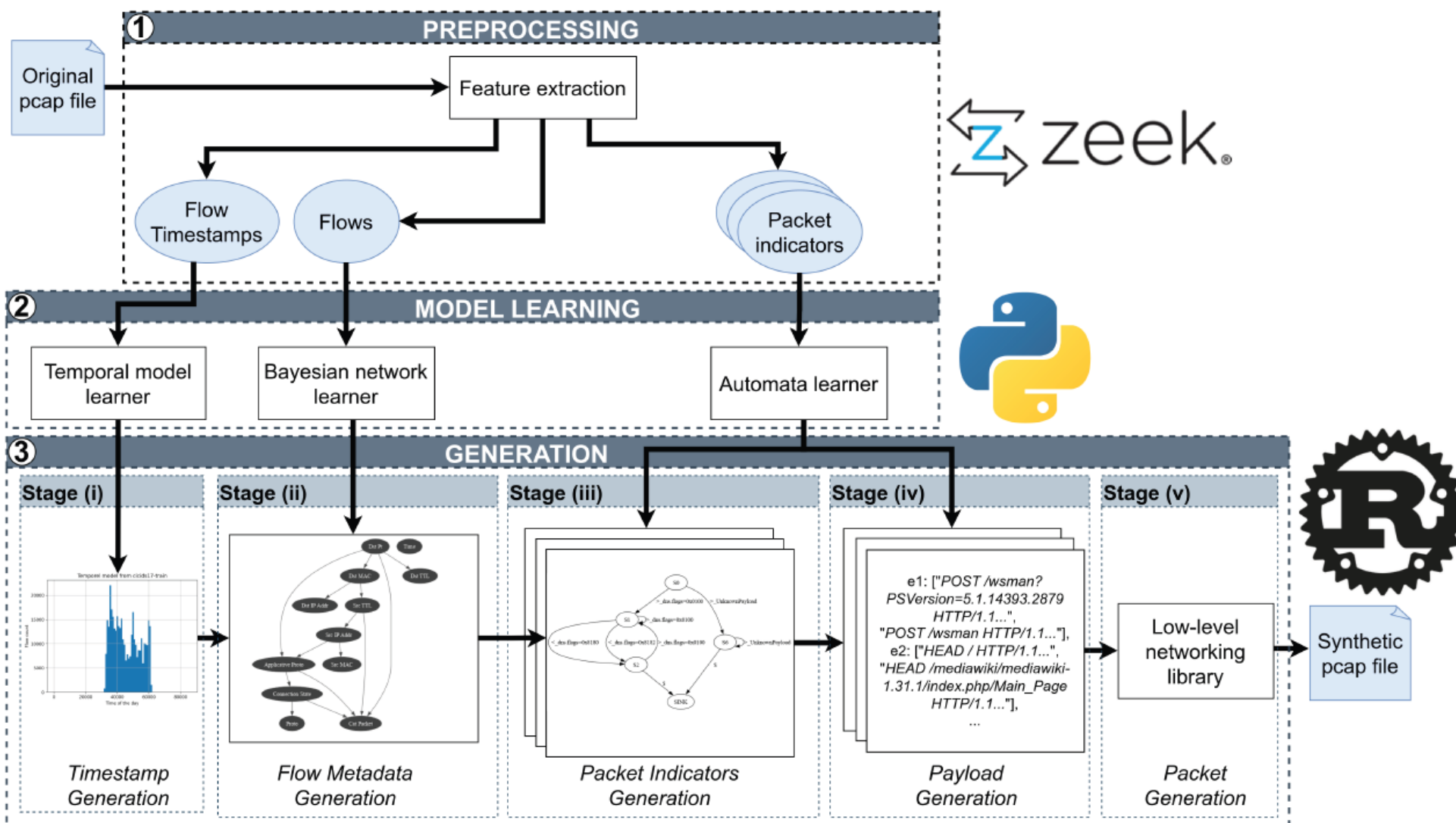
Sources of data

- ▶ We rely on public datasets created by research institutions
- ▶ They have many issues: obsolescence, errors, lack of realism, etc.
- ▶ It makes evaluation difficult: how to be sure our ideas are the correct ones?

Fos-R

I develop Fos-R, a software that generate network traffic (pcap) files with machine learning models, and inject it in a network

Fos-R: inner working overview



Fos-R specification

Functional specification

- ▶ Traffic generation with machine learning models
- ▶ Low-level networking
- ▶ Kernel-based routing
- ▶ User interface

Non-functional specification

- ▶ High performances
- ▶ Reproducibility
- ▶ Portability (Linux and Windows, ARM and x86)
- ▶ Easy to test without downloading anything





1. Rust popularity 😞

Why is it important?

All people working on a project must master its language

In research

- ▶ Many PhD students and engineers do not know Rust
- ▶ Most of our team have CDD (PhD students, most engineers, etc.), so many people work on a project
- ▶ More and more are learning it, but they are still in the minority

For Fos-R

- ▶ I am the main developer and I reimplement the work of PhD students
- ▶ I teach Rust at CentraleSupélec since 2020
- ▶ I can propose students to work on Fos-R as part of their teaching
- ▶ Luckily we have a Rust senior research engineer to help us!



2. Prototype-friendly ☹️

Why is it important?

Agile method require quick iterations

In research

- ▶ In research we iterate a lot on our ideas (typically one iteration per week)
- ▶ Hackability is more important than maintainability

For Fos-R

- ▶ Prototyping is mostly done in Python
- ▶ A modular architecture with traits allows for easier change of individual component

3. Library ecosystem 😊

Why is it important?

Library allows for faster development

In research

It highly depends on the types of library:

- ▶ There is much fewer ML library than in Python
- ▶ Libraries are less mature: their API are more unstable and they do not have as many features
- ▶ But there are some great libraries (e.g., GUI library), and even a full Rust SMT solver

For Fos-R

- ▶ Model learning is written in Python
- ▶ Inference is written in Rust
- ▶ I had to reimplement several machine learning algorithms manually
- ▶ I had to hack some libraries to use them



4. Dependency management 😊

Why is it important?

Anyone should be able to easily compile our code

In research

- ▶ Python is a mess, and code breaks quickly due to non-pinned dependency, breaking changes somewhere, and no unified project management system
- ▶ It makes projects very difficult to reproduce, which hamper comparison between methods and overall scientific progress

For Fos-R

- ▶ Cargo is a fantastic tool, crates.io is very convenient
- ▶ We know it won't break randomly in two months
- ▶ Features with conditional dependency is very useful as well



5. Cross-compilation ☺

Why is it important?

Cross-compilation allows to distribute easily binaries for multiple platforms

In research

We generally do not require cross-compilation

For Fos-R

- ▶ Fos-R requires cross-compilation to be deployed on Linux and Windows honeypots
- ▶ We cross-compile to x86 and ARM platform (e.g., Raspberry Pi)
- ▶ We also export to WASM for easy demo without downloading anything
- ▶ WASM export requires conditional compilation however



6. Performances 😊

Why is it important?

Nobody likes to wait, and RAM is too expensive to waste :)

In research

Prototypes are not optimized: as long as it takes a reasonable time, it is enough

For Fos-R

- ▶ State-of-the-art methods are really slow, so a fast generation is a way to differentiate
- ▶ We changed the default Linux allocator for a $5\times$ improvement
- ▶ This is in strong contrast with Python



7. System programming language 😊

Why is it important?

Some projects require low level access to OS features

In research

Most research projects that require system programming use C or C++

For Fos-R

- ▶ Rust allows us to use raw sockets, to specify allocators, to use eBPF, etc.
- ▶ We need to use conditional compilation to disable some low-level features for the WASM export

8. Safety 😊

Why is it important?

Nobody wants to ship vulnerable software

In research

The software never leaves the lab: there is little to no security concern

For Fos-R

- ▶ Vulnerabilities are not considered
- ▶ But the borrow checker allows to identifies many issues during compilation
- ▶ If it compiles, there is a good chance it's correct
- ▶ Even if compilation is slower, iterations are faster
- ▶ Fearless concurrency is real: with the proper architecture, making Fos-R concurrent was effortless



Assessment for Fos-R

Functional specification

- ▶ Traffic generation with machine learning models  (manual reimplementation)
- ▶ Low-level networking  (libpnet)
- ▶ Kernel-based routing  (eBPF with Aya)
- ▶ User interface  (egui)

Non-functional specification

- ▶ High performances  (about 6.8 GB/s)
- ▶ Reproducibility 
- ▶ Portability (Linux and Windows, ARM and x86)  (6 exports)
- ▶ Easy to test without downloading anything  (WASM export)



Conclusion

Rust for Fos-R

- ▶ Overall, a very positive experience using Rust for Fos-R
- ▶ The biggest issue was a lack of ML models
- ▶ Part of the project is still in Python

Rust for research

- ▶ The drawbacks probably outweigh the benefits
- ▶ Some drawbacks may disappear (language popularity, ecosystem maturity)
- ▶ Lack of hackability is the most fundamental issue
- ▶ The hegemony of Python makes it easier to build on each other's work

Rust is helpful for my work, but I do not expect it to become popular within the computer science scientific community

Rolland Dudemaine



Directeur Solutions Engineering
EMEA/APAC
TrustInSoft

TRUST  SOFT

What's left to find in Rust code?

What's left to find in Rust code?

Rust Paris, 2026-06-09

Rolland Dudemaine, Director, Solutions Engineering

I'm a convert.

```
1 fn find(p: &[i32], v: i32) -> i32 {
2     for n in 1..=10 {
3         if p[n] == v {
4             return n as i32;
5         }
6     }
7     -1
8 }
9 pub fn badfind() {
10     let array2: [i32; 10] = [0; 10];
11     find(p: &array2, v: 1);
12 }
```

What does the above code do?

(read the title again)



Rust: moving industry forward!



- 11-years old and counting
- Took lessons from many other languages
- Very modular
- Generally harder to write bad code
 - Borrow checker is a great example
 - (almost) No Undefined Behavior
- Integrated test framework
- Integrated documentation
- Qualified tools available



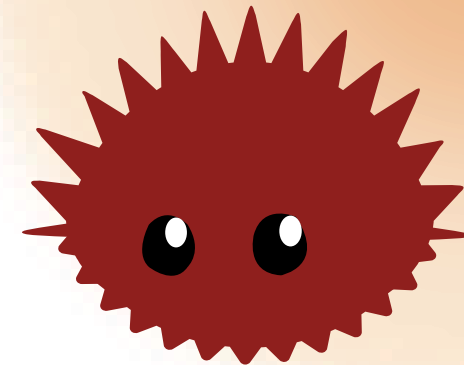
It builds. Then panics.

```
1 fn find(p: &[i32], v: i32) -> i32 {
2     for n in 1..=10 {
3         if p[n] == v {
4             return n as i32;
5         }
6     }
7     -1
8 }
9 pub fn badfind() {
10     let array2: [i32; 10] = [0; 10];
11     find(p: &array2, v: 1);
12 }
```

```
rolland@pasiphae:~/playground/rustquestions$ rustc q1.rs
rolland@pasiphae:~/playground/rustquestions$ ./q1

thread 'main' panicked at q1.rs:3:12:
index out of bounds: the len is 10 but the index is 10
note: run with 'RUST_BACKTRACE=1' environment variable to display a backtrace
rolland@pasiphae:~/playground/rustquestions$
```

Yeah, no warning. And it's safe.



Rust: Corro the unsafe rusturchin

- unsafe: the power to shoot yourself in the foot:
 - Useful for FFI, HW access/low-level device drivers
 - Voluntary decision to tell the compiler to silence
 - Human is expected to fulfill contract
- FFI: when not all code is built with Rust (always?)
 - Very few tools cover Rust $\leftrightarrow$ C/C++ interoperability
 - Manual review is impractical
 - Integrated verification is tough
 - Rust $\leftrightarrow$ Rust interop happens (e.g. shared libs)

What do you still need to search when using Rust?

```
unsafe extern "C" fn c_div(a:i32,b:i32) -> i32;

fn main() {
    // Safe Rust
    let mut num: i32 = -1;
    let arr = [1, 2, 3];
    let ptr: *mut i32 = &mut num;

    num += arr[0];

    unsafe {
        // unsafe code
        *ptr += 1;

        // call to external code
        num = c_div(num, 0);
    }
    println("Result: {}", num);
}
```

```
int c_div(int a, int b)
{
    // potential division by zero!
    return a / b;
}
```

1

safe Rust

- ✓ Crash freedom: avoid panic

2

unsafe Rust

- ✓ Crash freedom: avoid panic
- ✓ Comply with Unsafe Rust coding rules: avoid Undefined behaviors

3

foreign calls (using **extern "C"**)

The Rust code interfacing with other languages (e.g. C) is unsafe.

- ✓ Crash freedom: avoid panic
- ✓ Compliance with Unsafe Rust coding rules: avoid Undefined behaviors

4

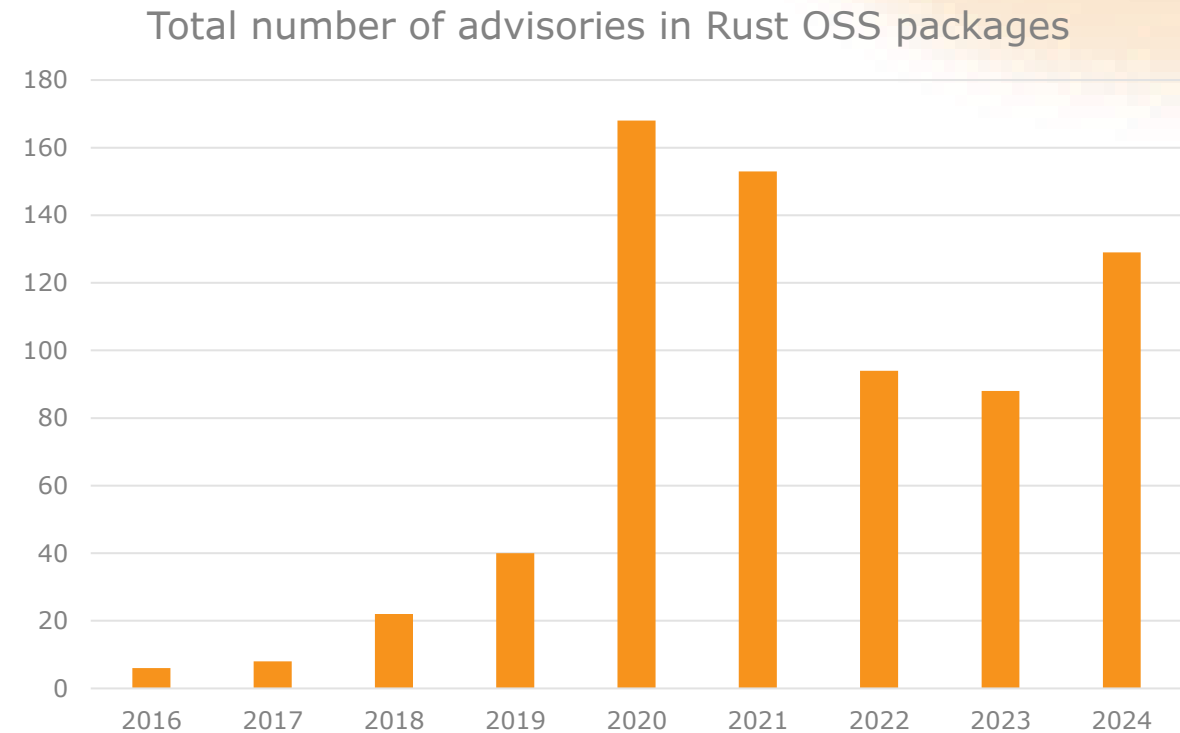
C/C++

- ✓ No runtime errors / no memory safety issues: avoid Undefined Behaviors



Rust vulnerabilities do happen

- rustsec.org inventories Rust code security advisories on open source* packages
- **Sharp increase of detection of vulnerabilities** since 2020
 - Rust's growing adoption
 - Improved code analysis tools
- **A lot is probably still not detected** due to limitations of current Rust static code analyzers
- Although less measurable than for security, it can be expected that **similar bugs will cause safety problems**



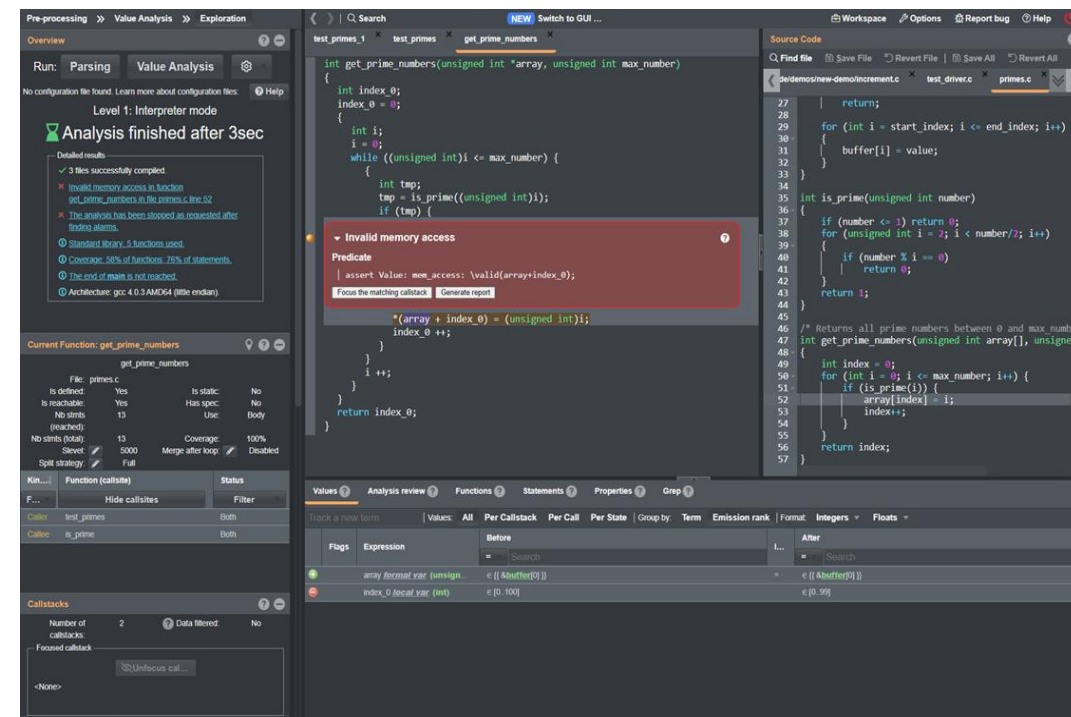
* These statistics only cover crates.io open source packages, not private code vulnerabilities

** rudra, clippy

- Founded in 2013
- Headquarters: **Paris**, France  
- Specializes in advanced application security analysis for **C**, **C++** and **Rust** software.

Products and services

- TrustInSoft Analyzer: hybrid code analysis tool designed to prove the absence of runtime errors.
- Dedicated expert-driven verification service



minimal-rust-demo	100%	96%	96%	✗ 1 alarm has been found. ▼
drivers.c	100%	89%	89%	
src	100%	100%	100%	
decrypt.c	100%	100%	100%	
lib.rs	100%	100%	100%	
verify_something			100%	


```

20 pub fn verify_something(x: u32) -> u32 {
21     let keys = Keys {
22         public: [3057, 6115, 1753],
23         private: [812, 2439, 303],
24     };
25     let mut key_index = 0;
26     let mut acc = 0u32;
27
28     while key_index < 3 {
29         // Decrypt the current key element.
30         acc = acc + keys.decrypt_element(x, key_index);
31
32         // Treat next key element.
33         key_index += 1;
34     }
35
36     // Do the final verification.
37     // According to the chinese remainder theorem, there is one unique number
38     // between 0 and 3057 * 6115 * 1753 = 32_769_801_915 such that acc will be
39     // 0. Since 32_769_801_915 is bigger than u32::MAX (4_294_967_295), there
40     // a chance that this function is completely safe.
41     1 / acc
42 }

```

Div_by_zero in function **verify_something** in file **src/lib.rs** line **41**

Predicate

acc == 0

Values and expressions

From analysis generalized

acc ∈ [0..10922]

Callstack

Production project 1: Phone ROM bootloader

- 26.5k Statements, >80% coverage reached
- 12 alarms/bugs reported:
 - All of them come from un-sanitized values read from the memory
 - Varying impact: *Invalid memory access*, *Out of bound slice access* and *Unsigned overflow*
 - 7 cause a direct call to panic
- 50% (only?) are in unsafe
- 1 unsigned overflow: panic in Debug mode, wrap-around in Release mode and the unexpected value triggers a panic during an out-of-bound slice access

Production project 2: upgradable bootloader

- 37.5k Statements, ~70% statement coverage
- 13 alarms reported of same pattern
 - Panic in Debug mode
 - Silently return unexpected value or panic in Release mode
- They all happen in safe Rust, but 12 of 13 values are coming from unsafe code
 - Mostly, unsanitized results
- For 2/13 the error is a logical error in the code

Project 3: Linux-class OS

- Scope: crates containing unsafe
 - Low layer of device drivers
 - Internal unsafe framework for memory-management, interrupt handling, timer...
- 10 alarms reported:
 - Invalid allocation size
 - Unsigned overflows
 - Unexpected value
- All alarms
 - In safe code
 - Originate from unsanitized input
- Consequences:
 - Failing to boot
 - Kernel panic

How to verify Rust in production code

- *Stable toolchain*: upstream, Ferrocene/AdaCore/Hightec
- *Unit test*: cargo test (or cmake test, bazel test...)
 - May run using TrustInSoft Analyzer
- *Traceability*: comment-based, nothing new here
- *Reliability*: clippy, miri, TrustInSoft Analyzer
- *Coverage*: LLVM, TrustInSoft Analyzer
- *Cybersecurity*: TrustInSoft Analyzer

THANK YOU



@TrustInSoft



trust-in-soft.com



@TrustInSoft



contact@trust-in-soft.com

Rust Paris 2026

Mot de la fin



Sponsors Gold

CÉNOTÉLIE

TITAGONE
by OCaml PRO

THALES
Building a future we can all trust

Sponsors Silver

AdaCore

oxid[os]

Wyliodrin

Rust Paris 2026

Arthur Carcano

Ingénieur R&D Rust |

Docteur en mathématiques et informatique

Ocamlpro



Sponsors Gold

CÉNOTÉLIE

TITAGONE
by OCaml PRO

THALES
Building a future we can all trust

Sponsors Silver

AdaCore

oxid[os]

Wyliodrin

ça bouge du côté Rust...

- **Rust Lille** | <https://www.meetup.com/fr-FR/meetup-group-zgphbyet/>
=> Événements réguliers autour de Rust, gratuits et ouverts à toutes et à tous, débutants, curieux ou experts.
- **Rust Lyon** | <https://www.meetup.com/fr-fr/rust-lyon/>
=> Une rencontre par mois, gratuite et ouverte à toutes et tous, afin d'échanger, partager des bonnes pratiques, des retours d'expériences, apprendre ensemble, voire contribuer ensemble à l'écosystème Rust.
- **Grenoble - tupperlibre** | tupperrust.github.io/
=> Les TupperRust sont des évènements autour du langage Rust.
- **Rust meetup** | ~ 1 fois par mois | Paris | https://mobilizon.fr/@rust_paris
=> Objectif de ce groupe : partage de connaissance, apprendre le Rust, se rencontrer, contribuer à des projets en Rust...
- **EuroRust 2026** | Barcelone + Online | 14 - 17 octobre 2026
- **RustConf 2026** | Montreal + Online | 8 -11 septembre 2026
=> <https://rustconf.com/>



Open Source Experience 2026 | 09 & 10 décembre 2026 | Paris

Appel à conférences ouvert du 29 avril au 14 juin

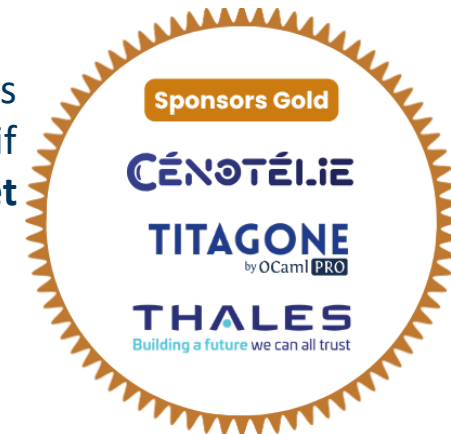
Pour cette 6^e édition, le fil rouge « L'open source en action » mettra en avant des **retours d'expérience concrets**, des solutions déployées et **des cas d'usage** capables de répondre à des problématiques réelles. Plus que jamais, l'objectif est de **montrer comment les technologies ouvertes permettent de construire un numérique fiable, durable et souverain**.

6 thématiques retenues cette année :

- **Track 1** : Souveraineté numérique, gouvernance et modèles économiques open source
- **Track 2** : Infrastructures ouvertes, cloud et plateformes souveraines
- **Track 3** : IA ouverte, modèles, data et écosystèmes souverains
- **Track 4** : Cybersécurité, confiance et résilience open source
- **Track 5** : Plateforme applicative d'entreprise souveraine
- **Track 6** : Culture, compétences et métiers de l'open source

👉 Appel à conférences :

<https://www.opensource-experience.com/programme/appel-a-conferences/>



Cocktail de clôture

Merci à vous !



Sponsors Gold

CÉNOTÉLIE

TITAGONE
by OCaml PRO

THALES
Building a future we can all trust

Sponsors Silver

AdaCore

oxid[os]

Wyliodrin

Systematic

Paris Region Deep Tech Ecosystem

L'action du pôle est soutenue par :



Partenaires officiels :



Innovation Collaborative, Croissance Collective

RÉPUBLIQUE FRANÇAISE